國立陽明交通大學
NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Biological Databases: Theories and Practice 430032

Ch 1. Database System and Architectures

Instructor: Prof. LEE, Tzong-Yi (李宗夷)
Email: leetzongyi@nycu.edu.ctw

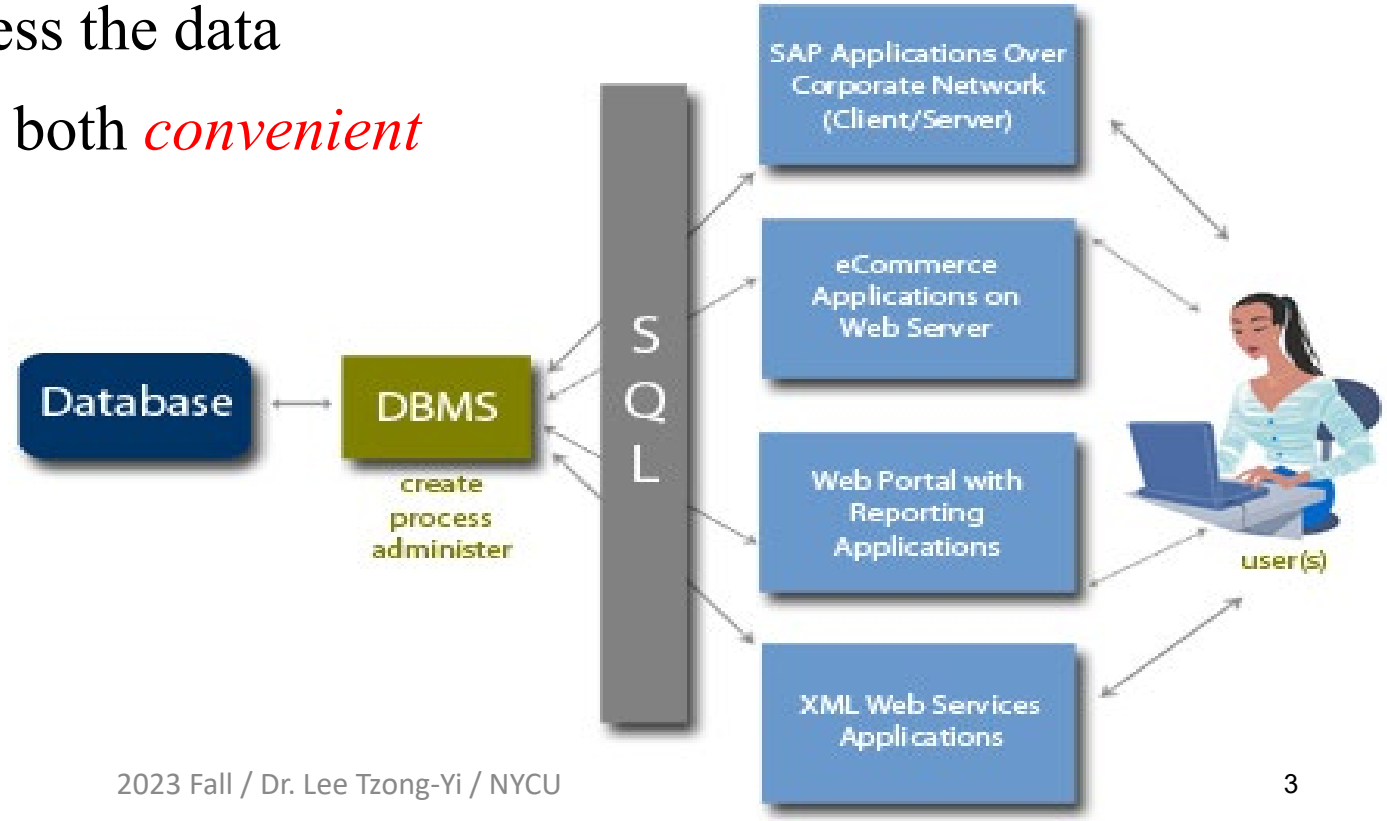
Professor
Institute of Bioinformatics and Systems Biology,
National Yang Ming Chiao Tung University

Lecture outline

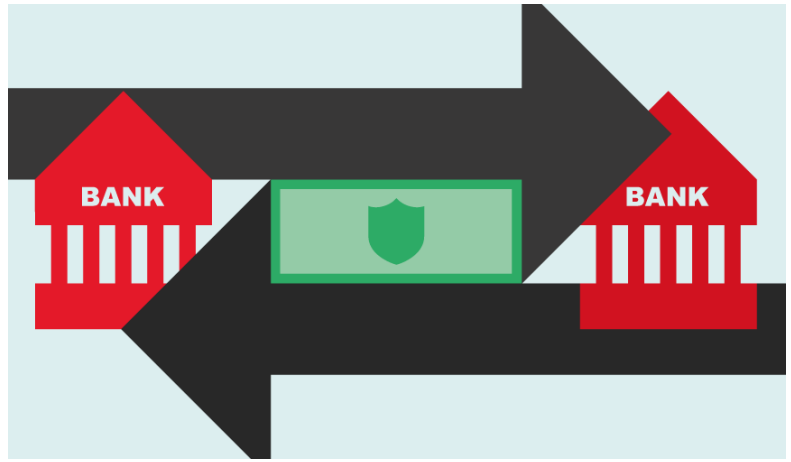
1. Database Management System (DBMS)
2. Purpose of Database Systems
3. View of Data
4. Database Languages
5. Relational Databases
6. Database Design
7. Data Storage and Querying
8. Transaction Management
9. Database Architectures

Database Management System (DBMS)

- **DBMS** contains information about a particular enterprise
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both *convenient* and *efficient* to use



Database Applications



- Banking: transactions
- Airlines: reservations, schedules
- Universities: students, instructors, grades
- Sales: customers, products, purchases
- Online retailers: order tracking, customized recommendations
- Manufacturing: production, inventory, orders, supply chain
- Health care: clinic, prognosis, drugs, diseases, patients



Purpose of Database Systems

- Drawbacks of using file systems to store data:
  - **Data redundancy** and **inconsistency**
    - Multiple file formats, duplication of information in different files
  - **Difficulty in accessing data**
    - Need to write a new program to carry out each new task
  - **Data isolation** — multiple files and formats
  - **Integrity problems**
    - **Consistency constraints** (e.g., age > 0) become "buried" in program code rather than being stated explicitly
    - Hard to add new constraints or change existing ones

Purpose of Database Systems (Cont.)

- Drawbacks of using file systems (cont.)
  - **Atomicity of updates**
    - Failures may leave database in an inconsistent state with partial updates carried out
    - Example: Transfer of funds from one account to another should either complete or not happen at all
  - **Concurrent access by multiple users**
    - Concurrent access needed for performance
    - Uncontrolled concurrent accesses can lead to inconsistencies
      - Example: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
  - **Security problems**
    - Not every user of the database system should be able to access all the data.
      - Example: In an university, payroll personnel need to see only that part of the database that has financial information.
- Database systems offer solutions to all the above problems

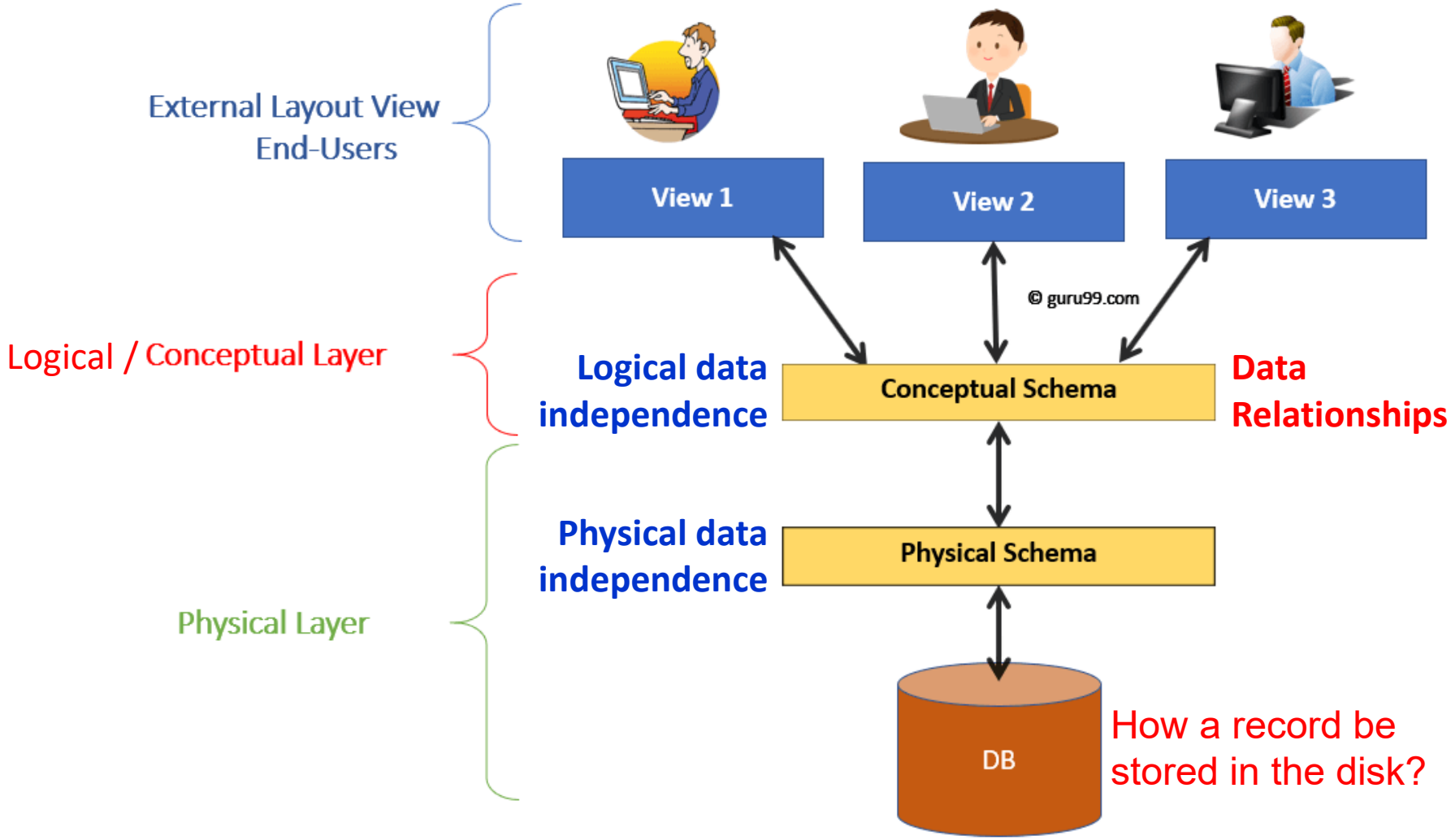
Data Abstraction (three levels)

- **Physical level:** the lowest level describing **how** a record (e.g., customer) is stored.
- **Logical/conceptual level:** describes **what** data are stored in database, and **what relationships** exist among those data.

```
type instructor = record
        ID : char (5);
        name : char (20);
        dept_name : char (20);
        salary : number (8,2);
        end;
```

- **View level:** the highest level describing only **part of the entire database**. Application programs hide details of data types. **To simplify** the interaction with the system and may provide **many views** for the same database. Views can also hide information (such as an employee's salary) for **security purposes**.

Levels of DBMS Architecture

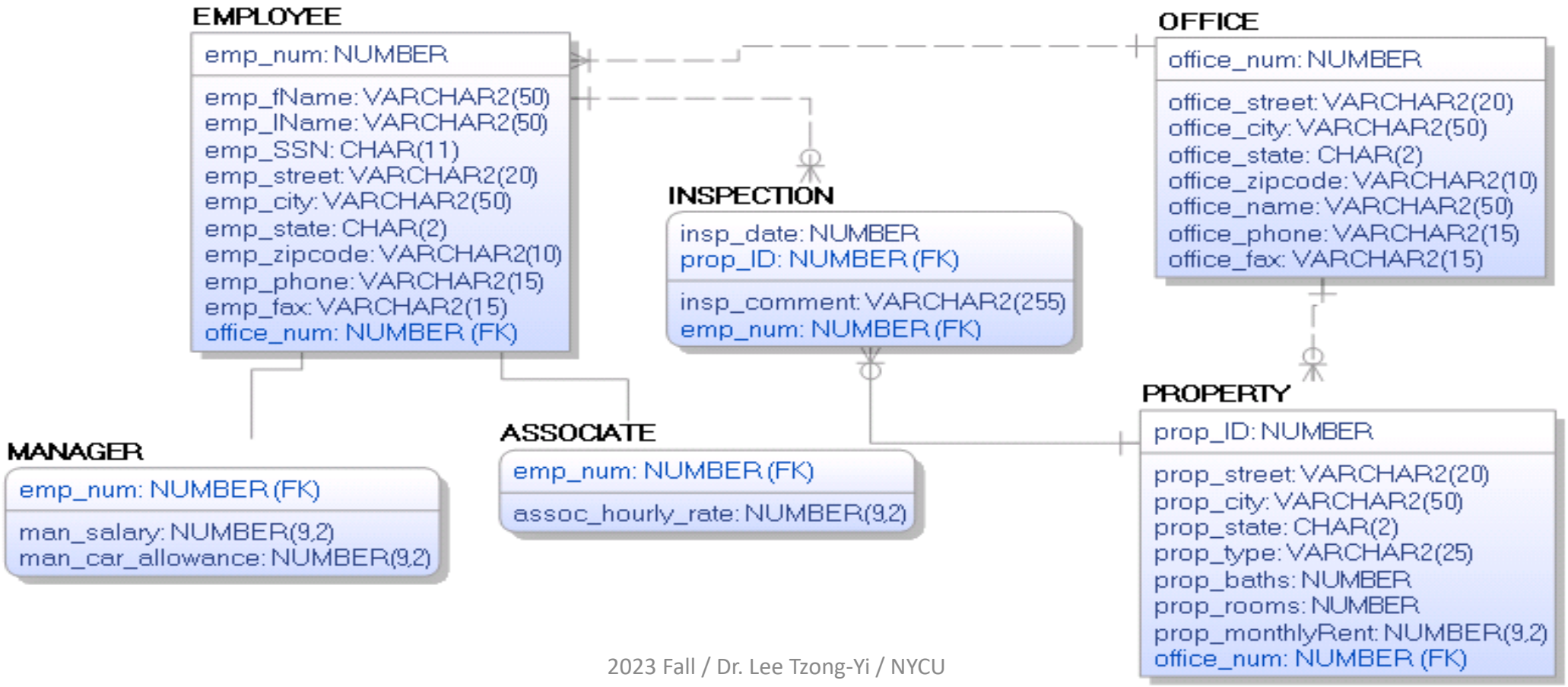


Instances and Schemas

- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable
- **Schema** – the overall design of the database
  - Example: The database consists of information about a set of customers and accounts and the relationship between them
  - Analogous to type information of a variable in a program
  - **Physical schema**: database design at the physical level
  - **Logical schema**: database design at the logical level
- **Logical Data Independence** - the ability to change the conceptual scheme without changing external views or API programs
  - When compared to Physical Data independence, it is challenging to achieve logical data independence.
- **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

Instances and Schemas

- An example of logical schema for a company:



Examples of changes under Logical Data Independence

- Due to Logical independence, any of the below change will not affect the external layer.
  - Add/Modify/Delete a new attribute, entity or relationship is possible without a rewrite of existing application programs
  - Merging two records into one
  - Breaking an existing record into two or more records

Examples of changes under Physical Data Independence

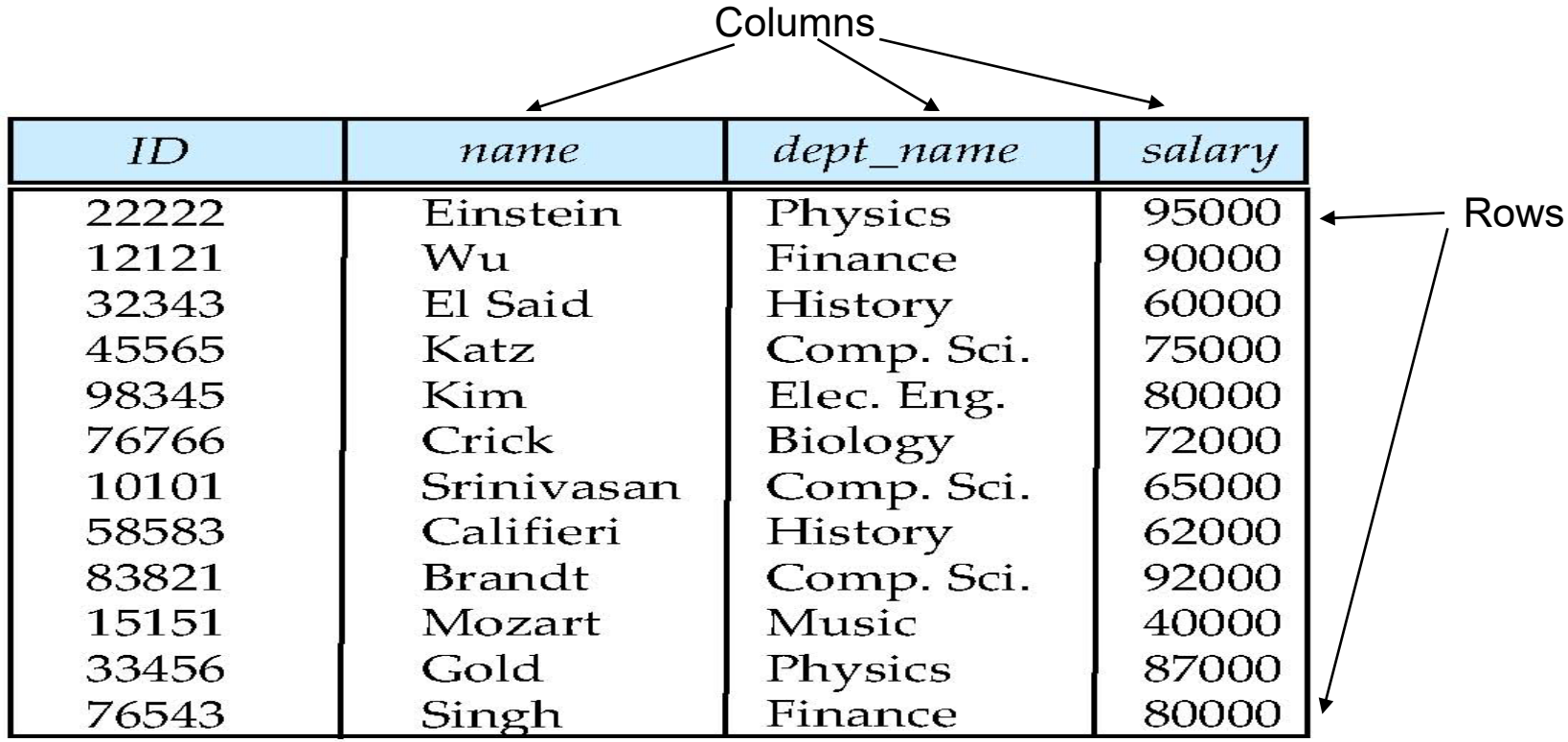
- Due to Physical independence, any of the below change will not affect the logical layer.
  - Using a new storage device like Hard Drive or Magnetic Tapes
  - Modifying the file organization technique in the Database
  - Switching to different data structures.
  - Changing the access method.
  - Modifying indexes.
  - Changes to compression techniques or hashing algorithms.
  - Change of Location of Database from C drive to D Drive

Data Models

- Data mode is a collection of conceptual tools for describing Data, Data relationships, Data semantics, and Consistency constraints.
  - **Relational model**: a collection of tables to represent both data and relationships among these data. (Ch.2)
  - **Entity-Relationship (E-R) model**: a collection of basic objects, called ***entities*** and ***relationships*** among these objects. (Ch.7)
  - Object-based data models: combines the Object-oriented programming concepts (C++ and Java) and relational data model.
  - Semistructured data model: the Extensible Markup Language (XML) is widely used to represent semistructured data.
  - Other older models:
    - Network model, Hierarchical model, etc.

Relational Model

- Relational model (will be introduced in Chapter 2)
- Example of tabular data in the relational model



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

(a) The *instructor* table

A Sample Relational Database

- An example of relationship

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

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

Data Manipulation Language (DML)

- Language for accessing and manipulating the data organized by the appropriate data model
  - DML also known as query language
- Two classes of languages
  - **Procedural** – user specifies what data is required and how to get those data
  - **Declarative (nonprocedural)** – user specifies what data is required without specifying how to get those data
- **Structured Query Language (SQL)** is the most widely used declarative language

Data Definition Language (DDL)

- Specification notation for defining the database schema

  Example: **create table** *instructor* (
  ID **char**(5),
  name **varchar**(20),
  dept_name **varchar**(20),
  salary **numeric**(8,2))

- DDL compiler generates a set of tables stored in a *data dictionary*
- Data dictionary contains **metadata** (i.e., data about data)
  - Database schema
  - Integrity constraints
    - Primary key (ID uniquely identifies instructors)
    - Referential integrity (**references** constraint in SQL)
      - e.g. *dept_name* value in any ***instructor*** tuple must appear in ***department*** relation
  - Authorization

Structured Query Language (SQL)

- **SQL**: widely used non-procedural language (Chapter 3)
  - Example: Find the name of the instructor with ID 22222

    select *name*
    from *instructor*
    where *instructor.ID* = '22222'
  - **select** *instructor.ID*, *department.dept_name*
    from *instructor*, *department*
    where *instructor.dept_name*= *department.dept_name* **and** *department.budget* > 95000

- Application programs generally access databases through one of
  - Language extensions to allow embedded SQL
  - Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

Database Design

The process of designing the general structure of the database:

- **Logical Design** – Deciding on the database schema. Database design requires that we find a "good" collection of relation schemas.
  - Business decision – What attributes should we record in the database?
  - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?

- **Physical Design** – Deciding on the physical layout of the database

What is a Good Database Design?

- Is there any problem with this design?

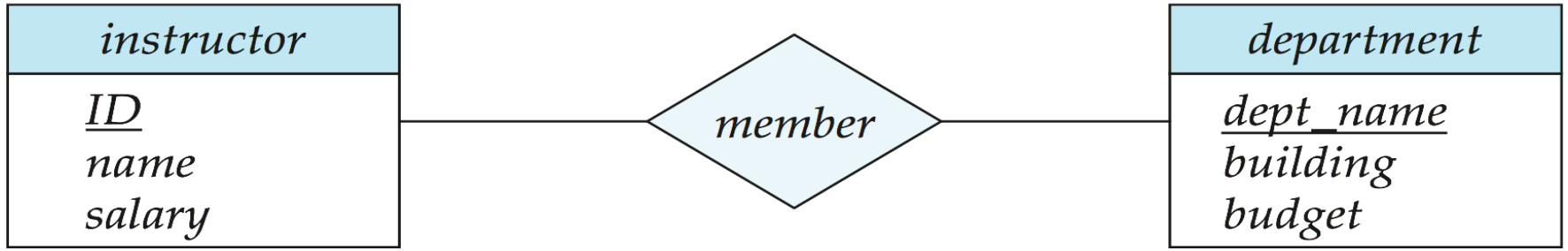
ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Design Approaches

- Normalization Theory
  - Formalize what designs are bad, and test for them
- Entity-Relationship Model
  - Models an enterprise as a collection of *entities* and *relationships*
    - Entity: a “thing” or “object” in the enterprise that is distinguishable from other objects
      - Described by a set of *attributes (e.g. name, age, department, salary, etc.)*
    - Relationship: an association among several entities
  - Represented diagrammatically by an *entity-relationship diagram:*

The Entity-Relationship Model

- Models an enterprise as a collection of *entities* and *relationships*
  - Entity: a “thing” or “object” in the enterprise that is distinguishable from other objects
    - Described by a set of *attributes*
  - Relationship: an association among several entities
- Represented diagrammatically by an *entity-relationship diagram:*



What happened to department and instructor?

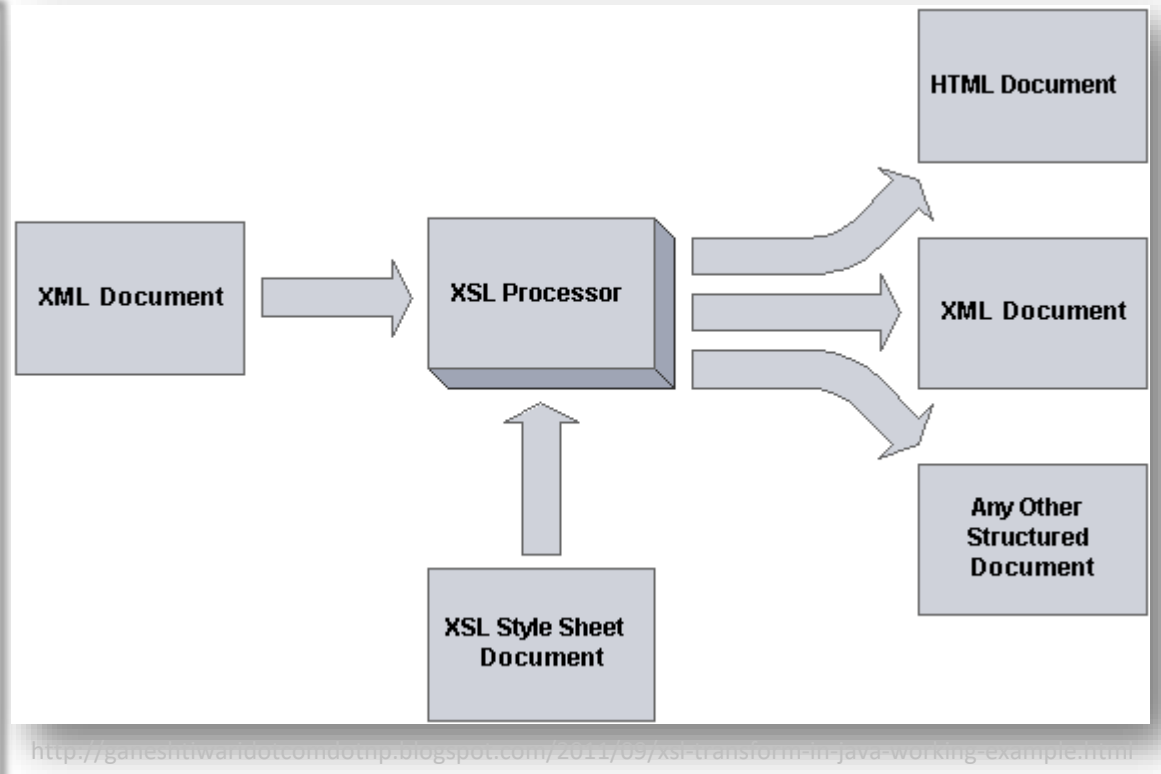
XML: Extensible Markup Language

- Defined by the WWW Consortium (W3C)
- Originally intended as a document markup language not a database language
- The ability to specify new tags, and to create nested tag structures made XML a great way to **exchange data**, not just documents
- XML has become the basis for all new generation data interchange formats.
- A wide variety of tools is available for parsing, browsing and querying XML documents/data

An example of XML

```
<Books>
    <Book ISBN="0553212419">
        <title>Sherlock Holmes: Complete Novels...
        <author>Sir Arthur Conan Doyle</author>
    </Book>
    <Book ISBN="0743273567">
        <title>The Great Gatsby</title>
        <author>F. Scott Fitzgerald</author>
    </Book>
    <Book ISBN="0684826976">
        <title>Undaunted Courage</title>
        <author>Stephen E. Ambrose</author>
    </Book>
    <Book ISBN="0743203178">
        <title>Nothing Like It In the World</title>
        <author>Stephen E. Ambrose</author>
    </Book>
</Books>
```

XML transformation

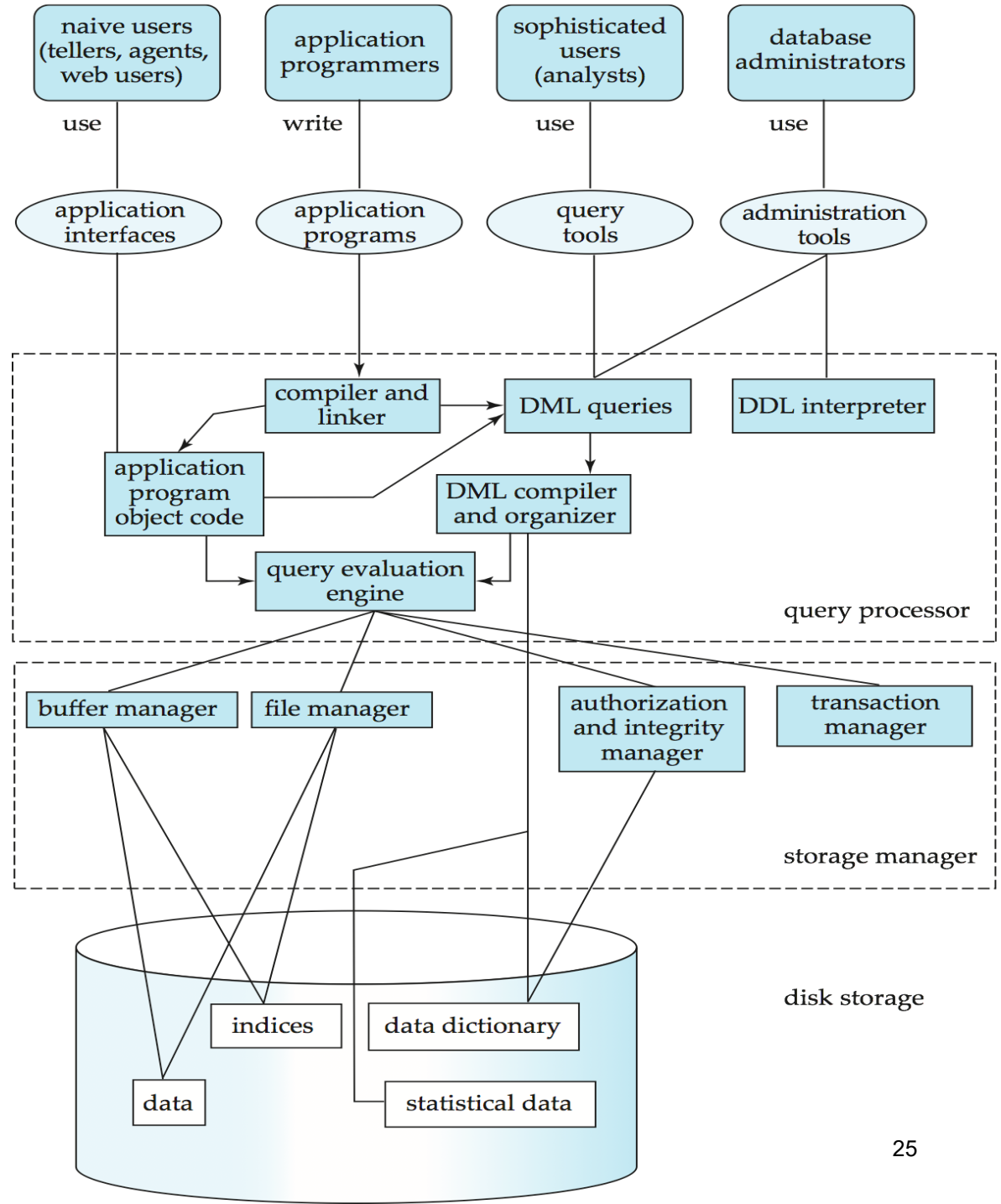


http://ganeshtiwaridotcomdotnp.blogspot.com/2011/09/xsl-transform-in-java-working-example.html

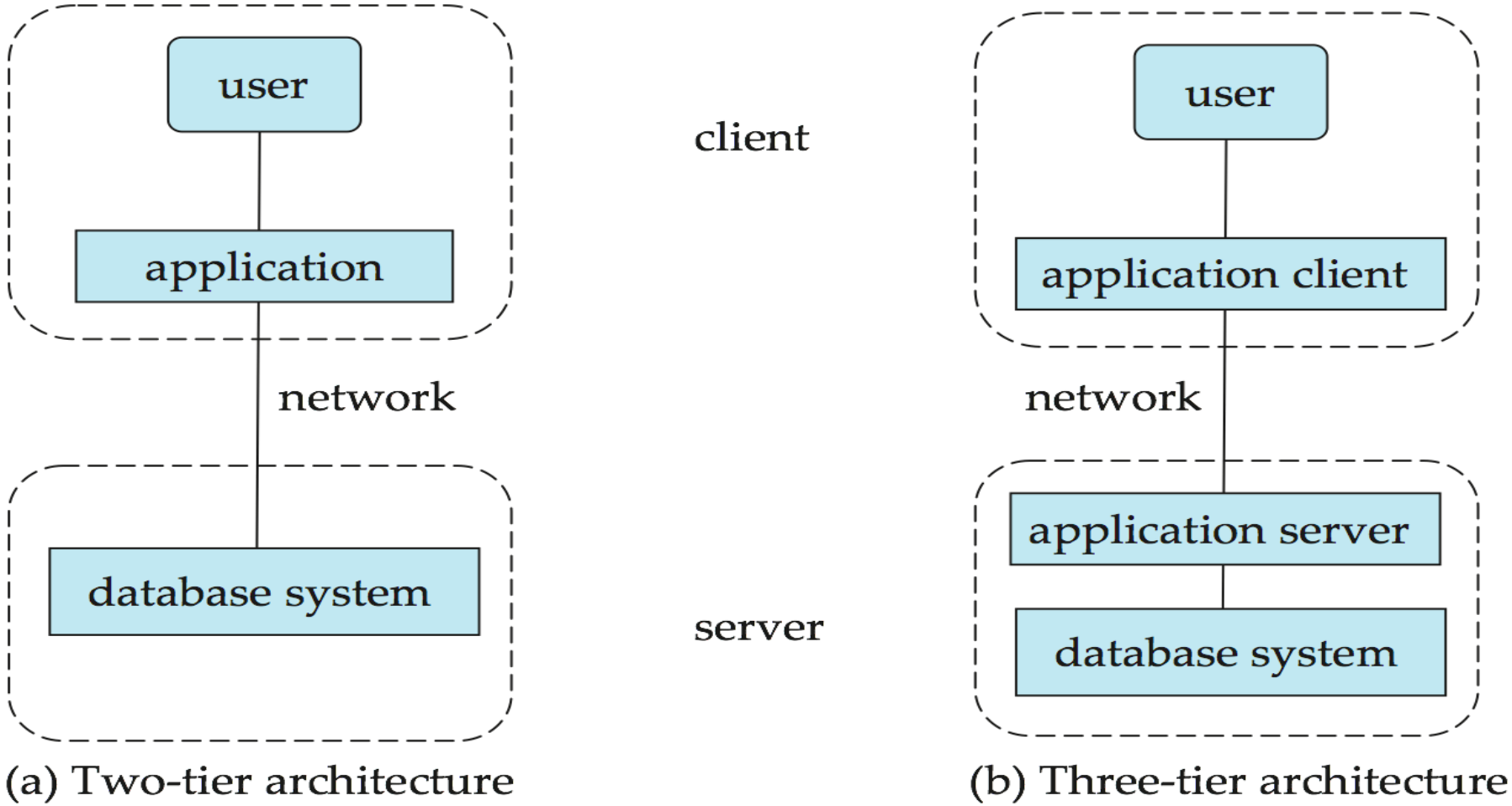
XSL (e**X**tensible **S**tylesheet **L**anguage)

System Structure

- The various components of a database system and the connections among them.
- Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines.
  - We can therefore differentiate between **client** machines, on which database users work, and **server** machines, on which the database system runs.



Two-tier and three-tier architectures



(a) Two-tier architecture

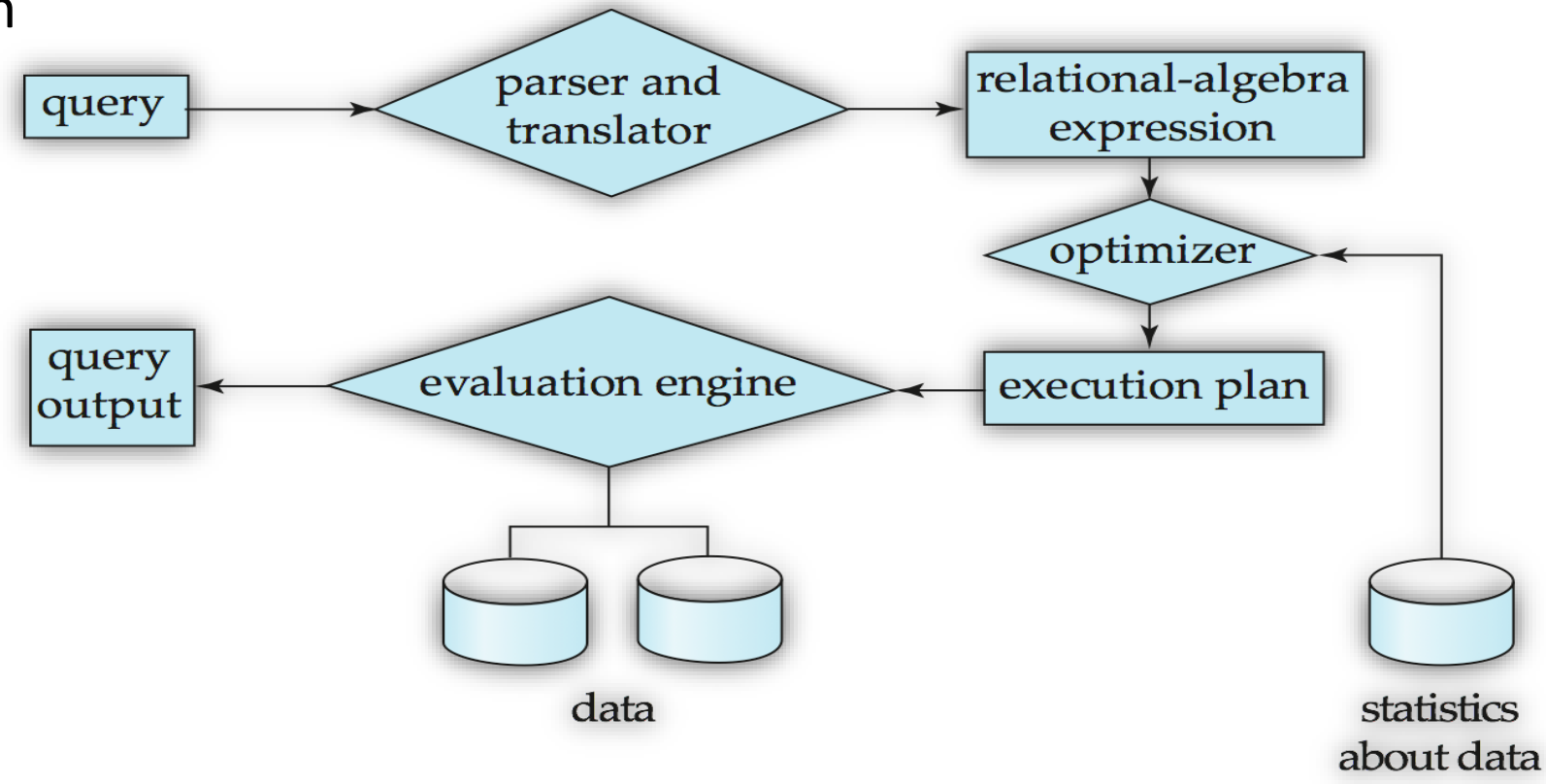
(b) Three-tier architecture

Storage Management

- **Storage manager** is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible to the following tasks:
  - Interaction with the file manager
  - Efficient storing, retrieving and updating of data
- Issues:
  - Storage access
  - File organization
  - Indexing and hashing

Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Transaction Management

- What if the system fails?
- What if more than one user is concurrently updating the same data?
- A **transaction** is a collection of operations that performs a single logical function in a database application
- **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database.

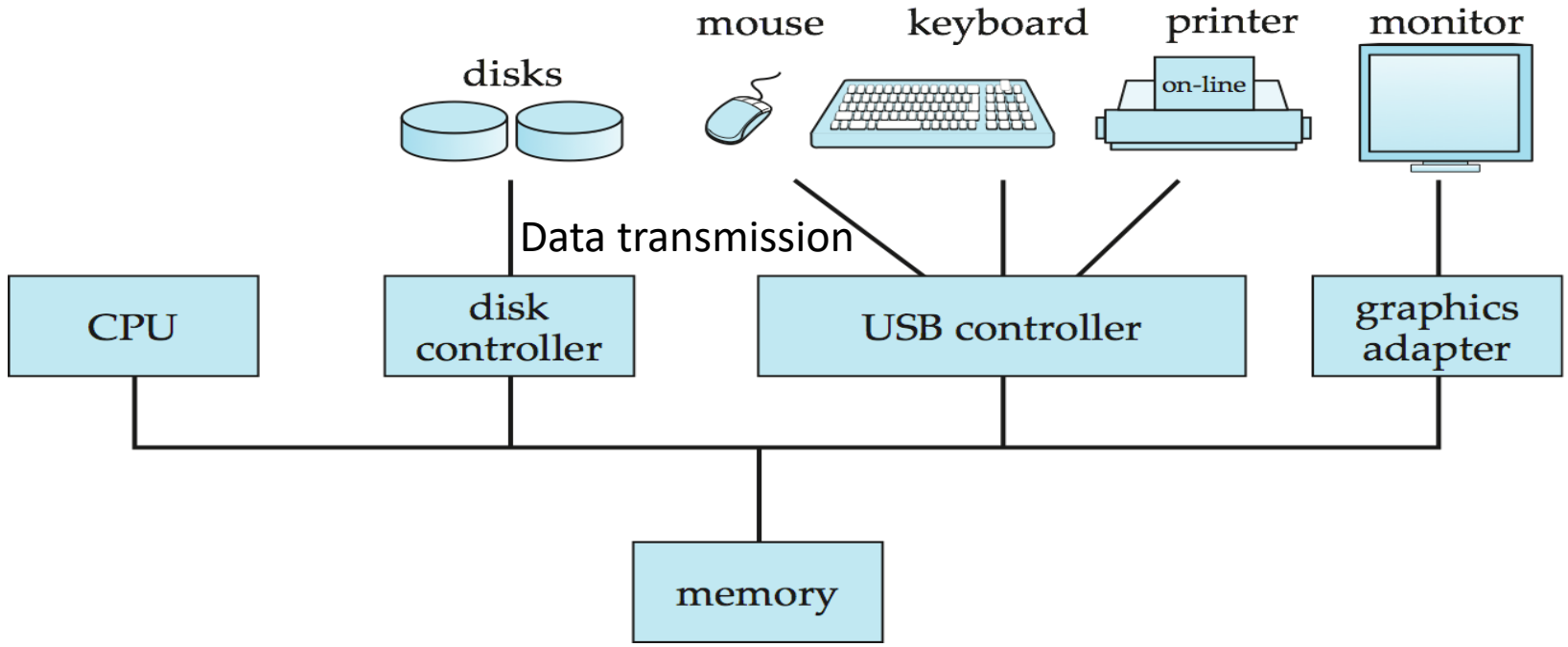
Database Architecture

The architecture of a database systems is greatly influenced by the underlying computer system on which the database is running:

- Centralized
- Client-server
- Parallel (multi-processor)
- Distributed

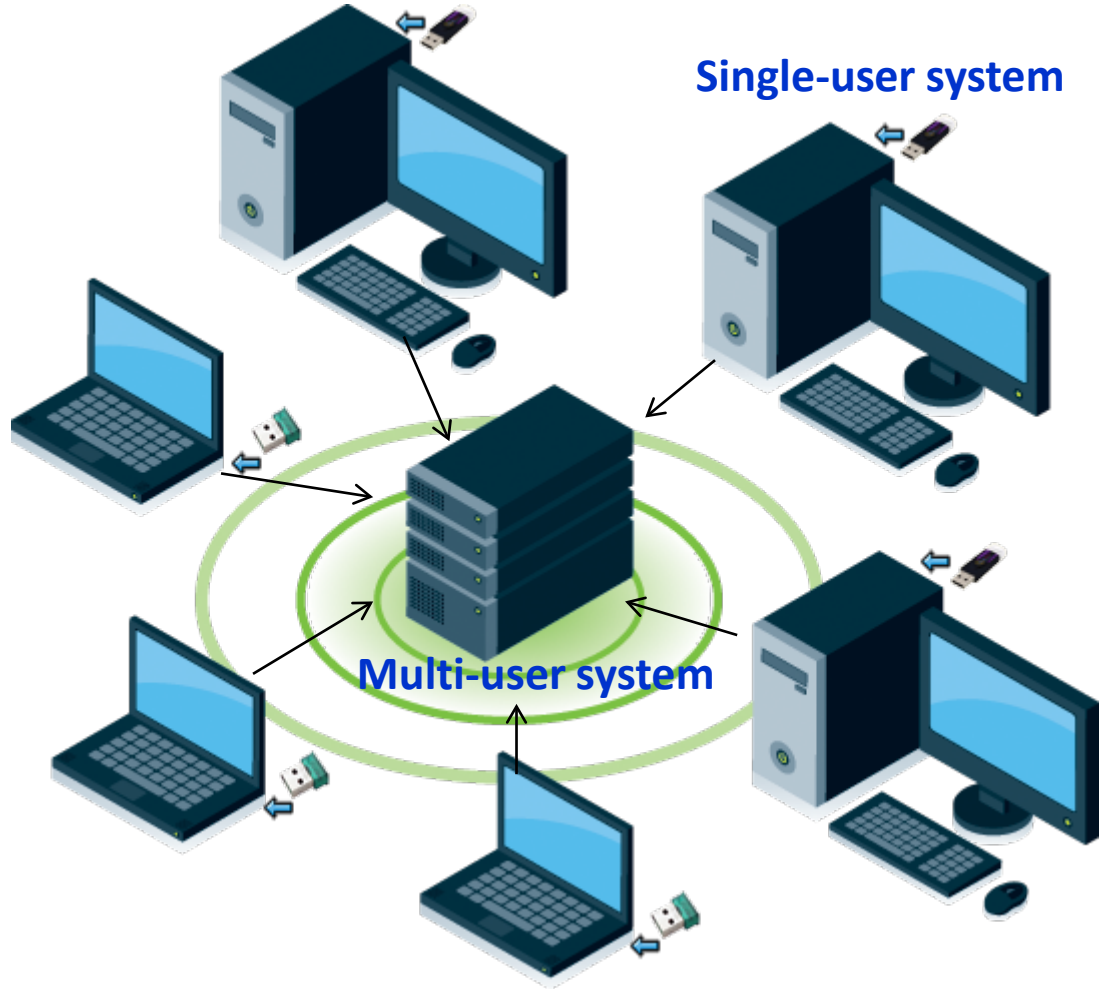
Centralized Systems

- 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.



Centralized Systems

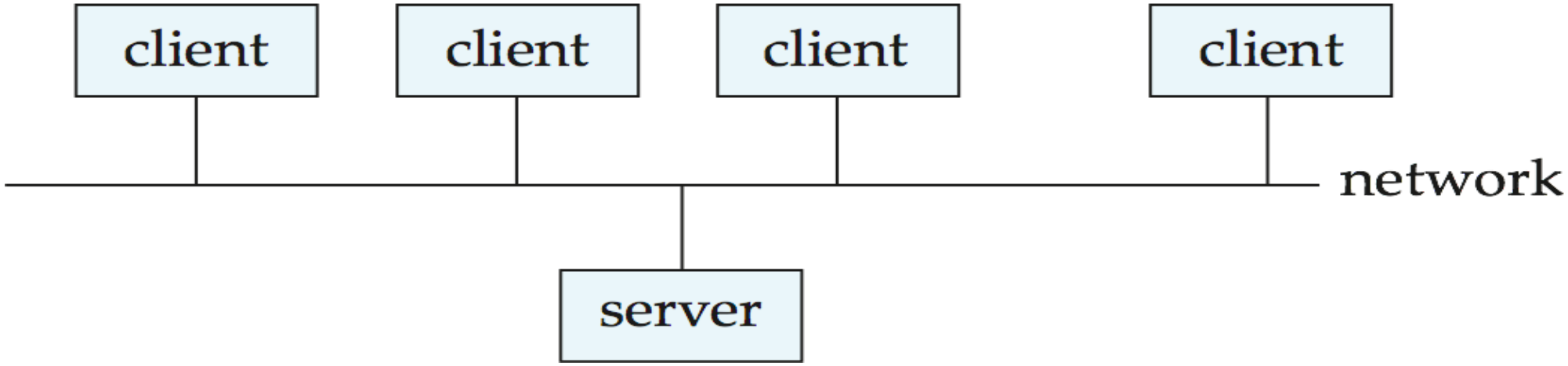
- **Single-user system** (e.g., personal computer or workstation): desk-top unit, single user, usually has only one CPU and one or two hard disks; the OS may support only one user. Often called *user systems*.
- **Multi-user system**: more disks, more memory, multiple CPUs, and a multi-user OS. It serves a large number of users who are connecting to the system remotely. Often called *server systems*.



Client-Server Systems

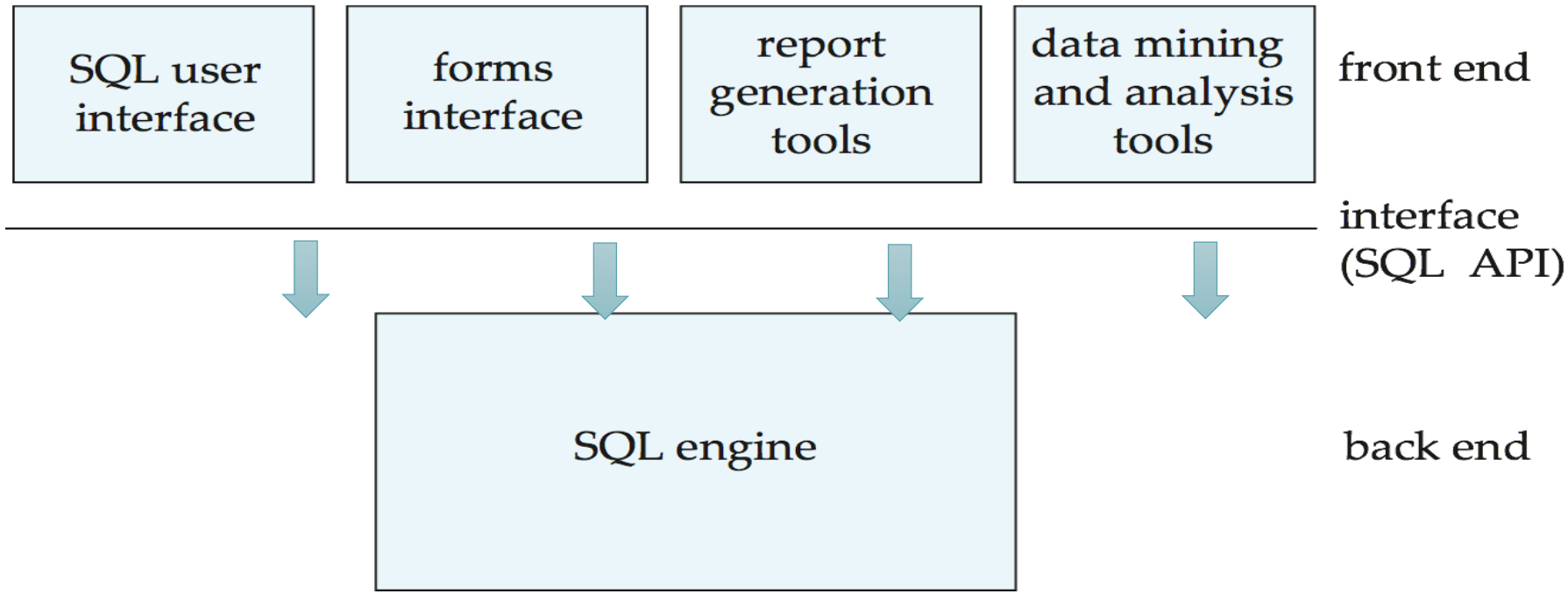
- As personal computers became faster, more powerful, and cheaper, the centralized systems today act as **server systems** that satisfy requests generated from multiple **client systems**.

- The general structure of client-server systems is shown below:



Client-Server Systems (Cont.)

- Certain application programs in **front-end** client systems, such as mobile banking apps or statistical analysis tools, use the client-server interface directly to access data form a **back-end** server system.



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 is through SQL or through an application program interface (API), based on database standards such as *ODBC* or *JDBC*.

Client-Server Systems (Cont.)

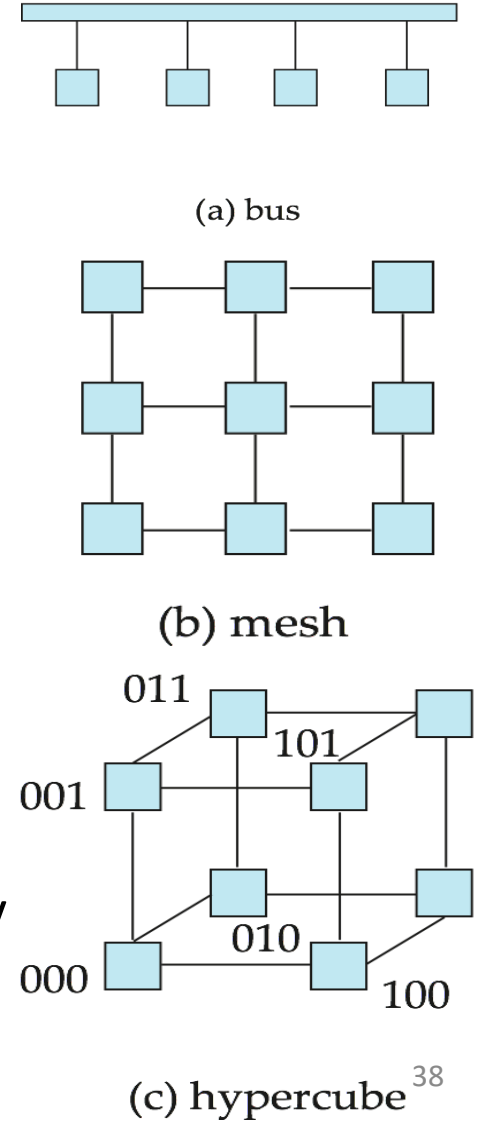
- **Advantages** of replacing mainframes with networks of workstations or personal computers connected to back-end server machines:
  - better functionality for the cost
  - flexibility in locating resources and expanding facilities
  - better user interfaces
  - easier maintenance

Parallel Systems

- Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network. Thus, many operations can be performed simultaneously.
  - 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

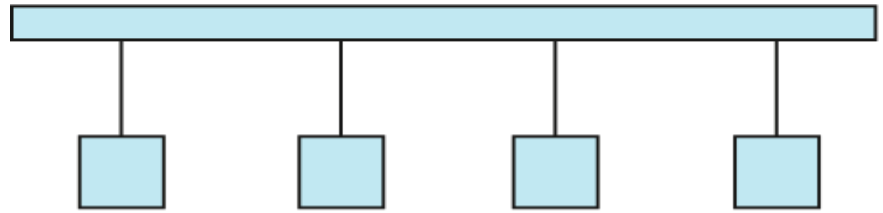
Interconnection Network Architectures

- Parallel systems consists of a set of components (processors, memory, and disks) that can communicate with each other via an **interconnection network.**
  - **Bus**: System components send data on and receive data from a single communication bus;
  - **Mesh**: Components are arranged as nodes in a grid, and each component is connected to all adjacent components
  - **Hypercube**: Components are numbered in binary; components are connected to one another if their binary representations differ in exactly one bit.



Interconnection Network Architectures

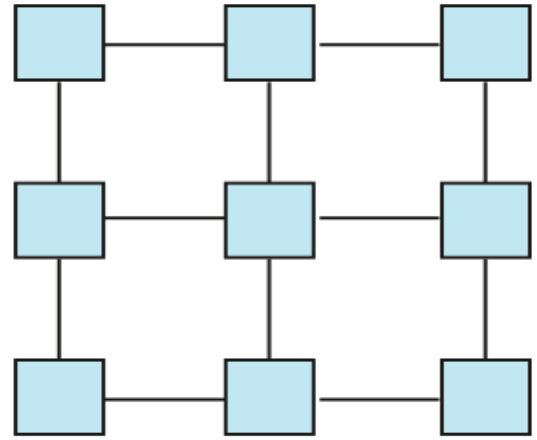
- **Bus architecture**: the system components send data on and receive data from a single communication bus.
- Bus architectures work well for small numbers of processors.
- However, they do not scale well with increasing parallelism, since the bus can handle communication from only one component at a time



(a) bus

Interconnection Network Architectures

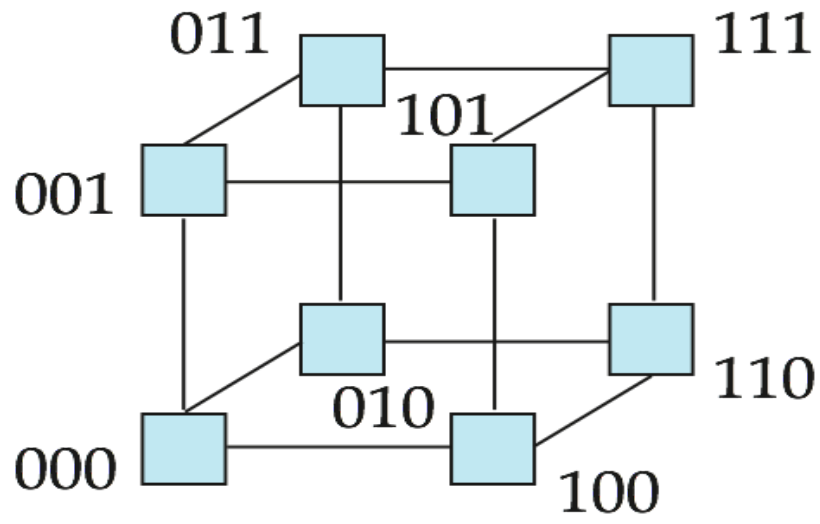
- **Mesh architecture:** the components are arranged as nodes in a grid, and each component connects to all its adjacent components in the grid.
  - In a two-dimensional mesh, each node, other than those on the border, connects to four adjacent nodes
  - Nodes that are not directly connected can communicate with one another by routing messages via intermediate nodes.
- The number of communication links grows as the number of components grows, and so **scales better**.
  - But it may require $2(\sqrt{n}-1)$ links to send message to a node, given n components.



(b) mesh

Interconnection Network Architectures

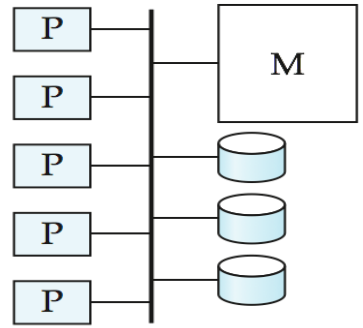
- **Hypercube architecture**: the components are numbered in binary, and a component is connected to another if the binary representations differ in exactly one bit.
  - Each of the *n* components is connected to ***log(n)*** other components.
  - A message from a component can reach any other component by going through at most ***log(n)*** links.
  - **Communication delays** in a hypercube are **significantly lower** than in a mesh.



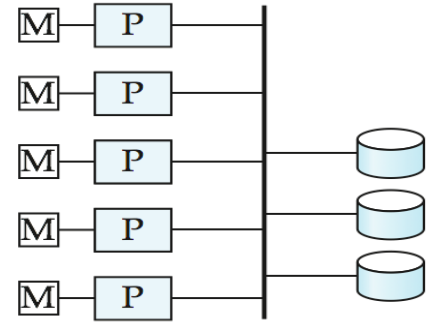
(c) hypercube

Parallel Database Architectures

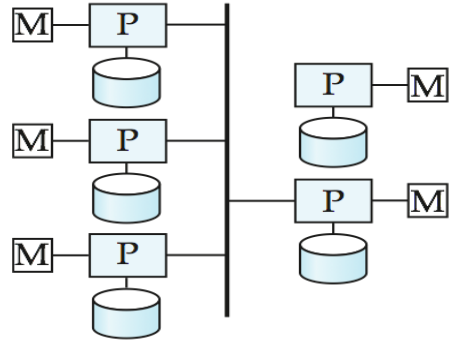
- **Shared memory** -- processors share a common memory
- **Shared disk** -- processors share a common set of disks, also called **clusters**.
- **Shared nothing** -- processors share neither a common memory nor common disk
- **Hierarchical** -- hybrid of the above three architectures



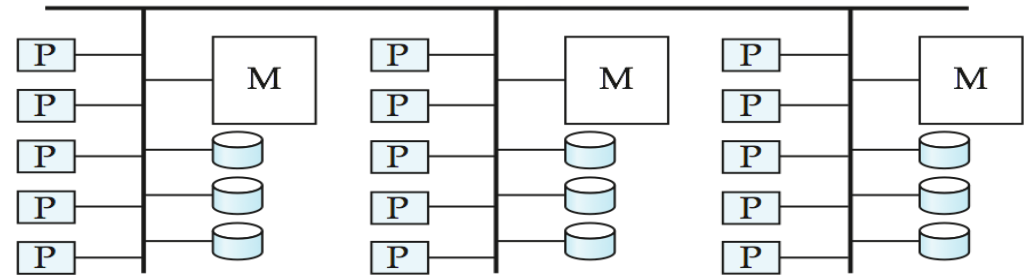
(a) shared memory



(b) shared disk



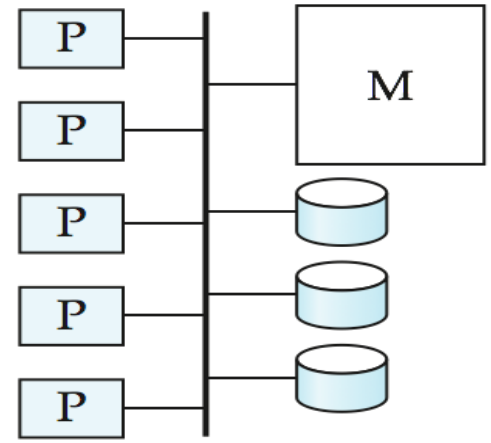
(c) shared nothing



(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.

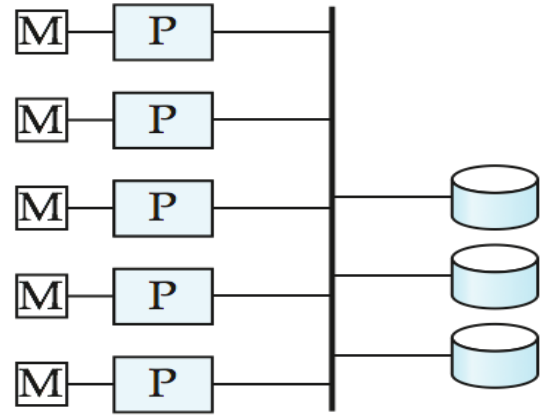


(a) shared memory

- Downside – architecture is not scalable beyond 32 or 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.

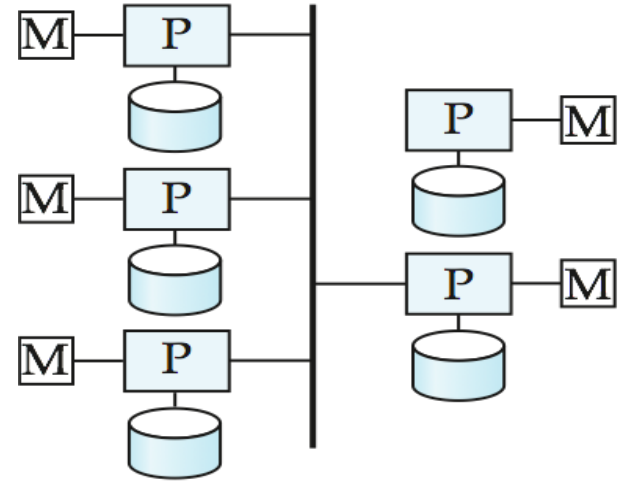


(b) shared disk

- Downside: bottleneck now occurs at interconnection to the disk subsystem, especially for a situation where the database makes a large number of accesses to disks.
- Compared to shared-memory systems, shared-disk systems can scale to a somewhat larger number of processors, but communication across processors is slower.

Shared Nothing

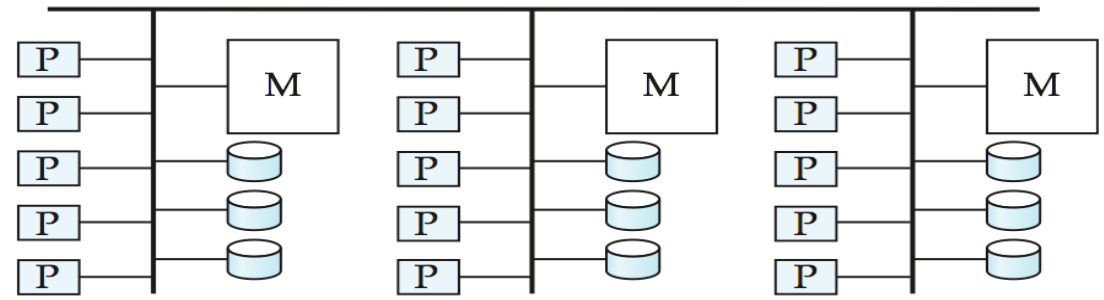
- Each node contains a processor, memory, and one or more disks. Processors at one node communicate with another processor at another node using an interconnection network.
  - A node functions as the server for the data on the disk or disks the node owns.
- Data accessed from local disks (and local memory accesses) do not pass through interconnection network, thereby **minimizing the interference of resource sharing**.



(c) shared nothing

- Shared-nothing multiprocessors can be scaled up to thousands of processors without interference.
- Main drawback: costs of communication and of non-local disk access, since sending data involves software interaction at both ends.

Hierarchical

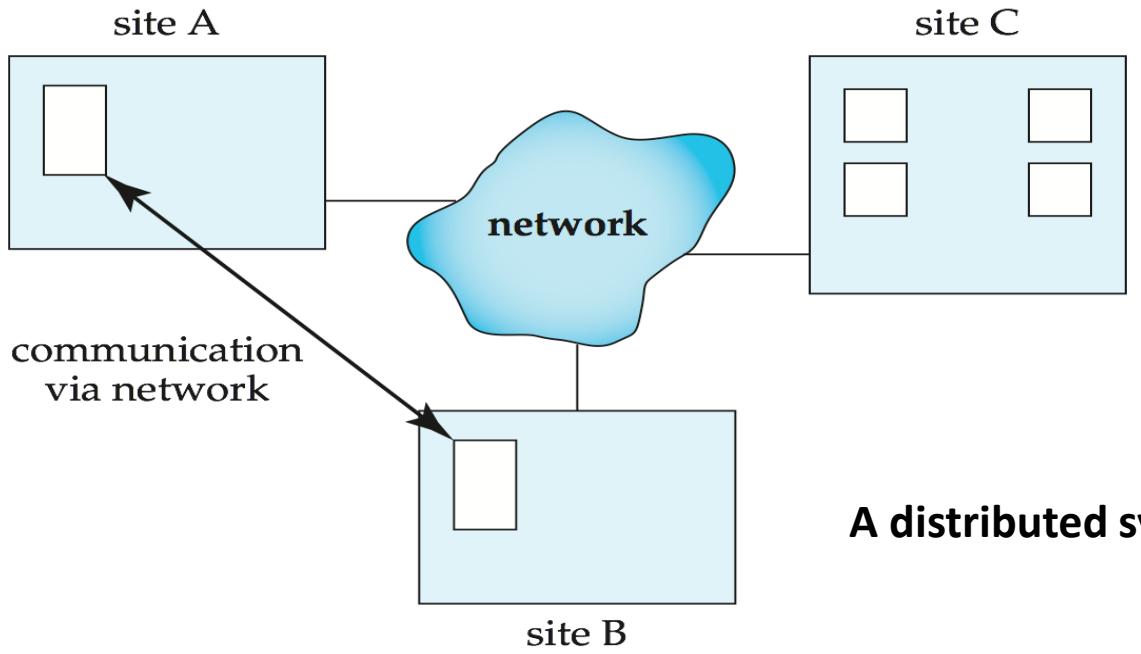


(d) hierarchical

- 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
  - Logically it is a single shared memory, but physically are multiple disjoint memory systems, allowing each processor to view the disjoint memories as a single virtual memory.

Distributed Database Systems

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



A distributed system

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.

Advantages and Disadvantage

- **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.
  - In a centralized system, the administrator of the central site controls the database.
  - In a distributed system, parts of the responsibilities is delegated to the local administrators for each site.
- **Availability** – 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 overheads.

Thank you and any questions?