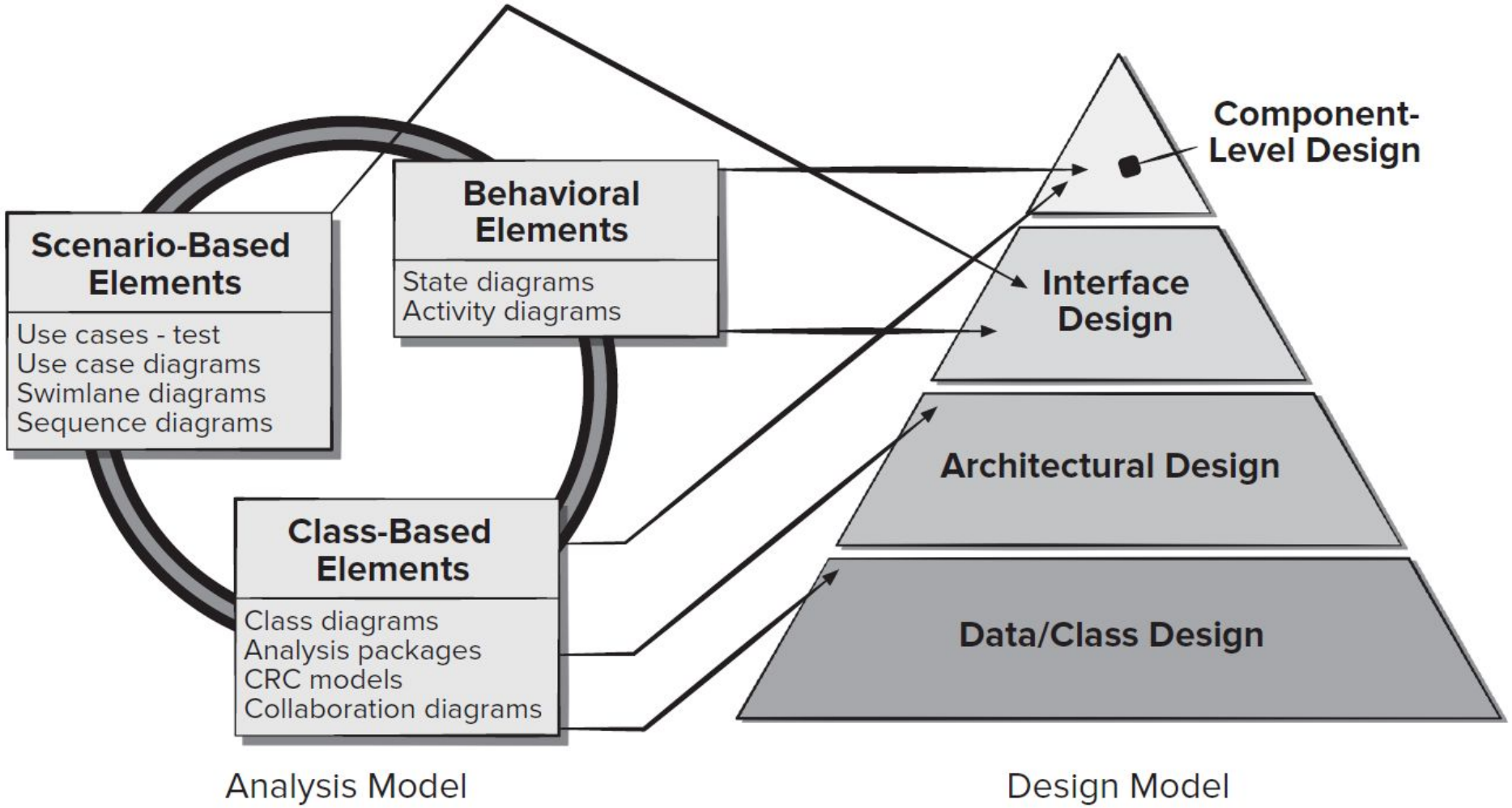
Software Engineering

Part (VI)- System Design (I)

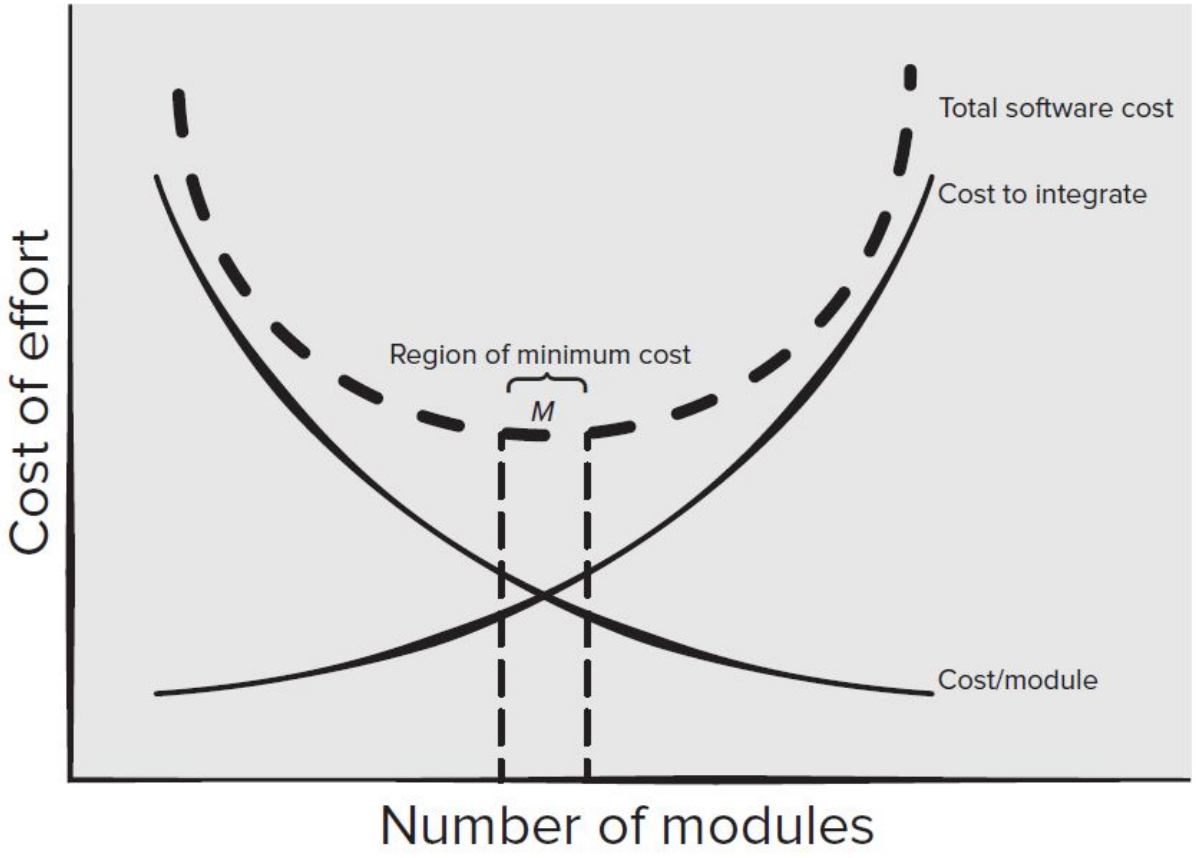
By: Mehran Alidoost Nia
Shahid Beheshti University, Fall 2023

Requirements vs. Design Model



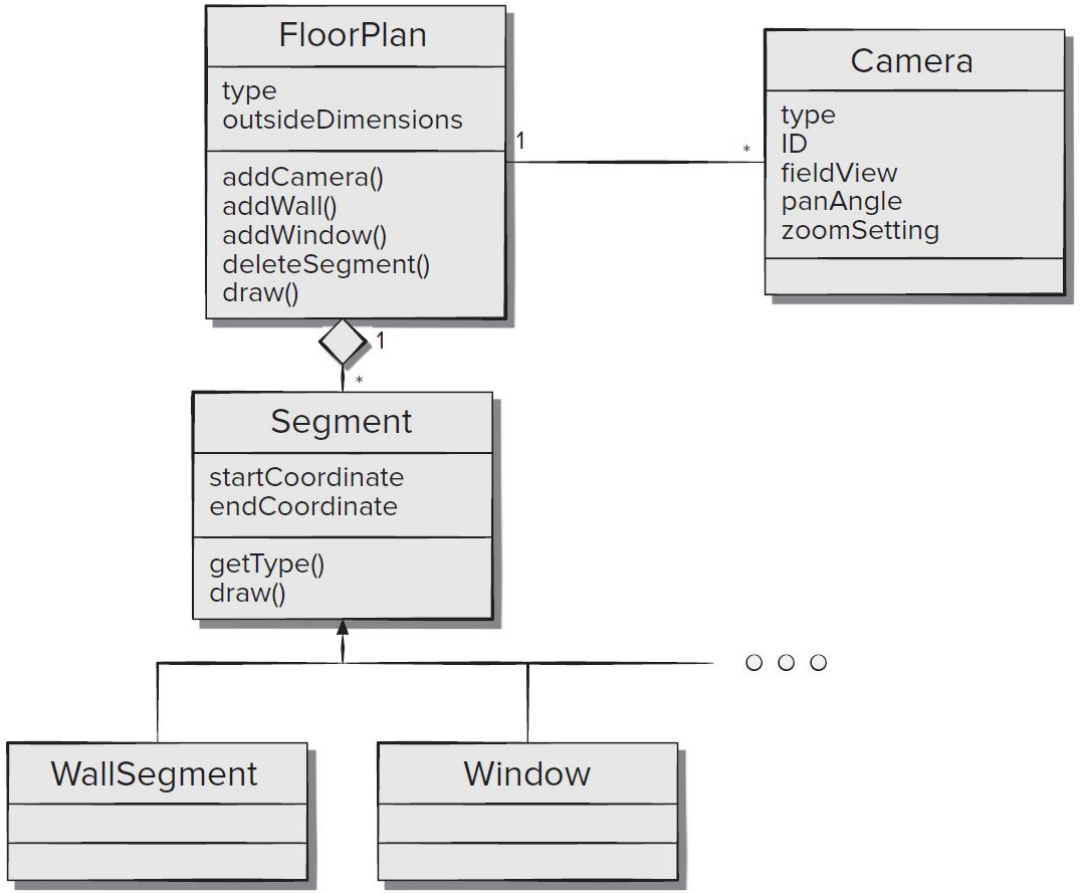
Design Concepts

- Abstraction
- Architecture
- Patterns
- Separation of Concerns
- Modularity
- Information Hiding
- Functional Independence
- Refactoring
- Design Classes



Design Classes

- Complete and sufficient
- Primitiveness
- High cohesion
- Low coupling



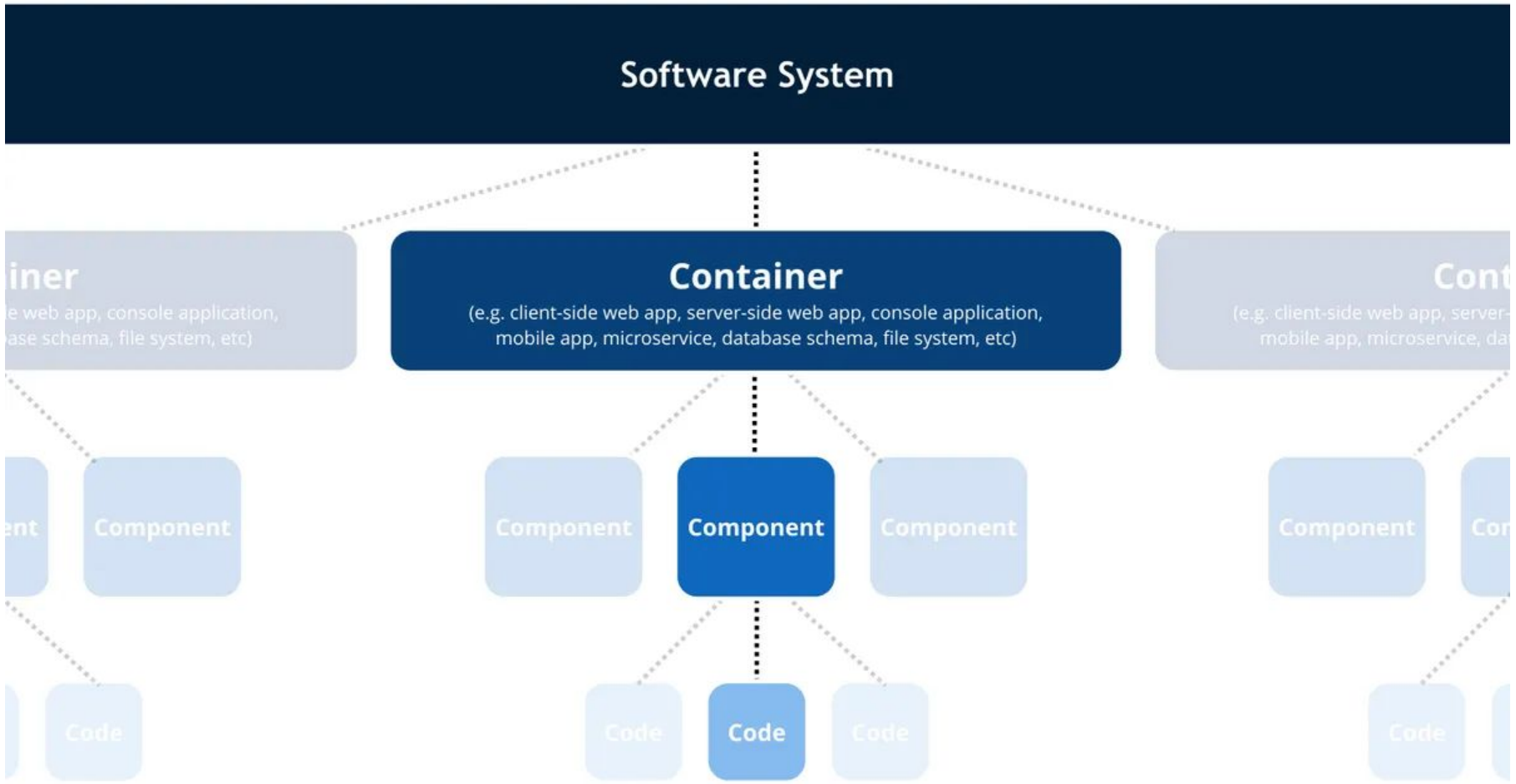
Design Modeling Principles

- Design should be traceable to the requirements model.
- Always consider the architecture of the system to be built.
- Design of data is as important as design of processing functions.
- Interfaces (both internal and external) must be designed with care.
- User interface design should be tuned to the needs of the end user.

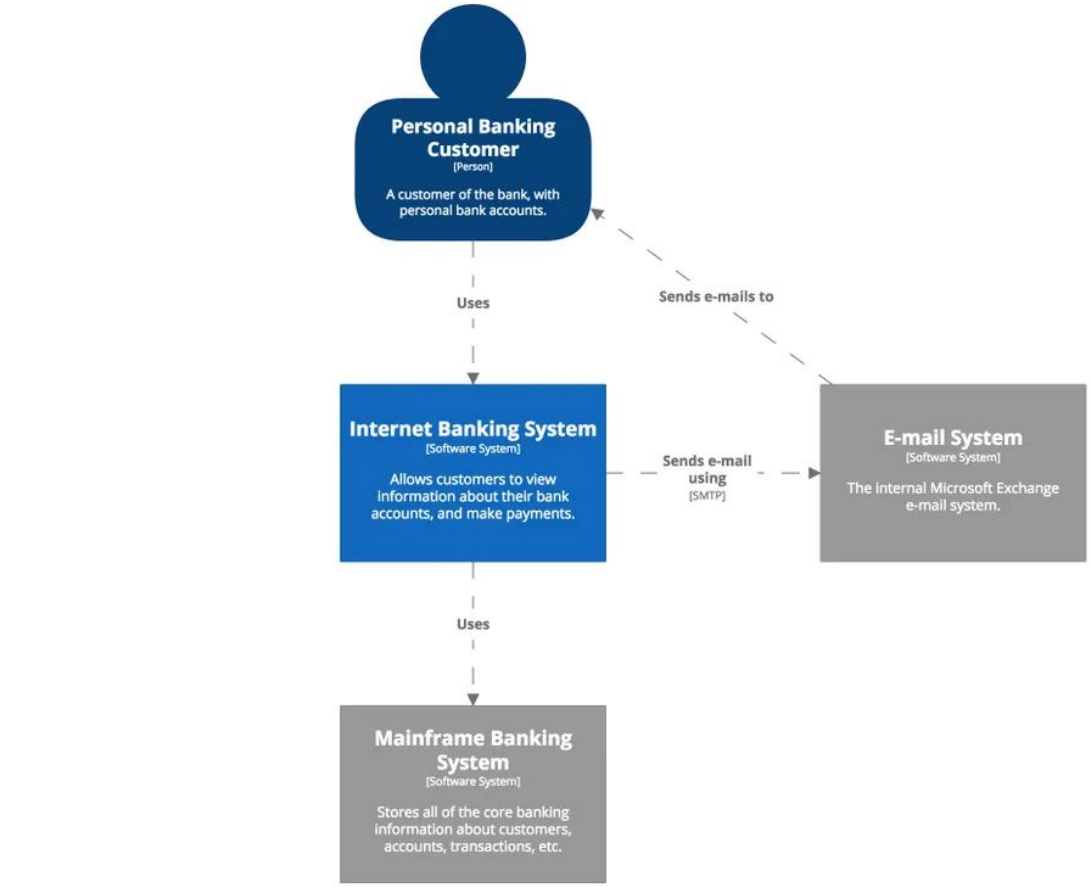
Design Modeling Principles

- Component-level design should be functionally independent.
- Components should be loosely coupled to one another and to the external environment.
- Design models should be easily understandable.
- The design should be developed iteratively.
- Design model does not preclude an agile approach.

The C4 Model for Software Architecture



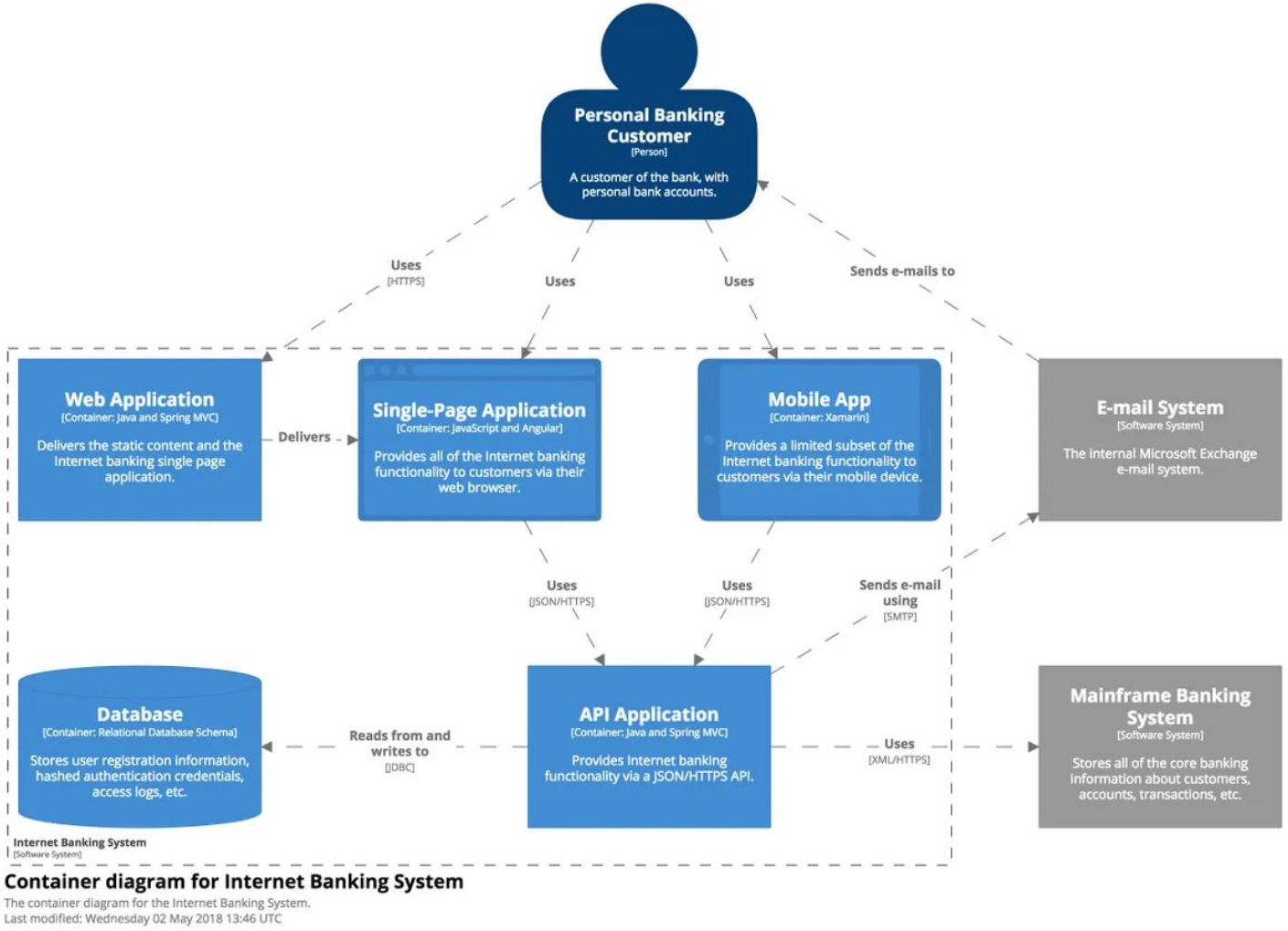
System Context Diagram



System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.
Last modified: Wednesday 02 May 2018 13:46 UTC

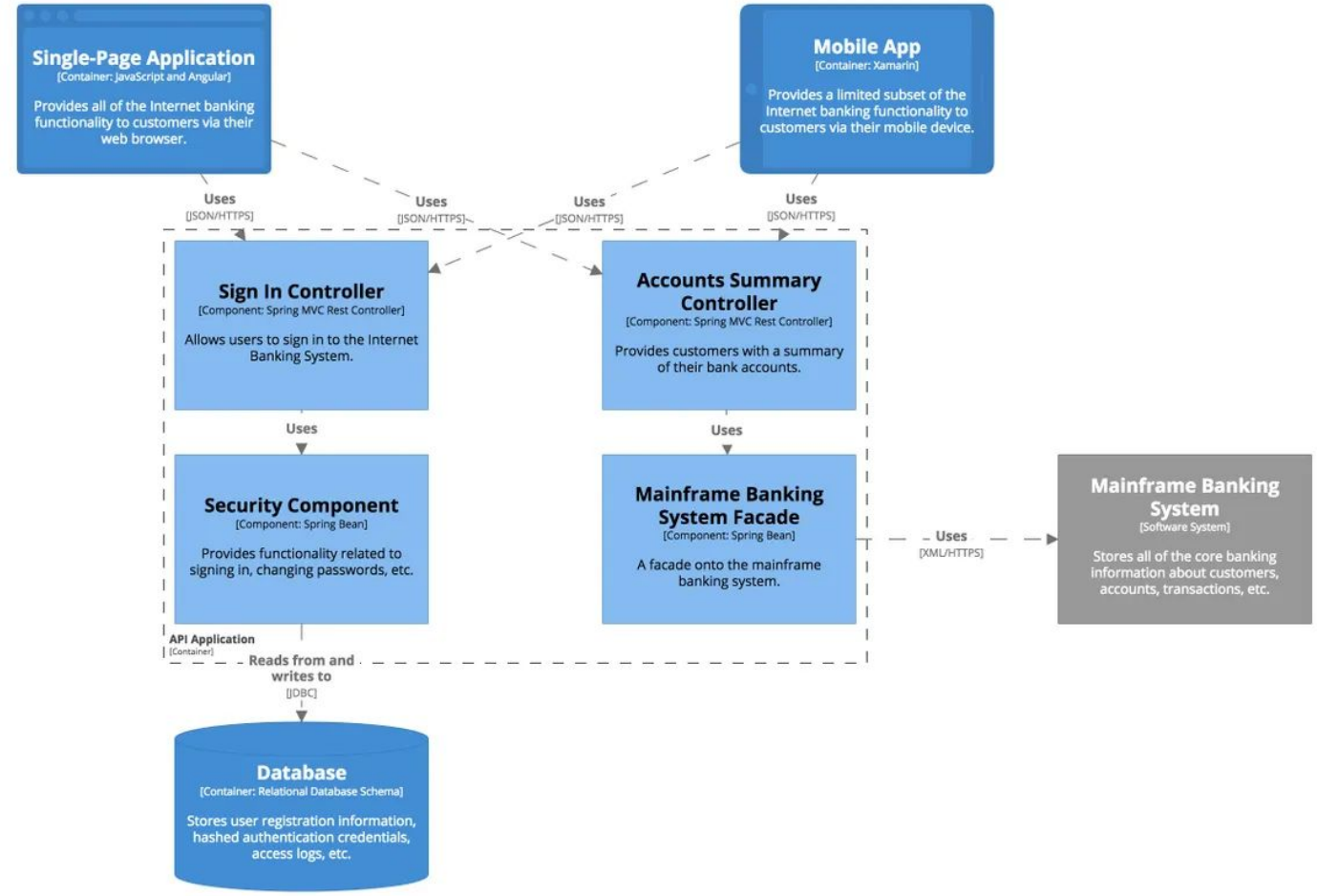
Container Diagram



Container diagram for Internet Banking System

The container diagram for the Internet Banking System.
Last modified: Wednesday 02 May 2018 13:46 UTC

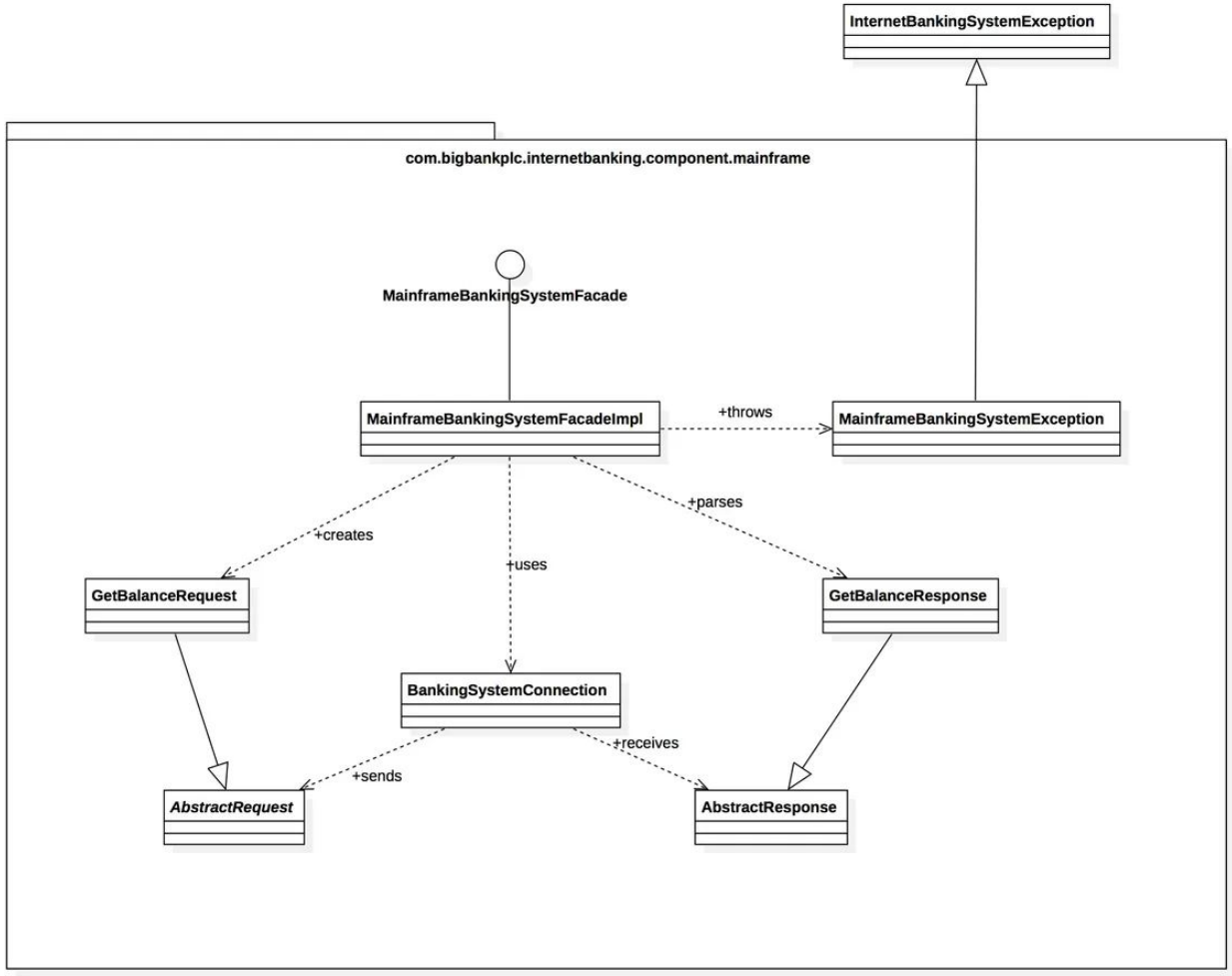
Component Diagram



Component diagram for Internet Banking System - API Application

The component diagram for the API Application.
Last modified: Wednesday 02 May 2018 13:46 UTC

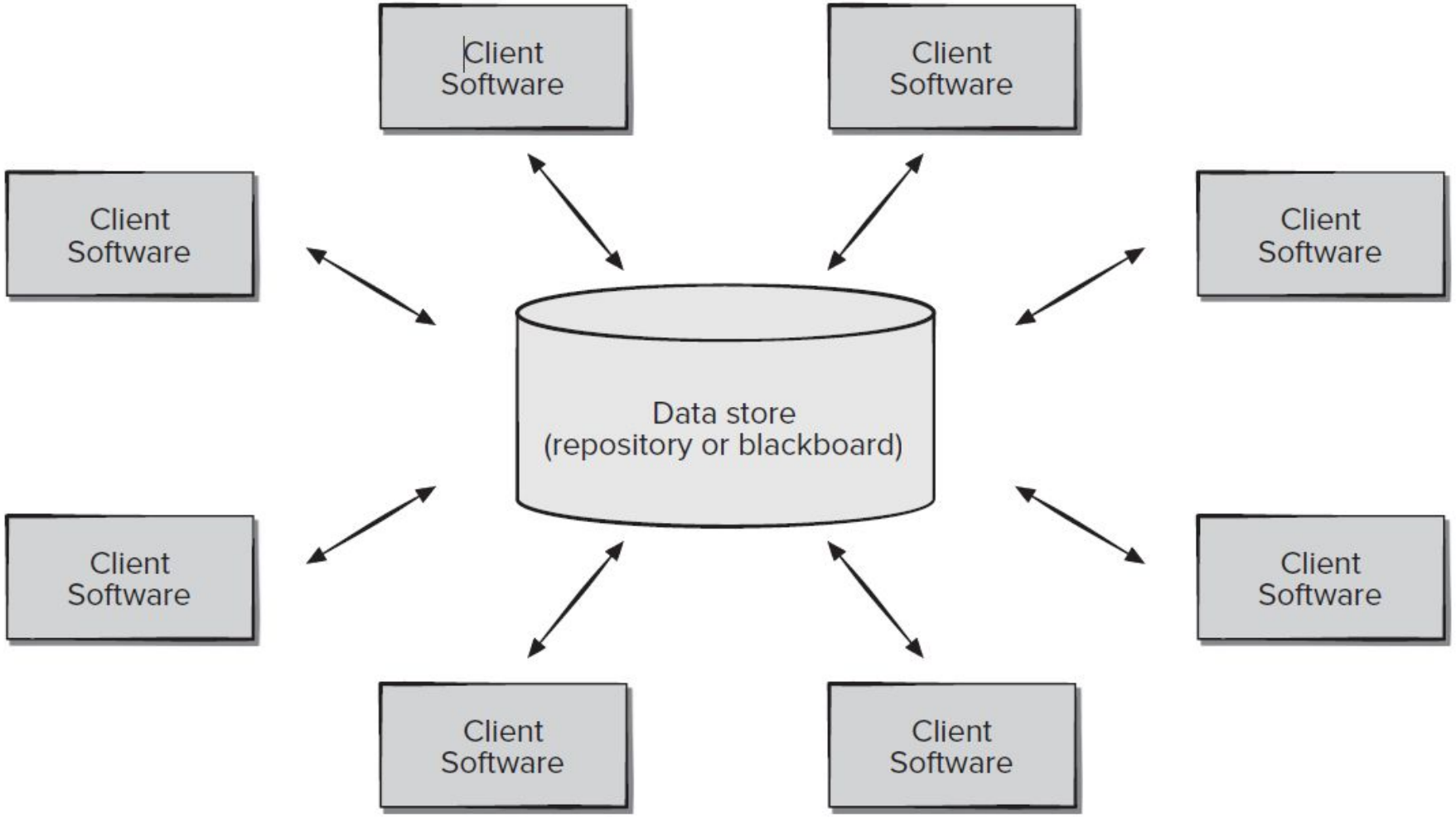
Code (Class)


InternetBankingSystemException
com.bigbankplc.internetbanking.component.mainframe
MainframeBankingSystemFacade
MainframeBankingSystemFacadeImpl
+throws
MainframeBankingSystemException
+creates
+uses
+parses
GetBalanceRequest
GetBalanceResponse
BankingSystemConnection
+receives
+sends
AbstractRequest
AbstractResponse

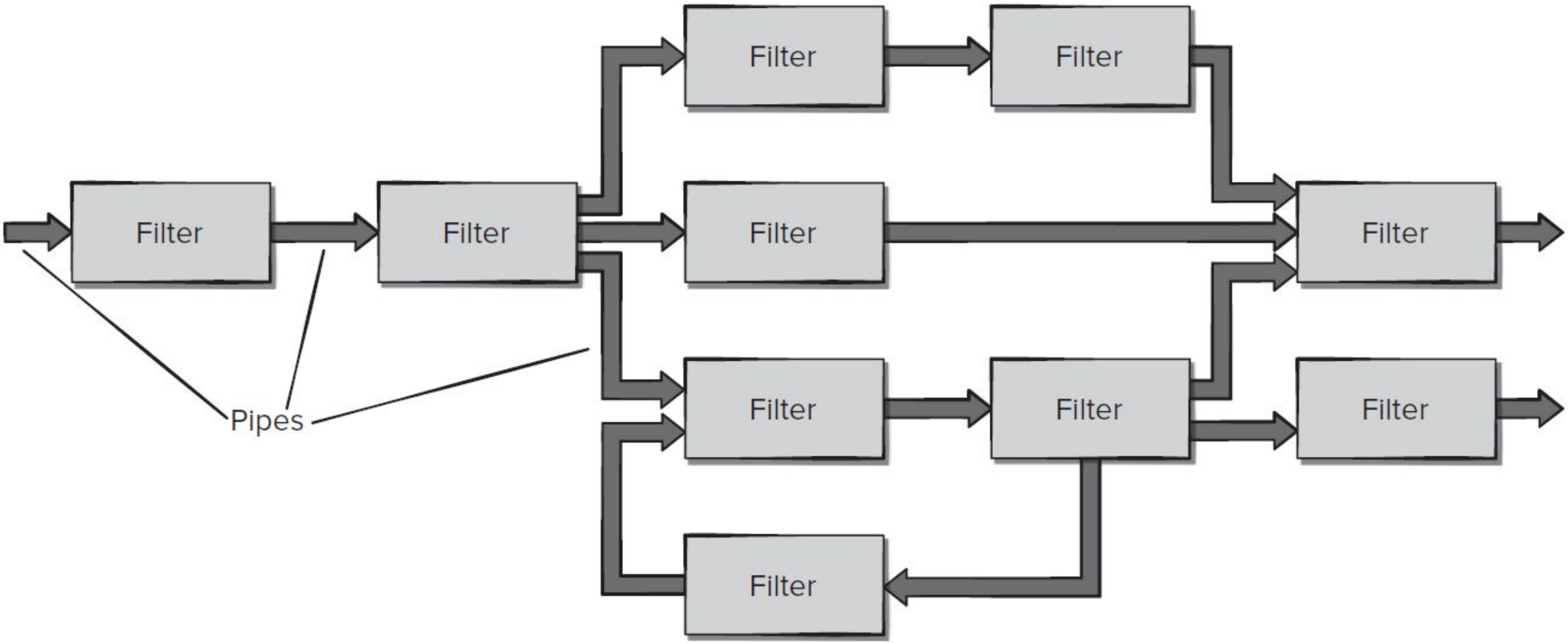
Architectural Styles

- Data-Centered Architectures
- Data-Flow Architectures
- Call-and-Return Architectures
  - Main program/subprogram architectures
  - Remote procedure call architectures
- Object-Oriented Architectures
- Layered Architectures (e.g., MVC)
- Multitier architecture

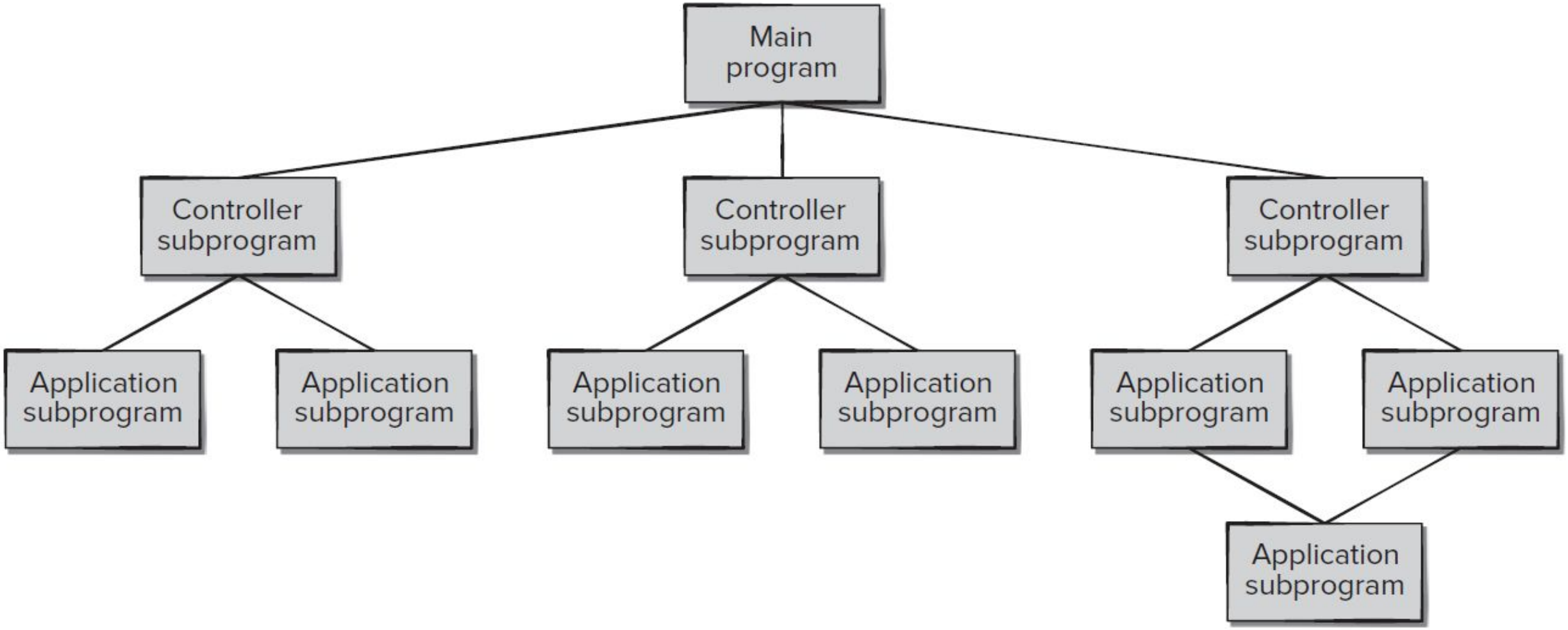
Data-Centered Architectures



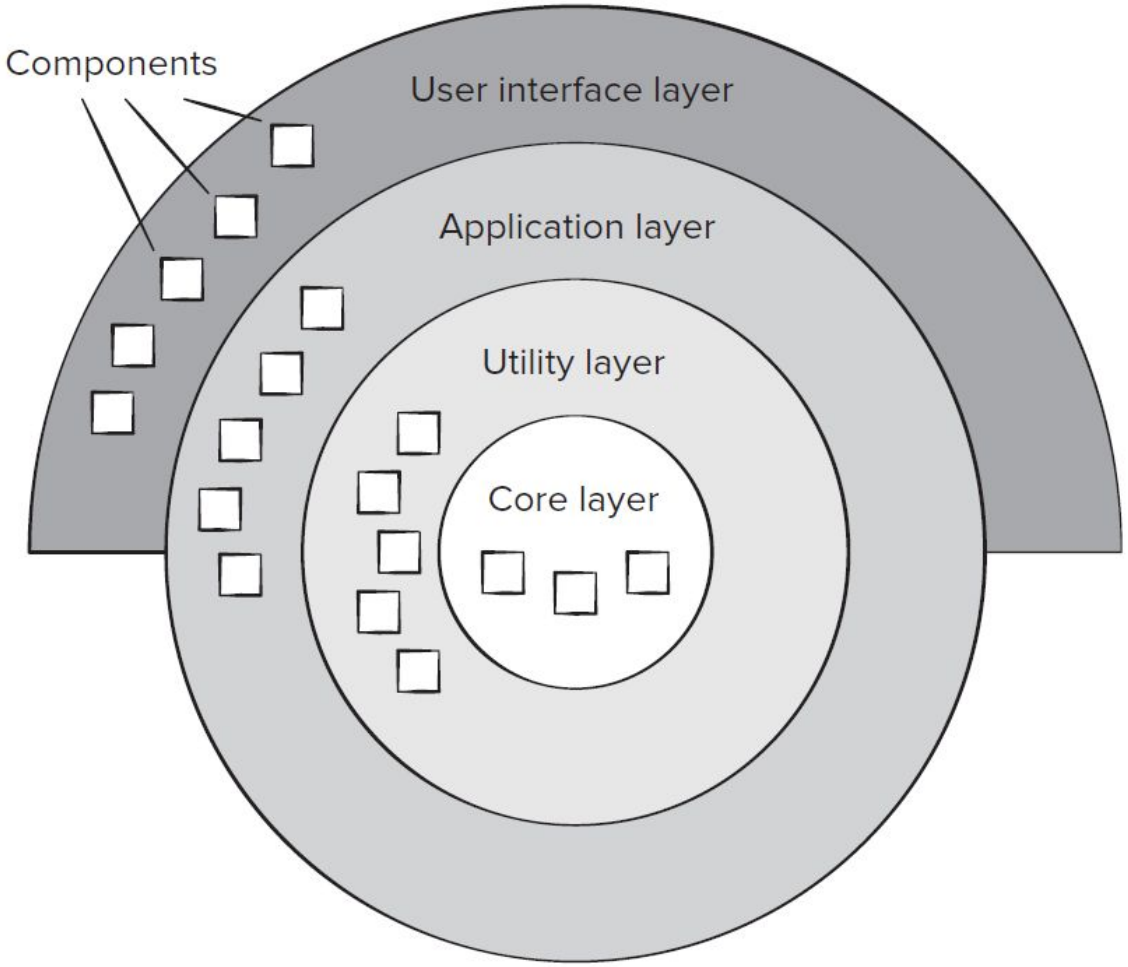
Data-Flow Architectures



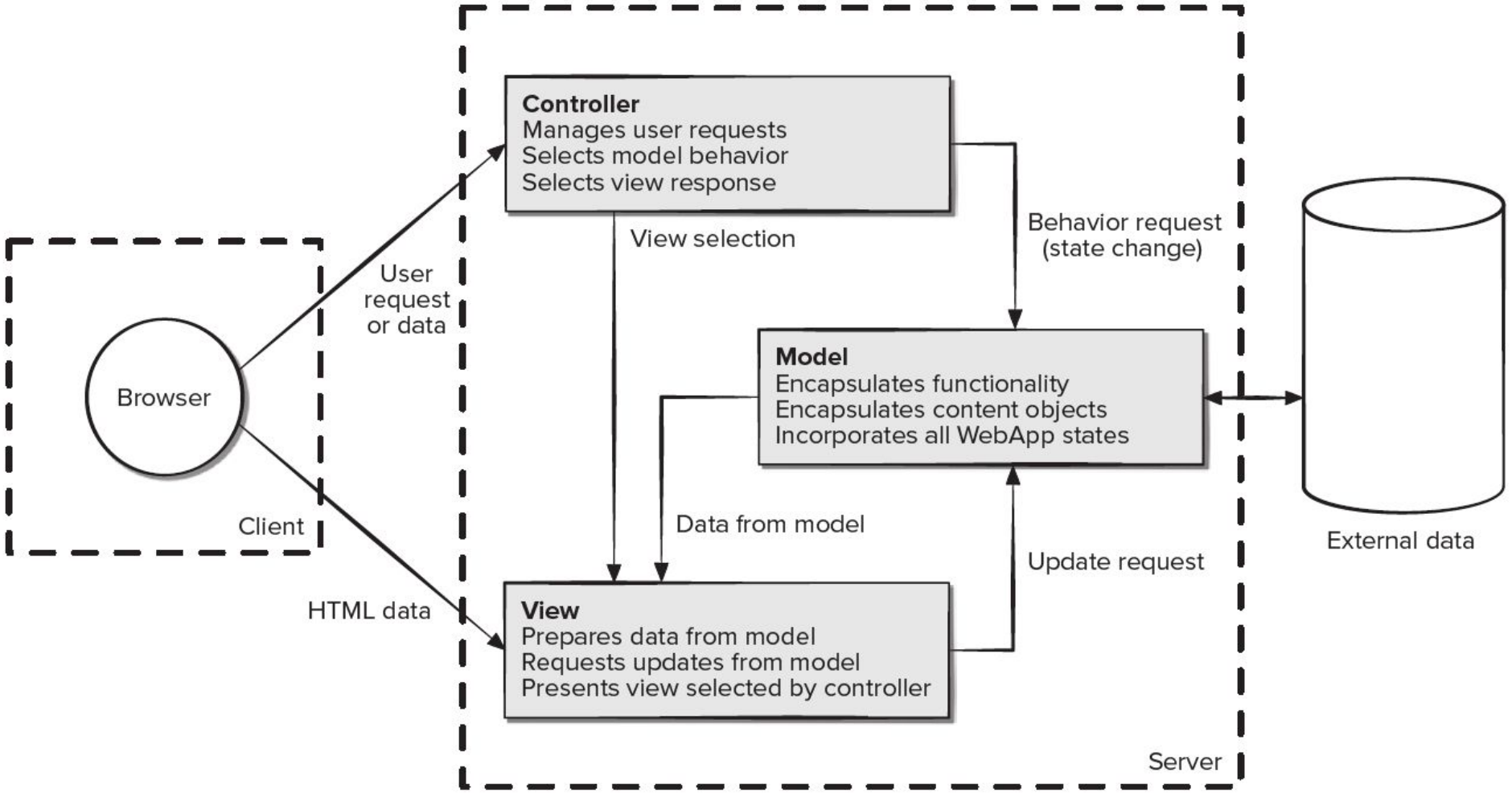
Call-and-Return Architectures



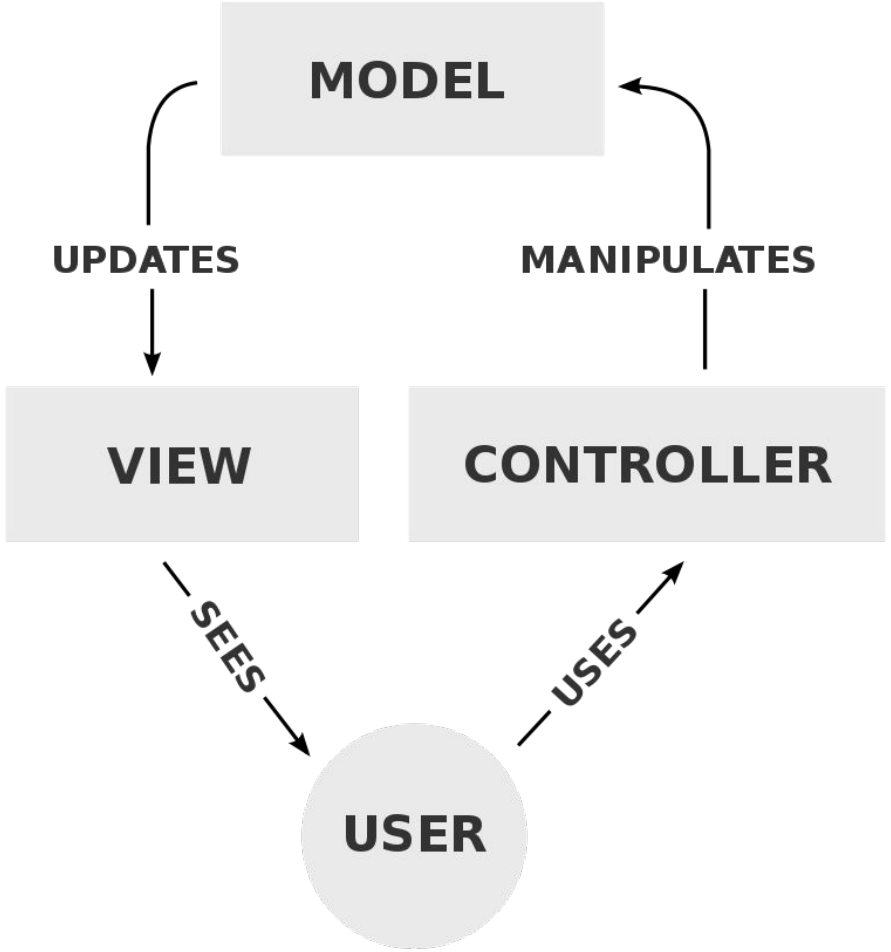
Layered Architectures



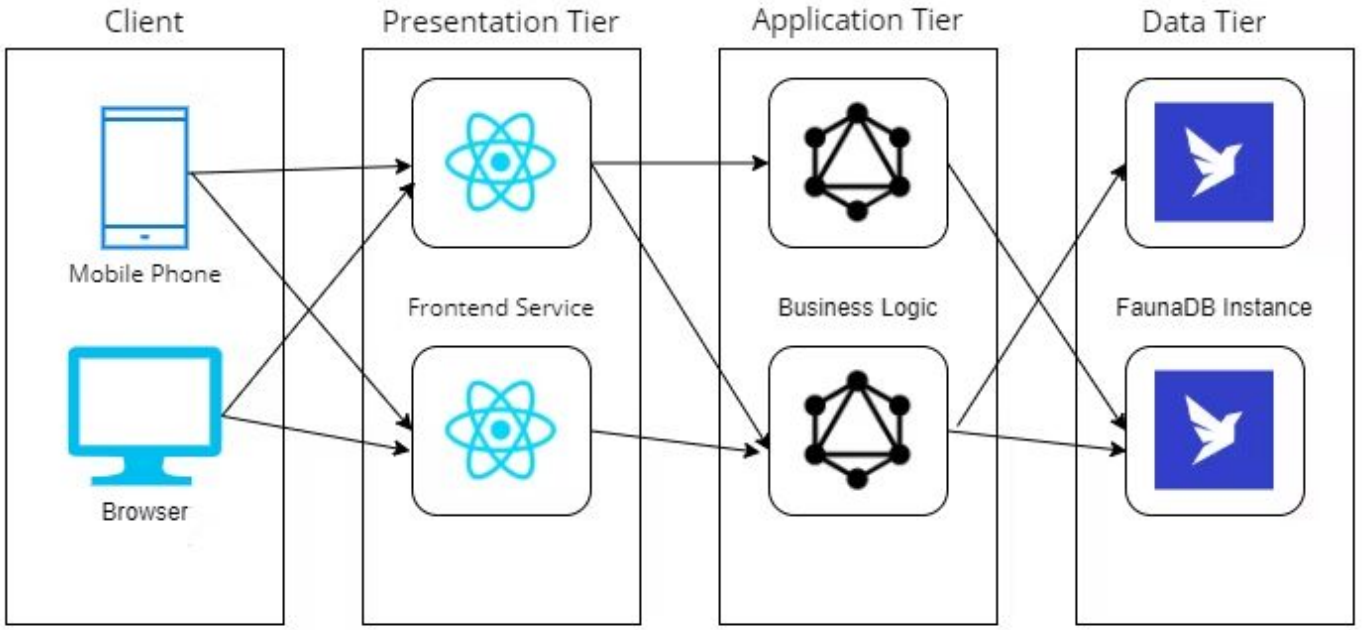
The MVC Architectures



The MVC Architectures



Multi Tier Architecture



Modern Architectural Styles

- Monolithic Applications
- Service-Oriented Architecture (SOA)
  - Microservice Architecture
- Asynchronous Messaging
- Domain-Driven Design (DDD)

Monolithic Architecture

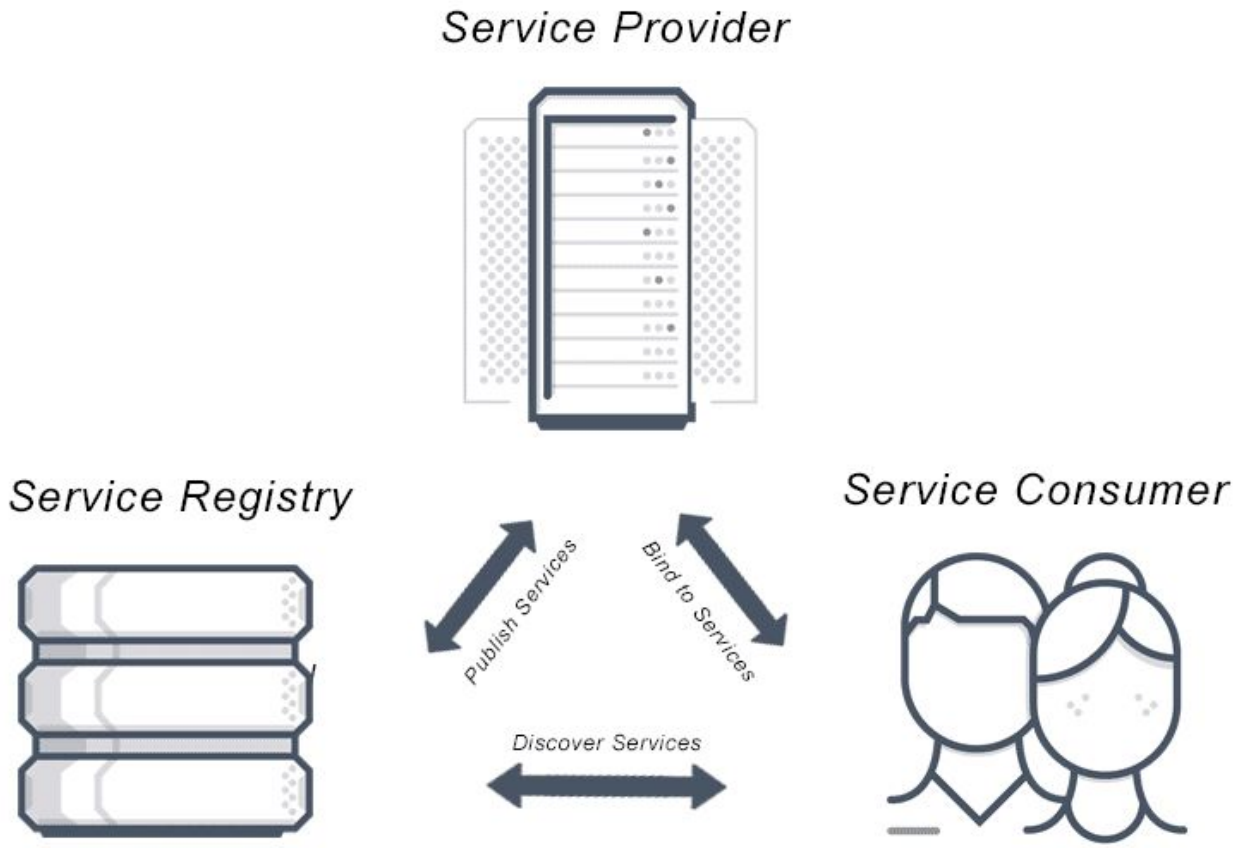
- A monolithic architecture is a traditional model of a software program, which is built as a unified unit that is self-contained and independent from other applications.
- It is generally considered a good choice for smaller programs that need quick and cheap deployment.
- They lack flexibility and can be difficult to scale.

SOA Architectural Style

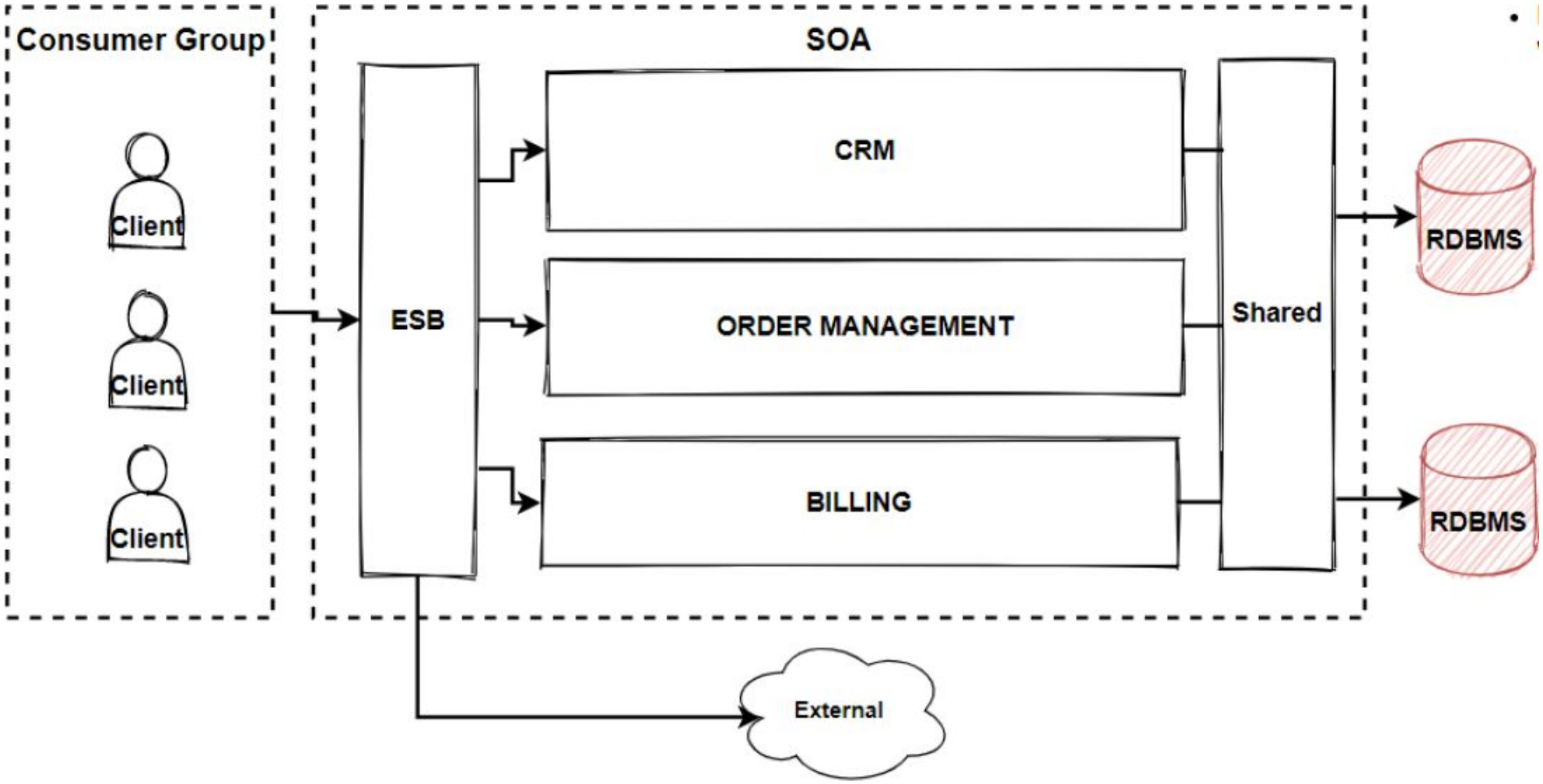
- SOA is a method of software development that uses software components called services to create business applications.
- Each service provides a business capability.
- Services can also communicate with each other across platforms and languages.

SOA Architectural Style

The Service Oriented Architecture Triangle



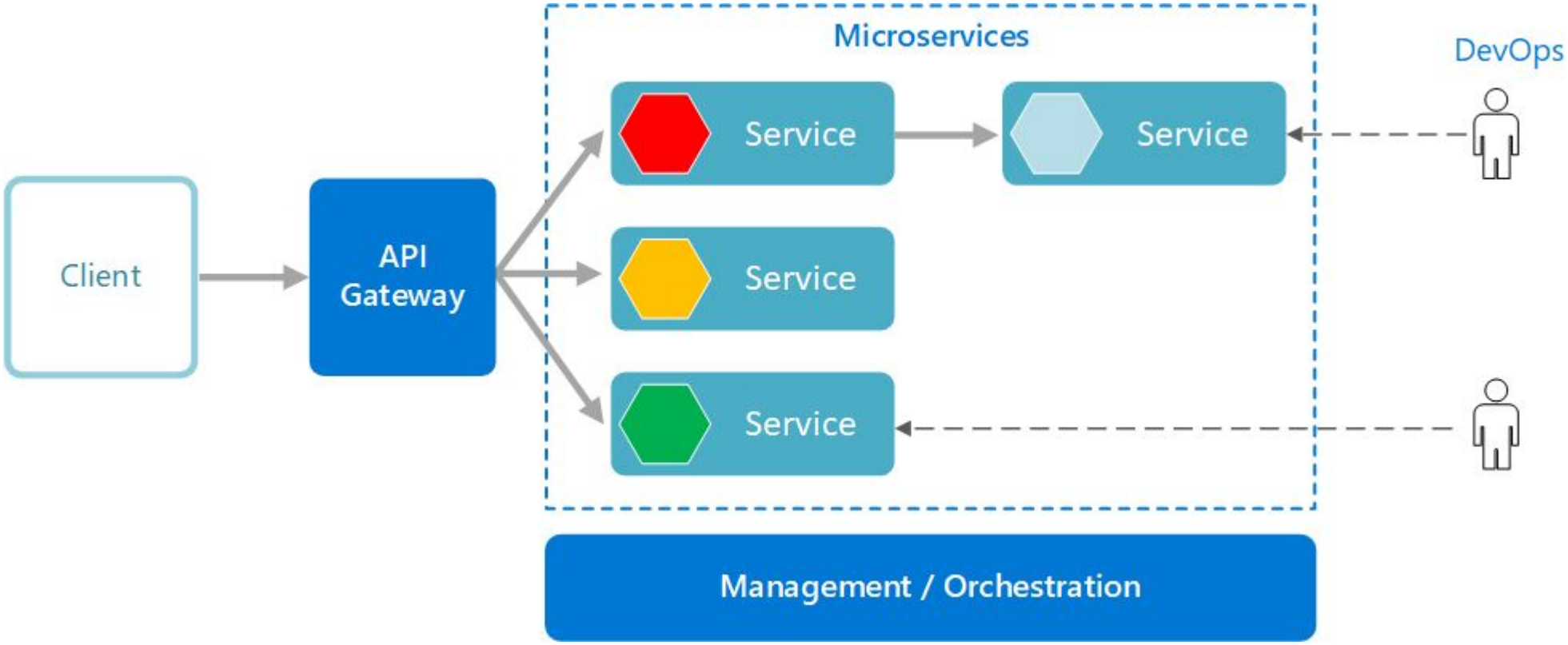
SOA Architectural Style



Microservices

- Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs.
- These services are owned by small, self-contained teams.

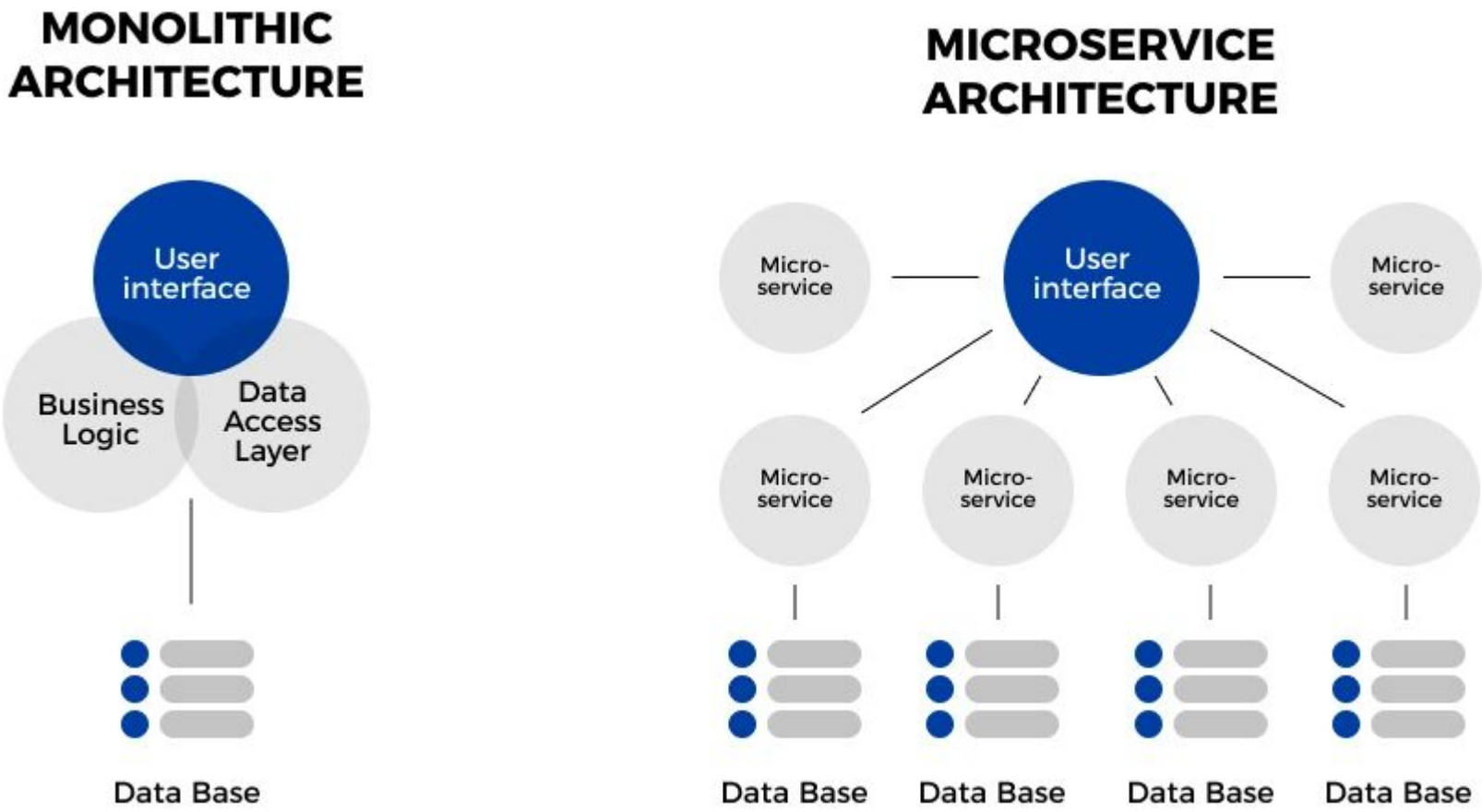
Microservice Architecture



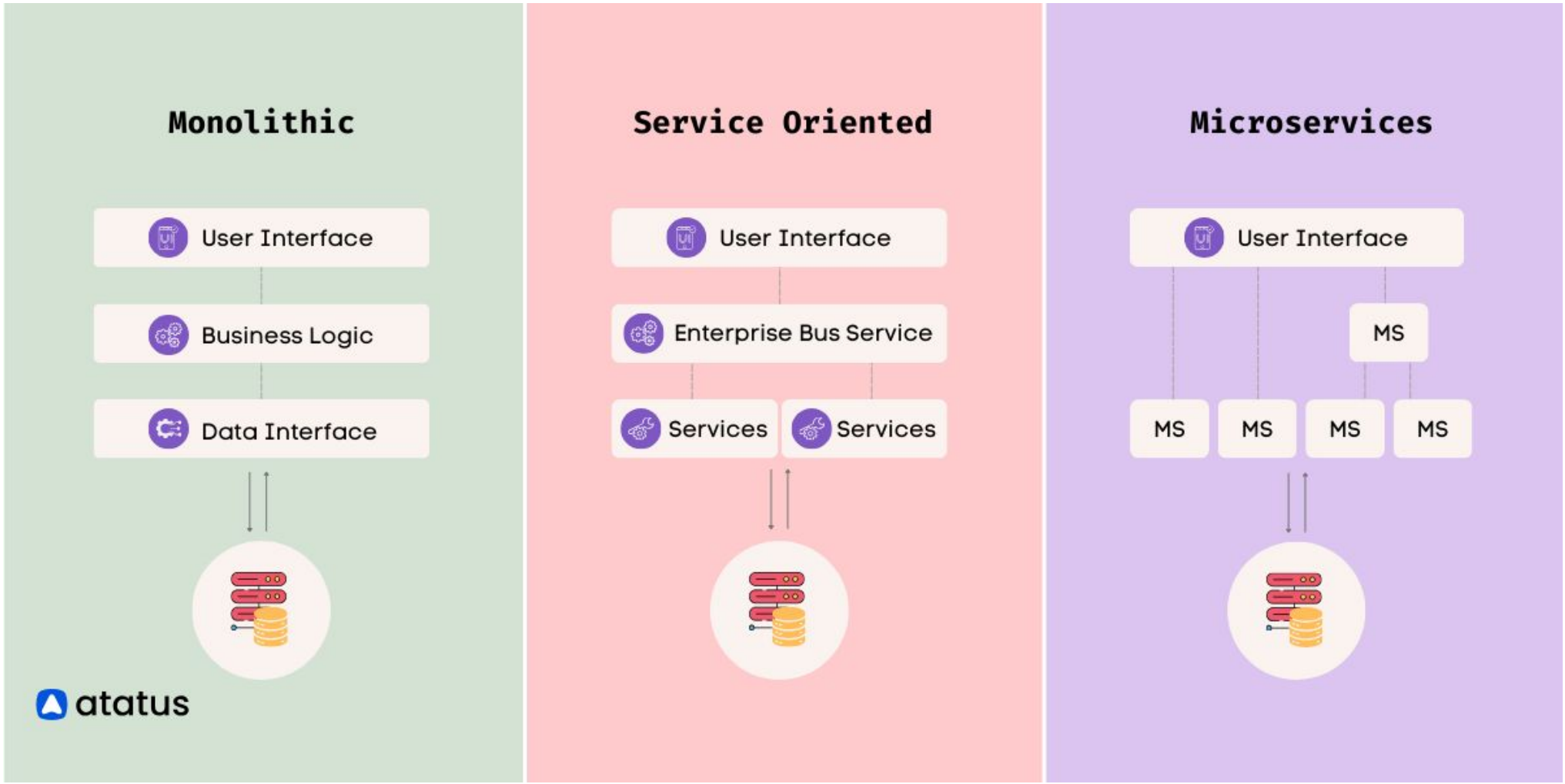
SOA vs. Microservices

	Microservices	SOA
Architecture	Designed to host services which can function independently	Designed to share resources across services
Component sharing	Typically does not involve component sharing	Frequently involves component sharing
Granularity	Fine-grained services	Larger, more modular services
Data storage	Each service can have an independent data storage	Involves sharing data storage between services
Governance	Requires collaboration between teams	Common governance protocols across teams
Size and scope	Better for smaller and web-based applications	Better for large scale integrations
Communication	Communicates through an API layer	Communicates through an ESB
Coupling and cohesion	Relies on bounded context for coupling	Relies on sharing resources
Remote services	Uses REST and JMS	Uses protocols like SOAP and AMQP
Deployment	Quick and easy deployment	Less flexibility in deployment

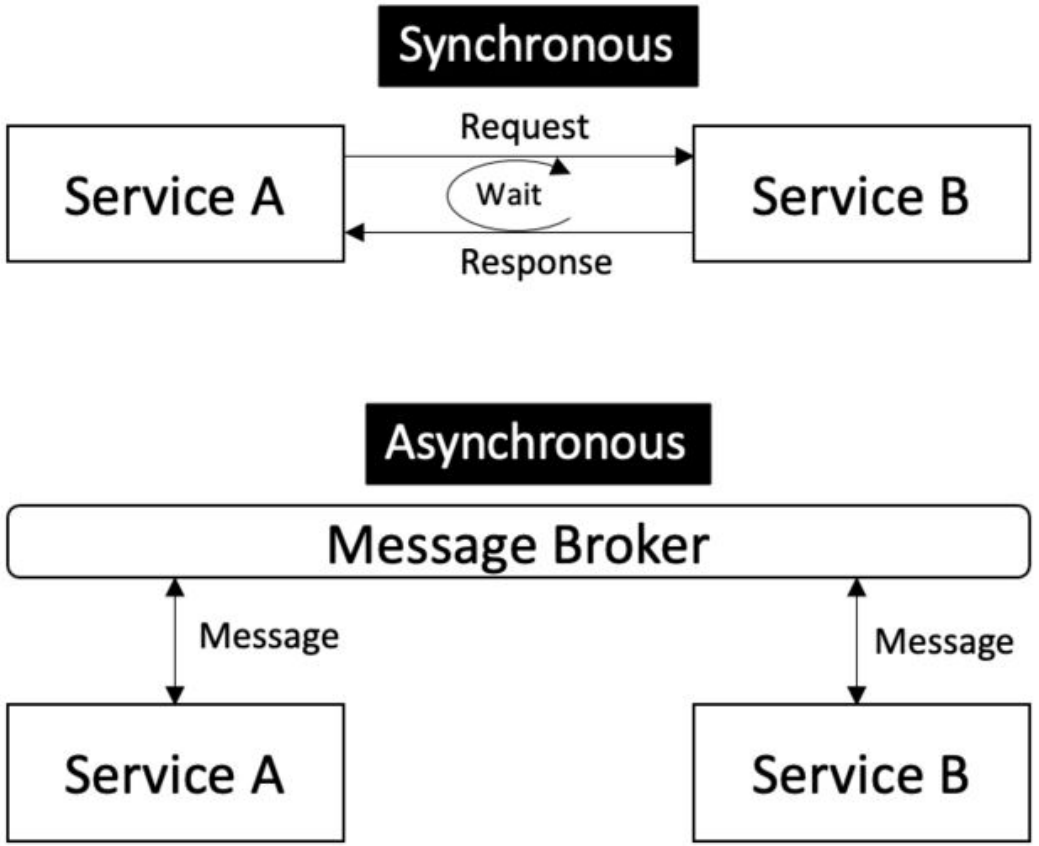
Monolithic vs. Microservices



Monolithic vs. SOA vs. Microservices



Synchronous vs. Asynchronous Com.



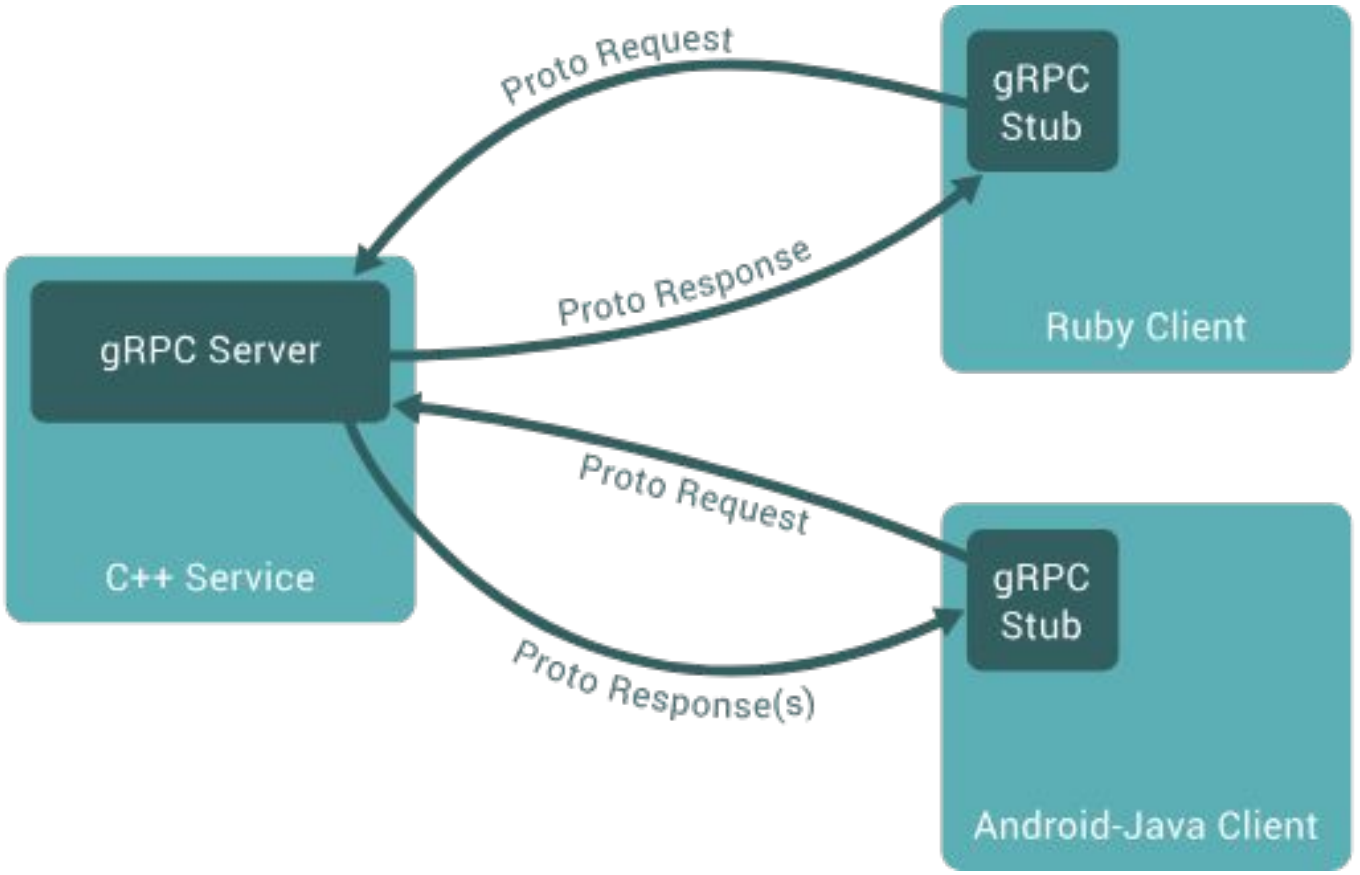
Synchronous Communications: REST

- HTTP verbs (Get, Post, Patch, Delete, etc.)
- JSON payloads & responses
- Serialisation & Deserialisation
- TCP handshakes for each request if not using HTTP 2
- Hard to fully adhere to all principles: too strict for most apps

Synchronous Communications: gRPC

- gRPC is a cross-platform open source high performance remote procedure call (RPC) framework.
- gRPC was initially created by Google, which used to connect the large number of microservices running within and across its data centers.

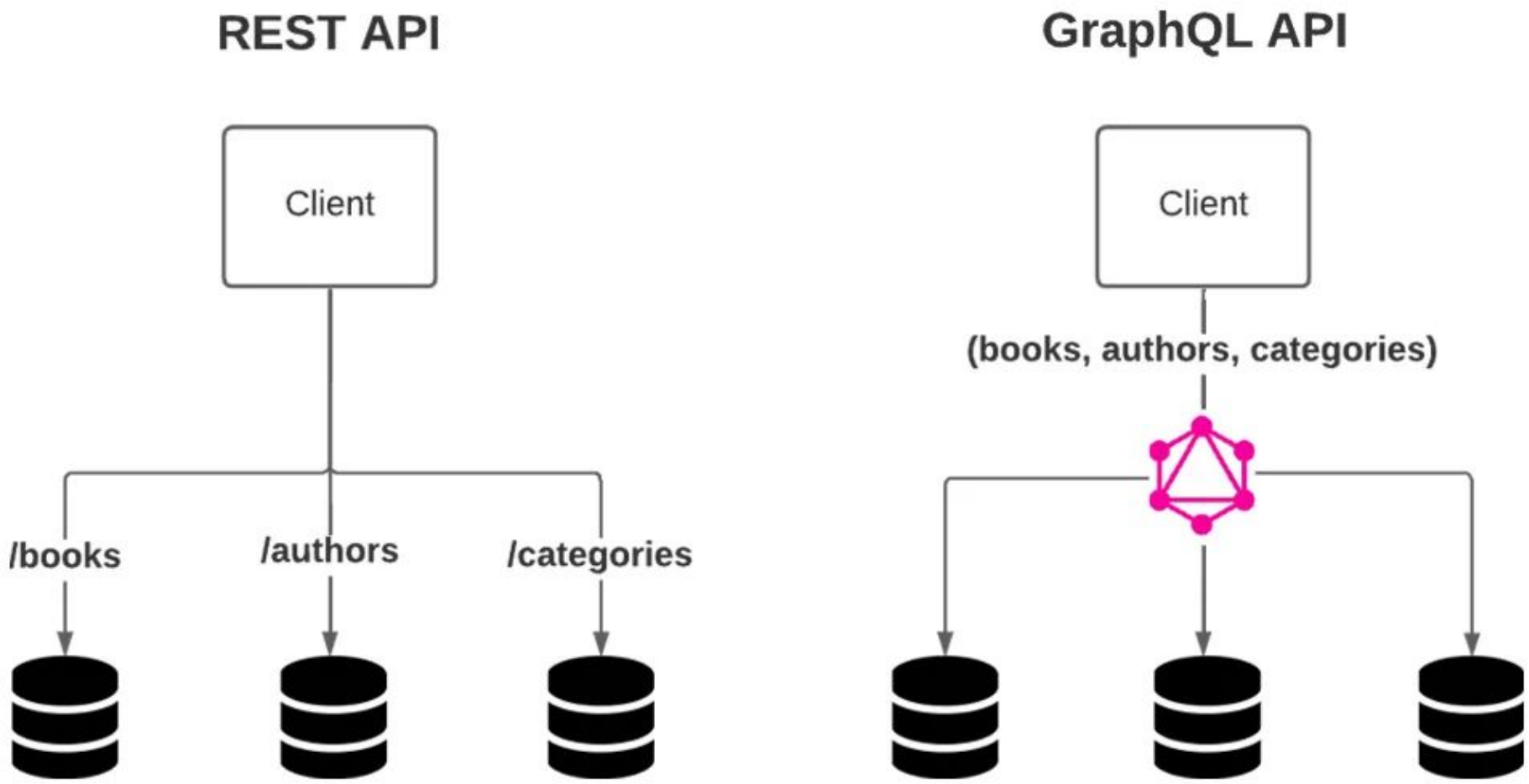
Synchronous Communications: gRPC



Synchronous Communications: GraphQL

- GraphQL is an open-source data query and manipulation language for APIs and a query runtime engine.
- GraphQL's single greatest benefit is the developer experience it provides.

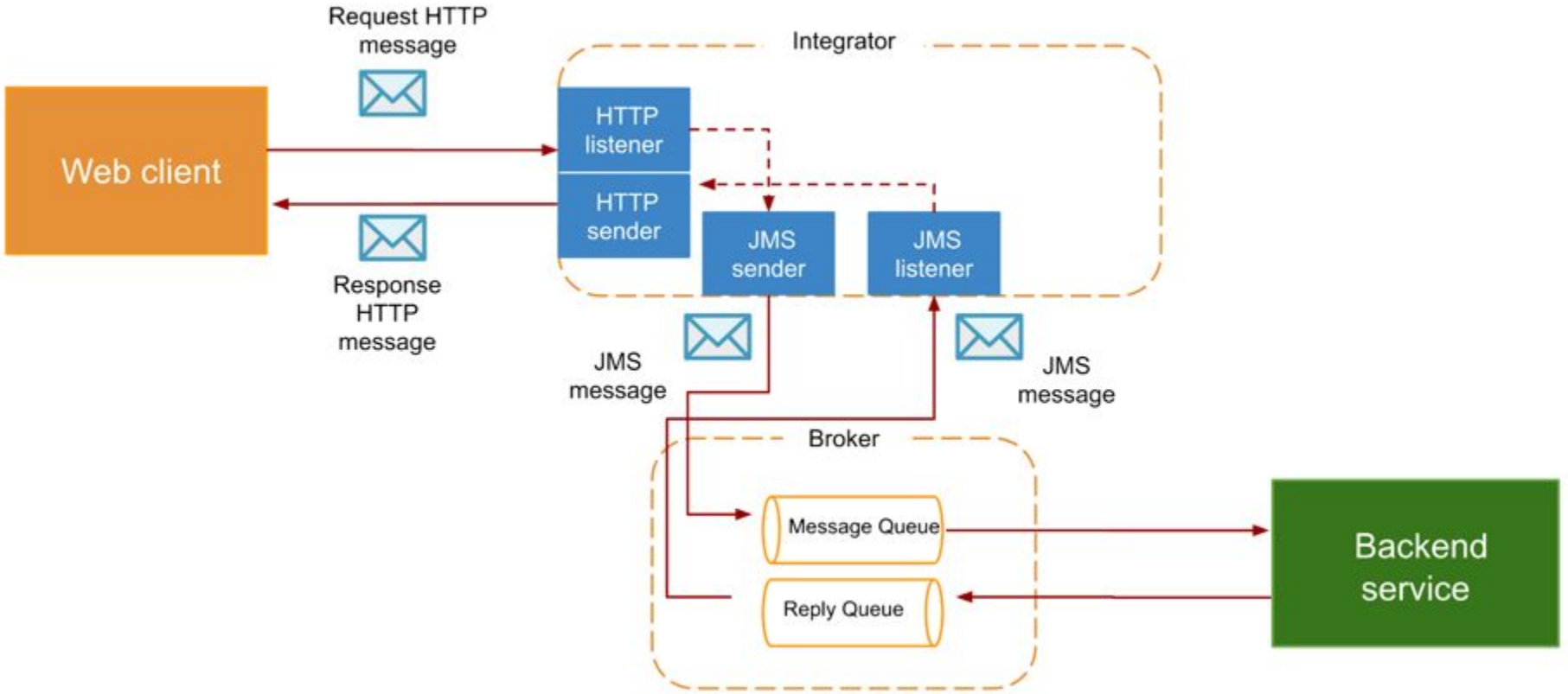
GraphQL vs. REST



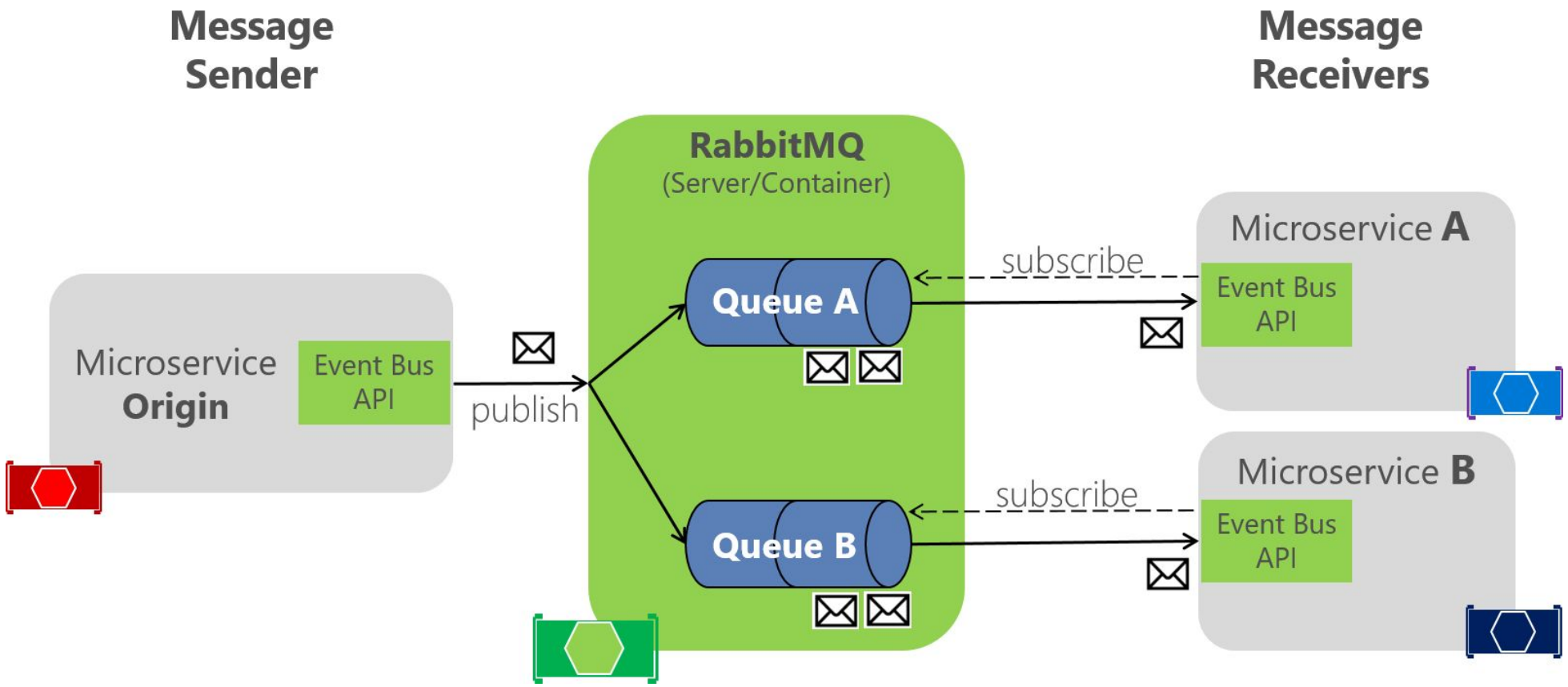
Asynchronous Communications

- Asynchronous communication means interaction without real-time conversation.
  - The operation involves multiple services reacting to it.
  - The operation must be performed while allowing failures & retries.
  - The operation takes a lot of time.

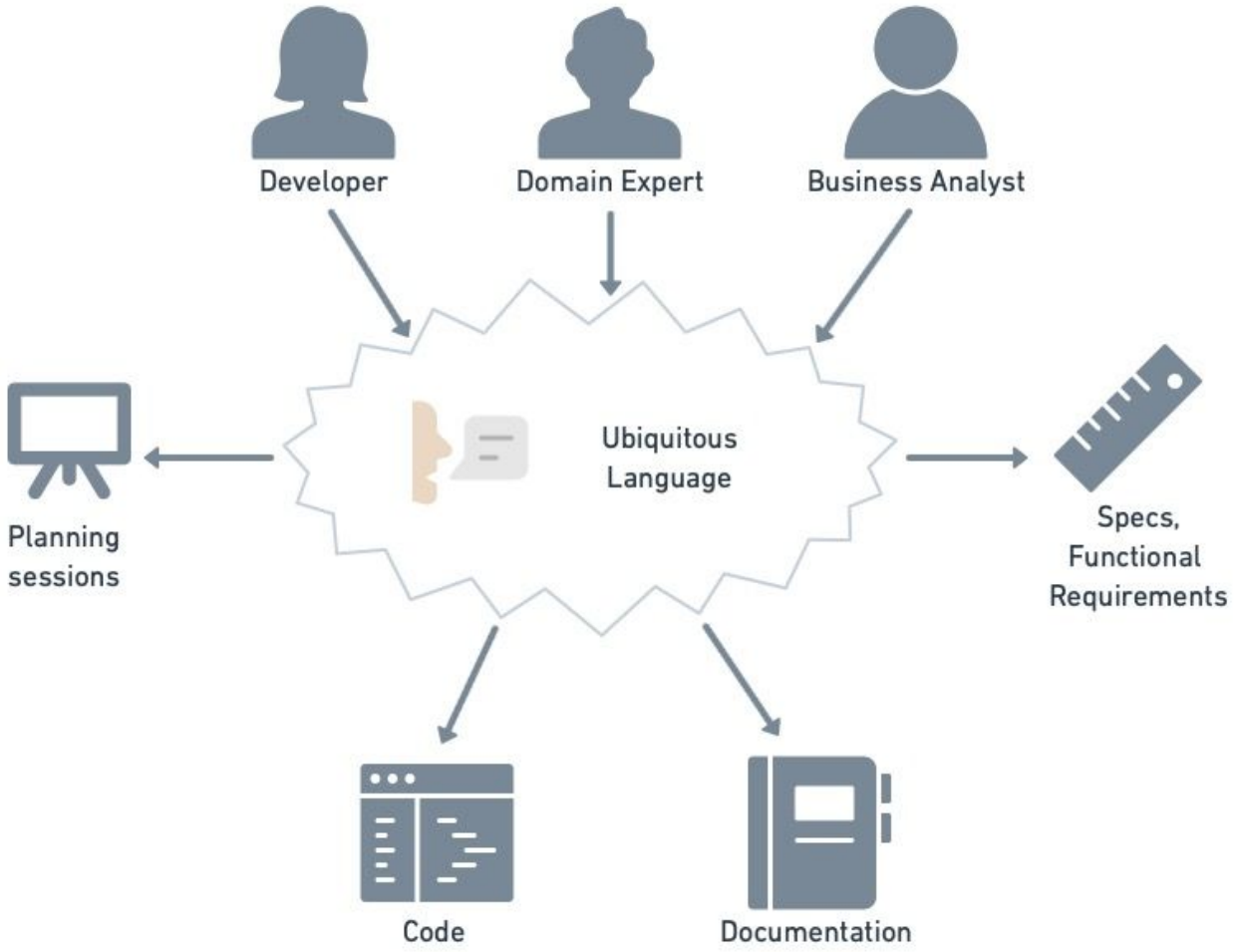
Asynchronous Messaging



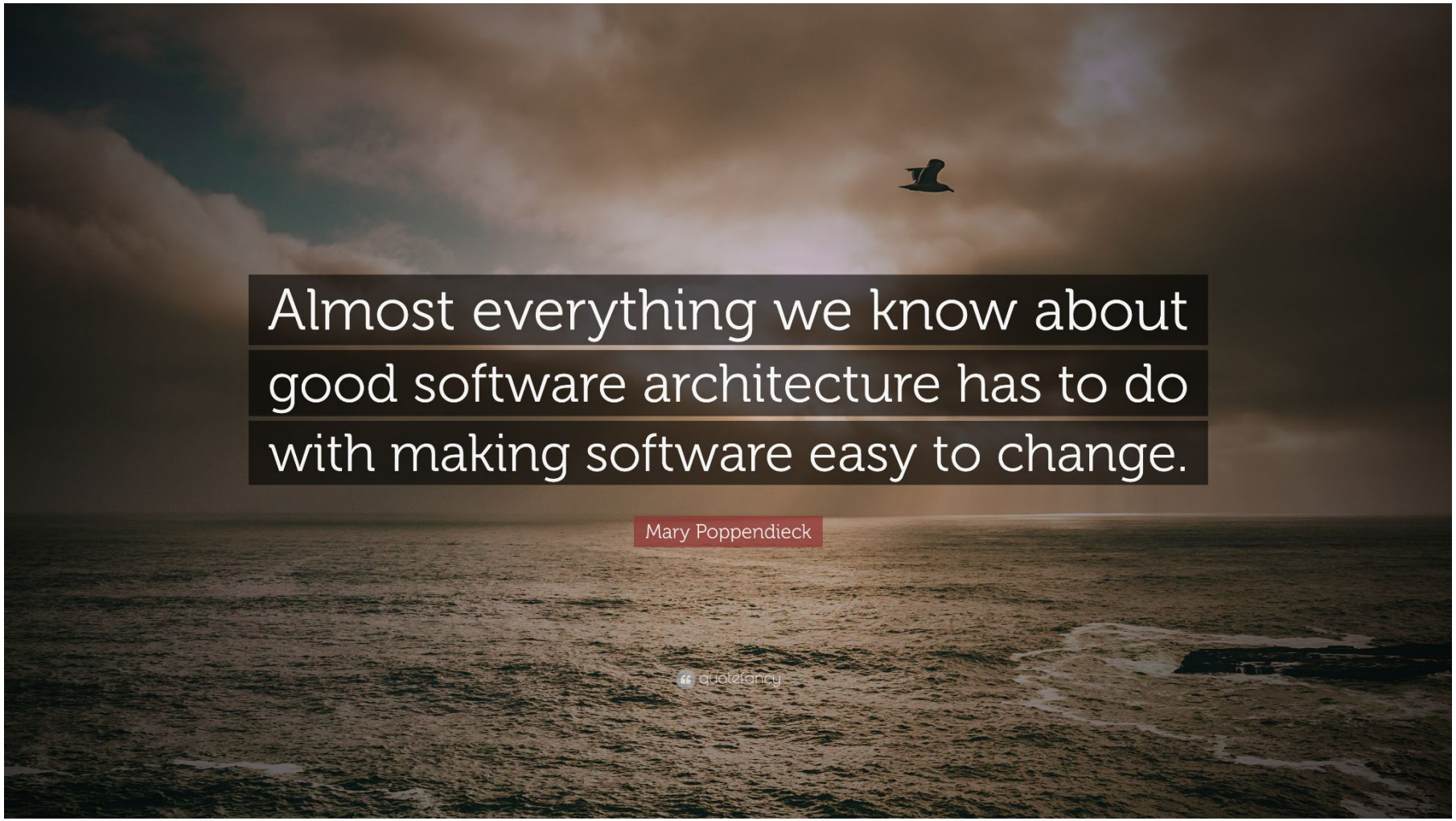
Asynchronous Messaging: RabbitMQ



Domain Driven Design



The Quote of the Day



Readings

- Software Engineering: A Practitioner's Approach, Roger Pressman and Bruce Maxim, 9th Edition, September 2019, Chapters 9 and 10.