Data Flow

Data Flow Architecture

Step	Details
1. What is?	**Definition:** A software architectural style where the availability of data controls computation. The structure is decided by the orderly motion of data from process to process, without interaction between processes except for data exchange. **Characteristics:** - Data flow system - Components (processing steps) are independent programs - No other interaction between processes except for data transfer
2. Model	**Components:** Data processing components, such as filters that read input data, perform computations, and write output data. **Connectors:** Data flow (stream); unidirectional, usually asynchronous, buffered.
3. Suitable For	This architecture is suitable for systems that handle continuous data processing with sequential transformations, like image processing, signal processing, and data transformation tasks. It is not suitable for interactive applications.
4. Context	Data Flow Architecture is typically used in systems where data moves through a series of independent processing steps, with each filter processing a portion of the data. It was initially proposed in the context of sequential data processing.
5. Adv & Disadv	**Advantages:** - High cohesion in filters - Low coupling between components - Reusability of filters - Extendable and flexible, supports both sequential and concurrent systems. **Disadvantages:** - Not suitable for interactive applications - Performance issues with complex filter designs - Not ideal for large shared data processing due to lack of common standards

Batch Sequential Architecture

Step	Details
1. What is?	**Definition:** In this architecture, tasks are decomposed into a series of computing units that interact only through data transfer. Each task in the sequence must be completed before the next starts. **Characteristics:** - Tasks are processed in sequence - Data is processed as a whole between steps
2. Model	**Components:** Independent processing steps. **Connectors:** Data transfer between tasks.
3. Suitable For	Batch Sequential Architecture is suitable for systems that perform large-scale data processing where tasks must be completed in a strict sequence and data can be processed in bulk, such as large-scale computation tasks or scientific simulations.
4. Context	This architecture is appropriate when data must be processed in large chunks or batches and sequential task execution is required. Typically used in batch processing or tasks that involve long-duration computations.
5. Adv & Disadv	**Advantages:** - Simple to implement - Predictable execution flow - Suitable for large, non-interactive datasets. **Disadvantages:** - High latency - No concurrency - Not ideal for interactive or real-time applications.

Pipe-and-Filter Architecture

Step	Details
1. What is?	**Definition:** A pipeline consisting of a chain of processing elements (filters). The output of each filter is passed to the input of the next filter in the sequence. Filters process data streams incrementally. **Characteristics:** - Data flows in one direction through filters - Each filter performs a transformation or computation on the data
2. Model	**Components:** Filters (processing steps) and pipes (data transmission channels). **Connectors:** Pipes transfer data from one filter's output to another filter's input.
3. Suitable For	Pipe-and-Filter Architecture is ideal for data stream processing tasks that involve continuous data flows, such as audio/video processing, compilers, and other signal processing tasks where incremental transformations are required.
4. Context	Used in environments where data can be processed in discrete steps, with each step (filter) transforming the data in some way before passing it on. Often used in compilers, image processing, and real-time data systems.
5. Adv & Disadv	**Advantages:** - High cohesion - Low coupling - Reusability of filters - Extensibility **Disadvantages:** - Not suitable for interactive applications - Can suffer from performance issues due to complex data transformations - Inflexible for large shared data processing tasks

Event Systems

Step	Details
1. What is?	**Definition:** Event Systems architecture is based on the publication and handling of events. It uses an implicit invocation style, where the triggerer of the event does not know which components will be affected. Components communicate through event announcements. **Characteristics:** - Implicit invocation - Separate interaction between event publishers and subscribers - One-to-many communication through publish/subscribe - Asynchronous operations
2. Model	**Components:** Event Source, Event Handler, Event Manager. Components announce events and register interest. The Event Manager invokes registered procedures when the event occurs. **Connectors:** Event-procedure bindings, event-event bindings. Components announce events at appropriate times, and the Event Manager dispatches them to registered procedures.
3. Suitable For	Event Systems architecture is ideal for systems where components need to operate independently, reacting to events in an asynchronous manner. It is commonly used in event-driven systems like editors, debuggers, and distributed systems that require decoupled communication.
4. Context	This architecture style is typically used in scenarios where multiple components need to independently handle specific events, such as real-time data processing, systems with multiple observers (e.g., user interface applications, monitoring systems).
5. Adv & Disadv	**Advantages:** - Decoupled components - Easy replacement and addition of components - High reuse potential - Fault isolation between managers and handlers. **Disadvantages:** - Dispatcher bottleneck - Difficult to handle simultaneous inputs - Potential system failure if the dispatcher malfunctions - No guarantee that an event will be handled.

Call/Return Style

Call/Return Style

Step	Details
1. What is?	**Definition:** A classic programming paradigm based on functional decomposition, where the system uses a hierarchy of procedure calls. **Characteristics:** - Single thread of control - Components communicate through procedure calls and explicit data sharing
2. Model	**Components:** Procedures and data **Connectors:** Procedure calls, data sharing
3. Suitable For	Suitable for applications where computation can be defined through hierarchical procedures, such as modular programming with clearly defined functions and data.
4. Context	Typically used in programming languages with native support for defining procedures and organizing them hierarchically, like C, Pascal, and other procedural languages.
5. Adv & Disadv	**Advantages:** - Modular design - Clear hierarchy of procedures **Disadvantages:** - Can lead to rigid structure - Hard to scale for large systems with many modules

Main Program & Subroutine

Step	Details
1. What is?	**Definition:** In this style, a main program controls the flow of execution by calling subroutines. **Characteristics:** - Single thread of control - Subroutines defined hierarchically
2. Model	**Components:** Main program, subroutines **Connectors:** Procedure calls and data sharing between the main program and subroutines
3. Suitable For	Suitable for applications that follow a clear control flow and can be divided into discrete processing steps or tasks, such as command-line utilities or small programs.
4. Context	This architecture is often used in systems where tasks can be broken down into smaller subroutines, common in procedural programming.
5. Adv & Disadv	**Advantages:** - Easy to manage and understand - Can be easily extended with more subroutines **Disadvantages:** - Can become cumbersome with many subroutines - Lack of flexibility for complex systems

Object Oriented Style

Step	Details
1. What is?	**Definition:** The Object Oriented (OO) style focuses on encapsulating data and behavior into objects, with support for inheritance, polymorphism, and dynamic binding. **Characteristics:** - Encapsulation - Polymorphism - Inheritance
2. Model	**Components:** Objects (with states and operations) **Connectors:** Procedure calls between objects
3. Suitable For	Suitable for systems that need to model complex entities and their interactions, such as GUI applications, simulations, and real-time systems.
4. Context	This style is used in object-oriented programming languages like Java, C++, and Python, where modeling real-world entities as objects is essential.
5. Adv & Disadv	**Advantages:** - Reusability through inheritance - Easy to maintain and scale **Disadvantages:** - Can result in large systems with many objects - Complex interactions between objects can be hard to manage

Layered Style

Step	Details
1. What is?	**Definition:** The Layered style organizes the system into layers, with each layer providing a distinct class of services. **Characteristics:** - Hierarchical organization - Each layer communicates only with the adjacent layers
2. Model	**Components:** Layers, each responsible for a specific class of services **Connectors:** Procedure calls between layers
3. Suitable For	Suitable for systems that require clear separation of concerns and services, such as web applications or operating systems.
4. Context	Often used in large systems where modularity and separation of concerns are critical for maintainability, such as client-server architectures or enterprise applications.
5. Adv & Disadv	**Advantages:** - High cohesion within layers - Easy to modify or replace individual layers **Disadvantages:** - Performance can be impacted by many layers - Difficult to structure cleanly with many dependencies

Client/Server (C/S) Style

Step	Details
1. What is?	**Definition:** Client/Server architecture divides the system into clients that make requests and servers that handle those requests. **Characteristics:** - Asymmetric communication - Clients depend on servers
2. Model	**Components:** Client applications and server applications **Connectors:** RPC-based network protocols
3. Suitable For	Suitable for distributed systems where data needs to be accessed by multiple clients, such as web applications and database systems.
4. Context	Commonly used in multi-user environments and applications where different entities need to share resources, like web services or enterprise systems.
5. Adv & Disadv	**Advantages:** - Scalable - Easy to add or upgrade servers **Disadvantages:** - Redundant management on each server - Difficult to find available services - Performance limitations based on server and network capacity

Data-centered style

Data-Centered Style

Step	Details
1. What is?	**Definition:** Data-Centered Architecture involves a shared data source approach for information delivery, where the data source is the central component. **Characteristics:** - Centralized data - Shared access to data by multiple components
2. Model	**Components:** Central data structure, data-processing components **Connectors:** Data-sharing mechanisms, clipboard or central repositories
3. Suitable For	Ideal for systems where multiple applications need to access a central data store, such as configurations or data exchange platforms.
4. Context	Used in scenarios like configuration management systems, such as Windows Registry or clipboard mechanisms.
5. Adv & Disadv	**Advantages:** - Centralized management of data - Easier to manage and access data **Disadvantages:** - Potential bottlenecks - Lack of direct communication between components can limit efficiency

Repository Style

Step	Details
1. What is?	**Definition:** A repository is a central place where data is stored and maintained. Components operate on this central data store. **Characteristics:** - Centralized data - Independent components interacting with the data
2. Model	**Components:** Central repository, independent components **Connectors:** Database-based or blackboard-based interactions with the central data store
3. Suitable For	Suitable for applications that need to establish and maintain a complex central body of information, such as compilers or business decision systems.
4. Context	Common in data processing systems, where information is shared and processed through a central repository.
5. Adv & Disadv	**Advantages:** - Centralized data management - Easier to maintain consistency - Supports multiple components accessing the same data **Disadvantages:** - Performance issues with large data volumes - Potential bottleneck with large number of components interacting

Blackboard Style

Step	Details
1. What is?	**Definition:** Blackboard Style is used for problems that lack a definite algorithm and require collaboration between multiple solvers. The central data structure (blackboard) is modified by independent knowledge sources. **Characteristics:** - Multiple solvers - Central blackboard to store and exchange data - Dynamic coordination between solvers
2. Model	**Components:** Blackboard (central data structure), Knowledge Sources (solvers), Controller (coordinates actions) **Connectors:** Knowledge sources interact via the blackboard, updating data incrementally
3. Suitable For	Ideal for complex, open-ended problems requiring expertise from multiple domains, such as AI applications like natural language processing, image processing, and pattern recognition.
4. Context	Used in AI systems, where the problem is decomposed into subproblems, and each solver works independently on its assigned task.
5. Adv & Disadv	**Advantages:** - Dynamic and flexible problem-solving - Easy to add new knowledge sources - Effective for complex, collaborative problem-solving **Disadvantages:** - Can be difficult to manage large numbers of solvers - Performance issues due to frequent data access on the blackboard - Complexity in ensuring correct coordination between solvers

Virtual Machine style

Virtual Machine style

Step	Details
1. What is?	**Definition:** Interpreters simulate functionality that is not native to the hardware. **Characteristics:** - Simulates non-native functionality
2. Model	**Components:** Interpreters, virtual machine **Connectors:** Simulate the functionality of non-native systems
3. Suitable For	Suitable for systems that need to run code or perform tasks in environments that are abstracted from the physical hardware.
4. Context	Virtual machines are often used for interpreting languages that are not natively supported by hardware, or for running applications that require emulation.
5. Adv & Disadv	**Advantages:** - Can simulate functionality which is not native to the hardware **Disadvantages:** - Much slower than hardware execution - Much slower than compiled systems

Interpreter Style

Step	Details
1. What is?	**Definition:** The interpreter pattern is suitable for applications in which the most appropriate language or machine for executing the solution is not directly available. **Characteristics:** - Used to define notations or languages for expressing solutions
2. Model	**Components:** One state machine (execution engine), three memories (current state, program being interpreted, current state of program) **Connectors:** Data access, procedure call
3. Suitable For	Suitable for systems that need to interpret or simulate a language or machine not natively supported, like scripting or language translation.
4. Context	Commonly used to bridge gaps between desired languages/machines and those already available in the execution environment.
5. Adv & Disadv	**Advantages:** - Can simulate non-native functionality - Can simulate "disaster" modes for testing safety-critical applications - Flexible, general-purpose tool **Disadvantages:** - Much slower than hardware execution - Slower than compiled systems - Additional layer of software needs verification

Rule-based System

Step	Details
1. What is?	**Definition:** Rule-based systems use an inference engine to match facts and rules, triggering actions when conditions are met. **Characteristics:** - Code to be executed (knowledge base) - Interpretation engine (rule interpreter) - Control state of interpreter
2. Model	**Components:** Knowledge base (code to be executed), rule interpreter (execution engine), current state of the code (working memory) **Connectors:** Rule/data selection, knowledge base interaction
3. Suitable For	Ideal for applications that need to process a set of rules to solve a problem, such as expert systems or business rule engines.
4. Context	Rule-based systems are typically used in scenarios where business logic needs to be automated and decision-making relies on predefined rules.
5. Adv & Disadv	**Advantages:** - Declarative programming: Define "what" to do, not "how" - Logic and data separation - Fast and flexible - Centralized knowledge base - Tool integration and good explanation mechanisms **Disadvantages:** - Can become inefficient with large rule sets - Complexity in managing large numbers of interacting rules - Slow in real-time applications