Software Design and Architecture

[Architecture Patterns] – Chapter 04, L01

HW: Serverless architecture

Lecture Outlines

- Overview of Architecture Styles and Patterns
- History of Architectural Styles and Patterns
  - ✓ Origin of styles and patterns

  Extra:
- **Architectural Style:** A high-level template that defines the overall structure of the system (like Layered).
- **Architectural Pattern:** A specific, reusable solution to a common problem inside that structure (like MVC within a Layered style).
- **In short:** Styles set the big picture, patterns provide the detailed recipe.

- Classification of Architectural Styles and Patterns
  - ✓ Data-Centered
  - ✓ Data flow
  - ✓ Distributed
  - ✓ Interactive
  - ✓ Hierarchical

Extra:

Example: Calculating the Average

- **Step 1: Define the list of numbers.**
- **Step 2: Find the total.**
- **Step 3: Calculate the average.**

Applying to Different Problems

1. **Students' Average Score: Calculate average scores of students in a session.**
2. **Weekly Temperature Average: Calculate average temperature over a week.**

Connection to Software Engineering Concepts

Architectural Style:

Data flows **Style:**

Each step in the process (input, processing, output) can be viewed as a stage in the flow of data.

For example, input (list of numbers), processing (sum), output (average).

Architectural Pattern

Pipes-and-Filters:

Each step (define list → find total → calculate average) acts as a filter, transforming the data and passing it to the next stage.

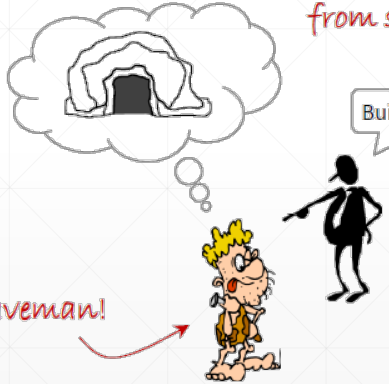
This allows scalability and reuse for different average calculations.

Overview

- Software systems need to be carefully architected and evaluated from different perspectives.
  - ✓ This is necessary to address multiple concerns that shape the quality of the system from different stakeholders with different backgrounds.
- In this chapter, we pay special attention (mostly) on decomposing the software system into logical components that represent the structural integrity that supports functional and non-functional (quality) requirements.
- When designing logical architectures, it is important to use past experience to discover overall strategies that have been used successfully in the development of software systems.
  - ✓ To this end, the concepts of architectural styles and architectural patterns have emerged as mainstream approach for achieving reuse at the architectural level.

Overview

It is much harder for the Caveman to come up with the structural architecture of the house from scratch!

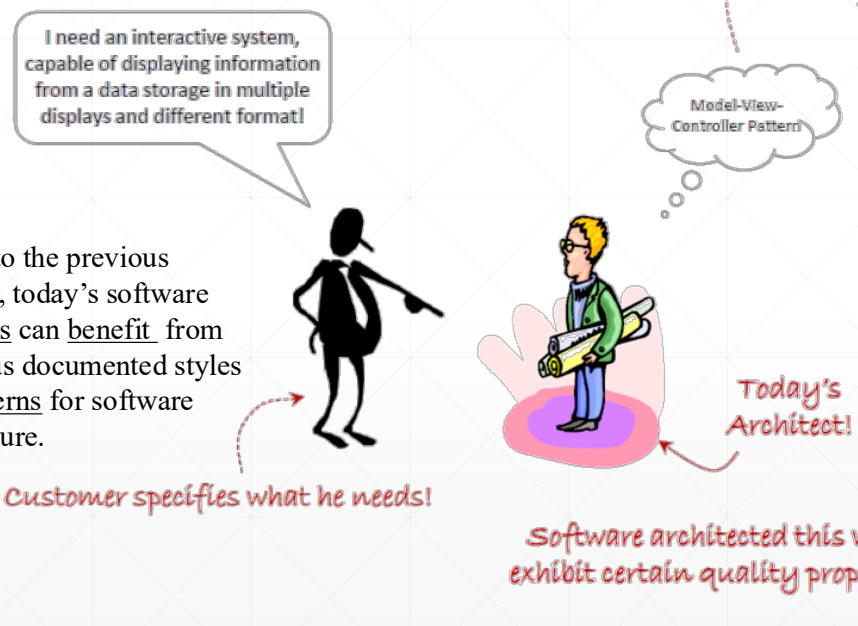


Today's Architects benefit from years of documented experience!

Important:
It is more efficient to build systems based on successfully employed designs than to come up with designs from scratch!

Overview

Similar to the previous example, today's software architects can benefit from numerous documented styles and patterns for software architecture.



Overview

- Architectural styles and patterns provide **generic, reusable solutions** that can be easily understood.
  - ✓ These can be easily applied to new problems requiring similar architectural features.
  - ✓ What is the benefit of this reuse???? (discuss)
    - Extra:
      1. **Improved Quality**: Patterns are based on best practices that have been refined over time, leading to more robust and reliable software.
      2. **Consistency**: Ensures uniformity in architecture across different parts of an application or different projects within an organization, making the system easier to understand and maintain.
      3. **Increased Efficiency**: Using well-understood and tested patterns speeds up the design and implementation phases.
- **Decisions based on architectural styles and patterns benefit from years of documented experience that highlights**
  - ✓ The **solution approach** to a given problem.
  - ✓ The **benefits** of these approaches.
  - ✓ The **consequences** of employing these approaches.

History of Software Architecture Styles and Patterns

- In 1977, Christopher Alexander presented a language intended to help individuals, or teams, design quality structures of different **sizes, shapes, and complexities**.
  (i.e. this was for buildings)

- Alexander's work resulted in a catalogue of 253 patterns, each describing in detail the essential information required for documenting the patterns, including:
  - ✓ Picture of the pattern
  - ✓ Context of the pattern
  - ✓ Problem that the pattern attempts to solve
  - ✓ Evidence for its validity
  - ✓ Related patterns

History of Software Architecture Styles and Patterns

- In the **1990s**, the software engineering community began researching and finding recurring high-level problem solutions in terms of specific elements and their relationships; these were originally referred to as **architectural styles**.
  - ✓ Similar to Alexander's work, Architectural Styles provided the means for software architects to reuse design solutions in different projects; that is, to use a "solution a million times over, without ever doing it the same way twice." (its focus: **solution)**

Extra:

1. **Alexander's example:** windows placement in buildings.
   - ✓ Example: He described a pattern like "Light on Two Sides of Every Room" → if you design a room, give it windows on two sides for better natural light.
   - ✓ Focus: documenting patterns for reusing good design ideas in physical buildings.
2. **Software's example:** Pattern idea: *Client–Server* → clients send requests, server responds.
   - ✓ **Reuse:** Many projects use this idea.
   - ✓ **Adapt differently:**
     - In **Gmail**, the client is a browser; the server handles emails.
     - In a **Banking App**, the client is a mobile app; the server handles transactions.
   - ✓ 👉 Same **pattern reused**, but each project **implements it in its own way**.

History of Software Architecture Styles and Patterns

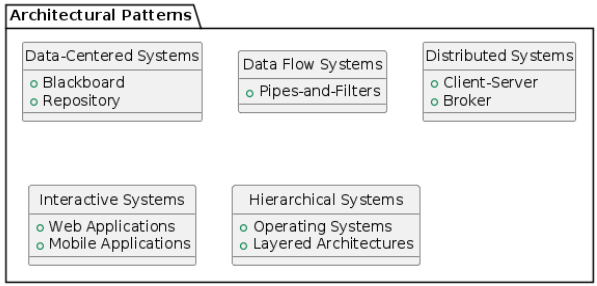
- In **1994,** Gamma, Helm, Johnson, and Vlissides—better known as the Gang of Four (GoF)—published their influential work that focused on a finer-grained set of object-oriented detailed design solutions that could be used in different problems "a million times over, without ever doing it the same way twice."
  - ✓ Influenced by Alexander's work, they called these **Design Patterns**.
  - ✓ Their work resulted in the creation of a catalogue of 23 (detailed design) patterns.
  - ✓ Each pattern was described in detail, using a specific pattern specification format.

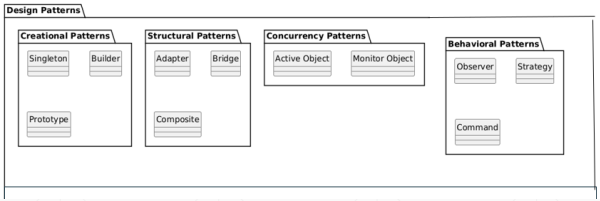
  Extra: In other words,

  These patterns allow developers to apply similar solutions in various situations without repeating the same code, inspired by Alexander's architectural design principles. Each pattern is thoroughly described in a specific format to make them easy to understand and implement.

History of Software Architecture Styles and Patterns

- In **1996**, the work of Buschmann, Meunier, Rohnert, Sommerland, and Stal, integrated the work of architectural styles and design patterns by providing a set of well-known architectural styles using a pattern-like approach.
  - ✓ They referred to these as **Architectural Patterns**. (its focus: **context-problem-solution)**
- In their work, Buschmann et al. provided their views about architectural patterns vs. design patterns:
  - ✓ "***Architectural patterns*** *... specify the system-wide structural properties of an application. They specify the system-wide structural properties of an application, and have an impact on the architecture of its subsystems.*"
  - ✓ "**Design patterns** are medium-scale patterns."
  - ✓ "*The application of a design pattern has no effect on the fundamental structure of a software system, but may have a strong influence on the architecture of a subsystem.*"





Extra:

Architectural Patterns vs. Design Patterns (Buschmann et al., 1996)

- **Architectural Patterns**
  - Define the **overall structure** of the whole system.
  - Affect how **all subsystems** are organized and interact.
  - Example: **Client–Server**, **Layers**, **Pipes-and-Filters**.
- **Design Patterns**
  - Define solutions at a **smaller, subsystem or component level**.
  - Do **not change the overall system structure**, but influence how parts inside subsystems are built.
  - Example: **Observer**, **Factory**, **Strategy**.

In short:

- **Architectural Pattern = Big picture (system-wide structure).**
- **Design Pattern = Medium picture (subsystem/component behavior).**

Extra

طارح سؤال عن السؤال بس ممكن سؤال
ممكن وش هو؟

MCQs xTorF

1977 – Christopher Alexander

- What he did: Created a catalogue of 253 patterns (originally for buildings).
- Focus: Documenting patterns → described problems + solutions + context.
- Key Idea: Gave the world the language of patterns.

📌 **1990s – Software Engineering Community**

- What they did: Introduced Architectural Styles.
- Focus: Solution reuse at a high level (whole system structures).
- Key Idea: Reuse big system structures (e.g., client–server, layered).

📌 **1994 – Gang of Four (GoF)**

- What they did: Published the Design Patterns book (23 OO patterns).
- Focus: Problem–solution reuse at a smaller scale (classes, objects).
- Key Idea: Standardized and popularized design patterns.

object oriented

📌 **1996 – Buschmann et al.**

- What they did: Combined Architectural Styles + Design Patterns.
- Focus: Context–problem–solution at system level.
- Key Idea: Introduced Architectural Patterns (system-wide) vs Design Patterns (subsystem-level).

✅ **In short:**

1. Alexander (1977): First to document many patterns → **gave the language.**
2. SE Community (1990s): Added architectural styles → **reuse at system level.**
3. GoF (1994): Focused on design patterns → **reuse at class/object level.**
4. Buschmann et al. (1996): Unified everything → **introduced architectural patterns (big) vs design patterns (smaller).**

Architecture Patterns Classification

- **The choice of applying architectural patterns depend on** the type of system, requirements, and desired quality attributes.
  - ✓ These characteristics help guide the choice of selecting one particular pattern over another.
- In some cases, **more than one architectural pattern can be used** in combination to collectively provide the appropriate architectural solution. (specially for large systems)
- Architectural patterns can be classified depending on the type of system.

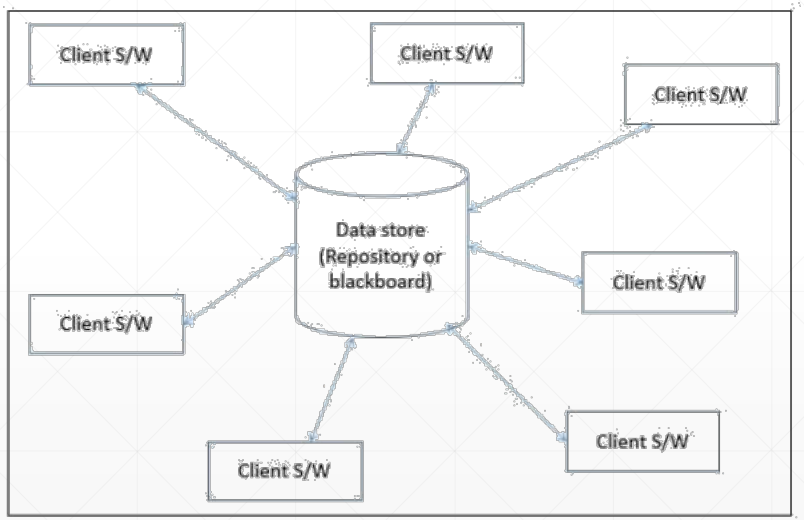
Architecture Patterns Classification

- **Major types of systems:**

Type	Description
Data-Centered	Systems that serve as a centralized repository for data, while allowing clients to access and perform work on the data.
Data Flow	Systems oriented around the transport and transformation of a stream of data.
Distributed	Systems primarily involve interaction between several independent processing units connected via a network.
Interactive	Systems that serve users or user-centric systems.
Hierarchical	Systems where components can be structured as a hierarchy (vertically and horizontally) to reflect different levels of abstraction and responsibility.

Data Centered Systems

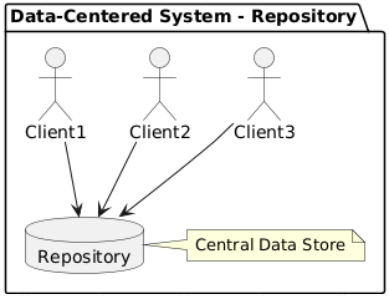
- Systems that serve as a centralized repository for data, while allowing clients to access and perform work on the data



- **Used in DBMS, library information system, the interface repository in CORBA, compilers and CASE tools, speech recognition, image recognition, security system, and business resource management systems**

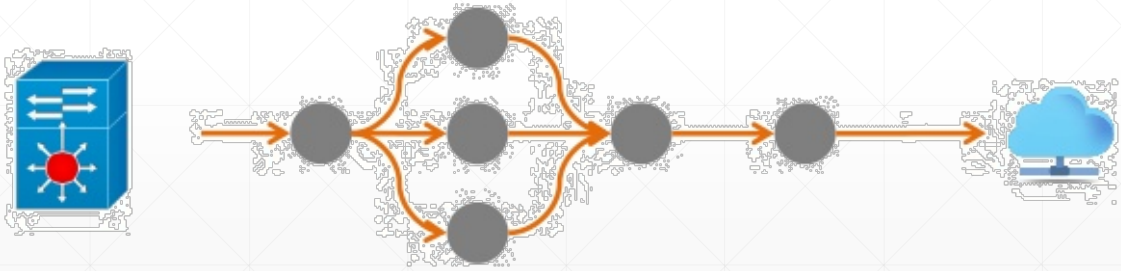
Extra: Data Centered Systems

- Patterns:
- Blackboard, Repository
- Example:
- Central data repository systems where different components interact primarily through a central data store.
- Explanation: In such systems, the core component is a central database or repository where data is stored and managed. Components of the system interact with this central repository to retrieve, add, or modify data. *They do not communicate directly with each other but rather use the repository as an intermediary.*



Data Flow Systems

- Systems oriented around the **transport** and (or) **transformation** of a stream of data (Example: video streaming)



Extra: Data Flow Systems

- Pattern:

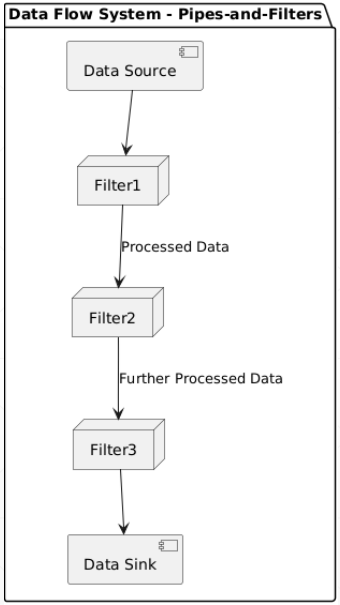
  Pipes-and-Filters

- Example:

- Data processing systems where data passes through a chain of components, each transforming the data.

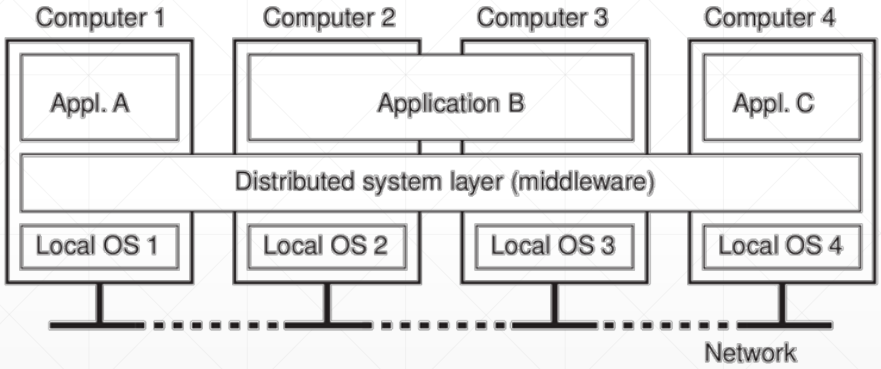
- Explanation:

- This pattern involves a series of processing elements called filters, connected by pipes which transmit data from one filter to the next. Each filter processes the input data and produces output data, often transforming the data in some way. This setup is commonly used in signal processing or data streaming applications.



Distributed Systems

- Systems primarily involved in interaction between several independent processing units connected via a network
- **Example:** network applications like the world wide web (web applications)



Extra: Distributed Systems

Patterns: **Client-Server**, **Broker**

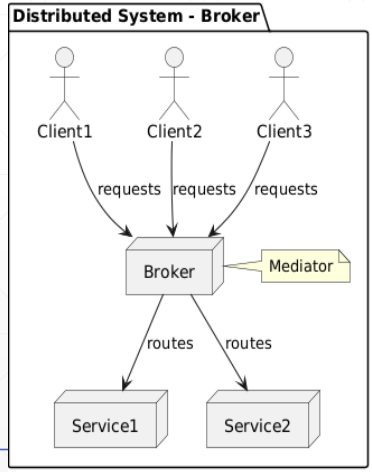
Example:

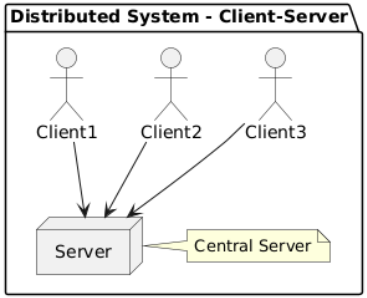
Systems distributed across multiple networked components, managing communication and data exchange.

-Explanation:

- **Client-Server**: This is a decentralized pattern where multiple clients (users' machines) request resources and services from a central server. The server processes these requests and returns the responses. This model is fundamental in web interactions. Such as, in web applications where a web server handles requests from web browsers (clients) to serve web pages

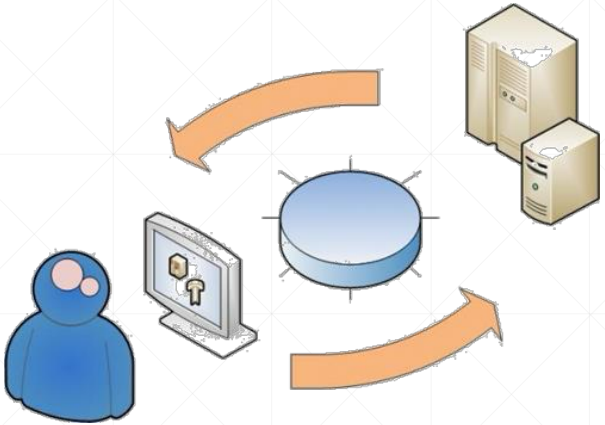
- **Broker**: In this pattern, a broker component is responsible for coordinating communication between components, often in different processes or across networks. The broker routes requests, translates messages, and often ensures security and scalability. such as in service-oriented architectures (SOA) where multiple services interact through a messaging system.





Interactive Systems

- Systems that serve users (user-centric systems) and require heavy interaction between the user and the system.
- **Example:** mobile applications



Extra:

- **If the mobile app works offline only (e.g., calculator, offline dictionary, offline game) → Interactive System.**
- **If the mobile app requires server communication (e.g., WhatsApp, Gmail, banking app) → Distributed System.**

Extra: Interactive Systems

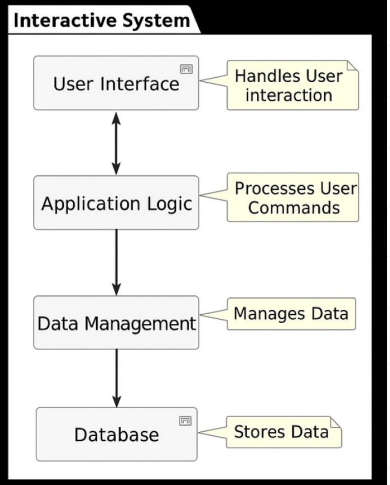
Interactive Systems

- **Description:**

  Systems where the user interacts directly with the software, and all logic + data are handled on the same machine.

- **Example:**
  - Desktop Calculator App
  - Flow:

    User Interface → Application Logic → Data Management → Local Database



Extra

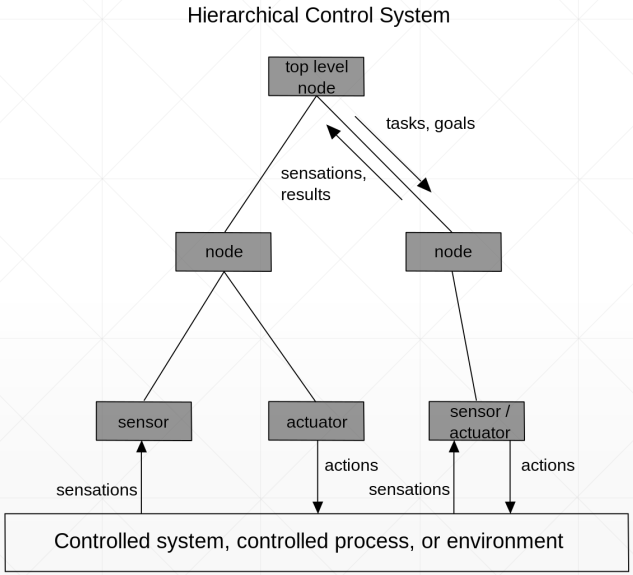
Interactive vs Distributed Systems (Simplified)

- **Interactive (Local)** → offline
  - Runs fully on one machine.
  - Example: Calculator, offline dictionary.
  - Works offline, single user.

- **Distributed (Networked)**
  - Split across multiple machines (client + server).
  - Example: Gmail, banking app.
  - Needs internet, supports many users.

Hierarchal Systems

- System where components can be structured as a hierarchy (vertically and horizontally) to reflect different levels of abstraction and responsibility
- **Example:** control systems in manufacturing, robotics and vehicles



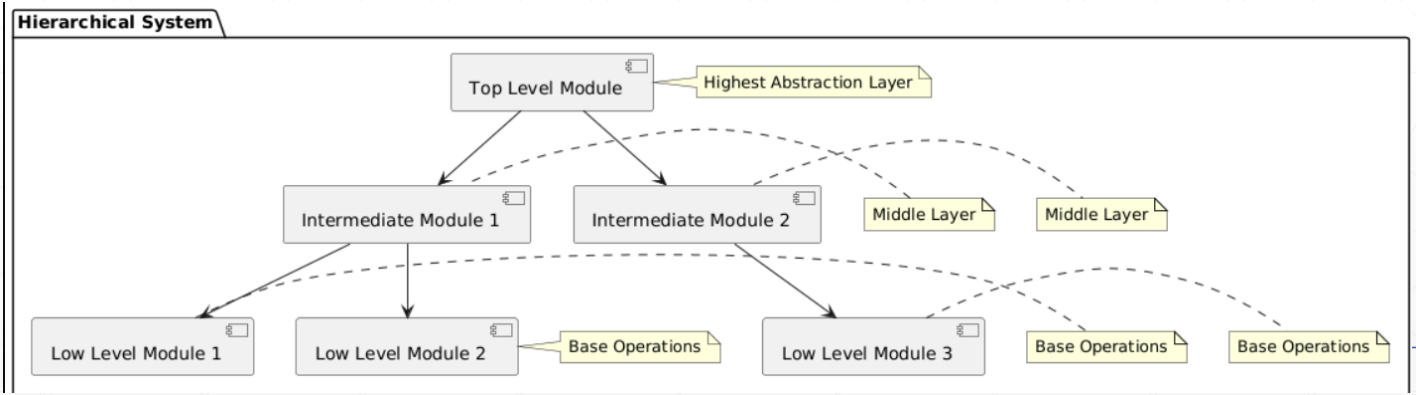
Extra: Hierarchal Systems

•Description:

These systems are structured in layers, each layer having a specific role or responsibility, allowing for the separation of concerns across a vertical (layered) or horizontal (module) scale.

•Example:

•Operating systems where different layers handle tasks ranging from hardware interaction at the lowest level to user interface management at the highest level.



Extra: Question

For each of the specified system requirements below, identify the architectural pattern that best fits from the following patterns

[Blackboard, Pipe-Filter, Client-Server, Broker Client-Server]

No.	Requirements	Pattern/Patterns
1	A video processing application that sequentially performs a set of operations on video files, such as decoding, applying filters (e.g., brightness, contrast adjustments), and then encoding the output.	Pipe-and-Filter data flow
2	An online banking system that allows customers to perform transactions, manage accounts, and communicate with customer service through a web interface, mobile app, and desktop application.	Client-Server disk related System
3	A financial analysis system addresses complex problems through collaborative modules. Each module contributes its expertise towards crafting personalized investment strategies, dynamically responding to new market data. This collaborative approach, essential for integrating diverse insights, is crucial for coming up with correct decisions.	Blackboard Data Centered

Extra: A **DBMS** can be viewed in **two ways**:

- As a **Repository** (Data-Centered Style) → all apps use one central shared database.
- As part of a **Client-Server system** (Distributed Style) → clients send SQL queries, server processes them.

Summary

Our work will be:

- Overview of Architecture Styles and Patterns
- History of Architectural Styles and Patterns
  - ✓ Origin of styles and patterns
- Classification of Architectural Styles and Patterns
  - ✓ Data-Centered
  - ✓ Data flow
  - ✓ Distributed
  - ✓ Interactive
  - ✓ Hierarchical
- Next …, we will discuss two types of systems: Data-centered and Data Flow, together with essential architectural patterns for these systems, including: **Blackboard** and **Pipes-and-Filters**