UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SOFWARE DESIGN & ARCHITECTURE

Chapter 4 - Styles in Architecture

Prepared By: Dr Muhammad Khatibsyarbini

Innovating Solutions

ARCHITECTURAL STYLE vs PATTERN

ARCHITECTURAL STYLE

•**Definition:** An architectural style defines a **high-level structure** for organizing software components and their interactions. It provides a general blueprint or strategy for designing the system.

•**Focus: System-wide organization** — how components interact with each other and with the environment.

•**Examples:**

- Client-Server
- Layered Architecture
- Microservices
- Event-Driven Architecture

Usage:

An architectural style answers questions like:

•Should components be tightly or loosely coupled?

•Should the system be centralized or distributed?

ARCHITECTURAL PATTERN

•**Definition:** An architectural pattern offers **detailed solutions** to specific, recurring design problems within a certain context. Patterns work **within or across architectural styles** to refine and implement the system design.

•**Focus: Solution to a design problem** within a structure defined by an architectural style.

•**Examples:**

- Facade
- Factory
- Singleton
- Observer

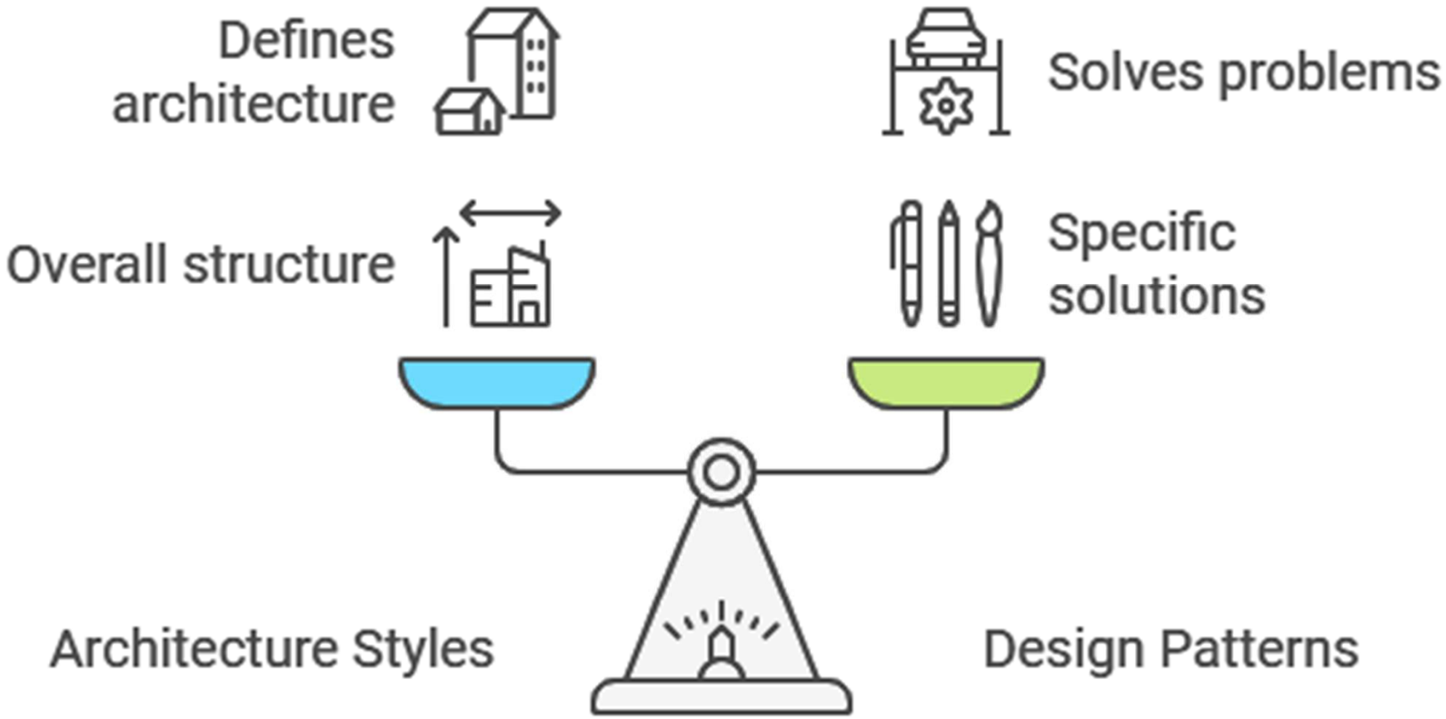
Usage:

Architectural patterns answer questions like:

•How should data flow between different modules ?

•How can we ensure a single instance of an object?

ARCHITECTURAL STYLE vs PATTERN



Styles define structure; patterns provide solutions.

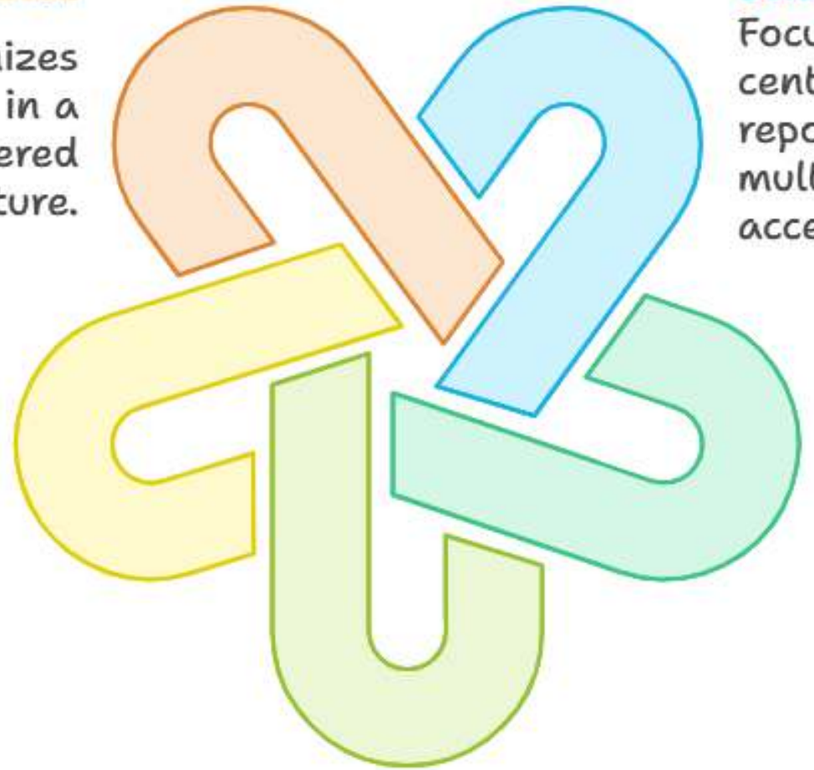
OVERVIEW SYSTEM TYPE

Hierarchical

Organizes components in a layered or tiered structure.

Data-Centered

Focuses on a central data repository with multiple clients accessing it.



Interactive

Centers on user interaction and real-time feedback.

Data Flow

Emphasizes the movement of data between processes or systems.

Distributed

Involves multiple interconnected systems working together.

SYSTEM TYPES

The choice of applying architectural style depend on the type of system, requirements, and desired quality attributes.
These characteristics help guide the choice of selecting one particular style over another.

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.

SYSTEM TYPES AND IT'S ARCHITECTURAL STYLES

System Type	Suitable Architectural Styles
Data Centred	Blackboard Repository
Data Flow	Pipe-and-Filter Blockchain
Distributed	Client-Server Microservices Blockchain
Interactive	Layered Architecture Microservices Model-View-Controller (MVC)
Hierarchical	Layered Architecture Component-Based Architecture

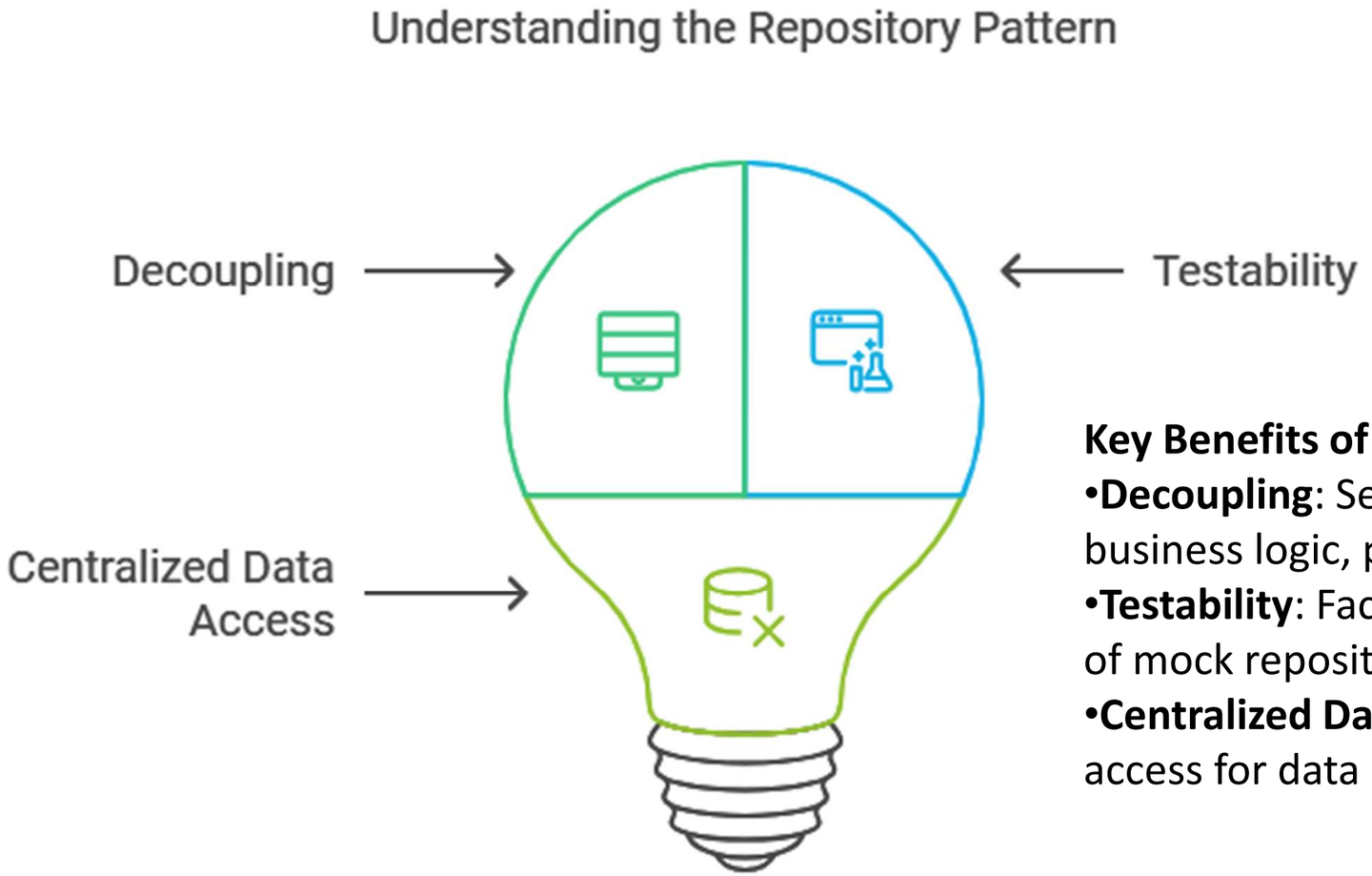
DATA CENTRE & REPOSITORY STYLE

Data-centered systems are systems primarily decomposed around a main central repository of data. These include:

- Data management component
  - The data management component controls, provides, and manages access to the system's data.
- Worker components
  - Worker components execute operations and perform work based on the data.

Quality	Description
Modifiability	Agents are compartmentalized and independent from each other; therefore, it is easy to add or remove agents to fit new systems.
Reusability	Specialized components can be reused easily in other applications.
Maintainability	Allows for separation of concerns and independence of the knowledge based agents; therefore, maintaining existing components becomes easier.

REPOSITORY STYLE



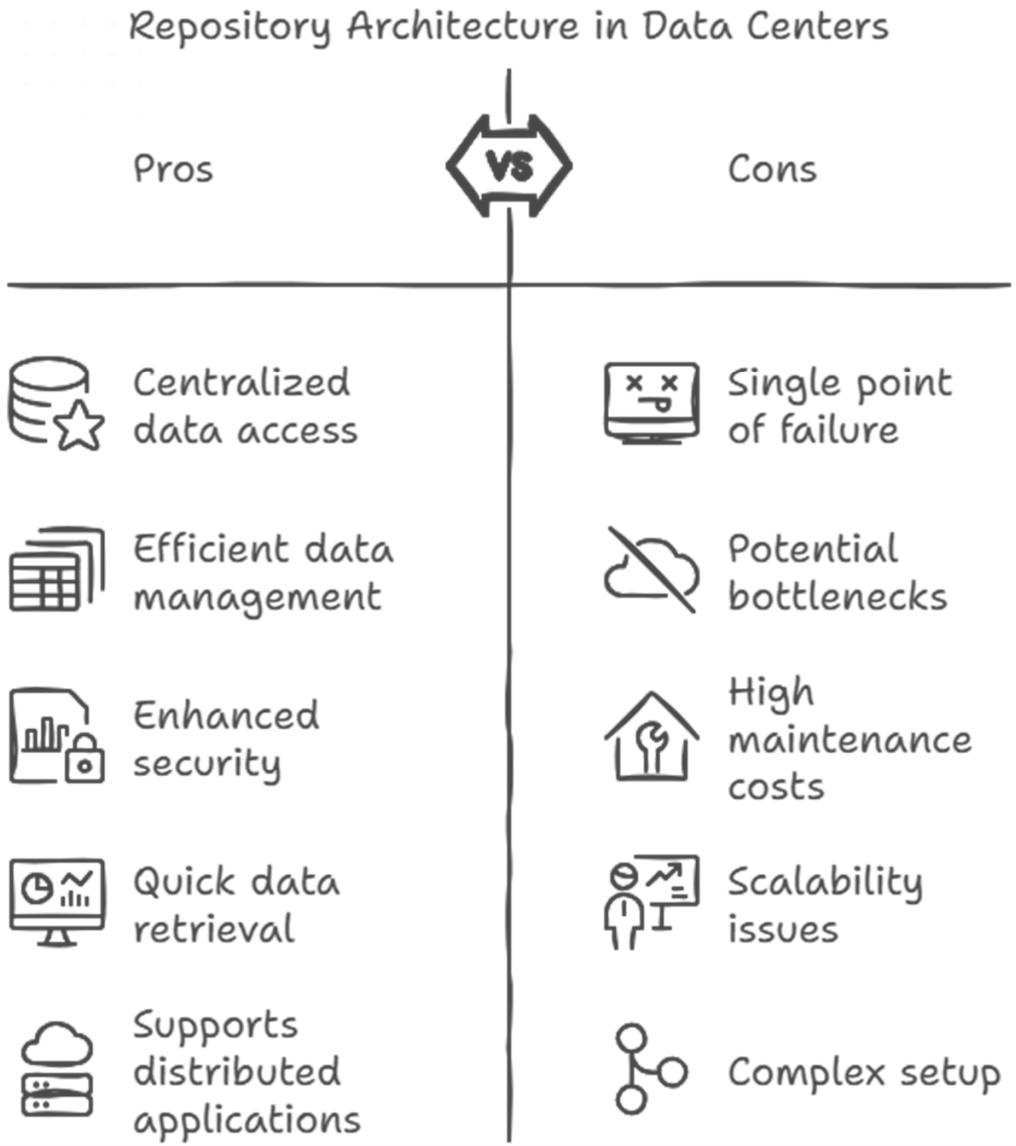
Key Benefits of the Repository Pattern

- **Decoupling**: Separates the data access logic from the business logic, promoting cleaner code.
- **Testability**: Facilitates unit testing by allowing the use of mock repositories.
- **Centralized Data Access**: Provides a single point of access for data operations, simplifying maintenance.

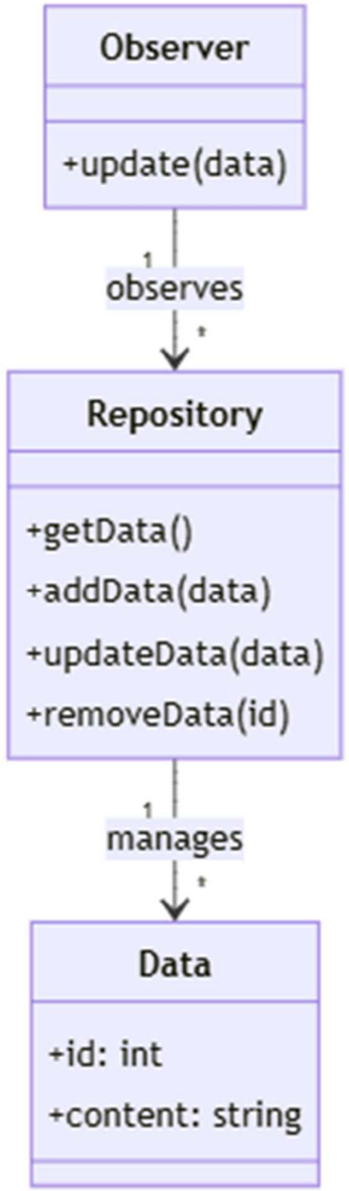
REPOSITORY STYLE

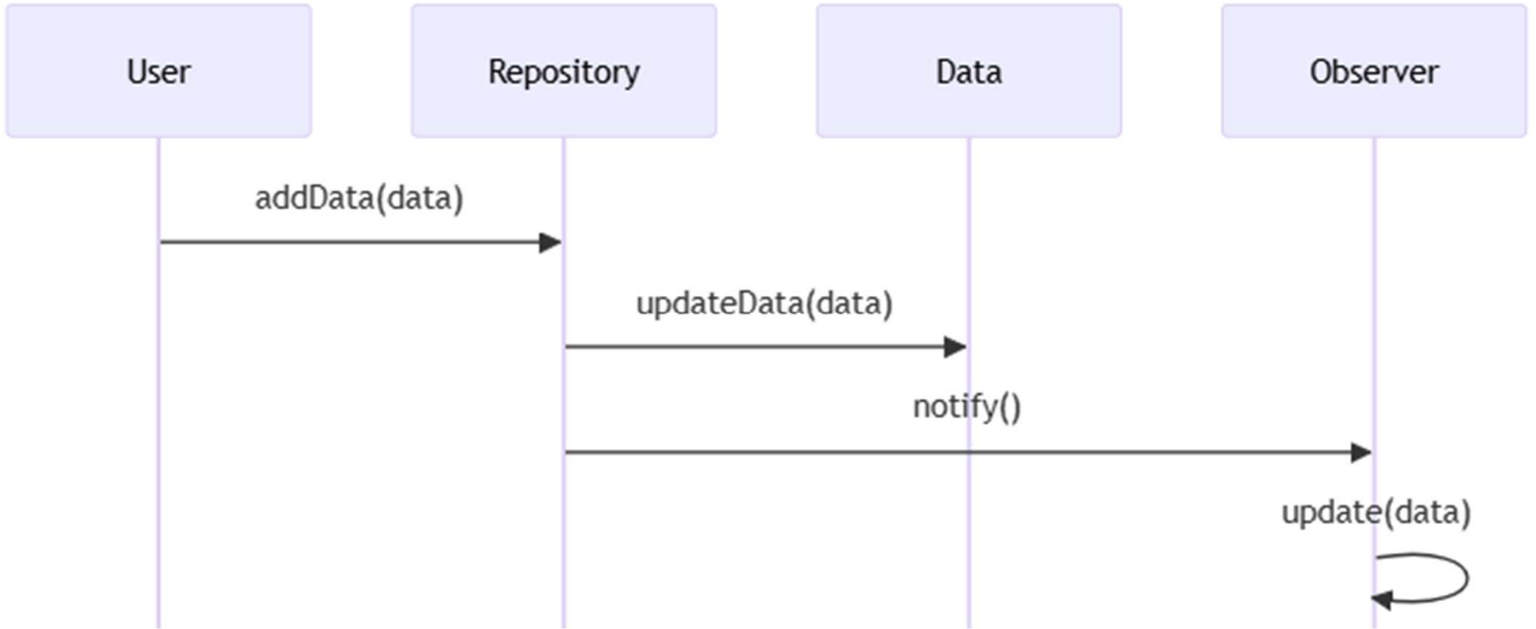
Data center systems store, manage, and distribute vast amounts of data and resources. Such systems often centralize data access and provide a shared database or repository, used by various applications or services

Data centers rely on the repository architecture to manage and provide centralized data access. This setup allows efficient data management, security, and quick data retrieval for distributed applications within the data center.



REPOSITORY STYLE





DISCLAIMER THE DIAGRAM DRAWN MAY NOT COMPLETELY CORRECT.
JUST TO ILLUSTRATE THE DIAGRAM STRUCTURE FOR THE ARCHITECTURAL STYLE.

DATA FLOW SYSTEM & BLOCKCHAIN STYLE

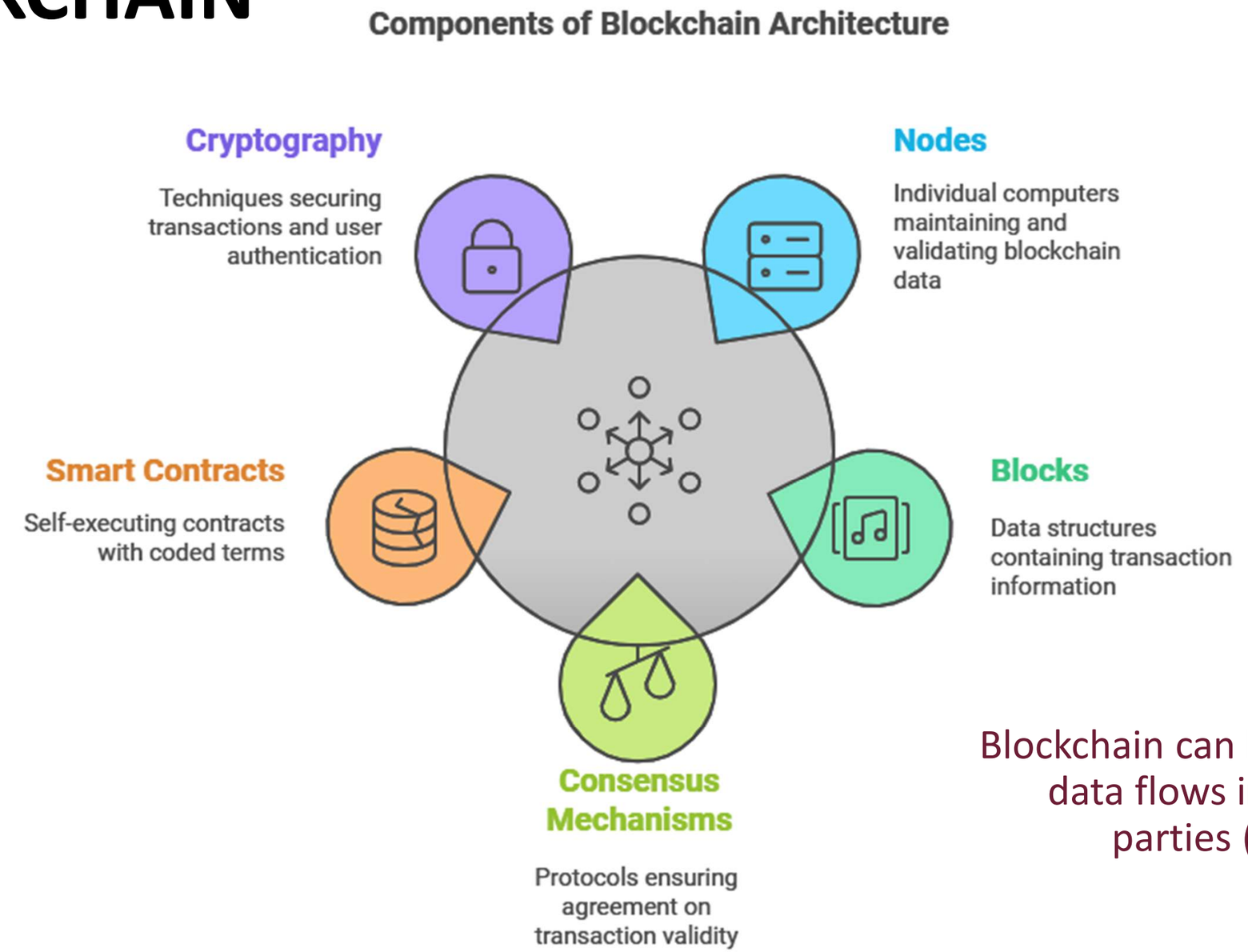
- Data flow systems are decomposed around the central theme of transporting data (or data streams) and transforming the data along the way to meet application-specific requirements.
  - Typical responsibilities found in components of data-flow systems include:
    - Worker components, those that perform work on data
    - Transport components, those that transporting data

- Worker components abstract data transformations and processing that need to take place before forwarding data streams in the system, e.g.,
  - Encryption and decryption
  - Compression and decompression
  - Changing data format, e.g. ,from binary to XML, from raw data to information, etc.
  - Enhancing, modifying, storing, etc. of the data

- Transport components abstract the management and control of the data transport mechanisms, which could include:
  - Inter-process communication
    - Sockets, serial, pipes, etc.
  - Intra-process communication
    - Direct function call, etc.

- An example of an architectural style for data flow systems is the *Blockchain*.

BLOCKCHAIN STYLE



Blockchain can handle transactional data flows in real-time between parties (e.g., cryptocurrency transactions).

BLOCKCHAIN STYLE

Blockchain Architecture

Immutability

Guarantees that once data is recorded, it cannot be altered.

Security

Protects data from unauthorized access and breaches.

Transparency

Allows all participants to see the data flow and changes.

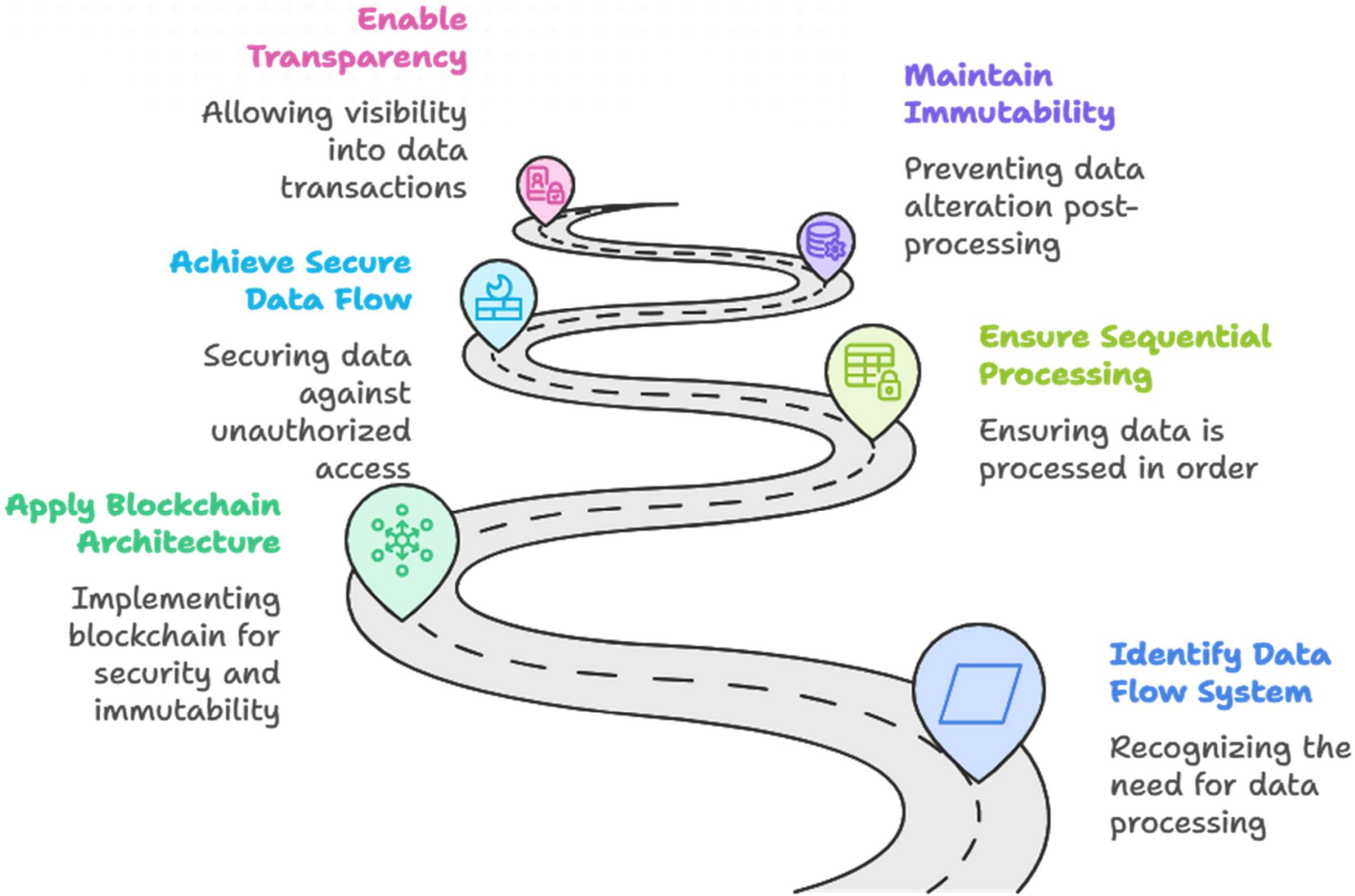
Sequential Processing

Ensures data is processed in a specific, ordered sequence.

Decentralization

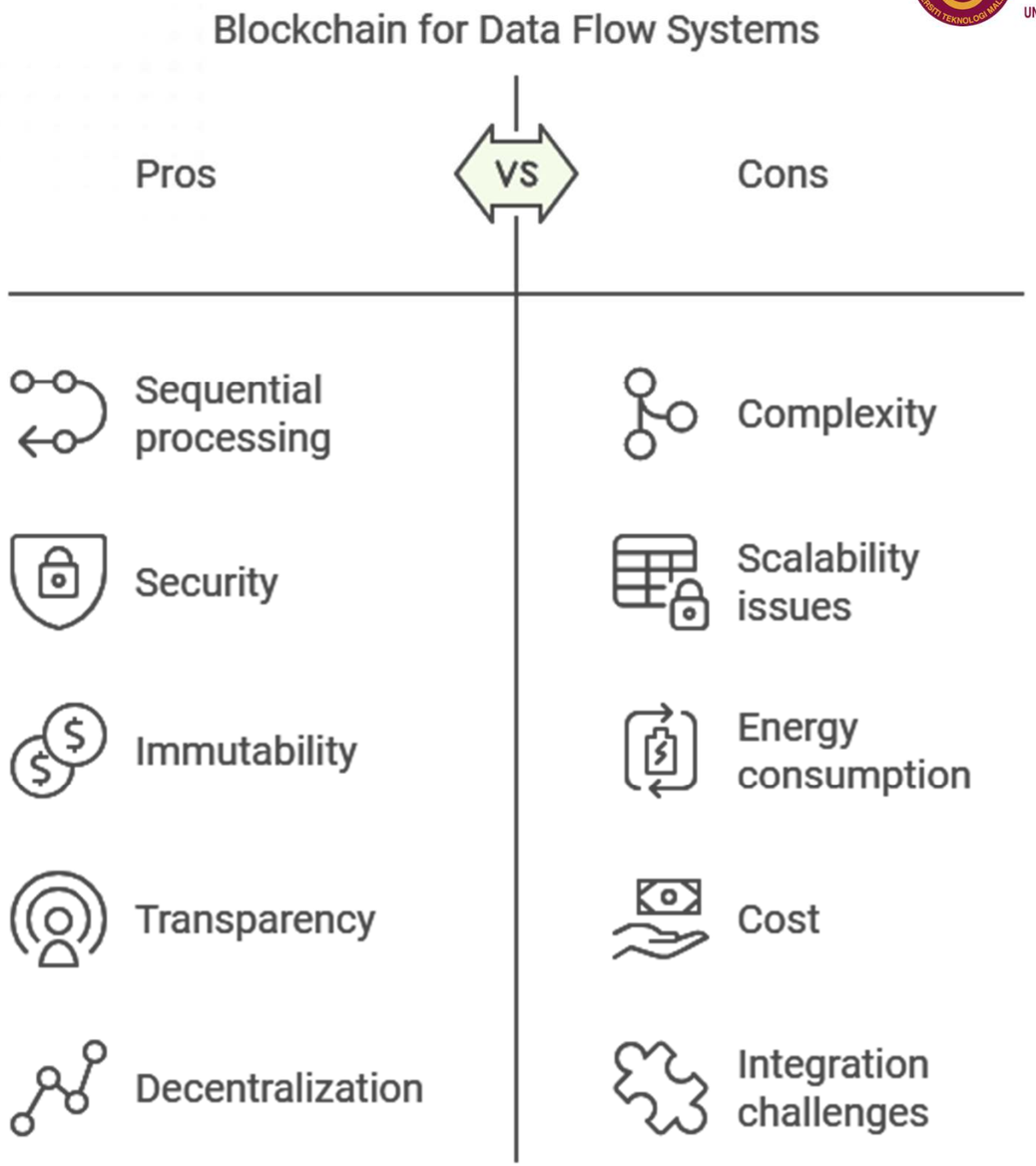
Distributes data across a network to avoid central control.

Data Flow System and Blockchain Integration

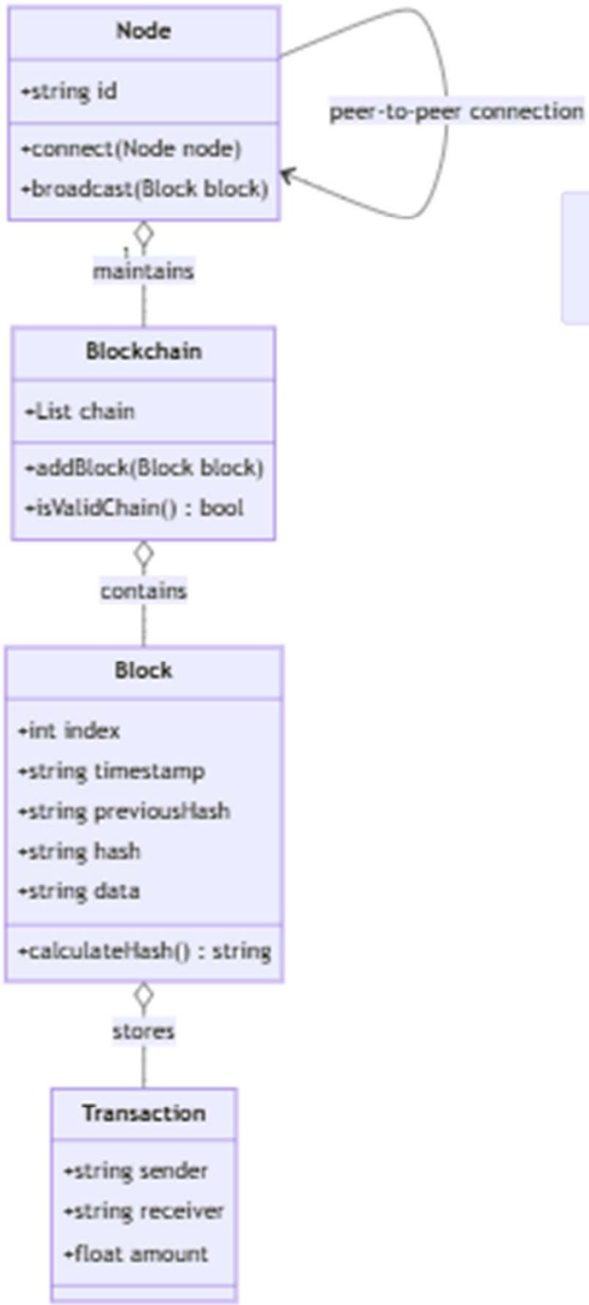


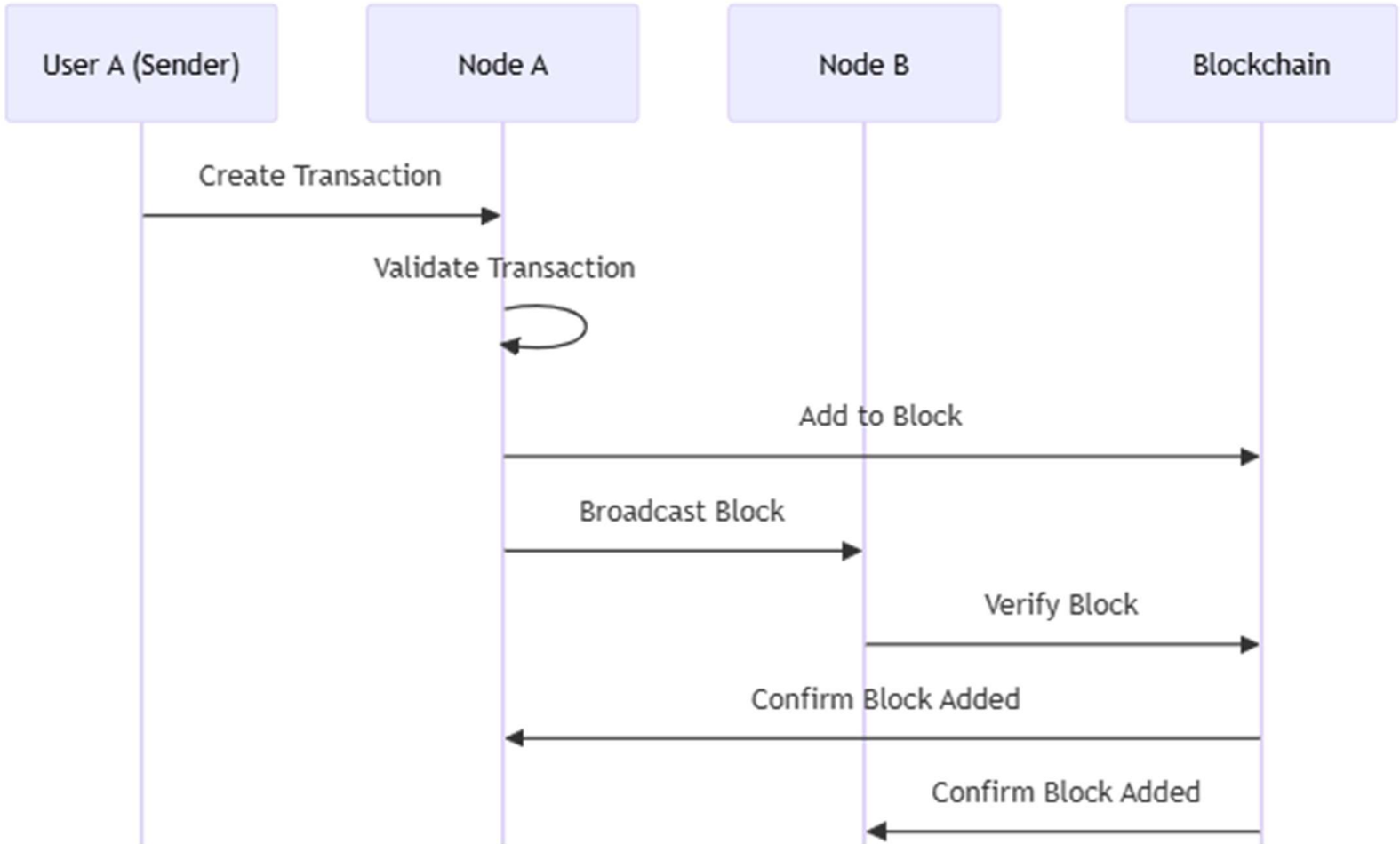
Data flow systems focus on processing data as it flows through different stages or processing units. They emphasize the continuous, orderly flow of data, often in a pipeline structure.

Blockchain architecture suits data flow systems where data must be processed sequentially, securely, and immutably. The blockchain enables a transparent, tamper-resistant, and verifiable flow of data through a decentralized network.



BLOCKCHAIN STYLE





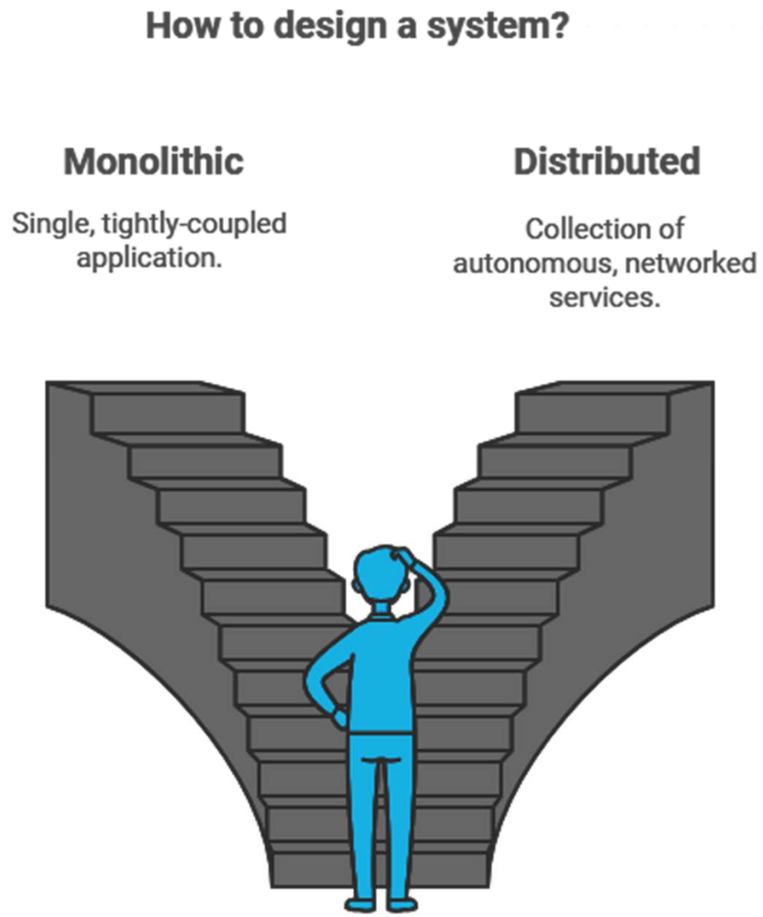
***DISCLAIMER* THE DIAGRAM DRAWN MAY NOT COMPLETELY CORRECT.**
JUST TO ILLUSTRATE THE DIAGRAM STRUCTURE FOR THE ARCHITECTURAL STYLE.

DISTRIBUTED SYSTEM

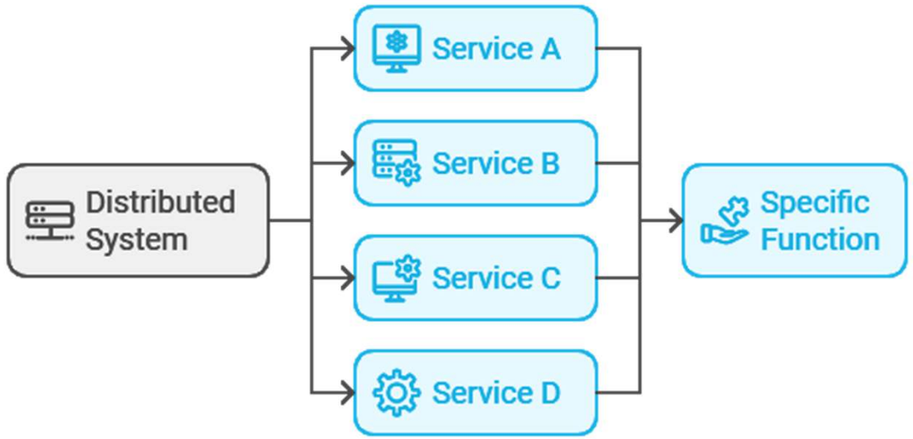
- Distributed systems are decomposed into multiple processes that (typically) collaborate through the network.
  - ✓ These systems are ubiquitous in today's modern systems thanks to wireless, mobile, and internet technology.
    - In some distributed systems, one or more distributed processes perform work on behalf of client users and provide a bridge to some server computer, typically located remotely and performing work delegated to it by the client part of the system.
    - Other distributed systems may be composed of peer nodes, each with similar capabilities and collaborating together to provide enhanced services, such as music-sharing distributed applications.
  - ✓ These types of distributed systems are easy to spot, since their deployment architecture entails multiple physical nodes.
  - ✓ However, with the advent of multi-core processors, distributed architectures are also relevant to software that executes on a single node with multiprocessor capability.

- Some examples of distributed systems include:
  - ✓ Internet systems, web services, file- or music-sharing systems, high-performance systems, etc.

DISTRIBUTED SYSTEM & MICROSERVICES



Distributed systems are collections of autonomous computers that communicate through a network to achieve a shared goal. Each component, or service, performs a specific function and is independently deployable.



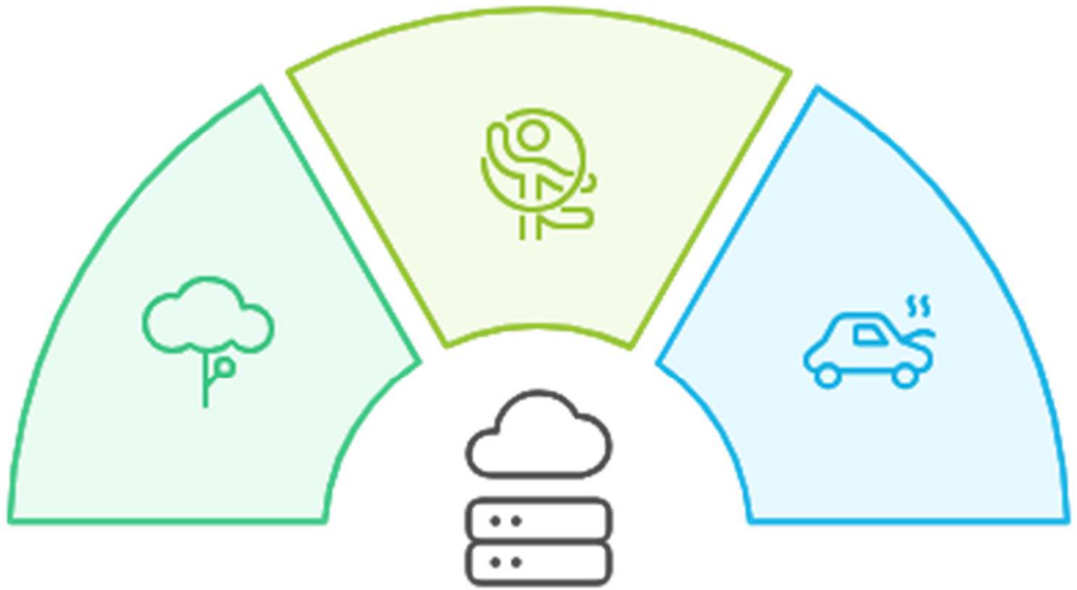
Microservices Architecture

Flexibility

The capacity to adapt and change services without affecting the entire system.

Scalability

The ability to efficiently handle increasing loads by scaling services independently.



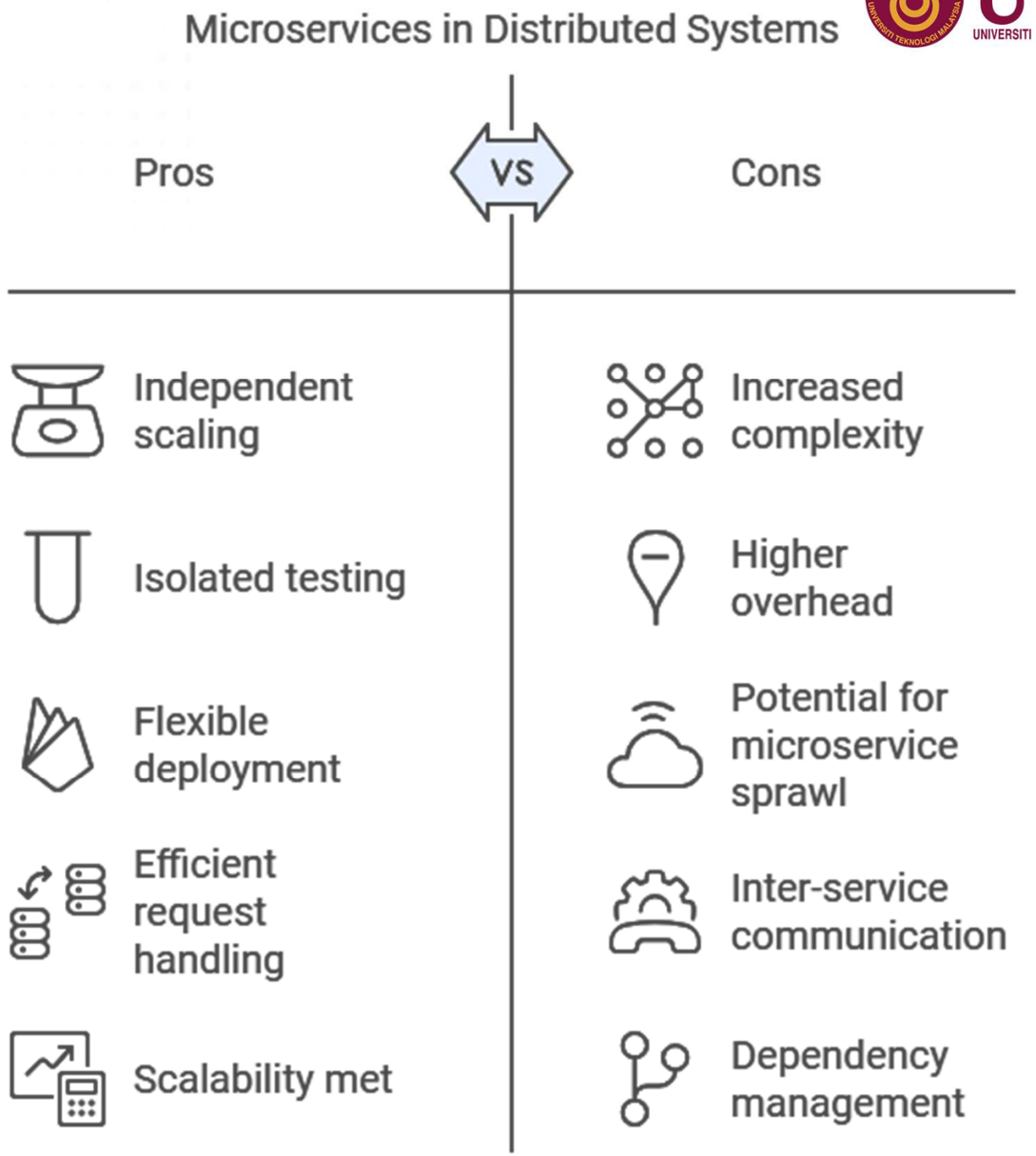
Maintenance

The ease of updating and maintaining services individually without system-wide disruptions.

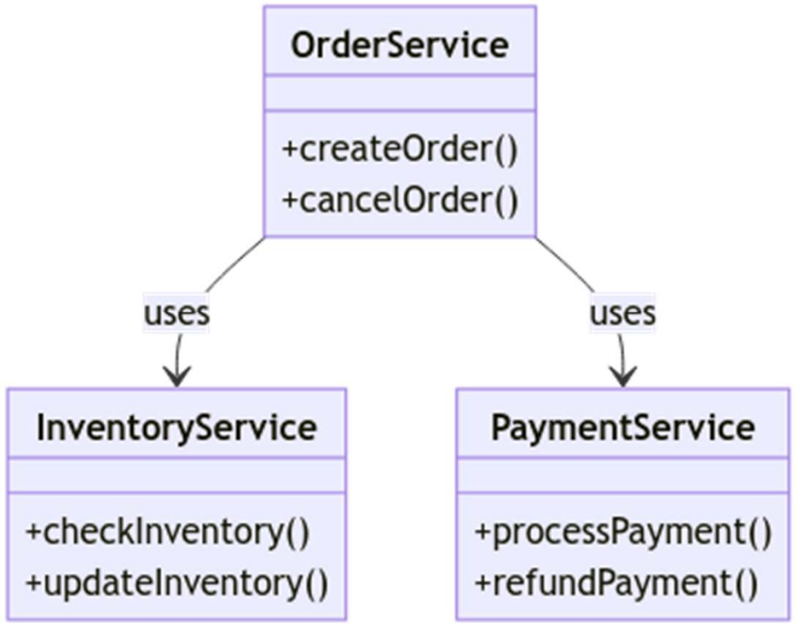
DISTRIBUTED SYSTEM & MICROSERVICES

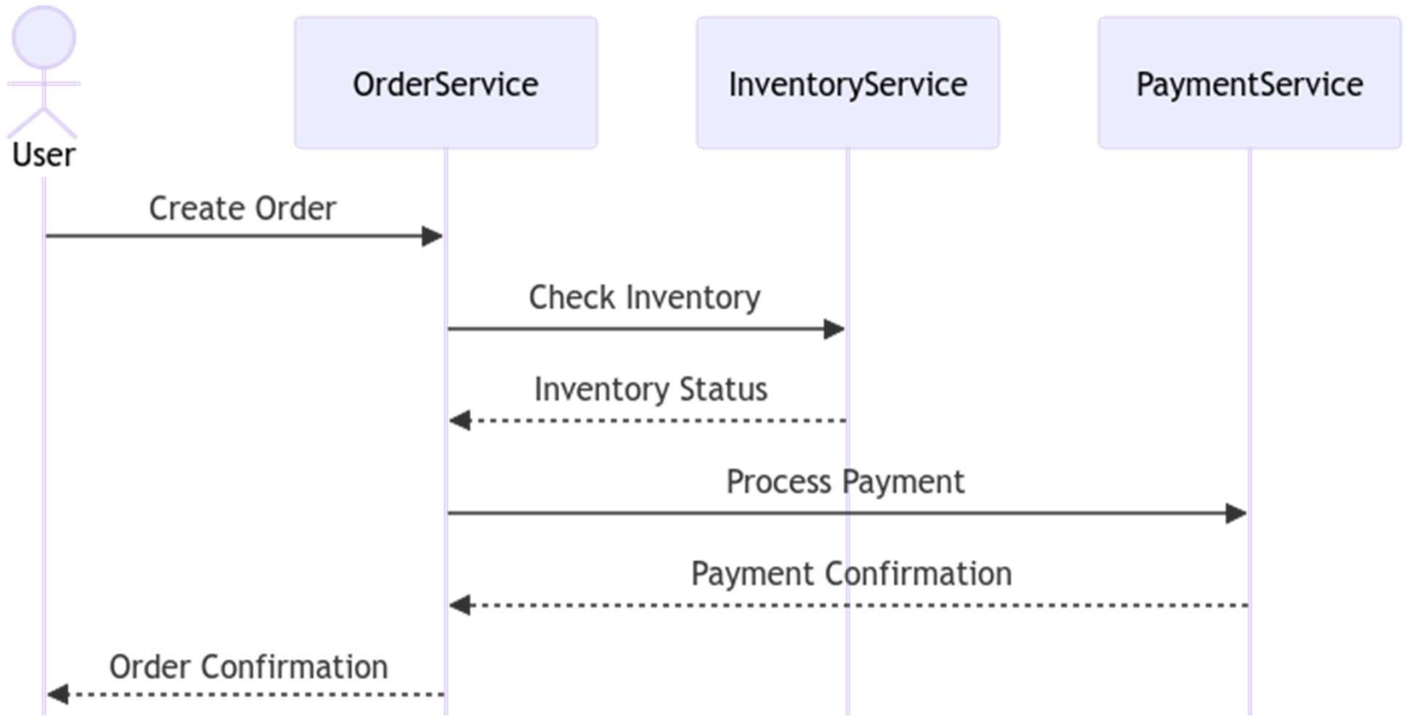
Relation between Distributed System and Microservices Style:

Distributed systems benefit from the microservices style because microservices can be independently scaled, tested, and deployed. This allows distributed systems to handle large volumes of requests efficiently and meet scalability demands.



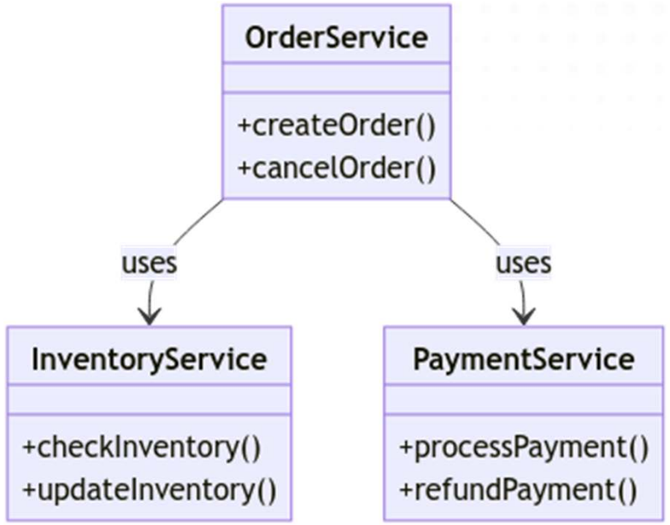
MICROSERVICES





***DISCLAIMER* THE DIAGRAM DRAWN MAY NOT COMPLETELY CORRECT.**
JUST TO ILLUSTRATE THE DIAGRAM STRUCTURE FOR THE ARCHITECTURAL STYLE.

MICROSERVICES




```
// OrderService.java
public class OrderService {
    public String createOrder() {
        InventoryService inventoryService = new InventoryService();
        PaymentService paymentService = new PaymentService();

        if (inventoryService.checkInventory()) {
            paymentService.processPayment();
            return "Order Confirmed";
        }
        return "Order Failed";
    }
}
```

```
// InventoryService.java
public class InventoryService {
    public boolean checkInventory() {
        // Logic to check inventory
        return true;
    }

    public void updateInventory() {
        // Logic to update inventory
    }
}
```

```
// PaymentService.java
public class PaymentService {
    public void processPayment() {
        // Logic to process payment
        System.out.println("Payment Processed");
    }

    public void refundPayment() {
        // Logic to refund payment
    }
}
```

```
// Main.java
public class Main {
    public static void main(String[] args) {
        OrderService orderService = new OrderService();
        System.out.println(orderService.createOrder());
    }
}
```

INTERACTIVE SYSTEMS & MVC STYLE

- Interactive systems support user interactions, typically through user interfaces.
  - ✓ When designing these systems, two main quality attributes are of interest:
    - Usability
    - Modifiability

- The mainstream architectural pattern employed in most interactive systems is the Model-View-Controller (MVC).

- The MVC pattern is used in interactive applications that require flexible incorporation of human-computer interfaces. With the MVC, systems are decomposed into three main types of components:

Component	Description
Model	Component that represents the system's core, including its major processing capabilities and data.
View	Component that represents the output representation of the system (e.g., graphical output or console-based).
Controller	Component (associated with a view) that handles user inputs.

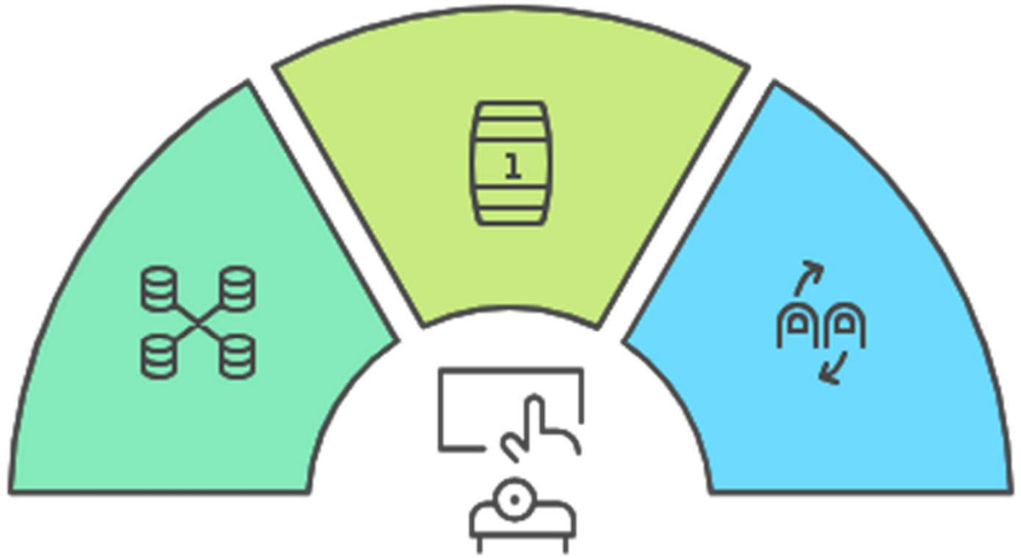
MODEL VIEW CONTROLLER

View Component

Displays the user interface and visual elements for interaction.

Model Component

Manages data logic and business rules within the system.



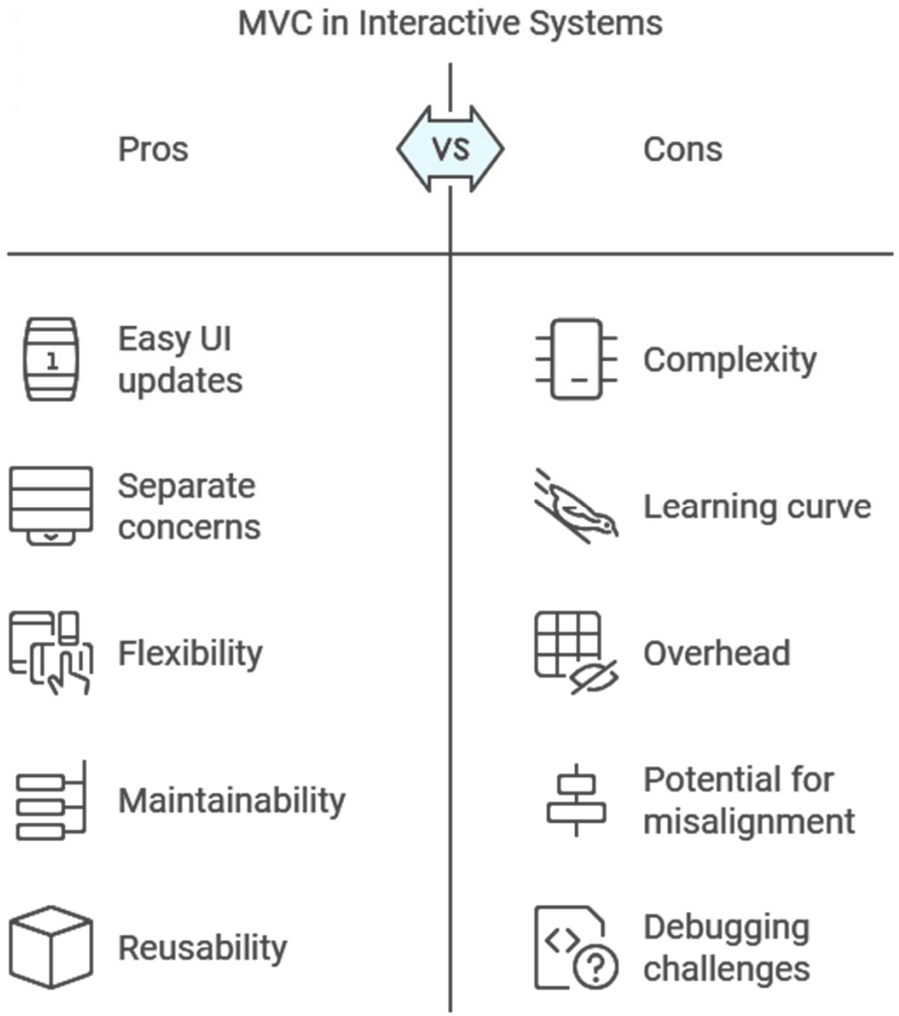
Controller Component

Handles user inputs and directs them to appropriate components.

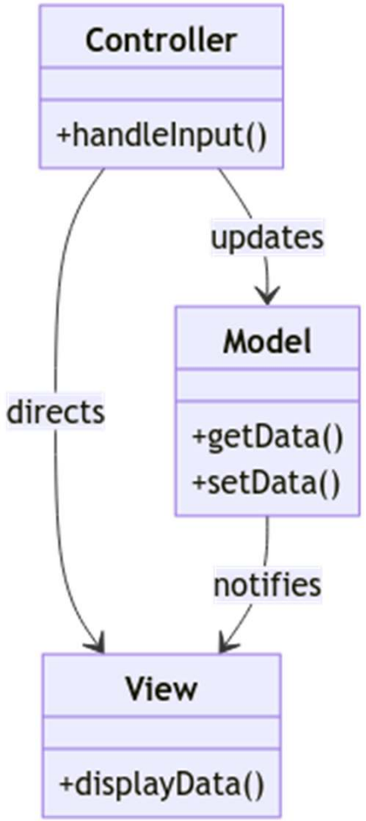
INTERACTIVE SYSTEM & MVC STYLE

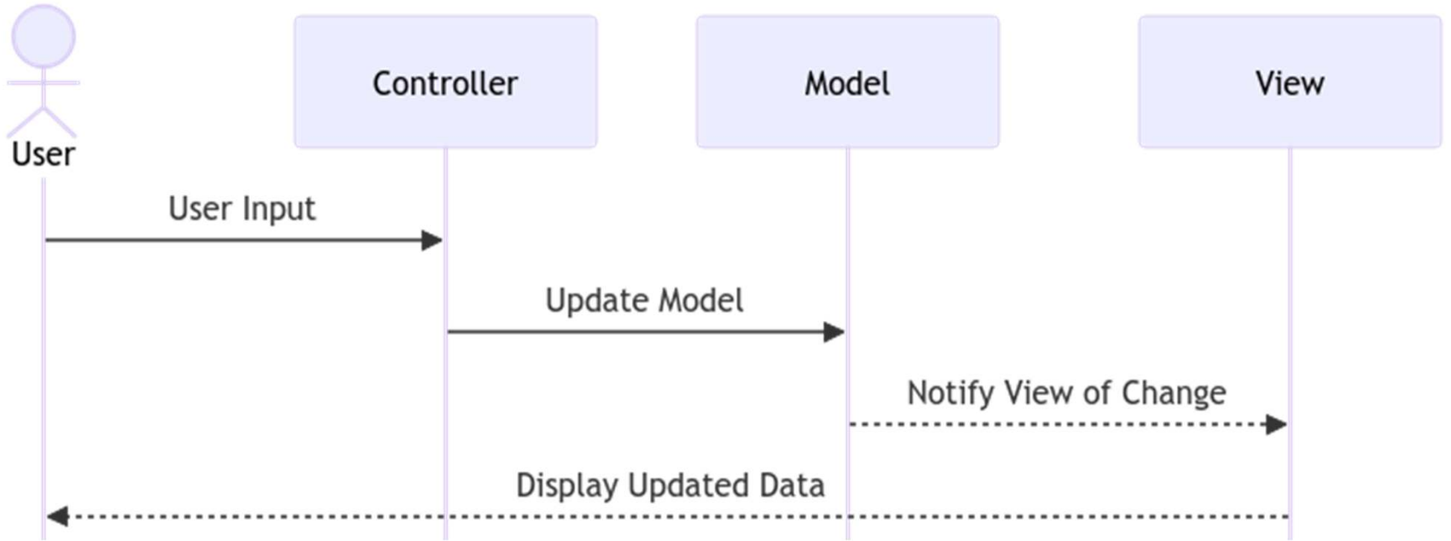
MVC is ideal for interactive systems as it allows user interface components (Views) to be easily updated and modified without affecting data or input logic (Model and Controller).

Quality	Description
Modifiability	Easy to exchange, enhance, or add additional user interfaces.
Usability	By allowing easy exchangeability of user interfaces, systems can be configured with different user interfaces to meet different usability needs of particular groups of customers.
Reusability	Be separating the concerns of the model, view, and controller components, the can all be reused in other systems.



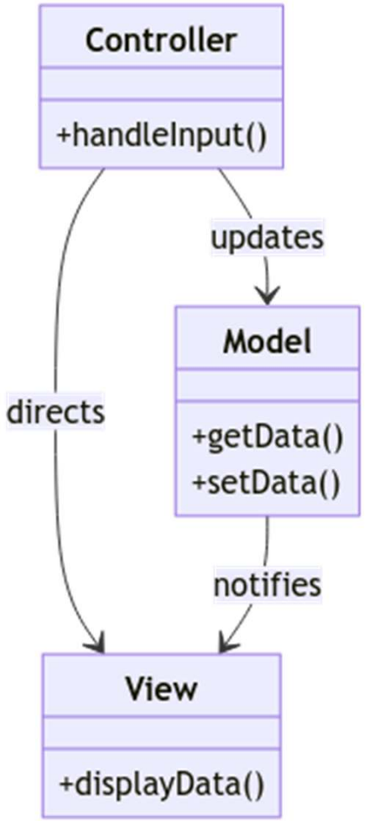
MVC STYLE





DISCLAIMER THE DIAGRAM DRAWN MAY NOT COMPLETELY CORRECT.
JUST TO ILLUSTRATE THE DIAGRAM STRUCTURE FOR THE ARCHITECTURAL STYLE.

MVC STYLE




```
// Model.java
public class Model {
    private String data = "Initial Data";

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}
```

```
// View.java
public class View {
    public void displayData(String data) {
        System.out.println("Data: " + data);
    }
}
```

```
// Controller.java
public class Controller {
    private Model model;
    private View view;

    public Controller(Model model, View view) {
        this.model = model;
        this.view = view;
    }

    public void handleInput(String data) {
        model.setData(data);
        view.displayData(model.getData());
    }
}
```

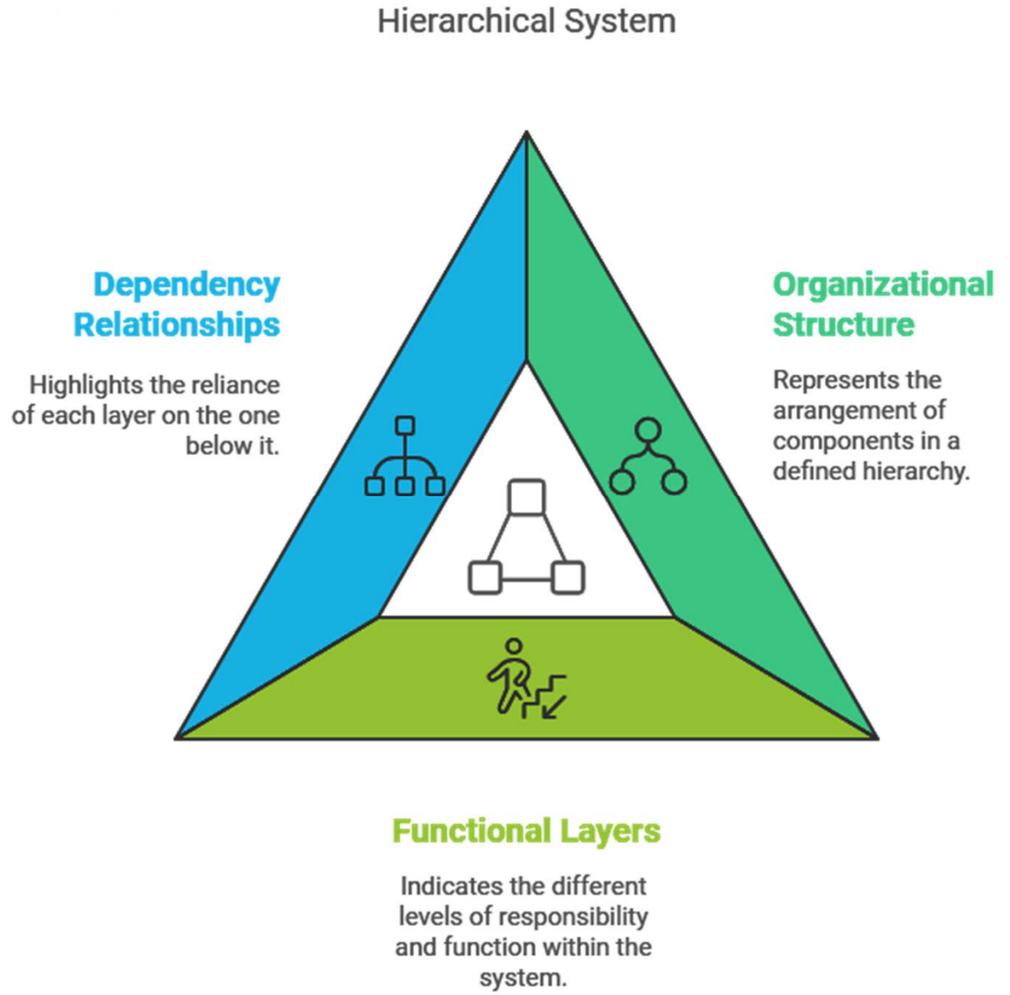
```
// Main.java
public class Main {
    public static void main(String[] args) {
        Model model = new Model();
        View view = new View();
        Controller controller = new Controller(model, view);
        controller.handleInput("Updated Data");
    }
}
```

HIERARCHICAL SYSTEMS & LAYERED STYLE

- Hierarchical systems can be decomposed and structured in hierarchical fashion. Common architectural patterns for hierarchical systems is Layered

- Quality properties of the Layered architectural pattern include the ones specified below.

Quality	Description
Modifiability	Dependencies are kept local within layer components. Since components can only access other components through a well-defined and unified interface, the system can be modified easily by swapping layer components with other enhanced or new layer components.
Portability	Services that deal directly with platform's API's can be encapsulated using a system layer component. Higher level layers rely on this component for providing system services to the application, therefore, by porting the system's API layer to other platforms systems become more portable.
Security	The controlled hierarchical structure of layered systems allow for easy incorporation of security components to encrypt/decrypt incoming/outgoing data.
Reusability	By compartmentalizing each layer's services, they become easier to reuse.



Hierarchical System with Layered Architecture

Business Logic Layer

Responsible for processing data and implementing business rules, this layer acts as the system's core.

Presentation Layer

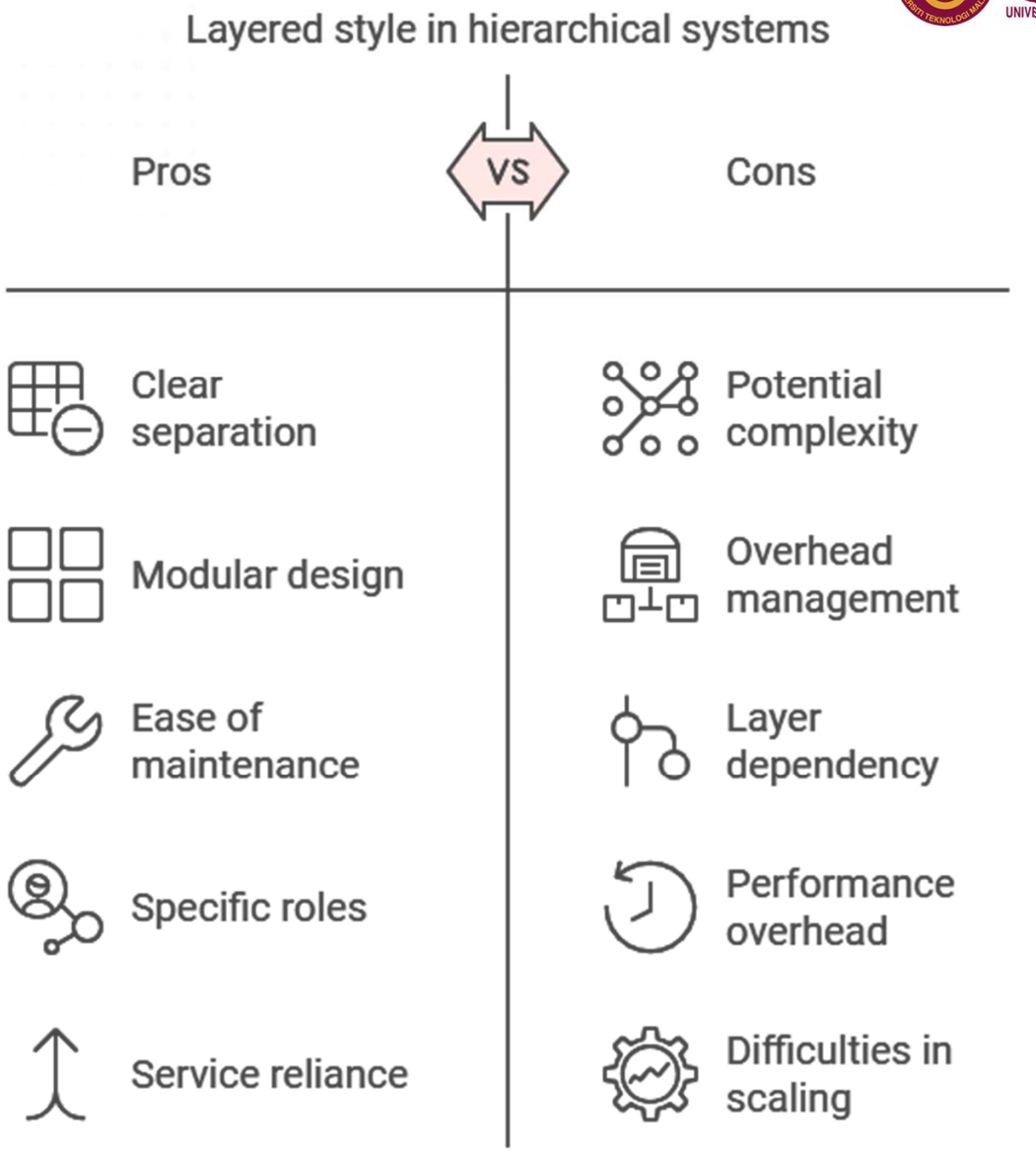
This layer handles user interface and interaction, serving as the system's front end.

Data Access Layer

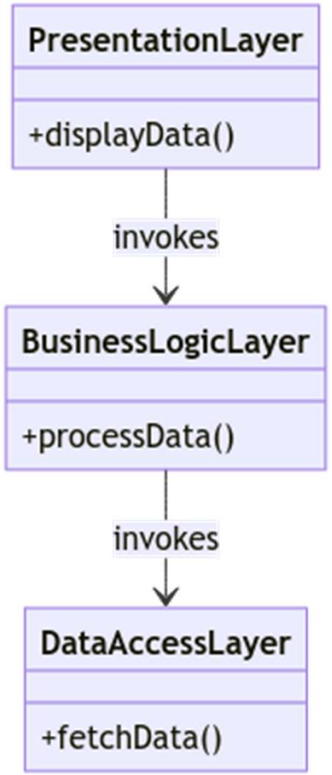
This layer manages data storage and retrieval, ensuring data flow within the system.

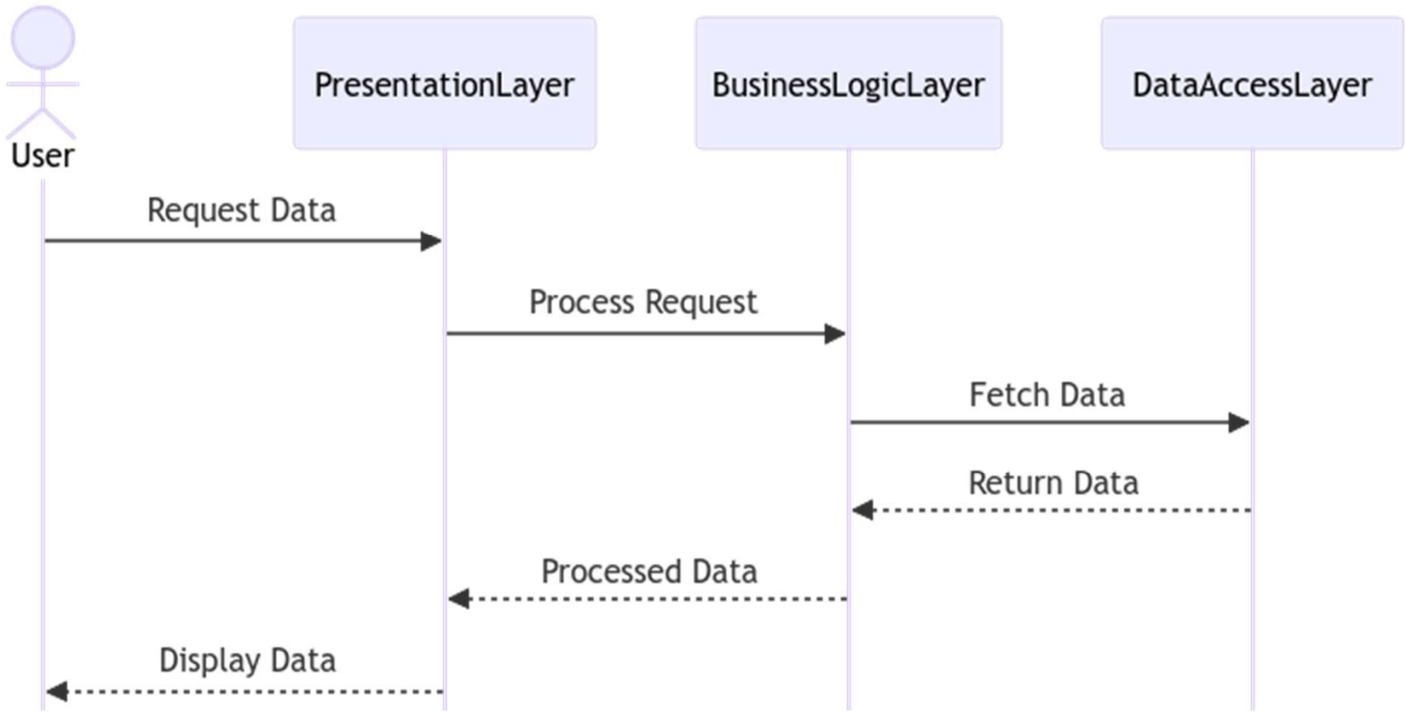
HIERARCHICAL SYSTEM & LAYERED STYLE

Hierarchical systems use the layered style as it provides clear separation of concerns. Each layer has a specific role, with the topmost layers relying on the services of lower layers, promoting modularity and ease of maintenance.



LAYERED STYLE





***DISCLAIMER* THE DIAGRAM DRAWN MAY NOT COMPLETELY CORRECT.**
JUST TO ILLUSTRATE THE DIAGRAM STRUCTURE FOR THE ARCHITECTURAL STYLE.

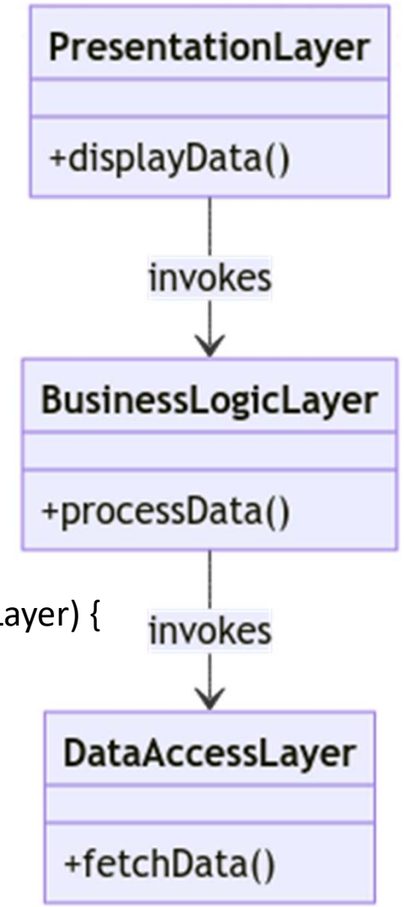
LAYERED STYLE

```
// DataAccessLayer.java
public class DataAccessLayer {
    public String fetchData() {
        // Logic to fetch data
        return "Fetched Data";
    }
}
```

```
// BusinessLogicLayer.java
public class BusinessLogicLayer {
    private DataAccessLayer dataAccessLayer;

    public BusinessLogicLayer(DataAccessLayer dataAccessLayer) {
        this.dataAccessLayer = dataAccessLayer;
    }

    public String processData() {
        String data = dataAccessLayer.fetchData();
        // Logic to process data
        return "Processed " + data;
    }
}
```




```
// PresentationLayer.java
public class PresentationLayer {
    private BusinessLogicLayer businessLogicLayer;

    public PresentationLayer(BusinessLogicLayer businessLogicLayer) {
        this.businessLogicLayer = businessLogicLayer;
    }

    public void displayData() {
        String data = businessLogicLayer.processData();
        System.out.println("Displaying: " + data);
    }
}
```

```
// Main.java
public class Main {
    public static void main(String[] args) {
        DataAccessLayer dataAccessLayer = new DataAccessLayer();
        BusinessLogicLayer businessLogicLayer = new
BusinessLogicLayer(dataAccessLayer);
        PresentationLayer presentationLayer = new
PresentationLayer(businessLogicLayer);
        presentationLayer.displayData();
    }
}
```

THANK YOU

univteknologimalaysia utm.my utmofficial

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Innovating Solutions

Innovating Solutions

Menginovasi Penyelesaian

Menginovasi Penyelesaian