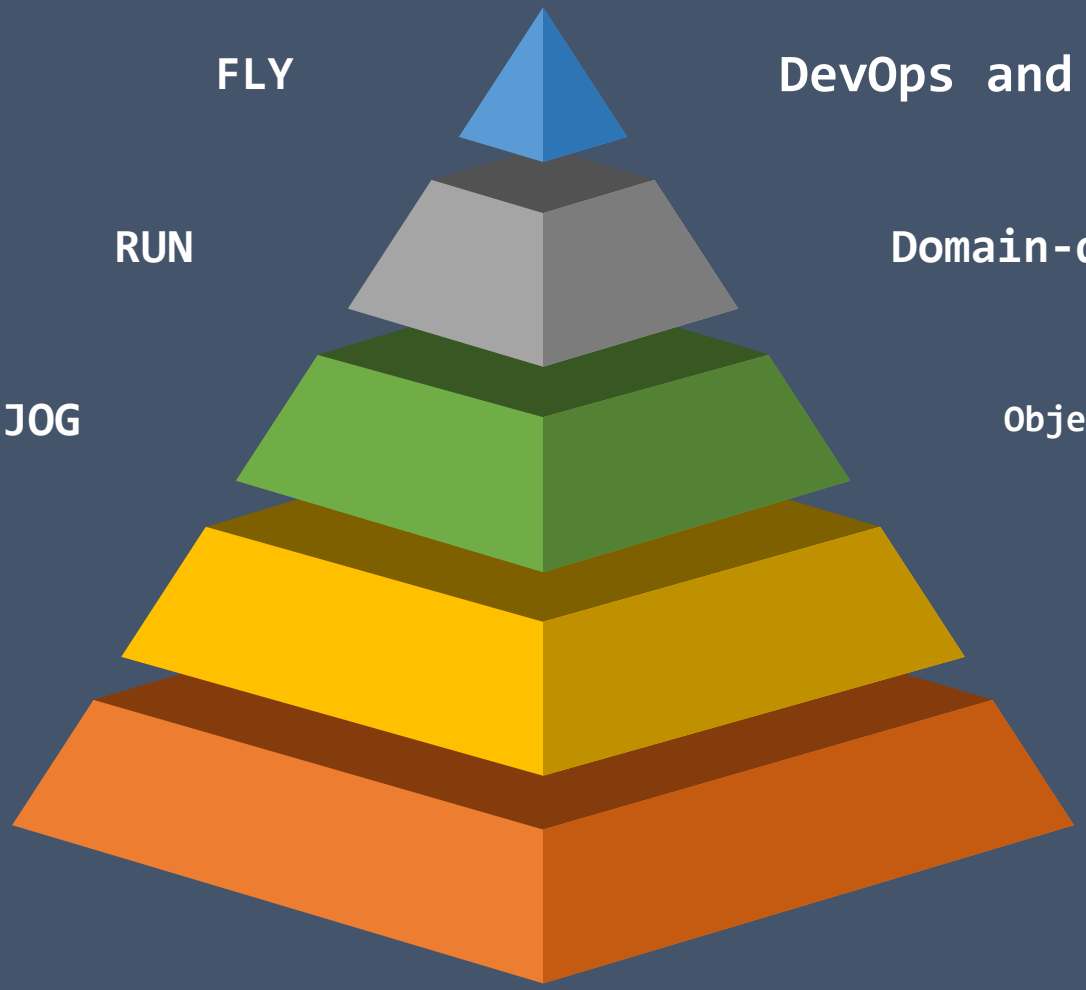
MICROSERVICE

Osman Tulgar Yaycıoğlu

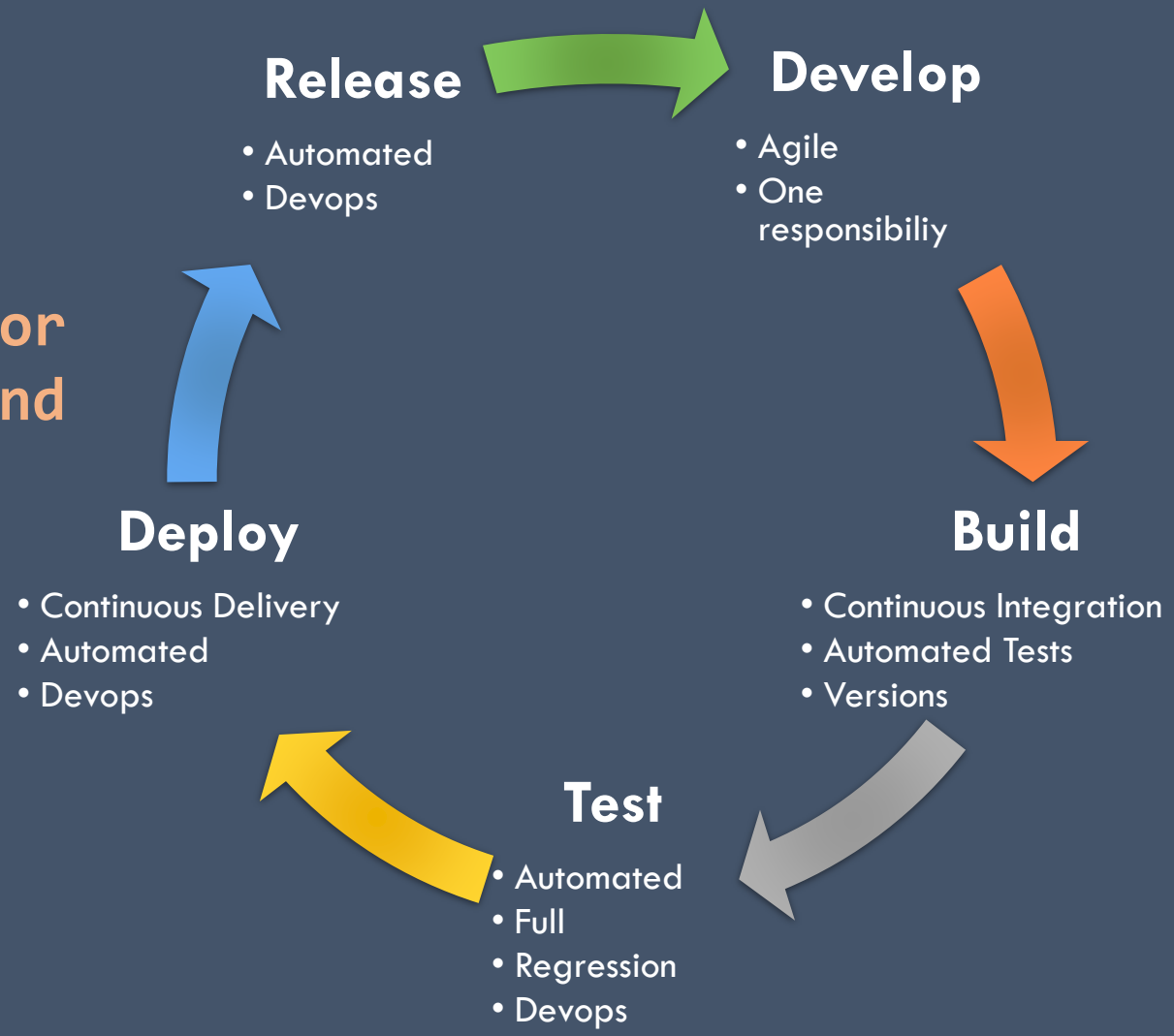
EVOLUTION


FLY
DevOps and Microservices
RUN
Domain-driven design
JOG
Object-Oriented Design
WALK
Unit testing
Automated testing
Continuous Integration
CRAWL
Clean Code

CONTINUOUS DELIVERY

- Ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way.

  - Low risk releases.
  - Faster time to market.
  - Higher quality.
  - Lower costs.
  - Better products.



-Y OF ARCHITECTURE

- Availability
- Flexibility
- Maintainability
- Deployability
- Testability
- Evolvability
- Scalability
- Reliability
- Stability
- Forward Compatibility
- Backward Compatibility
- Resiliency
- Elasticity
- Reusability
- Granularity

AVAILABILITY

- 99.9 - 8h 45m 56s
- 99.99 - 52m 35s
- 99.999 - 5m 15s
- 99.9999 - 31s
- 99.8 - 17h 31m 53s
- 99.5 - 1d 19h 49m 44s

COMMON OBSTACLES TO RAPID, FREQUENT AND RELIABLE SOFTWARE DELIVERY

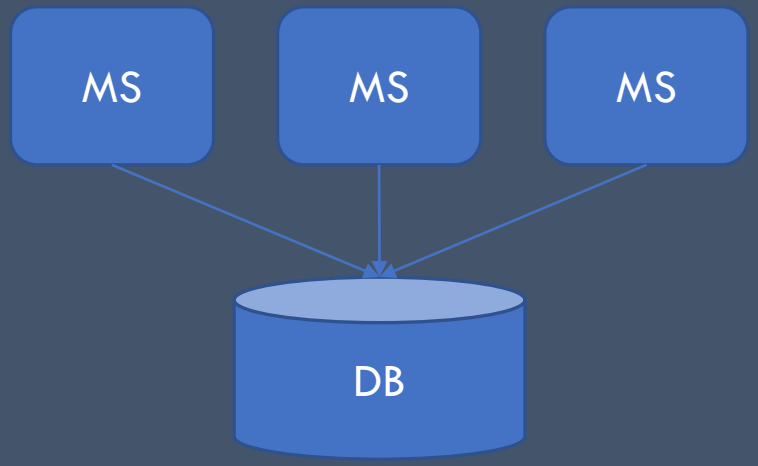
- Slow, silo’ed, manual development, testing and deployment process
- Applications are big balls of mud
- Stinky code
- Duplicate code bases
- Monthly deploys at midnight
- Testing cost bigger than development cost

OBJECT ORIENTED

- Inheritance: child classes inherit data and behaviors from parent class
- Encapsulation: containing information in an object, exposing only selected information
- Abstraction: only exposing high level public methods for accessing an object
- Polymorphism: many methods can do the same task

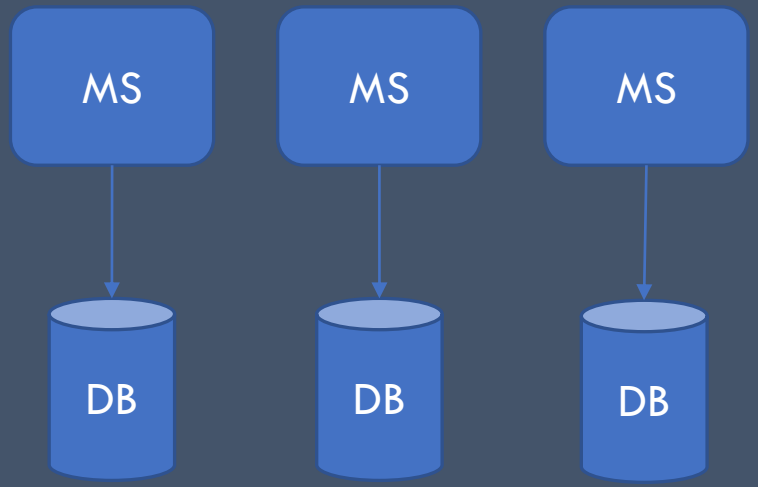
- Segregation of duty
- Single responsibility
- Interface segregation
- Do one thing and do it right
- MISS Make it simple and stupid
- Open Close principle
- Right size
- Loose coupled
- High cohesion

Shared Database



- Business transactions may need to query data that is owned by multiple services.
- Aggregating data that is owned by multiple services.
- A single database is much simpler to operate.
- Creates development time & run time coupling
- Widely Considered as anti-pattern.

Database per service



- Data needs to be encapsulated with the business logic that operates on the data.
- Data access only via a published service interface.
- No direct database access is allowed from outside the service.
- No data sharing among the service
- Enables polyglot persistence

COMMUNICATION

- Synchronous protocol. HTTP is a synchronous protocol. The client sends a request and waits for a response from the service. Client code can only continue its task when it receives the HTTP server response. Usually follow REST architectural style.

- Asynchronous protocol. Protocol like AMQP use asynchronous messages. The client code or message sender usually does not wait for a response. Follows Smart endpoints and dumb pipes pattern.

EVENT DRIVEN ARCHITECTURE

- Event Notification
- Event Carried State Transfer
- Event Sourcing
- CQRS –Command Query Responsibility Segregation

API DESIGN

An API gateway sits between clients and micro services. It acts as a reverse proxy, routing requests from clients to services.

- Single URL and consistent interface.
- Reduces the coupling between client and backend.
- Act as aggregator.
- Exposes client-friendly protocol such as HTTP or WebSocket.
- SSL termination
- Authentication
- IP whitelisting
- Client rate limiting (throttling)
- Logging and monitoring
- Response caching

API DESIGN

- API design is important in a microservices architecture -all data exchange between services happens either through messages or API calls.
- An API is a contract between a service and clients or consumers of that service. If an API changes, there is a risk of breaking clients that depend on the API.
- REST over HTTP using JSON is a common choice for API interface.
- Consider using an Interface definition language (IDL) for designing API contracts. Latest standard is Open API.
- APIs should not leak internal implementation. It should change only when new functionality is added.
- Consider using the Backends for Frontends pattern to create separate backends for each client.
- For operations with side effects, consider making them idempotent and implementing them as PUT methods.
- Return proper http status codes.
- Consistent error handling. Create custom envelop for communicating errors.

CHARACTERISTICS OF SERVICE

- Independent
- Isolated
- Autonomous
- Decentralized
- Reusable
- Failure isolation
- Auto-Provisioning
- Discoverable
- Self repair
- Self registered
- Fine grained
- Componentization
- Loosely Coupled
- Maintainable
- Map to Business

SERVICE

- Standardized service contract (services follow a standardized description)
- Loose coupling (minimal dependencies)
- Service abstraction (services hide their internal logic)
- High cohesion
- Fine grained
- Service reusability (service structure is planned according to the DRY principle)
- Service autonomy (services internally control their own logic)
- Service statelessness (services don’t persist state from former requests)
- Service discoverability (services come with discoverable metadata and/or a service registry)
- Service composability (services can be used together)

PROMISES OF MICROSERVICES

- Promise of agility, and faster time-to-market
- Promise of teams to be autonomous.
- Promise of more innovation
- Promise of experimenting and adoption of new technologies.
- Promise of better resilience
- Promise of better fault isolation
- Promise of better scalability
- Promise of better reusability
- Promise of Improved Return on Investment
- Promise of CD and deployment of large, complex applications.
- Promise of small and easily maintained services
- Promise of independently deployable services.

MICROSERVICE CHALLENGES

- Managing Microservices
- Monitoring
- Configuration
- Embracing DevOps Culture
- Fault Tolerance
- Testing
- Cyclic Dependencies
- Centralization in Decentralized world
- Maintaining services
- Debugging
- Needs more collaboration
- Operational Complexity
- Performance Hit Due to Network Latency
- Harder to test and monitor
- Poorer performance
- More Resources
- Harder to maintain the network
- Security issues
- Living with a Legacy
- Impact Analysis
- Robust monitoring is a must
- Data centric thinking
- Monolith paradigm
- Migration from monolith
- Embrace Change
- Harder to Troubleshoot
- Architectural Complexity
- Higher Costs
- High initial investments

MICROSERVICE BENEFITS

- Easier to Build and Maintain Apps
- Organized Around Business Capabilities
- Improved Productivity and Speed
- Autonomous, Cross-functional Teams
- Better fault isolation
- Deploy and redeploy Independently
- Easier continuous delivery
- Easy to understand
- Scalability and reusability and efficiency
- Distributed system
- Fine grained
- Promotes containers
- Flexibility in Using Technologies and Scalability
- Increase the autonomy in every part of development
- CI/CD pipeline for single-responsibility services

MICROSERVICE BENEFITS

- Fine-grained Scaling
- Technology Independence
- Parallel Development
- Better Fault Isolation
- Easier to Refactor
- Easy to Understand
- Faster Developer Onboarding
- Complement cloud activities, Cloud-readiness
- Availability
- ...

- Greater agility
- Faster time to market
- Faster development cycles
- Faster deployment cycles
- Isolated services have better fault tolerance
- Better Performance
- Security
- Small Team, Big Job
- ...

BREAKING THE DOMAIN USING DDD

- Domain Driven Design (DDD) is about designing software based on models of the underlying domain. A model acts as a Ubiquitous Language to help communication between software developers and domain experts.
- Ubiquitous Language –A common language between developers and the business.
- Bounded Context –The setting in which a word or statement appears that determines its meaning.
- Domain Entity –A domain object means something to the business, defined primarily by its identity.
- Value Object –Defined primarily by its attributes and is immutable.
- Aggregate Root –A group of entities and value objects that needs to be consistent while persisting.
- Core Subdomain -The core domain is so critical and fundamental to the business that it gives you a competitive advantage and is a foundational concept behind the business.
- Supporting Subdomain -Help perform ancillary or supporting functions related directly to what the business does. In these cases, high-quality code and perfectly designed structure are not necessary.
- Generic Subdomain -In general, these types of pieces can be purchased from a vendor and then wrapped in such a way to communicate with the rest of the enterprise as necessary (Anti-corruption layer)

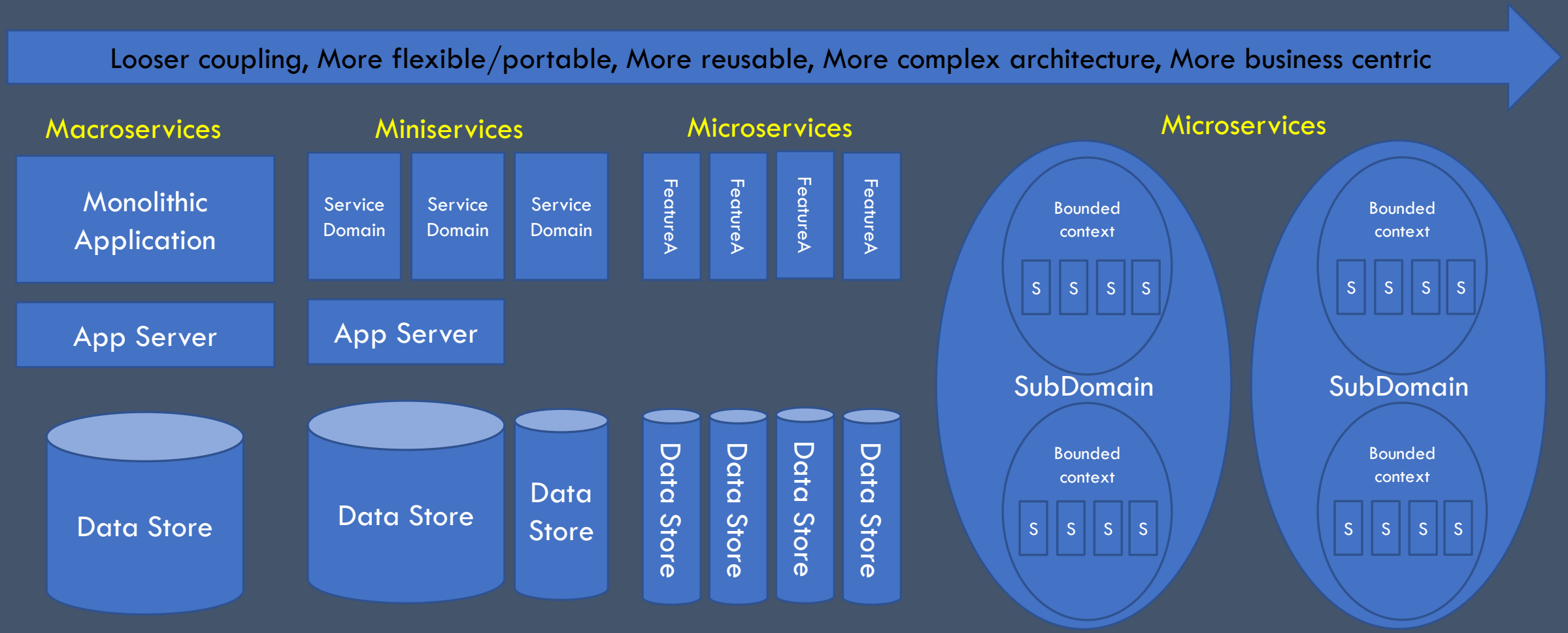
BREAKING THE DOMAIN USING DDD

- Always decompose to Bounded Contexts
- Don’t go further, unless you have to
- Buy/adopt generic subdomains
- Core subdomain –don’t rush
- Evaluate consistency requirements
- Evaluate reasons for change
- Don’t expect the ideal model. Expect many iterations.
- Do not over engineer.

Bounded context

Sales context

Support context


Looser coupling, More flexible/portable, More reusable, More complex architecture, More business centric
Macroservices
Monolithic Application
App Server
Data Store
Miniservices
Service Domain
Service Domain
Service Domain
App Server
Data Store
Data Store
Microservices
FeatureA
FeatureA
FeatureA
FeatureA
Data Store
Data Store
Data Store
Data Store
Microservices
Bounded context
S S S S
SubDomain
Bounded context
S S S S
Bounded context
S S S S
SubDomain
Bounded context
S S S S

Process: DevOps/Continuous Delivery & Deployment

Application has outgrown its monolithic architecture
Refactor to microservices

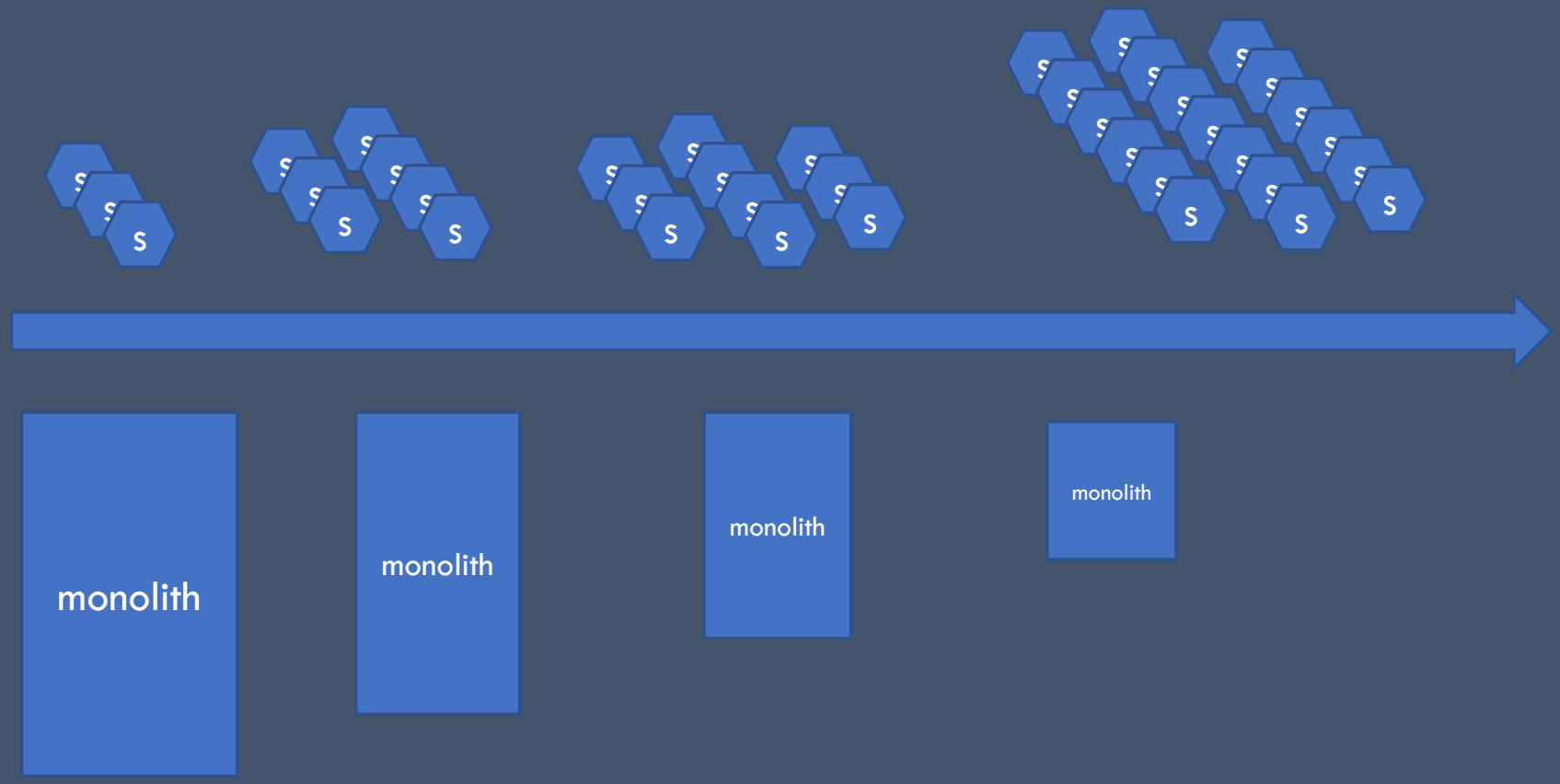
Testability
Deployability

Organization: Small, autonomous teams

Enables Autonomy

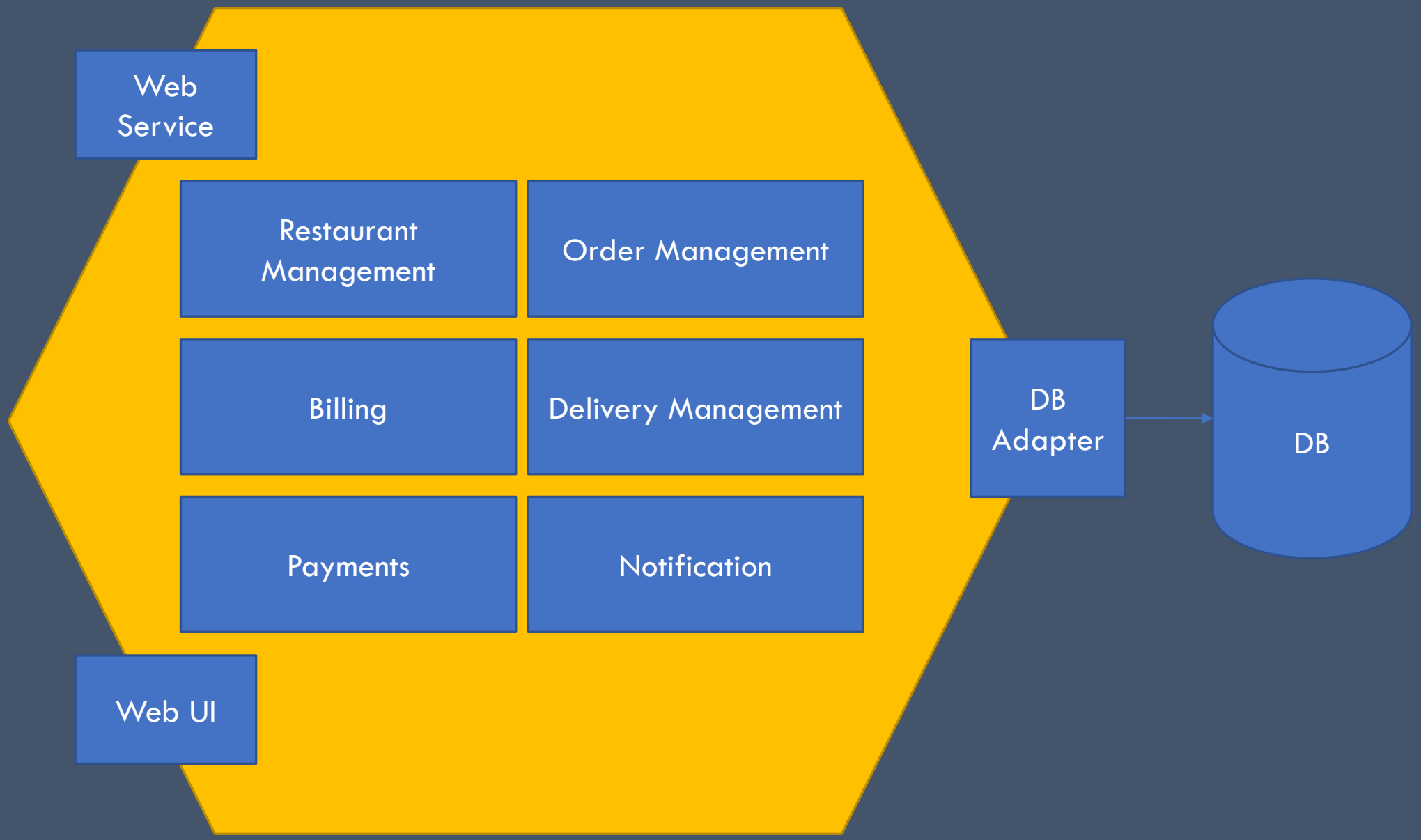
Architecture: microservices

- Microservice Architecture
- Devops
- Small, autonomous teams, etc

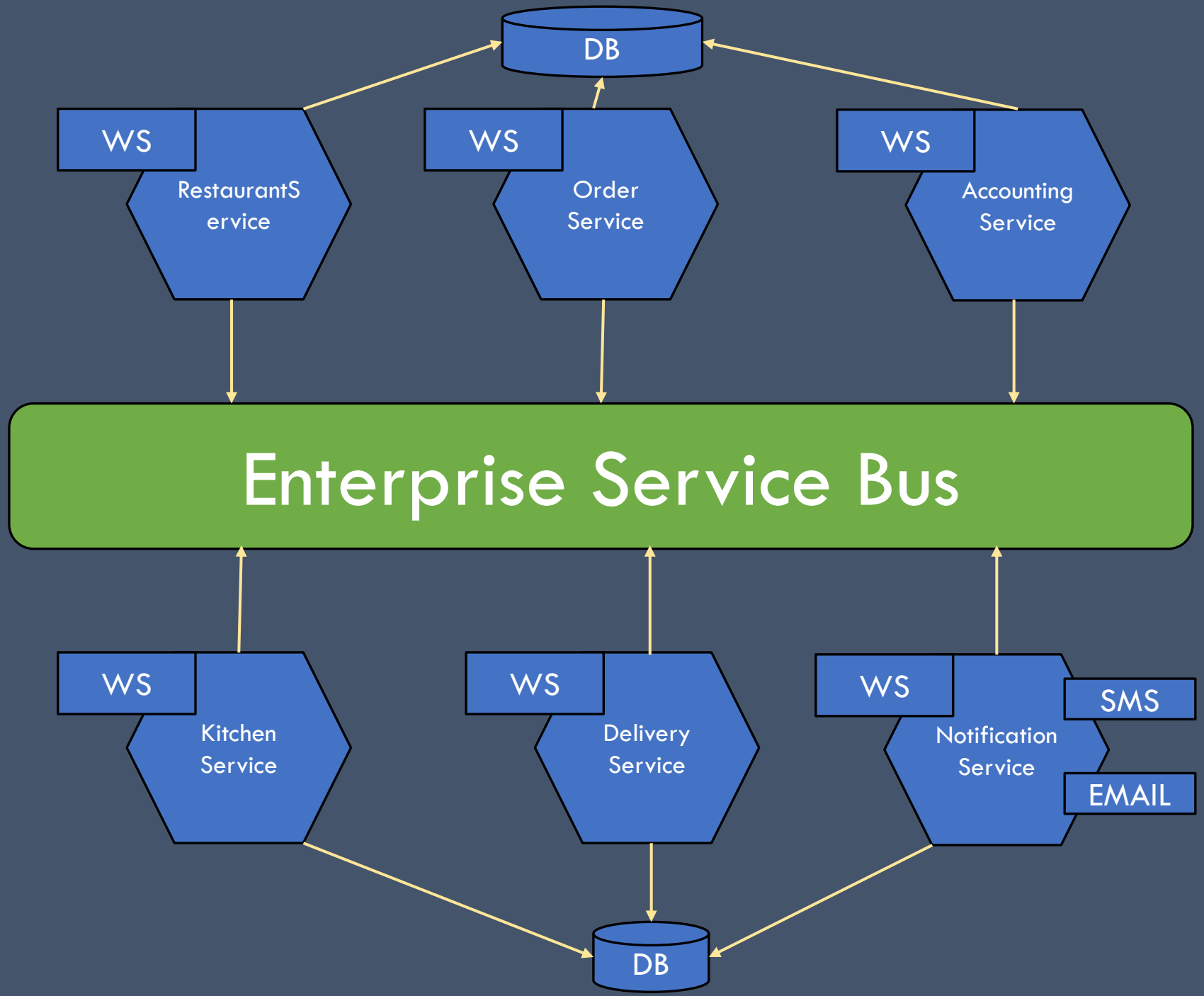


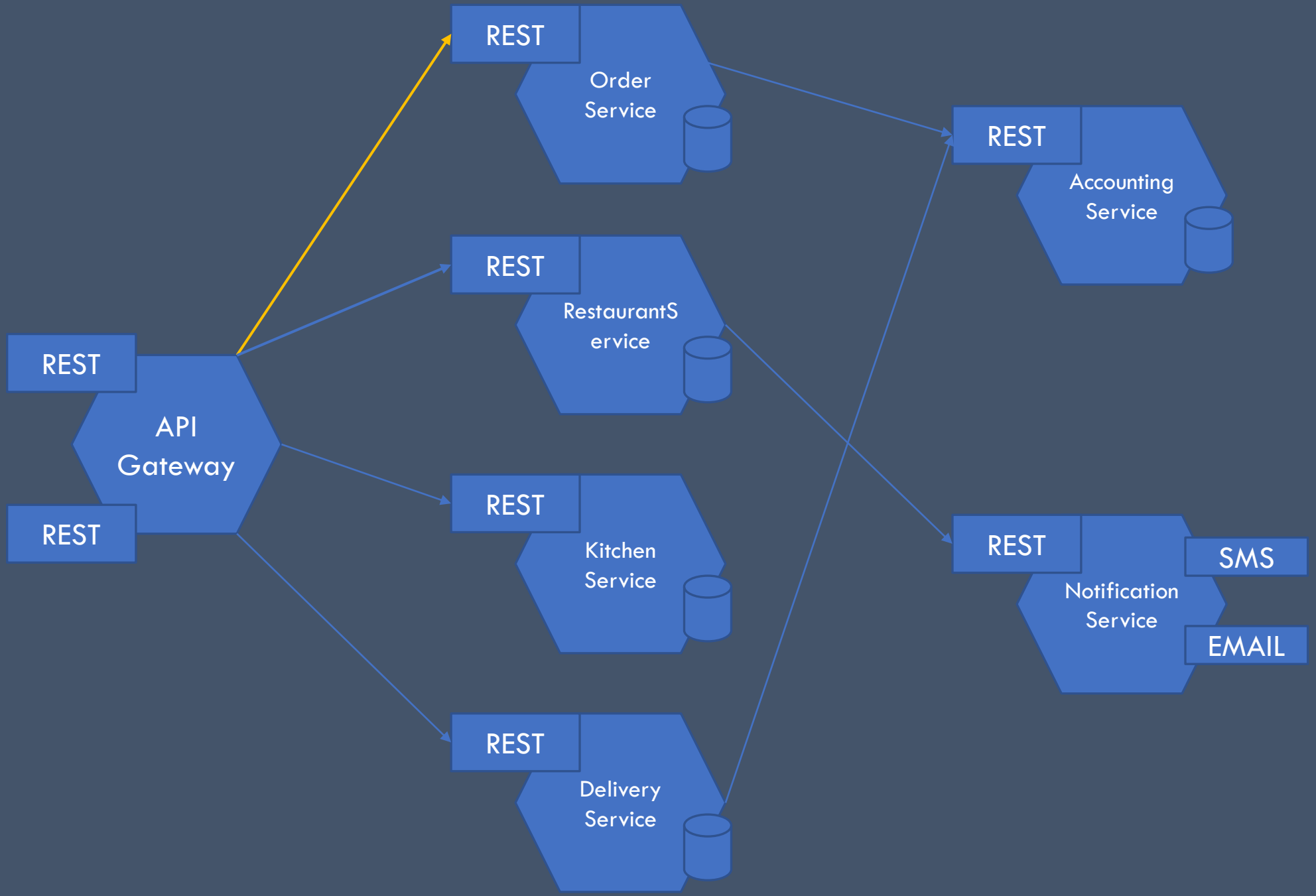
Legacy process and organization

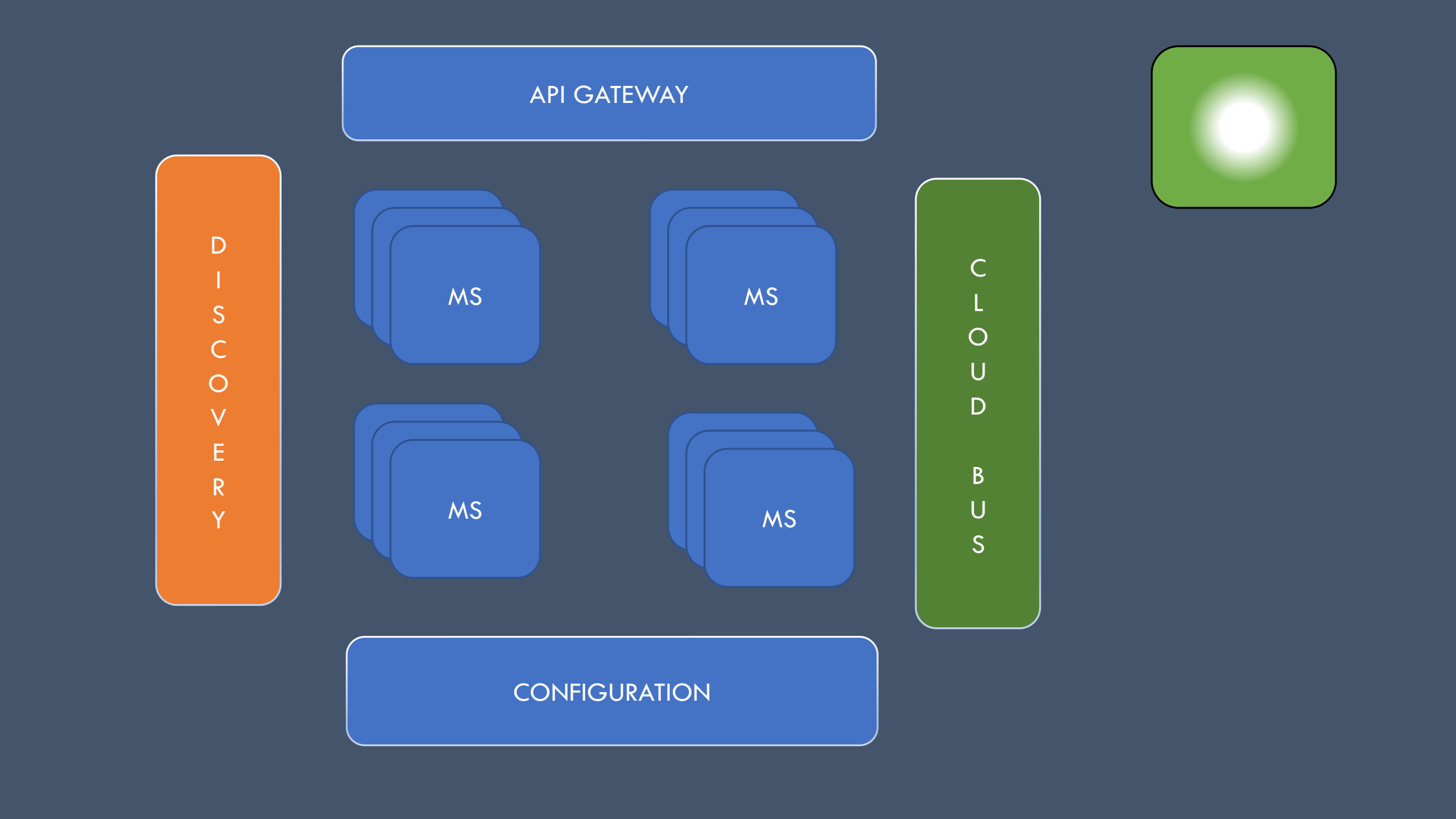
MONOLITH APPLICATION



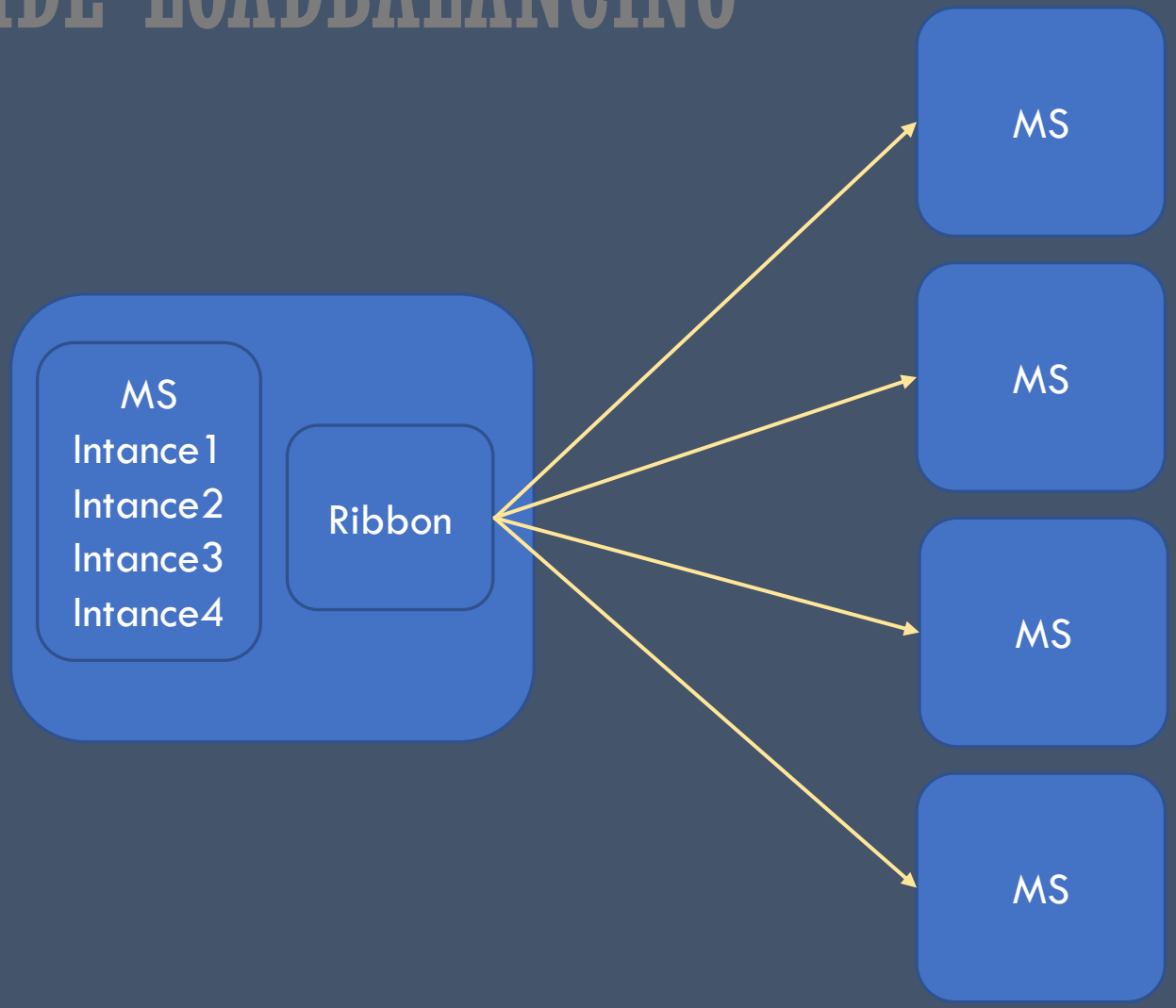
SOA


DB
WS
RestaurantService
WS
Order Service
WS
Accounting Service
Enterprise Service Bus
WS
Kitchen Service
WS
Delivery Service
WS
Notification Service
SMS
EMAIL
DB

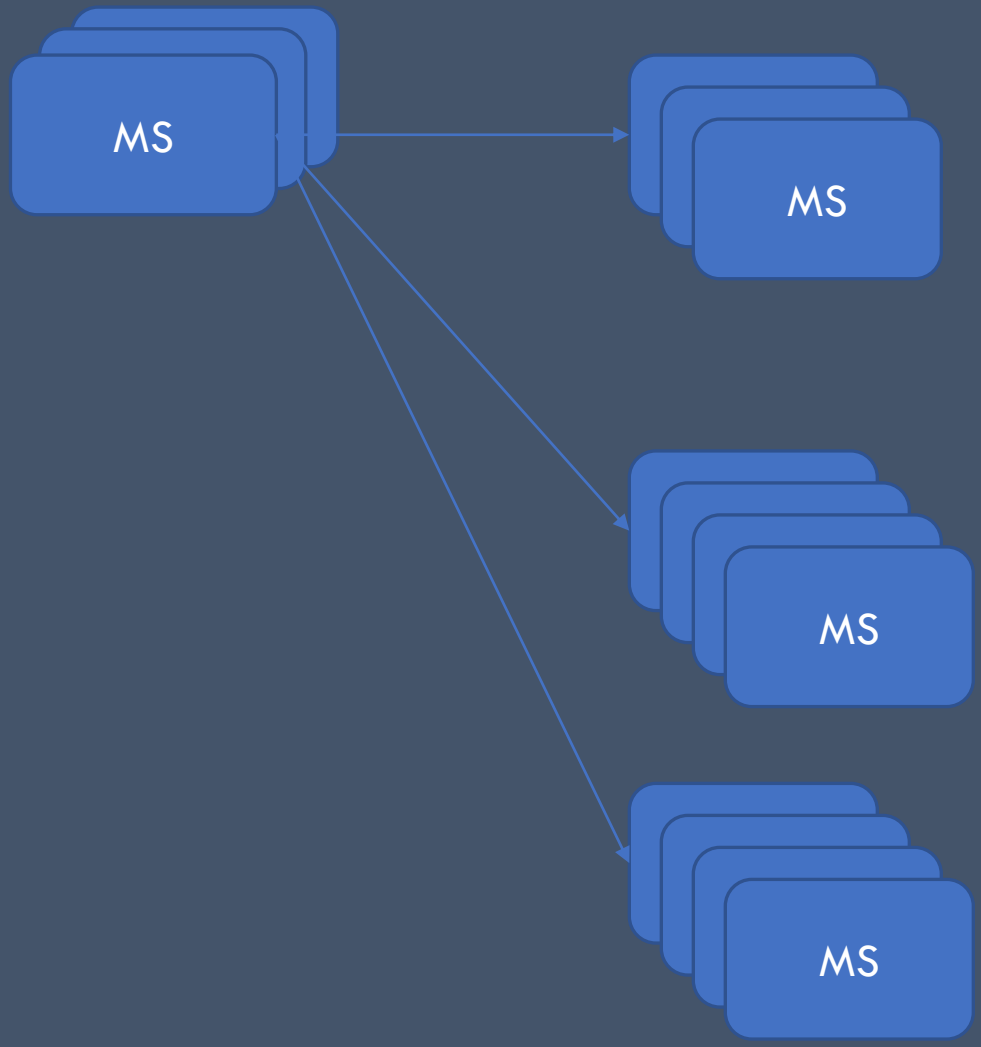

REST
API
Gateway
REST
REST
Order
Service
REST
RestaurantS
ervice
REST
Kitchen
Service
REST
Delivery
Service
REST
Accounting
Service
REST
Notification
Service
SMS
EMAIL


API GATEWAY
DISCOVERY
MS
MS
MS
MS
CLOUD BUS
CONFIGURATION

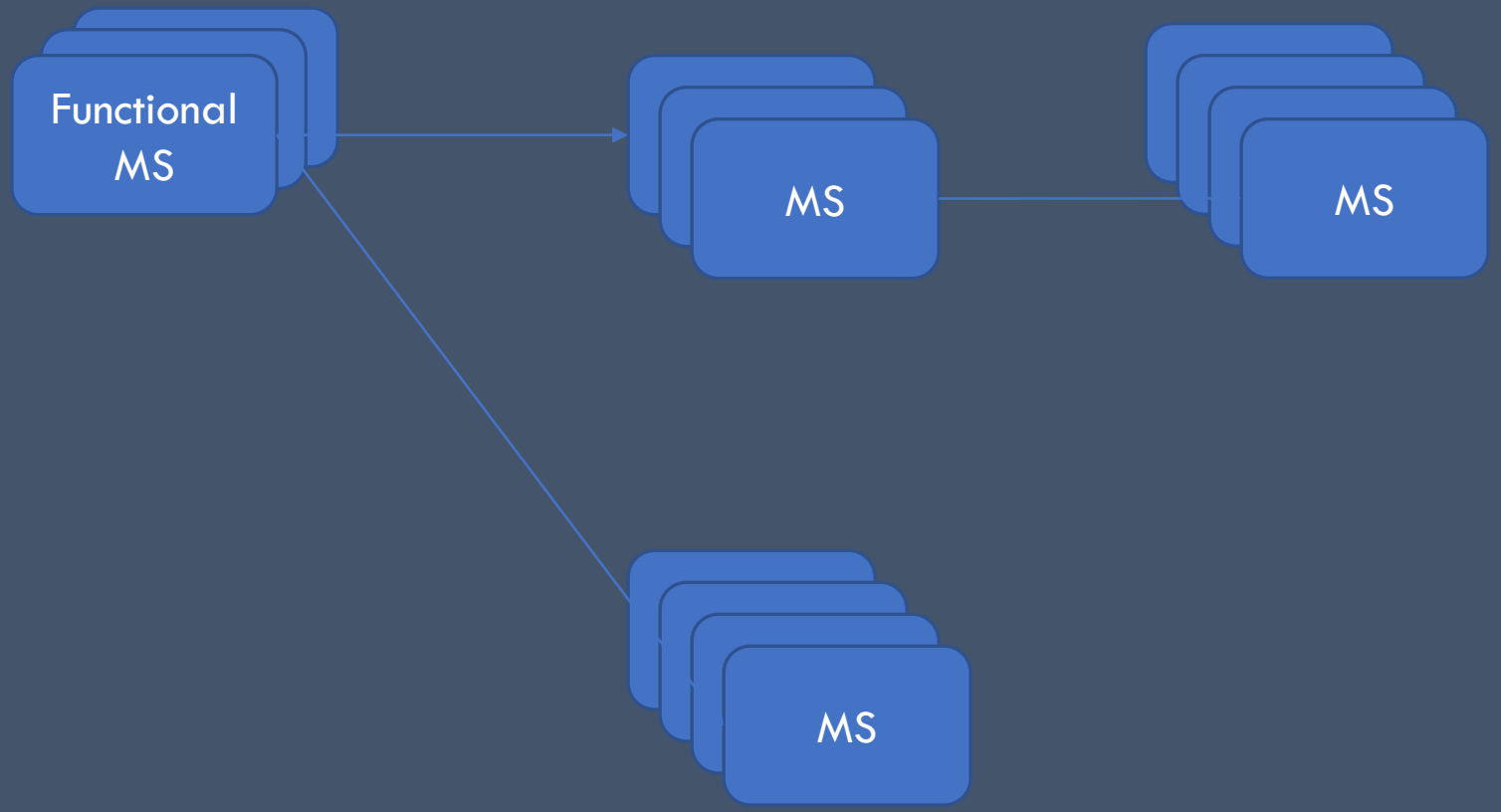
CLIENT SIDE LOADBALANCING



DIRECT PATTERN

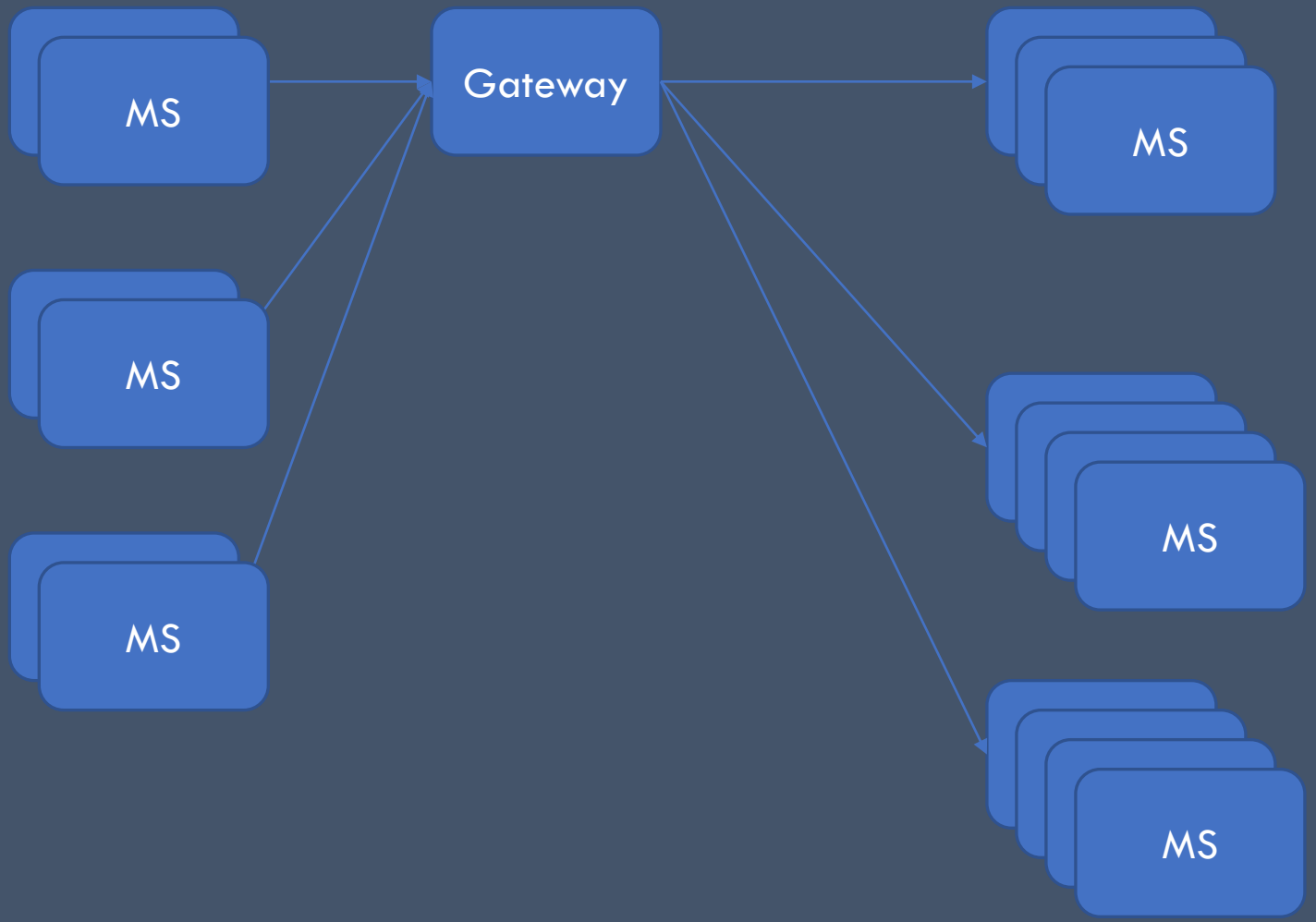


BRANCH PATTERN

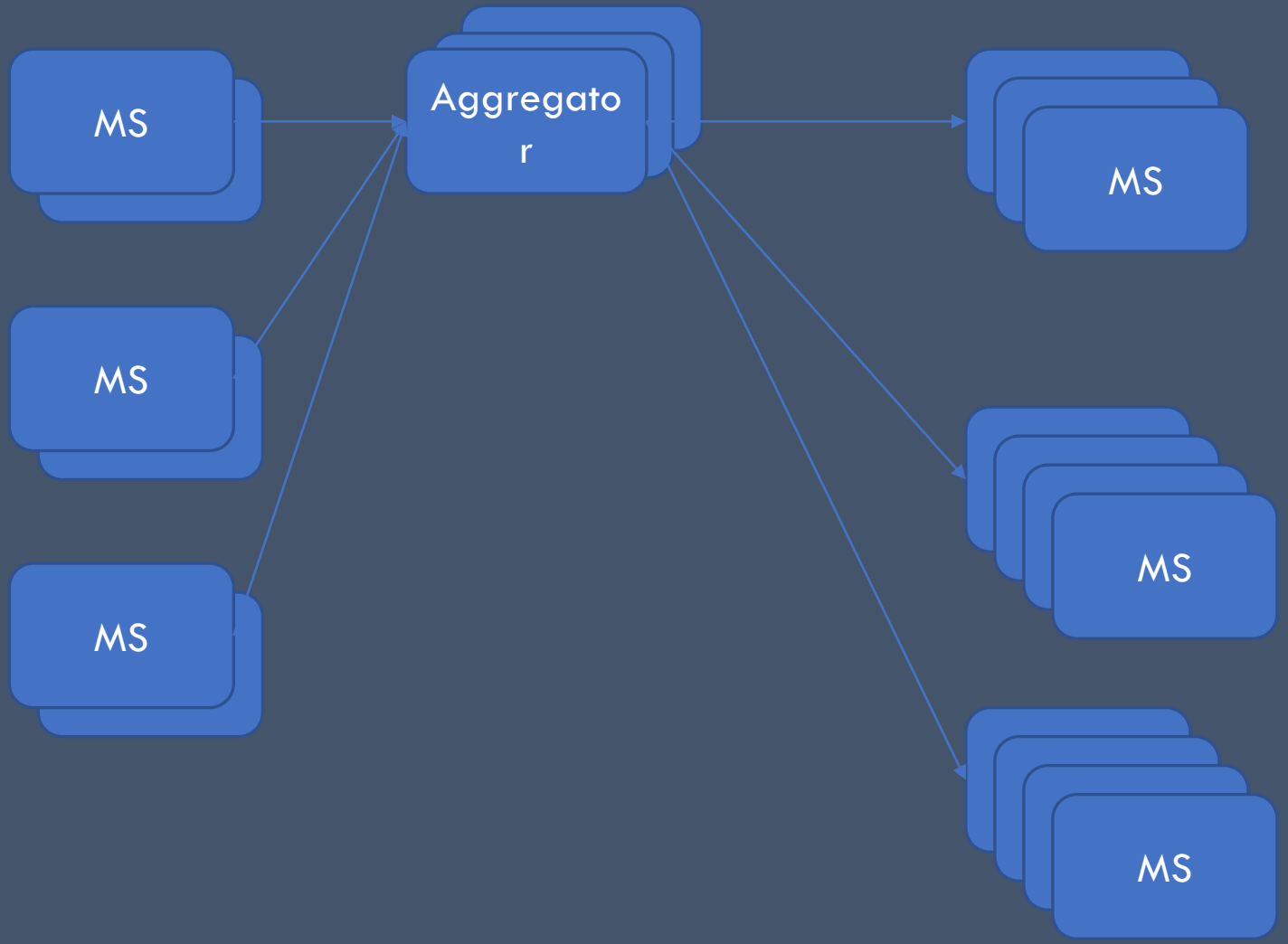


CHAINED PATTERN

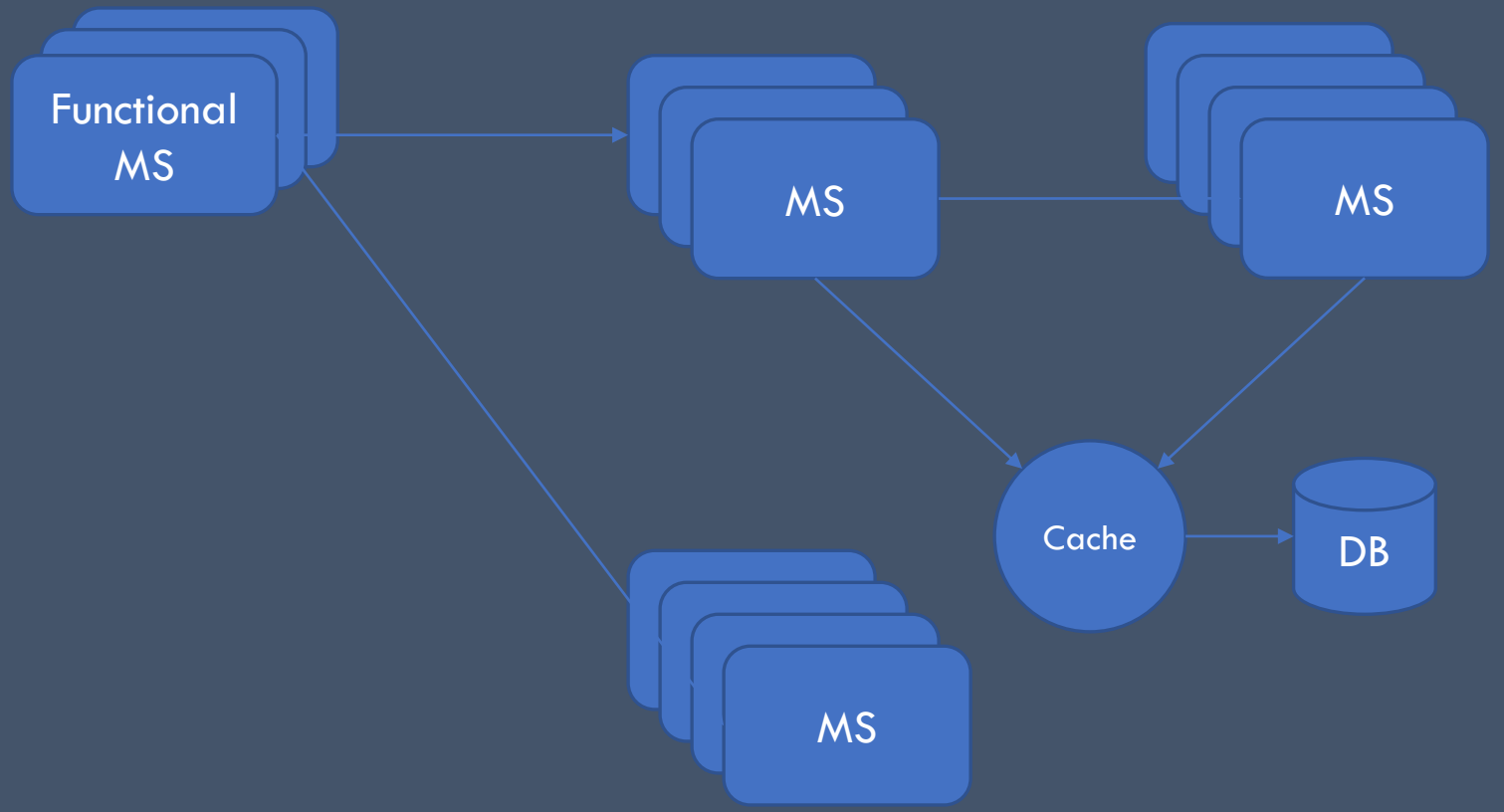
PROXY PATTERN



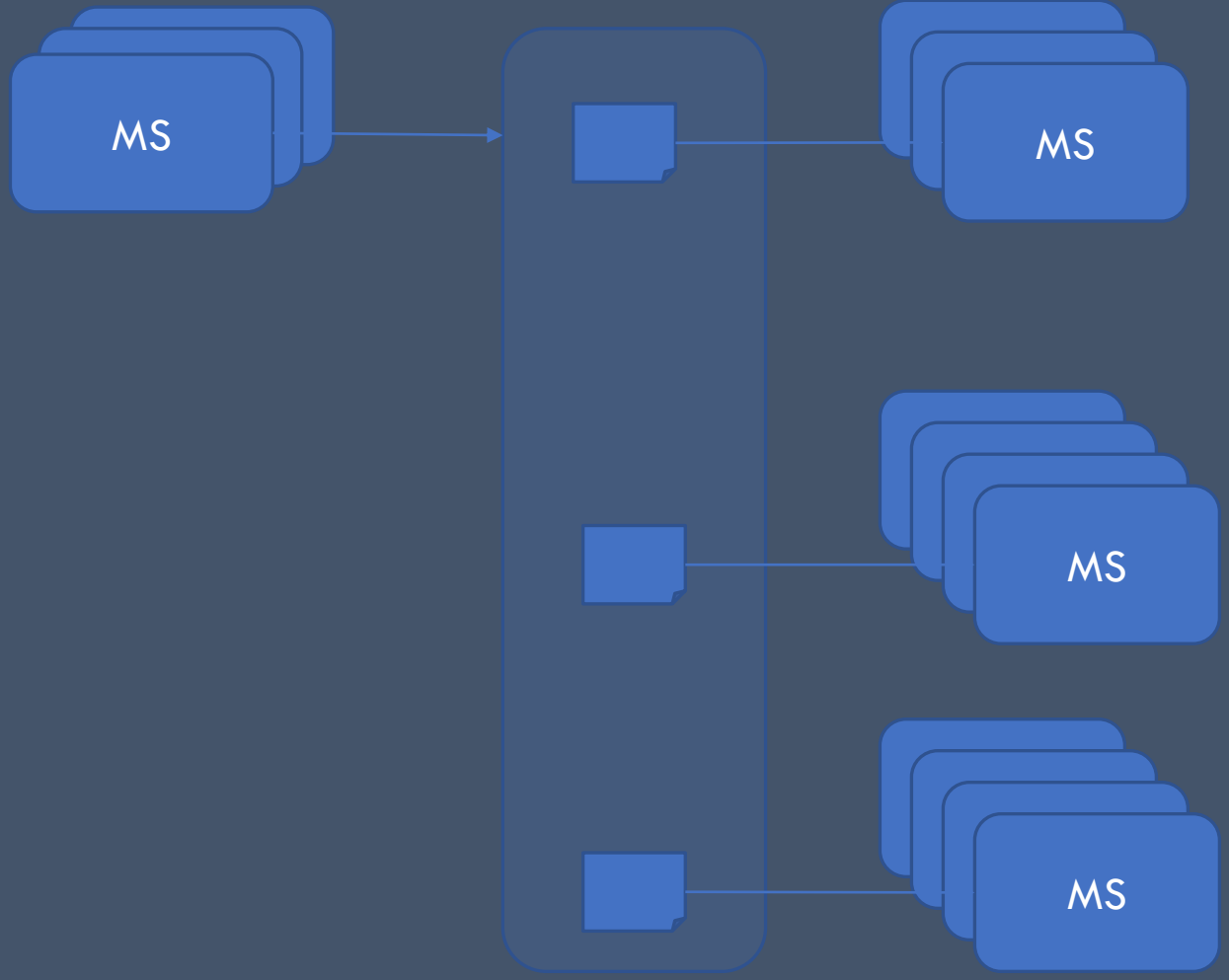
AGGREGATOR PATTERN



SHARED RESOURCE PATTERN

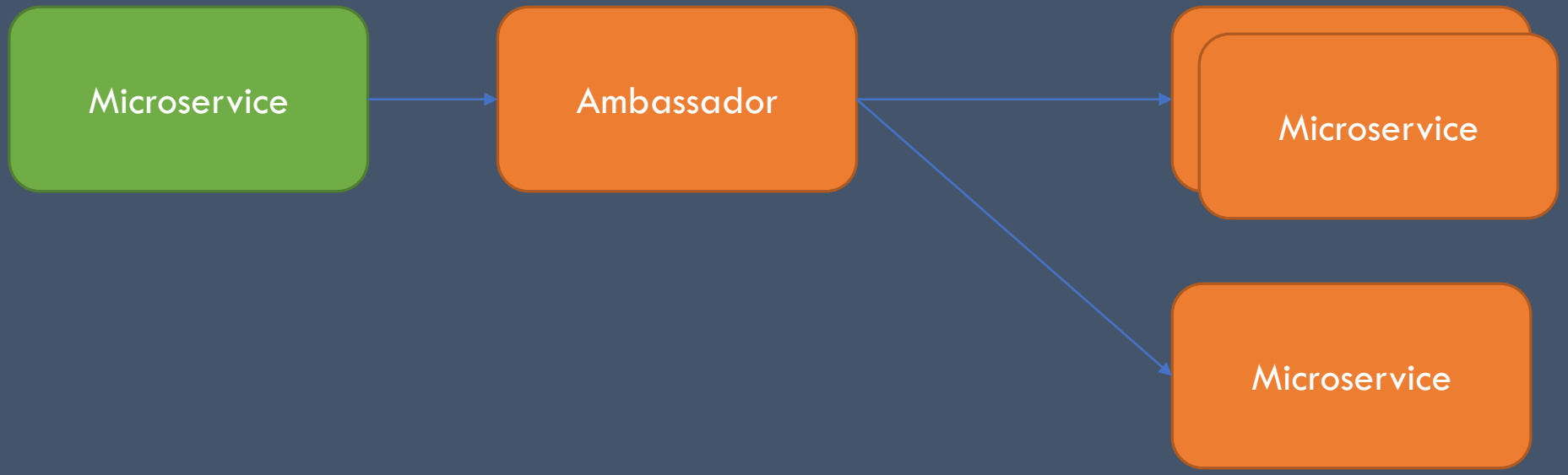


ASYNC PATTERN



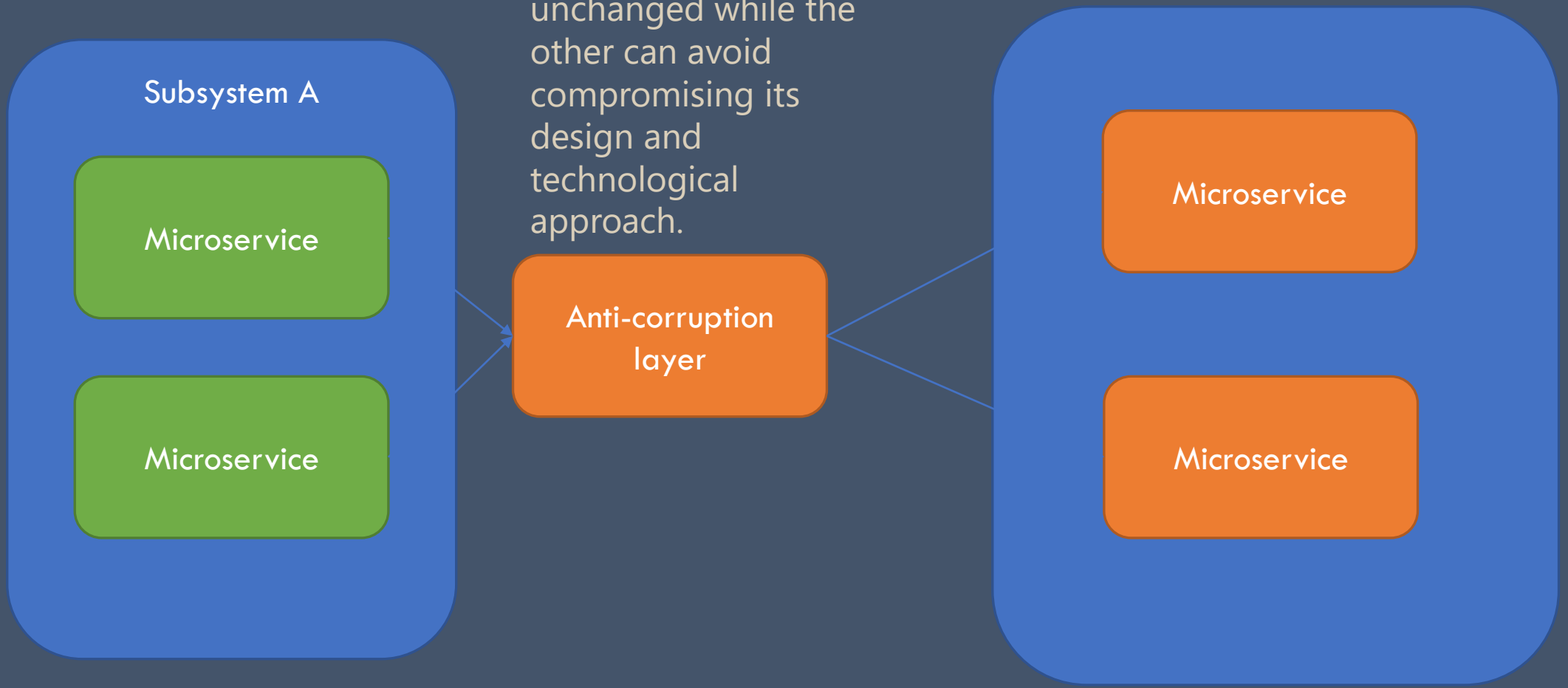
AMBASSADOR PATTERN

- Route
- Check circuit breaker state
- Retry if fails
- Load balancing
- Monitoring and Trace
- Verify and validate security

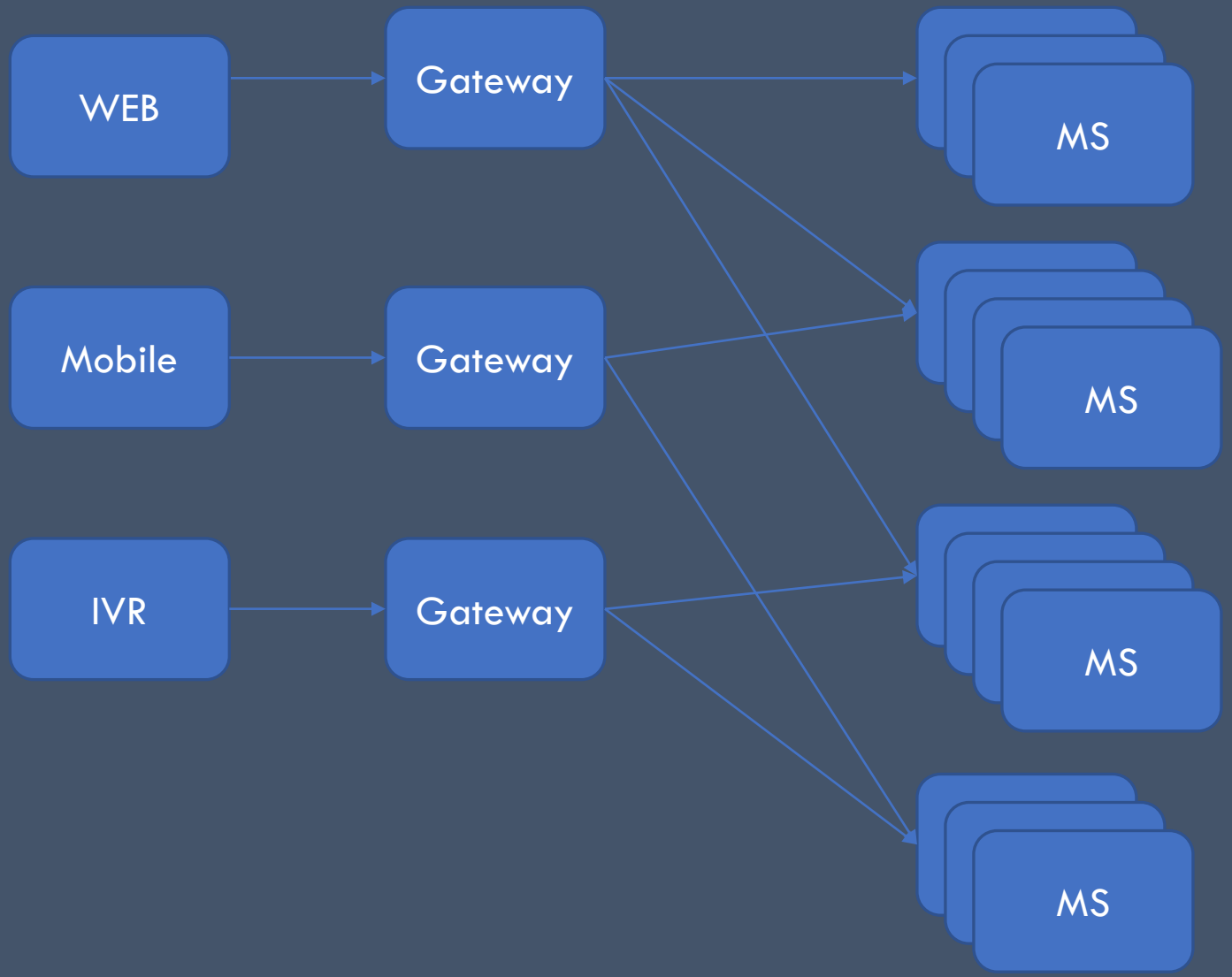


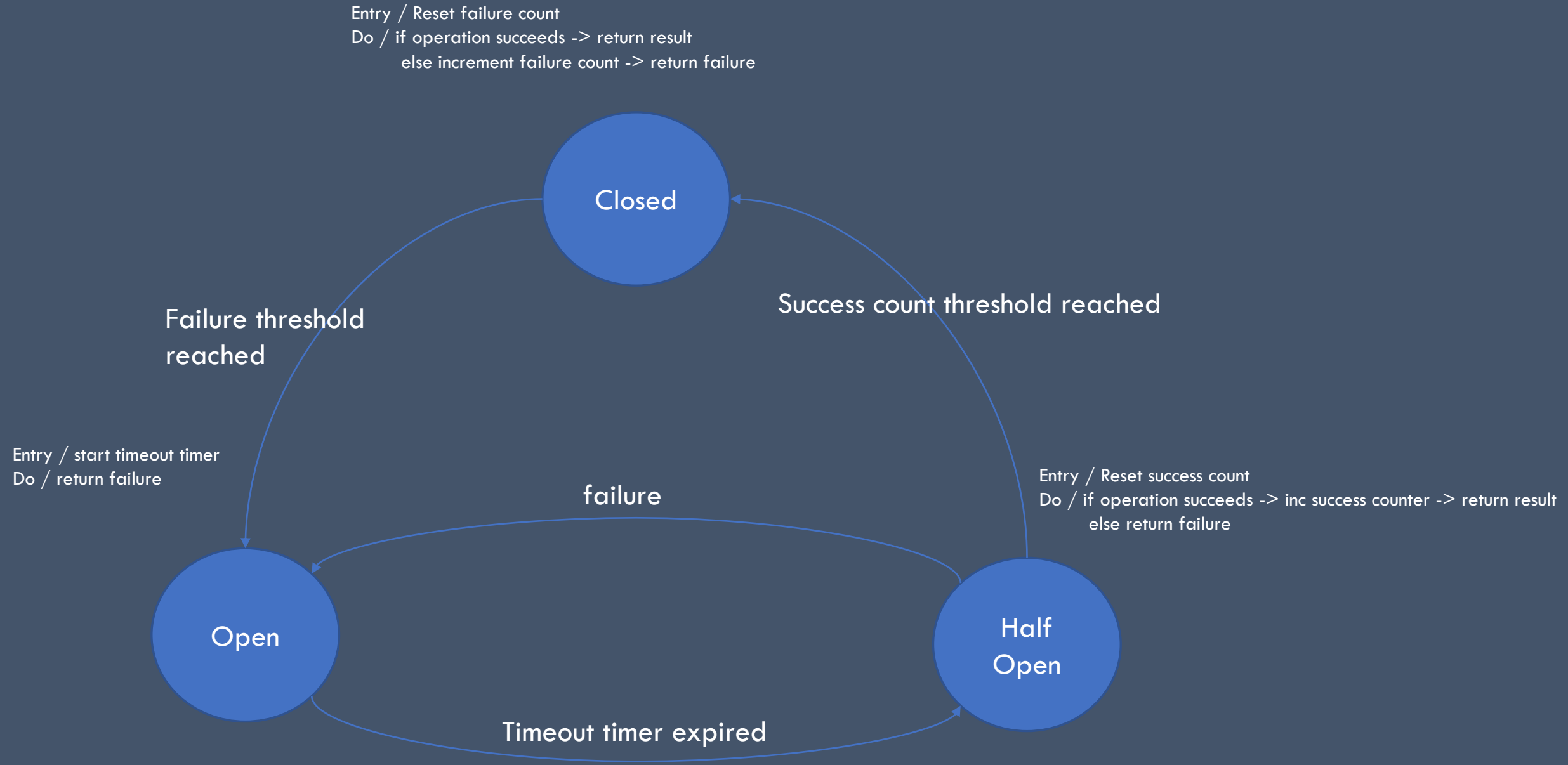
ANTI-CORRUPTION LAYER PATTERN

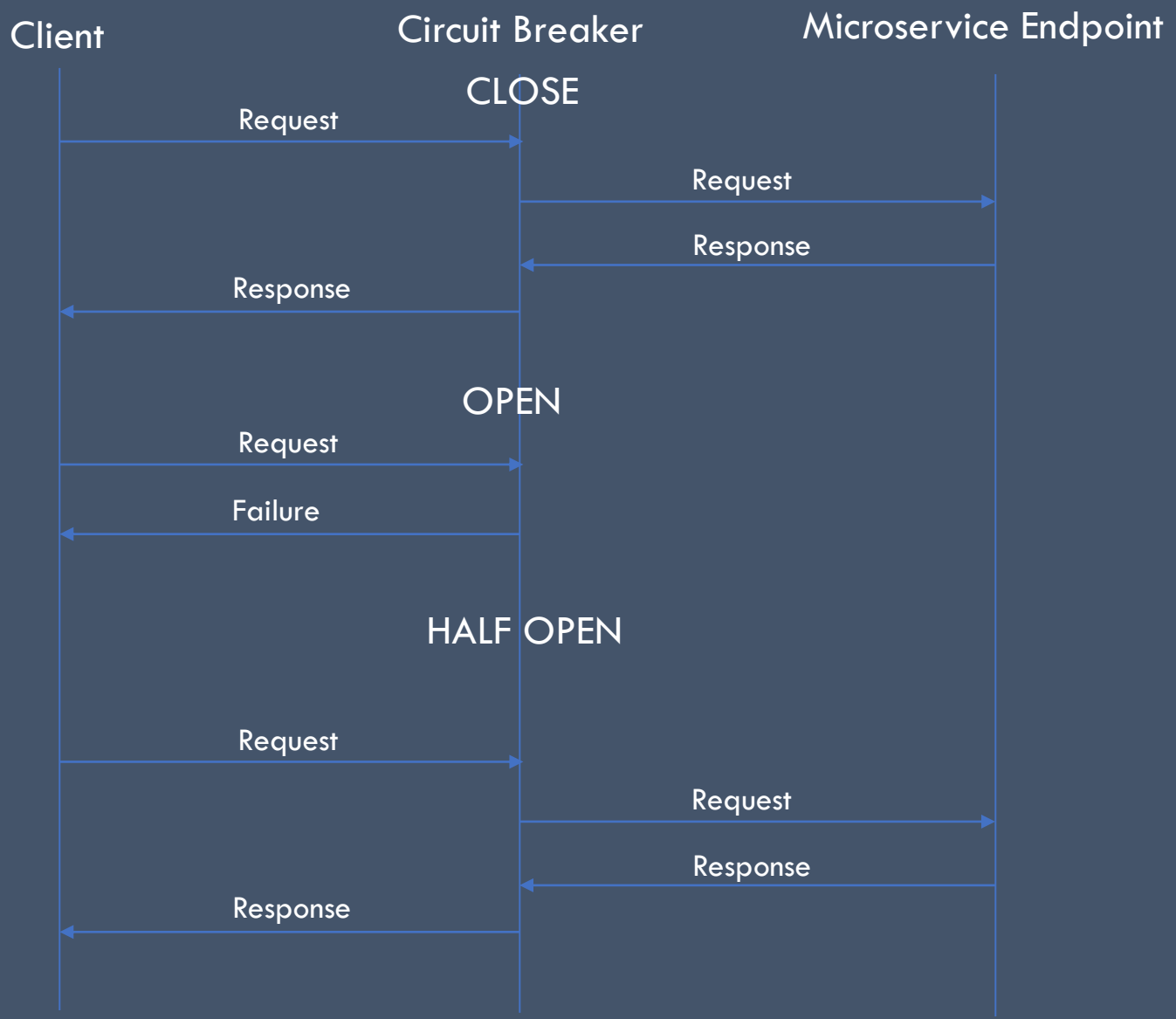
Translates communications between the two systems, allowing one system to remain unchanged while the other can avoid compromising its design and technological approach.

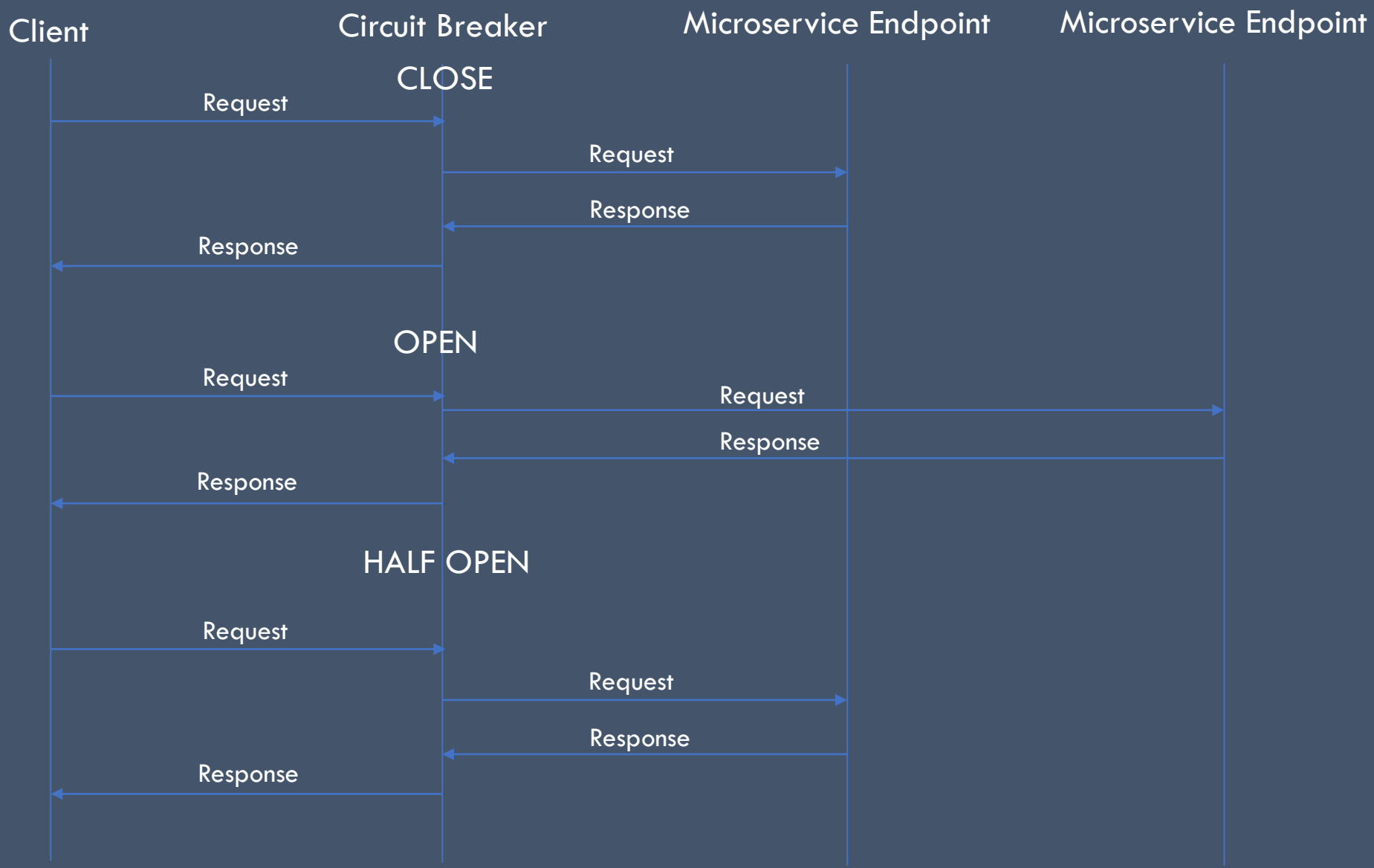


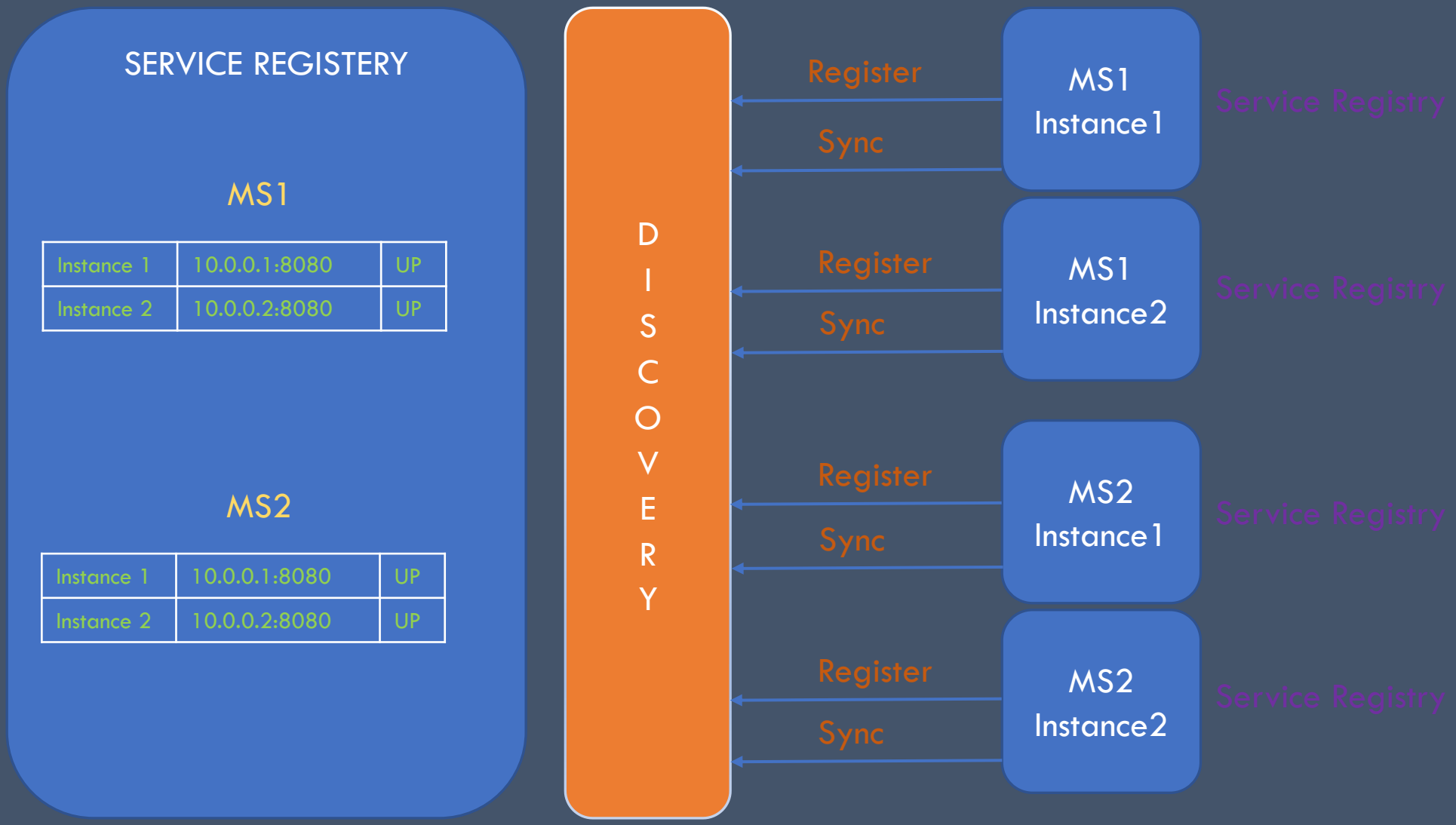
BACKENDS FOR FRONTENDS PATTERN




Entry / Reset failure count
Do / if operation succeeds -> return result
else increment failure count -> return failure
Closed
Failure threshold reached
Success count threshold reached
Entry / start timeout timer
Do / return failure
failure
Entry / Reset success count
Do / if operation succeeds -> inc success counter -> return result
else return failure
Open
Half Open
Timeout timer expired


Client
Circuit Breaker
Microservice Endpoint
CLOSE
Request
Request
Response
Response
OPEN
Request
Failure
HALF OPEN
Request
Request
Response
Response


Client
Circuit Breaker
Microservice Endpoint
Microservice Endpoint
CLOSE
Request
Request
Response
Response
OPEN
Request
Request
Response
Response
HALF OPEN
Request
Request
Response
Response


SERVICE REGISTERY
MS1
Instance 1 10.0.0.1:8080 UP
Instance 2 10.0.0.2:8080 UP
MS2
Instance 1 10.0.0.1:8080 UP
Instance 2 10.0.0.2:8080 UP
DISCOVERY
Register
Sync
Register
Sync
Register
Sync
Register
Sync
MS1 Instance1
MS1 Instance2
MS2 Instance1
MS2 Instance2
Service Registry
Service Registry
Service Registry
Service Registry