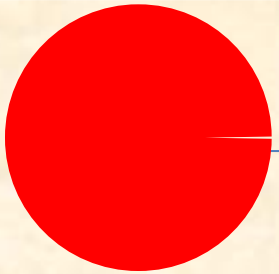


Database System Architectures

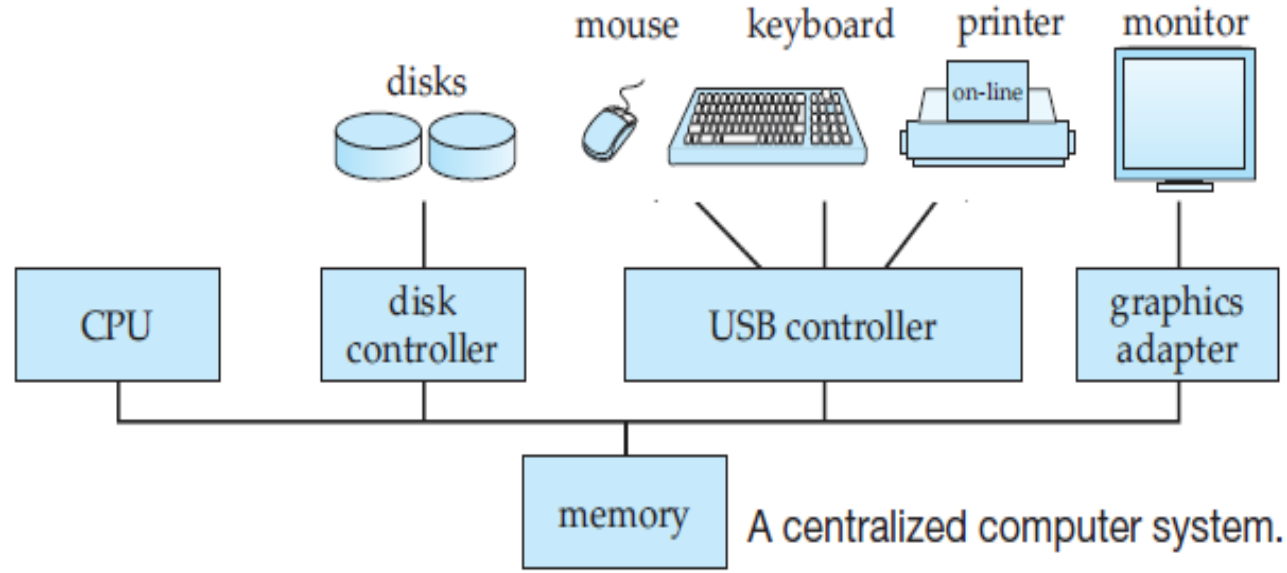
OUTLINE

- ✓ Centralized Systems
- ✓ Client–Server Systems
- ✓ Transaction Server System
- ✓ Data-server systems
- ✓ Computer System Architecture
- ✓ Parallel Processing Systems
- ✓ Speedup
- ✓ Scaleup
- ✓ Distributed Database System
- ✓ Multiprocessor systems vs DDBS
- ✓ Promises of DDBSs
- ✓ Complications Introduced by DDBS
- ✓ What is being distributed in DDBS?
- ✓ Classification of DDBMS
- ✓ Data Fragmentations
- ✓ Correctness Rules for Data Fragmentation
- ✓ Distributed Database System Design Issues
- ✓ Distributed DBMS Architecture
- ✓ ANSI/SPARC Architecture
- ✓ Peer-to-Peer Distributed Systems
- ✓ Components of a Distributed DBMS
- ✓ Multidatabase System (MDBS) Architecture
- ✓ Component-based architectural model of a multi-DBMS

Database System Architectures

- **Centralized Systems**

  Centralized database systems are those that run on a single computer system and do not interact with other computer systems.



A centralized computer system.

Database System Architectures

- **Centralized Systems**

  Single-user system is a desktop unit used by a single person, usually with only one processor and one or two hard disks, and usually only one person using the machine at a time. Personal computers and workstations fall into the first category.

  Multiuser system has more disks and more memory, may have multiple CPUs and has a multiuser operating system. It serves a large number of users who are connected to the system via terminals.

Database System Architectures

- **Centralized Systems**

  These systems have **coarse-granularity parallelism.** Databases running on such machines usually do not attempt to partition a single query among the processors; instead, they run each query on a single processor, allowing multiple queries to run concurrently. Thus, such systems support a higher **throughput**; that is, they allow a greater number of transactions to run per second, although individual transactions do not run any faster.

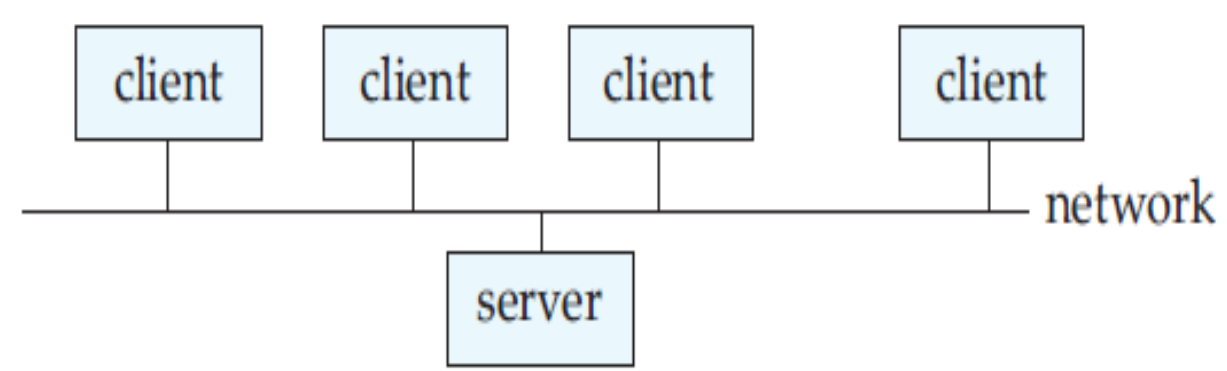
  Machines with **fine-granularity parallelism** have a large number of processors, and database systems running on such machines attempt to parallelize single tasks (queries, for example) submitted by users.

Database System Architectures

- **Client–Server Systems**

  Database functionality can be broadly divided into two parts:

  - Front end and
  - Back end.



General structure of a client–server system.

Database System Architectures

- **Client–Server Systems**

  The back end manages access structures, query evaluation and optimization, concurrency control, and recovery.

  The front end consists of tools such as forms, report writers, and graphical user interface facilities.

  The interface between the front end and the back end is through SQL, or through an application program. Standards such as *ODBC* and *JDBC* , were developed to interface clients with servers.

Database System Architectures

q Server System Architectures

Server systems can be broadly categorized as:

- Transaction servers and
- Data servers.

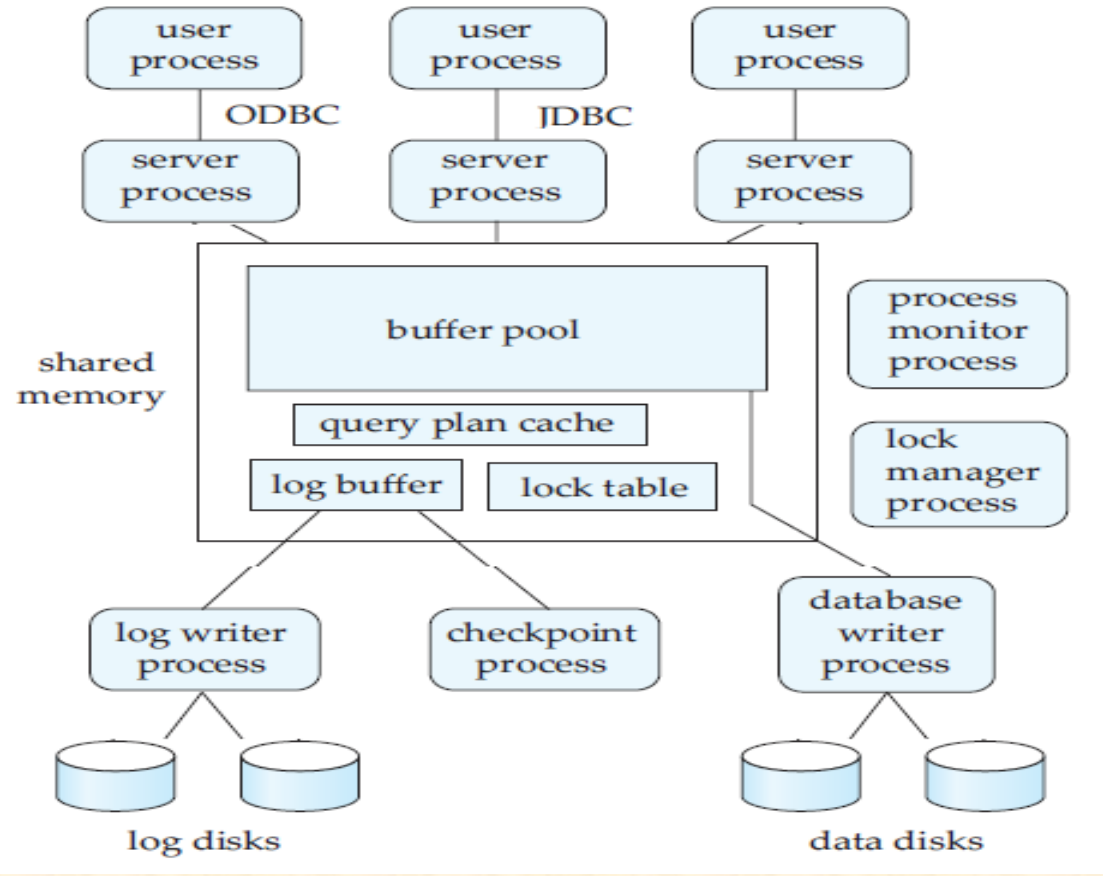
q Transaction Server System

- Also called query-server systems provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client. Usually, client machines ship transactions to the server systems, where those transactions are executed, and results are shipped back to clients that are in charge of displaying the data.

Database System Architectures

- **Transaction Server System**

  A typical transaction-server system today consists of multiple processes accessing data in shared memory.



Database System Architectures

- **Transaction Server System**

  Server processes: These are processes that receive user queries (transactions), execute them, and send the results back.

  Lock manager process: This process implements lock manager functionality, which includes lock grant, lock release, and deadlock detection.

  Database writer process: There are one or more processes that output modified buffer blocks back to disk on a continuous basis.

Database System Architectures

- **Transaction Server System**

  Log writer process: This process outputs log records from the log record buffer to stable storage.

  Checkpoint process: This process performs periodic checkpoints. It consults log to determine those transactions that need to redone or undone.

  Process monitor process: This process monitors other processes, and if any of them fails, it takes recovery actions for the process.

Database System Architectures

- **Data-server systems**

  - Allow clients to interact with the servers by making requests to read or update data, in units such as files or pages. Data server systems supply raw data to clients. Such systems strive to minimize communication between clients and servers by caching data and locks at the clients. Data servers for database systems offer much more functionality; they support units of data—such as pages, tuples, or objects —that are smaller than a file. They provide indexing facilities for data, and provide transaction facilities so that the data are never left in an inconsistent state if a client machine or process fails. Data-server systems are used in local-area networks, where there is a high-speed connection between the clients and the server.

Database System Architectures

- **Computer System Architecture**

  Computer system architectures consisting of interconnected multiple processors are basically classified into two different categories:

  - Tightly coupled system and
  - Loosely coupled system.

Database System Architectures

- **Tightly Coupled System**

  There is a single system wide global primary memory that is shared by processors connected to the system. If any processor writes some information in the global memory, it can be shared by all other processors. For example, if a processor writes the value 200 to a memory location *y*, any other processors reading from the location *y* will get the value 200. In this system, communication takes place through shared memory.

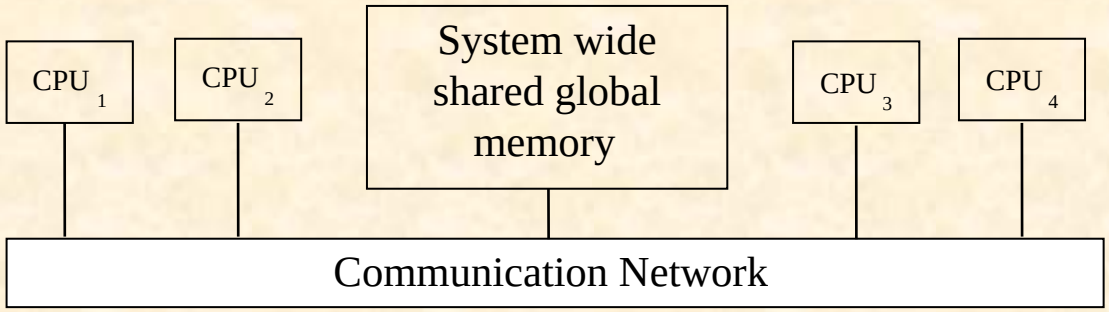


Fig. A Tightly coupled multiprocessor system

Also called parallel processing system.

Database System Architectures

- **Loosely Coupled System**

  Processors don't share memory. Each processor has its own local memory. If a processor writes a value 200 to a memory location *y*, it only changes the content of its own local memory and does not affect the content of the memory of any other processors. In this system, communications are established by passing messages across the network.

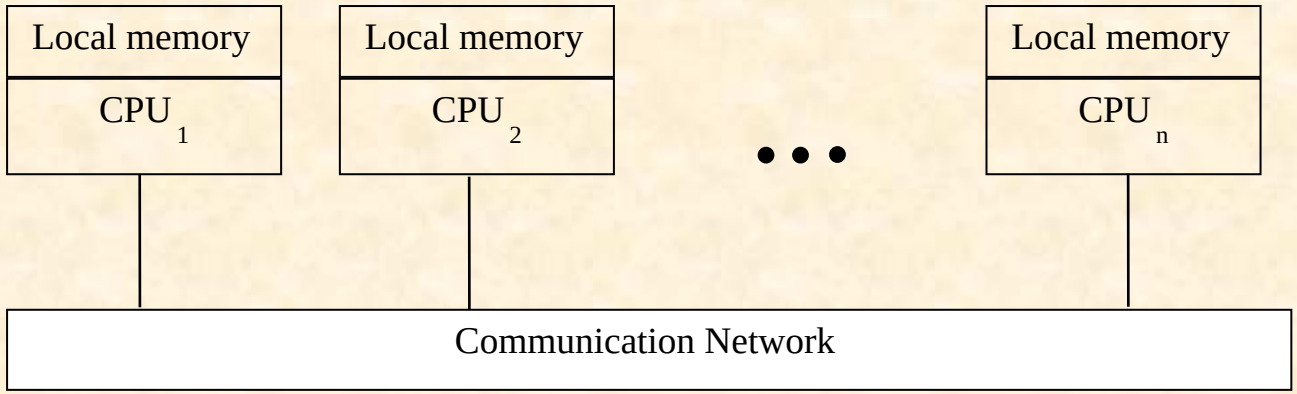


Fig. A Loosely coupled multiprocessor system

Also called distributed system.

Database System Architectures

q Parallel Processing Systems

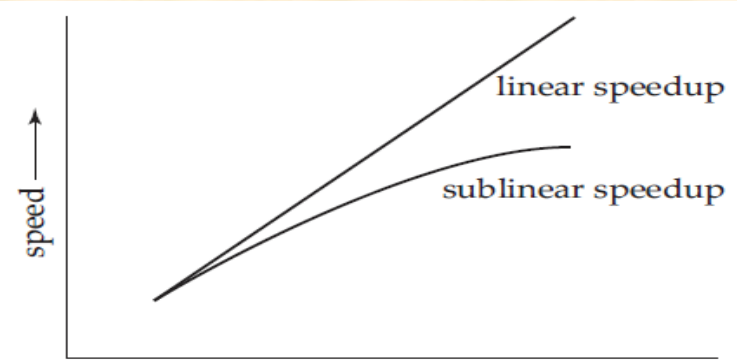
A parallel processing system involves multiple processes that are active simultaneously and solving a given problem, generally on multiple processors. It divides a large task into several subtasks and executes these subtasks concurrently on several processors.

Characteristics

- Parallel systems improve processing and I/O speeds by using multiple CPUs and disks in parallel.
- All processors in the system can perform their tasks concurrently.
- Tasks need to be synchronized.
- Processors usually share resources such as data, disks, and other devices.
- Parallel computers with hundreds of CPUs and disks are available commercially.
- Parallel system improves the system performance in terms of two important properties: **Speedup** and **Scaleup**.

Database System Architectures

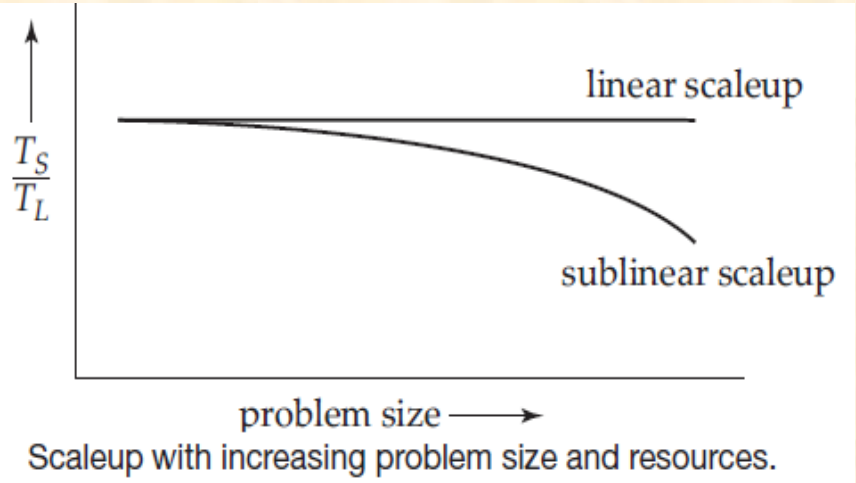
- **Speedup**
  - Running a given task in less time by increasing the degree of parallelism is called speedup.
  - Suppose that the execution time of a task on a larger machine is T_L. The execution time of the same task on a smaller machine is T_S. Thus, **Speedup** = T_S/T_L.
  - If the speedup is N when the larger system has N times the resources (processors, disk, and so on) of the smaller system, the system is said to demonstrate **linear speedup**. If the speedup is less than N, the system is said to demonstrate **sublinear speedup**.



Speedup with increasing resources.

Database System Architectures

- **Scaleup**
  - Handling larger tasks by increasing the degree of parallelism is called scaleup.
  - Let Q be a task, and let Q_N be another task that is N times bigger than Q. Suppose the execution time of task Q on a smaller machine M_S is T_S, and the execution time of task Q_N on a parallel machine M_L is T_L. Thus, scaleup = T_S/T_L
  - The parallel system M_L is said to demonstrate linear scaleup on task Q if $T_L = T_S$. If $T_L > T_S$, the system is said to demonstrate sublinear scaleup



Scaleup with increasing problem size and resources.

Database System Architectures

q **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.

- 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.

Database System Architectures

q **Multiprocessor systems VS DDBS ?**

- Multiprocessor systems are not considered as DDBSs.
- In DDBS, the communication between nodes is done over a network instead of through shared memory or shared disk, with the network as the only shared resource.
- Although shared-nothing multiprocessors, where each processor node has its own primary and secondary memory, and may also have its own peripherals, are quite similar to the distributed environment, there are differences.

Database System Architectures

q **Multiprocessor systems VS DDBS ?**

- A multiprocessor system design is 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.
- Database systems that run over multiprocessor systems are called **parallel database systems.**

Database System Architectures

- **What is Distributed Data Processing or distributed computing system?**

  - A distributed computing system states that 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.

Database System Architectures

q **Distributed Database Management System**

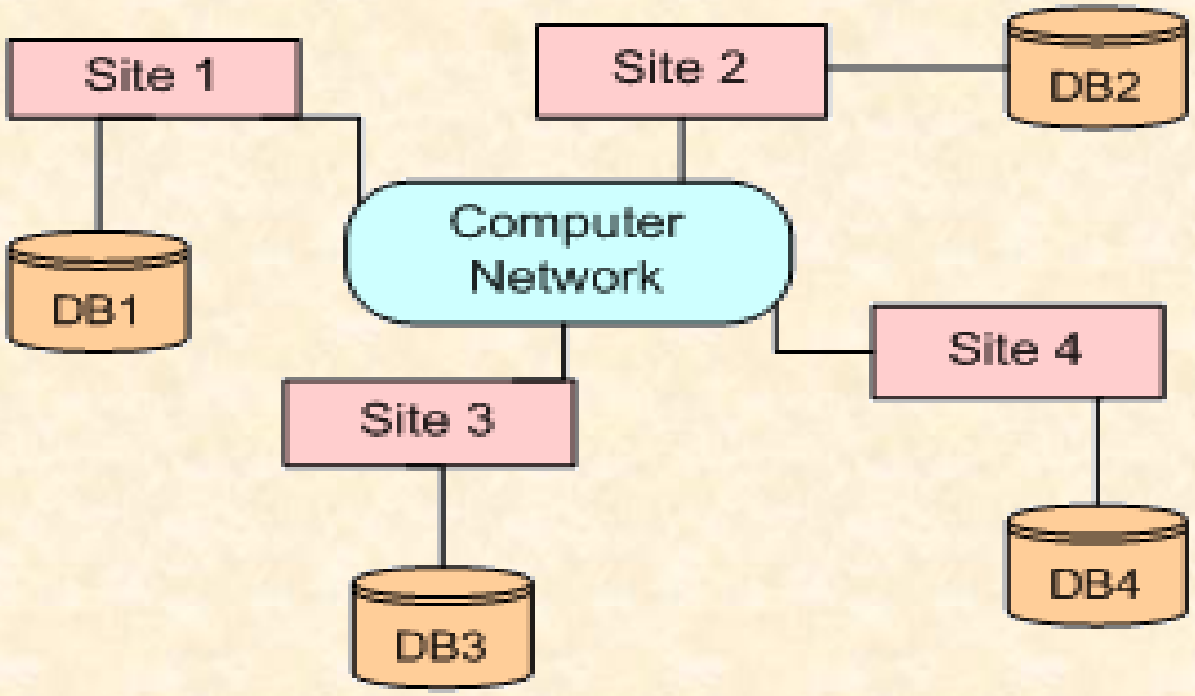
- Consists of a single logical database that is split into a number of fragments. Each fragment is stored on one or more computers.
- The computers in a distributed system communicate with one another through various communication media, such as high-speed networks or telephone lines.
- They do not share main memory or disks.
- Each site is capable of independently processing user requests that require access to local data as well as it is capable of processing user requests that require access to remote data stored on other computers in the network.

The general structure of a distributed system appears in the following figure.

Database System Architectures

- **Distributed Database Management System**

The general structure of a distributed system appears in the following figure.



Distributed Database System

Database System Architectures

q **Distributed Database Management System**

- A **local transaction** is one that accesses data only from sites where the transaction was initiated.
- A **global transaction**, on the other hand, is one that either accesses data in a site different from the one at which the transaction was initiated, or accesses data in several different sites.

Database System Architectures

q **Promises of DDBSs**

Many advantages of DDBSs have been cited in literature, ranging from sociological reasons for decentralization to better economics. All of these can be distilled to four fundamentals which may also be viewed as promises of DDBS technology:

- Transparent management of distributed and replicated data.
- Reliable access to data through distributed transactions
- Improved performance and
- Easier system expansion.

Database System Architectures

- **Promises of DDBSs**

  Transparent Management of Distributed and Replicated Data

  - 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.

Database System Architectures

- **Promises of DDBSs**

  Reliability through Distributed Transactions

  - Distributed DBMSs are intended to improve reliability since they have replicated components and, thereby eliminate single points of failure.

  - The failure of a single site, or the failure of a communication link which makes one or more sites unreachable, is not sufficient to bring down the entire system.

Database System Architectures

q **Promises of DDBSs: Improved Performance**

- A distributed DBMS fragments the conceptual database, enabling data to be stored in close proximity to its points of use (also called **data localization**). This has two potential advantages:
  1. Since each site handles only a portion of the database, contention for CPU and I/O services is not as severe as for centralized databases.
  2. Localization reduces remote access delays.

- 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.

Database System Architectures

- **Promises of DDBSs:**

  Easier System Expansion

  - In a distributed environment, it is much easier to accommodate increasing database sizes.
  - Expansion can usually be handled by adding processing and storage power to the network.
  - Obviously, it may not be possible to obtain a linear increase in "power," since this also depends on the overhead of distribution.

Database System Architectures

- **Complications Introduced by DDBS**

1. Data may be replicated in a distributed environment. A distributed database can be designed so that the entire database, or portions of it, reside at different sites of a computer network.
2. If some sites fail or if some communication links fail while an update is being executed, the system must make sure that the effects will be reflected on the data residing at the failing or unreachable sites as soon as the system can recover from the failure.
3. The exchange of messages and the additional computation required to achieve inter-site coordination are a form of overhead that does not arise in centralized systems.

Database System Architectures

- **Complications Introduced by DDBS**

4. As data in a distributed DBMS are located at multiple sites, the probability of security lapses increases. Further, all communications between different sites in a distributed DBMS are conveyed through the network, so the underlying network has to be made secure to maintain system security.
5. Since each site cannot have instantaneous information on the actions currently being carried out at the other sites, the synchronization of transactions on multiple sites is considerably harder than for a centralized system.

Database System Architectures

q **What is being distributed in DDBS?**

- **Processing logic** or processing elements are distributed. The "processing element" is a computing device that can execute a program on its own.
- **Function.** Various functions of a computer system could be delegated to various pieces of hardware or software.
- **Data.** Data used by a number of applications may be distributed to a number of processing sites.
- Finally, **control** can be distributed. The control of the execution of various tasks might be distributed instead of being performed by one computer system.

Database System Architectures

- **Classification of DDBMS**
  - Homogeneous and
  - Heterogeneous Databases

- **Homogeneous Distributed Database**
  - All sites have identical database management system software, are aware of one another, and agree to cooperate in processing users' requests.
  - Use same DB schemas at all sites.
  - Easier to design and manage
  - Addition of a new site is much easier.

Database System Architectures

- **Heterogeneous distributed database**
  - Usually constructed over a no. of existing sites.
  - Each site has its local database. Different sites may use different schemas (relational model, OO model etc.).
  - Use different DBMS software.
  - Query processing is more difficult.
  - Use gateways (as query translator) which convert the language and data model of each different DBMS into the language and data model of the relational system.

Database System Architectures

- **Data Storage in DDBMS**
  - ✓ **Replication**. The system maintains several identical replicas (copies) of the relation, and stores each replica at a different site.
  - ✓ **Fragmentation**. The system partitions the relation into several fragments, and stores each fragment at a different site.
  - ✓ **Fragmentation and replication can be combined.**

- **Advantages and disadvantages of Replication**
  - ✓ **Availability**.
  - ✓ **Increased parallelism**.
  - ✓ **Increased overhead on update**.

Database System Architectures

- **Data Fragmentation**
  - ✓ If relation *r* is fragmented, *r* is divided into a number of *fragments* $r_1, r_2, \ldots, r_n$.
  - ✓ These fragments contain sufficient information to allow reconstruction of the original relation *r*.
  - ✓ There are two different schemes for fragmenting a relation:

    - **H*orizontal* fragmentation** and
    - **V*ertical* fragmentation**

Database System Architectures

- **Horizontal Fragmentation**
  - In horizontal fragmentation, a relation r is partitioned into a number of subsets, $r_1, r_2, \ldots, r_n$.
  - Each tuple of relation r must belong to at least one of the fragments, so that the original relation can be reconstructed, if needed.

As an illustration, consider the *account* relation:

***Account-schema* = (*account-number*, *branch-name*, *balance*)**

The relation can be divided into several different fragments. If the banking system has only two branches - Savar and Dhanmondi - then there are two different fragments:

$$account_1 = \sigma_{branch\text{-}name = \text{“Savar”}}(account)$$
$$account_2 = \sigma_{branch\text{-}name = \text{“Dhanmondi”}}(account)$$

Database System Architectures

- **Horizontal Fragmentation**
  - Use a predicate P_i to construct fragment r_i:
    $$r_i = \sigma_{Pi}(r)$$
  - Reconstruct the relation r by taking the union of all fragments. That is,
    $$r = r_1 \quad r_2 \quad \cdots \quad r_n$$
  - The fragments are disjoint.

Database System Architectures

- **Vertical Fragmentation**
  - Vertical fragmentation of $r(R)$ involves the definition of several subsets of attributes $R_1, R_2, \ldots, R_n$ of the schema R so that
    $$R = R_1 \quad R_2 \quad \cdots \quad R_n$$
  - Each fragment r_i of r is defined by
    $$r_i = \Pi_{Ri}(r)$$
  - We can reconstruct relation r from the fragments by taking the natural join
    $$r = r_1 \quad r_2 \quad r_3 \quad \cdots \quad r_n$$
  - One way of ensuring that the relation r can be reconstructed is to include the primary-key attributes of R in each of the R_i.

Database System Architectures

q Vertical Fragmentation

- To illustrate vertical fragmentation, consider the following relation:

employee-info=(employee-id, name , designation , salary)

- For privacy reasons, the relation may be fragmented into a relation ***employee- privateinfo*** containing ***employee-id*** and ***salary*** , and another relation ***employee-public-info*** containing attributes ***employee-id*** , ***name*** , and ***designation*** .

employee- privateinfo =(employee-id , salary)
employee-publicinfo =(employee-id , name , designation)

-These may be stored at different sites, again for security reasons.

Database System Architectures

q Correctness Rules for Data Fragmentation

To ensure no loss of information and no redundancy of data, there are three different rules that must be considered during fragmentation.

Completeness

If a relation instance R is decomposed into fragments R_1, R_2, ... R_n, each data item in R must appear in at least one of the fragments. It is necessary in fragmentation to ensure that there is no loss of data during data fragmentation.

Reconstruction

If relation R is decomposed into fragments R_1, R_2, ... R_n, it must be possible to define a relational operation that will reconstruct the relation R from fragments R_1, R_2, ... R_n. This rule ensures that constrains defined on the data are preserved during data fragmentation.

Database System Architectures

q Correctness Rules for Data Fragmentation

To ensure no loss of information and no redundancy of data, there are three different rules that must be considered during fragmentation.

Disjointness

If a relation R is decomposed into fragments R_1, R_2, ... R_n and if a data item is found in the fragment R_i, then it must not appear in any other fragments. **This rule ensures minimal data redundancy.**

In case of vertical fragmentation, primary key attribute must be repeated to allow reconstruction. Therefore, in case of vertical fragmentation, disjointness is defined only on non-primary key attributes of a relation.

Database System Architectures

- **Example**

Let us consider the relational schema Project where project-type represents whether the project is an **inside project** or **abroad project**. Assume that P1 and P2 are two horizontal fragments of the relation Project, which are obtained by using the predicate "whether the value of project-type attribute is 'inside' or 'abroad'.

Project

Project-id	Project-name	Project-type	Project-leader-id	Branch-no	Amount
P01	Inventory	Inside	E001	B10	$1000000
P02	Sales	Inside	E001	B20	$300000
P03	R&D	Abroad	E004	B70	$8000000
P04	Educational	Inside	E003	B20	$400000
P05	Health	Abroad	E005	B60	$7000000

Database System Architectures

- **Example (Horizontal Fragmentation)**

P1: $\sigma_{project\text{-}type = \text{“inside”}}$ **(Project)**

P2: $\sigma_{project\text{-}type = \text{“abroad”}}$ **(Project)**

P1

Project-id	Project-name	Project-type	Project-leader-id	Branch-no	Amount
P01	Inventory	Inside	E001	B10	$1000000
P02	Sales	Inside	E001	B20	$300000
P04	Educational	Inside	E003	B20	$400000

P2

Project-id	Project-name	Project-type	Project-leader-id	Branch-no	Amount
P03	R&D	Abroad	E004	B70	$8000000
P05	Health	Abroad	E005	B60	$7000000

Database System Architectures

- **Example (Horizontal Fragmentation)**

These horizontal fragments satisfy all the correctness rules of fragmentation as shown below.

Completeness: Each tuple in the relation Project appears either in fragment P1 or P2. Thus, it satisfies completeness rule for fragmentation.

Reconstruction: The Project relation can be reconstructed from the horizontal fragments P1 and P2 by using the union operation of relational algebra, which ensures the reconstruction rule.

Thus, P1 P2 = Project.

Disjointness: The fragments P1 and P2 are disjoint, since there can be no such project whose project type is both "inside" and "abroad".

Database System Architectures

- **Example (Vertical Fragmentation)**

In this case, the Project relation is partitioned into two vertical fragments V1 and V2, which are described below

$$V1: \Pi_{\text{Project-id, branch-no, project-leader-id}}(\text{Project})$$

$$V2: \Pi_{\text{Project-id, Project-name, Project-type, amount}}(\text{Project}).$$

V1

Project-id	Project-leader-id	Branch-no
P01	E001	B10
P02	E001	B20
P03	E004	B70
P04	E003	B20
P05	E005	B60

V2

Project-id	Project-name	Project-type	Amount
P01	Inventory	Inside	$1000000
P02	Sales	Inside	$300000
P03	R&D	Abroad	$8000000
P04	Educational	Inside	$400000
P05	Health	Abroad	$7000000

Database System Architectures

- **Example (Vertical Fragmentation)**

These vertical fragments also ensure the correctness rules of fragmentation as shown below.

Completeness: Each tuple in the relation Project appears either in fragment V1 or V2 which satisfies completeness rule for fragmentation.

Reconstruction: The Project relation can be reconstructed from the vertical fragments V1 and V2 by using the natural join operation of relational algebra, which ensures the reconstruction rule.

Thus, V1 V2 = Project.

Disjointness: The fragments V1 and V2 are disjoint, except for the primary key project-id, which is repeated in both fragments and is necessary for reconstruction.

Database System Architectures

- **Distributed Database System Design Issues**
  - Distributed Database Design
  - Distributed Directory Management
  - Distributed Query Processing
  - Distributed Concurrency Control
  - Distributed Deadlock Management
  - Reliability of Distributed DBMS
  - Replication
  - Relationship among Problems

Database System Architectures

q **Distributed Database Design**

There are two basic alternatives to placing data:

- Partitioned (or non-replicated) and
- Replicated

- In the partitioned scheme the database is divided into a number of disjoint partitions each of which is placed at a different site.
- Replicated designs can be either fully replicated (also called fully duplicated) where the entire database is stored at each site, or partially replicated (or partially duplicated) where each partition of the database is stored at more than one site, but not at all the sites.

Database System Architectures

q **Distributed Directory Management**

- A directory contains information (such as descriptions and locations) about data items in the database.
- A directory may be global to the entire DDBS or local to each site;
- It can be centralized at one site or distributed over several sites; there can be a single copy or multiple copies.

Database System Architectures

q **Distributed Query Processing**

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

Database System Architectures

- **Distributed Concurrency Control**

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

- **Distributed Deadlock Management**

  - 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.

Database System Architectures

q **Reliability of Distributed DBMS**

- One of the potential advantages of distributed systems is improved reliability and availability. This, however, is not a feature that comes automatically.
- It is important that mechanisms be provided to ensure the consistency of the database as well as to detect failures and recover from them.
- When a failure occurs and various sites become either inoperable or inaccessible, the databases at the operational sites remain consistent and up to date.

Database System Architectures

- **Replication**

  - If the distributed database is (partially or fully) replicated, it is necessary to implement protocols that ensure the consistency of the replicas,

- **Relationship among Problems**

  - The problems are not isolated from one another. Each problem is affected by the solutions found for the others.
  - The relationship among the components is shown in the following Figure. 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 as well as the strategies that may be employed to manage them.

Database System Architectures

q **Relationship among Problems**

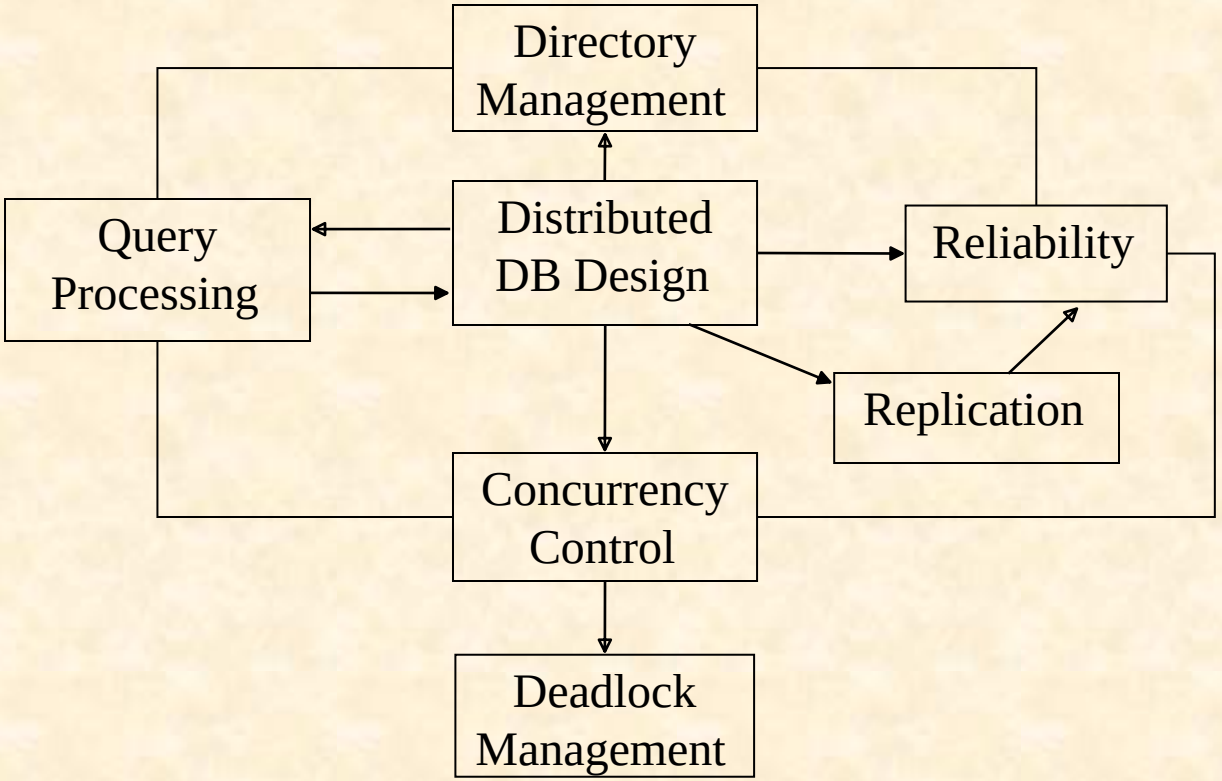


Fig.: Relationship among Research Issues

Database System Architectures

q **Relationship among Problems**

- The same information (i.e., fragment structure and placement) is used by the query processor to determine the query evaluation strategy. Similarly, directory placement and contents influence the processing of queries.

- The replication of fragments when they are distributed affects the concurrency control strategies that might be employed.

- Some concurrency control algorithms cannot be easily used with replicated databases.

Database System Architectures

q **Relationship among Problems**

- There is a strong relationship among the concurrency control problem, the deadlock management problem, and reliability issues. The concurrency control algorithm that is employed will determine whether or not a separate deadlock management facility is required. If a locking-based algorithm is used, deadlocks will occur, whereas they will not if timestamping is the chosen alternative.
- Reliability mechanisms involve both local recovery techniques and distributed reliability protocols. Techniques to provide reliability also make use of data placement information since the existence of duplicate copies of the data serve as a safeguard to maintain reliable operation.

Database System Architectures

- **Relationship among Problems**

  - Finally, the need for replication protocols arise if data distribution involves replicas.

Database System Architectures

- **Distributed Database System**
  - Each site has autonomous processing capability and can perform local applications.
  - Each site also participates in the execution of at least one global application which requires accessing data at several sites.

Database System Architectures

q **Distributed DBMS Architecture**

- The architecture of a system defines its structure. This means that the components of the system are identified, the function of each component is specified, and the interrelationships and interactions among these components are defined.

- Three reference architectures for a distributed DBMS:
  1. Client/Server systems
  2. Peer-to-Peer distributed DBMS and
  3. Multidatabase systems.

- A reference architecture is commonly created by standards developers to clearly define the interfaces that need to be standardized.

Database System Architectures

q **ANSI/SPARC Architecture**

A simplified version of the ANSI/SPARC architecture is depicted in the following Figure.

There are three views of data:

- **External view**, which is that of the end user,
- **Internal view**, that of the system or machine
- **Conceptual view**, that of the enterprise. For each of these views, an appropriate schema definition is required.

Database System Architectures

- **ANSI/SPARC Architecture**

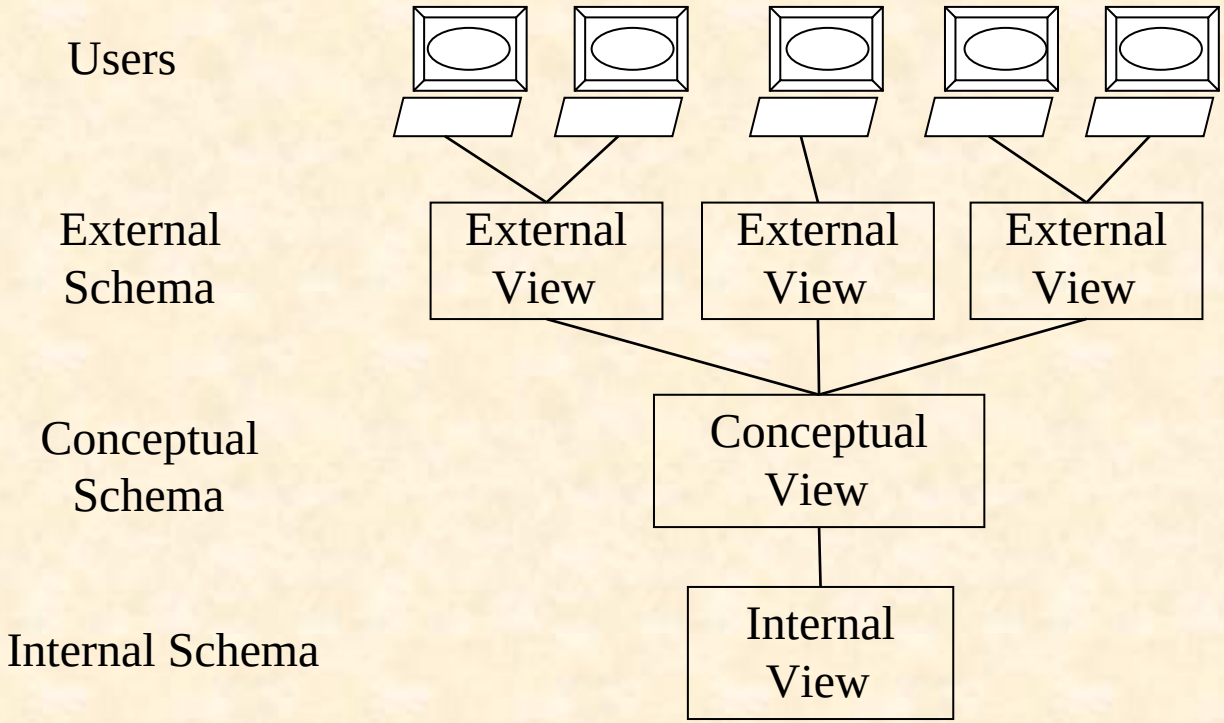


Fig.: The ANSI/SPARC Architecture

Database System Architectures

q **ANSI/SPARC Architecture**

- At the lowest level of the architecture is the **internal view**, which deals with the physical definition and organization of data. The location of data on different storage devices and the **access mechanisms used to reach and manipulate data** are the issues dealt with at this level.

- At the other extreme is the **external view**, which is concerned with **how users view the database**. An individual user's view represents the portion of the database that will be accessed by that user. A view can be shared among a number of users, with the collection of user views making up the external schema.

Database System Architectures

- **ANSI/SPARC Architecture**

  - In between these two ends is the **conceptual schema**, which is an abstract **definition** of the database.

  - American National Standards Institute (ANSI) established a Study Group on Database Management Systems under the auspices of its Standards Planning and Requirements Committee (SPARC).

Database System Architectures

- **Distributed DBMS Architecture**

  - The client/server distribution concentrates data management duties at servers while the clients focus on providing the application environment including the user interface.

  - In peer-to-peer distribution, there is no distinction of client machines versus servers. Each machine has full DBMS functionality and can communicate with other machines to execute queries and transactions.

Database System Architectures

q **Peer-to-Peer Distributed Systems**

➢The physical data organization on each machine is different. There needs to be an individual internal schema definition at each site (**Local Internal Schema (LIS)**).

➢The enterprise view of the data is described by the **Global Conceptual Schema (GCS)**- it describes the logical structure of the data at all the sites.

➢Data in a distributed database is usually fragmented and replicated.

Database System Architectures

- **Peer-to-Peer Distributed Systems**

  - To handle 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).**

  - The GCS is the union of the LCSs. User applications and user access to the database is supported by **External Schemas (ESs)**

Database System Architectures

- **Peer-to-Peer Distributed Systems**

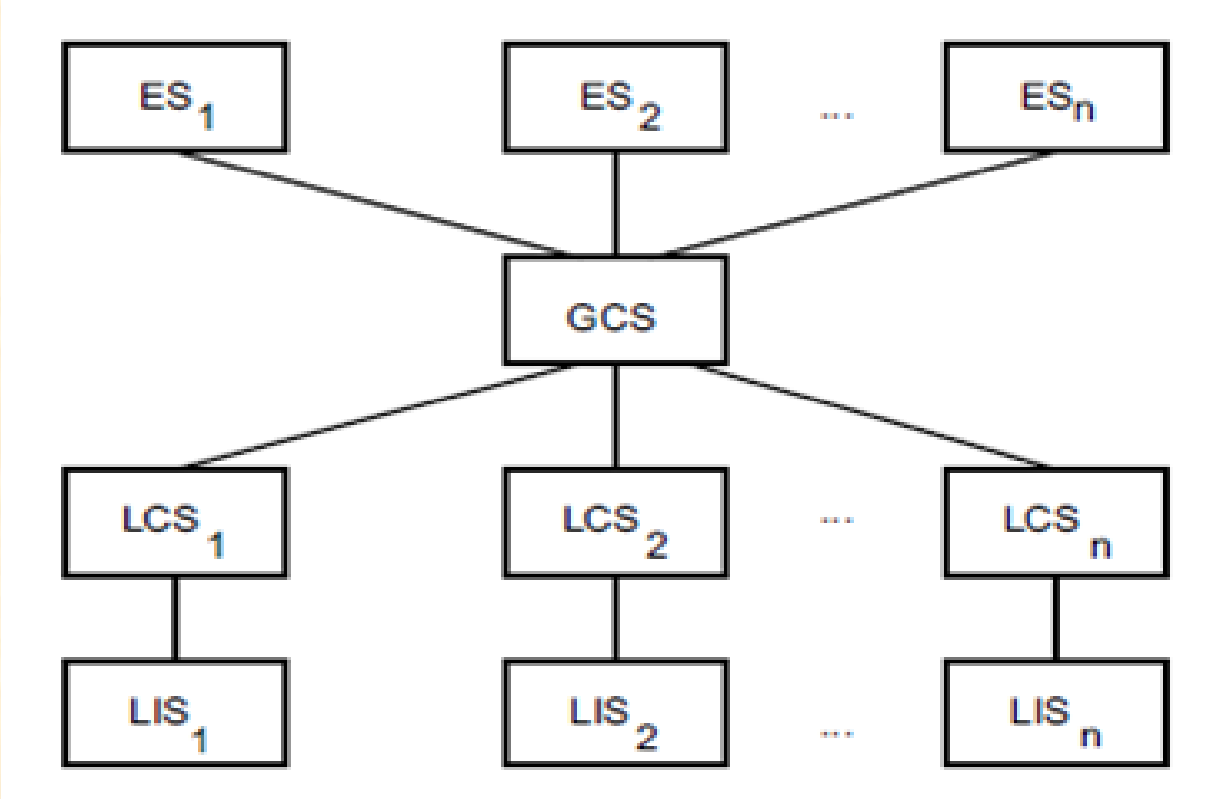


Fig.: Distributed Database Reference Architecture

Database System Architectures

- **Peer-to-Peer Distributed Systems**

➢The distributed DBMS translates global queries into a group of local queries, which are executed by distributed DBMS components at different sites that communicate one another.

Database System Architectures

- **Components of a Distributed DBMS**

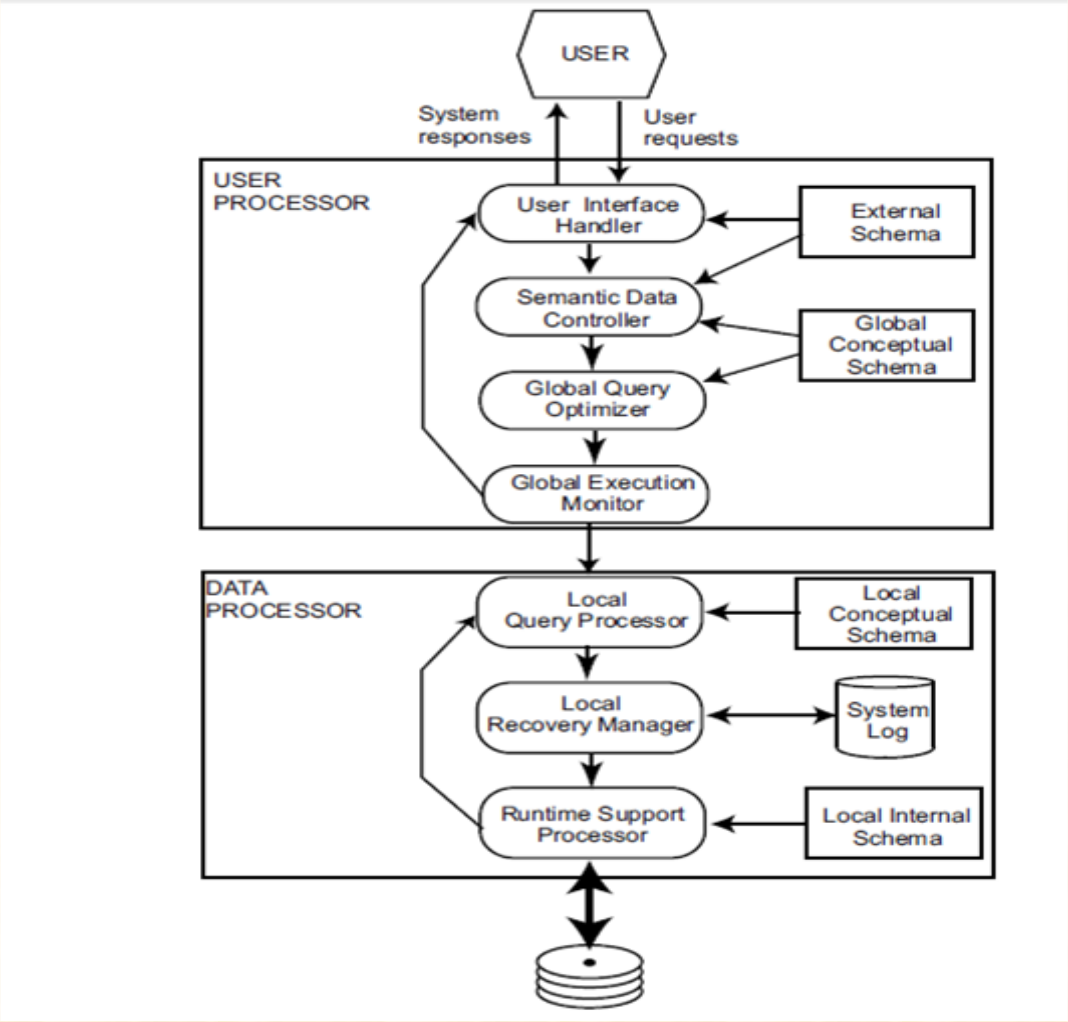
  Major two components:

  - User Processor:
    Handles the interaction with users and

  - Data Processor:
    Deals with the storage.

Database System Architectures

- **Components of a Distributed DBMS**



Database System Architectures

- **Components of a Distributed DBMS**

The first component User Processor consists of four elements:

i) User interface handler
ii) Semantic data controller
iii) Global query optimizer and decomposer
iv) Distributed execution monitor

1. The user interface handler is responsible for interpreting user commands as they come in, and formatting the result data as it is sent to the user.
2. The semantic data controller uses the integrity constraints and authorizations that are defined as part of the global conceptual schema to check if the user query can be processed.

Database System Architectures

- **Components of a Distributed DBMS**

3. The global query optimizer and decomposer determines an execution strategy to minimize a cost function, and translates the global queries into local ones using the global and local conceptual schemas. The global query optimizer is responsible, among other things, for generating the best strategy to execute distributed join operations.
4. The distributed execution monitor coordinates the distributed execution of the user request. The execution monitor is also called the distributed transaction manager. In executing queries in a distributed fashion, the execution monitors at various sites may, and usually do, communicate with one another.

Database System Architectures

q **Components of a Distributed DBMS**

The second component Data Processor consists of three elements:

i) Local query optimizer
ii) Local recovery manager
iii) Run-time support processor

In peer-to-peer systems, one expects to find both the user processor modules and the data processor modules on each machine.

1. The local query optimizer, which actually acts as the access path selector, is responsible for choosing the best access path to access any data item.

Database System Architectures

q **Components of a Distributed DBMS**

2. The local recovery manager is responsible for making sure that the local database remains consistent even when failures occur.

3. The run-time support processor physically accesses the database according to the physical commands in the schedule generated by the query optimizer. The run-time support processor is the interface to the operating system and contains the database buffer (or cache) manager, which is responsible for maintaining the main memory buffers and managing the data accesses.

Database System Architectures

q **Multidatabase System (MDBS) Architecture**

- Multidatabase systems (MDBS) represent the case where individual DBMSs (whether distributed or not) are fully autonomous and have no concept of cooperation;

- They may not even "know" of each other's existence or how to talk to each other.

Database System Architectures

q **Distributed multi-DBMSs VS distributed DBMSs**

- The fundamental difference relates to the definition of the global conceptual schema. In the case of logically integrated distributed DBMSs, the global conceptual schema defines the conceptual view of the entire database, while in the case of distributed multi-DBMSs, it represents only the collection of some of the local databases that each local DBMS wants to share.

- The individual DBMSs may choose to make some of their data available for access by others by defining an export schema.

Database System Architectures

q **Distributed multi-DBMSs VS distributed DBMSs**

- In MDBSs, the global database is equal to the union of local databases, whereas in distributed DBMSs it is only a subset of the same union.
- In a MDBS, the GCS is defined by integrating either the external schemas of local autonomous databases or local conceptual schemas.

Database System Architectures

- **MDBS Architecture**

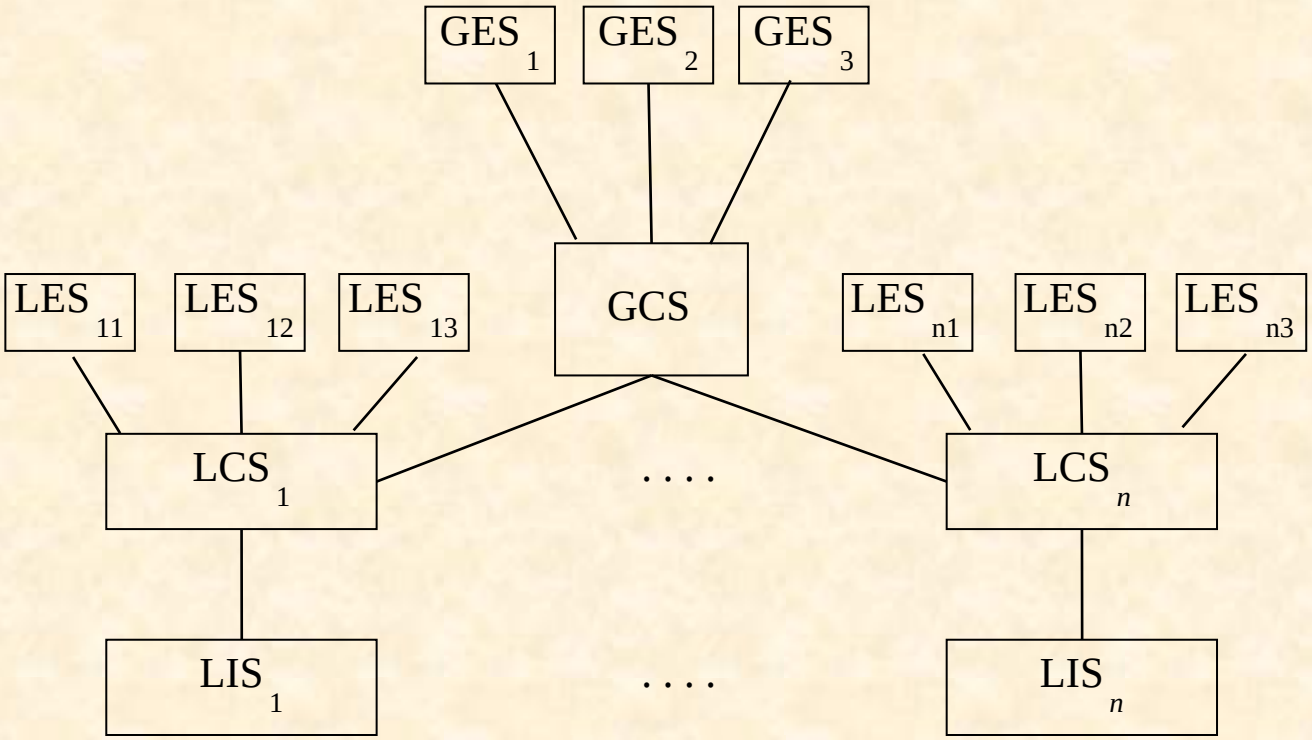


Fig.: MDBS Architecture with a GCS

Database System Architectures

q **MDBS Architecture**

- Users of a local DBMS define their own views on the local database and do not need to change their applications if they do not want to access data from another database. This is again an issue of autonomy.
- Designing the global conceptual schema in multidatabase systems involves the integration of either the local conceptual schemas or the local external schemas.
- Once the GCS has been designed, views over the global schema can be defined for users who require global access. It is not necessary for the GES and GCS to be defined using the same data model and language; whether they do or not determines whether the system is homogeneous or heterogeneous.

Database System Architectures

- **MDBS Architecture**

LIS Local Internal Schema: Physical data organization and techniques to manipulate data at different sites.

LCS Local Conceptual Schema: data definition at different sites.

LES Local External Schema: Local user data view.

GCS Global Conceptual Schema: Union of LCSs who want to share data. Data definition for global users.

GES Global External Schema: Global user data view.

Database System Architectures

- **Component-based architectural model of a multi-DBMS**

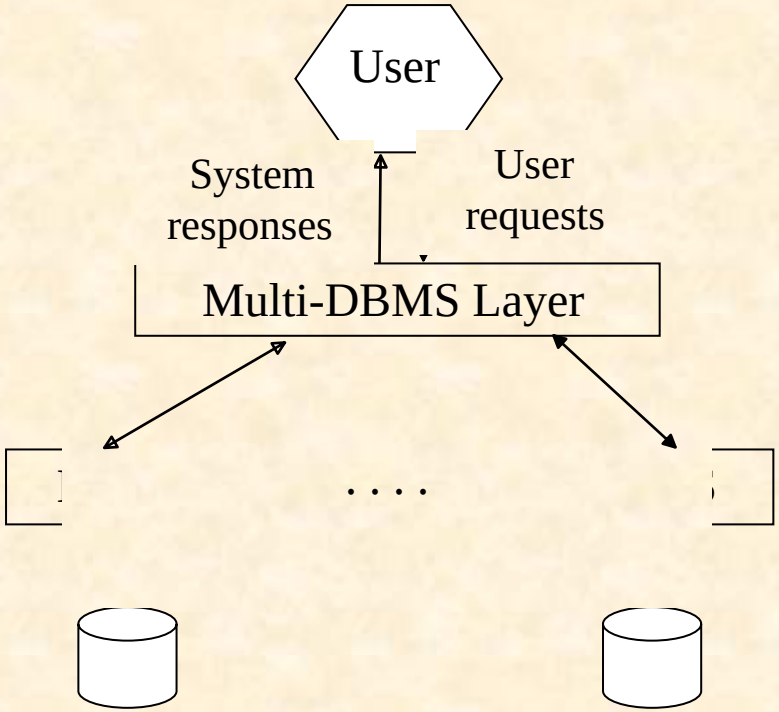


Fig.: Components of an MDBS

Database System Architectures

q **Component-based architectural model of a multi-DBS**

- MDBS provides a layer of software that runs on top of these individual DBMSs and provides users with the facilities of accessing various databases.
- In a distributed MDBS, the multi-DBMS layer may run on multiple sites or there may be central site where those services are offered.
- The MDBS layer is simply another application that submits requests and receives answers.

Database System Architectures

Thanks