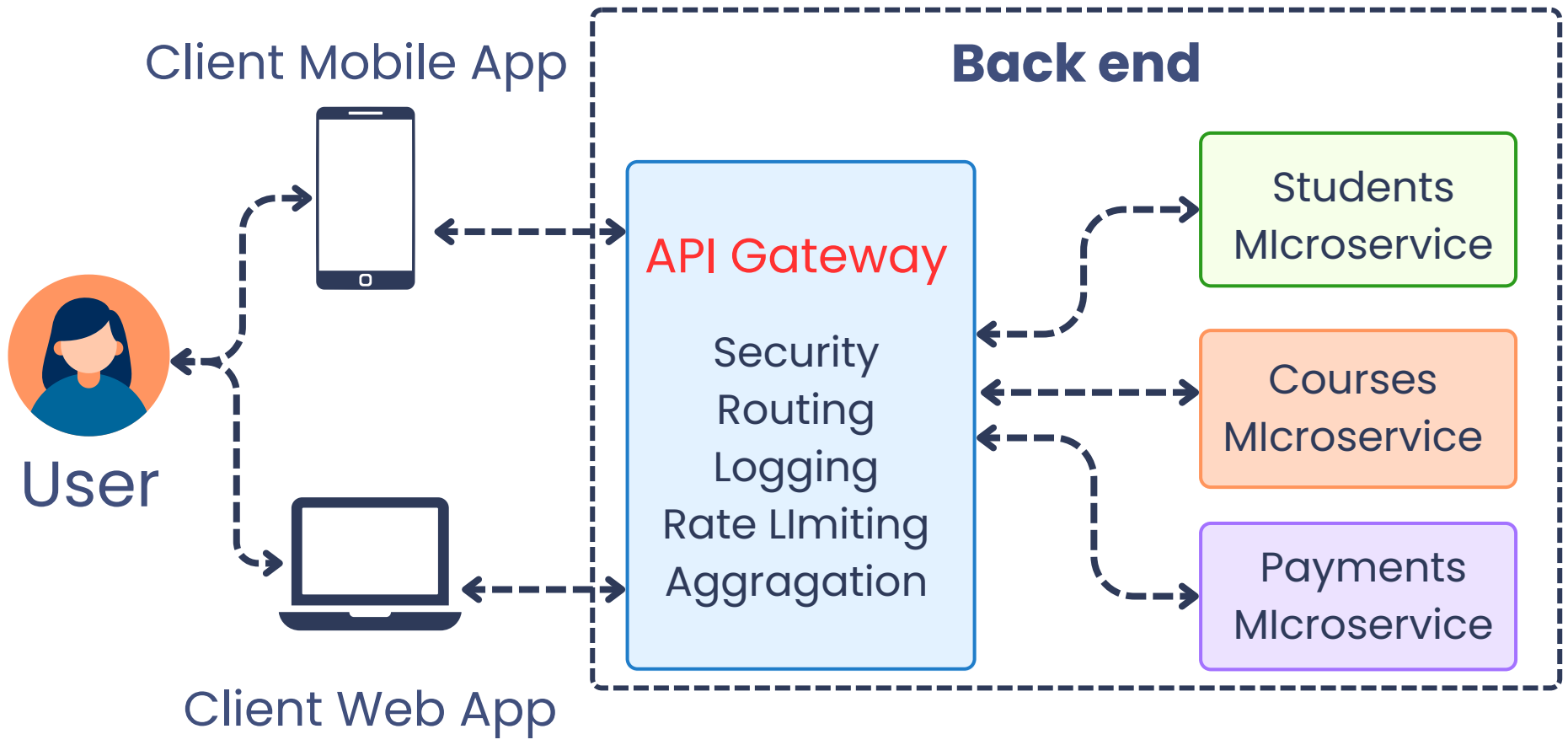
Sathish Kumar Subramani

14 DESIGN PATTERNS FOR SCALABLE MICROSERVICES

Explore proven system design patterns for building scalable, reliable, and maintainable microservices architectures with ease.

Sathish Kumar Subramani

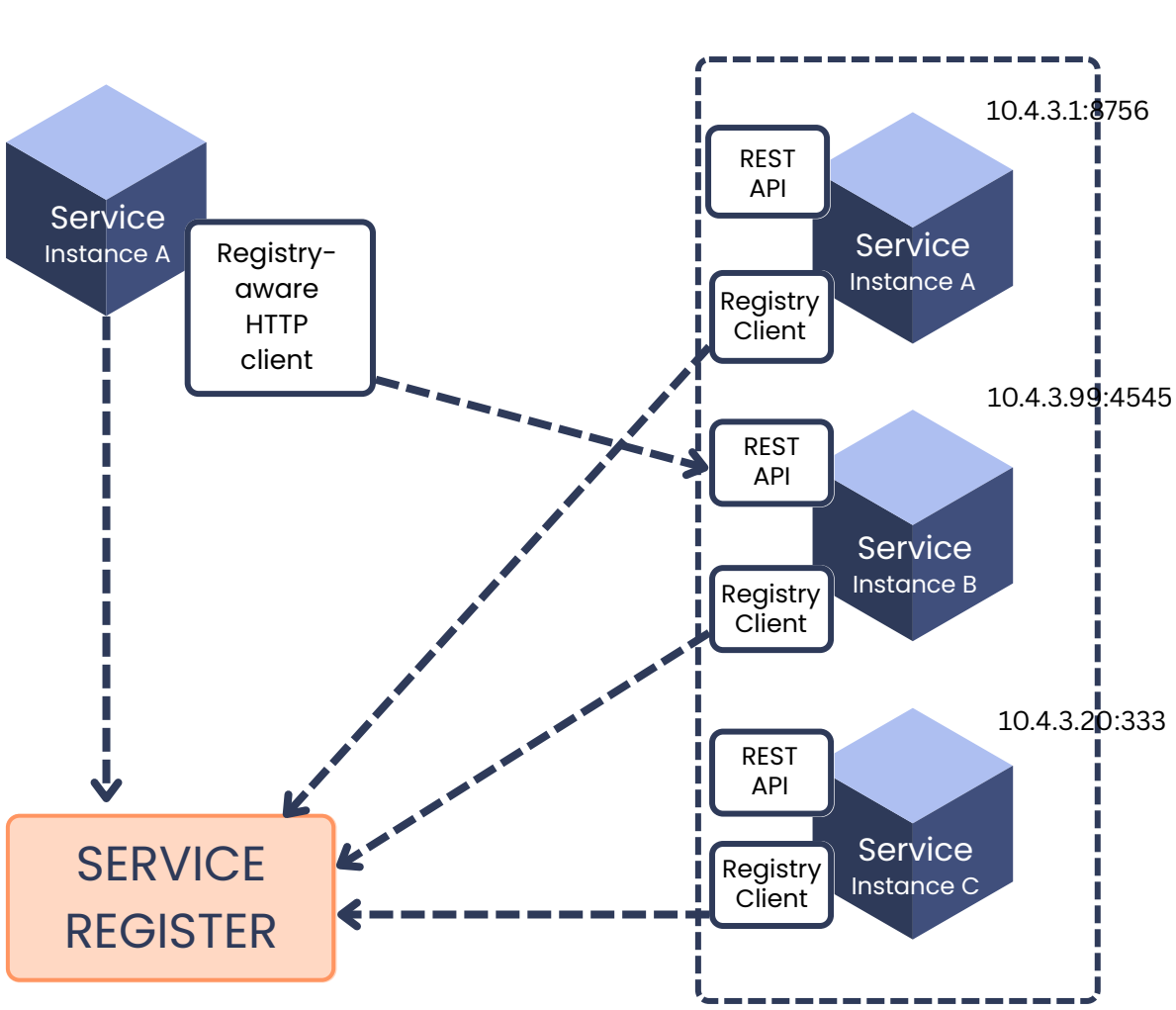
Gateway Pattern



An API Gateway manages client requests by handling routing, authentication, logging, and load balancing across multiple backend microservices efficiently and centrally.

Sathish Kumar Subramani

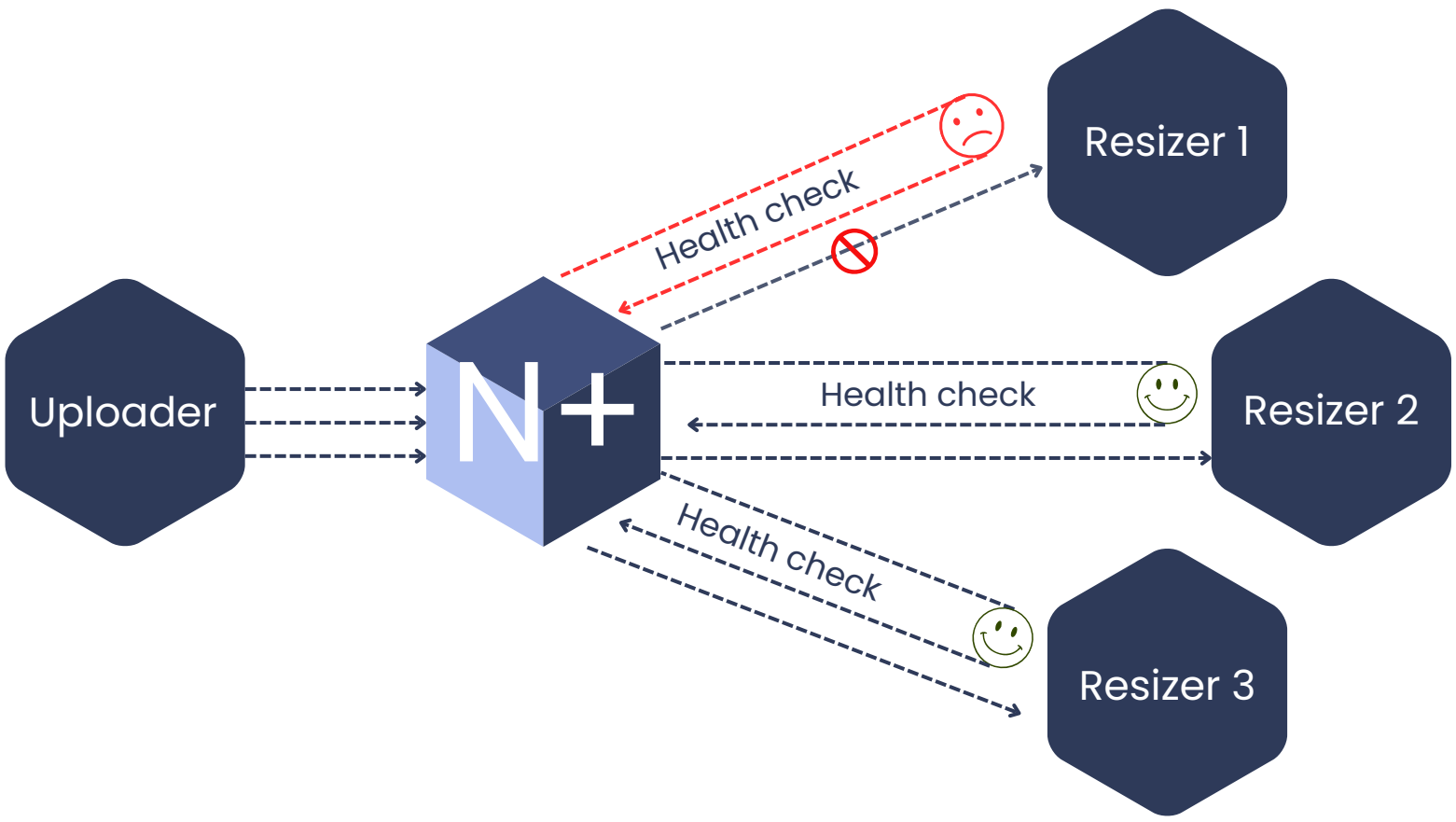
Service Registry Pattern



A service registry allows microservices to register and discover each other dynamically, enabling flexible, scalable communication without hardcoded service addresses.

Sathish Kumar Subramani

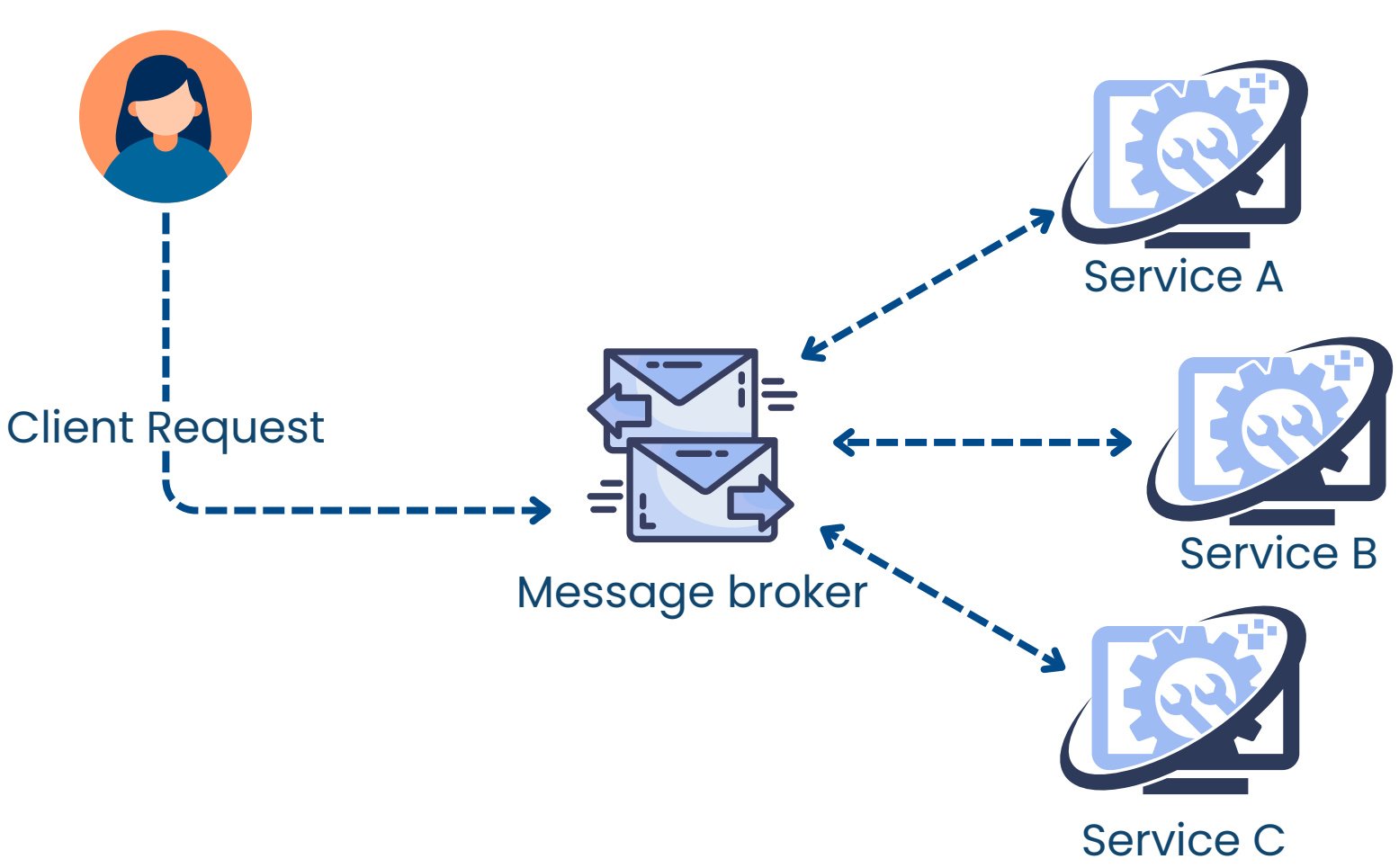
Circuit Breaker Pattern



Circuit breakers detect failing services and temporarily block requests to them, preventing cascading system failures and enabling fallback logic to maintain resilience.

Sathish Kumar Subramani

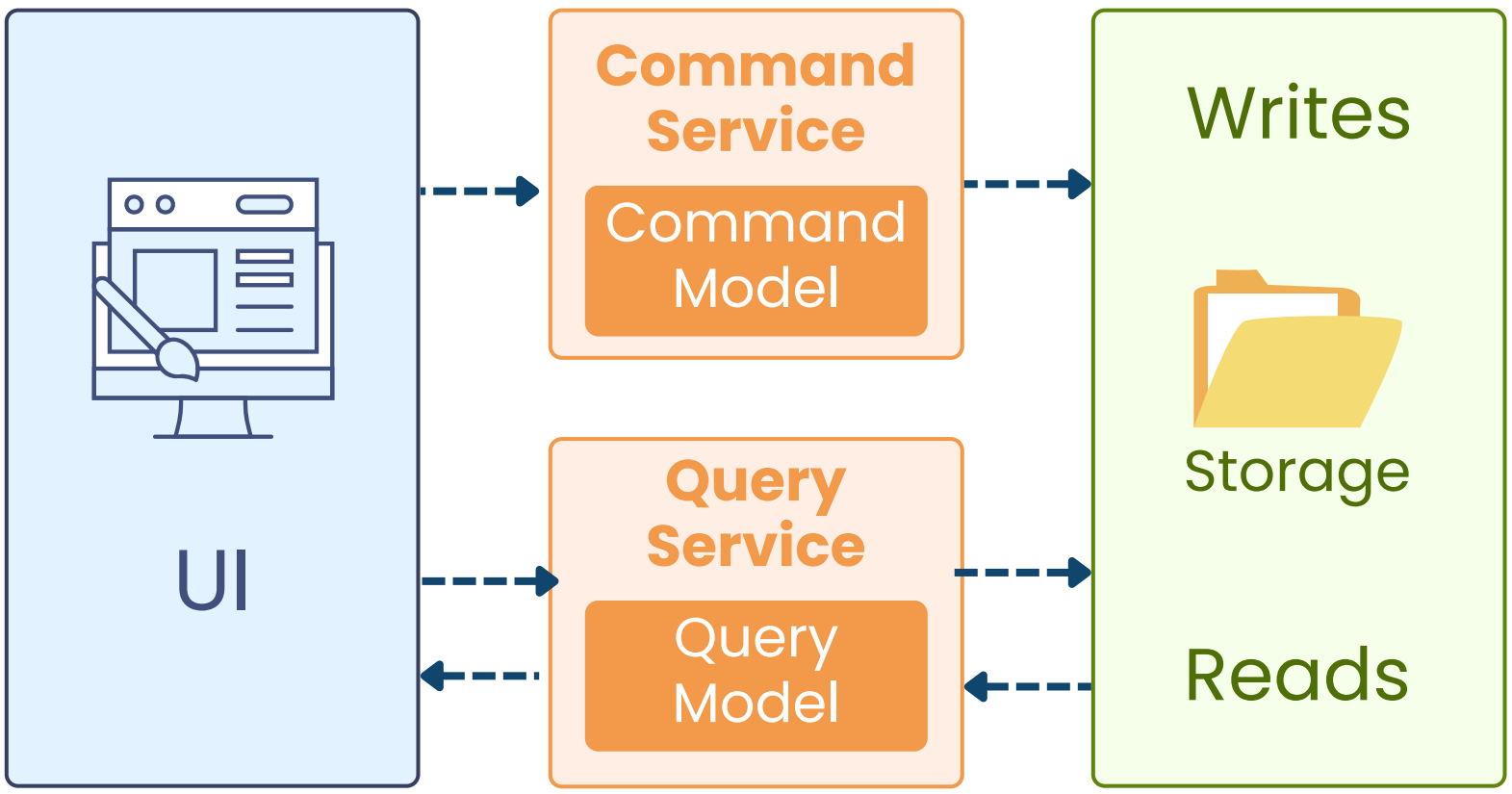
Saga Pattern



The Saga pattern divides long-running business transactions into smaller coordinated steps across services, ensuring consistency in distributed systems without locking resources.

Sathish Kumar Subramani

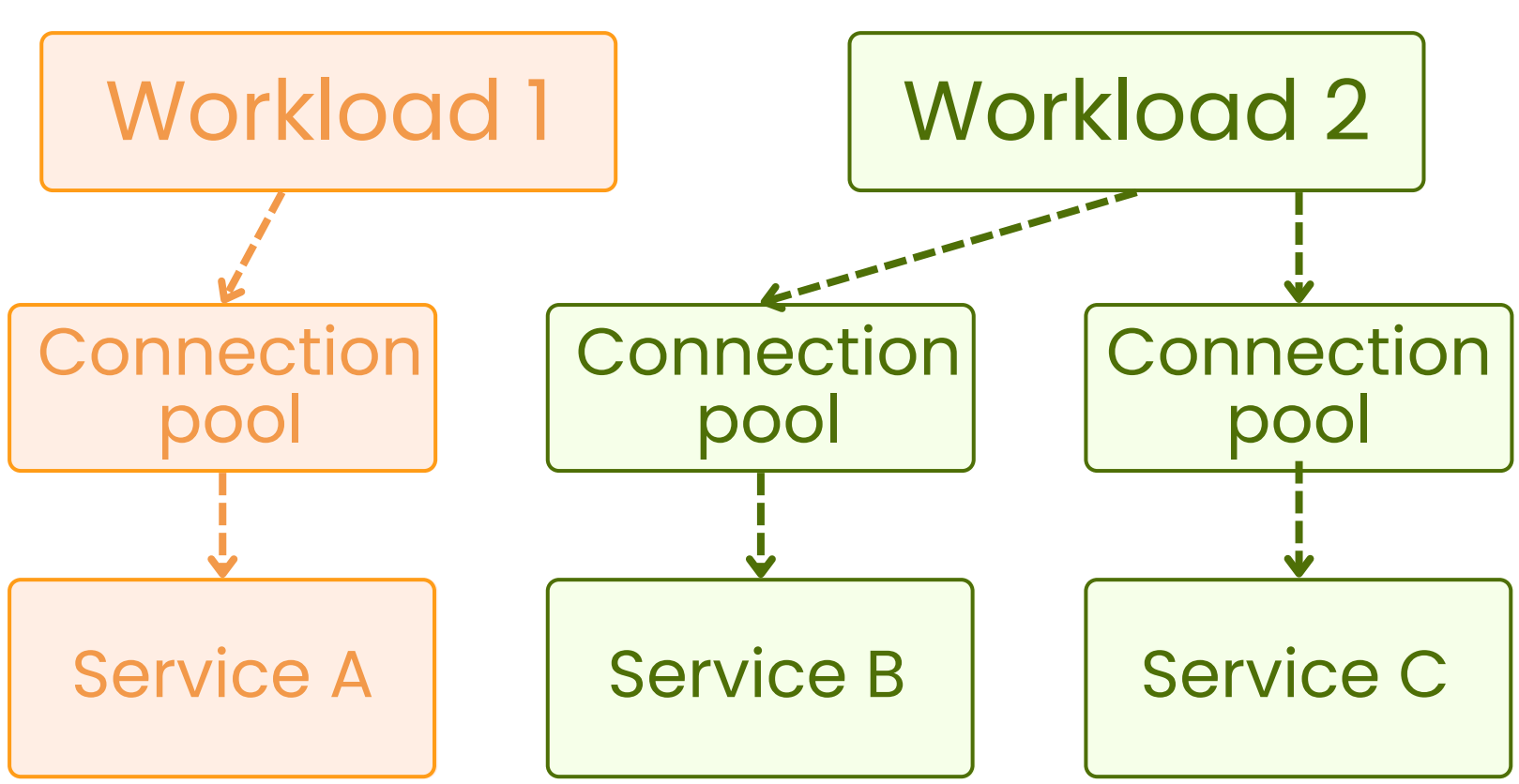
CQRS Pattern



The CQRS pattern separates the system's read and write operations into different models for better scalability, maintainability, and system performance.

Sathish Kumar Subramani

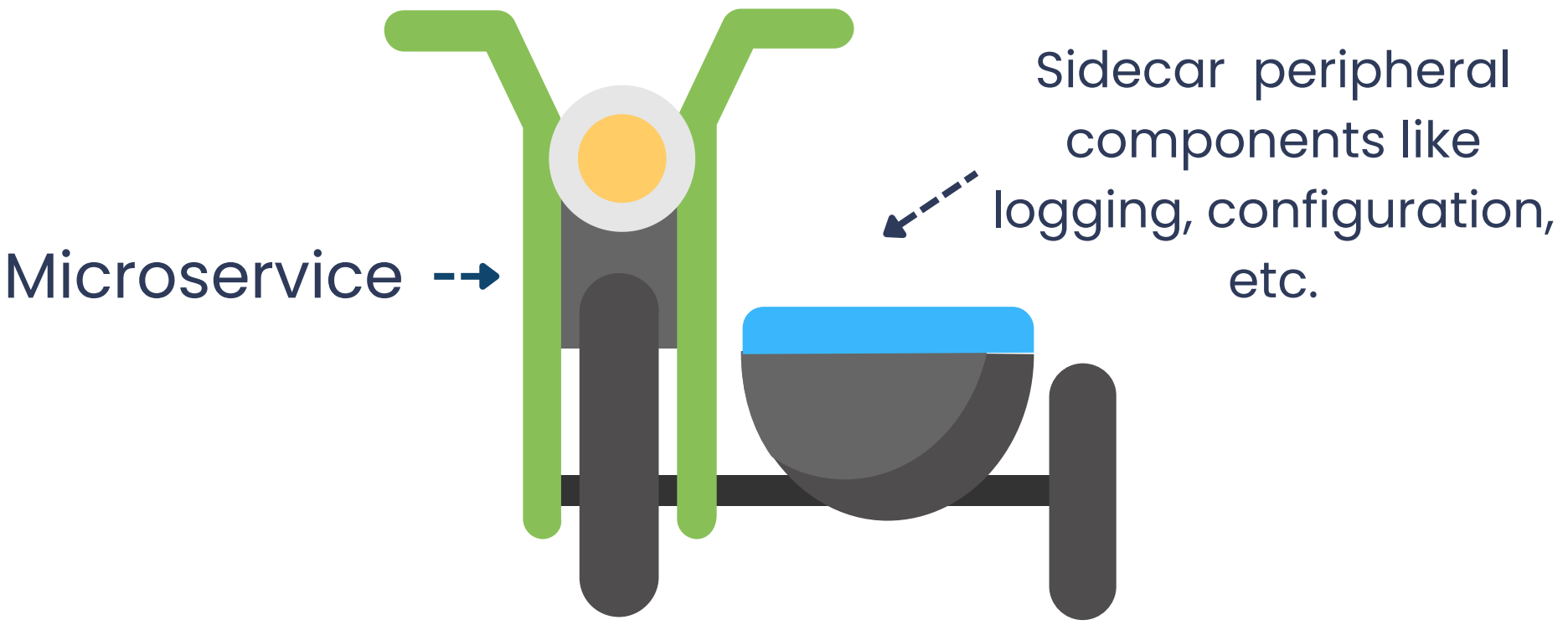
Bulkhead Pattern



Bulkhead pattern isolates services into compartments so failures in one area don’t bring down the entire system architecture.

Sathish Kumar Subramani

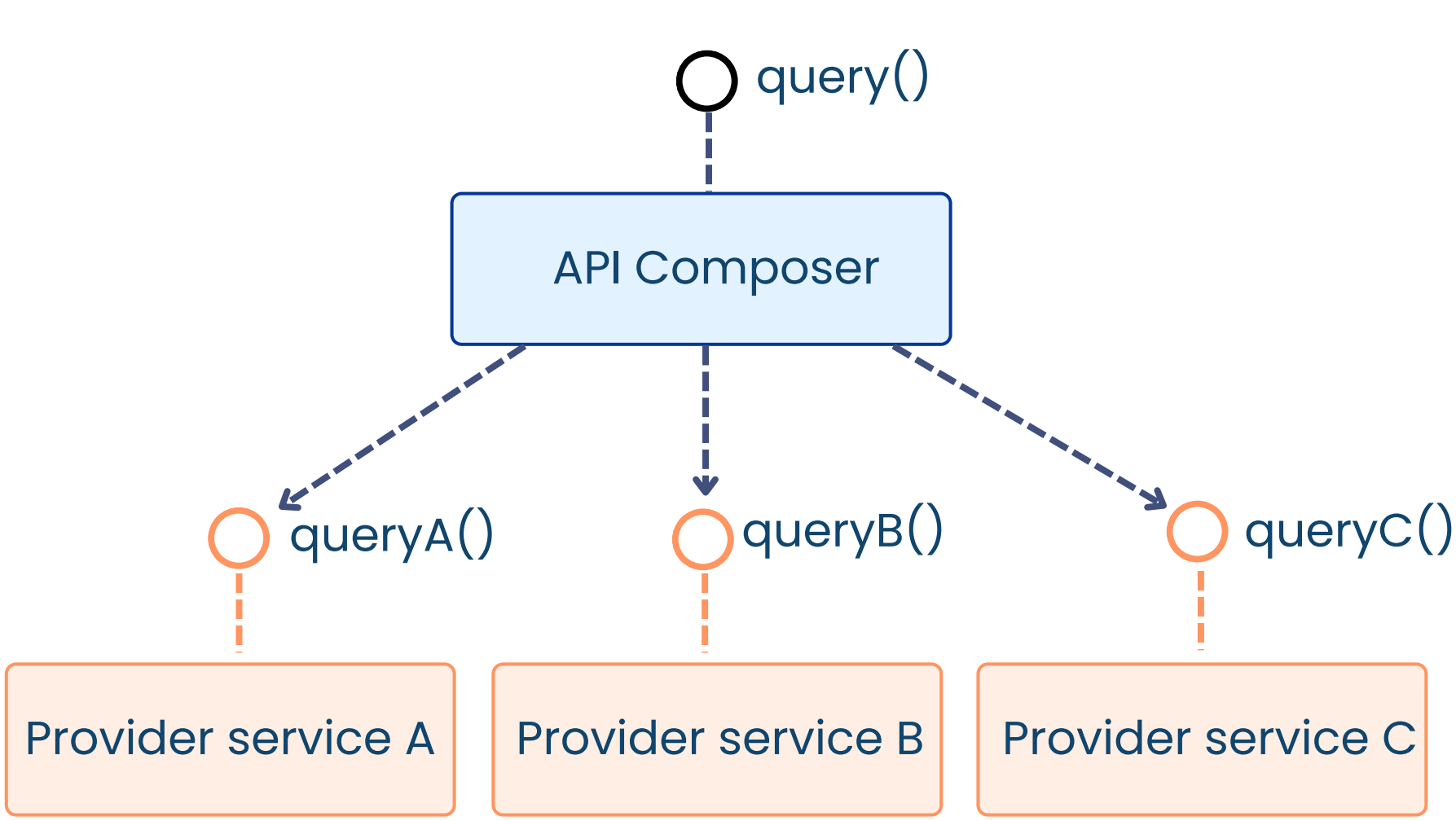
Sidecar Pattern



Sidecar attaches auxiliary microservices to a core service to handle cross-cutting concerns like configuration, logging, monitoring, and service discovery.

Sathish Kumar Subramani

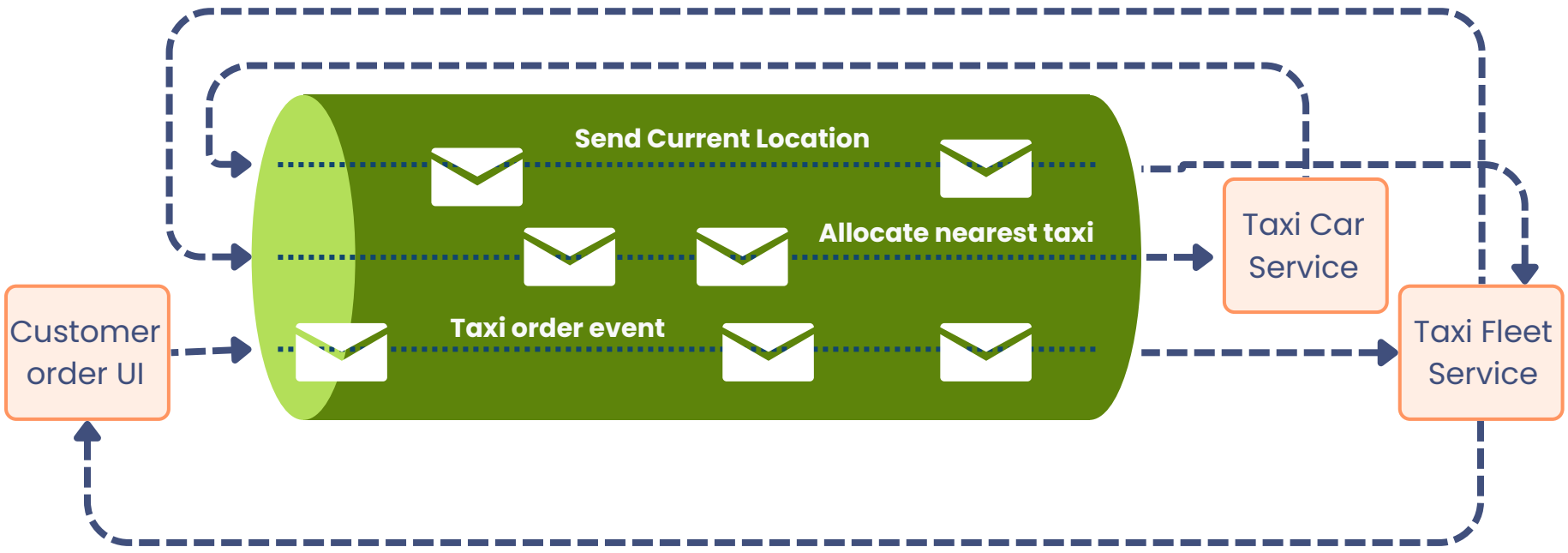
API Composition Pattern



Combine data from multiple services into a single response by composing APIs, delivering rich, client-optimized responses with reduced client overhead.

Sathish Kumar Subramani

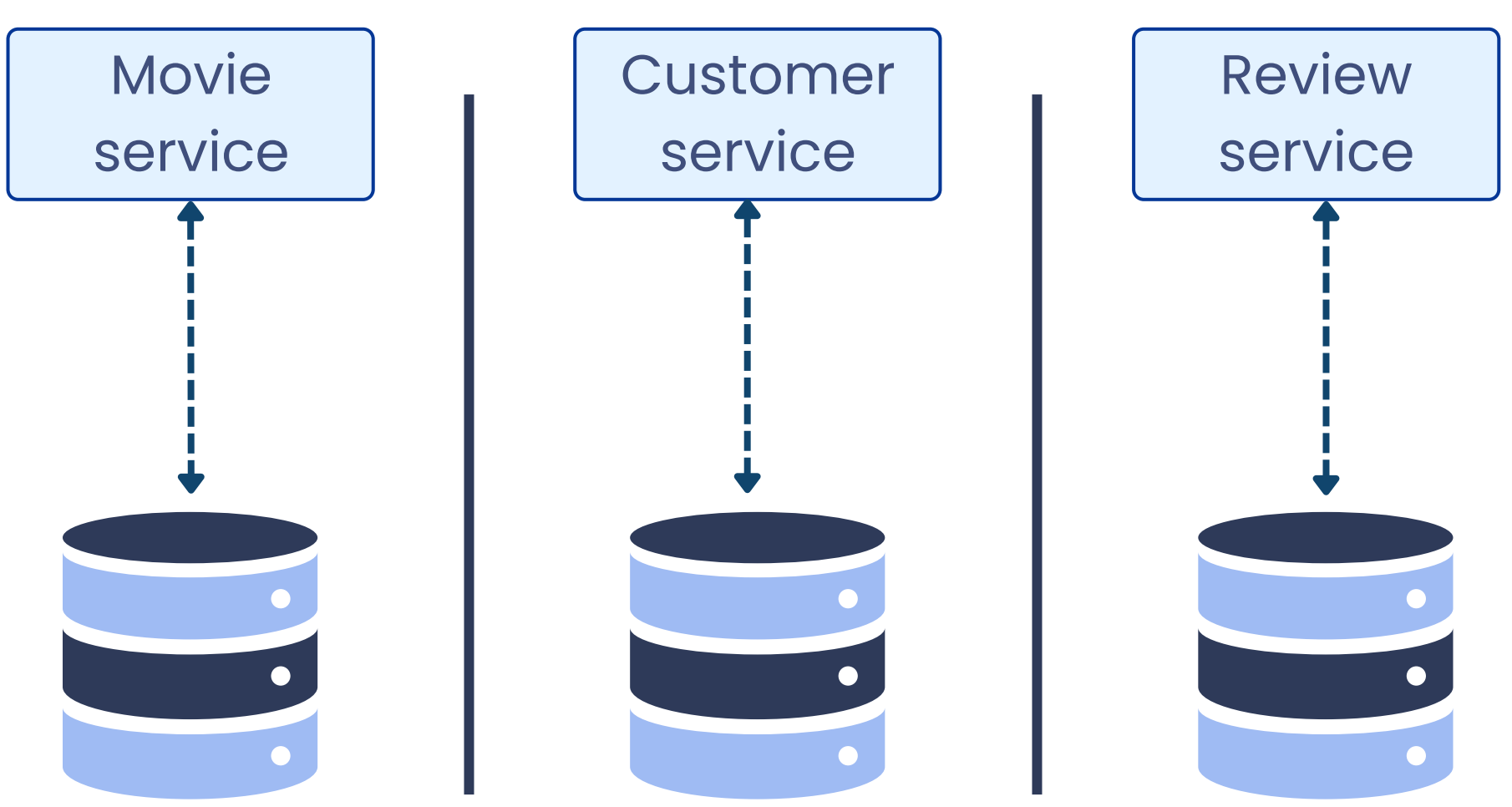
Event-Driven Architecture Pattern



Microservices communicate asynchronously via events, promoting scalability, loose coupling, and independent service evolution in distributed systems architecture.

Sathish Kumar Subramani

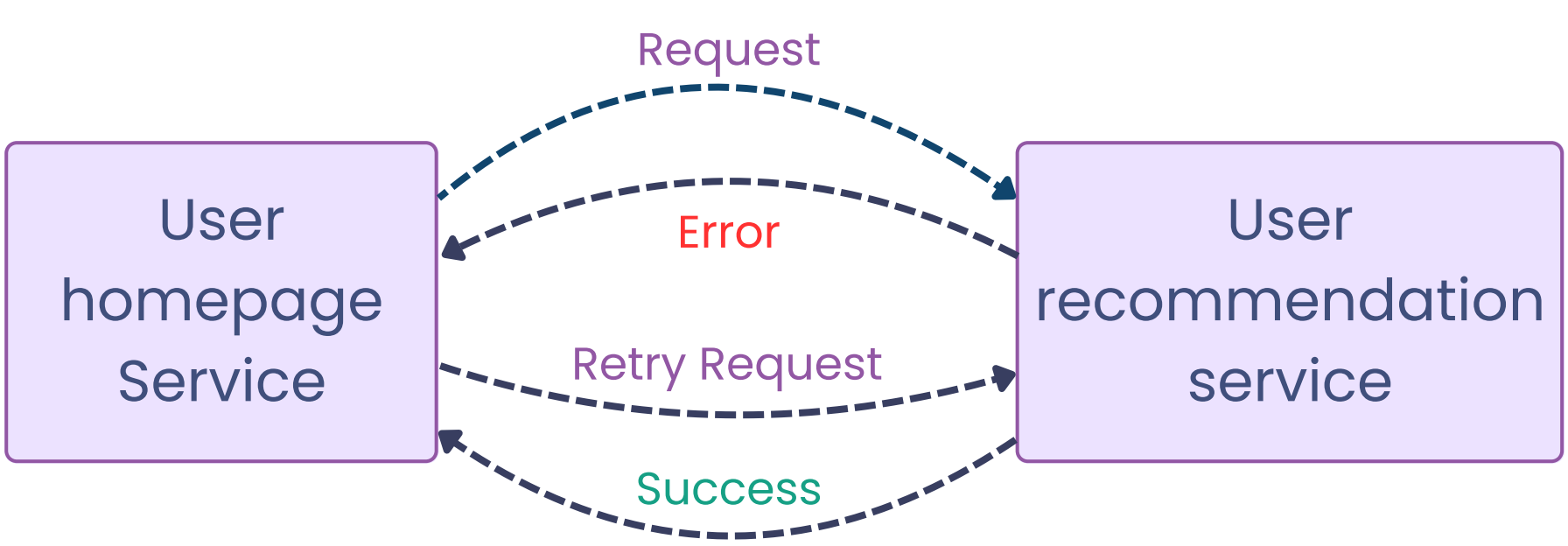
Database per Service Pattern



Each microservice uses its own database, promoting autonomy, decoupling, and independent scaling, updates, and deployments for better microservice isolation.

Sathish Kumar Subramani

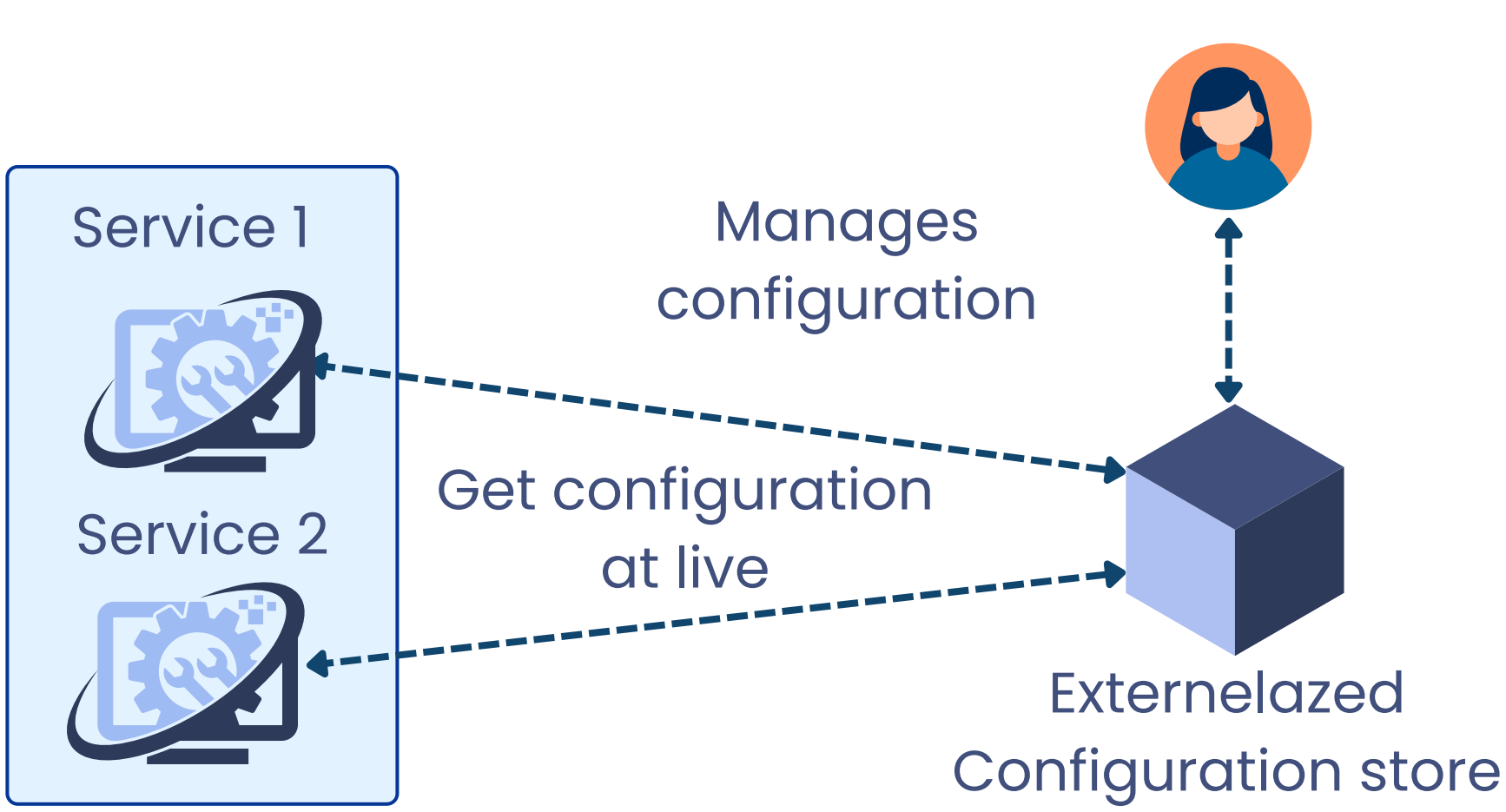
Retry Pattern



The Retry pattern automatically re-attempts failed operations after delays, increasing the system’s reliability, fault-tolerance, and chances of successful responses.

Sathish Kumar Subramani

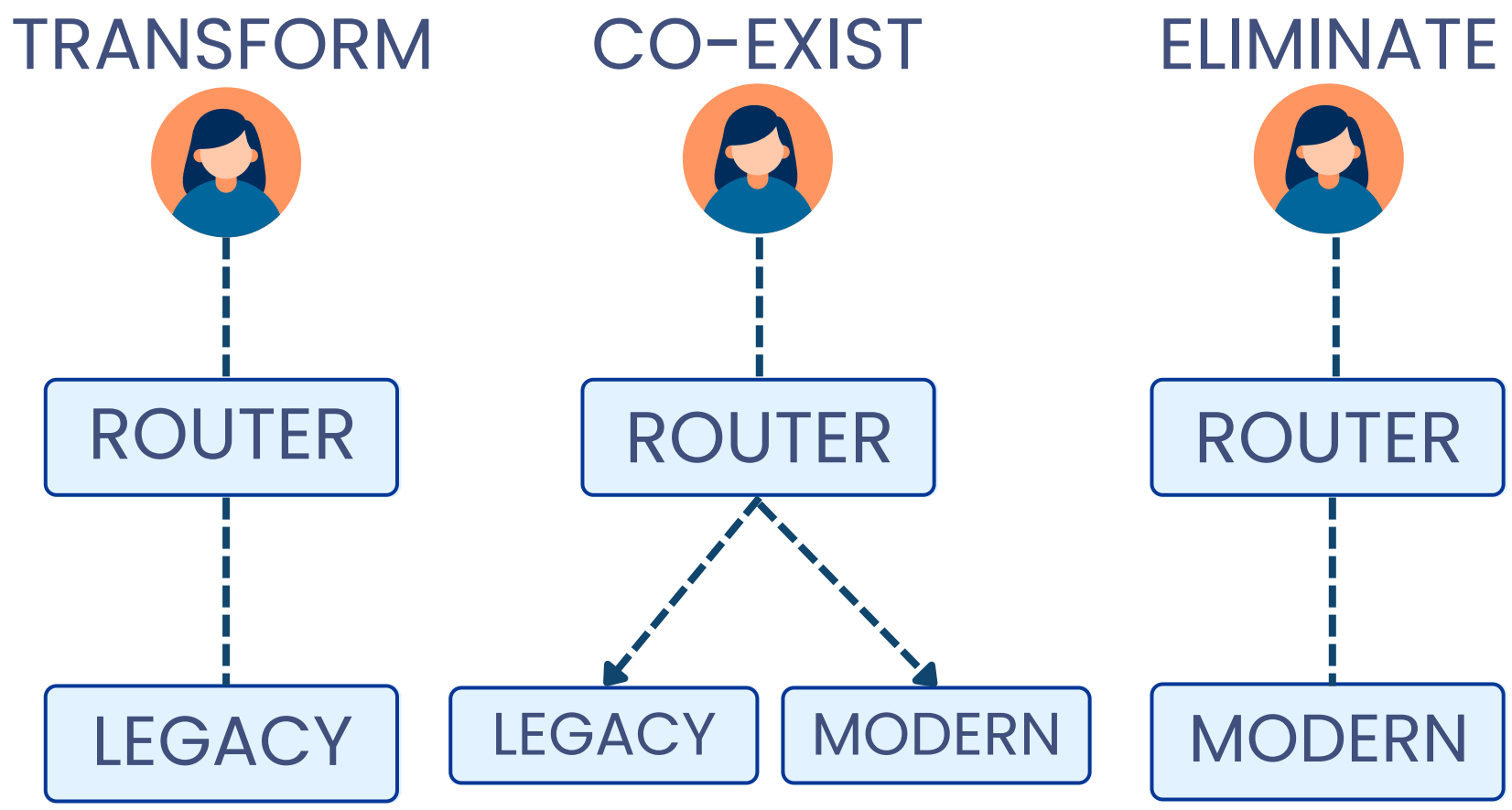
Configuration Externalization Pattern



Externalize configuration settings from application code for centralized management, making updates easier and ensuring consistency across environments.

Sathish Kumar Subramani

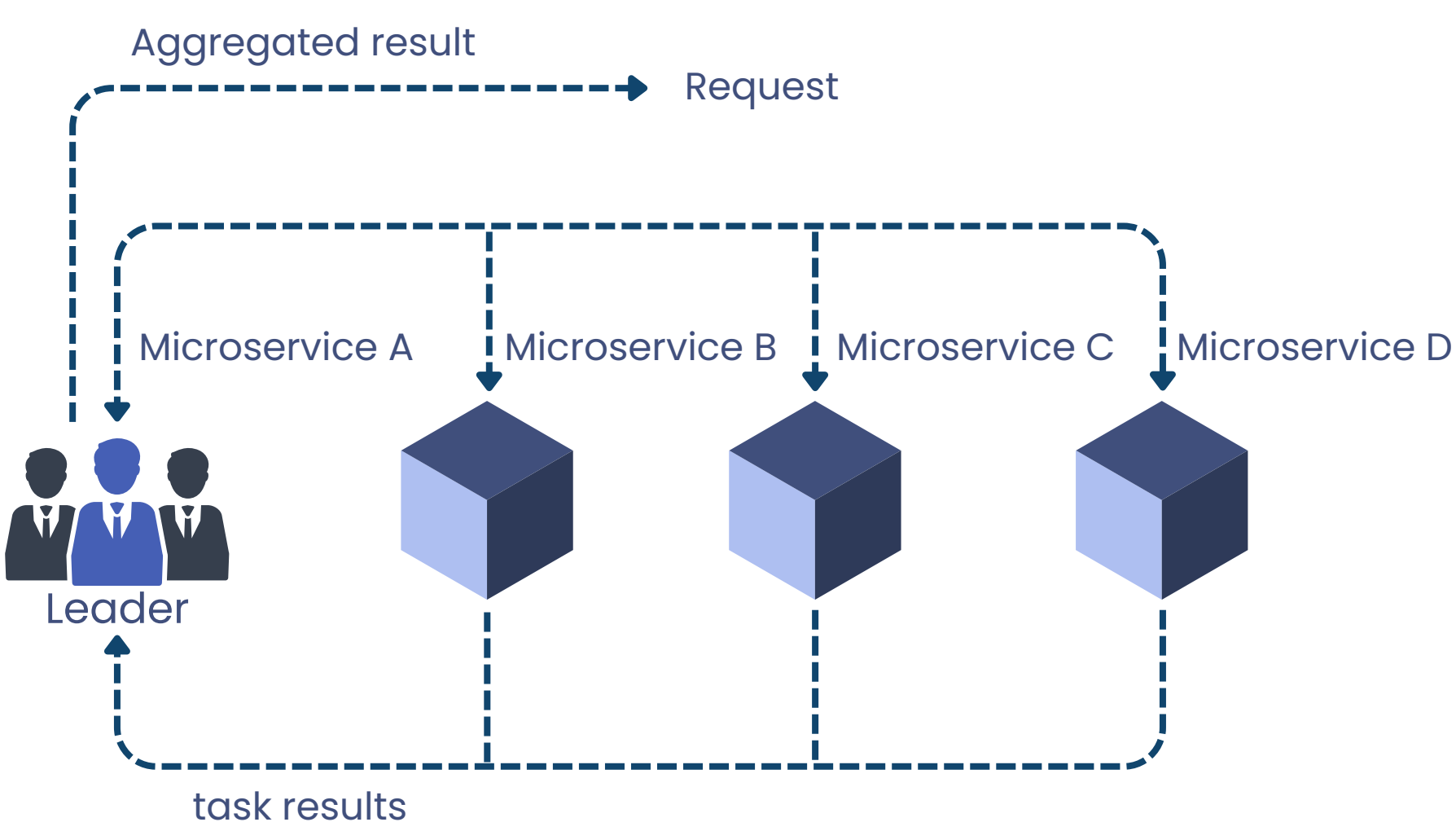
Strangler Fig Pattern



Strangler Fig gradually replaces legacy systems by routing traffic to new components, allowing safe, staged modernization without full replacement.

Sathish Kumar Subramani

Leader Election Pattern



Elect a leader among service instances to coordinate tasks, improve consistency, and manage distributed workflows requiring single ownership.

Sathish Kumar Subramani

Ready to Architect Better Systems?

Follow me