

# Non-Standard Database Systems

## Database System Architectures

Nikolaus Augsten

`nikolaus.augsten@sbg.ac.at`  
Department of Computer Sciences  
University of Salzburg



SS 2017/18

Version 19. März 2018

# Outline

- 1 Centralized and Client-Server Systems
- 2 Server System Architecture
- 3 Parallel Systems
- 4 Distributed Systems
- 5 Network Types

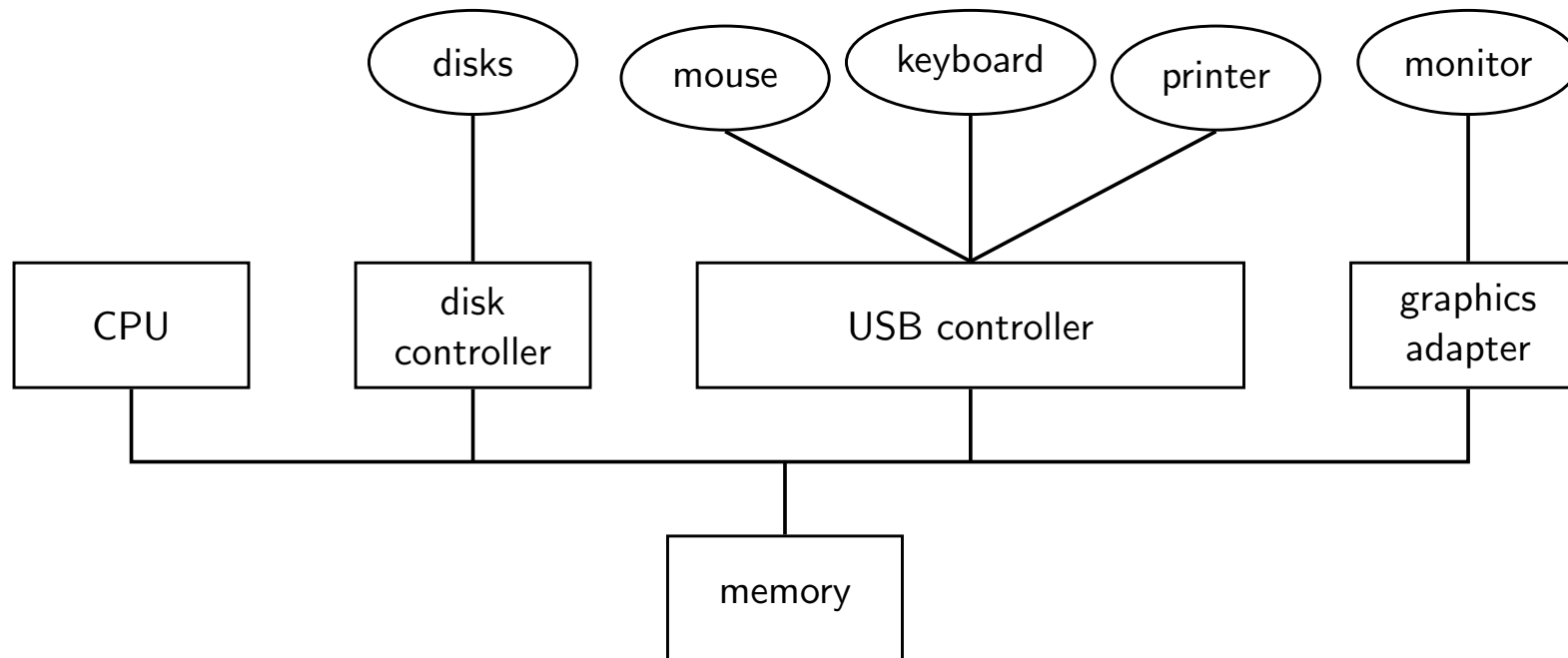
# Outline

- 1 Centralized and Client-Server Systems
- 2 Server System Architecture
- 3 Parallel Systems
- 4 Distributed Systems
- 5 Network Types

# Centralized Systems

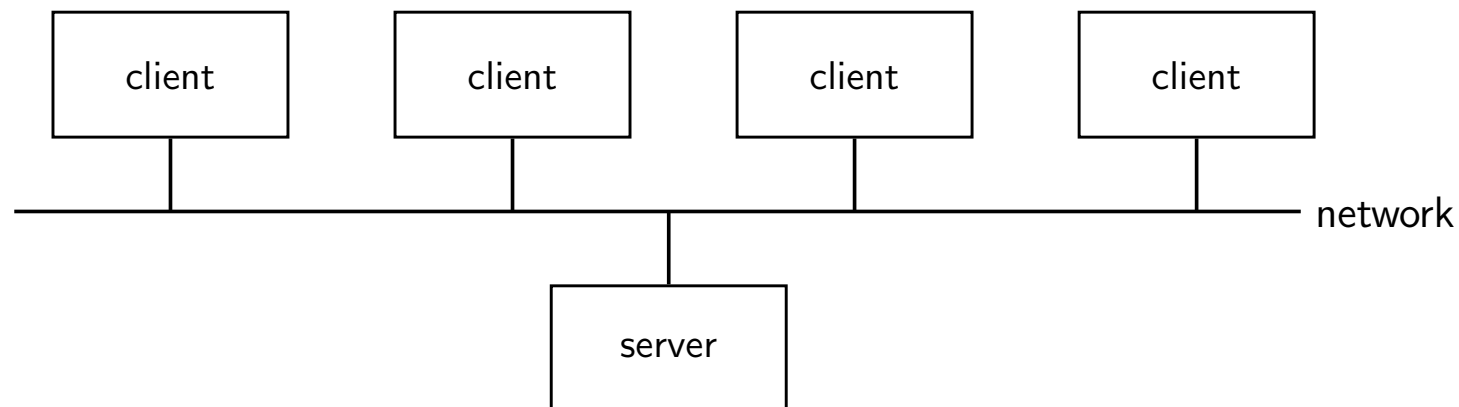
- Run on a **single computer system** and do not interact with other computer systems.
- **General-purpose computer system**: one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory.
- **Single-user system** (e.g., personal computer 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



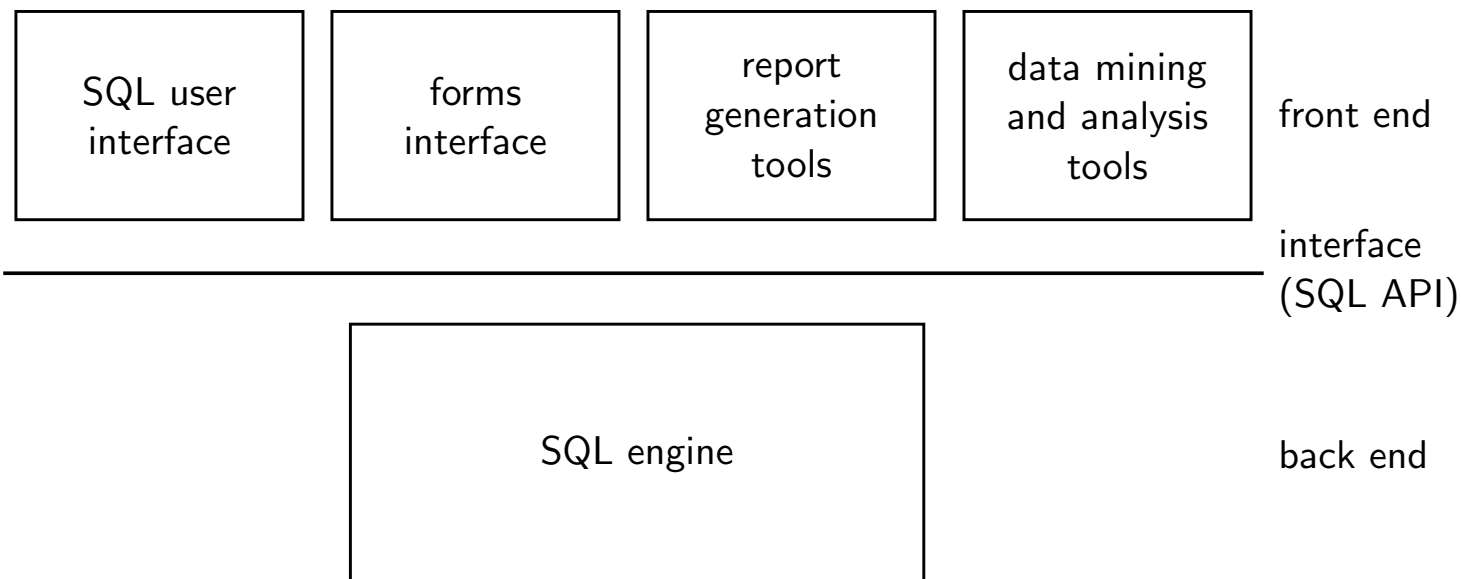
# Client-Server Systems/1

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



# Client-Server Systems/2

- 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/3

- **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**



# Outline

- 1 Centralized and Client-Server Systems
- 2 Server System Architecture**
- 3 Parallel Systems
- 4 Distributed Systems
- 5 Network Types

# Server System Architecture

- 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 Servers

- Also called **query server** systems or SQL server systems
  - Clients send requests to the server
  - Transactions are executed at the server
  - 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

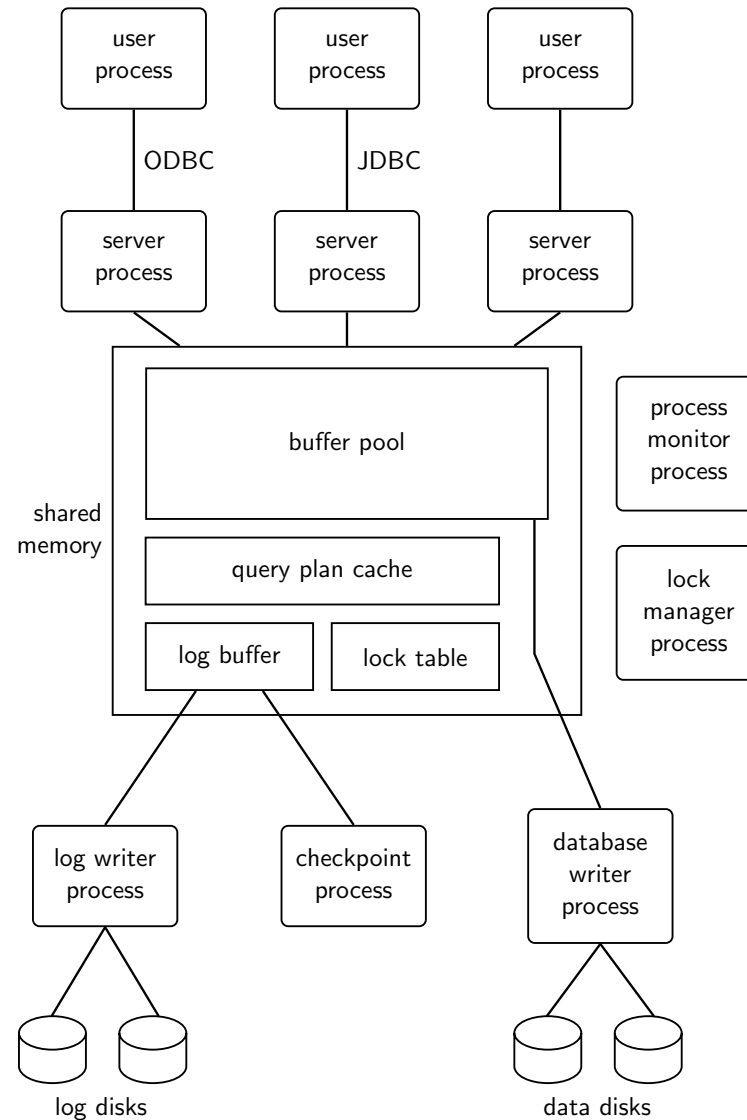
# Transaction Server Process Structure/1

- A typical transaction server consists of **multiple processes** accessing data in **shared memory**.
- **Server processes**
  - Receive user queries (transactions), execute them and send results back
  - Processes may be multithreaded, allowing a single process to execute several user queries concurrently
  - Typically multiple multithreaded server processes
- **Lock manager process**
  - More on this later
- **Database writer process**
  - Output modified buffer blocks to disks continually

# Transaction Server Process Structure/2

- Log writer process
  - Server processes 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

# Transaction Server Process Structure/3



# Transaction Server Process Structure/4

- 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/1

- Used in **high-speed LANs**, in cases where
  - The clients are **comparable** in processing power to the server
  - The tasks to be executed are **compute intensive**.
- Data are shipped to clients where processing is performed, and then **shipped results back** to the server.
- 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/2

- Page-shipping versus item-shipping
  - Smaller unit of shipping → more messages
  - 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
  - Can grant locks on requested and prefetched items; with page shipping, transaction is granted lock on whole page.
  - Locks on a prefetched item can be called back by the server, and returned by client transaction if the prefetched item has not been used.
  - Locks on the page can be deescalated to locks on items in the page when there are lock conflicts. Locks on unused items can then be returned to server.

# Data Servers/3

- Data Caching
  - Data can be cached at client even in between transactions
  - But check that data is up-to-date before it is used ([cache coherency](#))
  - Check can be done when requesting lock on data item
- Lock Caching
  - Locks can be retained by client system even in between transactions
  - Transactions can [acquire cached locks locally](#), without contacting server
  - Server [calls back](#) locks from clients when it receives conflicting lock request. Client returns lock once no local transaction is using it.
  - Similar to deescalation, but across transactions.

# Outline

- 1 Centralized and Client-Server Systems
- 2 Server System Architecture
- 3 Parallel Systems**
- 4 Distributed Systems
- 5 Network Types

# Parallel Systems

- Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network.
- 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:
  - **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

- **Speedup**: a fixed-sized problem executing on a small system is given to a system which is  $N$ -times larger.

- Measured by:

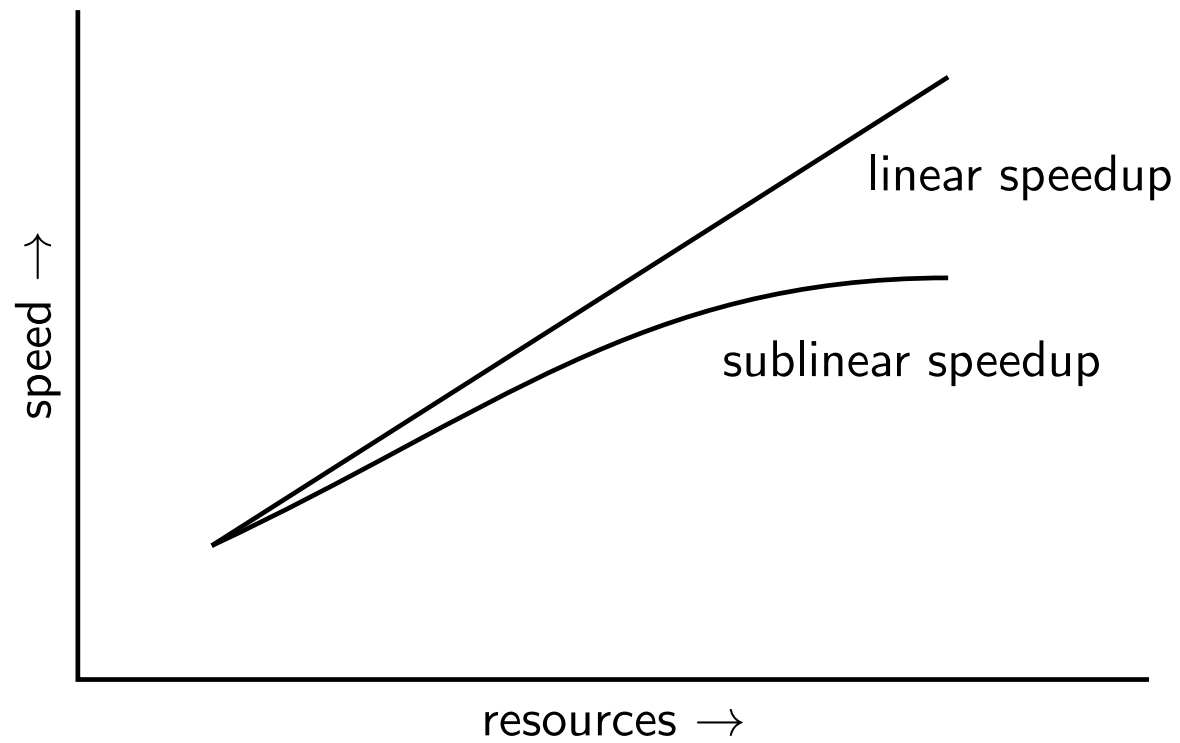
$$speedup = \frac{\textit{small system elapsed time}}{\textit{large system elapsed time}}$$

- Speedup is **linear** if equation equals  $N$ .
- **Scaleup**: increase the size of both the problem and the system
  - $N$ -times larger system used to perform  $N$ -times larger job
  - Measured by:

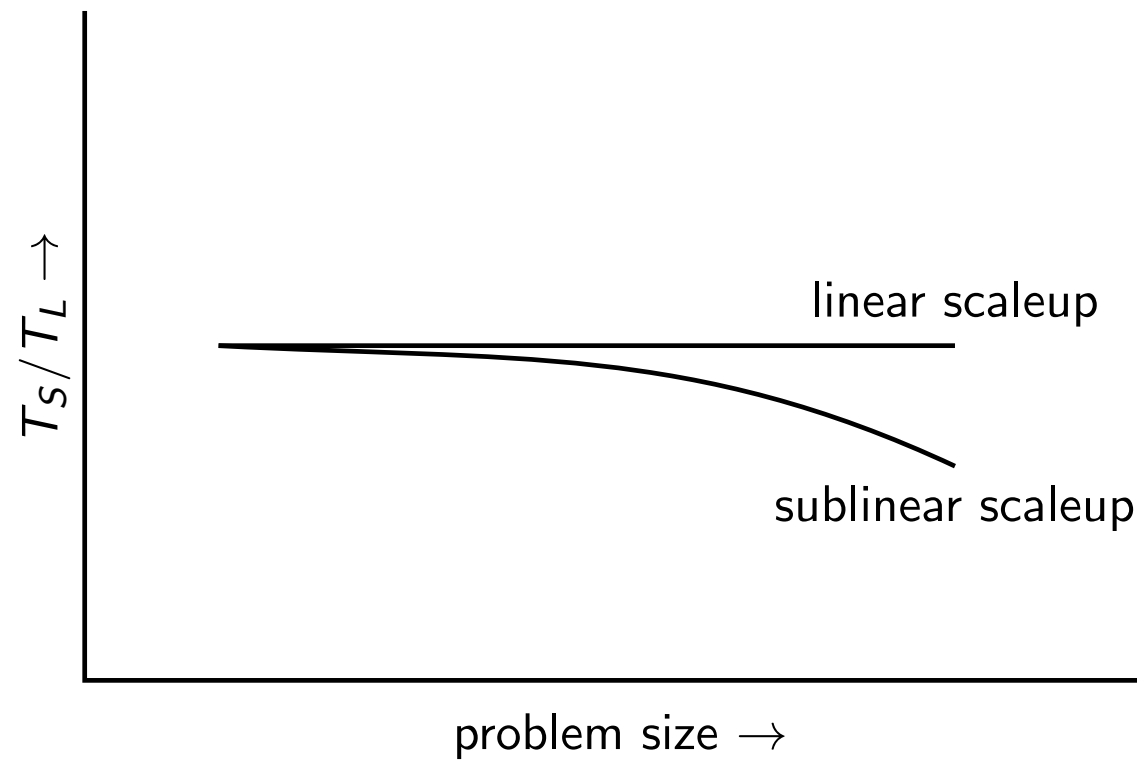
$$scaleup = \frac{\textit{small system smallproblem elapsed time}}{\textit{big system big problem elapsed time}}$$

- Scale up is **linear** if equation equals 1.

# Speedup



# Scaleup



# Batch and Transaction Scaleup

- **Batch scaleup:**
  - A single large job; typical of most decision support queries and scientific simulation.
  - Use an  $N$ -times larger computer on  $N$ -times larger problem.
- **Transaction scaleup:**
  - Numerous small queries submitted by independent users to a shared database; typical transaction processing and timesharing systems.
  - $N$ -times as many users submitting requests (hence,  $N$ -times as many requests) to an  $N$ -times larger database, on an  $N$ -times larger computer.
  - Well-suited to parallel execution.



# Factors Limiting Speedup and Scaleup

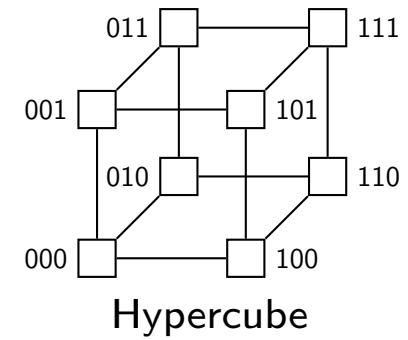
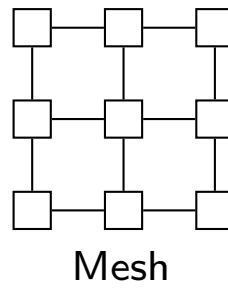
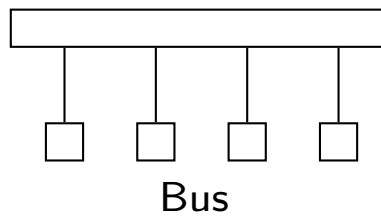
Speedup and scaleup are often sublinear due to:

- **Startup costs:** Cost of starting up multiple processes may dominate computation time, if the degree of parallelism is high.
- **Interference:** Processes accessing shared resources (e.g., system bus, disks, or locks) compete with each other, thus spending time waiting on other processes, rather than performing useful work.
- **Skew:** Increasing the degree of parallelism increases the variance in service times of parallelly executing tasks. Overall execution time determined by slowest of parallelly executing tasks.

# Interconnection Network Architectures

- **Bus:** System components send data on and receive data from a single communication bus;
  - Does not scale well with increasing parallelism.
- **Mesh:** Components are arranged as nodes in a grid, and each component is connected to all adjacent components
  - Communication links grow with growing number of components, and so scales better.
  - But may require  $2\sqrt{n}$  hops to send message to a node (or  $\sqrt{n}$  with wraparound connections at edge of grid).
- **Hypercube:** Components are numbered in binary; components are connected to one another if their binary representations differ in exactly one bit.
  - $n$  components are connected to  $\log(n)$  other components and can reach each other via at most  $\log(n)$  links; reduces communication delays.

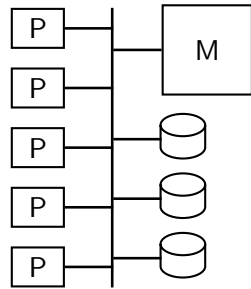
# Interconnection Architectures



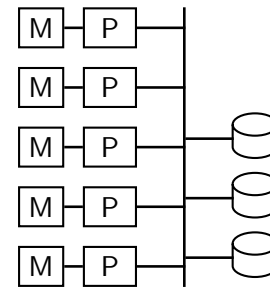
# Parallel Database Architectures

- **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

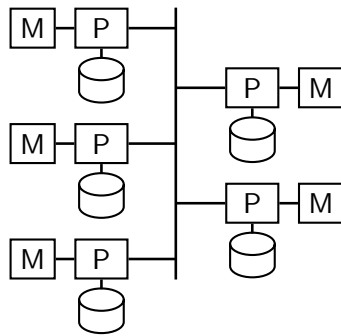
# Parallel Database Architectures



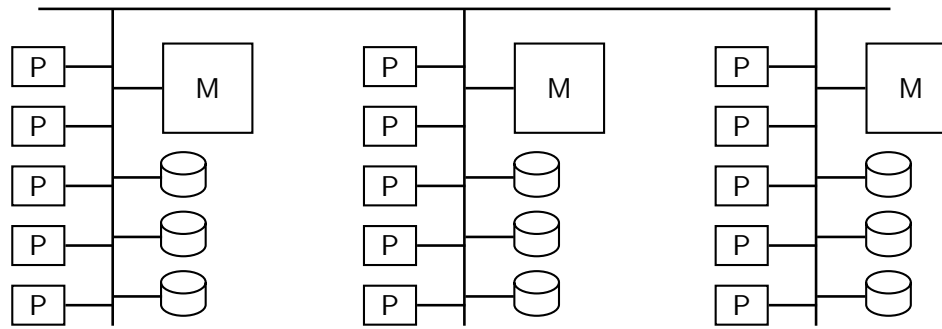
shared memory



shared disk



shared nothing



hierarchical

# 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 ( $< 1\mu s$ ) — data in shared memory can be accessed by any processor.
- Downside: architecture is **not scalable** beyond about 64 processors since 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**.
  - The **memory bus** is not a bottleneck
  - Architecture provides a degree of **fault-tolerance** — if a processor fails, the other processors can take over its tasks since the database is resident on disks that are accessible from all processors.
- **Examples:** IBM Sysplex and DEC clusters (now part of Compaq) running Rdb (now Oracle Rdb) were early commercial users
- **Downside:** bottleneck now occurs at **interconnection to the disk subsystem**.
- Shared-disk systems **scale to a larger number of processors**, but communication between processors is slower (some *ms*).

# Shared Nothing

- **Node** consists of a processor, memory and one or more disks.
- Node function as **servers for the data on the disks** they own.
- **Communication** between processors through interconnection network.
- **Examples:** Teradata, Tandem, Oracle-n CUBE
- **Minimize interference** of resource sharing: data accessed from local disks (and local memory accesses) do not pass through interconnection network
- Can be scaled up to **thousands of processors** without interference.
- **Main drawbacks:**
  - cost of communication
  - cost of non-local disk access
  - sending data involves software interaction at both ends



# Hierarchical

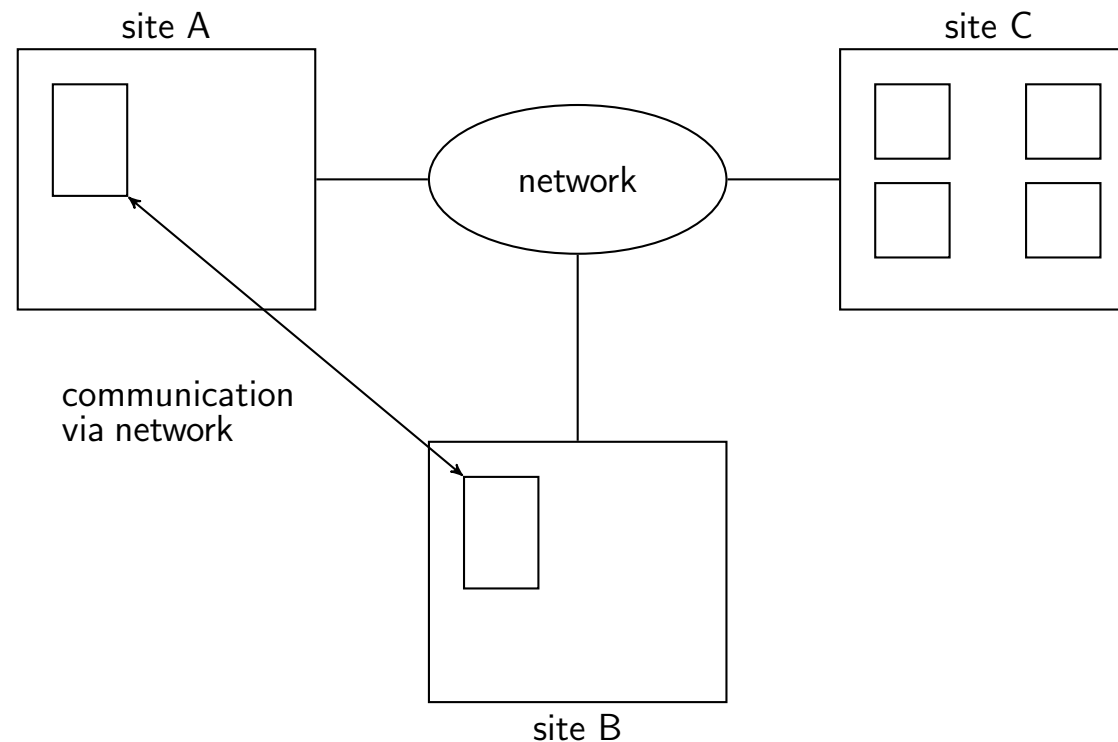
- Combines characteristics of shared-memory, shared-disk, and shared-nothing architectures.
- Top level is a shared-nothing architecture – nodes connected by an interconnection network, and do not share disks or memory with each other.
- Each node of the system could be a shared-memory system with a 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.
- Reduce the complexity of programming such systems by distributed virtual-memory architectures
  - Also called non-uniform memory architecture (NUMA)

# Outline

- 1 Centralized and Client-Server Systems
- 2 Server System Architecture
- 3 Parallel Systems
- 4 Distributed Systems**
- 5 Network Types

# Distributed Systems

- Data spread over multiple machines (also referred to as **sites** or **nodes**).
- Network interconnects the machines
- Data shared by users on multiple machines



# 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
- 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.

# Differences to Shared-Nothing Parallel Systems

- sites geographically separated
- sites separately administrated
- slower interconnection between sites
- differentiation between local and global transactions

# Trade-offs in Distributed Systems

- **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.
- **Disadvantage: added complexity required to ensure proper coordination among sites.**
  - Software development cost
  - Greater potential for bugs
  - Increased processing overhead

# Implementation Issues for Distributed Databases

- **Atomicity** needed even for transactions that update data at multiple sites
- The **two-phase commit protocol** (2PC) is used to ensure atomicity
  - Basic idea: each site executes transaction until just before commit, and then leaves final decision to a coordinator
  - Each site must follow decision of coordinator, even if there is a failure while waiting for coordinators decision
- 2PC is not always appropriate: other transaction models based on **persistent messaging** or **workflows** are also used
- **Distributed concurrency control** (and deadlock detection) required
- Data items may be **replicated** to improve data availability

# Outline

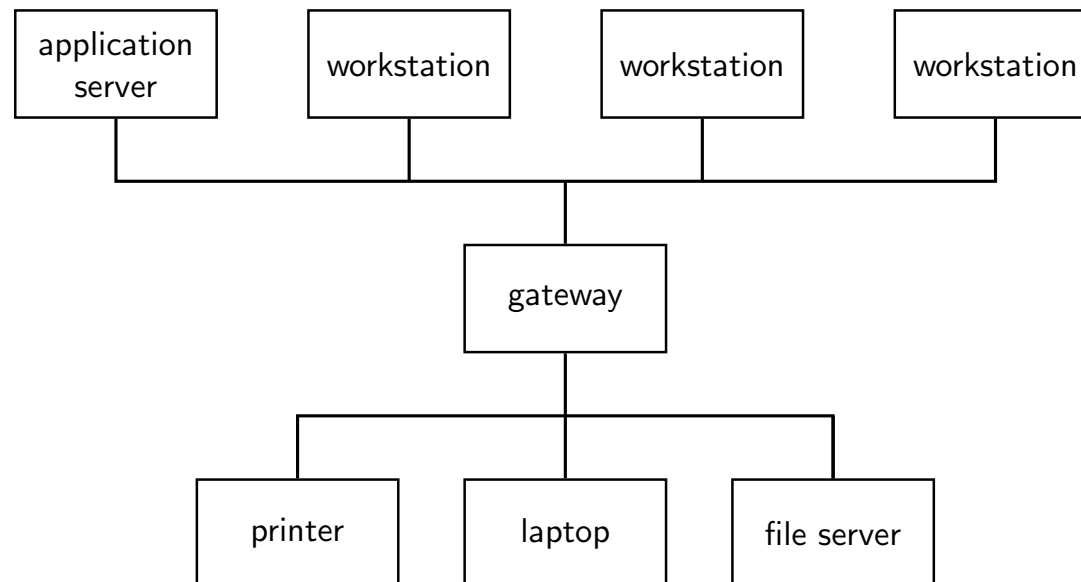
- 1 Centralized and Client-Server Systems
- 2 Server System Architecture
- 3 Parallel Systems
- 4 Distributed Systems
- 5 Network Types



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

# Local-Area Network/1



# Local-Area Network/2

- **Link technology:** twisted pair, coaxial cable, fiber optics, wireless connection
- **Ethernet:** Specification for computer networks
  - Software (e.g., protocols)
  - Hardware (e.g., cables, network cards, switches)
- **Transfer rates**
  - Fast Ethernet: 1, 10, 100 Mb/s (1 Mb/s =  $10^6$  bits / second )
  - Gigabit Ethernet: 1 Gb/s
  - Widely used: 10 Gb/s, highest transfer rate: 400 Gb/s
  - Higher transfer rates (1 Tb/s) require new technologies
- **Distances:**
  - usually single building or neighboring buildings
  - up to 70km with fiber optics

# Wide-Area Network/1

- Fast wide-area links (fiber optics, satellite channel): hundreds of gigabits
- Last link typically slower (e.g., cable modem, wireless connection): some megabits
- **Latency** higher than in LAN (ping, traceroute)
  - speed of light delay
  - queuing delay at routers

# Wide-Area Network/2

- WANs with **continuous connection** (e.g., the Internet) are needed for implementing distributed database systems.
- Groupware applications such as Lotus notes can work on WANs with **discontinuous connection**:
  - Data is replicated.
  - Updates are propagated to replicas periodically.
  - Copies of data may be updated independently.
  - Non-serializable executions can thus result. Resolution is application dependent.