UNIVERSITÄT MANNHEIM

Database Technology
Database Architectures

Heiko Paulheim

Today

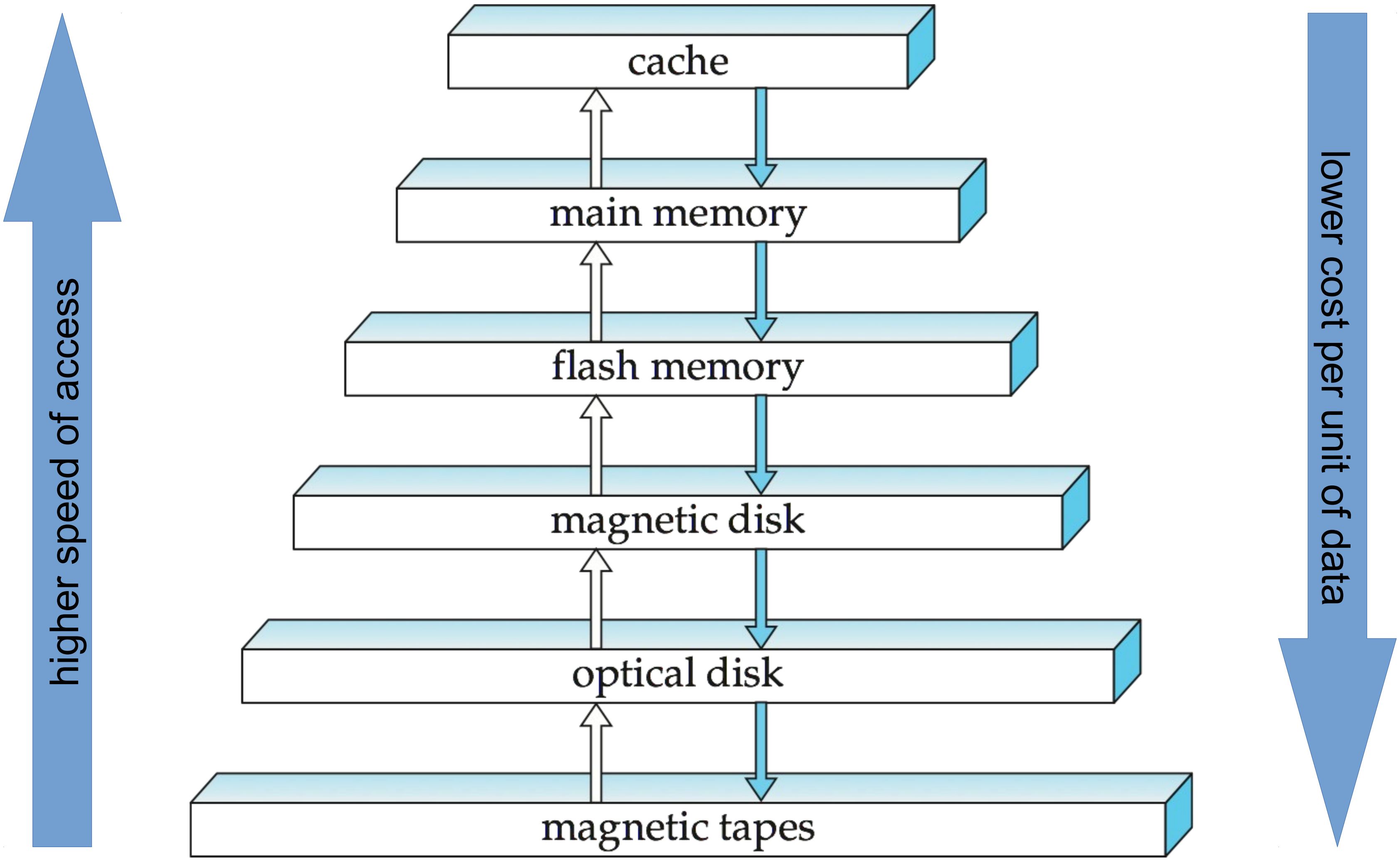
- So far, we have treated Database Systems as a “black box”
  - We can define a schema
  - ...and write data into it…
  - ...and read data from it

- Today
  - Opening the “black box”
  - How is data stored?
  - Architectures for larger database systems

Physical Data Storage

- A manifold of options
  - Hard disks, flash memory, magnetic tape, CDs, DVDs, BluRays, …
- Considerations
  - Speed with which data can be accessed
  - Cost per unit of data
  - Reliability
    - data loss on power failure or system crash
    - physical failure of the storage device
  - Can differentiate storage into:
    - **volatile storage:** loses contents when power is switched off
    - **non-volatile storage**:
      - Contents persist even when power is switched off
      - secondary & tertiary storage, battery backed up main-memory

Storage Hierarchy

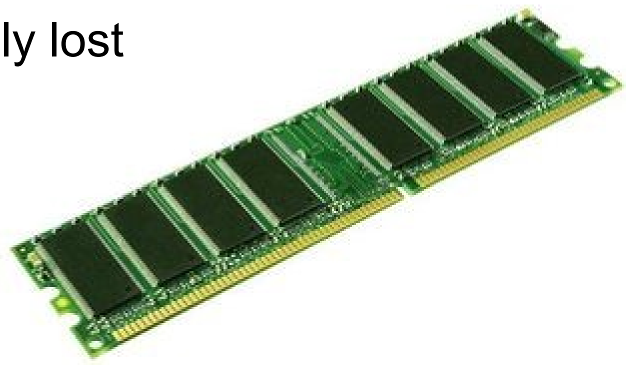


Storage Hierarchy

- **primary storage**: Fastest media but volatile (cache, main memory)
  - data on which the processor operates
- **secondary storage**: next level in hierarchy, non-volatile, moderately fast access time
  - also called **on-line storage**
  - e.g., flash memory, magnetic disks
  - needs to be loaded in memory for processing
- **tertiary storage**: lowest level in hierarchy, non-volatile, slow access time
  - also called **off-line storage**
  - e.g., magnetic tape, optical storage
  - typically used for backup

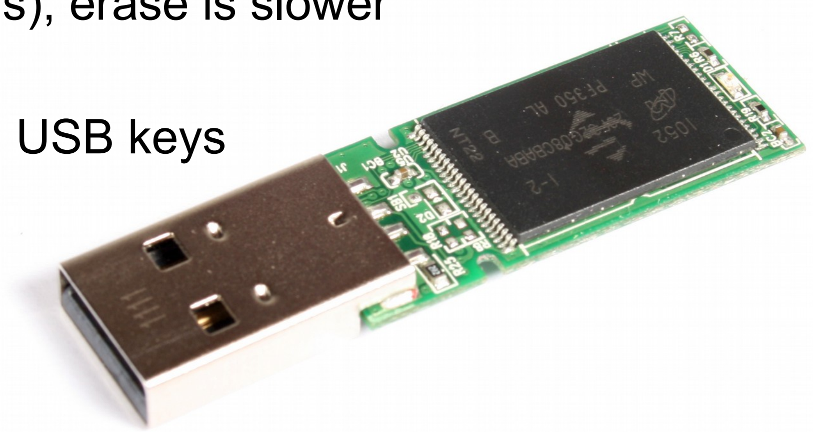
Physical Storage

- **Cache**
  - fastest and most costly form of storage; volatile; managed by the computer system hardware
- **Main memory**
  - fast access (10s to 100s of nanoseconds (1 ns = 10^{-9} seconds)
  - generally too small (or too expensive) to store the entire database
    - typically: gigabyte capacity
    - capacities have gone up and per-byte costs have decreased steadily and rapidly (roughly factor of 2 every 2 to 3 years)
  - **Volatile** — contents of main memory are usually lost if a power failure or system crash occurs



Physical Storage

- **Flash memory**
  - Data survives power failure
  - Data can be written at a location only once, but location can be erased and written to again
    - Can support only a limited number (10K – 1M) of write/erase cycles
    - Erasing of memory has to be done to an entire bank of memory
  - Reads are roughly as fast as main memory
  - But writes are slow (few microseconds), erase is slower
  - Widely used in embedded devices
    such as digital cameras, phones, and USB keys

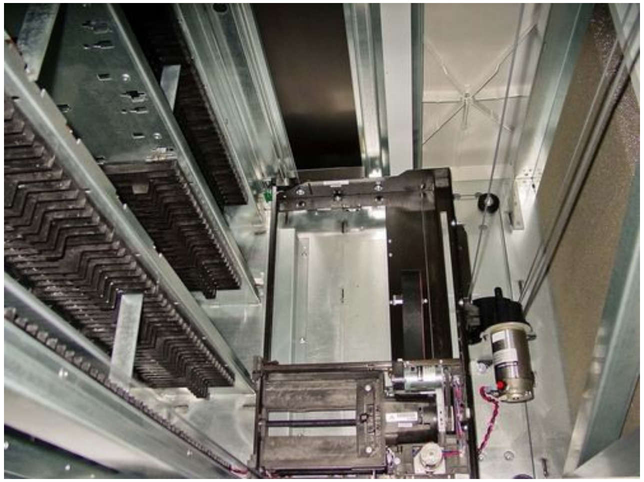


Physical Storage

- **Magnetic disk (hard disk)**
  - Data is stored on spinning disk, and read/written magnetically
  - Primary medium for the long-term storage of data
  - typically stores entire database
  - Data must be moved from disk to main memory for access, and written back for storage
    - Much slower access than main memory
  - **direct-access** – possible to read data on disk in any order, unlike magnetic tape
  - terabyte sized
    - Much larger capacity and and lower cost/byte than (flash) memory
    - Growing constantly and rapidly with technology improvements (factor of 2 to 3 every 2 years)
  - Survives power failures and system crashes
    - disk failure can destroy data, but is rare

Physical Storage

- **Optical storage**
  - non-volatile, data is read optically from a spinning disk using a laser
  - CD-ROM (640 MB), DVD (4.7 to 17 GB), Blu-ray (27 to 54 GB)
  - Write-once, read-many (WORM) optical disks for archival storage
    - Multiple write versions also available (CD-RW, DVD-RW, DVD+RW, and DVD-RAM)
  - Reads and writes are slower than with magnetic disk
- **Juke-box** systems
  - for storing large volumes of data
  - large numbers of removable disks
  - a few drives
  - mechanism for automatic loading/unloading of disks

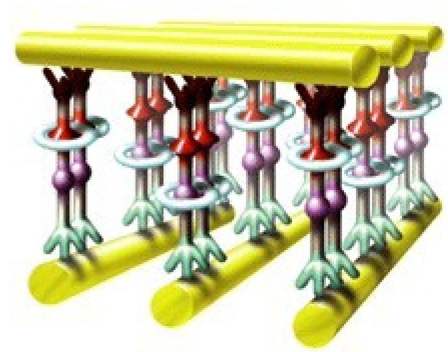


Physical Storage

- **Tape storage**
  - non-volatile, used primarily for backup (to recover from disk failure), and for archival data
  - **sequential access** – much slower than disk
  - very high capacity (terabyte scale)
  - tape can be removed from drive
    - storage costs much cheaper than disk, but drives are expensive
  - Tape jukeboxes available
    for storing massive amounts of data

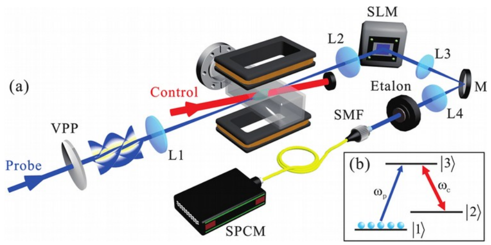
Physical Storage

- Modern, experimental and exotic trends
- Molecular memory
  - bits are stored as charge of single molecules
  - using polymer molecules for storage
  - experimental state (NASA, Hewlett Packard…)



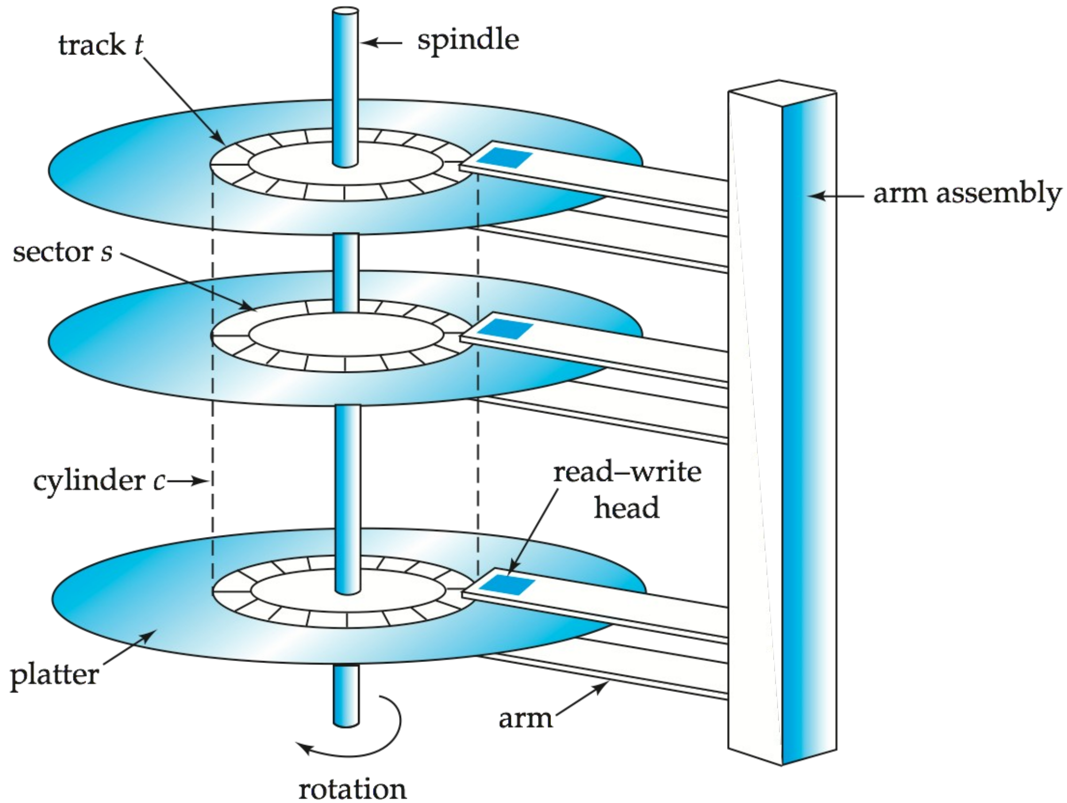
- DNA storage
  - idea: DNA stores information
    (i.e.: genetic instructions)
  - synthesizing DNA for data storage
  - in theory, 1g of DNA can store 215 PB

- Quantum Storage
  - Light photons have to be “caught”
    between mirrors



Anatomy of a Hard Disk Drive

- Schematic view
  - sectors are the smallest unit to be read or written
  - also called *blocks*
- Goal for storage
  - minimize number of blocks transferred



File Organization

- The database is stored as a collection of *files*
  - each file is a sequence of *records*
  - each record is a sequence of *fields*
- Simple approach:
  - assume record size is fixed
  - each file has records of one particular type only
  - different files are used for different relations
  - This case is easiest to implement; will consider variable length records later

File Organization

- Simple approach:
  - Store record *i* starting from byte *n* ∗ (*i* – 1), where *n* is the size of each record
  - Record access is simple but records may cross disk blocks
    - Modification: do not allow records to cross block boundaries

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
~~record 3~~	~~22222~~	~~Einstein~~	~~Physics~~	~~95000~~
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

- Deletion of record *i:* alternatives*:*
  - move records *i* + 1, . . ., *n* to *i, . . . , n* – 1
  - move record *n* to *i*
  - do not move records, but link all free records on a *free list*

Record Deletion – Compacting

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Record Deletion – Moving Last Record

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

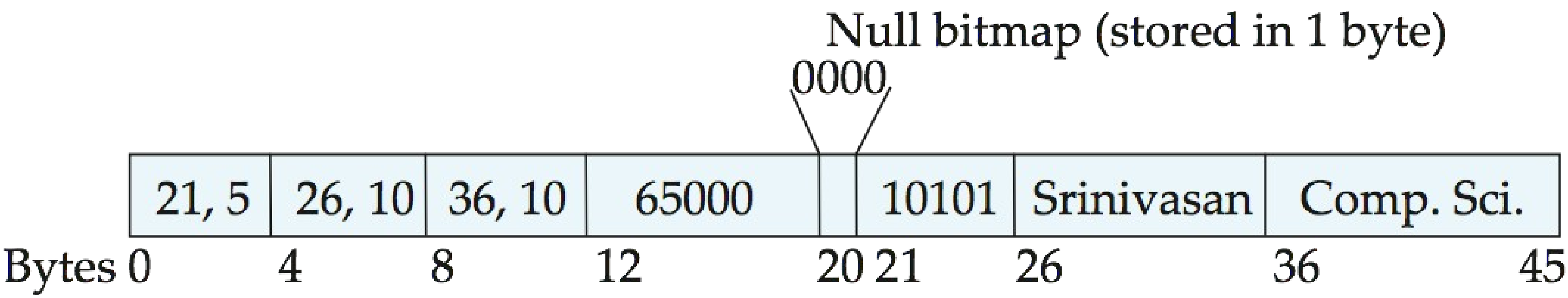
Record Deletion – Free Lists

- Store the address of the first deleted record in the file header
- Use this first record to store the address of the second deleted record, and so on
- Can think of these stored addresses as pointers since they “point” to the location of a record
- More space efficient representation:
  - reuse space for normal attributes of free records to store pointers
- Insertion:
  - find last deleted record and fill in data there
  - remove previous pointer

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

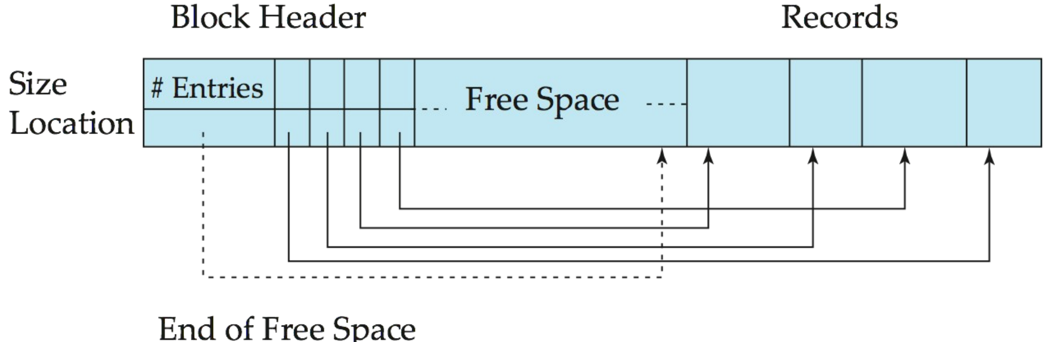
Storing Variable Length Records

- Variable-length records arise in database systems in several ways:
  - e.g., storage of multiple record types in a file
  - e.g., record types that allow variable lengths for one or more fields such as strings (**varchar**)
- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap



Storing Variable Length Records

- **Slotted page** header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record
- Records can be moved around within a page
  - to keep them contiguous with no empty space between them
  - entry in the header must be updated
- Pointers (e.g., foreign keys) should not point directly to record, but to entry for the record in header



Organization of Records in Files

- **Heap**
  - a record can be placed anywhere in the file where there is space
- **Sequential**
  - store records in sequential order, based on the value of the search key of each record
  - requires re-organizations
- **Hashing**
  - a hash function computed on some attribute(s) of each record
  - the result specifies in which block of the file the record should be placed
- Records of different relations
  - stored either in separate files
  - or: store related relations in one file (called: **multitable clustering file organization**)
    - Motivation: store related records on the same block to minimize I/O

Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Sequential File Organization

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

32222	Verdi	Music	48000	

Multitable Clustering File Organization

- Store several relations in one file using a **multitable clustering** file organization

department

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

multitable clustering of *department* and *instructor*

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

Multitable Clustering File Organization

- good for queries
  - involving *department* $\bowtie$ *instructor*
  - involving one single department (and its instructors)
  - involving only the *instructor* relation
- bad for queries involving only the *department* relation
- results in variable size records
- can add pointer chains to link records of a particular relation

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

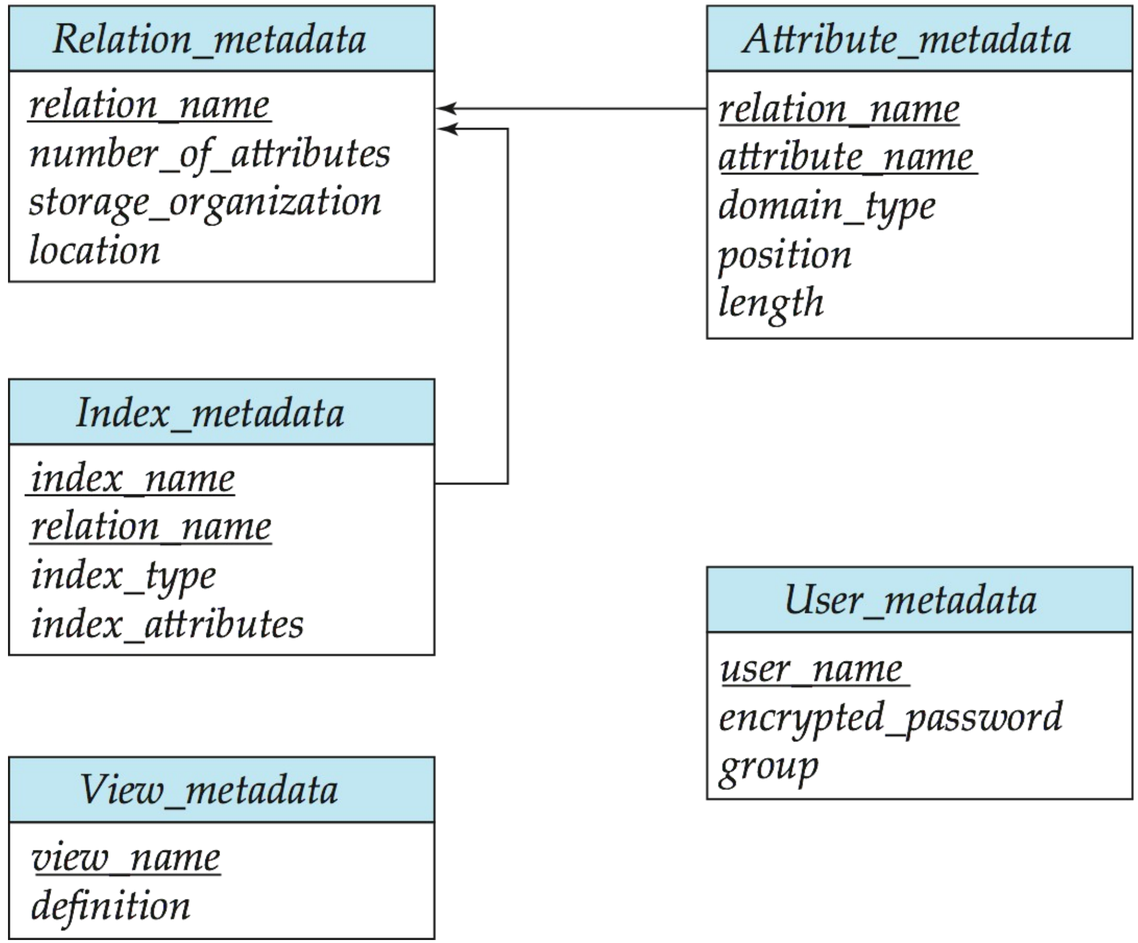
Data Dictionary Storage

The **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data, such as

- Information about relations
  - names of relations
  - names, types and lengths of attributes of each relation
  - names and definitions of views
  - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation
- Physical file organization information
  - How relation is stored (sequential/hash/…)
  - Physical location of relation
  - Information about indices

Data Dictionary Storage

- Many RDBMS use relations also for the data dictionary
- Those relations are typically held in memory for fast access
- Details may vary



Storage Access

- A database file is partitioned into fixed-length storage units called **blocks**
  - blocks are units of both storage allocation and data transfer
- Database system seeks to minimize the number of block transfers between the disk and memory
  - simple: by keeping as many blocks as possible in main memory
  - advanced: planning which blocks to keep in memory
- **Buffer** – portion of main memory available to store copies of disk blocks
- **Buffer manager** – subsystem responsible for allocating buffer space in main memory

Buffer Manager

- Programs call on the buffer manager when they need a block from disk
  - If the block is already in the buffer, buffer manager returns the address of the block in main memory
  - If the block is not in the buffer, the buffer manager
    - Allocates space in the buffer for the block
    - Replaces (i.e., removes) some other block, if required, to make space for the new block
      - If replaced block was changed: write back to disk
      - Read the block from the disk to the buffer
      - return the address of the block in main memory to requester

Potential for optimization

Buffer Replacement Strategies

- Most operating systems replace the block **least recently used** (LRU strategy):
  - use past pattern of block references as a predictor of future references
- Queries have well-defined access patterns (such as sequential scans), and a database system can use the information in a user's query to predict future references
  - LRU can be a bad strategy for certain access patterns involving repeated scans of data
    - Example: when computing the join of 2 relations r and s by a nested loops

      for each tuple *tr* of *r* do
        for each tuple *ts* of *s* do
          if the tuples *tr* and *ts* match …
  - Mixed strategy with hints on replacement strategy provided by the query optimizer is preferable

Buffer Replacement Strategies

- **Pinned block** – memory block that is not allowed to be replaced
- **Toss-immediate** strategy – frees the space occupied by a block as soon as the final tuple of that block has been processed
- **Most recently used (MRU) strategy** – system must pin the block currently being processed
  - After processing the final tuple, the block is unpinned
  - and it becomes the most recently used block.
- Buffer manager can use statistical information regarding the probability that a request will reference a particular relation
  - e.g., the data dictionary is frequently accessed.
    Heuristic: keep data-dictionary blocks in main memory buffer
- Buffer managers also support **forced output** of blocks for the purpose of recovery (coming back to this in a few weeks)

Database System Architectures

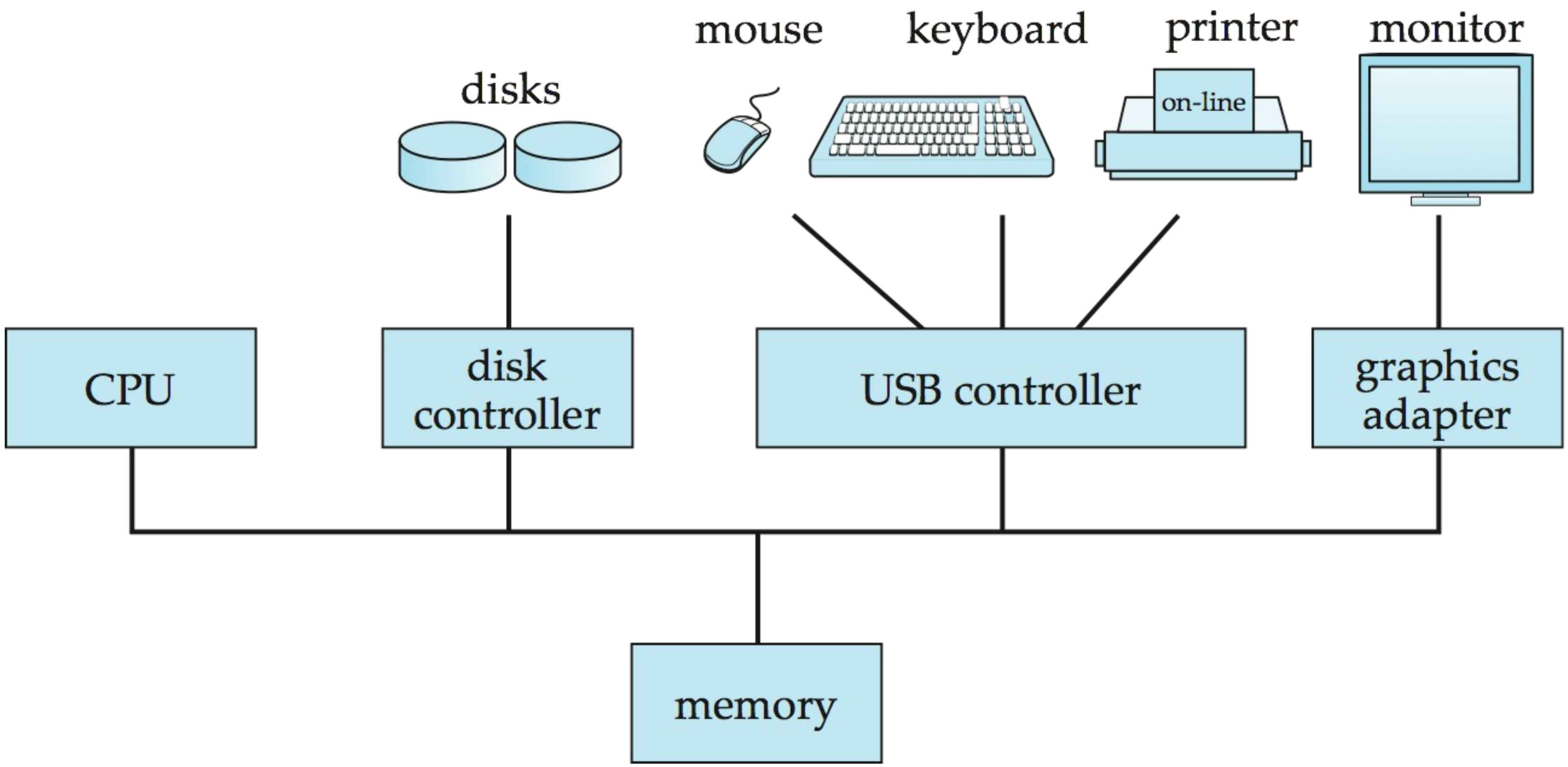
- Variants for creating a database system
  - Centralized and Client-Server Systems
  - Server System Architectures
  - Parallel Systems
  - Distributed Systems

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
  - shared memory
- Single-user system
  - e.g., personal computer or workstation
  - desk-top unit, single user, usually one CPU and one or two hard disks
- Multi-user system
  - more disks, more memory, multiple CPUs
  - serve a large number of users, usually connected to the system via terminals
  - also called *server systems*

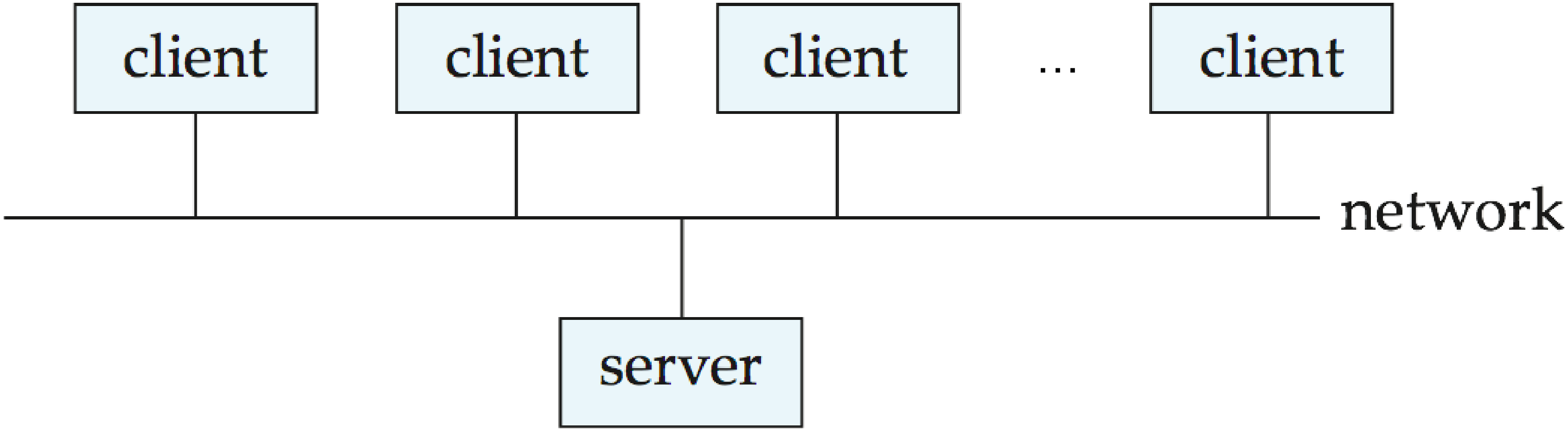
Centralized Systems

- Simplified Architecture



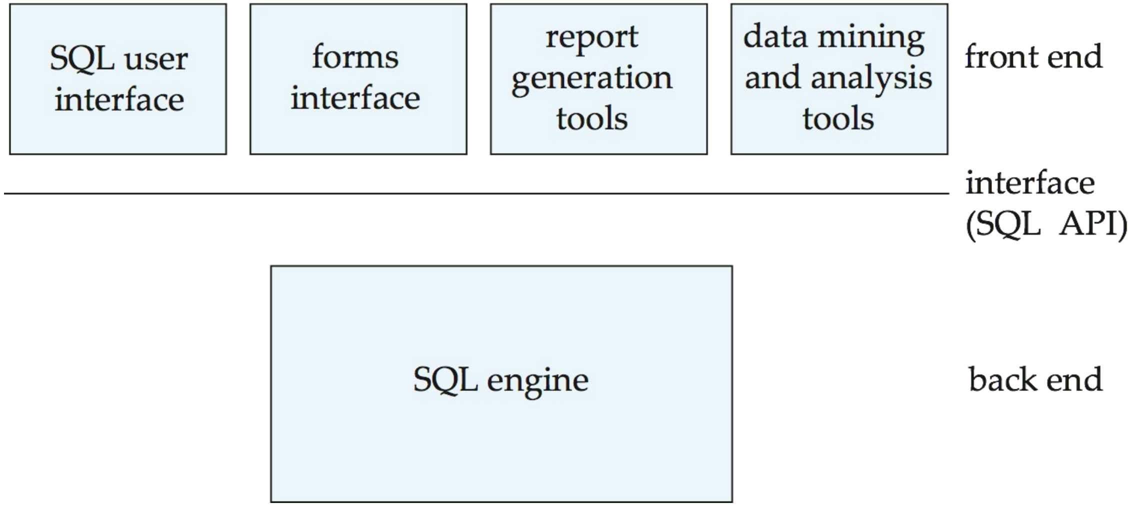
Client Server Systems

- Server systems satisfy requests generated at *m* client systems
- They are connected to the server via a network
  - local or internet
  - LAN or WIFI
  - ...



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
- Interface between the front-end and the back-end:
  - SQL or proprietary application program interface (API)



Client-Server Systems

- Advantages of client-server systems over single machine systems:
  - better functionality for the cost
  - flexibility in locating resources and expanding facilities
  - better user interfaces
  - easier maintenance

- Server systems can be broadly categorized into two kinds:
  - **transaction servers** (used for RDBMS, aka SQL servers)
  - **data servers** (used for object-oriented databases)

SQL Servers

- Also called **query server** systems or transaction servers
  - 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
  - similar implementations exist for Python etc.

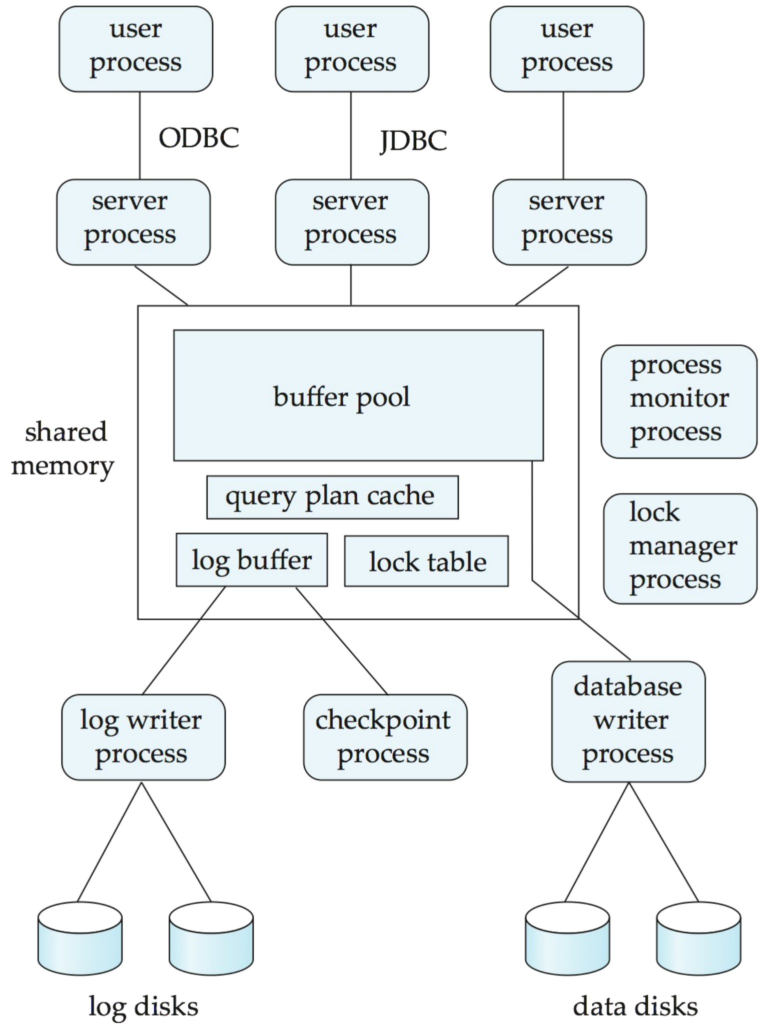
SQL Servers

- A typical SQL 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
  - Typically multiple multithreaded server processes
- Lock manager process
  - More on this later
- Database writer process
  - Output modified buffer blocks to disks continually

SQL Server Processes

- 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

SQL Server Processes: Overview



SQL Server Processes: Overview

- 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 avoid concurrency, DBMS 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

Parallel Database 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

Speedup and Scaleup

- Question: how much performance do we gain by enlarging the system?
  - Optimum: linear scalability: doubling the system doubles the performance
- **Speedup**: a fixed-sized problem executing on a small system is given to a system which is *N*-times larger
- Measured by:

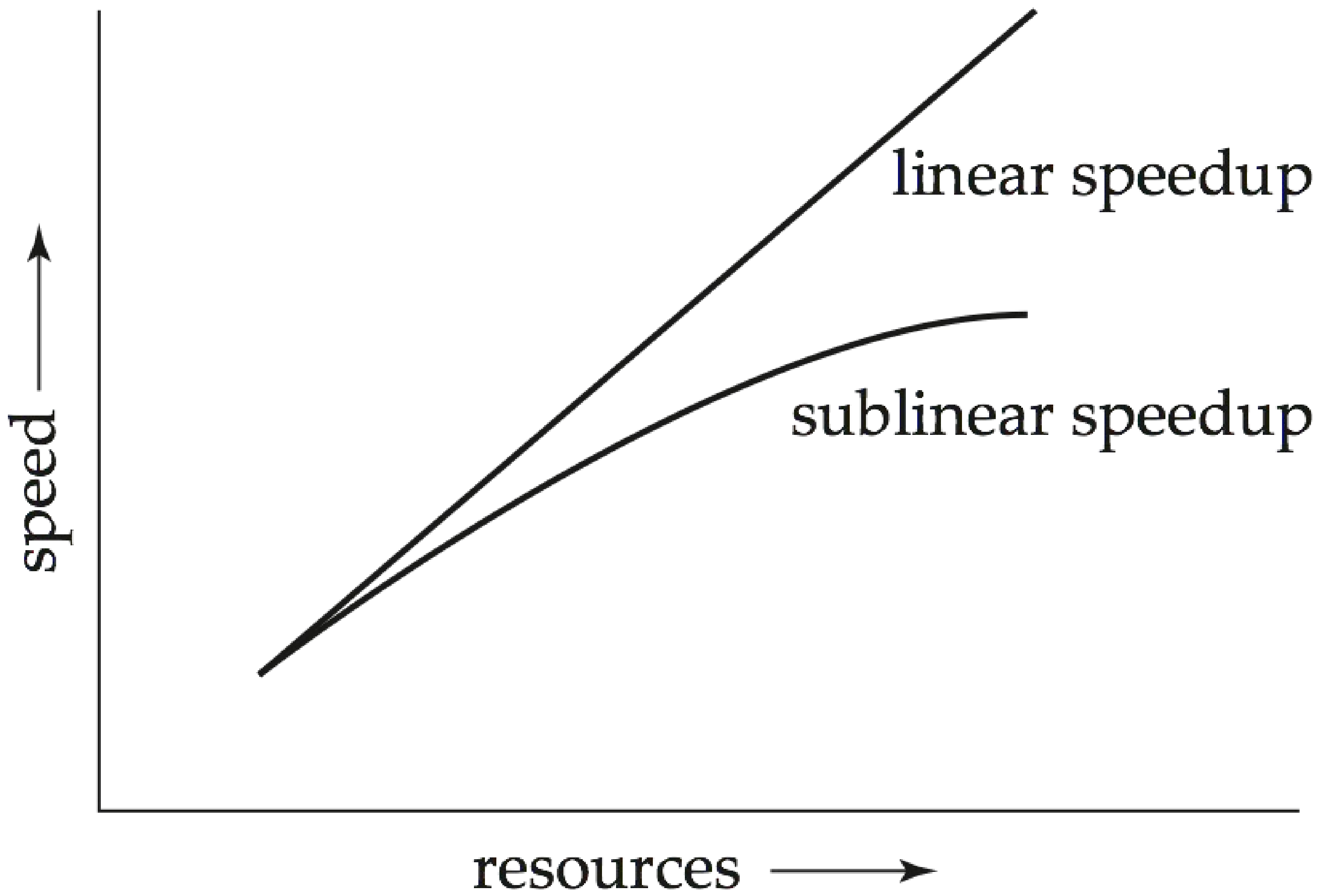
$$\textit{speedup} = \frac{\textit{small system elapsed time}}{\textit{large system elapsed time}}$$

- 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:

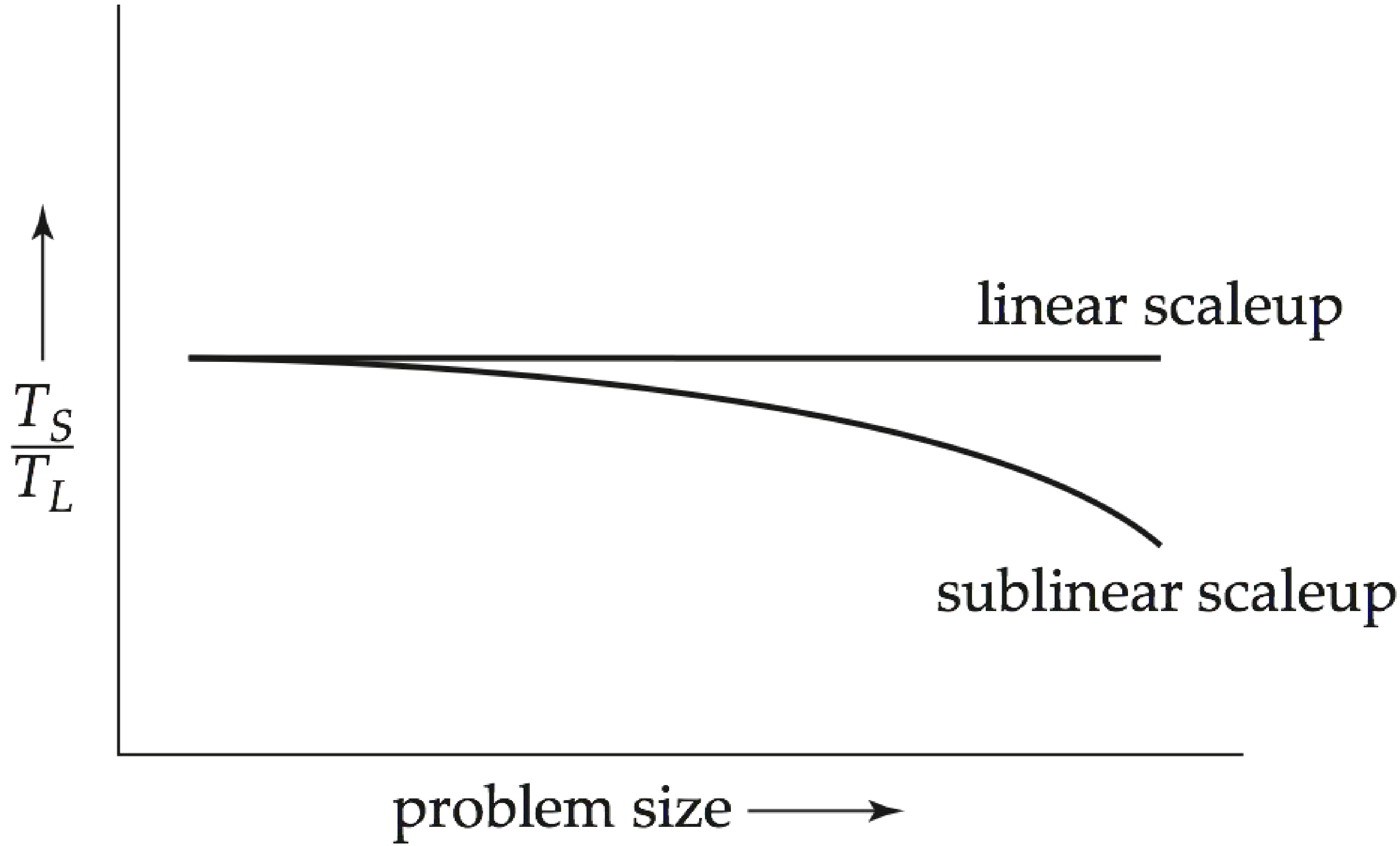
$$\textit{scaleup} = \frac{\textit{small system small problem elapsed time}}{\textit{big system big problem elapsed time}}$$

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

Speedup



Scaleup



Batch and Transaction Scaleup

- **Batch scaleup**:
  - A single large job
  - Use an *N*-times larger computer on *N*-times larger problem
- **Transaction scaleup**:
  - Numerous small queries submitted by independent users to a shared database
  - *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 for parallel execution

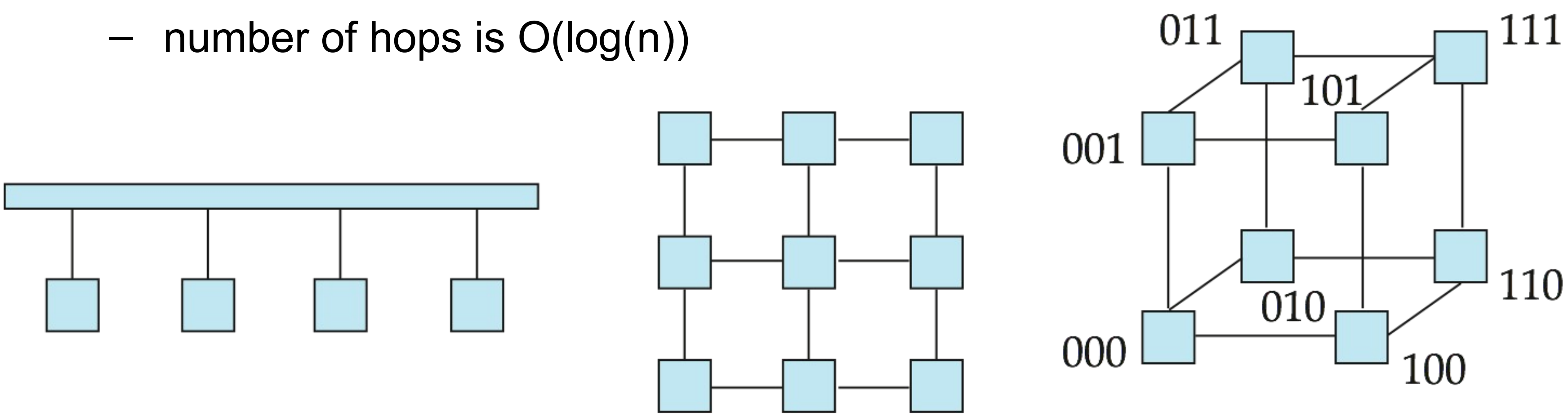
Limitation of Speedup and Scaleup

Speedup and scaleup are often sublinear due to:

- **Startup costs**
  - cost of starting up multiple processes may dominate computation time
  - esp. if the degree of parallelism is high
- **Interference**
  - processes accessing shared resources (e.g., system bus, disks, or locks) compete with each other → bottlenecks
  - 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 parallely executing tasks
  - Overall execution time determined by **slowest** of parallely executing task

Interconnection Networks

- Bus: does not scale well with increasing parallelism
- Mesh:
  - scalability grows with number of links
  - but number of hops grows at O(√(n))
- Hypercube:
  - good tradeoff
  - number of hops is O(log(n))

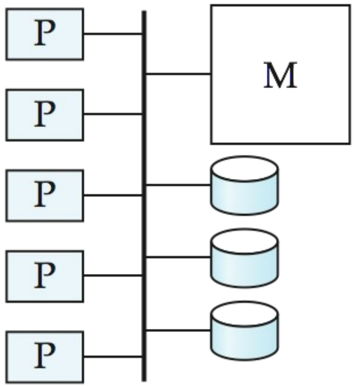


(a) bus (b) mesh (c) hypercube

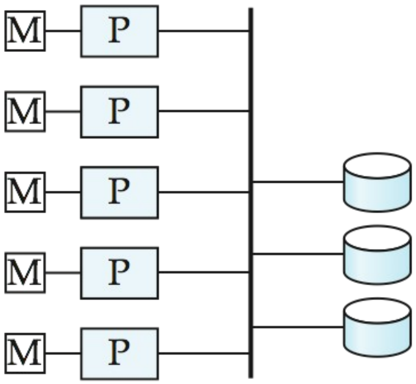
Parallel Database Architectures

- **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

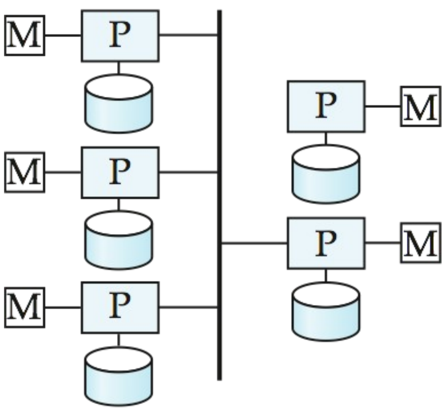
Parallel Database Architectures



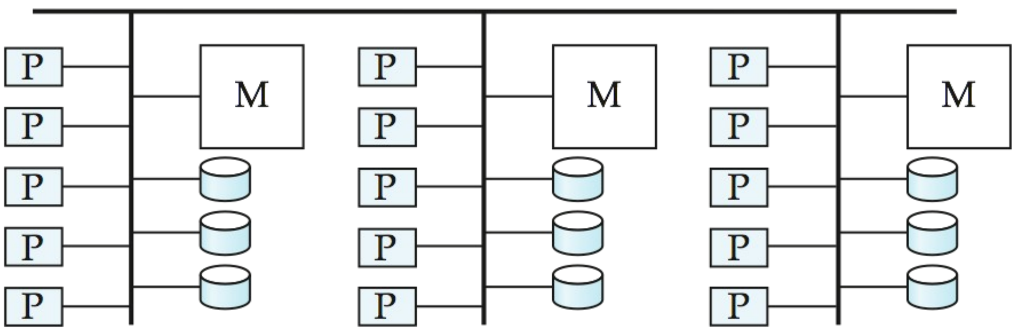
(a) shared memory



(b) shared disk



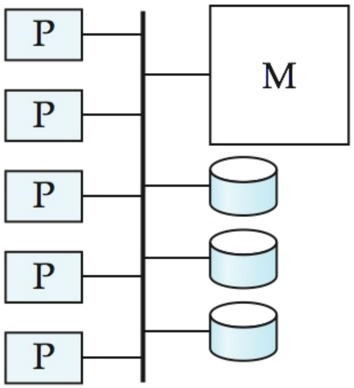
(c) shared nothing



(d) hierarchical

Shared Memory

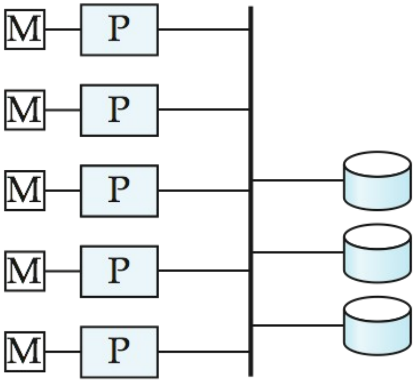
- Processors and disks have access to a common memory
  – typically via a bus or through an interconnection network
- Extremely efficient communication between processors
  – data in shared memory can be accessed by any processor
  – without having to move it using software
- Architecture is not scalable beyond 32 or 64 processors
  – interconnection network becomes a bottleneck
- Widely used for lower degrees of parallelism (4 to 8)



(a) shared memory

Shared Disk

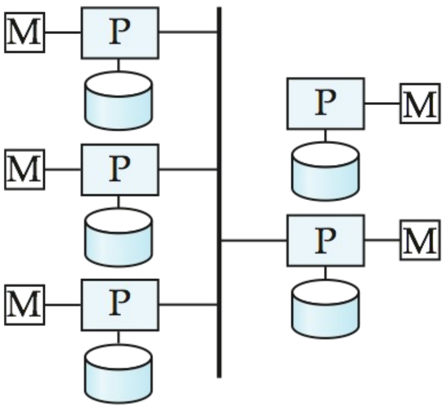
- All processors can directly access all disks via an interconnection network, but the processors have private memories
  - i.e., the memory bus is not a bottleneck
- Downside
  - bottleneck now occurs at interconnection to the disk subsystem
- Shared-disk systems can scale to a somewhat larger number of processors, but communication between processors is slower



(b) shared disk

Shared Nothing

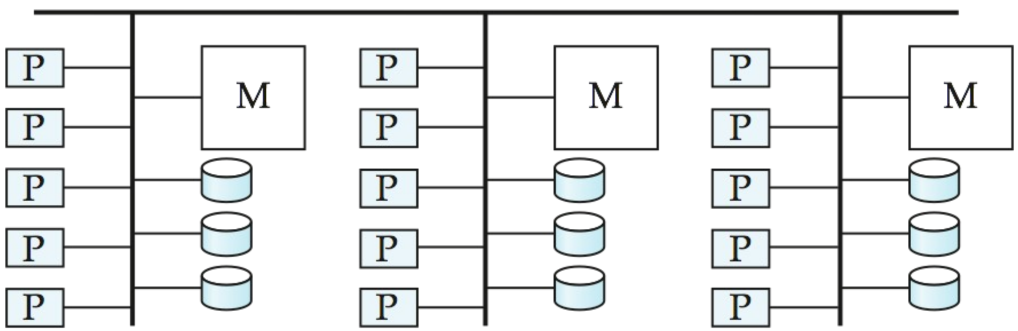
- Each node consists of a processor, memory, and one or more disks
- Node functions as the server for the data on the disk(s) it owns
- Data accessed from local disks (and local memory accesses) do not pass through interconnection network, thereby minimizing the interference of resource sharing
- Shared-nothing 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



(c) shared nothing

Hierarchical

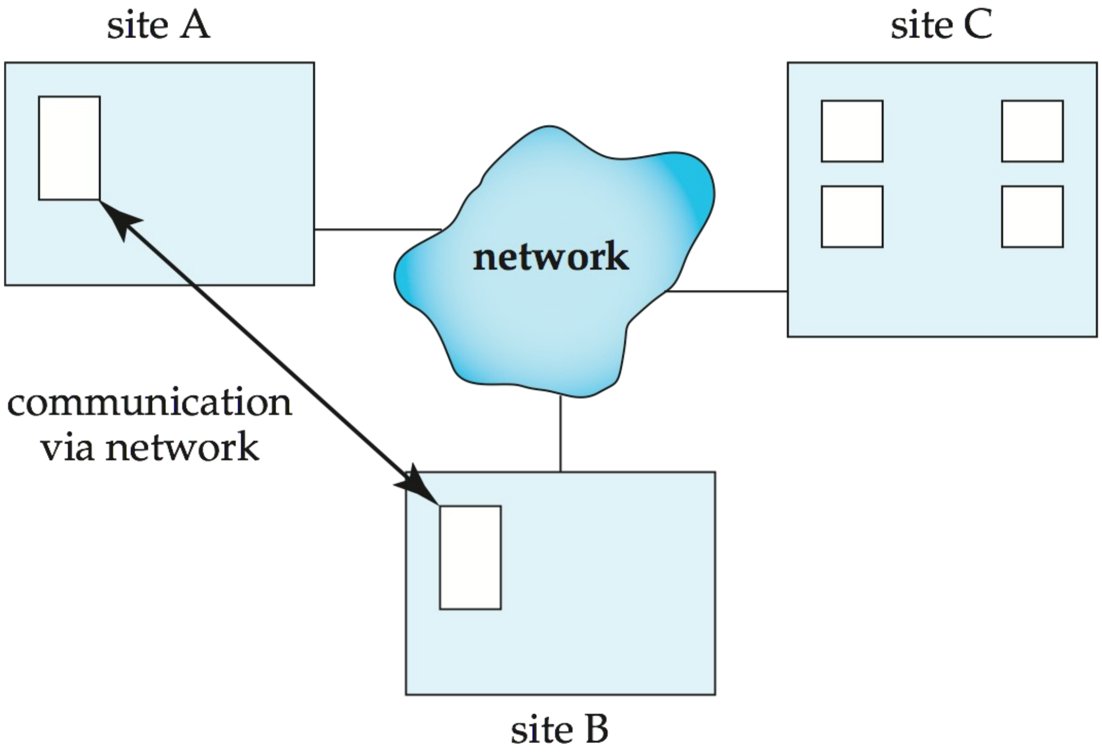
- Combines characteristics of all three architectures
- Top level is a shared-nothing architecture
  - Each node of the system could be a shared-memory system or a shared-disk system
- Reduce the complexity of programming such systems by **distributed virtual-memory** architectures
- Also called **non-uniform memory architecture (NUMA)**



(d) hierarchical

Distributed Database Systems

- Data spread over multiple machines (also: *sites*)
- Network interconnects the machines
- Data shared by users on multiple machines



Distributed Database Systems

- 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

Trade Offs in Distributed Database 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

- 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 the 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

Summary

- Data storage is layered
  - trading off cost/byte vs. access speed
- Data organization in files
  - trading off disk usage vs. reorganization cost
  - minimize block transfer
- Database architectures
  - single machine vs. distributed
  - scalability of distributed databases (speedup/scaleup)
  - design issues of distributed databases

Questions?