Distributed Databases

Unit-I

INTRODUCTION TO DISTRIBUTED DATABASE

INTRODUCTION TO DISTRIBUTED DATABASE

Introduction of Distributed Databases

Features of Distributed Databases

Distributed databases versus Centralized Databases

Principles-- Levels Of Distribution-Transparency

Reference Architecture

Types of Data Fragmentation

Integrity Constraints in Distributed Databases

Architectural Issues

Alternative Client/Server Architecture

What is a Distributed Database System?

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

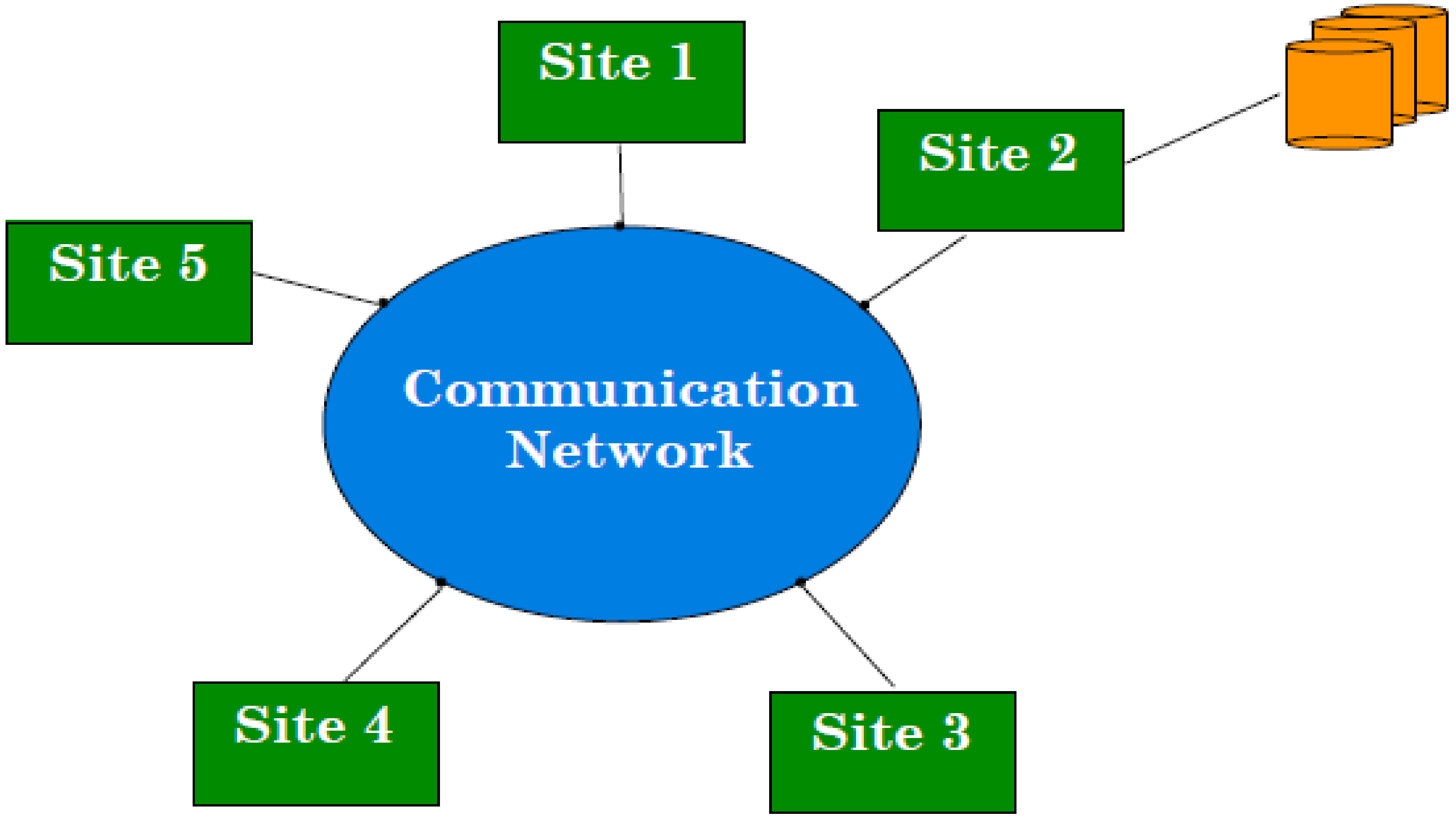
- A distributed database management system (D-DBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users.

- Distributed database system (DDBS) = DDB + D–DBMS

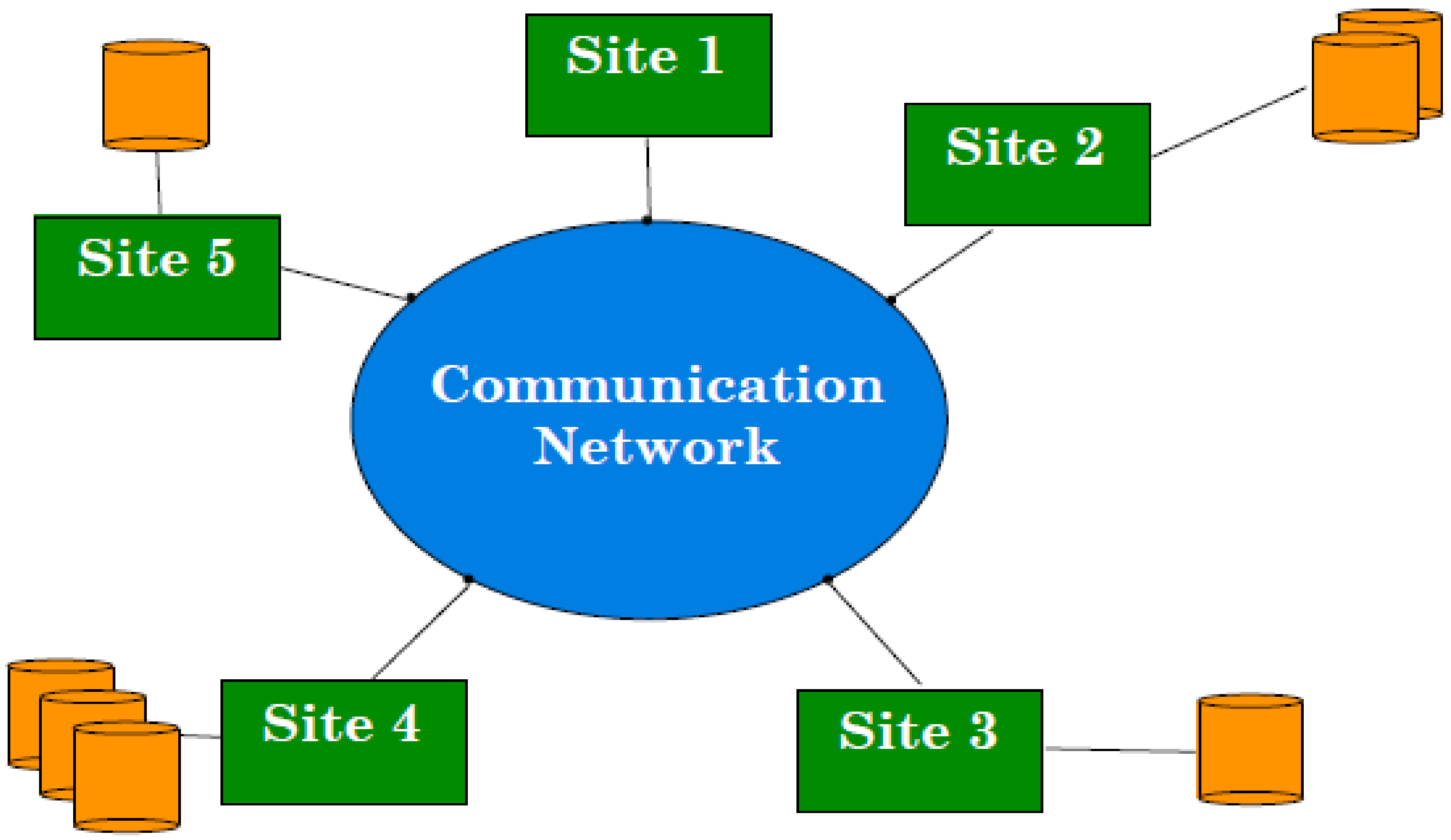
Distributed Databases

Reality (e.g., WWW, Grids, Cloud, Sensors, Mobiles, …)

Centralized DBMS on Network



Distributed DBMS Environment



Features of Distributed Databases

- Databases in the collection are logically interrelated with each other. Often they represent a single logical database.
- Data is physically stored across multiple sites. Data in each site can be managed by a DBMS independent of the other sites.
- The processors in the sites are connected via a network. They do not have any multiprocessor configuration.
- A distributed database is not a loosely connected file system.
- A distributed database incorporates transaction processing, but it is not synonymous with a transaction processing system.

Features of Distributed Databases

- Data stored at a number of sites ⇨ each site *logically* consists of a single processor.
- Processors at different sites are interconnected by a computer network ⇨ no multiprocessors
  - parallel database systems
- Distributed database is a database, not a collection of files ⇨ data logically related as exhibited in the users' access patterns
  - relational data model
- D-DBMS is a full-fledged DBMS
  - not remote file system, not a TP system

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.

Why Distributed Databases

- Organizational and economic reasons
- Interconnection of existing databases
- Incremental growth
- Reduced communication overhead
- Performance considerations
- Reliability and availability

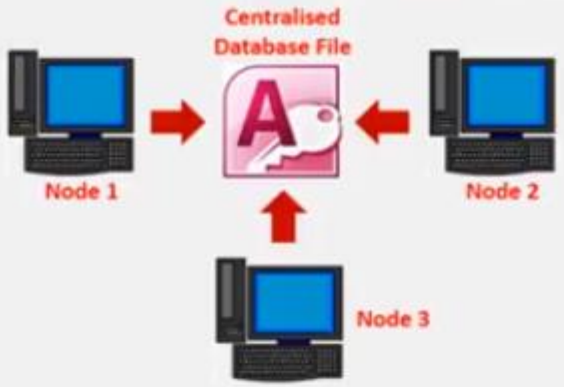
Difficult of Distributed Databases

- **Need for complex and expensive software** − DDBMS demands complex and often expensive software to provide data transparency and co-ordination across the several sites.
- **Processing overhead** − Even simple operations may require a large number of communications and additional calculations to provide uniformity in data across the sites.
- **Data integrity** − The need for updating data in multiple sites pose problems of data integrity.
- **Overheads for improper data distribution** − Responsiveness of queries is largely dependent upon proper data distribution. Improper data distribution often leads to very slow response to user requests.

Distributed databases versus Centralized Databases

Centralised Databases

A **single** database located at **1 site** on a network



Advantages:

Since there is only 1 database file, it is easier to:

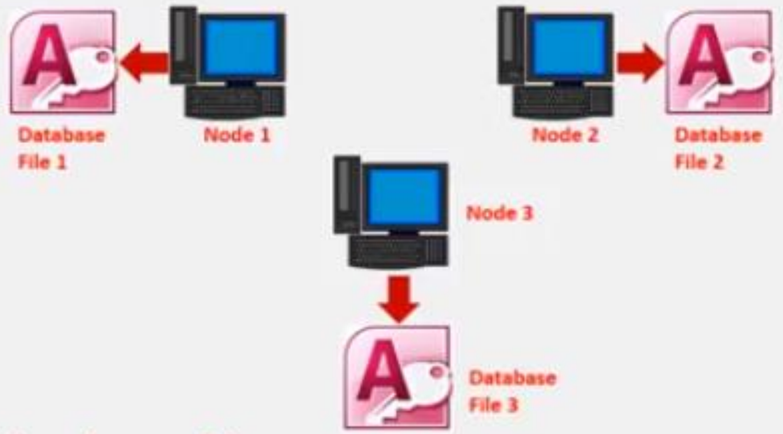
- Get a complete view of Data
- Manage, update an backup Data

Disadvantages:

- Bottle necking from multiple users accessing the same file – slowing down productivity

Distributed Databases

Consists of **2 or more files** located at different sites on a network



Advantages:

Having multiple database files means:

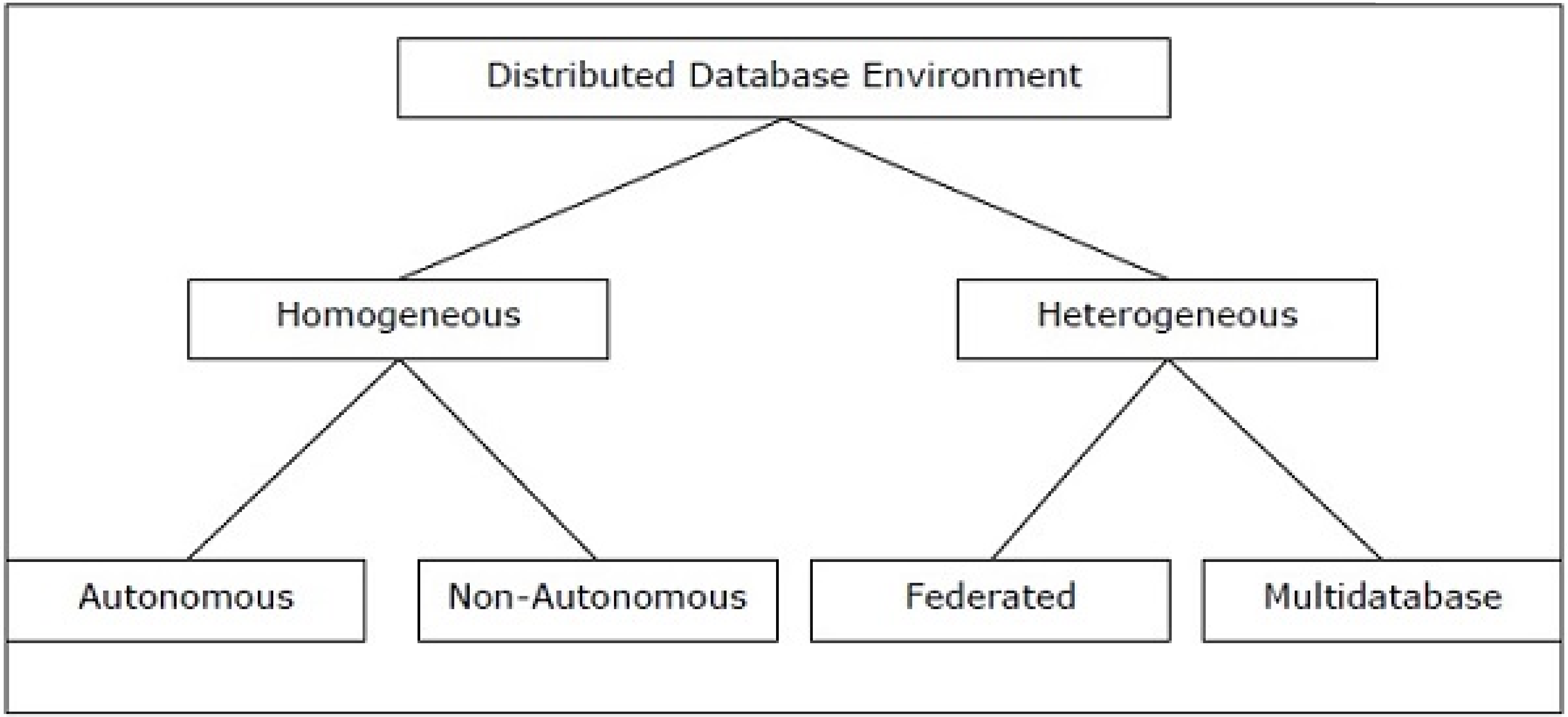
- Users wont interfere with each other when accessing / manipulating Data
- Speed since files are retrieved from nearest location
- If one site fails, the system can still run

Disadvantages:

- Time for Synchronisation of the multiple databases
- Data Replication for each different database file

Distributed database	Centralized database
Consists of multiple database files located at different sites	Consists of a single, central database file
Allows multiple users to access and manipulate data	Bottlenecks when multiple users access same file simultaneously
Files delivered quickly from location nearest the user	Files may take longer to deliver to users
If one site fails, data is retrievable	Single site means downtime in cases of system failures
Multiple files from dispersed databases must be synchronized	Simpler to update and manage data in single, central system

Types of Distributed Databases



Homogeneous Distributed Databases

- In a homogeneous distributed database, all the sites use identical DBMS and operating systems.

Its properties are –

- The sites use very similar software.
- The sites use identical DBMS or DBMS from the same vendor.
- Each site is aware of all other sites and cooperates with other sites to process user requests.
- The database is accessed through a single interface as if it is a single database.

Types of Homogeneous Distributed Database

- There are 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 Distributed Databases

- In a heterogeneous distributed database, different sites have different operating systems, DBMS products and data models.

Its properties are –

- Different sites use dissimilar schemas and software.
- The system may be composed of a variety of DBMSs like relational, network, hierarchical or object oriented.
- Query processing is complex due to dissimilar schemas.
- Transaction processing is complex due to dissimilar software.
- A site may not be aware of other sites and 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.

Transparency

- Transparency is the separation of the higher level semantics of a system from the lower level implementation issues.
- Fundamental issue is to provide

  data independence

  in the distributed environment

  - Network (distribution) transparency
  - Replication transparency
  - Fragmentation transparency
    - horizontal fragmentation: selection
    - vertical fragmentation: projection
    - hybrid

Distributed Database Transparency Features

- Distribution transparency
- Transaction transparency
- Failure transparency
- Performance transparency
- Heterogeneity transparency

Distributed Database Transparency Features

- Allow end user to feel like database's only user. User feels like they are working with a centralized database
- Features include:
  - *Distribution transparency* – user does not know where data is located and if replicated or partitioned
  - *Transaction transparency* – transaction can update at several network sites to ensure data integrity

Distributed Database Transparency Features

- *Failure transparency* – system continues to operate in the event of a node failure (other nodes pick up lost functionality)
- *Performance transparency* – allows system to perform as if it were a centralized DBMS. No performance degradation due to use of a network or platform differences
- *Heterogeneity transparency* – allows the integration of several different local DBMSs under a common schema

Distribution Transparency

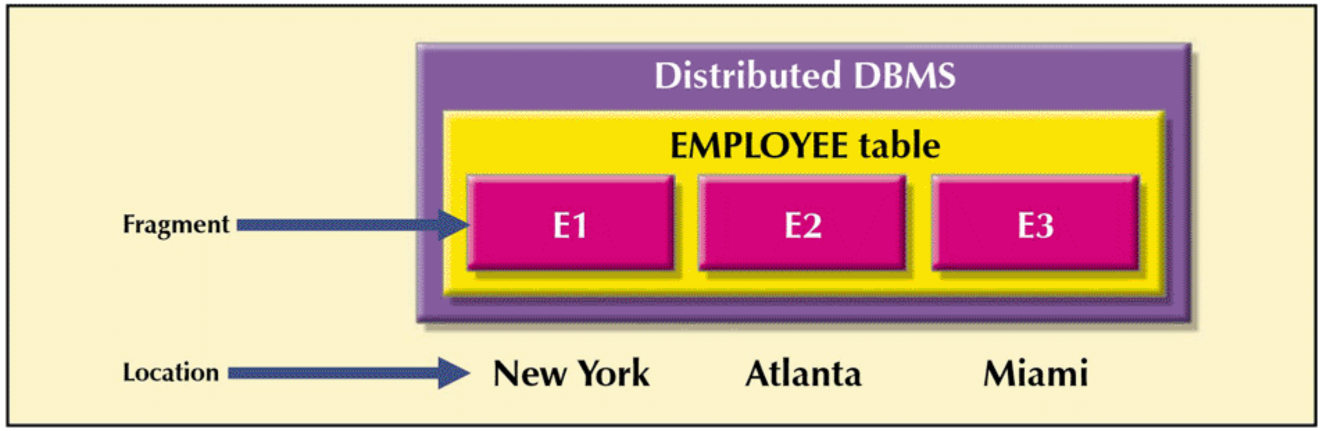
- Allows management of a physically dispersed database as though it were a centralized database
- Supported by a distributed data dictionary (DDD) which contains the description of the entire database as seen by the DBA
  - The DDD is itself distributed and replicated at the network nodes
- Three levels of distribution transparency are recognized:
  - *Fragmentation transparency* – user does not need to know if a database is partitioned; fragment names and/or fragment locations are not needed
  - *Location transparency* – fragment name, but not location, is required
  - *Local mapping transparency* – user must specify fragment name and location

A Summary of Transparency Features

TABLE 10.2 A Summary of Transparency Features

IF THE SQL STATEMENT REQUIRES:			
FRAGMENT NAME?	LOCATION NAME?	THEN THE DBMS SUPPORTS	LEVEL OF DISTRIBUTION TRANSPARENCY
Yes	Yes	Local mapping	Low
Yes	No	Location transparency	Medium
No	No	Fragmentation transparency	High

FIGURE 10.9 Fragment Locations



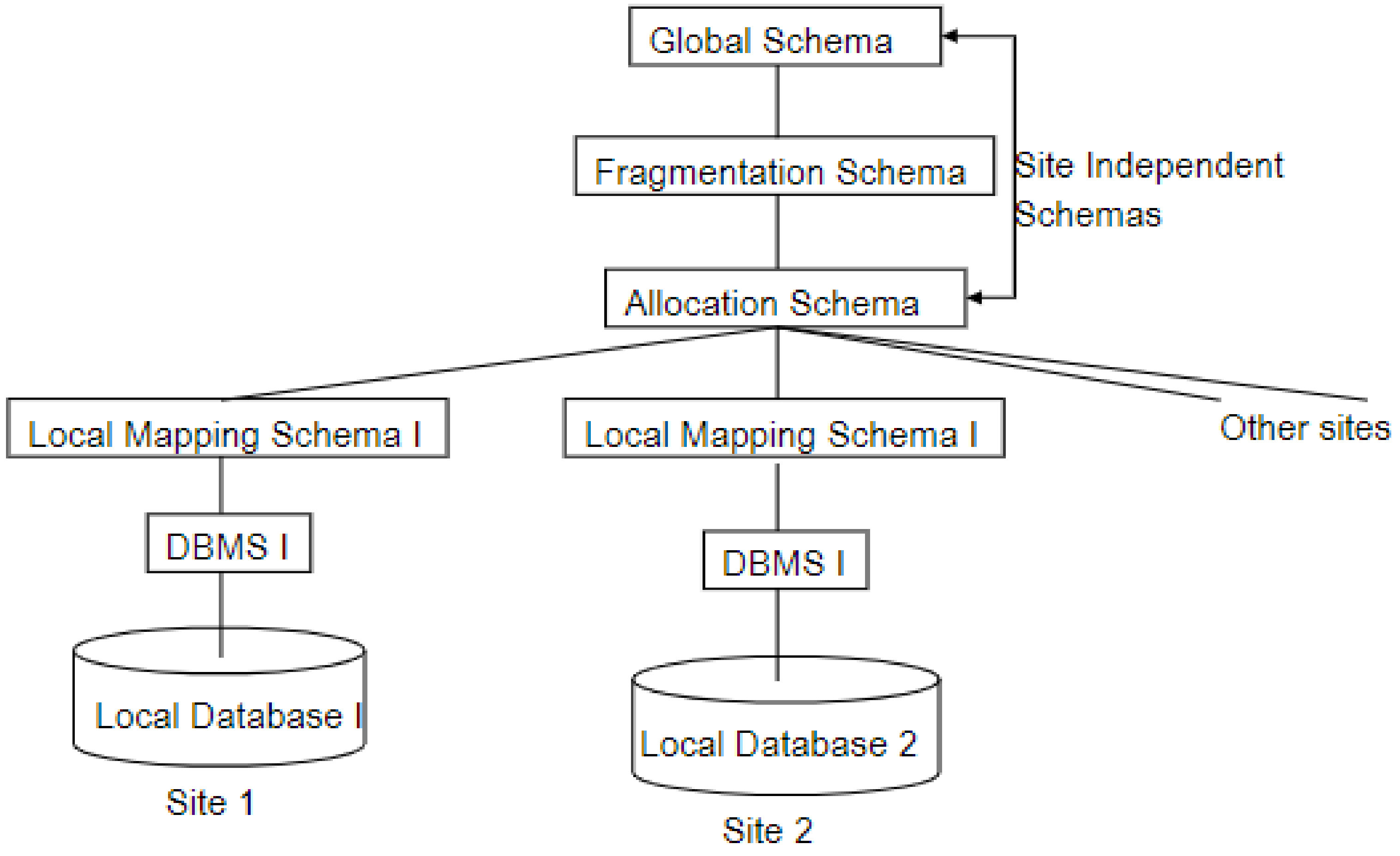
Distribution Transparency

- The EMPLOYEE table is divided among three locations (no replication)
- Suppose an employee wants to find all employees with a birthdate prior to jan 1, 1940
  - Fragmentation transparency-
    - SELECT * FROM EMPLOYEE WHERE EMP_DOB < ’01-JAN-1940’;
  - Location transparency-
    - SELECT * FROM E1 WHERE EMP_DOB < ’01-JAN-1940’ UNION SELECT * FROM E2 … UNION SELECT * FROM E3…;
  - Local Mapping Transparency
    - SELECT * FROM E1 NODE NY WHERE EMP_DOB < ’01-JAN-1940’ UNION SELECT * FROM E2 NODE ATL … UNION SELECT * FROM E3 NODE MIA…;

Transaction Transparency

- Ensures database transactions will maintain distributed database’s integrity and consistency
- A DDBMS transaction can update data stored in many different computers connected in a network
  - Transaction transparency ensures that the transaction will be completed only if all database sites involved in the transaction complete their part of the transaction

Reference Architecture for Distributed Database



Reference Architecture for Distributed Database

Global Schema: a set of global relations as if database were not distributed at all

Fragmentation Schema: global relation is split into “non-overlapping” (logical) fragments. 1:n mapping from relation R to fragments R_i.

Allocation Schema: 1:1 or 1:n (redundant) mapping from fragments to sites. All fragments corresponding to the same relation R at a site j constitute the physical image R^j. A copy of a fragment is denoted by R^j_i.

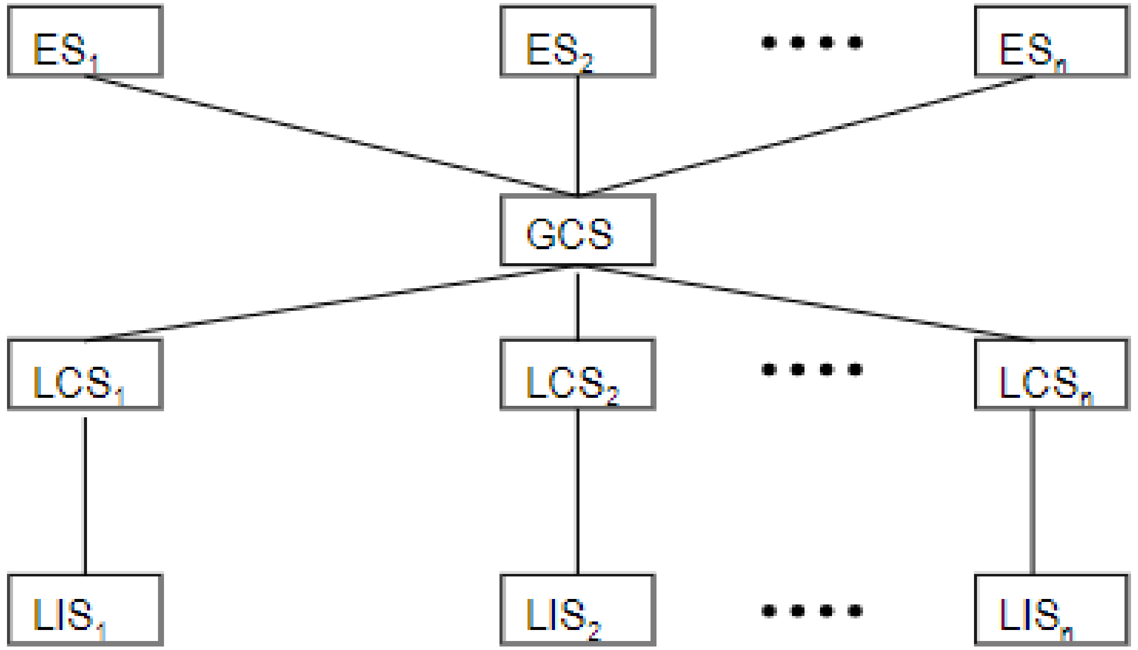
Local Mapping Schema: a mapping from physical images to physical objects, which are manipulated by local DBMSs.

Reference Architecture for Distributed Database

- Global schema defines all the data Which are contained in the distributed database as if the database were not distributed at all.

  The Global schema consists of the definition of a set of global relations. Each global relation can be split into several non overlapping portions which are called fragments. The mapping between global relations and fragments is defined in the fragmentation schema. This mapping is one to many.
- Fragments are logical portions of global relations which are physically located at one or several sites of the network. The allocation schema defines at which site(s) a fragment is located.
- The Local mapping schema maps fragments in the allocation schema onto external objects in the local database.

Distributed Database Reference Architecture



Fragmentation

- Fragmentation is the task of dividing a table into a set of smaller The subsets of the table are called **fragments**.
- Fragmentation can be of three types:
  1. Horizontal
  2. vertical,
  3. hybrid (combination of horizontal and vertical).
- Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called “reconstructiveness.”

Advantages of Fragmentation

- Since data is stored close to the site of usage, efficiency of the database system is increased.
- Local query optimization techniques are sufficient for most queries since data is locally available.
- Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Disadvantages of Fragmentation

- When data from different fragments are required, the access speeds may be very high.
- In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
- Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

Vertical Fragmentation

- In vertical fragmentation, the fields or columns of a table are grouped into fragments.
- In order to maintain reconstructiveness, each fragment should contain the primary key field(s) of the table.
- Vertical fragmentation can be used to enforce privacy of data.

- For example, let us consider that a University database keeps records of all registered students in a Student table having the following schema

Regd_No	Name	Course	Address	Semester	Fees	Marks

- Now, the fees details are maintained in the accounts section. In this case, the designer will fragment the database as follows –

- CREATE TABLE STD_FEES AS SELECT Regd_No, Fees FROM STUDENT;

Horizontal fragmentation

- Horizontal fragmentation groups the tuples of a table in accordance to values of one or more fields. Horizontal fragmentation should also confirm to the rule of reconstructiveness. Each horizontal fragment must have all columns of the original base table.
- Horizontal fragmentation can further be classified into two techniques: primary horizontal fragmentation and derived horizontal fragmentation.
- For example, in the student schema, if the details of all students of Computer Science Course needs to be maintained at the School of Computer Science, then the designer will horizontally fragment the database as follows −
- CREATE COMP_STD AS SELECT * FROM STUDENT WHERE COURSE = "Computer Science";

Hybrid Fragmentation

- In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques are used. This is the most flexible fragmentation technique since it generates fragments with minimal extraneous information. However, reconstruction of the original table is often an expensive task.
- Hybrid fragmentation can be done in two alternative ways −

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

Integrity Constraints in Distributed Database

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

The integrity constraints are as follows –

- Data type integrity constraint
- Entity integrity constraint
- Referential integrity constraint

Data Type Integrity Constraint

- A data type constraint restricts the range of values and the type of operations that can be applied to the field with the specified data type.
- For example, let us consider that a table "HOSTEL" has three fields - the hostel number, hostel name and capacity. The hostel number should start with capital letter "H" and cannot be NULL, and the capacity should not be more than 150. The following SQL command can be used for data definition −

```
CREATE TABLE HOSTEL ( H_NO VARCHAR2(5) NOT NULL, H_NAME VARCHAR2(15), CAPACITY INTEGER, CHECK ( H_NO LIKE 'H%'), CHECK ( CAPACITY <= 150) );
```

Entity Integrity Control

- Entity integrity control enforces the rules so that each tuple can be uniquely identified from other tuples. For this a primary key is defined. A primary key is a set of minimal fields that can uniquely identify a tuple. Entity integrity constraint states that no two tuples in a table can have identical values for primary keys and that no field which is a part of the primary key can have NULL value.
- For example, in the above hostel table, the hostel number can be assigned as the primary key through the following SQL statement (ignoring the checks) −

```
CREATE TABLE HOSTEL ( H_NO VARCHAR2(5) PRIMARY KEY, H_NAME
  VARCHAR2(15), CAPACITY INTEGER );
```

Referential integrity

- Referential integrity constraint lays down the rules of foreign keys. A foreign key is a field in a data table that is the primary key of a related table. The referential integrity constraint lays down the rule that the value of the foreign key field should either be among the values of the primary key of the referenced table or be entirely NULL.
- For example, let us consider a student table where a student may opt to live in a hostel. To include this, the primary key of hostel table should be included as a foreign key in the student table. The following SQL statement incorporates this
- CREATE TABLE STUDENT (S_ROLL INTEGER PRIMARY KEY, S_NAME VARCHAR2(25) NOT NULL, S_COURSE VARCHAR2(10), S_HOSTEL VARCHAR2(5) REFERENCES HOSTEL);

Architectural Issues

1. Heterogeneity

The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Internet consists of many different sorts of network their differences are masked by the fact that all of the computers attached to them use the Internet protocols to communicate with one another. For eg., a computer attached to an Ethernet has an implementation of the Internet protocols over the Ethernet, whereas a computer on a different sort of network will need an implementation of the Internet protocols for that network.

2. Openness

The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways. The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.

3. Security

Many of the information resources that are made available and maintained in distributed systems have a high intrinsic value to their users. Their security is therefore of considerable importance. Security for information resources has three components: confidentiality, integrity, and availability.

4. Scalability

Distributed systems operate effectively and efficiently at many different scales, ranging from a small intranet to the Internet. A system is described as scalable if it will remain effective when there is a significant increase in the number of resources and the number of users.

5. Failure handling

Computer systems sometimes fail. When faults occur in hardware or software, programs may produce incorrect results or may stop before they have completed the intended computation. Failures in a distributed system are partial – that is, some components fail while others continue to function. Therefore the handling of failures is particularly difficult.

6. Concurrency

Both services and applications provide resources that can be shared by clients in a distributed system. There is therefore a possibility that several clients will attempt to access a shared resource at the same time. Object that represents a shared resource in a distributed system must be responsible for ensuring that it operates correctly in a concurrent environment. This applies not only to servers but also to objects in applications. Therefore any programmer who takes an implementation of an object that was not intended for use in a distributed system must do whatever is necessary to make it safe in a concurrent environment.

7. Transparency

Transparency can be achieved at two different levels. Easiest to do is to hide the distribution from the users. The concept of transparency can be applied to several aspects of a distributed system.

a) **Location transparency:** The users cannot tell where resources are located

b) **Migration transparency:** Resources can move at will without changing their names

c) **Replication transparency:** The users cannot tell how many copies exist.

d) **Concurrency transparency:** Multiple users can share resources automatically.

e) **Parallelism transparency:** Activities can happen in parallel without users knowing.

8. Quality of service

Once users are provided with the functionality that they require of a service, such as the file service in a distributed system, we can go on to ask about the quality of the service provided. The main nonfunctional properties of systems that affect the quality of the service experienced by clients and users are reliability, security and performance. Adaptability to meet changing system configurations and resource availability has been recognized as a further important aspect of service quality.

9. Reliability

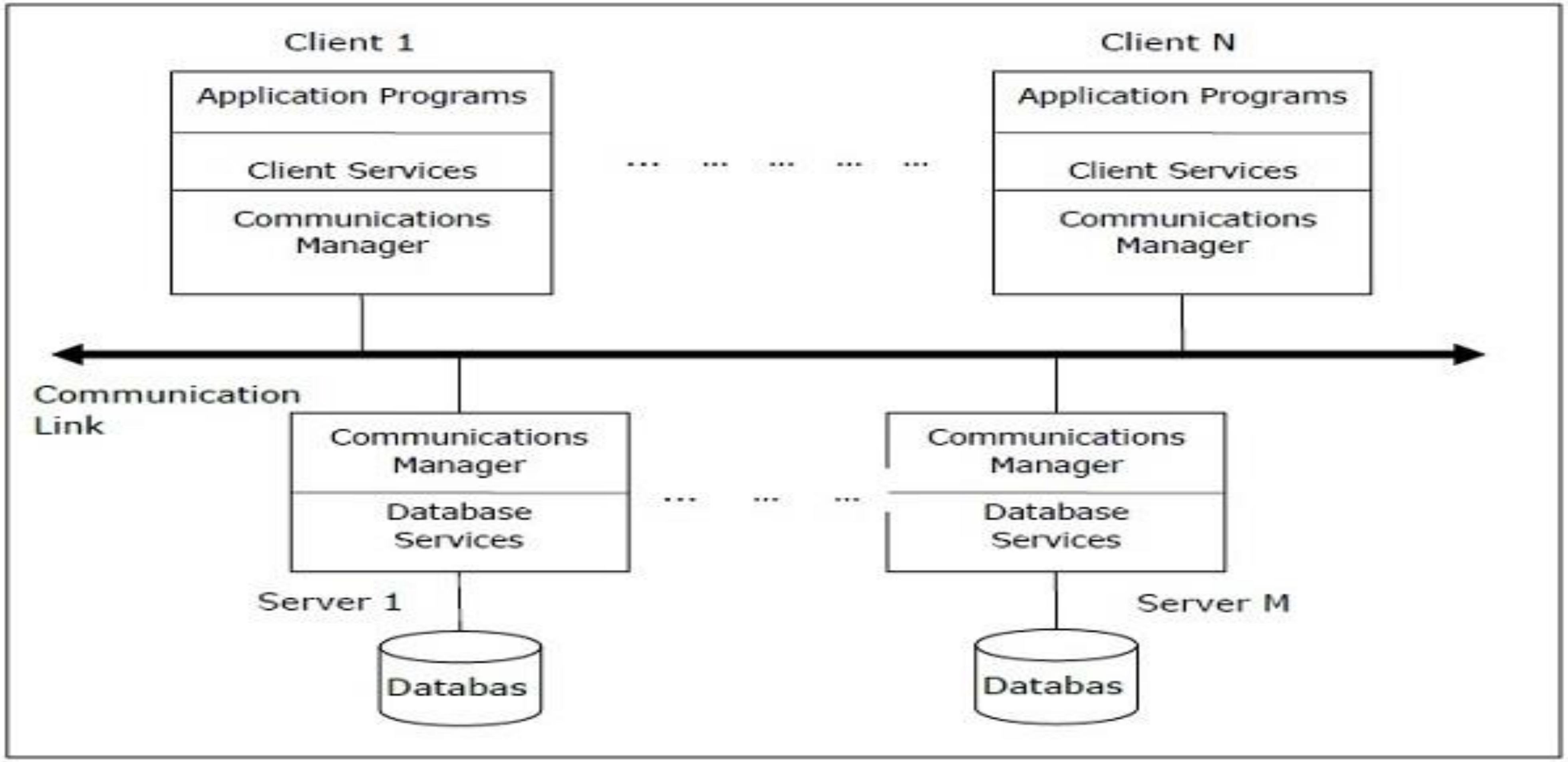
One of the original goals of building distributed systems was to make them more reliable than single-processor systems. The idea is that if a machine goes down, some other machine takes over the job. A highly reliable system must be highly available, but that is not enough. Data entrusted to the system must not be lost or garbled in any way, and if files are stored redundantly on multiple servers, all the copies must be kept consistent. In general, the more copies that are kept, the better the availability, but the greater the chance that they will be inconsistent, especially if updates are frequent.

10. Performance

Always the hidden data in the background is the issue of performance. Building a transparent, flexible, reliable distributed system, more important lies in its performance. In particular, when running a particular application on a distributed system, it should not be appreciably worse than running the same application on a single processor. Unfortunately, achieving this is easier said than done.

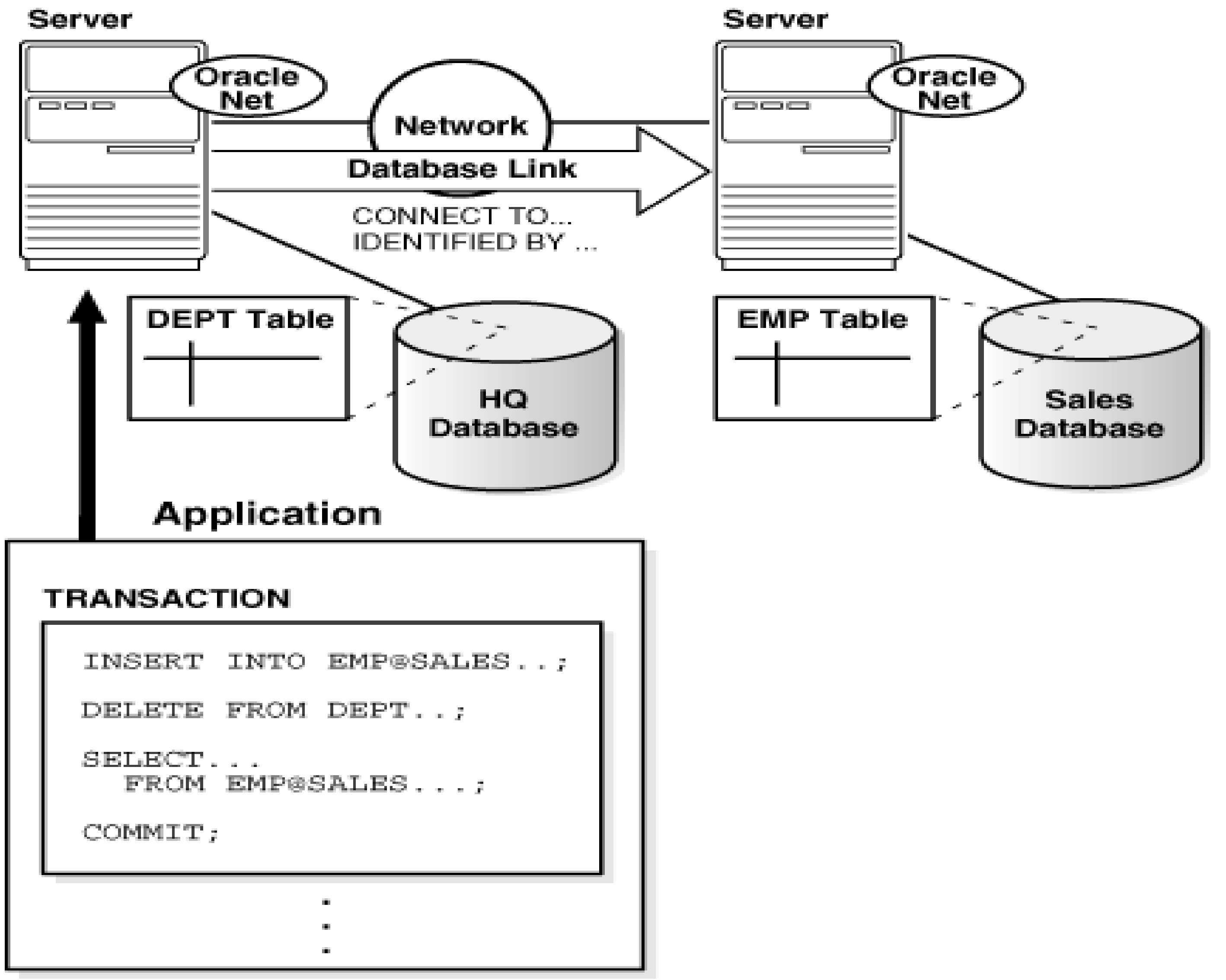
Client - Server Architecture for DDBMS

- This is a two-level architecture where the functionality is divided into servers and clients. The server functions primarily encompass data management, query processing, optimization and transaction management. Client functions include mainly user interface. However, they have some functions like consistency checking and transaction management.
- The two different client - server architecture are –
  - ✓Single Server Multiple Client
  - ✓Multiple Server Multiple Client


Client 1
Application Programs
Client Services
Communications Manager
Client N
Application Programs
Client Services
Communications Manager
Communication Link
Communications Manager
Database Services
Server 1
Databas
Communications Manager
Database Services
Server M
Databas

Case study

- A database server is the Oracle software managing a database, and a client is an application that requests information from a server. Each computer in a network is a node that can host one or more databases. Each node in a distributed database system can act as a client, a server, or both, depending on the situation.
- In the figure, the host for the hq database is acting as a database server when a statement is issued against its local data (for example, the second statement in each transaction issues a statement against the local dept table), but is acting as a client when it issues a statement against remote data (for example, the first statement in each transaction is issued against the remote table emp in the sales database).



```
INSERT INTO EMP@SALES..;

DELETE FROM DEPT..;

SELECT...
  FROM EMP@SALES...;

COMMIT;
```

- A client can connect **directly** or **indirectly** to a database server. A direct connection occurs when a client connects to a server and accesses information from a database contained on that server. For example, if you connect to the hq database and access the dept table on this database as in the figure, you can issue the following:
- SELECT * FROM dept; This query is direct because you are not accessing an object on a remote database.
- In contrast, an indirect connection occurs when a client connects to a server and then accesses information contained in a database on a different server. For example, if you connect to the hq database but access the emp table on the remote sales database as in the figure, you can issue the following:
- SELECT * FROM emp@sales; This query is indirect because the object you are accessing is not on the database to which you are directly connecte