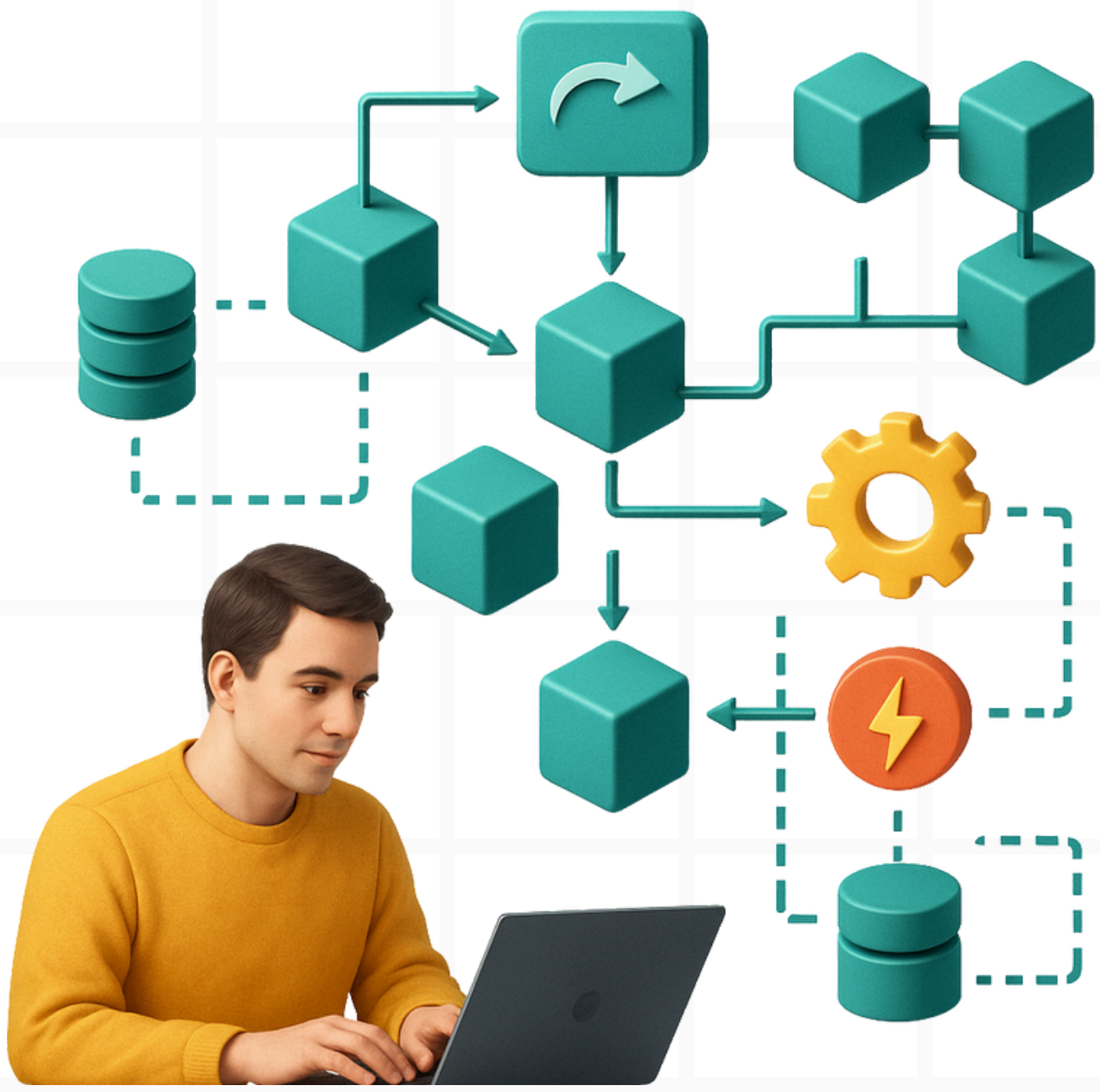
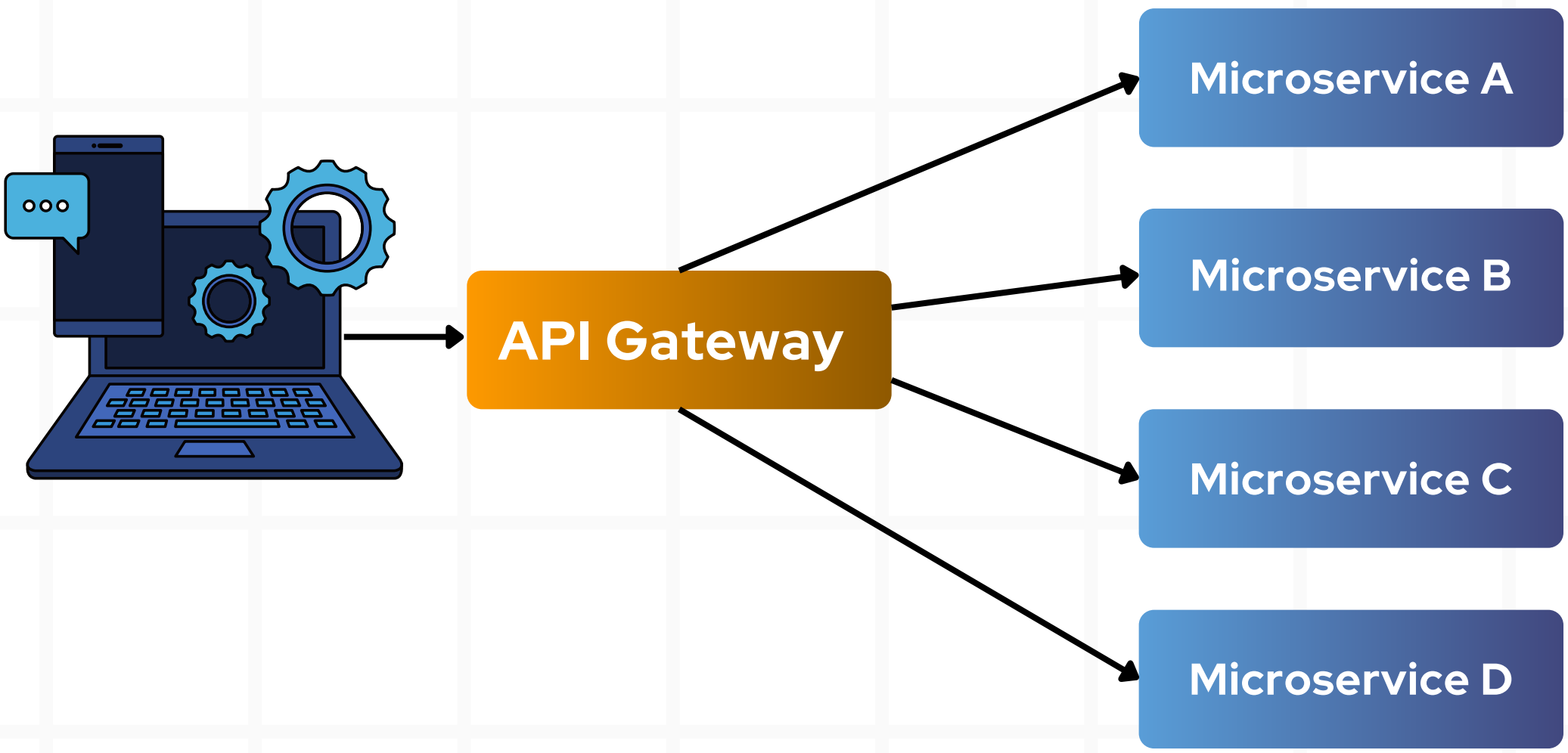
DESIGN PATTERNS FOR MICROSERVICES

Proven patterns to enhance scalability, resilience, and maintainability in microservices by improving service communication, fault isolation, and overall system efficiency.



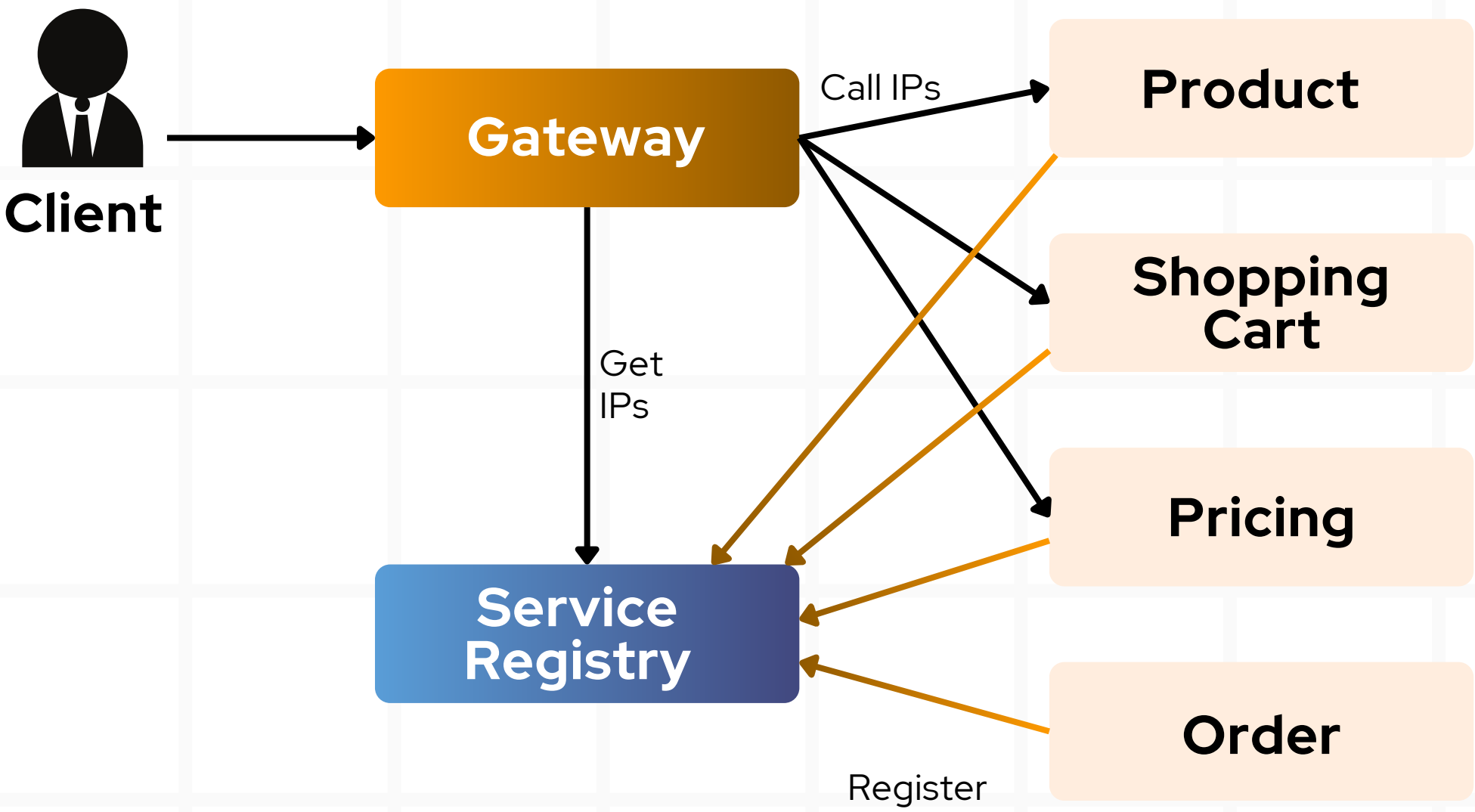
GATEWAY PATTERN

A single entry point (API Gateway) routes external requests to services while handling security, load balancing, and traffic logic centrally and efficiently.



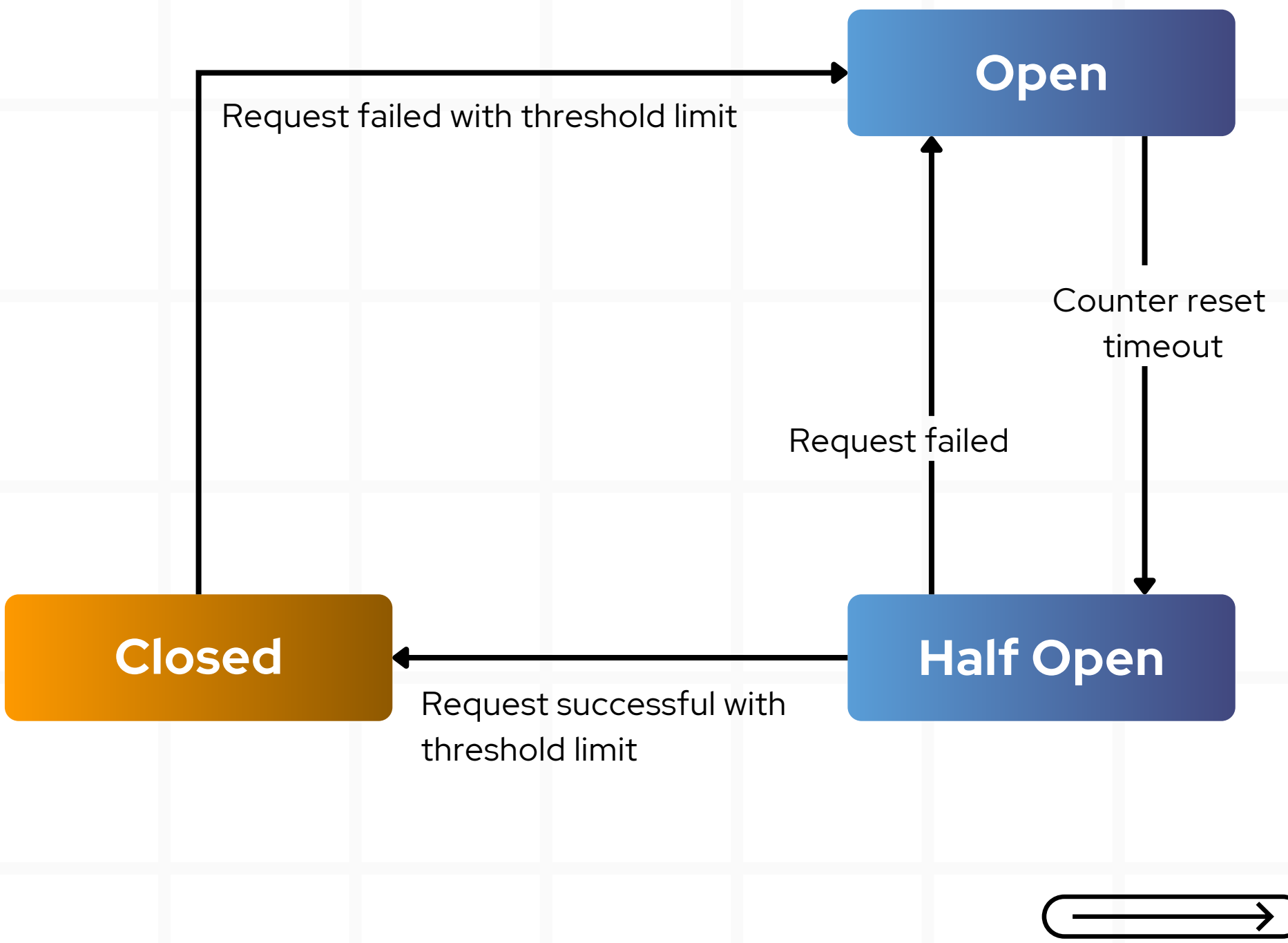
SERVICE REGISTRY PATTERN

Microservices register themselves into a central directory, making it easy for other services to discover and communicate with them in real time.



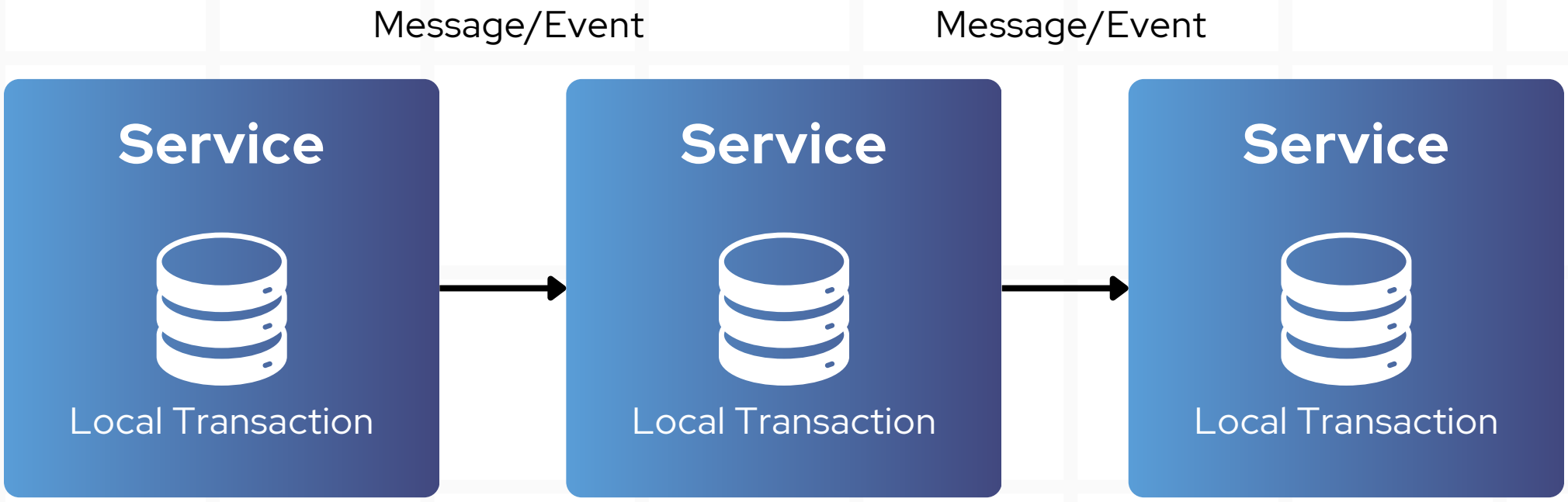
CIRCUIT BREAKER PATTERN

Circuit breakers protect your system from failure loops by halting calls to unstable services and switching to safe fallback responses during downtime.



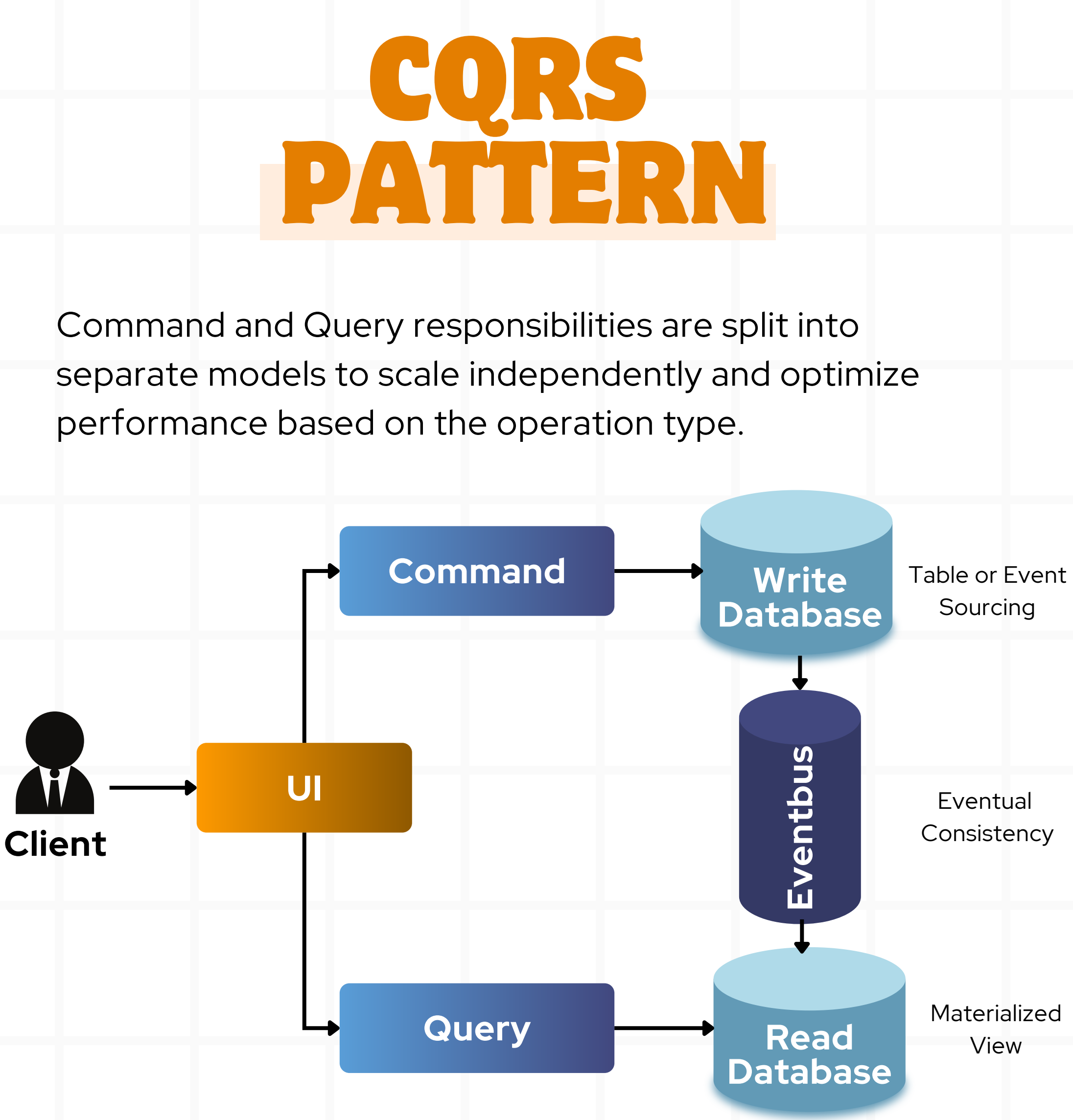
SAGA PATTERN

For long-running operations, break them into a chain of local transactions, each with its own rollback or compensating action if something fails.



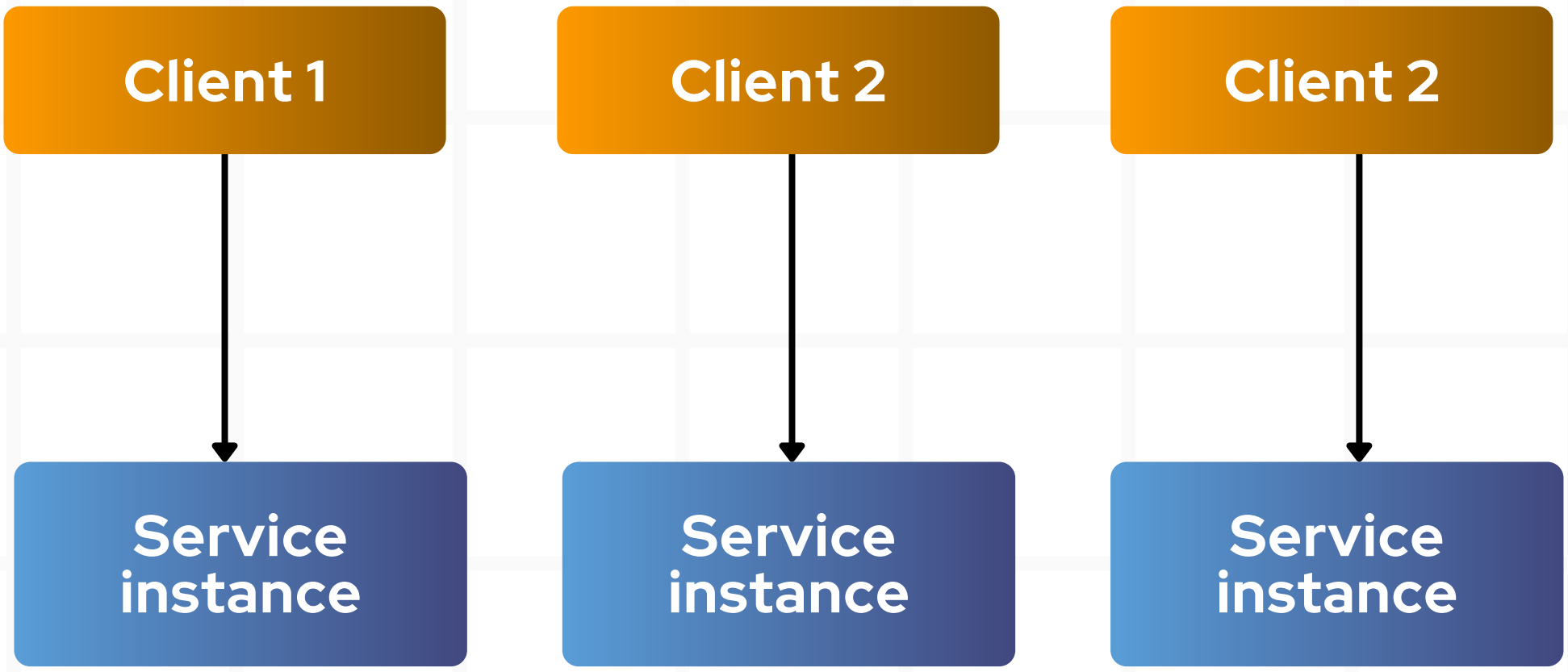
CQRS PATTERN

Command and Query responsibilities are split into separate models to scale independently and optimize performance based on the operation type.



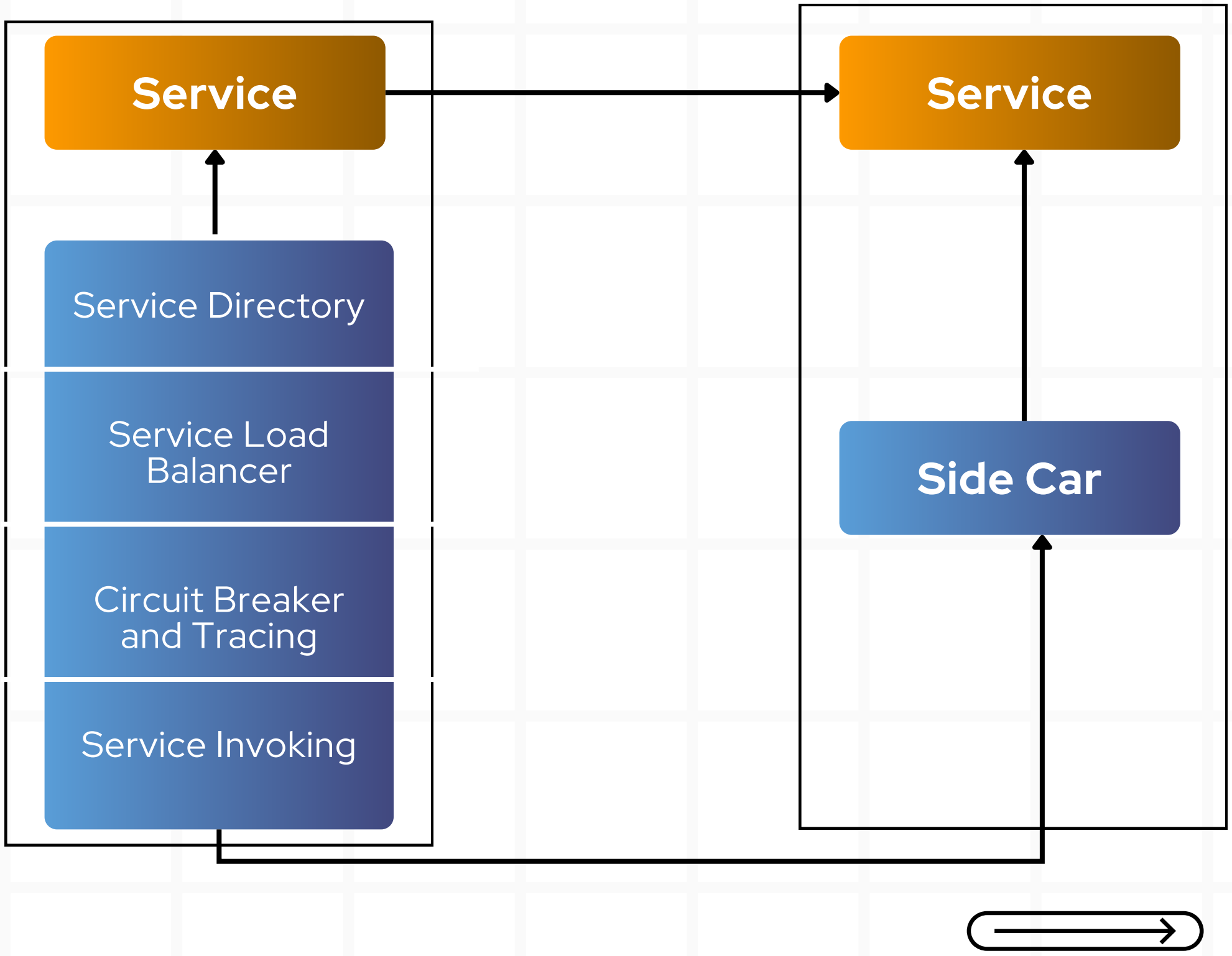
BULKHEAD PATTERN

Limit failures from spreading by isolating parts of your system into independent units, ensuring other components keep running if one fails.



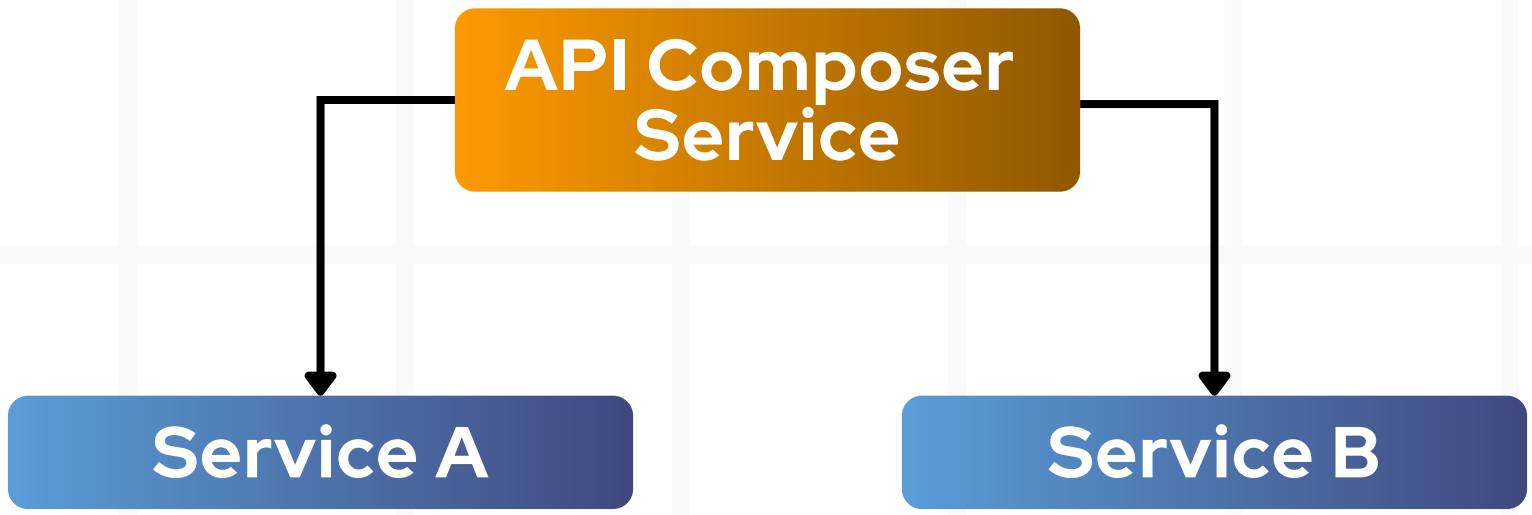
SIDECAR PATTERN

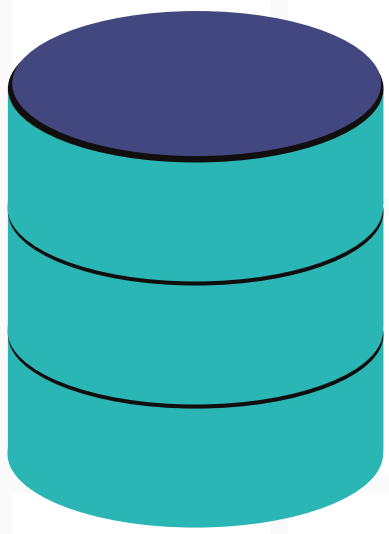
Offload responsibilities like monitoring or proxying into a paired helper service (sidecar) running alongside the main service container.



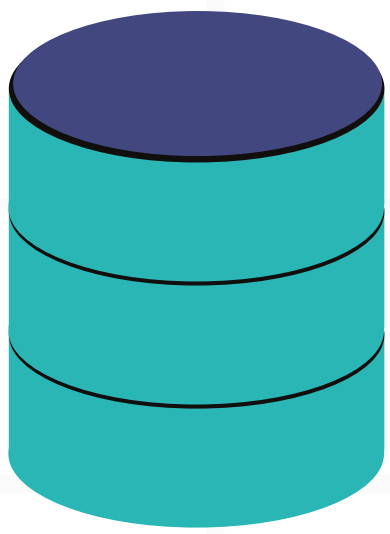
API COMPOSITION PATTERN

Merge data from multiple microservices through a composition layer to present a unified and enriched response to client applications.





Database A



Database B

DATABASE PER SERVICE PATTERN

Assigning a separate database to each microservice maintains modularity, allowing independent scaling, development, and versioning of service data.