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)

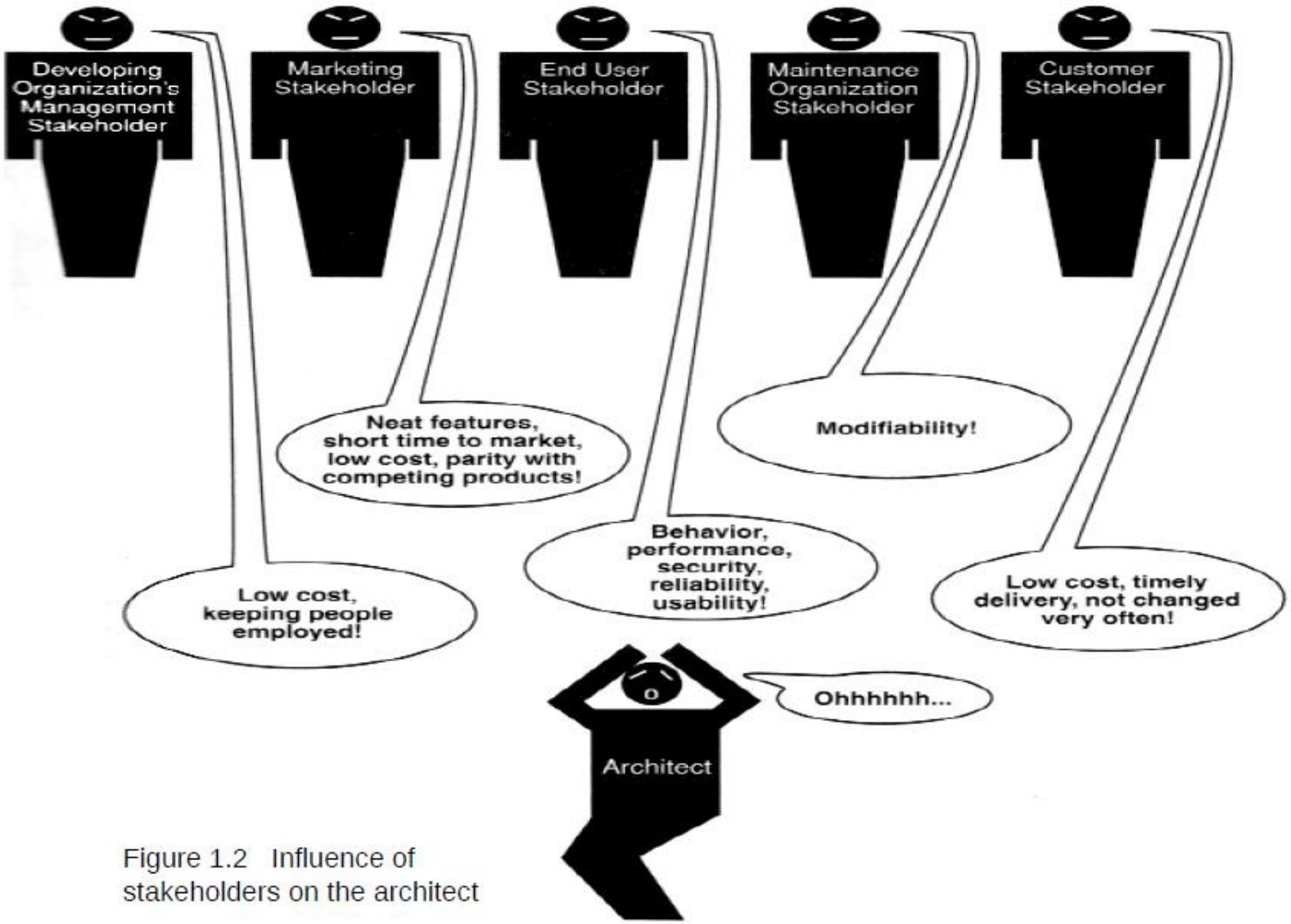


Figure 1.2 Influence of stakeholders on the architect

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

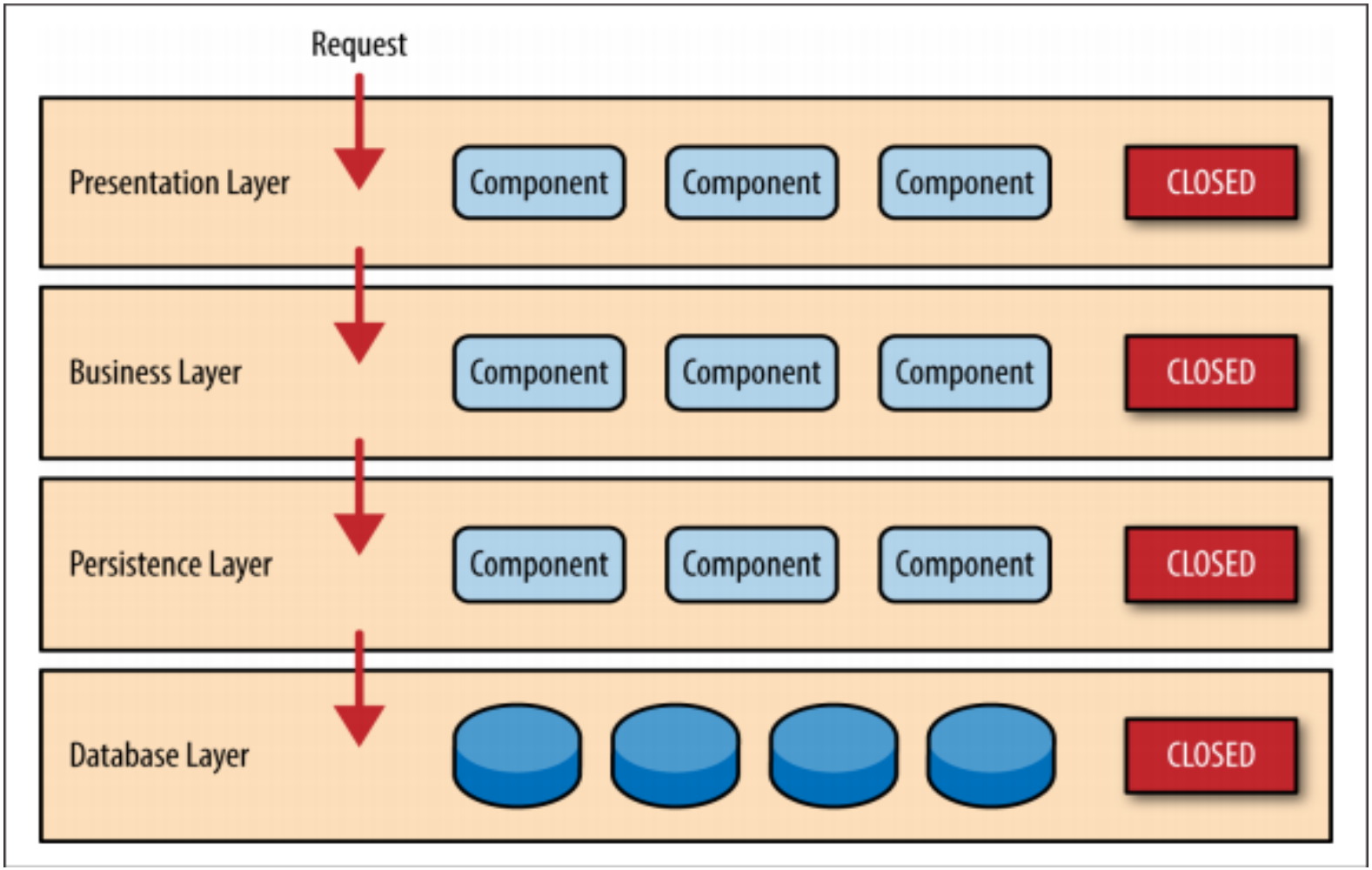
Why Use a Pattern?

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

Software Architecture Patterns

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

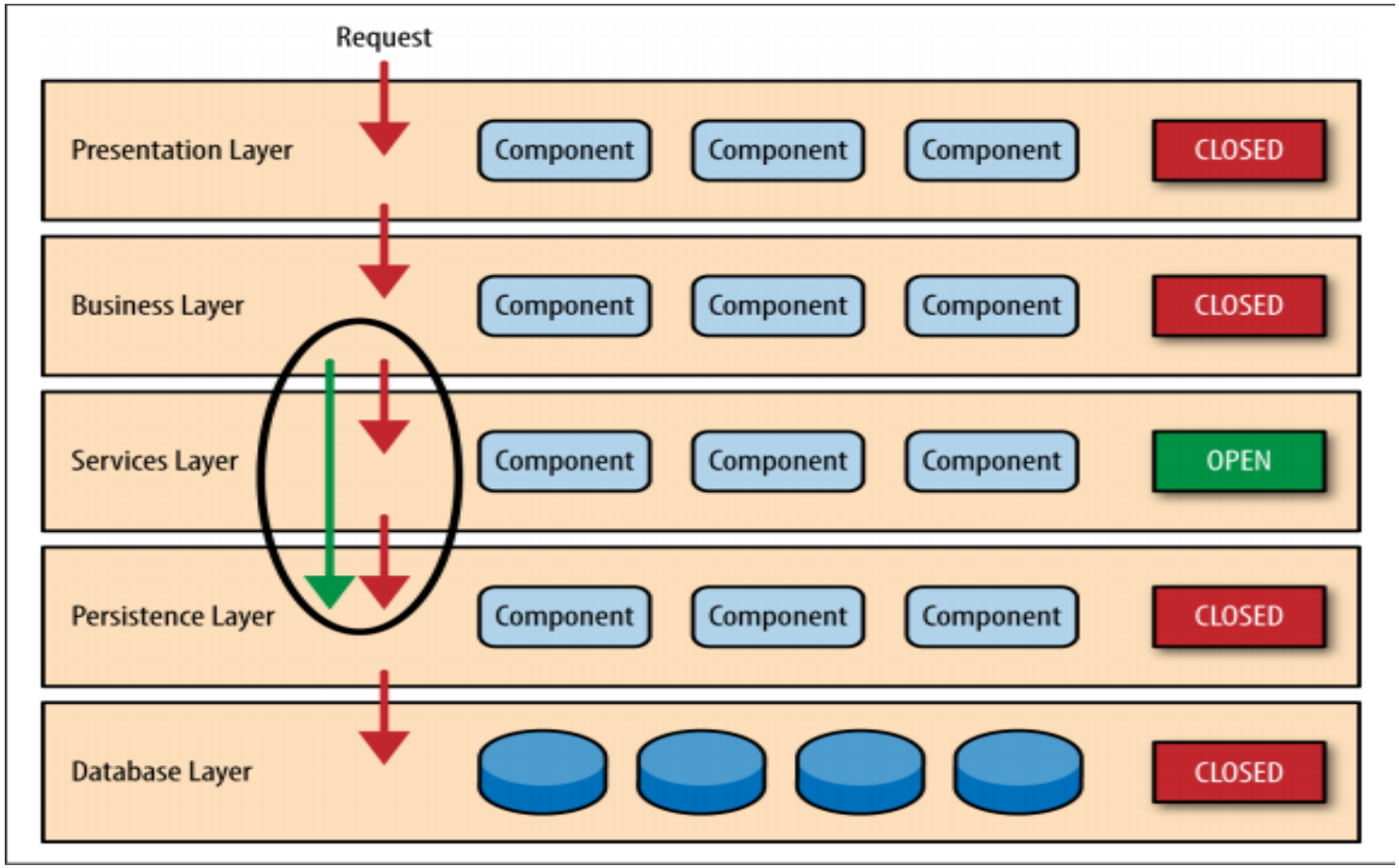
Layered Architecture



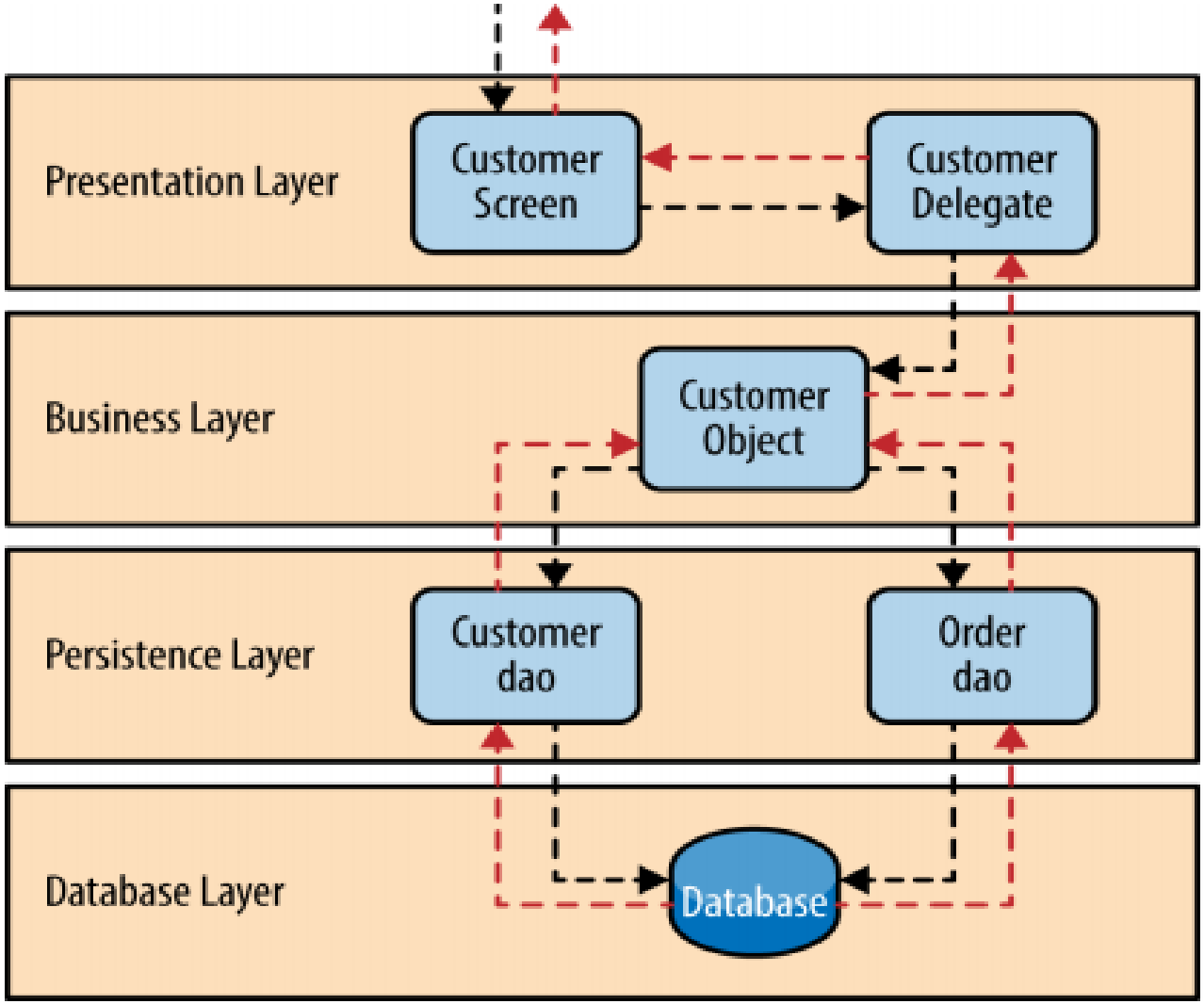
Key Concepts

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

Open/Closed Layers



Pattern Example



The Architecture Sinkhole Anti-Pattern

Requests flow through layers without processing

Layered Architecture Pattern Analysis

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

Event Driven Architecture

- Distributed
- Asynchronous
- Highly scalable
- Highly adaptable

Two main topologies:

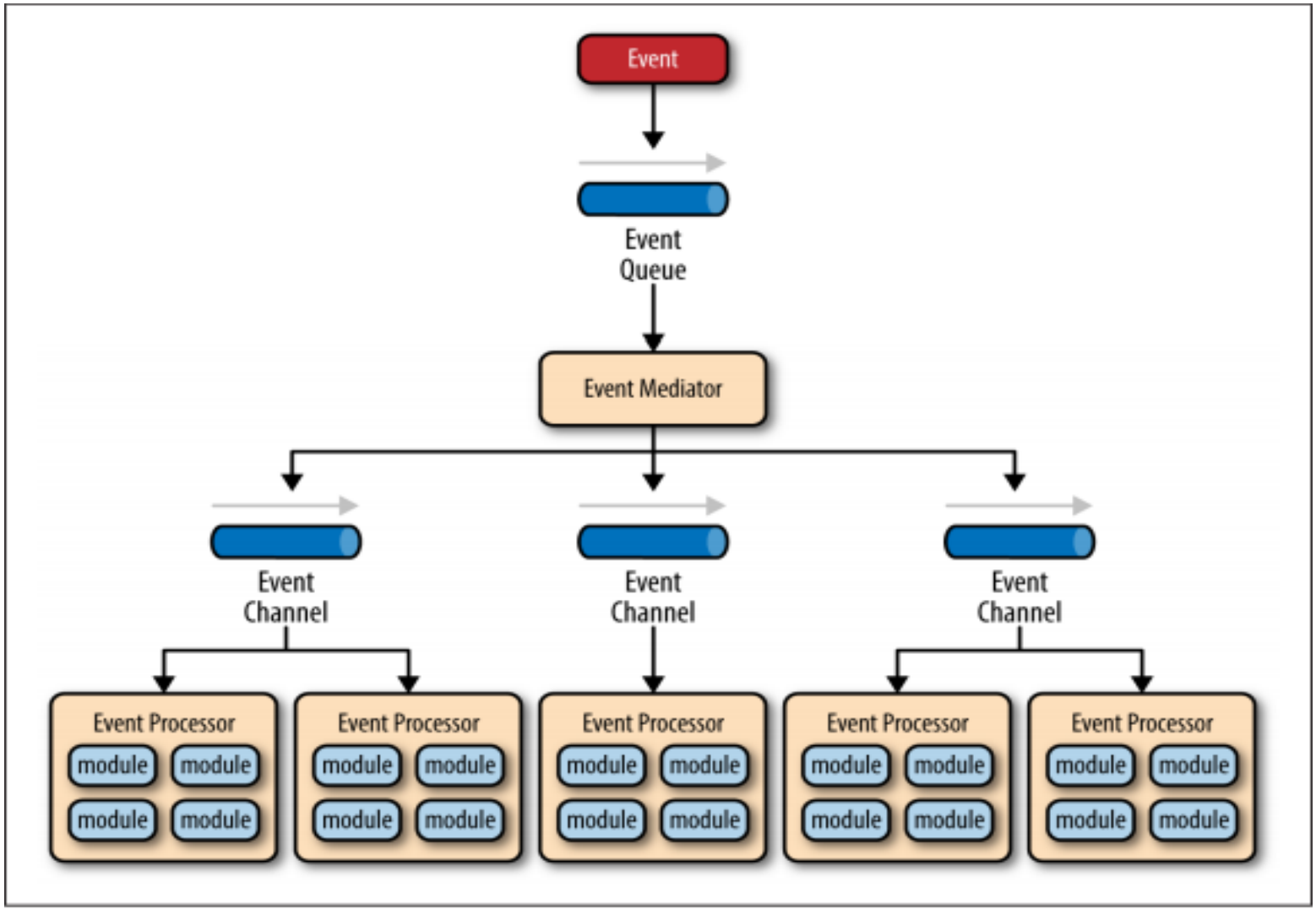
- Mediator
- Broker

Mediator Topology

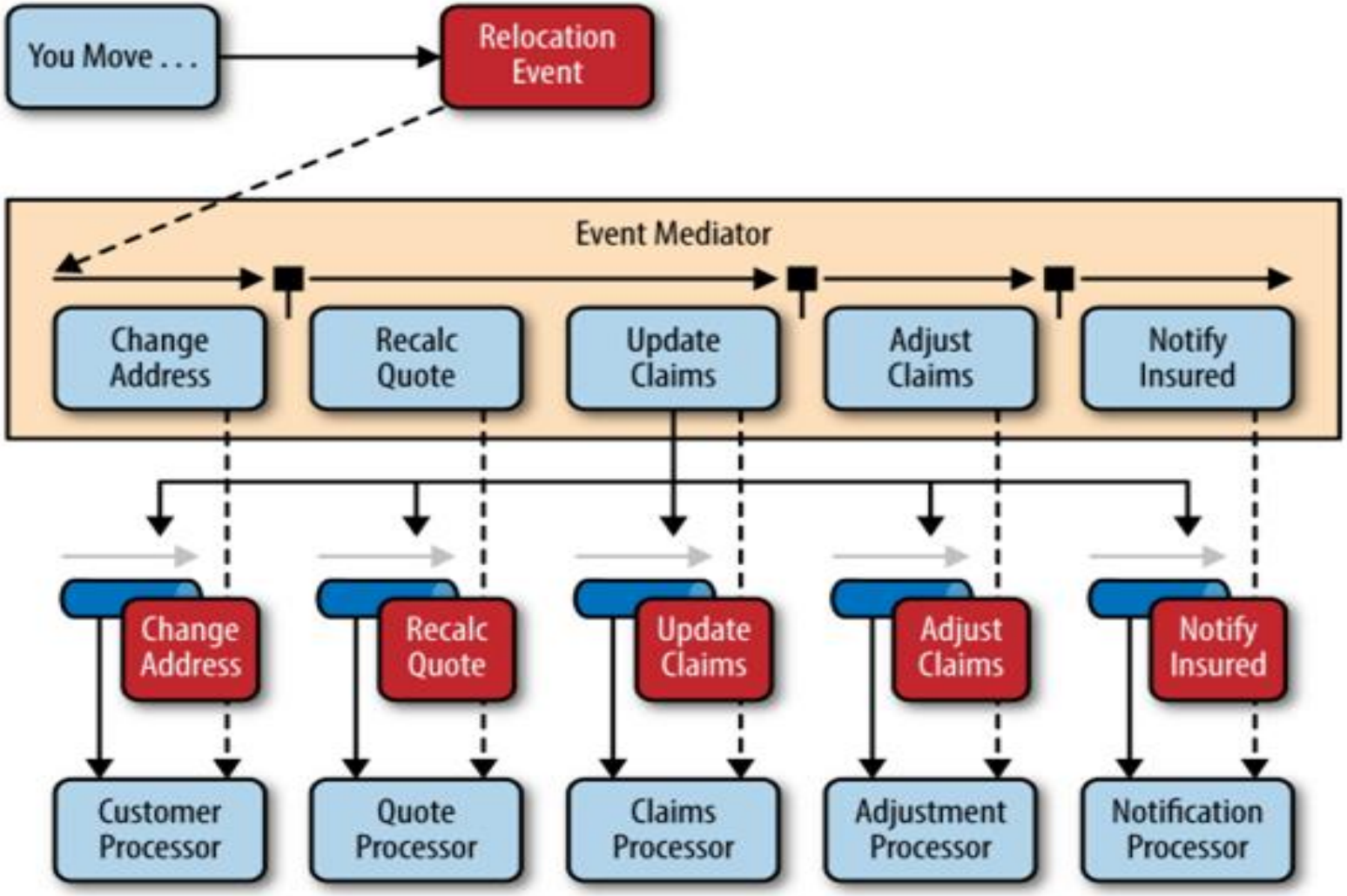
- Events processing have multiple steps that require orchestration

- Four main components:
  - Event Queues
  - Event Mediator
  - Event Channels
  - Event Processors

Mediator Topology



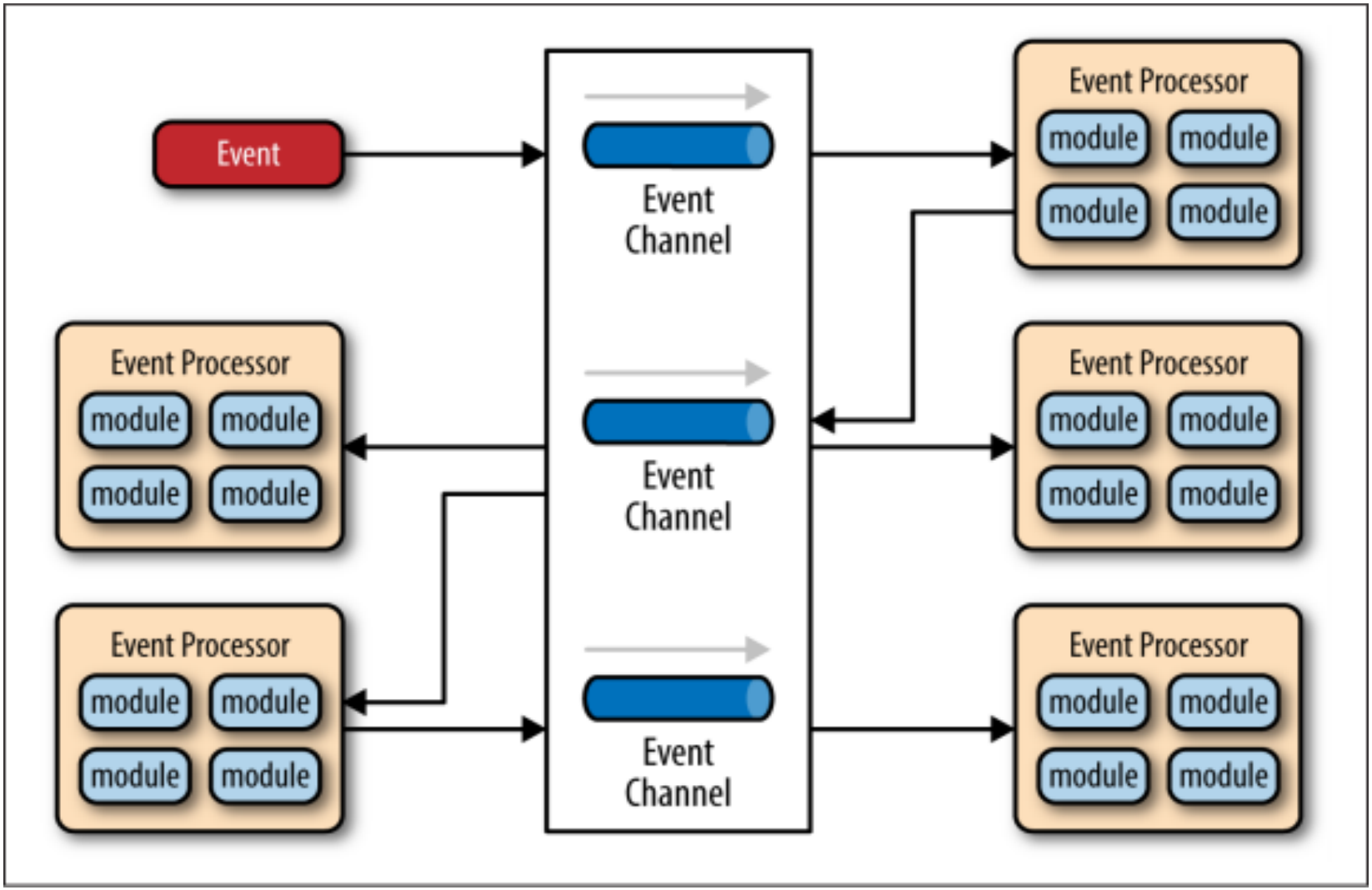
Mediator Topology Example



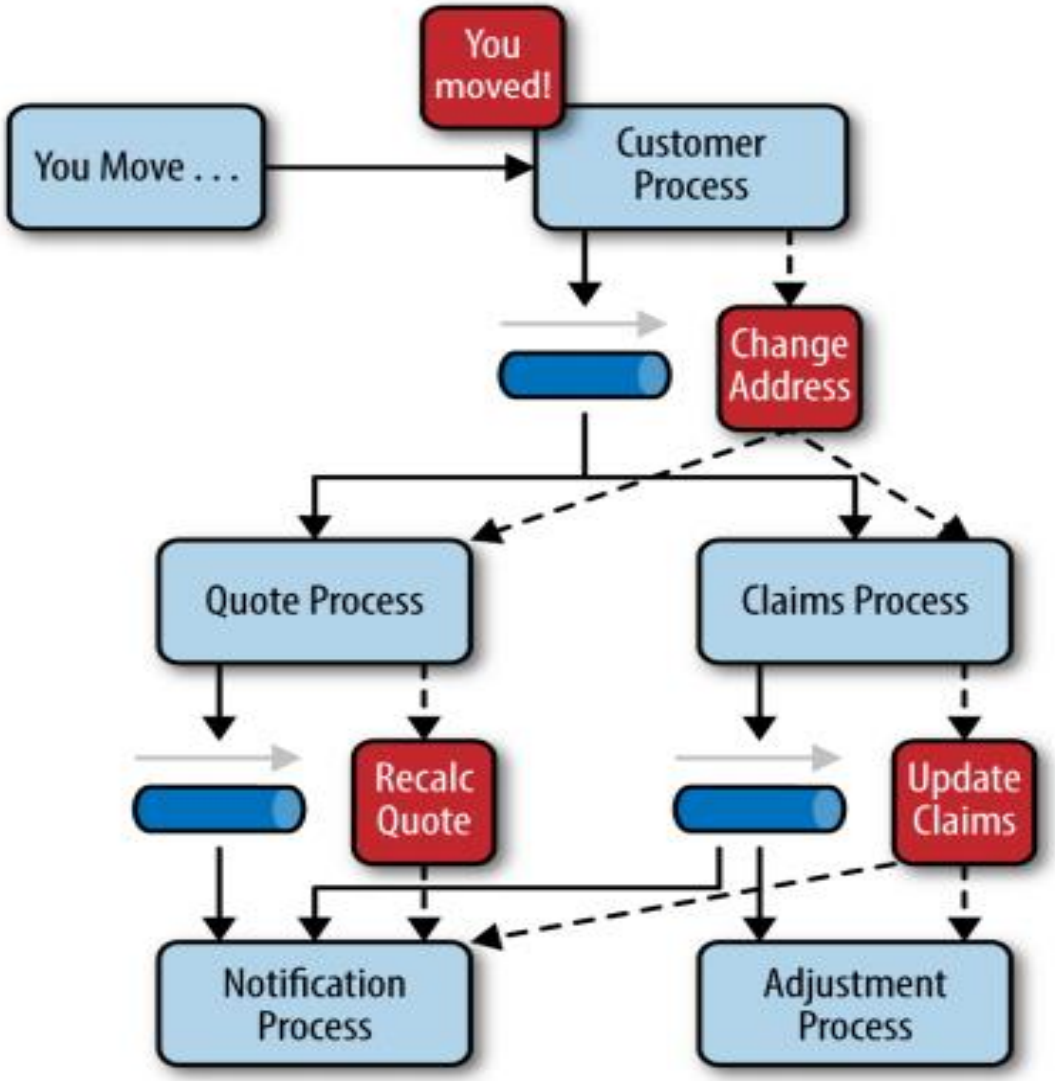
Broker Topology

- No central event mediator
- Message flows across processors in a chain like fashion
- Main components:
  - Message Broker
  - Event Processor

Broker Topology



Broker Topology Example

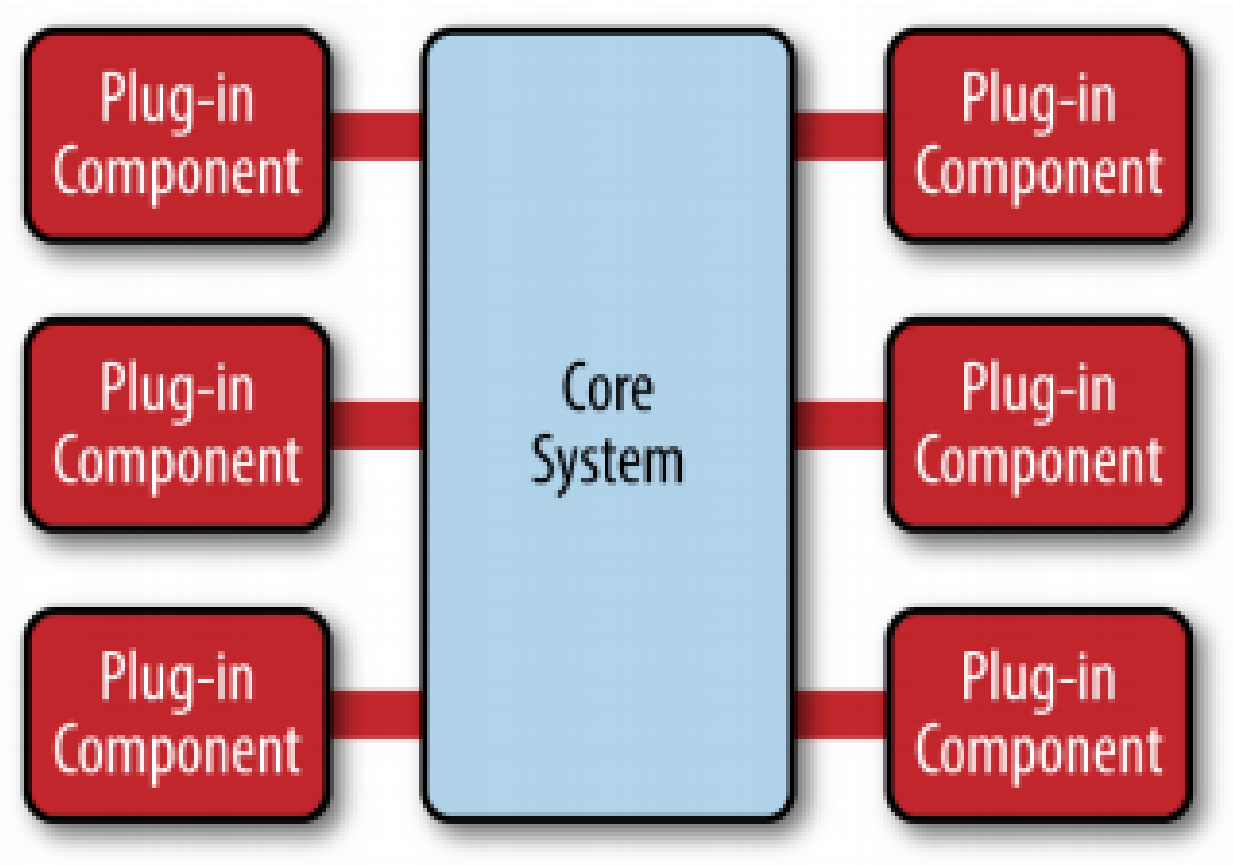


Microkernel Architecture

Microkernel Architecture

- Plugin Architecture Pattern
- Natural for Product Based Apps
- Consists of:
  - Core System
  - Plugins
- Can be embedded in other patterns

Microkernel Architecture



Microkernel Pattern Analysis

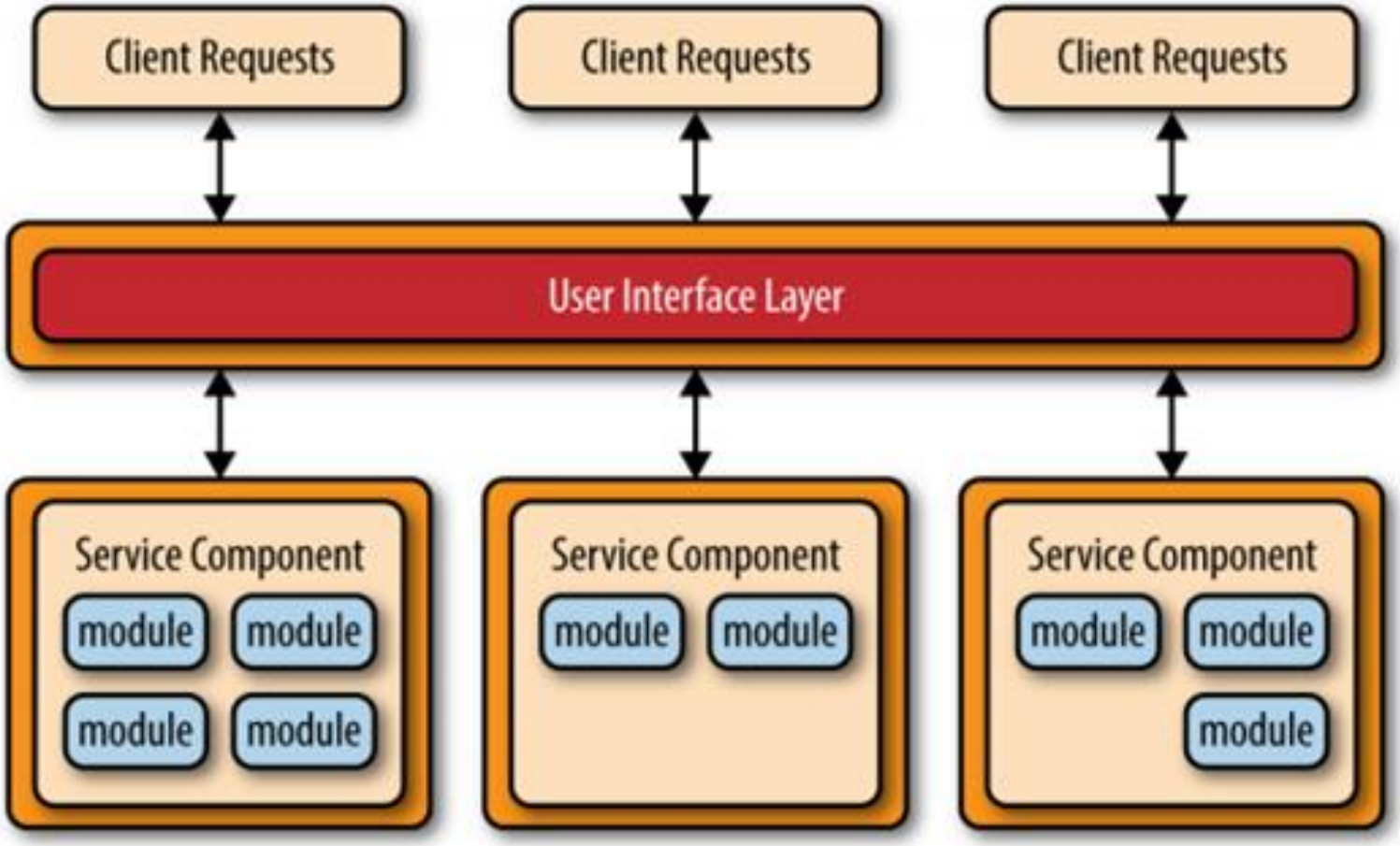
- Overall Agility - High
- Ease of Deployment - High
- Testability - High
- Performance - High
- Scalability - Low
- Ease of Development - Low

Micro services Architecture

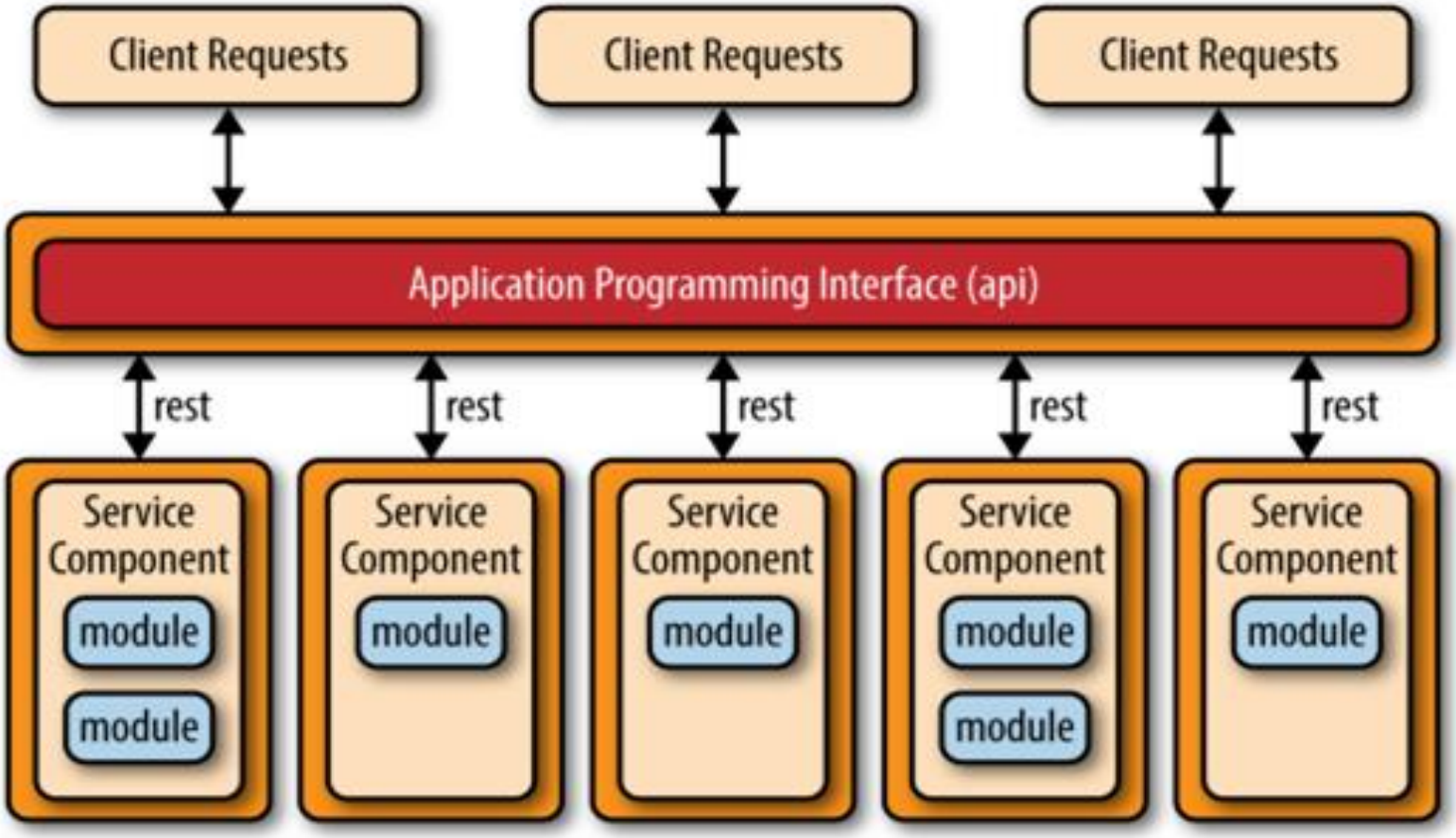
Micro services

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

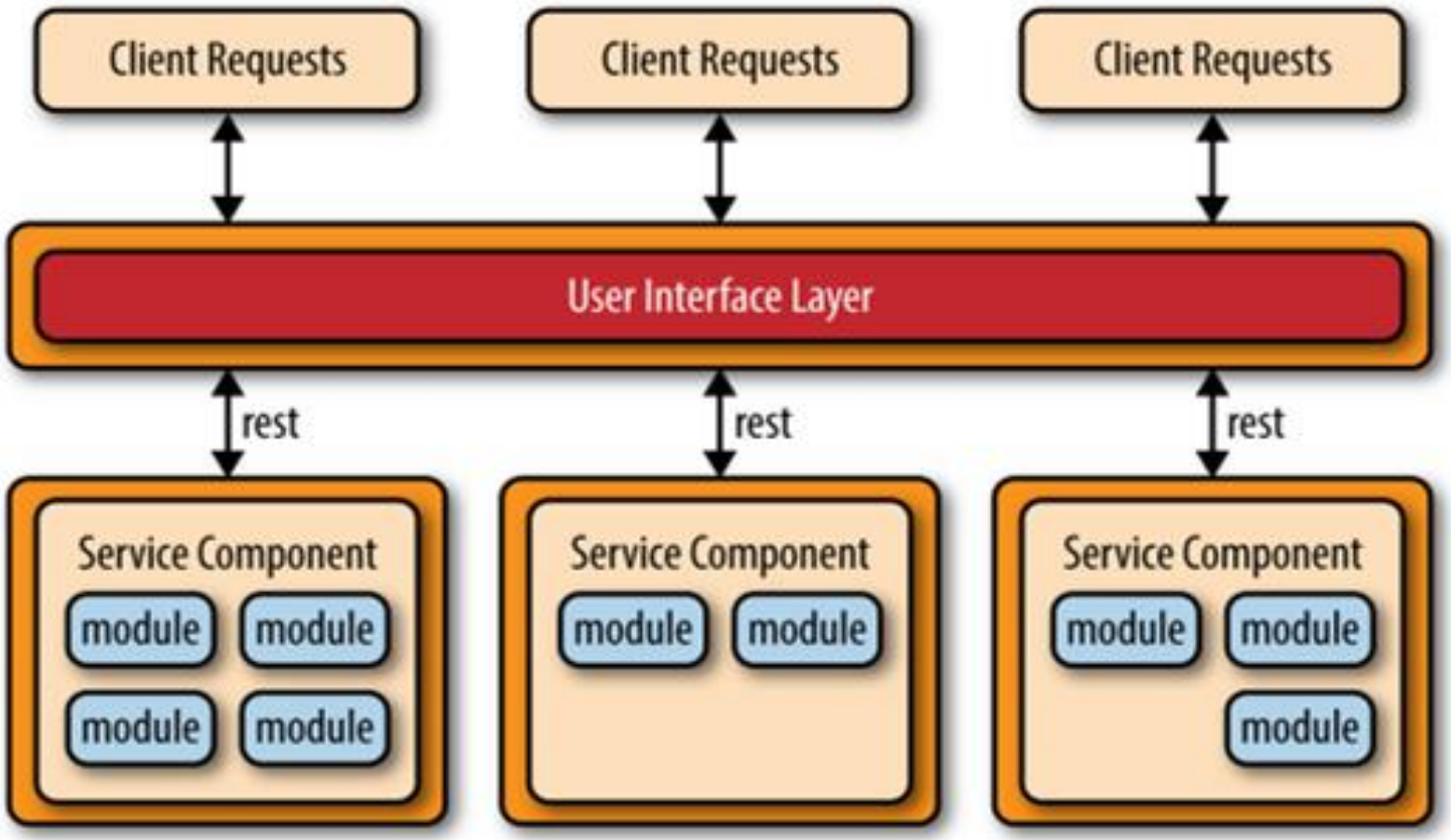
Basic Layout



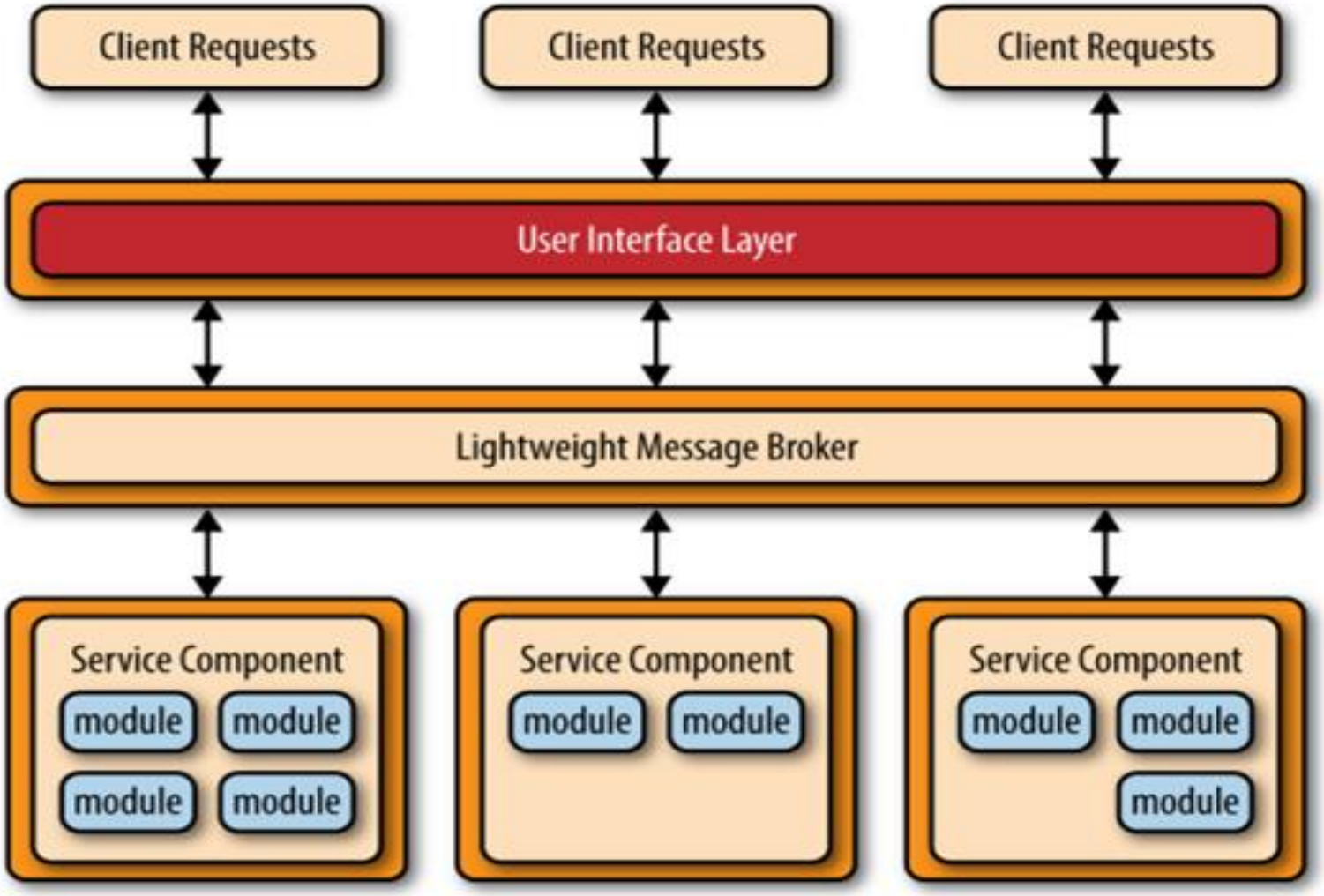
API REST-Based



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

Pattern Comparison

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

The End