Unit 2
Database System – Concepts and Architecture

- Data Models, Schemas, and Instances
- Three-Schema Architecture and Data Independence
- Database Languages and Interfaces
- Database System Environment
- Centralized and Client/Server Architectures for DBMSs
- Classification of Database Management Systems

Data Models

- One of the main characteristics of the database approach is to support ***data abstraction***.
- **Data abstraction** generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features.
- A **data model** – a collection of concepts that can be used to describe the structure of a database such as data types, relationships, and constraints – provides the necessary means to achieve data abstraction.
- Most data models also include a set of basic operations for specifying retrievals and updates on the database. Data models also allow to specify a set of valid user-defined operations that are allowed on the database objects.

Data Models

- **Categories of Data Models:**
  - We categorize data models according to the types of concepts they use to describe the database structure.
  - **High-level** or **conceptual data models** provide concepts that are close to the way many users perceive data, whereas **low-level** or **physical data models** provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks.
  - Concepts provided by physical data models are generally meant for computer specialists, not for end users.
  - Between these two data models is a class of **representational** (or **implementation**) **data models**, which provide concepts that may be easily understood by end users but that are not too far away from the way data is organized in computer storage.

Data Models

- Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.
- Conceptual data models use concepts such as **entities**, **attributes**, and **relationships**. An entity represents a real-world object or concept. An attribute represents some property of interest that further describes an entity. A relationship among two or more entities represents an association among the entities. **Entity-relationship model** is a popular high-level conceptual data model.
- Representational or implementation data models include the widely used **relational data model**, as well as **legacy data models** (**network** and **hierarchical models** that have been widely used in the past). Representational data models represent data by using record structures and hence are sometimes called **record-based data models**.

Data Models

- **Object data model** is a new family of higher-level implementation data models that are closer to conceptual data models. A standard for object databases called the ODMG object model has been proposed by the Object Data Management Group (ODMG).
- **Physical data models** describe how data is stored as files in the computer by representing information such as record formats, record orderings, and access paths. Access path is a search structure that makes the search for particular database records efficient, such as indexing or hashing.
- Another class of data models is known as **self-describing data models**. The data storage in systems based on these models combines the description of the data with the data values themselves. In traditional DBMSs, the description (schema) is separated from the data. These models include **XML** as well as many of the **key-value stores** and **NOSQL systems**.

Schemas and Instances

- **Schemas:**
  - The description of a database is called the **database schema**, which is specified during database design and is not expected to change frequently.
  - Most data models have certain conventions for displaying schemas as diagrams. **Schema diagram** displays the structure of each record type but not the actual instances of records.
  - A schema diagram displays only some aspects of a schema, such as the names of record types and data items, and some types of constraints. Other aspects are not specified in the schema diagram. Many types of constraints are not represented in schema diagrams.
  - Although, the schema is not supposed to change frequently, it is not uncommon that changes occasionally need to be applied to the schema as the application requirements change. This is known as **schema evolution**.

Schemas and Instances

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Fig: Schema Diagram

Schemas and Instances

- **Instances and Database States:**
  - Actual data in databases may change frequently when we insert or delete a record or change the value of a data item in a record.
  - The data in the database at a particular moment in time is called a **database state** or **snapshot**. It is also called the current set of **occurrences** or **instances** in the database.
  - When we define a new database, we specify its database schema only to the DBMS. At this point, the corresponding database state is the empty state with no data. We get the initial state of the database when the database is first populated or loaded with the initial data. From then on, every time an update operation is applied to the database, we get another database state. At any point in time, the database has a current state.

Schemas and Instances

- The DBMS is partly responsible for ensuring that every state of the database is a valid state, that is, a state that satisfies the structure and constraints specified in the schema.
- The DBMS stores the descriptions of the schema constructs and constraints – also called the meta-data – in the DBMS catalog so that DBMS software can refer to the schema whenever it needs to.
- The schema is sometimes called the **intension**, and a database state is called an **extension** of the schema.

Three Schema Architecture

- The goal of the three-schema architecture is to separate the user applications from physical database.
- This architecture is also known as the **ANSI/SPARC** (American National Standards Institute/ Standards Planning And Requirements Committee) architecture. This architecture defines schemas at three levels.

1. **Internal Level:**
   - The internal level has an internal schema, which describes the physical storage structure of the database.
   - The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. **Conceptual Level:**
   - The conceptual level has a conceptual schema, which describes the structure of the whole database.

Three Schema Architecture

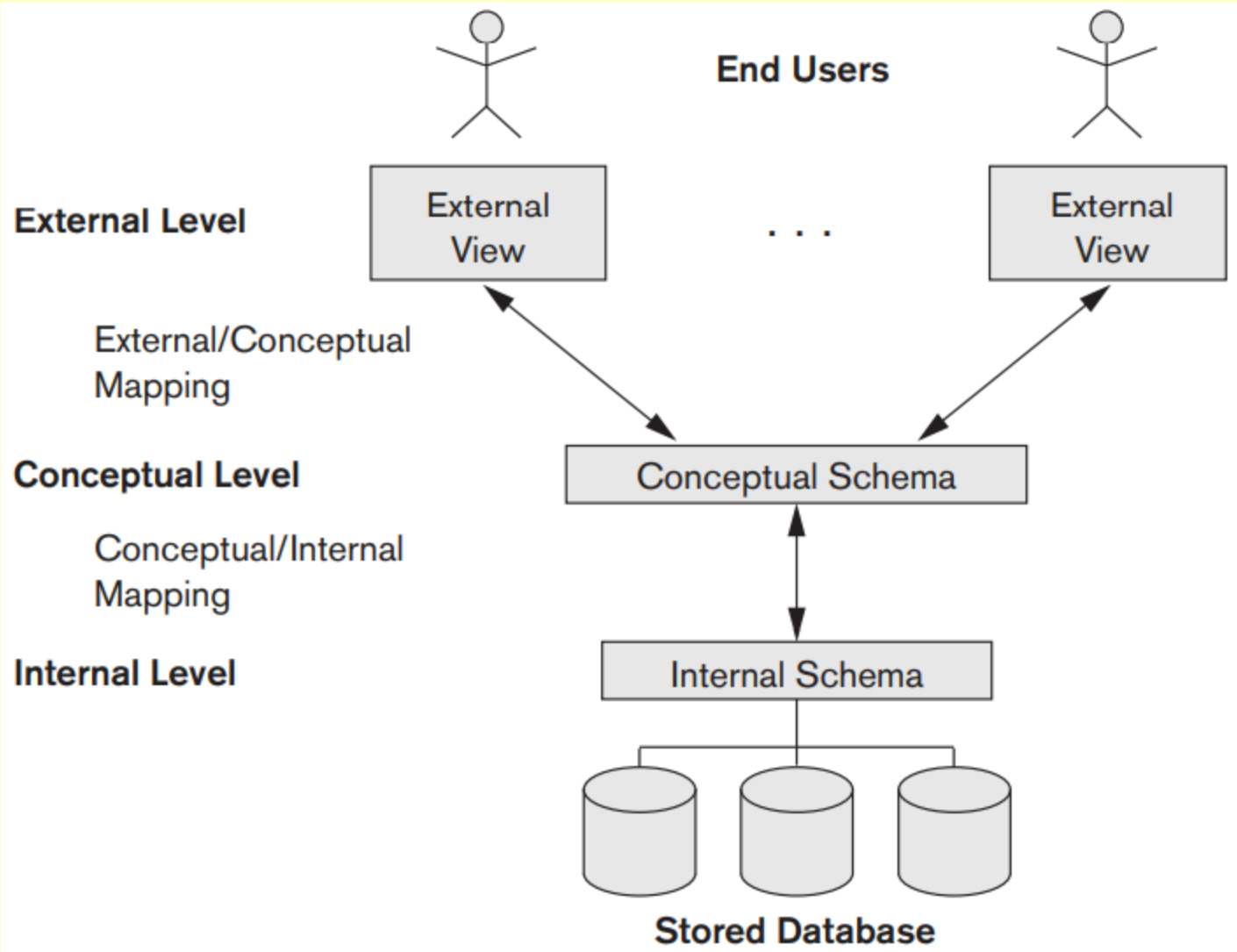

Fig: Three-schema Architecture

Three Schema Architecture

- This schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.
- Usually, a representational data model is used to describe the conceptual schema when a database system is implemented.
- This implementation is often based on a conceptual schema design in a high-level data model.

3. External or View Level:

- The external or view level includes a number of external schemas or user views.
- This schema describes part of the database that a particular user group is interested in and hides the rest of the database from that user group.
- Each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level conceptual data model.

Three Schema Architecture

- Notice that the three schemas are only descriptions of data; the actual data is stored at the physical level only.
- The DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database.
- If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view.
- The processes of transforming requests and results between levels are called **mappings**.

Data Independence

- The three-schema architecture can be used to further explain the concept of data independence.
- Data independence can be defined as the capacity to change schema at one level of a database system without having to change the schema at the next higher level.
- We can define two types of data independence: ***logical data independence*** and ***physical data independence***.
- **Logical Data Independence:**
  - It is the capacity to change the conceptual schema without having to change external schemas or application programs.
  - We may change the conceptual schema to expand the database, to change constraints, or to reduce the database.
  - Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence.

Data Independence

- **Physical Data Independence:**
  - It is the capacity to change internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well.
  - Changes to the internal schema may be needed because some physical files were reorganized.
- When the schema is changed at some level, the schema at the next higher level remains unchanged; only the mapping between the two levels is changed. Hence, application programs referring to the higher-level schema need not be changed

DBMS Languages

- In many DBMSs where there is no strict separation of levels, **data definition language (DDL)**, is used to define both conceptual and internal schemas.
- In DBMSs where a clear separation is maintained between the conceptual and internal levels, the **DDL** is used to specify the conceptual schema only and **storage definition language (SDL)** is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages.
- In most relational DBMSs today, there is no specific language that performs the role of **SDL**. Instead, the internal schema is specified by a combination of functions, parameters, and specifications related to storage of files.

DBMS Languages

- For a true three-schema architecture, we would need a third language, the **view definition language (VDL)**, to specify user views and their mappings to the conceptual schema, but in most DBMSs the DDL is used to define both conceptual and external schemas.
- To manipulate data in the database, we need **data manipulation language (DML)**. Typical manipulations include retrieval, insertion, deletion, and modification.
- In current DBMSs, a comprehensive integrated language is used for conceptual schema definition, view definition, and data manipulation. Storage definition is typically kept separate. For example, SQL is comprehensive language.
- There are two main types of DMLs **high-level** or **nonprocedural** and **low-level** or **procedural**.

DBMS Languages

- **High-level DML** can specify and retrieve many records in a single DML statement. So, these are also called **set-at-a-time** or **set-oriented** DMLs. Query in a high-level DML specifies which data to retrieve rather than how to retrieve it. Such languages are therefore also **called declarative**.
- **Low-level DML** must be embedded in a general-purpose programming language. This type of DML typically retrieves individual records or objects from the database. Hence, low-level DMLs are also called **record-at-a-time** DMLs.
- When DML commands are embedded in a programming language, the language is called the **host language** and the DML is called the **data sublanguage**. A high-level DML used in a standalone interactive manner is called a **query language**.

DBMS Interfaces

- **Menu-based Interfaces for Web Clients or Browsing**. List of options (called menus) that lead the user through the formulation of a request. **Pull-down menus** are a very popular technique in Web-based user interfaces.
- **Apps for Mobile Devices.** These interfaces present mobile users with access to their data through a mobile phone or mobile device. For example, mobile banking apps.
- **Forms-based Interfaces.** A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data, or they can fill out only certain entries to retrieve matching data for the remaining entries. Many DBMSs have forms specification languages, such as, SQL*Forms.

DBMS Interfaces

- **Graphical User Interfaces.** A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms.
- **Natural Language Interfaces.** These interfaces accept requests written in English or some other language and attempt to understand them.
- **Keyword-based Database Search.** These are somewhat similar to Web search engines, which accept strings of natural language words and match them with documents at specific sites or Web pages. They use predefined indexes on words and use ranking functions to retrieve and present resulting documents in a decreasing degree of match.

DBMS Interfaces

- **Speech Input and Output.** Applications with limited vocabularies are allowing speech for input and output to access data.
- **Interfaces for Parametric Users.** Parametric users often have a small set of operations that they must perform repeatedly. A special interface is designed and implemented for each known class of naive users. Usually a small set of abbreviated commands is included.
- **Interfaces for the DBA.** Most database systems contain privileged commands that can be used only by the DBA staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing schema, and reorganizing the storage structures.

Database System Environment

- A DBMS is a complex software system and contains different software components.
- DBMS also interacts different types of computer system software.
- **DBMS Component Modules:**
  - The top part of the figure (see next slide) refers to the various users and their interfaces. The lower part shows the internal modules responsible for storage of data and processing of transactions.
  - The **DBA staff** works on defining the database and tuning it by making changes to its definition using the DDL statements and other privileged commands.
  - The **DDL compiler** processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.

Database System Environment

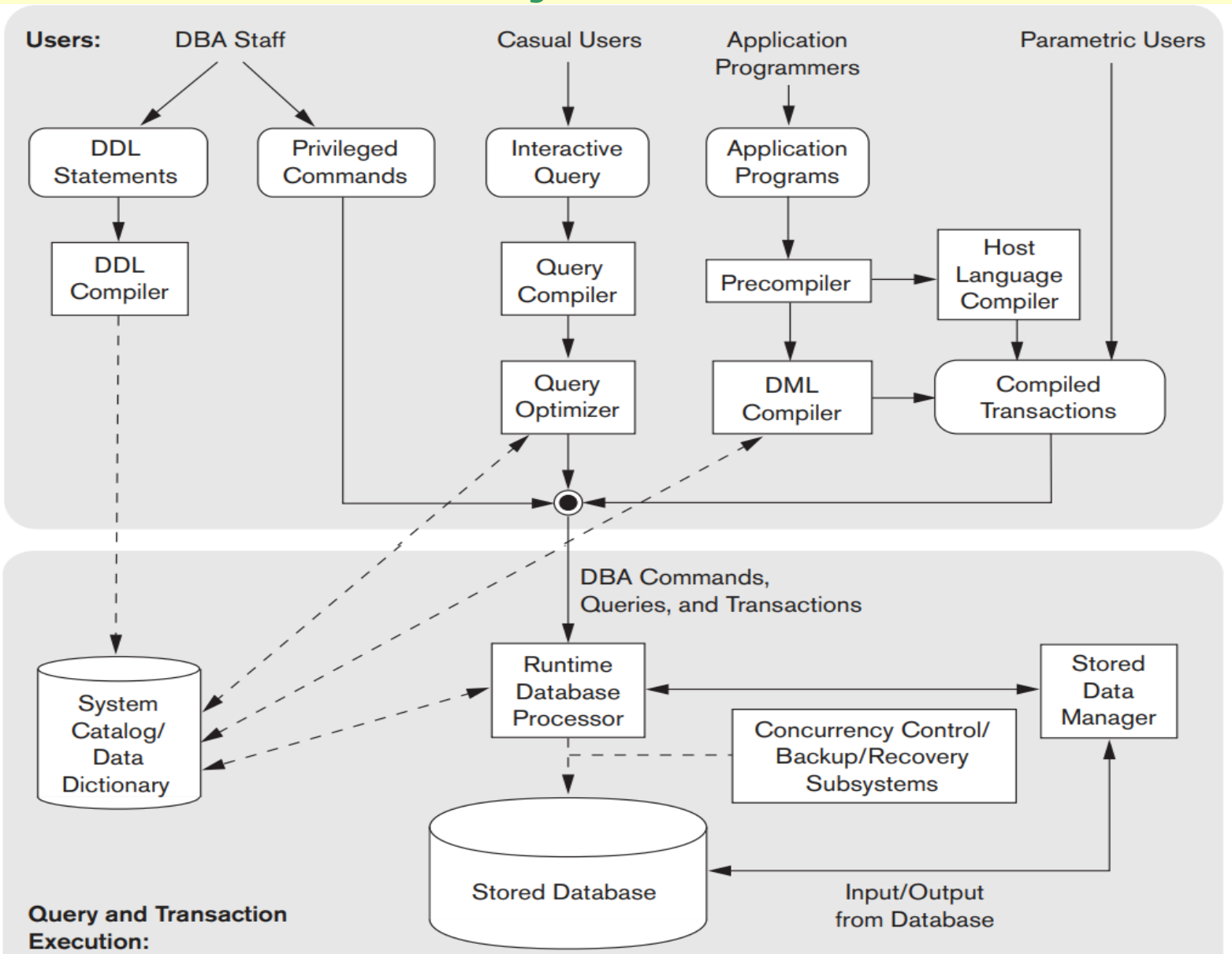

Fig: DBMS components modules

Database System Environment

- **Casual users** and persons with occasional need for information from the database interact using the **interactive query** interface. These queries are parsed and validated for correctness of the query syntax, the names of files and data elements, and so on by a **query compiler** that compiles them into an internal form. This internal query is subjected to **query optimization**.
- **Application programmers** write programs in host languages that are submitted to a **precompiler**. The precompiler extracts DML commands from the program. These commands are sent to the **DML compiler** for compilation into object code for database access. The rest of the program is sent to the **host language compiler**. The object codes for the DML commands and the rest of the program are linked, forming a **canned transaction** whose executable code includes calls to the **runtime database processor**. Canned transactions are executed repeatedly by **parametric users**. Each execution is considered to be a separate transaction.

Database System Environment

- The **runtime database processor** executes (1) the privileged commands, (2) the executable query plans, and (3) the canned transactions with runtime parameters. It works with the **system catalog** and may update it with statistics. It also works with the **stored data manager**, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory. The runtime database processor handles other aspects of data transfer, such as **management of buffers** in the main memory. Some DBMSs have their own buffer management module whereas others depend on the OS for buffer management.
- **Concurrency control** and **backup and recovery systems** are integrated into the working of the runtime database processor for purposes of transaction management.

Database System Environment

- **Database System Utilities:**
  - In addition to different DBMS component modules, most DBMSs have database utilities that help the DBA manage the database system. Common utilities have the following types of functions:
  - **Loading:** A loading utility is used to load existing data files – such as text files or sequential files – into the database by automatically reformatting the data. Transferring data from one DBMS to another is also becoming common.
  - **Backup:** A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium. The backup copy can be used to restore the database in case of catastrophic failure. Incremental backups are also often used, where only changes since the previous backup are recorded.

Database System Environment

- **Database storage reorganization:** This utility can be used to reorganize a set of database files into different file organizations and create new access paths to improve performance.
- **Performance monitoring**: Such a utility monitors database usage and provides statistics to the DBA. The DBA uses the statistics in making decisions such as whether or not to reorganize files or whether to add or drop indexes to improve performance.
- Other utilities may be available for **sorting files**, **handling data compression**, **monitoring access by users**, **interfacing with the network**, and **performing other functions**.

Database System Environment

- **Tools, Application Environments, and Communication Facilities:**
  - **Computer Aided Software Engineering (CASE) tools** are used in the design phase of database systems.
  - **Expanded data dictionary** (or **data/information repository**) **tool** stores information, such as design decisions, usage standards, application program descriptions, and user information in addition to storing catalog information about schemas and constraints. A data dictionary utility is similar to the DBMS catalog, but it includes a wider variety of information.
  - **Application development environments** provide an environment for developing database applications. These environments help in database design, GUI development, querying and updating, and application program development.
  - **Interface with communications software** allows users at locations remote from the database system site to access the database.

Centralized and Client/Server Architectures

- **Centralized DBMS Architecture:**
  - Older architectures used central mainframe computers housing all the DBMS functionality, application programs, and user interfaces. Only display information and controls were sent from the computer to the display terminals, which were connected to the central computer via various types of communications networks.
  - As the prices of hardware declined, most users replaced their terminals with PCs and workstations, and more recently with mobile devices. Database systems used these computers similarly to how they had used display terminals, so that the DBMS itself was still a centralized DBMS in which all the DBMS functionality, application program execution, and user interface processing were carried out on one machine.

Centralized and Client/Server Architectures

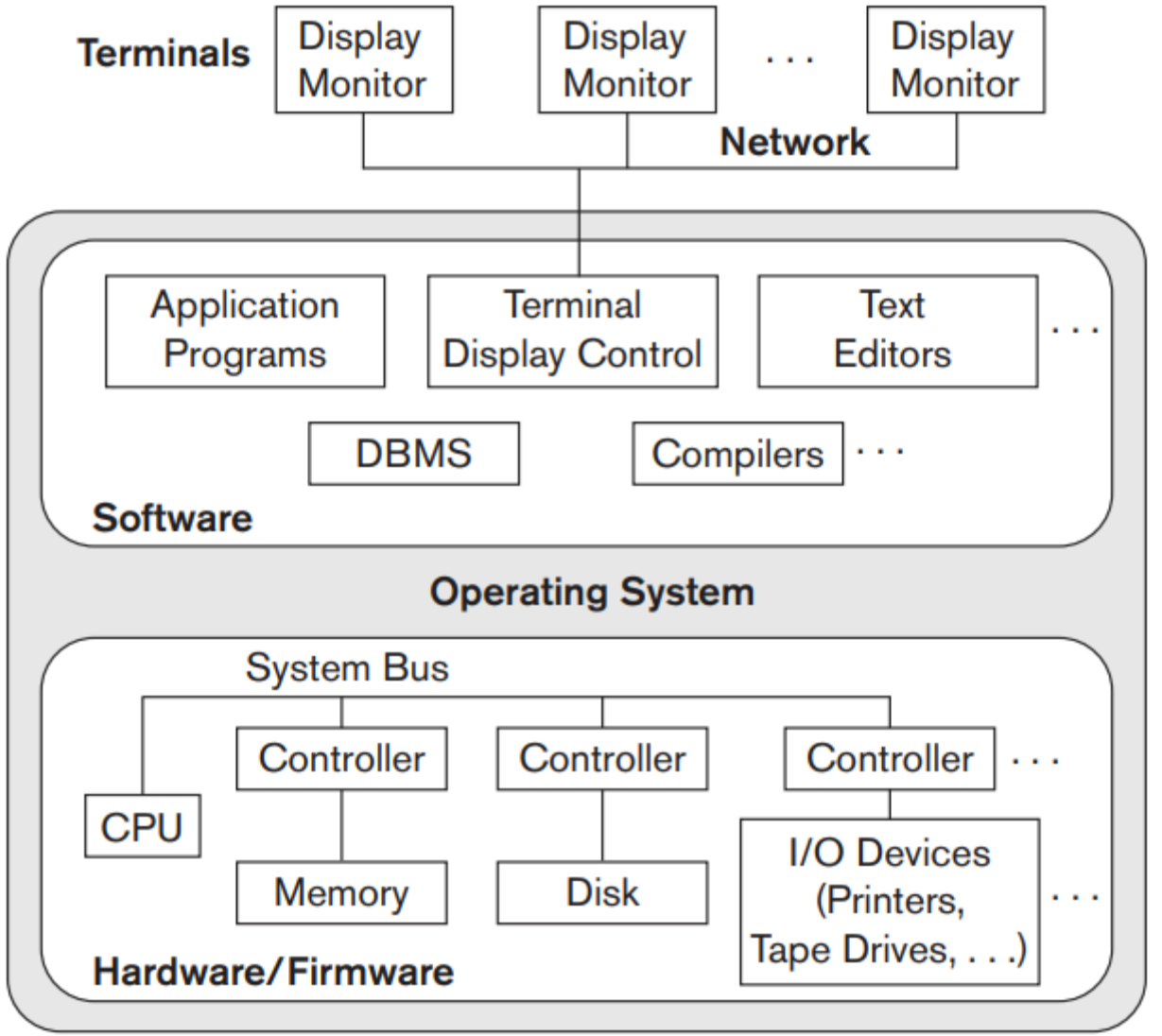

Fig: A centralized architecture

Centralized and Client/Server Architectures

- **Basic Client/Server Architectures:**
  - Specialized servers with specific functionalities, such as file servers, database servers, Web servers etc. will be defined.
  - The client machines provide interfaces to access these servers, as well as with local processing power to run local applications.
  - Some machines would be client sites only. Other machines would be dedicated servers, and others would have both client and server functionality.
  - When a client requires access to additional functionality that does not exist at the client, it connects to a server that provides the needed functionality.
  - In general, some machines install only client software, others only server software, and still others may include both client and server.
  - Two main types of basic DBMS architectures were created on this underlying client/server framework: **two-tier** and **three-tier**.

Centralized and Client/Server Architectures

- **Two-Tier Client/Server Architectures for DBMSs:**
  - The user interface programs and application programs can run on the client side.
  - To access DBMS, the program establishes a connection to the DBMS (which is on the server side); once the connection is created, the client program can communicate with the DBMS.
  - The architectures described here are called two-tier architectures because the software components are distributed over two systems: client and server. The advantages of this architecture are its simplicity and seamless compatibility with existing systems.

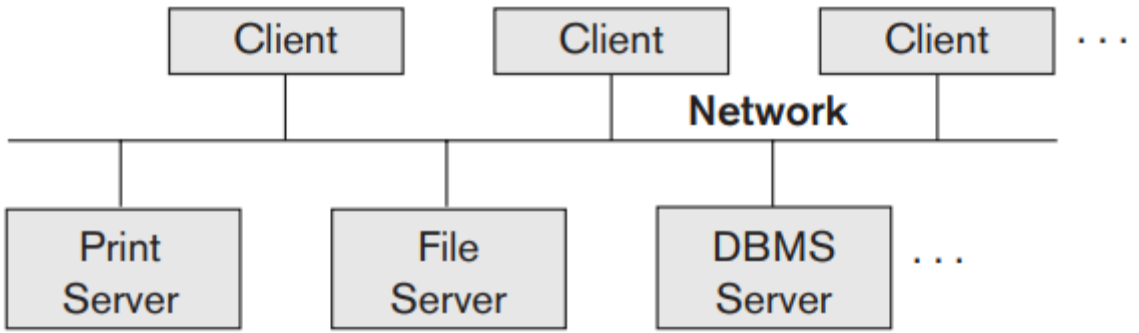

Fig: A two-tier client/server architecture

Centralized and Client/Server Architectures

- **Three-Tier and n-Tier Architectures for Web Applications:**
  - Many Web applications use three-tier architecture, which adds an intermediate layer between the client and the database server. Clients contain user interfaces and Web browsers.
  - This intermediate layer or middle tier is called the application server or the Web server, depending on the application. This server plays an intermediary role by running application programs and storing business rules (procedures or constraints) that are used to access data from the database server.
  - The intermediate server accepts requests from the client, processes the request and sends database queries to the database server, and passes processed data from the database server back to the clients.
  - Thus, the user interface, application rules, and data access act as the three tiers.

Centralized and Client/Server Architectures

- Three-tier architecture can also be used by database and other application package vendors. The presentation layer displays information to the user and allows data entry. The business logic layer handles intermediate rules and constraints before data is passed up to the user or down to the DBMS. The bottom layer includes all data management services.
- It is possible to divide the layers between the user and the stored data further into finer components, thereby giving rise to **n-tier architectures**, where n may be four or five tiers. Typically, the business logic layer is divided into multiple layers. Any one tier can run on an appropriate processor or operating system platform and can be handled independently.

Centralized and Client/Server Architectures

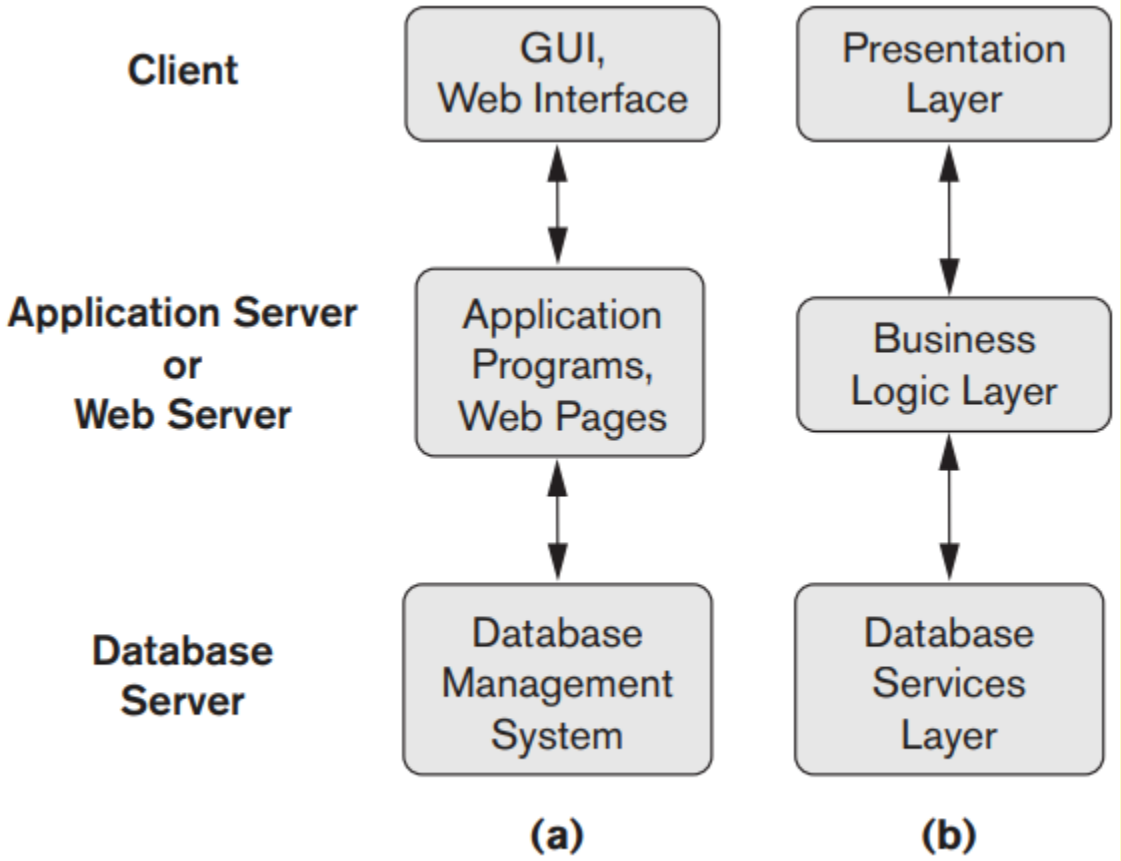

Fig: Three-tier architecture for (a) Web and (b) other applications

Classification of DBMSs

- Several criteria can be used to classify DBMSs.
- **Based on Data Model of DBMS:**
  - **Relational data model:** Relational data model represents a database as a collection of tables. Most relational databases use the high-level query language called SQL. The systems based on this model are known as **SQL systems**.
  - **Object data model:** This model defines a database in terms of objects, their properties, and their operations. Objects with the same structure and behavior belong to a class, and classes are organized into hierarchies.
  - **Object-relational model:** Relational DBMSs have been extending their models to incorporate object database concepts and other capabilities; these systems are referred to as object-relational or extended relational systems.

Classification of DBMSs

- **Big data systems:** These systems are also know as **key-value storage systems** and **NOSQL systems**. Big data systems are based on various data models, with the following four data models most common. The **key-value data model** associates a unique key with each value and provides very fast access to a value given its key. The **document data model** is based on JSON (Java Script Object Notation) and stores the data as documents, which somewhat resemble complex objects. The **graph data model** stores objects as graph nodes and relationships among objects as directed graph edges. Finally, the **column-based data models** store the columns of rows clustered on disk pages for fast access and allow multiple versions of the data.
- **XML model:** XML uses hierarchical tree structures. It combines database concepts with concepts from document representation models. Data is represented as elements with the use of tags.

Classification of DBMSs

- **Legacy data models:** The two legacy data models are the **network** and **hierarchical models**. The network model represents data as record types and stores data using graph like structure. The hierarchical model also represents data as record types and represents data as hierarchical tree structures. Each hierarchy represents a number of related records.

- **Based on Number of users supported by DBMS:**
  - **Single-user systems** support only one user at a time and are mostly used with PCs.
  - **Multiuser systems**, which include the majority of DBMSs, support concurrent multiple users.

Classification of DBMSs

- **Based on Number of Sites over which DBMS is Distributed:**
  - A **centralized DBMS** stores data at a single computer site. A centralized DBMS can support multiple users, but the DBMS and the database reside totally at a single computer site.
  - A **distributed DBMS** (DDBMS) can have the actual database and DBMS software distributed over many sites connected by a computer network. The data is often replicated on multiple sites so that failure of a site will not make some data unavailable. **Homogeneous DDBMSs** use the same DBMS software at all the sites, whereas **heterogeneous DDBMSs** can use different DBMS software at each site. In **federated DBMS** (or **multidatabase system**), the participating DBMSs are loosely coupled and have a degree of local autonomy.

Classification of DBMSs

- **Based on Cost of DBMS:**
  - It is difficult to propose a classification of DBMSs based on cost.
  - Today we have open source (free) DBMS products like MySQL and PostgreSQL.
  - The giant systems are being sold in modular form. Commercial DBMSS offer additional specialized functionality whey purchased separately.
  - DBMSs are sold in the form of licenses: **site licenses** (unlimited use of the database system with any number of copies running at the customer site), **seat licenses** (number of concurrent users or the number of user seats at a location), and **single user license**.
  - In addition, data warehousing and mining features, as well as support for additional data types, are made available at extra cost.
  - It is possible to pay millions of dollars for the installation and maintenance of large database systems annually.

Classification of DBMSs

- **Based on Types of Access Path Options for Storing Files:**
  - One well-known family of DBMSs is based on inverted file structures.
  - Fully indexed databases provide access by any keyword.
- **Based on Purpose of DBMS:**
  - A DBMS can be **general purpose** or **special purpose**.
  - When performance is a primary consideration, a special-purpose DBMS can be designed and built for a specific application; such a system cannot be used for other applications without major changes.