Topics Covered : Distributed Databases: Introduction to Distributed Database Systems, Distributed Database System Architecture;

Dr. V.C.V. Rao

Office @Pune University Campus, Pune, India

CDAC is a R&D Institute of Dept. of Electronics & Information Technology (DeitY) , Ministry of Communications & Information Technology (MCIT); Govt. of India

NIT-Warangal:

Dept. Of Comp. Sc & Engg.

July – Nov 2023

Course Name (CS5305) : Advanced DataBases

Course Name (CS5309) : Advanced DataBases

Advanced Data Bases (References & Source for Presentation

The Presentation is prepared based on Author's experience on various research projects on Distributed Computing – Distributed Data Bases (NoSQL) well as references given in this presentation.

Source :
Text Books, Research Articles, Web Sites as indicated in many slides and References of this presentation

1. M T Ozsu, Patrick Valduriez, Principles of Distributed Database Systems, Prentice Hall, 3rd Edition 2011

NIT-Warangal: Dept. Of Comp. Sc & Engg. July – Nov 2023

Course Name : Advanced databases

M.C.A 2nd Year, July 2023

Syllabus:

UNIT 1 : Distributed Databases: Introduction to Distributed Database Systems, Distributed Database System Architecture;

UNIT 2 : Top-Down Approach, Distributed Database Design Issues, Fragmentation, Allocation, Database Integration, Bottom-up approach, Schema Matching, Schema Integration, Schema Mapping;

UNIT 3: Data and Access Control, View Management, Data Security; Query processing problem, Objectives of Query processing, Complexity of Relational Algebra Operations, Characterization of Query Processors, Layers of Query Processing;

UNIT 4 : Query Decomposition, Normalization, Analysis, Elimination of Redundancy and Rewriting;

UNIT 5 : Localization of Distributed Data, Reduction for primary Horizontal, Vertical, derived Fragmentation;

UNIT 6 : Distributed Query Execution, Query Optimization, Join Ordering, Static& Dynamic Approach, Semi-joins, Hybrid Approach;

NIT-Warangal: Dept. Of Comp. Sc & Engg. July – Nov 2023

Course Name : Advanced databases

Syllabus:

M.C.A 2nd Year, July 2023

UNIT 7 : Taxonomy of Concurrency control Mechanisms, Lock-Based Concurrency Control, Timestamp-Based Concurrency Control, Optimistic Concurrency Control, Deadlock Management;

UNIT 8 : Heterogeneity issues Advanced Transaction Models, Distributed systems 2PC& 3PC protocols, Replication protocols, Replication and Failures, HotSpares;

UNIT 9: Parallel Databases: Introduction to Parallel Databases, Parallel Database System Architectures, Parallel Data Placement, Full Partitioning; Parallel Query Processing, Query Parallelism;

UNIT 10 : Parallel Query Optimization, Search Space, Cost Model, Search Strategy; Load Balancing.

Text Books/ Reference Books / Online Resources:

M T Ozsu, Patrick Valduriez, Principles of Distributed Database Systems, Prentice Hall, 1999.

S. Ceri and G. Pelaggati, Distributed Database System Principles and Systems, MGH, 1985.

Course Name : Advanced databases

Topics Covered in this Lecture

UNIT 1 : Distributed Databases: Introduction to Distributed Database Systems, Distributed Database System Architecture;

UNIT 2 : Top-Down Approach, Distributed Database Design Issues, Fragmentation, Allocation, Database Integration, Bottom-up approach, Schema Matching, Schema Integration, Schema Mapping;

Chapter-01 & Chapter-02 of Reference Book No. 1

Part-01

History - Database Management

Database systems have taken us from a paradigm of data processing in which each application defined and maintained its own data (Figure 1) to one in which the data are defined and administered centrally (Figure 2).

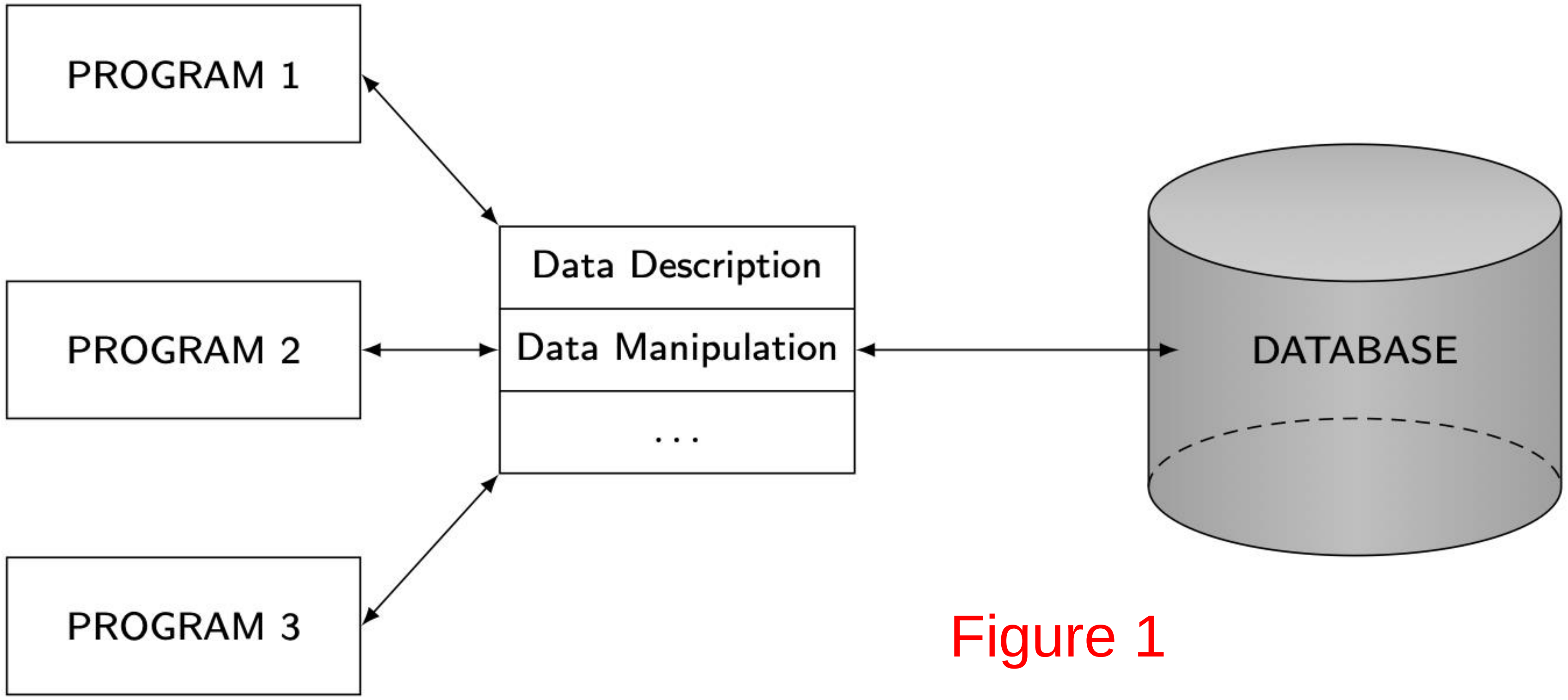


Figure 1

History - Database Management

Database System and Computer Network

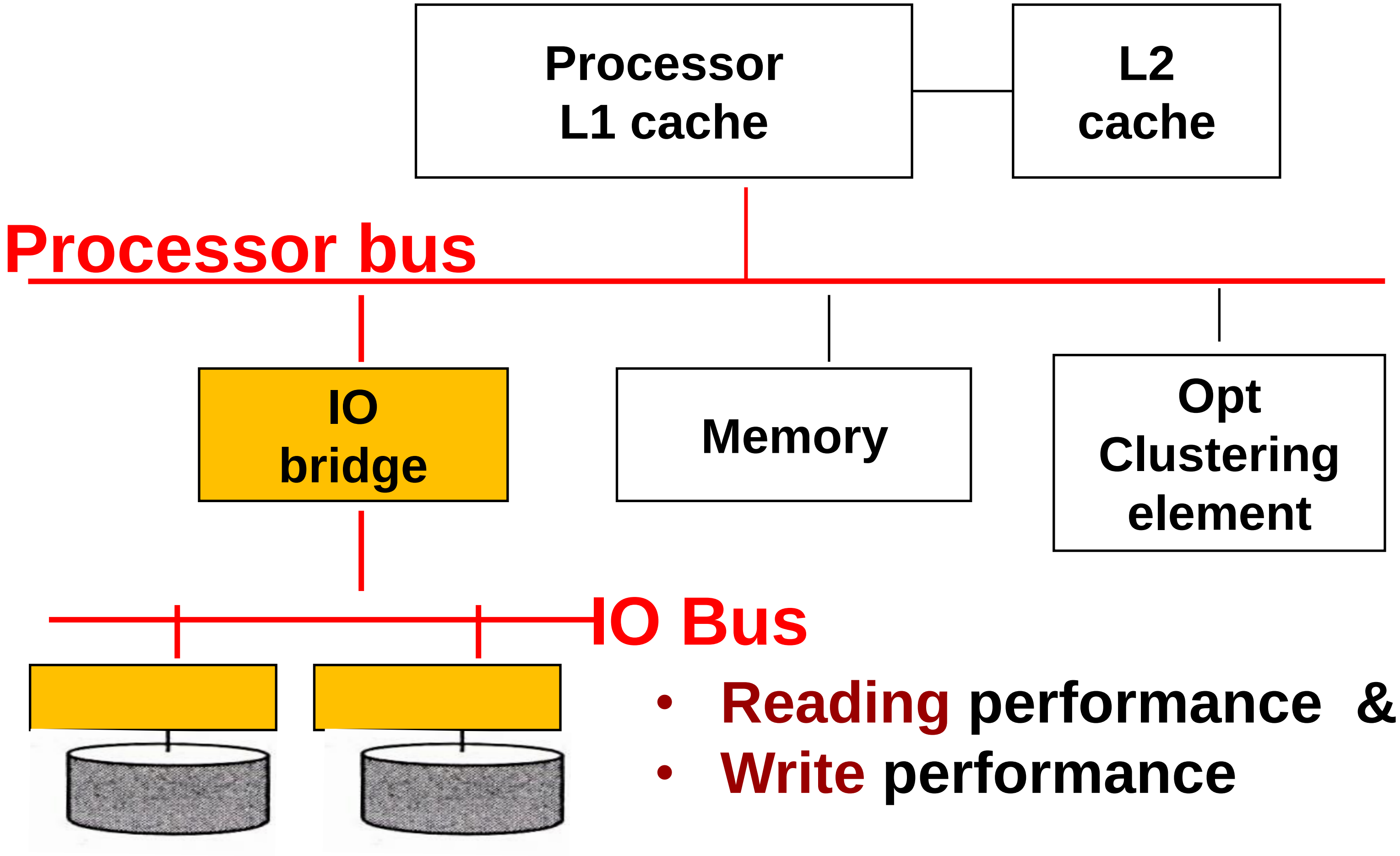
One of the major motivations behind the use of database systems is the desire to **integrate the operational data** of an enterprise and to provide centralized, thus controlled access to that data..

The technology of computer networks, on the other hand, promotes a mode of work that goes against all centralization efforts.

The key to this understanding is the realization that the most important **objective of the database technology** is integration, not centralization.

Centralised DataBase System : Shared Memory Sys

Centralized Database on Muti-Core System - I/O Bandwidth

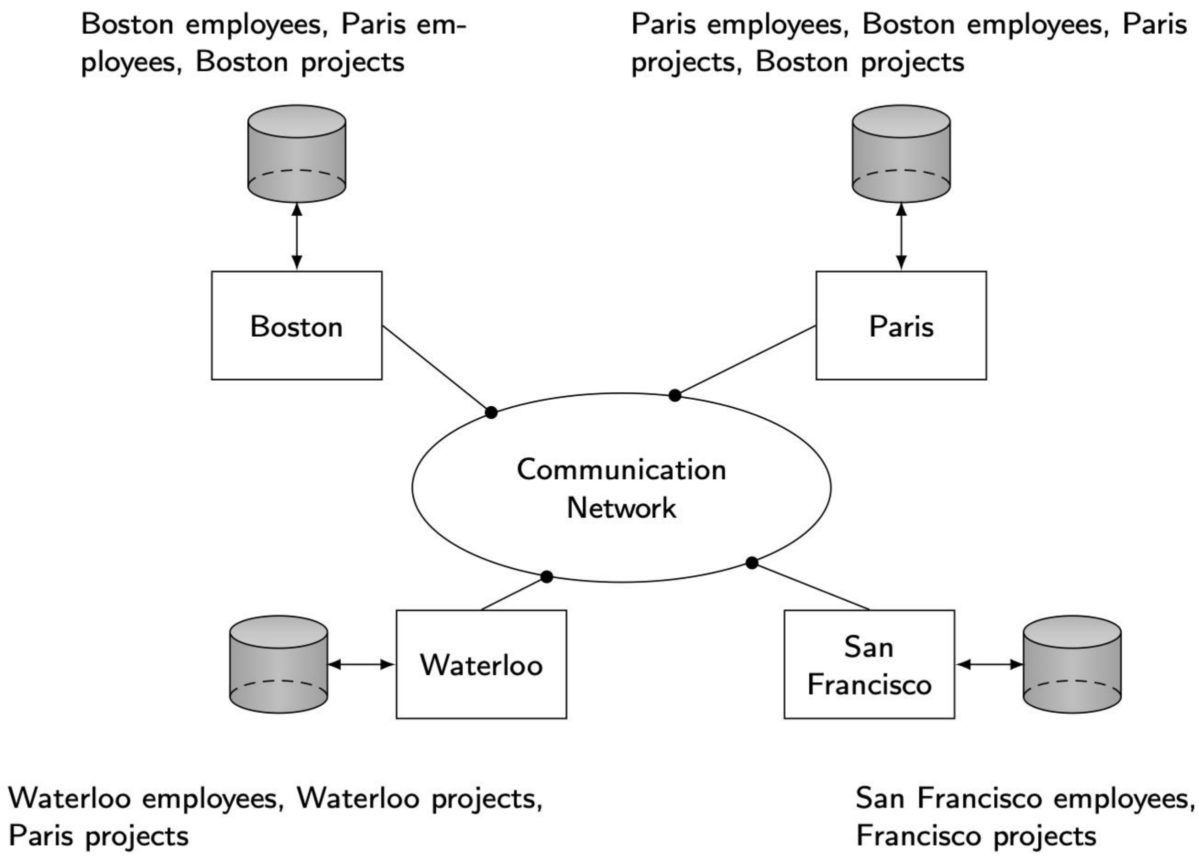


- Reading performance &
- Write performance

Source **:** References given in the presentation

History - Database Management

Peer-to-Peer (P2P)



Boston employees, Paris employees, Boston projects
Paris employees, Boston employees, Paris projects, Boston projects
Boston
Paris
Communication Network
Waterloo
San Francisco
Waterloo employees, Waterloo projects, Paris projects
San Francisco employees, San Francisco projects

Figure 2

History - Database Management

Database System and Computer Network

Define the fundamental concepts and set the framework for discussing distributed databases.

We start by examining distributed systems in general in order to clarify the role of database technology within distributed data processing, and then move on to topics that are more directly related to DDBS.

DDBS : Distributed Data Processing

1. Single-processor computers :The central processing unit (CPU) and input/ output (I/O) functions are separated and overlapped

   - This separation and overlap can be considered as one form of distributed processing.

2. define distributed processing in such a way that it leads to a definition of a distributed database system

DDBS : Distributed Data Processing

1. Distributed computing system Def : it is a number of autonomous processing elements (not necessarily homogeneous) that are interconnected by a computer network and that cooperate in performing their assigned tasks.

   The “processing element” referred to in this definition is a computing device that can execute a program on its own.

DDBS : Distributed Data Processing

A fundamental question that needs to be asked is:

What is being distributed?

- **Processing Log**ic or Processing Elements)
- **Function** (Various functions of a computer system could be delegated to various pieces of hardware or software
- **Data** : Distribution - processing sites
- **Control** can be distributed - Control of various tasks

- Distributed database systems should also be viewed within this framework and
- Treated as **tools** that could make distributed processing easier and more efficient

What is a Distributed Database System ?

- We **define a distributed database** as a collection of multiple, logically interrelated databases distributed over a computer network.

- **A distributed database management system** (**distributed DBMS**) is then defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users.

- Sometimes "**distributed database system**" (DDBS) is used to refer jointly to the **distributed database** and **the distributed DBMS**

What is a Distributed Database System ?

- The two important terms in these definitions are "**logically interrelated**" and "**distributed over a computer network**."

- They help eliminate certain cases that have sometimes been accepted to represent a DDBS.

Remark :

- A DDBS **is not** a "**collection of files**" that can be individually stored at each node of a computer network
- To form a DDBS, files should not only be logically related, but there should be structured among the files, and access should be via a common interface.

What is a Distributed Database System ?

- We **define a distributed database** as a collection of multiple, logically interrelated databases distributed over a computer network.

- **A distributed database management system** (**distributed DBMS**) is then defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users.

- Sometimes "**distributed database system**" (DDBS) is used to refer jointly to the **distributed database** and **the distributed DBMS**

What is a Distributed Database System ?

- ❖ It is important to make a distinction between a DDBS where this requirement is met, and more general distributed data management systems that provide a “DBMS-like” access to data.

- ❖ It has sometimes been assumed that the **physical distribution of data** is not the most significant issue. (Labelling as a distributed database). However, the **physical distribution of data is important**

- ❖ It creates problems that are not encountered when the databases reside in the same computer

What is a Distributed Database System ?

❖ Note that physical distribution implies that the communication between them is done over a network instead of through shared memory or shared disk (as would be the case with **multiprocessor systems**), with the network as the only shared resource.

❖ A **multiprocessor system design** is rather symmetrical, consisting of a number of identical processor and memory components, and controlled by one or more copies of the same operating system that is responsible for a strict control of the task assignment to each processor.

❖ This is not true in distributed computing systems, where heterogeneity of the operating system as well as the hardware is quite common.

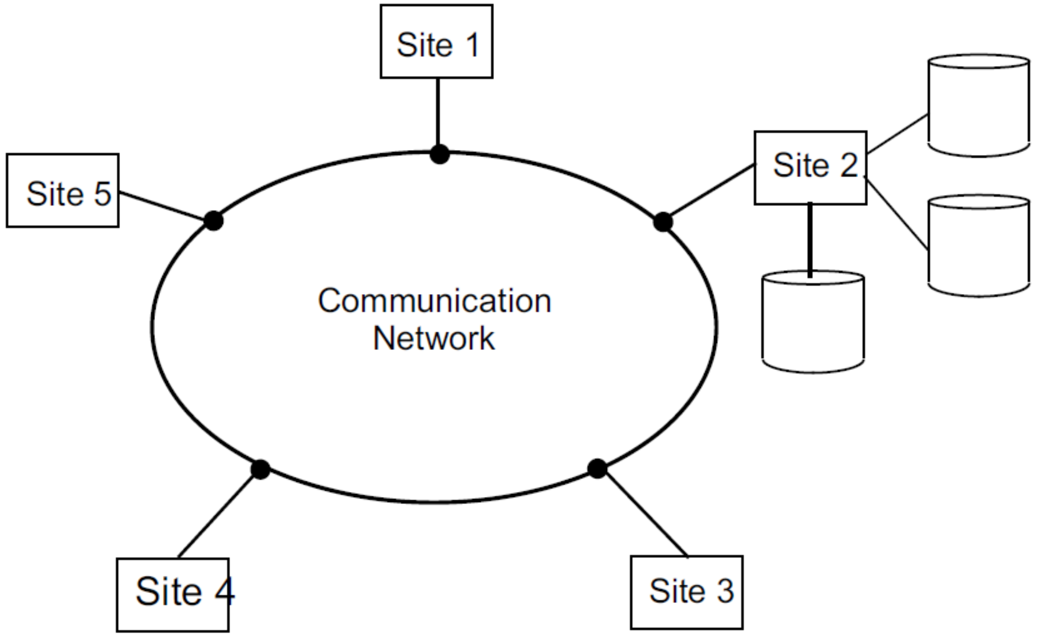
What is a Distributed Database System ?

- Note that physical distribution implies that the communication between them is done over a network instead of through shared memory or shared disk (as would be the case with **multiprocessor systems**), with the network as the only shared resource.

- A **multiprocessor system design** is rather symmetrical, consisting of a number of identical processor and memory components, and controlled by one or more copies of the same operating system that is responsible for a strict control of the task assignment to each processor.

DDBS : Distributed Data Processing

A **DDBS** is also not a system where, despite the existence of a network, the database resides at only one node of the network

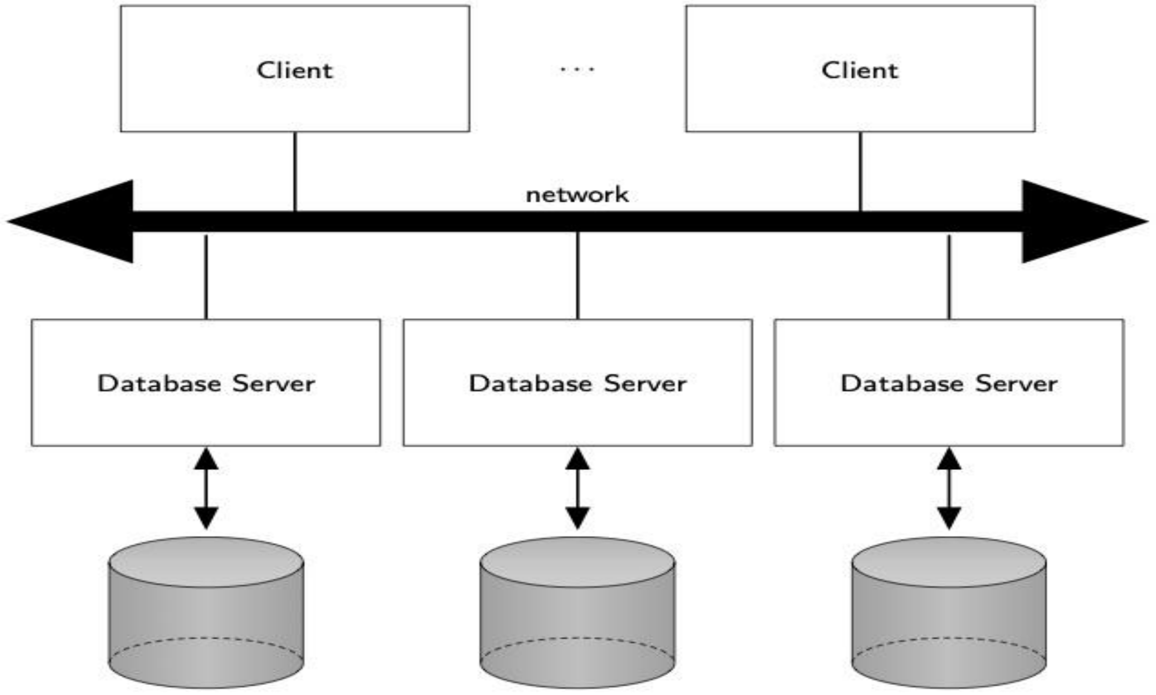


Central Database on a Network

In this case, the problems of database management are no different than the problems encountered in a centralized database environment **(Client -Server** DBMSs may relax this requirement)

DDBS : Distributed Data Processing

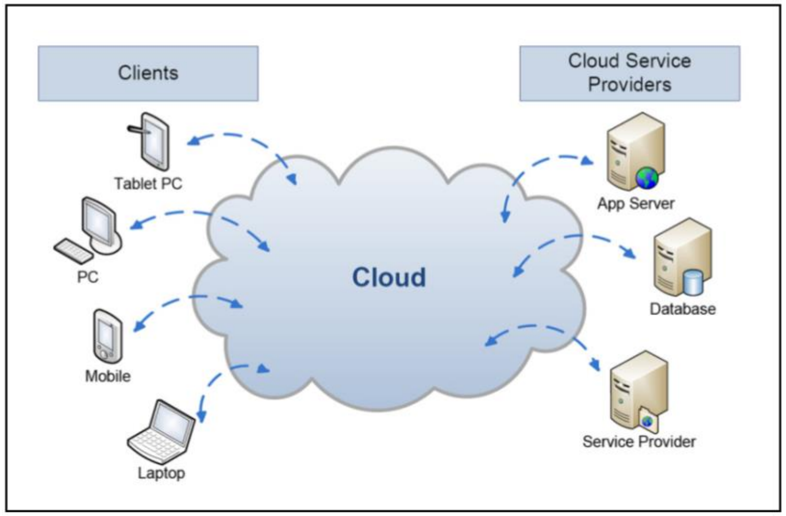
A **DDBS** is also not a system where, despite the existence of a network, the database resides at only one node of the network



Client Server

DDBS : Distributed Data Processing Cloud

A **DDBS** is also not a system where, despite the existence of a network, the database resides at only one node of the network

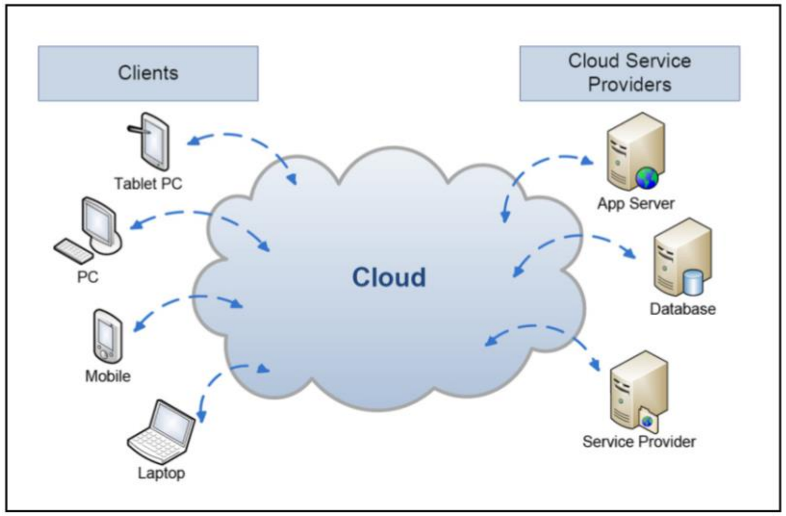


Cloud

DDBS : Distributed Data Processing

On-demand, reliable services provided over the Internet in a cost-efficient manner

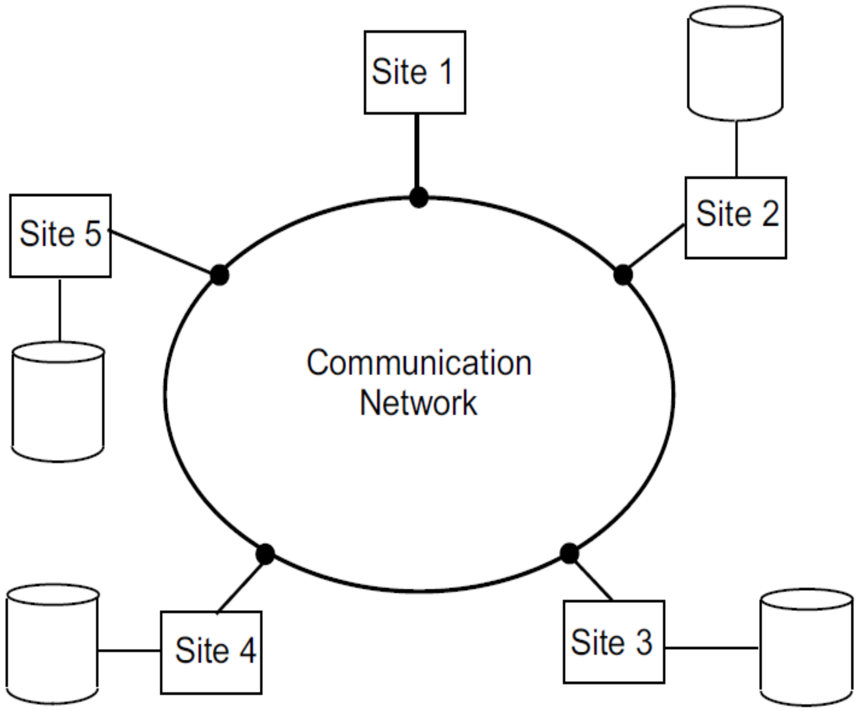
- Cost savings: no need to maintain dedicated compute power
- Elasticity: better adaptivity to changing workload



DDBS : Distributed Data Processing

The existence of a computer network or a collection of “files” is not sufficient to form a distributed database system.

Interest : Provide an environment where data are distributed among a number of sites



DDBS Environment

DDBS : Data Delivery Alternatives

❖ In distributed databases, data are "**delivered"** from the sites where they are stored to where the query is posed.

❖ Characterize the data delivery alternatives along three orthogonal dimensions:

- delivery modes,
- frequency and
- communication methods.

Remark : The combinations of alternatives along each of these dimensions (that we discuss next) provide a rich design space.

DDBS : Data Delivery Alternatives

❖ The alternative delivery modes are

- Pull-only modes,
- Push-only modes and
- Hybrid modes

DDBS : Data Delivery Alternatives : Pull-Only Mode

- ❖ The transfer of data from servers to clients is initiated by a client pull.

- ❖ When a client request is received at a server, the server responds by locating the requested information.

- ❖ **Main characteristics** : the arrival of new data items or updates to existing data items are carried out at a server without notification to clients unless clients explicitly poll the server.

DDBS : Data Delivery Alternatives : Pull-Only Mode

❖ In pull-based mode, servers must be interrupted continuously to deal with requests from clients.

❖ Furthermore, the information that clients can obtain from a server is limited to when and what clients know to ask for.

❖ Conventional DBMSs offer primarily pull-based data delivery.

DDBS : Data Delivery Alternatives :Push-Only Mode

- ❖ In the push-only mode of data delivery, the transfer of data from servers to clients is initiated by a server push in the absence of any specific request from clients.

- ❖ The main difficulty of the push-based approach is in deciding which data would be of common interest, and when to send them to clients – alternatives are
  - periodic,
  - irregular, or
  - conditional.

DDBS : Data Delivery Alternatives : Pull-Only Mode

❖ Thus, the usefulness of server push depends heavily upon the accuracy of a server to predict the needs of clients.

❖ In push-based mode, servers disseminate information to either an unbounded set of clients (random broadcast)

- who can listen to a medium or selective set of clients (multicast),
- who belong to some categories of recipients that may receive the data.

DDBS : Data Delivery Alternatives : Hybrid Mode Client Pull and server-push mechanisms-Only Mode

- ❖ The hybrid mode of data delivery combines the client-pull and server-push mechanisms.
- ❖ The continuous (or continual) query approach presents one possible way of combining the pull and push modes: namely,

  - the transfer of information from servers to clients is first initiated by a client pull (by posing the query), and
  - the subsequent transfer of updated information to clients is initiated by a server push.

DDBS : Data Delivery Alternatives : Hybrid Mode Client Pull and server-push mechanisms-Only Mode

❖ There are three typical frequency measurements that can be used to classify the regularity of data delivery. They are periodic,

- **Periodic**
- **Conditional**
- **ad-hoc or irregular**

Data Delivery Alternatives - Communication modes

❖ These methods determine the various ways in which servers and clients communicate for delivering information to clients. The alternatives are

- unicast
- One-many

Data Delivery Alternatives - Communication modes

- **In unicast**, the communication from a server to a client is one-to-one: the server sends data to one client using a particular delivery mode with some frequency.

- **In one-to-many**, as the name implies, the server sends data to a number of clients. Note that we are not referring here to a specific protocol; one-to-many communication may use a multicast or broadcast protocol.

Chapter-01

Part-02

Promises of DDBSs

1. transparent management of distributed and replicated data,

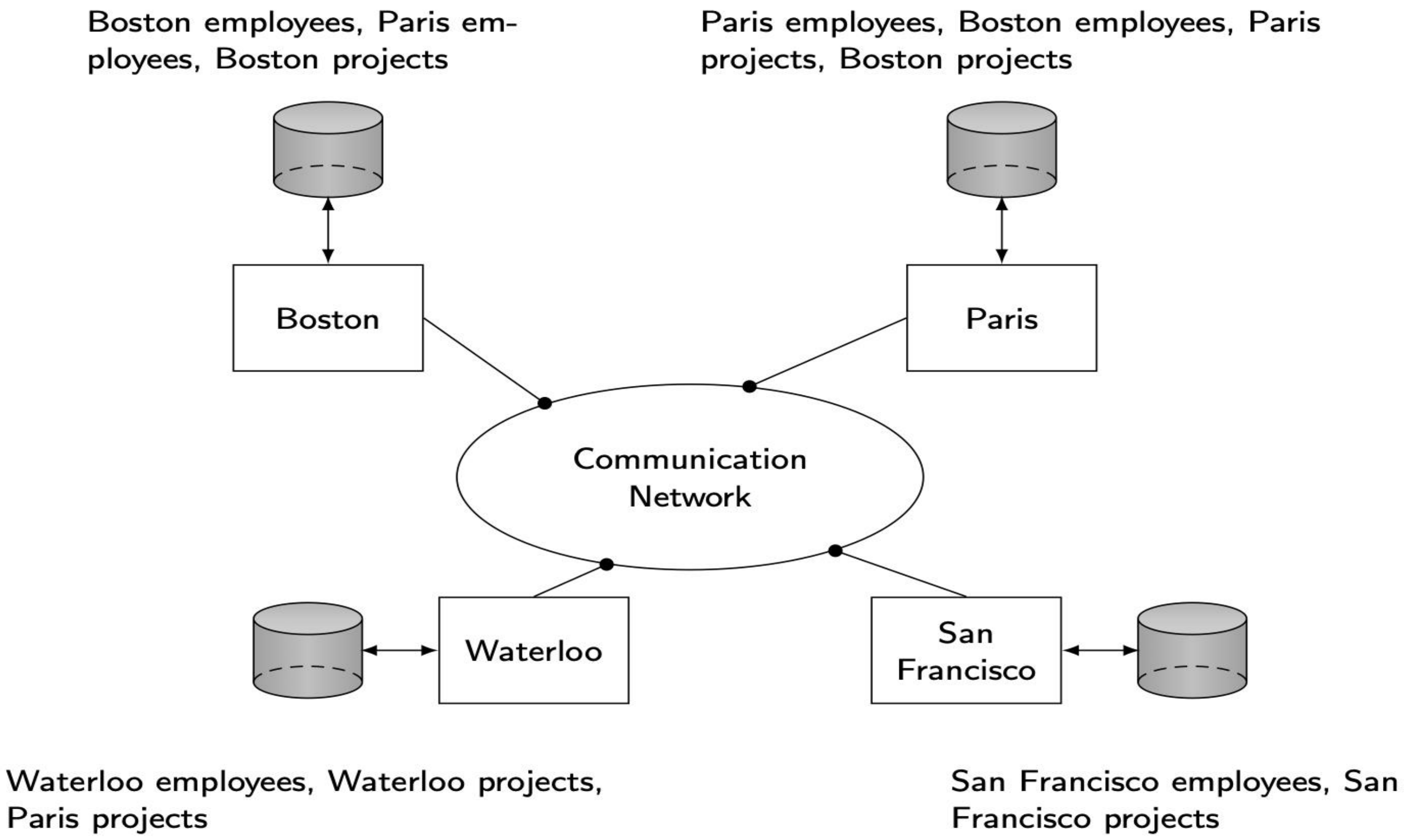
1. Reliable access to data through distributed transactions,

2. improved performance, and

3. Easier system expansion.

Promises of DDBSs

1. Transparent management of distributed & replicated data


Boston employees, Paris employees, Boston projects
Paris employees, Boston employees, Paris projects, Boston projects
Boston
Paris
Communication Network
Waterloo
San Francisco
Waterloo employees, Waterloo projects, Paris projects
San Francisco employees, San Francisco projects

Transparency

- Transparency refers to separation of the higher-level semantics of a system from lower-level implementation issues.

- In other words, a transparent system “hides” the implementation details from users.

- The advantage of a fully transparent DBMS is the high level of support that it provides for the development of complex applications

- **Example :** Consider an engineering firm that has offices in Boston, Waterloo, Paris and San Francisco.

Relational Database Concepts

Example : Use a database that models an engineering company. The entities to be modelled are the employees (**EMP**) and projects (**PROJ**).

For each employee, keep track of the

- employee number (**ENO**),
- name (**ENAME**),
- title in the company (**TITLE**),
- Salary (**SAL**),
- identification number of the project(s) the employee is working on (**PNO**),
- responsibility within the project (**RESP**), and duration of the assignment to the project (**DUR**) in months.
- Similarly, for each project, to store the project number (**PNO**), the project name (**PNAME**), & the project budget (**BUDGET**).

Transparency

They run projects at each of these sites & would like to maintain a database of their employees, the projects and other related data. Assuming that the database is relational, we can store this information in two relations:

first relation **second relation**

EMP(ENO, ENAME, TITLE) & **PROJ(PNO, PNAME, BUDGET)**

Introduce a

❖ **third relation** to store salary information: **SAL(TITLE, AMT) &**

❖ **fourth relation ASG**

which indicates which employees have been assigned to **which projects** for **what duration** with **what responsibility**:

ASG(ENO, PNO, RESP, DUR).

For the time being, it is sufficient to note that this nomenclature indicates that we have just defined a relation with **three attributes**: **ENO, ENAME, TITLE** **ENO** (which is the key, identified by underlining), **ENAME** and **TITLE**.

Transparency

If all of this data were stored in a **centralized DBMS**, and we wanted to find out the names and employees who worked on a project for **more than 12 months**, we would specify this using the following **SQL** query:

```
SELECT      ENAME, AMT
FROM        EMP, ASG, SAL
WHERE       ASG.DUR > 12
AND         EMP.ENO = ASG.ENO
AND         SAL.TITLE = EMP.TITLE
```

However, given the distributed nature of this firm’s business, it is preferable, under these circumstances, to localize data such that data about the employees in **Waterloo office** are stored in Waterloo, those in the **Boston office** are stored in Boston, and so forth.

Transparency

The same applies to the project and salary information. Thus, what we are engaged in is a process where we partition each of the relations and store each partition at a different site. This is known as **fragmentation**

Furthermore, it may be preferable to duplicate some of this data at other sites for performance and reliability reasons

Data Independence

Data independence is a fundamental form of transparency that we look for within a DBMS.

It is also the only type that is important within the context of a centralized DBMS.

Data Definition :
One Level (the logical structure of the data are specified) (Schema Definition)

Other level its physical structure (Physical data description) Logical data description

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

Data Fragmentation : Horizontal vs Vertical

- Division of relation r into fragments $r_1, r_2, \ldots, r_n$ which contain sufficient information to reconstruct relation r.
- **Horizontal fragmentation**: each tuple of *r* is assigned to one or more fragments
- **Vertical fragmentation**: the schema for relation *r* is split into several smaller schemas
  - All schemas must contain a common candidate key (or superkey) to ensure lossless join property.
  - A special attribute, the tuple-id attribute may be added to each schema to serve as a candidate key.

Horizontal Fragmentation of account Relation

branch_name	*account_number*	*balance*
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

$account_1 = \sigma_{branch\_name=\text{“Hillside”}}$
$(account)$

branch_name	*account_number*	*balance*
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

$account_2 = \sigma_{branch\_name=\text{“Valleyview”}}$
$(account)$

Vertical Fragmentation of *employee_info* Relation

branch_name	***customer_name***	***tuple_id***
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

deposit$_1$ = $\Pi_{branch\_name,\ customer\_name,\ tuple\_id}$ ***(employee_info)***

account_number	***balance***	***tuple_id***
A-305	500	1
A-226	336	2
A-177	205	3
A-402	10000	4
A-155	62	5
A-408	1123	6
A-639	750	7

deposit$_2$ = $\Pi_{account\_number,\ balance,\ tuple\_id}$ ***(employee_info)***

Transparency : Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E8	P3	Manager	40

PROJ

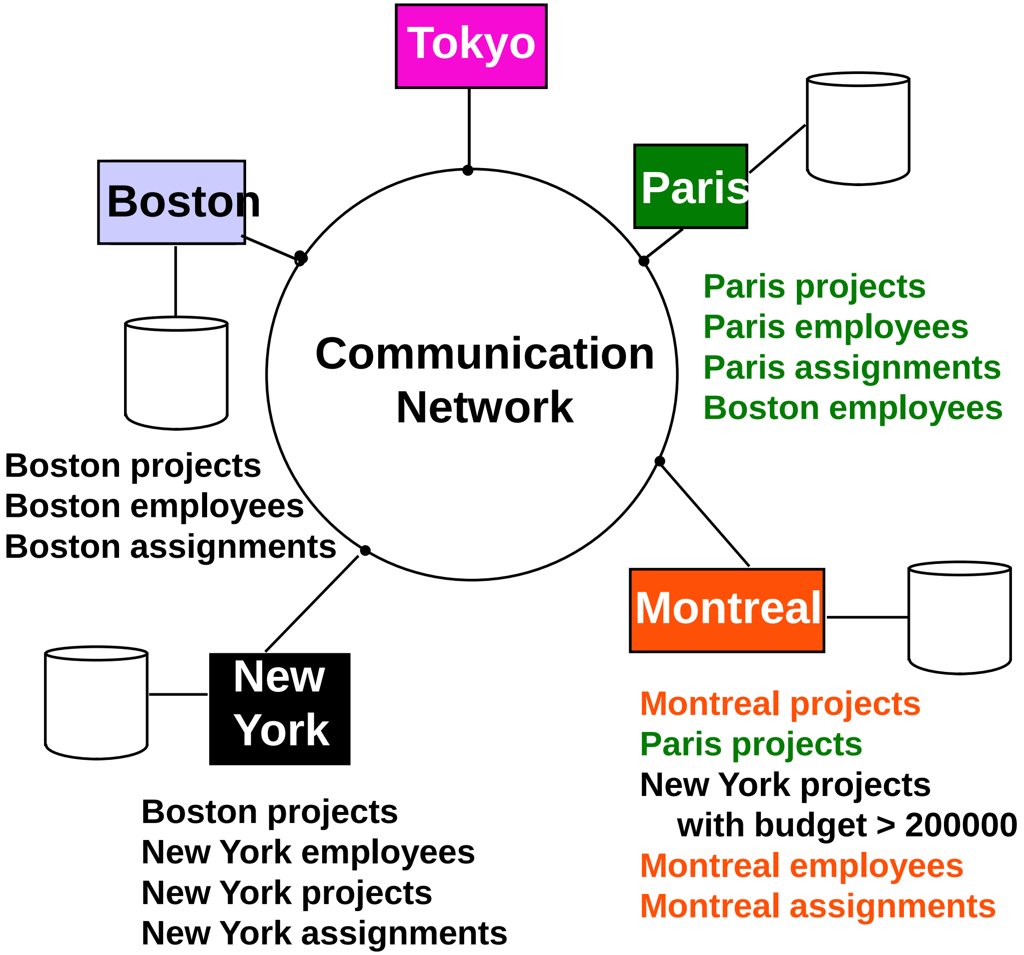
PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

PAY

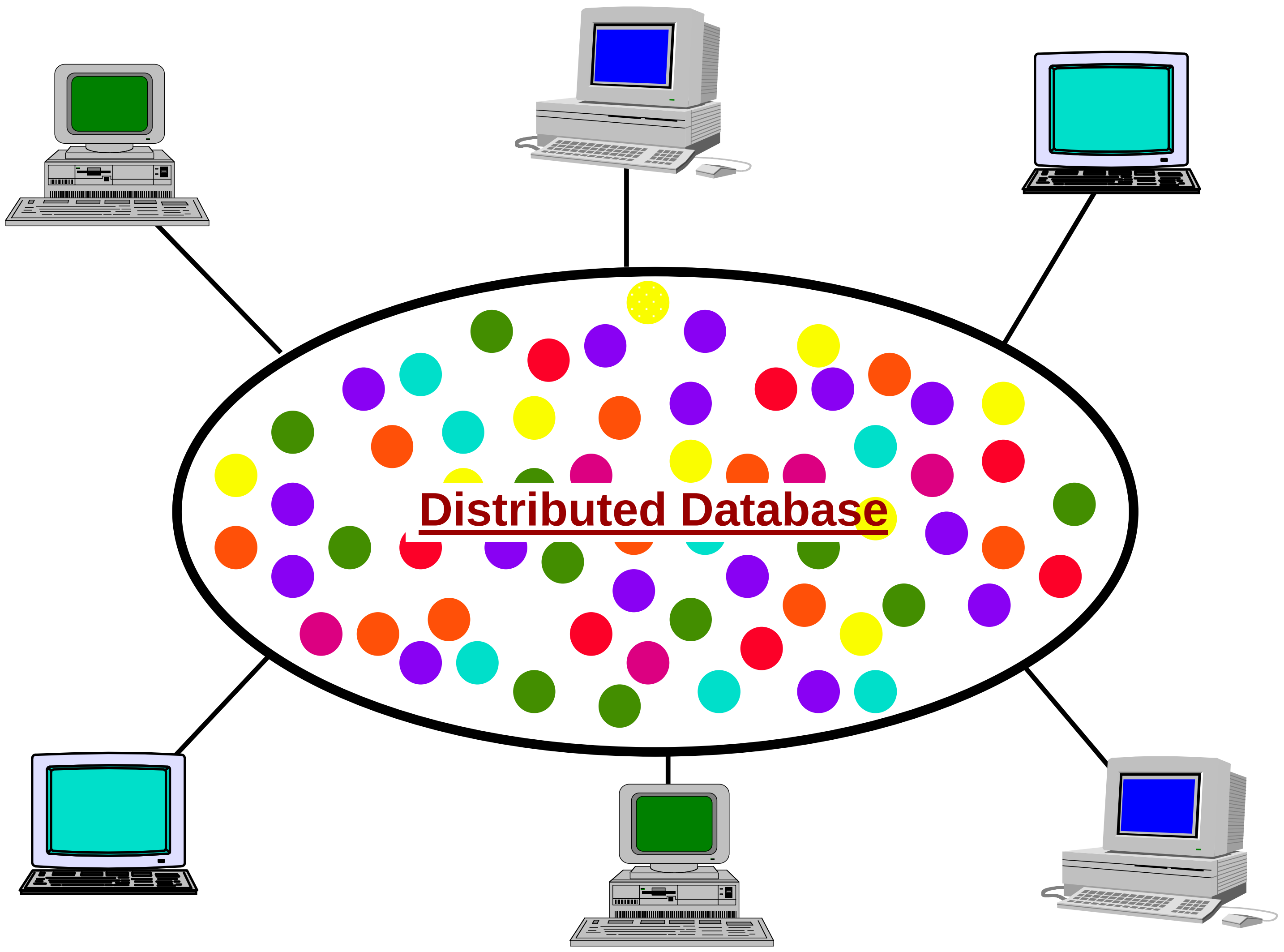
TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

Transparency : Access

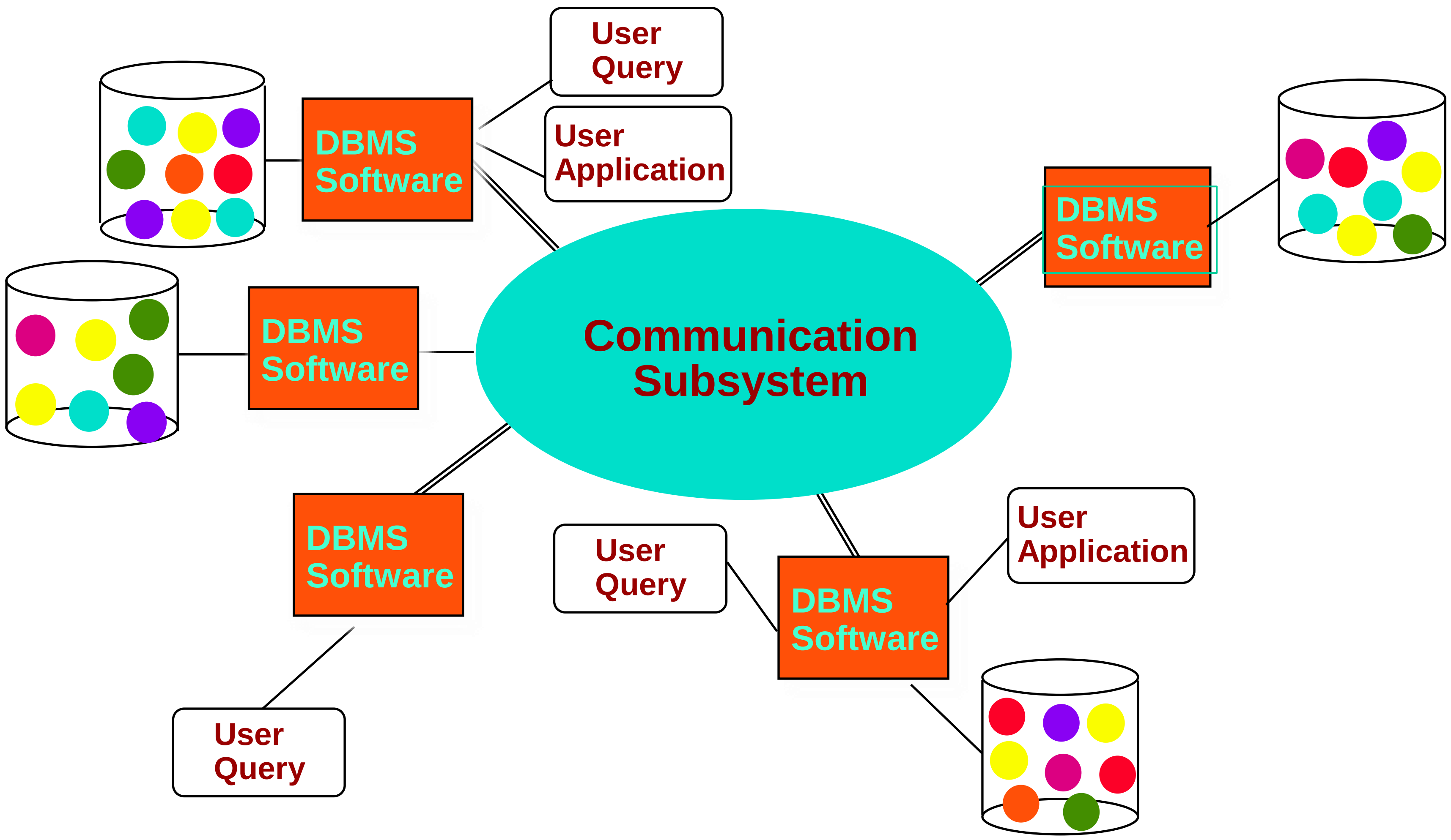
```
SELECT ENAME,SAL
FROM   EMP,ASG,PAY
WHERE  DUR > 12
AND    EMP.ENO = ASG.ENO
AND    PAY.TITLE = EMP.TITLE
```



Distributed Database - User View



Distributed Database - Reality



Promises of DDBSs

2. Reliable access to data through distributed transactions,

3. improved performance, and

4. Easier system expansion.

Promises of DDBSs

1. transparent management of distributed and replicated data,

   a) Data Dependence
   b) Network transparency
   c) Replication transparency
   d) Fragmentation transparency
   e) Who Should Provide Transparency?

Network Transparency

- ❖ In **centralized database systems**, the only available resource that needs to be shielded from the user is the data (i.e., **the storage system**).
- ❖ In a **distributed database environment**, however, there is a second resource that needs to be managed in much the same manner: **the network**.

Preferably, the user should be protected from the operational details of the network; possibly even hiding the existence of the network.

This type of transparency is referred to as **network transparency** or **distribution transparency**.
Sometimes two types of distribution transparency are identified: **location transparency** and **naming transparency**

Replica Transparency

- ❖ From performance, reliability, and availability reasons, it is usually **desirable to be able to distribute data in a replicated fashion across** the machines on a network.
- ❖ Such replication helps performance since diverse and conflicting user requirements can be more easily accommodated.
- ❖ For example, data that are commonly accessed by one user **can be placed on that user's local machine** as well as on the machine of another user with the same access requirements.

Assuming that data are replicated, the transparency issue is whether the users should be aware of the existence of copies or whether the system should handle the management of copies and the user should act as if there is a single copy of the data (note that we are not referring to the placement of copies, only their existence).

Fragmentation /Transparency

- it is commonly desirable **to divide each database relation i**nto smaller fragments and treat each **fragment as a separate database object** (i.e., another relation).

- For reasons of **performance, availability**, and **reliability.**

- Furthermore, fragmentation **can reduce** the negative effects of replication.

- **Each replica** is not the full relation but only a subset of it; thus less space is required and fewer data items need be managed.

Fragmentation /Transparency

- There **are two general types of fragmentation** alternatives.
- In one case, called **horizontal fragmentation**, a relation is partitioned into a set of sub-relations each of which have a subset of the tuples (**rows)** of the original relation.

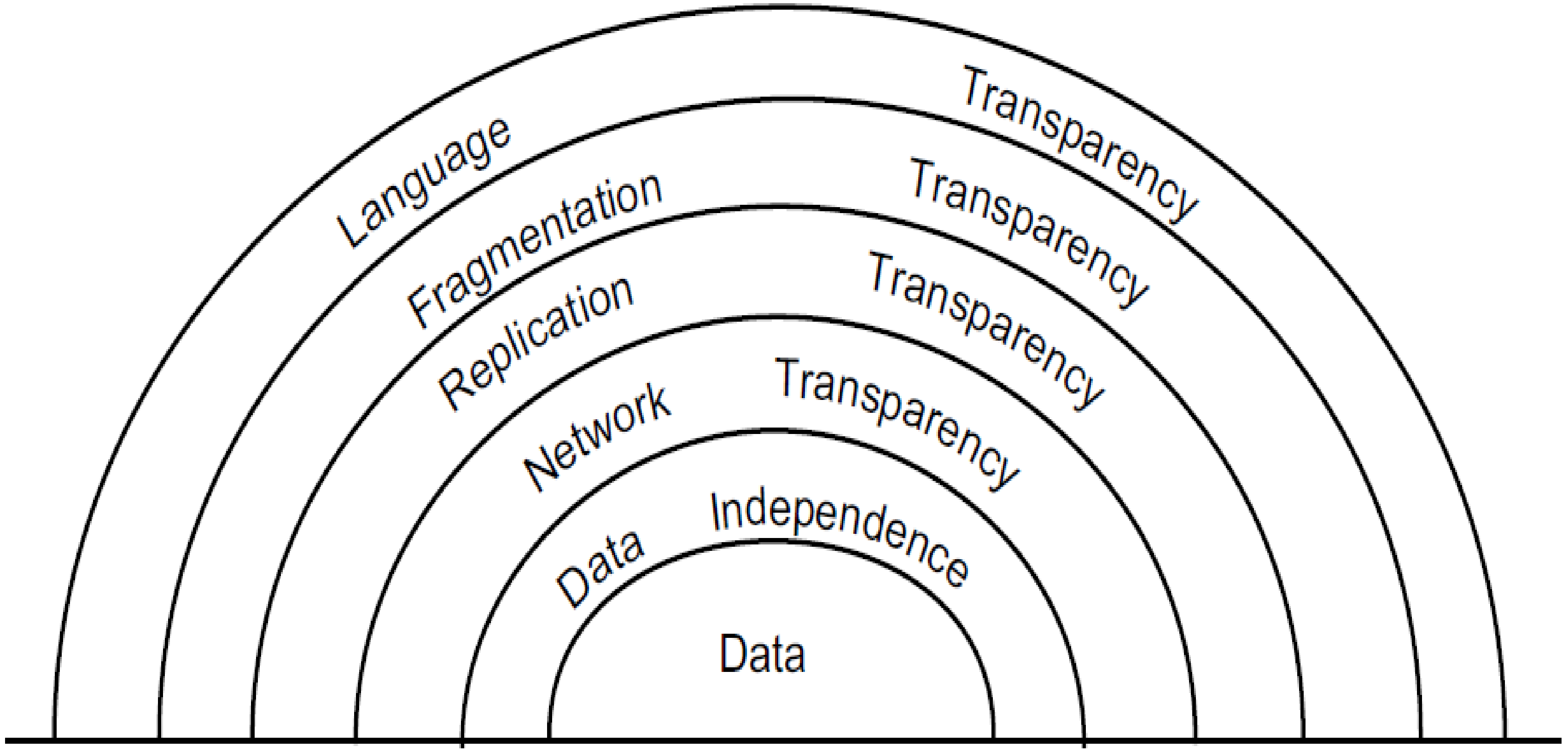
- The second alternative is **vertical fragmentation** where each sub-relation is defined on a subset of the attributes (**columns**) of the original relation.

- When database objects are fragmented, we have to deal with the problem of handling user queries that are specified on entire relations but have to be executed on sub-relation

- Typically, sometimes, it requires a translation from what is called a *global query* to several *fragment queries*

Promises of DDBSs

1.transparent management of distributed and replicated data,



Layers of Transparency

Reliability Through Distributed Transactions

- ❖ A transaction is a basic unit of consistent and reliable computing, consisting of a sequence of database operations executed as an atomic action
- ❖ DBMS that provides full transaction support guarantees that concurrent execution of user transactions will not violate database consistency in the face of system failures as long as each transaction is correct, i.e., obeys the integrity rules specified on the database
- ❖ Distributed transactions execute at a number of sites at which they access the local database.

Example :

```
Begin transaction SALARY UPDATE
begin
EXEC SQL UPDATE PAY
SET SAL = SAL*1.1
end.
```

Improved Performance - DBMS

- ❖ First, a distributed DBMS fragments the conceptual database, enabling data to be stored in close proximity to its points of use (also called data localization).

- ❖ The second case point is that the inherent parallelism of distributed systems may be exploited for inter-query and intra-query parallelism.

  - Inter-query parallelism results from the ability to execute multiple queries at the same time while intra-query parallelism is achieved by breaking up a single query into a number of subqueries each of which is executed at a different site, accessing a different part of the distributed database.

Promises of DDBS : Design Issues

1. Distributed Database Design
2. Distributed Directory Management
3. Distributed Query Processing
4. Distributed Concurrency Control
5. Distributed Deadlock Management
6. Reliability of Distributed DBMS
7. Replication
8. Relationship among problems
9. Additional Issues

Promises of DDBS : Design Issues

1. **Distributed Directory Management**
   - A directory contains information (such as descriptions and locations) about data items in the database
   - Problems related to directory management are similar in nature to the database placement problem.
   - It can be centralized at one site or distributed over several sites; there can be a single copy or multiple copies.

Promises of DDBS : Design Issues

1. **Query processing deals** with designing algorithms that analyze queries and convert them into a series of data manipulation operations.

- The problem is **how to decide** on a strategy for executing each query over the network in the most cost-effective way, however cost is defined.

- The factors to be considered are the **distribution of data**, **communication costs**, and **lack of sufficient locally-available** information.

Promises of DDBS : Design Issues

1. **Query processing deals** with designing algorithms that analyze queries and convert them into a series of data manipulation operations.

- The objective is to optimize where the inherent parallelism is used to improve the performance of executing the transaction, subject to the above-mentioned constraints.
- The problem is NP-hard in nature, and the approaches are usually heuristic

Promises of DDBS : Design Issues

1. **Concurrency control** involves the synchronization of accesses to the distributed database, such that the **integrity of the database is maintained**.

- One of the most extensively studied problems in the DDBS field.
- The concurrency control problem in a distributed context is somewhat different than in a centralized framework.

Promises of DDBS : Design Issues

1. **Concurrency control** involves the synchronization of accesses to the distributed database, such that the **integrity of the database is maintained**.

- One not only has to worry about the integrity of a single database, but also about the consistency of multiple copies of the database.

- The condition that requires all the values of multiple copies of every data item to converge to the same value is called mutual consistency.

Promises of DDBS : Design Issues

1. **Distributed Deadlock Management** : The deadlock problem in **DDBSs** is similar in nature to that encountered in operating systems.

- The competition among users for access to a set of resources (data, in this case) can result in a deadlock if the synchronization mechanism is based on locking.

- The well-known alternatives of prevention, avoidance, and detection/recovery also apply to **DDBSs**.

Promises of DDBS : Design Issues

1. **Replica** : If the distributed database is (partially or fully) replicated, it is necessary to implement protocols that ensure the consistency of the replicas, i.e., copies of the same data item have the same value.

- These protocols can be eager in that they force the **updates to be applied** to all the replicas before the transaction completes, or

- they may be lazy so that the transaction **updates one copy** (called the **master**) from which updates are propagated to the others after the transaction completes.

DDBS : Relationship among the Problems

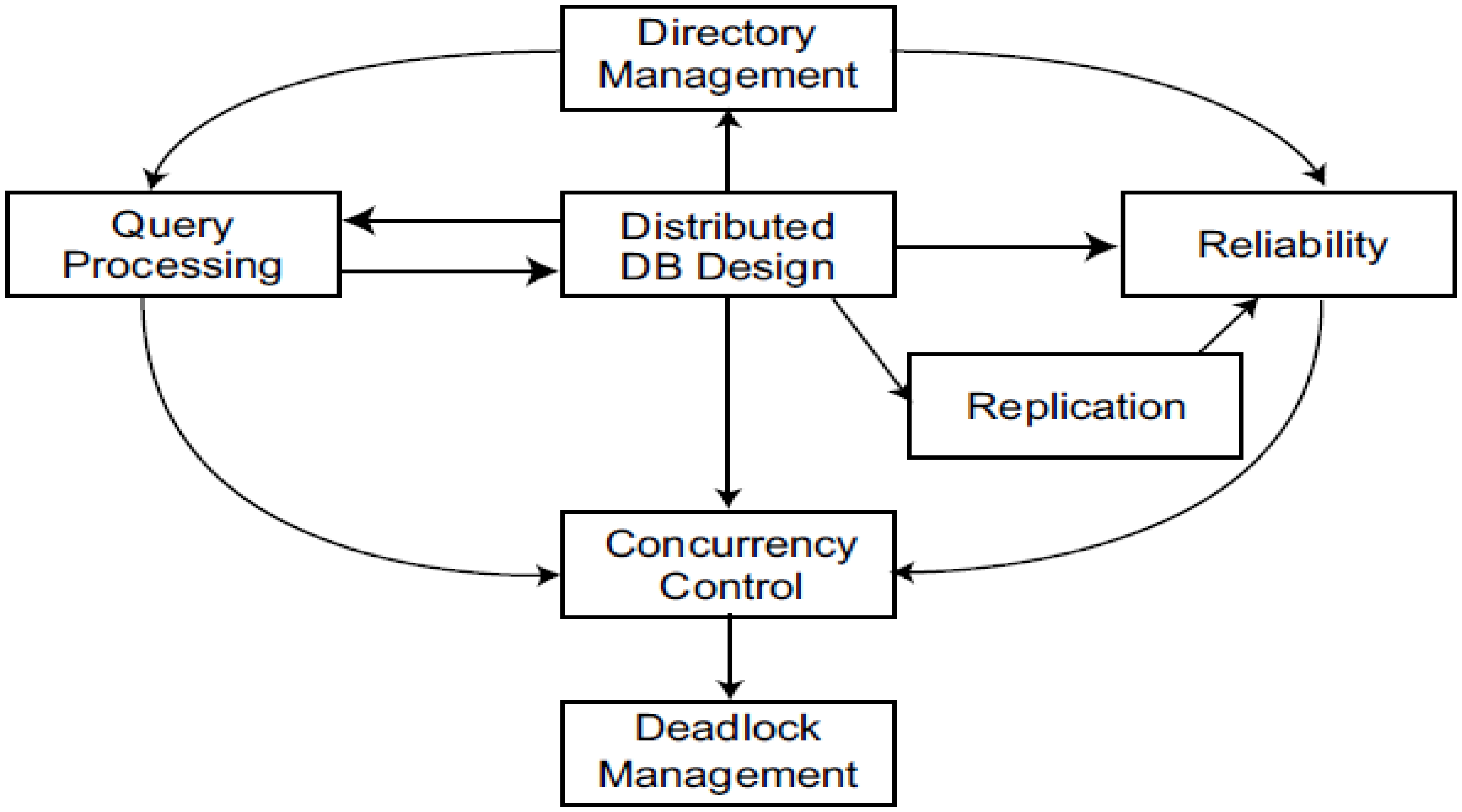
Relationship Among Research Issues

- ❖ Each problem is affected by the solutions found for the others, and in turn affects the set of feasible solutions for them.

- ❖ The design of distributed databases affects many areas. It affects directory management, because the definition of fragments and their placement determine the contents of the directory (or directories) as well as the strategies that may be employed to manage them.

DDBS : Relationship among the Problems

Relationship Among Research Issues



DDBS : Additional Issues

1. Multi-database systems (also called federated databases and data integration systems - database integration)
2. Growth of Internet - Network Problems (Wi-Fi & Mobile)
3. Peer-to-Peer Computing
4. Peer-to-Peer Data Management
5. Distributed Data Bases and Parallel Databases - Relationship
6. Each Site - Single Logical Computer or Parallel Cluster
7. Parallel File System - Data Base Integration

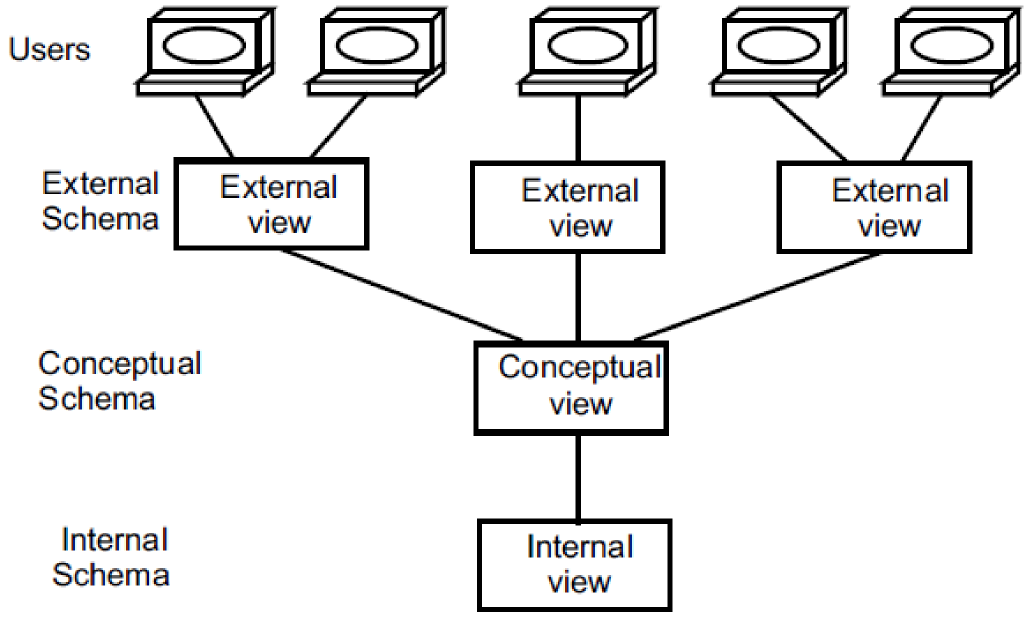
Distributed DBMS Architecture

Distributed DBMS Architecture

1. ANSI/SPARC Architecture
2. Generic Centralized DBMS Architecture
3. Architectural Models for Distributed DBMSs
4. Autonomy
5. Distribution
6. Heterogeneity
7. Architectural Alternatives
8. Client/Server Systems
9. Peer-to-Peer Systems
10. Multidatabase System Architecture

ANSI/SPARC DBMS Architecture

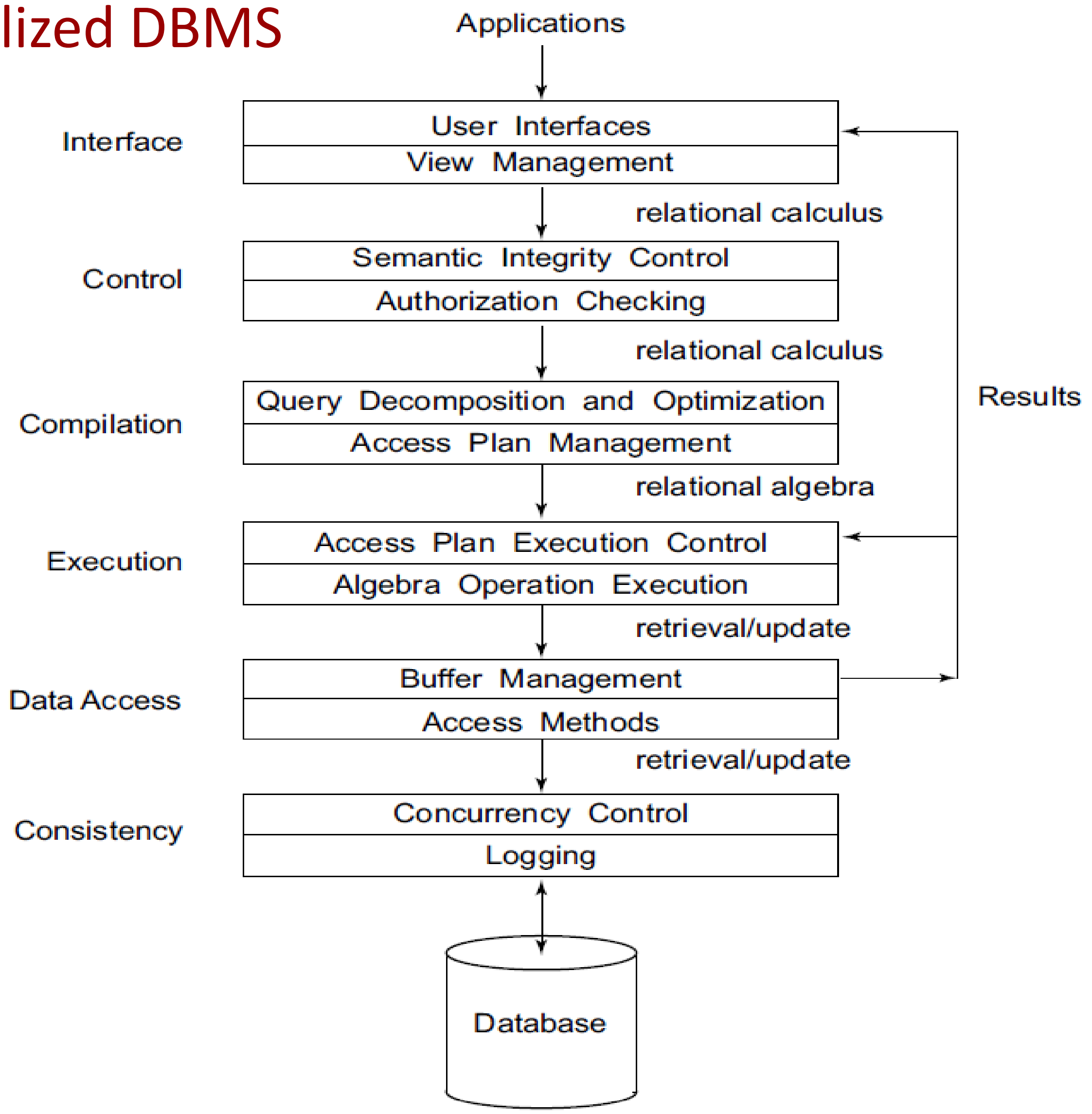
There are three views of data: the external view, which is that of the end user, who might be a programmer; the internal view, that of the system or machine; and the conceptual view, that of the enterprise. For each of these views, an appropriate schema definition is required.



A Generic Centralized DBMS Architecture

Functional Layers of a Centralized DBMS

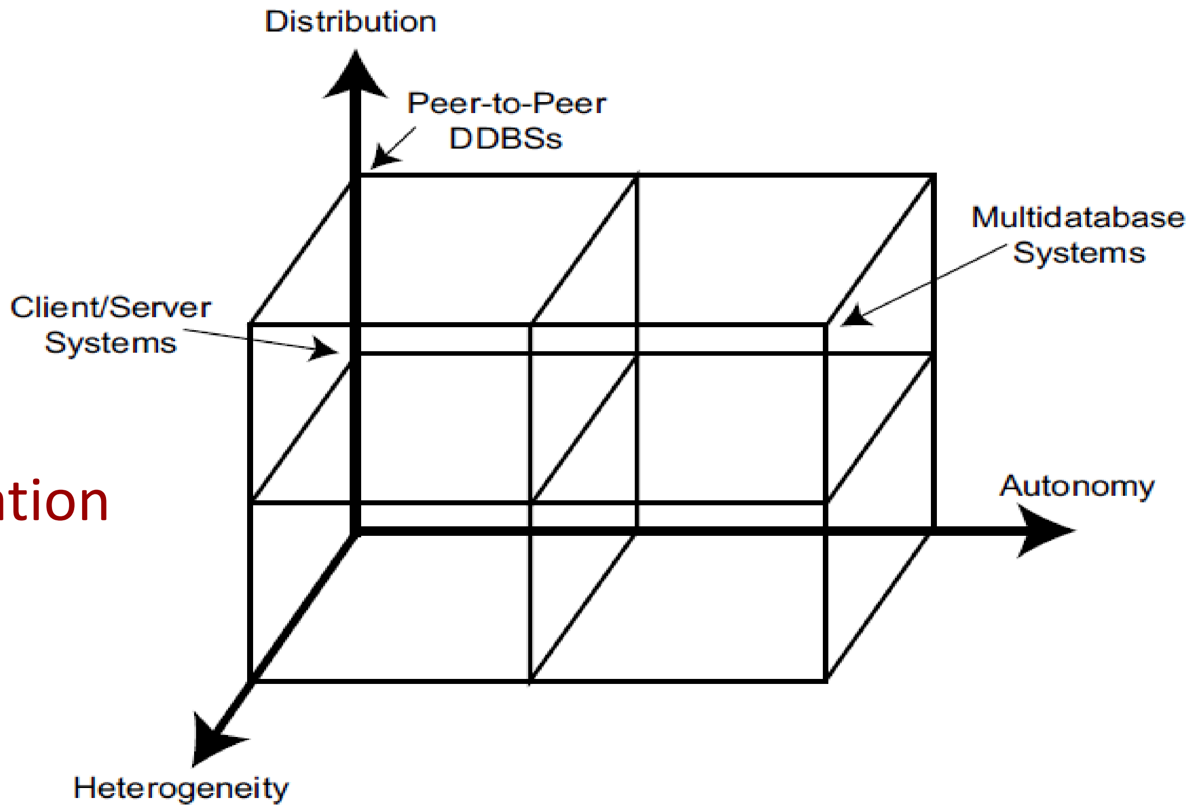
The functions performed by a DBMS can be layered as in below Figure where the arrows indicate the direction of the data and the control flow. Taking a top-down approach, the layers are the interface, control, compilation, execution, data access, and consistency management.



Architectural Models for Distributed DBMSs

We now consider the possible ways in which a distributed DBMS may be architected. We use a classification (Refer below Figure) that organizes the systems as characterized with respect to (1) the autonomy of local systems, (2) their distribution, and (3) their heterogeneity.

DBMS Implementation Alternatives



Autonomy

- ❖ Autonomy, in this context, refers to the distribution of control, not of data.

- ❖ It indicates the degree to which individual DBMSs can operate independently.

- ❖ Autonomy is a function of a number of factors such as

  - whether the component systems (i.e., individual DBMSs) exchange information,
  - whether they can independently execute transactions, and
  - whether one is allowed to modify them.

Autonomy

Autonomy refers to the distribution (or decentralization) of control

Requirements of an autonomous system have been specified as follows It indicates the degree to which individual DBMSs can operate independently.

1. The local operations of the individual DBMSs are not affected by their participation in the distributed system.

2. The manner in which the individual DBMSs process queries and optimize them should not be affected by the execution of global queries that access multiple databases.

3. System consistency or operation should not be compromised when individual DBMSs join or leave the distributed system.

Autonomy

Autonomy refers to the distribution (or decentralization) of control

On the other hand, the dimensions of autonomy can be specified as follows.

1. Design autonomy: Individual DBMSs are free to use the data models and transaction management techniques that they prefer.
2. Communication autonomy: Each of the individual DBMSs is free to make it own decision as to what type of information it wants to provide to the other DBMSs or to the software that controls their global execution.
3. Execution autonomy: Each DBMS can execute the transactions that are submitted to it in any way that it wants to.

Distribution

Whereas autonomy refers to the distribution (or decentralization) of control, the distribution dimension of the taxonomy deals with data

- Physical distribution of data over multiple sites; as we discussed earlier, the user sees the data as one logical pool.
- Number of ways DBMSs for distribution of data (client/server distribution and peer-to-peer distribution (or full distribution).

Heterogeneity

Heterogeneity may occur in various forms in distributed systems, ranging from hardware heterogeneity and differences in networking protocols to variations in data managers.

- Query languages, and transaction management protocols.
- Representing data with different modelling tools creates heterogeneity.
- SQL : standard relational query language,
- Many different implementations and every vendor’s language has a slightly different flavor

Architecture Alternatives

The distribution of databases, their possible heterogeneity, and their autonomy are orthogonal issues.

Consequently, following the above characterization, there are many different possible architectures

Architectural Models : Client Server Systems

❖ Provides a two-level architecture which makes it easier to manage the complexity of modern DBMSs and the complexity of distribution.

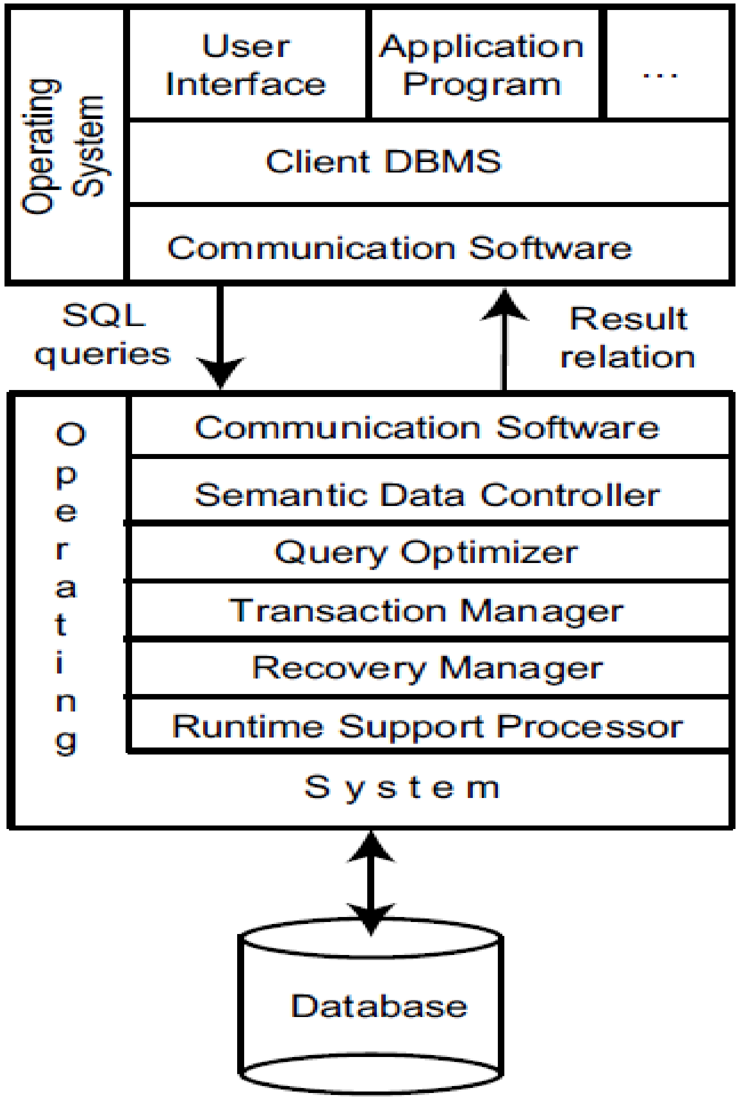
- If one takes a process-centric view, then any process that requests the services of another process is its client and vice versa.
- Important : Note that "**client/server computing**" and "**client/server DBMS**," as it is used in our context, do not refer to processes, but to actual machines.
- Focus on

what software should run on the **client machines** and what software should run on the **server machine**.

Architectural Models : Client Server Systems

❖ Client/Server Reference Architecture

- Operating system and communication software that runs on both the client and the server, but we only focus on the DBMS related functionality.



Architectural Models : Client Server Systems

- The functionality allocation between clients and serves differ in different types of distributed DBMSs

  (e.g., relational versus object-oriented).

- In **relational systems**, the server does most of the data management work.
  - ✓ This means that all of
    query processing and optimization,
    transaction management and storage management
  is done at the server.

Architectural Models : Client Server Systems

- Number of different types of **client/server** architecture. (Simplest : there is only one server which is accessed by multiple clients.) Call this multiple **client/single server.**
- A more sophisticated client/server architecture is one where there are multiple servers in the system (the so-called **multiple client/multiple server** approach).
- In this case, two alternative management strategies are possible:

either each client manages its own connection to the appropriate server

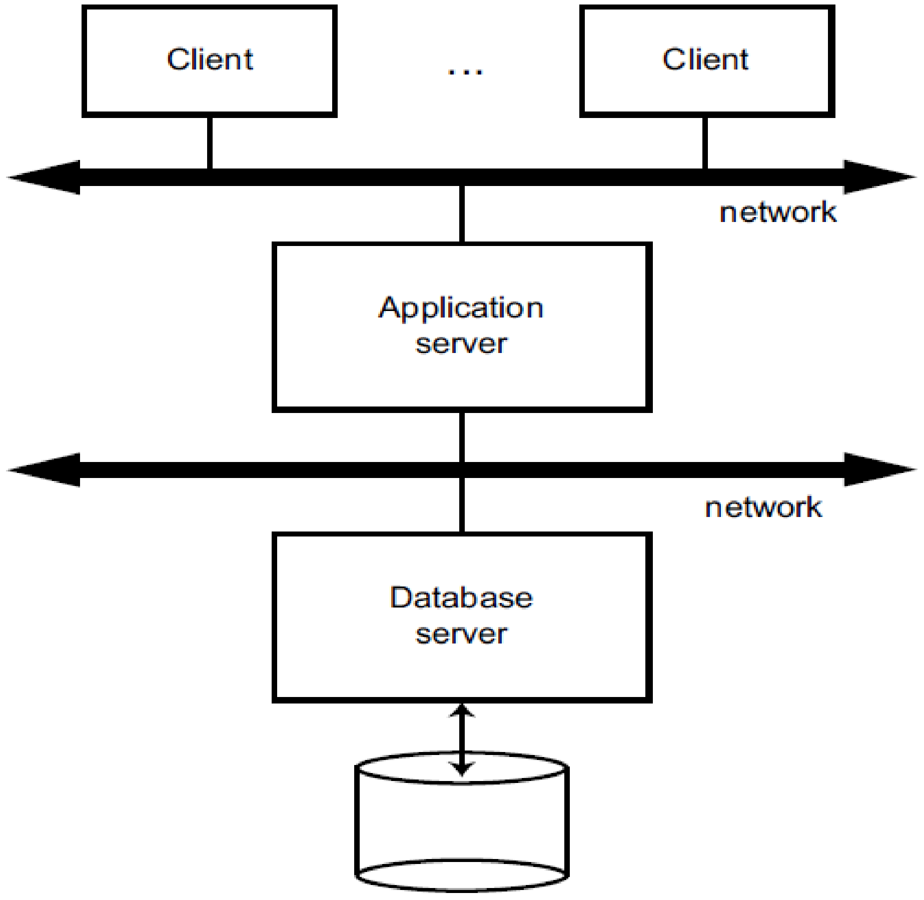
or each client knows of only its "**home server**" which then communicates with other servers as required.

Architectural Models : Client Server Systems

- The functionality allocation between clients and serves differ in different types of distributed DBMSs (e.g., relational versus object-oriented). In relational systems, the server does most of the data management work. This means that all of query processing and optimization, transaction management and storage management is done at the server.
- The client, in addition to the application and the user interface, has a DBMS client module that is responsible for managing the data that is cached to the client and (sometimes) managing the transaction locks that may have been cached as well.

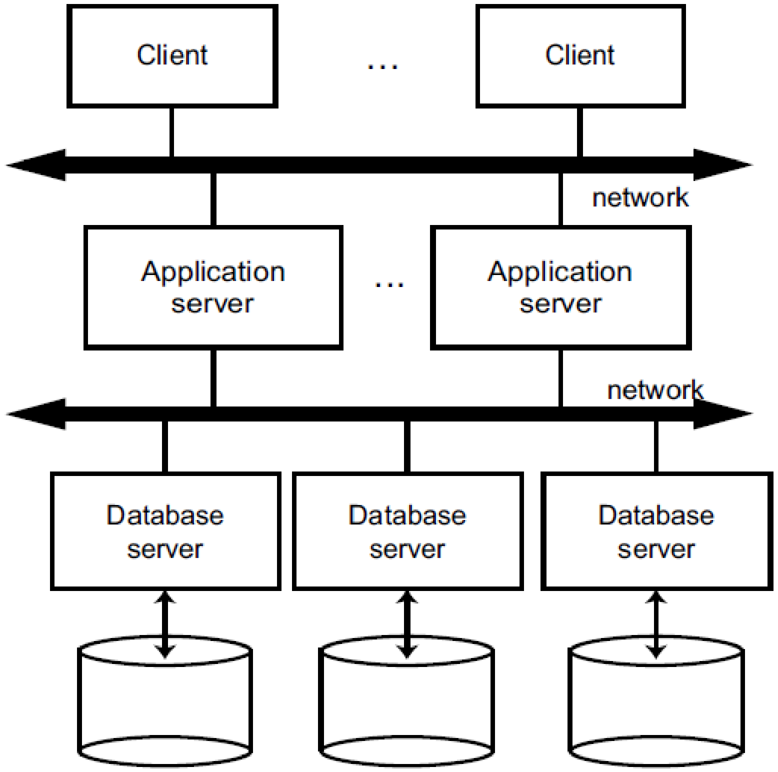
Distributed DBMS Architecture : Database Server Approach

Figure illustrates a simple view of the database server approach, with application servers connected to one database server via a communication network.



Distributed DBMS Architecture : Database Servers Approach

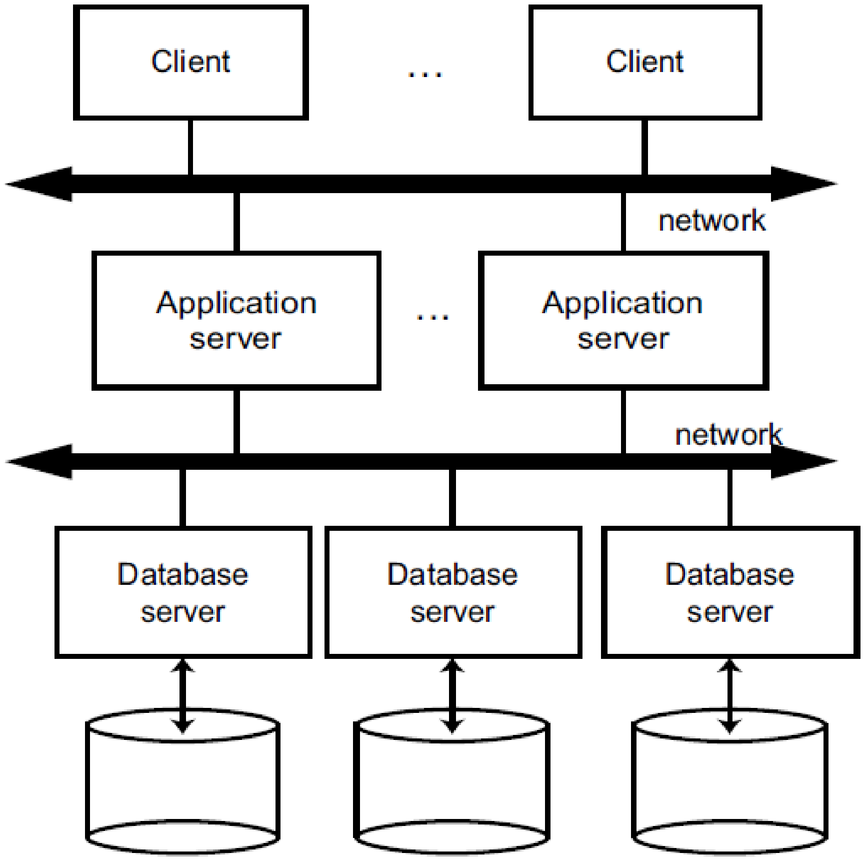
The application server approach (indeed, a **n-tier** distributed approach) can be extended by the introduction of multiple database servers and multiple application servers (Figure), as can be done in classical client/server architectures.



Distributed DBMS Architecture : Database Servers Approach

each application server is dedicated to one or a few applications,

while database servers operate in the multiple server fashion discussed above.



Distributed DBMS Architecture : Peer-to-Peer Systems

- ❖ The early works on distributed DBMSs all focused on peer-to-peer architectures where there was no differentiation between the functionality of each site in the system
- ❖ After a decade of popularity of client/server computing, peer-to-peer have made a comeback in the last few years (primarily spurred by file sharing applications) and some have even positioned peer-to-peer data management as an alternative to distributed DBMSs.
- ❖ Designing peer-to-peer database systems - understand fundamental techniques

Distributed DBMS Architecture : Peer-to-Peer Systems

❖ Modern peer-to-peer systems have two important differences from their earlier relatives.

- The **first** is the massive distribution in current systems. While in the early days we focused on a few (perhaps at most tens of) sites, current systems consider thousands of sites.
- The **second** is the inherent heterogeneity of every aspect of the sites and their autonomy

Distributed DBMS Architecture : Peer-to-Peer Systems

Data Organisation Point of View

First note that the physical data organization on each machine may be, and probably is, different.

- This means that there needs to be an individual internal schema definition at each site, which we call the **local internal schema (LIS)**.
- The enterprise view of the data is described by the **global conceptual schema (GCS),** which is global because it describes the logical structure of the data at all the sites.

Distributed DBMS Architecture : Peer-to-Peer Systems

Data Organisation Point of View

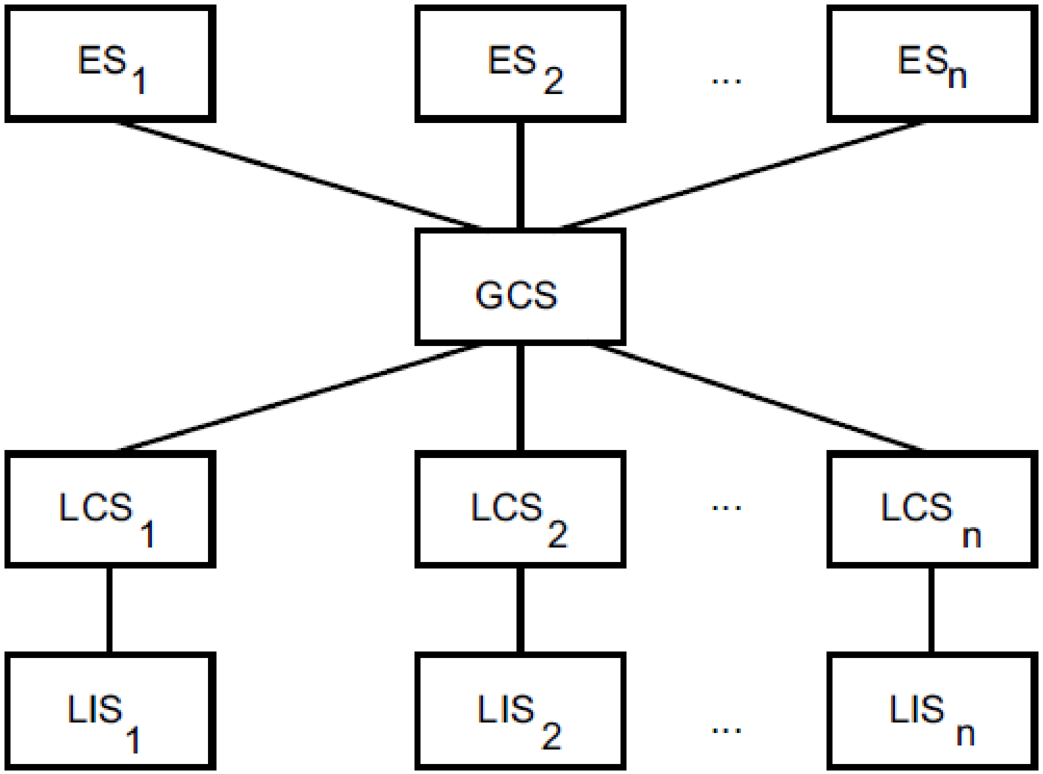
- To handle data fragmentation and replication, the logical organization of data at each site needs to be described. Therefore, there needs to be a third layer in the architecture, the **local conceptual schema (LCS).**

- In the architectural model, the global conceptual schema is the union of the local conceptual schemas.

- Finally, user applications and user access to the database is supported by external schemas (ESs), defined as being above the global conceptual schema.

Distributed DBMS Architecture : Peer-to-Peer Systems

Distributed Database Reference Architecture : The architecture model - provides levels of transparency.



Distributed DBMS Architecture : Peer-to-Peer Systems

Distributed Database Reference Architecture : The architecture model - provides levels of transparency.

- Location and replication transparencies are supported by the definition of the local and global conceptual schemas and the mapping in between.

- Network transparency, on the other hand, is supported by the definition of the global conceptual schema

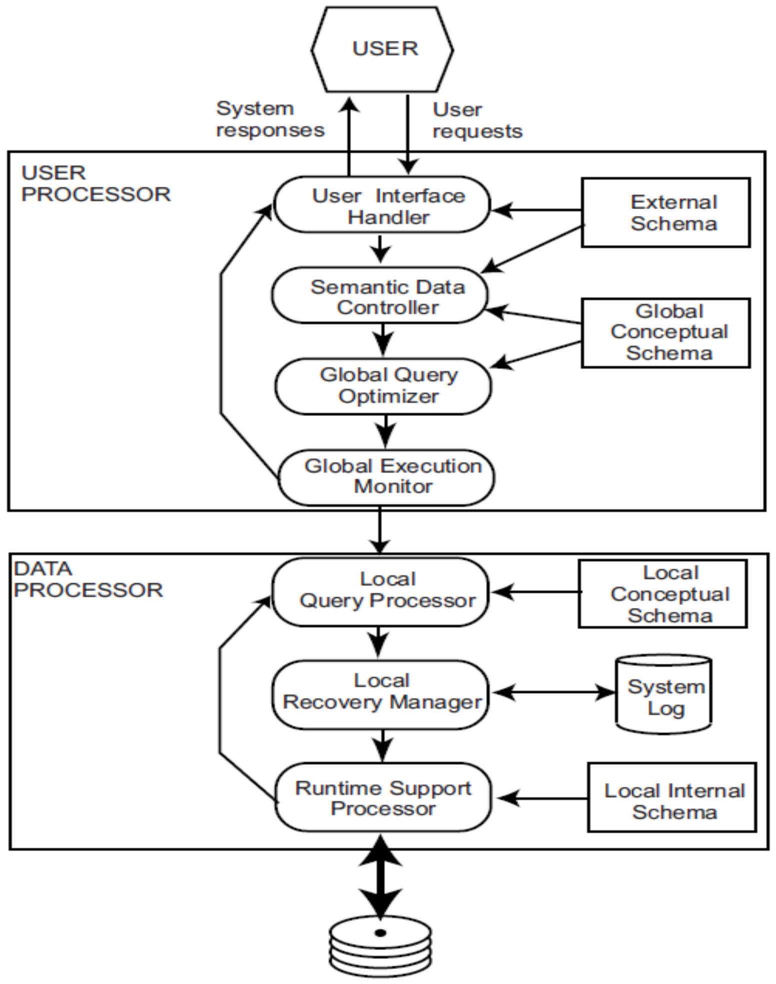
Distributed DBMS Architecture : Components of a Distributed DBMS

One component handles the interaction with users, and another deals with the storage.

The ***first major*** component, which we call the ***user processor***, and the ***second major*** component of a distributed DBMS is the ***data processor***

It is important to note, at this point, that our use of the terms "***user processor***" and ***"data processor"*** does not imply a functional division similar to ***client/server*** systems.

Distributed DBMS Architecture : Components of a Distributed DBMS

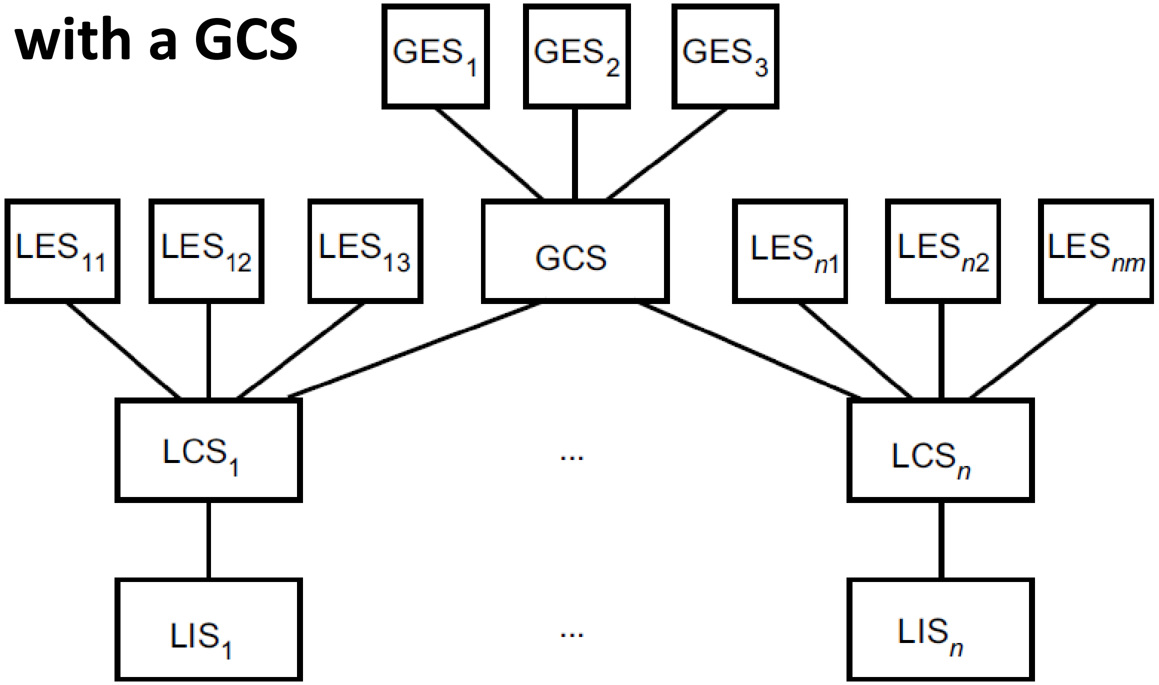


Distributed DBMS Architecture

Multi-database System Architecture

Designing the global conceptual schema in multi-database systems involves the integration of either the local conceptual schemas or the local external schemas

MDBS Architecture with a GCS

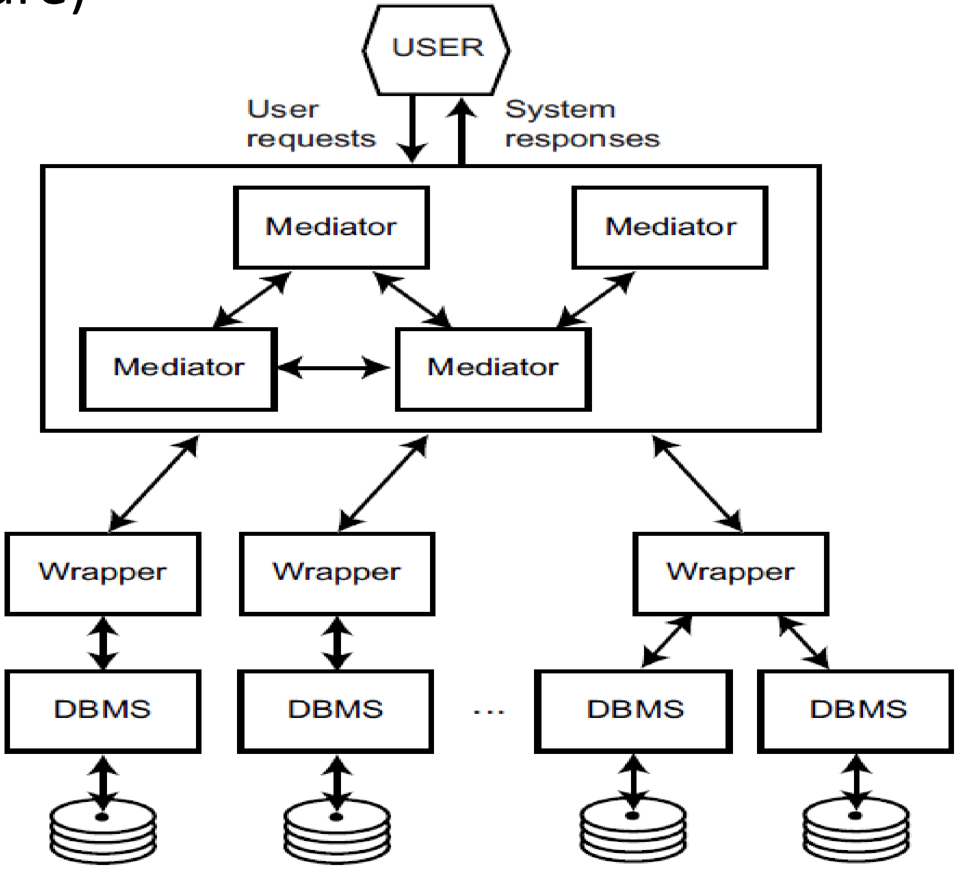


Distributed DBMS Architecture

A popular implementation architecture for MDBSs is the mediator/wrapper approach (Figure)

A **mediator** "is a software module that exploits encoded knowledge about certain sets or subsets of data to create information for a higher layer of applications."

Thus, each mediator performs a particular function with clearly defined interfaces



Mediator/Wrapper Architecture

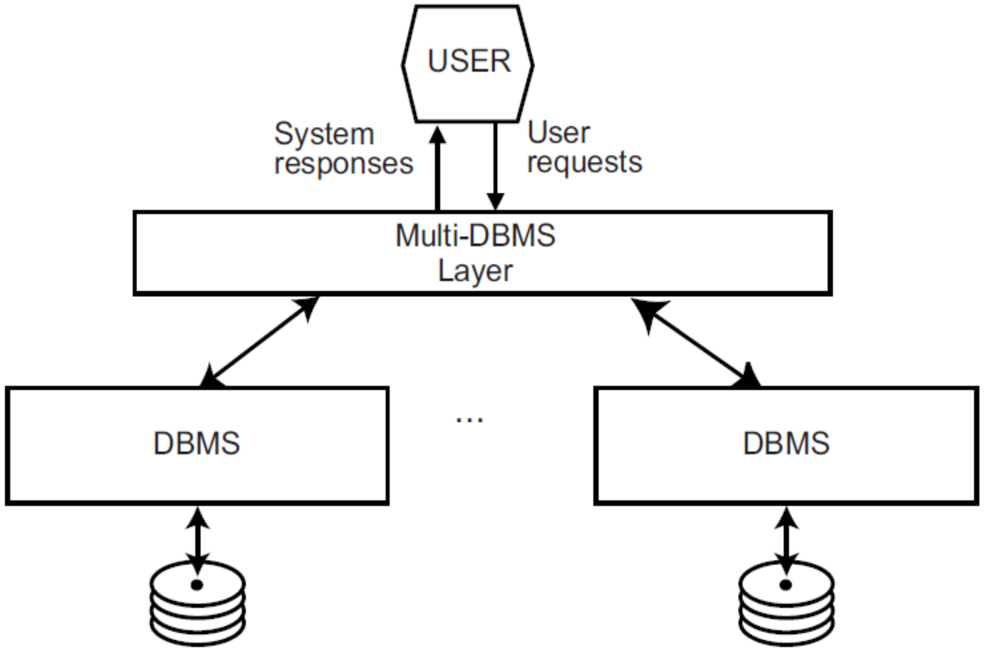
Distributed DBMS Architecture

Multi-database System Architecture

Using this architecture to implement a MDBS, each module in the multi-DBMS layer of Figure is realized as a mediator.

Components of an MDBS

Since mediators can be built on top of other mediators, it is possible to construct a layered implementation. In mapping this architecture to the datalogical view of Figure (Two pages previous) , the mediator level implements the GCS.



Chapter-02

Principles of Distributed Database Systems
Third Edition