OODB and Distributed Database

Unit 6

Object Database Management System

ODBMS

Data is stored in form of **objects**, which are instances of classes.

Adding DBMS functionality to a programming language environment.

In comparison with RDBMS, where data is stored in tables with rows and columns, ODBMS stores information as **objects.**

Limitation of Relational Databases

Relational DBMSs support a **small**, **fixed collection of data types** (e.g. integer, dates, string, etc.).
This is **adequate for traditional application domains** such as administrative and business data processing.

Many other application domains need **complex kinds of data** such as CAD/CAM, multimedia repositories, and document management. To support such applications, DBMSs must support **complex** data types.

Relational model **does not allow extending** these data types or **creating the user's own** data types. Limiting the types of data that may be represented using relational databases.

RDBMS does not offer inheritance & other OOPS features.

Object-database systems have developed along two distinct paths:

(1) **Object-Oriented Database Systems.** DBMS functionality added to a OO programming language environment.

– **The Object Database Management Group** (ODMG) - developed a standard **Object Data Model (ODM) Object Query Language (OQL) (Object Definition Language (ODL)** and **Object Manipulation Language (OML)**) which are the equivalent of the SQL standard for relational database systems.

(2) **Object-Relational Database Systems.** Relational database systems extended with the functionality to **support a broader class of application domains.**

Provide a bridge between the relational and object-oriented paradigms.

This approach attempts to get the best of both.

– RDDMS vendors, such as IBM, Informix, ORACLE have added ORDBMS functionality to their products.

Basic OO Concepts

Object and Class

A conceptual entity is anything that exists and can be distinctly identified. E.g. a person, an employee, a car, a part

- **In an OO system - all conceptual entities are modeled as objects.**

- An object has **structural properties** defined by a finite set of **attributes** and **behavioral properties** defined by a finite **set of methods**.

- **An Object has**
  - **A unique identifier (OID)**
  - **A name**
  - **A lifetime defining whether it is persistent or not, and**
  - **A structure that may be created using a type constructor.**

Persistent object (that is, a database object)

Transient object (that is, an object in an executing program that disappears after the program terminates).

Object is constructed using the type constructors.

Object Identifier (OID)

Each object is associated with a **logical non-reusable and unique object identifier** (OID).

The OID of an object is independent of the values of its attributes.

• All objects with the same set of attributes and methods are grouped into a **class, and form instances of that class.**

Object Identifier (OID)…….

OID is different from key in the relational data model.

A key is defined by the value of one or more attributes and can be modified.

Example of OID:
000028020948A19E8DE0697291E0340800208D6C1D48A19E8DE0687291E034
0800208D6C1D04000AE10000

Classes are classified as lexical classes and non-lexical classes.

- **Lexical class** - contains objects directly represented by their values.
  E.g. integer, string.

- **Non-lexical class** - contains objects, each represented by a set of attributes and methods.

- **Instances of a non-lexical class are referred to by their OIDs.**
  E.g. PERSON, EMPLOYEE, PART are non-lexical classes.

Type Constructors in OO databases:

In OO databases, the state (current value) of a complex object may be constructed from other objects (or other values) by using certain type constructors.

The **three most basic constructors** are **atom**, **tuple**, and **set**.

Other commonly used constructors include **list**, **bag**, and **array**.

The **atom constructor** is used to represent **all basic** atomic values, such as integers, real numbers, character strings, Booleans, and any other basic data types that the system supports directly.
Tuple constructor is used to create a tuple. **Tuple()** creates empty tuple.

Set, List, Array, Bag Constructors

- Also referred as **"generator"** constructors.
- Set, list, array, and bag—are collection types or bulk types or complex types.

- A **set** is a group of similar (alike) things.
- A **list** is similar to a set, only it is specifically ordered. Because we know the sequence, we can refer to it by position, such as the nth object in a list.
- A **bag** is also similar to a set except that it allows **duplicates** to exist within the set captured in the complex object.
- An **array** is similar to a list, with a third dimension added that we can also address by positional reference.

Object Representation

Examples:

o0 = (i0, atom, 815)

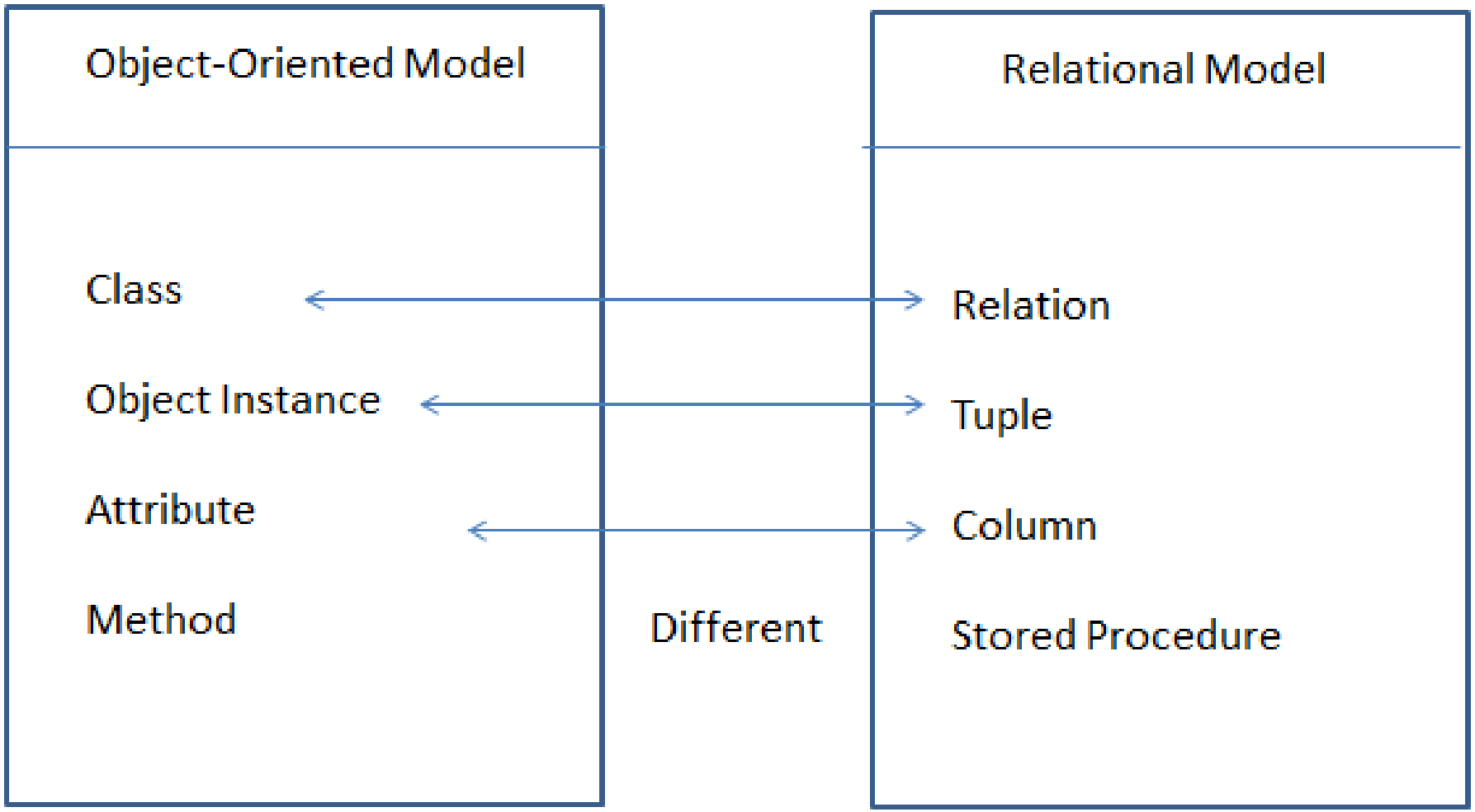
o1 = (i1, atom, UC Davis)

o2 = (i2, atom, Computer Science)

o3 = (i3, atom, Art)

o4 = (i4, set, {i1, i2})

o5 = (i5, tuple, □ University: i1, Major: i2□)


Object-Oriented Model
Relational Model
Class
Relation
Object Instance
Tuple
Attribute
Column
Method
Different
Stored Procedure

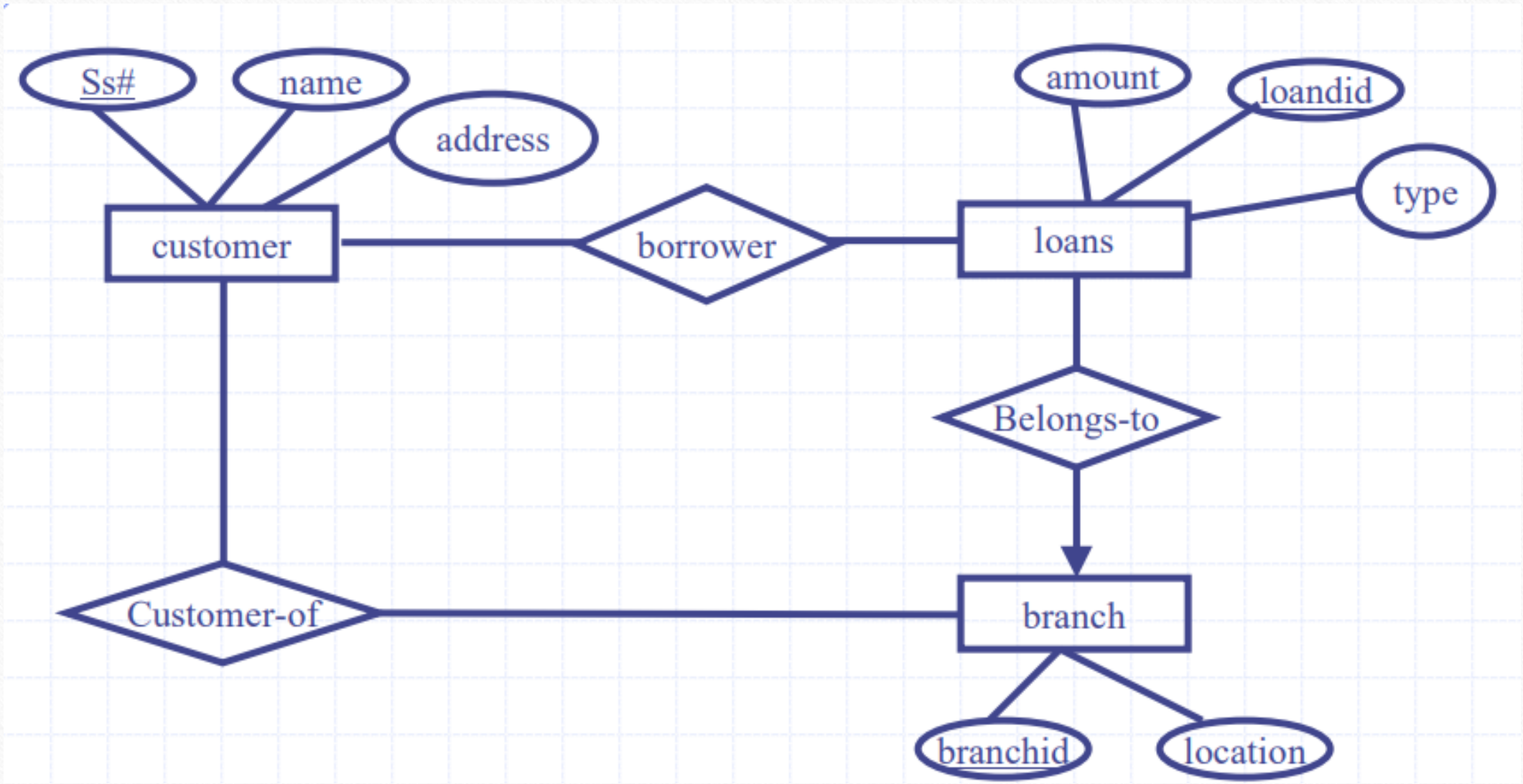
Object Definition Language
ODL

ODL

Design & Specification language derived from the OO community.

To define object specifications.
Defines properties and operations.

Define the structure of an Entity-relationship diagram.


Ss#
name
address
customer
borrower
amount
loandid
type
loans
Belongs-to
Customer-of
branch
branchid
location

Object Query Language

Object Query Language (OQL):
Version of SQL to interact with Object oriented databases.

Declarative (not procedural) language

Statements in OQL

```
create database staff;


create table staff.managers
(
        EmployeeID int   NOT NULL   PRIMARY KEY, Name text  NOT NULL,
        Department text  default "Sales", Gender text, Age int, unique ( EmployeeID )
);
```

```
insert into database_name.table_name
(column [ , column ] [ , column ]  [ ... ]  ) values (data [ , data ] [ , data ]  [ ... ] );

insert into staff.managers
(
        EmployeeID, Name, Department, Gender, Age
)
values
(
        1, “Ajay", "Development", "M", 28
);
```

select * from staff.managers;

{

EmployeeID=1;

Name='Matt';

Department='Development';

Gender='M';

Age=28;

}

{

EmployeeID=2;

Name=Ajay';

Department=‘Testing';

Gender='M';

Age=26;

}

```
select EmployeeID, Name from staff.managers where Department = "Marketing";

select Age, Gender into staff.employees from staff.managers;

update staff.managers set Age=27 where Name="John";
```

Distributed Database

A **distributed database (DDB)** is a collection of **multiple, logically interrelated databases** distributed over a computer network.

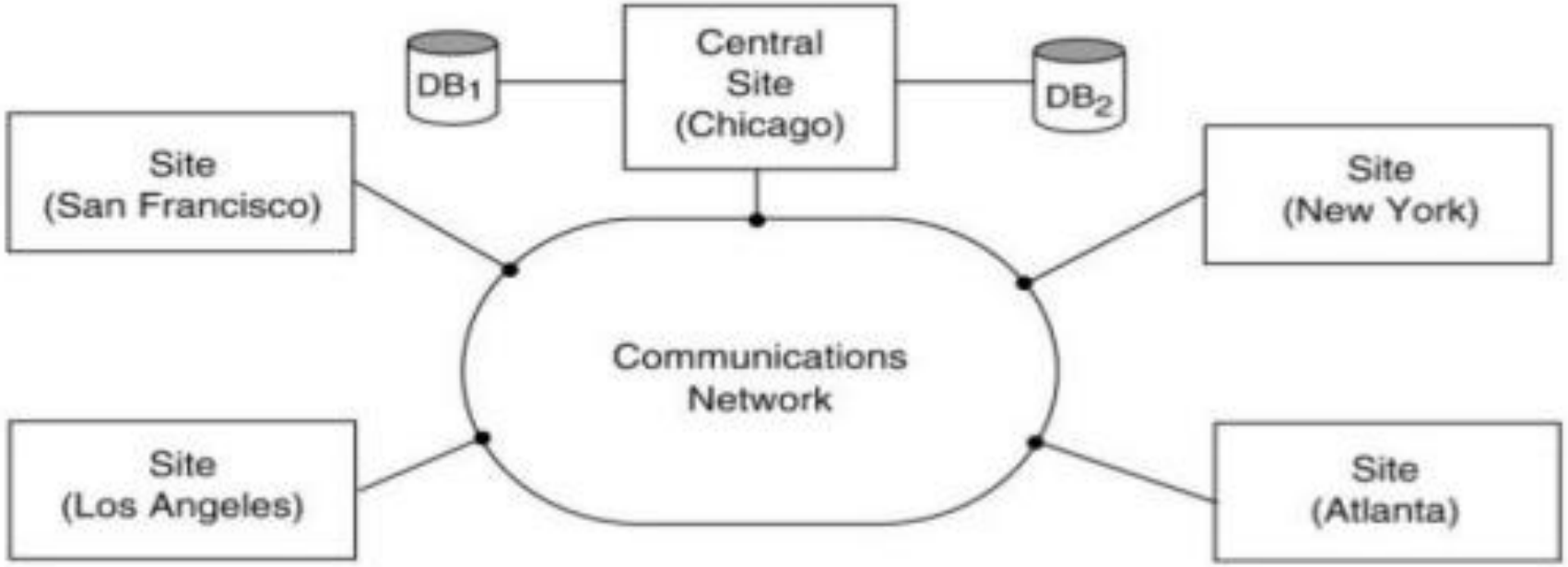
A **distributed database management system (D–DBMS)** is the software that manages the DDB and provides an access mechanism that makes this distribution **transparent** to the users.

Appears to a user as a single database

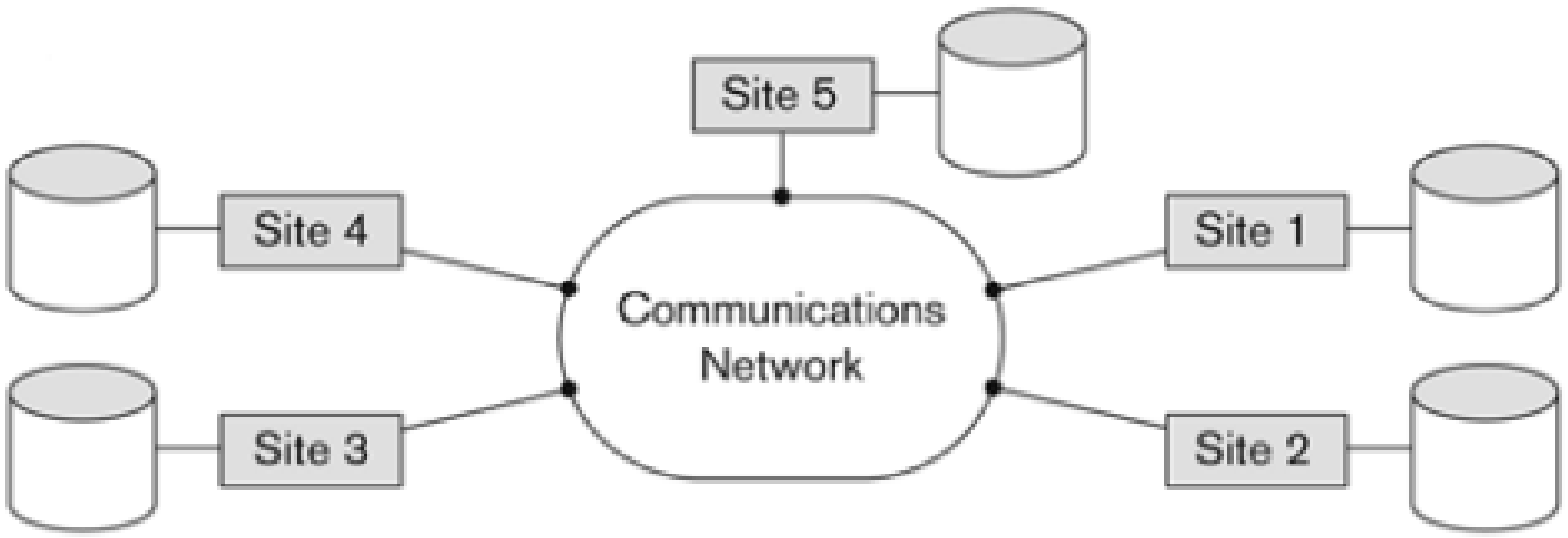
Data on several computers can be simultaneously accessed and modified.

Database server in the distributed database is controlled by its local DBMS.

Centralized database



Distributed database



Advantages of Distributed Databases

- **Modular Development** − If the system needs to be expanded to new locations or new units, in centralized database systems, the action requires substantial efforts and disruption in the existing functioning. However, in distributed databases, the work simply requires adding new computers and local data to the new site and finally connecting them to the distributed system, with no interruption in current functions.

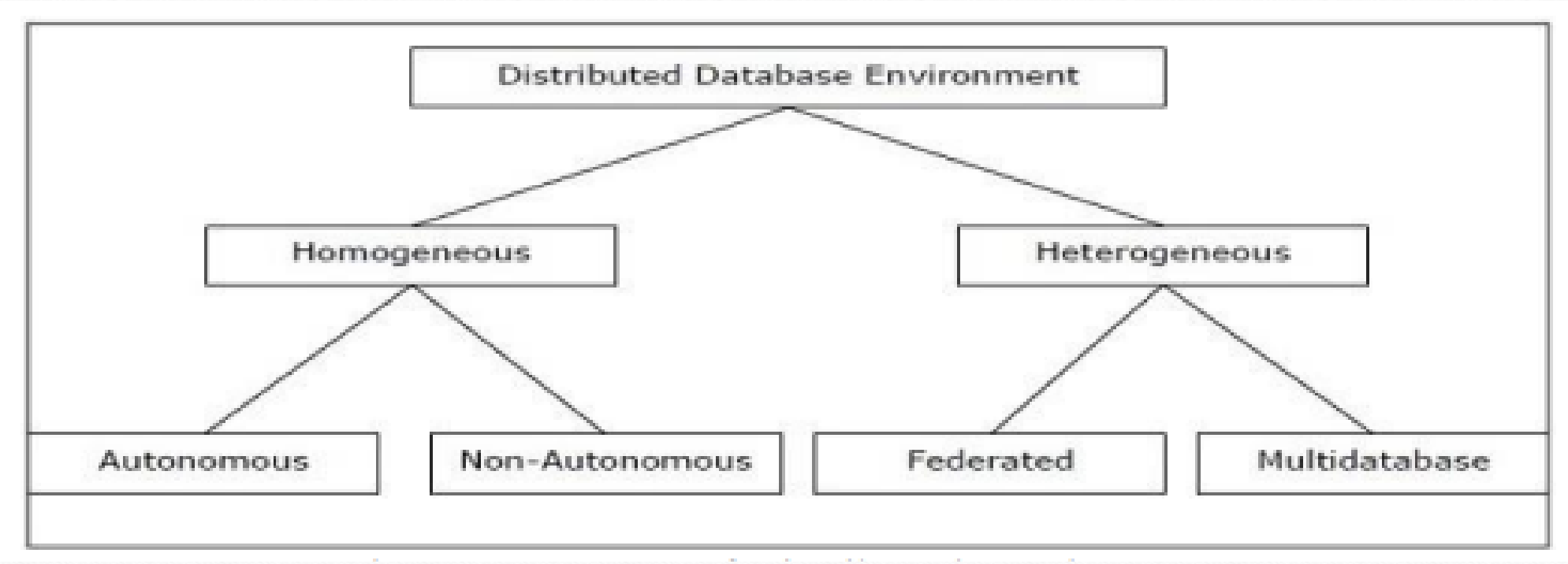
- **More Reliable** − In case of database failures, the total system of centralized databases comes to a halt. However, in distributed systems, when a component fails, the functioning of the system continues may be at a reduced performance. Hence DDBMS is more reliable.

- **Better Response** − If data is distributed in an efficient manner, then user requests can be met from local data itself, thus providing faster response. On the other hand, in centralized systems, all queries have to pass through the central computer for processing, which increases the response time.

- **Lower Communication Cost** − In distributed database systems, if data is located locally where it is mostly used, then the communication costs for data manipulation can be minimized. This is not feasible in centralized systems.

Centralized DBMS	Distributed DBMS
In Centralized DBMS the database are stored in a only **one site**	In Distributed DBMS the database are stored in **different site** and help of network it can access it
If the data is stored at a **single computer site**, which can be used by **multiple users**	Database and DBMS software **distributed over many sites**, connected by a computer network
Database is maintained **at one site**	Database is maintained at a **number of different sites**
If centralized system fails, entire system is halted	If one system fails,s ystem continues work with other site
It is a less reliable	It is a more reliable

Types of Distributed Databases



Two types:
homogeneous
and
heterogeneous

Homogeneous Distributed Databases

All different sites store database identically.

The operating system, database management system, and the data structures used – all are the same at all sites.

- All sites have identical software
- Are aware of each other and agree to cooperate in processing user requests.
- Each site surrenders part of its autonomy in terms of right to change schemas or software
- Database is accessed through a single interface as if it is a single database.

Two types of homogeneous distributed database −

Autonomous − Each database is independent that functions on its own. They are integrated by a controlling application and use message passing to share data updates.

Non-autonomous − Data is distributed across the homogeneous nodes and a **central or master DBMS** co-ordinates data updates across the sites.

Heterogeneous

Uses different schemas, operating systems, DDBMS, and different data models.

A particular site can be completely unaware of other sites.

- Different sites may use different schemas and software
- Composed of a variety of DBMSs like relational, network, hierarchical or object oriented.
- Difference in schema is a major problem for query processing
- A site may not be aware of other sites - so there is limited co-operation in processing user requests.

Types of Heterogeneous Distributed Databases

Federated − The heterogeneous database systems are independent in nature and integrated together so that they function as a single database system.

Un-federated − The database systems employ a central coordinating module through which the databases are accessed.

Date's Rules for distributed database systems

To the user, a distributed system should look exactly like a non-distributed system

Types of Distributed Databases

Homogeneous

1. Share a common **global schema.**
2. Run **identical DBMS** software.
3. Each site provides part of its autonomy in terms of right to change schema or s/w.
4. Same S/W, **no issues** in transaction processing.
5. Same Schema, **No issues** in Query processing.

Heterogeneous

1. Different sites can have **different schema.**
2. Run **different DBMS** software.
3. Each site maintains its own right to change the schema or s/w.
4. Different S/W – **Major problem** in transaction processing.
5. Different Schema - Problem in **query processing**.

Distributed Data Storage

Approaches for distributed data storage:

1. Fragmentation

2. Replication

A table is divided into two or more pieces referred to as **fragments or partitions.**

Each fragment can be stored at different sites.

All data stored in a table is required at a given site.

Fragmentation should be done in a way so that the original table can be reconstructed from the fragments.

Advantages:
Fragmentation increases parallelism
Provides better disaster recovery.
Only one copy of each fragment in the system, i.e. no redundant data.

There are three types of fragmentation

1. Horizontal Fragmentation (Selection)
2. Vertical Fragmentation (Projection)
3. Hybrid

Vertical Fragmentation

The fields or columns of a table are grouped into fragments.

To maintain reconstructiveness, each fragment should contain the primary key field(s) of the table.

Horizontal Fragmentation

Tuples of a table are grouped in accordance to values of one or more fields.

Should also confirm to the rule of reconstructiveness.

Each horizontal fragment **must have all columns** of the original base table.

- **Horizontal fragmentation**
  - refers to cut between tuples
  - achieved by selection
  - same as data partitioning in parallel databases
  - efficient for parallel scanning of the relation

- **Vertical fragmentation**
  - refers to the cut of the schema
  - achieved by a projection
  - efficient if there is frequent access to different attribute groups

Hybrid Fragmentation

In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques are used.

Hybrid fragmentation can be done in two alternative ways −

At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.

At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.

Replication

System maintains **multiple copies of data, stored in different sites**, for **faster retrieval** and **fault tolerance**.

Full replication of a relation is the case where the **relation is stored at all sites.**

Fully redundant databases are those in which every site contains a copy of the entire database.

Advantages of Replication

Availability: failure of site containing relation r does not result in unavailability of r if replicas exist.

Parallelism: queries on r may be processed by several nodes in parallel.
Reduced data transfer: relation r is available locally at each site containing a replica of r.

Disadvantages of Replication
Increased cost of updates: each replica of relation r must be updated.

Increased complexity of concurrency control:

Data Transparency

Data transparency: Degree to which system user may remain unaware of the details of **how and where** the data items are stored in a distributed system.

Following are three types of transparency:

- Fragmentation transparency
- Replication transparency
- Location transparency

Location transparency: Users should not have to know **where data is physically stored** but rather should be able to behave – at least from a logical standpoint – as if the data was all stored at their own local site.

Fragmentation independence (Transparency): Enables users to query upon any table as if it were unfragmented. Thus, it hides the fact that the table the user is querying on is actually a fragment or union of some fragments.

Replication Transparency: Ensures that replication of databases are hidden from the users. It enables users to query upon a table as if only a single copy of the table exists.