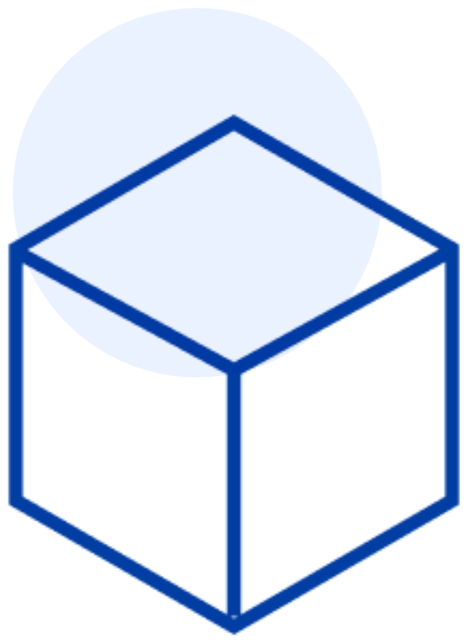
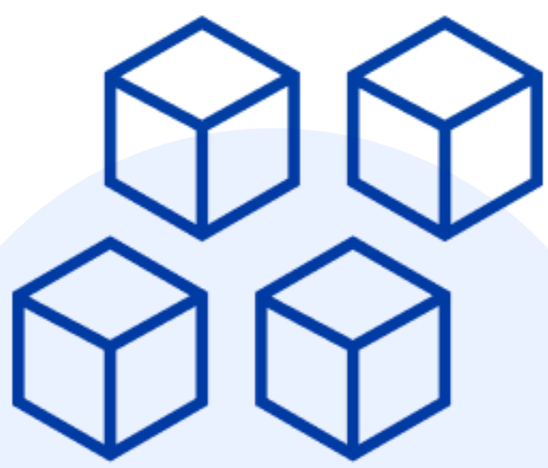
BOSSCODER ACADEMY

TOP MICROSERVICES PATTERNS

for System Design Interview



MONOLITHIC
Single unit



SOA
Coarse-grained

MICROSERVICES
Fine-grained

Microservices Architecture Overview

1 What are Microservices?

An architectural style that structures an application as a collection of small, loosely coupled, and independently deployable services.

2 Key Concepts

Independence:

Each service has specific business function. Developed & scaled separately.

Modularity:

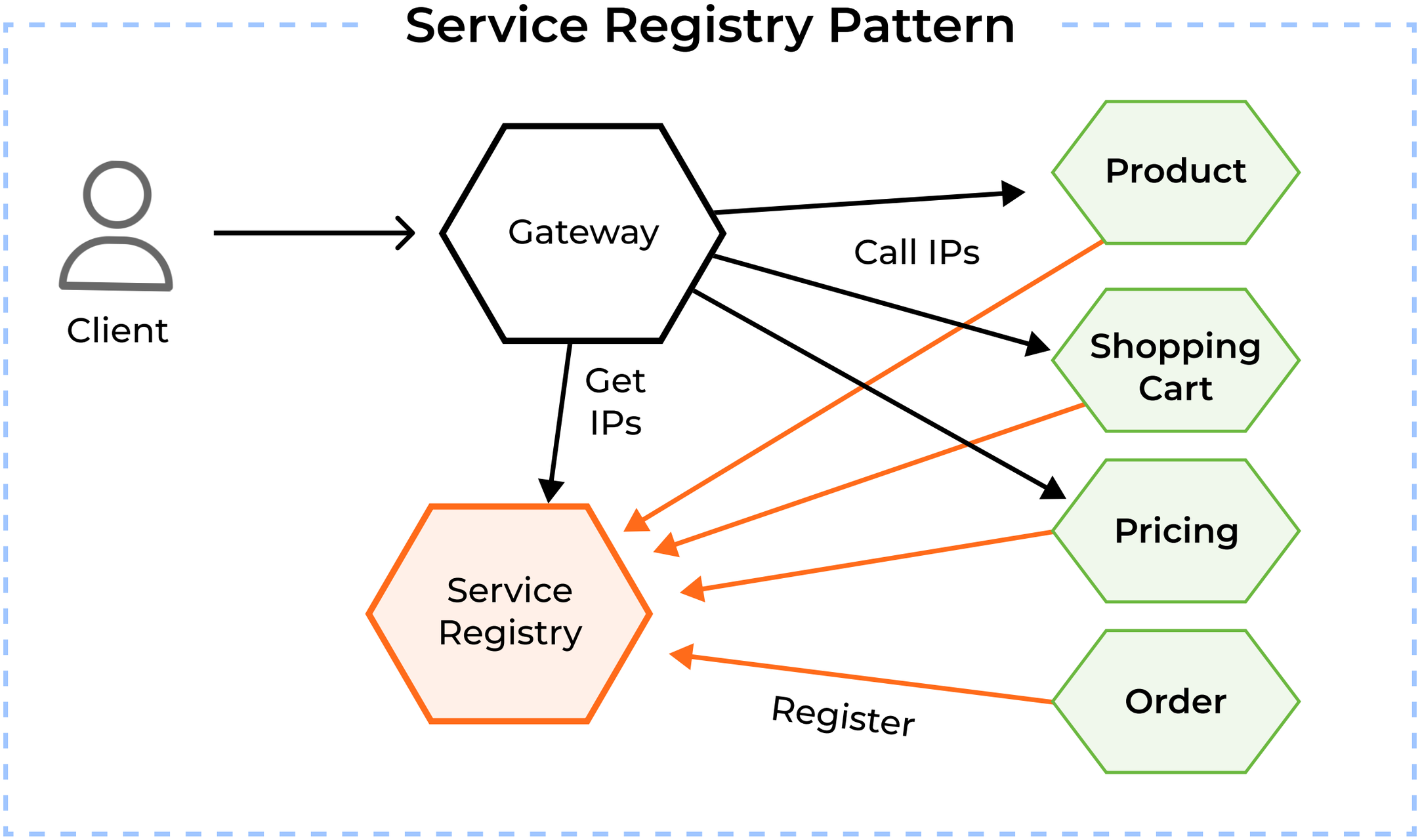
Breaking down a large, monolithic application into smaller, manageable pieces.

Why Companies like Amazon, eBay, Netlix, Spotify, etc. use

Microservices Architecture?

Flexibility	Independent development and scaling of services.
Scalability	Granular scaling of individual services.
Maintainability	Easier updates and maintenance.
Interoperability	Services communicate via well-defined APIs and protocols.

Service Registry Pattern in Microservices



#1 Definition:

CQRS Pattern: A microservices design pattern that separates command (write) and query (read) operations into distinct models with separate databases.

#2 Purpose:

Helps Microservices find and talk to each other without central control.

#3 Key Components:

- **Central Service Registry:** A list of all services and where to find them.
- **Microservices Registration:** Services register themselves with the registry, providing their names and endpoints.

#4 Example Use Case:

Large e-commerce website with various Microservices (e.g., order, payment, shipping, customer).

#5 Implementation:

Use tools like Consul or Eureka to make and manage the registry.

#6 Registration Example:

Order service registers as "order-service" with an endpoint like "http://order-service:8080".

#7 Dynamic Communication:

- Payment service can look up "order-service" in the registry to send payment info.
- Shipping service can find "order-service" to retrieve shipping details.

#8 Benefits:

- **Independent Development:** Services can be developed and deployed independently.
- **Flexibility:** No need to hard-code service endpoints; dynamic discovery.
- **Resilience:** Can adjust when things move or change, making it strong and flexible.

Circuit Breaker Pattern in Microservices

Think of it like a safety switch for microservices. If one part breaks down, the switch stops it from affecting everything else.

#1 Purpose:

Keeps apps running even when a service isn't working right.

#2 Key Components:

- **Circuit Breaker:** Acts as a protective barrier between client and service.
- **Monitoring:** Keeps an eye on the service to see if it's working.
- **State Control:** Can open the circuit to block requests upon detecting service failures.

#3 Example Scenario:

Microservice application relies on an unreliable external service.

#4 Functionality:

- **Detects Unreliability:** Monitors the service's health and identifies failures.
- **Circuit Open:** Prevents further requests from reaching the failing service.
- **Graceful Degradation:** Allows the application to switch to an alternative or fallback service.

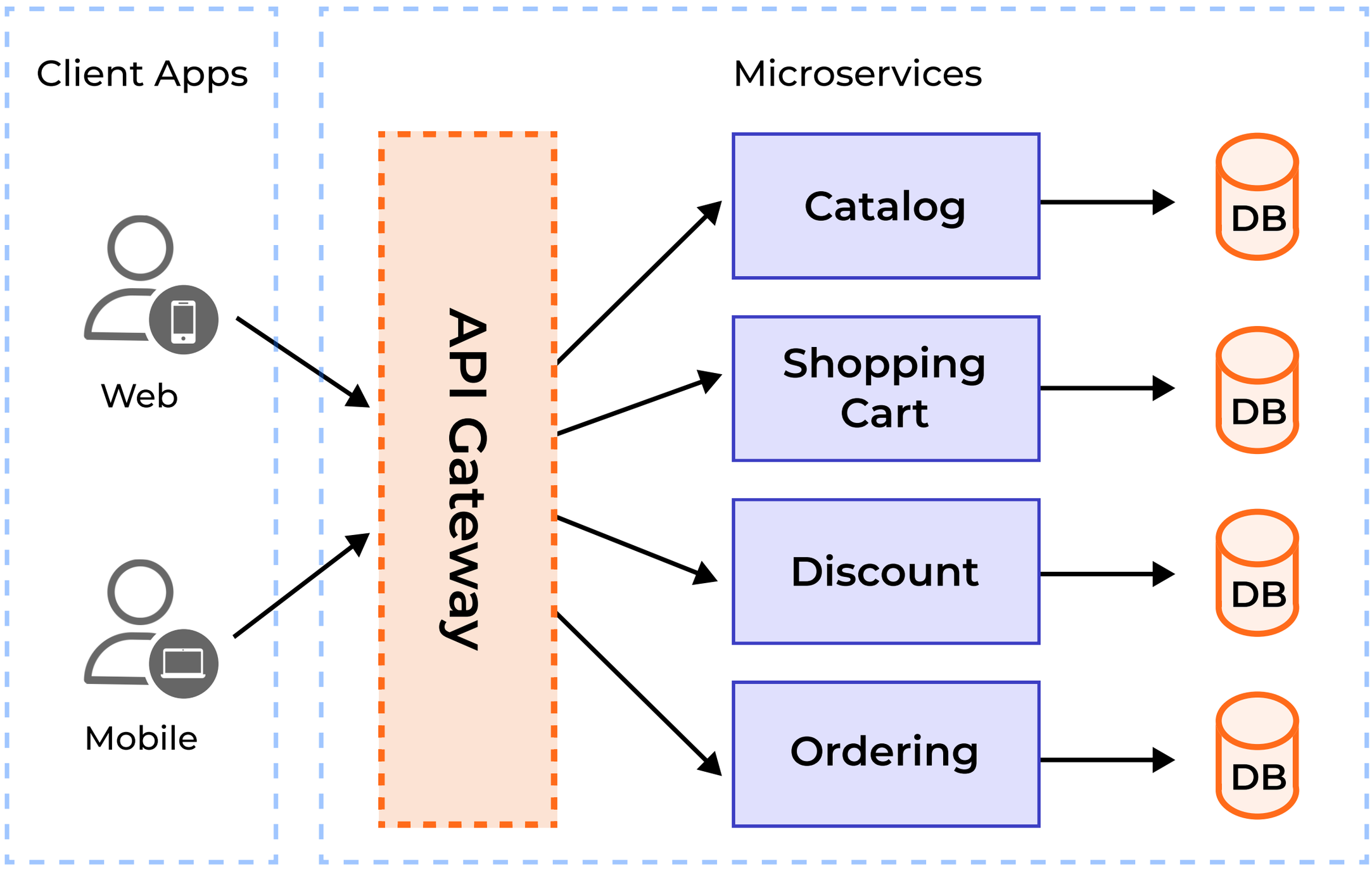
#5 Implementation:

Hystrix or Spring Cloud help manage this safety switch.

#6 Benefits:

- **Resilience:** Protects the application from service disruptions.
- **Continuity:** Enables the application to function despite service failures.
- **Controlled Response:** Can switch to another service smoothly.

API Gateway Pattern in Microservices



#1 Definition:

A fundamental microservices architecture pattern that employs an API gateway as a central entry point for all incoming API requests.

#2 Purpose:

Decouples clients from microservices, simplifying access and abstracting system complexity.

#3 Key Components:

- **API Gateway:** Serves as a proxy between clients and microservices.
- **Routing:** Directs requests to the appropriate microservice based on the endpoint.

#4 Main Functions:

- **Abstraction:** Presents a simplified and consistent API, eliminating the need to remember multiple microservice addresses.
- **Security & Governance:** Enforces access control, monitors system performance, and enforces policies.

#5 Example Scenario:

E-commerce system with microservices for order management, product catalog, and user authentication.

#6 Implementation:

- API Gateway acts as a reverse proxy, receiving all client requests.
- Routes requests based on endpoints (e.g., /orders, /products) to respective microservices.

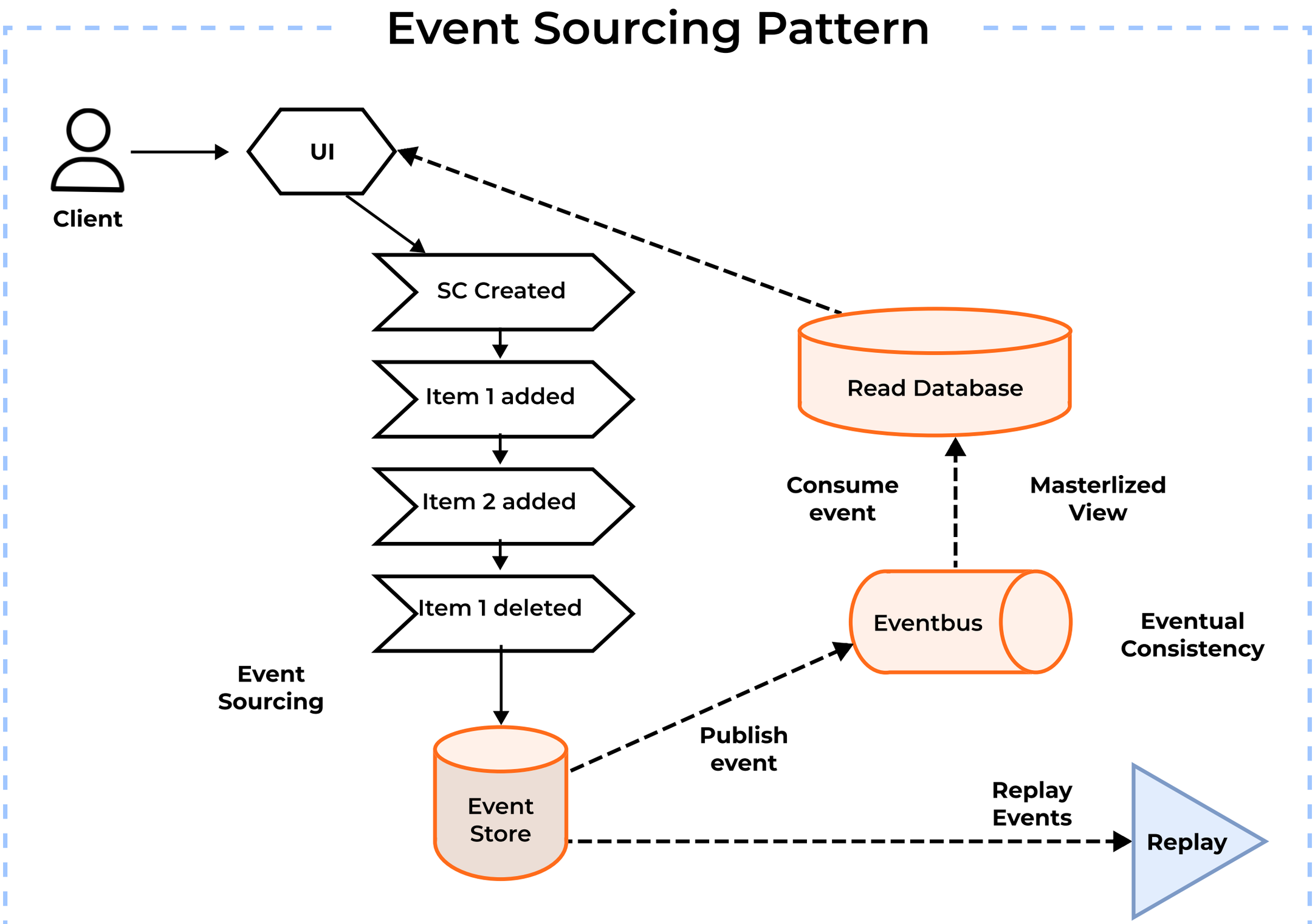
#7 Additional Functions:

- **Transformation:** Performs request/response transformation.
- **Rate Limiting:** Controls request rates.
- **Authentication & Authorization:** Ensures secure access.
- **Caching:** Improves performance with data caching.

#8 Benefits:

- **Simplification:** Provides a unified API for clients, hiding microservices' internal details.
- **Scalability:** Facilitates easy scaling and management of microservices.
- **Security:** Enhances security and control over service access.
- **Flexibility:** Streamlines development, deployment, and maintenance of microservices-based applications.`

Event Sourcing in Microservices



#1 Definition:

A microservices pattern for persisting and querying data by capturing all state changes as events, enabling reconstruction of application state.

#2 Core Concept:

Event Logging: Records every state change as an event, creating a historical log.

#3 Application Scenario:

E-commerce application example: Events like OrderPlaced, ShipmentMade, and OrderCanceled.

#4 Benefits:

- **Auditability:** Provides an auditable log of all system changes.
- **Scalability:** Allows parallel processing of events for improved scalability.
- **Flexibility:** Enables changes in data querying and persistence without altering data.
- **Fault-tolerance:** Immutable events prevent data modification, ensuring data integrity.

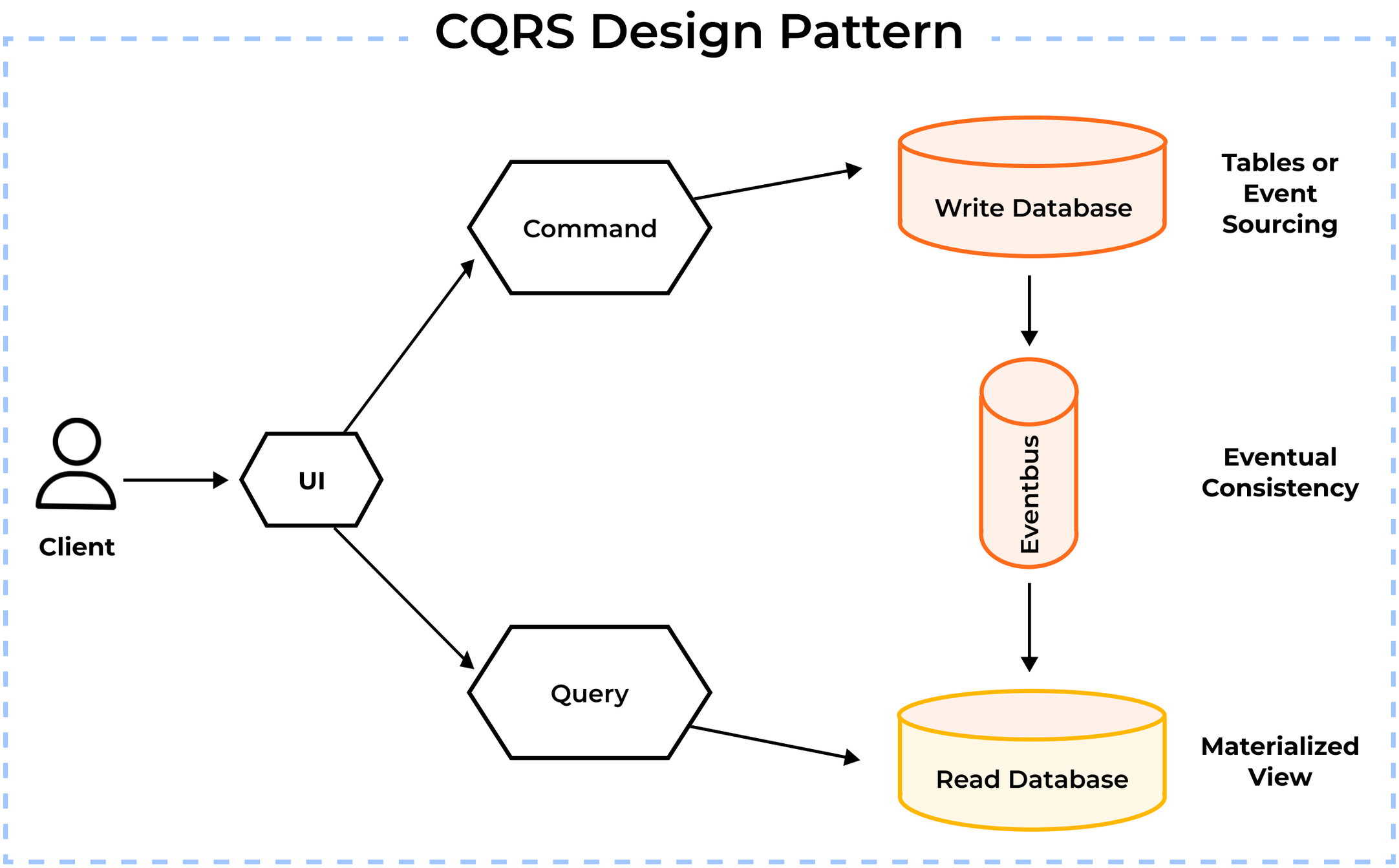
#5 Considerations:

- **Complexity:** Implementing Event Sourcing requires careful planning and design.
- **Querying Overhead:** Querying data may be slower due to event replay; use judiciously.

#6 Caution:

- Evaluate if Event Sourcing is the right fit for your application, as alternative solutions may be more suitable in some cases.

Command Query Responsibility Segregation (CQRS) Pattern in Microservices



#1 Definition:

CQRS Pattern: A microservices design pattern that separates command (write) and query (read) operations into distinct models with separate databases.

#2 Core Concept:

Distinct Models: Recognizes that models for writing data differ from models for reading data.

#3 Implementation:

- **Command Model:** Receives commands, writes to the database.
- **Query Model:** Reads from the database, sends data to clients.

#4 Benefits:

- **Performance:** Optimizes each model for its specific task, enhancing system performance.
- **Scalability:** Allows independent optimization and scaling of command and query models.
- **Simplified Code:** Separation of concerns simplifies the codebase.

#5 Example Scenario:

E-commerce application with separate models for writing and reading product information.

#6 Optimization:

- **Command Model:** Write-optimized, stores data in a write-optimized database.
- **Query Model:** Read-optimized, stores data in a read-optimized database.

#7 Communication:

Models communicate through an event bus or message queue.

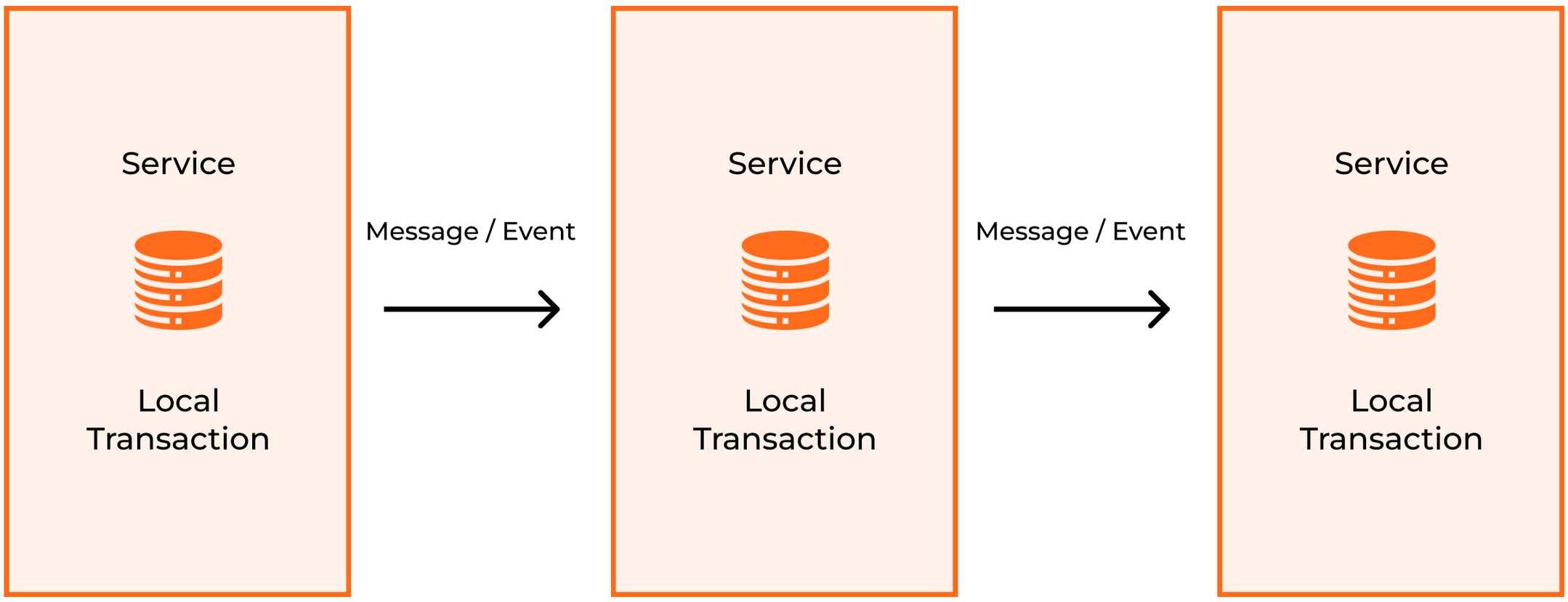
#8 Considerations:

- **Complexity:** May introduce complexity with separate models and databases.
- **Development Effort:** Requires additional development effort to maintain separate models.`

#9 Overall Impact:

CQRS enhances system scalability, performance, and maintainability by isolating command and query operations, but it should be carefully considered for suitability in each application.

SAGA Pattern in Distributed Architecture



#1 Definition:

A solution for maintaining data consistency in a distributed architecture without relying on ACID principles. It allows for committing multiple compensatory transactions.

#2 Two Approaches:

- **Choreography:** No central orchestration; services execute transactions, publish events, and respond to events from other services.
- **Orchestration:** Each service performs transactions, publishes events, and other services respond to these events to complete their tasks.

#3 Advantage:

Data Consistency: Ensures data consistency across multiple services without creating tight coupling.

#4 Disadvantage:

- **Complexity:** High complexity in designing and implementing sagas compared to traditional transactions.
- **Developer Familiarity:** Developers may not be well-versed in writing sagas as they would with traditional transactions.

Bulkhead Pattern

#1 Definition:

A fault tolerance design that isolates elements of an application into partitions, preventing failure in one part from affecting others, similar to ship bulkheads.

#2 Context and Problem:

- In a cloud-based application, excessive load or service failure can impact all consumers.
- Resource exhaustion may affect consumers and services, causing cascading failures.

#3 Solution:

- Partition service instances based on consumer load and availability requirements.
- Partition consumer resources to prevent resource exhaustion.

#4 Benefits:

- Limits failure spread.
- Maintains some function during issues.
- Offers varied service quality.

#5 Diagrams:

- Bulkheads structured around connection pools calling services.
- Multiple clients calling a single service with isolated instances.

#6 Issues and Considerations:

- Define partitions by business/tech needs.
- Weigh isolation, cost, performance, manageability.
- Use retry, circuit breaker, throttling pattern.
- Consider VMs, containers, processes for deployment.

#7 Use Cases:

- Separate resources for backend services.
- Isolate critical from standard consumers.
- Protect against cascading failures.

#8 Limitations:

- May be less resource-efficient.
- Added complexity might not be always needed.

BOSSCODER ACADEMY

WHY BOSSCODER?

1000+ Alumni placed at Top Product-based companies.

More than **136% hike** for every **2 out of 3** working professional.

Average package of **24LPA.**

The syllabus is most **up-to-date** and the list of problems provided covers **all important topics.**

Lavanya
Meta

Course is very well **structured** and **streamlined** to crack any **MAANG** company

Rahul .
Google

EXPLORE MORE