

**Spring 2023**



# **Software Architecture Styles/Patterns**

**Seonah Lee**

**Gyeongsang National University**



# Architecture Style







# Architecture Style





# Architecture Style



# 목 차

- ▶ 소프트웨어 아키텍처 스타일
- ▶ 소프트웨어 아키텍처 패턴
- ▶ 아키텍처 스타일 **vs.** 패턴
- ▶ 진화하는 소프트웨어 아키텍처 스타일
- ▶ 소프트웨어 아키텍처 설계에서의 활용

# 소프트웨어 아키텍처 스타일

- ▶ 스타일
- ▶ 정의
- ▶ 예제

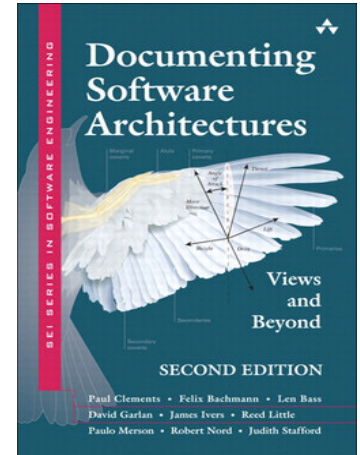
# Software Architecture Style

## ▶ 스타일

- ▶ 소프트웨어 구조의 유형
- ▶ **SW** 시스템의 설계 작업의 기초를 제공

## ▶ 스타일의 4가지의 스키마

- ▶ 구성요소들 (**A set of element types**)
- ▶ 구성요소들의 관계와 구성방식 (**a topological layout of elements**)
- ▶ 구성요소들의 정확한 의미와 한계 (**a set of semantic constraints**)
- ▶ 구성방식에 맞춰 구성요소들이 상호작용하는 메커니즘 (**a set of interaction mechanism**)



# Software Architecture Style

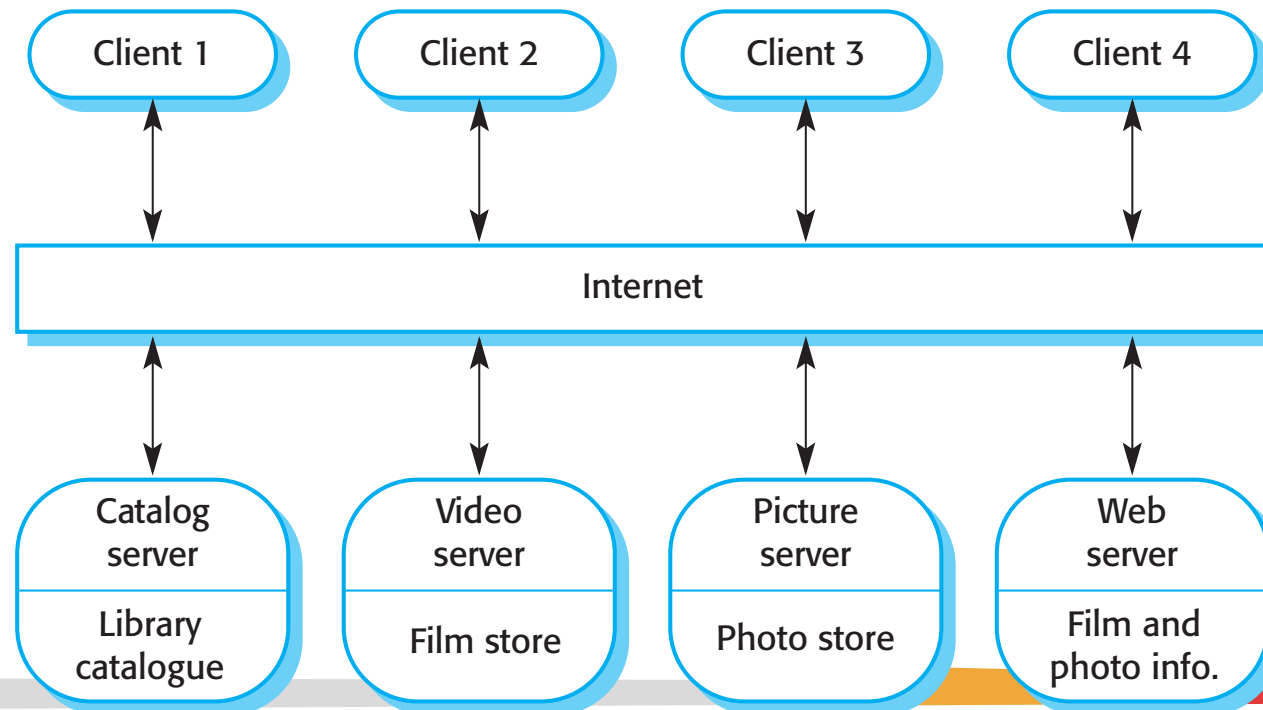
## ▶ Style Definition

- ▶ An architectural style is a named collection of architectural design decisions that
  - ▶ (1) are applicable in a given development **context**
  - ▶ (2) constrain architectural **design decision** that are specific to a particular system within that context
  - ▶ (3) elicit **beneficial qualities** in each resulting system



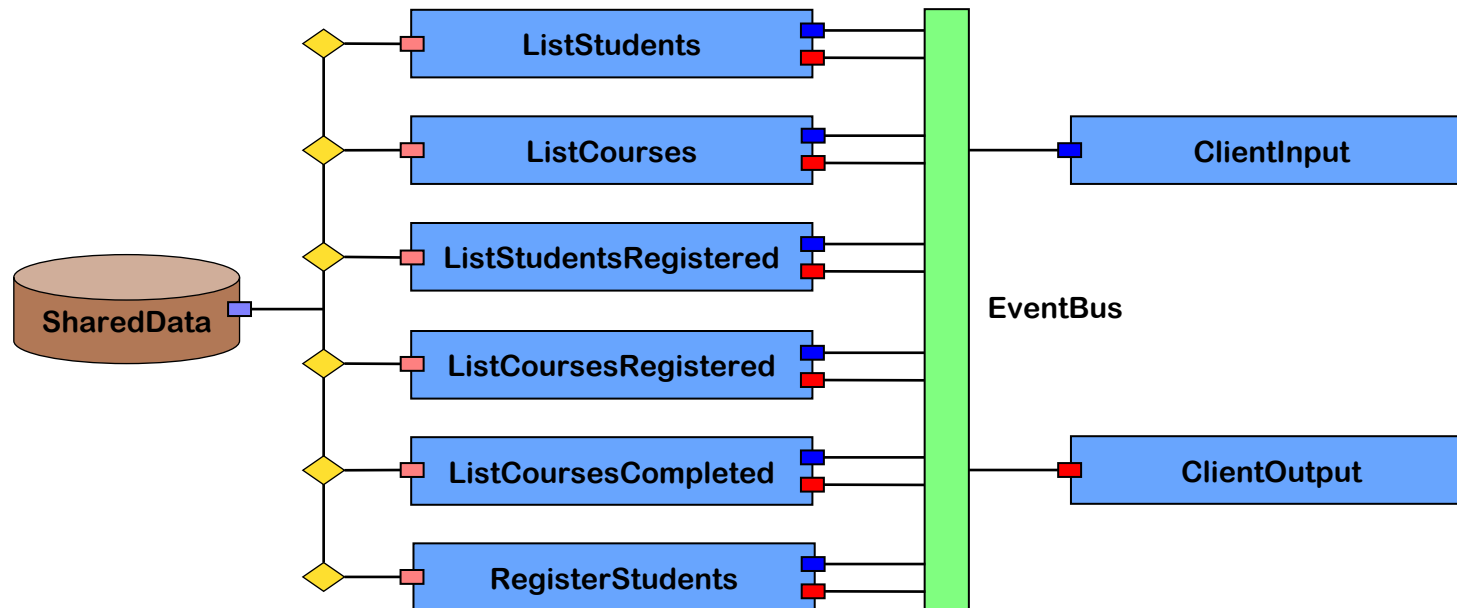
# Example: Client-Server Style

- ▶ 서비스(혹은 데이터)의 생산자, 소비자 간 결합 분리; 비대칭적 통신
  - ▶ 클라이언트: 통신 시작, 서비스 요청 및 사용
  - ▶ 서버: 하나 이상의 인터페이스를 통해 요청받은 서비스(기능 혹은 자원) 제공



# Example: Event-based Style

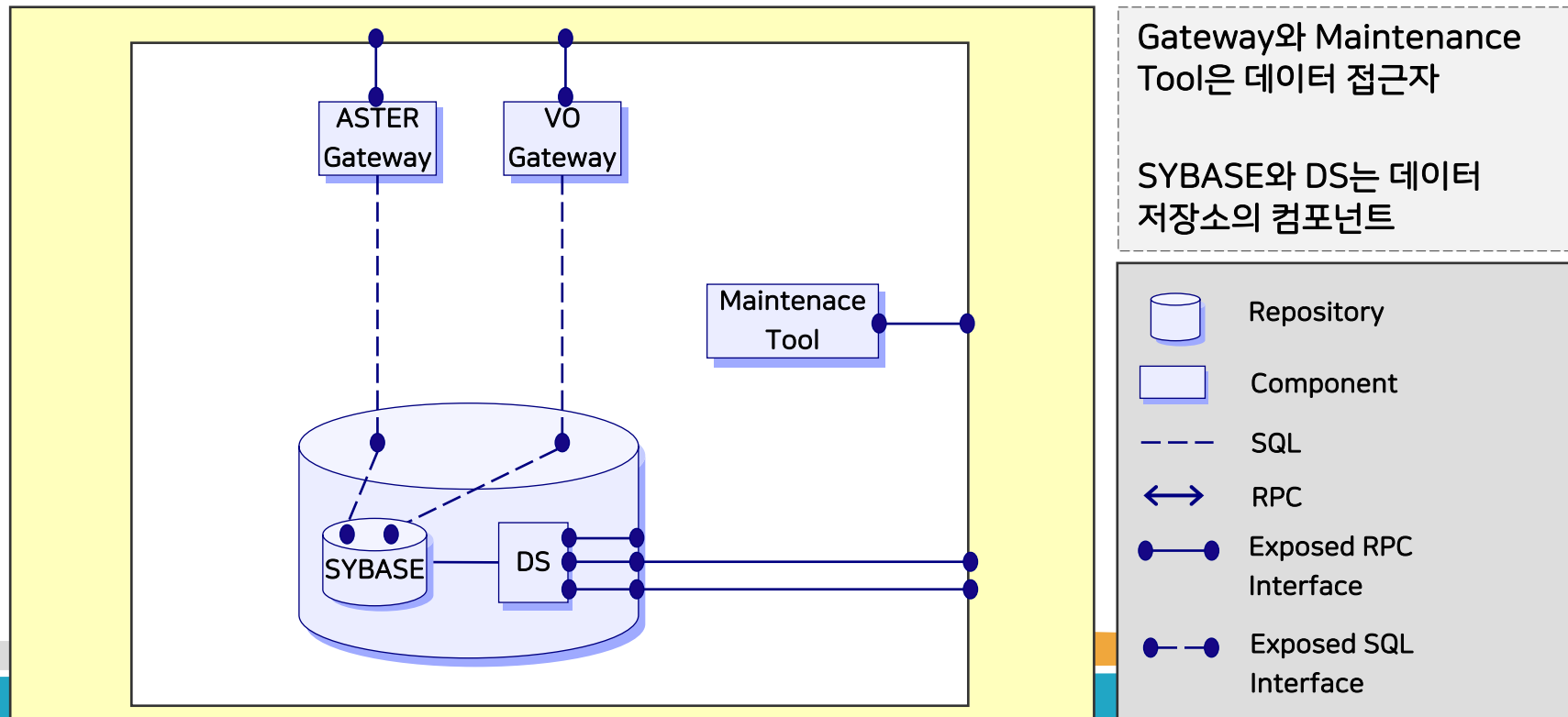
- ▶ 컴포넌트는 통보된 이벤트를 통해 상호작용
  - ▶ 이벤트 생산자: 이벤트를 생성하여 통보하는 컴포넌트
  - ▶ 이벤트 소비자: 이벤트를 받아 처리하는 컴포넌트



# Example:

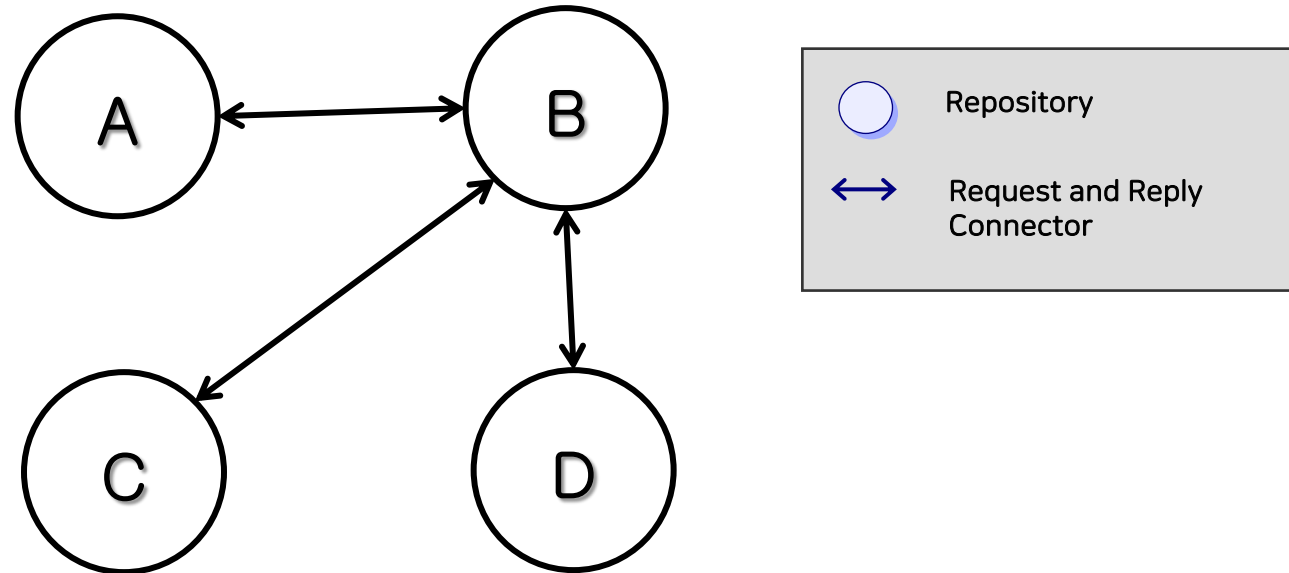
## Shared-Data Style

- ▶ 상호작용 패턴은 지속성 데이터 교환에 의해 통제
  - ▶ 공유 데이터 저장소: 데이터 소비자가 접근할 데이터 관리
  - ▶ 데이터 소비자: 데이터에 접근



# Example: Peer-to-Peer

- ▶ 서비스(혹은 데이터)의 생산자, 소비자 간 결합 분리; 대칭적 통신
  - ▶ 피어: 클라이언트와 서버의 역할을 동시에 수행



# Software Architecture Style

## ▶ 피어 투 피어 스타일 예

요소	<ul style="list-style-type: none"><li>컴포넌트 타입: 피어(peer).</li><li>커넥터 타입: 프로시저 호출(involve procedure).</li></ul>
관계	<ul style="list-style-type: none"><li>부착(attachment) 관계는 피어(peer)를 프로시저 호출(involve procedure) 커넥터와 연관시킴</li><li>가능한 컴포넌트 상호작용 그래프를 결정.</li></ul>
연산 모델	<ul style="list-style-type: none"><li>피어는 인터페이스를 제공하고 상태를 캡슐화.</li><li>계산은 서로에게 서비스를 요청하는 협력 피어들에 의해 수행.</li></ul>
속성	<ul style="list-style-type: none"><li>컴포넌트-커넥터 뷰타입에 정의된 것과 동일하고 상호작용 프로토콜과 성능-지향 요소를 강조.</li><li>부착은 실행 시간에 변경 가능.</li></ul>
토폴로지	<ul style="list-style-type: none"><li>주어진 포트나 역할에 가능한 부착의 수를 제한 가능.</li><li>컴포넌트가 다른 컴포넌트에 대해 알 수 있는가에 대한 제약사항인 가시성 제한이 가해질 수 있음.</li></ul>



# 소프트웨어 아키텍처 패턴

- ▶ 패턴
- ▶ 정의
- ▶ 예제

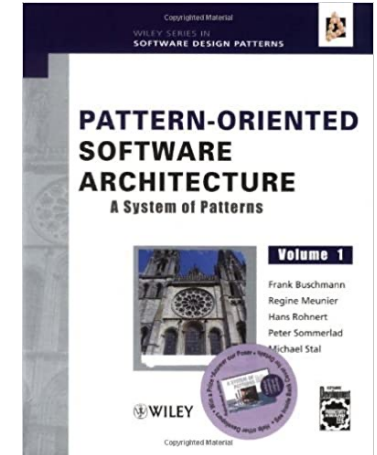
# Software Architecture Pattern

## ▶ 패턴

- ▶ 특정 문제에 대한 해법을 추상화
- ▶ 해법 내의 공통된 요인을 추출하여 정형화한 것을 패턴이라고 함

## ▶ 패턴의 3가지의 스키마

- ▶ **정황(Context)** : 문제를 발생시키는 상황
- ▶ **문제(Problem)** : 해당 정황에서 반복적으로 발생하는 문제
- ▶ **해법(Solution)** : 해당 문제에 대해 검증된 해답





# Software Architecture Pattern



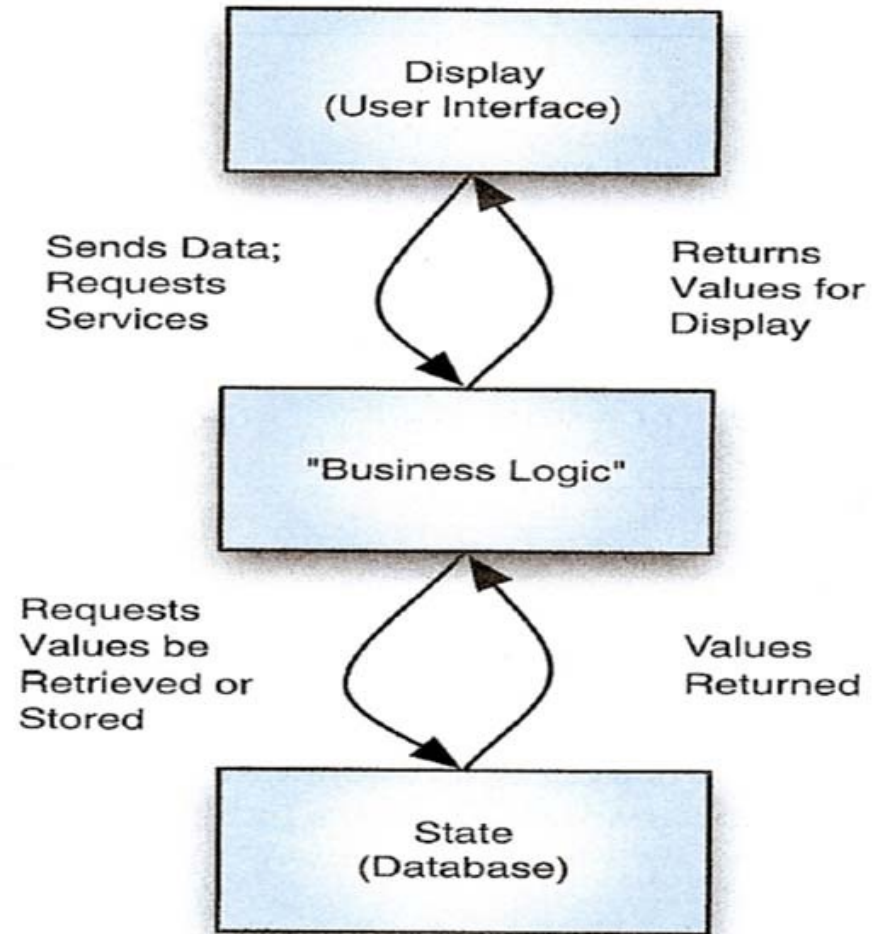
## ▶ Pattern 정의

- ▶ 다양한 상황에서 소프트웨어 아키텍처 수립하는 방식을 정형화한 것
- ▶ 아키텍처 설계에서 반복해서 나타나는 문제를 해결하고 아키텍처가 만족시켜야 하는 시스템 품질속성을 달성할 수 있는 방법을 문서로 정리

# Software Architecture Pattern

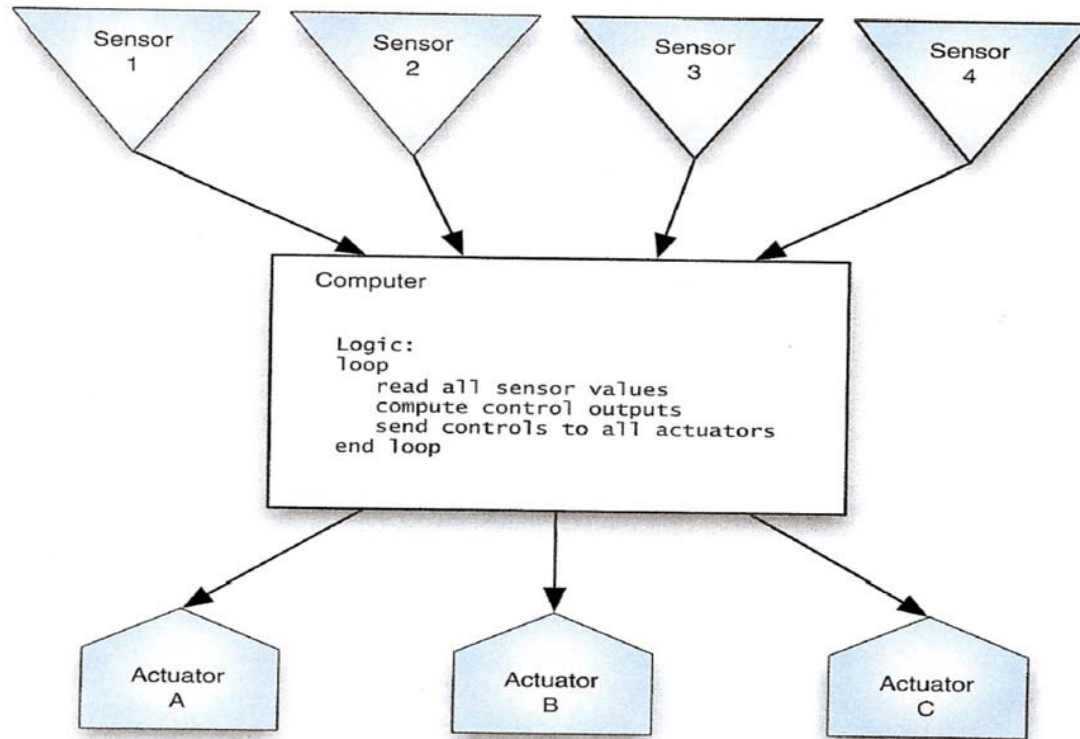
## ▶ State-Logic-Display(3-tier) 패턴

- ▶ 비즈니스 애플리케이션을 개발할 때 가장 일반적으로 사용되는 패턴
- ▶ 사용자 인터페이스(UI)와 비즈니스 로직, 데이터를 구분하여 변경 용이성이 좋음
- ▶ 게임, 웹 어플리케이션 등 많은 분야에서 사용되고 있음



# Software Architecture Pattern

## ▶ Sense-Compute-Control 패턴 예



- ▶ 임베디드 애플리케이션을 개발할 때 주로 사용되는 패턴
- ▶ 내제되는 일정한 시간 별로 센서의 값을 읽어들이는 **Sense**
- ▶ 센서의 값을 계산하여 해야할 행위를 정의한 **compute**
- ▶ **actuator**에 해야 할 기능이나 행위를 전달하는 **control**로 모듈을 구분하는 패턴



# ○○ 소프트웨어 아키텍처 스타일 **vs.** 패턴 ○○

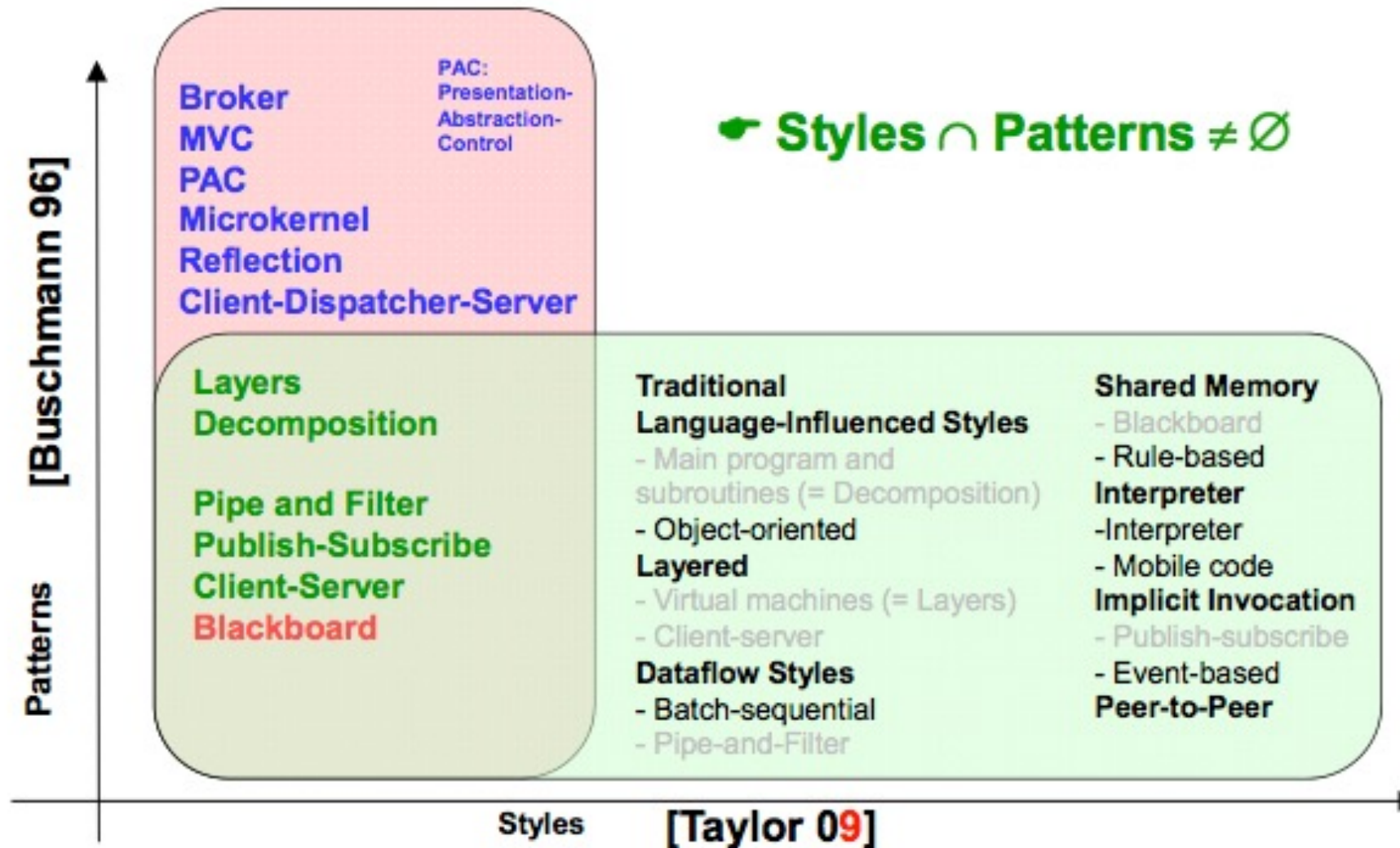


▶ 스타일 vs. 패턴

# Style vs. Pattern

아키텍처 스타일	아키텍처 패턴
구조적인 표현에 초점	문제와 연관하여 구조적 솔루션 제시
품질 속성과 관련하여 어떠한 좋은 점이 있는지를 내용적으로 기술	특정 도메인과의 연관성을 명시적으로 드러내는 경우가 많음

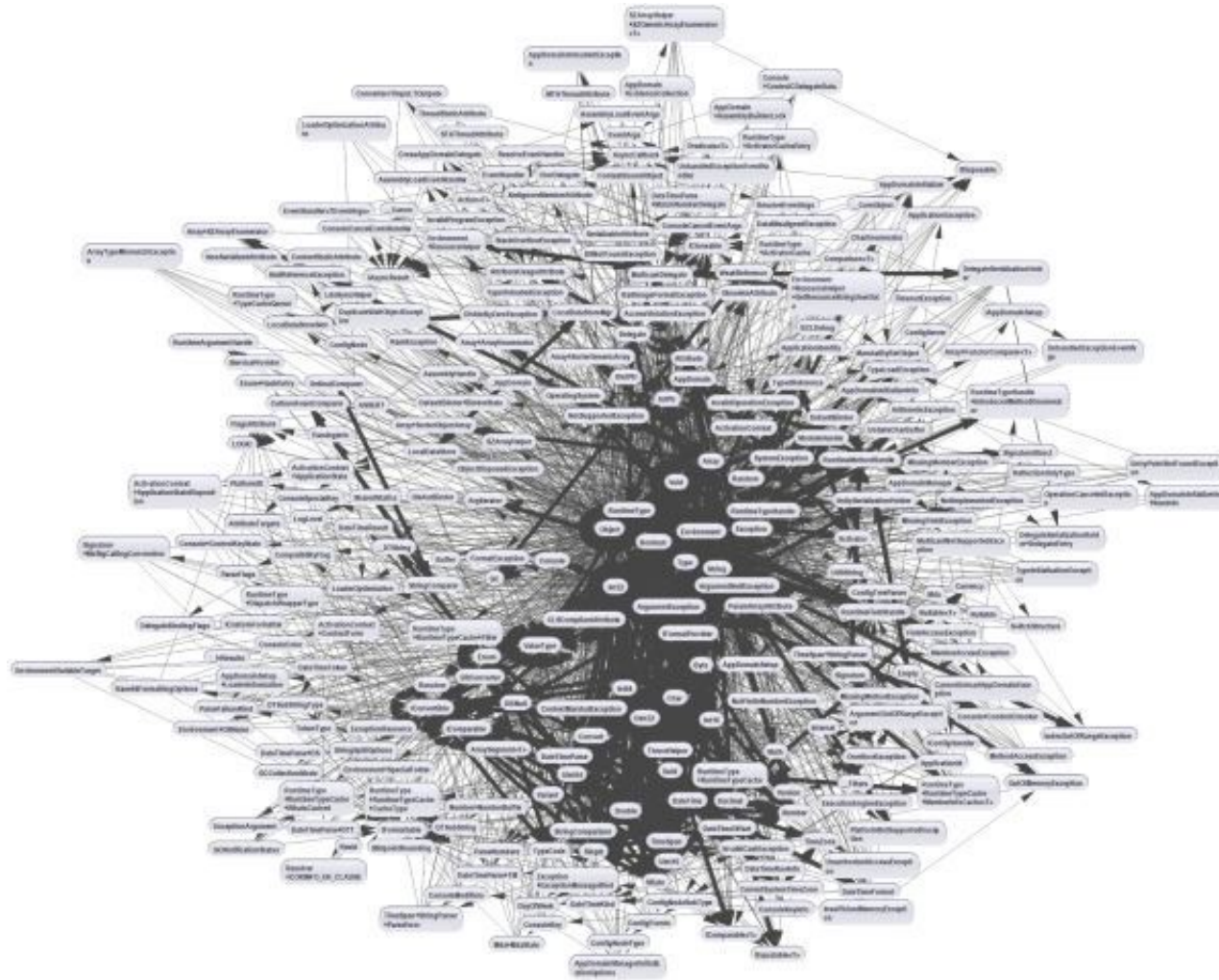
# Style vs. Pattern



# 진화하는 소프트웨어 아키텍처 스타일

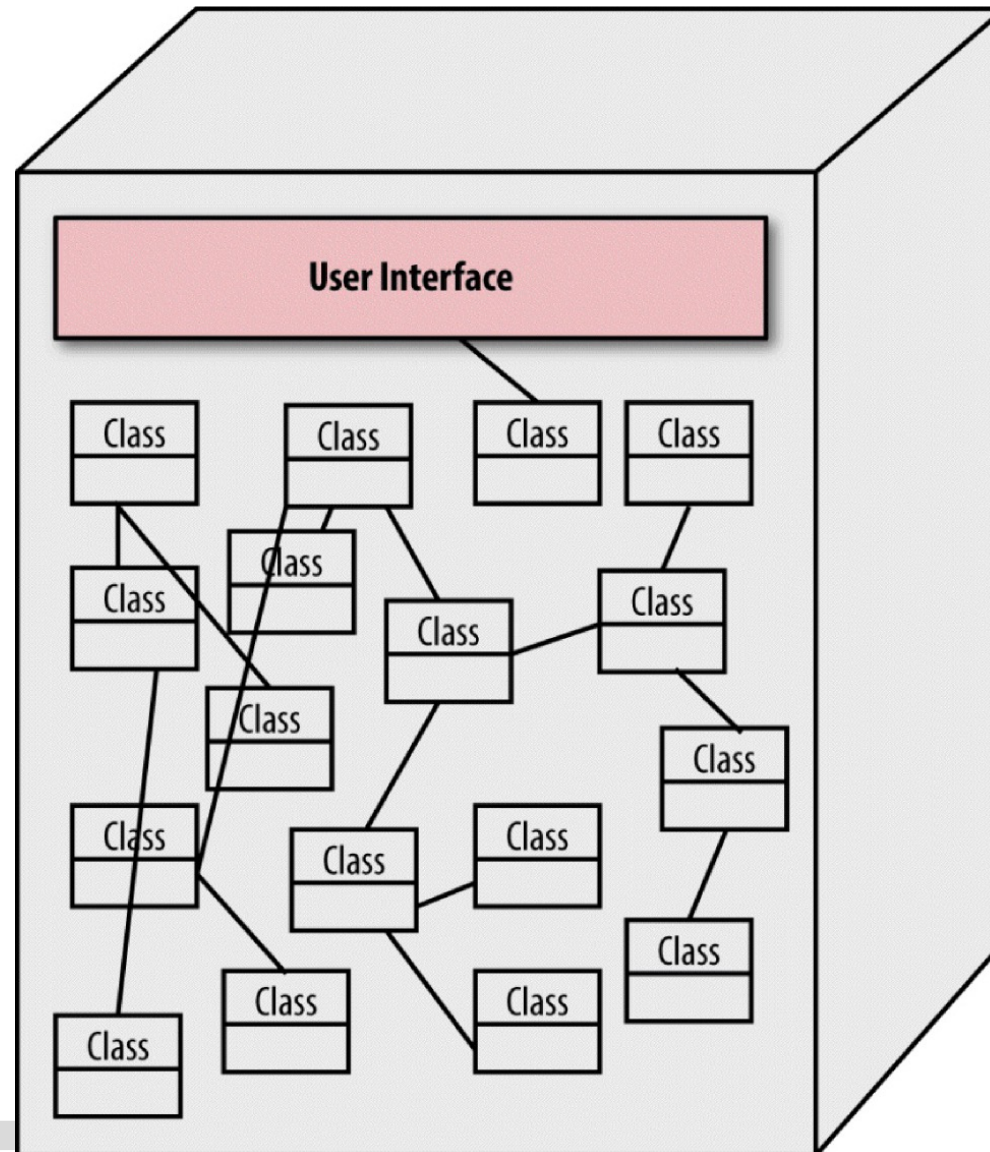
- ▶ Big Ball of Mud
- ▶ Monoliths
- ▶ Event-Driven Architectures
- ▶ Service-Oriented Architectures
- ▶ Serverless Architectures

# Big Ball of Mud

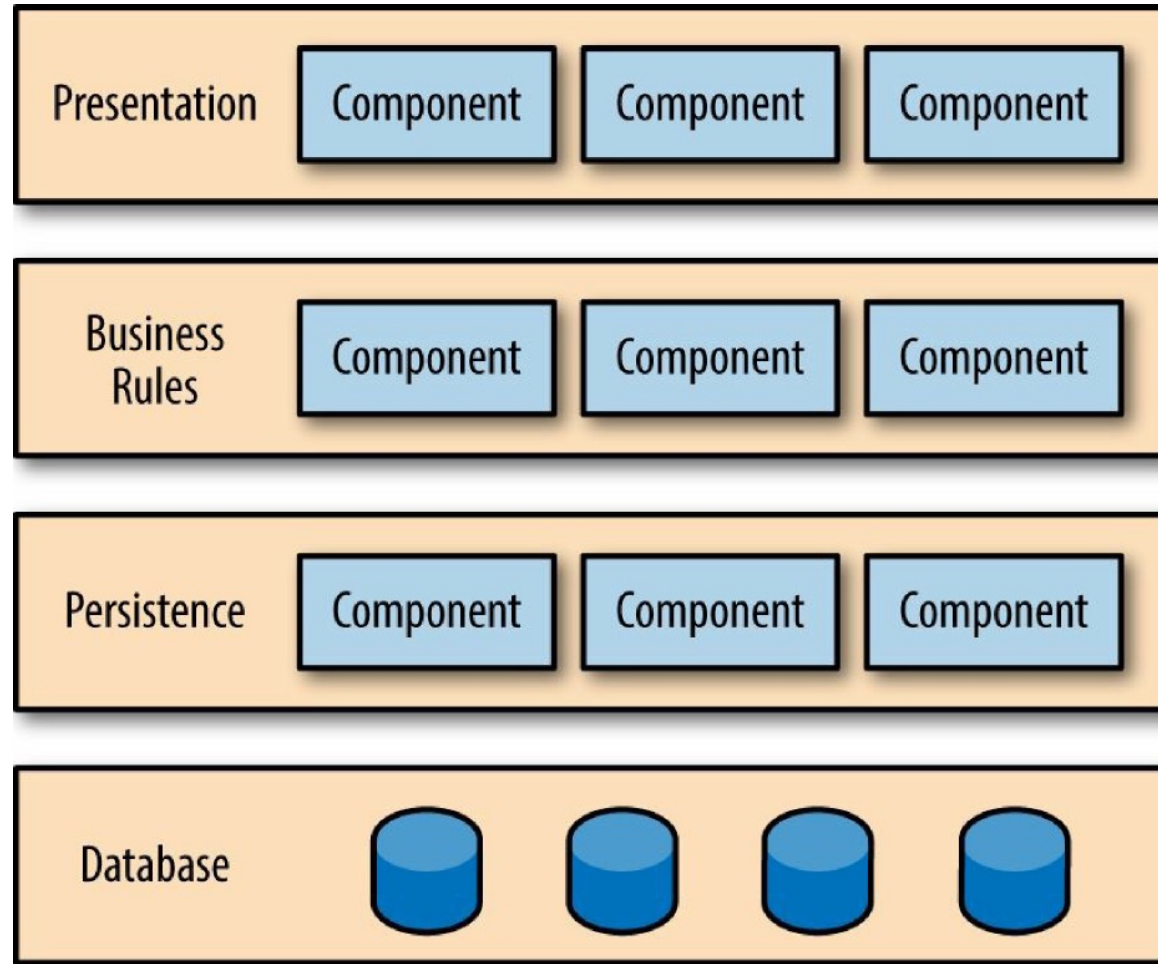




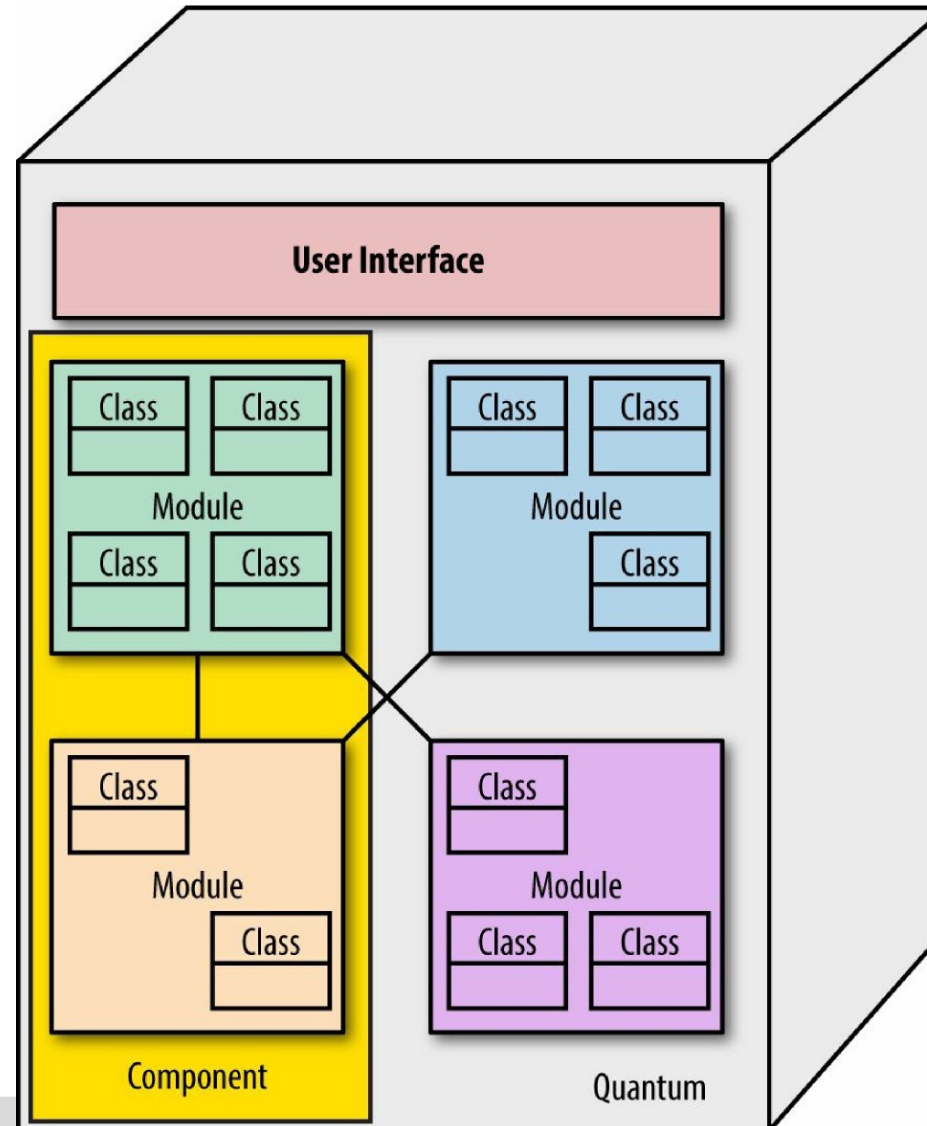
# Monoliths



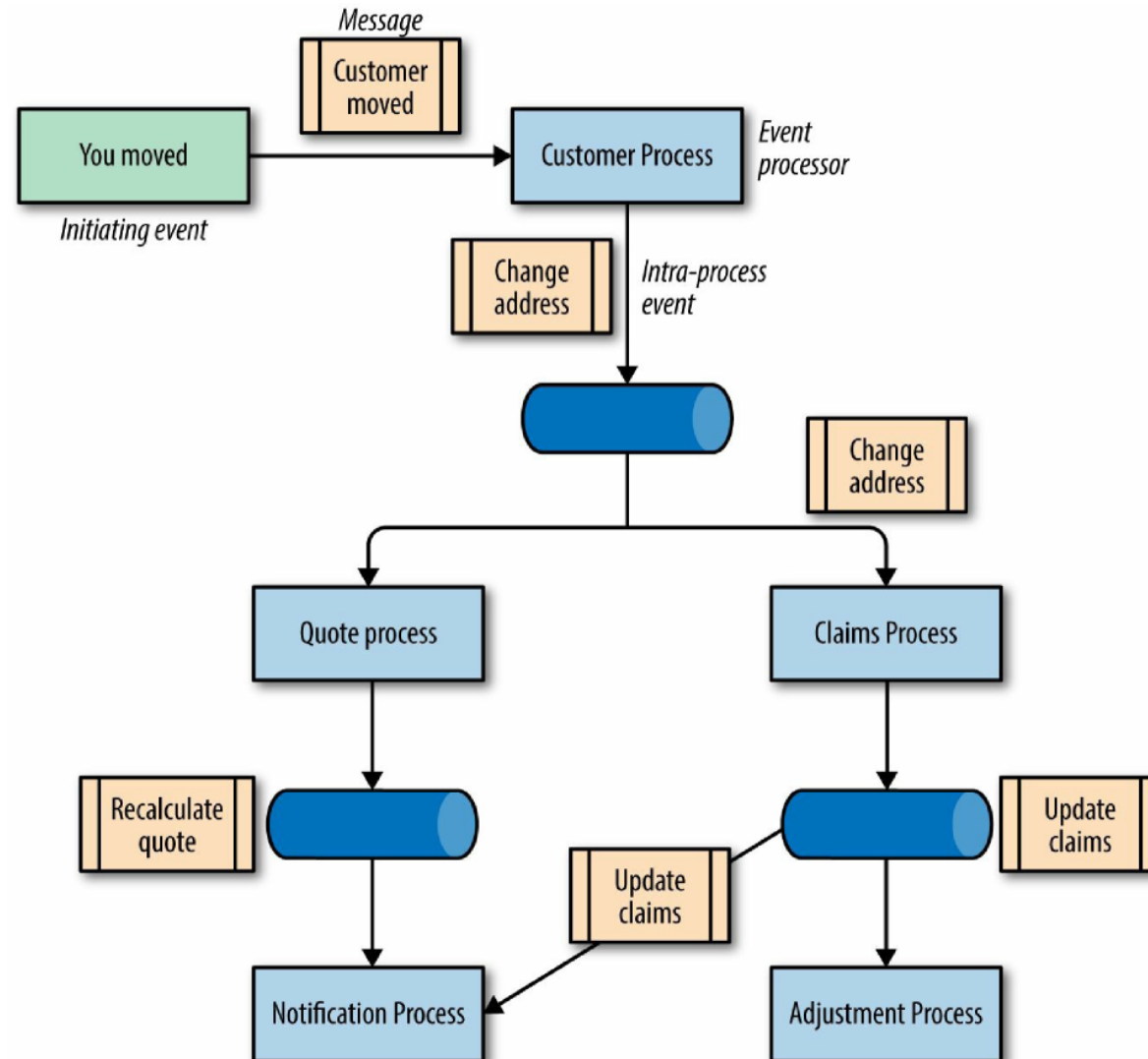
# Monoliths



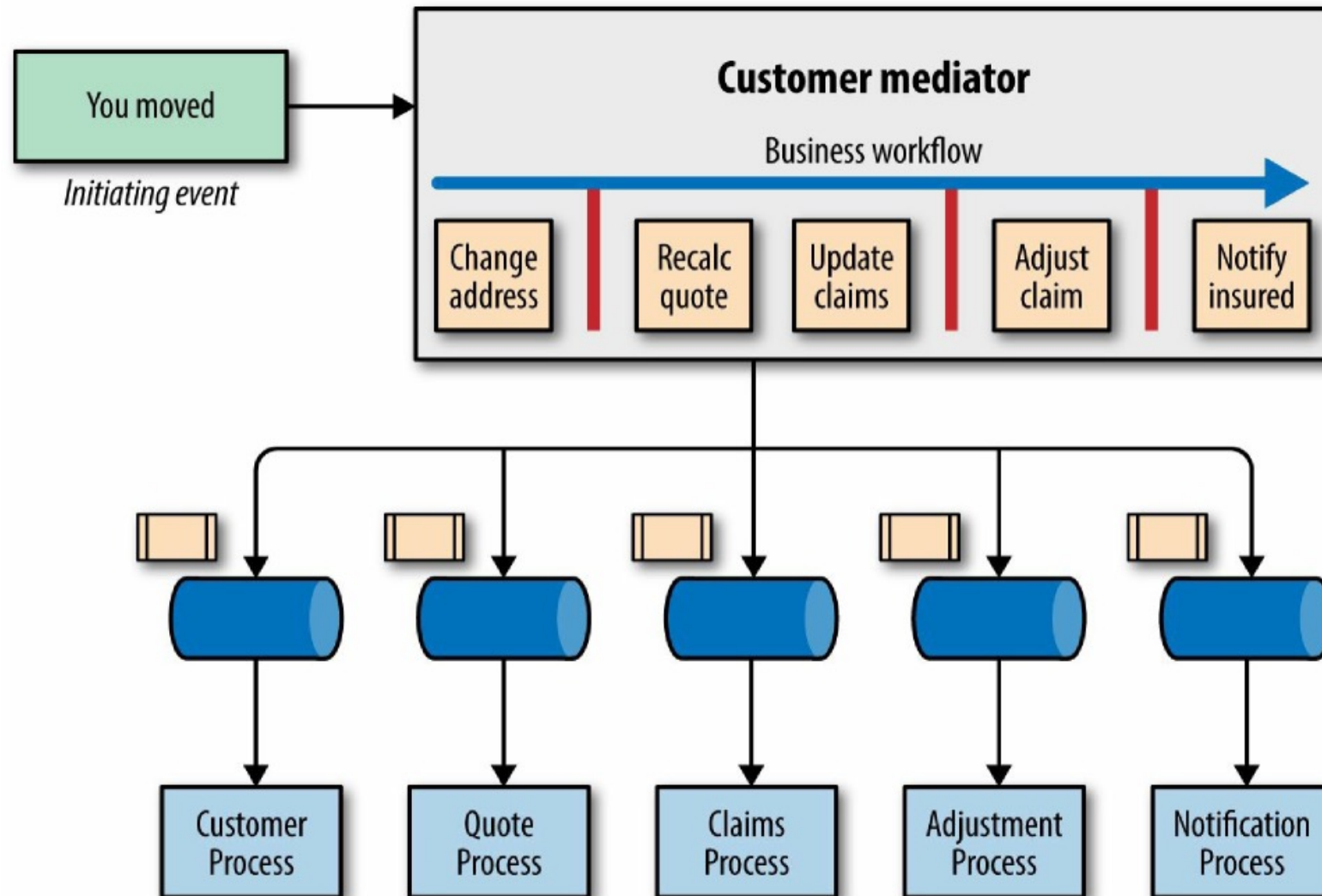
# Monoliths



# Event-Driven Architectures

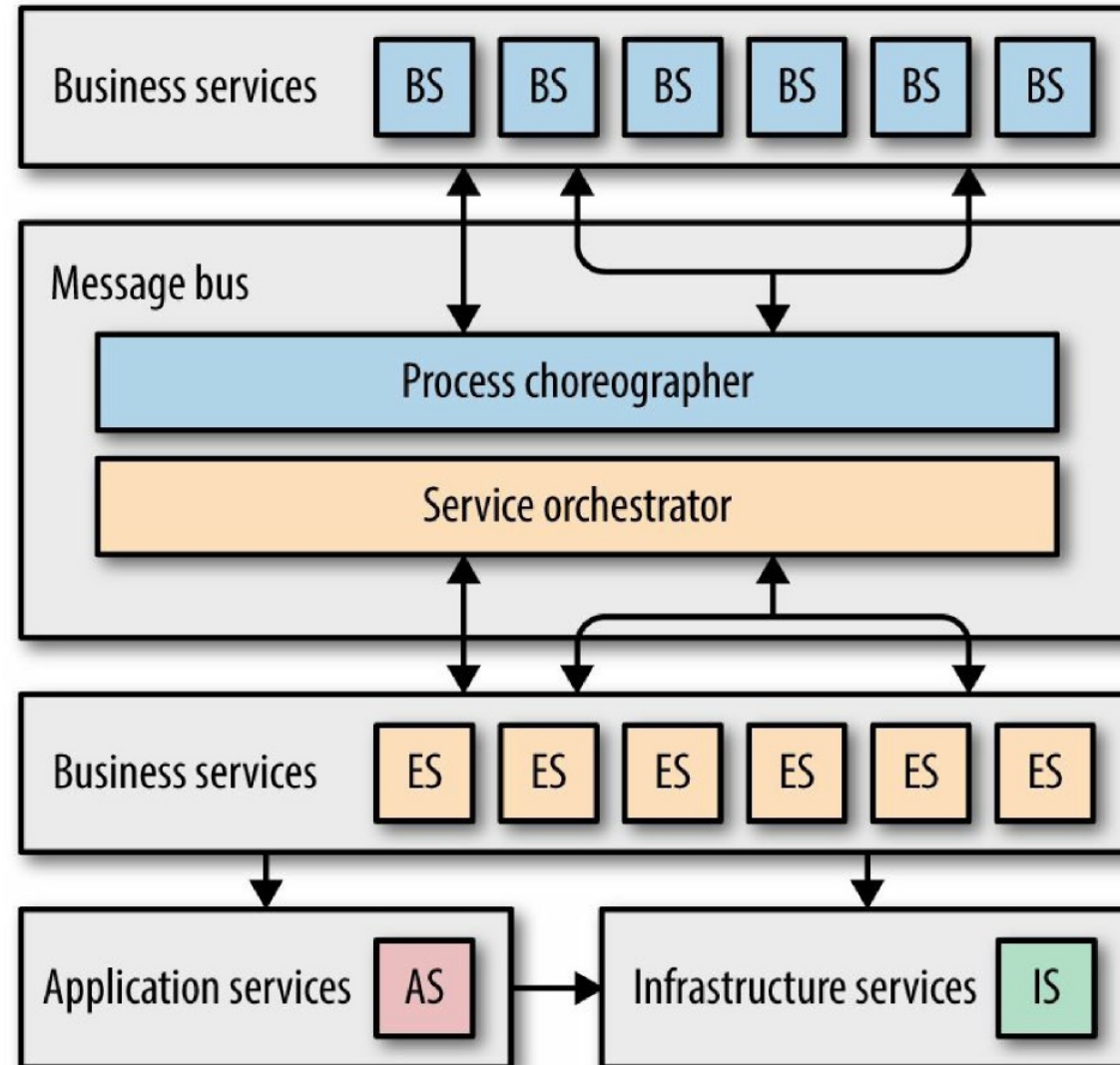


# Event-Driven Architectures

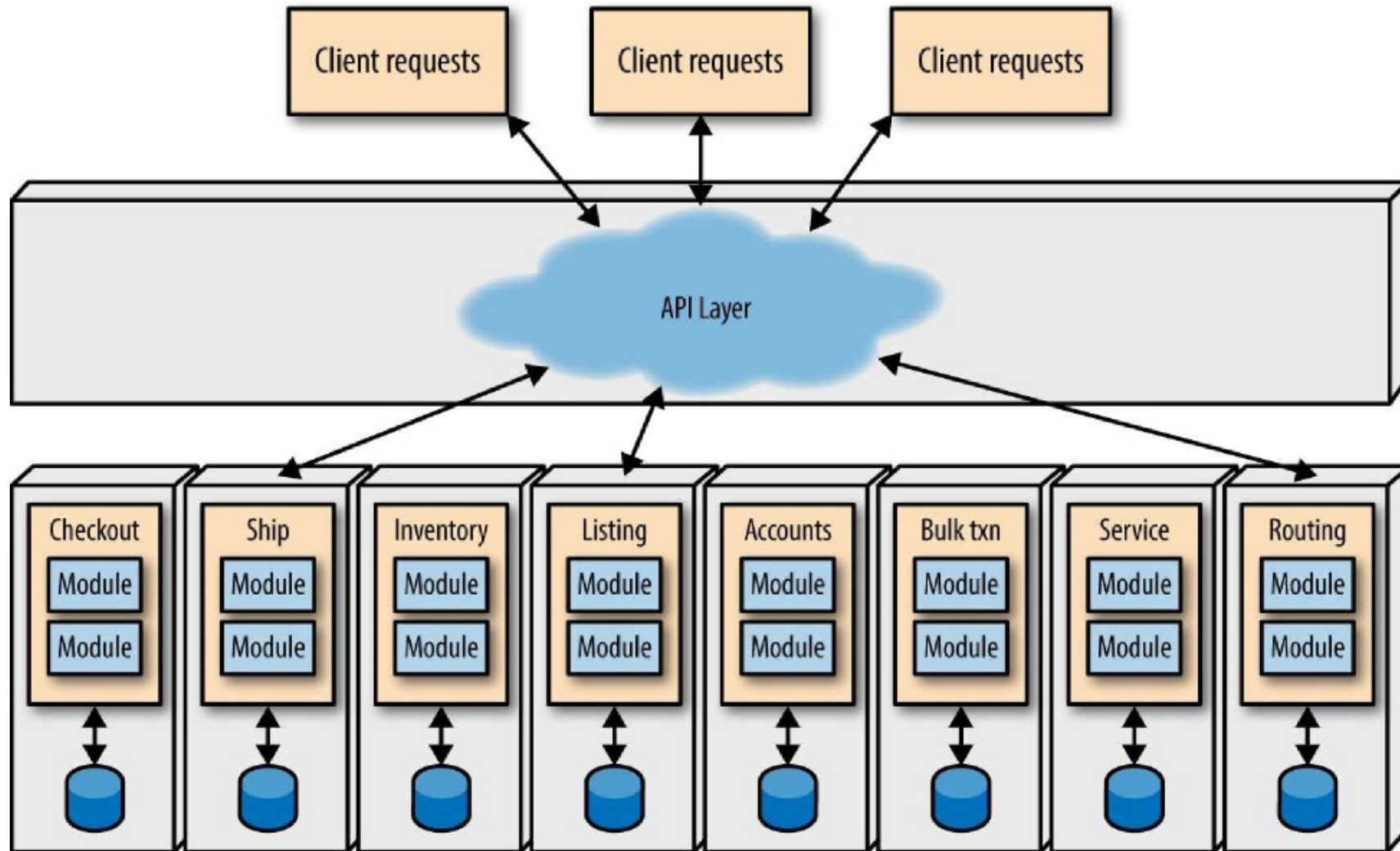




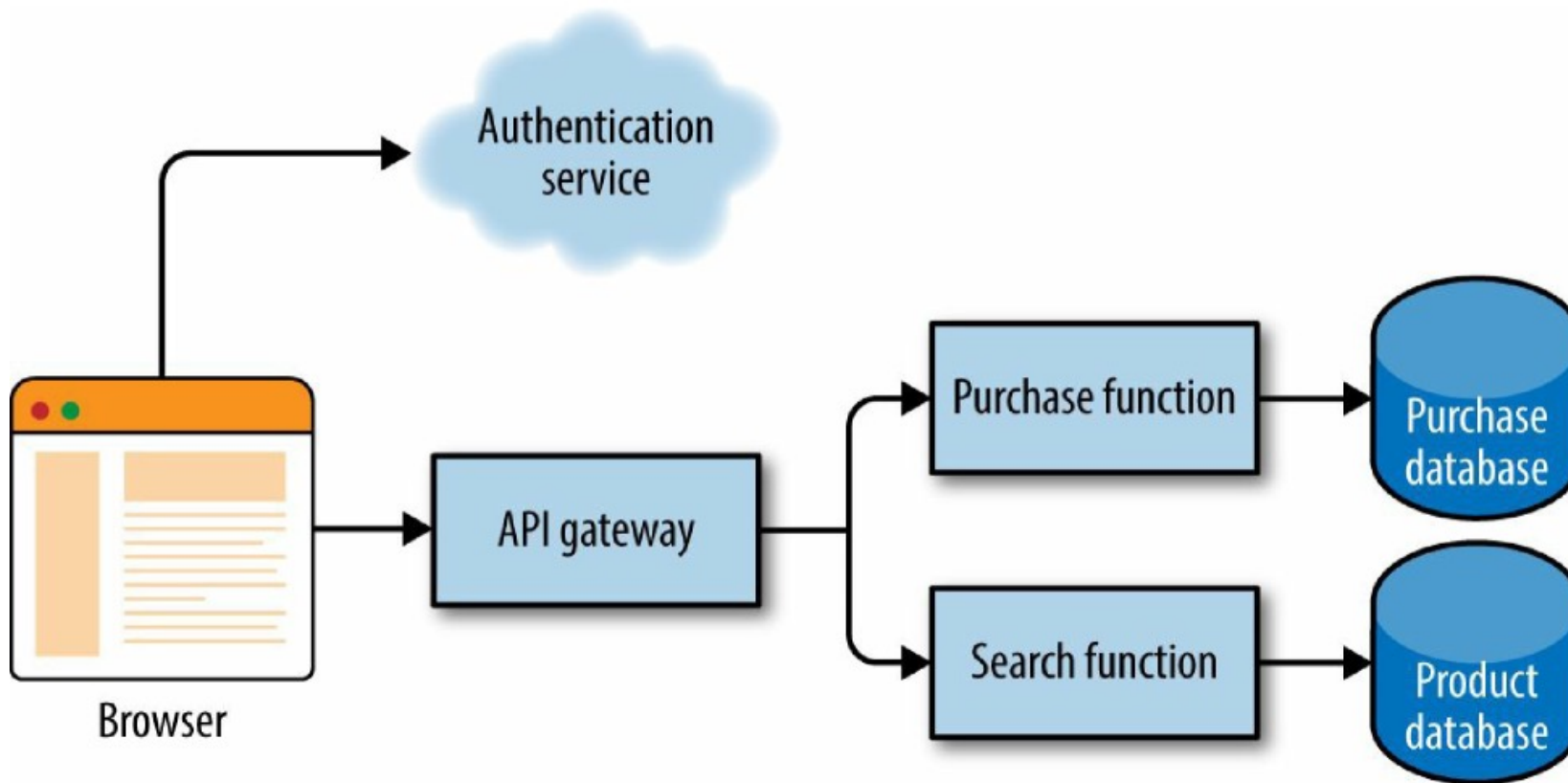
# Service-Oriented Architectures



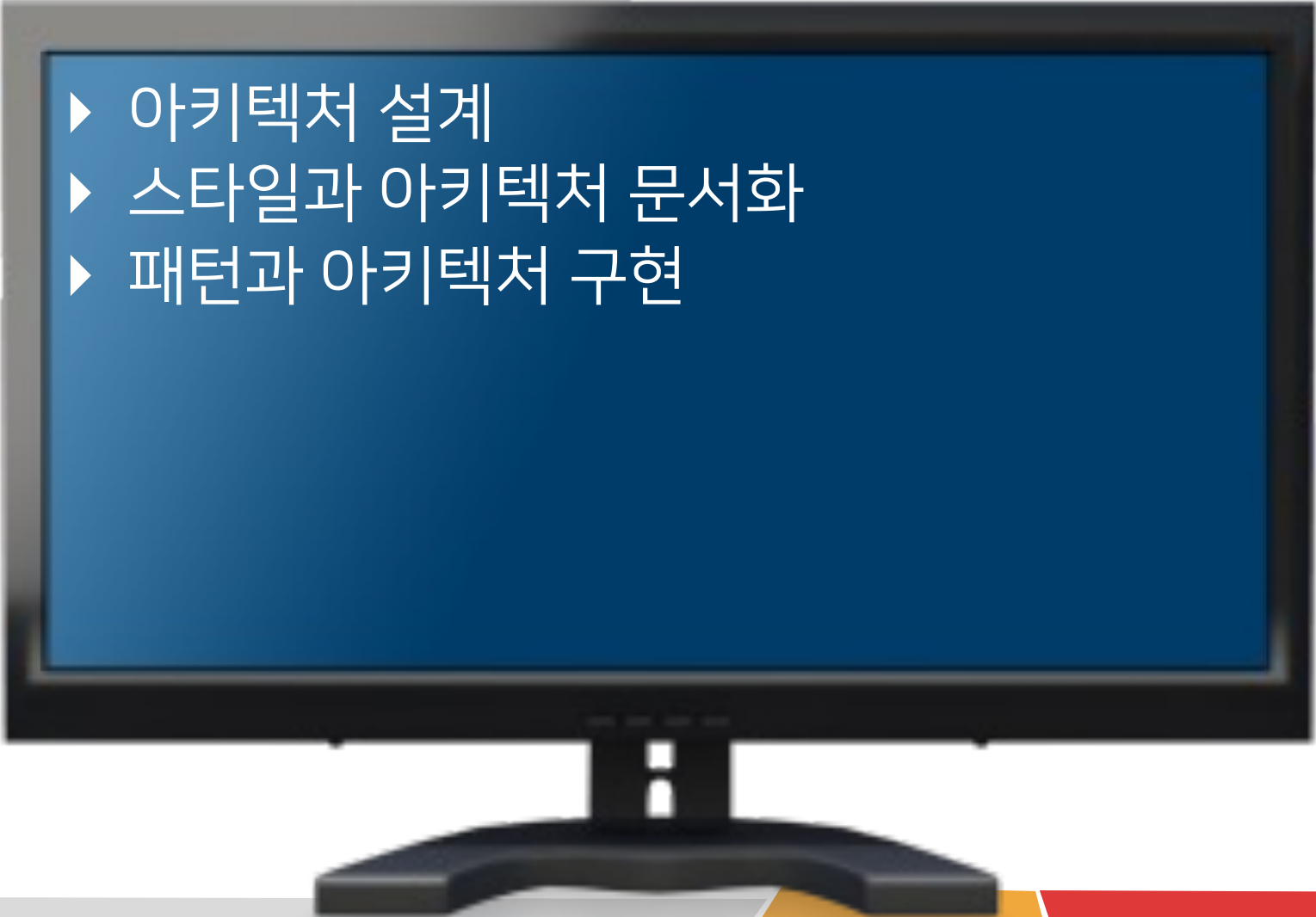
# Service-Oriented Architectures



# Serverless Architectures



# ○○ 소프트웨어 아키텍처 설계에서의 활용 ○○

- 
- ▶ 아키텍처 설계
  - ▶ 스타일과 아키텍처 문서화
  - ▶ 패턴과 아키텍처 구현



# Architecture Design

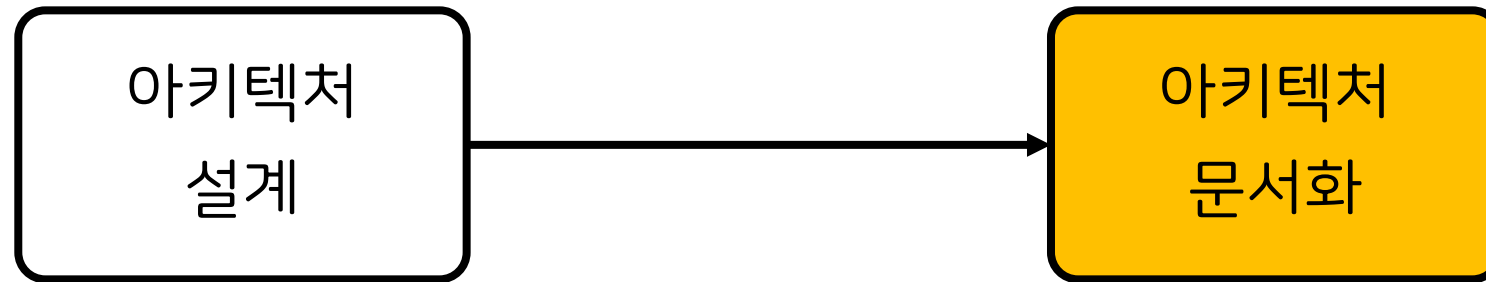


문제 분할

# Architecture Documentation

## ▶ 아키텍처 스타일

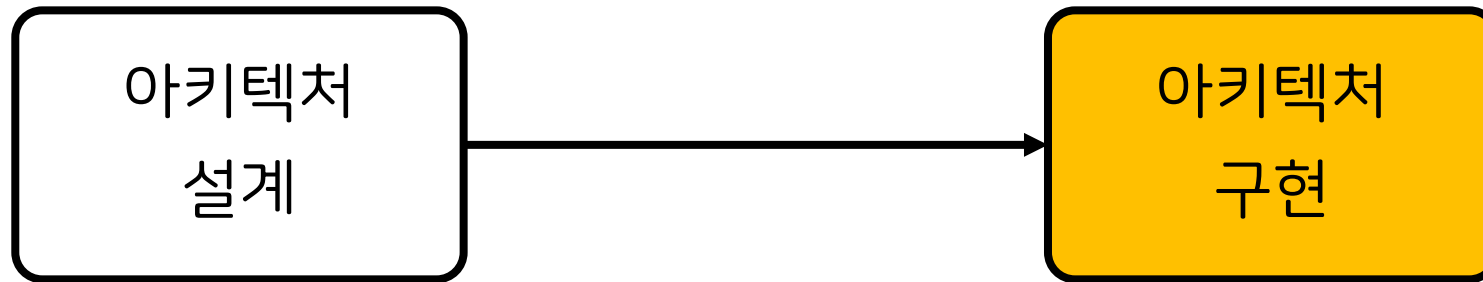
- ▶ 아키텍처 결정을 포함하는 아키텍처 뷰를 어떻게 작성하는지에 초점을 맞추어 목록화 함



# Architectural Code

## ▶ 아키텍처 패턴

- ▶ 특정한 영역의 설계 솔루션을 패턴으로 도출하여 다른 설계에서 참조할 수 있도록 제시되는 형태로 구현을 염두에 둠







# Question?



**Seonah Lee**  
**[saleese@gmail.com](mailto:saleese@gmail.com)**