



Introduction to Software Architecture

Software Architecture
3rd Year – Semester 1
Lecture 1
By Udara Samaratunge

What is Software Architecture?

Let's Explore with an Analogy...

Software Architecture: ***What, Where, How?***

What is Software Architecture?

- *Software architecture* encompasses the structures of large software systems:
 - abstract view
 - eliminates details of implementation, algorithm, & data representation
 - concentrates on the behavior & interaction of “black box” elements

Definition

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Food for Thought...

- Quick Exercise:
 - What is the relationship of a system's software architecture to the environment in which the system will be constructed and exist?

- Answer:

- Software architecture is a result of *technical*, *business*, and *social* influences.
- In turn, it affects each of these environments.

Architecture Business Cycle (ABC)

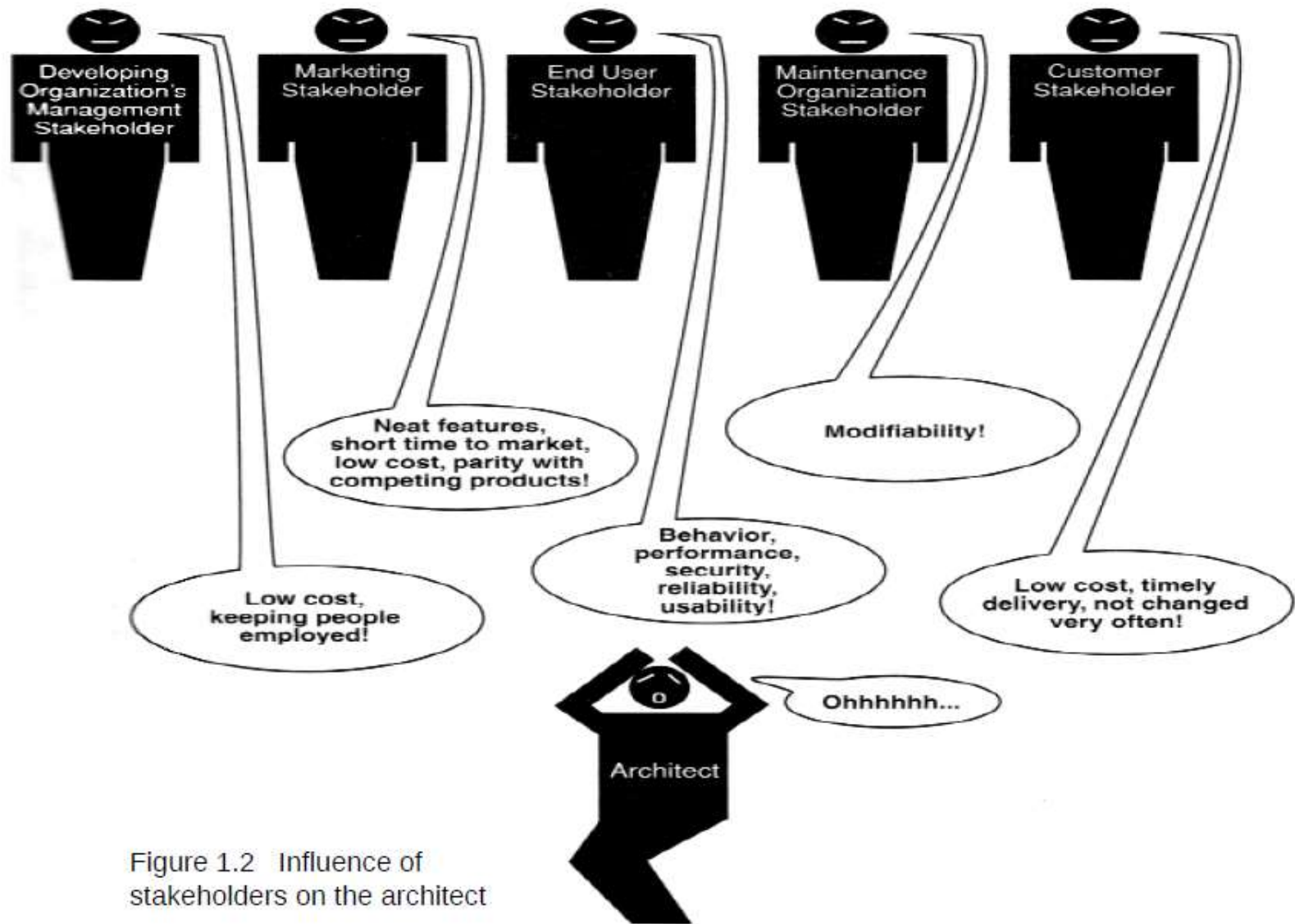


Figure 1.2 Influence of stakeholders on the architect

Architectural Influences

- **Stakeholders**
 - each stakeholder has different concerns & goals, some contradictory
- **Development Organization**
 - immediate business, long-term business, and organizational (staff skills, schedule, & budget)
- **Background & Experience of the Architects**
 - repeat good results, avoid duplicating disasters
- **The Technical Environment**
 - standard industry practices or common SE techniques

Software Architecture Patterns

Software Architecture Patterns

Why Use a Pattern?

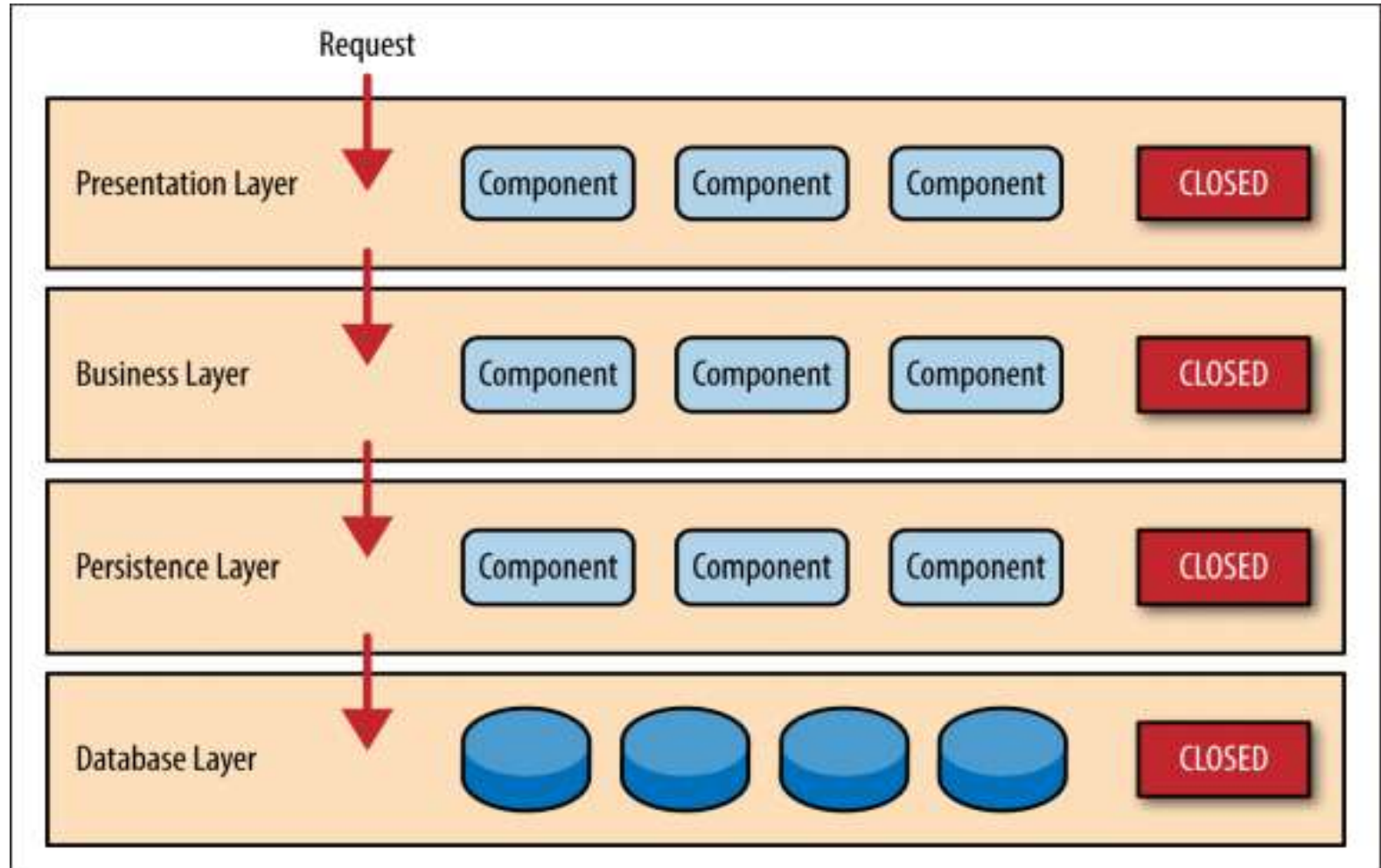
- Proven construct
- Easy to communicate
- Keep things in order

Software Architecture Patterns

- Layered Architecture
- Event-Driven Architecture
- Microkernel Architecture
- Micro services Architecture

Layered Architecture

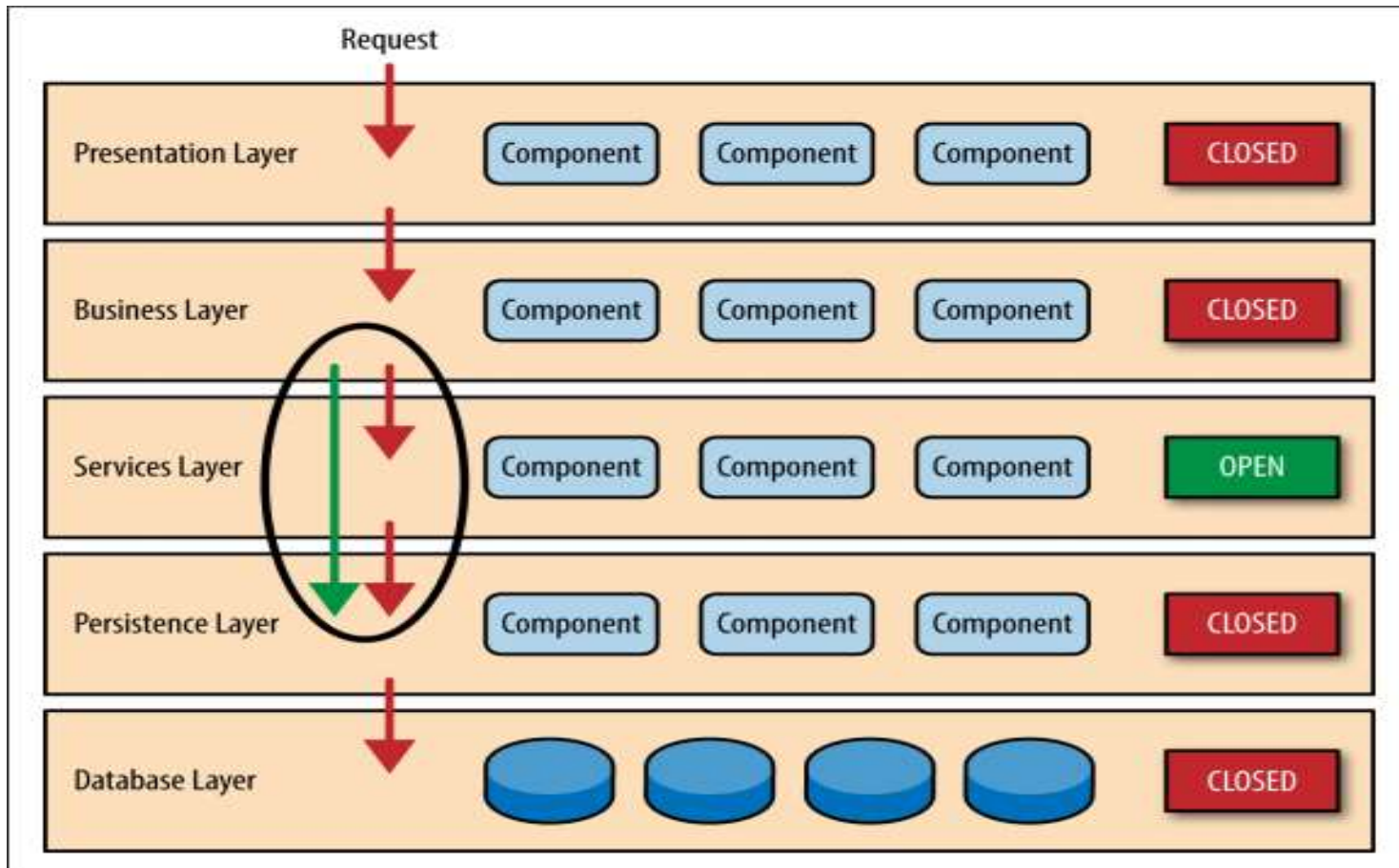
Layered Architecture is a widely used design pattern that organizes software into separate layers, each with a distinct responsibility.



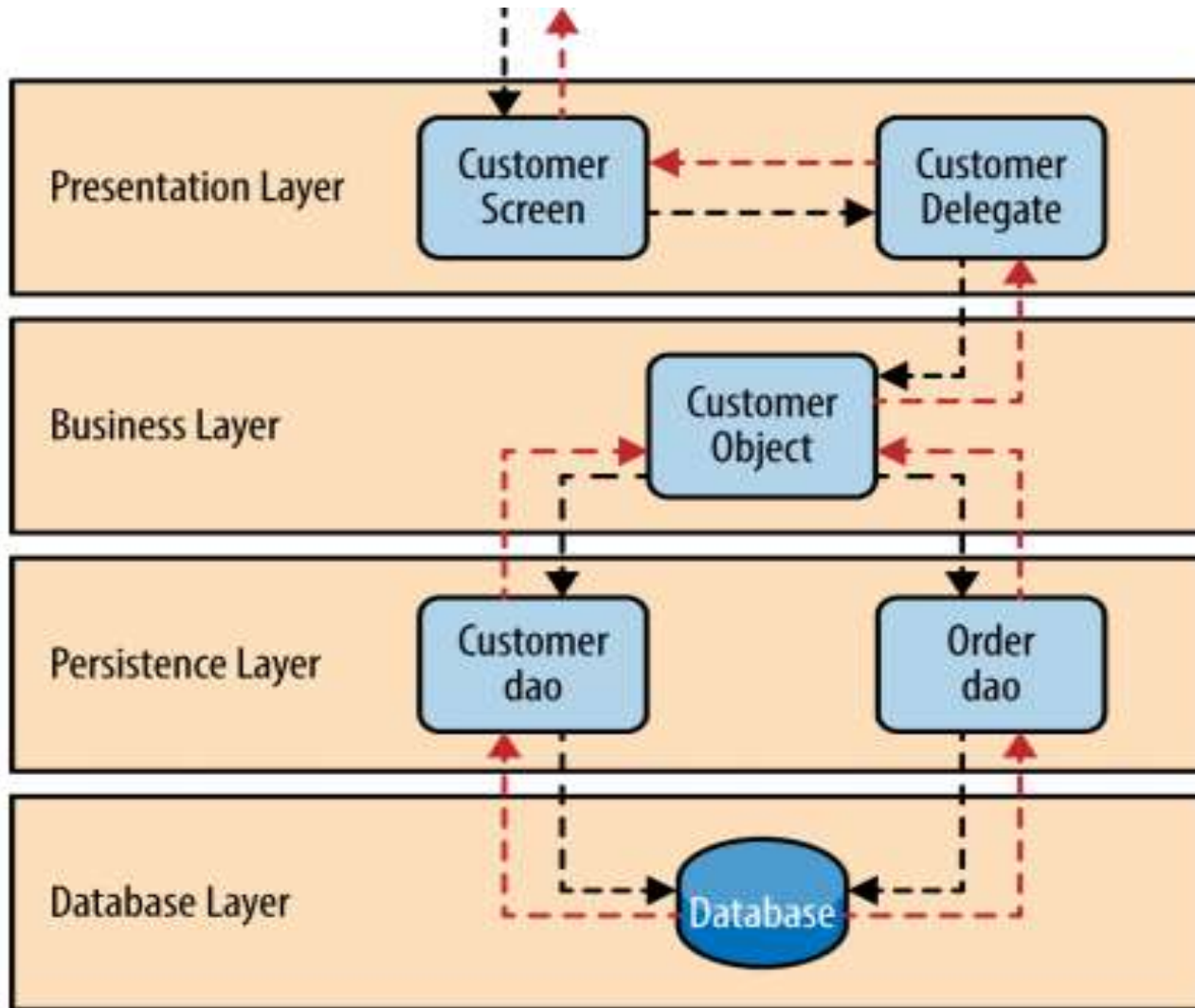
Key Concepts

- Separation of concerns
- Layers' Isolation
- Open/Closed Layers

Open/Closed Layers



Pattern Example



The Architecture Sinkhole Anti-Pattern

Requests flow through layers
without processing

This anti-pattern occurs when requests flow through multiple layers without meaningful processing, leading to unnecessary complexity and performance degradation.

Layered Architecture Pattern Analysis

- Overall Agility - Low
- Ease of Deployment - Low
- Testability - High
- Performance - Low
- Scalability - Low
- Ease of Development - High

Event Driven Architecture

Event-Driven Architecture (EDA) is a software design pattern that enables distributed, asynchronous, highly scalable, and adaptable systems

- **Distributed** Works across multiple services or microservices.
- **Asynchronous** Components interact without waiting, improving performance.
- **Highly scalable** Can handle large volumes of events efficiently.
- **Highly adaptable** Easily integrates new services without major changes.

Two main topologies:

- **Mediator**

- **Broker**

Mediator Topology

- Events processing have multiple steps that require orchestration

Used when event processing requires multiple steps and coordination.

- Four main components:

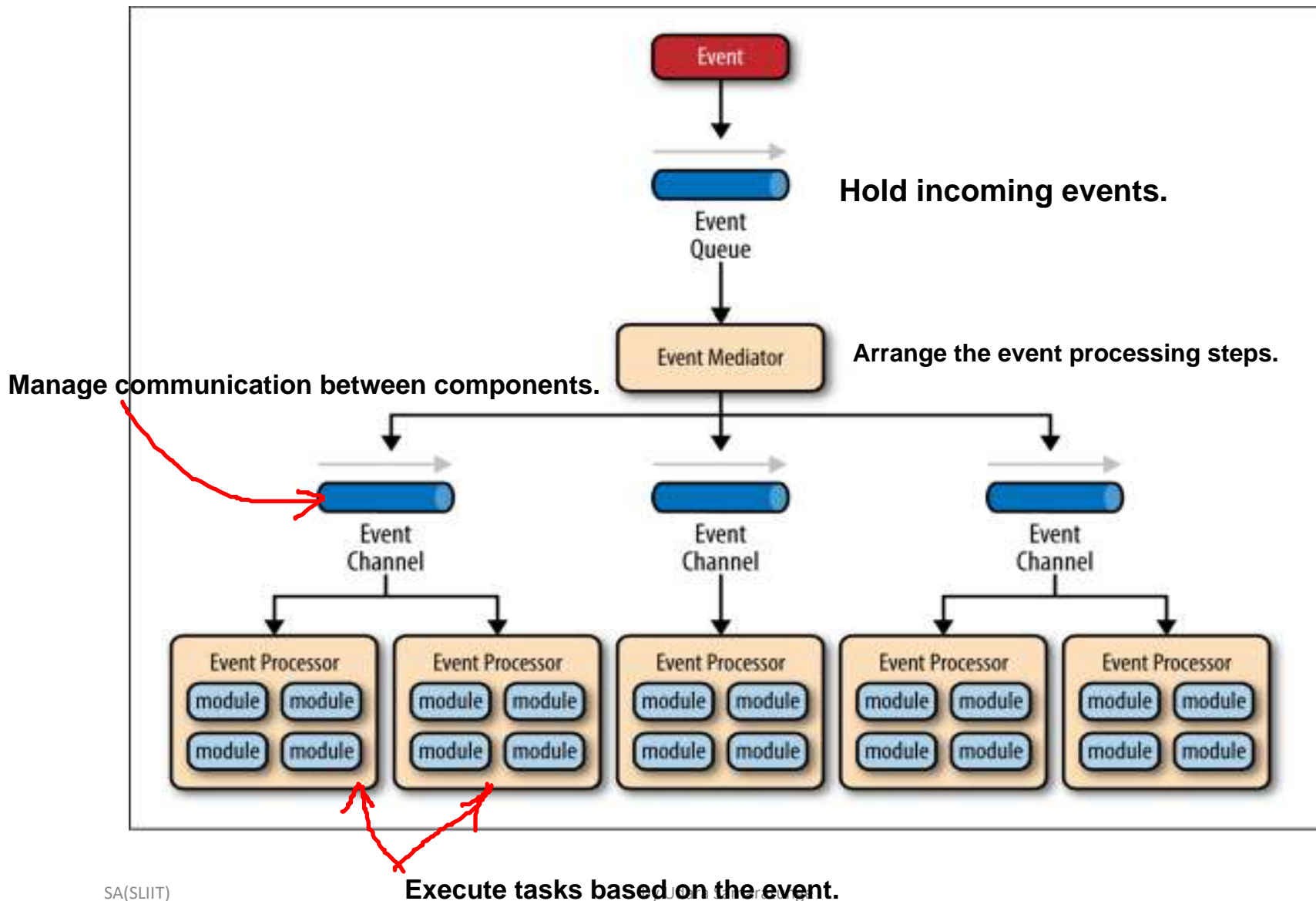
- Event Queues **Hold incoming events.**

- Event Mediator **Arrange the event processing steps.**

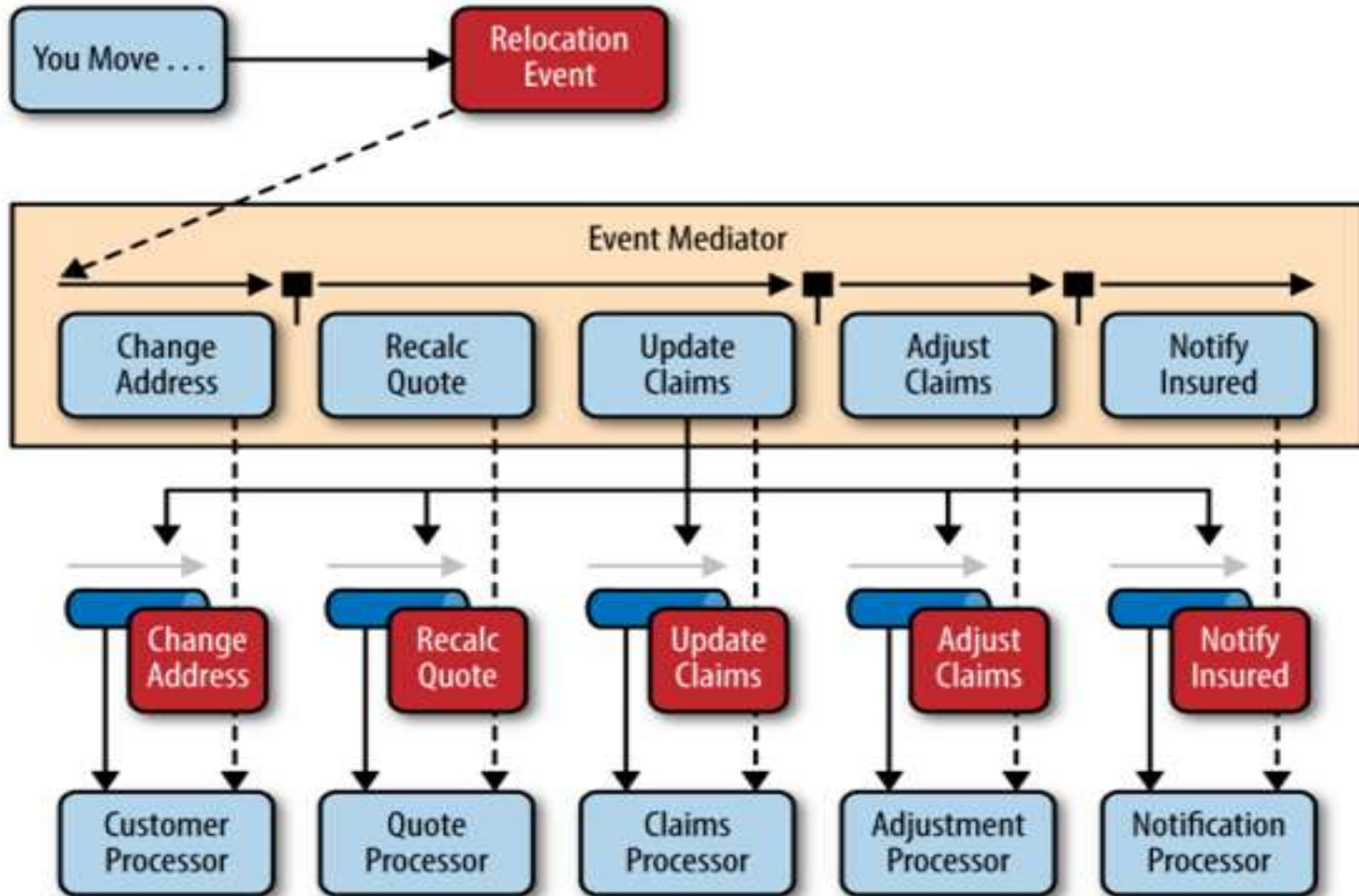
- Event Channels **Manage communication between components.**

- Event Processors **Execute tasks based on the event.**

Mediator Topology



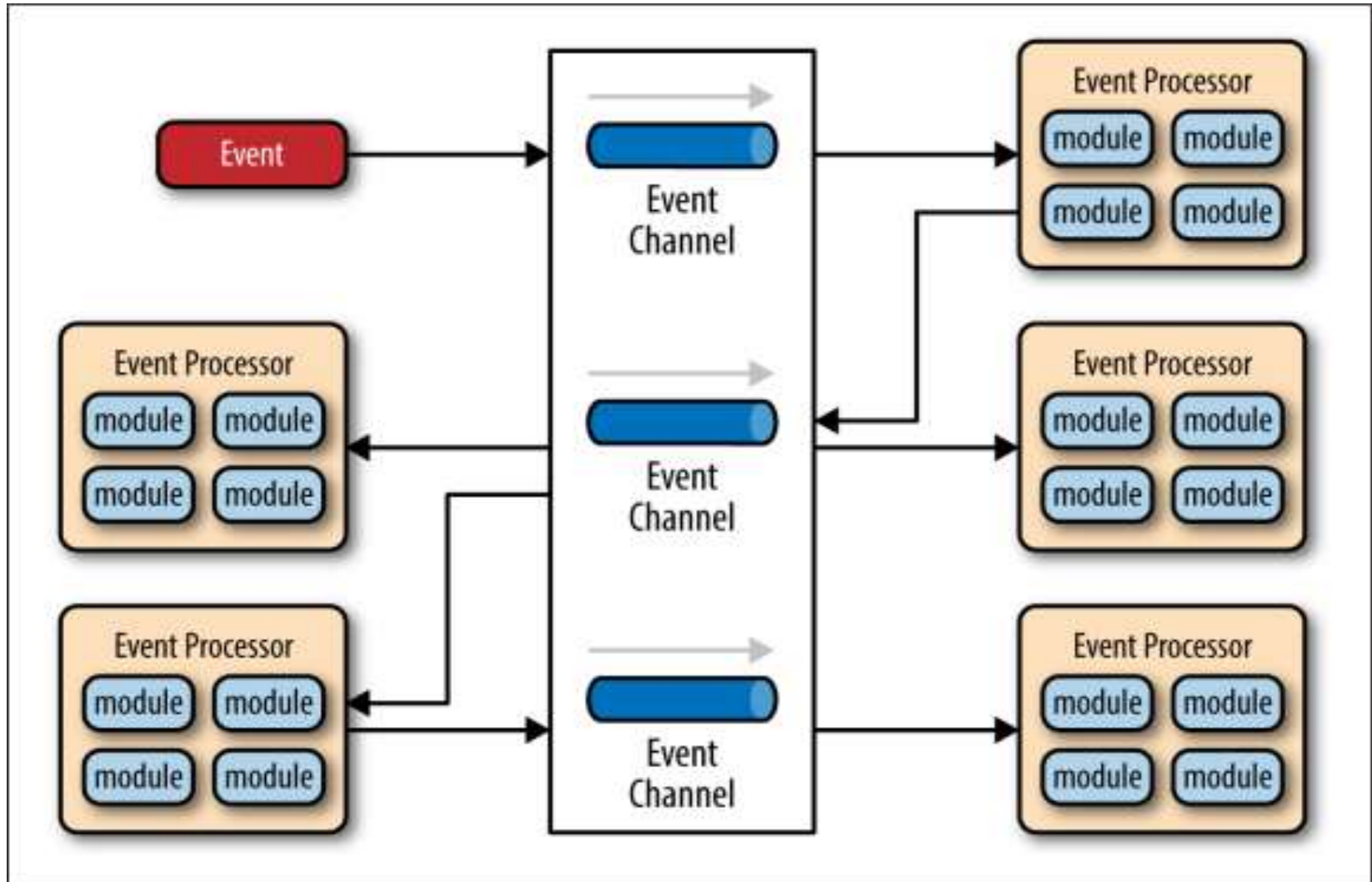
Mediator Topology Example



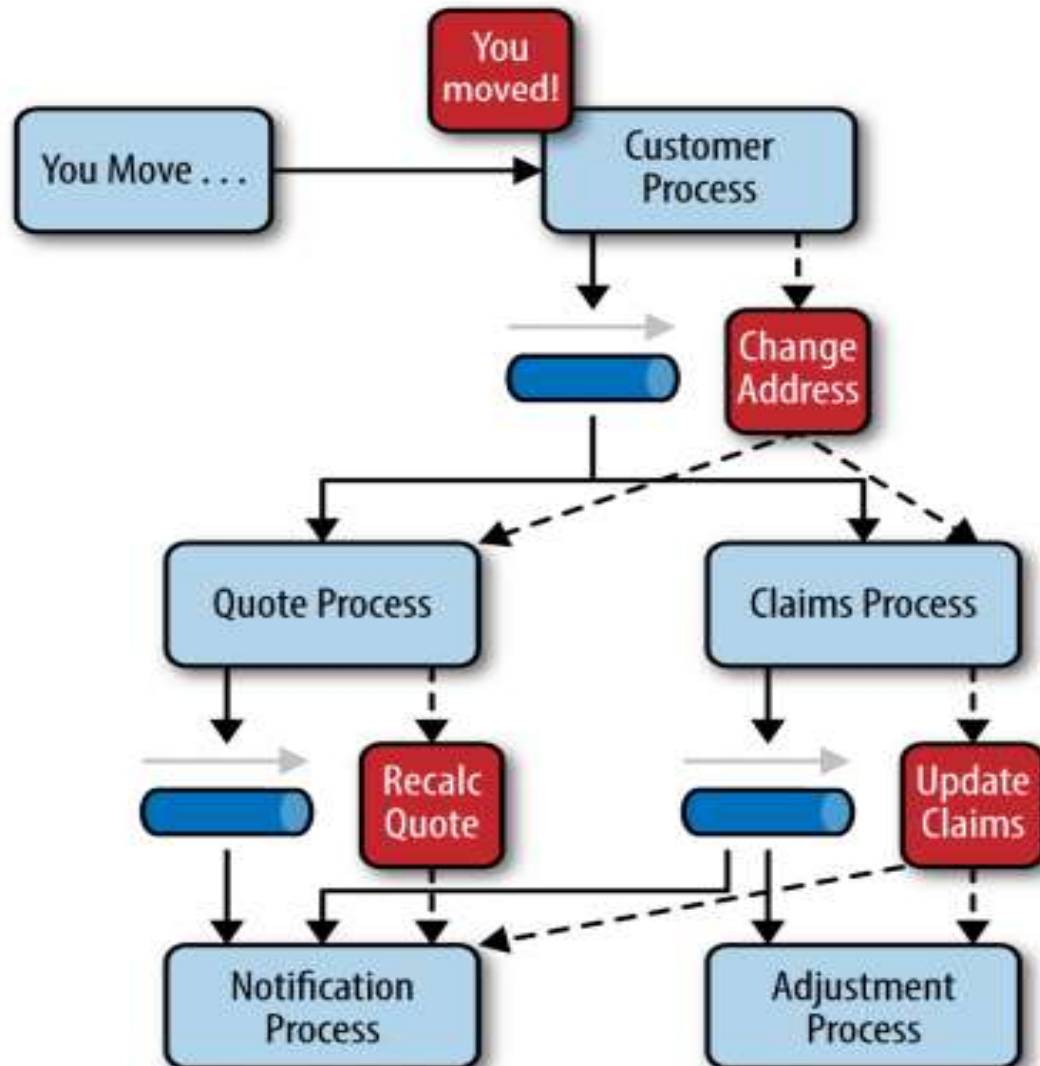
Broker Topology

- No central event mediator
- Message flows across processors in a chain like fashion
- Main components:
 - Message Broker Routes messages between processors.
 - Event Processor Consume and process events sequentially.

Broker Topology



Broker Topology Example



Microkernel Architecture

Plugin Architecture Pattern

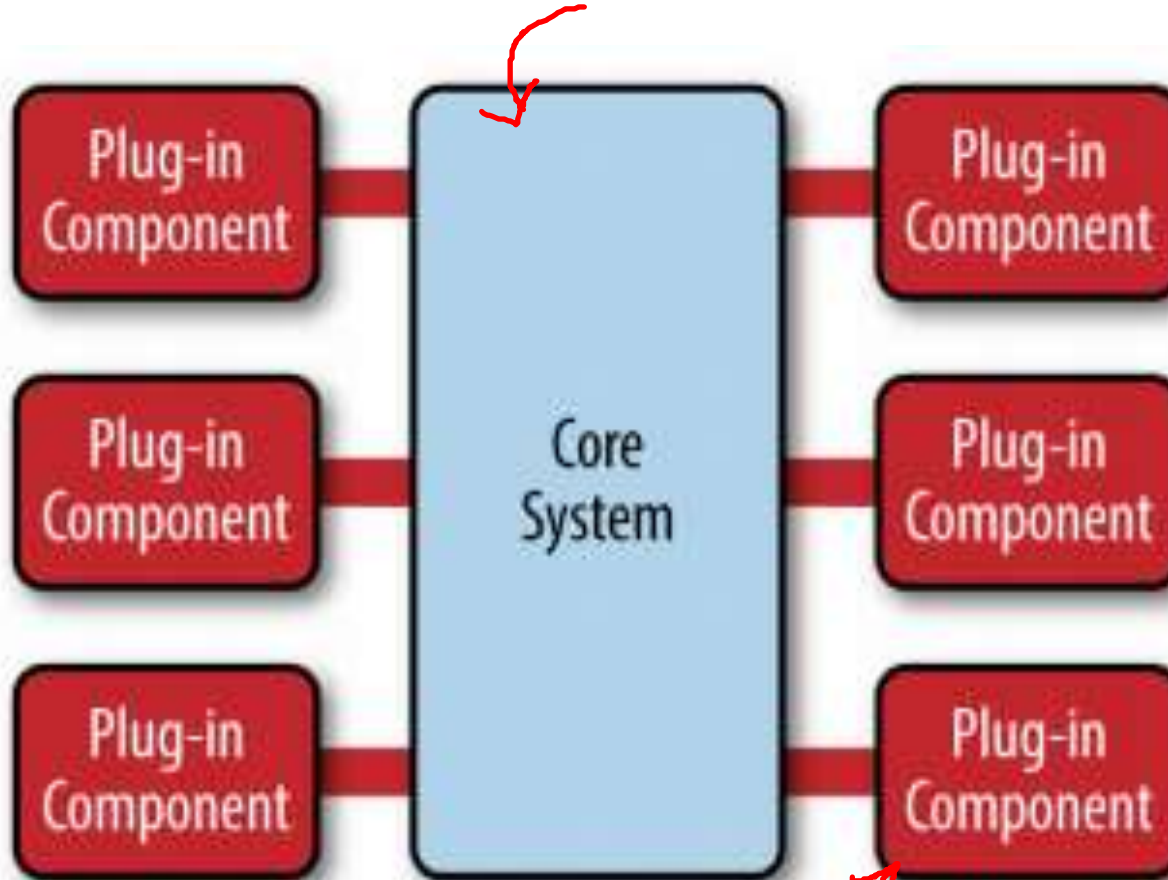
Microkernel Architecture

ideal for product-based applications where additional features can be integrated as plugins without modifying the core system.

- Plugin Architecture Pattern
- Natural for Product Based Apps
- Consists of:
 - Core System Provides essential services and the foundation of the application.
 - Plugins Extend functionality without altering the core system.
- Can be embedded in other patterns

Microkernel Architecture

Provides essential services and the foundation of the application



Extend functionality without altering the core system

Microkernel Pattern Analysis

- Overall Agility - High
- Ease of Deployment - High
- Testability - High
- Performance - High

- Scalability - Low
 - Use Cases:
IDE Plugins (Eclipse, IntelliJ)
E-Commerce Platforms (WooCommerce, Shopify)
Enterprise Applications (ERP Systems with Modular Features)
Game Engines (Unity, Unreal Engine)
- Ease of Development - Low

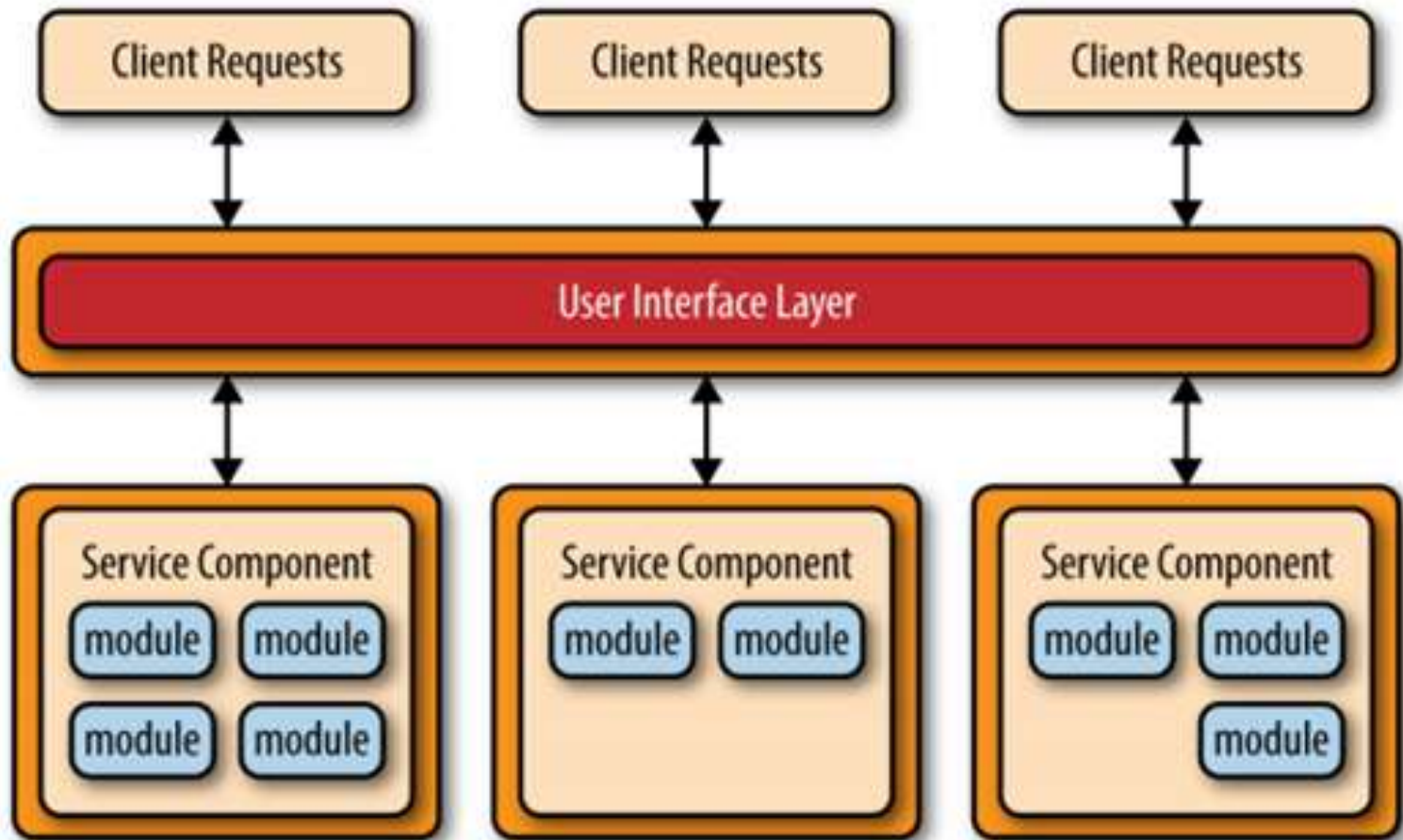
Micro services Architecture

Microservices Architecture is a modular approach to software development that structures an application as a collection of loosely coupled services. It evolved as an alternative to Monolithic Applications and Service-Oriented Architecture (SOA).

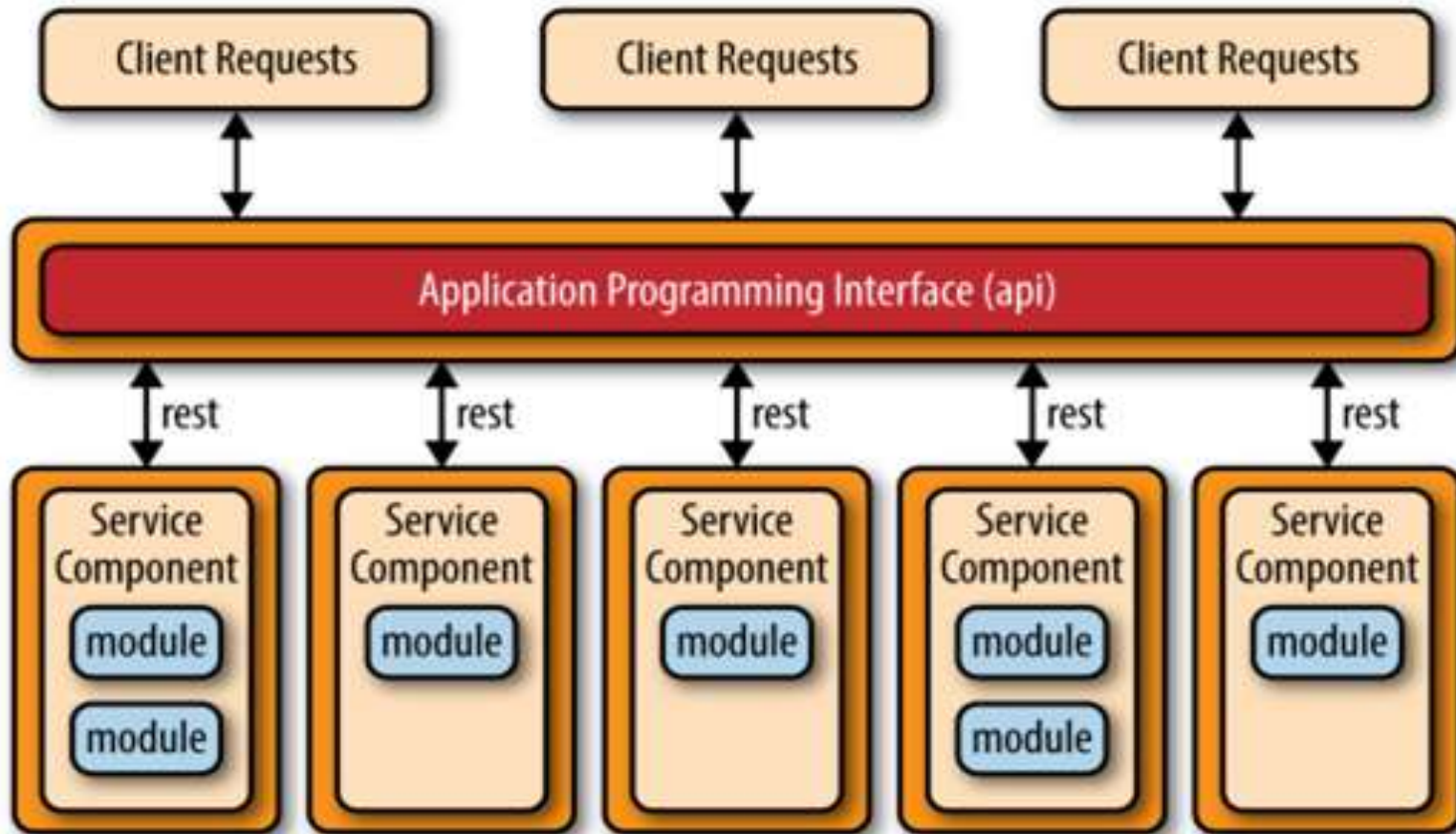
Micro services

- Evolved from other patterns
- Alternative to Monolithic Applications and SOA Architecture
- Still evolving
- Many ways to implement

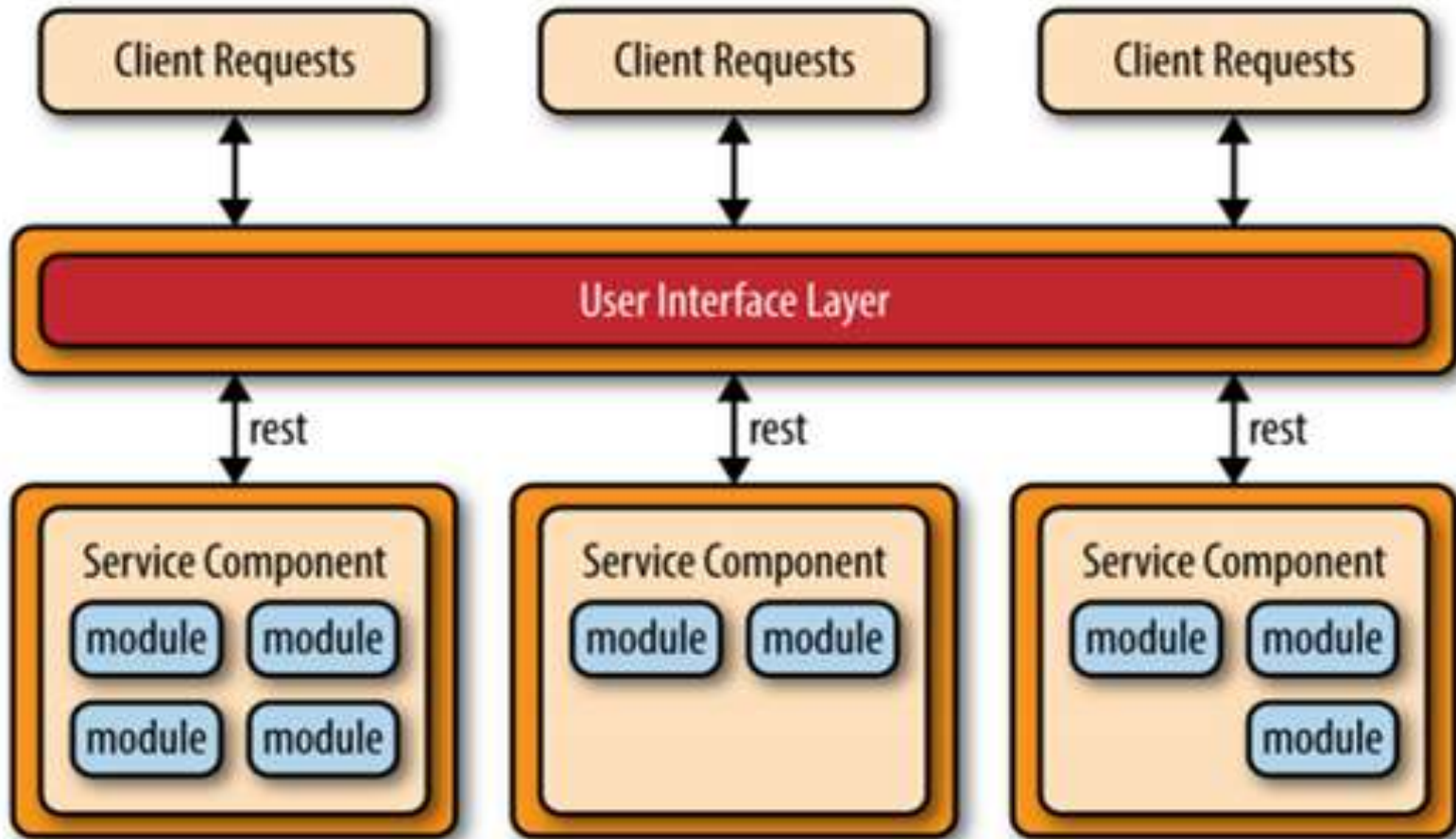
Basic Layout



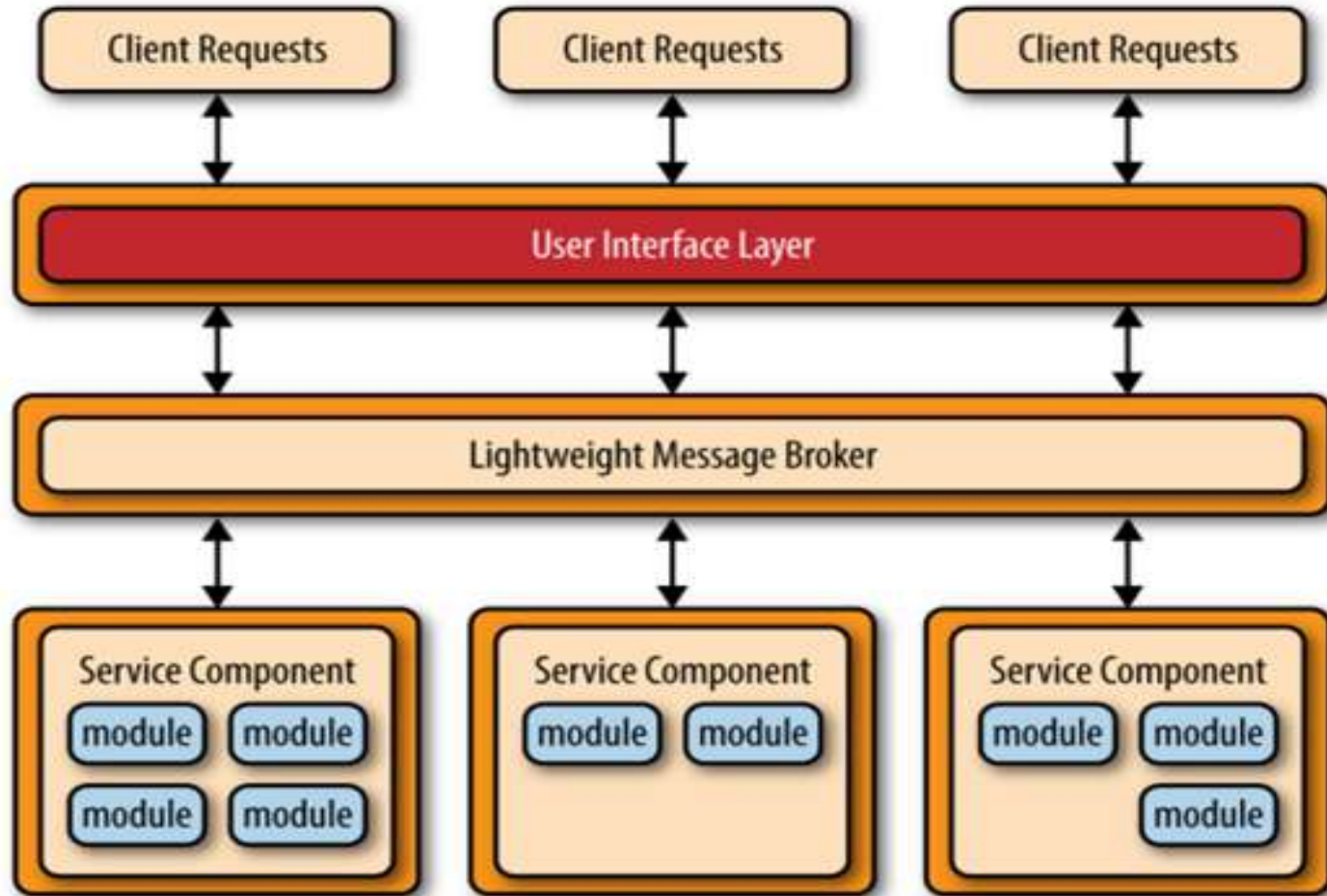
API REST-Based



Application REST-based



Centralized Messaging Topology



























Micro Services Architecture Pattern Analysis

- Overall Agility - High
- Ease of Deployment - High
- Testability - High
- Performance - Low
- Scalability - High
- Ease of Development - High

Pattern Comparison

Layered Pattern	<p>A solid general purpose pattern - best when you are not sure.</p> <p>Avoid the “Sinkhole Anti-Pattern”</p> <p>Tends to encourage Monolithic Applications</p>
Event-Driven	<p>Relatively complex. Distributed architectures issues must be addressed, such as remote processor availability, lack of responsiveness, reconnection logic, and failure recovery.</p> <p>No transactions across processors.</p> <p>Difficult to create and maintain processor contracts</p>
Microkernel	<p>Can be embedded and used within other patterns.</p> <p>Great support for evolutionary design and incremental development.</p> <p>Should always be the first choice for product-based applications</p>
Microservices	<p>Easy to perform real-time production deployments.</p> <p>Very agile-oriented architecture</p> <p>Shares complexity issues with data-driven pattern</p>

Pattern Comparison

	Layered	Event-driven	Microkernel	Microservices
Overall Agility				
Deployment				
Testability				
Performance				
Scalability				
Development				

The End