# Unit IV – ADVANCE CONCEPTS OF DATABASES

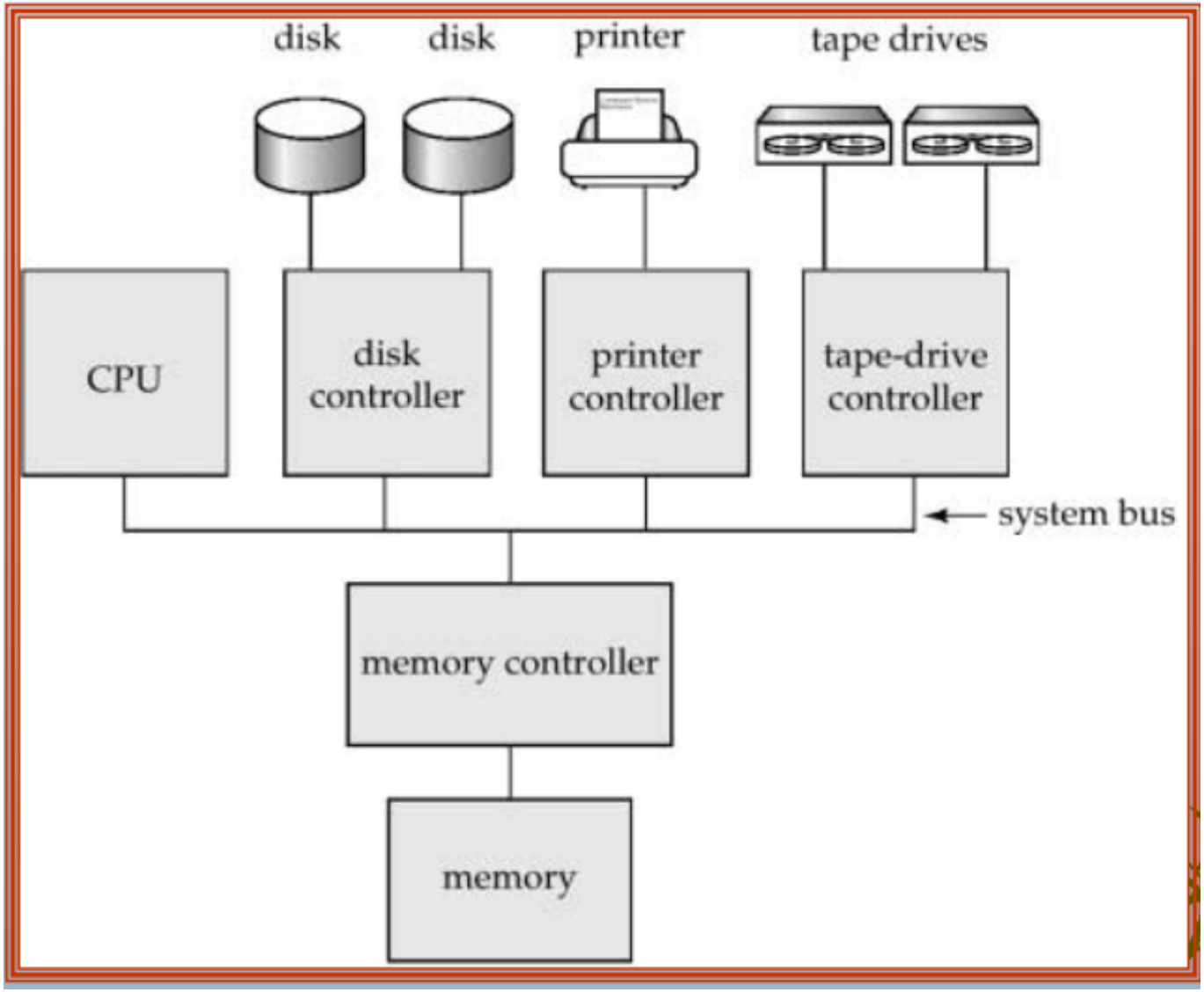
Prof. Sonali Deshpande

- Database Architectures: Centralized and ClientServer Architectures, Database Connectivity using Java/Python /php with SQL and NoSQL databases.
- Introduction to Parallel Databases, Architecture of Parallel Databases. Introduction to Distributed Databases, Distributed Transactions.
- 2PC, 3PC protocols, Introduction to Data Mining and clustering.

# Centralized Systems

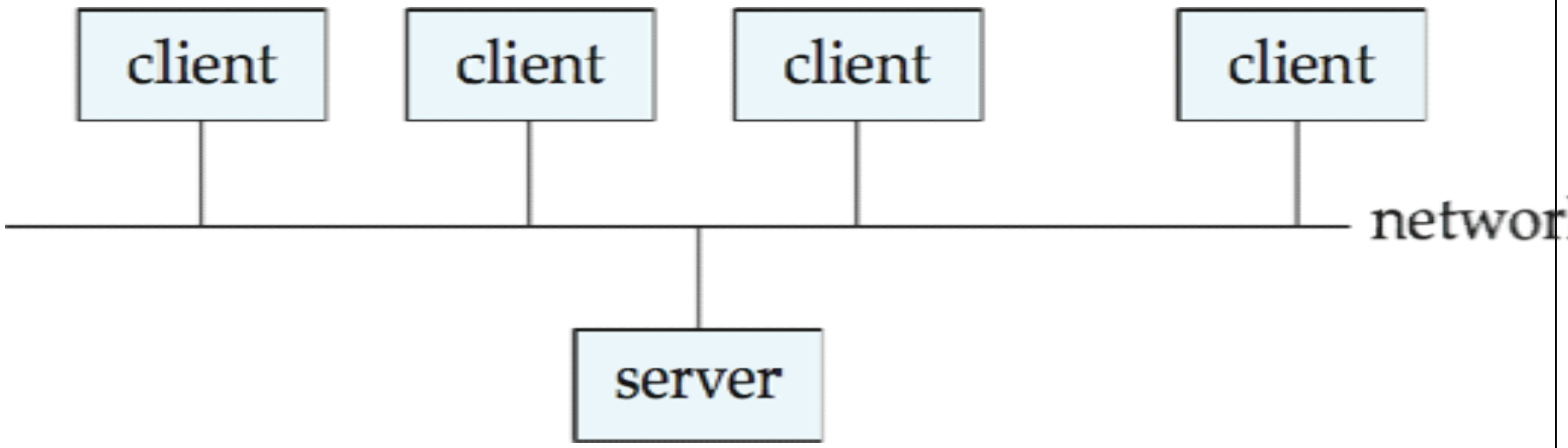
- Run on a single computer system and do not interact with other computer systems.
- **General-purpose computer system**: one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory.
- **Single-user system** (e.g., personal computer or workstation): desktop unit, single user, usually has only one CPU and one or two hard disks; the OS may support only one user.
- **Multi-user system**: more disks, more memory, multiple CPUs, and a multi- user OS. Serve a large number of users who are connected to the system via terminals. Often called server systems.

# A Centralized Computer System



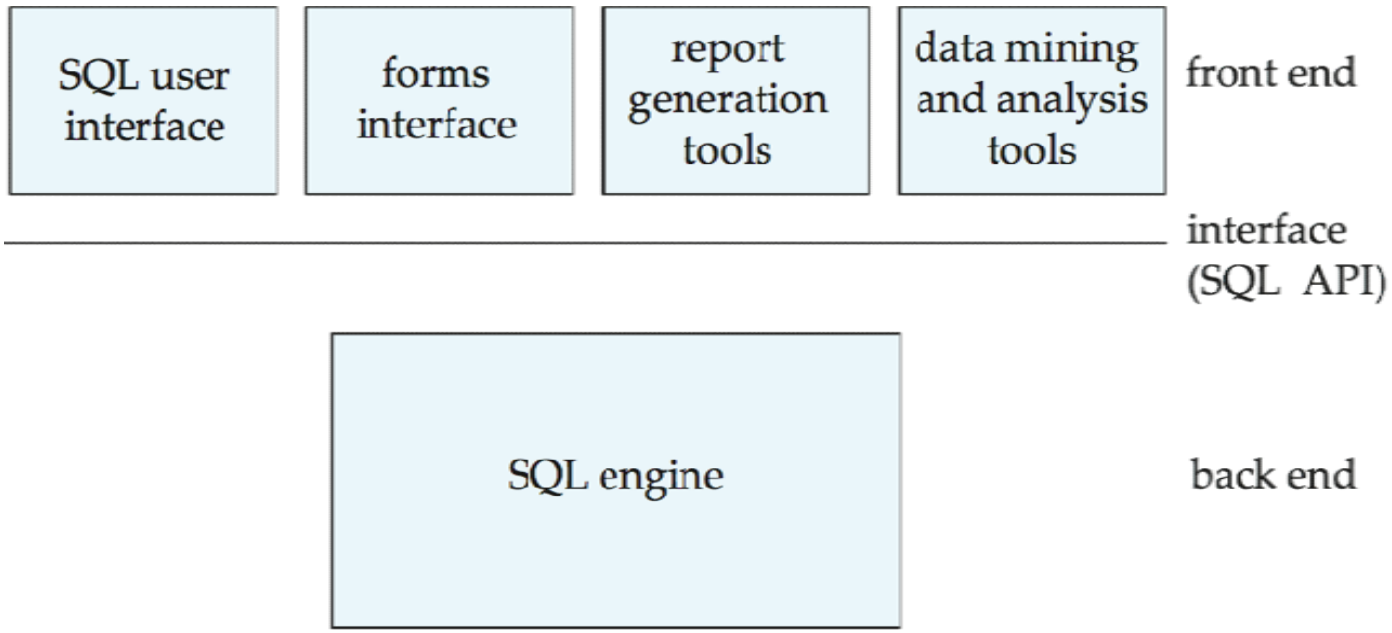
# Client-Server Systems

- Server systems satisfy requests generated at m client systems, whose general structure is shown below:



# Client-Server Systems

- Database functionality can be divided into:
- Back-end: manages access structures, query evaluation and optimization, concurrency control and recovery.
- 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 interface.



# Client-Server Systems

- Advantages of replacing mainframes with networks of workstations or personal computers connected to back-end server machines:
- better functionality for the cost
- flexibility in locating resources and expanding facilities
- better user interfaces
- easier maintenance

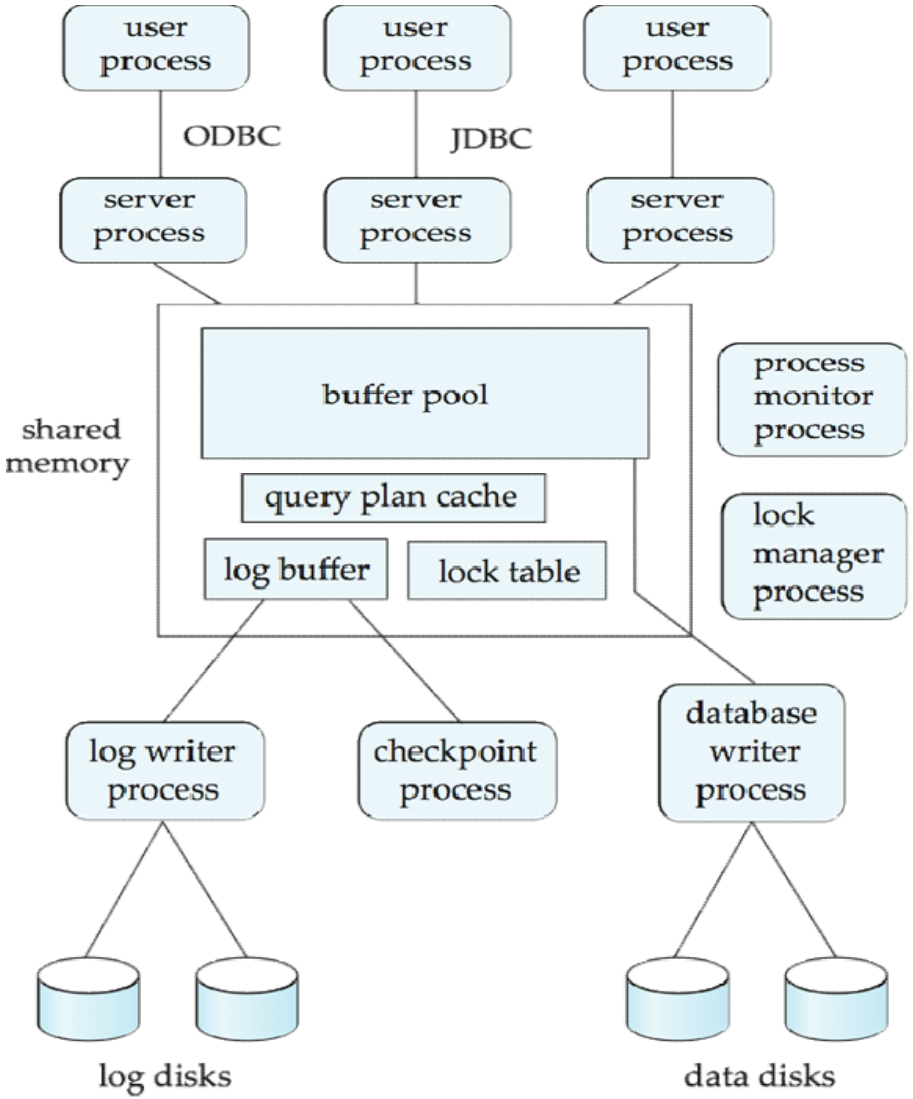
Server System Architecture

- Server systems can be broadly categorized into two kinds:
- **transaction servers** which are widely used in relational database systems
- **data servers**, used in object-oriented database systems

# Transaction Servers

- Also called query server systems or SQL server systems.
- Clients send requests to the server
- Transactions are executed at the server and results are shipped back to the client.
- Requests are specified in SQL, and communicated to the server through a remote procedure call (RPC) mechanism.
- Transactional RPC allows many RPC calls to form a transaction.
- Open Database Connectivity (ODBC) is a C language application program interface standard from Microsoft for connecting to a server, sending SQL requests, and receiving results.
- JDBC standard is similar to ODBC, for Java

# Transaction System Processes



# Transaction Server Process Structure

- A typical transaction server consists of multiple processes accessing data in shared memory.
- Server processes
  - These receive user queries (transactions), execute them and send results back
  - Processes may be multithreaded, allowing a single process to execute several user queries concurrently
- **Lock manager process** - The Lock Manager manages locks for both shared data and for internal system resources. For shared data, the Lock Manager manages row locks, page locks, and table locks on tables, as well as data pages, text pages, and leaf level index pages.
- **Database writer process** - writes the contents of buffers to data files. The DBW processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk. Although one database writer process (DBW) is adequate for most systems, you can configure additional processes to improve write performance if your system modifies data heavily.

# Transaction Server Processes

- Log writer process
  - Server process simply add log records to log record buffer
  - Log writer process outputs log records to stable storage.
- Checkpoint process
  - Performs periodic checkpoints
- Process monitor process
  - Monitors other processes, and takes recovery actions if any of the other processes fail
  - E.g. aborting any transactions being executed by a server process and restarting it

- Shared memory contains shared data
  - Buffer pool
  - Lock table
  - Log buffer
  - Cached query plans (reused if same query submitted again)
- All database processes can access shared memory
- To ensure that no two processes are accessing the same data structure at the same time, databases systems implement mutual exclusion using either
  - Operating system semaphores
  - Atomic instructions such as test-and-set
- To avoid overhead of interprocess communication for lock request/grant, each database process operates directly on the lock table instead of sending requests to lock manager process
- Lock manager process still used for deadlock detection

# Data Servers

- Data-server systems are used in local-area networks, where there is a high-speed connection between the clients and the server.
- the client machines are comparable in processing power to the server machine the tasks to be executed are computation intensive.
- In such an environment, it makes sense to ship data to client machines, to perform all processing at the client machine and then to ship the data back to the server machine.
- Interesting issues arise in such an architecture are
- **Page shipping-** can be considered a form of prefetching if multiple items reside on a page, since all the items in the page are shipped when a process desires to access a single item in the page.

# Data Servers

- **Locking-** Locks are usually granted by the server for the data items that it ships to the client machines.
- A disadvantage of page shipping is , if the client is not accessing some items in the page, it has implicitly acquired locks on all pre-fetched items, Other client machines that require locks on those items may be blocked unnecessarily.
- If the client machine does not need a prefetched item, it can transfer locks on the item back to the server, and the locks can then be allocated to other clients.
- **Data caching-** Data that are shipped to a client on behalf of a transaction can be cached at the client, even after the transaction completes, if sufficient storage space is available.
- Successive transactions at the same client may be able to make use of the cached data.
- Even if a transaction finds cached data, it must make sure that those data are up to date

- **Lock caching**-Suppose that a client finds a data item in the cache, and that it also finds the lock required for an access to the data item in the cache. Then, the access can proceed without any communication with the server.
- However, the server must keep track of cached locks; if a client requests a lock from the server, the server must call back all conflicting locks on the data item from any other client machines that have cached the locks.

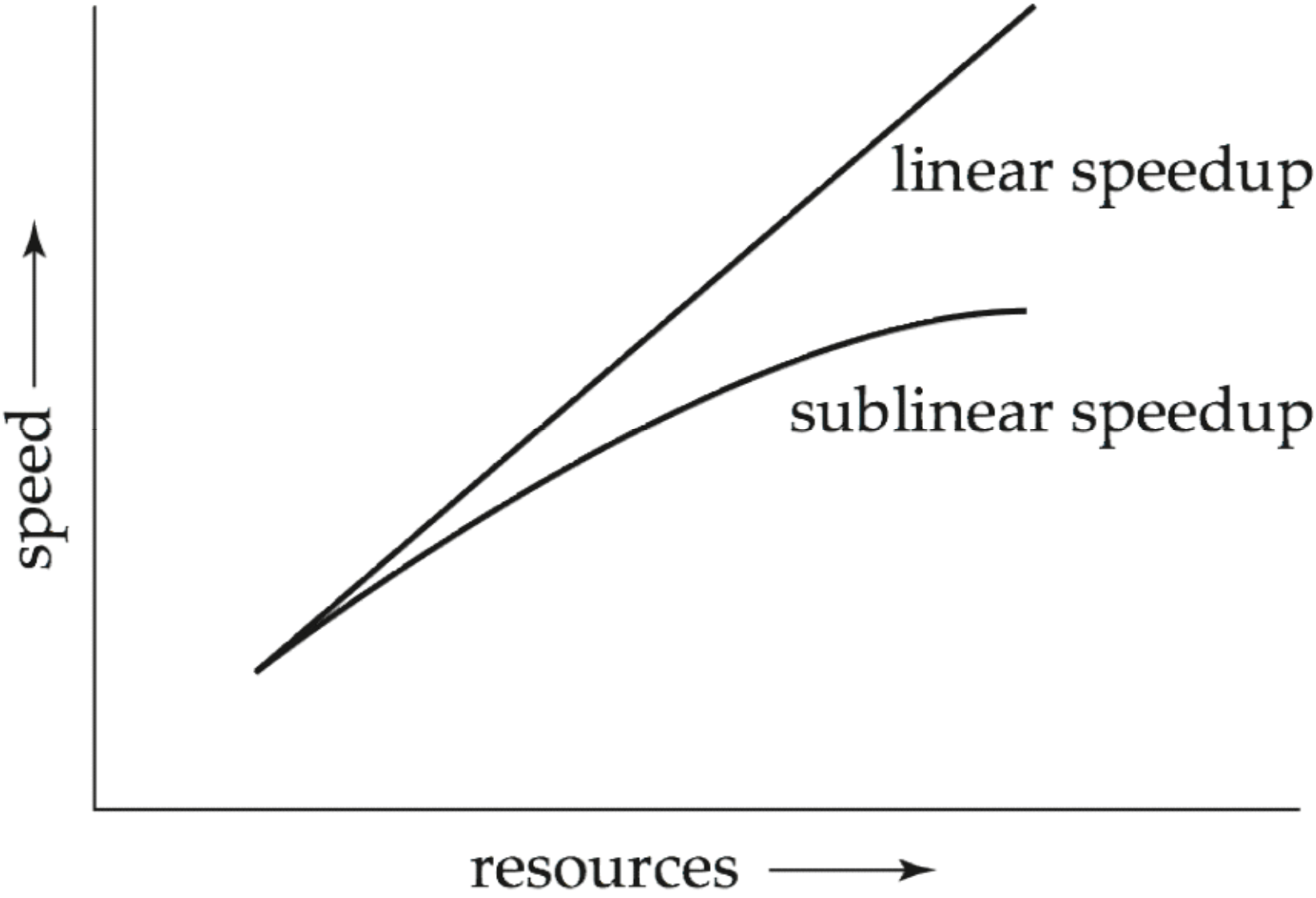
# Parallel Systems

- Parallel systems improve processing and I/O speeds by using multiple CPUs and disks in parallel.
- The driving force behind parallel database systems is the demands of applications that have to query extremely large databases or that have to process an extremely large number of transactions per second.
- Centralized and client–server database systems are not powerful enough to handle such applications.
- Two main performance measures:
  - throughput --- the number of tasks that can be completed in a given time interval
  - response time --- the amount of time it takes to complete a single task from the time it is submitted

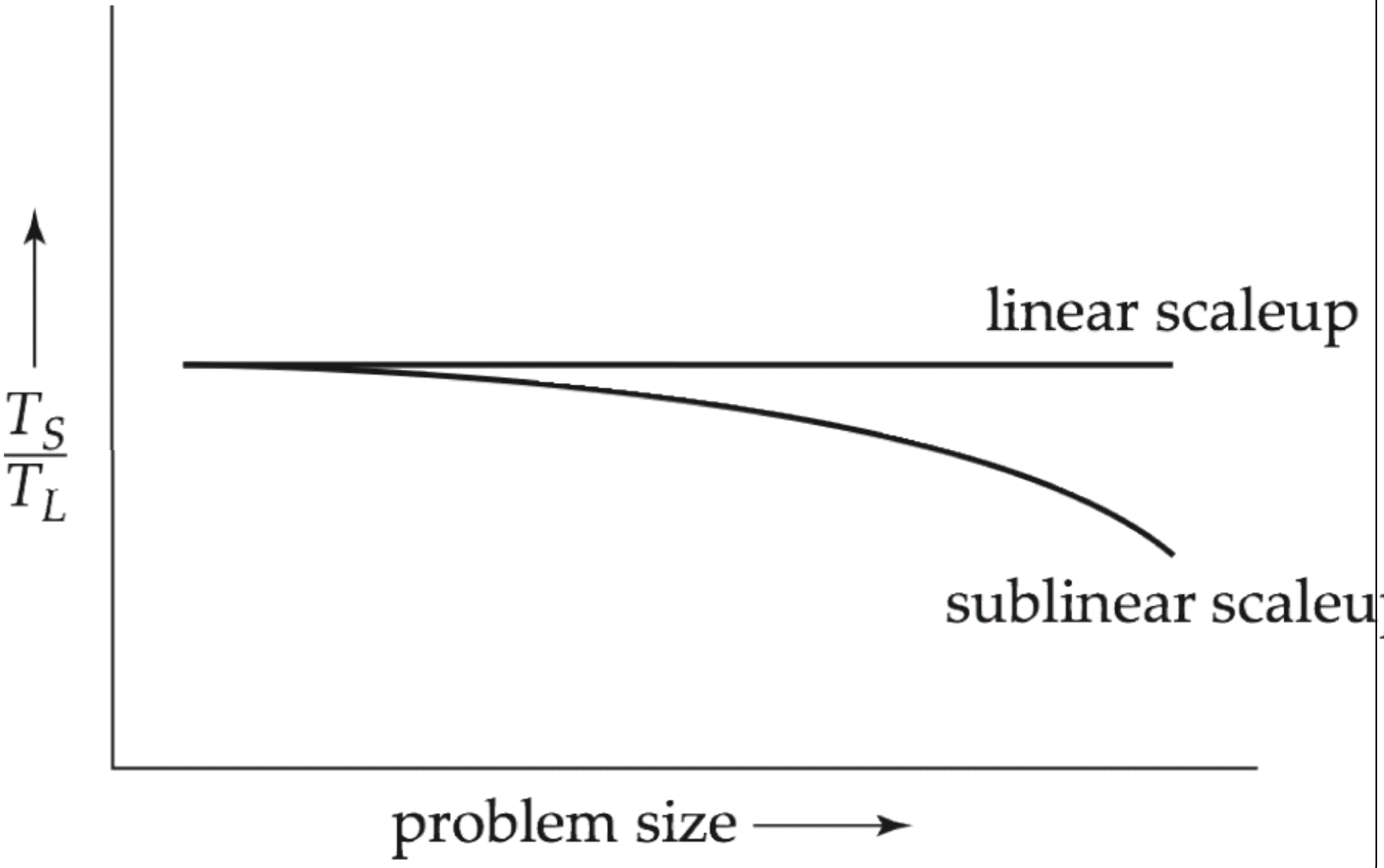
# Speed-Up and Scale-Up

- Two important issues in studying parallelism are speedup and scaleup.
- Running a given task in less time by increasing the degree of parallelism is called **speedup.**
- Handling larger tasks by increasing the degree of parallelism is called **scaleup.**
- Consider a database application running on a parallel system with a certain number of processors and disks.
- suppose that we increase the size of the system by increasing the number or processors, disks, and other components of the system.
- The goal is to process the task in time inversely proportional to the number of processors and disks allocated.
- Suppose that the execution time of a task on the larger machine is TL, and that the execution time of the same task on the smaller machine is TS.

# Speedup



# Scaleup



# Batch and Transaction Scaleup

- There are two kinds of scaleup that are relevant in parallel database systems, depending on how the size of the task is measured:
- Batch scaleup: In batch scaleup, the size of the database increases, and the tasks are large jobs whose runtime depends on the size of the database.
- Transaction scaleup: In transaction scaleup, the rate at which transactions are submitted to the database increases and the size of the database increases proportionally to the transaction rate.
- for example, a deposit or withdrawal from an account and transaction rates grow as more accounts are created. Such transaction processing is especially well adapted for parallel execution, since transactions can run concurrently and independently on separate processors, and each transaction takes roughly the same amount of time, even if the database grows.

# Factors Limiting Speedup and Scaleup

Speedup and scaleup are often sublinear due to:

- Startup costs: Cost of starting up multiple processes may dominate computation time, if the degree of parallelism is high.
- Interference: Since processes executing in a parallel system often access shared resources, a slowdown may result from the interference of each new process as it competes with existing processes for commonly held resources
- Skew: By breaking down a single task into a number of parallel steps, we reduce the size of the average step.
- For example, if a task of size 100 is divided into 10 parts, and the division is skewed, there may be some tasks of size less than 10 and some tasks of size more than 10; if even one task happens to be of size 20, the speedup obtained by running the tasks in parallel is only five, instead of ten as we would have hoped.

# Interconnection Network Architectures

- Parallel systems consist of a set of components (processors, memory, and disks) that can communicate with each other via an interconnection network.

- Bus. System components send data on and receive data from a single communication bus.

  Does not scale well with increasing parallelism, since the bus can handle communication from only one component at a time.

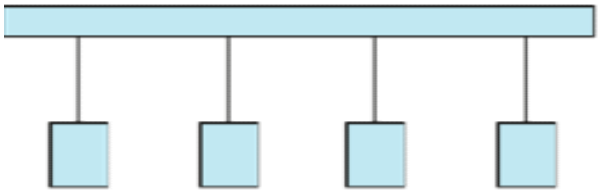
- Mesh. Components are arranged as nodes in a grid, and each component is connected to all adjacent components.

  In a two-dimensional mesh each node connects to four adjacent nodes, while in a three-dimensional mesh each node connects to six adjacent nodes. Figure b shows a two-dimensional mesh.

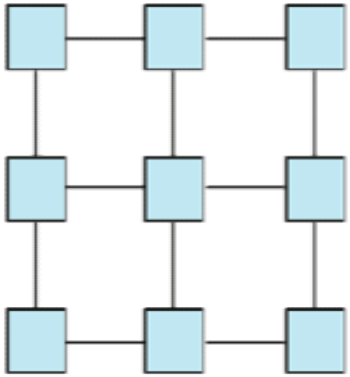
- Hyper cube. Components are numbered in binary; components are connected to one another if their binary representations differ in exactly one bit.

  In a hypercube interconnection, a message from a component can reach any other component by going through at most log(n) links.

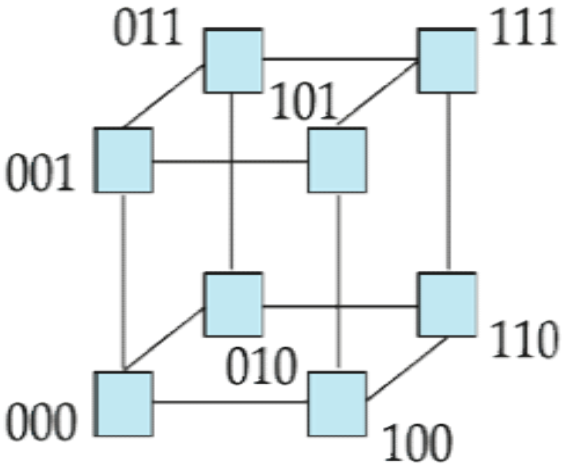
# Interconnection Architectures



(a) bus



(b) mesh



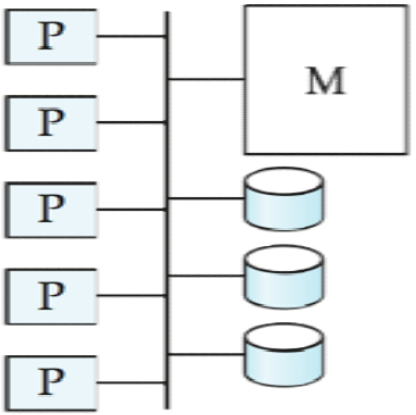

(c) hypercube

# Parallel Database Architectures

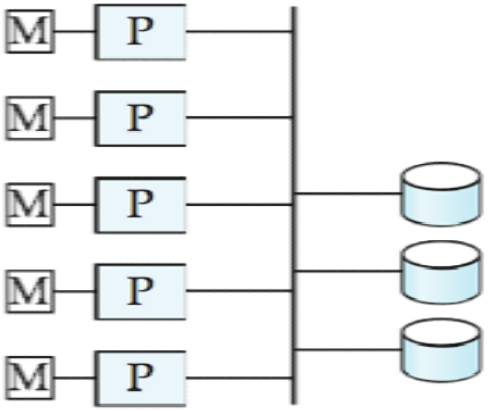
There are several architectural models for parallel machines. Among the most prominent ones are those in Figure

- Shared memory -- processors share a common memory
- Shared disk -- processors share a common disk
- Shared nothing -- processors share neither a common memory nor common disk
- Hierarchical -- hybrid of the above architectures

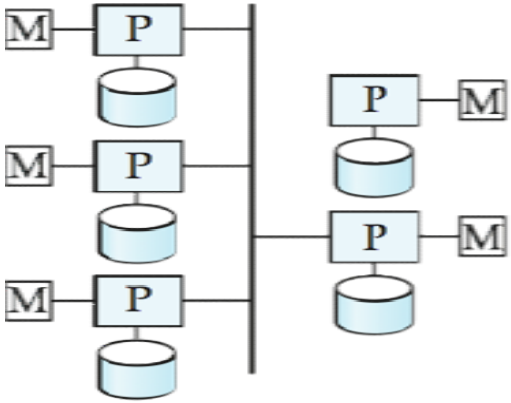
# Parallel Database Architectures



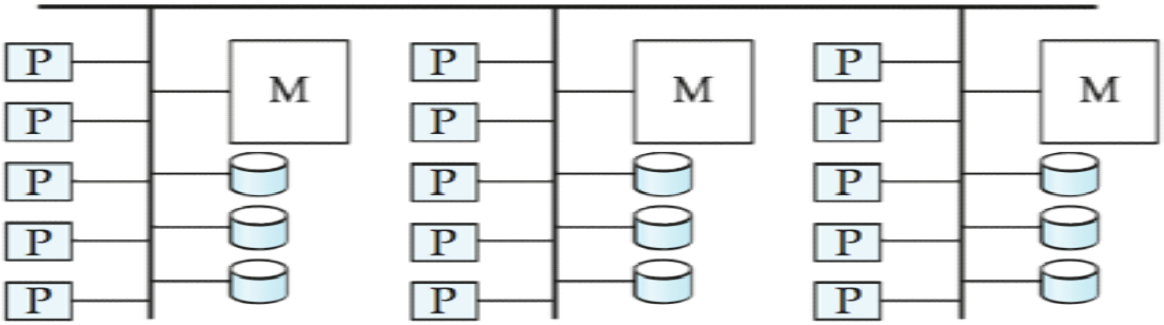

(a) shared memory




(b) shared disk




(c) shared nothing




(d) hierarchical

# Shared Memory

- Processors and disks have access to a common memory, typically via bus or through an interconnection network.
- The benefit of shared memory is , efficient communication between processors, data in shared memory can be accessed by any processor without being moved with software.
- The downside of shared-memory machines is that the architecture is not scalable beyond 32 or 64 processors because the bus or the interconnection network becomes a bottleneck
- Widely used for lower degrees of parallelism (4 to 8).

# Shared Disk

- All processors can directly access all disks via an interconnection network, but the processors have private memories.
- There are two advantages of this architecture over a shared-memory architecture.
- First, since each processor has its own memory, the memory bus is not a bottleneck.
- Second, it offers a cheap way to provide a degree of fault tolerance If a processor (or its memory) fails, the other processors can take over its tasks
- Shared-disk systems can scale to a somewhat larger number of processors, but communication between processors is slower.

# Shared Nothing

- Each node consists of a processor, memory, and one or more disks. Processors at one node communicate with another processor at another node using an interconnection network. A node functions as the server for the data on the disk or disks the node owns.
- Data accessed from local disks (and local memory accesses) do not pass through interconnection network, thereby minimizing the interference of resource sharing.
- Shared-nothing multiprocessors can be scaled up to thousands of processors without interference.
- Main drawback: cost of communication and non-local disk access; sending data involves software interaction at both ends.

# Hierarchical

- Combines characteristics of shared-memory, shared-disk, and shared-nothing architectures.
- At the top level, the system consists of nodes connected by an interconnection network, and do not share disks or memory with one another.
- Each node of the system could be a shared-memory system with a few processors.
- Alternatively, each node could be a shared-disk system, and each of the systems sharing a set of disks could be a shared-memory system.

# I/O Parallelism

- I/O parallelism refers to reducing the time required to retrieve relations from disk by partitioning the relations on multiple disks.
- The most common form of data partitioning in a parallel database environment is horizontal partitioning.
- In horizontal partitioning, the tuples of a relation are divided among many disks, so that each tuple resides on one disk.
- Several partitioning strategies have been proposed.
- Round-robin- it scans the relation in any order and sends the ith tuple to disk
- Hash partitioning- Hash partitioning is a method of separating out rows and spreading them evenly in sub-tables within databases
- Range partitioning-This strategy distributes contiguous attribute-value ranges to each disk.

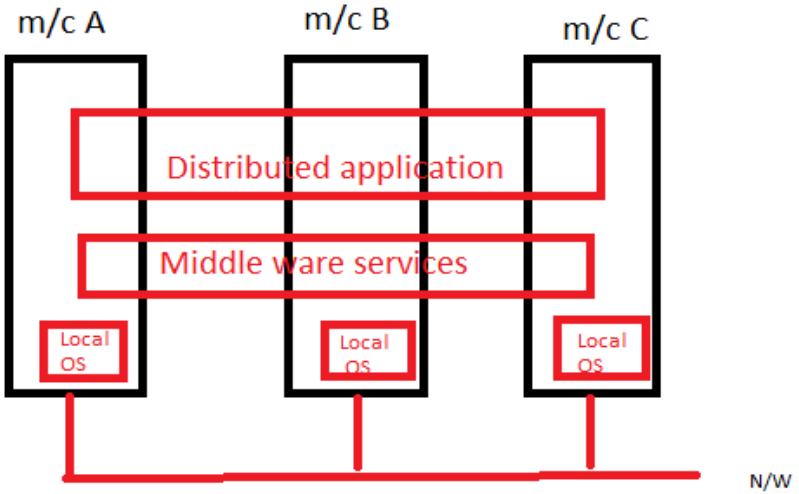
# Interquery Parallelism

• In interquery parallelism, different queries or transactions execute in parallel with one another.

- Transaction throughput can be increased by this form of parallelism.
- However, the response times of individual transactions are no faster than they would be if the transactions were run in isolation.
- primary use of interquery parallelism is to scaleup a transaction-processing system to support a larger number of transactions per second.

# Distributed Systems

- It is collection of autonomous computers linked by computer network and equipped with distributed systems software.
- This software enables computer to coordinate their activities and shares resources of systems like s/w, h/w, data.
- The database is stored on several computers.
- The computers in a distributed system communicate with one another through various communication media, such as high-speed networks or telephone lines.
- Data shared by users on multiple machines



# Characteristic of Distributed systems

- **Resources sharing-** is a ability to use any h/w , s/w or data anywhere in the system.
- **Concurrency**- arises naturally in distributed system as they are using different computers.
- **Openness**- is concerned with extension and improvement of distributed system.
- **Fault tolerance-** it cares for the reliability of system means even some failure occur system will run properly
- **Transparency**-it hides the complexity of distributed system to users and application programs.

# Application of Distributed systems

**Information society**-search engines, social networking,wikipedia

**Finance and commerce**- ecommerce like amazon,flipkart,online banking

**Entertainment**-online gaming music YouTube

# Distributed Databases

- The computers in a distributed system may vary in size and function.
- The main differences between shared-nothing parallel databases and distributed databases are that distributed databases are typically geographically separated, are separately administered, and have a slower interconnection.
- in a distributed database system, we differentiate between local and global transactions.
- Differentiate between local and global transactions
  - A local transaction accesses data in the single site at which the transaction was initiated.
  - A global transaction either accesses data in a site different from the one at which the transaction was initiated or accesses data in several different sites.

# Distributed Databases

- Homogeneous distributed databases
  - Same software/schema on all sites, data may be partitioned among sites
  - Goal: provide a view of a single database, hiding details of distribution
- Heterogeneous distributed databases
  - Different software/schema on different sites
  - Goal: integrate existing databases to provide useful functionality

# Trade-offs in Distributed Systems

- There are several reasons for building distributed database systems, including sharing of data, autonomy, and availability.
- Sharing data – users at one site able to access the data residing at some other sites.
- Autonomy – each site is able to retain a degree of control over data stored locally.
- Higher system availability through redundancy — data can be replicated at remote sites, and system can function even if a site fails means If one site fails in a distributed system, the remaining sites may be able to continue operating.
- Disadvantage: added complexity required to ensure proper coordination among sites.
  - Software development cost.
  - Greater potential for bugs.
  - Increased processing overhead.

# Distributed Data Storage

There are 2 ways in which data can be stored on different sites.

1. Replication -In this approach, the entire relation is stored redundantly at 2 or more sites. If the entire database is available at all sites, it is a fully redundant database. Hence, in replication, systems maintain copies of data.

   Advantage- it increases the availability of data at different sites.

   - query requests can be processed in parallel.

   Disadvantage-Data needs to be constantly updated.

   -Any change made at one site needs to be recorded at every site that relation is stored or else it may lead to inconsistency.

2. Fragmentation - the relations are fragmented (i.e., they're divided into smaller parts) and each of the fragments is stored in different sites where they're required.

   Advantage- it doesn't create copies of data, consistency is not a problem.

# Data Replication

- **Data Replication** is the process of storing data in more than one site or node.
- It is simply copying data from a database server to another server so that all the users can share the same data without any inconsistency.
- The result is a **distributed database** in which users can access data relevant to their tasks without interfering with the work of others.

- Advantages of Replication
  - Availability: failure of site containing relation does not result in unavailability of data.
  - Parallelism: queries may be processed by several nodes in parallel.
  - Reduced data transfer: relation is available locally at each site.
- Disadvantages of Replication
  - Increased cost of updates: each replica of relation must be updated.
  - Increased complexity of concurrency control: concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented.
    - One solution: choose one copy as primary copy and apply concurrency control operations on primary copy

# Data Fragmentation

- Division of relation into fragments r1, r2, .., rn which contain sufficient information to reconstruct relation.
- Horizontal fragmentation: groups the tuples of a table in accordance to values of one or more fields.
- Vertical fragmentation: the fields or columns of a table are grouped into fragments.

# Data Transparency

- The user of a distributed database system should not be required to know where the data are physically located.
- The user of a distributed database system should not be required to know how the data can be accessed at the specific local site

– Several Forms of Data Transparency

  - Fragmentation transparency

    Users are not required to know how a relation has been fragmented

  - Replication transparency

    Users view each data object as logically unique

    Users do not have to be concerned with what data objects have been replicated, or where replicas have been placed

  - Location transparency

    Users are not required to know the physical location of the data

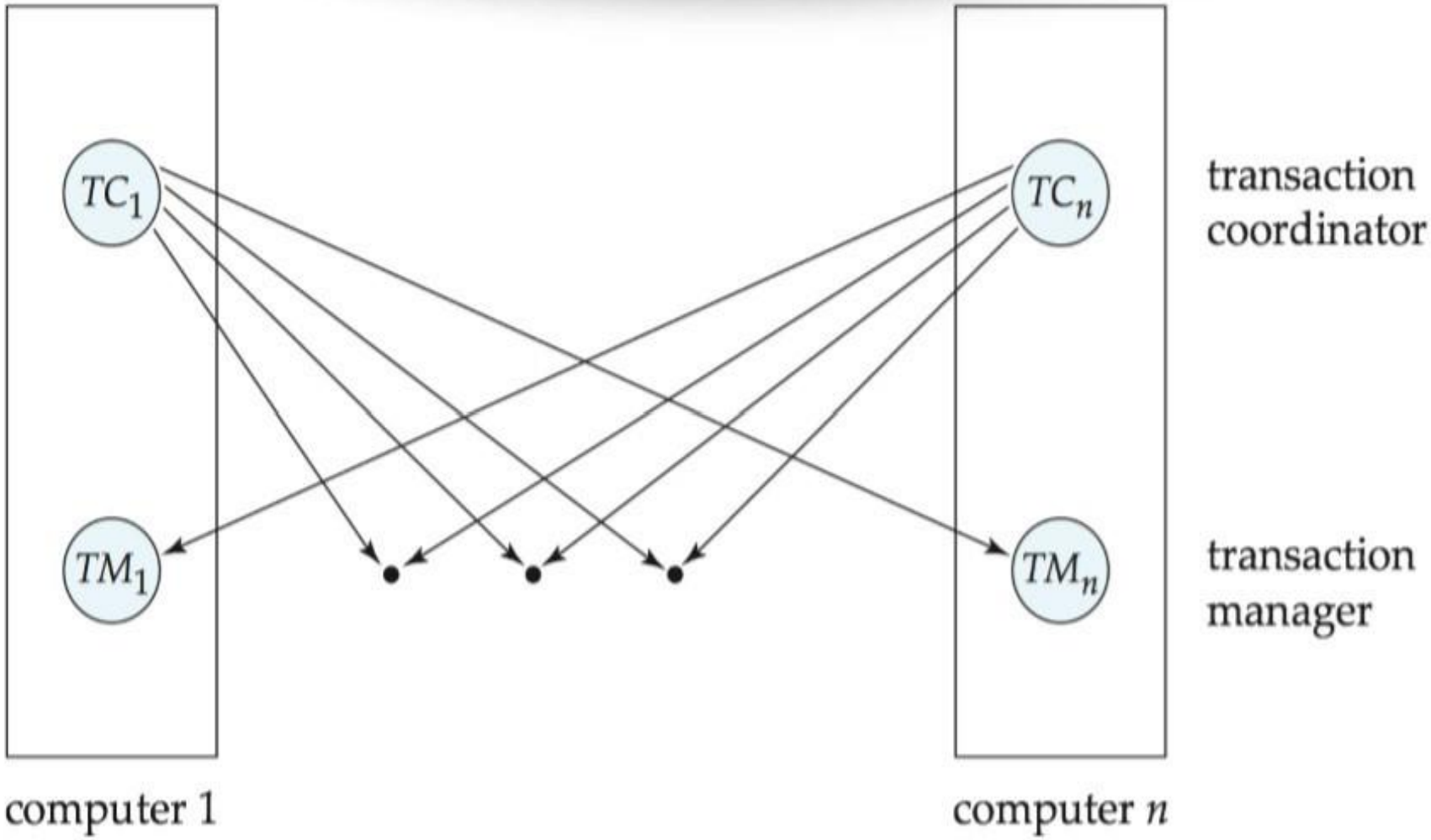
# Naming of Data Items - Criteria

- Every data item must have a system wide unique name.
- It should be possible to find the location of data items efficiently.
- It should be possible to change the location of data items transparently.
- Each site should be able to create new data items autonomously.

# Distributed Transactions

- Transaction may access data at several sites.
- Each site has a local transaction manager responsible for:
  - Maintaining a log for recovery purposes
  - Participating in coordinating the concurrent execution of the transactions executing at that site.
- Each site has a transaction coordinator, which is responsible for:
  - Starting the execution of transactions that originate at the site.
  - Distributing sub transactions at appropriate sites for execution.
  - Coordinating the termination of each transaction that originates at the site, which may result in the transaction being committed at all sites or aborted at all sites.

# Transaction System Architecture



# System Failure Modes

- Failures unique to distributed systems:
  - Failure of a site.
  - Loss of massages
    - Handled by network transmission control protocols such as TCP-IP
  - Failure of a communication link
    - Handled by network protocols, by routing messages via alternative links
  - Network partition
    - A network is said to be partitioned when it has been split into two or more subsystems that lack any connection between them
    - Note: a subsystem may consist of a single node

# Commit Protocols

- Commit protocols are used to ensure atomicity across sites
  - a transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.
  - not acceptable to have a transaction committed at one site and aborted at another

The different distributed commit protocols are −
One-phase commit
Two-phase commit
Three-phase commit

# One-phase commit

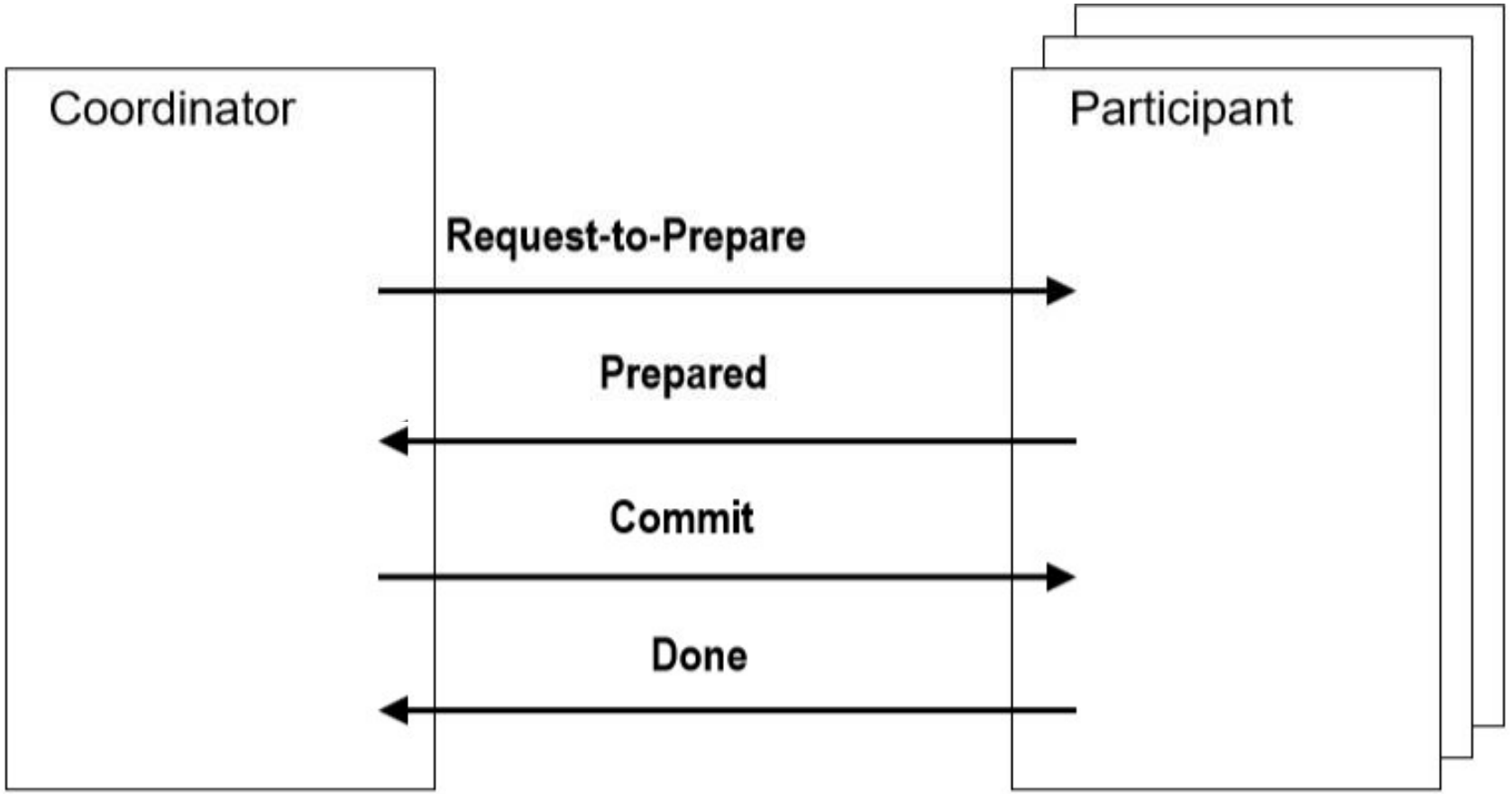
- One-phase commit is the simplest commit protocol. Let us consider that there is a controlling site and a number of slave sites where the transaction is being executed. The steps in distributed commit are −
- After each slave has locally completed its transaction, it sends a "DONE" message to the controlling site.
- The slaves wait for "Commit" or "Abort" message from the controlling site. This waiting time is called **window of vulnerability**.
- When the controlling site receives "DONE" message from each slave, it makes a decision to commit or abort. This is called the commit point. Then, it sends this message to all the slaves.
- On receiving this message, a slave either commits or aborts and then sends an acknowledgement message to the controlling site.

## Two-Phase Commit Protocol

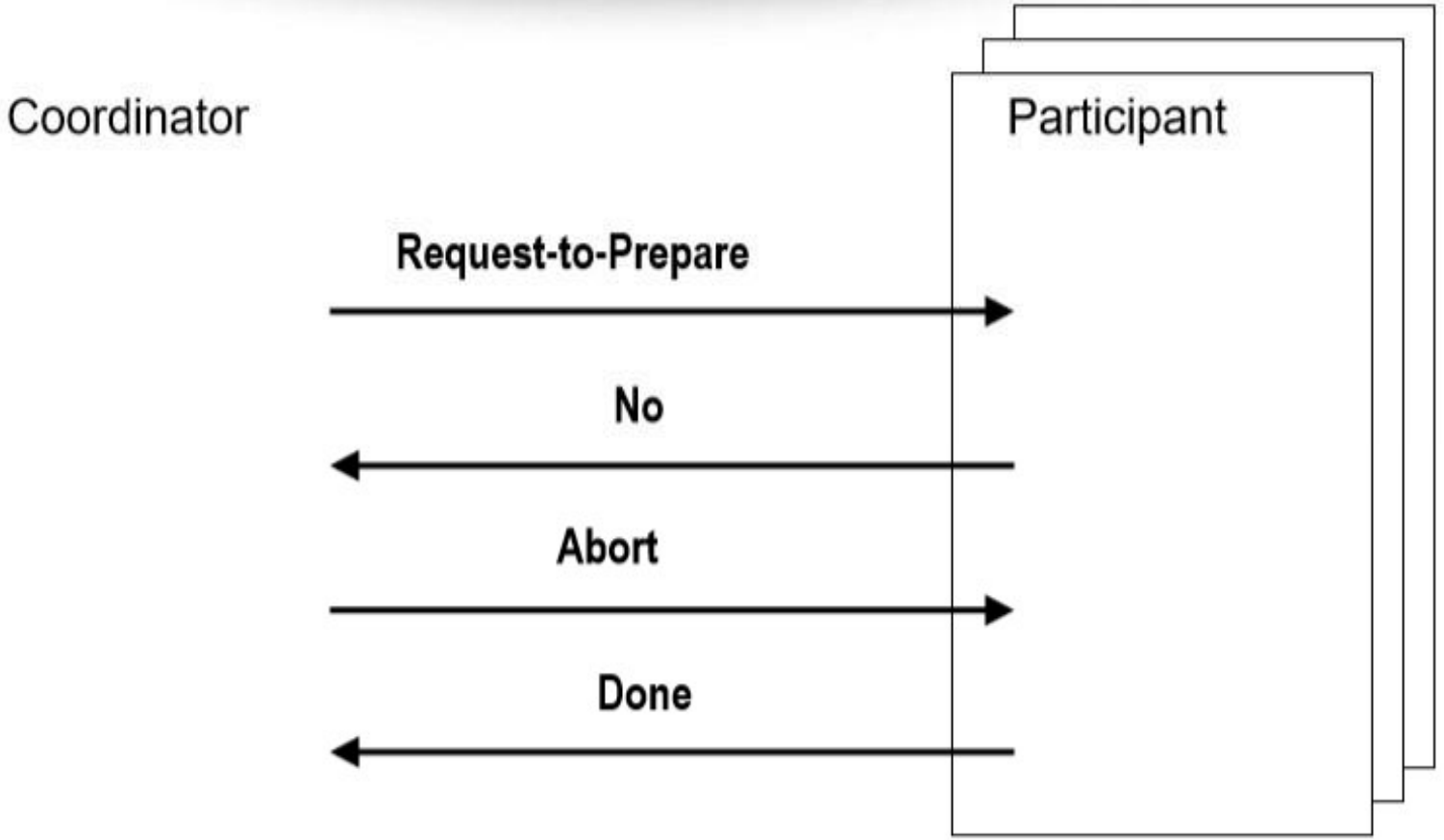
- Goal - ensure the atomicity of a transaction that executing on multiple sites
- Coordinator – the transaction manager of home site where the transaction is originated
- Subordinates- transaction manager of site where the sub transactions execute

1. (Begin Phase 1) The coordinator sends a Prepare message to each subordinates/participants
2. The coordinator waits for all participants to vote
3. Each subordinates/participants
   - votes Prepared if it‘s ready to commit
   - may vote No for any reason
   - may delay voting indefinitely
   - Make entry in local log
4. (Begin Phase 2) If coordinator receives Prepared from all participants, it decides to commit. (The transaction is now committed.) Otherwise, it decides to abort
5. The coordinator sends its decision to all participants (i.e. Commit or Abort) and update log file
6. Participants acknowledge receipt of Commit or Abort by replying Done.

# Case 1: Commit



# Case 2: Abort



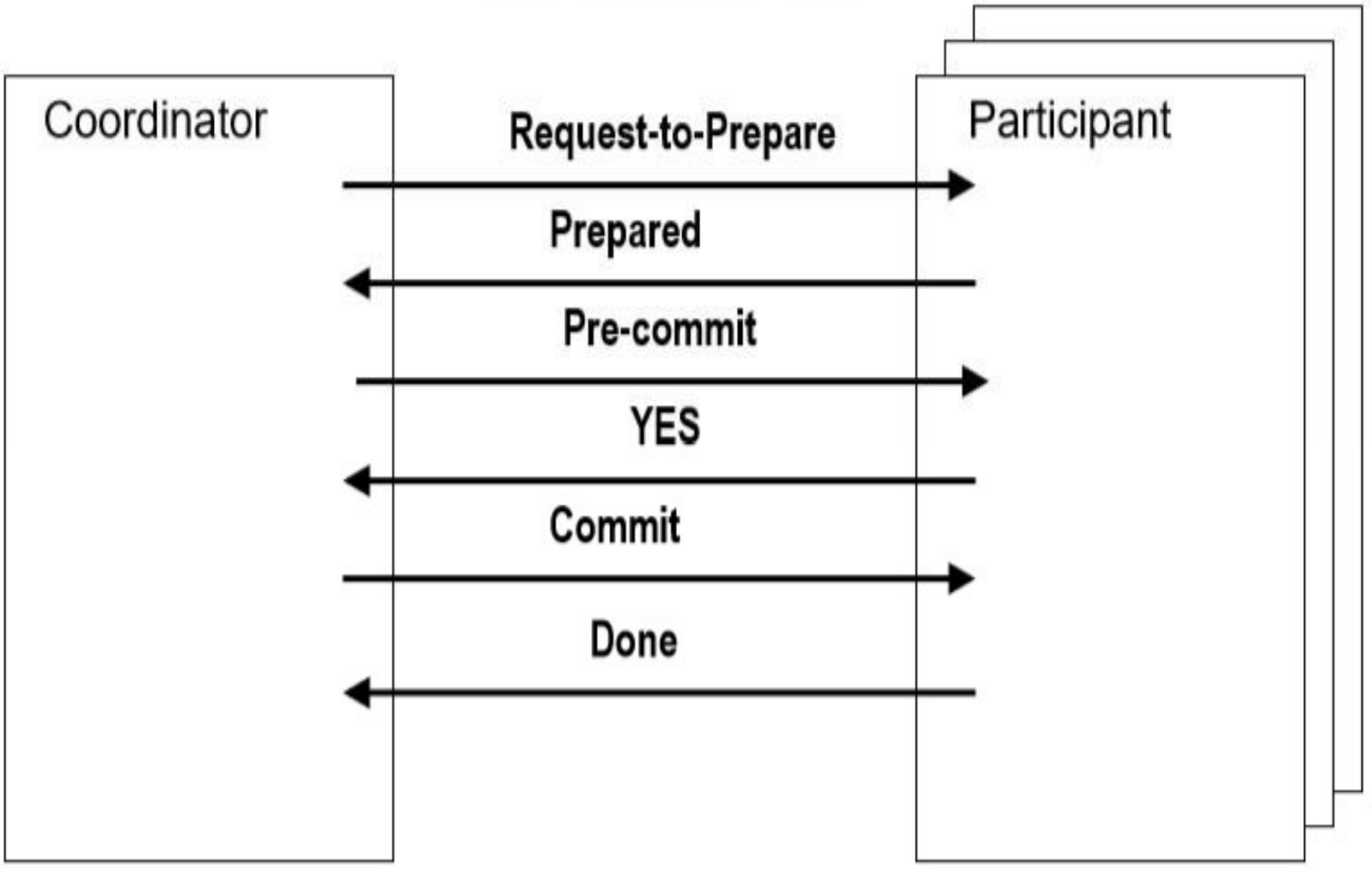
# Three Phase Commit

- 3PC prevents blocking in the absence of communications failures. It can be made resilient to communications failures, but then it may block
- 3PC is much more complex than 2PC, but only marginally improves reliability
  - prevents some blocking situations.
- Main idea: becoming certain and deciding to commit are separate steps.

1. (Begin phase 1) Coordinator sends Request-to prepare to all subordinates/Participants

2. Subordinates vote Prepared or No, just like 2PC.
3. If Coordinator receives Prepared from all participants, then (begin phase 2) it sends Pre-Commit to all subordinates/Participants.
4. Subordinates/Participants wait for Abort or Pre-Commit. subordinates acknowledges Pre-commit.
5. After Coordinator receives acks from all subordinates, or times out on some of them, it (begin third phase) sends Commit to all participants (that are up)

# Three Phase Commit



# Data Warehousing and Mining

**What is Data Warehousing?**

A **Data Warehousing** (DW) is process for collecting and managing data from varied sources to provide meaningful business insights. A Data warehouse is typically used to connect and analyze business data from heterogeneous sources.

It is a blend of technologies and components which aids the strategic use of data.

**Functions of Data Warehouse Tools and Utilities**

The following are the functions of data warehouse tools and utilities –

**Data Extraction** – Involves gathering data from multiple heterogeneous sources.

**Data Cleaning** – Involves finding and correcting the errors in data.

**Data Transformation** – Involves converting the data from legacy format to warehouse format.

**Data Loading** – Involves sorting, summarizing, consolidating, checking integrity, and building indices and partitions.

**Refreshing** – Involves updating from data sources to warehouse.

**Que. Who needs Data warehouse?**

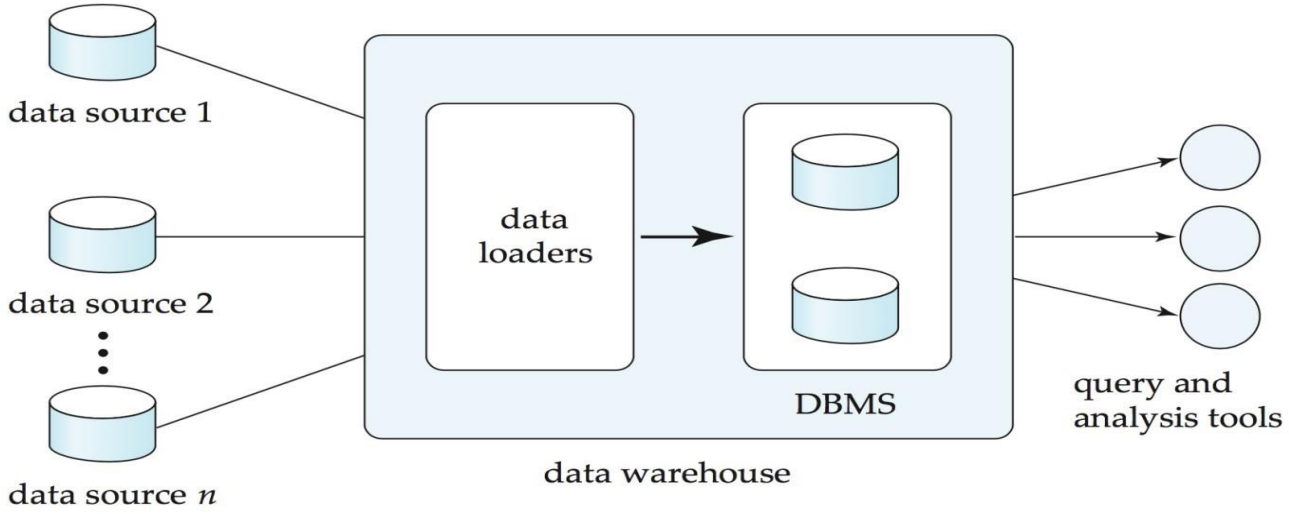
# Data Warehousing

Data sources often store only current data, not historical data

Corporate decision making requires a unified view of all organizational data, including historical data

A data warehouse is a repository (archive) of information gathered from multiple sources, stored under a unified schema, at a single site

- Greatly simplifies querying, permits study of historical trends
- Shifts decision support query load away from transaction processing systems



Design Issues

• When and how to gather data
– Source driven architecture: data sources transmit new information to warehouse, either continuously or periodically (e.g., at night)

– Destination driven architecture: warehouse periodically requests new information from data sources

– Keeping warehouse exactly synchronized with data sources (e.g., using two-phase commit) is too expensive

• Data/updates are periodically downloaded form online transaction processing (OLTP) systems.

• What schema to use
– Schema integration

Data cleansing

- E.g.:correct mistakes in addresses
- Merge address lists from different sources and remove duplicates

What data to summarize

- Raw data may be too large to store on-line
- Aggregate values (totals/subtotals) often suffice
- Queries on raw data can often be transformed by query optimizer to use aggregate values

# Warehouse Schemas

A database uses relational model, while a data warehouse uses Star, Snowflake, and Fact Constellation schema.

## Star Schema:-

The star schema is the simplest data warehouse schema. It is called star schema because the structure of star schema resembles a star, with points radiating from the center.

## Snowflake Schema:-

Snowflake schema is an extension of star schema means it is more complex than star schema.
In star schema each dimension is represented by a single dimension table whereas in snowflake schema each dimension is grouped into multiple lookup table to eliminate the redundancy.

## Fact Constellation Schema:-

A fact constellation schema has more than one fact table. For each star schema, it is possible to create fact constellation schema by splitting the original star schema into more star schemes.

# Introduction of Data Mining

Data mining is the process of semi-automatically analyzing large databases to find useful patterns

Prediction based on past history

- Predict if a credit card applicant poses a good credit risk, based on some attributes (income, job type, age, ..) and past history
- Predict if a pattern of phone calling card usage is likely to be fraudulent

Some examples of prediction mechanisms:

- Classification
  - Given a new item whose class is unknown, predict to which class it belongs(eg. Bank loan Manager, Marketing manager)
- Regression formulae
  - Given a set of mappings for an unknown function, predict the function result for a new parameter value

# Classification rules

- Decision tree.
- Regression
- Association Rules
- Support
- Confidence

# Clustering

- Clustering: Intuitively, finding clusters of points in the given data such that similar points lie in the same cluster
- Can be formalized using distance metrics in several ways
  - Group points into *k* sets (for a given *k*) such that the average distance of points from the centroid of their assigned group is minimized
    - Centroid: point defined by taking average of coordinates in each dimension.
  - Data mining systems aim at clustering techniques that can handle very large data sets
  - E.g., the Birch clustering algorithm
  - k means
  - fuzzy c means