Database Architecture

CS 341 Database Systems

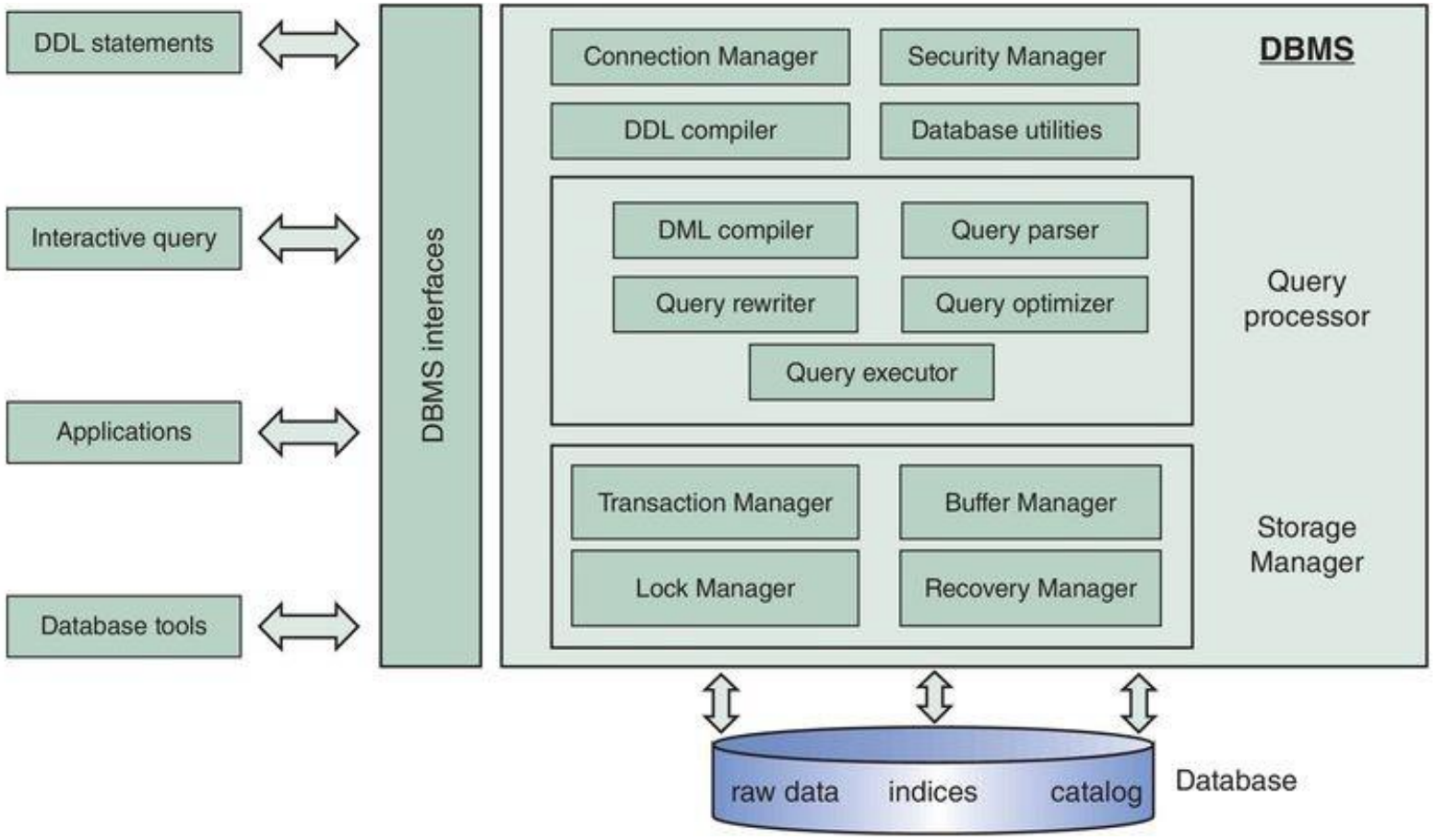
What is Database Architecture

Representation of DBMS design

Helps design, develop, implement, maintain the database management system.

Understanding of individual components of a database

Architecture of a database management system (DBMS)

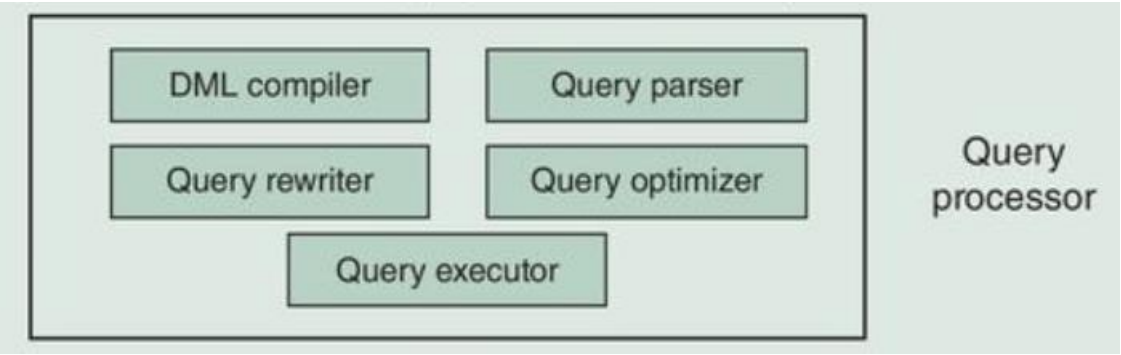


DBMS Architecture

- **Connection manager** provides facilities to setup a database connection (locally or through a network)
- **Security manager** verifies whether a user has the right privileges
- **DDL compiler** compiles the data definitions specified in DDL. Upon successful compilation, it registers the data definitions in the catalog

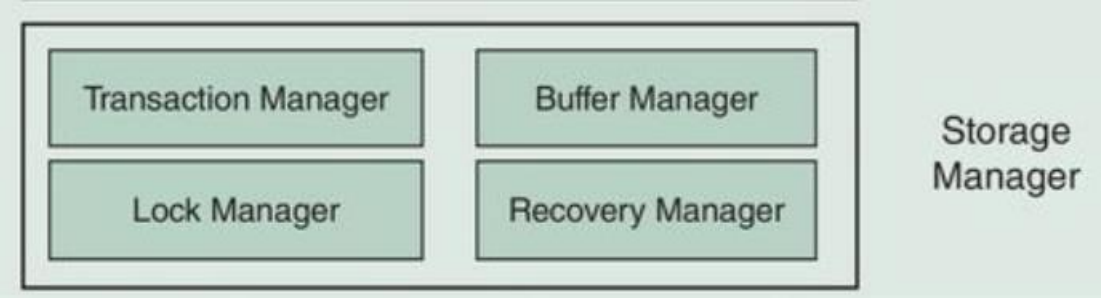
Query processor

- Query processor assists in the execution of database queries such as retrieval, insertion, update or removal of data
- Key components:
  - DML compiler
  - Query parser
  - Query rewriter
  - Query optimizer
  - Query executor



Storage Manager

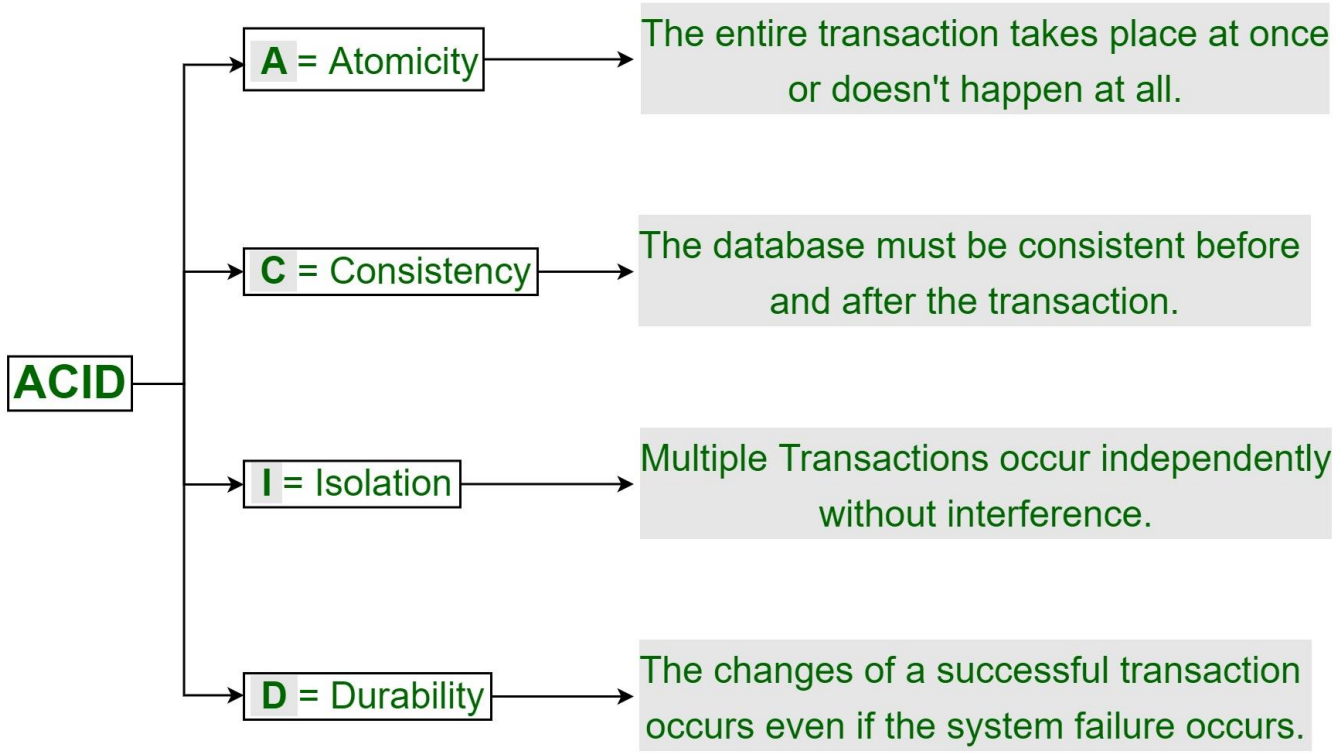
- Storage manager governs physical file access and supervises the correct and efficient storage of data
- Storage manager consists of
  - Transaction Manager
  - Buffer Manager
  - Lock Manager
  - Recovery Manager



Transaction Manager

- Transaction manager supervises execution of database transactions
  - a database transaction is a sequence of read/write operations considered to be an atomic unit
- Transaction manager creates a schedule with interleaved read/write operations
- Transaction manager guarantees **ACID properties**
- COMMIT a transaction upon successful execution and ROLLBACK a transaction upon unsuccessful execution

ACID Properties in DBMS



DBMS Architecture

- **Buffer manager** manages buffer memory of the DBMS
- **Lock manager** provides concurrency control which ensures data integrity at all times. Lock manager makes use of a *locking protocol* which describes the locking rules (Read/Write Locks), and a lock table with the lock information.
- **Recovery manager** keeps track of all database operations in a log file. Will be called upon to undo actions of aborted transactions of during crash recovery.

Interacting with the Database

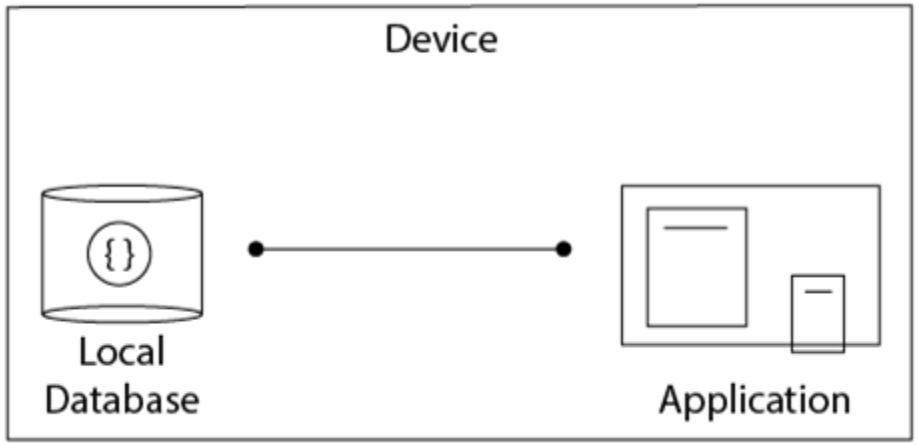
- One-Tier Architecture or Single Tier Architecture
- Two-Tier Architecture
- Three-Tier Architecture

Tier = Level = Layer

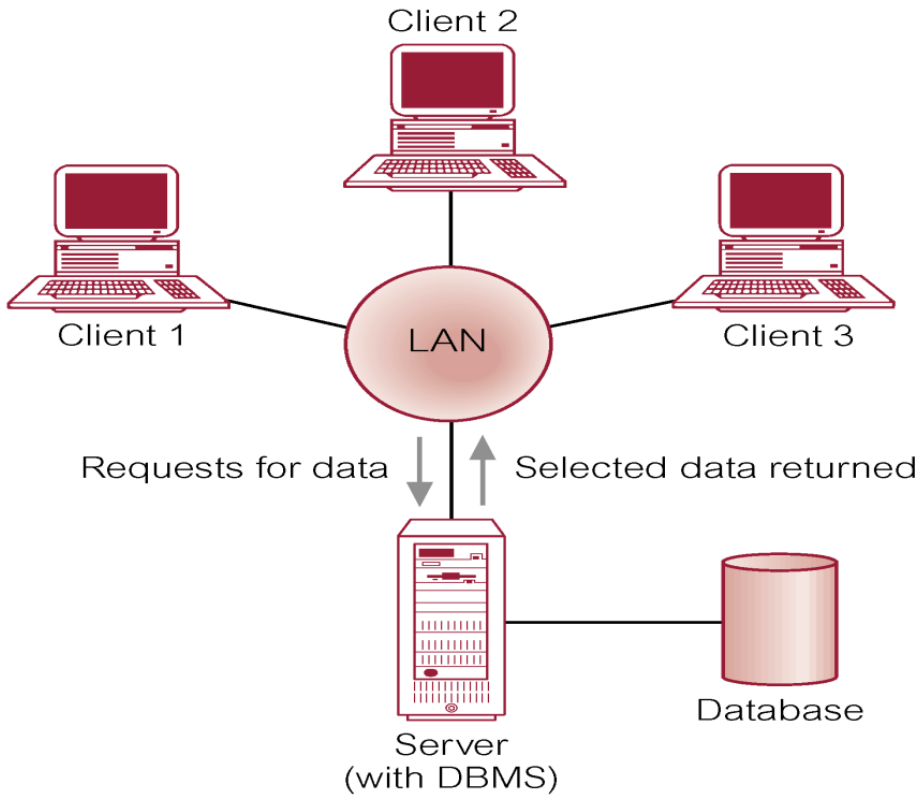
Interchangeable terms in types of architecture

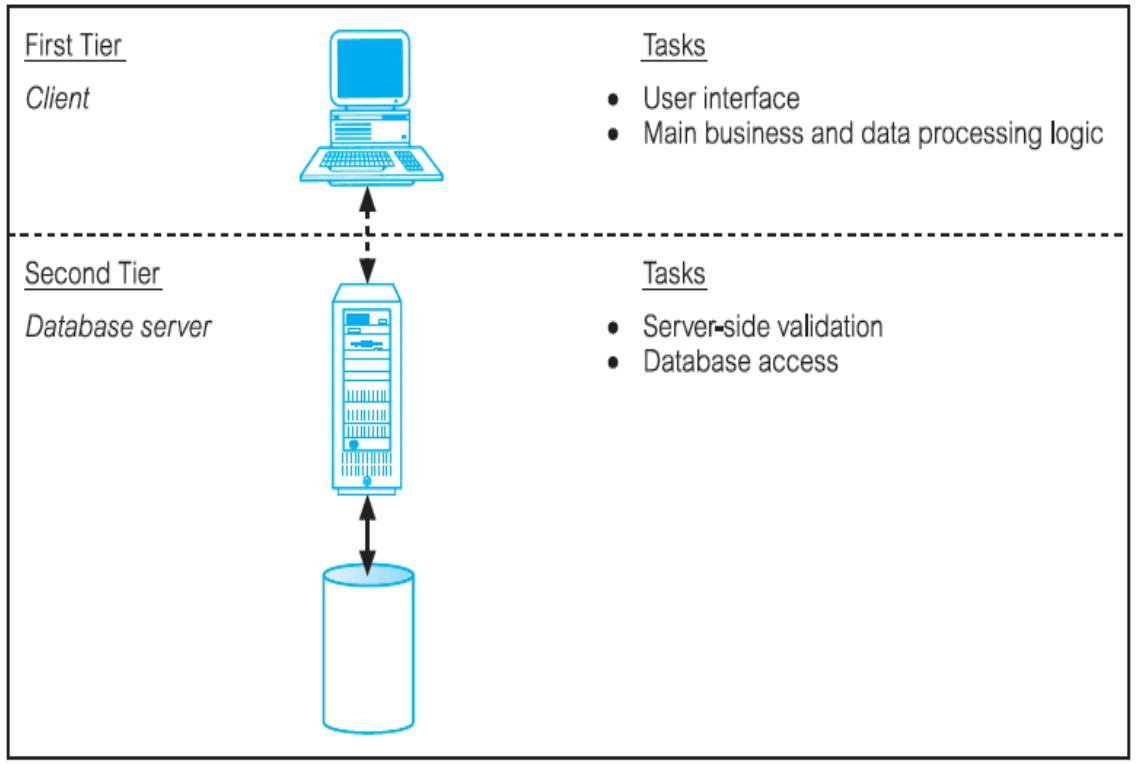
Single-Tier

- *Client, server and database all reside on the same machine.*
- Example: Installing database on your system and practicing SQL queries.
- Rarely used in production.



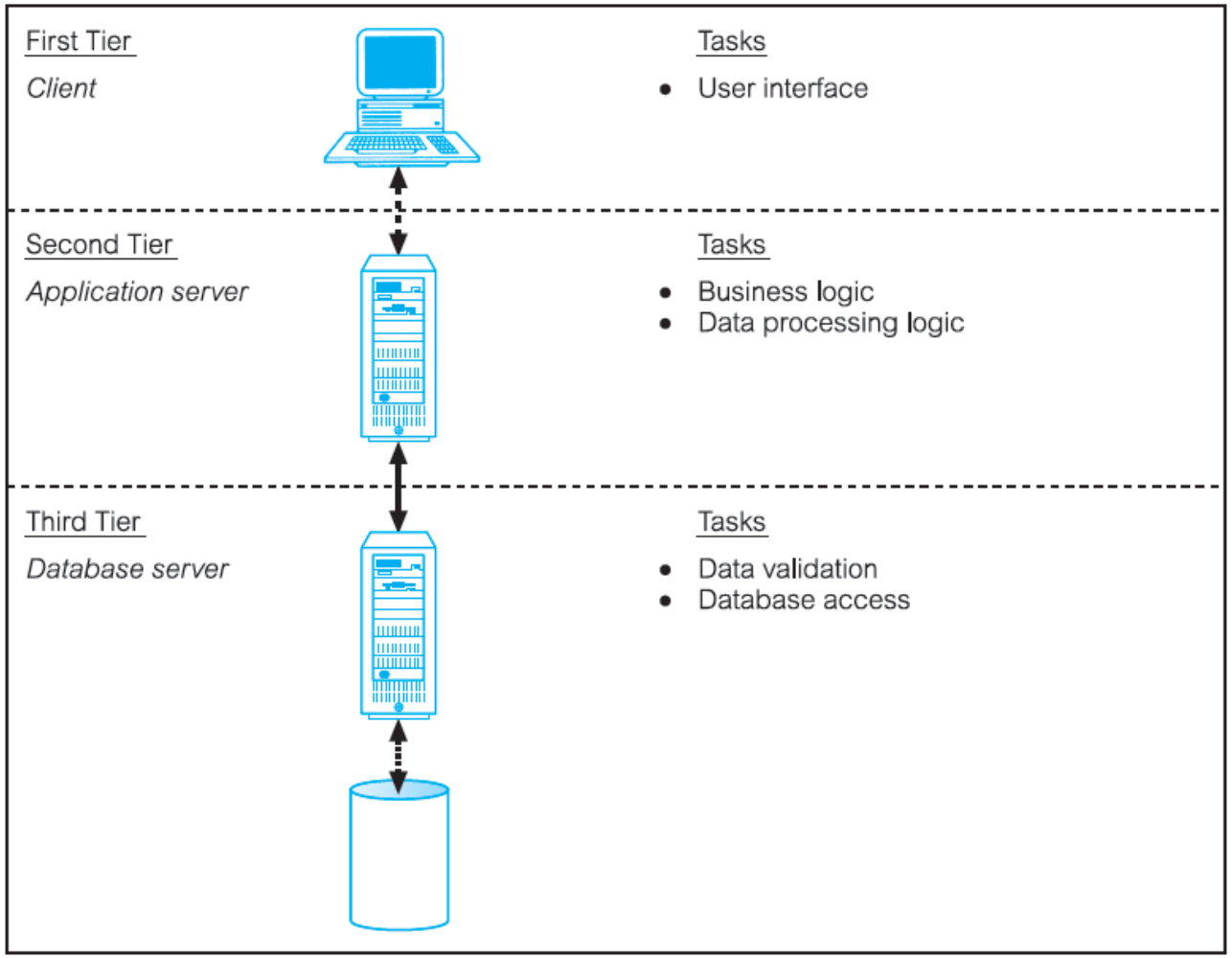
Two-Tier Client-Server Architecture





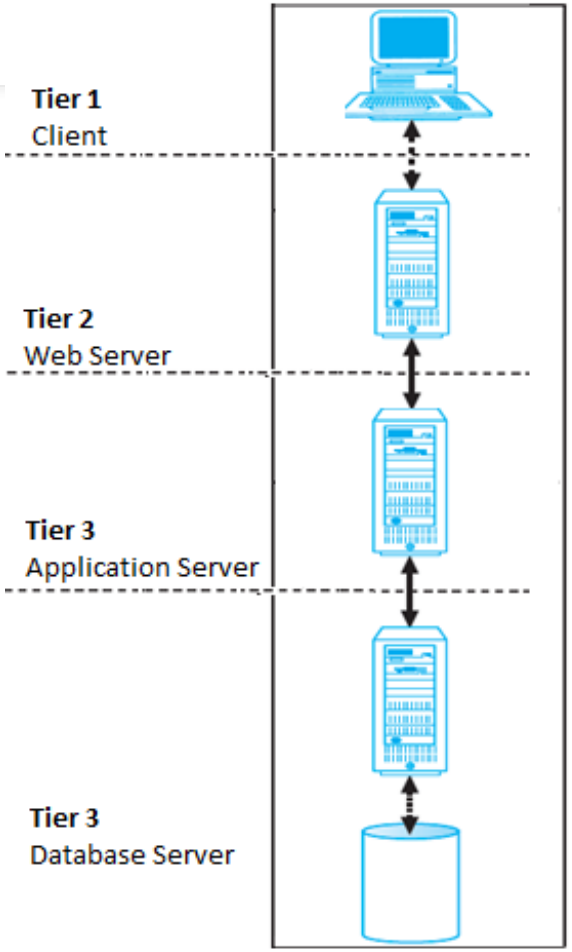
Three-Tier

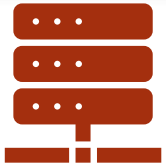
Client-Server Architecture



N-Tier or Multi-tier Architecture

- N-tier architecture extends the three-tier model by adding more layers or tiers, each handling specific tasks
- The "n" represents the number of layers, which can be more than three depending on the complexity of the application.





n-tier DBMS architecture

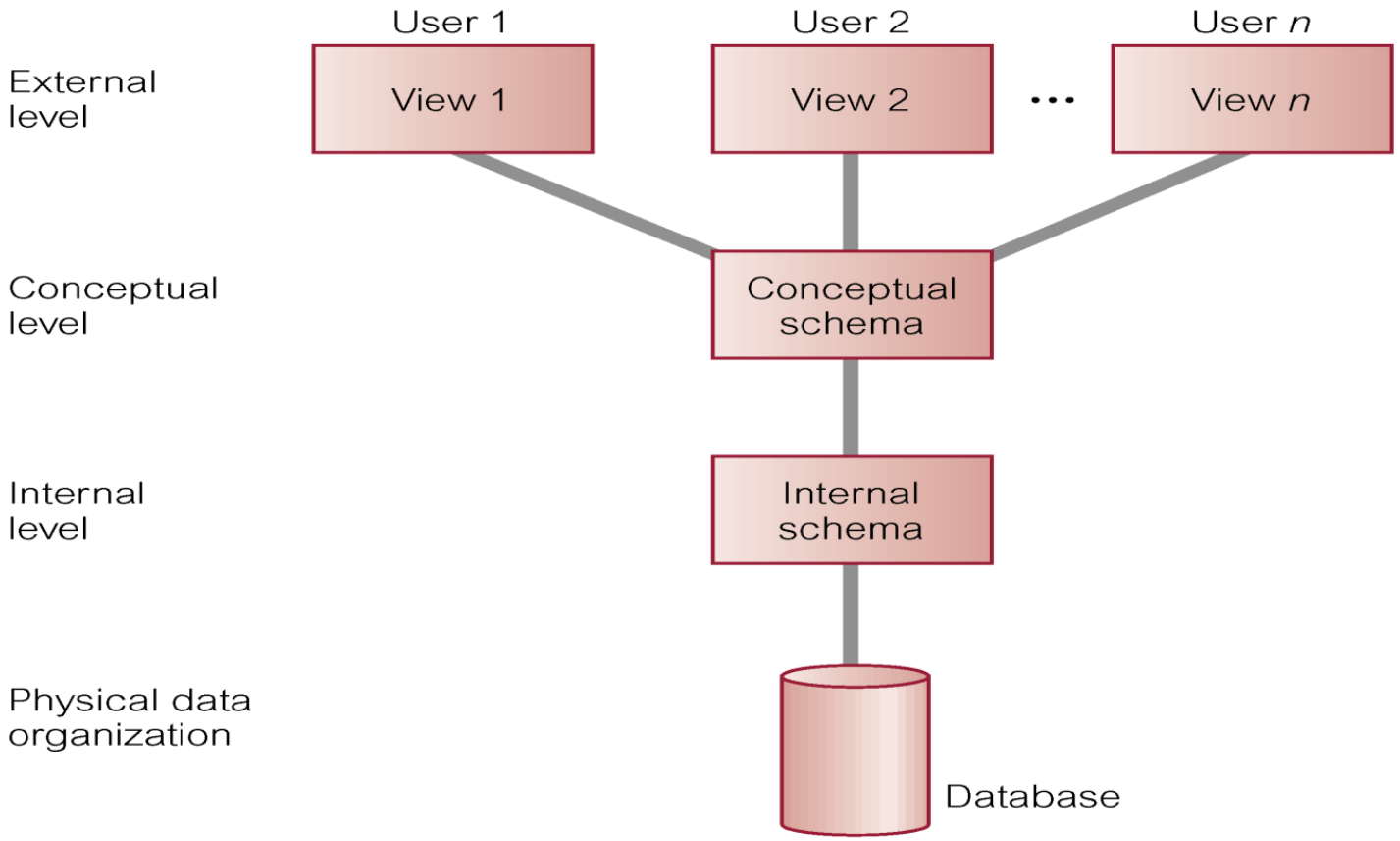
Client with GUI functionality, application server with applications, database server with DBMS and database, and web server for web-based access

Understanding by Example

- ***Two-Tier for Enterprise-Level Banking:*** Suitable for smaller, internal banking applications where the system doesn't need to support a massive number of users or complex transactions.
- ***Three-Tier for Online Banking*:** Ideal for online banking, providing a balanced approach to scalability, security, and maintainability, especially important for web-based or mobile banking.
- ***N-Tier for Large-Scale Applications:*** Best for very large and complex systems that require high scalability and modularity, with multiple layers to handle different aspects of the application, ensuring the system can efficiently cater to a growing number of users and services.

ANSI-SPARC Three-level Architecture (1975)

Basic framework for designing a database management system.



Objectives of Three-Level Architecture

All users should be able to access same data.

A user's view is immune to changes made in other views.

Users should not need to know physical database storage details.

DBA should be able to change database storage structures without affecting the users' views.

Internal structure of database should be unaffected by changes to physical aspects of storage.

DBA should be able to change conceptual structure of database without affecting all users.

ANSI-SPARC Three-level Architecture

External Level

- Users' view of the database.
- Describes that part of database that is relevant to a particular user.

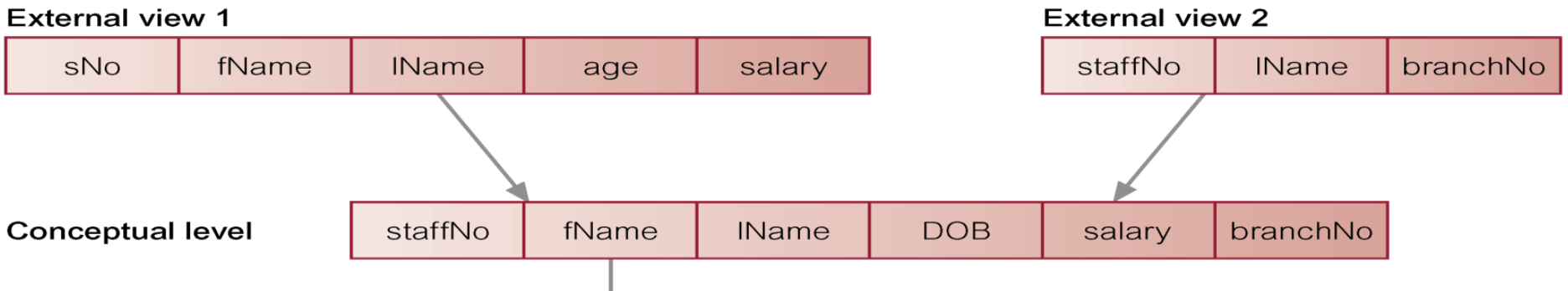
Conceptual Level

- Community view of the database.
- Describes what data is stored in database and relationships among the data.

Internal Level

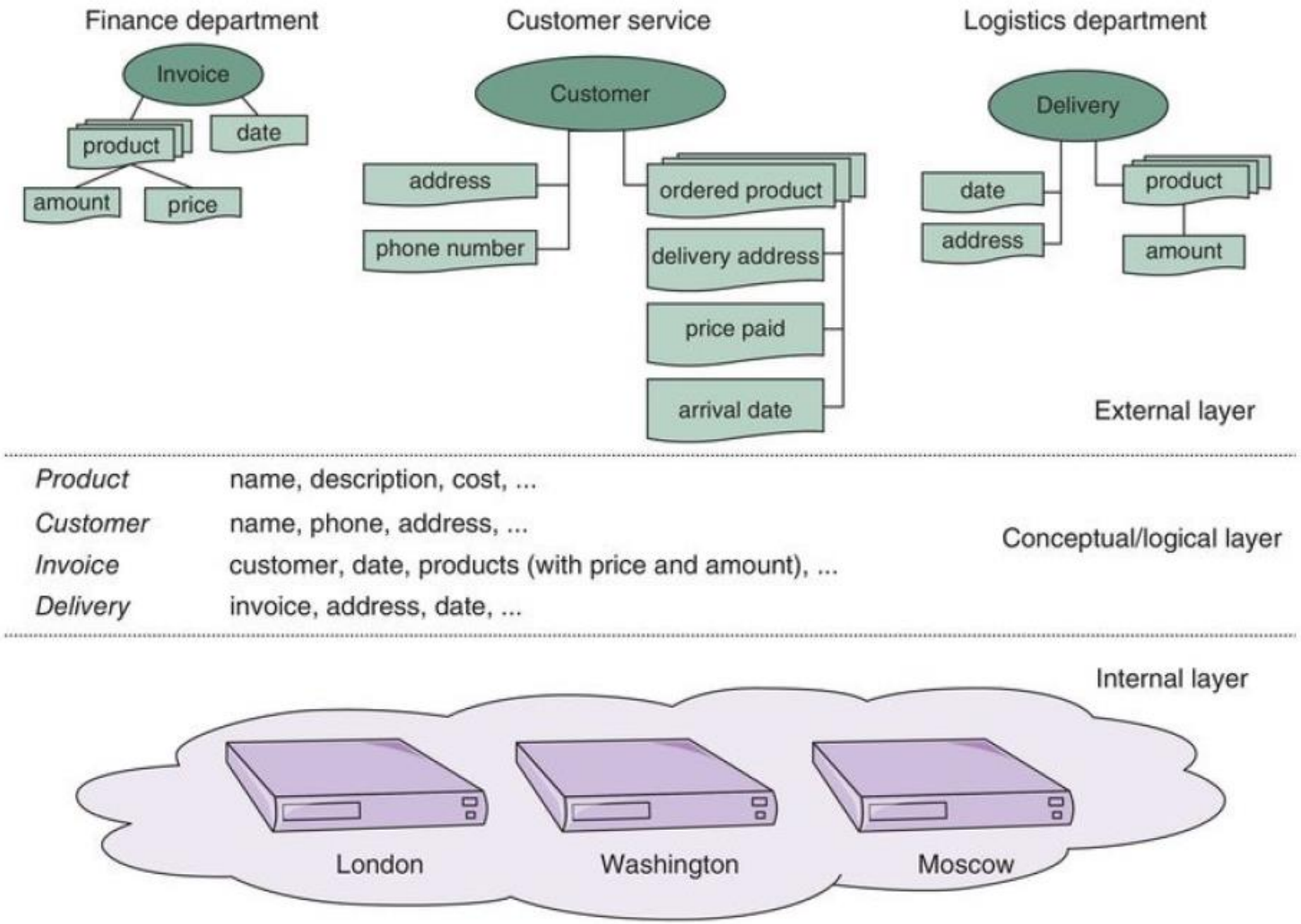
- Physical representation of the database on the computer.
- Describes how the data is stored in the database.

Differences between Three Levels of ANSI-SPARC Architecture



Internal level

```
struct STAFF {
        int staffNo;
        int branchNo;
        char fName [15];
        char lName [15];
        struct date dateOf Birth;
        float salary;
        struct STAFF *next;                    /* pointer to next Staff record */
};
index staffNo; index branchNo;                 /* define indexes for staff */
```

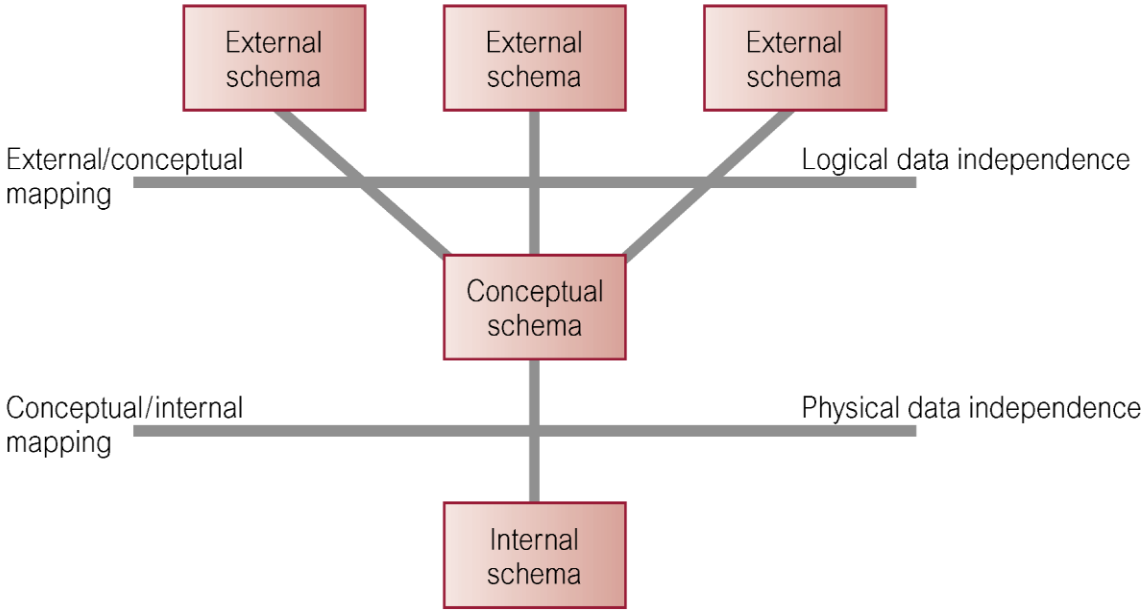

Finance department
Invoice
product
date
amount
price
Customer service
Customer
address
phone number
ordered product
delivery address
price paid
arrival date
Logistics department
Delivery
date
address
product
amount
External layer
Product name, description, cost, ...
Customer name, phone, address, ...
Invoice customer, date, products (with price and amount), ...
Delivery invoice, address, date, ...
Conceptual/logical layer
Internal layer
London
Washington
Moscow

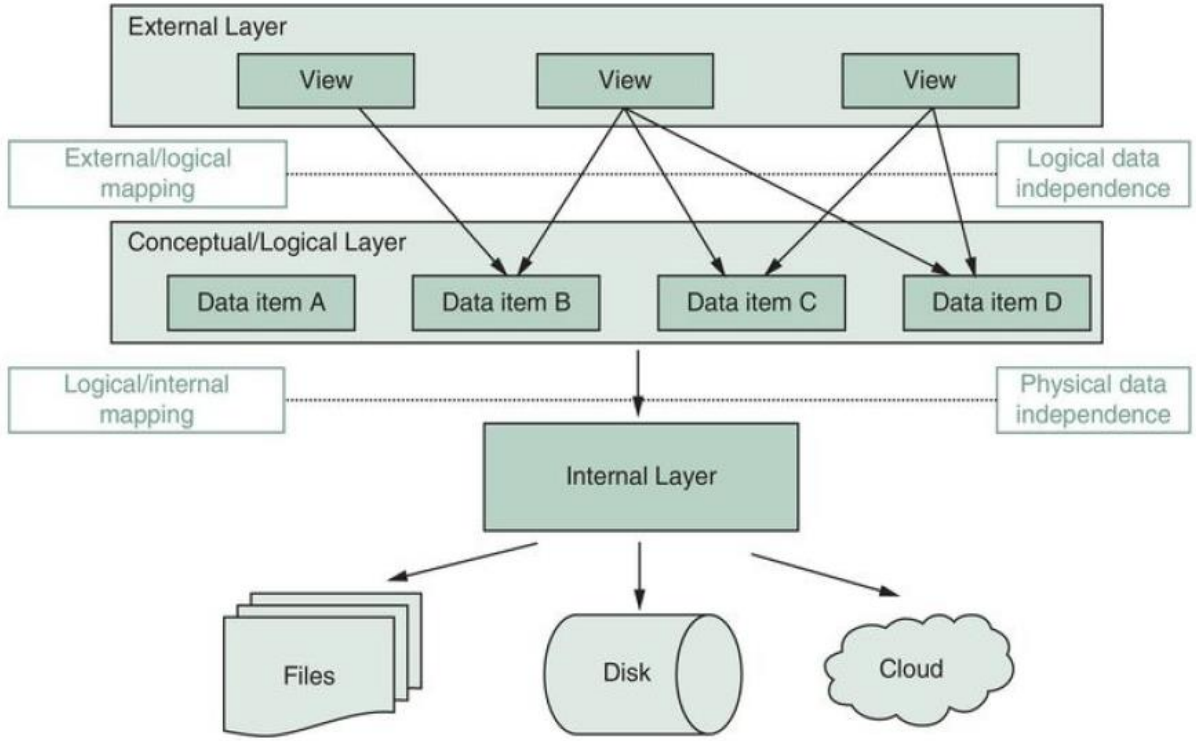
Data Independence

- *A major objective for the three-level architecture is to provide data independence, which means that upper levels are unaffected by changes to lower levels.*

- There are two kinds of data independence: **logical and physical.**

Data Independence and 3-level / 3-Layer Architecture

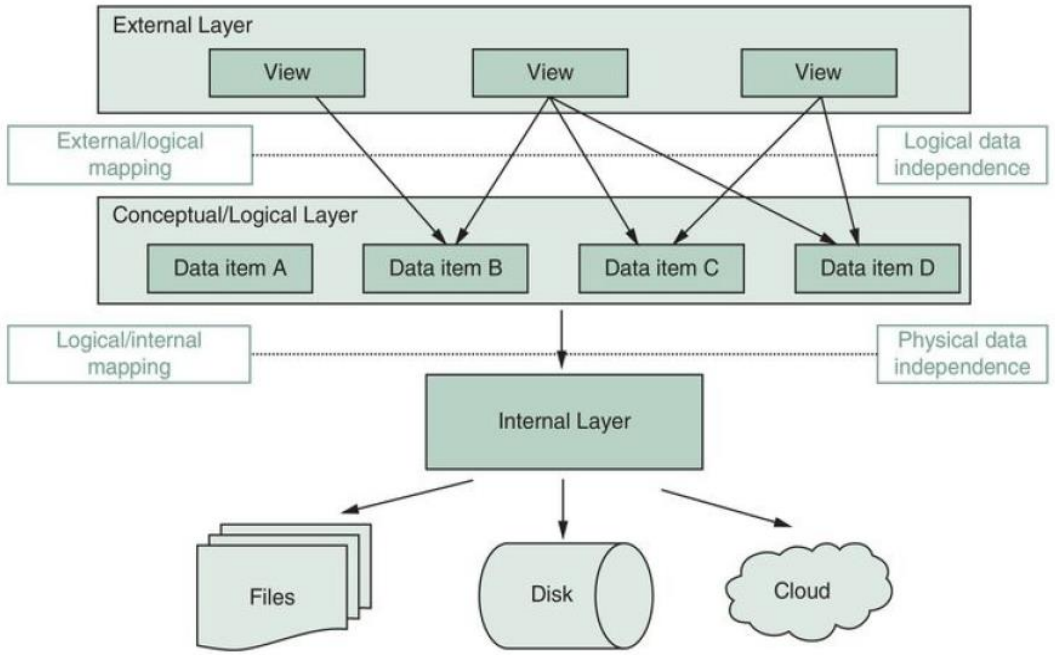




Data Independence

Logical Data Independence

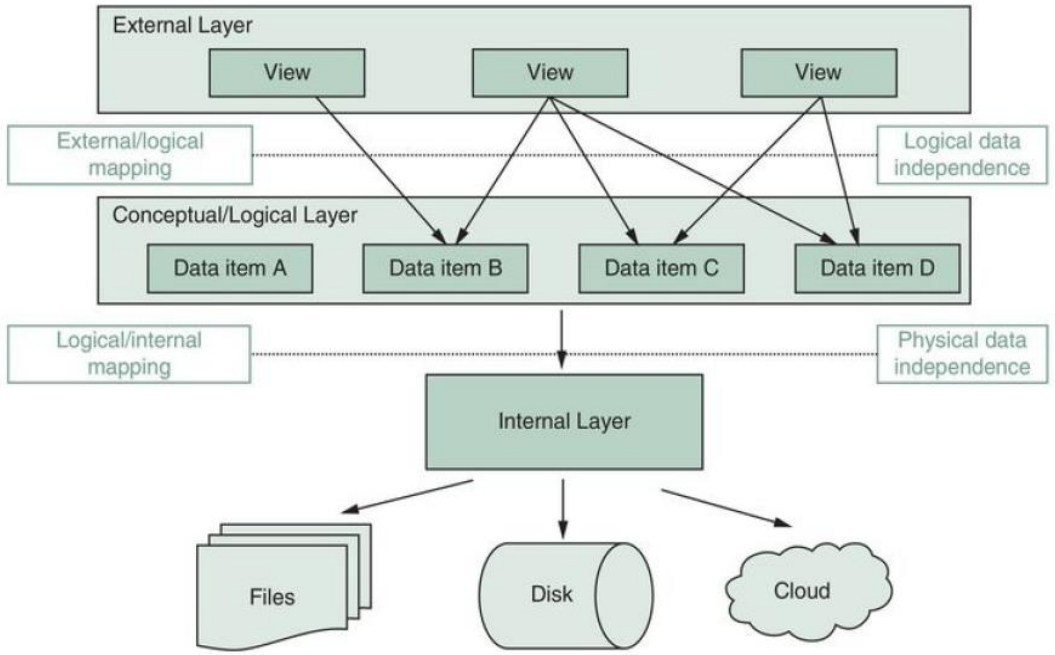
- Refers to immunity of external schemas to changes in conceptual schema.
- Conceptual schema changes (e.g. addition/removal of entities) should not require changes to external schema or rewrites of application programs.



Data Independence

Physical Data Independence

- Refers to immunity of conceptual schema to changes in the internal schema.
- Internal schema changes (e.g. using different file organizations, storage structures/devices) should not require change to conceptual or external schemas.



Database Languages

Data Definition Language (DDL)

- Allows the DBA or user to describe and name entities, attributes, and relationships required for the application
- plus, any associated integrity and security constraints.

Database Languages

Data Manipulation Language (DML)

Provides basic data manipulation operations on data held in the database.

- **Procedural DML**
  - allows user to tell system exactly how to manipulate data.
  - PL/SQL (Procedural Language for SQL)
- **Non-Procedural DML**
  - allows user to state what data is needed rather than how it is to be retrieved.
  - SQL is a Non-procedural DML

Data Model

Integrated collection of concepts for describing data, relationships between data, and constraints on the data in an organization.

Data Model comprises:

- A structural part – set of rules according to which DBs are constructed.
- A manipulative part – defining types of operations allowed on this data
- Possibly a set of integrity rules – ensures data is accurate

Data Model

Purpose

- To represent data in an understandable way.

Categories of data models include:

- Object-based
- Record-based

Conceptual and External

- Physical → Internal

Data Models

Object-based Data Models

- Entity-Relationship
- Semantic
- Functional
- Object-Oriented

Record-based Data Models

- Relational Data Model
- Network Data Model
- Hierarchical Data Model

Physical Data Models

Data Models

Object-based Data Models

- **Entity-Relationship -** Uses diagrams to show data entities and their relationships.
- **Semantic -** Structures data to reflect real-world meanings and logical relationships.
- **Functional** - Defines data processing through functions and operations.
- **Object-Oriented -** Represents data as objects with attributes and methods.

Data Models

Record-based Data Models

- **Relational Data Model** - proposed by E.F. Codd to model data in the form of relations or tables with rows and columns.
- **Network Data Model -** multiple member records or files can be linked to multiple owner files and vice versa. It uses graph structure.
- **Hierarchical Data Model -** uses a one-to-many relationship for data elements. Hierarchical database models use a tree structure that links a number of disparate elements to one "owner," or "parent," primary record.

Data Models

- **Physical Data Models** - defines all of the logical database components and services that are required to build a database or can be the layout of an existing database

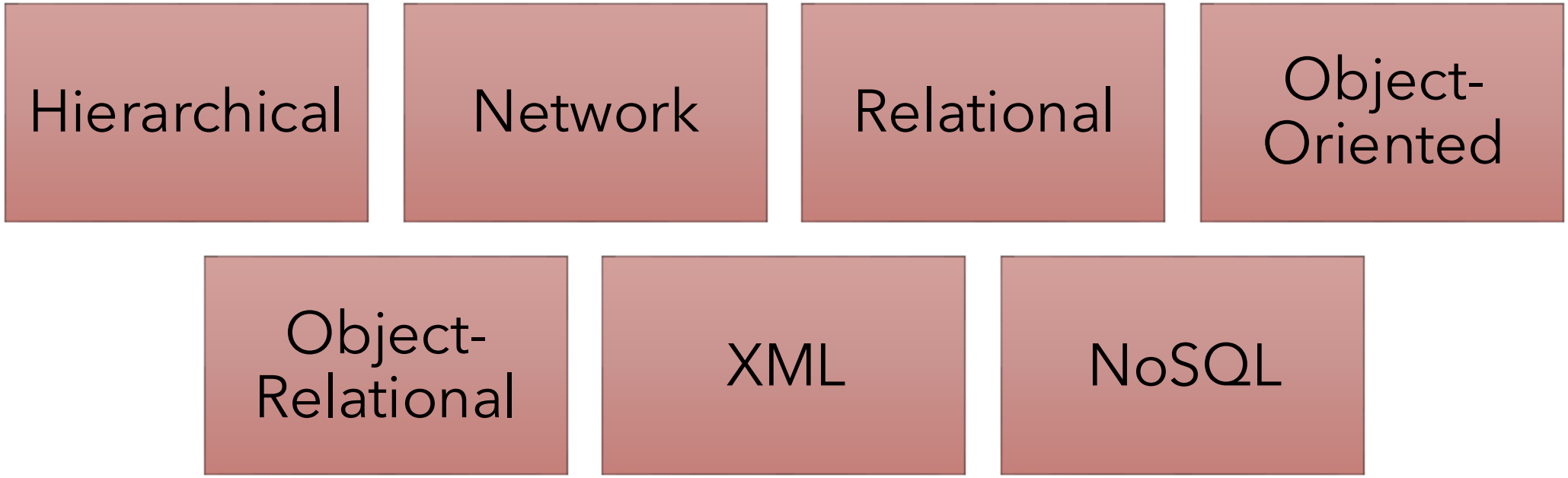
Conceptual Modeling

- *Conceptual schema* is the core of a system supporting all user views.
- Should be complete and accurate representation of an organization's data requirements.
- Conceptual modelling is process of developing a model of information use that is independent of implementation details.

Categorization of DBMS

1. Categorization based on **data model**
2. Categorization based on **degree of simultaneous access**
3. Categorization based on **architecture**
4. Categorization based on **usage**

Reading Assignment



1. Categorization based on data model

1. Categorization based on data model

- **Hierarchical DBMSs**
  - Adopt a tree like data model
  - DML is procedural and record oriented
  - No query processor (logical and internal data model intertwined)
  - E.g., IMS (IBM)
- **Network DBMSs**
  - Use a network data model
  - CODASYL DBMSs
  - DML is procedural and record oriented
  - No query processor (logical and internal data model intertwined)
  - CA-IDMS (Computer Associates)

1. Categorization based on data model

- **Relational DBMSs**
  - Use the relational data model
  - Currently the most popular in industry
  - SQL (declarative and set oriented)
  - Query processor
  - Strict separation between the logical and internal data model
  - E.g., MySQL (open source, Oracle), Oracle DBMS (Oracle), DB2 (IBM), Microsoft SQL (Microsoft)

1. Categorization based on data model

- **Object-Oriented DBMSs (OODBMS)**
  - Based upon the OO data model
  - No impedance mismatch in combination with OO host language
  - E.g., db4o (open source, Versant), Caché (Intersystems) GemStone/S (GemTalk Systems)
  - Only successful in niche markets, due to their complexity

1. Categorization based on data model

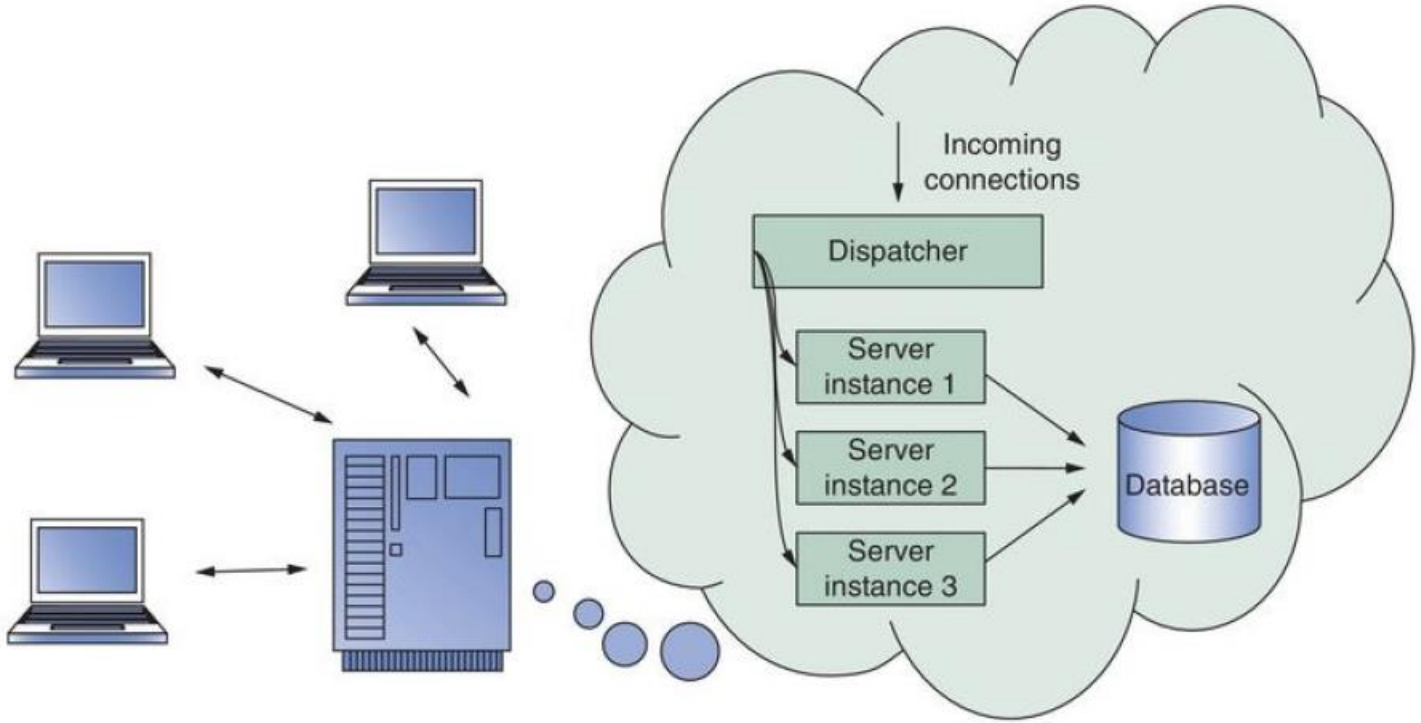
- **Object-Relational DBMSs (ORDBMSs)**
  - Also referred to as Extended Relational DBMSs (ERDBMSs)
  - Use a relational model extended with OO concepts
  - DML is SQL (declarative and set oriented)
  - E.g., Oracle DBMS (Oracle), DB2 (IBM), Microsoft SQL (Microsoft)

1. Categorization based on data model

- **XML DBMSs**
  - Use the XML data model to store data
  - Native XML DBMSs (e.g., BaseX, eXist) map the tree structure of an XML document to a physical storage structure
  - XML-enabled DBMSs (e.g., Oracle, IBM DB2) are existing DBMSs that are extended with facilities to store XML data

1. Categorization based on data model

- **NoSQL DBMSs**
  - Targeted at storing big and unstructured data
  - Can be classified into key-value stores, column-oriented databases and graph databases
  - Focus on scalability and the ability to cope with irregular or highly volatile data structures
  - E.g., Apache Hadoop, MongoDB, Neo4j



- Single user versus Multi-user systems

2. Categorization based upon degree of simultaneous access

3. Categorization based on Architecture

Centralized DBMS architecture

Data is maintained on a centralized server

Queries will have to be processed by this single host

Client server DBMS architecture (Two Tier/Three Tier architecture)

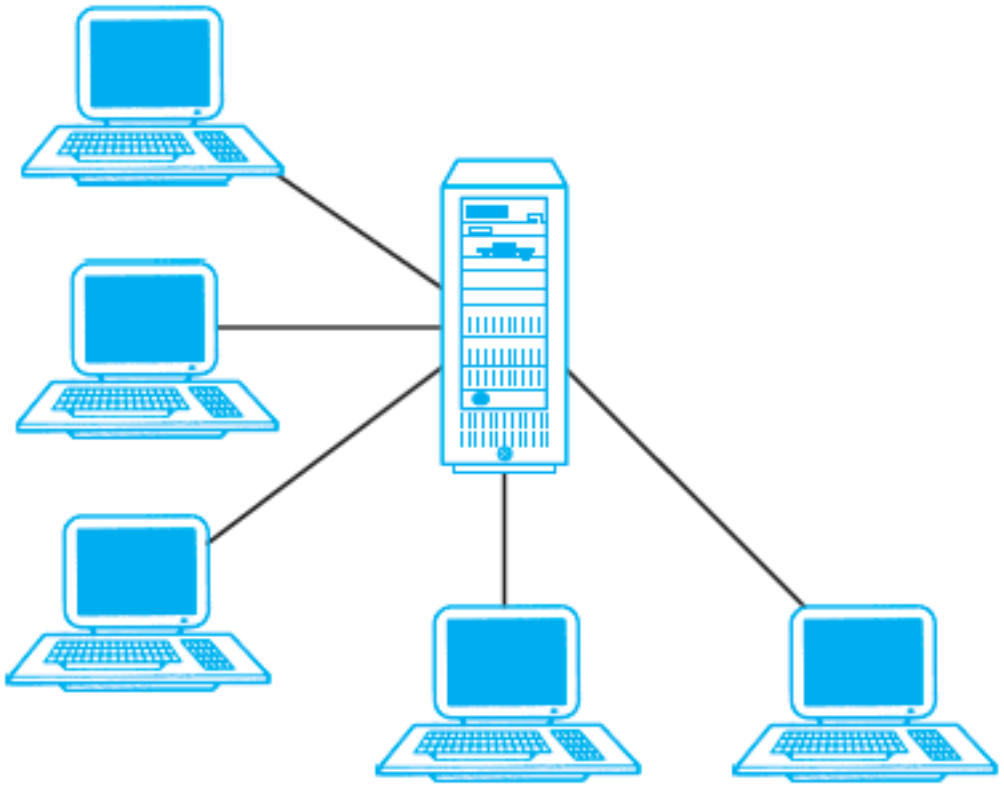
Active clients request services from passive servers

n-tier DBMS architecture

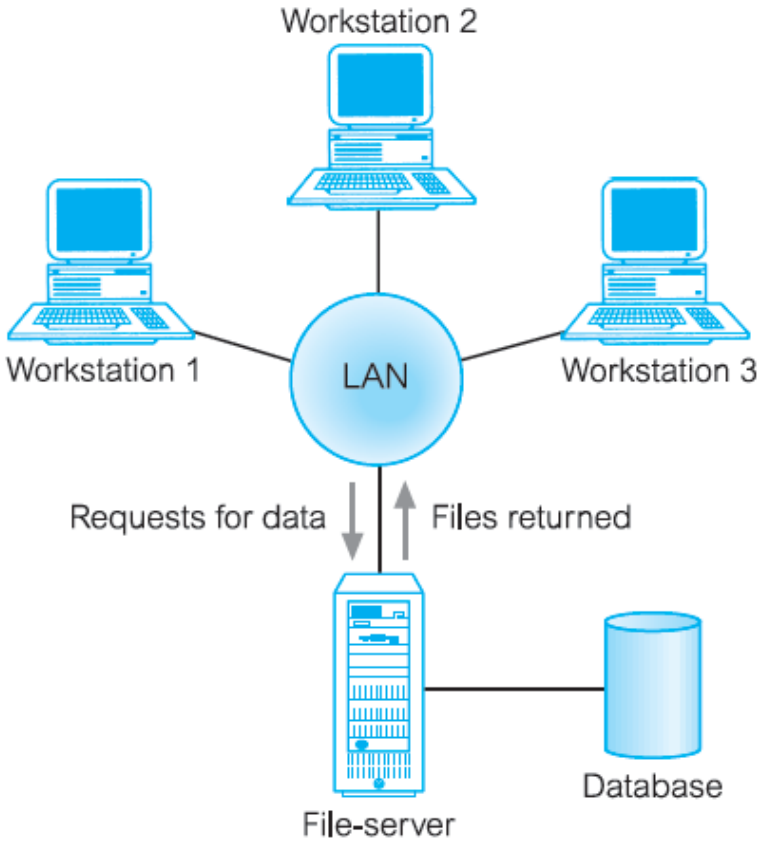
Client with GUI functionality, application server with applications, database server with DBMS and database, and web server for web-based access

3. Categorization based on architecture

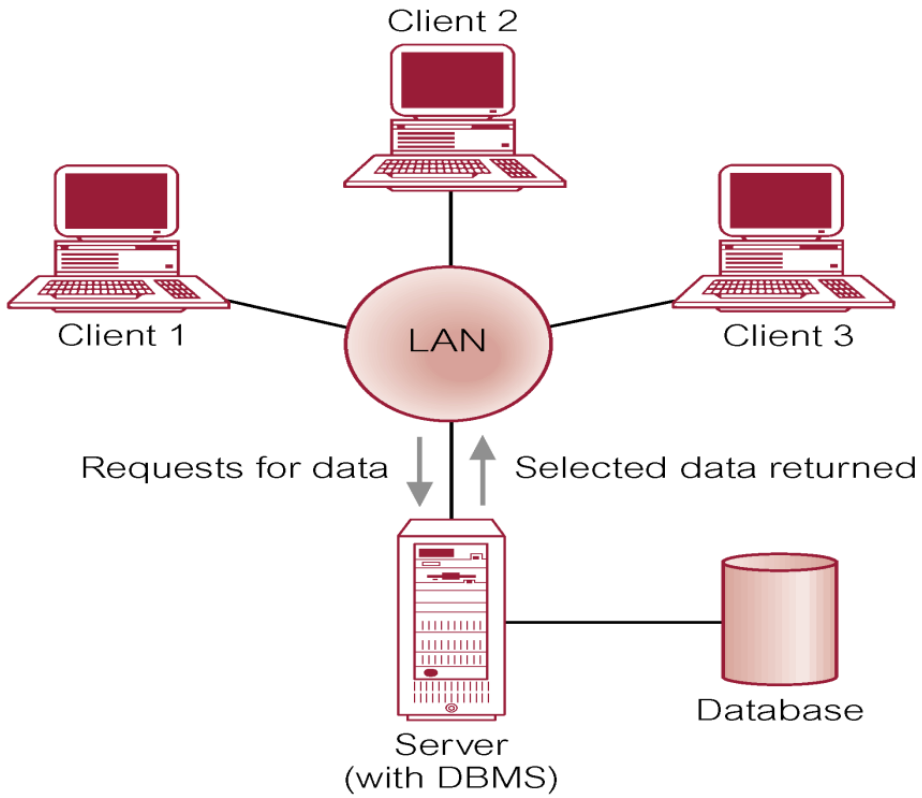
Teleprocessing

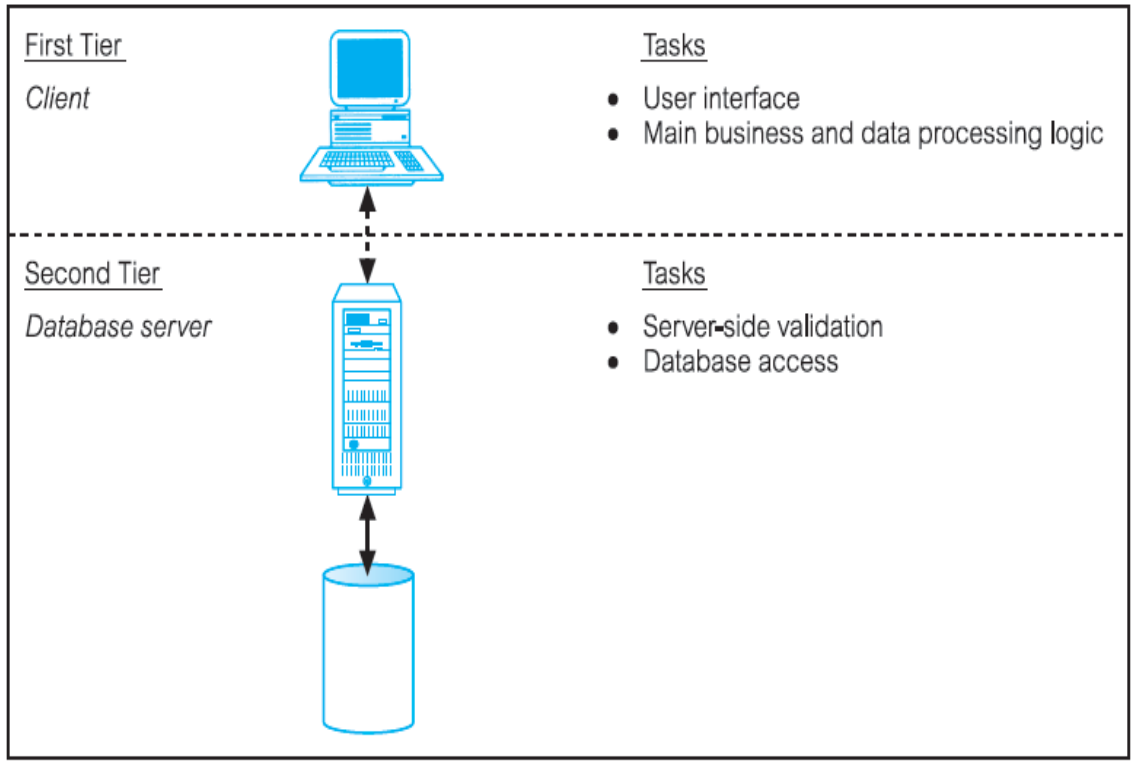


File Server Architecture

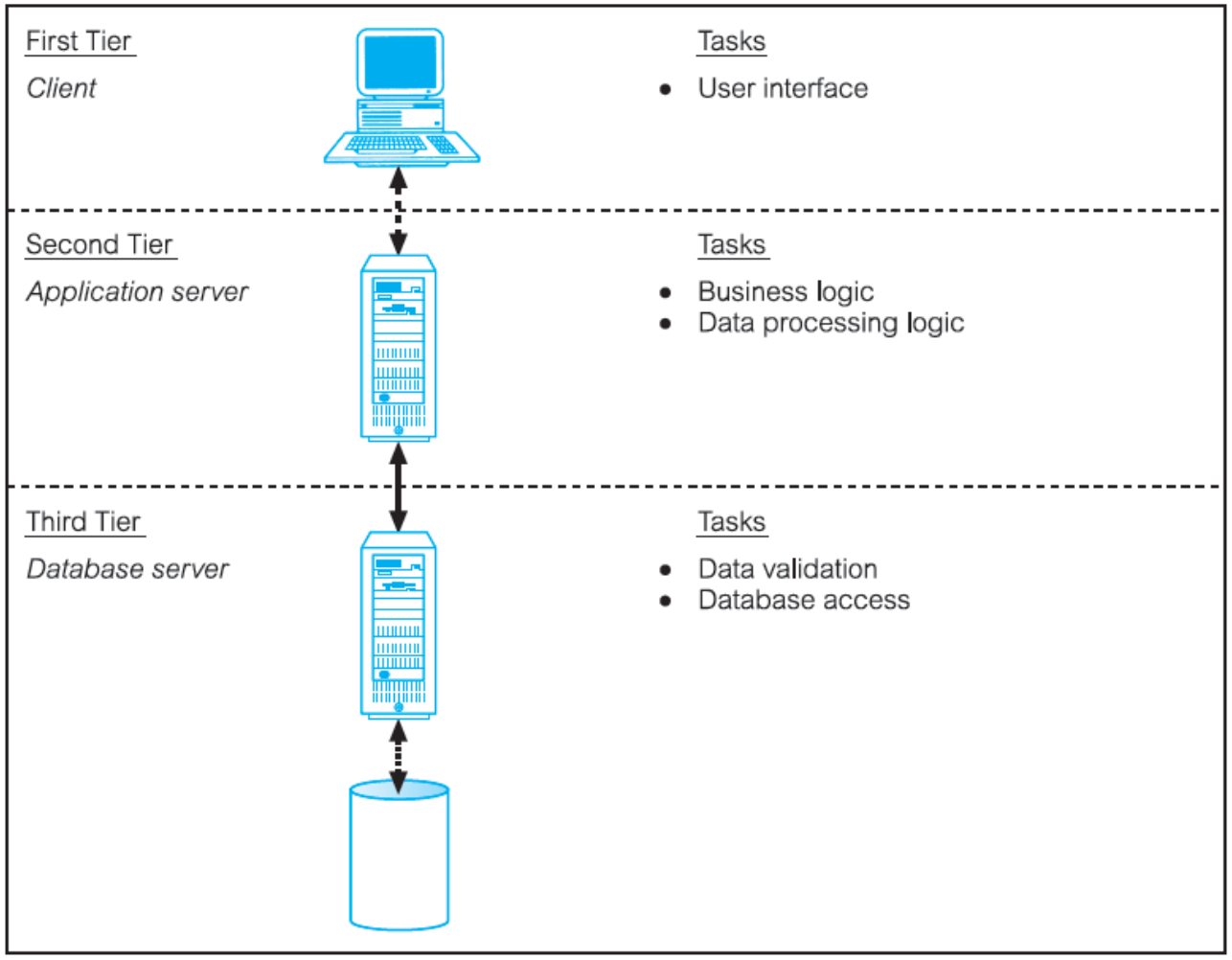


Two-Tier Client-Server Architecture

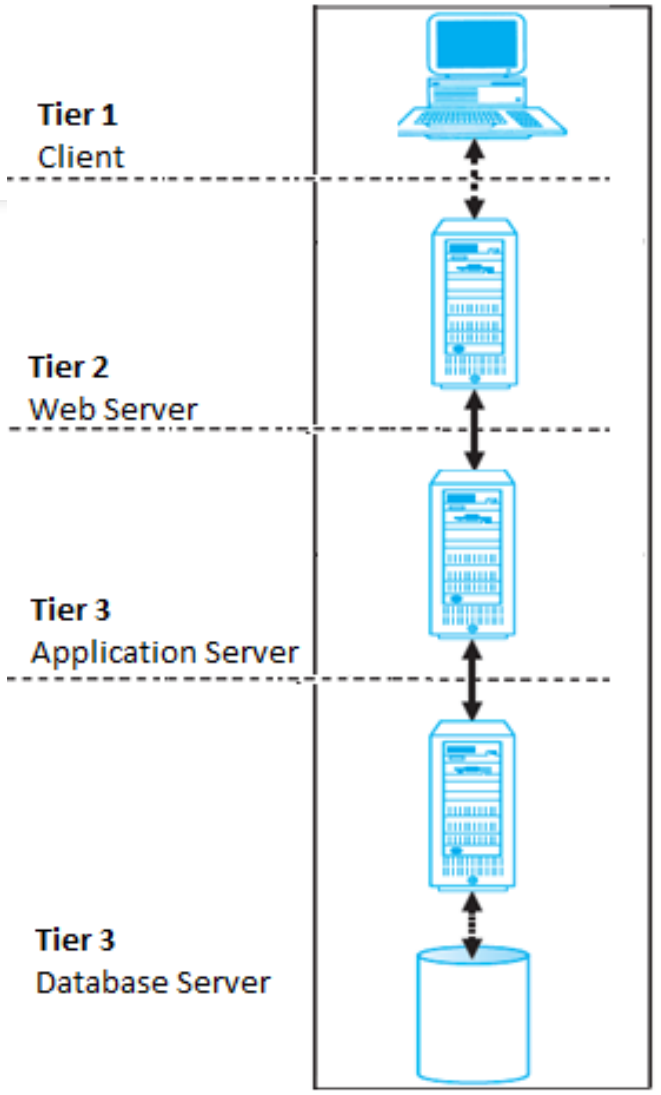




Three-Tier Client-Server Architecture



N-Tier Architectures



3. Categorization based on architecture

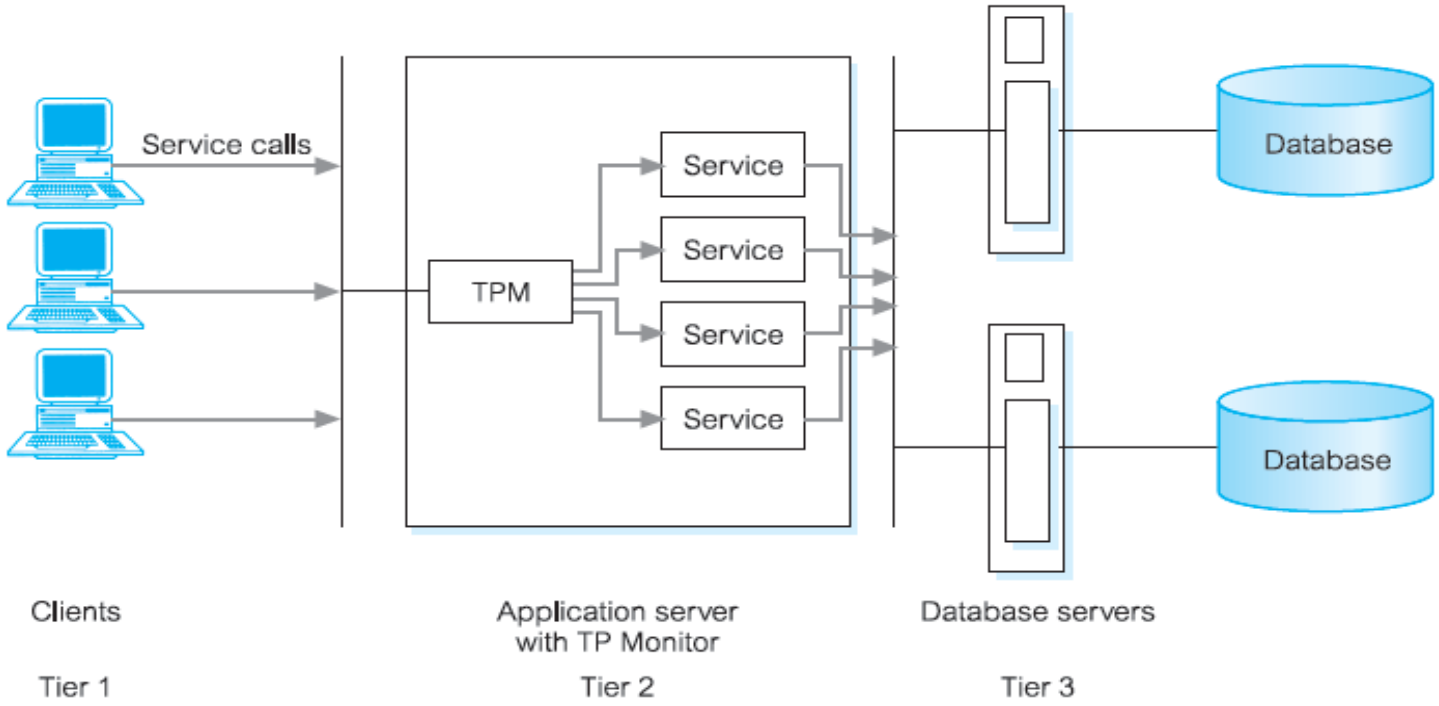
- **Advantages of n-tier architecture:**
  - Wider access to existing databases
  - Increased performance
  - Possible reduction in hardware costs
  - Reduction in communication costs
  - Increased consistency.

Transaction Processing Monitors

Improving the n-tier architecture with an additional component - TPM

Transaction Processing Monitors

- A program that controls data transfer between clients and servers in order to provide a consistent environment, particularly for online transaction processing (OLTP).



Advantages of TP Monitor

- *Transaction routing* – directing transactions to specific DBMS
- *Managing distributed transactions* – accessing data from multiple DBMS
- *Load balancing* – Balance load across multiple DBMS on one or more computers
- *Funnelling* – Multiple users logged on but require continuous requests so TPM can establish connections with the user and funnel request through it.
- *Increased reliability* – acts a transaction manager to maintain consistency. If DBMS fails, TPM can resubmit the transaction or hold until DBMS is available again.

3. Categorization based on architecture

- **Cloud DBMS architecture**
  - DBMS and database are hosted by a third-party cloud provider
  - E.g. Google Cloud SQL, Microsoft Azure Cosmos DB, Amazon DynamoDB etc
- **Federated DBMS**
  - unified interface to multiple, heterogeneous data sources, allowing users to interact with and query data from various databases as if they were a single, integrated system.
  - hides the underlying storage details to facilitate data access

3. Categorization based on architecture

- **In-memory DBMS**
  - stores all data in internal memory instead of slower external storage (e.g., disk)
  - often used for real-time purposes
  - E.g., HANA (SAP)

4. Categorization based on Usage

4. Categorization based on usage

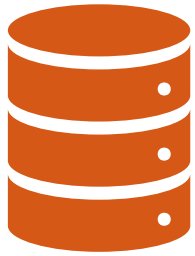
On-line transaction processing (OLTP)

- Focus on managing operational or transactional data
- Database server must be able to process lots of simple transactions per unit of time
- DBMS must have good support for processing a high volume of short, simple queries

On-line analytical processing (OLAP)

- Focus on using historic data for tactical or strategical decision making
- A limited number of users formulate complex queries
- DBMS should support efficient processing of complex queries which often come in smaller volumes

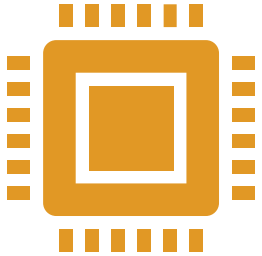
4. Categorization based on usage



Distributed database

Distributed DBMS

Distributed processing



Data warehousing

Subject-oriented

Integrated

Time-variant

Nonvolatile

4. Categorization based on usage

- **Big Data & Analytics**
  - NoSQL databases
  - focus on more flexible, or even schema-less, database structures
  - store unstructured information such as emails, text documents, Twitter tweets, Facebook posts, etc.
- **Multimedia**
  - Multimedia DBMSs provide storage of multimedia data such as text, images, audio, video, 3D games, etc.
  - should also provide content-based query facilities

4. Categorization based on usage

- **Spatial applications**
  - Spatial DBMSs support storage and querying of spatial data (both 2D and 3D)
  - Geographical Information Systems (GIS)
- **Sensor DBMS**
  - Sensor DBMSs manage sensor data such as biometric data from wearables, or telematics data

4. Categorization based on usage

- **Mobile**
  - Mobile DBMSs run on smartphones, tablets or other mobile devices.
  - should always be online, have a small footprint and be able to deal with limited processing power, storage and battery life
- **Open source**
  - code of open source DBMSs is publicly available and can be extended by anyone
  - E.g., MySQL (Oracle)