

OVERVIEW **Architecture Patterns: Overview, History and Classification**

Architects rarely design from scratch: architectural styles and patterns are documented, reusable solutions with known benefits and consequences. This lecture separates the terms (a style is the overall shape, a pattern a specific recipe; architectural patterns are system-wide, design patterns work inside subsystems), traces their history from Alexander's building patterns to Buschmann's architectural patterns, and classifies systems into five types, each with its typical patterns. Lectures 2 and 3 then study those patterns one by one.

Style vs. pattern

A style is a high-level template for the whole structure (e.g. data flow); a pattern is a specific reusable solution within it (e.g. pipes and filters).

History

1977 Alexander (253 building patterns) → 1990s architectural styles → 1994 Gang of Four (23 design patterns) → 1996 Buschmann et al. (architectural patterns).

Architectural vs. design patterns

Architectural patterns fix system-wide structure; design patterns are medium-scale and don't change the fundamental structure.

Five system types

Data-centered (blackboard, repository), data flow (pipes and filters), distributed (client-server, broker), interactive (MVC), hierarchical (layered).

HOW THE LECTURE IS ORGANIZED

Slides 1–7 Styles vs. patterns, and why reuse them

Slides 8–13 History: Alexander, styles, Gang of Four, Buschmann

Slides 14–26 Classification: data-centered, data flow, distributed, interactive, hierarchical

Slides 27–28 Exercise and summary

EXAM FOCUS

The classification table (system type → pattern) is the base of every pattern question. Midterm 261 Q6 (3.5 marks) matched seven scenarios to layered, pipe-and-filter, broker, blackboard, MVC, client-server and peer-to-peer; Final 252 Q4 and Midterm 252 Q5 asked you to pick and draw a pattern for a scenario.

HOW TO READ THIS PAGE

Every original slide is shown unchanged. **Yellow marks** on a slide point to a numbered note: hover or tap a mark to jump to its note, and hover a note to see its marks. Questions tagged **Past exam** appeared on a real paper; **Prediction** questions are likely angles, not confirmed.



Software Design and Architecture

[Architecture Patterns] – Chapter 04, L01

HW: Serverless architecture

NOTES

- 1 Chapter 4 is about **architecture patterns**: ready-made system structures. This first lecture is the overview and classification; the patterns themselves come in Lectures 2 and 3.
- 2 Homework on the title slide: research **serverless architecture** (applications built from cloud functions that the provider runs on demand, with no server you manage).

SLIDE 2 / 28 Lecture Outlines

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**

NOTES

- 1 An **architectural style** is a high-level template that defines the overall structure of a system, like "layered".
- 2 An **architectural pattern** is a specific, reusable solution to a common problem inside that structure, like MVC within a layered system. In short: styles set the big picture, patterns give the detailed recipe.
- 3 The classification at the end uses five system types: data-centered, data flow, distributed, interactive and hierarchical.

SLIDE 3 / 28 Extra: Calculating the average, a reusable recipe

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.

3

NOTES

- 1 An analogy for reuse: once you know the three steps for an average (list the numbers, total them, divide), you reuse them for students' scores, weekly temperatures, anything.
- 2 Seen as a **style**, the steps are a flow of data: input (the list) → processing (the sum) → output (the average).
- 3 Seen as a **pattern**, each step is a *filter* that transforms the data and passes it to the next one: pipes and filters. Because each step is independent, the same pipeline is reused for different averages.

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.

NOTES

- 1 Systems must be architected and evaluated from **different perspectives**, because stakeholders with different backgrounds care about different qualities.
- 2 This chapter focuses on decomposing a system into **logical components** whose structure supports the functional and non-functional requirements.
- 3 Good logical architectures come from **past experience**: strategies that already worked in other systems.
- 4 That experience is packaged as **architectural styles and patterns**, the mainstream way to achieve reuse at the architectural level.

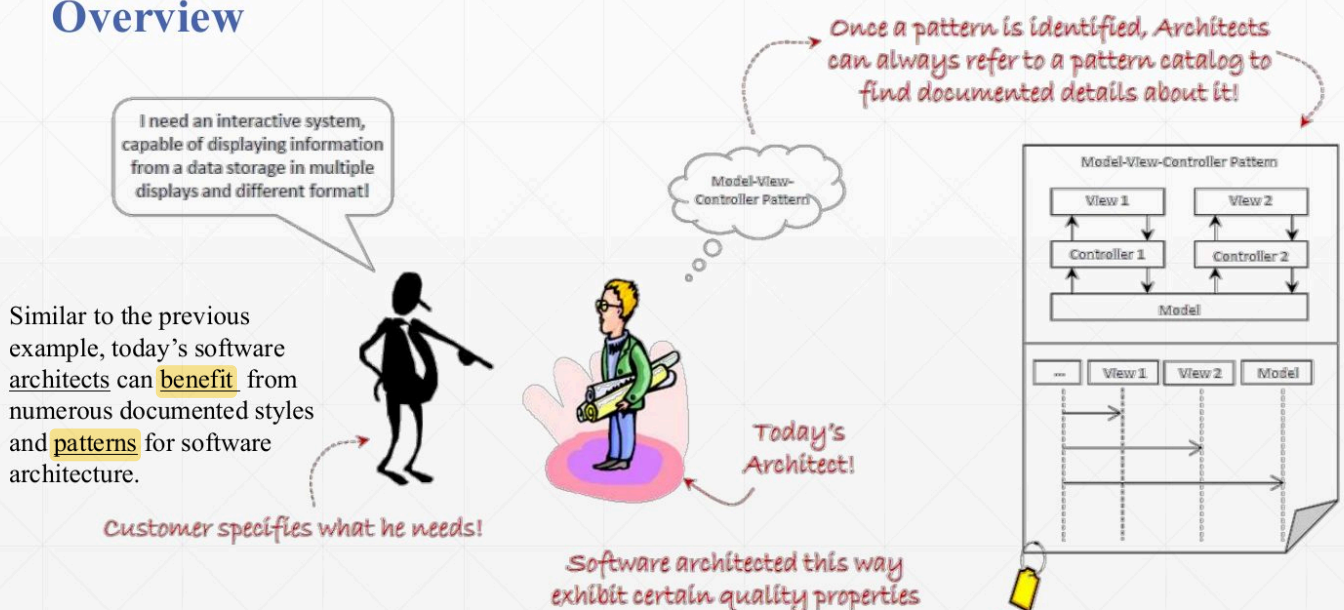
Overview



NOTES

- 1 The caveman has to invent the structure of a house from scratch, which is slow and risky.
- 2 Today's architect reuses years of documented experience. The boxed point is the lesson: it is more efficient to build systems from successfully employed designs than to design from scratch.

Overview



NOTES

- 1 The software version of the same story: the customer asks for “an interactive system that displays data from storage in multiple displays and formats”, and the architect recognises the requirement as the **Model-View-Controller** pattern.
- 2 Once a pattern is identified, the architect looks it up in a pattern catalogue for its documented structure (the MVC class and sequence diagrams on the right) and knows the quality properties a system built that way will have.

SLIDE 7 / 28 Overview

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.

NOTES

- 1 Styles and patterns are **generic, reusable** solutions that are easy to understand and apply to new problems with similar needs.
- 2 Benefits of that reuse: **improved quality** (they encode refined best practice), **consistency** (the same structure across an application or across projects, so it is easier to understand and maintain), and **efficiency** (well-understood, tested designs speed up design and implementation).
- 3 Decisions based on them benefit from **years of documented experience** describing the solution, its benefits, and the consequences (trade-offs) of using it. Knowing the consequences in advance is the real value.

Prediction

Q1

What is the benefit of reusing architectural styles and patterns?

Answer: They are proven, documented solutions: quality improves because they encode best practice, designs are consistent and easier to maintain, design and implementation are faster, and their benefits and consequences are known in advance, which lowers risk and gives the team a shared vocabulary.

SLIDE 8 / 28 History of Software Architecture Styles and Patterns

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

Prince Sultan University | Slides prepared from "Software Engineering Design: Theory and Practice" book and slides

8

NOTES

- 1 In 1977 **Christopher Alexander**, a building architect, published a pattern language to help people design good buildings and towns of any size.
- 2 It became a catalogue of **253 patterns**, each documented with a picture, its context, the problem it solves, evidence that it works, and related patterns. That documentation format is the ancestor of every software pattern catalogue.

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**.

NOTES

- 1 In the **1990s** software engineers began cataloguing recurring high-level solutions (elements and their relationships) and called them **architectural styles**. Their focus was the solution.
- 2 Like Alexander's patterns, a style can be used "**a million times over, without ever doing it the same way twice**": the idea is reused, the details differ every time.
- 3 Alexander's example: "light on two sides of every room". The software example: client-server, reused in Gmail (browser client, mail server) and a banking app (mobile client, transaction server), each implemented 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.

10

NOTES

- 1 In **1994** Gamma, Helm, Johnson and Vlissides, the **Gang of Four (GoF)**, published finer-grained, object-oriented **detailed design** solutions.
- 2 Inspired by Alexander, they called them **design patterns**: a catalogue of **23** patterns, each described in a fixed specification format. Chapters 6–7 study several of them (Singleton, Builder, Composite, Observer...).

EXAM ANGLE

Prediction

Q1

The Gang of Four (1994) catalogued how many design patterns, at what level?

- A. 23 detailed-design (object-oriented) patterns
- B. 253 building patterns
- C. 23 architectural styles
- D. 5 system types

Answer: A. 23 detailed-design (object-oriented) patterns

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.”

11

NOTES

- 1 In 1996 **Buschmann** et al. combined the two lines of work: well-known architectural styles written in a pattern format (context – problem – solution). They called these **architectural patterns**.
- 2 Their definition: architectural patterns specify the **system-wide structural properties** of an application and affect the architecture of its subsystems.
- 3 Design patterns are **medium-scale**: applying one has **no effect on the fundamental structure** of the system, though it may strongly shape one subsystem. That sentence is the clearest way to tell the two apart.

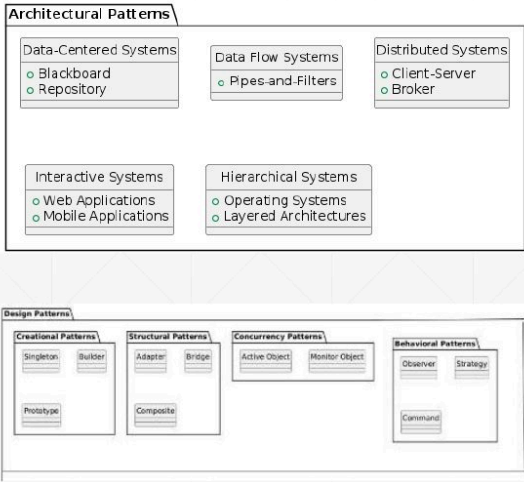
EXAM ANGLE

Prediction

Q1

How do architectural patterns differ from design patterns (Buschmann et al.)?

Answer: Architectural patterns specify system-wide structural properties and affect the architecture of all subsystems (e.g. client-server, layers). Design patterns are medium-scale: they do not change the fundamental structure of the system, only the design inside a subsystem (e.g. Observer, Factory).



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).**

NOTES

- 1 **Architectural patterns** define the overall structure of the whole system and how all subsystems are organised and interact: client-server, layers, pipes and filters. The diagram groups them by system type (data-centered: blackboard, repository; data flow: pipes and filters; distributed: client-server, broker; interactive and hierarchical).
- 2 **Design patterns** are solutions at the subsystem or component level that do not change the overall structure: Observer, Factory, Strategy. The lower diagram groups them as creational, structural, concurrency and behavioural.
- 3 In short: architectural pattern = the big picture (system-wide structure); design pattern = the medium picture (subsystem or component behaviour).

Extra

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.

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)**.

13

NOTES

- 1 1977, Alexander:** 253 building patterns documenting problem, solution and context. He gave the world the language of patterns.
- 2 1990s, the software community:** architectural styles, reuse of whole-system structures (client–server, layered).
- 3 1994, GoF:** the Design Patterns book, 23 object-oriented patterns at the class and object level.
- 4 1996, Buschmann et al.:** combined both into architectural patterns (system-wide) as opposed to design patterns (subsystem level).

EXAM ANGLE**Prediction****Q1**

Put in order and match: Gang of Four, Buschmann et al., Alexander, architectural styles.

Answer: Alexander 1977 (253 building patterns) → architectural styles in the 1990s → Gang of Four 1994 (23 design patterns) → Buschmann et al. 1996 (architectural patterns).

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.

NOTES

- 1 Which pattern to apply depends on the **type of system, its requirements and the desired quality attributes**. These, not the pattern's popularity, decide between patterns.
- 2 Large systems often **combine patterns**: a client-server system whose server is internally layered, for example.
- 3 Patterns are classified by the **type of system** they suit (next slide).

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.

NOTES

- 1 The five system types, each with its typical patterns (slides 16–26).
- 2 The table:
 - Data-centered**
A centralised repository of data that clients access and work on → blackboard, repository.
 - Data flow**
Built around transporting and transforming a stream of data → pipes and filters.
 - Distributed**
Independent processing units interacting over a network → client–server, broker.
 - Interactive**
User-centric systems → MVC.
 - Hierarchical**
Components arranged in levels of abstraction and responsibility → layered (main program and subroutine).

Past exam · Midterm 261 · Q6 · 3.5 marks

Q1

Match each scenario with the most suitable architectural style (each used once): (1) payroll app whose UI, salary rules and data storage must evolve independently; (2) security-log app where separate units extract fields, normalise timestamps, remove sensitive values and summarise; (3) hospital app talking to lab equipment from different vendors through one request format; (4) speech-understanding app whose components repeatedly add and reconsider partial interpretations; (5) museum ticketing shown differently on desktop and kiosks, updated by user actions; (6) classroom response app with one authoritative session for hundreds of phones; (7) emergency messaging between nearby phones with no Internet and no central node.

Answer: (1) **Layered**. (2) **Pipe-and-filter**. (3) **Broker**. (4) **Blackboard**. (5) **MVC**. (6) **Client-server**. (7) **Peer-to-peer**. Layered separates responsibilities; pipe-and-filter chains independent transformations; broker hides vendor-specific communication; blackboard combines partial solutions through shared knowledge; MVC separates model, views and control; client-server uses one authoritative service; peer-to-peer needs no central node.

Prediction

Q2

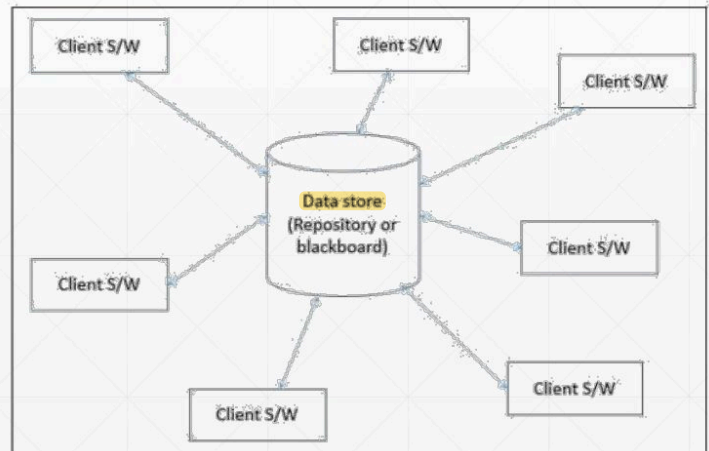
Pipes and filters belongs to which system type?

- A. Data flow
- B. Data-centered
- C. Distributed
- D. Hierarchical

Answer: A. Data flow

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



16

NOTES

- 1 **Data-centered** systems are a centralised repository of data that clients access and work on.
- 2 Uses: DBMSs, library information systems, the CORBA interface repository, compilers and CASE tools, speech and image recognition, security systems, business resource management.
- 3 The diagram: many client software components around one **data store** (repository or blackboard). Clients don't talk to each other, only to the store. Sharing data is easy; the store can become a bottleneck and a single point of failure.

Extra: Data Centered Systems

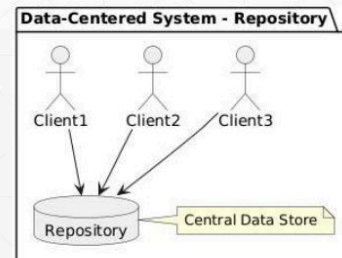
➤ Patterns:

➤ 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.*

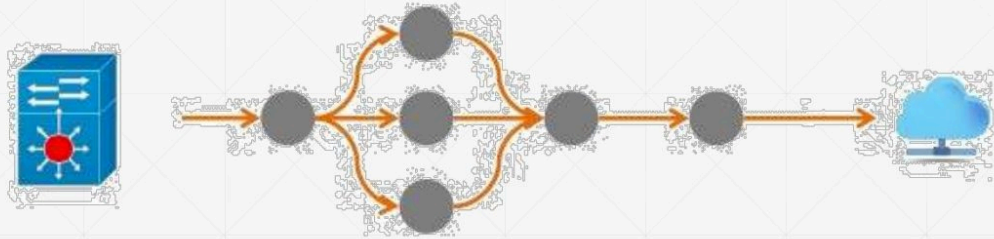


NOTES

- 1 Patterns for data-centered systems: **blackboard** and **repository**.
- 2 Components interact through a **central data store**: they retrieve, add and modify data there.
- 3 They **do not communicate directly** with each other; the repository is the intermediary. In a repository the store is passive; in a blackboard, components react to changes on the shared data (Lecture 2).

Data Flow Systems

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



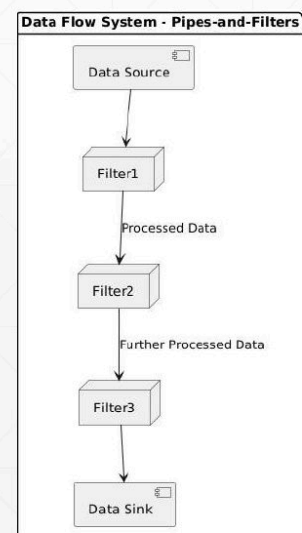
NOTES

- 1 **Data flow** systems are organised around the **transport and/or transformation** of a stream of data.
- 2 Example: video streaming. The picture shows data entering from a source, passing through processing nodes (which may branch and rejoin), and leaving to the cloud: each node transforms the stream and hands it on.

SLIDE 19 / 28 Extra: Data Flow Systems

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.



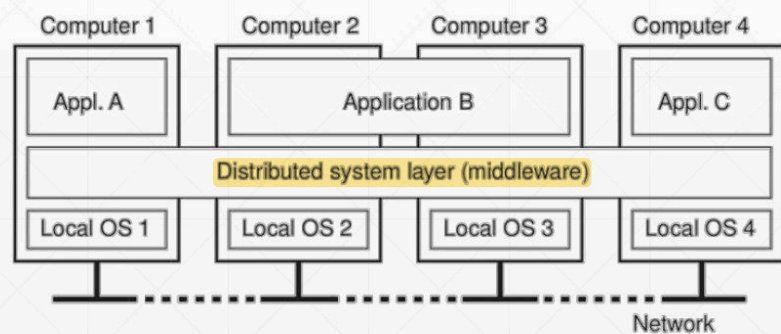
NOTES

- 1 Pattern for data flow systems: **pipes and filters**.
- 2 Processing elements called **filters** are connected by **pipes** that carry data from one filter to the next. Each filter takes input, transforms it, and produces output.
- 3 Typical uses: signal processing and data streaming, compilers (lexer → parser → code generator), image and video processing.

SLIDE 20 / 28 Distributed Systems

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)



NOTES

- 1 **Distributed** systems are made of **several independent processing units** that interact over a network.
- 2 Example: web applications (the World Wide Web).
- 3 The diagram: four computers, each with its own local OS, running applications A, B (spread over two machines) and C, joined by a **middleware** layer that hides the network so the applications can talk as if local. Network delays and partial failures are the price of distribution.

Extra: Distributed Systems

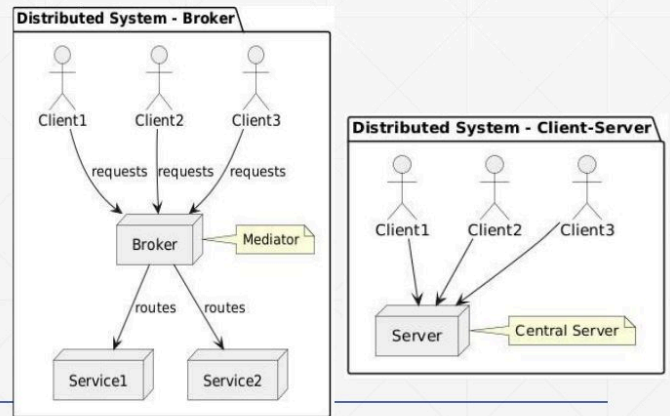
Patterns: **Client-Server**, **Broker**

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.



21

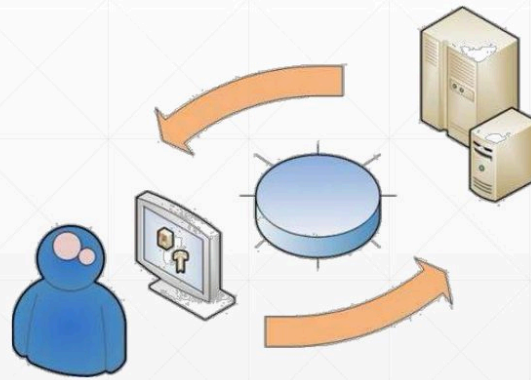
NOTES

- 1 Patterns for distributed systems: **client-server** and **broker**.
- 2 **Client-server**: many clients request resources and services from a central server, which processes the requests and returns responses. It is the basis of the web (browser clients, web server).
- 3 **Broker**: a broker component coordinates communication between components in different processes or machines. It routes requests, translates messages, and adds security and scalability, as in service-oriented architectures where services talk 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.

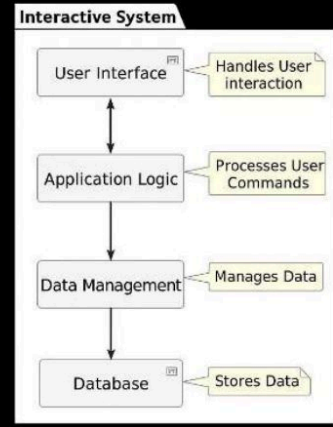
NOTES

- 1 **Interactive** systems serve users and need heavy interaction between the user and the system (user-centric).
- 2 Example: mobile applications.
- 3 The Extra rule: an app that works **offline only** (calculator, offline dictionary, offline game) is interactive; an app that **needs a server** (WhatsApp, Gmail, banking) is distributed. Many real apps are both: interactive on the phone, distributed overall.

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



NOTES

- 1 An interactive system as described here: the user interacts directly with the software, and all logic and data are handled on the same machine.
- 2 Example: a desktop calculator. The flow is user interface (handles interaction) → application logic (processes commands) → data management → local database. This separation of interface from logic is what MVC formalises.

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.

NOTES

- 1 **Interactive (local)**: runs fully on one machine, works offline, one user (calculator, offline dictionary).
- 2 **Distributed (networked)**: split across machines (client + server), needs the Internet, supports many users (Gmail, banking).

EXAM ANGLE

Prediction

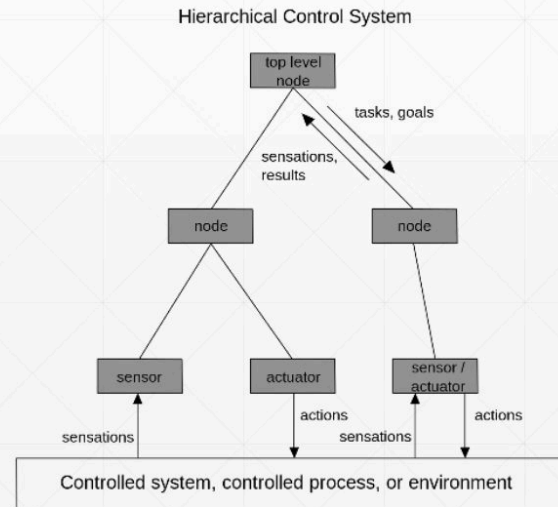
Q1

Is an offline dictionary app interactive or distributed? Is WhatsApp?

Answer: The offline dictionary is interactive (runs fully on one machine). WhatsApp is distributed (clients communicate through servers over the network).

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



NOTES

- 1 **Hierarchical** systems arrange components in a **hierarchy**, vertically and horizontally, reflecting levels of abstraction and responsibility.
- 2 Example: control systems in manufacturing, robotics and vehicles. In the diagram the top-level node sends tasks and goals down; lower nodes pass sensations and results up; sensors and actuators at the bottom touch the controlled process.

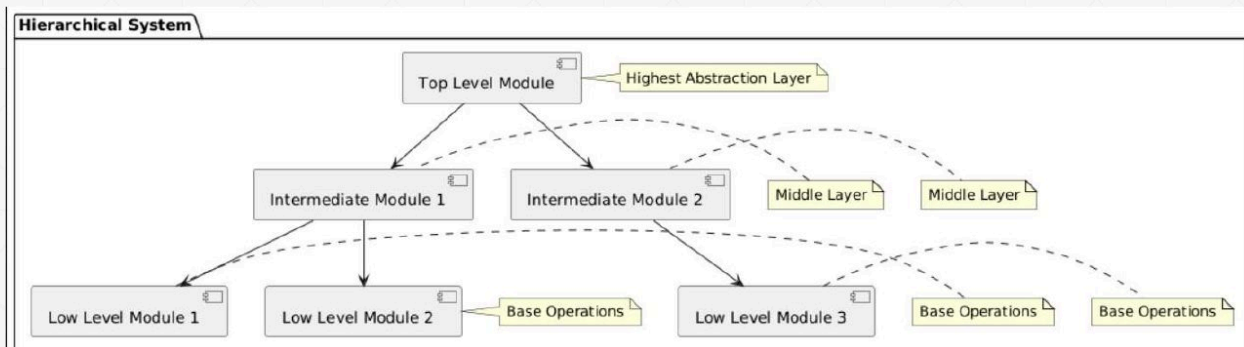
Extra: Hierarchal Systems

•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.



26

NOTES

- 1 Hierarchical systems are structured in **layers**, each with a specific responsibility, separating concerns vertically (layers) or horizontally (modules).
- 2 Example: an **operating system**, from hardware interaction at the lowest layer to user-interface management at the top. Each layer uses only the services of the layer below it. The pattern is **layered** (and main program and subroutine).

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 <i>data flow</i>
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 <i>distributed system</i>
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 <i>data centered</i>

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.

27

NOTES

- The exercise: pick the pattern for each requirement.
 - Video processing in a fixed sequence (decode, filter, encode)**
Pipe-and-filter: a chain of transformations.
 - Online banking used through web, mobile and desktop**
Client-server: many clients, one central service.
 - Financial analysis where collaborating modules contribute expertise to one evolving solution**
Blackboard: independent knowledge sources refine shared data.
- A **DBMS** can be seen two ways: as a **repository** (data-centered: all applications share one database) or as part of a **client-server** system (distributed: clients send SQL queries, the server processes them). The classification depends on the viewpoint.

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**

NOTES

- 1 Recap: styles vs. patterns, their history, and the five system types.
- 2 Next lecture: data-centered and data flow systems in detail, with the **blackboard** and **pipes-and-filters** patterns.

Original slides from the SE322 lecture deck, with study notes. Check exact wording against the [original deck](#).
Notes and questions are study aids. Made by Shoug Alomran.