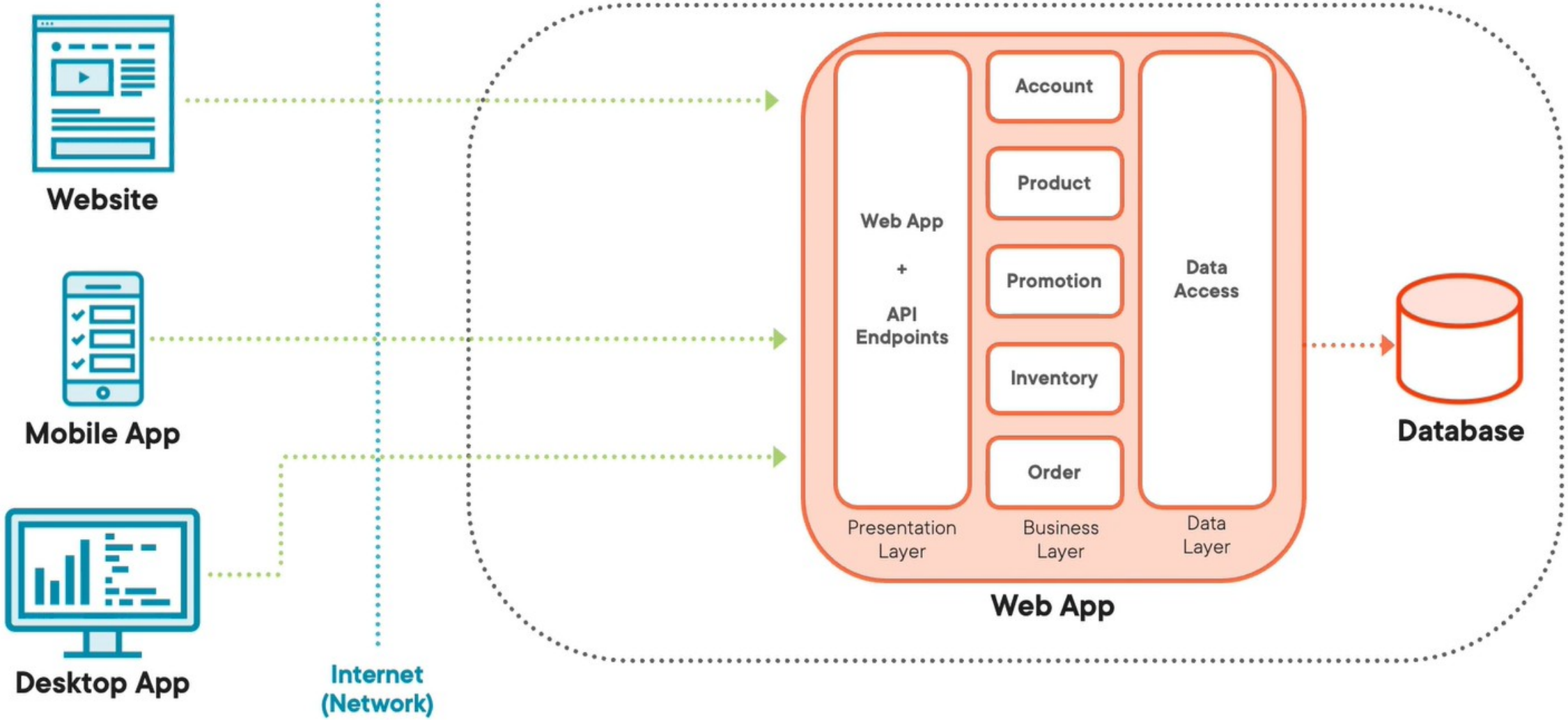
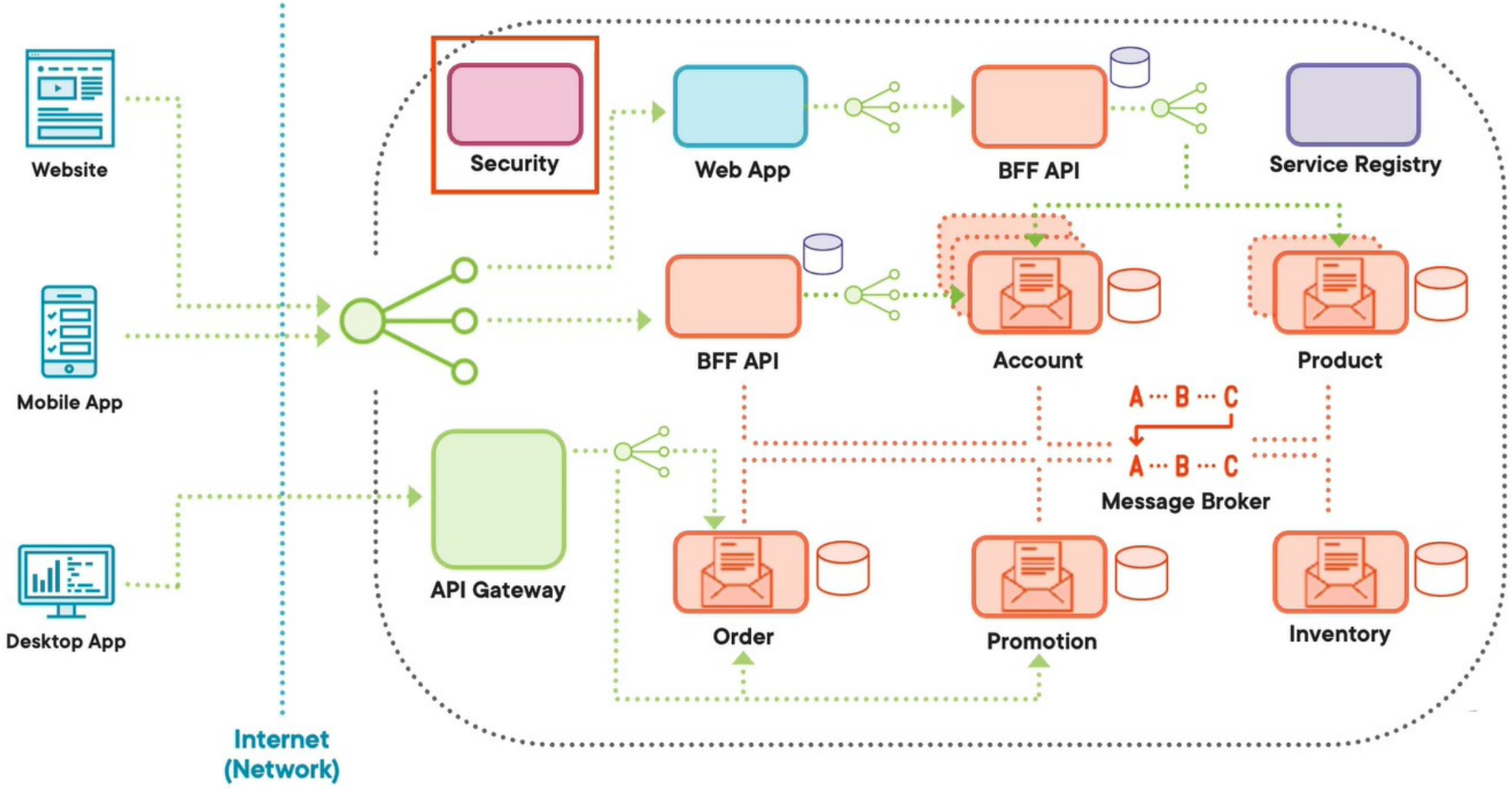
Capgemini

Microservices architecture

Traditional applications



Microservices Applications



Pros/Cons

Monolithic

- Simplicity of development
- Ease of deployment
- Fewer security concerns
- Better performance
- Easy to scale

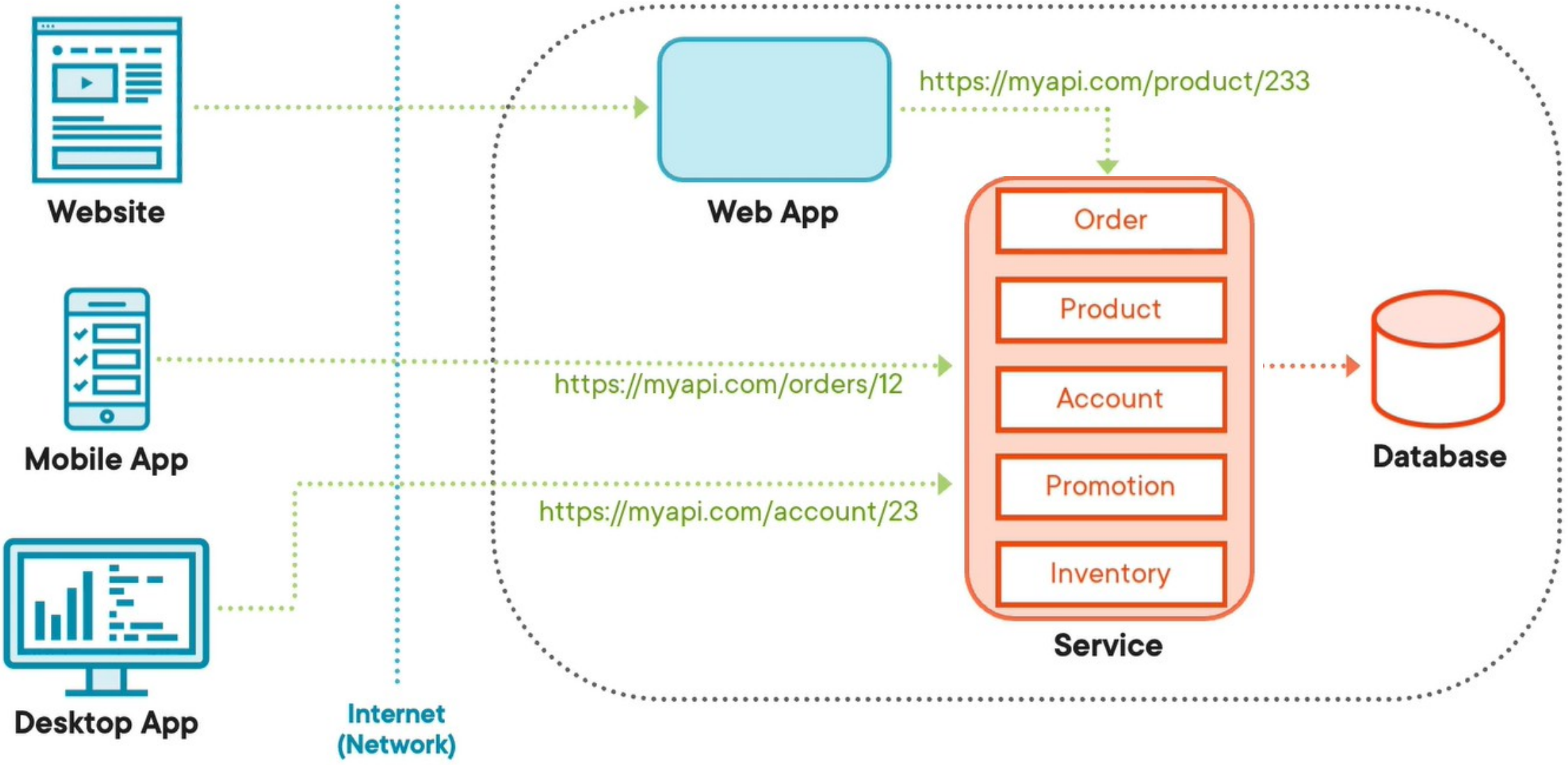
- Limited scalability
- Difficulty in maintenance - Rigidity
- Higher risk - Error dependency

Microservices

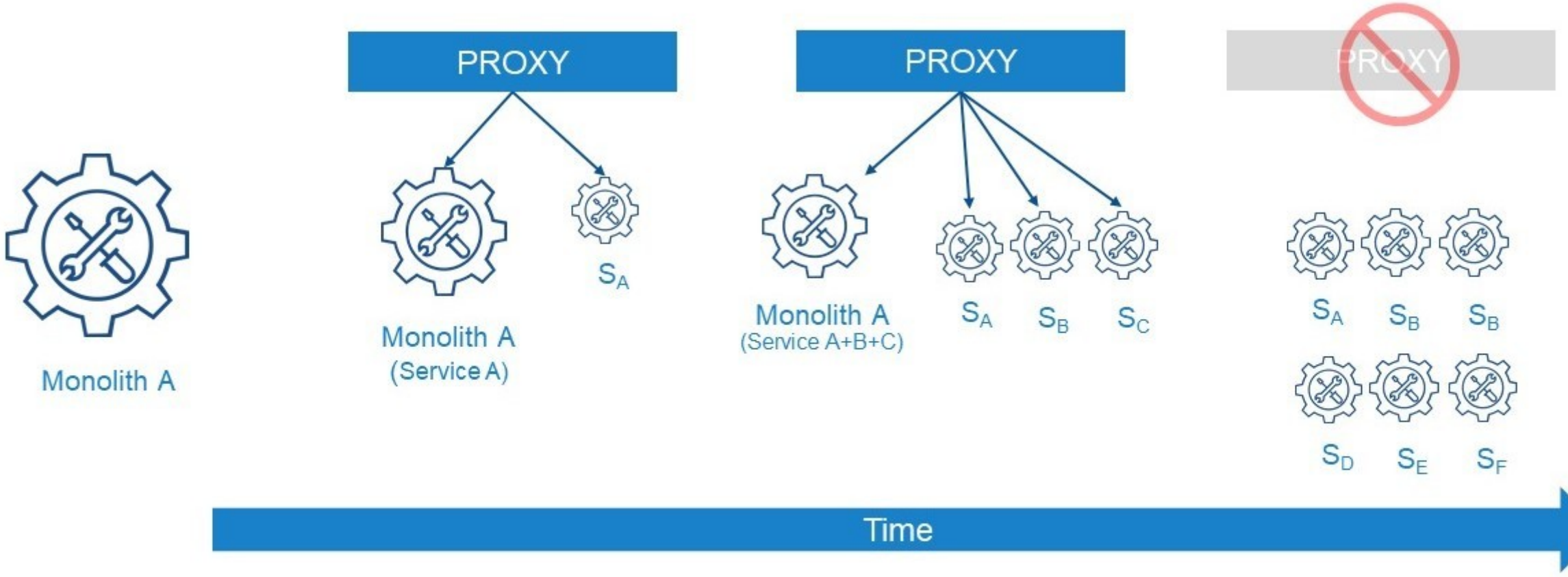
- Scalability
- Simple deployment
- Reusable code (different programming languages)
- Agility in changes
- Independent application
- Lower risk (fails)
- More efficient costs

- Complexity
- Latency
- Higher overhead
- Difficulty in transaction
- Greater complexity in configuration management
- Greater complexity in monitoring and debugging

Separate presentation layer from service layer



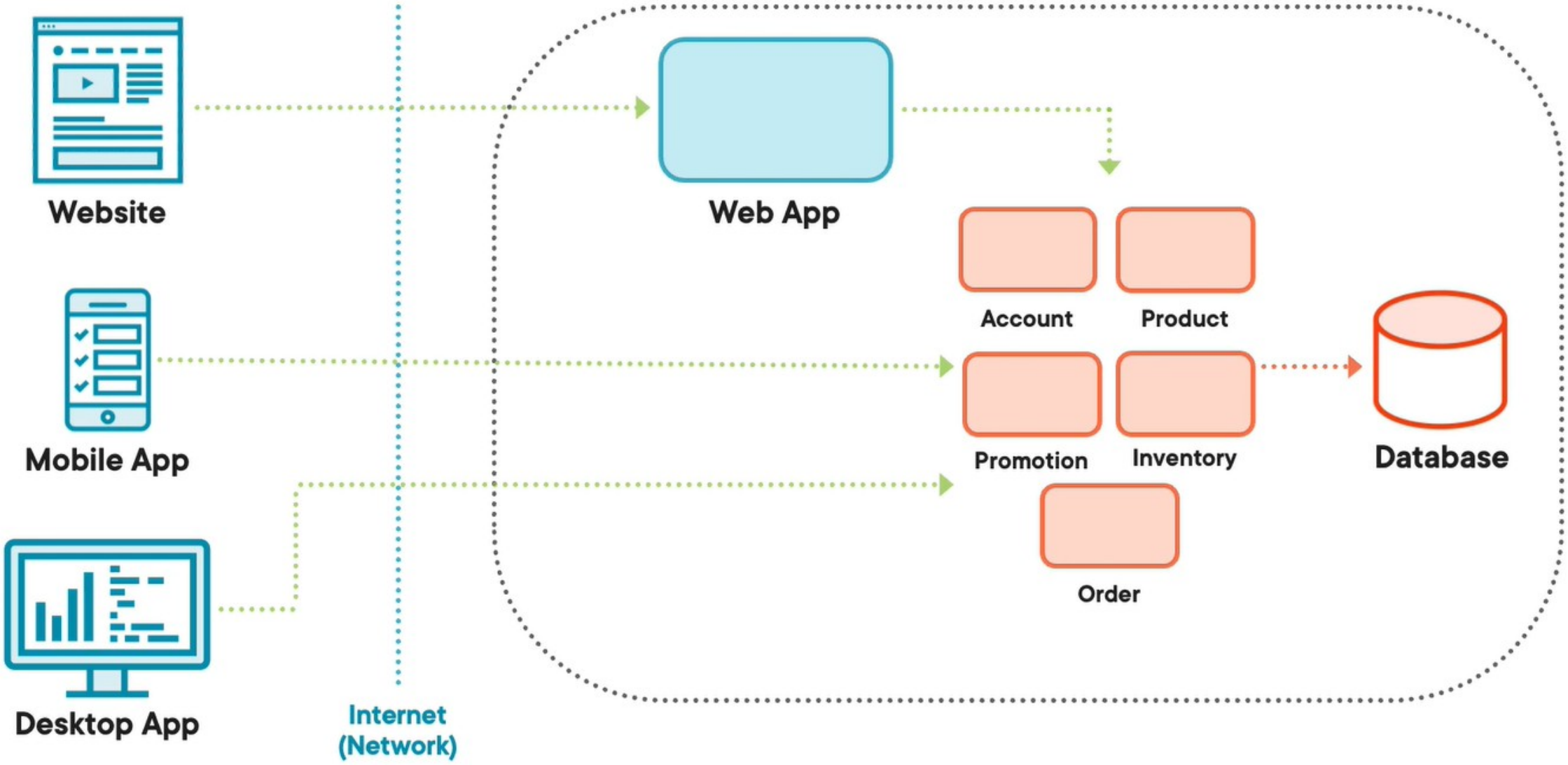
Strangler pattern



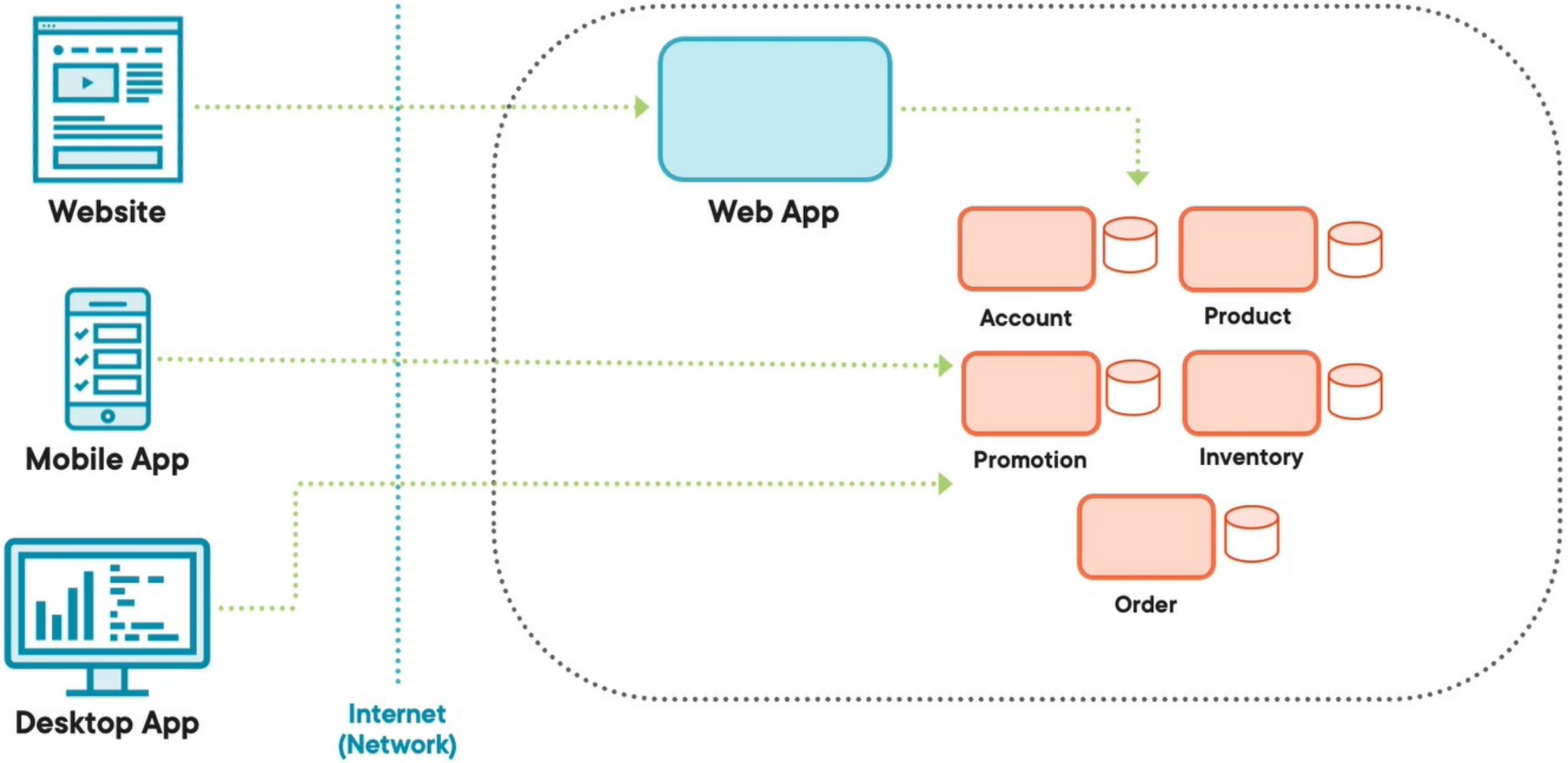
Monolith slowly reduced

(Proxy serves to select service end point and is ultimately eliminated once migration complete)

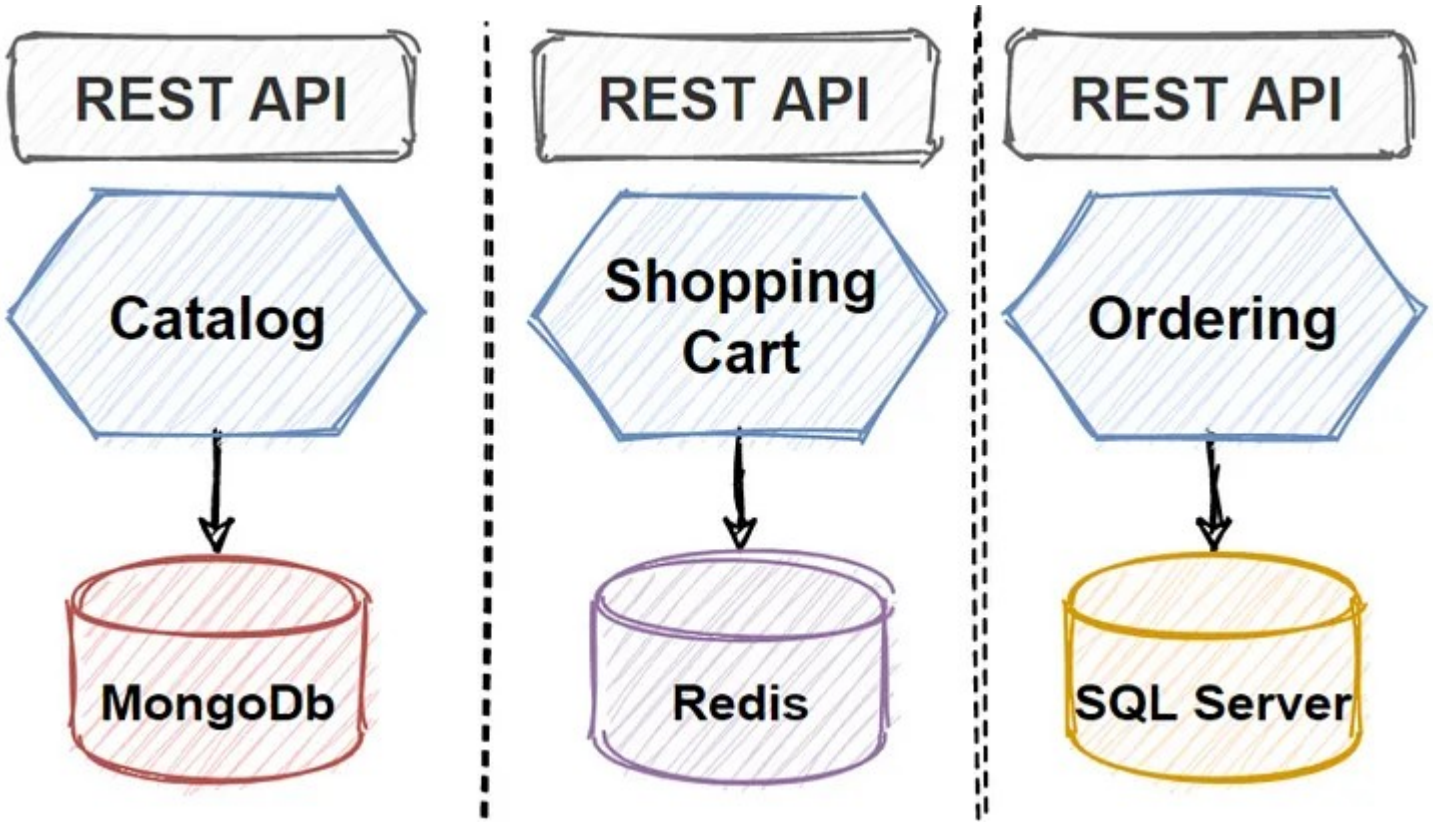
Split the service layer in microservices



Responsible for their own data



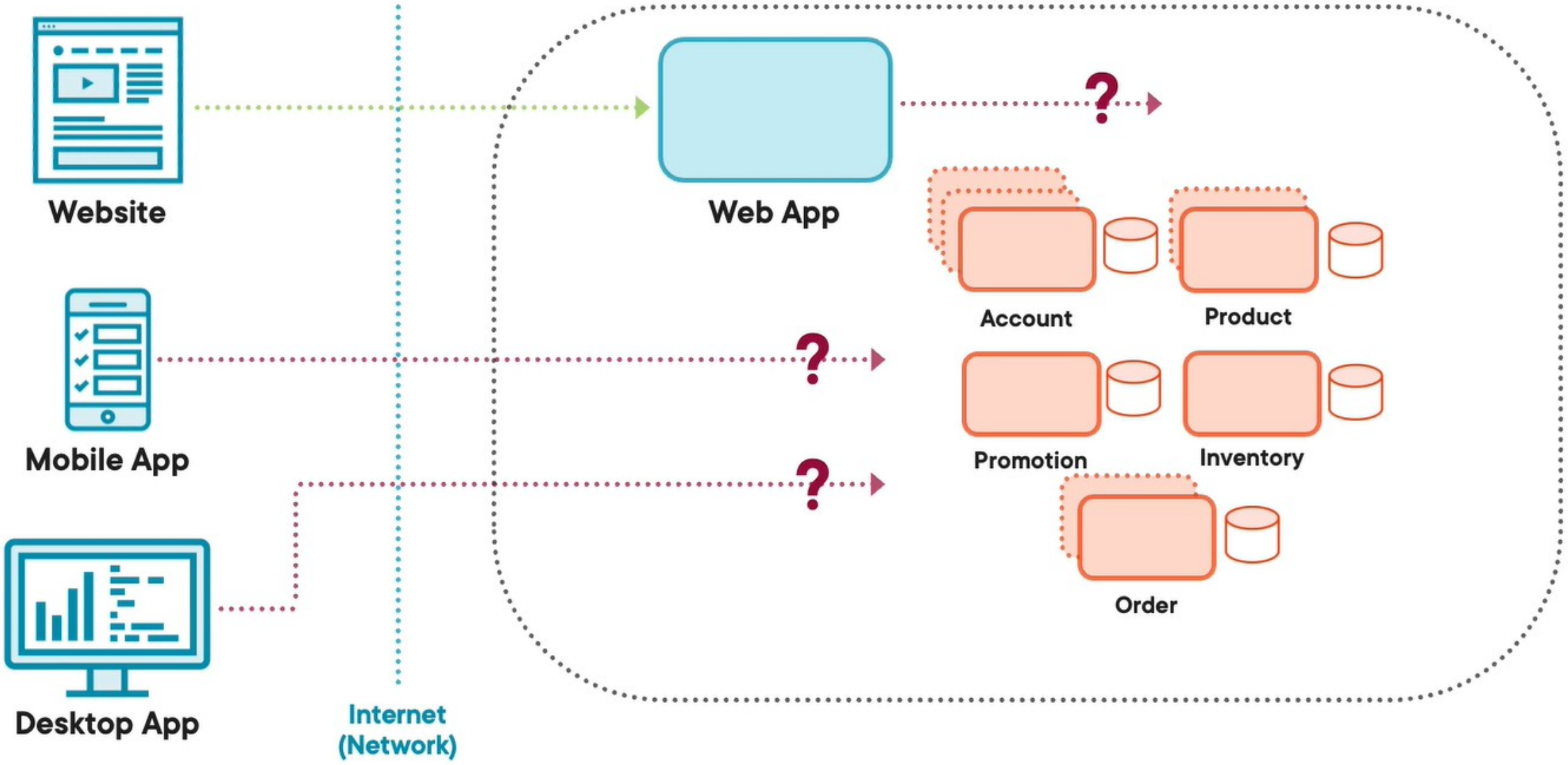
Database per microservice pattern



For example, if you are using a relational database, you can use three specific options:

- **Private-tables-per-service:** Each service has an exclusive set of tables not accessible to other services.
- **Schema-per-service:** Every service has a specific database schema that is not accessible to other services.
- **Database-server-per-service:** Each service has its own database server

Challenges in a microservices architecture



Communication between microservices

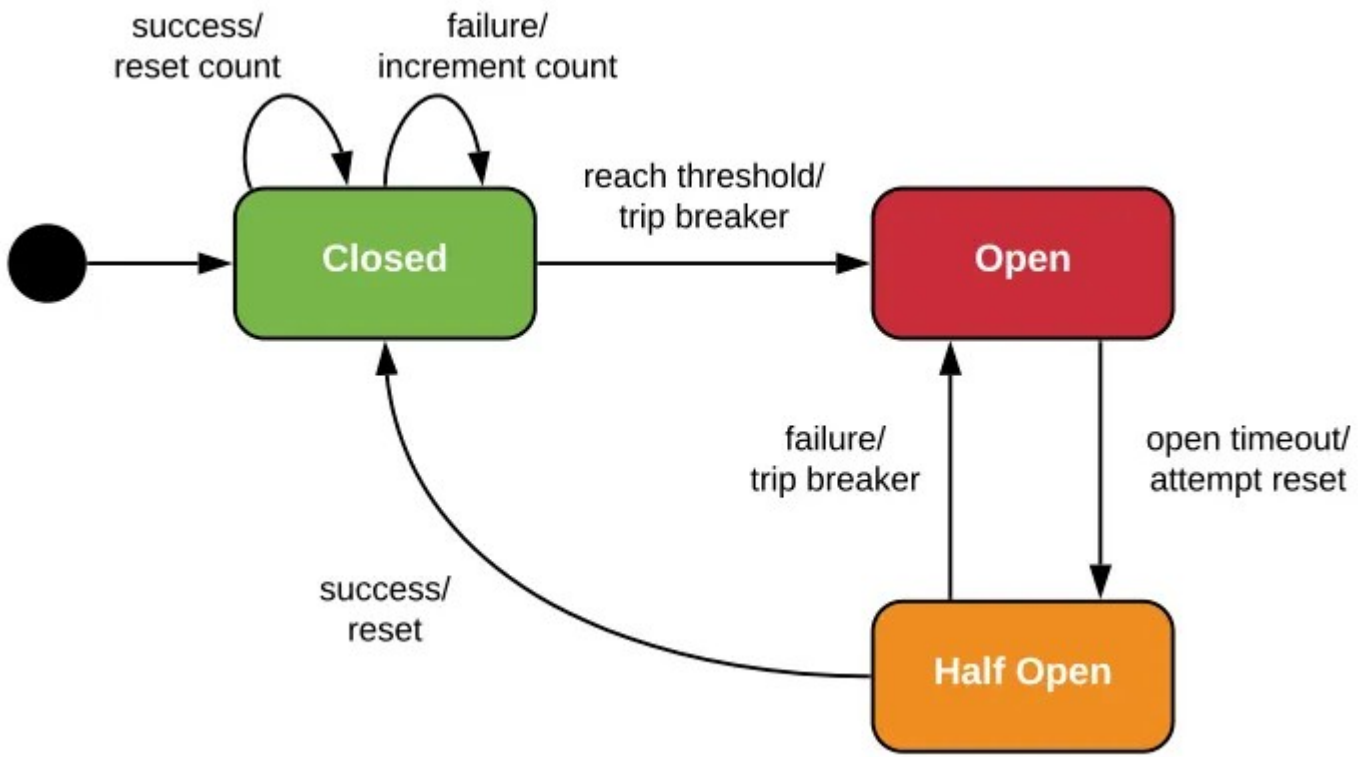
Communication between different services is carried out through lightweight communication mechanisms, such as:

- REST API requests
- or message-based protocols like AMQP (Advanced Message Queuing Protocol)

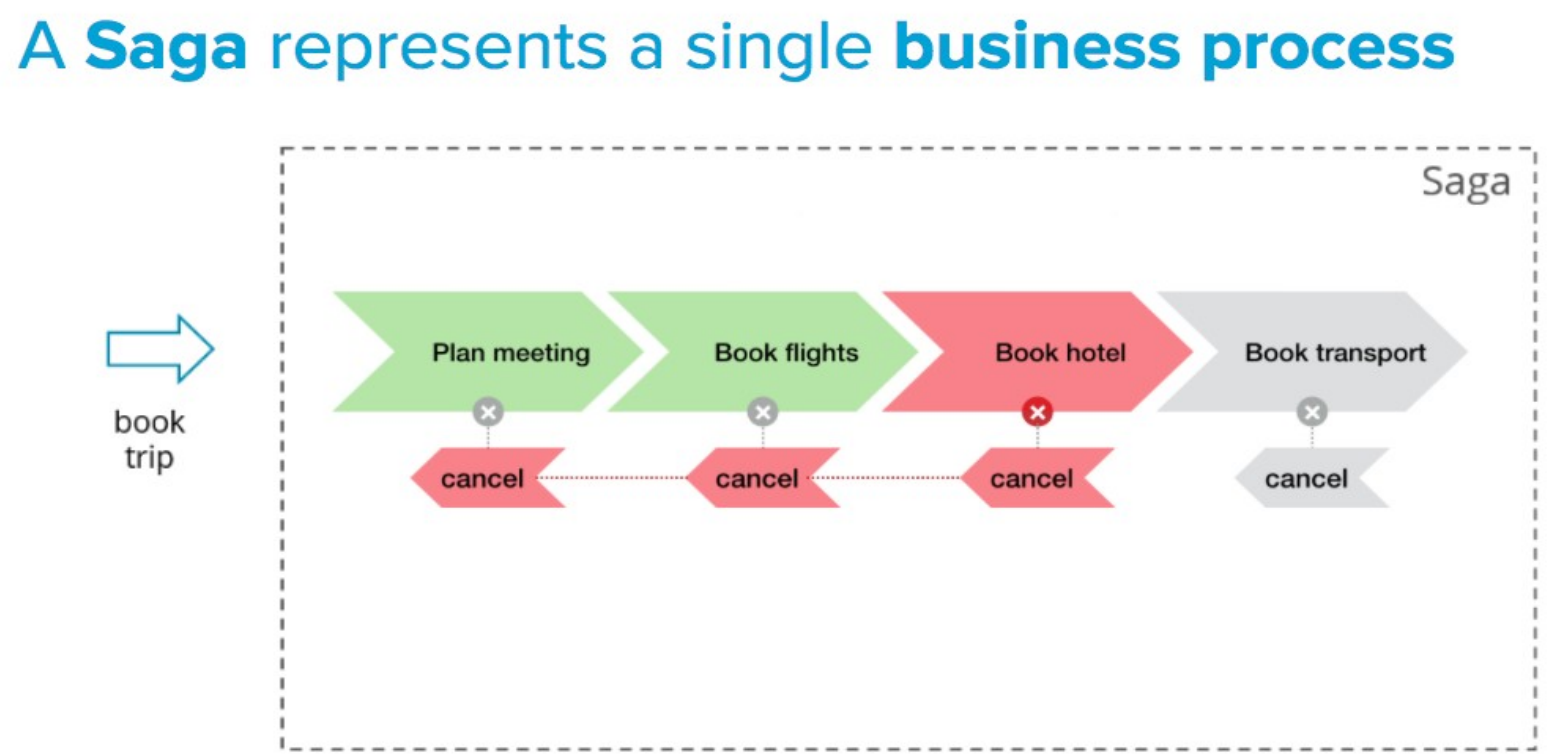
Synchronous communication: the service waits for a response from another service before continuing.

Asynchronous communication:, the a service sends a message to another service and then continues with other tasks without waiting for a response.

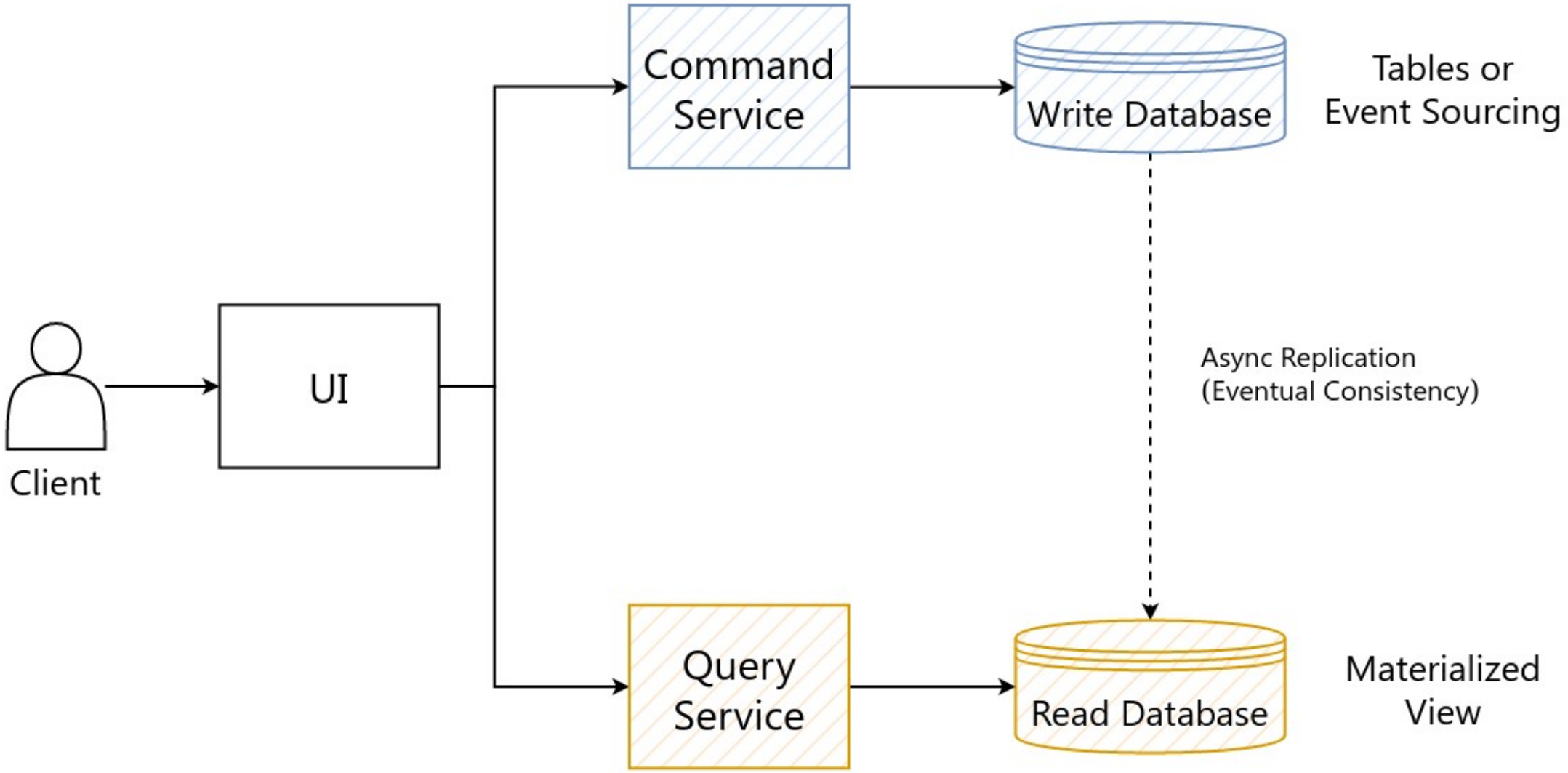
Circuit breaker pattern – Performance



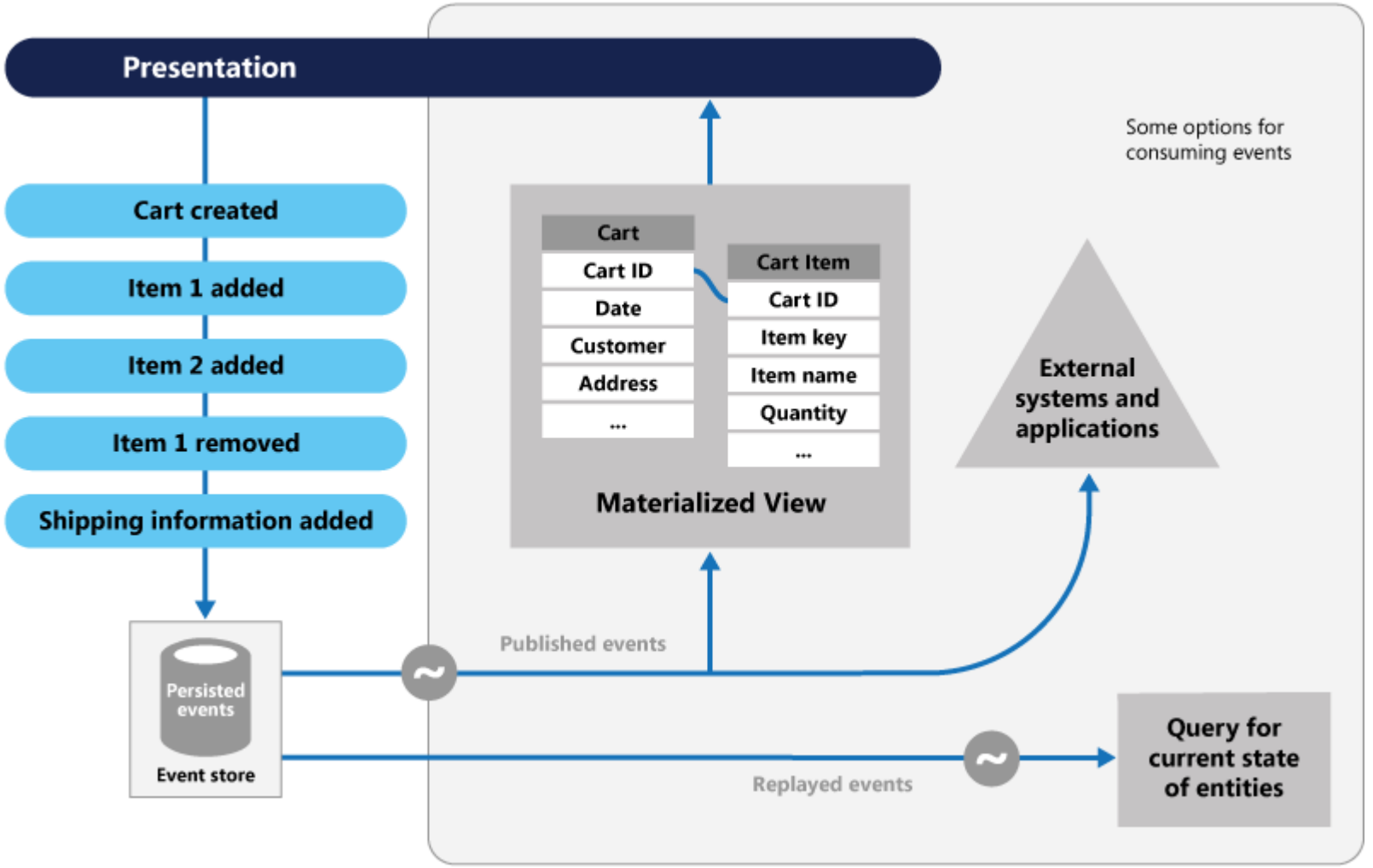
Saga pattern – Transactions



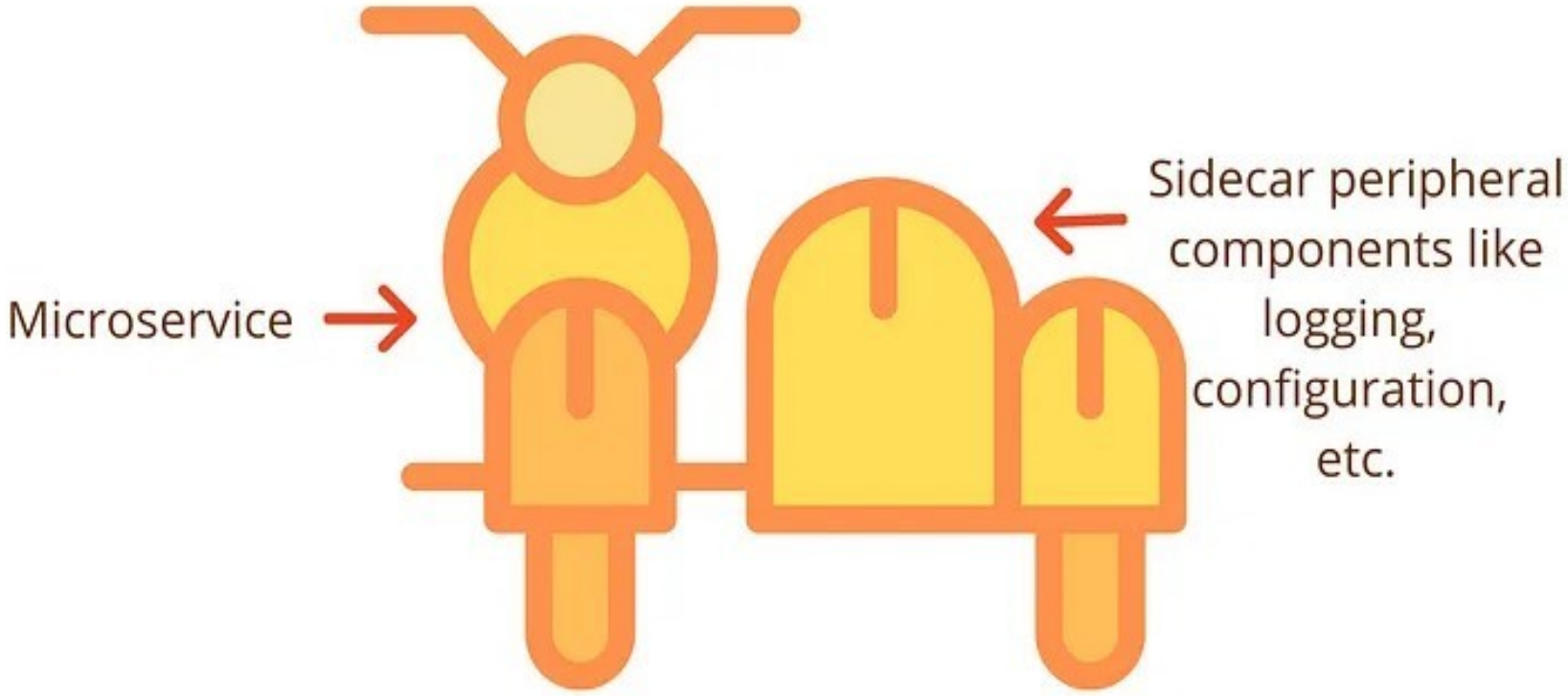
Command Query Responsibility Segregation pattern – Performance



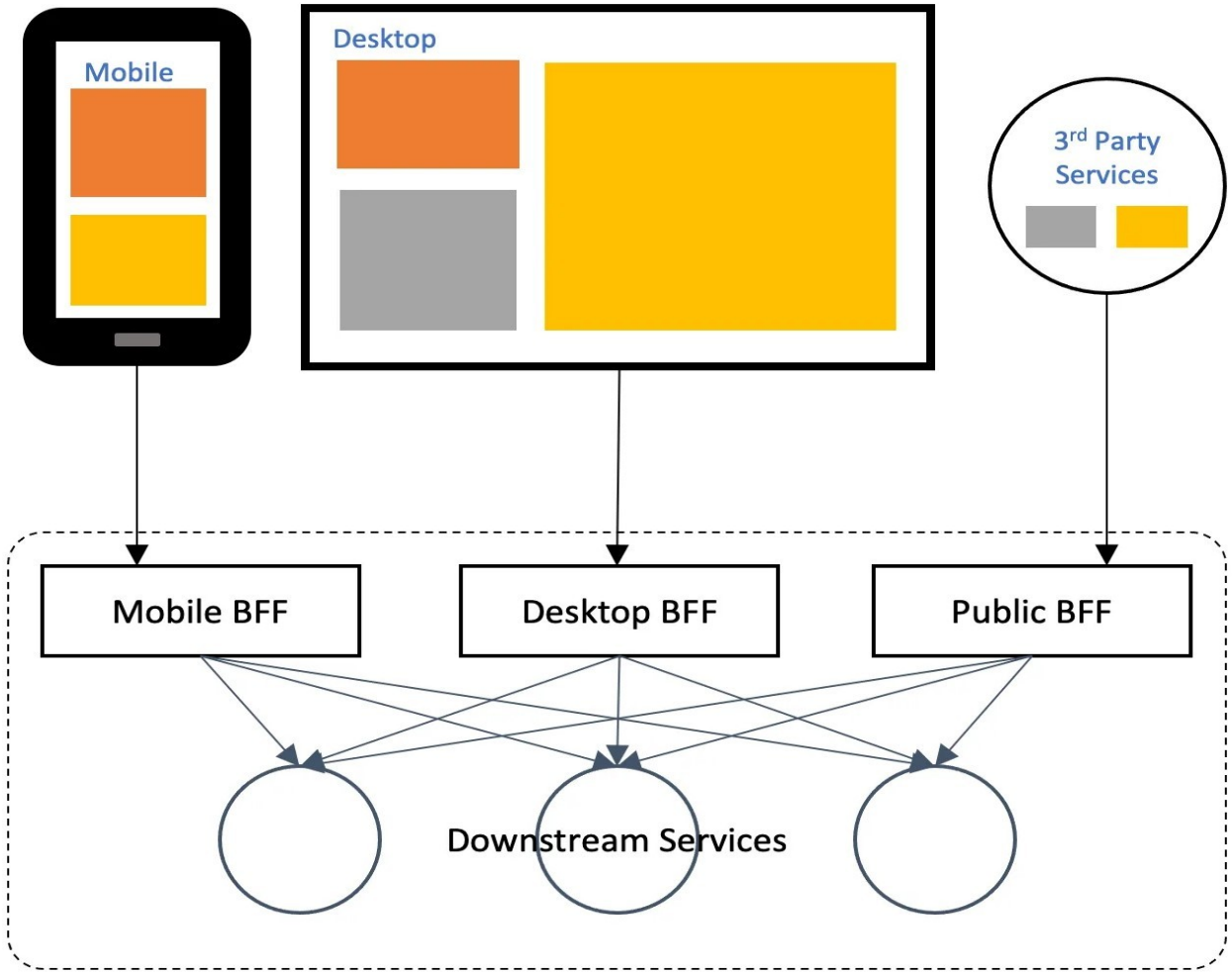
Event Sourcing pattern – Auditing



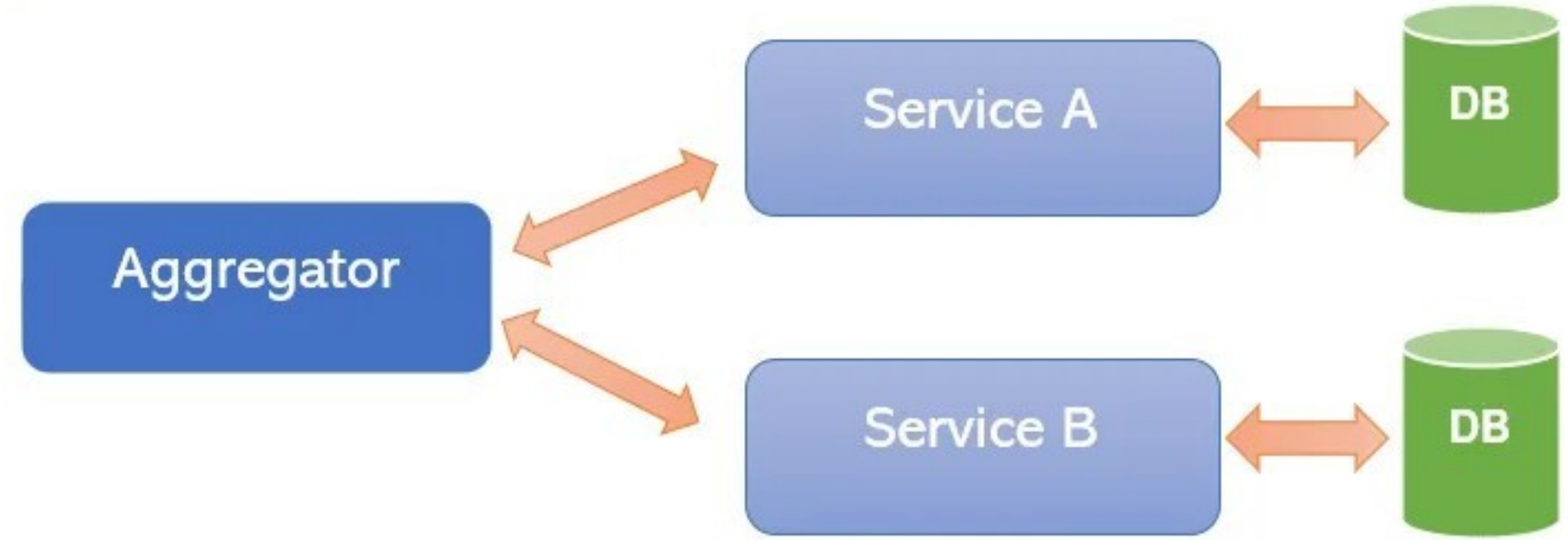
Sidecar pattern – Flexibility



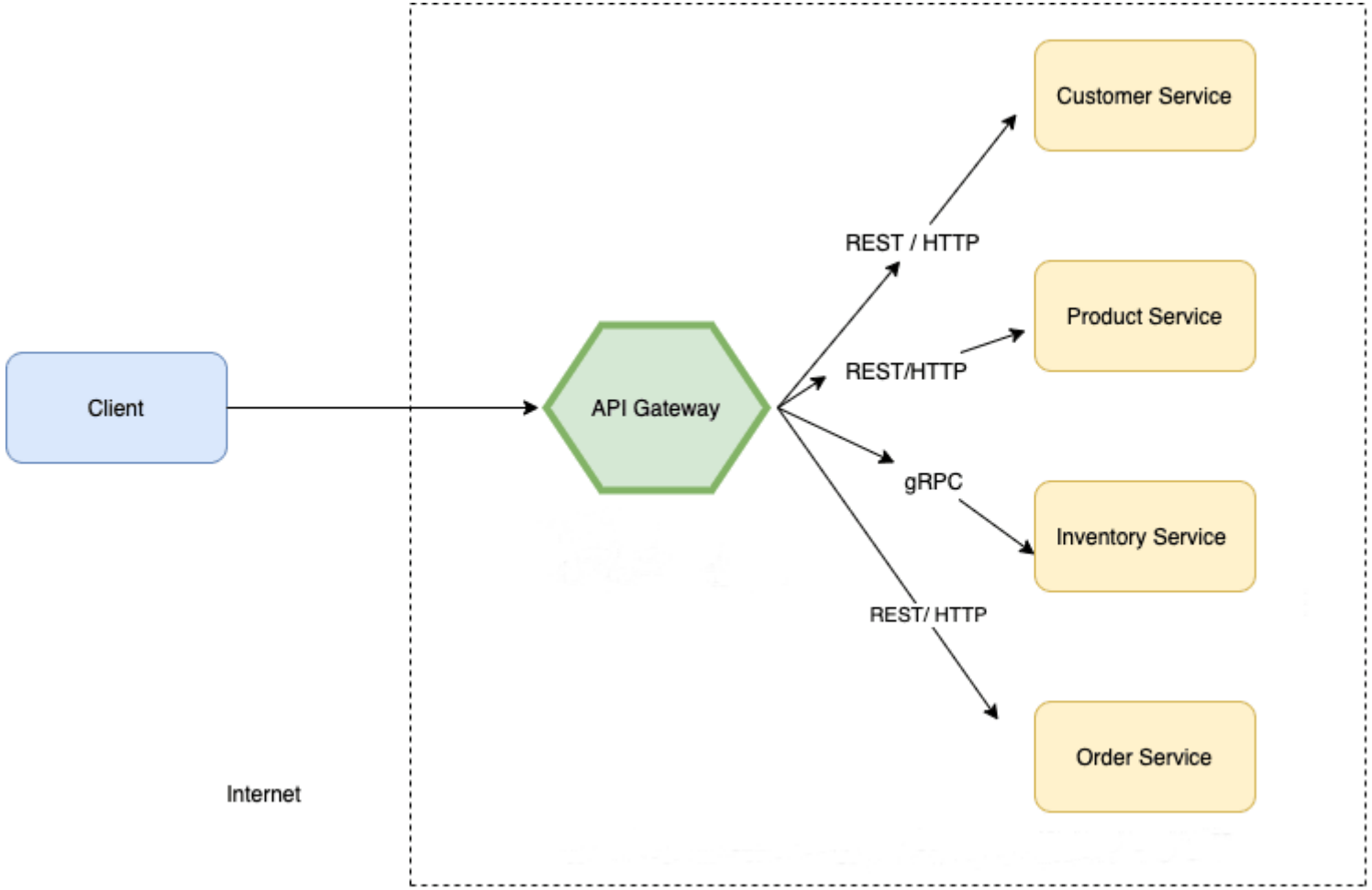
Backends for Frontends (BFF) pattern



Aggregator pattern



API gateway pattern



Microservices architecture