

B0B36DBS, BD6B36DBS: **Database Systems**

<http://www.ksi.mff.cuni.cz/~svoboda/courses/192-B0B36DBS/>

Lecture 11

# Database Architectures

Authors: **Tomáš Skopal, Irena Holubová**

Lecturer: **Martin Svoboda**, [martin.svoboda@fel.cvut.cz](mailto:martin.svoboda@fel.cvut.cz)

28. 4. 2020

**Czech Technical University in Prague**, Faculty of Electrical Engineering

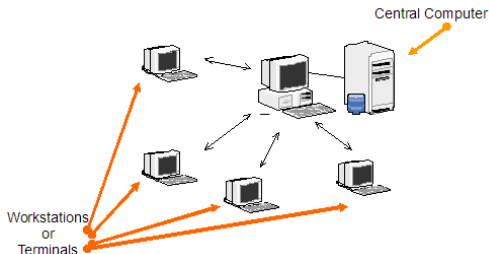
# Today's Lecture Outline

- architectures of database systems
  - centralized systems
  - client – server systems
  - parallel systems
  - distributed systems
- types of queries

# Centralized Systems

- run on a single computer system
- do not interact with other computer systems
- **general-purpose computer system**
  - one to a few CPUs and a number of device controllers
  - connected through a common bus
    - provides access to a shared memory
- **single-user system** (e.g., personal computer or workstation)
  - **desk-top unit**, single user, usually has one or two CPUs 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**

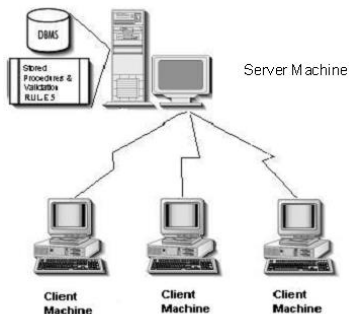
# Multi-User Systems



# Client-Server Systems

- server systems satisfy requests generated at  $m$  client 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

# Client-Server Systems



# Front-End vs. Back-End

- 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
- interface between the front-end and the back-end:
  - SQL
  - application program interface

# Parallel 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
  - **latency** (response time) – the amount of time it takes to complete a single task from the time it is submitted



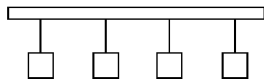
# Parallel Systems

- **speed-up**: a fixed-sized problem executing on a small system is given to a system which is **N**-times larger (more efficient)
- **scale-up**: increase the size of both the problem and the system
  - **N**-times larger system used to perform **N**-times larger job
- both often sub-linear due to:
  - Start-up costs: Cost of **starting up multiple processes** > 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  $\Rightarrow$  spend time waiting on other processes rather than performing useful work
  - Skew: Increasing the degree of parallelism increases the variance in service times of tasks executed in parallel
    - Overall execution time is **determined by the slowest of executing tasks**

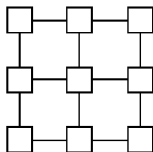
# Interconnection Architectures

- **Bus:** components send data on and receive data from a single communication bus
  - cons: does not scale well with increasing parallelism
- **Mesh:** components are arranged as nodes in a grid, and each component is connected to adjacent components
  - pros: communication links grow with growing number of components
    - scales better
  - cons: may require  $2\sqrt{n}$  hops to send message to a node
- **Hypercube:** components are numbered in binary representation  $\Rightarrow$  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

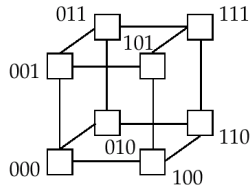
# Interconnection Architectures



(a) bus



(b) mesh

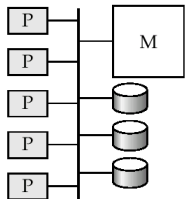


(c) hypercube

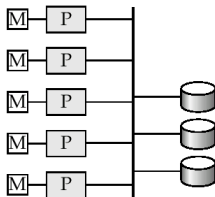
# Parallel (Database) Architectures

- **Shared memory** – processors share a common memory
  - **efficient communication** between processors
  - not scalable much
    - the bus or the interconnection network becomes a bottleneck
- **Shared disk** – processors share a common disk
  - a degree of **fault tolerance** – if a processor fails, other processors can take over its tasks
    - data are accessible from all processors
  - bottleneck = interconnection to the disk
- **Shared nothing** – processors share neither a common memory nor common disk
  - processors communicate using an interconnection network
  - drawback: cost of communication and non-local disk access
- **Hierarchical** – combination of the above architectures
  - top level is a shared-nothing
  - each node of the system could be a shared-memory sub-system

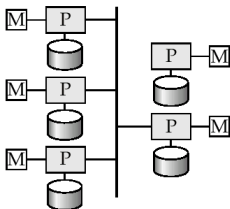
# Parallel Database Architectures



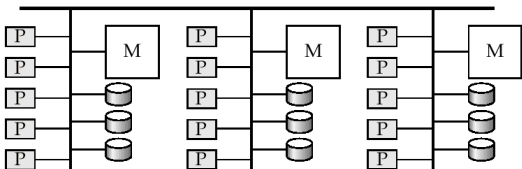
(a) shared memory



(b) shared disk



(c) shared nothing



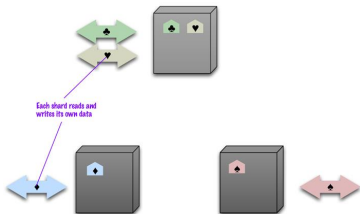
(d) hierarchical

# Distributed Systems

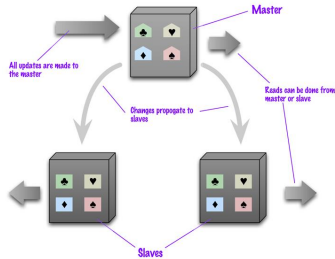
- **scale-out**: data are **distributed** (spread) over multiple machines = nodes
- data are **replicated**
  - system can work even if a node fails
- **homogeneous** distributed databases
  - same software/schema on all nodes, data may be partitioned among nodes
  - goal: provide a view of a single database, hiding details of distribution
- **heterogeneous** distributed databases
  - different software/schema on different nodes
  - goal: integrate existing databases to provide useful functionality

# Distribution Models

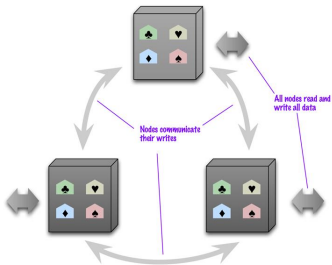
- **single server** – no distribution
- **sharding** – putting different parts of the data onto different servers
  - too many data to be stored on a single node
- **master/slave replication** – master provides reads/writes, slaves provide reads
  - no scalability of writes
- **peer-to-peer replication** – all replicas have equivalent weight
  - each node is a master
- often: **combination** of sharding and replication



sharding = **distribution**



master/slave **replication**



peer-to-peer **replication**



# Types of Queries

- **declarative**
  - we describe the **data we want**, but not how to get it
  - e.g., DRC, TRC
- **procedural**
  - we describe **how to get the data** we want
    - i.e., what operations should be done
  - e.g., relational algebra (partially)
- SQL has both the features
- **QBE** (Query by Example)
  - graphical query language from mid 70-ies (IBM)
    - developed as an alternative to SQL
  - many graphical front-ends for databases re-use the idea today

# QBE

Sailors (sid: integer, sname: string, rating: integer, age: real)

Boats (bid: integer, bname: string, color: string)

Reserves (sid: integer, bid: integer, day: dates)

Sailors	sid	sname	rating	age
P.			10	

Sailors with rating 10

Sailors	sid	sname	rating	age
		P..N		P..A

Names and ages of all sailors

Sailors	sid	sname	rating	age	Reserves	sid	bid	day
	_Id	P..S		> 25		_Id		'8/24/96'

Sailors who have reserved a boat for 8/24/96 and who are older than 25

Sailors	sid	sname	rating	age
	_Id			> 25

Reserves	sid	bid	day	Boats	bid	bname	color
	_Id	_B	'8/24/96'		_B	Interlake	P.

Colors of boats Interlake reserved by sailors who have reserved a boat for 8/24/96 and who are older than 25

