الجامعة السورية الخاصة
SYRIAN PRIVATE UNIVERSITY

Topic 4

Software Architecture

Microservices Architecture

Dr. Riad Sonbol

Course Roadmap

- General Principles in Software Design and Architecture: *The Context of Software Architecture, System Decomposition, What is Good Design?*
- Layered Architecture
- Practical Concerns 1: Communication between Layers, Façade Design Pattern, and Dependency Injection
- N-Tier, Client-server, Peer-to-Peer, MVC
- Microservices
- Practical Concerns 2: Message Broker, Related Design Patterns: CQRS, API Gateway, Strangler Pattern, Externalized Configuration, case study
- Clean Architecture: The Hexagonal Architecture, Ports and Adapters
- More Architectural Styles: Repository, Service-Oriented Architecture (SOA), Pipes and Filters

الجامعة السورية الخاصة
SYRIAN PRIVATE UNIVERSITY

Microservices Architecture: introduction

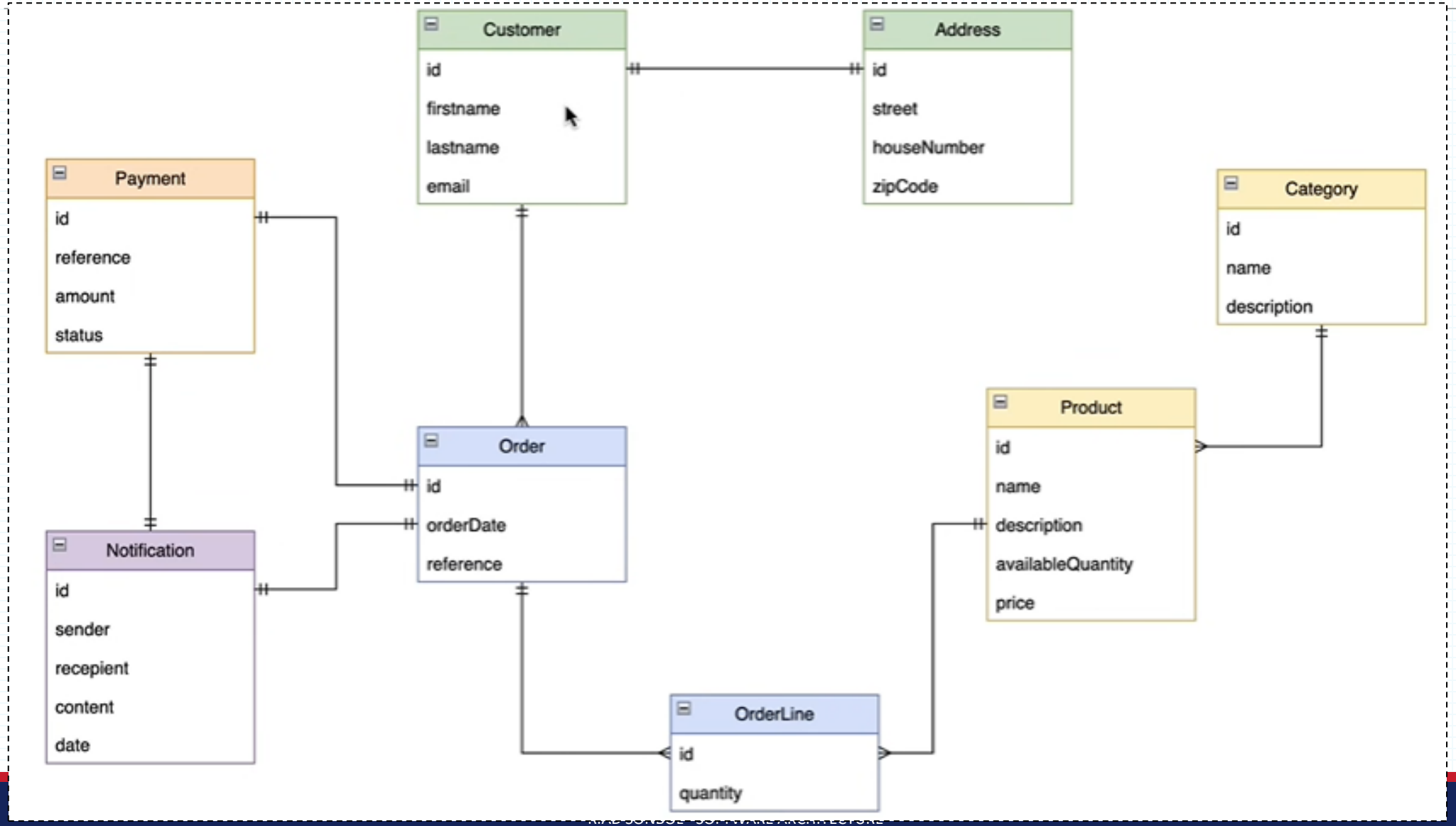
What is Domain-Driven Design?

- Domain-driven design (DDD) is a major software design approach, focusing on modeling software to match a domain according to input from that domain's experts.
- Under domain-driven design, the structure and language of software code (class names, class methods, class variables) should match the business domain
- Approach to software development focusing on domain complexity
- The term was coined by Eric Evans in his book of the same name published in 2003.

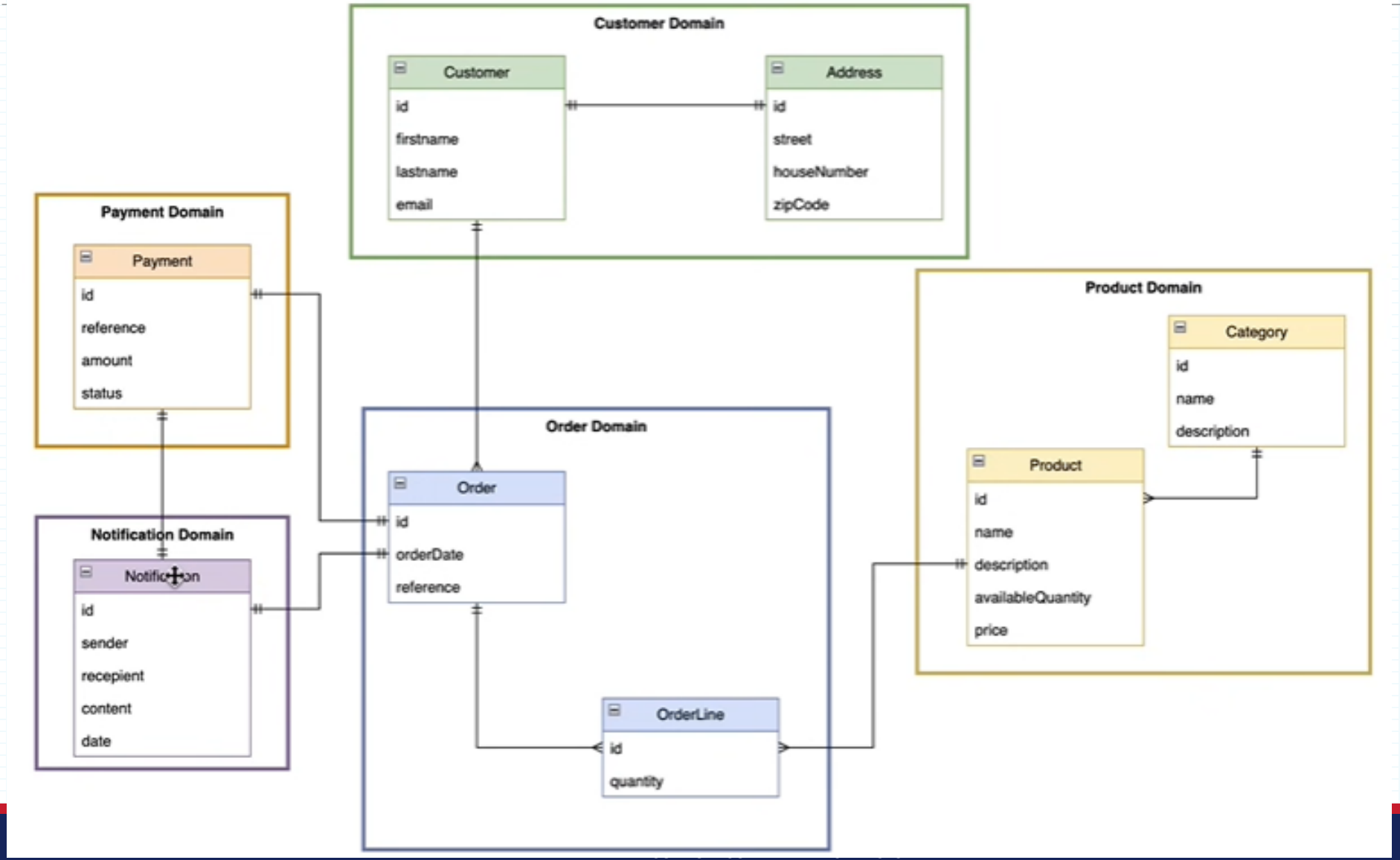
Why Use DDD?

- **Manages complexity**: DDD breaks down complex business logic into smaller, understandable models within clearly defined boundaries.
- **Clear communication using domain language**: It promotes a shared vocabulary (ubiquitous language) between developers and domain experts to avoid misunderstandings.
- **Aligns software with business goals**: DDD ensures the software design reflects real-world business processes and priorities.
- **Foundation for microservices architecture**: Each bounded context in DDD can map naturally to a microservice, encouraging modular and scalable systems.

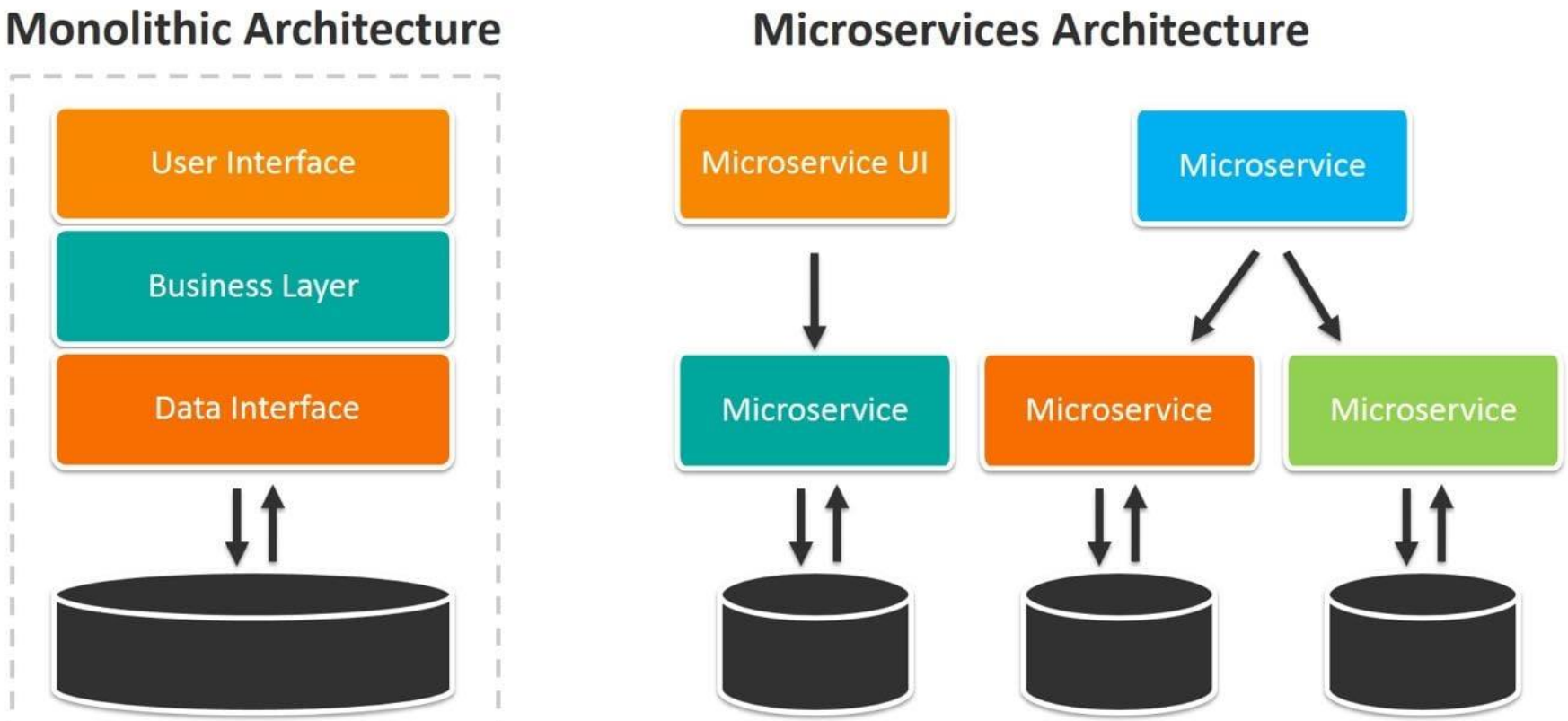
Example


Customer
id
firstname
lastname
email
Address
id
street
houseNumber
zipCode
Payment
id
reference
amount
status
Category
id
name
description
Order
id
orderDate
reference
Product
id
name
description
availableQuantity
price
Notification
id
sender
recepient
content
date
OrderLine
id
quantity

Example


Customer Domain
Customer
id
firstname
lastname
email
Address
id
street
houseNumber
zipCode
Payment Domain
Payment
id
reference
amount
status
Product Domain
Category
id
name
description
Product
id
name
description
availableQuantity
price
Order Domain
Order
id
orderDate
reference
Notification Domain
Notification
id
sender
recepient
content
date
OrderLine
id
quantity

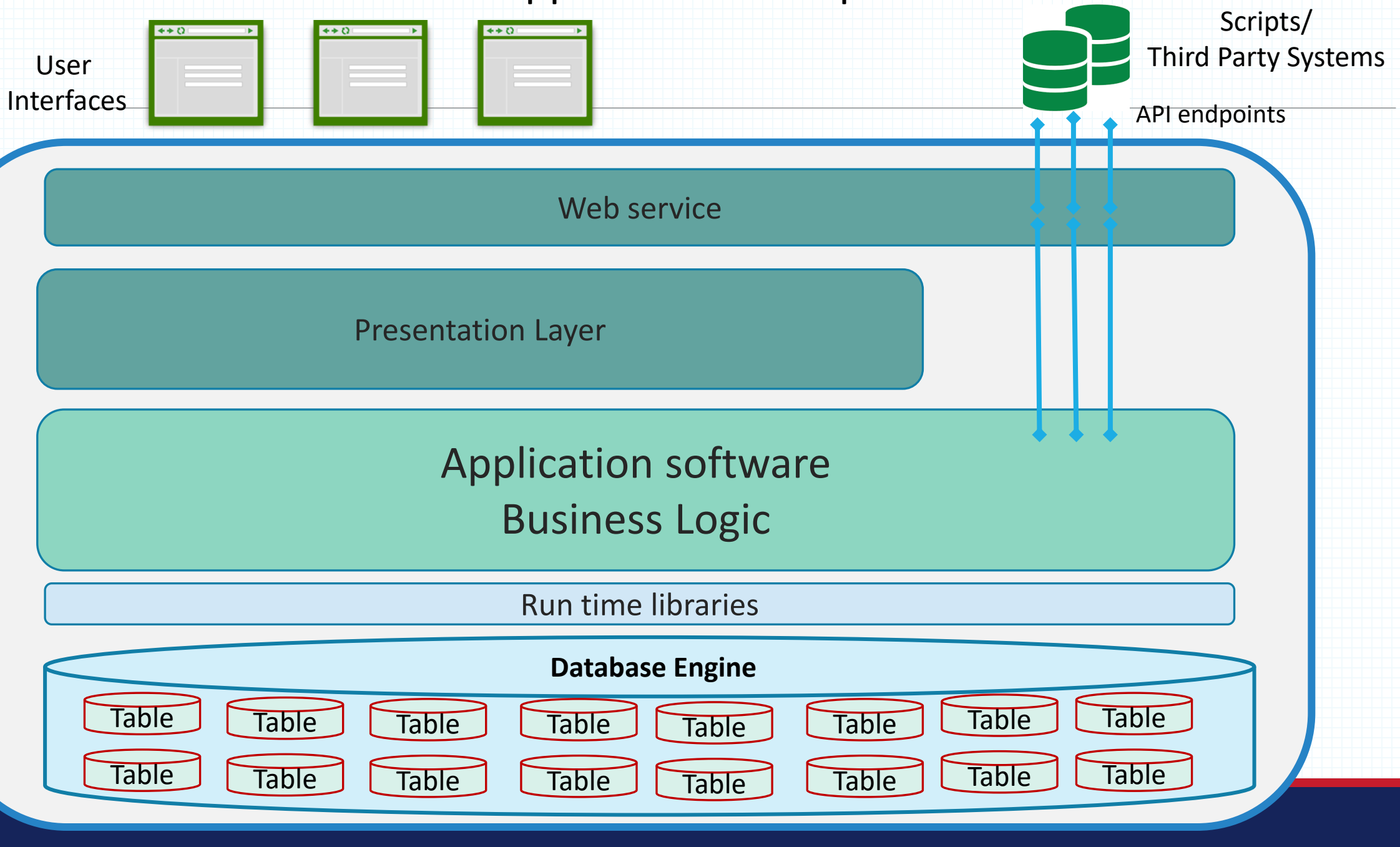
Monolithic vs Microservices



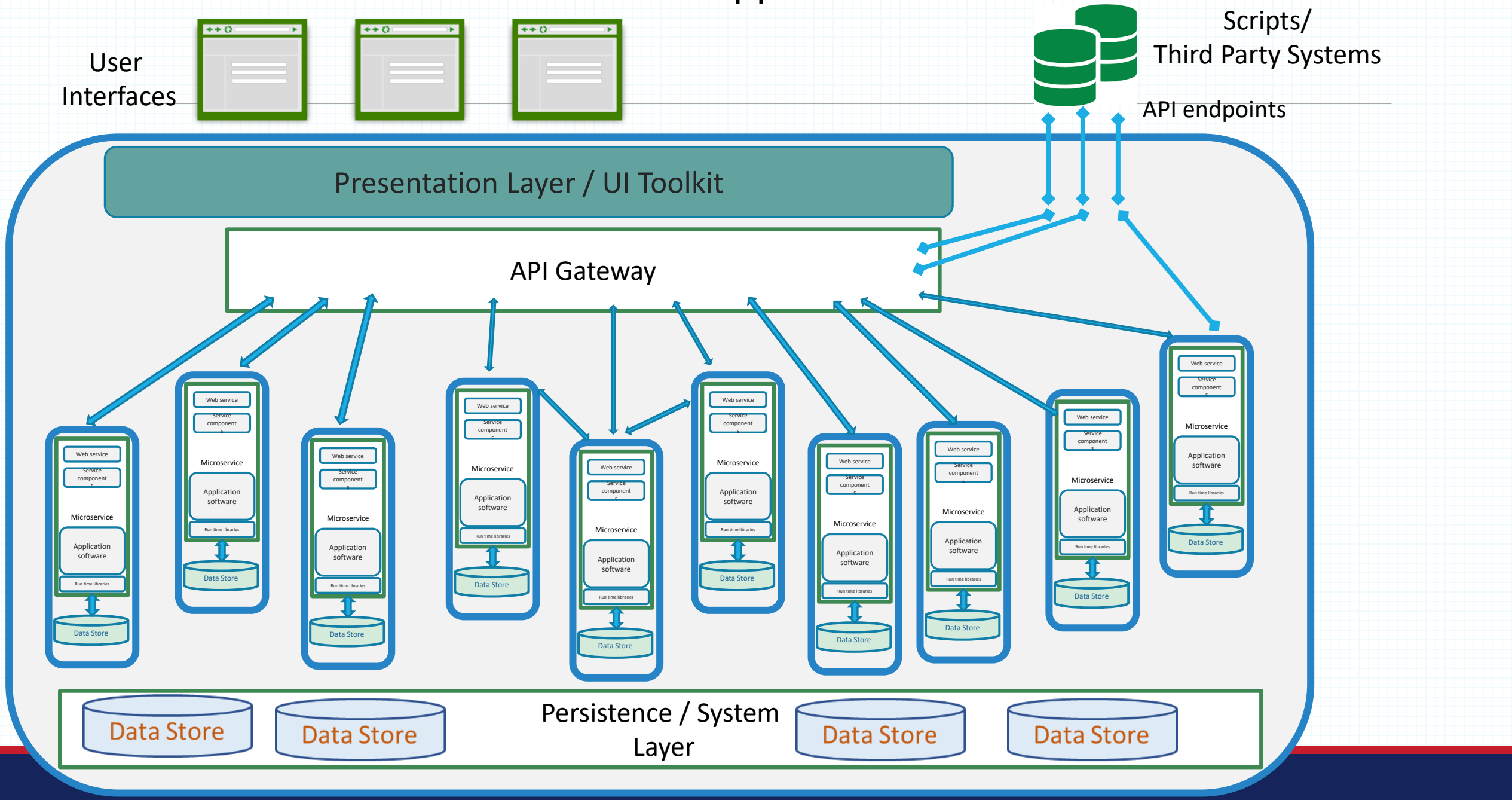
deployed as a single bundle of executables and libraries on a unified platform

Multiple independent software components orchestrated to form a unified application

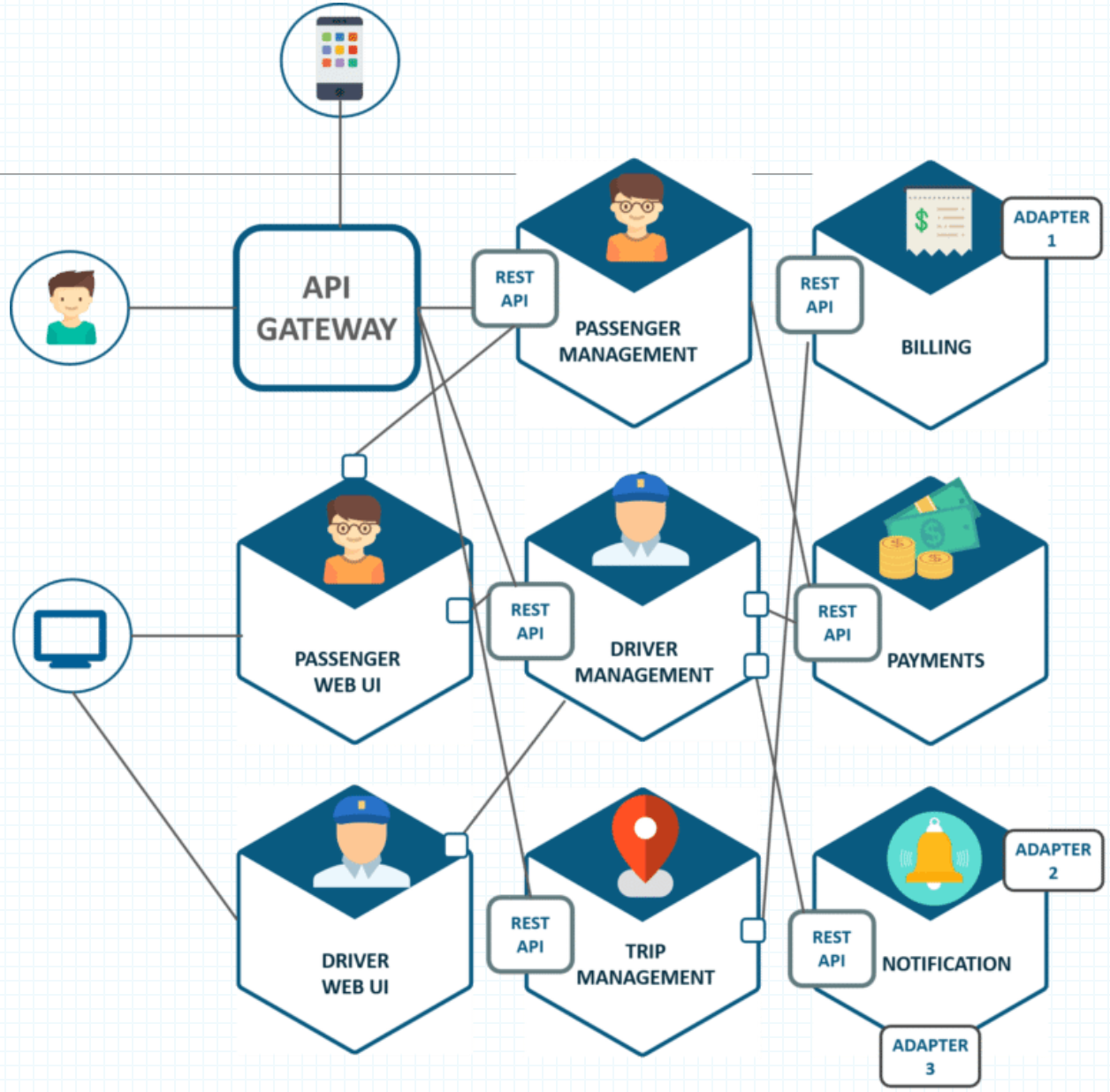
Monolithic Application Conceptual Model



Microservices-based Application



Example


API GATEWAY
REST API
PASSENGER MANAGEMENT
REST API
BILLING
ADAPTER 1
PASSENGER WEB UI
REST API
DRIVER MANAGEMENT
REST API
PAYMENTS
DRIVER WEB UI
REST API
TRIP MANAGEMENT
REST API
NOTIFICATION
ADAPTER 2
ADAPTER 3

Why Microservices!

- Successful applications often live a very long time + **Technology changes**

⇒ Need to be able to easily “modernize” application!

⇒ Need to deliver changes rapidly, frequently, and reliably.

- More complex Applications.. App **keep growing up**

⇒Need to divide team

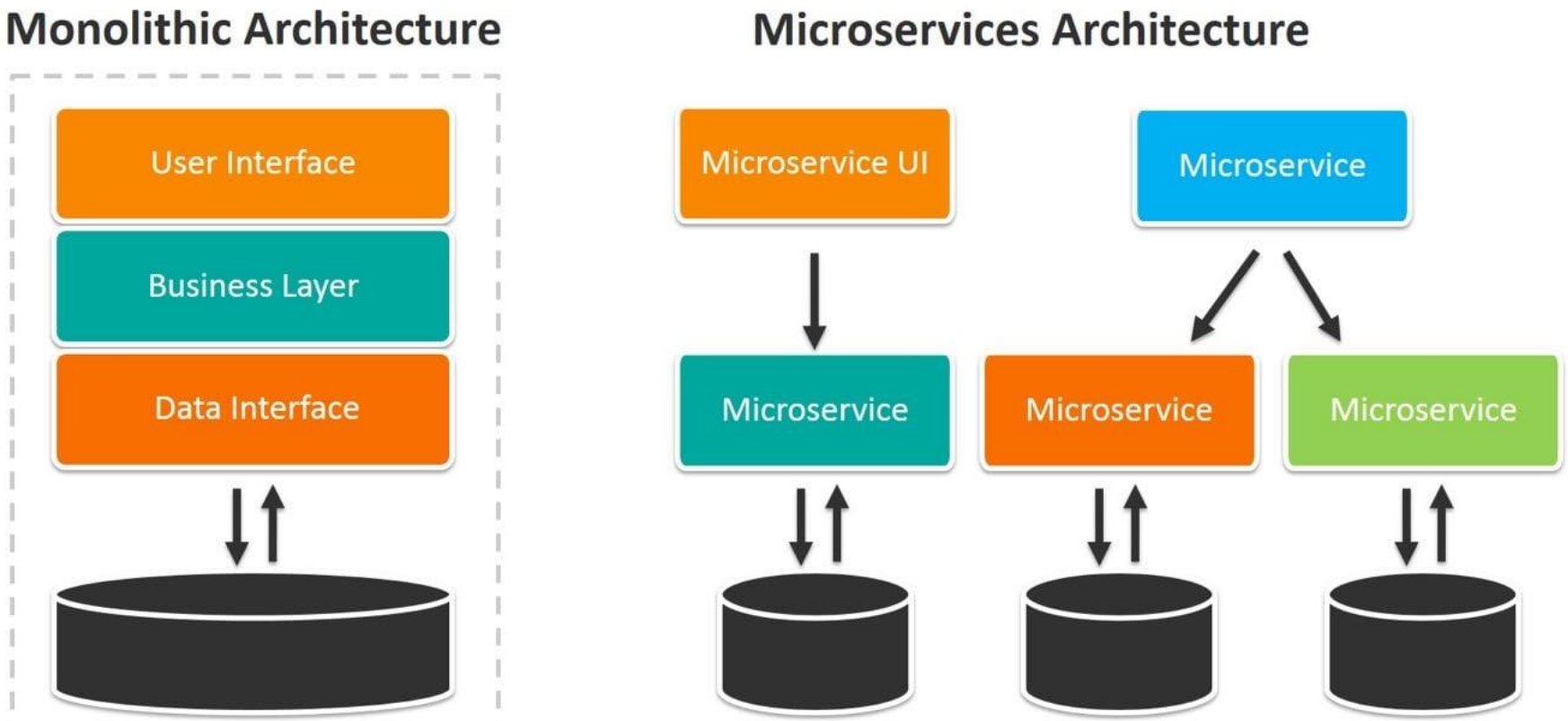
⇒Need to improve testability, deployability, maintainability, modularity, evolvability.

Characteristics of Microservices

- **Independently deployable**: Each microservice can be developed, deployed, and updated without affecting other services.
- **Decentralized data management**: Every microservice owns and manages its own database, avoiding tight data coupling between services.
- **Technology agnostic**: Microservices can be built using different programming languages, databases, and frameworks, depending on the service's needs.
- **Focused on a single business capability**: Each microservice is designed to handle one specific business function, making the system modular and easier to understand.

Microservices Architecture: Practical Concerns

Monolithic vs Microservices



deployed as a single bundle of executables and libraries on a unified platform

Multiple independent software components orchestrated to form a unified application

But.. Loose coupling is essential!

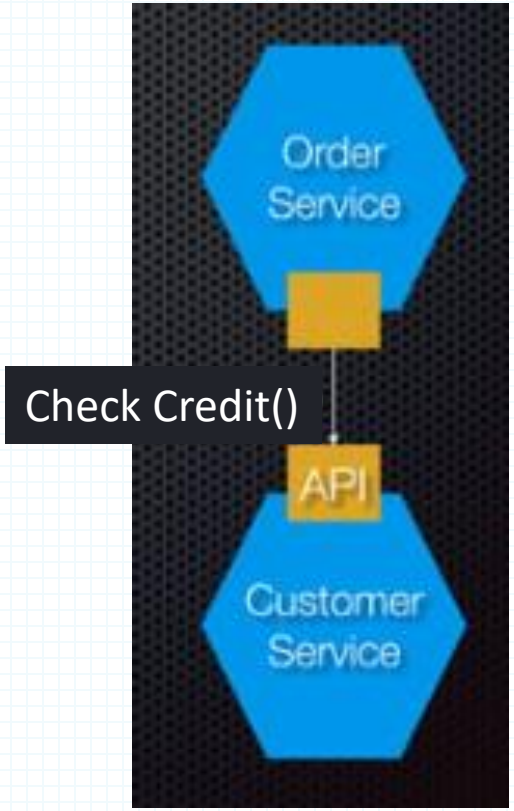
- **Service collaborate.. Thus:**

⇒ Design Time Coupling:

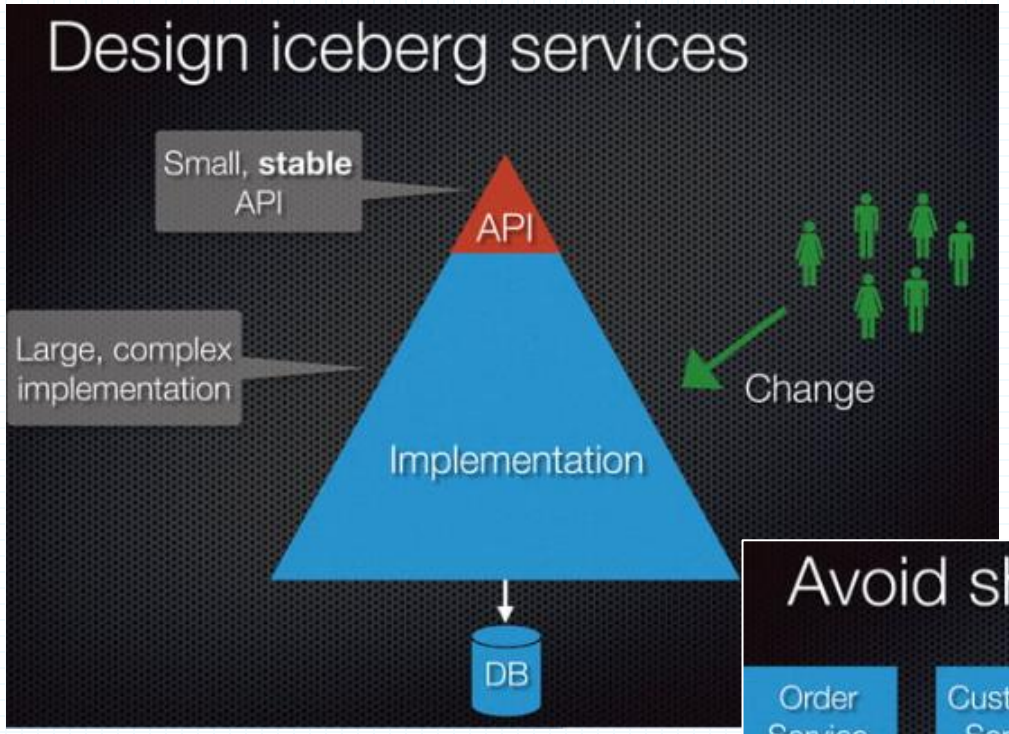
Change Service A -> change Service B

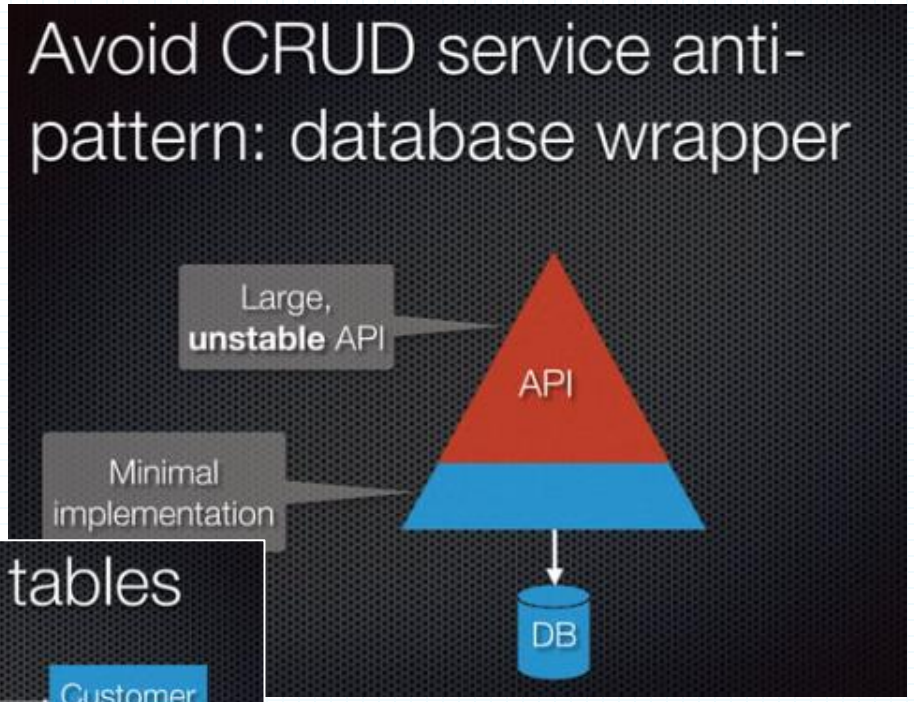
⇒ Runtime Coupling:

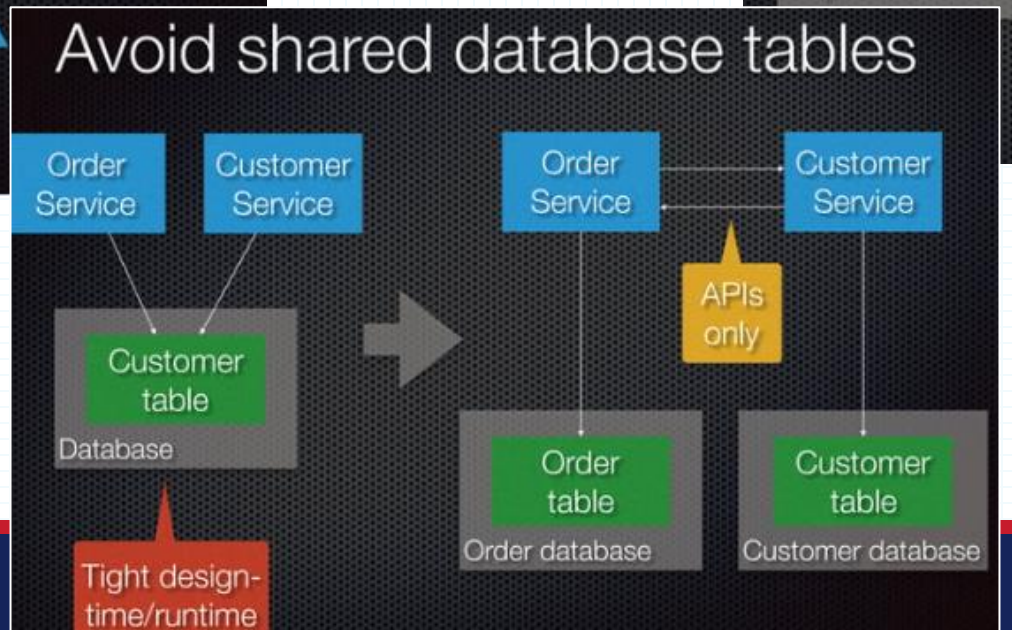
Service A cannot respond to a synchronous request until service B responds



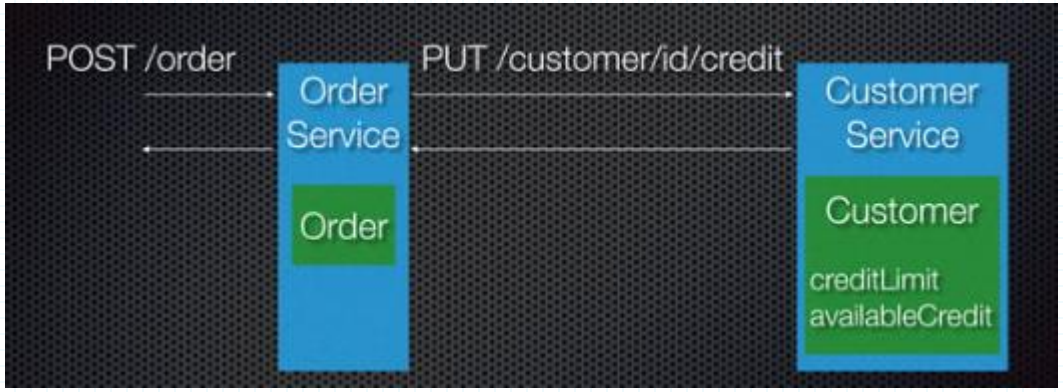
Design Time Coupling… Solutions

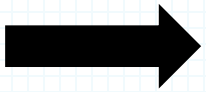


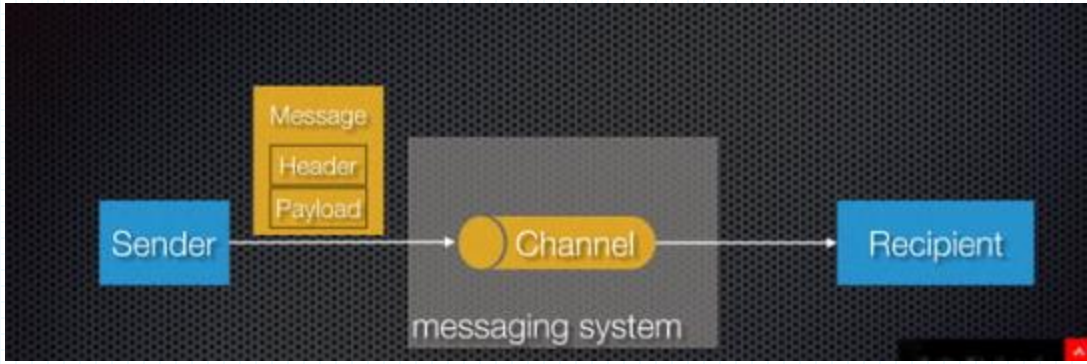




Runtime Coupling … Solution







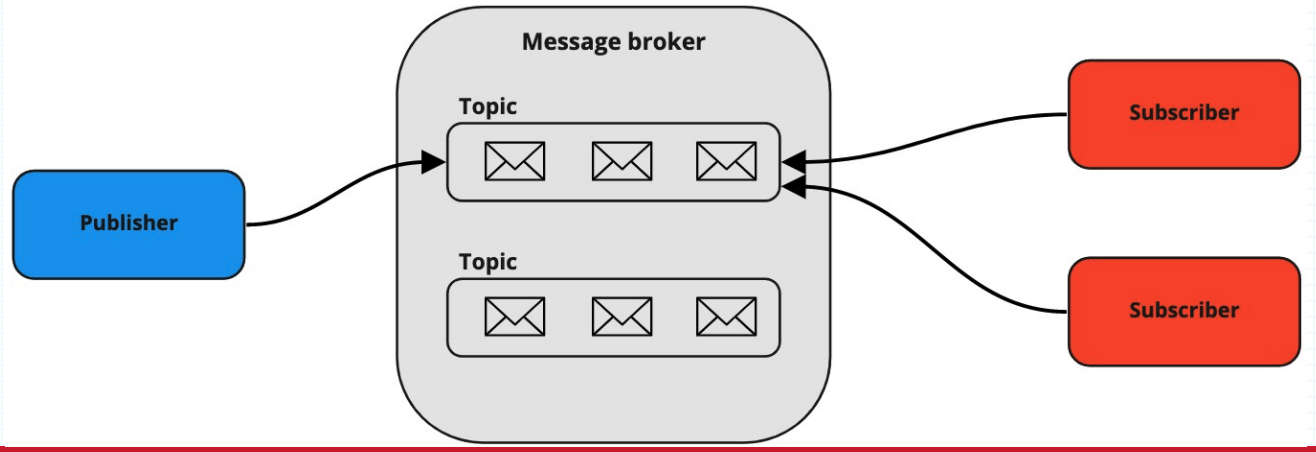
Reduce Availability!

availability(createOrder) =
availability(OrderService) x
availability(CustomerService)

Use Async Messaging

What is a Message Broker?

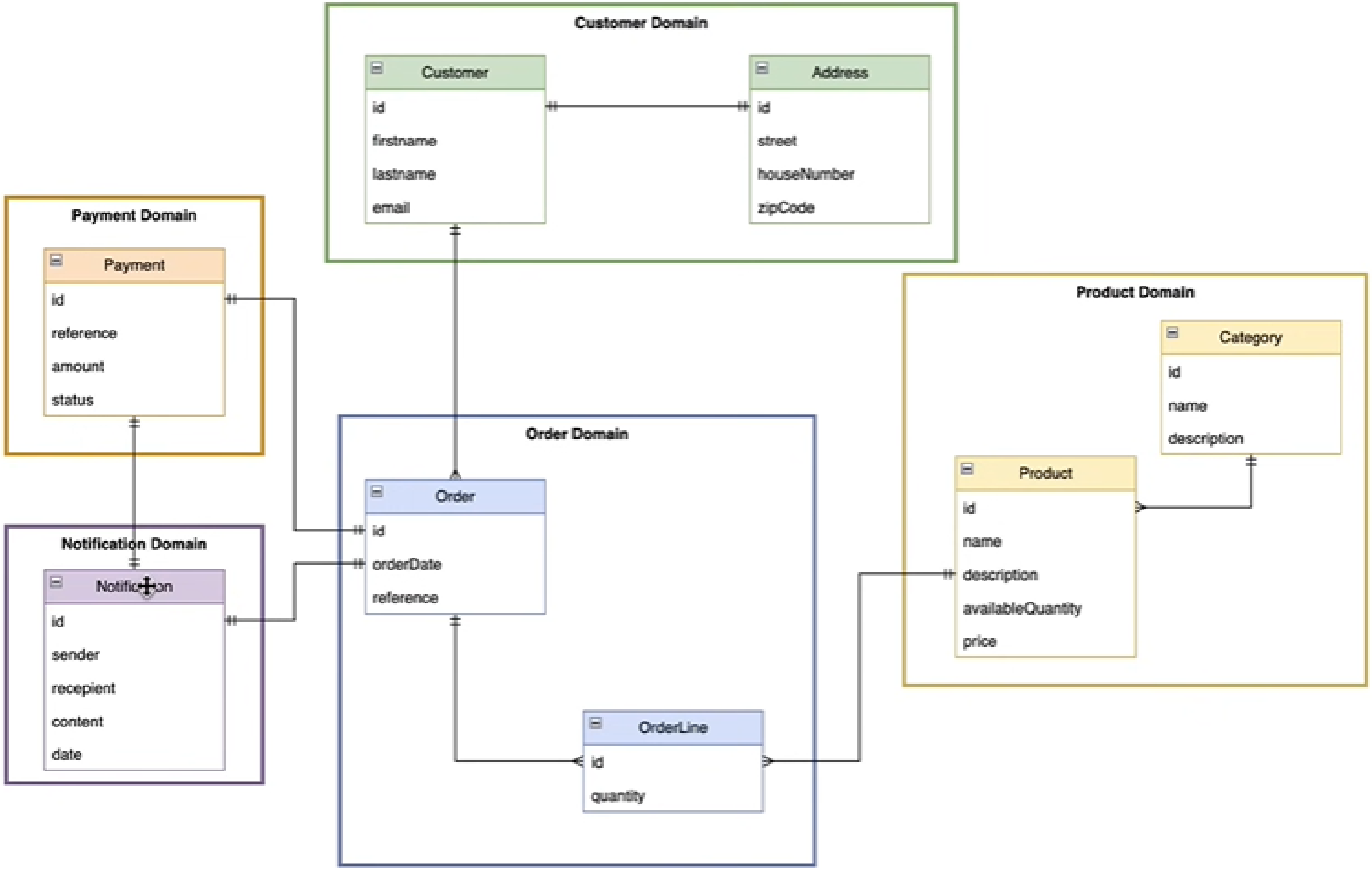
- **Middleware for message transmission between services**: A message broker acts as an intermediary that routes messages between services to enable smooth and reliable communication.
- **Decouples sender and receiver**: It allows services to interact without needing to know each other directly, increasing flexibility and resilience.
- **Supports asynchronous communication**: Message brokers let services send and receive messages without blocking, enabling faster and more scalable systems.



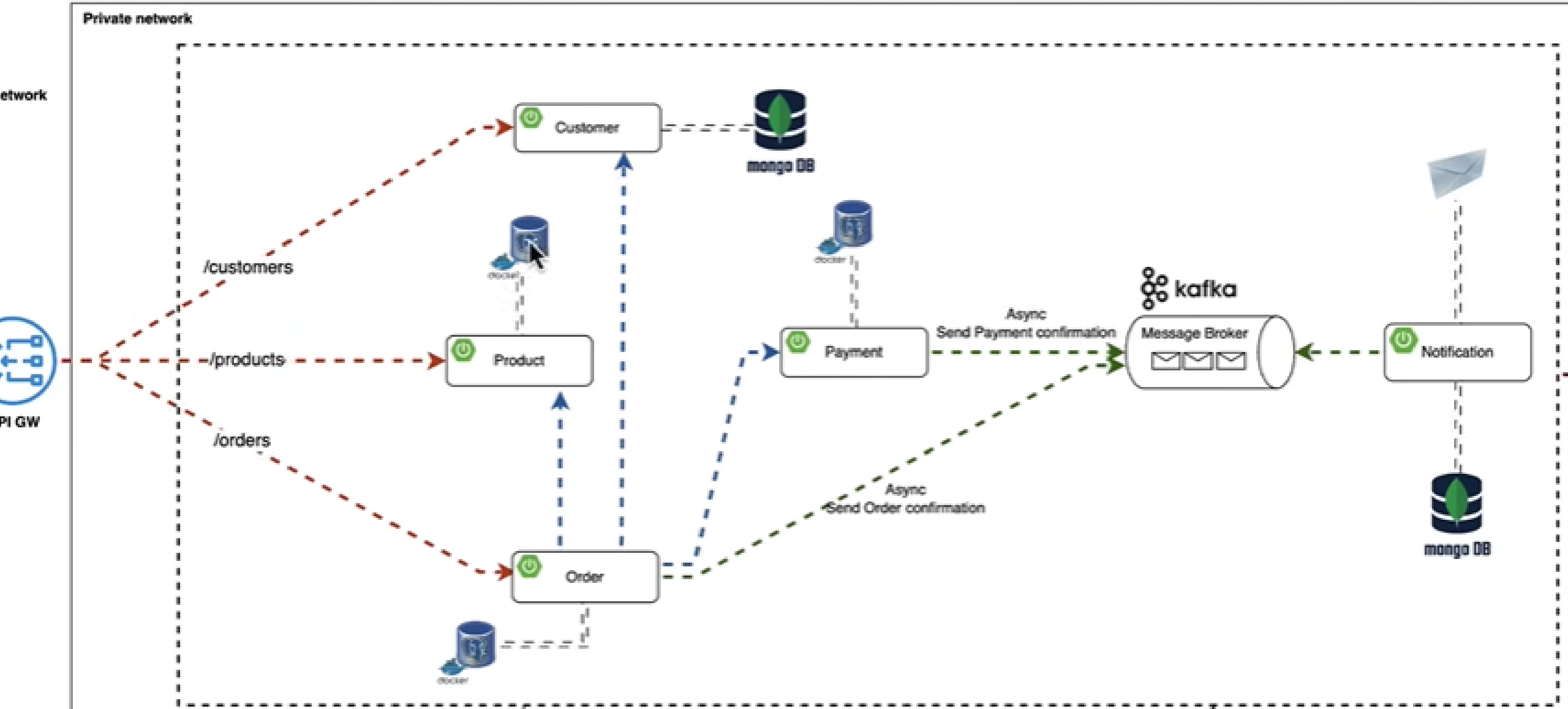
Popular Message Brokers

- RabbitMQ: Queue-based, AMQP support
- Apache Kafka: Distributed, high-throughput, stream processing

Example

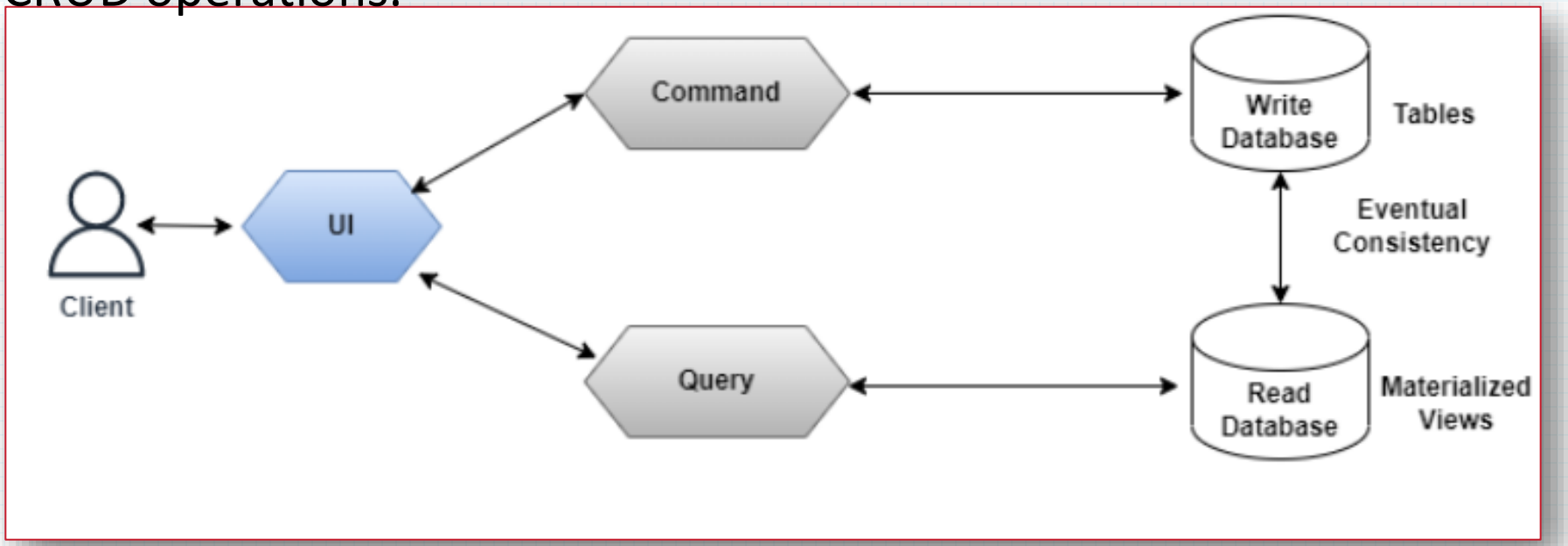

Customer Domain
Customer
id
firstname
lastname
email
Address
id
street
houseNumber
zipCode
Payment Domain
Payment
id
reference
amount
status
Order Domain
Order
id
orderDate
reference
Product Domain
Category
id
name
description
Product
id
name
description
availableQuantity
price
Notification Domain
Notification
id
sender
recepient
content
date
OrderLine
id
quantity

More details example



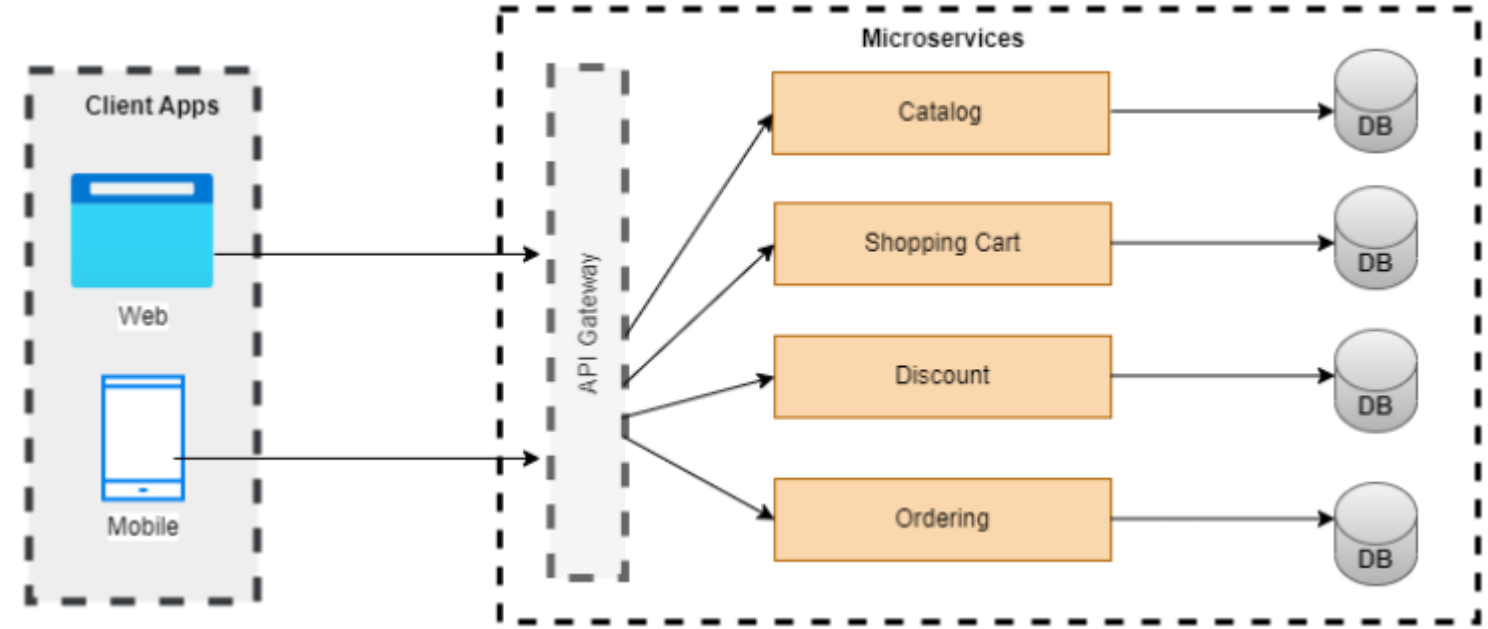
Related Design Patterns
(Command and Query Responsibility Segregation - CQRS)

- CQRS offers to use “separation of concerns” principles and separate reads and writes into two databases.
- By this principle, we can use different databases for reading and writing database types, like using NoSQL for reading and using a relational database for CRUD operations.



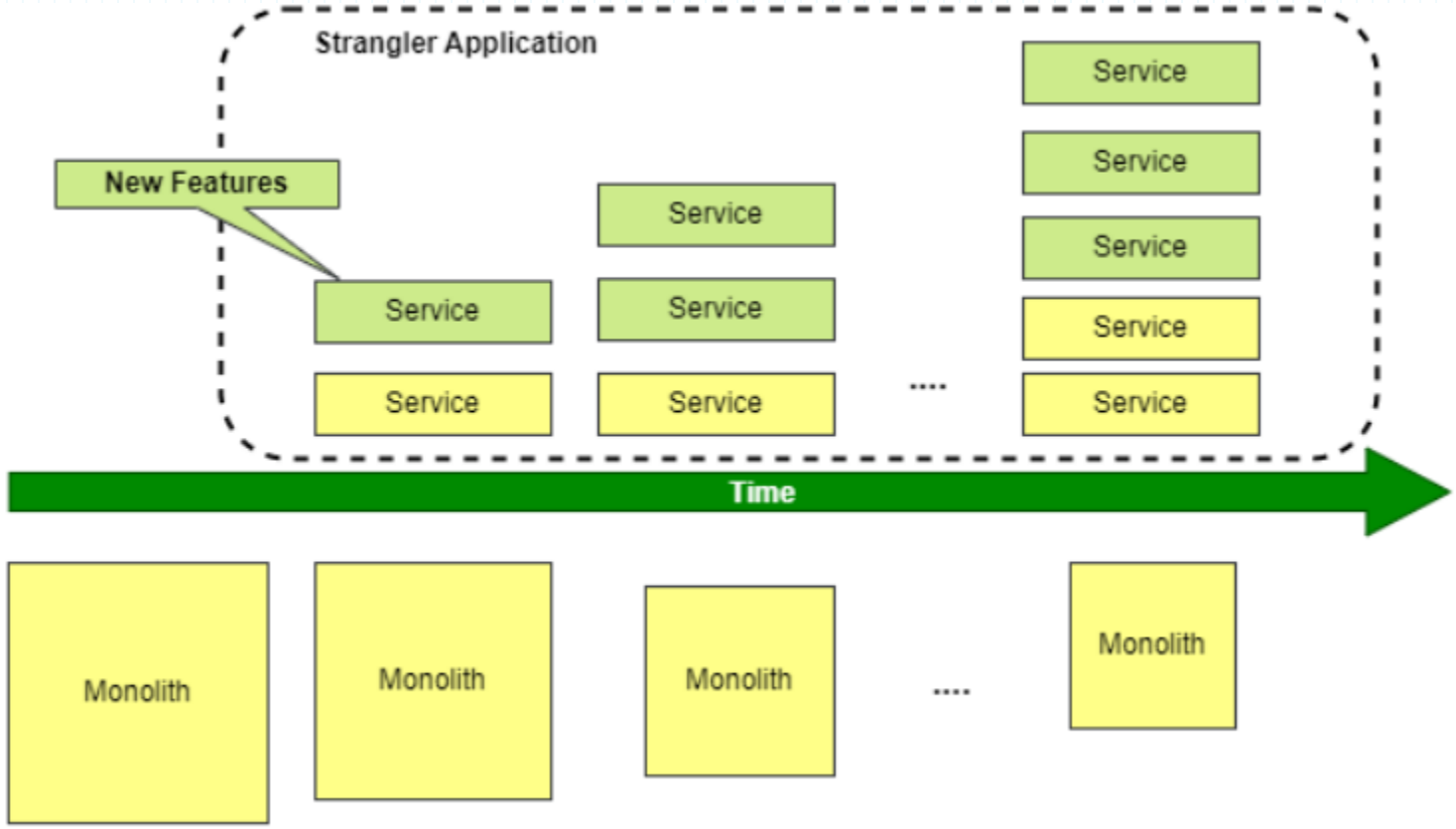
Related Design Patterns
(API Gateway)

- API gateway acts as communication between the client and underlying services and serves as a single-entry point to the system while abstracting the underlying architecture.
- It also provides cross-cutting concerns like authentication, SSL termination, and caching



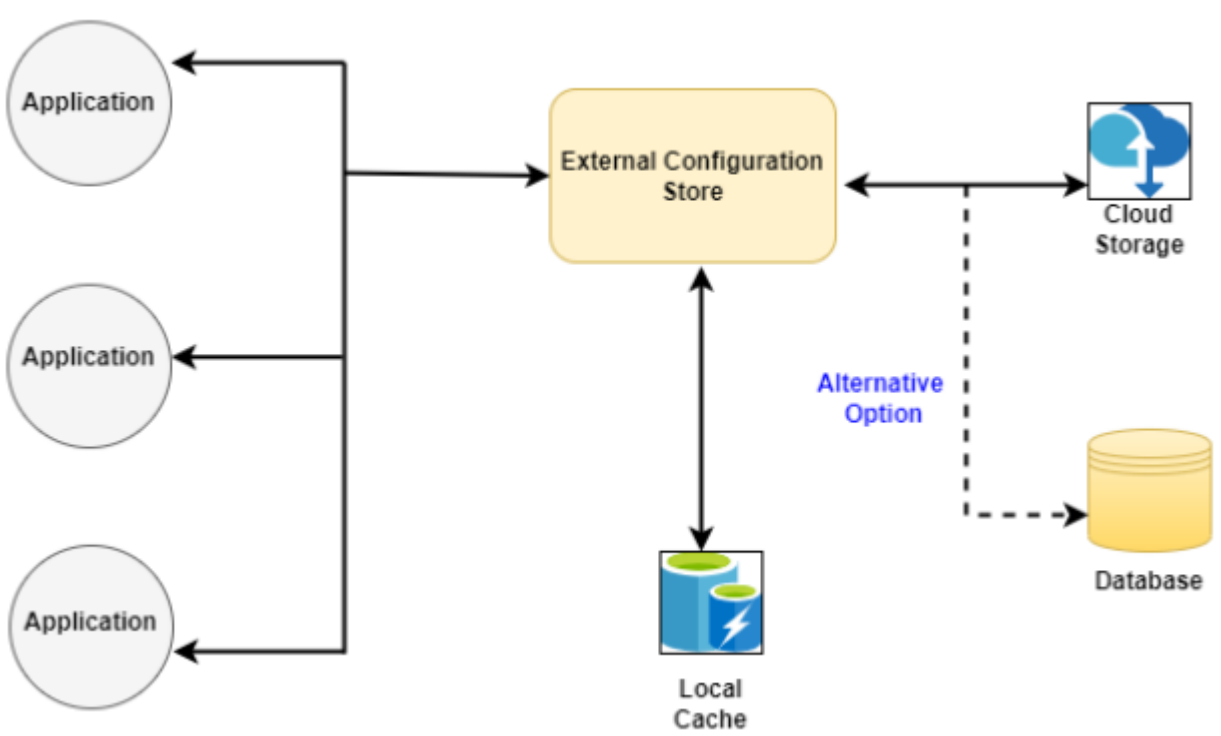
Related Design Patterns
(Strangler Pattern)

- to gradually migrate a Monolithic application to a Microservices-based architecture.
- This pattern is useful when the application is too large and complex to migrate simultaneously.

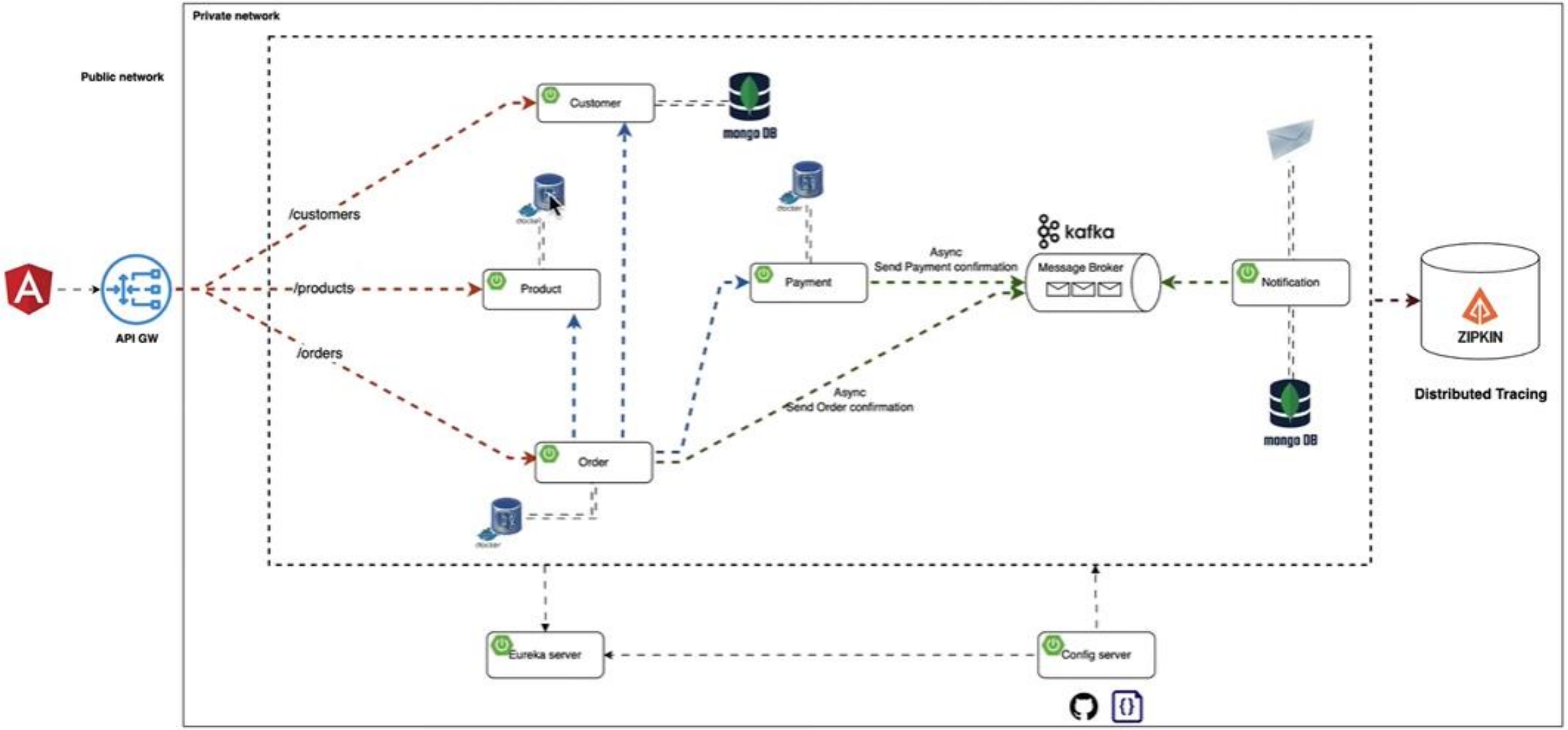


Related Design Patterns
(Externalized Configuration)

- Configuration Externalization is a design pattern where the configuration settings of an application are stored outside the codebase.
- This allows you to modify settings without changing or redeploying the application, enhancing flexibility, maintainability, and portability across different environments like development, testing, and production.



The Complete Example



حسن التصميم السابق باستخدام ما تعرفنا عليه في محاضرة اليوم!

حالة عملية للنقاش

شركة تطوير تعمل على بناء نظام برمجي لإدارة خدمات الرعاية الصحية عن بُعد، ويتميز النظام المراد بالخصائص التالية:

- يتكامل النظام مع أجهزة قياس العلامات الحيوية للمرضى لتحصيل البيانات والقياسات الحيوية مباشرة لتحليلها.
- يوفر واجهة مستخدم تفاعلية للأطباء والممرضين تتيح لهم مراقبة الحالة الصحية للمرضى بشكل لحظي عند الحاجة وتقديم الاستشارات عبر الإنترنت.
- يتيح النظام للمرضى حجز المواعيد والتواصل مع الأطباء عبر المحادثات النصية أو مكالمات الفيديو.
- يقدم النظام توصيات طبية مخصصة بناءً على بيانات المرضى وسجلهم الصحي باستخدام تقنيات الذكاء الصنعي.
- يوفر النظام خدمة إدارة الوصفات الطبية الإلكترونية، حيث يمكن للأطباء إصدار الوصفات وتحويلها مباشرة إلى الصيدليات.
- يدعم النظام تخزين الملفات الطبية بشكل آمن مع إمكانية مشاركتها مع مقدمي الرعاية الصحية بناءً على موافقة المريض.