الجامعة السورية الخاصة
SYRIAN PRIVATE UNIVERSITY

مقرر بنيان البرمجيات | كلية الهندسة المعلوماتية | المحاضرة الثالثة

Layered Architecture

د. رياض سنبل

Quick Review

- Software Architecture.
- Design Goals.
- Subsystem.
- Service.
- Subsystem interface.
- Application programmer’s interface (API).
- Layers and Partitions.
- Layer relationships: “depends on” vs “calls”.
- Separation of Concerns (SoC).
- Coupling and Coherence of Subsystems

Software Architecture Style

- A software architecture style is a solution to a **recurring problem** that is well understood, in a particular context.
- Each pattern consists of a **context, a problem, and a solution.**
- Using patterns simplifies design and allows us to gain the benefits of **using a solution that is proven to solve** a particular design problem.
- **More than one software architecture pattern** can be used in a single software system. These patterns can be combined to solve problems.

In this Lecture

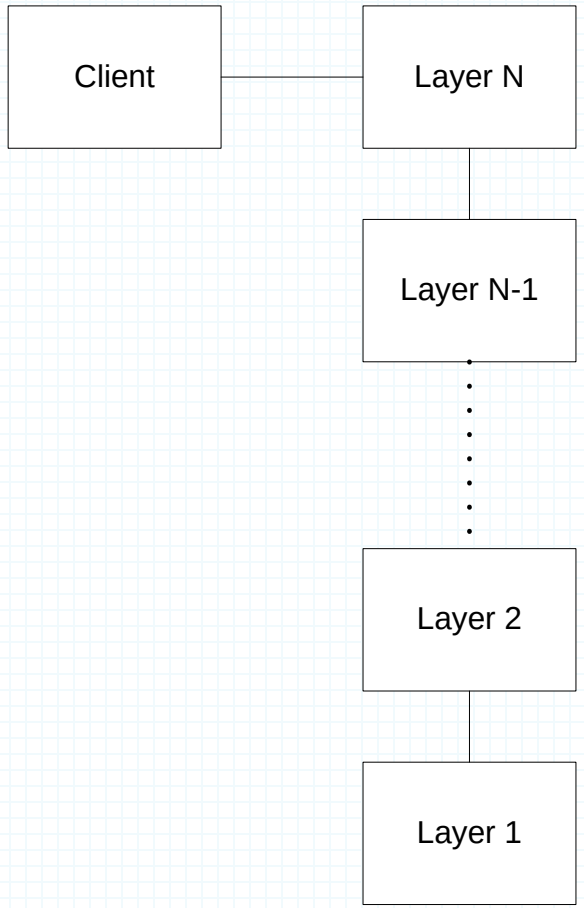
Layered Architecture

The Problem!

- You are designing a system that needs to handle **a mix of low-level and high-level issues**
- High-level operations rely on lower-level ones
- The system is **large**, and a methodical approach to organizing it is needed to keep it understandable

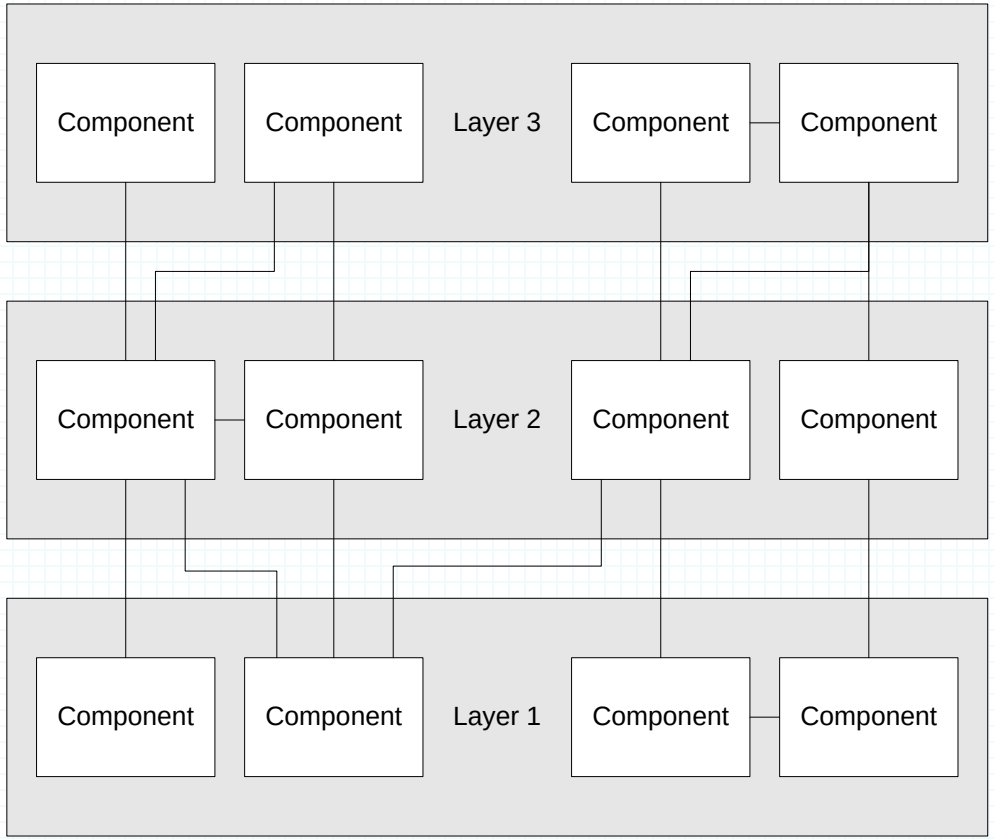
Solution

- Structure the system into an appropriate **number of layers**, and place them on top of each other
- Each layer represents a **level of abstraction**
- Classes are placed in layers based on their levels of abstraction
- Layer 1 is at the bottom and contains classes **closest** to the hardware/OS
- Layer N is at the top and contains classes that **interact directly** with the system's clients



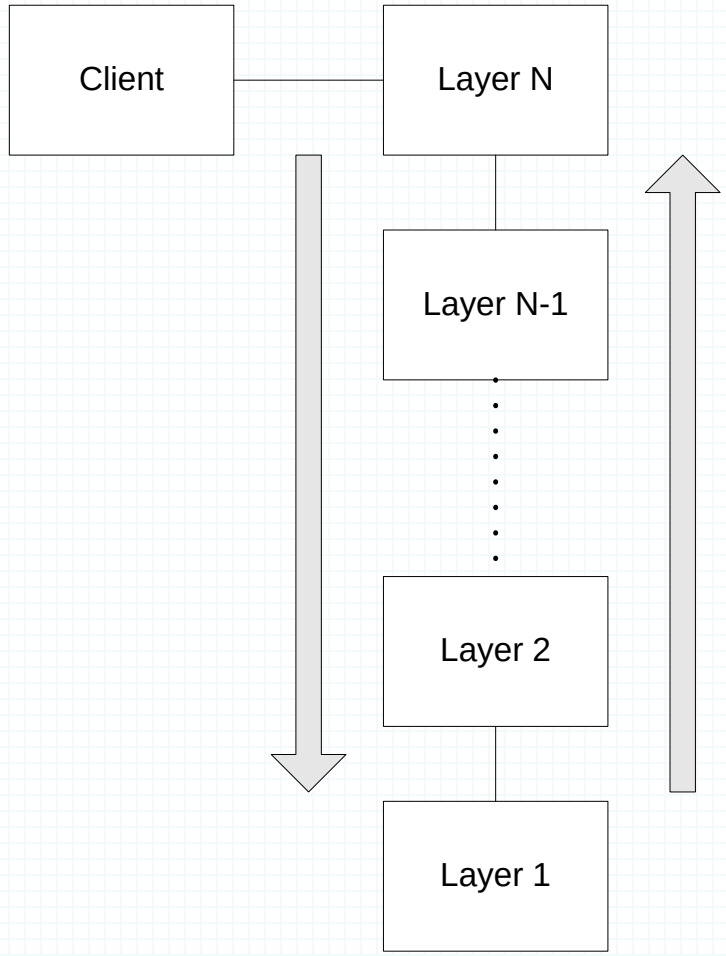
Solution

- Layer J uses the services of Layer J-1 and provides services to Layer J+1
- Most of Layer J's services are implemented by composing Layer J-1's services in meaningful ways
- Layer J **raises the level of abstraction** by one level
- Classes in Layer J may also use each other

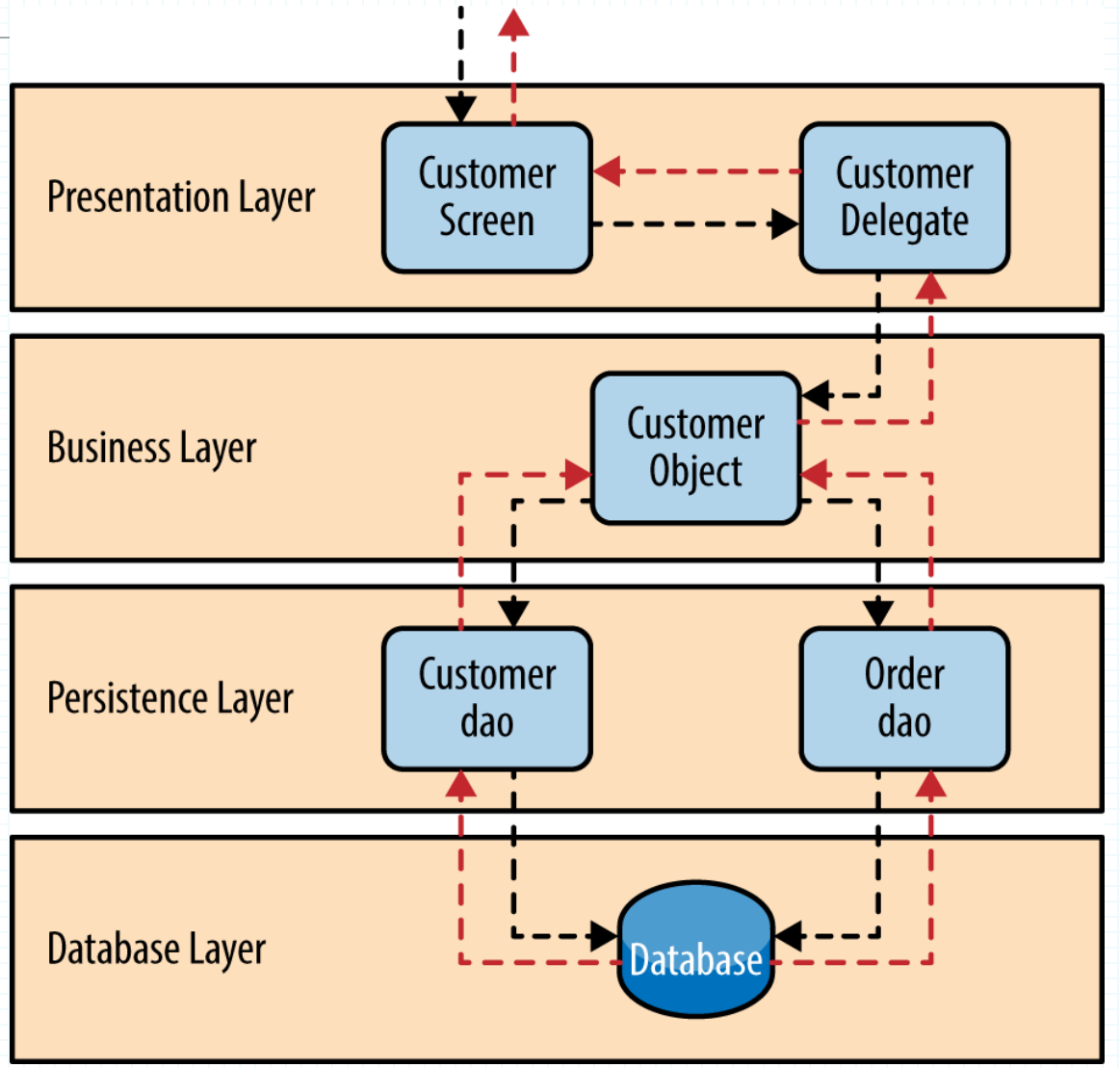


Dynamic Behavior (The most common case)

- Request at Layer N travels down through each successive layer, finally reaching Layer 1
- The top-level request results in a tree of requests that travel down through the layers (each request at Layer J is **translated into multiple requests** to Layer J-1)
- Results from below are combined to produce a higher-level result that is passed to the layer above



Example


Presentation Layer
Customer Screen
Customer Delegate
Business Layer
Customer Object
Persistence Layer
Customer dao
Order dao
Database Layer
Database

Implementation

- Determine the number of layers (i.e., abstraction levels)
- Name the layers and assign responsibilities to them
- Define the interface for each layer
- Error handling strategy
  - Errors received from the layer below should be mapped into higher-level errors that are appropriate for the layer above

Relaxed (Open) vs. strict (Closed) Layers

RELAXED (OPEN)

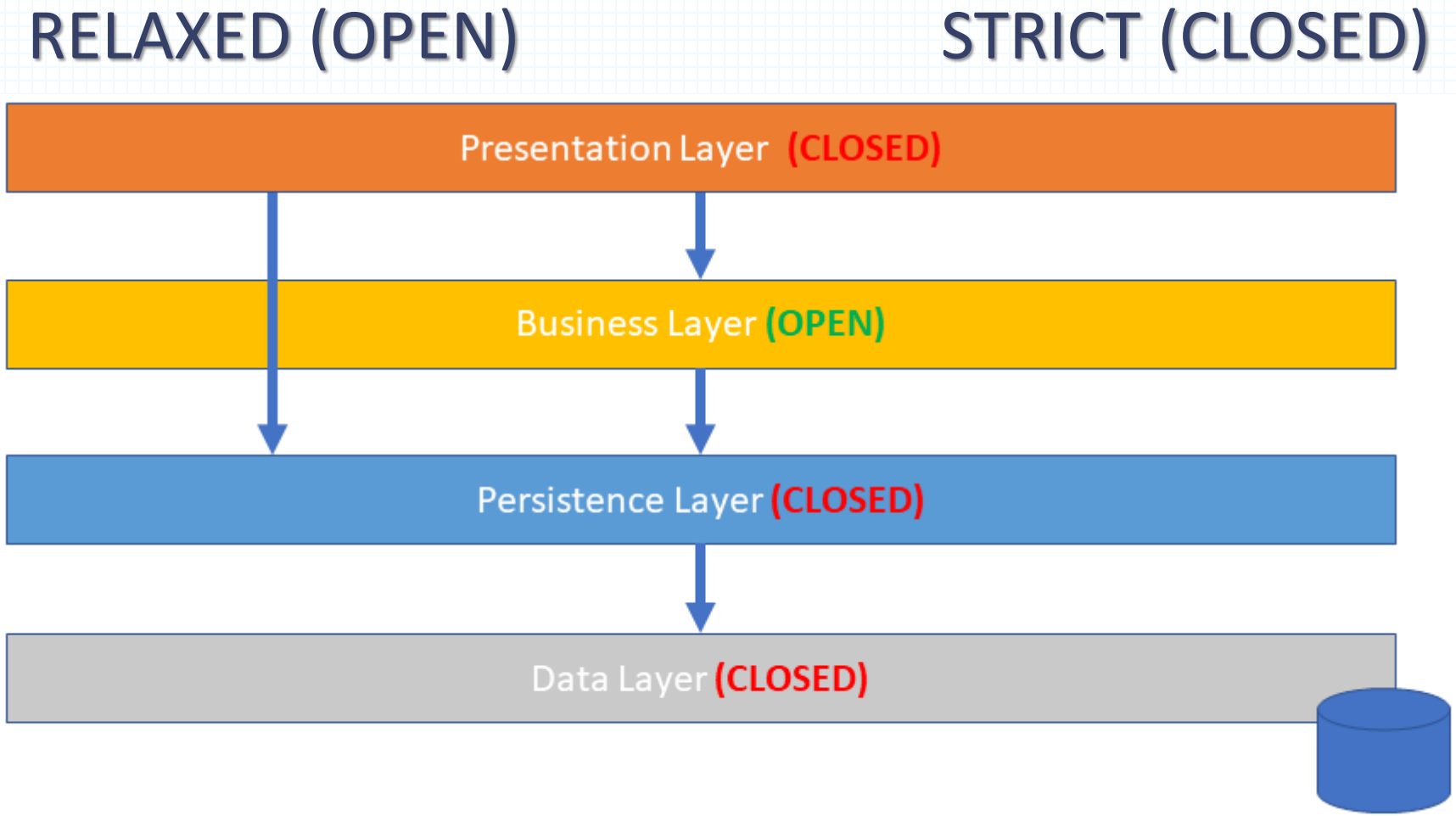
- Layer J can call directly into any layer below it, not just Layer J-1.
- **Pros:** More flexible and efficient than strict layers:
- **Cons:** Increases complexity, less understandable and maintainable than strict layers

STRICT (CLOSED)

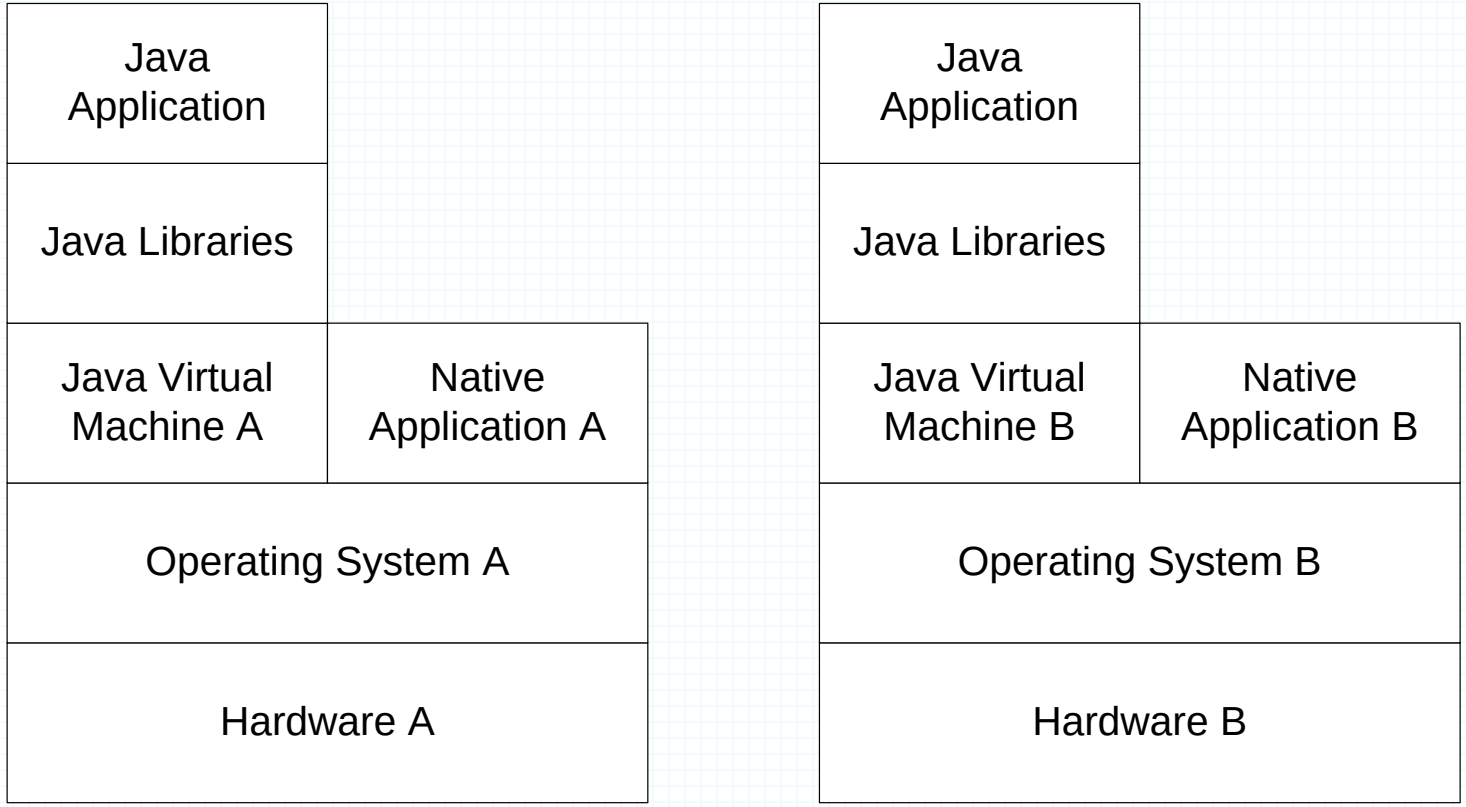
- Layer J can ONLY call Layer J-1
- **Pros:** makes code easier to change, write, and understand.
- **Cons:** unnecessary traffic can result

It is not necessary to make all of the layers open or closed.
You may selectively choose which layers, if any, are open.

Relaxed (Open) vs. strict (Closed) Layers



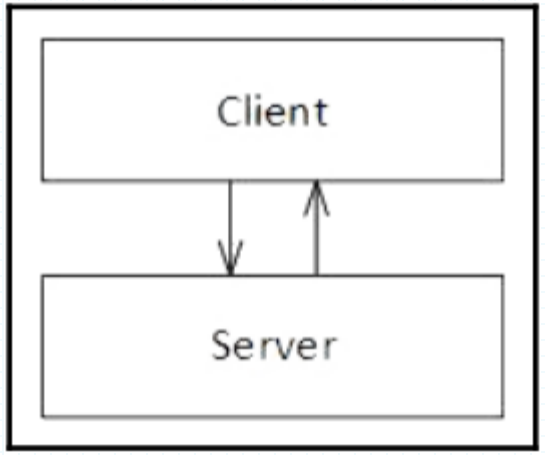
Known Uses: Virtual machines - Java



Client-server architecture (2-tier architecture)

- In a distributed application that uses a client-server architecture, also known as a two-tier architecture, clients and servers communicate with each other directly.
- A client requests some resource or calls some service provided by a server and the server responds to the requests of clients.
- There can be multiple clients connected to a single server.

contains the user interface code

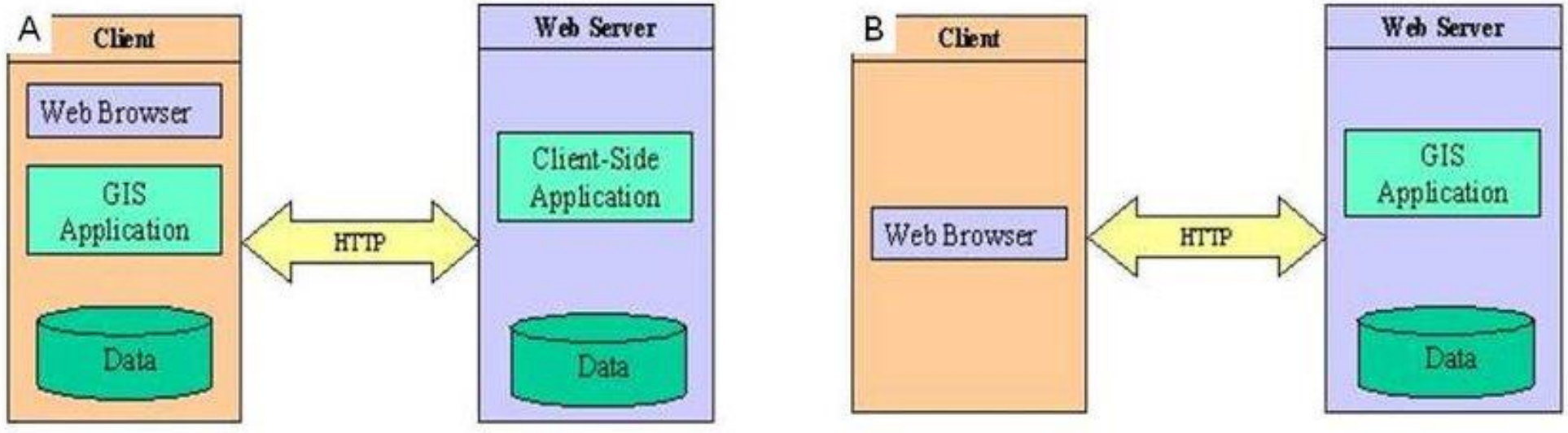


Contains the relational database management system (RDBMS)

The majority of application logic

Client-server architecture (2-tier architecture)

- When the client contains a significant portion of the logic and is handling a large share of the workload, it is known as a thick, or fat, client. When the server is doing that instead, the client is known as a thin client.

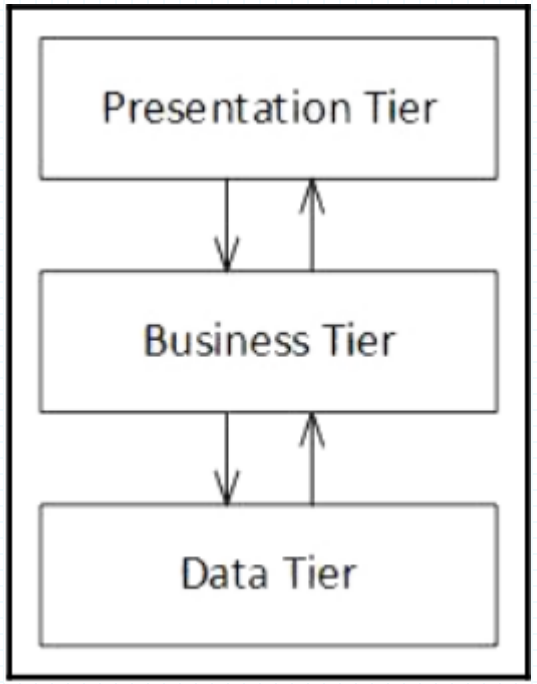


a) Thick client architecture

b) Thin client architecture

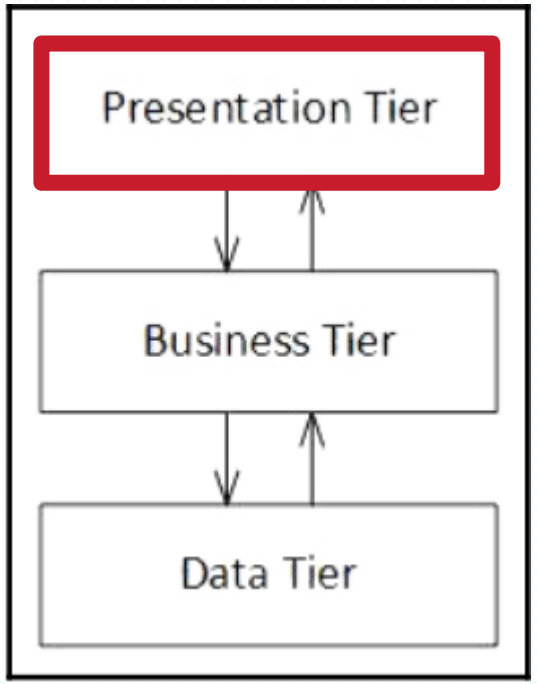
3-Tier architecture

- One of the most widely-used variations of this type of layered architecture is the three-tier architecture.
- The rise of the web coincided with a shift from two-tier (client-server) architectures to three-tier architectures.
- With web applications and the use of web browsers, rich client applications containing business logic were not ideal.
- The three-tier architecture separates logic into presentation, business, and data layers



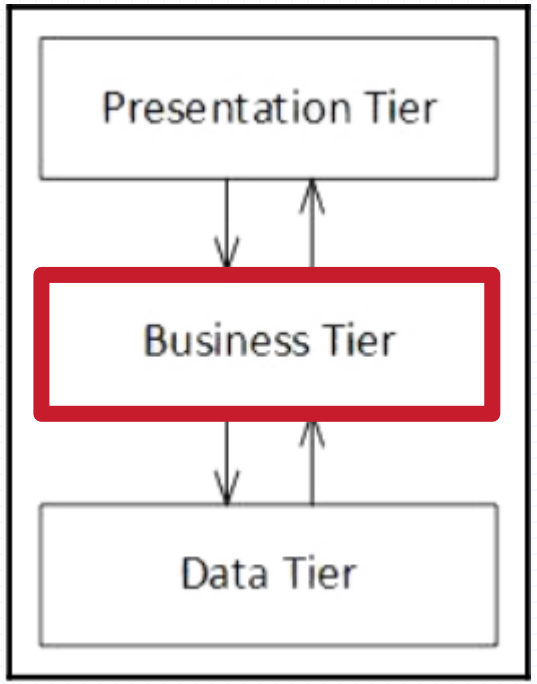
3-Tier architecture - Presentation tier

- The logic in the presentation tier should focus on user interface concerns.
  - Data is presented to the user and input is received from users in this tier.
  - Logic to render the user interface, including the placement of data, formatting the data, and hiding/showing UI components as required
- Basic validation.
  - Developers should be careful not to introduce business logic into the validation, which should be handled by the business tier.



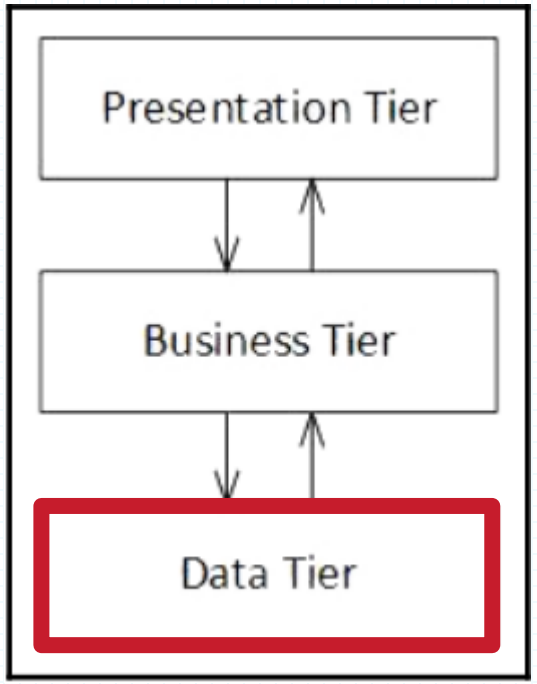
3-Tier architecture - Business tier

- Provides the implementation for the business logic of the application, including such things as business rules, validations, and calculation logic.
- Business entities for the application's domain are placed in this tier.
- The business tier is the center of the application and serves as an intermediary between the presentation and data tiers.
  - It provides the presentation tier with services, commands, and data that it can use,
  - and it interacts with the data tier to retrieve and manipulate data.



3-Tier architecture - Data tier

- The data tier provides functionality to access and manage data.
- In some systems, there is a data access or persistence layer in addition to a data or database layer.
  - The persistence layer contains components for data access, such as an object relational mapping (ORM) tool,
  - and the database layer contains the actual data store, such as an RDBMS.
- One reason to separate these into two distinct layers is if you wanted the ability to switch out your data access or database technology for a different one.



Advantages

- This pattern reduces complexity by achieving a **Separation of Concerns (SoC)**.
  - Dependencies are organized in an understandable way
- This architecture pattern **increases the testability quality attribute** of software application
  - For example, you can perform unit testing on classes in your business layer without the presentation and data layers.
- **Individual layers can be reused, modified, or replaced**
  - Peel off UI layer, replace with different style of UI
  - Modify data storage layer to use a different database
  - Lower layers can be reused in an entirely different application

Disadvantages

- Lower efficiency
  - **Overhead** involved in moving between layers
- A requirement change **may require changes in multiple layers**. This type of coupling lowers the overall agility of the software application.
  - For example, adding a new field will require changes to multiple layers
- **Dependencies** between layers can cause problems when a layer needs to be modified.
  - See next slid!