

Parallel and Distributed Data Management

Database System Architectures

Nikolaus Augsten

`nikolaus.augsten@plus.ac.at`
Department of Computer Science
University of Salzburg



Sommersemester 2026

Version 22. Mai 2026

Outline

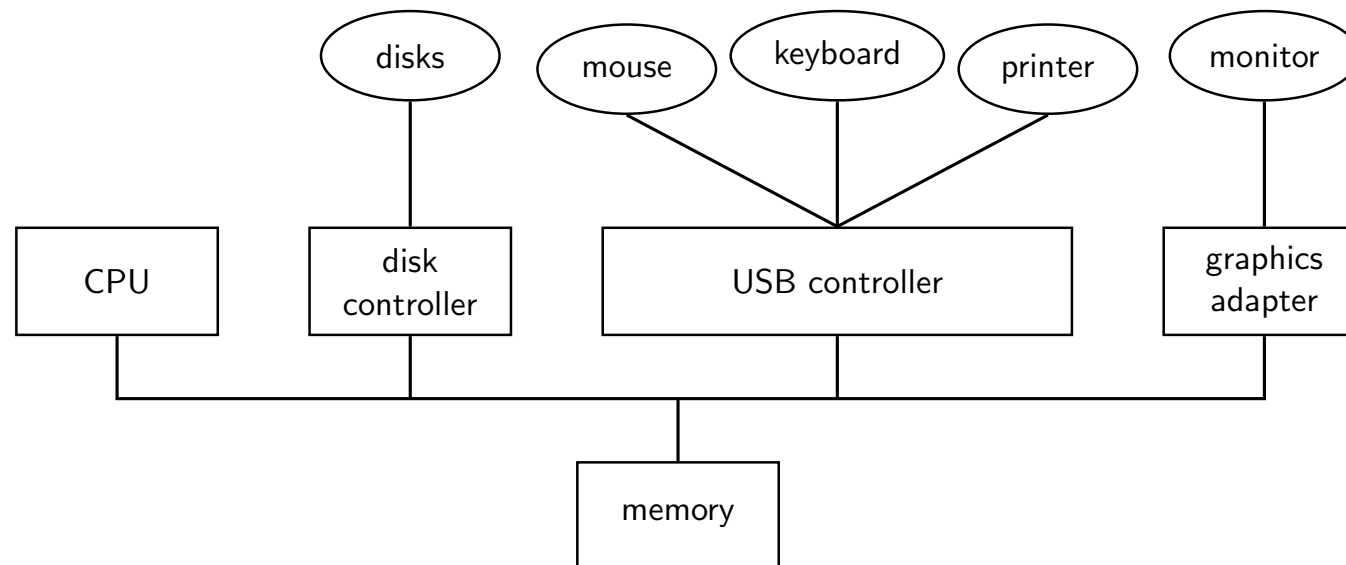
- ① Centralized and Client-Server Systems
- ② Server System Architecture
- ③ Parallel Systems
 - Performance Measures
 - Interconnection Networks
 - Parallel Database System Architecture
- ④ Distributed Systems

Outline

- 1 Centralized and Client-Server Systems
- 2 Server System Architecture
- 3 Parallel Systems
 - Performance Measures
 - Interconnection Networks
 - Parallel Database System Architecture
- 4 Distributed Systems

Centralized Database Systems

- Run on a **single, centralized computer system** that does not interact with other computer systems.
- A centralized computer system may run **single-user** or **multi-user** systems.



Single-User and Multi-User Systems

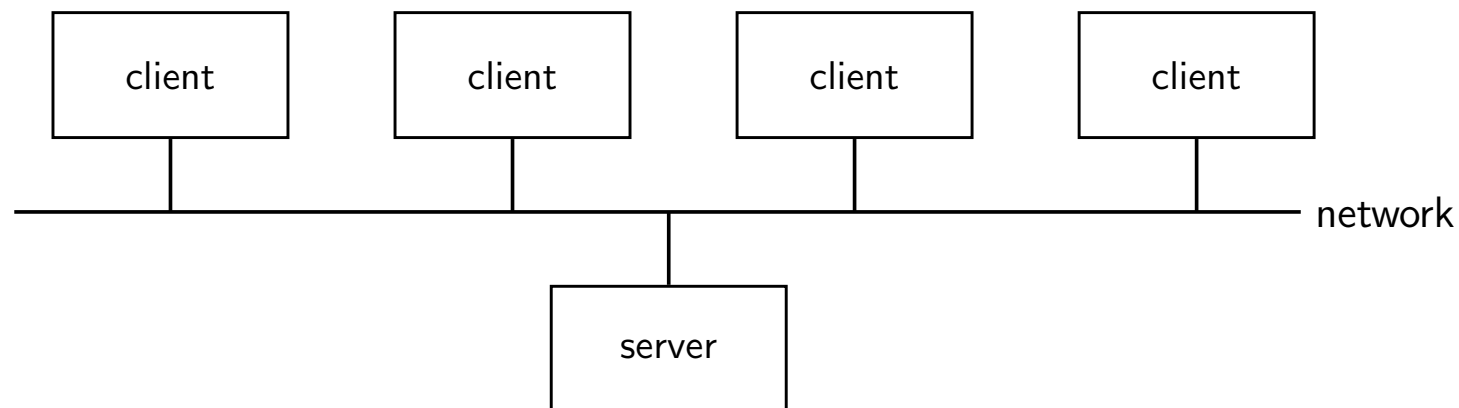
- **Single-user system** (e.g., smartphone or personal computer): single user, usually has only one CPU (with multiple cores) and one or two disks; the OS may support only one user.
- **Multi-user system**: more disks, more memory, multiple CPUs, and a multi-user OS. Serve a large number of users who are connected to the system remotely. Often called **server systems**.

Embedded Databases

- Databases on single-user systems may come with limited functionality:
 - simple concurrency control schemes
 - basic (e.g., copy before update) or no recovery mechanisms
 - provide API instead of declarative interface like SQL
- Such systems are called **embedded databases** and are typically linked to a single application (e.g., SQLite, DuckDB, LevelDB, BerkleyDB).

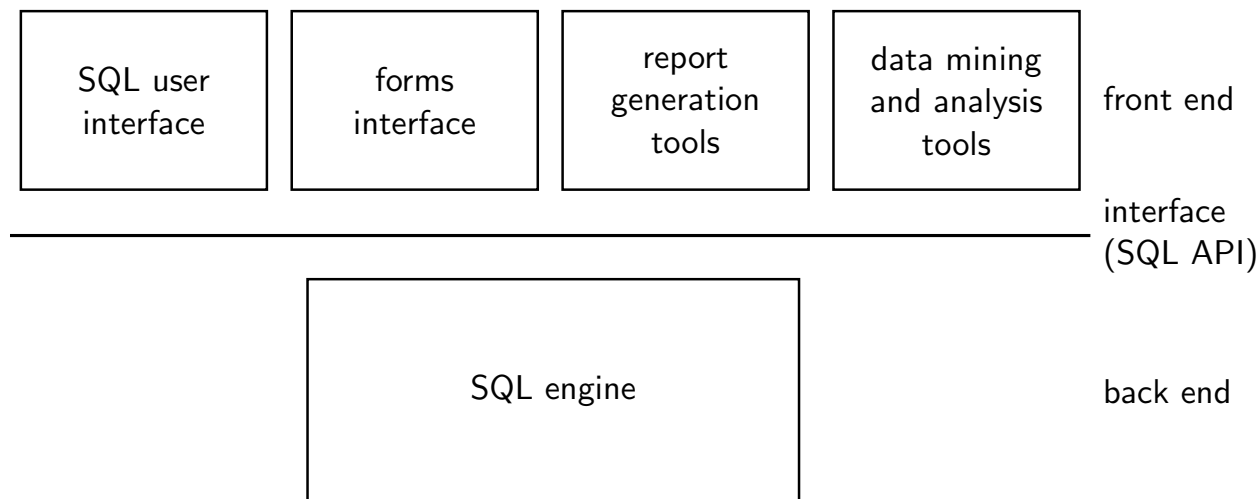
Client-Server Systems/1

- Server systems satisfy requests generated at client systems.



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.



Outline

- 1 Centralized and Client-Server Systems
- 2 Server System Architecture
- 3 Parallel Systems
 - Performance Measures
 - Interconnection Networks
 - Parallel Database System Architecture
- 4 Distributed Systems

Server System Architecture

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

Transaction Servers

- Also called **query server** or **SQL server**:
 - 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 API (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.

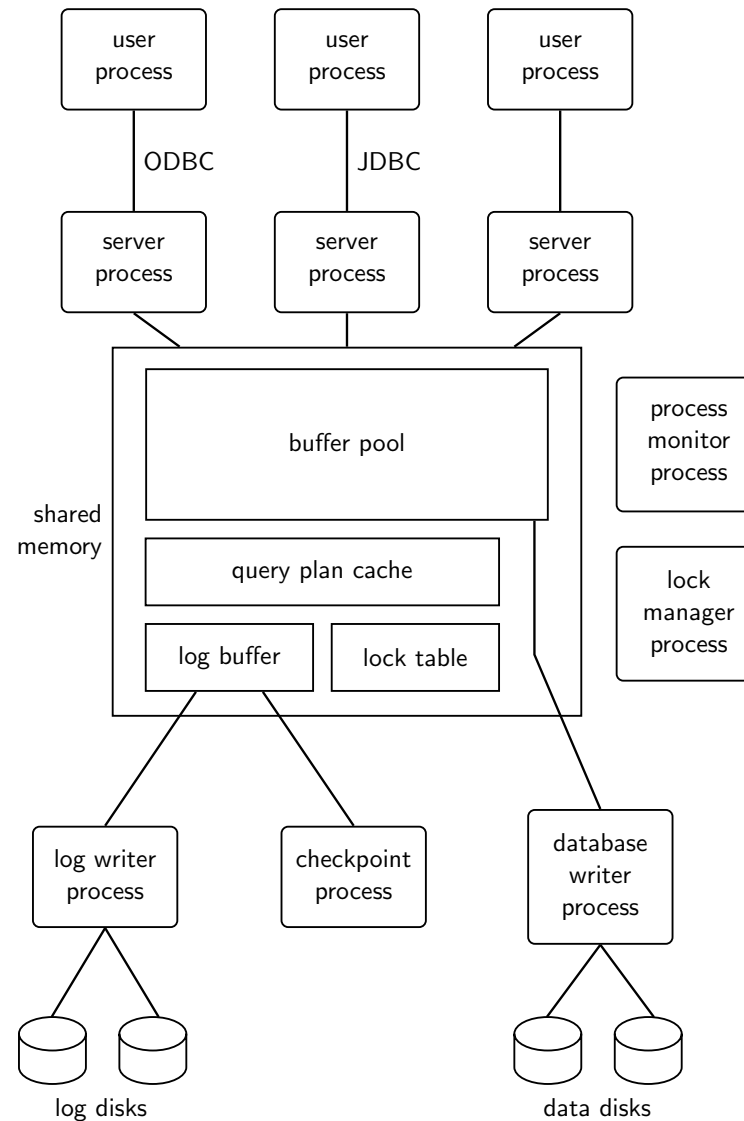
Transaction Server Process Structure/1

- 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 run concurrently (e.g., one multithreaded process per user session)
- **Lock manager process**
 - grants and releases locks, detects deadlocks
- **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 a 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 them fails
e.g., abort transaction of a failed server process and restart process

Transaction Server Process Structure/3



Transaction Server Process Structure/4

- All database processes can access **shared data**:
 - buffer pool
 - lock table
 - log buffer
 - cached query plans (reuse plan if same query is submitted again)
- To avoid two processes accessing the same data structure at the same time, databases systems implement **mutual exclusion** using either
 - operating system **semaphores** (wait and signal operations)
 - **atomic instructions** (test-and-set or compare-and-swap)
- To avoid overhead of **message passing** (sending requests to lock manager) for lock request/grant, database processes may operate directly on the **lock table**
- **Lock manager process** still used for deadlock detection

Data Servers

- Originally developed for **object-oriented databases**:
 - create, retrieve, and update persistent objects
 - persistent objects are accessed like main memory objects in programming languages
- All **computations performed on client**:
 - server ships required data items to client
 - client performs compute intensive tasks on data items
 - updated or new data items are shipped from client to server
- **Server** only needs to **store and fetch data**.
- Data servers are **typically used** when
 - the client performs **extensive computations**, e.g., a CAD system fetches a computer chip model and runs simulations
 - client and server are connected via **high-speed network**

Caching at Clients/1

- Client and server **communicate via network**:
 - **network latency** (also network round-trip time) is the time to send a message over a network and get the response back
 - much **slower than local memory** references, e.g., milliseconds vs. 100 nanoseconds even in LAN (local are network)
- **Optimization strategies** to reduce the effect of network latency:
 - prefetching: send a data item before it is requested
 - data caching: client caches data received from server for future use
 - lock caching: client keeps the lock also after accessing the data
 - adaptive lock granularity: use coarse- and fine-grained locks to balance number of lock requests and lock contention

Caching at Clients/2

- Prefetching

- network latency is per request: similar for large and small messages
- sending one item at a time has a large overhead
- prefetching sends also data items that are not requested, but are likely to be used in the near future

- 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 coherence](#))
- check can be done when requesting lock on data item

Caching at Clients/3

- Lock caching
 - requesting and granting a lock requires a network round trip
 - 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
 - works well when data is partitioned among clients, i.e., two different clients rarely request lock on the same data item

Caching at Clients/4

- Adaptive lock granularity

- multi-granularity locking: locks not only on individual data items (fine granularity), but also on pages, tables, etc. (coarse granularity)
- avoid large number of locks, e.g., get a single page lock instead of multiple item locks on that page
- coarse-granularity locks decrease number of locks but increase lock contention (i.e., transactions have to wait for a lock)
- **lock de-escalation** adaptively decreases the lock granularity when there is lock contention:
 1. server sends de-escalation request to client
 2. client requests finer-granularity locks
 3. when finer-granularity locks are granted, coarse-granularity lock is released

Outline

- 1 Centralized and Client-Server Systems
- 2 Server System Architecture
- 3 **Parallel Systems**
 - Performance Measures
 - Interconnection Networks
 - Parallel Database System Architecture
- 4 Distributed Systems

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:

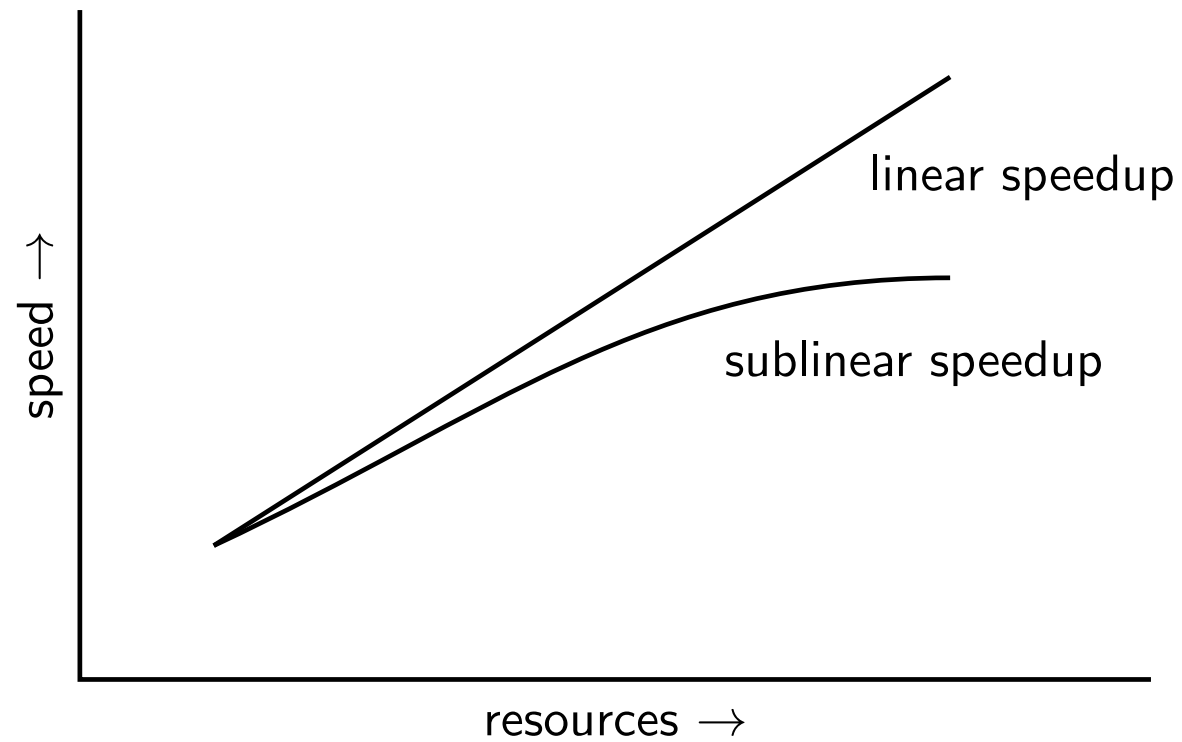
$$\text{speedup} = \frac{\text{small system elapsed time}}{\text{large system elapsed time}}$$

- 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:

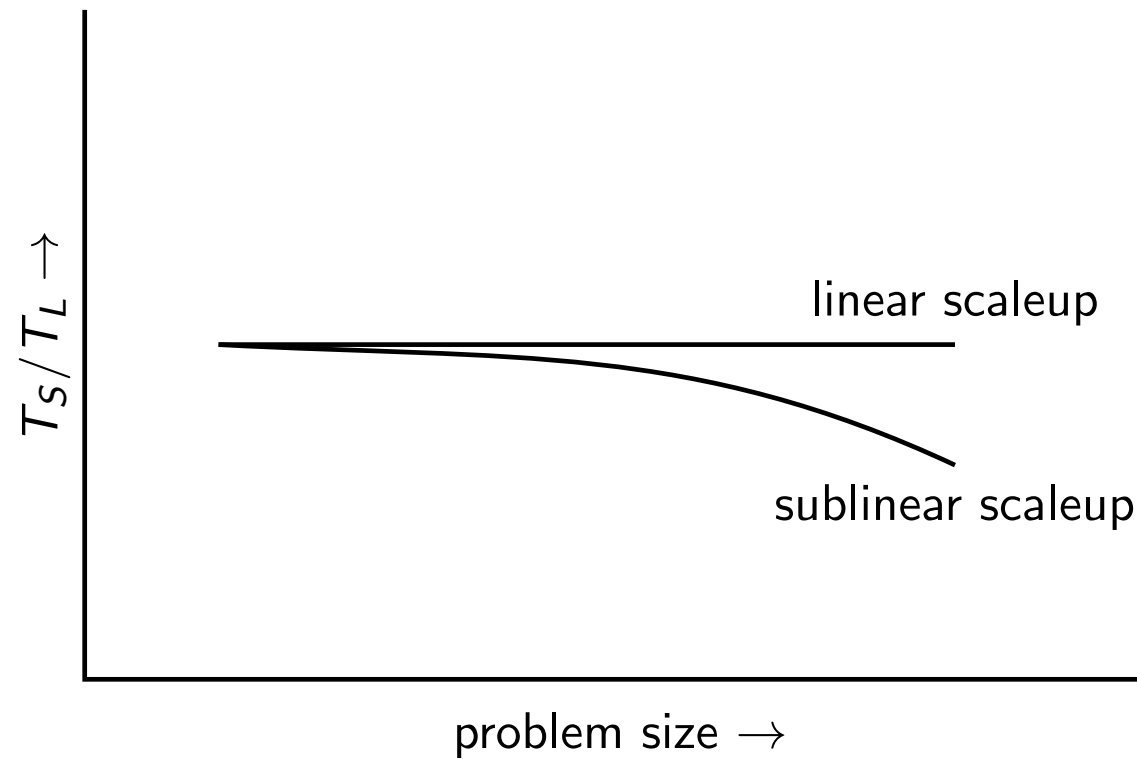
$$\text{scaleup} = \frac{\text{small system small problem elapsed time}}{\text{big system big problem elapsed time}}$$

- Scaleup is **linear** if equation equals 1.

Speedup



Scaleup



Batch and Transaction Scaleup

- **Batch scaleup**: single large job.
 - typical for decision support queries and scientific simulations
 - 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 for 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

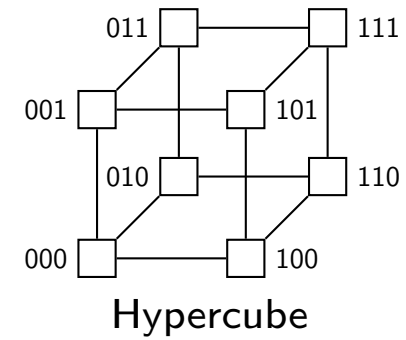
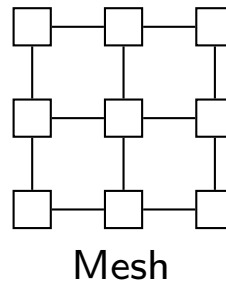
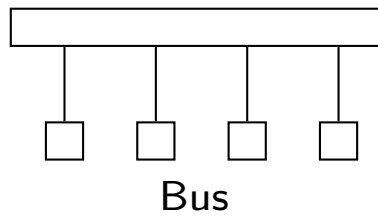
Speedup and scaleup are often sublinear due to:

- **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 tasks executing in parallel. Overall execution time determined by slowest of tasks executing in parallel.

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:
 - number communication links grow with growing number of components, and so scales better
 - the number of hops to send message to a node is proportional to $\sqrt{n}$
- **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
 - can reach each other via at most $\log(n)$ links
 - reduces communication delays

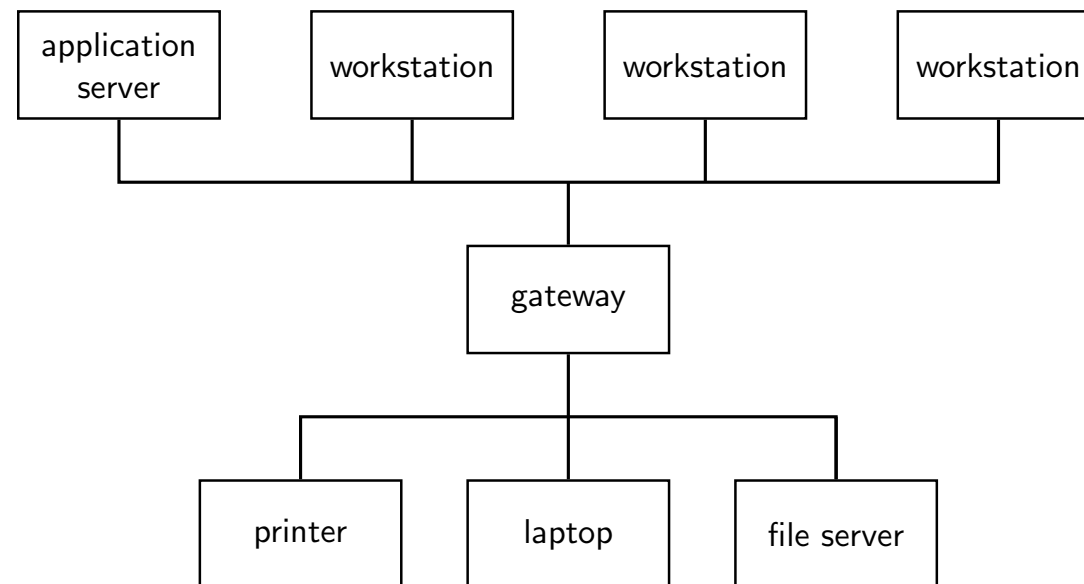
Interconnection Architectures



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 ($1\text{Mb/s} = 1\text{Mbps} = 10^6 \text{ bits/second}$)
 - 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

- Fast wide-area links (fiber optics, satellite channel):
 - hundreds of GB/s
- Last link typically slower (e.g., cable modem, wireless connection):
 - tens or hundreds of Mb/s
- **Latency** higher than in LAN
 - speed of light delay
 - queuing delay at routers
- WANs with **continuous connection** (e.g., the Internet) are needed for implementing distributed database systems.

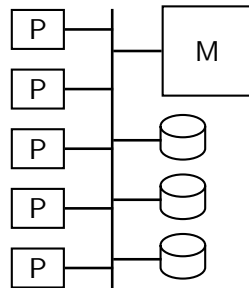
Example: Network Latency

- What are the number of routers and the **network latency** for
 - localhost (i.e., network connection to the local machine)
 - in the LAN (e.g., ssh.cosy.sbg.ac.at)
 - in the WAN (e.g., www.tum.de, unsw.edu.au)
- Use **ping** to measure latency and **traceroute** to learn how the network packets are routed

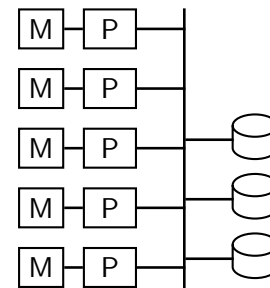
Parallel Database Architectures

- **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

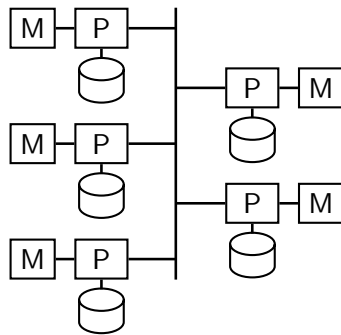
Parallel Database Architectures



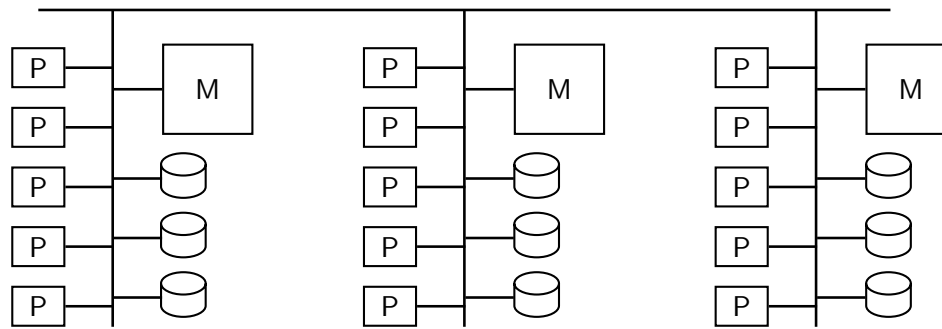
shared memory



shared disk



shared nothing



hierarchical

Shared Memory

- Processors have access to a **common memory** via bus or interprocessor communication network.
- Extremely **efficient communication** between processors ($< 1\mu s$) — data in shared memory can be accessed by any processor.
- **Memory bus** becomes a bottleneck since only a single processor at a time can use the bus.
- **NUMA** (non-uniform memory access):
 - each processor has locally connected memory
 - processors can access memory of other processors through a high-speed **interprocessor communication network**
 - locally connected memory is faster
- **Does not scalable** beyond a few hundred cores:
 - limited by bus speed and number processors that can be interconnected
 - widely used for lower degrees of parallelism

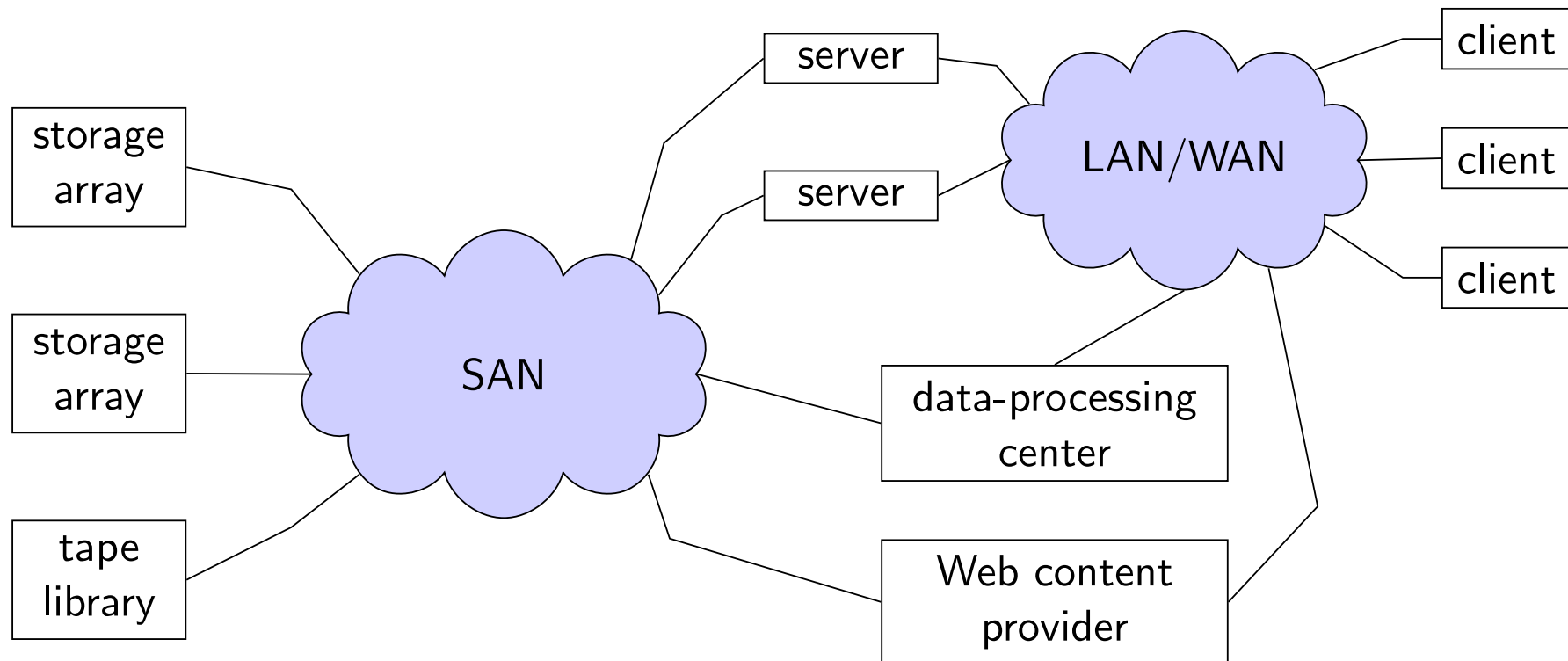
Shared Disk/1

- 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 Disk/2

- File server / NAS (Network Attached Storage)
 - disks connected via RAID controller
 - mounted as directory in file system
 - Samba
 - NFS - Network File System
- SAN - Storage Array Network
 - block level access
 - appears to be locally attached block device
 - shared disk file system runs on top of SAN
 - IBM GPFS (General Parallel FS)
 - Oracle Cluster FS
 - Lustre (mainly super computing/Linux)

Storage Area Network



Shared Nothing

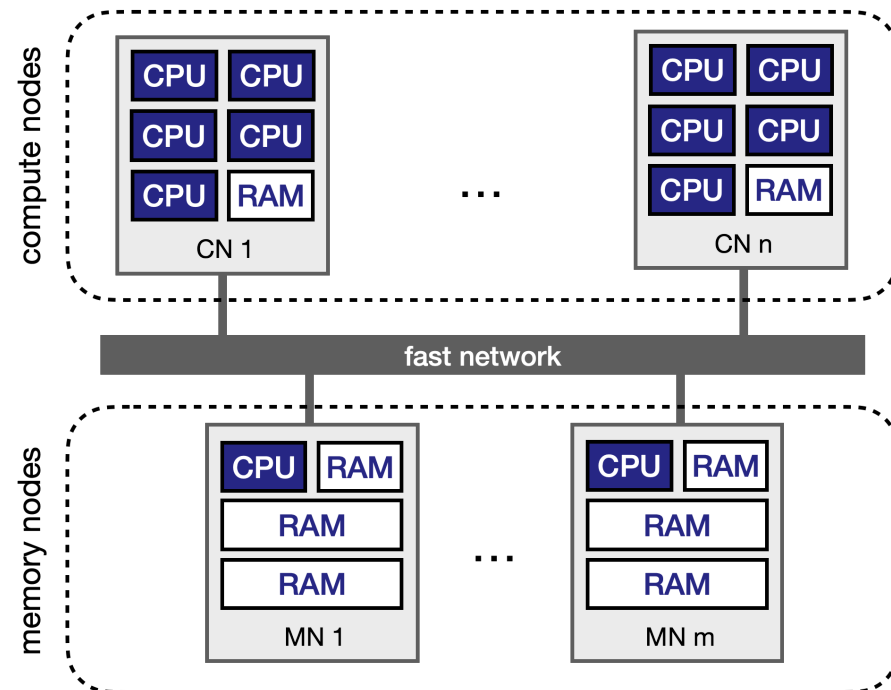
- **Node** consists of a processor, memory, and one or more disks.
- Nodes function as **servers for the data on the disks** they own.
- **Communication** between processors through interconnection network.
- **Examples:** Teradata, Tandem, Oracle-n CUBE
- **Minimize interference** of resource sharing: 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

Disaggregated Memory Architecture

- compute nodes: strong CPUs, small memory (cache)
- memory nodes: large memory, wimpy CPU
- fast network: low latency and high bandwidth

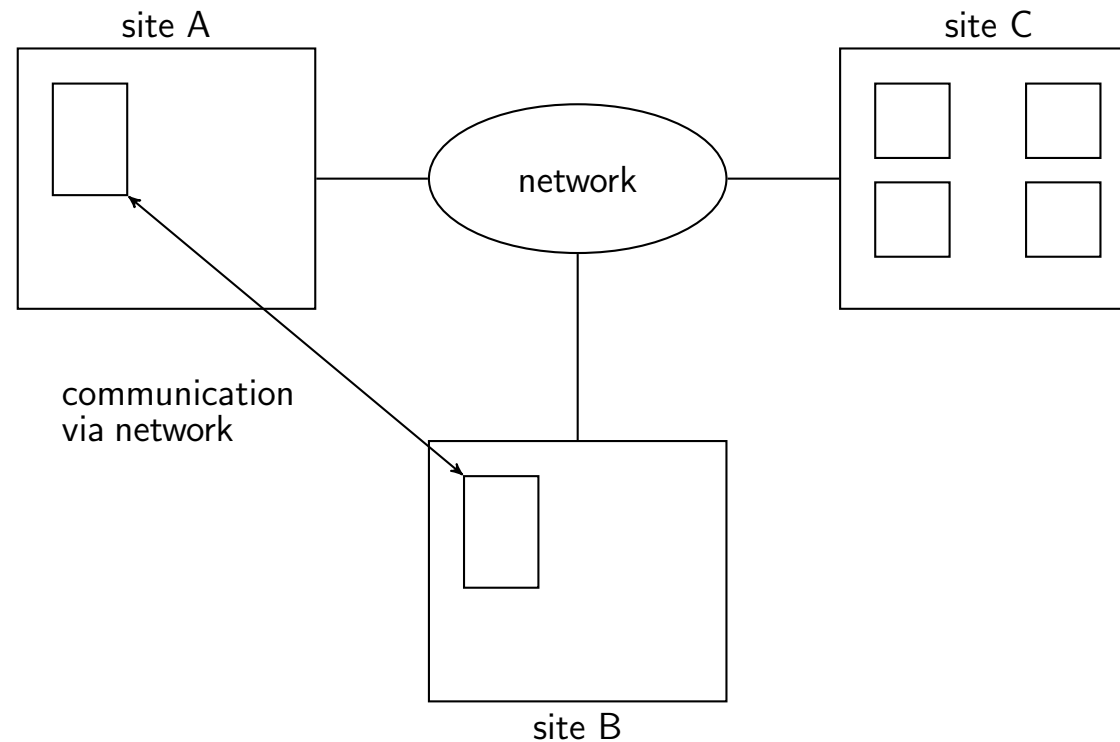


Outline

- 1 Centralized and Client-Server Systems
- 2 Server System Architecture
- 3 Parallel Systems
 - Performance Measures
 - Interconnection Networks
 - Parallel Database System Architecture
- 4 Distributed Systems

Distributed Systems

- Data spread over multiple machines (called **sites** or **nodes**).
- Network interconnects the 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

Differences to Shared-Nothing Parallel Systems

- Sites **geographically separated**.
- Sites may be **separately administrated**.
- Slower and less reliable **interconnection between sites**.
 - higher latency, lower bandwidth
 - greater potential for network failure (network partitioning)
- 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.

Trade-offs in Distributed Systems

- **Sharing data:** users can access data residing at some other sites (heterogeneous distributed databases)
- **Autonomy:** each site retains a degree of control over data stored locally (heterogeneous distributed databases)
- **Higher system availability through redundancy:** data can be replicated at remote sites, and system can function even if a site fails.
- **Disadvantage:** proper coordination among sites adds complexity.
 - software development cost
 - greater potential for bugs
 - increased processing overhead

Implementation Issues for Distributed Databases

- **Atomicity** 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** and **workflows** are also used
- **Distributed concurrency control** (and deadlock detection) required
- Data items may be **replicated** to improve data availability

Conclusion

- Homogeneous vs. heterogeneous distributed database systems.
- Distributed database different from a shared nothing parallel systems.
- Geographical separation of sites comes with opportunities and challenges:
 - higher availability through geographically distributed redundancy
 - new implementation challenges