P.B. SIDDHARTHA COLLEGE OF ARTS & SCIENCE

Siddhartha Nagar, Vijayawada – 520 010

Autonomous - Re-accredited at 'A+' by the NAAC - ISO 9001 – 2015 Certified

College with ---Potential for Excellence-Phase-II (Awarded by the UGC)

2.3.2 - Teachers use ICT-enabled tools including online resources for effective teaching and learning

LECTURE NOTES

PARVATHANENI BRAHMAYYA(P.B.)
SIDDHARTHA COLLEGE OF ARTS & SCIENCE
VIJAYAWADA, ANDHRA PRADESH

LECTURE NOTES PREPARED BY Dr.T.S.RAVI KIRAN, ASSISTANT

PROFESSOR & HOD DEPARTMENT OF COMPUTER SCIENECE

P.B.SIDDHARHA COLLEE OF ARTS & SCIENECE, VIJAYAWADA, AP, INDIA PIN: 520010

Email: tsravikiran@pbsiddhartha.ac.in Mobile: 9441176980

COURESE: ADVANCED DATABASE MANAGEMENT SYSTEMS

ACADEMIC YEAR: 2021-2012 **COURESE CODE: 21DS1T4**

CONTENTS:

1. BASIC & MORE SQL

2. RELATIONAL ALGEBRA & RELATIONAL CALCULUS

3. ER- MODELLING

4. NORMALIZATION

5. TRANSACTION PROCESSING

6. CONCURRENCY CONTROL

7. DISTRIBUTED DATABASE CONCEPTS

8. NO SQL

9. AGGREGATE DATA MODELS

10. MORE DETAILS ON DATA MODELS

11. DISTRIBUTION MODELS

12. CONSISTENCY

13. MongoDB CRUD OPERATIONS

14. DATA LAKES

CHAPTER 1
BASIC & MORE SQL

- Data: Collection of *facts* (or) *raw material* for information.
- Information: Processed data.
- Schema: Structure of database (or) Structure of database objects.
- Entity: A *thing* (or) *object which* have some implicit meaning.
- Record: The values associated for collection of fields.
- Domain: The range of finite values associated for attributes.
- Database: Collection of files (or) collection of tables (or) collection of database objects.
- Attribute: Property of entity.
- Database management system: Set of software modules used to create database objects and access the database objects.

TOPIC1: SQL AND SQL DATA TYPES

1.1 SQL

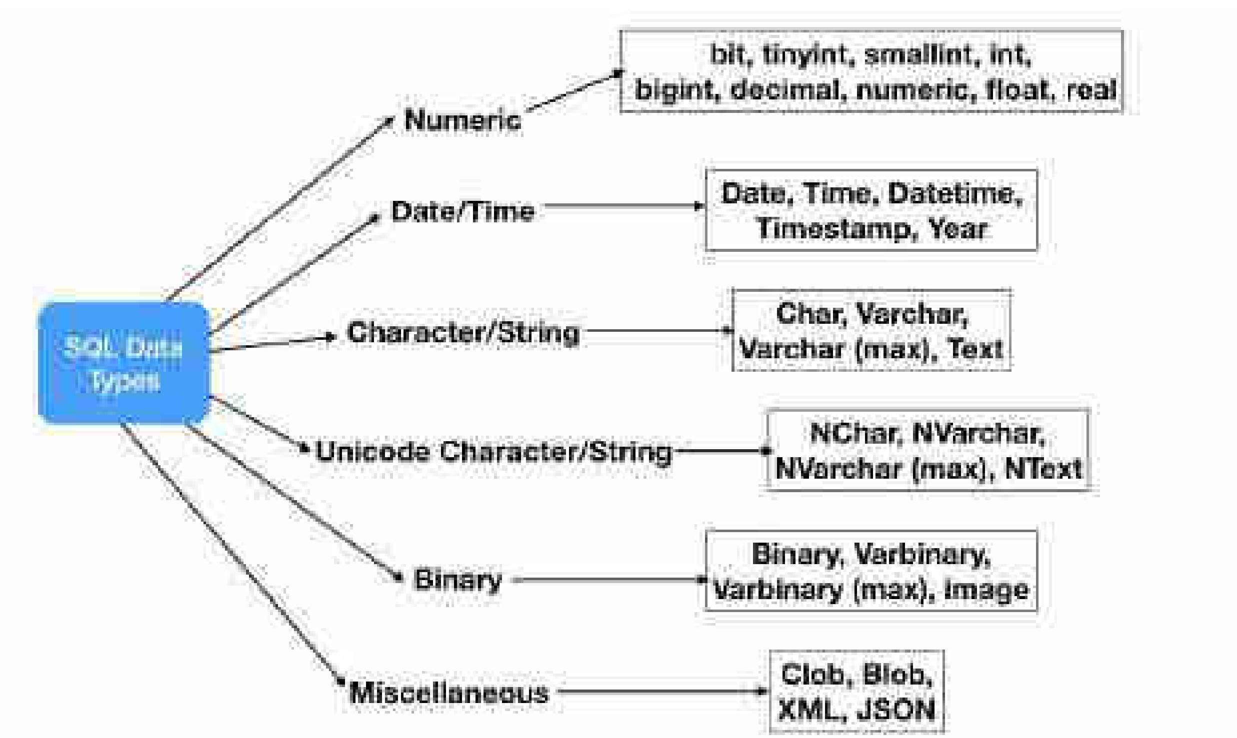
- The name SQL is presently expanded as Structured Query Language.
- Originally, SQL was called SEQUEL (Structured English QUEry Language) and was designed and implemented at IBM Research as the interface for an experimental relational database system called SYSTEM R.
- SQL is now the standard language for commercial relational DBMSs.
- The standardization of SQL is a joint effort by the American National Standards Institute (ANSI) and the International Standards Organization (ISO).

SQL Versions:

1. The first SQL version is SQL-86 or SQL1.
2. The second SQL version is SQL-92 or SQL2
3. The next standard that is well-recognized is SQL:1999 (SQL3)
4. Additional updates to the standard are SQL:2003 and SQL:2006, which added XML features.
5. Another update in 2008 incorporated more object database features into SQL, and a further update is SQL:2011.

1.2 SQL Data Types IMP

- SQL data types can be broadly divided into following categories.



1.2.1 SQL Numeric Data Types

Datatype	From	To
bit	0	1
tinyint	0	255
smallint	-32,768	32,767
int	-2,147,483,648	2,147,483,647
bigint	-9,223,372,036, 854,775,808	9,223,372,036, 854,775,807
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
float	-1.79E + 308	1.79E + 308

real	-3.40E + 38	3.40E + 38

1.2.2 SQL Date and Time Data Types

Datatype	Description
DATE	Stores date in the format YYYY-MM-DD
TIME	Stores time in the format HH:MI:SS
DATETIME	Stores date and time information in the format YYYY-MM-DD HH:MI:SS
TIMESTAMP	Stores number of seconds passed since the Unix epoch (‘1970-01-01 00:00:00’ UTC)
YEAR	Stores year in 2 digits or 4 digit format. Range 1901 to 2155 in 4-digit format. Range 70 to 69, representing 1970 to 2069.

1.2.3 SQL Character and String Data Types

Datatype	Description
CHAR	Fixed length with a maximum length of 8,000 characters
VARCHAR	Variable-length storage with a maximum length of 8,000 characters
VARCHAR(max)	Variable-length storage with provided max characters, not supported in MySQL
TEXT	Variable-length storage with maximum size of 2GB data

Note that all the above data types are for character stream, they should not be used with Unicode data.

1.2.4 SQL Unicode Character and String Data Types

Datatype	Description
NCHAR	Fixed length with maximum length of 4,000 characters
NVARCHAR	Variable-length storage with a maximum length of 4,000 characters
NVARCHAR(max)	Variable-length storage with provided max characters
NTEXT	Variable-length storage with a maximum size of 1GB data

Note that above data types are not supported in MySQL database.

1.2.5 Boolean:

- A Boolean data type has the traditional values of TRUE or FALSE. In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a Boolean data type is UNKNOWN.

TOPIC 2: DBMS LANGUAGES AND INTERFACES IMP

2.1 DDL (Data Definition Language)

- Language used to create database objects.
- Used to *create* & *alter* the database objects.

E.g. create, alter commands.

```
E.g.  create table emp
      (empno number(10),
       ename  varchar2(15));

E.g.  alter table emp
      modify ( empno number(15));
```

2.2 SDL (Data Definition Language)

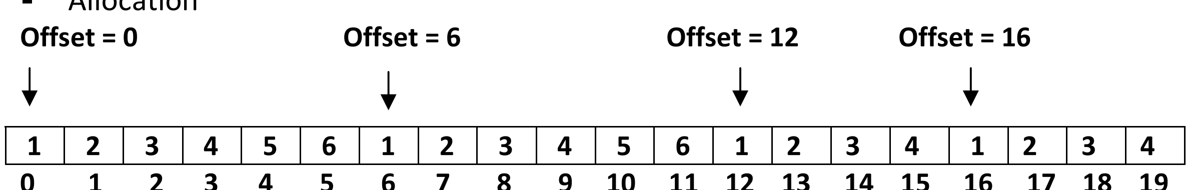
- SDL (Storage Definition Language): Used to specify the *internal schema* of database objects.

- When employee table with 3 attributes EMPNO, DEPTNO & PAY is stored in database persistently the following number of bytes allocated according to the shown format.

```
> create table emp
```

```
(empno varchar2(5),
 deptno  varchar2(10),
 pay number(5));

STORED EMP LENGTH = 20
       PREFIX  TYPE = BYTE(6), OFFSET = 0
       EMPNO    TYPE = BYTE(6), OFFSET = 6, INDEX = EMPX
       DEPTNO   TYPE = BYTE(4), OFFSET = 12
       PAY         TYPE = FULLWORD, OFFSET = 16
```

- Allocation



- From the given example

PREFIX	System generated attributed where the values of attributes is fixed.
OFFSET	Indicates starting address of next attribute.

2.3 DML (Data Manipulation Language)

- DML (Data Manipulation Language): Used to *access* database objects.
  E.g. insert, delete, update commands

E.g. insert into emp
    values (&empno,'&ename');

E.g. update emp
    set ename='john'
    where empno=1;

E.g. delete from emp;

2.4 DCL (Data Control Language)

- Used to provide persistency and recovery to database objects.

E.g.

Commit;	Commits the transaction.
Rollback;	Database items acquires previous state.
Savepoint <savepoint_name>;	Creates savepoint in a transaction.
Rollback to <savepoint_name>;	Databse item acquires savepoint status.

2.5 TCL (Transaction Control Language)

(a) *Grant* option assign privileges to the users.
(b) *Revoke* option cancel access privileges from the users.

E.g. GRANT INSERT, DELETE ON EMPLOYEE, DEPARTMENT TO A2;

E.g. GRANT SELECT ON A3EMPLOYEE TO A3 WITH GRANT OPTION;

E.g. REVOKE SELECT ON EMPLOYEE FROM A2;

TOPIC 3: RELATIONAL CONSTRAINTS VVIMP

- Constraint: A rule at *column level* or *table level*.
- Relational constraints

1. Not null (Defined at either column level or table level)
2. Unique (Defined at either column level or table level)
3. Primary key (Defined at either column level or table level)
4. Foreign key (Defined at either column level or table level)
6. Check (Defined at either column level or table level)

1. Not null: Specifies that may not contain null value.

E.g.

```
SQL> create table emp
        (empno number(4),
         ename varchar2(10) not null);
```

2. Unique: Every value in a column or set of columns be unique.

E.g.

```
SQL> create table dept
        (deptno number(2),
         dname varchar2(14),
         constraint dept_dname_uk unique(dname));
```

3. Primary key: Creates primary key for the table.

- The value assigned for an attribute must satisfy the constraints *unique* and *not null*.

E.g.

```
SQL> create table dept
        (deptno number(2),
         dname varchar2(14),
         constraint dept_dname_pk primary key(dname));
```

4. Foreign key: The foreign key is defined in the child table, and the table containing the reference column is in the parent table.

E.g.

```
SQL> create table emp
        (empno number(2),
         ename varchar2(14) not null,
         deptno number(2),
         constraint emp_dept_fk foreign key(deptno) references dept (deptno));
```

5. Check: Defines the condition that the each row must satisfy.

E.g.

```
SQL> create table dept
        (deptno number(2),
         dname varchar2(14),
         constraint emp_dept_ck check (deptno between 10 and 99));
```

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

TOPIC 4: VIEW VVIMP

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

Syntax:
CREATE VIEW *view_name* AS
SELECT *column1*, *column2*, ...
FROM *table_name*
WHERE *condition*;

Example:
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';

TOPIC 5: CURSORS VIMP

- A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor.
- A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

1. There are two types of cursors
2. Implicit cursors
3. Explicit cursors

5.1 Implicit Cursors

- Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement.
- Programmers cannot control the implicit cursors and the information in it.
- Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

Example for Implicit Cursor:

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE customers
  SET salary = salary + 500;
  IF sql%notfound THEN
    dbms_output.put_line('no customers selected');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows ||' customers selected ');
  END IF;
END;
```

5.2 Explicit Cursor

- Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

Example for Explicit Cursor:

```
DECLARE
  c_id customers.id%type;
  c_name customers.name%type;
  c_addr customers.address%type;
  CURSOR c_customers is
    SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
  FETCH c_customers into c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
```

TOPIC 6: TRIGGERS VIMP

- A trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table.
- A trigger is triggered automatically when an associated DML statement is executed.

Example 1:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_diff number;
BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

TOPIC: JOIN OPERATION IN RELATIONAL ALGEBRA VIMP

Join Operation: It is used to combine tuples from multiple relations so that related information can be presented.

- **It is denoted by join symbol ⋈.**

Syntax: $R \bowtie_{<Join\ Condition>} S$

Types of joins:

(a)Theta Join: **A general join condition.**

- **Syntax: Ai θ Bi**
- **θ is one of the comparison operators of {=,>,>=,<,<=, ≠}**

(b)Equi Join: **Join based on the common columns in different relations.**

Syntax: $R \bowtie_{<Column\ =\ Column>} S$
E.g. $DEPARTMENT \bowtie_{<MGRSSN\ =\ SSN>} EMPLOYEE$

DEPT		
ENO	DEPTNO	DNAME
1	10	RESEARCH
2	20	OPERATIONS

EMP	
ENO	ENAME
1	KING
2	BLACK

DEPT			
ENO	ENAME	DEPTNO	DNAME
1	KING	10	RESEARCH
2	BLACK	20	OPERATIONS

(c)Non Equi Join: **Join of two relations.**

- **Non equi join always return "*Cartesian Product*" or "*Cross Product*"**
- **It is also known as natural join.**
- **It is denoted by ***

Syntax: $R *_{(<list1>\ <list2>)} S$
E.g. DEPARTMENT ⋈ EMPLOYEE

EMP	
ENO	ENAME
1	KING
2	BLACK

DEPT	
DEPTNO	DNAME
10	RESEARCH
20	OPERATIONS

EMP_CROSS_DEPT			
ENO	ENAME	DEPTNO	DNAME
1	KING	10	RESEARCH
1	KING	10	OPERATIONS
2	BLACK	20	RESEARCH
2	BLACK	20	OPERATIONS

(d)Outer Join: Join condition that returns tuples with direct map.

(i) Left Outer Join: Returns the unmatched tuples in the left side relation.

- It is denoted by the symbol ⟕.

Syntax: R ⟕$_{\text{<Condition>}}$ S

EMP	
ENAME	DEPTNO
KING	10
BLAKE	30
CLARK	10
JONES	20
JAMES	40

DEPT	
DEPTNO	DNAME
10	ACCOUNTING
20	SALES
30	RESEARCH

Note: The tuple with DEPTNO 40 in EMP relation does not participate in the relation.

(ii) Right Outer Join: Join condition that returns tuples in the right side relation.

- It is denoted by the symbol ⟖

Syntax: R ⟖ S

EMP	
ENAME	DEPTNO
KING	10
BLAKE	30
CLARK	10
JONES	20

DEPT	
DEPTNO	DNAME
10	ACCOUNTING
20	SALES
30	RESEARCH
40	OPERATIONS

Note: The tuple with DEPTNO 40 in relation DEPT does not participates in relation.

(iii) Full Outer Join: Join condition that returns tuples in both the relations that participates relation.

- It is denoted by the symbol ⟗ .

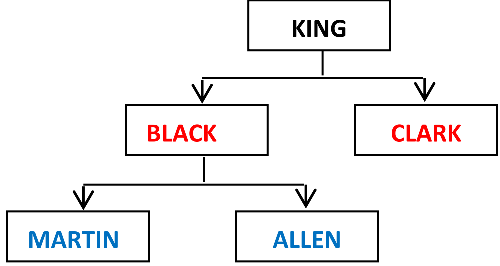
Syntax: R ⟗ S

EMP	
ENAME	DEPTNO
KING	10
BLAKE	30
CLARK	10
JONES	20
JAMES	40

DEPT	
DEPTNO	DNAME
10	ACCOUNTING
20	SALES
30	RESEARCH
40	OPERATIONS

(e)Self Join: Joining a relation itself.

EMP		
EMPNO	ENAME	MGR
7839	KING	
7698	BLACK	7839
7782	CLARK	7839
7654	MARTIN	7698
7499	ALLEN	7698



RELATIONAL CALCULUS IMP

- Relational calculus is declarative *Non-procedural Language* based on mathematics.
- Relational calculus are two types (a) Tuple Relational Calculus (b) Domain Relational Calculus.

1.1 TUPLE RELATIONAL CALCULUS

- Tuple Relational Calculus: Tuple Relational Calculus is a *Non-procedural Query Language* unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it.
- Tuple Relational Calculus range over tuple.

Syntax: { t | cond(t) }
T: Tuple
Cond(t): Formula

Query 0: Find all employees whose salary is above $50000

Relational Calculus Expression:

Q0 : { t | employee(t) and t.salary > $50000}

Note: t is the tuple variable that ranges that ranges over the relation employee.

Query 1: Retrieve the Birthdate and Address of the employee whose name is "John B Smith"

Q1 : {t.Birthdate, t.Address | Employee(t) and t.Fname = 'John' t.Minit='B' and t.Lname='Smith'}

1.2 DOMAIN RELATIONAL CALCULUS

- Domain Relation Calculus ranges over finite values assigned for an attributes.
- Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it.

Syntax: $\{x_1, x_2, \ldots, x_n \mid COND(x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{n+m})\}$

Note: COND is Condition (or) Formula

A domain relational calculus is of the form

$\{x_1, x_2, \ldots, x_n \mid R(x_1, x_2, x_3 \ldots, x_n)\}$

Note: R is a relation

X_i is Domain Variable (Attribute)

An ATOM is of the form **Xi OP X_j (or) X_i OP C**

Note: OP is a operation

ATOM: Relation between two Attributes

C is constant.

$X_{i,}$ X_j is attributes.

OP: (=,<>,<=,>,>=)

Eg: X_i OP X_j

e.sal > d.sal

Query 0: Retrieve the Birthday and Address of Employee whose Name is ‘John B Smith’.
Note: Use quantifiers to express the formula.

Q0 : { u v | (∃q) (∃r) (∃s) (∃t) (∃w) (∃x) (∃y) (∃z) (Employee (q r s t u v w x y z) AND q=’John’ AND r=’B’ AND S=’Smith’)}

CHAPTER 2
RELATIONAL ALGEBRA & RELATIONAL CALCULUS

TOPIC 1: SELECT OPERATION (σ) VIMP

- Used to select subset of tuples (rows) from relation (table) that satisfies a select condition.
  Syantax: $\sigma_{<\text{Select Condition}>}(R)$

Syntax for Select Condition:
1.<attribute_name> *<comparision_operation>*<constrant_value>
E.g. sal > 5000;

2.<attribute_name> *<comparision_operation>*<attribute_name>
E.g. e.esal > d.esal
Note: e and d are table aliases

3.<selection_condition>*<boolean_operator>*<selection_condition>
E.g. ename= 'James' and salary>5000

- Comparison operators: There are normally one of the operations from the set {=, <, >, ≤, ≥, ≠}

- Boolean operators: Boolean operation is normally one of the operations of the set {AND, OR, NOT}

- Selection operations is commutative:
  $\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$

- We can always combine a cascade (or sequence) of SELECT operations into a single SELECT operation with a conjunctive (AND) condition; that is,
  $\sigma_{<cond1>}(\sigma_{<cond2>}(...(\sigma_{<condn>}(R)) ...)) = \sigma_{<cond1>\ AND<cond2>\ AND...AND\ <condn>}(R)$

Qeury 0: Select list of employees who work for department number 4

Sql query:
```
sql>  select *
        from employee
        where dno=4 and salary>25000;
```

Select expression:
$\sigma_{dno=4\ and\ salary>25000}$ (employee)

TOPIC 2: PROJECT OPERATION (∏) VIMP

- Selects certain columns from a relation.
  Syntax: $\prod_{<attribute_list>}(R)$

- Degree of relation: Number of attributes in a relation.

- Duplicate elimination: Duplicate rows are eliminated when projection operation done.

- Super key: Key may have *single attribute* or *group of attributes* used to distinguish one record with another record.

  $\prod_{<list1>}(\prod_{<list2>}(R)) = \prod_{<list1>}(R)$ when list1 and list2 is same.

Qeury 0: List each employee first name, last name and salary from the relation employee.

Sql query:
select fname, lname, salary .
from employee;

Project expression:
$\prod_{<fname,lname,salary>}(employee)$

TOPIC 3: RENAME OPERATION (ρ)

- To rename relation and columns of the relation.

Syntax to rename relations and attributes: $\rho_{s\ (B1,B2,B3,\dots,Bn)}(R)$

Syntax to rename relation: $\rho\,(R)$

Qeury 0: Retrive the first name, last name and salary of employees who work for department 5.

Sql query:
select fname, lname, salary
from employee
where dno=5.

Rename expression:
$DEP5_EMPS \leftarrow \sigma_{dno=5}(EMPLOYEE)$
$RESULT \leftarrow \prod_{fname,lname,salary}(DEP5_EMPS)$

TOPIC 4: SET OPERATIONS VIMP

UNION: Include all the tuples in R and S by eliminating duplicate rows.

- The result of operation is represented by R ∪ S.

R	SSN
	123456789
	333445555
	666884444
	453453453

S	SSN
	333445555
	888665555

R∪S	R∪S
	123456789
	333445555
	666884444
	453453453
	888665555

INTERSECTION: Includes all the tuples that are common in both R and S.

- The result of operation is represented by R ∩ S.

R∩S	R∩S
	888665555

DIFFERENCE: Includes all the tuples that are in 'R' but not in 'S'.

- The result of operation is represented by R – S.

R – S	R – S
	123456789
	666884444
	453453453

TOPIC 5: DIVISION OPERATION

- Used to execute special kinds of queries.

Syntax: R ÷ S

- The division operation can be expressed as follows.

T1 ← ∏y (R)

T2 ← ∏y ((S×T1) – R)

T ← T1 − T2

R	A	B
	a1	b1
	a2	b1
	a3	b1
	a4	b1
	a1	b2
	a3	b2
	a2	b3
	a3	b3
	a4	b3
	a1	b4
	a2	b4
	a3	b4

S	A
	a1
	a2
	a3

Step 1: Let us assume that

X = {A}

Y = {B}

Z = {A,B}

Step 2:

T1 ← ∏y (R)

T1	B
	b1
	b2
	b3
	b4

Step 3:

T2 ← ∏y ((S×T1) – R)

S × T1	A	B
	a1	b1 ×
	a1	b2 ×
	a1	b3
	a1	b4 ×
	a2	b1 ×
	a2	b2
	a2	b3 ×
	a2	b4 ×
	a3	b1 ×
	a3	b2 ×
	a3	b3 ×
	a3	b4 ×

R	A	B
	a1	b1 ×
	a2	b1 ×
	a3	b1 ×
	a4	b1
	a1	b2 ×
	a3	b2 ×
	a2	b3 ×
	a3	b3 ×
	a4	b3
	a1	b4 ×
	a2	b4 ×
	a3	b4 ×

S × T1	A	B
	a1	b3
	a2	b2

$\prod_Y$ (S × T1 – R)	B
	b3
	b2

T2	B
	b3
	b2

Step 4:

T ← T1 – T2

T1	B
	b1
	b2 ×
	b3 ×
	b4

T2	B
	b3
	b2

T	B
	b1
	b4

CHAPTER 3
ER- MODELLING

3.1 Entity types, Entity sets, Attributes & Keys — IMP

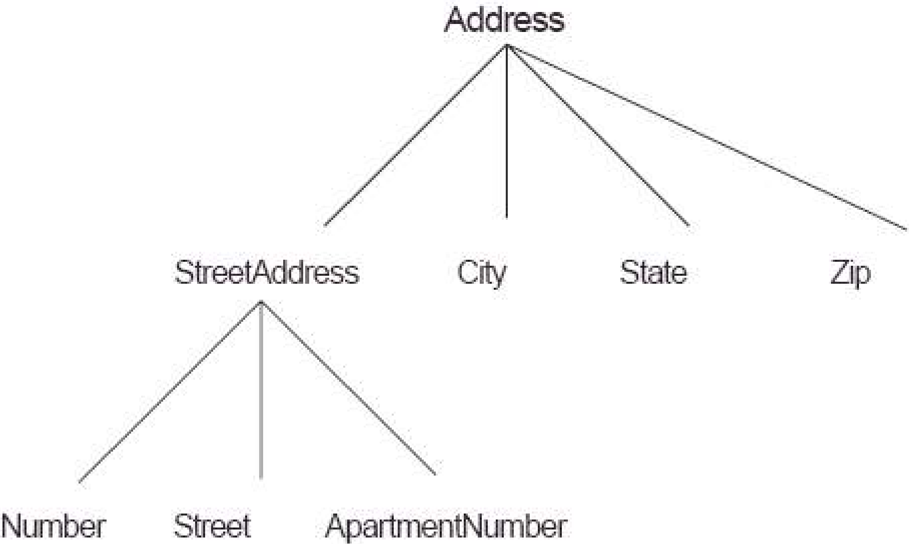
- **Entity: An object in the real world with an independent existence.**

- **Attribute: Property that describes an aspect of the entity.**

- **Attribute types:**

(a) Simple: Attribute that is not divisible
E.g. Age
(b) Composite: Attribute that can be divided into smaller parts.
E.g. Address

Figure 3.4 A hierarchy of composite attributes; the StreetAddress component of an Address is further composed of Number, Street, and ApartmentNumber.



(c) Single-valued: Attribute containing single value.
E.g. Age, City

(d) Multi-valued: Attribute containing set of values.
E.g. Locations for DEPARTMENT
E.g. College degree

(e) Stored: Attribute having fixed value.
E.g. Date Of Birth

(f) Derived: Attribute derived from the value of the existing attribute.
E.g. NumberOfEmployees for DEPARTMENT
E.g. Age

(g) Complex attribute: Composite and multivalued attributes can be nested in an arbitrary way.

Figure 3.7 The CAR entity type, with two key attributes Registration and VehicleID. Multivalued attributes are shown between set braces {}. Components of a composite attribute are shown between parentheses ().

CAR
Registration(RegistrationNumber, State), VehicleID, Make, Model, Year, {Color}

car_1 •

((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 1998, {red, black})

car_2 •

((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 1999, {blue})

car_3 •

((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 1995, {white, blue})

⋮

- **Entity sets:** **The collection of all entities of a particular entity type in the database at any point of time.**

Figure 3.6 Two entity types named EMPLOYEE and COMPANY, and some of the member entities in the collection of entities (or entity set) of each type.

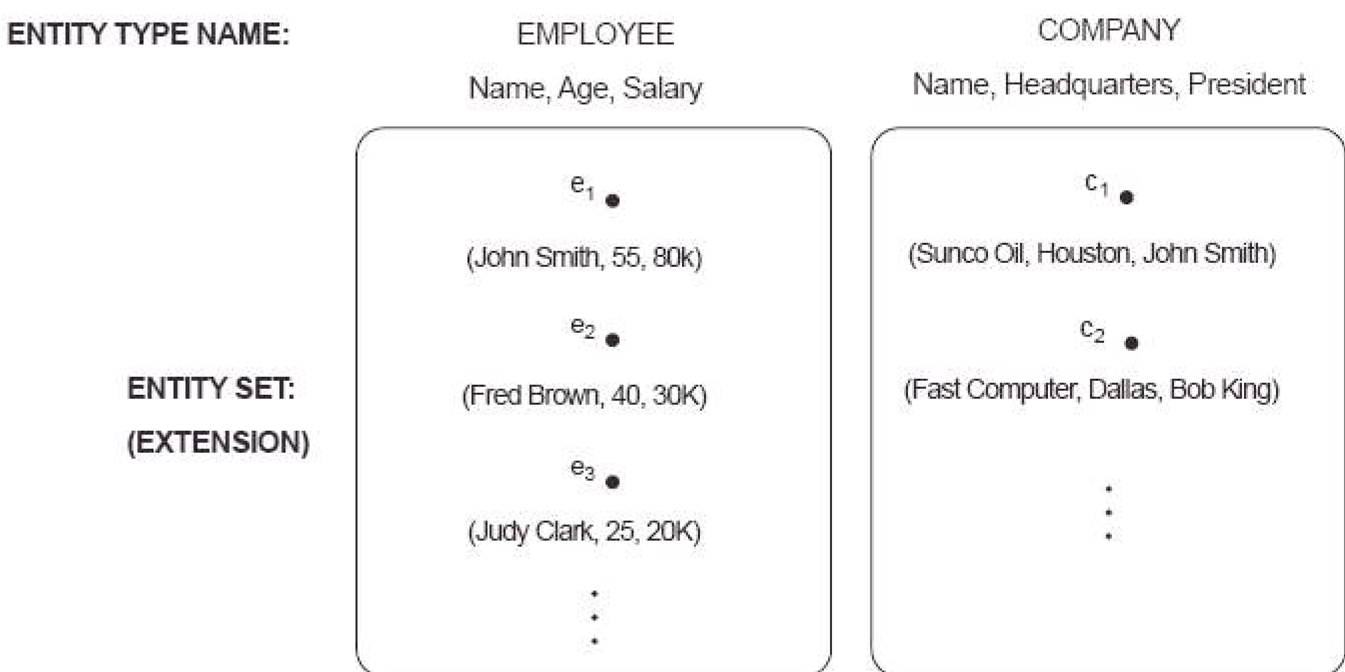
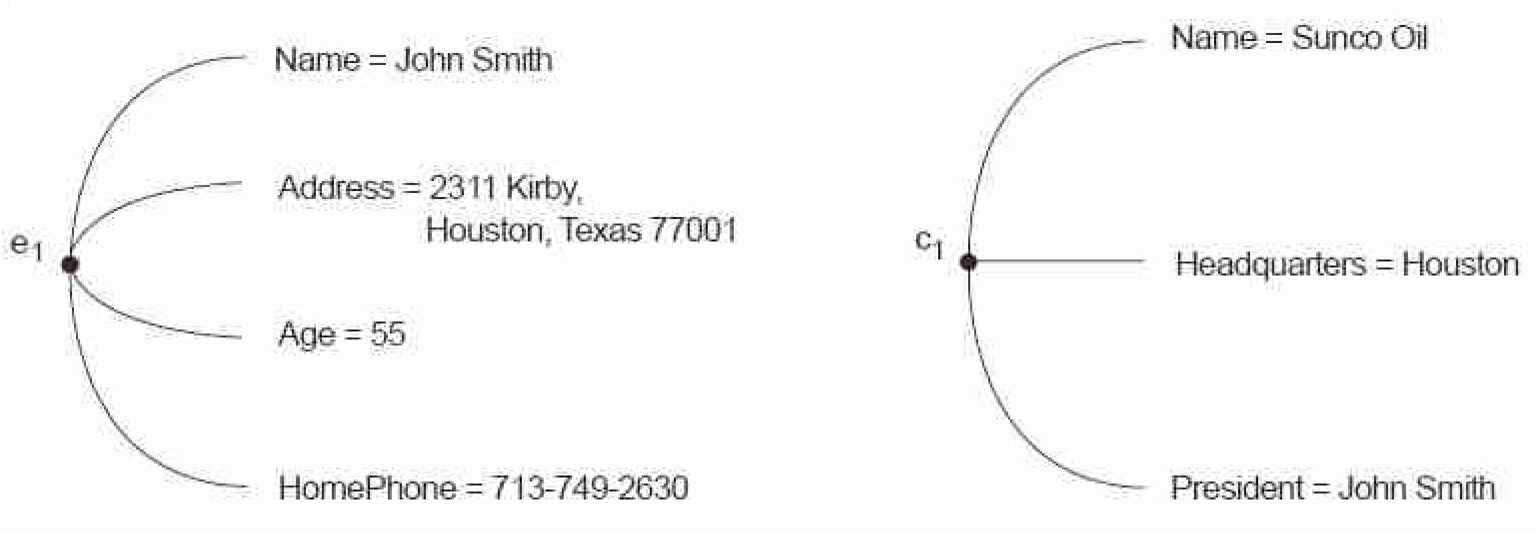


Figure 3.3 Two entities, an employee e_1 and a company c_1, and their attribute values.



SECOND

S#	STATUS	CITY
S1	20	LONDON
S2	10	PARIS
S3	10	PARIS
S4	20	LONDON
S5	30	ATHENS

↑ Domain

A: E→ P(V)

A : Attribute
E : Entity set
V : Value set
V = P(V1) x ... x P(Vn) for composite attributes
A(e) denotes the value of attribute A for entity e

- **Relationship:** The reference between two tables.

- **Relationship type:** Set of associations between n entities .

Note: Relationship type R among n entity types E1, ..., En defines a set of associations among entities from these types. Each association will be denoted as (e1, ..., en) where ei belongs to Ei, 1 <= i <= n.

- **Relationship degree:** The number of entity types participates in a relationship type.

- **Binary Relationship degree:** When two entity types participates in a relationship type.

- **Ternary Relationship degree:** When three entity types participates in a relationship type.

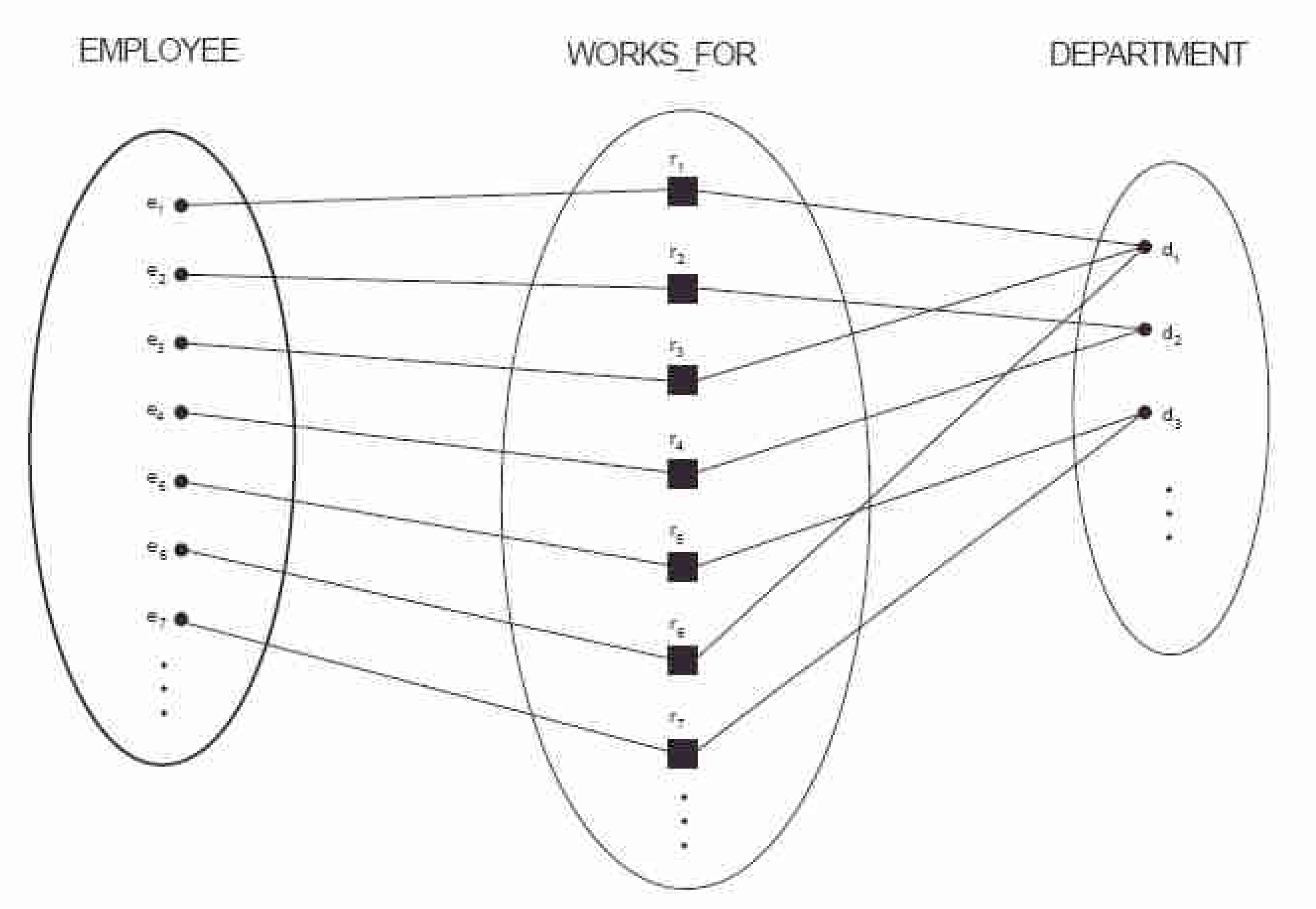
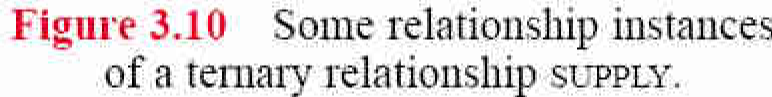
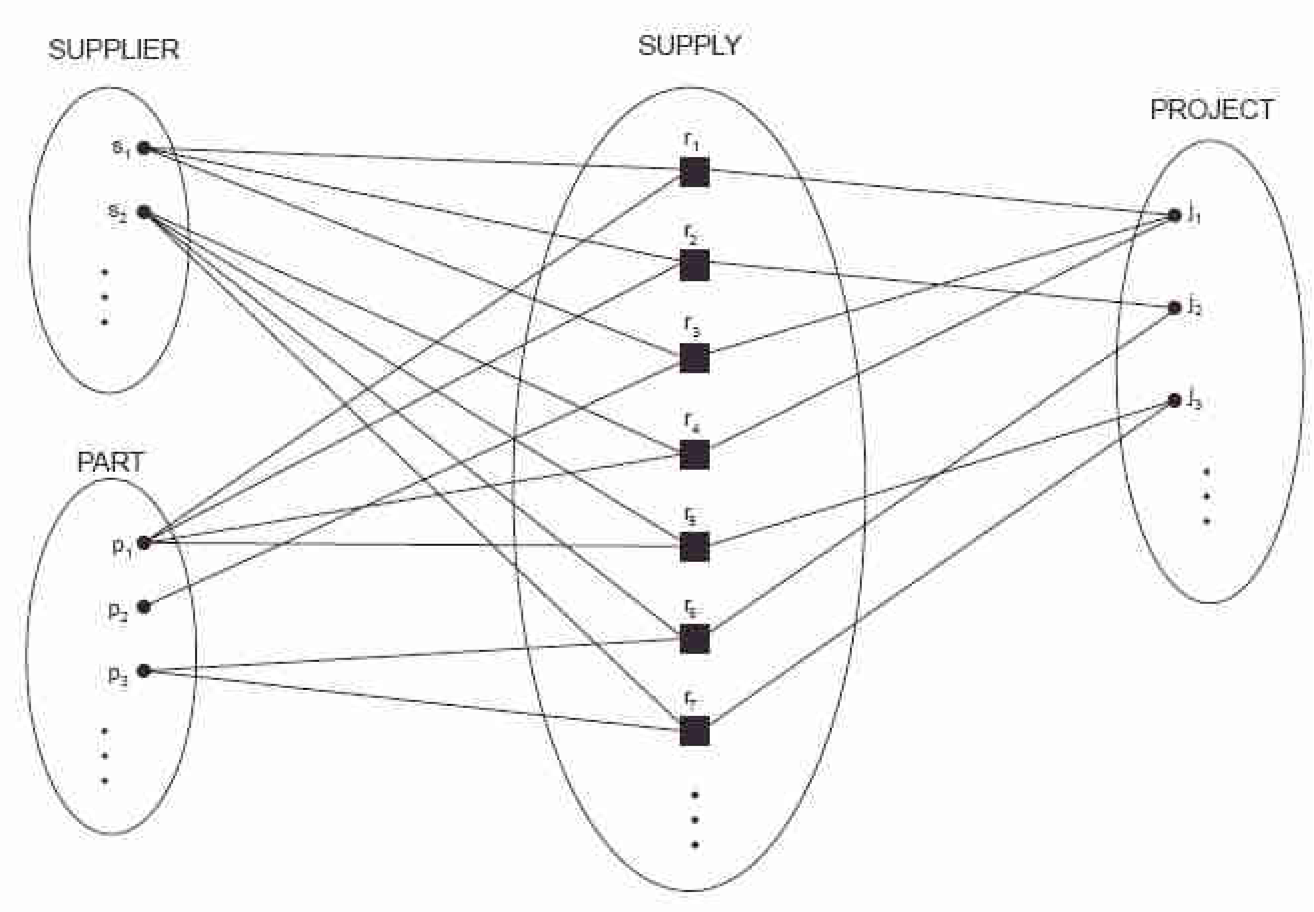


Figure 3.9 Some instances of the WORKS_FOR relationship between EMPLOYEE and DEPARTMENT.

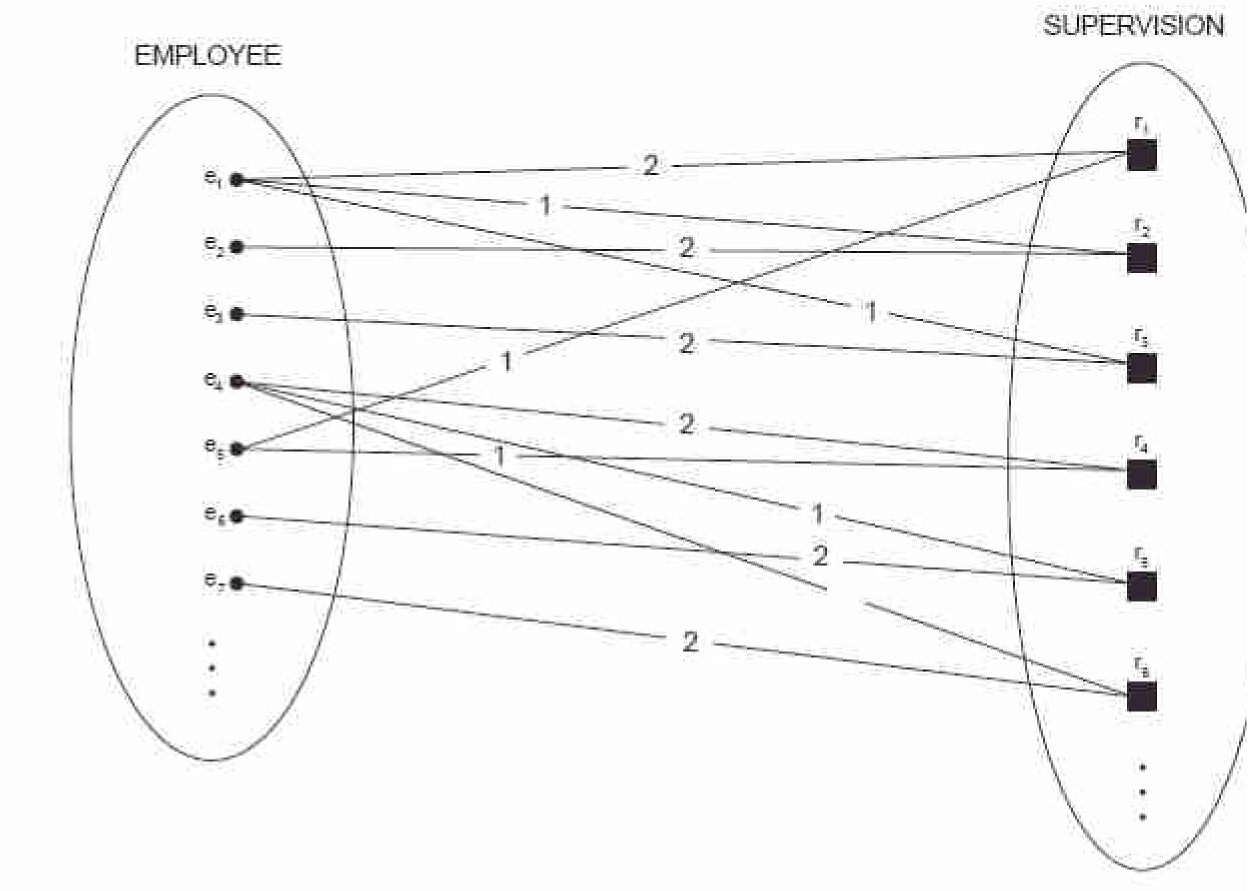




- **Role:** **Each entity participating in a relationship has a role.**
  E.g. Employee plays the role of worker and department plays the role of employer in the WORKS_FOR relationship.
  Note: Role names are more important in recursive relationships.

- **Recursive relationship: If one entity plays two roles then the relationship is called recursive relation.**

Figure 3.11 The recursive relationship SUPERVISION, where the EMPI
entity type plays the two roles of supervisor (1) and supervisee (2



- **Structural constraints on relationships: We have two relationship constraints.**

(a) Cardinality ratio constraint (1-1, 1-N, M-N)
(b) Participation constraint 1. **Total participation (existence ependency): All the tuples of the participating entity relates with the tuples of related entity represented with double line.** 2. **Partial participation : Some tuples of the participating entity relates with all the tuples of related entity represented with single line.** - **Note: Participation constraint is represented by (min,max) pair.** **If min value = 0 then the is patial paticipation.** **If min value >= 0 then the is total paticipation.**

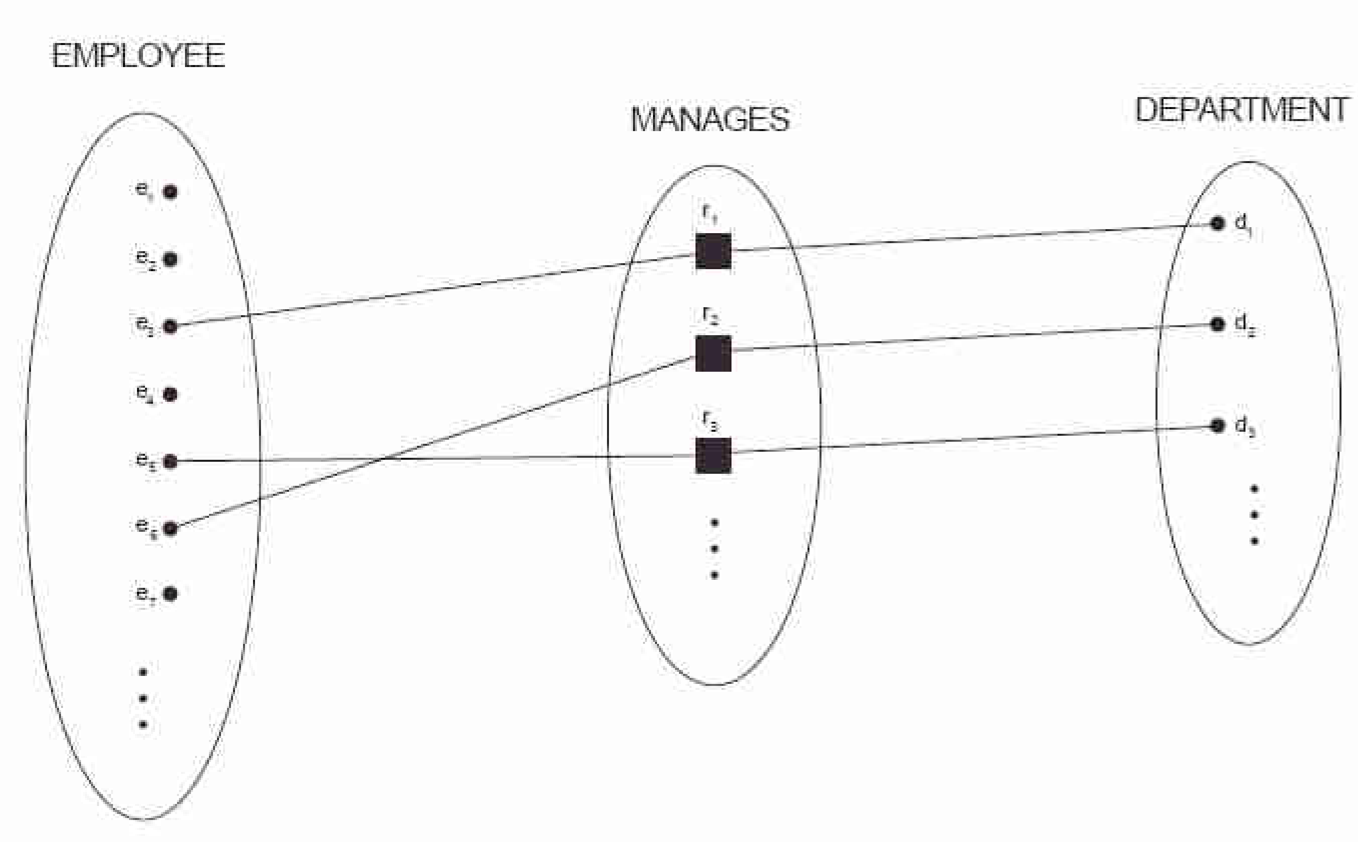
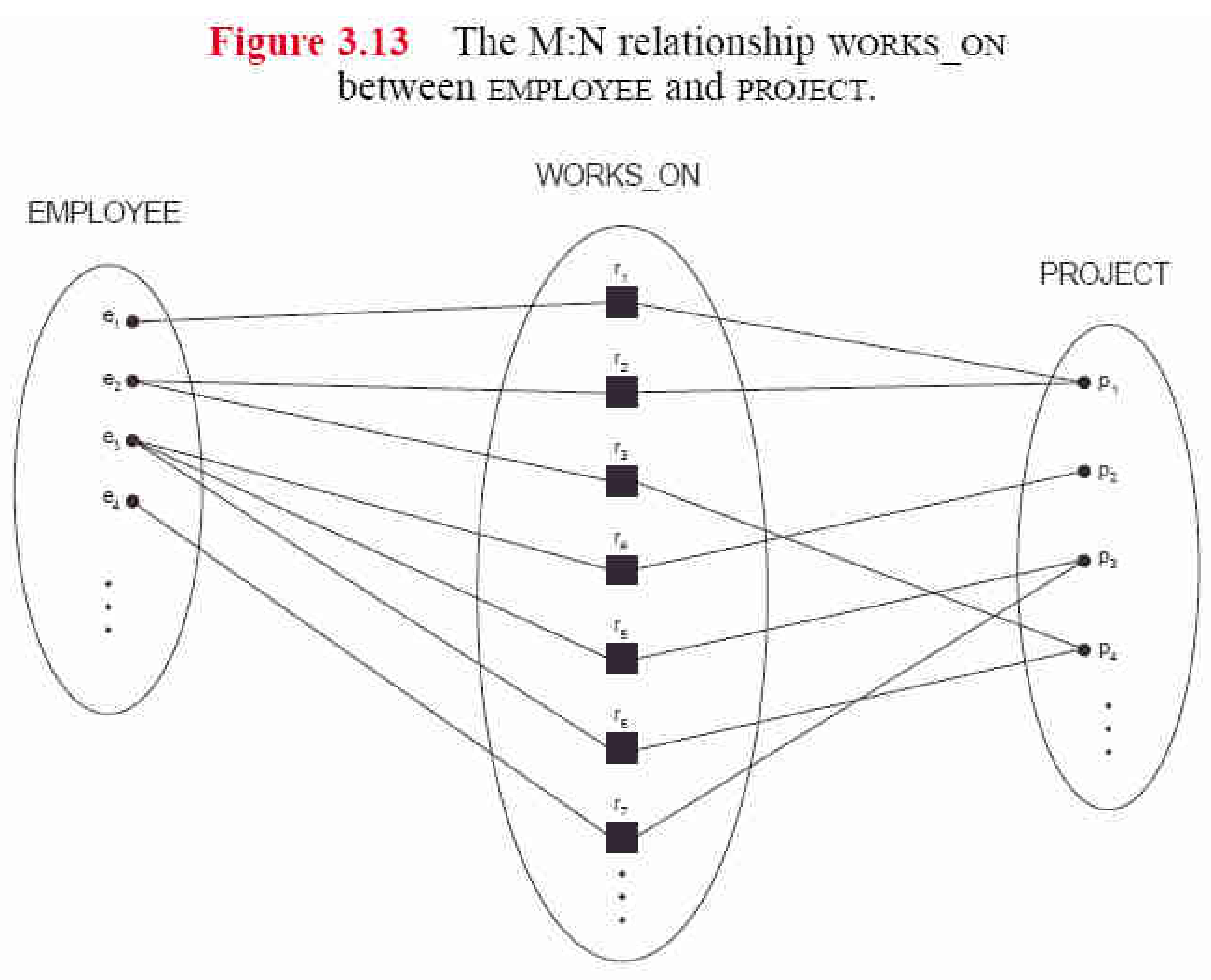


Figure 3.12 The 1:1 relationship MANAGES, with partial participation of employee and total participation of DEPARTMENT.

Figure 3.13 The M:N relationship WORKS_ON between EMPLOYEE and PROJECT.



- **Attributes of relationships**:

1. **Note: Relationship types can have attributes, similar to entity types.**

E.g. Hours attribute for WORKS_ON relationship

2. **If relationship is 1-N or 1-1, these attributes can be be migrated to the entity sets involved in the relationship.**
   a) **1-N: migrate to N side**
   b) **1-1: migrate to either side**

3.2 Weak Entity types — IMP

- **Weak entity: Entity that do not have key attribute.**

E.g. DEPENDENT

- **Weak entity types: Entity Types that do not have key attributes.**

- **Weak entities are identified by being related to other entity sets called identifying owner. This relationship is called identifying relationship.**

- **Partial key: Attributes that uniquely identify entities within the identifying relationship.**

  (or)

  A partial key is a key used to distinguish one record with another record but not guarentted to be a key when new records are inserted.

- **A weak entity type always has TOTAL participation.**

3.4 ER Schema Constructs (or) Symbols to write ER diagrams — IMP

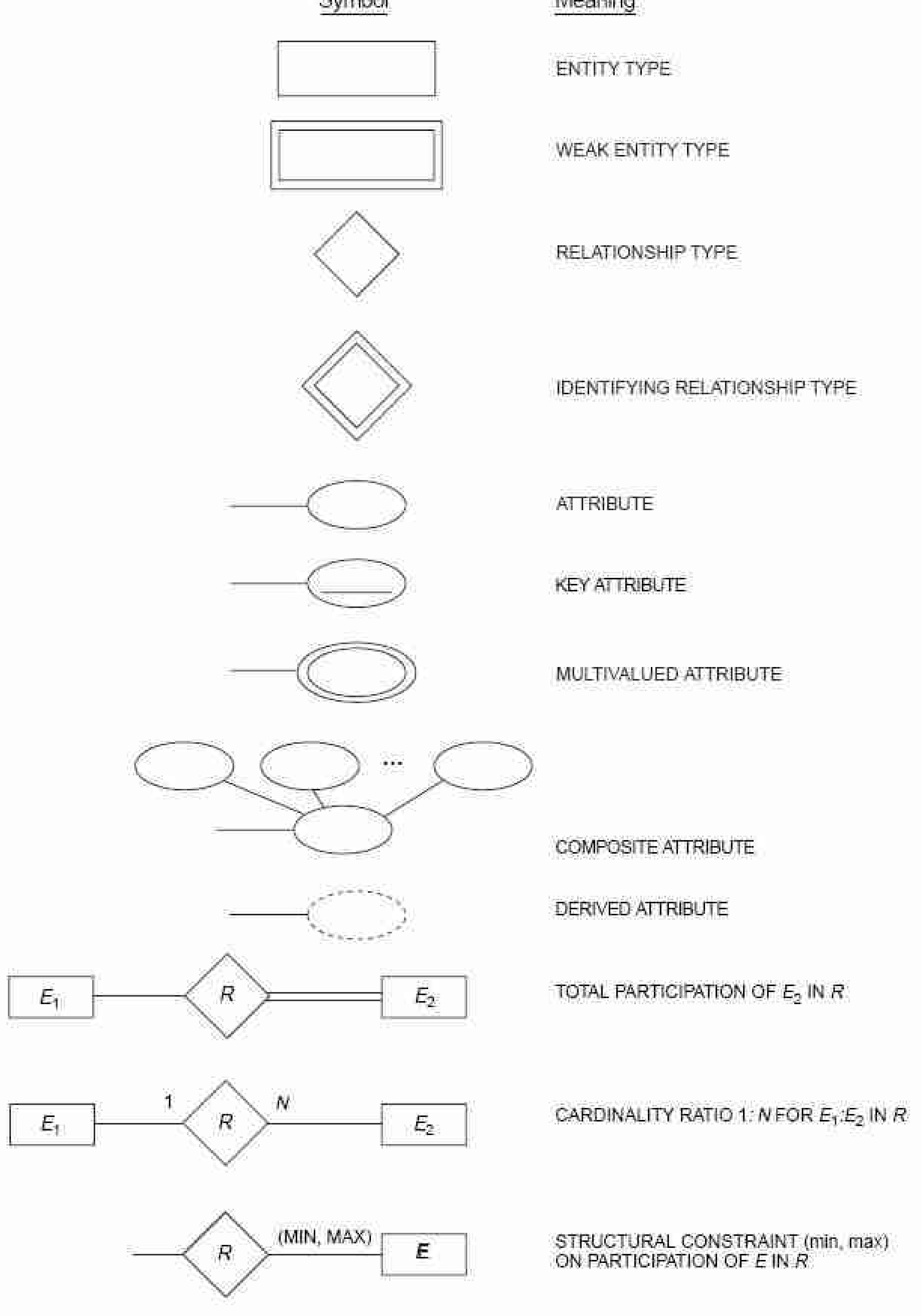
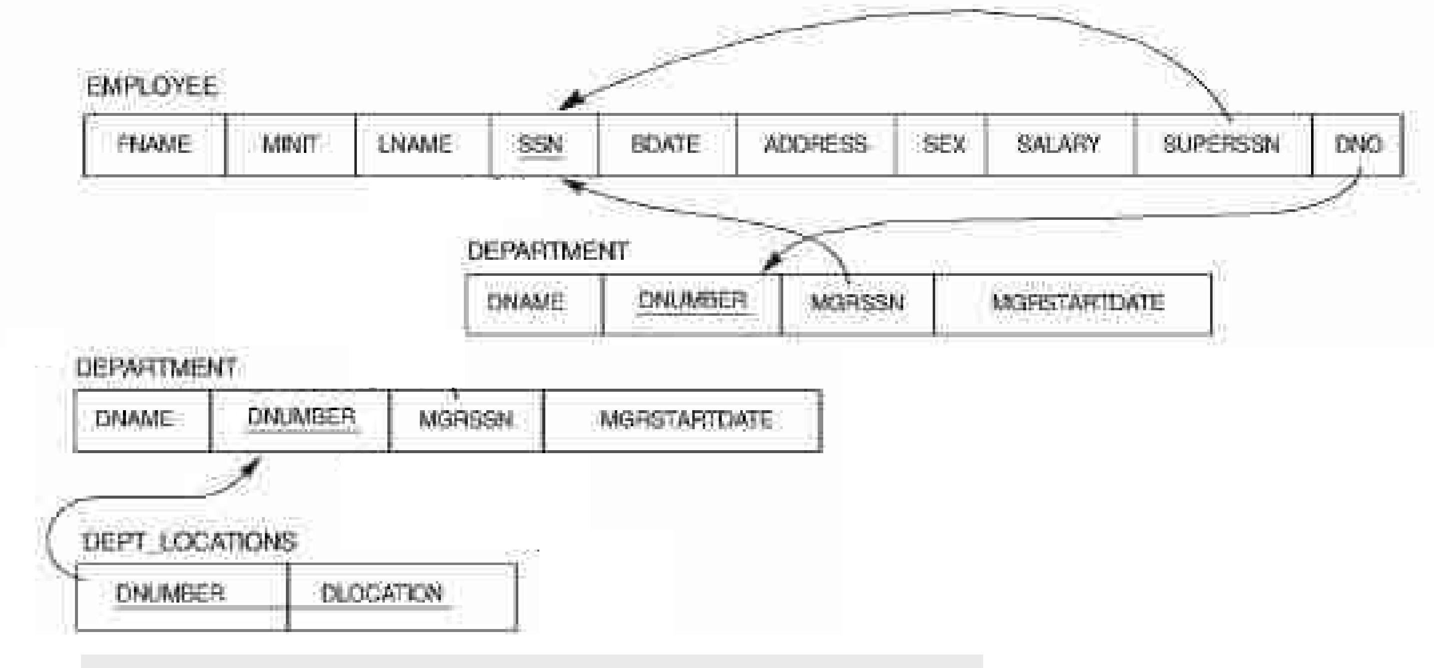


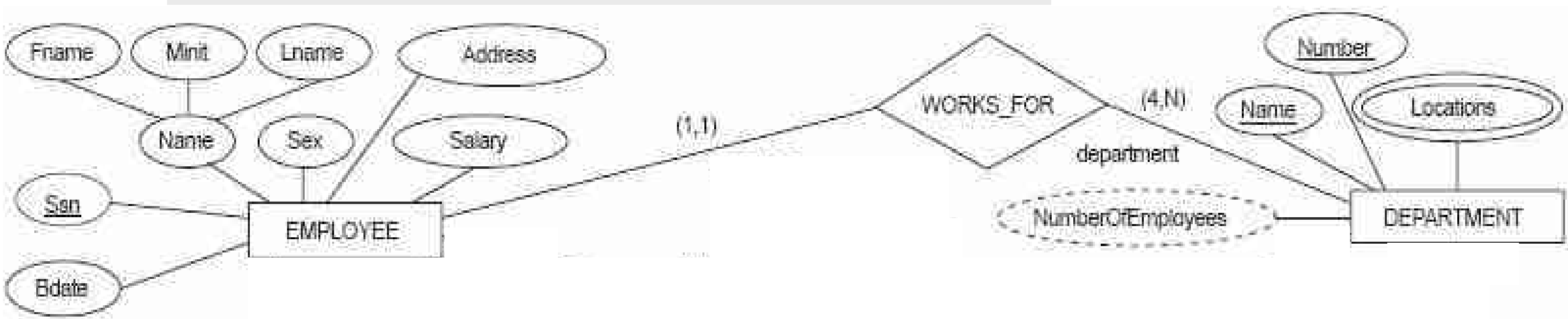
Figure 3.14 Summary of ER diagram notation.

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John		Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin		Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia		Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer		Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh		Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce		English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad		Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James		Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1



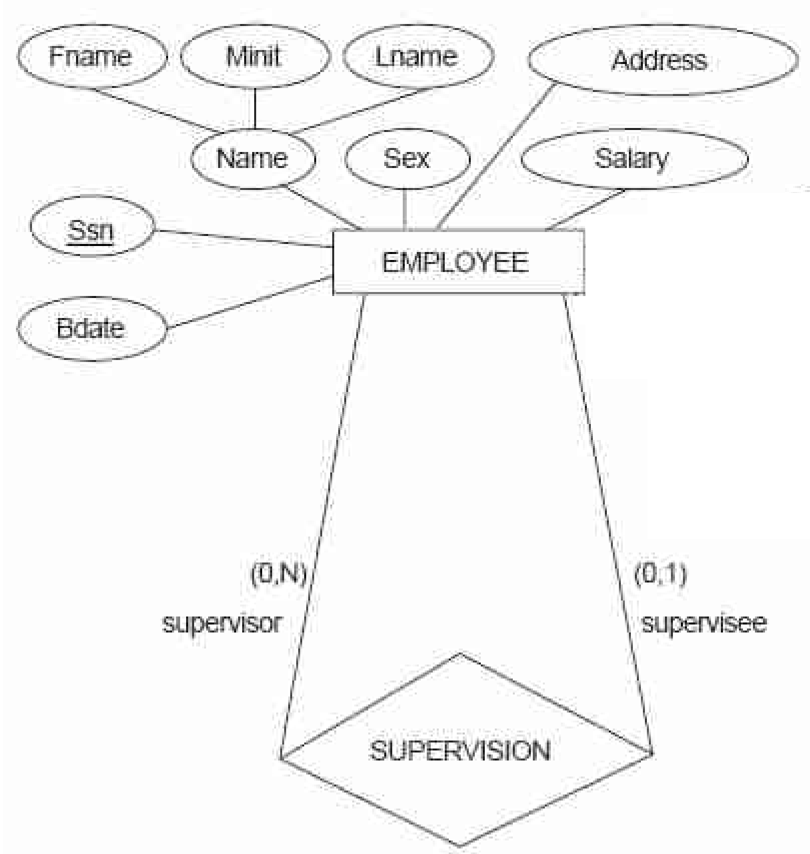
- **Constraints:**

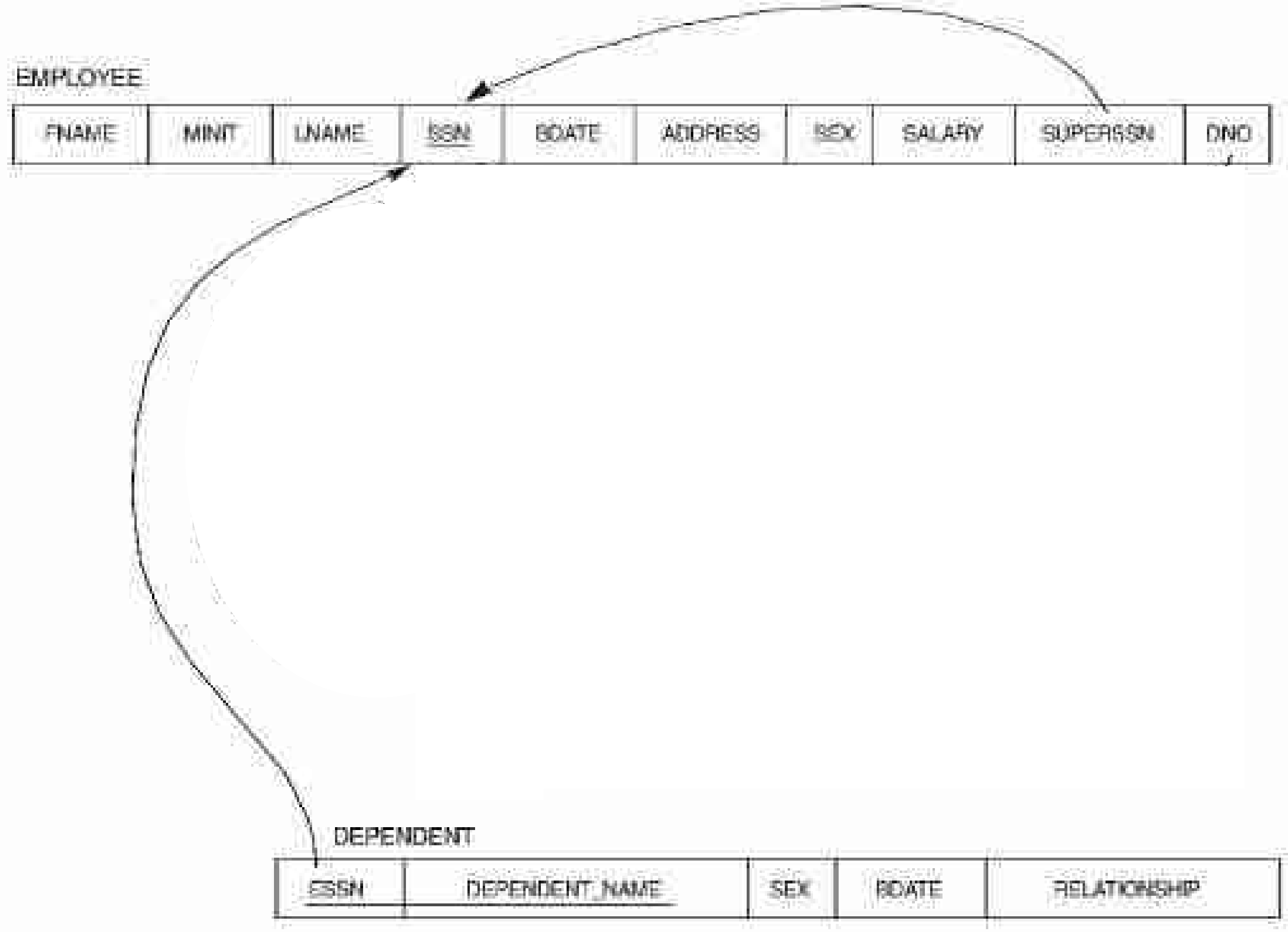
1. **An employee work for one department.**
2. **Department may have any number of employees.**
3. **Adepartment can be located any number of locations.**



- **Constraints:**

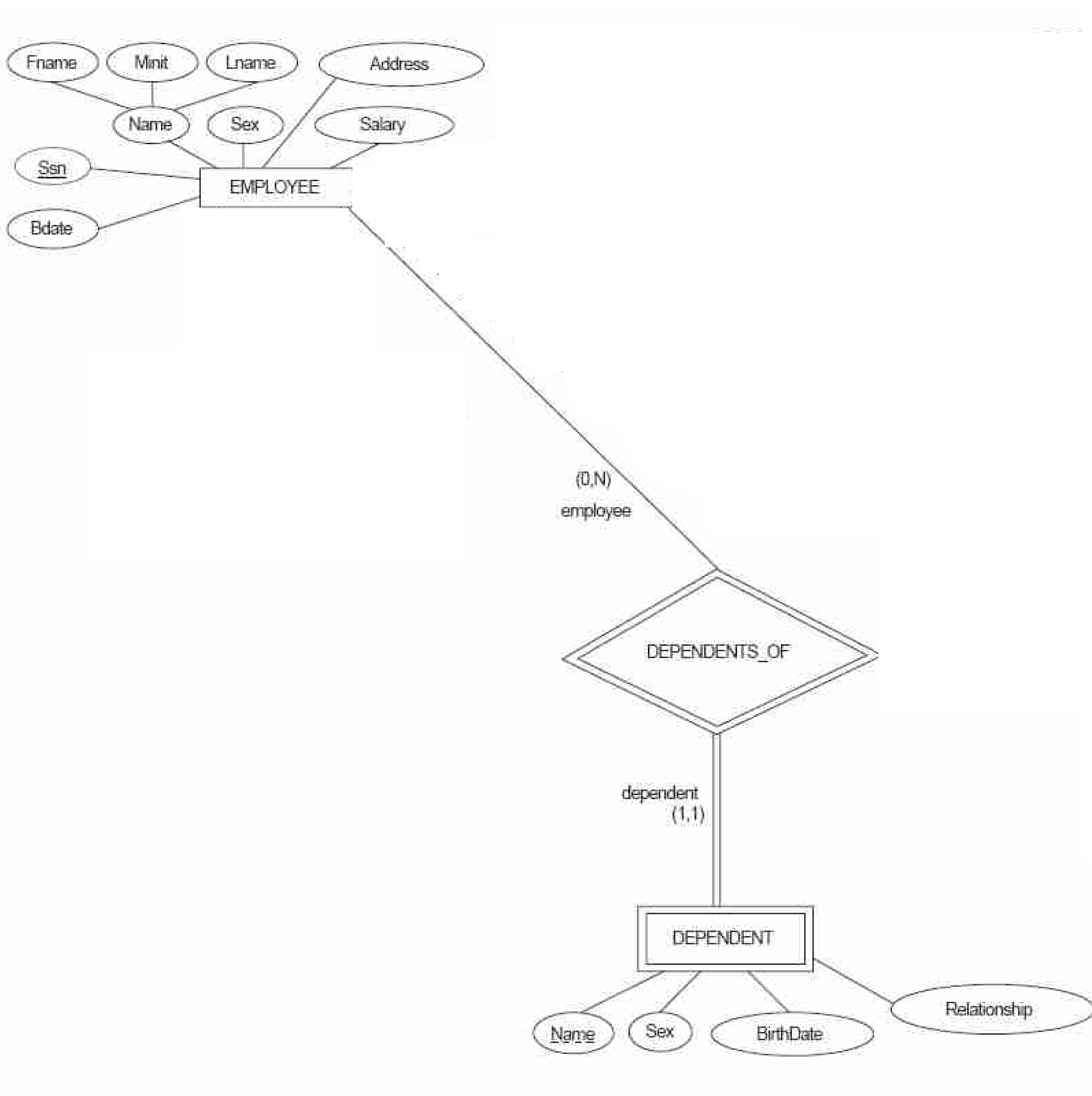
1. **An employee who is manager is supervisor.**
2. **Manager supervise any number of employees.**
3. **A manager is supervised by another manager.**





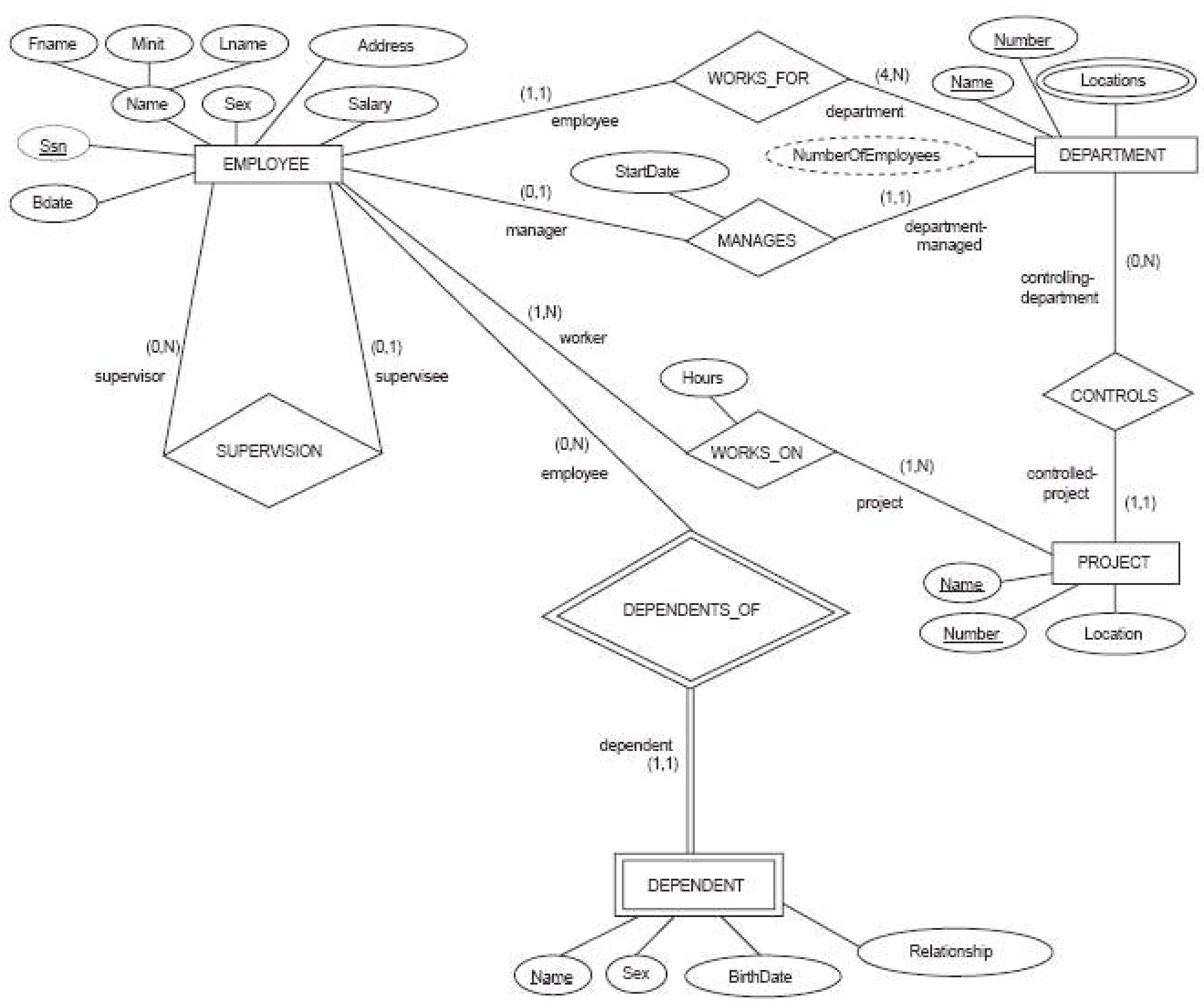
- **Constraints**

1. **An employee may not have dependents.**
2. **An employee may have any number of dependents.**
3. **Every dependent must belongs to an employee.**



Fname
Minit
Lname
Address
Name
Sex
Salary
Ssn
EMPLOYEE
Bdate
(0,N)
employee
DEPENDENTS_OF
dependent
(1,1)
DEPENDENT
Name
Sex
BirthDate
Relationship

Figure 3.15 ER diagram for the COMPANY schema, with all role names included and with structural constraints on relationships specified using the alternate notation (min, max).



- **Relationship attribute:** **Attribute derived from the relationship.**

E.g. Hours, StratDate

Functional Dependency: The functional dependency is a relationship that exists between two attributes of relation.

NORMALIZATION

First Normal Form

- **First Normal Form: A relation is said to be in first normal form if and only if it contains atomic or scalar values.**

SCP

S#	STATUS	CITY	P#	QTY
S1	{ 20,20,20,20,20,20 }	LONDON	{ P1,P2,P3,P4,P5,P6 }	{ 200,200,400,200,100,100 }
S2	{10,10 }	PARIS	{ P1,P2 }	{ 300,400 }
S3	10	PARIS	P2	200
S4	{ 20,20,20 }	LONDON	{ P2,P4,P5 }	{ 200,300,400 }

Note: The relation should not contain sets

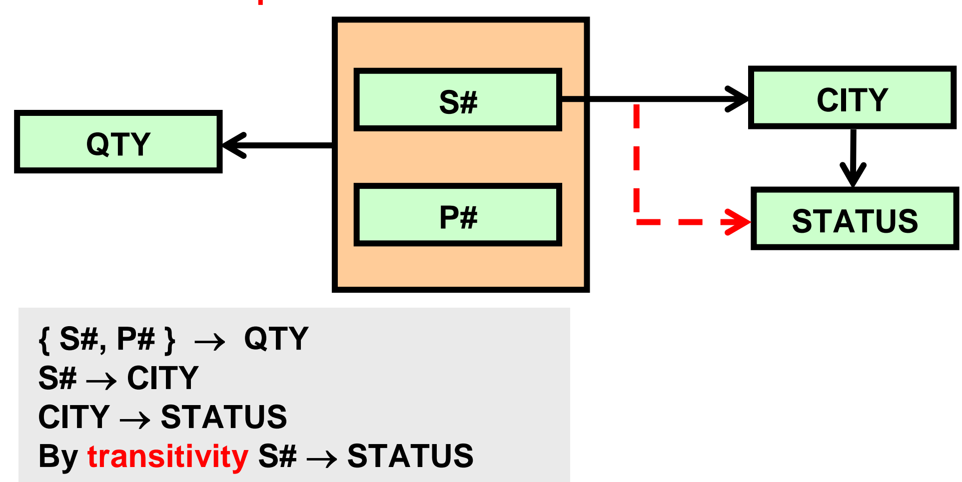
FIRST

S#	STATUS	CITY	P#	QTY
S1			P1	200
S1			P2	200
S1			P3	400
S1			P4	200
S1			P5	100
S1			P6	100
S2			P1	300
S2			P2	400
S3			P2	200
S4			P2	200
S4			P4	300
S4			P5	400

- The relation FIRST stated as

FIRST{ S#, STATUS, CITY, P#, QTY }
PRIMARY KEY { S#, P# }

- Functional Dependencies in relation FIRST are



{ S#, P# } → QTY
S# → CITY
CITY → STATUS
By transitivity S# → STATUS

- **UPDATE ANOMALIES**

INSERT: We can not insert particular suppliers located in particular city until the supplier supplies at least one part.

UPDATE: If supplier S1 moves from LONDON to AMSTERDAM, we need to change city as AMSTERDAM for every supplier S1 otherwise the database contains inconsistent values or data.

DELETE: If we delete first tuple of S1 among six tuples we lost the information of part P1.

Second Normal Form

- **The relation FIRST is projected into two relations SECOND, SP**

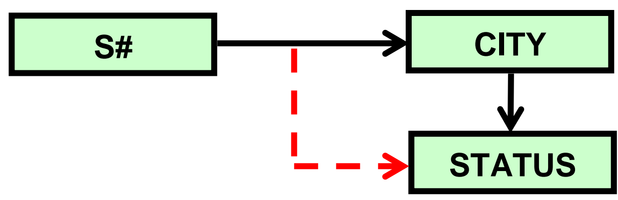
SECOND

S#	STATUS	CITY
S1	20	LONDON
S2	10	PARIS
S3	10	PARIS
S4	20	LONDON
S5	30	ATHENS

- **The relation SECOND is stated as**

SECOND { S#, STATUS, CITY }
PRIMARY KEY S#

- **Functional Dependencies in relation SECOND are**



S# → CITY
CITY → STATUS
By transitivity S# → STATUS

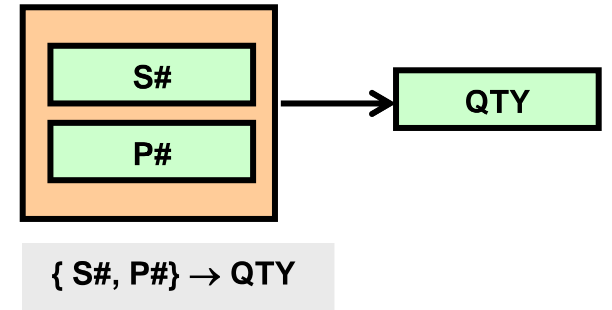
SP

S#	P#	QTY
S1	P1	200
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

- The relation SP is stated as

SP { S#, P#, QTY }
PRIMARY KEY { S#,P# }

- Functional Dependencies in relation SP are



{ S#, P#} → QTY

Second Normal Form: A relation is said to be in 2NF if and only if it is in 1NF and every non key attribute is irreducibly (loss less) depend on key attribute (Primary key).

- **UPDATE ANOMALIES**

INSERT: We can not insert the fact that a particular city have particular status.

E.g. We can not state that any supplier in ROME must have status of 50 until we have some suppliers actually located in that city.

UPDATE: If we need to change STATUS of LONDON from 20 to 40, we should change for all the tuples otherwise the database consists inconsistent data.

DELETE: If we delete second tuple of city PARIS, the information of supplier S2 will be lost.

Third Normal Form

Third Normal Form: A relation is said to be in 3NF if and only if it is in 2NF and every non key attribute non transitively depend upon the key attribute.

Decomposition - A

- Decompose the relation SECOND into two relations SS and CS

SS

S#	STATUS
S1	20
S2	10
S3	10
S4	20
S5	30

- The relation SS is stated as

  SS { S#, STATUS }
  PRIMARY KEY S#

- Functional Dependencies in relation SECOND are

S# → STATUS

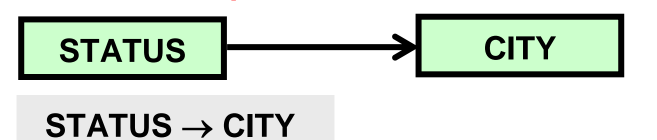
SC

STATUS	CITY
20	LONDON
10	PARIS
30	ATHENS

- The relation SC is stated as

  SC { STATUS, CITY }
  PRIMARY KEY STATUS

- Functional Dependencies in relation SC are



STATUS → CITY

- This decomposition is not correct since the Functional Dependency S# → STATUS produces transitivity.

Decomposition - B

- Decompose the relation SECOND into two relations SC and CS

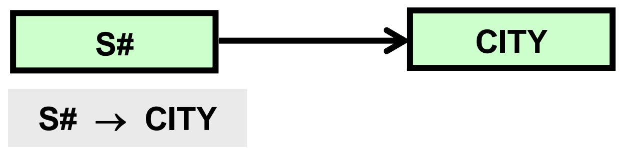
SC

S#	CITY
S1	LONDON
S2	PARIS
S3	PARIS
S4	LONDON
S5	ATHENS

- The relation SC is stated as

SC { S#, CITY }
PRIMARY KEY S#

- Functional Dependencies in relation SC are



S# → CITY

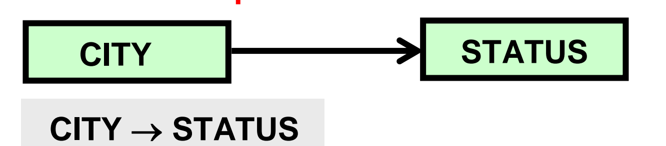
CS

CITY	STATUS
LONDON	20
PARIS	10
ATHENS	30

- The relation CS is stated as

SC { CITY, STATUS }
PRIMARY KEY CITY

- Functional Dependencies in relation CS are



CITY → STATUS

- This decomposition is correct .Therefore the relations SC and CS satisfies 3NF.

- UPDATE ANOMALIES

  INSERT: We can not insert the fact that a particular city have particular status.

  E.g. We can not state that any supplier in ROME must have status of 50 until we have some suppliers actually located in that city.

  UPDATE: If supplier S1 moves from LONDON to AMSTERDAM we need to change city as AMSTERDAM for every supplier S1 otherwise database contains inconsistent data.

  DELETE: If we delete second tuple of city PARIS the information of S2 will be lost.

Dependency Preservation

- Consider the relation SECOND

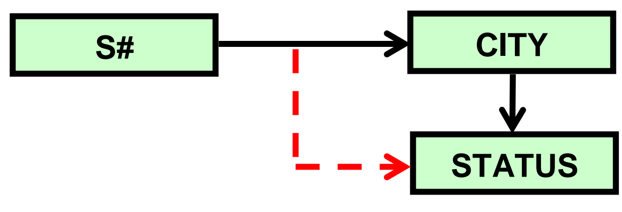
SECOND

S#	STATUS	CITY
S1	20	LONDON
S2	10	PARIS
S3	10	PARIS
S4	20	LONDON
S5	30	ATHENS

- The relation SECOND is stated as

SECOND { S#, STATUS, CITY }
PRIMARY KEY S#

- Functional Dependencies in relation SECOND are



S# → CITY
CITY → STATUS
By transitivity S# → STATUS

- Decompose the relation SECOND into two relations SC and CS

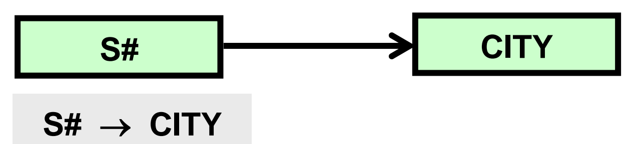
SC

S#	CITY
S1	LONDON
S2	PARIS
S3	PARIS
S4	LONDON
S5	ATHENS

- The relation SC is stated as

SC { S#, CITY }
PRIMARY KEY S#

- Functional Dependencies in relation SC are



S# → CITY

CS

CITY	STATUS
LONDON	20
PARIS	10
ATHENS	30

- The relation CS is stated as

SC { CITY, STATUS }
PRIMARY KEY CITY

- Functional Dependencies in relation CS are

CITY → STATUS

- Dependency preservation: Functional dependencies of the projected tables are implied by the dependencies of original table.

- From the above tables the FD's of SC and CS are implied by the FD's of SECOND. Therefore dependencies are preserved.

BCNF (Boycee Code Normal Form)

BCNF: A relation is said to be in BCNF if and only if it is in 3NF and only determents are candidate keys.

- Candidate key : An attribute (or) group of attributes which is used to distinguish one record with another record.

- Consider the relation SECOND

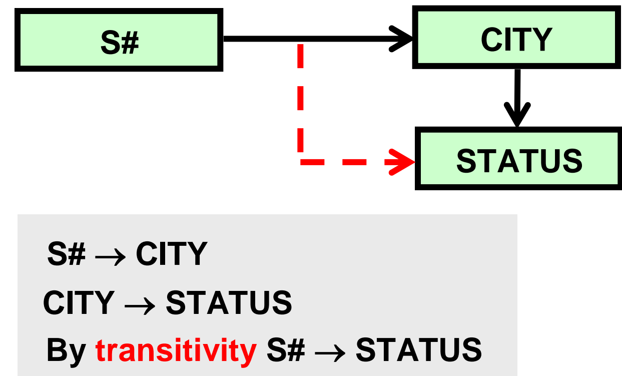
SECOND

S#	STATUS	CITY
S1	20	LONDON
S2	10	PARIS
S3	10	PARIS
S4	20	LONDON
S5	30	ATHENS

- The relation SECOND is stated as

SECOND { S#, STATUS, CITY }
PRIMARY KEY S#

- Functional Dependencies in relation SECOND are



S# → CITY
CITY → STATUS
By transitivity S# → STATUS

- The above relation does not doesn't satisfies 3NF and all the determinants are not the candidate keys. Therefore the relation SECOND is not in BCNF.

- The relation SECOND is projected into two relations SC and CS

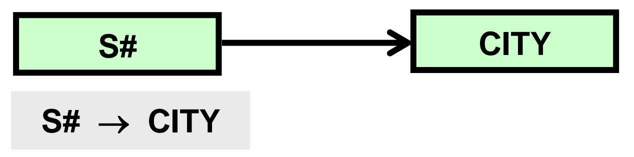
SC

S#	CITY
S1	LONDON
S2	PARIS
S3	PARIS
S4	LONDON
S5	ATHENS

- The relation SC is stated as

SC { S#, CITY }
PRIMARY KEY S#

- Functional Dependencies in relation SC are



S# → CITY

- In the relation SC is the only determinant S# is the primary key .

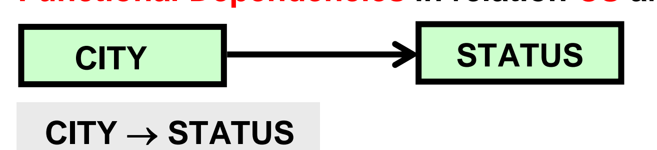
CS

CITY	STATUS
LONDON	20
PARIS	10
ATHENS	30

- The relation CS is stated as

SC { CITY, STATUS }
PRIMARY KEY CITY

- Functional Dependencies in relation CS are



CITY → STATUS

- In the relation CS is the only determinant CITY is the primary key .
- There fore the relations SC and CS satisfies BCNF and all the determinants are candiadate keys

---

4NF (Fourth Normal Form)

- Consider the relation UCTX (Un normalized relation)

UCTX

COURSE	TEACHER	TEXT
PHYSICS	{Prof. GREEN, Prof. BROWN}	{ BASIC MECHANICS, PRINCIPLES OF OPTICS}
MATHS	{Prof.GREEN}	{BASIC MECHANICS, VECTOR ANALYSIS, TRIGNOMETRY}

- Constraints on table UCTX

1. A specified course can be taught by specified teacher and uses all the specified textbooks as references.
2. A teacher can teach any number of courses.
3. There are no functional dependencies for the table UCTX.

- Now we are normalizing UCTX table as CTX

CTX

COURSE	TEACHER	TEXT
PHYSICS	Prof. GREEN	BASIC MECHANICS
PHYSICS	Prof. GREEN	PRINCIPLES OF OPTICS
PHYSICS	Prof. BROWN	BASIC MECHANICS
PHYSICS	Prof. BROWN	PRINCIPLES OF OPTICS
MATHS	Prof. GREEN	BASIC MECHANICS
MATHS	Prof. GREEN	VECTOR ANALYSIS
MATHS	Prof. GREEN	TRIGNOMETRY

- The relation CTX is stated as

CTX {COURSE, TEACHER,TEXT }
CANDIDATE KEY {COURSE, TEACHER, TEXT}

- The above relation satisfies 1NF,2NF, 3NF and BCNF since the determinant itself is the candidate key.

- **Update anomalies:**
To add information about a teacher who teaches a course PHYSICS, it is necessary to create two tuples one for each of the text books BASIC MECHANICS and PRINCIPLES OF OPTICS.

- MVD'S: Multi valued dependencies are general dependencies based on the attributes and their domains.

- Let R be a relation and let A,B,&C be arbitrary subsets of R then we can say that A⟶>>B if and only if the set B values matching with pair (A,C) dependents only on A value independent of C value.

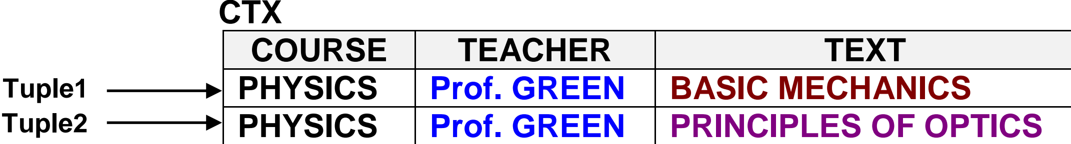
COURSE⟶>>TEXT

COURSE⟶>>TEACHER

The above MVD's exits in CTX.

- Consider first two tuples from the relation CTX.

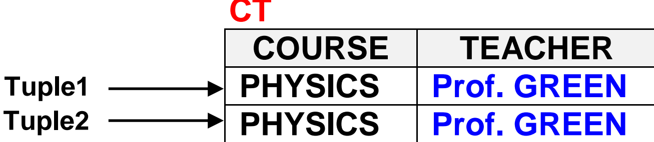
CTX



	COURSE	TEACHER	TEXT
Tuple1 →	PHYSICS	Prof. GREEN	BASIC MECHANICS
Tuple2 →	PHYSICS	Prof. GREEN	PRINCIPLES OF OPTICS

- Consider two subsets CT and CX from CTX

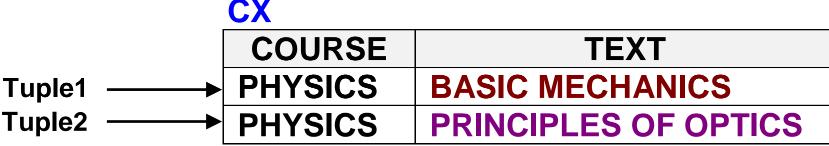
CT



	COURSE	TEACHER
Tuple1 →	PHYSICS	Prof. GREEN
Tuple2 →	PHYSICS	Prof. GREEN

Tuple1 = Tuple2

CX



	COURSE	TEXT
Tuple1 →	PHYSICS	BASIC MECHANICS
Tuple2 →	PHYSICS	PRINCIPLES OF OPTICS

Tuple1 ≠ Tuple2

- Therefore the relation CTX violating the condition of FD.
- Project the table CTX into CT and CX

CT

COURSE	TEACHER
PHYSICS	Prof. GREEN
PHYSICS	Prof. BROWN
MATHS	Prof. GREEN

- The relation CT is stated as

CT {COURSE, TEACHER}
CANDIDATE KEY {COURSE, TEACHER}

CX

COURSE	TEXT
PHYSICS	BASIC MECHANICS
PHYSICS	PRINCIPLES OF OPTICS
MATHS	BASIC MECHANICS
MATHS	VECTOR ANALYSIS
MATHS	TRIGNOMETRY

- The relation CX is stated as

CTX {COURSE, TEXT }
CANDIDATE KEY {COURSE, TEXT}

- Therefore the relations CT and CX satisfies 1NF,2NF,3NF,BCNF and preserves **MVD's** and **FD's** .

- Therefore the relations CT and CX satisfies 4NF.

4NF: The relation is said to be in 4NF if and only it is in BCNF and must have both **MVD's** and **FD's**.

Fagin's theorem: Let R=A,B,C be a relation where A,B,C are set of attributes the R = join of its projection on {A,B} and {A,C} if and only if R satisfies A →>> B and A→>> C

5NF (Fifth Normal Form)

Join Dependencies

- Consider the following relation

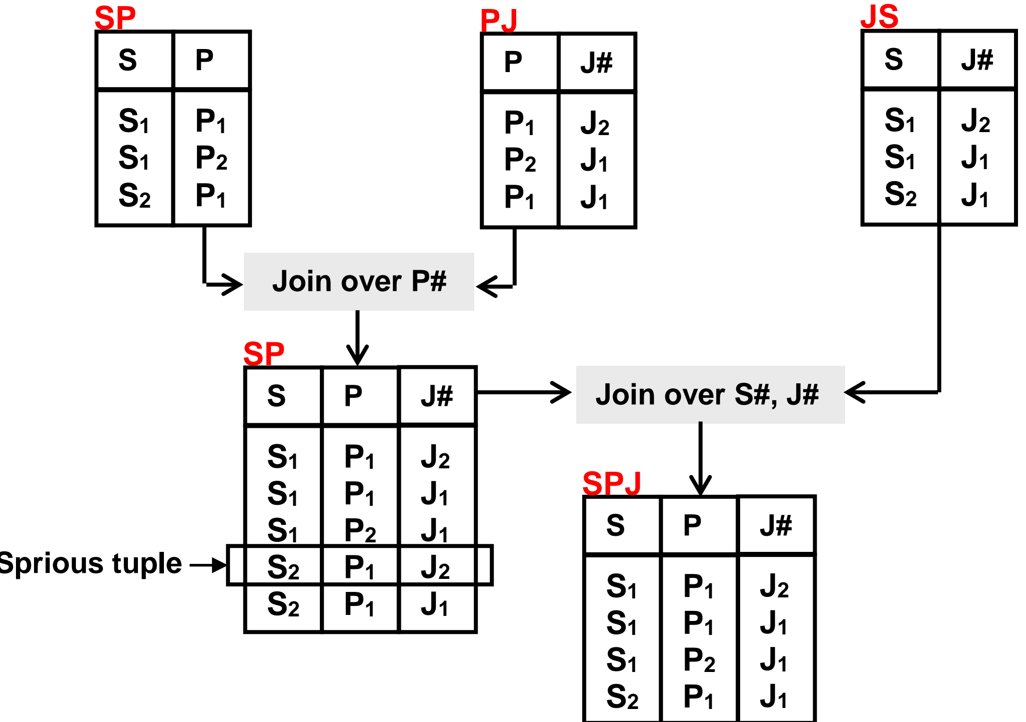
SPJ

S#	P#	J#
S_1	P_1	J_2
S_1	P_2	J_1
S_2	P_1	J_1
S_1	P_1	J_1

- The relation SPJ is stated as

SPJ {S#,P#,J#}
CANDIDATE KEY { S#,P#,J#}

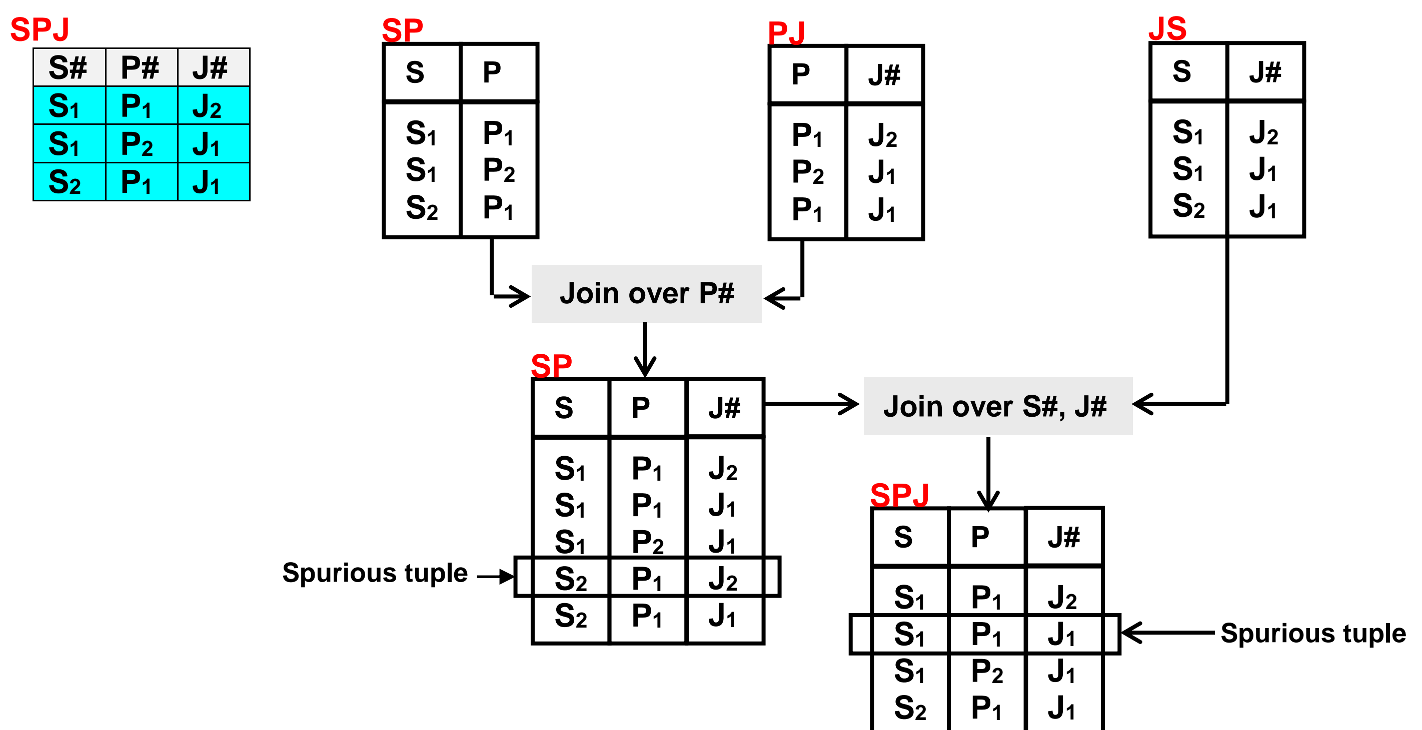
- **Join Dependencies**: Let R be a relation and let A,B,…,Z be arbitrary subsets of set of attributes of R. Then we can say that R satisfies join dependency *(A,B,…,Z) if and only if R is equal to join of its projections on A,B,…,Z



- Constraint on the following table is

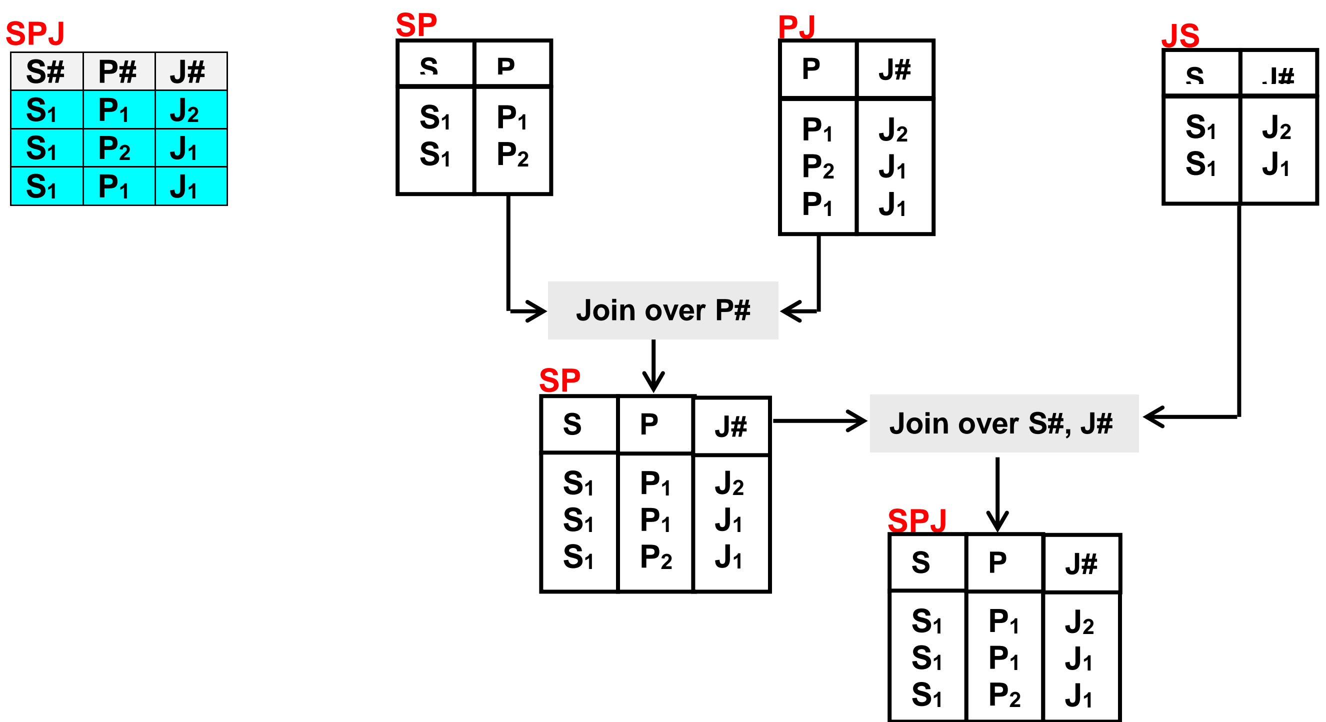
If the pair (S_1, P_1) appear in SP, the pair (P_1 ,J_1) appear in PJ the the pair (J_1, S_1) appear in JS then the triple (S_1, P_1 ,J_1) appear in SPJ

Table violating the specified constraint



SPJ

S#	P#	J#
S_1	P_1	J_2
S_1	P_2	J_1
S_2	P_1	J_1



5NF: A relation R is in 5NF is also called PJNF (Projection Join Normal From) if and only if every join dependency in R is implied by the candidate key of R.

- Consider the relation SPJ

  SPJ {S#,P#,J#}
  CANDIDATE KEY { S#,P#,J#}

- SPJ is projected into SP,PJ and JS

  SP {S#,P#}
  CANDIDATE KEY { S#,P# }

  PJ {S#,P#}
  CANDIDATE KEY { P#,J# }

  JS {J#,S#}
  CANDIDATE KEY { J#,S# }

- Candidate keys of SP, PJ and JS is not implied by the candidate key of SPJ. Therefore SP, PJ and JS are not satisfies 5NF.

- Consider the relation

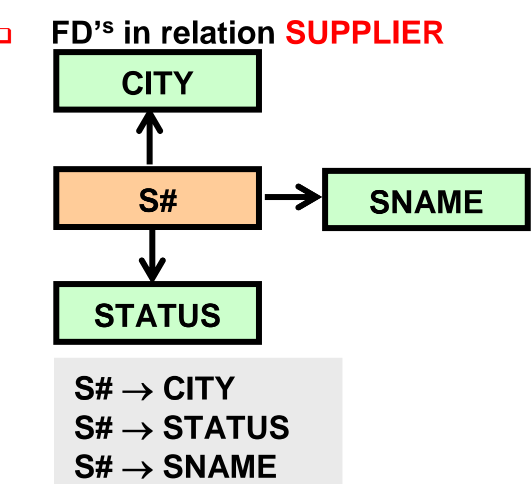
SUPPLIER

S#	SNAME	STATUS	CITY
S_1 S_2	SMITH JOHN	20 20	LONDON PARIS

- The relation SUPPLIER is stated as

SPJ {S#,SNAME,STAYUS,CITY}
CANDIDATE KEY S#

- FD'S in relation SUPPLIER



S# → CITY
S# → STATUS
S# → SNAME

- The above relation satisfies 1NF,2NF, 3NF, BCNF (No need to check 4NF since there are FD'S).

- The relation SUPPLIER is projected into SS and SC.

SS

S#	SNAME	STATUS
S1	SMITH	20
S2	JOHN	10

SS{ S#,SNAME,STATUS}
CANDIDATE KEY S#

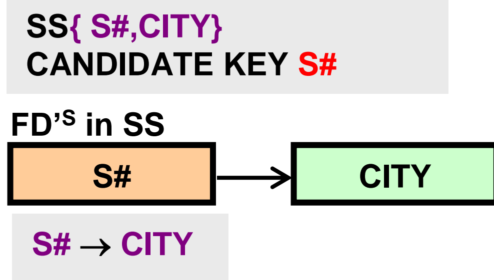
FD'S in SS

S#
SNAME
STATUS

S# → SNAME
S# → STATUS

SC

S#	CITY
S1	LONDON
S2	PARIS



- The relations SS and SC satisfies 5NF since the candidate keys of SS and CS is implied by candidate key of SUPPLIER.

Domain Key Normal Form

1. This is proposed by FAGIN and it is also called FAGIN NF.
2. It is not defined in terms of FD's ,MVD's,JD's .

A relation R is said to be in DKNF if and only if it should contain KEY CONSTRAINT and DOMAIN CONSTRAINT.

- Key constraint : The relation must have primary key .
- Domain constraint : The set of values associated for an attribute must have finite values.

Transaction Processing

- Transaction : Logical unit of sequence of processing.
- Transaction operations: Insert, Delete, Update & Retrieve.
- Transaction boundaries: Begin, End.
- Begin transaction: Transaction begins when user enters into SQL plus environment.
- End transaction: Transaction ends when one of the following statements are executed.

(a) Exit
(b) Commit
(c) Rollback

Topic 1: Transaction read & write

read_item(x)

- Reads a database item into a program variable.

1. Find the address of the disk block that contain data item x.
2. Copy the disk block contain the data item x into primary memory.
3. Copy data item x to program variable named x.

write_item(x)

- Writes the value of program variable x into database.

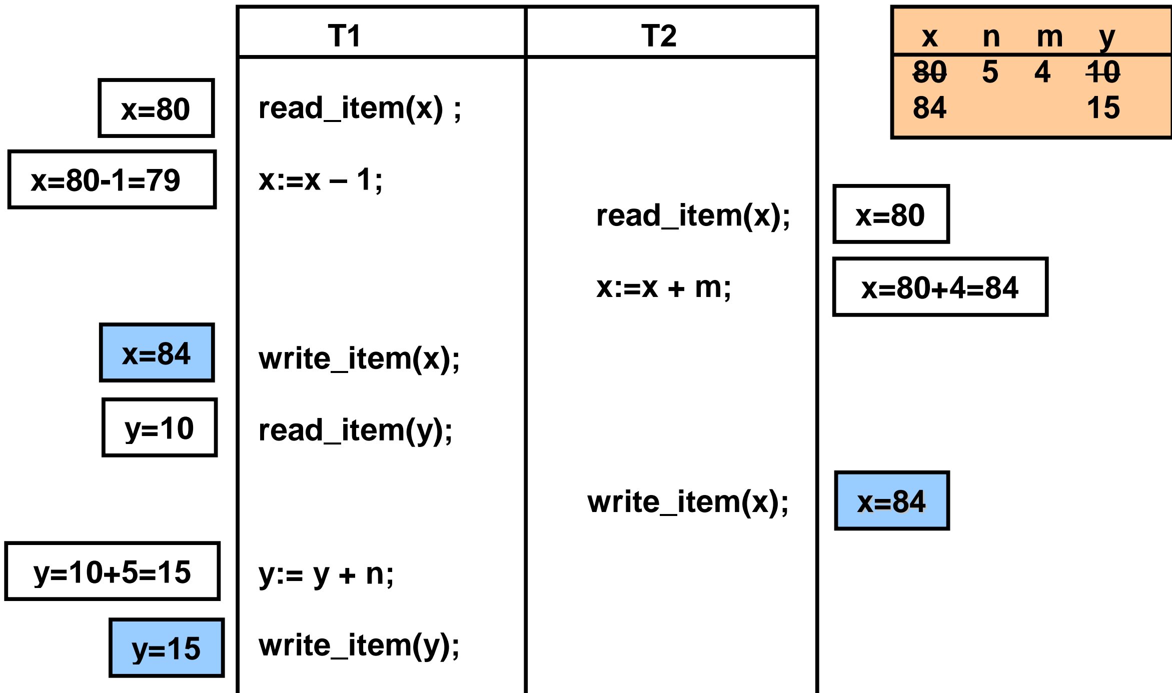
1. Find the address of the disk block that contain data item x.
2. Copy the disk block contain the data item x into primary memory.
3. Copy data item x to program variable named x.
4. Store the updated data item contained in disk block to secondary memory.

Topic2: What are the problems in transaction processing (or) What are the problems due to concurrency (or) Why concurrency control is needed [VIMP]

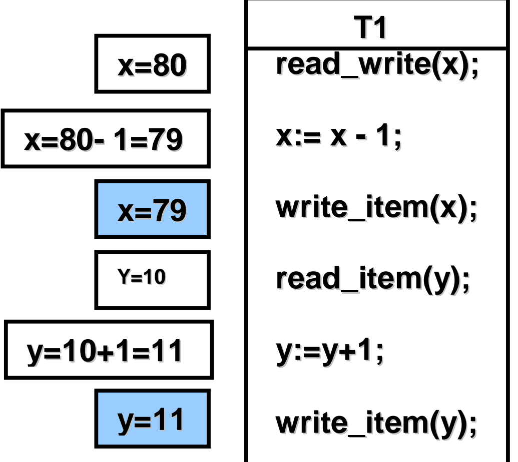
- Concurrency : When multiple transactions request single database item simultaneously, the updations of transactions produce inconsistent data.

The lost update problem

- The updations of interleaving transactions produce inconsistent data.



- Serial operations of transaction T1 is

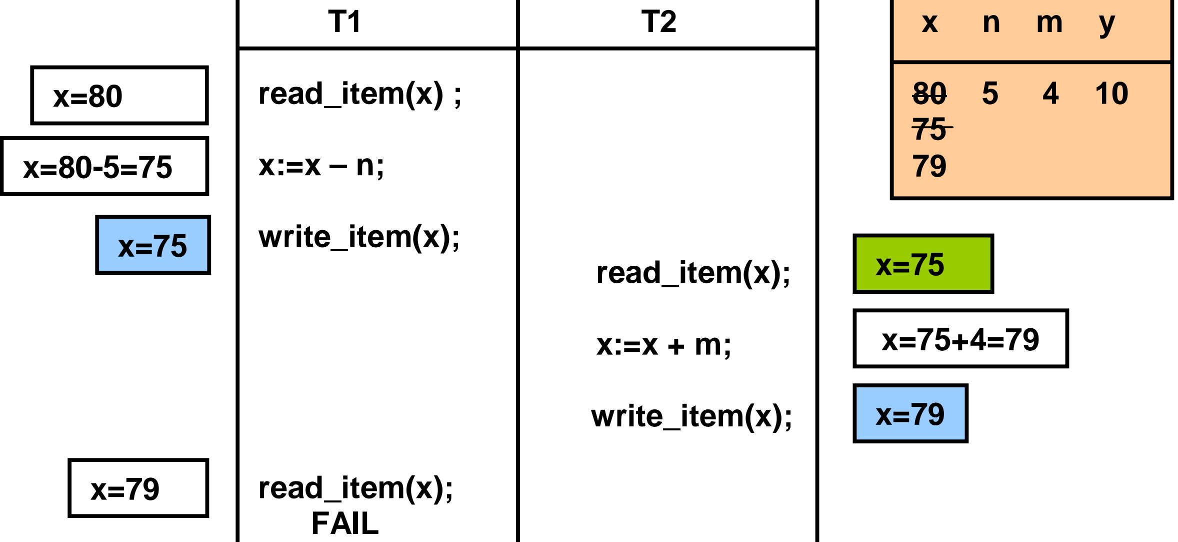


x	n	y
~~80~~	5	~~10~~
79		11

- The final result of serial transaction is x=79 but the interleaving transaction operation shows that x=84.

The temporary update problem (Dirty read)

- The problem occurs when one transaction update the database item and then fails



- If transaction T1 fails the database item roll backs to its previous state, but transaction T2 has read the incorrect value of x.

The incorrect summary problem

- A transaction T receives different values for two reads on the same data item.

y	sum	x	n	a
~~20~~ 25	0	~~10~~ 5	5	5

	T1	T2	
		sum:=0;	
		read_item(a);	a =5
		sum:=sum +a;	sum=0+5=5
		. . .	
x=10	read_item(x);		
x=10-5=5	x:=x - n;		
x =5	write_item(x);		
		read_item(x);	x = 5
		sum:=sum+x;	sum=0+5=5
		read_item(y);	y=20
		sum:=sum+y;	sum=0+20=20
y=20	read_item(y);		
y=20 +5=25	y:=y+n;		
y=25	write_item(y);		

Topic3: Types of transaction failures (or) Why recovery is needed [IMP]

- Types of failures

(a) Physical (or) Catastrophic
(b) Disk failure
(c) Transaction failure

Physical (or) Catastrophic failure

1. A/C failure.
2. Theft.
3. Sabotage.
4. Mounting wrong tape.
5. Over riding disks (or) taps.
6. Giving same name to the file.

Disk failure

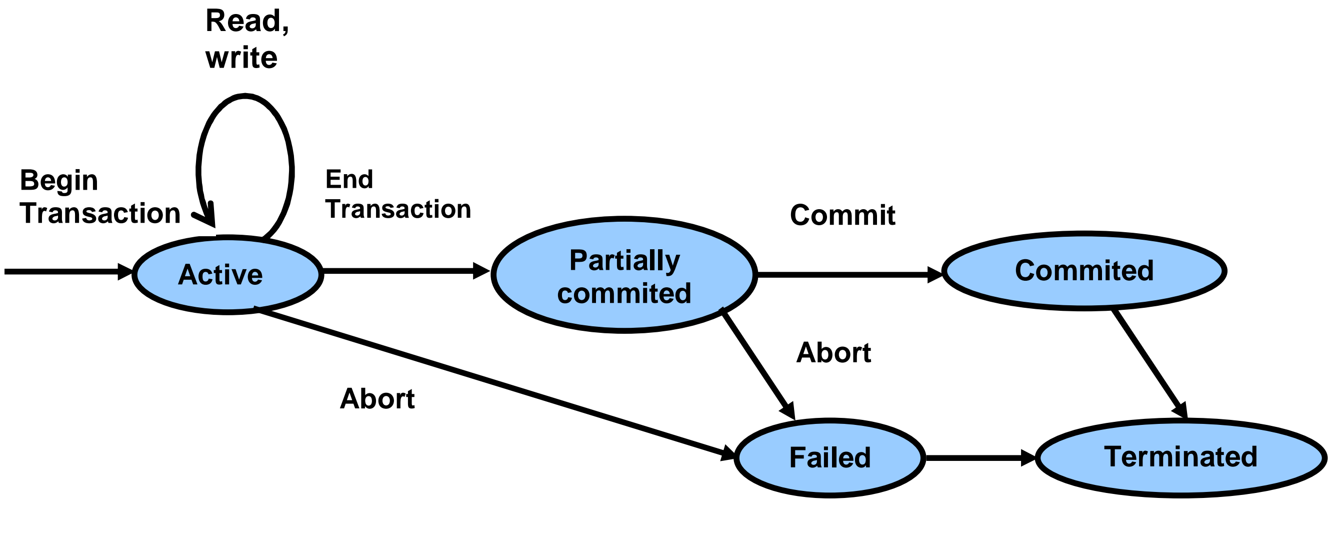
Read / write head malfunctioning.

Transaction failure

1. Operations of transaction may fail due to logical errors. E.g. Division by 0. Integer over flow. Error in parameters. Error in control structures.
2. Transactions may fail due to concurrency.
3. Transaction may fail due to S/W, H/W errors during the execution of transaction.
4. Transaction may fail if the data for the transaction *may not be found* or *insufficient.*

Topic4: Transaction states

Begin_transaction	This makes begin the transaction.
Read / Write	Specifies either *read operation* or *write operation.*
End_transaction	Specifies all the read & write operations are commited (or) Specifies all the read & write operations are partially commited and aborted.
Commit	Indicates successful updation of transaction.
Rollback / Abort	Indicates that the transaction ended unsuccessfully.



Topic5: The system log [IMP]

- **System log: System log is file keep track of all transaction operations on database items.**

- **Advantages**

1.	The log file entries used for recovery purpose.
2.	The log file is periodically backed up into magnetic tape (Archival storage)
3.	Log record: Entry of log file.

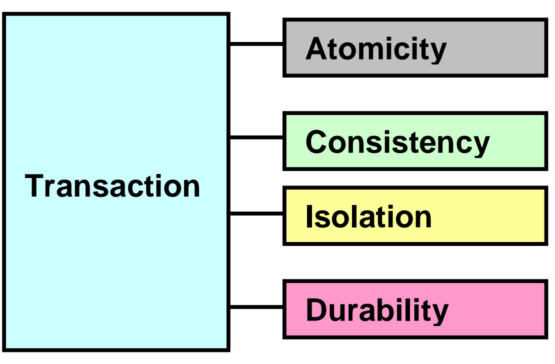
- **Entries of log file**

[start_transaction,T]	Indicates that transaction T has started execution
[write_item,T,X,old_value,new_value]	Indicates that transaction T has changed the value of database item X from old value to new value.
[read_item,T,X]	Indicates the transaction T has read the value of database item X.
[commit,T]	Indicates that transaction T has completed successfully and effect of transaction can be recorded permanently in the database.
[abort,T]	Indicates that transaction T has been aborted.

Topic 6: Desirable properties of transaction (or) ACID properties of transaction [IMP]

Atomicity

- The transaction is atomic unit of processing i.e. either performed entirely or none performed at all.



Consistency

- A correct execution of transaction must take the database item from one consistent state to another consistent state.

Isolation

- A transaction is isolated from other transactions when multiple transactions are accessing the same data item concurrently in a *serial* or *interleaving* fashion.

Durability

- The changes applied to the database item by a commited transaction must persistent in the database.

Topic 7: Schedules & Recoverability

- Schedule (or) History: The order of execution of operations of *serial* or *interleaving* transactions.

T1	T2
read_item(x) x:=x-n;	
	read_item(x); x:=x+n;

write_item(x);
read_item(y);
y:=y+n;	

	write_item(x);
write_item(y);	

S_A: r1(x); r2(x); w1(x); r1(y); w2(x); w1(y);

T1	T2

read_item(x)
x:=x-n;
write_item(x);	

	read_item(x);
	x:=x+m;
	write_item(x);

read_item(y); Fails	

S_B: $r_1(x)$; $w_1(x)$; $r_2(x)$; $w_2(x)$; $r_2(y)$; a1;

Topic8: Conflict operations

- Two operations in a schedule are said to be conflict, if they satisfy all three of the following conditions.

1. They belongs to different transactions.
2. They access the same data item.
3. At least one of the operation is a write operation.

- Consider schedule S_A

  S_A: $r_1(x)$; $r_2(x)$; $w_1(x)$; $r_1(y)$; $w_2(x)$; $w_1(y)$;

- Consider two operations from schedule

  $r_1(x)$ & $w_2(x)$;

  The above two operations are conflict since

The operations belongs to different transactions T1 &T2.
They access the same data item x.
One of the operation is write operation.

- Verify which of the operations are conflict

Operations	Result
$r_2(x)$ & $w_1(x)$;	Conflict
$w_1(x)$ & $w_2(x)$;	Conflict
$r_1(x)$ & $r_2(x)$;	Not conflict
$w_2(x)$ & $w_1(y)$;	Not conflict

Topic 8: Complete schedule

- A sehedule **'s'** of **'n'** transactions $T_1, T_{2,\dots,}T_n$, is said to be a complete schedule if the following conditions hold.

1.The operations in **'s'** are exactly those operations in $T_1, T_{2,\dots,}T_n$, including a **commit** or **abort** operation as the last operation for each transaction in the schedule.
2. For any pair of operations from the same transaction T_i, their order of appearance is **'s'** is the same as their order of appearance in T_i
3. For any two conflicting operations, one of the two must occur before the order in the schedule.

Topic 9: Serial schedule

- **Serial schedule: The order of execution of operations of serial transactions.**

T1	T2

read_item(x) ;
x:=x – n;
write_item(x);
read_item(y);
y:=y + n;
write_item(y);	

	read_item(x);
	x:=x + m;
	write_item(x);

S_A: $r_1(x)$; $w_1(x)$; $r_1(y)$; $w_1(y)$; $r_2(x)$; $w_2(x)$;

T1	T2

	read_item(x) ;
	x:=x +m;
	write_item(x);

read_item(x);
x:=x - n;
write_item(x);
read_item(y);
y:=y + n;
write_item(y);	

S_B: $r_2(x)$; $w_2(x)$; $r_1(x)$; $w_1(x)$; $r_1(y)$; $w_1(y)$;

Topic 10: Non serial schedule

- **Non serial schedule: The order of execution of operations of interleaving transactions.**

T1	T2
read_item(x) ; x:=x + n;	
	read_item(x); x:= x + m;
write_item(x); read_item(y);	
	write_item(x);
x:=y + n; write_item(y);	

S_c: $r_1(x)$; $r_2(x)$; $w_1(x)$; $r_1(y)$; $w_2(x)$; $w_1(y)$;

T1	T2

read_item(x) ;
x:=x - n;
write_item(x);	

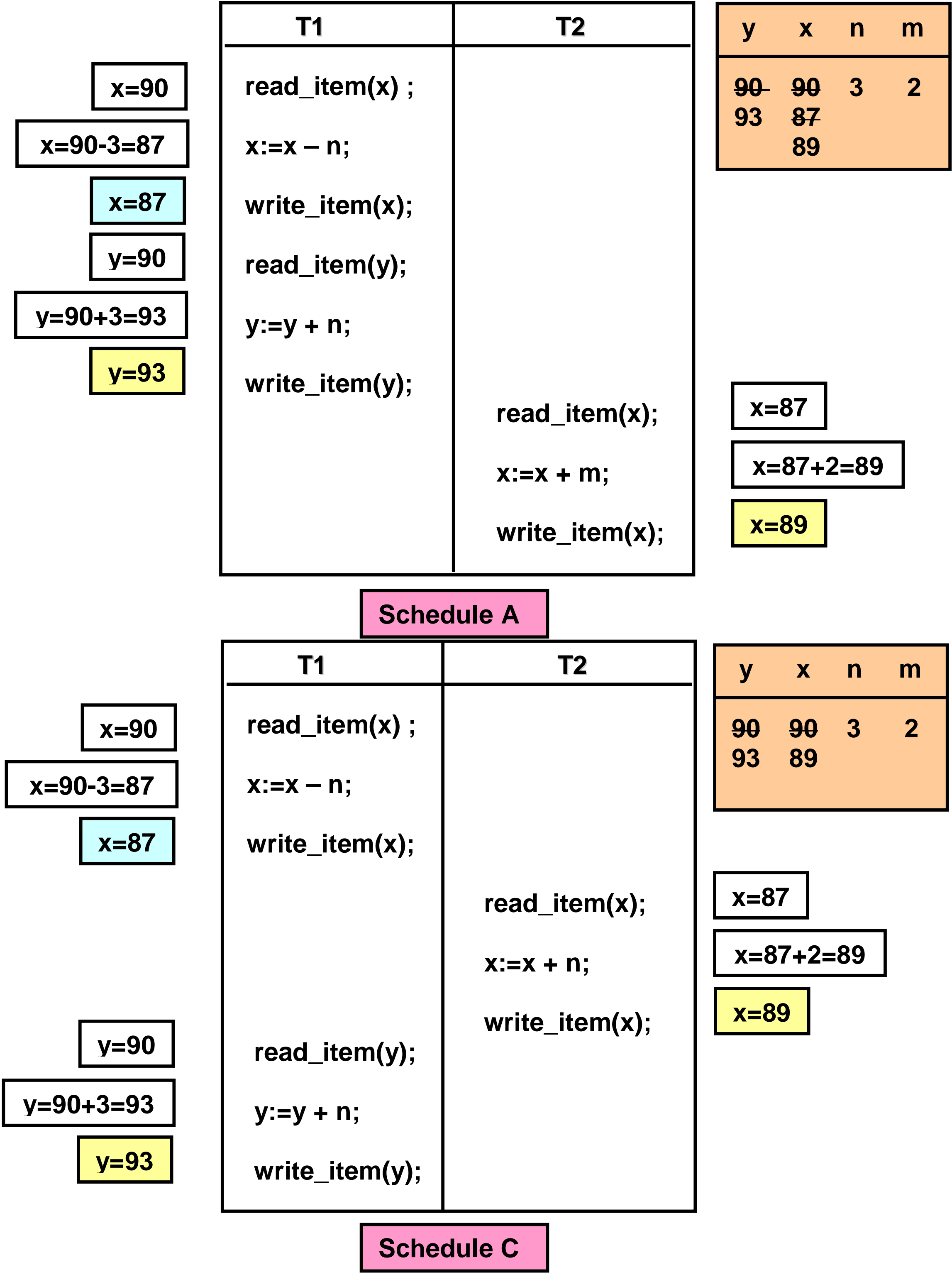
	read_item(x);
	x:=x + m;
	write_item(x);

read_item(y);
y:=y + n;
write_item(y);	

S_D: $r_1(x)$; $w_1(x)$; $r_2(x)$; $w_2(x)$; $r_1(y)$; $w_1(y)$;

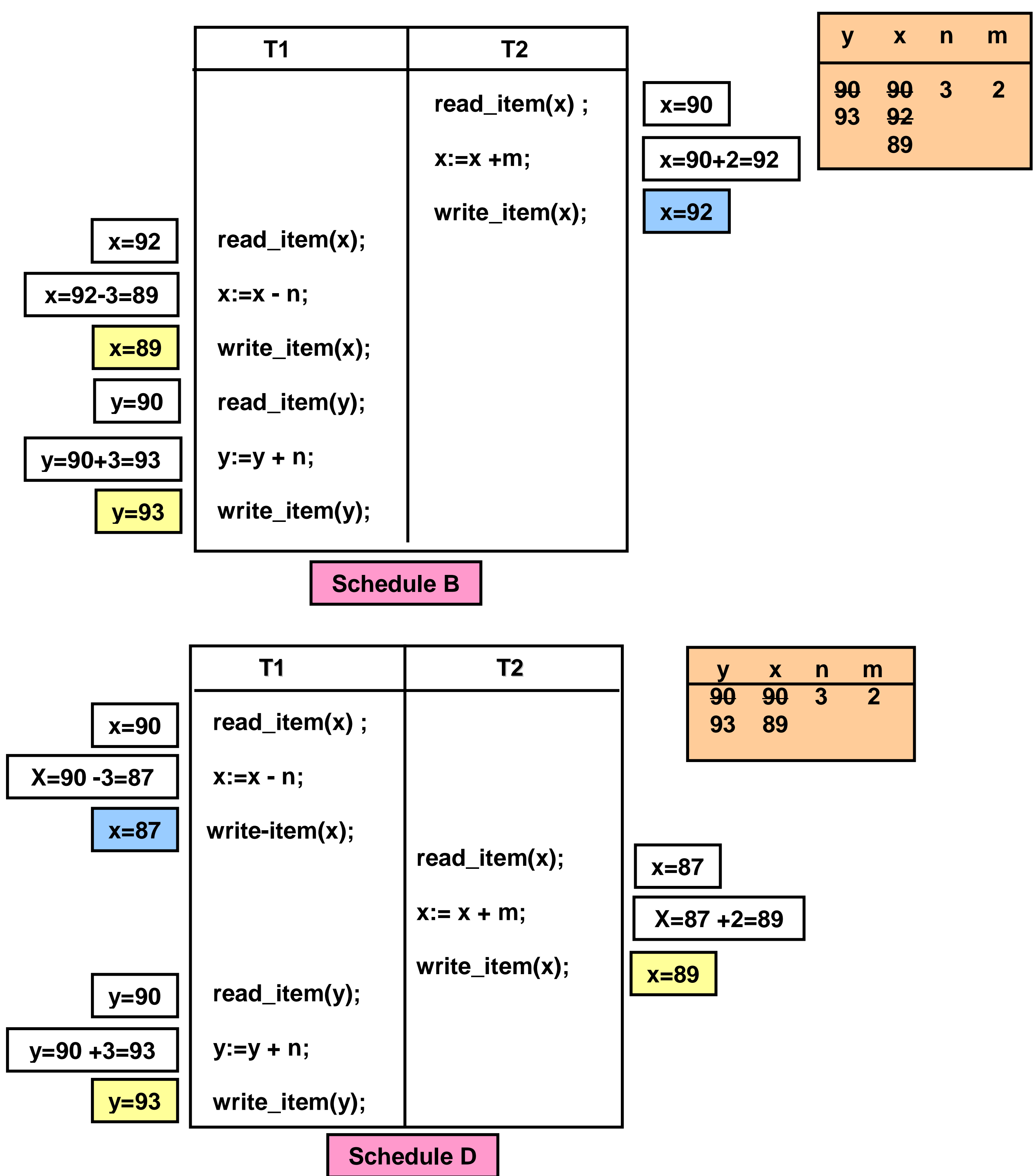
Topic 11: Result equivalent schedule

- Two schedules are result equivalent if they produce the same final states of the database item.



Topic 12 : Serializable schedule | VIMP

- Serializability: A schedule **'s'** of **'n'** transactions is serializable if it equal to some serial schedule of same **'n'** transactions.



- A non serial schedule , *schedule D* is serializable to serial schedule *schedule B ,* since they are equivalent.

Topic14: Conflict Equivalent Schedule [IMP]

- Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both the schedules.

- Two operations in a schedule are said to be conflict, if they satisfy all three of the following conditions.

1. They belongs to different transactions.
2. They access the same data item.
3. At least one of the operation is a write operation.

E.g1: S_1: $r_1(x)$; $w_2(x)$
S_2: $w_2(x)$; $r_1(x)$

Both S_1 and S_2 contains conflict operations but they are NOT CONFLICT equivalent.

E.g2: S_1: $w_1(x)$; $w_2(x)$
S_2: $w_2(x)$; $w_1(x)$

Both S_1 and S_2 contains conflict operations but they are NOT CONFLICT equivalent.

E.g3: S_A: $w_1(x)$; $r_2(x)$
S_B: $w_1(x)$; $r_2(x)$

Both S_A and S_B contains conflict operations but they are CONFLICT equivalent.

Topic 15: Uses Of Serializability

1. A Serializable schedule gives the benefits of concurrent execution

2. The interleaving of operations from concurrent transactions is typically determined by OS SCHEDULER which allocates resources to all the processes

3. Enforces TWO PHASE LOCKING PROTOCOL (Concurrency control protocol)

4. Ensures TIME STAMP ORDERING PROTOL (Concurrency control protocol)

5. Ensures MULTIVERSION PROTOCOL (Maintains multiple versions of database item)

6. Ensures OPTIMISTIC (or) CERTIFICATION (or) VALIDATION protocol (Concurrency control protocols)

Topic16: Conflict Serializable Schedule [IMP]

Conflict Serializable: A schedule S is conflict serializable if it is conflict equivalent to some serial schedule S^I

- **Consider the schedules S_A and S_D**

S_A: $r_1(x)$; $\underline{w_1(x)}$; $r_1(y)$; $w_1(y)$; $\underline{r_2(x)}$; $w_2(x)$;

S_D: $r_1(x)$; $\underline{w_1(x); \ r_2(x)}$; $w_2(x)$; $r_1(y)$; $w_1(y)$;

- **Consider**

$w_1(x)$;$r_2(x)$ in S_A
$w_1(x)$;$r_2(x)$ in S_D

1. **In both the schedules Transaction 2 reads the values after Transaction 1 writes**
2. **The operations $w_1(x)$;$r_2(x)$ is conflicting operations**
3. **Last operations are write operation in both the schedules**

- **Therefore S_D is conflict serializable to S_A**

Topic 17: View Equivalence [IMP]

Consider two schedules S and S_g

S: $r_1(x)$; $w_1(x)$; $r_1(y)$; $w_2(x)$; $w_3(x)$;

S_g: $r_1(x)$; $w_2(x)$; $w_1(x)$; $w_3(x)$; c_1; c_2; c_3;

- **Two schedules S and S_g are said to be view equivalent**

1. **Bothe schedules have same transactions and same operations**

S	S_g
T1,T2,T3	T1,T2,T3
read_item(x) write_item(x)	read_item(x) write_item(x)

2. The first operation of first schedule should be same as the first operation of second schedule

3. The last operation of first schedule should be same as the last operation of second schedule

4. Both the schedules enforce conflict equivalence

Topic 18: View Serializability [IMP]

- A Schedule S_g is view serializable if it is view equivalent to some serial schedule S

S: $r_1(x)$; $w_1(x)$; $r_1(y)$; $w_2(x)$; $w_3(x)$;

S_g: $r_1(x)$; $w_2(x)$; $w_1(x)$; $w_3(x)$; c_1; c_2; c_3;

Non serial schedule S_g is view serializable to serial schedule S

- CONSTRAIND WRITE ASSUMPTION :

Any write operation $w_i(x)$ in T_i is preceded by a read operation $r_i(x)$.

S_g: $r_1(x)$; $w_2(x)$; $w_1(x)$; $w_3(x)$; c_1; c_2; c_3;

Note : Read value for data item before writing it.

Note : $w_1(x)$ preceded by the $r_1(x)$

Note: In the above schedule $w_1(x)$ is preceded by $r_1(x)$

- UN CONSTRAIND WRITE ASSUMPTION (OR) BLIND WRITES

The read operation $r_i(x)$ is not preceded by write operation $w_i(x)$

Note: Data item value can't be read before writing it

Note: $w_2(x)$ and $w_3(x)$ is not preceded by the read operations $r_2(x)$ and $r_3(x)$

Topic 19: Transaction Support In SQL	IMP

- Transaction: Sequence of statements that performs specified task.
- A single SQL statement always considered to be atomic.

- Characteristics :

(a) Access Mode

Access Mode specified as READ ONLY (or) READ WRITE

READ ONLY: Read the data from database (SELECT)
READ WRITE: INSERT, UPDATE, DELETE

(b) Diagnostics Area Size

DIAGNOSTIC SIZE N

The integer value N indicating the number of error conditions or exceptions.

These conditions supply feedback information to the users on the most recently executed SQL statements.

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;

EXEC SQL SET TRANSACTION
        READ WRITE
        DIAGNOSTICS SIZE 5
        ISOLATION LEVEL SERIALIZABLE;

EXEC SQL INSERT INTO EMPLOYEE(Fname,Lname,SSN,Dno,Salary)
VALUES('Robert','Smith','991004321',2,35000);

EXEC SQL UPDATE EMPLOYEE
    SET Salary = Salary * 1.1 WHERE Dno = 2;

EXEC SQL COMMIT;

UNDO: EXEC SQL ROLLBACK;
THE_END: …;
```

(c) ISOLATION LEVEL

Isolation level specified with option

ISOLATION LEVEL < ISOLATION>

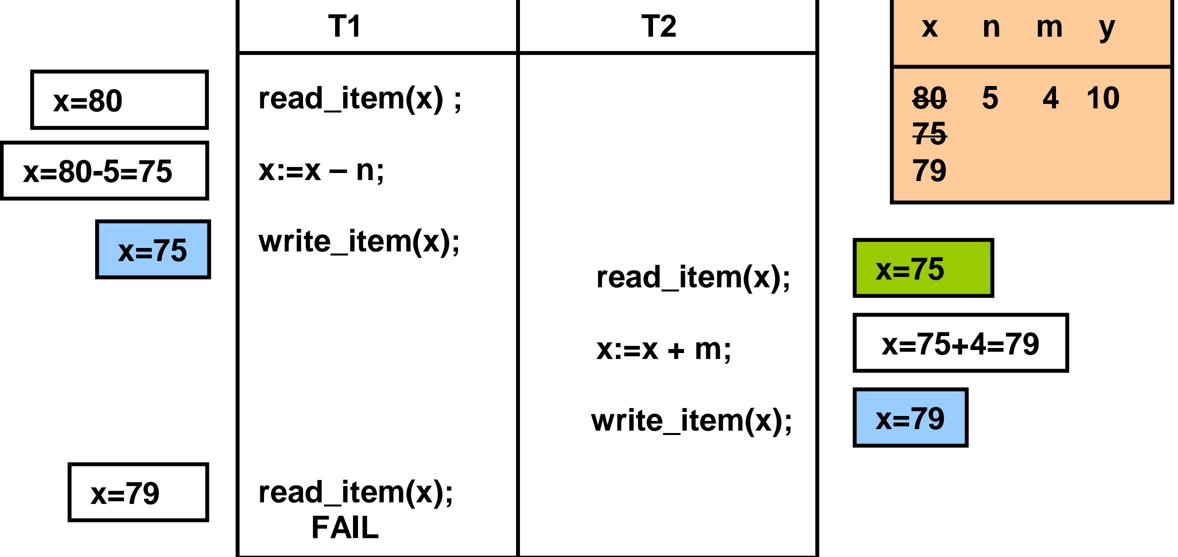
ISOLATION:

1. **READ UNCOMMITED**
2. **READ COMMITED**
3. **REPEATABLE READ**
4. **SERIALIZABLE**

There may be three of the following violations in SQL TRANSACTIONS

1. Dirty read :

- **The problem occurs when one transaction update the database item and then fails**



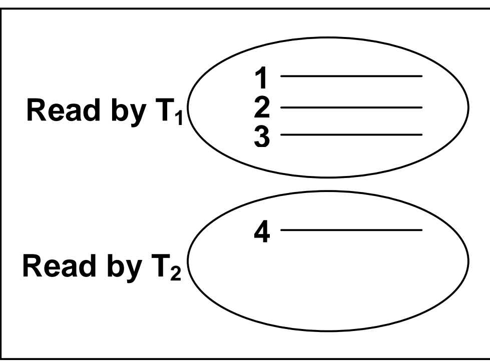
- **If transaction T1 fails the database item roll backs to its previous state, but transaction T2 has read the incorrect value of x.**

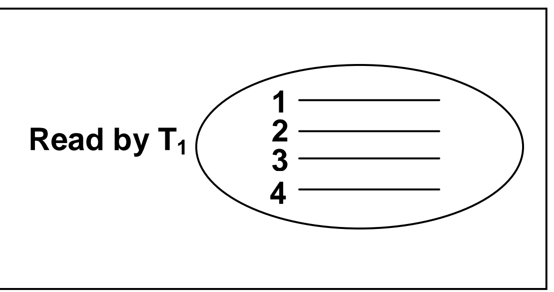
2. Nonrepeatable read

A transaction T_1 may read a given value from a table. If another transaction T_2 later updates that value and T_1 reads that value again, T_1 will see a different values.

3. Phontoms

1. A transaction T_1 may read the rows from table based on the where condition.
2. A transaction T_2 inserts a new record that satisfy where condition used in T_1.
3. If T_1 is repeated, then T_1 will see a PHONTOM, a row that previously did not exist.

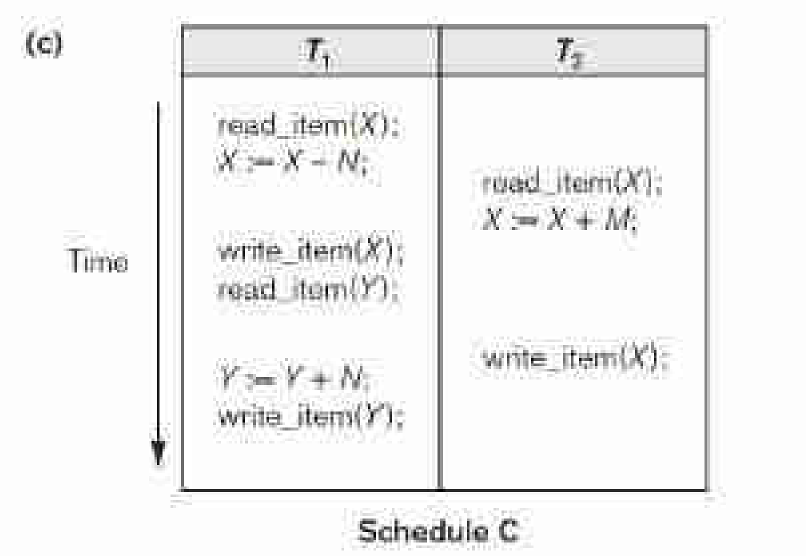




Possible violations Based on isolation levels as defined in SQL			
	TYPE OF VIOLATION		
Isolation Level	Dirty Read	Non repeatable Read	Phantom
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

Testing Serializability (or) Confilct Serializability

Problem 1: Verify whether the following Transactions ensures serializablity.



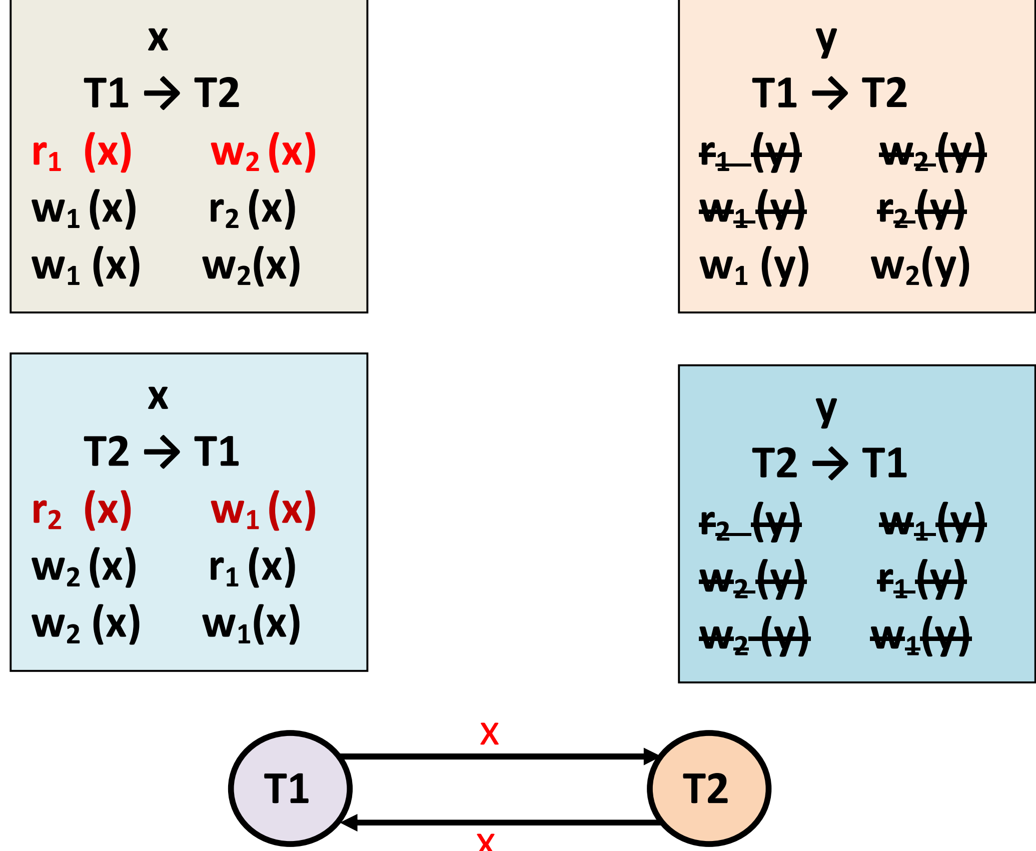
(c)

T_1	T_2
read_item(X);	
X := X - N;	
	read_item(X);
	X := X + M;
write_item(X);	
read_item(Y);	
	write_item(X);
Y := Y + N;	
write_item(Y);	

Time

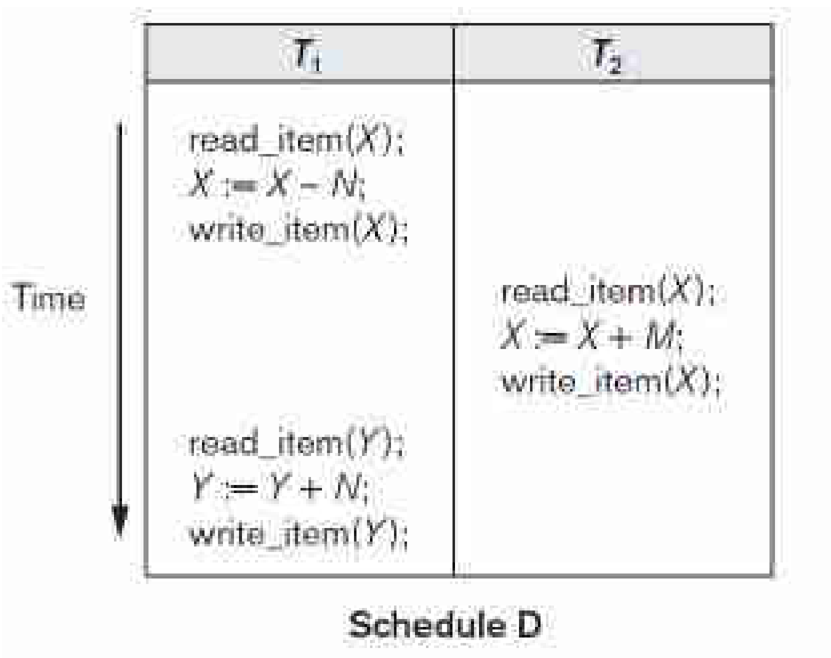
Schedule C

$$S_c: r_1(x)\ ;\ r_2(x);\ w_1(x);\ r_1(y);\ w_2(x);\ w_1(y);$$

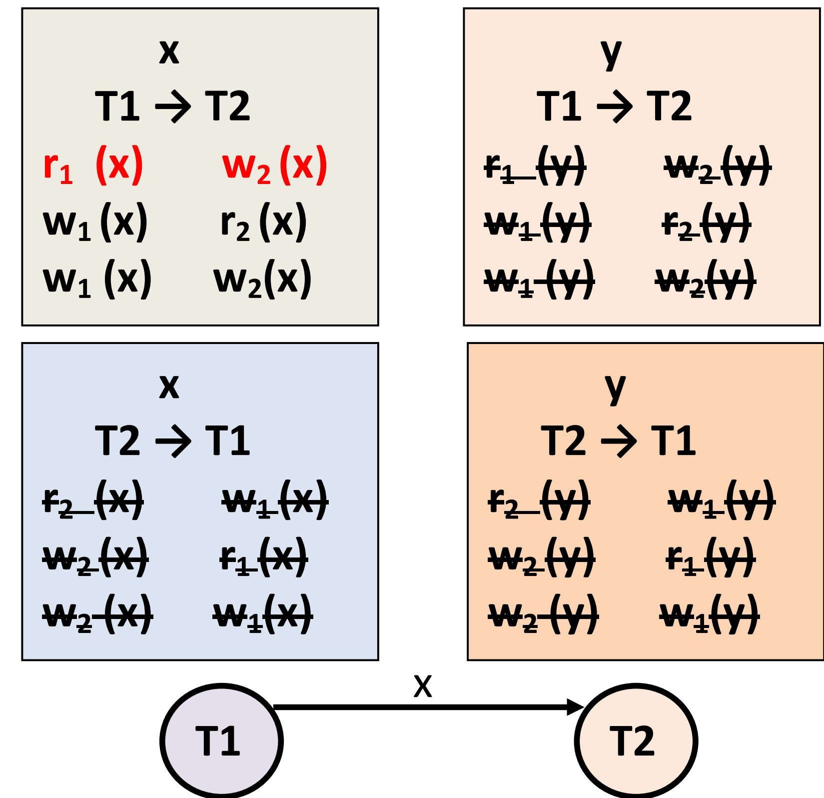


- **Since the Precedence Graph has Cycle, the schedule Sc is not *Serializable* or *Conflict Serializable.***

Problem 2: Verify whether the following Transactions ensures serializablity.



S_D: $r_1(x)$; $w_1(x)$; $r_2(x)$; $w_2(x)$; $r_1(y)$; $w_1(y)$;



- Since the **Precedence Graph** has **No Cycle**, the schedule S_D is ***Serializable*** or ***Conflict Serializable.***

Problem 3: Verify whether the following schedule enforces conflict serializability.

Time	Transaction T_1	Transaction T_2	Transaction T_3
↓		read_item(Z);	
		read_item(Y);	
		write_item(Y);	
			read_item(Y);
			read_item(Z);
	read_item(X);		
	write_item(X);		
			write_item(Y);
			write_item(Z);
		read_item(X);	
	read_item(Y);		
	write_item(Y);		
		write_item(X);	

Schedule E

S_E: $r_2(z)$; $r_2(y)$; $w_2(y)$; $r_3(y)$; $r_3(z)$; $r_1(x)$; $w_1(x)$; $w_3(y)$; $w_3(z)$; $r_2(x)$; $r_1(y)$; $w_1(y)$; $w_2(x)$;

x T1 → T2		y T1 → T2		z T1 → T2	
$r_1(x)$	$w_2(x)$	~~$r_1(y)$~~	~~$w_2(y)$~~	~~$r_1(z)$~~	~~$w_2(z)$~~
$w_1(x)$	$r_2(x)$	~~$w_1(y)$~~	~~$r_2(y)$~~	~~$w_1(z)$~~	~~$r_2(z)$~~
$w_1(x)$	$w_2(x)$	~~$w_1(y)$~~	~~$w_2(y)$~~	~~$w_1(z)$~~	~~$w_2(z)$~~

x T1 → T3		y T1 → T3		z T1 → T3	
~~$r_1(x)$~~	~~$w_3(x)$~~	~~$r_1(y)$~~	~~$w_3(y)$~~	~~$r_1(z)$~~	~~$w_3(z)$~~
~~$w_1(x)$~~	~~$r_3(x)$~~	~~$w_1(y)$~~	~~$r_3(y)$~~	~~$w_1(z)$~~	~~$r_3(z)$~~
~~$w_1(x)$~~	~~$w_3(x)$~~	~~$w_1(y)$~~	~~$w_3(y)$~~	~~$w_1(z)$~~	~~$w_3(z)$~~

x

T3 → T1

~~r_3 (x)~~ ~~w_1 (x)~~

~~w_3 (x)~~ ~~r_1 (x)~~

~~w_3 (x)~~ ~~w_1(x)~~

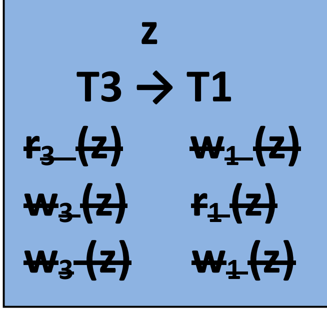
y

T3 → T1

r_3 (y) w_1 (y)

w_3 (y) r_1 (y)

w_3 (y) w_1(y)



x

T3 → T2

~~r_3 (x)~~ ~~w_2 (x)~~

~~w_3 (x)~~ ~~r_2 (x)~~

~~w_3 (x)~~ ~~w_2(x)~~

y

T3 → T2

~~r_3 (y)~~ ~~w_2 (y)~~

~~w_3 (y)~~ ~~r_2 (y)~~

~~w_3 (y)~~ ~~w_2(y)~~

z

T3 → T2

~~r_3 (z)~~ ~~w_2 (z)~~

~~w_3 (z)~~ ~~r_2 (z)~~

~~w_3 (z)~~ ~~w_2 (z)~~

x

T2 → T1

~~r_2 (x)~~ ~~w_1 (x)~~

~~w_2 (x)~~ ~~r_1 (x)~~

~~w_2 (x)~~ ~~w_1(x)~~

y

T2 → T1

r_2 (y) w_1 (y)

w_2 (y) r_1 (y)

w_2 (y) w_1(y)

z

T2 → T1

~~r_2 (z)~~ ~~w_1 (z)~~

~~w_2 (z)~~ ~~r_1 (z)~~

~~w_2 (z)~~ ~~w_1 (z)~~

x

T2 → T3

~~r_2 (x)~~ ~~w_3 (x)~~

~~w_2 (x)~~ ~~r_3 (x)~~

~~w_2 (x)~~ ~~w_3(x)~~

y

T2 → T3

r_2 (y) w_3 (y)

w_2 (y) r_3 (y)

w_2 (y) w_3(y)

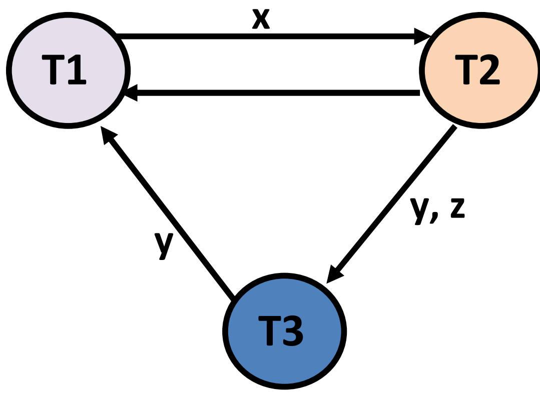
z

T2 → T3

r_2 (z) w_3 (z)

w_2 (z) r_3 (z)

w_2 (z) w_3 (z)



- **Since the precedence has cycles , the schedule S_E is not conflict serializable or serializable.**

CONCURRENCY CONTROL

- Concurrency: A problem situation as a single data item is required by the multiple transactions simultaneously.

Topic 1: Two Phase Locking Techniques for Concurrency Control

- Lock: A lock is a variable that prescribe the state of database items.

1.1 Binary Locks

- A binary lock can have two states or values locked and unlocked (or) 1 and 0.
- ***States:***

1. Lock(x)=0 indicates that the database item is available.
2. Lock(x)=1 indicates that the database item is not available.

- ***Operations:***

1. Lock_item(x);
2. Unlock_item(x):

Note: x is a database item.

- ***Algorithm:***

```
lock_item(X):
B:  if LOCK(X) = 0                    (* item is unlocked *)
        then LOCK(X) ←1               (* lock the item *)
    else
        begin
        wait (until LOCK(X) = 0
            and the lock manager wakes up the transaction);
        go to B
        end;
unlock_item(X):
    LOCK(X) ← 0;                      (* unlock the item *)
    if any transactions are waiting
        then wakeup one of the waiting transactions;
```

- ***Rules for Binary Locking:***

T_1

read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
X := X + Y;
write_item(X);
unlock(X);

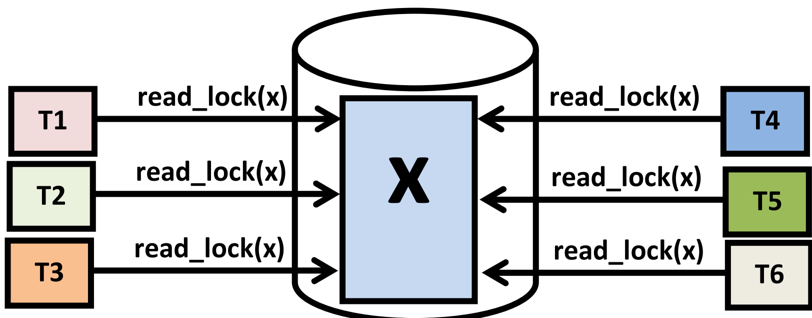
1. The read/write operation is preceded by lock operation.
2. A transaction must be unlocked if it is locked.
3. A transaction must not issue lock operation once it is locked.
4. A transaction will not issue unlock if it is not locked.

1.2 Shared/Exclusive Lock (or) Read/Write Lock

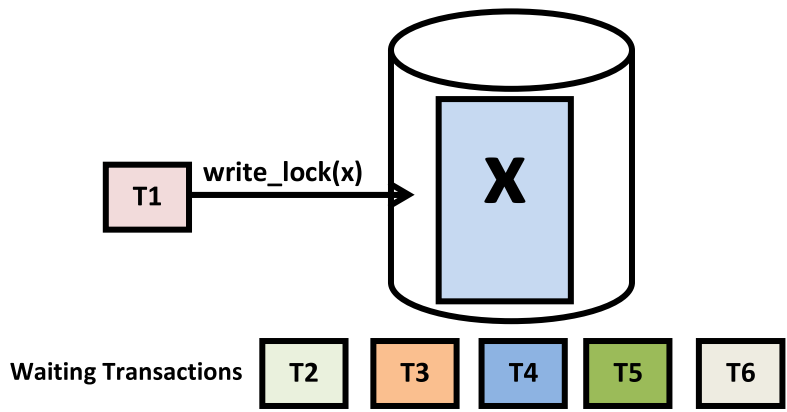
- Multiple transactions can simultaneously read the data but single transaction exclusively updates the data while remaining transactions are waiting.

- ***Operations:***

1. read_lock(x): Multiple transactions are allowed simultaneously to read the data base item.



2. write_lock(x): A single transaction exclusively holds lock on the database item.



3. unlock(x): Release of lock on data base item after completion of transaction.

- ***Algorithm:***

```
read_lock(X):
B:  if LOCK(X) = "unlocked"
        then begin LOCK(X) ← "read-locked";
                no_of_reads(X) ← 1
                end
    else if LOCK(X) = "read-locked"
        then no_of_reads(X) ← no_of_reads(X) + 1
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
```

```
write_lock(X):
B:  if LOCK(X) = "unlocked"
        then LOCK(X) ← "write-locked"
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
```

```
unlock (X):
    if LOCK(X) = "write-locked"
        then begin LOCK(X) ← "unlocked";
                wakeup one of the waiting transactions, if any
                end
    else if LOCK(X) = "read-locked"
        then begin
                no_of_reads(X) ← no_of_reads(X) -1;
                if no_of_reads(X) = 0
                    then begin LOCK(X) = "unlocked";
                            wakeup one of the waiting transactions, if any
                            end
                end;
```

- ***Rules for Shared/Exclusive Locking***

T_1

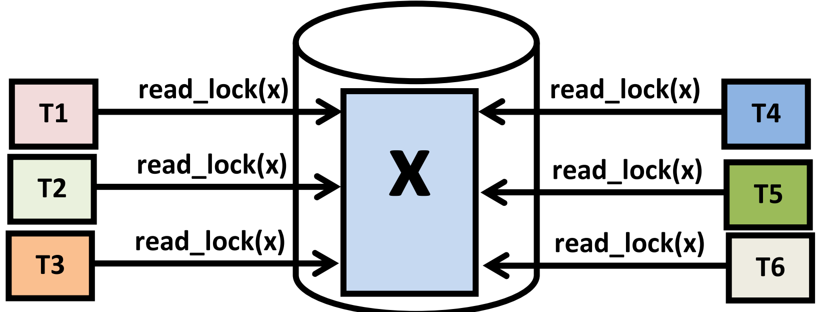
read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
X := X + Y;
write_item(X);
unlock(X);

1. The read/write operation is preceded by lock operation.
2. A transaction must issue **write lock** before **write operation**.
3. A transaction must be unlocked if it is locked.
4. A transaction must not issue lock operation once it is locked.
5. A transaction will not issue unlock if it is not locked.

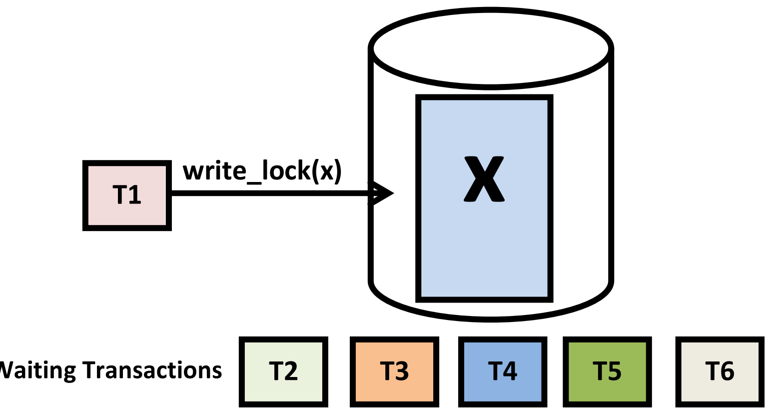
Topic 2: Guaranteeing Serializability by Two-Phase Locking

- Objective: Concurrency Control.

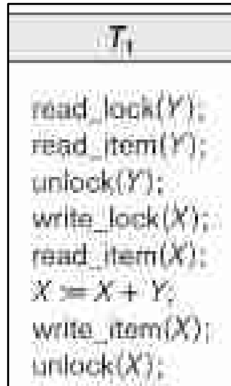
- read_lock(x): Multiple transactions are allowed simultaneously to read the data base item.



- write_lock(x): A single transaction exclusively holds lock on the database item.



- ***Rule:***



T_1

read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
X := X + Y;
write_item(X);
unlock(X);

- All the locking operations are preceded by the un lock operation.

- **Growing Phase:**

1. A new lock on data base item can be acquired but none can be released.
2. Up gradation of lock is possible (Changing *read lock* to *write lock*)

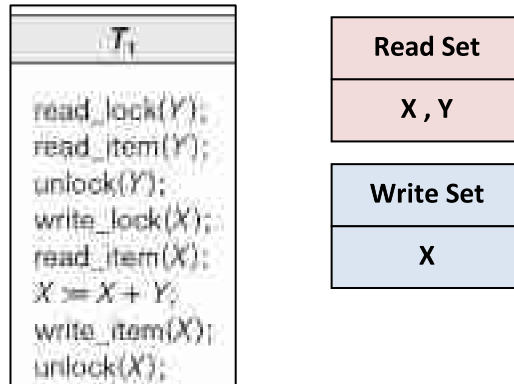
- **Shrinking Phase:**

3. Existing locks can be released but no new locks can be acquired.
4. Down gradation of lock is possible (Changing *write lock* to *read lock*)

2.1 Static 2PL (or) Conservative 2PL:

- All the locking operations are preceded by the un lock operation.
- *Read Set* & *Write Set* for the transactions are formulated.
- *Read Set*: Set of all the data items that the transaction reads.
- *Write Set*: Set of all the data items that the transaction writes.

Example: Find *Read Set* & *Write Set* for the following Transaction.



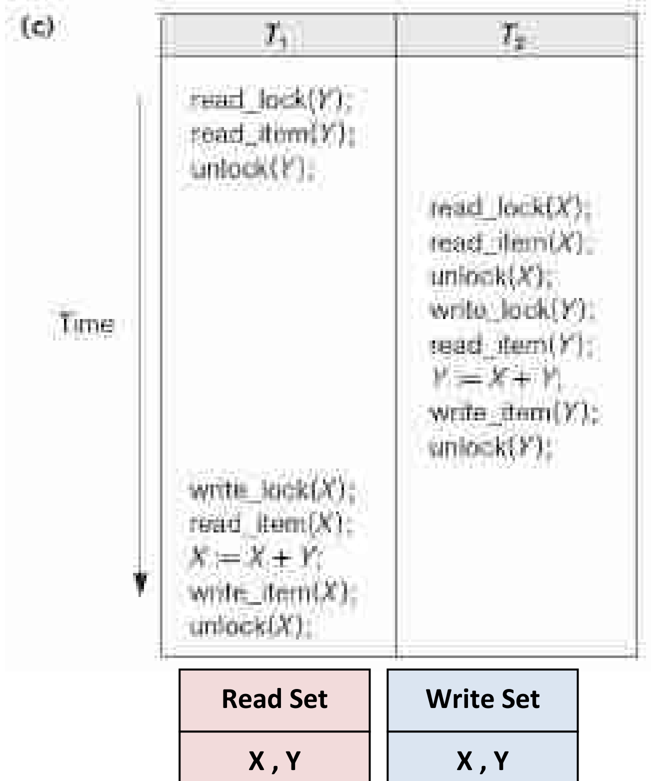
T_1
read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);

Read Set
X , Y

Write Set
X

PROBLEMS

Problem 1: Find *Read Set* & *Write Set* for the following Transactions.



(c)

Time	T_1	T_2

↓	read_lock(Y);
	read_item(Y);
	unlock(Y);	

		read_lock(X);
		read_item(X);
		unlock(X);
		write_lock(Y);
		read_item(Y);
		$Y := X + Y$;
		write_item(Y);
		unlock(Y);

	write_lock(X);
	read_item(X);
	$X := X + Y$;
	write_item(X);
	unlock(X);	

Read Set	Write Set
X , Y	X , Y

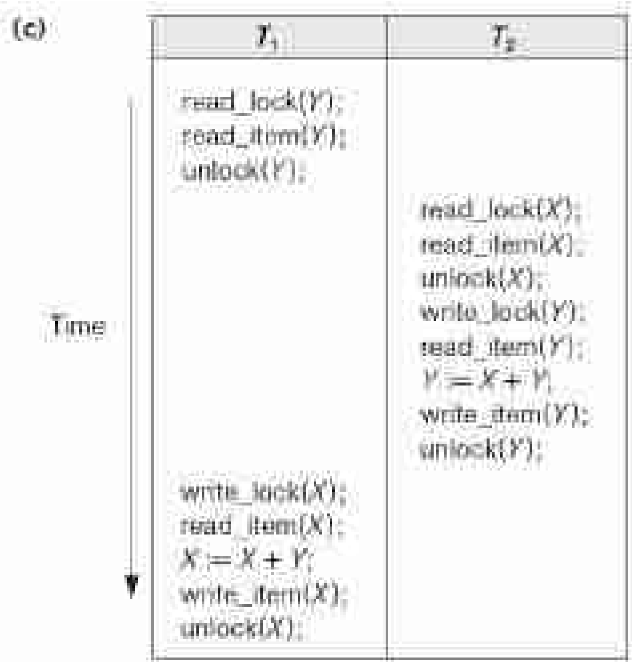
Problem 2: Verify whether the following transaction ensures 2PL.

T_2

read_lock(X);
read_item(X);
unlock(X);
write_lock(Y);
read_item(Y);
$Y := X + Y$;
write_item(Y);
unlock(Y);

Result: The Transaction T_2 ensure 2PL since all locking operations precedes un lock operations.

Problem 3: Verify whether the following transactions ensures 2PL.



Result: The Transaction T_2 and T_2 ensure 2PL since all locking operations precedes un lock operations.

CONCURRENCY CONTROL BASED ON TIME STAMP ORDERING

(i) Time Stamp Ordering Algorithm (TO): The Timestap ordering protocol is a protocol used to sequence the transactions based on their Timestamps.

- Each transaction must have time stamp when it gets started.
- Time Stamp: The unique identification number assigned for transaction.
- The time stamp of T is also called as TS (T).

Operations:

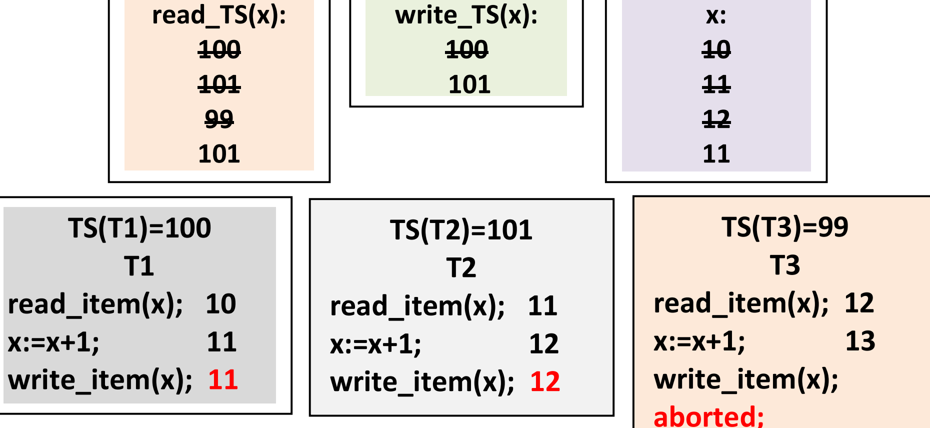
- read_TS(X): The read timestamp of item X is the largest timestamp among all the timestamps of transactions that have successfully read item X.
- write_TS(X): The write timestamp of item X is the largest of all the timestamps of transactions that have successfully written item X.

read_TS(x):	write_TS(x):	x:
~~100~~	~~100~~	~~10~~
~~101~~	~~101~~	~~11~~
102	102	~~12~~

TS(T1)=100		TS(T2)=101		TS(T3)=102	
T1		T2		T3	
read_item(x);	10	read_item(x);	11	read_item(x);	12
x:=x+1;	11	x:=x+1;	12	x:=x+1;	13
write_item(x);	11	write_item(x);	12	write_item(x);	13

(ii) Basic Time Stamp Ordering Algorithm: The order of transaction is nothing but the ascending order of the transaction creation.

- Every transaction is issued a timestamp based on when it enters the system. Suppose, if an old transaction T_i has timestamp $TS(T_i)$, a new transaction T_j is assigned timestamp $TS(T_j)$ such that $TS(T_i) < TS(T_j)$.



Condition 1:

- **Whenever a transaction *T* issues a write_item(X) operation, the following is checked:**

  99 > 101   101 > 99

  a. **If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then abort and roll back *T* and reject the operation.**

  b. **If the condition in part (a) does not occur, then execute the write_item(X) operation of *T* and set write_TS(X) to TS(T).**

Condition 2:

Whenever a transaction T issues a read_item(X) operation, the following is checked:

101 > 99

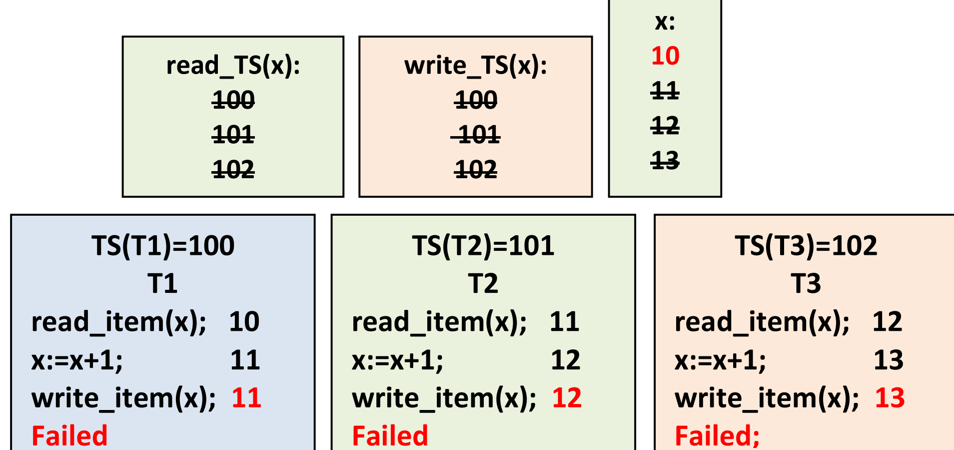
a. If write_TS(X) > TS(T), then abort and roll back T and reject the operation.

101 <= 99

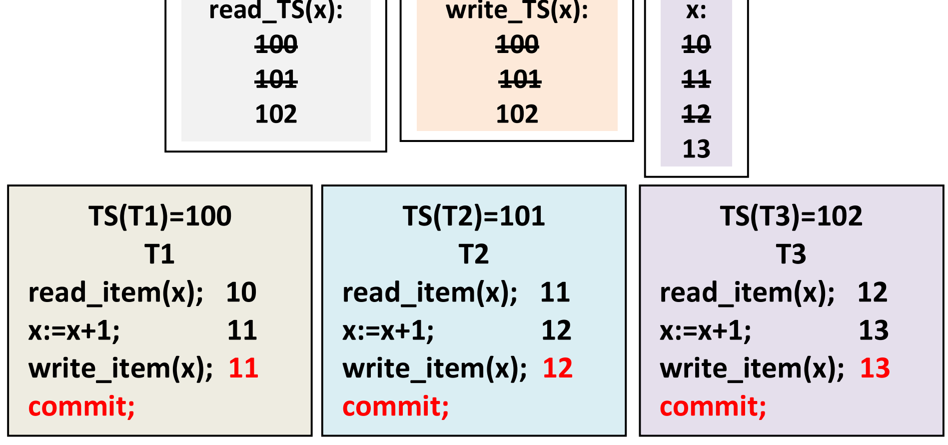
b. If write_TS(X) <= TS(T), then execute the read_item(X) operation of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

Condition 3:

Cascading Rollback: If transaction T is aborted and rolled back, any transaction T_i that may have used a value written by T must also be rolled back.



(iii) Strict Time Stamp Ordering Algorithm (TO): Every transaction must commit when it exits.



102 > 101

- **A transaction T that issues a read_item(X) or write_item(X) such that TS(T) > write_TS(X) has its read or write operation *delayed* until the transaction T that *wrote* the value of X has committed or aborted.**

- **Thomas's Write Rule: This is modification of the basic TO algorithm. When the current transaction (TS(T)) fails then continue the read operation and abort write operation.**

  102 > 102
  1. **If read_TS(*X*) > TS(T), then abort and roll back T and reject the operation.**

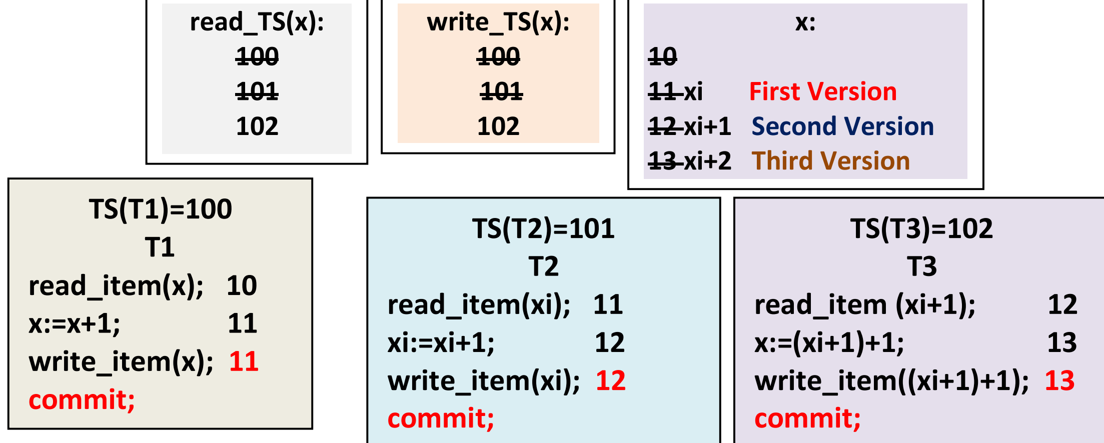
  102 > 102
  2. **If write_TS(*X*) > TS(T), then do not execute the write operation but continue processing.**

read_TS(x):	write_TS(x):	x:
~~100~~	~~100~~	~~10~~
~~101~~	~~101~~	~~11~~
102	~~102~~	12
	101	

TS(T1)=100		TS(T2)=101		TS(T3)=102	
T1		T2		T3	
read_item(x);	10	read_item(x);	11	read_item(x);	12
x:=x+1;	11	x:=x+1;	12	x:=x+1;	13
write_item(x);	11	write_item(x);	12	write_item(x);	13
commit;		commit;		Failed;	

1.4 Multiversion Concurrency Control Techniques

- **Version: State of database item after successful updation.**



- **Temporal Database: Database keeps track of multiple versions of database items.**
  E.g. xi, xi+1, xi+2, xi+3 etc.

- **Advantage: Multiple versions of database items are maintained so that concurrency is avoided to certain extent.**

- **Operations:**

1. read_TS(Xi): The read timestamp of Xi is the largest of all the timestamps of transactions that have successfully read version Xi.

2. write_TS(Xi): The write timestamp of Xi is the timestamp of the transaction that wrote the value of version Xi.

- **Serializability in Multiversion Concurrency Control:**

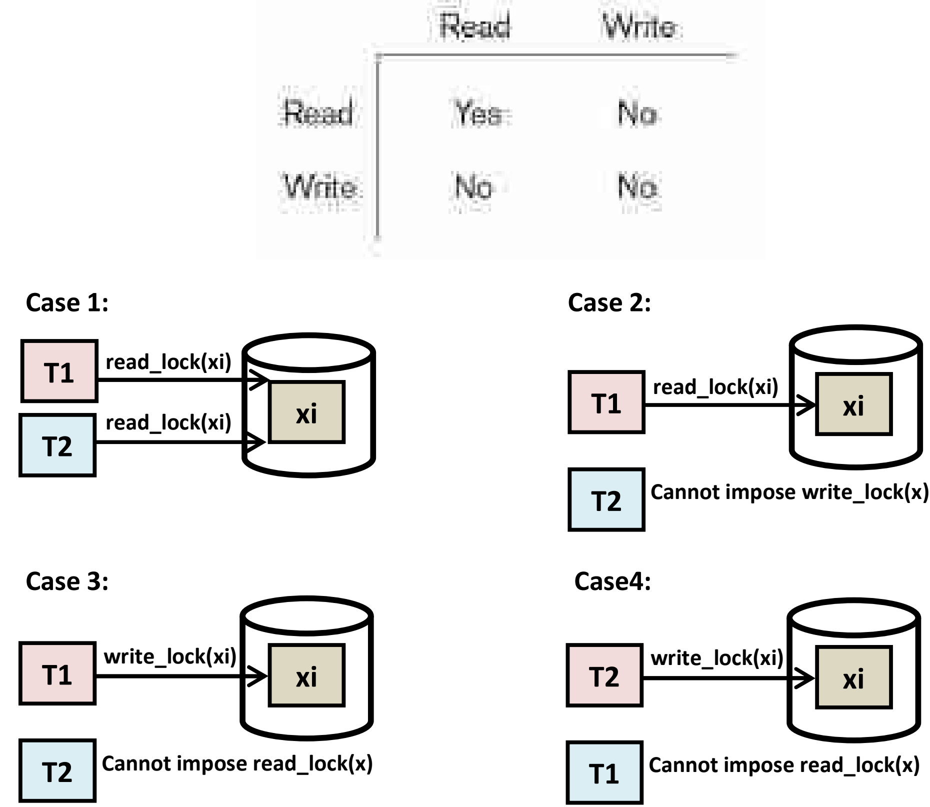
To ensure the serializability the follow rules are used

1. **If transaction T issues a write_item(X) operation, and version i of X has the highest write_TS(Xi) of all versions of X that is also less than or equal to TS(T), and read_TS(Xi) > TS(T), then abort and roll back transaction T; otherwise, create a new version Xj of X with read_TS(Xj) = write_TS(Xj) = TS(T).**

2. **If transaction T issues a read_item(X) operation, find the version i of X that has the highest write_TS(Xi) of all versions of X that is also less than or equal to TS(T); then return the value of Xi to transaction T, and set the value of read_TS(Xi) to the larger of TS(T) and the current read_TS(Xi).**

- **Multiversion Two-Phase Locking Using Certify Locks:**

1. **Compatability table for *read/write* locks:**



	Read	Write
Read	Yes	No
Write	No	No

2. **Compatability table for read/write/certify locks:**

Certify Lock (or) Notify Lock: Certify lock has highest priority. It is the monarch lock where other locks can not be done.

	Read	Write	Certify
Read	Yes	Yes	No
Write	Yes	No	No
Certify	No	No	No

DISTRIBUTED DATABASE CONCEPTS

TOPIC1: DISTRIBUTED DATABASE CONCEPTS

- Centralized database: Centralized database is a software system that creates & access data base at central site
- Disadvantage of central site: If central site fails, database is not available for other sites in the network

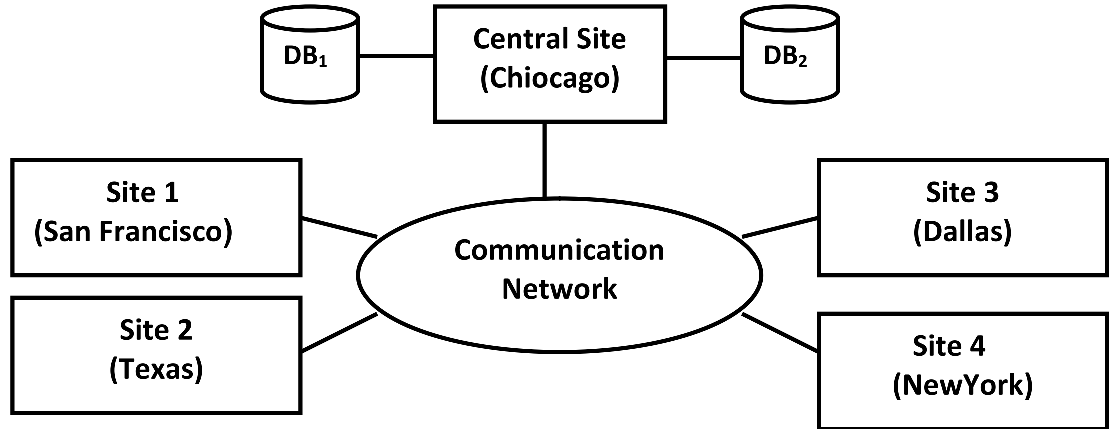


Figure 1. Centralized Database Architecture.

- Distributed Database Management System (DDBMS): Distributed database management system is a software system that *creates & access* the distributed database over a computer network.

1.1 DIFFERENCES BETWEEN DDB AND MULTIPROCESSOR SYSTEMS

- Connection of database nodes over a computer network: There are multiple computers, called sites or nodes. These sites must be connected by an underlying communication o transmit data and commands among sites.
- Logical interrelation of the connected databases: It is essential that the information in the databases be logically related.
- Absence of homogeneity constraint among connected nodes: It is not necessary that all nodes be identical in terms of data, hardware, and software.

1.2 TRANSPARENCY

(a) *Data Organization Transparency* (or) *Distribution or Network Transparency*	This refers the user can access any site across the network. ▪ *Location Transparency*: The commands used to perform a task is independent of the location of data and location of system across the network. ▪ *Naming Transparency*: The name of the objects can be accessed unambiguously without additional specifications.
(b) Replication Transparency	Copies of data may be stored at multiple sites for better *availability*, *performance*, & *reliability*.
(c) Fragmentation Transparency	Two types of fragmentation are possible 1. *Horizontal Fragmentation*: Distributes the relation into set of tuples (rows). 2. *Vertical Fragmentation*: Distributes the relation into sub relations where each sub relation is defined by a subset of the columns of the original relation.

Figure 2. A Truly Distributed Database Architecture

1.3 AUTONOMY

- Autonomy determines the extent to which individual nodes or databases in a connected Distribute Data Base can operate independently.
- A high degree of autonomy is desirable for increased flexibility and customized maintenance of an individual node.
- Autonomy can be applied to design, communication, and execution.

1. *Design Autonomy*: Design autonomy refers to independence of data model usage and transaction management techniques among nodes.
2. *Communication Autonomy*: Communication autonomy determines the extent to which each node can decide on sharing of information with other nodes.
3. *Execution Autonomy*: Execution autonomy refers to independence of users to act as they please.

1.4 RELIABILITY & AVALIABILITY

- Reliability: Reliability is broadly defined as the probability that a system is running (not down) at a certain time point.

- Availability: The probability that the system is continuously available during a time interval.

1.5 ADVANTAGES OF DISTRIBUTED DATABASE

1. Improved ease and flexibility of application development.

2. Increased reliability and availability

3. Improved performance

- Data localization reduces the contention for CPU and I/O services and reduce access delay.

- Local queries & transactions accessing data at a single site have better performance because of the smaller local databases.

- *Inter query* & *Intra query* parallelism can be achieved by executing multiple queries at different queries at different sites or by breaking up query into a number of sub queries that execute in parallel.

4. Easier expansion.

1.6 FUNCTIONS OF DDBMS

2. *Keeping Track of Data Distribution:* The ability to keep track of the data distribution, fragmentation, and replication by expanding the DDBMS catalog.

3. *Distributed Query Processing:* The ability to access remote sites and transmit queries and data among the various sites via a communication network.

4. *Distributed Transaction Management:* The ability to access remote sites and transmit queries and data among the various sites via a communication network.

5. *Replicated Data Management:* The ability to decide which copy of a replicated data item to access.

6. *Distributed Database Recovery:* The ability to recover from individual site crashes and from new types of failures, such as the failure of communication links.

7. *Security:* Distributed transactions must be executed with the proper management of the security of the data and the authorization/access privileges of users.

8. *Distributed Directory (Catalog) Management:* A directory contains information (metadata) about data in the database. The directory may be global for the entire DDB, or local for each site.

TOPIC 2: DATA FRAGMENTATION, REPLICATION, AND ALLOCATION TECHNIQUES FOR DISTRIBUTED DATABASE DESIGN VIMP

- **Fragment:** Subset of table.

2.1 DATA FRAGMENTATION

- Fragmentation is the task of dividing a table into a set of smaller tables.
- The subsets of the table are called fragments.
- Fragmentation can be of three types: horizontal, vertical, and hybrid.

EMPLOYEE		
EMPNO	**ENAME**	**SALARY**
1	**JOHN**	**3400**
2	**MARTIN**	**4300**
3	**ELICA**	**4500**
4	**MILLER**	**3400**

2.1.1 Horizontal Fragmentation: A horizontal fragment of a relation is a subset of the tuples in that relation. The tuples that belong to the horizontal fragment are specified by a condition on one or more attributes of the relation.

EMPLOYEE		
EMPNO	**ENAME**	**SALARY**
1	**JOHN**	**3400**
2	**MARTIN**	**4300**

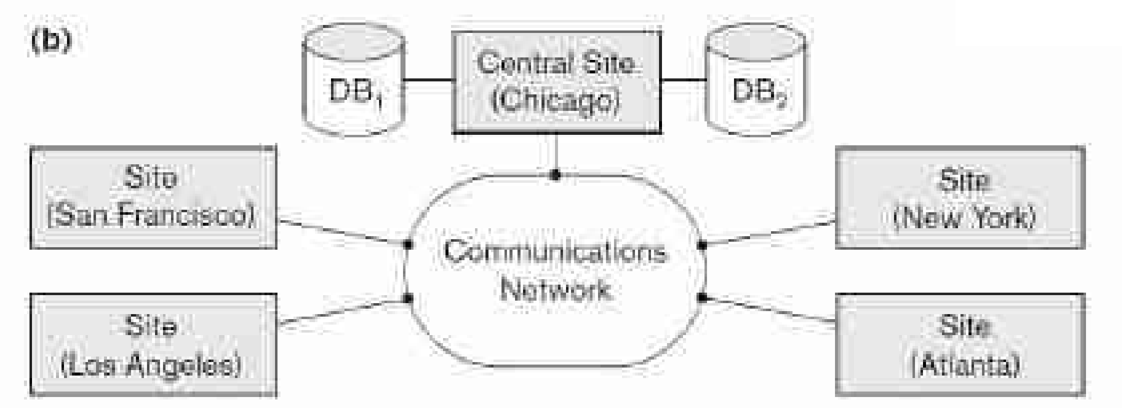
2.1.2 Vertical Fragmentation: Vertical fragmentation divides a relation “vertically” by columns. A vertical fragment of a relation keeps only certain attributes of the relation.

EMPLOYEE	
EMPNO	**ENAME**
1	**JOHN**
2	**MARTIN**
3	**ELICA**
4	**MILLER**

2.1.3 Mixed Fragmentation: We can intermix the two types of fragmentation, yielding a mixed fragmentation.

EMPLOYEE	
EMPNO	**ENAME**
1	**JOHN**
2	**MARTIN**

2.2 DATA REPLICATION AND ALLOCATION



- Data Replication is the process of storing data in more than one site or node.
- It is useful in improving the availability of data.
- It is simply copying data from a database from one server to another server so that all the users can share the same data without any inconsistency.
- The most extreme case is replication of the whole database at every site in the distributed system, thus creating a fully replicated distributed database.
- The other extreme from full replication involves having no replication that is, each fragment is stored at exactly one site. This is also called non redundant allocation.

TOPIC 3: LOCKS FOR CONCURRENCY CONTROL IN INDEXES

- Types of Locks for Concurrency Control in indexes are IX (Intension Exclusive), IS (Intension Shared), Shared (S), Exclusive (X).

- All the locking operations must precede with unlock operations in the same order and indexes.

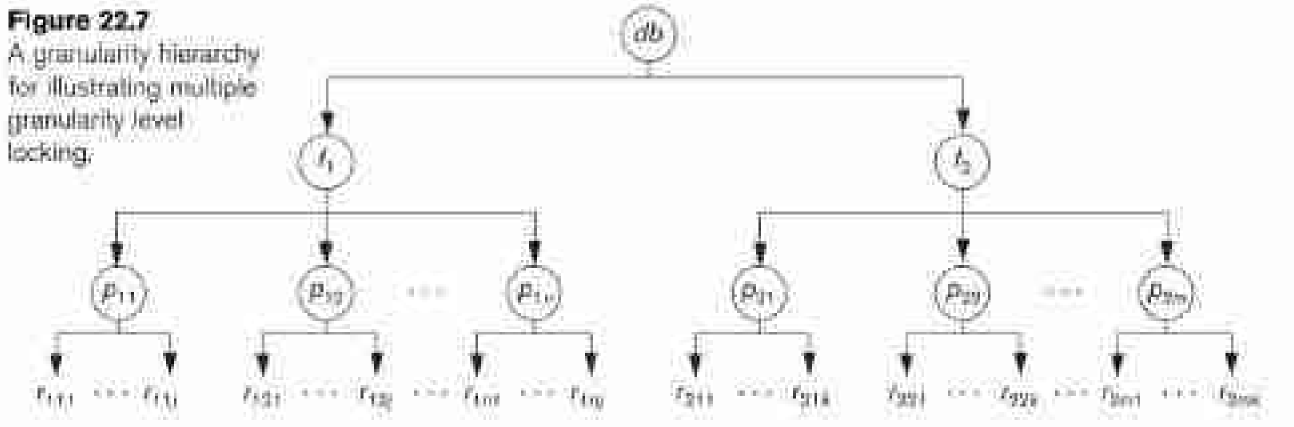


Figure 22.7
A granularity hierarchy for illustrating multiple granularity level locking.

T_1	T_2	T_3
IX(db) IX(f_1)		
	IX(db)	
		IS(db) IS(f_1) IS(p_{11})
IX(p_{11}) X(r_{111})		
	IX(f_1) X(p_{12})	
		S(r_{11j})
IX(f_2) IX(p_{21}) X(p_{211})		
unlock(r_{211}) unlock(p_{21}) unlock(f_2)		
		S(f_2)
	unlock(p_{12}) unlock(f_1) unlock(db)	
unlock(r_{111}) unlock(p_{11}) unlock(f_1) unlock(db)		
		unlock(r_{11j}) unlock(p_{11}) unlock(f_1) unlock(f_2) unlock(db)

TOPIC 4: QUERY PROCESSING AND OPTIMIZATION

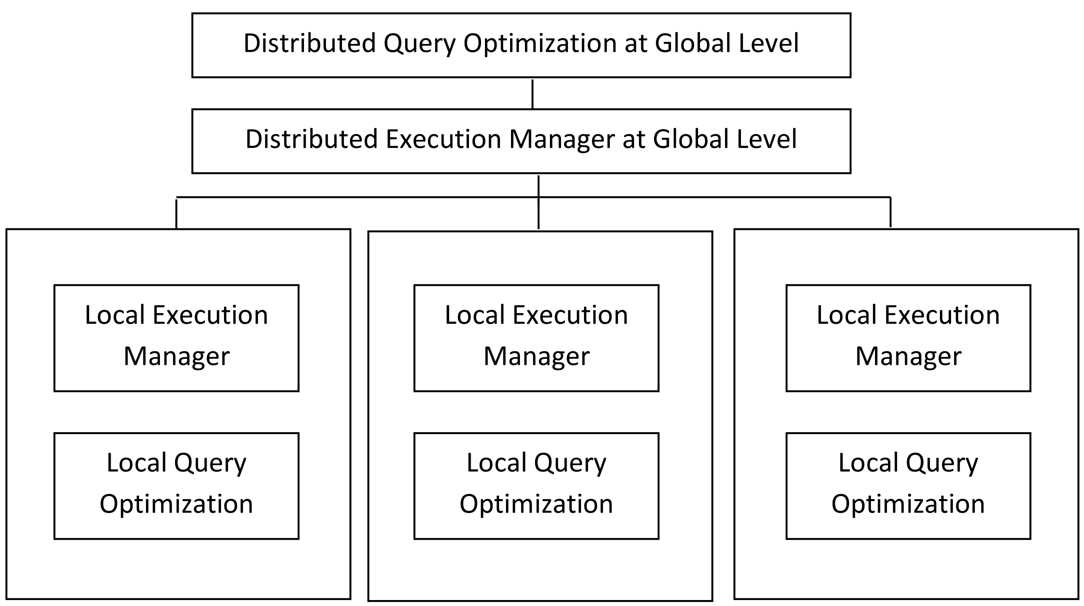
- Query Processing and Optimization determines the efficient way to execute a query with different possible query plans.
- It cannot be accessed directly by users once the queries are submitted to the database server or parsed by the parser.
- A query is passed to the query optimizer where optimization occurs.
- Main aim of Query Optimization is to minimize the cost function i.e. *I/O Cost*, *CPU Cost*, *Communication Cost*.
- It defines how an RDBMS can improve the performance of the query by re-ordering the operations.
- It is the process of selecting the most efficient query evaluation plan from among various strategies if the query is complex.
- It computes the same result as per the given expression, but it is a least costly way of generating result.

Advantages:

- It requires less cost per query.
- It gives less stress to the database.
- It provides high performance of the system.
- It consumes less memory.

A distributed database query is processed in stages as follows:

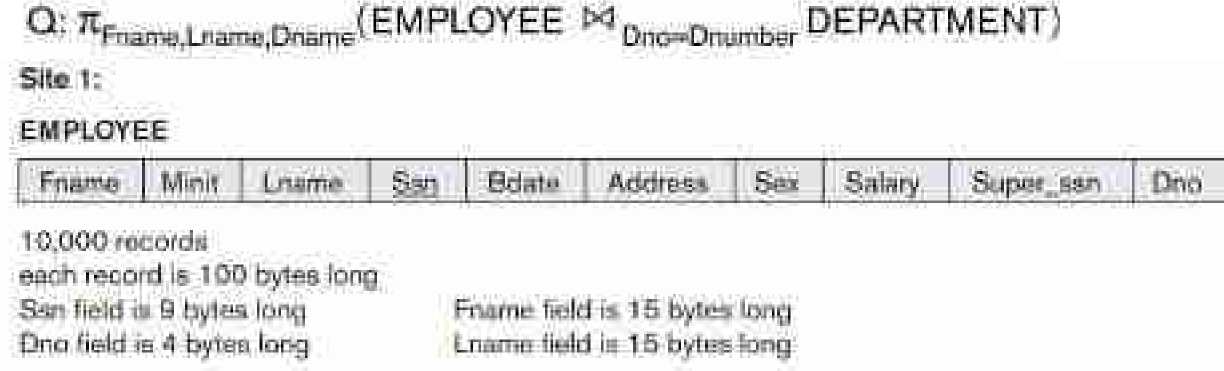
1. Query Mapping: Mapping joins multiple source tables.
2. Localization: Data localization is the practice of keeping data within the region it originated from. For example, if an organization collects data in the UK, they store it in the UK rather than transferring it to another country for processing.
3. Global Query Optimization: The tables required in a global query have fragments distributed across multiple sites. The local databases have information only about local data. The controlling site uses the global data dictionary to gather information about the distribution and reconstructs the global view from the fragments.



4.1 Data Transfer Costs of Distributed Query Processing

- Suppose that the EMPLOYEE and DEPARTMENT relations are distributed at two sites.
- Consider the query Q: For each employee, retrieve the employee name and the name of the department for which the employee works. This can be stated as follows in the relational algebra:

$$Q: \pi_{Fname,Lname,Dname}(EMPLOYEE \bowtie_{Dno=Dnumber} DEPARTMENT)$$

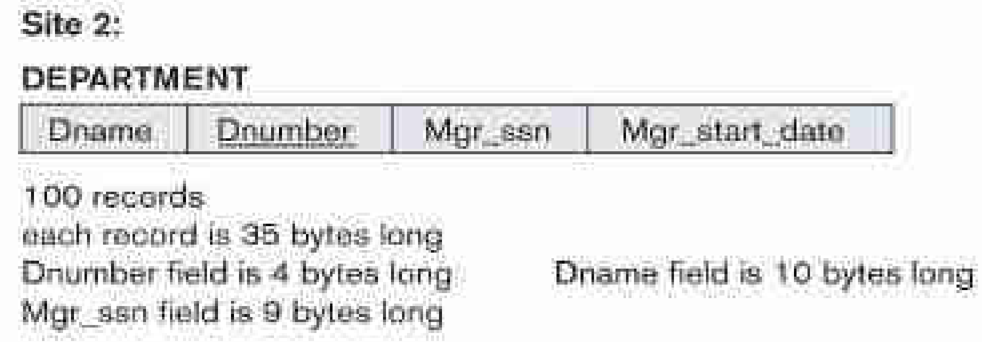

Site 1:

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

10,000 records
each record is 100 bytes long
Ssn field is 9 bytes long
Dno field is 4 bytes long
Fname field is 15 bytes long
Lname field is 15 bytes long

- The size of EMPLOYEE relation is 10,000 × 100 = 1,000,000 bytes.


Site 2:

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

100 records
each record is 35 bytes long
Dnumber field is 4 bytes long
Mgr_ssn field is 9 bytes long
Dname field is 10 bytes long

- The size of DEPARTMENT relation is 100 × 35 = 3,500 bytes.
- The size of the attributes Fname, Lname and Dname is 40 bytes.

There are three simple strategies for executing this distributed query:

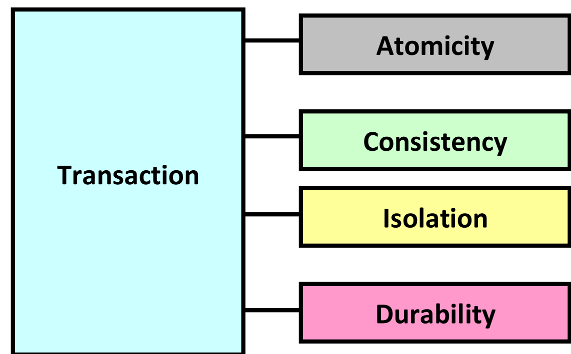
1. Transfer both the EMPLOYEE and the DEPARTMENT relations to the result site, and perform the join at site 3. In this case, a total of 1,000,000 + 3,500 = 1,003,500 bytes must be transferred.
2. Transfer the EMPLOYEE relation to site 2, execute the join at site 2, and send the result to site 3. The size of the query result is 40 × 10,000 = 400,000 bytes, so 400,000 + 1,000,000 = 1,400,000 bytes must be transferred.
3. Transfer the DEPARTMENT relation to site 1, execute the join at site 1, and send the result to site 3. In this case, 400,000 + 3,500 = 403,500 bytes must be transferred.

   We choose strategy 3 for optimization criteria.

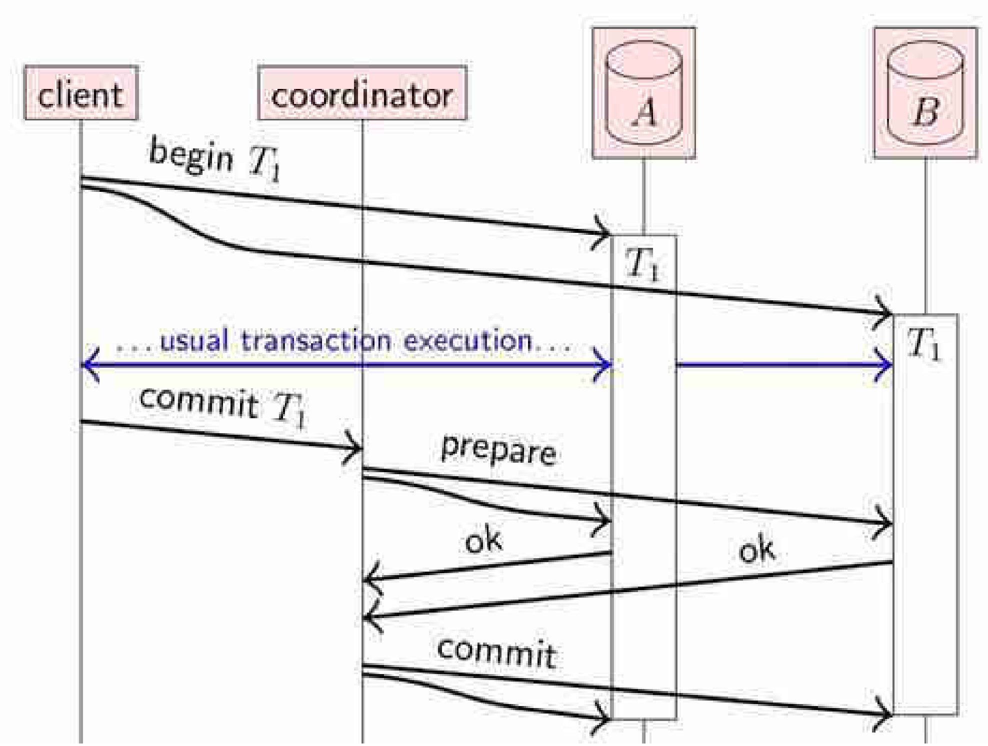
TOPIC 5: TRANSACTION MANAGEMENT IN DISTRIBUTED DATABASES

- Global Transaction Manager is introduced for supporting distributed transactions.
- Global Transaction Manager coordinate the execution of database operations like BEGIN_TRANSACTION, READ or WRITE, END_TRANSACTION, COMMIT_TRANSACTION, and ROLLBACK (or ABORT) across multiple sites.
- The Global Transaction Manager stores bookkeeping information related to each transaction, such as a Unique Identifier, Originating Site, Name, and so on.
- For READ operations, it returns a local copy if valid and available.
- For WRITE operations, it ensures that updates are visible across all sites containing copies (replicas) of the data item.
  - For ABORT operations, the manager ensures that no effects of the transaction are reflected in any site of the distributed database.
  - For COMMIT operations, it ensures that the effects of a write are persistently recorded on all databases containing copies of the data item.
  - Atomic termination (COMMIT/ ABORT) of distributed transactions is commonly implemented using the two-phase commit protocol.

5.1 ACID properties of Transaction



- Atomicity: The transaction is atomic unit of processing i.e. either performed entirely or none performed at all.
- Consistency: A correct execution of transaction must take the database item from one consistent state to another consistent state.
- Isolation: A transaction is isolated from other transactions when multiple transactions are accessing the same data item concurrently in a *serial* or *interleaving* fashion.
- Durability: The changes applied to the database item by a commited transaction must persistent in the database.



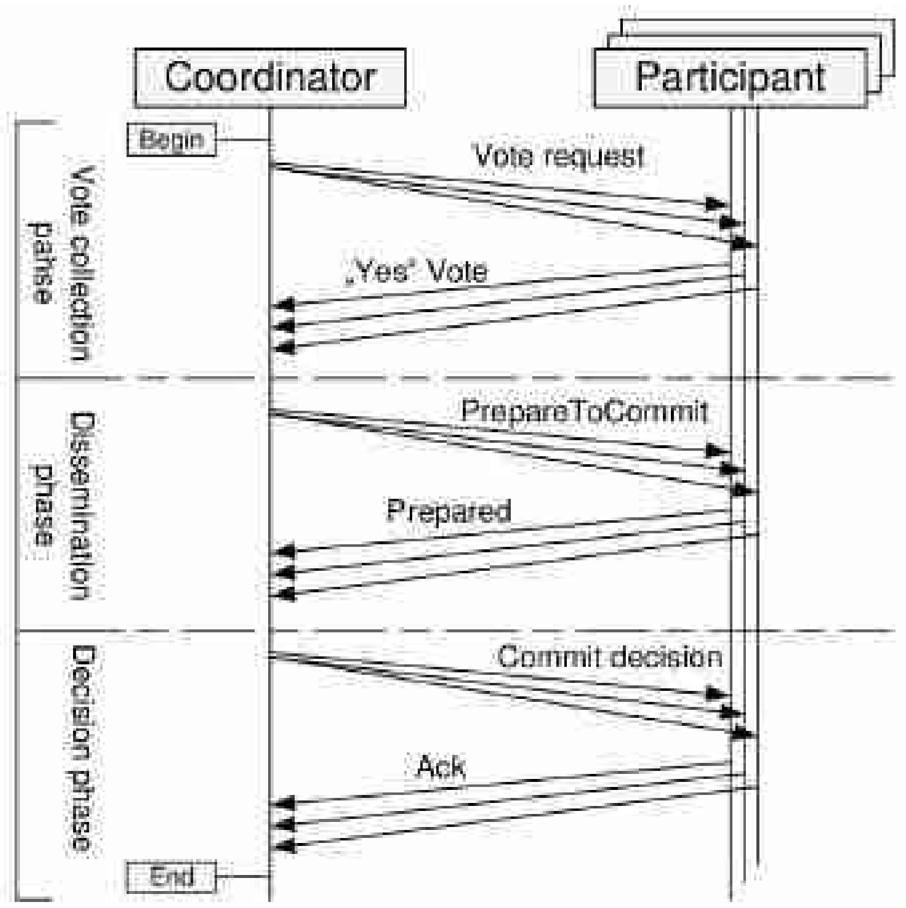
5.2 Two Phase Commit Protocol

If transaction updates data on multiple nodes, this implies

- Either all nodes must omit, or all nodes must abort
- If an node crashes, all must abort

5.3 Three-Phase Commit Protocol

- The biggest drawback of 2PC is that it is a blocking protocol. Failure of the coordinator blocks all participating sites, causing them to wait until the coordinator recovers.
- This can cause performance degradation, especially if participants are holding locks to shared resources. Other types of problems may also occur that make the outcome of the transaction nondeterministic.



5.4 Operating System support for Transaction Management

- Semaphore: Semaphore in OS is an **integer value** that indicates whether the resource required by the process is available or not. The value of a semaphore is modified by wait () or signal () operation where the wait () operation decrements the value of semaphore and the signal () operation increments the value of the semaphore.
- OS supports semaphore to guarantee mutually exclusive access to shared resources.
- Specialized hardware support for locking can be provided.
- Common transaction support operations can be supported.

TOPIC 6: CONCURRENCY CONTROL AND RECOVERY IN DISTRIBUTED DATABASES

- For concurrency control and recovery purposes, numerous problems arise in a distributed DBMS environment that are not encountered in a centralized DBMS environment. These include the following.

1. Dealing with multiple copies of the data items.
2. Failure of individual sites.
3. Failure of communication links.
4. Distributed commit.
5. Distributed deadlock.

6.1 Distributed Concurrency Control Based on a Distinguished Copy of a Data Item

Primary Site Technique: In this method, a single primary site is designated to be the coordinator site for all database items. Hence, all locks are kept at that site, and all requests for locking or unlocking are sent there.

Primary Site with Backup Site: In this method, all locking information is maintained at both the primary and the backup sites. In case of primary site failure, the backup site takes over as the primary site, and a new backup site is chosen.

This simplifies the process of recovery from failure of the primary site, since the backup site takes over and processing can resume after a new backup site is chosen and the lock status information is copied to that site.

Primary Copy Technique:

- This method attempts to distribute the load of lock coordination among various sites by having the distinguished copies of different data items stored at different sites. Failure of one site affects any transactions that are accessing locks on items whose primary copies reside at that site, but other transactions are not affected. This method can also use backup sites to enhance reliability and availability.

Choosing a New Coordinator Site in Case of Failure:

- Whenever a coordinator site fails in any of the preceding techniques, the sites that are still running must choose a new coordinator.
- If a backup site X is about to become the new primary site, X can choose the new backup site from among the system's running sites. However, if no backup site existed, or if both the primary and the backup sites are down, a process called election can be used to choose the new coordinator site.
- In this process, any site Y that attempts to communicate with the coordinator site repeatedly and fails to do so can assume that the coordinator is down and can start the election process by sending a message to all running sites proposing that Y become the new coordinator. As soon as Y receives a majority of yes votes, Y can declare that it is the new coordinator. The election algorithm itself is complex, but this is the main idea behind the election method. The algorithm also resolves any attempt by two or more sites to become coordinator at the same time.

6.2 Distributed Concurrency Control Based on Voting:

- If a lock request is sent to all sites that includes a copy of the data item. Each copy maintains its own lock and can grant or deny the request for it.
- If a transaction that requests a lock is granted that lock by a majority of the copies, it holds the lock and informs all copies that it has been granted the lock.
- If a transaction does not receive a majority of votes granting it a lock within a certain time-out period, it cancels its request and informs all sites of the cancellation.

TOPIC7: TYPES OF DISYRIBUTED DATBASES

7.1 FEDERATED DATABASE MANAGEMENT SYSTEMS

- A Federated Database System (FDBS) is a collection of cooperating database systems that are autonomous and possibly heterogeneous.

- The type of heterogeneity present in FDBSs may arise from several sources.

TYPES OF HETEROGENEITY:

1. *Differences in Data Models:*

- Data is represented using different data models. (Hierarchical, Relational, Network & Object Oriented data models).

2. *Differences in Constraints:*

- The global schema must also deal with different constraints. (Not null, Unique, Primary Key, Foreign Key, Check)
- ER models are represented as referential integrity constraints in the relational model.
- Triggers may have to be used to implement certain constraints in the relational model.

3. *Differences in Query Languages:*

- Even with the same data model, the languages and their versions vary. For example, SQL has multiple versions like SQL-89, SQL-92, SQL-99, and SQL: 2008, and each system has its own set of data types, comparison operators, string manipulation features, and so on.

7.1.2 SEMANTIC HETEROGENEITY.

- Semantic heterogeneity occurs when there are differences in the meaning, interpretation, and intended use of the same or related data.

1. *The universe of discourse from which the data is drawn.*

- For example, for two customer accounts, databases in the federation may be from the United States and Japan and have entirely different sets of attributes about customer accounts required by the accounting practices. Currency rate fluctuations would also present a problem. Hence, relations in these two databases that have identical names-CUSTOMER or ACCOUNT-may have some common and some entirely distinct information.

2. *Representation and Naming.*

- The representation and naming of data elements and the structure of the data model may be pre specified for each local database.

3. The understanding, meaning, and subjective interpretation of data.

- This is a chief contributor to semantic heterogeneity.

4. *Transaction and Policy Constraints.*

- These deal with *serializability criteria*, *compensating transactions*, and *other transaction policies*.

5. *Derivation of Summaries.*

- Aggregation, summarization, and other data processing features and operations supported by the system.

TOPIC 9: DISTRIBUTED DATABASE ARCHITECTURES IMP

9.1 PARALLEL VERSUS DISTRIBUTED ARCHITECTURES

There are two main types of multiprocessor system architectures that are commonplace:

- Shared Memory (tightly coupled) Architecture: Multiple processors share secondary (disk) storage and also share primary memory.
- Shared Disk (loosely coupled) Architecture: Multiple processors share secondary (disk) storage but each has their own primary memory.

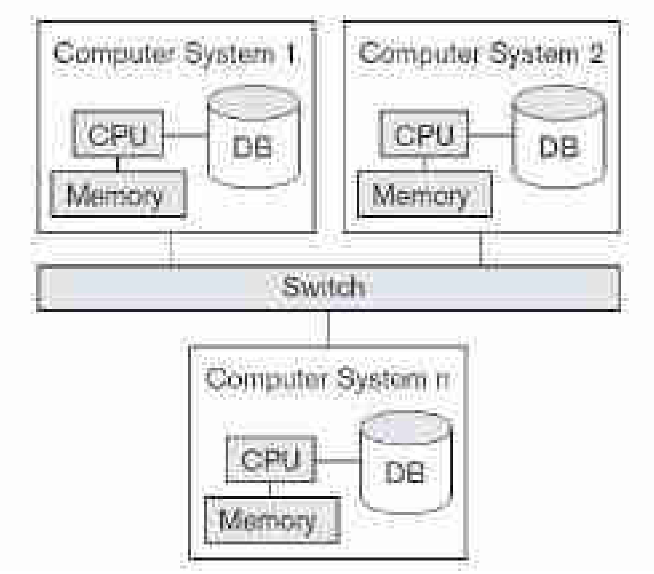


Fig 1. Shared nothing architecture.

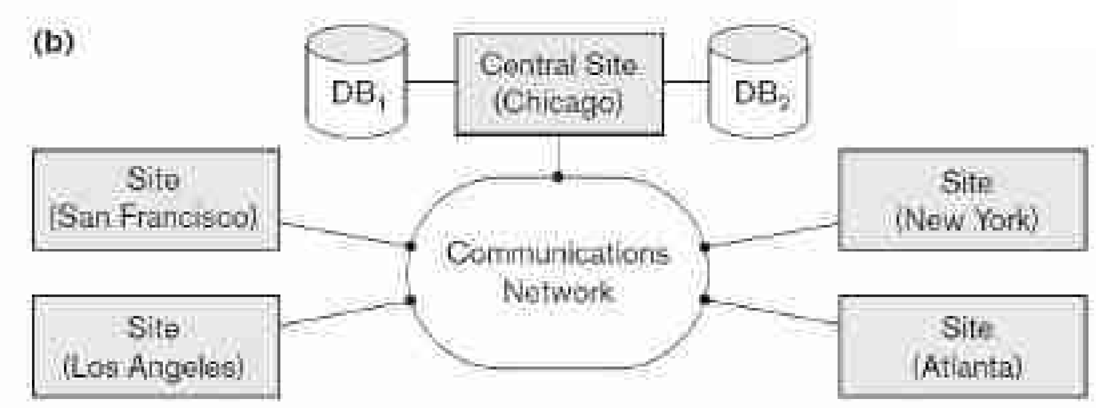


Fig 2. A networked architecture with a centralized database at one of the sites.

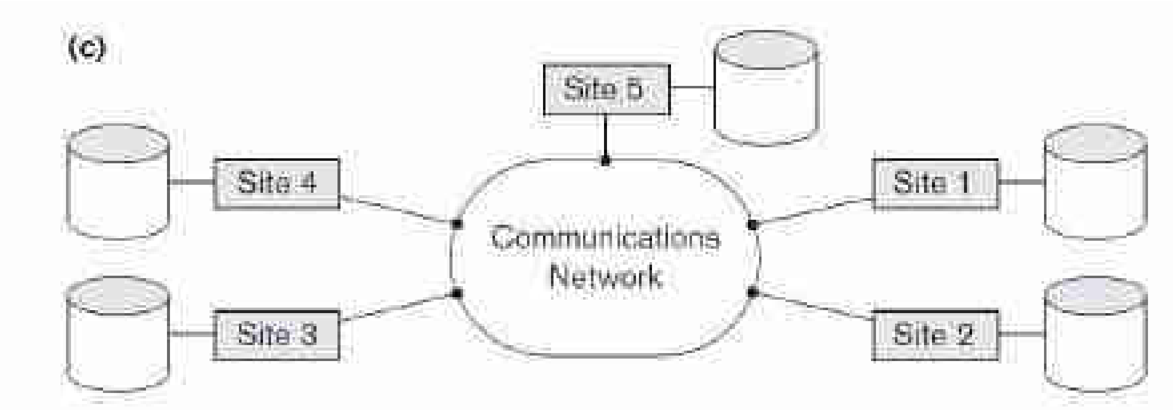


Fig 3. A truly distributed database architecture

9.2 GENERAL ARCHITECTURE OF PURE DISTRIBUTED DATABASES

- In this section we discuss both the logical and component architectural models of a DDB.

9.2.1 SCHEMA ARCHITECTURE OF DISTRIBUTED DATABASES.

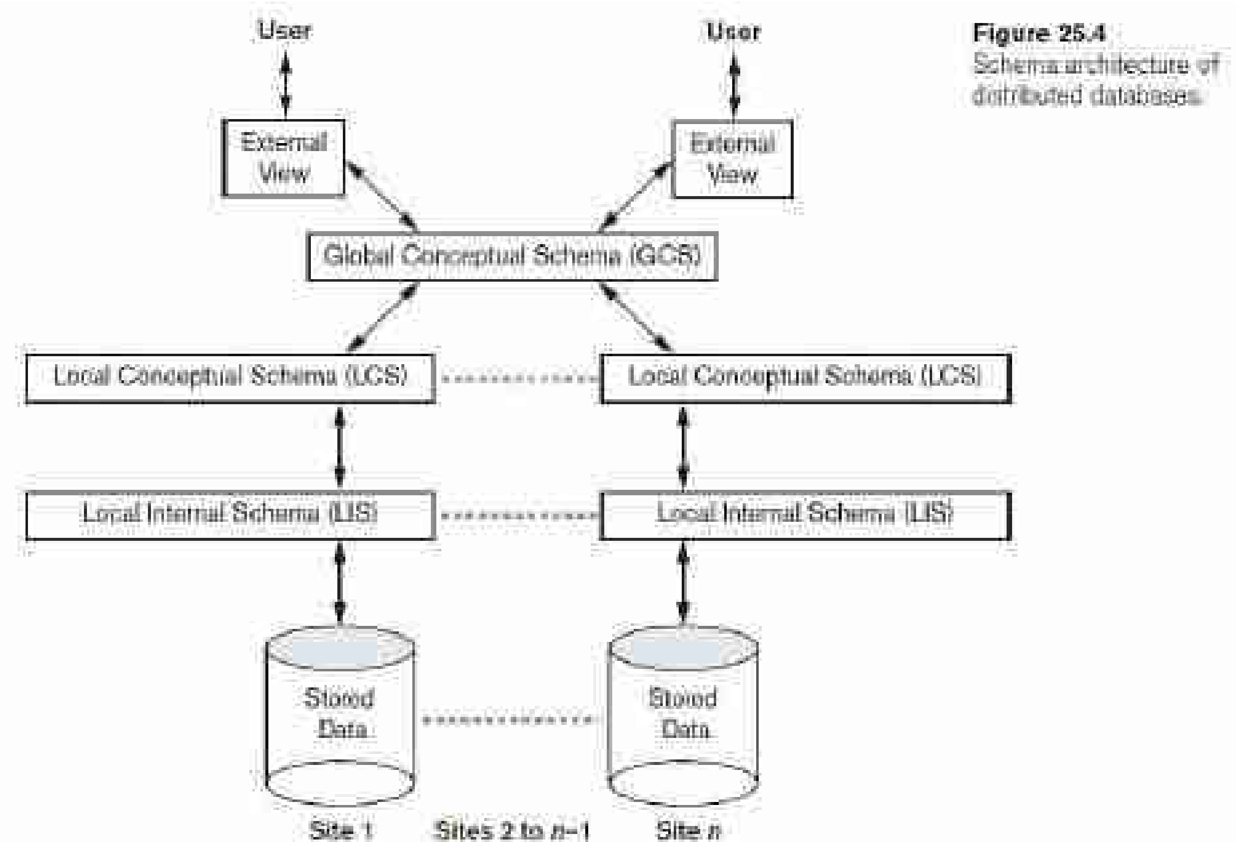


Fig 4. Schema Architecture of Distributed Database.

- *Logical Internal Schema (LIS)* specifies *physical organization* details at that particular site.
- *Logical Conceptual Schema (LCS)* specifies the *logical organization* of data at each site is specified by the local conceptual schema
- *The Global Conceptual Scheme* (GCS) specifies provides *network transparency.*

9.2.2 COMPONENT ARCHITECTURE OF DISTRIBUTED DATABASES.

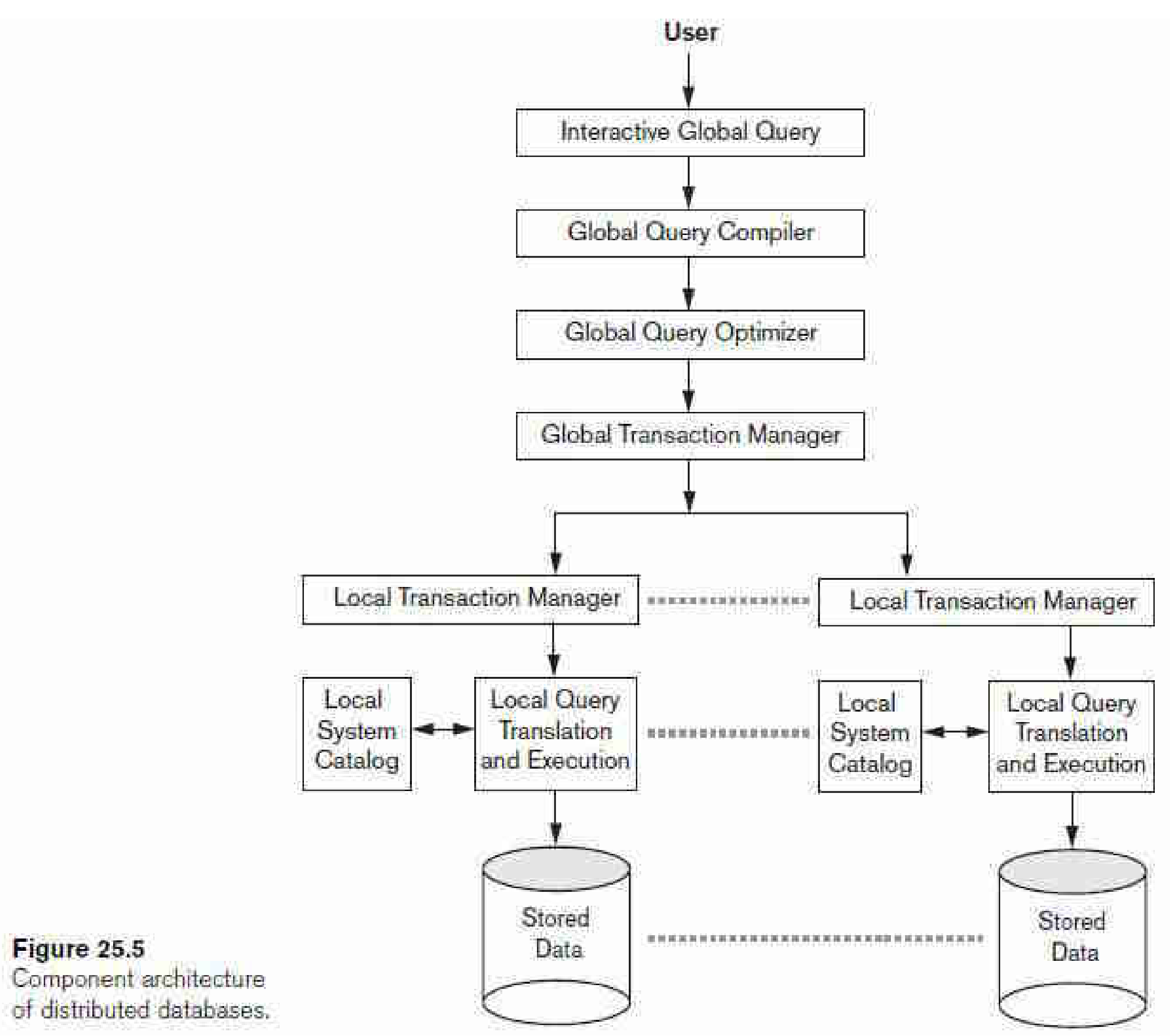


Fig 5. Component architecture of distributed database.

- Query Compiler: Inspects the process of query compilation.
- Query Optimizer: The optimizer selects the minimum cost for execution.
- Transaction Manager: Coordinates and executes the transactions.

- System Catalog: The system catalogue stores meta-data including the schemas of the databases.

9.2.3 FEDERATED DATABASE SCHEMA ARCHITECTURE

- Federated Schema is an integration of multiple export schemas.

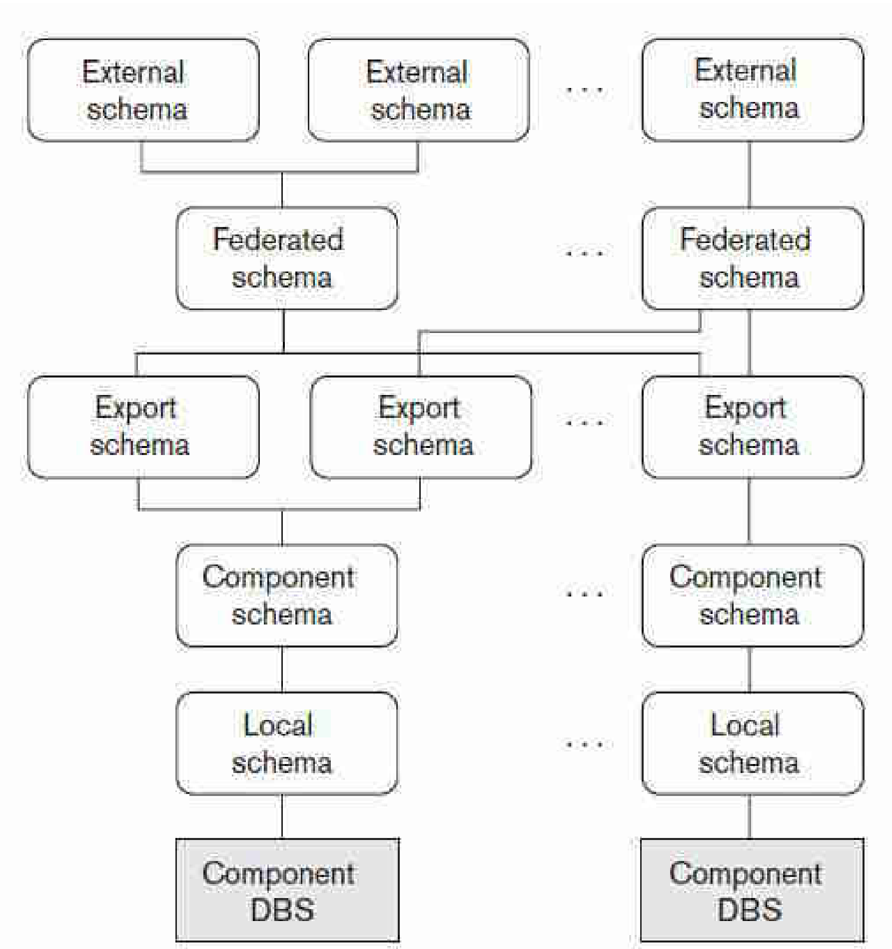


Fig 6. The five-level schema architecture in a federated database system (FDBS).

- Component DBS: Database control system.
- Local schema: Conceptual schema.
- Component schema: Translates the local schema into common data model.
- Export schema: Subset of component schema.
- Federated schema: The federated schema is the global schema or view, which is the result of integrating all the shareable export schemas.
- External schema: External schemas define the schema for a user group or an application.

9.2.4 THREE-TIER CLIENT-SERVER ARCHITECTURE

In the three-tier client-server architecture, the following three layers exist:

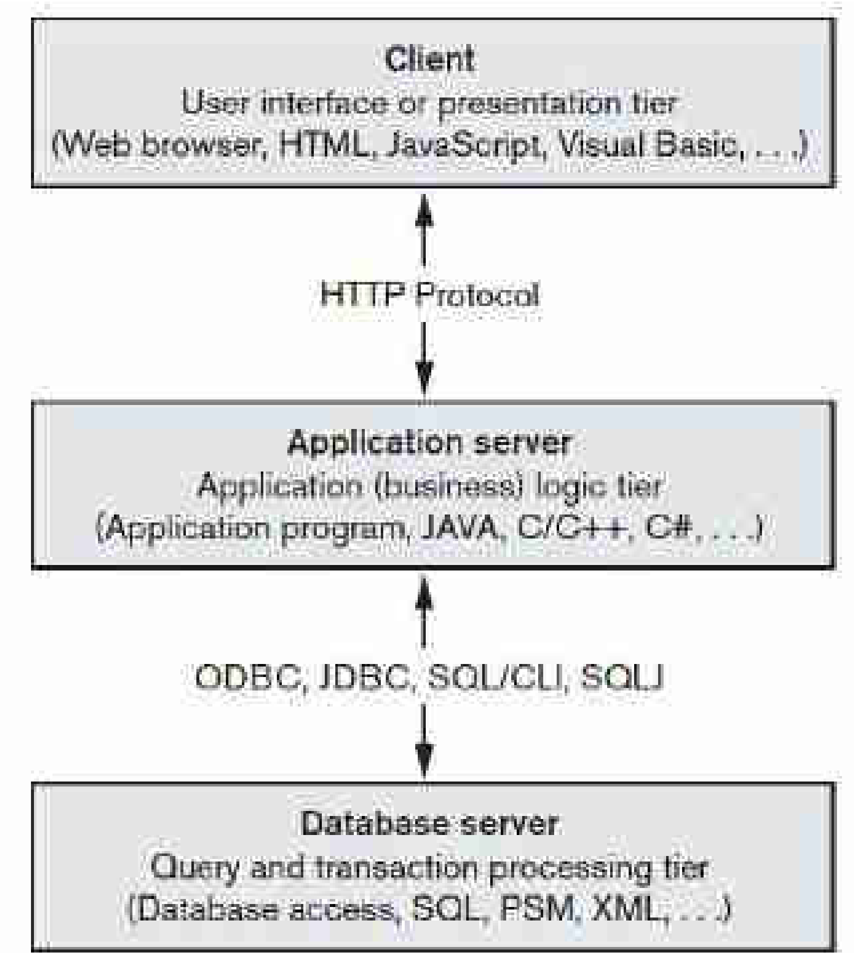


Fig 7. The three tire client server architecture.

1. Presentation Layer (Client):

- This provides the user interface and interacts with the user.
- The programs at this layer present *web interfaces* or *forms* to the client in order to interact with the application.
- Web browsers are often utilize the languages like *HTML*, *XHTML*, *CSS*, *Java*, *JavaScript* and others to perform *client side validations*.

2. Application Layer (Business Logic):

- Application logic is written in this phase.
- Queries can be formulated based on user input from the client.
- Query results can be formatted and sent to the client for presentation.
- Additional application functionality can be handled such as security checks, identity verification, and other functions.
- The application layer can interact with one or more databases or data sources as needed by connecting to the database using ODBC, JDBC, SQL/CLI, or other database access techniques.

3. Database server.

  - This layer handles query and update requests from the application layer, processes the requests, and sends the results.
  - Usually SQL is used to access the database if it is relational or object-relational and stored database procedures may also be invoked.

Query results (and queries) may be formatted into XML when transmitted between the application server and the database ser

---

TOPIC 10: VALIDATION (OPTIMISTIC) CONCURRENCY CONTROL TECHNIQUES

- In all the concurrency control techniques we have discussed so far, a certain degree of checking is done before a database operation can be executed. For example, in locking, a check is done to determine whether the item being accessed is locked.
- In timestamp ordering, the transaction timestamp is checked against the read and write timestamps of the item. Such checking represents overhead during transaction execution, with the effect of slowing down the transactions.
- In optimistic concurrency control techniques, also known as validation or certification techniques, no checking is done while the transaction is executing.
- In this scheme, updates in the transaction are not applied directly to the database items until the transaction reaches its end.
- During transaction execution, all updates are applied to local copies of the data items that are kept for the transaction.
- At the end of transaction execution, a validation phase checks whether any of the transaction's updates violate serializability.

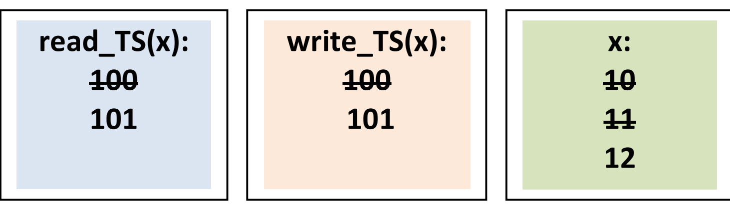
- If serializability is not violated, the transaction is committed and the database is updated from the local copies; otherwise, the transaction is aborted and then restarted later.

There are three phases for this *Concurrency Control Protocol*:

1. Read phase: A transaction can read values of committed data items from the database. However, updates are applied only to local copies (versions) of the data items kept in the transaction workspace.
2. Validation phase: Checking is performed to ensure that serializability will not be violated if the transaction updates are applied to the database.
3. Write phase: If the validation phase is successful, the transaction updates are applied to the database; otherwise, the updates are discarded and the transaction is restarted.

- The validation phase for Ti checks that, for each such transaction Tj that is either committed or is in its validation phase, one of the following conditions holds:

1. Transaction T_j completes its write phase before T_i starts its read phase.
2. T_i starts its write phase after T_j completes its write phase, and the read_set of T_i has no items in common with the write_set of T_j.
3. Both the read_set and write_set of T_i have no items in common with the write_set of T_j, and T_j completes its read phase before T_i completes its read phase.



$TS(T_j)$=100
T_j
read_item(x); 10
x:=x+1; 11
write_item(x); 11
commit;

$TS(T_i)$=101
T_i
read_item(x); 11
x:=x+1; 12
write_item(x); 12
commit;

Read Set of T_i	Read Set of T_j
x	x

Write Set of T_i	Write Set of T_j
x	x

CHAPTER 1
WHY NoSQL?

1.1 WHY NOSQL-INTRODUCTION

- For more than four decades people are using relational databases as a primary data storage mechanism.
- Structured Query Language (SQL) happens to be the more structured, rigid way of storing data, like a phone book. They are designed for reliable transactions and follow a proper structure to store data in a very organized manner.
- Relational databases can handle thousands of queries in just a fraction of seconds but this is possible at the small scale applications.
- When the application grows relational databases start facing the ***scalability issue***. If we talk about the big gigantic website (such as Facebook, Google, Amazon) that throws billions or trillions of queries within a small amount of time then relational databases get failed in handling the queries.
- To get rid of this limitation in relational databases NoSQL comes in the picture that mainly focuses on ***high operations speed*** **and** ***flexibility*** in storing the data.

Let's discuss 5 important features of NoSQL databases.

a. Multi-Model

- Relational databases store data in a **fixed and predefined structure**. It means when you start development you will have to define your data schema in terms of tables and columns. You have to change the schema every time the requirements change. This will lead to creating new columns, defining new relations, reflecting the changes in your application, discussing with your database administrators, etc.

- NoSQL database provides much more flexibility when it comes to handling data. There is **no requirement to specify the schema** to start working with the application. Also, the NoSQL database doesn't put a restriction on the types of data you can store together. It allows you to add more new types as your needs change. These all are the reasons NoSQL is best suited for **agile development** which requires **fast implementation**. Developers and architects choose NoSQL to handle data easily for various kinds of agile development application requirements.

b. Easily Scalable

- The primary reason to choose a NoSQL database is easy scalability. Relational databases can also be scaled but not easily and at a lower cost. Relational databases are built on the

concept of traditional master-slave architecture. Scaling up means upgrading your servers by adding more processors, RAM, and hard-disks to your machine to handle more load and increase capacity. You will have to divide the databases into smaller chunks across multiple hardware servers instead of a single large server. This is called **sharding** which is very complicated in relational databases. Replacing and upgrading your database server machines to accommodate more throughput results in downtime as well. These things become a difficulty for developers and architects.

- NoSQL database built with a **masterless, peer-to-peer architecture**. Data is partitioned and balanced across **multiple nodes** in a cluster, and aggregate queries are distributed by default. This allows easy scaling in no time. Just executing a few commands will add the new server to the cluster. This scalability also improves performance, allowing for continuous availability and very **high read/write speeds**.

c. Distributed

- Relational databases use a **centralized application** that is **location-dependent** (e.g. single location), especially for write operations. On the other hand, the NoSQL database is designed to distribute data on a global scale.

- It uses **multiple locations involving multiple data centers** and/or cloud regions for write and read operations. This distributed database has a great advantage with master class architecture. You can maintain continuous **availability** because data is distributed with multiple copies where it needs to be.

d. Redundancy and Zero Downtime

- NoSQL is also designed to handle hardware failures. Hardware failure is a serious concern while building an application. Rather than requiring developers, DBAs, and operations staff to build their redundant solutions, it can be addressed at the architectural level of the database in NoSQL. If we talk about **Cassandra** then it uses several heuristics to determine the likelihood of node failure. **Riak** follows the network partitioning approach (when one or more nodes in a cluster become isolated) and repair itself.
- The masterclass architecture of the NoSQL database allows multiple copies of data to be maintained across different nodes. If one node goes down then another node will have a copy of the data for easy and fast access. This leads to zero downtime in the NoSQL database. The down time of resource is almost zero.

e. Big Data Applications

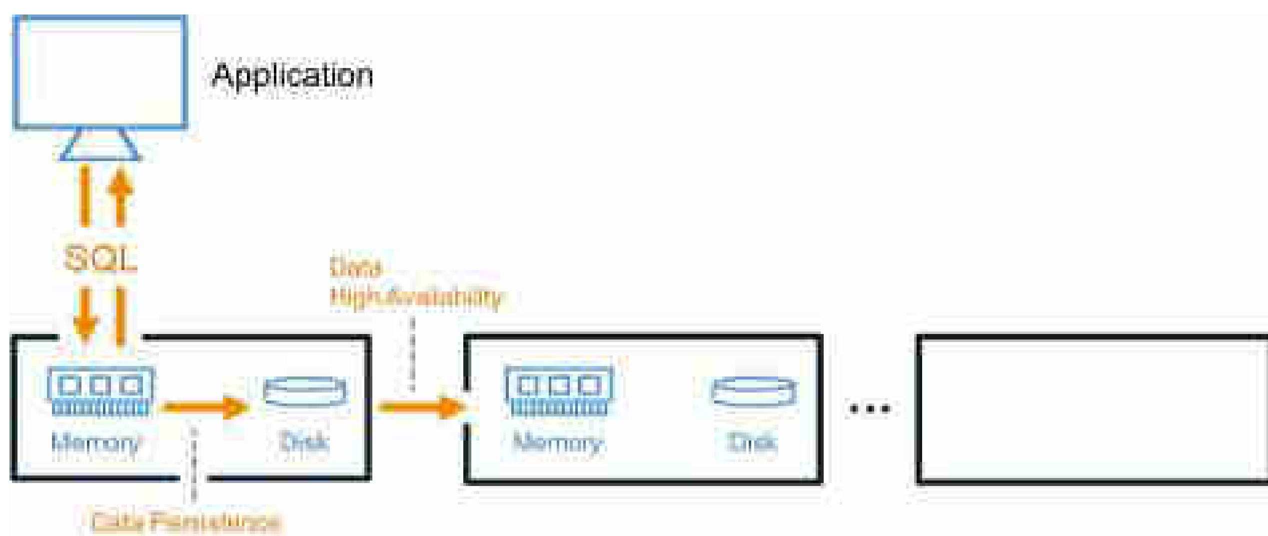
- NoSQL can handle the Massive Amount of data very quickly and that’s the reason it is best suited for the big data applications.

- NoSQL databases ensure data doesn't become the bottleneck when all of the other components of your server-side application are designed to be faultless and fast.

1.2 THE VALUE OF RELATIONAL DATABASES (OR) ADVANTAGES OF RELATIONAL DATABASES IMP

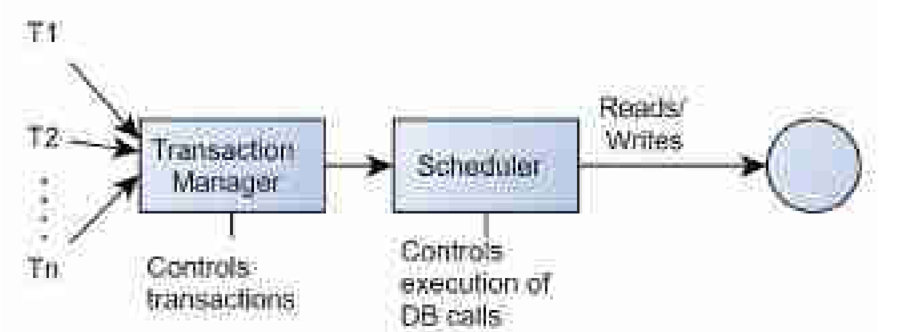
1.2.1 Getting at Persistent Data

- Data Persistence is a concept that defines a data is going to be saved on media, and therefore you can recover it even if your computer process is deletes it or corrupts it.
- Traditional Relational Database Management Systems (RDBMS) store persistent data in the form of records and tables. The records are permanent in the database even hardware or software changes.



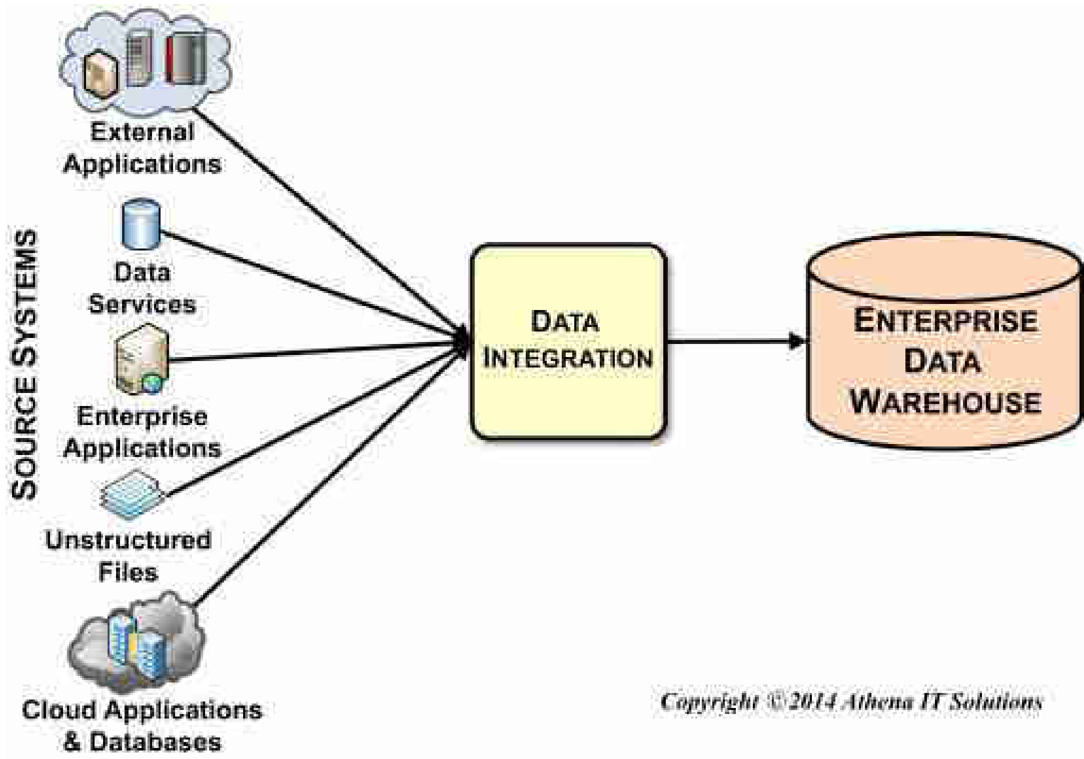
1.2.2 Concurrency Control

- Concurrency: A problem situation as a single data item is required by the multiple transactions simultaneously.
- In a Database Management System (DBMS), concurrency control manages simultaneous access to a database. It prevents two users from writing the same data item simultaneously and also serializes transactions for backup and recovery.



1.2.3. Integration

- In shared integration multiple applications store their data in a single database.



1.2.4 A Standard Model

- Developers and Database Professionals can learn the **Basic Relational Model** and apply it in many projects.
- Although there are differences between different relational databases, *SQL Type System* and *Transaction Processing* are same for all DBMS's.

1.2.5 Simple Model

- A Relational Database system is the simplest model, as it does not require any complex structuring or querying processes. It follows Network Model to establish relationships between various tables.

1.2.6 Data Accuracy

- Various Relational Model Constraints validates the data at time during entry time to ensure accuracy of data. In the relational database system, there can be multiple tables related to one another with the use of a primary key and foreign key concepts. This makes the data to be non-repetitive. There is no chance for duplication of data. Hence the accuracy of data in the relational database is more than any other database system.

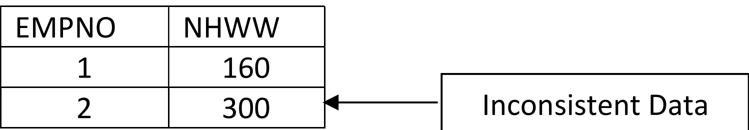
1.2.7. Easy Access to Data

- In the Relational Database System, there is no pattern or pathway for accessing the data, as to another type of databases can be accessed only by navigating through a tree or a hierarchical model. Anyone who accesses the data can query any table in the relational database. Using join queries and conditional statements one can combine all or any number of related tables in order to fetch the required data. Resulting data can be modified based on the values from any column, on any number of columns, which permits the user to effortlessly recover the relevant data as the result. It allows one to pick on the desired columns to be incorporated in the outcome so that only appropriate data will be displayed.

1.2.8 Data Integrity

The process of ensuring the correct data in the database is called Data Integrity. Integrity Constraints and relationships can be established to ensure the data integrity to certain extent.

E.g. Number of hours worked per week



EMPNO	NHWW
1	160
2	300

1.2.9 Multiple Transactions can be supported. (Multi Transaction System)

E.g. Online transaction processing system (Railway reservation)

- In multi transaction system, a single database item can be accessed by multiple users simultaneously.

1.2.10 Provides Security

- DBA assigns privileges (read, write & modify) to individual users and group of users to restrict the database access from unauthorized users.

1.2.11 Provide Multiple User Interface

- DBMS provides GUI’s, Query Languages, Application Programs and Menu Driven Interfaces to users.

1.2.12 Complex relationships can be established between the database objects using referential integrity.

1.2.13 Integrity constraints can be enforced.

1.2.14 Provides Backup and Recovery

- Using backup & recovery subsystems, software & hardware failures can be recovered.

1.2.15 Standards can be enforced

- DBA ensures all applicable standards while assigning privileges to individual users (or) user groups.

1.2.16 Conflicting requirements can be balanced

- Knowing the overall requirements of enterprise the DBA structures the system and provides resources.

1.3 EMERGENCE OF NOSQL

- NO SQL: A NoSQL (originally referring to "non-SQL" or "non-relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

1.3.1 Brief History of NoSQL Databases:

- 1998: Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database.
- 2000: Graph database Neo4j is launched
- 2004: Google BigTable is launched
- 2005: CouchDB is launched
- 2007: The research paper on Amazon Dynamo is released
- 2008: Facebooks open sources the Cassandra project
- 2009: The term NoSQL was reintroduced

1.3.2 RDBMS versus NoSQL

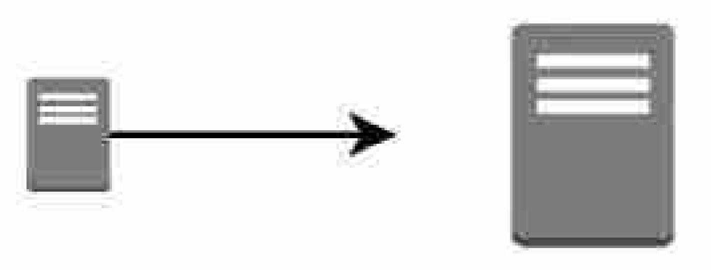
- Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data.

  Note: Polymorphic Data: A single column or field to contain data of different types.

- NoSQL Database is a non-relational Data Management System does not require a fixed schema. It avoids joins, and is easy to scale. The major purpose of using a NoSQL database is to process the huge data in distributed environment. NoSQL is used for Big data and Real-time Web Apps. For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.

- **Scalability: T**he ability to expand or contract the capacity of system resources in order to support the changing usage of application.

- **Vertical scaling:** Vertical scaling refers to increasing the processing power of a single server or cluster. Both relational and non-relational databases can scale up, but eventually, there will be a limit in terms of maximum processing power and throughput.

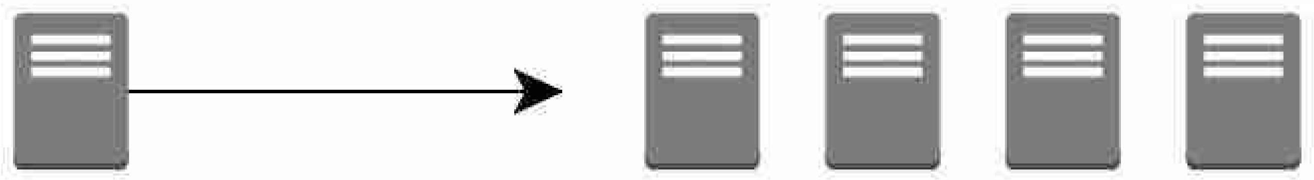


Advantage: No change is required for database infrastructure other than the hardware specifications of the machine running the database.

Disadvantages:

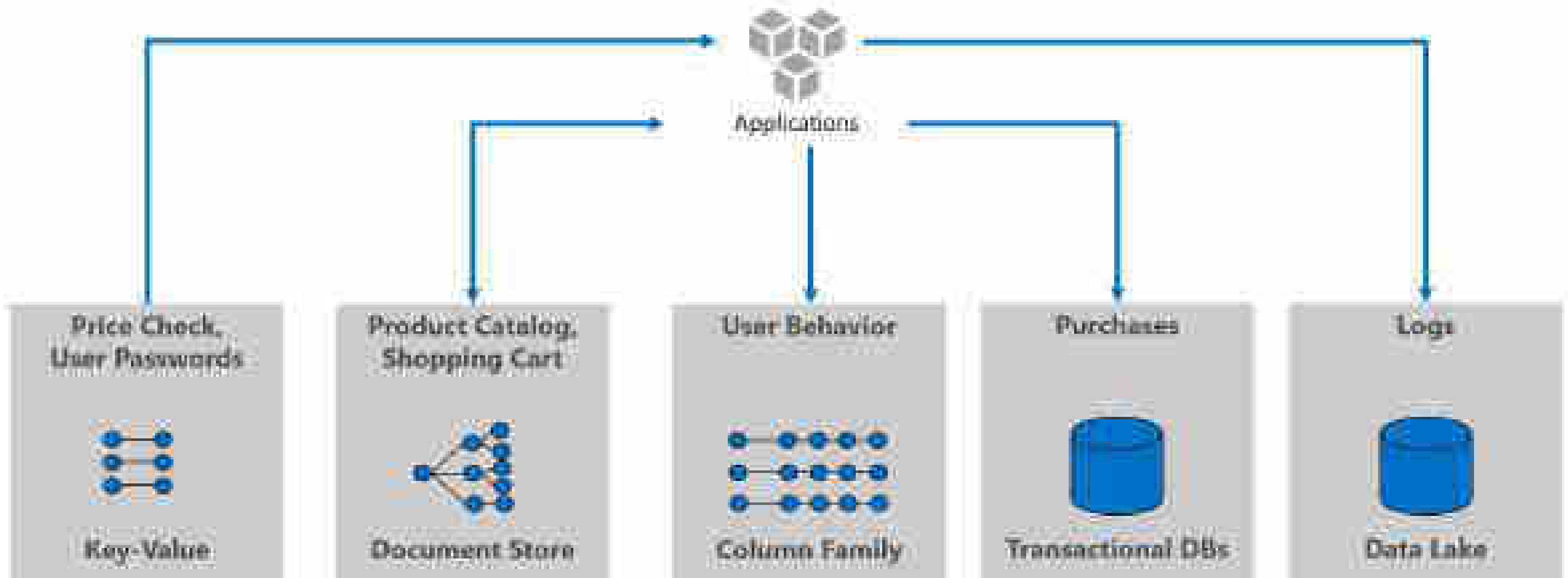
- More storage and processing power can be a lot more expensive.
- There is a physical limit on the amount of CPUs, memory, network interfaces, and hard-drives that can be used on a single machine.
- Requires a migration between hardware's, it could result in downtime or service disruption.

- Horizontal scaling: Horizontal scaling refers to add additional nodes to share the load. This is difficult with relational databases to distribute the database tables across the network. With non-relational databases, this is made simpler since collections are self-contained and not coupled relationally. This allows them to be distributed across nodes more simply, as queries do not have to "join" them together across nodes.



1.3.3 Polyglot Persistence: Polyglot persistence is a term that refers to using multiple data storage technologies for varying data storage needs across an application.

Example:



1.4 IMPEDANCE MISMATCH PROBLEM

- Impedance mismatch is the term used to refer to the problems that occurs due to differences between the database model and the programming language model.

Following problems may occur due to the impedance mismatch:

- The Type System of *Data Sub Language (SQL Queries)* is differs from the *Type System* of *Host Language (Programming Language)*.
  Note: Type system include
  Syntax: Set of rules
  Semantics: Meaningful representation of the rule
  Pragmatics: Practical approach

- Data type mismatch: The data types of different programming languages are differ.

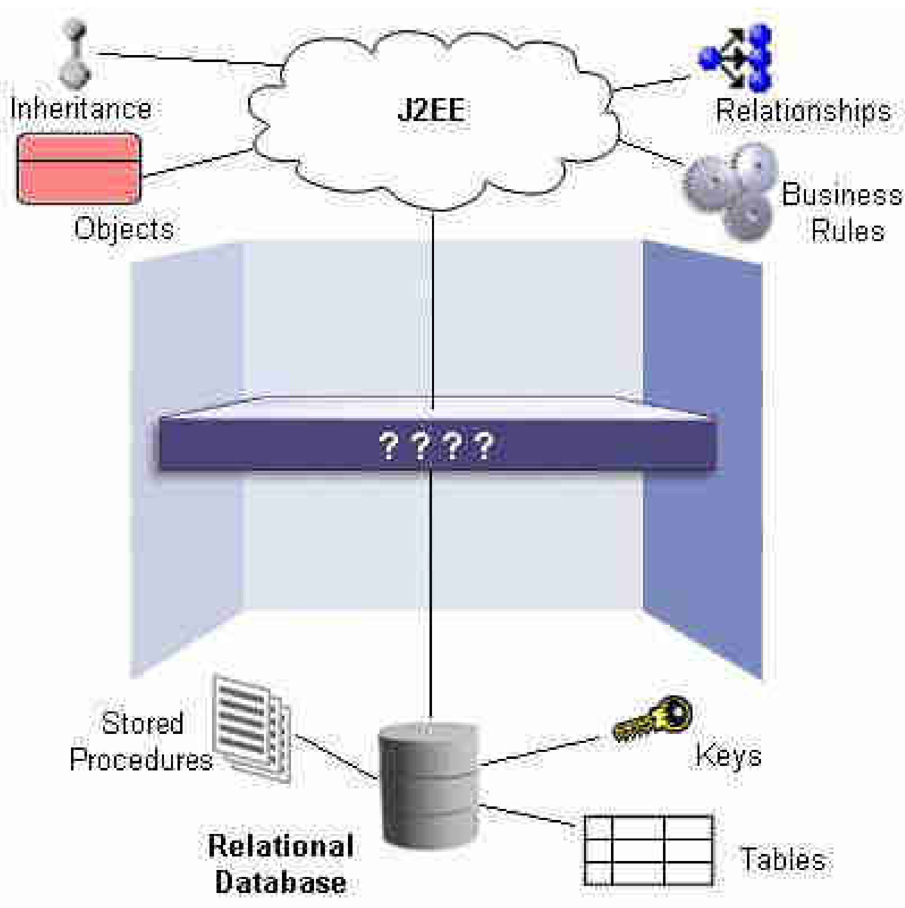
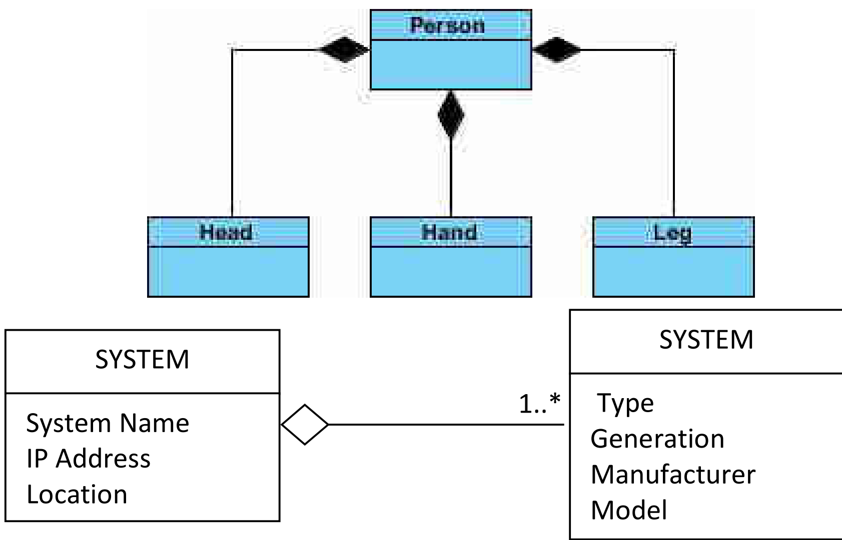


Fig. The Impedance mismatch between Relational Model and Object Oriented Model (The Type System of Relational Model differs with the Type System of Object Oriented Model).

CHAPTER 2
AGGREGATE DATA MODELS

2.1 AGGREGATE DATA MODELS VIMP

- Aggregation refers to the process by which entities are combined to form a single meaningful entity. The specific entities are combined because they do not make sense on their own.



- Structural constraints on relationships: We have two relationship constraints.

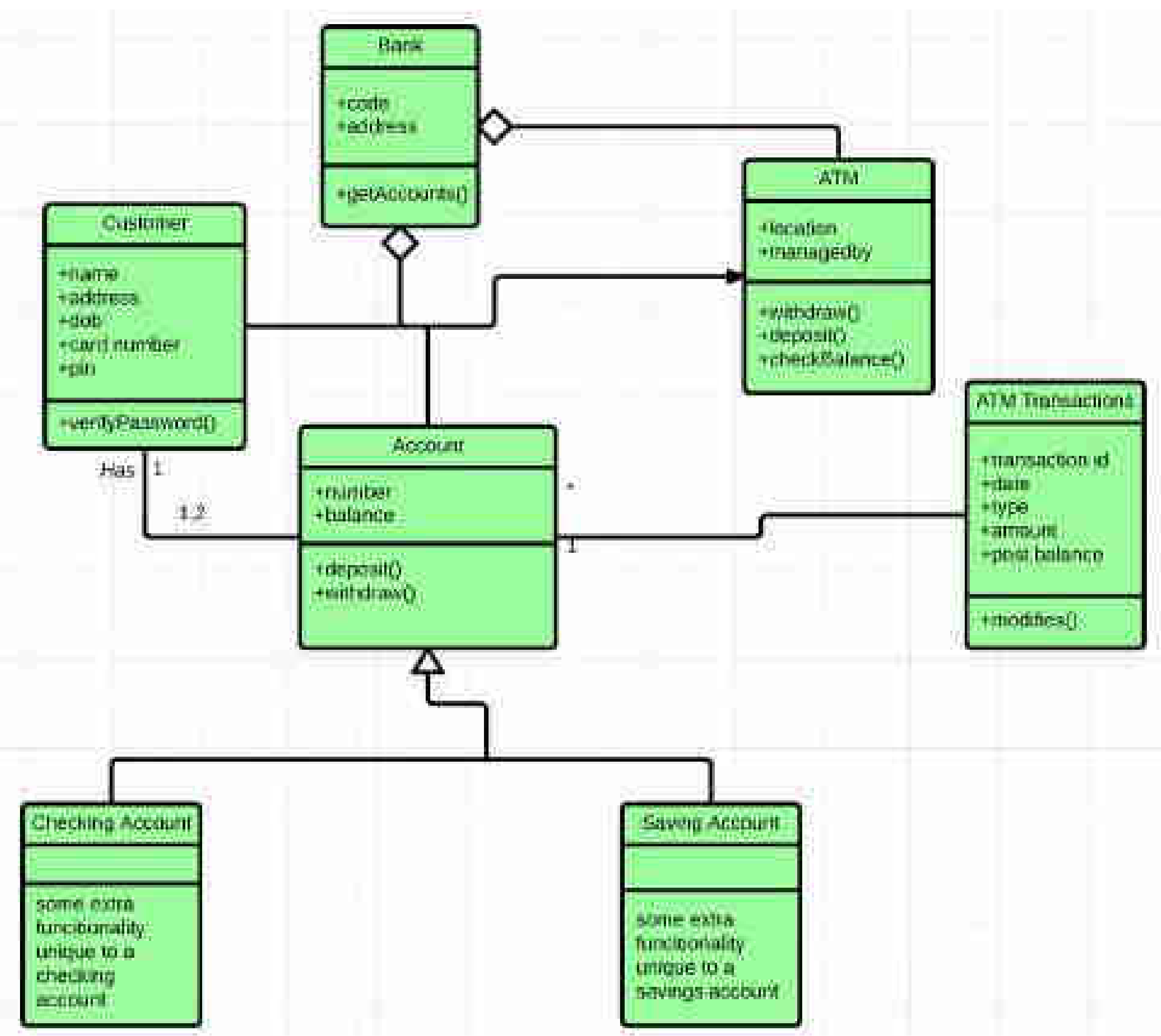
(a) Cardinality ratio constraint (1-1, 1-N, M-N)

(b) Participation constraint

1. Total participation (existence ependency): All the tuples of the participating entity relates with the tuples of related entity represented with double line.
2. Partial participation : Some tuples of the participating entity relates with all the tuples of related entity represented with single line.

- Note: Participation constraint is represented by (min,max) pair.
  If min value = 0 then the is patial paticipation.

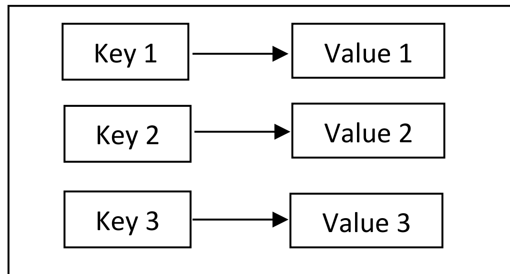
  If min value >= 0 then the is total paticipation.

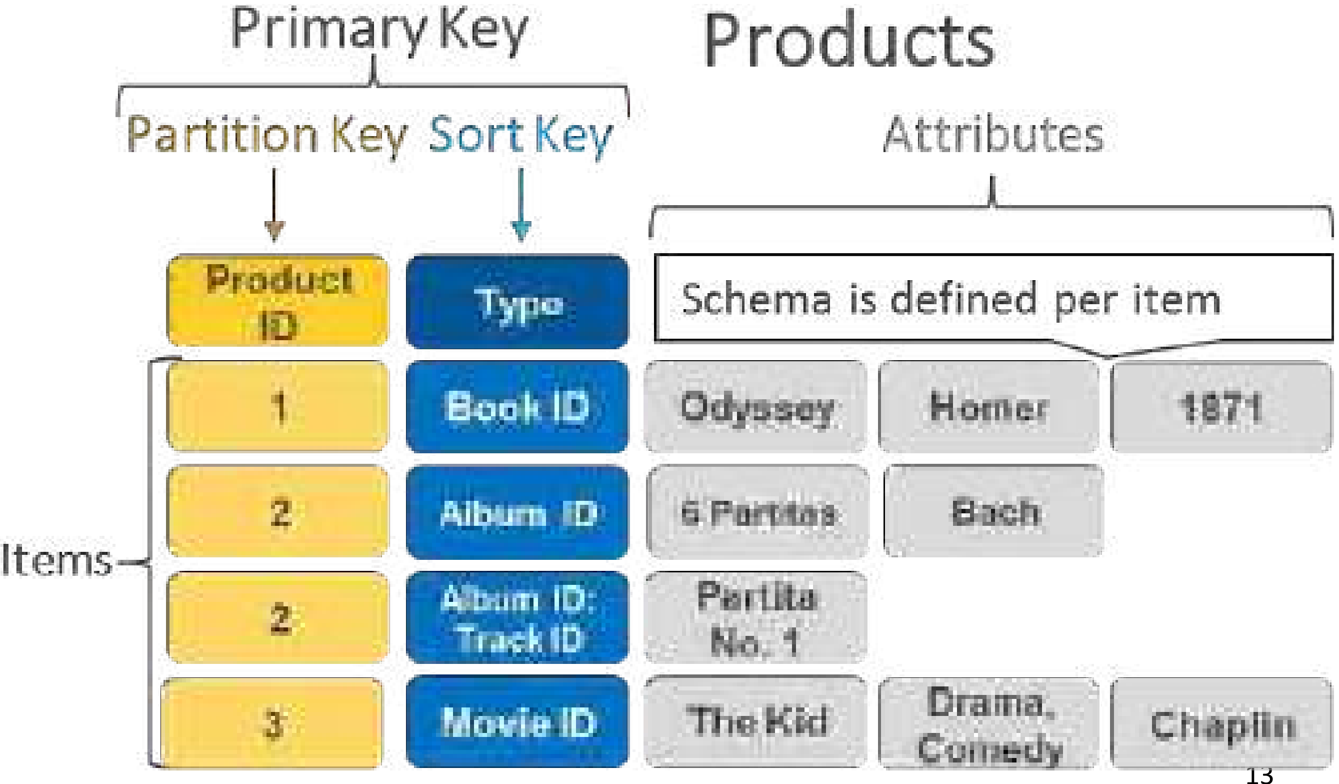


Example of Aggregation.

2.2 KEY-VALUE AND DOCUMENT DATA MODELS

- The data is fetched by a unique key or a number of unique keys to retrieve the associated value with each key.
- The values can be data types like strings and numbers or complex objects.
- Data is accessed (inserted, updated, and deleted) and queried based on the key to store/retrieve its value.
- Key-value databases use compact, efficient index structures to be able to quickly and reliably locate a value by its key, making them ideal for systems that need to be able to find and retrieve data in constant time.





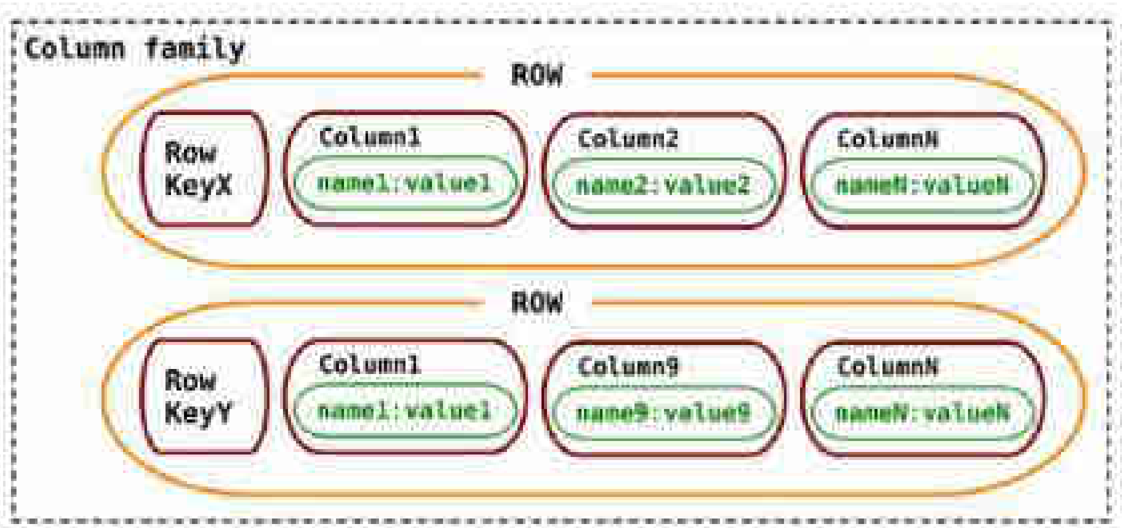
2.3 COLUMN-FAMILY STORES

- Column-family stores, such as allow you to store data with keys mapped to values and the values grouped into multiple column families, each column family being a map of data.

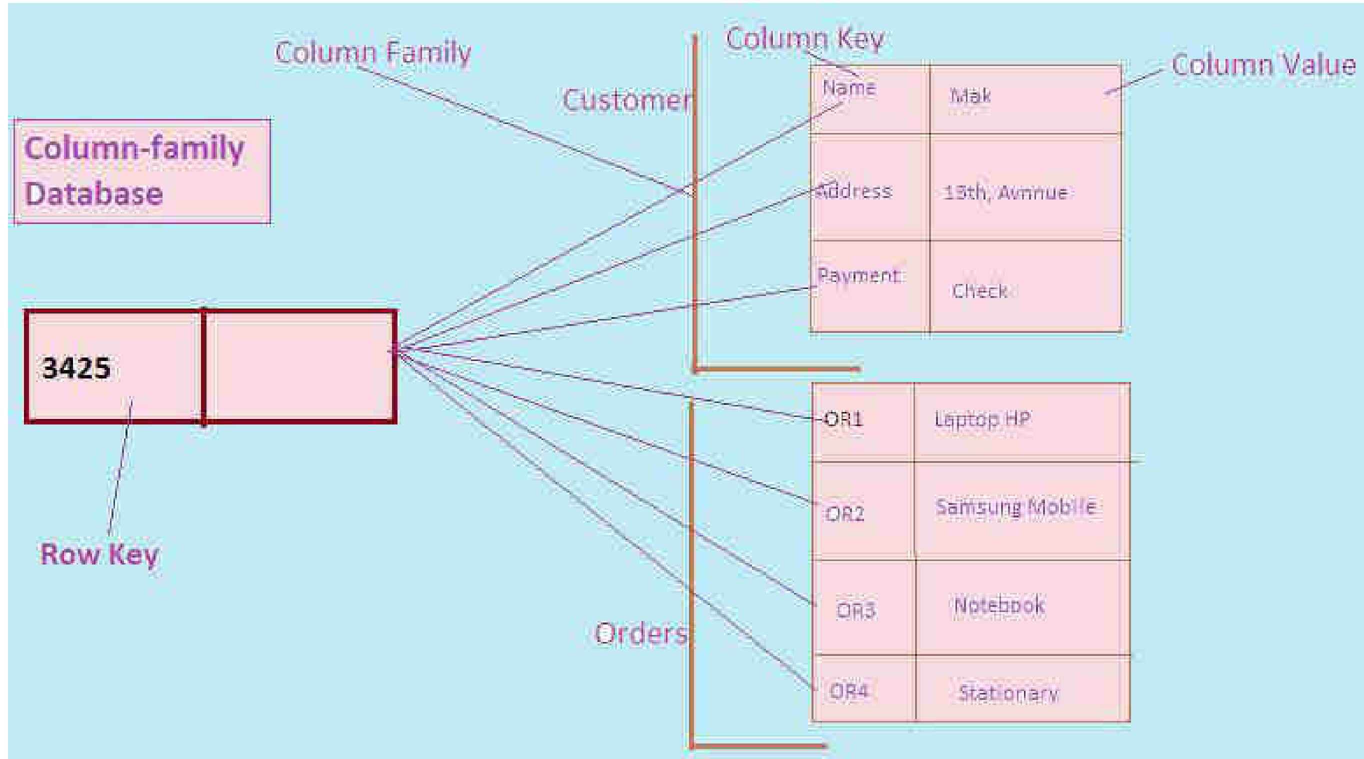
  Examples: Cassandra, HBase, Hypertable, and Amazon SimpleDB.

- Column-family databases store data in column families as rows that have many columns associated with a row key.

- Column families are groups of related data that is often accessed together. For a Customer, we would often access their profile information at the same time, but not their orders.



Cassandra's data model with column families.



Column family Structure of Database.

- **Skinny rows** have few columns with the same columns used across the many different rows. In this case, the column family defines a record type, each row is a record, and each column is a field.
- **A wide row** has many columns (perhaps thousands), with rows having very different columns. A wide column family models a list, with each column being one element in that list.

Example:

```
//column family
{
//row
"arjun-thatavarthy" : {
firstName: "arjun",
lastName: "thatavarthy",
lastVisit: "2012/12/12"
}
//row
"martin-fowler" : {
firstName: "martin",
lastName: "fowler",
location: "boston"
}
}
```

Note: We have the *pramod-sadalage* row and the *martin-fowler* row with different columns; both rows are part of the column family.

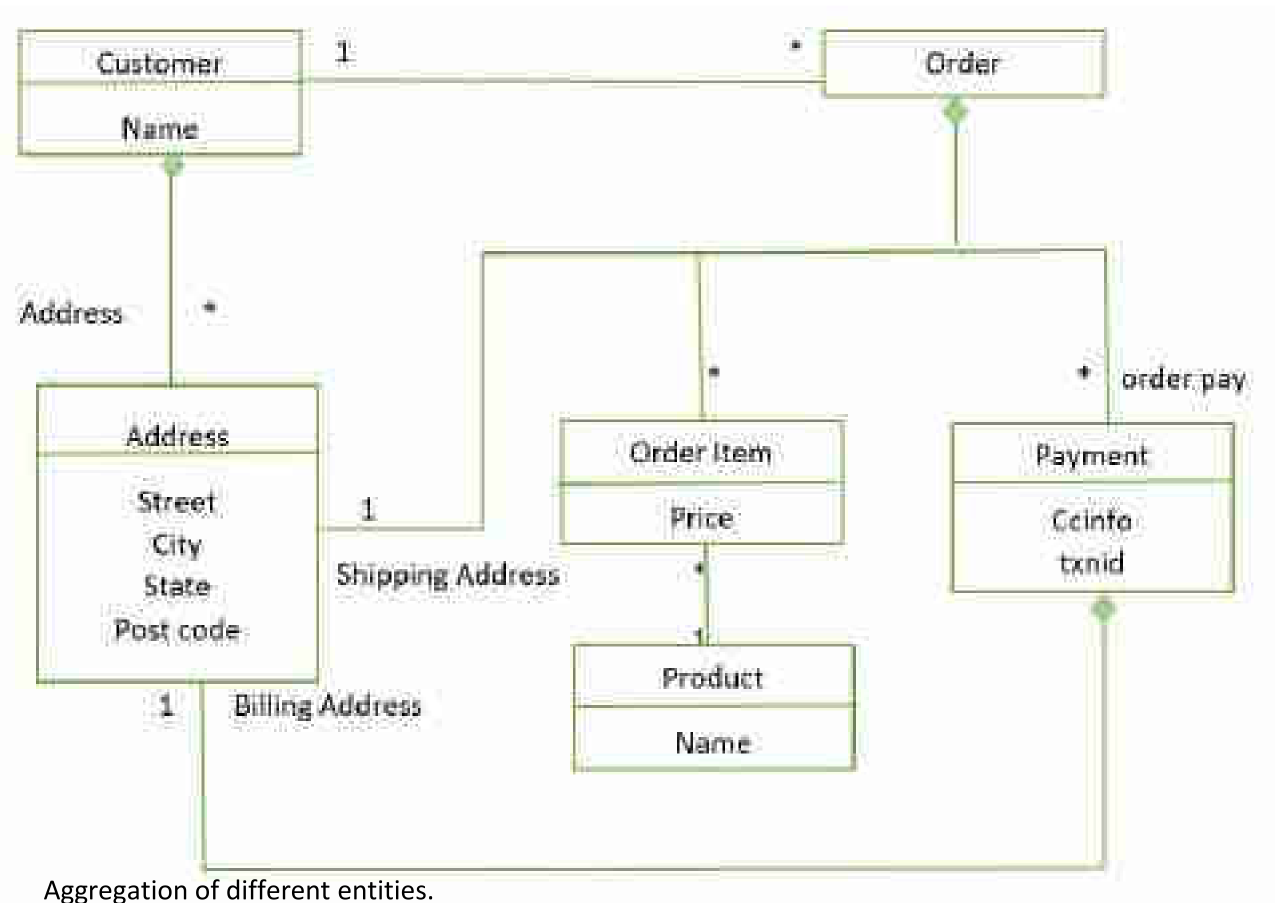
CHAPTER 3
MORE DETAILS ON DATA MODELS

3.1 RELATIONSHIPS

- Relationship: The logical connection between the documents.

- MonagoDb relations are
1. One to One
2. One to Many
3. Many to One
4. Many to Many

- Aggregation: The term aggregate means a collection of objects that we use to treat as a unit.



Aggregation of different entities.

3.2 GRAPH DATABASES

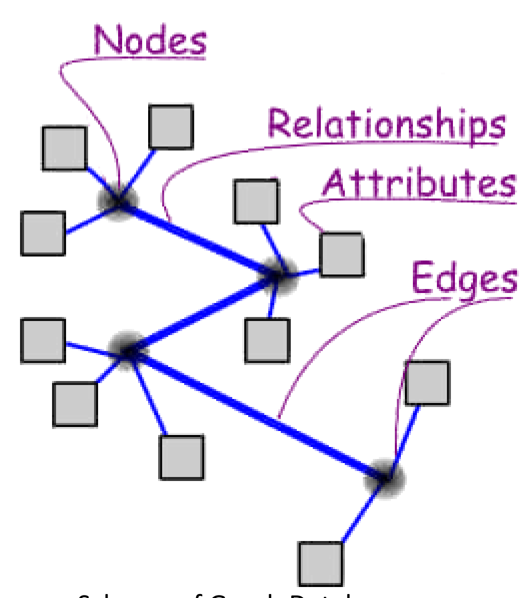
- A graph database is a type of database used to represent the data in the form of a graph.
- It has three components: nodes, relationships, and properties. These components are used to model the data.
- The concept of a Graph Database is based on the theory of graphs. It was introduced in the year 2000. They are commonly referred to NoSql databases as data is stored using nodes, relationships and properties instead of traditional databases. A graph database is very useful for heavily interconnected data.

The description of components are as follows:

- Nodes: represent the objects or instances. They are equivalent to a row in database. The node basically acts as a vertex in a graph. The nodes are grouped by applying a label to each member.

- Relationships: They are basically the edges in the graph. They have a specific direction, type and form patterns of the data. They basically establish relationship between nodes.

- Properties: They are the information associated with the nodes.
  E.g. Neo4j, Oracle NoSQL DB, Graph base etc.



Schema of Graph Databases.

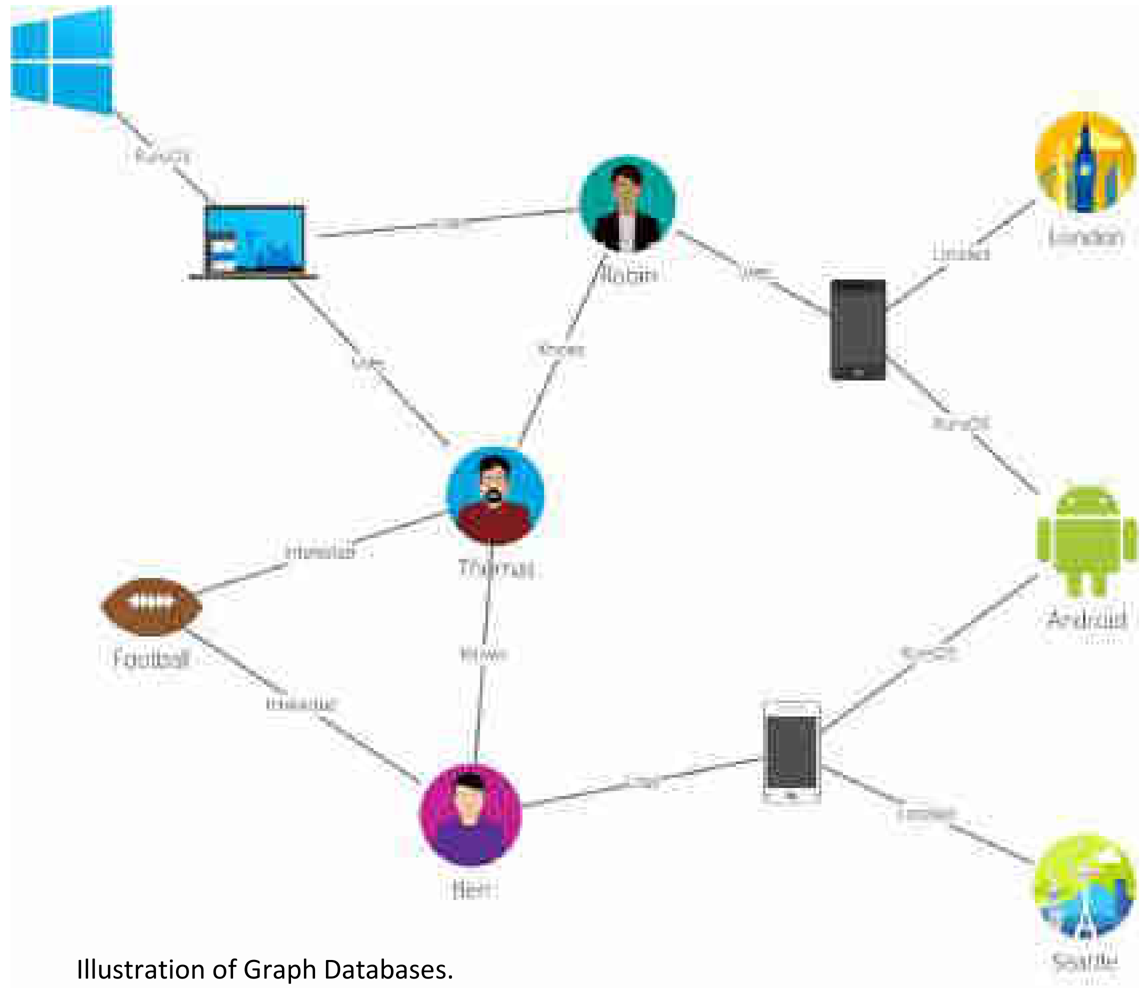


Illustration of Graph Databases.

When to Use Graph Database?

- Graph databases should be used for heavily interconnected data.
- It should be used when amount of data is larger and relationships are present.
- It can be used to represent the cohesive picture of the data.

How Graph and Graph Databases Work?

- Graph databases provide graph models. They allow users to perform traversal queries since data is connected. Graph algorithms are also applied to find patterns, paths and other relationships this enabling more analysis of the data.

Example of Graph Database:

- Recommendation engines in **E-Commerce** use graph databases to provide customers with accurate recommendations, updates about new products thus increasing sales and satisfying the customer's desires.
- **Social media companies** use graph databases to find the "friends of friends" or products that the user's friends like and send suggestions accordingly to user.

Advantages of Graph Database:

- Establishing the relationships with external sources as well
- No joins are required since relationships is already specified.
- Query is dependent on concrete relationships and not on the amount of data.
- It is flexible and agile.
- it is easy to manage the data in terms of graph.

Disadvantages of Graph Database:

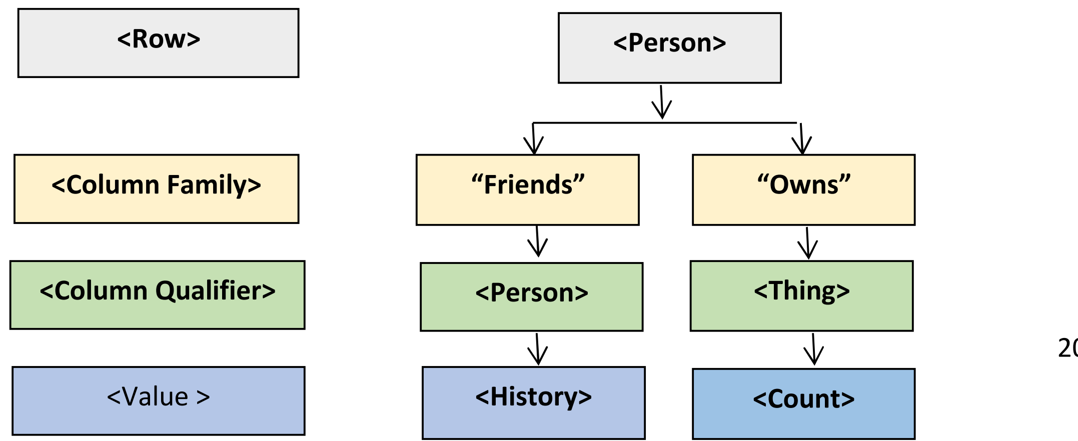
- Often for complex relationships speed becomes slower in searching.
- The query language is platform dependent.
- They are inappropriate for transactional data.
- It has smaller user base.

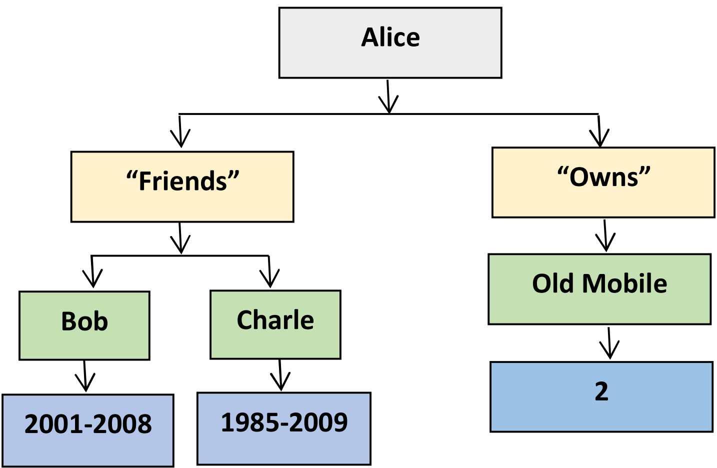
3.3 SCHEMALESS DB

- Traditional relational databases are well-defined, using a schema to describe tables, rows views, indexes, and relationships. By exerting a high degree of control, the database administrator can improve performance and prevent capture of inconsistent or incomplete data. In a SQL database, the schema is enforced by the Relational Database Management System (RDBMS) whenever data is written to disk.

- Schemaless databases are designed to store and query unstructured data, they do not require schemas used by relational databases. Although a schema can be applied at the application level, NoSQL databases retain all of your unstructured data in its original raw format.

- **Example of Schemaless Database:**





Schemaless Database:

- Does not require conforming to a rigid schema (database, schema, data types, tables, etc.) that one is required use through the life of a system.
- Does not enforce data type limitations on individual values pertaining to one single column type.
- Models the business usage and not a database schema, application, or product.
- Can store structured and unstructured data.
- Eliminates the need to introduce additional layers (ORM layer) to abstract the relational model and expose it in an object-oriented format.

Advantages of Schemaless Databases:

1.Greater flexibility over data types

- By operating without a schema, schemeless databases can store, retrieve, and query any data type, perfect for big data analytics and similar operations that are powered by unstructured data. Relational databases apply rigid schema rules to data, limiting what can be stored.

2.No pre-defined database schemas

The lack of schema means that your NoSQL database can accept any data type, including those that you do not yet use.

3.No data truncation

- A schemaless database makes almost no changes to your data, each item is saved in its own document with a partial schema, leaving the raw information untouched. This means that every detail is always available and nothing is stripped to match the current schema. This is particularly valuable if your analytics needs to change at some point in the future.

4.Suitable for real-time analytics functions

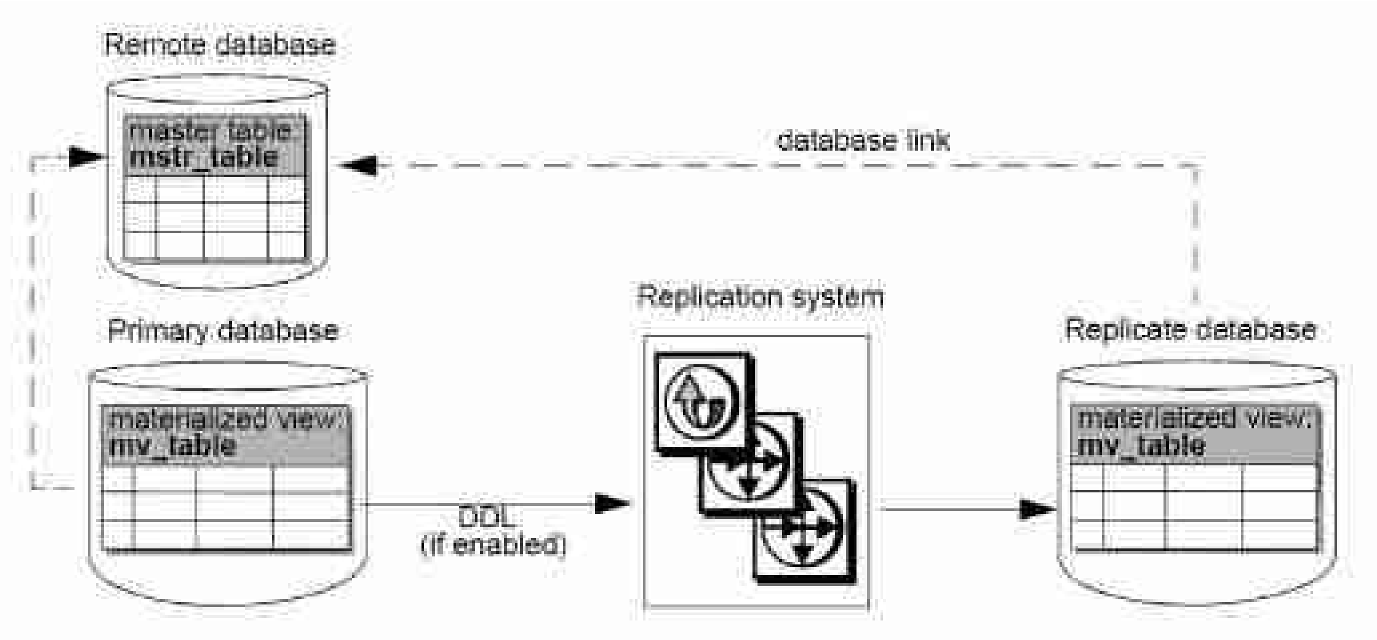
- With the ability to process unstructured data, applications built on NoSQL databases are better able to process real-time data, such as readings and measurements from IoT sensors.
- Schemaless databases are also ideal for use with machine learning and artificial intelligence operations, helping to accelerate automated actions in your business.

5.Enhanced scalability and flexibility

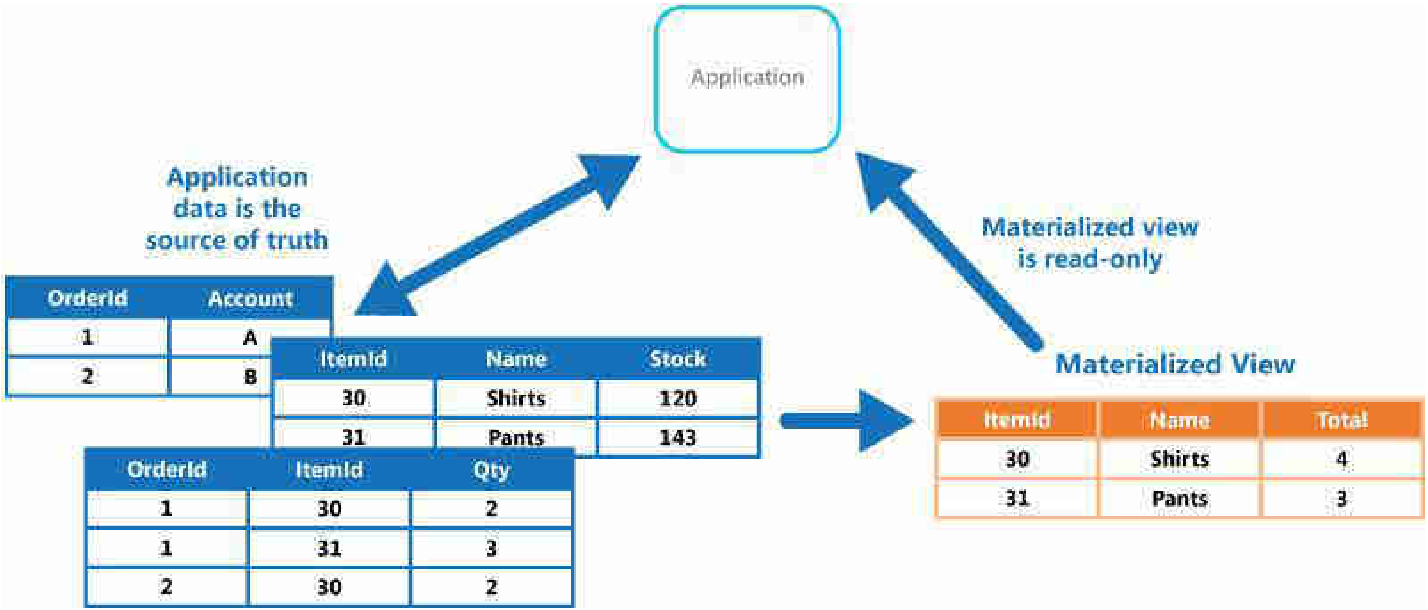
- With NoSQL, you can use whichever data model is best suited to the job. Graph databases allow you to view relationships between data points, or you can use traditional wide table views with an exceptionally large number of columns. You can query, report, and model information however you choose. And as your requirements grow, you can keep adding nodes to increase capacity and power.

3.4. MATERIALIZED VIEWS

- A materialized view is a replica of a target master from a single point in time.
- The master can be either a master table at a master site or a master materialized view at a materialized view site.
- Whereas in multi master replication tables are continuously updated by other master sites, materialized views are updated from one or more masters through individual batch updates, known as a **refreshes**, from a single master site or master



Example of Materialized View.



Example of Materialized View.

3.4.1 Advantages of Materialized Views

You can use materialized views to achieve one or more of the following goals

3.4.1.1 Ease Network Loads

- Reduce network loads that is use of materialized views distribute corporate database to regional sites.
- Instead of the entire company accessing a single database server, user load is distributed across multiple database servers.
- Materialized views are updated through an efficient batch process from a single master site or master materialized view site.
- They have lower network requirements and dependencies than multi master replication because of the point in time nature of materialized view replication.
- Whereas multi master replication requires constant communication over the network, materialized view replication requires only periodic refreshes.
- Materialized views increase data availability by providing local access to the target data.
- Materialized views greatly enhance the performance and reliability of your replicated database.

3.4.1.2 Create a Mass Deployment Environment

- Deployment templates create a materialized view environment locally.
- This technology enables efficient and effective use of database infrastructure to hundreds or thousands of users.

3.4.1.3 Enable Data Subsetting

- Materialized views allow you to replicate data based on column and row level subsetting, while multimaster replication requires replication of the entire table.
- Data subsetting enables you to replicate information that relates only to a particular site. For example, if you have a regional sales office, then you might replicate only the data that is needed in that region, thereby cutting down on unnecessary network traffic.

3.4.1.4 Enable Disconnected Computing

- Materialized views do not require a dedicated network connection. Though you have the option of automating the refresh process by scheduling a job, you can manually refresh your materialized view on-demand, which is an ideal solution for sales applications running on a laptop. For example, a developer can integrate the replication management API for refresh on-demand into the sales application. When the salesperson has completed the day's orders, the salesperson simply dials up the network and uses the integrated mechanism to refresh the database, thus transferring the orders to the main office.

3.4.2 Types of Materialized Views

3.4.2.1 Read-Only Materialized Views: Materialized view read-only during creation.

3.4.2.2 Updatable Materialized Views: Materialized view updatable during creation.

3.4.2.3 Writeable Materialized Views: Created using the FOR UPDATE clause but the updates are confined to local site. Users can perform DML operations on a writeable materialized view, when materialized view is refreshed, then these changes are not updated in master site. Writeable materialized views are typically allowed wherever fast-refreshable read-only materialized views are allowed.

3.4.3 Required Privileges for Materialized View Operations

Three distinct types of users perform operations on materialized views:

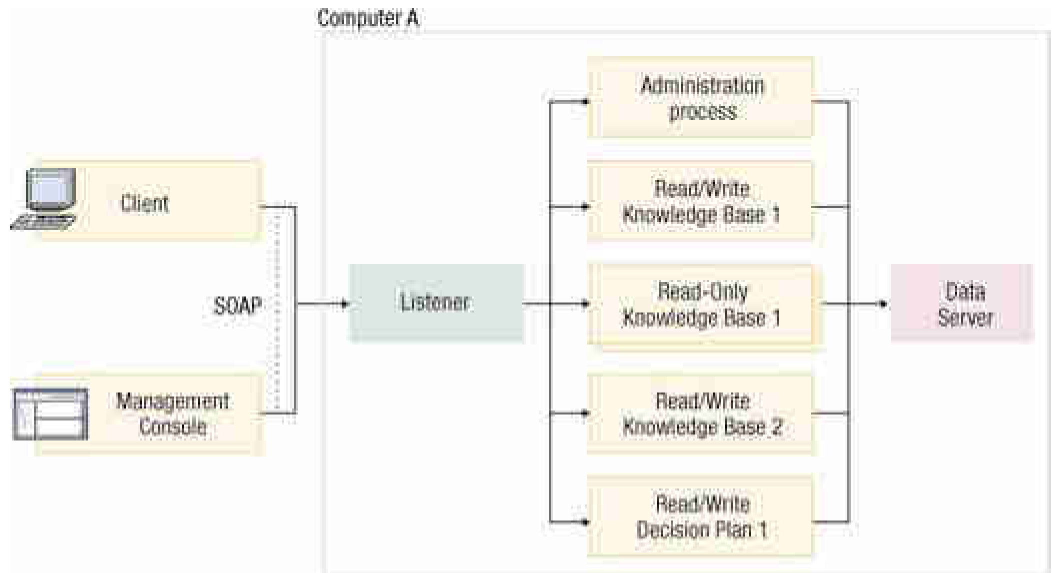
- Creator: The user who creates the materialized view
- Refresher: The user who refreshes the materialized view
- Owner: The user who owns the materialized view. The materialized view resides in this user's schema.

CHAPTER 4
DISTRIBUTION MODELS

4.1. SINGLE SERVER

- In the single server configuration, all server components run on the same computer.

- A single server configuration is useful on multiprocessor computers and in small-scale or development environments.



Single server configuration

4.1.1 Client Responsibilities

- Client interact with sever with Custom Application.

- The applications communicate with the listener that runs on the server by using the SOAP protocol.
  NOTE: SOAP is a messaging protocol specification for exchanging structured information in the implementation of web services in computer networks.

- The listener forwards the request to the appropriate server component for handling.

- Administration requests are forwarded to the administration process, and runtime requests on a specific knowledge base or decision plan are routed to the read/write or read-only instances, depending on the request.

- Read/write requests are forwarded to read/write instances, and read-only requests are forwarded to read-only instances.

4.1.2 Server responsibilities:

- The function of a computer server is to store, retrieve and send or "serve" files and data to other computers on its network.

- Monitor whether **Load Balancing** is to be performed by the listener.

- Load balancing lets evenly distribute network traffic to prevent failure caused by overloading a particular resource. This strategy improves the performance and availability of applications, websites, databases, and other computing resources. It also helps process user requests quickly and accurately.

- Monitor the **Route Requests**. The routing algorithm that provides the best path from the source to the destination.

4.2. SHARDING

- Database partitioning in which a large Database is divided or partitioned into smaller data, also known as shards. These shards are not only smaller, but also faster and hence easily manageable.

4.2.1 Need for Sharding

- Consider a very large database whose sharding has not been done. For example, let’s take a database of a college in which all the student’s record the whole college are maintained in a single database. So, it would contain very large number of data, say 100, 000 records.
- Now when we need to find a student from this database, each time around 100, 000 transactions has to be done to find the student, which is very costly.
- Now consider the same college students records, divided into smaller data shards based on years. Now each data shard will have around 1000-5000 students records only. So not only the database became much more manageable, but also the transaction cost of each time also reduces by a huge factor, which is achieved by Sharding. Hence this is why Sharding is needed.

4.2.2 Features of Sharding

- Sharding makes the database smaller
- Sharding makes the database faster
- Sharding makes the database much more easily manageable
- Sharding can be a complex operation sometimes
- Sharding reduces the transaction cost of the database

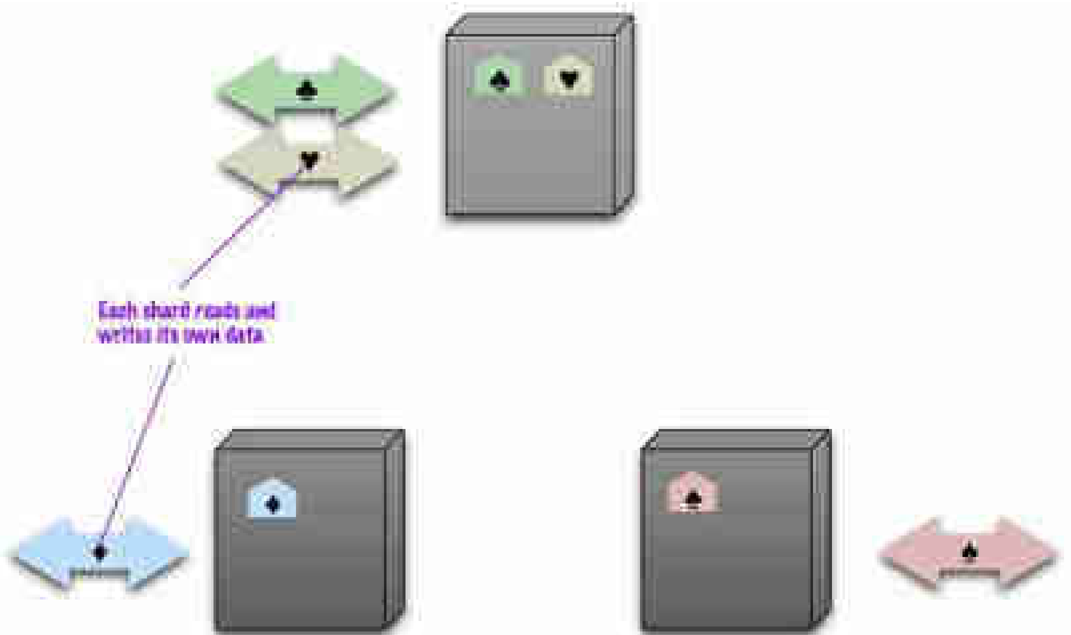
Employee		
Empno	Ename	Salary
1	John	3400
2	Martin	4300
3	Elica	4500
4	Miller	3400

Horizontal Sharding : A horizontal sharding is a subset of the tuples in that relation. The tuples that belong to the horizontal sharding are specified by a condition on one or more attributes of the relation.

Employee		
Empno	Ename	Salary
1	John	3400
2	Martin	4300

Vertical Sharding: Vertical sharding divides a relation "vertically" by columns. vertical sharding of a relation keeps only certain attributes of the relation.

Employee	
Empno	Ename
1	John
2	Martin
3	Elica
4	Miller



Sharding puts different data on separate nodes, each of which does its own reads and writes.

4.3. MASTER-SLAVE REPLICATION

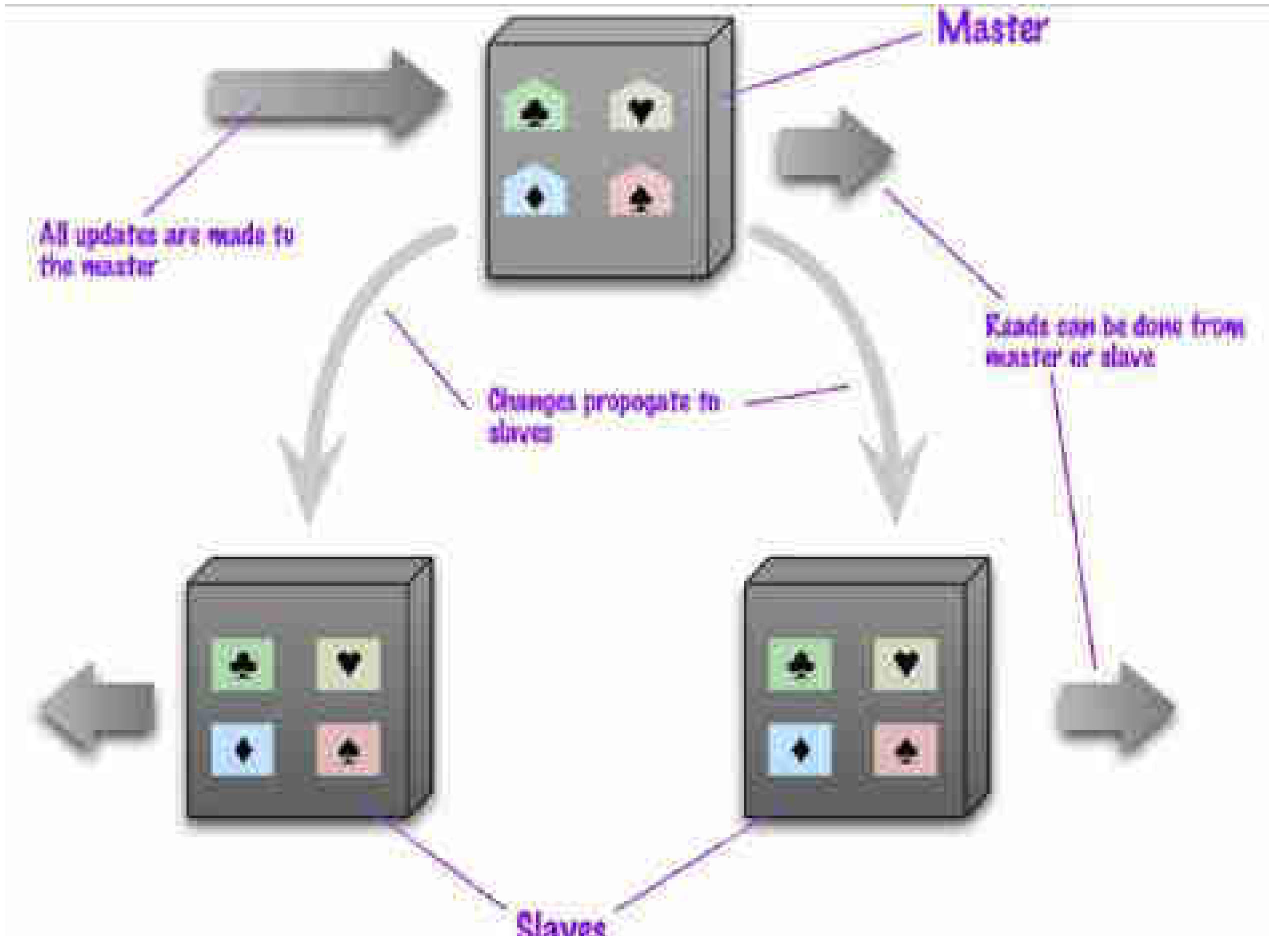
- In master-slave distribution data can be replicated across multiple nodes.
- One node is designated as the master, or primary. This master is the authoritative source for the data and is usually responsible for processing any updates to that data.
- The other nodes are slaves, or secondaries.
- The changes made at maser node is replicated in slave node.
- Master-slave replication is most helpful for scaling when you have a read-intensive dataset.
- Horizontal scaling handle more read requests by adding more slave nodes and ensuring that all read requests are routed to the slaves.

Advantages:

- Master-slave replication is **read resilience** that is if master fail, the slaves can still handle read requests.
- Master speed up recovery after a failure of by appointing new master very quickly.

Disadvantages:

- The failure of the master does eliminate the ability to handle writes until either the master is restored, or a new master is appointed.
- All read and write traffic can go to the master while the slave acts as a hot backup.

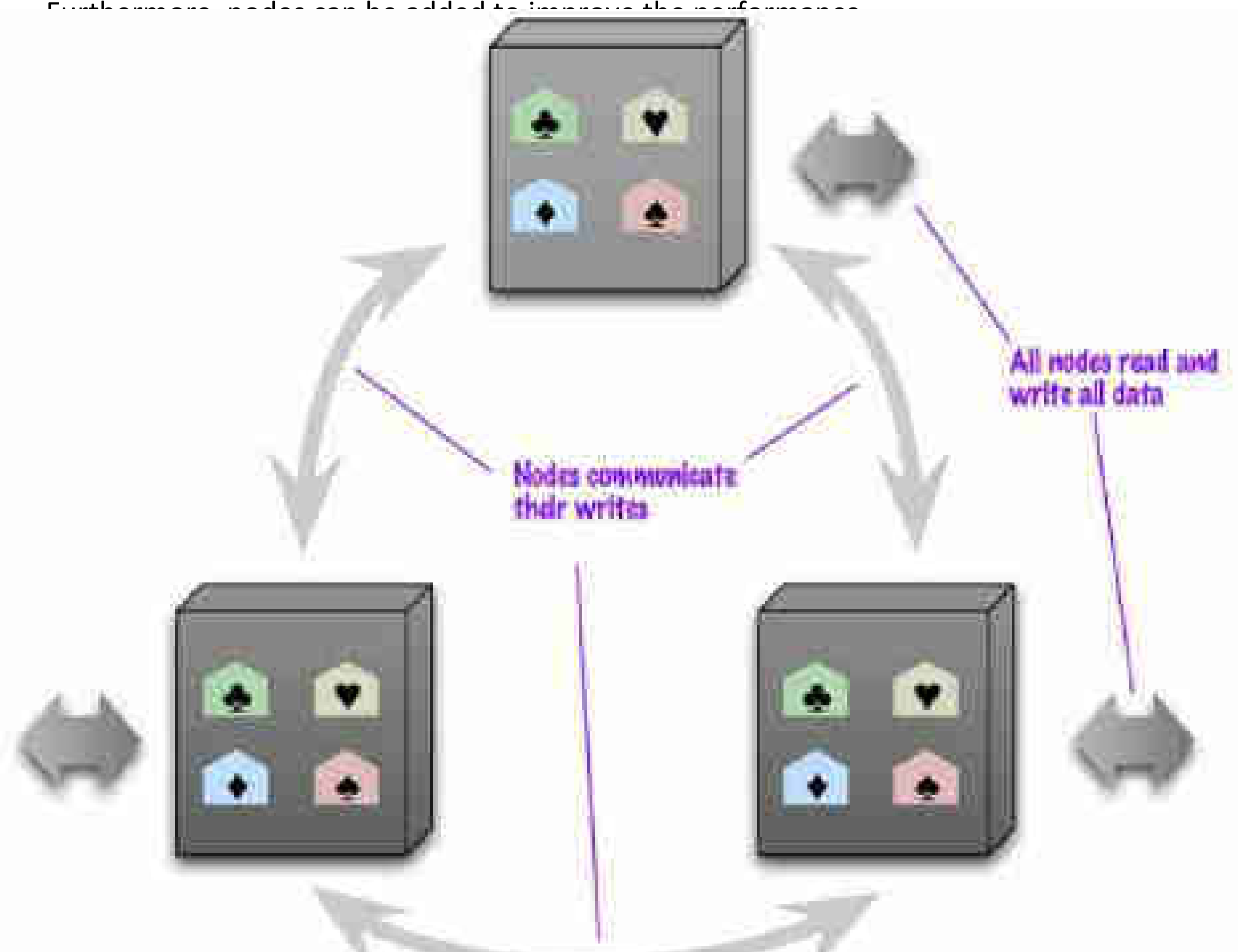


Data is replicated from master to slaves. The master services all writes; reads may come from either master or slaves.

4.4. PEER-TO-PEER REPLICATION

- Peer-to-peer replication provides a scale-out and high-availability solution by maintaining copies of data across multiple server instances, also referred to as nodes.

- With a peer-to-peer replication cluster override node failures without losing access to data. Furthermore, nodes can be added to improve the performance.



Peer-to-peer replication has all nodes applying reads and writes to all the data.

Advantages:

- Reads are spread across multiple nodes. This enables performance to remain consistent as reads increase.

- If one of the nodes in the system fails, an application layer can redirect the writes for that node to another node. This maintains availability.

- If a node requires maintenance or the whole system requires an upgrade, each node can be taken offline and added back to the system without affecting the availability of the application.

Disadvantage:

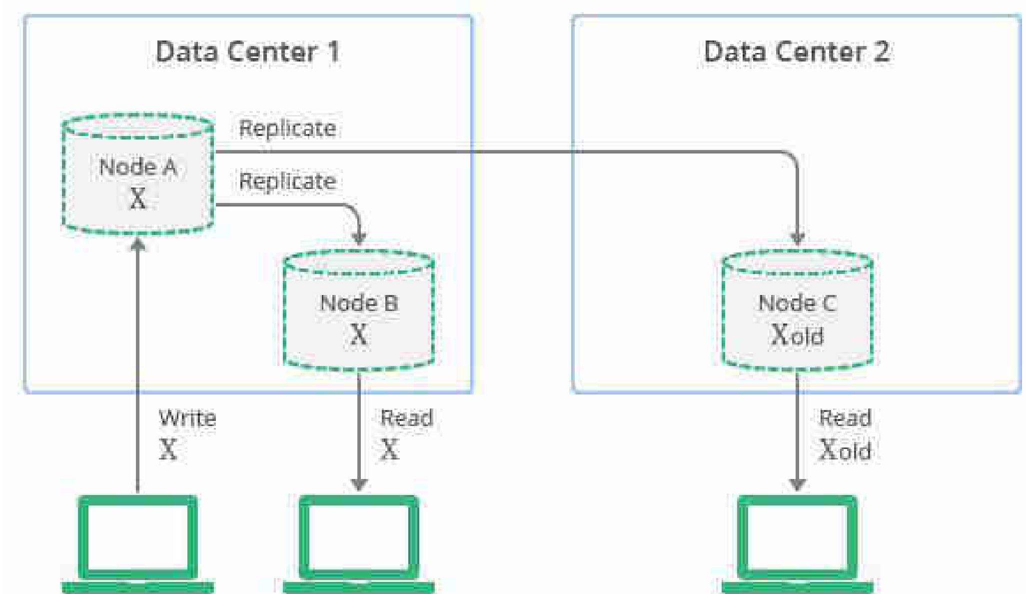
- If a row is modified at more than one node, it can cause a conflict or even a lost update when the row is propagated to other nodes.

- There is always some latency involved when changes are replicated. For applications that require the latest change to be seen immediately, dynamically load balancing the application across multiple nodes can be problematic.

CHAPTER 5
CONSISTENCY

5.1 READ CONSISTENCY

- **Read Consistency:** A read operation of data item / object must wait until the write commits before being able to read the new version.



read_TS(x):
~~100~~
~~101~~
102

write_TS(x):
~~100~~
~~101~~
102

x:	
~~10~~	
~~11~~ xi	First Version
~~12~~ xi+1	Second Version
~~13~~ xi+2	Third Version

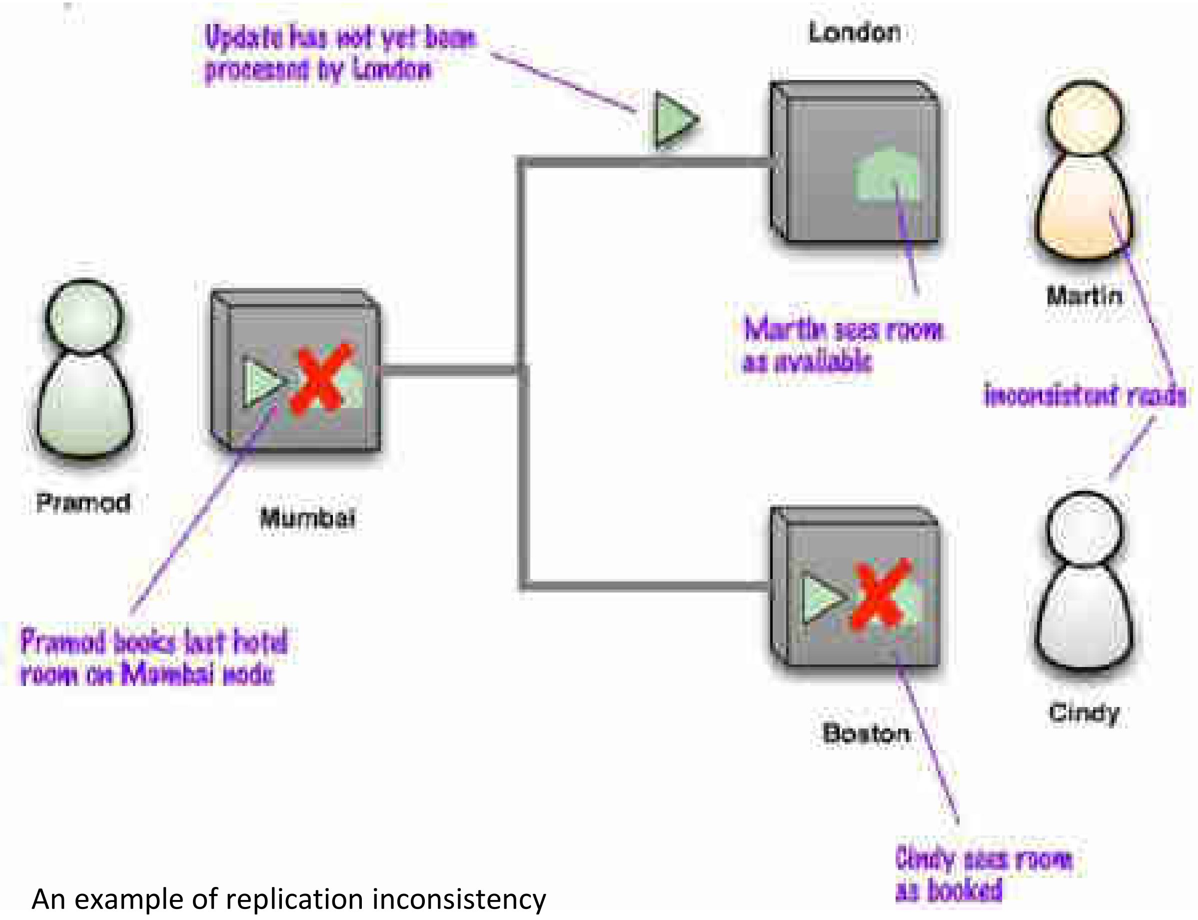
TS(T1)=100 T1	
read_item(x);	10
x:=x+1;	11
write_item(x);	11
commit;	

TS(T2)=101 T2	
read_item(xi);	11
xi:=xi+1;	12
write_item(xi);	12
commit;	

TS(T3)=102 T3	
read_item (xi+1);	12
x:=(xi+1)+1;	13
write_item((xi+1)+1);	13
commit;	

5.2 UPDATE CONSISTENCY

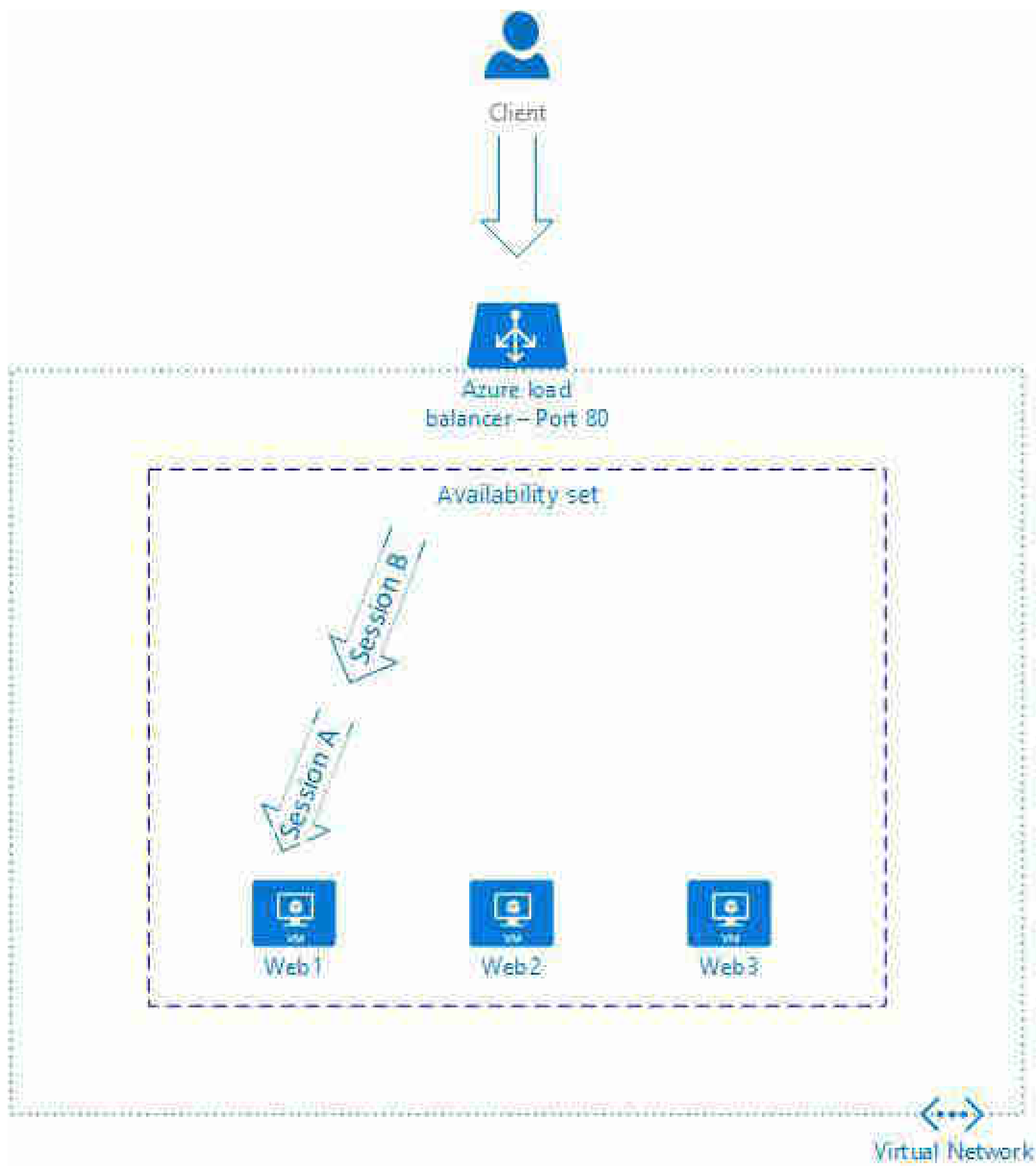
Update Consistency: The database must reflect the latest changes**.**



An example of replication inconsistency

Solution for replication inconsistency:

- **Read your- writes consistency**: The successful updation is visible to other reads.
- **Sticky Session:** When a client starts a session on one of your web servers, session stays on that specific server.

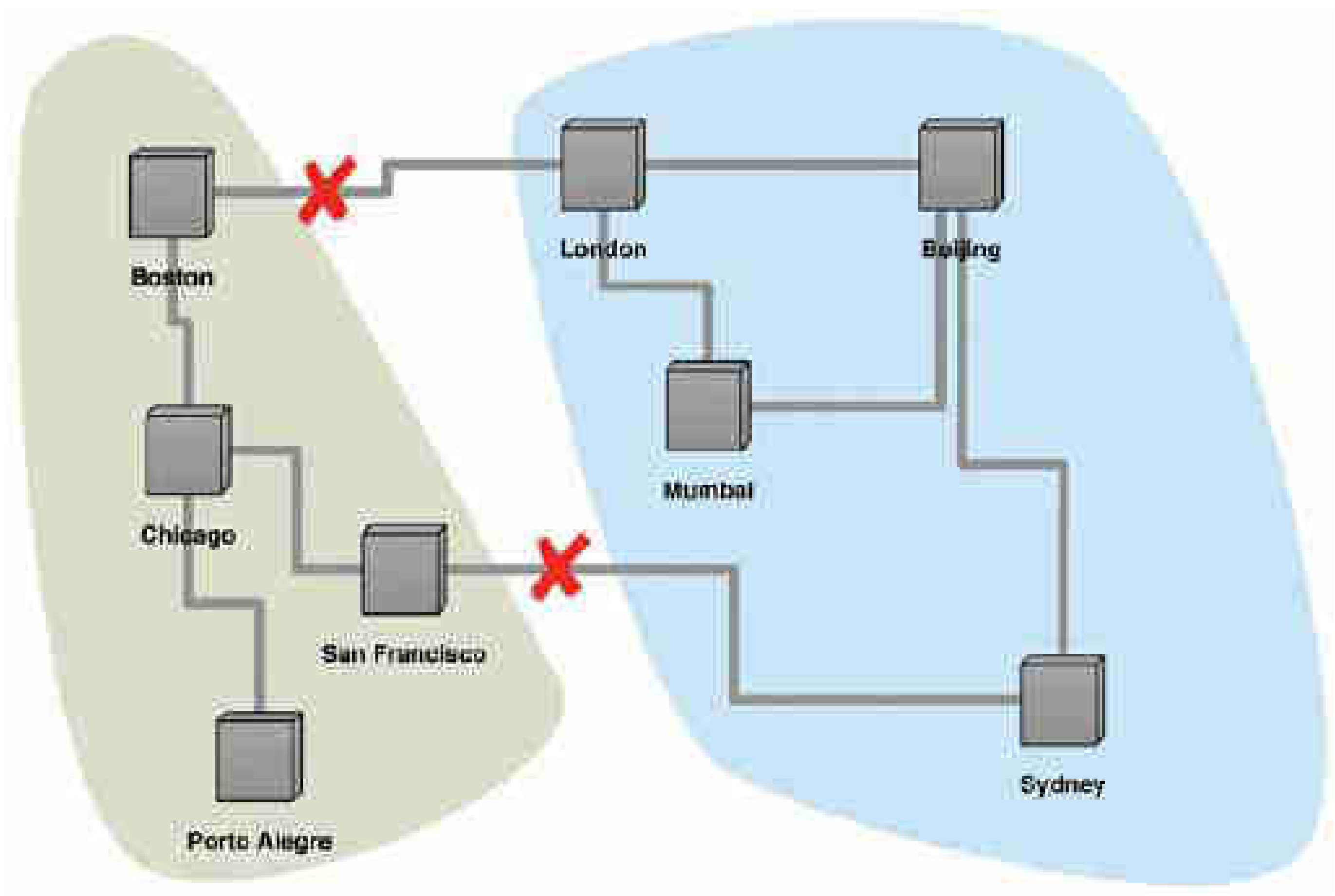


- **Version Stamps:** Follows Multiversion Concurrency Control Protocol.

5.3. RELAXING CONSISTENCY

- By relaxing consistency, distributed systems can be highly available and durable. It's possible for data to be inconsistent; a query might return old or stale data.

- Relaxing consistency is the phenomenon described as being eventually consistent. Over time, data that is spread across storage nodes will replicate and become consistent.



With two breaks in the communication lines, the network partitions into two groups.

5.3.1 The CAP Theorem

- The CAP theorem applies a similar type of logic to distributed systems—namely, that a distributed system can deliver only two of three desired characteristics: ***Consistency, Availability,*** and ***Partition tolerance*** (the '**C**,' '**A**' and '**P**' in CAP).

- **Consistency:** Consistency means that all clients see the same data at the same time, no matter which node they connect to. For this to happen, whenever data is written to one node, it must be instantly forwarded or replicated to all the other nodes in the system before the write is deemed 'successful.'

- **Availability:** Availability means that that any client making a request for data gets a response, even if one or more nodes are down. Another way to state this—all working nodes in the distributed system return a valid response for any request, without exception.

- **Partition tolerance:** A *partition* is a communications break within a distributed system—a lost or temporarily delayed connection between two nodes. Partition tolerance means that the cluster must continue to work despite any number of communication breakdowns between nodes in the system.

Today, NoSQL databases are classified based on the two CAP characteristics they support:

- **CP database:** A CP database delivers consistency and partition tolerance at the expense of availability. When a partition occurs between any two nodes, the system has to **shut down the non-consistent node** (i.e., make it unavailable) until the partition is resolved.

- **AP database:** An AP database delivers availability and partition tolerance at the expense of consistency. When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an **older version of data** than others. (When the partition is resolved, the AP databases typically resync the nodes to repair all inconsistencies in the system.)

- **CA database:** A CA database delivers consistency and availability across all nodes. It can't do this if there is a partition between any two nodes in the system, however, and therefore can't deliver fault tolerance.

MongoDB CRUD OPERATIONS VVIMP

TO SHOW THE DATABASE

> show dbs

admin 0.000GB

config 0.000GB

local 0.000GB

TO CREATE THE DATABASE

> use Employee

switched to db Employee

TO CREATE COLLECTION

> db.createCollection("EmployeeDetails");

{ "ok" : 1 }

TO INSERT DATA INTO COLLECTION

> db.EmployeeDetails.insert({eno:'1',ename:'kiran',esal:'10000'});

WriteResult({ "nInserted" : 1 })

> db.EmployeeDetails.insert({eno:'2',ename:'samrat',esal:'12000'});

WriteResult({ "nInserted" : 1 })

> db.EmployeeDetails.insert({eno:'3',ename:'tulasi',esal:'13000'});

WriteResult({ "nInserted" : 1 })

TO DISPLAY DATA FROM COLLECTION

> db.EmployeeDetails.find().pretty();

{

"_id" : ObjectId("628840622f3012f1bb3bfc46"),

"eno" : "1",

```
    "ename" : "kiran",
    "esal" : "10000"
}
{
    "_id" : ObjectId("628840dd2f3012f1bb3bfc47"),
    "eno" : "2",
    "ename" : "samrat",
    "esal" : "12000"
}
{
    "_id" : ObjectId("628841532f3012f1bb3bfc48"),
    "eno" : "3",
    "ename" : "tulasi",
    "esal" : "13000"
}
```

BATCH INSERTION

db.EmployeeDetails.insertMany([{eno:'4',ename:'raju',esal:'4000'},{eno:'5',ename:'anil',esal:'5000'},{ eno:'4',ename:'padma',esal:'4500'}]);

```
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("628acca167dc6828a6781837"),
        ObjectId("628acca167dc6828a6781838"),
        ObjectId("628acca167dc6828a6781839")
    ]
```

}

DISPLAY DATA

> db.EmployeeDetails.find();

```
{ "_id" : ObjectId("628acca167dc6828a6781837"), "eno" : "4", "ename" : "raju", "esal" : "4000" }

{ "_id" : ObjectId("628acca167dc6828a6781838"), "eno" : "5", "ename" : "anil", "esal" : "5000" }

{ "_id" : ObjectId("628acca167dc6828a6781839"), "eno" : "4", "ename" : "padma", "esal" : "4500" }

>
```

> db.EmployeeDetails.find().pretty();

```
{
    "_id" : ObjectId("628acca167dc6828a6781837"),
    "eno" : "4",
    "ename" : "raju",
    "esal" : "4000"
}
{
    "_id" : ObjectId("628acca167dc6828a6781838"),
    "eno" : "5",
    "ename" : "anil",
    "esal" : "5000"
}
{
    "_id" : ObjectId("628acca167dc6828a6781839"),
    "eno" : "4",
    "ename" : "padma",
```

"esal" : "4500"

}

>

UPDATE DABASE

> db.EmployeeDetails.update({'eno':'4'},{$set:{'ename':"cherry"}});

> db.EmployeeDetails.find().pretty();

```
{
	"_id" : ObjectId("628acca167dc6828a6781837"),
	"eno" : "4",
	"ename" : "cherry",
	"esal" : "4000"
}
{
	"_id" : ObjectId("628acca167dc6828a6781838"),
	"eno" : "5",
	"ename" : "anil",
	"esal" : "5000"
}
{
	"_id" : ObjectId("628acca167dc6828a6781839"),
	"eno" : "4",
	"ename" : "padma",
	"esal" : "4500"
}
```

>

TO SAVE THE RECORD WITH SPECIFIED OBJECT ID

> db.EmployeeDetails.find();

{ "_id" : ObjectId("628acca167dc6828a6781837"), "eno" : "4", "ename" : "cherry", "esal" : "4000" }

{ "_id" : ObjectId("628acca167dc6828a6781838"), "eno" : "5", "ename" : "anil", "esal" : "5000" }

{ "_id" : ObjectId("628acca167dc6828a6781839"), "eno" : "4", "ename" : "padma", "esal" : "4500" }

> db.EmployeeDetails.save({"_id":new ObjectId("628acca167dc6828a6781842"),no:"5",name:"ratna",sal:"5600"});

WriteResult({

    "nMatched" : 0,

    "nUpserted" : 1,

    "nModified" : 0,

    "_id" : ObjectId("628acca167dc6828a6781842")

})

TO REMOVE SPECIFIC RECORD

> db.EmployeeDetails.remove({'ename':'cherry'})

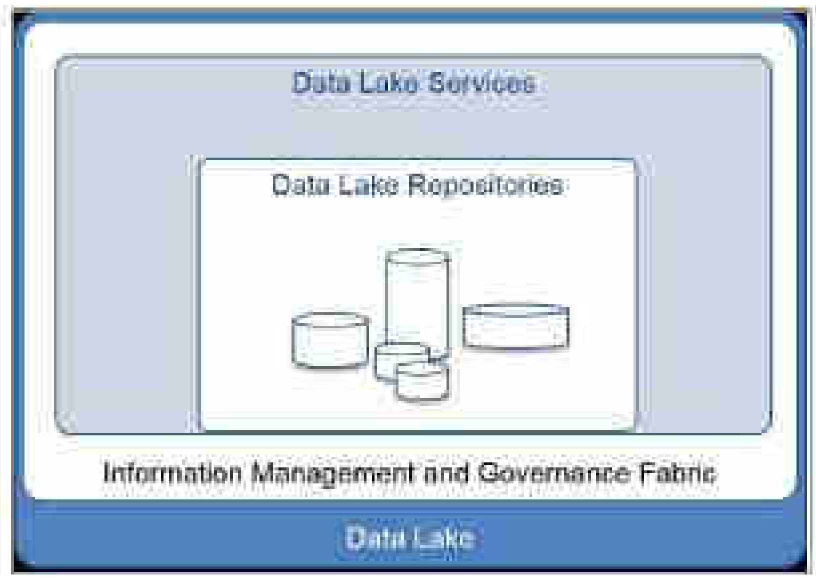
TO DROP THE TABLE

> db.EmployeeDetails.drop()

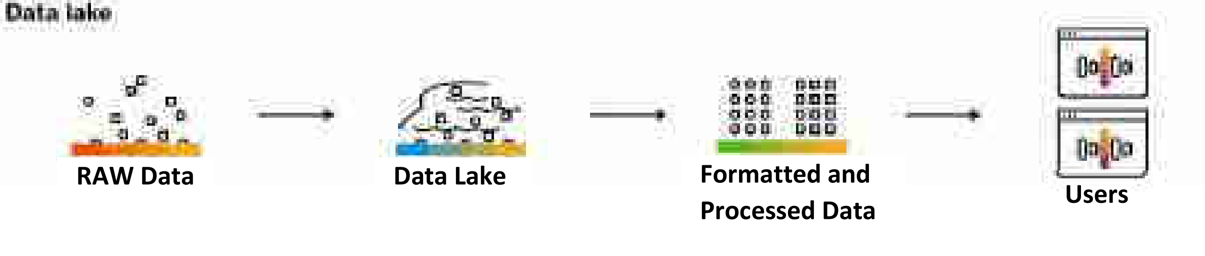
DATA LAKES

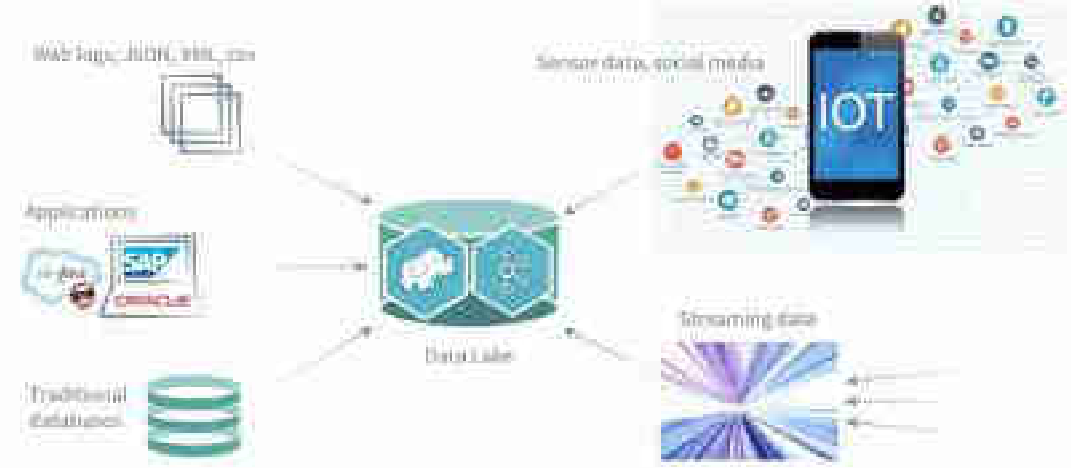
TOPIC 1: WHAT IS DATA LAKE

- Data Lake: Data Lake is a single repository for both structured and unstructured data, making it easy for data scientists to pull exactly what they need for analysis.



IBM Data Lake Architecture





Data Lake Data source. (Azure Data Lake, AWS Data Lake)

1.1 Key Features of Data Lake

1. Data Lake is repository of Structured Data (Invoices and Receipts, Sensor Data, Online Forms, Spread Sheets, CRM Profiles) and Unstructured Data (Social Media Content, Emails, Podcasts (Voice Recognition), Security Footage, Transcripts).

2. **Data Ingestion:** Data is gathered from multiple data sources and loaded into the data lake. The process supports all data structures, including unstructured data. It also supports batch and one-time ingestion

3. **Security:** Security Protocols are implemented from *loading*, *search*, *storage*, and *accessibility*. Other facets of data security such as *data protection*, *authentication*, *accounting*, and *access control* to prevent unauthorized access are also paramount to data lakes.

4. **Data quality:** Information in a data lake is used for decision making, which makes it important for the data to be of high quality. Poor quality data can lead to bad decisions, which can be catastrophic to the organization.

5. **Data governance:** Administering and managing *data integrity*, *availability*, *usability*, and *security* within an organization.

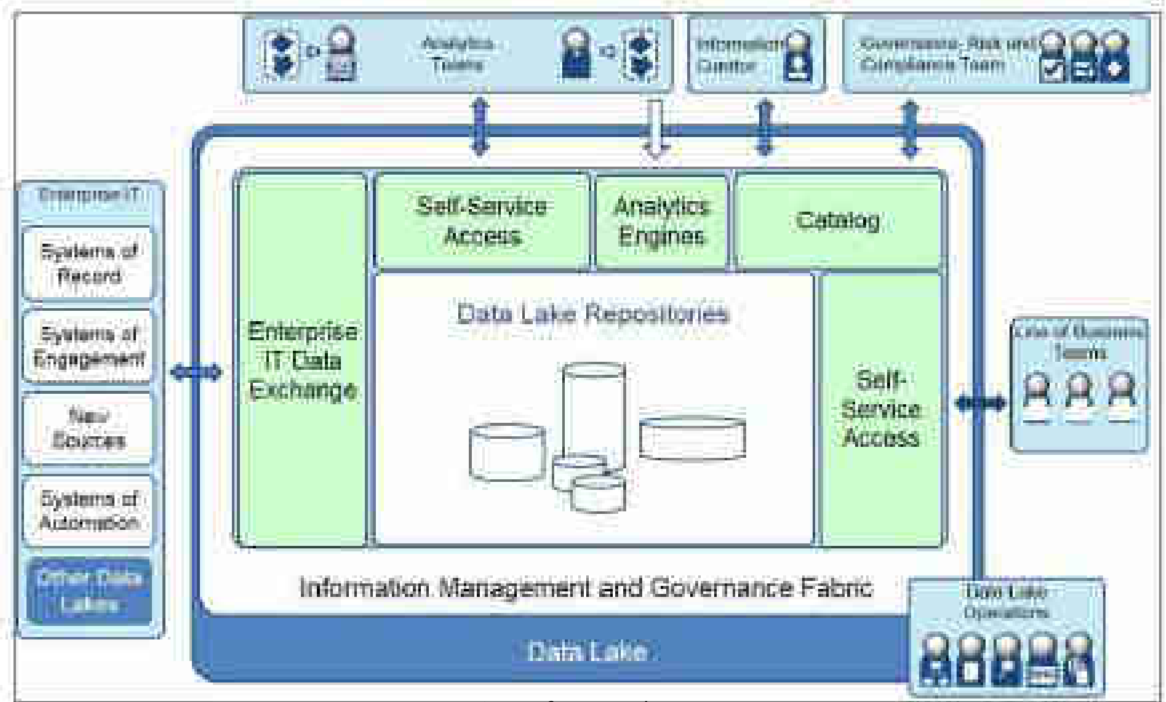
6. **Data discovery:** Discovering required data is important before data preparation and analysis. It is the process of collecting data from multiple sources and consolidating it in the lake, making use of tagging techniques to detect patterns enabling better data understandability.

7. **Data exploration:** Data exploration starts just before the data analytics stage. It assists in identifying the right dataset for the analysis.

8. **Data storage:** Data storage should support multiple data formats, be scalable, accessible easily and quickly, and be cost-effective.

9. **Data auditing:** Facilitates evaluation of risk and compliance and tracks any changes made to crucial data elements, including identifying who made the changes, how data was changed, and when the changes took place.

10. **Data lineage:** Concerned with the data flow from its source or origin and its path as it is moved within the data lake. Data lineage smoothens error corrections in a data analytics process from its source to its destination.



Services of Data Lake

1.2 Key services of Data Lake

- **Governance metadata**: Metadata management involves managing data about data. Defines the governance program and the glossaries of business terminology that describes the types of data held and used by the organization.

- **Technical metadata:** Provides the inventory of the data assets of the organization. These data assets are used by numerous run times, such as applications, data movement and transformation engines, and databases and reporting platforms.

- **Operational metadata**: Provides transparency on the operation of the information supply chains as they copy data between the systems and data platforms, which is often referred to as lineage.

- **Maintenance and Support:** There's no need to worry about the hardware, software, security, or compatibility with other systems. This service includes the initial setup of all these elements as well as any ongoing maintenance and support that is needed.

- **Limitless Capacity:** When operating an on-prem data lake, user must balance storage space with hardware availability. When using data lake as a service, however, storage capacity is always available – you only pay for what you use.

- **Active Backups:** As with any data storage solution, an effective backup program is critical for loss avoidance. Data lake as a service includes a full backup of your data that you can rely on.

- **Enhanced Business Intelligence:** When using data lake as a service on a cloud platform, required data is available for analytics using modern tools.

1.2 Benefits of a Data Lake

2. A data lake is an agile storage platform that can be easily configured for any given *data model*, *structure*, *application*, or *query*. Data lake agility enables multiple and advanced analytical methods to interpret the data.

3. Being a *schema on read* makes a data lake scalable and flexible.

4. Data lakes support queries that require a *deep analysis* by exploring information down to its source to queries that require a simple report with summary data.

5. Most data lakes software applications are open source and can be installed using low-cost hardware.

6. Schema development is deferred until an organization finds a business case for the data. Hence, no time and costs are wasted on schema development.

7. Data lakes offer centralization of different data sources. They provide value for all data types as well as the long-term cost of ownership.

8. Cloud-based data lakes are easier and faster to implement, cost-effective with a pay-as-you-use model, and are easier to scale up as the need arises. It also saves on space and real estate costs.

1.3 Challenges and Criticism of Data Lakes

1. Data lakes are at risk of losing relevance and becoming data swamps over time if they are not properly governed.

2. It is difficult to ensure data security and access control as some data is dumped in the lake without proper oversight.

3. There is no trail of previous analytics on the data to assist new users.

4. Storage and processing costs may increase as more data is added to the lake.

5. On-premises data lakes face challenges such as space constraints, hardware and data center setup, storage scalability, cost, and resource budgeting.

1.4 Popular Data Lake Technology Vendors

Popular data lake technology providers include the following:

- Amazon S3: Offers unlimited scalability
- Apache: Uses Hadoop open-source ecosystem
- Google Cloud Platform (GCP): Google cloud storage
- Oracle Big Data Cloud
- Microsoft Azure Data Lake and Azure Data Analytics
- Snowflake: Processes structured and semi-structured datasets, notably JSON, XML, and Parquet

---

TOPIC 2: THE VALUE OF THE DATA LAKE TO ING

Today, almost any company has the publicly stated ambition to become data driven.

Everybody recognizes the value of data and the potential for data to both transform existing processes and to create new value.

Facebook, who collectively make billions from data, many organizations are struggling to deliver on their strategy.

2.1 Problems for companies to become data driven and to obtain value from their data.

1. Historical perspective: Data has never been managed as an asset.
2. Not knowing what data is exchanged is a result of lacking documentation.
3. Many point-to-point connections built over time as “the fastest choice”.
4. Every new interface starts from scratch because we can’t re-use.
5. Conflicting definitions make it difficult to exchange data across domains and countries.
6. The need to go to real time.
7. The big data hype has led to more data that is less understood.
8. Is governance a four-letter word in an agile world?
9. Creating the data lake architecture was a major step in our journey.
10. Selling a complex architecture is more work than creating it.

2.2 ING means Information Governance Catalogue.

1. Understand the meaning of the data

- Understanding data provides a common business language and vocabulary to enable a deeper understand of data assets (Structured, Semi-structured and Unstructured).

2. Establish a governance framework

Documents governance policies and enacts rules to help you define how information should be *structured*, *stored, transformed* and *moved.*

3. Explore data lineage

- Enables to understand *how data flows across the information landscape*, to track where data was sourced from or being consumed.
- Allows to develop trust and confidence in such data.

4. Benefit from the cloud

- Helps the user to move to the cloud without the need for an on-premises environment and reduces administration costs.

5. Strengthening data access across the organization without any problems.

6. **Delivering the right data to the right people at the right time**

TOPIC 3: THE 5-LEVEL MODEL OF GOVERNANCE MATURITY IMP

Data governance model addresses a total of 11 domains mentioned below:

1. Data Risk Management and Compliance
2. Value Creation
3. Organizational structure and awareness
4. Policy
5. Stewardship (Company executives protect the interests of the owners or shareholders and make decisions on their behalf.)
6. Data Quality Management
7. Information Lifecycle Management
8. Information Security and Privacy
9. Data Architecture
10. Classification and Metadata
11. Audit Information, Logging, and Reporting

3.1 Structuring the Governance Program

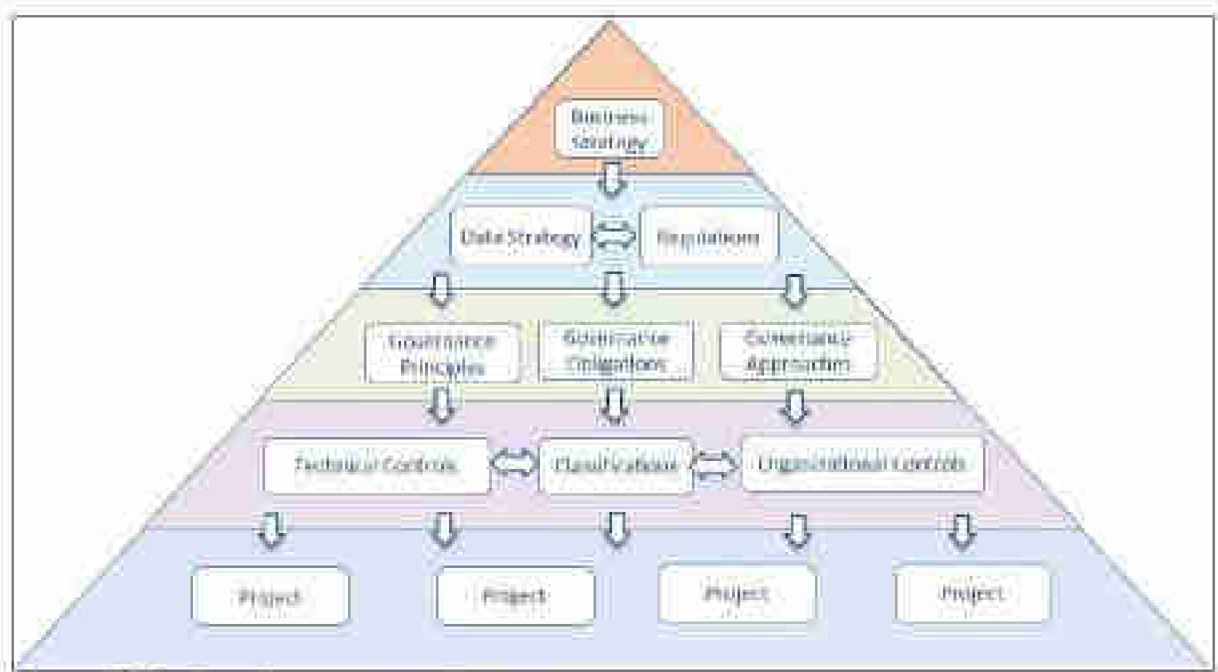
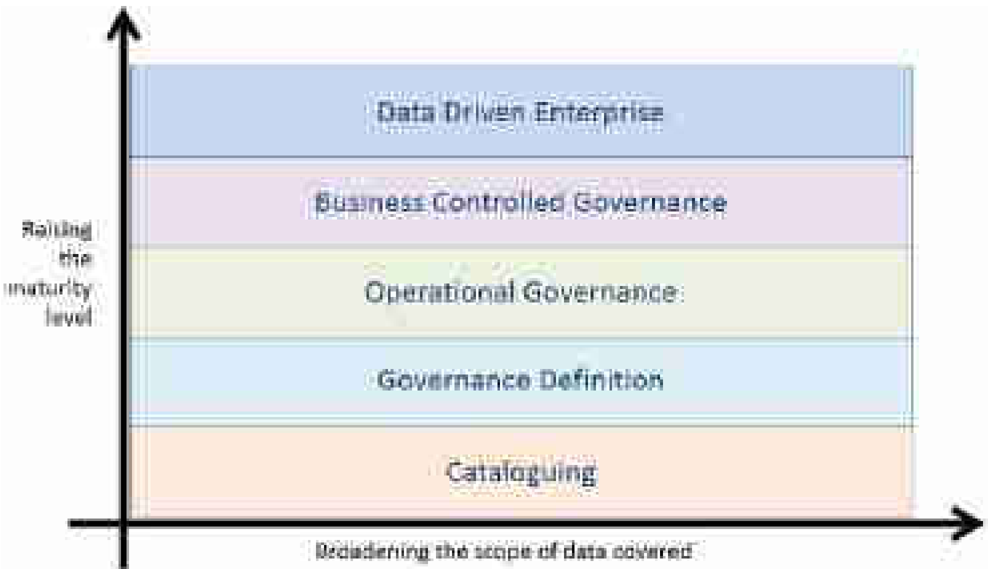


Figure 5 Structure of a governance program

3.2 Governance Maturity Model



Maturity Level 1: Cataloging

- Cataloging refers all the data source used by the organization.
- Cataloging stores the information about *location*, *description*, and *owner* of data stored in the Data Lake.
- Analytical functions can help to identify the logical type and classifications for the data, but the human expert is needed to confirm the classification is correct based on their understanding of the business context around the data.
- This is the stage at which a lack of data governance becomes evident. Business and IT leaders start to understand and acknowledge the value of information and EIM (Enterprise Information Management).

Maturity Level 2: Defining Governance

- Governance is the collective tools, processes and methodologies that enable an organization to align business strategy and goals with IT services, infrastructure or the environment
- The next level of maturity is where the governance drivers, requirements, and controls are linked to the catalog of data stores, typically via the classifications. This process creates an authoritative definition of how data should be governed.

Maturity Level 3: Operational Governance

- Operational governance is centred on key operating decisions made by executives and managers and follow-through on the execution of policies. It presents a framework for managers to improve how decisions are made and carried out.

- The IT governance framework highlights the 'who' and 'how' elements of the operating model. It defines the principles, rules and processes that enable effective decision-making. It provides the framework to address how decisions are made, who has the authority to make decisions and how decisions are communicated.

Maturity Level 4: Business-Controlled Governance

- There are seven broad categories of information system management controls in an organization i.e. top management controls, systems development management controls, programming management controls, data resources management controls, security management controls, operations management controls, and quality assurance management controls.

- Top management controls involve activities like planning, organizing, leading, and monitoring / evaluation.

- Systems development management controls involve activities like feasibility study and project initiation, system analysis and specifying user requirements, systems design and development, acceptance testing, implementation and maintenance, and auditing the systems development management function.

- Programming management controls involve activities like planning, control, design, coding, testing, and operation and maintenance.

- Data resources management controls include defining, creating, redefining, and retiring data; making the database available to the users; informing and servicing users; maintaining the integrity of the database; and monitoring operations and performance.

- Security management control involves conducting security programs.

- Operations management controls include control of computer and network operations; maintaining data files, program files, and documentation; help desk and technical support; and management of outsourced operations.

- Quality assurance management controls include establishing quality goals and standards; checking conformity with QA standards; identifying areas for improvement; reporting to the management; and training employees in QA standards.

Maturity Level 5: Data Driven Enterprises

- Finally, the data-driven enterprise is one where decisions are routinely made using a wide variety of data, from both inside and outside of the organization.

- Individuals own and manage data and develop analysis and visualizations of data that is sharable. They collaborate around the use of data, provide feedback opinions, and share knowledge about the data they are using.

PARVATHANENI BRAHMAYYA(P.B.)

SIDDHARTHA COLLEGE OF ARTS & SCIENCE

VIJAYAWADA, ANDHRA PRADESH

Autonomous Since 1988 NAAC Accredited at 'A+' (Cycle III) ISO 9001:2015 Certified

LECTURE NOTES PREPARED BY

G.SMART KRISHNA, ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENECE

P.B.SIDDHARHA COLLEE OF ARTS & SCIENECE, VIJAYAWADA, AP, INDIA PIN: 520010

COURESE: DATABASE MANAGEMENT SYSTEMS

CONTENTS:

1. **Introduction**
2. **ER Modeling**
3. **EER Modeling**
4. **Record Storage**
5. **Concurrency Control**
6. **Indexing (B-Trees & B+-Trees)**
7. **Normalization**
8. **Relational Algebra Operations**
9. **Secondary Storage Devices**
10. **Transaction Processing**
11. **Triggers, Cursors & Constraints**

Introduction to Data Base Management System

- **Data: Collection of *facts* (or) *raw material* for information.**
- **Information: Processed data.**
- **Schema: Structure of database (or) Structure of database objects.**
- **Entity: A *thing* (or) *object which* have some implicit meaning.**
- **Record: The values associated for collection of fields.**
- **Domain: The range of finite values associated for attributes.**
- **Database: Collection of files (or) collection of tables (or) collection of database objects.**
- **Attribute: Property of entity.**
- **Database management system: Set of software modules used to *create* database objects and *access* the database objects.**

1.1 How does traditional file processing system differs from DBMS — IMP

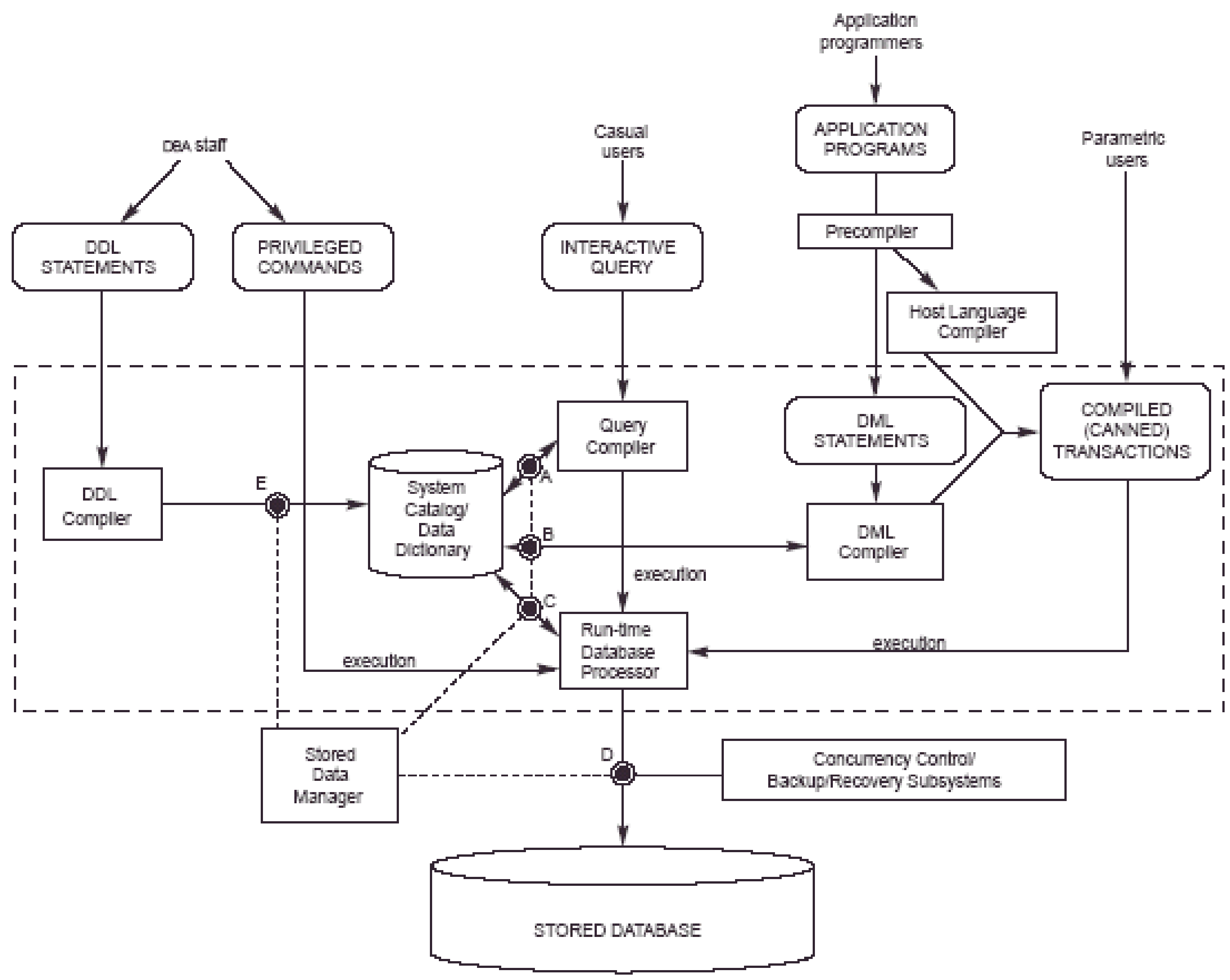
- **In traditional file processing system an *entity* can be described as a *file* using programming language by the application programmer.**
  - **To access the file i.e. to *add the data into the file, to delete the data from the file & to modify the content of the file*, the file must be *opened* and *closed* . Only an application programmer can access the file.**
- **In DBMS an entity is describes as tables. An end user can *create* & *access* the table by invoking SQL commands.**
  - **An end user having knowledge about SQL can *create* & *access* the table.**
  - **DBMS also provide GUI's (Graphical User Interfaces) to design front-end's.**
  - **An end user who doesn't having knowledge about SQL can also easily access the table by operating the controls in the front-end's.**

- **Problems with traditional file system**

1. Data redundancy. **E.g. In university database registration personnel and accounting office maintains separate data files of same data for handling its data processing applications.**
2. Data inconsistency. **E.g. Files may consists inconsistent data since there are no validation checks while accepting the data .**
3. Does not have support for multiple transactions.
4. Security features are limited.
5. Complex relationships can't be established.
6. Does not have integrity constraints for validation checks.
7. Backup and recovery is difficult

1.2 Components of DBMS (or) Database System Environment

- **Users: People who access the database.**
- **DBA Staff: DBA uses *DDL statements* and *privileged commands* to assign access privileges to end users.**
- **DDL Compiler: Compiles DDL statements.**
- **Casual users: By invoking interactive queries, users *creates* & *access* database according to the privileges assigned by DBA.**
- **Query compiler: Compiles the user queries.**
- **System catalogue (or) Data dictionary: A catalogue is a table that stores entire description of database.**


Application programmers
APPLICATION PROGRAMS
Casual users
Parametric users
DBA staff
DDL STATEMENTS
PRIVILEGED COMMANDS
INTERACTIVE QUERY
Precompiler
Host Language Compiler
Query Compiler
DML STATEMENTS
COMPILED (CANNED) TRANSACTIONS
DDL Compiler
E
System Catalog/ Data Dictionary
A
B
C
DML Compiler
execution
Run-time Database Processor
execution
execution
Stored Data Manager
D
Concurrency Control/ Backup/Recovery Subsystems
STORED DATABASE

- **Application programmers: They access the database by writing application programs compatible to DBMS.**

- **Pre compiler: The *pre compiler* separates *DSL (Data Sub Language)* and *Host Language*.**

- **Host languages compiler: Compiles the host language(Programming language).**

- **DML compiler : Compiles the DML statements.**

- **Runtime database processor: Access the database *(insert, delete and update)* when user request is successful.**

- **Parametric users: The persons who regularly access the database by invoking transactions.**
  E.g. Bank clerks.

- **Canned transactions: Transactions by parametric end users.**
  E.g. Bank credit / debit transactions.

- **Store data manager: Manages the data stored persistently in the database.**

- **Concurrency control subsystem: Avoids concurrency of multiple transactions by enforcing concurrency control techniques.**

- **Backup subsystem: Provides backup to the database or database objects.**

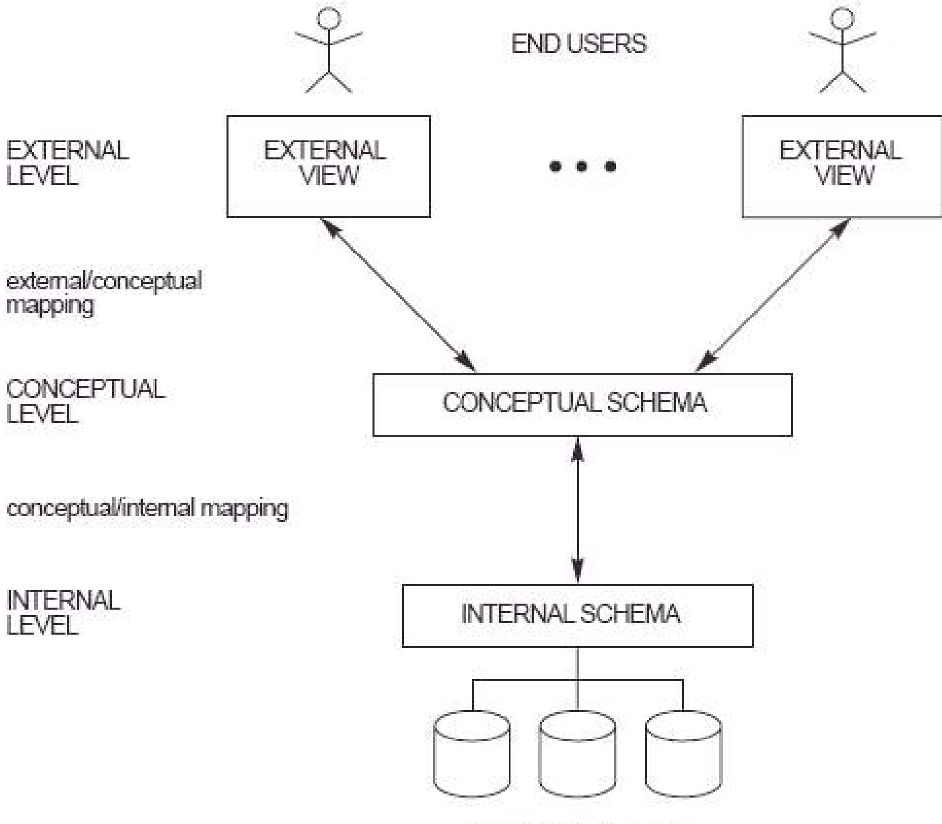
1.3 Architecture of DBMS	VV IMP

- **ANSI (American National Standard Institute) / SPARC (Standards Planning And Requirements Committee) formulated architecture of DBMS.**

- **Architecture is divided into 3 levels**

1. Internal level
2. Conceptual level
3. External level

Figure 2.2 Illustrating the three-schema architecture.



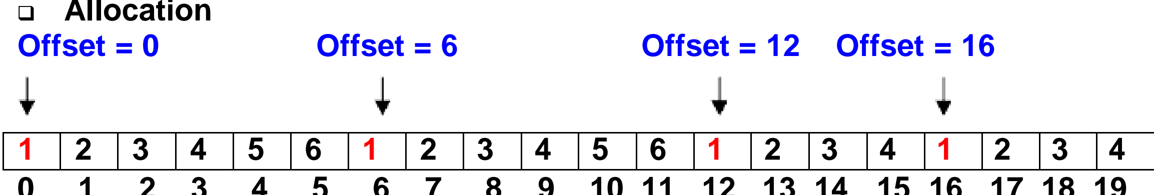
Internal level

- **Internal level Indicates number of bytes allocated by the table when it is stored physically.**

- **When employee table with 3 attributes EMPNO, DEPTNO & PAY is stored in database persistently the following number of bytes allocated according to the shown format.**

STORED EMP LENGTH = 20
PREFIX TYPE = BYTE(6), OFFSET = 0
EMP# TYPE = BYTE(6), OFFSET = 6, INDEX = EMPX
DEPT# TYPE = BYTE(4), OFFSET = 12
PAY TYPE = FULLWORD, OFFSET = 16

- Allocation



- From the given example

PREFIX	System generated attributed where the values of attributes is fixed.
OFFSET	Indicates starting address of next attribute.

Conceptual level

- Using interactive queries user defines

a. Entities
b. Attributes
c. Datatypes for attributes
d. Constraints imported over the database objects
e. Relationship between the entities

E.g. Create table EMP (EMP_NO varchar2(6),
DEPT_NO varchar2(4),
SAL Number(4));

E.g. Alter table emp
add constraint ENO_PK primary key (EMP_NO);

External level

- Entity can be represented and accessed using different programming languages by different users.

- Different *Application programmers* uses different *Programming languages* comptable to DBMS.

- Programming languages are combination of *Host language* and *DSL (SQL queries).*

- Embedded SQL: SQL statements embedded in programming languages.

- Both DSL and Host langage are tightly coupled.

```
EXTERNAL (PL/I)                              COBOL
        DCL   1. EMP                         01      EMP
              2. EMP# CHAR(6),                       02      EMPNO PIC X(6)
              3. SAL    FIXEDBIN(31);                02      DEPTNO PIC X(4)
----------------------------------------------------------------------------
CONCEPTUAL
              EMP
              EMP_NO      CHARACTER(6)
              DEPT_NO     CHARACTER(6)
              SAL         NUMBER(5)
----------------------------------------------------------------------------
INTERNAL

        STORED EMP LENGTH = 20
              PREFIX  TYPE  = BYTE(6), OFFSET = 0
              EMP#     TYPE = BYTE(6), OFFSET=6, INDEX = EMPX
              DEPT#    TYPE = BYTE(4), OFFSET = 12
              PAY      TYPE = FULLWORD, OFFSET = 16
```

1.4 Data Independence — V V IMP

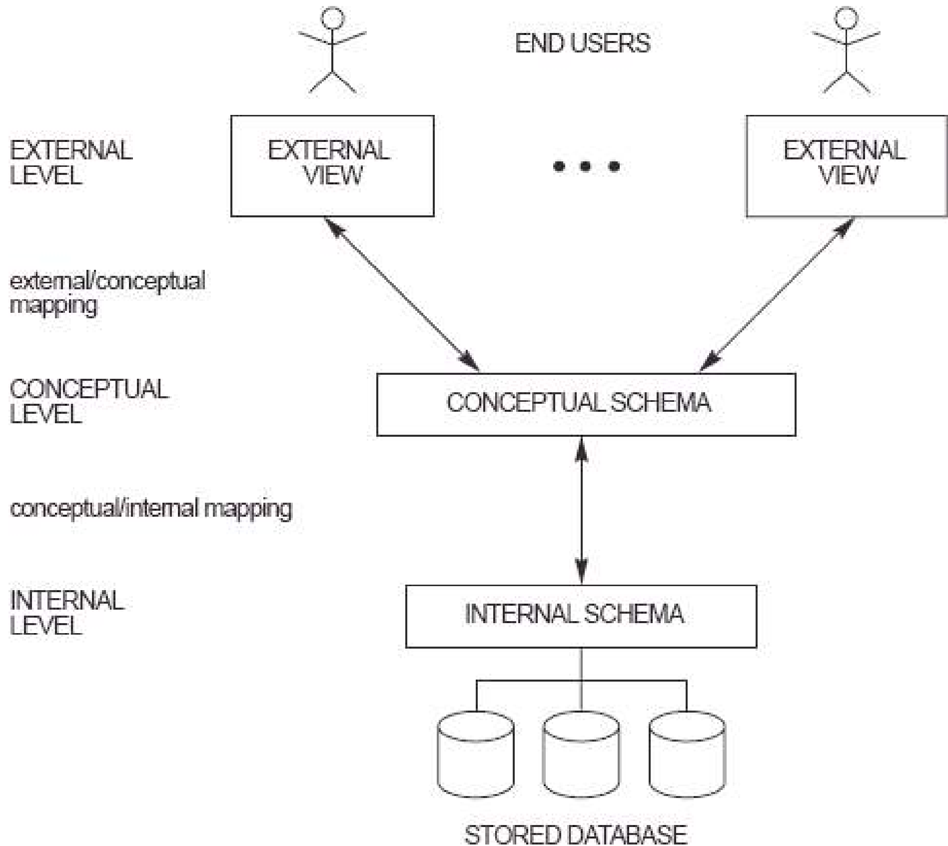
- Data Independence: The *immunity* of application (components of DBMS) to change its *storage structure* and *accessing techniques.*

  E.g. Create table EMP(empno varchar2(10),
  ename varchar2(10));

  E.g. Alter table EMP modify ename varchar2(20);

(or)

- The capacity to change schema at one level of a DBMS without having to change the schema at next higher level.



- **Physical data independence: The capacity to change *internal schema* without having to change *conceptual* and *external schemas*.**

- **Logical data independence: The capacity to change *conceptual schema* without having to change *external schema*.**

1.5 Advantages of DBMS — IMP

1. **Redundancy can be reduced.**
   Note: Duplication can be reduced by establishing relationship between tables.

2. **Inconsistency can be avoided.**
   E.g. Number of hours worked per week

EMPNO	NHWW
1	160
2	300

← **Inconsistent data**

3. **Multiple transactions can be supported (Multi transaction system).**
   E.g. Online transaction processing system (Railway reservation)
   - **In multi transaction system, a single database item can be accessed by multiple user simultaneously.**

4. **Concurrency can be avoided.**
   - **When multiple transactions update the single table simultaneously, then concurrency occurs. DBMS solves the problem of concurrency by using Concurrency Control Algorithms.**

5. **DBMS provides security to database items.**
   - **DBA assigns privileges (*read*, *write* & *modify*) to *individual users* and *group of users* to restrict the database access from unauthorized users.**

6. **Provide persistent storage to database objects and data structures.**

7. **Provide multiple user interface.**
   - **DBMS provides**

GUI^s
Query languages
Application programs
Menu driven interfaces to users

7. **Complex relationships can be establish between the database objects using referential integrity.**

8. **Integrity constraints can be enforced.**

9. **DBMS provides backup and recovery.**
   - **Using backup & recovery subsystems, software & hardware failures can be recovered.**

10. **Standards can be enforced.**
    - **DBA ensures all applicable standards while assigning privileges to *individual users* (or) *user groups.***

11. **Conflicting requirements can be balanced.**
    - **Knowing the overall requirements of enterprise the DBA structures the system and provide resources.**

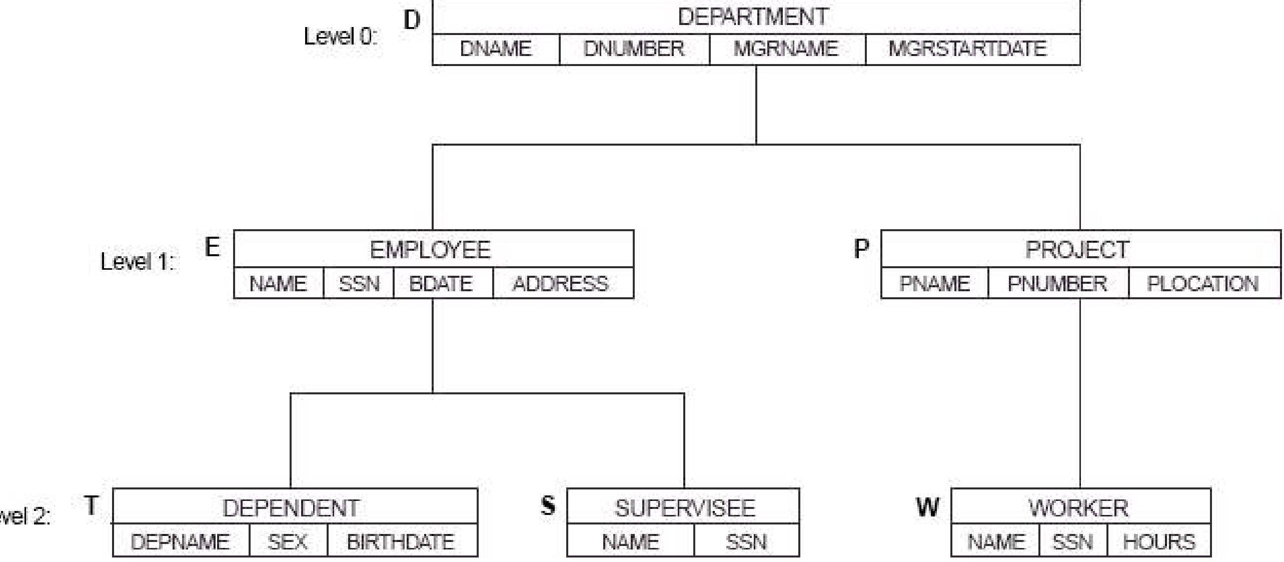
1.6 Classification of DBMS (or) Different data models of DBMS	**IMP**

- **DBMS are classified into 4 types**

(a) Hierarchical model

(b) **Relational model**
(c) **Network model**
(d) **Object oriented model**

1. Hierarchical model



- **Represent data as hierarchical tree structures.**
  E.g. Top down approach

- **Parent table is represented at Level 0 and child's are represented at next higher level.**

- **Advantages**

(a) **Parent and child relationship is represented as hierarchy.**
(b) **Easy to represent and simple to construct.**
(c) ***Navigational* & *procedural nature* of processing.**

- **Disadvantage**

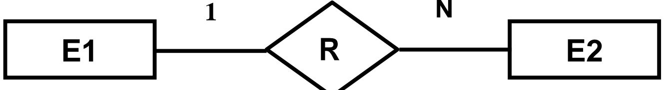
There is no structured query language for hierarchical model .

2. Relational model

- **Represents relationship between tables in the database.**

- **Referential integrity can be enforced between any number of tables.**

- **Query language is used to access the table.**



- **Advantages**

(a) Data redundancy can be reduced.
(b) Retrieval scope is more.

3. Network model

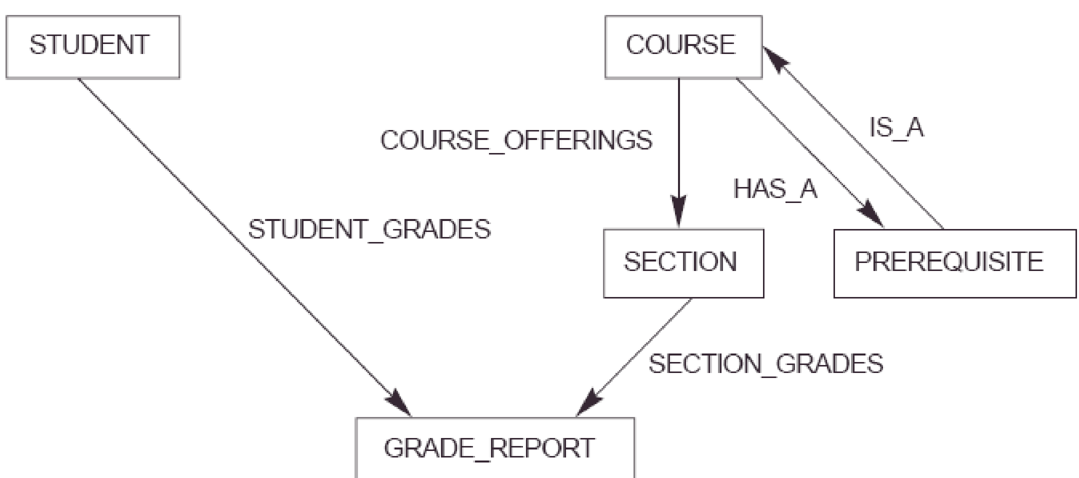
- **Multiple relations between the multiple tables are formulated.**

Record type	A Table is Record type.
Set type	1 : N relationship between tables.

- **Example for Record type**

Student

- **Example for Set type**



- **Advantages**

(a) Complex relationship between entities should be expressed.
(b) Entities (Record types) can be added or deleted from Set type.

- **Disadvantages**

(a) Record types and relations can't be specified exactly.
(b) *Navigational* & *procedural* nature of processing.
(c) Database contains complex array of pointers to establish the relations between the Record type.

4. Object oriented model

- **Class : Description of entity in terms of *data* (attributes) & *operations* (functions).**

E.g.

```
Class emp
{
    int   empno;
    char  ename;
    public :
    void accept_empno( );
    void accept_ename( );
}
emp x,y,z
```

- **Object: Instance of class (or) runtime entity in the application software domain.**

- **OODBMS: Collection of software modules used to *create the object* , provide *persistence* & *access* to the object .**

- **Features**

(a) Encapsulation	Wrapping up of *data* (attributes) & *functions* into a single unit.
(b) Data Abstraction	Representing essential features without including the explanation or implementation. E.g. #include<stdio.h>
(c) Polymorphism	Different behaviors of same object.
(d) Inheritance	The mechanism of deriving a *new class* from an *existing class* without modifying the original one.
(e) Functional Programming	Every component (sub module) can be implemented as functions.
(f) Information hiding	Hiding *implementation* and *complexity* of object. E.g. Predefined functions like SUM,AVG.

- **Advantages**

(a) Complex *engineering* & *scientific* applications can be developed.
(b) Support for *distributed database environment*.
(c) Uses the concepts of *artificial intelligence & neural networks.*

1.7Actors on the scene (or) Important persons involved in RDBMS

DBA(Database Administrator) VIMP

(a) Responsible for ***authorizing access*** to the database.
(b) Formulates ***software*** & ***hardware*** resources needed in the enterprise.
(c) Verifies ***system log*** to keep track activities of users.
(d) Responsible for ***database security***. ❑ Security can be imposed by assigning ***access privileges*** to the users. ❑ There are two types of access privileges. 1. Discretionary access privileges: Each individual user will be assigned **USER ID**, **PASSWORD**, and **ACCESS PRIVILEGES** (Read / Write / Update). 2. Mandatory access privileges: Users are classified into user groups **TS(TOPSECRETS)**, **S(SECRETS)**, **C(CLASSIFIED)**, **U(UNCLASSIFIED)** and appropriate access privileges will be assigned.

Database Designers

❑ Database designer is person

(a) Responsible for identifying data to be stored in database.
(b) Represent appropriate structure of database.
(c) ***Database designer*** interact with ***users*** and develop ***database view*** according to users requirement.
(d) Designer must have complete exposure of DBMS architecture.

End Users

❑ End users are the people who access the database.

(a) Casual users	❑ Users who ***occasionally*** access the database using query languages. E.g. ***Middle level*** or ***high level managers***.
(b) Naïve(or) Parametric end users	❑ Users who ***regularly*** access the database. E.g. Blank clerks. ❑ Naive user may or may not have knowledge about queries. ❑ Canned transaction: Transaction by Naive users.
(c) Sophisticated end users	Users DBMS to facilitate ***complex*** operations. E.g. DBA, DA, Business Analysts, Scientists, Engineers, Professors etc.

(d) Stand alone users	❑ Access database through GUI's. ❑ User doesn't have knowledge about SQL queries and Programming languages.

System Analysts

(a) Identifies overall requirements to formulate Physical Design.

Application Programmers

(a) Responsible for developing applications using programming languages.
(b) Application programmer must be expert in both ***Host Language*** and ***Data Sub Language***.

1.7 Workers behind the scene (or) Persons involved in the development RDBMS

Tool Developers

- Tool: Tool is a software package.
- Tool developer: Person who design tools to access the database.
- Database contain the following tools

Forms	Forms used to design GUI[s].
Wizards	Wizards are windows to facilitate automated tasks.
Prototypes	Prototype is resembling model for actual model.
Simulation	Simulation is a technique used to transform one object into another object.
Natural Languages	Natural languages are high level languages.
Text Data Generator	Program used to generate data from the database.

Operator & Maintenance personals

- Persons who are responsible for actual ***running*** and ***maintenance*** of ***hardware*** and ***software*** resources of database systems.

DBMS system designers

- Person who designs the DBMS modules

1.8 Implication of Database Approaches (or) Changes occurred when DBMS is introduced

1. **Potential for enforcing standards.**
- DBA enforce applicable standards for normal running and maintenance of system. This facilitates ***communication*** and ***co-operation*** among ***various departments***, ***projects***, ***users*** within the organization.

2. **Reduces application time.**
- Application development time is reduced by 1/6th to 1/4th due to the usage of ***forms***, ***wizards***, ***prototypes***, ***simulation*** and ***natural languages***.

4. **Data *retrieval* and *representation* is efficient due to reporting.**

5 **Flexibility.**
- DBMS is flexibility due to ***data independence*** (easy to change structure of database objects).

6 **Reliability.**
- Database is available always i.e., lost data can be recovered by recovery sub system.

7 **Economy of sale.**

(a)	Project development cost will be reduced by using DBMS since ***software*** and ***hardware*** resources can be shared.
(b)	***Operation*** and ***maintenance*** cost can be reduced by adopting advanced ***software*** and ***hardware*** technologies.

1.9 DBMS languages and Interfaces — VVIMP

DDL (Data Definition Language)

- Language used to create database objects.
- Used to ***create*** & ***alter*** the database objects.
  E.g. Create, Alter commands.

E.g.
```
Create table emp
    (empno number(10),
    ename varchar2(15));
```

E.g.
```
Alter table emp
    modify ( empno number(15));
```

SDL (Data Definition Language)

- SDL (Storage Definition Language): Used to specify the *internal schema* of database objects.

E.g. STORED-EMP LENGTH = 20
PREFIX TYPE = BYTE(6), OFFSET = 0
EMP# TYPE = BYTE(6), OFFSET = 6, INDEX = EMP
DEPT# TYPE = BYTE(4), OFFSET = 12
PAY# TYPE = FULLWORD, OFFSET = 16

DML (Data Manipulation Language)

- DML (Data Manipulation Language): Used to *access* database objects.
  E.g. Insert, Delete, Update commands

E.g. Insert into emp
values(&empno,'&ename');

E.g. Update emp
set ename='john'
where empno=1;

E.g. Delete from emp;

DCL (Data Control Language)

- Used to provide persistency and recovery to database objects.

E.g.

Commit;	Commits the transaction.
Rollback;	Database items acquires previous state.
Savepoint <savepoint_name>;	Creates savepoint in a transaction.
Rollback to <savepoint_name>;	Databse item acquires savepoint status.

TCL (Transaction Control Language)

(a) *Grant* option assign privileges to the users.
(b) *Revoke* option cancel access privileges from the users.

E.g. GRANT INSERT, DELETE ON EMPLOYEE, DEPARTMENT TO A2;

E.g. GRANT SELECT ON A3EMPLOYEE TO A3 WITH GRANT OPTION;

E.g. REVOKE SELECT ON EMPLOYEE FROM A3;

1.10 DBMS Interfaces

1. Forms	For *front-end* design using *GUI's*.
2. Menu's	(a) Menu's are command interpreters. (b) By choosing menu options user can execute commands. (c) Menu's are classified into Pull-down menu's ,Pop-up menu's.
3. Natural Languages	RDBMS provides support to multiple programming languages.
4. Wizards	Windows used to perform automated tasks.
5. Prototypes	Resembling model for actual model.
6. Simulation	Transformation of one object into another object.
7. Interfaces to DBA	DBA creates *user accounts* and provide security restrictions over the assets of DBMS.
8. Interfaces to Parametric users	User defined packages.

1.11 Database system utilities

Loading

- The process of storing data files into database.
- Data file is represented in *internal schema* when it is stored physically in the database.

Conversion tool

- Data base objects are represented in *physical format (flat files)* when they are stored persistently in the database.
- Data base objects are represented in *logical format (tables)* when they are retrieved into primary memory.
- Conversion tool converts source format to desired format and vice versa.

Back up

- The process of taking duplicates for the actual data.
- Entire database is dumped into magnetic tape.
- Incremental backup: Notes the change since previous backup are recorded.

File reorganization

- Reorganizing the data files after they are stored.
  E.g. Indexing, Garbage collection

Performance monitor's

- Performance monitors are programs to determine the performance of *software* , *hardware*, *data*, & *users* and provide statistics to the DBA.

1.12 Tools , Application environment , Communication facilities

Tools

1. CASE (Computer Aided Software Engineering)	Used in the *design phase*.
2. Data dictionary	Stores the *entire description* of the database.
3. Contains	(a) Information about *designs*. (b) Application program description. (c) User information. (d) Usage standards.

Application environment

- Provide

(a) Application programs E.g. Host language
(b) Queries E.g. DSL (Data sub language)

Communication facilities

(a) Access the data in *distributed database environment* using communication software.
(b) Integration of *DBMS* & *Data communication system* is also called DB/DC systems.

Chapter –3
ER Model

3.1 Main phases of database design

1. Requirements collection and analysis	Database designers interview prospective database users to understand their requirements.
2. Conceptual design	Data requirements are formulated into entities, relationships and constraints.
3. Logical design	Database schema formulated using data models.
4. Physical model	Internal storage structures, access paths, and file organizations for database files are specified.

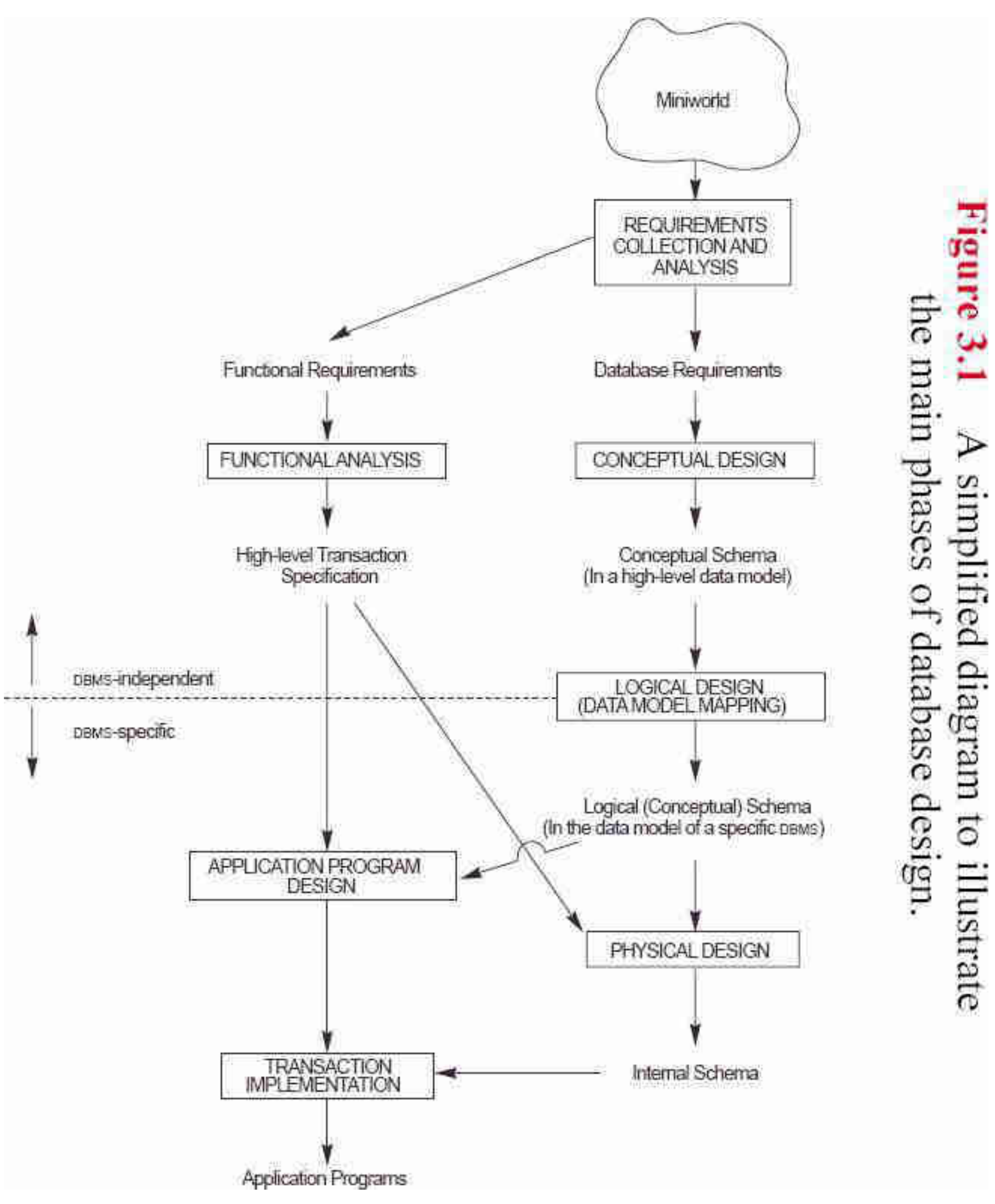


Figure 3.1 A simplified diagram to illustrate the main phases of database design.

3.2 Entity types, Entity sets, Attributes & Keys — IMP

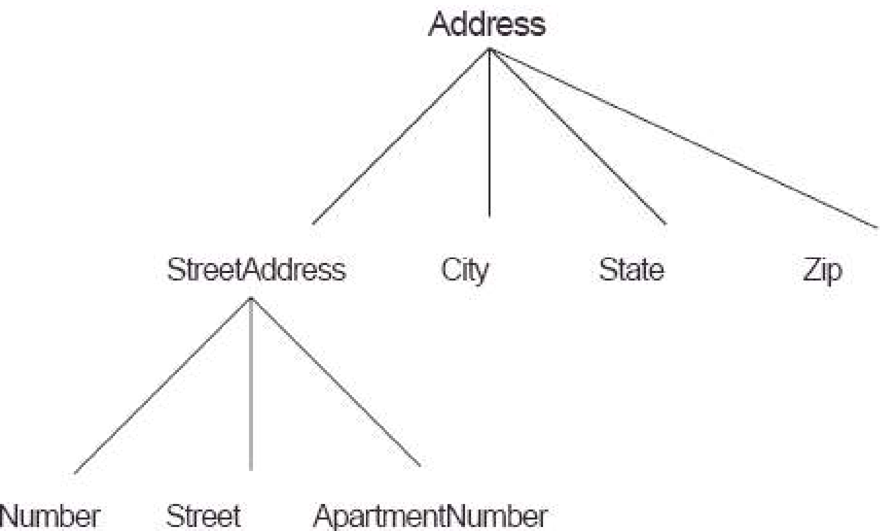
- **Entity:** **An object in the real world with an independent existence.**

- **Attribute:** **Property that describes an aspect of the entity.**

- **Attribute types:**

(a) Simple: Attribute that is not divisible
E.g. Age
(b) Composite: Attribute that can be divided into smaller parts.
E.g. Address

Figure 3.4 A hierarchy of composite attributes; the StreetAddress component of an Address is further composed of Number, Street, and ApartmentNumber.



(c) Single-valued: Attribute containing single value.
E.g. Age, City

(d) Multi-valued: Attribute containing set of values.
E.g. Locations for DEPARTMENT
E.g. College degree

(e) Stored: Attribute having fixed value.
E.g. Date Of Birth

(f) Derived: Attribute derived from the value of the existing attribute.
E.g. NumberOfEmployees for DEPARTMENT
E.g. Age

(g) Complex attribute: Composite and multivalued attributes can be nested in an arbitrary way.

Figure 3.5 A complex attribute AddressPhone with multivalued and composite components.

{AddressPhone({Phone(AreaCode,PhoneNumber)}, Address(StreetAddress(Number,Street,ApartmentNumber), City,State,Zip)) }

Figure 3.7 The CAR entity type, with two key attributes Registration and VehicleID. Multivalued attributes are shown between set braces {}. Components of a composite attribute are shown between parentheses ().

CAR
Registration(RegistrationNumber, State), VehicleID, Make, Model, Year, {Color}

car_1 •
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 1998, {red, black})

car_2 •
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 1999, {blue})

car_3 •
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 1995, {white, blue})

⋮

- **Entity sets : The collection of all entities of a particular entity type in the database at any point of time.**

Figure 3.6 Two entity types named EMPLOYEE and COMPANY, and some of the member entities in the collection of entities (or entity set) of each type.

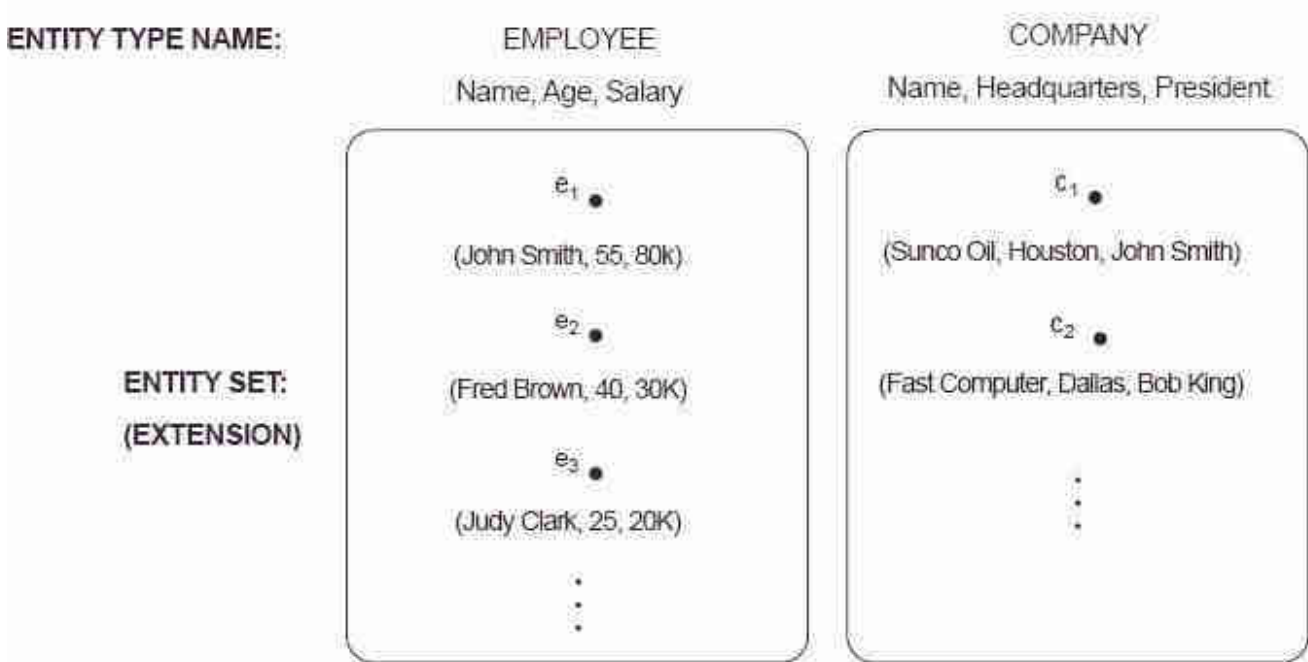
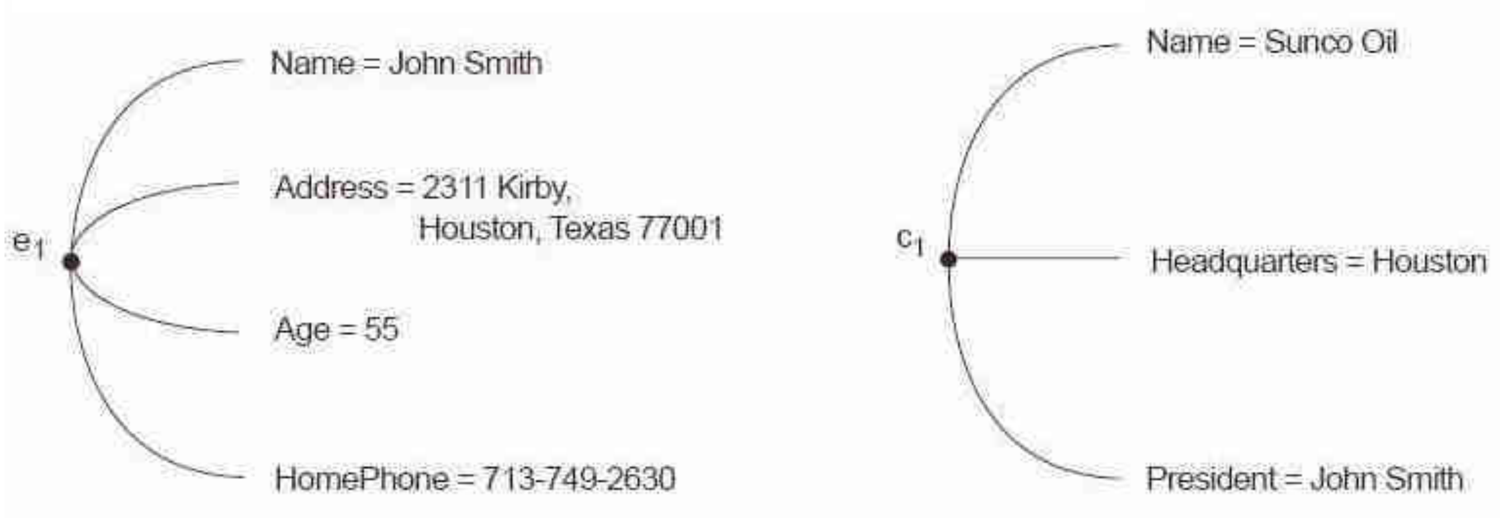


Figure 3.3 Two entities, an employee e_1 and a company c_1, and their attribute values.



SECOND

S#	STATUS	CITY
S1	20	LONDON
S2	10	PARIS
S3	10	PARIS
S4	20	LONDON
S5	30	ATHENS

Domain

$A: E \rightarrow P(V)$

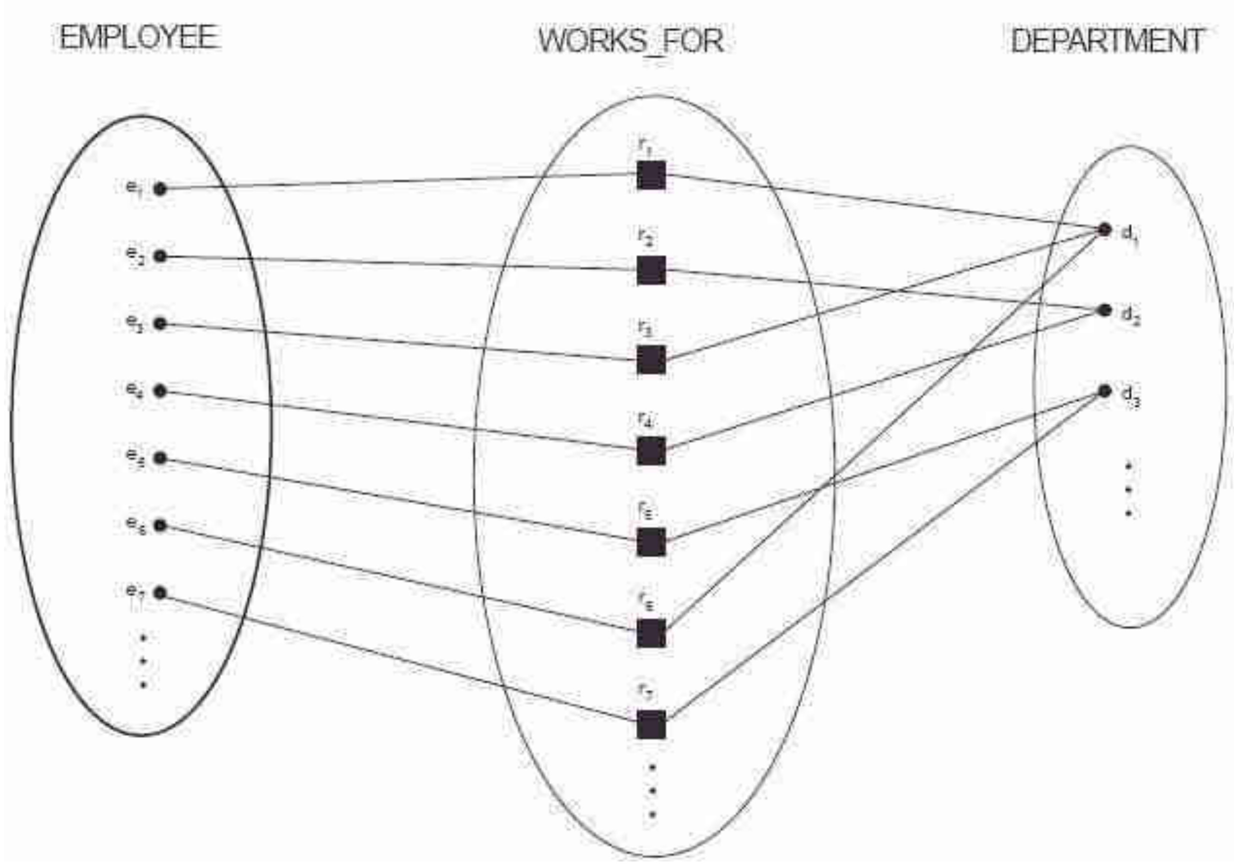
A : Attribute
E : Entity set
V : Value set
$V = P(V_1) \times \ldots \times P(V_n)$ for composite attributes
A(e) denotes the value of attribute A for entity e

- **Relationship:** The reference between two tables.
- **Relationship type:** Set of associations between n entities .

Note: Relationship type R among n entity types E1, ..., En defines a set of associations among entities from these types. Each association will be denoted as (e1, ..., en) where ei belongs to Ei, $1 \le i \le n$.

- **Relationship degree:** The number of entity types participates in a relationship type.
- **Binary Relationship degree:** When two entity types participates in a relationship type.
- **Ternary Relationship degree:** When three entity types participates in a relationship type.

Figure 3.9 Some instances of the WORKS_FOR relationship between EMPLOYEE and DEPARTMENT.



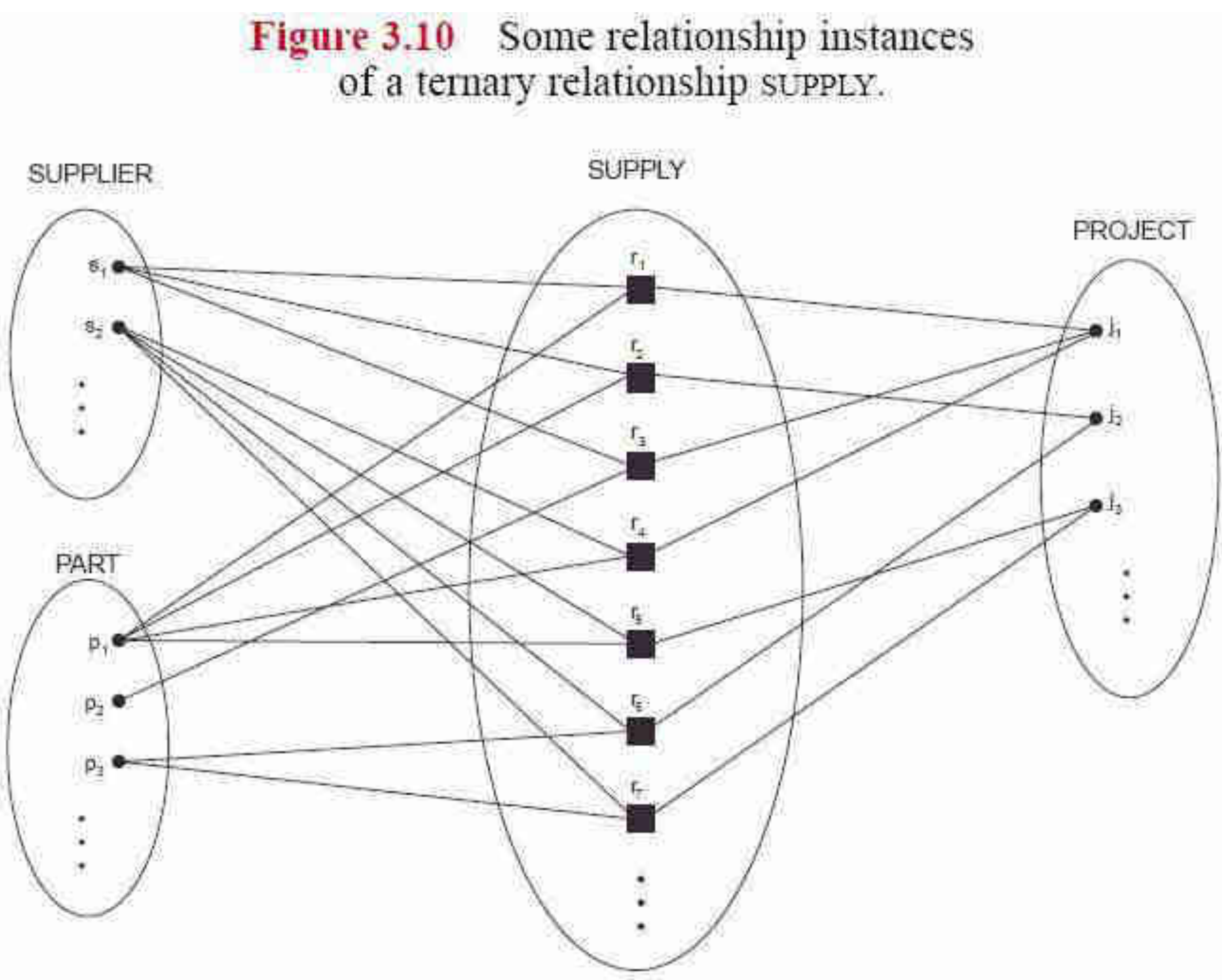
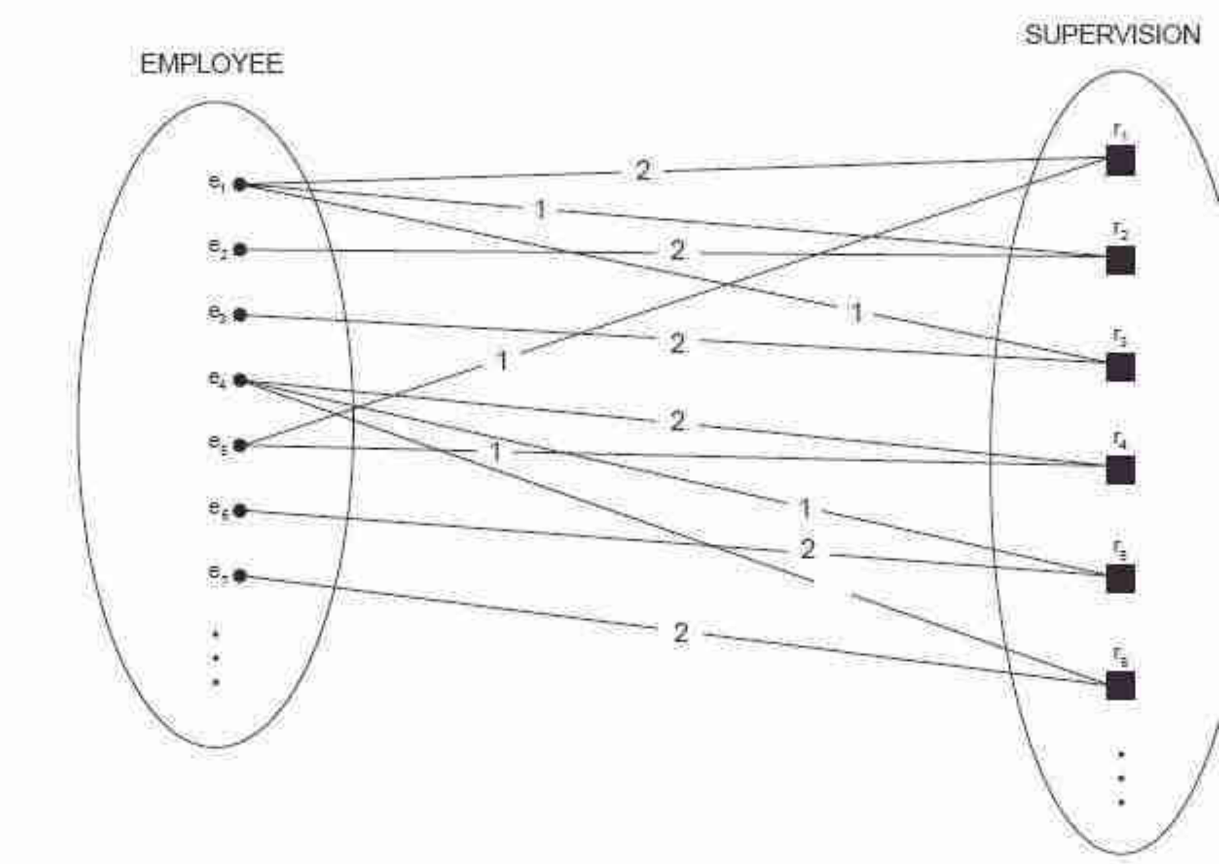


Figure 3.10 Some relationship instances of a ternary relationship SUPPLY.

- **Role:** Each entity participating in a relationship has a role.
  E.g. Employee plays the role of worker and department plays the role of employer in the WORKS_FOR relationship.
  Note: Role names are more important in recursive relationships.

- **Recursive relationship:** If one entity plays two roles then the relationship is called recursive relation.

Figure 3.11 The recursive relationship SUPERVISION, where the EMPI
entity type plays the two roles of supervisor (1) and supervisee (2



- Structural constraints on relationships: We have two relationship constraints.

(a) Cardinality ratio constraint (1-1, 1-N, M-N)
(b) Participation constraint 1. Total participation (existence ependency):All the tuples of the participating entity relates with the tuples of related entity represented with double line. 2. Partial participation : Some tuples of the participating entity relates with all the tuples of related entity represented with single line. - Note: Participation constraint is represented by (min,max) pair. If min value = 0 then the is patial paticipation. If min value >= 0 then the is total paticipation.

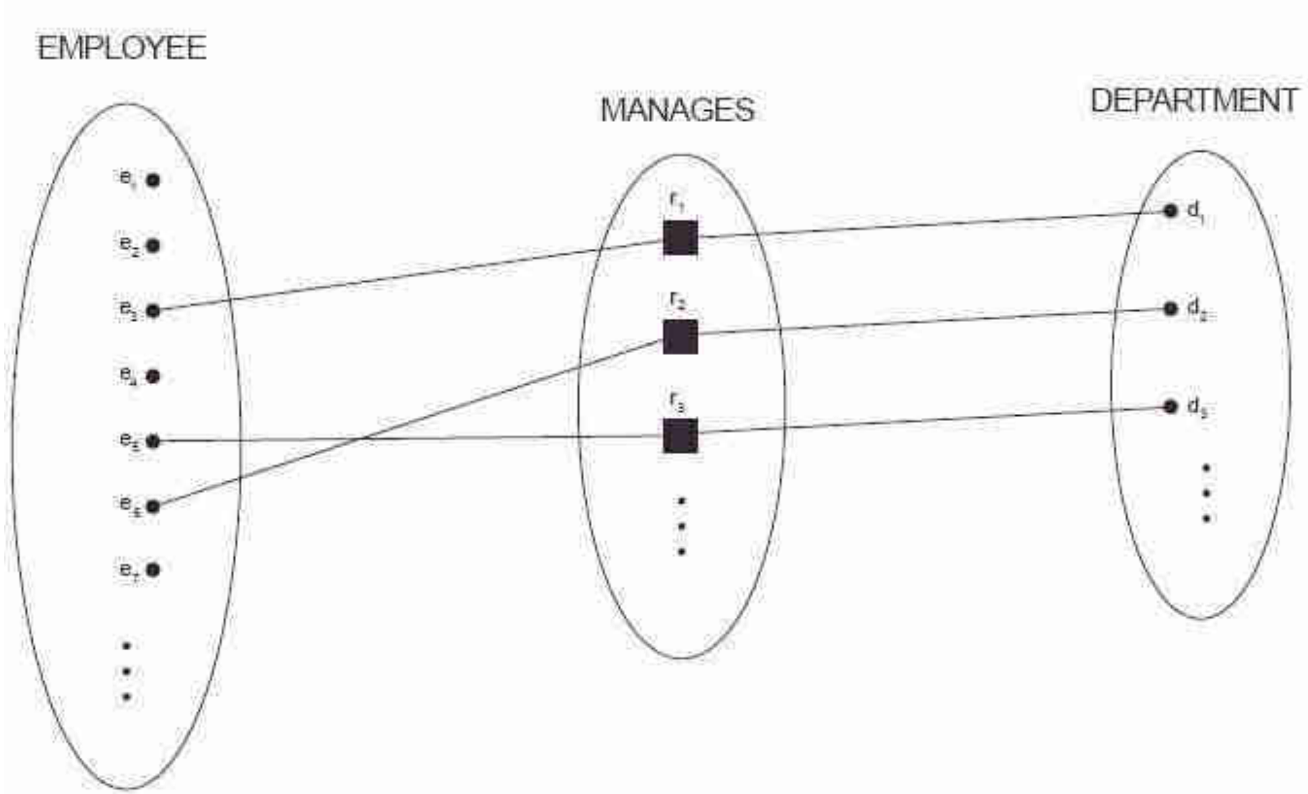


Figure 3.12 The 1:1 relationship MANAGES, with partial participation of employee and total participation of DEPARTMENT.

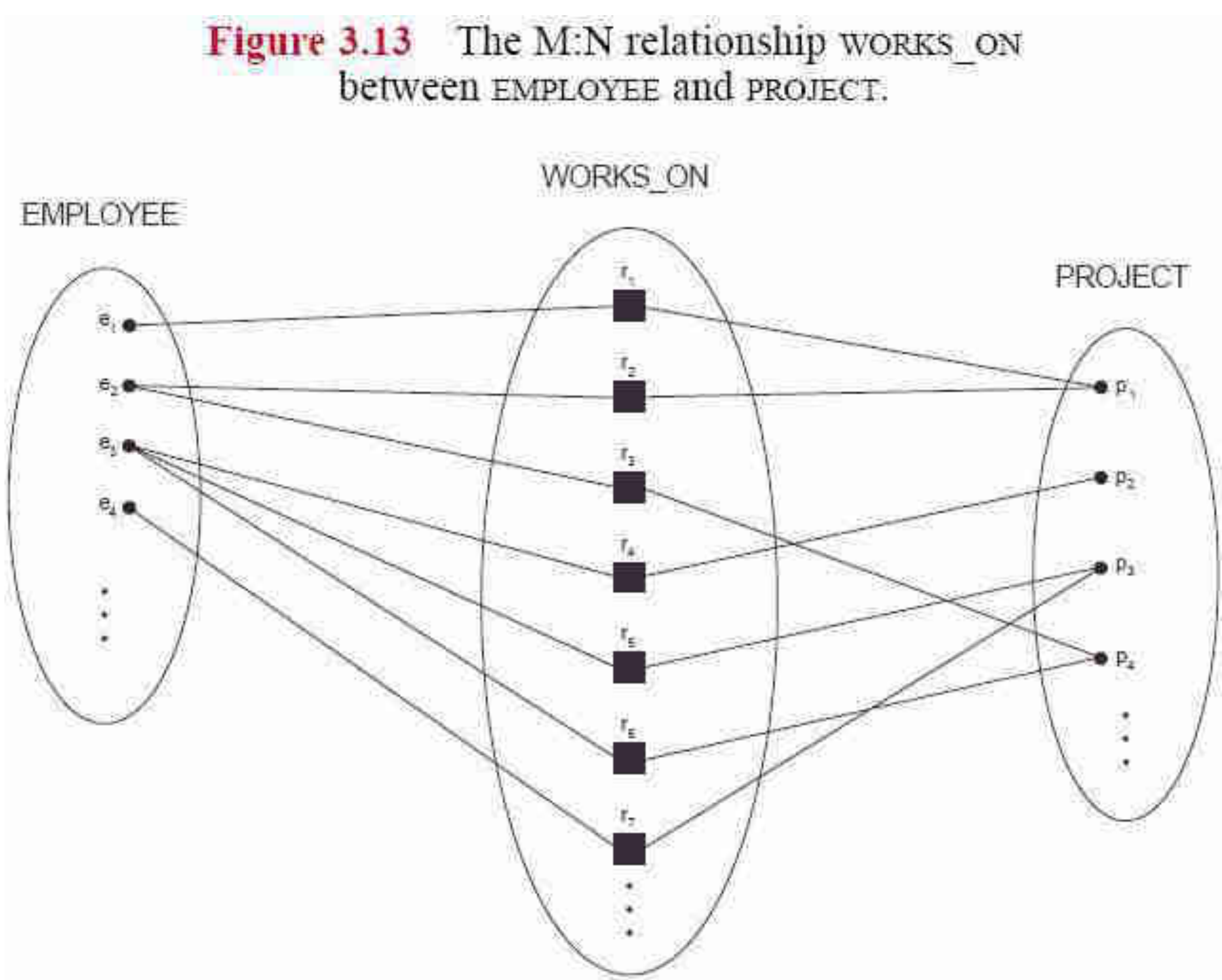


Figure 3.13 The M:N relationship WORKS_ON between EMPLOYEE and PROJECT.

- **Attributes of relationships:**

1. **Note: Relationship types can have attributes, similar to entity types.**

E.g. Hours attribute for WORKS_ON relationship

2. **If relationship is 1-N or 1-1, these attributes can be be migrated to the entity sets involved in the relationship.**
   a) **1-N: migrate to N side**
   b) **1-1: migrate to either side**

3.3 Weak Entity types	IMP

- **Weak entity: Entity that do not have key attribute.**

E.g. DEPENDENT

- **Weak entity types: Entity Types that do not have key attributes.**

- **Weak entities are identified by being related to other entity sets called identifying owner. This relationship is called identifying relationship.**

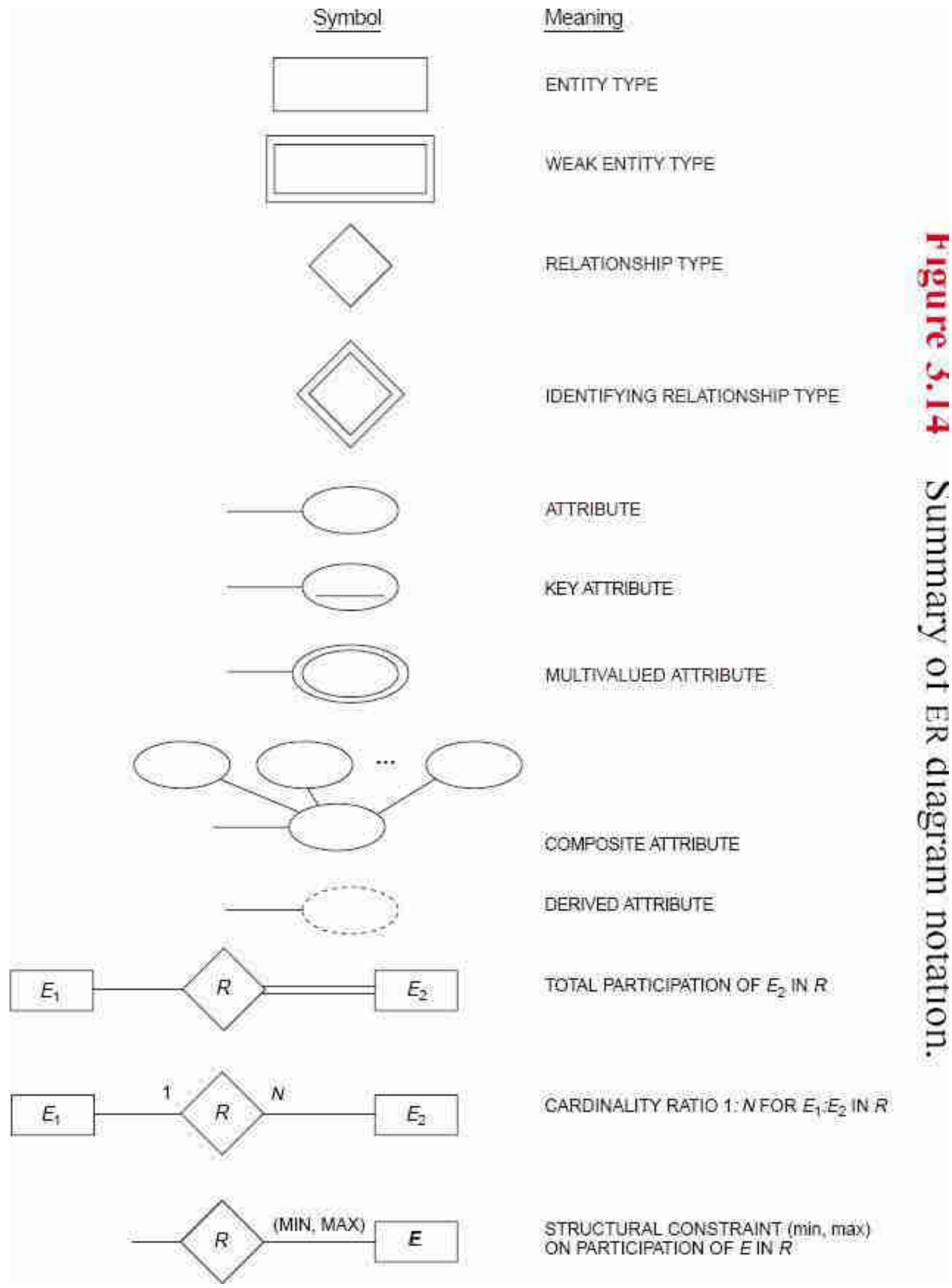
- **Partial key: Attributes that uniquely identify entities within the identifying relationship.**

  (or)

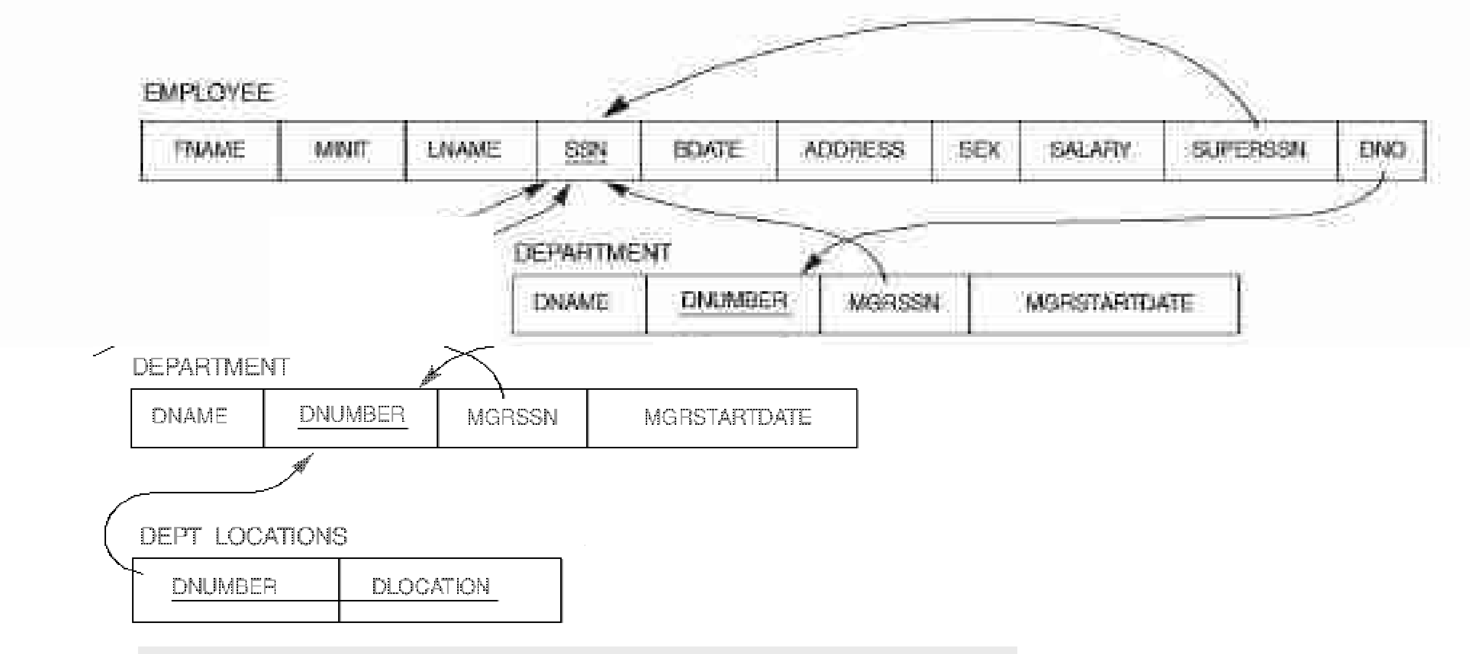
  A partial key is a key used to distinguish one record with another record but not guarentted to be a key when new records are inserted.

- **A weak entity type always has TOTAL participation.**

3.4 ER schema constructs (or) Symbols to write ER diagrams — IMP

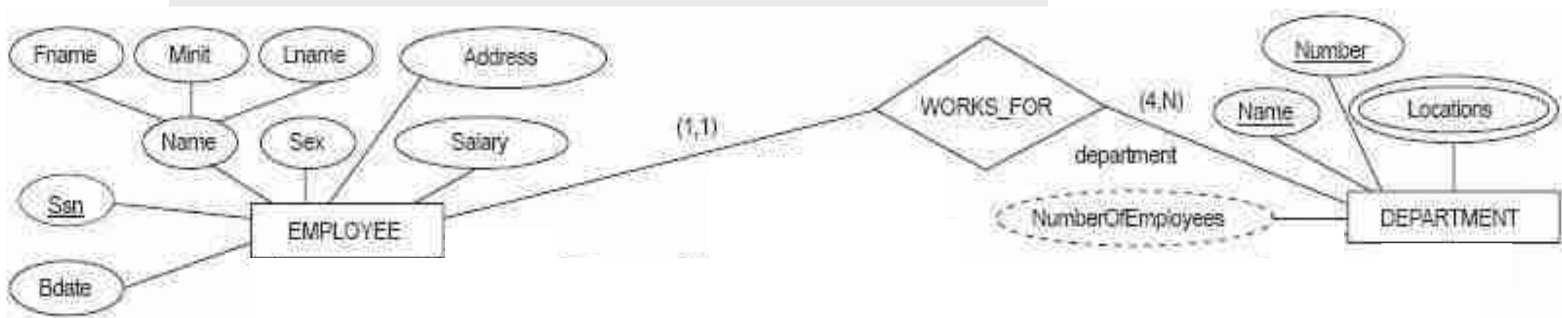


EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John		Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin		Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia		Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer		Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh		Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce		English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad		Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James		Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1



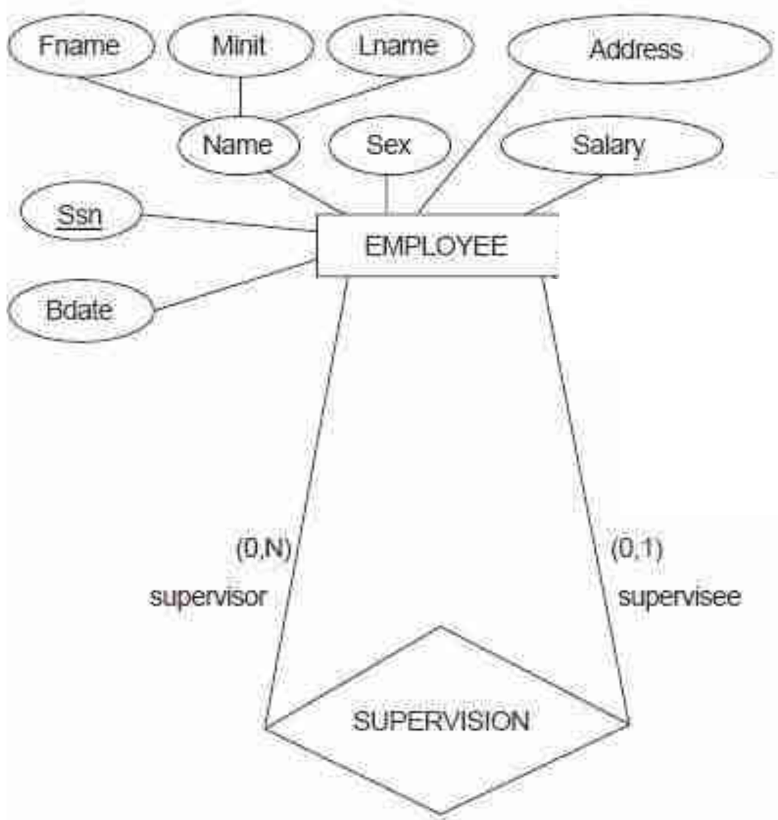
- **Constraints:**

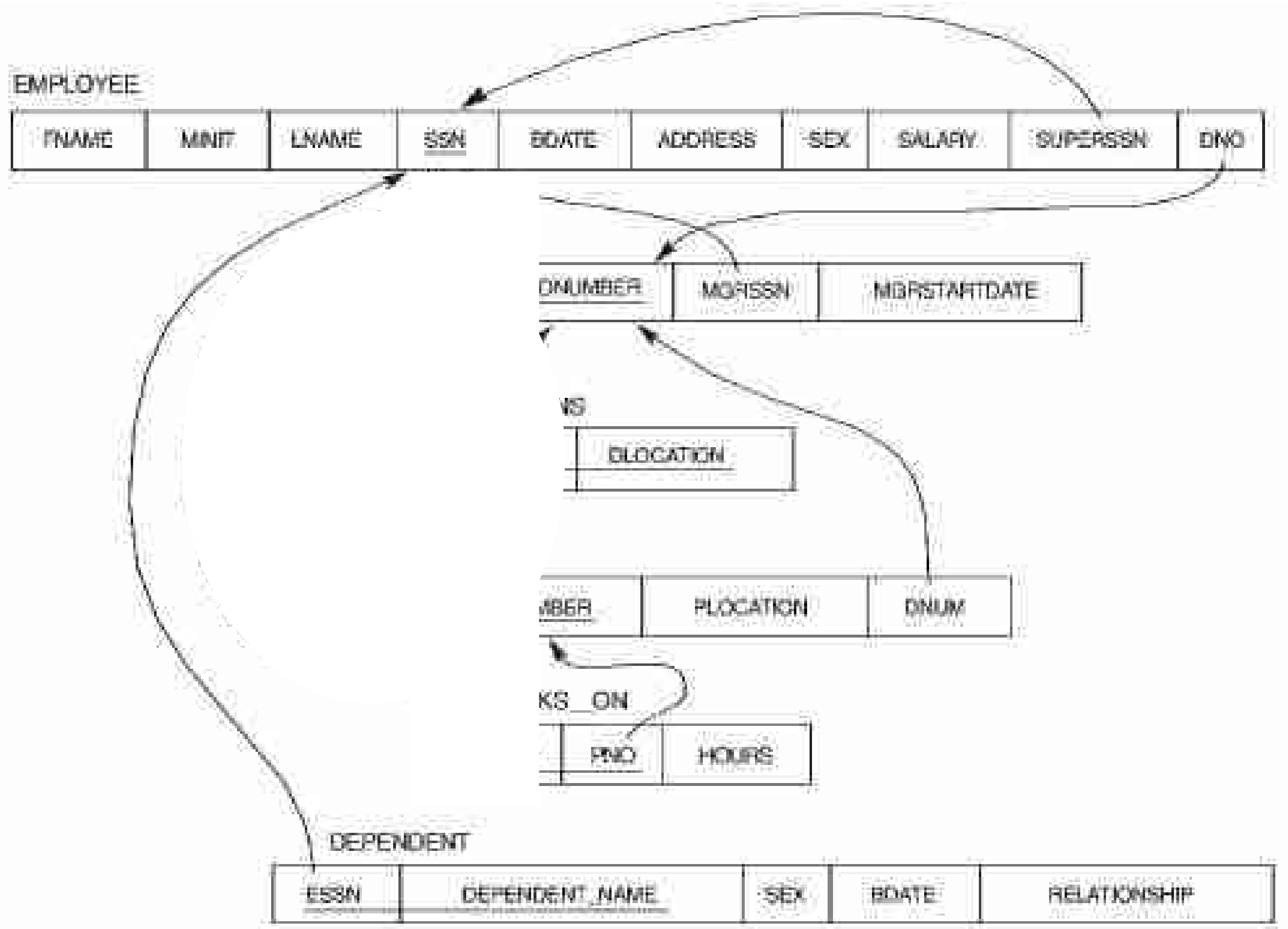
1. **An employee work for one department.**
2. **Department may have any number of employees.**
3. **Adepartment can be located any number of locations.**



- Constraints:

1. An employee who is manager is supervisor.
2. Manager supervise any number of employees.
3. A manager is supervised by another manager.





- **Constraints**

1. **An employee may not have dependents.**
2. **An employee may have any number of dependents.**
3. **Every dependent must belongs to an employee.**

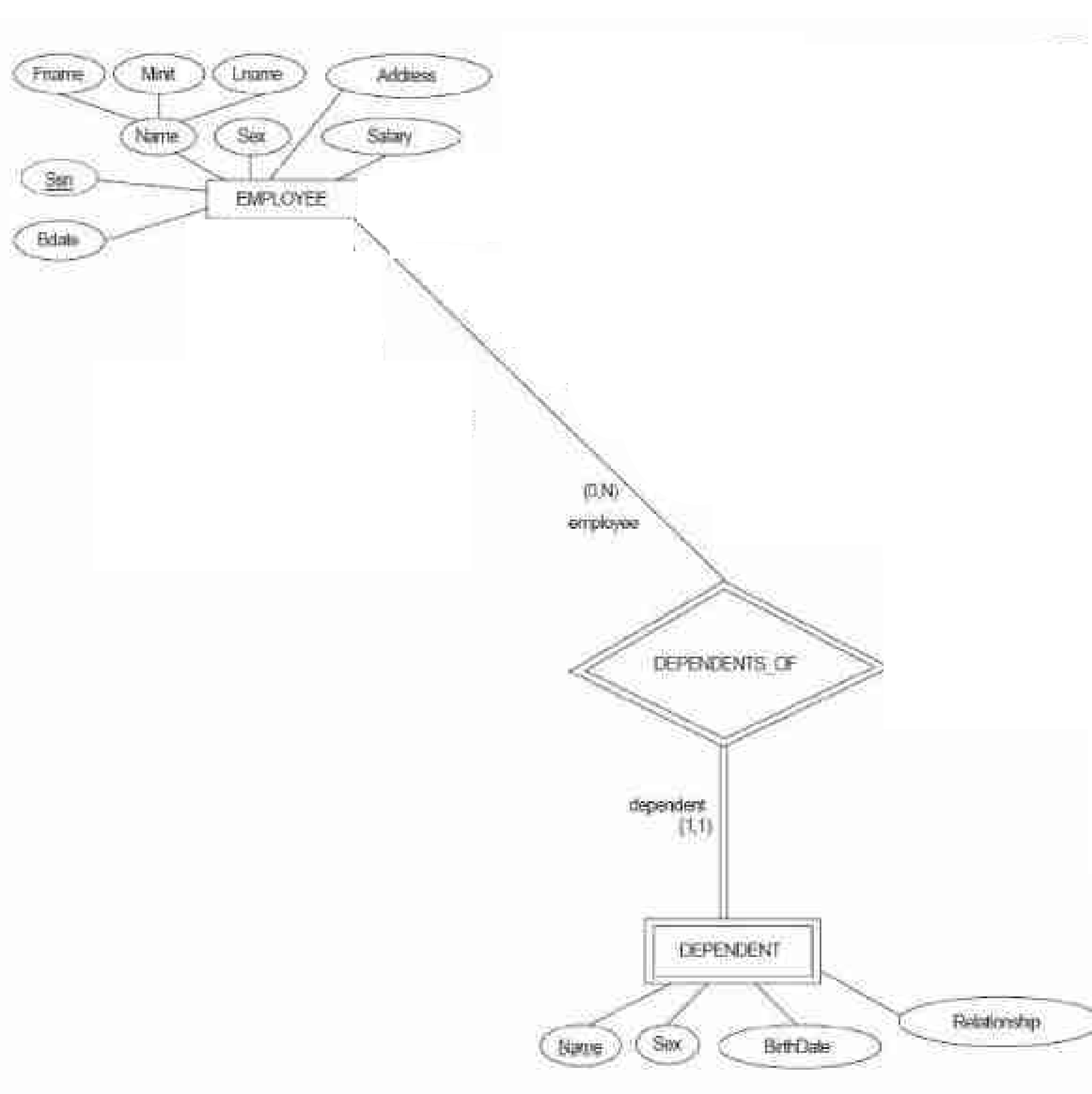
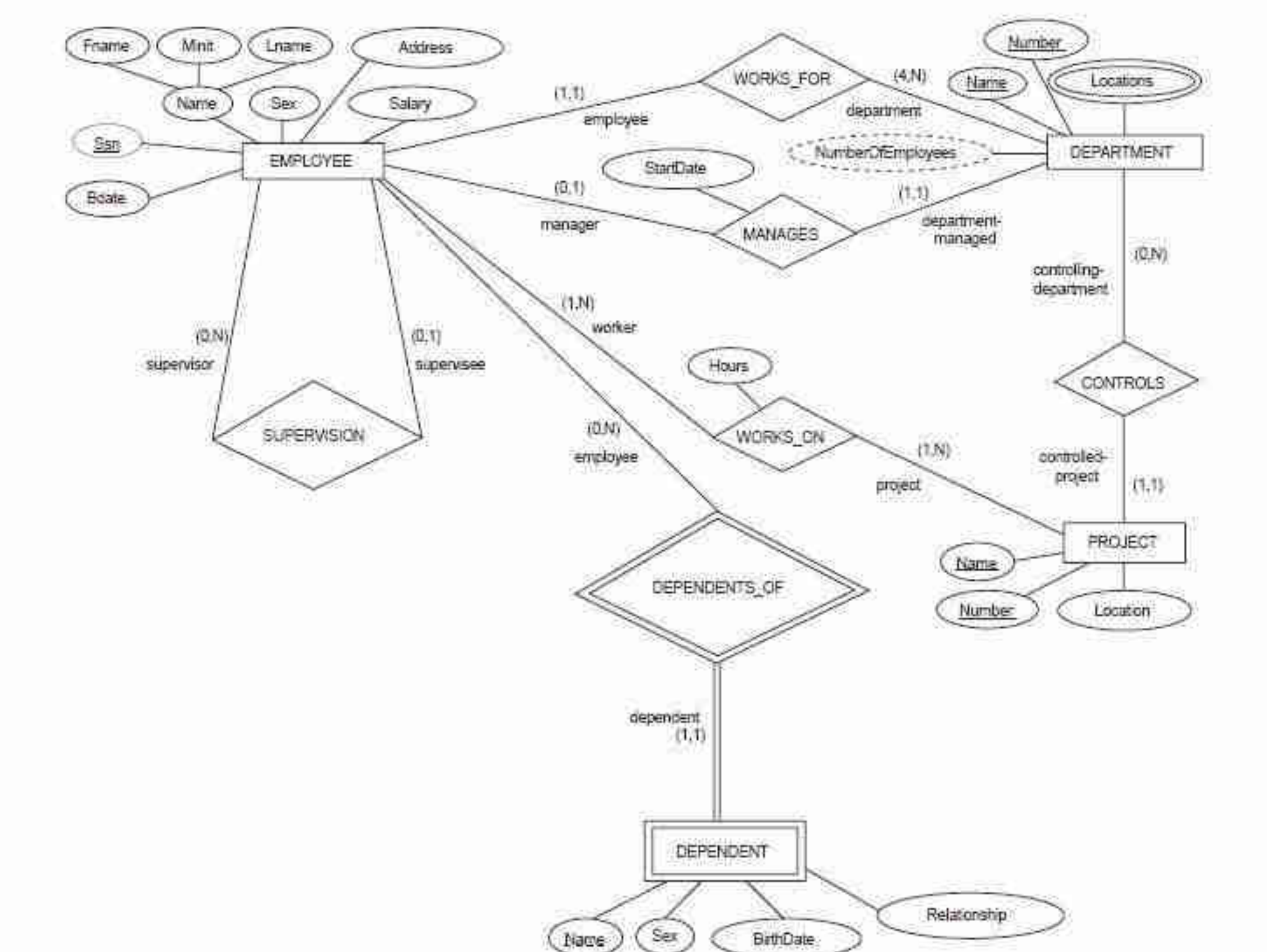

Fname
Minit
Lname
Address
Name
Sex
Salary
Ssn
EMPLOYEE
Bdate
(0,N)
employee
DEPENDENTS_OF
dependent
(1,1)
DEPENDENT
Name
Sex
BirthDate
Relationship

Figure 3.15 ER diagram for the COMPANY schema, with all role names included and with structural constraints on relationships specified using the alternate notation (min, max).



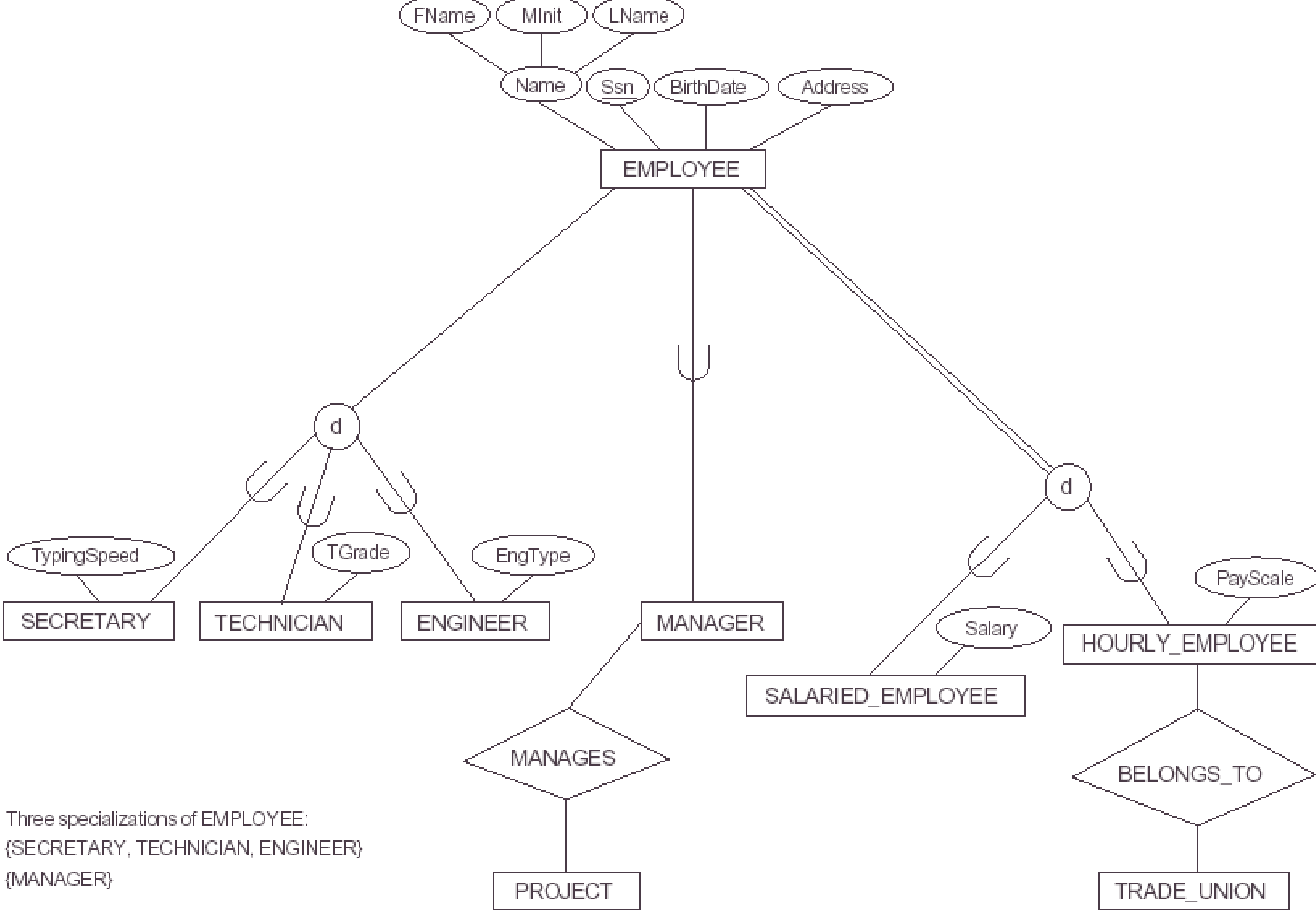
- **Relationship attribute:** **Attribute derived from the relationship.**
E.g. Hours, StratDate

Chapter-4
Enhanced Entity-Relationship & Object Modeling

- **Super class:** **A parent class is super class.**
- **Sub class:** **A child class is called sub class.**
- **Inheritance:** **The mechanism of applying the properties of super class to sub class is called inheritance.**

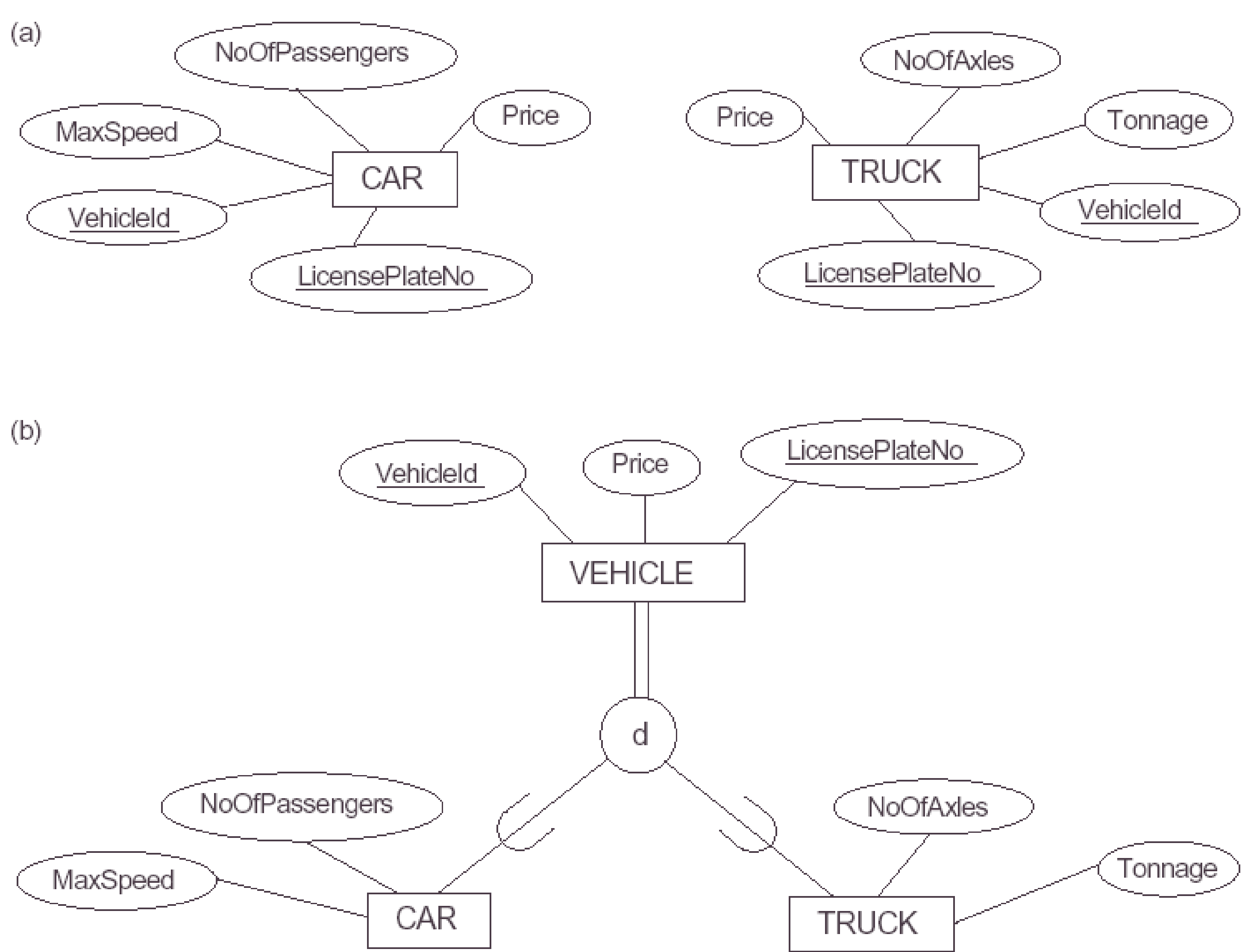
4.1 Specialization | IMP

- **Specialization:** **The process of defining a set of subclasses of an entity type.**
  E.g. Top-down hierarchy.



4.2 Generalization | **IMP**

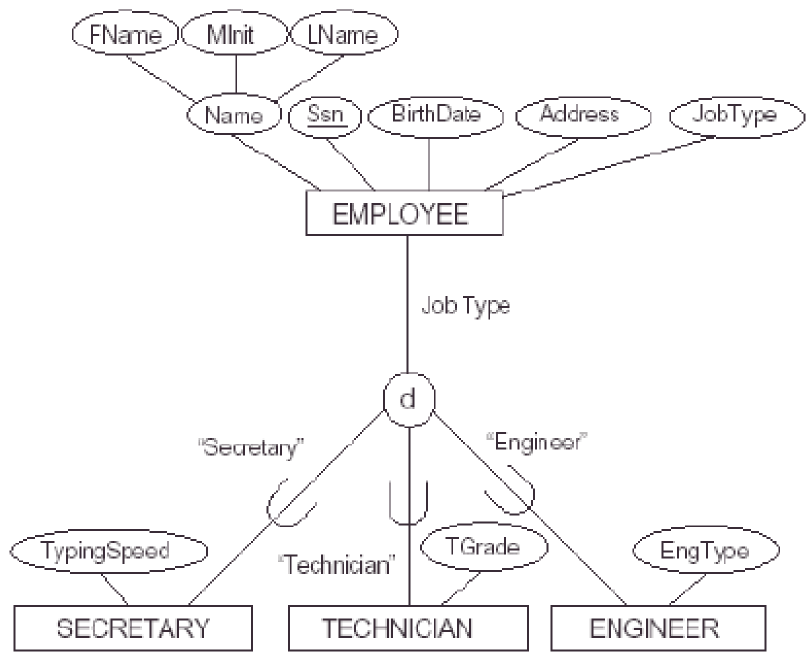
- **Generalization: The process of inheriting the features of super class from the subclass is called generalization.**
  E.g. Bottom_up hierachy.



4.3 Constraints on Specialization and Generalization	IMP

(a) Disjointness constraint: Subclasses of the specialization must be disjoint.

Figure 4.4 An attribute-defined specialization on the JobType attribute of EMPLOYEE.



(b) Completeness constraint: Specifies every entity in the super class must be a member of some subclass or not.

- **Completeness constraint may be total or partial.**

1. **Total specialization: Every entity in the super class must be a member of some sub class.**

   E.g. EMPLOYEE must be either a HOURLY_EMPLOYEE or a SALARIED_EMPPLOYEE.

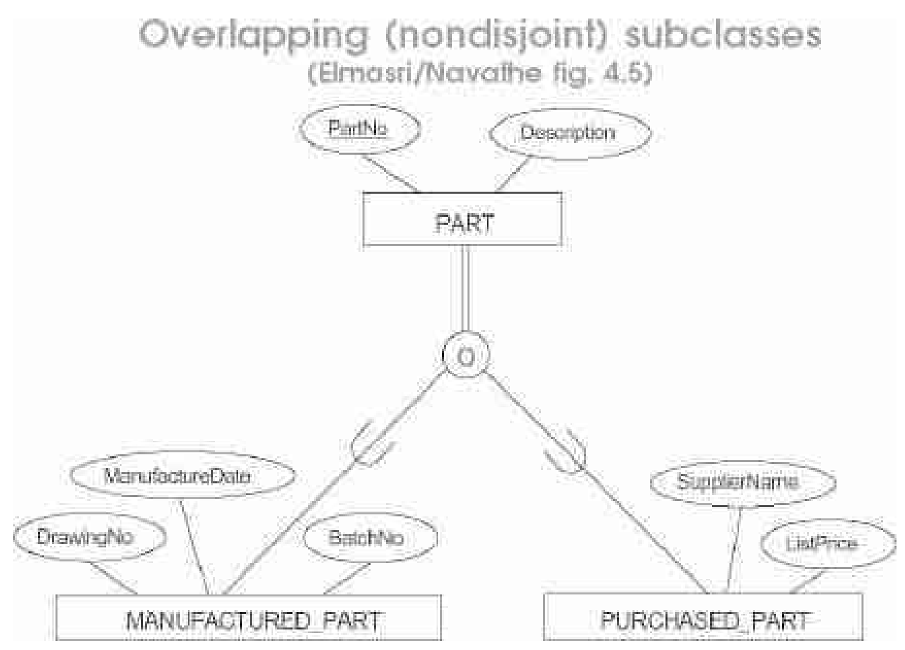
2. **Partial specialization: An entity not belonging to any of the subclass.**

   E.g. Some EMPLOYEE entities do not belong to any of subclasses { SECRETARY,ENGINEER,TECHNICIAN }.

FName
MInit
LName
Name
Ssn
BirthDate
Address
EMPLOYEE
d
d
TypingSpeed
TGrade
EngType
SECRETARY
TECHNICIAN
ENGINEER
MANAGER
Salary
PayScale
HOURLY_EMPLOYEE
SALARIED_EMPLOYEE
MANAGES
BELONGS_TO
PROJECT
TRADE_UNION

Three specializations of EMPLOYEE:
{SECRETARY, TECHNICIAN, ENGINEER}
{MANAGER}

(c) Overlap constraint: Same entity may be member of more than one sub class of the specialization.



- We have four possible constraints on specialization

(a) Disjoint , total
(b) Disjoint, partial
(c) Overlapping, total
(d) Overlapping, partial

- Insertion and deletion rules apply to specialization

1. Deleting an entity from super class implies that it is automatically deleted from all the subclasses.

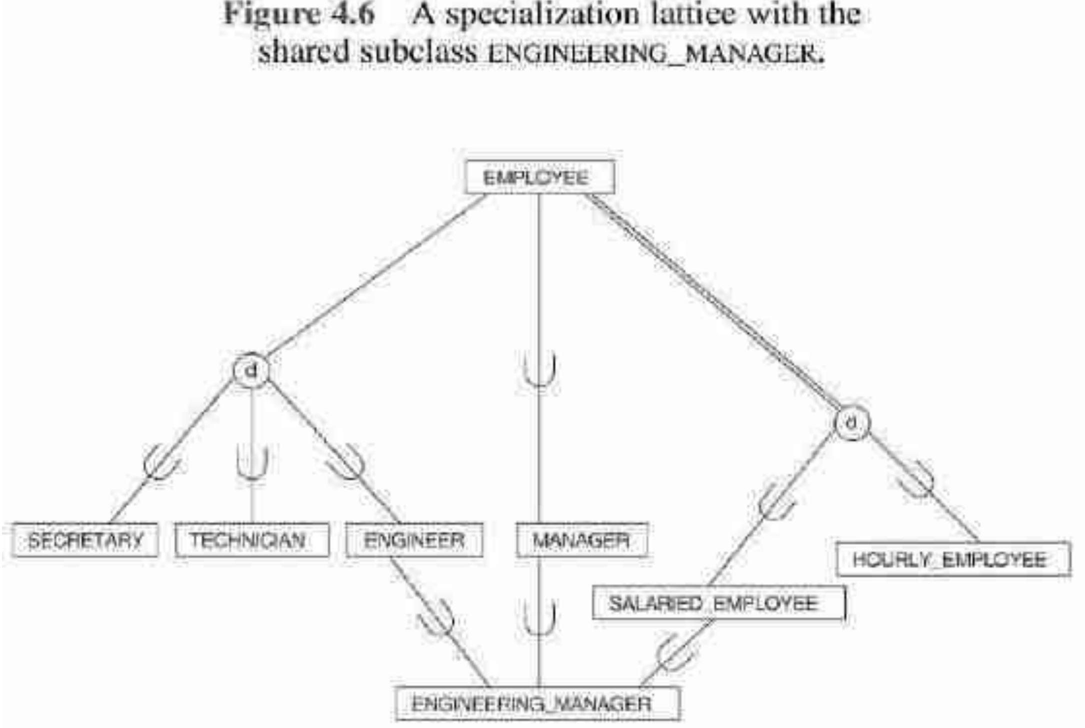
2. Inserting an entity in a super class imply that the entity mandatory inserted in all the subclasses.

3. Inserting an entity in a super class of a total specialization implies that the entity is mandatory inserted in at least one of the subclasses of the specialization.

4.4 Specialization and Generalization Lattice

- If a subclass itself may have further subclasses specified on it, forming a hierarchy called lattice of specialization.

Figure 4.6 A specialization lattice with the shared subclass ENGINEERING_MANAGER.



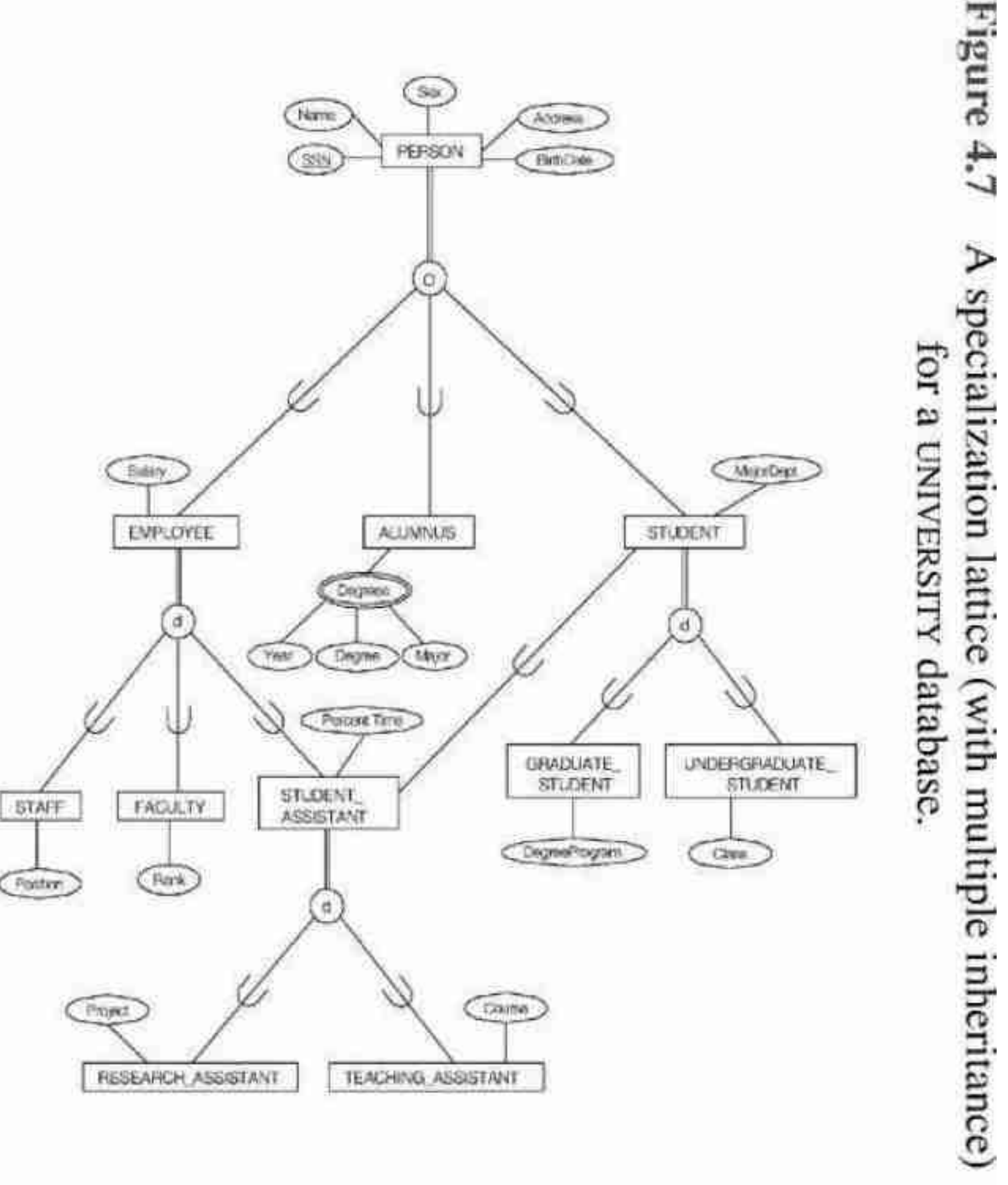
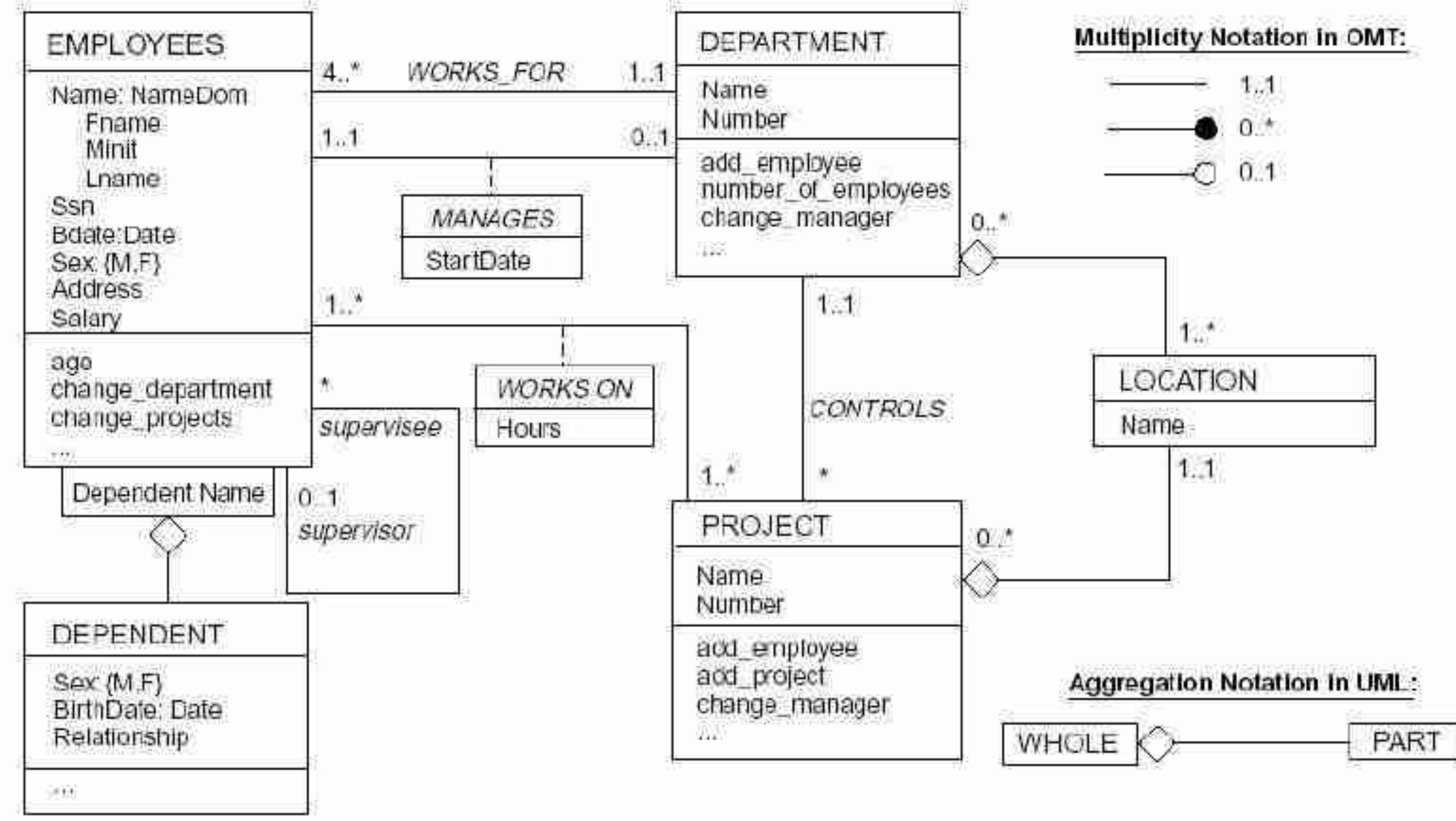


Figure 4.7 A specialization lattice (with multiple inheritance) for a UNIVERSITY database.

A UML conceptual schema
(Elmasri/Navathe fig. 4.11)



Chapter –5
Record storage & Primary File Organizations

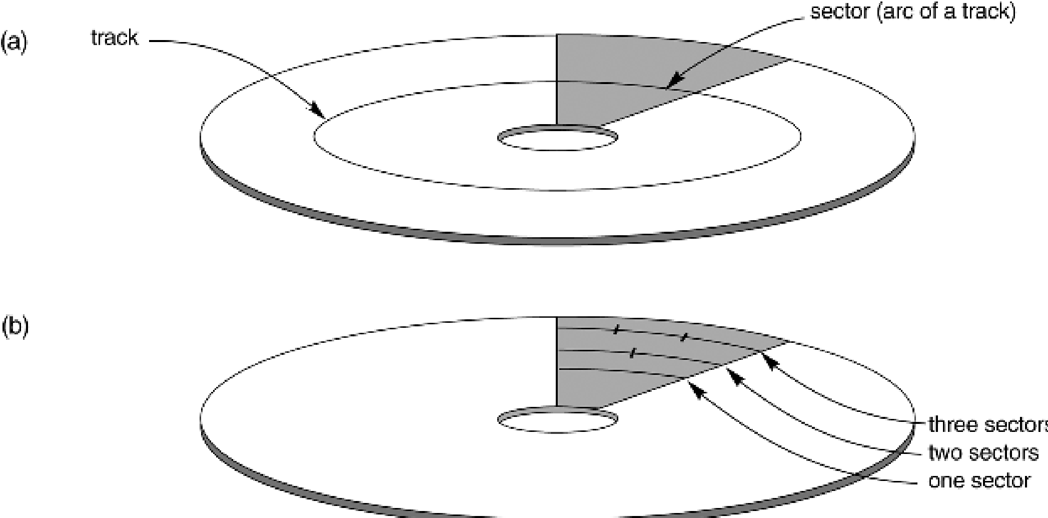
5.1 Memory Hierarchies

Priority	Device
1	High speed registers
2	Cache (or) Static RAM
3	Main memory (or) DRAM
4	Optical disk (or) CDROM
5	Magnetic disk
6	Magnetic tap

5.2 Secondary Storage Devices

1. Magnetic Disk

- Magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material
- Both sides of the disk are used
- Several disks may be stacked on one spindle with Read/Write heads available on each surface
- All disks rotate together at high speed and not stopped or started for access purpose
- Used to store large amount of data .
- Basic unit of storage is BIT .
- BIT is 1 by magnetizing area on disk .
- Character: Group of bits.
- Capacity: Number of bytes it can store.
- Number of tracks: ranges from few hundred to few thousand.
- Capacity of each track: ranges from tens of Kbytes to 150 Kbytes.
- Formatting: Tracks are divided into equal sized disk blocks called sectors.
- Sector size is fixed range from 512 to 4096 bytes.



- Read/Write head is classified into
  1. Fixed head system.
  2. Moving head system.

- Rotate disk pack of cylinders ranging from **3600 and 7200 rpm.**

- **Access time:** Time required to access the data.

  t_A = Access time

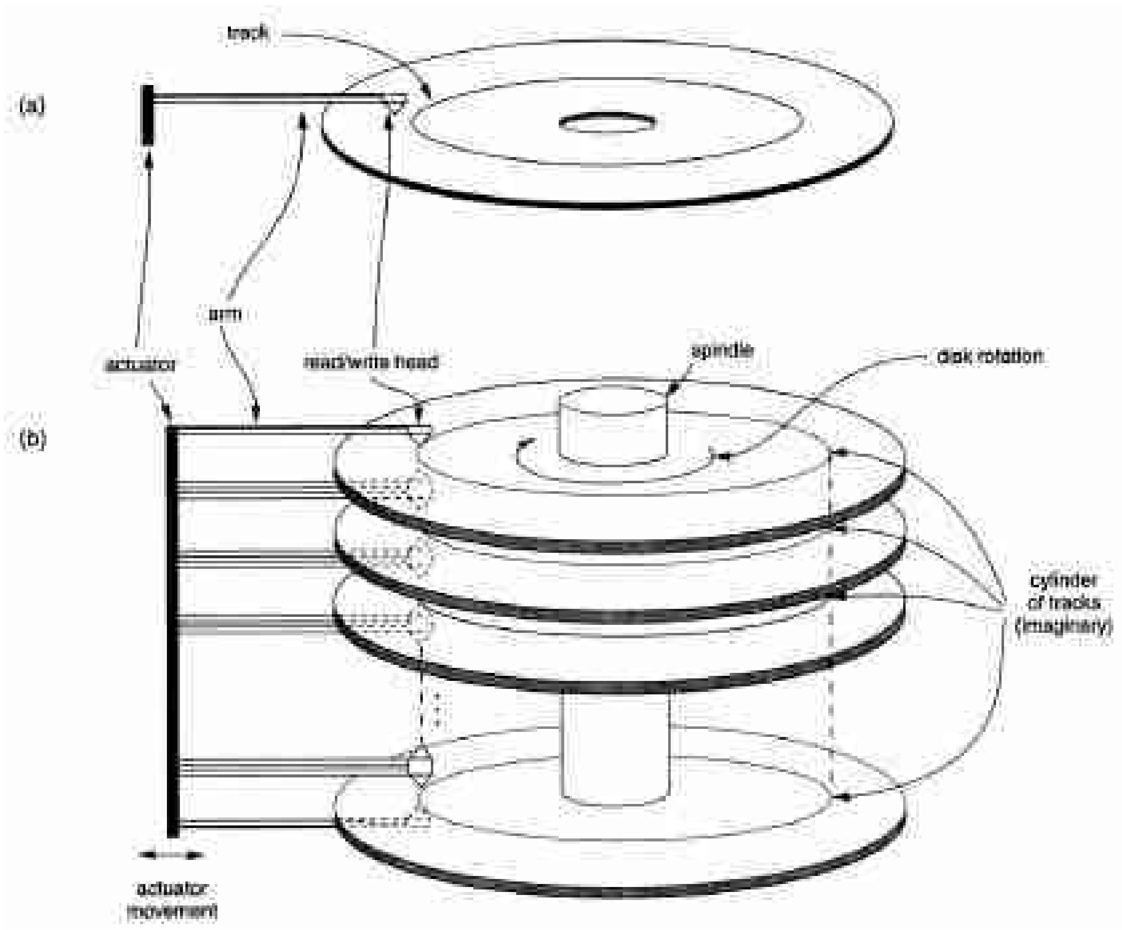
  $t_A = t_S + t_L$

- **Seek time:** Time required to position the read/write head on the **correct track** & **correct surface.**

  t_S = Seek time

- **Latency time:** Time required to locate the required disk block from the **beginning.**

  t_L = Latency time



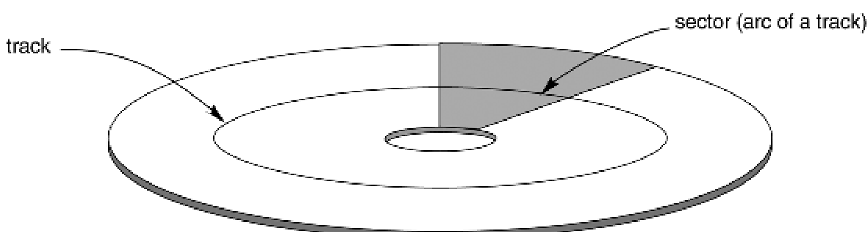
2. Floppy Disk

- A disk drive with removable disks are called a Floppy Disk
- The disks used with a Floppy disk drive are small disks made of plastic coated with magnetic recording material
- There are two sizes commonly used, with diameters of 5.25 and 3.5 inches
- The 3.5 inch disks are smaller and can store more data than 5.25 inch disks
- Floppy disks extensively used in personal computer as a medium for distributing software to computer users
- Basic unit of storage is BIT .
- BIT is 1 by magnetizing area on disk .

- Character: Group of bits.
- Capacity: Number of bytes it can store.
- Density : Single density and Double density
- Data stored in: Tracks & Sectors
- Track: Data is placed in concentric magnetic circles called tracks
- Sector :Each track is divided into storage units called sectors
  - FAT uses 512 byte sectors exclusively
  - The number of Sectors per Track vary depending on the media and format
  - Physical Sector Numbering starts with the number 1at the beginning of each track and side

- Number of Tracks: 80 in 3.5 inch floppy disk

Disk Formats - 3.5”		
	Low Density	High Density
Tracks	80	80
Sectors per track	9	18
Bytes per sector	512	512
Total capacity	720K	1.44M



- Formatting: Tracks are divided into equal sized disk blocks called sectors.

- Access time: Time required to access the data.

t_A = Access time

$t_A = t_S + t_L$

- **Seek time:** Time required to position the read/write head on the correct track & correct surface.

  t_S = Seek time

- **Latency time:** Time required to locate the required disk block from the beginning.

  t_L = Latency time

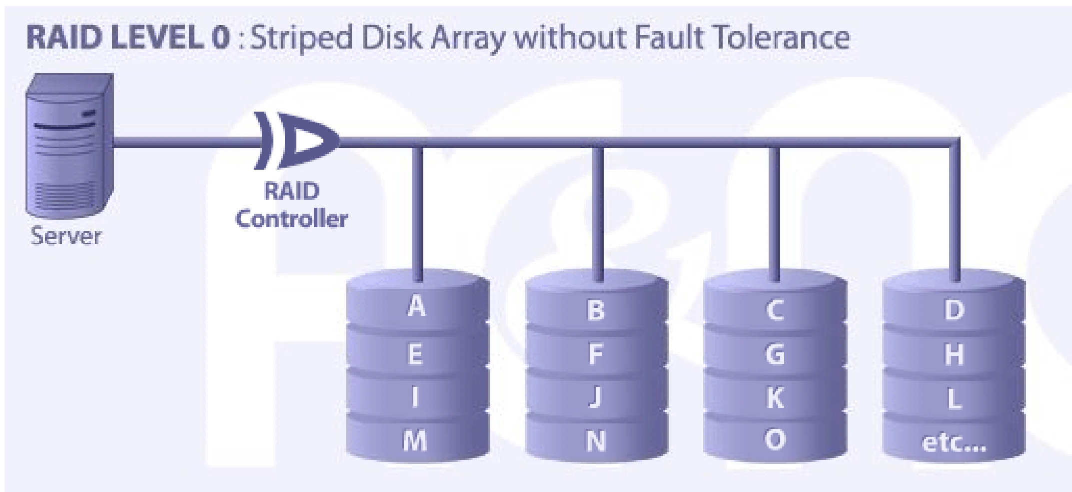
3. Magnetic Tape

- Magnetic tape transport consists of the electrical, mechanical and electronic components provide the parts and control mechanism for magnetic tape unit
- The tape is a strip of plastic coated with magnetic recording media
- Bits are recorded as magnetic spots on the tape along with several tracks
- Usually, seven or nine bits are recorded simultaneously to form a character together with a parity bit.
- The information is recorded in blocks referred to as records
- Each record on tape has an identification bit pattern at the beginning and end. By recording the bit pattern at the beginning, the tape control identifies the record number.
- By reading the bit pattern at the end of the record, the control recognizes the beginning of a gap
- A tape unit is addressed by specifying the record number and number of characters in the record
- Record may be fixed length or variable length
- Sequential access.
- Read/Write head: Reads or writes data on tape.
- Storage: 1600 to 6250 bytes per inch.
- Inter block gap: 0.6 inches.

Block		Block		Block		Block		Block		Block

5.3 RAID VIMP

- Redundant Array Independent Disk



- RAID Level 0 requires a minimum of 2 drives to implement.

- **Characteristics & Advantages:**

1. RAID 0 implements a striped disk array, the data is broken down into blocks and each block is written to a separate disk drive.

2. I/O performance is greatly improved by spreading the I/O load across many channels and drives.

3. Best performance is achieved when data is striped across multiple controllers with only one drive per controller.

4. No parity calculation overhead is involved.

5. Very simple design.

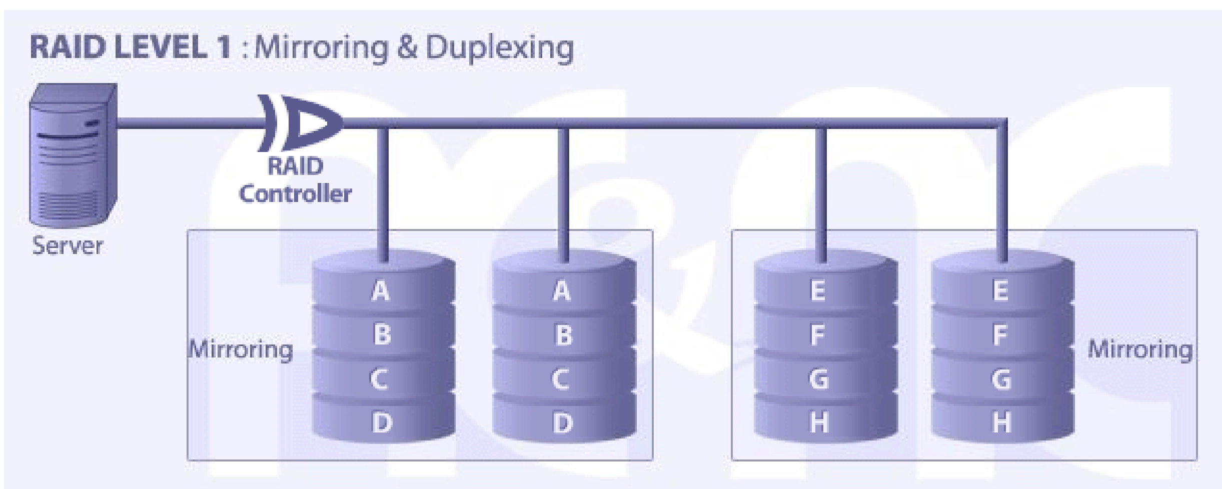
6. Easy to implement.

- **Disadvantages**

1. Not a "True" RAID because it is NOT fault-tolerant.

2. The failure of just one drive will result in all data in an array being lost.

3. Should never be used in mission critical environments.



- For Highest performance, the controller must be able to perform two concurrent separate reads per mirrored pair or two duplicate Writes per mirrored pair.

- RAID Level 1 requires a minimum of 2 drives to implement.

- Characteristics & Advantages:

1. One Write or two Reads possible per mirrored pair.

2. Twice the Read transaction rate of single disks, same Write transaction rate as single disks.

3. 100% redundancy of data means no rebuild is necessary in case of a disk failure, just a copy to the replacement disk.

4. Transfer rate per block is equal to that of a single disk.

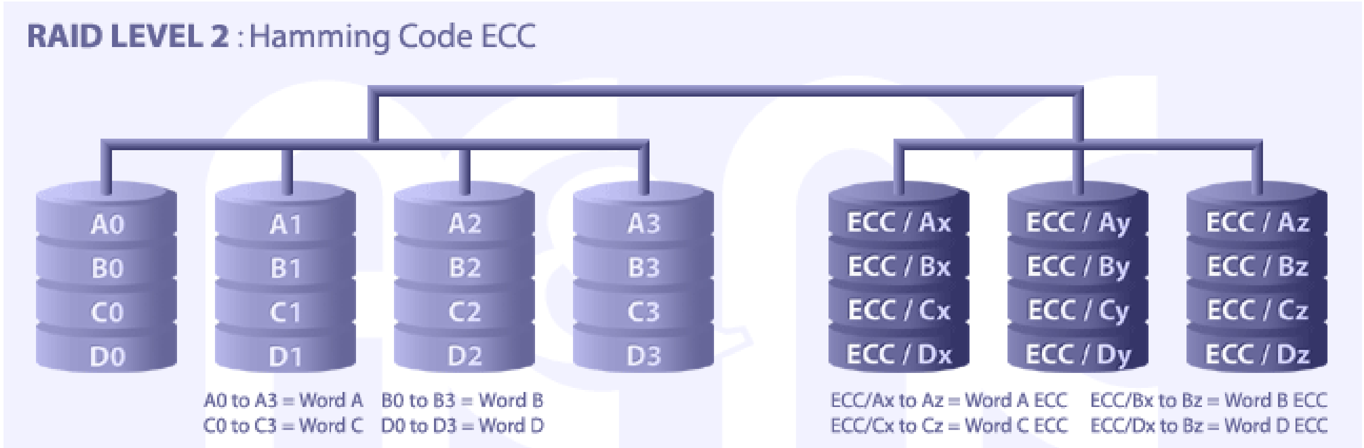
5. Under certain circumstances, RAID 1 can sustain multiple simultaneous drive failures.

- Disadvantages:

1. Highest disk overhead of all RAID types (100%) – inefficient

2. Typically the RAID function is done by system software, loading the CPU/Server and possibly degrading throughput at high activity levels. Hardware implementation is strongly recommended

3. May not support hot swap of failed disk when implemented in "software"

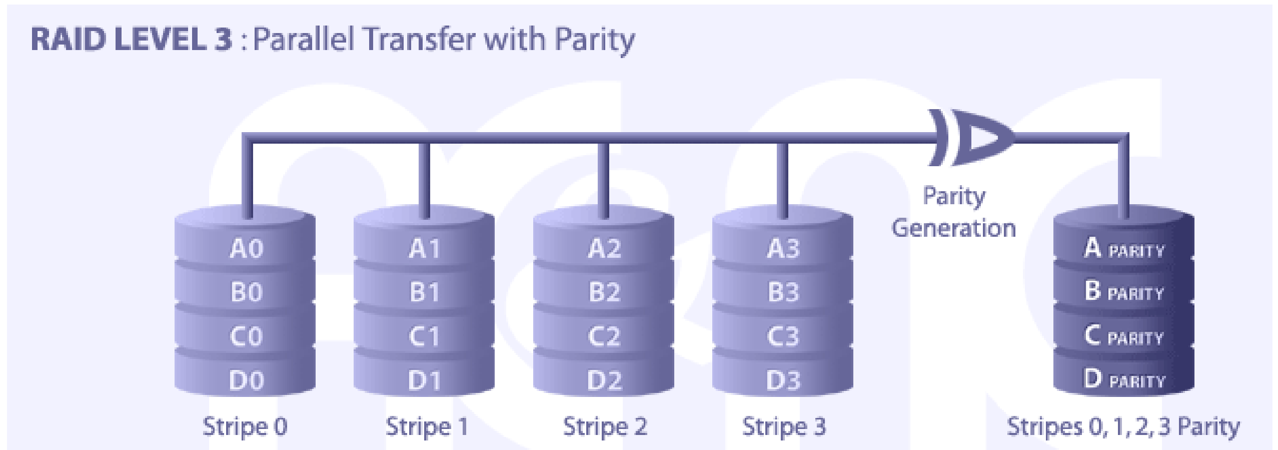


- **Characteristics & Advantages:**

1. Each bit of data word is written to a data disk drive (4 in this example: 0 to 3).
2. Each data word has its Hamming Code ECC word recorded on the ECC disks.
3. On Read, the ECC code verifies correct data or corrects single disk errors.

- **Disadvantages:**

1. Very high ratio of ECC disks to data disks with smaller word sizes – inefficient.
2. Entry level cost very high - requires very high transfer rate requirement to justify
3. Transaction rate is equal to that of a single disk at best (with spindle synchronization)
4. No commercial implementations exist / not commercially viable.



- **The data block is subdivided ("striped") and written on the data disks. Stripe parity is generated on Writes, recorded on the parity disk and checked on Reads.**

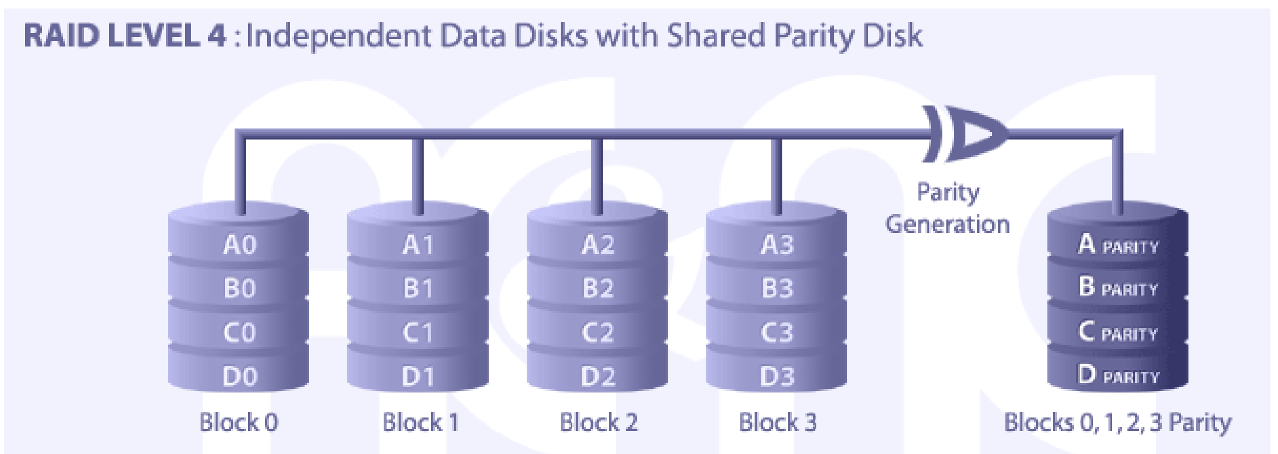
- **RAID Level 3 requires a minimum of 3 drives to implement.**

- **Characteristics & Advantages:**

1. **Very high Read data transfer rate.**
2. **Very high Write data transfer rate.**
3. **Disk failure has an insignificant impact on throughput.**
4. **Low ratio of ECC (Parity) disks to data disks means high efficiency.**

- **Disadvantages:**

1. **Transaction rate equal to that of a single disk drive at best (if spindles are synchronized).**
2. **Controller design is fairly complex.**
3. **Very difficult and resource intensive to do as a "software" RAID.**



- **Each entire block is written onto a data disk. Parity for same rank blocks is generated on Writes, recorded on the parity disk and checked on Reads.**

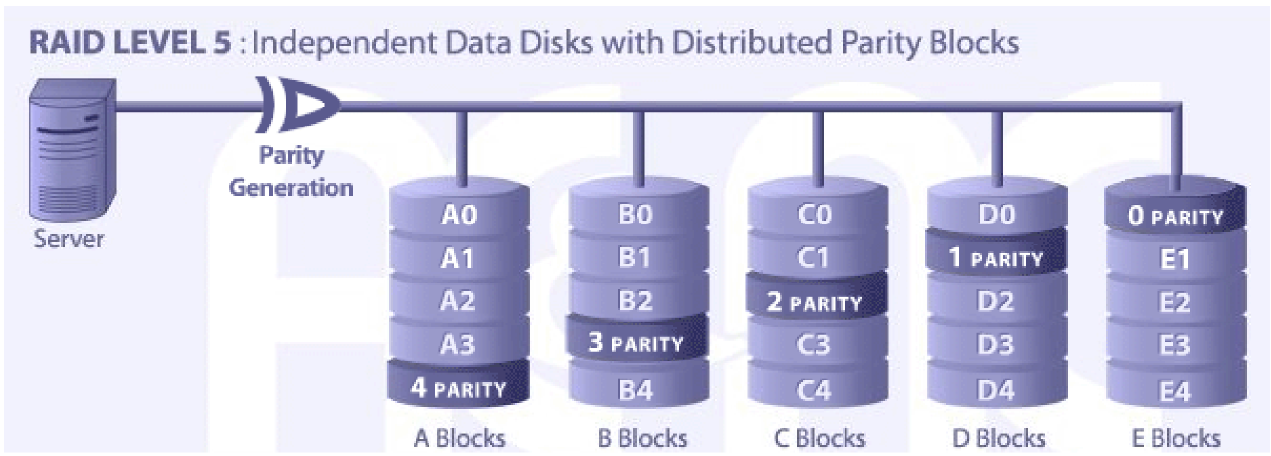
- **RAID Level 4 requires a minimum of 3 drives to implement.**

- **Characteristics & Advantages:**

1. **Very high Read data transaction rate.**
2. **Low ratio of ECC (Parity) disks to data disks means high efficiency.**
3. **High aggregate Read transfer rate.**

- **Disadvantages:**

1. **Quite complex controller design.**
2. **Worst Write transaction rate and Write aggregate transfer rate.**
3. **Difficult and inefficient data rebuild in the event of disk failure.**
4. **Block Read transfer rate equal to that of a single disk.**

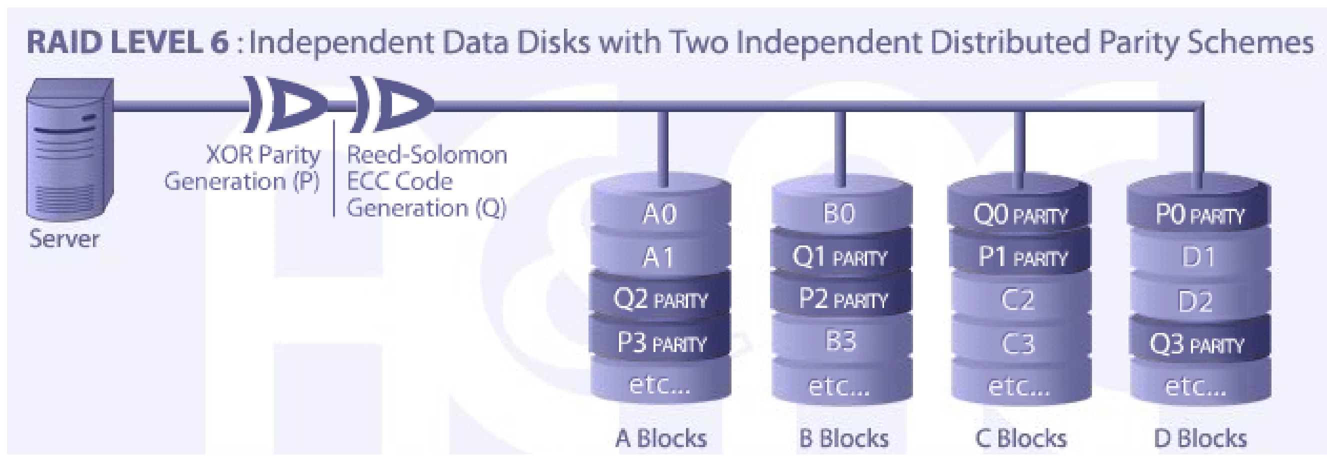


- **Each entire data block is written on a data disk; parity for blocks in the same rank is generated on Writes, recorded in a distributed location and checked on Reads.**
- **RAID Level 5 requires a minimum of 3 drives to implement.**
- **Characteristics & Advantages:**

1. **Highest Read data transaction rate.**
2. **Medium Write data transaction rate.**
3. **Low ratio of ECC (Parity) disks to data disks means high efficiency.**
4. **Good aggregate transfer rate.**

- **Disadvantages:**

1. **Disk failure has a medium impact on throughput.**
2. **Most complex controller design.**
3. **Difficult to rebuild in the event of a disk failure (as compared to RAID level 1).**
4. **Individual block data transfer rate same as single disk.**



- **Two independent parity computations must be used in order to provide protection against double disk failure. Two different algorithms are employed to achieve this purpose.**

- **RAID Level 6 requires a minimum of 4 drives to implement**

- **Characteristics & Advantages:**

1. **RAID 6 is essentially an extension of RAID level 5 which allows for additional fault tolerance by using a second independent distributed parity scheme (dual parity).**
2. **Data is striped on a block level across a set of drives, just like in RAID 5, and a second set of parity is calculated and written across all the drives; RAID 6 provides for an extremely high data fault tolerance and can sustain multiple simultaneous drive failures.**
3. **RAID 6 prodects against a single bad block failure while operating in a degraded mode.**
4. **Perfect solution for mission critical applications.**

- **Disadvantages:**

1. **More complex controller design.**
2. **Controller overhead to compute parity addresses is extremely high.**
3. **Write performance can be brought on par with RAID Level 5 by using a custom ASIC for computing Reed-Solomon parity.**

4. Requires **N+2 drives** to implement because of **dual parity scheme.**

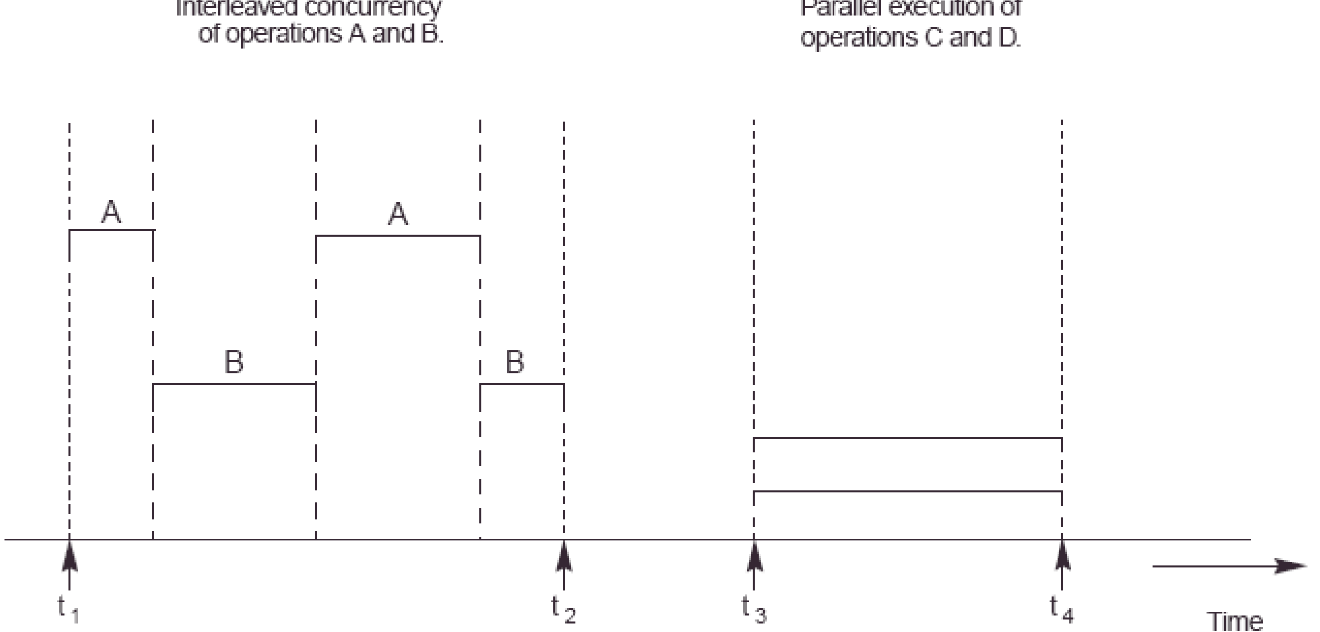
5.4 Buffering of Blocks

- **Buffering:** The process of employing **buffers** in main memory to speed up the execution.
- There are **three processors** in the architecture of computer to speed up the execution

1. Processor
2. Mathematical co processor
3. I/O Processor

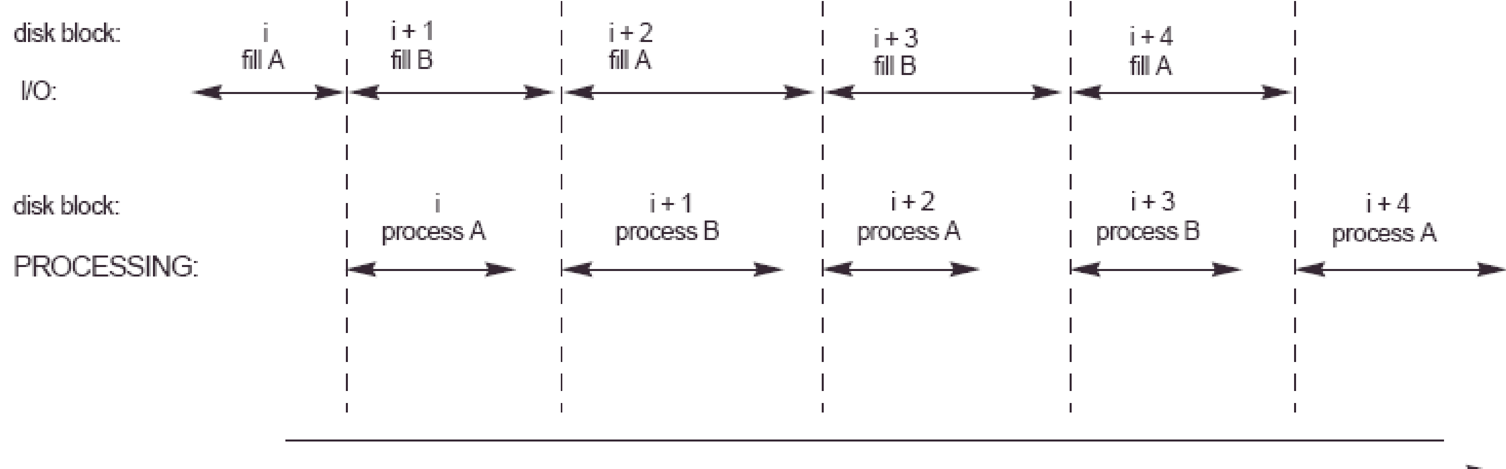
- **Parallel processing**
  - (a) The process of employing **several buffers** in main memory to speedup the execution is called **double buffering.**
  - (b) While **one buffer** is being **read** (or) **written** handled by the **I/O processor** the CPU can process data in the **other buffer**.

Figure 5.5 Interleaved concurrency versus parallel execution.



- **Double buffering: VIMP**
  (a) Data processing using the CPU and I/O PROCESSOR in multiprocessing system.
  (b) The CPU can start processing a block once its transfer to main memory is completed, at the same time the disk I/O PROCESSOR can be reading and transfer the next block into a different buffer.

Figure 5.6 Use of two buffers, A and B, for reading from disk.



5.4 Placing File Records on Disk

- **Data usually stored in the form of records.**
- **Record: Collection of related data items.**
- **Record type: Collection of attributes and their corresponding data types constitutes a record type (or) record format.**

E.g. RECORD TYPE using the C programming language.

```
struct employee {
char     name[30];
char ssn[9];
int      salary;
int      jobcode;
char     department[20];
};
```

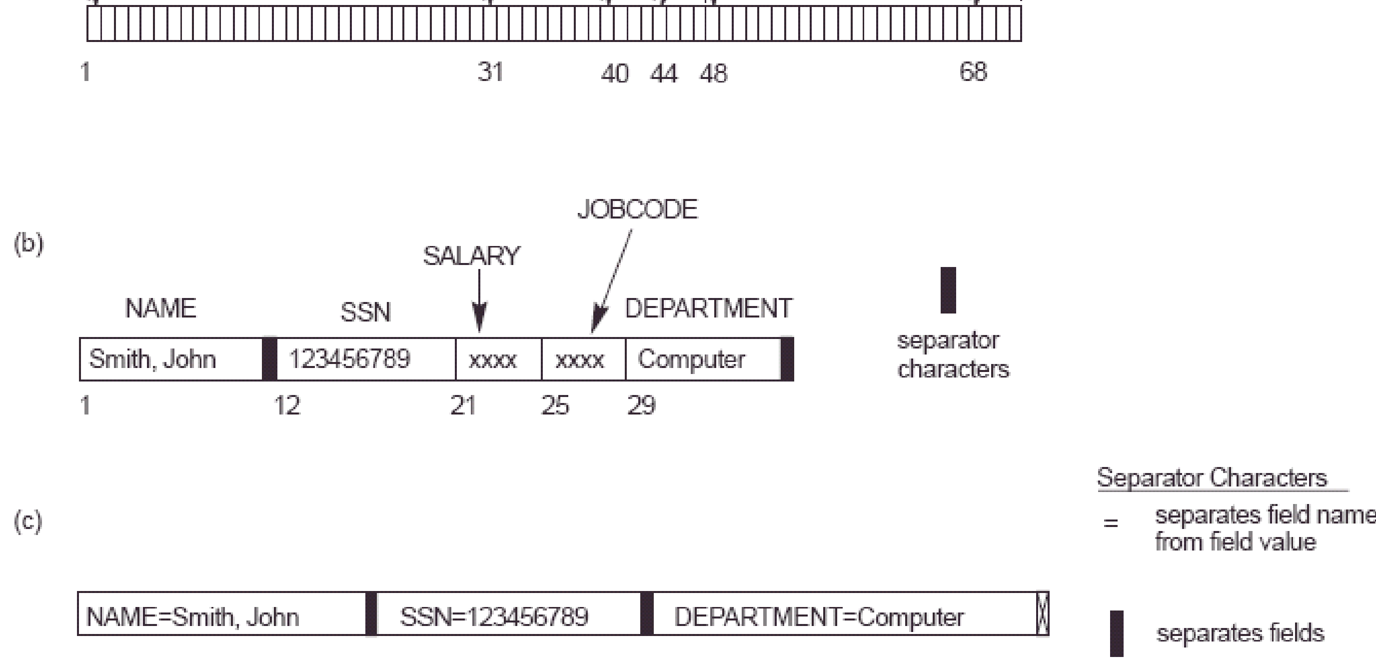
- **File: Sequence of records.**
- **Records are classified into two types**

  (a) Fixed length record: Field length is fixed.

  Note: Assign NULL values if no values exists for that field.

  (b) Variable length record: Field length may vary.

Figure 5.7 Three record storage formats. (a) A fixed-length record with six fields and size of 71 bytes. (b) A record with two variable-length fields and three fixed-length fields. (c) A variable-field record with three types of separator characters.



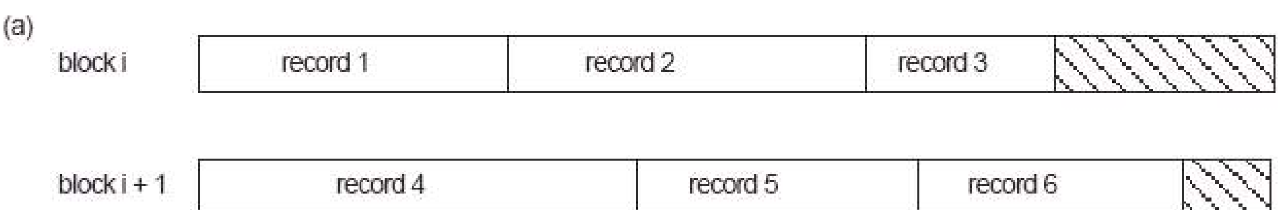
- **Block: Group of records.**

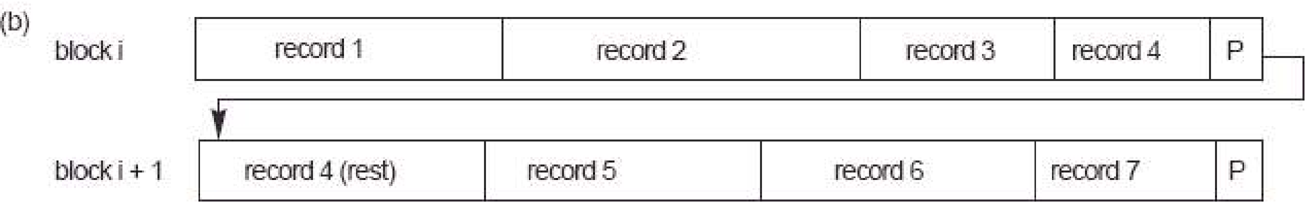
- When Block Size > Record Size, each block contain numerous records.

- Blocking factor(bfr) : Number of records per block.

- Let B is block size in bytes, R is fixed length record in bytes with condition B >= R, we can fit records $bfr = \lfloor B / R \rfloor$ per block.
- Unused space in block equal to $B - (bfr * R)$ Bytes
- Number of blocks needed to store r records $B = \lceil (r / bfr) \rceil$ Blocks

Figure 5.8 Types of record organization. (a) Unspanned. (b) Spanned.





- There are several standard techniques are available for allocating disk blocks

Contiguous allocation	Disk blocks are allocated contiguously. Advantage: Fast reading using double buffering. Disadvantage: Difficult to expand.
Linked allocation	Each file blocks contains pointer to next file block. Advantage: Easy to expand. Disadvantage: Slow to read the whole file.
Cluster (or) File segments (or) Extents	- Group of consecutive disk blocks. - Cluster are linked.

	Advantage: Fast reading .
Indexed allocation	Index block contain pointer to actual block. Advantage: Fast accessing.

- File header (or) File descriptor
- File header : Table that contains information about blocks that is needed by the system program.
- Includes information of

(a) Disk addresses of blocks.
(b) Record format descriptions of block i.e. fixed length (or) variable length (or) spanned (or) un spanned.

5.5 Operations on Files

Open	Prepares the file for reading (or) writing.
Reset	Sets file pointer to the beginning of the file.
Find (or) Locate	Searches the first record that satisfies the search condition.
Read (or) Get	Reads the content of the file.
Find Next	Searches for next record into the file that satisfies the search condition.
Delete	Deletes the current record.
Modify	Modifies field value for current record.
Close	Completes the file access.
Scan	If the file has just been opened (or) reset, scan returns the first record, otherwise it returns the next record.

Record at a time operations	
Reset	Sets the file pointer to the beginning of the file.
Insert	Insert new record in the file

Set at a time operations	
Find All	Locates all the records in the file that satisfy a search condition.

Find Ordered	Retrieves all the records in the file in some specified order.
Reorganize	Starts the reorganize process i.e. organization of data file into blocks, records.

5.6 Heap File (or) Pile File

- Heap File: File which stores the record in the order in which they are inserted.
- New records are inserted at the end of file.
- The last disk block of the file is copied into buffer, the new records are added and the block is re written back to disk.
- The address of last file block is kept in the file header.
- The deletion of record leaves unused space in the disk block. Deleting a large number of records in this way result in wasted storage space.
- Periodic reorganization is needed to reclaim the unused space of deleted records.

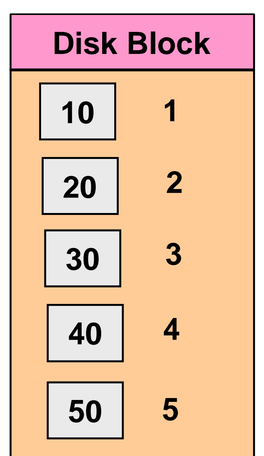
5.7 Sorted Files (or) Files of Ordered Records

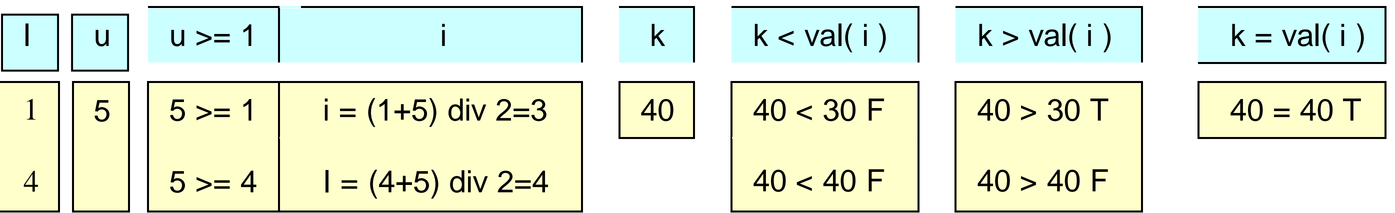
- Blocks are allocated contiguously in the file.
- Records are ordered based on key field (or) ordering field.
- Advantages:

  (a) Reading become efficient because no sorting is required.

  (b) Searching based on key field is faster.
- A Binary search for disk files can be done on the blocks rather than on the records.

```
l ← 1; u ← b; (* b is the number of file blocks *)
while (u >= l) do
begin
 i ← (l + u ) div 2;
 read block i of the file into the buffer;
 if k < (ordering key field value of the first record in the block i) then
 u ← i - 1
 else
 if k > (ordering key field value of the first record in the block i) then
 l ← i + 1
 else
 if the record with ordering key field value = k is in the buffer then goto
 found
 else goto notfound;
 end;
 goto notfound;
```





l	u	u >= 1	i	k	k < val(i)	k > val(i)	k = val(i)
1	5	5 >= 1	i = (1+5) div 2=3	40	40 < 30 F	40 > 30 T	40 = 40 T
4		5 >= 4	I = (4+5) div 2=4		40 < 40 F	40 > 40 F	

Figure 5.9 Some blocks of an ordered (sequential) file of EMPLOYEE records with NAME as the ordering key field.

	NAME	SSN	BIRTHDATE	JOB	SALARY	SEX
block 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
block 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
block 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					
block 4	Allen, Troy					
	Anders, Keith					
	⋮					
	Anderson, Rob					
block 5	Anderson, Zach					
	Angeli, Joe					
	⋮					
	Archer, Sue					
block 6	Arnold, Mack					
	Arnold, Steven					
	⋮					
	Atkins, Timothy					
⋮						
block n –1	Wong, James					
	Wood, Donald					
	⋮					
	Woods, Manny					
block n	Wright, Pam					
	Wyatt, Charles					
	⋮					
	Zimmer, Byron					

5.9 Hashing | VIMP

- **Hashing : Technique used to provide address to records stored in disk.**
- **Hash file (or) Direct file : File that implement hashing.**
- **Hash field (or) Hash key : Single field that hold hash address.**

Internal Hashing

- **Internal Hashing: The process of allocating disk addresses in a hash table is called internal hashing.**
- **Hash Table: Hash table is array of records where as array index ranges from 0 to M-1.**
- **Hash Function: Function that transforms the hash field value into integer between 0 to M-1.**

Figure 5.10 Internal hashing data structures. (a) Array of M positions for use in internal hashing. (b) Collision resolution by chaining records.

(a)

	NAME	SSN	JOB	SALARY
0				
1				
2				
3				
⋮				
M – 2				
M – 1				

(b)

- null pointer = –1.
- overflow pointer refers to position of next record in linked list.

- Two simple hash algorithms

```
(a) Applying the mod hash function to character string k
    temp ← 1;
     for i ← 1 to 20 do
     temp ← temp * code( K[i] ) mod M;
     hash_address ← temp mod M;

Note : code returns ASCII value of the key.
```

```
(b) Collision resolution by open addressing
 i ← hash_address(k); a ← i;
 if location i is occupied then
 begin
    i ← (i+1) mod m;
    while (i != a) and location i is occupied
         do i ← (i+1) mod m;
    if (i=a) then all position are full
         else new_hash_address ← i;
 end;
```

m	k	i	a	loc(i)= occupied	i!= a && loc(i) =occupied	
10	40	7	7	True		
			(7+1) mod 10 =8		8!=7 && loc(i) =occupied	True
			(8+1) mod 10 =9			
					9!=7 && loc(i) =occupied	False

Index	Value
0	
1	
2	
3	
4	10
5	20
6	
7	40
8	
9	

CONCURRENCY CONTROL

- Concurrency: A problem situation as a single data item is required by the multiple transactions simultaneously.

Topic 1: Two Phase Locking Techniques for Concurrency Control

- Lock: A lock is a variable that prescribe the state of database items.

1.1 Binary Locks

- A binary lock can have two states or values locked and unlocked (or) 1 and 0.
- ***States:***

1. Lock(x)=0 indicates that the database item is available.
2. Lock(x)=1 indicates that the database item is not available.

- ***Operations:***

1. Lock_item(x);
2. Unlock_item(x):

Note: x is a database item.

- ***Algorithm:***

```
lock_item(X):
B:  if LOCK(X) = 0                 (* item is unlocked *)
        then LOCK(X) ←1            (* lock the item *)
    else
        begin
        wait (until LOCK(X) = 0
            and the lock manager wakes up the transaction);
        go to B
        end;
unlock_item(X):
    LOCK(X) ←0;                        (* unlock the item *)
    if any transactions are waiting
        then wakeup one of the waiting transactions;
```

- ***Rules for Binary Locking:***

T_1

read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
X := X + Y;
write_item(X);
unlock(X);

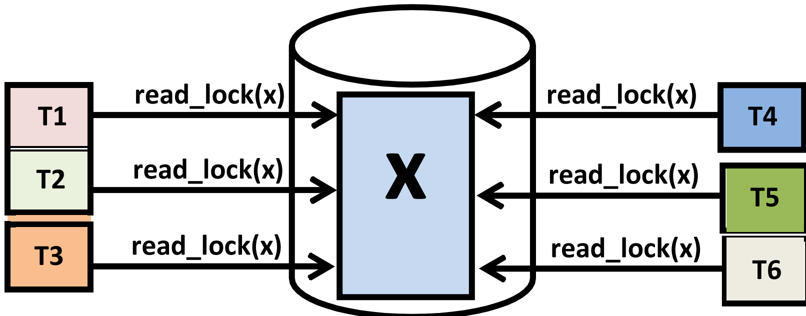
1. The read/write operation is preceded by lock operation.
2. A transaction must be unlocked if it is locked.
3. A transaction must not issue lock operation once it is locked.
4. A transaction will not issue unlock if it is not locked.

1.2 Shared/Exclusive Lock (or) Read/Write Lock

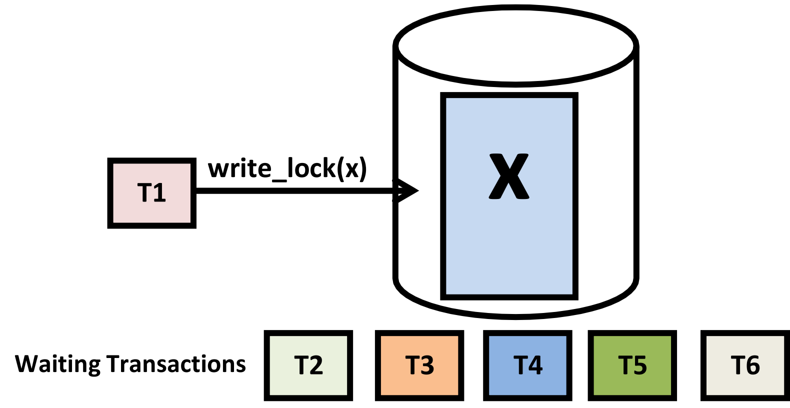
- Multiple transactions can simultaneously read the data but single transaction exclusively updates the data while remaining transactions are waiting.

- ***Operations:***

1. read_lock(x): Multiple transactions are allowed simultaneously to read the data base item.



2. write_lock(x): A single transaction exclusively holds lock on the database item.



3. unlock(x): Release of lock on data base item after completion of transaction.

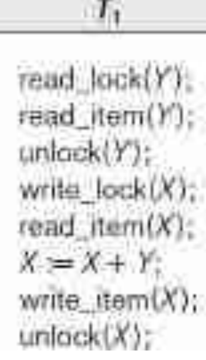
- ***Algorithm:***

```
read_lock(X):
B:  if LOCK(X) = "unlocked"
        then begin LOCK(X) ← "read-locked";
                no_of_reads(X) ← 1
                end
    else if LOCK(X) = "read-locked"
        then no_of_reads(X) ← no_of_reads(X) + 1
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
```

```
write_lock(X):
B:  if LOCK(X) = "unlocked"
        then LOCK(X) ← "write-locked"
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
```

```
unlock (X):
    if LOCK(X) = "write-locked"
        then begin LOCK(X) ← "unlocked";
                wakeup one of the waiting transactions, if any
                end
    else if LOCK(X) = "read-locked"
        then begin
                no_of_reads(X) ← no_of_reads(X) -1;
                if no_of_reads(X) = 0
                    then begin LOCK(X) = "unlocked";
                            wakeup one of the waiting transactions, if any
                            end
                end;
```

- ***Rules for Shared/Exclusive Locking***



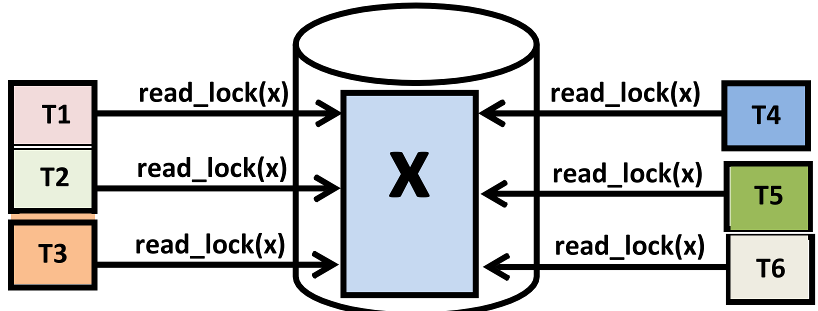
T_1
read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
X := X + Y;
write_item(X);
unlock(X);

1. The read/write operation is preceded by lock operation.
2. A transaction must issue **write lock** before **write operation**.
3. A transaction must be unlocked if it is locked.
4. A transaction must not issue lock operation once it is locked.
5. A transaction will not issue unlock if it is not locked.

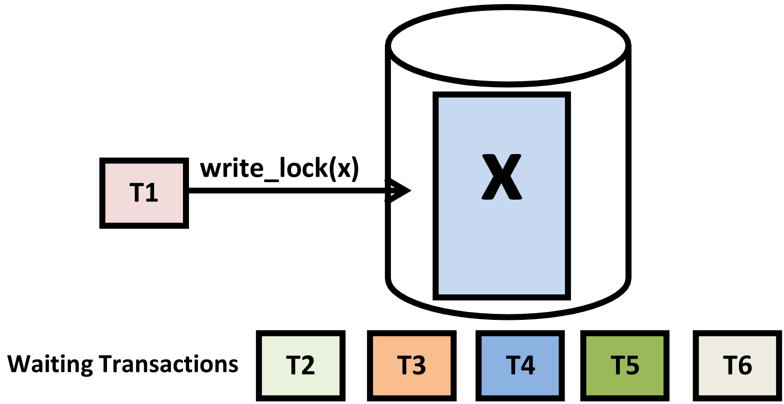
Topic 2: Guaranteeing Serializability by Two-Phase Locking

- Objective: Concurrency Control.

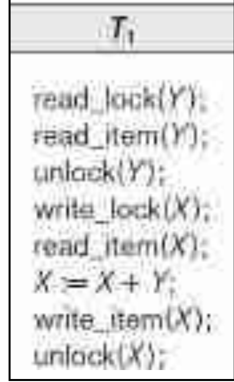
- read_lock(x): Multiple transactions are allowed simultaneously to read the data base item.



- write_lock(x): A single transaction exclusively holds lock on the database item.



- ***Rule:***



T_1

read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);

- All the locking operations are preceded by the un lock operation.

- **Growing Phase:**

1. A new lock on data base item can be acquired but none can be released.
2. Up gradation of lock is possible (Changing *read lock* to *write lock*)

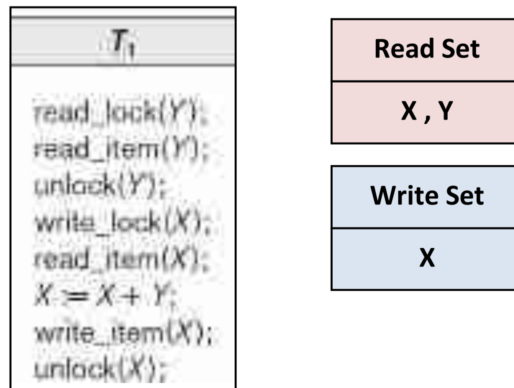
- **Shrinking Phase:**

3. Existing locks can be released but no new locks can be acquired.
4. Down gradation of lock is possible (Changing *write lock* to *read lock*)

2.1 Static 2PL (or) Conservative 2PL:

- All the locking operations are preceded by the un lock operation.
- *Read Set* & *Write Set* for the transactions are formulated.
- *Read Set*: Set of all the data items that the transaction reads.
- *Write Set*: Set of all the data items that the transaction writes.

Example: Find *Read Set* & *Write Set* for the following Transaction.

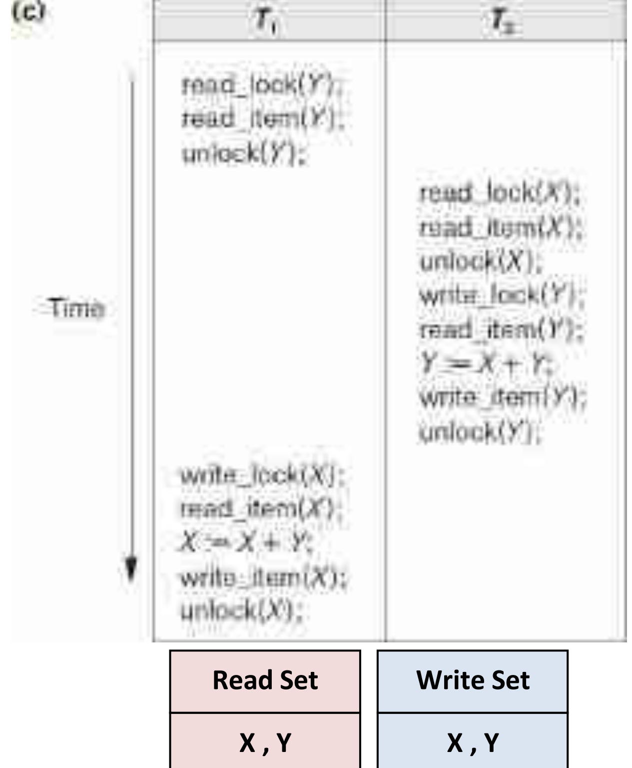


Read Set
X , Y

Write Set
X

PROBLEMS

Problem 1: Find *Read Set* & *Write Set* for the following Transactions.



Read Set	Write Set
X , Y	X , Y

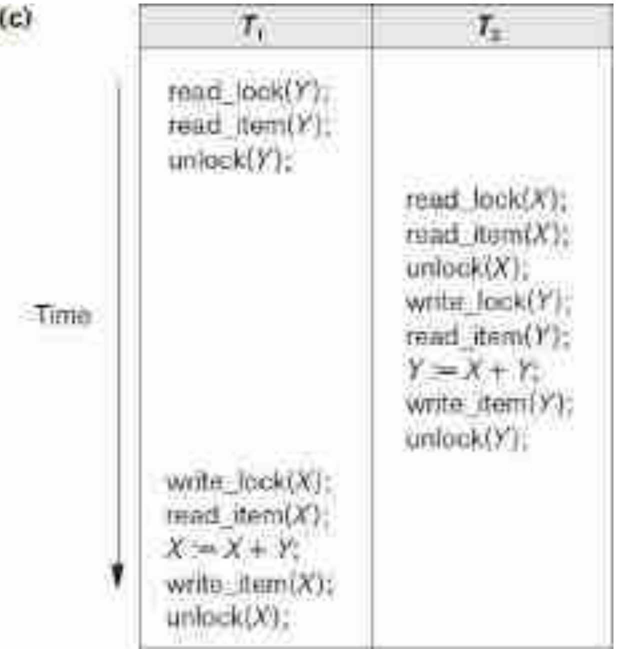
Problem 2: Verify whether the following transaction ensures 2PL.

T_2

read_lock(X);
read_item(X);
unlock(X);
write_lock(Y);
read_item(Y);
$Y := X + Y$;
write_item(Y);
unlock(Y);

Result: The Transaction T_2 ensure 2PL since all locking operations precedes un lock operations.

Problem 3: Verify whether the following transactions ensures 2PL.



(c)

T_1	T_2

read_lock(Y);
read_item(Y);
unlock(Y);	

	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);

write_lock(X);
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);	

Result: The Transaction T_2 and T_2 ensure 2PL since all locking operations precedes un lock operations.

CONCURRENCY CONTROL BASED ON TIME STAMP ORDERING

(i) Time Stamp Ordering Algorithm (TO): The Timestap ordering protocol is a protocol used to sequence the transactions based on their Timestamps.

- **Each transaction must have time stamp when it gets started.**
- **Time Stamp: The unique identification number assigned for transaction.**
- **The time stamp of T is also called as TS (T).**

Operations:

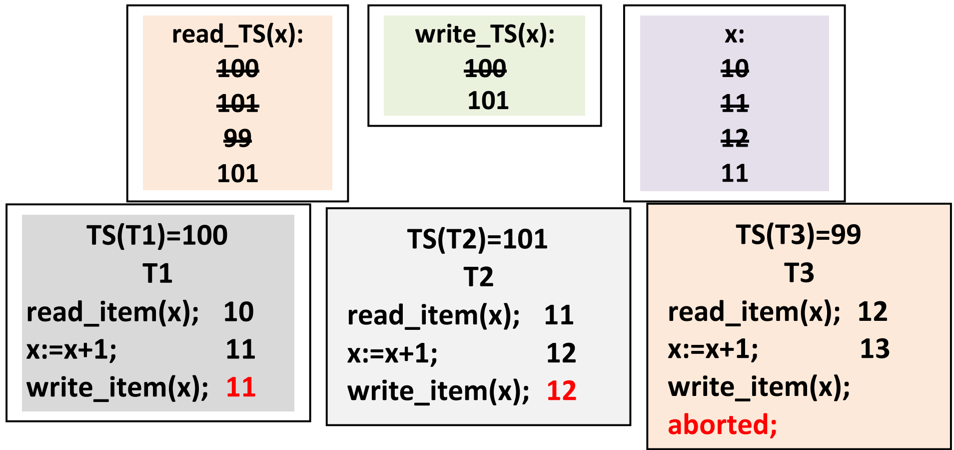
- **read_TS(X): The read timestamp of item X is the largest timestamp among all the timestamps of transactions that have successfully read item X.**
- **write_TS(X): The write timestamp of item X is the largest of all the timestamps of transactions that have successfully written item X.**

read_TS(x):	write_TS(x):	x:
~~100~~	~~100~~	~~10~~
~~101~~	~~101~~	~~11~~
102	102	~~12~~

TS(T1)=100		TS(T2)=101		TS(T3)=102	
T1		T2		T3	
read_item(x);	10	read_item(x);	11	read_item(x);	12
x:=x+1;	11	x:=x+1;	12	x:=x+1;	13
write_item(x);	11	write_item(x);	12	write_item(x);	13

(ii) Basic Time Stamp Ordering Algorithm: The order of transaction is nothing but the ascending order of the transaction creation.

- **Every transaction is issued a timestamp based on when it enters the system. Suppose, if an old transaction T_i has timestamp $TS(T_i)$, a new transaction T_j is assigned timestamp $TS(T_j)$ such that $TS(T_i) < TS(T_j)$.**



Condition 1:

- **Whenever a transaction *T* issues a write_item(X) operation, the following is checked:**

  99 > 101 101 > 99

  a. **If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then abort and roll back *T* and reject the operation.**

  b. **If the condition in part (a) does not occur, then execute the write_item(X) operation of *T* and set write_TS(X) to TS(T).**

Condition 2:

Whenever a transaction T issues a read_item(X) operation, the following is checked:

101 > 99

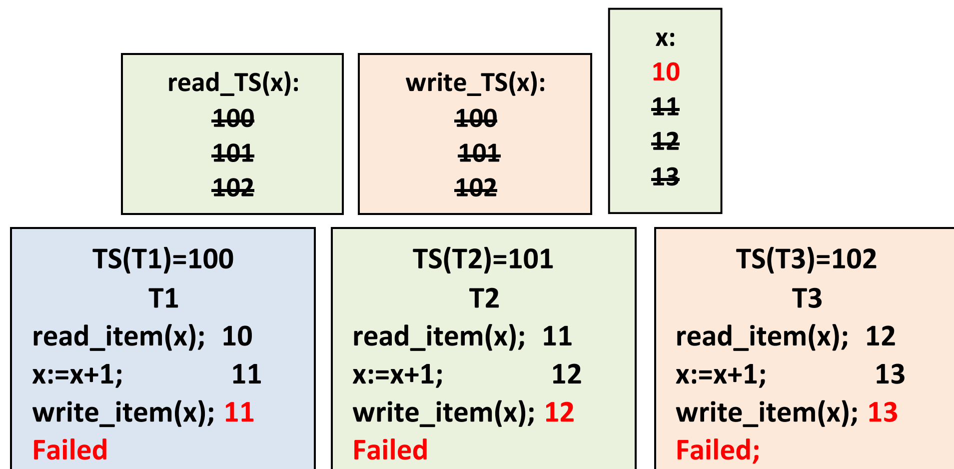
a. If write_TS(X) > TS(T), then abort and roll back T and reject the operation.

101 <= 99

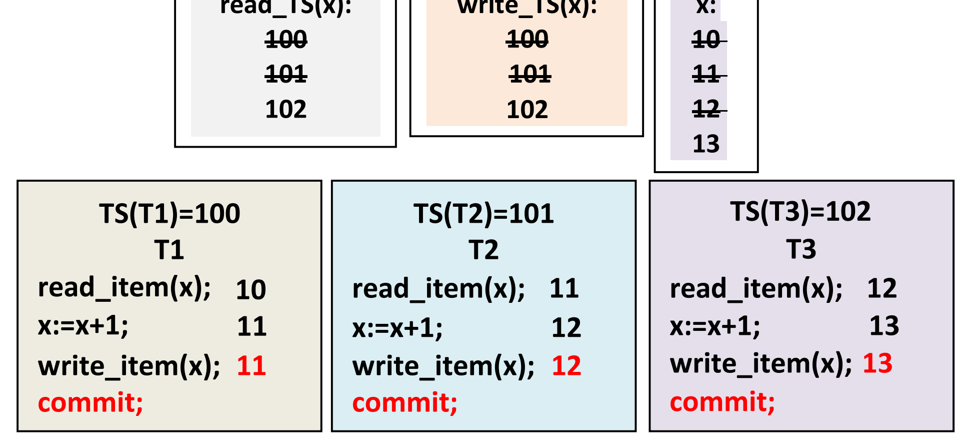
b. If write_TS(X) <= TS(T), then execute the read_item(X) operation of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

Condition 3:

Cascading Rollback: If transaction T is aborted and rolled back, any transaction T_i that may have used a value written by T must also be rolled back.



(iii) Strict Time Stamp Ordering Algorithm (TO): Every transaction must commit when it exits.



102 > 101

- **A transaction T that issues a read_item(X) or write_item(X) such that TS(T) > write_TS(X) has its read or write operation *delayed* until the transaction T that *wrote* the value of X has committed or aborted.**

- **Thomas's Write Rule: This is modification of the basic TO algorithm. When the current transaction (TS(T)) fails then continue the read operation and abort write operation.**

  102 > 102

  1. **If read_TS(*X*) > TS(T), then abort and roll back T and reject the operation.**

  102 > 102

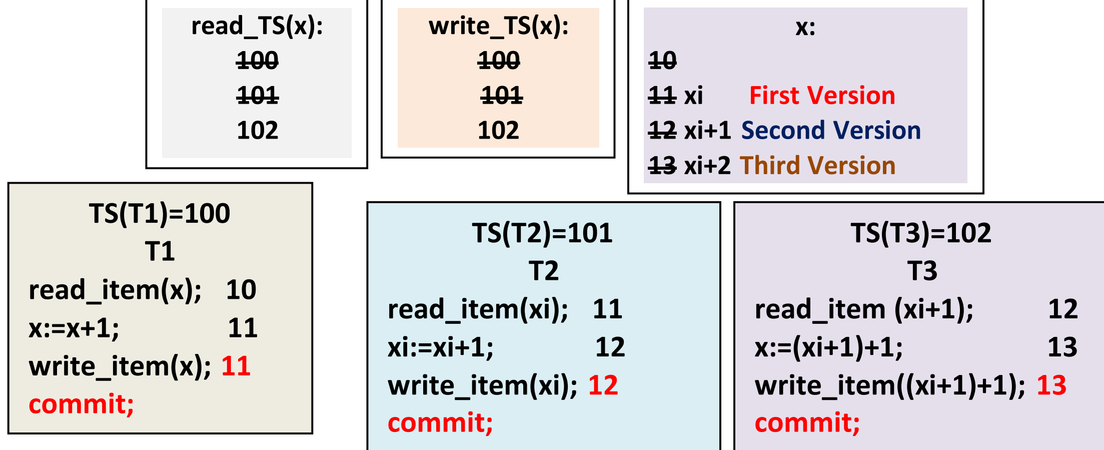
  2. **If write_TS(*X*) > TS(T), then do not execute the write operation but continue processing.**

read_TS(x):	write_TS(x):	x:
~~100~~	~~100~~	~~10~~
~~101~~	~~101~~	~~11~~
102	~~102~~	12
	101	

TS(T1)=100	TS(T2)=101	TS(T3)=102
T1	T2	T3
read_item(x); 10	read_item(x); 11	read_item(x); 12
x:=x+1; 11	x:=x+1; 12	x:=x+1; 13
write_item(x); 11	write_item(x); 12	write_item(x); 13
commit;	commit;	Failed;

1.4 Multiversion Concurrency Control Techniques

- **Version: State of database item after successful updation.**



- **Temporal Database: Database keeps track of multiple versions of database items.**
  E.g. xi, xi+1, xi+2, xi+3 etc.

- **Advantage: Multiple versions of database items are maintained so that concurrency is avoided to certain extent.**

- **Operations:**

1. read_TS(Xi): The read timestamp of Xi is the largest of all the timestamps of transactions that have successfully read version Xi.

2. write_TS(Xi): The write timestamp of Xi is the timestamp of the transaction that wrote the value of version Xi.

- **Serializability in Multiversion Concurrency Control:**

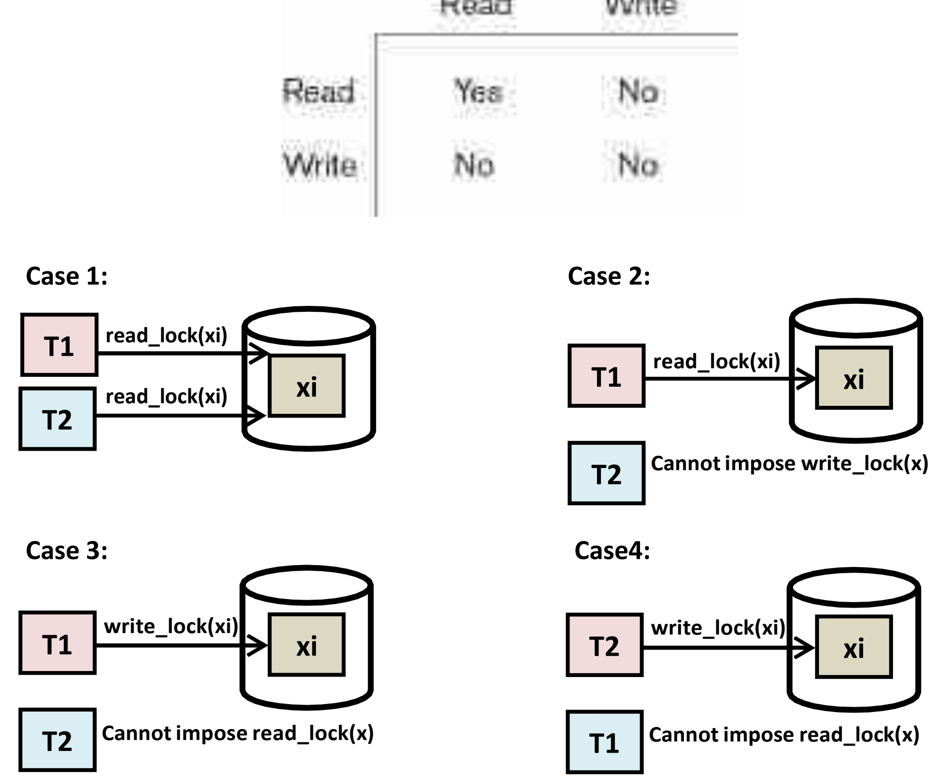
To ensure the serializability the follow rules are used

1. **If transaction T issues a write_item(X) operation, and version i of X has the highest write_TS(Xi) of all versions of X that is also less than or equal to TS(T), and read_TS(Xi) > TS(T), then abort and roll back transaction T; otherwise, create a new version Xj of X with read_TS(Xj) = write_TS(Xj) = TS(T).**

2. **If transaction T issues a read_item(X) operation, find the version i of X that has the highest write_TS(Xi) of all versions of X that is also less than or equal to TS(T); then return the value of Xi to transaction T, and set the value of read_TS(Xi) to the larger of TS(T) and the current read_TS(Xi).**

- **Multiversion Two-Phase Locking Using Certify Locks:**

1. Compatability table for *read/write* locks:



	Read	Write
Read	Yes	No
Write	No	No

2. Compatability table for read/write/certify locks:

Certify Lock (or) Notify Lock: Certify lock has highest priority. It is the monarch lock where other locks can not be done.

	Read	Write	Certify
Read	Yes	Yes	No
Write	Yes	No	No
Certify	No	No	No

DISTRIBUTED DATABASES

TOPIC 1: DISTRIBUTED DATABASE CONCEPTS VVIMP

- **Centralized database: Centralized database is a software system that creates & access data base at central site**

- **Disadvantage of central site: If central site fails, database is not available for other sites in the network**

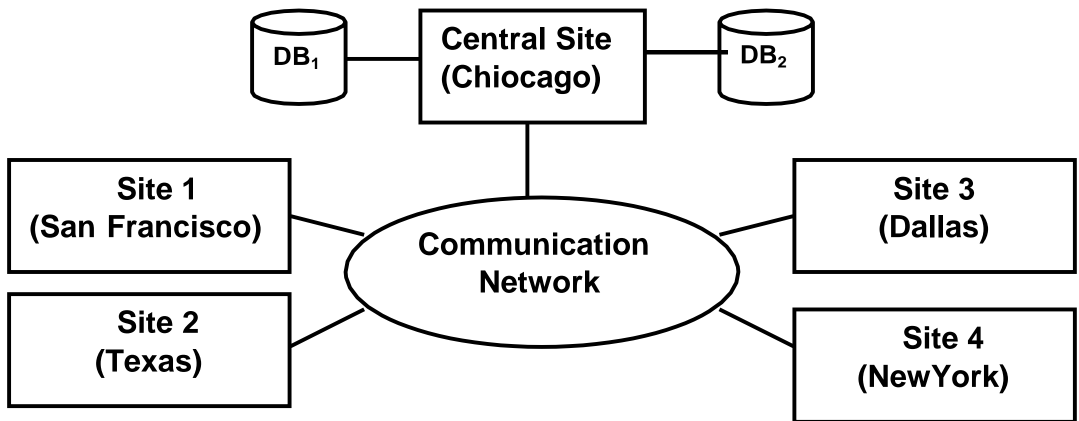


Figure 1. Centralized Database Architecture.

- **Distributed Database Management System (DDBMS): Distributed database management system is a software system that *creates & access* the distributed database over a computer network.**

1.1 DIFFERENCES BETWEEN DDB AND MULTIPROCESSOR SYSTEMS

- **Connection of database nodes over a computer network: There are multiple computers, called sites or nodes. These sites must be connected by an underlying communication o transmit data and commands among sites.**
- **Logical interrelation of the connected databases: It is essential that the information in the databases be logically related.**
- **Absence of homogeneity constraint among connected nodes: It is not necessary that all nodes be identical in terms of data, hardware, and software.**

1.2 TRANSPARENCY

(a) ***Data Organization Transparency*** **(or)** ***Distribution or Network Transparency***	**This refers the user can access any site across the network.** ▪ ***Location Transparency*****: The commands used to perform a task is independent of the location of data and location of system across the network.** ▪ ***Naming Transparency*****: The name of the objects can be accessed unambiguously without additional specifications.**
(b) Replication Transparency	**Copies of data may be stored at multiple sites for better** ***availability*****,** ***performance*****, &** ***reliability*****.**
(c) Fragmentation Transparency	**Two types of fragmentation are possible** **1.** ***Horizontal Fragmentation*****: Distributes the relation into set of tuples (rows).** **2.** ***Vertical Fragmentation*****: Distributes the relation into sub relations where each sub relation is defined by a subset of the columns of the original relation.**

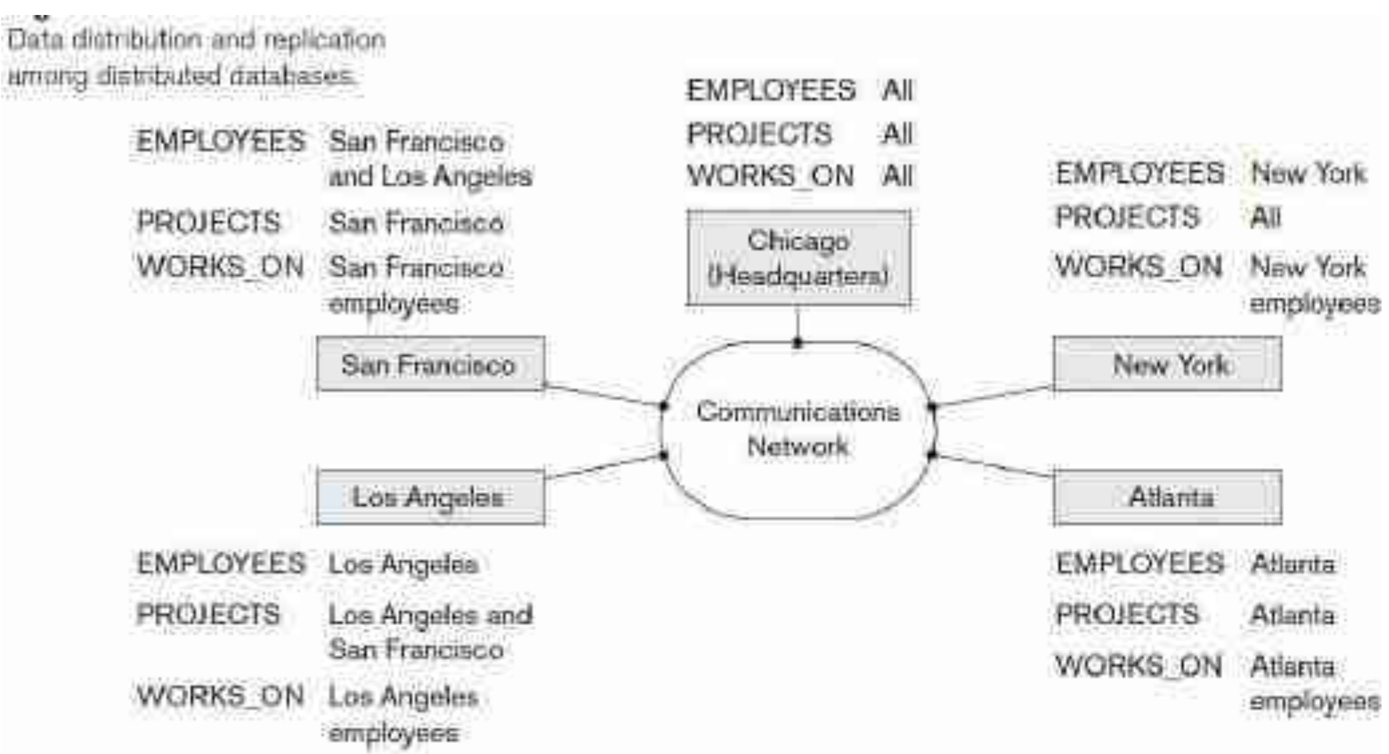


Figure 2. A Truly Distributed Database Architecture

1.3 AUTONOMY

- **Autonomy determines the extent to which individual nodes or databases in a connected Distribute Data Base can operate independently.**
- **A high degree of autonomy is desirable for increased flexibility and customized maintenance of an individual node.**
- **Autonomy can be applied to design, communication, and execution.**

1. ***Design Autonomy*: Design autonomy refers to independence of data model usage and transaction management techniques among nodes.**
2. ***Communication Autonomy*: Communication autonomy determines the extent to which each node can decide on sharing of information with other nodes.**
3. ***Execution Autonomy*: Execution autonomy refers to independence of users to act as they please.**

1.4 RELIABILITY & AVALIABILITY

- **Reliability: Reliability is broadly defined as the probability that a system is running (not down) at a certain time point.**

- **Availability: The probability that the system is continuously available during a time interval.**

1.5 ADVANTAGES OF DISTRIBUTED DATABASE

1. **Improved ease and flexibility of application development.**

2. **Increased reliability and availability**
3. **Improved performance**

- **Data localization reduces the contention for CPU and I/O services and reduce access delay.**

- **Local queries & transactions accessing data at a single site have better performance because of the smaller local databases.**

- ***Inter query* & *Intra query* parallelism can be achieved by executing multiple queries at different queries at different sites or by breaking up query into a number of sub queries that execute in parallel.**

4. **Easier expansion.**

1.6 FUNCTIONS OF DDBMS

1. ***Keeping Track of Data Distribution:*** **The ability to keep track of the data distribution, fragmentation, and replication by expanding the DDBMS catalog.**

2. ***Distributed Query Processing:*** **The ability to access remote sites and transmit queries and data among the various sites via a communication network.**

3. ***Distributed Transaction Management:*** **The ability to access remote sites and transmit queries and data among the various sites via a communication network.**

4. ***Replicated Data Management:*** **The ability to decide which copy of a replicated data item to access.**

5. ***Distributed Database Recovery:*** **The ability to recover from individual site crashes and from new types of failures, such as the failure of communication links.**

6. ***Security:*** **Distributed transactions must be executed with the proper management of the security of the data and the authorization/access privileges of users.**

7. ***Distributed Directory (Catalog) Management:*** **A directory contains information (metadata) about data in the database. The directory may be global for the entire DDB, or local for each site.**

TOPIC 2: TYPES OF DISTRIBUTED DATABASE SYSTEMS

2.1 FEDERATED DATABASE MANAGEMENT SYSTEMS

- **A Federated Database System (FDBS) is a collection of cooperating database systems that are autonomous and possibly heterogeneous.**

- **The type of heterogeneity present in FDBSs may arise from several sources.**

TYPES OF HETEROGENEITY:

1. ***Differences in Data Models:***

- **Data is represented using different data models. (Hierarchical, Relational, Network & Object Oriented data models).**

2. ***Differences in Constraints:***

- **The global schema must also deal with different constraints. (Not null, Unique, Primary Key, Foreign Key, Check)**
- **ER models are represented as referential integrity constraints in the relational model.**
- **Triggers may have to be used to implement certain constraints in the relational model.**

3. ***Differences in Query Languages:***

- **Even with the same data model, the languages and their versions vary. For example, SQL has multiple versions like SQL-89, SQL-92, SQL-99, and SQL: 2008, and each system has its own set of data types, comparison operators, string manipulation features, and so on.**

2.1.2 SEMANTIC HETEROGENEITY.

- **Semantic heterogeneity occurs when there are differences in the meaning, interpretation, and intended use of the same or related data.**

1. ***The universe of discourse from which the data is drawn.***

- **For example, for two customer accounts, databases in the federation may be from the United States and Japan and have entirely different sets of attributes about customer accounts required by the accounting practices. Currency rate fluctuations would also present a problem. Hence, relations in these two**

databases that have identical names-CUSTOMER or ACCOUNT-may have some common and some entirely distinct information.

2. Representation and Naming.

- **The representation and naming of data elements and the structure of the data model may be pre specified for each local database.**

3. **The understanding, meaning, and subjective interpretation of data.**

- **This is a chief contributor to semantic heterogeneity.**

4. Transaction and Policy Constraints.

- **These deal with *serializability criteria*, *compensating transactions*, and *other transaction policies*.**

5. Derivation of Summaries.

- **Aggregation, summarization, and other data processing features and operations supported by the system.**

TOPIC 3: DISTRIBUTED DATABASE ARCHITECTURES

3.1 PARALLEL VERSUS DISTRIBUTED ARCHITECTURES

There are two main types of multiprocessor system architectures that are commonplace:

- **Shared Memory (tightly coupled) Architecture: Multiple processors share secondary (disk) storage and also share primary memory.**
- **Shared Disk (loosely coupled) Architecture: Multiple processors share secondary (disk) storage but each has their own primary memory.**

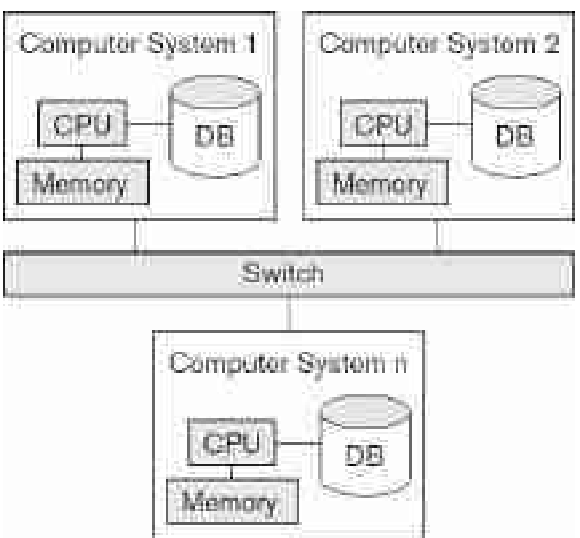


Fig 1. Shared nothing architecture.

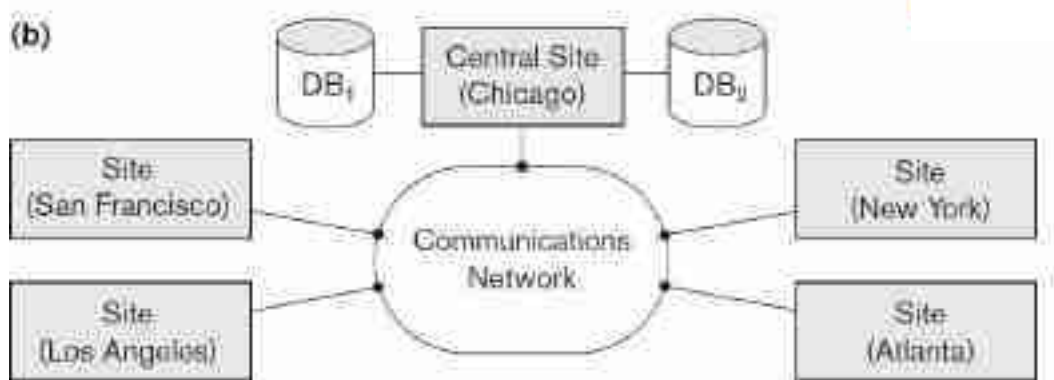


Fig 2. A networked architecture with a centralized database at one of the sites.

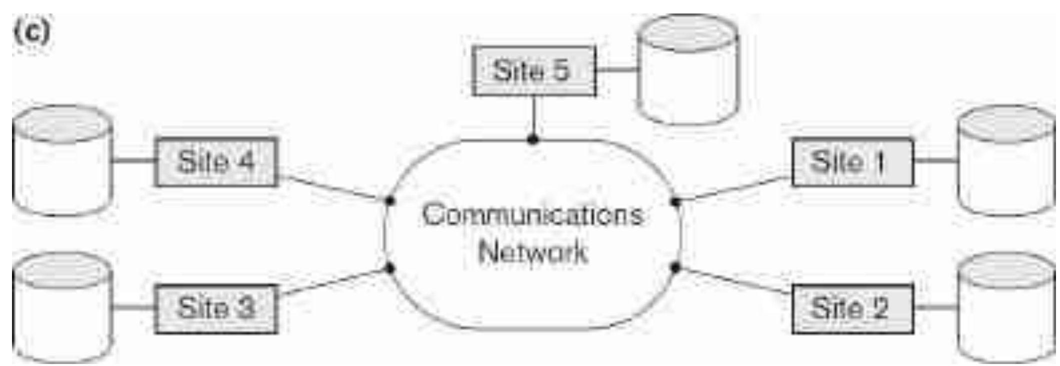


Fig 3. A truly distributed database architecture

3.2 GENERAL ARCHITECTURE OF PURE DISTRIBUTED DATABASES

- **In this section we discuss both the logical and component architectural models of a DDB.**

3.2.1 SCHEMA ARCHITECTURE OF DISTRIBUTED DATABASES.

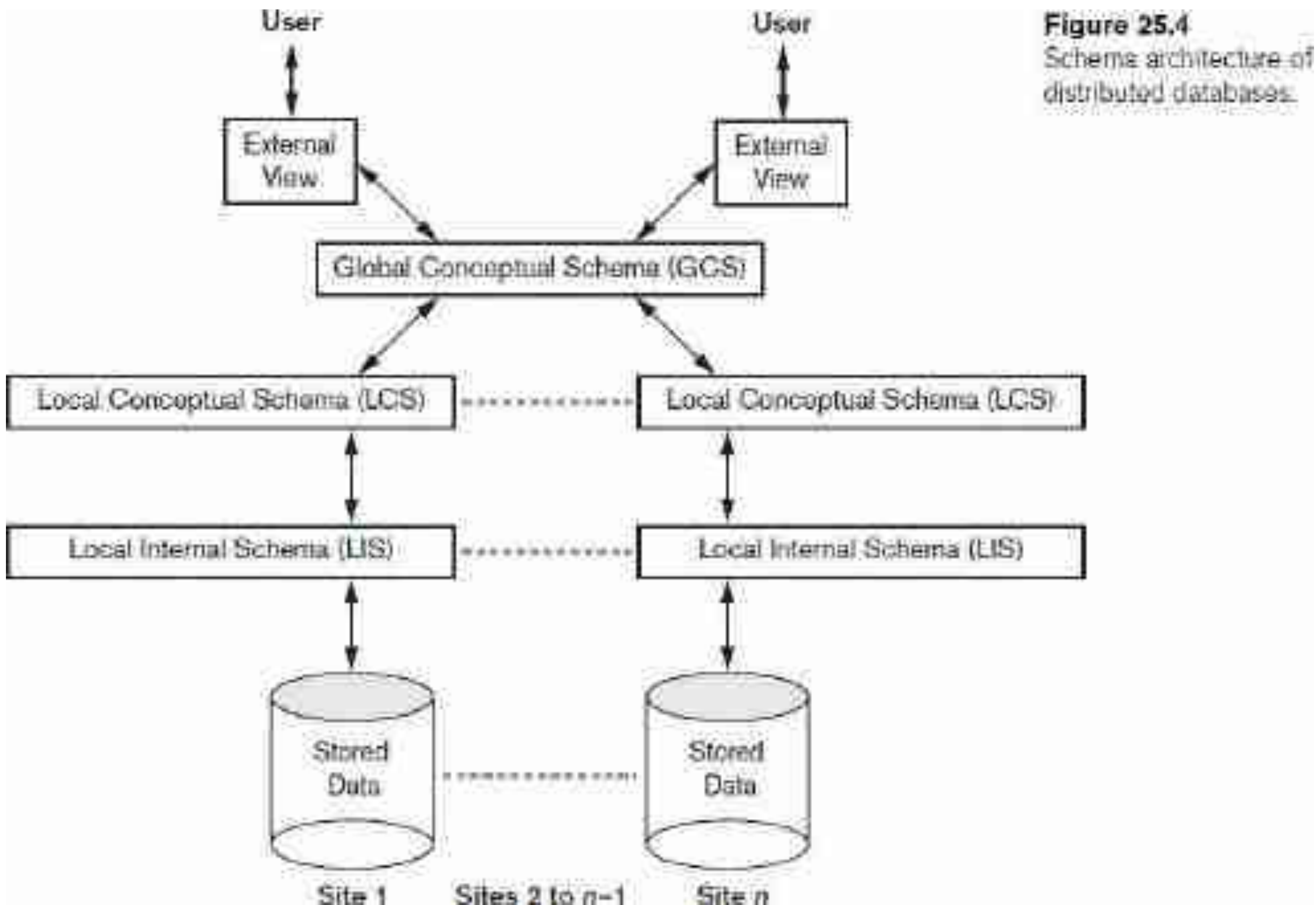


Fig 4. Schema Architecture of Distributed Database.

- ***Logical Internal Schema (LIS)*** **specifies** ***physical organization*** **details at that particular site.**
- ***Logical Conceptual Schema (LCS)*** **specifies the** ***logical organization*** **of data at each site is specified by the local conceptual schema**
- ***The Global Conceptual Scheme*** **(GCS) specifies provides** ***network transparency.***

3.2.2 COMPONENT ARCHITECTURE OF DISTRIBUTED DATABASES.

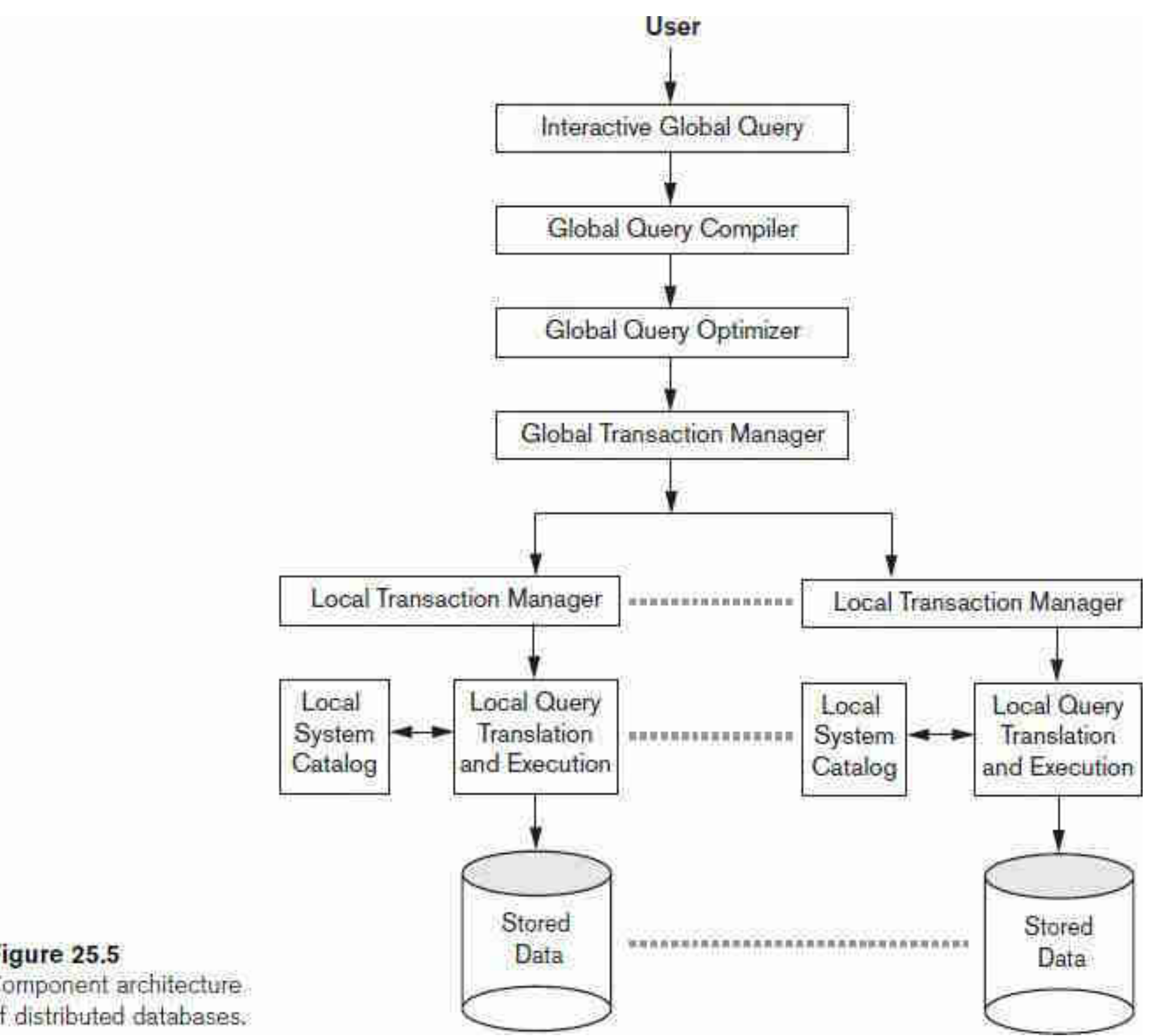


Fig 5. Component architecture of distributed database.

- **Query Compiler: Inspects the process of query compilation.**
- **Query Optimizer: The optimizer selects the minimum cost for execution.**
- **Transaction Manager: Coordinates and executes the transactions.**

- **System Catalog: The system catalogue stores meta-data including the schemas of the databases.**

3.2.3 FEDERATED DATABASE SCHEMA ARCHITECTURE

- **Federated Schema is an integration of multiple export schemas.**

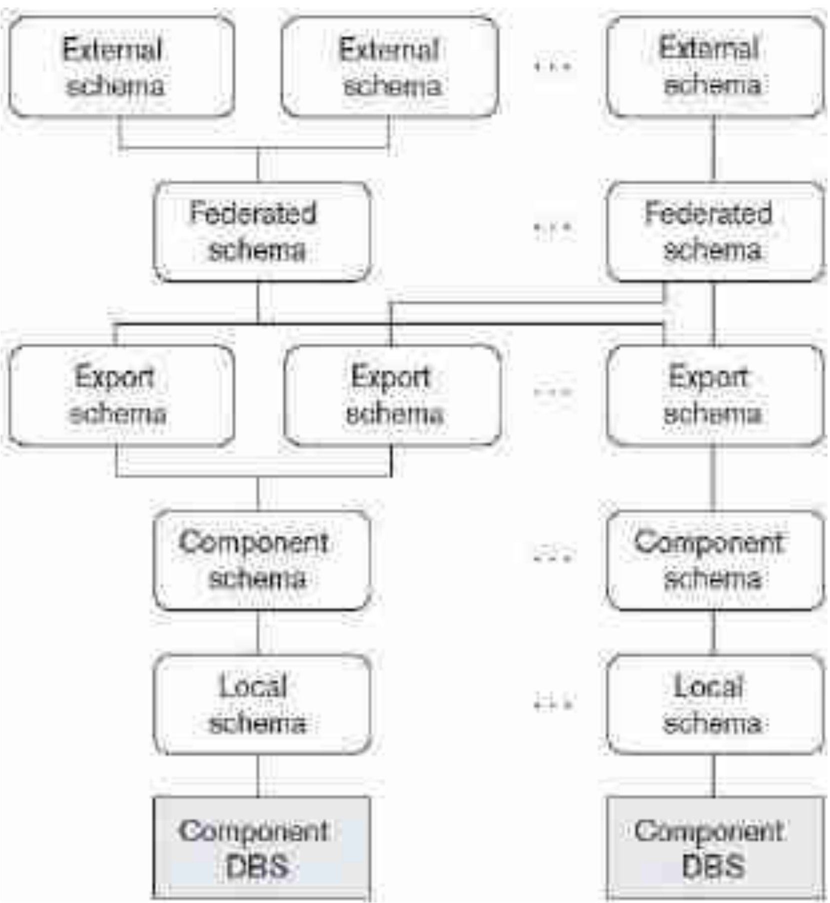


Fig 6. The five-level schema architecture in a federated database system (FDBS).

- **Component DBS: Database control system.**
- **Local schema: Conceptual schema.**
- **Component schema: Translates the local schema into common data model.**
- **Export schema: Subset of component schema.**
- **Federated schema: The federated schema is the global schema or view, which is the result of integrating all the shareable export schemas.**

- **External schema: External schemas define the schema for a user group or an application.**

3.2.4 THREE-TIER CLIENT-SERVER ARCHITECTURE

In the three-tier client-server architecture, the following three layers exist:

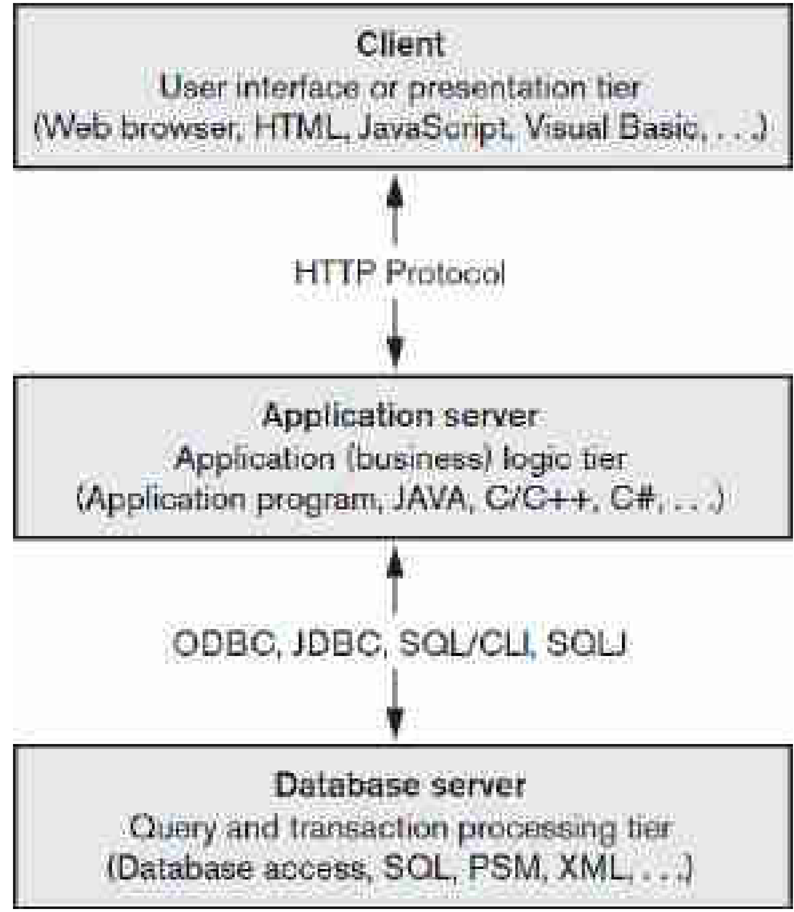


Fig 7. The three tire client server architecture.

1. Presentation Layer (Client):

- **This provides the user interface and interacts with the user.**
- **The programs at this layer present *web interfaces* or *forms* to the client in order to interact with the application.**
- **Web browsers are often utilize the languages like *HTML*, *XHTML*, *CSS*, *Java*, *JavaScript* and others to perform *client side validations*.**

2. Application Layer (Business Logic):

- **Application logic is written in this phase.**
- **Queries can be formulated based on user input from the client.**
- **Query results can be formatted and sent to the client for presentation.**
- **Additional application functionality can be handled such as security checks, identity verification, and other functions.**
- **The application layer can interact with one or more databases or data sources as needed by connecting to the database using ODBC, JDBC, SQL/CLI, or other database access techniques.**

3. Database server.

- **This layer handles query and update requests from the application layer, processes the requests, and sends the results.**
- **Usually SQL is used to access the database if it is relational or object-relational and stored database procedures may also be invoked.**
- **Query results (and queries) may be formatted into XML when transmitted between the application server and the database server.**

TOPIC 4: DATA FRAGMENTATION, REPLICATION, AND ALLOCATION TECHNIQUES FOR DISTRIBUTED DATABASE DESIGN

- **Fragment: Subset of table.**

3.1 DATA FRAGMENTATION

- **Fragmentation is the task of dividing a table into a set of smaller tables.**
- **The subsets of the table are called fragments.**
- **Fragmentation can be of three types: horizontal, vertical, and hybrid.**

EMPLOYEE		
EMPNO	ENAME	SALARY
1	JOHN	3400
2	MARTIN	4300
3	ELICA	4500
4	MILLER	3400

3.1.1 Horizontal Fragmentation: A horizontal fragment of a relation is a subset of the tuples in that relation. The tuples that belong to the horizontal fragment are specified by a condition on one or more attributes of the relation.

EMPLOYEE		
EMPNO	ENAME	SALARY
1	JOHN	3400
2	MARTIN	4300

3.1.2 Vertical Fragmentation: Vertical fragmentation divides a relation "vertically" by columns. A vertical fragment of a relation keeps only certain attributes of the relation.

EMPLOYEE	
EMPNO	ENAME
1	JOHN
2	MARTIN
3	ELICA
4	MILLER

3.1.3 Mixed Fragmentation: We can intermix the two types of fragmentation, yielding a mixed fragmentation.

EMPLOYEE	
EMPNO	ENAME
1	JOHN
2	MARTIN

3.2 DATA REPLICATION AND ALLOCATION

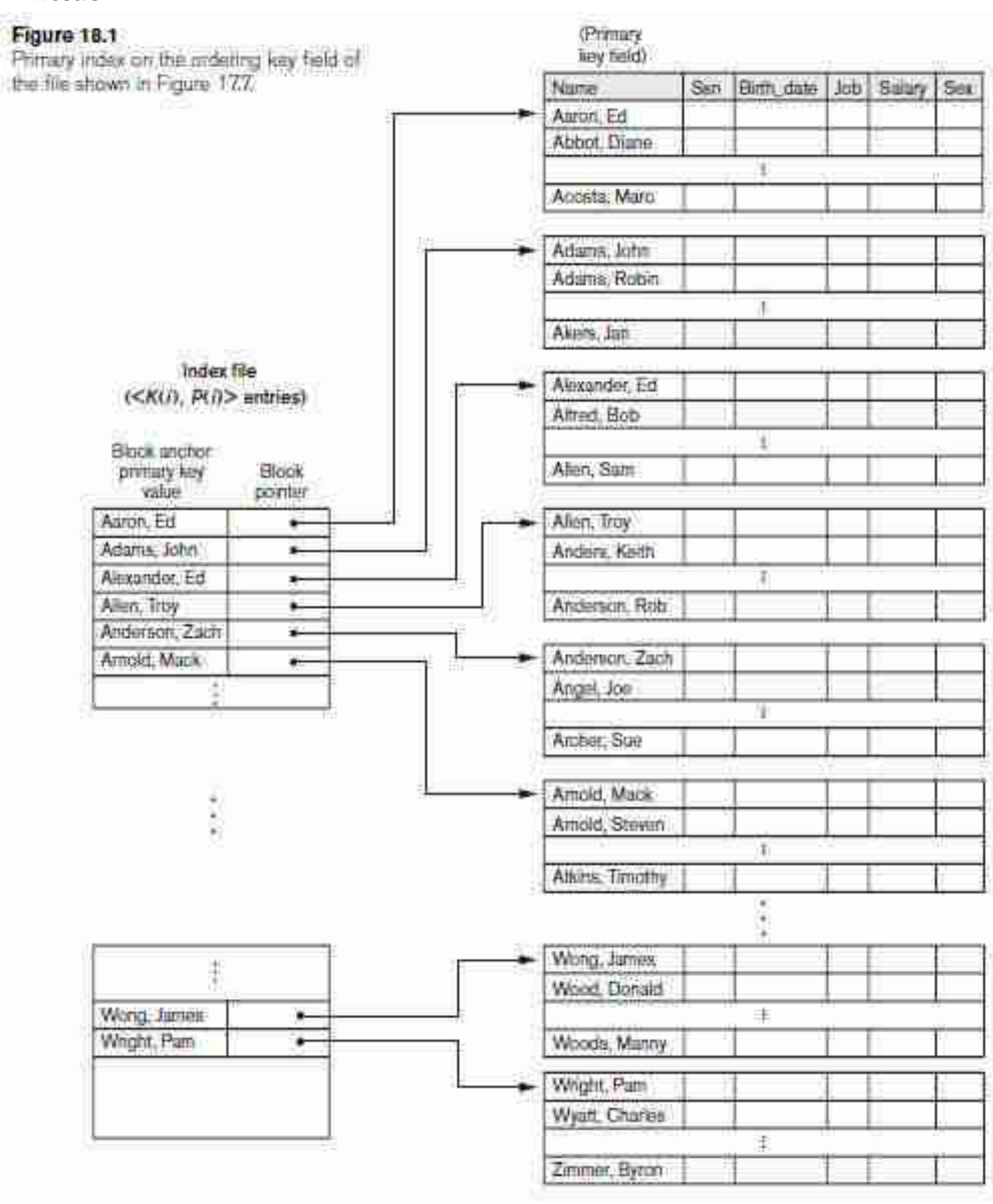
- **Data Replication is the process of storing data in more than one site or node.**
- **It is useful in improving the availability of data.**
- **It is simply copying data from a database from one server to another server so that all the users can share the same data without any inconsistency.**
- **The most extreme case is replication of the whole database at every site in the distributed system, thus creating a fully replicated distributed database.**
- **The other extreme from full replication involves having no replication that is, each fragment is stored at exactly one site. This is also called non redundant allocation.**

CHAPTER 6
INDEXING STRUCTURES FOR FILES

6.1 SINGLE LEVEL ORDERD INDEXES

6.1.1 PRIMARY INDEXES

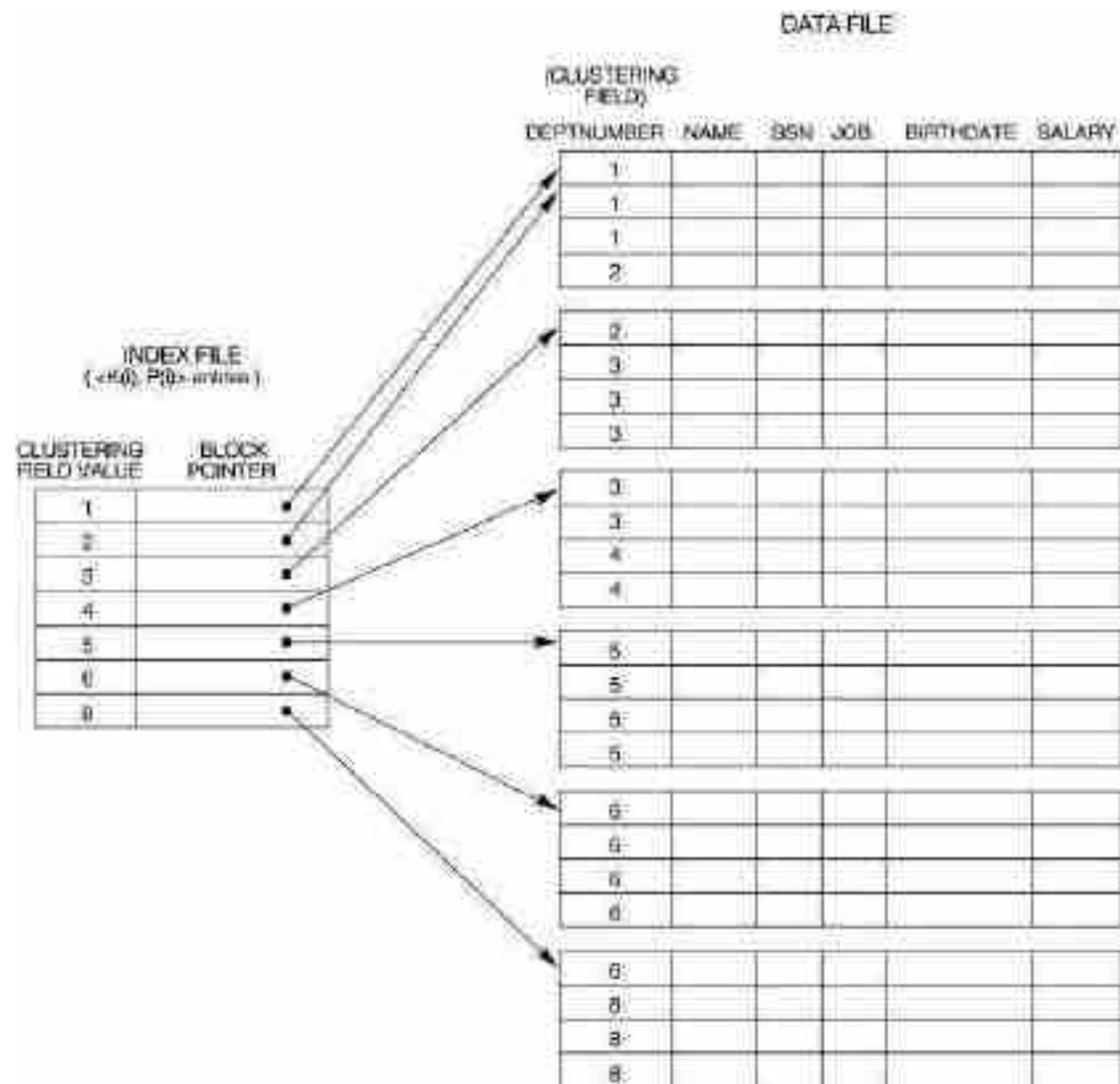
- **A Primary index is an ordered file whose records are of fixed length with two fields**



- **Anchor record: The first record in each block**
- **Dense index: Index entry for every search key value in the data file.**
- **Sparce index: Index entry for some search key value in the data file.**

6.1.2 CLUSTERING INDEXES

- **A clustered index defines the order in which data is physically stored in a table.**
- **Clustering field: Field that orders the records physically.**



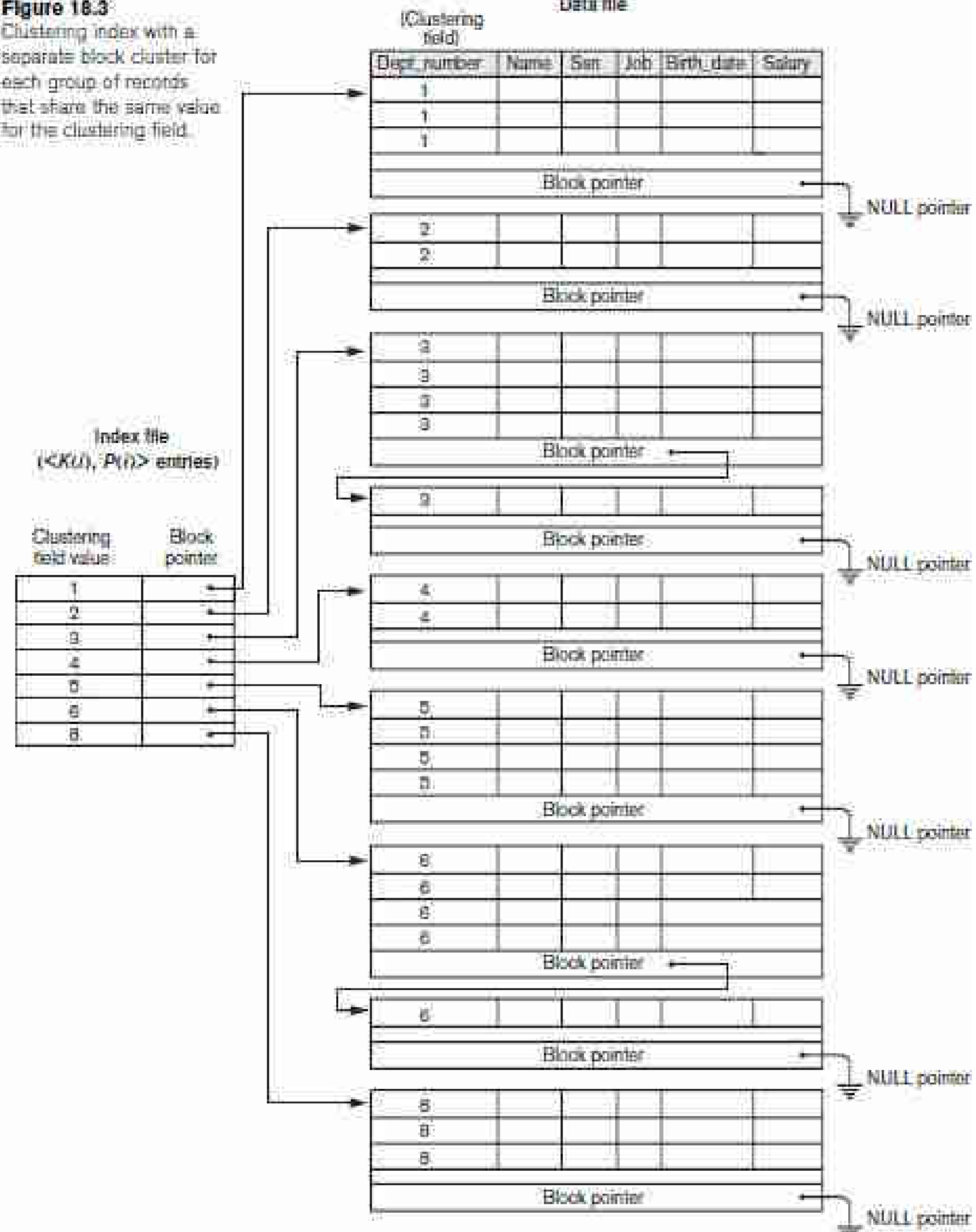


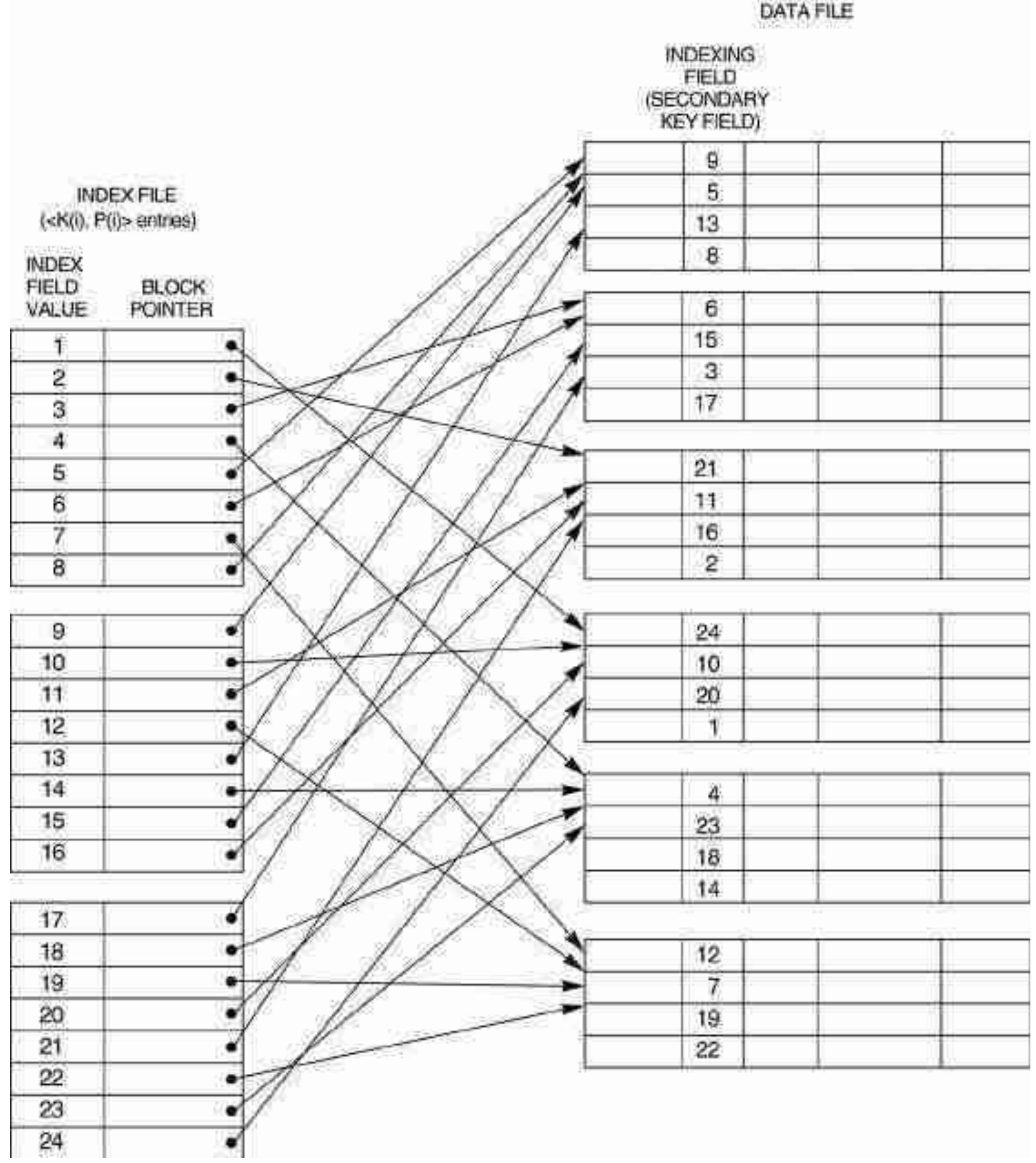
Figure 18.3
Clustering index with a separate block cluster for each group of records that share the same value for the clustering field.

6.1.3 SECONDARY INDEXES WITH BLOCK POINTERS (A dense secondary index)

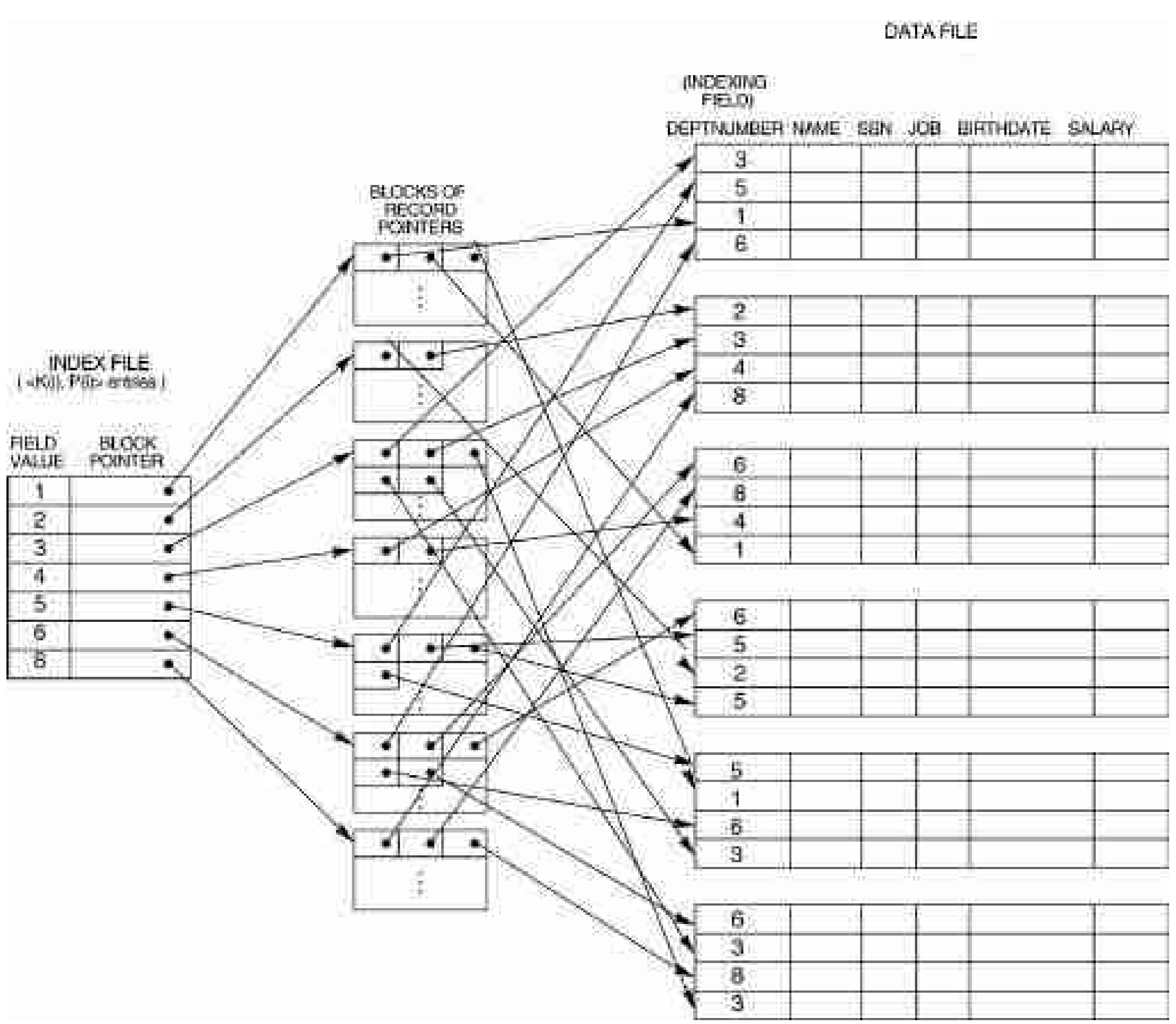
- **A secondary index provides a secondary means of accessing a file for which some primary access already exists.**

- **The index file has two fields**

(a) The first field is Index field value
(b) The second field is Block pointer

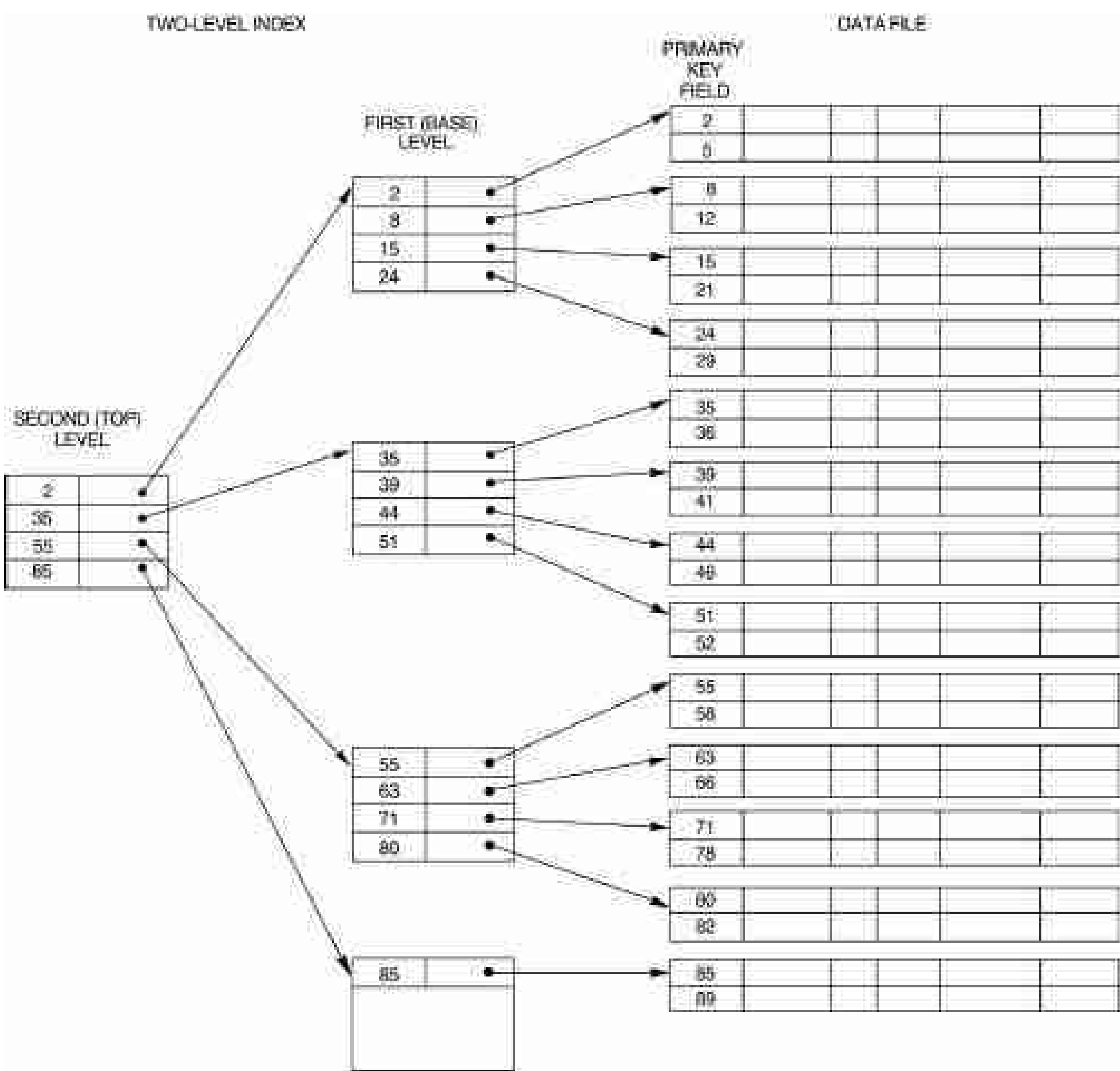


6.1.4 SECONDARY INDEXES WITH RECORD POINTER



6.2 MULTI LEVEL INDEXES

- **Multilevel indexing is a two-level primary index resembling ISAM (Indexed Sequential Access Method) organization.**



6.3 DYNAMIC MULTI LEVEL INDEXES

- **B Trees**
- **B+ Trees**

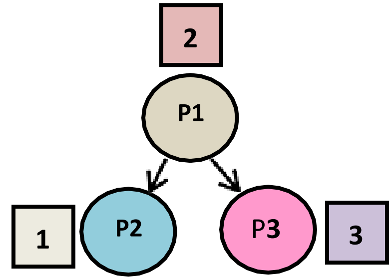
B-Trees

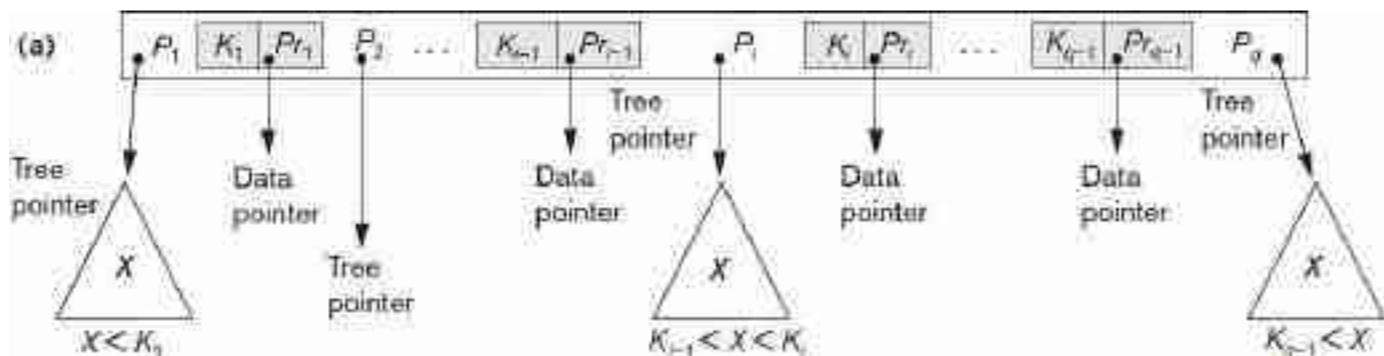
- **B-Tree: B-Tree is a specialized multi way tree designated especially for the use on disk.**

Properties:

1. **The root node has at least two children.**
2. **All the nodes other than root node not have at least $\lceil m/2 \rceil$ child.**
3. **All failure nodes at the same level.**
4. **When a key value is to be inserted in a node which already has the maximum number of key values, the following steps are to be adopted**
   a) **Insert the value, say 'x' into the list of values in the node in ascending order.**
   b) **Split the list of values into three parts P1,P2 and P3**
      - **P1 contains first $\lceil m/2 \rceil - 1$ key values.** **E.g. $\lceil 3/2 \rceil - 1 = 2 - 1 = 1$**
      - **P3 contains $\lceil m/2 \rceil + 1, \dots, m^{th}$ values** **E.g. $\lceil 3/2 \rceil + 1 = 2 + 1 = 3$**
      - **P2 contains $\lceil m/2 \rceil$ value** **E.g. $\lceil 3/2 \rceil = 2$**

Note: P1,P2 & P3 must be in ascending order.





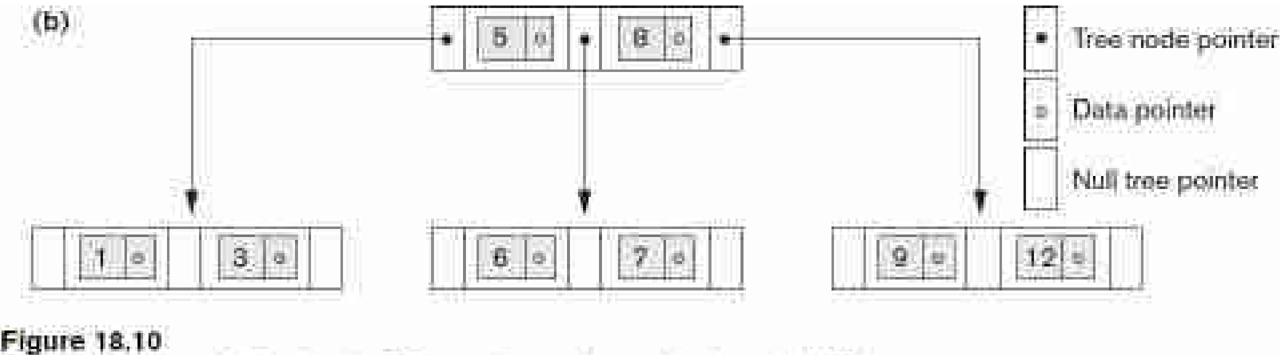
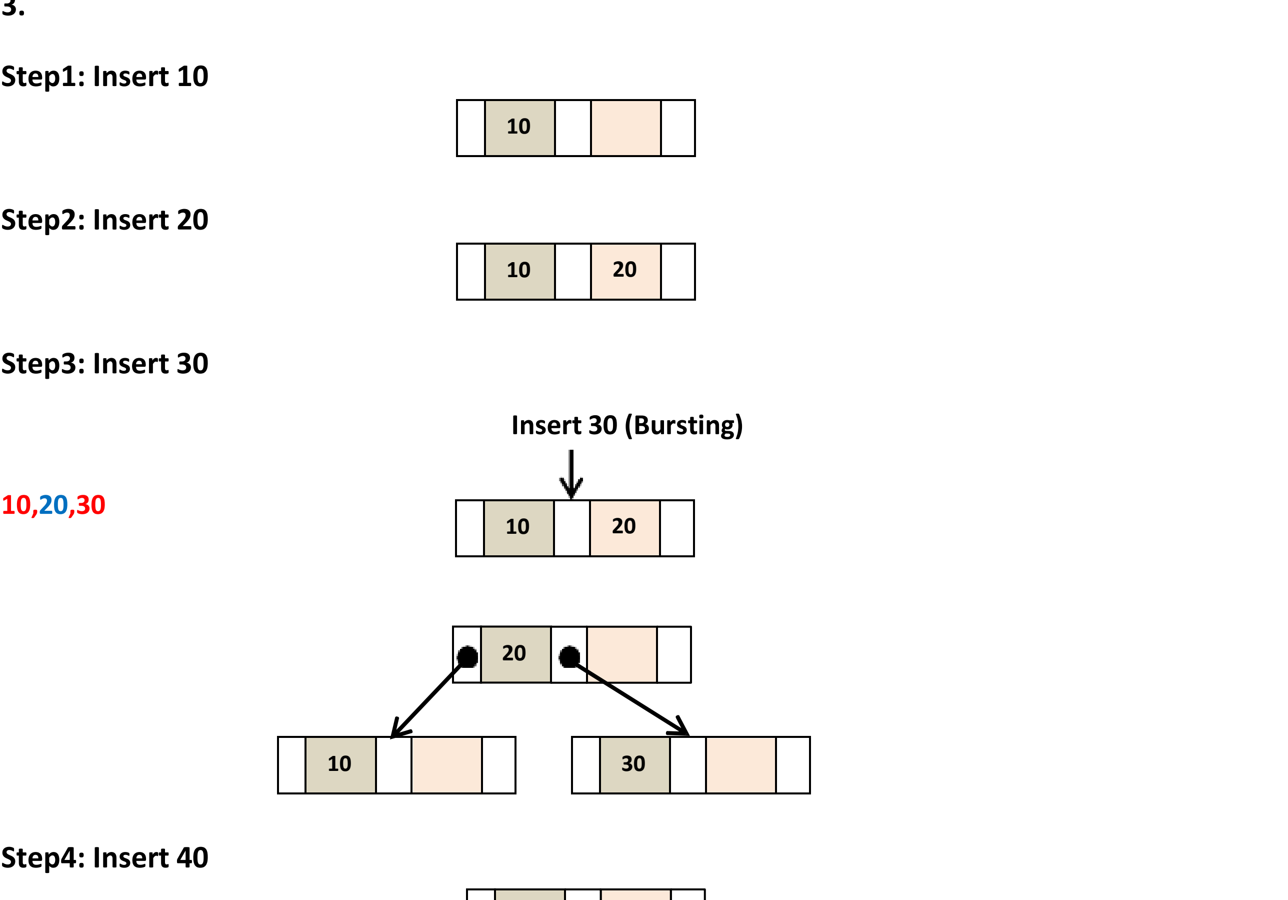


Figure 18.10
B-tree structures. (a) A node in a B-tree with $q - 1$ search values. (b) A B-tree of order $p = 3$. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

Problem 1: Construct B-Tree for the following values 10, 20, 30, 40, 50, 60, 70, 80, 90 of order 3.

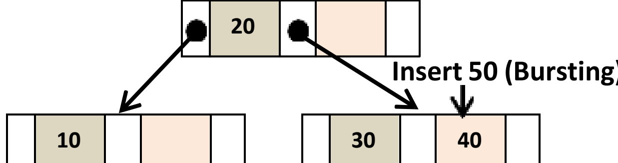


Step4: Insert 40

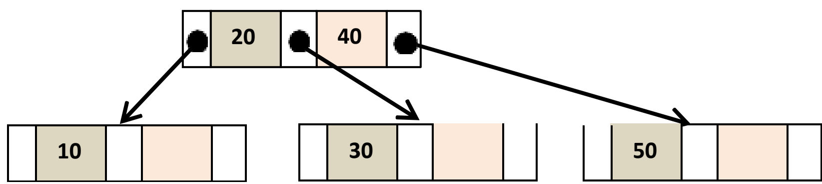
20

10 | 30 | 40

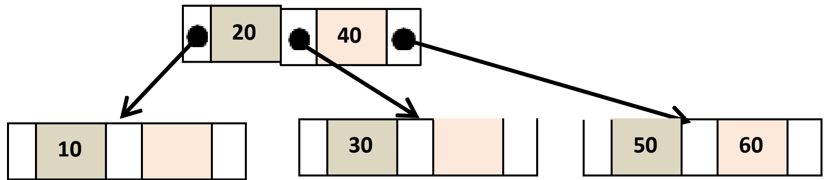
Step5: Insert 50



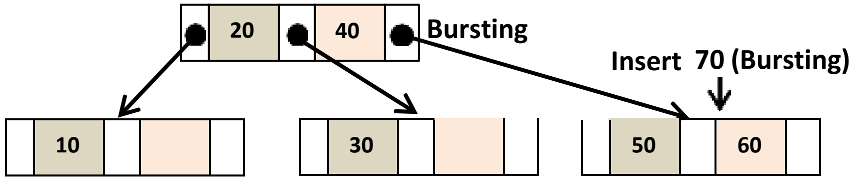
30,40,50



Step 6: Insert 60

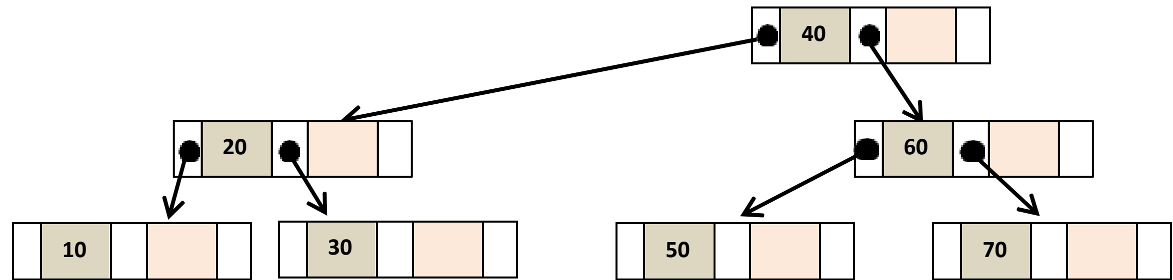


Step 7: Insert 70

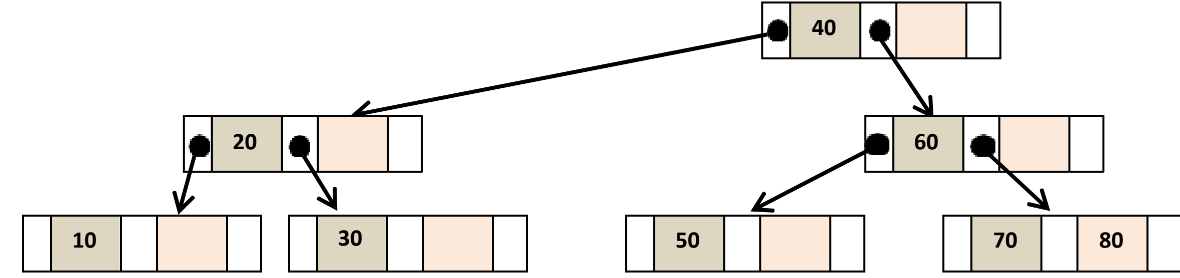


50,60,70

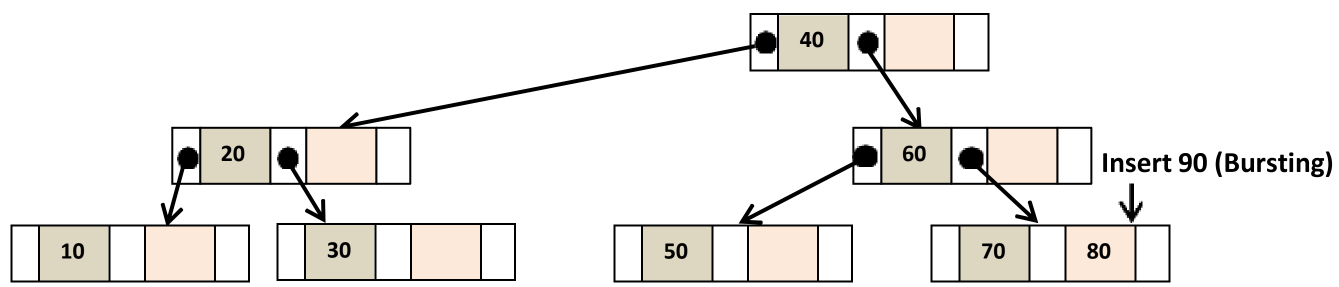
20,40,60



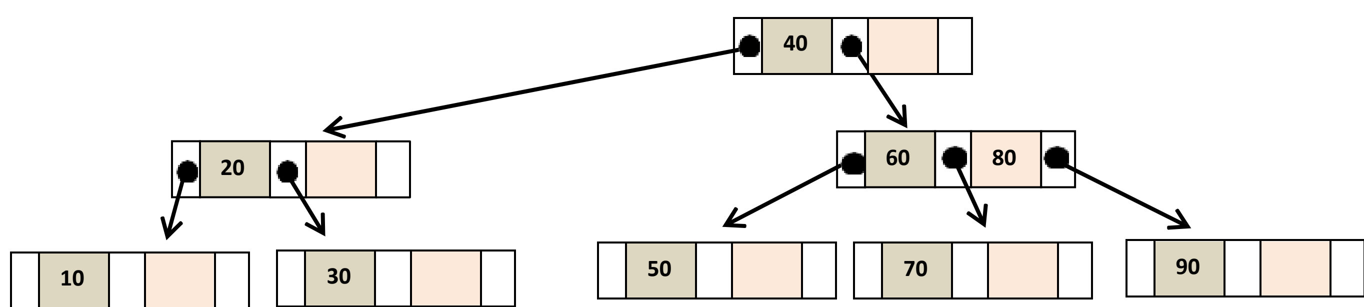
Step 8: Insert 80



Step 9: Insert 90

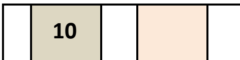


70,80, 90

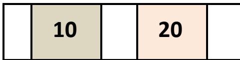


B-Trees

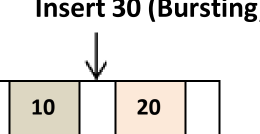
Construct B-Tree for the following values 10, 20, 30, 40, 50, 60, 70, 80, 90 of order 3.

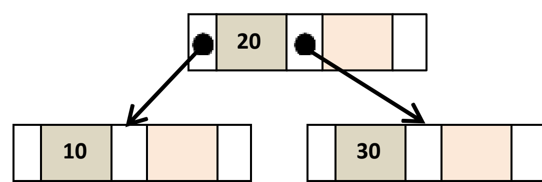


Step2: Insert 20

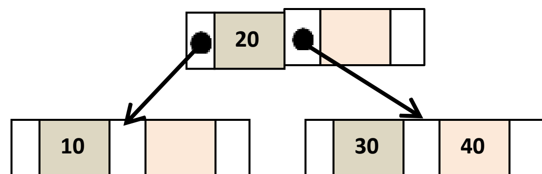


Step3: Insert 30

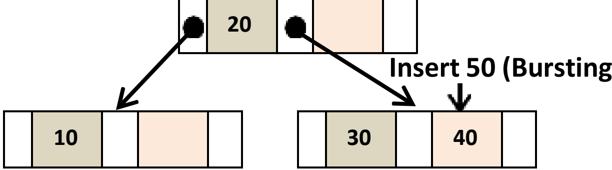




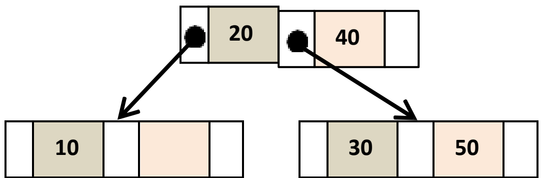
Step4: Insert 40



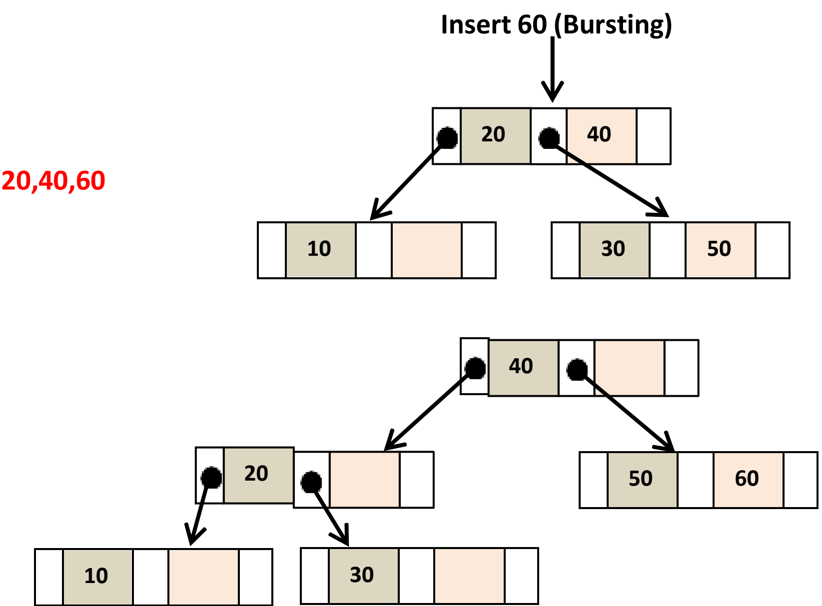
Step5: Insert 50



30,40,50

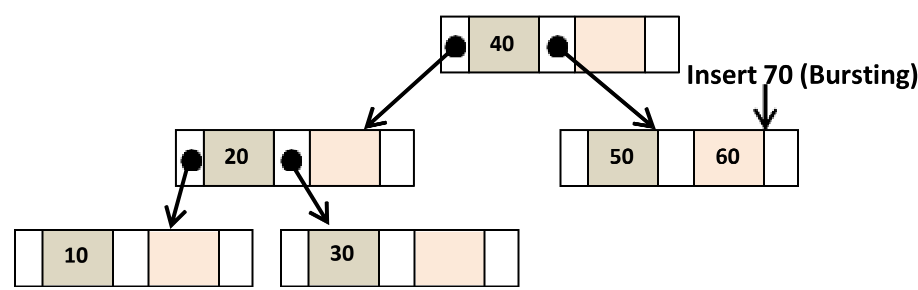


Step 7: Insert 60



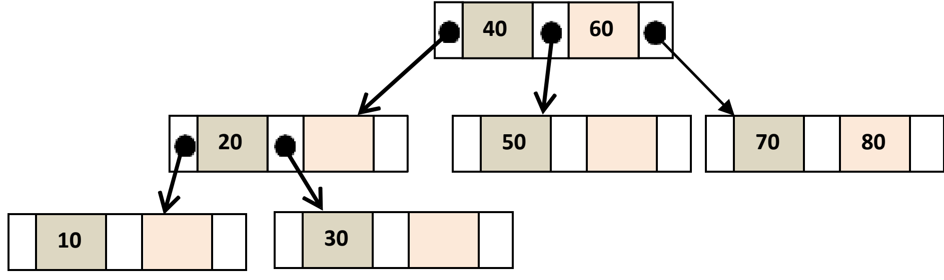
20,40,60

Step 8: Insert 70

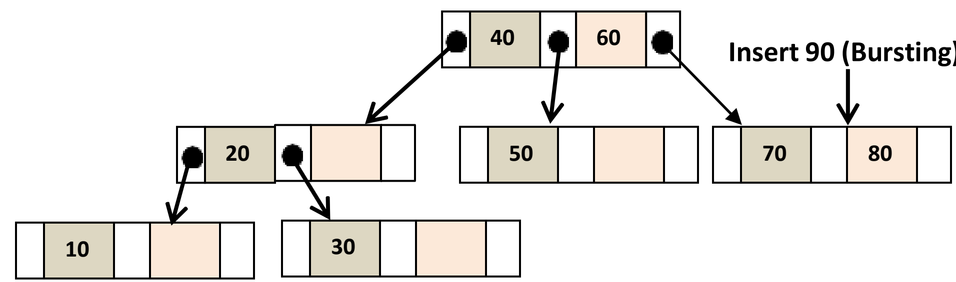


50,60,70

Step 9: Inset 80

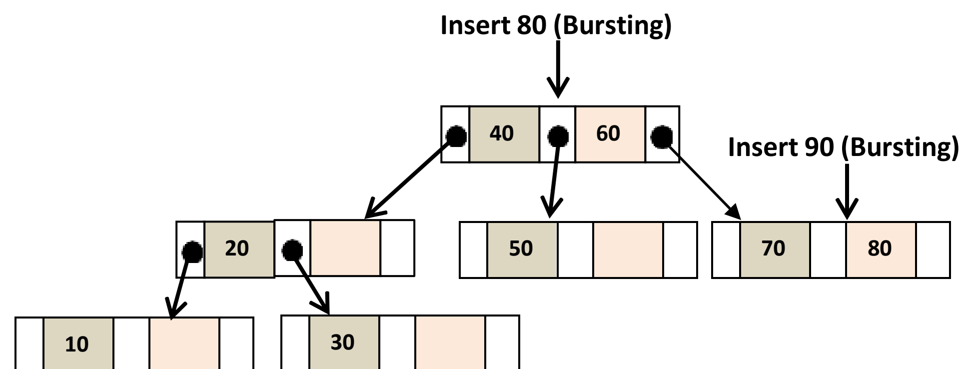


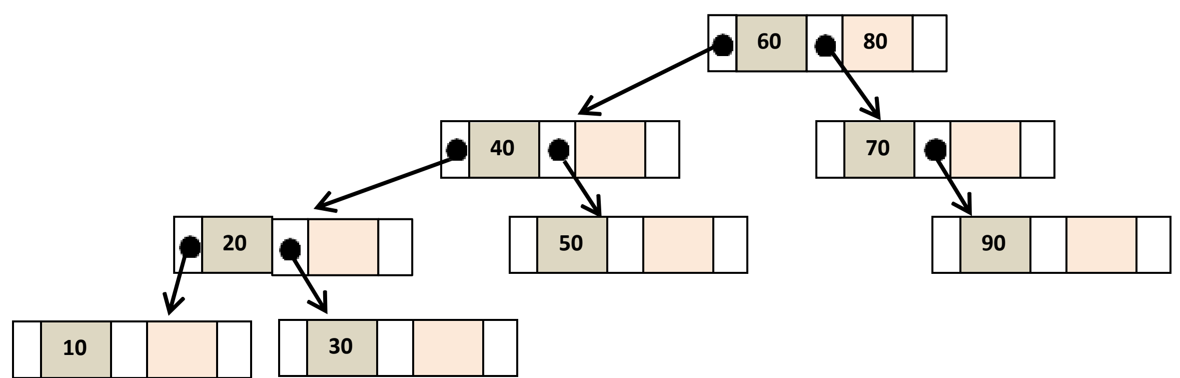
Step 10: Insert 90



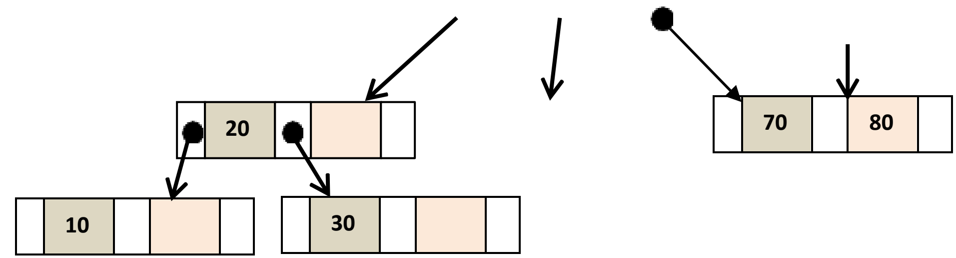
70,80,90

40,60,80





60
80
40
70
20
50
90
10
30

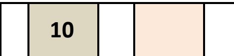


20
70
80
10
30

Date: *27/05/2020* **Subject:** *MCA II Semester* **Time:** *11A.M to 12.15 A.M*
Subject: *DBMS* **Topic:** *B Trees Examples* **Total No Students:** *21* **No of Students Participated:** *15*
Faculty: *Dr.T.S.Ravi Kiran*

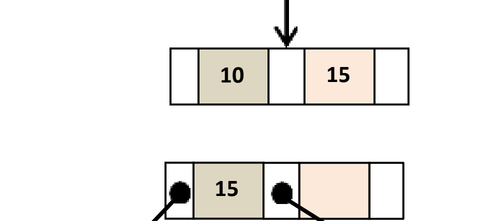
Problem 2: **Construct B-Tree for the following values 10, 15, 20, 25, 30, 35, 40, 45, 50 of order 3.**

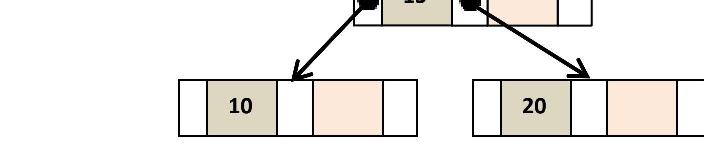
Step1: Insert 10



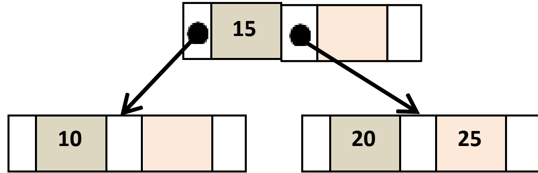
Step2: Insert 15

Step3: Insert 30

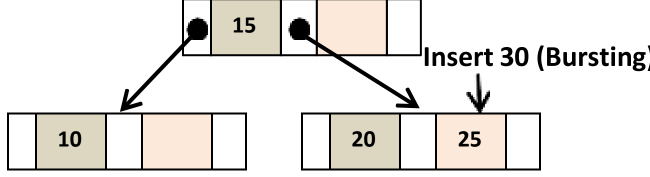




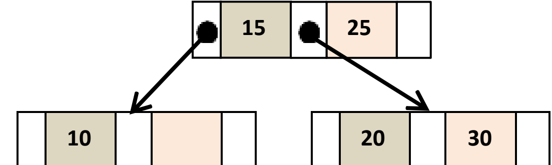
Step4: Insert 25



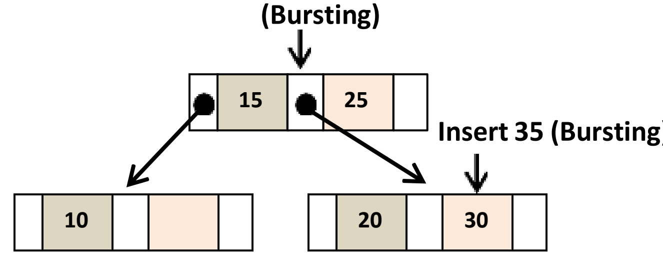
Step5: Insert 30



20,25,30

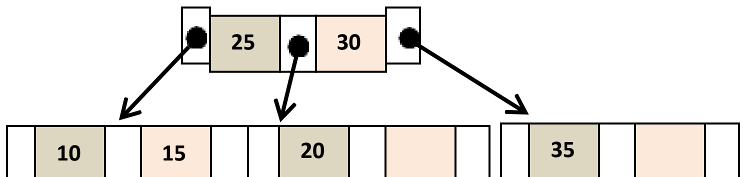


Step 6: Insert 35

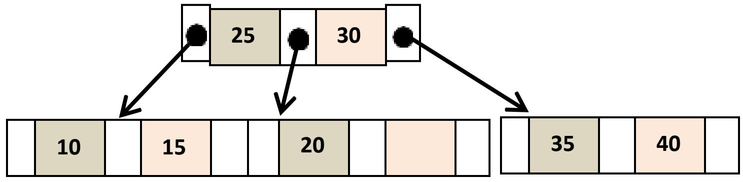


20,30,35

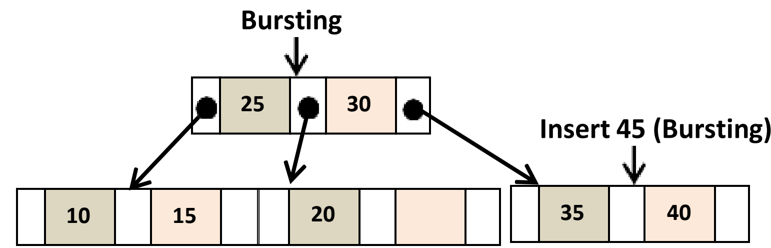
15,25,30



Step 7: Insert 40

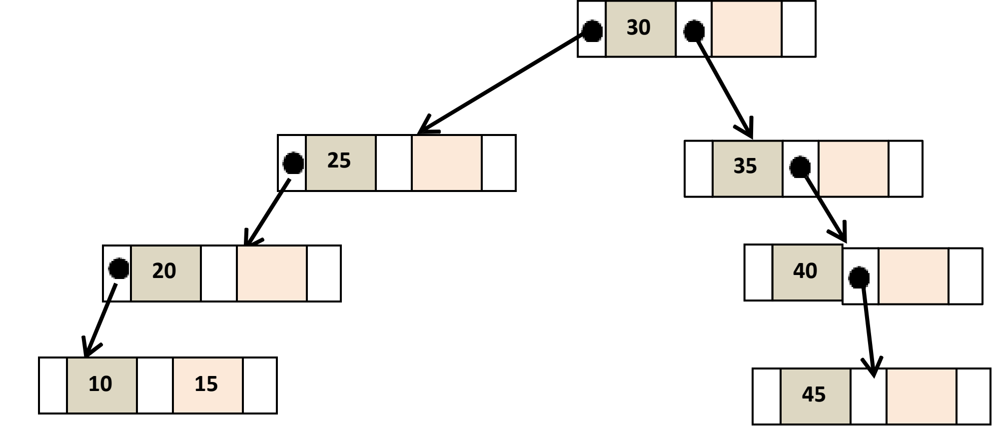


Step 8: Insert 45

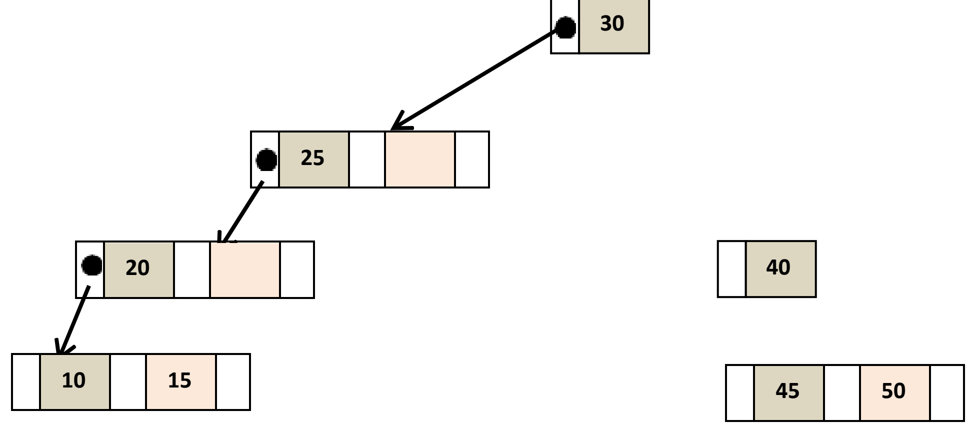


35,40,45

25,30,40



Step 9: Insert 50

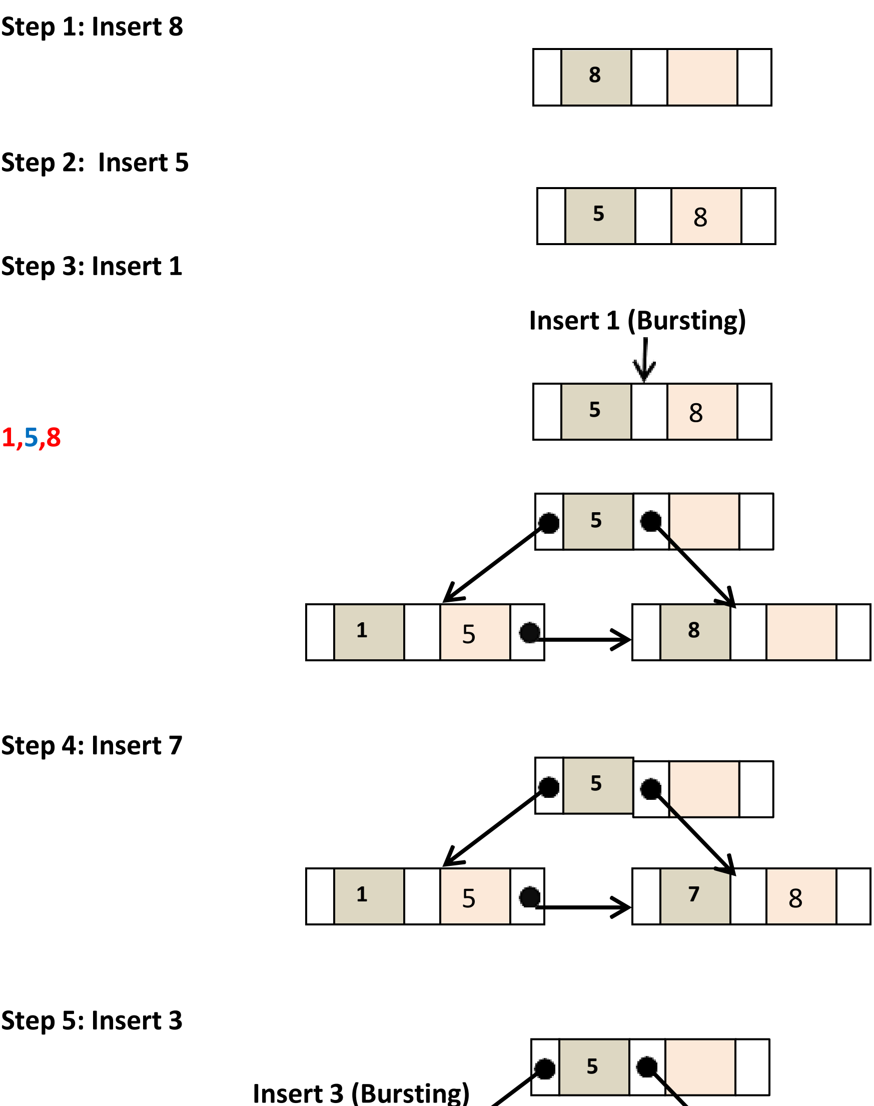


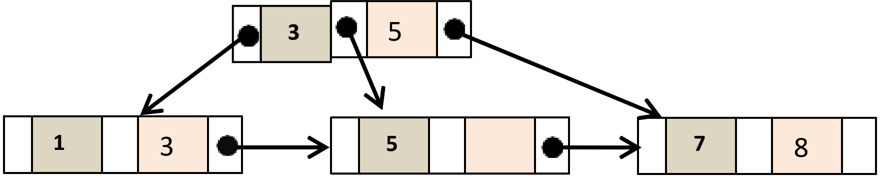
B + TREES

- **B+ Tree:** A B + Tree is a balanced tree in which every path from the root to leaf must have the same length.
- All the values must present in the leaf nodes in ascending order.
- B+ Tree allow redundant storage of key values.

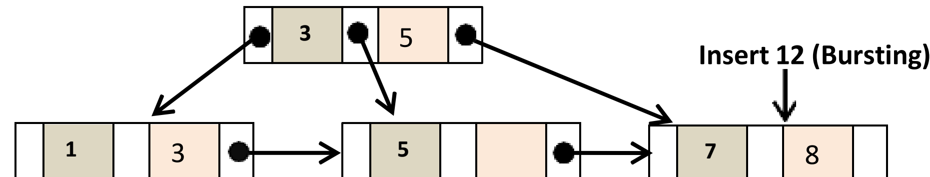
Example 1: Construct B+ Tree for the following sequence 8, 5, 1, 7, 3, 12, 9 of order 3.

Step 1: Insert 8



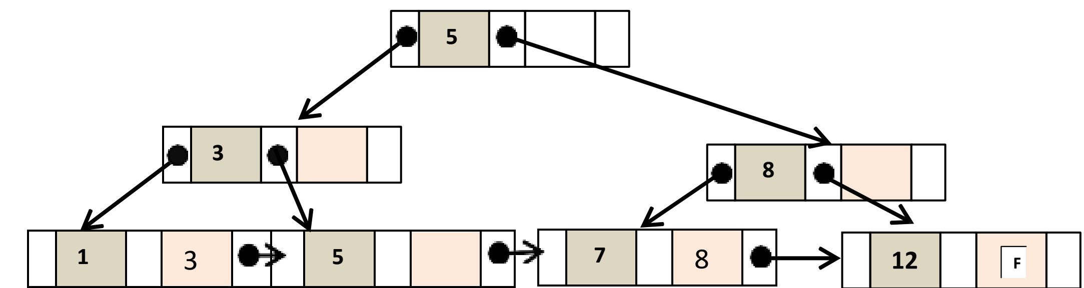


Step 6: Insert 12

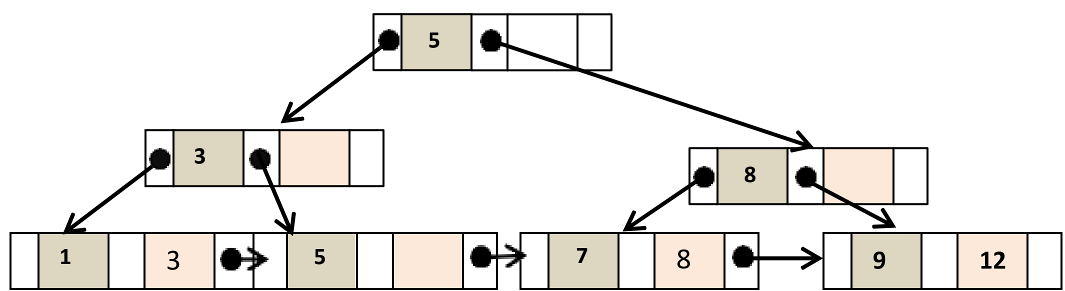


7, 8, 12

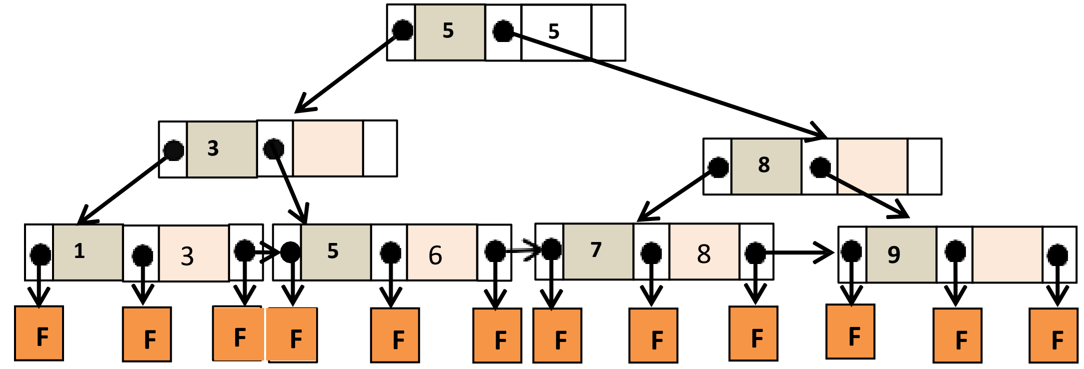
3, 5, 8



Step 7: Insert 9



Step 8: Insert 6

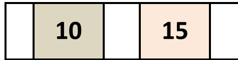


Example 2: Construct B+ Tree for the following sequence 10,15,20,25,30,35,40,45,50 of order 3.

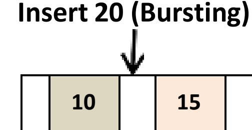
Step 1: Insert 10

10

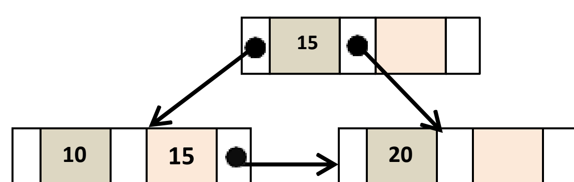
Step 2: Insert 15



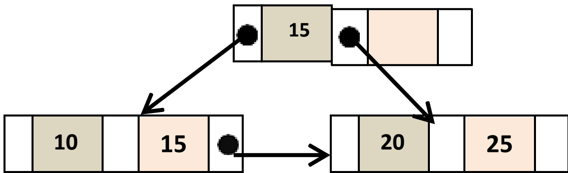
Step 3: Insert 20



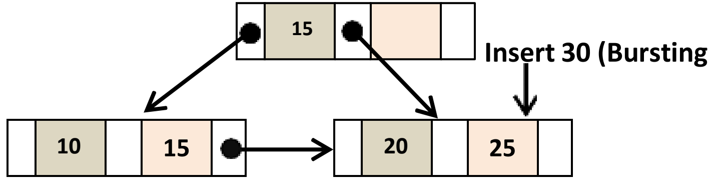
10,15,20



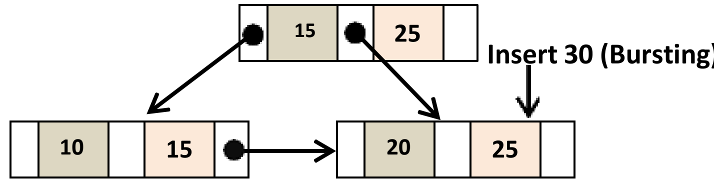
Step 4: Insert 25



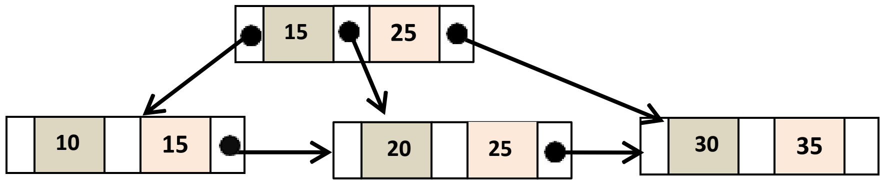
Step 5: Insert 30



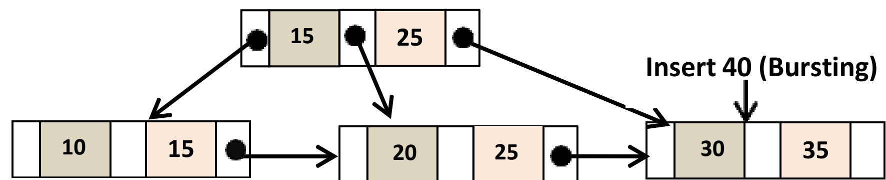
20, 25, 30



Step 6: Insert 35

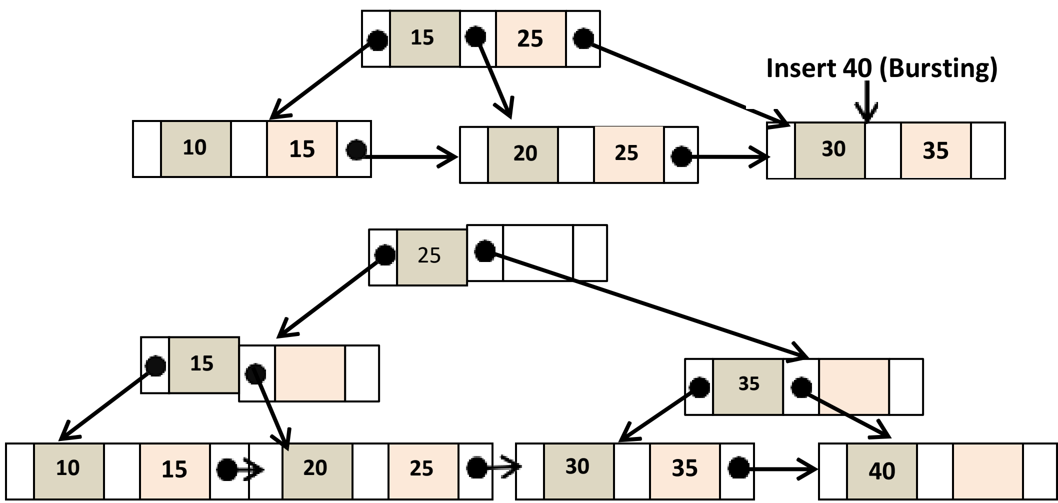


Step 7: Insert 40

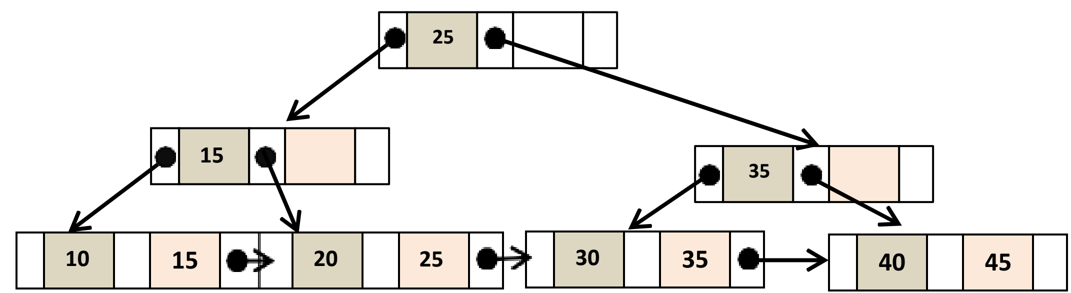


15,25,35

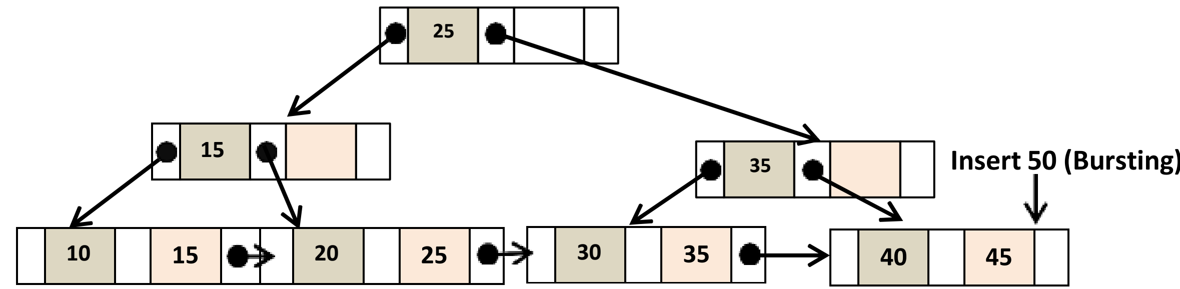
30,35,40



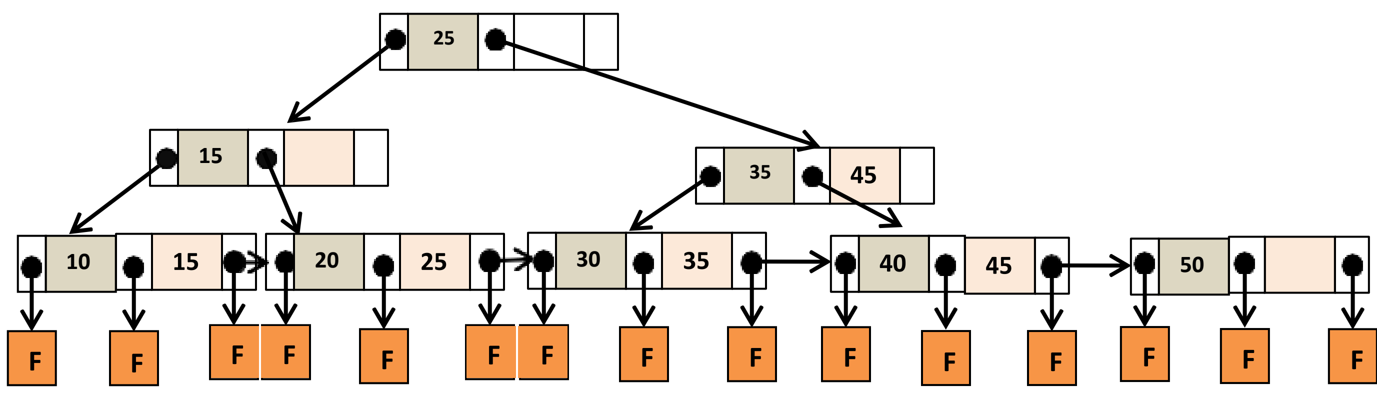
Step 7: Insert 45



Insert 50:



40,45,50



Functional Dependency: The functional dependency is a relationship that exists between two attributes of relation.

NORMALIZATION

First Normal Form

- **First Normal Form: A relation is said to be in first normal form if and only if it contains atomic or scalar values.**

SCP

S#	STATUS	CITY	P#	QTY
S1	{ 20,20,20,20,20,20 }	LONDON	{ P1,P2,P3,P4,P5,P6 }	{ 200,200,400,200,100,100 }
S2	{10,10 }	PARIS	{ P1,P2 }	{ 300,400 }
S3	10	PARIS	P2	200
S4	{ 20,20,20 }	LONDON	{ P2,P4,P5 }	{ 200,300,400 }

Note: The relation should not contain sets

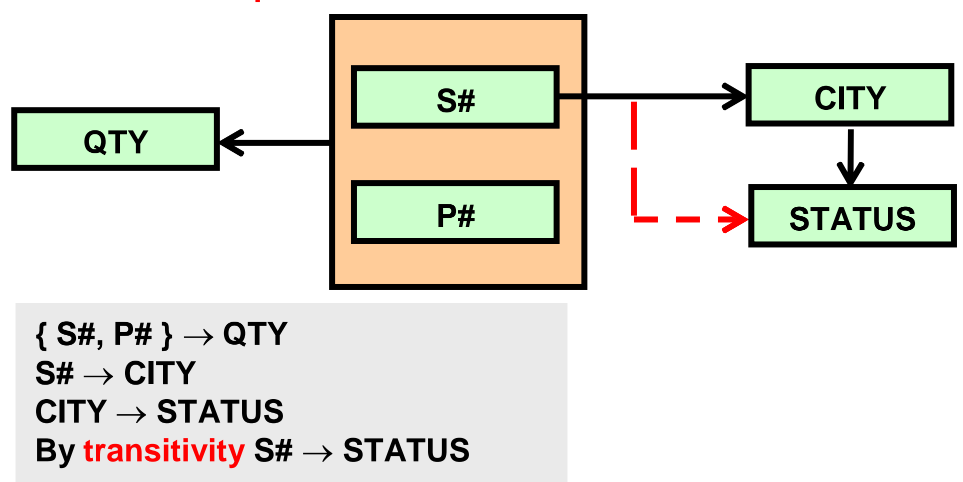
FIRST

S#	STATUS	CITY	P#	QTY
S1			P1	200
S1			P2	200
S1			P3	400
S1			P4	200
S1			P5	100
S1			P6	100
S2			P1	300
S2			P2	400
S3			P2	200
S4			P2	200
S4			P4	300
S4			P5	400

- The relation FIRST stated as

  FIRST{ S#, STATUS, CITY, P#, QTY }
  PRIMARY KEY { S#, P# }

- Functional Dependencies in relation FIRST are



{ S#, P# } → QTY
S# → CITY
CITY → STATUS
By transitivity S# → STATUS

- **UPDATE ANOMALIES**

INSERT: We can not insert particular suppliers located in particular city until the supplier supplies at least one part.

UPDATE: If supplier S1 moves from LONDON to AMSTERDAM, we need to change city as AMSTERDAM for every supplier S1 otherwise the database contains inconsistent values or data.

DELETE: If we delete first tuple of S1 among six tuples we lost the information of part P1.

Second Normal Form

- **The relation FIRST is projected into two relations SECOND, SP**

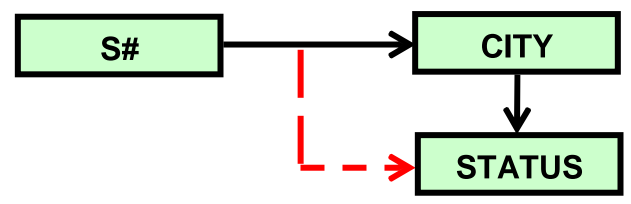
SECOND

S#	STATUS	CITY
S1	20	LONDON
S2	10	PARIS
S3	10	PARIS
S4	20	LONDON
S5	30	ATHENS

- **The relation SECOND is stated as**

SECOND { S#, STATUS, CITY }
PRIMARY KEY S#

- **Functional Dependencies in relation SECOND are**



S# $\rightarrow$ CITY
CITY $\rightarrow$ STATUS
By transitivity S# $\rightarrow$ STATUS

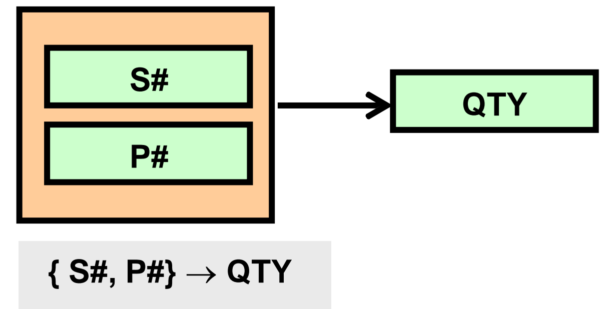
SP

S#	P#	QTY
S1	P1	200
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

- The relation SP is stated as

SP { S#, P#, QTY }
PRIMARY KEY { S#,P# }

- Functional Dependencies in relation SP are



{ S#, P#} → QTY

Second Normal Form: A relation is said to be in 2NF if and only if it is in 1NF and every non key attribute is irreducibly (loss less) depend on key attribute (Primary key).

- **UPDATE ANOMALIES**

INSERT: We can not insert the fact that a particular city have particular status.

E.g. We can not state that any supplier in ROME must have status of 50 until we have some suppliers actually located in that city.

UPDATE: If we need to change STATUS of LONDON from 20 to 40, we should change for all the tuples otherwise the database consists inconsistent data.

DELETE: If we delete second tuple of city PARIS, the information of supplier S2 will be lost.

Third Normal Form

Third Normal Form: A relation is said to be in 3NF if and only if it is in 2NF and every non key attribute non transitively depend upon the key attribute.

Decomposition - A

- Decompose the relation SECOND into two relations SS and CS

SS

S#	STATUS
S1	20
S2	10
S3	10
S4	20
S5	30

- The relation SS is stated as

SS { S#, STATUS }
PRIMARY KEY S#

- Functional Dependencies in relation SECOND are

S# → STATUS

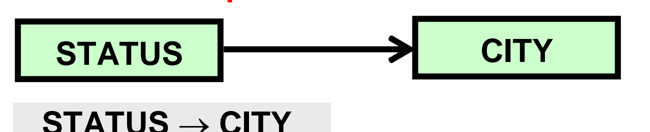
SC

STATUS	CITY
20	LONDON
10	PARIS
30	ATHENS

- The relation SC is stated as

SC { STATUS, CITY }
PRIMARY KEY STATUS

- Functional Dependencies in relation SC are



STATUS → CITY

- This decomposition is not correct since the Functional Dependency S# → STATUS produces transitivity.

Decomposition - B

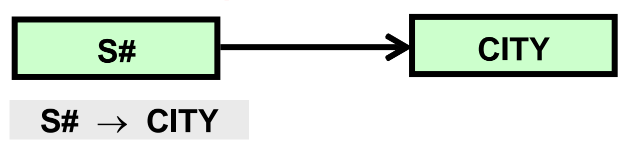
- Decompose the relation SECOND into two relations SC and CS

SC

S#	CITY
S1	LONDON
S2	PARIS
S3	PARIS
S4	LONDON
S5	ATHENS

- The relation SC is stated as

- Functional Dependencies in relation SC are

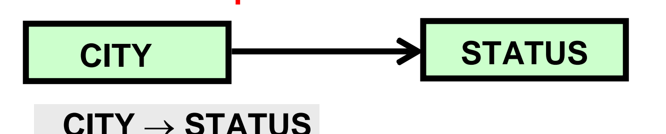


CS

CITY	STATUS
LONDON	20
PARIS	10
ATHENS	30

- The relation CS is stated as

- Functional Dependencies in relation CS are



- This decomposition is correct .Therefore the relations SC and CS satisfies 3NF.

- UPDATE ANOMALIES

  INSERT: We can not insert the fact that a particular city have particular status.

  E.g. We can not state that any supplier in ROME must have status of 50 until we have some suppliers actually located in that city.

  UPDATE: If supplier S1 moves from LONDON to AMSTERDAM we need to change city as AMSTERDAM for every supplier S1 otherwise database contains inconsistent data.

  DELETE: If we delete second tuple of city PARIS the information of S2 will be lost.

Dependency Preservation

- Consider the relation **SECOND**

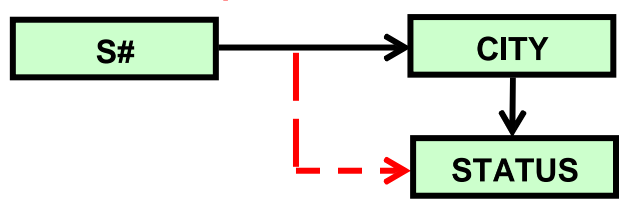
SECOND

S#	STATUS	CITY
S1	20	LONDON
S2	10	PARIS
S3	10	PARIS
S4	20	LONDON
S5	30	ATHENS

- The relation **SECOND** is stated as

SECOND { S#, STATUS, CITY }
PRIMARY KEY S#

- **Functional Dependencies** in relation **SECOND** are



S# → CITY
CITY → STATUS
By transitivity S# → STATUS

- Decompose the relation **SECOND** into two relations **SC** and **CS**

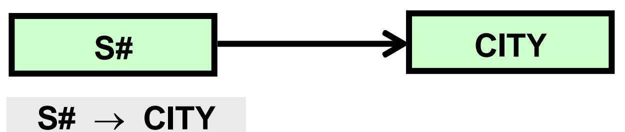
SC

S#	CITY
S1	LONDON
S2	PARIS
S3	PARIS
S4	LONDON
S5	ATHENS

- The relation **SC** is stated as

SC { S#, CITY }
PRIMARY KEY S#

- **Functional Dependencies** in relation **SC** are



S# → CITY

CS

CITY	STATUS
LONDON	20
PARIS	10
ATHENS	30

- The relation **CS** is stated as

SC { CITY, STATUS }
PRIMARY KEY CITY

- **Functional Dependencies** in relation **CS** are

CITY → STATUS

- Dependency preservation: Functional dependencies of the projected tables are implied by the dependencies of original table.

- From the above tables the FD's of SC and CS are implied by the FD's of SECOND. Therefore dependencies are preserved.

BCNF (Boycee Code Normal Form)

BCNF: A relation is said to be in BCNF if and only if it is in 3NF and only determents are candidate keys.

- Candidate key : An attribute (or) group of attributes which is used to distinguish one record with another record.

- Consider the relation SECOND

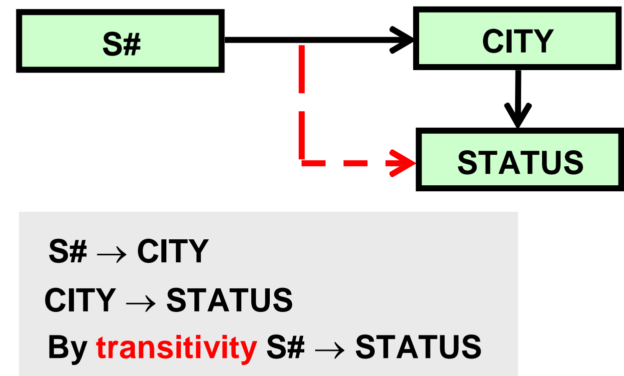
SECOND

S#	STATUS	CITY
S1	20	LONDON
S2	10	PARIS
S3	10	PARIS
S4	20	LONDON
S5	30	ATHENS

- The relation SECOND is stated as

SECOND { S#, STATUS, CITY }
PRIMARY KEY S#

- Functional Dependencies in relation SECOND are



S# → CITY
CITY → STATUS
By transitivity S# → STATUS

- The above relation does not doesn't satisfies 3NF and all the determinants are not the candidate keys. Therefore the relation SECOND is not in BCNF.

- The relation **SECOND** is projected into two relations **SC** and **CS**

SC

S#	CITY
S1	LONDON
S2	PARIS
S3	PARIS
S4	LONDON
S5	ATHENS

- The relation **SC** is stated as

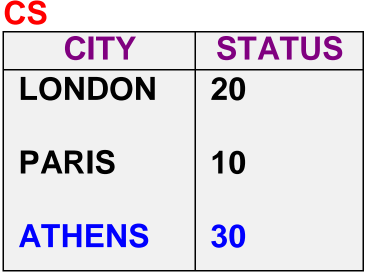
  SC { S#, CITY }
  PRIMARY KEY S#

- **Functional Dependencies** in relation **SC** are

S# $\rightarrow$ CITY

- In the relation **SC** is the only determinant **S#** is the **primary key** .

CS

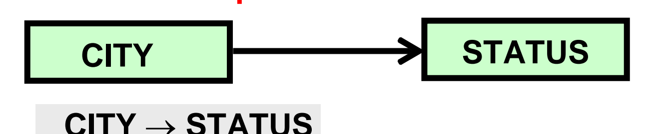


CITY	STATUS
LONDON	20
PARIS	10
ATHENS	30

- The relation **CS** is stated as

  SC { CITY, STATUS }
  PRIMARY KEY CITY

- **Functional Dependencies** in relation **CS** are



CITY $\rightarrow$ STATUS

- In the relation **CS** is the only determinant **CITY** is the **primary key** .
- There fore the relations **SC** and **CS** satisfies BCNF and all the determinants are candiadate keys

---

4NF (Fourth Normal Form)

- Consider the relation UCTX (Un normalized relation)

UCTX

COURSE	TEACHER	TEXT
PHYSICS	{Prof. GREEN, Prof. BROWN}	{ BASIC MECHANICS, PRINCIPLES OF OPTICS}
MATHS	{Prof.GREEN}	{BASIC MECHANICS, VECTOR ANALYSIS, TRIGNOMETRY}

- Constraints on table UCTX

1. A specified course can be taught by specified teacher and uses all the specified textbooks as references.
2. A teacher can teach any number of courses.
3. There are no functional dependencies for the table UCTX.

- Now we are normalizing UCTX table as CTX

CTX

COURSE	TEACHER	TEXT
PHYSICS	Prof. GREEN	BASIC MECHANICS
PHYSICS	Prof. GREEN	PRINCIPLES OF OPTICS
PHYSICS	Prof. BROWN	BASIC MECHANICS
PHYSICS	Prof. BROWN	PRINCIPLES OF OPTICS
MATHS	Prof. GREEN	BASIC MECHANICS
MATHS	Prof. GREEN	VECTOR ANALYSIS
MATHS	Prof. GREEN	TRIGNOMETRY

- The relation CTX is stated as

CTX {COURSE, TEACHER,TEXT }
CANDIDATE KEY {COURSE, TEACHER, TEXT}

- The above relation satisfies 1NF,2NF, 3NF and BCNF since the determinant itself is the candidate key.

- **Update anomalies:**
To add information about a teacher who teaches a course PHYSICS, it is necessary to create two tuples one for each of the text books BASIC MECHANICS and PRINCIPLES OF OPTICS.

- MVD'S: Multi valued dependencies are general dependencies based on the attributes and their domains.

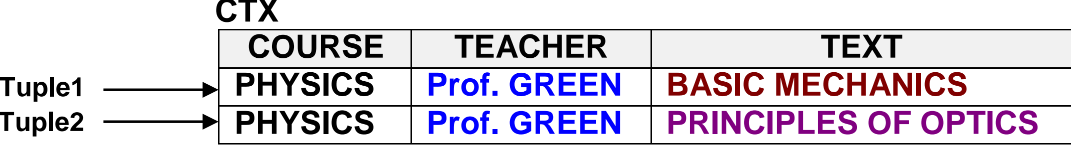
- Let R be a relation and let A,B,&C be arbitrary subsets of R then we can say that
A ⟶>> B if and only if the set B values matching with pair (A,C) dependents only on A value independent of C value.

COURSE ⟶>> TEXT

COURSE ⟶>> TEACHER

The above MVD's exits in CTX.

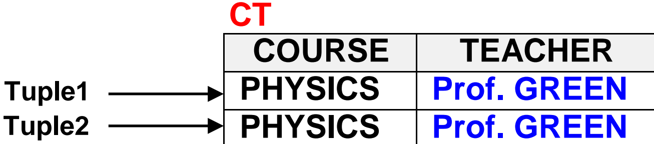
- Consider first two tuples from the relation CTX.



CTX

	COURSE	TEACHER	TEXT
Tuple1 →	PHYSICS	Prof. GREEN	BASIC MECHANICS
Tuple2 →	PHYSICS	Prof. GREEN	PRINCIPLES OF OPTICS

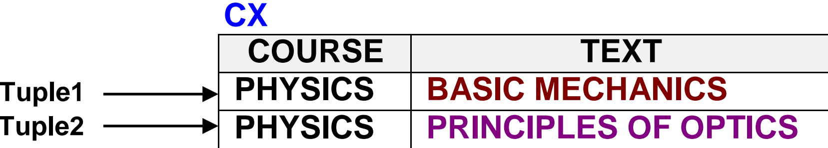
- Consider two subsets CT and CX from CTX



CT

	COURSE	TEACHER
Tuple1 →	PHYSICS	Prof. GREEN
Tuple2 →	PHYSICS	Prof. GREEN

Tuple1 = Tuple2



CX

	COURSE	TEXT
Tuple1 →	PHYSICS	BASIC MECHANICS
Tuple2 →	PHYSICS	PRINCIPLES OF OPTICS

Tuple1 $\neq$ Tuple2

- Therefore the relation CTX violating the condition of FD.
- Project the table CTX into CT and CX

CT

COURSE	TEACHER
PHYSICS	Prof. GREEN
PHYSICS	Prof. BROWN
MATHS	Prof. GREEN

- The relation CT is stated as

CT {COURSE, TEACHER}
CANDIDATE KEY {COURSE, TEACHER}

CX

COURSE	TEXT
PHYSICS	BASIC MECHANICS
PHYSICS	PRINCIPLES OF OPTICS
MATHS	BASIC MECHANICS
MATHS	VECTOR ANALYSIS
MATHS	TRIGNOMETRY

- The relation CX is stated as

CTX {COURSE, TEXT }
CANDIDATE KEY {COURSE, TEXT}

- Therefore the relations CT and CX satisfies 1NF,2NF,3NF,BCNF and preserves MVD's and FD's .

- Therefore the relations CT and CX satisfies 4NF.

4NF: The relation is said to be in 4NF if and only it is in BCNF and must have both MVD's and FD's.

Fagin's theorem: Let R=A,B,C be a relation where A,B,C are set of attributes the R = join of its projection on {A,B} and {A,C} if and only if R satisfies A →>> B and A→>> C

5NF (Fifth Normal Form)

Join Dependencies

- Consider the following relation

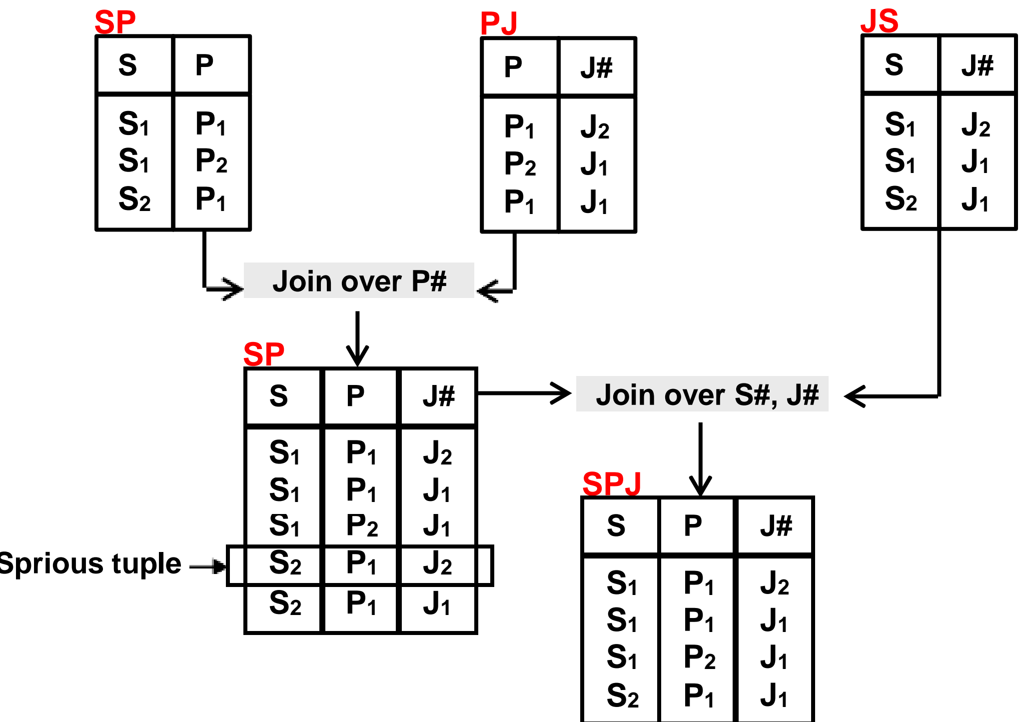
SPJ

S#	P#	J#
S_1	P_1	J_2
S_1	P_2	J_1
S_2	P_1	J_1
S_1	P_1	J_1

- The relation SPJ is stated as

SPJ {S#,P#,J#}
CANDIDATE KEY { S#,P#,J#}

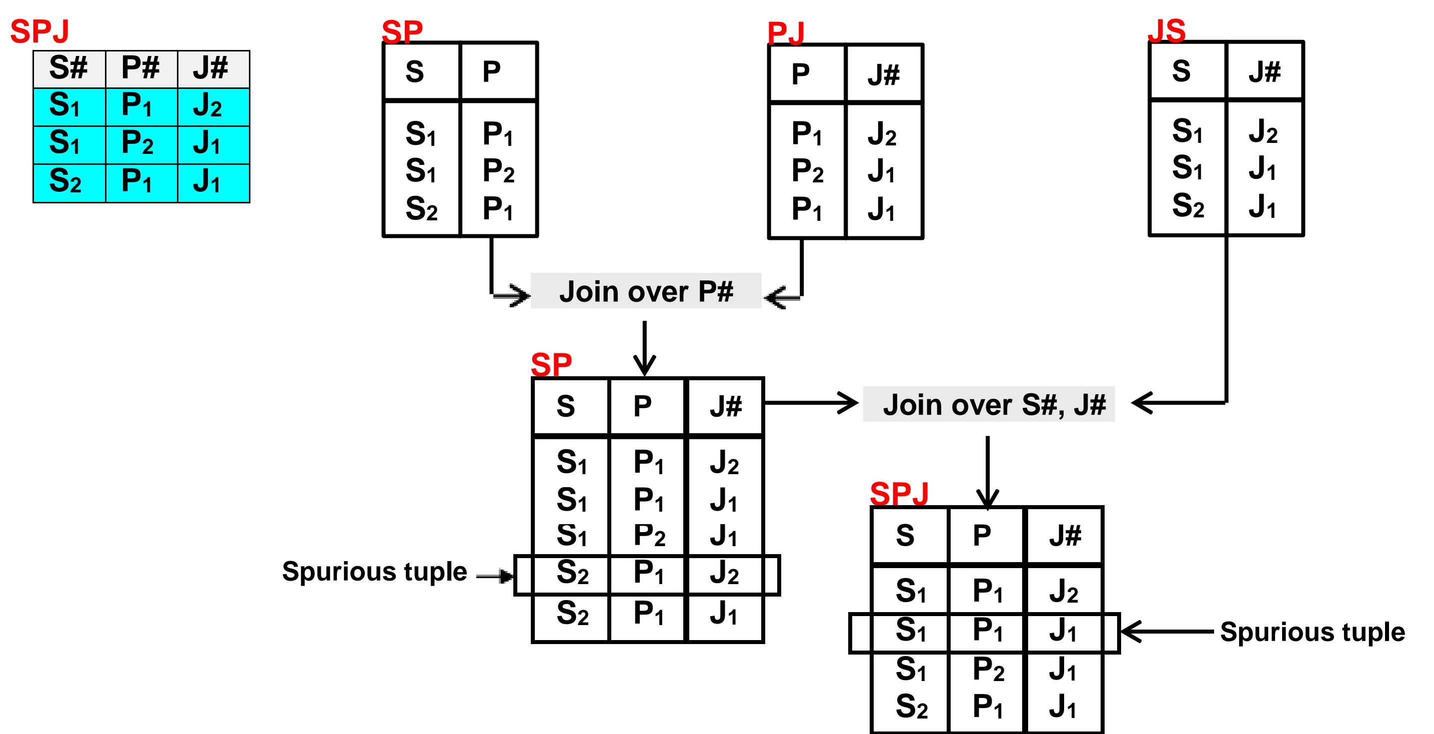
- **Join Dependencies**: Let R be a relation and let A,B,…,Z be arbitrary subsets of set of attributes of R. Then we can say that R satisfies join dependency *(A,B,…,Z) if and only if R is equal to join of its projections on A,B,…,Z

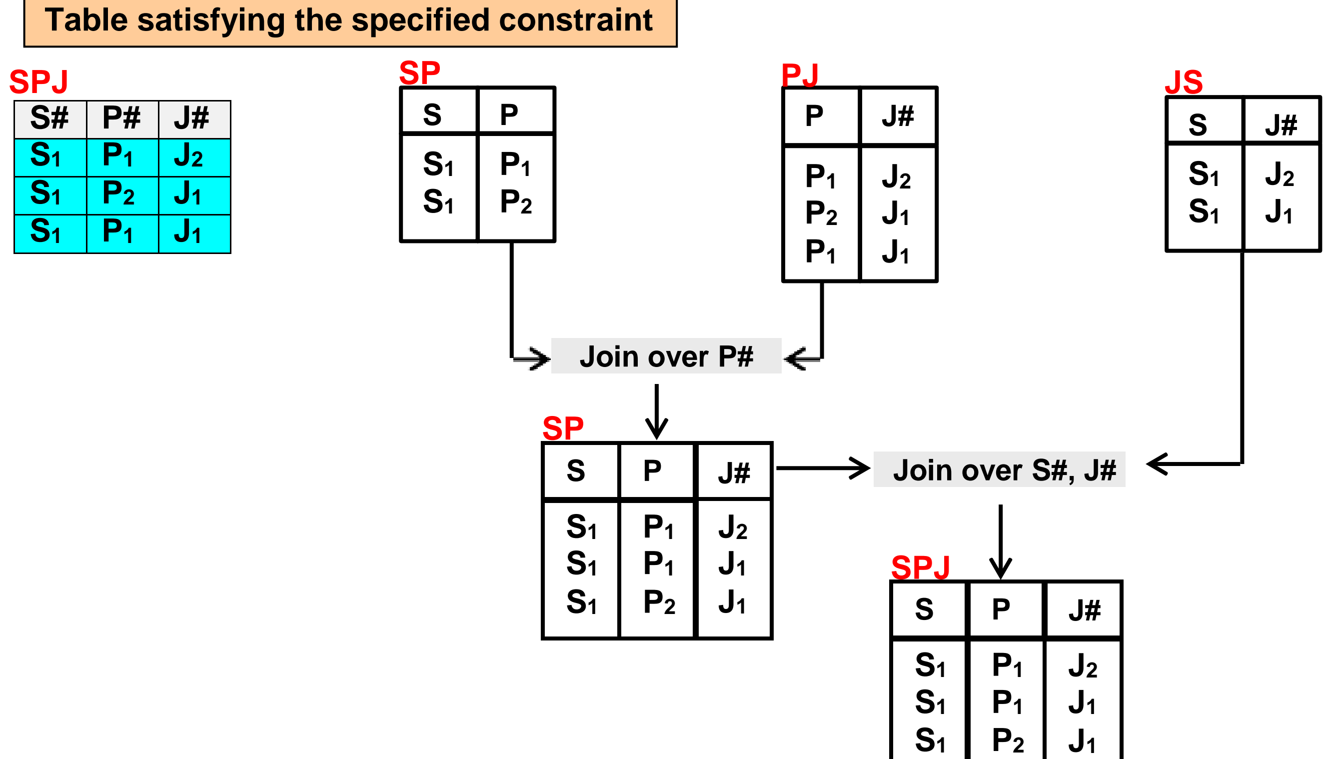


- Constraint on the following table is

If the pair (S_1, P_1) appear in SP, the pair (P_1, J_1) appear in PJ the the pair (J_1, S_1) appear in JS then the triple (S_1, P_1, J_1) appear in SPJ

Table violating the specified constraint





5NF: A relation R is in **5NF** is also called **PJNF** (**Projection Join Normal From**) if and only if **every join dependency** in R is implied by the **candidate key of R.**

- Consider the relation **SPJ**

  SPJ {S#,P#,J#}
  CANDIDATE KEY **{ S#,P#,J#}**

- SPJ is projected into **SP**,**PJ** and **JS**

  SP {S#,P#}
  CANDIDATE KEY **{ S#,P# }**

  PJ {S#,P#}
  CANDIDATE KEY **{ P#,J# }**

  JS {J#,S#}
  CANDIDATE KEY **{ J#,S# }**

- **Candidate keys of SP, PJ and JS is not implied by the candidate key of SPJ. Therefore SP, PJ and JS are not satisfies 5NF.**

- **Consider the relation**

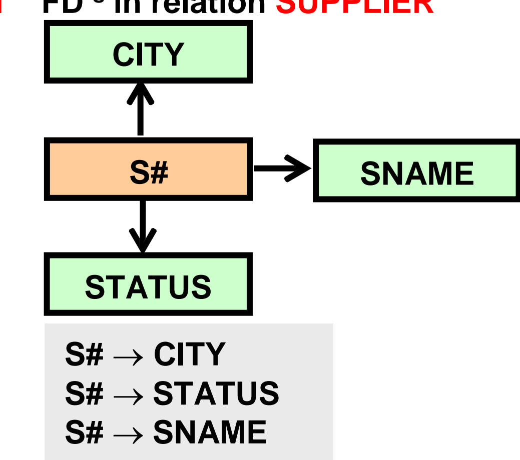
SUPPLIER

S#	SNAME	STATUS	CITY
S_1 S_2	SMITH JOHN	20 20	LONDON PARIS

- **The relation SUPPLIER is stated as**

SPJ {S#,SNAME,STAYUS,CITY}
CANDIDATE KEY S#

- **FD'S in relation SUPPLIER**



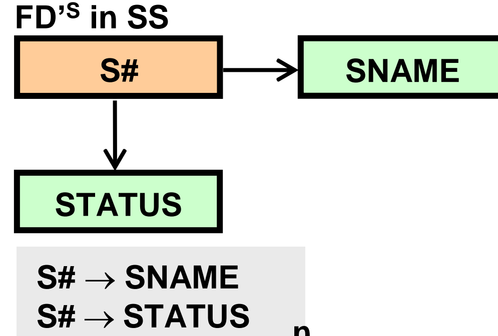
- **The above relation satisfies 1NF,2NF, 3NF, BCNF (No need to check 4NF since there are FD'S).**

- **The relation SUPPLIER is projected into SS and SC.**

SS

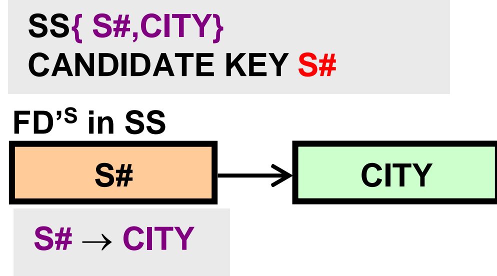
S#	SNAME	STATUS
S1	SMITH	20
S2	JOHN	10

FD'S in SS



SC

S#	CITY
S1	LONDON
S2	PARIS



- The relations SS and SC satisfies 5NF since the candidate keys of SS and CS is implied by candidate key of SUPPLIER.

Domain Key Normal Form

1. This is proposed by FAGIN and it is also called FAGIN NF.
2. It is not defined in terms of FD's ,MVD's,JD's .

A relation R is said to be in DKNF if and only if it should contain KEY CONSTRAINT and DOMAIN CONSTRAINT.

- Key constraint : The relation must have primary key .
- Domain constraint : The set of values associated for an attribute must have finite values.

RELATIONAL ALGEBRA & RELATIONAL CALCULUS

TOPIC 1: SELECT OPERATION (σ) VIMP

- **Used to select subset of tuples (rows) from relation (table) that satisfies a select condition.**

Syantax: $\sigma_{<Select\ Condition>}(R)$

Syntax for Select Condition:

1. <attribute_name> ***<comparision_operation>*****<constrant_value>**

E.g. sal > 5000;

2. <attribute_name> ***<comparision_operation>*****<attribute_name>**

E.g. e.esal > d.esal

Note: e and d are table aliases

3.<selection_condition>***<boolean_operator>*****<selection_condition>**

E.g. Ename= 'James' and Salary>5000

- **Comparison operators: There are normally one of the operations from the set {=, <, >, ≤, ≥, ≠}**
- **Boolean operators: Boolean operation is normally one of the operations of the set {AND, OR, NOT}**
- **Selection operations is commutative:**
  $\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$
- **We can always combine a cascade (or sequence) of SELECT operations into a single SELECT operation with a conjunctive (AND) condition; that is,**
  $\sigma_{<cond1>}(\sigma_{<cond2>}(\ldots(\sigma_{<condn>}(R))\ldots)) = \sigma_{<cond1>\ AND <cond2>\ AND \ldots AND\ <condn>}(R)$

Qeury 0: Select list of employees who work for department number 4

Sql query:

```
SELECT *
FROM EMPLOYEE
WHERE dno=4 AND salary>25000;
```

Select expression:

$\sigma_{Dno=4\ AND\ Salary>25000}$ **(EMPLOYEE)**

TOPIC 2: PROJECT OPERATION (∏) VIMP

- **Selects certain columns from a relation.**

Syntax: $\prod_{<attribute_list>}(R)$

- **Degree of relation: Number of attributes in a relation.**

- **Duplicate elimination: Duplicate rows are eliminated when projection operation done.**

- **Super key: Key may have *single attribute* or *group of attributes* used to distinguish one record with another record.**

  $\prod_{<list1>} (\prod_{<list2>}(R)) = \prod_{<list1>} (R)$ **when list1 and list2 is same.**

Qeury 0: List each employee first name, last name and salary from the relation employee.

Sql query:

```
SELECT fname,lname,salary .
FROM EMPLOYEE;
```

Project expression:

$\prod_{<fname,lname,salary>} (EMPLOYEE)$

TOPIC 3: RENAME OPERATION (ρ)

- **To rename relation and columns of the relation.**

Syntax to rename relations and attributes: $\rho_{S\ (B1,B2,B3,...,Bn)} (R)$

Syntax to rename relation: $\rho\ (R)$

Qeury 0: Retrive the first name, last name and salary of employees who work for department 5.

Sql query:

```
SELECT fname,lname,salary
FROM EMPLOYEE
WHERE dno=5.
```

Rename expression:

$DEP5_EMPS \leftarrow \sigma_{dno=5} (EMPLOYEE)$

$RESULT \leftarrow \prod_{fname,lname,salary} (DEP5_EMPS)$

TOPIC 4: SET OPERATIONS VIMP

UNION: Include all the tuples in R and S by eliminating duplicate rows.

- **The result of operation is represented by R ∪ S.**

R	SSN
	123456789
	333445555
	666884444
	453453453

S	SSN
	333445555
	888665555

RUS	RUS
	123456789
	333445555
	666884444
	453453453
	888665555

INTERSECTION: Includes all the tuples that are common in both R and S.

- **The result of operation is represented by R ∩ S.**

R∩S	R∩S
	888665555

DIFFERENCE: Includes all the tuples that are in 'R' but not in 'S'.

- **The result of operation is represented by R – S.**

R – S	R – S
	123456789
	666884444
	453453453

TOPIC: JOIN OPERATION IN RELATIONAL ALGEBRA VIMP

Join Operation: It is used to combine tuples from multiple relations so that related information can be presented.

- **It is denoted by join symbol ⋈.**

Syntax: R $\bowtie_{\text{<Join Condition>}}$ S

Types of joins:

(a) Theta Join: A general join condition.

- **Syntax: Ai θ Bi**
- **θ is one of the comparison operators of {=,>,>=,<,<=, ≠}**

(b) Equi Join: Join based on the common columns in different relations.

Syntax: R $\bowtie_{\text{<Column = Column>}}$ S

E.g. DEPARTMENT $\bowtie_{\text{<MGRSSN = SSN>}}$ EMPLOYEE

DEPT		
ENO	DEPTNO	DNAME
1	10	RESEARCH
2	20	OPERATIONS

EMP	
ENO	ENAME
1	KING
2	BLACK

DEPT			
ENO	ENAME	DEPTNO	DNAME
1	KING	10	RESEARCH
2	BLACK	20	OPERATIONS

(c) Non Equi Join: Join of two relations.

- **Non equi join always return "*Cartesian Product*" or "*Cross Product*"**
- **It is also known as natural join.**
- **It is denoted by ***

Syntax: R $*_{\text{(<list1> <list2>)}}$ S

E.g. DEPARTMENT ⋈ EMPLOYEE

EMP	
ENO	ENAME
1	KING
2	BLACK

DEPT	
DEPTNO	DNAME
10	RESEARCH
20	OPERATIONS

EMP_CROSS_DEPT			
ENO	ENAME	DEPTNO	DNAME
1	KING	10	RESEARCH
1	KING	10	OPERATIONS
2	BLACK	20	RESEARCH
2	BLACK	20	OPERATIONS

(d) Outer Join: Join condition that returns tuples with direct map.

(i) Left Outer Join: Returns the unmatched tuples in the left side relation.

- It is denoted by the symbol ⟕.

Syntax: R ⟕<Condition> S

EMP	
ENAME	DEPTNO
KING	10
BLAKE	30
CLARK	10
JONES	20
JAMES	40

DEPT	
DEPTNO	DNAME
10	ACCOUNTING
20	SALES
30	RESEARCH

Note: The tuple with DEPTNO 40 in EMP relation does not participate in the relation.

(ii) Right Outer Join: Join condition that returns tuples in the right side relation.

- It is denoted by the symbol ⟖

Syntax: R ⟖ S

EMP	
ENAME	DEPTNO
KING	10
BLAKE	30
CLARK	10
JONES	20

DEPT	
DEPTNO	DNAME
10	ACCOUNTING
20	SALES
30	RESEARCH
40	OPERATIONS

Note: The tuple with DEPTNO 40 in relation DEPT does not participates in relation.

(iii) Full Outer Join: Join condition that returns tuples in both the relations that participates relation.

- It is denoted by the symbol ⟗ .

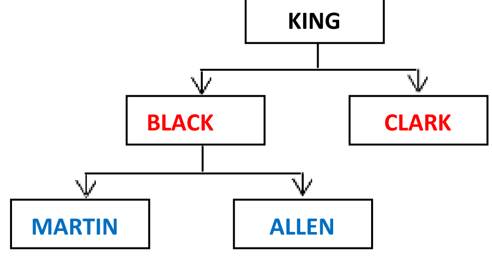
Syntax: R ⟗ S

EMP	
ENAME	DEPTNO
KING	10
BLAKE	30
CLARK	10
JONES	20
JAMES	40

DEPT	
DEPTNO	DNAME
10	ACCOUNTING
20	SALES
30	RESEARCH
40	OPERATIONS

(e) Self Join: Joining a relation itself.

EMP		
EMPNO	ENAME	MGR
7839	KING	
7698	BLACK	7839
7782	CLARK	7839
7654	MARTIN	7698
7499	ALLEN	7698



DIVISION OPERATION

- **Used to execute special kinds of queries.**

Syntax: R ÷ S

- **The division operation can be expressed as follows.**

T1 ← ∏y (R)

T2 ← ∏y ((S×T1) – R)

T ← T1 – T2

R	A	B
	a1	b1
	a2	b1
	a3	b1
	a4	b1
	a1	b2
	a3	b2
	a2	b3
	a3	b3
	a4	b3
	a1	b4
	a2	b4
	a3	b4

S	A
	a1
	a2
	a3

Step 1: Let us assume that

X = {A}

Y = {B}

Z = {A,B}

Step 2:

T1 ← ∏y (R)

T1	B
	b1
	b2
	b3
	b4

Step 3:

T2 ← ∏y ((S×T1) – R)

S × T1	A	B
	a1	b1 ×
	a1	b2 ×
	a1	b3
	a1	b4 ×
	a2	b1 ×
	a2	b2
	a2	b3 ×
	a2	b4 ×
	a3	b1 ×
	a3	b2 ×
	a3	b3 ×
	a3	b4 ×

R	A	B
	a1	b1 ×
	a2	b1 ×
	a3	b1 ×
	a4	b1
	a1	b2 ×
	a3	b2 ×
	a2	b3
	a3	b3 ×
	a4	b3
	a1	b4 ×
	a2	b4 ×
	a3	b4 ×

S × T1	A	B
	a1	b3
	a2	b2

$\prod_Y$ (S × T1 – R)	B
	b3
	b2

T2	B
	b3
	b2

Step 4:

T ← T1 – T2

T1	B
	b1
	b2 ×
	b3 ×
	b4

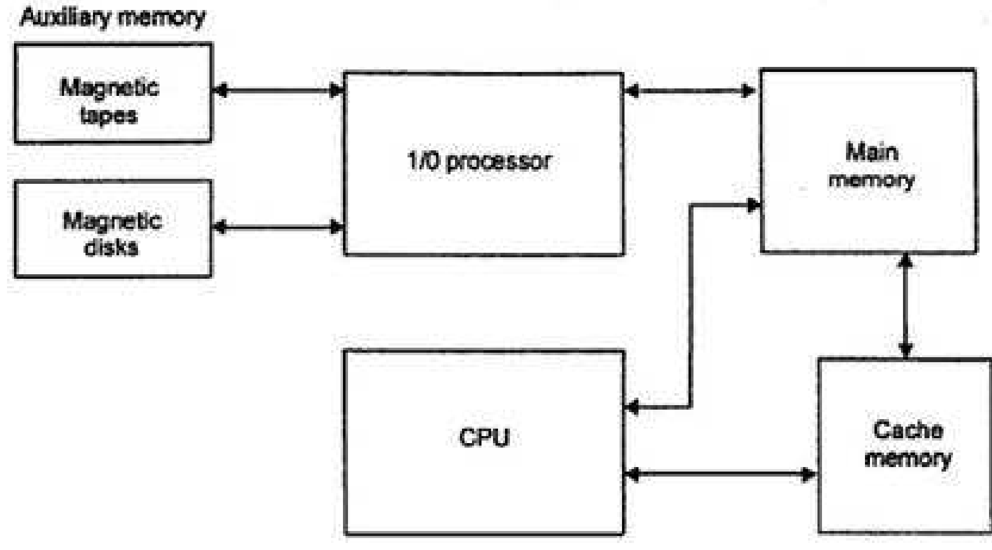
T2	B
	b3
	b2

T	B
	b1
	b4

Topic 4: Memory Hierarchies

Priority	Device
1	High speed registers
2	Cache (or) Static RAM
3	Main memory (or) DRAM
4	Optical disk (or) CDROM
5	Magnetic disk
6	Magnetic tap

\

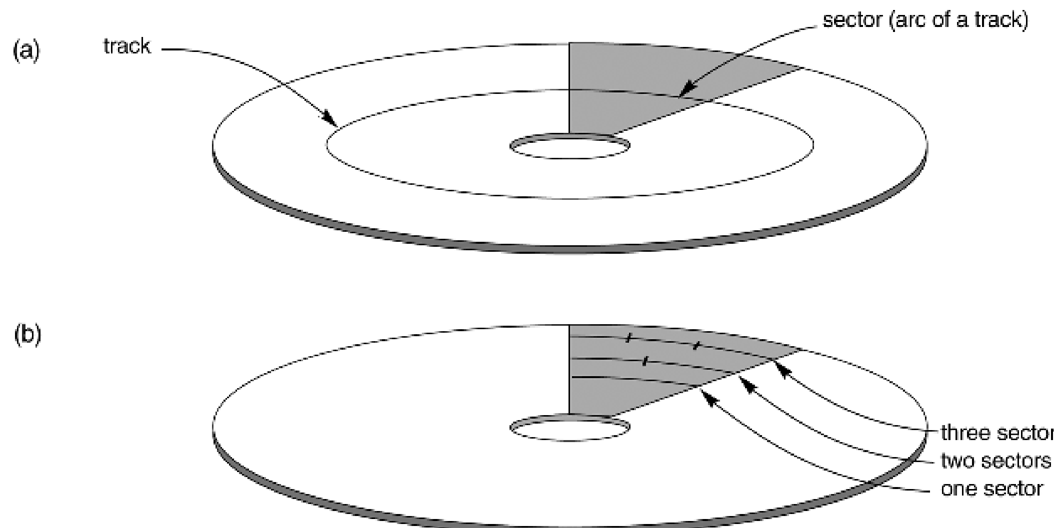


Topic 5:Secondary Storage Devices

1. Magnetic Disk

- **Magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material**
- **Both sides of the disk are used**
- **Several disks may be stacked on one spindle with Read/Write heads available on each surface**
- **All disks rotate together at high speed and not stopped or started for access purpose**
- **Used to store large amount of data .**
- **Basic unit of storage is BIT .**
- **BIT is 1 by magnetizing area on disk .**

- **Character: Group of bits.**
- **Capacity: Number of bytes it can store.**
- **Number of tracks: ranges from few hundred to few thousand.**
- **Capacity of each track: ranges from tens of Kbytes to 150 Kbytes.**
- **Formatting: Tracks are divided into equal sized disk blocks called sectors.**
- **Sector size is fixed range from 512 to 4096 bytes.**



- **Read/Write head is classified into**
  1. **Fixed head system.**
  2. **Moving head system.**
- **Rotate disk pack of cylinders ranging from 3600 and 7200 rpm.**
- **Access time: Time required to access the data.**

t_A **= Access time**

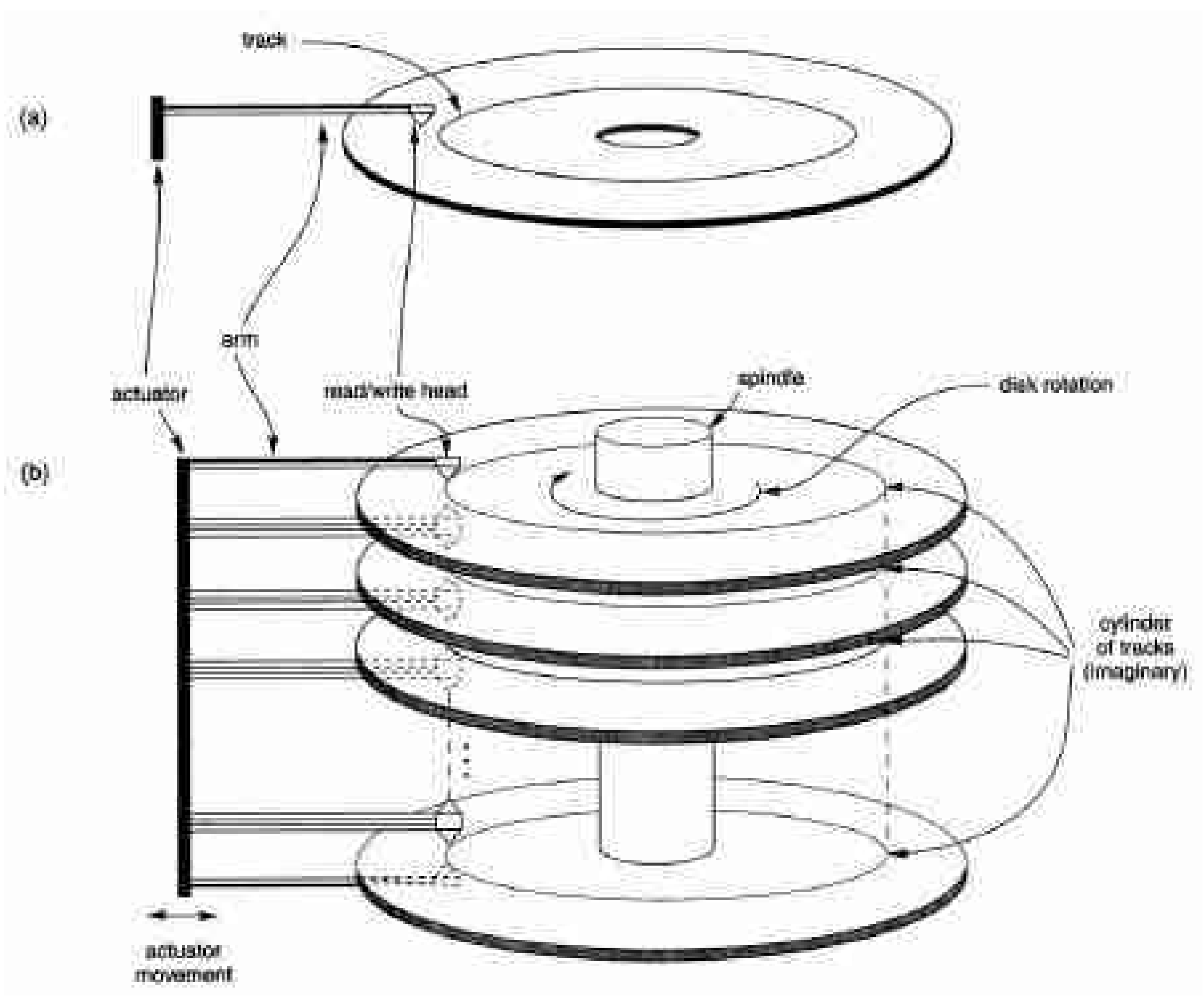
$$t_A = t_S + t_L$$

- **Seek time: Time required to position the read/write head on the correct track & correct surface.**

t_S **= Seek time**

- **Latency time: Time required to locate the required disk block from the beginning.**

t_L = Latency time



2. Floppy Disk

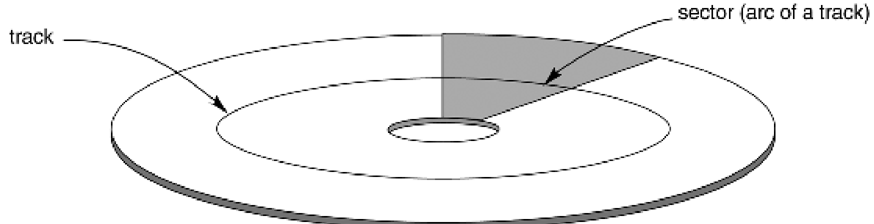
- **A disk drive with removable disks are called a Floppy Disk**
- **The disks used with a Floppy disk drive are small disks made of plastic coated with magnetic recording material**
- **There are two sizes commonly used, with diameters of 5.25 and 3.5 inches**
- **The 3.5 inch disks are smaller and can store more data than 5.25 inch disks**
- **Floppy disks extensively used in personal computer as a medium for distributing software to computer users**
- **Basic unit of storage is BIT .**

- BIT is 1 by magnetizing area on disk .

- Character: Group of bits.
- Capacity: Number of bytes it can store.
- Density : Single density and Double density
- Data stored in: Tracks & Sectors
- Track: Data is placed in concentric magnetic circles called tracks
- Sector :Each track is divided into storage units called sectors
  - FAT uses 512 byte sectors exclusively
  - The number of Sectors per Track vary depending on the media and format
  - Physical Sector Numbering starts with the number 1at the beginning of each track and side

- Number of Tracks: 80 in 3.5 inch floppy disk

Disk Formats - 3.5”		
	Low Density	High Density
Tracks	80	80
Sectors per track	9	18
Bytes per sector	512	512
Total capacity	720K	1.44M



- Formatting: Tracks are divided into equal sized disk blocks called sectors.

- Access time: Time required to access the data.

t_A = Access time

$t_A = t_S + t_L$

- Seek time: Time required to position the read/write head on the correct track & correct surface.

  t_S = Seek time

- Latency time: Time required to locate the required disk block from the beginning.

  t_L = Latency time

3. Magnetic Tape

- Magnetic tape transport consists of the electrical, mechanical and electronic components provide the parts and control mechanism for magnetic tape unit
- The tape is a strip of plastic coated with magnetic recording media
- Bits are recorded as magnetic spots on the tape along with several tracks
- Usually, seven or nine bits are recorded simultaneously to form a character together with a parity bit.
- The information is recorded in blocks referred to as records
- Each record on tape has an identification bit pattern at the beginning and end. By recording the bit pattern at the beginning, the tape control identifies the record number.
- By reading the bit pattern at the end of the record, the control recognizes the beginning of a gap
- A tape unit is addressed by specifying the record number and number of characters in the record
- Record may be fixed length or variable length
- Sequential access.
- Read/Write head: Reads or writes data on tape.
- Storage: 1600 to 6250 bytes per inch.
- Inter block gap: 0.6 inches.

Block		Block		Block		Block		Block		Block

Transaction Processing

- Transaction : Logical unit of sequence of processing.
- Transaction operations: Insert, Delete, Update & Retrieve.
- Transaction boundaries: Begin, End.
- Begin transaction: Transaction begins when user enters into SQL plus environment.
- End transaction: Transaction ends when one of the following statements are executed.

(a) Exit
(b) Commit
(c) Rollback

Topic 1: Transaction read & write

read_item(x)

- Reads a database item into a program variable.

1. Find the address of the disk block that contain data item x.
2. Copy the disk block contain the data item x into primary memory.
3. Copy data item x to program variable named x.

write_item(x)

- Writes the value of program variable x into database.

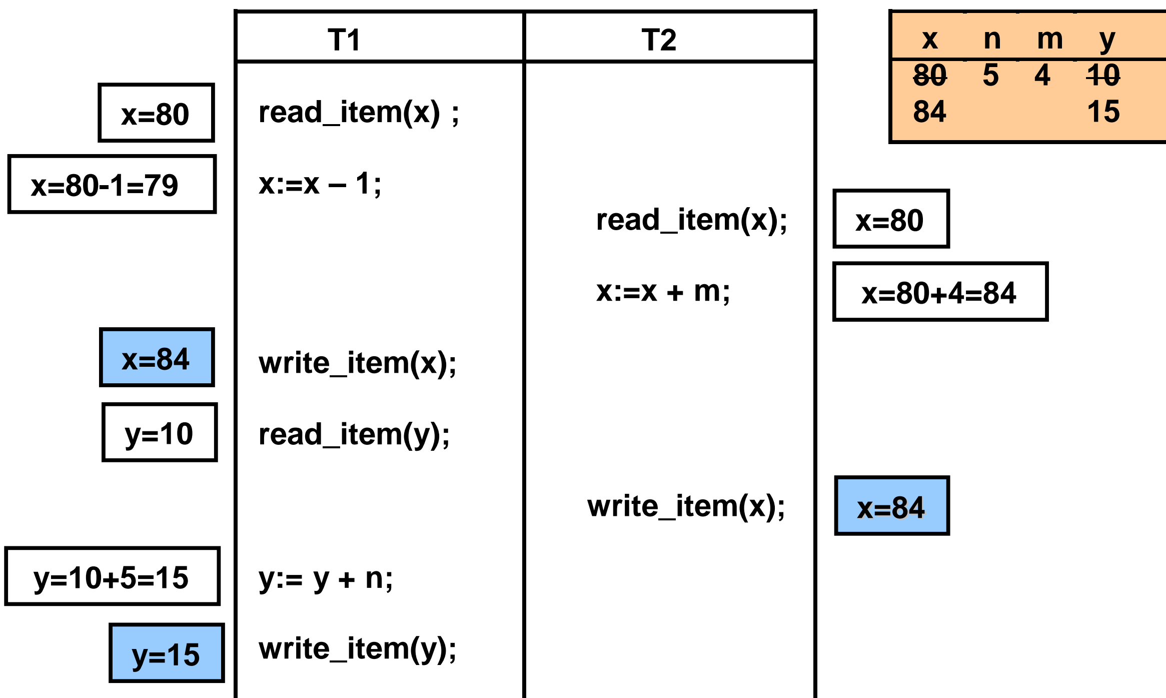
1. Find the address of the disk block that contain data item x.
2. Copy the disk block contain the data item x into primary memory.
3. Copy data item x to program variable named x.
4. Store the updated data item contained in disk block to secondary memory.

Topic2: What are the problems in transaction processing (or) What are the problems due to concurrency (or) Why concurrency control is needed [VIMP]

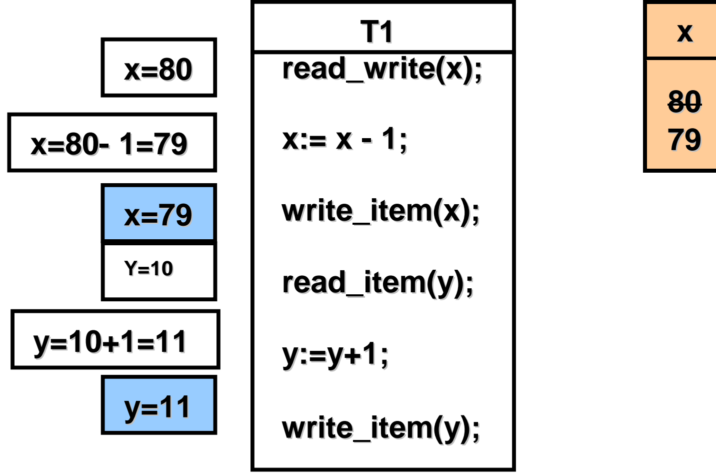
- Concurrency : When multiple transactions request single database item simultaneously, the updations of transactions produce inconsistent data.

The lost update problem

- The updations of interleaving transactions produce inconsistent data.



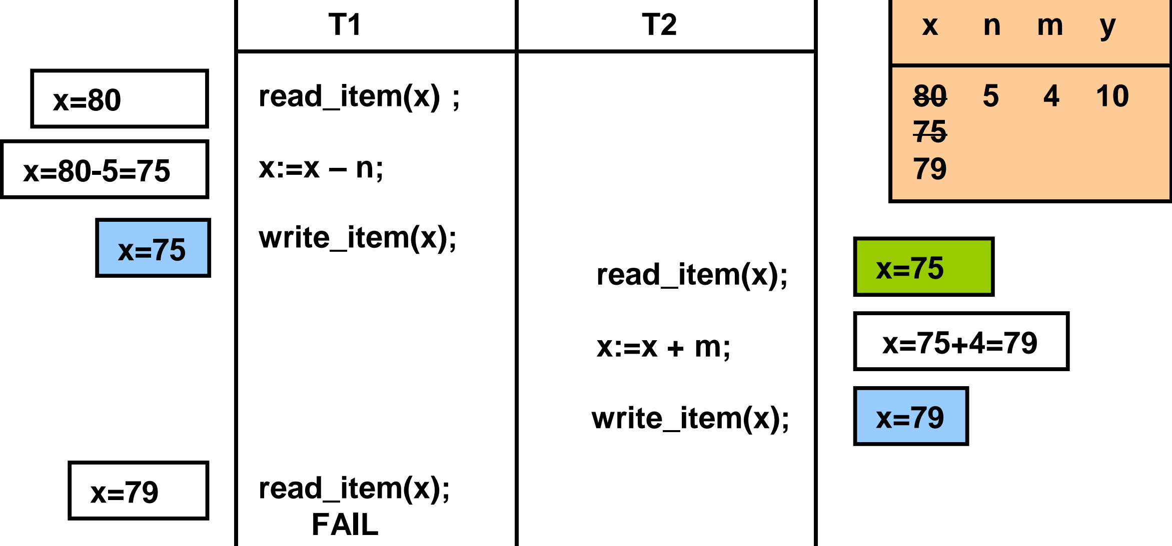
- Serial operations of transaction T1 is



- The final result of serial transaction is x=79 but the interleaving transaction operation shows that x=84.

The temporary update problem (Dirty read)

- The problem occurs when one transaction update the database item and then fails



- If transaction T1 fails the database item roll backs to its previous state, but transaction T2 has read the incorrect value of x.

The incorrect summary problem

- A transaction T receives different values for two reads on the same data item.

y	sum	x	n	a
~~20~~ 25	0	~~10~~ 5	5	5

	T1	T2	
		sum:=0;	
		read_item(a);	a =5
		sum:=sum +a;	sum=0+5=5
		. . .	
x=10	read_item(x);		
x=10-5=5	x:=x - n;		
x =5	write_item(x);		
		read_item(x);	x = 5
		sum:=sum+x;	sum=0+5=5
		read_item(y);	y=20
		sum:=sum+y;	sum=0+20=20
y=20	read_item(y);		
y=20 +5=25	y:=y+n;		
y=25	write_item(y);		

Topic3: Types of transaction failures (or) Why recovery is needed [IMP]

- **Types of failures**

(a) Physical (or) Catastrophic
(b) Disk failure
(c) Transaction failure

Physical (or) Catastrophic failure

1. A/C failure.
2. Theft.
3. Sabotage.
4. Mounting wrong tape.
5. Over riding disks (or) taps.
6. Giving same name to the file.

Disk failure

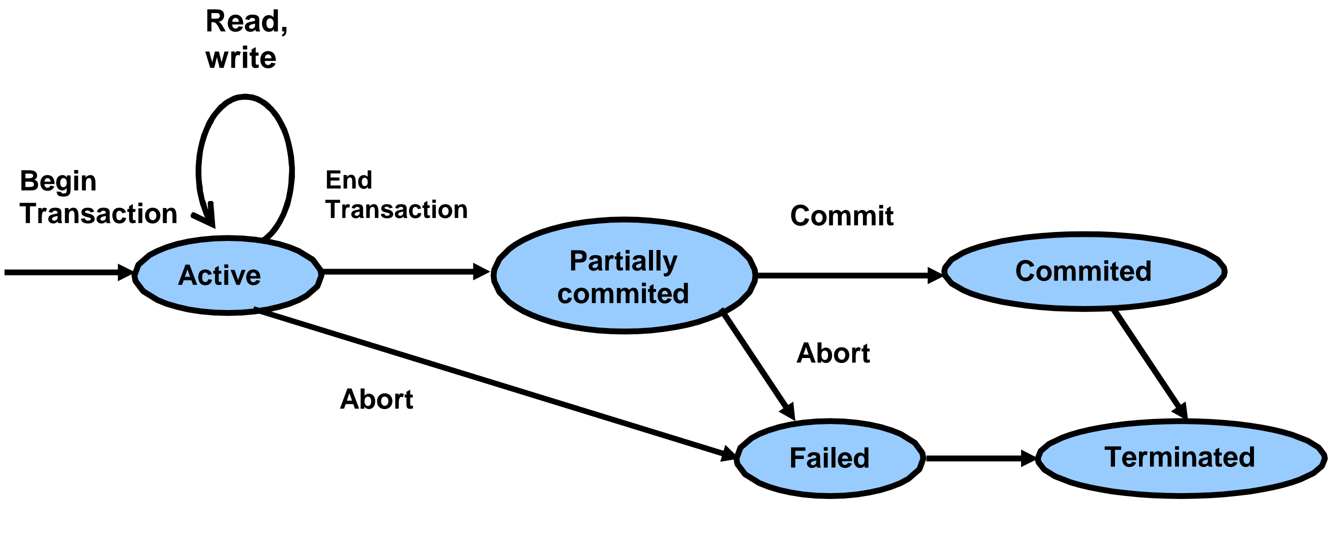
Read / write head malfunctioning.

Transaction failure

1. Operations of transaction may fail due to logical errors. E.g. Division by 0. Integer over flow. Error in parameters. Error in control structures.
2. Transactions may fail due to concurrency.
3. Transaction may fail due to S/W, H/W errors during the execution of transaction.
4. Transaction may fail if the data for the transaction *may not be found* or *insufficient.*

Topic4: Transaction states

Begin_transaction	This makes begin the transaction.
Read / Write	Specifies either *read operation* or *write operation*.
End_transaction	Specifies all the read & write operations are commited (or) Specifies all the read & write operations are partially commited and aborted.
Commit	Indicates successful updation of transaction.
Rollback / Abort	Indicates that the transaction ended unsuccessfully.



Topic5: The system log [IMP]

- **System log: System log is file keep track of all transaction operations on database items.**

- **Advantages**

1. The log file entries used for recovery purpose.
2. The log file is periodically backed up into magnetic tape (Archival storage)
3. Log record: Entry of log file.

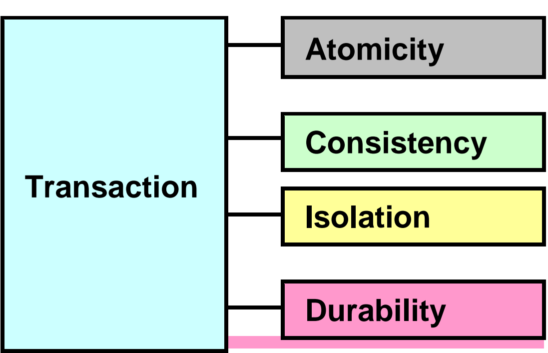
- **Entries of log file**

[start_transaction,T]	Indicates that transaction T has started execution
[write_item,T,X,old_value,new_value]	Indicates that transaction T has changed the value of database item X from old value to new value.
[read_item,T,X]	Indicates the transaction T has read the value of database item X.
[commit,T]	Indicates that transaction T has completed successfully and effect of transaction can be recorded permanently in the database.
[abort,T]	Indicates that transaction T has been aborted.

Topic 6: Desirable properties of transaction (or) ACID properties of transaction [IMP]

Atomicity

- The transaction is atomic unit of processing i.e. either performed entirely or none performed at all.



Consistency

- A correct execution of transaction must take the database item from one consistent state to another consistent state.

Isolation

- A transaction is isolated from other transactions when multiple transactions are accessing the same data item concurrently in a *serial* or *interleaving* fashion.

Durability

- The changes applied to the database item by a commited transaction must persistent in the database.

Topic 7: Schedules & Recoverability

- Schedule (or) History: The order of execution of operations of *serial* or *interleaving* transactions.

T1	T2
read_item(x) x:=x-n;	
	read_item(x); x:=x+n;

write_item(x);
read_item(y);
y:=y+n;	

	write_item(x);
write_item(y);	

S_A: r1(x); r2(x); w1(x); r1(y); w2(x); w1(y);

T1	T2

read_item(x)
x:=x-n;
write_item(x);	

	read_item(x);
	x:=x+m;
	write_item(x);

read_item(y); Fails	

S_B: $r_1(x)$; $w_1(x)$; $r_2(x)$; $w_2(x)$; $r_2(y)$; a1;

Topic8: Conflict operations

- Two operations in a schedule are said to be conflict, if they satisfy all three of the following conditions.

1. They belongs to different transactions.
2. They access the same data item.
3. At least one of the operation is a write operation.

- Consider schedule S_A

  S_A: $r_1(x)$; $r_2(x)$; $w_1(x)$; $r_1(y)$; $w_2(x)$; $w_1(y)$;

- Consider two operations from schedule

  $r_1(x)$ & $w_2(x)$;

  The above two operations are conflict since

The operations belongs to different transactions T1 &T2.
They access the same data item x.
One of the operation is write operation.

- Verify which of the operations are conflict

$r_2(x)$ & $w_1(x)$;	Conflict
$w_1(x)$ & $w_2(x)$;	Conflict
$r_1(x)$ & $r_2(x)$;	Not conflict
$w_2(x)$ & $w_1(y)$;	Not conflict

Topic 8: Complete schedule

- A sehedule 's' of 'n' transactions $T_1, T_{2,...,}T_{n,}$ is said to be a complete schedule if the following conditions hold.

1.The operations in 's' are exactly those operations in $T_1, T_{2,...,}T_{n,}$ including a commit or abort operation as the last operation for each transaction in the schedule.
2. For any pair of operations from the same transaction $T_{i,}$ their order of appearance is 's' is the same as their order of appearance in T_i
3. For any two conflicting operations, one of the two must occur before the order in the schedule.

Topic 9: Serial schedule

- Serial schedule: The order of execution of operations of serial transactions.

T1	T2

read_item(x) ;
x:=x – n;
write_item(x);
read_item(y);
y:=y + n;
write_item(y);	

	read_item(x);
	x:=x + m;
	write_item(x);

S_A: $r_1(x)$; $w_1(x)$; $r_1(y)$; $w_1(y)$; $r_2(x)$; $w_2(x)$;

T1	T2

	read_item(x) ;
	x:=x +m;
	write_item(x);

read_item(x);
x:=x - n;
write_item(x);
read_item(y);
y:=y + n;
write_item(y);	

S_B: $r_2(x)$; $w_2(x)$; $r_1(x)$; $w_1(x)$; $r_1(y)$; $w_1(y)$;

Topic 10: Non serial schedule

- **Non serial schedule: The order of execution of operations of interleaving transactions.**

T1	T2
read_item(x) ;	
x:=x + n;	
	read_item(x);
	x:= x + m;
write_item(x);	
read_item(y);	
	write_item(x);
x:=y + n;	
write_item(y);	

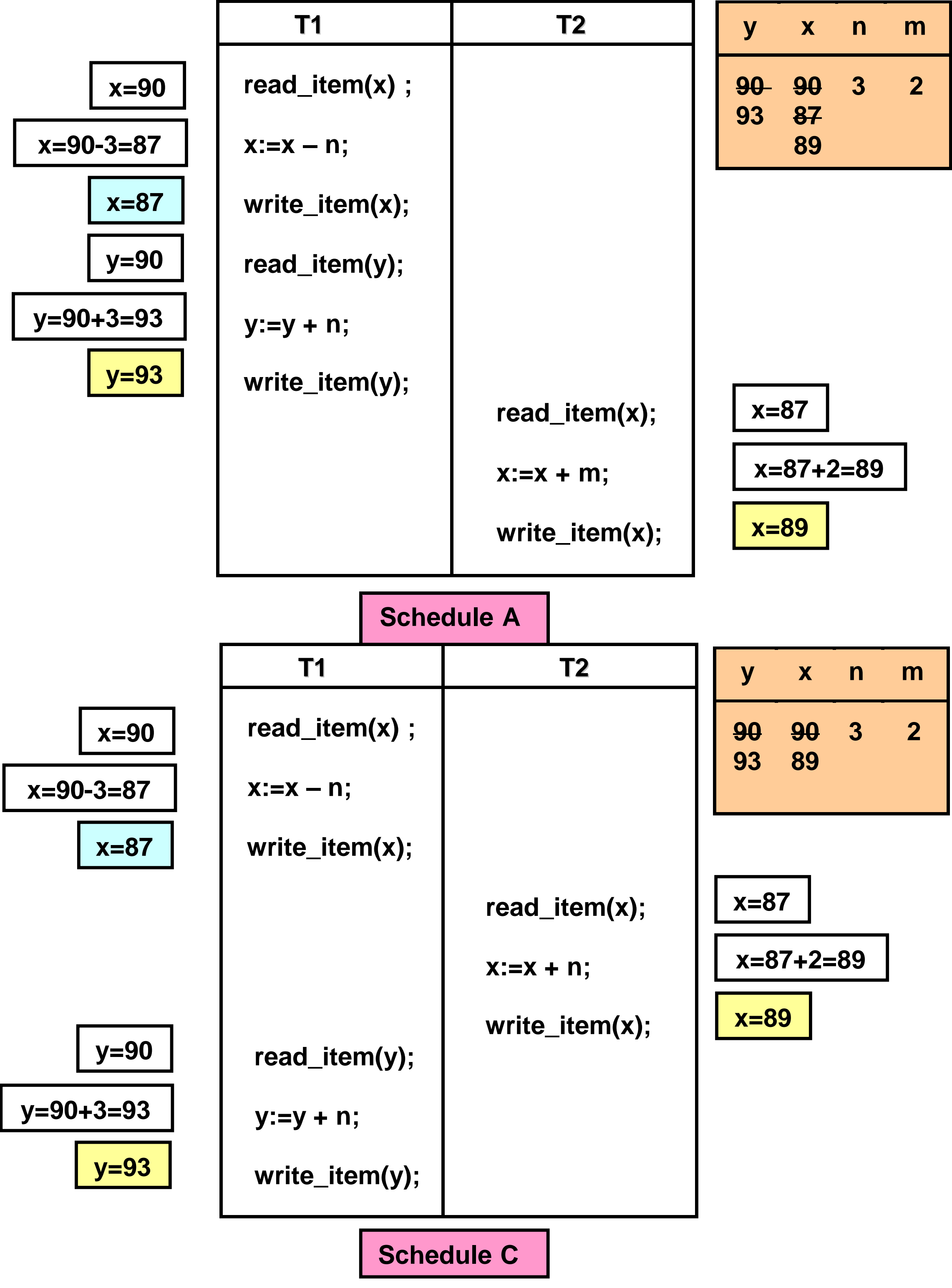
S_c: $r_1(x)$; $r_2(x)$; $w_1(x)$; $r_1(y)$; $w_2(x)$; $w_1(y)$;

T1	T2
read_item(x) ;	
x:=x - n;	
write_item(x);	
	read_item(x);
	x:=x + m;
	write_item(x);
read_item(y);	
y:=y + n;	
write_item(y);	

S_D: $r_1(x)$; $w_1(x)$; $r_2(x)$; $w_2(x)$; $r_1(y)$; $w_1(y)$;

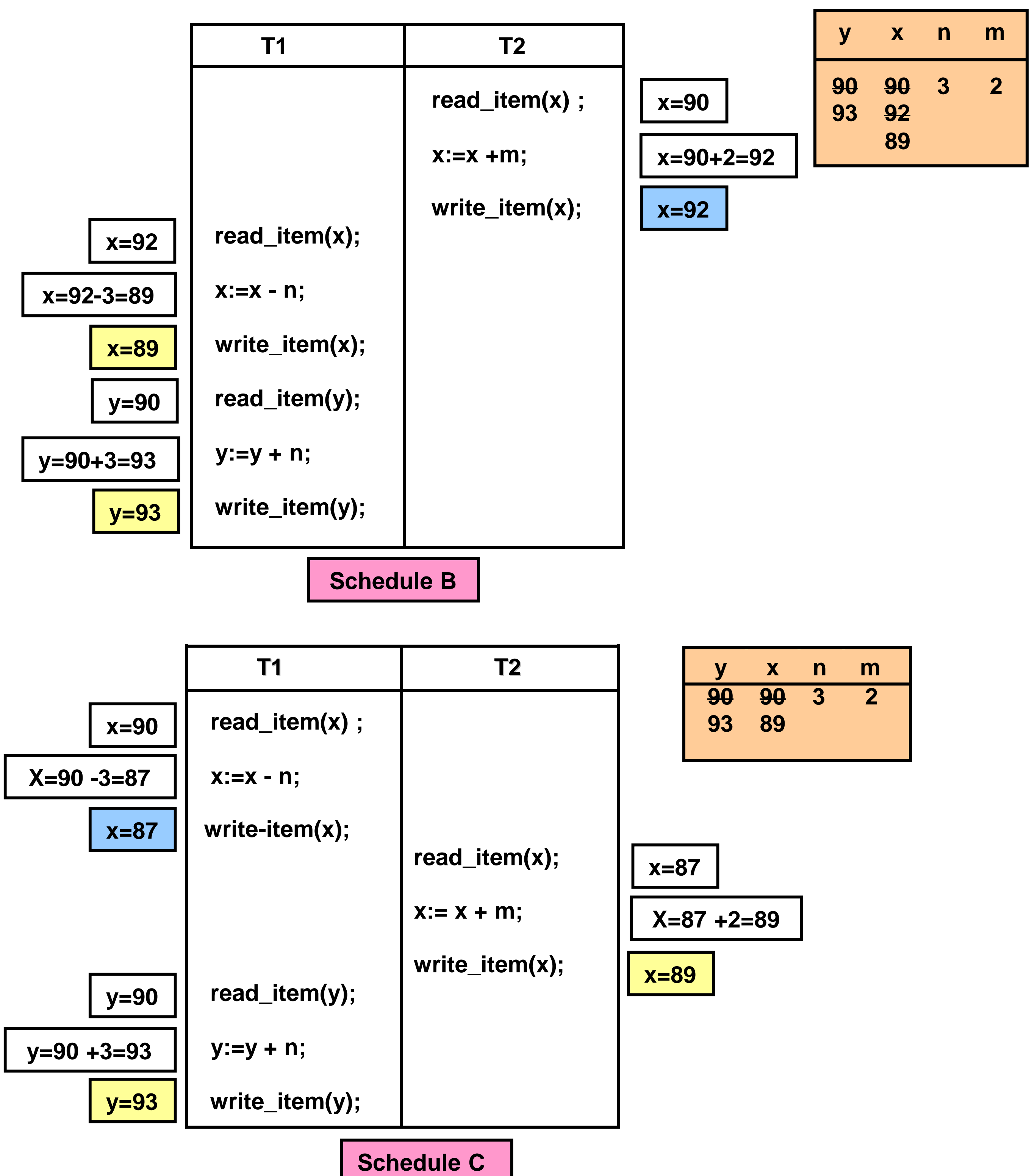
Topic 11: Result equivalent schedule

- **Two schedules are result equivalent if they produce the same final states of the database item.**



Topic 12 : Serializable schedule	VIMP

- **Serializability: A schedule 's' of 'n' transactions is serializable if it equal to some serial schedule of same 'n' transactions.**



- A non serial schedule , *schedule D* is serializable to serial schedule *schedule B ,* since they are equivalent.

Topic14: Conflict Equivalent Schedule [IMP]

- Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both the schedules.

- Two operations in a schedule are said to be conflict, if they satisfy all three of the following conditions.

1. They belongs to different transactions.
2. They access the same data item.
3. At least one of the operation is a write operation.

E.g1: S_1: $r_1(x)$; $w_2(x)$
S_2: $w_2(x)$; $r_1(x)$

Both S_1 and S_2 contains conflict operations but they are NOT CONFLICT equivalent.

E.g2: S_1: $w_1(x)$; $w_2(x)$
S_2: $w_2(x)$; $w_1(x)$

Both S_1 and S_2 contains conflict operations but they are NOT CONFLICT equivalent.

E.g3: S_A: $w_1(x)$; $r_2(x)$
S_B: $w_1(x)$; $r_2(x)$

Both S_A and S_B contains conflict operations but they are CONFLICT equivalent.

Topic 15: Uses Of Serializability

1. A Serializable schedule gives the benefits of concurrent execution
2. The interleaving of operations from concurrent transactions is typically determined by OS SCHEDULER which allocates resources to all the processes
3. Enforces TWO PHASE LOCKING PROTOCOL (Concurrency control protocol)
4. Ensures TIME STAMP ORDERING PROTOL (Concurrency control protocol)
5. Ensures MULTIVERSION PROTOCOL (Maintains multiple versions of database item)
6. Ensures OPTIMISTIC (or) CERTIFICATION (or) VALIDATION protocol (Concurrency control protocols)

Topic16: Conflict Serializable Schedule [IMP]

Conflict Serializable: A schedule S is conflict serializable if it is conflict equivalent to some serial schedule S^I

- **Consider the schedules S_A and S_D**

  S_A: $r_1(x)$; $\underline{w_1(x)}$; $r_1(y)$; $w_1(y)$; $\underline{r_2(x)}$; $w_2(x)$;

  S_D: $r_1(x)$; $\underline{w_1(x); \ r_2(x)}$; $w_2(x)$; $r_1(y)$; $w_1(y)$;

- **Consider**

$w_1(x)$;$r_2(x)$ in S_A
$w_1(x)$;$r_2(x)$ in S_D

1. **In both the schedules Transaction 2 reads the values after Transaction 1 writes**
2. **The operations $w_1(x)$;$r_2(x)$ is conflicting operations**
3. **Last operations are write operation in both the schedules**

- **Therefore S_D is conflict serializable to S_A**

Topic 17: View Equivalence [IMP]

Consider two schedules S and S_g

S: $r_1(x)$; $w_1(x)$; $r_1(y)$; $w_2(x)$; $w_3(x)$;

S_g: $r_1(x)$; $w_2(x)$; $w_1(x)$; $w_3(x)$; c_1; c_2; c_3;

- **Two schedules S and S_g are said to be view equivalent**

1. **Bothe schedules have same transactions and same operations**

S	S_g
T1,T2,T3	T1,T2,T3
read_item(x) write_item(x)	read_item(x) write_item(x)

2. The first operation of first schedule should be same as the first operation of second schedule

3. The last operation of first schedule should be same as the last operation of second schedule

4. Both the schedules enforce conflict equivalence

Topic 18: View Serializability [IMP]

- A Schedule S_g is view serializable if it is view equivalent to some serial schedule S

S: $r_1(x)$; $w_1(x)$; $r_1(y)$; $w_2(x)$; $w_3(x)$;

S_g: $r_1(x)$; $w_2(x)$; $w_1(x)$; $w_3(x)$; c_1; c_2; c_3;

Non serial schedule S_g is view serializable to serial schedule S

- CONSTRAIND WRITE ASSUMPTION :

Any write operation $w_i(x)$ in T_i is preceded by a read operation $r_i(x)$.

S_g: $r_1(x)$; $w_2(x)$; $w_1(x)$; $w_3(x)$; c_1; c_2; c_3;

Note : Read value for data item before writing it.

Note : $w_1(x)$ preceded by the $r_1(x)$

Note: In the above schedule $w_1(x)$ is preceded by $r_1(x)$

- UN CONSTRAIND WRITE ASSUMPTION (OR) BLIND WRITES

The read operation $r_i(x)$ is not preceded by write operation $w_i(x)$

Note: Data item value can't be read before writing it

Note: $w_2(x)$ and $w_3(x)$ is not preceded by the read operations $r_2(x)$ and $r_3(x)$

Topic 19: Transaction Support In SQL	IMP

- Transaction: Sequence of statements that performs specified task.
- A single SQL statement always considered to be atomic.

- Characteristics :

(a) Access Mode

Access Mode specified as READ ONLY (or) READ WRITE

READ ONLY: Read the data from database (SELECT)
READ WRITE: INSERT, UPDATE, DELETE

(b) Diagnostics Area Size

DIAGNOSTIC SIZE N

The integer value N indicating the number of error conditions or exceptions.

These conditions supply feedback information to the users on the most recently executed SQL statements.

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;

EXEC SQL SET TRANSACTION
        READ WRITE
        DIAGNOSTICS SIZE 5
        ISOLATION LEVEL SERIALIZABLE;

EXEC SQL INSERT INTO EMPLOYEE(Fname,Lname,SSN,Dno,Salary)
VALUES('Robert','Smith','991004321',2,35000);

EXEC SQL UPDATE EMPLOYEE
    SET Salary = Salary * 1.1 WHERE Dno = 2;

EXEC SQL COMMIT;

UNDO: EXEC SQL ROLLBACK;
THE_END: ...;
```

(c) ISOLATION LEVEL

Isolation level specified with option

ISOLATION LEVEL < ISOLATION>

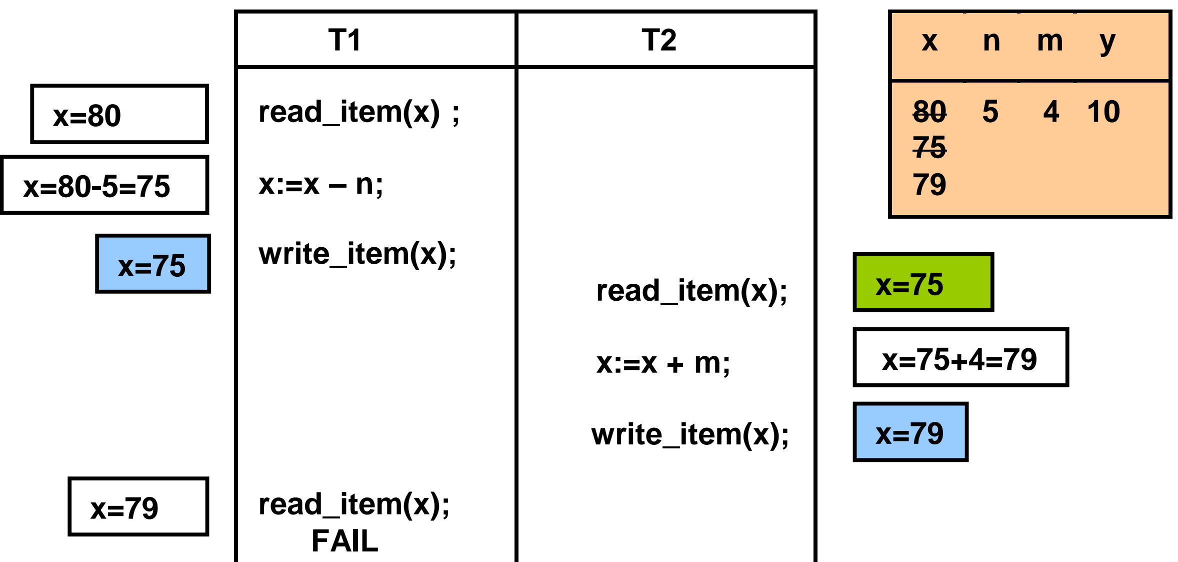
ISOLATION:

1. READ UNCOMMITED
2. READ COMMITED
3. REPEATABLE READ
4. SERIALIZABLE

There may be three of the following violations in SQL TRANSACTIONS

1. Dirty read :

- The problem occurs when one transaction update the database item and then fails



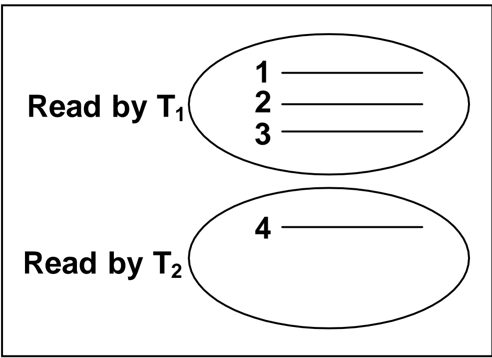
- If transaction T1 fails the database item roll backs to its previous state, but transaction T2 has read the incorrect value of x.

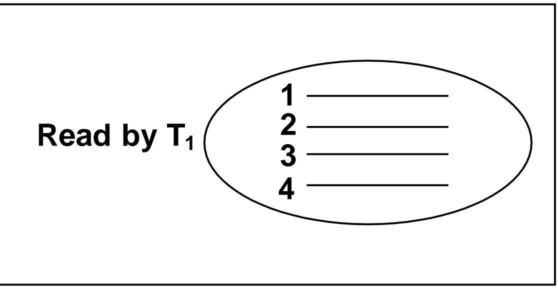
2. Nonrepeatable read

A transaction T_1 may read a given value from a table. If another transaction T_2 later updates that value and T_1 reads that value again, T_1 will see a different values.

3. Phontoms

1. A transaction T_1 may read the rows from table based on the where condition.
2. A transaction T_2 inserts a new record that satisfy where condition used in T_1.
3. If T_1 is repeated, then T_1 will see a PHONTOM, a row that previously did not exist.

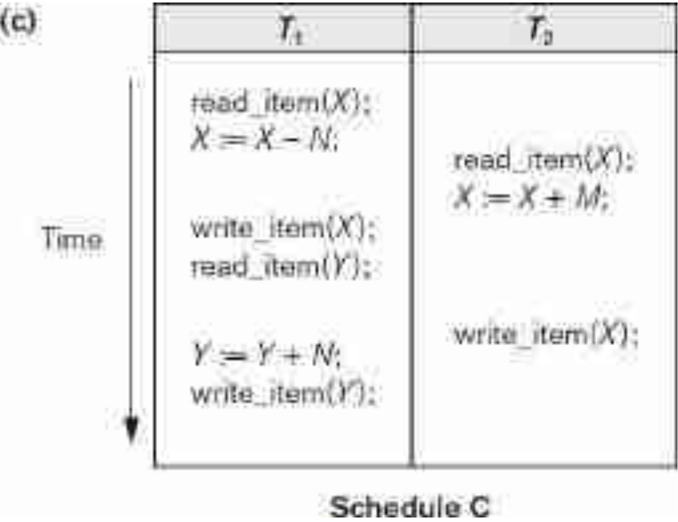




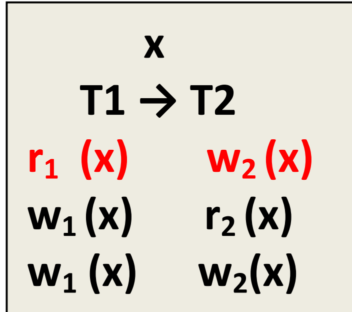
Possible violations Based on isolation levels as defined in SQL			
	TYPE OF VIOLATION		
Isolation Level	Dirty Read	Non repeatable Read	Phantom
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

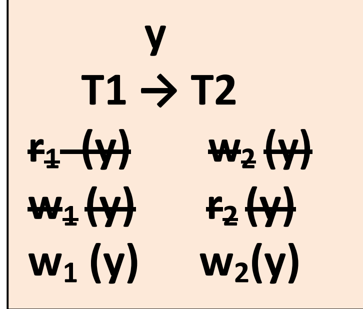
Testing Serializability (or) Confilct Serializability

Problem 1: Verify whether the following Transactions ensures serializablity.



S_c: $r_1(x)$; $r_2(x)$; $w_1(x)$; $r_1(y)$; $w_2(x)$; $w_1(y)$;

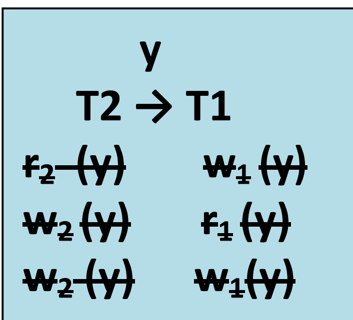


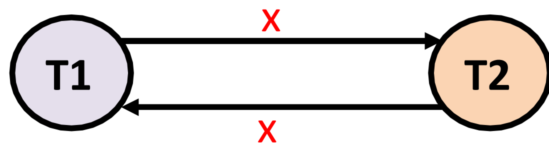


x

T2 → T1

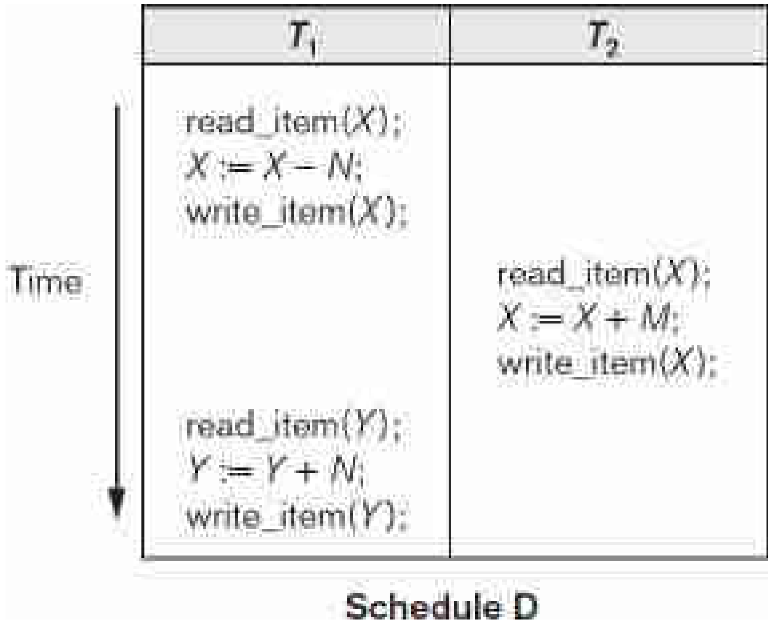
r_2 (x)	w_1 (x)
w_2 (x)	r_1 (x)
w_2 (x)	w_1(x)





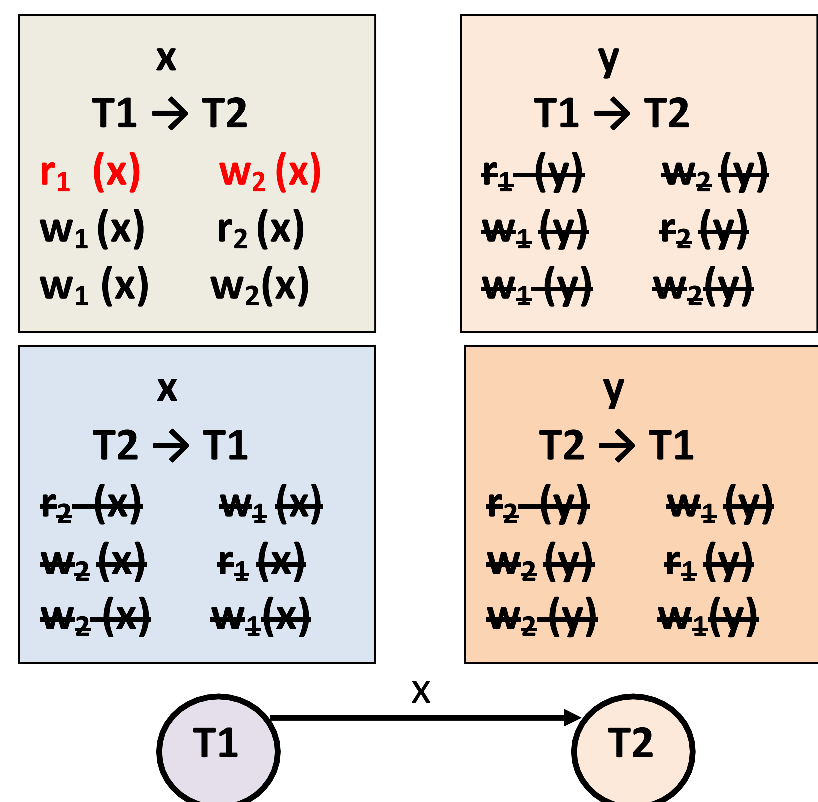
- Since the Precedence Graph has Cycle, the schedule Sc is not *Serializable* or *Conflict Serializable.*

Problem 2: Verify whether the following Transactions ensures serializablity.



Schedule D

S_D: $r_1(x)$; $w_1(x)$; $r_2(x)$; $w_2(x)$; $r_1(y)$; $w_1(y)$;



- Since the Precedence Graph has No Cycle, the schedule S_D is ***Serializable*** or ***Conflict Serializable.***

Problem 3: Verify whether the following schedule enforces conflict serializability.

Time ↓	Transaction T_1	Transaction T_2	Transaction T_3
		read_item(Z);	
		read_item(Y);	
		write_item(Y);	
			read_item(Y);
			read_item(Z);
	read_item(X);		
	write_item(X);		
			write_item(Y);
			write_item(Z);
		read_item(X);	
	read_item(Y);		
	write_item(Y);		
		write_item(X);	

Schedule E

S_E: $r_2(z)$; $r_2(y)$; $w_2(y)$; $r_3(y)$; $r_3(z)$; $r_1(x)$; $w_1(x)$; $w_3(y)$; $w_3(z)$; $r_2(x)$; $r_1(y)$; $w_1(y)$; $w_2(x)$;

x	
T1 →	T2
r_1 (x)	w_2 (x)
w_1 (x)	r_2 (x)
w_1 (x)	w_2(x)

y	
T1 →	T2
~~r_1 (y)~~	~~w_2 (y)~~
~~w_1 (y)~~	~~r_2 (y)~~
~~w_1 (y)~~	~~w_2(y)~~

z	
T1 →	T2
~~r_1 (z)~~	~~w_2 (z)~~
~~w_1 (z)~~	~~r_2 (z)~~
~~w_1 (z)~~	~~w_2(z)~~

x	
T1 →	T3
~~r_1 (x)~~	~~w_3 (x)~~
~~w_1 (x)~~	~~r_3 (x)~~
~~w_1 (x)~~	~~w_3(x)~~

y	
T1 →	T3
~~r_1 (y)~~	~~w_3 (y)~~
~~w_1 (y)~~	~~r_3 (y)~~
~~w_1 (y)~~	~~w_3(y)~~

z	
T1 →	T3
~~r_1 (z)~~	~~w_3 (z)~~
~~w_1 (z)~~	~~r_3 (z)~~
~~w_1 (z)~~	~~w_3(z)~~

x

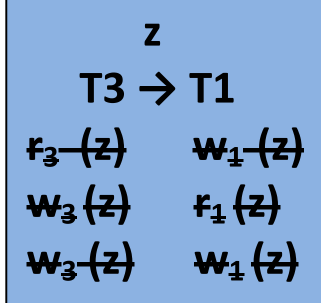
T3 → T1

~~r_3 (x)~~	~~w_1 (x)~~
~~w_3 (x)~~	~~r_1 (x)~~
~~w_3 (x)~~	~~w_1(x)~~

y

T3 → T1

r_3 (y)	w_1 (y)
w_3 (y)	r_1 (y)
w_3 (y)	w_1(y)



x

T3 → T2

~~r_3 (x)~~	~~w_2 (x)~~
~~w_3 (x)~~	~~r_2 (x)~~
~~w_3 (x)~~	~~w_2(x)~~

y

T3 → T2

~~r_3 (y)~~	~~w_2 (y)~~
~~w_3 (y)~~	~~r_2 (y)~~
~~w_3 (y)~~	~~w_2(y)~~

z

T3 → T2

~~r_3 (z)~~	~~w_2 (z)~~
~~w_3 (z)~~	~~r_2 (z)~~
~~w_3 (z)~~	~~w_2 (z)~~

x

T2 → T1

~~r_2 (x)~~	~~w_1 (x)~~
~~w_2 (x)~~	~~r_1 (x)~~
~~w_2 (x)~~	~~w_1(x)~~

y

T2 → T1

r_2 (y)	w_1 (y)
w_2 (y)	r_1 (y)
w_2 (y)	w_1(y)

z

T2 → T1

~~r_2 (z)~~	~~w_1 (z)~~
~~w_2 (z)~~	~~r_1 (z)~~
~~w_2 (z)~~	~~w_1 (z)~~

x

T2 → T3

~~r_2 (x)~~	~~w_3 (x)~~
~~w_2 (x)~~	~~r_3 (x)~~
~~w_2 (x)~~	~~w_3(x)~~

y

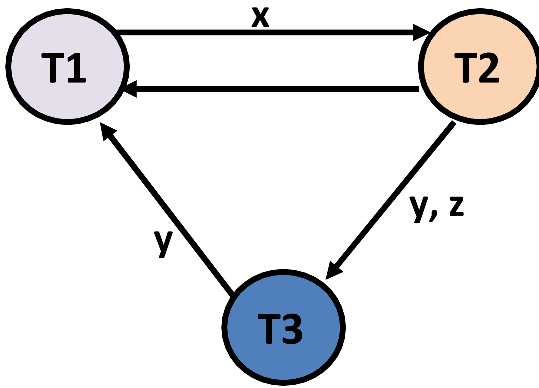
T2 → T3

r_2 (y)	w_3 (y)
w_2 (y)	r_3 (y)
w_2 (y)	w_3(y)

z

T2 → T3

r_2 (z)	w_3 (z)
w_2 (z)	r_3 (z)
w_2 (z)	w_3 (z)



- **Since the precedence has cycles , the schedule S_E is not conflict serializable or serializable.**

P/L SQL

1. TRIGGER VVIMP

- **A trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table.**
- **A trigger is triggered automatically when an associated DML statement is executed.**

Syntax:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Example:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

2. CURSORS VVIMP

- A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor.
- A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.
- There are two types of cursors
  1. Implicit cursors
  2. Explicit cursors

1. Implicit Cursors

- Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.
- Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.
- In PL/SQL, you can refer to the most recent implicit cursor as the SQL cursor, which always has attributes such as %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT. The SQL cursor has additional attributes, %BULK_ROWCOUNT and %BULK_EXCEPTIONS, designed for use with the FORALL statement.

```
DECLARE
   total_rows number(2);
BEGIN
   UPDATE customers
   SET salary = salary + 500;
   IF sql%notfound THEN
      dbms_output.put_line('no customers selected');
   ELSIF sql%found THEN
      total_rows := sql%rowcount;
      dbms_output.put_line( total_rows || ' customers
selected ');
   END IF;
END;
```

2. Explicit Cursors

- Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

```
DECLARE
   c_id customers.id%type;
   c_name customers.name%type;
   c_addr customers.address%type;
   CURSOR c_customers is
      SELECT id, name, address FROM customers;
BEGIN
   OPEN c_customers;
   LOOP
   FETCH c_customers into c_id, c_name, c_addr;
      EXIT WHEN c_customers%notfound;
      dbms_output.put_line(c_id || ' ' || c_name || ' ' ||
c_addr);
   END LOOP;
   CLOSE c_customers;
END;
/
```

3. VIEW VVIMP

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

----

4. RELATIONAL CONSTRAINTS VVIMP

- **Constraint: A rule at *column level* or *table level*.**
- **Relational constraints**

1. Not null (Defined at either column level or table level)
2. Unique (Defined at either column level or table level)
3. Primary key (Defined at either column level or table level)
4. Foreign key (Defined at either column level or table level)
6. Check (Defined at either column level or table level)

1. Not null: **Specifies that may not contain null value.**

E.g.

```
SQL> create table emp
        (empno number(4),
         ename varchar2(10) not null);
```

2. Unique: **Every value in a column or set of columns be unique.**

E.g.

```
SQL> create table dept
        (deptno number(2),
        dname varchar2(14),
         constraint dept_dname_uk unique(dname));
```

3. Primary key: Creates primary key for the table.

- **The value assigned for an attribute must satisfy the constraints *unique* and *not null*.**

E.g.

```
SQL> create table dept
        (deptno number(2),
        dname varchar2(14),
         constraint dept_dname_pk primary key(dname));
```

4. Foreign key: **The foreign key is defined in the child table, and the table containing the reference column is in the parent table.**

E.g.

```
SQL> create table emp
        (empno number(2),
         ename varchar2(14) not null,
         deptno number(2),
         constraint emp_dept_fk foreign key(deptno) references dept (deptno));
```

5. Check: Defines the condition that the each row must satisfy.

E.g.

```
SQL> create table dept
        (deptno number(2),
        dname varchar2(14),
         constraint emp_dept_ck check (deptno between 10 and 99));
```

LECTURE NOTES PREPARED BY

Dr.T.S.RAVI KIRAN, ASSISTANT PROFESSOR & HOD

DEPARTMENT OF COMPUTER SCIENECE

P.B.SIDDHARHA COLLEE OF ARTS & SCIENECE, VIJAYAWADA, AP, INDIA PIN: 520010

Email: tsravikiran@pbsiddhartha.ac.in

Mobile: 9441176980

COURESE: OPERATING SYSATEMS

CONTENTS:

1. **Introduction**
2. **OS Structure**
3. **Process**
4. **Threads**
5. **Synchronization**
6. **CPU Scheduling**
7. **Deadlocks**
8. **Memory Management**
9. **Virtual Memory**
10. **Mass Storage Structure**
11. **File Systems Interface**
12. **File System Implementation**

CHAPTER 1
INTRODUCTION

1. 1 WHAT OPERATING SYSTEMS DO (OR) FUNCTIONS OF OS

- **Operating system:** An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

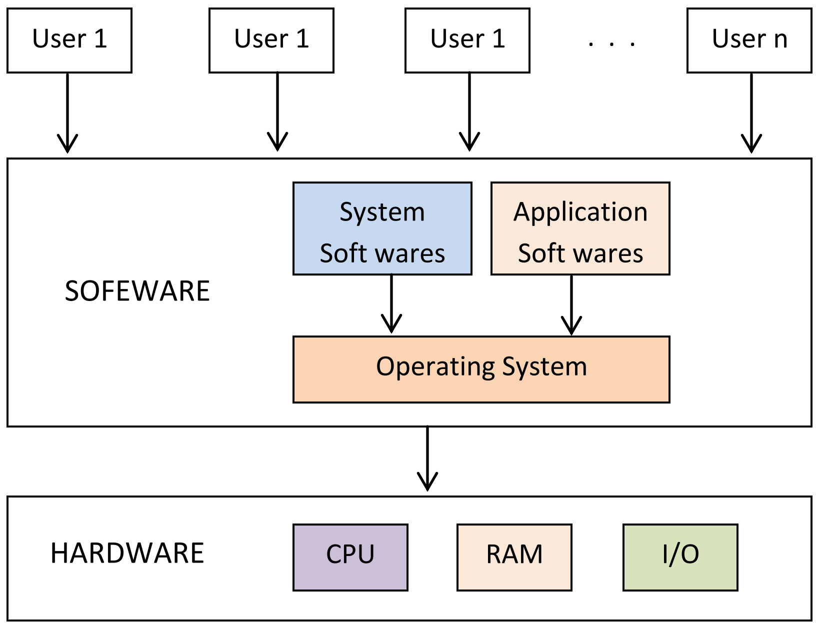


Fig 1.1: Abstract View of OS.

- Following are some of important functions of an Operating System.

1. Memory Management.
2. Processor Management.
3. Device Management.
4. File Management.
5. Security.
6. Control over system performance.
7. Job accounting.
8. Error detecting aids.
9. Coordination between other software and users.

Memory Management:

- Memory management refers to management of primary memory. An operating system performs the following activities for memory management.

1. Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.
2. In multiprogramming, the operating system decides which process will get memory when and how much.
3. Allocates the memory when a process requests it to do so.
4. De-allocates the memory when a process no longer needs it or has been terminated.

Processor Management:

- In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called **process scheduling**. An Operating System does the following activities for processor management.

1. Keeps tracks of processor and status of process. The program responsible for this task is known as traffic controller.
2. Allocates the processor (CPU) to a process.
3. De-allocates processor when a process is no longer required.

Device Management:

- An Operating System manages device communication via their respective drivers. It does the following activities for device management.

1. Keeps tracks of all devices. Program responsible for this task is known as the **I/O controller**.
2. Decides which process gets the device when and for how much time.
3. Allocates the device in the efficient way.
4. De-allocates devices.

File Management:

- A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.
- An Operating System does the following activities for file management

1. Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.
2. Decides who gets the resources.
3. Allocates the resources.
4. De-allocates the resources.

Other Important Activities:

- Following are some of the important activities that an Operating System performs −

1. **Security**: By means of password and similar other techniques, it prevents unauthorized access to programs and data.
2. **Control over system performance:** Recording delays between request for a service and response from the system.

3. **Job accounting:** Keeping track of time and resources used by various jobs and users.
4. **Error detecting aids:** Production of dumps, traces, error messages, and other debugging and error detecting aids.
5. **Coordination between other softwares and users:** Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

1.2 COMPUTER-SYSTEM ORGANIZATION

- **Computer Organization:** Computer organization concerned with the way the hardware components operate and the way they are connected to and from the computer.

1.2.1 Computer-System Operation

- A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory.
- Each device controller is incharge of a specific type of device (for example, disk drives, audio devices,or video displays).
- The CPU and the device controllers can execute in parallel, competing for memory cycles. A memory controller synchronizes access to the memory.

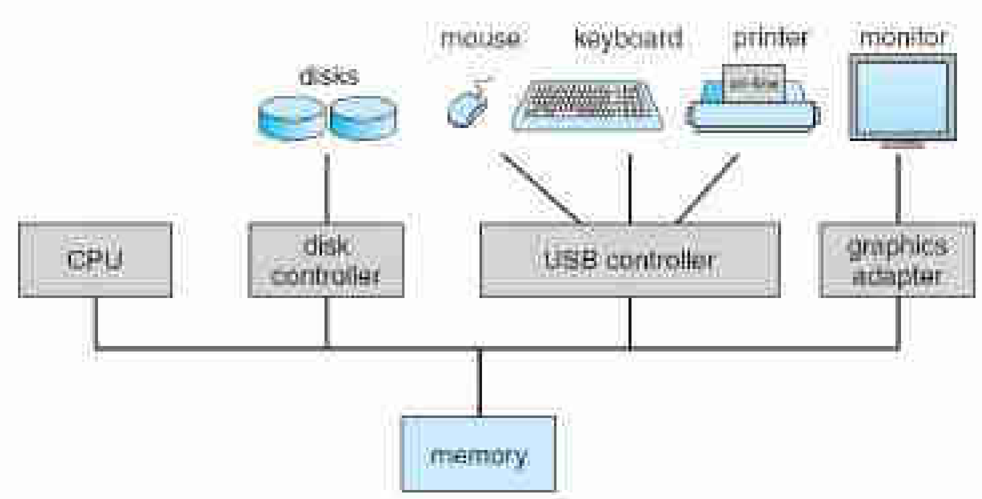


Fig 1.2: A modern computer system.

- **Bootstrap**: A bootstrap is the program that initializes the operating system (OS) during startup.

- When power is on, the bootstrap program loads the operating-system kernel in to memory. Once the kernel is loaded and executing, it can start providing services to the system and its users.
.
- On UNIX, the first system process is “init,” and it starts many other back ground processes. The process invoked by hardware or software request the service of CPU by passing an interrupt (system

call). When CPU is interrupted, it stops execution of current process temporarily and immediately provides service to the requested process and CPU resumes the stopped process later.

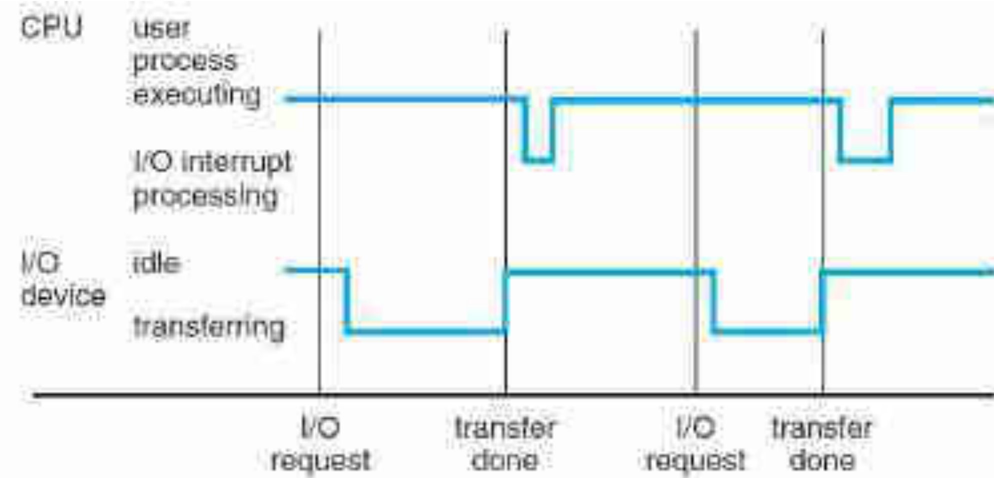


Fig1.3: Interrupt timeline for a single process doing

- When the CPU is interrupted, it stops what it is doing and immediately transfers the control to interrupt vector. The interrupt vector is an array that contains address of unique device numbers that request service.

1.2.2 Storage Structure

- The CPU can load instructions only from memory, so any programs to run must be stored there.
- **RAM**: RAM (Random Access Memory) is the internal memory of the CPU for storing data, program, and program result. It is a read/write memory which stores data until the machine is working. As soon as the machine is switched off, data is erased.
- **Dynamic random access memory (DRAM)** is a type of semiconductor memory that is typically used for the data or program code needed by a computer processor to function. DRAM is a common type of RAM that is used in personal computers (PCs), workstations and servers.
- **Memory hierarchy:** In computer architecture, the memory hierarchy separates computer storage into a hierarchy based on response time.

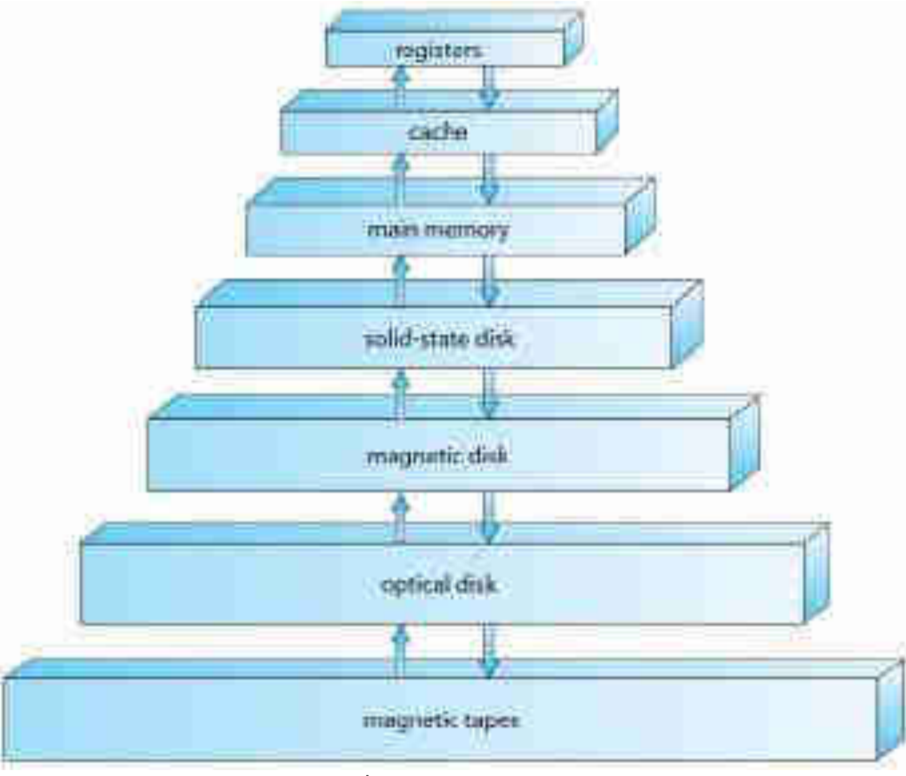


Fig 1.4: Storage Device Hierarchy.

1.2.3 I/O Structure

- A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device. For instance, seven or more devices can be attached to the **small computer-systems interface (SCSI)** controller.

- The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage. Typically, operating systems have a device driver for each device controller. This device driver understands the device controller and provides the rest of the operating system with a uniform interface to the device.

- DMA (Direct Memory Access): Direct memory access (DMA) is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations. The process is managed by a chip known as a DMA controller (DMAC).

1.3 COMPUTER-SYSTEM ARCHITECTURE

- Computer System Architecture: Computer System Architecture refers to how a computer system is designed and what technologies it is compatible with.

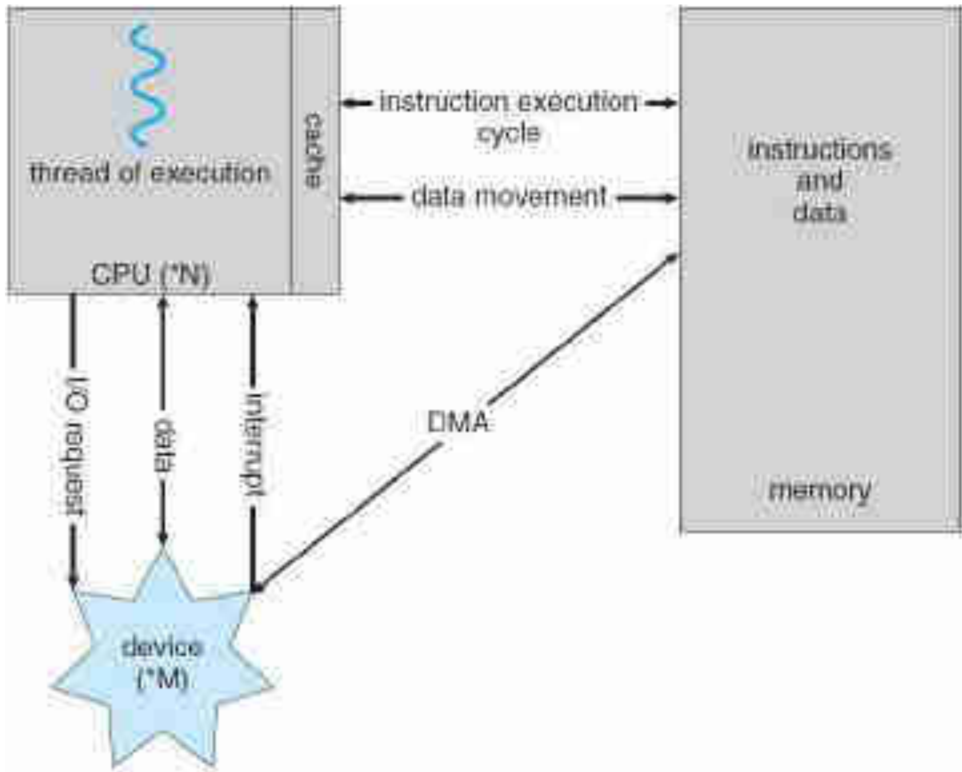


Fig 1.5: Storage Device Hierarchy.

- A computer system can be organized in a number of different ways, which we can categorize roughly according to the number of general-purpose processors used.

1.3.1 Single Processor System:

- A single processor system contains only one processor. So only one process can be executed at a time and then the process is selected from the ready queue. Even if there are multiple applications need to be executed, since the system contains a single processor and only one process can be executed at a time.

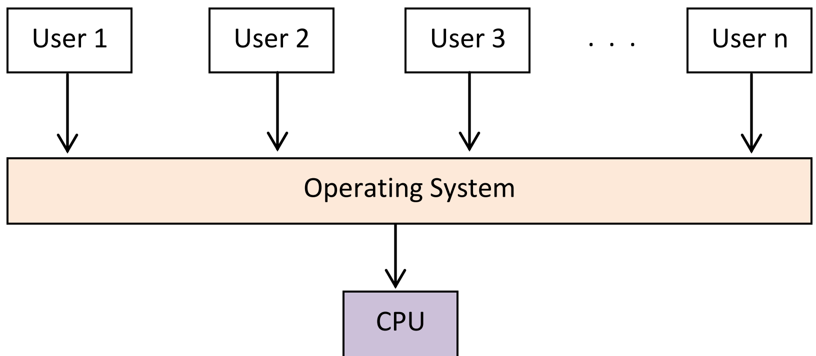


Fig 1.6: Single Processor System.

1.3.2 Multi Processor System

- A Multiprocessor is a computer system with two or more central processing units (CPUs) share full access to a common RAM. The main objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.

Multiprocessor systems have three main advantages:

1. **Increased throughput**: By increasing the number of processors, we expect to get more work done in less time.

2. **Economy of scale:** Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies.

3. **Increased reliability: T**he failure of one processor will not halt the system, only slow it down since the functions are distributed to multiple processes. If we have ten processors and one fails, then each of the remaining nine processors can pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether.

- Increased reliability of a computer system is crucial in many applications. **Graceful degradation** is the ability of a computer to maintain limited functionality even when a large portion of it has been destroyed or rendered inoperative. Some systems go beyond graceful degradation and are called **fault tolerant**, because they can suffer a failure of any single component and still continue operation.

- There are two main types of multiprocessor operating systems**.** The first one is *Asymmetric Multiprocessing System* (AMP or ASMP) and second one is *Symmetric Multiprocessing* (SMP).

- Asymmetric Multiprocessing system: Asymmetric Multiprocessing has the master-slave relationship among the processors. There is one master processor that controls remaining slave processor. The master processor allots processes to slave processor, or they may have some predefined task to perform. The master processor runs the tasks of the operating system.

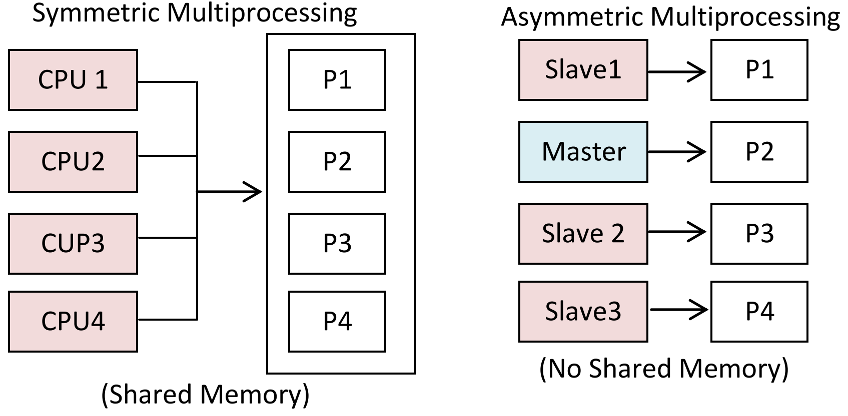


Fig 1.7: Symmetric and Asymmetric Multiprocessing.

- Symmetric Multiprocessing system: In symmetric multiprocessing, all the processors are treated equally. All processors communicate with another processor by a shared memory.

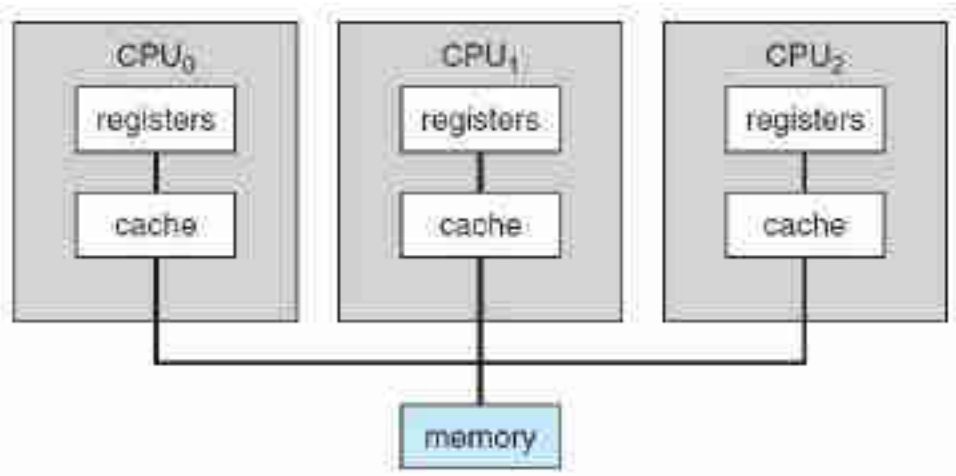


Fig 1.8: Symmetric multiprocessing architecture.

- A recent trend in CPU design is to include multiple computing cores on a single chip. Such multiprocessor systems are termed multicore.

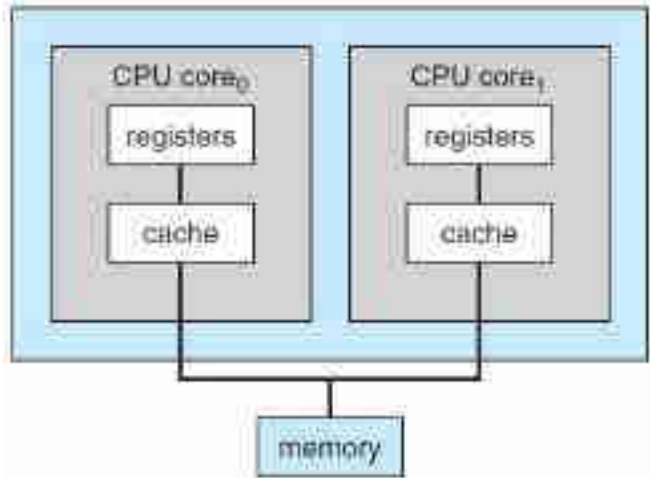


Fig 1.9: A dual-core design with two cores placed on the same chip.

1.3.3 Clustered Systems

- **Cluster Systems:** Cluster systems refer to a **group** or **cluster** of computers working together. This can be considered as a single system. The computer cluster can be lightly or loosely connected. The cluster systems are divided into nodes and each node have a specific task to accomplish. The task to be performed is assigned using the specially designed software.

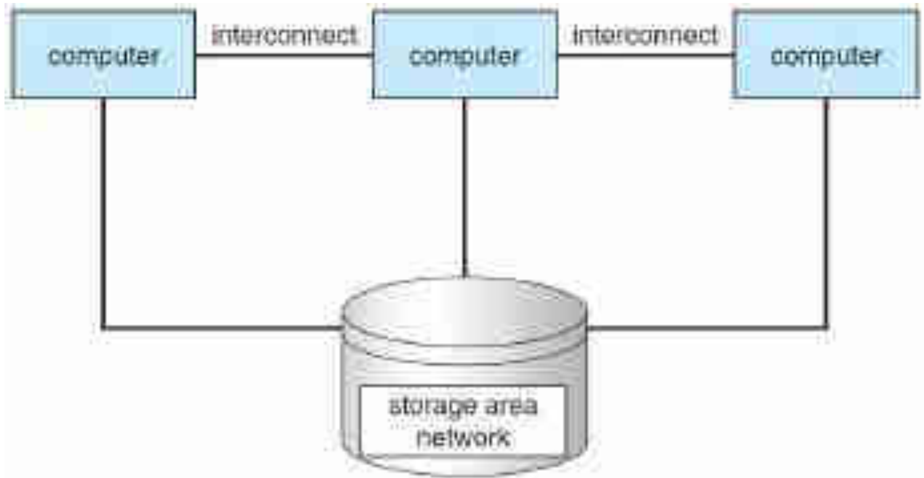


Fig 1.10: General Structure of Clustered System.

- Clustering can be structured asymmetrically or symmetrically.

- **Asymmetric Cluster System:**

In asymmetric clustering, one machine is in **hot-standby** mode while the other is running the applications. The hot-standby host machine does nothing but monitor the active server. If that server fails, the hot-standby host becomes the active server.

- **Symmetric Clustering System**

In symmetric clustering system two or more nodes all run applications as well as monitor each other. This is more efficient than asymmetric system as it uses all the hardware and doesn't keep a node merely as a hot standby.

1.3.3.1 Attributes of Clustered Systems

- There are many different purposes that a clustered system can be used for. Some of these can be scientific calculations, web support etc. The clustering systems that embody some major attributes are
- **Load Balancing Clusters**

In this type of clusters, the nodes in the system share the workload to provide a better performance. For example: A web based cluster may assign different web queries to different nodes so that the system performance is optimized. Some clustered systems use a round robin mechanism to assign requests to different nodes in the system.

- **High Availability Clusters**

These clusters improve the availability of the clustered system. They have extra nodes which are only used if some of the system components fail. So, high availability clusters remove single points of failure i.e. nodes whose failure leads to the failure of the system. These types of clusters are also known as failover clusters or HA clusters.

Advantages of clustered system:

The difference benefits of clustered systems are as follows

- **Performance:** Clustered systems result in high performance as they contain two or more individual computer systems merged together. These work as a parallel unit and result in much better performance for the system.
- **Fault Tolerance:** Clustered systems are quite fault tolerant and the loss of one node does not result in the loss of the system. They may even contain one or more nodes in hot standby mode which allows them to take the place of failed nodes.
- **Scalability:** Clustered systems are quite scalable as it is easy to add a new node to the system. There is no need to take the entire cluster down to add a new node.

1.4 OPERATING-SYSTEM STRUCTURE VVIMP

- Operating system can be implemented with the help of various structures.
- The structure of the OS depends mainly on how the various common components of the operating system are interconnected and melded into the kernel. Depending on this we have following structures of the operating system.

1.4.1 Simple Structure:

- Operating systems do not have well defined structure and offers limited features.
- MS-DOS is an example of such operating system. In MS-DOS application programs are able to access the basic I/O routines. These types of operating system cause the entire system to crash if one of the user programs fails.
- Diagram of the structure of MS-DOS is shown below.

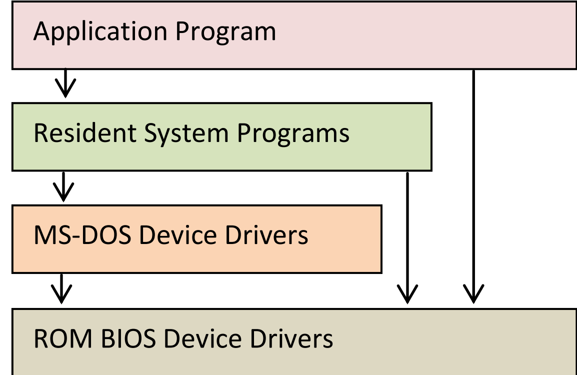


Fig 1.11: Simple structure of MS DOS Operating System.

Advantages:

- It delivers better application performance because of the few interfaces between the application program and the hardware.
- Easy for kernel developers to develop such an operating system.

Disadvantages:

- The structure is very complicated as no clear boundaries exist between modules.
- It does not enforce data hiding in the operating system.

1.4.2 Layered Structure:

- In this structure operating system is broken into number of layers (levels). The bottom layer (layer 0) is the hardware and the topmost layer (layer N) is the user interface.
- These layers are so designed that each layer uses the functions of the lower level layers only. This simplifies the debugging process since the error occurs in the higher level does not affect the lower level as the lower level layers have already been debugged.

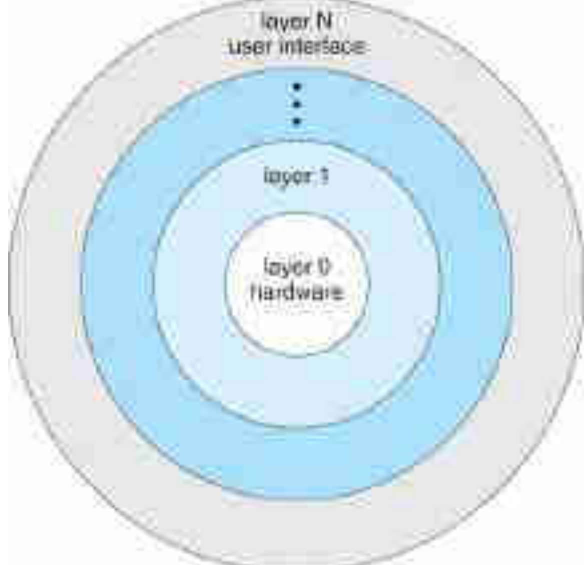


Fig 1.12: Layers of Operating System .

Advantages:

- Layering makes it easier to enhance the operating system as implementation of a layer can be changed easily without affecting the other layers.
- It is very easy to perform debugging and system verification.

Disadvantage:

- In this structure the application performance is degraded as compared to simple structure.
- It requires careful planning for designing the layers as higher layers use the functionalities of only the lower layers.

1.4.3 Micro-kernel:

- Microkernel is the minimum amount of software that can provide the mechanisms needed to implement services of an operating system (OS).
- These mechanisms include *low-level address space management*, *thread management*, and *inter-process communication (IPC)*.

Advantages:
- It makes the operating system portable to various platforms.
- As micro kernels are small so these can be tested effectively.

Disadvantages:
- Increased level of inter module communication degrades system performance.

1.4.4 Modular Structure:
- A modular operating system is built with its various functions broken up into distinct processes, each with its own interface. The main elements of a modular operating system are a kernel and a set of dynamically loadable applications with their own discrete memory spaces.

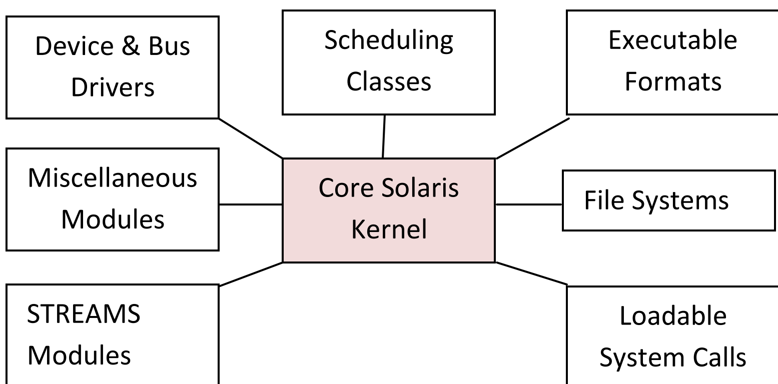


Fig 1.13: Solaris modular Structure.

Features of Modular Operating System:
- **Multiprogramming:** Several programs are run at the same time.

- **Time Sharing:** Time Sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

- **Job Scheduling:** Job Scheduling is the process of allocating system resources to many different tasks by an operating system (OS). The system handles prioritized job queues that are awaiting CPU time and it should determine which job to be taken from which queue and the amount of time to be allocated for the job.

- **Swapping:** Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

1.5 OPERARING SYSTEM OPERATIONS

1.5.1 Dual-Mode and Multimode Operation

- In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user defined code.
- There are two separate modes of operation the first one is **user mode** and the second one is **kernel mode** (also called supervisor mode, system mode, or privileged mode).

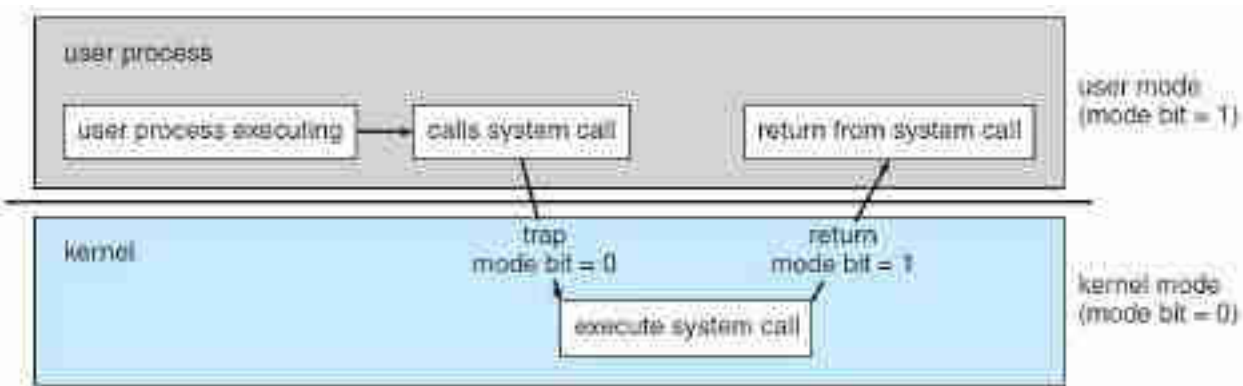


Fig 1.14 Transition from user to kernel mode.

- A bit, called the mode bit, is added to the hardware of the computer to indicate the current mode. If mode bit is 0 then the operating system is in kernel mode. If the mode bit is 1 then operation system is in user mode.
- When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), the system must transition from user to kernel mode to fulfill the request.

1.5.2 Timer

- We must ensure that the operating system maintains control over the CPU.
- We cannot allow a user program to get stuck in an infinite loop or to fail to call system services and never return control to the operating system. To accomplish this goal, we can use a timer.
- A timer can be set to interrupt the computer after a specified period. The period may be fixed (for example, 1/60 second) or variable (for example, from 1 millisecond to 1 second).

1.6 PROCESS MANAGEMENT

- Process: A process is a program in execution.
- A process needs certain resources including CPU time, memory, files, and I/O devices to accomplish its task.
- The operating system is responsible for the following activities in connection with process management.

1. Allocation of Process to Processor to perform its execution.
2. Deallocation of Process from Processor after its job is over.
3. Scheduling processes and threads on the CPUs.
4. Creating and deleting both user and system processes.
5. Suspending and resuming processes.
6. Providing mechanisms for process synchronization.
7. Providing mechanisms for process communication.

1.7 MEMORY MANAGEMENT

The operating system is responsible for the following activities in connection with memory management.

1. Allocation of memory to Process.
2. Deallocaton of Process from Memory.
3. Keeping track of which parts of memory are currently being used and who is using them.
4. Allocating and deallocating memory space as needed.

1.8 STORAGE MANAGEMENT

- The basic storage unit is File.

1.8.1 File System Management

- The operating system is responsible for the following activities in connection with file management

1. Creating and deleting files.
2. Creating and deleting directories to organize files.
3. Supporting primitives for manipulating files and directories.
4. Mapping files onto secondary storage.
5. Backing up files on stable (nonvolatile) storage media.

1.8.2 Mass Storage Management

- Most modern computer systems use disks as the principal online storage medium for both programs and data.
- Hence, the proper management of disk storage is of central importance to a computer system.
- The operating system is responsible for the following activities in connection with disk management.

1. Free-space Management

2. Storage Allocation
3. Disk Scheduling

1.8.3 Caching

- The process of storing and accessing data from the cache is known as Caching.
- A cache is a type of memory that is used to increase the speed of data access. In other words, a cache is a region of fast memory that holds copies of data.
- Access to the cached copy is more efficient than access to the original.

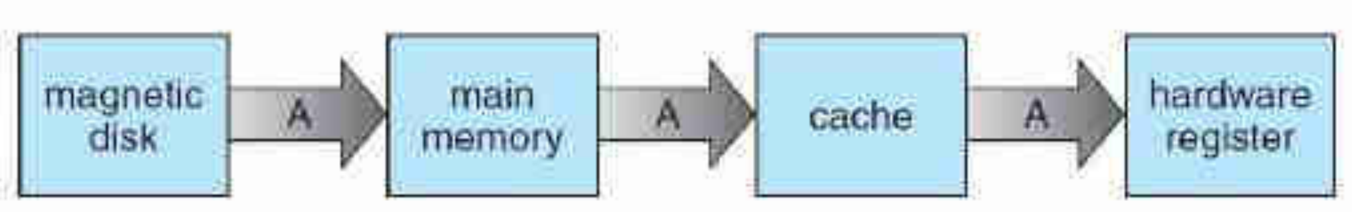


Fig 1.15: Migration of integer A from disk to register.

- **The Cache Coherence Problem:** In a multiprocessor system, data inconsistency may occur among adjacent levels or within the same level of the memory hierarchy. For example, the cache and the main memory may have inconsistent copies of the same object.

- As multiple processors operate in parallel, and independently multiple caches may possess different copies of the same memory block, this creates cache coherence problem. Cache coherence schemes help to avoid this problem by maintaining a uniform state for each cached block of data.

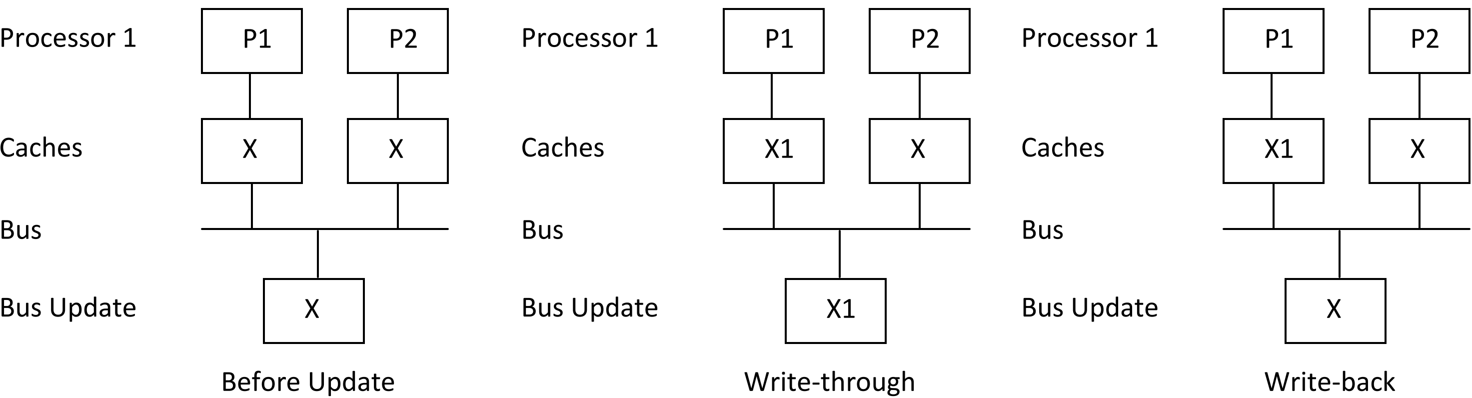


Fig 1.16: Illustration of Cache Coherence Problem.

- Let X be an element of shared data which has been referenced by two processors, P1 and P2. In the beginning, three copies of X are consistent. If the processor P1 writes a new data X1 into the cache, by using **write-through policy**, the same copy will be written immediately into the shared memory. In this case, inconsistency occurs between cache memory and the main memory. When a **write-back policy** is used, the main memory will be updated when the modified data in the cache is replaced or invalidated.

1.8.4 I/O Systems

- One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user.
- The I/O subsystem consists of several components

1. A memory-management component that includes buffering, caching, and spooling.
2. A general device-driver interface.
3. Drivers for specific hardware devices.

1.9 PROTECTION & SECURITY

- When the data is shared by multiple users simultaneously, it must be regulated.

- Protection: Protection refers to a mechanism which controls the access of programs, processes, or users to the resources defined by a computer system.

- Security: Operating system security is the process of ensuring operating system integrity, confidentiality and availability.

- Most operating systems maintain a list of user names and associated **user identifiers (user IDs)**

- Group functionality can be implemented as a system-wide list of group names and group identifiers.

- A user can be in one or more groups, depending on operating-system design decisions. The user's groups IDs are also included in every associated process and thread.

- However, a user sometimes needs to escalate privileges to gain extra permissions for an activity.

- The process runs with this effective UID until it turns off the extra privileges or terminates.

1.10 KERNEL DATA STRUCTURES

1.10.1 Lists, Stacks, and Queues

- The most common method for implementing this structure is a linked list, in which items are linked to one another.
- In a singly linked list, each item points to its successor

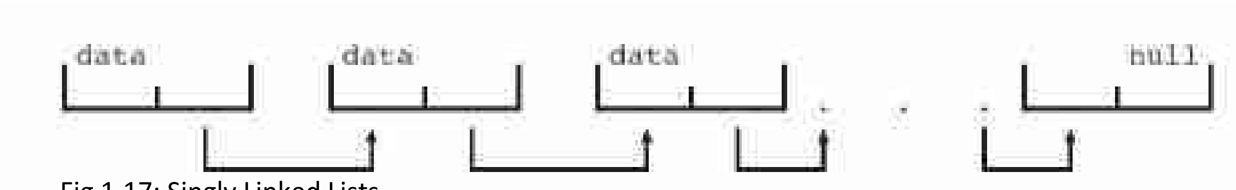


Fig 1.17: Singly Linked Lists.

- In a doubly linked list, a given item can refer either to its predecessor or to its successor.

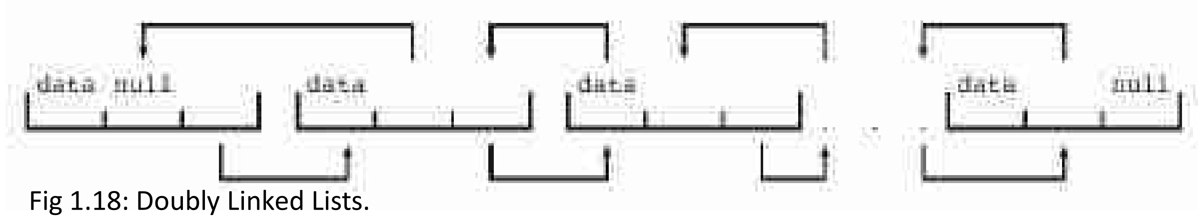


Fig 1.18: Doubly Linked Lists.

- In a circularly linked list, the last element in the list refers to the first element, rather than to null.

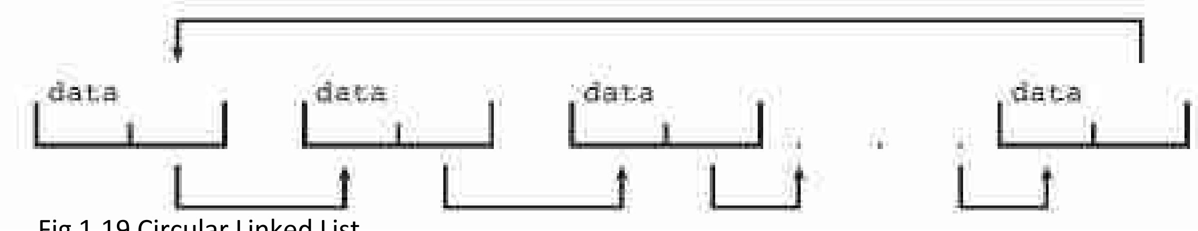


Fig 1.19 Circular Linked List.

1.10.2 Trees

- A tree is a data structure that can be used to represent data hierarchically.
- Data values in a tree structure are linked through *parent-child relationships*.
- In a general tree, a parent may have an unlimited number of children.
- In a binary tree, a parent may have at most two children, which we term the left child and the right child.
- A binary search tree additionally requires an ordering between the parent's two children in which le ft child <= right child.

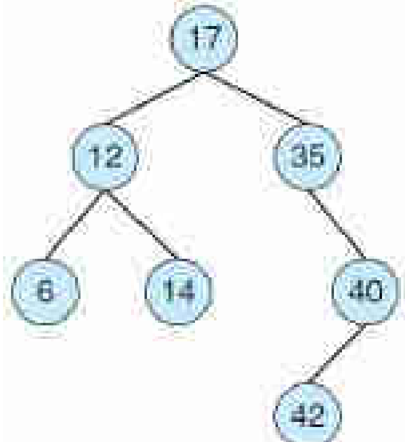


Fig 1.20 Binary Search Tree.

1.10.3 Hash Functions and Maps

- A **hash function** takes data as its input, performs a numeric operation on this data, and returns a numeric value.
- This numeric value can then be used as an index into a table (typically an array) to quickly retrieve the data.
- Whereas searching for a data item through a list of size n can require up to O(n) comparisons in the worst case, using a hash function for retrieving data from table can be as good as O(1) in the best case, depending on implementation details. Because of this performance, hash functions are used extensively in operating systems.

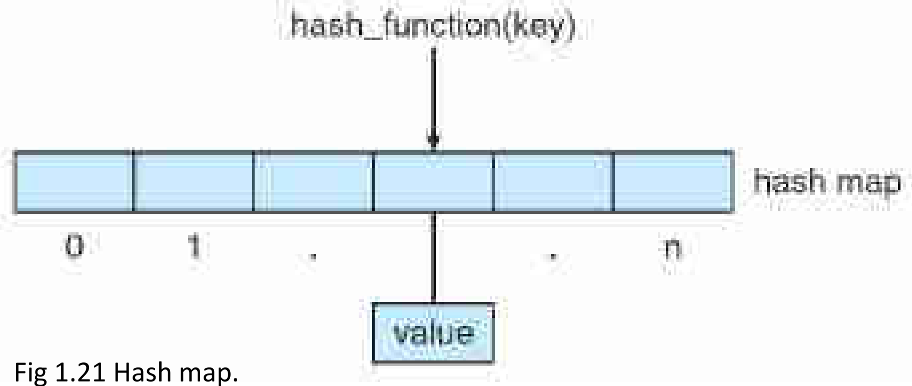
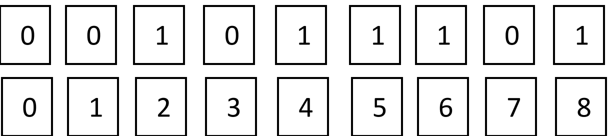


Fig 1.21 Hash map.

- One potential difficulty with hash functions is that two inputs can result in the same output value that is, they can link to the same table location. We can accommodate this hash collision by having a linked list at that table location that contains all of the items with the same hash value. Of course, the more collisions there are, the less efficient the hash function is.

1.10.4 Bitmaps

- A bitmap is a string of *n* binary digits that can be used to represent the status of *n* items. For example, suppose we have several resources, and the availability of each resource is indicated by the value of a binary digit: 0 means that the resource is available,while 1 indicates that it is unavailable (or vice-versa).
- The value of the ith position in the bitmap is associated with the ith resource. As an example, consider the bitmap shown below:



Resources 2, 4, 5, 6, and 8 are unavailable; resources 0, 1, 3, and 7 are available.

1.11 COMPUTING ENVIRNMENT

1.11.1 Traditional Computing

- This environment consisted of PCs connected to a network, with servers providing file and print services.

- Terminals attached to mainframes were common at many companies.

- Web Computing: Companies establish **portals**, which provide Web accessibility to their internal servers. Network computers (or **thin clients**) access the web portals.

- Mobile computers can also connect to wireless networks and cellular data networks to use the company's Web portal.

- Firewalls are used to protect networks from security breaches.

1.11.2 Mobile Computing

- Mobile Computing is a technology that allows transmission of data, voice and video via a computer or any other wireless enabled device without having to be connected to a fixed physical link.

- The main concept involves

1. Mobile communication
2. Mobile hardware
3. Mobile software

Mobile communication:

- The mobile communication in this case, refers to the infrastructure put in place to ensure that seamless and reliable communication goes on.

- These would include devices such as protocols, services, bandwidth, and portals necessary to facilitate and support the stated services.
- The data format is also defined at this stage. This ensures that there is no collision with other existing systems which offer the same service.
- Since the media is unguided / unbounded, the overlaying infrastructure is basically radio wave-oriented. That is, the signals are carried over the air to intended devices that are capable of receiving and sending similar kinds of signals.

Mobile Hardware:

- Mobile Hardware includes mobile devices or device components that receive or access the service of mobility.

E.g. Portable Laptops, Smartphones, Tablet Pc's, Personal Digital Assistants.

- Mobile Software: Mobile software is the actual program that runs on the mobile hardware.

E.g. Apple iOS, Google Android.

1.11.3 Distributed Systems

- A distributed system is a computing environment in which various resources are spread across multiple computers (or other computing devices) on a network.
- Access to a shared resource increases computation speed, functionality, data availability, and reliability.
- A network operating system allows file sharing across the networks.

1.11.4 Client-Server Computing

- Client-server computing is a distributed computing model in which client applications request services from server processes. Clients and servers typically run on different computers interconnected by a computer network
- Server systems can be broadly categorized as compute servers and file servers.

Computer server system: The compute server system provides an interface to which a client can send a request to perform an action (for example, read data). In response, the server executes the action and sends the results to the client. A server running a database that responds to client requests for data is an example of such a system.

File server system: The file-server system provides a file-system interface where clients can create, update, read, and delete files. An example of such a system is a web server that delivers files to clients running web browsers.

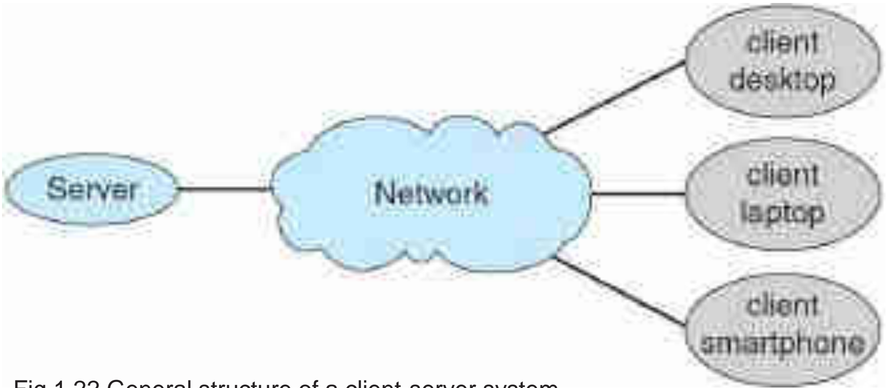


Fig 1.22 General structure of a client-server system.

1.11.5 Peer-to-Peer Computing

- In peer-to-peer (P2P) networking, a group of computers are linked together with equal permissions and responsibilities for processing data. Unlike traditional client-server networking, no devices in a P2P network are designated solely to serve or to receive data. Each connected machine has the same rights as its "peers", and can be used for the same purposes.

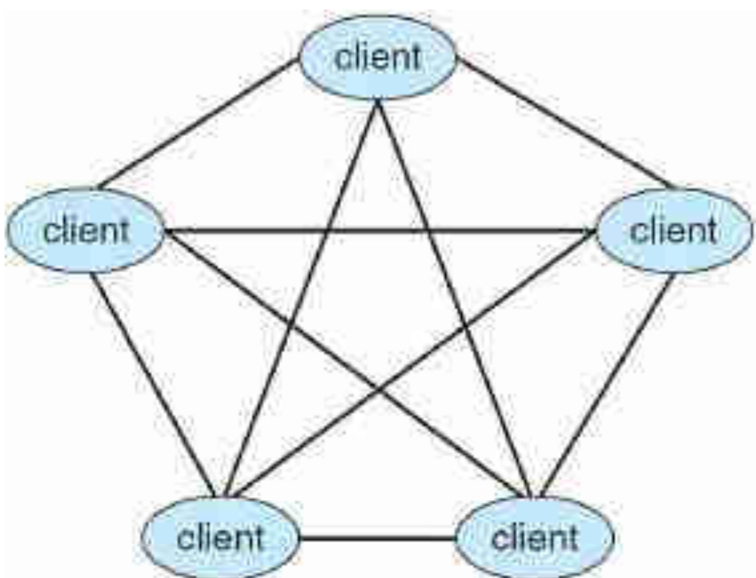
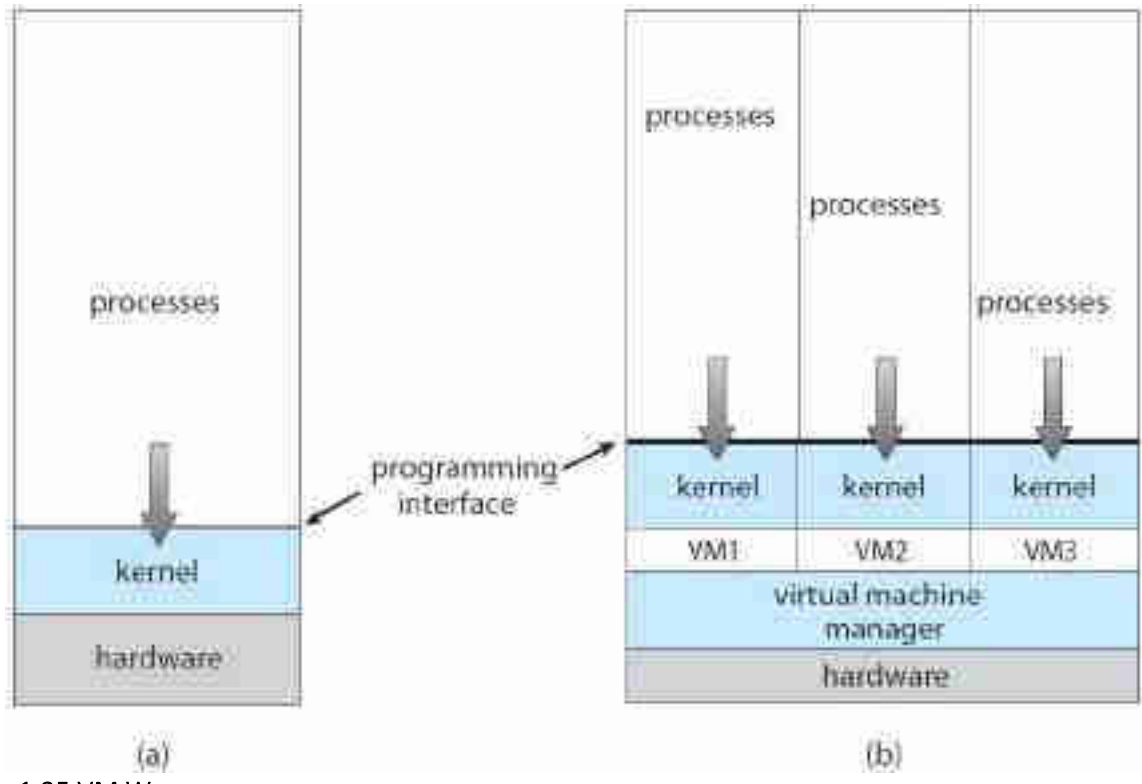


Fig 1.23 Peer-to-peer system with no centralized service.

- Skype is an example of peer-to-peer computing. It allows clients to make voice calls and video calls and to send text messages over the Internet using a technology known as voice over IP (VoIP).

1.11.6 Virtualization

- Virtualization is a technology that allows operating systems to run as applications within other operating systems.



1.25 VM Ware.

- Windows was the host operating system, and the VMware application was the virtual machine manager VMM. The VMM runs the guest operating systems, manages their resource use, and protects each guest from the others.

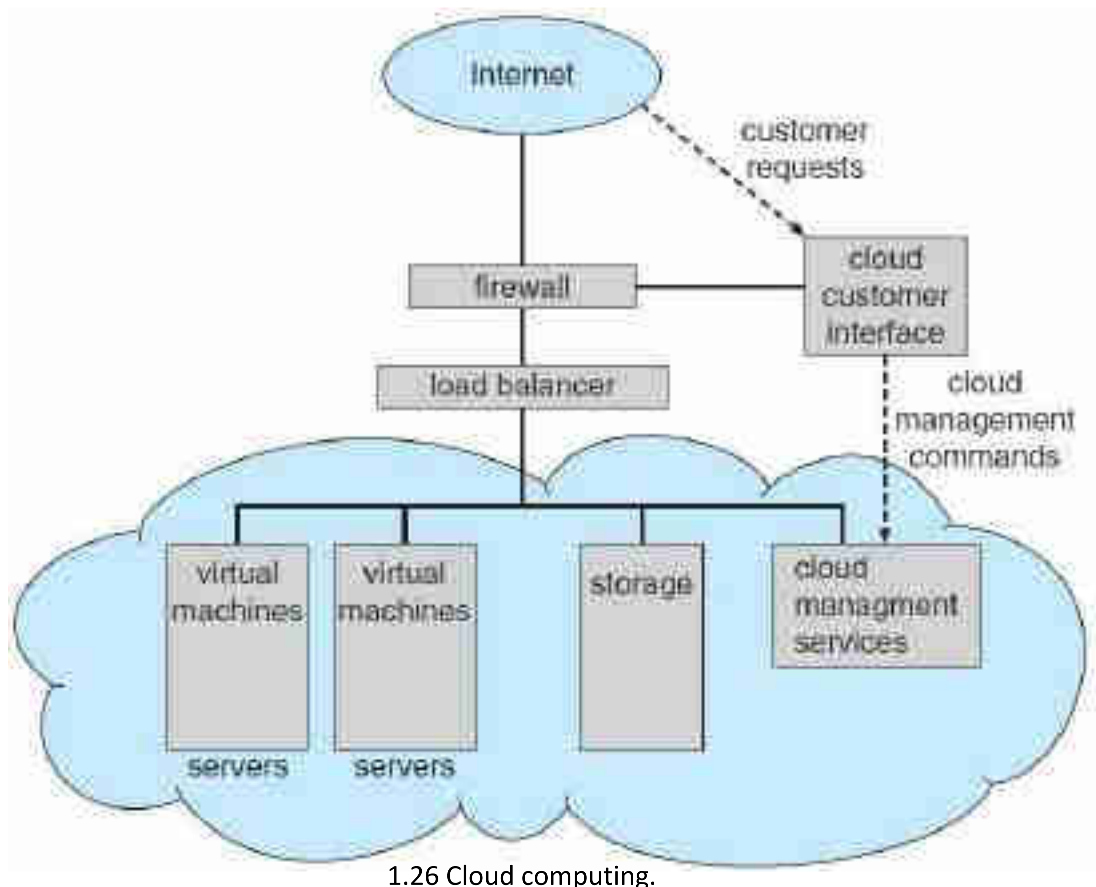
1.11.7 Cloud Computing:

- Cloud is a model of computing where servers, networks, storage, development tools, and even applications (apps) are enabled through the internet. Instead of organizations having to make major investments to buy equipment, train staff, and provide ongoing maintenance, some or all of these needs are handled by a cloud service provider.

There are actually many types of cloud computing, including the following:

- Public cloud: A cloud available via the Internet to anyone willing to pay for the services.
- Private cloud: A cloud run by a company for that company's own use.
- Hybrid cloud: A cloud that includes both public and private cloud components.
- Software as a service (SaaS): One or more applications (such as word processors or spreadsheets) available via the Internet.
- Platform as a service (PaaS): A software stack ready for application use via the Internet (for example, a database server).

- Infrastructure as a service (IaaS): Servers or storage available over the Internet (for example, storage available for making backup copies of production data).



1.26 Cloud computing.

1.11.7 Real-Time Embedded Systems

- An embedded system is a combination of computer hardware and software designed for a specific function.
- Embedded systems may also function within a larger system.
- The systems can be programmable or have a fixed functionality.
- Industrial machines, consumer electronics, agricultural and processing industry devices, automobiles, medical equipment, cameras, digital watches, household appliances, airplanes, vending machines and toys, as well as mobile devices, are possible locations for an embedded system.
- A real-time system has well-defined, fixed time constraints. Processing must be done within the defined constraints, or the system will fail. For instance, it would not do for a robot arm to be instructed to halt after it had smashed into the car it was building.

1.12 OPEN-SOURCE OPERATING SYSTEMS

- Open source software is code that is designed to be publicly accessible-anyone can see, modify, and distribute the code as they see fit.

1.12.1 Linux

- Linux is a free, open source operating system, released under the GNU General Public License (GPL). Anyone can run, study, modify, and redistribute the source code, or even sell copies of their modified code, as long as they do so under the same license.

Features:

- Portable
- Open Source
- Multi-User
- Multiprogramming Hierarchical File System
- Shell
- Security

- Major distributions include RedHat, SUSE, Fedora, Debian, Slackware, and Ubuntu.

1.12.2 BSD UNIX

- BSD UNIX stands for Berkeley Software Distribution UNIX, a version of UNIX that originated many common UNIX features such as the vi editor, C shell, and TCP/IP networking.

Features:

- TCP/IP
- Virtual Memory,
- Berkeley Fast File System.

1.12.4 Solaris

- Solaris is the commercial UNIX-based operating system of Sun Microsystems.

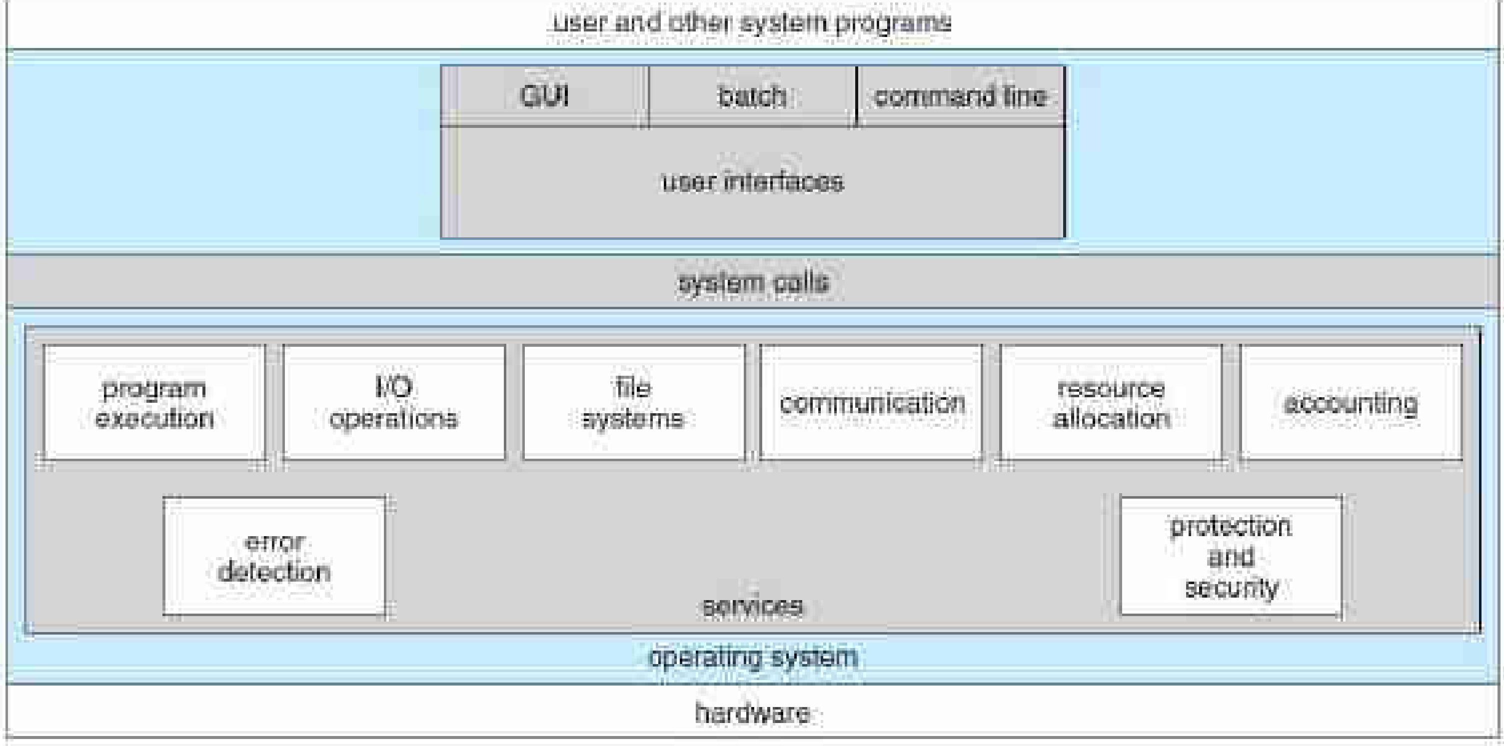
Features:

- Hardware Support.
- I/O Management
- Provide Device Drivers.
- Process Scheduling.
- Process Management.
- Memory Management.
- File System Implementation.
- Provide I/O Services.

CHAPTER 2
OPERRATING SYSTEM STRUCTURE

2.1 OPERATING SYSTEM SERVICES VIMP

- Operating system provides certain services to programs and to the users of those programs.



2.1 Overview of OS Services.

2.1.1 User Interface

- Almost all operating systems have a user interface (UI). This interface can take several forms. The first one is a command-line interface (CLI), which uses text commands.
- The second one is a batch interface, in which commands and directives to control those commands are entered into files, and those files are executed.
- The third one is a graphical user interface (GUI). Menus allow the user to execute commands by selecting from a list of choices. Options are selected with a mouse or other pointing device within a GUI. A keyboard may also be used. Menus are convenient because they show what commands are available within the software.

2.1.2 Program execution

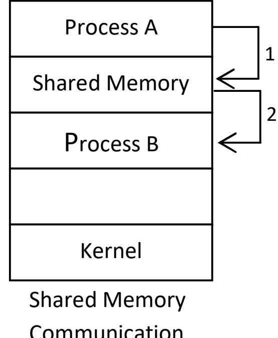
- The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).

2.1.3 File-system manipulation

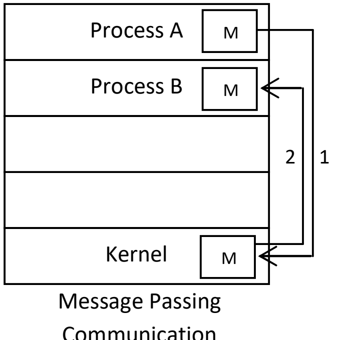
- File: A file is a collection of related information that is recorded on secondary storage.
- File Directories: Collection of files is a file directory.
- The directory contains information about the files, including attributes, location and ownership.
- Much of this information, especially that is concerned with storage, is managed by the operating system.

2.1.4 Communications

- Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions.
- Processes can communicate with each other through both:
  Shared Memory
  Message Passing



Shared Memory Communication



Message Passing Communication

2.1.5 Error detection

- The operating system needs to be detecting and correcting errors constantly.
- Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on disk, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time).

- For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing. Sometimes, it has no choice but to halt the system. At other times, it might terminate an error-causing process or return an error code to a process for the process to detect and possibly correct.

2.1.6. Resource allocation

- The Operating System allocates resources when a program needs them. When the program terminates, the resources are de-allocated, and allocated to other programs that need them.
- There are two Resource allocation techniques:

(a) **Resource Partitioning Approach:** In this approach, the operating system decides beforehand, that what resources should be allocated to which user program. It divides the resources in the system to many resource partitions, where each partition may include various resources.

(b) **Pool Based Approach:** In this approach, there is a *common pool of resources*. The operating System checks the allocation status in the resource table whenever a program makes a request for a resource. If the resource is free, it allocates the resource to the program.

2.1.7 Accounting

- Operating system keeps track of time and resources used by various tasks and users, this information can be used to track resource usage for a particular user or group of user.

2.1.8 Protection & Security

- Protection involves ensuring that all access to system resources is controlled.
- Security of the system from outsiders is also important. Such security starts with requiring each user to authenticate by means of a password, to gain access to system resources.

2.2 SYSTEMS CALLS VVIMP

- System Call: A system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.
- A computer program makes a system call when it makes a request to the operating system's kernel.
- System call provides the services of the operating system to the user programs via Application Program Interface (API).

2.2.1 Services Provided by System Calls:

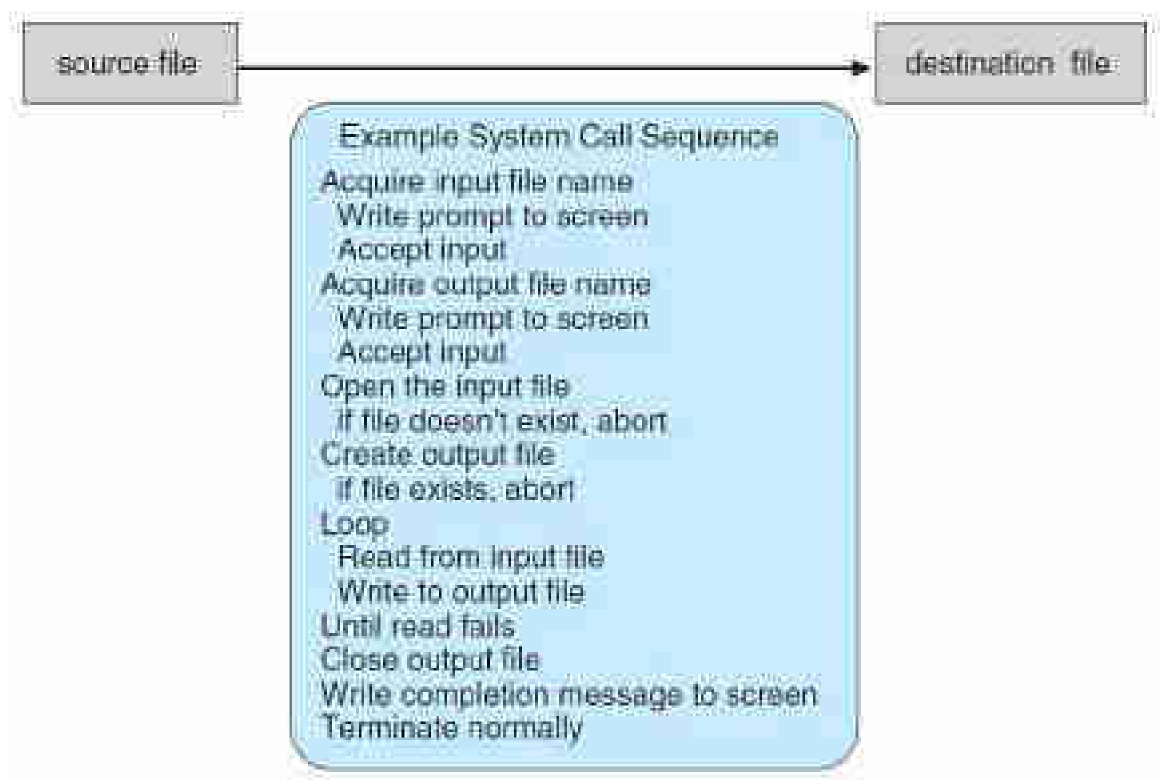
1. Process Creation and Management
2. Main Memory Management
3. File Access, Directory and File System Management

4. Device Handling(I/O)
5. Protection
6. Networking, etc.

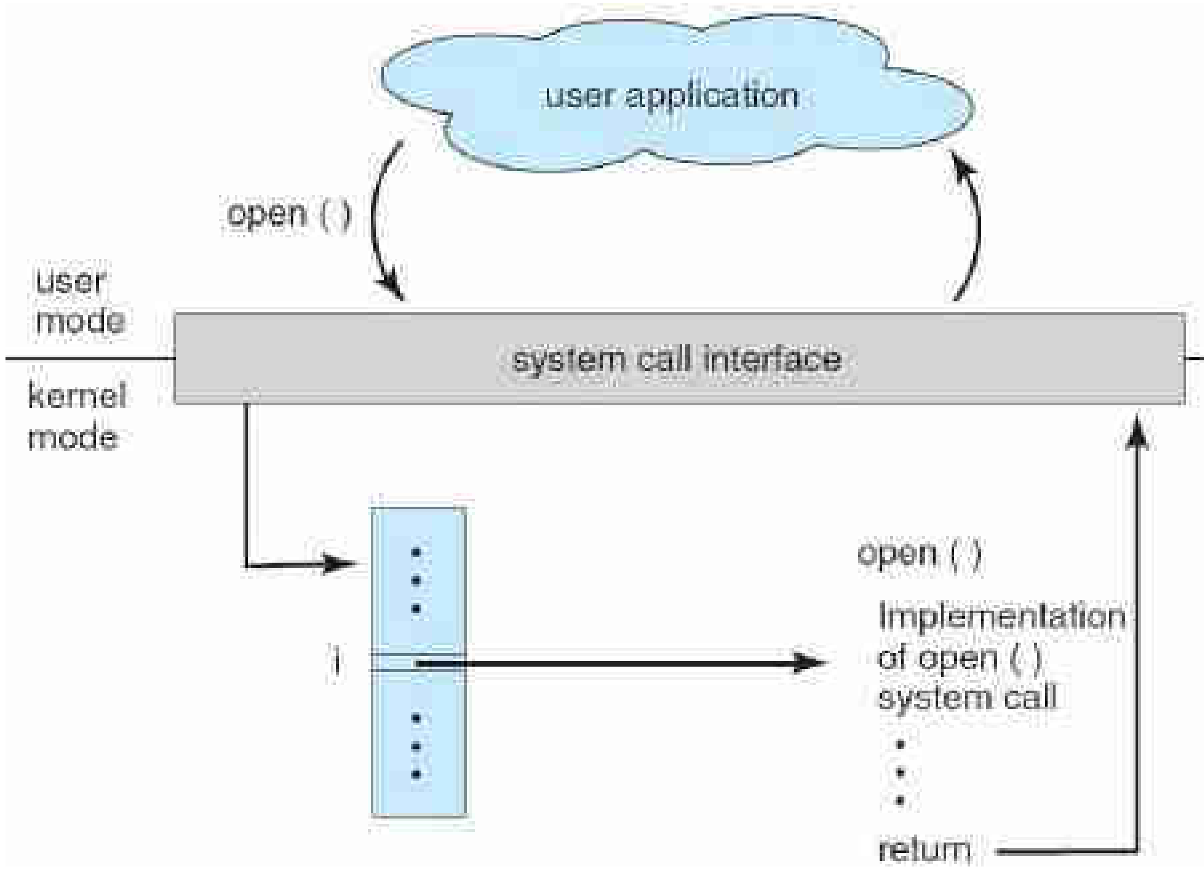
2.2.2 Types of System Calls

There are 5 different categories of system calls:

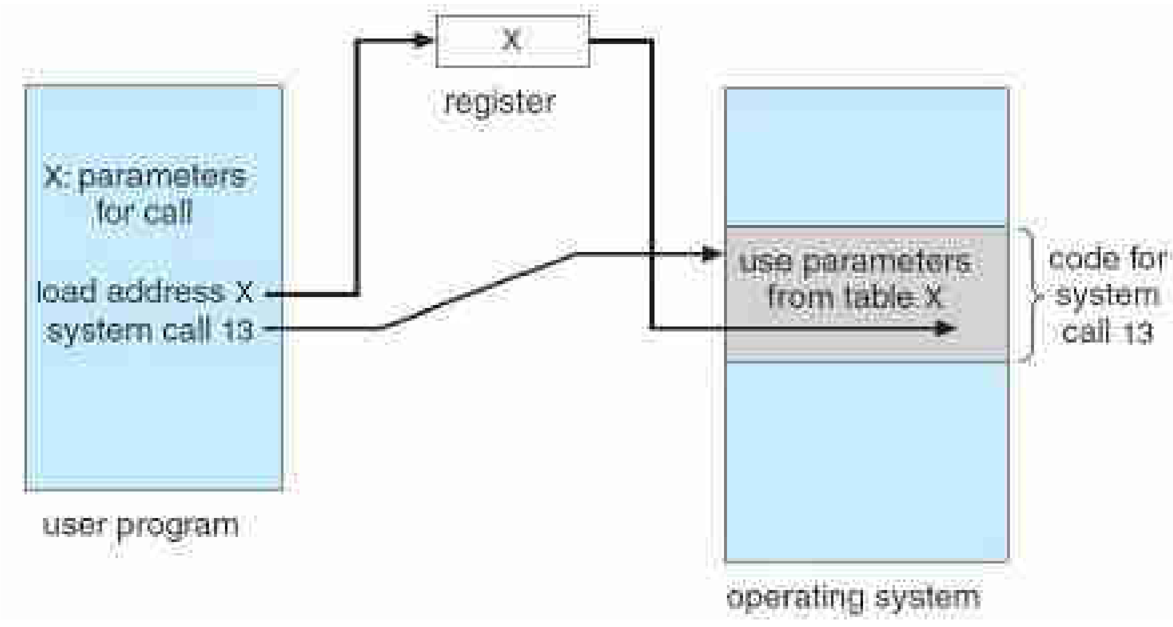
1. Process Control: end, abort, create, terminate, allocate and free memory.
2. File Management: create, open, close, delete, read file etc.
3. Device Management
4. Information Maintenance
5. Communication



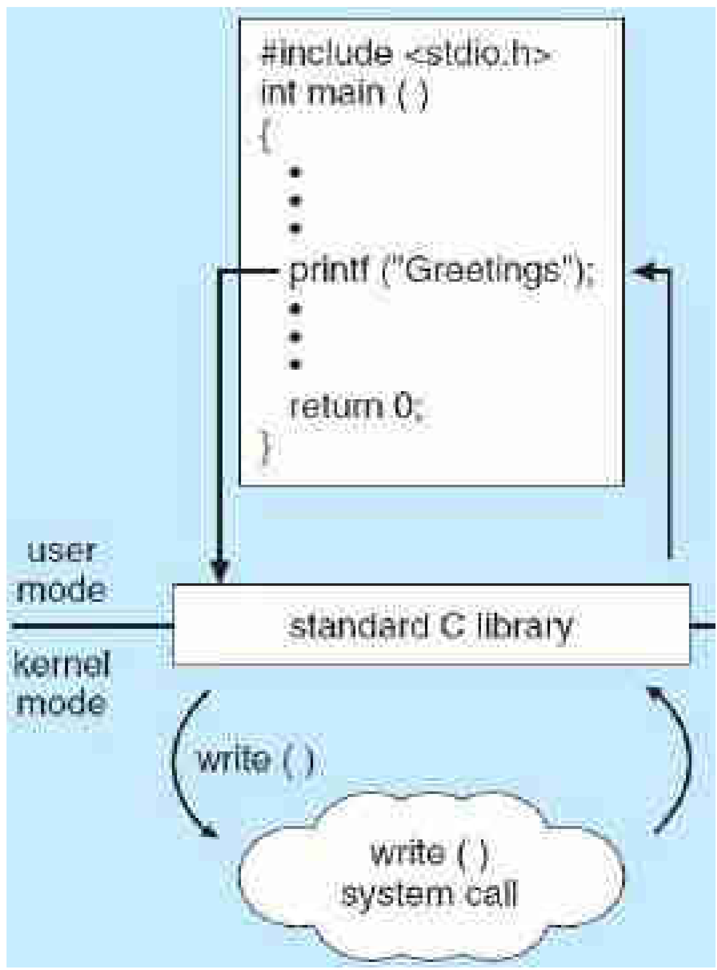
2.2 Example of how system calls are used.



2.3 The handling of a user application invoking the open()



2.4 Passing of parameters as a table.



2.5 System call interface of C Standard Library.

2.3 SYSTEM PROGRAMS

- The system program serves as a part of the operating system.
- System programs can be divide into seven categories

File management. These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.

2.4 TYPES OF SYSTEM CALLS VVIMP

- System calls can be grouped roughly into six major categories: Process Control, File Manipulation, Device Manipulation, Information Maintenance, Communications, and Protection.

PROCESS CONTROL
▪ end, abort ▪ load, execute ▪ create process, terminate process ▪ get process attributes, set process attributes ▪ wait for time ▪ wait event, signal event ▪ allocate and free memory
FILE MANAGEMENT ▪ create file, delete file ▪ open, close ▪ read, write, reposition ▪ get file attributes, set file attributes
DEVICE MANAGEMENT ▪ request device, release device ▪ read, write, reposition ▪ get device attributes, set device attributes ▪ logically attach or detach devices
INFORMATION MAINTENANCE ▪ get time or date, set time or date ▪ get system data, set system data ▪ get process, file, or device attributes ▪ set process, file, or device attributes
COMMUNICATIONS ▪ create, delete communication connection ▪ send, receive messages ▪ transfer status information ▪ attach or detach remote devices

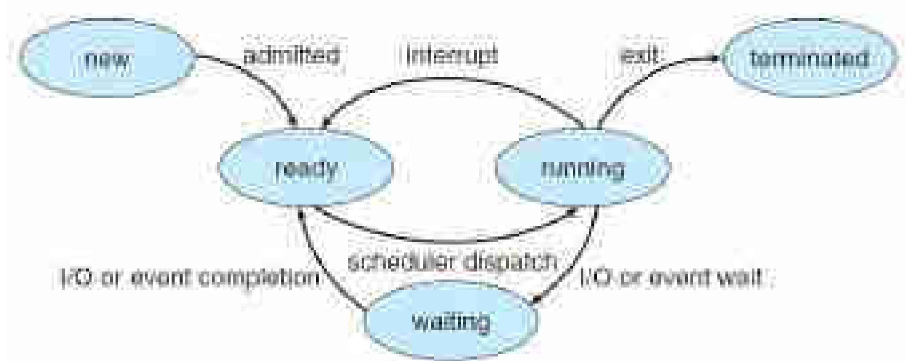
CHAPTER 3
PROCESSES

3.1 PROCESS CONCEPT IMP

- Process: A process is basically a program in execution.

3.1.2 Process States

A process may be in one of the following states.
New: The process is being created.
Running: Instructions are being executed.
Waiting: The process is waiting for some event to occur.
Ready: The process is waiting to be assigned to a processor.
Terminated: The process has finished execution.



3.1 Process States.

3.1.3 Process Control Block

- Each process is represented in the operating system by a Process Control Block (PCB).

3.2 Process Control Block.

- Process State: The state may be new, ready, running, waiting, halted, and so on.
- Program Counter: The counter indicates the address of next instruction to be executed for this process.
- CPU Registers: The registers may vary in number and type, depending on the computer architecture. They include *Accumulator*, *Index Registers*, *Stack Pointers* and *General Purpose Registers*.
- CPU Scheduling Information: This information includes a Process Priority.

- Memory Management Information: Include the value of the base and limit registers and the page tables.

- Accounting Information: This information includes the amount of CPU time used, time limits, account numbers, job or process numbers, and so on.

- I/O Status Information: This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

3.1.4 Threads

- A thread is a path of execution within a process.
- A thread is also called a lightweight process.

3.2 PROCESS SCHEDULING VIMP

- **Process Scheduling:** The act of determining which process is in the ready state, and should be moved to the running state is known as Process Scheduling.

- The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs.

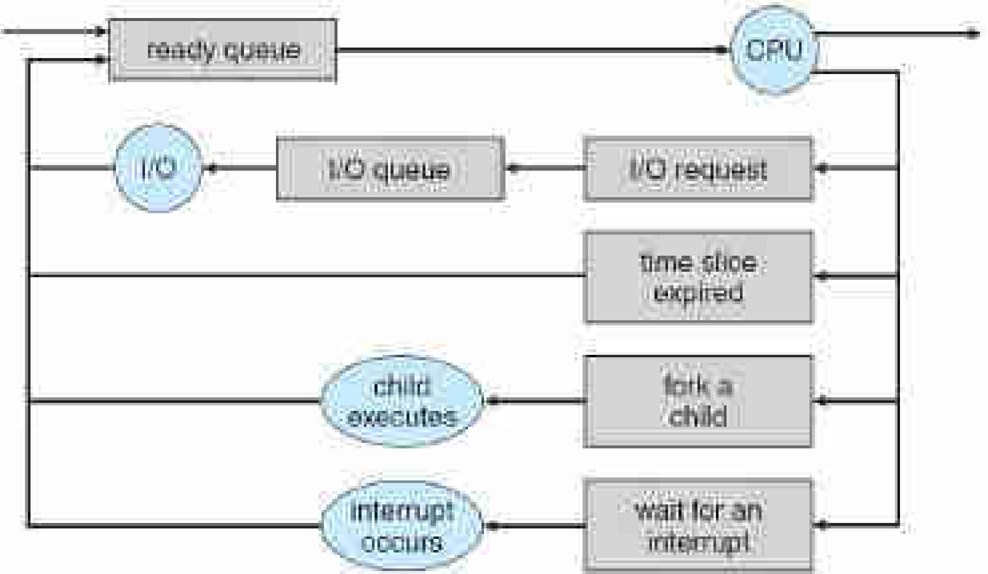
- Scheduling fell into one of the two general categories:

1. **Non Pre-emptive Scheduling:** The currently executing process releases the CPU voluntarily, after its job is over.
2. **Pre-emptive Scheduling:** The operating system allocates process to the CPU by pre-empting the currently executing process.

3.2.1 Scheduling Queues

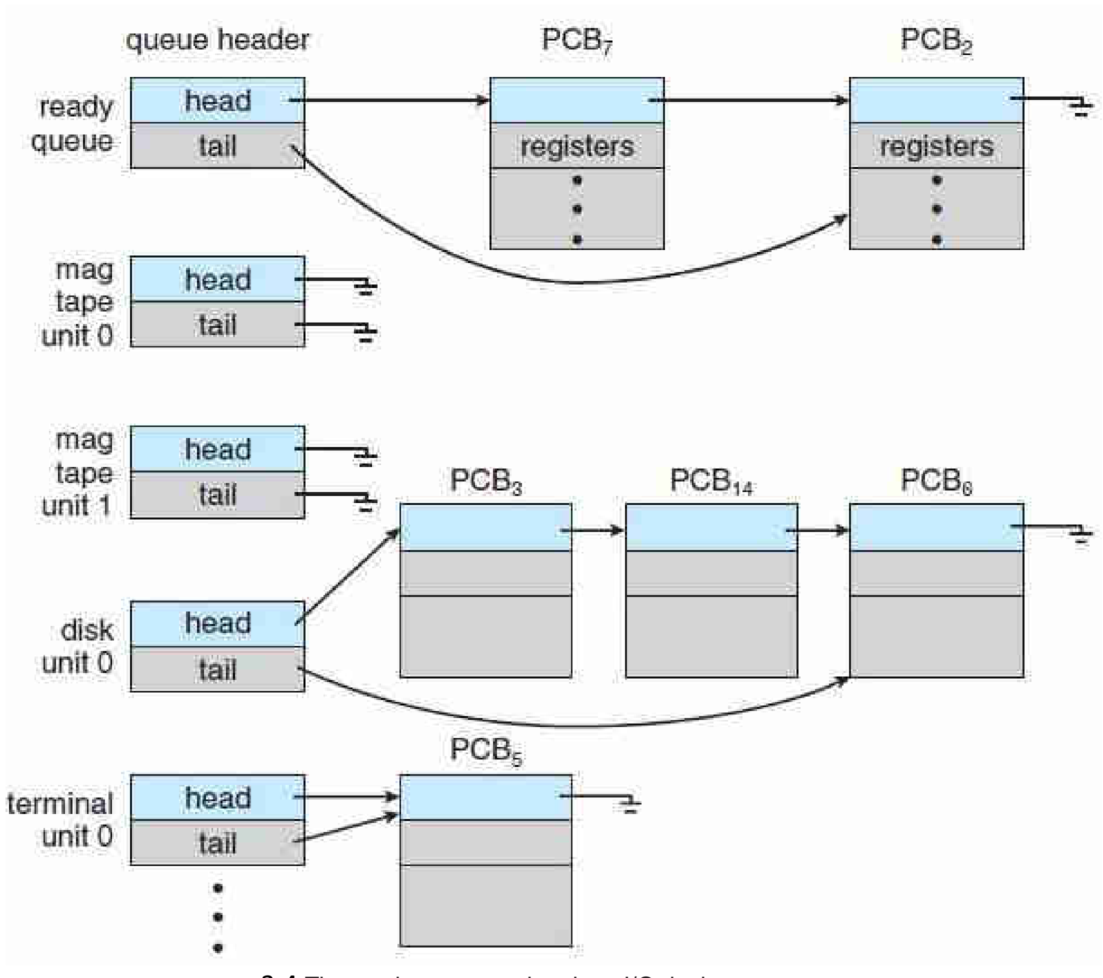
- A new process is initially put in the **ready queue**. It waits in the ready queue until it is selected for execution. Once the process is assigned to the CPU and is executing, one of the following several events can occur.

1. The process could issue an I/O request, and then be placed in the **I/O queue**.
2. The process could create a new sub process and wait for its termination.
3. The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.



3.3 Queueing-diagram representation of process scheduling.

- The process scheduler selects an available process for program execution on the CPU.



3.4 The ready queue and various I/O device queues.

3.2.2 Schedulers

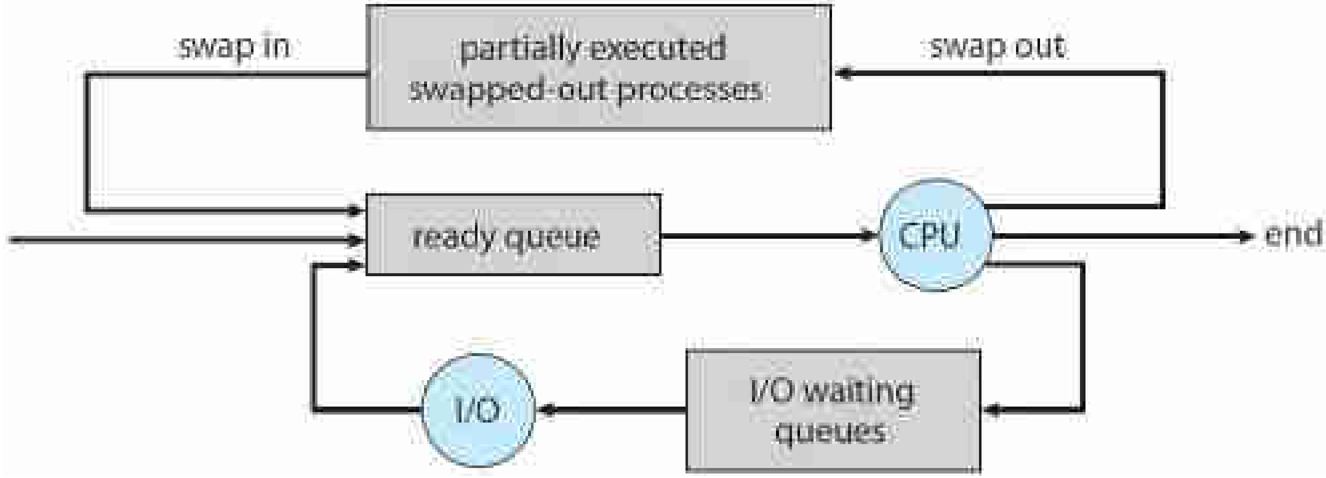
- A scheduler is a type of system software that allows you to handle process scheduling.

- There are three types of schedulers available
  1. Long Term Scheduler
  2. Short Term Scheduler
  3. Medium Term Scheduler

Long Term Scheduler: Long term scheduler is also known as a **job scheduler**. This scheduler regulates the program and select process from the queue and loads them into memory for execution. It also regulates the degree of multi-programing.

Medium Term Scheduler: Medium term is also called **swapping scheduler**. It handle the swapped out-processes. In this scheduler, a running process can become suspended, which makes an I/O request.

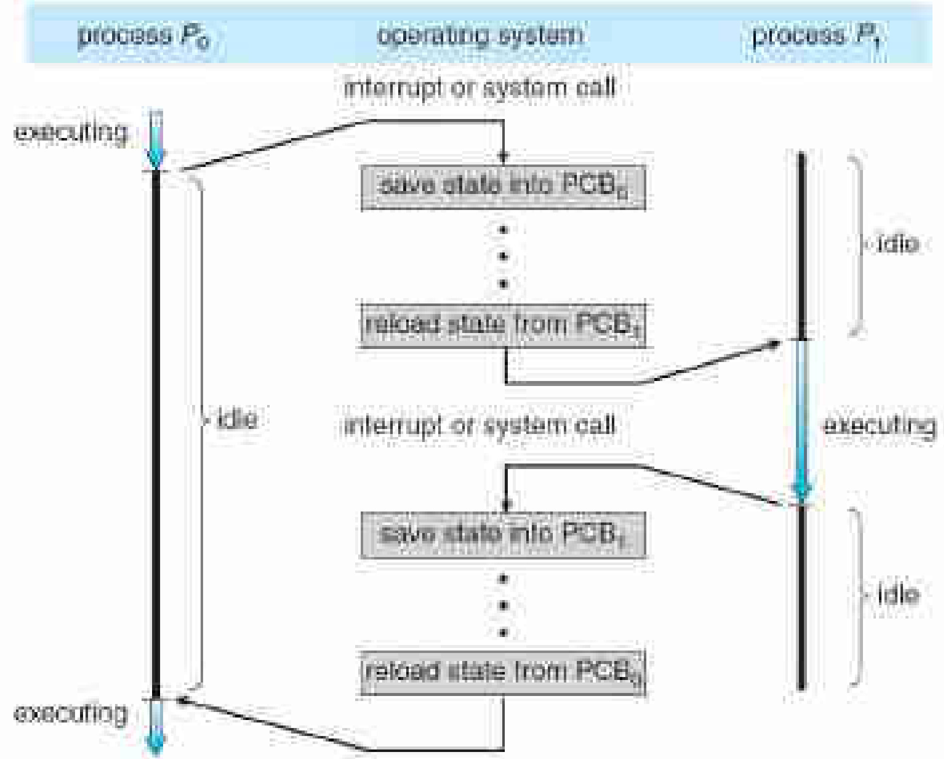
Short Term Scheduler: The primary aim of this scheduler is to enhance CPU performance and increase process execution rate. Short term scheduling is also known as **CPU scheduler**. This helps you to select from a group of processes that are ready to execute and allocates CPU to one of them. The dispatcher gives control of the CPU to the process selected by the short term scheduler.



3.5 Addition of medium-term scheduling to the queueing diagram.

3.2.3 Context Switch

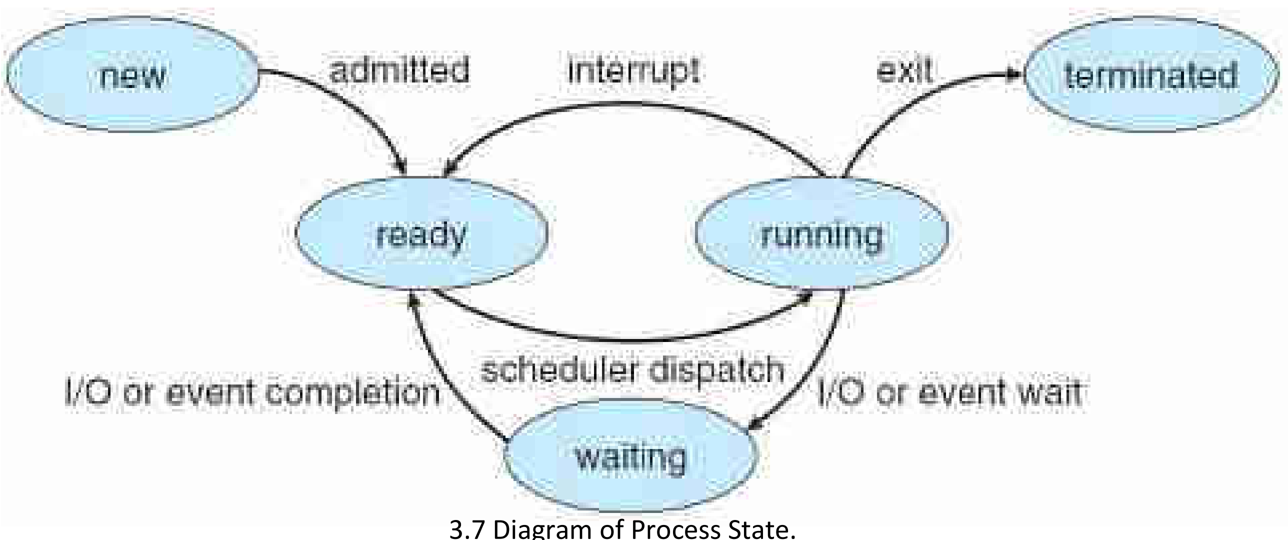
- In computing, a context switch is the process of storing the state of a process or thread, so that it can be restored and resume execution at a later point. This allows multiple processes to share a single central processing unit (CPU), and is an essential feature of a multitasking operating system.



3.6 Diagram showing CPU switch from Process to Process.

3.3 OPERATIONS ON PROCESSES IMP

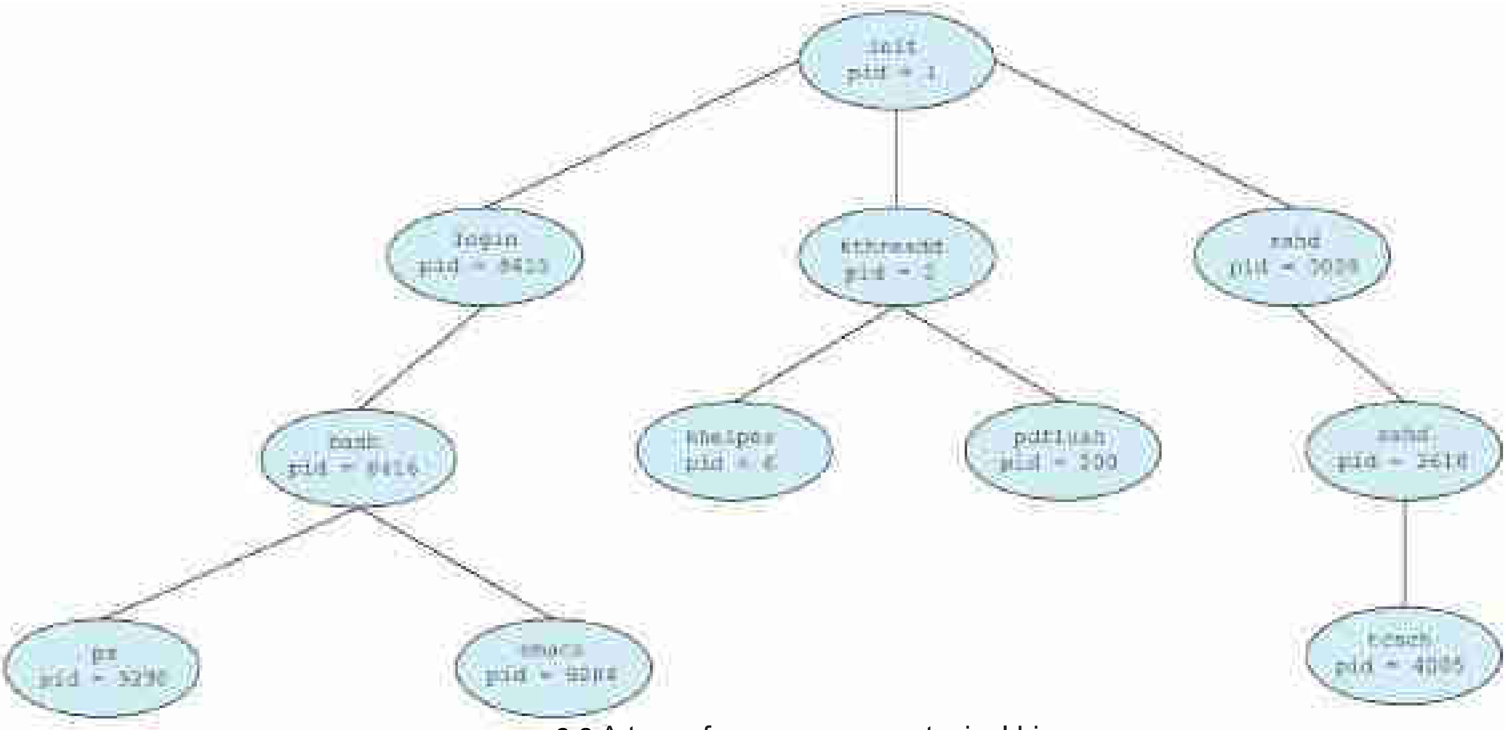
- Process: A process is an activity of executing a program. Basically, it is a program under execution. Every process needs certain resources to complete its task.
- Operation on a Process: The execution of a process is a complex activity. It involves various operations. Following are the operations that are performed while execution of a process.



3.7 Diagram of Process State.

3.3.1 Process Creation

- This is the initial step of process execution activity.
- During the execution, a process may create several new processes.
- The creating process is called a parent process, and the new processes are called the children of that process. Each of these new processes may in turn create other processes, forming a tree of processes.
- Most operating systems (including UNIX, Linux, and Windows) identify processes according to a unique process identifier (or pid), which is typically an integer number. The **pid** provides a unique value for each process in the system, and it can be used as an index to access various attributes of a process within the kernel.



3.8 A tree of processes on a typical Linux

- When a process creates a new process, two possibilities for execution exist:

1. The parent continues to execute concurrently with its children.
2. The parent waits until some or all of its children have terminated.

```
Program: Process creation using fork() system call.

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
pid t pid;
      /* fork a child process */
      pid = fork();
      if (pid < 0)
      {
             /* error occurred */
            fprintf(stderr, "Fork Failed");
            return 1;
      }
      else if (pid == 0)
      {
             /* child process */
            execlp("/bin/ls","ls",NULL);
      }
      else
      {
            /* parent process */
            /* parent will wait for the child to complete */
            wait(NULL);
            printf("Child Complete");
      }
      return 0;
}
```

Process Termination

- A process terminates when operating system executes exit() system call.

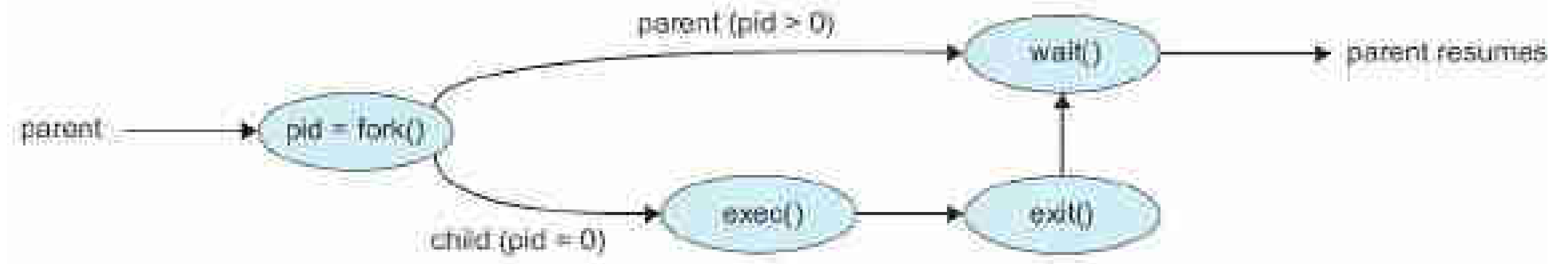
```
/* exit with status 1 */
        exit(1);
```

- At that point, the process may return a status value (typically an integer) to its parent process (via the wait() system call) if its process is not completed.
-

```
pid t pid;
int status;
pid = wait(&status);
```

- All the resources are deallocated when process terminates.
- A parent may terminate the execution of one of its children for a variety of reasons, such as these:

1. The child has exceeded its usage of some of the resources that it has been allocated.
2. The task assigned to the child is no longer required.
3. The parent is exiting, and the operating system does not allow a child to continue if its parent terminates.



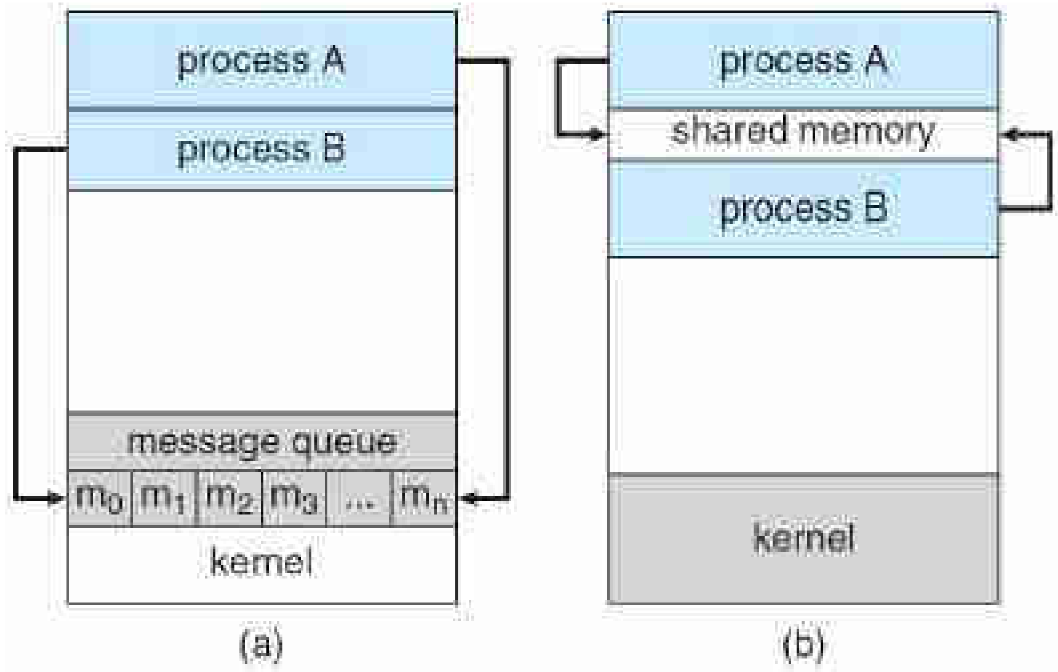
3.9 Process creation using the fork() system call.

3.4 INTERPROCESS COMMUNICATION VVIMP

- Inter process communication is the mechanism provided by the operating system that allows processes to communicate with each other.

Reasons for providing Inter Process Communication:

1. **Information Sharing:** Several processes may share the same information; we must provide an environment to allow concurrent access to same information.
2. **Computation Speedup:** If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others.
3. **Modularity:** Dividing the system functions into separate processes or threads.
4. **Convenience:** Even an individual user may work on many tasks at the same time.



3.10 Communications models. (a) Message passing. (b) Shared memory

3.4.1 Shared Memory Systems

- Shared memory is a memory shared between two or more processes.

PROCEDURE-CONSUMER PROBLEM

- A producer process produces information that is consumed by a consumer process.

```
#define BUFFER SIZE 5
typedef struct {
. . .
}item;
item buffer[BUFFER SIZE];
int in = 0;
int out = 0;
```

- The producer process using shared memory

```
item next produced;
while (true)
{
/* produce an item in next produced */
while (((in + 1) % BUFFER SIZE) == out);
      /* do nothing */
buffer[in] = next produced;
in = (in + 1) % BUFFER SIZE;
}
```

E	4
D	3
C	2
B	1
A	0

in	Buffer [in]	(in + 1) % BUFFER SIZE == OUT	next produced	in	(in+1) % BUFFER SIZE
0	Buffer [0] = A	(0 + 1) % 5 == 0 1 == 0 False	1	1	(1+1) % BUFFER SIZE 2 % 5 = 2
1	Buffer [1] = B	(1 + 1) % 5 == 0 2 == 0 False	2	2	(2+1) % BUFFER SIZE 3 % 5 = 3
2	Buffer [2] = C	(2 + 1) % 5 == 0 3 == 0 False	3	3	(3+1) % BUFFER SIZE 4 % 5 = 4
3	Buffer [3] = D	(3 + 1) % 5 == 0 4 == 0 False	4	4	(4+1) % BUFFER SIZE 5 % 5 = 0
4	Buffer [4] = E	(4 + 1) % 5 == 0 0 == 0 True			

▪ **The consumer process using shared memory**

```
item next consumed;
while (true)
{
while (in == out);
       /* do nothing */
next consumed = buffer [out];
out = (out + 1) % BUFFER SIZE;
/* consume the item in next consumed */
}
```

E	4
D	3
C	2
B	1
A	0

in	out	while (in == out)	Buffer [out]	next consumed	(out+1)%BUFFER SIZE
4	0	4 == 0 False	Buffer [0] =A	1	(0+1)%5 = 1
	1	4 == 1 False	Buffer [1] = B	2	(1+1)%5 = 2
	2	4 == 2 False	Buffer [2] = C	3	(2+1)%5 = 3
	3	4 == 3 False	Buffer [3] = D	4	(3+1)%5 = 4
	4	4 == 4 True	Buffer [4] = E		

3.4.2 Message-Passing Systems

- Message passing model allows multiple processes to read and write data to the message queue without being connected to each other.

- A message passing facility provides at least two operations:

1. `send(message)`
2. `receive(message)`

- There are three methods for logically implementing a link and the send() / receive() operations:

1. Direct or indirect communication
2. Synchronous or asynchronous communication
3. Automatic or explicit buffering

3.4.2.1 Naming

Direct Communication:

- Under direct communication the send() and receive() primitives are defined as:
  `Send (P, message):` `Send a message to process P.`
  `Receive (Q, message):` `Receive a message from process Q.`
- This scheme exhibits **symmetry** in addressing.
- A communication link in this scheme has the following properties:
  1. A link is established automatically between every pair of processes that want to communicate.
  2. A link is associated with exactly two processes.
  3. Between each pair of processes, there exists exactly one link.

- A variant of this scheme employs **asymmetry** in addressing.
- Here, only the sender names the recipient; the recipient is not required to name the sender.
- In this scheme, the send() and receive() primitives are defined as follows:
  `Send (P, message):` `Send a message to process P.`
  `Receive (id, message):` `Receive a message from any process.`
  The variable **id** is set to the name of the process with which communication has taken place.

Indirect Communication:

- With indirect communication, the messages are sent to and received from mailboxes, or ports.
- The send() and receive() primitives are defined as follows:
  `Send (A, message):` `Send a message to mailbox A.`
  `Receive (A, message):` `Receive a message from mailbox A.`

3.4.2.2 Synchronization

- Communication between processes takes place through calls to **send()** and **receive()** primitives. There are different design options for implementing each primitive.
- Message passing may be either **blocking** or **nonblocking** also known as synchronous and asynchronous.
  Blocking send: The sending process is blocked until the message is received by the receiving process or by the mailbox.
  Non blocking send: The sending process sends the message and resumes operation.
  Blocking receive: The receiver blocks until a message is available.
  Non blocking receive: The receiver retrieves either a valid message or a null.

3.4.2.3 Buffering

- Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue.
- Basically, such queues can be implemented in three ways:

1. **Zero Capacity**: The queue has a maximum length of zero; thus, the link cannot have any messages waiting in it. In this case, the sender must block until the recipient receives the message.

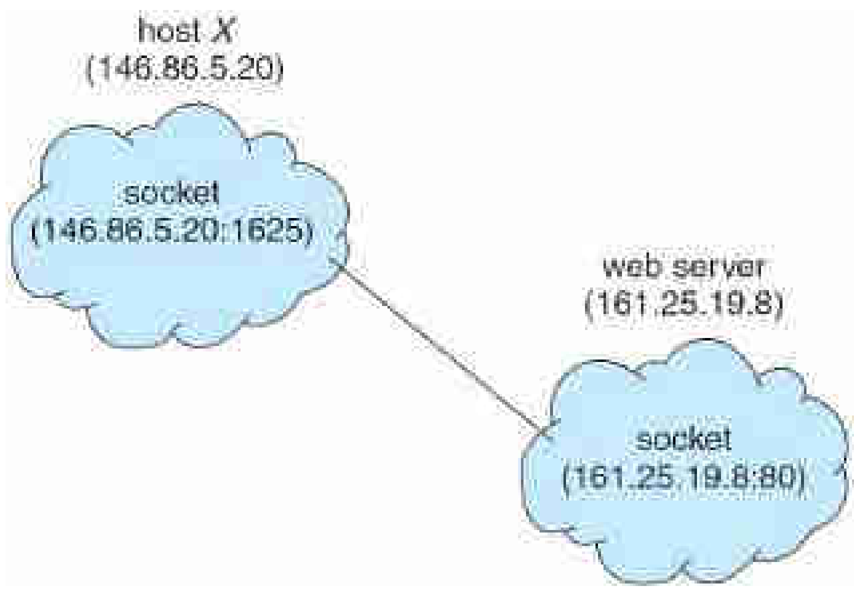
2. **Bounded Capacity:** The queue has finite length n; thus, at most n messages can reside in it. If the queue is not full when a new message is sent, the message is placed in the queue, and the sender can continue execution without waiting. The link's capacity is finite, however. If the link is full, the sender must block until space is available in the queue.
3. **Unbounded Capacity:** The queue's length is potentially infinite; thus, any number of messages can wait in it. The sender never blocks.

3.5 COMMUNICATION IN CLIENT SERVER SYSTEMS

- **Client-server Architecture:** Architecture of a computer network in which many clients (remote processors) request and receive service from a centralized server (host computer).
- In this section, we explore three other strategies for communication in client–server systems: sockets, remote procedure calls (RPCs), and pipes.

3.5.1 Sockets:

- A socket is one endpoint of a two way communication link between two programs running on the network.
- The socket mechanism provides a means of Inter-Process Communication (IPC) by establishing named contact points between which the communication takes place.
- A socket is identified by an **IP Address** concatenated with a **Port Number**.
- In general, sockets use client–server architecture. The server waits for incoming client requests by listening to a specified port.
- Once a request is received, the server accepts a connection from the client socket to complete the connection.



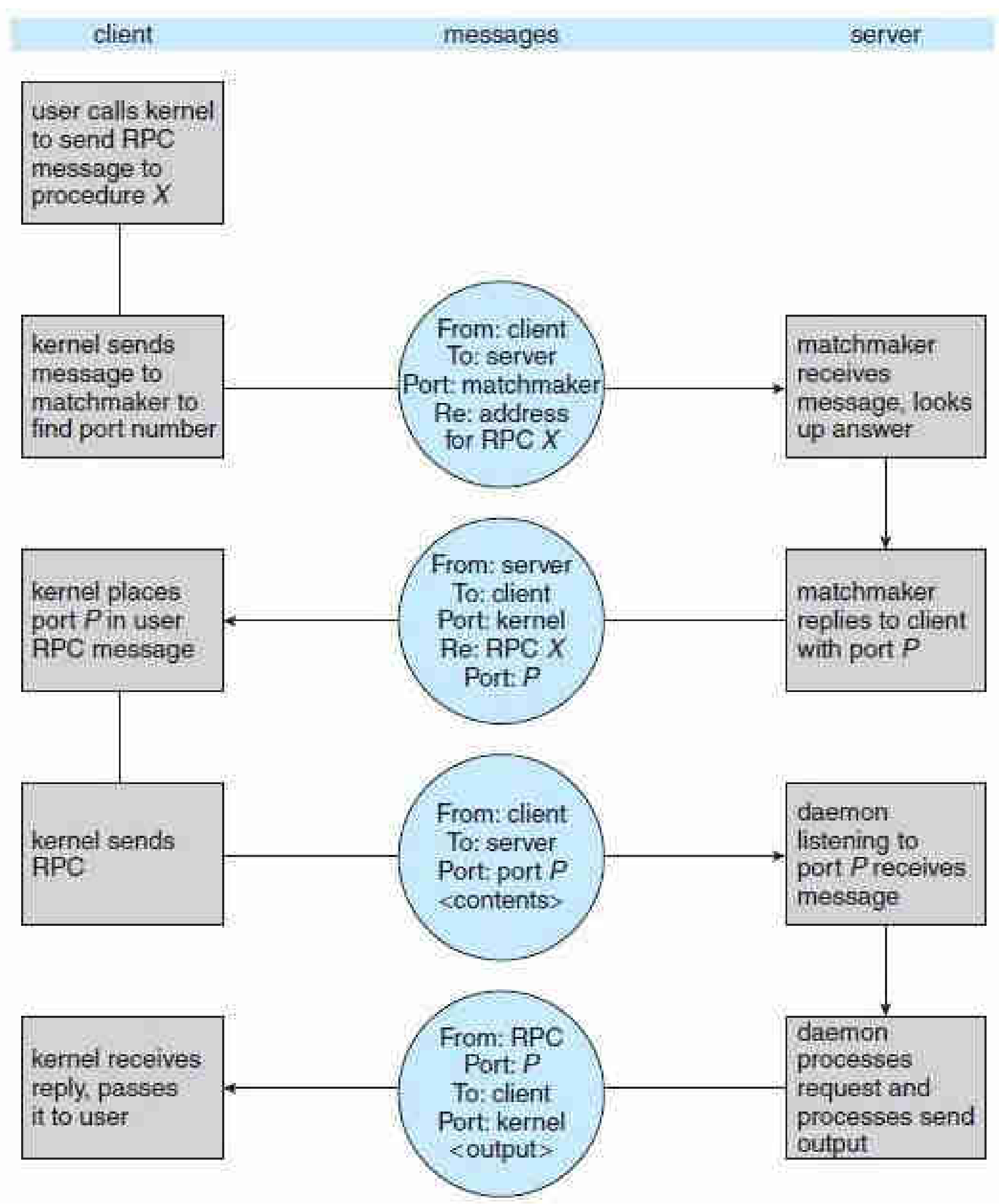
3.12 Communication using sockets.

```
Example: Data Server
```

```
import java.net.*;
import java.io.*;
public class DateServer
{
      public static void main(String[] args)
      {
       Try
        {
        ServerSocket sock = new ServerSocket(6013);

         /* now listen for connections */
         while (true)
         {
          Socket client = sock.accept();
          PrintWriter pout = new PrintWriter(client.getOutputStream(), true);

          /* write the Date to the socket */
          pout.println(new java.util.Date().toString());

          /* close the socket and resume */
          /* listening for connections */
          client.close();
          }
       }
       catch (IOException ioe)
        {
          System.err.println(ioe);
        }
      }
 }
```

3.5.2 Remote Procedure Calls

- A remote procedure call is an inter process communication technique that is used for client-server based applications.
- A client has a request message that the RPC translates and sends to the server. This request may be a procedure or a function call to a remote server.
- When the server receives the request, it sends the required response back to the client. The client is blocked while the server is processing the call and only resumed execution after the server is finished.

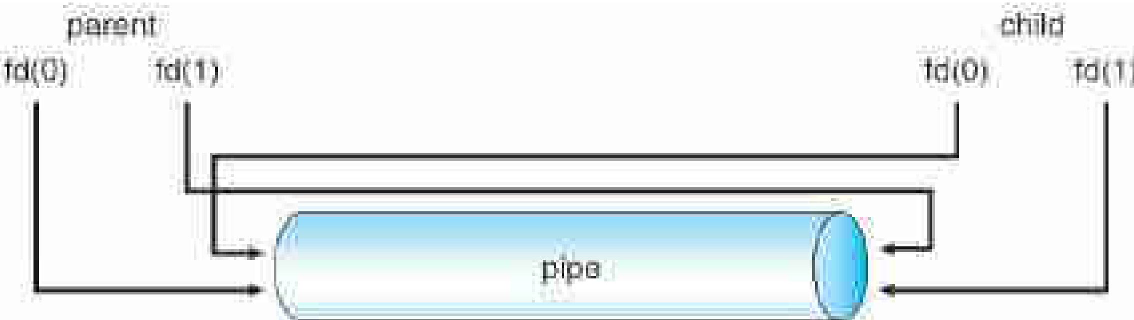


3.13 Execution of a remote procedure call (RPC).

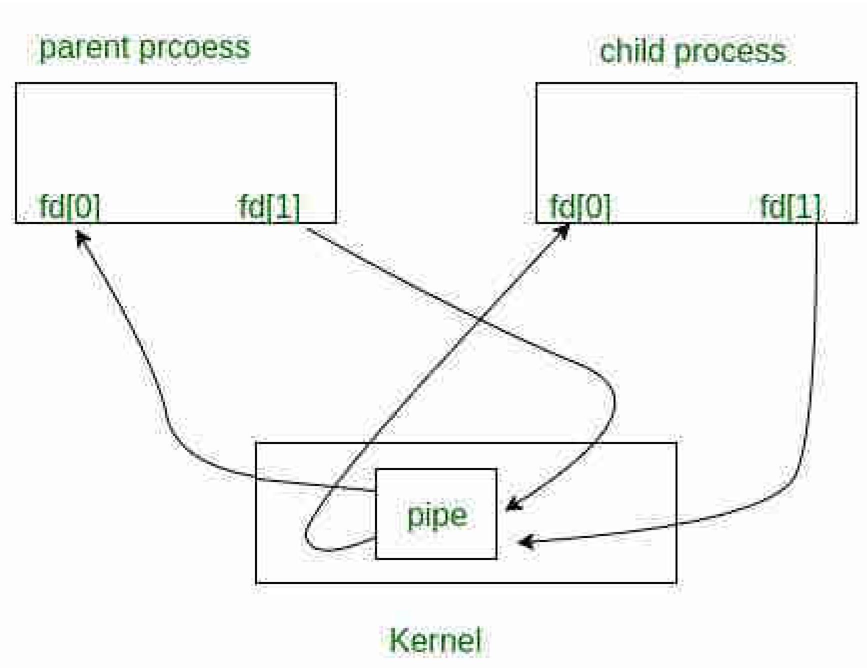
3.5.3 Pipes

- Pipe: Pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process.

- Pipe is one-way communication only i.e we can use a pipe such that one process write to the pipe, and the other process reads from the pipe. It opens a pipe, which is an area of main memory that is treated as a **"virtual file".**
- On UNIX systems, ordinary pipes are constructed using the function **pipe(int fd[])**
- This function creates a pipe that is accessed through the int fd[ ] file descriptors: **fd[0]** is the **read-end** of the pipe, and **fd[1]** is the **write-end**.



3.14 File descriptors for an ordinary pipe.



3.15 File Descriptor for parent-child processes.

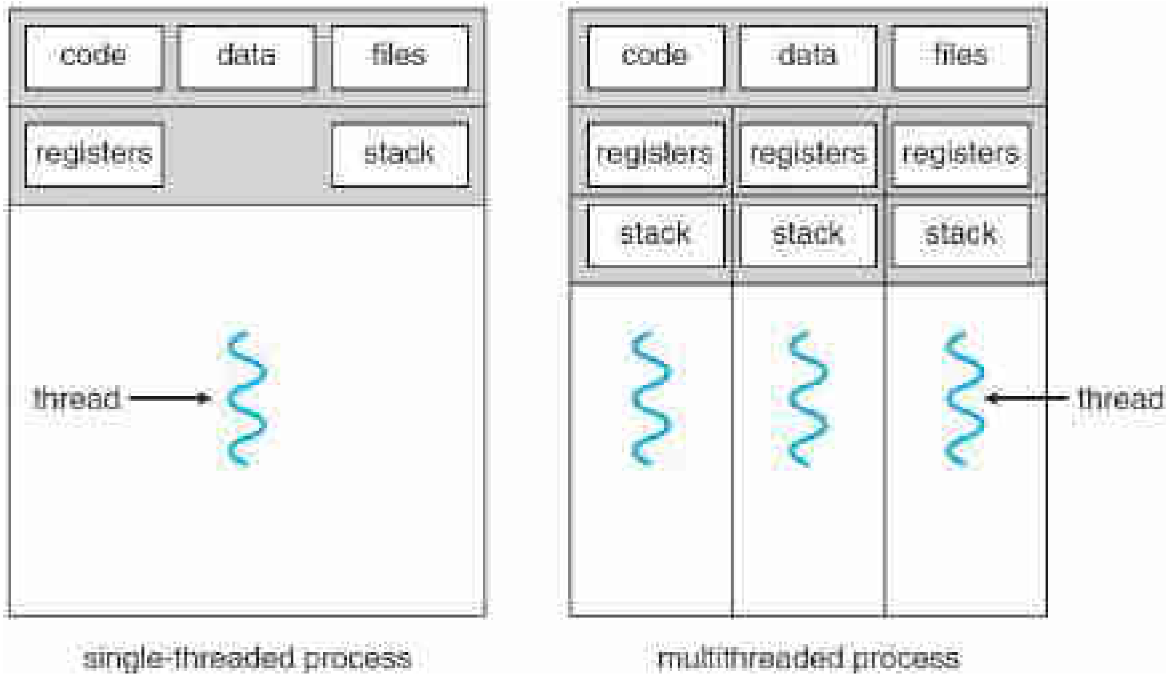
CHAPTER 4
THREADS

4.1 THREAD OVER VIEW

- Thread: A thread is a path of execution within a process.
- A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack.
- It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
- A process can contain multiple threads.

4.1.1 Motivation

- Most software applications that run on modern computers are multithreaded. An application typically is implemented as a separate process with several threads of control.
- A web browser might have one thread display images or text while another thread retrieves data from the network, for example.
- A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.



4.1 Single-threaded and multithreaded processes.

- Finally, most operating-system kernels are now multithreaded. Several threads operate in the kernel, and each thread performs a specific task, such as managing devices, managing memory, or interrupt handling.

4.1.2 Benefits

The benefits of multithreaded programming can be broken down into four major categories

1. **Responsiveness:** If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.
2. **Faster context switch:** Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.
3. **Effective utilization of multiprocessor system:** If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.
4. **Resource sharing:** Resources like code, data, and files can be shared among all threads within a process.
   Note: stack and registers can't be shared among the threads. Each thread has its own stack and registers.
5. **Communication:** Communication between multiple threads is easier, as the threads shares common address space. While in process we have to follow some specific communication technique for communication between two processes.
6. **Economy:** Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads.
7. **Scalability:** Threads may be running in parallel on different processing cores.

4.2 MULTICORE PROGRAMMING

4.2.1 Single Core Systems

- Single Core System: A single core system contains only one processor. So only one process can be executed at a time and then the process is selected from the ready queue. Most general purpose computers contain the single processor systems as they are commonly in use.

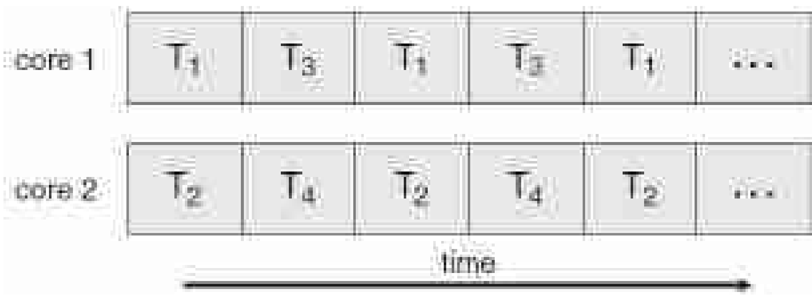
4.2 Concurrent execution on a single-core system.

4.2.2 Multicore Programming

- A processor that has more than one core is called Multicore Processor. Nowadays, most of systems have four cores (Quad-core) or eight cores (Octa-core).

- These cores can individually read and execute program instructions, giving feel like computer system has several processors but in reality, they are cores and not processors.

- Instructions can be calculation, data transferring instruction, branch instruction, etc. Processor can run instructions on separate cores at same time. This increases overall speed of program execution in system. Thus heat generated by processor gets reduced and increases overall speed of execution.



4.3 Parallel execution on a multicore system.

- Multicore systems support Multi Threading and Parallel Computing.

- Multicore processors are widely used across many application domains, including general-purpose, embedded, network, digital signal processing (DSP), and graphics (GPU).

- Efficient software algorithms should be used for implementation of cores to achieve higher performance. Software that can run parallelly is preferred because we want to achieve parallel execution with the help of multiple cores.

Advantages:
- These cores are usually integrated into single IC (integrated circuit) die, or onto multiple dies but in single chip package.Thus allowing higher Cache Coherency.
- These systems are energy efficient since they allow higher performance at lower energy.
- It will have less traffic (cores integrated into single chip and will require less time).

4.2.3 Programming Challenges: In general, five areas present challenges in programming for multicore systems:

- **Identifying tasks:** Tasks are independent of one another and thus can run in parallel on individual cores.

- **Balance:** While identifying tasks that can run in parallel, programmers must also ensure that the tasks perform equal work of equal value.

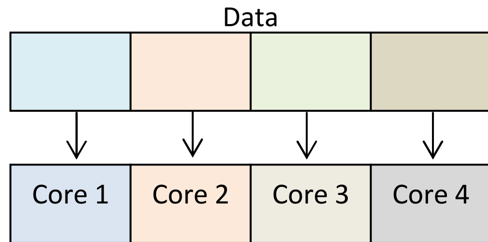
- **Data Splitting:** Just as applications are divided into separate tasks, the data accessed and manipulated by the tasks must be divided to run on separate cores.

- **Data Dependency:** The data accessed by the tasks must be examined for dependencies between two or more tasks. When one task depends on data from another, programmers must ensure that the execution of the tasks is synchronized to accommodate the data dependency.

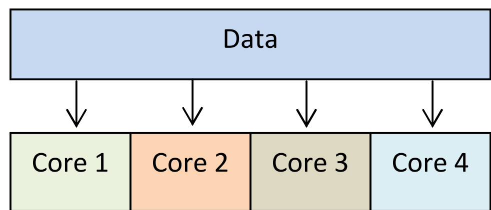
- **Testing and Debugging:** Testing and debugging such concurrent programs is inherently more difficult than testing and debugging single-threaded applications.

4.2.4 Types of Parallelism

- **Data Parallelism:** Data Parallelism means concurrent execution of the same task on each multiple computing core.
- Consider, for example, summing the contents of an array of size N. On a single-core system, one thread would simply sum the elements [0] . . . [N − 1].
- On a single-core system, one thread would simply sum the elements [0] . . . [N − 1]. On a dual-core system, however, thread A, running on core 0, could sum the elements [0] . . . [N/2 − 1] while thread B, running on core 1, could sum the elements [N/2] . . . [N − 1]. The two threads would be running in parallel on separate computing cores.



- **Task Parallelism:** Task Parallelism means concurrent execution of the different task on multiple computing cores.
- Consider again our example above, an example of task parallelism might involve two threads, each performing a unique statistical operation on the array of elements. Again the threads are operating in parallel on separate computing cores, but each is performing a unique operation.

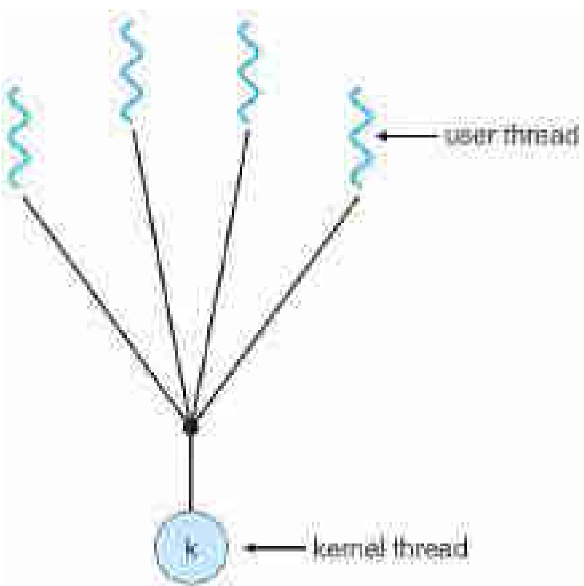


4.3 MULTITHREADING MODELS

- Multi Threading: It is a process of multiple threads executes at same time.
- Many operating systems support kernel thread and user thread in a combined way.
- Multi threading model are of three types.
  Many to many model.
  Many to one model.
  One to one model.

4.3.1 Many-to-One Model

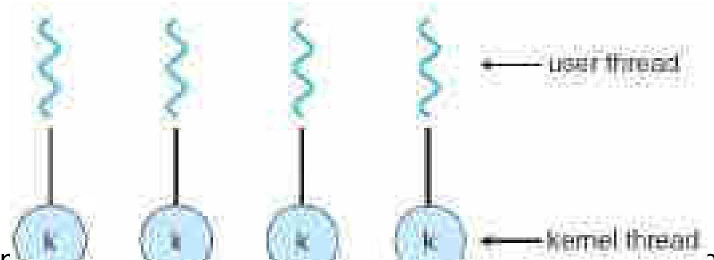
- The many to one model maps many of the user threads to a single kernel thread. This model is quite efficient as the user space manages the thread management.
- A disadvantage of the many to one model is that a thread blocking system call blocks the entire process. Also, multiple threads cannot run in parallel as only one thread can access the kernel at a time.



4.4 Many to one thread model.

4.3.2 One-to-One Model

- The one to one model maps each of the user threads to a kernel thread. This means that many threads can run in parallel on multiprocessors and other threads can run when one thread makes a blocking system call.
- A disadvantage of the one to one model is that the creation of a user thread requires a corresponding kernel thread. Since a lot of kernel threads burden the system, there is restriction on the number of threads in the system.

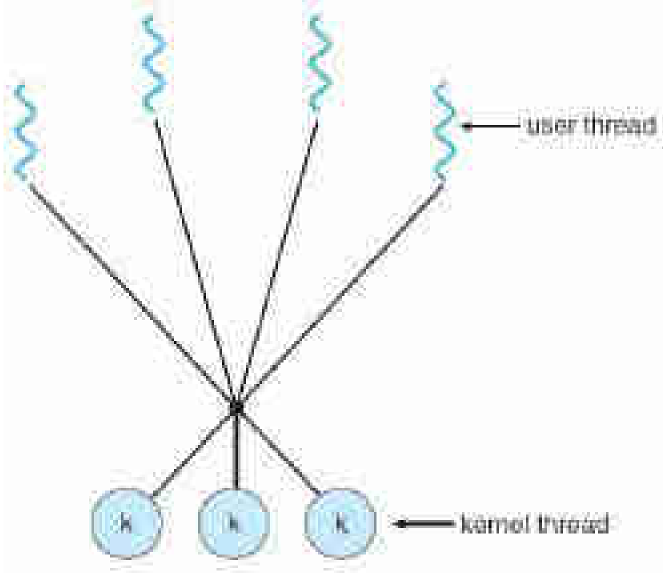


4.5 One to Many thread model.

4.3.3 Many-to-Many Model

- The many to many model r [illegible] esser kernel threads. The number of kernel threads depends on the application or machine.

- The many to many does not have the disadvantages of the one to one model or the many to one model. There can be as many user threads as required and their corresponding kernel threads can run in parallel on a multiprocessor.



4.6 Many to Many Thread Model

4.4 THREAD LIBRARIES

- A thread library provides the programmer an API for creating and managing threads.

- There are two primary ways of implementing a thread library. The first approach is to provide a library entirely in user space with no kernel support.

- All code and data structures for the library exist in user space. This means that invoking a function in the library results in a local function call in user space and not a system call.
- The second approach is to implement a kernel-level library supported directly by the operating system. In this case, code and data structures for the library exist in kernel space.

- Invoking a function in the API for the library typically results in a system call to the kernel.

- Three main thread libraries are in use today:
  1. Pthreads
  2. Windows Threads
  3. Java Threads

Example of Pthread: A multithreaded program that calculates the summation of a nonnegative integer in a separate thread.

```
#include <pthread.h>
#include <stdio.h>
int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */
int main(int argc, char *argv [])
{
Pthread_t tid; /* the thread identifier */
Pthread_attr_t attr; /* set of thread attributes */
if (argc != 2)
{
fprintf(stderr,"usage: a.out <integer value>\n");
return -1;
}
if (atoi(argv[1]) < 0)
{
fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
return -1;
}
/* get the default attributes */
pthread_attr_init(&attr);
/* create the thread */
pthread_create(&tid,&attr,runner,argv[1]);
/* wait for the thread to exit */
pthread_join(tid,NULL);
printf("sum = %d\n",sum);
}
/* The thread will begin control in this function */
void *runner(void *param)
{
int i, upper = atoi(param);
sum = 0;
for (i = 1; i <= upper; i++)
```

```
sum += i;

pthread exit(0);

}
```

Example of Windows Thread: A multithreaded program that calculates the summation of a nonnegative integer in a separate thread.

```
#include <windows.h>
#include <stdio.h>

DWORD Sum; /* data is shared by the thread(s) */

/* the thread runs in this separate function */
DWORD WINAPI Summation(LPVOID Param)
{
DWORD Upper = *(DWORD*)Param;
for (DWORD i = 0; i <= Upper; i++)
Sum += i;
return 0;
}

int main(int argc, char *argv[])
{
DWORD ThreadId;
HANDLE ThreadHandle;
int Param;
if (argc != 2) { fprintf(stderr,"An integer parameter is required\n");
return -1;
}
Param = atoi(argv[1]);
if (Param < 0)
{ fprintf(stderr,"An integer >= 0 is required\n");
return -1;
}

/* create the thread */
ThreadHandle = CreateThread(
NULL, /* default security attributes */
0, /* default stack size */
Summation, /* thread function */
&Param, /* parameter to thread function */
0, /* default creation flags */
&ThreadId); /* returns the thread identifier */

if (ThreadHandle != NULL)
{
/* now wait for the thread to finish */
WaitForSingleObject(ThreadHandle,INFINITE);
```

```
/* close the thread handle */
CloseHandle(ThreadHandle);
printf("sum = %d\n",Sum);
}
}
```

Example of Java Thread: A multithreaded program that calculates the summation of a nonnegative integer in a separate thread.

```
class Sum
{
private int sum;
public int getSum() { return sum;
}
public void setSum(int sum) { this.sum = sum;
}
}

class Summation implements Runnable
{
private int upper;
private Sum sumValue;

public Summation(int upper, Sum sumValue)
{
this.upper = upper;
this.sumValue = sumValue;
}

public void run()
{
int sum = 0;
for (int i = 0; i <= upper; i++)
sum += i;
sumValue.setSum(sum);
}
}

public class Driver
{
public static void main(String[] args)
{
if (args.length > 0)
{
if (Integer.parseInt(args[0]) < 0)
System.err.println(args[0] + " must be >= 0.");
```

```
Else
{
Sum sumObject = new Sum();
int upper = Integer.parseInt(args[0]);
Thread thrd = new Thread(new Summation(upper, sumObject));
thrd.start();
try
{
thrd.join();
System.out.println
("The sum of "+upper+" is "+sumObject.getSum());
} catch (InterruptedException ie) { }
}
}
else
System.err.println("Usage: Summation <integer value>");
}
}
```

CHAPTER 5
PROCESS
SYNCHRONIZATION

5.1 PROCESS SYNCHRONIZATION

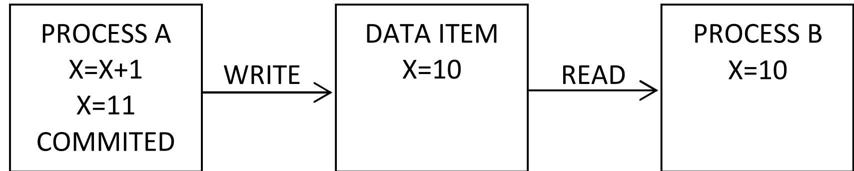
- **Process Synchronization:** Process Synchronization is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.

- It is specially needed in a multi-process system when multiple processes are running together, and more than one processes try to gain access to the same shared resource or data at the same time.

- This can lead to the inconsistency of shared data. So the change made by one process not necessarily reflected when other processes accessed the same shared data. To avoid this type of inconsistency of data, the processes need to be synchronized with each other.

5.1.1 How Process Synchronization Works?

- For Example, process A changing the data in a memory location while another process B is trying to read the data from the **same memory location**. There is a high probability that data read by the second process will be erroneous.



Example: Producer-Consumer Problem.

- A producer process produces information that is consumed by a consumer process.

```
#define BUFFER SIZE 5
typedef struct {
. . .
}item;
item buffer[BUFFER SIZE];
int in = 0;
int out = 0;
```

- The producer process using shared memory

```
while (true)
{
/* produce an item in next produced */
while (((in + 1) % BUFFER SIZE) == out);
      /* do nothing */
buffer[in] = next produced;
in = (in + 1) % BUFFER SIZE;
}
```

Item	Index
E	4
D	3
C	2
B	1
A	0

in	Buffer [in]	(in + 1) % BUFFER SIZE == OUT	next produced	in	(in+1) % BUFFER SIZE
0	Buffer [0] = A	(0 + 1) % 5 == 0 1 == 0 False	1	1	(1+1) % BUFFER SIZE 2 % 5 = 2
1	Buffer [1] = B	(1 + 1) % 5 == 0 2 == 0 False	2	2	(2+1) % BUFFER SIZE 3 % 5 = 3
2	Buffer [2] = C	(2 + 1) % 5 == 0 3 == 0 False	3	3	(3+1) % BUFFER SIZE 4 % 5 = 4
3	Buffer [3] = D	(3 + 1) % 5 == 0 4 == 0 False	4	4	(4+1) % BUFFER SIZE 5 % 5 = 0
4	Buffer [4] = E	(4 + 1) % 5 == 0 0 == 0 True			

- **The consumer process using shared memory**

```
while (true)
{
while (in == out);
       /* do nothing */
next consumed = buffer [out];
out = (out + 1) % BUFFER SIZE;
/* consume the item in next consumed */
}
```

Item	Index
E	4
D	3
C	2
B	1
A	0

in	out	while (in == out)	Buffer [out]	next consumed	(out+1)%BUFFER SIZE
4	0	4 == 0 False	Buffer [0] =A	1	(0+1)%5 = 1
	1	4 == 1 False	Buffer [1] = B	2	(1+1)%5 = 2
	2	4 == 2 False	Buffer [2] = C	3	(2+1)%5 = 3
	3	4 == 3 False	Buffer [3] = D	4	(3+1)%5 = 4
	4	4 == 4 True	Buffer [4] = E		

- **The statement "counter++" may be implemented follows:**

  register1 = counter

  register1 = register1 + 1

  counter = register1

- **Similarly, the statement "counter--" is implemented as follows:**

  register2 = counter

  register2 = register2 – 1

  counter = register2

- The in interleaving of **counter++** and **counter--** produces data inconsistency:

Step	Result	Counter	Register
T0: producer execute register1 = counter	**{register1 = 5}**	Counter: 5	Register 1: 5
T1: producer execute register1 = register1 + 1	**{register1 = 6}**		Register 1: 6
T2: consumer execute register2 = counter	**{register2 = 5}**		Register 2: 5
T3: consumer execute register2 = register2 – 1	**{register2 = 4}**		Register 2: 4
T4: producer execute counter = register1	**{counter = 6}**	Counter: 6	
T5: consumer execute counter = register2	**{counter = 4}**	Counter: 4	

- Finally producer process updates Register 1 as 6 and Counter as 6. Consumer process updates Register 2 as 4 and Counter as 4.

- **Race Condition:** A race condition occurs when several processes access a shared variable at the same time.

5.1.2 Process Synchronization using Critical Section

- **Critical Section:** Critical section is a segment of code that share common resources by more than one process.

- Each process request permission to enter its Critical Section.
- The critical section has three components
  1. Entry section: The section of code implementing process request to enter its critical section.
  2. Exit section: The critical section may be followed by an exit section.
  3. Remainder section: The remaining code is the remainder section.

General structure of a typical process *Pi*:

```
do
{
 [entry section]

  critical section

 [exit  section]

  remainder section

} while (true);
```

A solution to the critical-section problem must satisfy the following three requirements:

1. **Mutual exclusion:** If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.

2. **Progress:** If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.

3. **Bounded waiting:** There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

- Two general approaches are used to handle critical sections on operating systems

1. Preemptive Kernel: A preemptive kernel is where the kernel allows a process to be removed and replaced while it is running in kernel mode.

2. Non-Preemptive Kernel: A non preemptive kernel does not allow a process running in kernel mode to be preempted; a kernel-mode process will run until it exits kernel mode, blocks, or voluntarily yields control of the CPU.

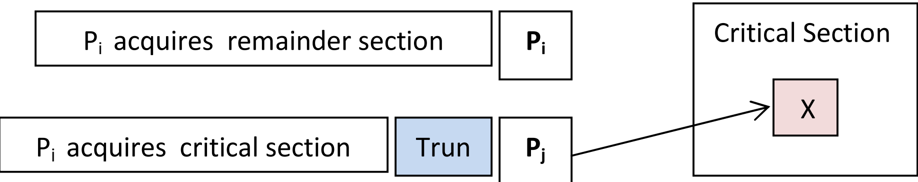
5.2 PETERSON'S SOLUTION

- It provides a good algorithmic description of solving the critical-section problem.

- Peterson's solution is restricted to two processes that alternate execution between their critical sections and remainder sections. The processes are numbered P0 and P1.

The structure of process Pi in Peterson's solution:

```
do
{
 flag[i] = true;
 turn = j;
 while (flag[j] && turn == j);

      critical section

 flag[i] = false;

      remainder section

} while (true);
```



- Peterson's solution proves
1. Mutual exclusion is preserved.
2. The progress requirement is satisfied.
3. **3.** The bounded-waiting requirement is met.

5.3 SYNCHRONIZATION HARDWARE (TEST AND SET, SWAP, UNLOCK AND LOCK)

- Process Synchronization problems occur when two processes running concurrently share the same data or same variable.

- The value of that variable may not be updated correctly before its being used by a second process. Such a condition is known as Race Around Condition.

- There are three algorithms in the hardware approach of solving Process Synchronization problem:

1. Test and Set Lock
2. Swap
3. Unlock and Lock

5.3.1 Test and Set Lock :

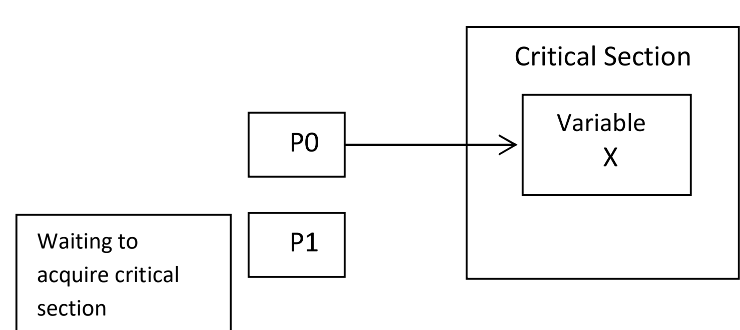
- A hardware solution to the synchronization problem.
- There is a shared link variable which can take either of the two values, 0 or 1.
- Before entering into the critical section, a process inquires about the lock.
- If it is locked, it keeps on waiting till it becomes free.
- If it is not locked, it takes the lock and executes the critical section.

The definition of the test and set() instruction.

```
boolean test and set(boolean *target)
{
boolean rv = *target;
*target = true;
return rv;
}
```

Mutual-exclusion implementation with test and set().

```
do
{
while (test and set(&lock));
/* do nothing */
/* critical section */
lock = false;
/* remainder section */
} while (true);
```



CHAPTER 6
CPU SCHEDULING

6.1 BASIC CONCEPTS

- In a single-processor system, only one process can run at a time. Others must wait until the CPU is free and can be rescheduled.
- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.

6.1.1 CPU - I/O Burst Cycle

- Process execution consists of a cycle of CPU execution and I/O wait. The state of process under execution is called CPU burst and the state of process under I/O request & its handling is called I/O burst.
- Processes alternate between these two states. Process execution begins with a CPU burst. That is followed by an I/O burst, which is followed by another CPU burst, then another I/O burst, and so on.

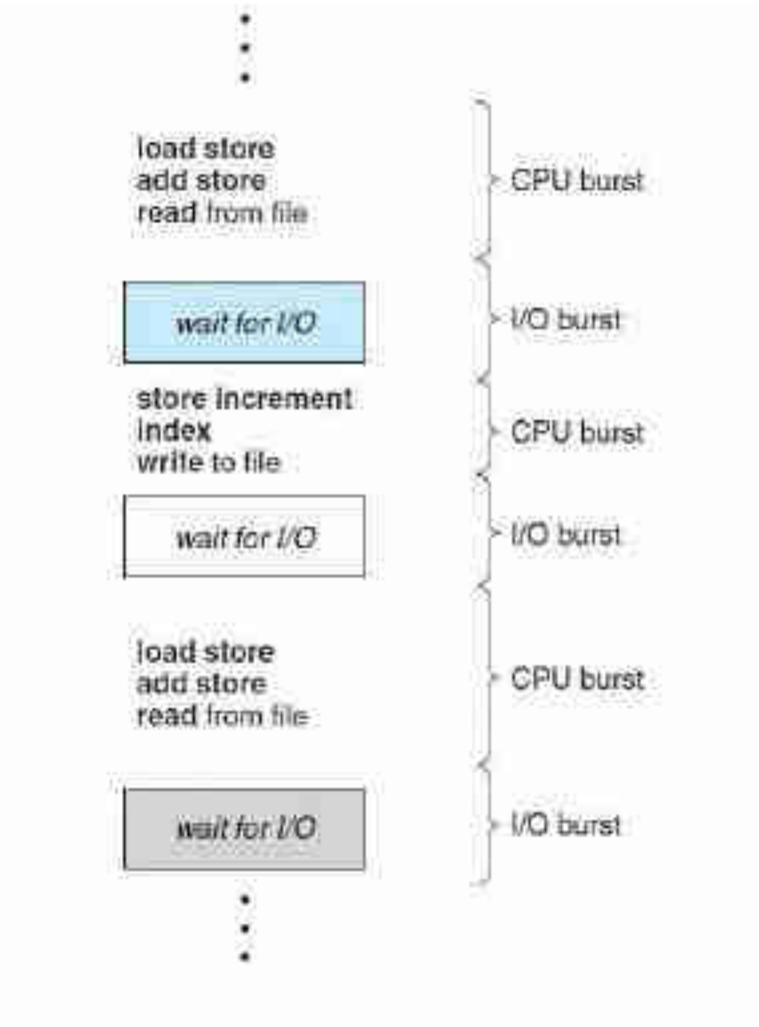


Figure 6.1 Alternating sequence of CPU and I/O bursts.

6.1.2 CPU Scheduler

- CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them. S
- Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

6.1.3 Preemptive Scheduling IMP 2M

- Preemptive Scheduling is a CPU scheduling technique that works by dividing time slots of CPU to a given process.
- The time slot given might be able to complete the whole process or might not be able to it. When the burst time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance. This scheduling is used when the process switch to ready state.

6.1.4 Dispatcher

- The dispatcher is the module that gives a process control over the CPU after it has been selected by the short-term scheduler.

6.2 SCHEDULING CRITERIA

- Different CPU Scheduling Algorithms have different properties, and the choice of a particular algorithm depends on the processor.
- Many criteria have been suggested for comparing CPU scheduling algorithms. The criteria include the following.

1. **CPU Utilization:** The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilisation can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system.
2. **Throughput**: Number of jobs completed in a unit of time. The throughput may vary depending upon the length or duration of processes.
3. **Turnaround Time:** Turnaround time may simply deal with the total time it takes for a program to provide the required output. Turn-around time is the sum of times spent waiting to get into memory, waiting in ready queue, executing in CPU, and waiting for I/O.
4. **Waiting Time:** Waiting time is the total time spent by the process in the ready queue.

5. **Response Time:** Time taken from submission of the process of request until the first response is produced. This measure is called response time.

6.3 SCHEDULING ALGORITHMS VVIMP

6.3.1 First Come First Serve

- First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

Example 1: Demonstrate First-Come, First-Served Scheduling with the following data

Process	Burst Time
P1	24
P2	3
P3	3

Solution: If the processes arrive in the order P_1, P_2, P_3, and are served in FCFS order,

- we get the result shown in the following **Gantt chart**, which is a bar chart that
- Illustrates a particular schedule, including the start and finish times of each of the participating processes.

- The waiting time is 0 milliseconds for process P_1, 24 milliseconds for process P_2, and 27 milliseconds for process P_3. Thus, the average waiting time is (0+ 24 + 27)/3 = 17 milliseconds.

Example 2: Demonstrate First-Come, First-Served Scheduling with the following data

Process	Burst Time
P2	3
P3	3
P1	24

The average waiting time is now (6 + 0 + 3)/3 = 3 milliseconds.

6.3.2 Shortest-Job-First Scheduling:

- Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next.
- Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
- It is a Greedy Algorithm.
- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.
- It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.

Example 1: Using SJF Algorithm draw the Gantt Chart using the following Processes and their Burst Times.

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

- The waiting time is 3 milliseconds for process P_1, 16milliseconds for processP_2, 9milliseconds for process P_3, and 0milliseconds for process P_4. Thus, the average waiting time is (3 + 16 + 9 + 0)/4 = **7 milliseconds.**
- The SJF algorithm can be either preemptive or nonpreemptive.
- Preemptive SJF scheduling is sometimes called shortest-remaining-time-first scheduling.

Example2: Using Shortest-Remaining-Time-First Scheduling Algorithm draw the Gantt Chart using the following Processes and their Arrival Time and Burst Time.

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

- Process P_1 is started at time 0, since it is the only process in the queue. ProcessP_2 arrives at time 1. The remaining time for process P_1 (7 milliseconds) is larger than the time required by process P_2 (4 milliseconds), so process P_1 is preempted, and process P_2 is scheduled.

- The average waiting time for this example is [(10 − 1) + (1 − 1) + (17 − 2) + (5 − 3)]/4 = 26/4 = 6.5 milliseconds.

6.3.2 Priority Scheduling:

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on. Processes with same priority are executed on first come first served basis.
- Processes with same priority are executed on first come first served basis.

Example1: Using Priority-Scheduling Scheduling Algorithm draw the Gantt Chart using the following Processes and their Burst Time and Priority.

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

The average waiting time is (0+1+6+16+18) / 5 = 8.2 milliseconds

- A major problem with priority scheduling algorithms is indefinite blocking, or starvation. A process that is ready to run but waiting for the CPU can be considered blocked. A priority scheduling algorithm can leave some low priority processes waiting indefinitely. In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

- A solution to the problem of indefinite blockage of low-priority processes is aging. Aging involves gradually increasing the priority of processes that wait in the system for a long time.

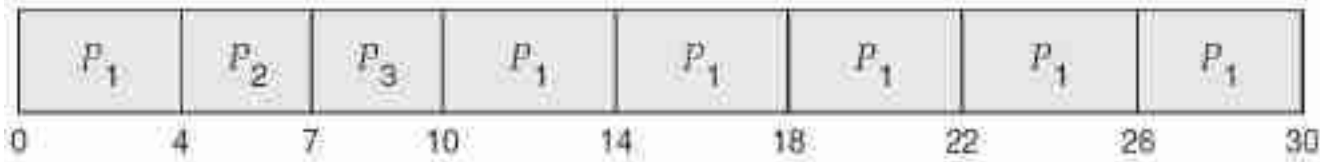
6.3.3 Round-Robin Scheduling:

- Round Robin is a CPU Scheduling Algorithm where each process is assigned a fixed time slot in a cyclic way.
- A small unit of time, called a time quantum or time slice, is defined. A time quantum is generally from10 to 100 milliseconds in length. The ready queue is treated as a circular queue.
- The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.
- It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
- One of the most commonly used technique in CPU scheduling as a core.
- It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
- The disadvantage of it is more overhead of context switching.

Example1: Using Round-Robin Scheduling Algorithm draw the Gantt Chart using the following Processes and their Burst Time and with a time quantum of 4 milliseconds

Process	Burst Time
P1	24
P2	3
P3	3

- If we use a time quantum of 4 milliseconds, then process P1 gets the first 4 milliseconds. Since it requires another 20 milliseconds, it is preempted after the first time quantum, and the CPU is given to the next process in the queue, process P2. Process P2 does not need 4 milliseconds, so it quits before its time quantum expires. The CPU is then given to the next process, process P3. Once each process has received 1 time quantum, the CPU is returned to process P1 for an additional time quantum.



- Let's calculate the average waiting time for this schedule. P1 waits for 6 milliseconds (10 - 4), P2 waits for 4 milliseconds, and P3 waits for 7 milliseconds. Thus, the average waiting time is 17/3 = 5.66 milliseconds.

  Average waiting time (10-4) + 4 + 7 = 17/3 = 5.66 milliseconds.

6.3.4 Multilevel Queue Scheduling

- A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

- For example, a common division is a **foreground (interactive)** process and a **background (batch)** process. These two classes have different scheduling needs. For this kind of situation Multilevel Queue Scheduling is used.

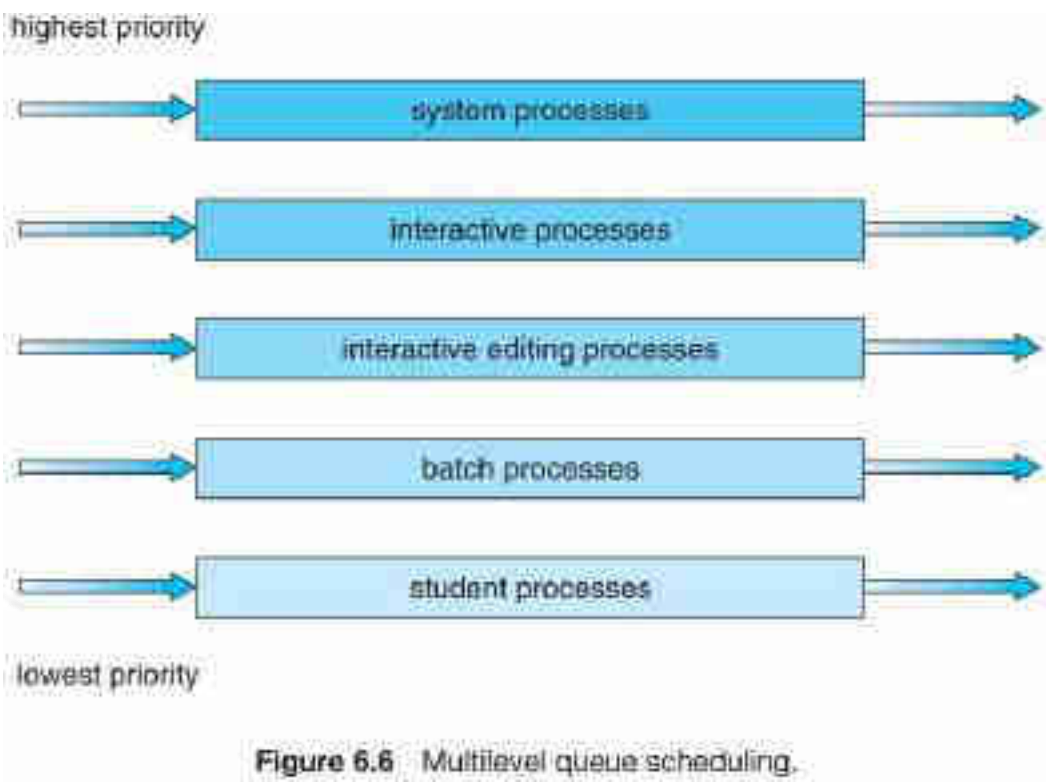


Figure 6.6 Multilevel queue scheduling.

The Description of the processes in the above diagram is as follows:

- **System Process** The Operating system itself has its own process to run and is termed as System Process.
- **Interactive Process** The Interactive Process is a process in which there should be the same kind of interaction (basically an online game).
- **Batch Processes** Batch processing is basically a technique in the Operating system that collects the programs and data together in the form of the **batch** before the **processing** starts. (Payroll System, Bank Statements)
- **Student Process** The system process always gets the highest priority while the student processes always get the lowest priority.

In an operating system, there are many processes, in order to obtain the result we cannot put all processes in a queue; thus this process is solved by Multilevel Queue Scheduling.

6.3.5 Multilevel Feedback Queue Scheduling

- In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system.
- Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.
- Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.

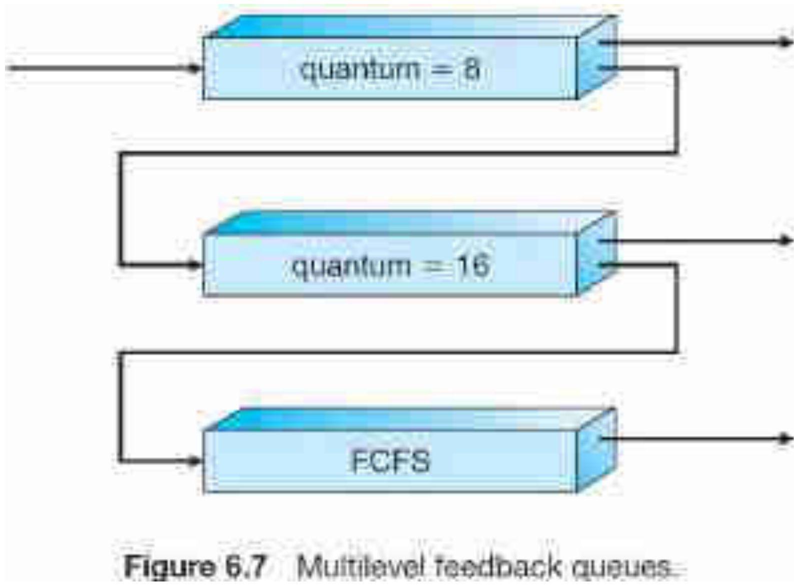


Figure 6.7 Multilevel feedback queues.

Explanation:

- First of all, **Suppose that queues 1 and 2 follow round robin** with time quantum 8 and 16 respectively and queue 3 follows FCFS.
- One of the implementations of Multilevel Feedback Queue Scheduling is as follows:

1. A process entering the ready queue is put in queue 0.
2. A process in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time, it is moved to the tail of queue 1.
3. If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2.
4. Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.

- In general, a multilevel feedback queue scheduler is defined by the following parameters:

1. The number of queues.
2. The scheduling algorithm for each queue.
3. The method used to determine when to upgrade a process to a higher priority queue.
4. The method used to determine when to demote a process to a lower priority queue.
5. The method used to determine which queue a process will enter when that process needs service.

Advantages of MFQS

1. This is a flexible Scheduling Algorithm
2. This scheduling algorithm allows different processes to move between different queues.
3. In this algorithm, A process that waits too long in a lower priority queue may be moved to a higher priority queue which helps in preventing starvation.

Disadvantages of MFQS

1. This algorithm is too complex.
2. As processes are moving around different queues which leads to the production of more CPU overheads.
3. In order to select the best scheduler this algorithm requires some other means to select the values.

6.4 MULTIPLE PROCESSOR SCHEDULING

- In multiple-processor scheduling **multiple CPU's** are available and hence **load sharing** becomes possible. However multiple processor scheduling is more complex as compared to single processor scheduling. In multiple processor scheduling there are cases when the processors are identical i.e. HOMOGENEOUS, in terms of their functionality, we can use any processor available to run any process in the queue.

6.4.1 Approaches to Multiple-Processor Scheduling

- Asymmetric Multiprocessing system: Asymmetric Multiprocessing has the master-slave relationship among the processors. There is one master processor that controls remaining slave processor. The master processor allots processes to slave processor, or they may have some predefined task to perform. The master processor runs the tasks of the operating system.

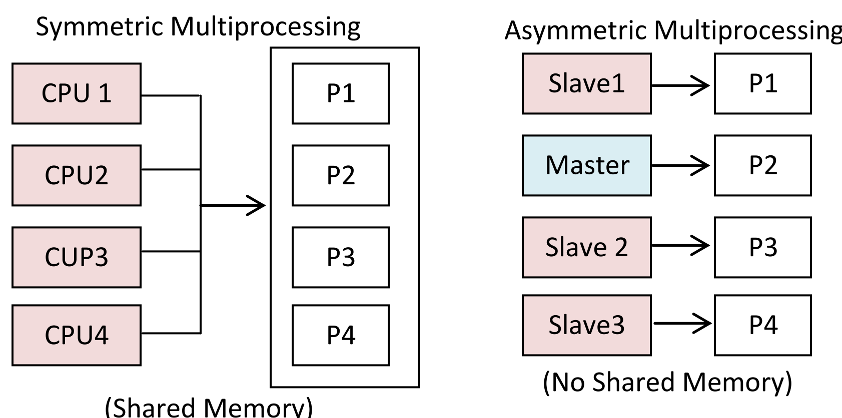


Fig 6.8: Symmetric and Asymmetric Multiprocessing.

- Symmetric Multiprocessing system: In symmetric multiprocessing, all the processors are treated equally. All processors communicate with another processor by a shared memory.

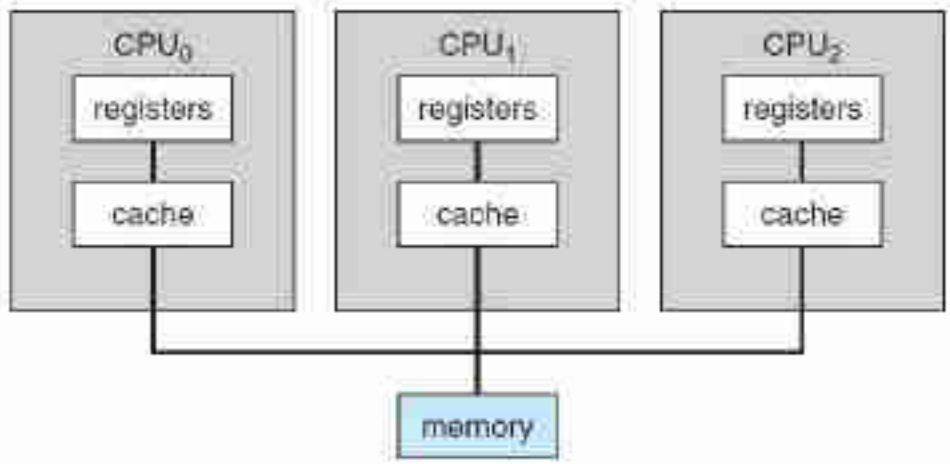


Fig 6.9.1 : Symmetric multiprocessing architecture.

- A recent trend in CPU design is to include multiple computing cores on a single chip. Such multiprocessor systems are termed multicore.

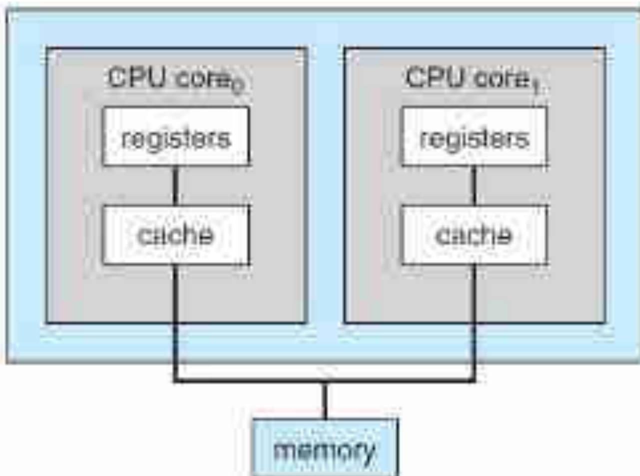


Fig 6.9.2: A dual-core design with two cores placed on the same chip.

6.4.2 Processor Affinity

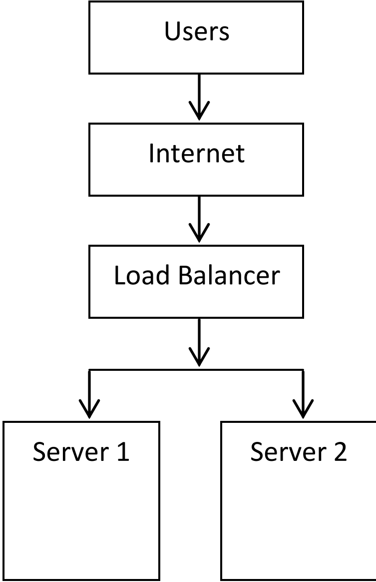
- **Processor affinity** enables the binding and unbinding of a process or a thread to a central processing unit (CPU) or a range of CPUs, so that the process or thread will execute only on the designated CPU or CPUs rather than any CPU.

- **Soft Affinity:** Processes are allowed to move between the Processors.

- **Hard Afinity:** Processes are allowed to move between the Processors. Linux and some other Operating Systems support hard affinity.

6.4.3 Load Balancing 2 MARKS IMP

- Load balancing is defined as the methodical and efficient distribution of network or application traffic across multiple servers.



- There are two general approaches to load balancing: push migration and pull migration.

- There are two general approaches to load balancing:

1. **Push Migration:** In push migration a task routinely checks the load on each processor and if it finds an imbalance then it evenly distributes load on each processors by moving the processes from overloaded to idle or less busy processors.
2. **Pull Migration:** Pull migration occurs when an idle processor pulls a waiting task from a busy processor for its execution.

6.4.4 Multicore Processors:

- In multicore processors **multiple processor** cores are places on the same physical chip. Each core has a register set to maintain its architectural state and thus appears to the operating system as a separate physical processor. **SMP systems** that use multicore processors are faster and consume **less power** than systems in which each processor has its own physical chip.

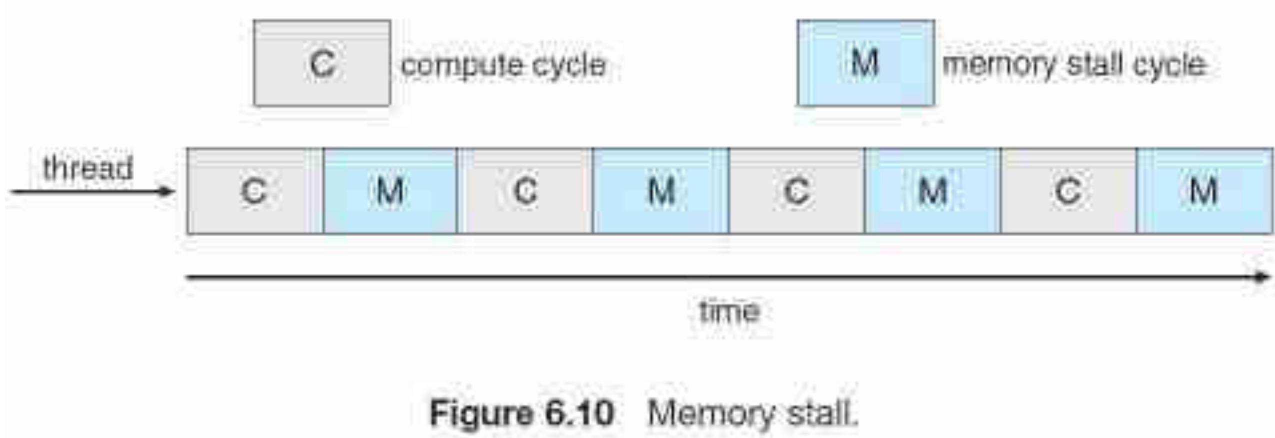


Figure 6.10 Memory stall.

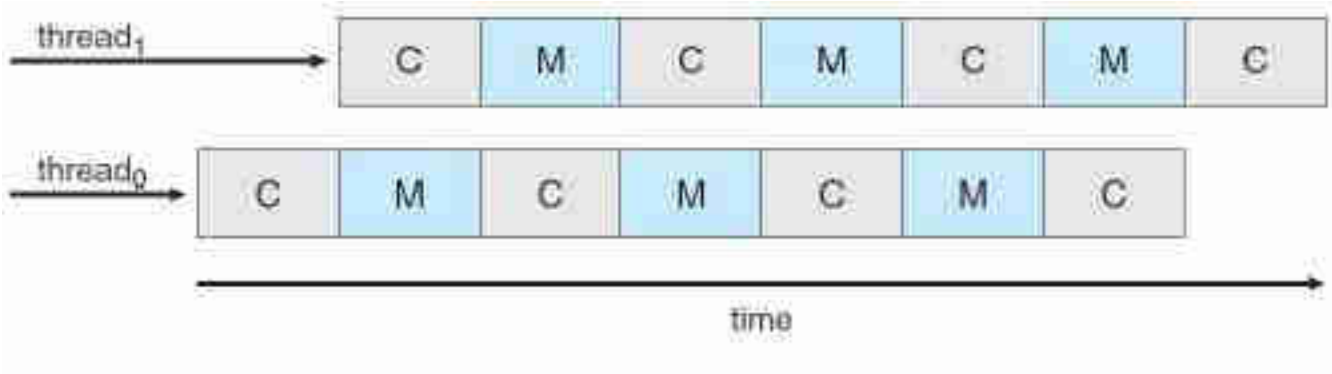


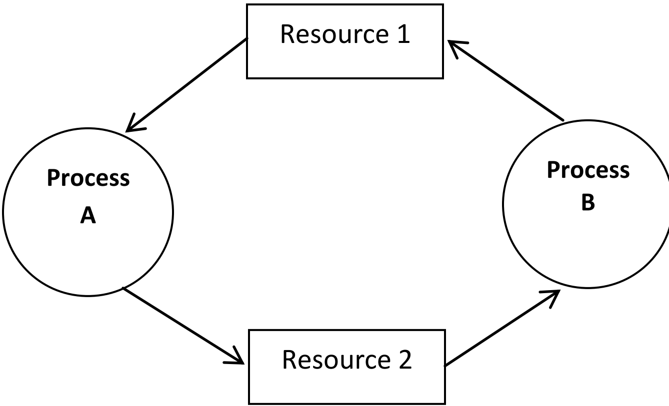
Figure 6.11 Multithreaded multicore system.

CHAPTER 7
DEADLOCKS

7.1 SYSTEM MODEL

- **Dead Lock:** Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

- For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



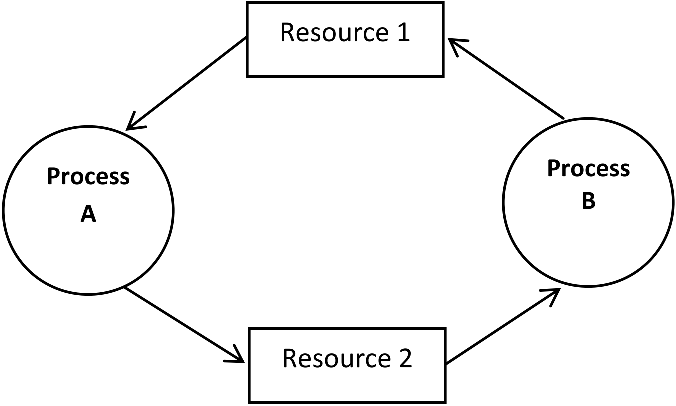
Under the normal mode of operation, a process may utilize a resource in only the following sequence:

- **Request:** The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
- **Use:** The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
- **Release:** The process releases the resource.

7.2 DEADLOCK CHARACTERIZATION

- **Dead Lock:** Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

- For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



7.2.1 Necessary Conditions VVIMP

Deadlock can arise if the following four conditions hold simultaneously (Necessary Conditions)

1. **Mutual Exclusion:** One or more than one resource are non-shareable (Only one process can use at a time).
2. **Hold and Wait:** A process is holding at least one resource and waiting for resources.
3. **No Preemption:** A resource cannot be taken from a process unless the process releases the resource.
4. **Circular Wait:** A set of processes are waiting for each other in circular form.

7.2.2 Resource-Allocation Graph

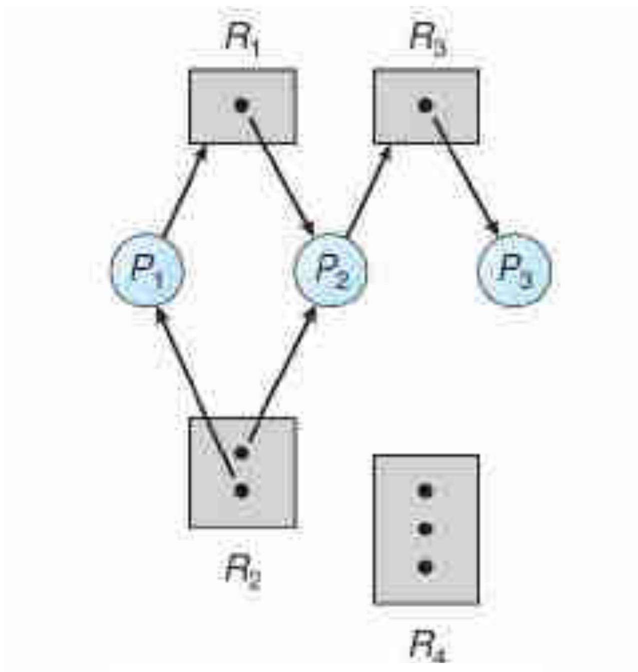
- Deadlocks can be described more precisely in terms of a directed graph called a system resource-allocation graph.

- This graph consists of a set of vertices V and a set of edges E. The set of vertices V is partitioned into two different types of nodes P = {P1, P2, ..., Pn}, the set consisting of all the active processes in the system, and R = {R1, R2, ..., Rm}, the set consisting of all resource types in the system.

- A directed edge from process Pi to resource type Rj is denoted by Pi → Rj; it signifies that process Pi has requested an instance of resource type Rj and is currently waiting for that resource.

- A directed edge from resource type R_j to process P_i is denoted by $R_j \rightarrow P_i$; it signifies that an instance of resource type R_j has been allocated to process P_i. A directed edge $P_i \rightarrow R_j$ is called a request edge; a directed edge $R_j \rightarrow P_i$ is called an assignment edge.

- The resource-allocation graph shown in the following figure depicts the following situation.
  The sets P, R, and E:
  P = {P_1, P_2, P_3}



7.1 Resource Allocation Graph.

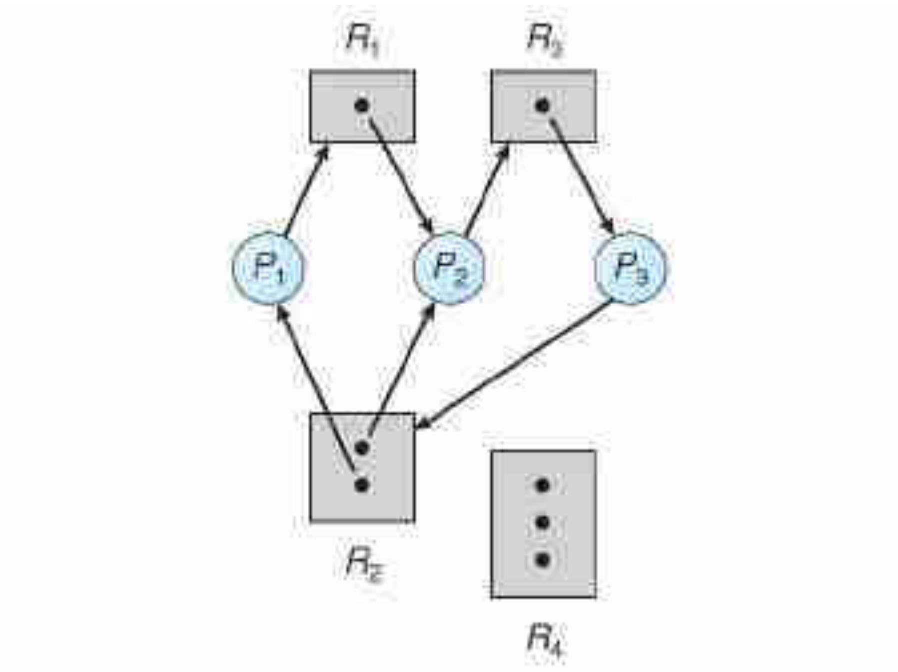
R = {R1, R2, R3, R4}
E = {P1 → R1, P2 → R3, R1 → P2, R2 → P2, R2 → P1, R3 → P3}

Resource instances:

- One instance of resource type R1
- Two instances of resource type R2
- One instance of resource type R3
- Three instances of resource type R4

Process states:

- Process P1 is holding an instance of resource type R2 and is waiting for an instance of resource type R1.
- Process P2 is holding an instance of R1 and an instance of R2 and is waiting for an instance of R3.
- Process P3 is holding an instance of R3.



7.2 Resource Allocation Graph with a Deadlock.

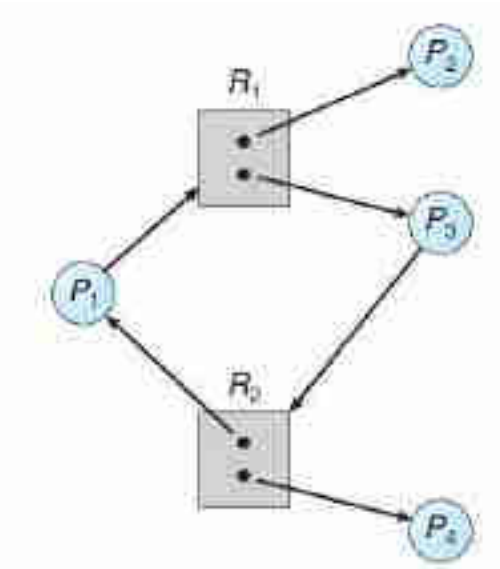
- Two minimal cycles exist in the system:

  $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

  $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

- Processes P_1, P_2, and P_3 are deadlocked. Process P_2 is waiting for the resource R_3, which is held by process P_3. Process P_3 is waiting for either process P_1 or process P_2 to release resource R_2. In addition, process P_1 is waiting for process P_2 to release resource R_1

Example2: Now consider the resource-allocation graph in Figure 7.3. In this example, we also have a cycle:
P1 → R1 → P3 → R2 → P1



7.3 Resource Allocation Graph with a Cycle but no Deadlock.

7.3 DEADLOCK PREVENTION VVIMP

- We can prevent Deadlock by eliminating any of the above four conditions.

1. **Eliminate Mutual Exclusion:** The mutual exclusion condition must hold for non-shareable resources that is only one process can use a resource at a time. If another process want to access that resource, then it must wait until the resource has been released.

2. **Eliminate No Preemption:** Preempt resources from the process when resources required by other high priority processes.

3. **Eliminate Hold and Wait:** Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. For example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remain blocked till it has completed its execution.

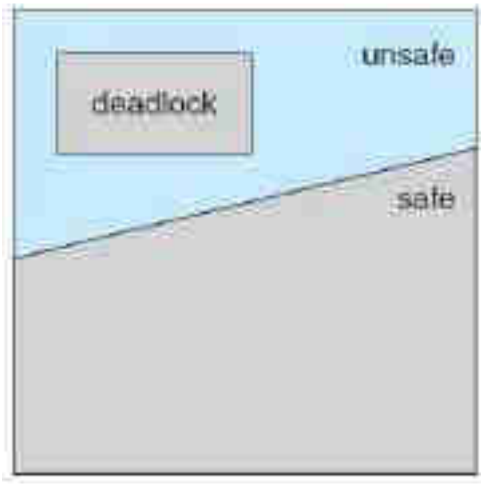
4. **Eliminate Circular Wait:** Processes waiting for resources from the others form a circular chain, that means all waiting processes form a circular chain or circular list where each process in the list is waiting for a resource held by next process in the list. This kind of situation is eliminated.

7.4 DEADLOCK AVOIDANCE

- In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.

7.4.1 Safe State

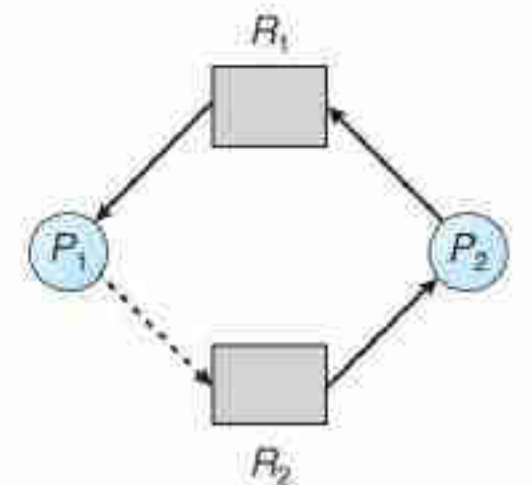
- The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.



7.6 Safe, unsafe, and deadlocked state spaces.

7.4.2 Resource-Allocation-Graph Algorithm

- Suppose that P_2 requests R_2. Although R_2 is currently free, we cannot allocate it to P_2, since this action will create a cycle in the graph. A cycle, as mentioned, indicates that the system is in an unsafe state. If P_1 requests R_2, and P_2 requests R_1, then a deadlock will occur.



7.8 An unsafe state in a Resource Allocation Graph.

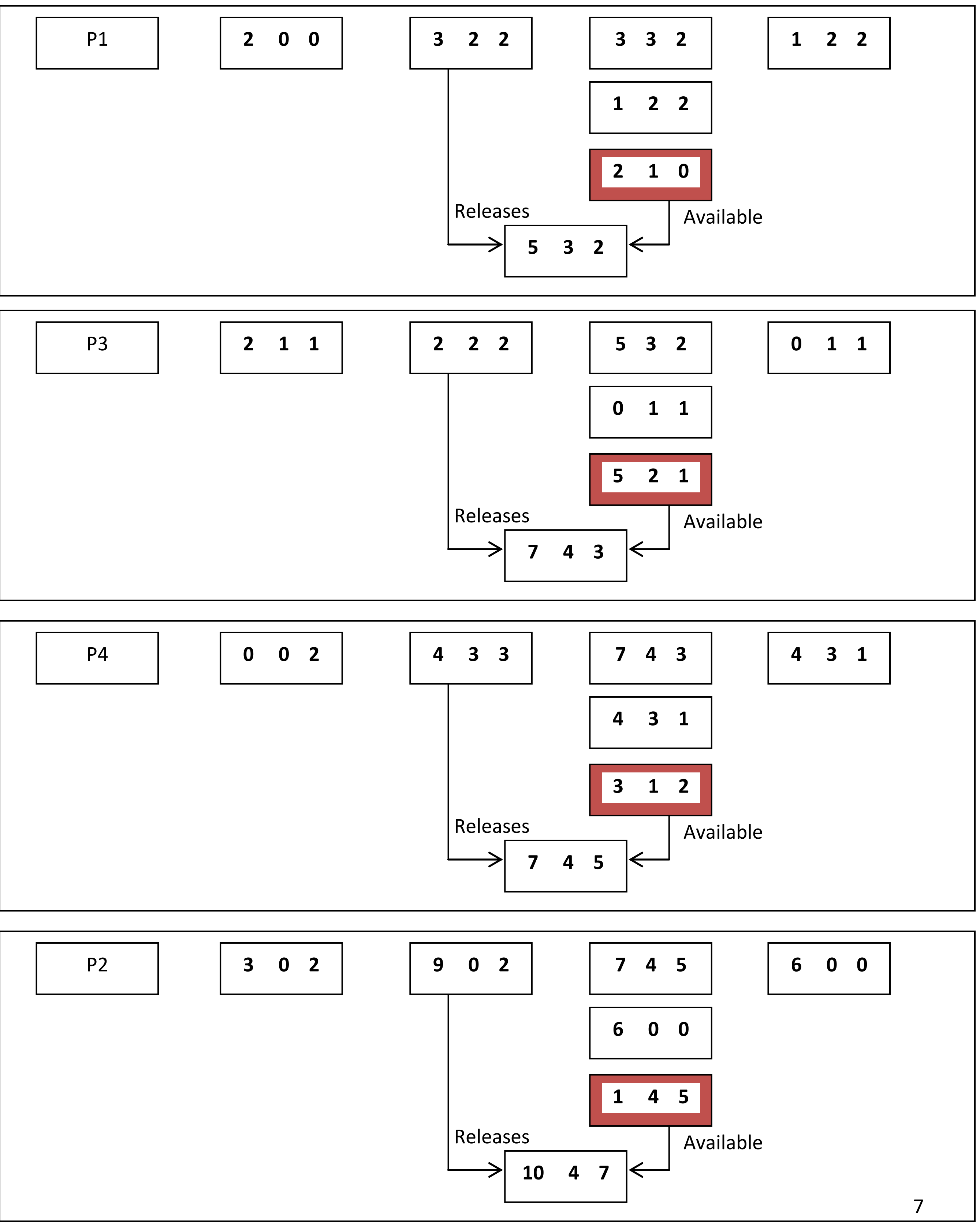
7.4.3 Banker's Algorithm VVIMP

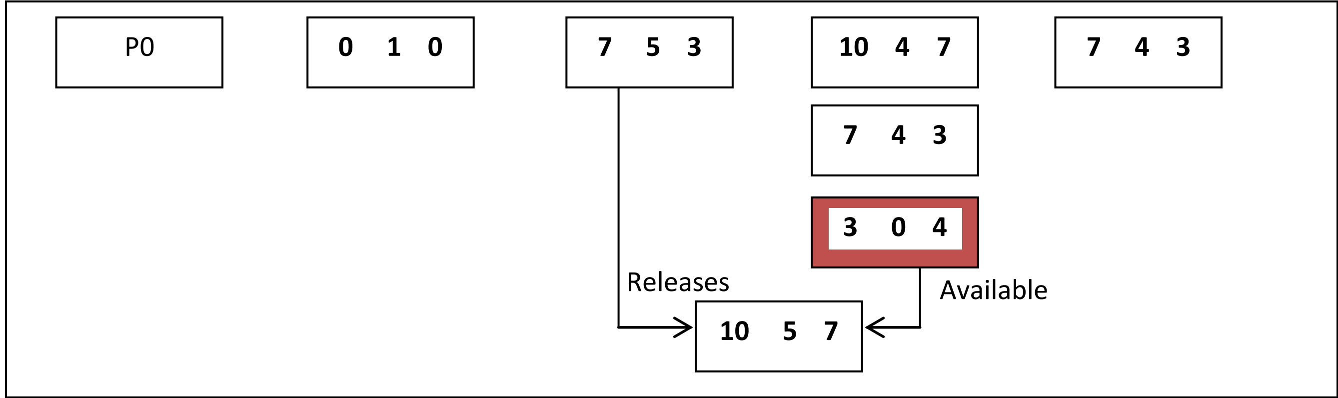
Bankers Algorithm is illustrated with the following example

Process	Allocation	Need	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

Solution:

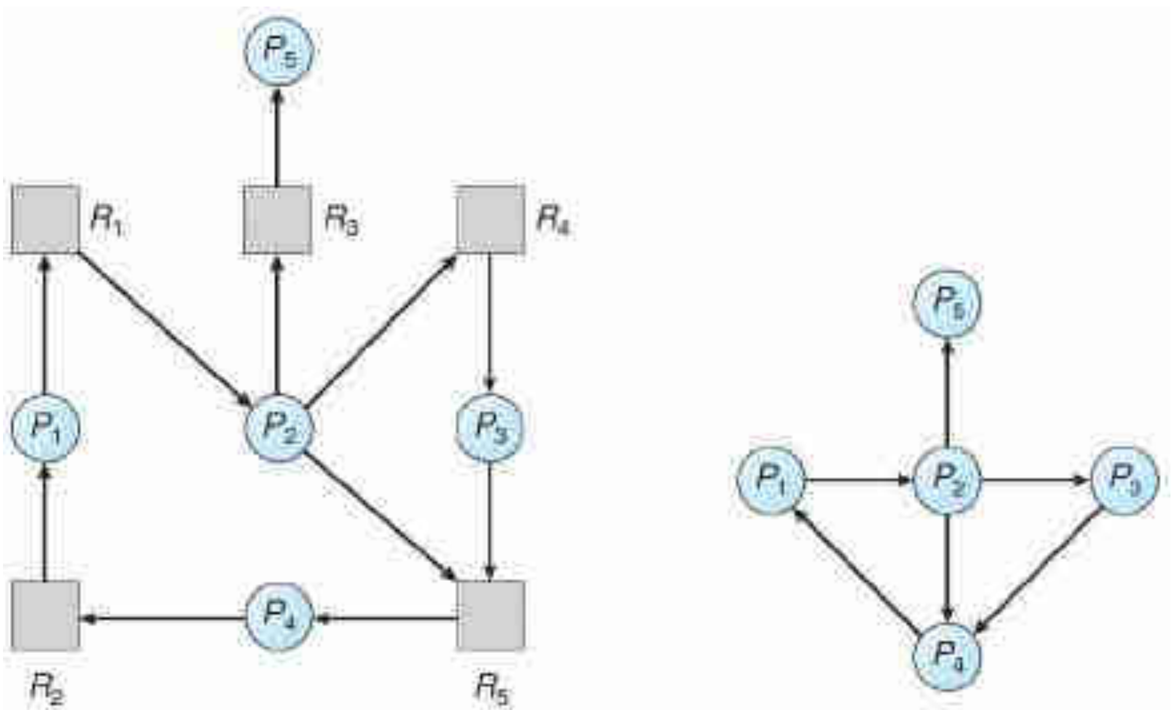
Process	Allocated	Need	Available	Maximum
	A B C	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2	7 4 3


P1
2 0 0
3 2 2
3 3 2
1 2 2
1 2 2
2 1 0
Releases
Available
5 3 2
P3
2 1 1
2 2 2
5 3 2
0 1 1
0 1 1
5 2 1
Releases
Available
7 4 3
P4
0 0 2
4 3 3
7 4 3
4 3 1
4 3 1
3 1 2
Releases
Available
7 4 5
P2
3 0 2
9 0 2
7 4 5
6 0 0
6 0 0
1 4 5
Releases
Available
10 4 7



7.5 CASE STUDIES

Problem 1: Verify whether the following resources allocation graphs are deadlocked or not.



(a) Resource-allocation graph. (b) Corresponding wait-for graph.

Solution: It is not possible to eliminate the conflicting situation of nodes P1 and P2. Since the above Resource Allocation Graph is Deadlocked.

Problem 2: **Illustrate the following example with Bankers Algorithm**

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

7.6 RECOVERY FROM DEADLOCK

7.6.1 Process Termination: To eliminate deadlocks by aborting a process, we use one of two methods. To eliminate the deadlock, we can simply kill one or more processes. For this, we use two methods:
(a) Abort all the Deadlocked Processes: Aborting all the processes will certainly break the deadlock, but with a great expense. The deadlocked processes may have computed for a long time and the result of those partial computations must be discarded and there is a probability to recalculate them later.

(b) Abort one process at a time until deadlock is eliminated: Abort one deadlocked process at a time, until deadlock cycle is eliminated from the system. Due to this method, there may be considerable overhead, because after aborting each process, we have to run deadlock detection algorithm to check whether any processes are still deadlocked.

2. Resource Preemption:T o eliminate deadlocks using resource preemption, we preempt some resources from processes and give those resources to other processes. This method will raise three issues
(a) Selecting a Victim: We must determine which resources and which processes are to be preempted and also the order to minimize the cost.

(b) Rollback: We must determine what should be done with the process from which resources are preempted. One simple idea is total rollback. That means abort the process and restart it.

(c) Starvation: In a system, it may happen that same process is always picked as a victim. As a result, that process will never complete its designated task. This situation is called **Starvation** and must be avoided. One solution is that a process must be picked as a victim only a finite number of times.

CHAPTER 8
MEMORY MANAGEMENT

8.1 BASICS OF MEMORY MANAGEMENT

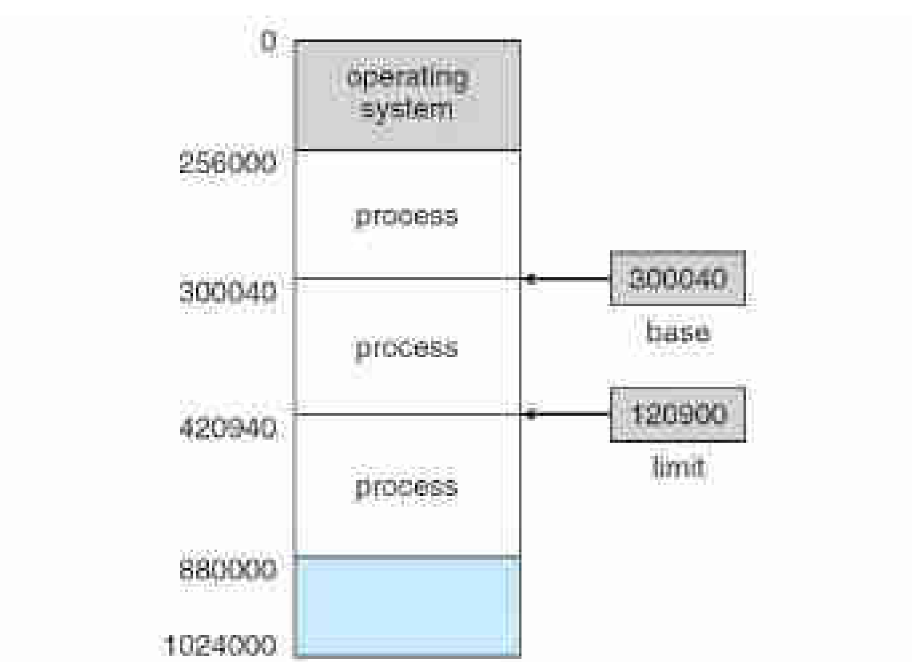
- Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

Why Memory Management is required:

- Allocate and de-allocate memory before and after process execution.
- To keep track of used memory space by processes.
- To minimize fragmentation issues.
- To proper utilization of main memory.
- To maintain data integrity while executing of process.

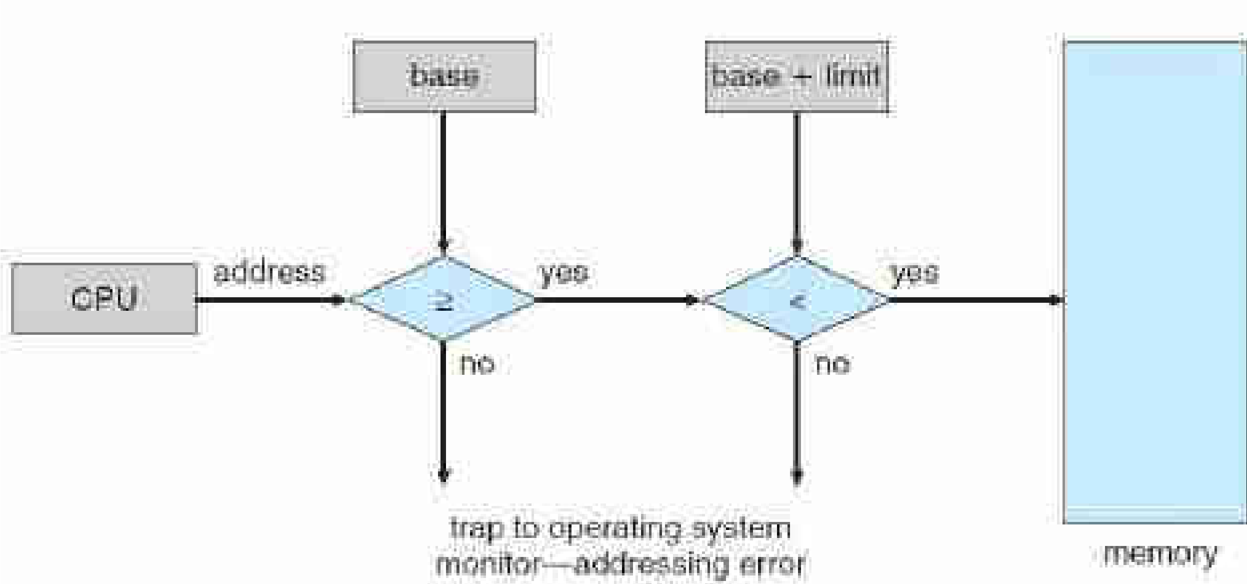
8.1.1 Basic Hardware

- Main memory and the registers built into the processor itself are the only general-purpose storage that the CPU can access directly



8.1 A base and a limit register define a logical address space.

- Memory protection is a way to control memory access rights on a computer, and is a part of most modern instruction set architectures and operating systems. This prevents a bug or malware within a process from affecting other processes, or the operating system itself.



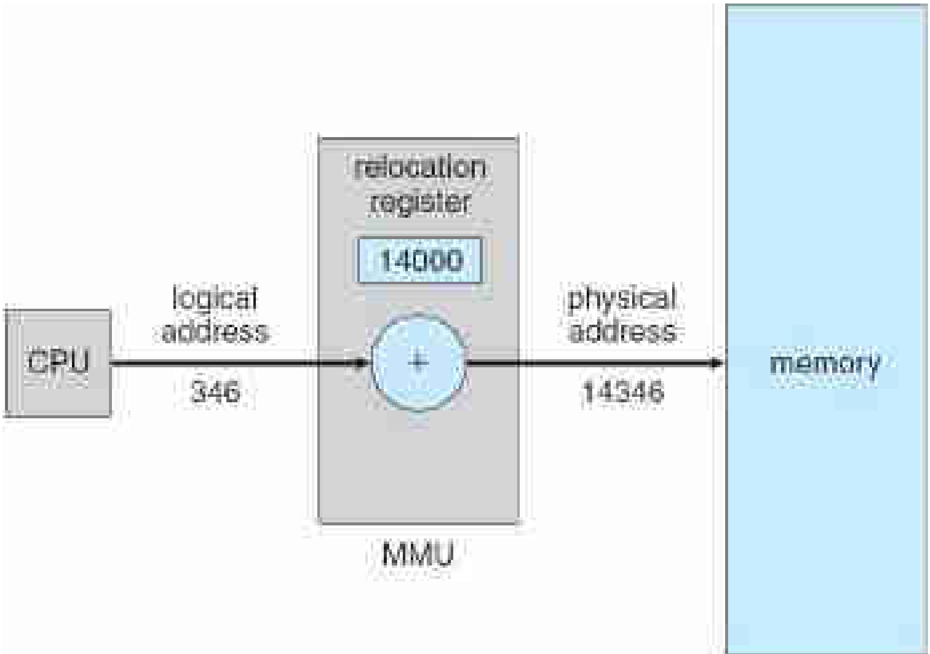
8.2 Hardware Address Protection with base and limit registers.

- **Now we are discussing the concept of logical address space and Physical address space:**

1. **Logical Address space:** An address generated by the CPU is known as "Logical Address". It is also known as a Virtual Address. Logical address space can be defined as the size of the process. A logical address can be changed.

2. **Physical Address space:** An address seen by the memory unit (i.e the one loaded into the memory address register of the memory) is commonly known as a "Physical Address".

- A Physical address is also known as a Real Address. The set of all physical addresses corresponding to these logical addresses is known as Physical Address Space.
- A physical address is computed by MMU. The run-time mapping from virtual to physical addresses is done by a hardware device Memory Management Unit (MMU). The physical address always remains constant.
- The base register holds the smallest legal physical memory address; the limit register specifies the size of the range. For example, if the base register holds. 300040 and the limit register is 120900, then the program can legally access all addresses from 300040 through 420939 (inclusive).

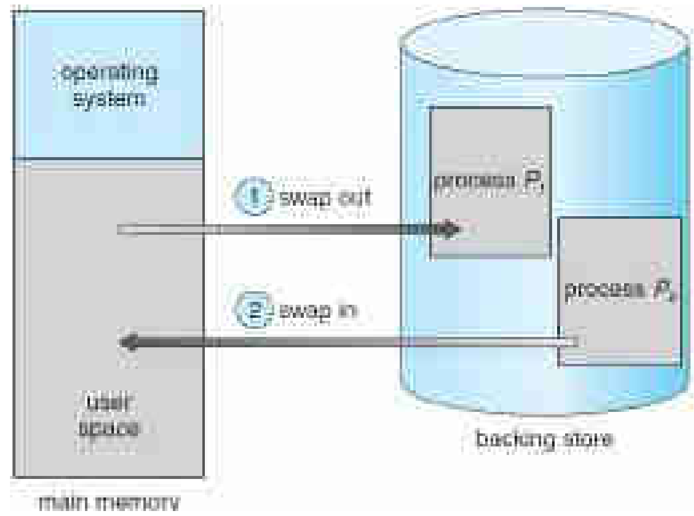


8.4 Dynamic relocation using a relocation register.

- In this case, we usually refer to the logical address as a virtual address. We use logical address and virtual address interchangeably in this text. The set of all logical addresses generated by a program is a logical address space. The set of all physical addresses corresponding to these logical addresses is a physical address space. Thus, in the execution-time address-binding scheme, the logical and physical address spaces differ.

8.2 SWAPPING IMP

- Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes.
- At some later time, the system swaps back the process from the secondary storage to main memory.



8.5 Swapping of two processes using a disk as a backing store

- The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Example: Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000 KB process to or from memory will take

```
2048KB / 1024KB per second
= 2 seconds
= 2000 milliseconds
```

8.2.1 Swapping on Mobile Systems: Although most operating systems for PCs and servers support some modified version of swapping, mobile systems typically do not support swapping in any form.

- Mobile devices generally use flash memory rather than more spacious hard disks as their persistent storage. The resulting space constraint is one reason why mobile operating system designers avoid swapping.

8.3 CONTIGUOUS MEMORY ALLOCATION

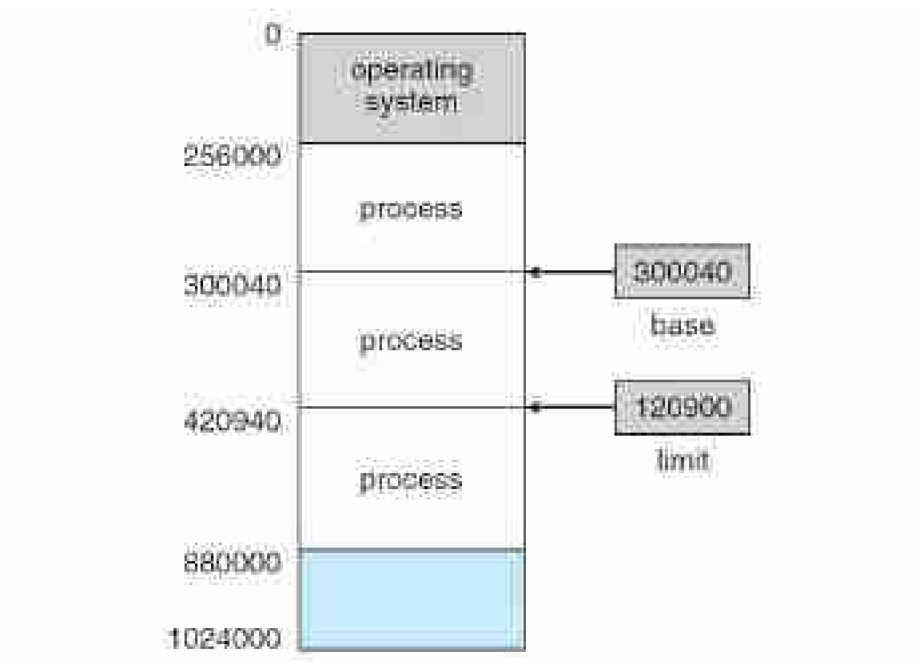
- **Contiguous Memory Allocation:** Contiguous memory allocation is basically a method in which a single contiguous section/part of memory is allocated to a process or file needing it. Because of this all the available memory space resides at the same place together, which means that the freely/unused available memory partitions are not distributed in a random fashion here and there across the whole memory space.

Processes

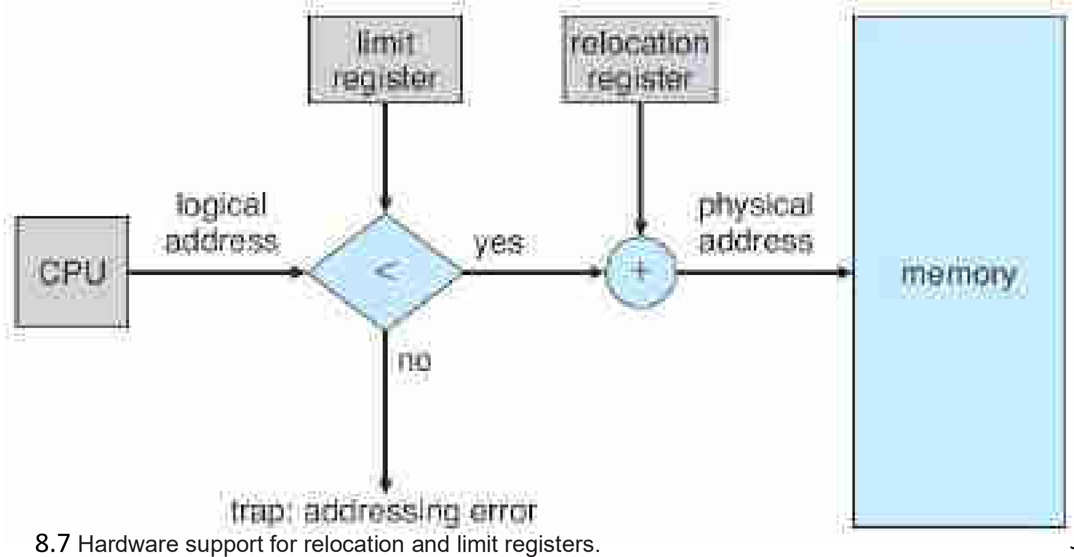
Memory

8.3.1 Memory Protection

- Memory protection is a way to control memory access rights on a computer, and is a part of most modern instruction set architectures and operating systems. This prevents a bug or malware within a process from affecting other processes, or the operating system itself.



8.6 A base and a limit register define a logical address space.



8.7 Hardware support for relocation and limit registers.

8.3.2 Memory Allocation & Fragmentation

- Fixed-size Partition Scheme: This technique is also known as Static Partitioning. The size of each partition is fixed.
- In this partition scheme, each partition may contain exactly one process. There is a problem that this technique will limit the degree of multiprogramming because the number of partitions will basically decide the number of processes.
- Whenever any process terminates then the partition becomes available for another process.

Example: Let's take an example of fixed size partitioning scheme, we will divide a memory size of 15 KB into fixed-size partitions.

3 KB	3 KB	3 KB	3 KB	3 KB

Advantages of Fixed-size Partition Scheme

1. This scheme is simple and is easy to implement.
2. It supports multiprogramming as multiple processes can be stored inside the main memory.
3. Management is easy using this scheme.

Disadvantages of Fixed-size Partition Scheme:

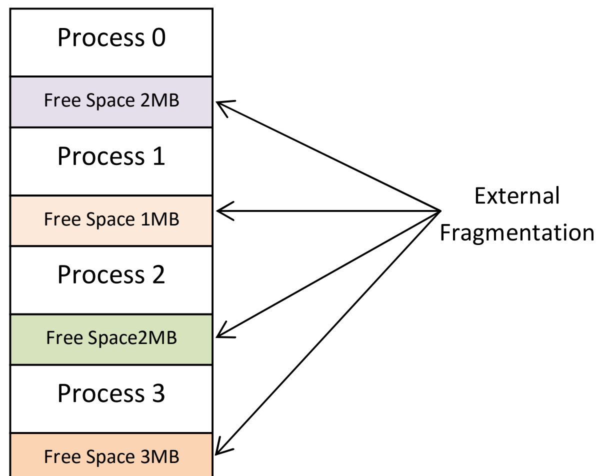
Some disadvantages of using this scheme are as follows:

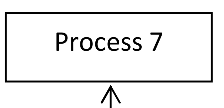
1. Internal Fragmentation: Suppose the size of the process is lesser than the size of the partition in that case some size of the partition gets wasted and remains unused. This wastage inside the memory is generally termed as internal fragmentation. As we have shown in the above diagram the 70 KB partition is used to load a process of 50 KB so the remaining 20 KB got wasted.

2. Limitation on the size of the process: If in a case size of a process is more than that of a maximum-sized partition then that process cannot be loaded into the memory.

3. External Fragmentation: It is another drawback of the fixed-size partition scheme as total unused space by various partitions cannot be used in order to load the processes even though there is the availability of space but it is not in the contiguous fashion.

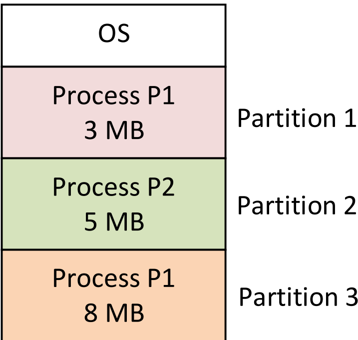
4. Degree of multiprogramming is less: In this partition scheme, as the size of the partition cannot change according to the size of the process. Thus the degree of multiprogramming is very less and is fixed.





Three is enough memory to run Process 7 but the memory is not contiguous.

- **Variable-size Partition Scheme:** This scheme is also known as Dynamic Partitioning and is come into existence to overcome the drawback i.e internal fragmentation that is caused by Static Partitioning. In this partitioning, scheme allocation is done dynamically.
- The size of the partition is not declared initially. Whenever any process arrives, a partition of size equal to the size of the process is created and then allocated to the process. Thus the size of each partition is equal to the size of the process.
- As partition size varies according to the need of the process so in this partition scheme there is no internal fragmentation.



Size of Partition = Size of Process

Advantages of Variable-size Partition Scheme

Some Advantages of using this partition scheme are as follows:

1. **No Internal Fragmentation:** As in this partition scheme space in the main memory is allocated strictly according to the requirement of the process thus there is no chance of internal fragmentation. Also, there will be no unused space left in the partition.
2. **Degree of Multiprogramming is Dynamic:** As there is no internal fragmentation in this partition scheme due to which there is no unused space in the memory. Thus more processes can be loaded into the memory at the same time.
3. **No Limitation on the Size of Process:** In this partition scheme as the partition is allocated to the process dynamically thus the size of the process cannot be restricted because the partition size is decided according to the process size.

Disadvantages of Variable-size Partition Scheme

Some Disadvantages of using this partition scheme are as follows:

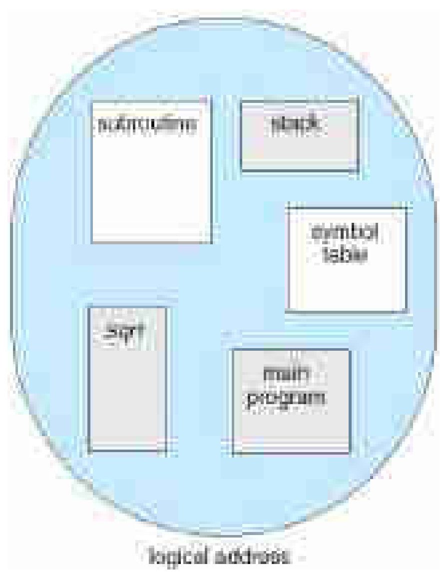
1. **External Fragmentation**: As there is no internal fragmentation which is an advantage of using this partition scheme does not mean there will no external fragmentation. Let us understand this with the help of an example. In the above diagram Process P1 (3MB) and Process P3 (8MB) completed their execution. Hence there are two spaces left i.e. 3MB and 8MB. Let's there is a Process P4 of size 15 MB comes. But the empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. Because the rule says that process must be continuously present in the main memory in order to get executed. Thus it results in External Fragmentation.

8.4 SEGMENTATION

- In Operating Systems, Segmentation is a memory management technique in which the memory is divided into the variable size parts. Each part is known as a segment which can be allocated to a process. The details about each segment are stored in a table called a segment table. Segment table is stored in one (or many) of the segments.

  Example: Normally, when a program is compiled, the compiler automatically constructs segments reflecting the input program.
  A C compiler might create separate segments for the following:
  1. The code
  2. Global variables
  3. The heap, from which memory is allocated
  4. The stacks used by each thread
  5. The standard C library

  Libraries that are linked in during compile time might be assigned separate segments. The loader would take all these segments and assign them segment numbers.



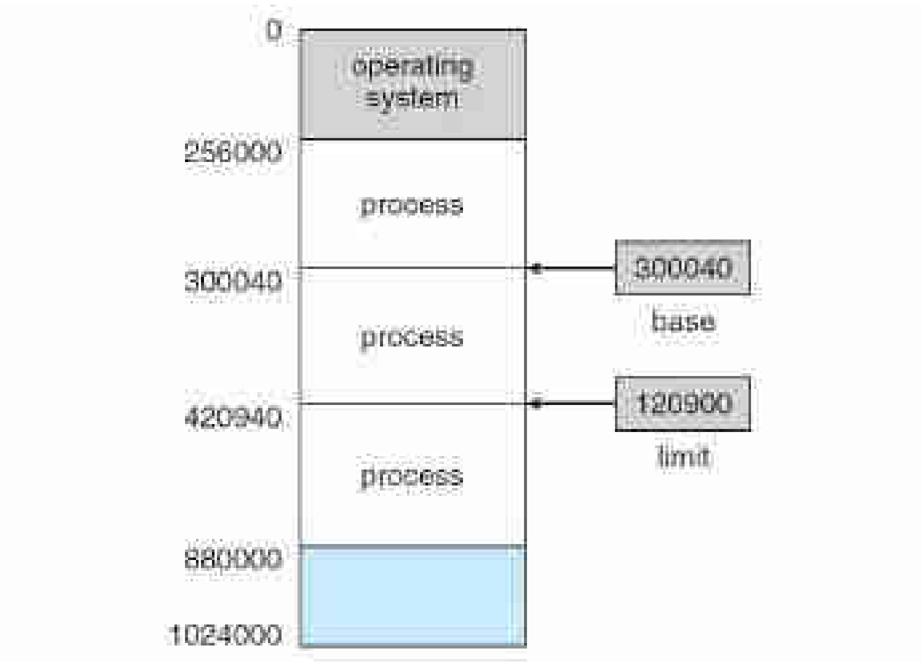
8.8 Programmer's view of a program.

8.4.2 Segmentation Hardware

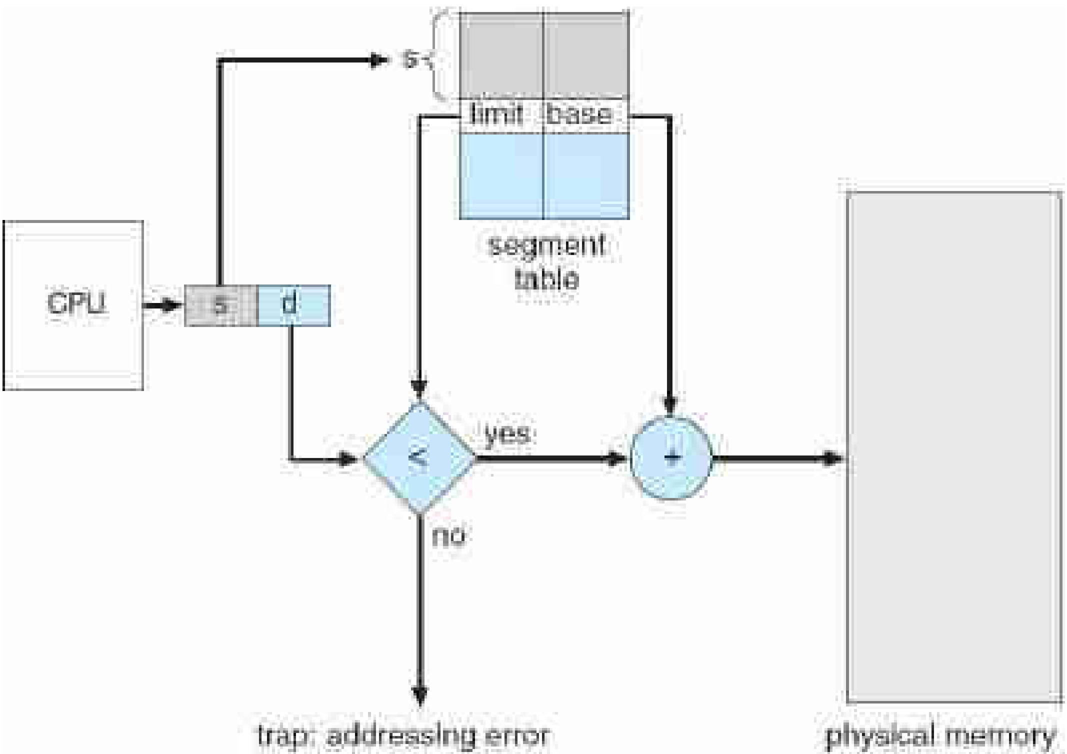
- Segment table contains mainly two information about segment:

Base: It is the base address of the segment
Limit: It is the length of the segment.

- Translation of Logical address into physical address by segment table

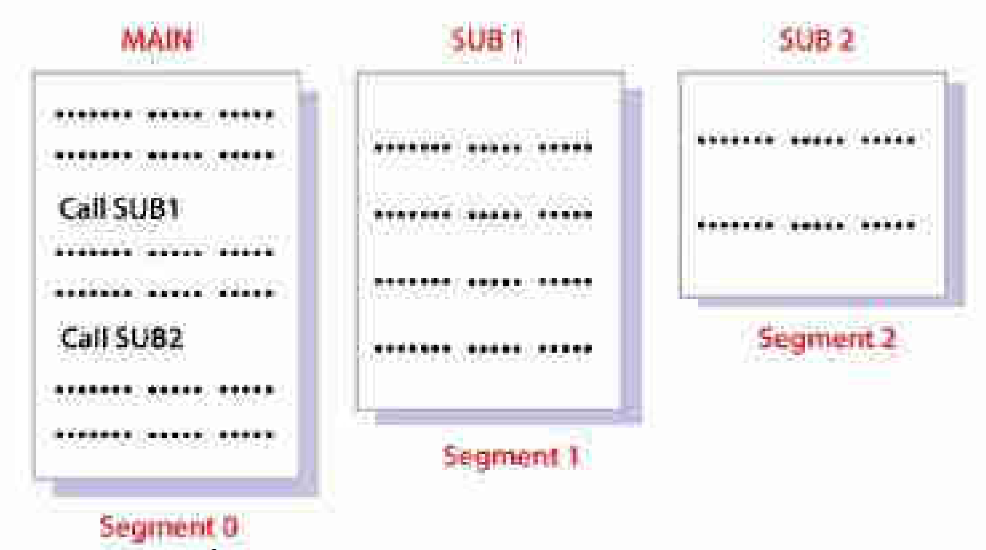


8.9 A base and a limit register define a logical address space.

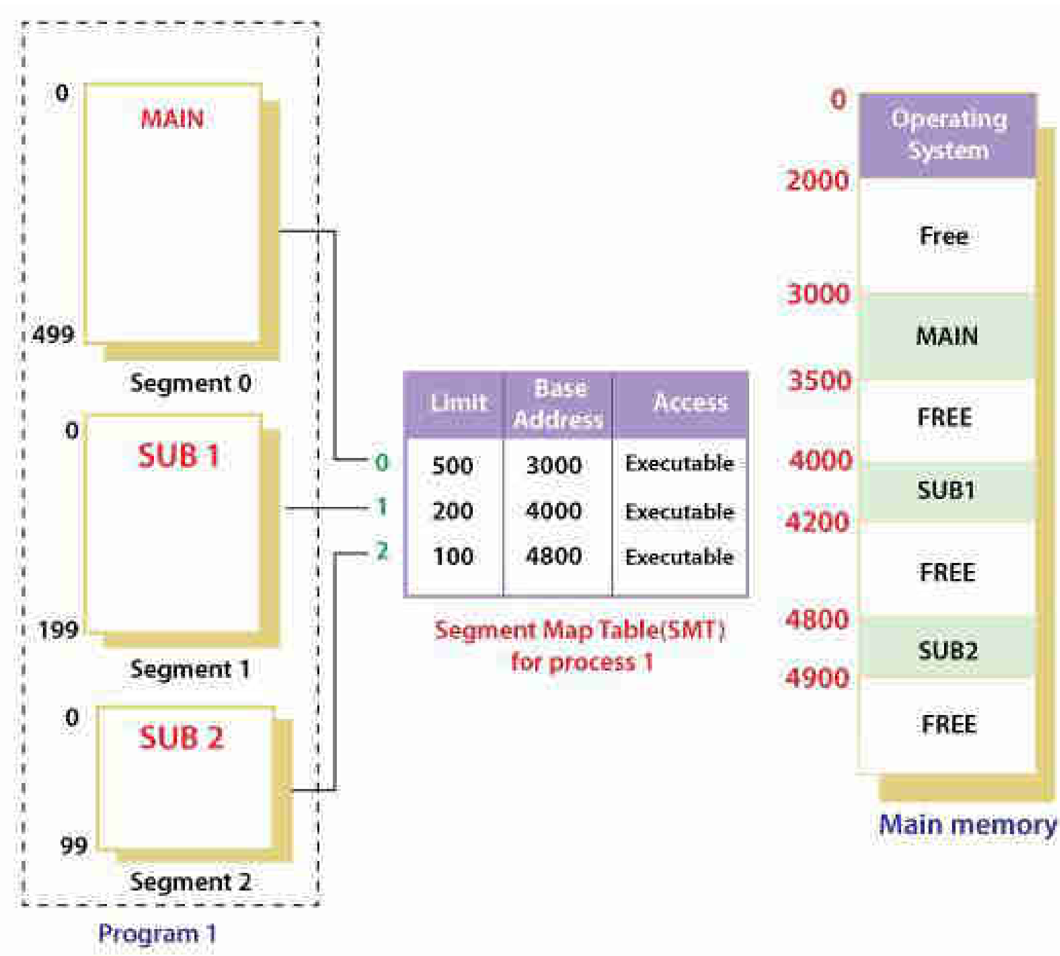


8.10 Translation of Logical address into physical address by segment table.

Example:



8.11 Division of main program into Segment.



8.12 Converting Logical Address into Physical Addresses.

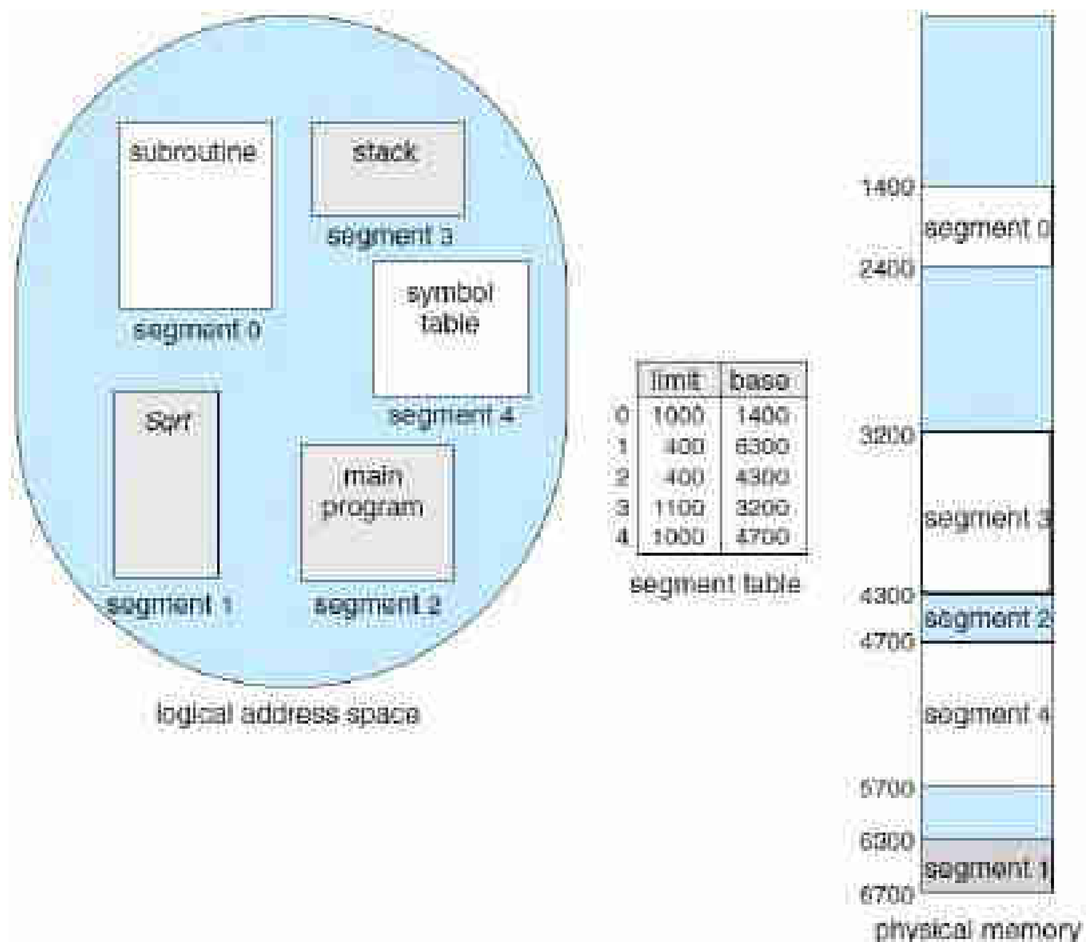
Advantages of Segmentation:
1. No internal fragmentation
2. Average Segment Size is larger than the actual page size.
3. Less overhead
4. It is easier to relocate segments than entire address space.
5. The segment table is of lesser size as compared to the page table in paging.

Disadvantages:
1. It can have external fragmentation.
2. It is difficult to allocate contiguous memory to variable sized partition.
3. Costly memory management algorithms.

8.5 PAGING

- Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.

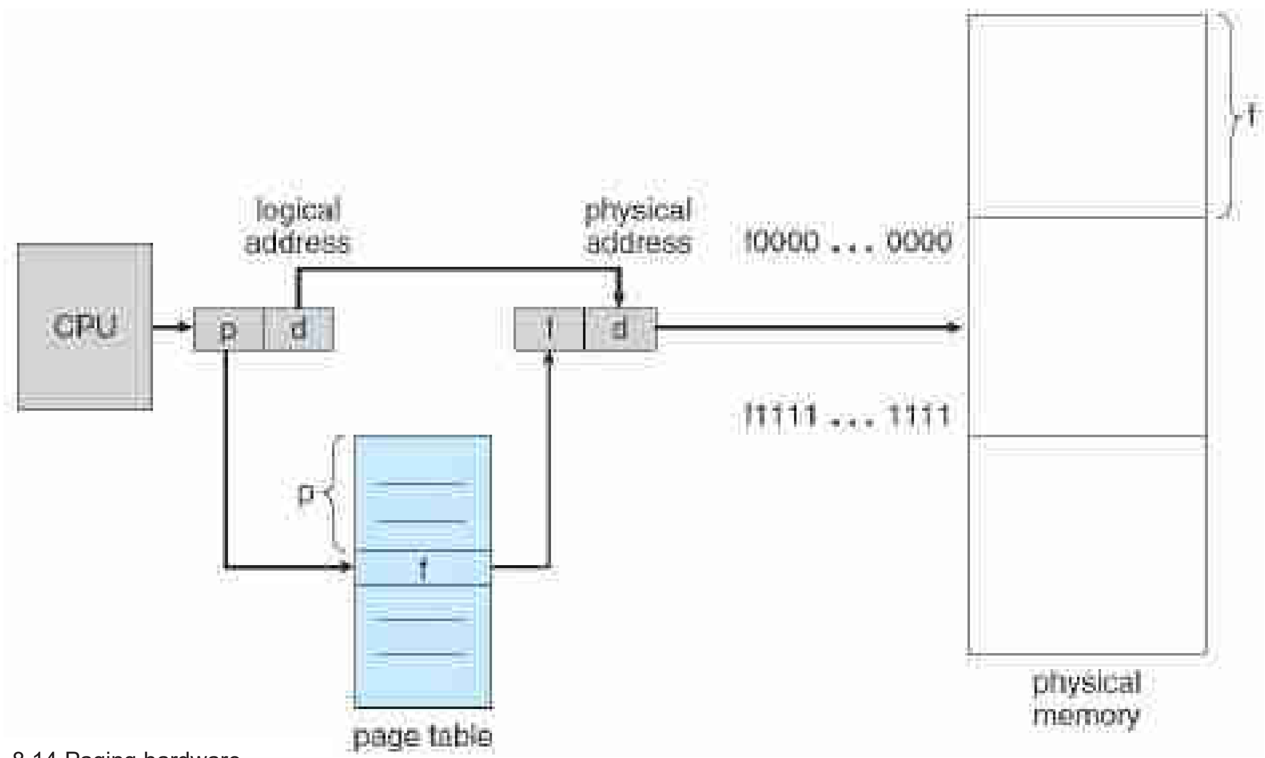
- Logical Address or Virtual Address: An address generated by the CPU.
- Logical Address Space or Virtual Address Space: The set of all logical addresses generated by a program
- Physical Address: An address actually available on memory unit.
- Physical Address Space: The set of all physical addresses corresponding to the logical addresses



8.13 Example of segmentation.

8.5.1 Hardware support for Paging

- The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as paging technique.
- The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.
- The Logical address Space is also splitted into fixed-size blocks, called **pages**.
  Page Size = Frame Size



8.14 Paging hardware.

PROCESS			MAIN MEMORY
Logical Address			Physical Address
Page			Index

logical memory

Page	Address	Value
Page 0	0	a
	1	b
	2	c
	3	d
Page 1	4	e
	5	f
	6	g
	7	h
Page 2	8	i
	9	j
	10	k
	11	l
Page 3	12	m
	13	n
	14	o
	15	p

page table

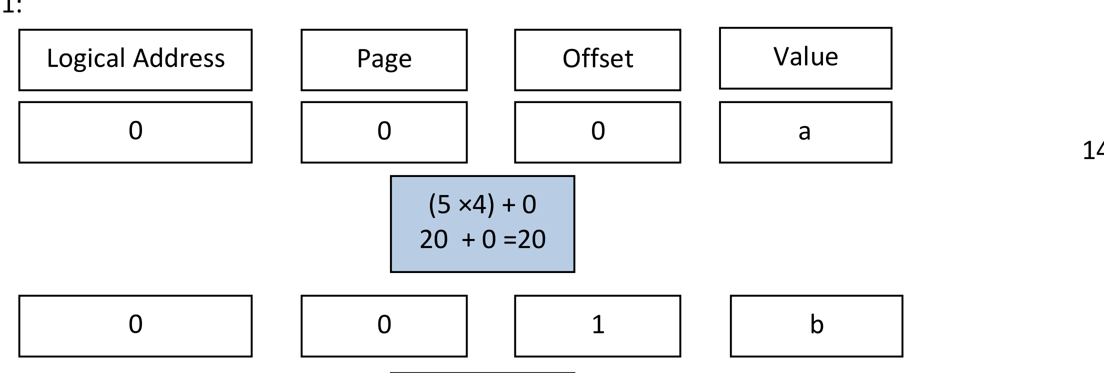
Page	Frame
0	5
1	6
2	1
3	2

physical memory

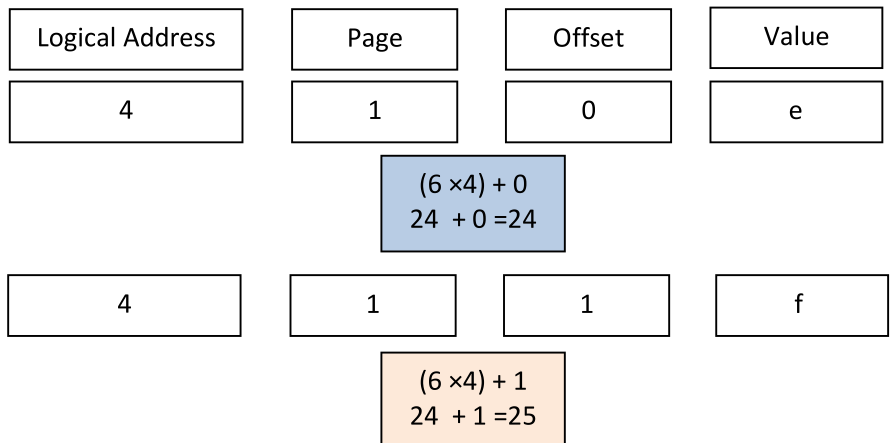
Address	Contents	Index
0		0
4	i j k l	1
8	m n o p	2
12		3
16		4
20	a b c d	5
24	e f g h	6
28		7

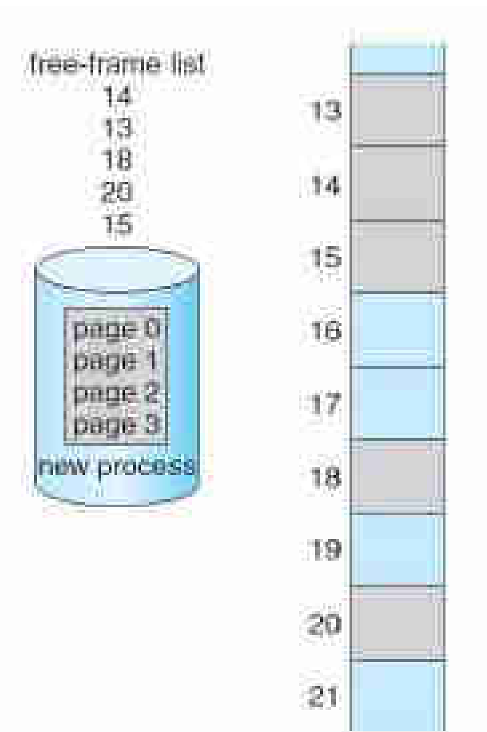
8.15 Paging example for a 32-byte memory with 4-byte pages.

Example 1:

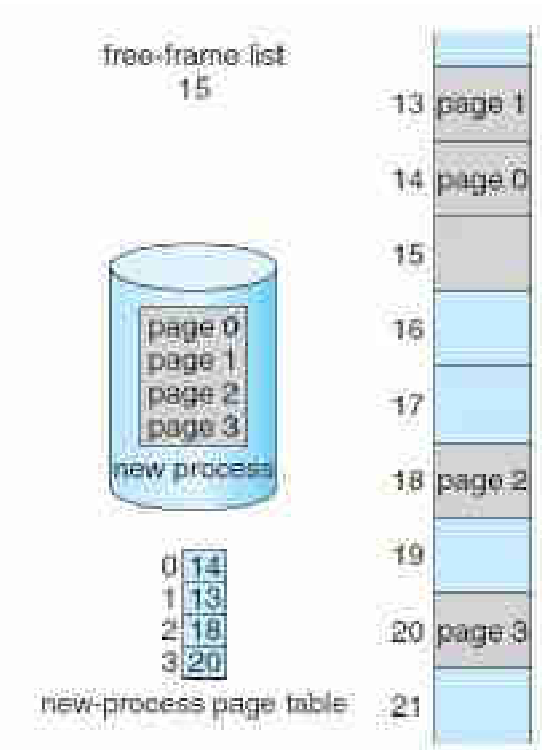


Example 1:





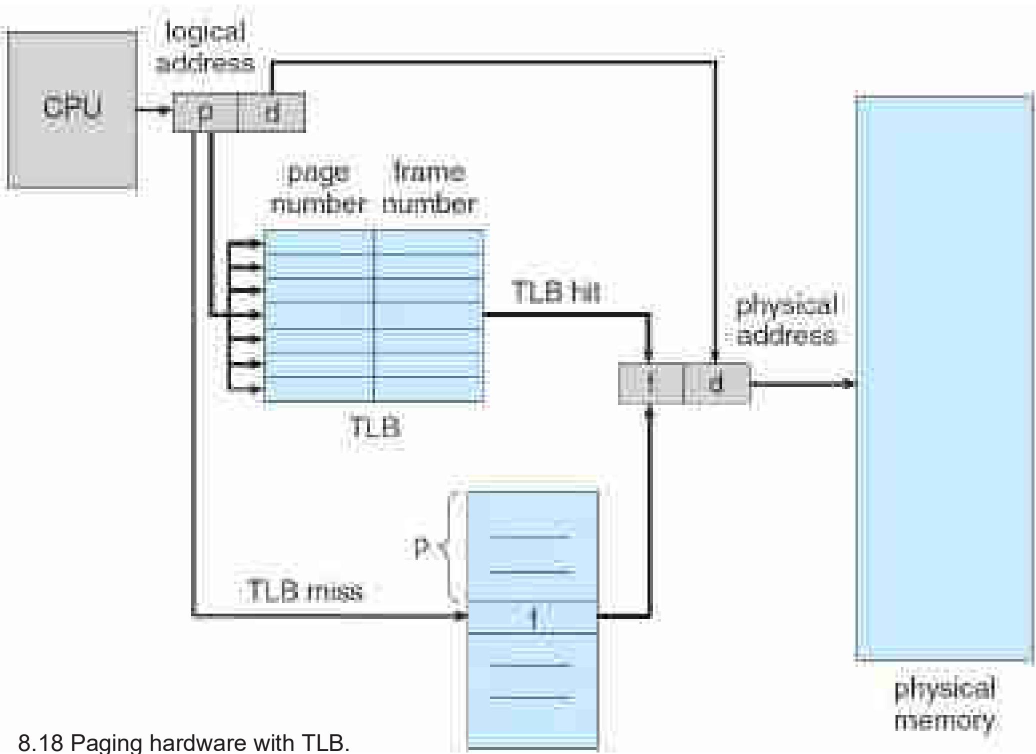
8.16 Free Frame before the allocation.



8.17 Free Frame after the Allocation.

8.5.2 Paging Hardware with TLB

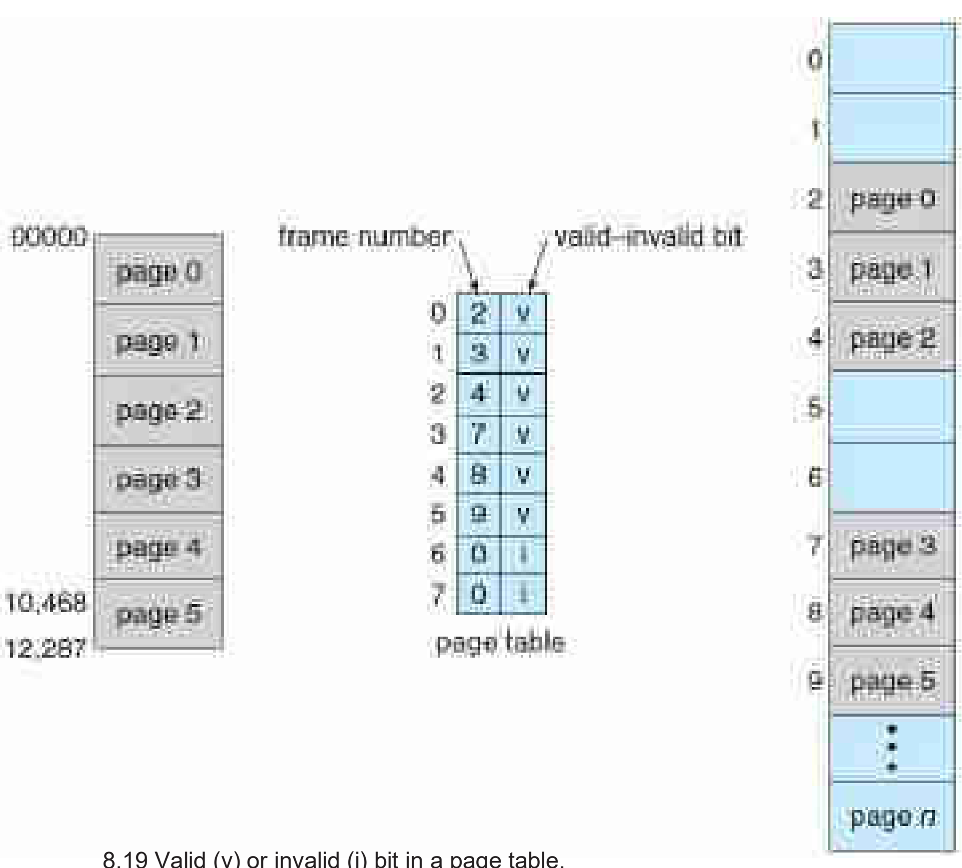
- Translation Lookaside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions.
- TLB contains page table entries that have been most recently used.
- Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved and the real address is formed.
- If a page table entry is not found in the TLB (TLB miss), the page number is used as index while processing page table. TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.



8.18 Paging hardware with TLB.

8.5.3 Protection

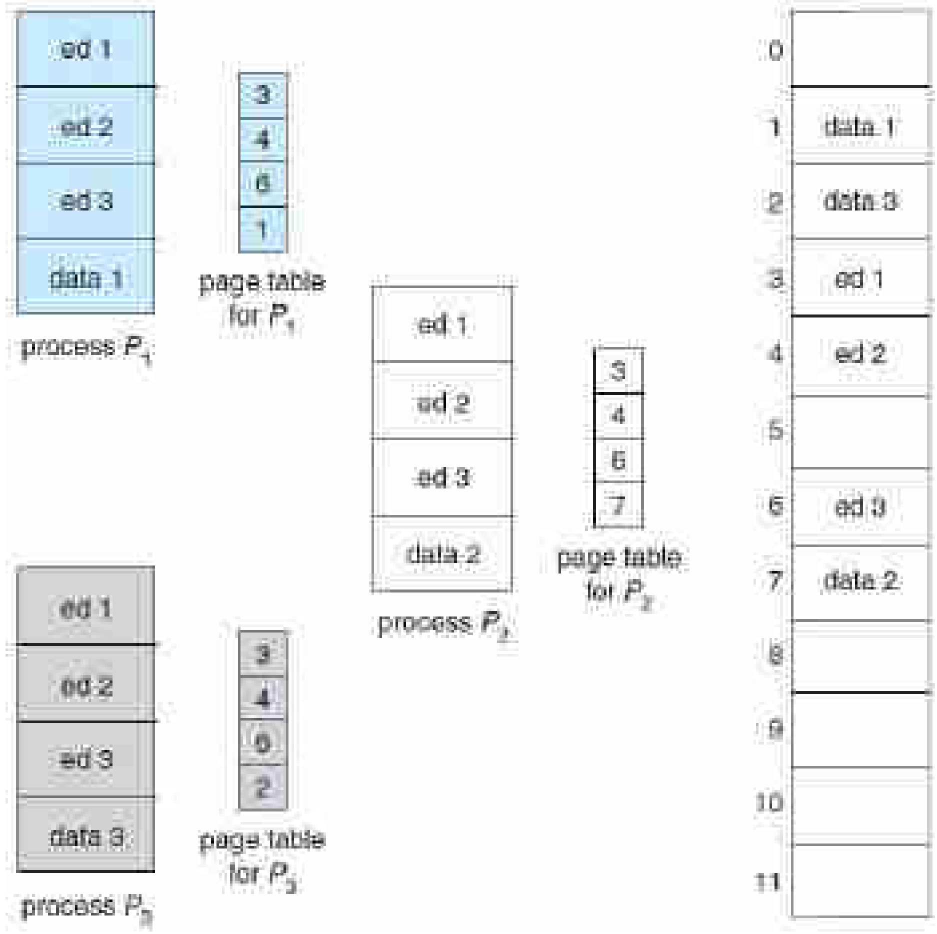
- Memory protection in a paged environment is accomplished by protection bits associated with each frame.
- One additional bit is generally attached to each entry in the page table: a valid-invalid bit.
- When this bit is set to valid, the associated page is in the process's logical address space and is thus a legal (or valid) page.
- When the bit is set to invalid, the page is not in the process's logical address space. Illegal addresses are trapped by use of the valid-invalid bit. The operating system sets this bit for each page to allow or disallow access to the page



8.19 Valid (v) or invalid (i) bit in a page table.

8.5.4 Shared Pages in OS

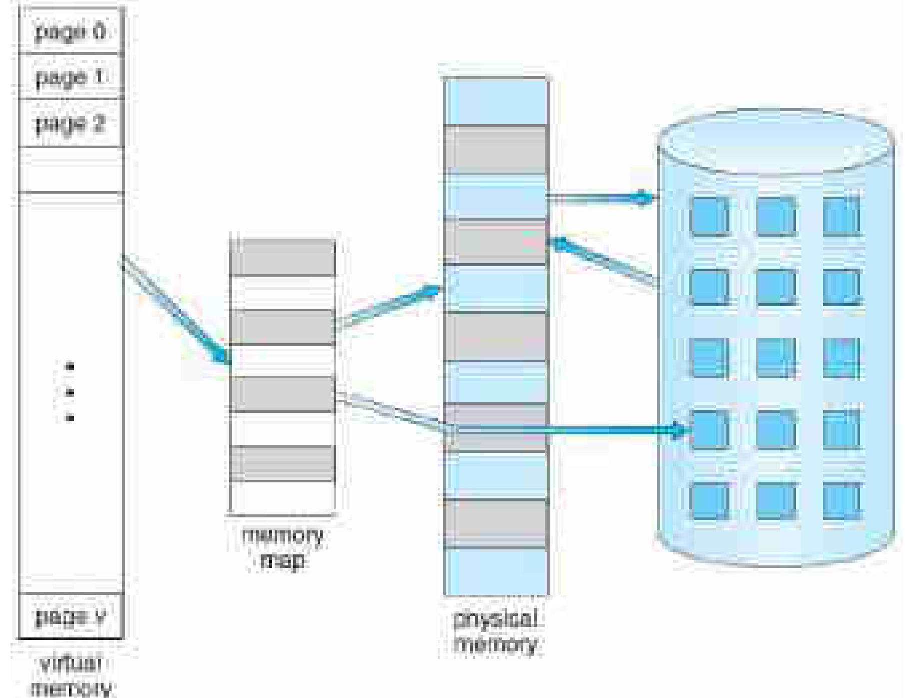
- Sharing is another design issue for paging system.
- It is very common for many computer users to be running the same program at the same time in a large multiprogramming computer system.
- Now, to avoid having two copies of same page in the memory at the same time, just share the pages.



8.20 Sharing of code in a paging environment.

CHAP 9
VIRTUAL MEMORY

- Virtual Memory: Virtual memory is a feature of an operating system that enables a computer to be able to compensate shortages of physical memory by transferring pages of data from disk storage to primary memory.



9.1 Diagram showing virtual memory that is larger than physical memory.

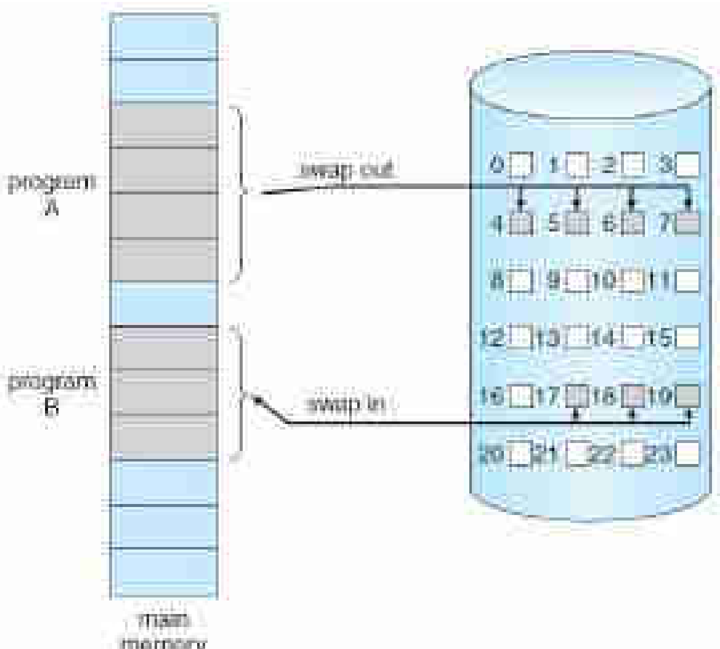
9.1 DEMAND PAGING

- Demand Paging: The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.

- Steps to be followed in Demand Paging:

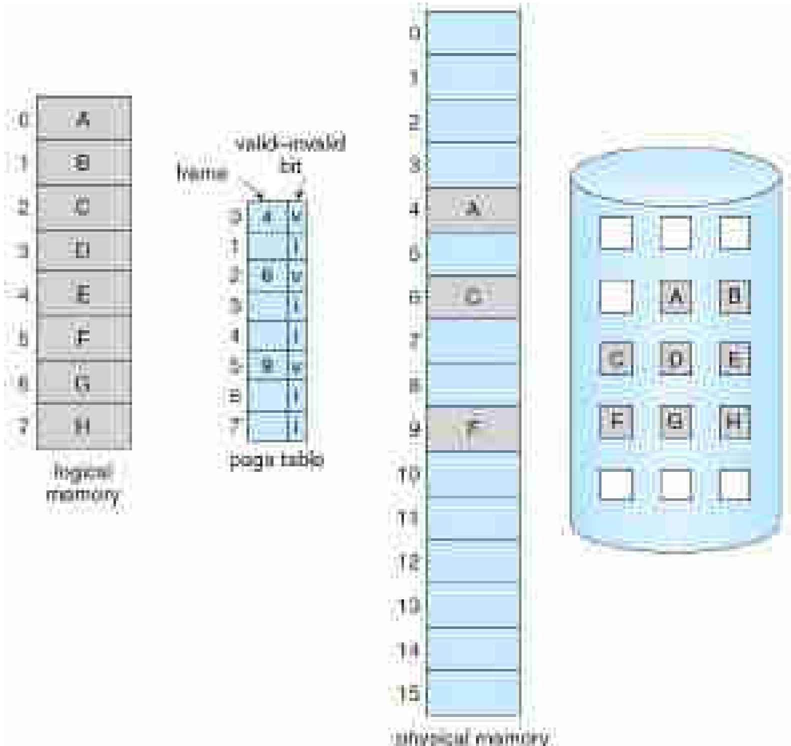
1. Attempt to access page.
2. If page is valid (in main memory) then continue processing instruction as normal.
3. If page is invalid then a **page-fault trap** occurs.
4. Check if the memory reference is a valid reference to a location on secondary memory. If not, the process is terminated (**illegal memory access**). Otherwise, we have to **page in** the required page.

5. Schedule disk operation to read the desired page into main memory.
6. Restart the instruction that was interrupted by the operating system trap.



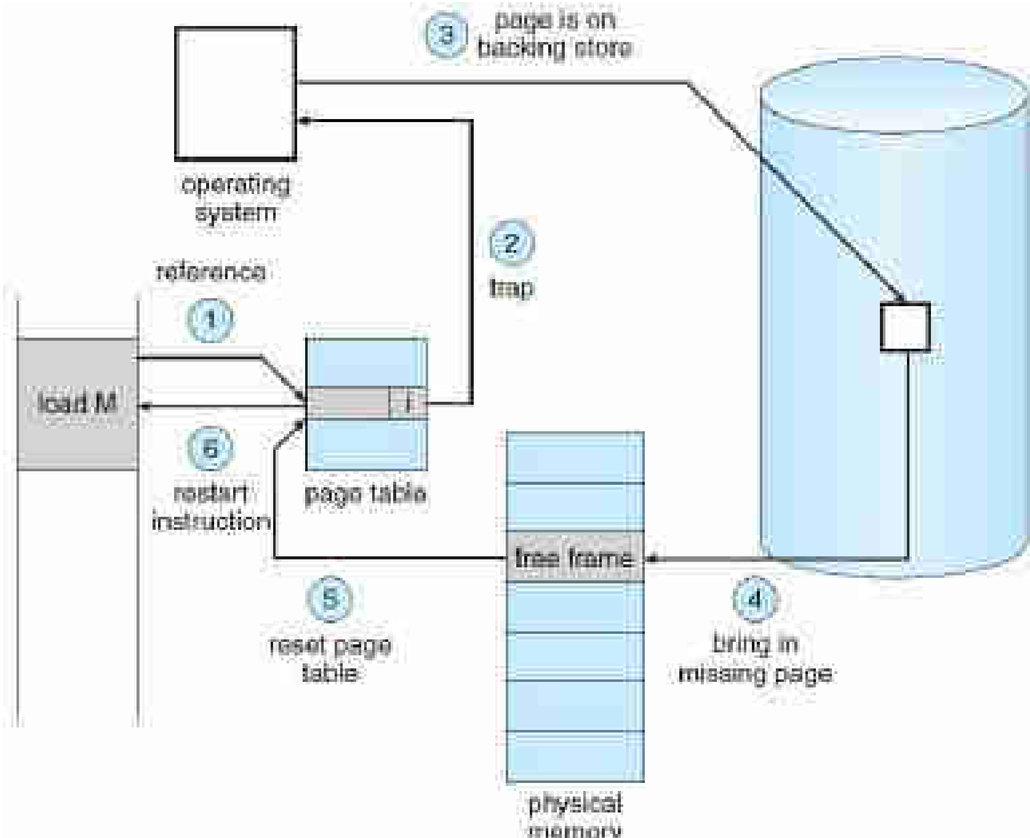
9.2 Transfer of a paged memory to contiguous disk space.

- Page table when some pages are not in main memory.



9.3 Page table when some pages are not in main memory.

- Steps in handling a page fault.

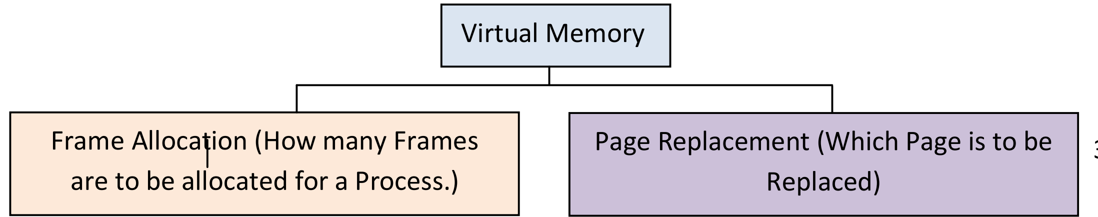


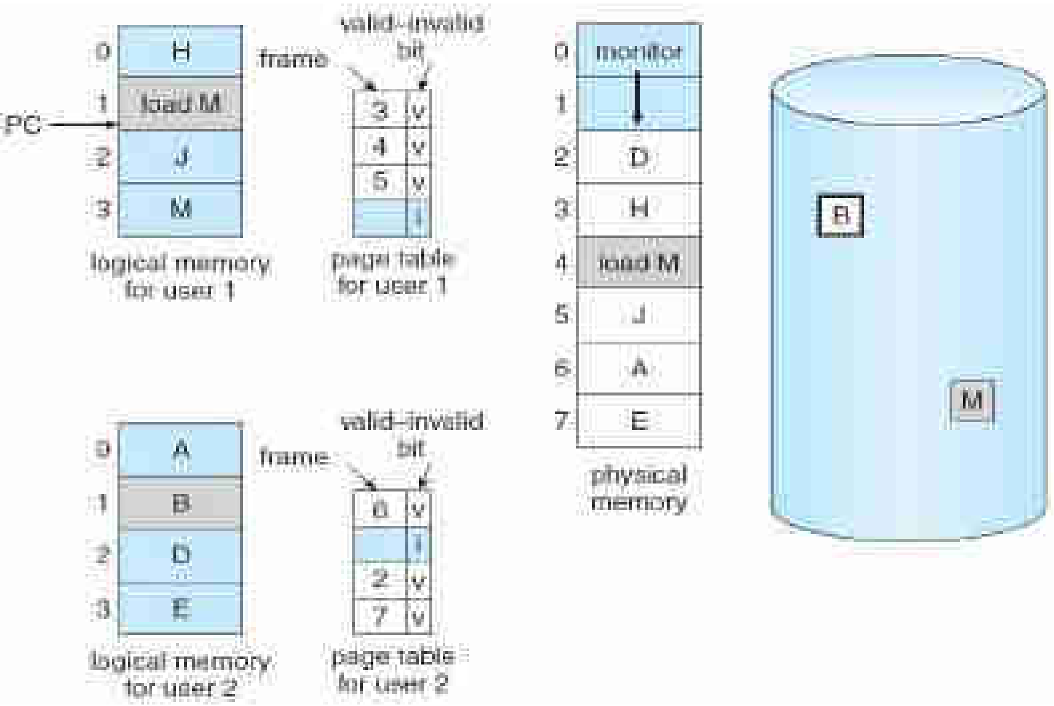
9.4 Steps in handling a page fault.

9.2 PAGE REPLACEMNT

- The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault).

- There are two main aspects of virtual memory, Frame allocation and Page Replacement. It is very important to have the optimal frame allocation and page replacement algorithm. Frame allocation is all about how many frames are to be allocated to the process while the page replacement is all about determining the page number which needs to be replaced in order to make space for the requested page

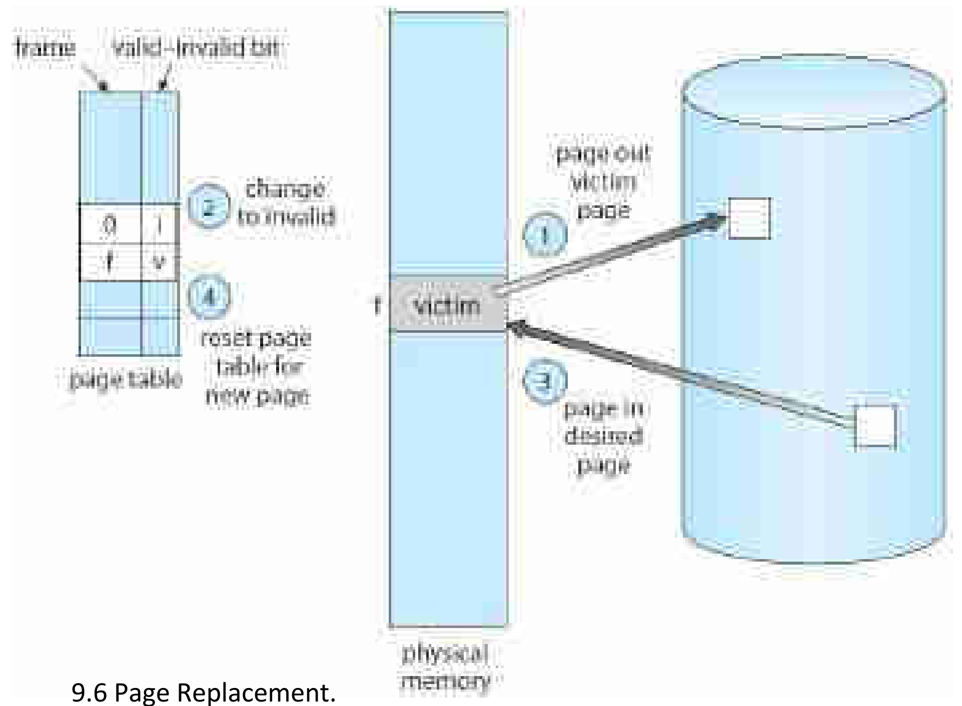




9.5 Need for page replacement.

9.2.1 Basic Page Replacement

- Page replacement takes the following approach. If no frame is free, we find one that is not currently being used and free it.



9.6 Page Replacement.

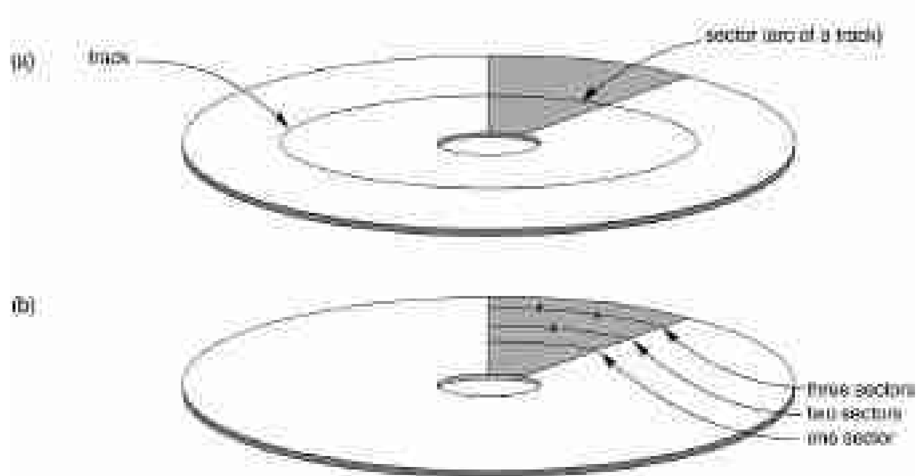
1. Find the location of the desired page on the disk.
2. Find a free frame:
    a. If there is a free frame, use it.
    b. If there is no free frame, use a page-replacement algorithm to select a victim frame.
    c. Write the victim frame to the disk; change the page and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Continue the user process from where the page fault occurred.

CHAPTER-10
MASS-STORAGE STRUCTURE

10.1 OVERVIEW OF MASS-STORAGE STRUCTURE

10.1.1 Magnetic Disks

- Magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material .
- Both sides of the disk are used.
- Several disks may be stacked on one spindle with Read/Write heads available on each surface.
- All disks rotate together at high speed and not stopped or started for access purpose.
- Used to store large amount of data .
- Basic unit of storage is BIT .
- BIT is 1 by magnetizing area on disk .
- Character: Group of bits.
- Capacity: Number of bytes it can store.
- Number of tracks: ranges from few hundred to few thousand.
- Capacity of each track: ranges from tens of Kbytes to 150 Kbytes.
- Formatting: Tracks are divided into equal sized disk blocks called sectors.
- Sector size is fixed range from 512 to 4096 bytes.



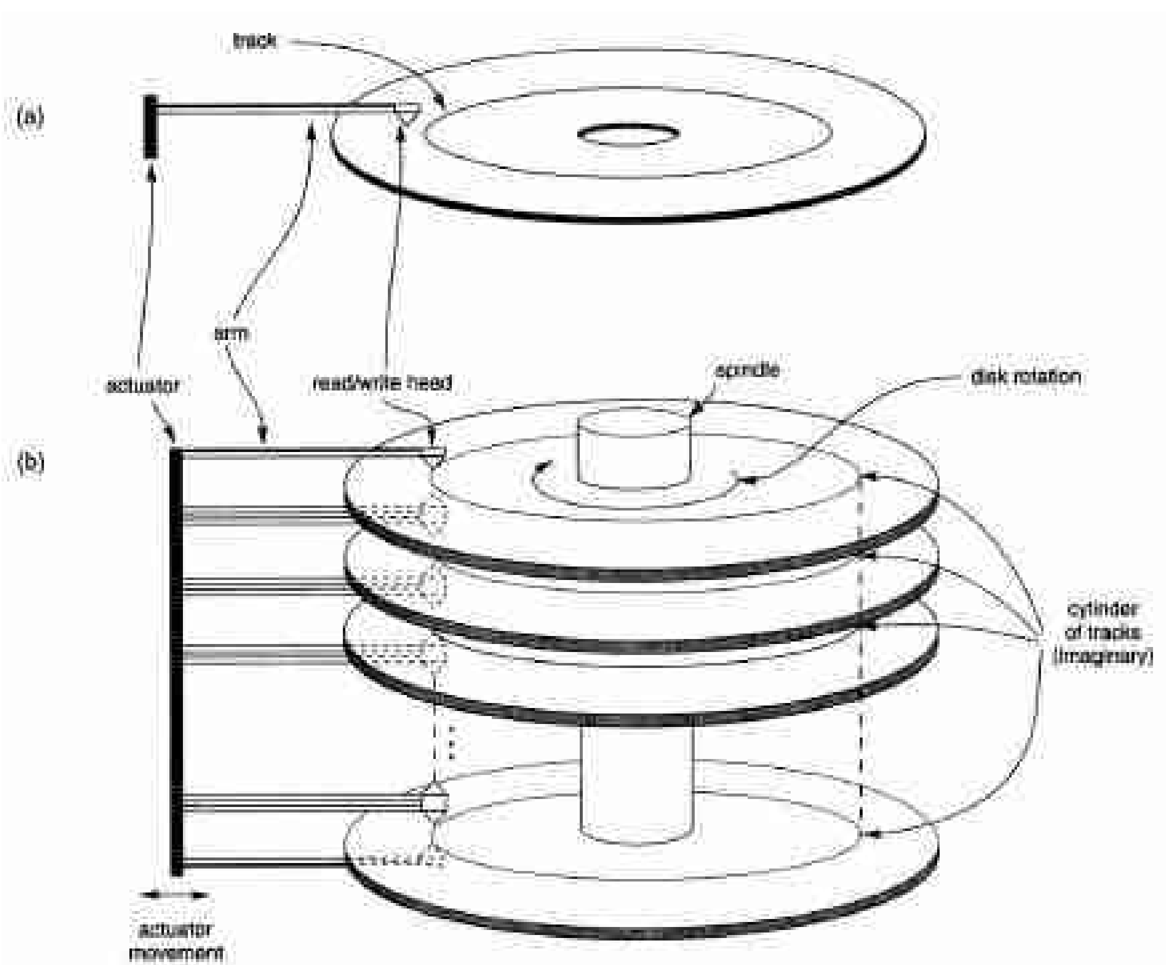
- Read/Write head is classified into
- Fixed head system.
- Moving head system.

- Rotate disk pack of cylinders ranging from 3600 and 7200 rpm

- Access time: Time required to access the data.
  t_A = Access time
  $t_A = t_S + t_L$

- Seek time: Time required to position the read/write head on the correct track & correct surface.
  t_S = Seek time

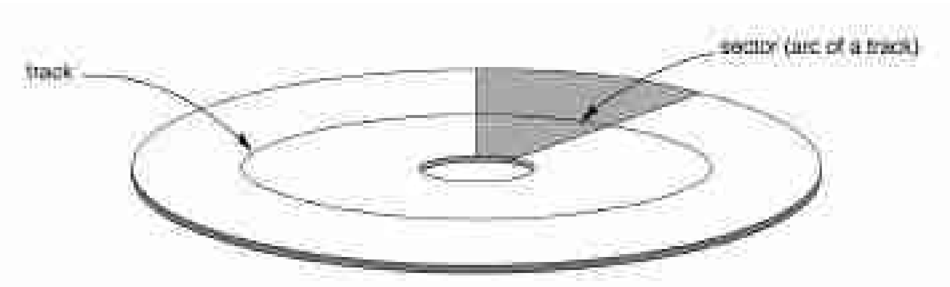
- Latency time: Time required to locate the required disk block from the beginning.
  t_L = Latency time



10.1.2 Floppy Disk

- A disk drive with removable disks are called a Floppy Disk
- The disks used with a Floppy disk drive are small disks made of plastic coated with magnetic recording material
- There are two sizes commonly used, with diameters of 5.25 and 3.5 inches
- The 3.5 inch disks are smaller and can store more data than 5.25 inch disks
- Floppy disks extensively used in personal computer as a medium for distributing software to computer users
- Basic unit of storage is BIT .
- BIT is 1 by magnetizing area on disk .
- Character: Group of bits.
- Capacity: Number of bytes it can store.
- Density : Single density and Double density
- Data stored in: Tracks & Sectors
- Track: Data is placed in concentric magnetic circles called tracks
- Sector :Each track is divided into storage units called sectors
- FAT uses 512 byte sectors exclusively.
- The number of Sectors per Track vary depending on the media and format.'
- Physical Sector Numbering starts with the number 1at the beginning of each track and side.
- Number of Tracks: 80 in 3.5 inch floppy disk

Disk Formats - 3.5"		
	Low Density	High Density
Tracks	80	80
Sectors per track	9	18
Bytes per sector	512	512
Total capacity	720K	1.44M



- Formatting: Tracks are divided into equal sized disk blocks called sectors.

- Access time: Time required to access the data.
  t_A = Access time
  $t_A = t_S + t_L$

- Seek time: Time required to position the read/write head on the correct track & correct surface.
  t_S = Seek time

- Latency time: Time required to locate the required disk block from the beginning.
  t_L = Latency time

10.1.3 Magnetic Tape

- Magnetic tape transport consists of the electrical, mechanical and electronic components provide the parts and control mechanism for magnetic tape unit.

- The tape is a strip of plastic coated with magnetic recording media .

- Bits are recorded as magnetic spots on the tape along with several tracks.

- Usually, seven or nine bits are recorded simultaneously to form a character together with a parity bit.

- The information is recorded in blocks referred to as records.

- Each record on tape has an identification bit pattern at the beginning and end. By recording the bit pattern at the beginning, the tape control identifies the record number.

- By reading the bit pattern at the end of the record, the control recognizes the beginning of a gap

- A tape unit is addressed by specifying the record number and number of characters in the record

- Record may be fixed length or variable length

- Sequential access.
- Read/Write head: Reads or writes data on tape.
- Strorage: 1600 to 6250 bytes per inch.
- Inter block gap: 0.6 inches.

Block		Block		Block		Block		Block		Block

10.2 DISK SCHEDULING IMP

- The algorithms used for disk scheduling are called as disk scheduling algorithms.
- The purpose of disk scheduling algorithms is to reduce the total seek time.
- **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.

10.2.1 FCFS (First Come First Served) Scheduling

- As the name suggests, this algorithm entertains requests in the order they arrive in the disk queue.
- It is the simplest disk scheduling algorithm.

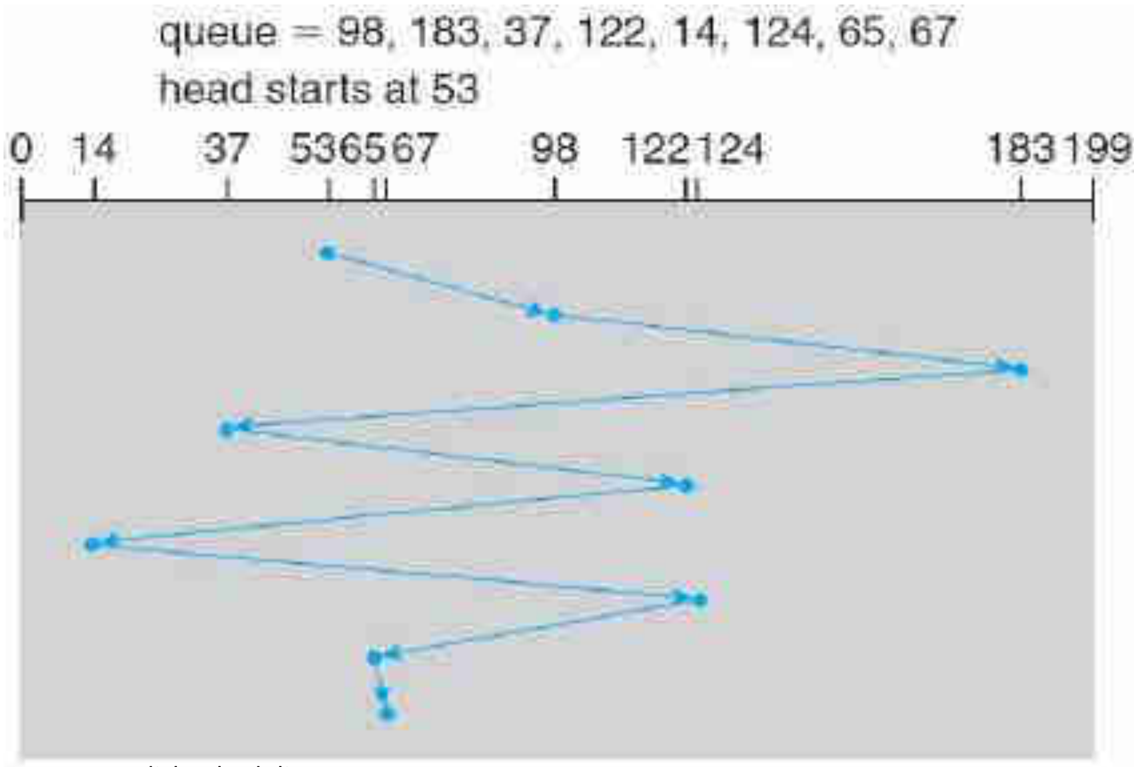
Advantages:

- It is simple, easy to understand and implement.
- It does not cause starvation to any request.

Disadvantages:

- It results in increased total seek time.
- It is inefficient.

Problem1: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. The FCFS scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. Find the total head movement (in number of cylinders) incurred while servicing these requests.



10.1 FCFS disk scheduling.

Total head movements incurred while servicing these requests: (98-53) + (183-98) + (183-37) +(122-37) +(122-14) + (124-14) + (124-65) + (67-65) = 45 + 85 + 146 + 85 + 108 + 110 + 59 + 2 = 640

Problem2: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The FCFS scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. Find the total head movement (in number of cylinders) incurred while servicing these requests.

10.2.2 SSTF (Shortest Seek Time First) Scheduling

- SSTF stands for Shortest Seek Time First.
- This algorithm services that request next which requires least number of head movements from its current position regardless of the direction.
- It breaks the tie in the direction of head movement.

Advantages:

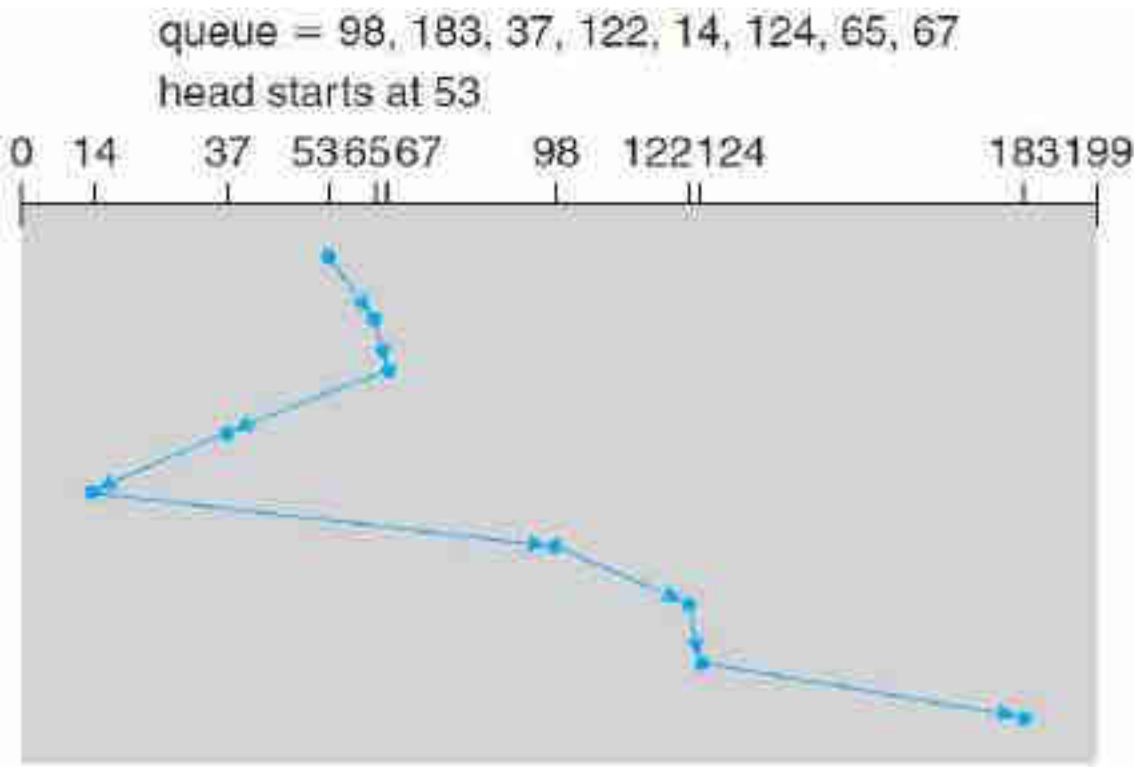
- It reduces the total seek time as compared to FCFS.
- It provides increased throughput.

- It provides less average response time and waiting time.

Disadvantages:

- There is an overhead of finding out the closest request.
- The requests which are far from the head might starve for the CPU.
- It provides high variance in response time and waiting time.
- Switching the direction of head frequently slows down the algorithm.

Example 2: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. The SSTF scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. Find the total head movement (in number of cylinders) incurred while servicing these requests.



10.2 SSTF disk scheduling.

Total head movements incurred while servicing these requests: (65-53) + (67-65) + (67-37) + (37-14) + (98-14) + (122-98) +(124-122) + (183-124) = 12 + 2 + 30 + 23 + 84 + 24 + 2 + 59 = 236

Example 2: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The SSTF scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders

are numbered from 0 to 199. Find the total head movement (in number of cylinders) incurred while servicing these requests.

10.2.3 SCAN Scheduling

- As the name suggests, this algorithm scans all the cylinders of the disk back and forth.
- Head starts from one end of the disk and move towards the other end servicing all the requests in between.
- After reaching the other end, head reverses its direction and move towards the starting end servicing all the requests in between.
- The same process repeats.

Note:

- SCAN Algorithm is also called as **Elevator Algorithm.**
- This is because its working resembles the working of an elevator.

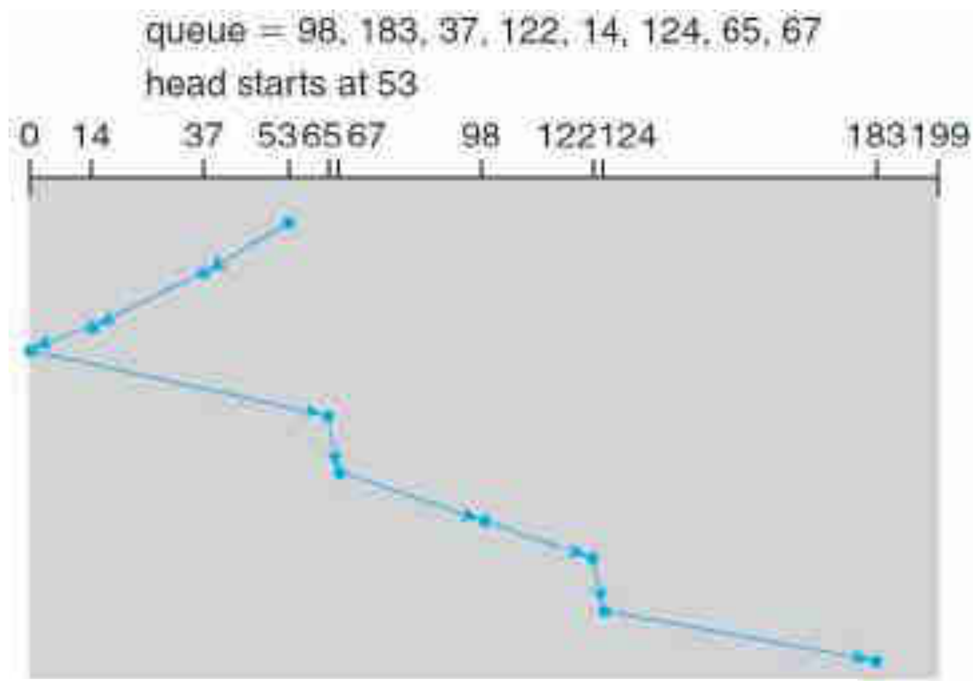
Advantages:

- It is simple, easy to understand and implement.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

Disadvantages:

- It causes long waiting time for the cylinders just visited by the head.
- It causes the head to move till the end of the disk even if there are no requests to be serviced.

Example 1: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The SCAN scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. Find the total head movement (in number of cylinders) incurred while servicing these requests.



10.3 SCAN Disk Scheduling.

Total head movements incurred while servicing these requests =

(53-37) + (37-14) +(14-0) + (65-0) + (67-65) + (98-67) + (122-98) + (124-122) + (183-124) = 16 +23 +14 + 65 + 2 +31+ 24 + 2 + 59 = 236

Example 2: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The SCAN scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. Find the total head movement (in number of cylinders) incurred while servicing these requests.

10.2.4 C-SCAN Scheduling

- Circular-SCAN Algorithm is an improved version of the SCAN Algorithm.
- Head starts from one end of the disk and move towards the other end servicing all the requests in between.
- After reaching the other end, head reverses its direction.
- It then returns to the starting end without servicing any request in between.
- The same process repeats.

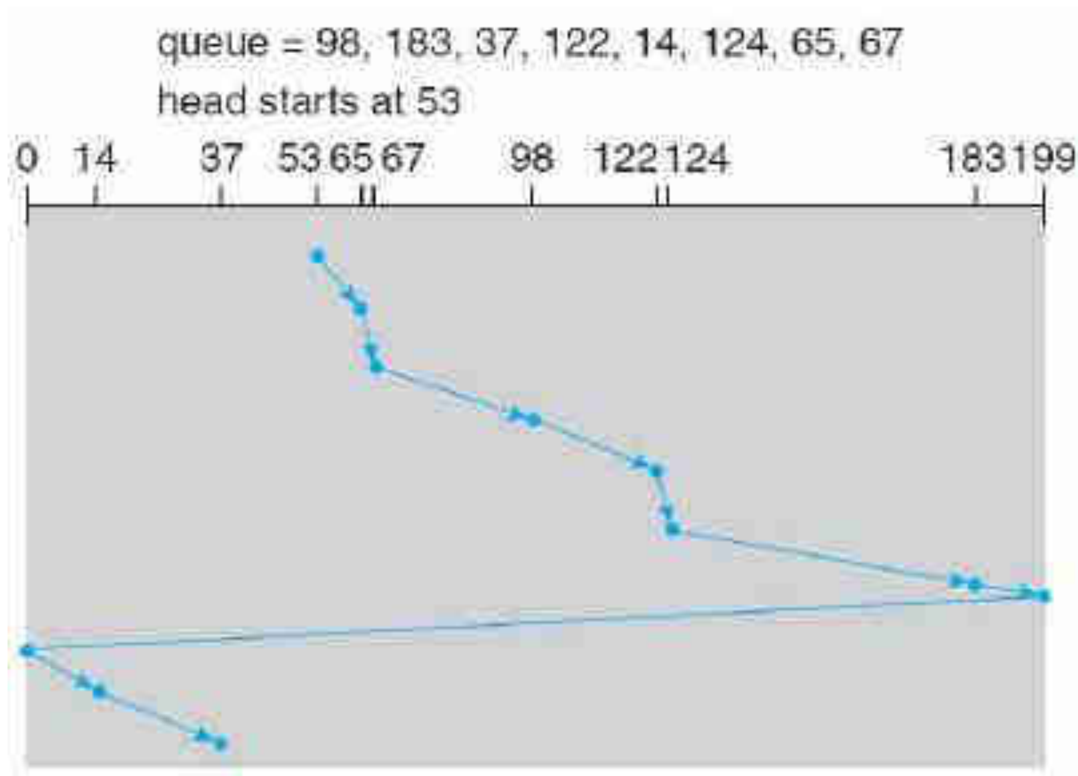
Advantages:

- The waiting time for the cylinders just visited by the head is reduced as compared to the SCAN Algorithm.
- It provides uniform waiting time.
- It provides better response time.

Disadvantages:

- It causes more seek movements as compared to SCAN Algorithm.
- It causes the head to move till the end of the disk even if there are no requests to be serviced.

Example 1: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. The C-SCAN scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. Find the total head movement (in number of cylinders) incurred while servicing these requests.



10.4 C-SCAN Disk Scheduling.

Total head movements incurred while servicing these requests =

(65-53) + (67-65) + (98-67) + (122-98) + (124-122) + (183-124) + (199-183) + (199-0) + (14-0)+ (37-14) =

12 + 2 + 31 + 24 + 2 + 59 + 16 + 199 +14+ 23 = 382

Example 2: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The C-SCAN scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. Find the total head movement (in number of cylinders) incurred while servicing these requests.

10.2.5 LOOK Scheduling

- LOOK Algorithm is an improved version of the SCAN Algorithm.
- Head starts from the first request at one end of the disk and moves towards the last request at the other end servicing all the requests in between.
- After reaching the last request at the other end, head reverses its direction.
- It then returns to the first request at the starting end servicing all the requests in between.
- The same process repeats.

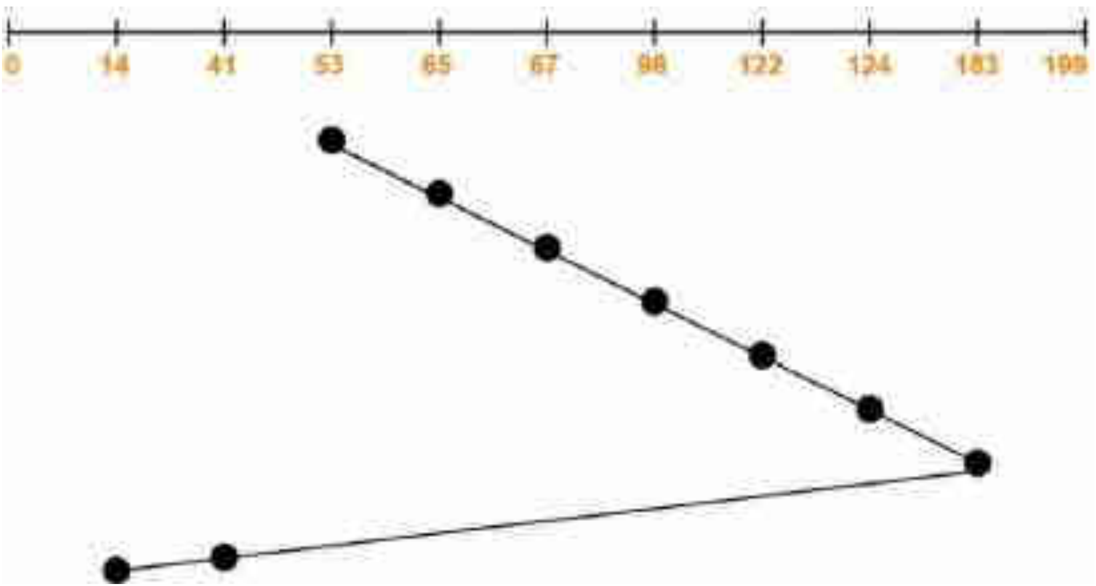
Advantages:

- It does not causes the head to move till the ends of the disk when there are no requests to be serviced.
- It provides better performance as compared to SCAN Algorithm.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

Disadvantages:

- There is an overhead of finding the end requests.
- It causes long waiting time for the cylinders just visited by the head.

Example 1: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The LOOK scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. Find the total head movement (in number of cylinders) incurred while servicing these requests.



10.5 LOOK Scheduling.

Total head movements incurred while servicing these requests = (65 -53) + (67 - 65) + (98 – 67) + (122 - 98) + (124 - 122) + (183 - 124) + (183 - 41) + (41 - 14)

= 12 + 2 + 31 + 24 + 2 + 59 + 142 + 27 = 299

Example 2: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. The LOOK scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. Find the total head movement (in number of cylinders) incurred while servicing these requests.

10.2.6 C-LOOK Scheduling

- Circular-LOOK Algorithm is an improved version of the LOOK Algorithm.
- Head starts from the first request at one end of the disk and moves towards the last request at the other end servicing all the requests in between.
- After reaching the last request at the other end, head reverses its direction.
- It then returns to the first request at the starting end without servicing any request in between.
- The same process repeats.

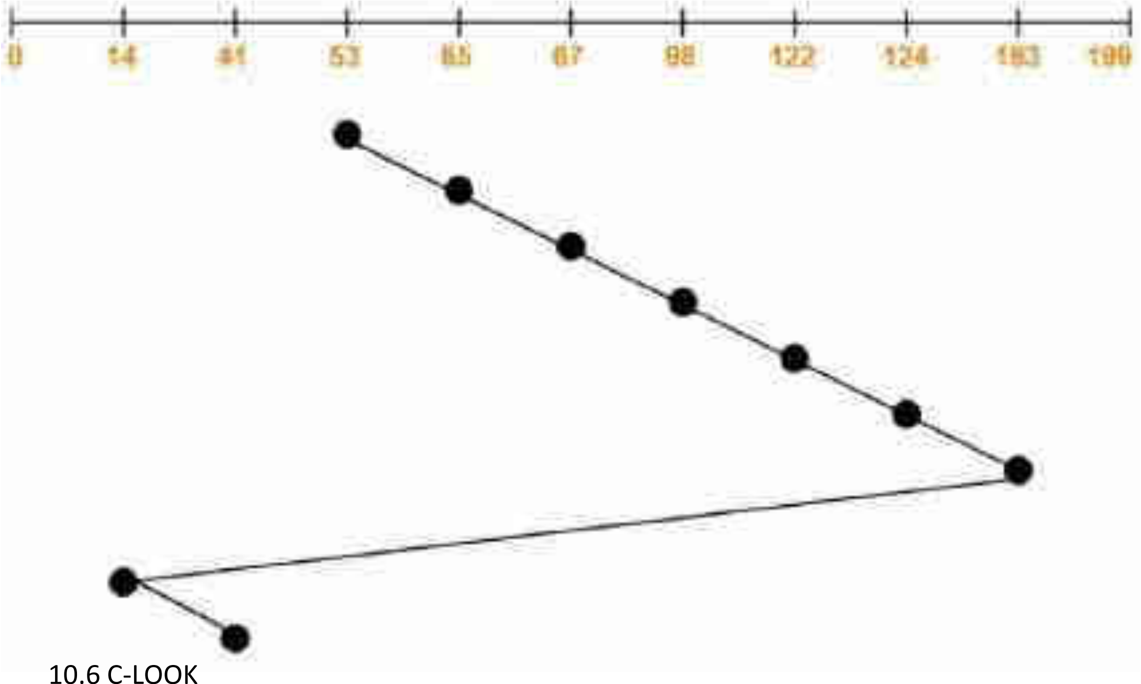
Advantages:

- It does not causes the head to move till the ends of the disk when there are no requests to be serviced.
- It reduces the waiting time for the cylinders just visited by the head.
- It provides better performance as compared to LOOK Algorithm.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

Disadvantages:

- There is an overhead of finding the end requests.

Example 1: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. Find the total head movement (in number of cylinders) incurred while servicing these requests.



10.6 C-LOOK

Total head movements incurred while servicing these requests
= (65 - 53) + (67 - 65) + (98 - 67) + (122 - 98) + (124 - 122) + (183 - 124) + (183 - 14) + (41 - 14)
= 12 + 2 + 31 + 24 + 2 + 59 + 169 + 27 = 326

Example 2: Consider a disk queue with requests for I/O to blocks on cylinders98, 183, 37, 122, 14, 124, 65, 67. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. Find the total head movement (in number of cylinders) incurred while servicing these requests.

10.3 SWAP-SPACE MANAGEMENT

- Swapping is a memory management scheme in which any process can be temporarily swapped from main memory to secondary memory so that the main memory can be made available for other processes. It is used to improve main memory utilization. In secondary memory, the place where the swapped-out process is stored is called swap space.

- The purpose of the swapping in operating system is to access the data present in the hard disk and bring it to RAM so that the application programs can use it. The thing to remember is that swapping is used only when data is not present in RAM.

- Although the process of swapping affects the performance of the system, it helps to run larger and more than one process. This is the reason why swapping is also referred to as memory compaction.

- The concept of swapping has divided into two more concepts: Swap-in and Swap-out.

- Swap-out is a method of removing a process from RAM and adding it to the hard disk.

- Swap-in is a method of removing a program from a hard disk and putting it back into the main memory or RAM.

Advantages of Swapping:

1. It helps the CPU to manage multiple processes within a single main memory.
2. It helps to create and use virtual memory.
3. Swapping allows the CPU to perform multiple tasks simultaneously. Therefore, processes do not have to wait very long before they are executed.
4. It improves the main memory utilization.

Disadvantages of Swapping:

1. If the computer system loses power, the user may lose all information related to the program in case of substantial swapping activity.
2. If the swapping algorithm is not good, the composite method can increase the number of Page Fault and decrease the overall processing performance.

10.3.1 Swap-Space: The area on the disk where the swapped-out processes are stored is called swap space.

10.3.2 Swap-Space Management: Swap-Swap management is another low-level task of the operating system. Disk space is used as an extension of main memory by the virtual memory. As we know the fact that disk access is much slower than memory access, In the swap-space management we are using disk space, so it will significantly decreases system performance. Basically, in all our systems we require the best throughput, so the goal of this swap-space implementation is to provide the virtual memory the best throughput. In these article, we are going to discuss how swap space is used, where swap space is located on disk, and how swap space is managed.

10.3.3 Swap-Space Use: Swap-space is used by the different operating-system in various ways. The systems which are implementing swapping may use swap space to hold the entire process. Paging systems may simply store pages that have been pushed out of the main memory.

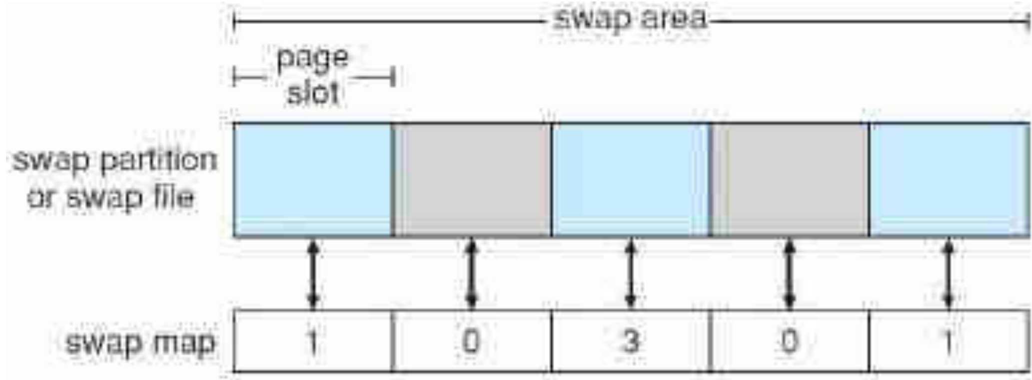
- Following table shows different system using amount of swap space:

System	Swap-Space
Solaris	Equal amount of Physical Memory
Linux	Double the amount of Physical Memory

- Solaris, setting swap space equal to the amount by which virtual memory exceeds page-able physical memory. In the past Linux has suggested setting swap space to double the amount of physical memory. Today, this limitation is gone, and most Linux systems use considerably less swap space.

10.3.4 Swap-Space Use Example:

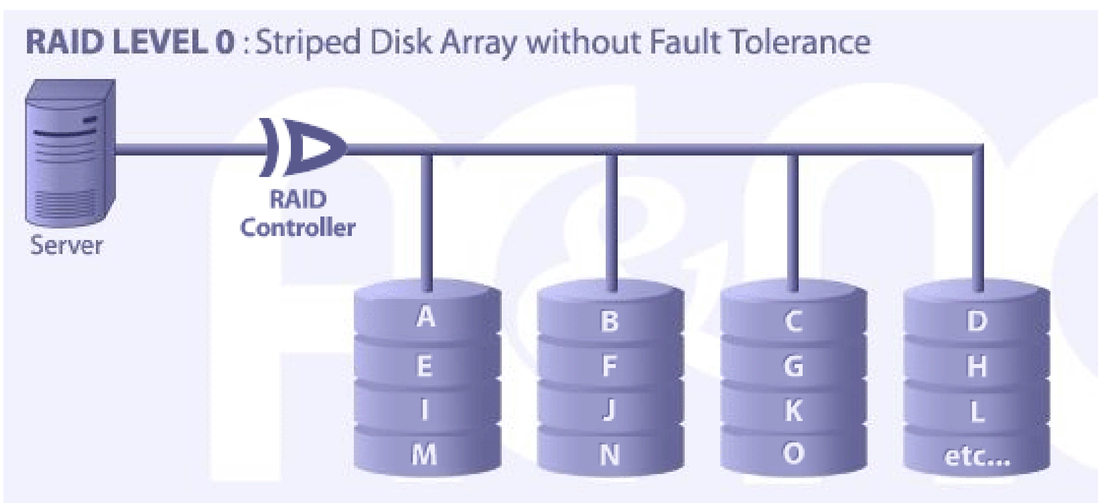
- Linux allows one or more swap areas to be established.
- A swap area may be in either a swap file on a regular file system or a dedicated swap partition.
- Each swap area consists of a series of 4-KB page slots, which are used to hold swapped pages.
- Associated with each swap area is a swap map-an array of integer counters, each corresponding to a page slot in the swap area.
- If the value of a counter is 0, the corresponding page slot is available.
- Values greater than 0 indicate that the page slot is occupied by a swapped page. For example , a value of 3 indicates that the swapped page is mapped to three different processes



10.7 The data structures for swapping on Linux systems.

10.4 RAID VVIMP

- Redundant Array Independent Disk



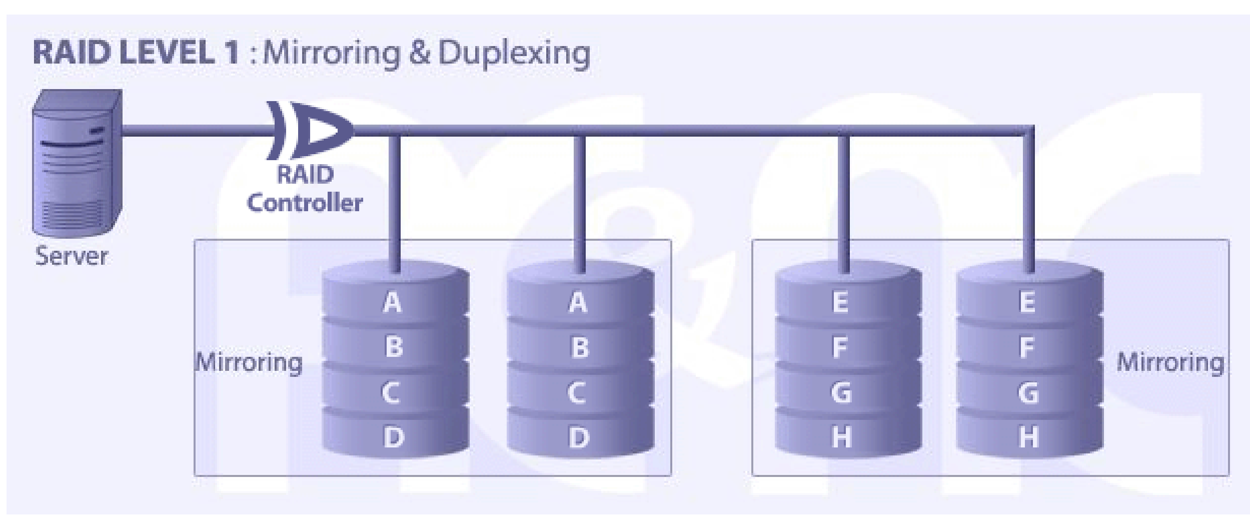
- RAID Level 0 requires a minimum of 2 drives to implement.

Characteristics & Advantages:

1. RAID 0 implements a striped disk array, the data is broken down into blocks and each block is written to a separate disk drive.
2. I/O performance is greatly improved by spreading the I/O load across many channels and drives.
3. Best performance is achieved when data is striped across multiple controllers with only one drive per controller.
4. No parity calculation overhead is involved.
5. Very simple design.
6. Easy to implement.

Disadvantages:

1. Not a "True" RAID because it is NOT fault-tolerant.
2. The failure of just one drive will result in all data in an array being lost.
3. Should never be used in mission critical environments.



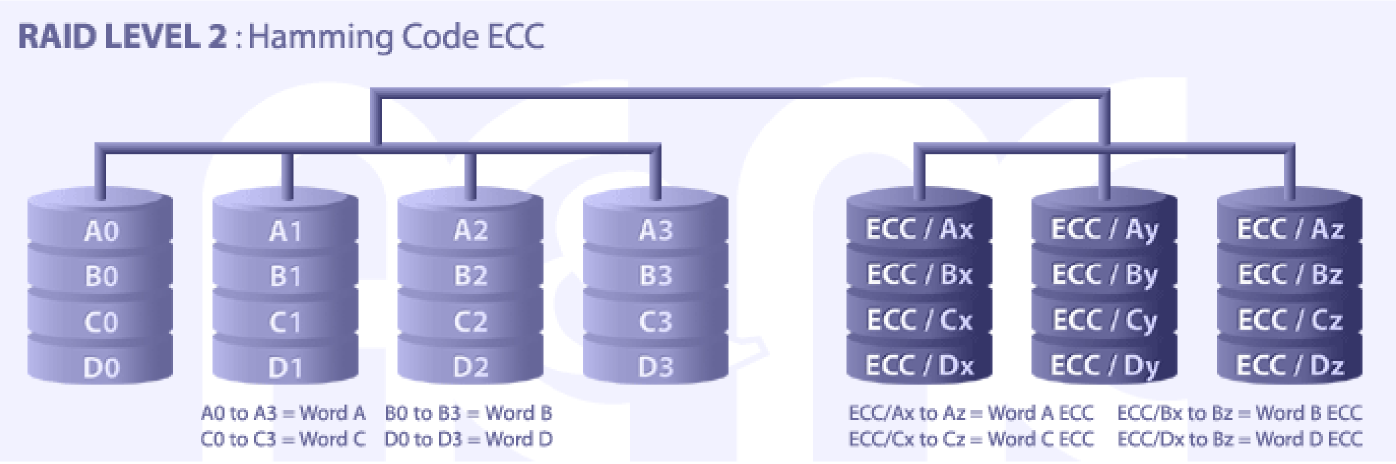
- For Highest performance, the controller must be able to perform two concurrent separate reads per mirrored pair or two duplicate Writes per mirrored pair.
- RAID Level 1 requires a minimum of 2 drives to implement.

Characteristics & Advantages:

1. One Write or two Reads possible per mirrored pair.
2. Twice the Read transaction rate of single disks, same Write transaction rate as single disks.
3. 100% redundancy of data means no rebuild is necessary in case of a disk failure, just a copy to the replacement disk.
4. Transfer rate per block is equal to that of a single disk.
5. Under certain circumstances, RAID 1 can sustain multiple simultaneous drive failures.

Disadvantages:

1. Highest disk overhead of all RAID types (100%) – inefficient
2. Typically the RAID function is done by system software, loading the CPU/Server and possibly degrading throughput at high activity levels. Hardware implementation is strongly recommended
3. May not support hot swap of failed disk when implemented in "software"

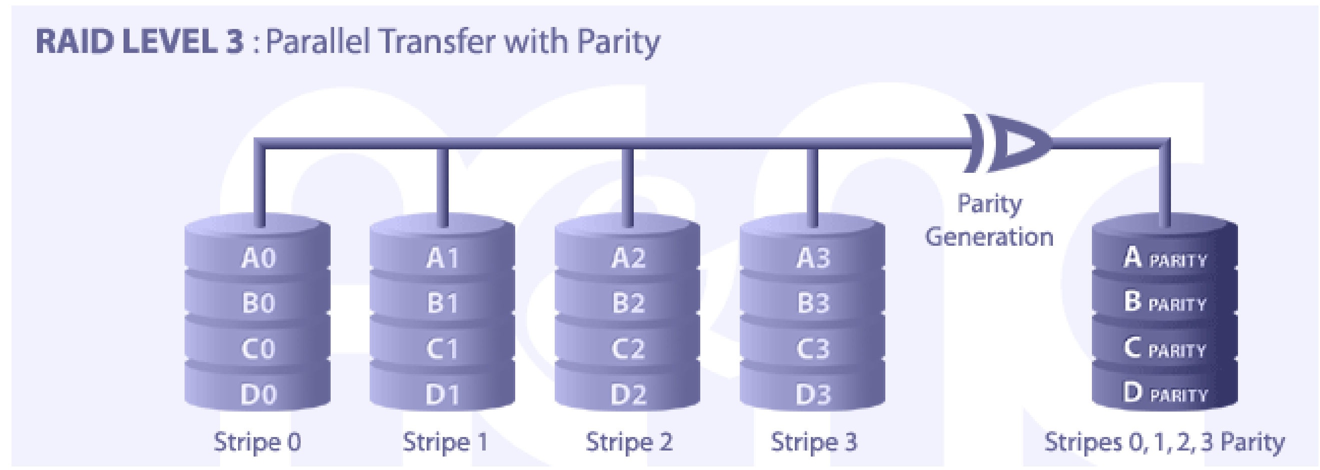


- Characteristics & Advantages:

1. Each bit of data word is written to a data disk drive (4 in this example: 0 to 3).
2. Each data word has its Hamming Code ECC word recorded on the ECC disks.
3. On Read, the ECC code verifies correct data or corrects single disk errors.

- Disadvantages:

1. Very high ratio of ECC disks to data disks with smaller word sizes – inefficient.
2. Entry level cost very high - requires very high transfer rate requirement to justify
3. Transaction rate is equal to that of a single disk at best (with spindle synchronization)
4. No commercial implementations exist / not commercially viable.



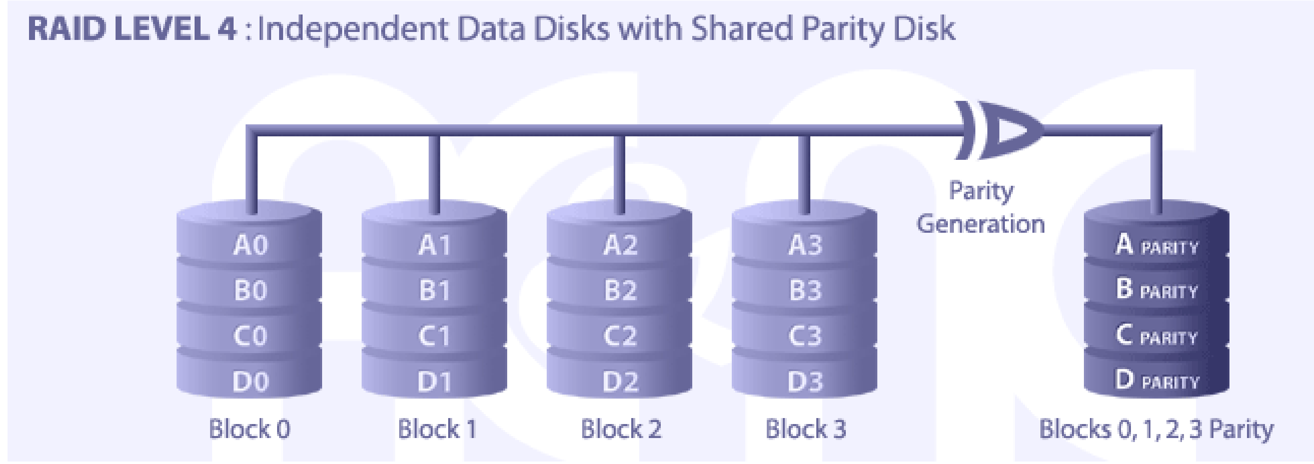
- The data block is subdivided ("striped") and written on the data disks. Stripe parity is generated on Writes, recorded on the parity disk and checked on Reads.
- RAID Level 3 requires a minimum of 3 drives to implement.

- Characteristics & Advantages:

1. Very high Read data transfer rate.
2. Very high Write data transfer rate.
3. Disk failure has an insignificant impact on throughput.
4. Low ratio of ECC (Parity) disks to data disks means high efficiency.

Disadvantages:

1. Transaction rate equal to that of a single disk drive at best (if spindles are synchronized).
2. Controller design is fairly complex.
3. Very difficult and resource intensive to do as a "software" RAID.



1. Each entire block is written onto a data disk. Parity for same rank blocks is generated on Writes, recorded on the parity disk and checked on Reads.

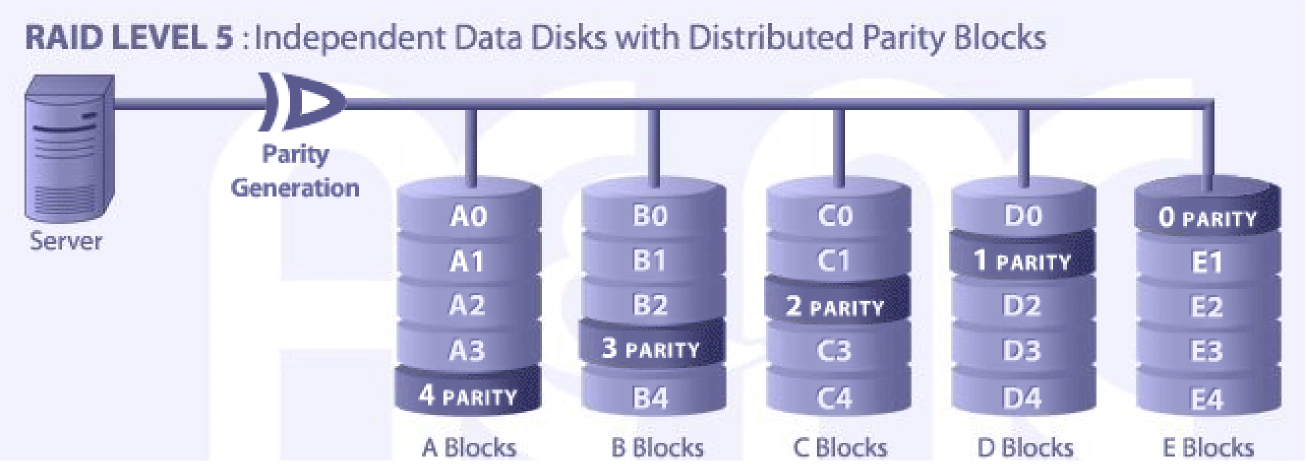
- RAID Level 4 requires a minimum of 3 drives to implement.

- Characteristics & Advantages:

1. Very high Read data transaction rate.
2. Low ratio of ECC (Parity) disks to data disks means high efficiency.High aggregate Read transfer rate.

- Disadvantages:

2. Quite complex controller design.
3. Worst Write transaction rate and Write aggregate transfer rate.
4. Difficult and inefficient data rebuild in the event of disk failure.
5. Block Read transfer rate equal to that of a single disk.



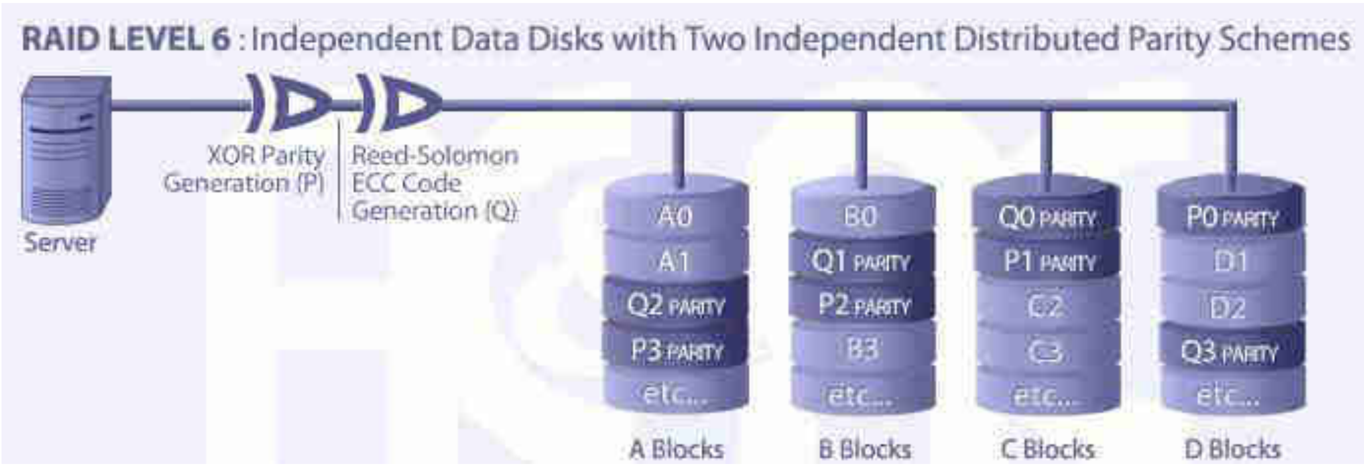
- Each entire data block is written on a data disk; parity for blocks in the same rank is generated on Writes, recorded in a distributed location and checked on Reads.
- RAID Level 5 requires a minimum of 3 drives to implement.

- Characteristics & Advantages:

1. Highest Read data transaction rate. Medium Write data transaction rate.
2. Low ratio of ECC (Parity) disks to data disks means high efficiency.
3. Good aggregate transfer rate.

- Disadvantages:

1. Disk failure has a medium impact on throughput.
2. Most complex controller design.
3. Difficult to rebuild in the event of a disk failure (as compared to RAID level 1).
4. Individual block data transfer rate same as single disk.



- Two independent parity computations must be used in order to provide protection against double disk failure. Two different algorithms are employed to achieve this purpose.
- RAID Level 6 requires a minimum of 4 drives to implement

- Characteristics & Advantages:

1. RAID 6 is essentially an extension of RAID level 5 which allows for additional fault tolerance by using a second independent distributed parity scheme (dual parity).
2. Data is striped on a block level across a set of drives, just like in RAID 5, and a second set of parity is calculated and written across all the drives; RAID 6 provides for an extremely high data fault tolerance and can sustain multiple simultaneous drive failures.
3. Perfect solution for mission critical applications.

- Disadvantages:

1. More complex controller design.
2. Controller overhead to compute parity addresses is extremely high.
3. Write performance can be brought on par with RAID Level 5 by using a custom ASIC for computing Reed-Solomon parity.
4. Requires N+2 drives to implement because of dual parity scheme.

CHAPTER 11
FILE SYSTEM INTERFACE

11.1 FILE CONCEPT

File: Collection of interrelated records.

11.1.1 File Attributes

- **Name:** The symbolic file name is the only information kept in human readable form.
- **Identifier:** This unique tag, usually a number, identifies the file within the file system.
- **Type:** This information is needed for systems that support different types of files.
- **Location:** This information is a pointer to a device and to the location of the file on that device.
- **Size:** The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- **Protection:** Access-control information determines who can do reading, writing, executing, and so on.
- **Time, date, and user identification:** This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

11.1.2 File Operations IMP

- File: A file is collection of interrelated records.

Create	Creates a file
Read (or) Get	Reads the content of the file.
Write	Writes data to file
Open	Prepares the file for reading (or) writing.
Truncate	Erase the contents of a file but keeps its attributes
Reset	Sets file pointer to the beginning of the file.
Find (or) Locate	Searches the first record that satisfies the search condition.
Find Next	Searches for next record into the file that satisfies the search condition.
Delete	Deletes the current record.
Modify	Modifies field value for current record.
Close	Completes the file access.
Scan	If the file has just been opened (or) reset, scan returns the first record, otherwise it returns the next record.

Record at a time operations	
Reset	Sets the file pointer to the beginning of the file.
Insert	Insert new record in the file

Set at a time operations	
Find All	Locates all the records in the file that satisfy a search condition.
Find Ordered	Retrieves all the records in the file in some specified order.
Reorganize	Starts the reorganize process i.e. organization of data file into blocks, records.

11.1.3 File Types

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

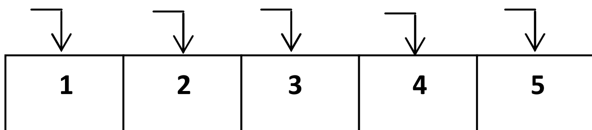
11.2 FILE ACCESS METHODS VIMP

11.2.1 Sequential Access

- In sequential access, the OS read the file word by word. A pointer is maintained which initially points to the base address of the file. If the user wants to read first word of the file then the pointer provides that word to the user and increases its value by 1 word. This process continues till the end of the file.
- Most of the files such as text files, audio files, video files, etc need to be sequentially accessed.

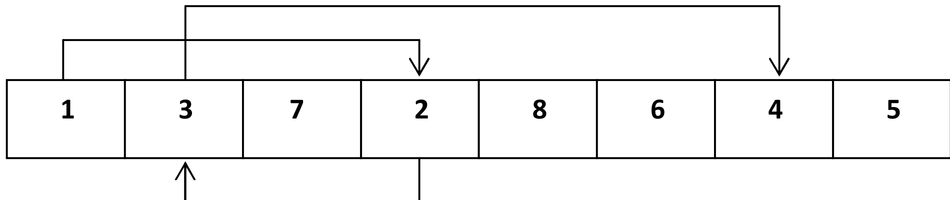
Key points:

- Data is accessed one record right after another record in an order.
- When we use read command, it move ahead pointer by one.
- When we use write command, it will allocate memory and move the pointer to the end of the file.
- Such a method is reasonable for tape.



11.2.2 Direct Access

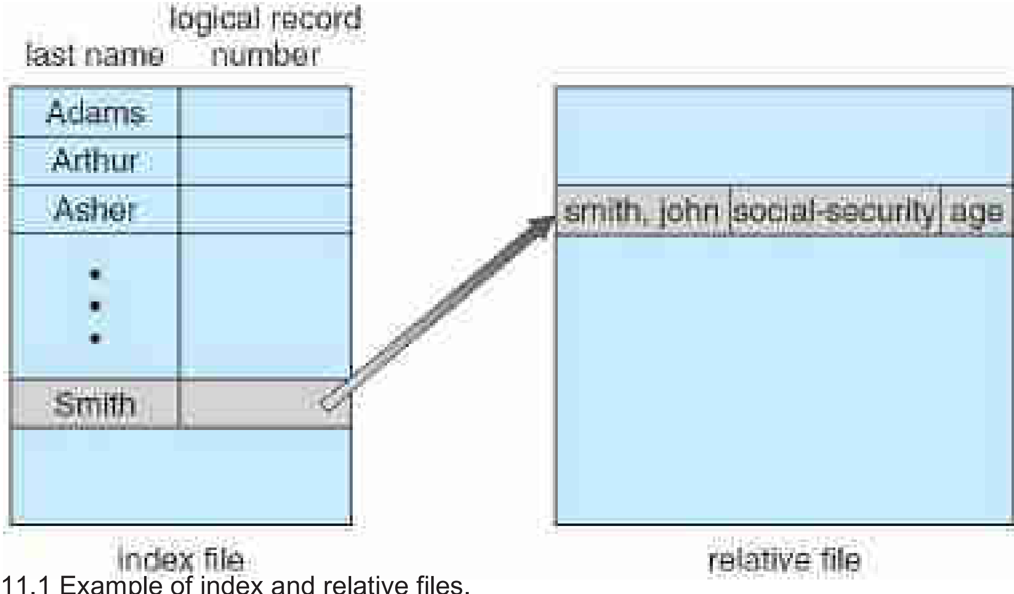
- Another method is direct access (or relative access).
- Here, a file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.



11.2.3 Index Sequential Method

- A particular record can be accessed by its index. The index is nothing but the address of a record in the file.

- In index accessing, searching in a large database became very quick and easy but we need to have some extra space in the memory to store the index value.



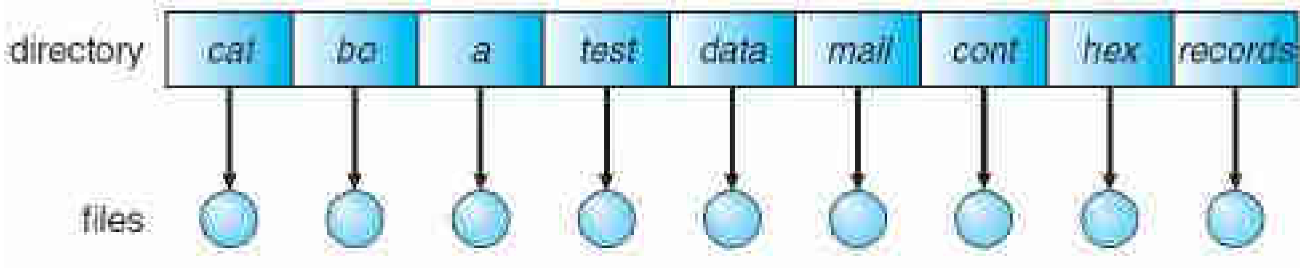
11.1 Example of index and relative files.

11.3 DIRECTORY OVER VIEW IMP

- Directory: A directory is a container that is used to contain folders and files.
- It organizes files and folders in a hierarchical manner.

11.3.1 Single-Level Directory

- All files are contained in the same directory which makes it easy to support and understand.



11.2 Single-level directory.

Advantages:

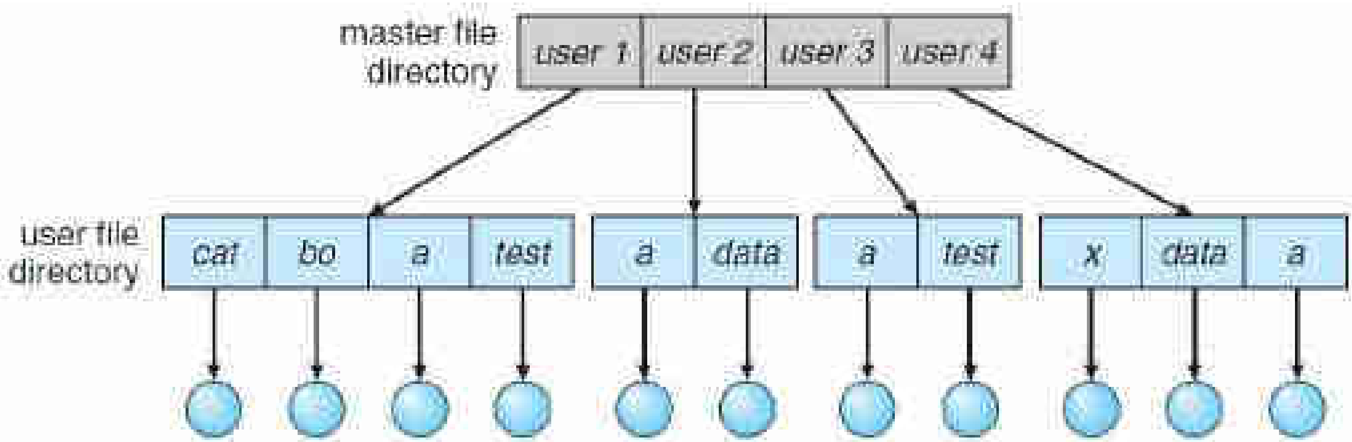
- All files are contained in the same directory, which is easy to support and understand.
- If the files are smaller in size, searching will become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

Disadvantages:

- There may chance of name collision because two files can not have the same name.
- Searching will become time taking if the directory is large.
- This can not group the same type of files together.

11.3.2 Two Level Directory

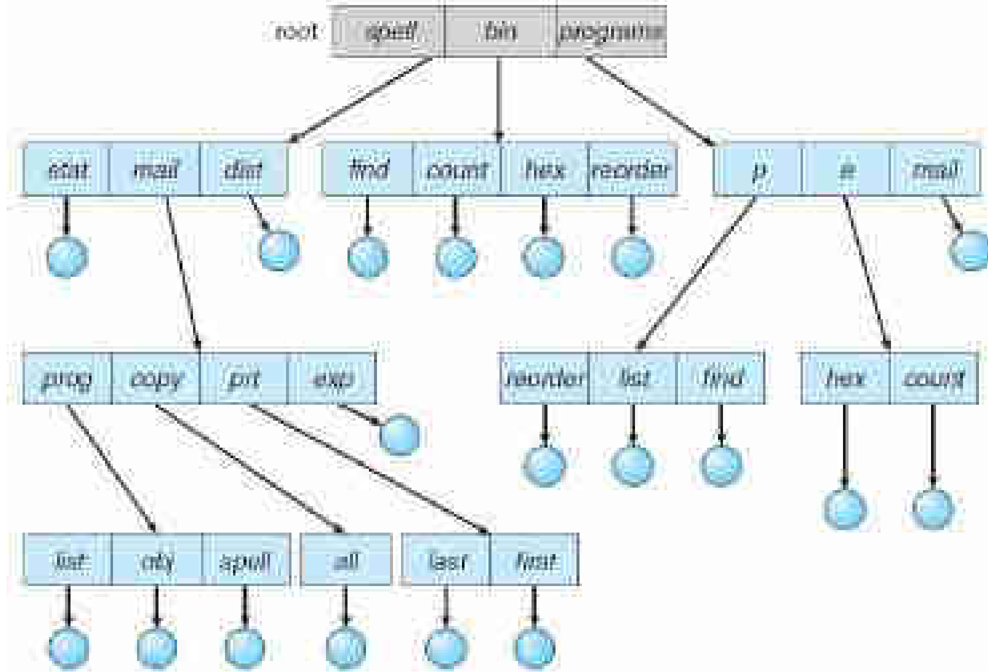
- In the two-level directory structure, Master File Directory contain list of Users.
- Each user has User File Directory.



11.3 Two Level Directory.

11.3.5 Tree-Structured Directories

- This generalization allows the user to create their own subdirectories and to organize their files accordingly.



11.4 Tree Structured Directory.

A tree structure is the most common directory structure. The tree has a root directory, and every file in the system has a unique path.

Advantages:

- Very general, since full pathname can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy, we can use both absolute paths as well as relative.

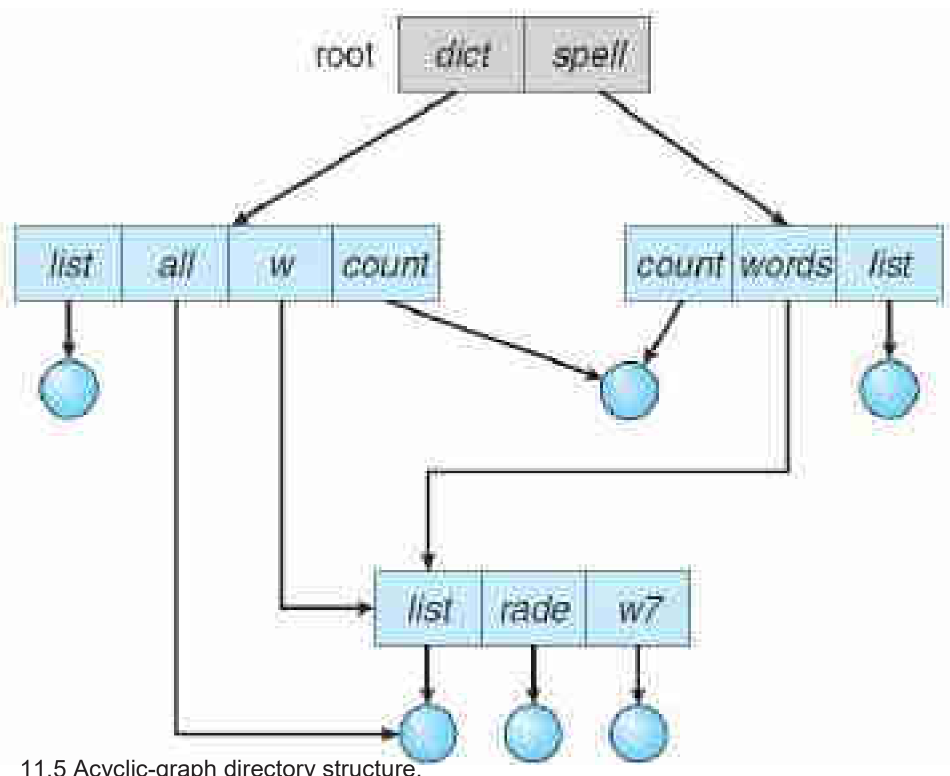
Disadvantages:

- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We can't share files.
- It is inefficient, because accessing a file may go under multiple directories.

11.3.6 Acyclic-Graph Directories

- An acyclic graph is a graph with no cycles and allows directories to share subdirectories and files.

- The same file or subdirectory may be in two different directories. The acyclic graph is a generalization of the tree-structured directory scheme.



11.5 Acyclic-graph directory structure.

- It is used in the situation like when two programmers are working on a joint project and they need to access files. The associated files are stored in a subdirectory, separating them from other projects and files of other programmers since they are working on a joint project so they want the subdirectories to be into their own directories. The common subdirectories should be shared. So here we use Acyclic directories.

- It is the point to note that the shared file is not the same as the copy file. If any programmer makes some changes in the subdirectory it will reflect in both subdirectories.

Advantages:

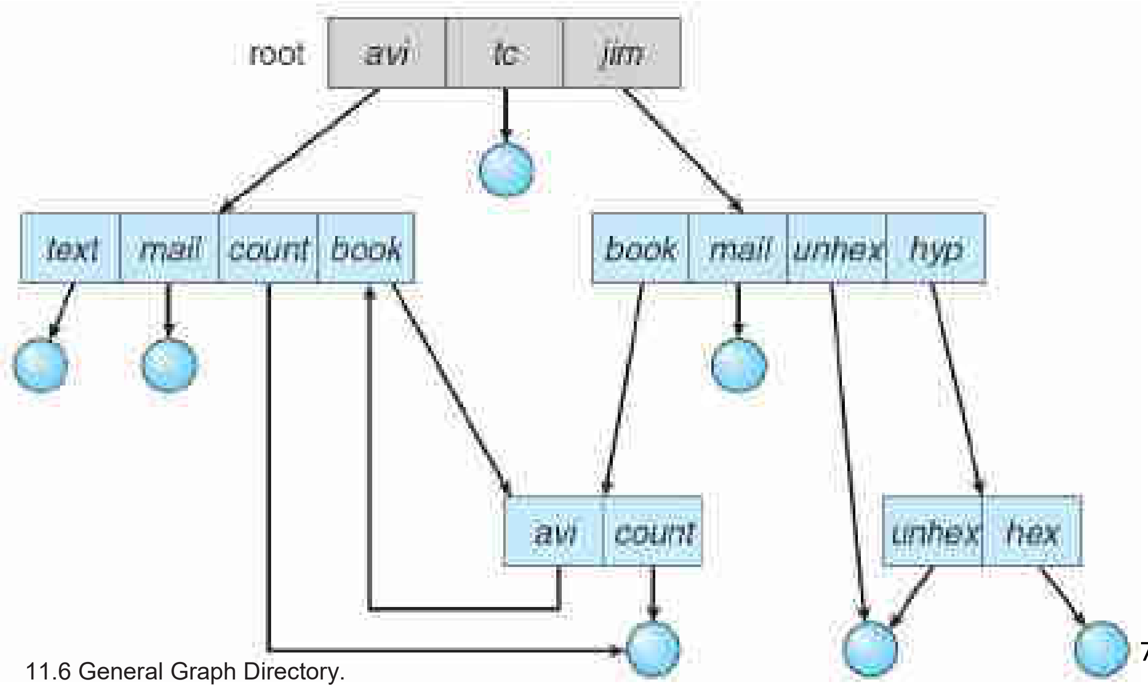
- We can share files.
- Searching is easy due to different-different paths.

Disadvantages:

- We share the files via linking, in case deleting it may create the problem.
- If the link is a soft link then after deleting the file we left with a dangling pointer.
- In the case of a hard link, to delete a file we have to delete all the references associated with it.

11.3.7 General Graph Directory Structure

- In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory.
- The main problem with this kind of directory structure is to calculate the total size or space that has been taken by the files and directories.



11.6 General Graph Directory.

Advantages:

- It allows cycles.
- It is more flexible than other directories structure.

Disadvantages:

- It is more costly than others.
- It needs garbage collection.

11.4 FILE SYSTEM MOUNTING

- Mounting: Attaching the files to file system.
- Operating system makes files and directories available at a specific path so that we can make use of all those files and directories at that specific path.
- Generally a computer can have partitions C,D, E etc.
- Generally C file system contains software's and D, E file system contains user programs or user related data.
- The root (/) file system is always mounted. Any other file system can be connected or disconnected from the root (/) file system.

Example:

Let us assume that we have turbo c in our system. By using turbo C we can execute file system.

C:\TURBO C

We want execute TURBO C from D:\

We have to attach the C:\ to D:\

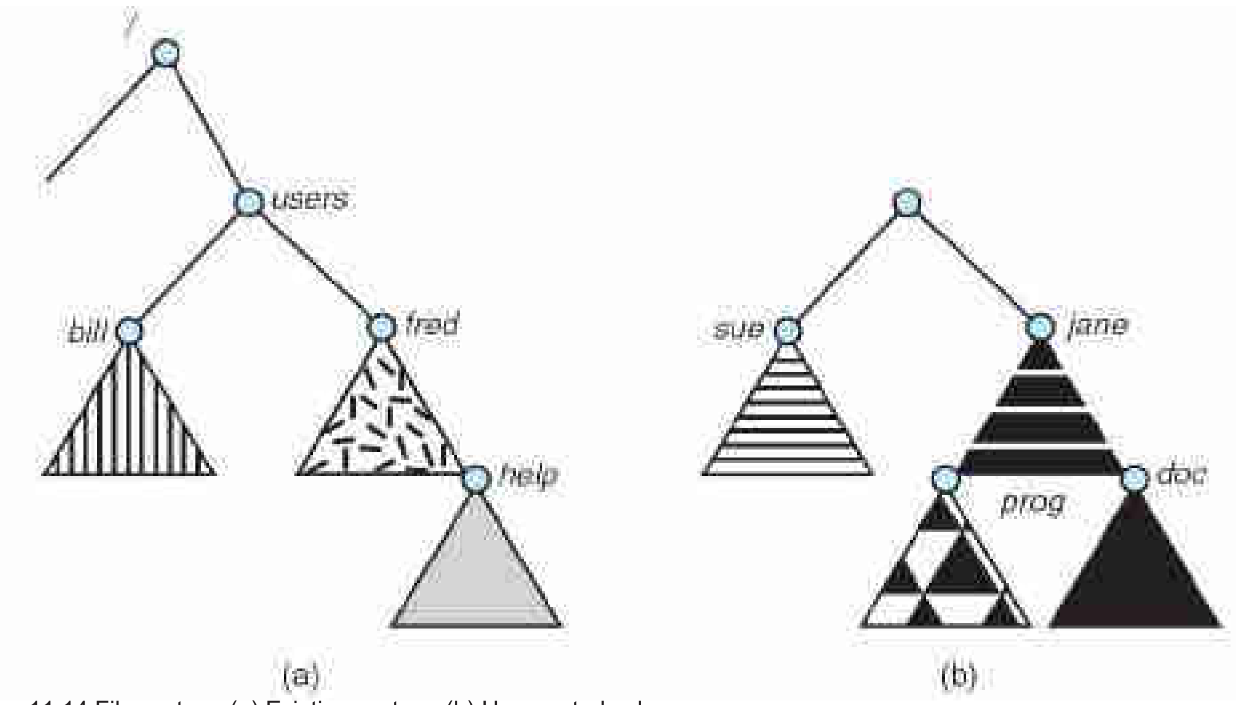
D:\MOUNT C:\TURBO C D:\

After executing the command we can execute C file system from D:\

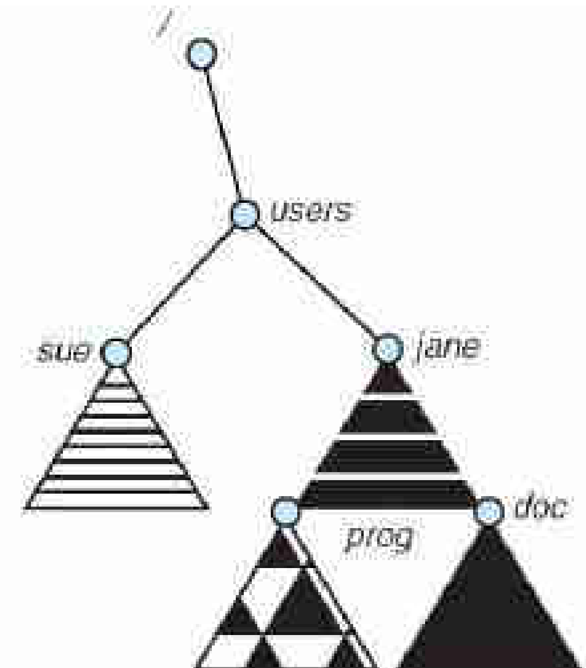
- Unmounting: Detaching file from file system.

D:\UMOUNT C:\TURBO C D:\

Mount Point: Mount point specifies the location at which the file system is attached.



11.14 File system. (a) Existing system. (b) Unmounted volume.



11.15 Mount Point.

11.5 PROTECTION

- **Protection** actually refers to a mechanism that restricts the access of the resources specified by a computer system to programs, processes, or users.

Need of System Protection:

- To prevent access of unauthorized users.
- To ensure that resources are used only as specified policy by each active program or mechanism in the scheme.
- To improve reliability by detecting latent errors.

11.6.1 Types of Access

- **Read**: Read from the file.
- **Write**: Write or rewrite the file.
- **Execute**: Load the file into memory and execute it.
- **Append**: Write new information at the end of the file.
- **Delete**: Delete the file and free its space for possible reuse.
- **List**: List the name and attributes of the file.

11.6.2 Access Control

- **Owner**. The user who created the file is the owner.
- **Group**. A set of users who are sharing the file and need similar access is a group, or work group.
- **Universe**. All other users in the system constitute the universe.
- The characters are pretty easy to remember.

r = read permission
w = write permission
x = execute permission
– = no permission

- File access permissions can also be changed by a numerical (octal) chmod specification. Read permission is given the value 4, write permission the value 2 and execute permission 1.

```
r w x
4 2 1
```

These values are added together for any one user category:

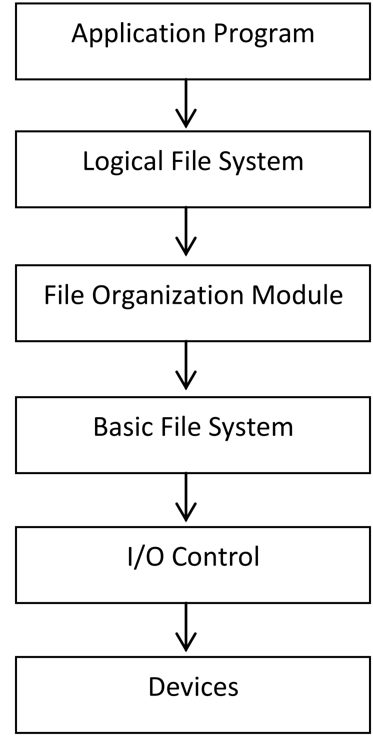
0 = no permissions
1 = execute only
2 = write only
3 = write and execute (1+2)
4 = read only
5 = read and execute (4+1)
6 = read and write (4+2)
7 = read and write and execute (4+2+1)

```
Example:

Access permissions can be expressed as three digits:
                      user    group   others
chmod 640 file1       rw-     r--     ---
chmod 754 file1       rwx     r-x     r--
chmod 664 file1       rw-     rw-     r--
```

CHAPTER12
FILE SYSTEM IMPLEMENTATION

12.1 FILE SYSTEM STRUCTURE

- File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way.
- A file System must be able to store the file, locate the file and retrieve the file.
- Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.



12.1 Layered File System.

- When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the *Meta Data* of the *File and Directory Structure*. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.

- Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.
- Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
- I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

12.2 FILE SYSTEM IMPLEMENTATION

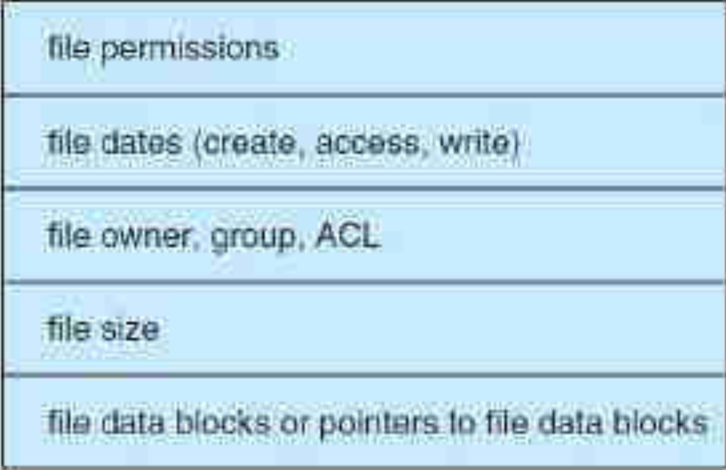
- There are various on disk data structures that are used to implement a file system. This structure may vary depending upon the operating system.

1. **Boot Control Block:** Boot Control Block contains all the information which is needed to boot an operating system from that volume. It is called **boot block** in **UNIX** file system. In **New Technology File System**, it is called the **partition boot sector**.

2. **Volume Control Block:** Volume control block all the information regarding that volume such as number of blocks, size of each block, partition table, pointers to free blocks and free File Control Blocks. In UNIX file system, it is known as **super block**. In NTFS, this information is stored inside master file table.

3. **Directory Structure (Per File System):** A directory structure (per file system) contains file names and pointers to corresponding FCBs. In UNIX, it includes inode numbers associated to file names.

4. **File Control Block:** File Control block contains all the details about the file such as ownership details, permission details, file size,etc. In UFS, this detail is stored in inode. In NTFS, this information is stored inside master file table as a relational database structure.

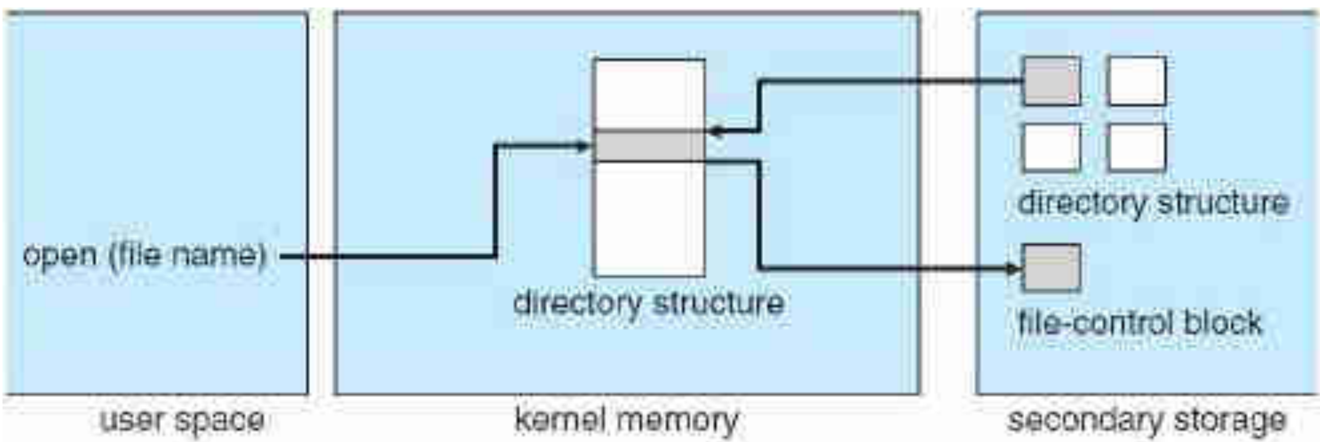


12.2 A typical file-control block.

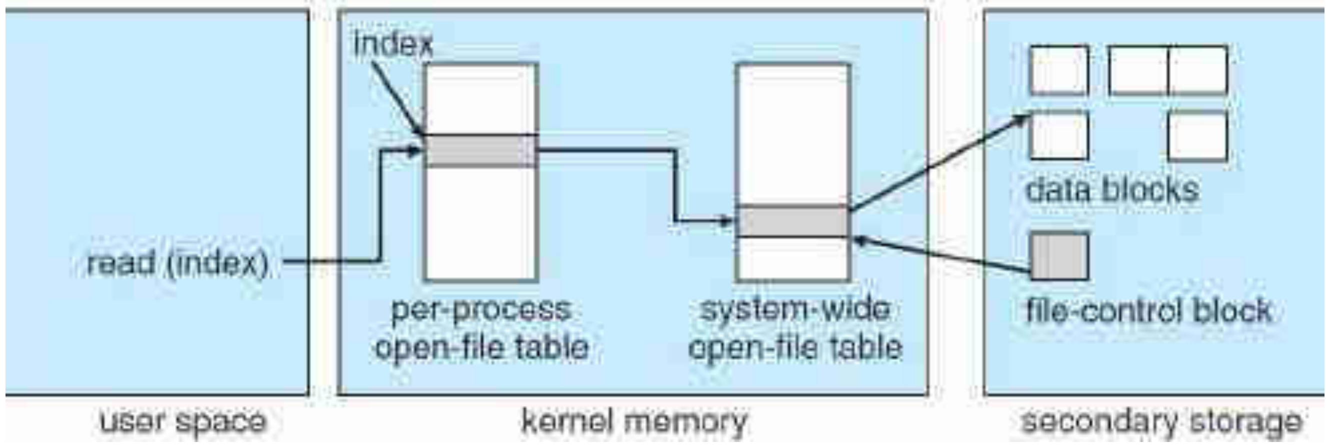
ACL: Access Control List.

12.2.1 In-memory file-system structures.

- They are maintained in main-memory and these are helpful for file system management for caching.



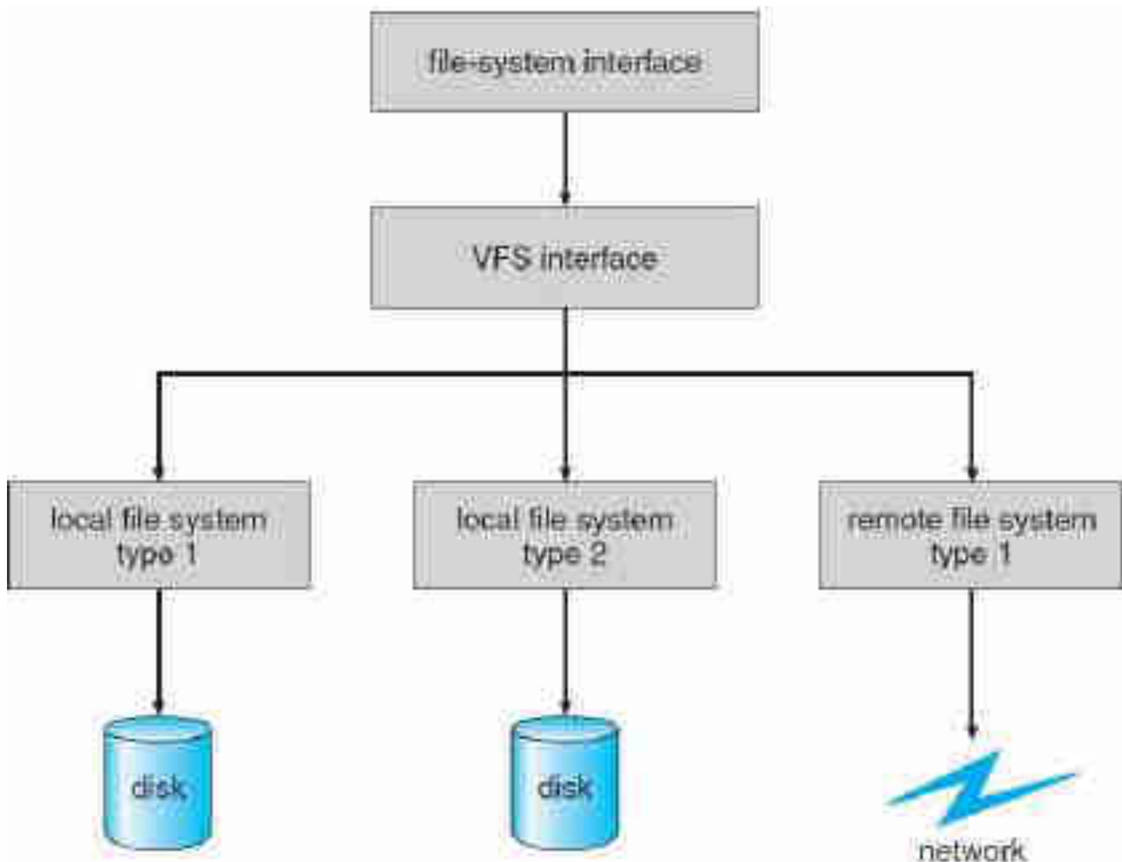
12.3 In-memory file-system structures - File Open



12.4 In-memory file-system structures - File Read

12.2.2 Virtual File Systems

- A virtual file system (VFS) is programming that forms an interface between an operating system's kernel and a more concrete file system. The VFS serves as an abstraction layer that gives applications access to different types of file systems and local and network storage devices.



12.5 Schematic view of a virtual file system.

12.4 ALLOCATION METHODS VIMP

- The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

  1. Contiguous Allocation
  2. Linked Allocation
  3. Indexed Allocation

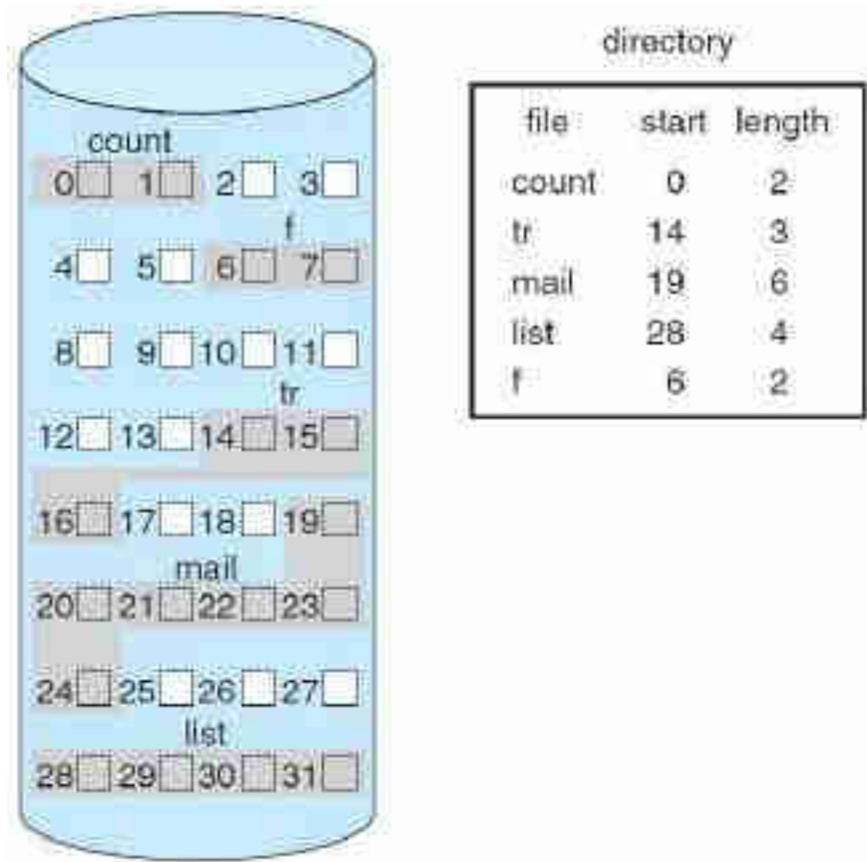
12.4.1 Contiguous Allocation

- In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file

will be *b, b+1, b+2,……b+n-1.* This means that given the starting block address and the length of the file, we can determine the blocks occupied by the file.

The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.



12.6 Contiguous allocation of disk space.

Advantages:

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

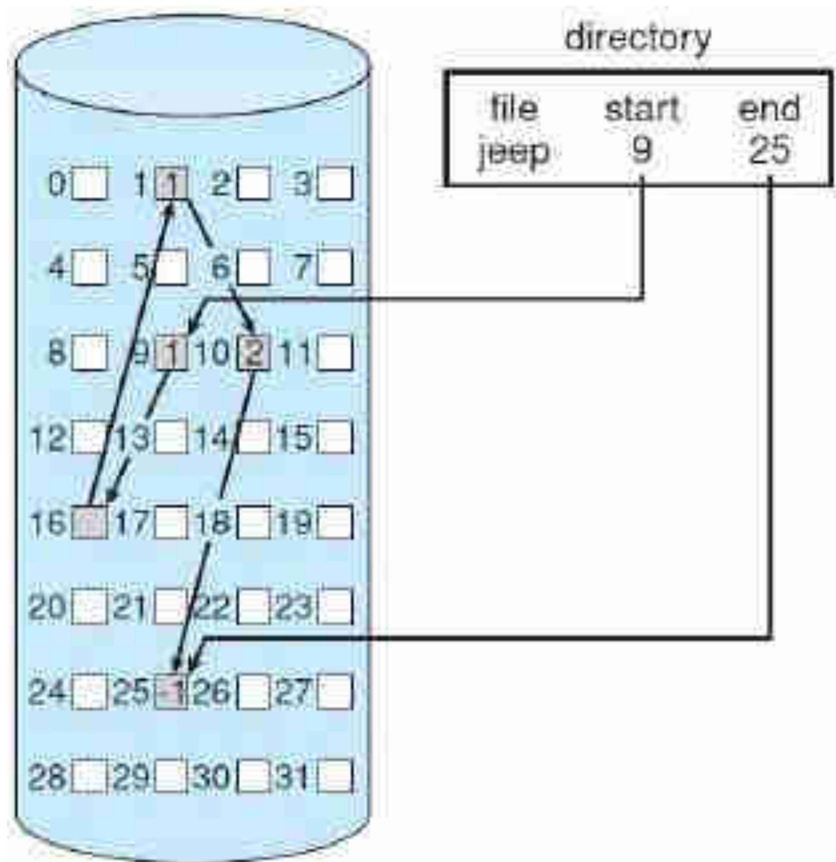
Disadvantages:

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

12.4.2 Linked Allocation

- In this scheme, each file is a linked list of disk blocks which **need not be** contiguous.
- The disk blocks can be scattered anywhere on the disk.

- The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.
- The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



12.7 Linked Allocation of Disk Space.

Advantages:

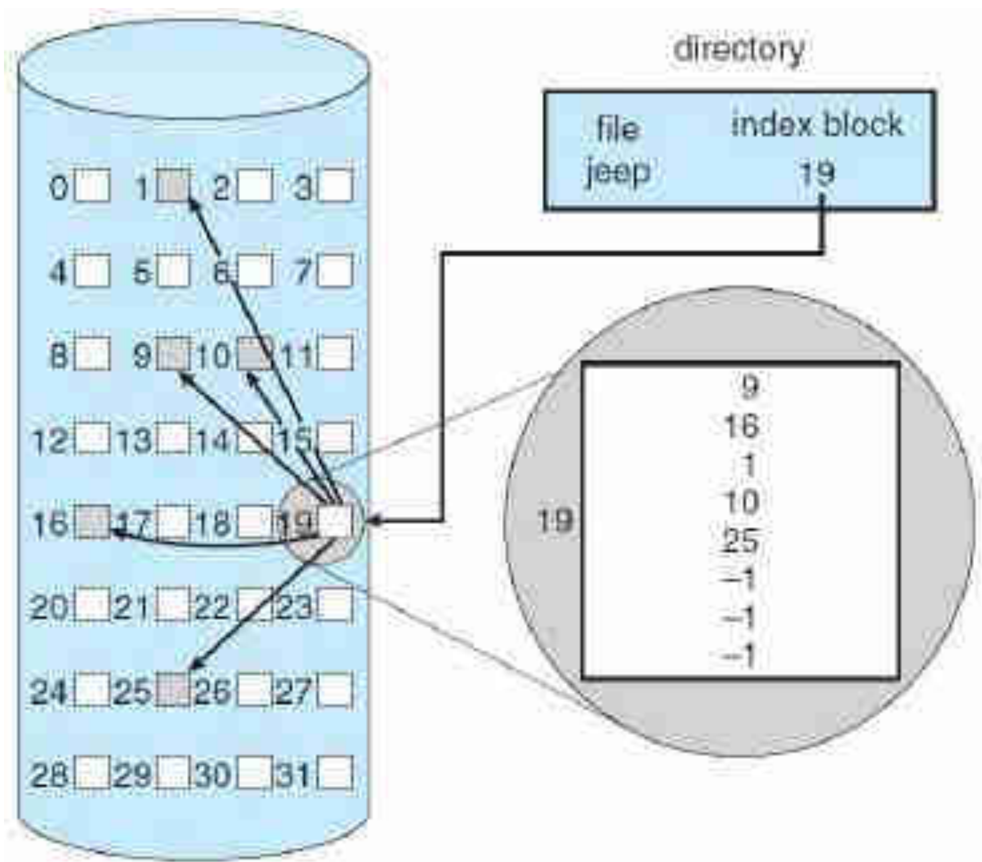
- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

12.4.3 Indexed Allocation

- In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file.
- Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image.



12.8 Indexed Allocation of Disk Space.

Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

12.5 FREE-SPACE MANAGEMENT

The system keeps tracks of the free disk blocks for allocating space to files when they are created.

12.5.1 Bitmap or Bit vector

- A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values 0 and 1.
- 0 indicates that the block is allocated and 1 indicates a free block.

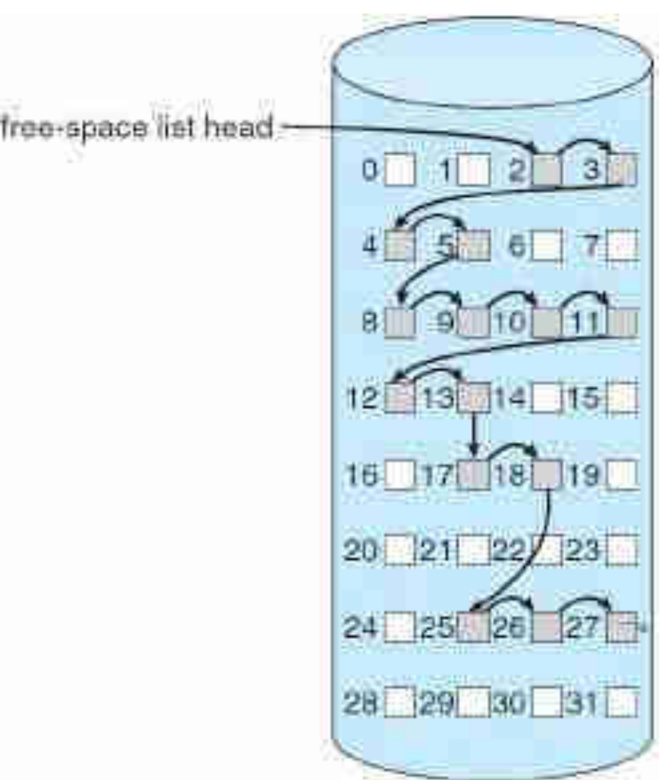
Block 0	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7
0	0	0	0	1	1	1	0

Advantages:

- Simple to understand.

12.5.2 Linked List Allocation

- In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block.
- The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.



12.9 Linked free-space list on disk.

Advantage:

- No wastage of Space.

Disadvantage:

- I/O required for free space list traversal.

12.5.3 Grouping

- This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks.

- Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of next free n blocks.

Advantage:

The addresses of a group of free disk blocks can be found easily.

12.5.4 Counting

This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block.

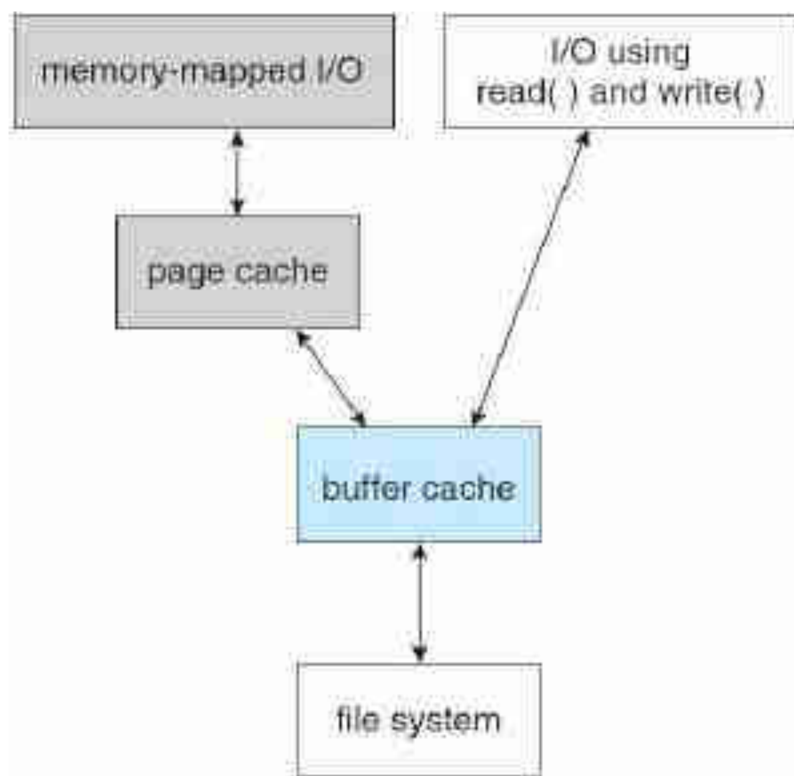
Every entry in the list would contain:

1. Address of first free disk block
2. A number n

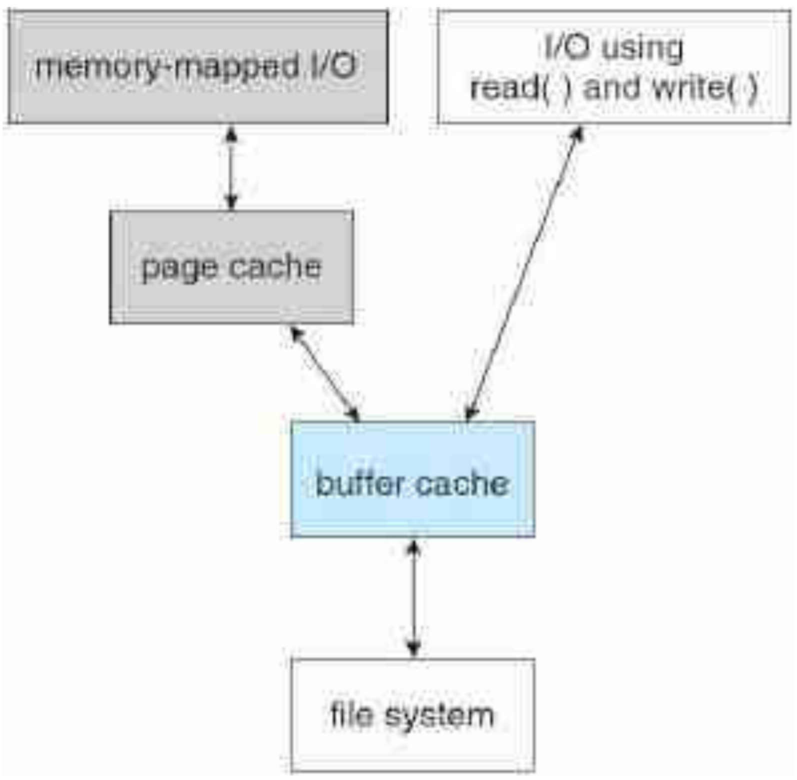
12.6 EFFICIENCY AND PERFORMANCE

- Efficiency: An OS allows the computer system resources to be used efficiently.

- The page cache caches pages of files to optimize file I/O. The buffer cache caches disk blocks to optimize block I/O.



12.10 I/O without a unified buffer.



12.11 I/O using a unified buffer cache.

12.7 RECOVERY

- Files and directories are kept both in main memory and on disk, and care must be taken to ensure that a system failure does not result in loss of data or in data inconsistency.

12.7.1 Consistency Checking

- The consistency checker is a systems program such as **fsck** in UNIX, compares the data in the directory structure with the data blocks on disk and tries to fix any inconsistencies it finds.
- The allocation and free space management algorithms dictate what types of problems the checker can find and how successful it will be in fixing them.
- For instance, if linked allocation is used and there is a link from any block to its next block, then the entire file can be reconstructed from the data blocks, and the directory structure can be recreated.
- The loss of a directory entry on an indexed allocation system can be disastrous, because the data blocks have no knowledge of one another.

12.7.2 Log-Structured File Systems

- A log structured file system (LFS) gives a completely different approach to managing a file system.
- The central idea behind LFS is that blocks are never modified; whenever an operation conceptually modifies a file, the operation instead places a *new* block at the end of the log. Writes always go to the end of the disk.
- For example, suppose I wished to change the first byte of a file. I would create a new copy of the direct block containing that byte, and place it at the end of the log. Since the address of that block has now changed, I would also create a copy of the inode for the file.
- An inode is a data structure in UNIX operating systems that contains important information pertaining to files within a file system.

12.7.3 Backup and Restore

A typical backup schedule may then be as follows:

Day 1: Copy to a backup medium all files from the disk. This is called a **full backup**.

Day 2. Copy to another medium all files changed since day 1. This is an **incremental backup**.

Day 3. Copy to another medium all files changed since day 2.

.

.

.

Day N. Copy to another medium all files changed since day N−1. Then go back to day 1.

LECTURE NOTES PREPARED BY

Dr.T.S.RAVI KIRAN, ASSISTANT PROFESSOR & HOD

DEPARTMENT OF COMPUTER SCIENECE

P.B.SIDDHARHA COLLEE OF ARTS & SCIENECE, VIJAYAWADA, AP, INDIA PIN: 520010

Email: tsravikiran@pbsiddhartha.ac.in
Mobile: 9441176980

COURSE:
PROBLEM SOLVING USING PYTHON POGRAMMING

CONTENTS:

1. **Introduction**
2. **Data Structures**
3. **Control Flow**
4. **Functions**
5. **Classes & Objects**
6. **Inheritance**
7. **Operator Overloading**
8. **Errors & Exceptions**
9. **Regular Expressions**
10. **File Handing**
11. **Python Databases**
12. **Additional Python Exercises**

INTRODUCTION TO PYTHON

1.1 ABOUT PYTHON

- Python is an *Interpreted*, *Object-Oriented*, *High-Level Programming Language* with *Dynamic Semantics*.
- Python is easy to learn and it is powerful *Programming Language*.
- It has efficient high level *Data Structures*.
- Provide efficient approach to *Object Oriented Programming*.
- Provide *Elegant Syntax*.
- It has *Interpreted Nature*.

1.2 HISTORY OF PYTHON

- Python was developed by **"GUIDO VAN ROSSUM"** in the late 80's and early 90's at "**National Research Institute for Mathematics and Computer Science"** in the Netherland.
- It has been derived from many languages such as ABC, MODUAL-3, C, C++, ALGOL-68, SMALL TALK, UNIX, and *Other Scripting Languages.*

1.3 FEATURES OF PYTHON VIMP

1. Simple

- Python is *Simple* and a *Small* language.
- Reading a program written in *Python* leads everyone like *Reading Language*.

2. Easy to Learn

- A python program is *Clearly Defined* and *Easily Readable*.
- The structure of the program is *Very Simple*.
- It uses *Few Keywords* and a *Clearly Defined Syntax*.

3. Portable

- Due to its *Open Source Nature* python has been ported to many platforms.
- All python programs can works on any platforms *without requiring any changes*.

4. Interpreted

- Python is processed at run time by the interpreter.
- No need to compile a program before executing it. We can simply run the program.
- Basically python converts the source code into an interpreted form called as **byte code**.

5. Object Oriented

- Python supports *Object Oriented* as well as *Program Oriented* style of programming.
- While *Object Oriented Technique* encapsulates *Code* and *Functionalities* with in Objects, *Procedure Oriented Technique*, as the other hand, builds the *Program* (or) *Functionalities* which are noting *Reusable Pieces of Program*.

6. Extensible

- Since python is open source software any one can add low level modules to the python interpreter.
- These modules enable programmers to *add* or *customize* their tools to work more efficiently.

7. Embeddable

- Programmers can embed python using their C, C++, ACTIVE X, COBAL, and Java programmers to give *Scripting Capabilities* for users.

8. Extensive Libraries

- Python has a huge *library* that is easily *portable* across different platforms.
- Their library functions are compatible as *UNIX*, *Windows* and allow programmers to perform a wide range of applications varying from *Text Processing*, *Maintaining Database*, to *GUI Programming*.

1.5 APPLICATIONS OF PYTHON (OR) WHO USES PYTHON

- Python is used to develop wide range of applications including *Image Processing*, *Text Processing*, *Enterprise Level Applications* using *Scientific* and *Numeric* data for Networks.

- Python is used in *Embedded Scripting Languages*, *3D Software*, *Web Development*, *GUI Based Develop Applications*, *Image Processing* of *Graphic Design Applications*, *Scripting or Computational Applications*, *Visualizations*, and *Parallel Computing*.

- Python is also used in developing *Games*, *Enterprise and Business Applications*, *Operating System*, *Language Development*, *Network Programming of Teaching*.

- Python is also being applied in revenue generations of products by companies. for instance , among the general known python uses base:

1. Google makes extensive use of python its *Web Based Systems.*
2. The popular ***Youtube*** *Video Sharing Services* is largely written in *Python*.
3. The *Drop Box Storage Service (Dropbox* is a free *service* that lets you bring your photos, docs, and videos anywhere and share them easily) codes and boots its server and *Desktop Client Software* is primarily written in python.
4. *Voice* and *Text Chat* for gamers uses python.
5. *Bit Torrent* peers to peer file sharing system began its life as python program.
6. Python is used in the production of *Animated Movies*.

7. *GIS Mapping Products* use python.
8. *Google Opening Web Development Framework* use python as *Application Language*.
9. *Cisco Email Server Products* uses more than one, million lines of code of Python to do its job.
10. MAYA a powerful interpreted 3D Modelling and Animation System provides a python scripting API.
11. The MSA Symmetric Key use python for *Cryptography* and *Intelligence Analysis*.
12. *Robotics* uses python to develop *Commercial* and *Military Robotic Devices*.
13. The civilizations games customizable scripted level are written in python.
14. The one laptop per child (olpc) project builds its user interface and activities involved in python.
15. Netflix and yelp have both documented the roleof pytshon in their software infrastructure.
16. Intel, Cisco,Hewlett-Packard, Seagate, Qualcomm core and IBM use python for hardware testing.
17. JP Morgan chase, UBS, get go and Citadel apply python to financial market for costing.
18. NASA, Los Alamos, Fermi lab, JPL and other use python for scientific programming.
19. In reality trending Instagram has more than 100 million people, interestingly. Its programming is written in python.
20. Pinterest and quora e viral network giants with large user base uses python for their primary backend system.

1.6 WHAT CAN I DO WITH PYTHON?

In addition to being a well-designed programming languages, python is useful for accomplishing real world basis

1.6.1 System Programming:

- Python's builts makes idle for writing, portable, maintainable *System Administration Tools* and *Utilities.*
- *Python Programmer* can send *Files* and *Directory Trees to*, other Programmers.
- Do Parallel Processing with *Process* and *Threads*, etc.

1.6.2 GUI'S:

Python provide TIC GUI API called 'TKINTER' that allows Python Program to implement portable GUI'S with machine look and feel.

1.6.3 Internet Scripting:

- Python comes with **Standard Internet Modules** that allows Python Program to perform a wide variety of Networking Tasks, in Client / Server Modes.

1.6.4 Component Integration:

- Python Scripts can easily communicate with other application using variety of Integration Mechanisms.
- For instance, integrating a library into python enables python to test and leaves the library's components and embedding python in a products enables onsite customization to be coded with having to recompile to entire product

1.6.5 Database Programming:

- Python interfaces to all commands used in Relational Database System such as Oracle, DDBC, MYSQL, and SQL Lite.

1.6.6 Rapid Prototyping:

- Rapid Prototyping with python allows a developer to incorporate several ideas into prototypes within budget.

1.6.7. Numeric and Scripting:

- Numerical Python adding a fast company Multidimensional Array Facilities to Python.
- NUMPY in the successor to both Numeric and Num Array.
- Additional numeric tools for *Python Animation*, *3D Visualization*.

1.7 VARIABLES IMP

- Variable: Variable is reserved memory location that stores value.
- In programming, a variable is a value that can change, depending on conditions or on information passed to the program.
- Variables are examples of identifiers.
- Identifiers are names given to identify something.

For naming any identifier there are some basic rules that you must follow. These rules are

1. The first character of an identifier must be an underscore ('_') or letter (upper or lowercase).
2. The rest of the identifier name can be underscore('_'), letter (upper or lowercase), or digits (0-9).
3. Identifiers names are case-sensitive. For example, `myvar` and `myVar` are not the same.
4. Punctuation characters such as @,$, and % are not allowed within identifiers.

Examples of valid identifier names are `sum, _my_var, num1, r, var_20, First. etc.`

Examples of invalid identifier names are `1num, my-var, %check, Basic Sal, H#R&A, etc.`

```
Example: Program to display data of different types using variables and literal constants.

num=7
amt=123.45
code = 'A'
pi=3.1415
population_of_India = 10000000000
msg = "Hi"

print("NUM ="+str(num))
print("\n AMT = "+str(amt))
print(" CODE = "+str(code))
print("POPULATION OF INDIA = " + str(population_of_India))
print("\n MESSAGE = "+str(msg))


OUTPUT

NUM =7
AMT = 123.45
CODE = A
POPULATION OF INDIA = 10000000000
MESSAGE = Hi
```

1.8 TYPED LANGAUGE

- The typing discipline of a language is defined by the nature of type constraint and the way they are evaluated and enforced.
- The Concept Of Typing Discipline Is Dealt With Four Types That Are As Follows.
  1. Statically Typed Languages
  2. Dynamically Typed Languages
  3. Strongly Typed Language
  4. Weakly Typed Language

1.Statically Typed Languages

- A language in which types are fixed at compile time.

  E.g. C and Java

2. Dynamically Typed Languages

- A language in which the types are discovered at execution time.

  E.g. Python, VB Script

Strongly Typed Language:

- A language in which the types are always enforced.
- For instance, Java and Python are strongly typed. If you have integer, you can not treat like a string without explicitly converting it.

Weakly Typed Language:

- A language in which the type may be ignored which is actually the opposite of strongly typed. VB Script is weakly typed. In VB Script, you can concatenate the string '12' and the integer 3 to get the string '123', then treat that as the integer 123, all without any explicit conversion.

- Python is both **dynamically typed** (because it does not use explicit data type declarations) and **strong typed** (because once a variable has a data type, it actually matters).

1.9 DATA TYPES IN PYTHON VIMP

Python has the following data types built-in by default, in these categories:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

- Print the data type of the variable x:

```
x = 5
print(type(x))
```

- Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	Data Type	Description
x = "Hello World"	str	
x = 20	int	
x = 20.5	float	
x = 1j	complex	
x = ["apple", "banana", "cherry"]	list	
x = ("apple", "banana", "cherry")	tuple	
x = range(6)	range	The rage specifies values between 0 to 6

x = {"name" : "John", "age" : 36}	dict	Create a dictionary containing personal information
x = {"apple", "banana", "cherry"}	set	
x = frozenset({"apple", "banana", "cherry"})	frozenset	The frozenset() function returns an unchangeable frozenset object (which is like a set object, only unchangeable). E.g. mylist = ['apple', 'banana', 'cherry'] x = frozenset(mylist) x[1] = "strawberry" print(x)
x = True	bool	

x = b"Hello"	bytes	Bytes objects are immutable sequences of single bytes.
		E.g.
		#triple single or double quotes allows multiple lines
		x = b'''Python Tutorial,
		Javascript Tutorial,
		MySQL Tutorial'''
		print(x)

x = bytearray(5)	bytearray	It gives a mutable sequence of integers in the range 0 <= x < 256.
		E.g.
		#create a bytearray from a list of integers in the range 0 through 255
		x = bytearray([94, 91, 101, 125, 111, 35, 120, 101, 115, 101, 200])
		print(x)

x = memory view(bytes(5))	memoryview	The python memoryview() function returns a memoryview object of the given argument.

1.10 LITERALS

Literal: In Python, literals refer to the data that is specified in a variable or constant.

1.10.1 Numbers

Can be integers (1 and 2), floats (1.1 and 1.2), fractions (1/2 and 2/3) or even complex numbers (3+7i)

1.10.1 Strings

- String: A string is group of characters.

- Strings in python are surrounded by either single quotation marks, or double quotation marks.
- 'hello' is the same as "hello".
- You can display a string literal with the print() function:

E.g1.

```
print("Hello")
print('Hello')
```

E.g.

```
a = ''' We are the students of P.B.Siddhartha College of Arts & Scince,
MCA Fourth Semester,
Leaning Python Programming.'''
print(a)
```

1.11 OPERATORS

Operators are symbols that represent that represent computations like addition, subtraction, multiplication and so on.

Operator categories:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment and Decrement Operators
- Bitwise Operators
- Special Operators

CHAPTER 2
DATA STRUCTURES

2.1 LISTS

- Lists are used to store multiple items in a single variable.
- Lists are created using square brackets.
- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.
- When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
- If you add new items to a list, the new items will be placed at the end of the list.
- Since lists are indexed, lists can have items with the same value.

```
Example 1:
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

```
Output:
['apple', 'banana', 'cherry', 'apple', 'cherry']
```

```
Example 2:
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

```
Output: 3
```

```
Example 3:
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

```
Output:
['apple', 'banana', 'cherry']
[1, 5, 7, 9, 3]
[True, False, False]
```

```
Example 4:
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

```
Output:
<class 'list'>
```

```
Example 5:
thislist = list(("apple", "banana", "cherry")) # note the double
round-brackets
print(thislist)
```

```
Output:
['apple', 'banana', 'cherry']
```

2.1.1 Access List Items

- List items are indexed and you can access them by referring to the index number.

```
Example1:
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
Output:
banana
```

2.1.2 Change Item Value

- To change the value of a specific item, refer to the index number.

```
Example1:
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

```
Output:
['apple', 'blackcurrant', 'cherry']
```

2.1.3 Appended Items

- To add an item to the end of the list, use the append() method.

```
Example1:
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

```
Output:
['apple', 'banana', 'cherry', 'orange']
```

2.1.4 Remove Specified Item

- The remove() method removes the specified item

```
Example1:
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

```
Output:
['apple', 'cherry']
```

2.1.5 Loop Through a List

- You can loop through the list items by using a for loop.

```
Example 1:
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

```
Output:
apple
banana
cherry
```

2.1.6 List Comprehension

- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

- Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.
- Without list comprehension you will have to write a `for` statement with a conditional test inside.

```
Example 1:
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)
print(newlist)
```

```
Output:
```

```
['apple', 'banana', 'mango']
```

2.1.7 Sort List Alphanumerically

- List objects have a `sort()` method that will sort the list alphanumerically, ascending, by default.

```
Example 1:
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

```
Output:
['banana', 'kiwi', 'mango', 'orange', 'pineapple']
```

```
Example2:
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

```
Output:
[23, 50, 65, 82, 100]
```

2.1.8 Copy a List

- You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a reference to `list1`, and changes made in `list1` will automatically also be made in `list2`.
- There are ways to make a copy, one way is to use the built-in List method `copy()`.

```
Example:
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

```
Output:
['apple', 'banana', 'cherry']
```

2.1.9 Join Two Lists

- There are several ways to join, or concatenate, two or more lists in Python.

- One of the easiest ways are by using the + operator.

```
Example 1:
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)

Output:
'a', 'b', 'c', 1, 2, 3]

Example 2:
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
for x in list2:
  list1.append(x)
print(list1)

Output:
['a', 'b', 'c', 1, 2, 3]


Example 1:
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

2.1.9 List Methods

Method	Description
append( )	Adds an element at the end of the list
clear( )	Removes all the elements from the list
copy( )	Returns a copy of the list
count( )	Returns the number of elements with the specified value
extend( )	Add the elements of a list (or any iterable), to the end of the current list
index( )	Returns the index of the first element with the specified value
insert( )	Adds an element at the specified position
pop( )	Removes the element at the specified position

remove( )	Removes the item with the specified value
reverse( )	Reverses the order of the list
sort( )	Sorts the list

```
Output:
 ('apple', 'banana', 'cherry')
```

2.2 TUPLE

- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is ordered and **unchangeable**.
- Tuples are written with round brackets.
- Tuple items are ordered, unchangeable, and allow duplicate values.
- Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

```
Example 1:
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

```
Output:
('apple', 'banana', 'cherry')
```

2.1.1 Access Tuple Items

- You can access tuple items by referring to the index number, inside square brackets.

```
Example 1:
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
Output:
banana
```

Negative Indexing:

- Negative indexing means start from the end.
- -1 refers to the last item, -2 refers to the second last item etc.

```
Example 1:
thistuple = ("apple", "banana", "cherry")
```

```
print(thistuple[-1])
```

```
Output:
cherry
```

Range of Indexes:

- You can specify a range of indexes by specifying where to start and where to end the range.
- When specifying a range, the return value will be a new tuple with the specified items.

```
Example 1:
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])

#This will return the items from position 2 to 5.

#Remember that the first item is position 0,
#and note that the item in position 5 is NOT included
```

```
Output:
('cherry', 'orange', 'kiwi')
```

```
Example 2:
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

```
Output:
('apple', 'banana', 'cherry', 'orange')
```

```
Example 3:
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

```
Output:
('cherry', 'orange', 'kiwi', 'melon', 'mango')
```

Range of Negative Indexes:

- Specify negative indexes if you want to start the search from the end of the tuple.

```
Example 4:
```

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])

Output:
('orange', 'kiwi', 'melon')

#Negative indexing means starting from the end of the tuple.

#This example returns the items from index -4 (included) to index -1
(excluded)

#Remember that the last item has the index -1
```

Check if Item Exists:

- Check if "apple" is present in the tuple.

```
Example 5:
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")

Output:
Yes, 'apple' is in the fruits tuple
```

2.1.2 Update Tuples

- Tuples are unchangeable, meaing that you cannot change, add, or remove items once the tuple is created.
- But there are some workarounds.

```
Example 1:
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)

Output:
("apple", "kiwi", "cherry")
```

Add Items:
- Once a tuple is created, you cannot add items to it.

```
Example 1:
thistuple = ("apple", "banana", "cherry")
thistuple.append("orange") # This will raise an error

print(thistuple)
```

```
Output:
Traceback (most recent call last):
  File "./prog.py", line 2, in <module>
AttributeError: 'tuple' object has no attribute 'append
```

Example 2:

- Convert the tuple into a list, add "orange", and convert it back into a tuple:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)

print(thistuple)
```

```
Output:
('apple', 'banana', 'cherry', 'orange')
```

Remove Items:

- Note: You cannot remove items in a tuple.

Example 3:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)

print(thistuple)
```

```
Output:
('banana', 'cherry')
```

Unpacking a Tuple:

- When we create a tuple, we normally assign values to it. This is called "packing" a tuple.

Example 4: (Packing tuple)

```
fruits = ("apple", "banana", "cherry")

print(fruits)
```

```
Output:
 ('apple', 'banana', 'cherry')
```

- But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking".

Example 5:

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

```
Output:
apple
banana
cherry
```

Using Asterisk*

- If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list.

Example 6:

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

```
Output:
apple
banana
['cherry', 'strawberry', 'raspberry']
```

- If the asterix is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

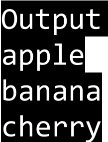
```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

print(green)
print(tropic)
print(red)
```

```
Output:
apple
['mango', 'papaya', 'pineapple']
cherry
```

2.1.3 Loop Through a Tuple

- You can loop through the tuple items by using a `for` loop.

```
Example 1:
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```



```
Output:
apple
banana
cherry
```

Loop Through the Index Numbers:

- You can also loop through the tuple items by referring to their index number.
- Use the range() and len() functions to create a suitable iterable.

```
Example 2:

thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```

```
Output:
apple
banana
```

```
cherry
```

Using a While Loop:

- You can loop through the list items by using a while loop.
- Use the len() function to determine the length of the tuple, then start at 0 and loop your way through the tuple items by refering to their indexes.
- Remember to increase the index by 1 after each iteration.

Example 3:

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

```
Output:
apple
banana
cherry
```

2.1.4 Python - Join Tuples

Join Two Tuples

- To join two or more tuples you can use the + operator.

Example 4:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
Output:
('a', 'b', 'c', 1, 2, 3)
```

Multiply Tuples

- If you want to multiply the content of a tuple a given number of times, you can use the * operator.

```
Example 5:

fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)

Output:
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

2.1.5 Tuple Methods

- Python has two built-in methods that you can use on tuples.

Method	Description
count( )	Returns the number of times a specified value occurs in a tuple
index( )	Searches the tuple for a specified value and returns the position of where it was found

2.3 SETS

- A set is a collection which is both *unordered* and *unindexed*.
- Unordered means that the items in a set do not have a defined order.
- Set items can appear in a different order every time you use them, and cannot be referred to by index or key.
- Sets are used to store multiple items in a single variable.
- Sets are written with curly brackets.

```
Example1:
thisset = {"apple", "banana", "cherry"}
print(thisset)

# Note: the set list is unordered, meaning: the items will appear in a random
order.

# Refresh this page to see the change in the result.

Output:
{'cherry', 'apple', 'banana'}
```

- Sets cannot have two items with the same value. (Duplicates Not Allowed)

```
Example1:
thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)
```

```
Output:
{'banana', 'cherry', 'apple'}
```

2.3.1 Access Set Items

- You cannot access items in a set by referring to an index or a key.
- But you can loop through the set items using a `for` loop, or ask if a specified value is present in a set, by using the `in` keyword.

```
Example1:
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

```
Output:
banana
apple
cherry
```

Change Items: Once a set is created, you cannot change its items, but you can add new items.

2.3.2 Add Set Items

- Once a set is created, you cannot change its items, but you can add new items.

```
Example1:
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

```
Output:
{'cherry', 'orange', 'banana', 'apple'}
```

2.3.2 Remove Set Items

- To remove an item in a set, use the `remove()`, or the `discard()` method.

```
Example1:
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```

```
Output:
{'cherry', 'apple'}
```

2.3.2 Loop Sets

- You can loop through the set items by using a `for` loop.

```
Example1:
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

```
Output:
apple
cherry
banana
```

2.3.3 Join Two Sets

- There are several ways to join two or more sets in Python.
- You can use the `union()` method that returns a new set containing all items from both sets, or the `update()` method that inserts all the items from one set into another.

```
Example1:
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)
```

```
Output:
{'b', 1, 'a', 'c', 3, 2}
```

```
Example2:
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1)
```

Keep ONLY the Duplicates:

- The `intersection_update()` method will keep only the items that are present in both sets.

```
Example1:
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
x.intersection_update(y)
print(x)
```

```
Output:
{'apple'}
```

```
Example2:
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.intersection(y)
print(z)
```

```
Output:
{'apple'}
```

Keep All, But NOT the Duplicates:

- The `symmetric_difference_update()` method will keep only the elements that are NOT present in both sets.

```
Example1:
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
x.symmetric_difference_update(y)
print(x)
```

```
Output:
{'google', 'banana', 'microsoft', 'cherry'}
```

- The `symmetric_difference()` method will return a new set, that contains only the elements that are NOT present in both sets.

```
Example1:
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.symmetric_difference(y)
print(z)
```

```
Output:
{'google', 'banana', 'microsoft', 'cherry'}
```

2.3.4 Set Methods

Method	Description
add( )	Adds an element to the set
clear( )	Removes all the elements from the set
copy( )	Returns a copy of the set
difference( )	Returns a set containing the difference between two or more sets
difference_update( )	Removes the items in this set that are also included in another, specified set
discard( )	Remove the specified item
intersection( )	Returns a set, that is the intersection of two other sets
intersection_update ()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint( )	Returns whether two sets have a intersection or not
issubset( )	Returns whether another set contains this set or not
issuperset( )	Returns whether this set contains another set or not
pop( )	Removes an element from the set
remove( )	Removes the specified element
symmetric_difference( )	Returns a set with the symmetric differences of two sets

symmetric_difference_update ( )	inserts the symmetric differences from this set and another
union( )	Return a set containing the union of sets
update( )	Update the set with the union of this set and others

2.4 DICTIONARY

- Dictionaries are used to store data values in `key : value` pairs.
- A dictionary is a collection which is ordered*, changeable and does not allow duplicates.
- Dictionaries are written with curly brackets, and have keys and values:

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
Output:
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Ordered or Unordered?

- When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.
- Unordered means that the items does not have a defined order, you cannot refer to an item by using an index.

Changeable

- Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

Duplicates Not Allowed

- Dictionaries cannot have two items with the same key.
- Duplicate values will overwrite existing values.

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
```

```
Output:
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

2.4.1 Accessing Items

- You can access the items of a dictionary by referring to its key name, inside square brackets.
- Get the value of the "model" key:

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
```

```
Output:
Mustang
```

- There is also a method called `get()` that will give you the same result:

```
Example2:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.get("model")
print(x)
```

```
Output:
Mustang
```

Change Values:

- You can change the value of a specific item by referring to its key name
- Change the "year" to 2018

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
print(thisdict)
```

```
Output:
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

Update Dictionary:

- The `update()` method will update the dictionary with the items from the given argument.
- The argument must be a dictionary, or an iterable object with `key:value` pairs.
- Update the "year" of the car by using the `update()` method.

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})
print(thisdict)
```

```
Output:
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

2.4.2 Adding Items

- Adding an item to the dictionary is done by using a new index key and assigning a value to it.

```
Example1:
thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

```
Output:
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Update Dictionary:

- The `update()` method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.
- The argument must be a dictionary, or an iterable object with `key:value` pairs.
- Add a color item to the dictionary by using the update() method:

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"color": "red"})
print(thisdict)
```

```
Output:
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

2.4.3 Remove Dictionary Items

- There are several methods to remove items from a dictionary.
- The `pop()` method removes the item with the specified key name.

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

```
Output:
{'brand': 'Ford', 'year': 1964}
```

- The `popitem()` method removes the last inserted item (in versions before 3.7, a random item is removed instead).

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

```
Output:
{'brand': 'Ford', 'model': 'Mustang'}
```

- The `del` keyword removes the item with the specified key name.

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
print(thisdict) #this will cause an error because "thisdict" no longer exists.
```

```
Output:
Traceback (most recent call last):
  File "demo_dictionary_del3.py", line 7, in <module>
    print(thisdict) #this will cause an error because "thisdict" no longer exists.
NameError: name 'thisdict' is not defined
```

- The `clear()` method empties the dictionary.

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

```
Output:
{}
```

2.4.4 Loop Through a Dictionary:

- You can loop through a dictionary by using a for loop.
- When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.
- Print all key names in the dictionary, one by one.

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(x)
```

```
Output:
brand
model
year
```

- Print all *values* in the dictionary, one by one.

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])
```

```
Output:
Ford
Mustang
1964
```

- You can use the `keys()` method to return the keys of a dictionary:

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.keys():
  print(x)
```

```
Output:
brand
model
year
```

- Loop through both *keys* and *values*, by using the `items()` method.

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)
```

```
Output:
```

```
brand Ford
model Mustang
year 1964
```

2.4.5 Copy a Dictionary

- You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a *reference* to dict1, and changes made in dict1 will automatically also be made in dict2.
- There are ways to make a copy, one way is to use the built-in Dictionary method copy().

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

```
Output:
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

- Another way to make a copy is to use the built-in function dict().

```
Example1:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

```
Output:
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

2.4.6 Nested Dictionaries:

- A dictionary can contain dictionaries, this is called nested dictionaries.
- Create a dictionary that contain three dictionaries:

```
Example1:
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
print(myfamily)
```

```
Output:
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias',
'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}
```

- Create three dictionaries, then create one dictionary that will contain the other three dictionaries.

```
Example1:

child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}
```

```
myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
print(myfamily)
```

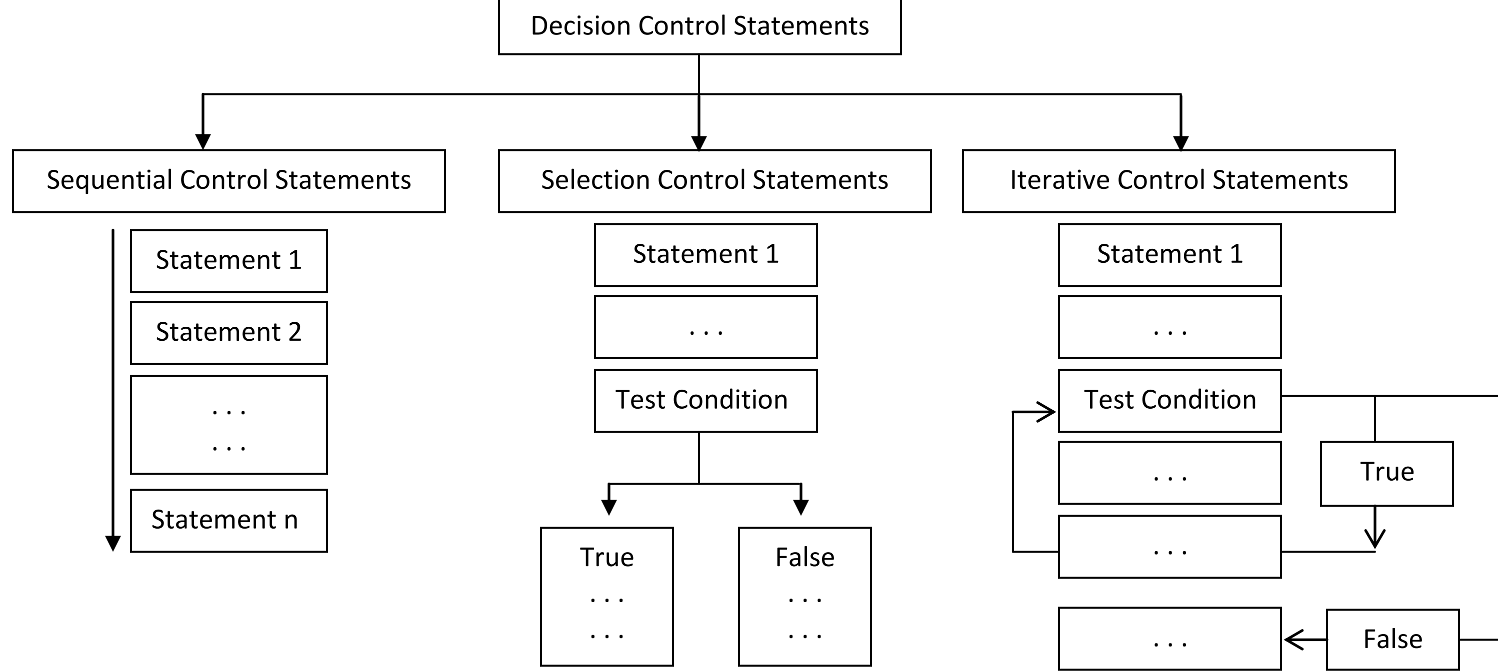
```
Output:
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias',
'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}
```

2.4.7 Dictionary Methods:

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

CHAPTER 3
CONTROL FLOW

TOPIC 1: DECISION CONTROL STATEMENTS



4.2 SELECTION / CONDITIONAL BRANCHING STATEMENTS

4.2.1 IF STATEMENT

- The if statement is the simplest form of decision control statement that is frequently used in decision making.

Figure: Flow chart for if statement

Test Expression
False
True
Statement Block 1
Statement Block x

```
Syntax:
If test_expression:
       Statement 1
         . . .
       Statement n
Statement x
```

```
Example 1:
x = 10        #Initialize the value of x
if (x>0):     #test the value of x
    x=x+1     #Increment the value of x if it is > 0
```

```
print(x)     #Print the value of x
```

```
Example 2:
# Write a program to determine whether a person is eligible to vote.
age = int(input("Enter the age : "))
if (age >= 18):
    print("You are eligible to vote")
```

```
Output:
Enter the age : 18
You are eligible to vote
```

```
Example 3:
# Write a program to determine the character entered by the user
char = input("Press any key : ")
if(char.isalpha()):
    print("The user has entered a character")
if(char.isdigit()):
    print("The user has enterd a digit")
if(char.isspace()):
    print("The user entered a white space character")
```

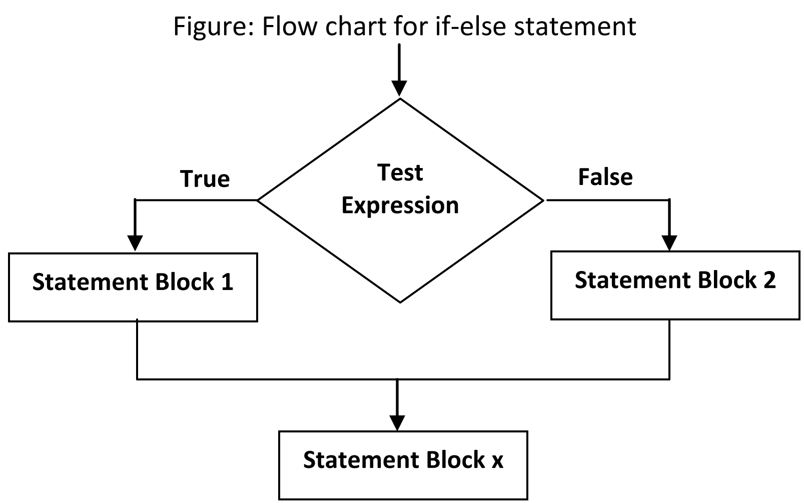
```
Output:
Press any key : a
The user has entered a character
```

4.2.2 if - else Statement

- The if..else statement evaluates test expression and will execute the body of if only when the test condition is True . If the condition is False , the body of else is executed.

```
Syntax:
If test_expression:
       Statement block 1
else:
       Statement block 2
Statement x
```

Figure: Flow chart for if-else statement



Example 1:

```
# Write a program to find whether the given number is even or odd
num = int(input("Enter any number :"))
if(num%2==0):
    print(num,"is even")
else:
    print(num,"is odd")
```

```
Output:
Enter any number :125
125 is odd
```

Example 2:

```
# Write a program to enter any character. If the entered character is in
lowercase then convert it into uppercase and if it is
# an uppercase character, then convert it into lowercase.
ch = input ("Enter any character : ")
if (ch >= 'A' and ch <= 'Z'):
    ch = ch.lower()
    print("The entered character was in uppercase. In lowercase it is :" +
ch)
else:
    ch = ch.upper()
    print("The entered character was in lowercase. In uppercase it is :" +
ch)
```

```
Output:
Enter any character : c
The entered character was in lowercase. In uppercase it is :C
```

```
Example 3:
# A company decides to give bonus to all its emplyess on Diwali. A 5% bonus
on salary
# is given to the male workers and 10% bonus on salary to the female workers.
Write a
# program to enter the salary of the employee and sex of the employee. If the
salary of
# the employee is less than Rs.10,000 then the employee get an extra 2%
bonuson salary.
# Calcullate the bonus that has to given to the employee and display the
salary that the
# employee will get.

ch=input("Enter the sex of the employee (m or f) :")
sal=int(input("Enter the salary of the employee :"))
if (ch=='m'):
    bonus = 0.05 * sal
else:
    bonus = 0.10 * sal
amt_to_be_paid = sal + bonus
print("Salary =", sal)
print("Bonus =", bonus)
print("****************************")
print("Amount to be paid : ",amt_to_be_paid)
```

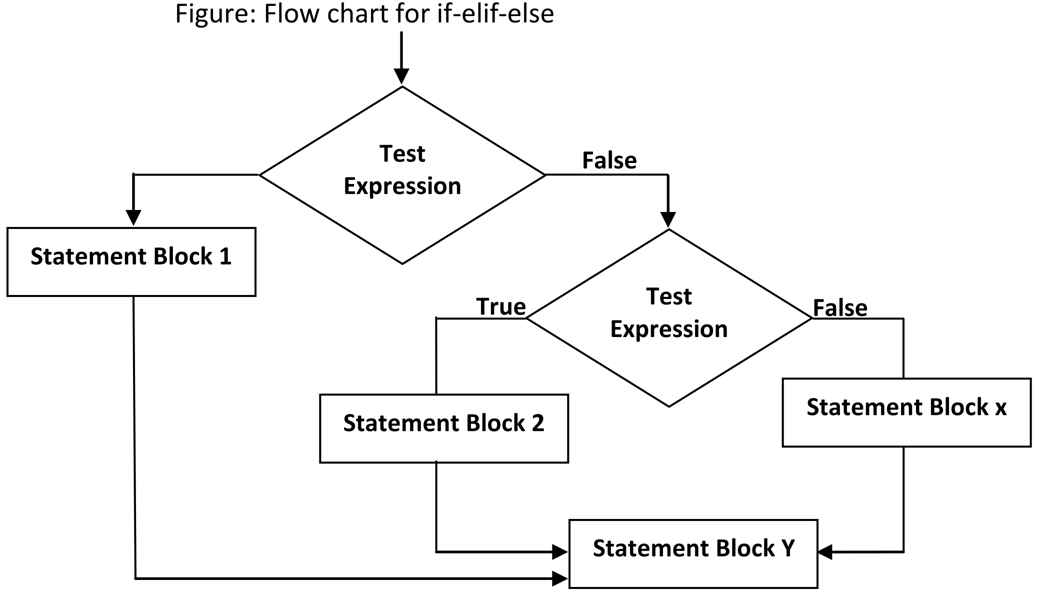
```
Output:
Enter the sex of the employee (m or f) :m
Enter the salary of the employee :30000
Salary = 30000
Bonus = 1500.0
****************************
Amount to be paid :  31500.0
```

4.2.3 if…elif…else in Python

- `if…elif…else` are conditional statements that provide you with the decision making that is required when you want to execute code based on a particular condition.

```
Syntax:
If (test_expression 1)
       Statement_block 1
Elif (test_expression 2)
       Statement_block 2
…
Elif (test_expression N)
       Statement_block N
Else
      Statement_block X
Statement_Y
```

Figure: Flow chart for if-elif-else




```
Example1:
# Write a program to test whether a number entered by the user is negative,
positive or equal to zero
num = int(input("Enter any number : "))
if (num==0):
    print("The value is equal to zero")
if (num>0):
    print("The number is positive")
else:
    print("The number is negative")
```

```
Output:
Enter any number : -10
The number is negative
```

```
Example2:
# Write a program to determine whether the character entered is vowel or not.
ch = input("Enter any character :")
if(ch=="A" or ch=="E" or ch=="I" or ch=="O" or ch=="U"):
    print (ch, "is vowel")
elif (ch=="a" or ch=="e" or ch=="i" or ch=="o" or ch=="u"):
    print(ch, "is vowel")
else:
    print(ch, "is not a vowel")
```

```
Output:
Enter any character :A
A is vowel
```

Example 3: Write a program to find the greatest number from three numbers.

```
# Write a program to find the greatest number from three numbers.
num1 = int(input("Enter the first number : "))
num2 = int(input("Enter the second number : "))
num3 = int(input("Enter the third number : "))
if(num1>num2):
    if(numl>num3):
        prit(num1, "is greater than", num2, "and", num3)
    else:
        print(num3, "is greater than", num1, "and", num2)
elif(num2>num3):
    print(num2, "is greater than", num1, "and", num3)
else:
    print("The three numbers are equal")
```

```
Output:
Enter the first number : 13
Enter the second number : 43
Enter the third number : 25
43 is greater than 13 and 25
```

Example 4: Write a program to calculate tax given the following conditions:
If income is less than 1,50, 000 then no tax
If taxable income is Rs.1,50,001 - Rs.300,000 then charge 10% tax
If taxable income is Rs.3,00,001 - Rs.500,000 then charge 20% tax
If taxable income is above Rs.5,00,001 then charge 30% tax

```
#Write a program to calculate tax given the following conditions:
#If income is less than 1,50, 000 then no tax
#If taxable income is Rs.1,50,001 - Rs.300,000 then charge 10% tax
#If taxable income is Rs.3,00,001 - Rs.500,000 then charge 20% tax
#If taxable income is above Rs.5,00,001 then charge 30% tax
MIN1 = 150001
MAX1 = 300000
RATE1 = 0.10
MIN2 = 300001
MAX2 = 500000
RATE2 = 0.20
MIN3 = 500001
RATE3 = 0.30
income = int(input("Enter the income : "))
taxable_income = income - 150000
if(taxable_income<= 0):
    print("No tax")
elif(taxable_income>=MIN1 and taxable_income<MAX1):
    tax = (taxable_income - MIN1) * RATE1
```

```
elif(taxable_income>=MIN2 and taxable_income<MAX2):
    tax = (taxable_income - MIN2) * RATE2
else:
    tax = (taxable_income - MIN3) * RATE3
print("TAX = ", tax)
```

```
Output:
Enter the income : 2000000
TAX =  404999.7
```

4.3 BASIC LOOP STRUCTURES / ITERATIVE STATEMENTS

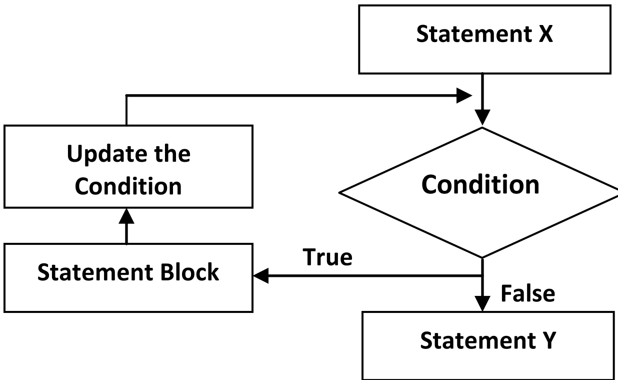
- Python supports basic loop structures through iterative statements.
- Iterative statements are used to repeat the execution of list of statements depending on the value of integer expression.

4.3.1 While Loop

- While loop provides a mechanism to repeat one or more statements while a particular condition is True.

```
Syntax:

Statement x
While (condition):
      Statement block
Statement y
```

Figure: Flow chart for while loop



Example 1: Write a program to print first 5 numbers using while loop.

```
i = 1
while i < 6:
  print(i)
  i = i + 1
```

```
Output:
1
2
3
4
5
```

Example 2: Write a program to calculate the sum of numbers from m to n

```
# Write a program to calculate the sum of numbers from m to n
m=int(input("Enter the value of m : "))
n=int(input("Enter the value of n : "))
s=0
while (m<=n):
    s=s+m
    m=m+1
print("sum = ", s)
```

```
Output:
Enter the value of m : 1
Enter the value of n : 10
sum =  55
```

Example 3: Write a program to read the numbers until -1 is encountered. Also count the negative, poitive, and zeros entered by the user.

```
# Write a program to read the numbers until -1 is encountered. Also count the
# negative, poitive, and zeros entered by the user.

negatives=0
positives=0
zeroes=0
print("Enter -1 to exit...")
while(1):
    num=int(input("Enter any number : "))
    if(num==-1):
        break
    if(num==0):
        zeroes=zeroes+1
    elif(num>0):
        positives = positives+1
```

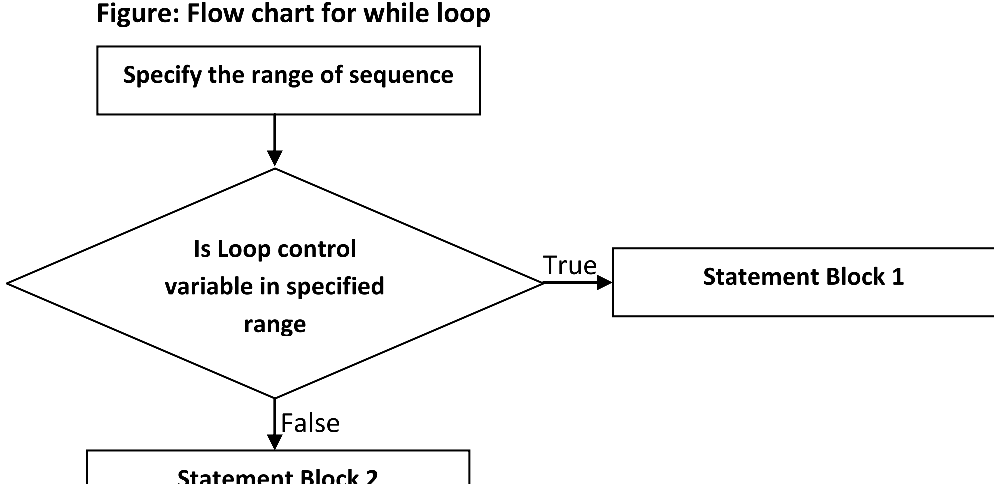
```
    else:
        negatives = negatives+1
print("Count of positive numbers entered : ", positives)
print("Count of negative numbers entered : ", negatives)
print("Count of zeroes entered : ", zeroes)
```

4.3.1 For Loop

- For loop provides a mechanism to repeat task until a particular condition is True.

```
Syntax:
for loop_control_var in sequence:
      Statement block
```

Figure: Flow chart for while loop



Example 1: Write program to print first 5 numbers using the range() in a for loop

```
# Write program to print first 5 numbers using the range() in a for loop
for i in range(1, 5):
    print(i)
```

```
Output:
1
2
3
4
```

Example 2: Write program to print first 10 numbers using the range() in a for loop using step value

```
# Write program to print first 5 numbers using the range() in a for loop
for i in range(1, 10, 2):
    print(i)
```

```
Output:
1
3
5
7
9
```

Example 3: Write a program using for loop to calculate the average of first n natural numbers

```
# Write a program using for loop to calculate the average of first n natural
numbers
n=int(input("Enter the value of n :"))
avg = 0.0
s = 0
for i in range(1,n+1):
    s=s+i
avg = s/i
print("The sum of first ",n,"numbers is",s)
print("The average of first",n,"natural numbers is",avg)
```

```
Output:
Enter the value of n :5
The sum of first  5 numbers is 15
The average of first 5 natural numbers is 3.0
```

4.4 NESTED LOOPS

- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop".

Example 1: Write a program to Print each adjective for every fruit.

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

```
Output:
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

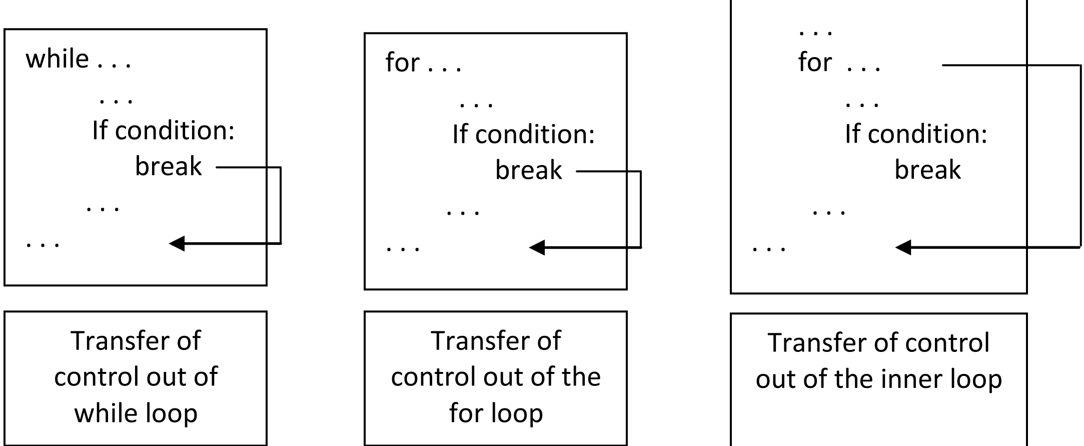
```
Example 2: Write a program to print the following pattern
Pass 1- 1 2 3 4 5
Pass 2- 1 2 3 4 5
Pass 3- 1 2 3 4 5
Pass 4- 1 2 3 4 5
Pass 5- 1 2 3 4 5

for i in range(1,6):
    print("Pass",i,"-",end=' ') # appends space instead of newline
    for j in range(1,6):
        print(j, end=' ') # appends space instead of newline
    print()
```

```
Output:
Pass 1 - 1 2 3 4 5
Pass 2 - 1 2 3 4 5
Pass 3 - 1 2 3 4 5
Pass 4 - 1 2 3 4 5
Pass 5 - 1 2 3 4 5
```

4.5 THE BREAK STATEMENT

- The break statement is used to terminate the execution of the nearest enclosing loop in which it appears.

Syantax:



Example1: Program to demonstrate the break statement.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

Output:
apple

Example 2: Write a program to print numbers 1 to 5 from the sequnce of ten numbers

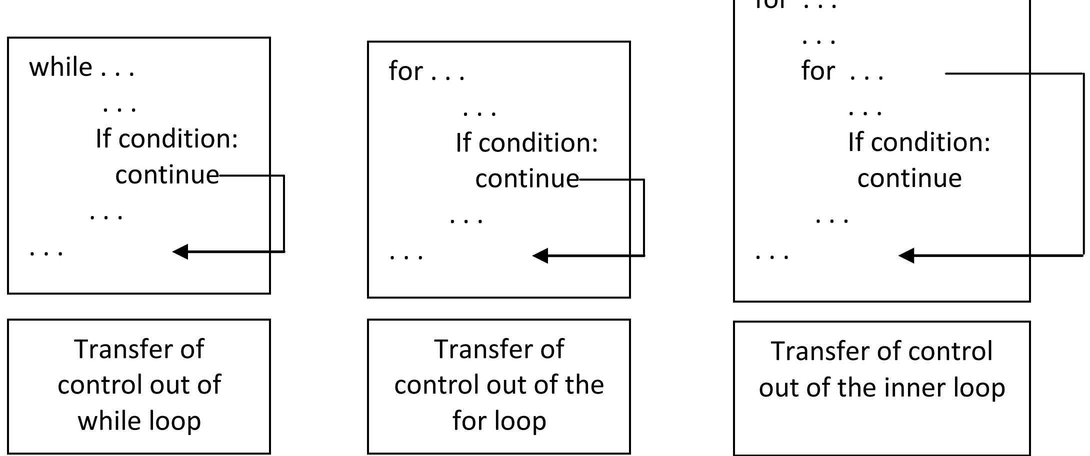
```
i = 1
while i <= 10:
    print(i, end = " ") # appends space instead of newline
    if i==5:
        break
    i = i + 1
print("\n Done")
```

Output:
1 2 3 4 5
Done

4.6 THE CONTINUE STATEMENT

- The continue statement is used to stop the current iteration of the loop and continue with the next one.

  Syntax:



Example 1: Write a program to skip the iteration if the variable i is 3, but continue with the next iteration:

```
for i in range(9):
  if i == 3:
    continue
  print(i)
```

Output:

```
Example 2: Write a program to print sequence of numbers 1 to 10 by excluding
number 5.

for i in range(1,11):
    if(i==5):
        continue
    print(i, end=" ") # appends space instead of newline
print("\n Done")
```

```
Output:
1 2 3 4 6 7 8 9 10
Done
```

4.7 THE PASS STATEMENT

- The pass statement acts as a placeholder and usually used when there is no need of code but a statement is still required to make a code syntactically correct.

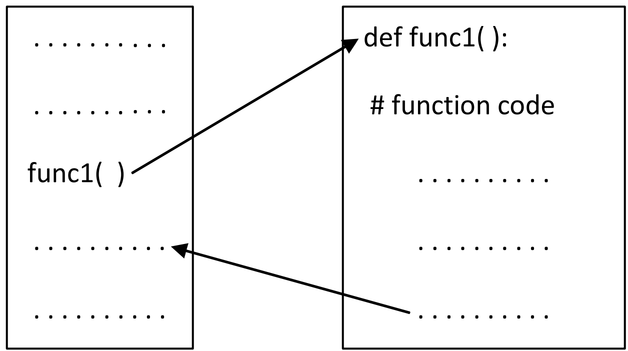
```
# Write a program to check if the number is even we are doing nothing and if
# it is odd then we are displaying the number.

for num in [20, 11, 9, 66, 4, 89, 44]:
    if num%2 == 0:
        pass
    else:
        print(num)
```

```
Output:
11
9
89
```

CHAPTER 4
FUNCTIONS

TOPIC 1: INTRODUCTION

- Function: A function is a block of organized and reusable program code that performs a single, specific, and well defined task.




```
Syntax:

def function_name(variable1, variable2,…)
      document string
      statement    block
      return [expression]
```

Example 1: Write a function to display message “Hello from a function”

```
def my_function():
  print("Hello from a function")
my_function()
```

```
Output:
Hello from a function
```

Example 2: Write a program that subtract two numbers using a function

```
# Write a program that subtract two numbers using a function
def diff(x,y):        # function to subtract two numbers
    return x - y
a = 20
b = 10
operation = diff      # function name assigned to a variable
print(operation(a,b)) # function called using variable name
```

```
Output:
10
```

Example 3: Write a function that displays a string repeatedly

```
# Write a function that displays a string repeatedly
def func():
    for i in range(4):
        print("Hello World")
func()   #function call
```

```
Output:
Hello World
Hello World
Hello World
Hello World
```

TOPIC 2: FUNCTION CALL

- Defining a function means specifying its name, parameters that are expected, and the set of instructions.

```
Syntax:
function_name(variable1, variable2, …)
```

2.1 Function Parameters

- Information can be passed into functions as arguments.
- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

Example 1: Program to add two integers using functions

```
# Program to add two integers using functions
def total(a,b):
    result = a+b
    print("Sum of ",a, "and" ,b, "=", result)
a=int(input("Enter the first number : "))
b=int(input("Enter the second number : "))
total(a,b) #function call with two arguments
```

```
Output:
Enter the first number : 10
Enter the second number : 20
Sum of  10 and 20 = 30
```

TOPIC 3: VARIABLE SCOPE AND LIFE TIME

Scope of the variable: Part of the program in which a variable is accessible is called its scope.
Lifetime of the variable: Duration for which the variable exits is called its life time.

LOCAL AND GLOBAL VARIABLE

- **Global Variable:** Global variable are those variables which are defined in the main body of the program file.
- They are visible throughout the program file.

- **Local Variable:** A variable which is defined within a function is local to that function.
- A local variable can be accessed from the point of its definition until the end of the function in which it is defined. It exists as long as the function is executing
- Function parameters behave like local variable in the function.

Example 1: Write a program to understand the difference between local and global variables.

```
num1=10 # global variable
print("Global variable num1 = ", num1)
def function1(num2): # num2 is function parameter
    print("In Function - Local Varible num2 =",num2)
    num3= 30 # num3 is a local variable
    print ("In Function - Local Variable num3 =",num3)
function1(20) # 20 is passed as an argument to the function
print("num1 again =", num1) #global variable is being accessed
# Error - local varible can't be used outside the function in which it is
defined
print ("num3 outide fuction =", num3)
```

```
Output:
Global variable num1 =  10
In Function - Local Varible num2 = 20
In Function - Local Variable num3 = 30
num1 again = 10
```

Example 2: Write to add two integers using functions with different arguments.

```
def total(x,y):
    result = x+y
    print("Sum of ",x, "and" ,y, "=", result)
a=int(input("Enter the first number : "))
b=int(input("Enter the second number : "))
total(a,b) #function call with two arguments
```

```
Output:
Enter the first number : 10
Enter the second number : 12
Sum of 10 and 12 = 22
```

USING THE GLOBAL STATEMENT

- To define a variable defined inside a function as a global, you must use the global statement.
- This declares the local or the inner variable of the function to have module scope.

```
# Example 1: Write a program to demonstrate the use of global statement

var = "Good"
def show():
    global var1
    var1="Morning"
    print("In function var is - ", var)
show()
print("Outside  function,  var1  is  -  ",var1) #  accessible  as  it  is  global
variable
print("var is - ", var)
```

```
Output:
In function var is -  Good
Outside function, var1 is - Morning
var is -  Good
```

```
# Example 2: Program to demonstrate modifying a global variable

var = "Good"
def show():
    global  var
    var = "Moring"
    print("In Function var is - ", var)
show()
print("Outside Function var is - ",var)
var = "Fantastic"
print("Outside Function, after modification,var is - ", var)
```

```
Output:
In Function var is - Moring
Outside Function var is -  Moring
Outside Function, after modification,var is -  Fantastic
```

THE RETURN STATEMENT

The return statement is used for two things.

- Return a value to the caller.
- To end and exit a function and go back to its caller.

Example1: Program demonstrate return statement.

```
def display(str):
    print(str)
x = display("Hello World") # assignig return value to another variable
print(x)
print(display("Hello  Again"))  #  print  the  value  without  assigning  it  to
another variable
# It should be noted that in the output None is returned from the function
```

```
Output:
Hello World
None
Hello Again
None
```

Example 2: Program to demonstrate function which returns an integer to the caller.

```
def cube(x):
    return(x*x*x)
num=10
result = cube(num)
print("Cube of",num, "=", result)
```

```
Output:
Cube of 10 = 1000
```

MORE ON DEFINING FUNCTIONS

- A function can be defined with

1. Required Arguments
2. Keyword Arguments
3. Default Arguments
4. Varible-length Arguments

Required Arguments

- In the required argument, the arguments are passed to a function in correct positional order. Also, the number of arguments in the function call should exactly match with the number of arguments specified in the function definition.

```
Example1: Program to demonstrate function with required argument
def display(str):
    print(str)
str="Hello"
display(str)
```

```
Output:
Hello
```

Keyword Arguments

- When we call a function with some value, the values are assigned to the arguments based on their position.
- Python also allows functions to be called using keyword arguments in which the order (or position) of the arguments can be changed.
- The values are not assigned to arguments according to their position but based on their name.

```
Example 1: Write a program to demonstrate the use of keyword arguments.

def display(str,int_x,float_y):
    print("The string is :", str)
    print("The integer value is :",int_x)
    print("The floating position value is : ",float_y)
display(float_y=56789.045, str="Hello",int_x=1234)
```

```
Output:
The string is : Hello
The integer value is : 1234
The floating position value is :  56789.045
```

Default Arguments

- Python allows users to specify function arguments that can have default values.
- The default value to an argument is provided by using the assignment operator (=).
- Users can specify a default value for one or more arguments.

```
Example 1: Write a program to demonstrate the use of default arguments.

def display(name, course = "M.C.A"):
    print("Name : " + name)
    print("Course : " + course)
display(course = "M.Sc", name = "Charishma") # Keyword Arguments
display(name = "Abhiram") # Default Argument
```

```
Output:
Name : Charishma
Course : M.Sc
Name : Abhiram
Course : M.C.A
```

Varible-length Arguments

- In some situations, it is not known in advance how many arguments will be passed to a function. In such cases, Python allows programmers to make function calls with arbitrary number of arguments .
- When we use arbitrary arguments or variable-length arguments, then the function definition uses an asterisk (*) before parameter name.

```
Syntax:

def functionaname([arg1, arg2, . . .] *var_args_tuple):
        function statements
        return [expression]
```

```
Example 1: Program to demonstrate the use of variable-length arguments.

def func(name, *fav_subjects):
    print("\n",name, " likes to read ", end=" ")
    for subjects in fav_subjects:
        print(subjects, end=" ")
func("Bindu", "CO", "DBMS")
func("Raju", "DMS", "Python", "DS", "SE")
func("Hemanth")
```

```
Output:
Bindu  likes to read  CO DBMS
Raju  likes to read  DMS Python DS SE
Hemanth  likes to read
```

LAMBDA FUNCTIONS (OR) ANONYMOUS FUNCTIONS

- Lambda or anonymous functions are so called because they are not declared as other functions using the def keyword.

- Lambda functions are throw-away functions, i.e. they just needed where they have been created and can be used anywhere a function is required.

Example 1: Write a program that adds two numbers using the syntax of lambda function.

```
sum = lambda x,y:x+y
print("Sum =", sum(3,5))
```

```
Output:
Sum = 8
```

Example 2: Write a program to find smaller of two numbers using lambda function.

```
def small(a,b): # a regular function that returns smaller value
    if(a<b):
        return a
    else:
        return b
sum = lambda x,y:x+y # lambda function to add two numbers
diff = lambda x,y:x-y # lambda function to subtract two numbers
# Pass lambda function as arguments to the regular function
print("Smaller of two numbers = ", small(sum(-3,-2),diff(-1,2)))
```

```
Output:
Smaller of two numbers =  -5
```

MULTILINE DOCUMENT STRINGS

- You can assign a multiline string to a variable by using three quotes.

- Docstrings (document strings) serve the same purpose as of comments, as they are designed to explain the code.

Example 1: Write a program to demonstrate multiline document string.

```
a = """Dear students
Welcome to Deparment of Computer Science,
P.B.Siddhartha College of Arts & Science,
Vijayawada-10, AP, INDIA."""
print(a)
```

```
Output:
Welcome to Deparment of Computer Science,
P.B.Siddhartha College of Arts & Science,
Vijayawada-10, AP, INDIA.
```

Example 2: Write a program to demonstrate multiline document string.

```
a = '''Dear students
Welcome to Deparment of Computer Science,
P.B.Siddhartha College of Arts & Science,
Vijayawada-10, AP, INDIA.'''
print(a)
```

```
Output:
Welcome to Deparment of Computer Science,
P.B.Siddhartha College of Arts & Science,
Vijayawada-10, AP, INDIA.
```

RECURSIVE FUNCTIONS

- Recursive Function: A function that calls itself.

Syntax:

```
def recurse():
    ...
    recurse()
    ...

recurse()
```

recursive call

Example 1: Write a program to calculate the factorial of number recursively.

```
def fact(n):
    if (n==1 or n==0):
        return 1
    else:
        return n*fact(n-1)
n = int(input("Enter the value of n : "))
print("The factorial of",n, "is", fact(n))
```

```
Output:
Enter the value of n : 4
The factorial of 4 is 24
```

Example 2: Write a program to calculate GCD using the recursive functions.

```
def GCD(x,y):
    rem = x % y
    if (rem == 0):
        return y
    else:
        return GCD(y, rem)
n=int(input("Enter the first number : "))
m=int(input("Enter the second number : "))
print("The GCD of numbers is", GCD(n,m))
```

```
Output:
Enter the first number : 62
Enter the second number : 8
The GCD of numbers is 2
```

```
Working:
Assume a = 62 and b = 8
GCD(62, 8)
      rem = 62 % 8 = 6
      GCD(8 % 6)
         rem = 8 % 6 = 2
         GCD(6, 2)
             rem = 6 % 2 = 0
         return 2
      return 2
return 2
```

MODULES

- Module: Modules are pre-written pieces of codes that are used to perform common tasks.

Create & use a Module

- To create a module just save the code you want in a file with the file extension `.py`

Example1: Write a program to create and use file f1 comprises of message “Hello this is TSR KIRAN”

Creating f1.py:

```
def my_function():
    print("Hello this is TSR KIRAN")
```

Creating f2.py:

```
from f1 import *
my_function()
```

Example2: Write a program to import a module that comprises of functions add and subtract to perform addition & subtraction operations.

Creating p3.py:

```
# A simple module, p3.py
def add(x, y):
      return (x+y)
def subtract(x, y):
      return (x-y)
```

Creating p4.py:

```
# importing module p3.py
from p3 import *
print(add(10,2))
print(subtract(10,2))
```

```
Output:
12
8
```

PACKAGES

- Package: A package is a hierarchical file directory structure that has modules and other packages within it.
- Every package in Python is a directory which must have a special file called **_init_.py**
- This file may not even have a single line of code. It is simply added to indicate that this directory is not an ordinary directory and contains a Python package.
- For example, to create a package called MyPackage, create a directory called MyPackage having the module MyModule and the **_init_.py** file. Now, to use MyModule in a program, you must first import it. This can be done in two ways.

```
import MyPackage.MyModule
            or
from MyPackage import MyModule
```

Example 1: Write a program that prints absolute value, square root, and cube of a number.

```
import math
def cube(x):
    return x**3
a=-100
print("a= ",a)
a=abs(a)
print("abs(a) = ", a)
print("Square Root of ",a, " = ", math.sqrt(a))
print("Cube of ",a, " = ", cube(a))
```

```
Output:
a=  -100
abs(a) =  100
Square Root of  100  =  10.0
Cube of  100  =  1000000
```

Example 2: Write a program to gnerate 10 random numbers between 1 to 100

```
import random
for i in range(10):
    value = random.randint(1, 100)
    print(value, end=" ")
```

```
Output:
73 92 65 25 16 83 78 84 76 7
```

Example 3: Write a program to display the date and time using the Time module.

```
import time
localtime = time.asctime(time.localtime(time.time()))
print("Local current time :", localtime)
```

```
Output:
Local current time : Sat Jun  5 00:13:04 2021
```

Example 4: Program that prints the clendar of a particular month.

```
import calendar
print(calendar.month(2017, 1))
```

```
Output:
   January 2017
Mo Tu We Th Fr Sa Su
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

CHAPTER 5
CLASSES & OBJECTS

5.1 CLASSES AND OBJECTE

5.1.1 CLASS

- A class is a user-defined blueprint or prototype from which objects are created.
- Classes provide a means of bundling data and functionality together.
- Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.

- **Some points on Python class:**

1. Classes are created by keyword class.
2. Attributes are the variables that belong to a class.
3. Attributes are always public and can be accessed using the dot (.) operator.
   Eg.: Myclass.Myattribute

Class Definition Syntax:

```
class ClassName:
   # Statement-1
   .
   .
   .
   # Statement-N
```

Example1: Write a program to demonstrate defining a class.

```
class Dog:

       pass
```

Note:

- The `pass` statement is used as a placeholder for future code.
- When the `pass` statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed.

5.1.2 OBJECT

- Object: An Object is an instance of a Class.

- An object consists of

1. **State**: It is represented by the attributes of an object. It also reflects the properties of an object.
2. **Behavior**: It is represented by the methods of an object. It also reflects the response of an object to other objects.
3. **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

Eample 1: Write a program to access class variable using class object.

```
class ABC:
    var = 10 # class variable
obj = ABC()
print(obj.var) # class variable is accessed using class object
```

```
Output:
10
```

Example 2: Write a python program to demonstrate instantiating a class.

```
class Dog:
    # A simple class attribute
    attr1 = "mammal"
    attr2 = "dog"

    # A sample method
    def fun(self):
        print("I'm a", self.attr1)
        print("I'm a", self.attr2)

# Driver code
# Object instantiation
Shadow = Dog()

# Accessing class attributes and method through objects
print(Shadow.attr1)
Shadow.fun()
```

```
Output:
mammal
I'm a mammal
I'm a dog
```

5.1.3 THE SELF

- The self keyword is used to represent an instance (object) of the given class.
- By using the "self" keyword we can access the attributes and methods of the class in python. It binds the attributes with the given arguments.

Example 1: Program to access class membership using the class object.

```
class ABC():
    var=10
    def display(self):
        print("In class method . . .")
obj = ABC()
print (obj.var)
obj.display()
```

```
Output:
10
In class method . . .
```

5.1.4 THE _init_() METHOD (THE CLASS CONSTRUCTOR)

- The _init_( ) method is automatically executed when an object of a class is created.
- The _init_( ) assign values to object properties, or other operations that are necessary to do when the object is being created.

Example 1: Write a program illustrating the use of _init_() method.

```
class ABC():
    def  __init__(self,val):
        print("In class method . . .")
        self.val = val
        print("The value is : ", val)
obj=ABC(10)
```

```
Output:
In class method . . .
The value is :  10
```

Example 2: Write a program to display the attributes name, age by using the _init_() method.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("Ravi Kiran", 45)

print(p1.name)
print(p1.age)
```

```
Output:
Ravi Kiran
45
```

5.1.5 CLASS VARIABLES & OBJECT VARIABLES

- Class Variable: Variable owned by the class is called class variable.
- Object Variable: Variable owned by the object is called object variable.

Example 1: Write a program to differentiate between class and object variable.

```
# Program to differentiate between class and object variable.
class ABC():
    class_var = 0 # class variable
    def __init__(self,object_var):
        ABC.class_var += 1
        self.object_var = object_var # object variable
        print("The Object value is : ",object_var)
        print("The variable of the class variable is :
",ABC.class_var)
obj1 = ABC(10)
obj2 = ABC(20)
obj3 = ABC(30)
```

```
Output:
The Object value is :  10
The variable of the class variable is :  1
The Object value is :  20
The variable of the class variable is :  2
The Object value is :  30
The variable of the class variable is :  3
```

Program 2: Write a python program to show that the variables with a value assigned in class declaration, are class variables.

```
# Python program to show that the variables with a value
# assigned in class declaration, are class variables

# Class for Computer Science Student
class CSStudent:
    stream = 'cse'  # Class Variable
    def __init__(self,name,roll):
        self.name = name         # Instance Variable
        self.roll = roll         # Instance Variable

# Objects of CSStudent class
a = CSStudent('Laksmi', 1)
b = CSStudent('Raju', 2)

print(a.stream) # prints "cse"
print(b.stream) # prints "cse"
print(a.name) # prints "lakshmi"
print(b.name) # prints "Raju"
print(a.roll) # prints "1"
print(b.roll) # prints "2"

# Class variables can be accessed using class
# name also
print(CSStudent.stream) # prints "cse"

# Now if we change the stream for just a it won't be changed for
b
a.stream = 'ece'
print(a.stream) # prints 'ece'
print(b.stream) # prints 'cse'

# To change the stream for all instances of the class we can
change it
# directly from the class
CSStudent.stream = 'mech'

print(a.stream) # prints 'mech'
print(b.stream) # prints 'mech'
```

```
Output:
cse
cse
Laksmi
Raju
1
2
cse
ece
cse
ece
mech
```

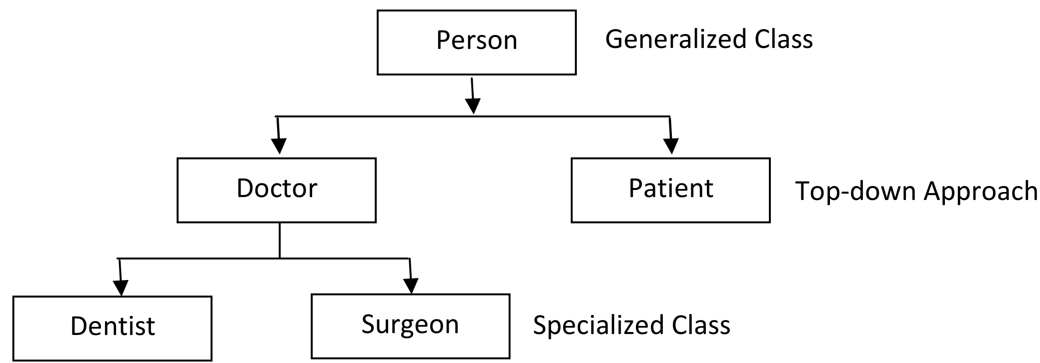
CHAPTER 6
INHERITANCE

6.1 INTRODUCTION

- Inheritance: The capability applying the properties of base class to child class.

- The benefits of inheritance are:

1. It represents real-world relationships well.
2. It provides reusability of a code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
3. It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.



Example 1: Program to demonstrate the use of inheritance.

```
# parent class
class Person:

        # __init__ is known as the constructor
        def __init__(self, name, idnumber):
                self.name = name
                self.idnumber = idnumber
        def display(self):
                print(self.name)
                print(self.idnumber)

# child class
class Employee( Person ):
        def __init__(self, name, idnumber, salary, post):
                self.salary = salary
                self.post = post
```

```
                # invoking the __init__ of the parent class
                Person.__init__(self, name, idnumber)


# creation of an object variable or an instance
a = Employee("TSR Kiran",109, 600000, "HoD")

# calling a function of the class Person using its instance
a.display()
```

```
Output:
TSR Kiran
109
```

6.2 USE THE SUPER( ) FUNCTION

- Python also has a super() function that will make the child class inherit all the methods and properties from its parent.
- By using the super() function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

```
Example1: Program to demonstrate the the use of super() function.
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of",
self.graduationyear)

x = Student("Venkat", "Pola", 2021)
x.welcome()
```

```
Output:
Welcome Venkat Pola to the class of 2021
```

6.2 POLYMORPHISM, FUNCTION OVERLOADING AND METHOD OVERRIDING

6.2.1 POLYMORPHISM

- Polymorphism: Polymorphism is taken from the Greek words Poly (many) and morphism (forms). It means that the same function name can be used for different types. This makes programming more intuitive and easier.

Example 1: Write a python program to demonstrate in-built poly-morphic functions.

```
# len() being used for a string
print(len("geeks"))

# len() being used for a list
print(len([10, 20, 30]))
```

```
Output:
5
3
```

Example2: Write a simple Python function to demonstrate polymorphism.

```
def add(x, y, z = 0):
    return x + y + z

# Driver code
print(add(2, 3))
print(add(2, 3, 4))
```

```
Output:
5
9
```

6.2.2 FUNCTION OVER LOADING

- Function overloading: Function overloading is the ability to have multiple functions with the same name but with different signatures/implementations.
- When an overloaded function is called, the runtime first evaluates the arguments / parameters passed to the function call and judging by this invokes the corresponding implementation.

Example 1: Program to demonstrate polymorphism using function overloading.

```
from multipledispatch import dispatch

#passing one parameter

def product(first,second):
    result = first*second
    print(result);

#passing two parameters

def product(first,second,third):
    result = first * second * third
    print(result)

#you can also pass data type of any value as per requirement

def product(first,second,third):
    result = first * second * third
    print(result);

#calling product method with 2 arguments
product(2,3,2) #this will give output of 12
product(2.2,3.4,2.3) # this will give output of 17.204
```

```
Output:
12
17.204
```

Example 2: Polymorphism using Function Overloading.

```
def add(datatype, *args):
```

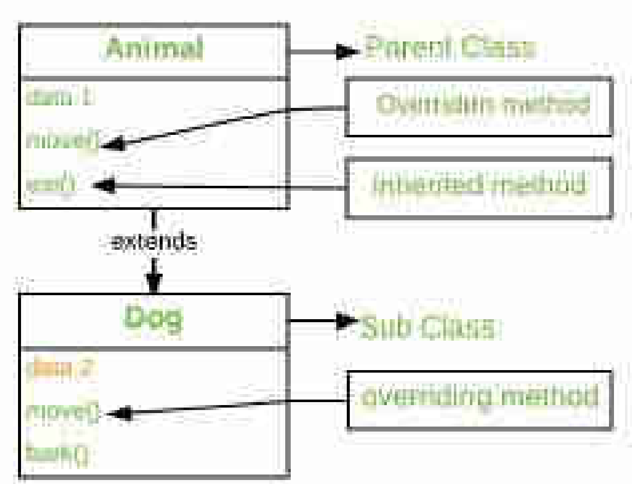
```
    # if datatype is int
    # initialize answer as 0
    if datatype =='int':
        answer = 0

    # if datatype is str
    # initialize answer as ''
    if datatype =='str':
        answer =''

    # Traverse through the arguments
    for x in args:

        # This will do addition if the
        # arguments are int. Or concatenation
        # if the arguments are str
        answer = answer + x

    print(answer)

# Integer
add('int', 5, 6)

# String
add('str', 'Hi ', 'TSR Kiran')
```

```
Output:
11
Hi TSR Kiran
```

6.2.3 METHOD OVERRIDING

- Method Overriding: Method overriding is ability allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super classes or parent classes.

- When a method in a subclass has the same name, same parameters or signature and same return type (or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.



Example1: Python program to demonstrate method overriding.

```
# Defining parent class
class Parent():
    # Constructor
    def __init__(self):
        self.value = "Inside Parent"

    # Parent's show method
    def show(self):
        print(self.value)

# Defining child class
class Child(Parent):
    # Constructor
    def __init__(self):
        self.value = "Inside Child"

    # Child's show method
    def show(self):
        print(self.value)

# Driver's code
obj1 = Parent()
obj2 = Child()

obj1.show()
obj2.show()
```

```
Output:
Inside Parent
Inside Child
```

Example2: Python program to demonstrate method overriding.

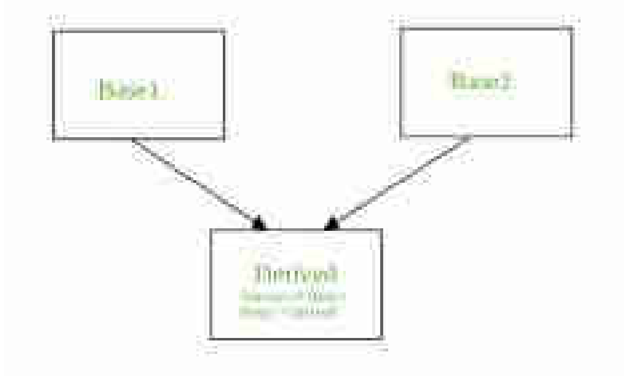
```
class Rectangle():
  def __init__(self,length,breadth):
    self.length = length
    self.breadth = breadth
  def getArea(self):
    print (self.length*self.breadth," is area of rectangle")
class Square(Rectangle):
  def __init__(self,side):
    self.side = side
    Rectangle.__init__(self,side,side)
  def getArea(self):
    print (self.side*self.side," is area of square")

r = Rectangle(3,5)
s = Square(10)

r.getArea()
s.getArea()
```

```
Output:
15  is area of rectangle
100  is area of square
```

6.3 TYPES OF INHERITANCE

6.3.1 MULTIPLE INHERITANCE

- **Multiple Inheritance:** When a class is derived from more than one base class it is called multiple Inheritance.



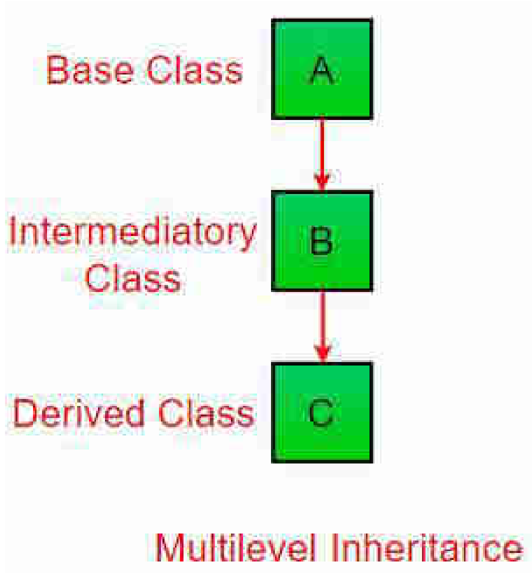
Example: Program to demonstrate multiple inheritance.

```
# Defining parent class 1
class Parent1():
    # Parent's show method
    def show(self):
        print("Inside Parent1")

# Defining Parent class 2
class Parent2():
    # Parent's show method
    def display(self):
        print("Inside Parent2")

# Defining child class
class Child(Parent1, Parent2):
    # Child's show method
    def show(self):
        print("Inside Child")


# Driver's code
obj = Child()
obj.show()
obj.display()
```

```
Output:
Inside Child
Inside Parent2
```

6.3.3 MULTILEVEL INHERITANCE

- Multilevel Inheritance: Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class as the base class for the new class.



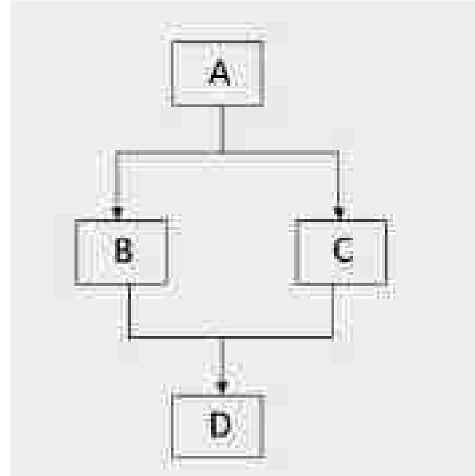
Example1: Python program to demonstrate multilevel inheritance.

```
class Parent():
   # Parent's show method
    def display(self):
        print("Inside Parent")

# Inherited or Sub class (Note Parent in bracket)
class Child(Parent):
    # Child's show method
    def show(self):
        print("Inside Child")

# Inherited or Sub class (Note Child in bracket)
class GrandChild(Child):
    # Child's show method
    def show(self):
        print("Inside GrandChild")

# Driver code
g = GrandChild()
g.show()
g.display()
```

```
Output:
Inside GrandChild
Inside Parent
```

6.3.4 MULTUPATH INHERITANCE (OR) HYBRID INHERITANCE

- Multipath Inheritance: When a class is derived from two or more classes which are derived from the same base class then such type of inheritance is called multipath inheritance. Here class 'D' derived from classes 'B' and 'C', which are derived from same base class 'A'.



Example 1: Program to demonstrate multipath inheritance (or) hybrid inheritance.

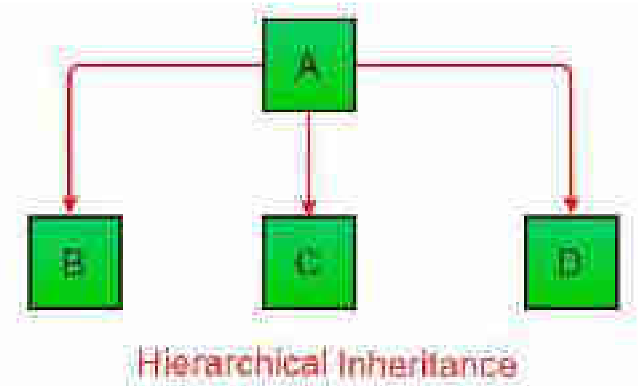
```
# Program to demonstrate multipath inheritance (or) hybrid inheritance

class School:
    def func1(self):
        print("This function is in school.")

class Student1(School):
    def func2(self):
        print("This function is in student 1. ")

class Student2(School):
    def func3(self):
        print("This function is in student 2.")

class Student3(Student1, Student2):
    def func4(self):
        print("This function is in student 3.")
```

```
# Driver's code
object = Student3()
object.func2()
object.func3()
```

```
Output:
This function is in student 1.
This function is in student 2.
```

6.3.5 HIERARCHICAL INHERITANCE

- **Hierarchical Inheritance:** When more than one derived classes are created from a single base this type of inheritance is called hierarchical inheritance. In this program, we have a parent (base) class and two child (derived) classes.



Example 1: Python program to demonstrate Hierarchical inheritance.

```
# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")

# Derived class1
class Child1(Parent):
    def func2(self):
        print("This function is in child 1.")

# Derivied class2
class Child2(Parent):
    def func3(self):
        print("This function is in child 2.")
```

```
# Driver's code
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

```
Output:
This function is in parent class.
This function is in child 1.
This function is in parent class.
This function is in child 2.
```

6.4 ABSTRACT CLASS

- Abstract Class: Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation.
- Abstract classes cannot be instantiated, and require subclasses to provide implementations for the abstract methods.

Example 1: Program to illustrate the concept of abstract class.

```
class Fruit:
    def taste(self):
        raise NotImplementedError()
    def rich_in(self):
        raise NotImplementedError()
    def color(self):
         raise NotImplementedError()

class Mango(Fruit):
    def taste(self):
        return "Sweet"
    def rich_in(self):
        return "Vitamin A"
    def color(self):
        return "Yellow"

class Orange(Fruit):
    def taste(self):
        return "Sour"
    def rich_in(self):
        return "Vitamin C"
```

```
    def color(self):
        return "Orange"
Alphanso = Mango()
print(Alphanso.taste(), Alphanso.rich_in(), Alphanso.color())
Org = Orange()
print(Org.taste(), Org.rich_in(), Org.color())
```

```
Output:
Sweet Vitamin A Yellow
Sour Vitamin C Orange
```

CHAP 7
OPERATOR OVERLOADING

- Operator Overloading: The feature that allows the same operator to have different meaning according to the context is called operator overloading.

```
Example 1: Python Program illustrate how to overload an binary +
operator.

class A:
    def __init__(self, a):
        self.a = a

    # adding two objects
    def __add__(self, o):
        return self.a + o.a
ob1 = A(1)
ob2 = A(2)
ob3 = A("RAVI")
ob4 = A("KIRAN")

print(ob1 + ob2)
print(ob3 + ob4)
```

```
Output:
3
RAVIKIRAN
```

Example 2: Program to overload the + operator on a complex object.

```
class Complex:
    def __init__(self):
        self.real = 0
        self.imag = 0
    def setValue(self, real, imag):
        self.real = real
        self.imag = imag
    def __add__(self, C):
        Temp = Complex()
        Temp.real = self.real + C.real
        Temp.imag = self.imag + C.imag
        return Temp
    def display(self):
        print("(", self.real, " + ", self.imag, "i)")
C1 = Complex()
C1.setValue(1,2)
C2 = Complex()
C2.setValue(3,4)
C3 = Complex()
C3 = C1 + C2
print("RESULT = ")
C3.display()
```

Output:

```
RESULT =
( 4  +  6 i)
```

1 + 2 i
2 + 4 i

CHAPTER 8
ERRORS AND EXCEPTION HANDLING

8.1 INTRODUCTION TO ERRORS AND EXCEPTIONS

- Error: Mistake in a program.
- The programs that we write may behave abnormally or unexpectedly because of *some errors* and *exceptions*.
- Basically, there are *syntax errors* and *logic errors*.
- Syntax error: A syntax error is an error in the *source code* of a program.

Example 1: Program to demonstrate syntax error.

```
# initialize the amount variable
amount = 10000

# check that You are eligible to
# purchase Dsa Self Paced or not
if(amount>2999)
    print("You are eligible to purchase Dsa Self Paced")
```

Output:

```
File "<ipython-input-1-417b442f6752>", line 6
   if(amount>2999)
          ^
SyntaxError: invalid syntax
```

- Logic error: Logic errors occur when there is a fault in the *logic* or *structure* of the problem.

8.2 EXCEPTIONS

- Exception: An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

8.2.1 Handling Exceptions

- An exception can be handled by using `try block` and `except` block.

```
Syntax:
try:
      statements
except ExceptionName:
      statements
```

- The try statement works as follows

Step 1: First, the try block is executed.

Step2 a: If no exception occurs, the except block is skipped.

Step 2 b: If exception occurs, during the execution of any statement in the try block, then

i. Rest of the statement in the try block are skipped.
ii. If the exception type matches the exception named after the except keyword, the except block is executed and then execution continues after the try statement.
iii. If the exception occurs which do not match the named in the except block, then it is passed on to outer try block (in case of nested try blocks). If no exception handler is found in the program, then it is an unhandled exception and the program is terminated with an error message.

Flowchart for Case iii under Step 2b for `try` statement

Example 1: Program to display divide by zero exception.

```
# initialize the amount variable
marks = 10000

# perform division with 0
a = marks / 0
print(a)
```

```
Output:
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-2-9ba2de7eb02e> in <module>
      5
      6 # perform division with 0
----> 7 a = marks / 0
      8 print(a)

ZeroDivisionError: division by zero
```

Example 2: Program to handle divide by zero exception.

```
num =  int(input("Enter the numerator:"))
deno = int(input("Enter the denominator:"))
try:
    quo = int(num / deno)
    print("Quotient:", quo)
except ZeroDivisionError:
    print("Denominator cannot be zero")
```

```
Output:
Enter the numerator:10
Enter the denominator:0
Denominator cannot be zero
```

Example 2: Program to handle multiple errors with one except statement.

```
try :
    a = 3
    if a < 4 :
        # throws ZeroDivisionError for a = 3
        b = a/(a-3)

    # throws NameError if a >= 4
    print ("Value of b = ", b)

# note that braces () are necessary here for multiple exceptions
except(ZeroDivisionError, NameError):
    print ("\nError Occurred and Handled")
```

```
Output:
Error Occurred and Handled
```

8.3 TYPES OF EXCEPTIONS VVIMP

Types of Exceptions		
S.No	Exception Name	Description
1	Exception	Base class for all exceptions
2	StopIteration	Raised when the next() method of an iterator does not point to any object.
3	SystemExit	Raised by the sys.exit() function.
4	StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.
5	ArithmeticError	Base class for all errors that occur for numeric calculation.
6	OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
7	FloatingPointError	Raised when a floating point calculation fails.
8	ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
9	AssertionError	Raised in case of failure of the Assert statement.
10	AttributeError	Raised in case of failure of attribute reference or assignment.
11	EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
12	ImportError	Raised when an import statement fails.

13	KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
14	LookupError	Base class for all lookup errors.
15	IndexError	Raised when an index is not found in a sequence.
16	KeyError	Raised when the specified key is not found in the dictionary.
17	NameError	Raised when an identifier is not found in the local or global namespace.
18	UnboundLocalError	Raised when trying to access a local variable in a function or method but no value has been assigned to it.
19	EnvironmentError	Base class for all exceptions that occur outside the Python environment.
20	IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
21	SyntaxError	Raised when there is an error in Python syntax.
22	IndentationError	Raised when indentation is not specified properly.
23	SystemError	Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
24	SystemExit	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit
25	TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
26	ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
27	RuntimeError	Raised when a generated error does not fall into any category.
28	NotImplementedError	Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

8.4 MULTIPLE EXCEPT BLOCKS

- Python allows multiple except blocks in a single try block. The block which matches with the exception generated will get executed.
- A try block can be associated with more than one except block to specify handlers for different exceptions. However, only one handler will be executed.

```
Syntax:
try:
      Operation are done in this block
      ……………………………………………
```

```
except Exception 1:
      If there is Exception1, then execute this block.
except Exception 2:
      If there is Exception1, then execute this block.
      ……………………………………………
else:
      If there is no exception then execute this block.
      ……………………………………………
```

Example 1: Program with multiple except blocks.

```
try:
    num = int(input("Enter the number : "))
    print(num**2)
except (KeyboardInterrupt):
    print("You have entered a character.....Program Terminating.....")
except (ValueError):
    print("Please check before you enter.....Program
Terminating.....")
print("Bye")
```

```
Output:
Enter the number : abc
Please check before you enter.....Program Terminating.....
Bye
```

8.5 MULTIPLE EXCEPTIONS IN A SINGLE BLOCK

- An `except` clause many have multiple exceptions as a parenthesized tuple. So whatever exception is raised, out of three exceptions specified, the same except block will be executed.

Example 1: Program having an except clause handling multiple exceptions simultaneously.

```
try:
    num = int(input("Enter the number : "))
    print(num**2)
except (KeyboardInterrupt, ValueError, TypeError):
```

```
    print("Please check before you enter.....Program Terminating.....")
print("Bye")
```

```
Output:
Enter the number : abc
Please check before you enter.....Program Terminating.....
Bye
```

8.6 EXCEPT BLOCK WITHOUT EXCEPTION

An except block can be specified without mentioning any exception.

Syntax:

```
try:
      Write the operation here
      ……………………………………………
except:
      If there is any exception, then execute this block.
      ……………………………………………
else:
      If there is no exception then execute this block.
```

Example1: Program to demonstarte the use of except: block

```
try:
    file = pen (File1.txt)
    str = f.readline()
    print(str)
except IOError:
    print("Error occured during Input.....Program Terminating.....")
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error.....Program Terminating.....")
```

```
Output:
Unexpected error.....Program Terminating.....
```

8.7 THE else CLAUSE

- The `try...except` block can optionally have an else clause, which, when present, must follow all `except` blocks.

- The statements in the else block is executed only if the try clause does not raise exception.

Example1: Program to demonstrate the use of else statement in exceptions. VVIMP

```
try:
    num = int(input("Enter a number: "))
    assert num % 2 == 0 # if condition returns True, then nothing happens with assert statement
except:
    print("Not an even number!")
else:
    reciprocal = 1/num
    print(reciprocal)
```

```
Output:
Enter a number: 1
Not an even number!
```

```
Output:
Enter a number: 4
0.25
```

Example 2: Python to demonstrate try...else blocks in exception handling.

```
def divide(x, y):
    try:
        # Floor Division : Gives only Fractional Part as Answer
        result = x // y
    except ZeroDivisionError:
        print("Sorry ! You are dividing by zero ")
    else:
        print("Yeah ! Your answer is :", result)

# Look at parameters and note the working of Program
divide(3, 2)
divide(3, 0)
```

```
Output:
Yeah ! Your answer is : 1
Sorry ! You are dividing by zero
```

8.8 THE finally BLOCK

- The try block has another optional block called finally which is used to define clean-up actions that must be executed under all circumstances.

```
Syntax:
try:
       # Some Code...
except:
       # Optional block
       # Handling of exception (if required)
else:
       # Execute if no exception
finally:
       # Some code ...(always executed)
```

- **Try**: This block will test the excepted error to occur.
- **Except**: Here you can handle the error.
- **Else**: If there is no exception then this block will be executed.
- **Finally**: Finally block always gets executed either exception is generated or not.

Example1: Simple program to demonstrate the use of try...catch...finally block.

```
a = 10
b = 2

try:
    c = int(a / b)
    print(c)
except ZeroDivisionError as error:
    print(error)
finally:
    print('Finishing up.')
```

```
Output:
5
Finishing up.
```

Example2: Simple program to demonstrate the use of try...catch...finally block. VVIMP

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))

try:
    c = int(a / b)
    print(c)
except ZeroDivisionError as error:
    print(error)
finally:
    print('Finishing up.')
```

```
Output:
Enter first number: 10
Enter second number: 0
division by zero
Finishing up.
```

8.9 USER DEFINED EXCEPTIONS

- Programmers may name their own exceptions by creating a new exception class.
- Exceptions need to be derived from the Exception class, either directly or indirectly. Although not mandatory, most of the exceptions are named as names that end in **“Error”** similar to naming of the standard exceptions in python.

Example: Demonstration of user defined exception. VVIMP

```
class Error(Exception):
    """Base class for other exceptions"""
    pass

class ValueTooSmallError(Error):
    """Raised when the input value is too small"""
    pass

class ValueTooLargeError(Error):
    """Raised when the input value is too large"""
    pass

# you need to guess this number
number = 10

# user guesses a number until he/she gets it right
while True:
    try:
        i_num = int(input("Enter a number: "))
        if i_num < number:
            raise ValueTooSmallError
        elif i_num > number:
            raise ValueTooLargeError
        break
    except ValueTooSmallError:
        print("This value is too small, try again!")
        print()
```

```
    except ValueTooLargeError:
        print("This value is too large, try again!")
        print()

print("Congratulations! You guessed it correctly.")
```

```
Output:
Enter a number: 12
This value is too large, try again!

Enter a number: 4
This value is too small, try again!

Enter a number: 10
Congratulations! You guessed it correctly.
```

CHAPTER 8
REGULAR EXPRESSION

- A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.
- RegEx can be used to check if a string contains the specified search pattern.

Example1: Program to check if the string starts with "The" and ends with "Spain".

```
import re
#Check if the string starts with "The" and ends with "Spain":
txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
if x:
  print("YES! We have a match!")
else:
  print("No match")
```

```
Output:
YES! We have a match!
```

8.1 RegEx FUNCTIONS

- The re module offers a set of functions that allows us to search a string for a match.

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

8.1.1 findall()

- The `findall` function returns a list containing all matches.

  Example1: Program to demonstrate findall function.

```
import re
#Return a list containing every occurrence of "ai":
txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

```
Output:
['ai', 'ai']
```

- The list contains the matches in the order they are found.
- If no matches are found, an empty list is returned.

Example2: Program to demonstrate No match in regular expression.

```
import re
txt = "The rain in Spain"
#Check if "Portugal" is in the string:
x = re.findall("Portugal", txt)
print(x)
if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
[]
No match
```

8.1.2 Search

- The `search()` function searches the string for a match, and returns a Match object if there is a match.
- If there is more than one match, only the first occurrence of the match will be returned.

Example1: Program to find first white space character located in position of a string.

```
import re
txt = "The rain in Spain"
x = re.search("\s", txt) # \s is white space character
print("The first white-space character is located in position:", x.start())
```

```
Output: The first white-space character is located in position: 3
```

8.1.3 Split

- The `split()` function returns a list where the string has been split at each match.

Example 1: Program to demonstrate split the string at every white space character.

```
import re
#Split the string at every white-space character:
txt = "The rain in Spain"
x = re.split("\s", txt) # \s is white space character
print(x)
```

```
Output Character:
['The', 'rain', 'in', 'Spain']
```

- You can control the number of occurrences by specifying the `maxsplit` parameter.

Example 2: Program to demonstrate to split the string at the first white space character.

```
import re
```

```
#Split the string at the first white-space character:
txt = "The rain in Spain"
x = re.split("\s", txt, 1) # \s is white space character
print(x)
```

```
Output:
['The', 'rain in Spain']
```

8.1.4 sub()

- The `sub()` function replaces the matches with the text of your choice.

Example 1: Program to replace all white space characters with the digit 9.

```
import re
#Replace all white-space characters with the digit "9":
txt = "The rain in Spain"
x = re.sub("\s", "9", txt) # \s is white space character
print(x)
```

```
Output:
The9rain9in9Spain
```

- You can control the number of replacements by specifying the count parameter.

Example 2: Program to replace the first two occurrences of a white-space character with the digit 9.

```
import re
txt = "The rain in Spain"
x = re.sub("\s", "9", txt, 2) # \s is white space character
print(x)
```

```
Output:
The9rain9in Spain
```

8.2 METACHARACTERS

- Metacharacters are characters with a special meaning.

<table>
<tr><th>Character</th><th>Description</th><th>Example</th></tr>
<tr><td>[]</td><td>A set of characters</td><td>"[a-m]"</td></tr>
<tr><td>\</td><td>Signals a special sequence (can also be used to escape special characters)</td><td>"\d"</td></tr>
<tr><td>.</td><td>Any character (except newline character)</td><td>"he..o"</td></tr>
<tr><td>^</td><td>Starts with</td><td>"^hello"</td></tr>
<tr><td>$</td><td>Ends with</td><td>"world$"</td></tr>
<tr><td>*</td><td>Zero or more occurrences</td><td>"aix*"</td></tr>
<tr><td>+</td><td>One or more occurrences</td><td>"aix+"</td></tr>
<tr><td>{}</td><td>Exactly the specified number of occurrences</td><td>"al{2}"</td></tr>
<tr><td>|</td><td>Exactly specify one or two strings</td><td>falls|stays</td></tr>
</table>

Example 1: Python program to match string using metacharacter []

```
import re
txt = "The rain in Spain"
#Check if the string starts with "The":
x = re.findall("\AThe", txt)
print(x)
if x:
  print("Yes, there is a match!")
else:
  print("No match")
```

```
Output:
['The']
Yes, there is a match!
```

Example 2: Program to find digits in character using metacharacter \

```
import re
txt = "That will be 59 dollars"
#Find all digit characters
x = re.findall("\d", txt)
print(x)
```

```
Output:
['5', '9']
```

Example 3: Program for sequence that starts with "he", followed by two (any) characters using metacharacter ..

```
import re
txt = "hello world"
#Search for a sequence that starts with "he", followed by two (any) characters, and an "o":
x = re.findall("he..o", txt)
print(x)
```

```
Output:
['hello']
```

Example 4: Program to check if the string starts with 'hello' using metacharacter ^

```
import re
txt = "hello world"
#Check if the string starts with 'hello':
x = re.findall("^hello", txt)
if x:
  print("Yes, the string starts with 'hello'")
else:
  print("No match")
```

```
Output:
Yes, the string starts with 'hello'
```

Example 5: Program to check the string ends with 'world' using metacharacter $

```
import re
txt = "hello world"
#Check if the string ends with 'world':
x = re.findall("world$", txt)
if x:
  print("Yes, the string ends with 'world'")
else:
  print("No match")
```

```
Output:
Yes, the string ends with 'world'
```

Example 6: Program to check the string contains "ai" followed by 0 or more "x" characters

```
import re
txt = "The rain in Spain falls mainly in the plain!"
#Check if the string contains "ai" followed by 0 or more "x" characters:
x = re.findall("aix*", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['ai', 'ai', 'ai', 'ai']
Yes, there is at least one match!
```

Example 7: Program to check the string contains "ai" followed by 1 or more "x" characters

```
import re
txt = "The rain in Spain falls mainly in the plain!"
#Check if the string contains "ai" followed by 1 or more "x" characters:
x = re.findall("aix+", txt)
print(x)

if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
[]
No match
```

Example 9: Program to check if the string contains "a" followed by exactly two "l" characters

```
import re
txt = "The rain in Spain falls mainly in the plain!"
#Check if the string contains "a" followed by exactly two "l" characters:
x = re.findall("all{2}", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['all']
Yes, there is at least one match!
```

Example 10: Program to check if the string contains either "falls" or "stays" using meta character |

```
import re
txt = "The rain in Spain falls mainly in the plain!"
#Check if the string contains either "falls" or "stays":
x = re.findall("falls|stays", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['falls']
Yes, there is at least one match!
```

8.3 SPECIAL SEQUENCES

- A special sequence is a \ followed by one of the characters in the list below, and has a special meaning.

Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word. (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word. (the "r" in the beginning is making sure that	r"\Bain" r"ain\B"

	the string is being treated as a "raw string")	
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

Example 1: Program to check if the string starts with "The"

```
import re
txt = "The rain in Spain"
#Check if the string starts with "The":
x = re.findall("\AThe", txt)
print(x)
if x:
  print("Yes, there is a match!")
else:
  print("No match")
```

```
Output:
['The']
Yes, there is a match!
```

Example 2: Program to check if "ain" is present at the begining of a word

```
import re
txt = "The rain in Spain"
#Check if "ain" is present at the beginning of a WORD:
x = re.findall(r"\bain", txt) # r specifies raw string
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
[]
No match
```

Example 3: Program to check if "ain" is present at the end of a word

```
import re
txt = "The rain in Spain"
#Check if "ain" is present at the end of a WORD:
x = re.findall(r"ain\b", txt) # r specifies raw string
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['ain', 'ain']
Yes, there is at least one match!
```

Example 3: # Program to check if "ain" is present, but NOT at the begining of a word

```
import re
txt = "The rain in Spain"
# Check if "ain" is present, but NOT at the beginning of a word
x = re.findall(r"\Bain", txt) # r specifies a raw string
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['ain', 'ain']
Yes, there is at least one match!
```

Example 4: Program to check if "ain" is present, but NOT at the end of a word

```
import re
txt = "The rain in Spain"
# Check if "ain" is present, but NOT at the end of a word
x = re.findall(r"ain\B", txt) # r specifies a raw string
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
[]
No match
```

Example 5: Program to Check if the string contains any digits (numbers from 0-9)

```
import re
```

```
txt = "The rain in Spain 1975"
# Check if the string contains any digits (numbers from 0-9):
x = re.findall("\d", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
[]
No match
```

Example 6: Program to return a match at every no-digit character.

```
import re
txt = "The rain in Spain"
# Return a match at every no-digit character
x = re.findall("\D", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['T', 'h', 'e', ' ', 'r', 'a', 'i', 'n', ' ', 'i', 'n', ' ', 'S', 'p', 'a', 'i', 'n']
Yes, there is at least one match!
```

Example 7: Program to return a match at every white-space character.

```
import re
txt = "The rain in Spain"
```

```
# Return a match at every white-space character:
x = re.findall("\s", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
[' ', ' ', ' ']
Yes, there is at least one match!
```

Example 8: Program to return a match at every NON white-space character.

```
import re
txt = "The rain in Spain"
#Return a match at every NON white-space character
x = re.findall("\S", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['T', 'h', 'e', 'r', 'a', 'i', 'n', 'i', 'n', 'S', 'p', 'a', 'i', 'n']
Yes, there is at least one match!
```

Example 9: Program to return a match at every word character (characters from a to Z, digits from 0-9, and the underscore _ character)

```
import re
txt = "The rain in Spain"
# Return a match at every word character (characters from a to Z, digits from 0-9, and the underscore _
character)
x = re.findall("\w", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['T', 'h', 'e', 'r', 'a', 'i', 'n', 'i', 'n', 'S', 'p', 'a', 'i', 'n']
Yes, there is at least one match!
```

Example 10: Program to return a match at every NON word character (characters NOT between a and Z. Like "!", "?" white-space etc.)

```
import re
txt = "The rain in Spain"
# Return a match at every NON word character (characters NOT between a and Z. Like "!", "?" white-space
etc.)
x = re.findall("\W", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
[' ', ' ', ' ']
```

```
Yes, there is at least one match!
```

Example 11: Program to check if the string ends with "Spain".

```
import re
txt = "The rain in Spain"
#Check if the string ends with "Spain":
x = re.findall("Spain\Z", txt)
print(x)
if x:
  print("Yes, there is a match!")
else:
  print("No match")
```

```
Output:
['Spain']
Yes, there is a match!
```

8.4 REGULAR EXPRESSION SETS

- A set is a set of characters inside a pair of square brackets [] with a special meaning.

Set	Description
[arn]	Returns a match where one of the specified characters (a, r, or n) are present
[a-n]	Returns a match for any lower case character, alphabetically between a and n
[^arn]	Returns a match for any character EXCEPT a, r, and n
[0123]	Returns a match where any of the specified digits (0, 1, 2, or 3) are present

[0-9]	Returns a match for any digit between 0 and 9
[0-5][0-9]	Returns a match for any two-digit numbers from 00 and 59
[a-zA-Z]	Returns a match for any character alphabetically between a and z, lower case OR upper case
[+]	In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string

Example 1: Program Check if the string has any a, r, or n characters.

```
import re
txt = "The rain in Spain"
# Check if the string has any a, r, or n characters:
x = re.findall("[arn]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['r', 'a', 'n', 'n', 'a', 'n']
Yes, there is at least one match!
```

In [ ]:

Example 2: Program to Check if the string has any characters between a and n.

```
import re
txt = "The rain in Spain"
# Check if the string has any characters between a and n
x = re.findall("[a-n]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
```

```
else:
  print("No match")
```

```
Output:
['h', 'e', 'a', 'i', 'n', 'i', 'n', 'a', 'i', 'n']
Yes, there is at least one match!
```

Program 3: Program to Check if the string has other characters than a, r, or n.

```
import re
txt = "The rain in Spain"
# Check if the string has other characters than a, r, or n
x = re.findall("[^arn]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['T', 'h', 'e', ' ', 'i', ' ', 'i', ' ', 'S', 'p', 'i']
Yes, there is at least one match!
```

Program 4: Program to check if the string has any 0, 1, 2, or 3 digits.

```
import re
txt = "The rain in Spain"
# Check if the string has any 0, 1, 2, or 3 digits
x = re.findall("[0123]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
```

```
else:
  print("No match")
```

```
Output:
[]
No match
```

Program 5: Program to check if a string has any digits.

```
import re
txt = "8 times before 11:45 AM"
# Check if the string has any digits
x = re.findall("[0-9]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['8', '1', '1', '4', '5']
Yes, there is at least one match!
```

Program 6: Program to check if the string has any two-digit numbers, from 00 to 59.

```
import re
txt = "8 times before 11:45 AM"
# Check if the string has any two-digit numbers, from 00 to 59
x = re.findall("[0-5][0-9]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['11', '45']
Yes, there is at least one match!
```

Program 7: Program to Check if the string has any characters from a to z lower case, and A to Z upper case.

```
import re
txt = "8 times before 11:45 AM"
x = re.findall("[a-zA-Z]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
['t', 'i', 'm', 'e', 's', 'b', 'e', 'f', 'o', 'r', 'e', 'A', 'M']
Yes, there is at least one match!
```

Program 8: Program to check if the string has any + characters.

```
import re
txt = "8 times before 11:45 AM"
# Check if the string has any + characters
x = re.findall("[+]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
Output:
```

[]
No match

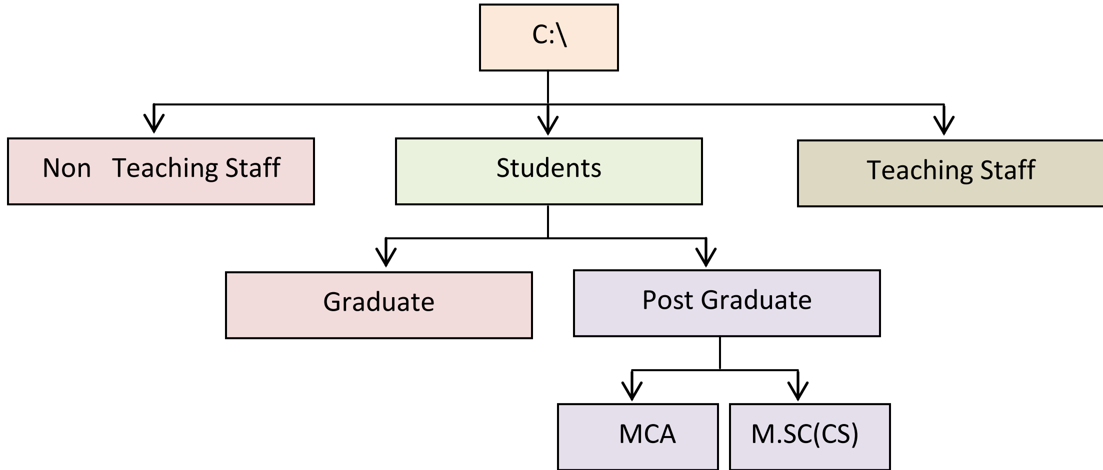
CHAPTER 10
FILE HANDLING

10.1 INTRODUCTION

- **File:** A file is some information or data which stays in the computer storage devices.

10.1.1 File Path

- Most file systems that are used to day stores files in Tree Structure.
- At the top of the Tree there is one Root Node. Under the Root Node, there are other files and folders. Even these folders can in turn contain other files and folders and this can go on to an almost limitless depth.
- A file path can be relative or absolute.



- **Absolute Path:** Absolute path always contains the root and complete directory list to specify the exact location of the file, relative path, on the other hand, needs to be combined with another path in order to access a file.

  E.g. **C:\Students\Post Graduate\MCA**

- Relative Path: The relative path is part of absolute path. The relative paths starts with respect to the current working directory.

  E.g. **Post Graduate\MCA**

10.2 FILE HANDLING

10.2.1 Open Function:

- We use open () function in Python to open a file in read or write mode.
- Syntax: open (filename, mode).
- There are three kinds of mode, that Python provides and how files can be opened:

```
"r", for reading.
"w", for writing.
"a", for appending.
"r+", for both reading and writing.
```

Program 1: Write a program to open a file using open method.

```
f = open('d://demofile.txt', 'r')
print(f.read())
```

```
Output:
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

10.2.2 Read Only Parts of the File

- By default the read() method returns the whole text, but you can also specify how many characters you want to return.

Program 2: Write a program to return the 5 first characters of the file.

```
f = open("d:\\demofile.txt", "r")
print(f.read(5))
```

```
Output:
Hello
```

10.2.3 Read Lines

- You can return one line by using the readline() method.

Program 3: Write a program to read a line from a file.

```
f=open("d:\\demofile.txt", "r")
print(f.readline())
```

```
Output:
Hello! Welcome to demofile.txt
```

10.2.4 Python File Write

Writing a File

```
Program 4: Program to write
# Python code to create a file in write mode
file = open('d:\\x.txt','w')
file.write("This is the write command")
file.write("It allows us to write in a particular file")
file.close()
```

Using write along with with() function

```
Program 5:  Write python code to illustrate with() along with write()

with open("d:\\y.txt", "w") as f:
    f.write("Hello World!!!")
```

Appending a File

- To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file.
"w" - Write - will overwrite any existing content.

```
Program 6: Program to append a new line to the existing file.

f = open("d:\\demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("d:\\demofile2.txt", "r")
print(f.read())
```

```
Output:
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!Now the file has more content!
```

```
Program 7: Program to open the file "demofile3.txt" and overwrite the content.

f = open("d:\\demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()
```

```
#open and read the file after the appending:
f = open("d:\\demofile3.txt", "r")
print(f.read())
```

```
Output:
Woops! I have deleted the content!
```

Removing a File:

Program 8: Program to remove the existing file demofile.

```
import os
if os.path.exists("d:\\demofile"):
   os.remove("d:\\demofile")
else:
      print("The file does not exist")
Output:
The file does not exist
```

Split() using File Handling:

Program 9: Program to illustrate split() function.

```
with open("d:\\x.txt", "r") as file:
    data = file.readlines()
    for line in data:
        word = line.split()
        print (word)
```

```
Output:
['This', 'is', 'the', 'write', 'command', 'It', 'allows', 'us', 'to',
'write', 'in', 'a', 'particular', 'file']
```

Program 10: Write a program to open the file and count the number of times a character appears in the file.

```
with open("d:\\x.txt") as file:
    text = file.read()
    letter = input("Enter the character to be searhed : ")
    count = 0
    for char in text:
        if char == letter:
            count = count+1
print(letter, "appears" , count, "times in file")
```

```
Output:
Enter the character to be searhed : a
a appears 5 times in file
```

```
Program 11: Write a program that reads data from a file and calculates
# the percentage of vowels and consonants in the file.

with  open("d:\\x.txt") as file:
    text = file.read()
    count_vowels = 0
    count_consonants = 0
    for char in text:
        if char in "aeiou":
            count_vowels = count_vowels + 1
        else:
            count_consonants = count_consonants + 1
print("Number of vowels = ", count_vowels)
print("Number of consonants = ",count_consonants)
print("Total Length of File = ", len(text))
print("Percentage of vowels in the file = ",((count_vowels)*100)/len(text),"%")
print("Percentage of consonants in the file = ",((count_consonants)*100)/len(text), "%")
```

```
Output:
Number of vowels =  21
Number of consonants =  47
Total Length of File =  68
Percentage of vowels in the file =  30.88235294117647 %
Percentage of consonants in the file =  69.11764705882354 %
```

CHAPTER 11
PYTHON DATABASES

Program 1: Write a python program to create EMP table with attributes ENO,ENAME and ESAL into PBS database.

```
import pymysql

#database connection
connection = pymysql.connect(host="localhost", user="root", passwd="",
database="PBS")
cursor = connection.cursor()
# Query for creating table
EmpTableSql = """CREATE TABLE EMP(ENO INT(5) PRIMARY KEY,ENAME  CHAR(10) NOT
NULL,ESAL INT(5))"""
cursor.execute(EmpTableSql)
connection.close()
```

Program 2: Write a python program to insert rows into EMP table of PBS database.

```
import pymysql

# Database connection
connection = pymysql.connect(host="localhost", user="root", passwd="",
database="PBS")
cursor = connection.cursor()

# Queries for inserting values
insert1 = "INSERT INTO EMP(ENO, ENAME, ESAL) VALUES(1, 'PAVANI', 6000 );"
insert2 = "INSERT INTO EMP(ENO, ENAME, ESAL) VALUES(2, 'VENKAT', 3000 );"

# Executing the quires
cursor.execute(insert1)
cursor.execute(insert2)


# Commiting the connection then closing it.
connection.commit()
connection.close()
```

Program 3: Write a python program to update rows of EMP table of PBS database.

```
import pymysql

# Database connection
connection = pymysql.connect(host="localhost", user="root", passwd="",
database="PBS")
cursor = connection.cursor()
updateSql = "UPDATE EMP SET ENAME = 'ABHI'  WHERE ENO = '1' ;"
cursor.execute(updateSql)
connection.commit()
connection.close()
```

Program 4: Write a python program to delete rows from EMP table of PBS database.

```
import pymysql

# Database connection
connection = pymysql.connect(host="localhost", user="root", passwd="",
database="PBS")
cursor = connection.cursor()
deleteSql = "DELETE FROM EMP WHERE ENO = '2'; "
cursor.execute(deleteSql )
connection.commit()
connection.close()
```

Program 5: Write a python program to drop EMP table of PBS database.

```
import pymysql

# Database connection
connection = pymysql.connect(host="localhost", user="root", passwd="",
database="PBS")
cursor = connection.cursor()
dropSql = "DROP TABLE IF EXISTS EMP;"
cursor.execute(dropSql)
connection.commit()
connection.close()
```