Chapter 18: Database System Architectures

- Centralized Systems
- Client--Server Systems
- Parallel Systems
- Distributed Systems
- Network Types

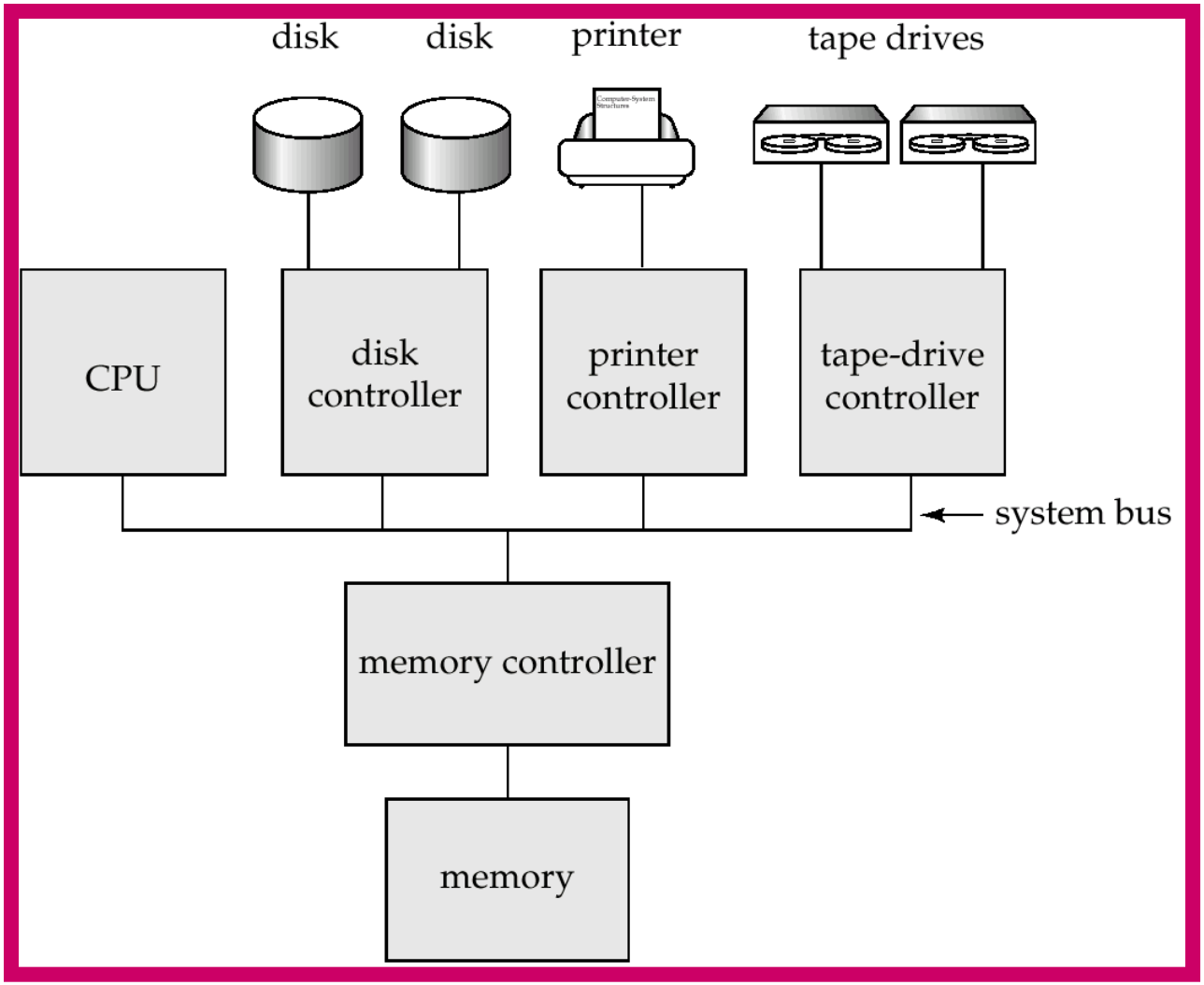
Different Systems

- Database system is influenced by underlying computer systems.
- **Client – server:** Tasks executed on server and client machines.
- **Parallel:** Database activities are processed parallely within computer system.
- **Distributed:** Data is distributed across sites or departments but accessible from other sites

Centralized Systems

- Centralized system 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.
- CPUs have cache memory that stores local copies of parts of memory to speed data access

A Centralized Computer System



- Single-user system (e.g., personal computer or workstation): desk-top 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 vie terminals Often called *server* systems.

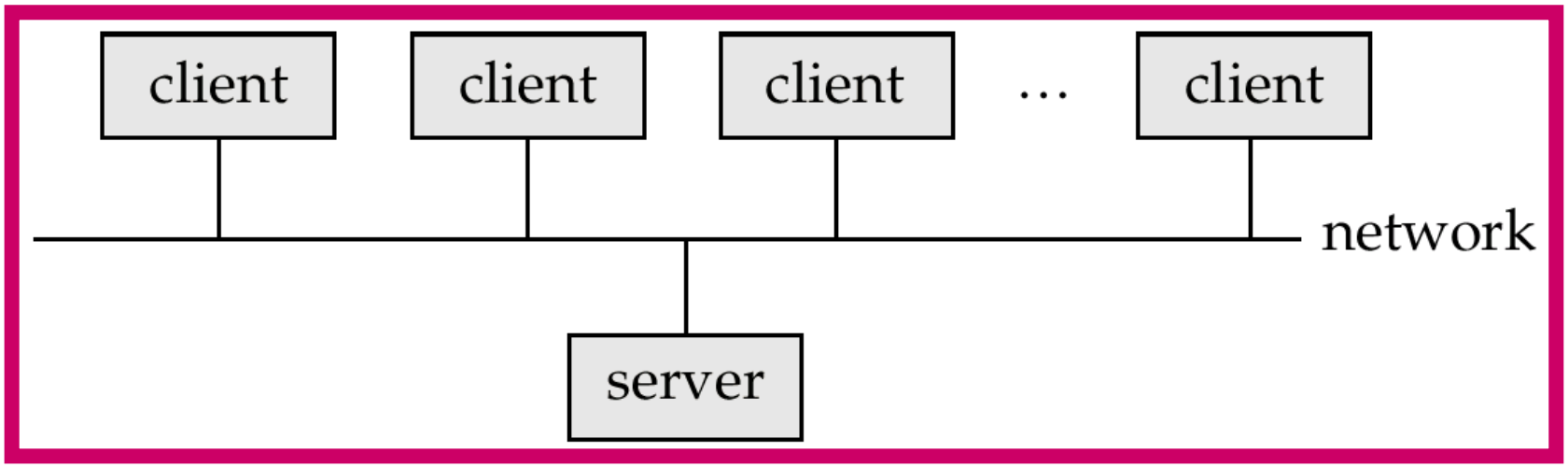
Centralized Systems (contd…)

- Single user system may not support concurrency control and crash recovery system may be absent or primitive.
- **Coarse-Granularity Parallelism:**
  - Only few processors sharing main memory
  - Database system do not partition single query among processes.
  - Run a query on single processor allowing multiple queries to run concurrently.
  - Higher throughput with more number of transaction per second, but individual transaction do not run faster.

- Fine Granularity Parallelism:
  - Large number of processors
  - Parallelizes tasks submitted by user

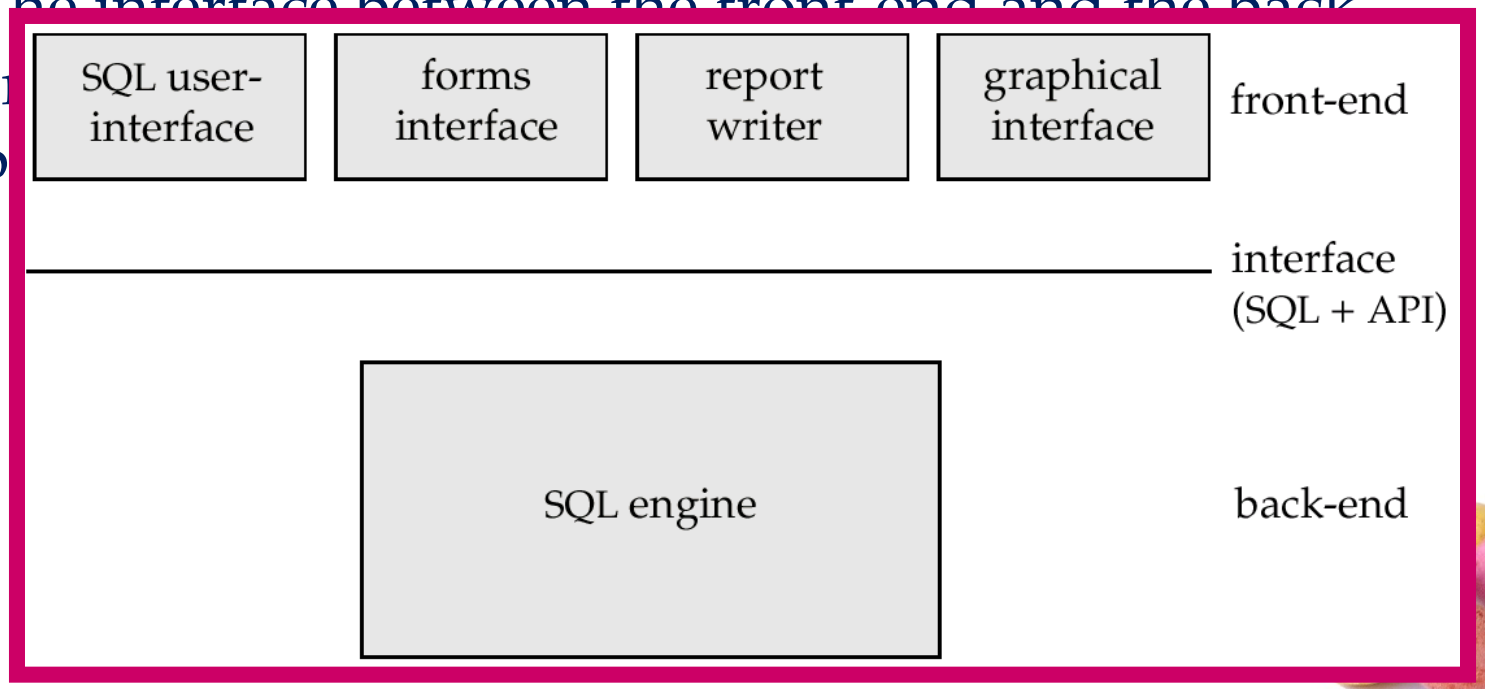
Client-Server Systems

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



Client-Server Systems (Cont.)

- 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 through which they communicate.



Client-Server Systems (Cont.)

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

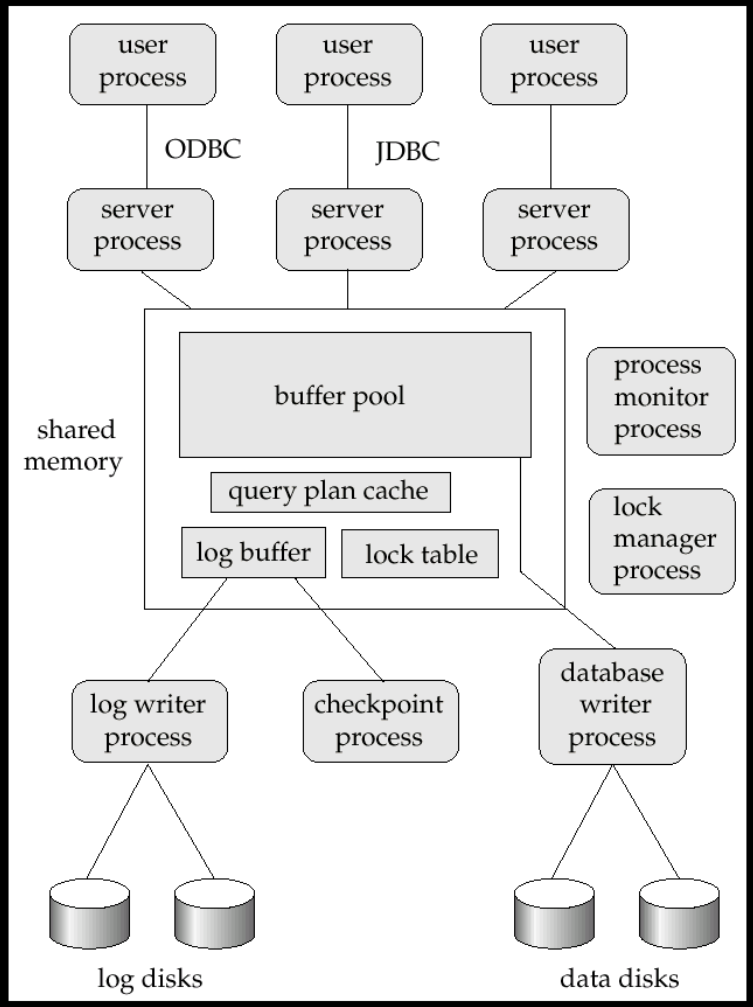
Transaction Server

- Also called **query server** systems or SQL *server* systems; clients send requests to the server system where the transactions are executed, and results are shipped back to the client.
- Requests specified in SQL, and communicated to the server through a *remote procedure call (RPC)* mechanism.
- Interface like *Open Database Connectivity (ODBC)* or JDBC can also be used.

Transaction Server Process Structure

- A typical transaction server consists of multiple processes accessing data in shared memory.
- Server process
  - These are processes that receive user queries, execute them , and send the results back.
  - This queries may be submitted via user interface or user process.
- Lock manager process
  - It implements lock manager functionality including granting of lock, lock release and deadlock detection.
- Database writer process
  - Output/write modified buffer blocks to disks continually.

- Log writer process
  - Server processes 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.


user process
user process
user process
ODBC
JDBC
server process
server process
server process
shared memory
buffer pool
process monitor process
query plan cache
lock manager process
log buffer
lock table
log writer process
checkpoint process
database writer process
log disks
data disks

Transaction System Processes (Cont.)

- 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 data in 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

Transaction System Processes (Cont.)

- To avoid overhead of inter-process communication for lock request/grant, each database process operates directly on the lock table data structure instead of sending requests to lock manager process.
- Mutual exclusion ensured on the lock table using semaphores, or more commonly, atomic instructions.
- If a lock can be obtained, the lock table is updated directly in shared memory.
- If a lock cannot be immediately obtained, a lock request is noted in the lock table and the process (or thread) then waits for lock to be granted.

- When a lock is released, releasing process updates lock table to record release of lock, as well as grant of lock to waiting requests (if any)

Data Servers

- Used in LANs, where there is a very high speed connection between the clients and the server,
- client machines are comparable in processing power to the server machine, and the tasks to be executed are compute intensive.
- Data sent to client machines, processing on client is performed, and then send results back to the server machine.
- This architecture requires full back-end functionality at the clients.
- Used in many object-oriented database systems

- Issues:
  - Page-Shipping versus Item-Shipping
  - Locking
  - Data Caching
  - Lock Caching

Data Servers (Cont.)

- **Page-Shipping** versus **Item-Shipping**
  - Unit of communication for data can be page (coarse granularity) or tuple (fine granularity)
  - Smaller unit of shipping ⇒ more message passing
  - Worth **prefetching** related items along with requested item
  - Page shipping can be thought of as a form of prefetching
- Locking
  - Overhead of requesting and getting locks from server is high due to message delays
  - Server can grant locks on requested and prefetched items; with page shipping

Data Servers (Cont.)

- Iet transaction is granted lock on whole page
- Disadvantage of page shipping is that lock on page will lock all items contained in that page
- Other client machines that require locks on those items may be blocked unnecessarily

Data Servers (Cont.)

- Data Caching
  - Data can be cached at client even after transaction completes
  - Successive transaction can use cached data
  - But check that data is up-to-date before it is used (**cache coherency**)
  - Message will be passed to server to check validity of data
- Lock Caching
  - If clients are requesting data which is rarely requested by other clients, locks can also be cached at client machine

- If client requires data and lock is also available in cache, no other processing is required
- Server must keep track of cached locks
- If other transactions requires lock, server **calls back** all conflicting locks that have been cached

Parallel Systems

- Parallel system improve processing and I/O speeds by using multiple CPUs and multiple disks in parallel
- Parallel systems are useful if database is very large or very large number of transactions are to be processed
- A **coarse-grain parallel** machine consists of a small number of powerful processors
- A **massively parallel** or **fine grain parallel** machine utilizes thousands of smaller processors.

- Two main performance measures of database system:
  - **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.

- Large number of small transactions can improve throughput by processing many transactions in parallel.
- When large transaction can improve response time by transaction in parallel.

Speed-Up and Scale-Up

- Important issues in parallesim.

- **Speedup**:
  - Running a given task in less time by increasing degree of parallelism is called as speedup.
- **Scaleup**
  - Handling larger tasks by increasing degree of parallelism is called as scaleup.

Interconnection Network Architectures

- Parallel system has components like processor, memory and disk that communicates with each other using interconnection network

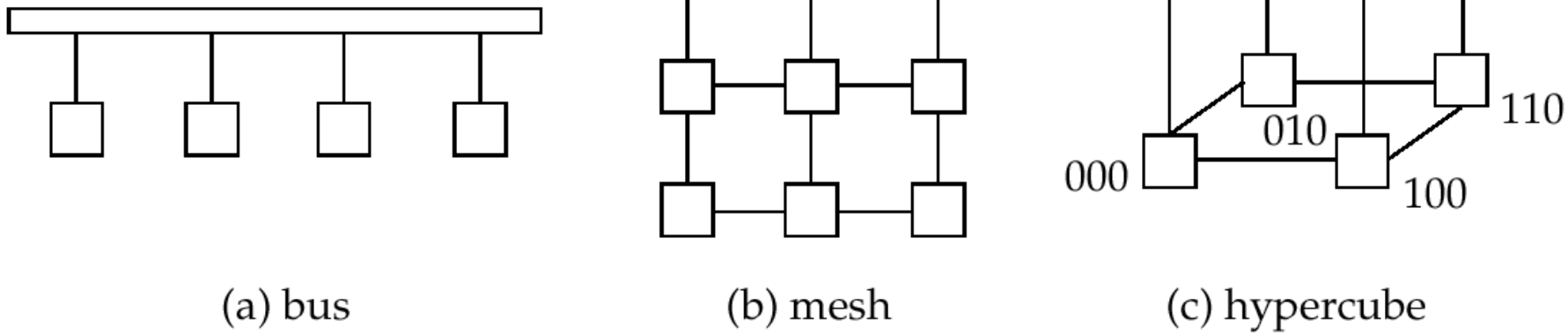
- Bus.
  - All system components can send data on and receive data from a single communication bus
  - Works well for small number of processors
  - Does not scale well with increasing parallelism because single communication from one component at a time
- Mesh
  - Components are arranged as nodes in a grid, and each component is connected to all adjacent components

Interconnection Network Architectures

- Communication links grow with growing number of components, and so scales better.
- Nodes that are not directly connected can communicate with another by routing messages to sequences of intermediate node.

- Hypercube
  - Components are numbered in binary
  - Components are connected to one another if their binary representations differ in exactly one bit.
  - Thus reduces communication delays.

Interconnection Architectures



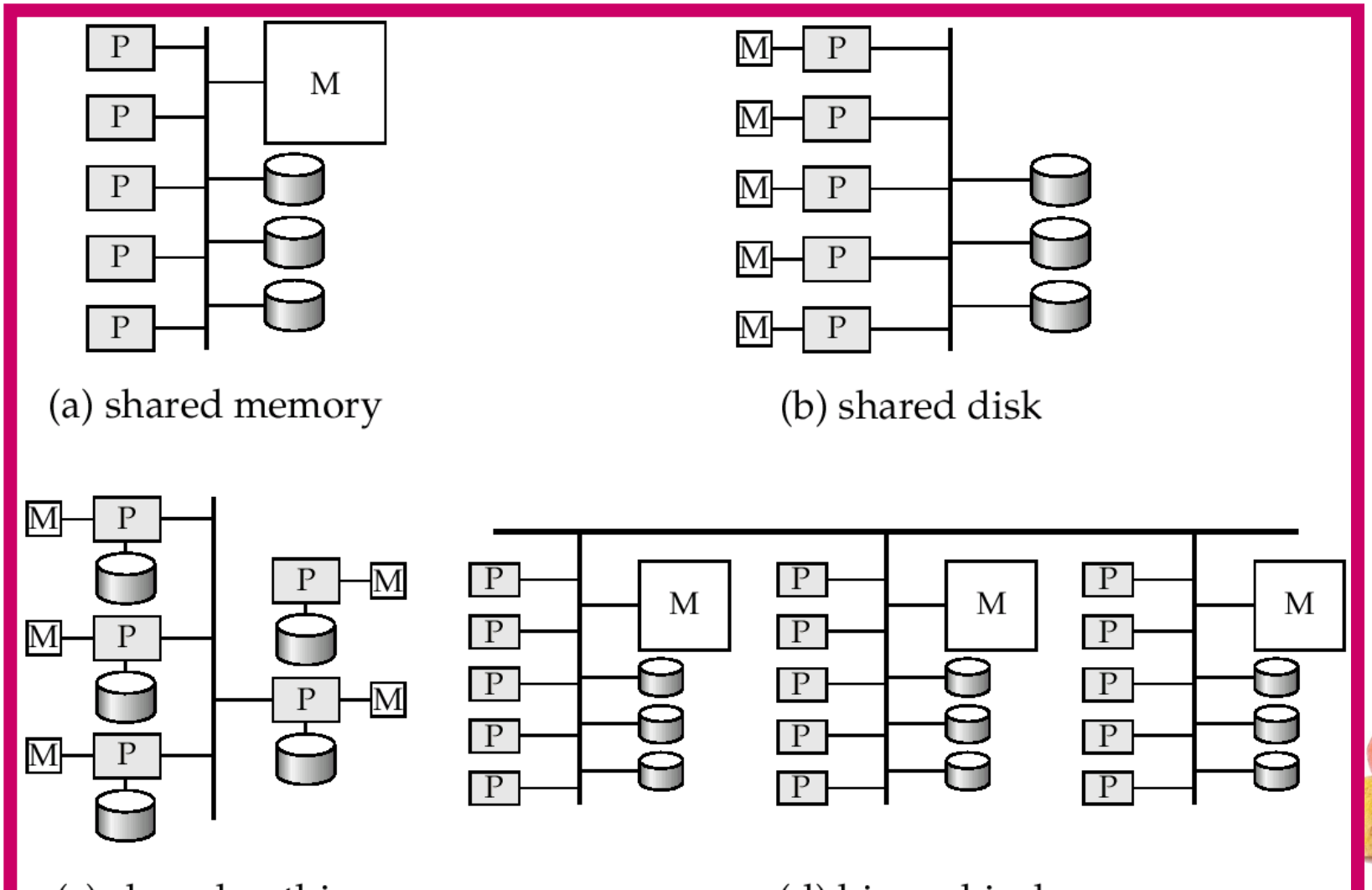
(a) bus (b) mesh (c) hypercube

Parallel Database Architectures

- Several architectural models for parallel machines.

- Shared memory
  - Processors share a common memory
- Shared disk
  - Processors share a common disk. Such systems are called as clusters
- Shared nothing
  - Processors share neither a common memory nor common disk.
- Hierarchical
  - Hybrid of the above architectures.

Parallel Database Architectures



Shared Memory

- Processors and disks have access to a common memory, typically via a bus or through an interconnection network.
- Extremely efficient communication between processors because data in shared memory can be accessed by any processor without having to move it using software.
- Processor can communicate by using memory writes.
- **Disadvantage**: Architecture is not scalable beyond 32 or 64 processors since the bus or the interconnection network becomes a bottleneck

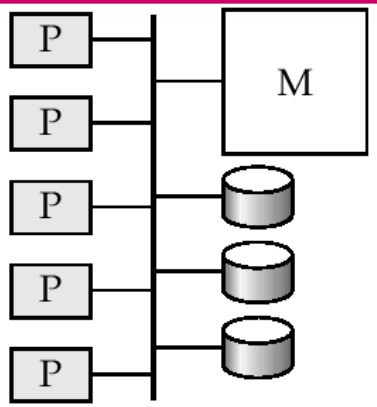
- Generally processor have large cache so that reference to shared memory is avoided whenever possible
- Cache need to be kept coherent ie if a processor performs write to memory location, data in that memory location should be either updated or removed from any processor where it is cached.
- Maintaining cache coherency is overhead with the large number of processor.

Shared Disk

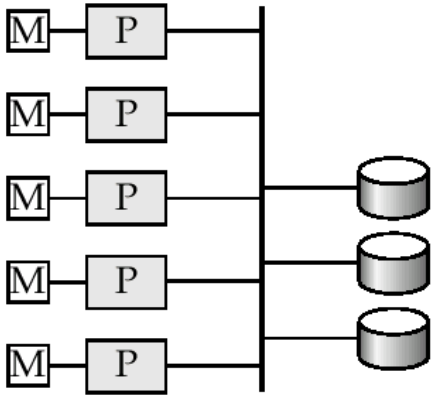
- All processors shares set of disks (called as clusters) but have private memory.
- Advantage
  - The memory bus is not a bottleneck
  - Architecture provides a degree of **fault-tolerance** — if one processor fails, the other processors can take over its tasks since database is resident on disks that are accessible from all processors.
- Interconnection to the disk subsystem is bottleneck because of large access to disk
- Shared-disk systems can scale to a somewhat larger number of processors, but communication between processors is slower.

Shared Nothing

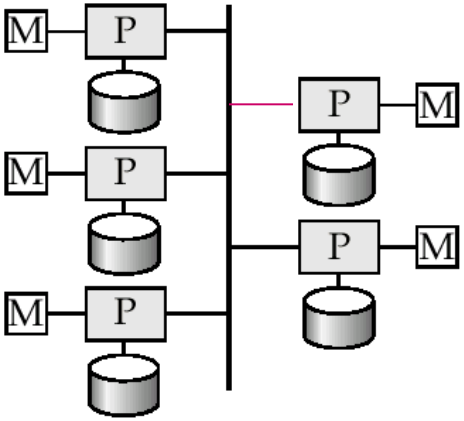
- Processor neither share memory nor disk
- Each node consist of processor, memory and disks.
- Processor at one node can communicate with other node using high speed network connection
- Only queries having access to non local disk is passed to network
- Nodes is work as the server for the data on the disks which is own by nodes.
- Data accessed from local disks do not pass through interconnection network, thereby minimizing the interference of resource sharing.



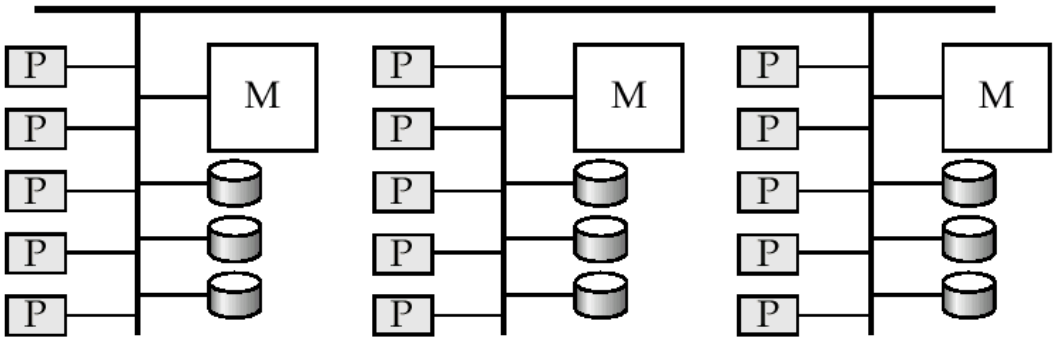
(a) shared memory



(b) shared disk



(c) shared nothing

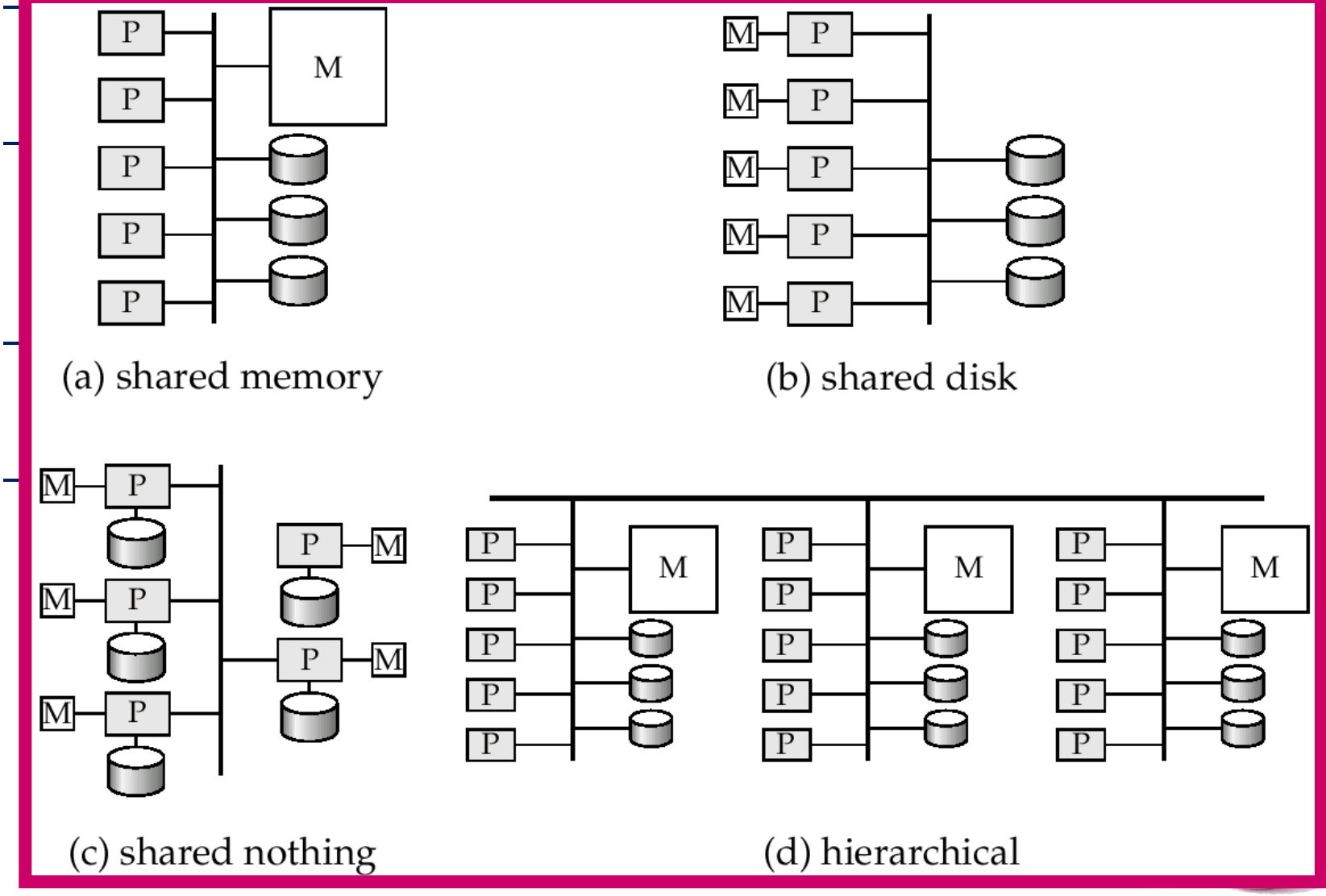


(d) hierarchical

- 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.
-

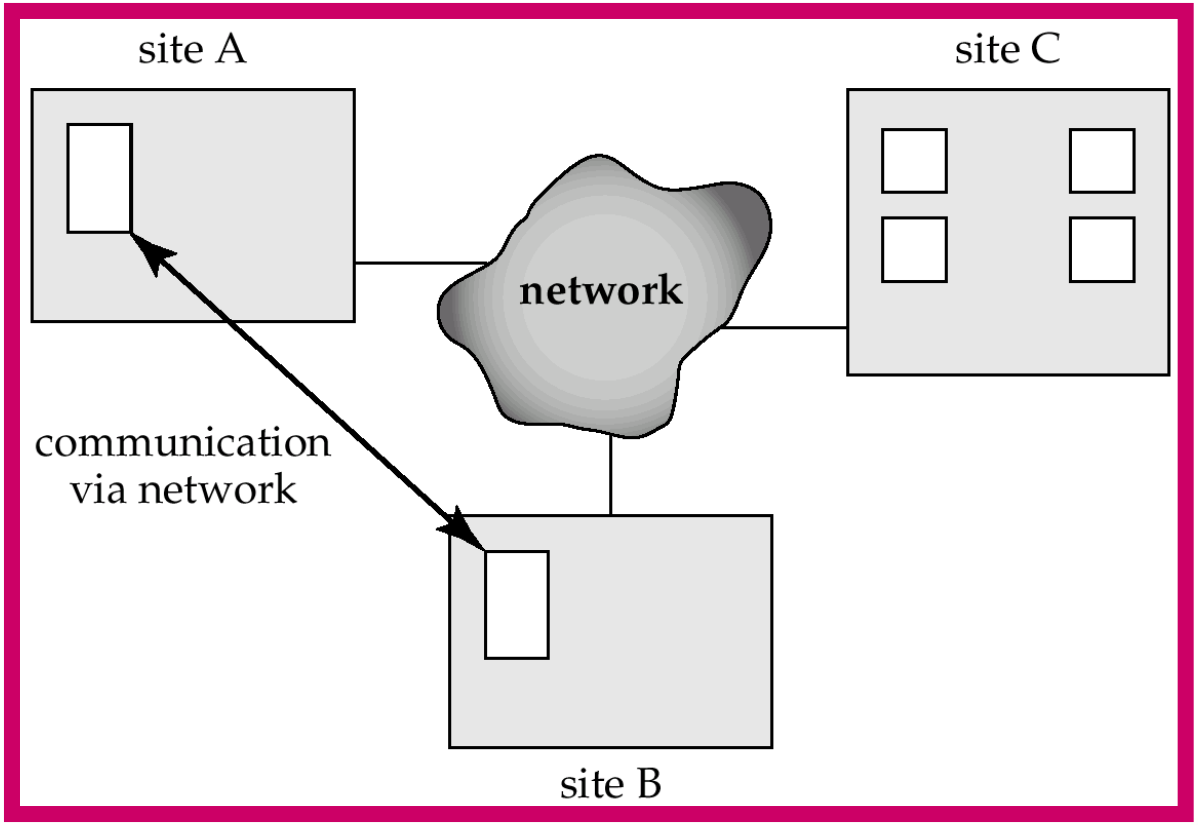
Hierarchical



- Distributed virtual-memory architectures
  - Logically single shared memory, but physically multiple disjoint memory systems
  - Virtual memory mapping hardware, coupled with system software, allows each processor to view disjoint memories as a single virtual memory
  - Also called **non-uniform memory architecture (NUMA)**

Distributed Systems

- Database is stored on several computers. The computers are communicate with each-other via various media(through network).
- Loosely coupled and do not share physical components
- Data spread over multiple machines (also referred to as **sites** or **nodes**.
- Data shared by users on multiple machines.
- Difference between shared-nothing architecture and distributed database is that– distributed **database are geographically separated and** separately administered with lower interconnection.


site A
site C
network
communication
via network
site B

Distributed System(conti.)

- 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.

- another difference is
  - 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.

- Reason for building distributed database systems.
  1. Sharing data :
     - User of one site can access the data residing at other sites.
     - E.g In a distributed banking system, where each branch stores data related to that branch, it is possible for a user in one branch to access data in another branch.
  2. Autonomy :
     - Each site is able to retain a degree of control over data that are stored locally.
     - Local administrator for local site and for global transaction there is a global administrator.

1. Availability :
   - If one site is fail remaining sites may continue operating
   - if data items are replicated in several sites, a transaction needing a particular data item may find that item in any of several sites. Thus, the failure of a site does not necessarily imply the shutdown of the system
   - when the failed site recovers or is repaired, mechanisms must be available to integrate it smoothly back into the system
   - Recovery form the failer of site is complex in distributed system.

- E.g
  - Consider a banking system consisting of four branches in four different cities. Each branch has its own computer, with a database of all the accounts maintained at that branch.
  - Each branch maintains (among others) a relation account(Account-schema),
  - where Account-schema = (account-number, branch-name, balance)
  - The site containing information about all the branches of the bank maintains the relation branch(Branch-schema), where
  - Branch-schema = (branch-name, branch-city, assets)

- Consider a transaction to add $50 to account number A-177 located at the Valleyview branch.
- If the transaction was initiated at the Valleyview branch, then it is considered local; otherwise, it is considered global.

- Implementation Issues

1. Automocity :
   - If a transaction runs across two sites, unless the system designers are careful, it may commit at one site and abort at another, leading to an inconsistent state
   - Can be handled by 2 phase commit protocol
   - Concurrency control is another issue in a distributed database.
   - Since a transaction may access data items at several sites, transaction managers at several sites may need to coordinate to implement concurrency control.

2. Concrreuncy control is another issue in a distributed database.

- Since a transaction may access data items at several sites, transaction managers at several sites may need to coordinate to implement concurrency control.

- Disadvantage:
- The primary disadvantage of distributed database systems is the added complexity required to ensure proper coordination among the sites.
- This increased complexity takes various forms:
  - Software development cost.
    - It is more difficult to implement a distributed database system; thus, it is more costly
  - Greater potential for bugs.
    - it is harder to ensure the correctness of algorithms,
  - Increased processing overhead.
    - The exchange of messages and the additional computation required to achieve intersite coordination are a form of overhead

Network Types

- **Local-area networks (**LANs) – composed of processors that are distributed over small geographical areas, such as a single building or a few adjacent buildings.
- LANs are generally used in an office environment. All the sites in such systems are close to one another, so the communication links tend to have a higher speed and lower error rate than do their counterparts in wide-area networks. The most common links in a local-area network are twisted pair, coaxial cable, fiber optics, and, increasingly, wireless connection

- **Wide-area networks (**WANs) – composed of processors distributed over a large geographical area.