Fundamentals of DATABASE SYSTEMS

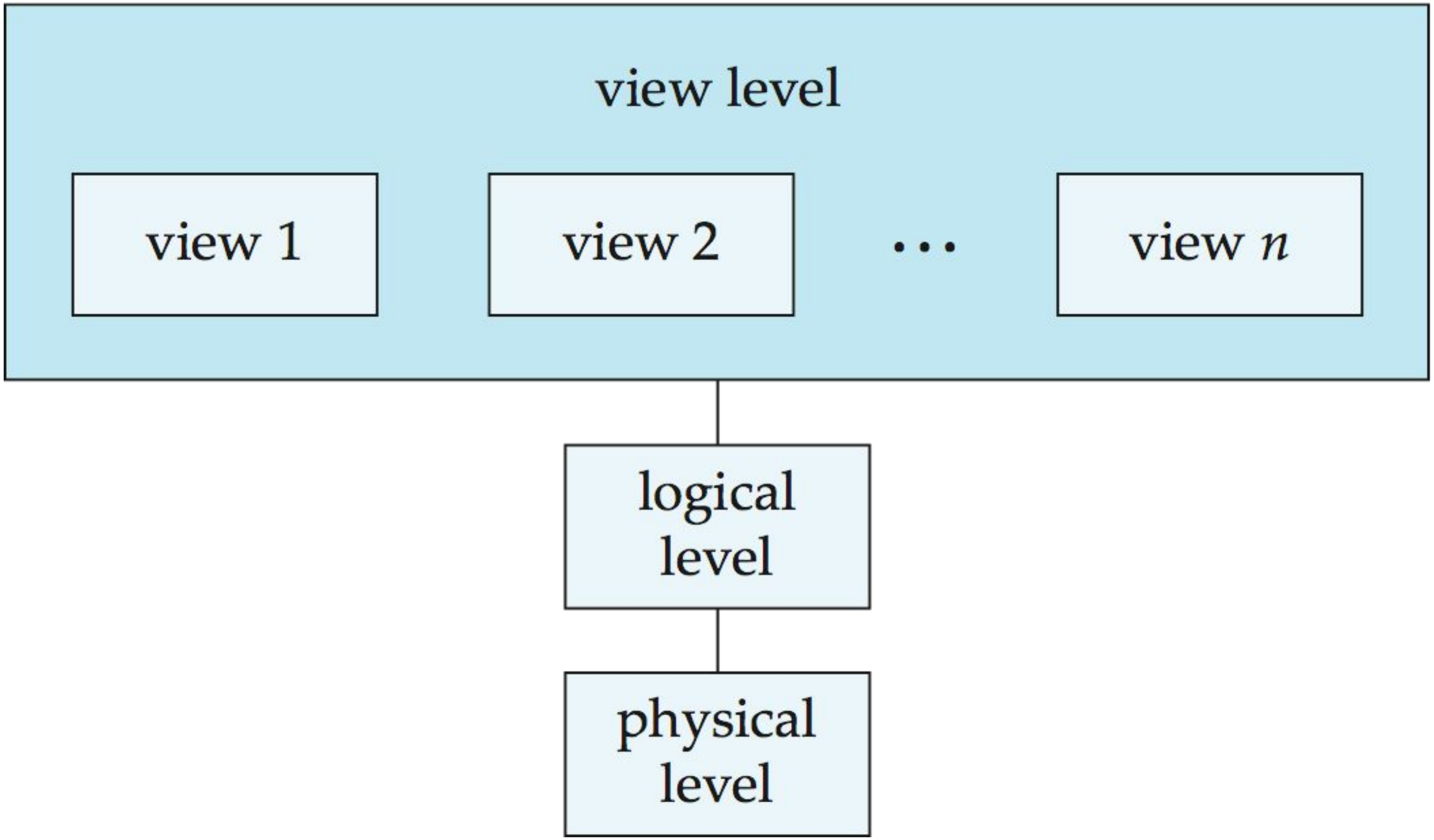
FOURTH EDITION

ELMASRI NAVATHE

Chapter 2

Database System Concepts and Architecture

View of Data



Data Models

- **Data Model**: A set of concepts to describe the *structure* of a database, and certain *constraints* that the database should obey.
- **Data Model Operations**: Operations for specifying database retrievals and updates by referring to the concepts of the data model. Operations on the data model may include *basic operations* and *user-defined operations*.

Data Models

- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semi structured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

Categories of data models

- **Conceptual** (**high-level**, **semantic**) data models: Provide concepts that are close to the way many users *perceive* data. (Also called **entity-based** or **object-based** data models.)
- **Physical** (**low-level**, **internal**) data models: Provide concepts that describe details of how data is stored in the computer.
- **Implementation** (**representational**) data models: Provide concepts that fall between the above two, balancing user views with some computer storage details.

History of Data Models

- Relational Model: proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82. Now in several commercial products (DB2, ORACLE, SQL Server, SYBASE, INFORMIX).
  Network Model: the first one to be implemented by Honeywell in 1964-65 (IDS System). Adopted heavily due to the support by CODASYL (CODASYL - DBTG report of 1971). Later implemented in a large variety of systems - IDMS (Cullinet - now CA), DMS 1100 (Unisys), IMAGE (H.P.), VAX -DBMS (Digital Equipment Corp.).
- Hierarchical Data Model: implemented in a joint effort by IBM and North American Rockwell around 1965. Resulted in the IMS family of systems. The most popular model. Other system based on this model: System 2k (SAS inc.)

History of Data Models

- Object-oriented Data Model(s): several models have been proposed for implementing in a database system. One set comprises models of persistent O-O Programming Languages such as C++ (e.g., in OBJECTSTORE or VERSANT), and Smalltalk (e.g., in GEMSTONE). Additionally, systems like O_2, ORION (at MCC - then ITASCA), IRIS (at H.P.- used in Open OODB).
- Object-Relational Models: Most Recent Trend. Started with Informix Universal Server. Exemplified in the latest versions of Oracle-10i, DB2, and SQL Server etc. systems.

Hierarchical Model

- ADVANTAGES:
  - Hierarchical Model is simple to construct and operate on
  - Corresponds to a number of natural hierarchically organized domains - e.g., assemblies in manufacturing, personnel organization in companies
  - Language is simple; uses constructs like GET, GET UNIQUE, GET NEXT, GET NEXT WITHIN PARENT etc.
- DISADVANTAGES:
  - Navigational and procedural nature of processing
  - Database is visualized as a linear arrangement of records
  - Little scope for "query optimization"

Network Model

- ADVANTAGES:
  - Network Model is able to model complex relationships and represents semantics of add/delete on the relationships.
  - Can handle most situations for modeling using record types and relationship types.
  - Language is navigational; uses constructs like FIND, FIND member, FIND owner, FIND NEXT within set, GET etc. Programmers can do optimal navigation through the database.
- DISADVANTAGES:
  - Navigational and procedural nature of processing
  - Database contains a complex array of pointers that thread through a set of records.
    Little scope for automated "query optimization"

Schemas versus Instances

- **Database Schema**: The *description* of a database. Includes descriptions of the database structure and the constraints that should hold on the database.
- **Schema Diagram**: A diagrammatic display of (some aspects of) a database schema.
- **Schema Construct**: A component of the schema or an object within the schema, e.g., STUDENT, COURSE.
- **Database Instance**: The actual data stored in a database at a *particular moment in time*. Also called **database state** (or **occurrence**).

Example of a Database Schema

STUDENT

Name	StudentNumber	Class	Major

COURSE

CourseName	CourseNumber	CreditHours	Department

PREREQUISITE

CourseNumber	PrerequisiteNumber

SECTION

SectionIdentifier	CourseNumber	Semester	Year	Instructor

GRADE_REPORT

StudentNumber	SectionIdentifier	Grade

Example of a database instance

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

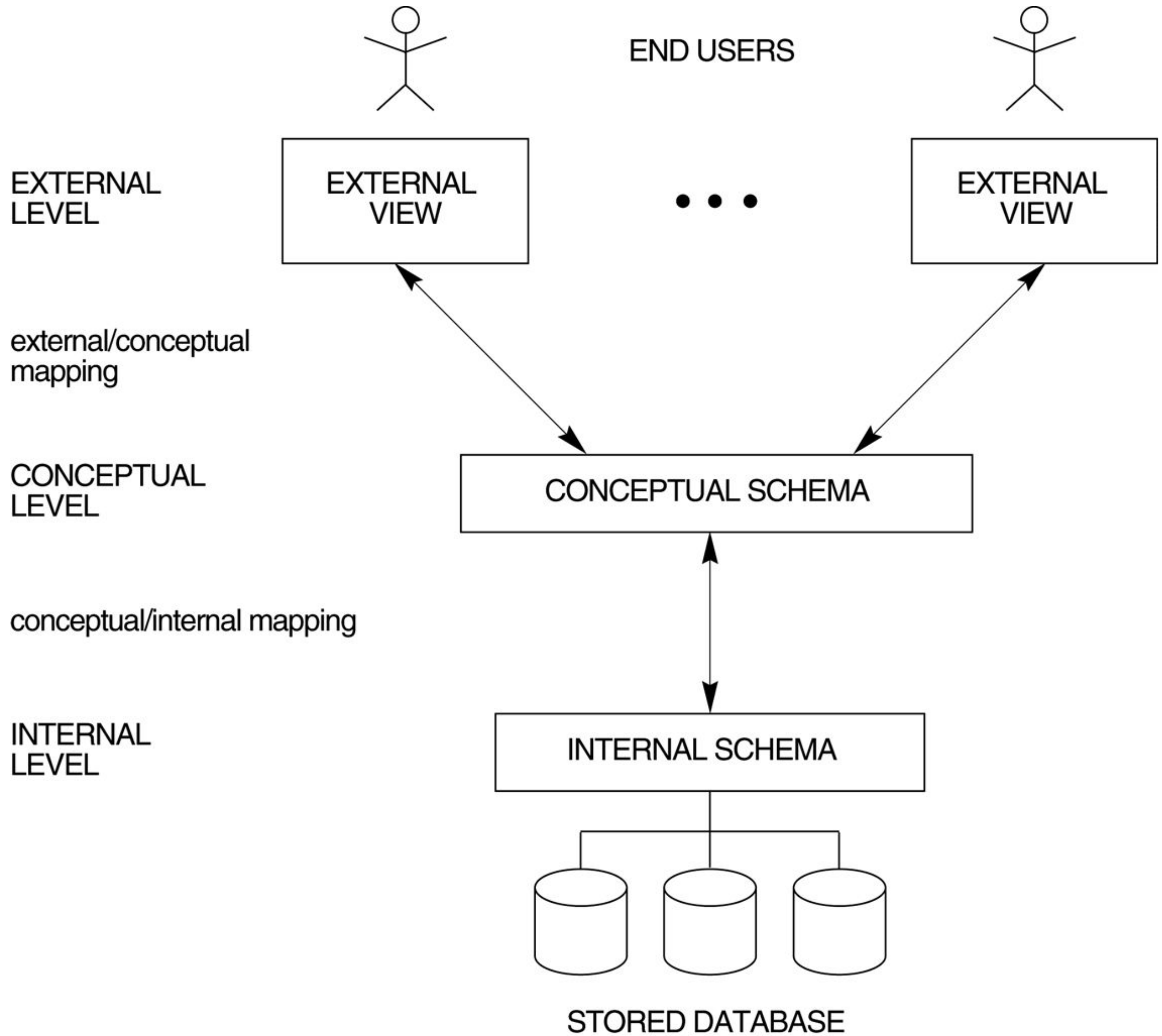
Database Schema Vs. Database State

- **Database State:** Refers to the content of a database at a moment in time.
- **Initial Database State:** Refers to the database when it is loaded
- **Valid State:** A state that satisfies the structure and constraints of the database.
- **Distinction**
  - The **database schema** changes *very infrequently*. The **database state** changes *every time the database is updated.*
  - **Schema** is also called **intension**, whereas **state** is called **extension**.

Three-Schema Architecture

- Proposed to support DBMS characteristics of:
  - **Program-data independence**.
  - Support of **multiple views** of the data.

FIGURE 2.2
The three-schema architecture.



Three-Schema Architecture

- Defines DBMS schemas at *three levels*:
  - **Internal schema** at the internal level to describe physical storage structures and access paths. Typically uses a *physical* data model.
  - **Conceptual schema** at the conceptual level to describe the structure and constraints for the *whole* database for a community of users. Uses a *conceptual* or an *implementation* data model.
  - **External schemas** at the external level to describe the various user views. Usually uses the same data model as the conceptual level.

Three-Schema Architecture

Mappings among schema levels are needed to transform requests and data. Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.

Data Independence

- **Logical Data Independence**: The capacity to change the conceptual schema without having to change the external schemas and their application programs.
- **Physical Data Independence**: The capacity to change the internal schema without having to change the conceptual schema.

  For example, the internal schema may be changed when certain file structures are reorganized or new indexes are created to improve database performance

Examples of changes under Physical Data Independence

- Due to Physical independence, any of the below change will not affect the conceptual 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 say C drive to D Drive

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

Data Independence(Contd..)

When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence.

The higher-level schemas themselves are *unchanged*. Hence, the application programs need not be changed since they refer to the external schemas.

DBMS Languages

- **Data Definition Language (DDL)**: Used by the DBA and database designers to specify the *conceptual schema* of a database.
- In many DBMSs, the DDL is also used to define internal and external schemas (views).
- In some DBMSs, separate **storage definition language (SDL)** and **view definition language (VDL)** are used to define internal and external schemas.

DBMS Languages

- **Data Manipulation Language (DML)**: Used to specify database retrievals and updates.
  - DML commands (**data sublanguage**) can be *embedded* in a general-purpose programming language (**host language**), such as COBOL, C or an Assembly Language.
  - Insert, update and delete are examples of DML.
  - Alternatively, *stand-alone* DML commands can be applied directly (**query language**).

Types of DML

- **High Level or Non-procedural Languages:** For e.g., SQL, are *set-oriented* and specify what data to retrieve rather than how to retrieve. Also called *declarative* languages.
- **Low Level** or **Procedural Languages:** Retrieve data one record-at-a-time; they specify *how* to retrieve data and include constructs such as looping are needed to retrieve multiple records, along with positioning pointers.

Types of 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
- SQL is the most widely used query language

DBMS Interfaces

- Stand-alone query language interfaces.
- Programmer interfaces for embedding DML in programming languages:
  - Pre-compiler Approach
  - Procedure (Subroutine) Call Approach
- User-friendly interfaces:
  - Menu-based, popular for browsing on the web
  - Forms-based, designed for naïve users
  - Graphics-based (Point and Click, Drag and Drop etc.)
  - Natural language: requests in written English
  - Combinations of the above

Other DBMS Interfaces

- Speech as Input (?) and Output
- Web Browser as an interface
- Parametric interfaces (e.g., bank tellers) using function keys.
- Interfaces for the DBA:
  - Creating accounts, granting authorizations
  - Setting system parameters
  - Changing schemas or access path

Database System Utilities

- To perform certain functions such as:
  - *Loading* data stored in files into a database. Includes data conversion tools.
  - *Backing up* the database periodically on tape.
  - *Reorganizing* database file structures.
  - *Report generation* utilities.
  - *Performance monitoring* utilities.
  - Other functions, such as *sorting*, *user monitoring*, *data compression*, etc.

Other Tools

- **Data dictionary / repository**:
  - Used to store schema descriptions and other information such as design decisions, application program descriptions, user information, usage standards, etc.
  - *Active* data dictionary is accessed by DBMS software and users/DBA.
  - *Passive* data dictionary is accessed by users/DBA only.
- **Application Development Environments and CASE (computer-aided software engineering) tools:**
  - Examples – Power builder (Sybase), Builder (Borland)

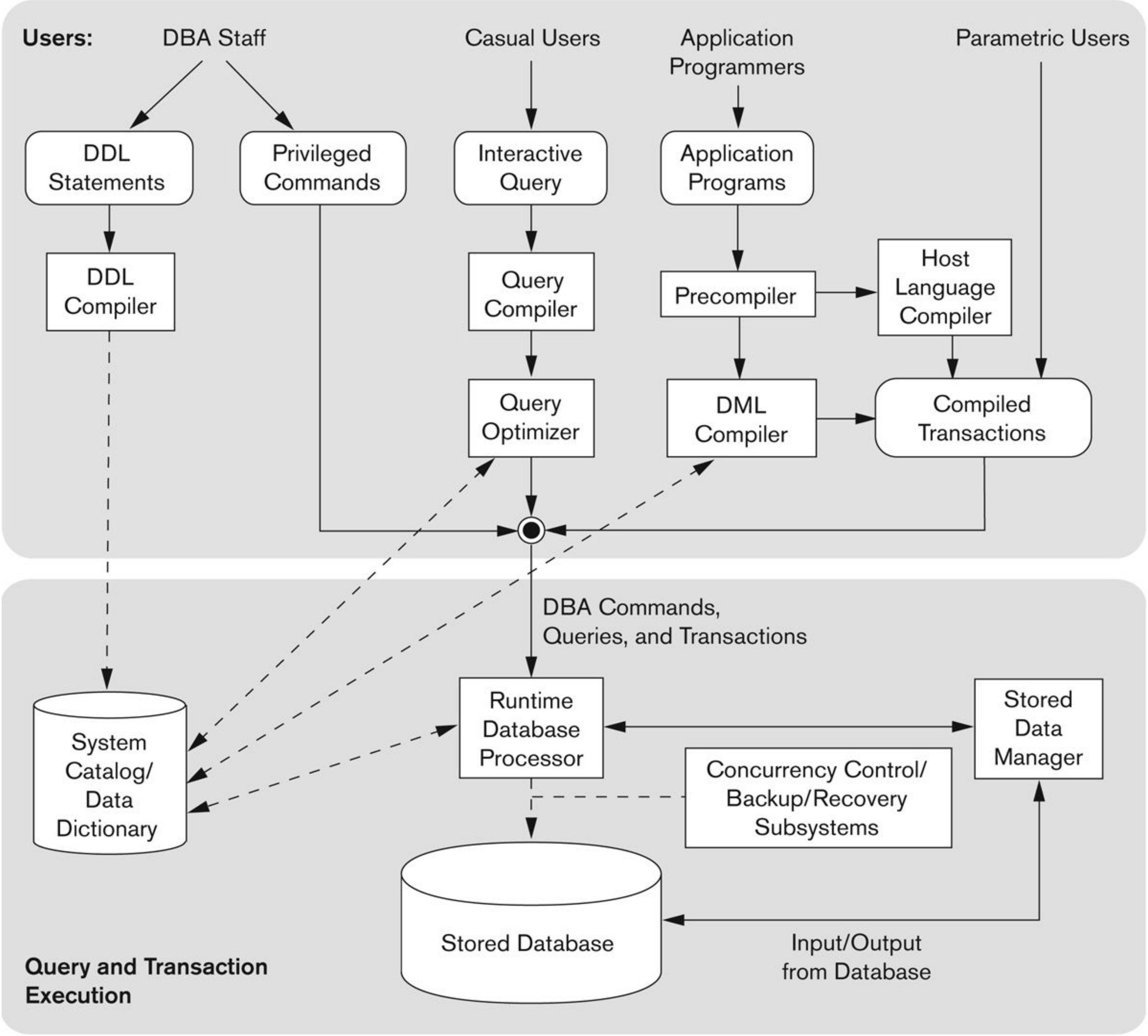


Figure 2.3
Component modules of a DBMS and their interactions.

Centralized and Client-Server Architectures

- **Centralized DBMS:** combines everything into single system including- DBMS software, hardware, application programs and user interface processing software.

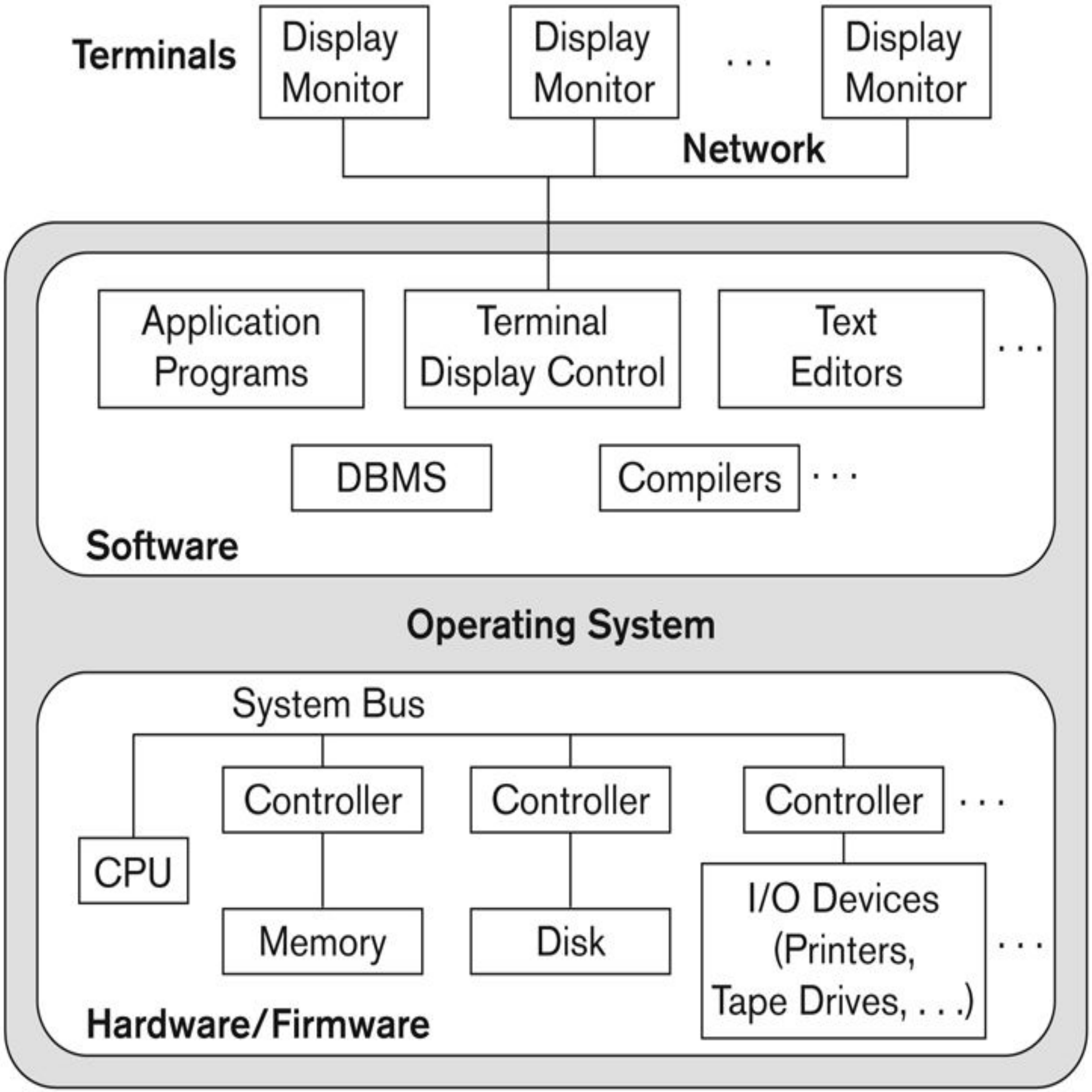


Figure 2.4
A physical centralized architecture.

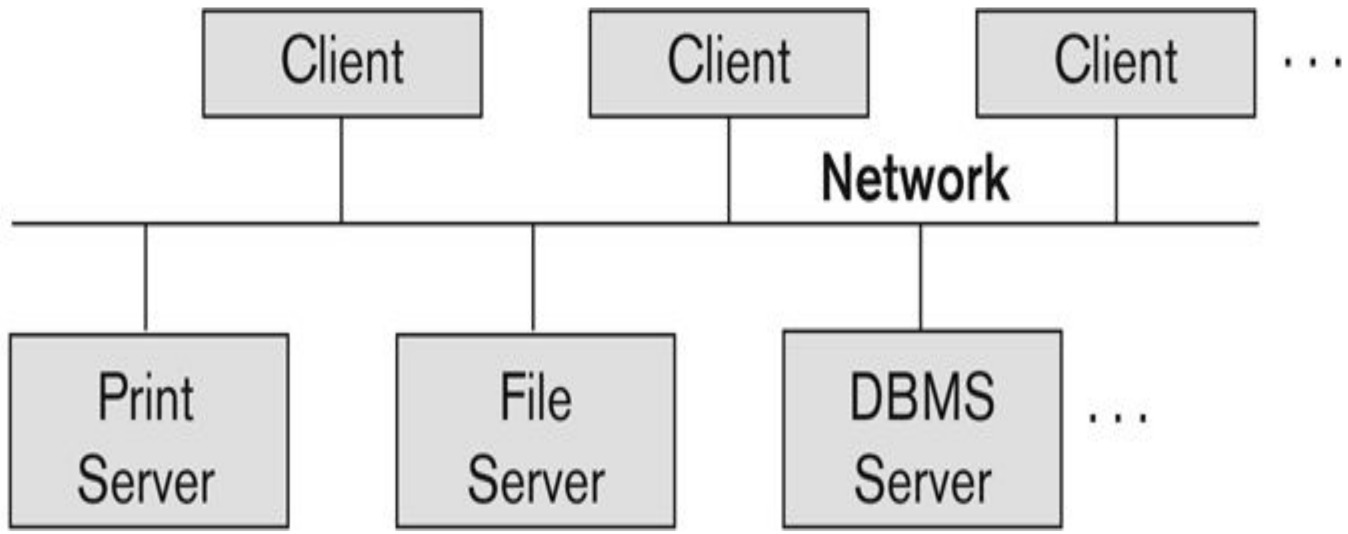
Basic Client-Server Architectures

- **Specialized Servers with Specialized functions**
- **Clients**
- **DBMS Server**

Specialized Servers with Specialized functions:

- File Servers
- Printer Servers
- Web Servers
- E-mail Servers

Figure 2.5
Logical two-tier client/server architecture.



Clients:

- Provide appropriate interfaces and a client-version of the system to access and utilize the server resources.
- Clients maybe diskless machines or PCs or Workstations with disks with only the client software installed.
- Connected to the servers via some form of a network.
- (LAN: local area network, wireless network, etc.)

DBMS Server

- Provides database query and transaction services to the clients
- Sometimes called query and transaction servers

Two Tier Client-Server Architecture

- **User Interface Programs and Application Programs** run on the client side
- Interface called **ODBC (Open Database Connectivity** – see Ch 9**)** provides an Application program interface (API) allow client side programs to call the DBMS. Most DBMS vendors provide ODBC drivers.

Two Tier Client-Server Architecture

- A client program may connect to several DBMSs.
- Other variations of clients are possible: e.g., in some DBMSs, more functionality is transferred to clients including data dictionary functions, optimization and recovery across multiple servers, etc. In such situations the server may be called the **Data Server**.

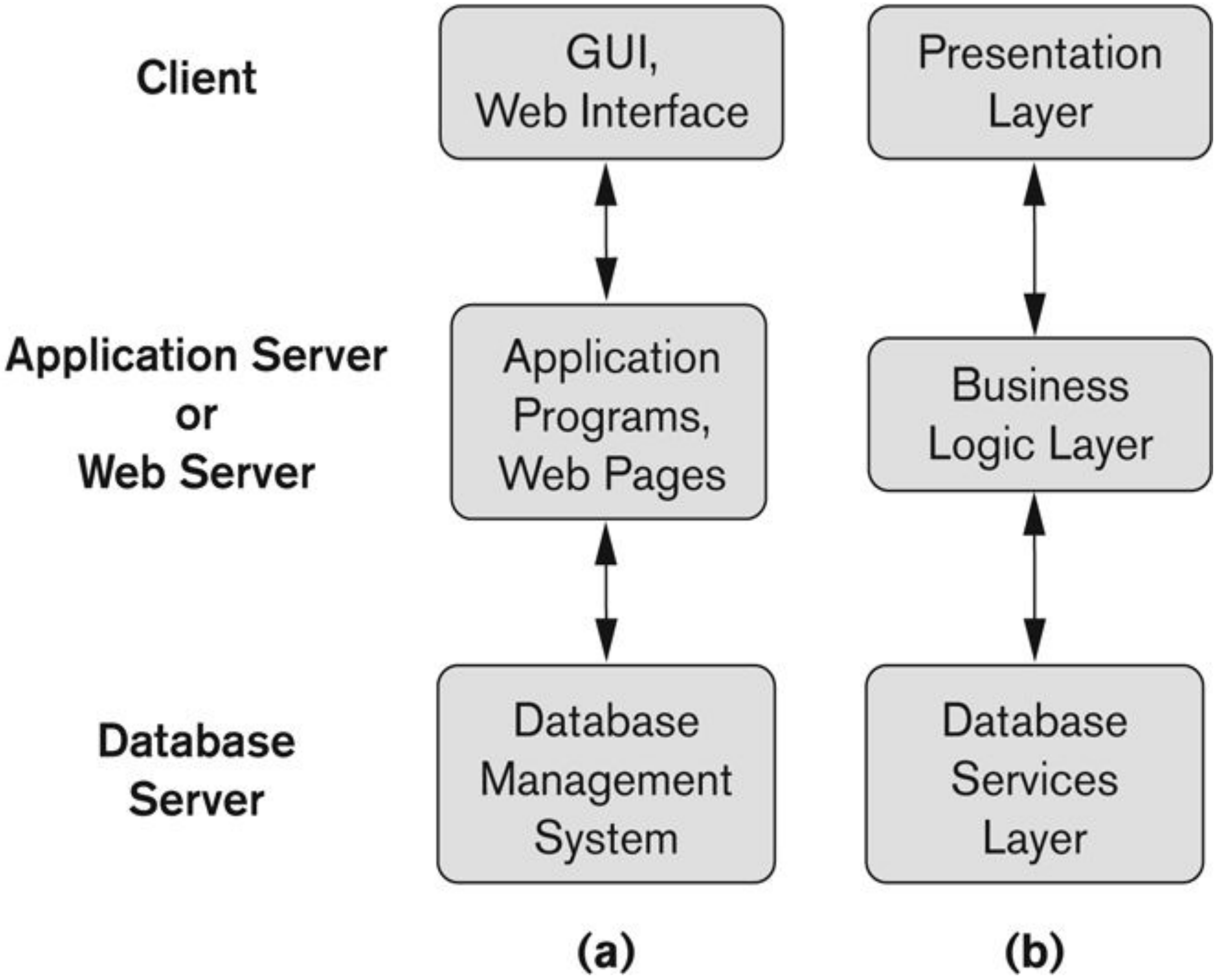
Three Tier Client-Server Architecture

- Common for **Web applications**
- Intermediate Layer called **Application Server** or **Web Server:**
  - stores the web connectivity software and **the rules and business logic (constraints)** part of the application used to access the right amount of data from the database server
  - acts like a conduit for sending partially processed data between the database server and the client.
- **Additional Features- Security:**
  - encrypt the data at the server before transmission
  - decrypt data at the client

Three-tier Architecture Can Enhance Security

- Database server only accessible via middle tier
- Clients cannot directly access database server
- Clients contain user interfaces and Web browsers
- The client is typically a PC or a mobile device connected to the Web

Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.



Classification of DBMSs

- **Based on the data model used**:
  - Traditional: Relational, Network, Hierarchical.
  - Emerging: Object-oriented, Object-relational.
- **Other classifications:**
  - Single-user (typically used with micro-computers) vs. multi-user (most DBMSs).
  - Centralized (uses a single computer with one database) vs. distributed (uses multiple computers, multiple databases)

Classification of DBMSs

Distributed Database Systems *have now come to be known as client server based database systems because they do not support a totally distributed environment, but rather a set of database servers supporting a set of clients.*

Variations of Distributed Environments:

- **Homogeneous DDBMS**
- **Heterogeneous DDBMS**
- **Federated or Multidatabase Systems**