# CSE 470
# Software Engineering

## Software Architecture

Imran Zahid
Lecturer
Computer Science and Engineering, BRAC University

# Software Architecture

## Software Architecture

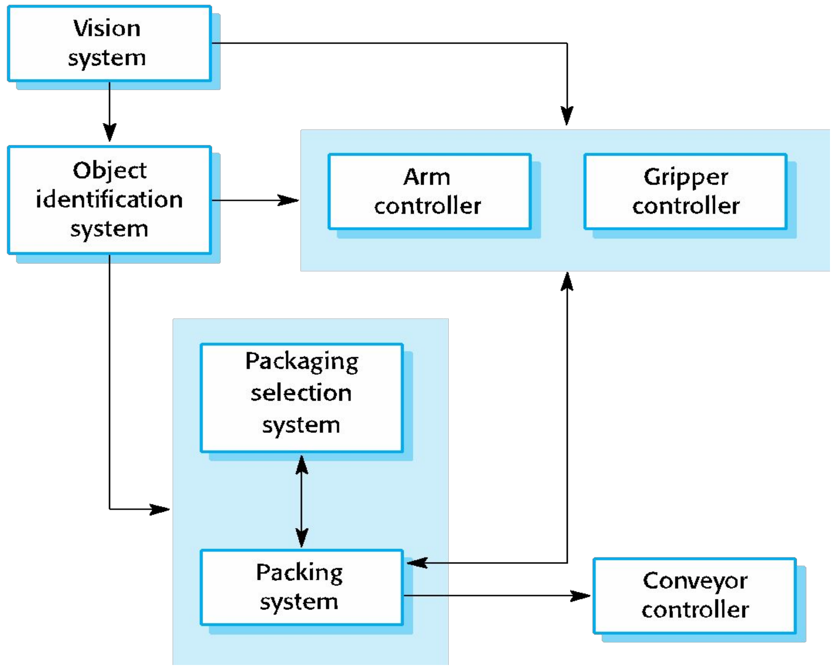
- Defines the high-level structure of a software system, setting the foundation for design and implementation.
- Outlines interactions between components and their organization within the system.
- Influences scalability, flexibility, and maintainability of the software.

## Software Architecture

- Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system.
- Software Architecture is how the defining components of a software system are organized and assembled, how they communicate each other, and the constraints the whole system is ruled by.
- The architectural model serves as a input to the development phase.

## Architectural Representation / Diagram

Simple, informal block diagrams showing entities and relationships are the most frequently used method for documenting software architectures.



# Advantages of Explicit Architecture

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
  - The architecture may be reusable across a range of systems

## Necessity of Software Architecture

- Provides a structured foundation for organized development.
- Enhances scalability and adaptability to changing requirements.
- Simplifies maintenance and debugging through clear component organization.
- Mitigates risks by identifying challenges early in the design process.

## Layered Architecture

## Lets recall Monolithic Software

- The end product comes at end of the process model
- There is no separation of concern of different software components.
- All code may be written in a single file with html, sql queries, logic checking etc.

The problem?

- No separation between components
- Changing a component affects other components. For example – What if I want to change the UI from JavaScript to Angular ?

## Layered Architecture

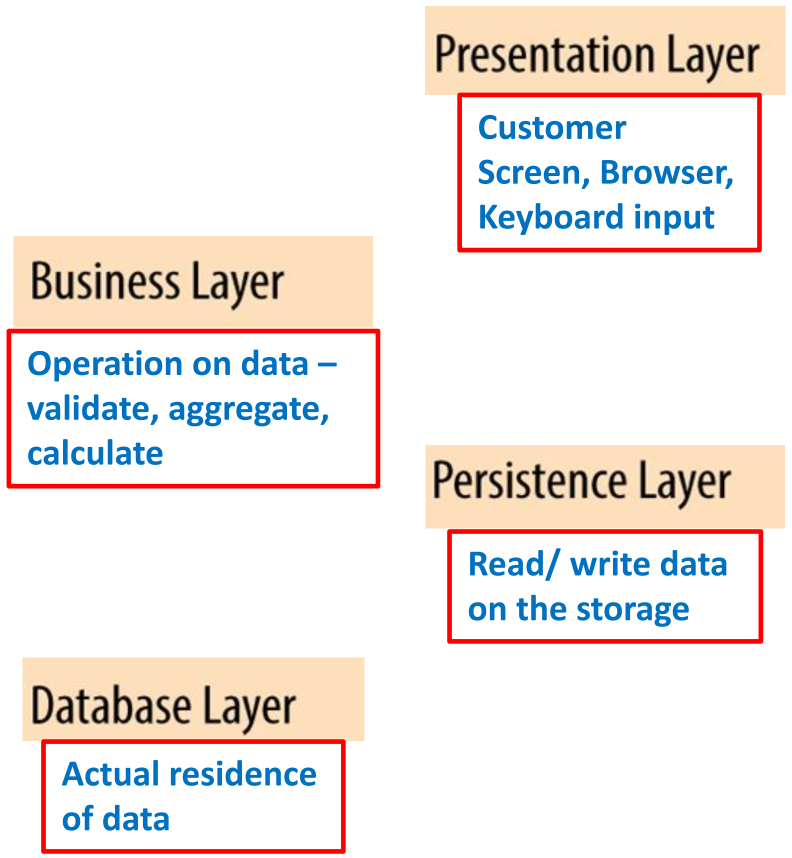
- One of the most common architecture pattern is the layered architecture pattern, otherwise known as the n-tier architecture pattern.
- This pattern is the de facto standard for most Java EE applications and therefore is widely known by most architects, designers, and developers.
- The layered architecture pattern closely matches the traditional IT communication and organizational structures found in most companies, making it a natural choice.

## Layered Architecture

- Organises the system into a set of layers (or abstract machines) each of which provide a set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.

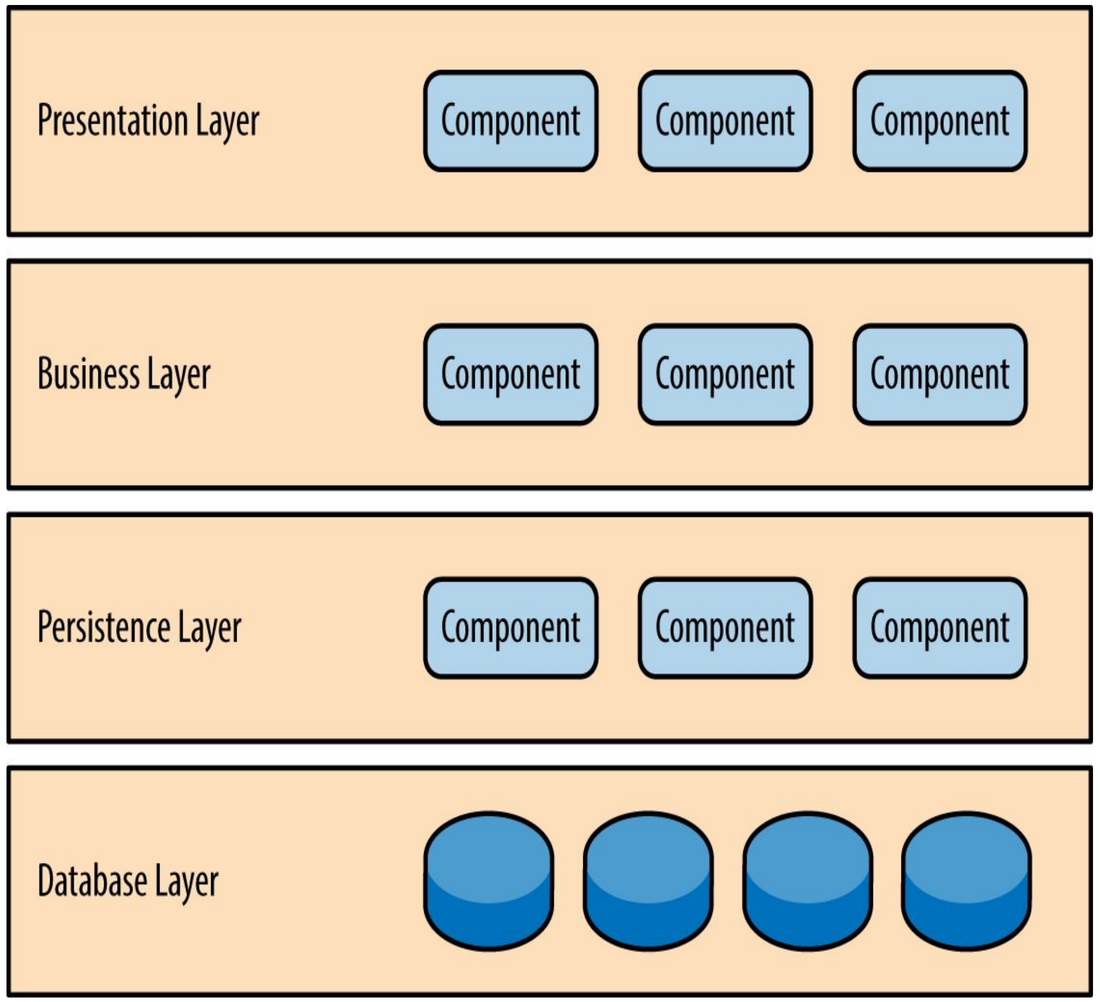
# Layered Architecture

- Although it does not specify the number and types of layers that must exist, most layered architectures consist of four standard layers
  - Presentation
  - Business
  - Persistence
  - Database


Presentation Layer
Customer Screen, Browser, Keyboard input
Business Layer
Operation on data – validate, aggregate, calculate
Persistence Layer
Read/ write data on the storage
Database Layer
Actual residence of data

# Layered Architecture

- Organizes the system into layers with related functionality associated with each layer.
- A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system.



## Layered Architecture

- Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request.
- For example, the presentation layer doesn’t need to know or worry about how to get customer data; it only needs to display that information on a screen in particular format.

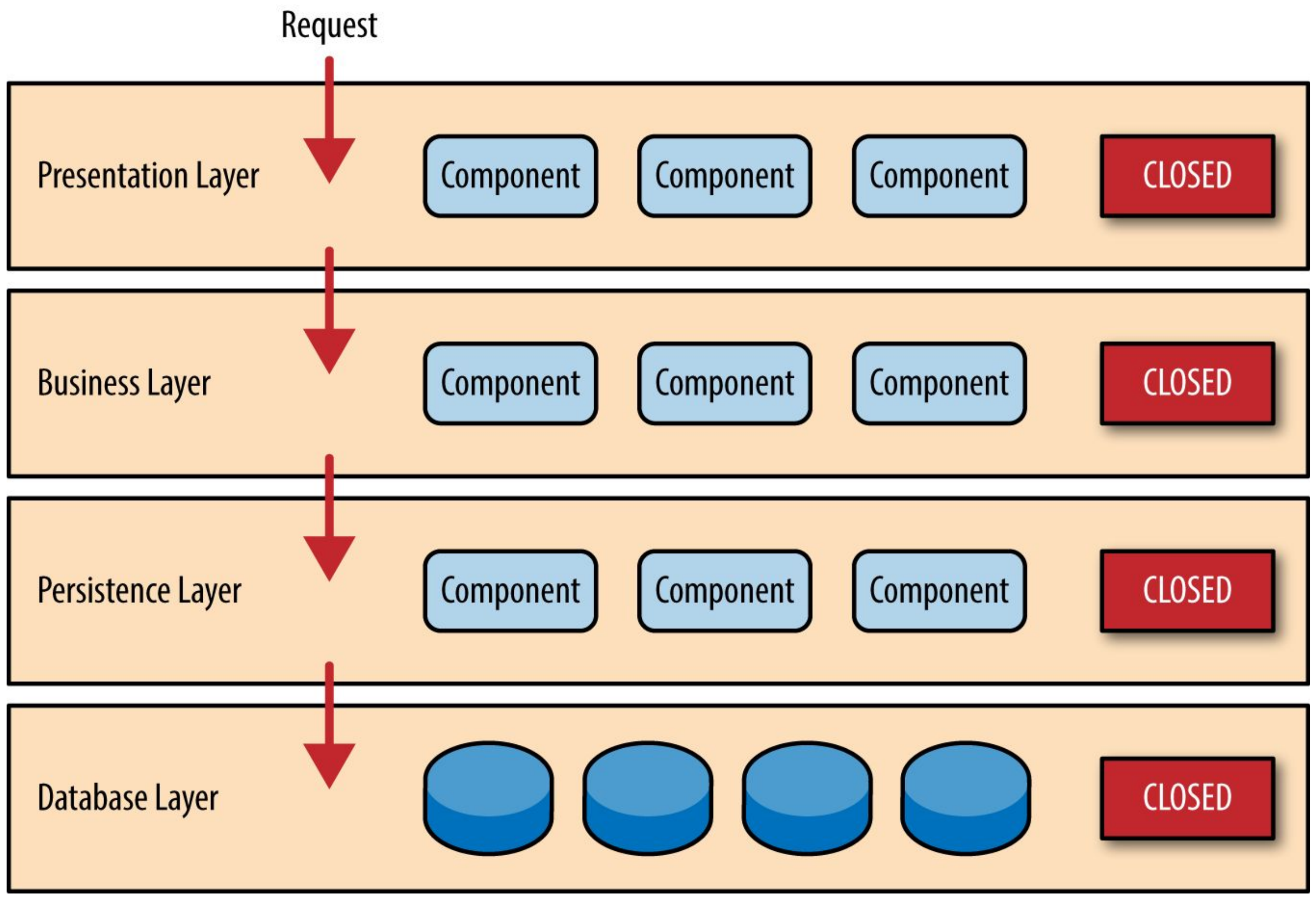
## Layered Architecture - Open and Closed

- Layers can be open or closed.
- A closed layer means that as a request moves from layer to layer, it must go through the layer right below it to get to the next layer below that one.
- For example, a request originating from the presentation layer must first go through the business layer and then to the persistence layer before finally hitting the database layer.
- A closed layer can only be accessed by the layer above.

## Layered Architecture - Open and Closed

- Layers of isolation: Changes in one layer don't impact components in other layers; changes are isolated within that layer.
- Each layer is independent, with little to no knowledge of the inner workings of other layers in the architecture.

# Layered Architecture - Open and Closed



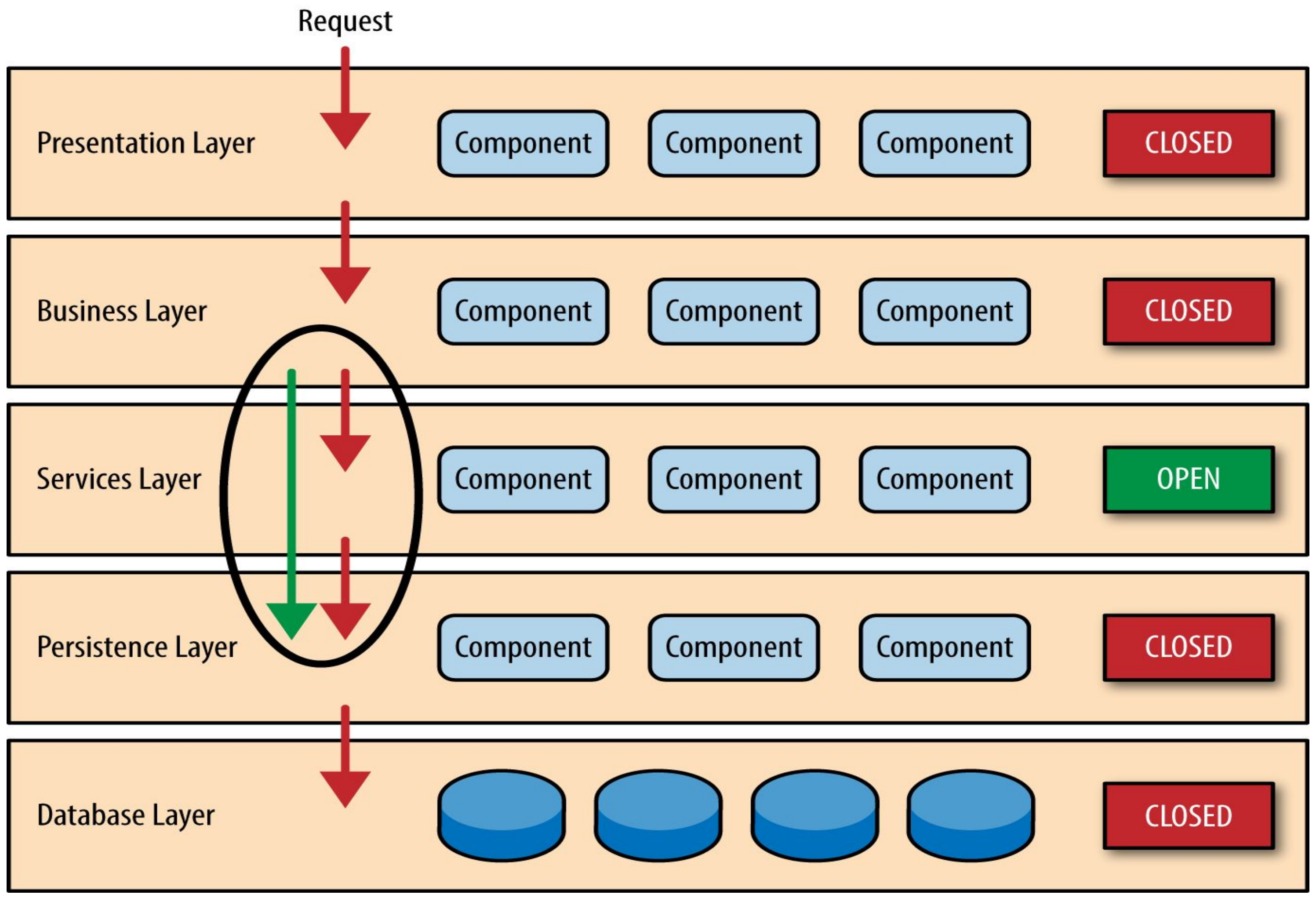
# Layered Architecture - Open and Closed

- Suppose, we add a shared-services layer containing common service components (e.g., data and string utility classes, auditing, logging).
- We position services layer below the business layer, making it inaccessible from the presentation layer.
- Problem: Business layer must go through the services layer to access the persistence layer, which is inefficient.
- Solution: Use open layers within the architecture to allow direct access where needed.

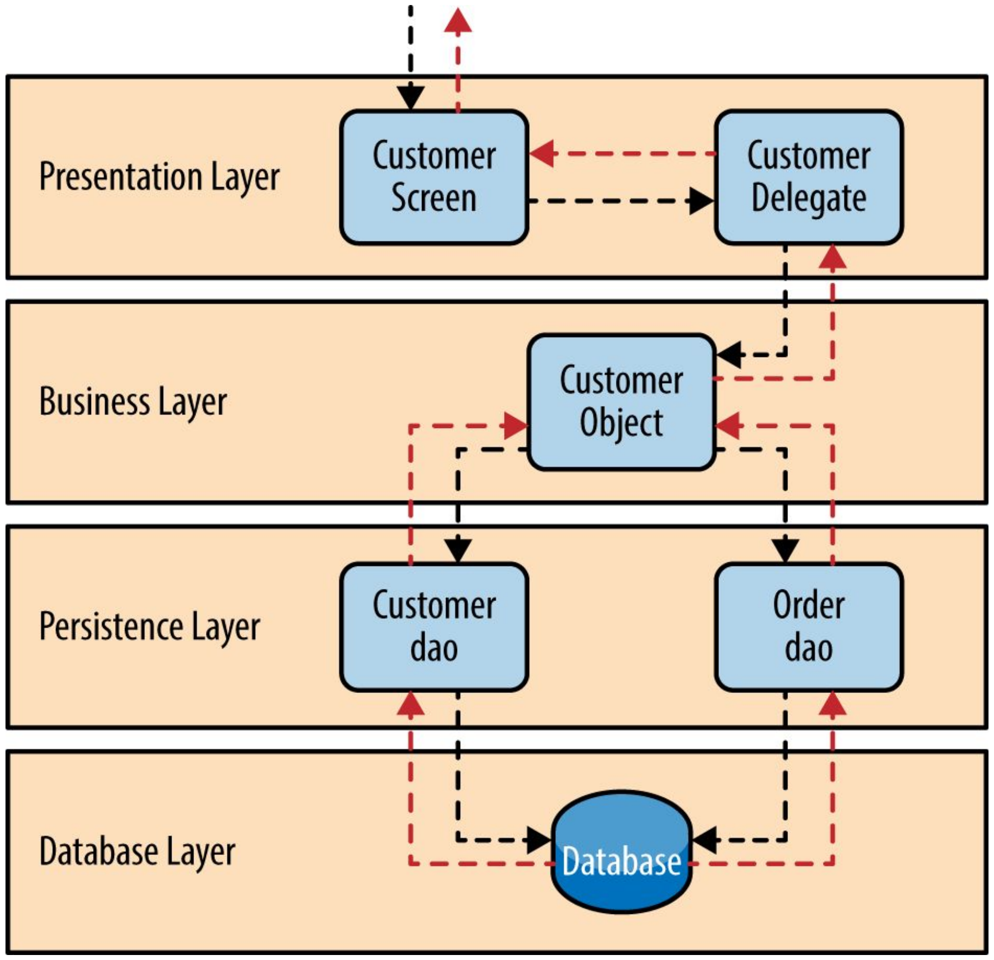
## Layered Architecture - Open and Closed

- An open layer can be bypassed by upper layers.
- Too many open layers may affect the actual essence of layered architecture.
- The services layer in this case is marked as open,  meaning requests are allowed to bypass this open layer and go directly to the layer below it.

# Layered Architecture - Open and Closed



# Example


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

# Layered Architecture - When to Use?

- Used when building new facilities on top of existing systems.
- When the development is spread across several teams with each team responsibility for a layer of functionality
- When there is a requirement for multi-level security.

## Layered Architecture - Advantages

- Allows easy replacement or addition of entire layers so long as the interface is maintained.
- Testing is easy as components are isolated.

## Layered Architecture - Disadvantages

- In practice, providing a clean separation between layers is often difficult. A high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it.
- Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.
- A change in any layer still requires to restart the application.

## Model-View-Controller Architecture

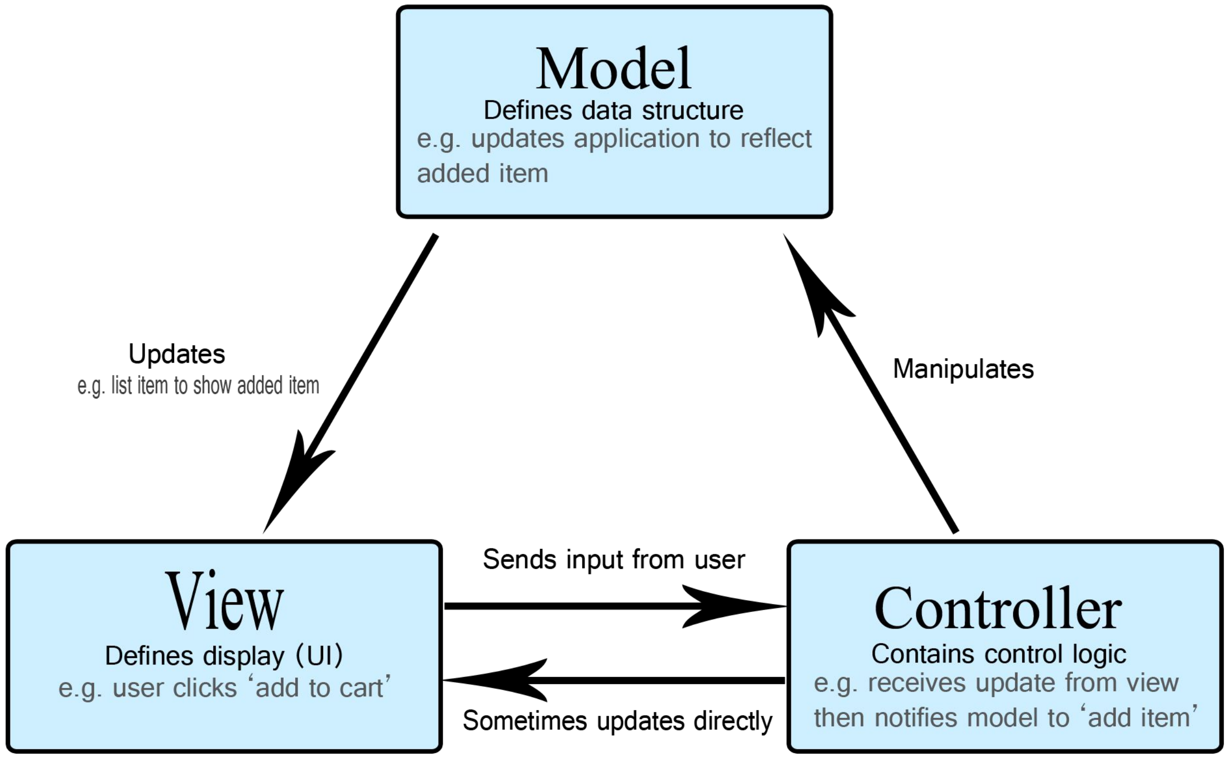
## MVC Architecture Pattern

- Stands for Model-View-Controller.
- Divides an application into three interconnected components: Model, View, and Controller.
- Separates internal representations of information from the ways the information is presented and accepted by the user.
- Promotes organized programming and maintainability.

## MVC System Components

- Model: Manages the application data and logic.
- View: Displays data and interacts with the user.
- Controller: Handles user input and updates the Model and View.

# MVC System Components


Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

## MVC Architecture - When to Use?

- Suitable for applications requiring a clean separation of concerns.
- When multiple developers are working simultaneously on different parts of the application.
- Ideal for applications requiring reusable components.

# MVC Architecture - Advantages

- Promotes parallel development as Model, View, and Controller can be worked on independently.
- Enhances code organization and separation of concerns.
- Easy to update the application's user interface without affecting business logic.
- In MVC, the components are more independent and designed to adapt to changes in one part of the system without making the entire system unusable, which is not as easily achievable in a tightly layered architecture.

## MVC Architecture - Disadvantages

- Can introduce complexity, especially for smaller projects.
- Requires careful coordination between the Model, View, and Controller components.
- Debugging can be more challenging due to indirect interaction between components.
- Excessive reliance on controllers can lead to bloated code.

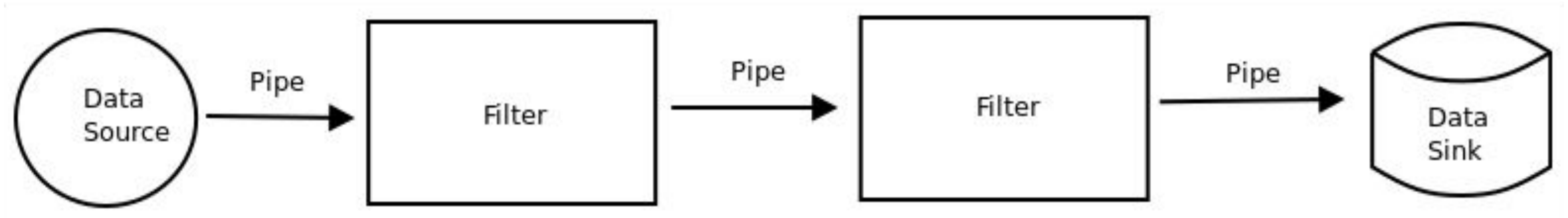
## Pipe-Filter Architecture

## Pipe-Filter Architecture

- This pattern can be used to structure systems which produce and process a stream of data.
- Each processing step is enclosed within a filter component.
- Data to be processed is passed through pipes. These pipes can be used for buffering or for synchronization purposes.

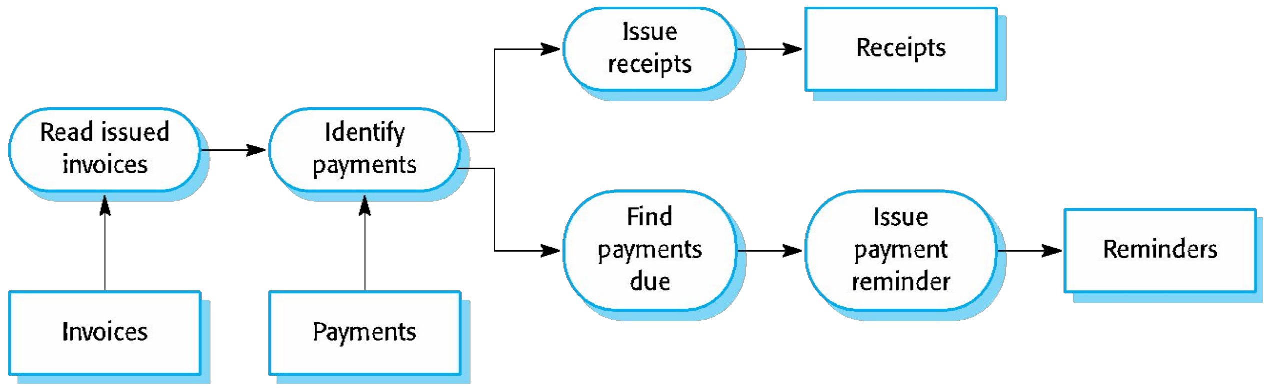
# Pipe-Filter Architecture

- The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.



## Examples

- Compilers: The consecutive filters perform lexical analysis, parsing, semantic analysis, and code generation.
- Workflows in bioinformatics.
- Figure beside is an example of the pipe and filter architecture used in a payments system



## Pipe-Filter Architecture - When to Use?

- Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.

## Pipe-Filter Architecture - Advantages

- Easy to understand and supports transformation reuse.
- Workflow style matches the structure of many business processes.
- Evolution by adding transformations is straightforward.
- Can be implemented as either a sequential or concurrent system.

## Pipe-Filter Architecture - Disadvantages

The format for data transfer has to be agreed upon between communicating transformations.

Each transformation must parse its input and unparse its output to the agreed form.

This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.

# Thank you