SE 3112

SOFTWARE ARCHITECTURE

Prepared by: Dams Gabriel Lazarus
ICT University, Fall 2022

CHAPTER 4:

Software Architecture Patterns

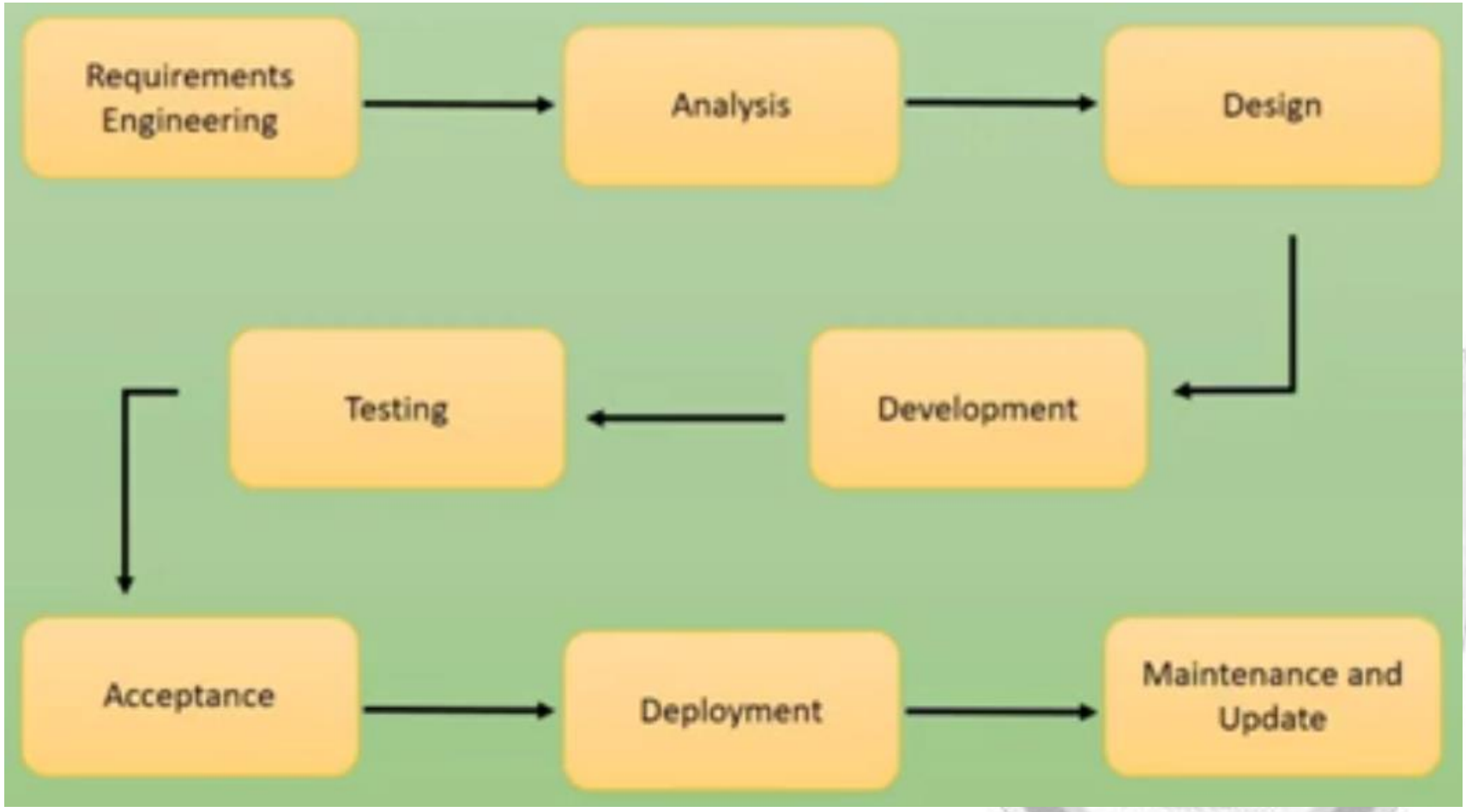
Topic Overview

- What are Architectural Patterns
- Purpose of Architectural Patterns
- Some Types of Architectural Patterns
- Applications/Usage
- Advantages and Disadvantages

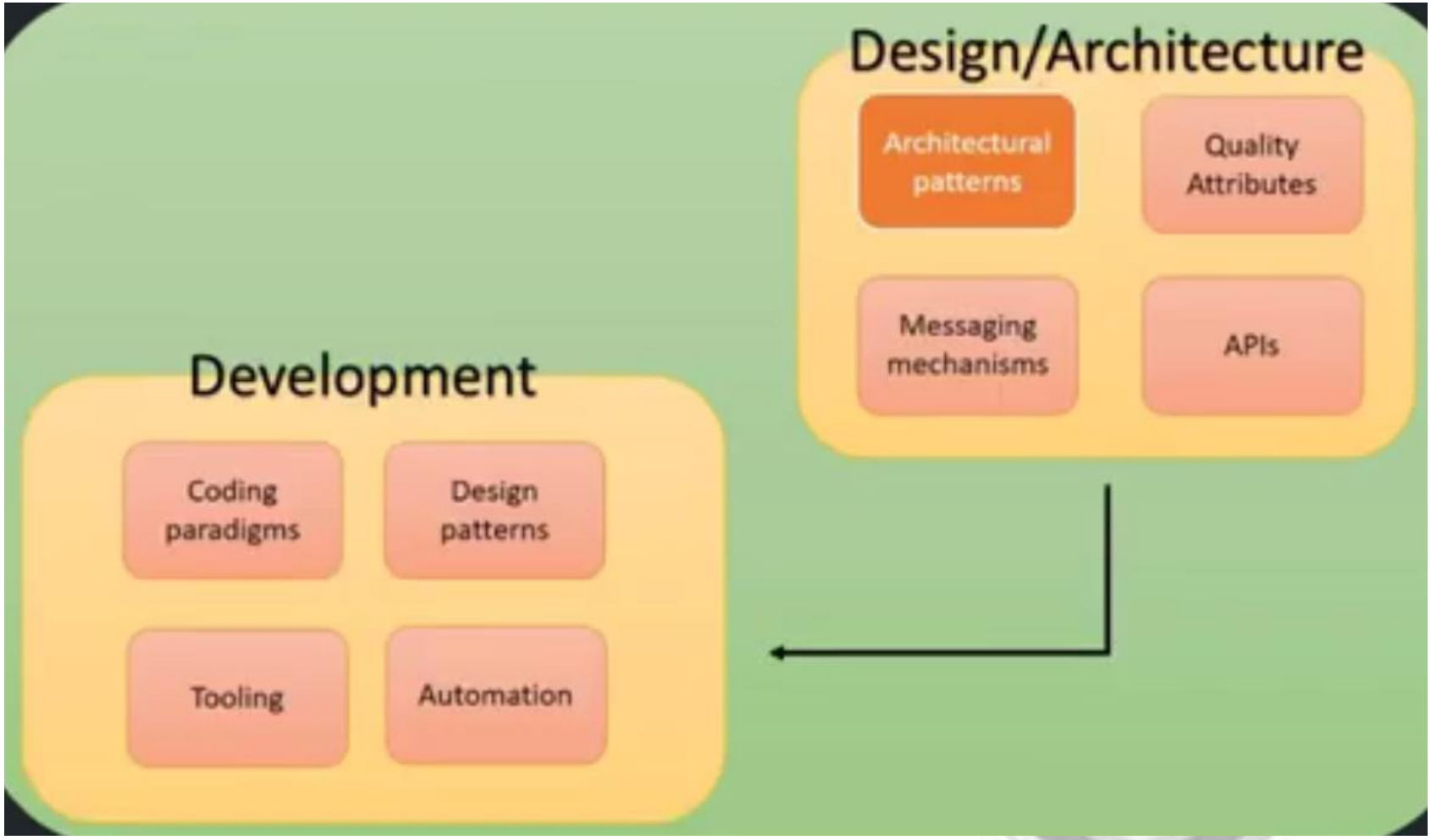
What is Architectural Pattern?

- Choosing a suitable architecture style before major software development helps to give the software development team the desired functionality and quality attributes required.

What is Architectural Pattern?



What is Architectural Pattern?

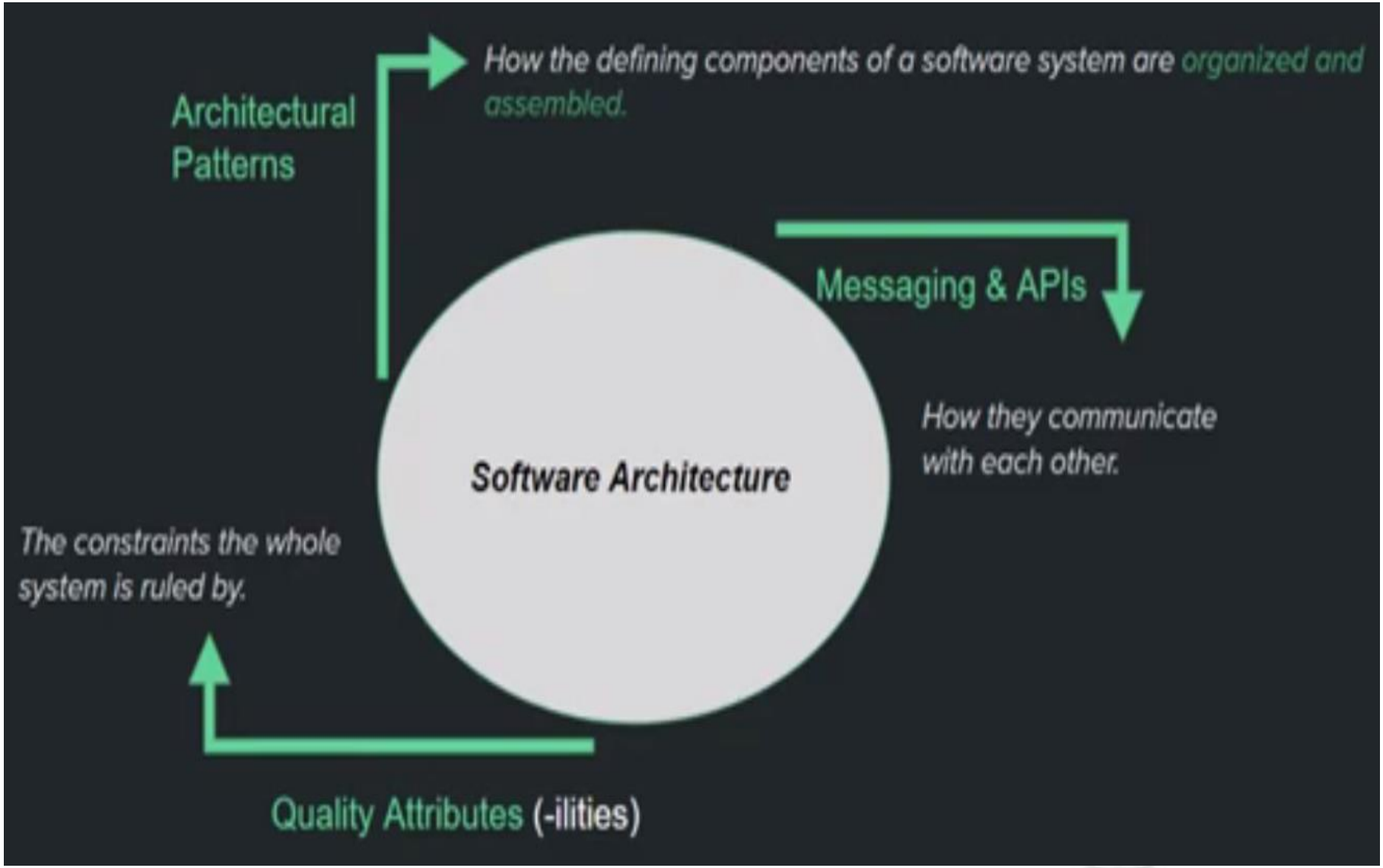


What is Architectural Pattern?

- Since the design output is the input to the development phase, the developers will have to care about the architecture of the software.

- Software architecture definition in a nutshell is:
  - ***How the defining components are organized and assembled***
  - ***How the components communicate with each other***
  - ***The constraints the whole system is ruled by.***

What is Architectural Pattern?



What is Architectural Pattern?

- According to Wikipedia,
  - *An architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture within a given context. Architectural patterns are similar to software design pattern but have a broader scope.*

- The **purpose** is to understand how the major parts of the system fit together and how messages and data flow through the system.

- We can have multiple patterns in a single system to optimize each section of our code.

What is Architectural Pattern?

- The architectural pattern defines the ***granularity*** of the components
  - ***Granularity*** = how big or small the components should be.
- One cannot start coding without knowing the architectural pattern.
- The mazing cannot start building without knowing the shape of the building.

What is Architectural Pattern?

- Knowing the architectural patterns helps us make good decision in the development phase.

- Decision such as choice of design patterns

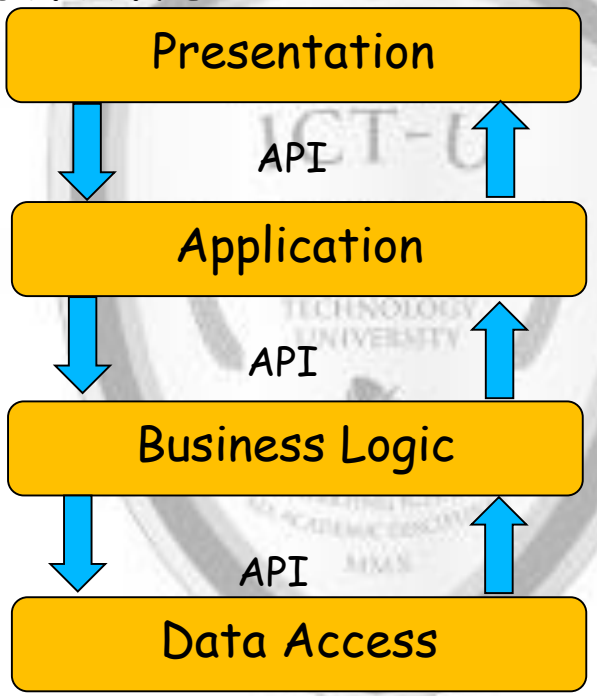
Types of Architectural Pattern?

- Layered Pattern
- Client-Server Pattern
- Peer-to-Peer Pattern
- Event-Driven Pattern
- Microkernel Pattern
- Microservices Pattern

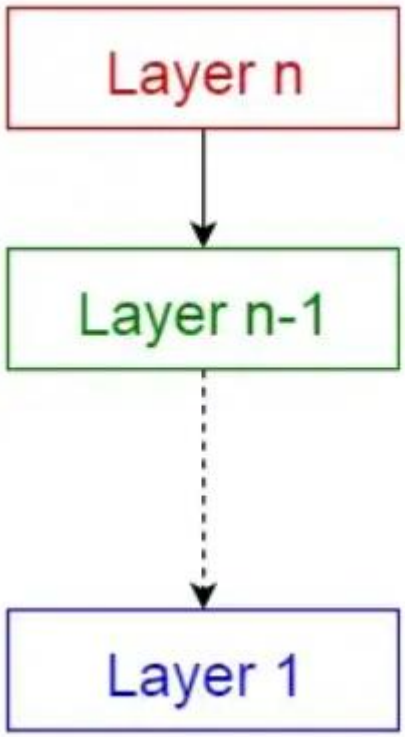
Layered Architecture (N-Tier)

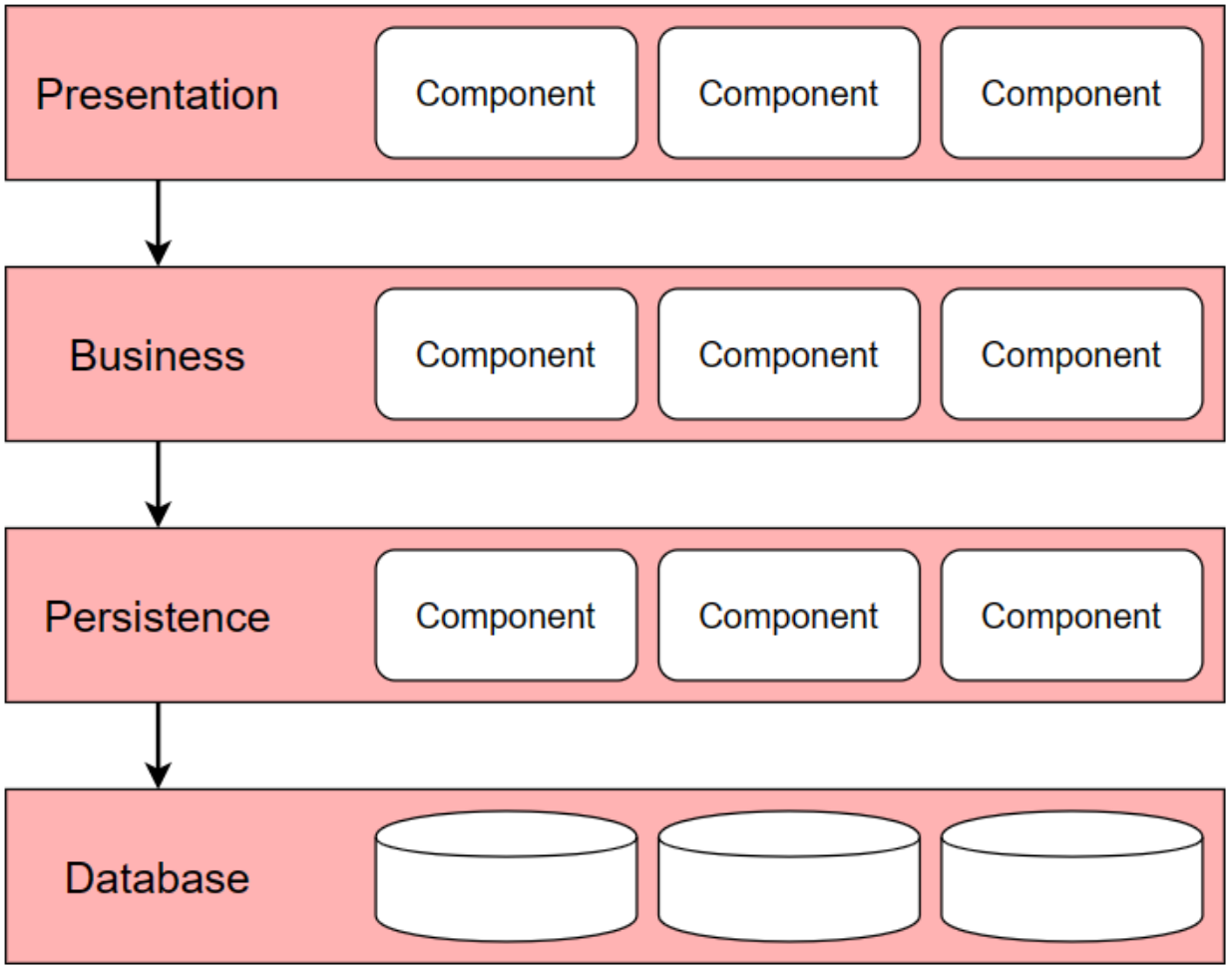
- Most used in software system
- Also known as the N-tier pattern
- Monolithic in nature
- It advocates for the separation of concerns
- Layers are the major components

- Made up of 4 layers
  - Presentation or UI
  - Application or Service
  - Business logic or Domain
  - Data access or Persistence

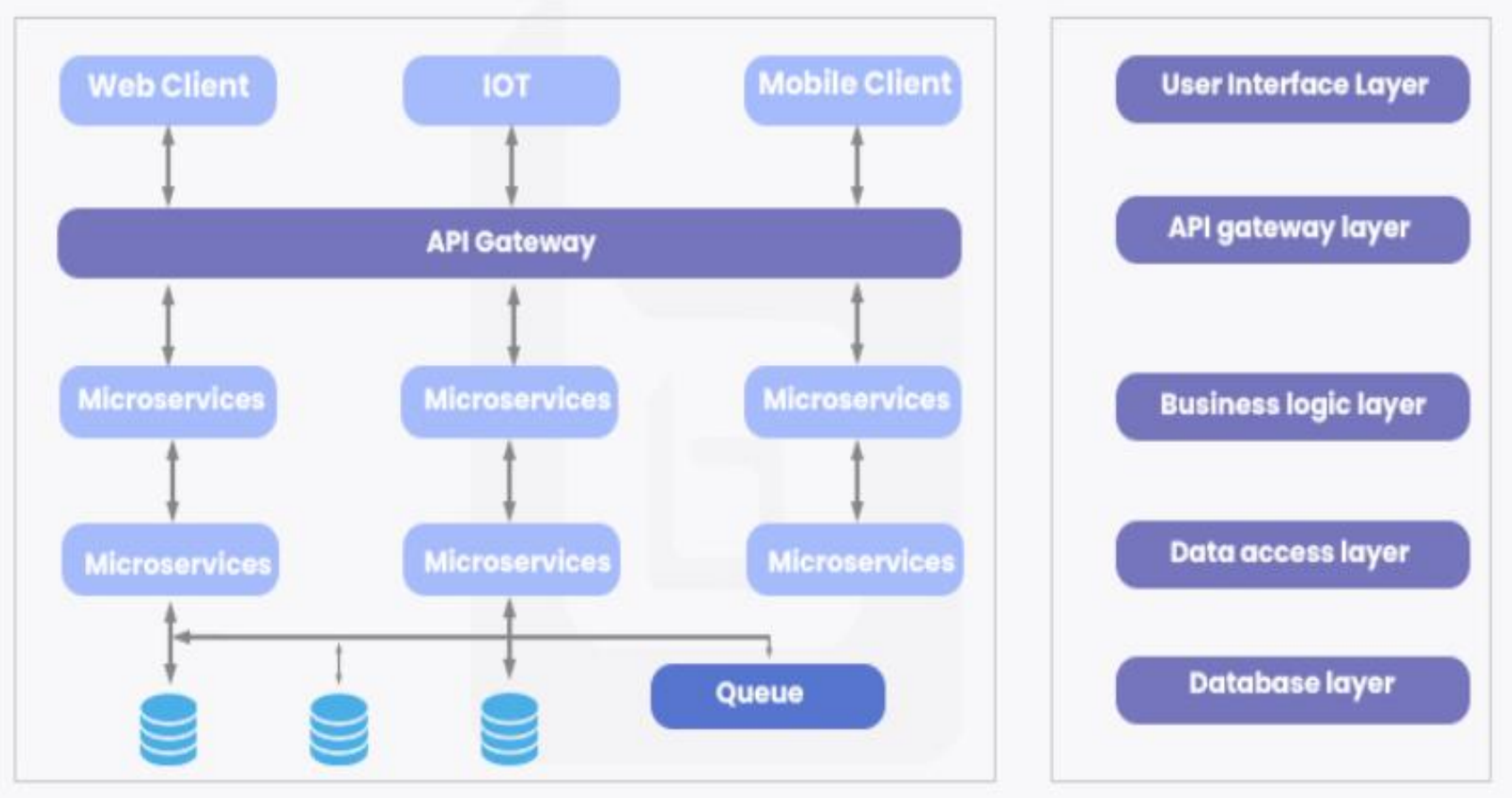


Layered Architecture (N-Tier)





Layered Architecture (N-Tier)



Layered Architecture (N-Tier)

- Also known as the N-tier pattern.
- The standard architecture used for most Java Enterprise applications.
- Divides components (or applications) into horizontal, logical layers.
- Each layer has a distinct role within the system.

Layered Architecture (N-Tier)

- One layer may be responsible for handling business logic, while another is responsible for handling presentation logic.
- This demonstrates a concept known as the separation of concerns (SoC).
- Components of a layer will only deal with logic within that layer.

Layered Architecture (N-Tier)

- Occasionally, the business layer and persistence layer are combined into a single layer, especially when the persistence logic (e.g., SQL) is contained within components in the business layer.

- **Ideal for**: E-commerce web applications development. E.g., the Amazon shopping app.

Layered Architecture (N-Tier)

- **Usage**
  - Typically used in enterprise applications.
  - When requiring strict standards of maintainability and testability.
  - Desktop applications or ecommerce web frontends
  - The right technology for the traditional IT department.
  - Perfect for the inexperienced team who may lack knowledge about architecture patterns

Layered Architecture (N-Tier)

- **Advantages**
  - Separating layers like this makes testing and developing software easier as the pattern itself isn't too complex.
  - Layers make standardization easier as we can clearly define levels
  - Changes can be made within the layer without affecting other layers

Layered Architecture (N-Tier)

- **Disadvantages**
- This isn't the most efficient pattern to use and can be difficult to scale up.

- Certain layers may have to be skipped in certain situations.

Client-Server Architecture

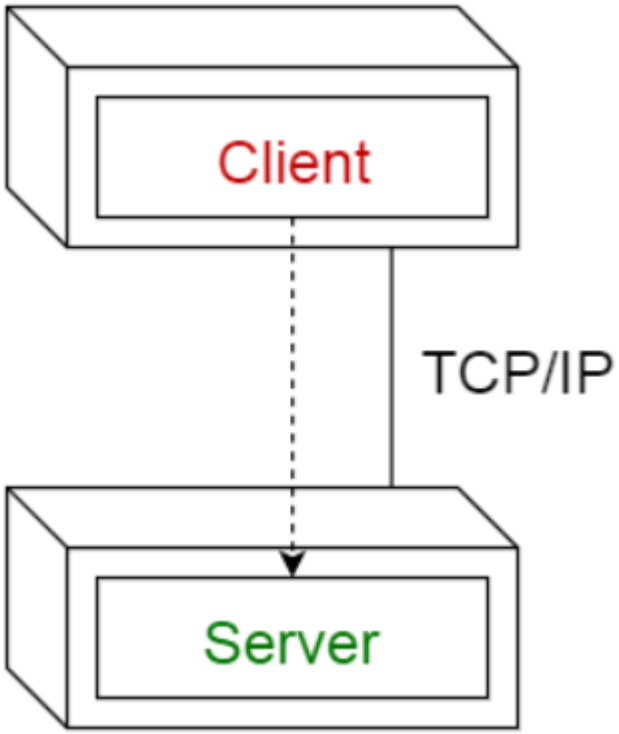
- Multiple nodes or clients connected over a network or internet connection who communicate with a central server.

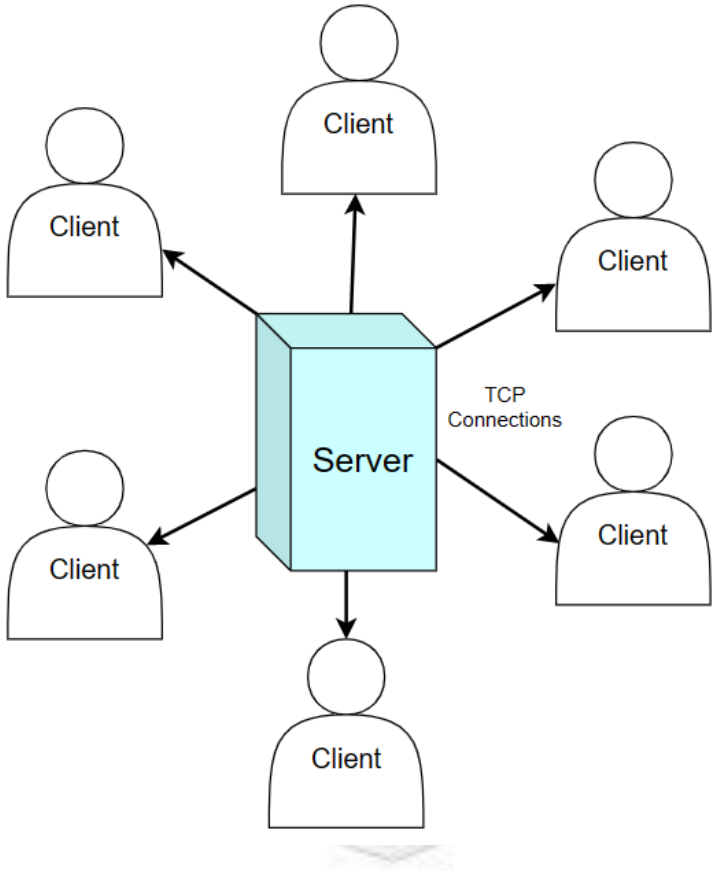
- There are two main types of components:
  - Service requesters (aka clients) that send requests
  - Service providers that respond to requests

- the server hosts, manages and delivers most of the resources and services a client requests. This is also known as a request-response messaging pattern.

Client-Server Architecture

- A couple of classic examples of applications with a client-server architecture are the World Wide Web and email.





Client-Server Architecture

- **Usage**:
  - When building applications with Email services
  - File sharing apps
  - Banking apps

- **Ideal for**: Multi utility apps with strong security features. E.g., Gmail.

Client-Server Architecture

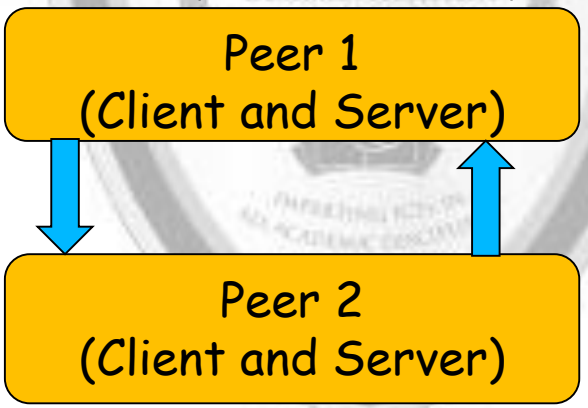
- **Advantage**
  - Good to model a set of services where clients can request them
  - The centralized network has complete leverage to control the processes and activities.
  - All devices in the network can be controlled centrally.
  - Users have the authority to access any file, residing in the central storage, at any time.
  - Provides a good user interface, easy file finding procedure, and management system for organizing files.
  - Easy sharing of resources across various platforms is possible.

Client-Server Architecture

- **Disadvantages**
  - If the primary server goes down, the entire architecture is disrupted.
  - It is expensive to operate because of the cost of heavy hardware and software tools.
  - This architecture requires particular OSs related to networking.
  - Too many users at once can cause the problem of traffic congestion.

Peer-to-Peer Architecture

- Individual components are known as peers.
- Peers may function both as a client, requesting services from other peers, and as a server, providing services to other peers.
- A peer may act as a client or as a server or as both, and it can change its role dynamically with time.



Peer-to-Peer Architecture

- **Usage**:
  - Good when considering File-sharing networks such as Xender, Feem etc
  - Ideal for equal distributed computational entities that are connected via a common protocol to share their services and provide high availability and scalability.
  - Multimedia protocols.
  - Cryptocurrency-based products such as Bitcoin and Blockchain

Peer-to-Peer Architecture

- **Advantages**
  - Supports decentralized computing
  - Highly robust in the failure of any given node
  - Highly scalable in terms of resources and computing power
- **Disadvantages**
  - No guarantee about quality of service, as nodes cooperate voluntarily.
  - Security is difficult to be guaranteed.
  - Performance depends on the nodes.

Event-driven Architecture

- Ability to detect "events" or important business moments (such as a transaction, site visit, shopping cart abandonment, etc) and act on them in real time or near real time.

- It replaces the traditional "request/response" architecture where services would have to wait for a reply before they could move onto the next task.

- often referred to as "asynchronous" communication.

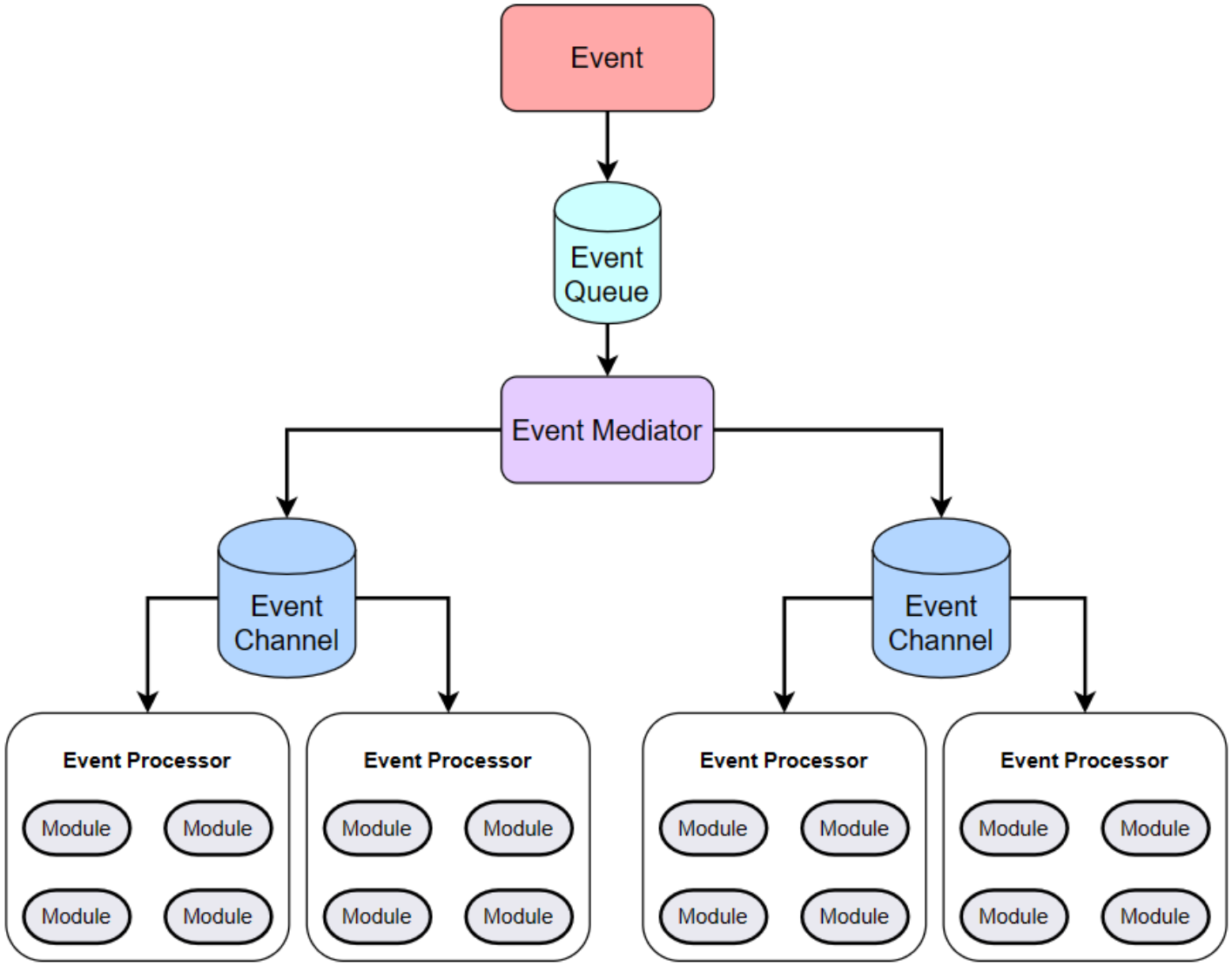
Event-driven Architecture

- Sender and recipient don't have to wait for each other to move onto their next task.
- Systems are not dependent on that one message
- For instance, phone call and text message.
  - Which is synchronous and which is asynchronous?

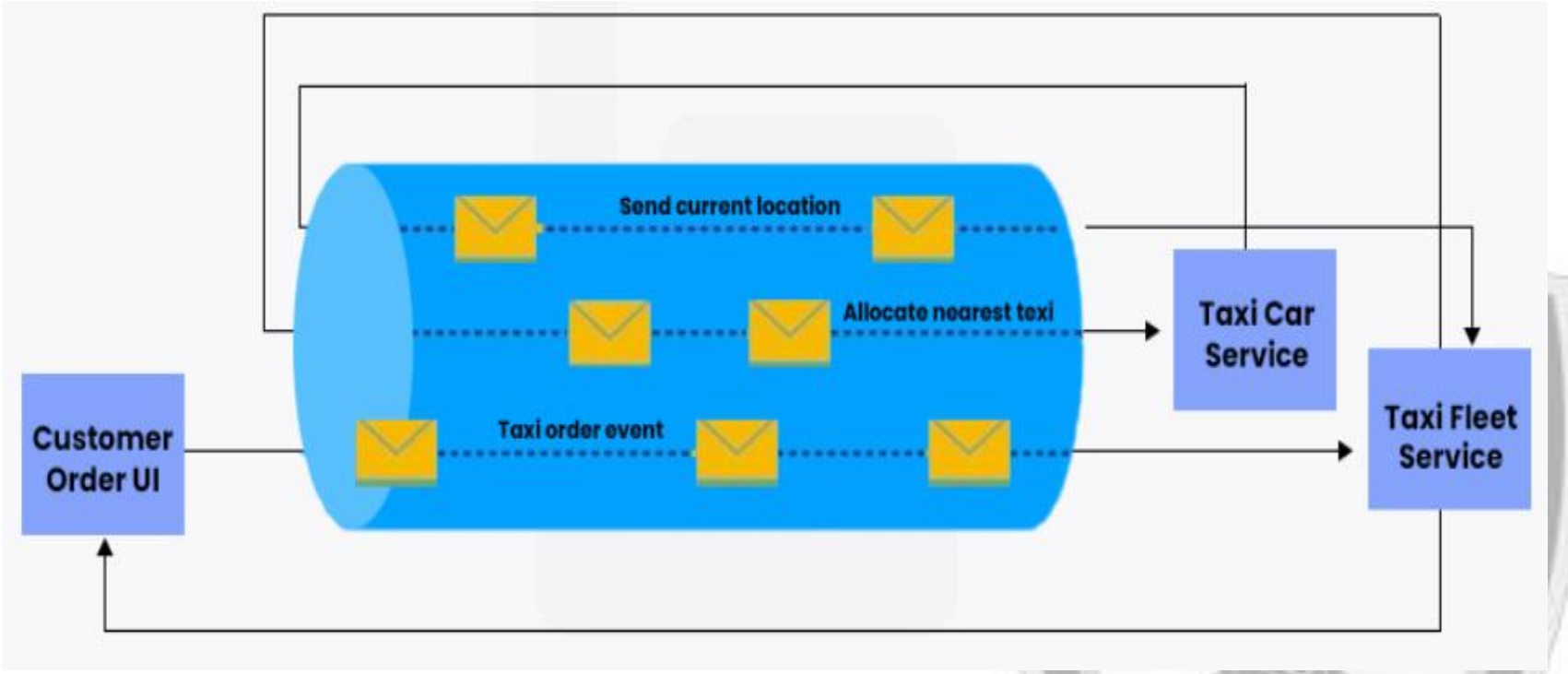
Event-driven Architecture

- **Usage**:
  - Used for interfaces that need individual data blocks to interact with just a few modules.
  - Useful when you want an easy-to-understand interface.
  - When don't have the space or time required by rows and columns of information in tables.
  - Best for Android development & Notification services.
  - **Ideal for**: Building websites with JavaScript and e-commerce in general. E.g., Credit card swipe.

Event-driven Architecture



Event-driven Architecture



Event-driven Architecture

- **Advantages**:
  - Better realtime user experiences.
  - Loose coupling between API producers and consumers.
  - Superior handling at scale.
  - More consistent handling of concurrency.

Event-driven Architecture

- **Disadvantages**:
  - Increased complexity for API provider
  - Ambiguity In API analytics.
  - Limited governance, standardization, and developer experience/support.

Microkernel Architecture

- Also known as plug-in architectures
- Typically used to implement applications that can be downloaded as a third-party product. This architecture is also commonly found in internal business applications.

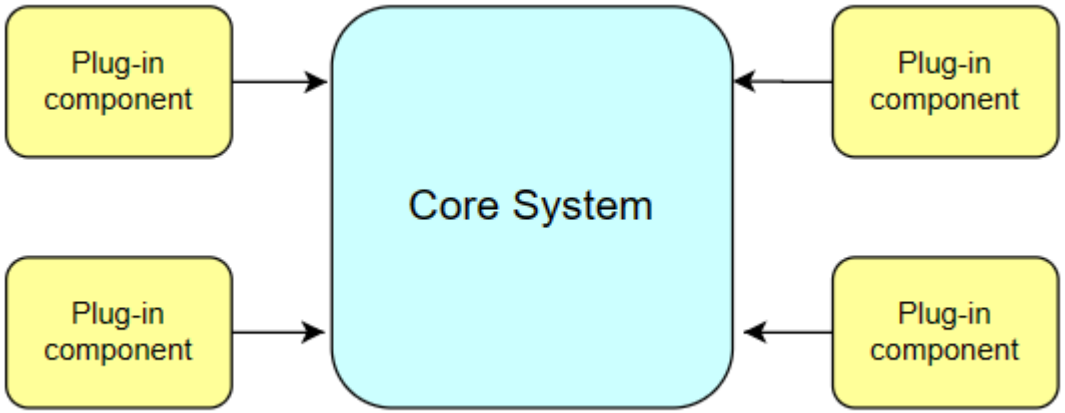
- Can actually be embedded within other patterns, like layered architectures.

- They are of two types:
  - a core system
  - plug-in modules.

Microkernel Architecture

- Core system contains the minimum business logic needed to make the software system operational.

- The software system's functionality can be extended by connecting plug-in components to add more features.



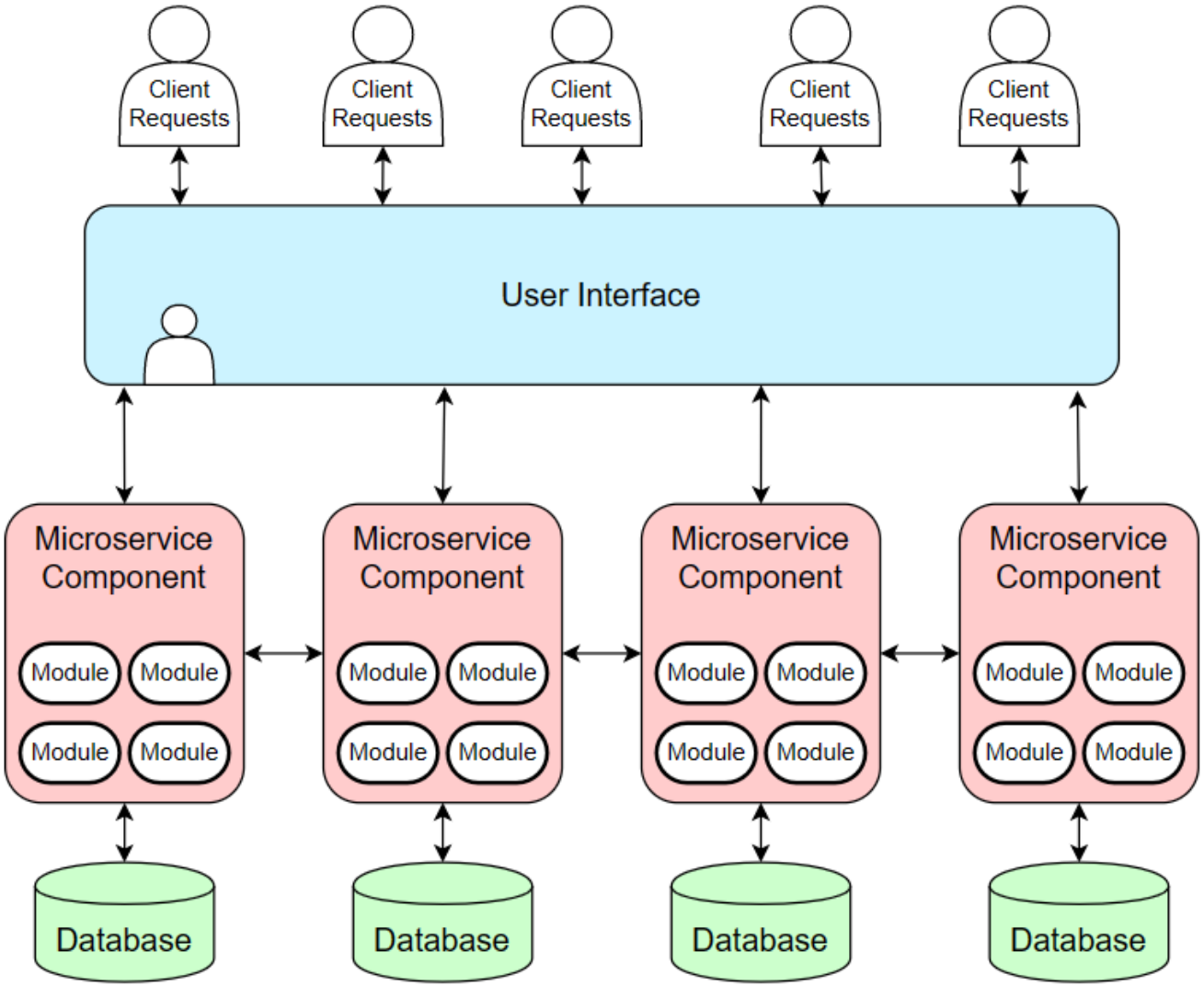
Microkernel Architecture

- Usage:
  - Can be very effective in certain industries such as healthcare and banking.
  - The right appropriate segmentation between basic routines, and higher-order rules.
  - Good for dynamic situations that need frequent updates.
  - **Ideal for**: Applications with a focus on products and scheduling. E.g., Instagram reels, YouTube Shorts, and others.

Microservices Architecture

- One of the most popular software trends at the moment.
- Attributed to the easy scalability of development.
- When microservices can no longer be maintained, they can be rewritten or replaced.
- Note:
  - No universally accepted definition for the term "microservice". We will define a microservice as an independently deployable module for this article.

Microservices Architecture



Microservices Architecture

- **Usage**:
  - Perfect for the fast and rapid development of a software or application system.
  - Good for websites that include fewer elements, high-level design components & only implement some specific function.
- **Ideal for**: Websites and web applications having small components. E.g., Spotify, Netflix etc..

Q & A