Cloud Design Patterns

» Skan.ai - chief Architect

» Ai.robotics - chief Architect

» Genpact - solution Architect

» Welldoc - chief Architect

» Microsoft

» Mercedes

» Siemens

» Honeywell

Mubarak

- Cloud Arch Styles
- Cloud Arch Patterns
- Case studies

Cost
Time to Market
Security
- GCP Cloud pub/sub
Arm based compute
- Locks
- Graph, columnar
- Multi tenant
- Availability

- https://neo4j.com/blog/why-database-query-language-matters/
- https://www.guru99.com/star-snowflake-data-warehousing.html

System

Arch Styles (big pattern)

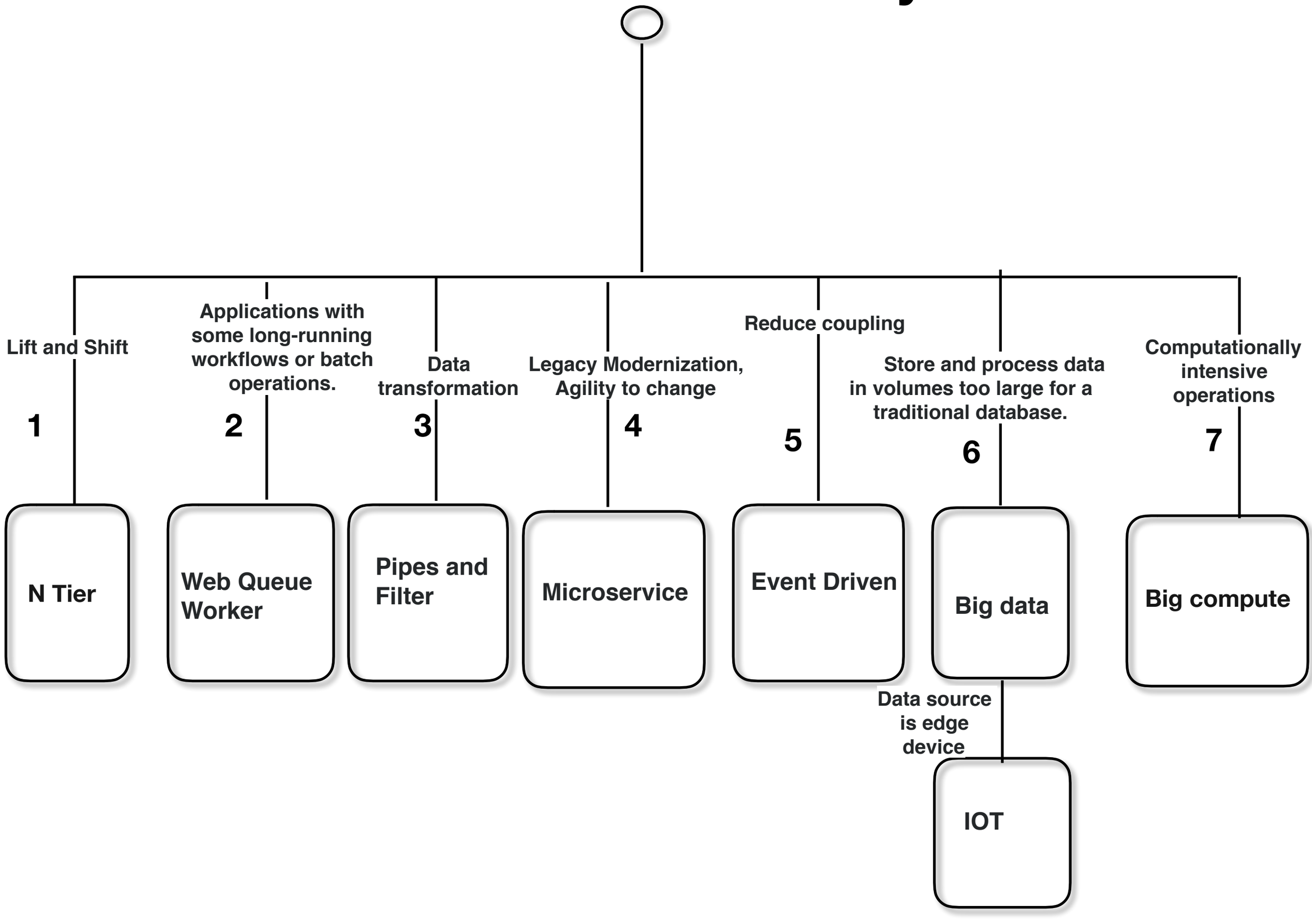
Arch patterns

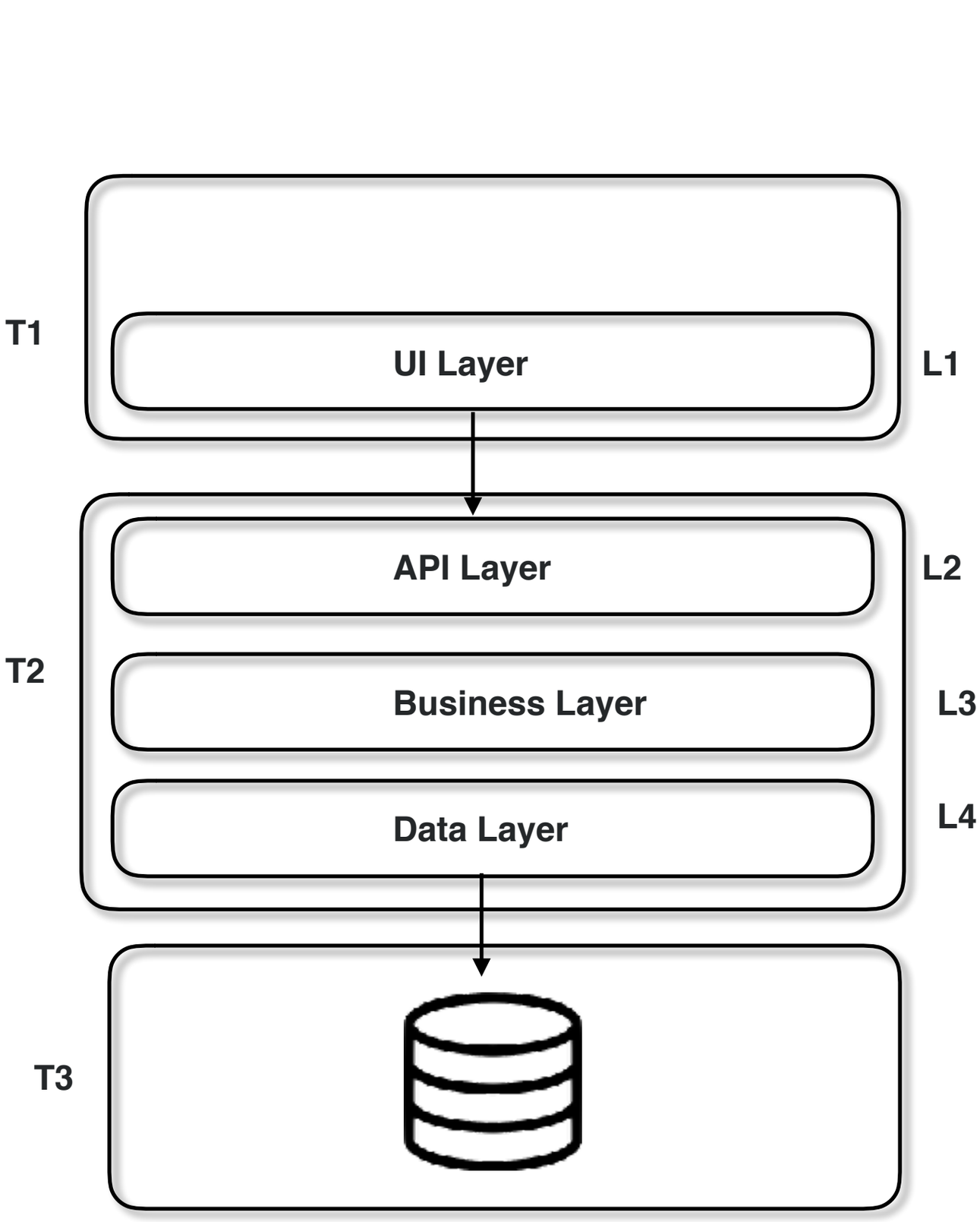
Arch tactics
(small pattern)

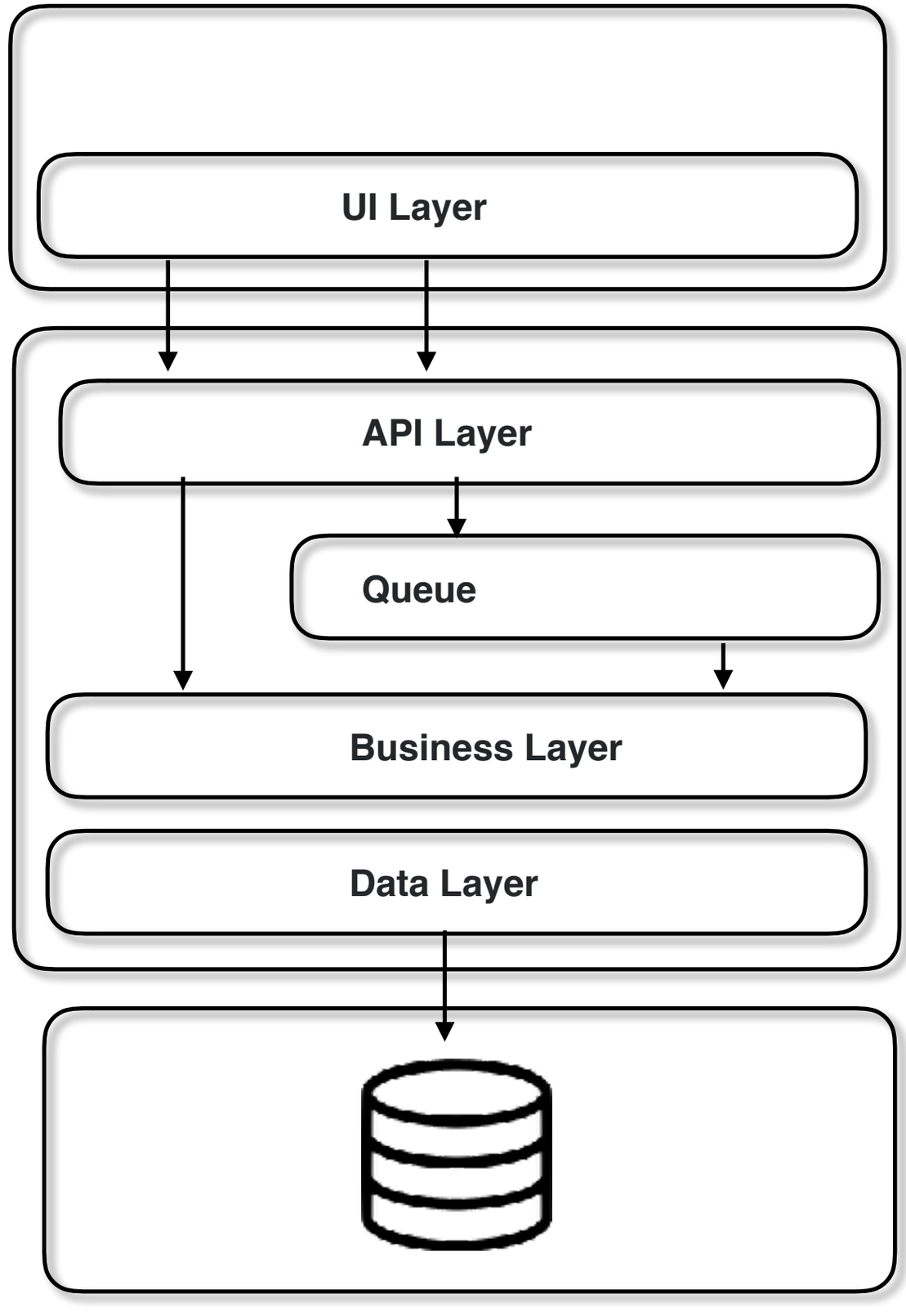
Arch patterns

Cloud Architecture Styles

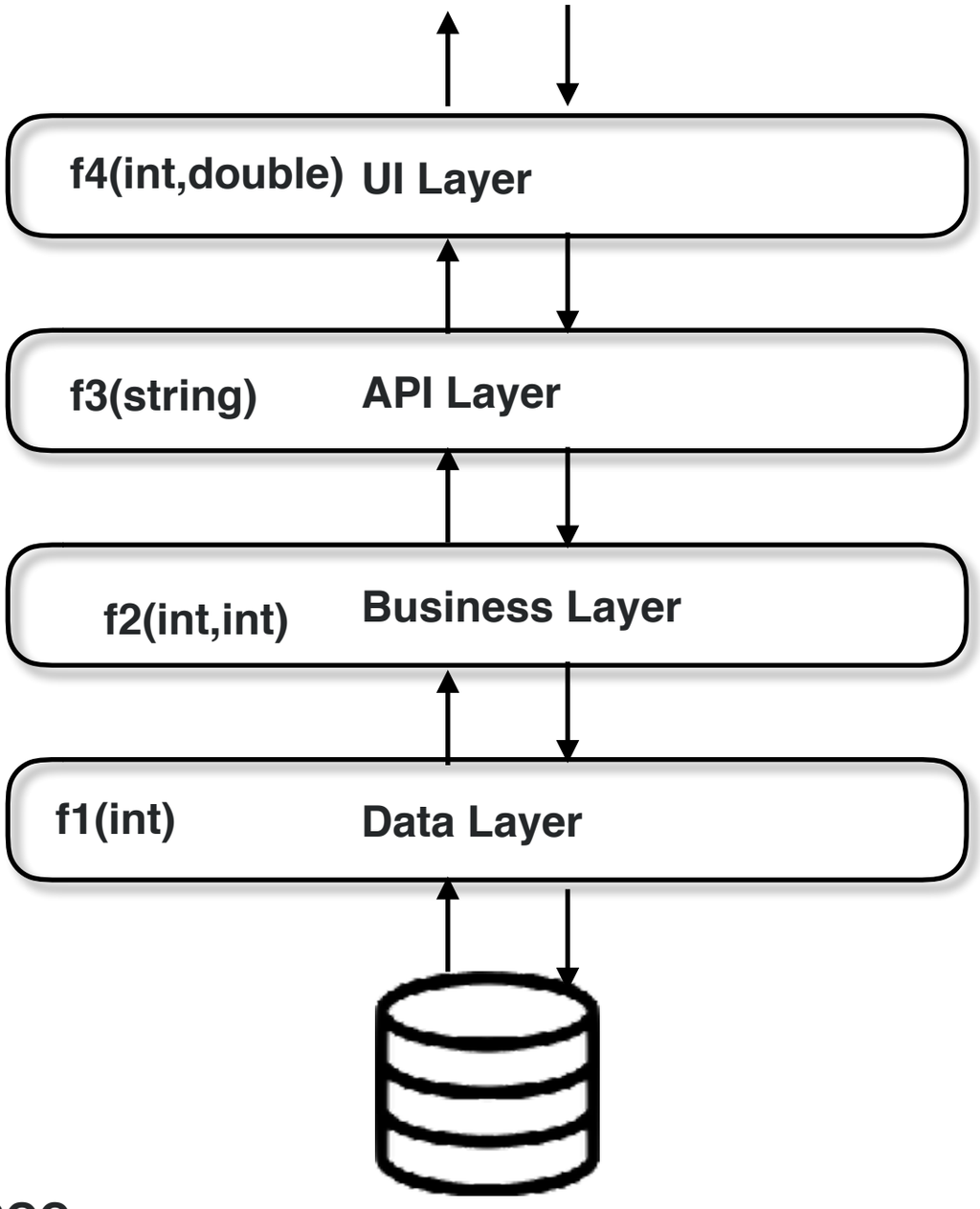
Cloud Architecture Styles



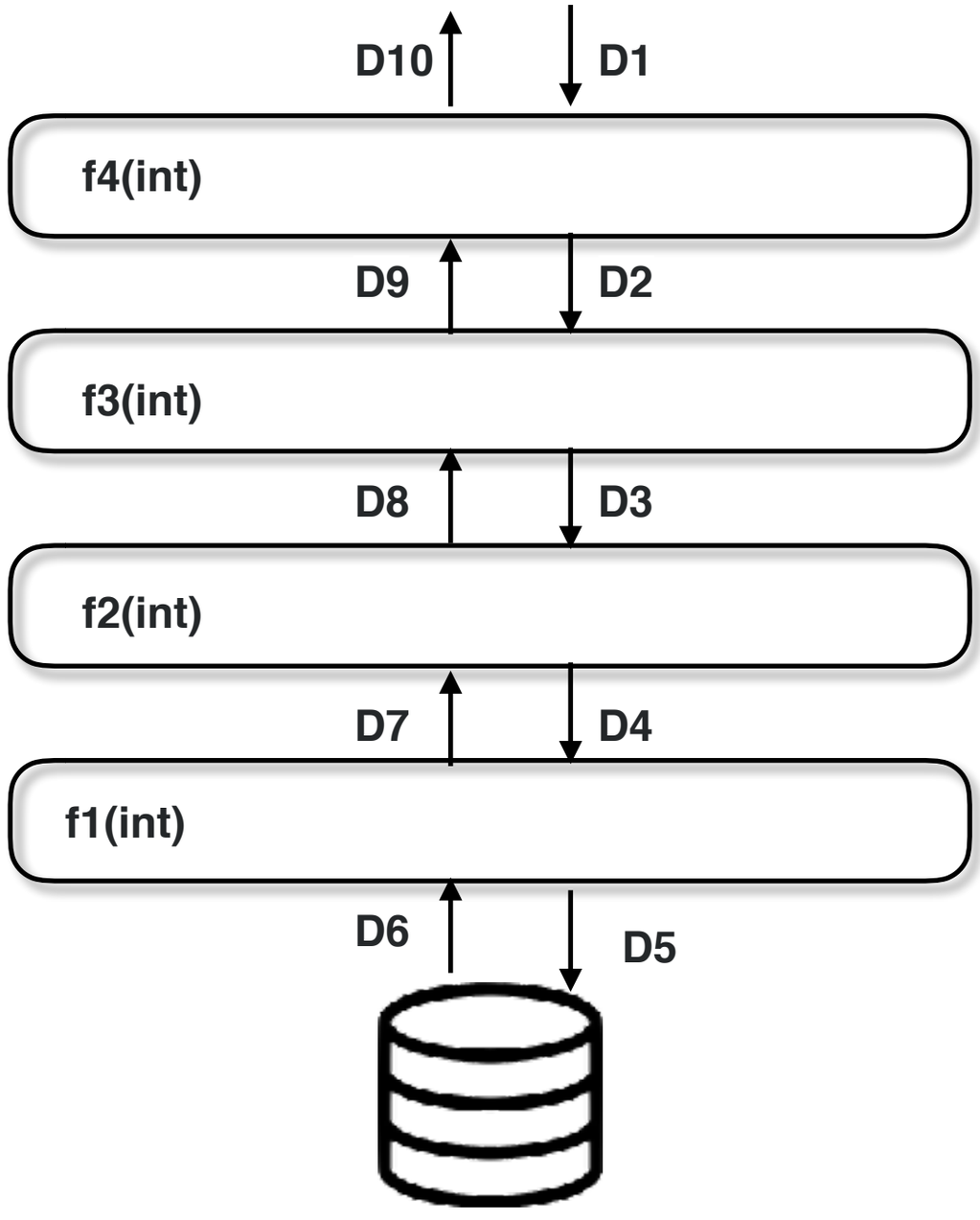




Web socket
Web Hooks
SSE

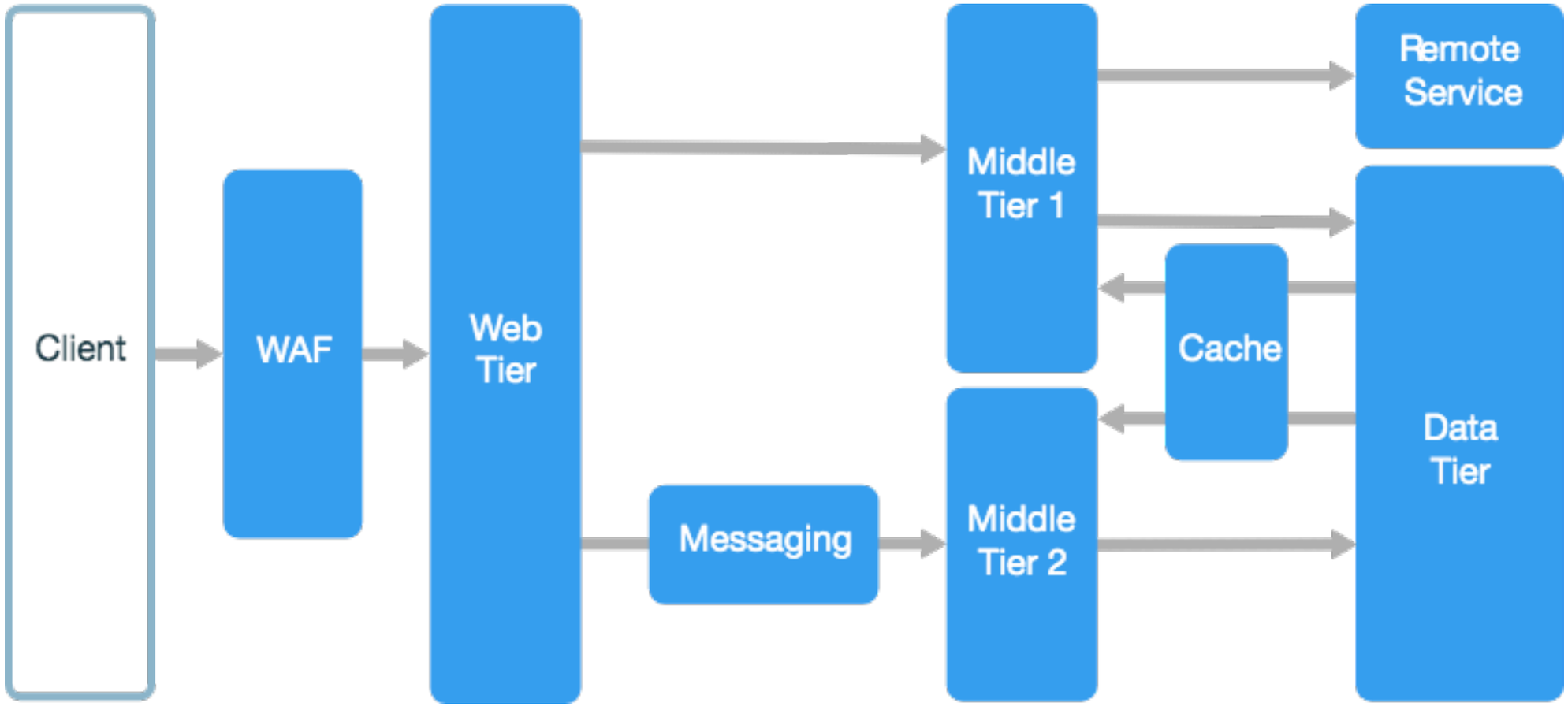


SOC
(each layer expose different contract)

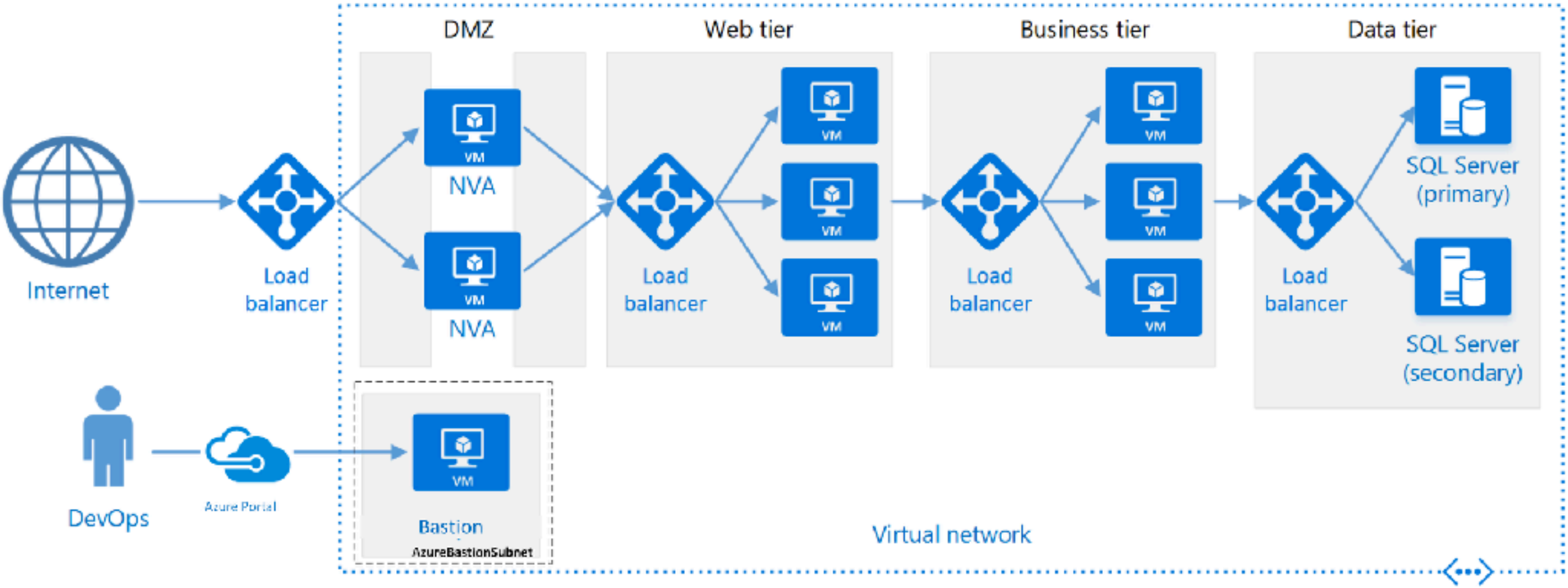


Homogenous
(all filters expose same contract)

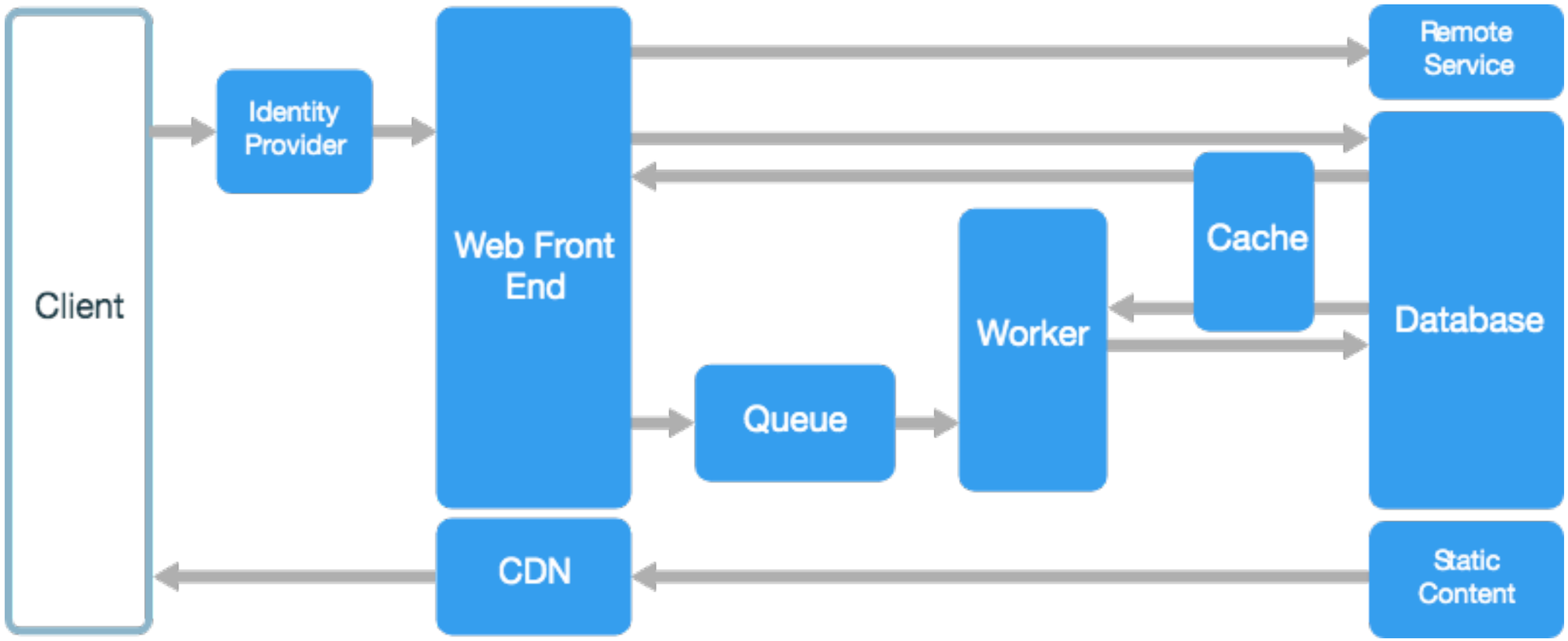
N Tier Reference Architecture



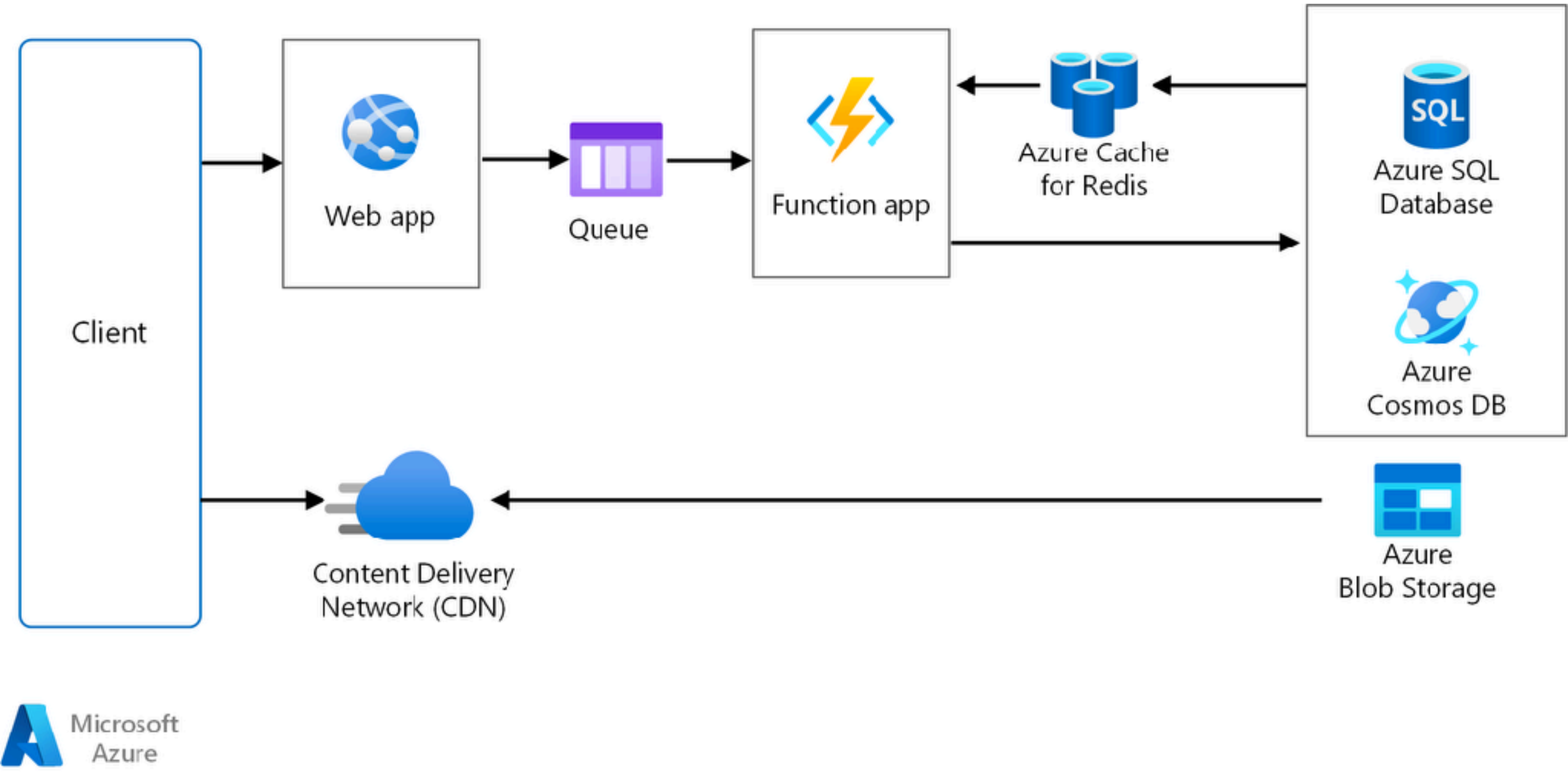
- Migrating an on-premises application to Azure with minimal refactoring.


DMZ
Web tier
Business tier
Data tier
Internet
Load balancer
VM
NVA
VM
NVA
Load balancer
VM
VM
VM
Load balancer
VM
VM
VM
Load balancer
SQL Server (primary)
SQL Server (secondary)
DevOps
Azure Portal
VM
Bastion
AzureBastionSubnet
Virtual network

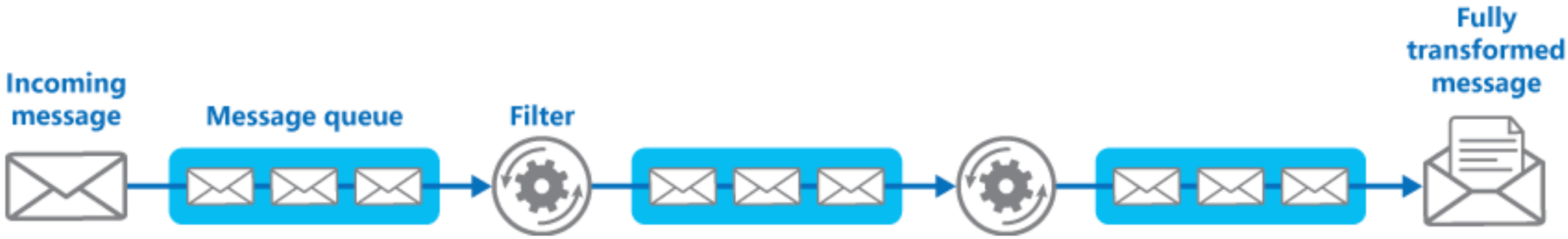
Web Q Worker

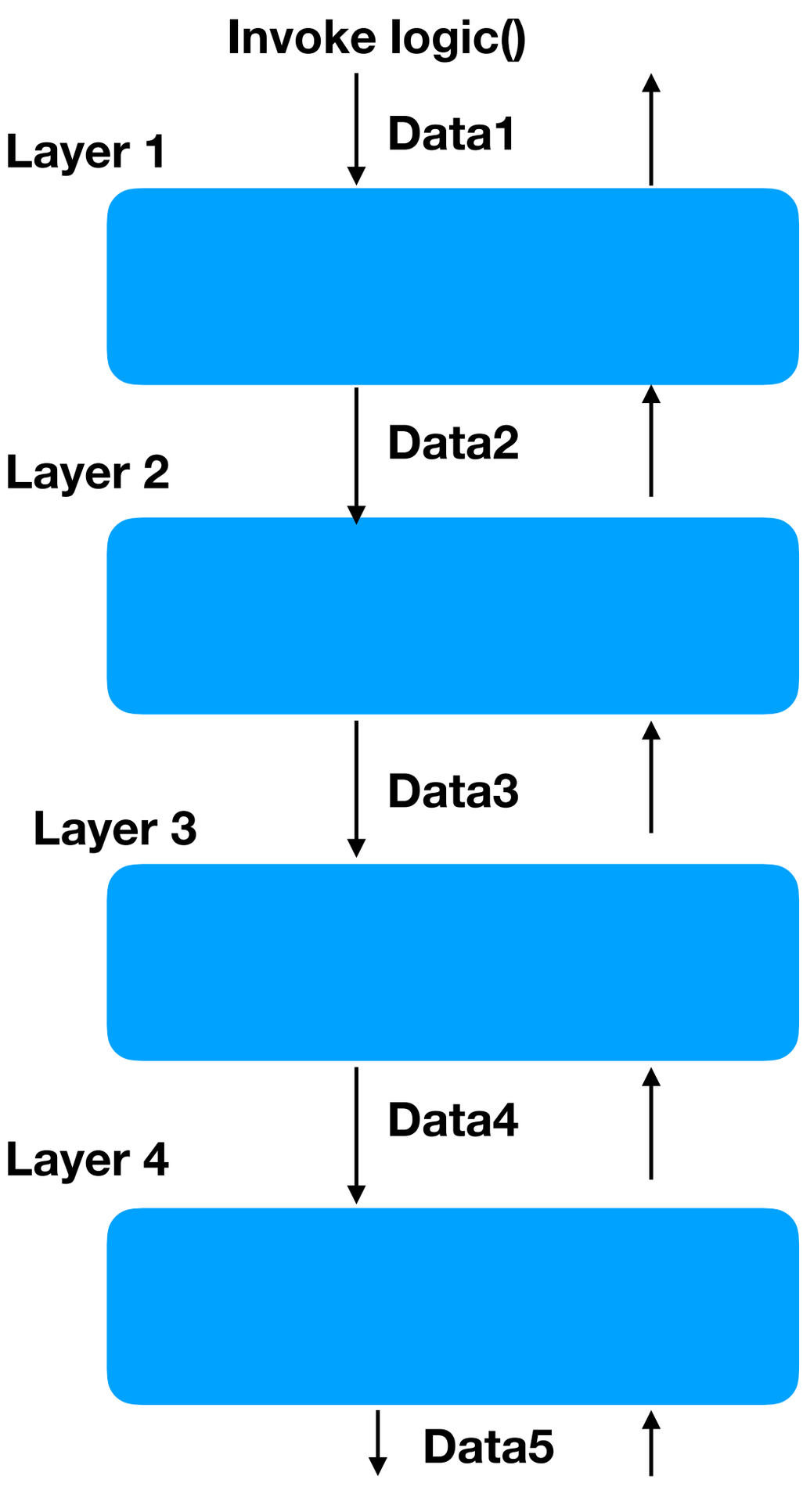


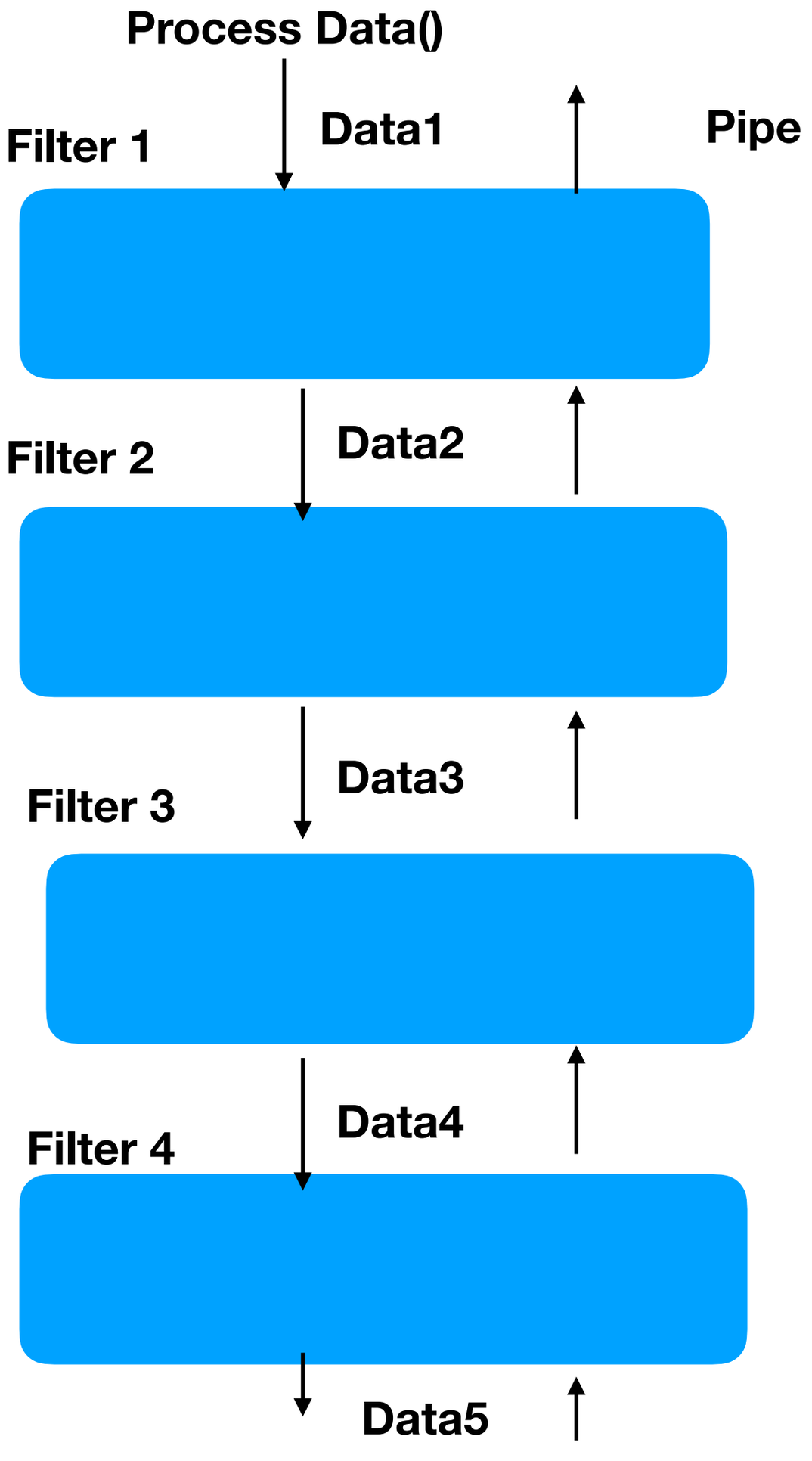
- Applications with some long-running workflows or batch operations.


Client
Web app
Queue
Function app
Azure Cache
for Redis
Azure SQL
Database
Azure
Cosmos DB
Content Delivery
Network (CDN)
Azure
Blob Storage
Microsoft
Azure

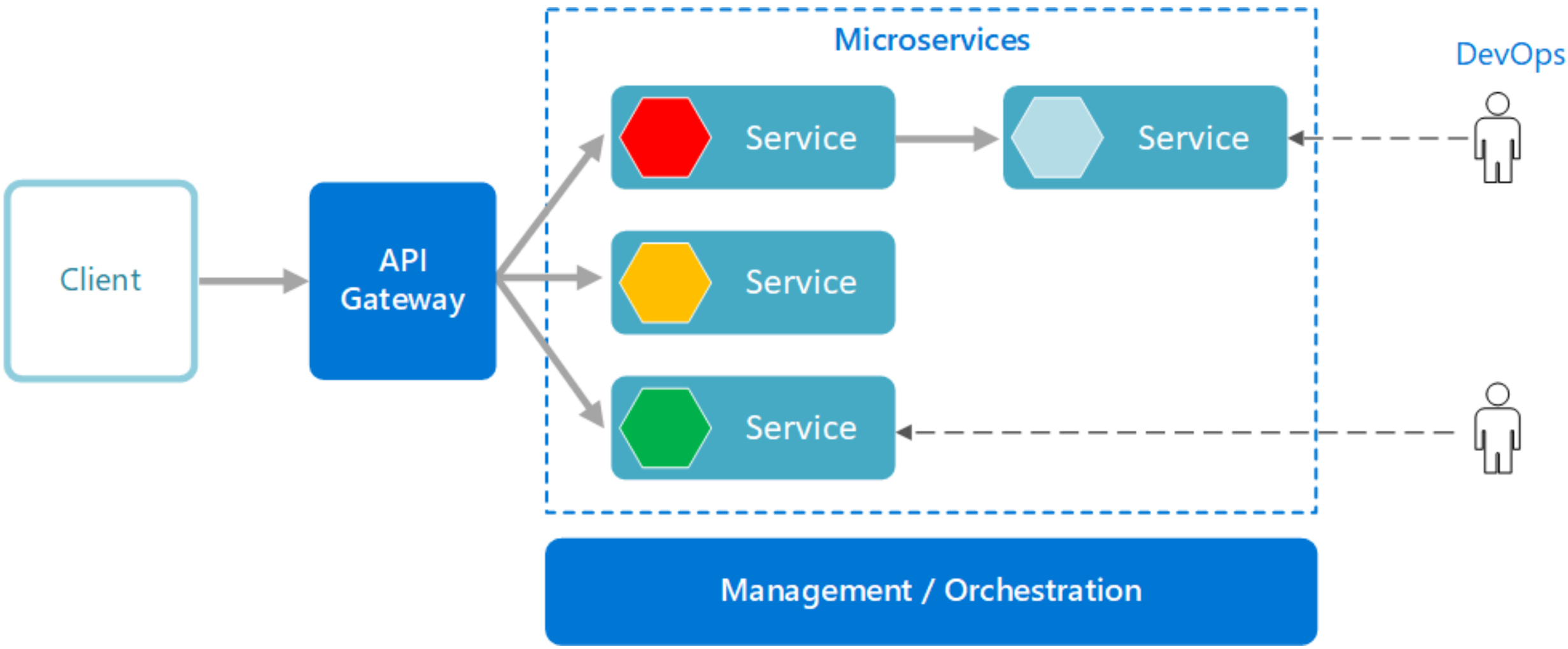
Pipe and Filter




Invoke logic()
Layer 1
Data1
Data2
Layer 2
Data3
Layer 3
Data4
Layer 4
Data5

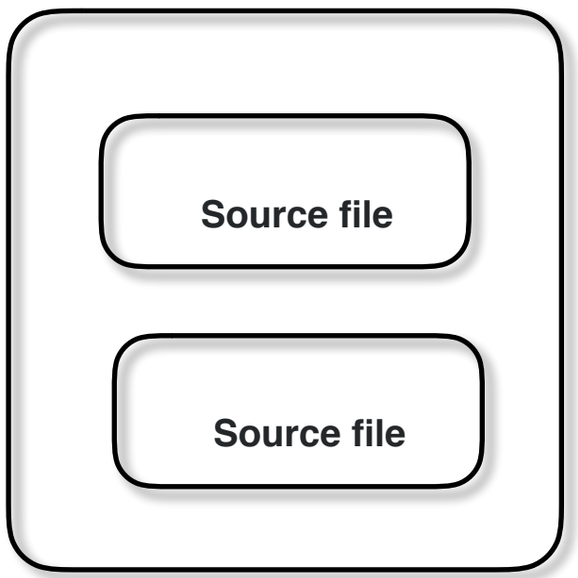

Process Data()
Filter 1
Data1
Pipe
Filter 2
Data2
Filter 3
Data3
Filter 4
Data4
Data5

Microservice



1

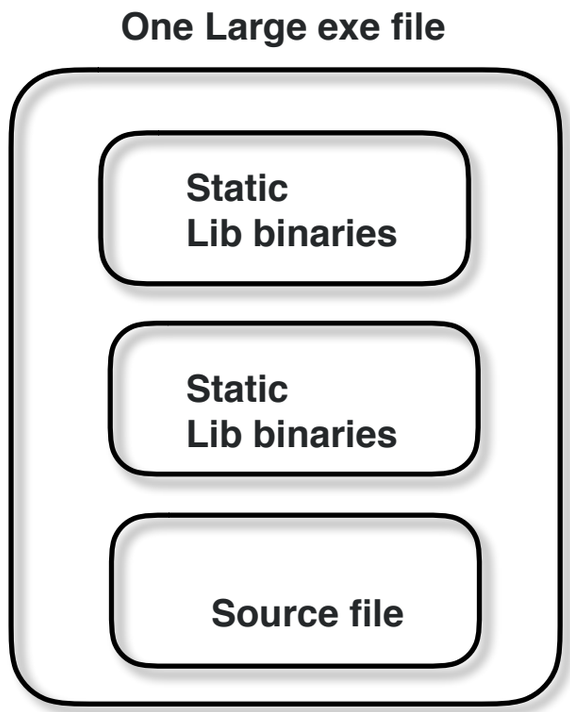
Ship one binary



Compile time monolithic

2

Ship one binary

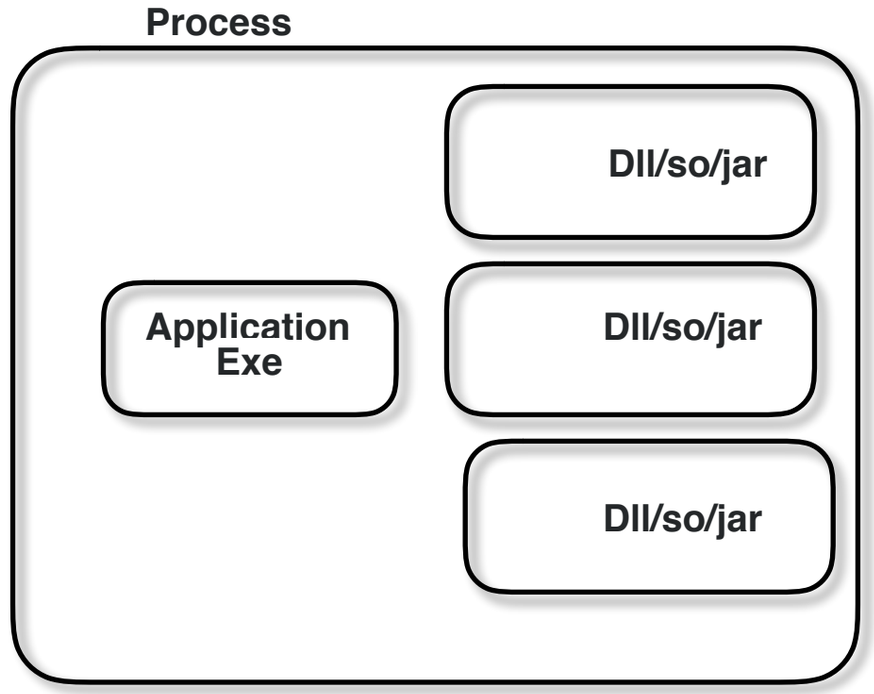


Link time monolithic

3

Ship multiple binary

COM



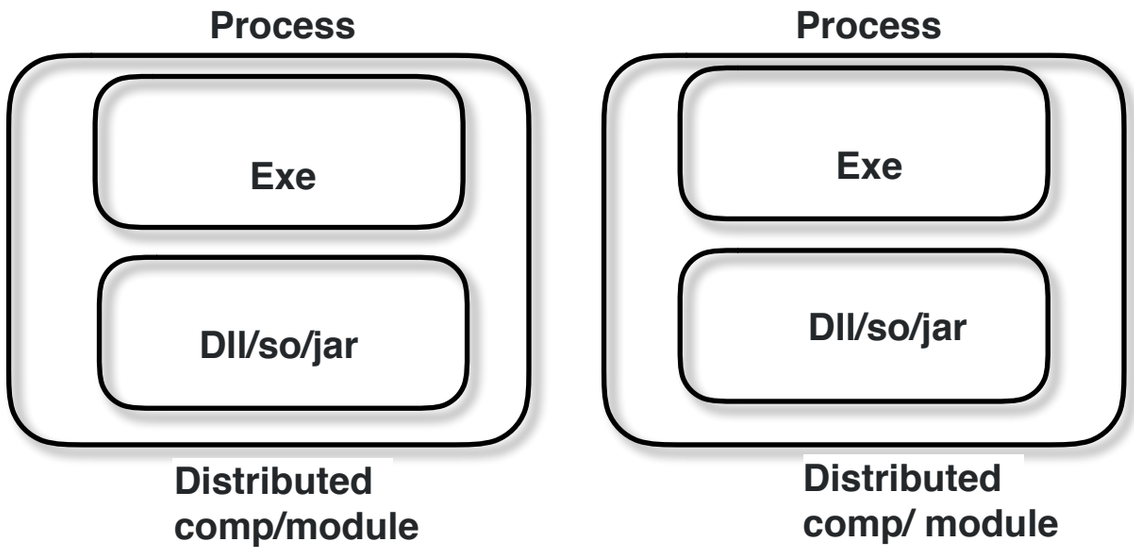
Runtime monolithic

In process comp

Ship multiple binary & host in multiple process

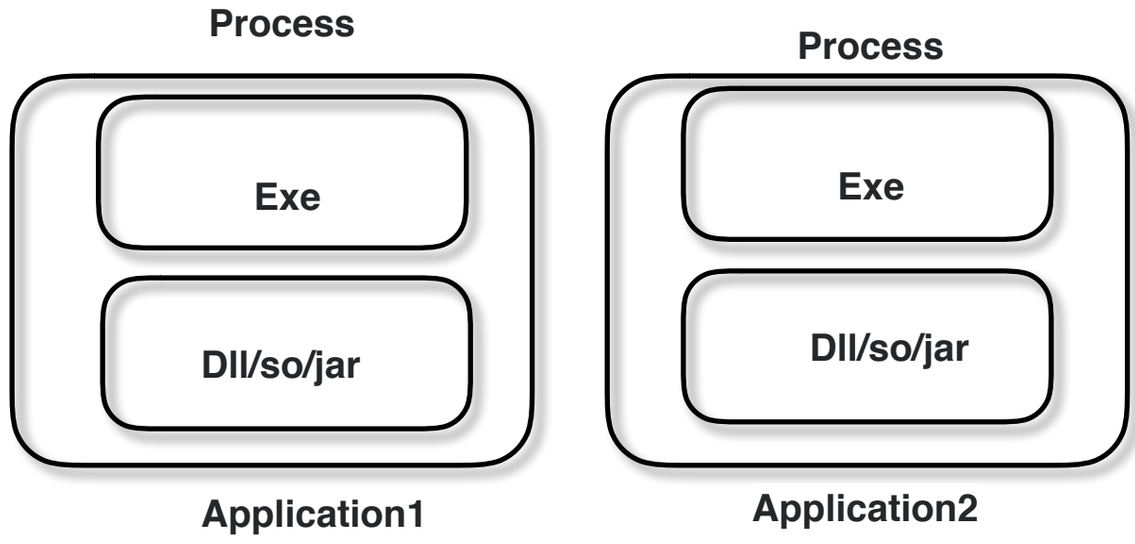
4

DCOM
CORBA
RMI
EJB
…



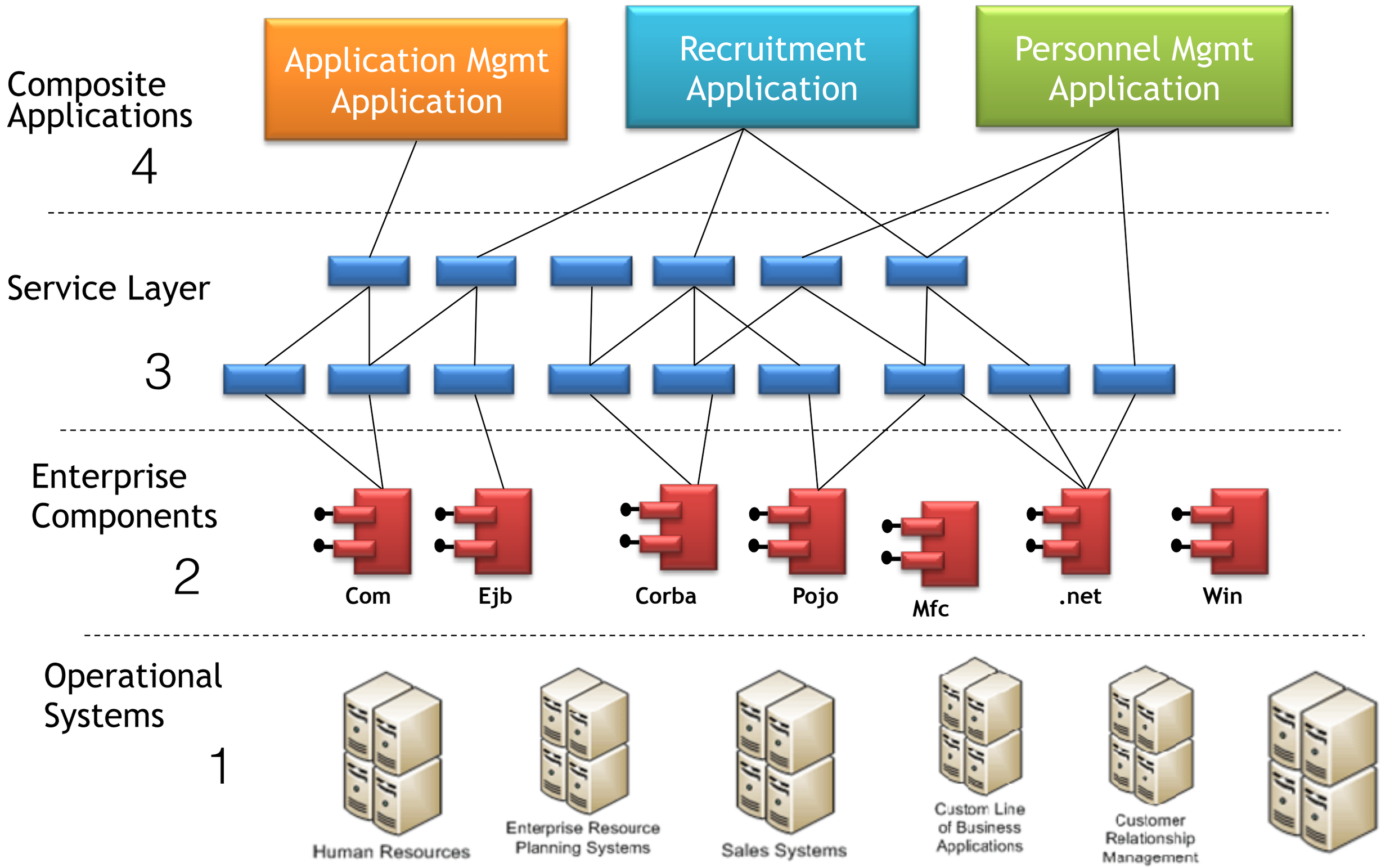
Distributed Application

5



Micro Application

Service Oriented Architecture

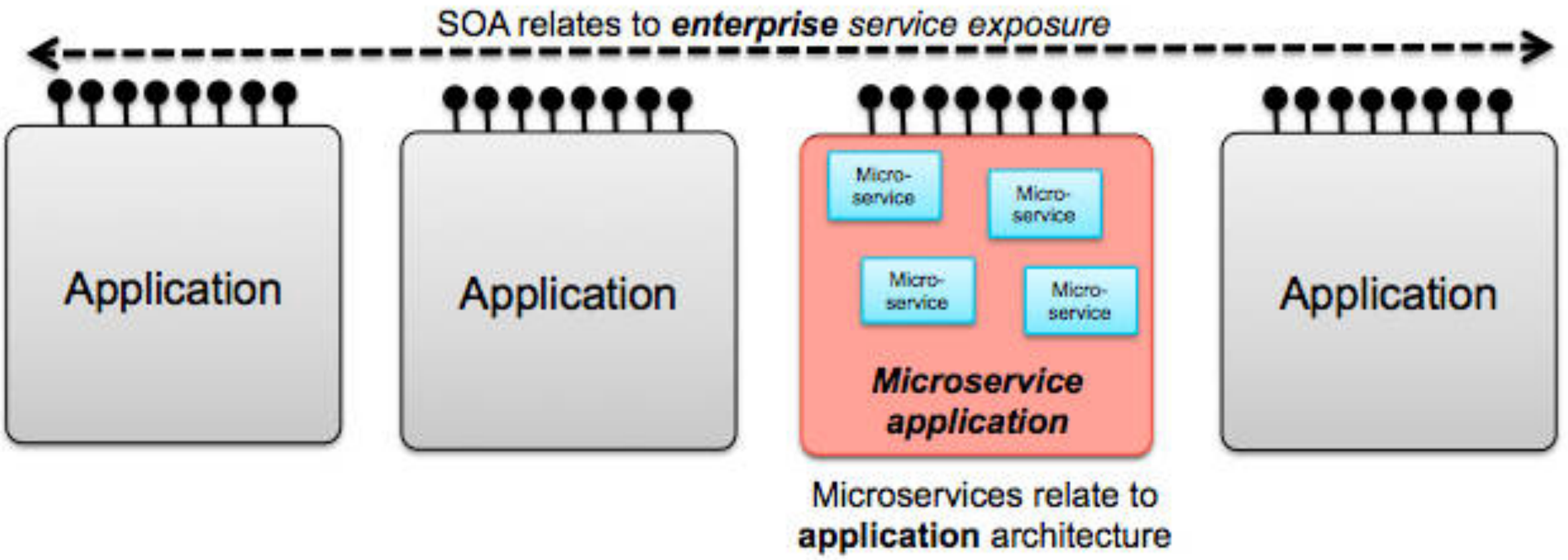


ToDo Portal
(Single Page Application)

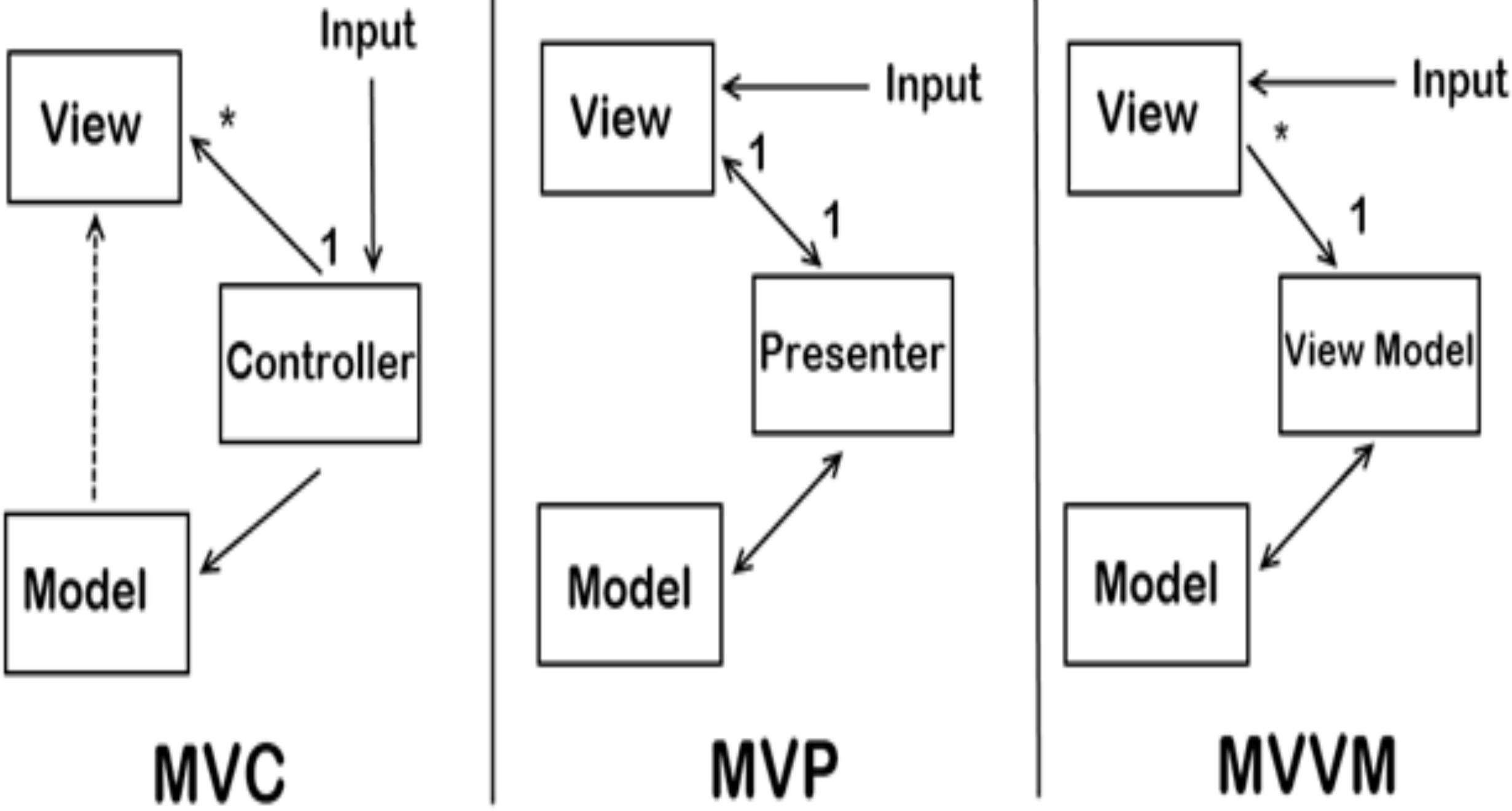
ToDo API Service
(Api Application)

ToDo Calendar Service
(Background Application)

ToDo Prediction Service
(Background Application)



Microservices relate to **application** architecture


View
Input
*
1
Controller
Model
MVC
View
Input
1
1
Presenter
Model
MVP
View
Input
*
1
View Model
Model
MVVM

	Runtime Mono (traiditonal)	Microservice
Developer Effort/ Less time		
Performance (response time)		
Security		
Infra Cost		

Break an app into smaller manageable piece

	Mod1 I Mod2	**App1 I App2**	**W**	**Score**
Share Database / Storage	Yes	No	2	
Share same Virtual Infra	Yes	No	3	
Share Sorce Control	Yes	No	2	
Share CI/CD (Build pipeline)	Yes	No	3	
Share domain code /logic	No	No No No	3	
Fun Requirements	Application level	It owns	1	
SCRUM Team / Sprint	Application Scope	Its owns	1	
Acceptance Test Cases	Application Scope	Its owns	1	
Architecture	Application Scope	Its owns	1	
Technology Stack / Fwks	Application Scope	Its owns	1	

Application

Modules

Modules

Modules

3 or 4

Application

Small App

Small App

Small App

Modules

Modules

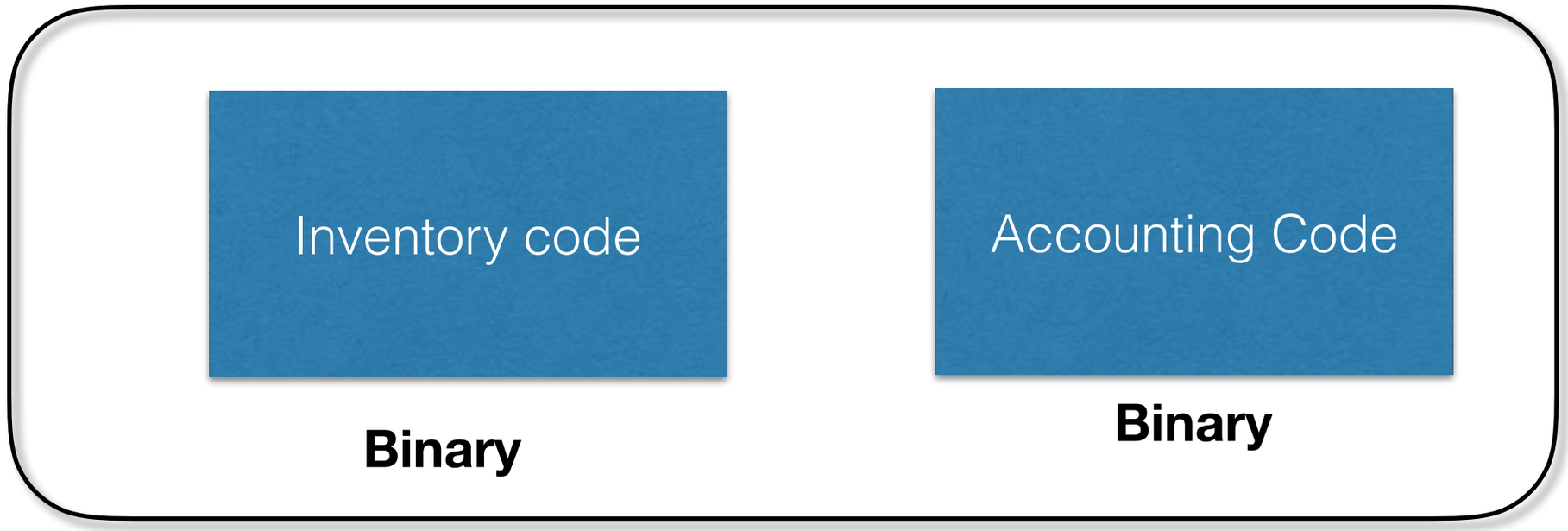
Modules

Modules

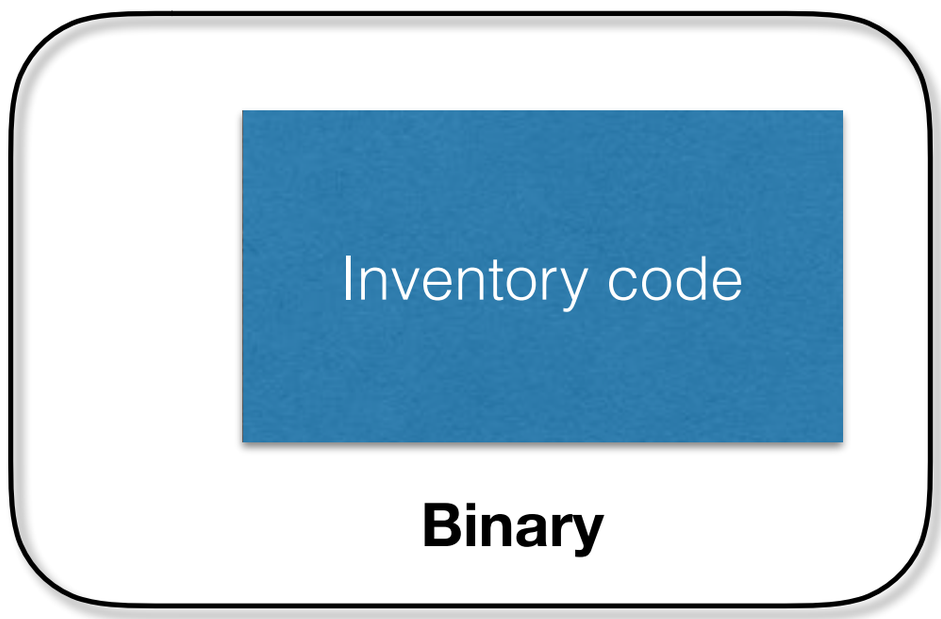
Modules

Modules

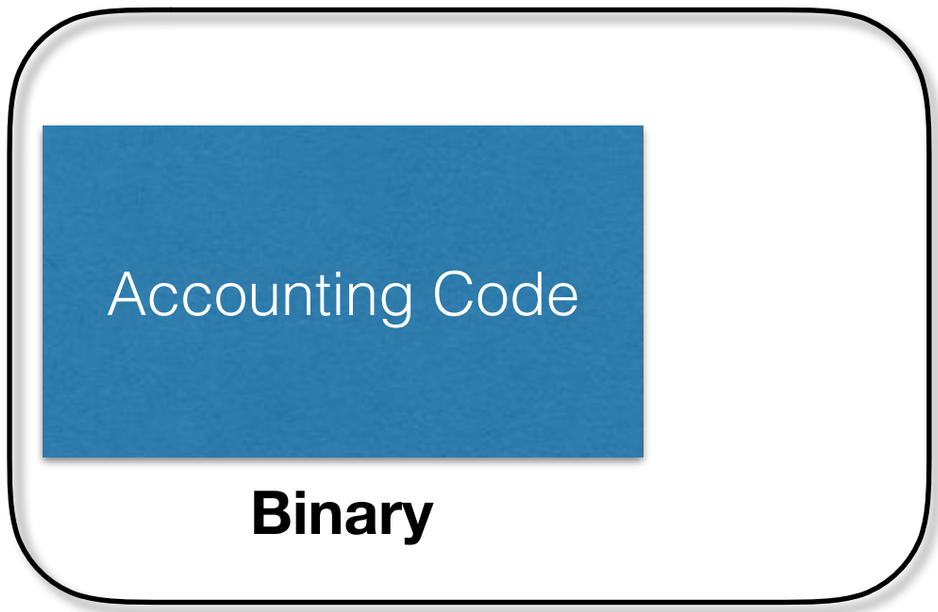
App

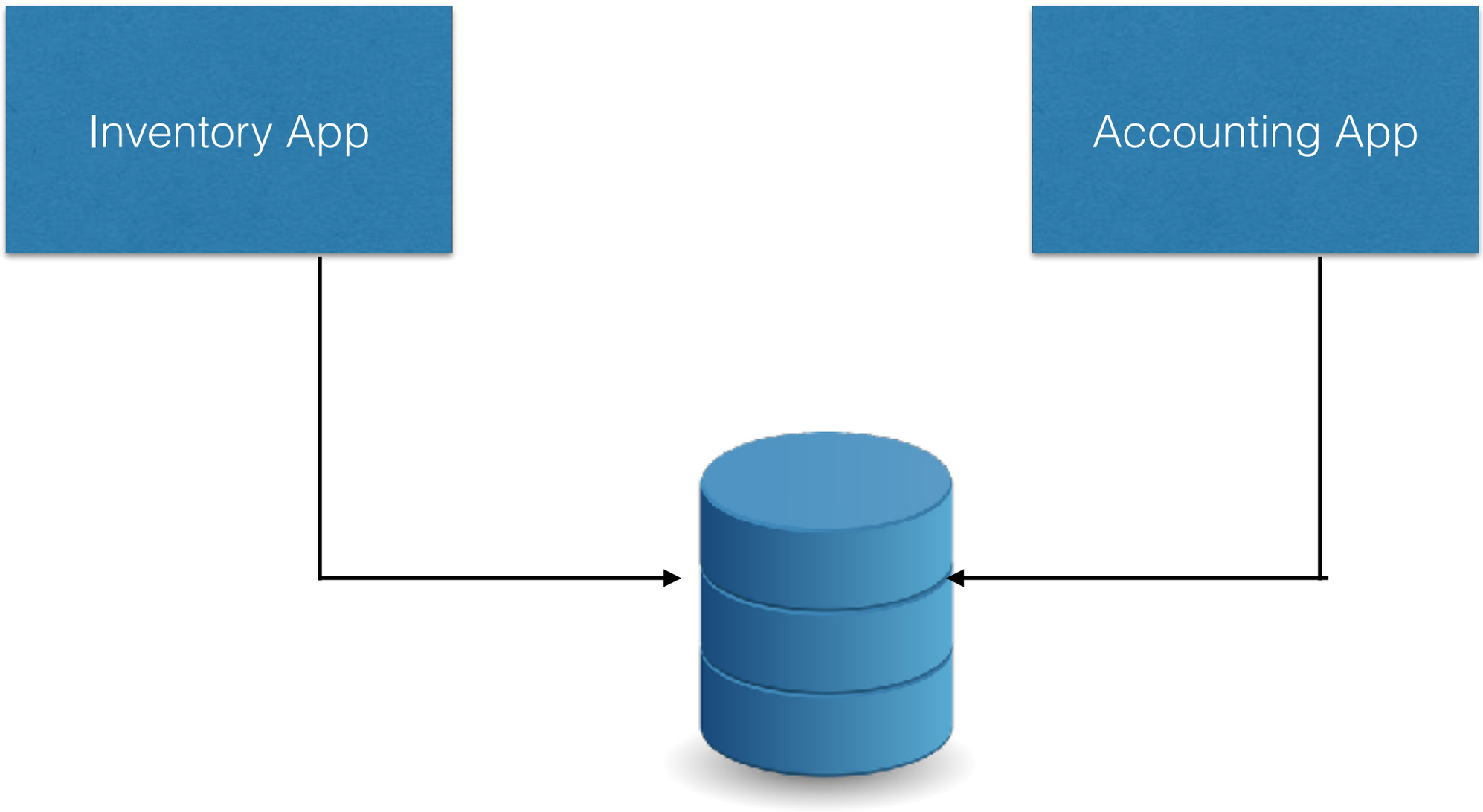

Inventory code
Accounting Code
Binary
Binary

App

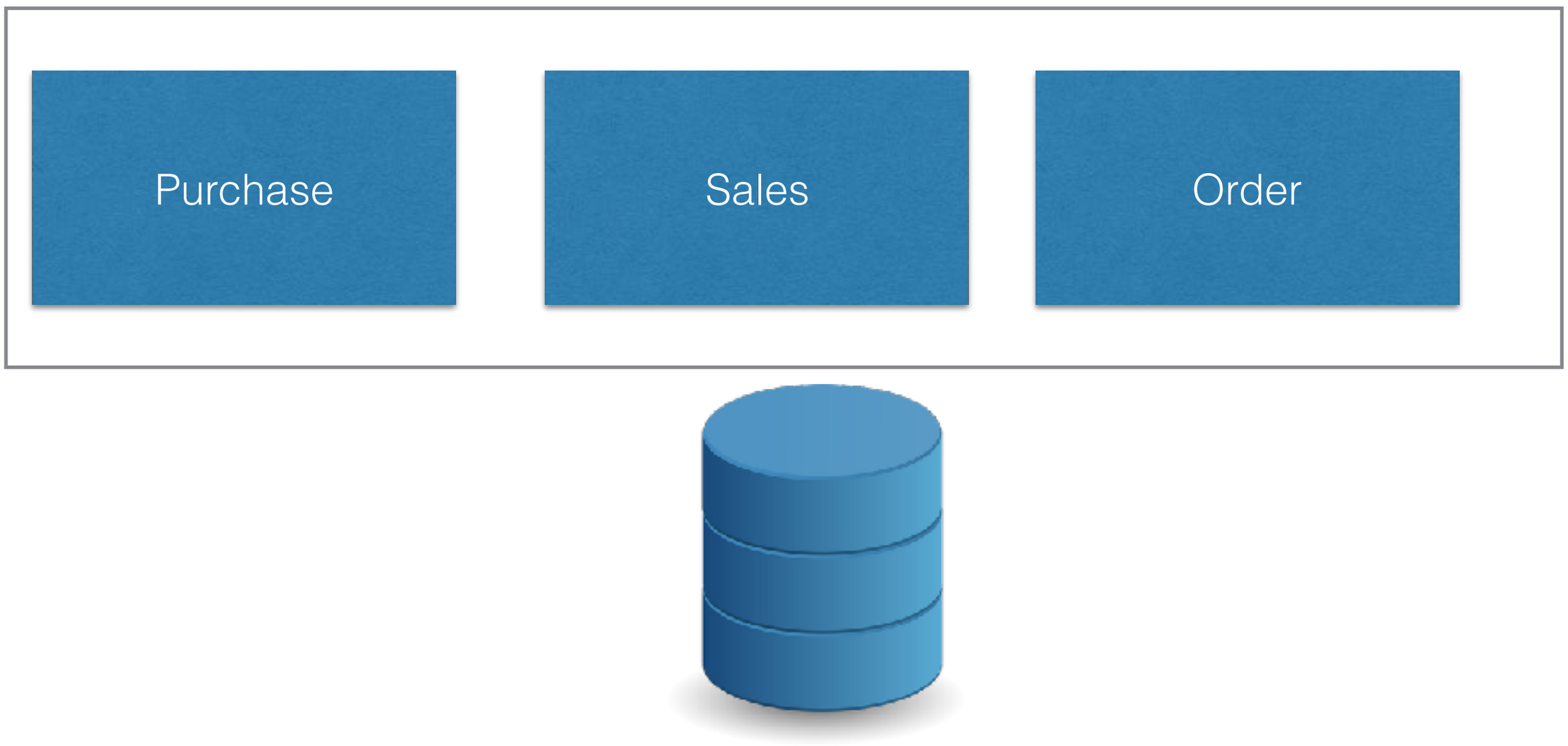

Inventory code
Binary

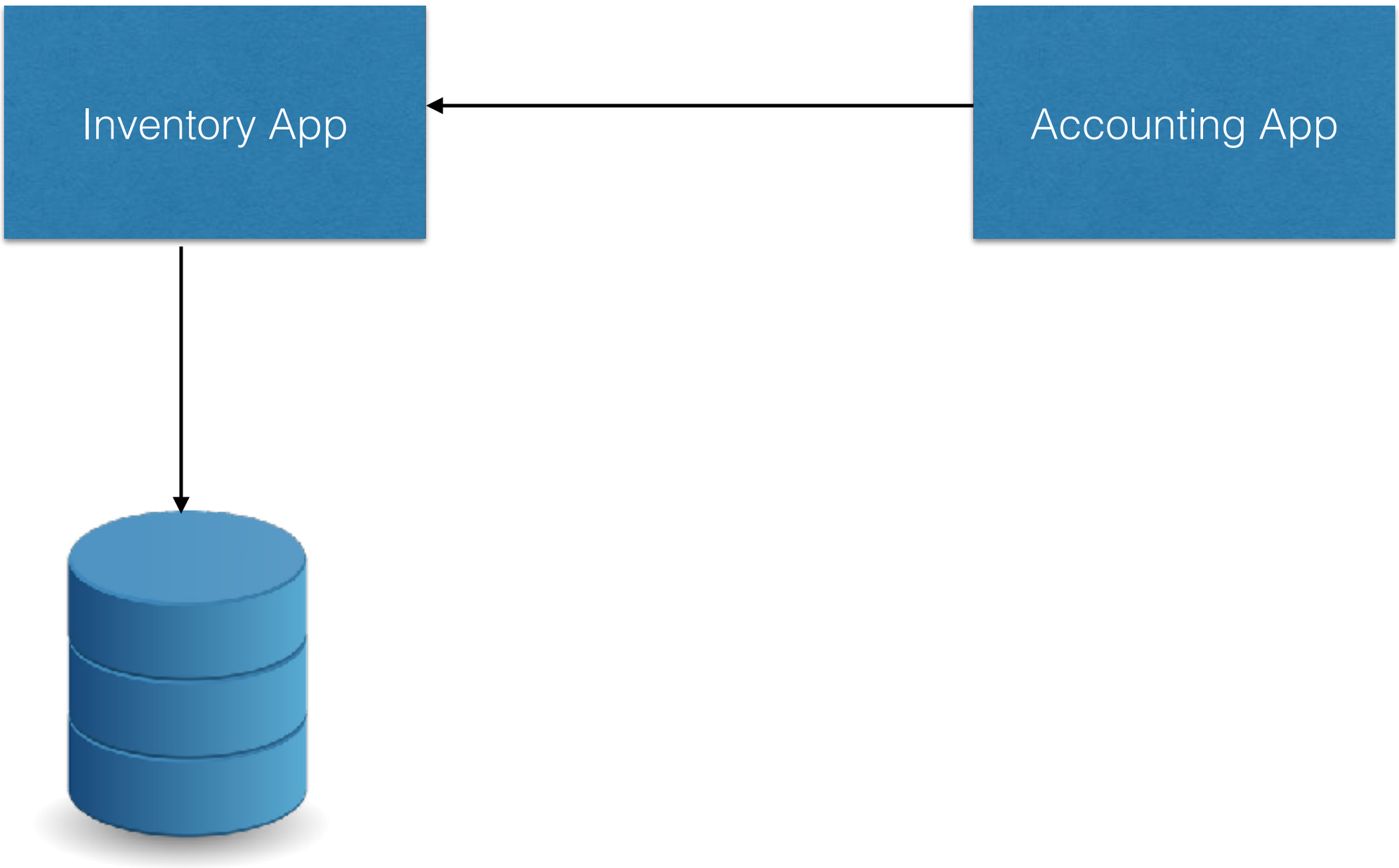
App


Accounting Code
Binary


Inventory App
Accounting App

Inventory Application




Inventory App
Accounting App

VM

Inventory App

Accounting App

APP

APP

APP

APP

APP

App

Mod1

Mod2

Mod3

Microservice
PLA
SOA

App

App1

Mod1

Mod2

Layered
Pipes& filter
Micro kernel
Hexagonal
...

App2

App3

Types of Microservice

App

App

1
Web Server

UI App (SPA)

API Layer

Domain Layer
<<feature1>>

Data Layer

Angular (SPA)

2
Web Server

Browser

UI Layer
(server page)

Domain Layer
<<feature2>>

Data Layer

JSP, Servlet, JSF (Server pages)

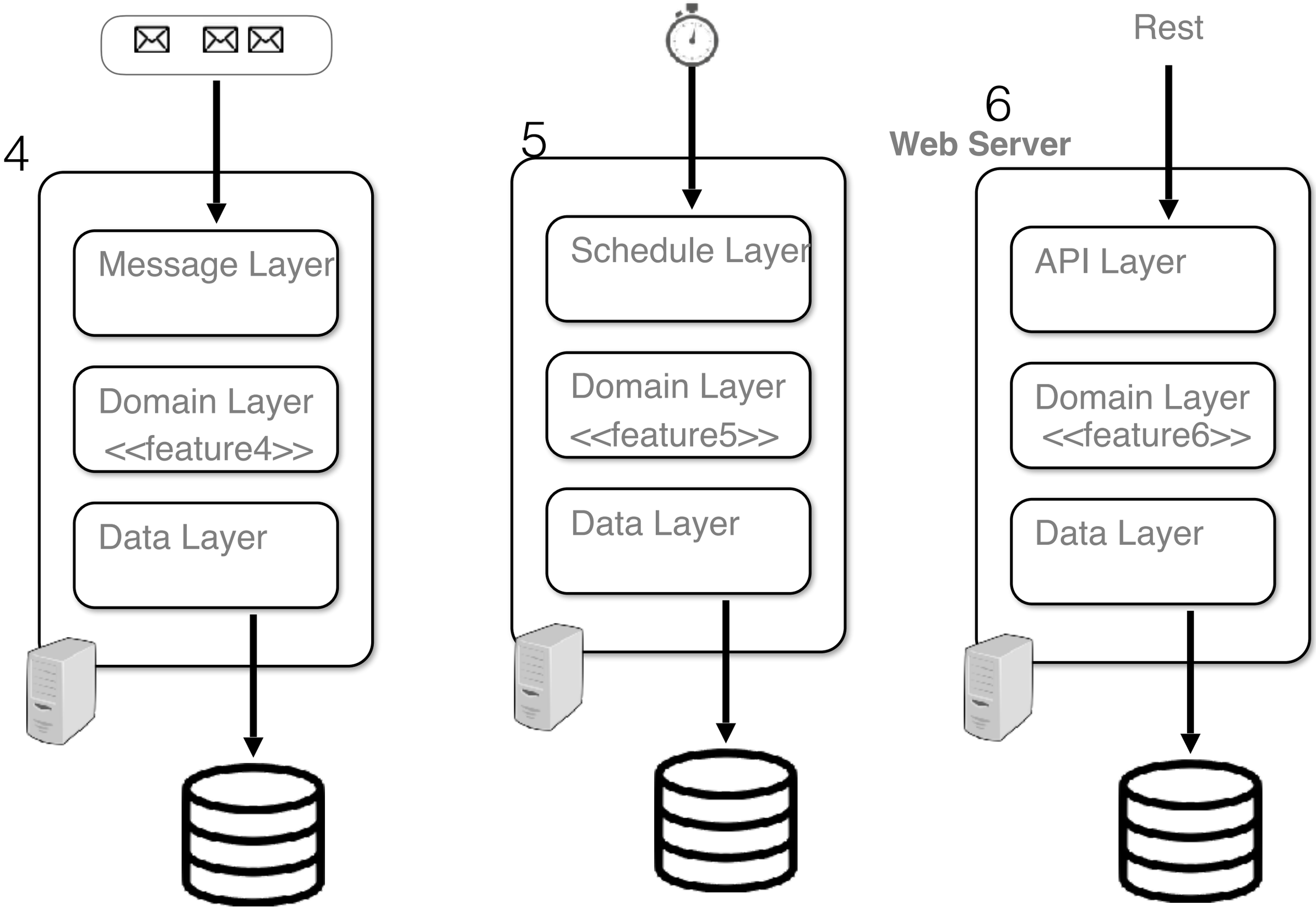
3
Desktop

UI Layer
(native)

Domain Layer
<<feature3>>

Data Layer

Swing, Android, Ios (Native)

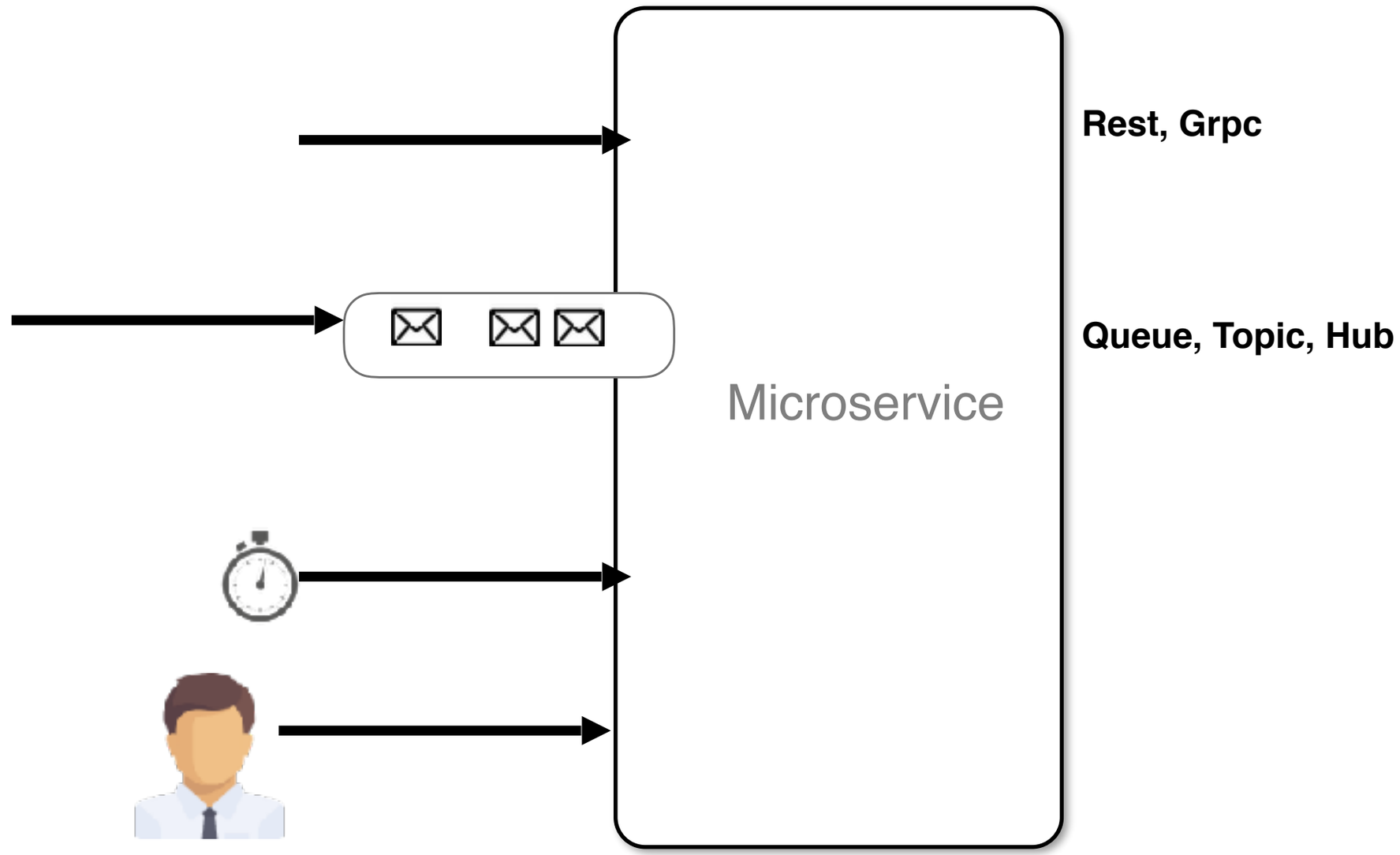

4
Message Layer
Domain Layer
<<feature4>>
Data Layer
5
Schedule Layer
Domain Layer
<<feature5>>
Data Layer
6
Web Server
Rest
API Layer
Domain Layer
<<feature6>>
Data Layer

API

* Message

Schedule

UI



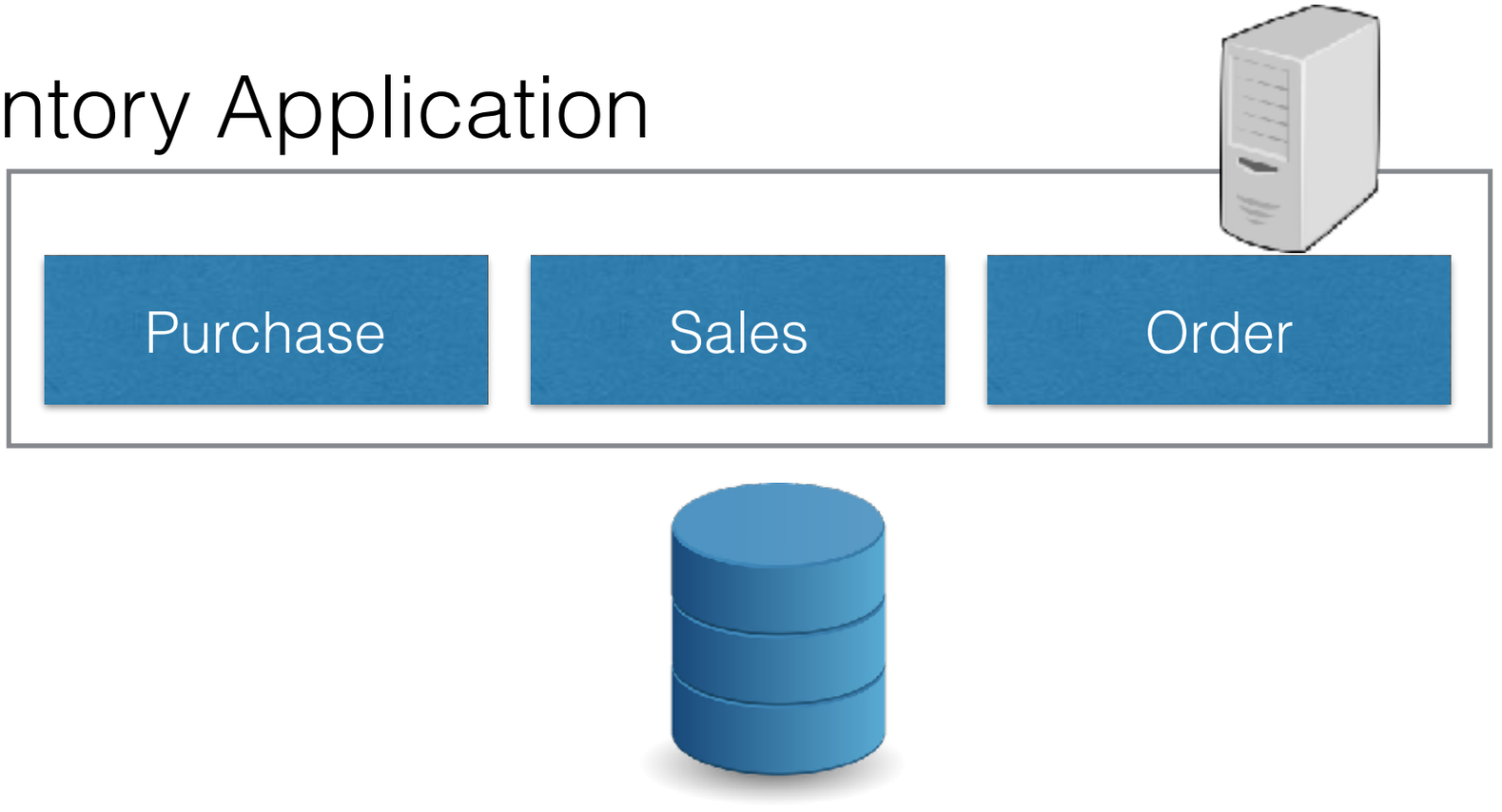
Rest, Grpc

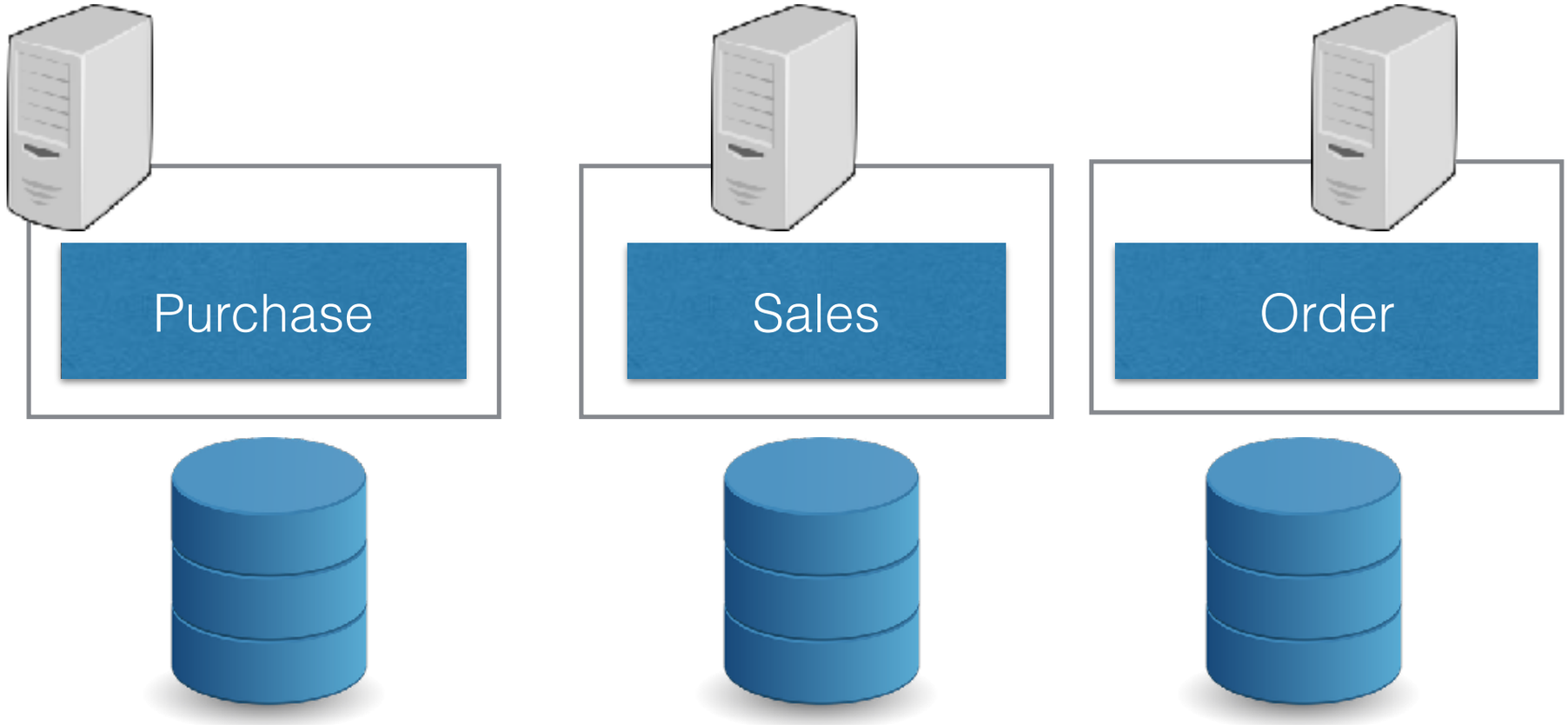
Queue, Topic, Hub

	Pros/	Solution
Development time	- -	
Micro-service practices Learning Curve	- -	
Resource Performance (CPU, Memory, I/O)	- -	Grpc, protobuf, ..
Db Transaction Management	- -	SAGA
Views / Report / Dash board/ join	- -	Materialized View
Infra Cost	- -	Containerization
First time Deployment effort	- -	Automation
Debugging, Error Handling (End to End)	- -	Distributed Tracing (Jaeger)
Integration Test	- -	
Log Mgmt (debug/ error)	- -	Centeralize EFK, ELK, Splunk
Config Mgmt	- -	Centralize Consul, Zookeeper, ..
Authentication (who)	- -	Centralize (Oauth2, OpenID connect, SAML, …)
Authorization (what can they do)	- -	Centeralize (Claims)
Audit Log mgmt (who accessed what)	- -	Centeralize (Event Store, …)
Monitoring / Alerting	- -	Centeralize
Data Security and Privacy (transit, storage)	- -	
Build Pipeline (CI/CD)	- -	Automation
Agile Architecture (Agility to change)	+ + +	fwk, technology, platform, …
Feature Shipping (Agility to ship)/ CD	+++	Timing of when to goto production
Scalability (volume - request, data, compute)	+	
Availability	+	
Ability to do Polygot	+	
Fault Isolation	+	

	Pros/ Cons	Solution
Development time	- -	
Micro-service practices Learning Curve	- -	
Resource Performance (CPU, Memory, I/O)	- -	
Db Transaction Management	- -	SAGA
Views / Report / Dash board/ join	- -	Materialized View, CQRS
Infra Cost	- -	Containerization
First time Deployment effort	- -	Automation (chef, puppet)
Debugging, Error Handling (End to End)	- -	Distributed Tracing (Jaeger, …)
Integration Test	- -	
Log Mgmt (debug/ error)	- -	Centralize - ELK, EFK, Splunk
Config Mgmt	- -	Centralize - Consul
Authentication (who)	- -	Centralize - Oauth2, SSO, SAML, …
Authorization (what can they do)	- -	Centralize - ?
Audit Log mgmt (who accessed what)	- -	Centralize - Event Sourcing
Monitoring / Alerting	- -	Centralize
Data Security and Privacy (transit, storage)	- -	
Build Pipeline (CI)	- -	Automation
Agile Architecture (Agility to change)	+ + +	
Feature Shipping (Agility to ship)/ CD	+ +	
Scalability (volume - request, data,	+	
Availability	+	
Ability to do Polygot	+	
Fault Isolation	+	

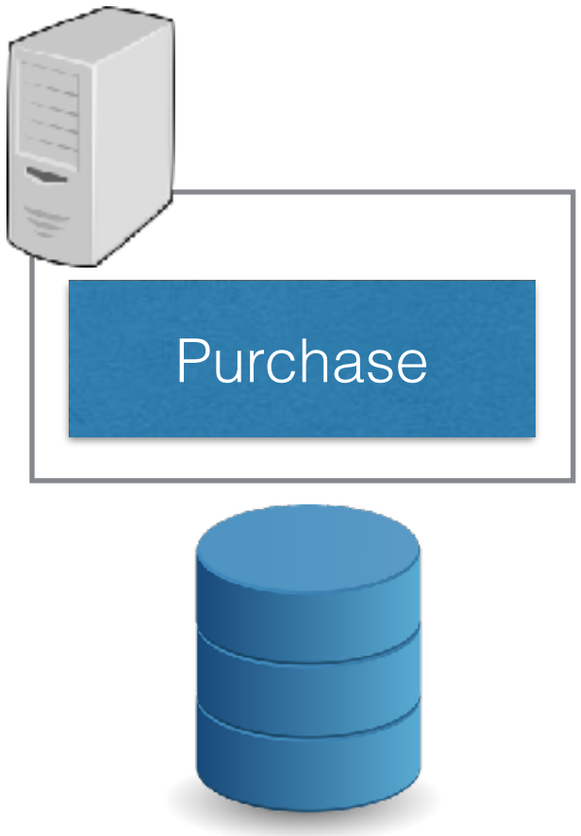
Inventory Application





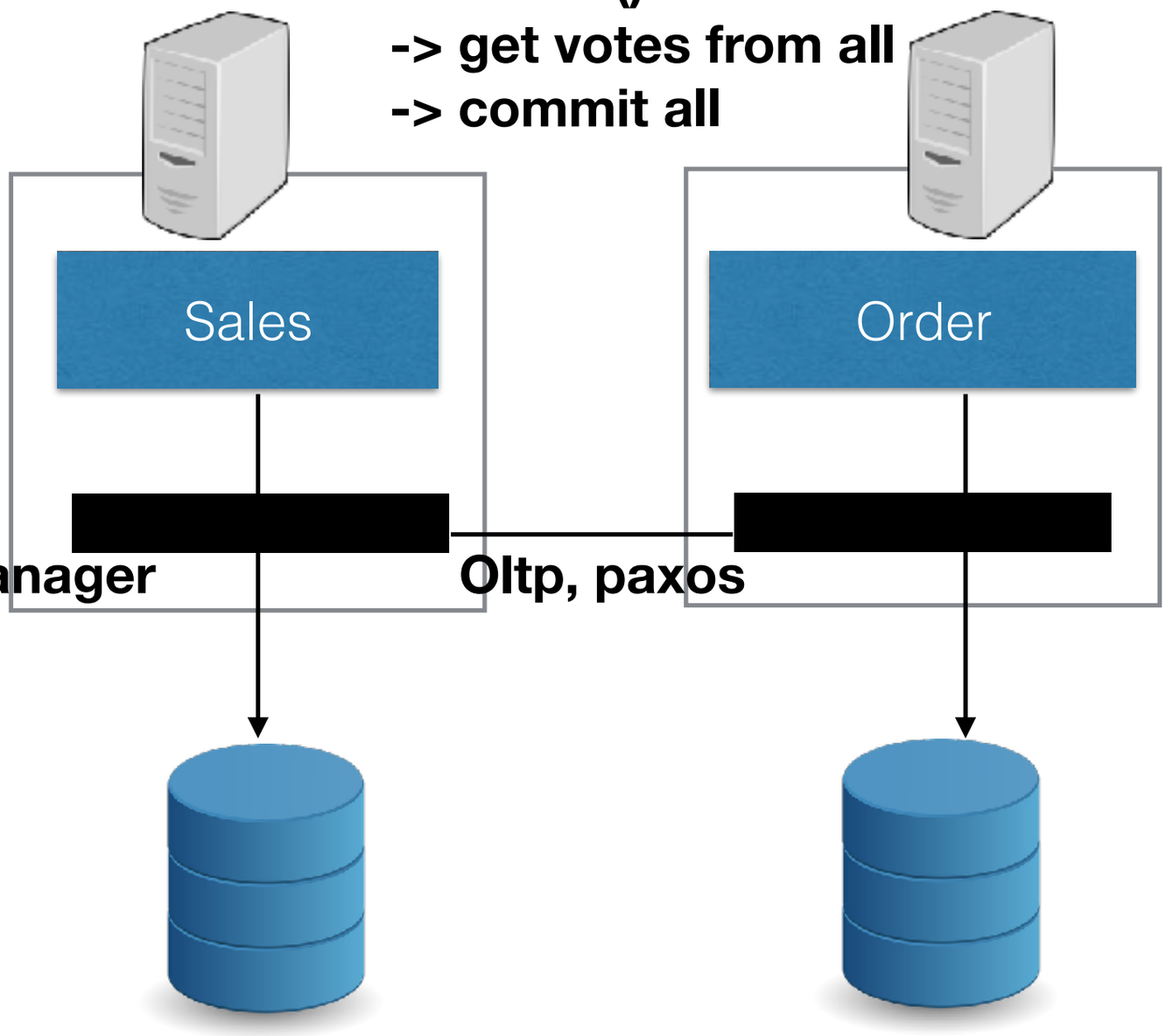
1 phase commit

db.Begin()
Cmd
Cmd
Cmd
db.Commit()



2 phase commit

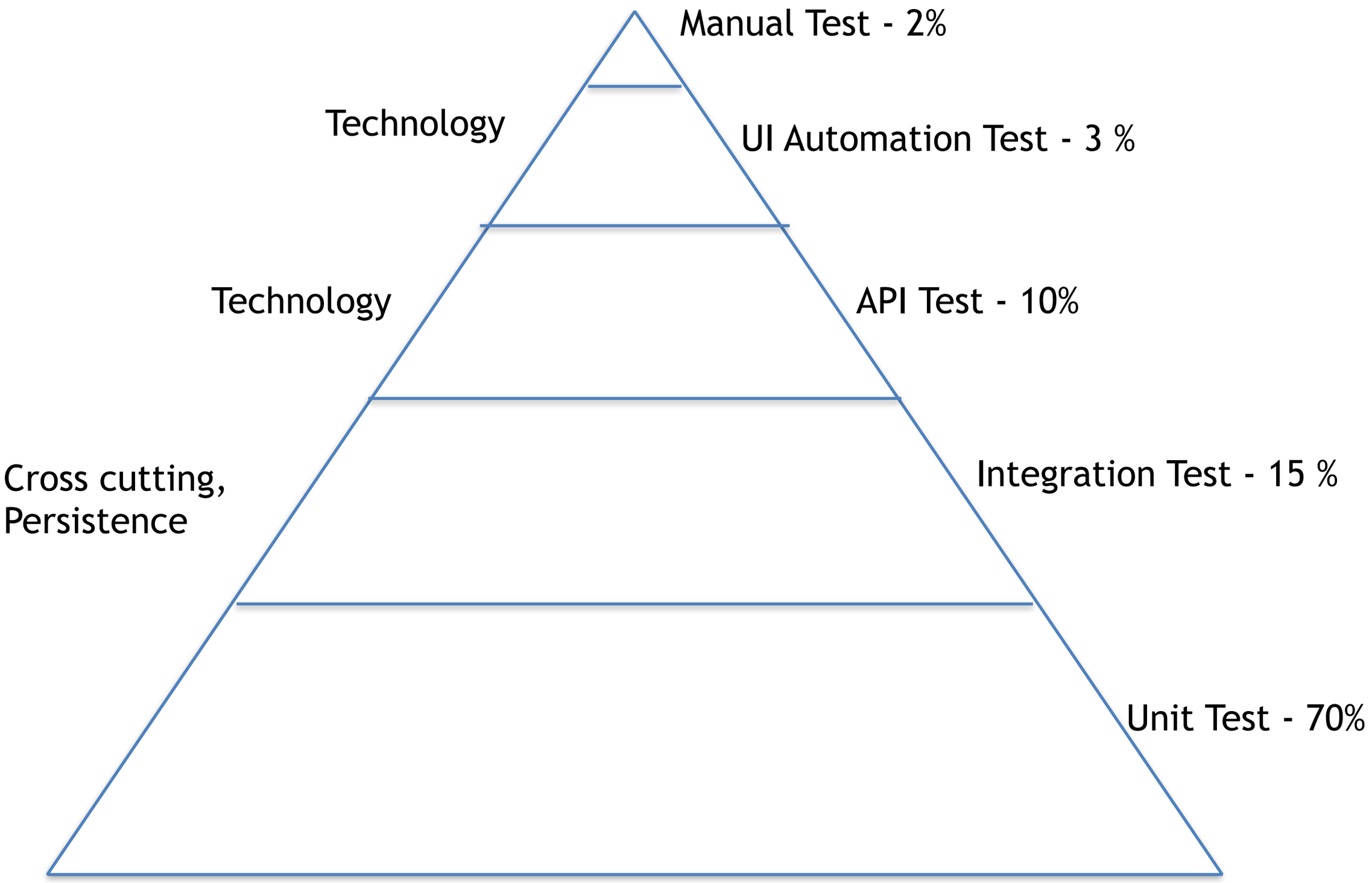
rm.Begin()
Cmd on db1
Cmd on db2
Cmd on db 1
…
rm.Commit()
-> get votes from all
-> commit all



Decentrailze Domain

Centrailze Tech

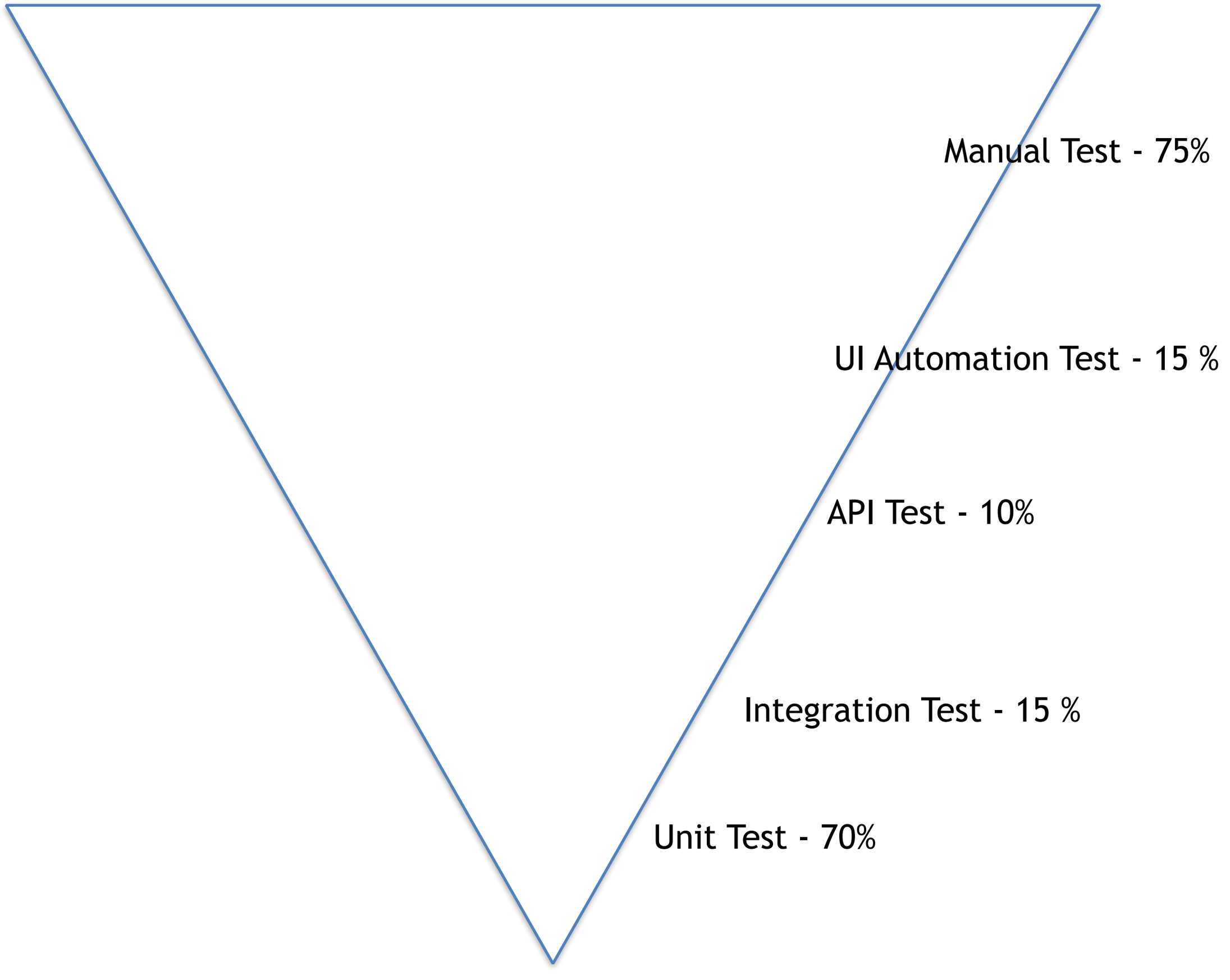
Pyramid test

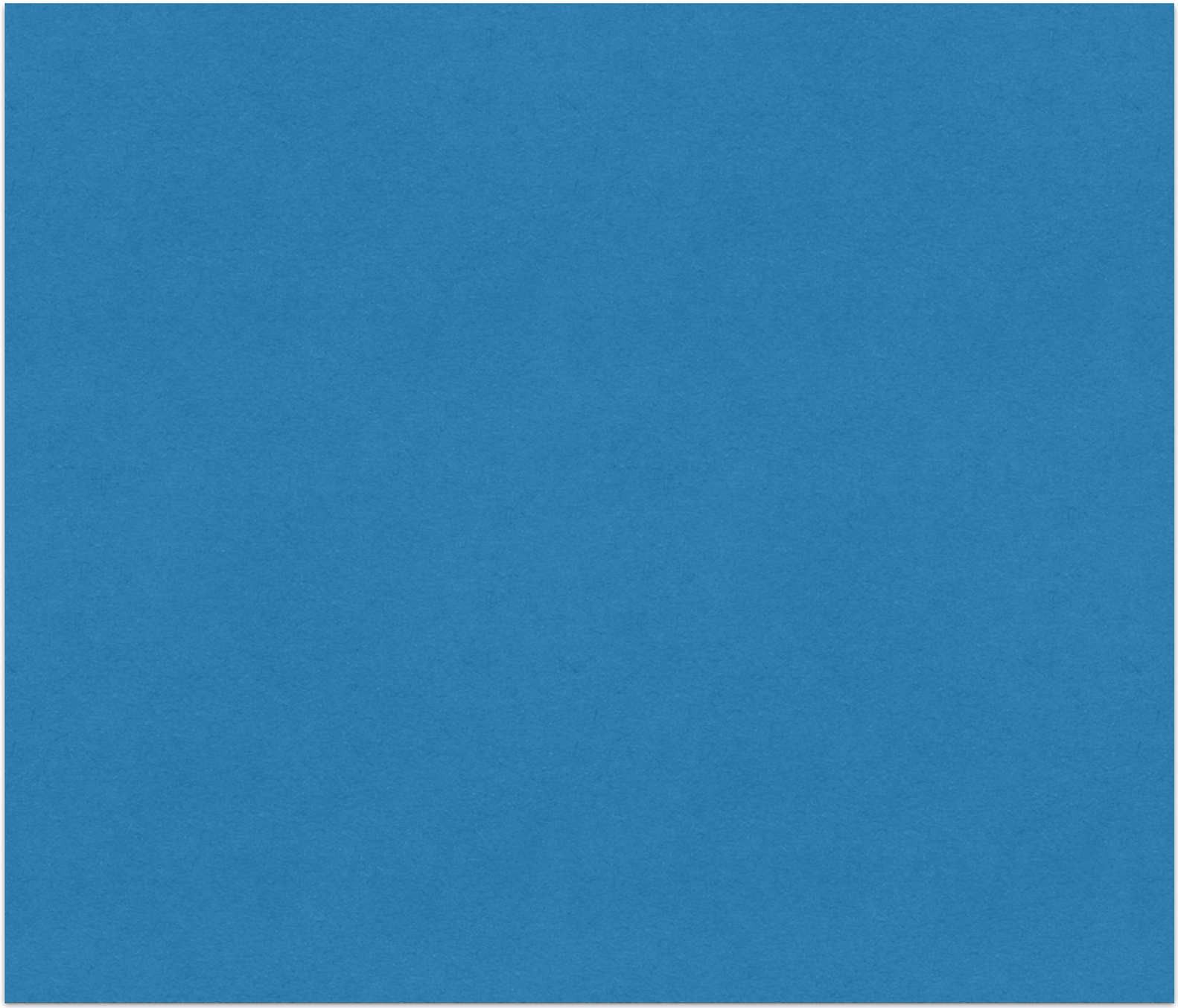


Unit Test

- Documentation for Code
- Design Code
- Regression
- Find Bugs

Its working don’t touch it


Manual Test - 75%
UI Automation Test - 15 %
API Test - 10%
Integration Test - 15 %
Unit Test - 70%



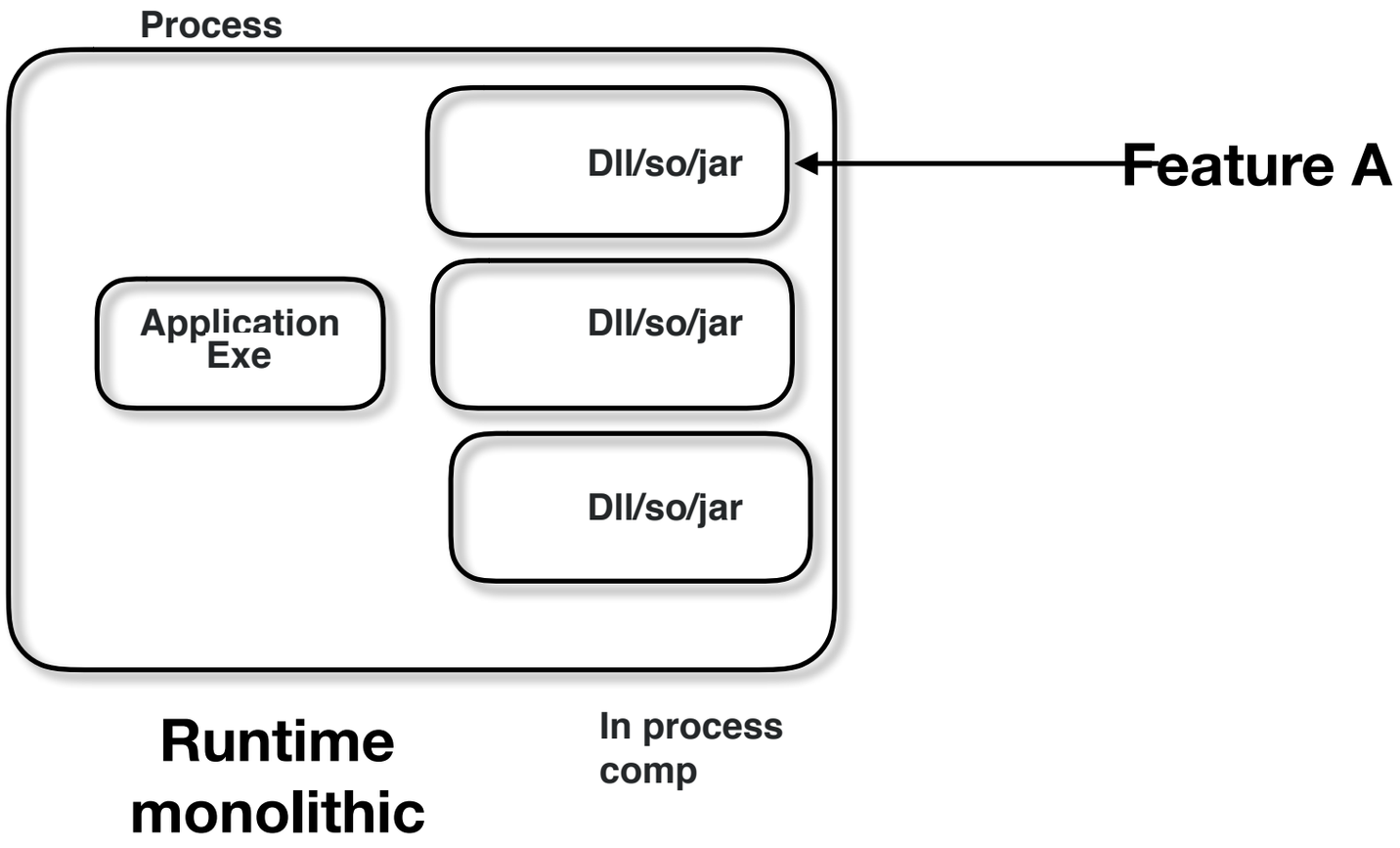
Manual Test - 2%

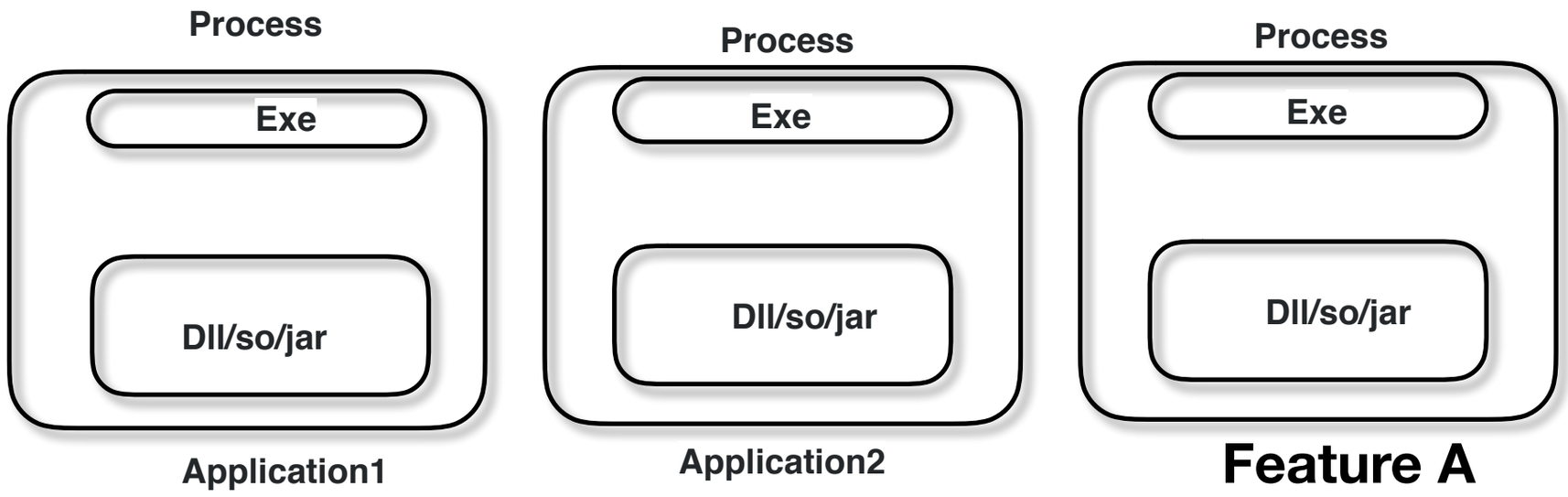
UI Automation Test - 3 %

API Test - 10%

Integration Test - 15 %

Unit Test - 70%

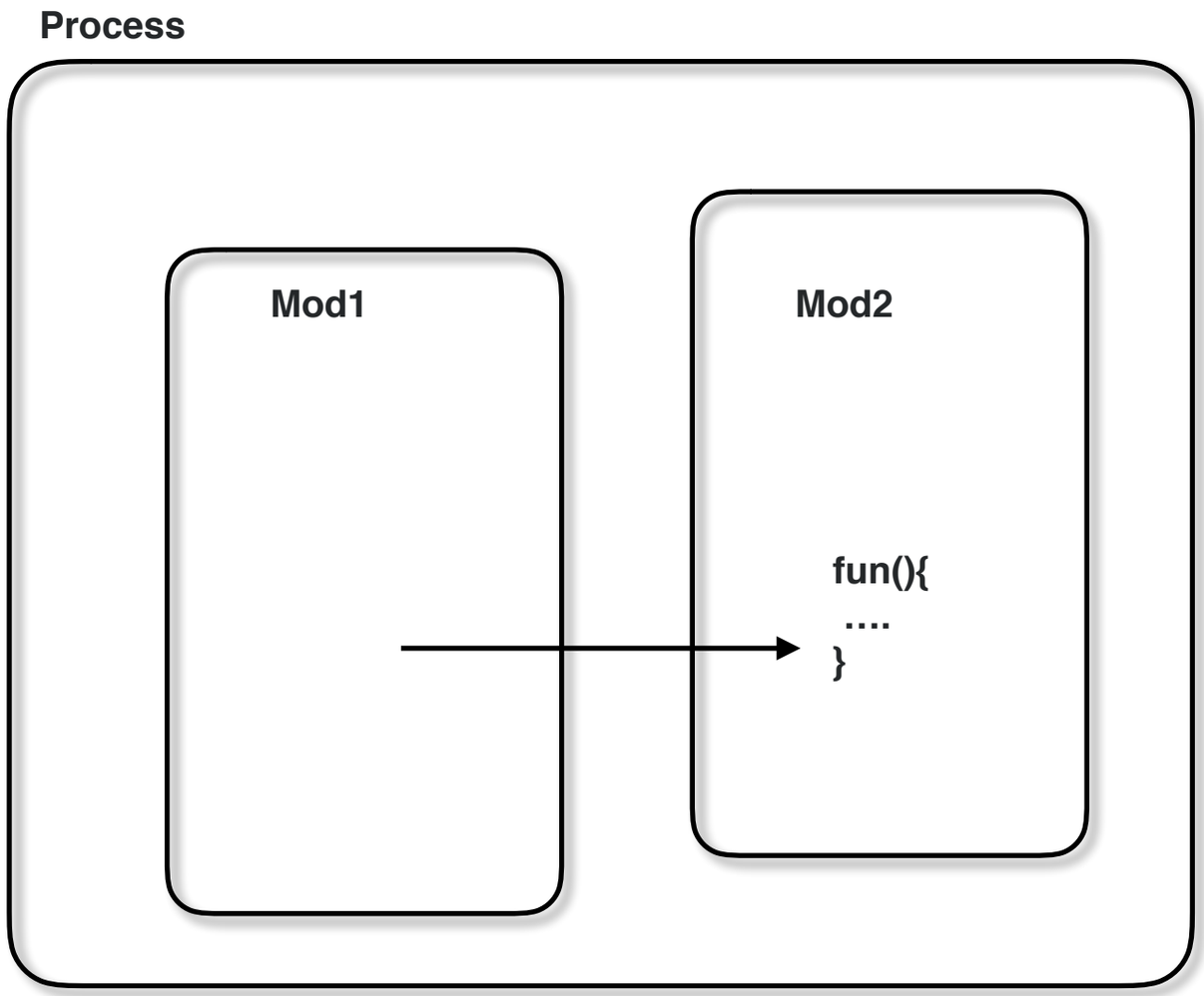




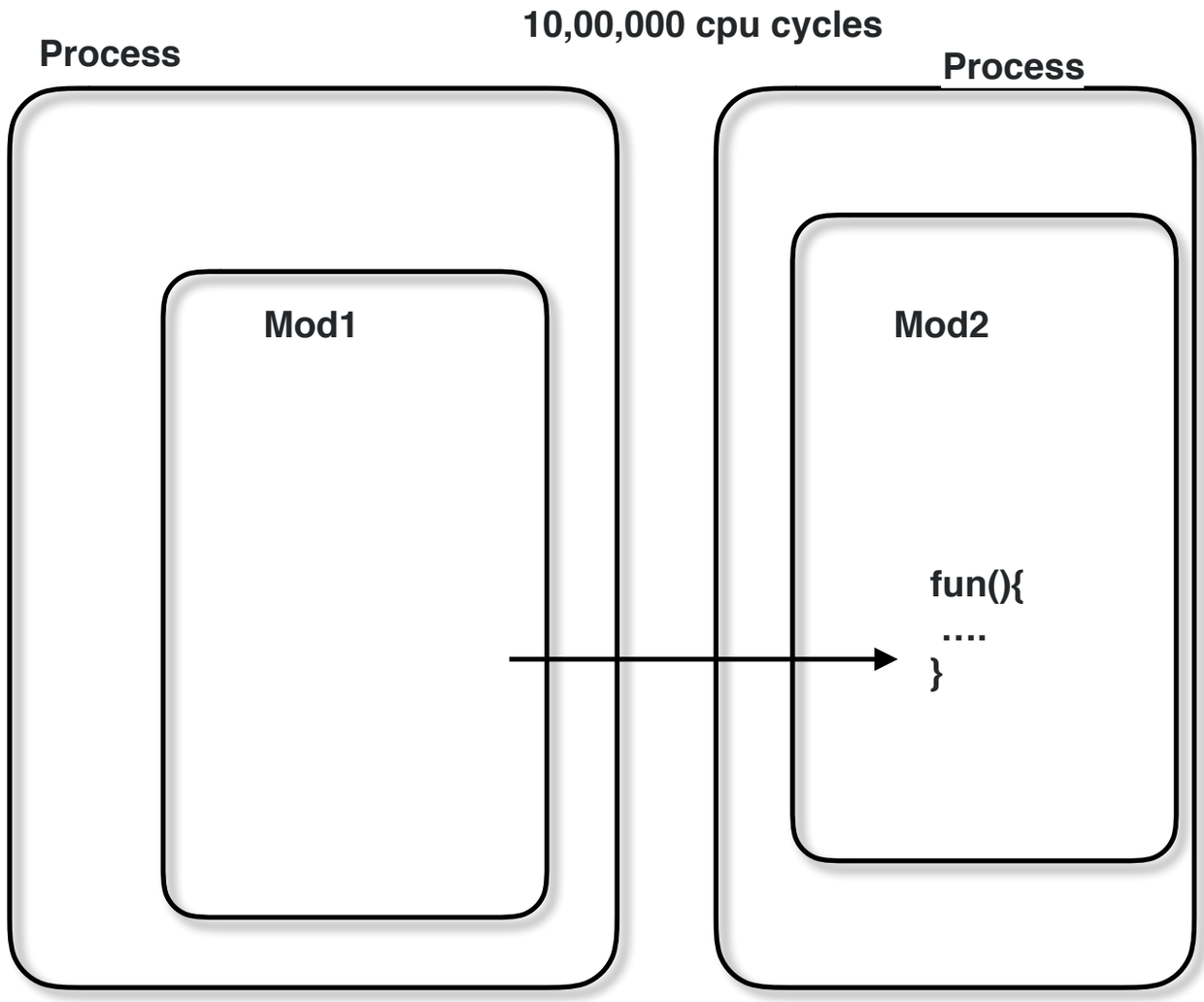
Micro Application

1. Performance

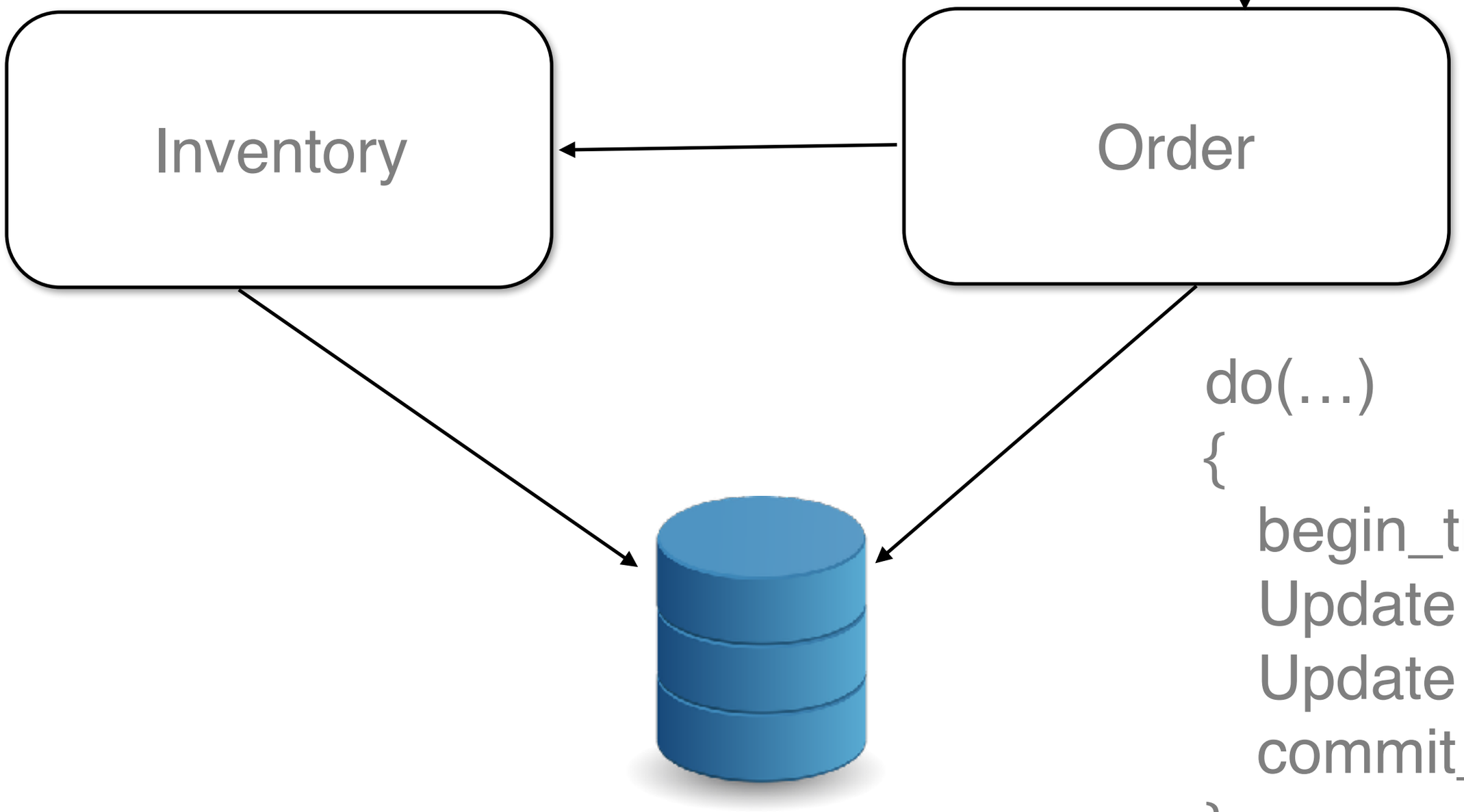
Operation	Cpu Cycles
• 10 + 12	3
• Calling a in-memory Method	10
• Create Thread	2,00,000
• Destroy Thread	1,00,000
• Database Call	40,00,000
• Distributed Fun Call	20,00,000
Write to disk	10,00,000



10 cpu cycles

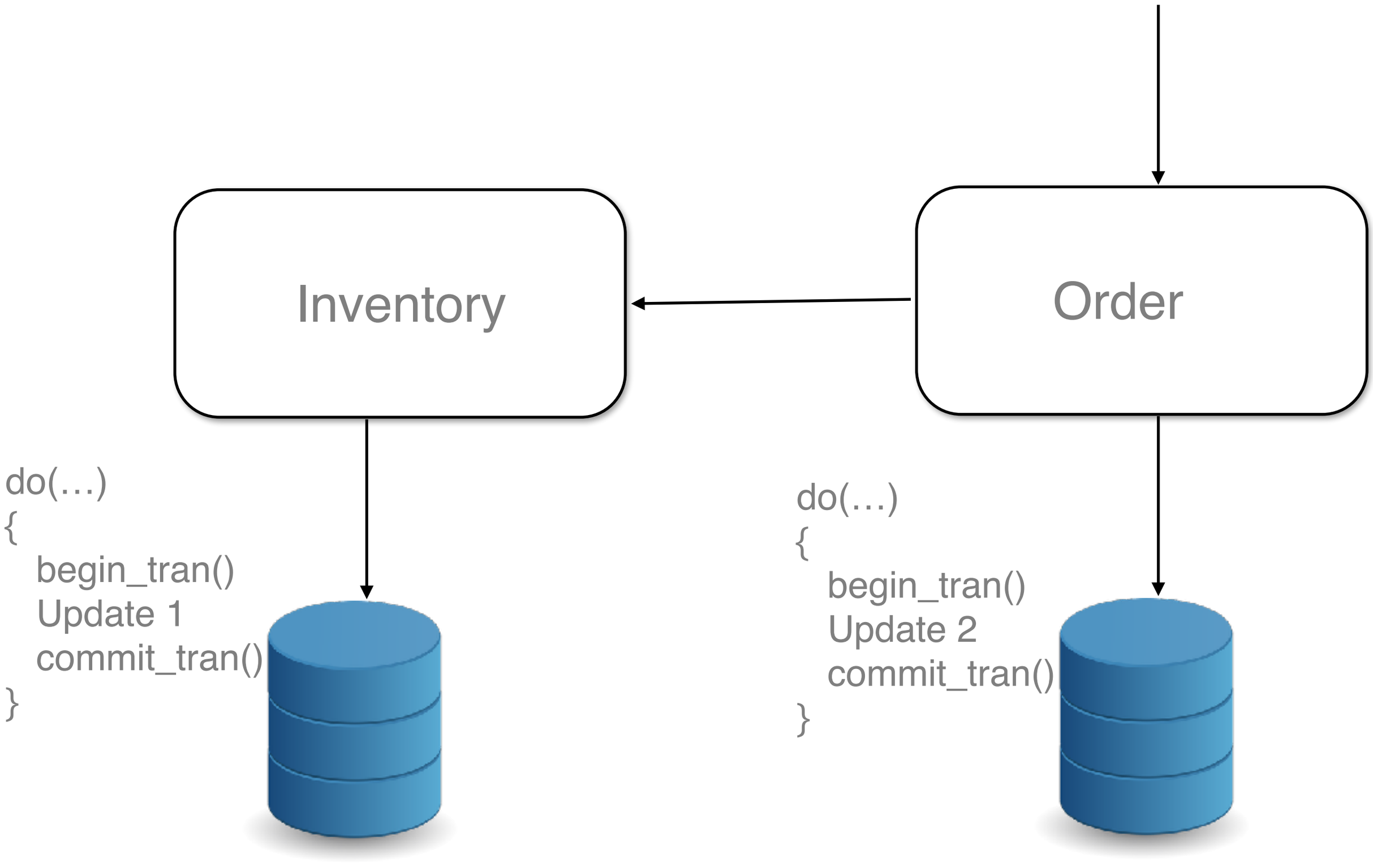


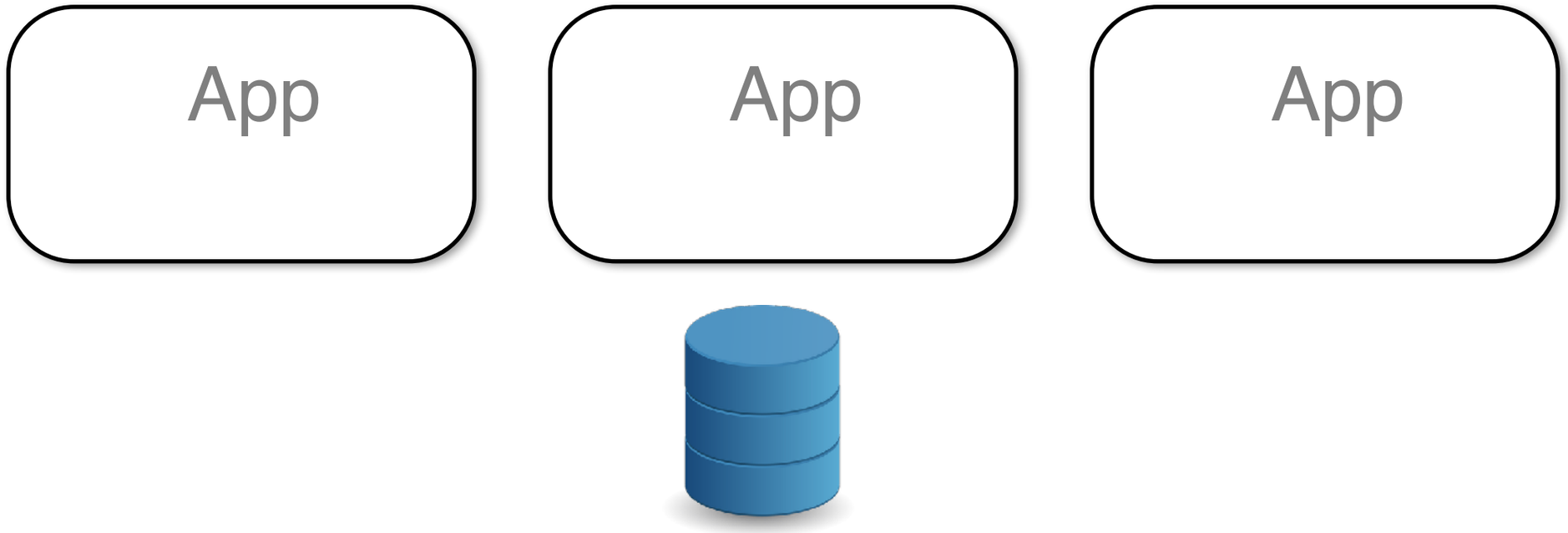
2. Db transaction



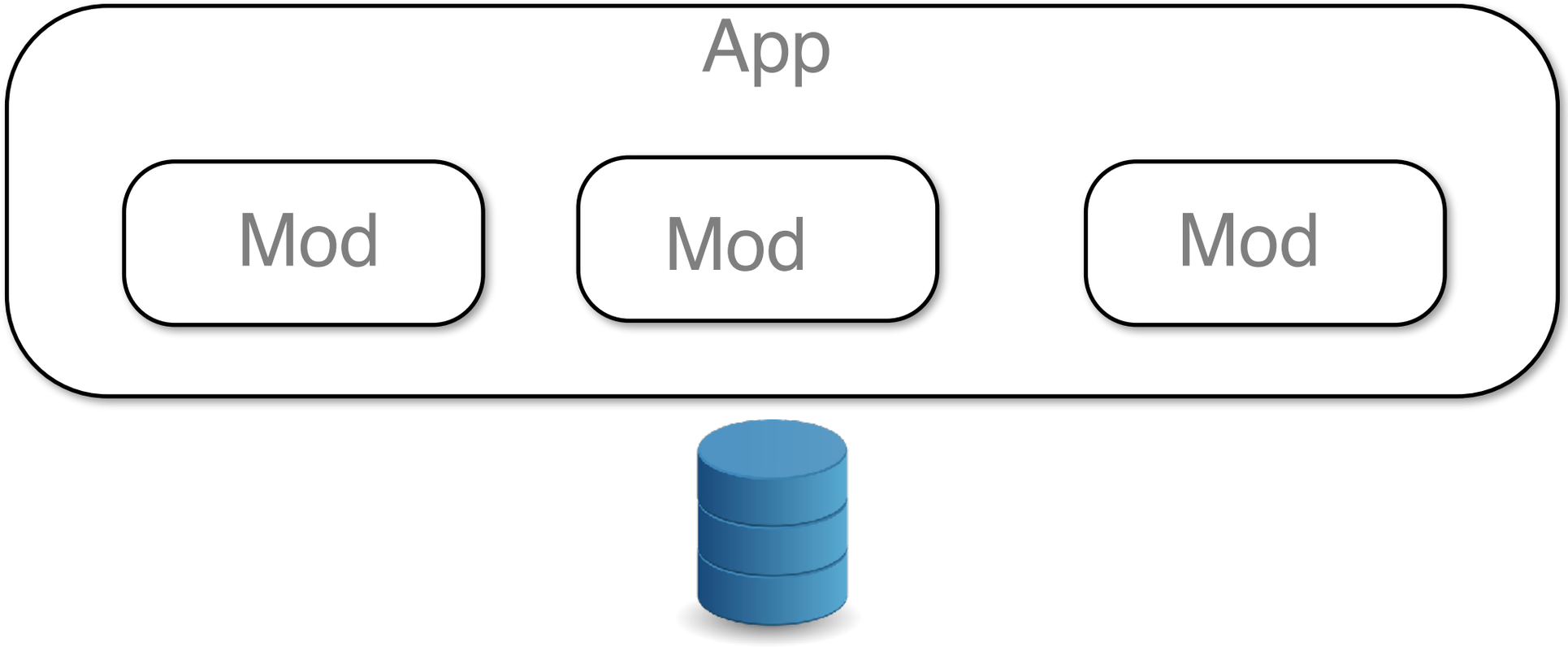

```
do(…)
{
   begin_tran()
   Update 1
   Update 2
   commit_tran()
}
```

2. Db transaction



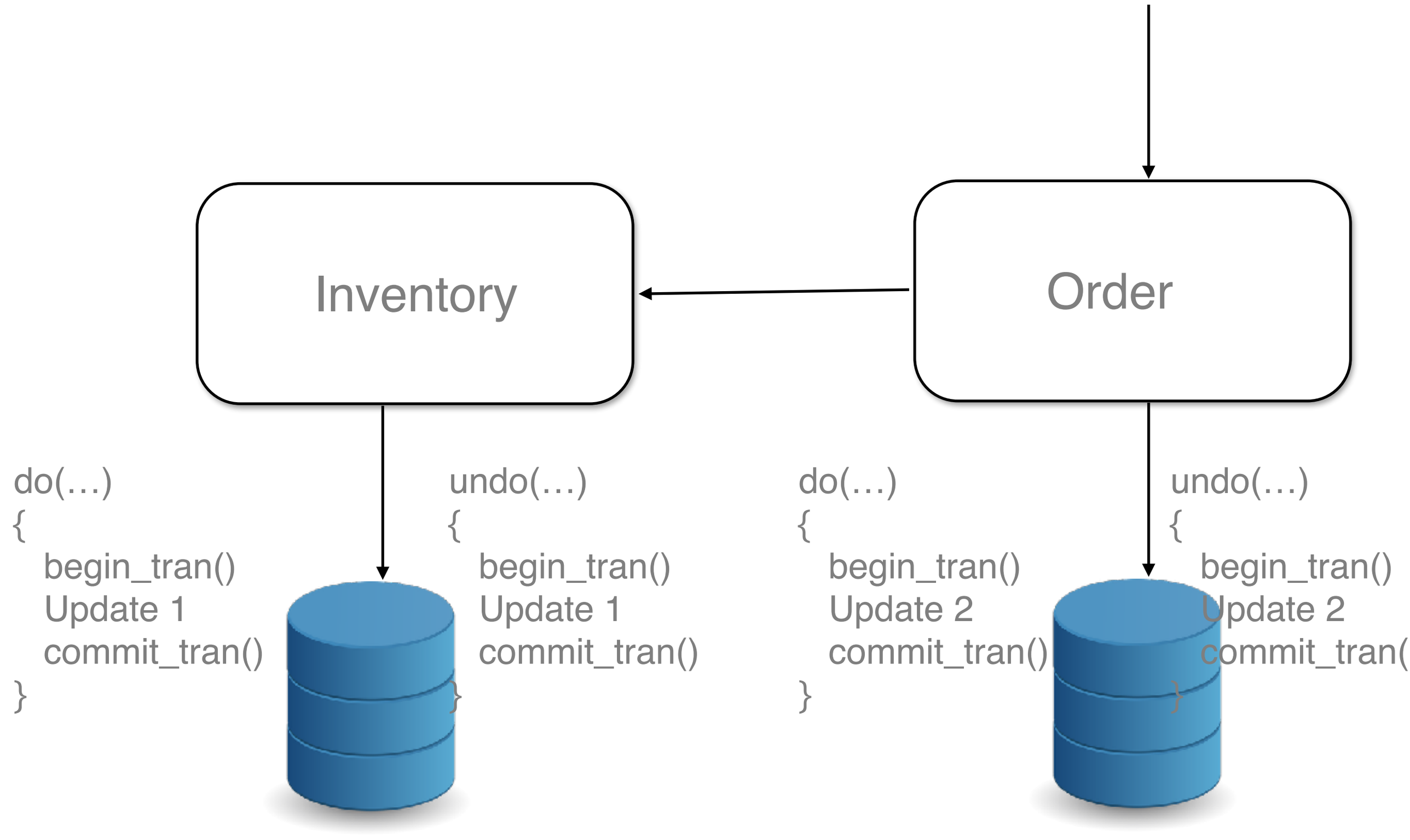

App
App
App

App
App
App


App
Mod
Mod
Mod

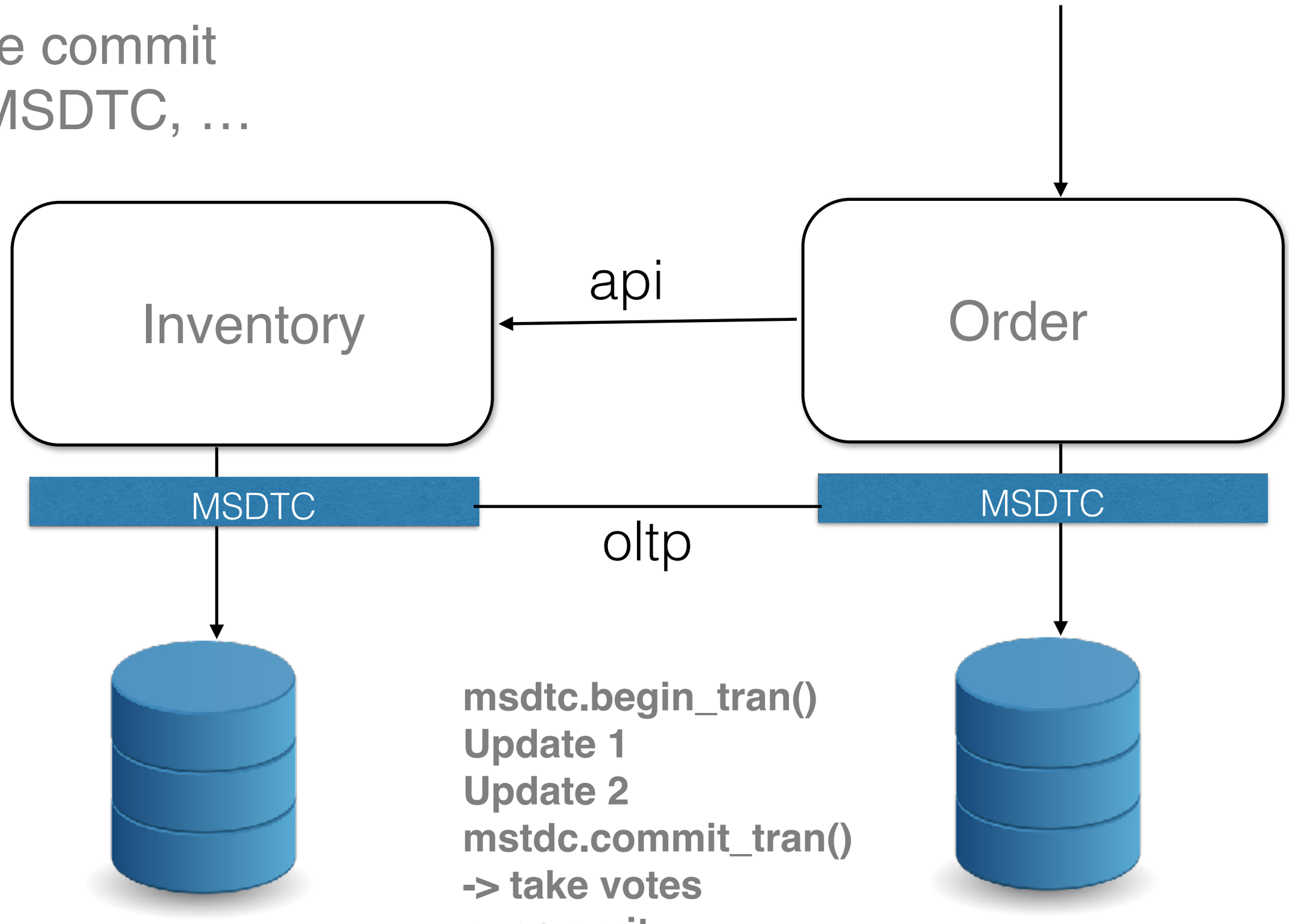

App
App
App

2. Db transaction



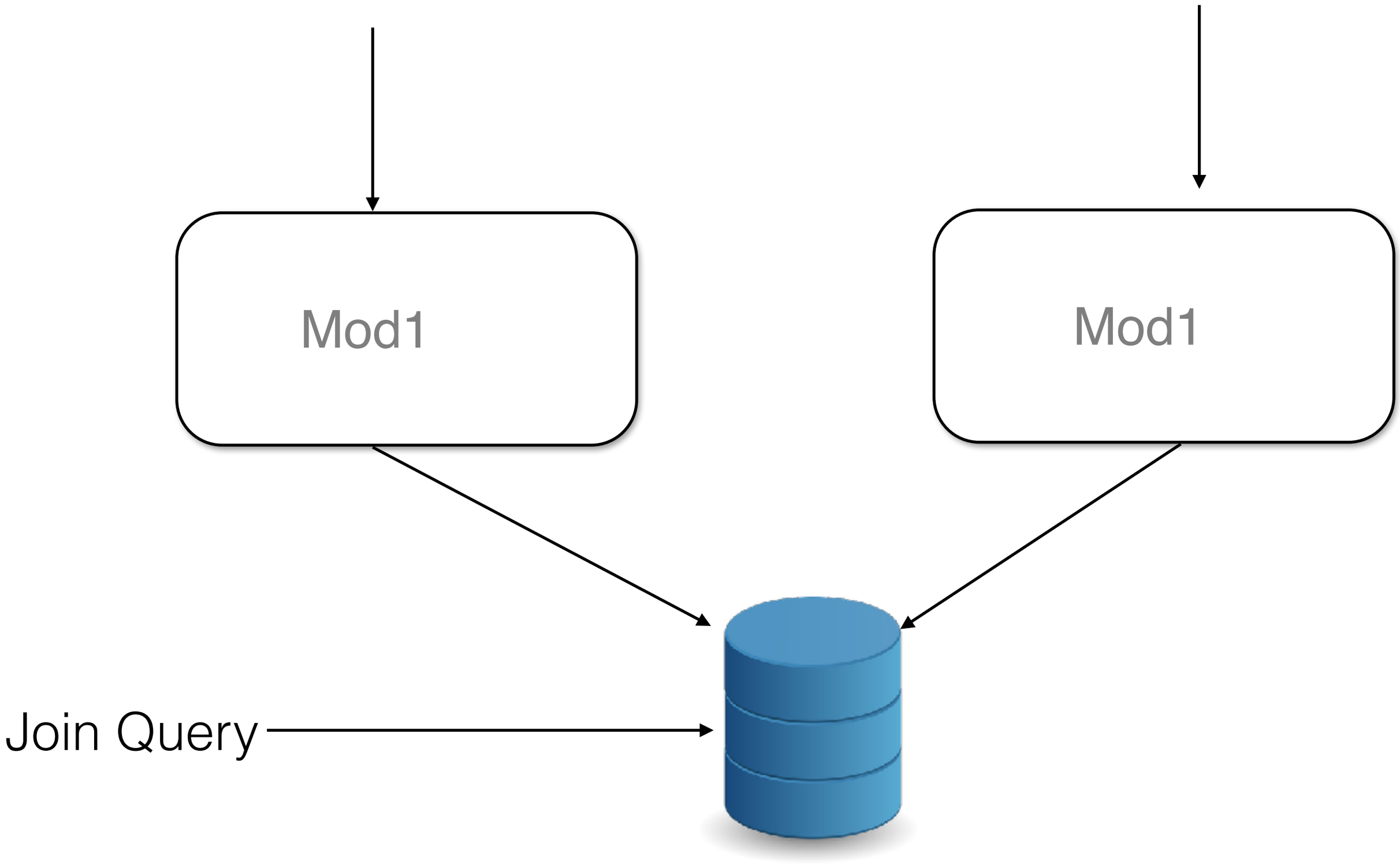
2. Db transaction

2 phase commit
JTX , MSDTC, …

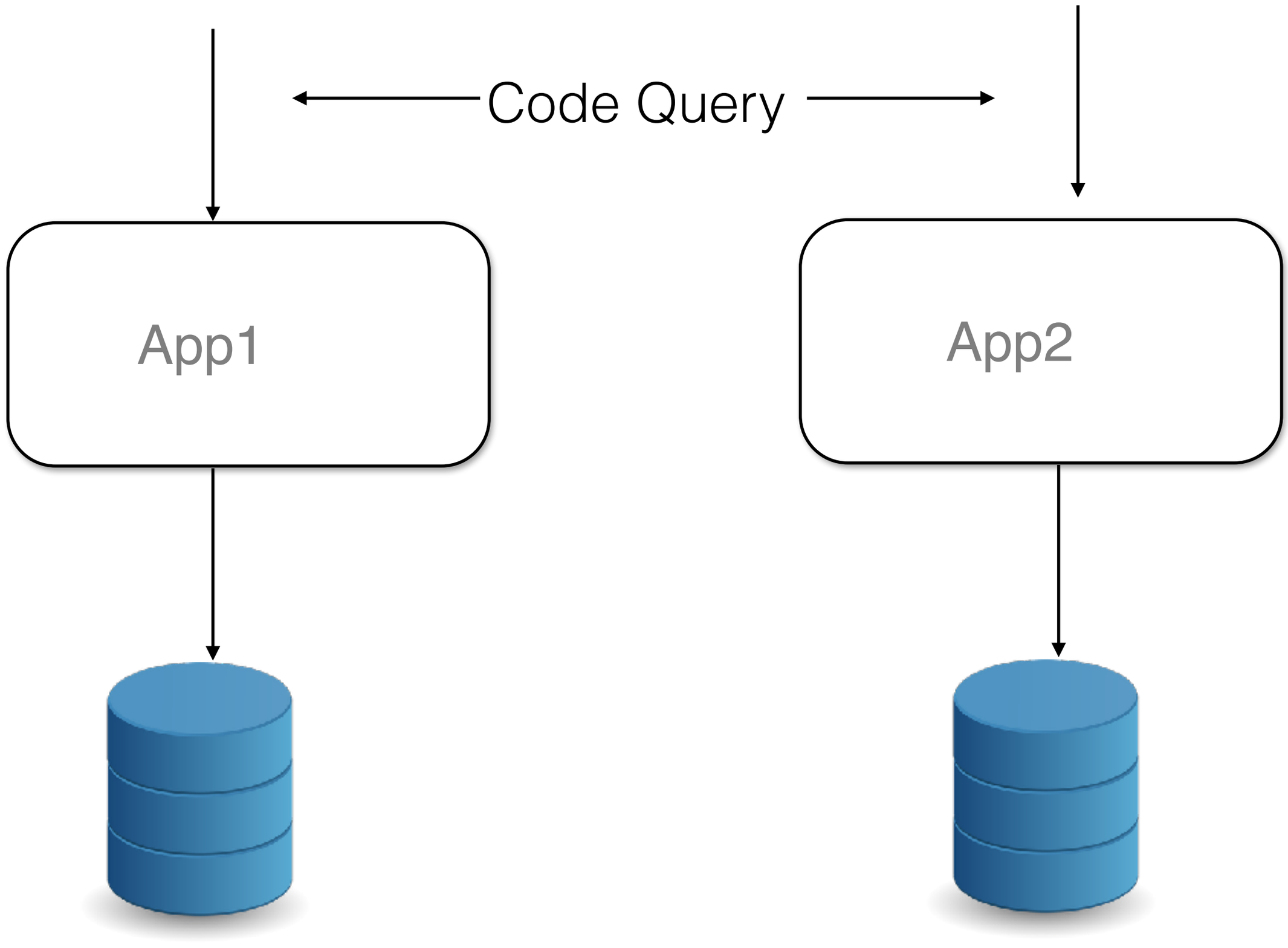


msdtc.begin_tran()
Update 1
Update 2
mstdc.commit_tran()
-> take votes
-> commit

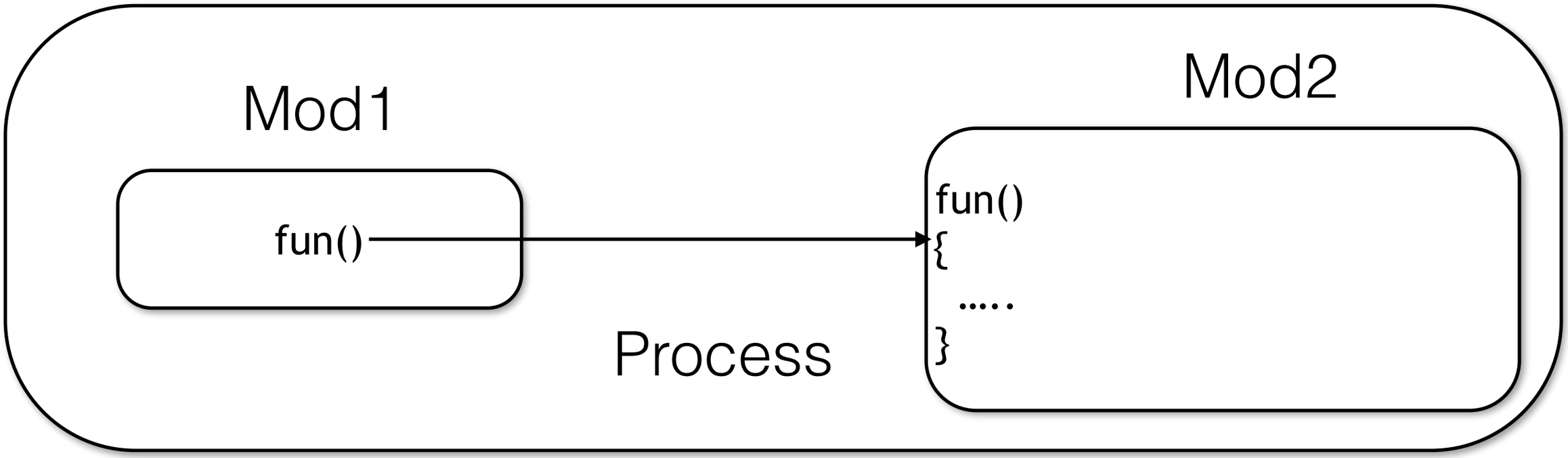
3. Db query

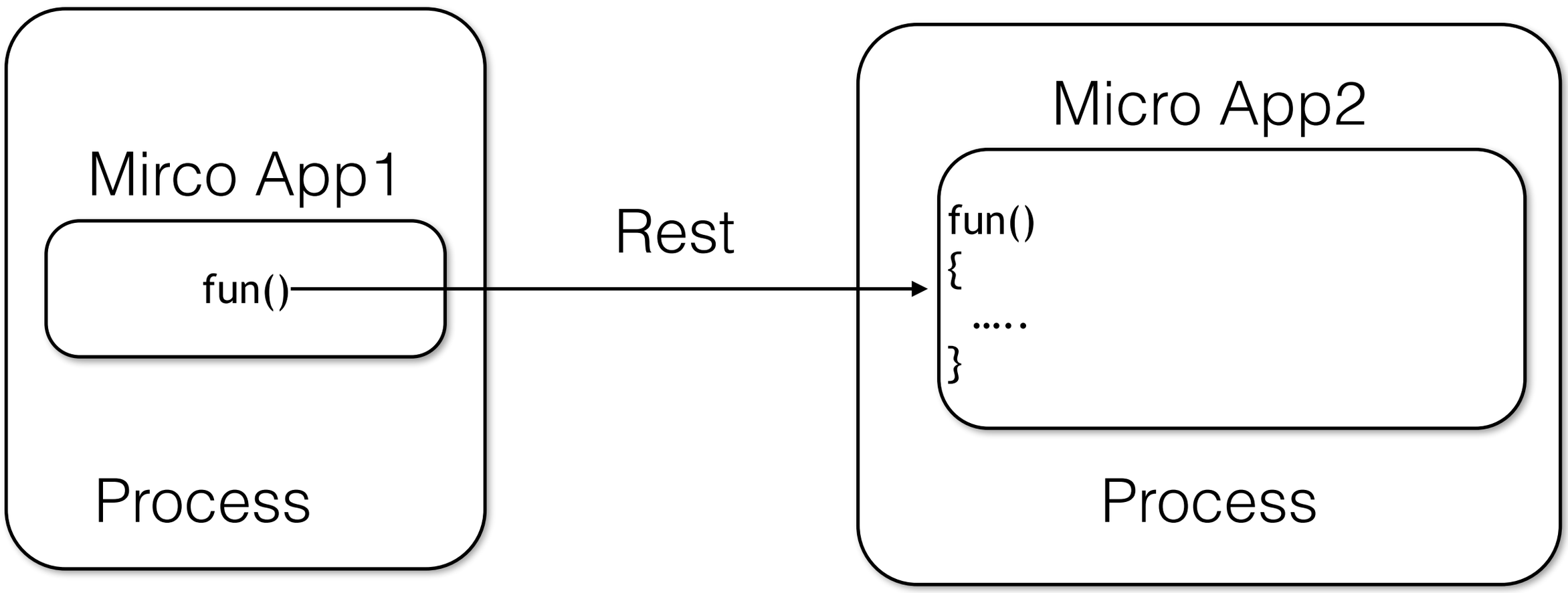


3. Query

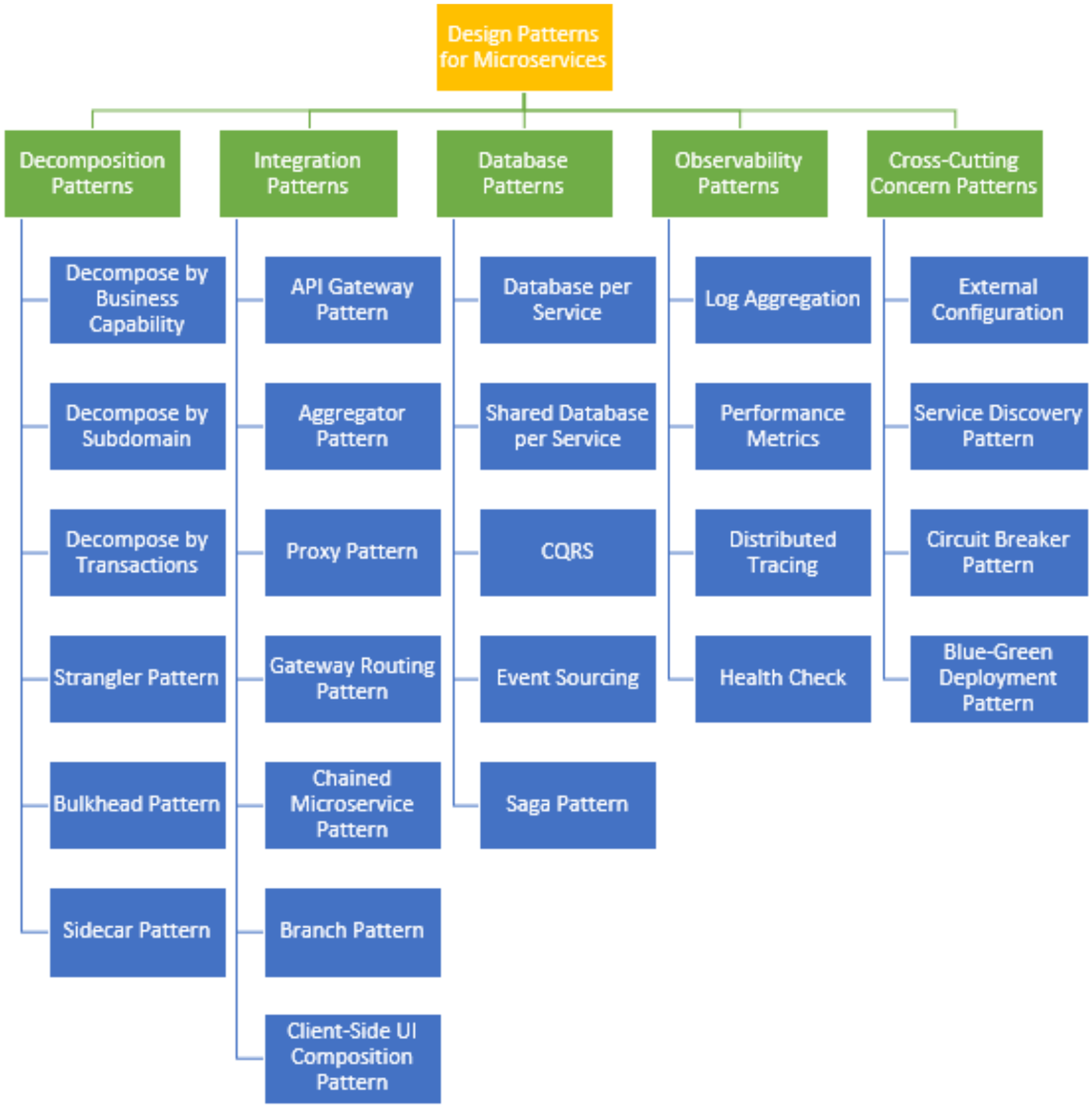


4. Development time

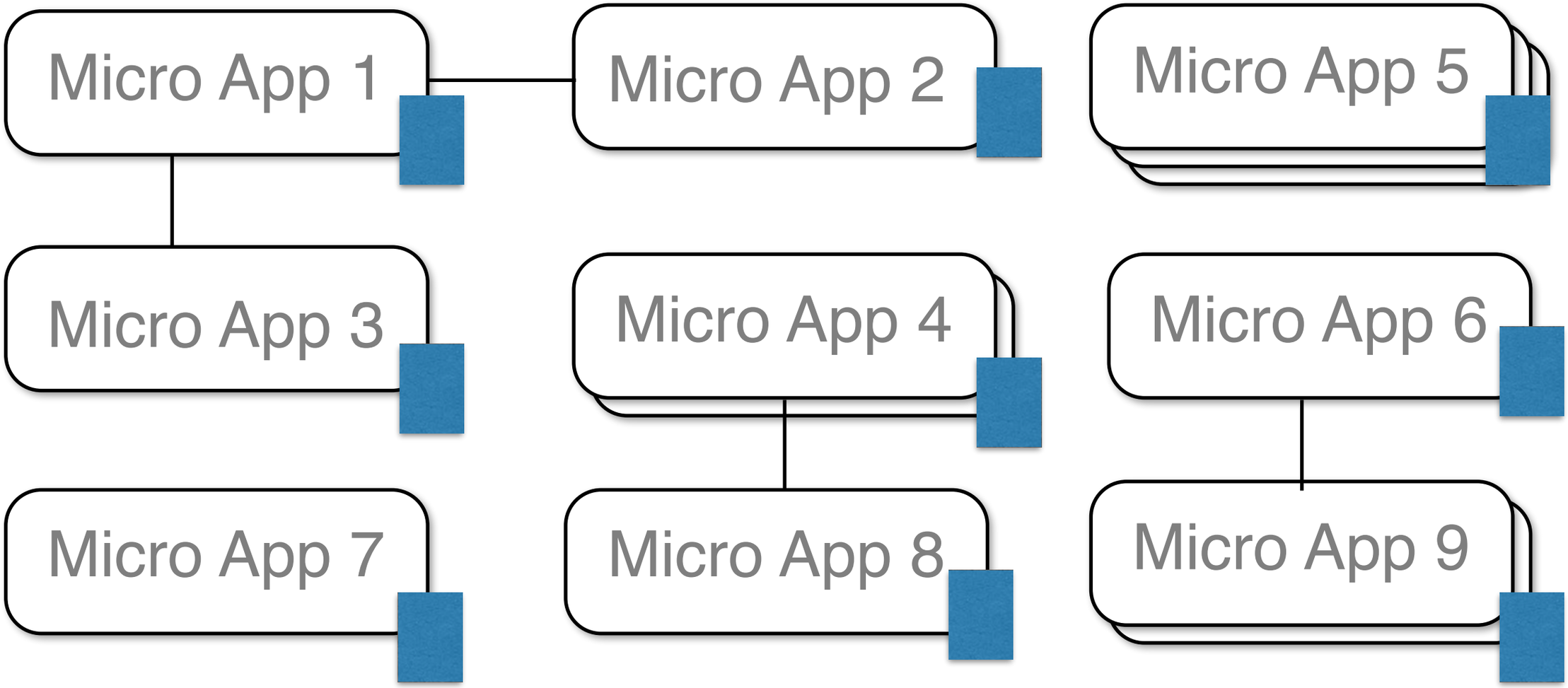




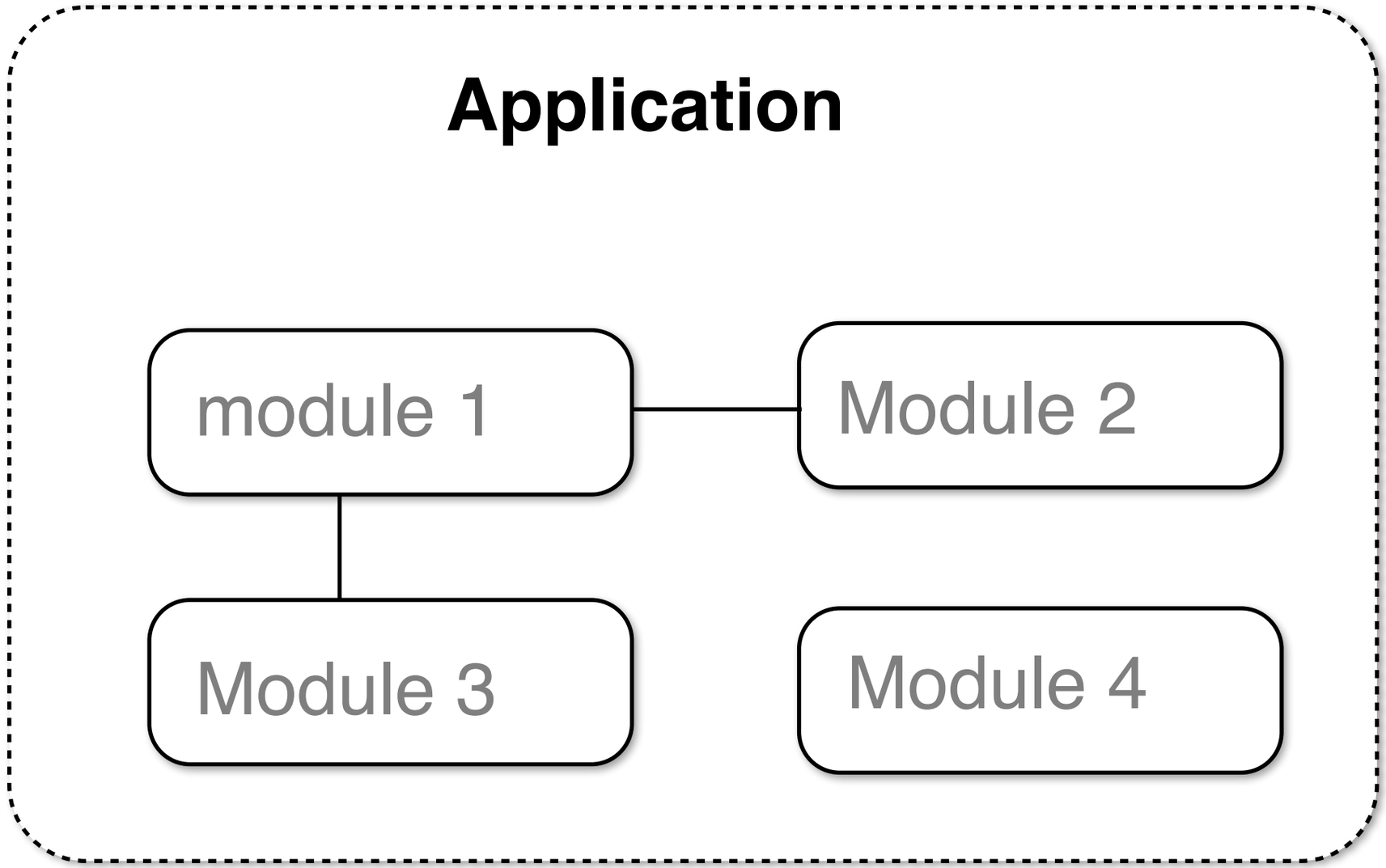
5. Learning Curve

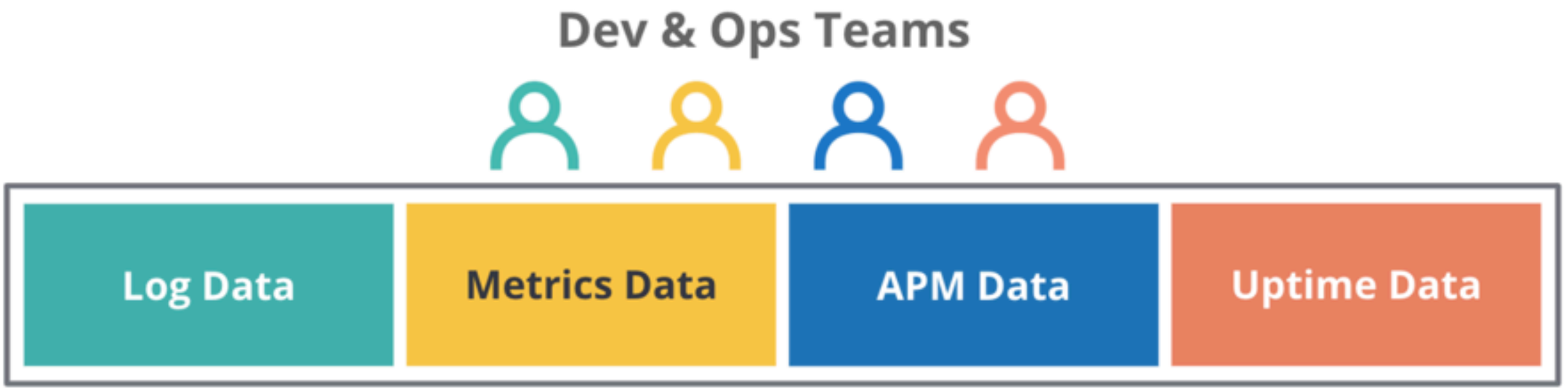


6. Infra Cost

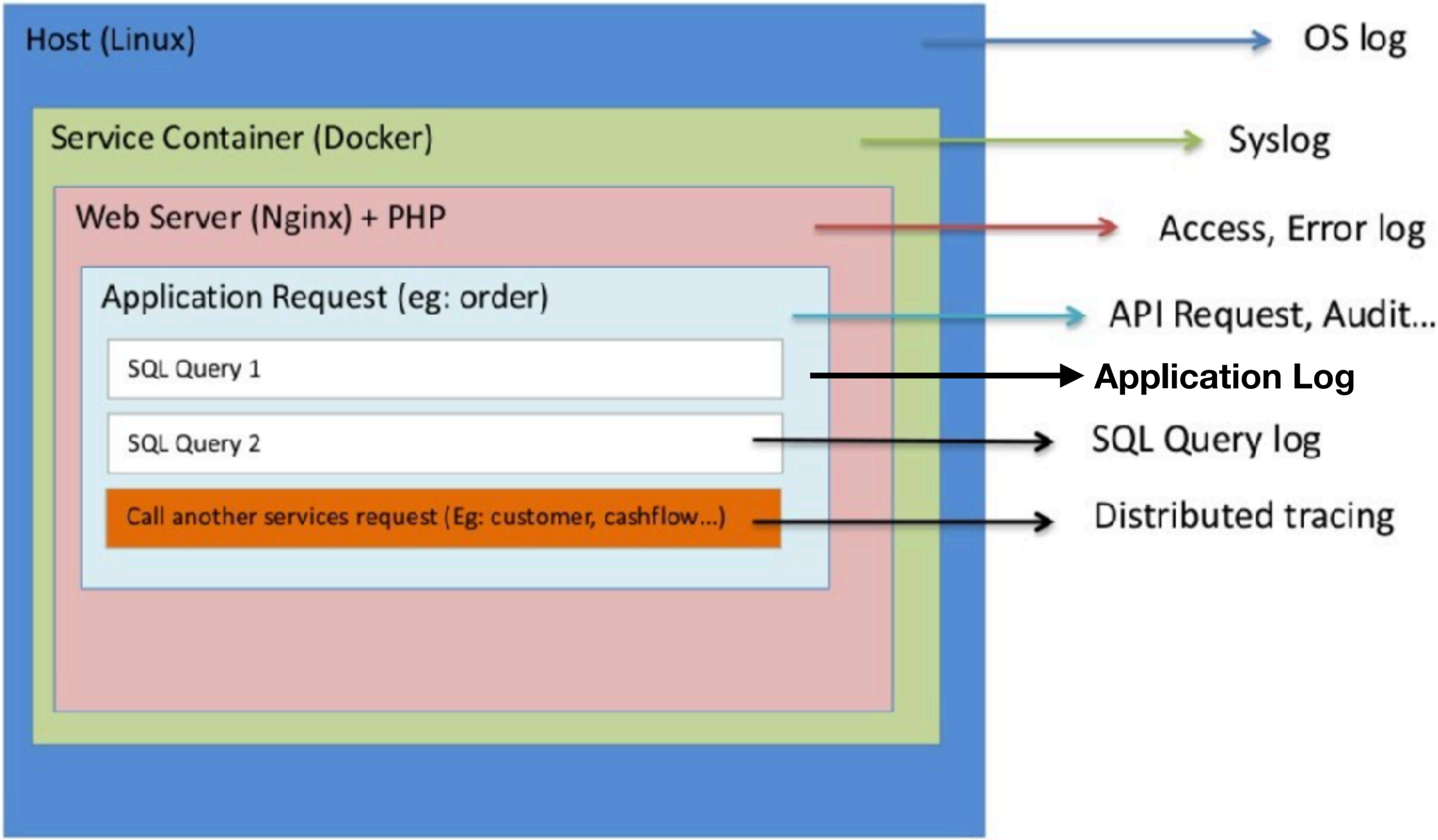

Micro App 1
Micro App 2
Micro App 5
Micro App 3
Micro App 4
Micro App 6
Micro App 7
Micro App 8
Micro App 9

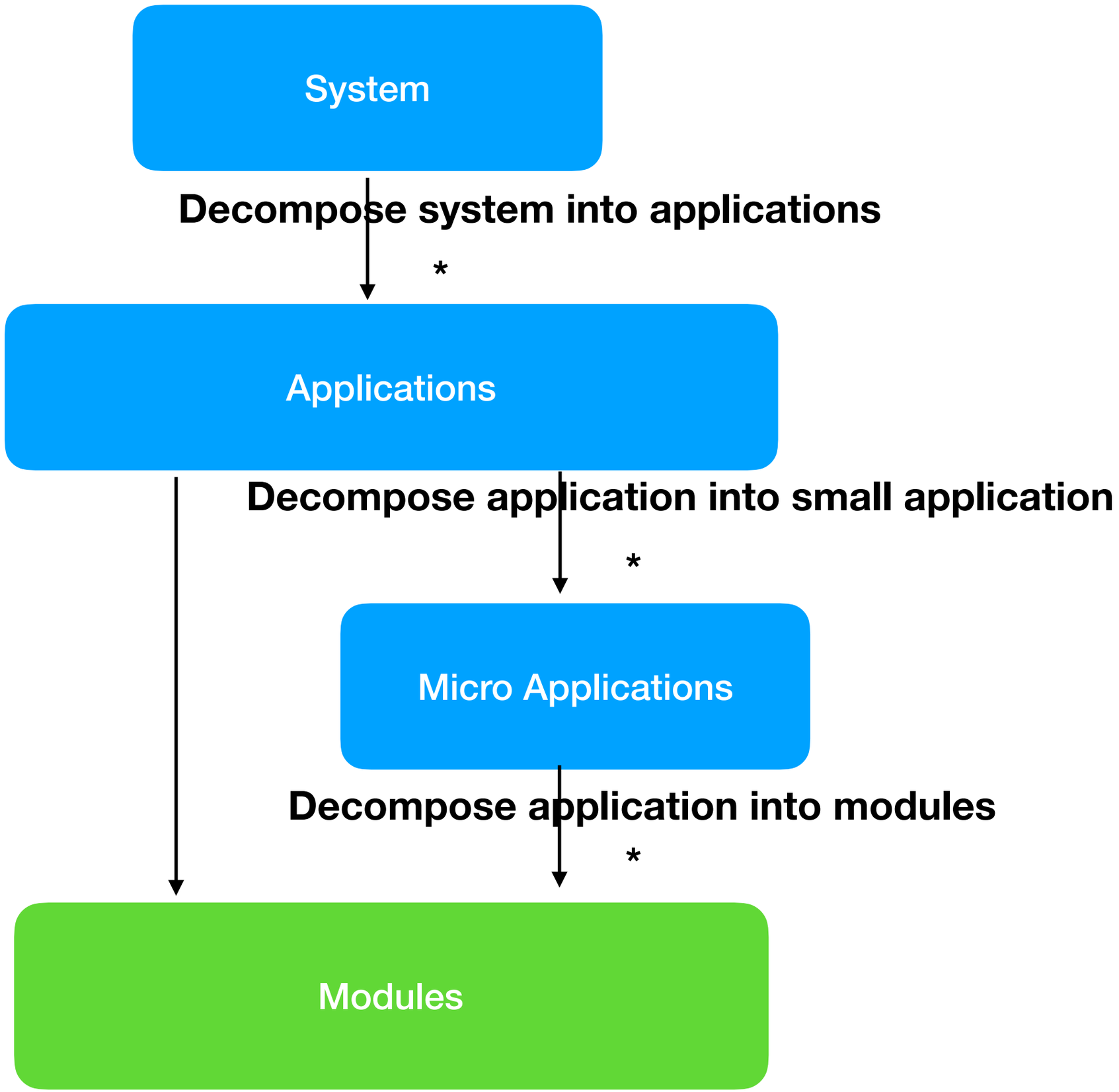
7. Debugging



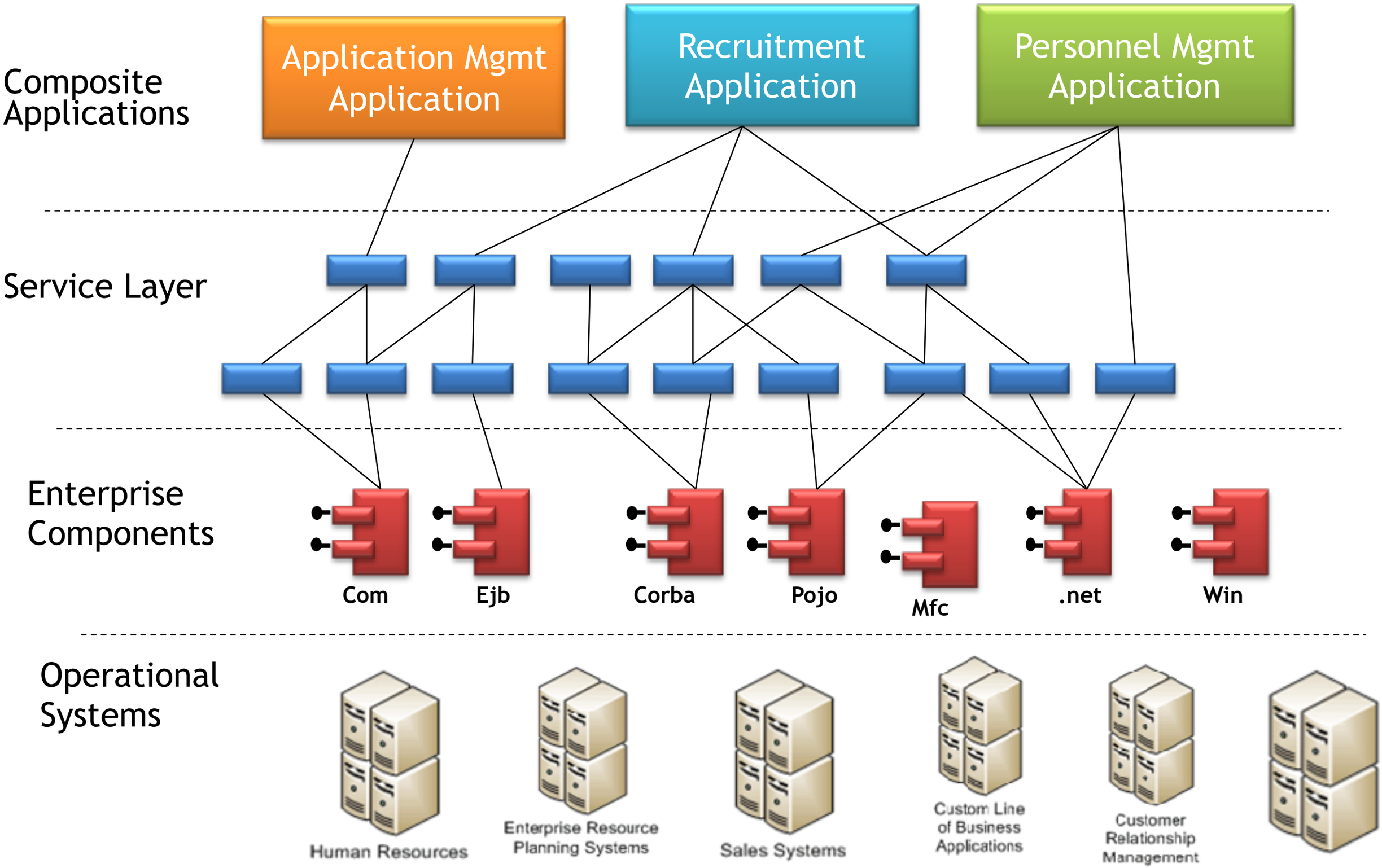


Web Logs	Container Metrics	Real User Monitoring	Uptime
App Logs	Host Metrics	Txn Perf Monitoring	Response Time
Database Logs	Database Metics	Distributed Tracing	
Container Logs	Network Metrics		
	Storage Metrics		


Host (Linux)
Service Container (Docker)
Web Server (Nginx) + PHP
Application Request (eg: order)
SQL Query 1
SQL Query 2
Call another services request (Eg: customer, cashflow...)
OS log
Syslog
Access, Error log
API Request, Audit...
Application Log
SQL Query log
Distributed tracing

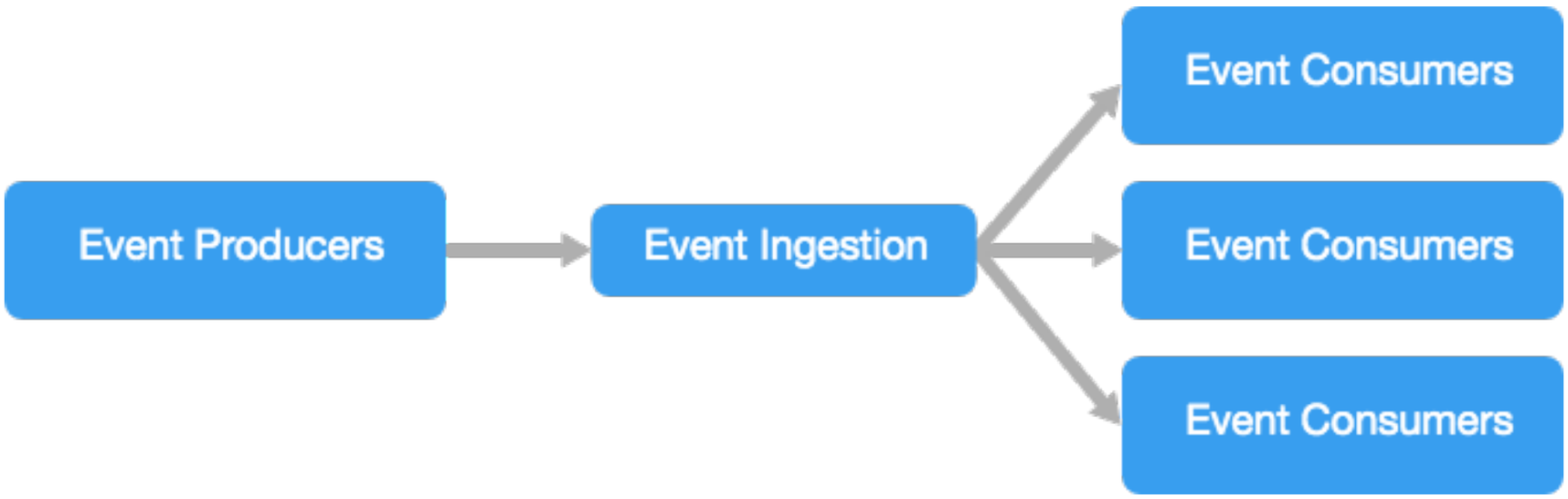

System
Decompose system into applications
*
Applications
Decompose application into small application
*
Micro Applications
Decompose application into modules
*
Modules

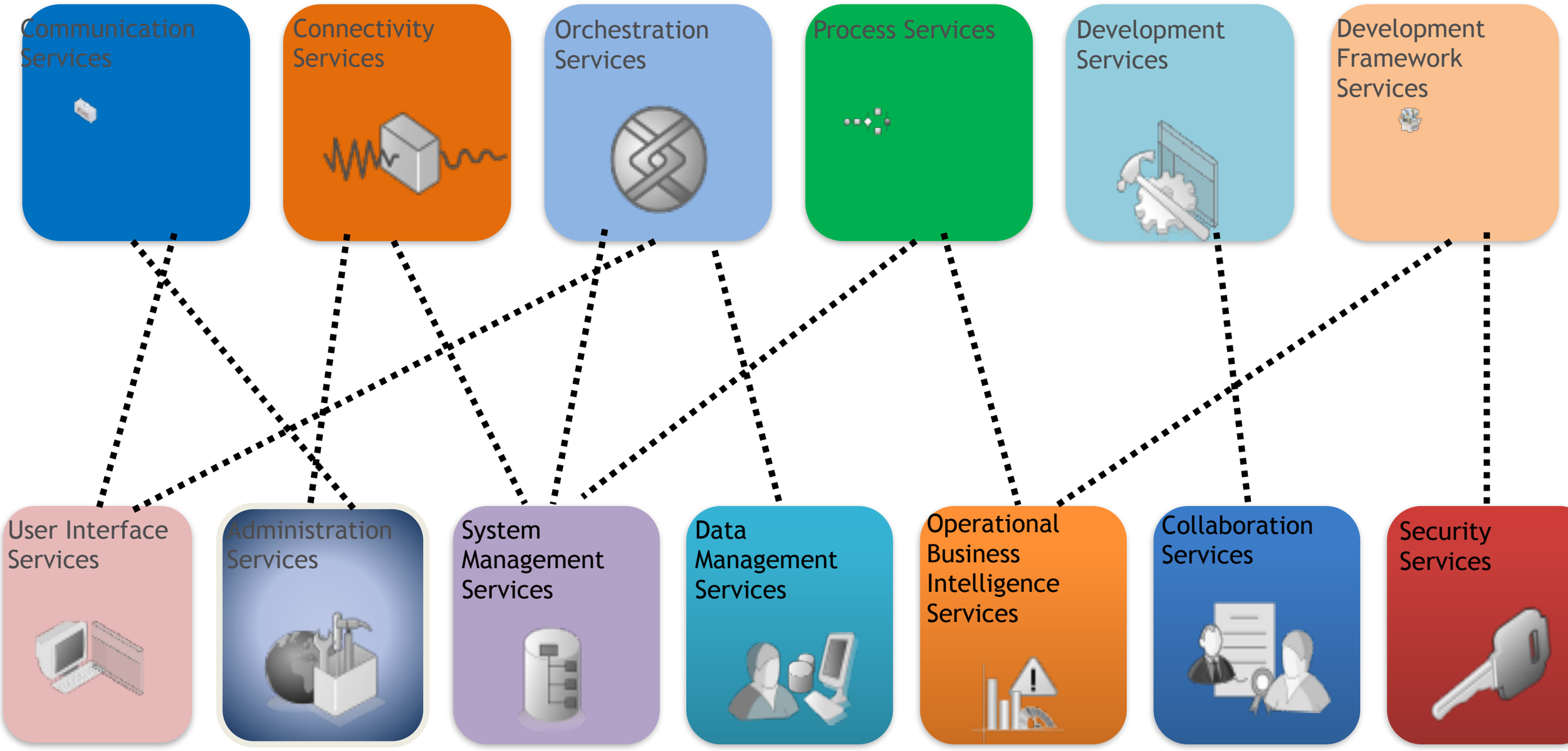
Service Oriented Architecture



- Almost all the successful microservice stories started with a monolith that got too big and was broken up.
- Almost all the cases where I've heard of a system that was built as a microservice system from scratch have ended up in serious trouble.

Event Driven Architecture




Communication Services
Connectivity Services
Orchestration Services
Process Services
Development Services
Development Framework Services
User Interface Services
Administration Services
System Management Services
Data Management Services
Operational Business Intelligence Services
Collaboration Services
Security Services

Communication Services

Connectivity Services

Orchestration Services

Process Services

Development Services

Development Framework Services

User Interface Services

Administration Services

System Management Services

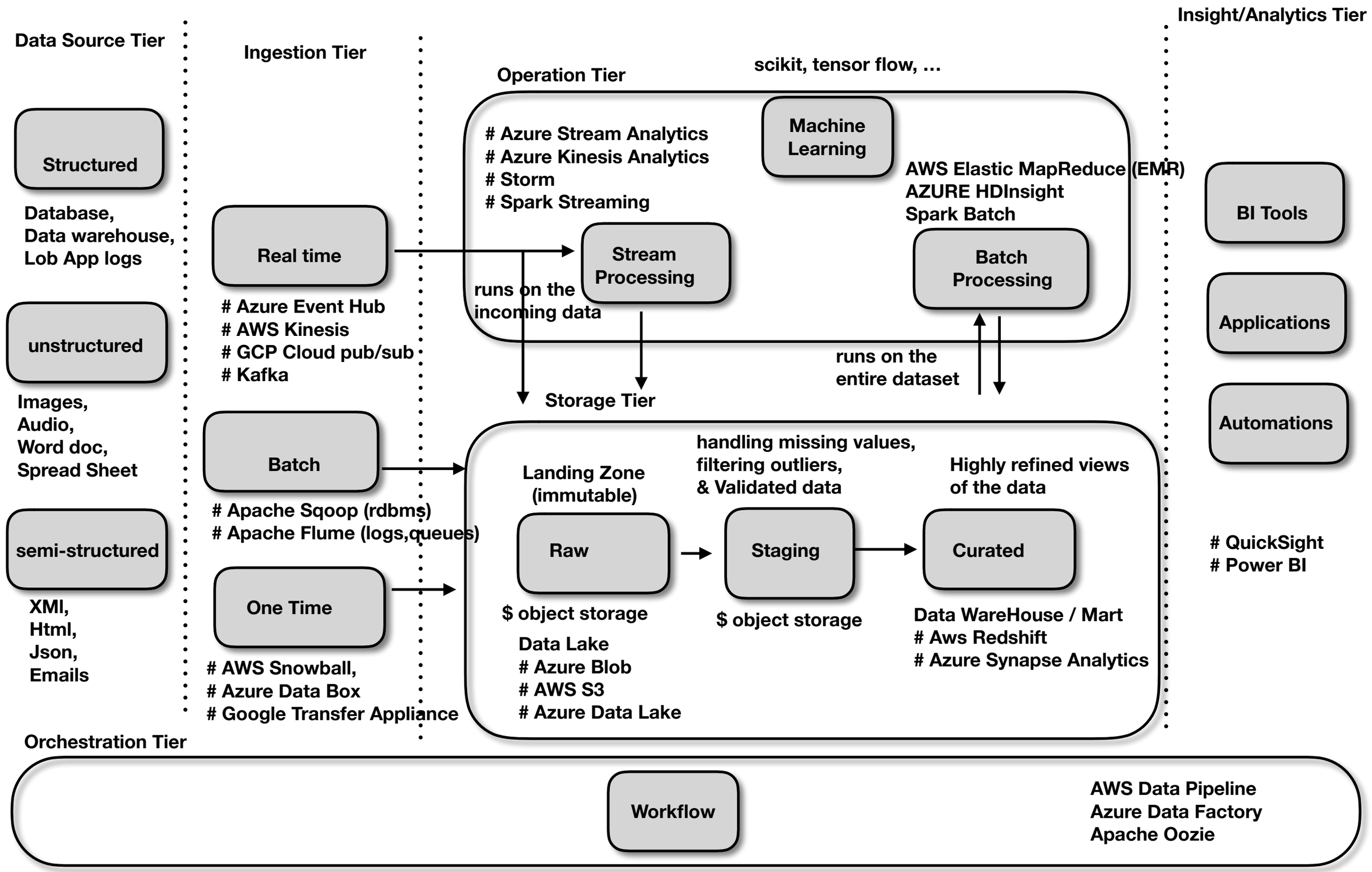
Data Management Services

Operational Business Intelligence Services

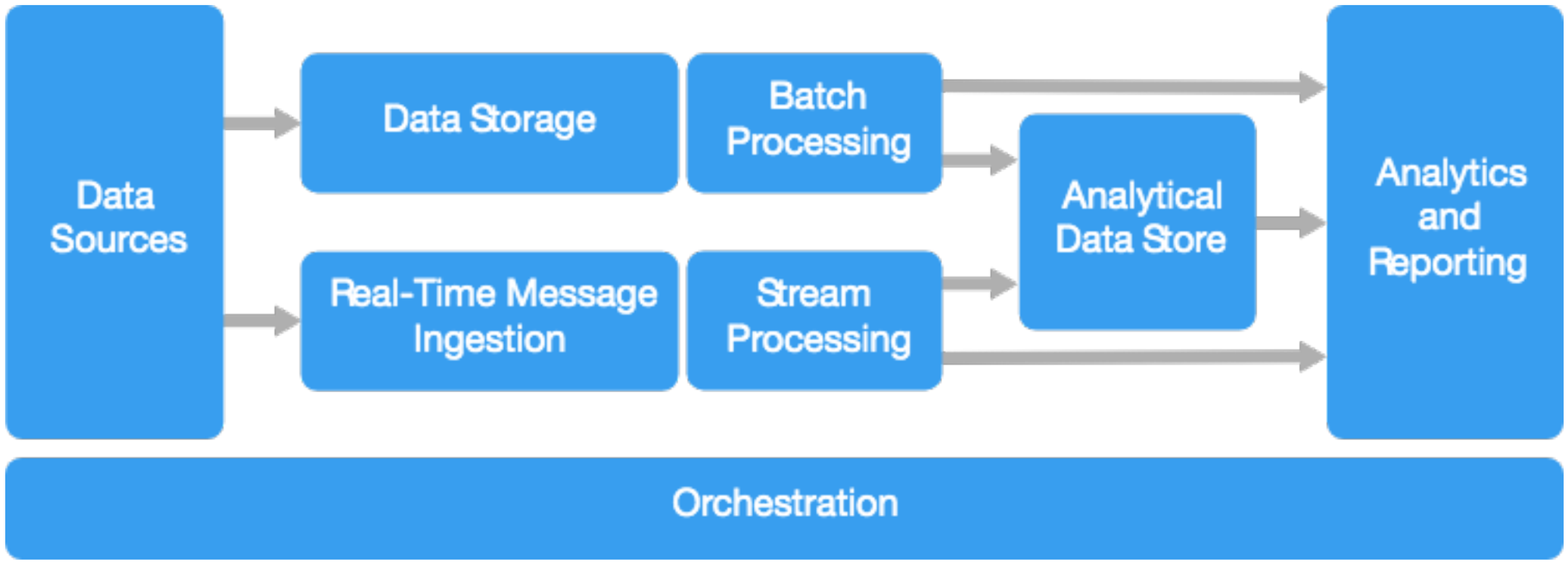
Collaboration Services

Security Services

BigData Reference Architecture



Big data architecture style



RAW DATA

DATA STORAGE

BATCH PROCESSING

PROCESSING DATA

BATCH LAYER

QUERY

DATA

VIEW 1

VIEW 2

VIEW 3

BATCH VIEWS

MERGED VIEW

PARTIAL AGGREGATIONS

REAL-TIME VIEWS

VIEW 1

VIEW 2

VIEW 3

SERVING LAYER

STREAM

DATA INCREMENT

REAL-TIME PROCESSING

PROCESSED INCREMENT

SPEED LAYER

SERVING LAYER

PARTIAL AGGREGATES

QUERY

REAL-TIME VIEWS

VIEW 1

VIEW 2

VIEW 3

DATA

STREAM

RAW DATA

IMMUTABLE DATA

LOG

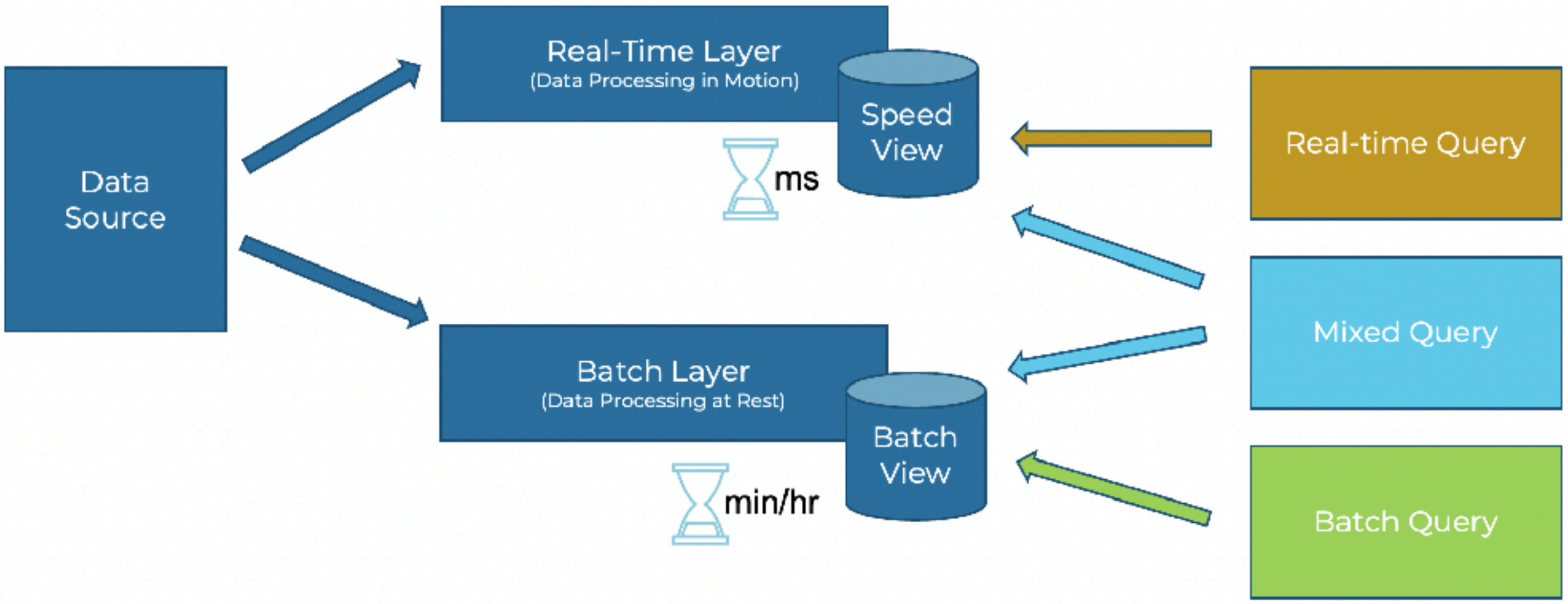
REAL-TIME PROCESSING

PROCESSED DATA

SPEED LAYER

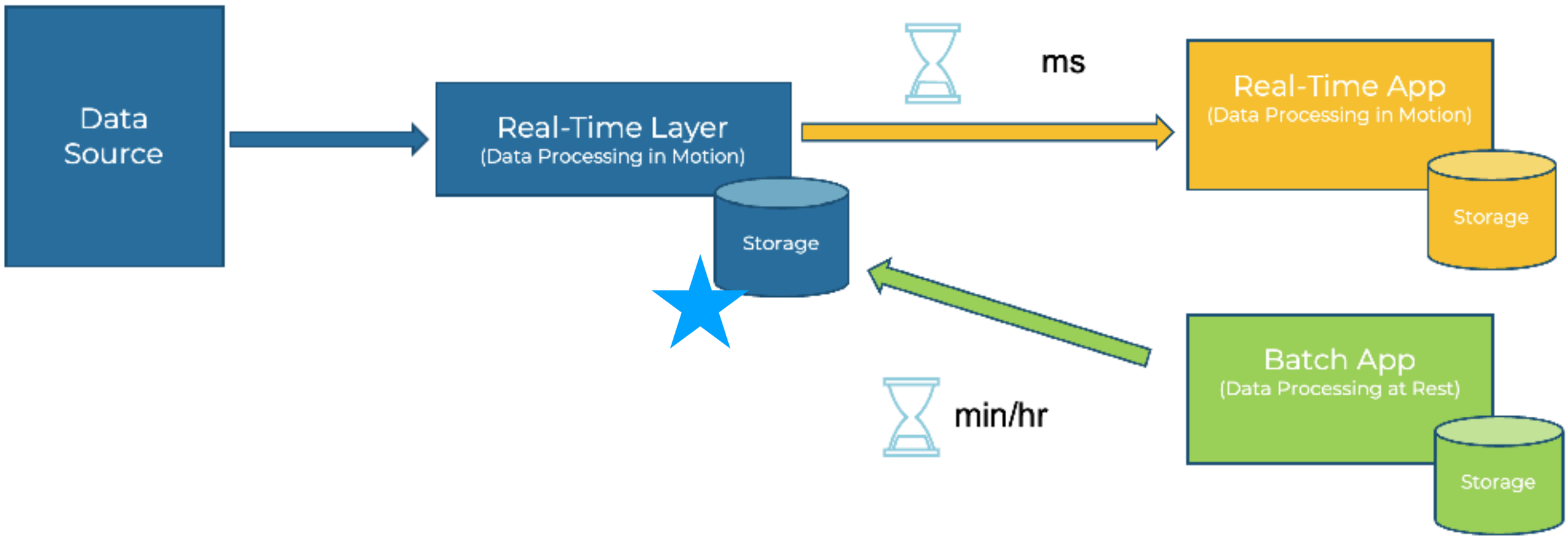
Lambda Architecture

Option 2: Separate serving layers

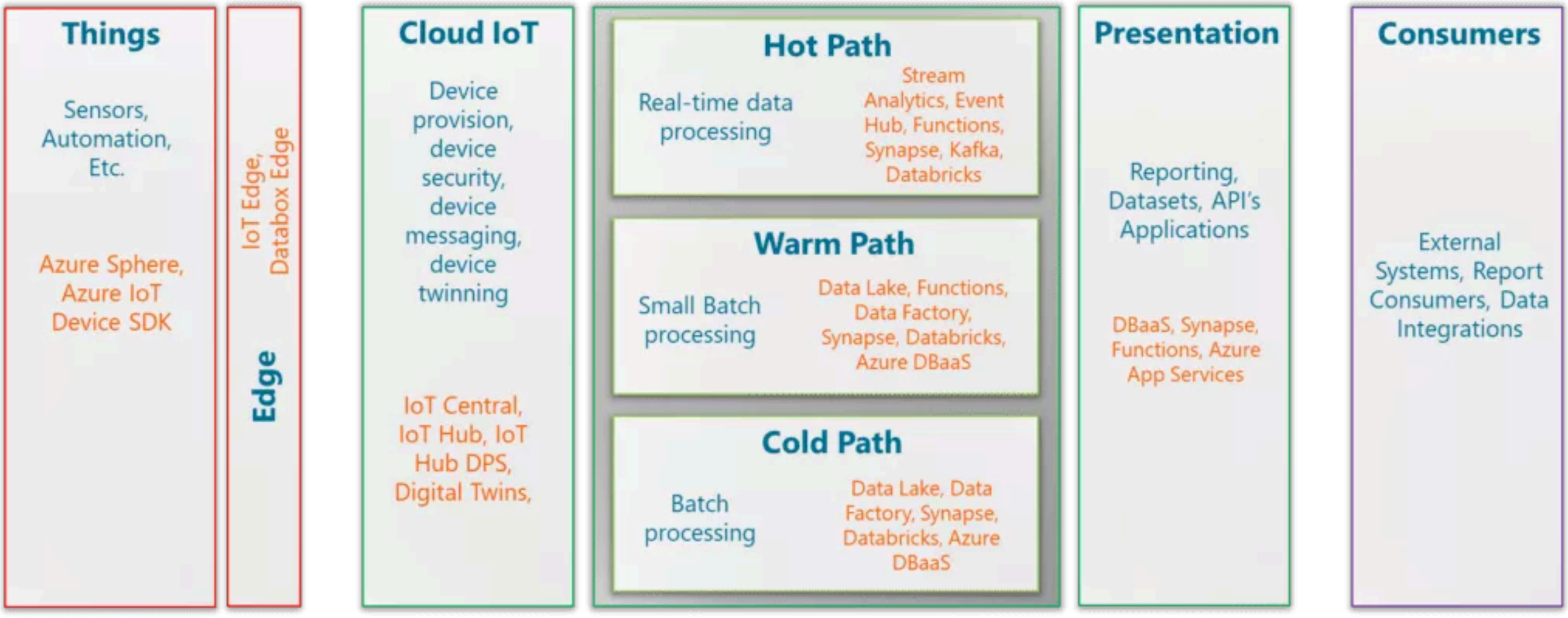


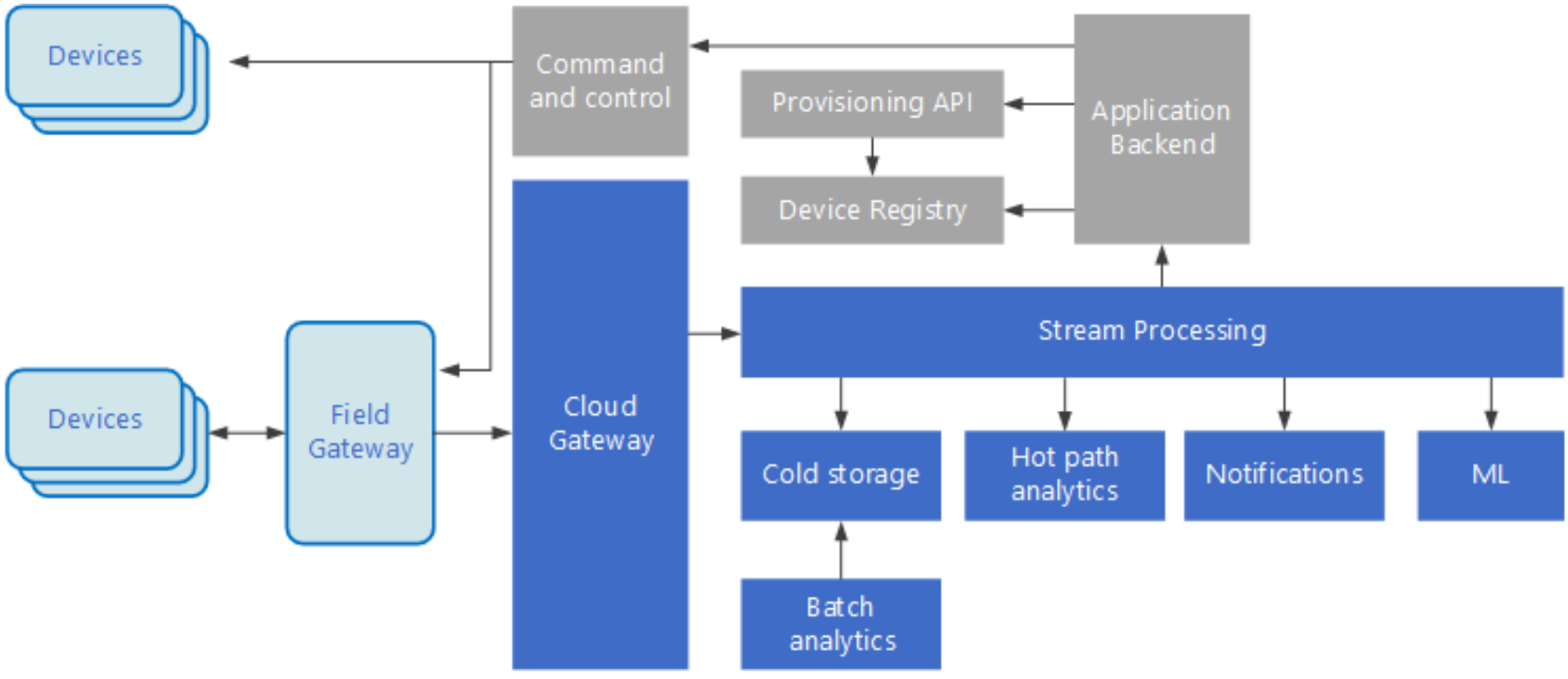
Kappa Architecture

One pipeline for real-time and batch consumers

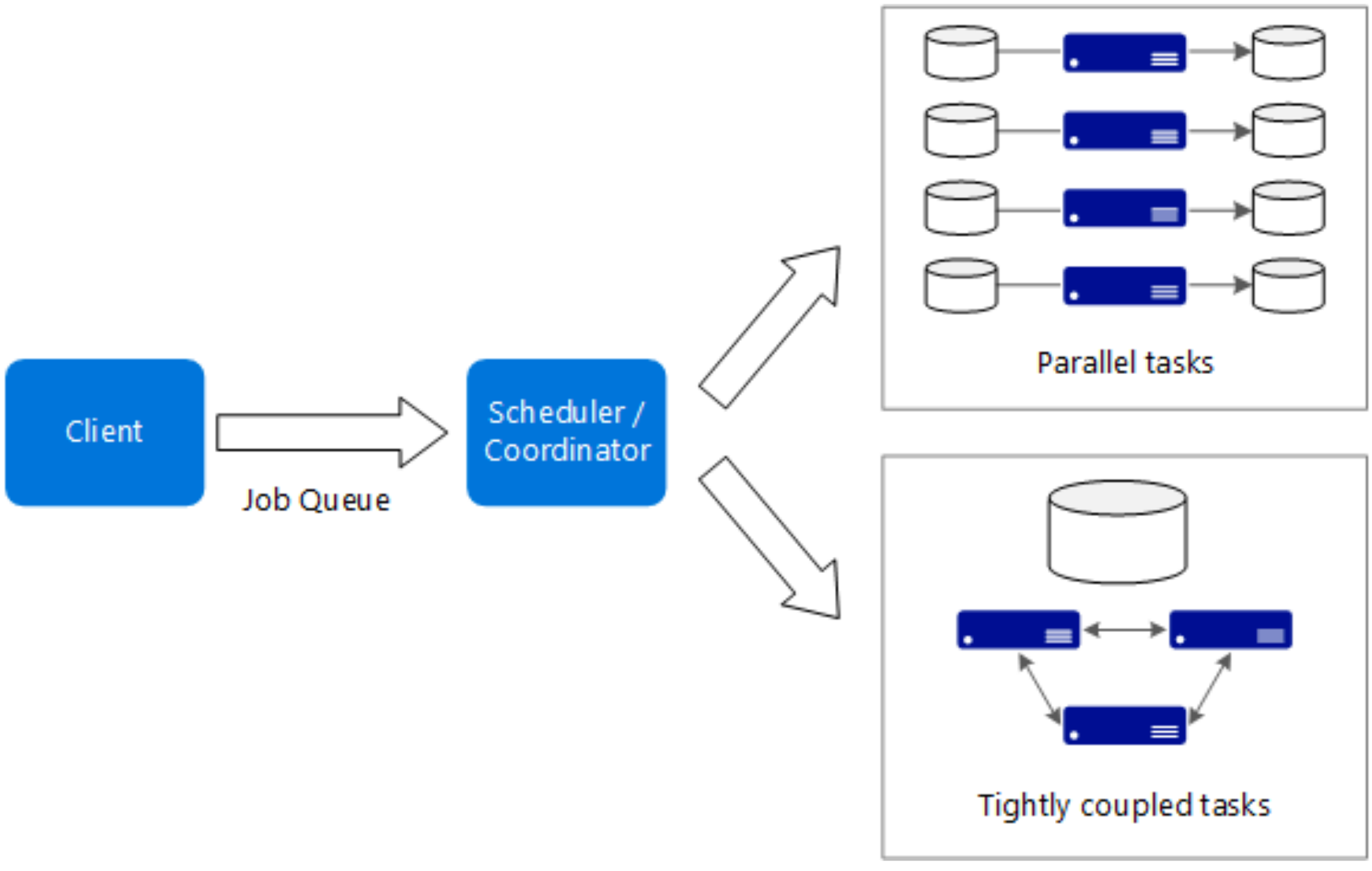


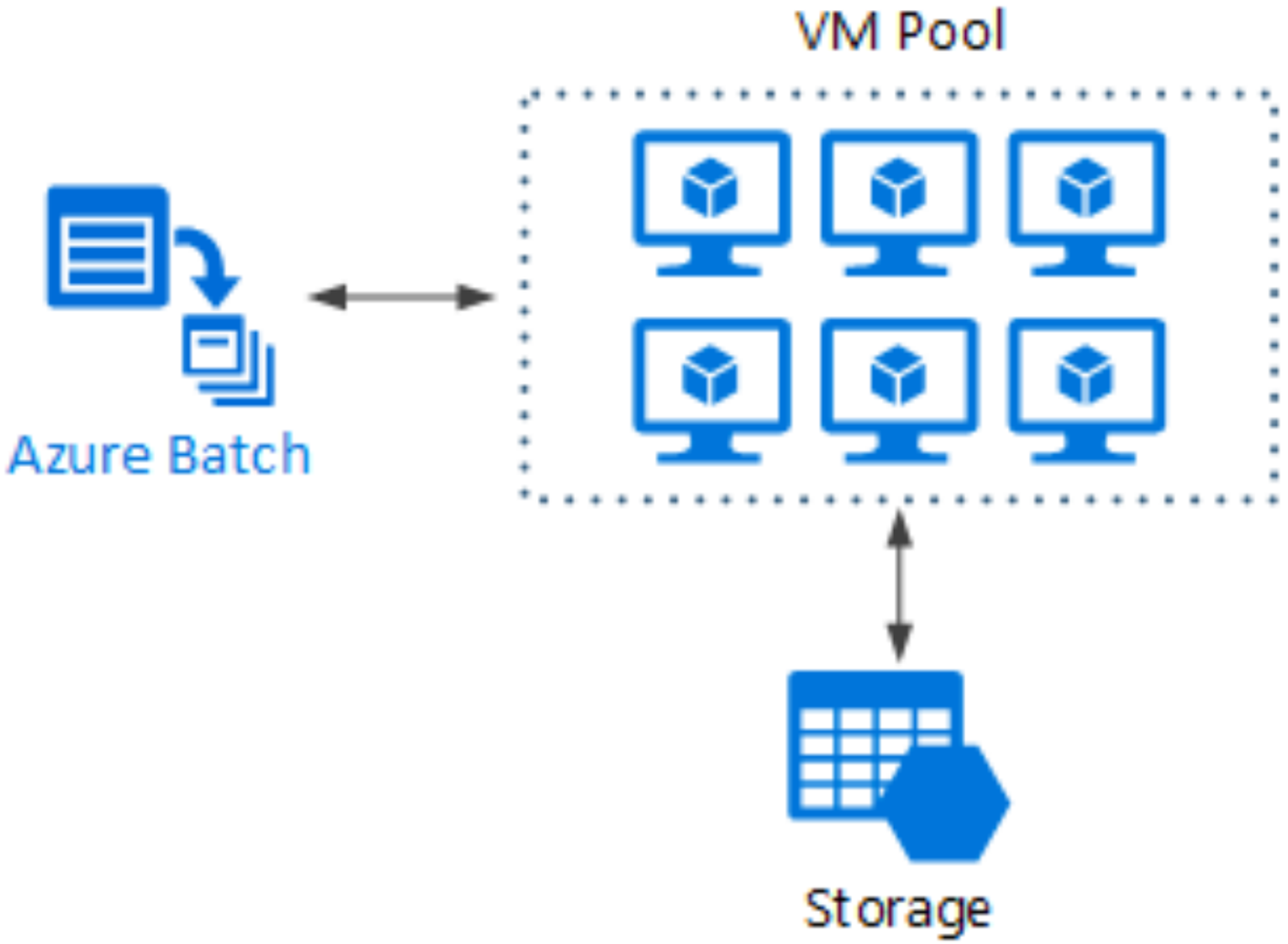
IOT Architecture



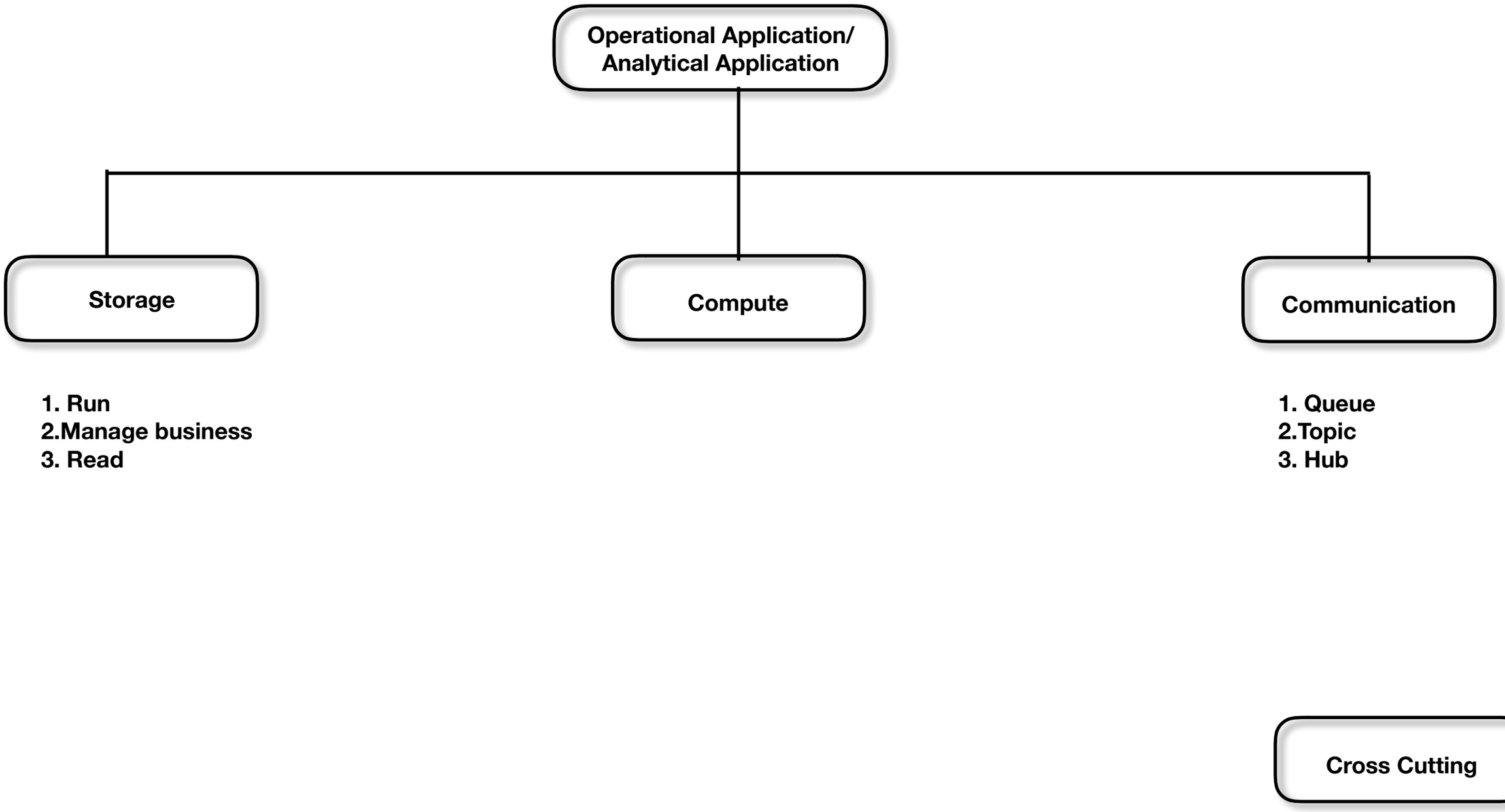

Devices
Command and control
Provisioning API
Application Backend
Device Registry
Stream Processing
Devices
Field Gateway
Cloud Gateway
Cold storage
Hot path analytics
Notifications
ML
Batch analytics

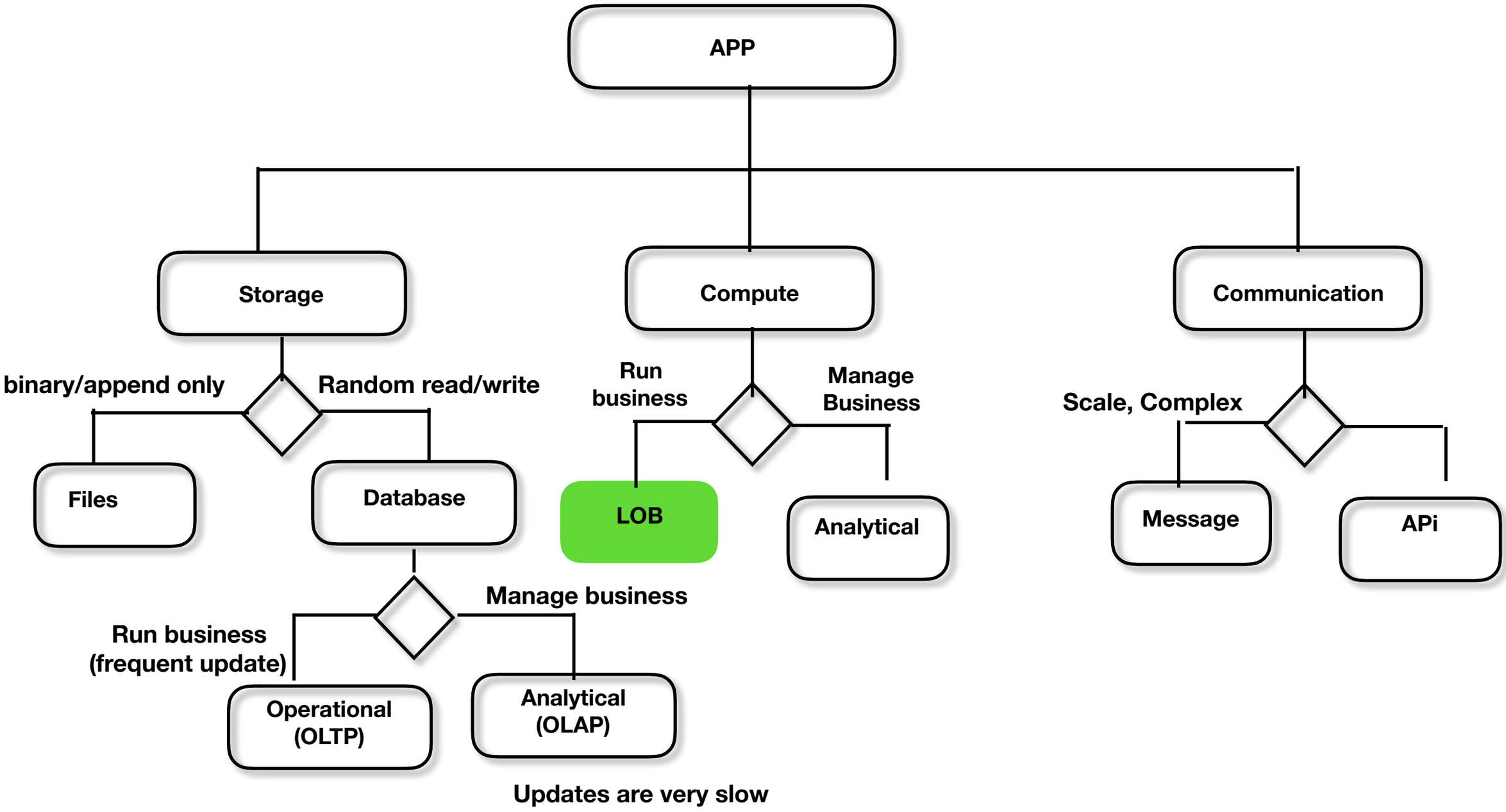
Big compute architecture style



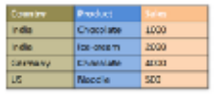

VM Pool
Azure Batch
Storage

Choosing Technology ?




APP
Storage
Compute
Communication
binary/append only
Random read/write
Run business
Manage Business
Scale, Complex
Files
Database
LOB
Analytical
Message
APi
Manage business
Run business (frequent update)
Operational (OLTP)
Analytical (OLAP)
Updates are very slow

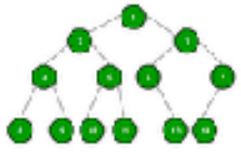
Shard *



Rdbms

- Postgres

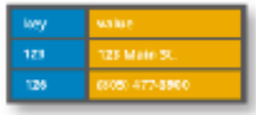
Shard



Document

- Mongo

Shard



Key Value

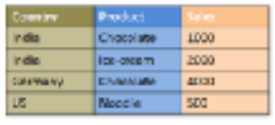
- Azure Table

Shard

Graph

- Neo4j

Split Shard



Wide Column

- Casandra

time series
geo spatial
audit

Columnar (DWH/OLAP)

- Synapse

Text

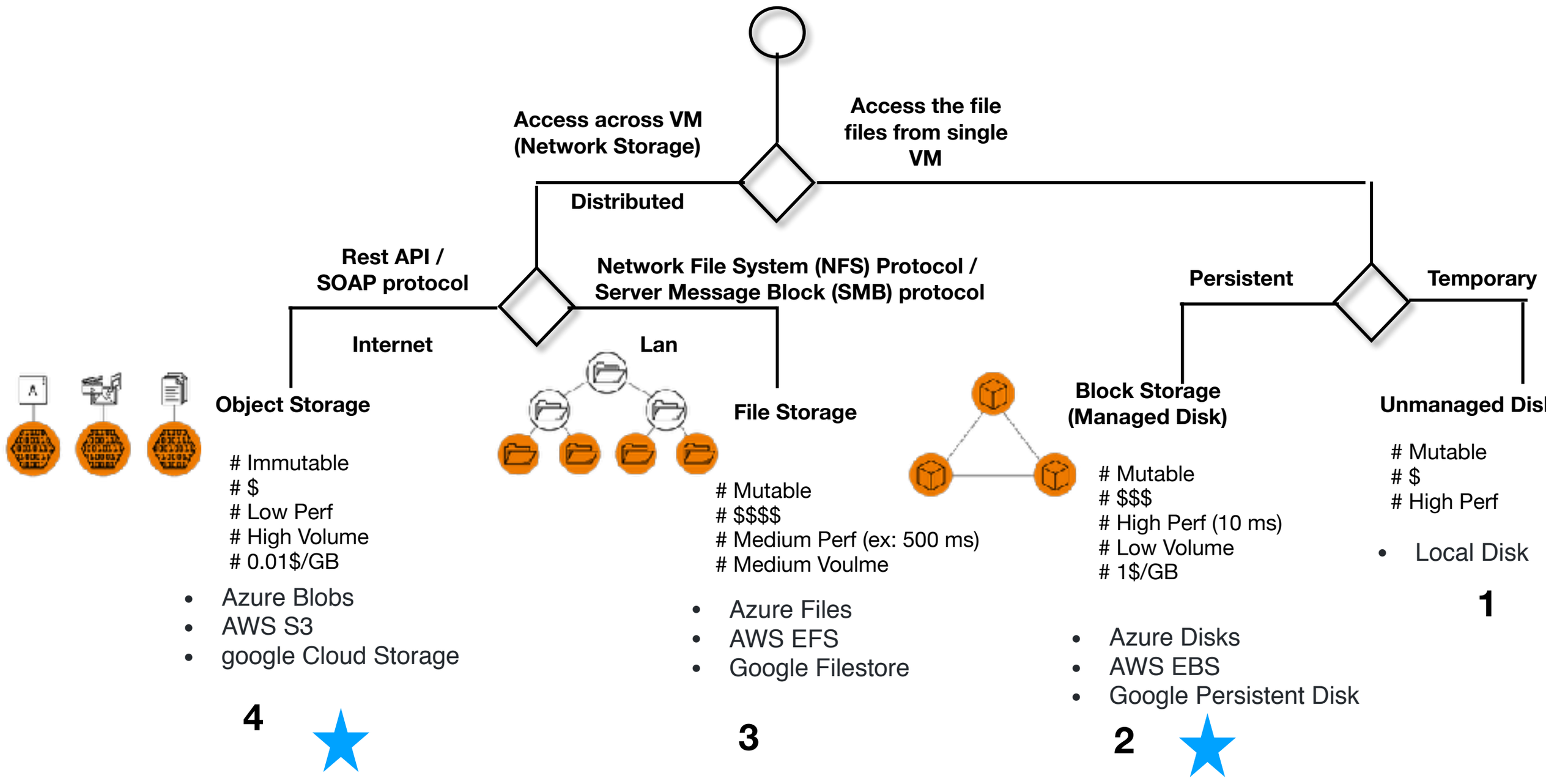
- Elastic Search

ELK *
EFK *
Splunk

Data Lake

- S3/ blob

Binary Storage



* AWS DataSync subscription is required to provide support for Server Message Block (SMB) protocol

Operational

Structured

+ ACID - transaction,
- LIMITED SCALABLE - not distributed
- Fixed Schema
CAP - CA

RDBMS (OLTP)

- AWS RDS
- Azure SQL
- Google Cloud Sql
- Google Cloud Spanner
- **Postgres**
- **Mysql**

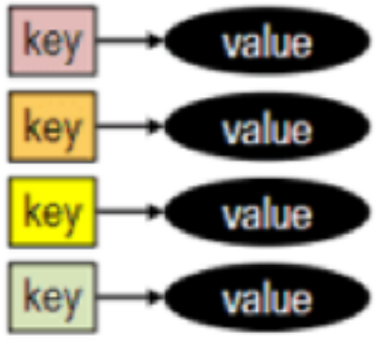
1

Semi Structured

- BASE,
+ SCALE- shareded (distributed),
+ Flexible Schema
CAP - CP | AP

Dictionary (key value)

- AWS Dynamo Db
- Azure Cosmos -> Table
- Azure Table
- Google Big Table



2

SHARD

Nested Objects

Document / Tabular

- AWS Document DB
- Azure Cosmos -> Document Db
- Firestore
- **Mongodb**, **Couchbase**,Solr

3

SHARD

Entity Relationship

Graph

- AWS Neptune
- Azure Cosmos -> Graph
- Google Janusgraph
- **Neo4j**
- Graph,orientdb

4

SHARD

Wide Column db (column-family)

frequently added but rarely updated.

- AWS Keyspaces (unlimited)
- Cosmos
- Google Big Table
- **Hbase, cassandra**
- Scylla

5

SPLIT & SHARD

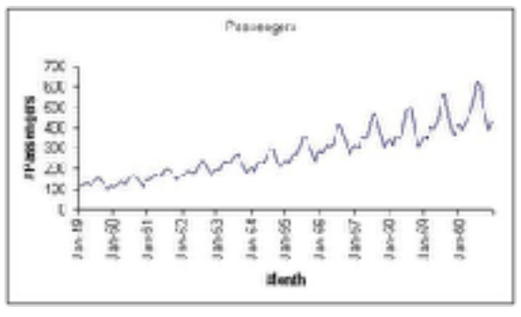
Wide Column db (column-family)

- AWS Keyspaces (unlimited)
- Cosmos
- Google Big Table
- **Hbase, cassandra**
- Scylla

KairosDB (cassandra)

Time Series

- AWS Timestream
- InfluxDB, Scylla
- OpenTSDB,
- KairosDB
-



Audit Trial

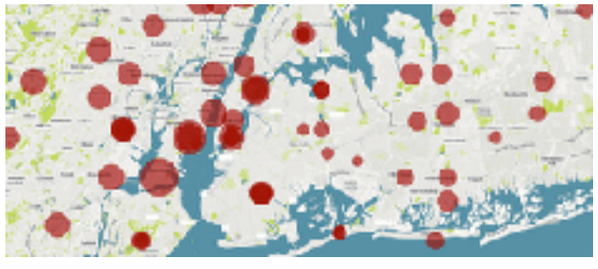
Immutable Logger

- AWS Quantum Ledger
- Azure SQL Database Ledger
- Google ?
- Hyper ledger Fabric

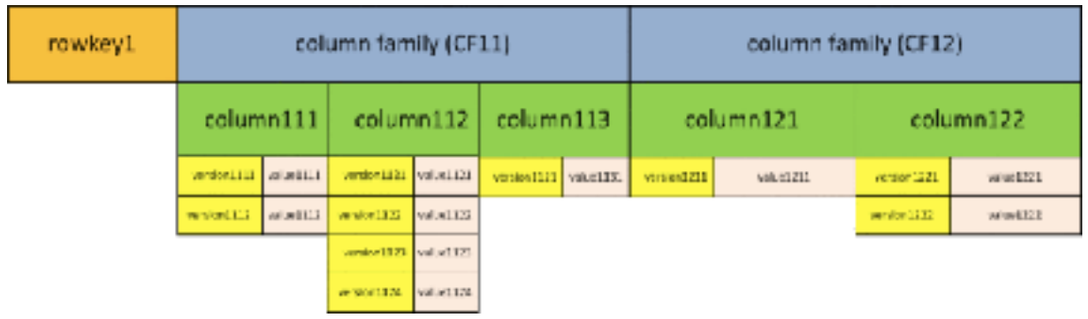
LIZARD GLOBAL

geo Spatial

- AWS Keyspaces
- Azure Cosmos
- Big Table, Big Query
- **Solr**, PostGIS
- Mongoldb (GeoJSON)



Analytical

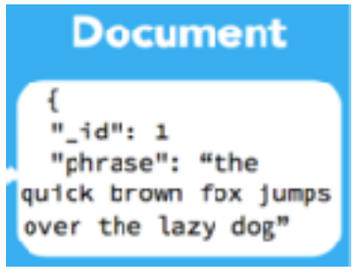


Read

Columnar db (column store)

Cloud DWH (OLAP)

- AWS Redshit (max: 8 PB)
- **Azure Synapse**
- GCP BigQuery
- Snow flakes
- Kudu, Druid, Pinot
- Click house



Read

Text Database

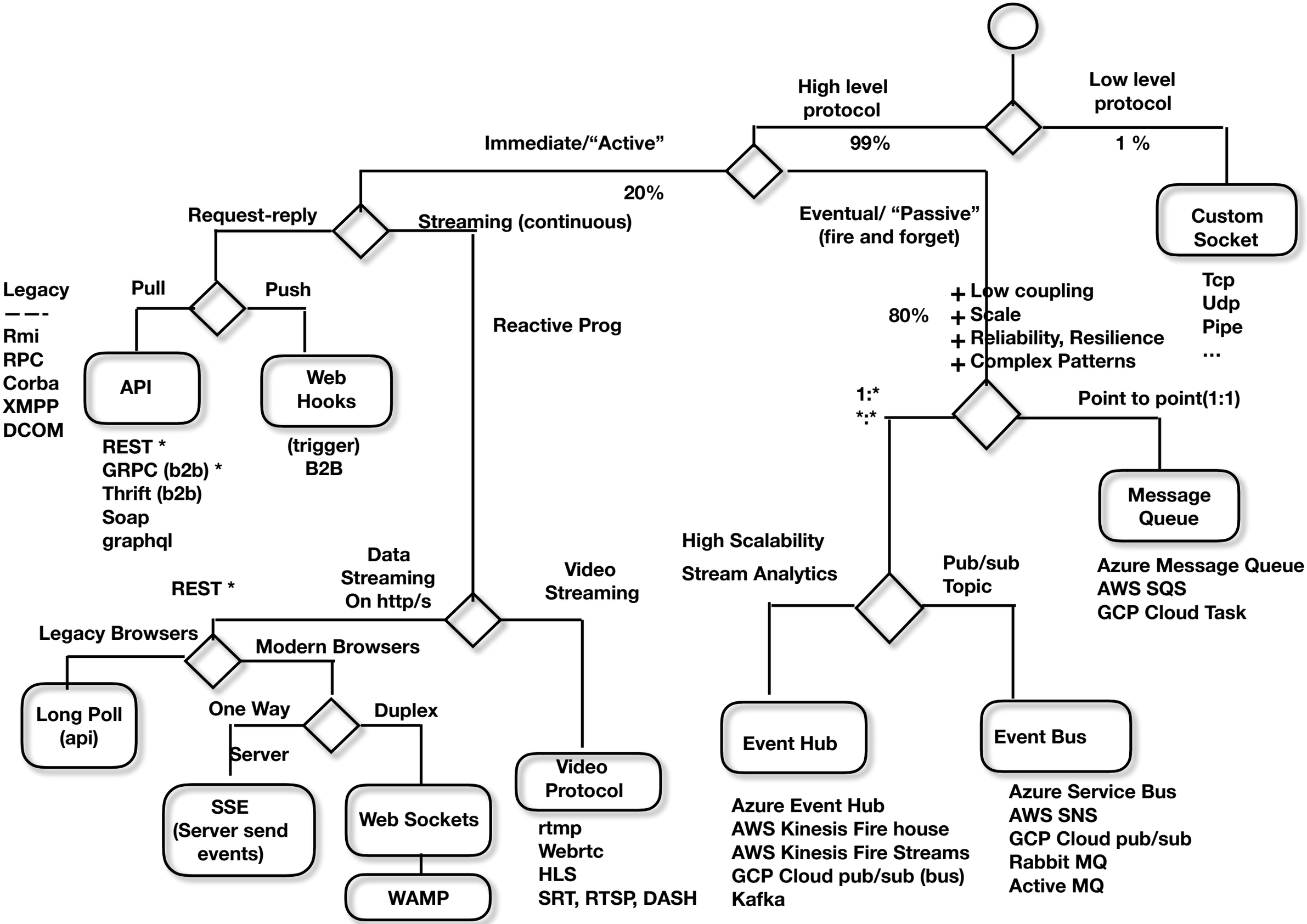
- AWS **Elastic Search**
- AWS cloud Search
- Azure Cognitive Search
- Google Search API
- Elastic Search, **Solr**

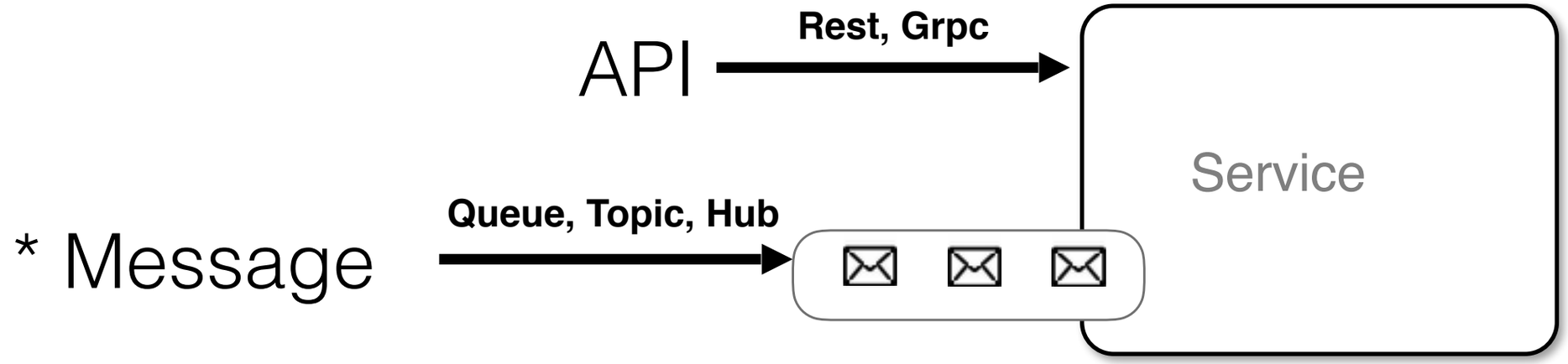
Read/write

Wide Column db (column-family)

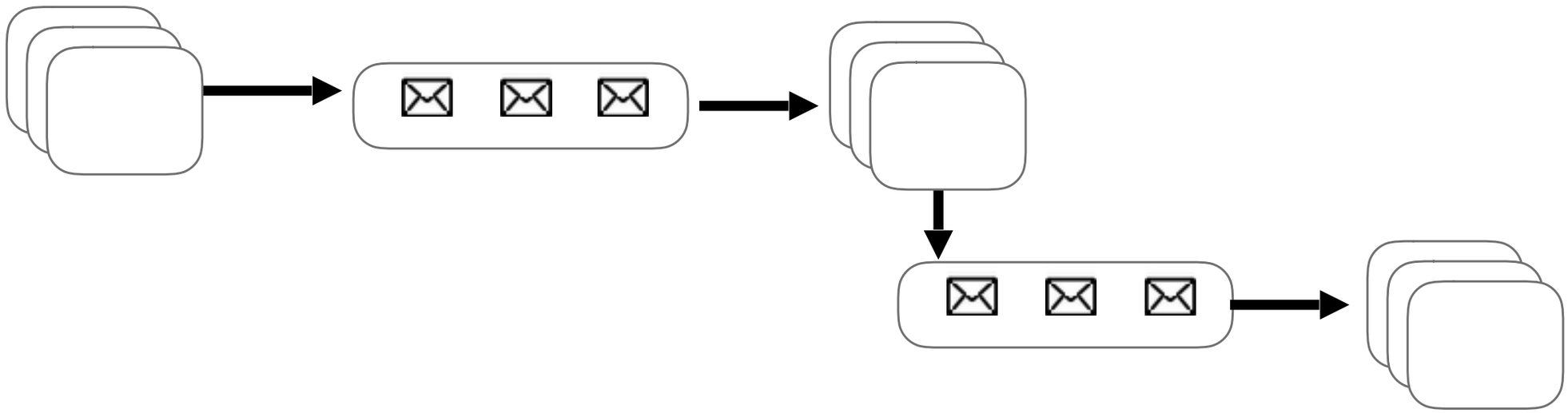
- AWS Keyspaces (unlimited)
- Cosmos
- Google Big Table
- **Hbase, cassandra**
- Scylla

Choose Communication protocol





	<< API >>	<< Message >>	
Ordering	Ordered (+)	Unordered(-)	
Duplicate	Yes (-)	Yes(-)	Idempotency
Protocol	2 way (+)	One way (-)	
Resilence (recover)	No (-) retry logic	Yes (+)	
Connection	Always connected	Occousionaly connected	
Scalability	- -	+ + +	
Consistency	Immediate (++)	Eventual (- -)	
Load Leveling	- -	+ +	
Low Coupling (maintainability)	- -	+ +	
Distributed Comm Patterns	- -	+ +	SAGA, Materialized View
Internet	Yes	Yes *	
Browser support	Yes	No	
Dev effort	++	- -	
Operational effort	+ +	- -	



p2p
 # SQS
 # MSMQ
 # …

Publisher

Publisher

Queue

Payroll Subscriber 1

Payroll Subscriber 2

pubSub
 # RabbitMQ
 # SNS
 # Redis
 # MQ Series

Publisher

Publisher

Topic

Tax Subscriber 1

Payroll Subscriber 2

Payroll Subscriber 3

T3 T2 T1

Hub

Publisher

Tax Subscriber

State P1 :t

Tax Subscriber

State p2: t

Payroll Subscriber

State P1:t

 # Kafka
 # Event Hub
 # Kinesis

Parallelism

Data Parallelism

 # same code
 # different data

process salary of all employees

Task Parallelism

 # different code
 # different data

When Order is created,
Dispatch the order
Update RO Level

Choose Operational Compute

API App/ Web App / Background App

Containers?

Yes

No (Legacy)

Distributed ?

Yes

No

Paas ?

Yes

No

High Frequency of usage

Simple Use case

Yes

High Frequency of usage

No

Yes

High Frequency of usage

No

Yes

IAAS

- Azure VM
- EC2
- Google Compute Engine

1

Serverless Kubernetes

- Azure Container Instance
- AWS Fargate for Amazon ECS
- Google Cloud Run

5

Serverless Container (PAAS 2.0)

- Azure Functions
- AWS Lambda
- Google Cloud Function

3

Serverless Webserver (PAAS 2.0)

- Azure Functions
- AWS Lambda
- Google Cloud Function

3

Managed Kubernetes (PAAS 1.0)

- Azure kubernetes Service
- Amazon Elastic Kubernetes Service (Amazon EKS)
- Amazon Elastic Container Service
- Google Kubernetes Engine (GKE)

4

Managed Container (PAAS 1.0)

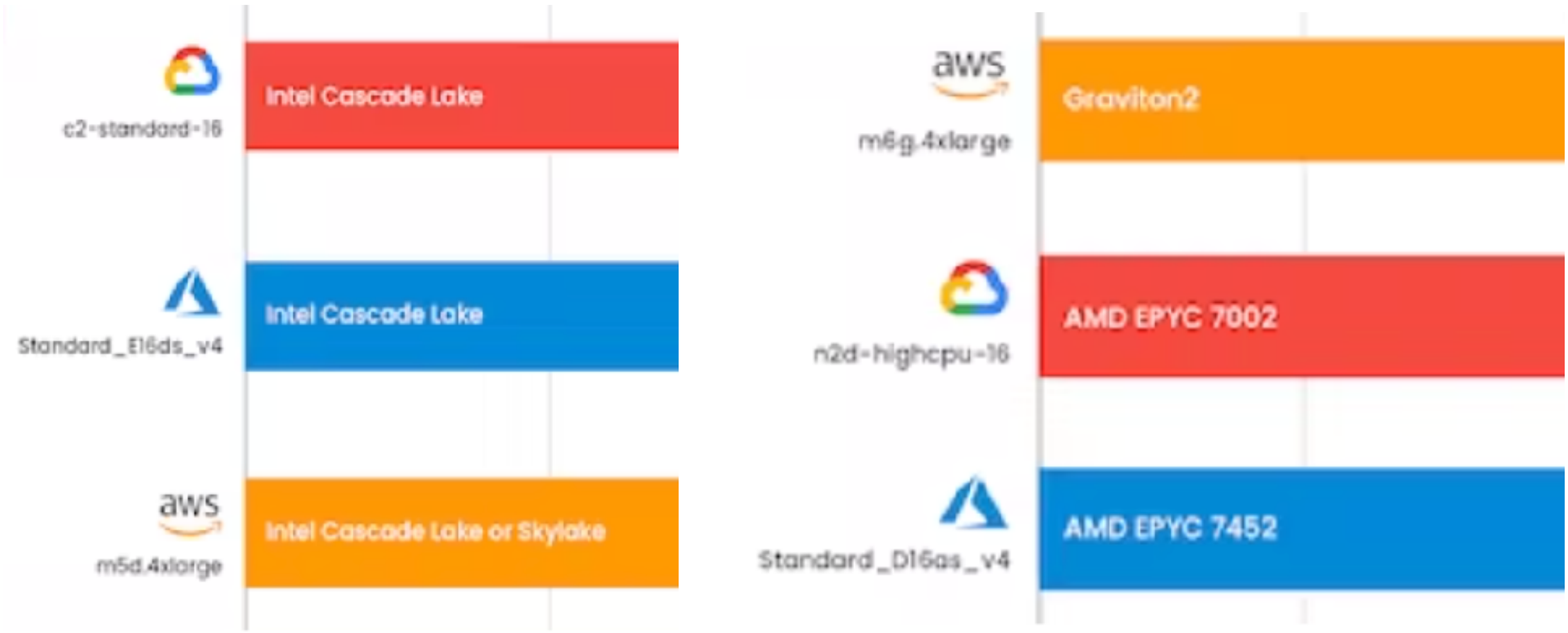
- Azure App Service
- AWS Elastic Beanstalk
- Google App Engine

2

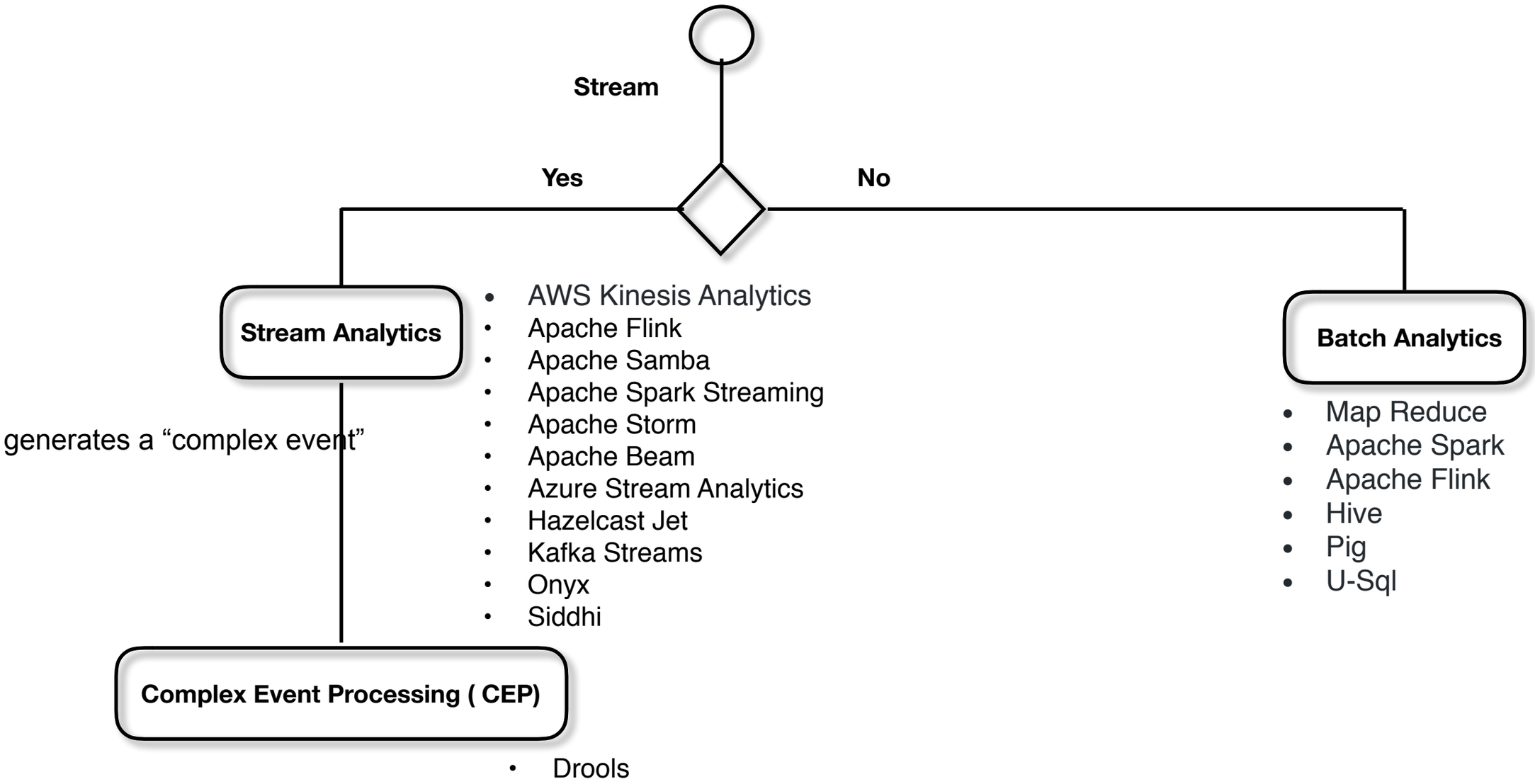
Managed Web Server (PAAS 1.0)

- Azure App Service
- AWS Elastic Beanstalk
- Google App Engine

2


c2-standard-16
Intel Cascade Lake
m6g.4xlarge
Graviton2
Standard_E16ds_v4
Intel Cascade Lake
n2d-highcpu-16
AMD EPYC 7002
m5d.4xlarge
Intel Cascade Lake or Skylake
Standard_D16as_v4
AMD EPYC 7452

Choose Analytical Compute



SERVING LAYER

PARTIAL AGGREGATES

QUERY

REAL-TIME VIEWS

VIEW 1

VIEW 2

VIEW 3

DATA

STREAM

RAW DATA

IMMUTABLE DATA

LOG

REAL-TIME PROCESSING

PROCESSED DATA

SPEED LAYER

RAW DATA

DATA STORAGE

BATCH PROCESSING

PROCESSING DATA

BATCH LAYER

QUERY

DATA

VIEW 1

VIEW 2

VIEW 3

BATCH VIEWS

MERGED VIEW

PARTIAL AGGREGATIONS

REAL-TIME VIEWS

VIEW 1

VIEW 2

VIEW 3

SERVING LAYER

STREAM

DATA INCREMENT

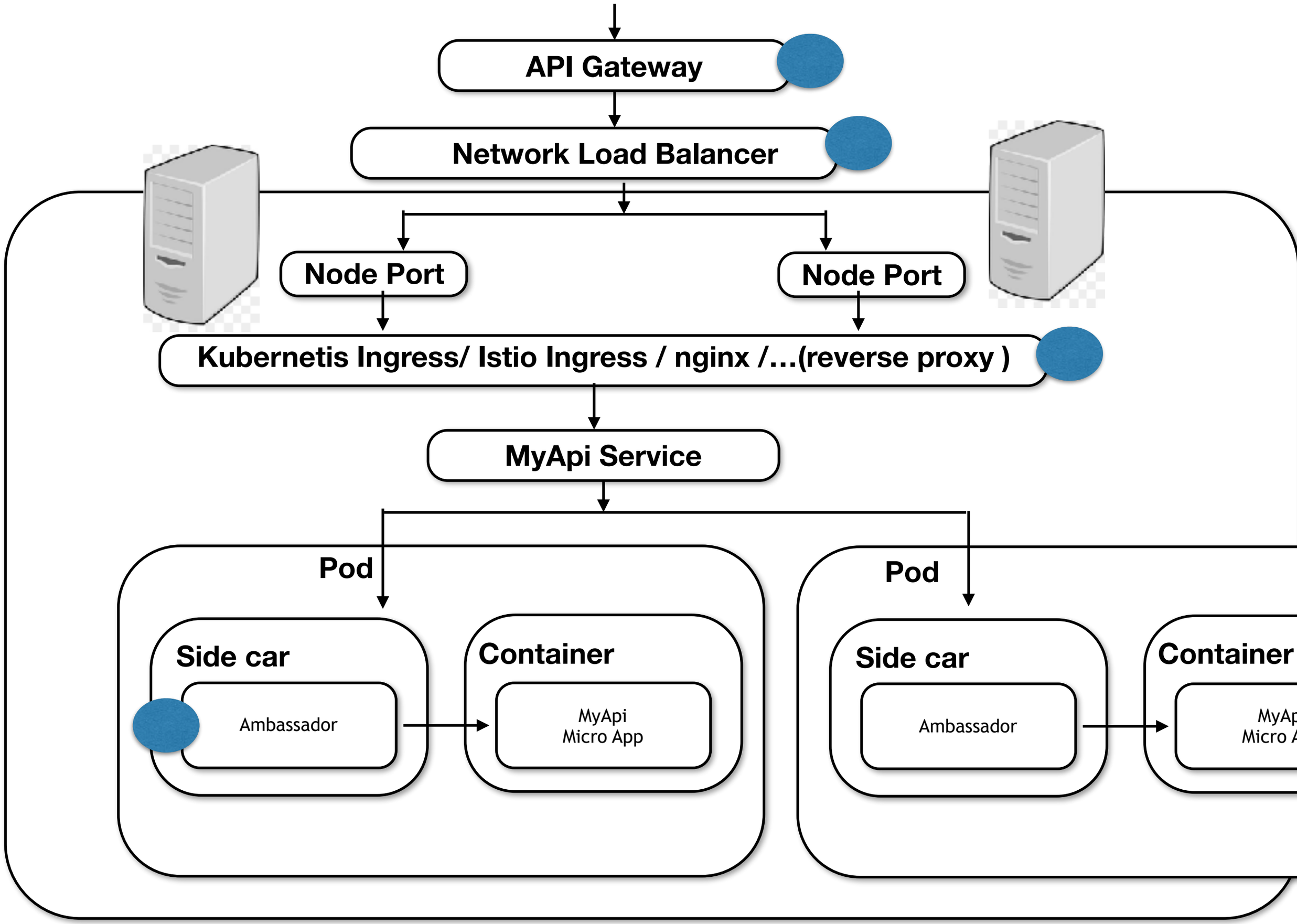
REAL-TIME PROCESSING

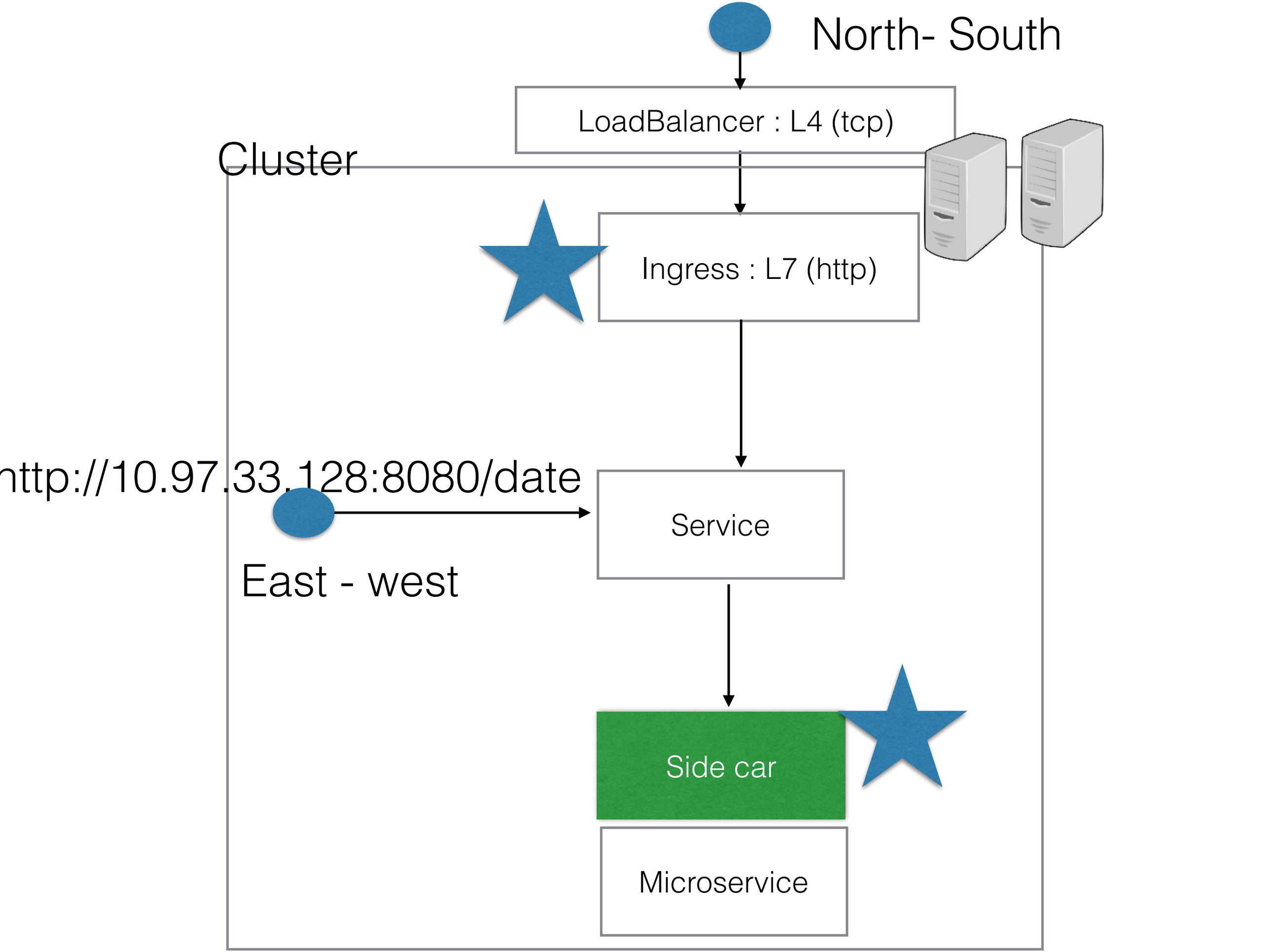
PROCESSED INCREMENT

SPEED LAYER

Centralize

Deployment Diagram




North- South
LoadBalancer : L4 (tcp)
Cluster
Ingress : L7 (http)
http://10.97.33.128:8080/date
Service
East - west
Side car
Microservice

START

Web application? (HTTP/HTTPS)

No

Internet facing application?

No

Azure Load Balancer

Yes

No

Global / Deployed in multiple regions?

Yes

Traffic Manager + Azure Load Balancer

Yes

Internet facing application?

No

Application Gateway

Yes

Global / Deployed in multiple regions?

Yes

Do you require SSL offload or application-layer processing per request?

Yes

Azure Front Door + Application Gateway

No

Hosting – PaaS, IaaS, AKS

PaaS

(App Service, Functions)

Azure Front Door

AKS

Azure Front Door + Application Gateway ingress controller

IaaS (VMs)

Azure Front Door + Azure Load Balancer

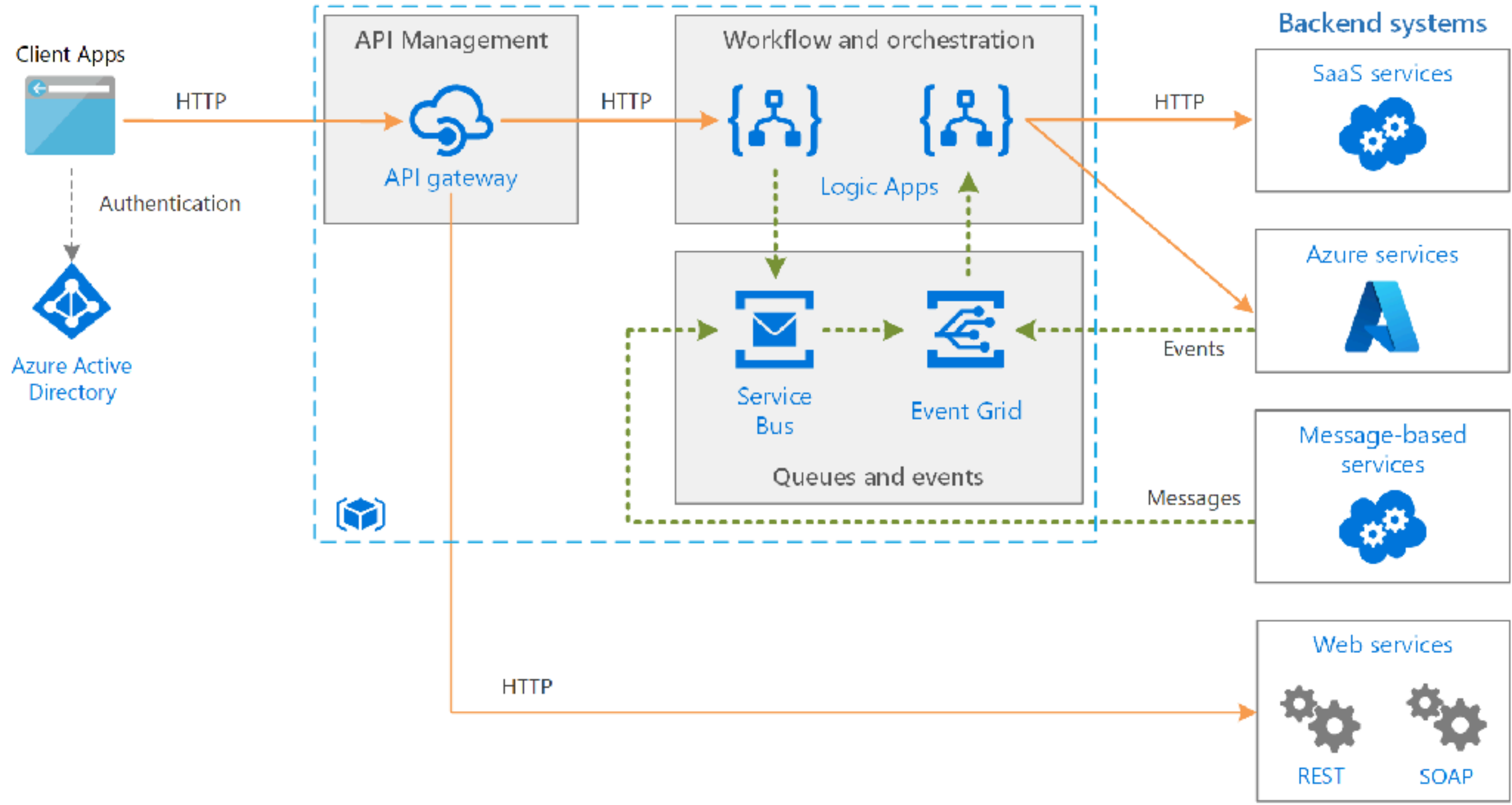
No

Yes

Do you require performance acceleration?

No

Application Gateway



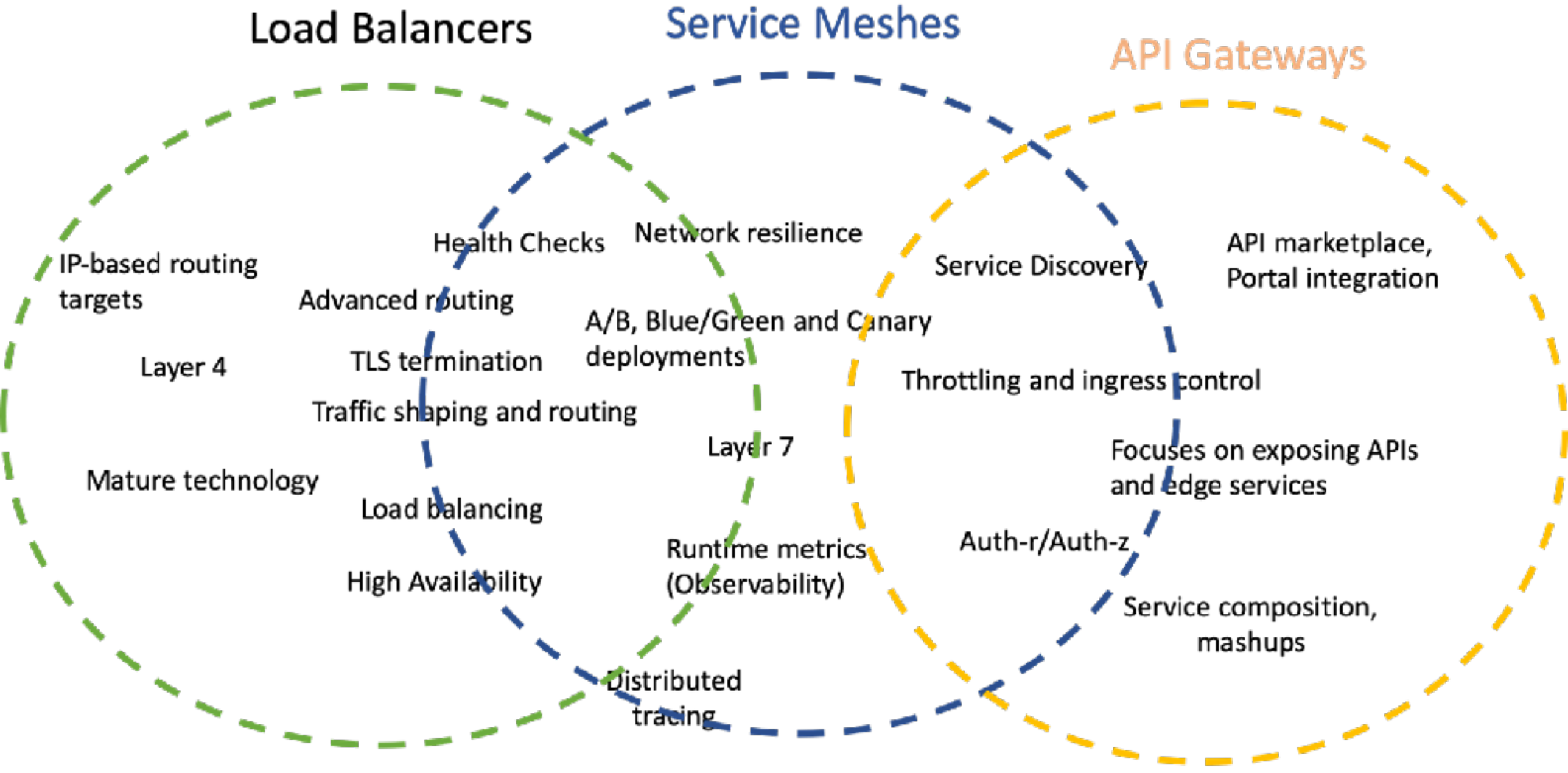
While API Gateways and API management can be used interchangeably, strictly speaking an API gateway refers to the individual proxy server, while API management refers to the overall solution of managing APIs in production which includes a set of API gateways acting in a cluster, an administrative UI, and may even include additional items such as a developer portal for customers to sign up and generate new API keys.

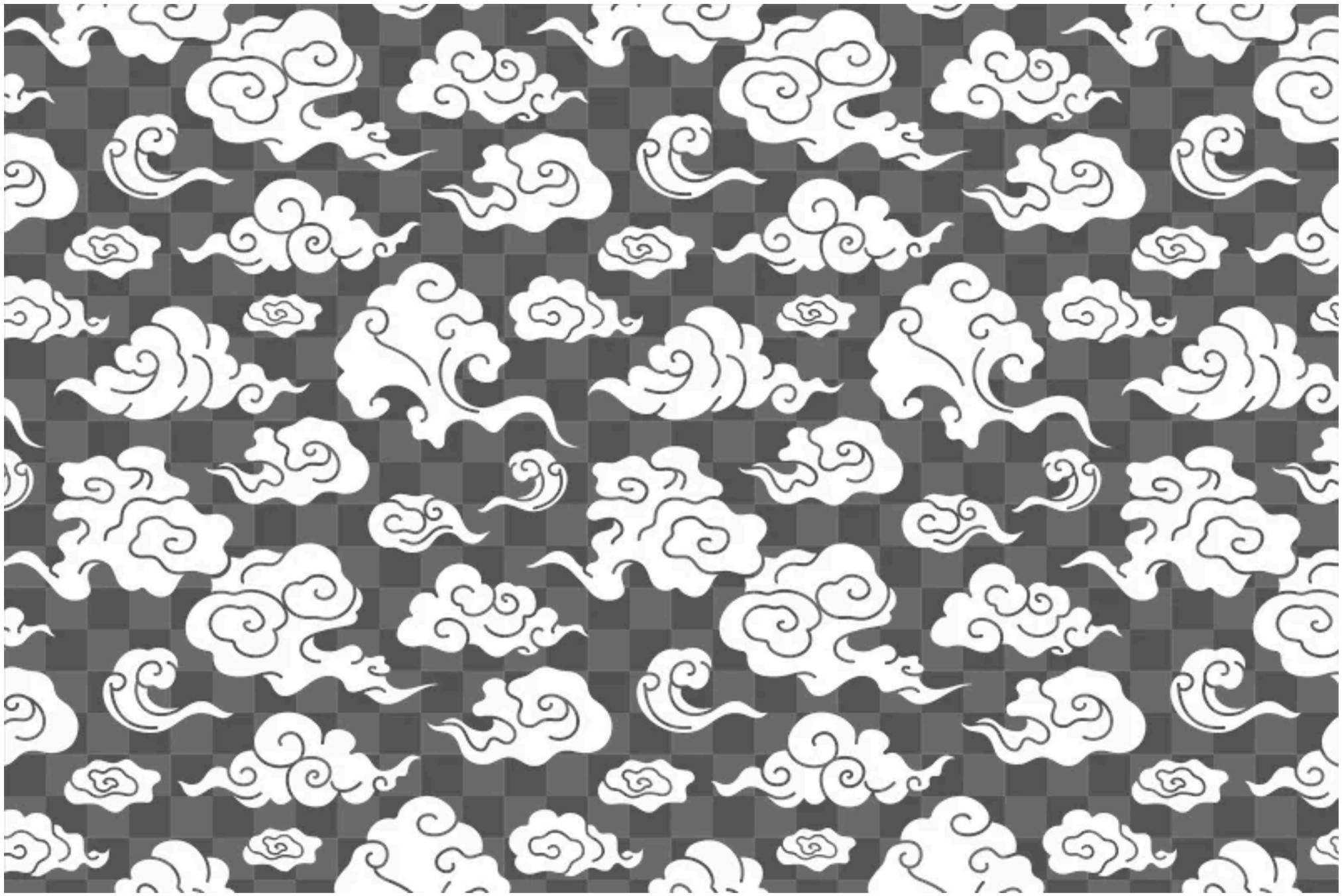
Service	Global/regional	Recommended traffic
Azure Front Door	Global	HTTP(S)
Traffic Manager	Global	non-HTTP(S)
Application Gateway	Regional	HTTP(S)
Azure Load Balancer	Regional	non-HTTP(S)

API Gateway: Abstraction, decoupling, edge routing / security

Service Mesh: endpoints, hosts, ports, metric collection, traffic routing, security

Deployment Platform: Cluster nodes, container schedule, resource management


Load Balancers
Service Meshes
API Gateways
IP-based routing targets
Layer 4
Mature technology
Load balancing
High Availability
Health Checks
Network resilience
Advanced routing
TLS termination
Traffic shaping and routing
A/B, Blue/Green and Canary deployments
Layer 7
Runtime metrics (Observability)
Distributed tracing
Service Discovery
Throttling and ingress control
Auth-r/Auth-z
API marketplace, Portal integration
Focuses on exposing APIs and edge services
Service composition, mashups



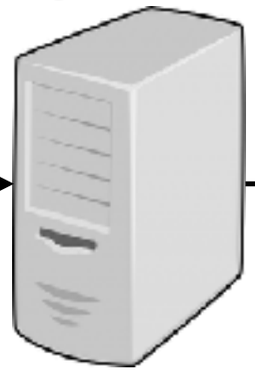
Architecture patterns

Tenant

Lob Compute
{ k8s }

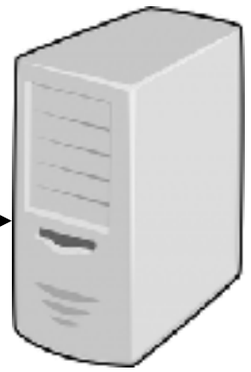
Analytical Compute
{ Databricks }

Images -> redhat



Device

Gateway



Azure/aws cloud

Patterns

Scalability Patterns

1. Competing Consumer *
2. Queue based load leveling *
3. Sharding *
4. Claim Check *
5. CQRS *
6. Deployment stamps *
7. Geode *
8. Split *
9. Clone *

Performance Patterns

1. Async Request Reply *
2. Static Content Hosting *
3. Materialized View *
4. Backends for front ends *
5. Cache Aside *
6. Index Table *
7. Throttling *
8. Rate Limiting *

Centralize Cross Cutting Concerns

1. Gateway Aggregation *
2. Gateway Routing *
3. Gateway Offloading *
4. Gatekeeper *
5. Side Car *
6. Ambassador *
7. Federated Identity *
8. External Config store *

Legacy modernisation Patterns

1. Strangler *
2. Anti Corruption Layer *

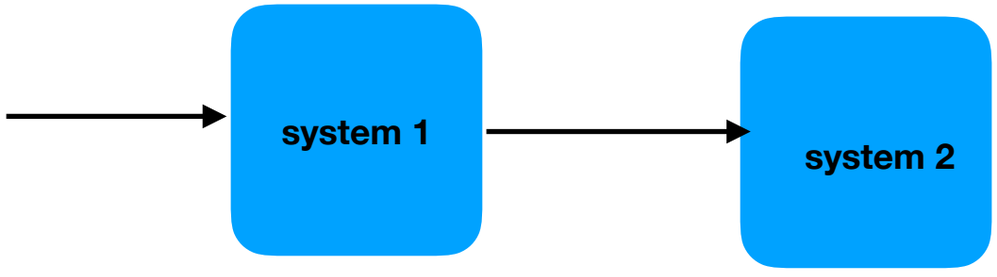
Reliability Availability Patterns

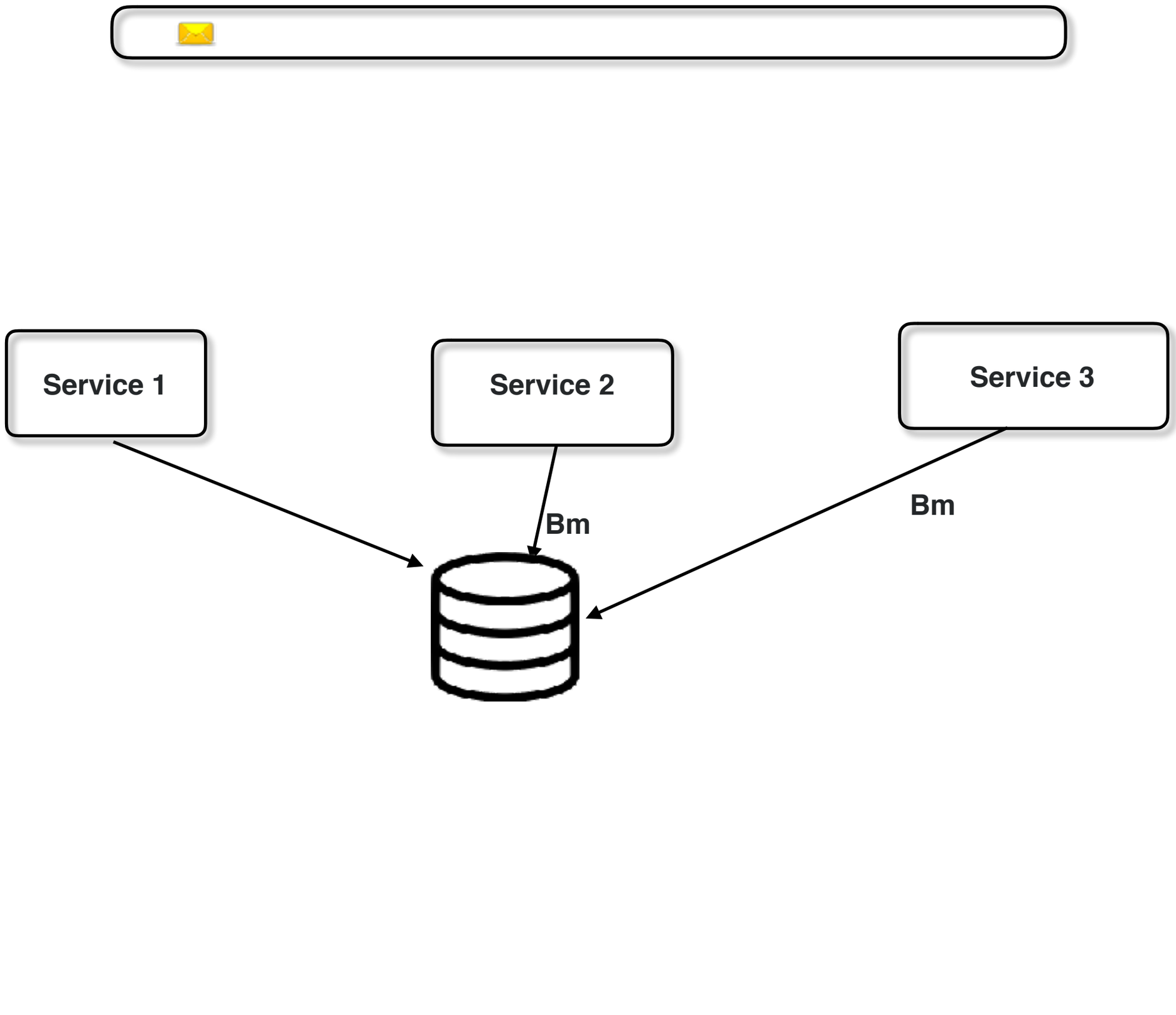
1. Retry *
2. Circuit breaker *
3. Bulk head *
4. Compensatable transaction *
5. SAGA *
6. Schedular Agent supervisor *
7. Health endpoint *

Communication Patterns

1. Pub sub *
2. Priority Queue *
3. Event Sourcing *
4. Choreography *
5. Leader Election (last weapon)
6. Sequential convoy *

1. Retry *
2. Circuit breaker *
3. Bulk head *
4. Clone
5. Deployment stamps
6. Geode
7. Split
8. Throttling
9. Rate limiting




Service 1
Service 2
Service 3
Bm
Bm

Client

Client
Metadata

Watcher
<<boundary>>

File System

Metadata Sync
<<boundary>>

Chunk Sync
<<boundary>>

Controller

Chunker

Compression

Cloud

Notification
Service

Chunk Api
Service

Https

Metadata Api
Service

enable encryption

"Block Storage”

Cloud chunk
Storage

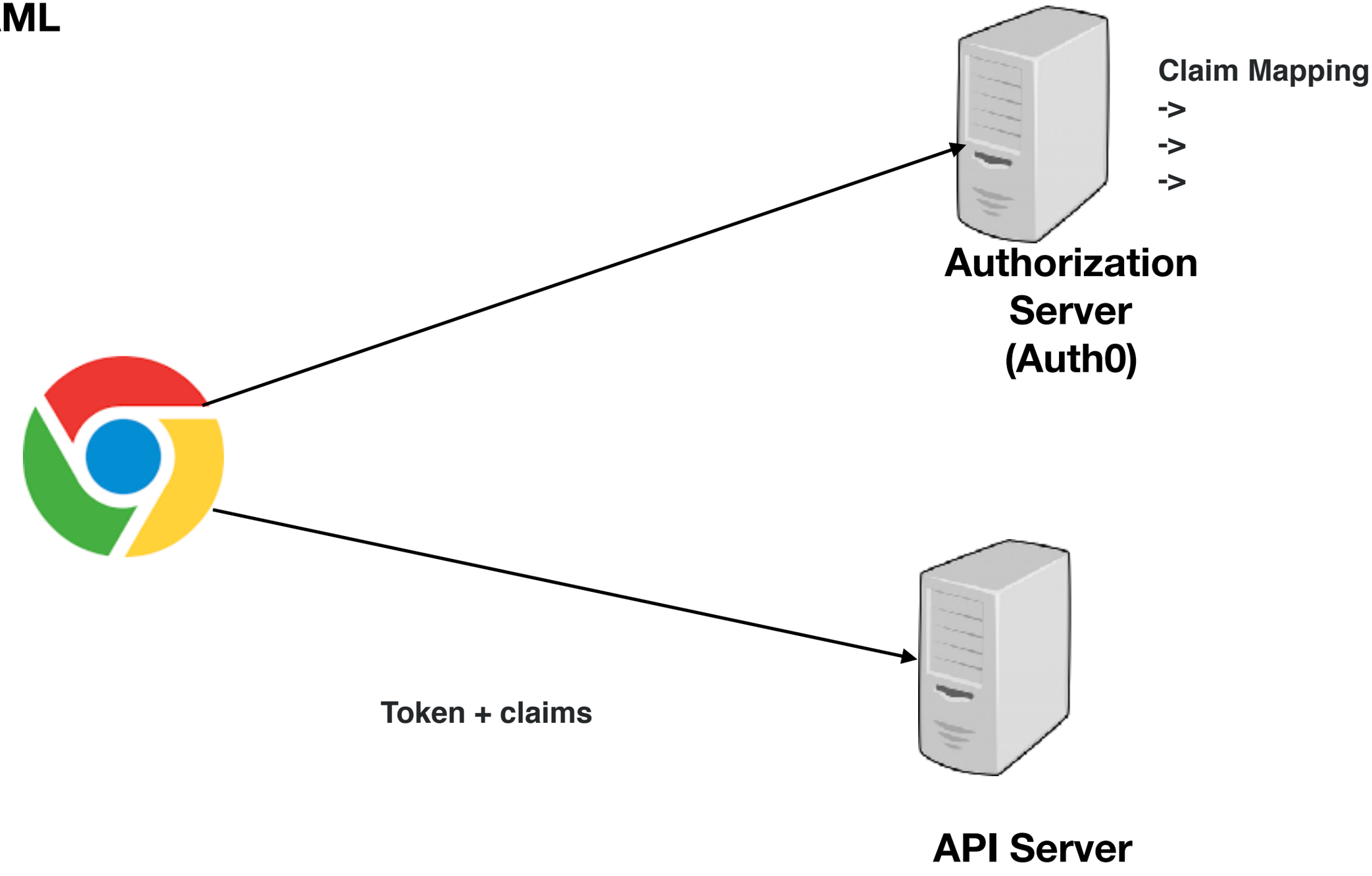
“Document Store/
Rdbms”

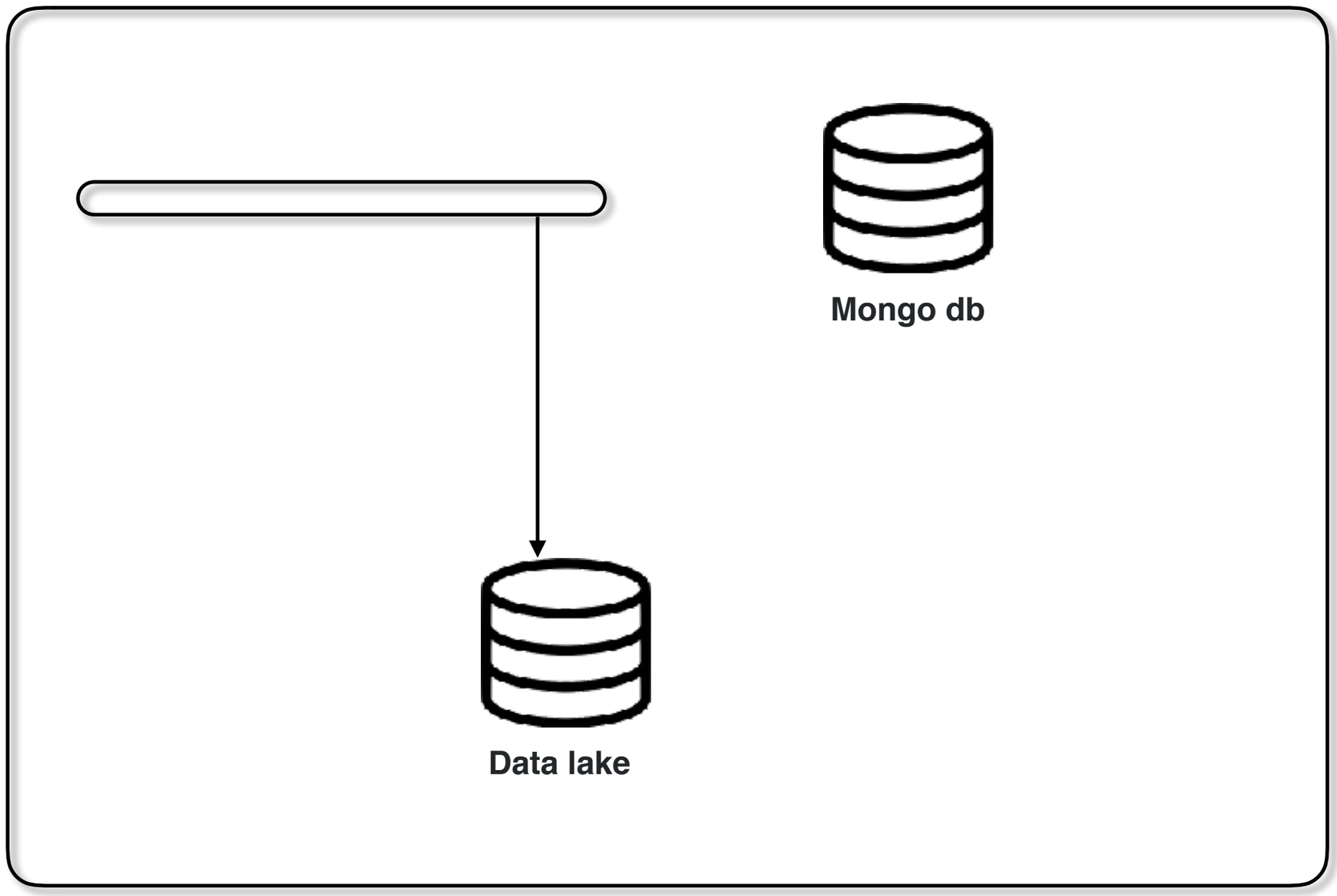
Cloud
Metadata

3 A

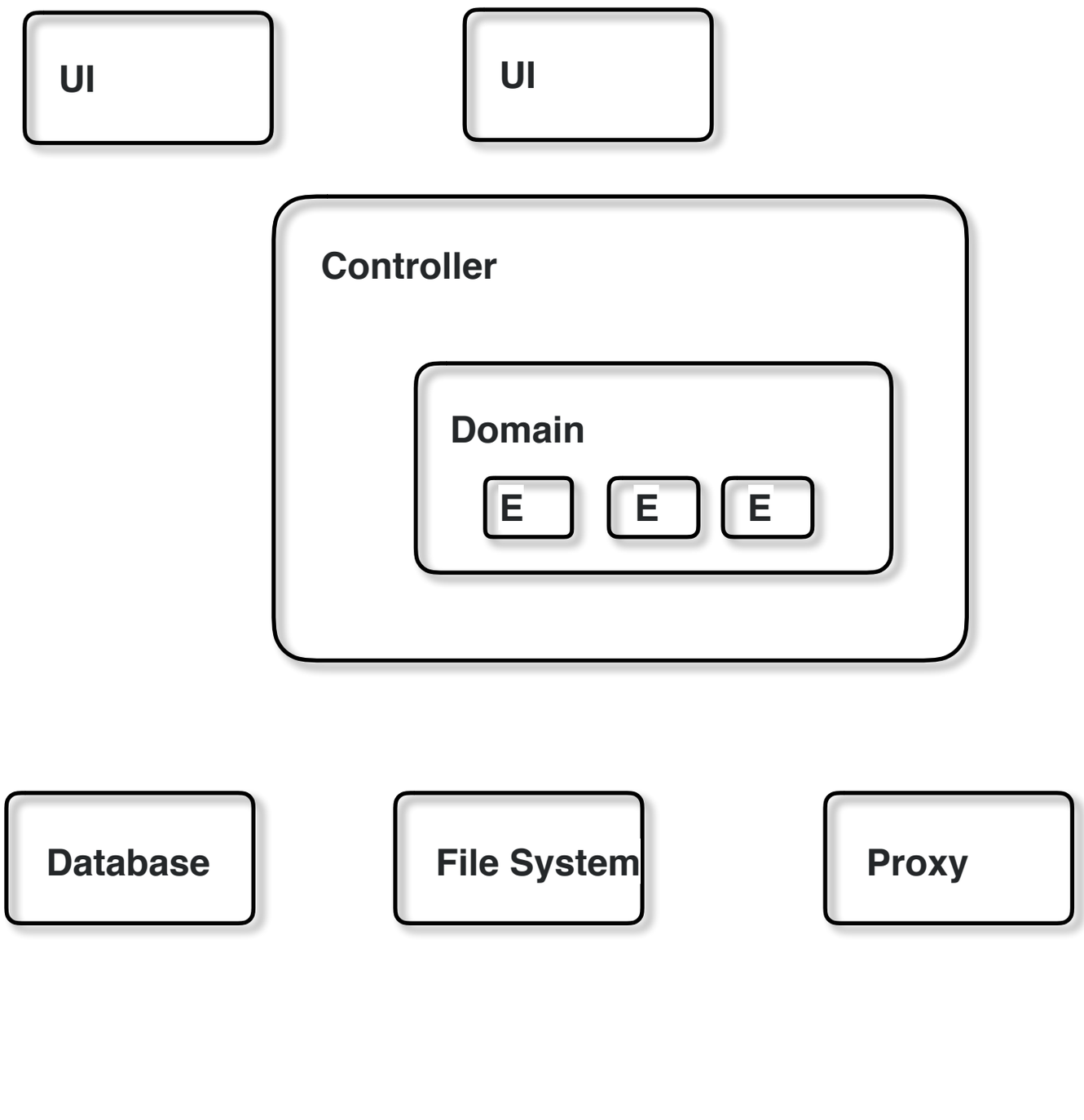
- Authentication -> OAth2/ SAML
- Authorization -> Claim
- Auditing -> Event Sourcing
- Input Validation -> WAF , App
- Exception Handling - Reverse Proxy, Application
- Asset Handling in Transit - Https /wss
- Asset Handling in Rest - Cloud storage Security
- Key Management - Key Vault
- Throttling/ Rate Limiting - Reverse Proxy/ Service Mesh

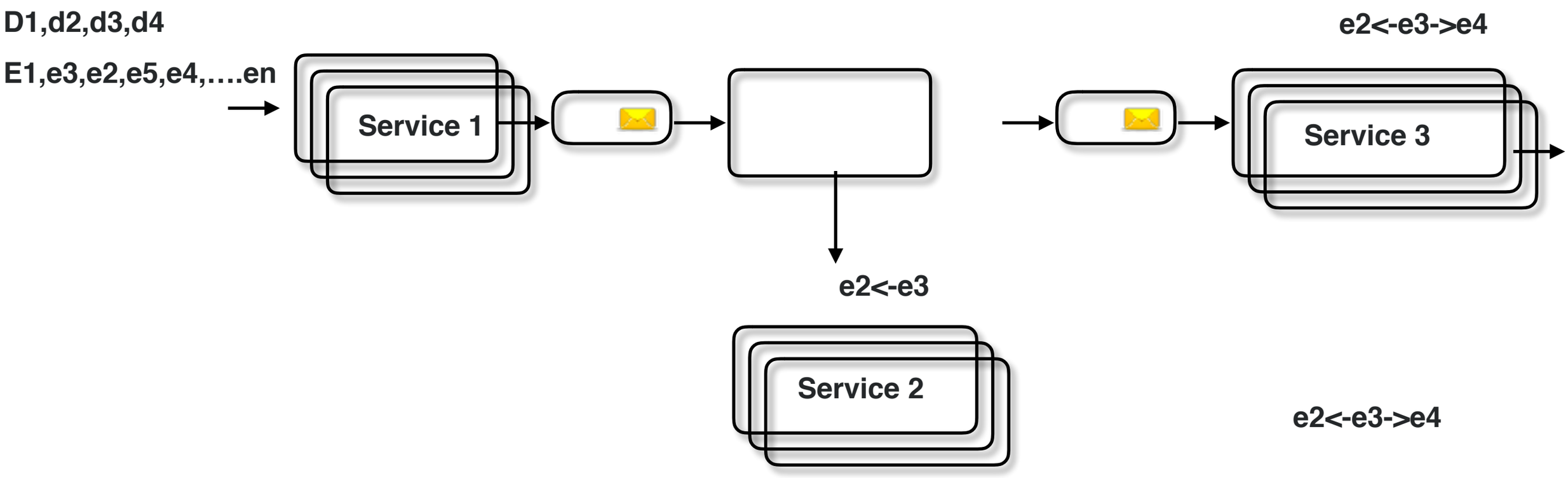
OAuth2 / SAML


Claim Mapping
-> -> ->
Authorization Server (Auth0)
Token + claims
API Server

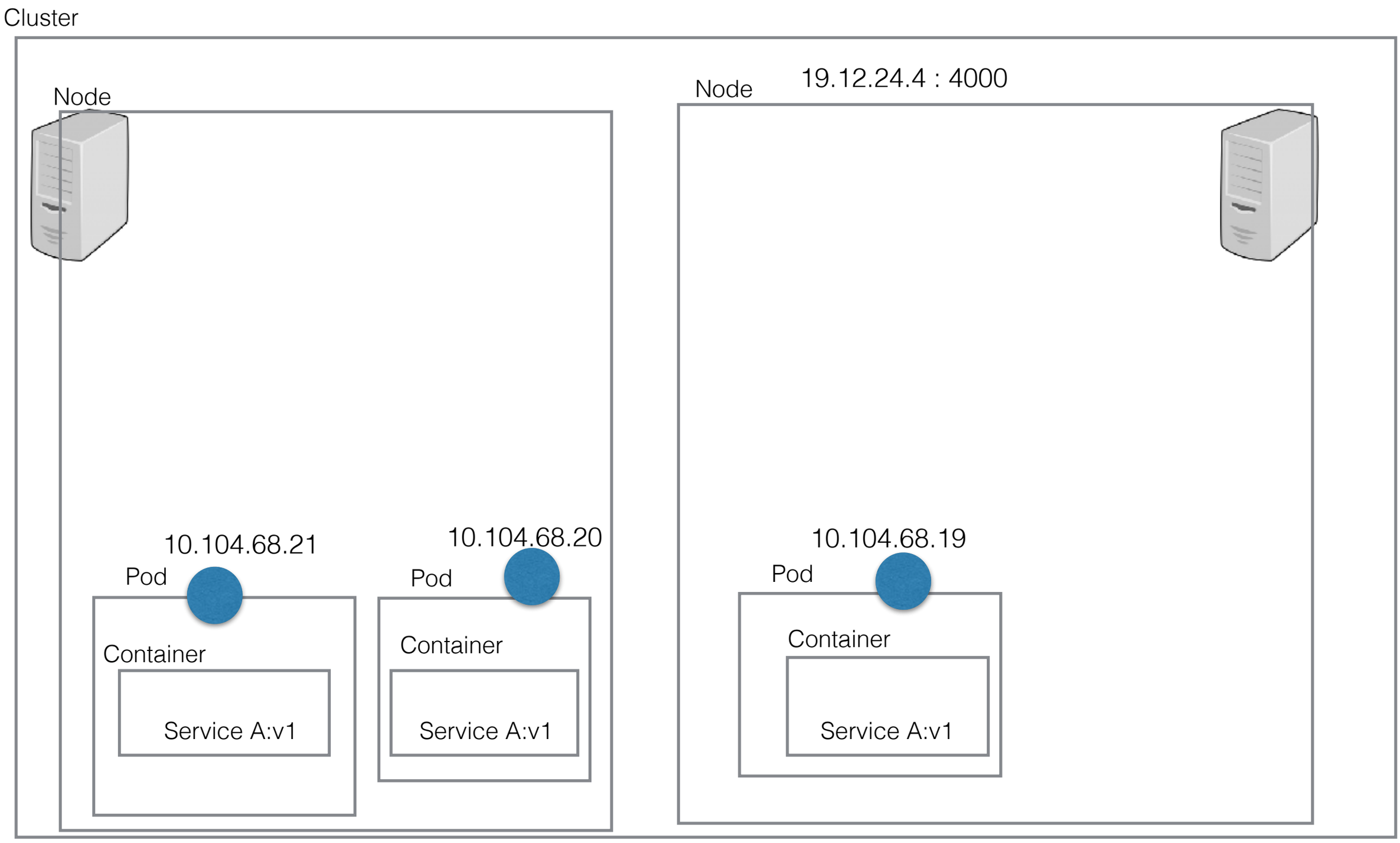

Mongo db
Data lake

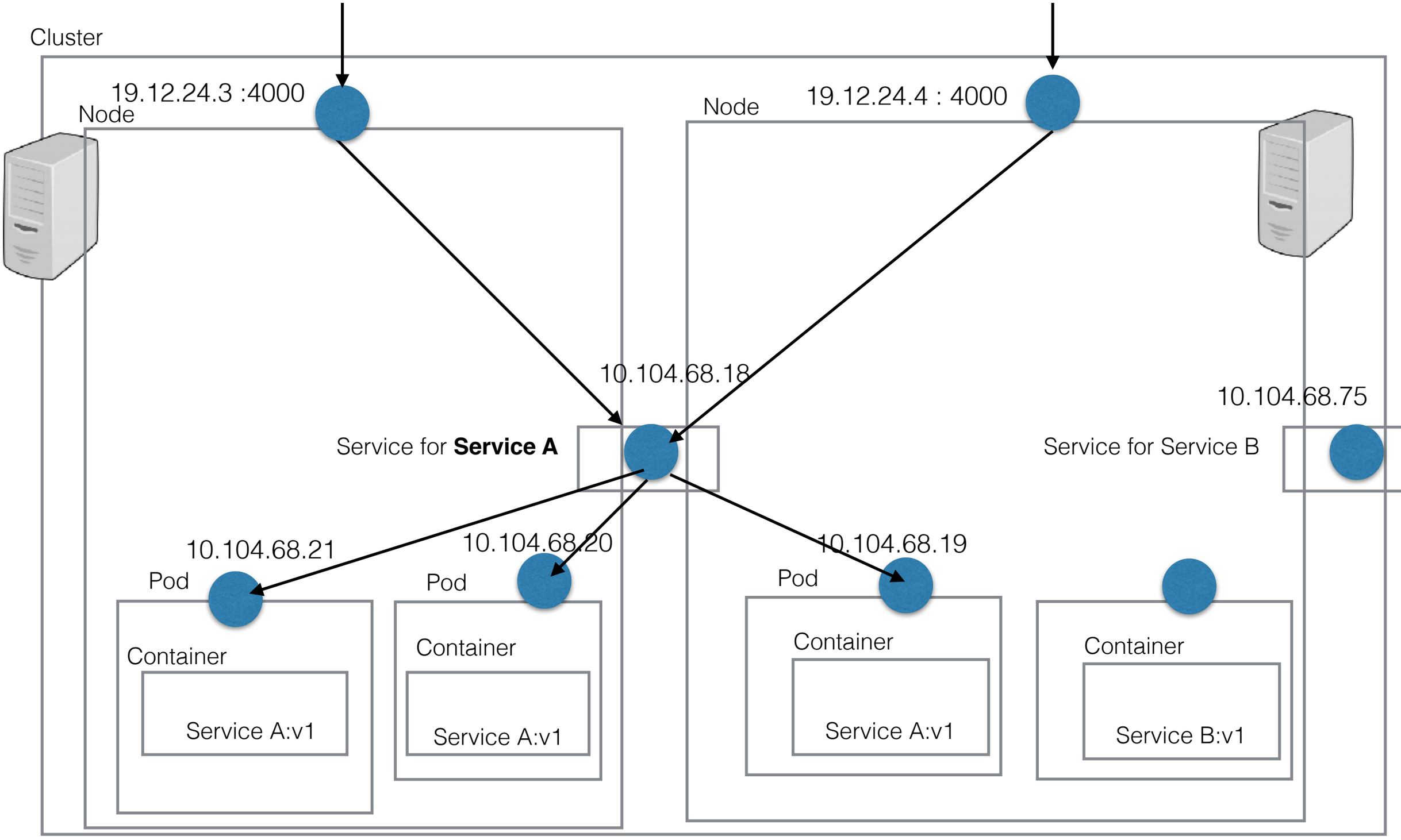
Boundary- Control - Entity

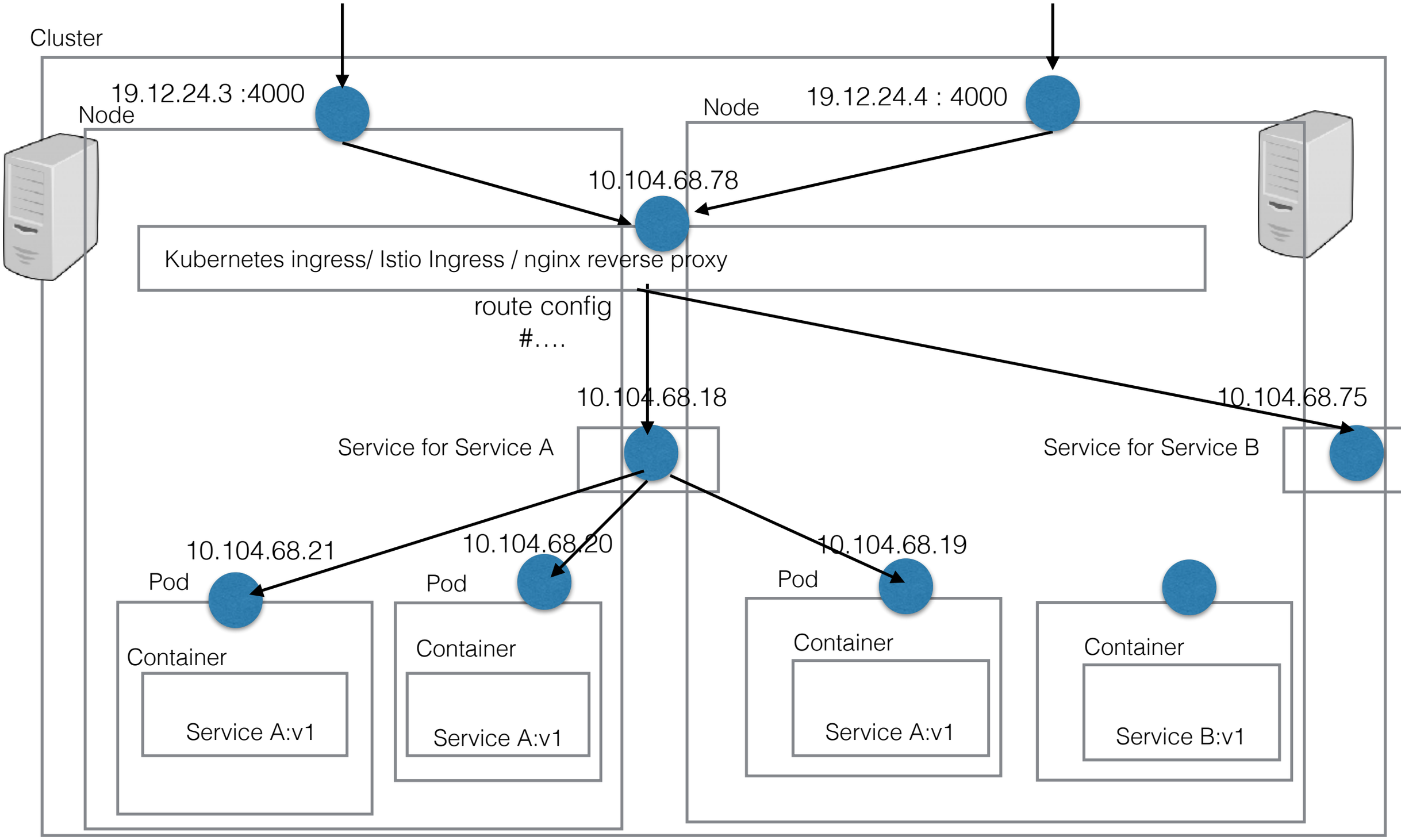


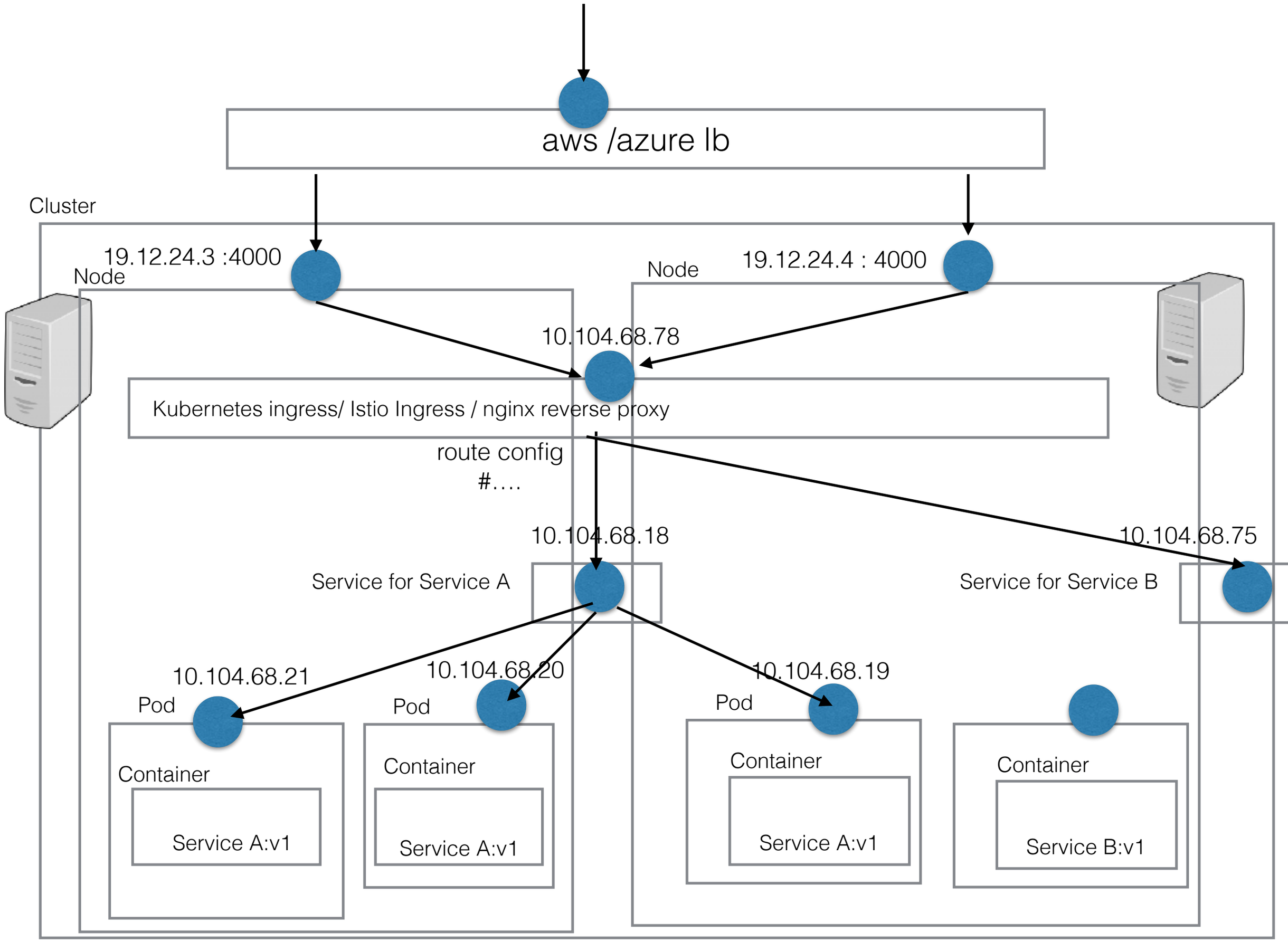

D1,d2,d3,d4
E1,e3,e2,e5,e4,….en
Service 1
e2<-e3
Service 2
e2<-e3->e4
Service 3
e2<-e3->e4

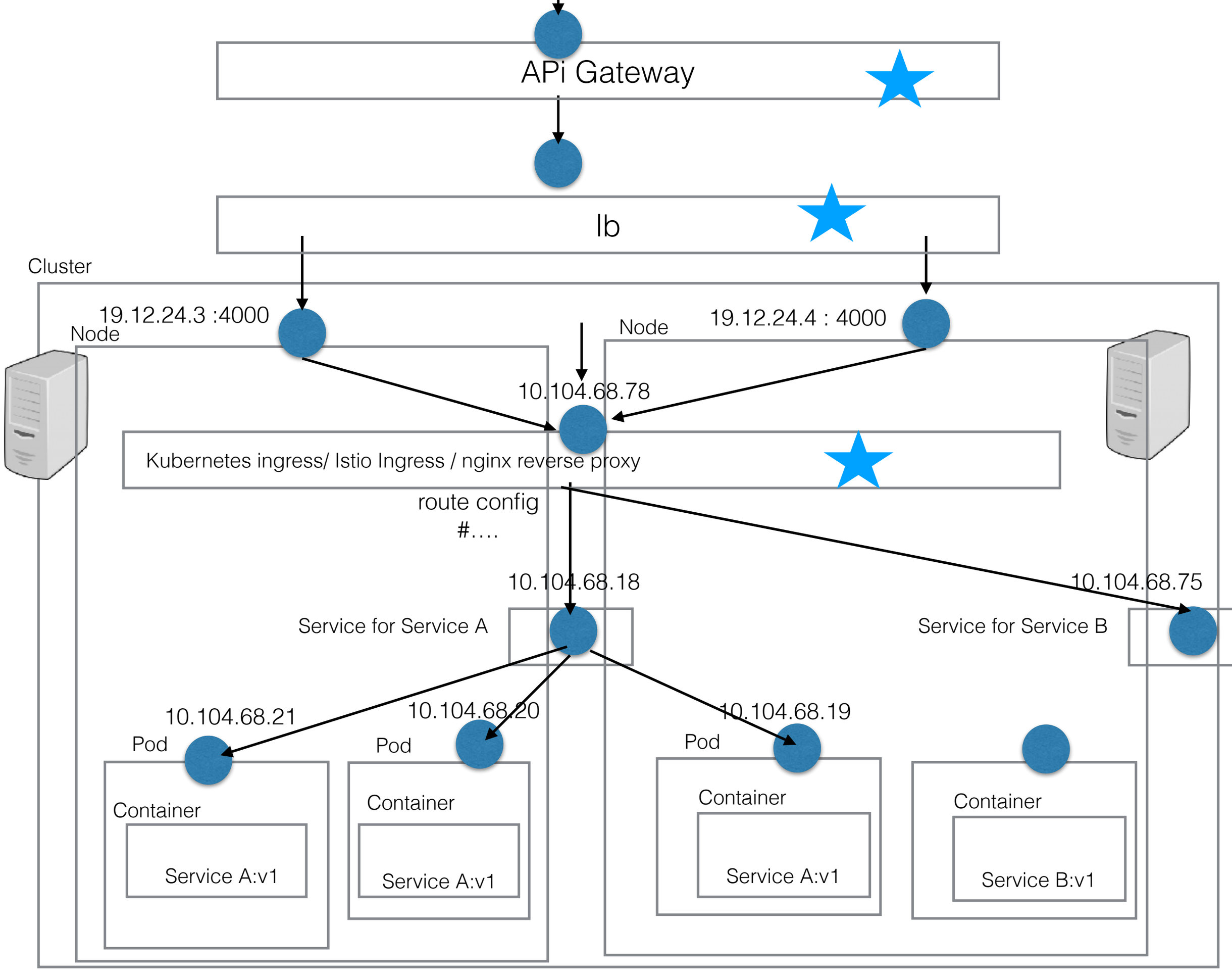
Service A : 3 clones




Cluster
19.12.24.3 :4000
Node
19.12.24.4 : 4000
Node
10.104.68.18
Service for **Service A**
10.104.68.75
Service for Service B
10.104.68.21
Pod
Container
Service A:v1
10.104.68.20
Pod
Container
Service A:v1
10.104.68.19
Pod
Container
Service A:v1
Container
Service B:v1


Cluster
19.12.24.3 :4000
Node
19.12.24.4 : 4000
Node
10.104.68.78
Kubernetes ingress/ Istio Ingress / nginx reverse proxy
route config
#....
10.104.68.18
Service for Service A
10.104.68.75
Service for Service B
10.104.68.21
Pod
Container
Service A:v1
10.104.68.20
Pod
Container
Service A:v1
10.104.68.19
Pod
Container
Service A:v1
Container
Service B:v1


aws /azure lb
Cluster
19.12.24.3 :4000
Node
19.12.24.4 : 4000
Node
10.104.68.78
Kubernetes ingress/ Istio Ingress / nginx reverse proxy
route config
#....
10.104.68.18
Service for Service A
10.104.68.75
Service for Service B
10.104.68.21
Pod
Container
Service A:v1
10.104.68.20
Pod
Container
Service A:v1
10.104.68.19
Pod
Container
Service A:v1
Container
Service B:v1


APi Gateway
lb
Cluster
19.12.24.3 :4000
Node
19.12.24.4 : 4000
Node
10.104.68.78
Kubernetes ingress/ Istio Ingress / nginx reverse proxy
route config
#....
10.104.68.18
Service for Service A
10.104.68.75
Service for Service B
10.104.68.21
Pod
Container
Service A:v1
10.104.68.20
Pod
Container
Service A:v1
10.104.68.19
Pod
Container
Service A:v1
Container
Service B:v1

North-South

APi Gateway - optional

172.17.0.98

North-South

aws lb

Cluster

19.12.24.3 :4000

Node

19.12.24.4 : 4000

Node

10.104.68.78

Kubernetes ingress/ Istio Ingress / nginx reverse proxy

East-West

route config
#....

10.104.68.18
->10.104.68.21
->10.104.68.20
->10.104.68.19

10.104.68.18

Service for Service A

10.104.68.75

Service for Service B

10.104.68.21

Pod

Side car

Service A:v1

Container

10.104.68.20

Pod

Side car

Service A:v1

Container

10.104.68.19

Pod

Side car

Service A:v1

Container

Side car

Container
Service B:v1

East-West

North-South

API Gateway (L7)

WAF

North-South

aws lb (L4)

Cluster

19.12.24.3 :4000

Node

Node

19.12.24.4 : 4000

10.104.68.78

Kubernetes ingress/ Istio Ingress / nginx reverse proxy

East-West

route config

#....

10.104.68.18

->10.104.68.21

->10.104.68.20

->10.104.68.19

10.104.68.18

Service for Service A

10.104.68.75

Service for Service B

10.104.68.21

Pod

10.104.68.20

Pod

10.104.68.19

Pod

Side car

Service A:v1

Container

Side car

Service A:v1

Container

Side car

Service A:v1

Container

Side car

Container

Service B:v1

East-West

API Management

...

API Gateway

...

LB

North-South

Cluster

19.12.24.3 :4000

Node

19.12.24.4 : 4000

Node

10.104.68.78

Kubernetes ingress/ Istio Ingress / nginx reverse proxy

East-West

route config

#....

10.104.68.18

->10.104.68.21

->10.104.68.20

->10.104.68.19

10.104.68.18

Service for Service A

10.104.68.75

Service for Service B

10.104.68.21

Pod

Side car

Service A:v1

Container

10.104.68.20

Pod

Side car

Service A:v1

Container

East-West

10.104.68.19

Pod

Side car

Service A:v1

Container

Side car

Container

Service B:v1

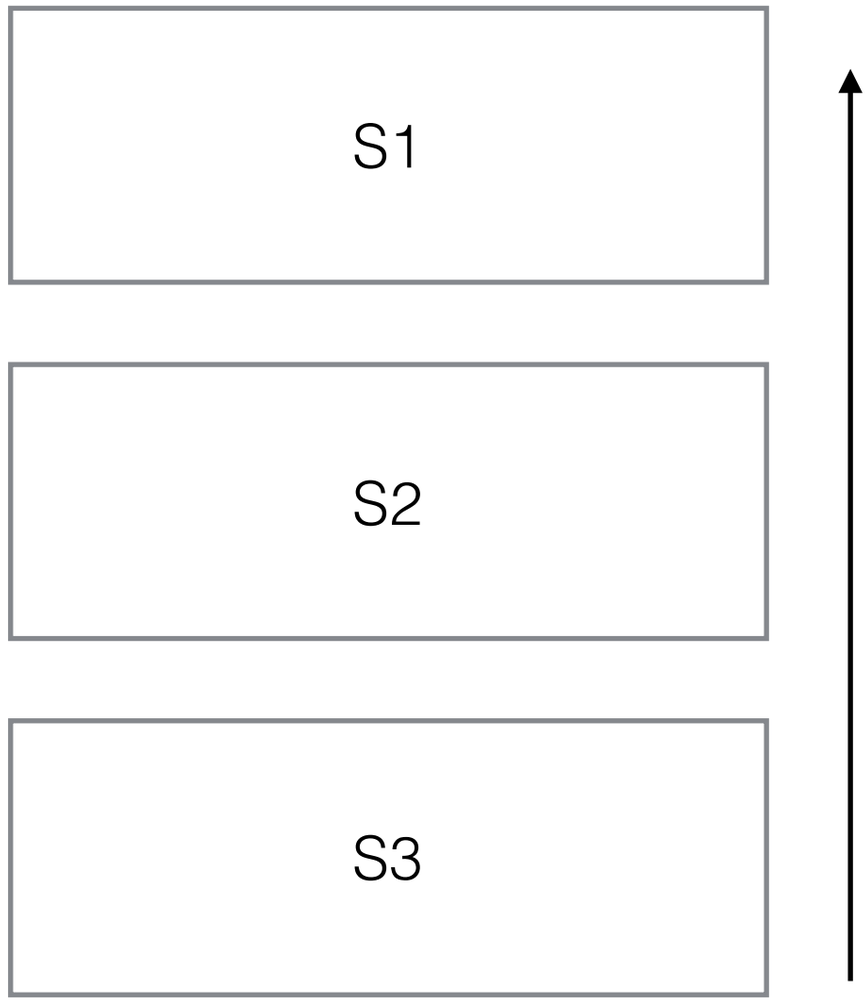
API Management

API Gateway

LB

Reverse proxy

Service Mesh


S1
S2
S3

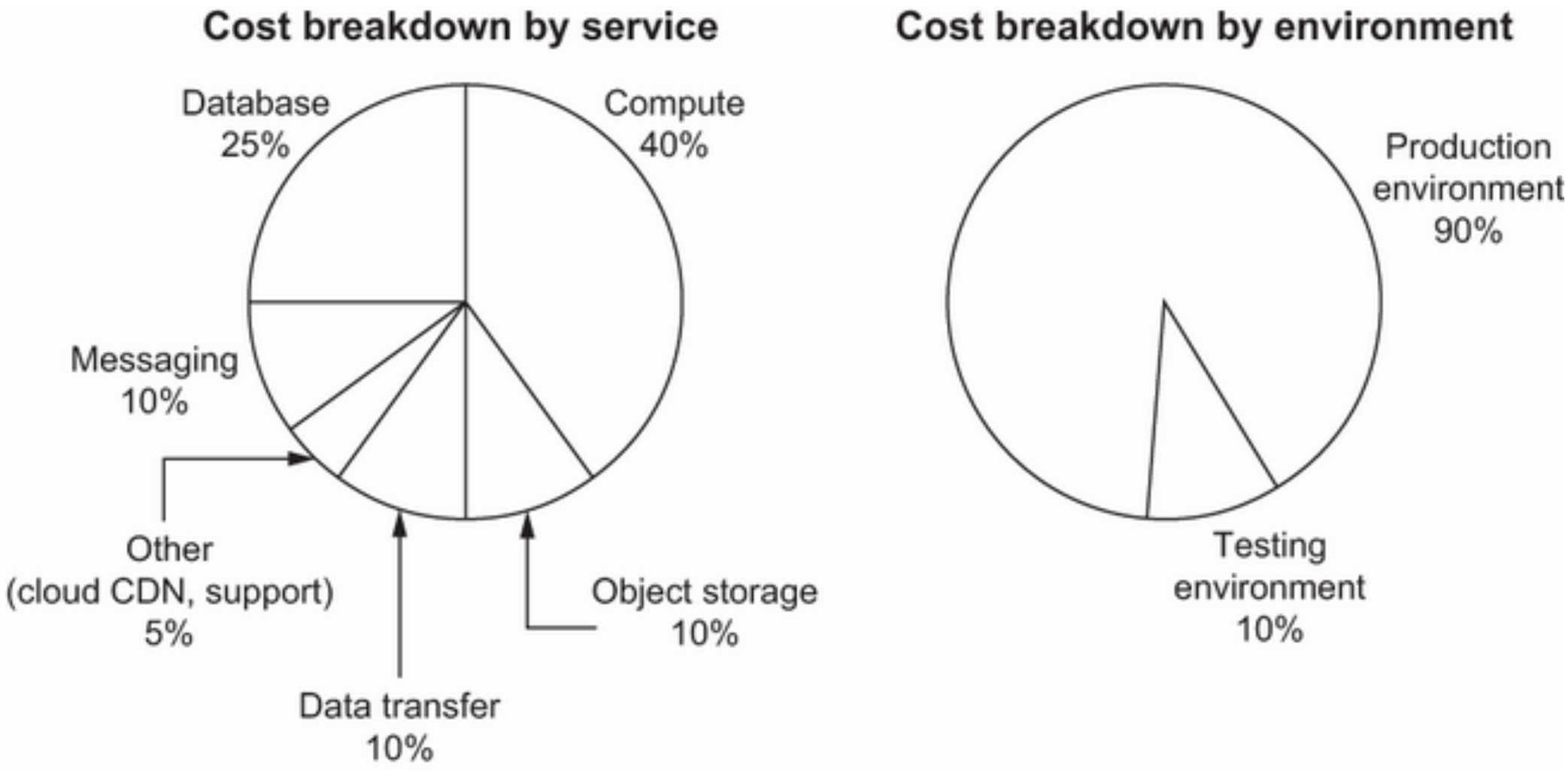
- Aggregation
- Authentication
- TLS offloading
- Separate Zone
-

Cost

- Compute Resource consolidation
- Savings Plans & Reserved Instances
- Delete Unused disk Snapshots
- Pause Idle DWH
- Choose the Right Storage Type
- Choose the Right Compute type (Spot,…)
- Rightsize Your Storage Resources Proactively
- Monitor & Correct Cost Anomalies
- Don't use serverless for constant, heavy loads. Serverless can be up to 3 times more expansive to operate than a dedicated server for such workloads.
- Know when to leave the cloud. At large consistent scale, on-premises hosting may be preferable to cloud hosting
- Set retention policy for storage
- Analyze Data Flow (in bound, out bound, firewall)
- Analyze Log levels

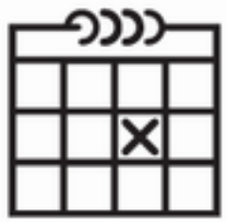
Tag resources

Offering	Subtotal, testing environment	Subtotal, production environment	Subtotal
Compute (servers)	$400	$3,600	$4,000
Database (Cloud SQL)	$250	$2,250	$2,500
Messaging (Pub/Sub)	$100	$900	$1,000
Object storage (Cloud Storage)	$100	$900	$1,000
Data transfer (networking egress)	$100	$900	$1,000
Other (Cloud CDN, Support)	$50	$450	$500
Total	$1,000	$9,000	$10,000


Cost breakdown by service
Database
25%
Compute
40%
Messaging
10%
Other
(cloud CDN, support)
5%
Data transfer
10%
Object storage
10%
Cost breakdown by environment
Production
environment
90%
Testing
environment
10%

Reducing Cloud Waste

Stop untagged or unused resources.

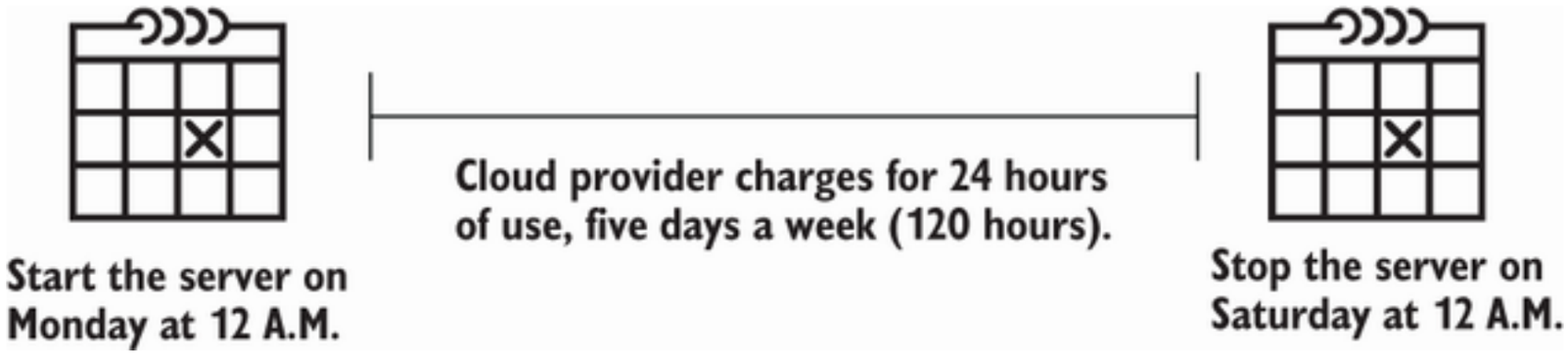


Start and stop resources on a schedule.

Choose the right resource type, size, and reservation.

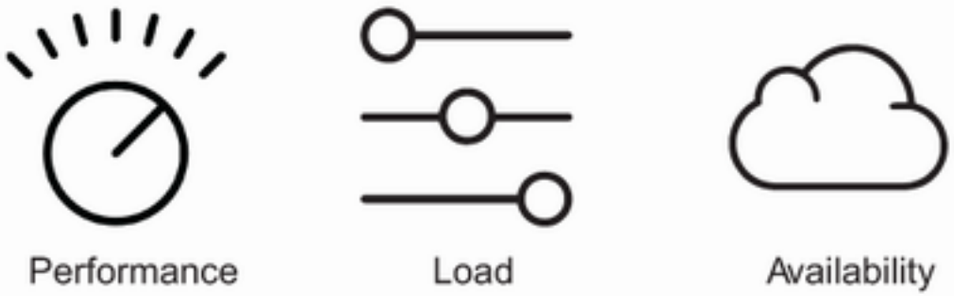
Enable autoscaling for resources.

Set a tag for resource expiration.

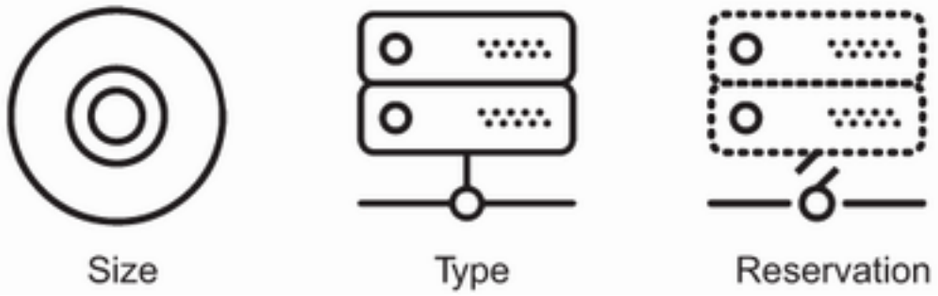


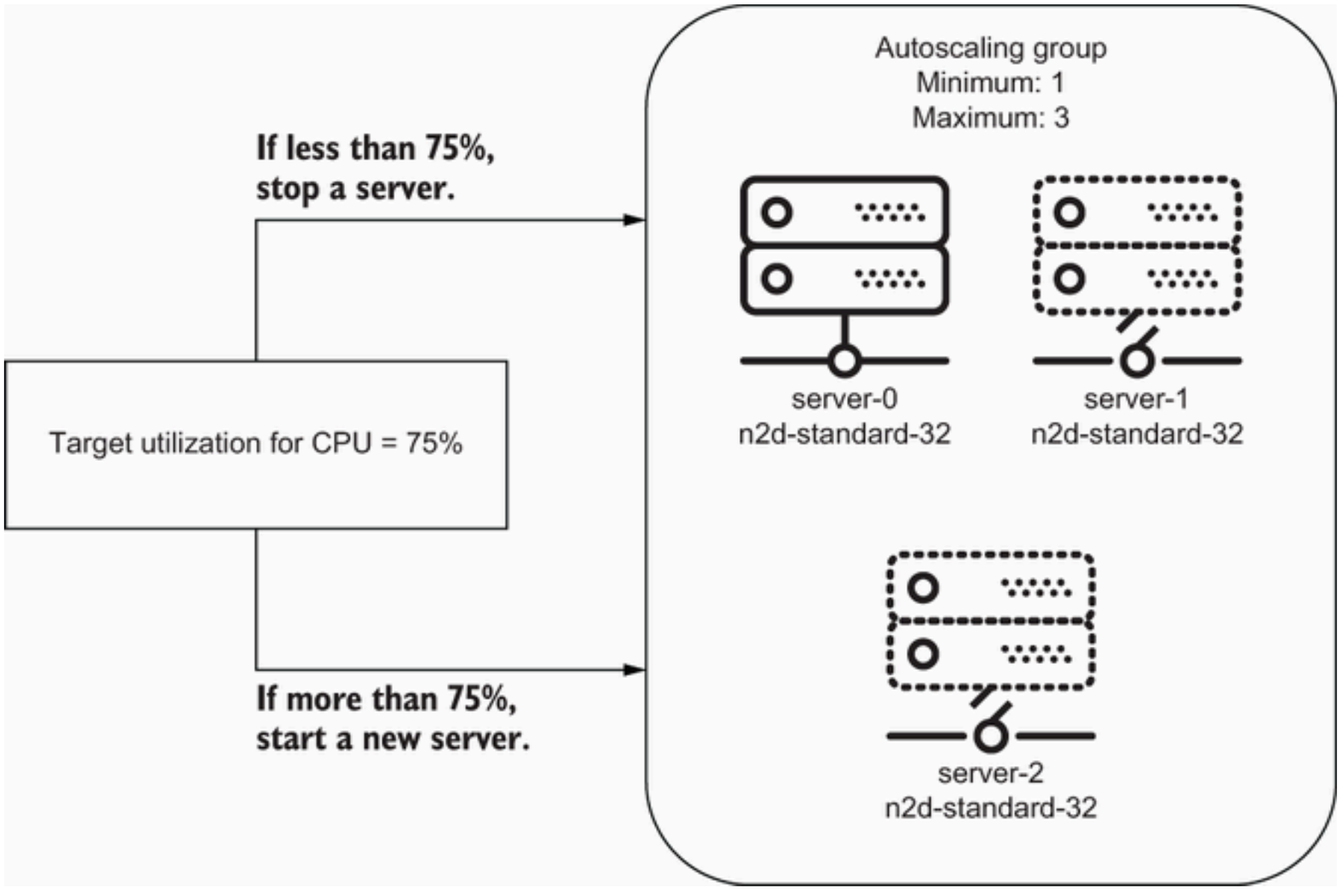
You can reduce cost by scheduling a resource to start and stop when not in use.

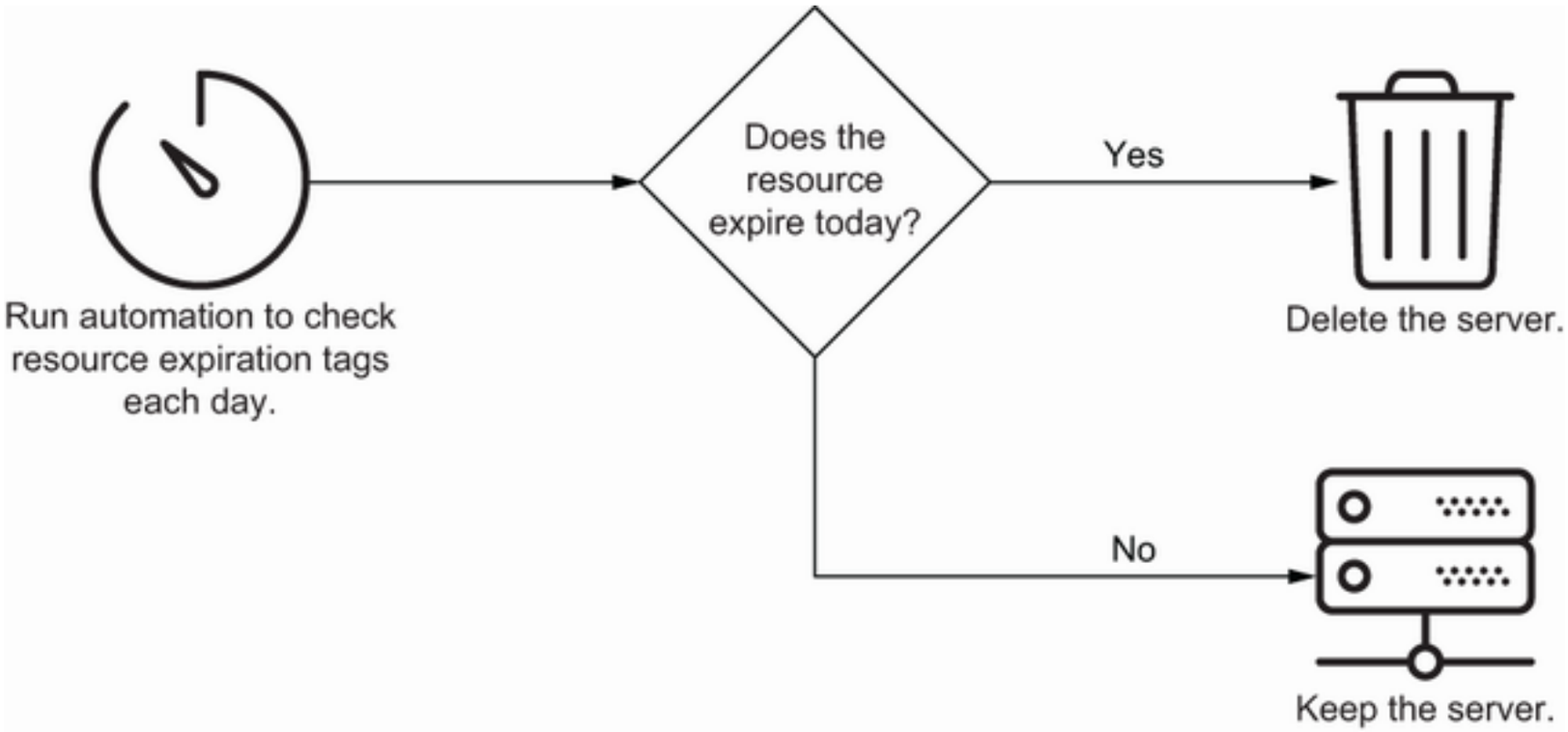
If the following does not affect your application and its ability to fulfill requests:



You can reduce the cost of infrastructure by changing a resource's…




Autoscaling group
Minimum: 1
Maximum: 3
If less than 75%,
stop a server.
Target utilization for CPU = 75%
If more than 75%,
start a new server.
server-0
n2d-standard-32
server-1
n2d-standard-32
server-2
n2d-standard-32


Does the resource expire today?
Yes
Run automation to check resource expiration tags each day.
Delete the server.
No
Keep the server.

Patterns

Scalability Patterns

1. Competing Consumer *
2. Queue based load leveling *
3. Sharding *
4. Claim Check *
5. CQRS *
6. Deployment stamps * -**welldoc**
7. Geode *
8. Split *
9. Clone *

Performance Patterns

1. Async Request Reply * -
2. Static Content Hosting * - **welldoc**
3. Materialized View *
4. Backends for front ends * - **welldoc**
5. Cache Aside *
6. Index Table *
7. Throttling *
8. Rate Limiting *

Centralize Cross Cutting Concerns

1. Gateway Aggregation * - **welldoc**
2. Gateway Routing * - **welldoc**
3. Gateway Offloading * - **welldoc**
4. Gatekeeper * - **welldoc**
5. Side Car * - **welldoc**
6. Ambassador * - **welldoc**
7. Federated Identity * - **welldoc**
8. External Config store * - **welldoc**

Legacy modernisation Patterns

1. Strangler * - **welldoc**
2. Anti Corruption Layer * - **welldoc**

Reliability Availability Patterns

1. Retry *
2. Circuit breaker *
3. Bulk head *
4. Compensatable transaction *
5. SAGA *
6. Schedular Agent supervisor *
7. Health endpoint *

Communication Patterns

1. Pub sub *
2. Priority Queue *
3. Event Sourcing * -**welldoc**
4. Choreography * -**welldoc**
5. Leader Election (last weapon)
6. Sequential convoy *

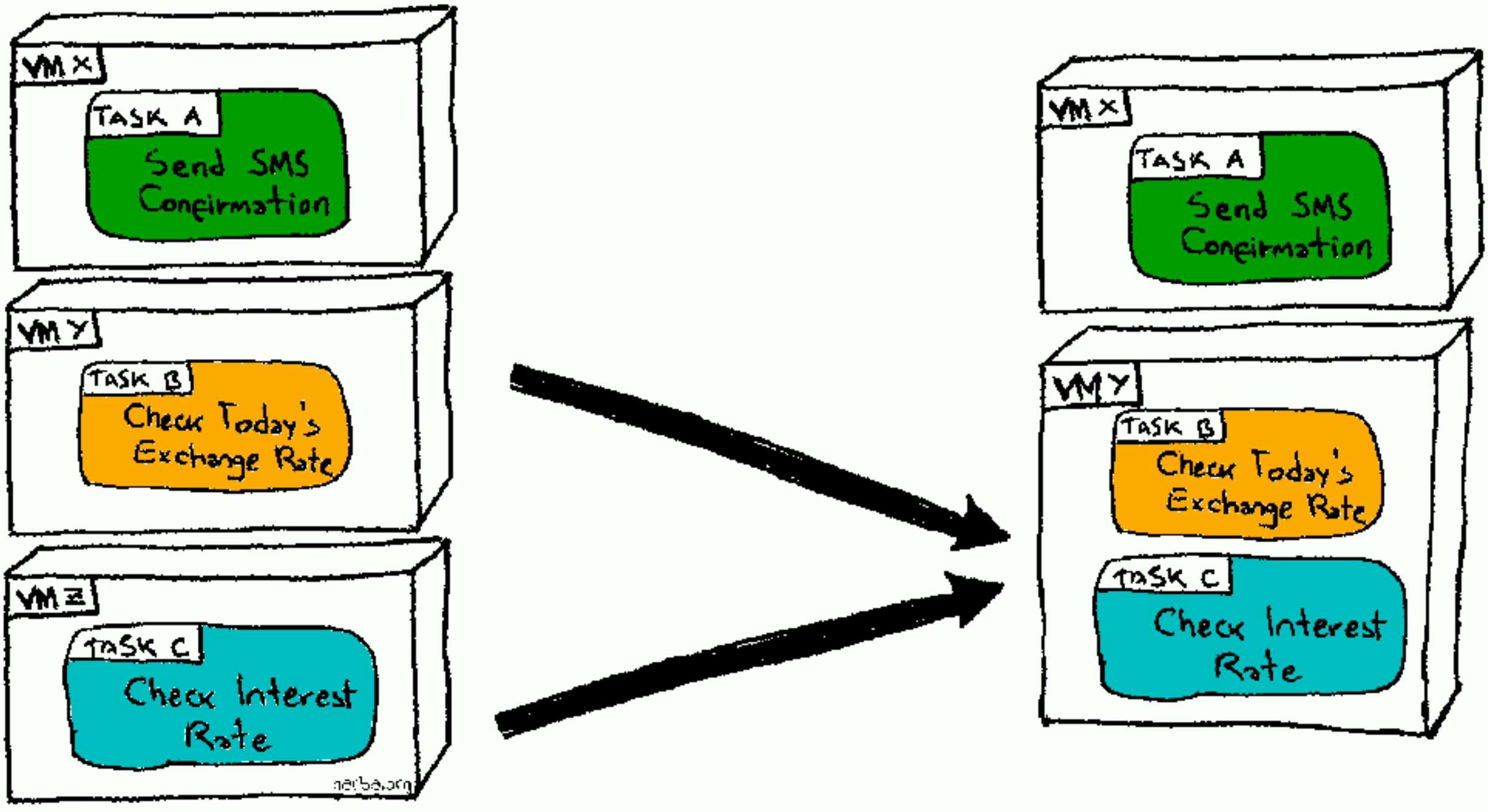
Maintainability

- Edge Workload config

Security

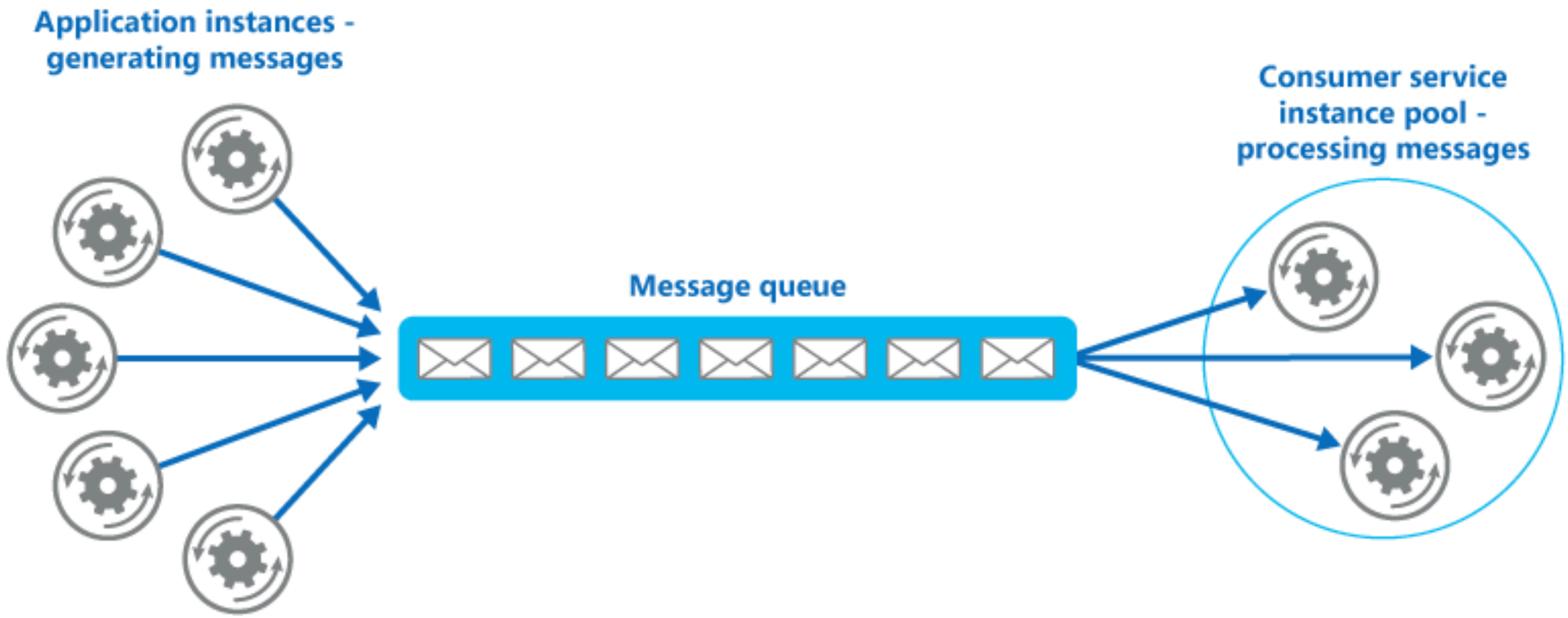
- Valet Key

Compute Resource Consolidation Pattern



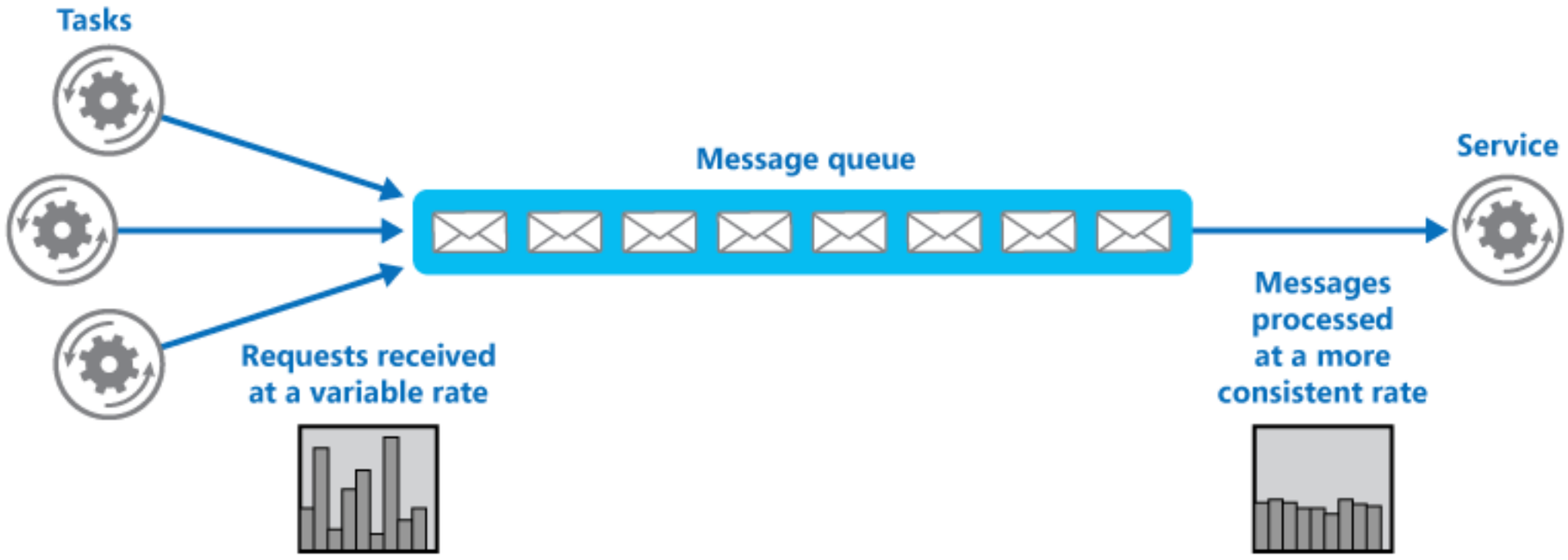
Scalability Patterns

Competing Consumers pattern



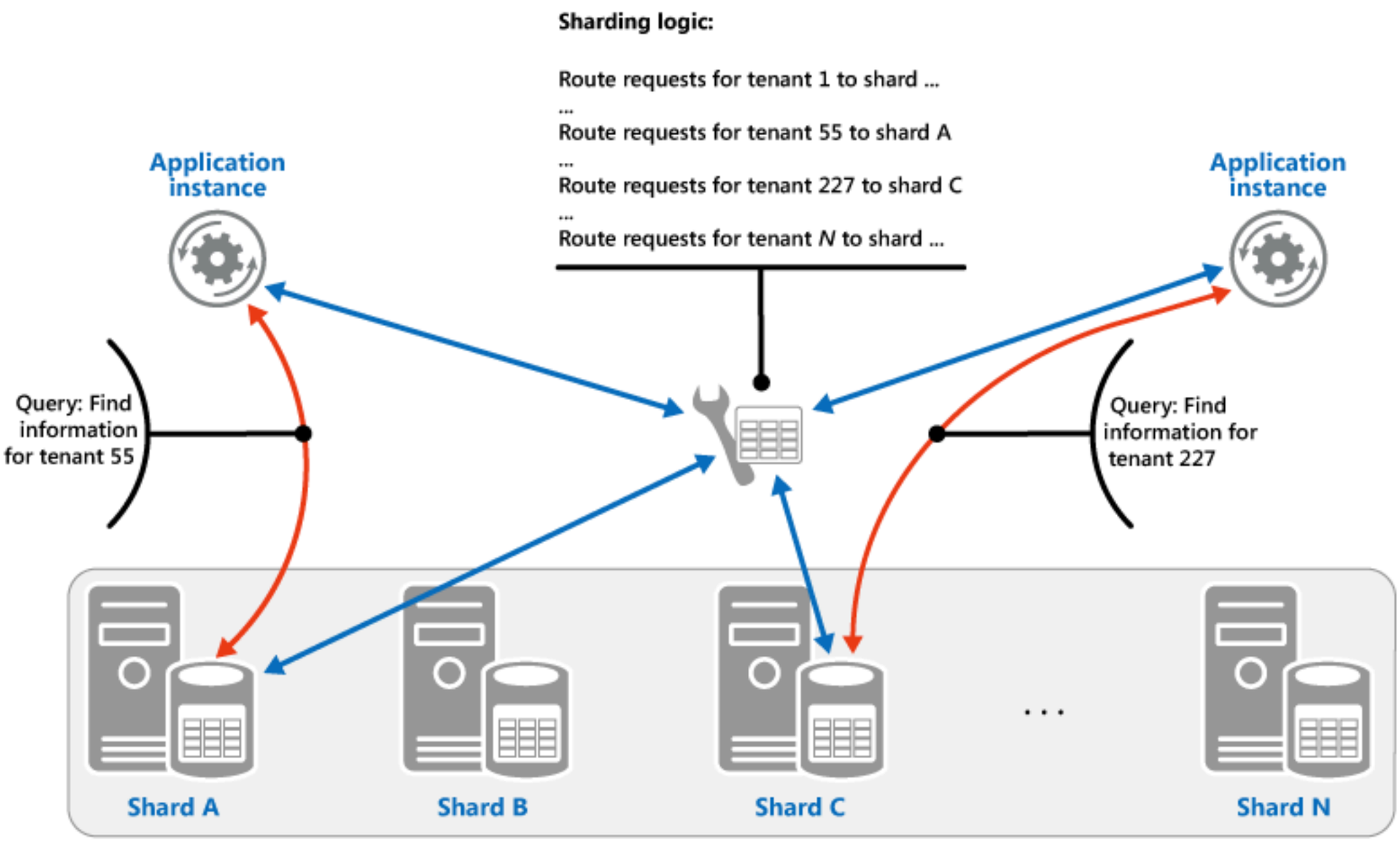
- The volume of work is highly variable, requiring a scalable solution.

Queue-Based Load Leveling pattern



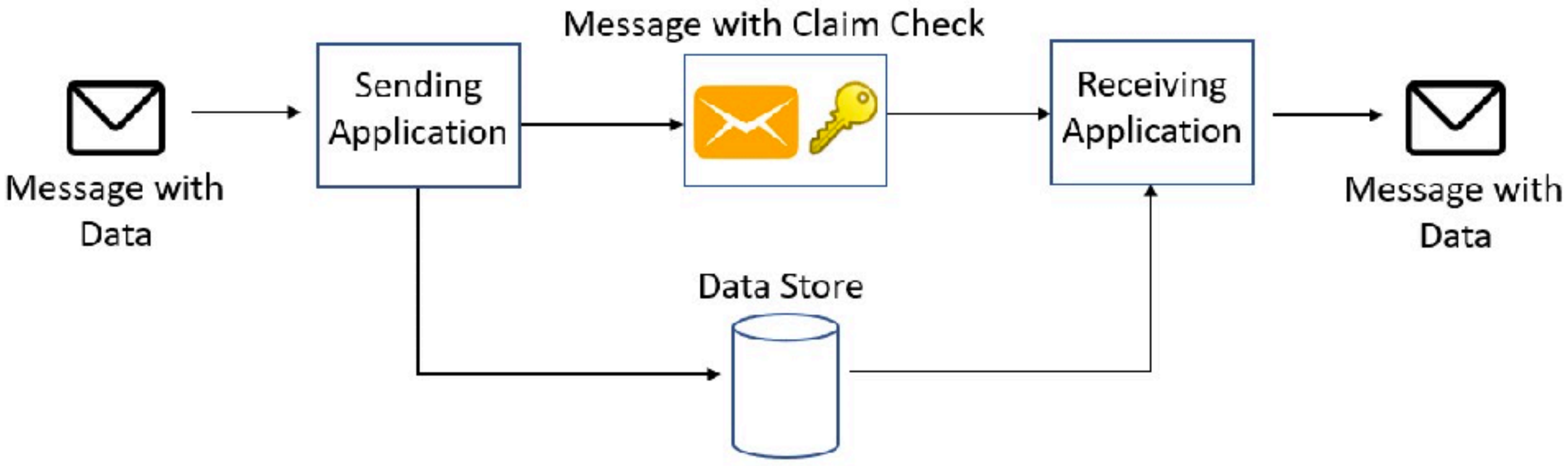
useful to any application that uses services that are subject to overloading.

Sharding



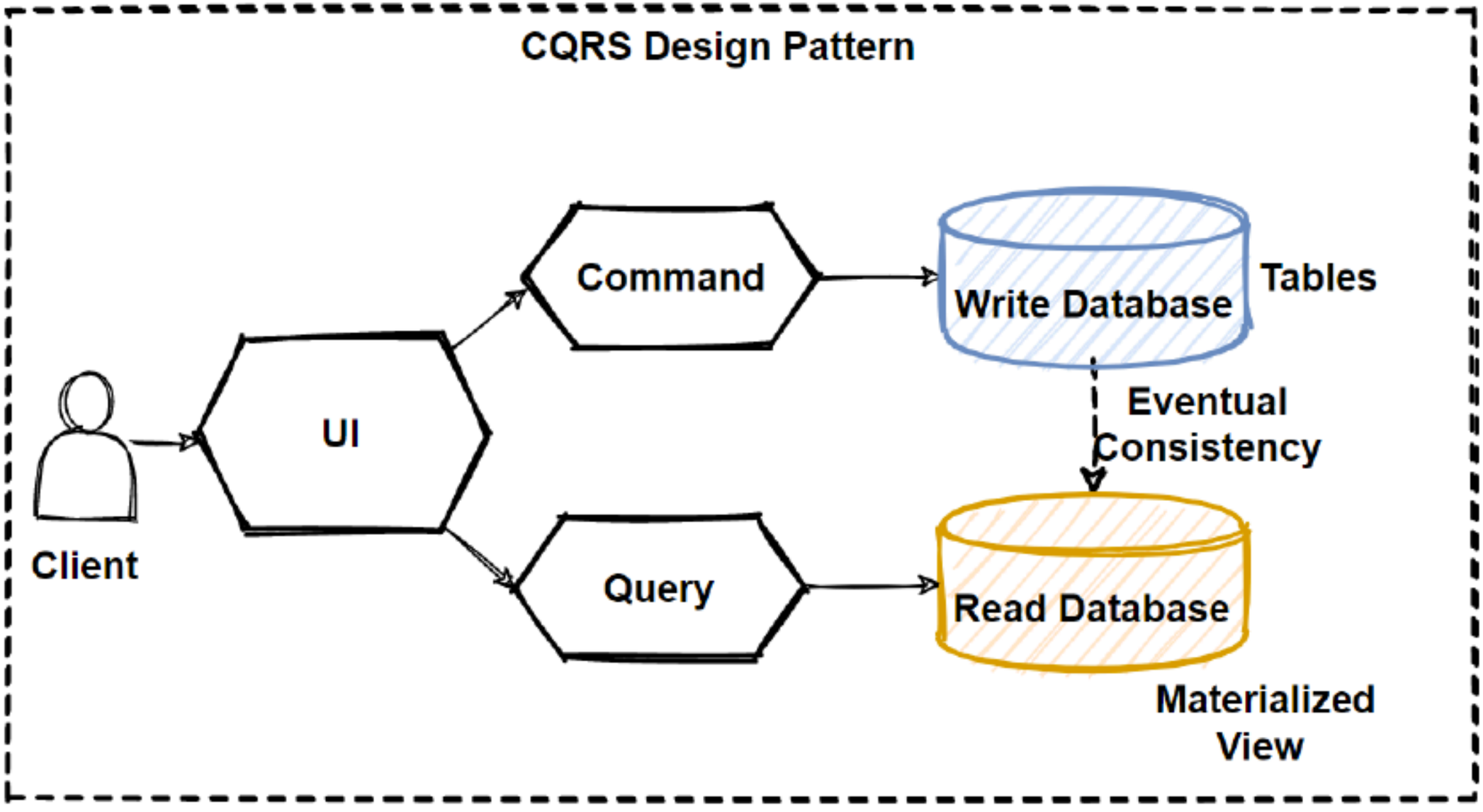
when a data store is likely to need to scale beyond the resources available to a single storage node

Claim-Check pattern



used whenever a message cannot fit the supported message limit of the chosen message bus

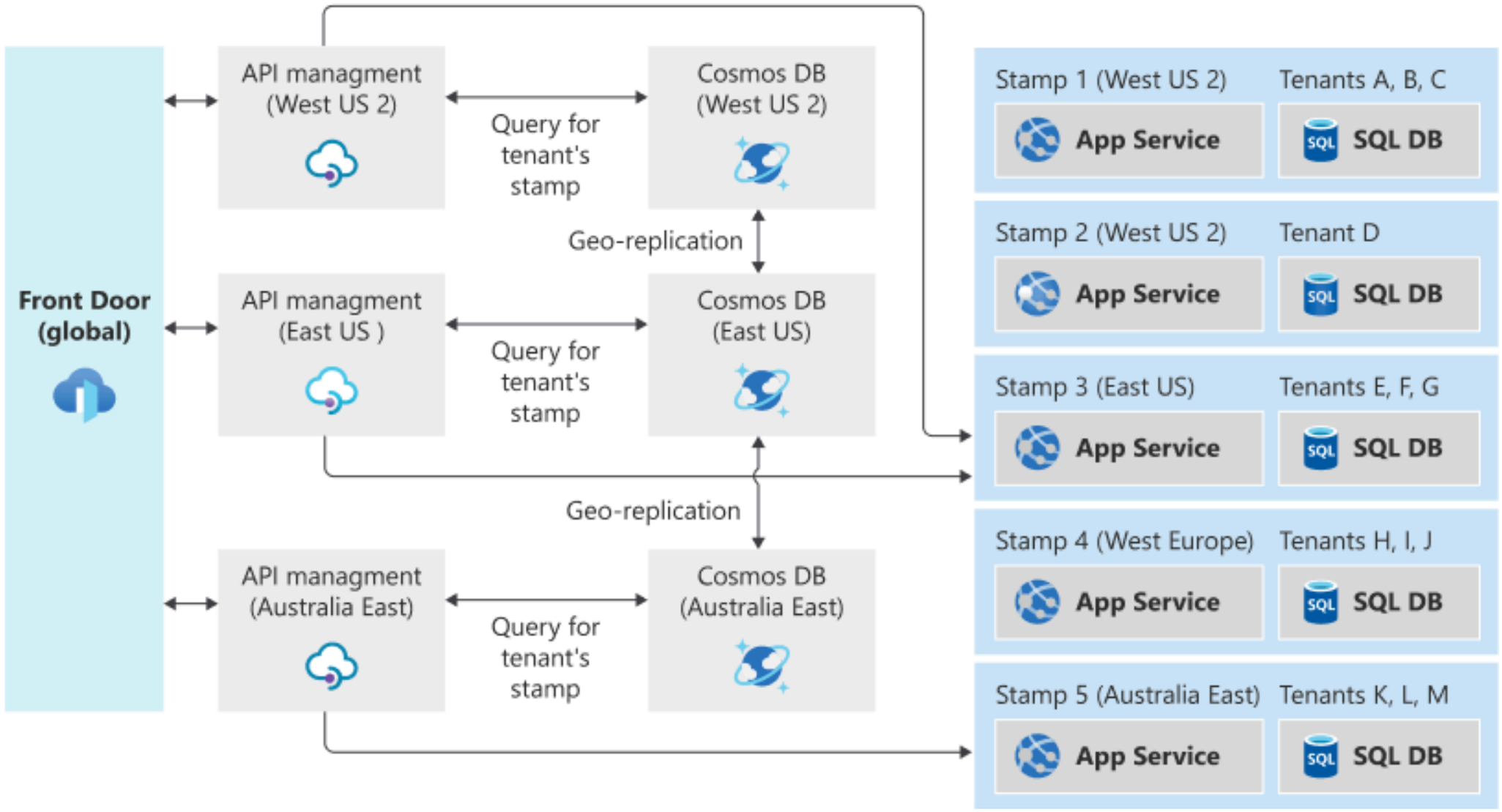
CQRS pattern



- Scenarios where one team of developers can focus on the complex domain model that is part of the write model, and another team can focus on the read model and the user interfaces.

Consider applying CQRS to limited sections of your system where it will be most valuable.

Deployment Stamps pattern



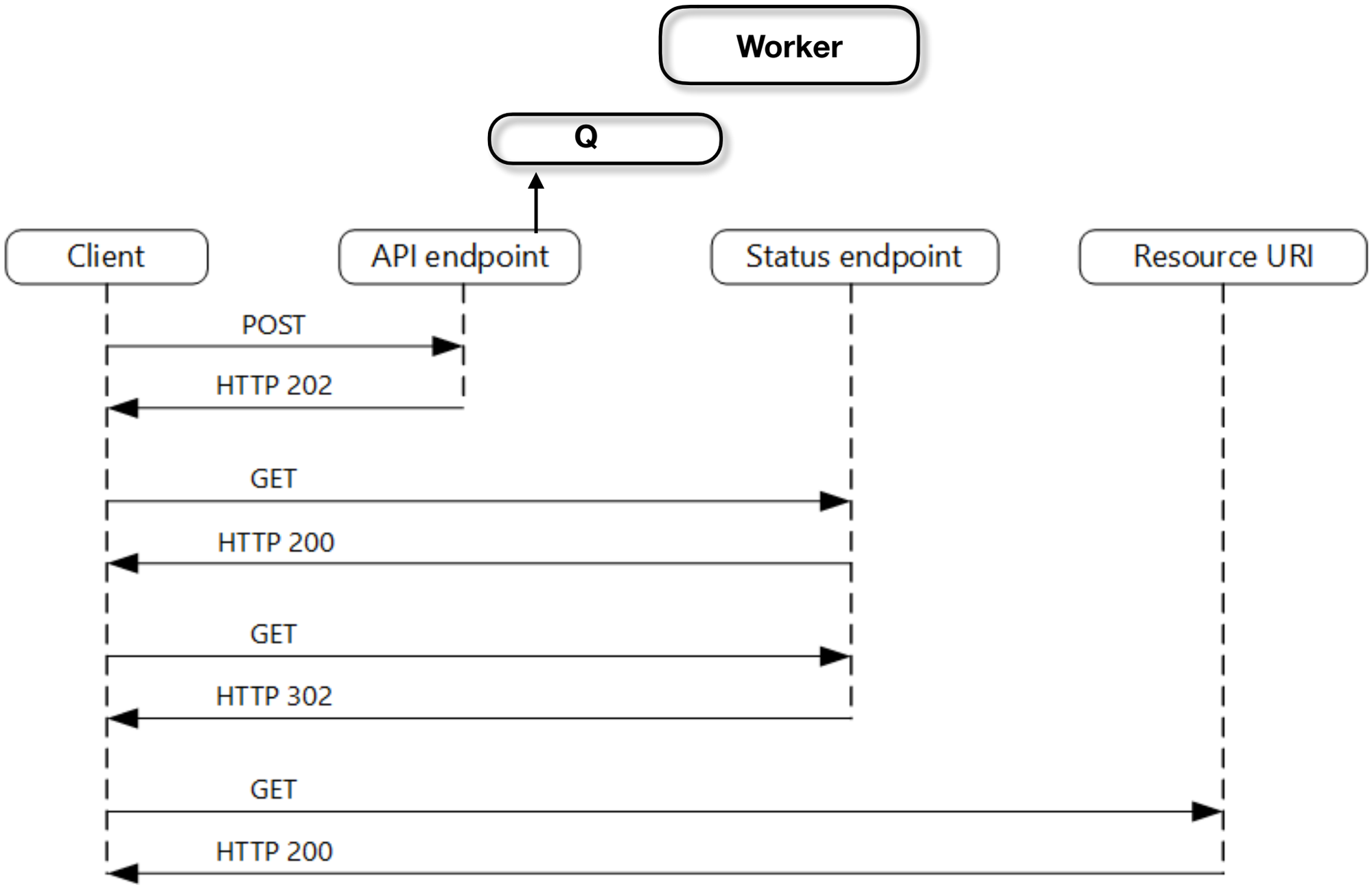
In a deployment stamp architecture, multiple independent instances of your system are deployed and contain a subset of your customers and users. In geodes, all instances can serve requests from any users, but this architecture is often more complex to design and build

If the user travels to California and then accesses the system, their connection will likely be routed through the West US 2 region because that's closest to where they are geographically when they make the request. However, the request has to ultimately be served by stamp 3, because that's where their data is stored. The traffic routing system ensures that the request is routed to the correct stamp.

Each geode is behind a global load balancer, and uses a geo-replicated read-write service like Azure Cosmos DB to host the data plane, ensuring cross-geode data consistency. Data replication services ensure that data stores are identical across geodes, so *all* requests can be served from *all* geodes. you can use Azure Traffic Manager or Azure Front Door for fronting the geodes, while Azure Cosmos DB can act as the replication backplane.

Performance Patterns

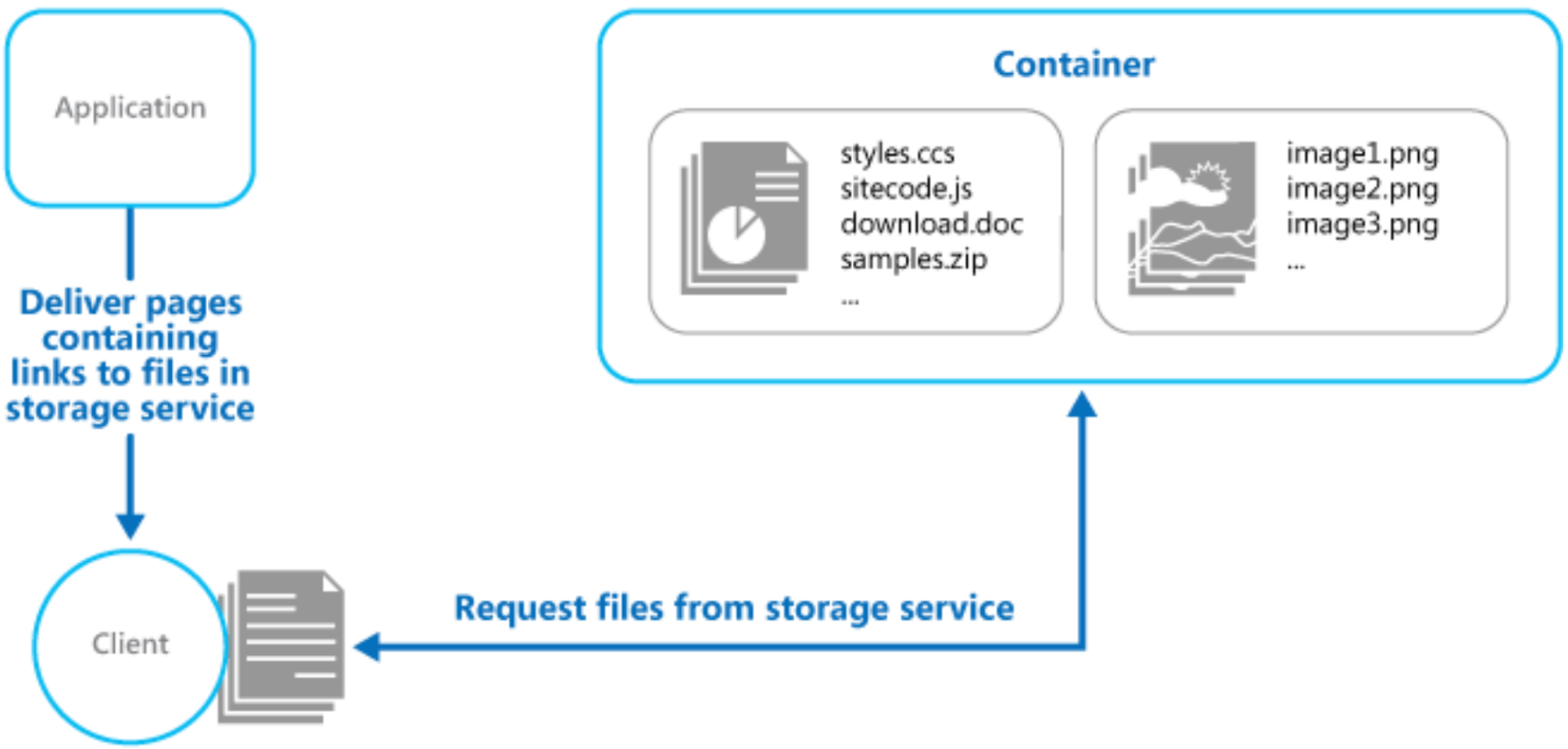
Asynchronous Request-Reply



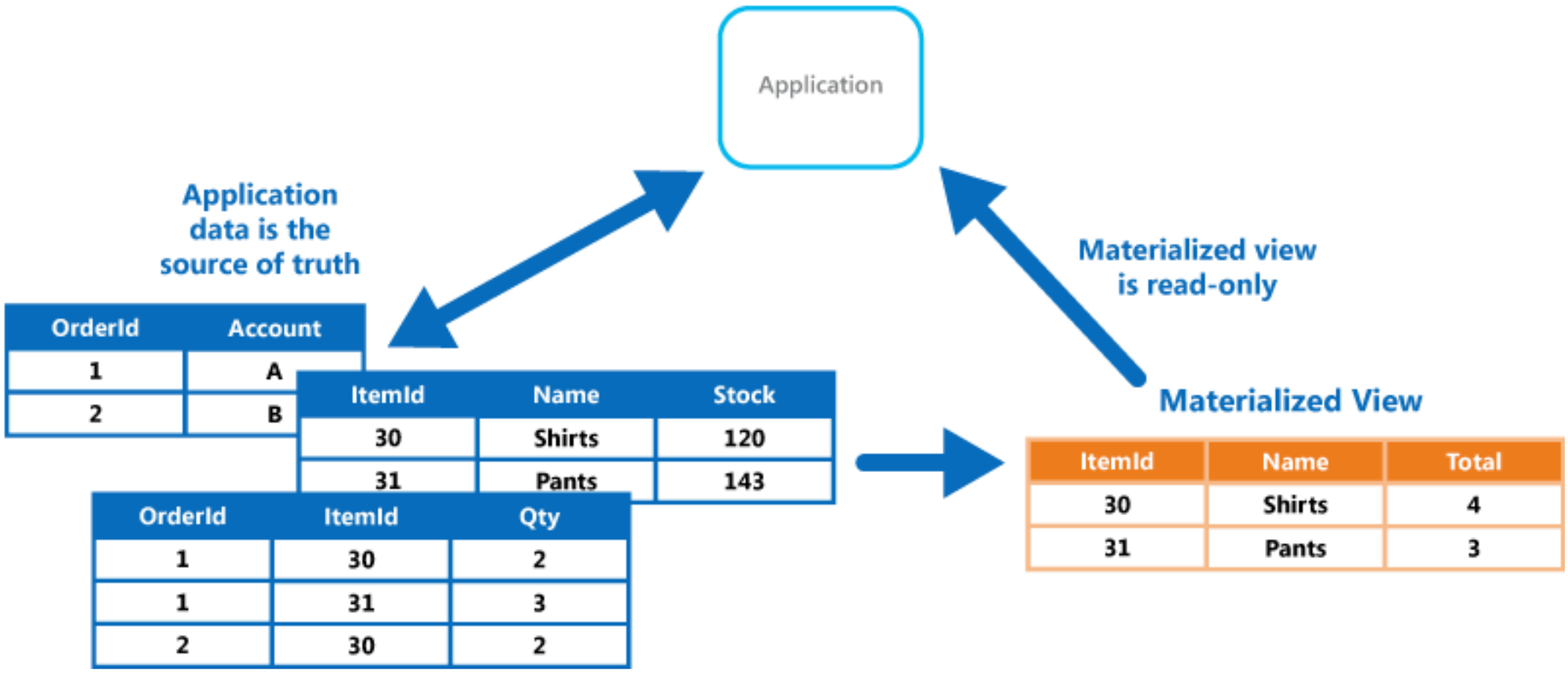
- Service calls that need to be integrated with legacy architectures that don't support modern callback technologies such as Notification, WebSockets or webhooks.

Static Content Hosting

Html
Js
Css
Image
— — — -
Jsp
Aspx
Rest api

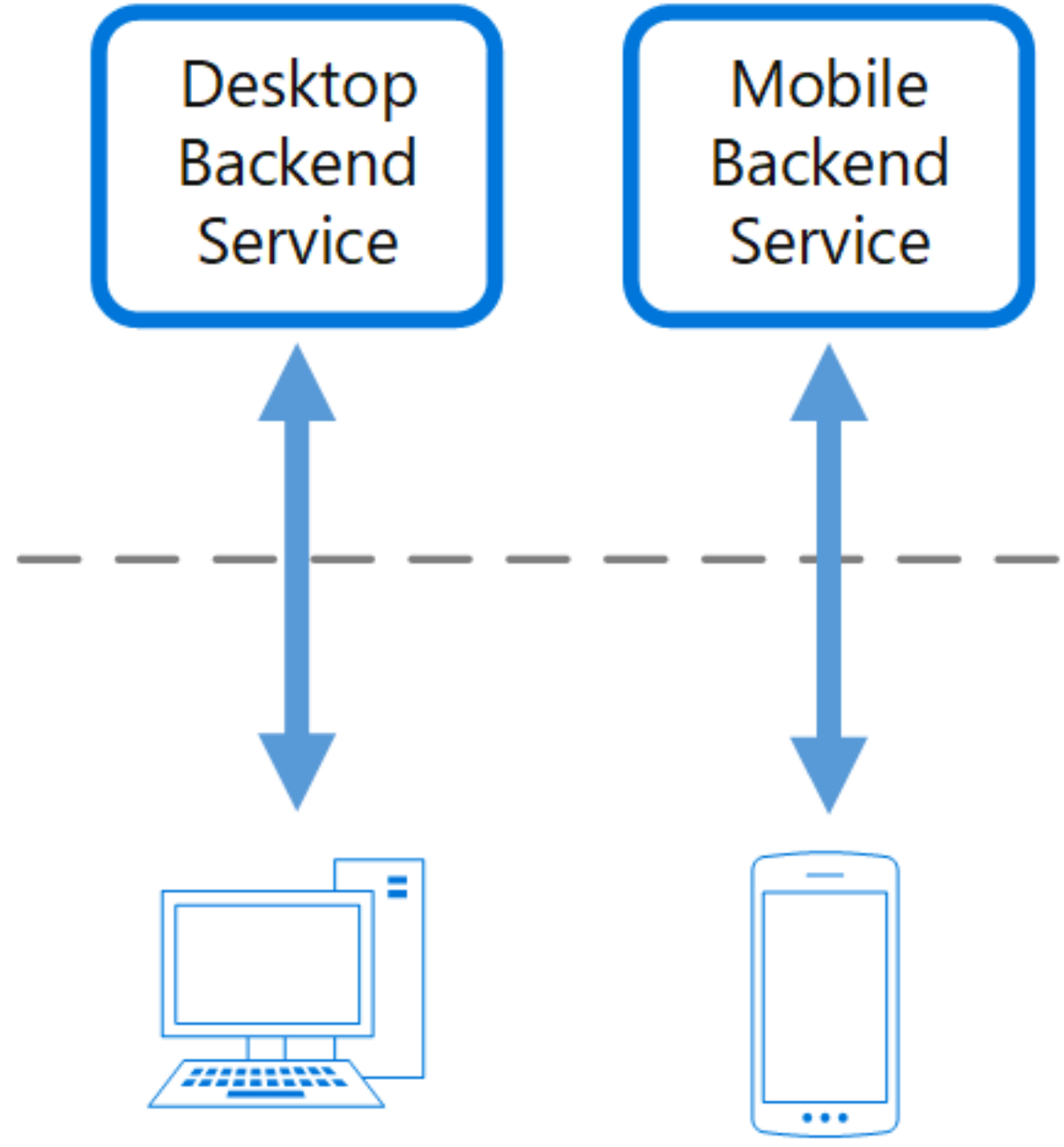


Materialized View



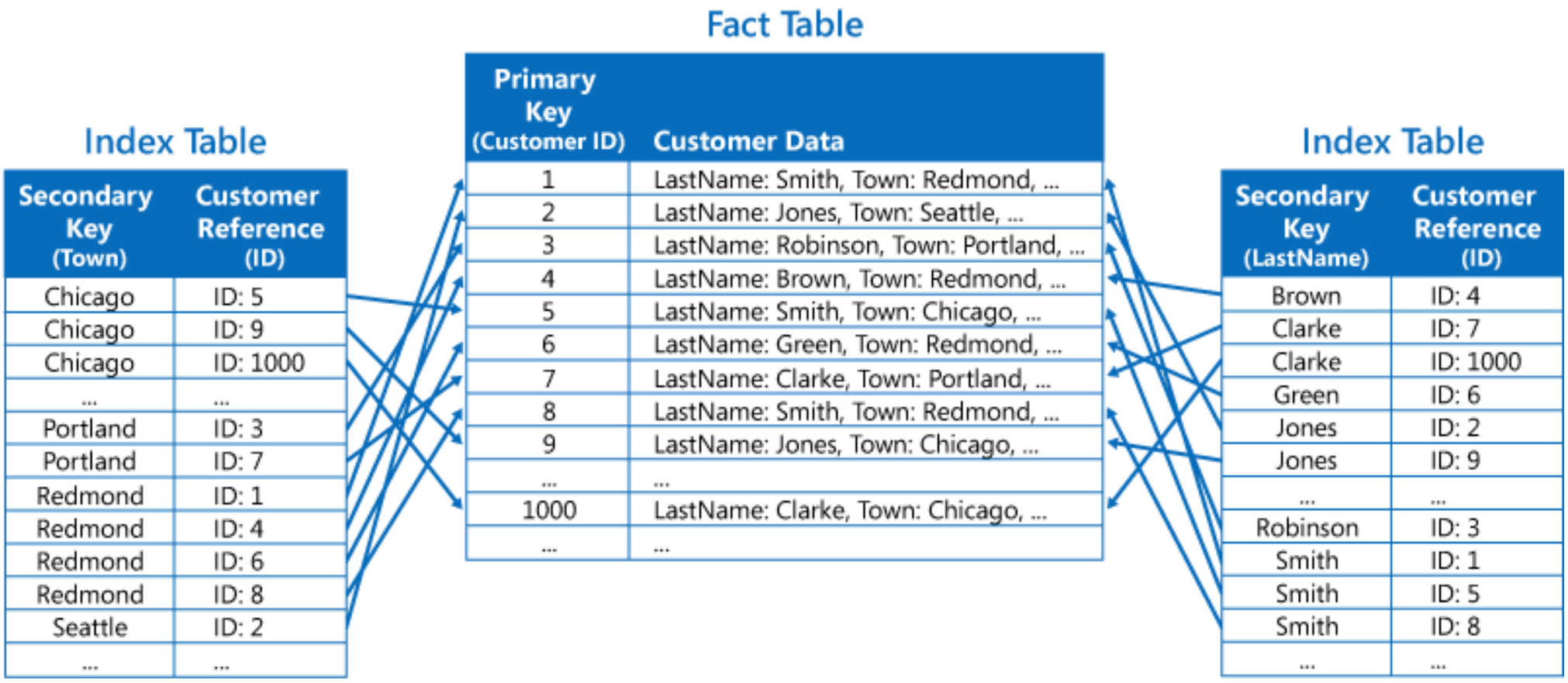
data that's difficult to query directly, or where queries must be very complex to extract data that's stored in a normalized, semi-structured, or unstructured way.

Backends for Frontends



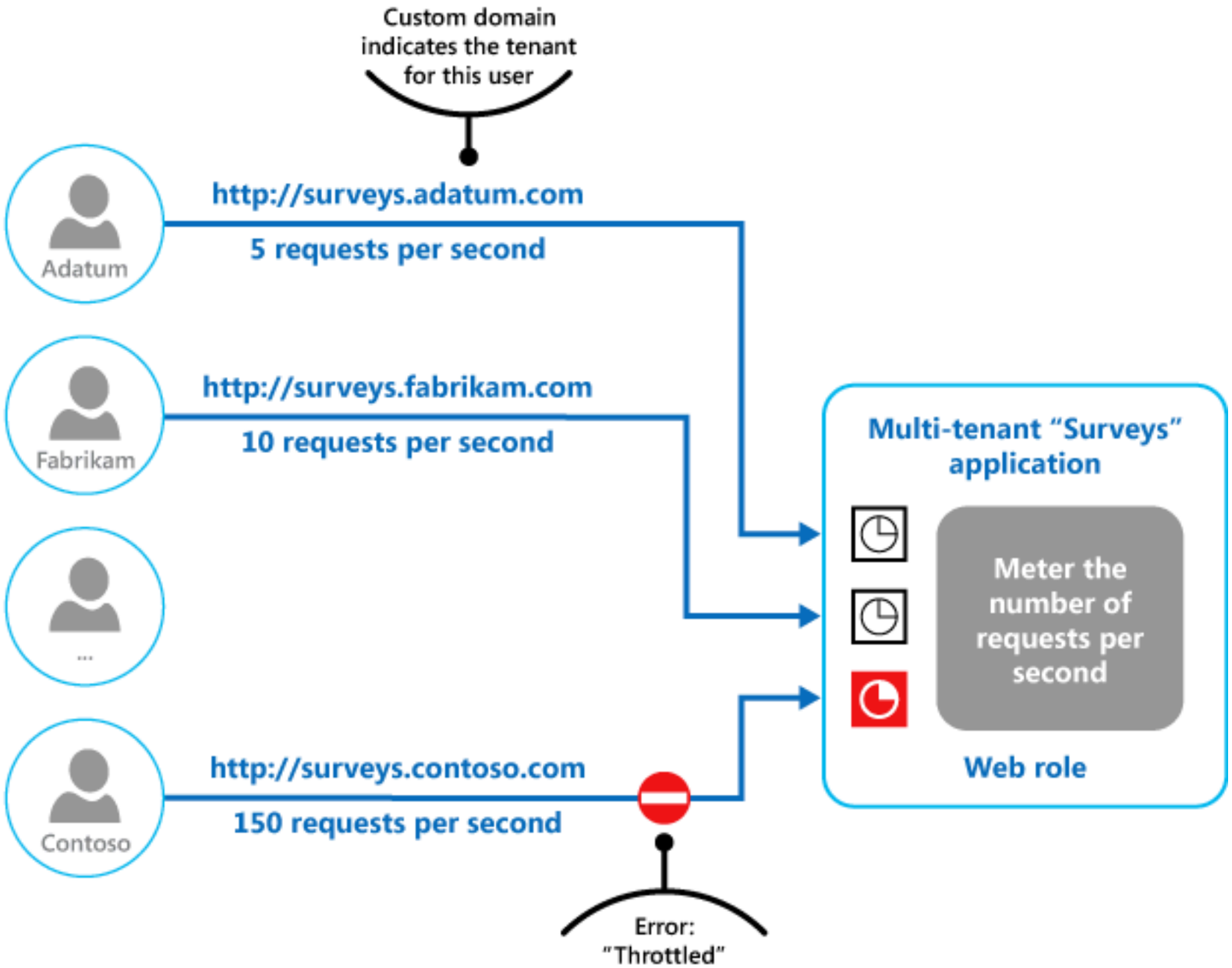
useful when you want to avoid customizing a single backend for multiple interfaces

Index Table



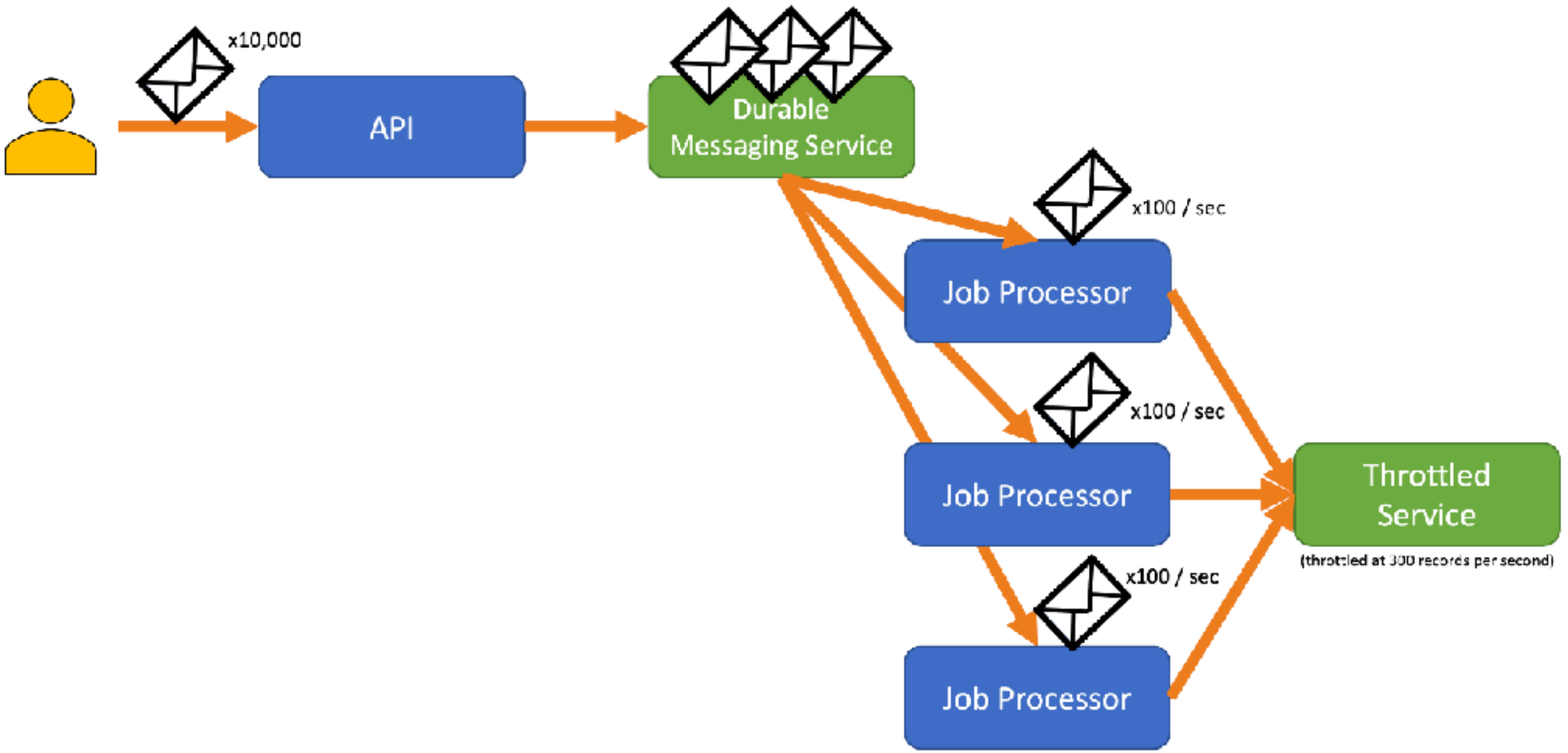
Use this pattern to improve query performance when an application frequently needs to retrieve data by using a key other than the primary

Throttling



To handle bursts

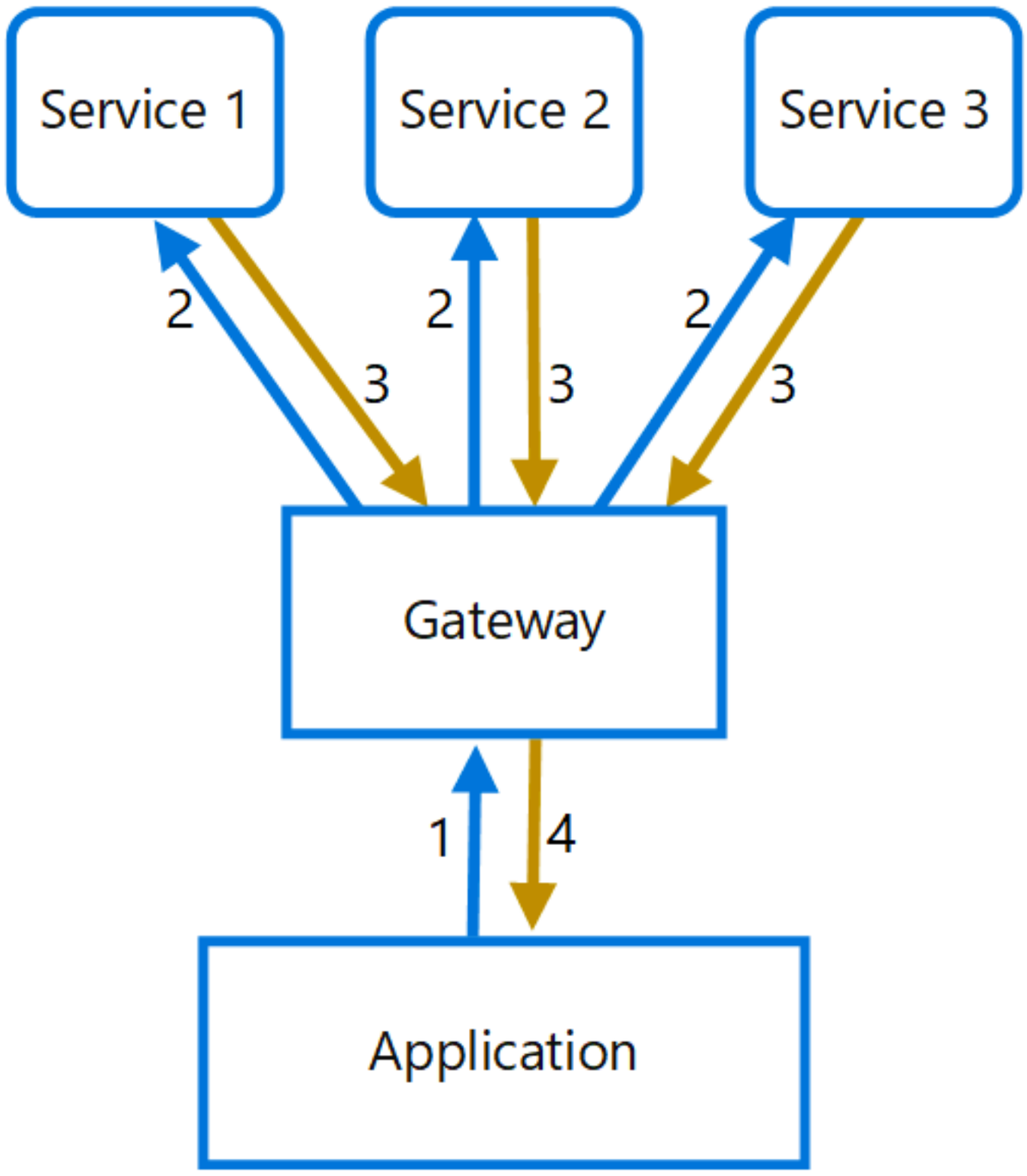
Rate Limiting



- Reduce throttling errors raised by a throttle-limited service.

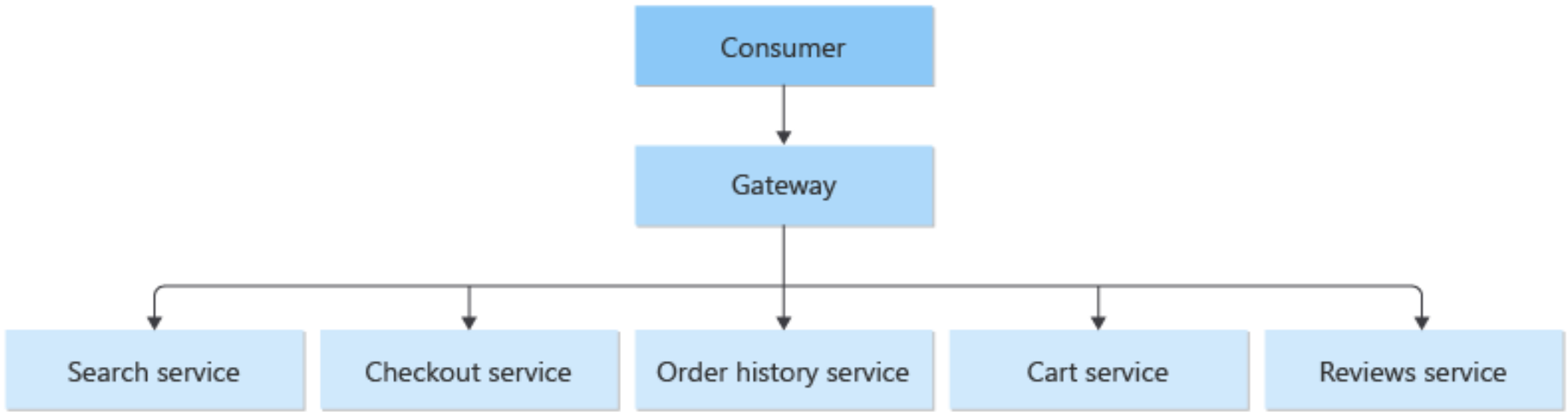
Centralize Cross Cutting Concerns

Gateway Aggregation



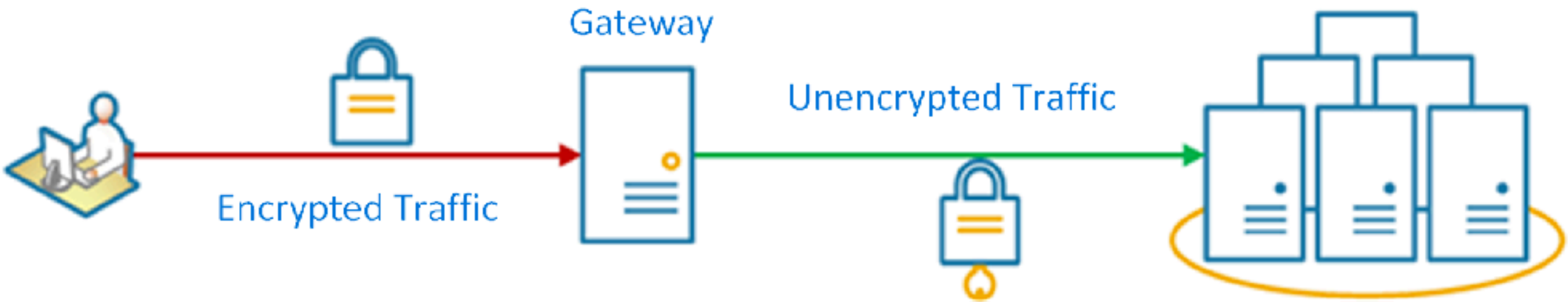
- A client needs to communicate with multiple backend services to perform an operation.

Gateway Routing



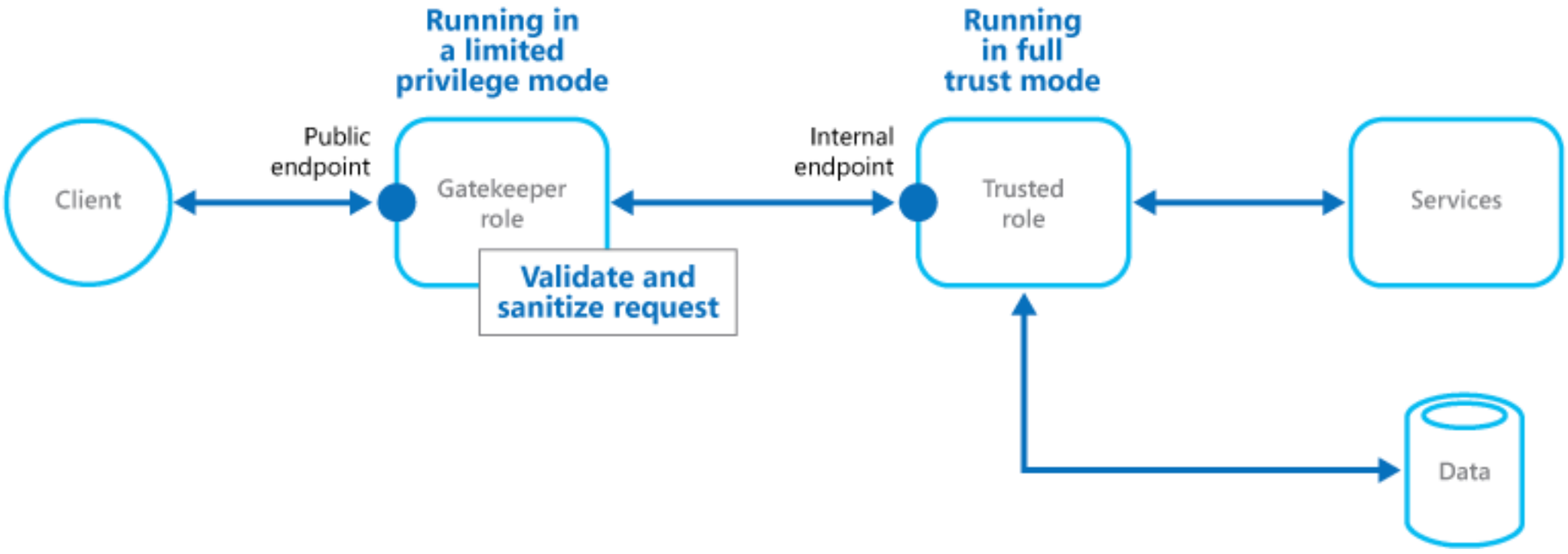
- abstract backend services from the clients
- Routing lets you control what version of the service is presented to the clients

Gateway Offloading



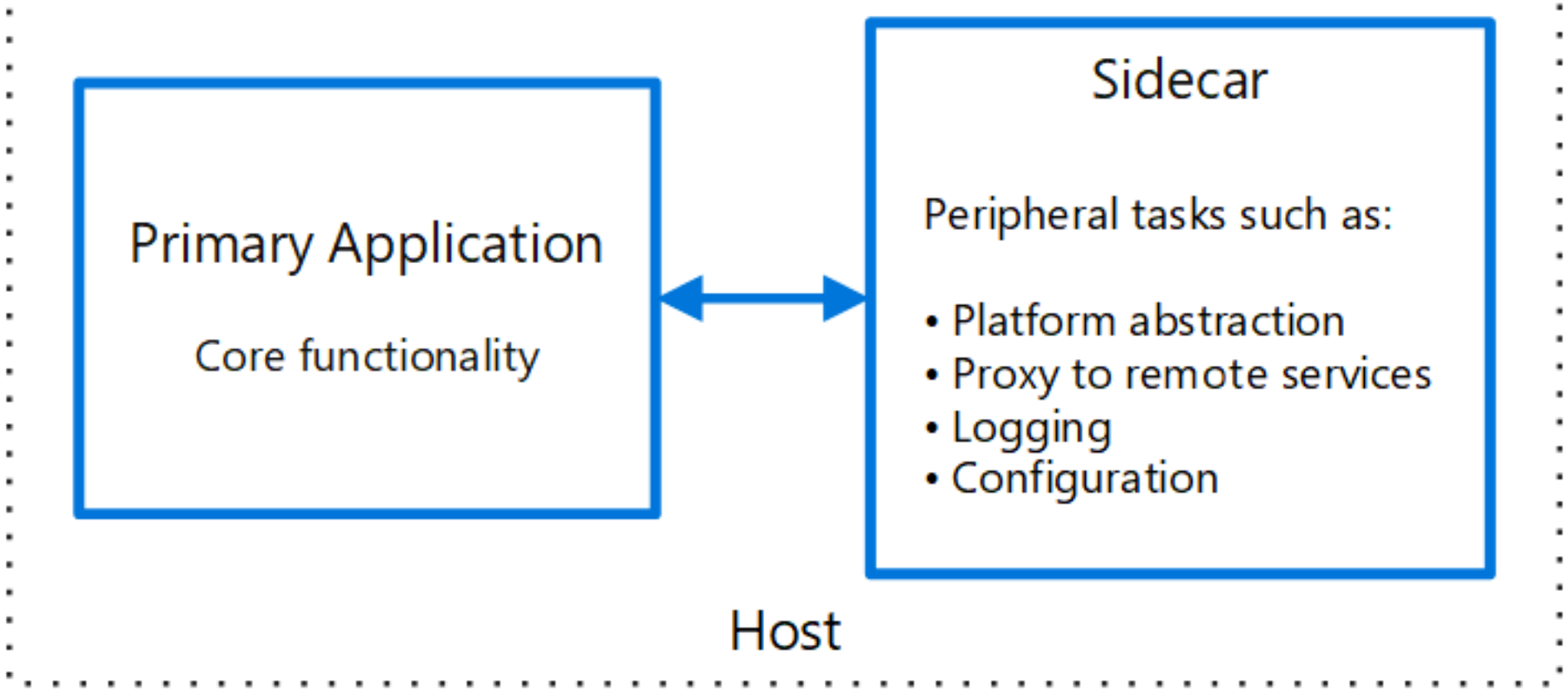
- wish to move the responsibility for issues such as network security, throttling, or other network boundary concerns to a more specialized team.

Gatekeeper



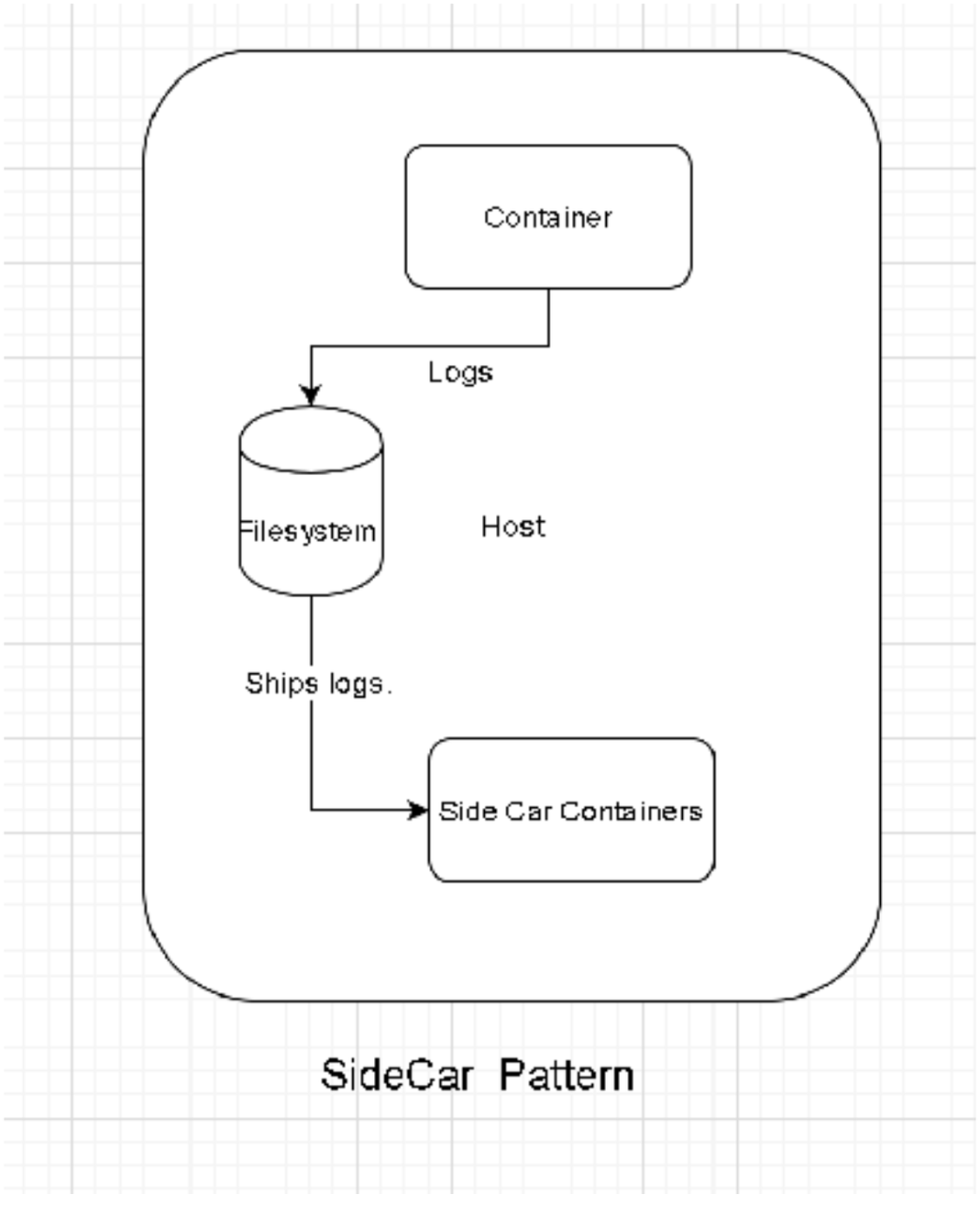
acts like a firewall in a typical network topography. It allows the gatekeeper to examine requests and make a decision about whether to pass the request on to the trusted host that performs the required tasks.

Sidecar

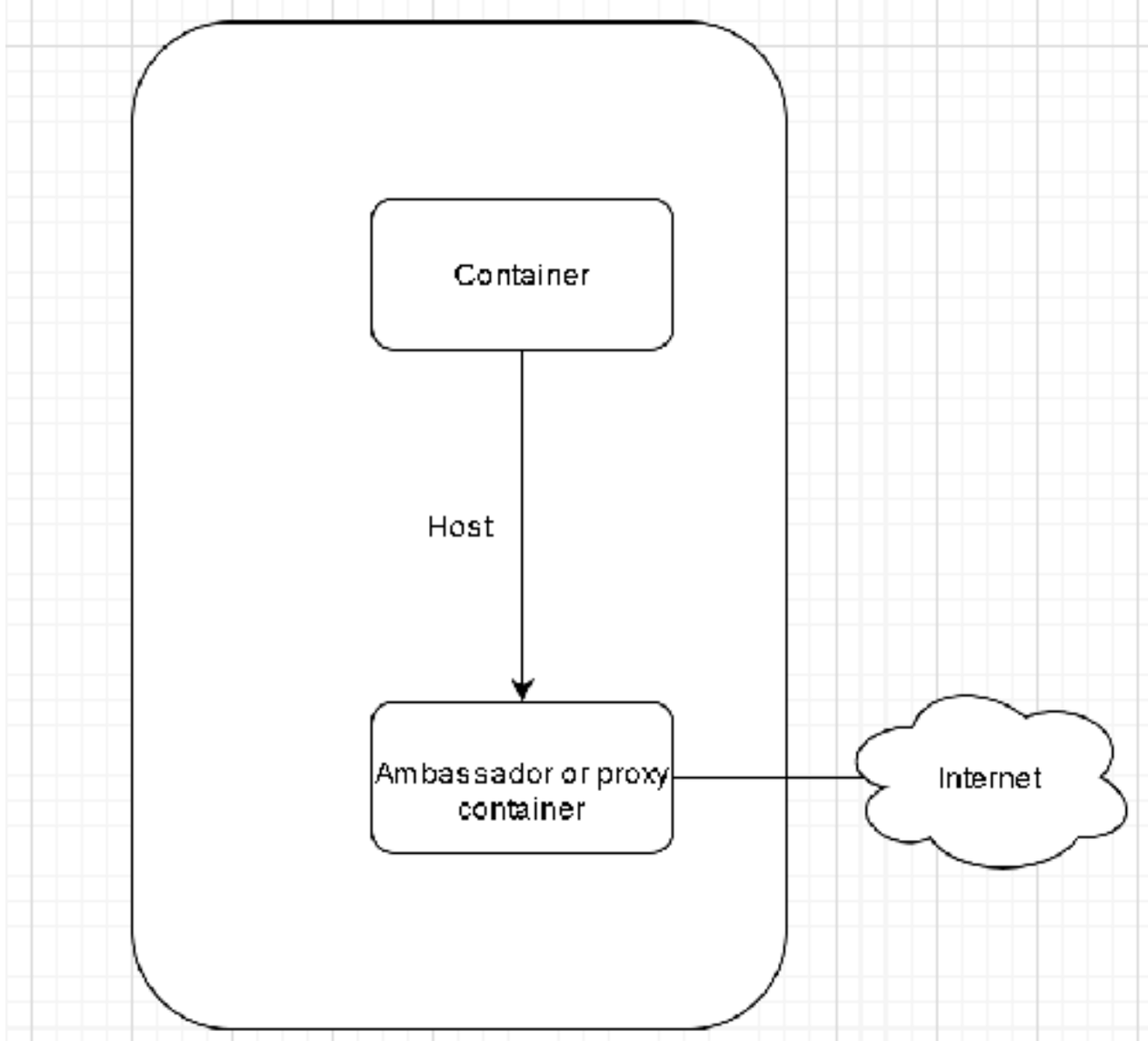


- Because of its proximity to the primary application, there's no significant latency when communicating between them.

Ambassador



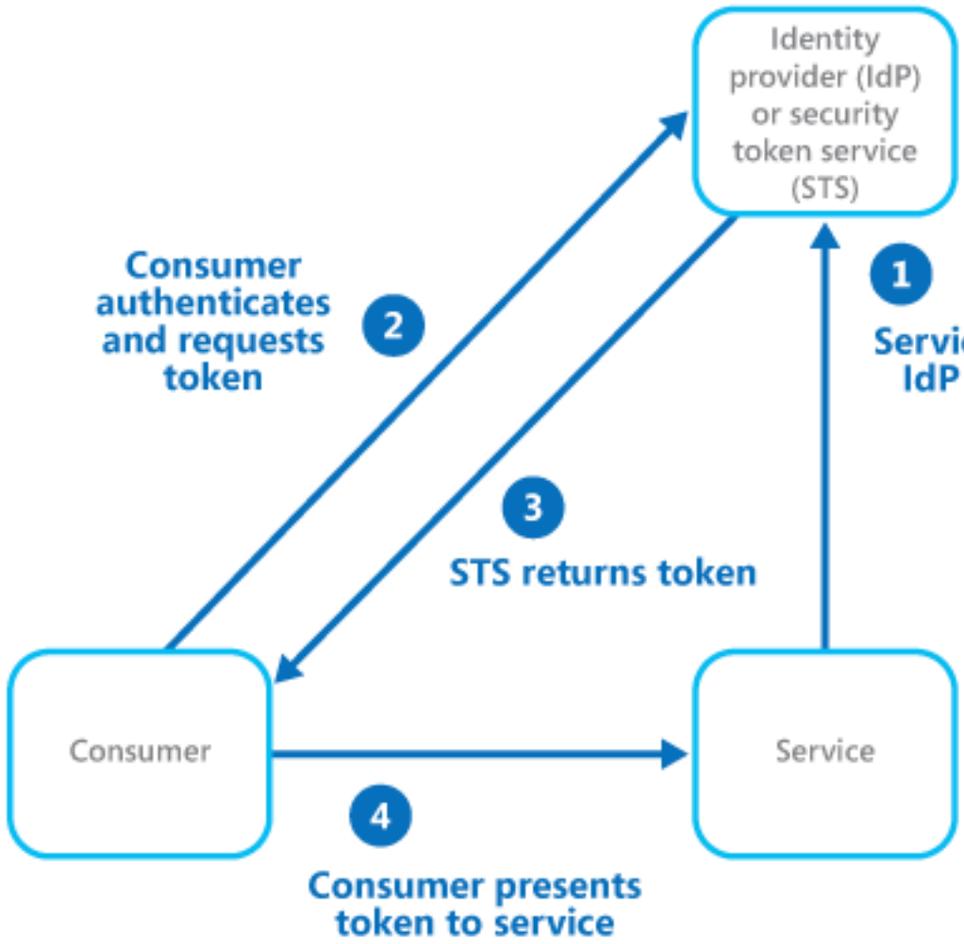
SideCar Pattern



Ambassador Pattern

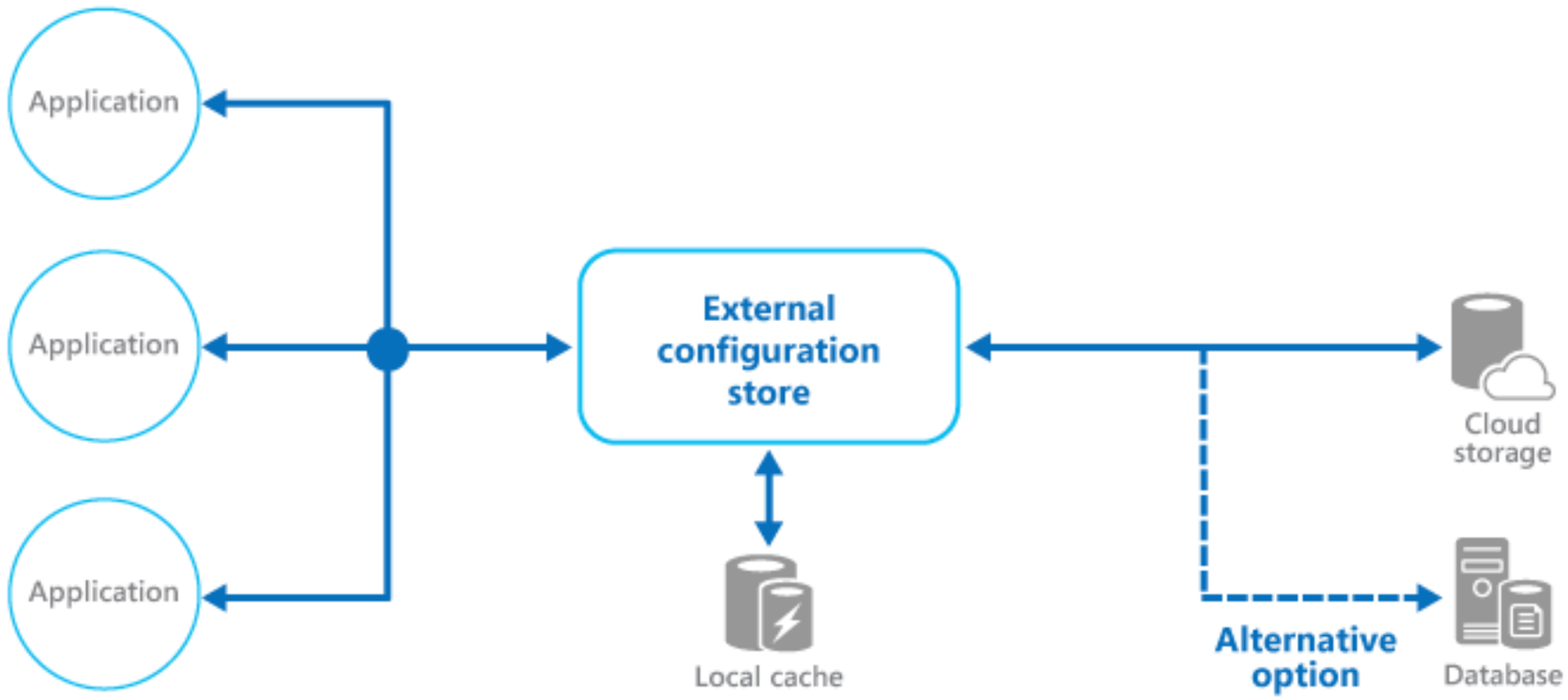
This pattern is almost the same as sidecar pattern the difference is that in ambassador pattern every interaction to the world goes through the ambassador container.

Federated Identity



Delegate authentication to an external identity provider.

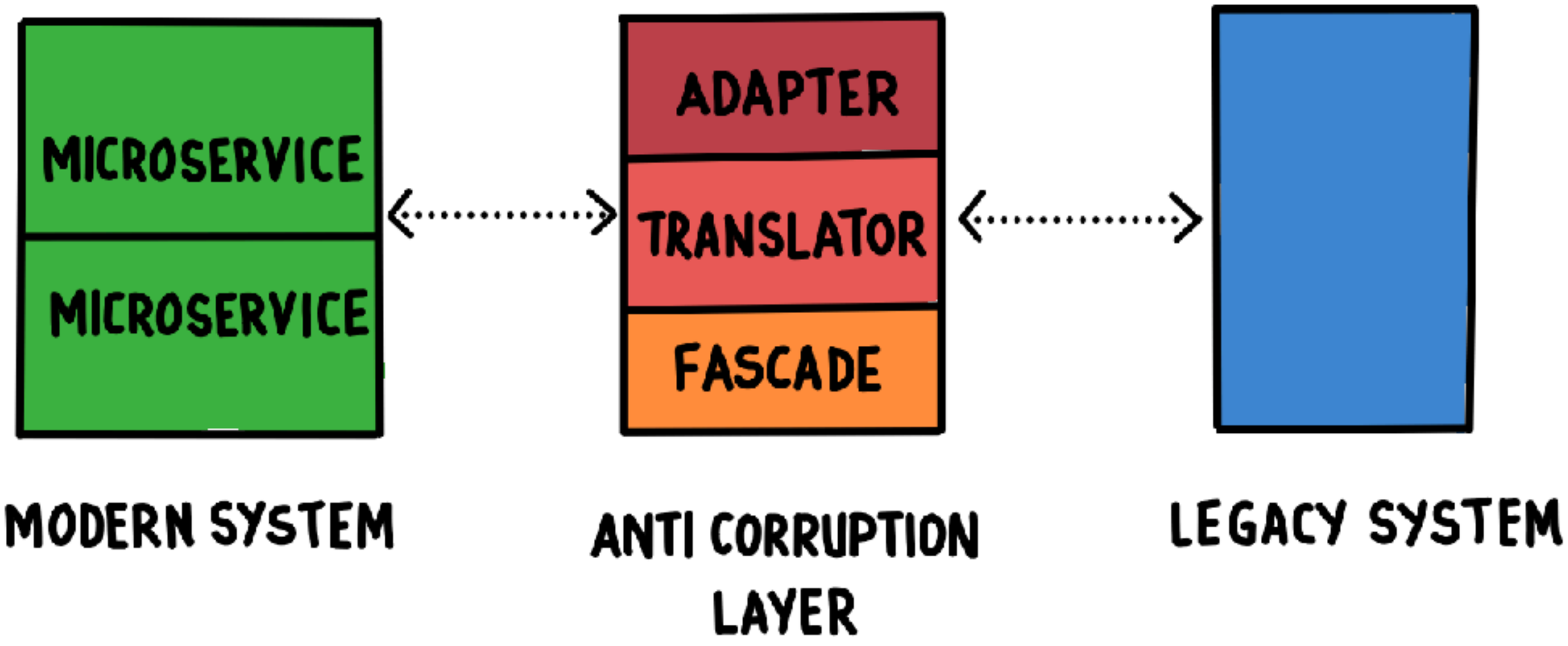
External Configuration Store



Move configuration information out of the application deployment package to a centralized location.

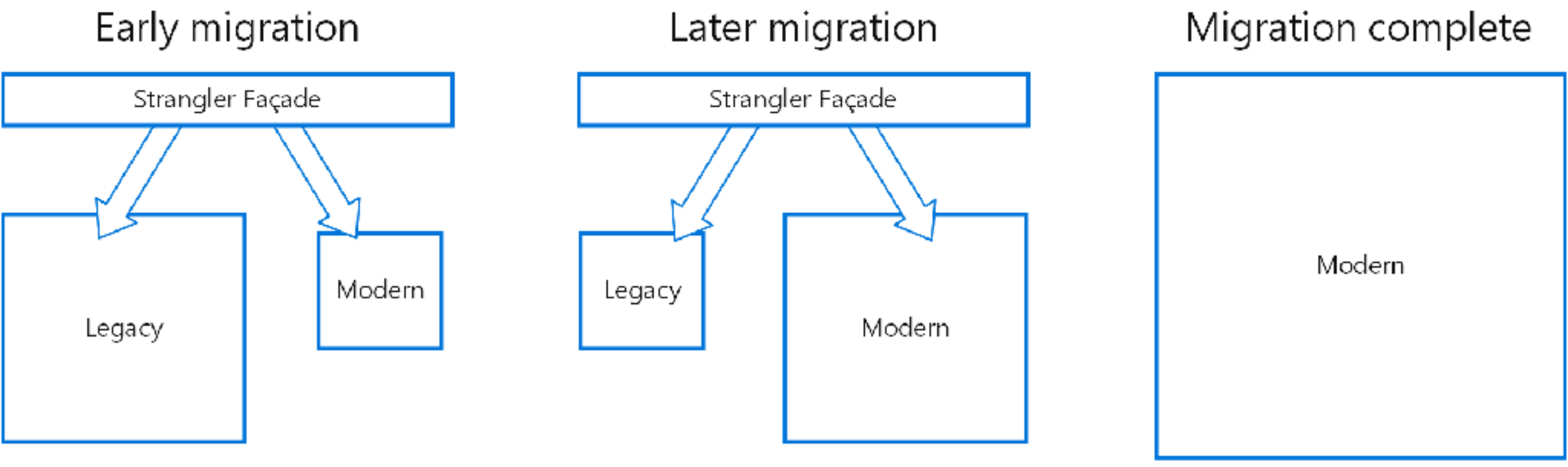
Legacy modernisation

Anti-corruption Layer



- How do you prevent a legacy monolith's domain model from polluting the domain model of a new service.
- The Anti Corruption Layer uses a combination of Fascade, Adpater, and Translators to isolate the system

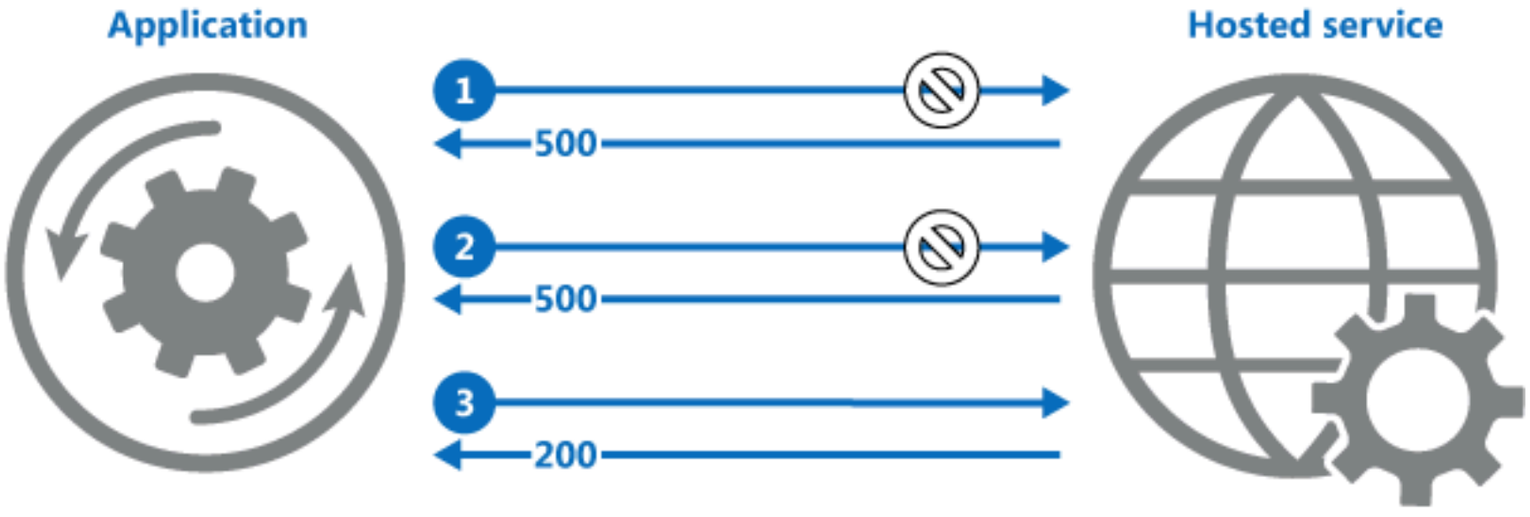
Strangler Fig



Incrementally replace specific pieces of functionality with new applications and services. Create a façade that intercepts requests going to the backend legacy system. The façade routes these requests either to the legacy application or the new services.

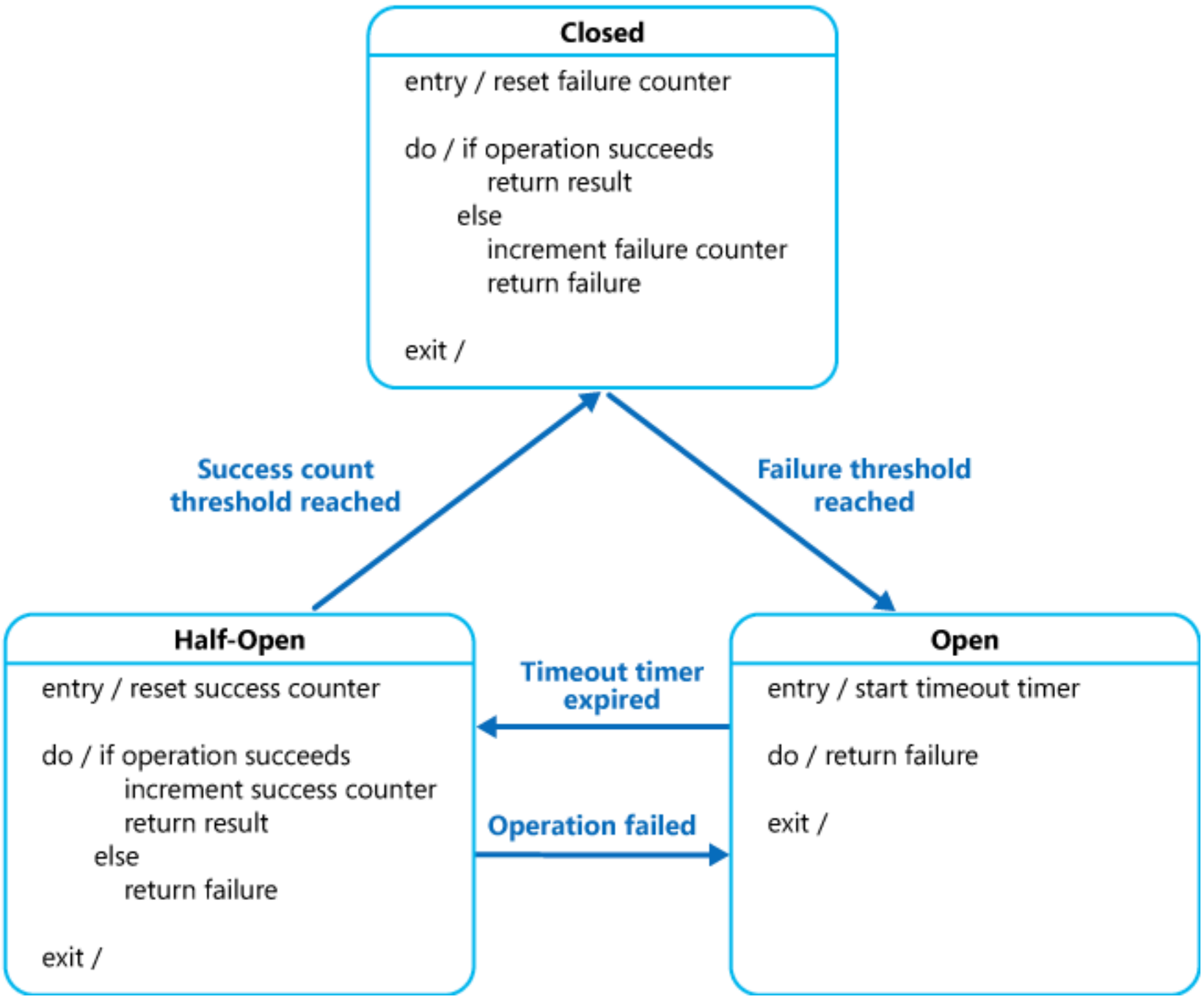
Reliability Patterns

Retry



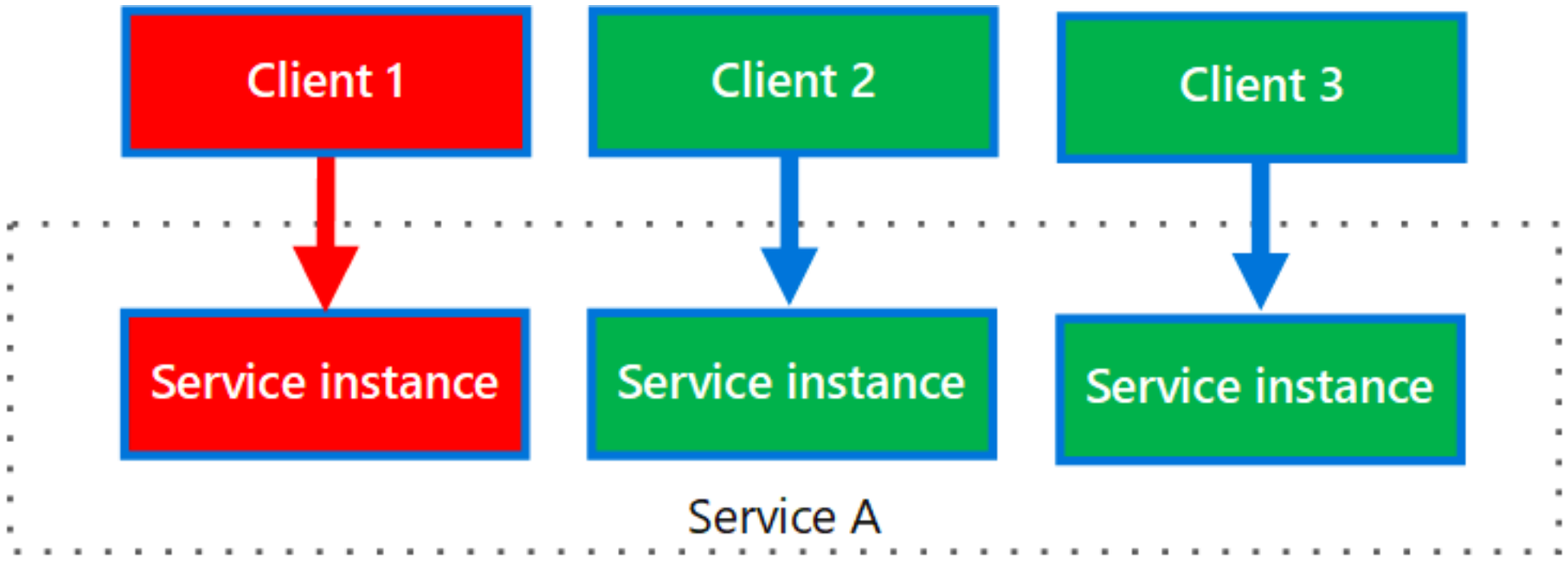
1: Application invokes operation on hosted service. The request fails, and the service host responds with HTTP response code 500 (internal server error).
2: Application waits for a short interval and tries again. The request still fails with HTTP response code 500.
3: Application waits for a longer interval and tries again. The request succeeds with HTTP response code 200 (OK).

Circuit Breaker



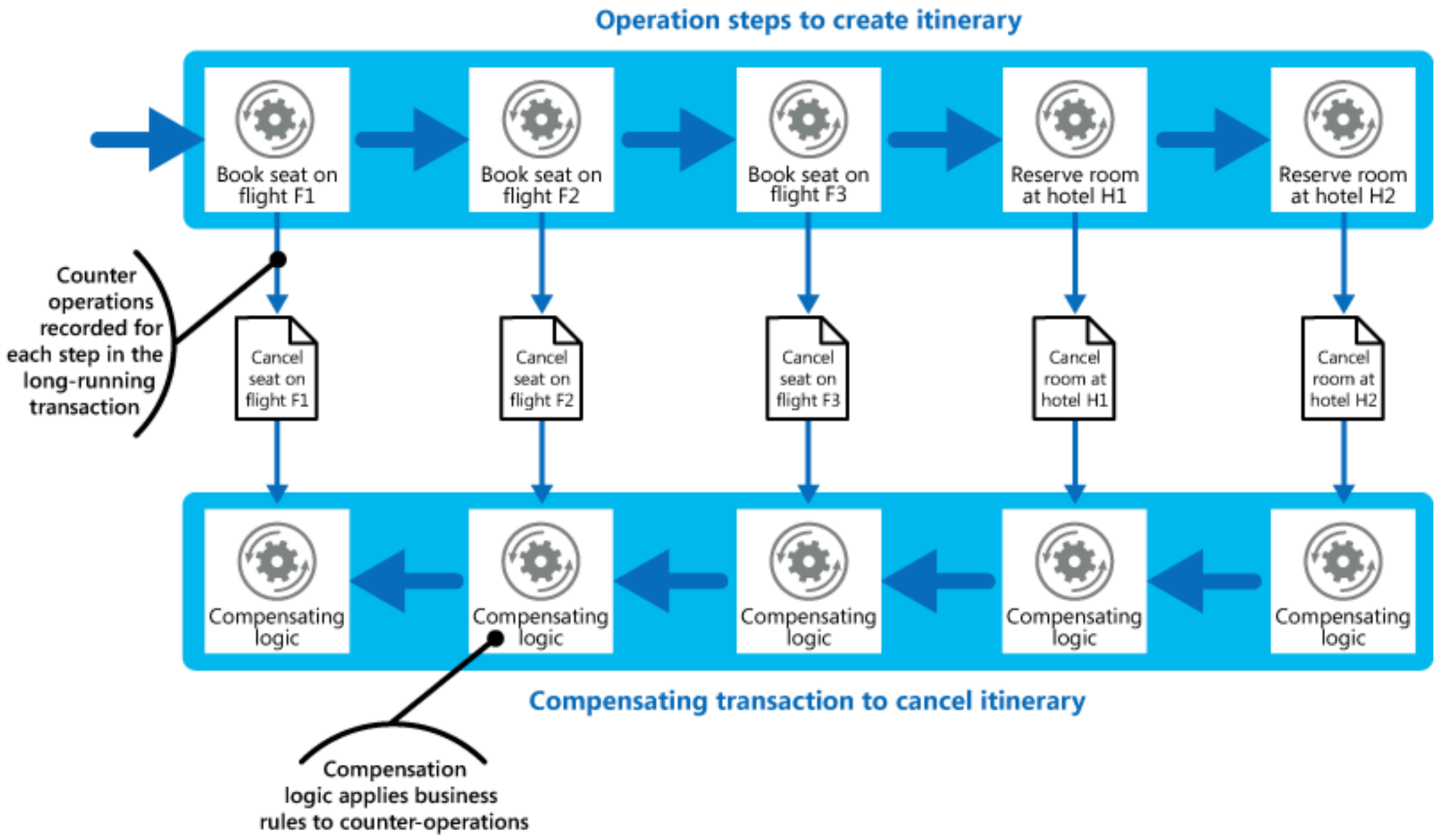
Fail Fast. This can improve the stability and resiliency of an application.

Bulkhead



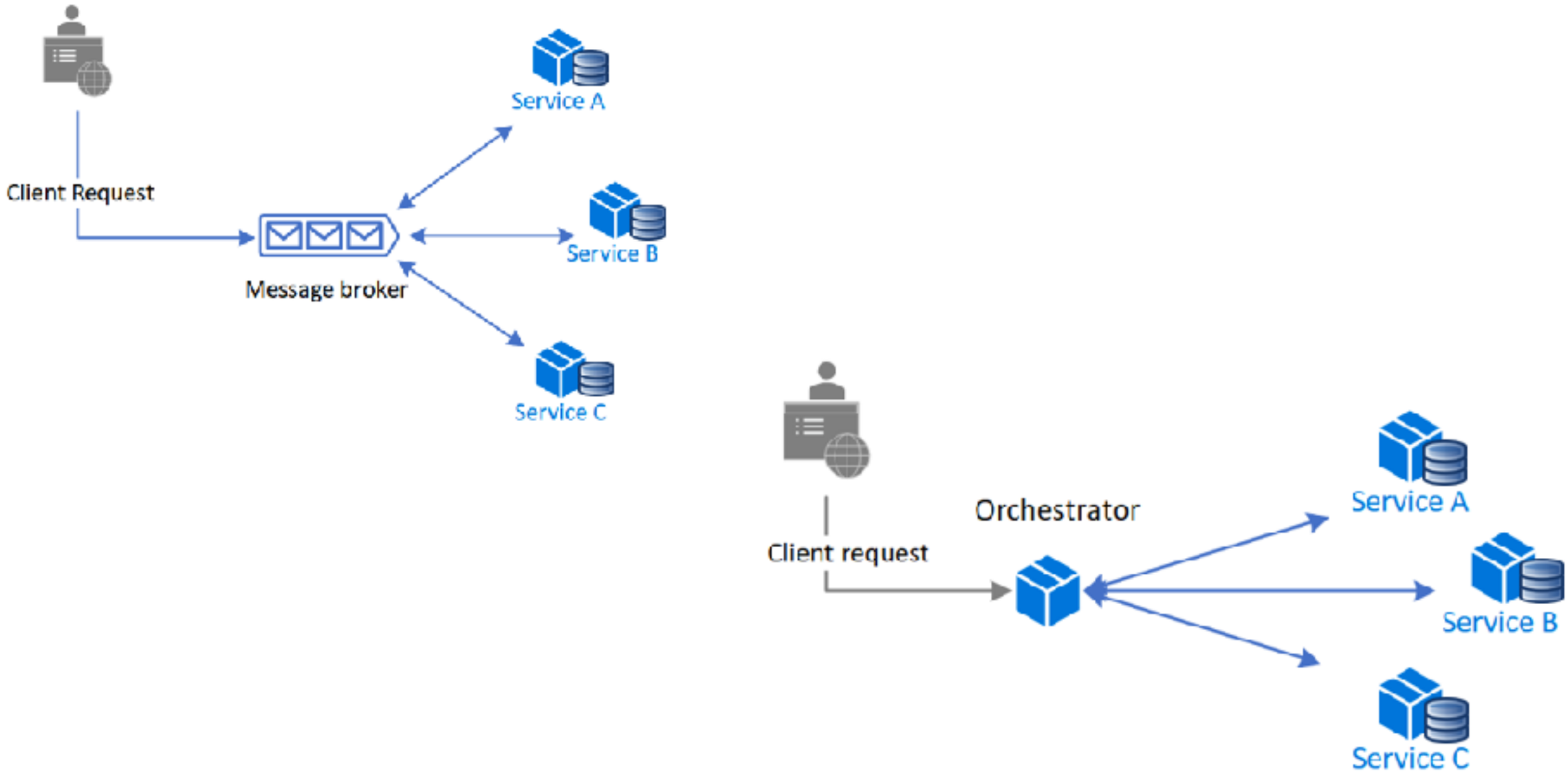
Each tenant is assigned a separate service instance. Tenant 1 has made too many requests and overwhelmed its instance.

Compensating Transaction



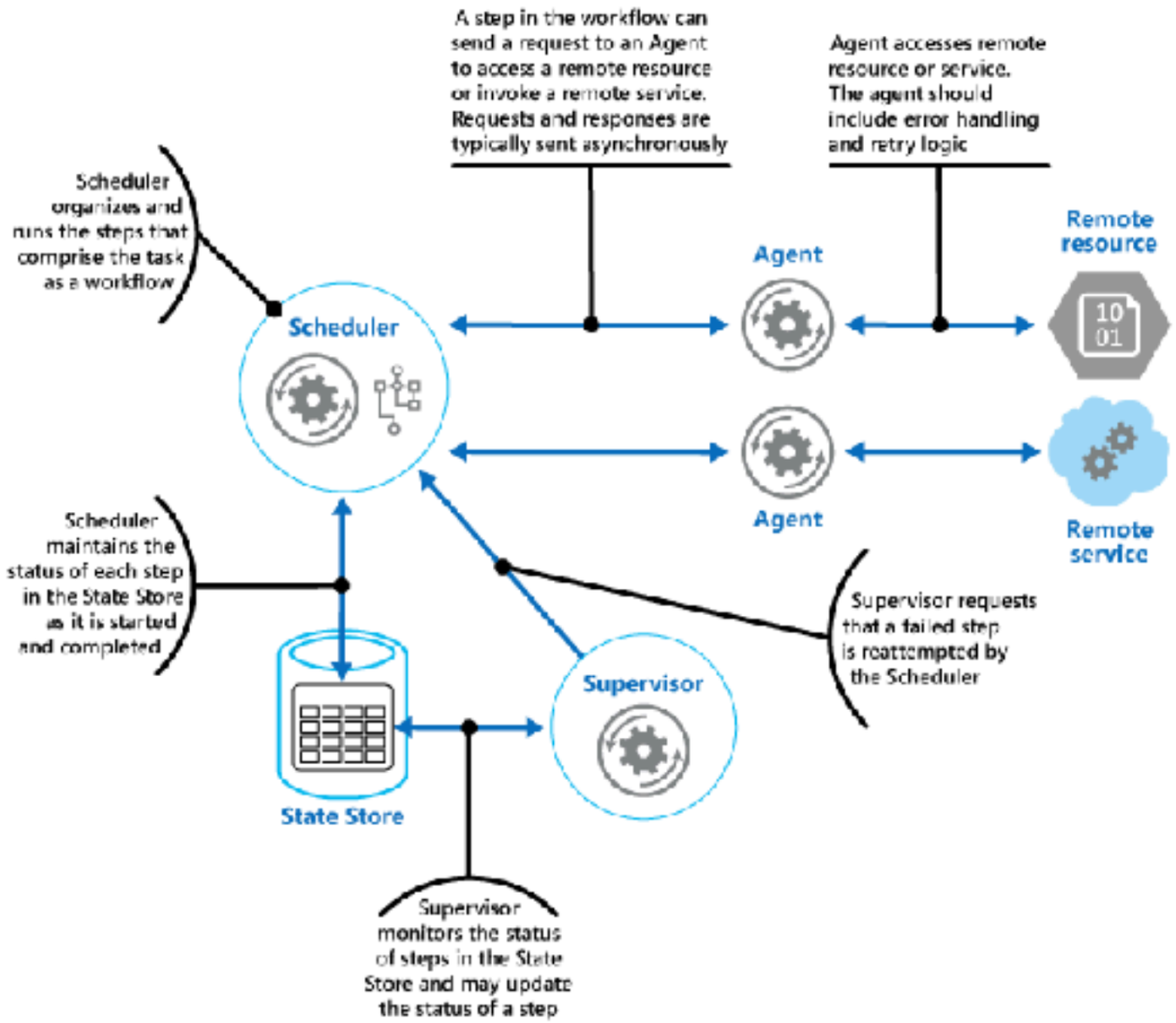
Undo the work performed by a series of steps, which together define an eventually consistent operation, if one or more of the steps fail.

Saga distributed transactions



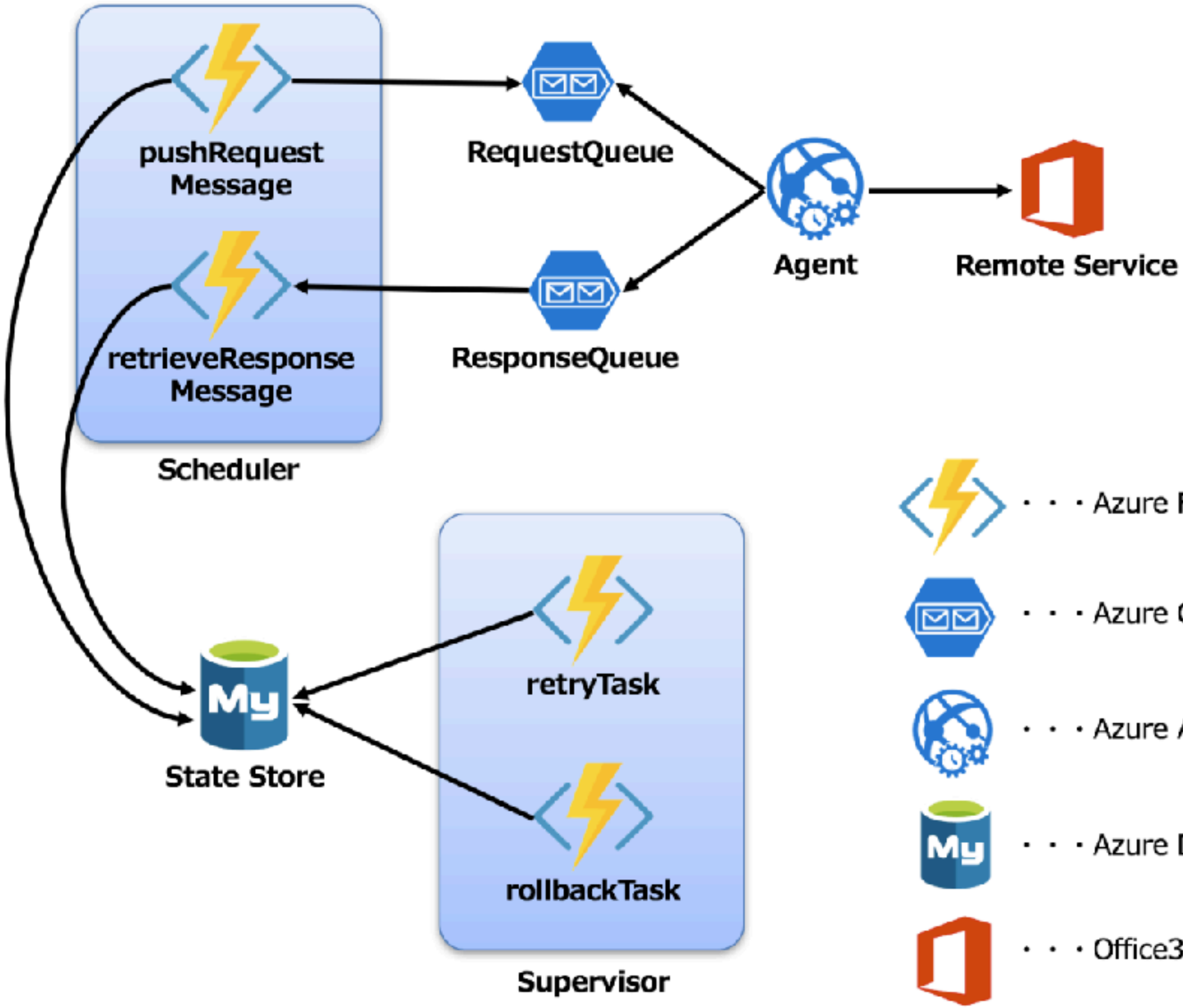
The Saga pattern provides transaction management using a sequence of *local transactions*. A local transaction is the atomic work effort performed by a saga participant. Each local transaction updates the database and publishes a message or event to trigger the next local transaction in the saga. If a local transaction fails, the saga executes a series of *compensating transactions* that undo the changes that were made by the preceding local transactions.

Scheduler Agent Supervisor



Access to remote services is subject to network congestion, overload on the remote service side, occurrence of throttling, and suspension of data center operations due to disasters such as earthquakes and fires. It fails for various reasons. **The Retry pattern** can be used to recover in a short period of time, but if a persistent error that cannot be easily recovered occurs, one of the following measures is required.

- Retry processing to the remote service.
- Rollback any changes to remote services to keep the system in a consistent state.
- Notify the administrator by email, and the administrator manually recovers.



・・・Azure Functions

・・・Azure Queue Storage

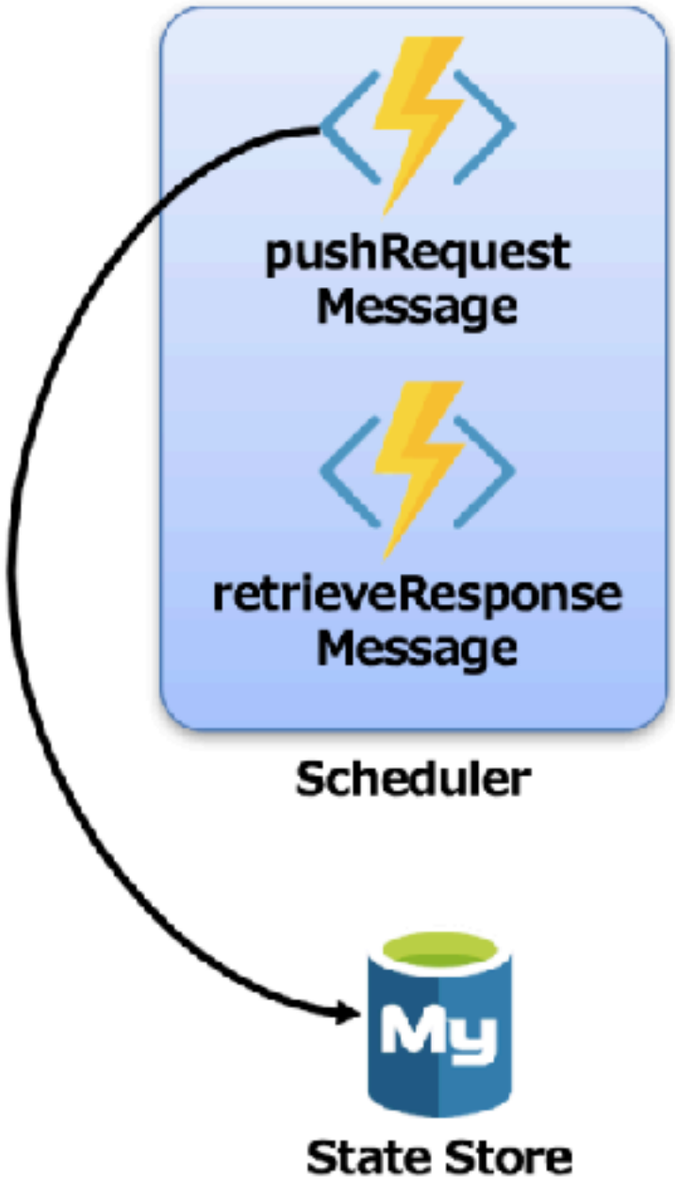
・・・Azure App Service WebJobs

・・・Azure Database for MySQL

・・・Office365

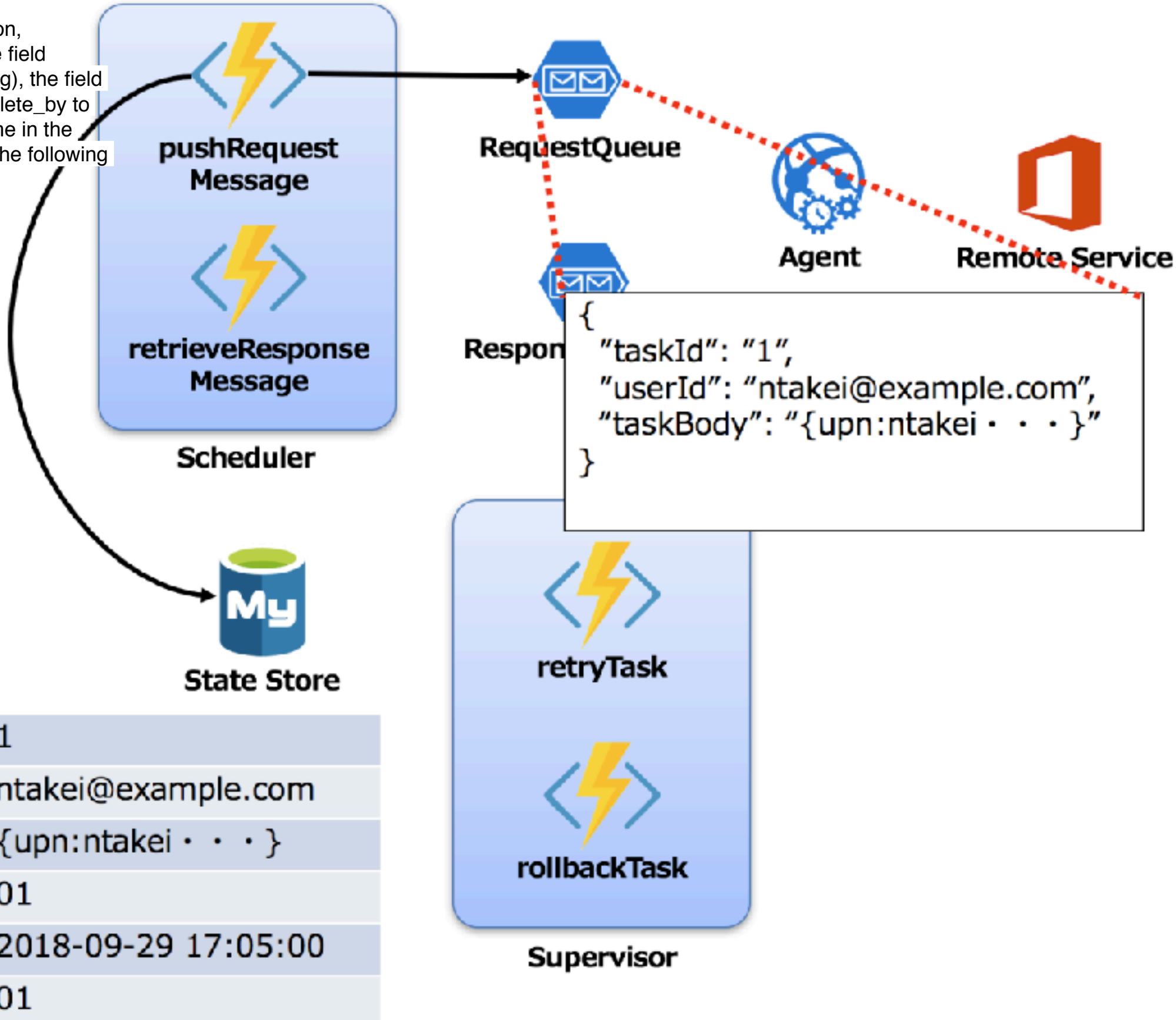
Steps

The Azure Function pushRequestMessage, which acts as a Scheduler, retrieves a task with field process_state of 00 (unprocessed) and locked_by of NULL in table t_state_store.

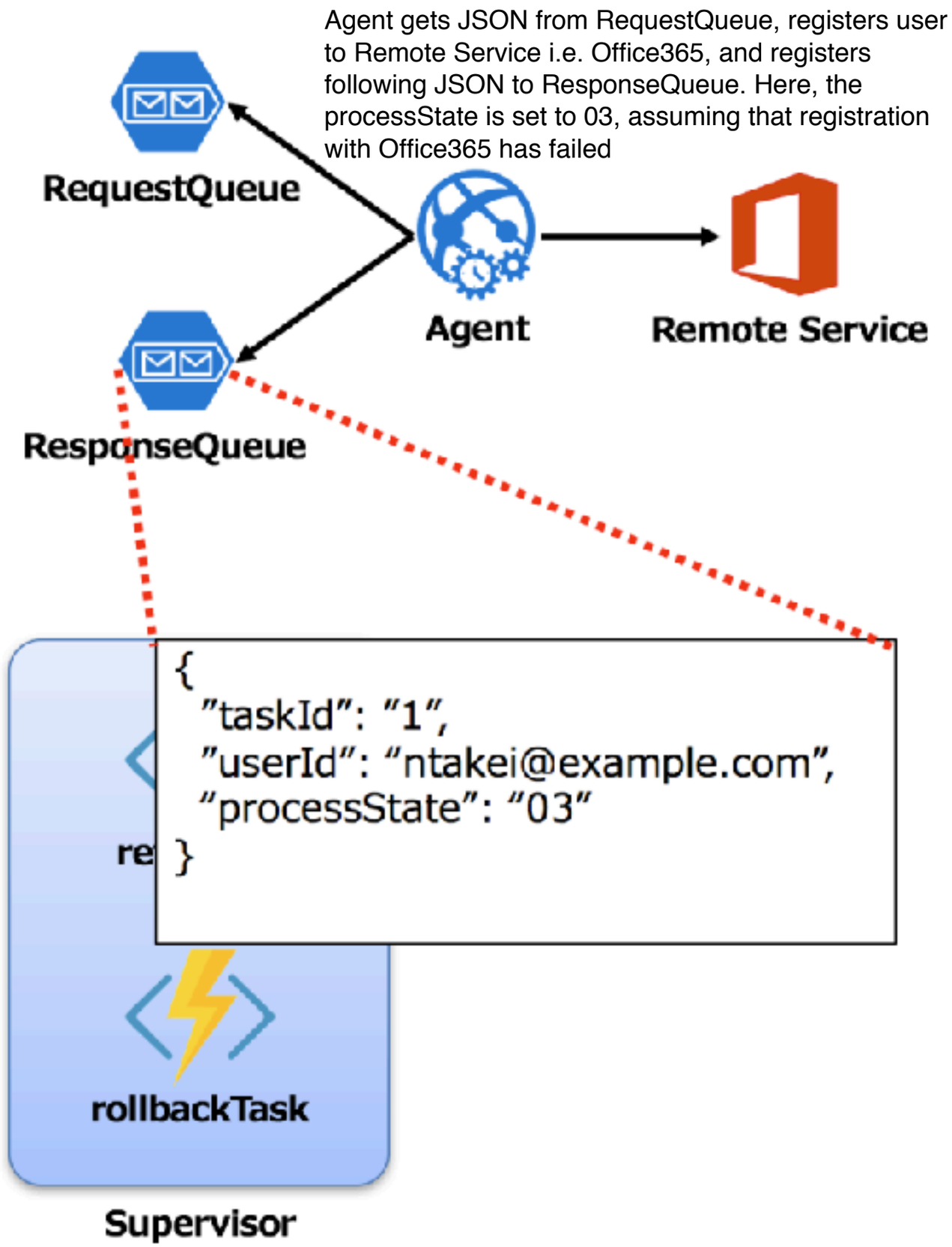


task_id	1
user_id	ntakei@example.com
task_body	{upn:ntakei・・・}
locked_by	NULL
complete_by	NULL
process_state	00
failuer_count	0

After getting the task information, pushRequestMessage sets the field process_state to 01 (processing), the field locked_by to 01, and the complete_by to 10 minutes after the current time in the table t_state_store. And push the following JSON to RequestQueue.

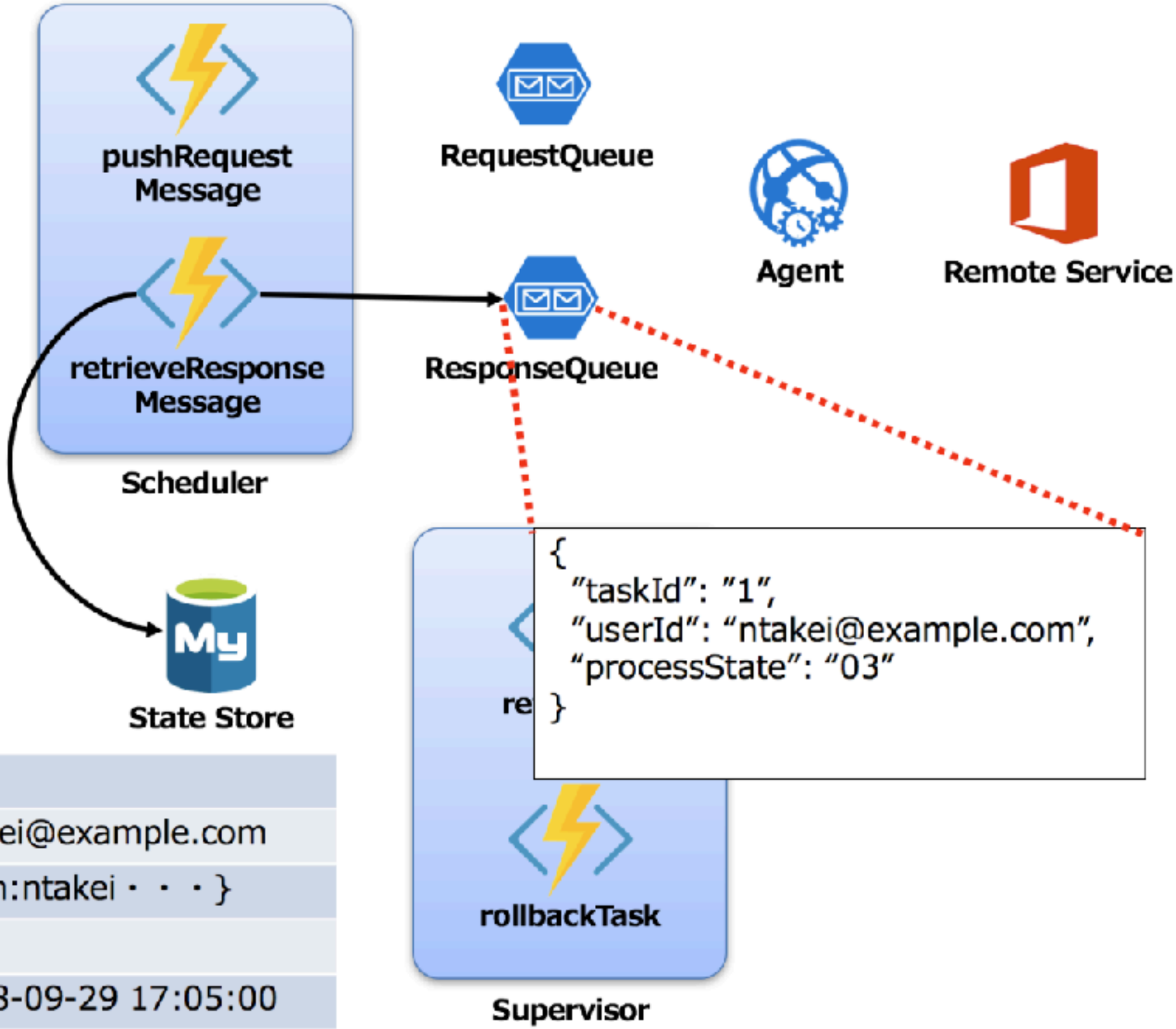


task_id	1
user_id	ntakei@example.com
task_body	{upn:ntakei・・・}
locked_by	01
complete_by	2018-09-29 17:05:00
process_state	01
failuer_count	0

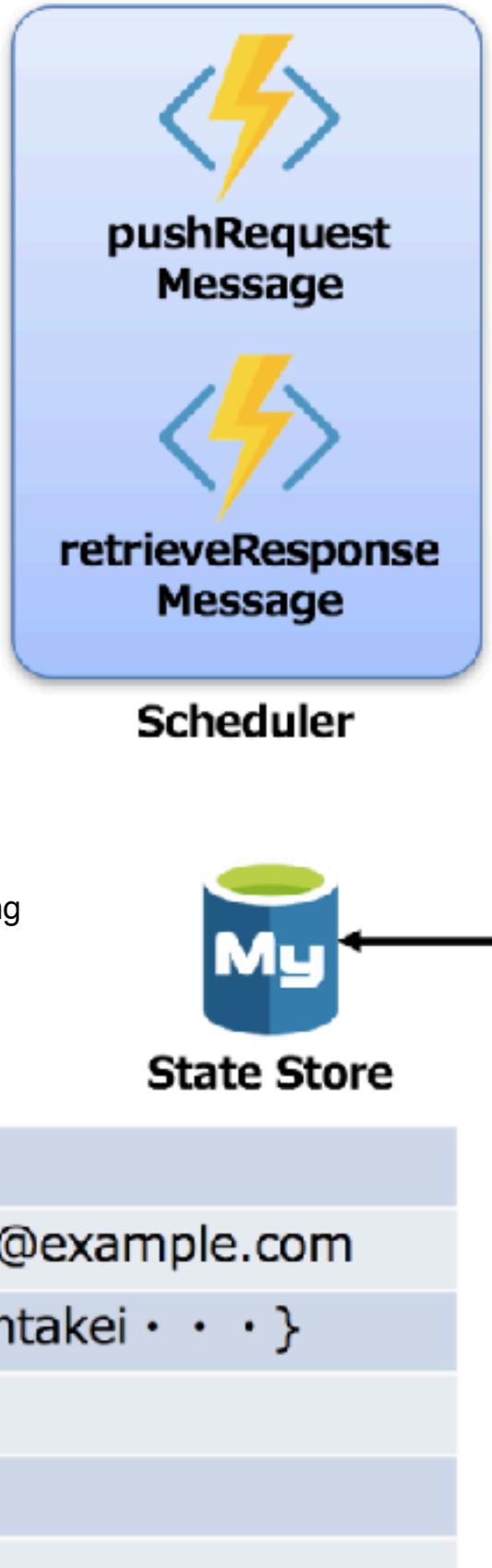


task_id	1
user_id	ntakei@example.com
task_body	{upn:ntakei・・・}
locked_by	01
complete_by	2018-09-29 17:05:00
process_state	01
failuer_count	0

retrieveResponseMessage checks the State Store every minute, retrieves the message from ResponseQueue and registers the value of processState in JSON as 03 (failure) in field process_state of table t_state_store.

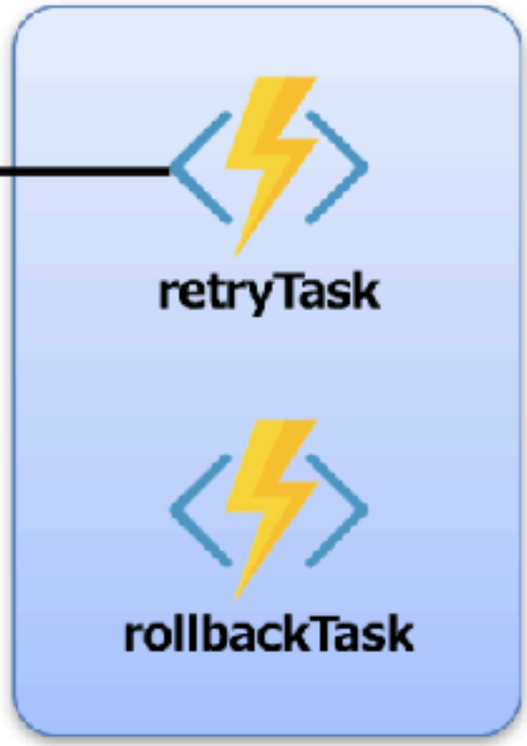


task_id	1
user_id	ntakei@example.com
task_body	{upn:ntakei・・・}
locked_by	01
complete_by	2018-09-29 17:05:00
process_state	03
failuer_count	0

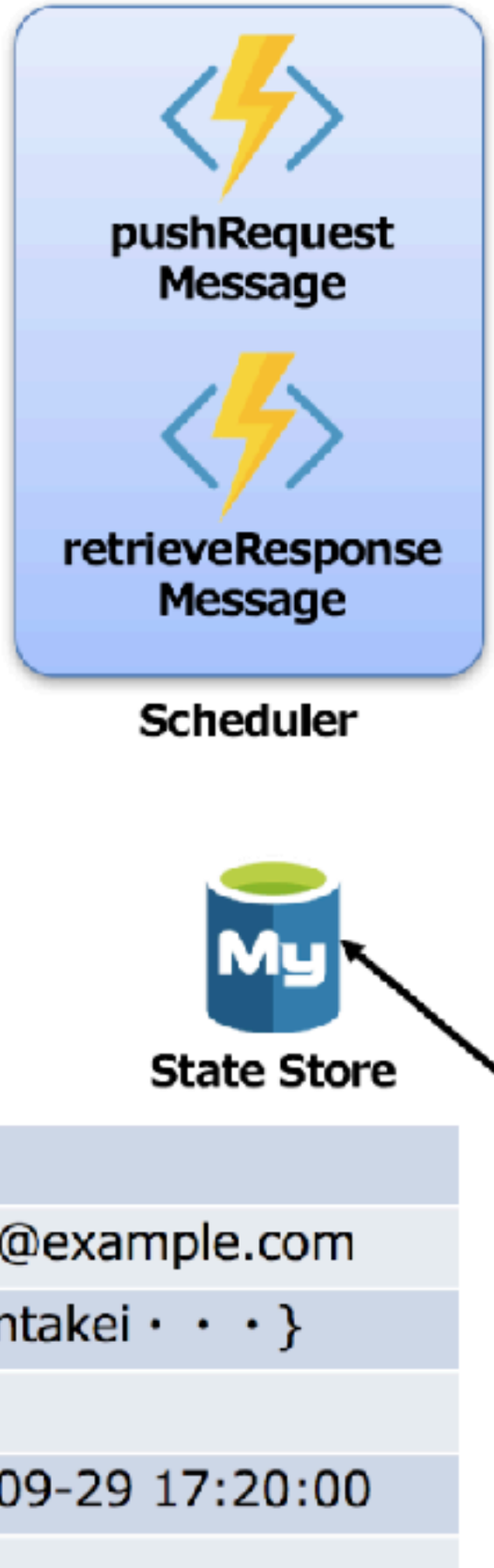


retryTask checks the State Store every minute and updates tasks with a process_state of 03 (failed) as follows: Then, the scheduler extracts the following records and requests processing to the agent again. In other words, it is a retry.

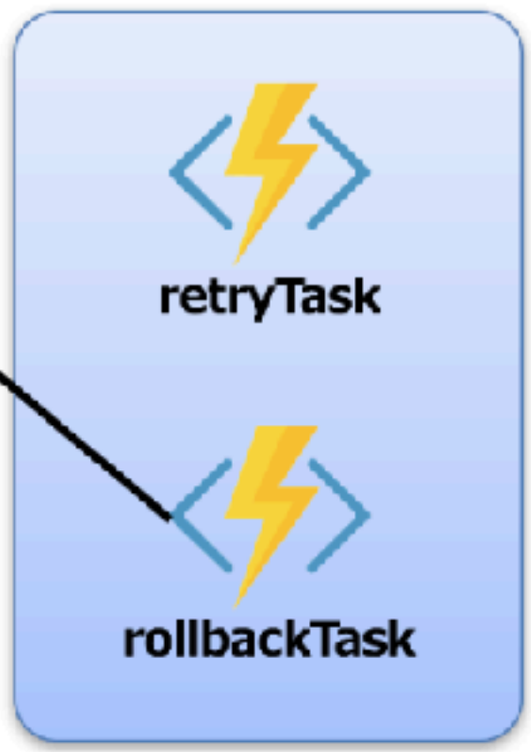
State Store



task_id	1
user_id	ntakei@example.com
task_body	{upn:ntakei・・・}
locked_by	NULL
complete_by	NULL
process_state	01
failuer_count	1



Then, let's say that we repeat the retry and finally fail three times. Supervisor's rollbackTask function determines that this is an unusual failure in Office365, gives up retry, rolls back, or notifies the administrator by e-mail.



task_id	1
user_id	ntakei@example.com
task_body	{upn:ntakei・・・}
locked_by	01
complete_by	2018-09-29 17:20:00
process_state	03
failuer_count	3

(1) First, register the contents (user name, password, etc.) of the process to be registered with the Remote Service (a service on the cloud such as Office365) in the State Store. State store implementation can be file, relational database, NoSQL database, or anything as long as data can be persisted. In addition, the following information is registered.

- Execution state of the task (open, in progress, successful, failed)
- Estimated completion time of the task
- Task failure count

(2) The Scheduler periodically checks the State Store, extracts tasks whose execution status is "Unprocessed", and requests the Agent to execute the task. At this time, change the State Store as follows.

- Task execution status: Processing
- Estimated task completion time: current time + task processing timeout time

(3) Upon receiving a request from Scheduler, Agent accesses Remote Service and executes processing.

(4) Agent returns the result of processing for Remote Service to Scheduler.

(5) Scheduler receives execution results from Agent and changes State Store as follows.

- Task execution status: success or failure

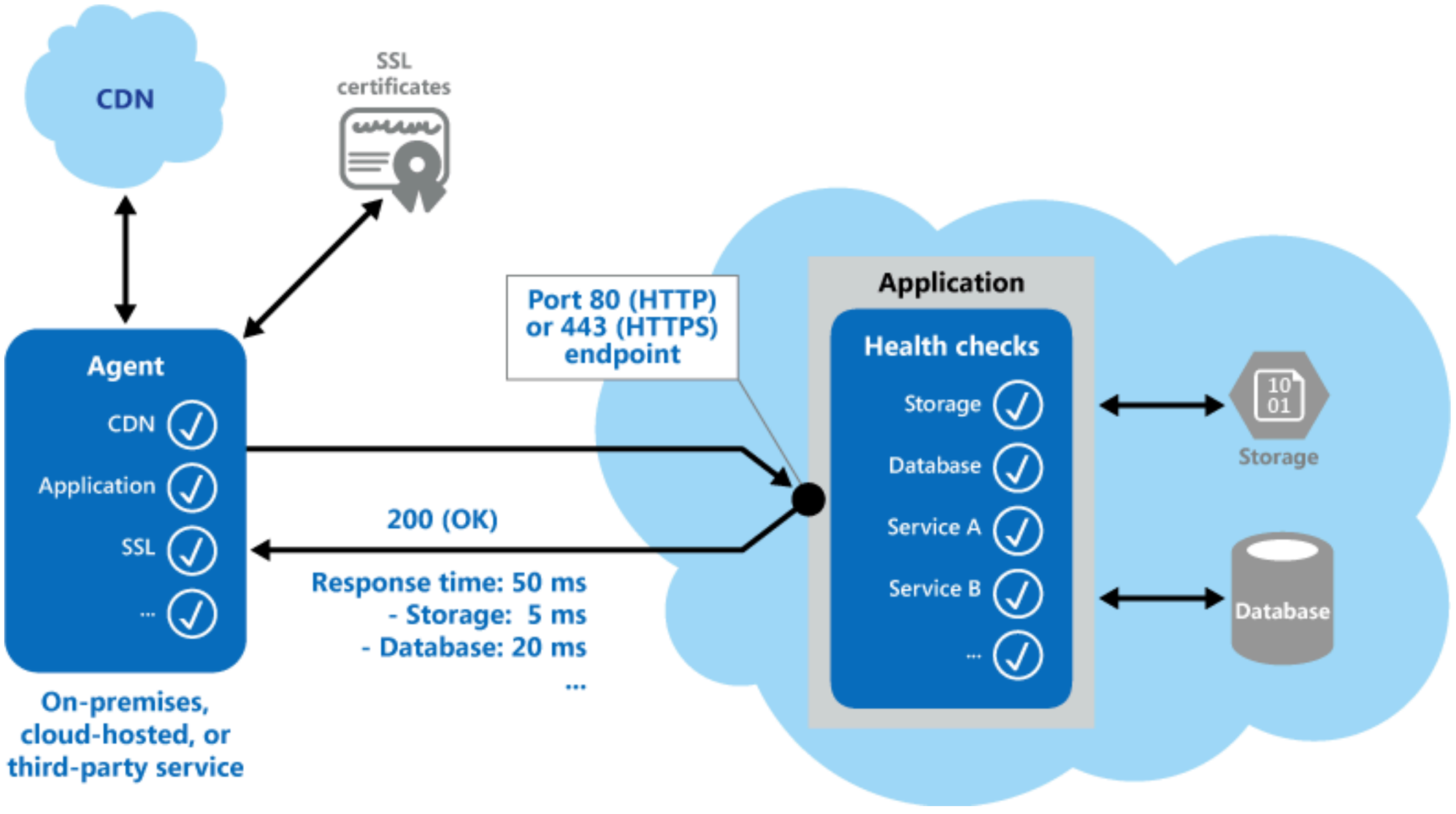
(6) Supervisor periodically checks the State Store and performs the following processing.

- If the number of task failures is within a specified number of times and the task execution status is failed, or if the scheduled completion time of the task has passed, the task execution status is changed to unprocessed. (In other words, let the Agent process again)
- If the number of task failures exceeds the specified number of times, recovery by retrying is impossible, notify by email, and the administrator manually corrects, rolls back each service, etc. Appropriate processing is performed according to the requirement

The State Store has a table called t_state_store with the following fields:

task_id	ID that uniquely identifies the task (Auto Increment)
user_id	ID of the user for whom the task is to be executed
task_body	Information necessary for remote service processing (UPN of Office365, etc.)
locked_by	Instance ID of the scheduler that performs the process (It is assumed that there are multiple schedulers, but in this blog there is only one, so 01 is registered as a
complete_by	Estimated time the task will be completed
process_state	Task execution status (00: Not processed, 01: Processing, 02: Succeeded, 03: Failed)
failure_count	Number of times the Agent failed to process a task

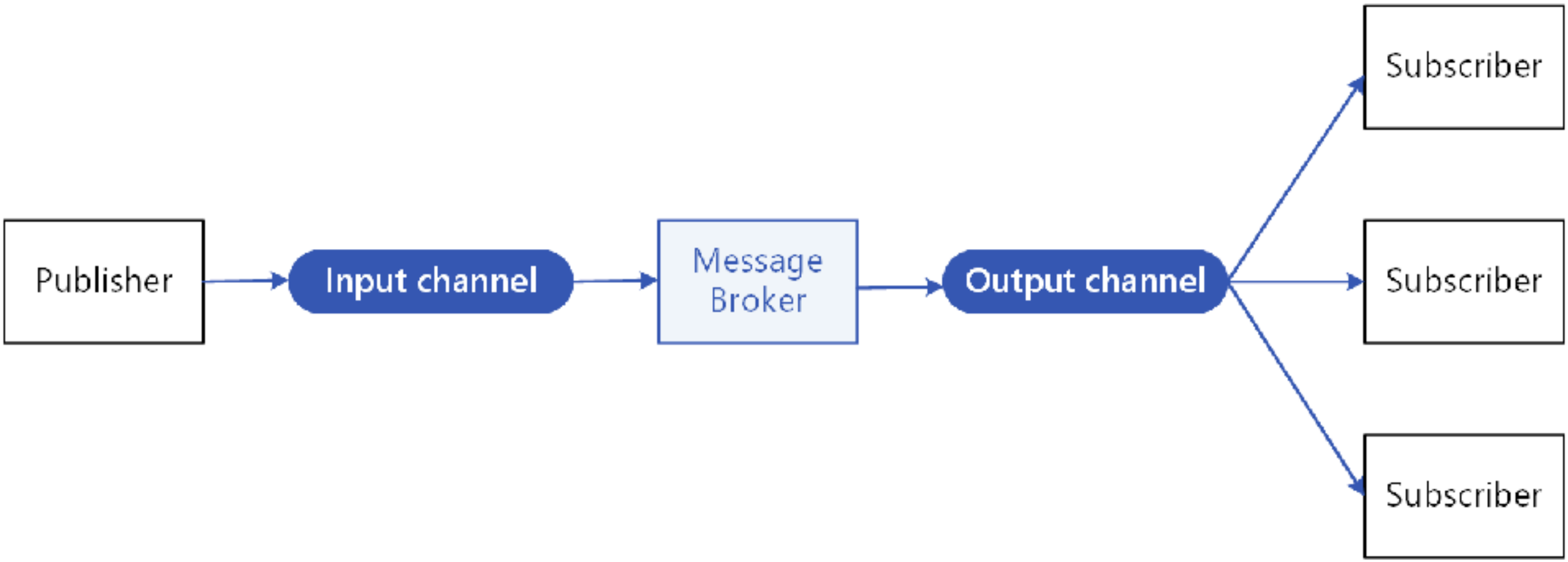
Health Endpoint Monitoring



Implement functional checks in an application that external tools can access through exposed endpoints at regular intervals.

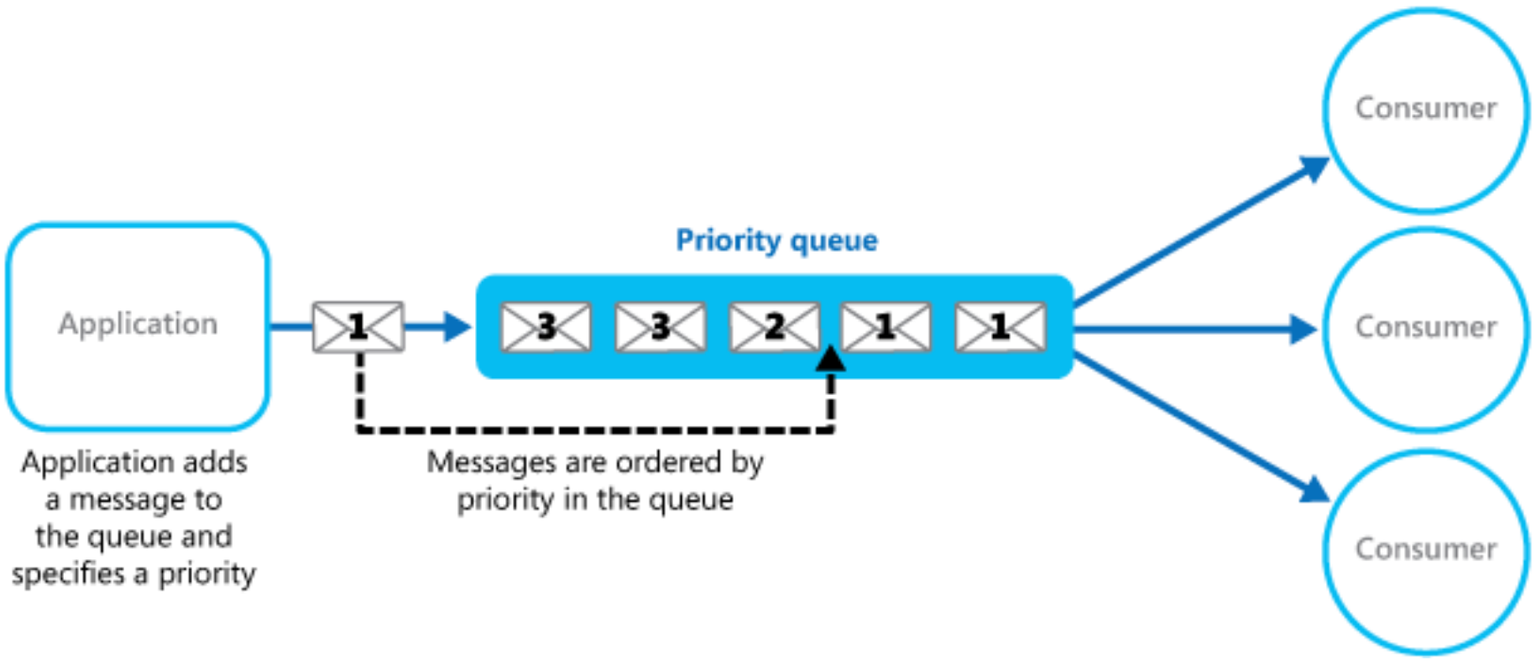
Communication

Publisher-Subscriber



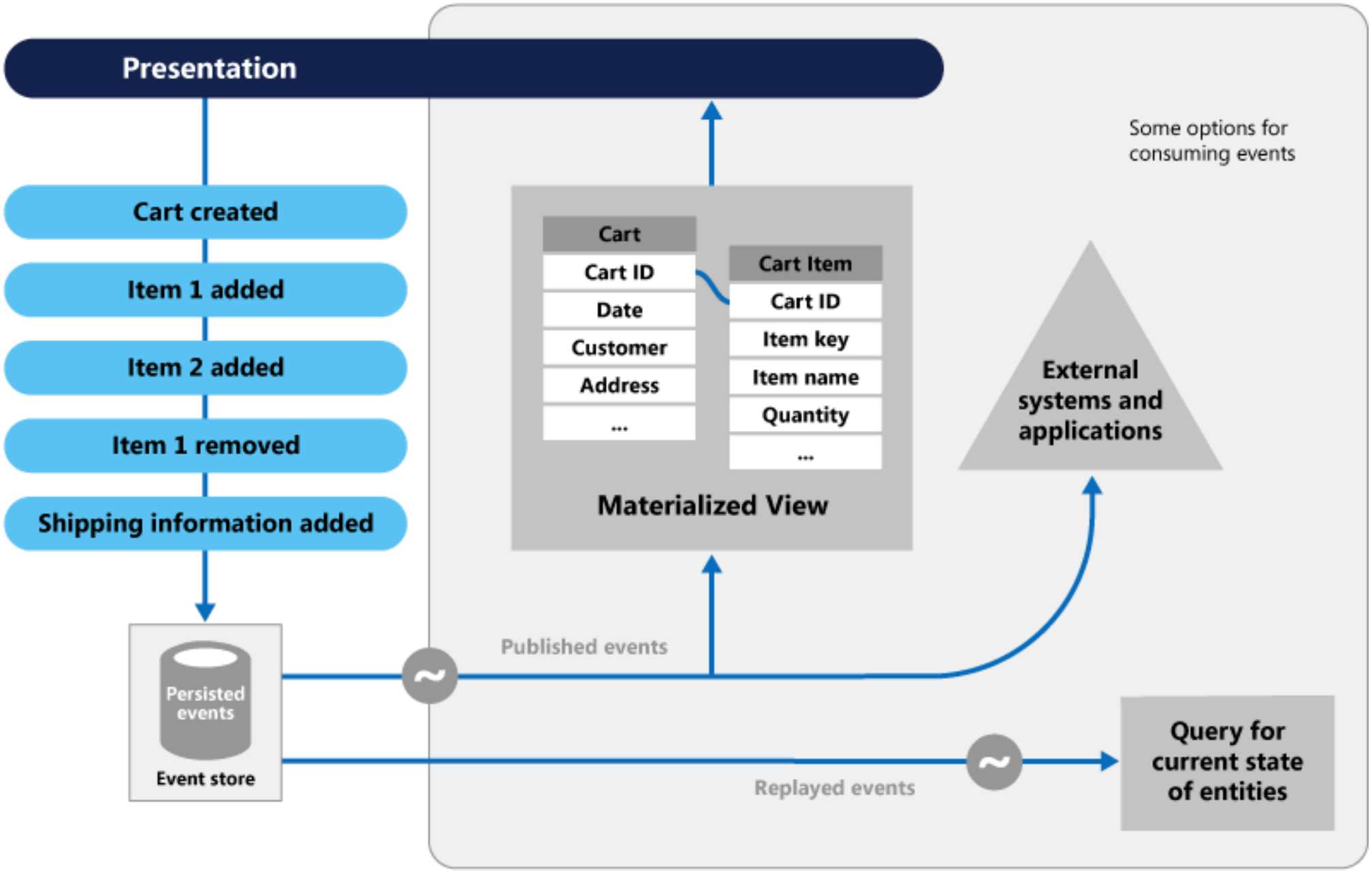
- An application needs to broadcast information to a significant number of consumers.

Priority Queue



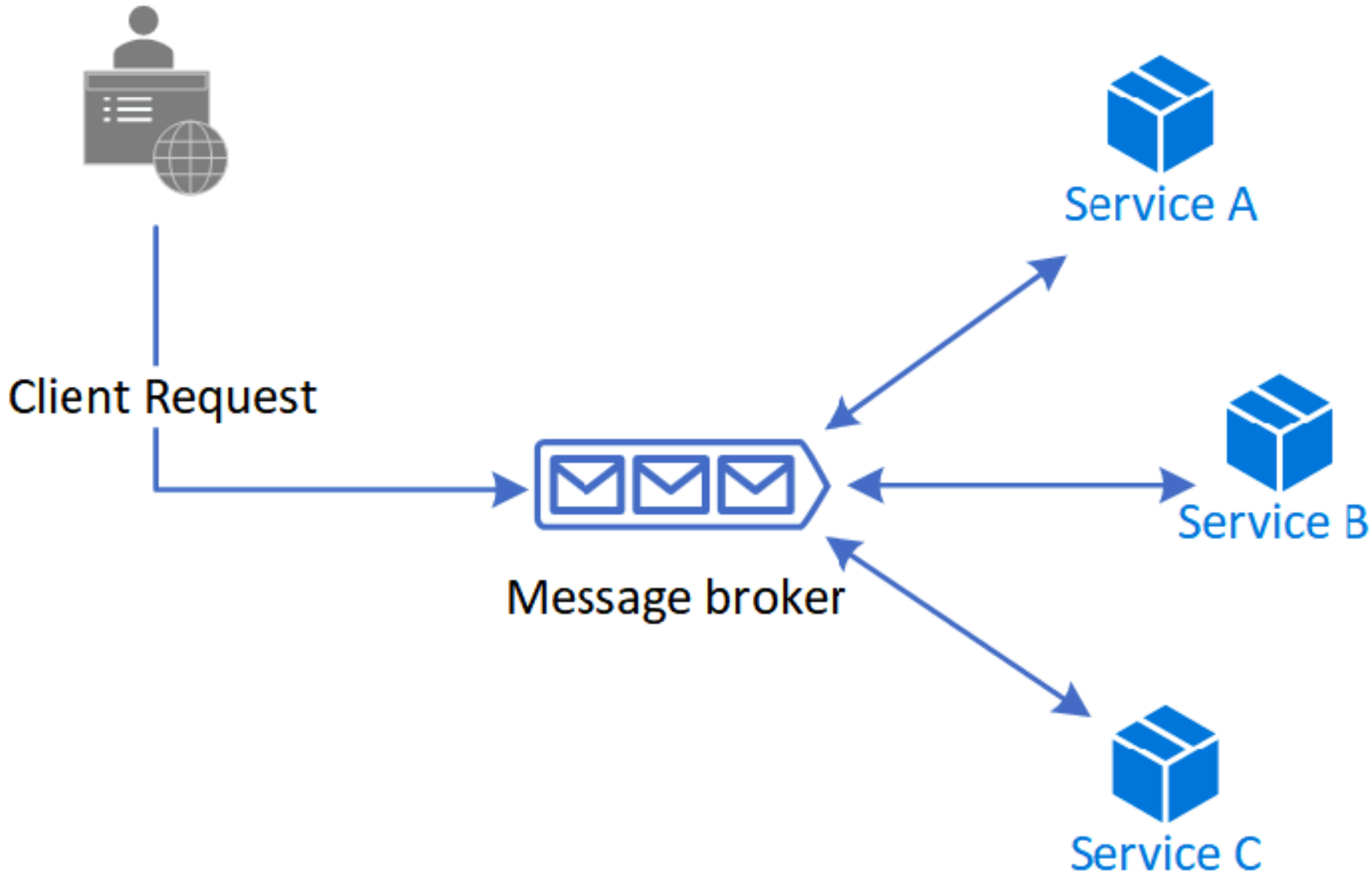
Prioritize requests sent to services so that requests with a higher priority are received and processed more quickly than those with a lower priority.

Event Sourcing



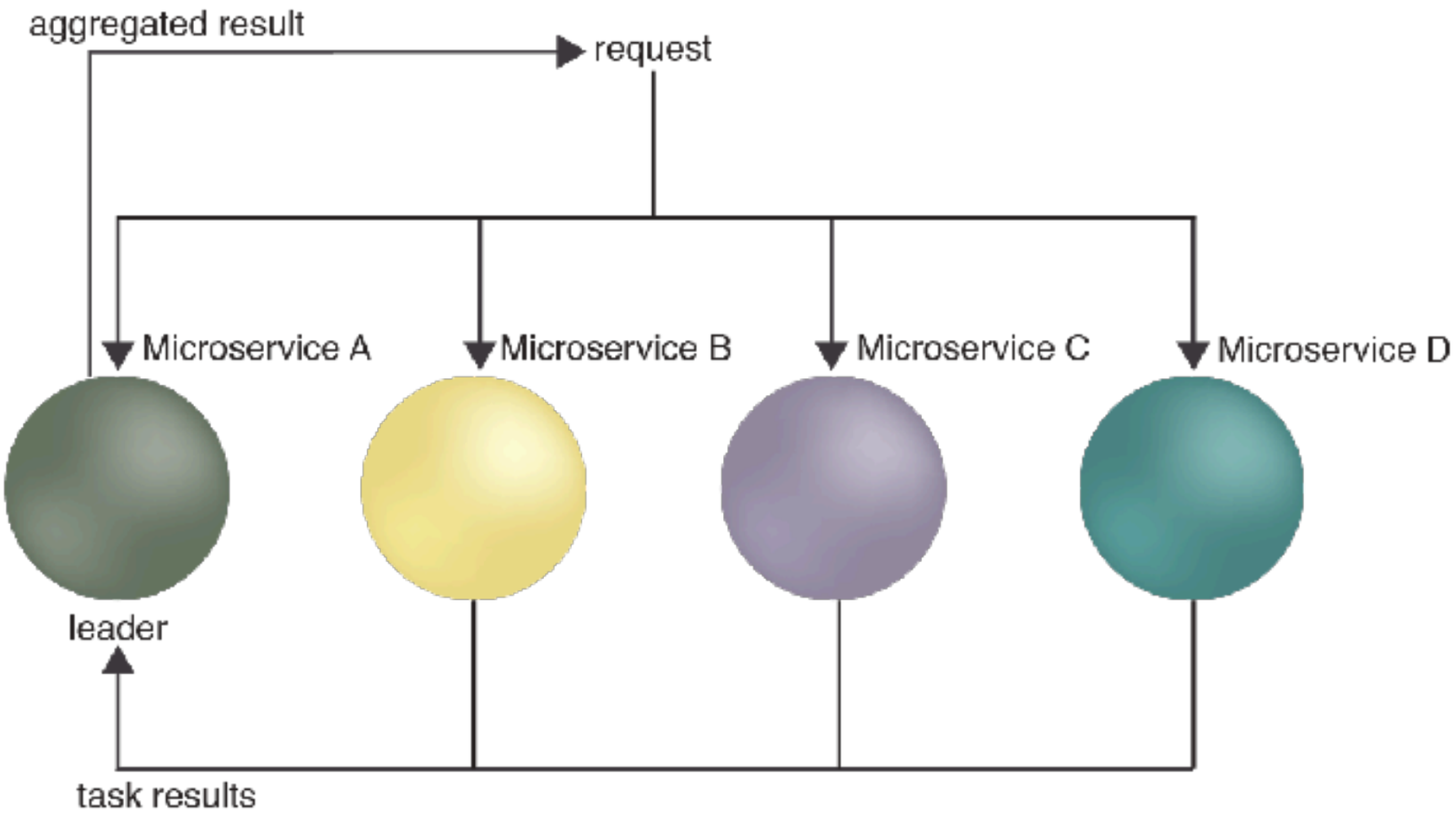
Instead of storing just the current state of the data in a domain, use an append-only store to record the full series of actions taken on that data.

Choreography



Have each component of the system participate in the decision-making process about the workflow of a business transaction, instead of relying on a central point of control.

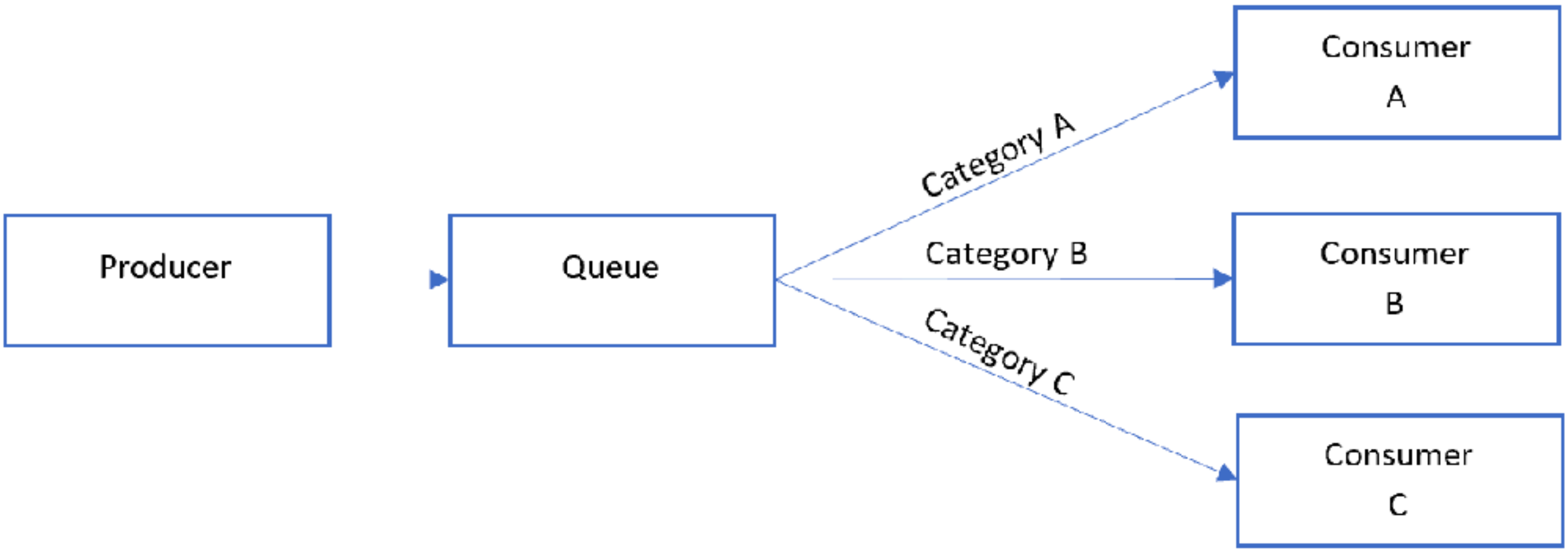
Leader Election



Coordinate the actions performed by a collection of collaborating instances in a distributed application by electing one instance as the leader that assumes responsibility for managing the others. , leader election can introduce new failure modes and scaling bottlenecks. Because of these complications, we carefully consider other options before implementing leader election.

Sequential Convoy

 # packetid
 # data structure (cache)



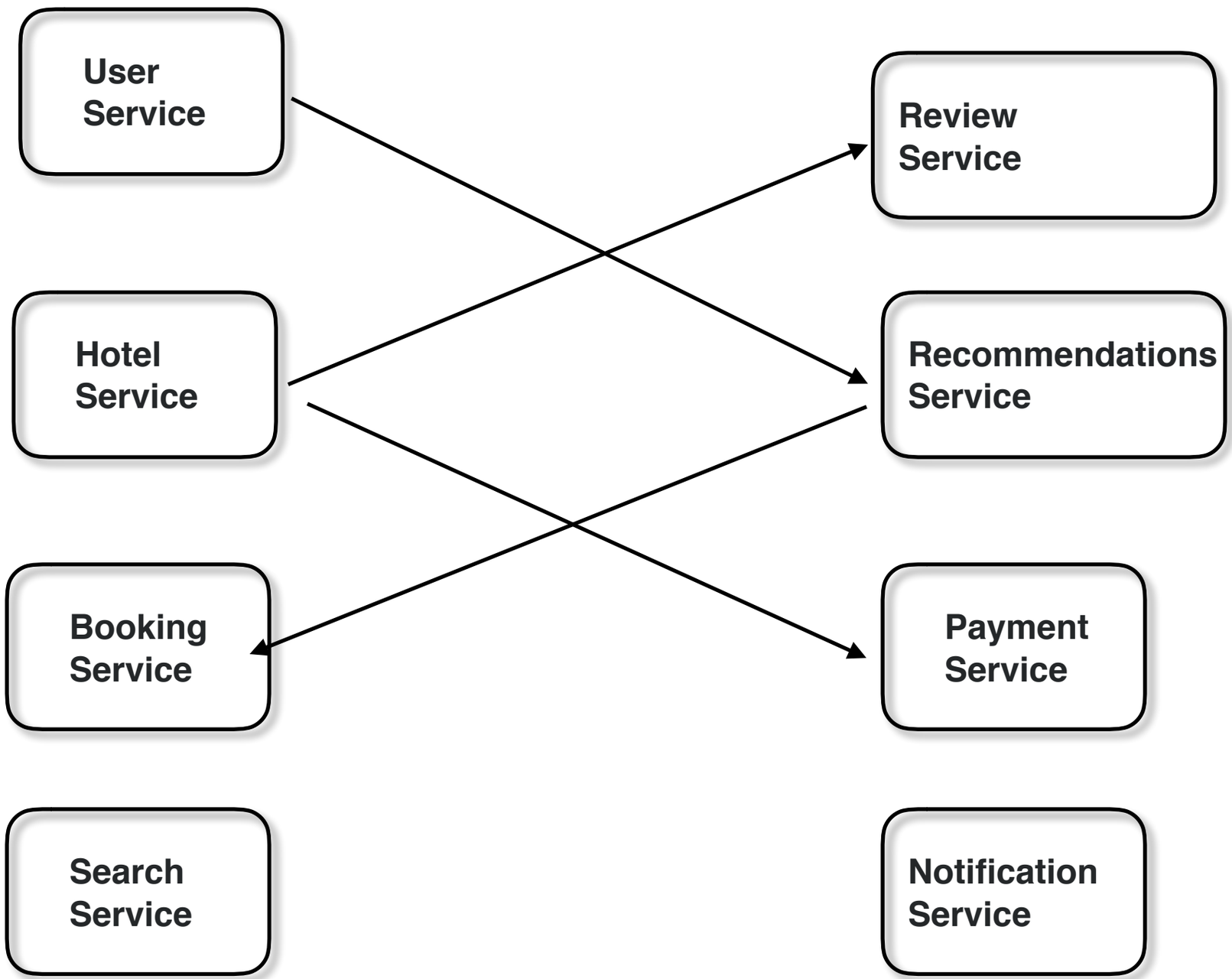
- You have messages that arrive in order and must be processed in the same order.

Push related messages into categories within the queuing system, and have the queue listeners lock and pull only from one category, one message at a time.

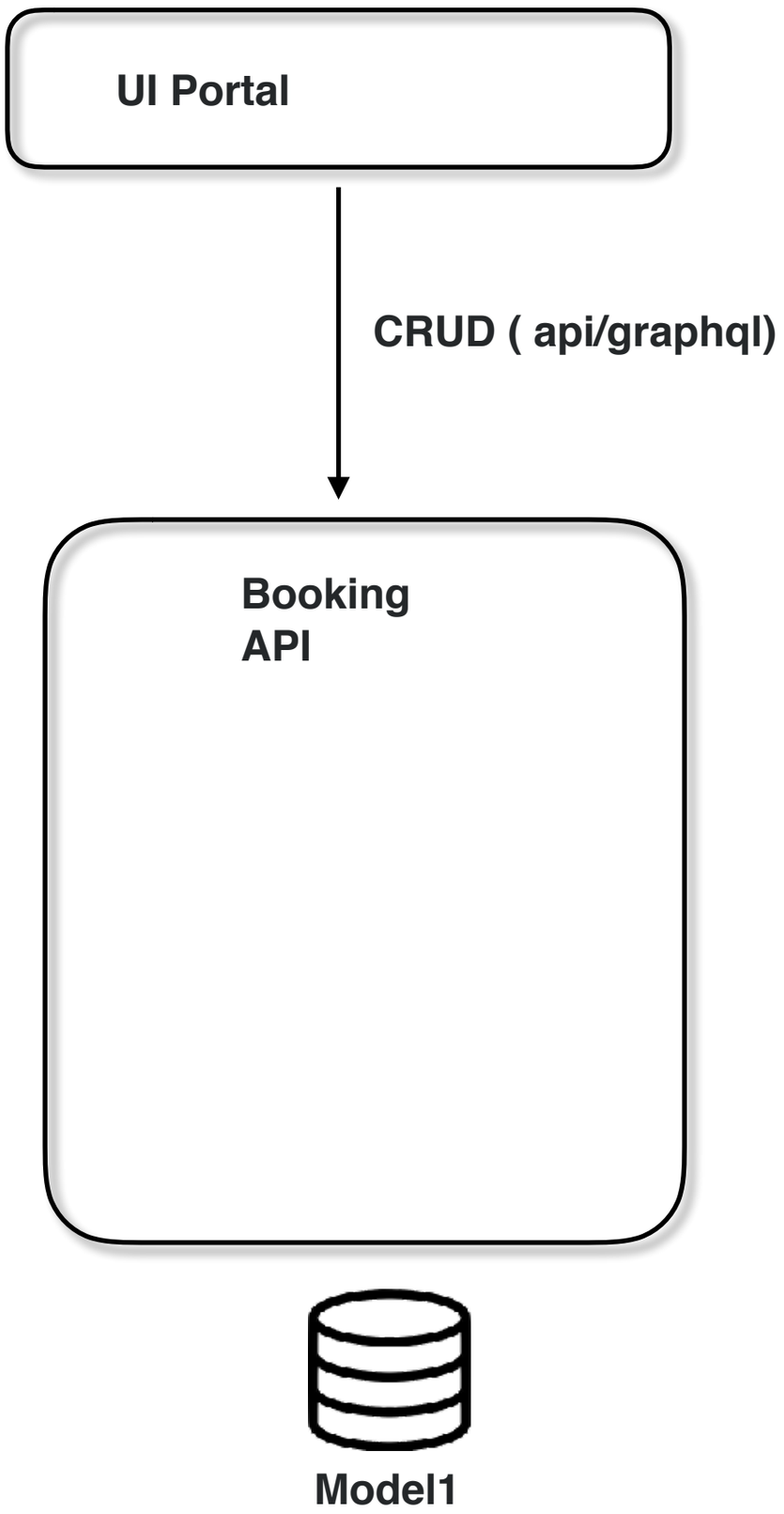
Case Study

Airbnb

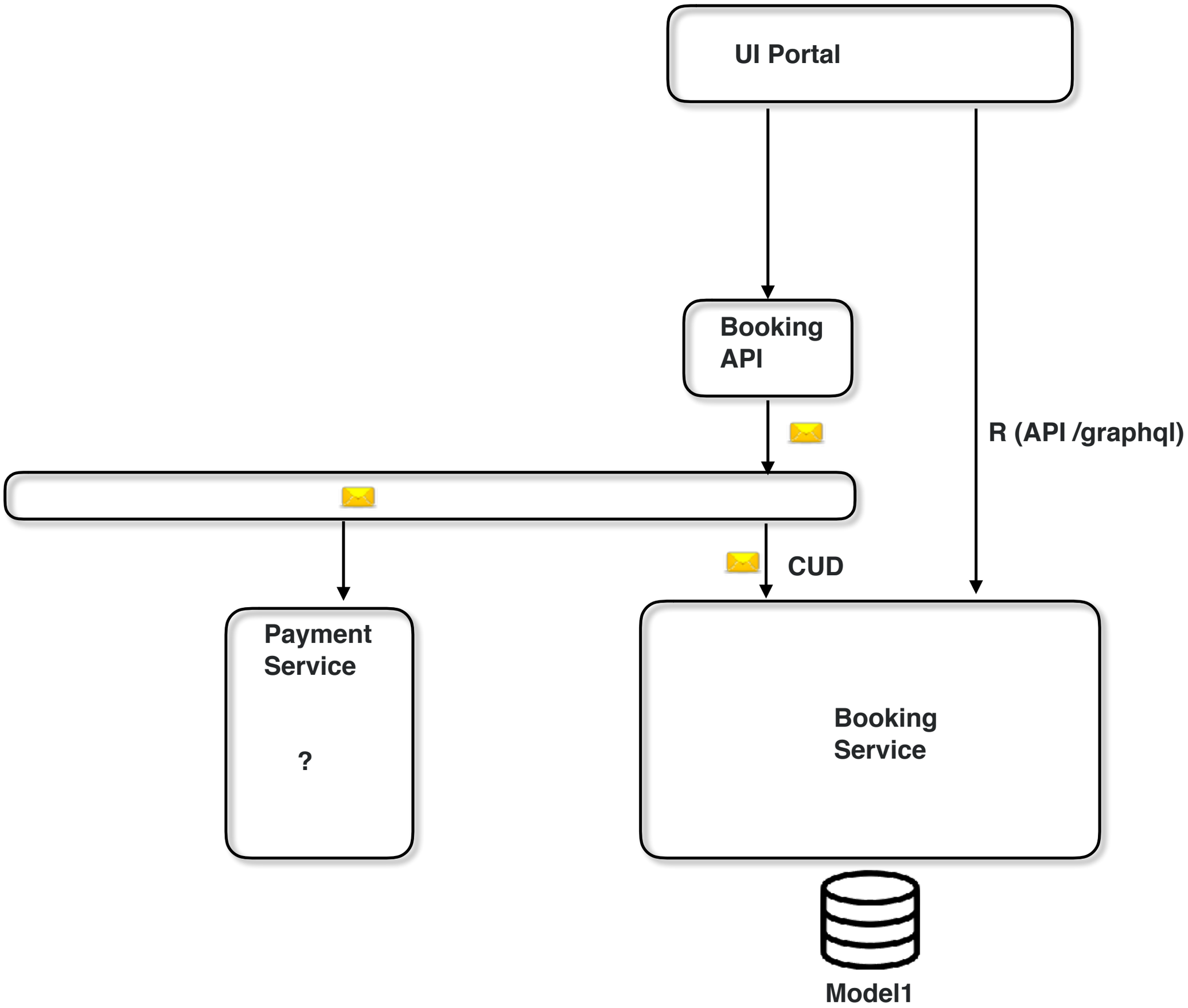
API
Message
Black board



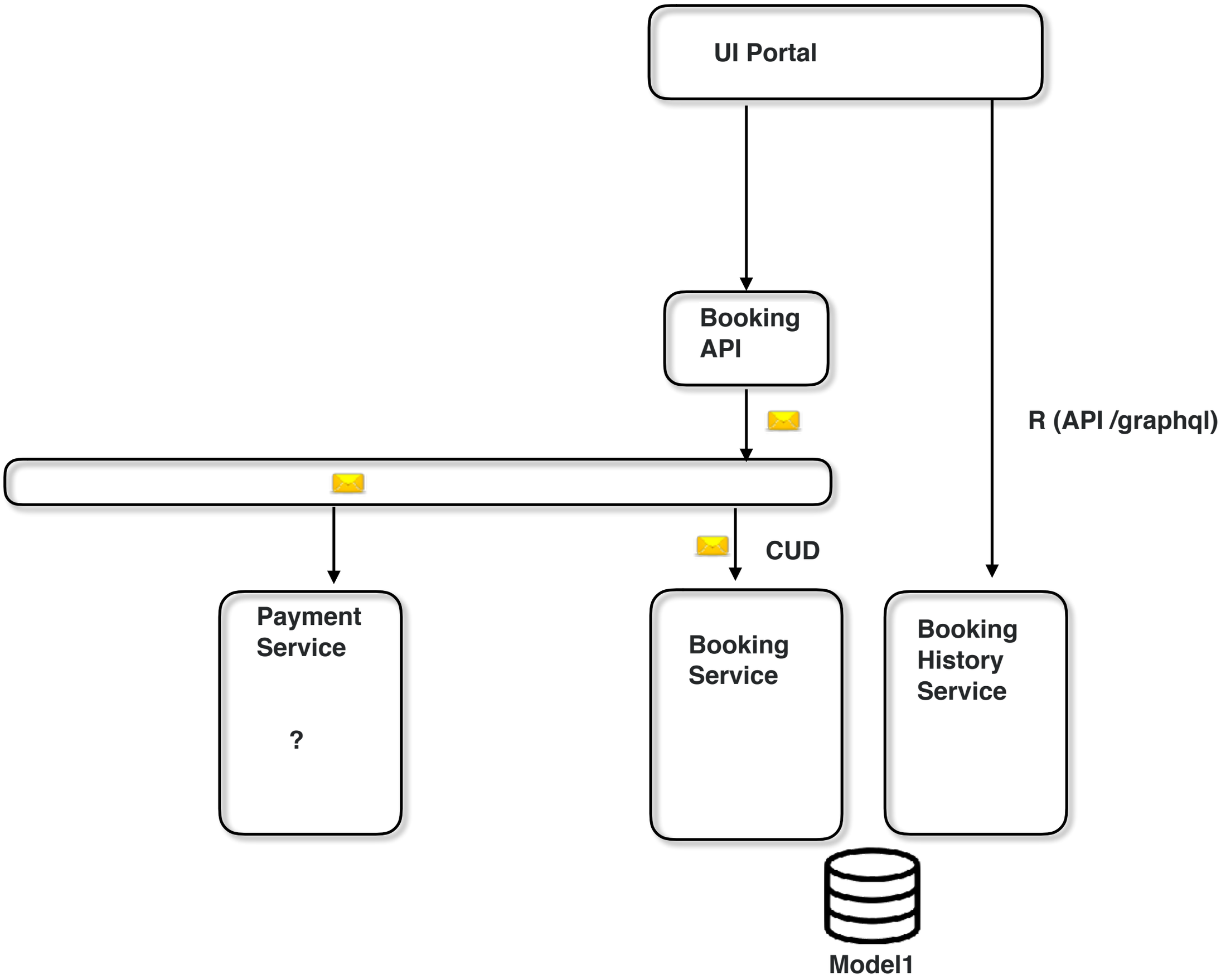
0



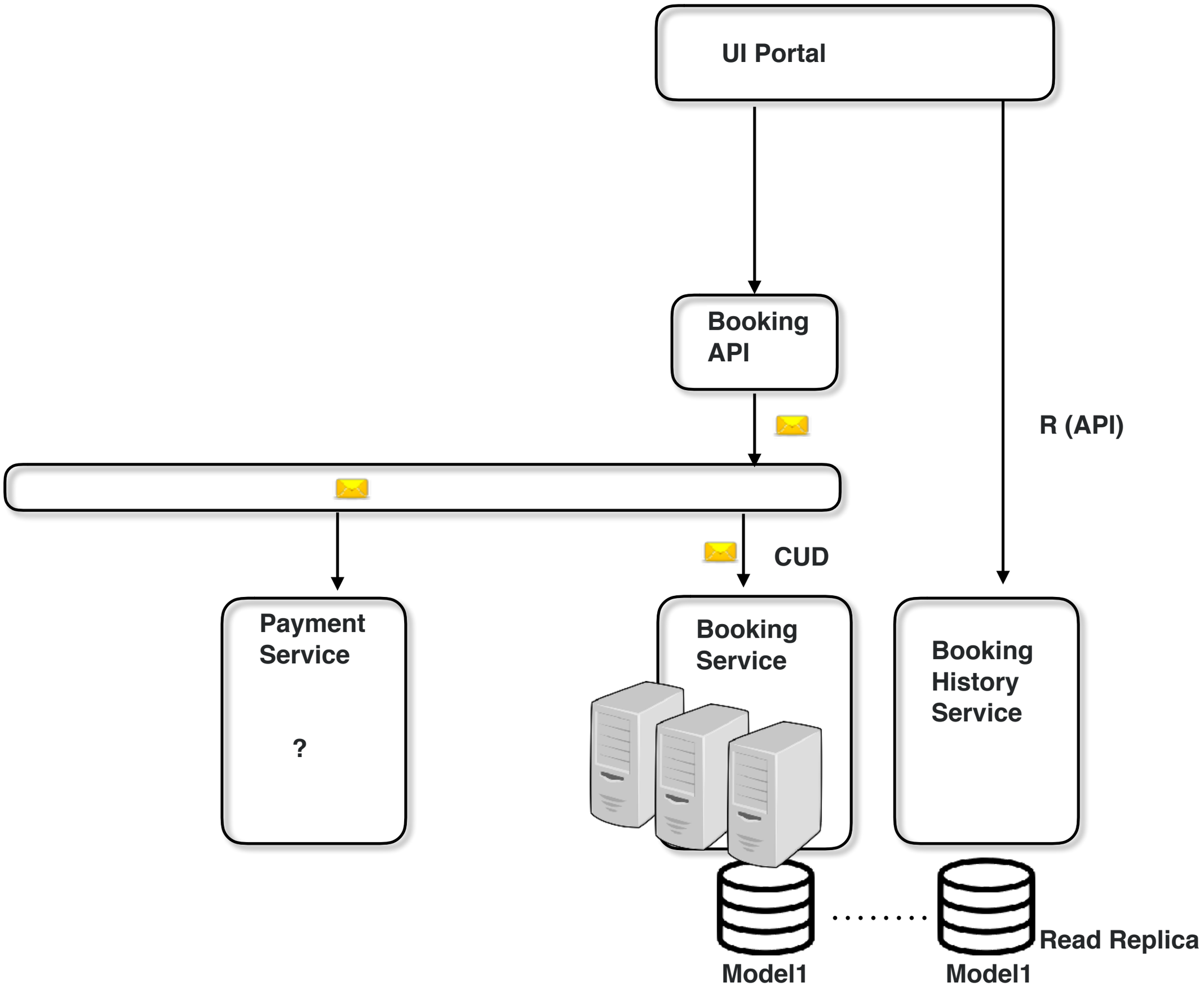
1


UI Portal
Booking API
R (API /graphql)
CUD
Payment Service
?
Booking Service
Model1

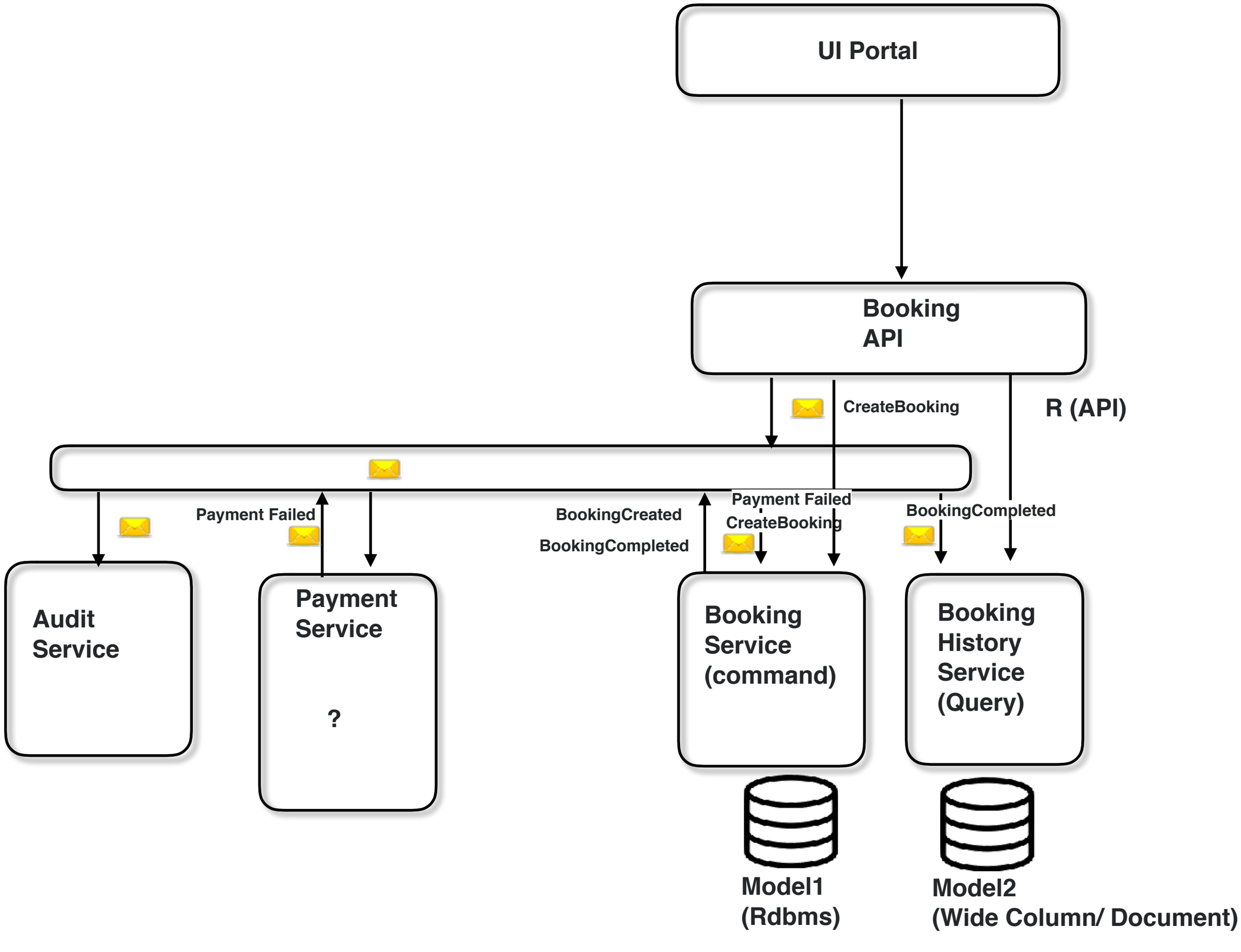
2


UI Portal
Booking API
R (API /graphql)
CUD
Payment Service
?
Booking Service
Booking History Service
Model1

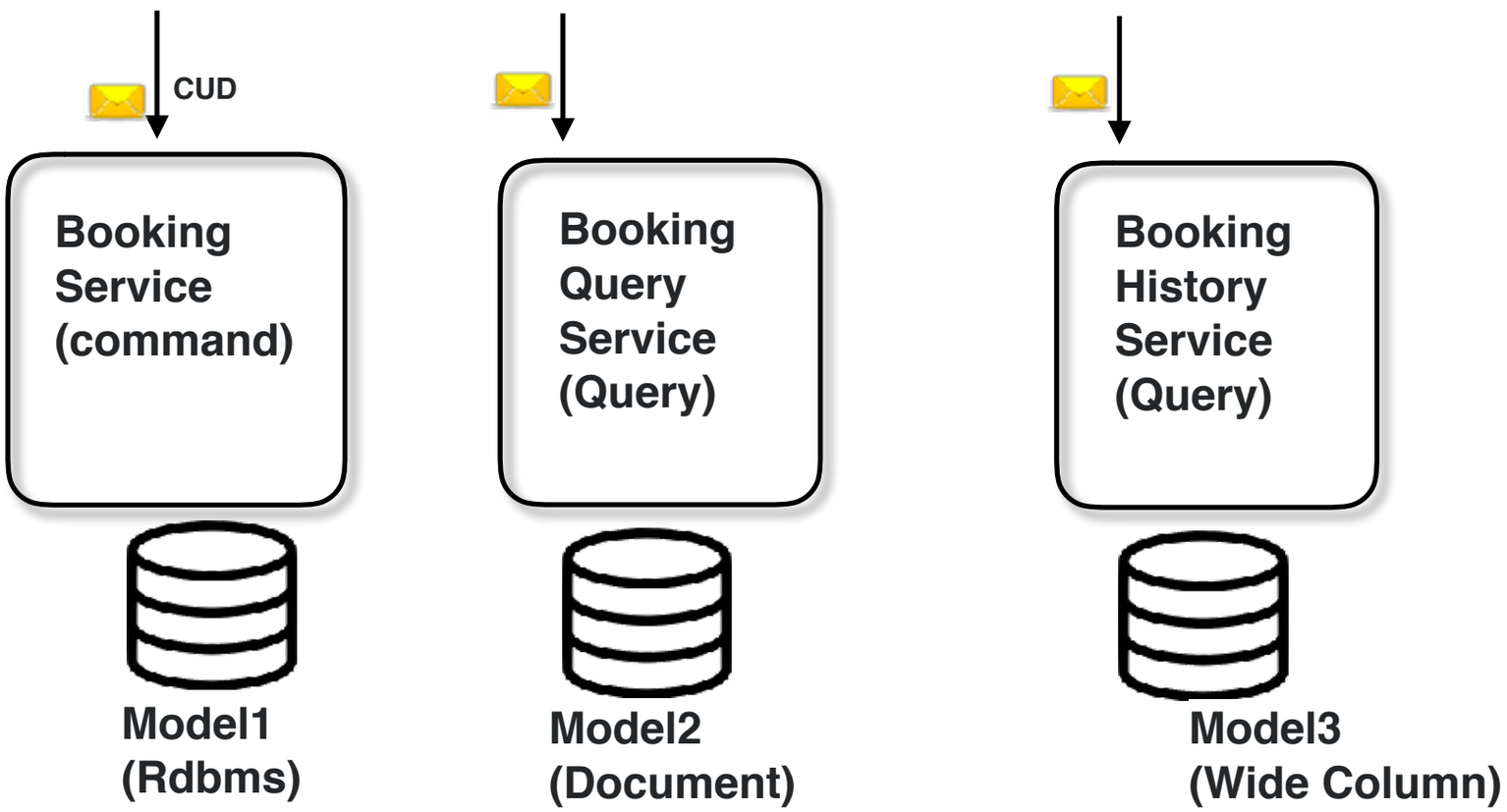
3


UI Portal
Booking API
R (API)
CUD
Payment Service
?
Booking Service
Booking History Service
Model1
Model1
Read Replica

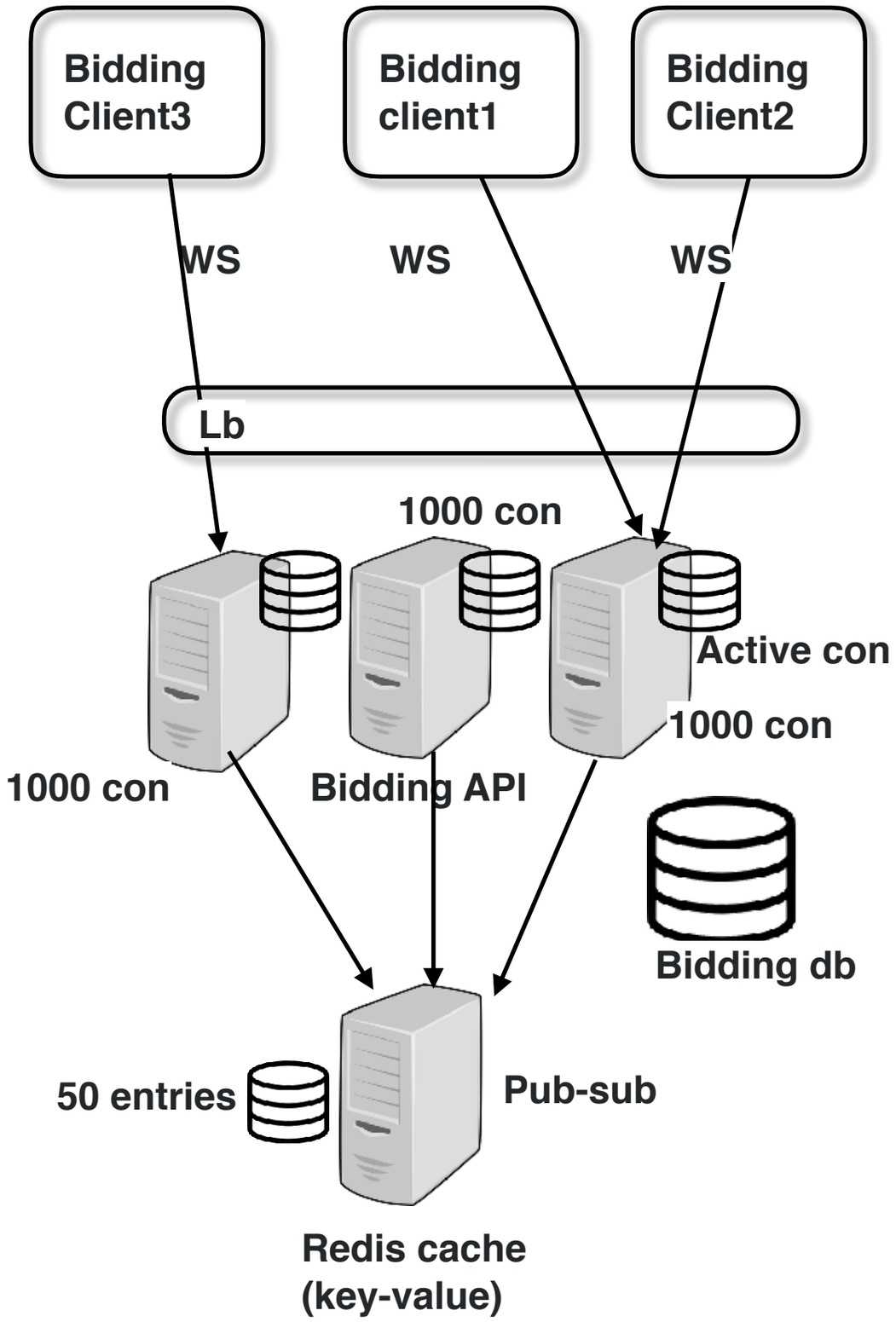
4


UI Portal
Booking
API
CreateBooking
R (API)
Payment Failed
BookingCreated
BookingCompleted
Payment Failed
CreateBooking
BookingCompleted
Audit
Service
Payment
Service
?
Booking
Service
(command)
Booking
History
Service
(Query)
Model1
(Rdbms)
Model2
(Wide Column/ Document)

4


CUD
Booking Service (command)
Booking Query Service (Query)
Booking History Service (Query)
Model1 (Rdbms)
Model2 (Document)
Model3 (Wide Column)
CurrentBookingHistory
PastBookingHistory

Bidding App



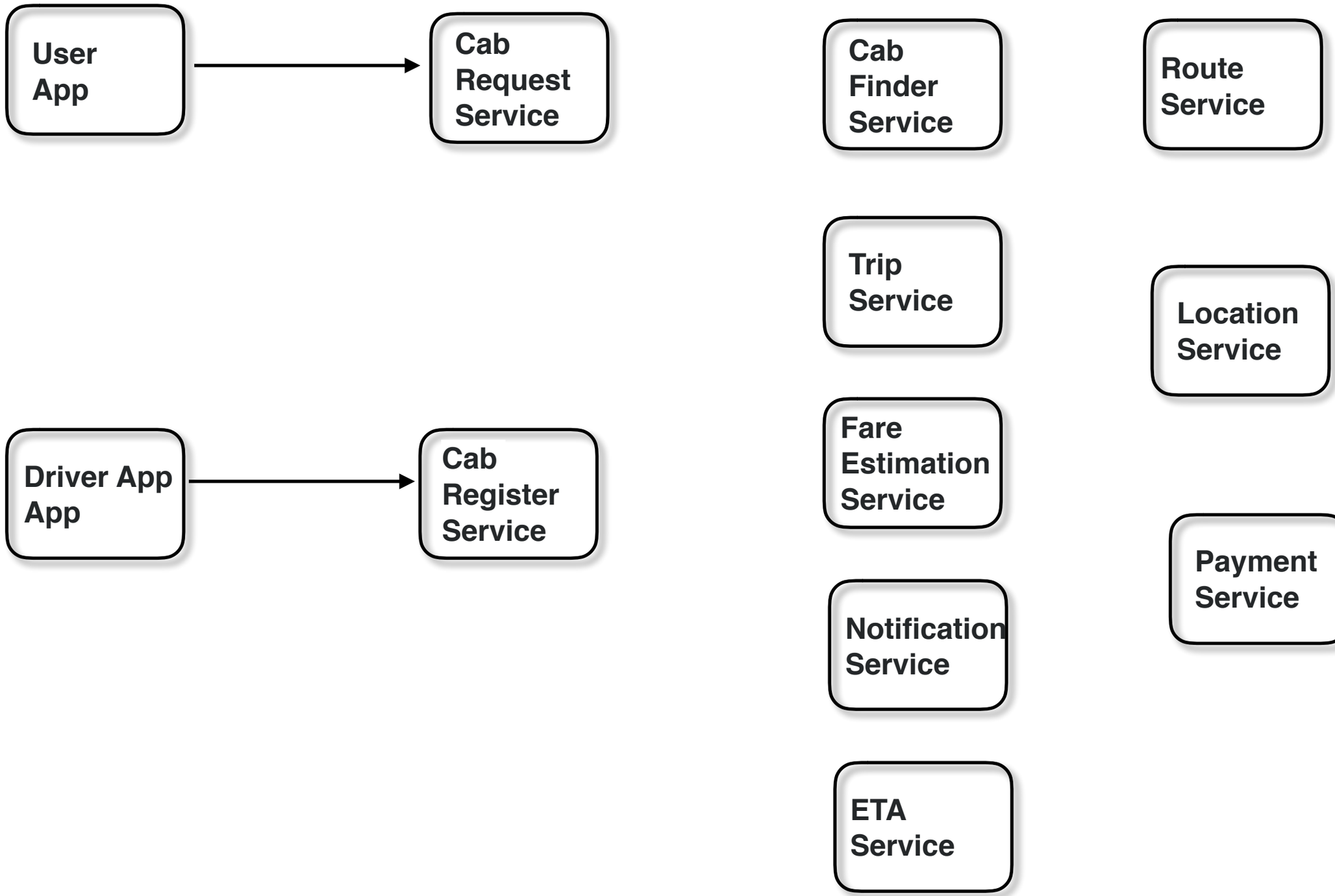
protocol
1. API* vs message
2. pull, push*
3. WS*

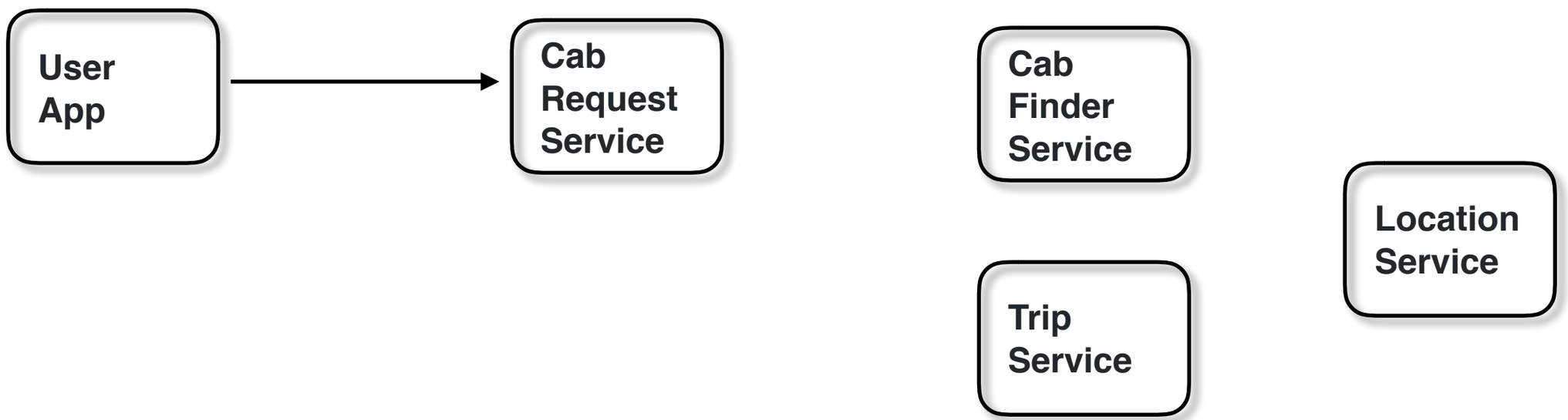
WS, SSE, cloud push notification, custom, webhooks, long polling

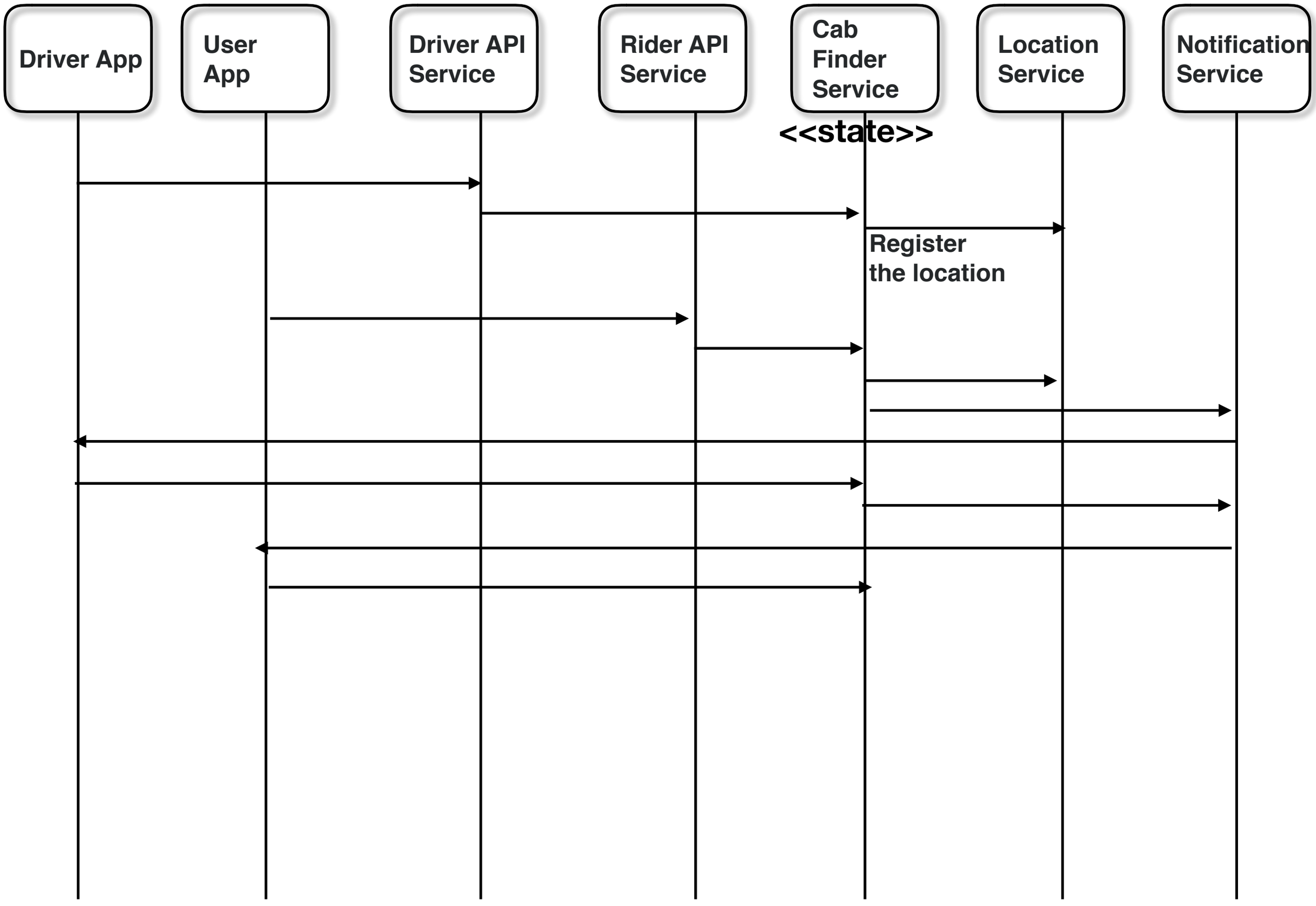
100 server
1000 connection per server
100000 clients
50 bid/house

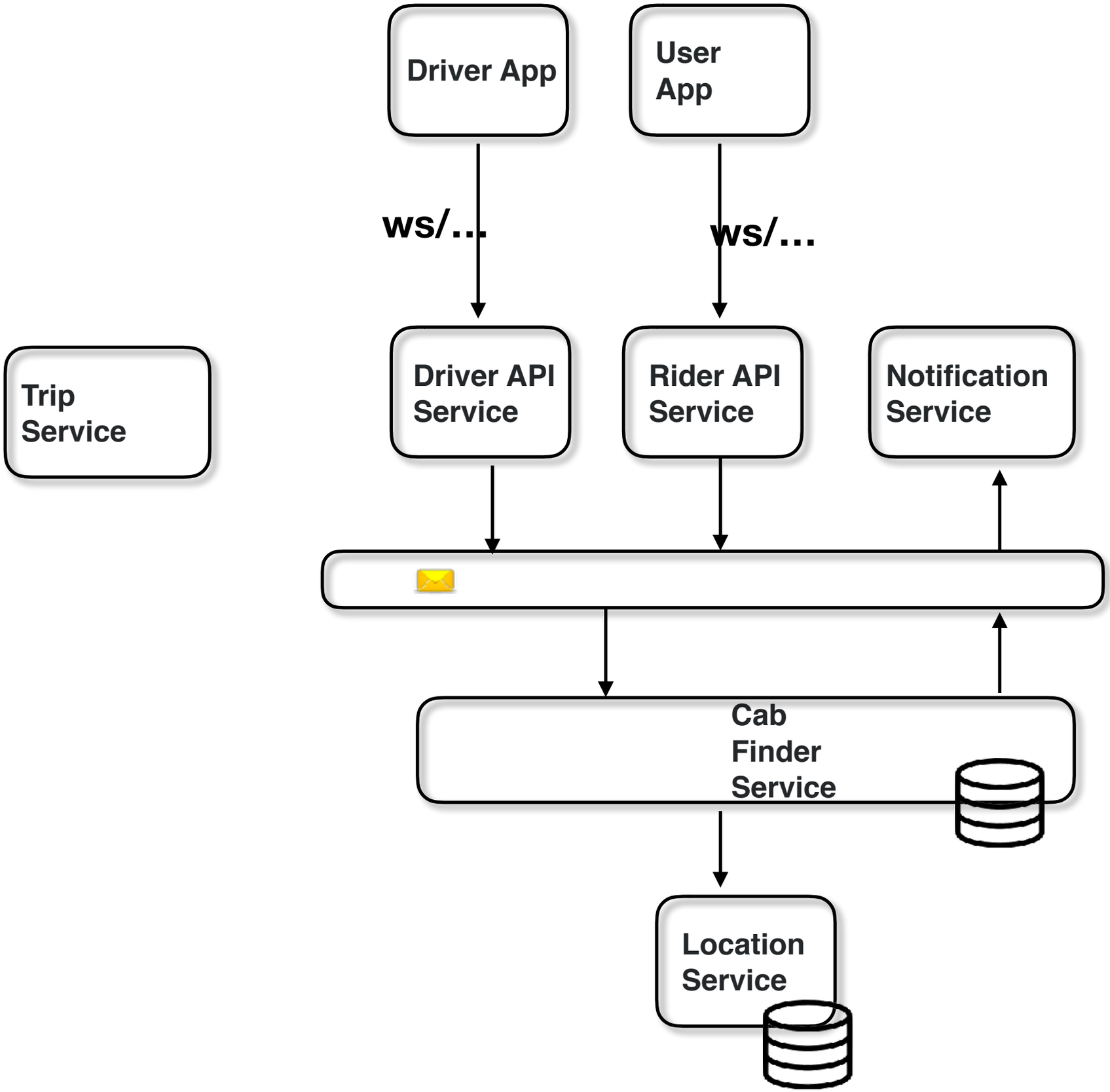
Add a entry to the list
optimistic lock (try until you win)
pessimistic lock (distributed lock)

Uber


User App
Cab Request Service
Cab Finder Service
Route Service
Trip Service
Location Service
Driver App App
Cab Register Service
Fare Estimation Service
Payment Service
Notification Service
ETA Service


User App
Cab Request Service
Cab Finder Service
Location Service
Trip Service
Driver App App
Cab Register Service
Notification Service
Payment Service


Driver App
User App
Driver API Service
Rider API Service
Cab Finder Service
Location Service
Notification Service
<<state>>
Register the location



Api vs queue ?

Api vs queue ?

IOT

Tenant n

100 tenants

Cloud

View outcome/ device config/ Gateway config

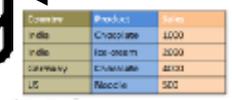
http

Decision
Maker/ Admin

Report

Web
Portal

- columnar db



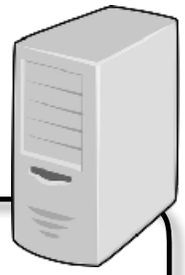
Out come/
Config

- Wide Column

Analytics
(Big Data
processing)

Spark, …

Edge Gateway
(Security)



ws(config updates)

Config
Publisher

1000 - 2000 devices

Device

E

Ingress

Egress

ws

Cloud
Ingress
Service

Reverse
proxy

Reverse
proxy

Stream
Analytics

8000 events/day
1 event = 1 mb
(text + 500 kb image)

Cache

Rest

Collected data
- data lake

Android App

IOS App

Black Berry App

J2ME App

Web App

API

Service 1

<—Bm

Service 2

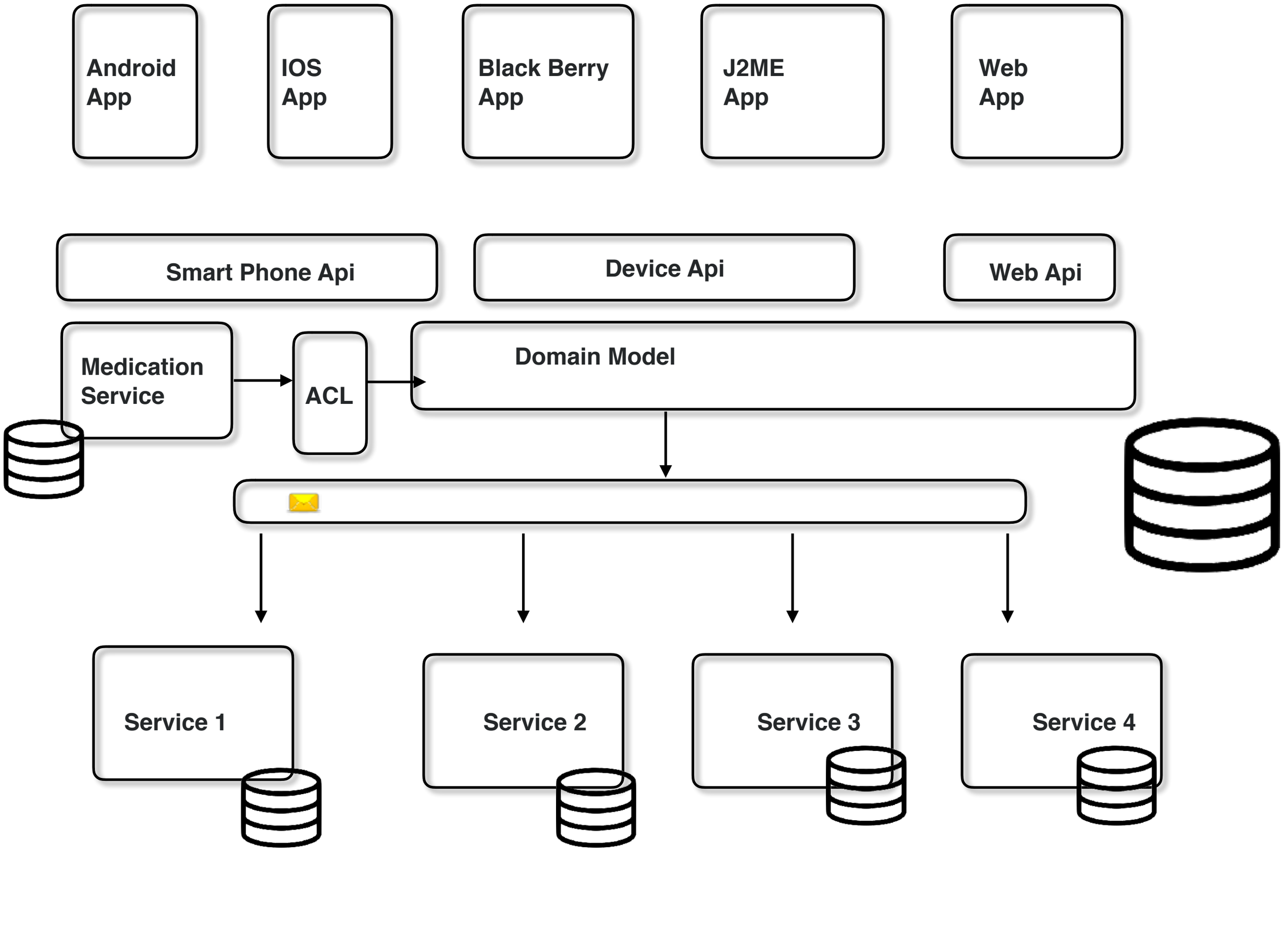
<—Bm

Service 3

<—Bm

Service 4

<—Bm


Android App
IOS App
Black Berry App
J2ME App
Web App
Smart Phone Api
Device Api
Web Api
Medication Service
ACL
Domain Model
Service 1
Service 2
Service 3
Service 4

Patterns

Scalability Patterns

1. Competing Consumer *
2. Queue based load leveling *
3. Sharding *
4. Claim Check *
5. CQRS *
6. Deployment stamps *
7. Geode *
8. Split *
9. Clone *

Performance Patterns

1. Async Request Reply *
2. Static Content Hosting *
3. Materialized View *
4. Backends for front ends *
5. Cache Aside *
6. Index Table *
7. Throttling *
8. Rate Limiting *

Centralize Cross Cutting Concerns

1. Gateway Aggregation *
2. Gateway Routing *
3. Gateway Offloading *
4. Gatekeeper *
5. Side Car *
6. Ambassador *
7. Federated Identity *
8. External Config store *

Legacy modernisation Patterns

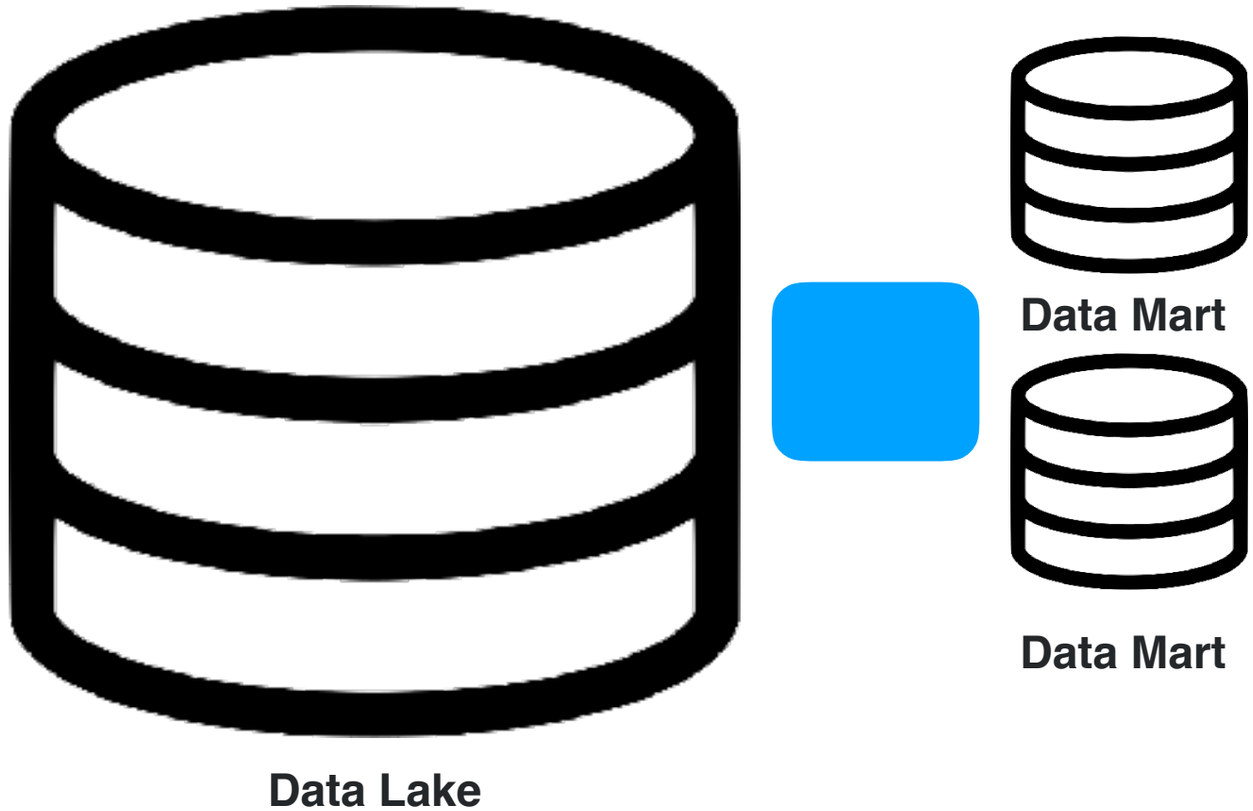
1. Strangler *
2. Anti Corruption Layer *

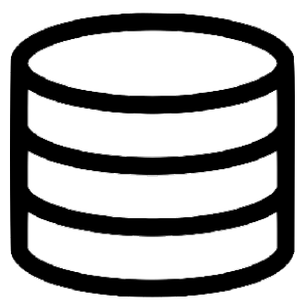
Reliability Availability Patterns

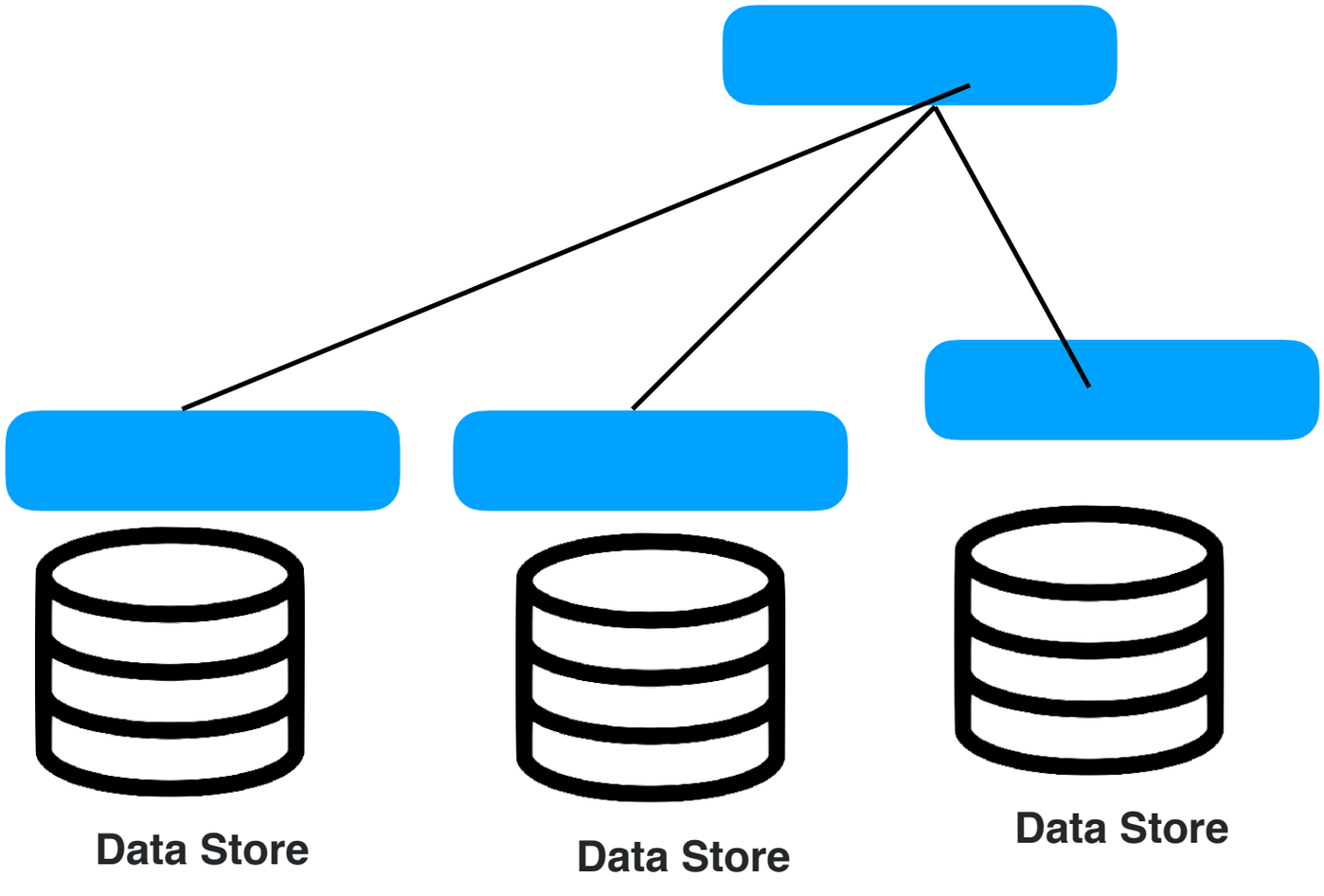
1. Retry *
2. Circuit breaker *
3. Bulk head *
4. Compensatable transaction *
5. SAGA *
6. Schedular Agent supervisor *
7. Health endpoint *

Communication Patterns

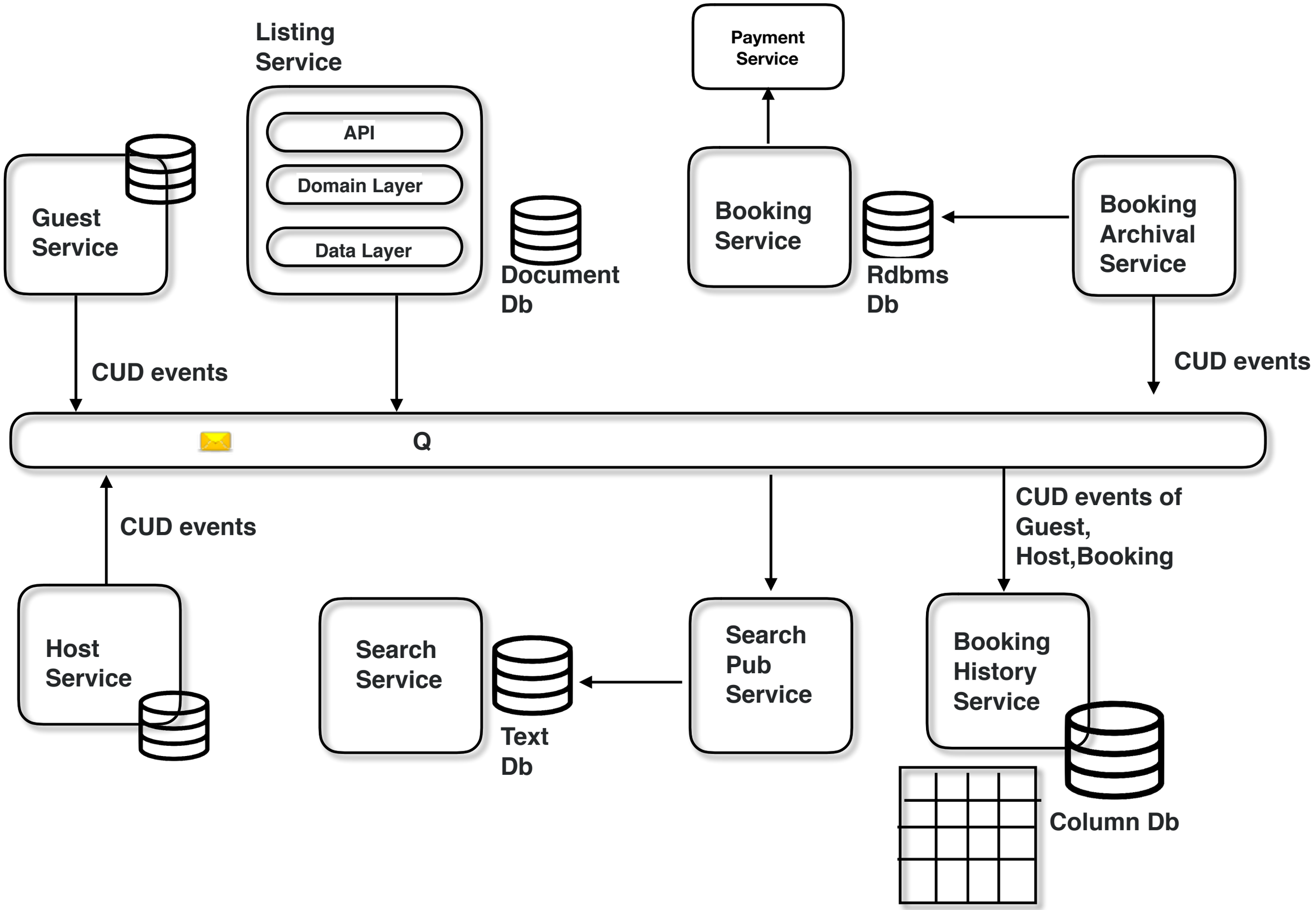
1. Pub sub *
2. Priority Queue *
3. Event Sourcing *
4. Choreography *
5. Leader Election (last weapon)
6. Sequential convoy *

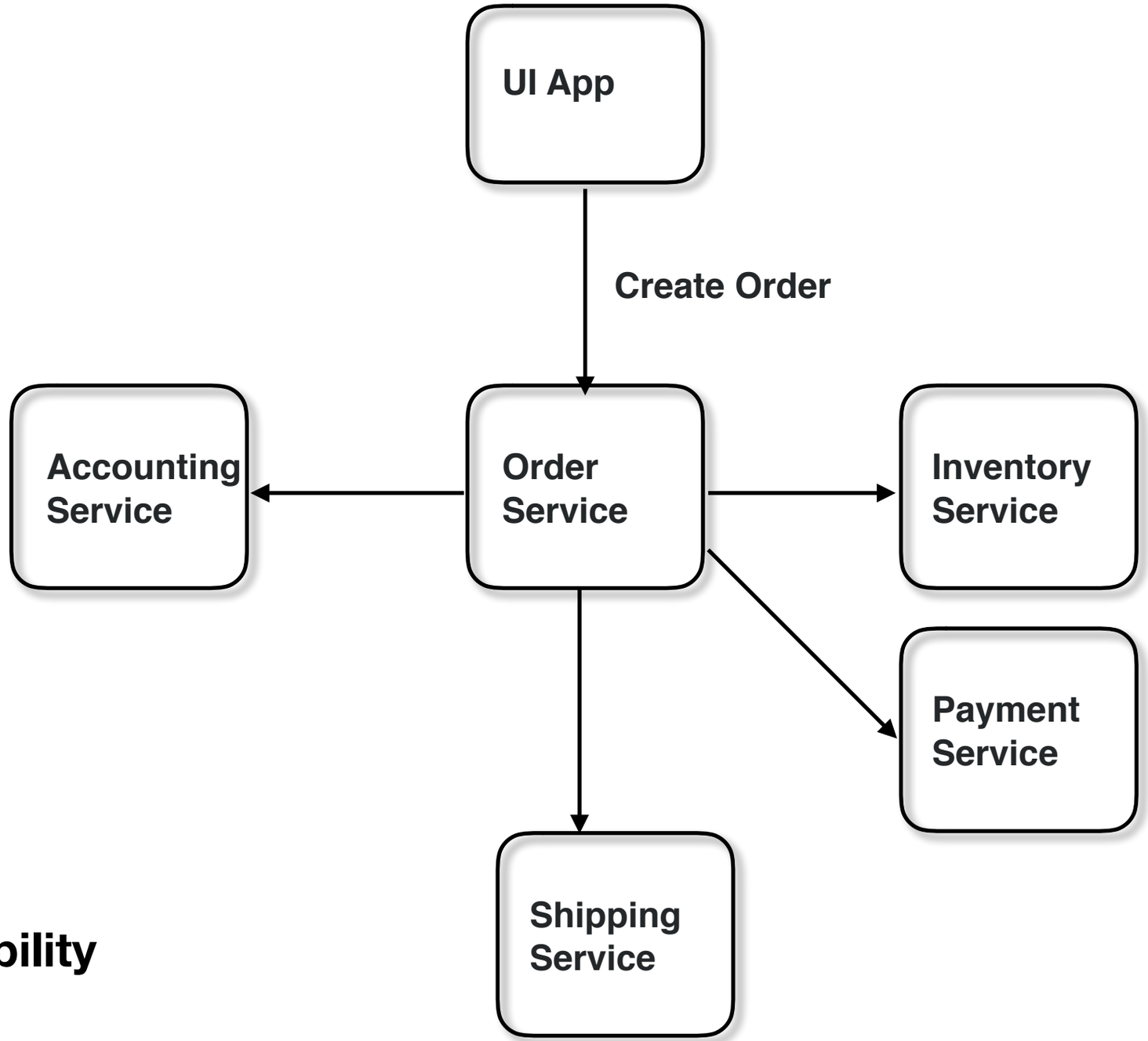

Data Mart
Data Mart
Data Lake


Data Store


Data Store
Data Store
Data Store

Appendix


Listing Service
API
Domain Layer
Data Layer
Document Db
Payment Service
Booking Service
Rdbms Db
Booking Archival Service
Guest Service
CUD events
CUD events
Q
CUD events
CUD events of Guest, Host,Booking
Host Service
Search Service
Text Db
Search Pub Service
Booking History Service
Column Db



testing
maintainability
scalability
reliability
deployment

EDA
DDD
- Hexagonal
- ubiquitous
- Event Sourcing

UI App

Create Order

Accounting Service

Do () I undo()

Order Service

Inventory Service

Do () I undo()

Stream Analytics < 7 days

Alerts

Order Created XXXFailed

Q

Order Account Updated

Order Created

XXXFailed

Order Created

Order Inventory Updated

Domain Event Bus (Topic/hub)

Event Service

Order Shipping Completed /Failed

Q

Order Created

Order Shipping Completed

Shipping Service

Do () I undo()

Wide Column

API Layer

Domain Layer

Data Layer

Data Lake

DWH (snapshot)

Rdbms (current)

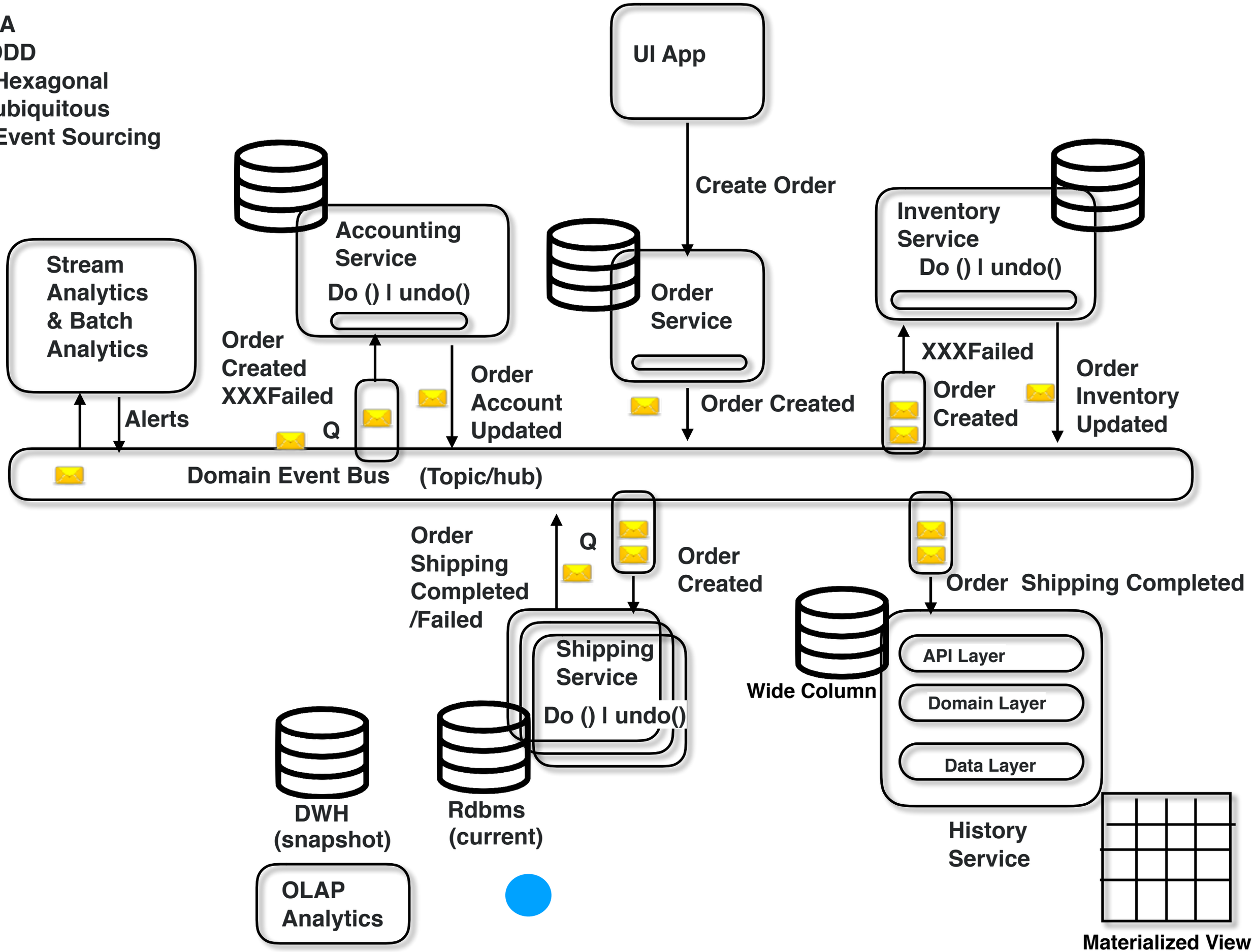
History Service

Big Data Analytics (Years)

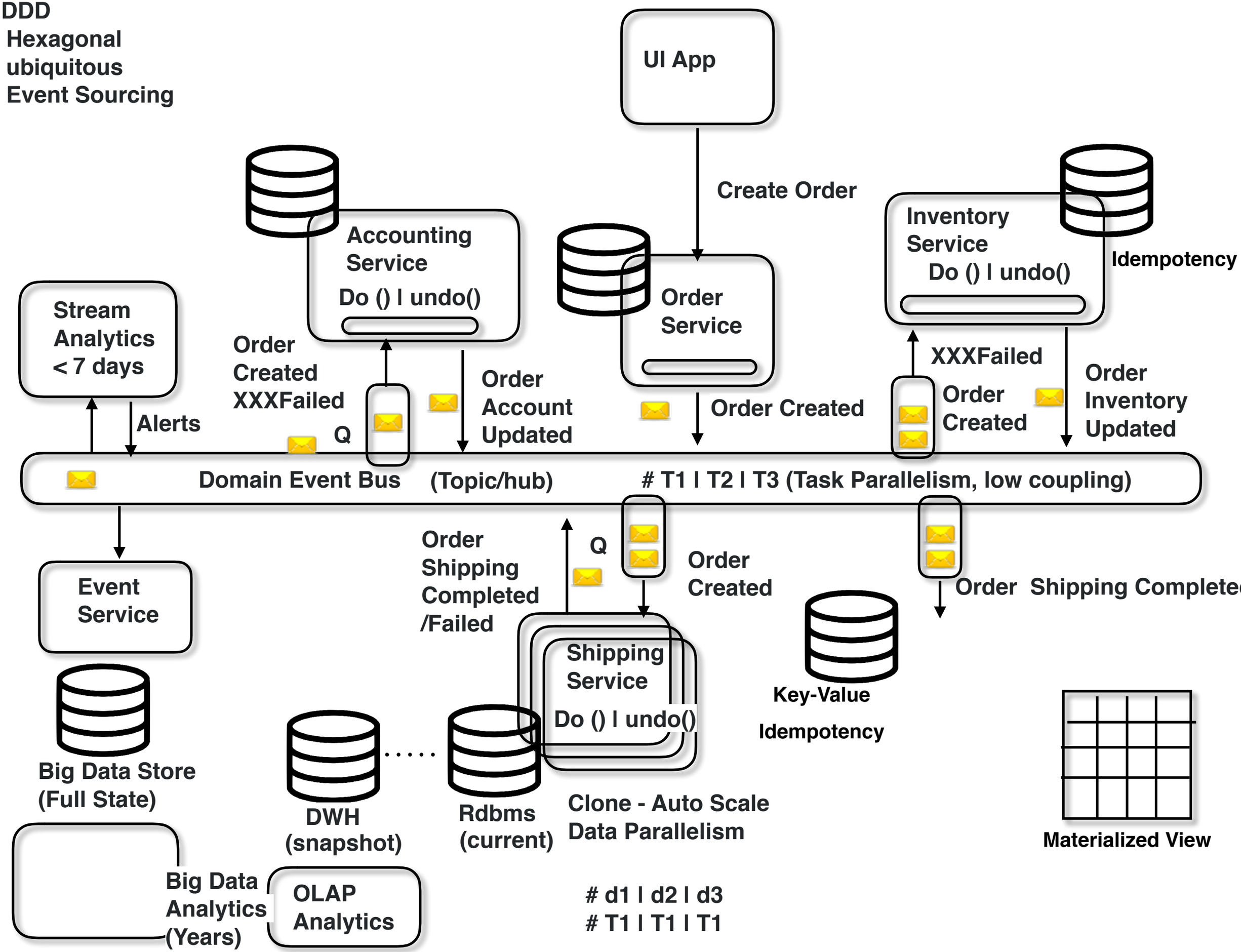
OLAP Analytics

Materialized View

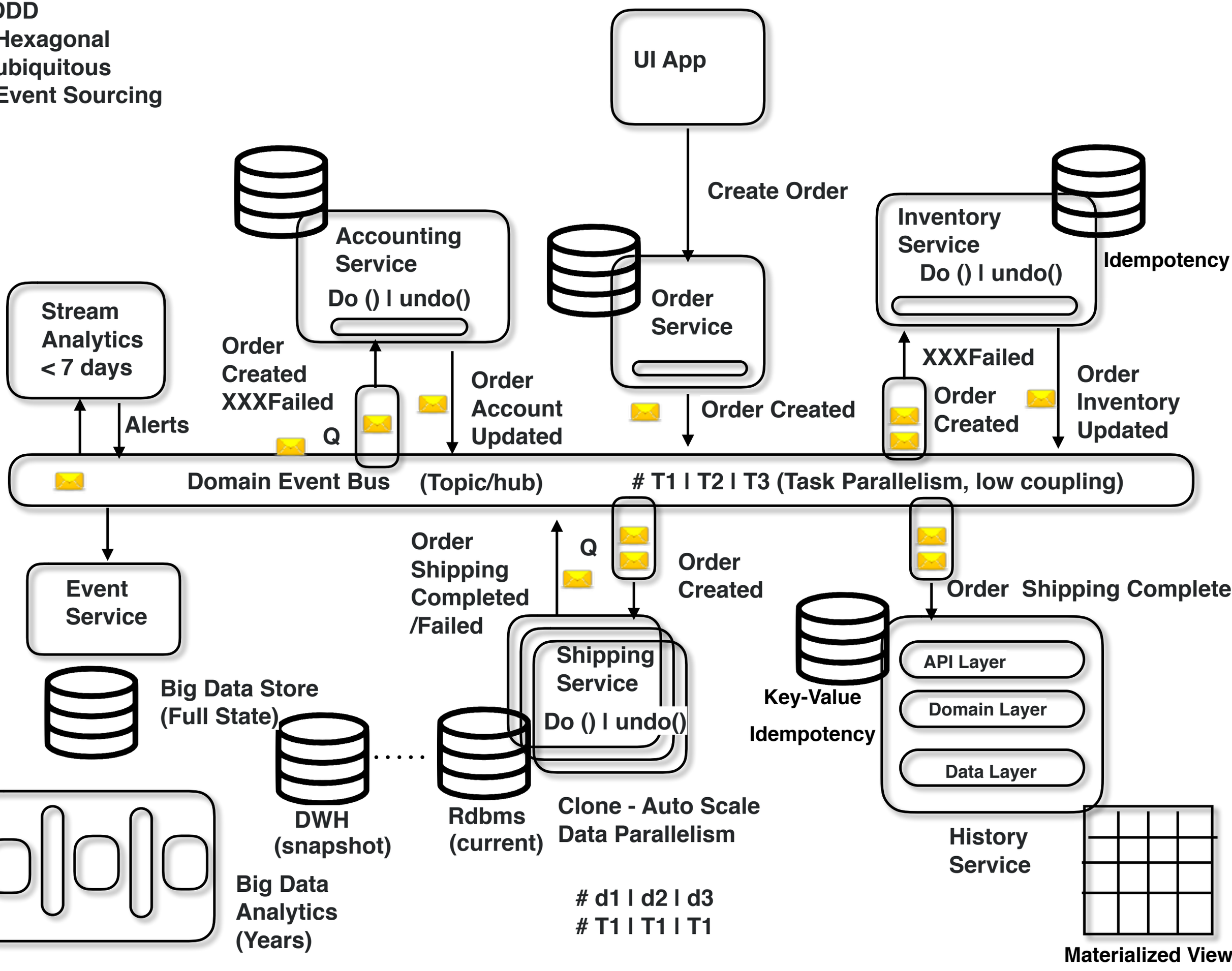
EDA
DDD
- Hexagonal
- ubiquitous
- Event Sourcing



DDD
- Hexagonal
- ubiquitous
- Event Sourcing



DDD
- Hexagonal
- ubiquitous
- Event Sourcing



DDD
- Hexagonal
- ubiquitous
- Event Sourcing

UI App

static
Web Resource

CDN

Create Order

Accounting
Service
Do () l undo()

Order
Service

Inventory
Service
Do () l undo()

Idempotency

Stream
Analytics
< 7 days

Alerts

Order
Created
XXXFailed

Q

Order
Account
Updated

Order Created

XXXFailed

Order
Created

Order
Inventory
Updated

Domain Event Bus (Topic/hub) # T1 l T2 l T3 (Task Parallelism, low coupling)

Logs
...

Big Data
Analytics
(Years)

Order
Shipping
Completed
/Failed

Q

Order
Created

Order Shipping Completed

Shipping
Service
Do () l undo()

Key-Value

Idempotency

History
Service

Reporting
tools

DWH
(snapshot)

Rdbms
(current)

Clone - Auto Scale
Data Parallelism

Materialized View

d1 l d2 l d3
T1 l T1 l T1

Response

token
+
Claim Set

token

AuthZero
token

Identity
Provider
(cloud)

Service 1

Service 2

Service 3

AuthZero
token
+
Claim Set

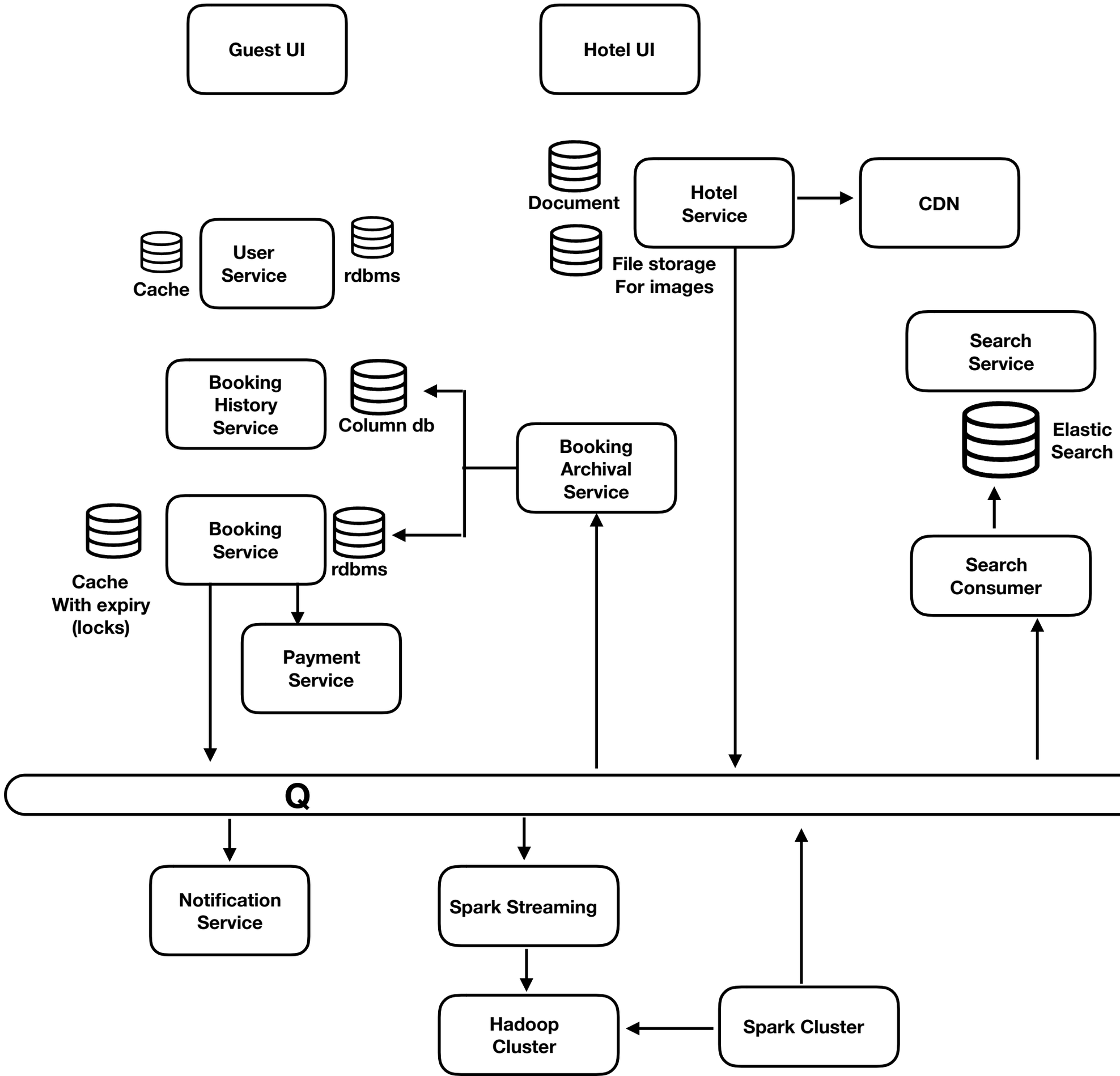
Credential

Identity
Provider
(AuthZero)

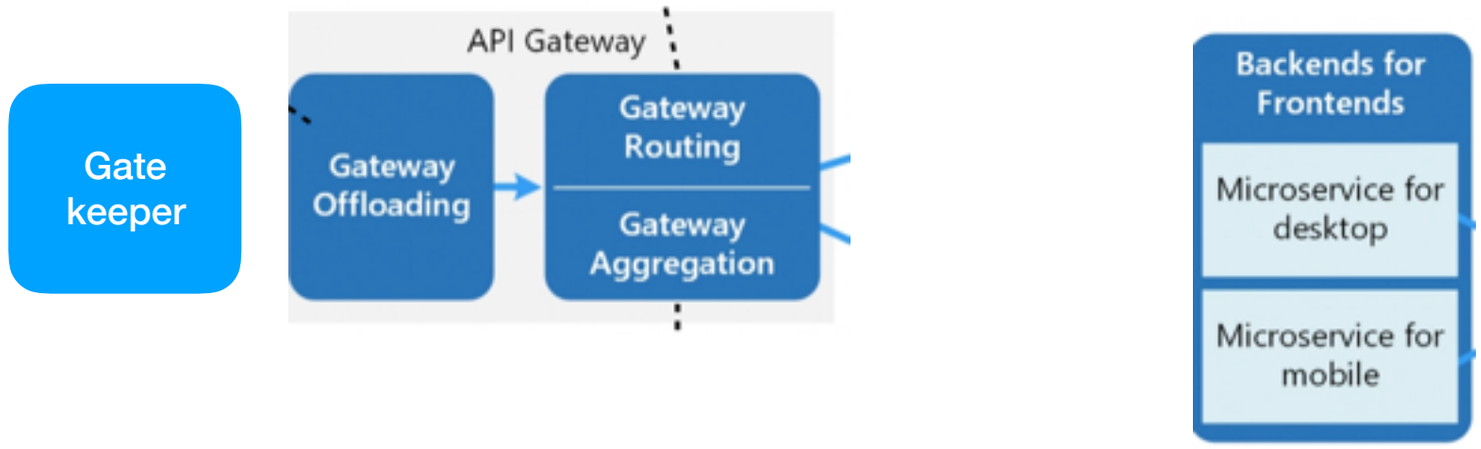
Mapping

HR-> claims[….]
India -> claims[…]

Configuration
Store
Consul, Redis, …


Guest UI
Hotel UI
Document
Hotel Service
CDN
File storage For images
Cache
User Service
rdbms
Search Service
Booking History Service
Column db
Elastic Search
Booking Archival Service
Search Consumer
Cache With expiry (locks)
Booking Service
rdbms
Payment Service
Q
Notification Service
Spark Streaming
Hadoop Cluster
Spark Cluster

- Azure Front door

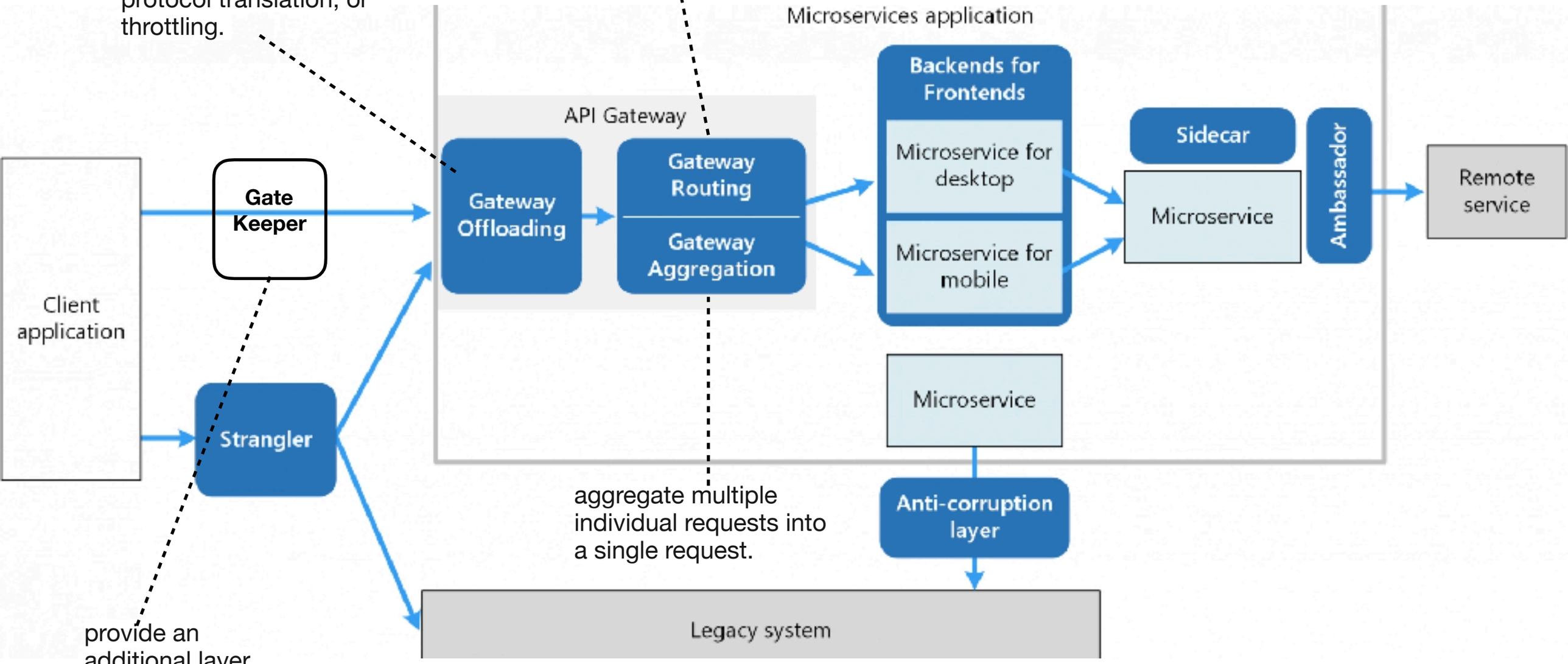


- External Config store

- Federated identity

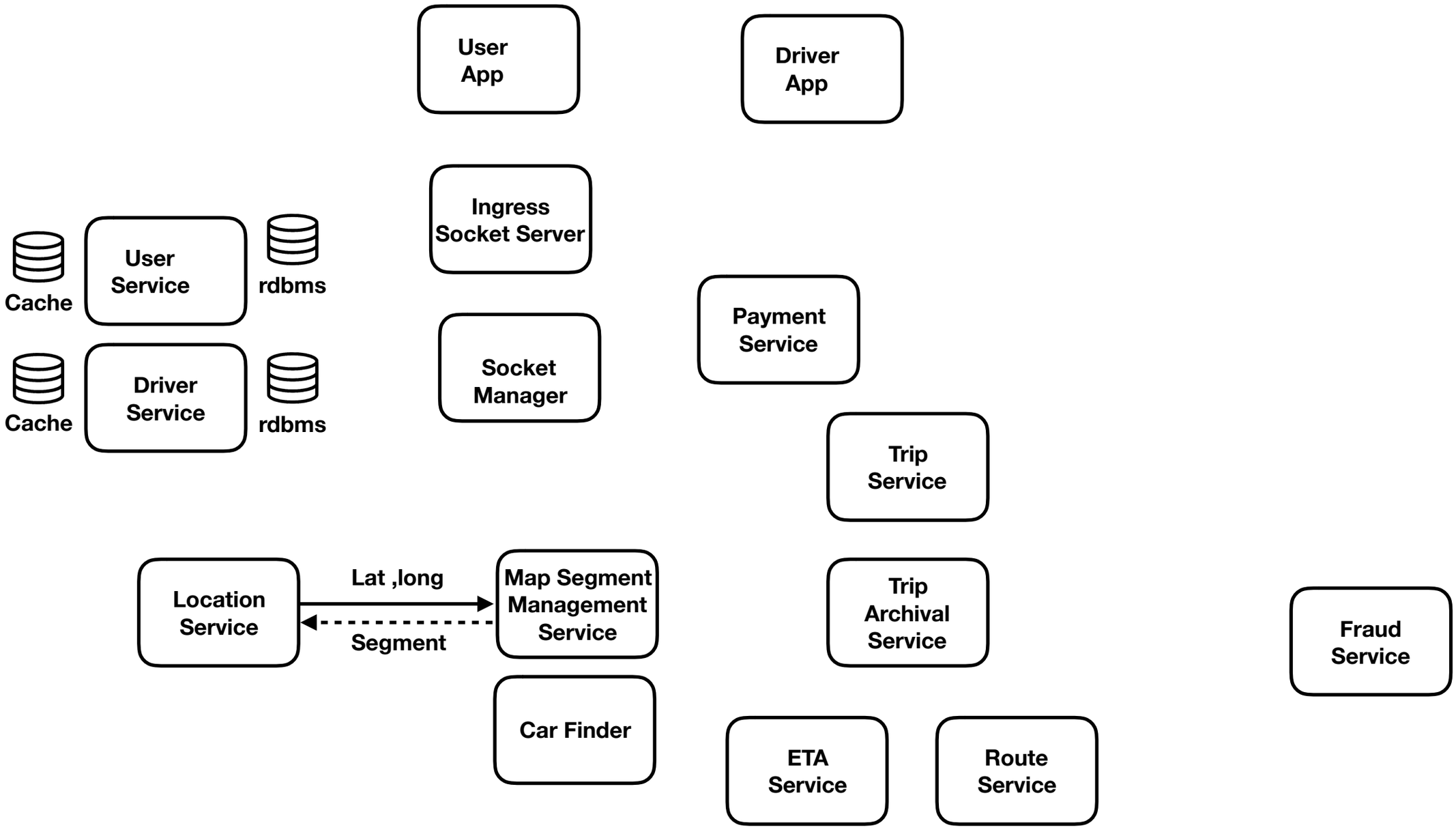
Offload some features into a gateway, particularly cross-cutting concerns such as certificate management, authentication, SSL termination, monitoring, protocol translation, or throttling.

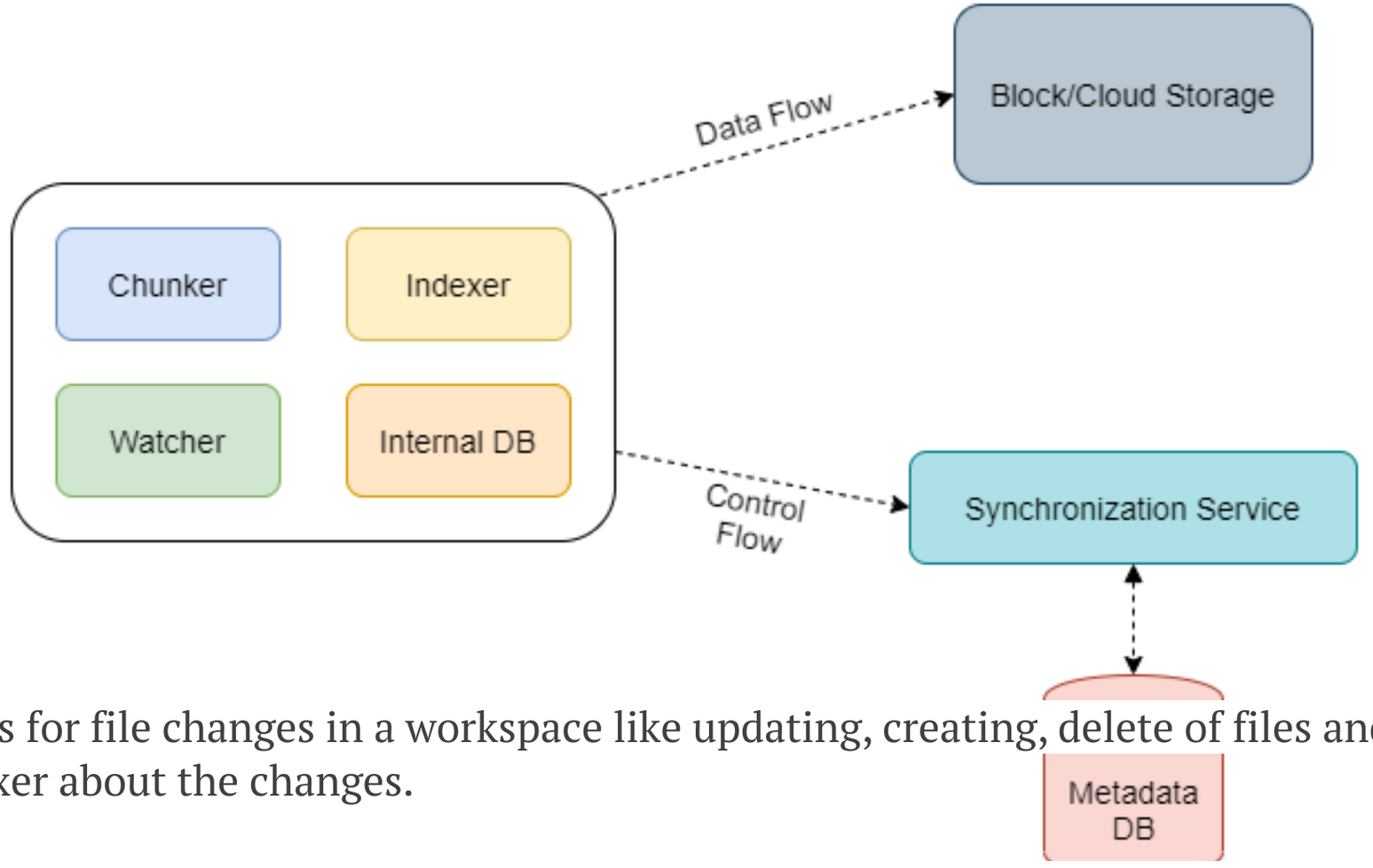
Use application Layer 7 routing to route the request to the appropriate instances.



aggregate multiple individual requests into a single request.

provide an additional layer of security and limit the system's attack surface.


User App
Driver App
Ingress Socket Server
User Service
Cache
rdbms
Payment Service
Socket Manager
Driver Service
Cache
rdbms
Trip Service
Location Service
Lat ,long
Segment
Map Segment Management Service
Trip Archival Service
Fraud Service
Car Finder
ETA Service
Route Service

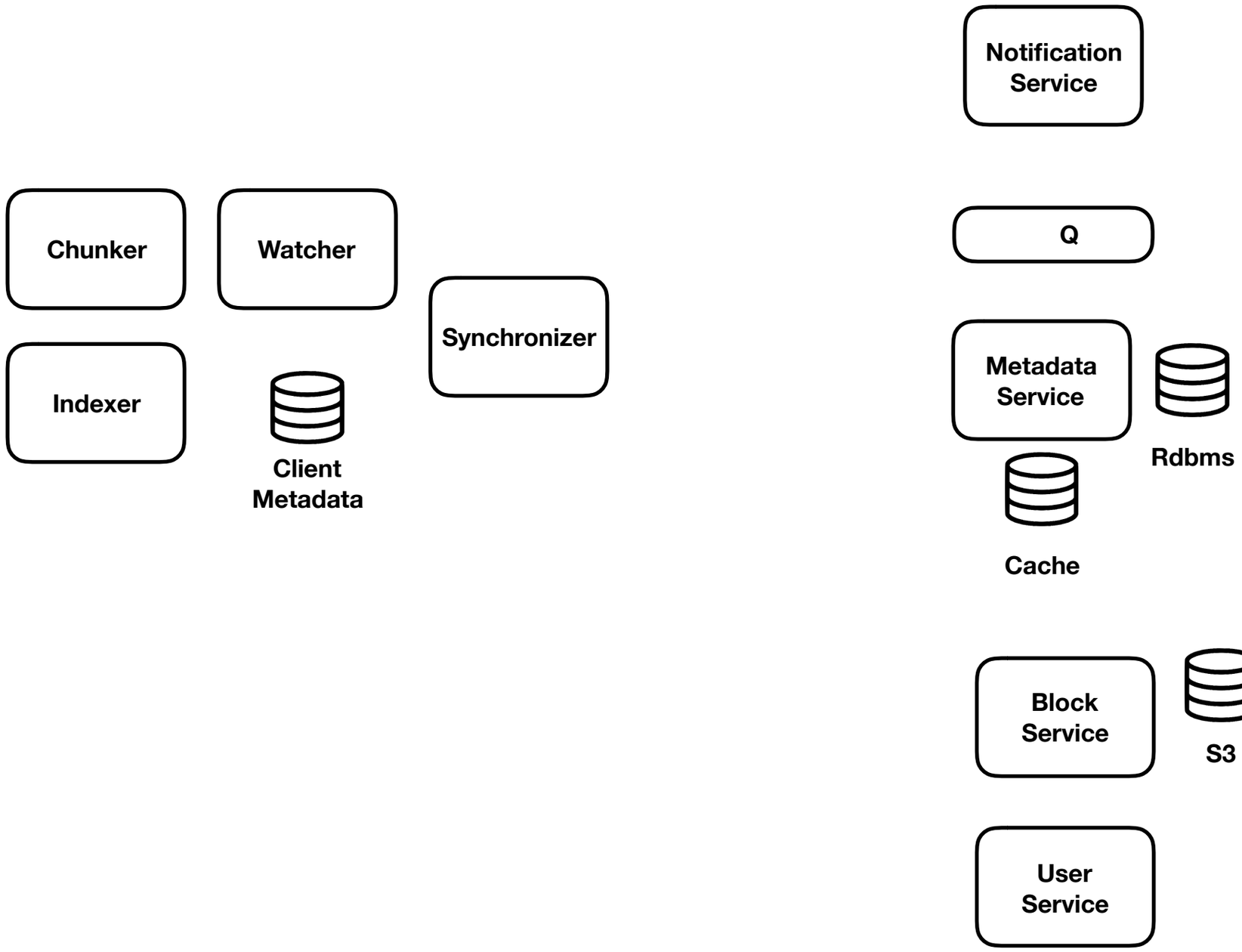


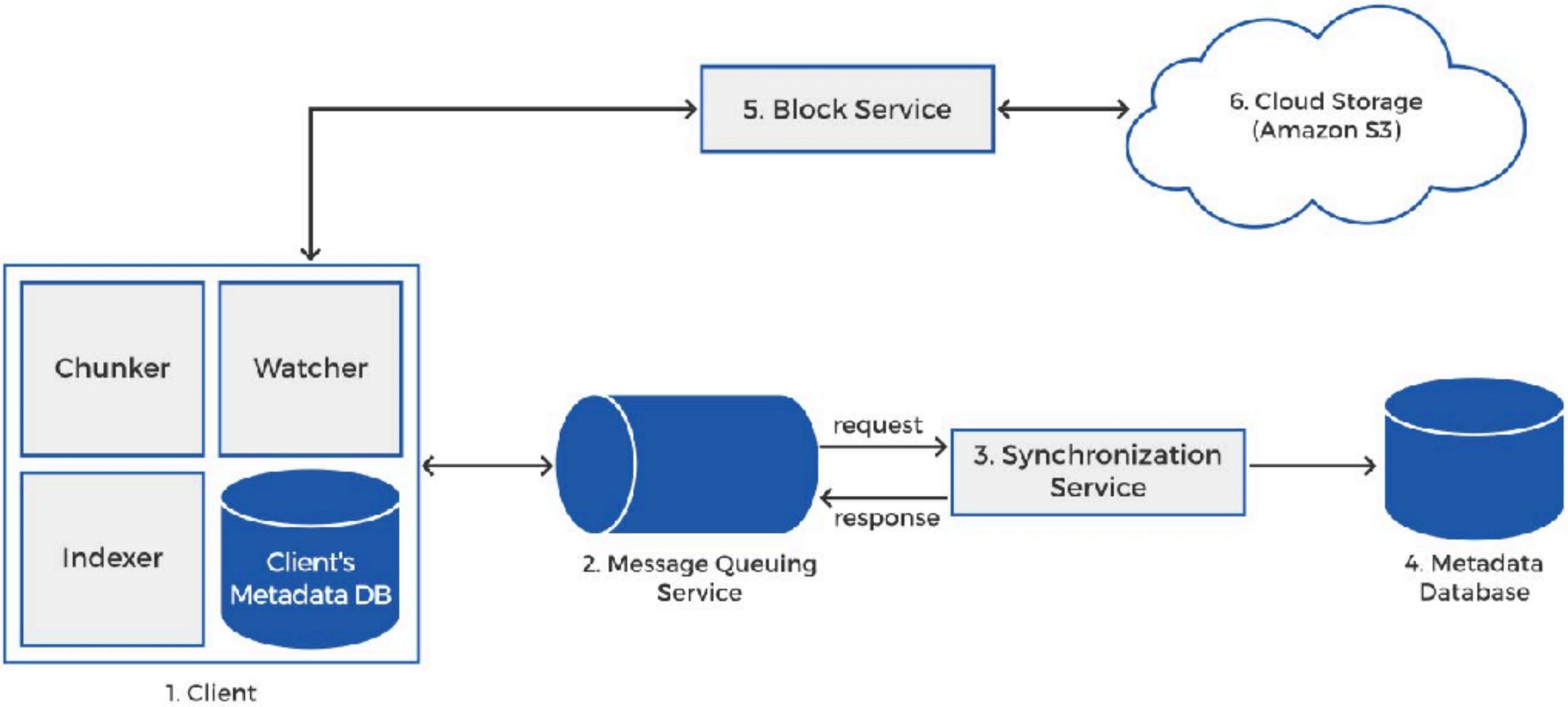
Watcher monitors for file changes in a workspace like updating, creating, delete of files and folders. The watcher notifies the Indexer about the changes.

Indexer listens for the events from the watcher and updates the Client Metadata Database with information about the chunks of the modified file.

Chunker – The chunker, responsible for splitting the file into smaller pieces, is present within the client. The chunker is also responsible for detecting the chunks that are updated by the client so that only these specific chunks are uploaded to the cloud storage, thus saving bandwidth. It also uploads these chunks to the S3 and notifies the indexer with the hash and the url received from the S3.

Chunker will also reassemble the file by putting together the chunks in their correct order.


Chunker
Watcher
Synchronizer
Indexer
Client
Metadata
Notification
Service
Q
Metadata
Service
Rdbms
Cache
Block
Service
S3
User
Service


5. Block Service
6. Cloud Storage
(Amazon S3)
Chunker
Watcher
Indexer
Client's
Metadata DB
request
response
3. Synchronization
Service
2. Message Queuing
Service
4. Metadata
Database
1. Client

Runs the Business

Read

Manage the Business

Composite UI

Read Service

Reporting Tools

DWH (cube)

Stream Analytics

Batch Analytics

Data Lake

Graph

Column db (big Table)

Time Series (big Table)

key Value

Document

Geo spatial (big Table)

Rdbms

Runs the Business

Read

Manage the Business

Composite UI

Portal

Power bi

Data Bricks (compute)

Batch Analytics

Batch Analytics

Batch Analytics

Data Lake (per tenant)

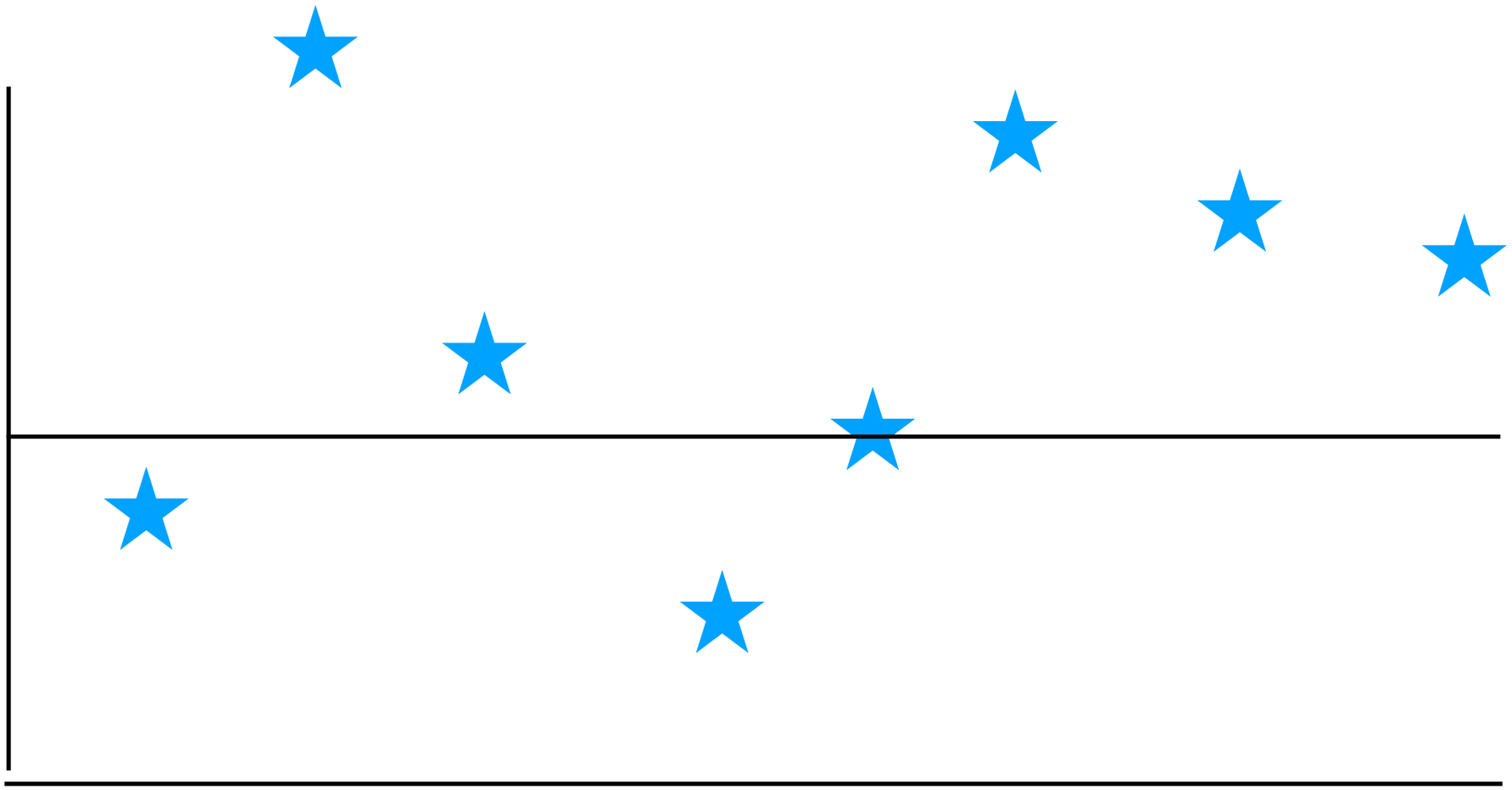
Mongo

Gold

Staging

Landing

Config/ portal



Observing of one item over time
Sensor - 23, 21,21,23,22,……..

Client App

Cloud App

Watcher

Chunker Service

Block Service

Z axis key ?
country
state
account type
file type

File Storage
(object storage)

Notification Service

Indexer Service

Sync Service

Metadata Service

User Metadata Service

Metadata Storage
(rdbms)

Disk Storage

User Metadata Storage
(rdbms)

Region 1

Block Service

Metadata Service

File Storage (object storage)

Notification Service

Metadata Storage (rdbms)

U1,u2,u3

Sync Service

Gateway/ Reverse proxy

Region 2

Block Service

Metadata Service

File Storage (object storage)

Notification Service

Metadata Storage (rdbms)

U4,u5,u6

Region 1

Block Service

Metadata Service

File Storage (object storage)

Notification Service

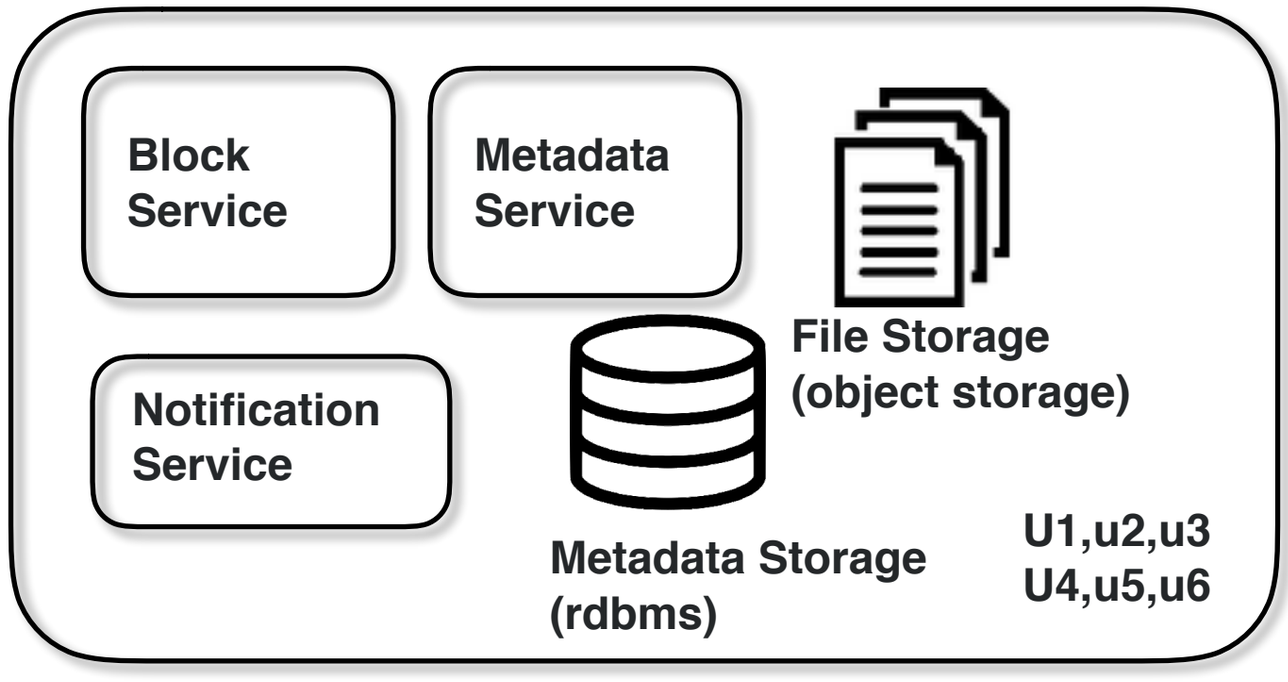
Metadata Storage (rdbms)

U1,u2,u3
U4,u5,u6

Sync Service

Gateway/ Reverse proxy

Region 2



Region 1

Block Service

Metadata Service

Notification Service

Sync Service

Region 2

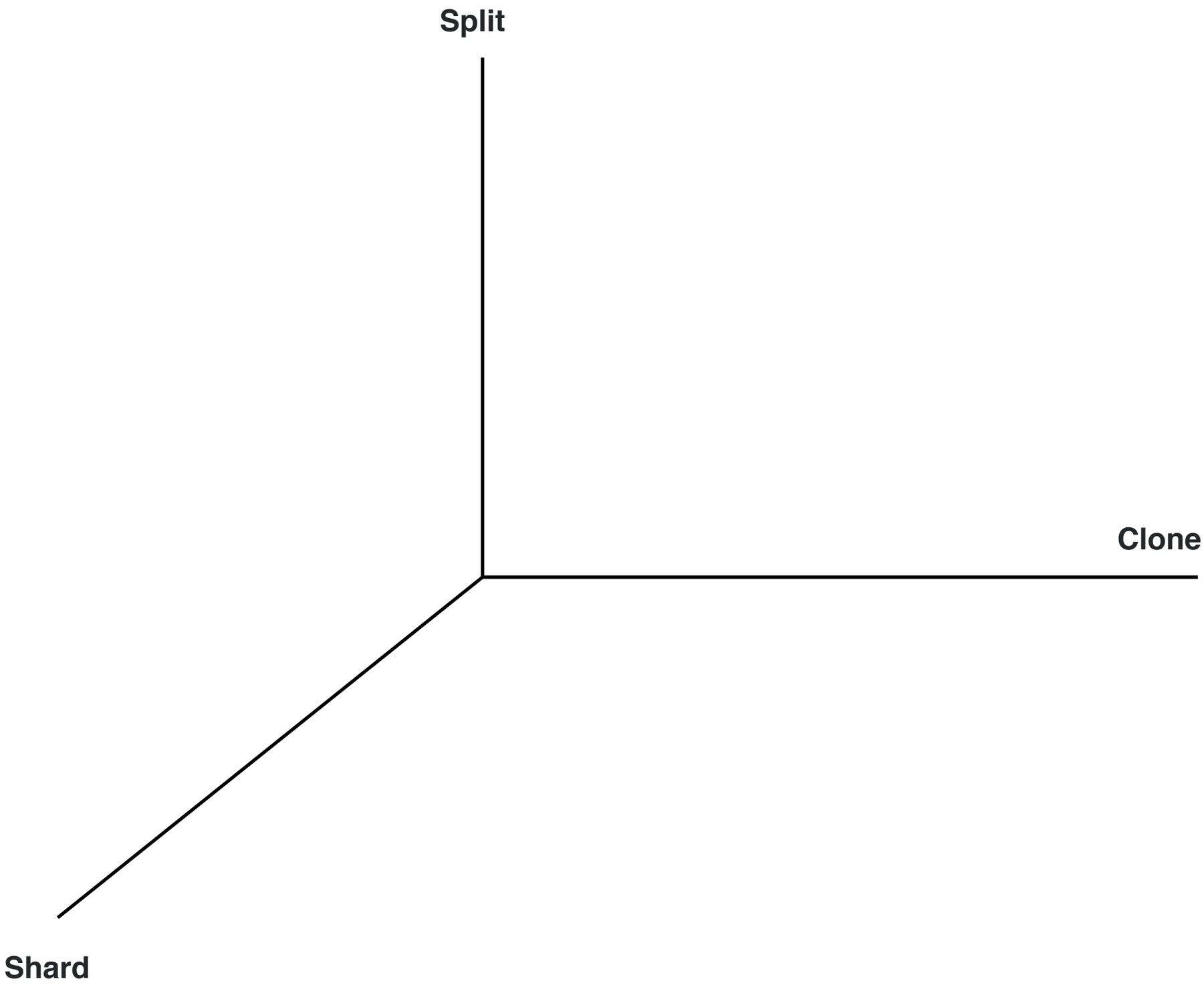
Block Service

Metadata Service

Notification Service

File Storage (object storage)

Metadata Storage (rdbms)


Split
Clone
Shard
Queue

Counter

C

BP

Counter

C

BP

Counter

E

BP

Counter

?

BP

General

C

E

?

BP

BP

BP

BP

BP

BP

BP

General

C

E

?

BP

D BP

I BP

BP

D BP

I BP

General

C

E

?

Split
General : BP

D BP

D BP

D BP

Clone
DBP : DBP

D BP

D BP

D BP

Shard
IBP : DBP

I BP

I BP

I BP

General

C

E

?

Split
General : BP

D BP

D BP

D BP

Shard
IBP : DBP

I BP

I BP

I BP

Scalability Cube - 50 rules for high Scalability

1

3

Split

Y axis

SC 1#
D 1#

Inventory

SC 2#
D 2#

Account

SC#
D#

Inventory
female

SC#
D#

Inventory
female

SRP / SOC

Clone

X axis

Vertical Slicing (by columns)

Split different things

SC#
D 1#

Inventory

male

SC#
D 2#

Inventory
female

Horizontal Slicing (by rows)

Split similar things

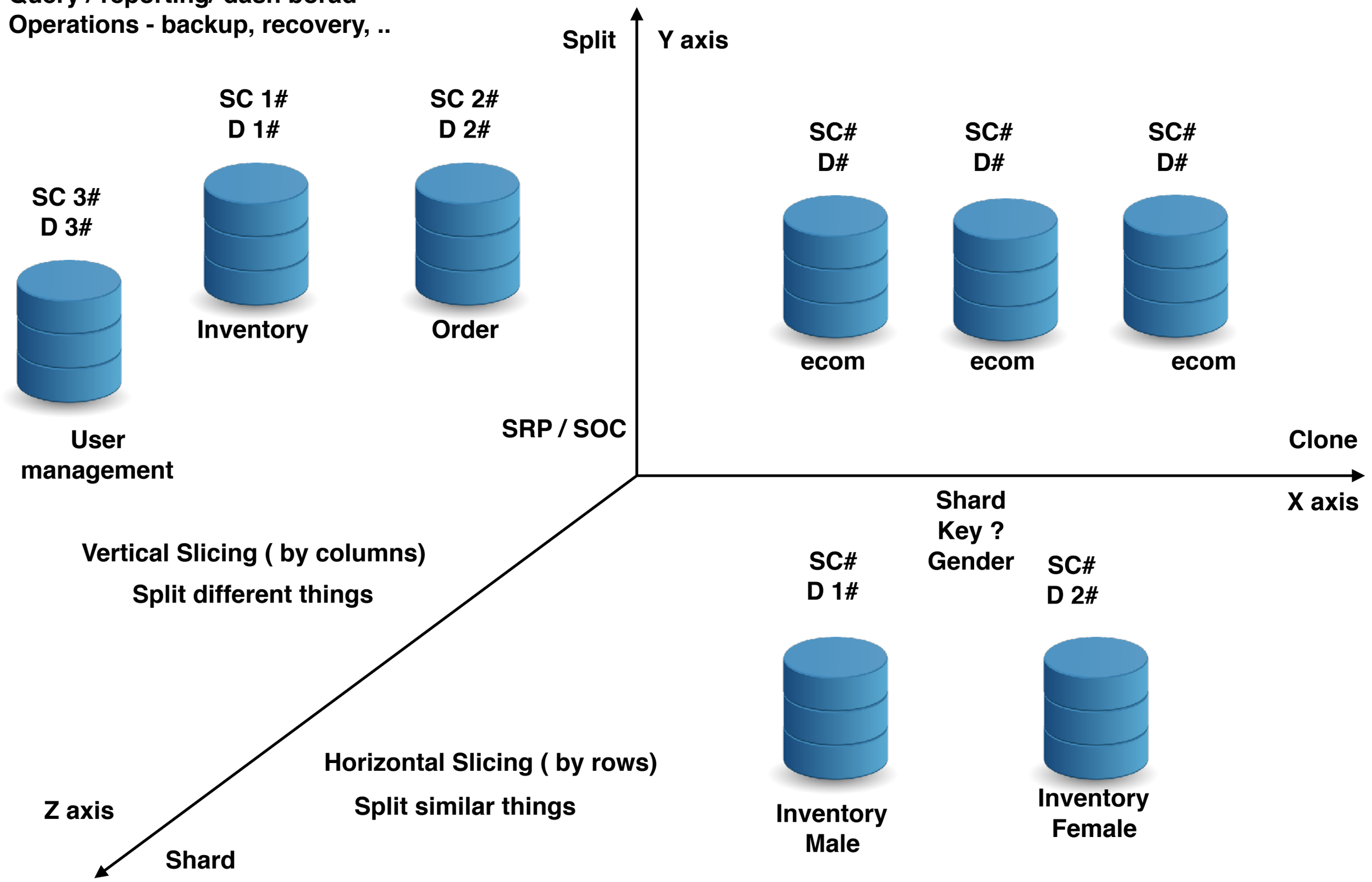
Z axis

Shard

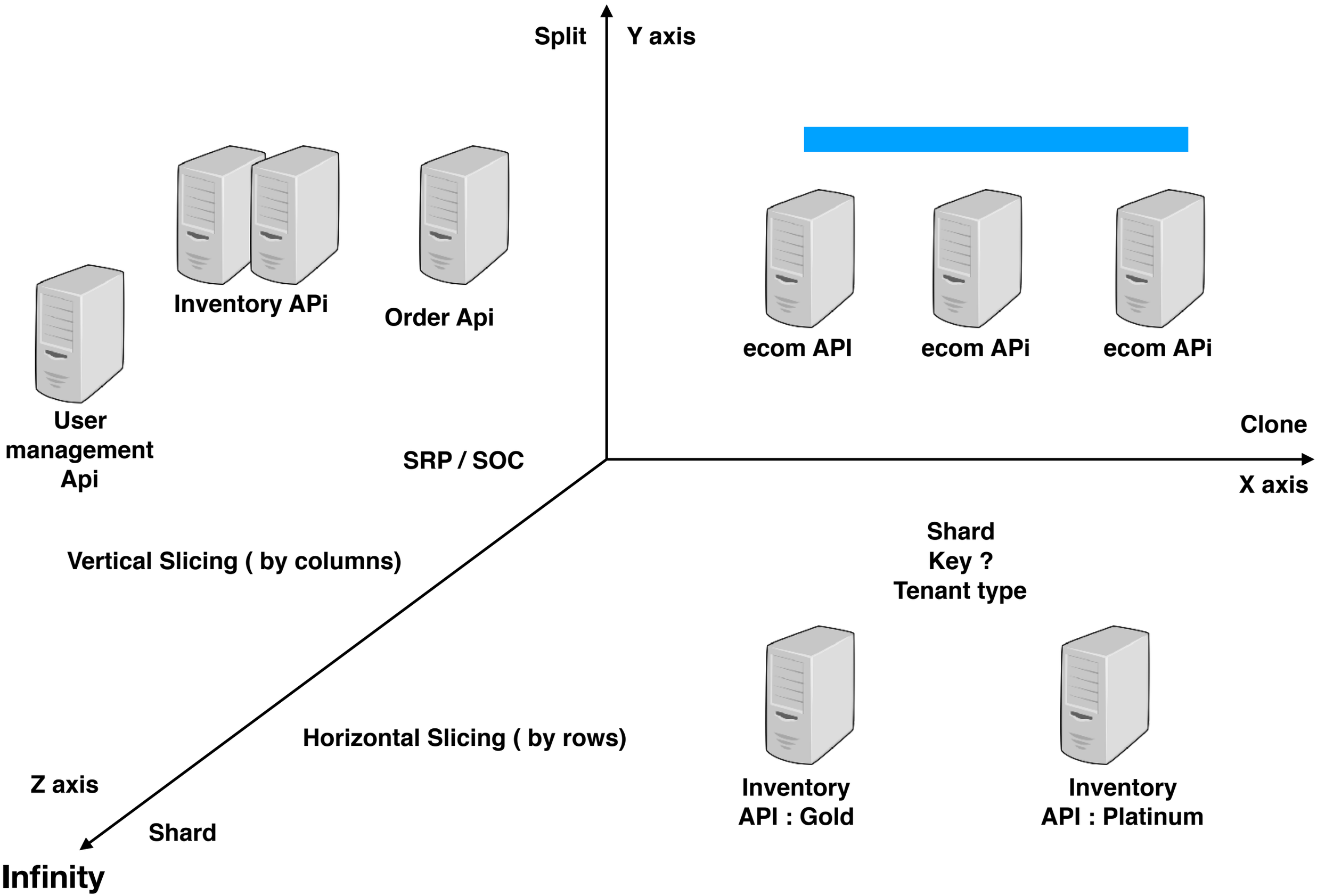
2

Scalability Cube - 50 rules for high Scalability

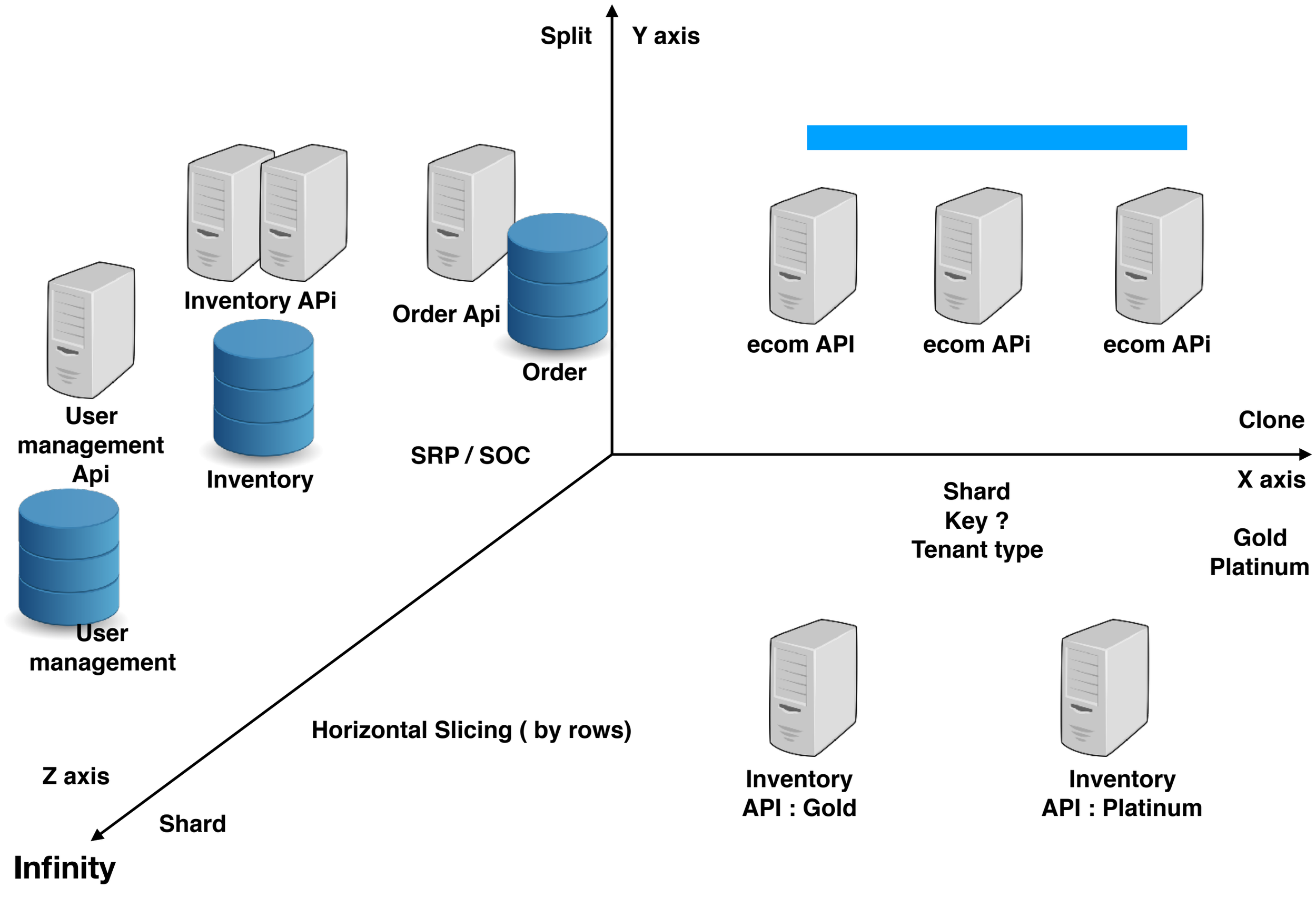
ACID - transaction
Query / reporting/ dash borad
Operations - backup, recovery, ..

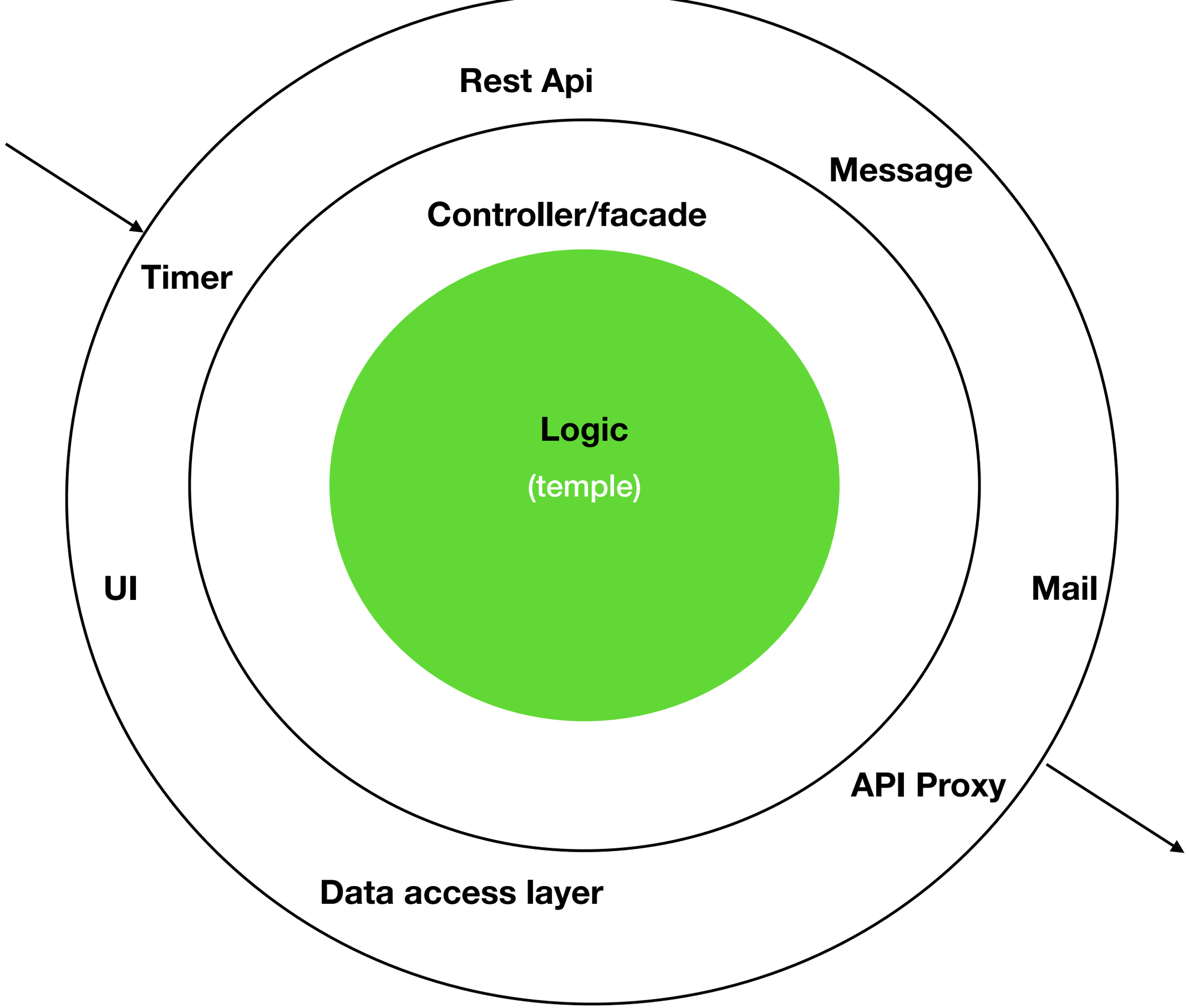


Scalability Cube - 50 rules for high Scalability



Scalability Cube - 50 rules for high Scalability




Rest Api
Message
Controller/facade
Timer
Logic
(temple)
UI
Mail
API Proxy
Data access layer

Boundary

UI

Update order

Schedule

Rest Api

Message

Delegate

Control Flow

Commons

Data Access Layer

API Proxy

Domain Logic

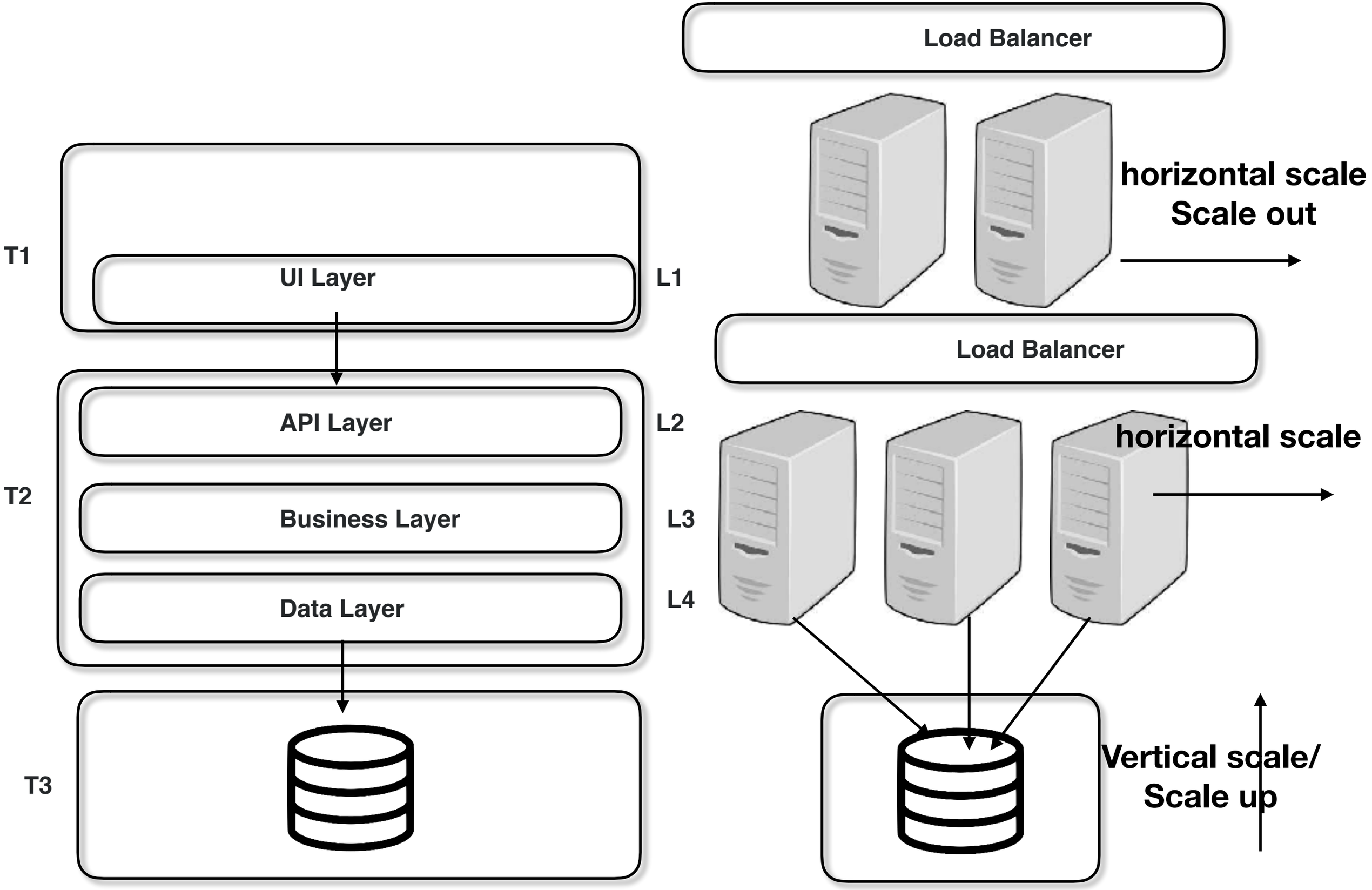
<<POJO>>

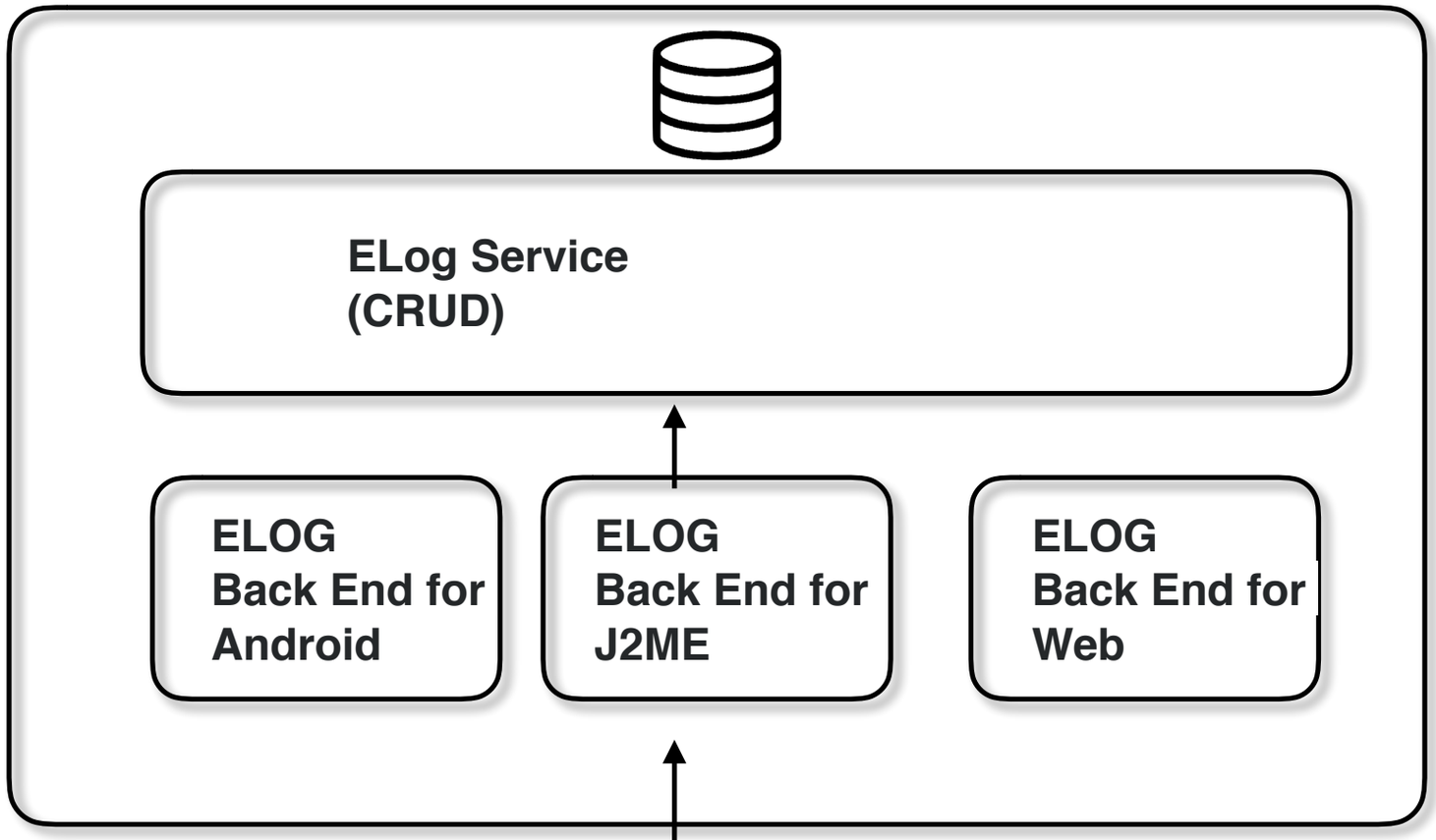
Domain Flow
Steps (actions)
Domain rules

Entity

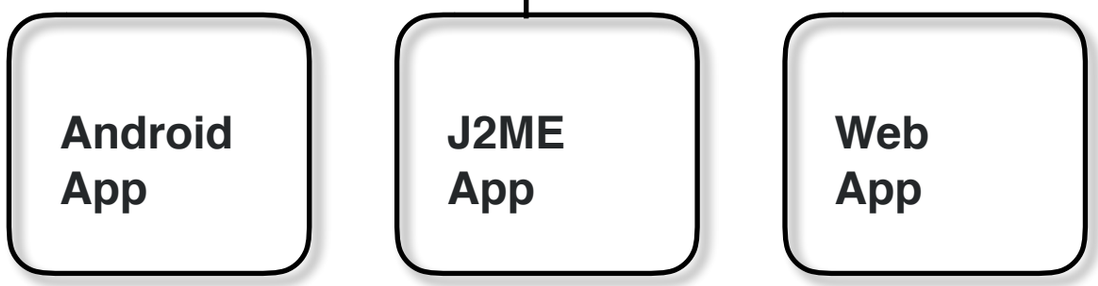
Hexagonal

Boundary

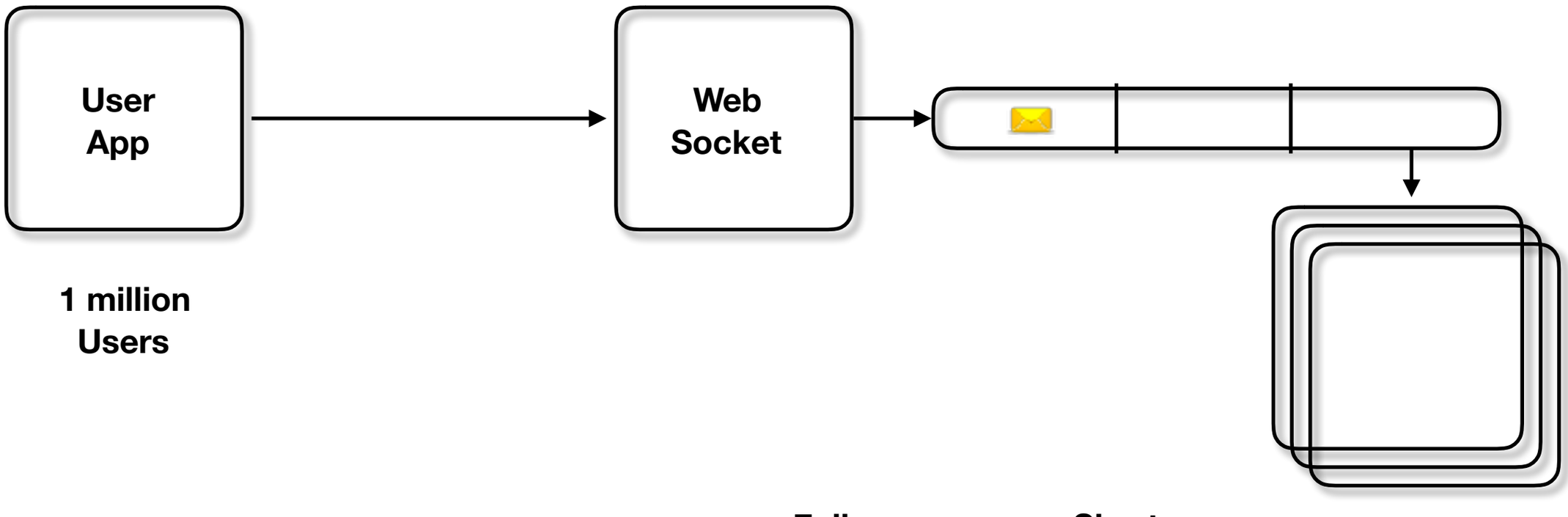

Load Balancer
horizontal scale
Scale out
T1
UI Layer
L1
Load Balancer
API Layer
L2
horizontal scale
T2
Business Layer
L3
Data Layer
L4
T3
Vertical scale/
Scale up



Compress
Minify
Cache



Patient



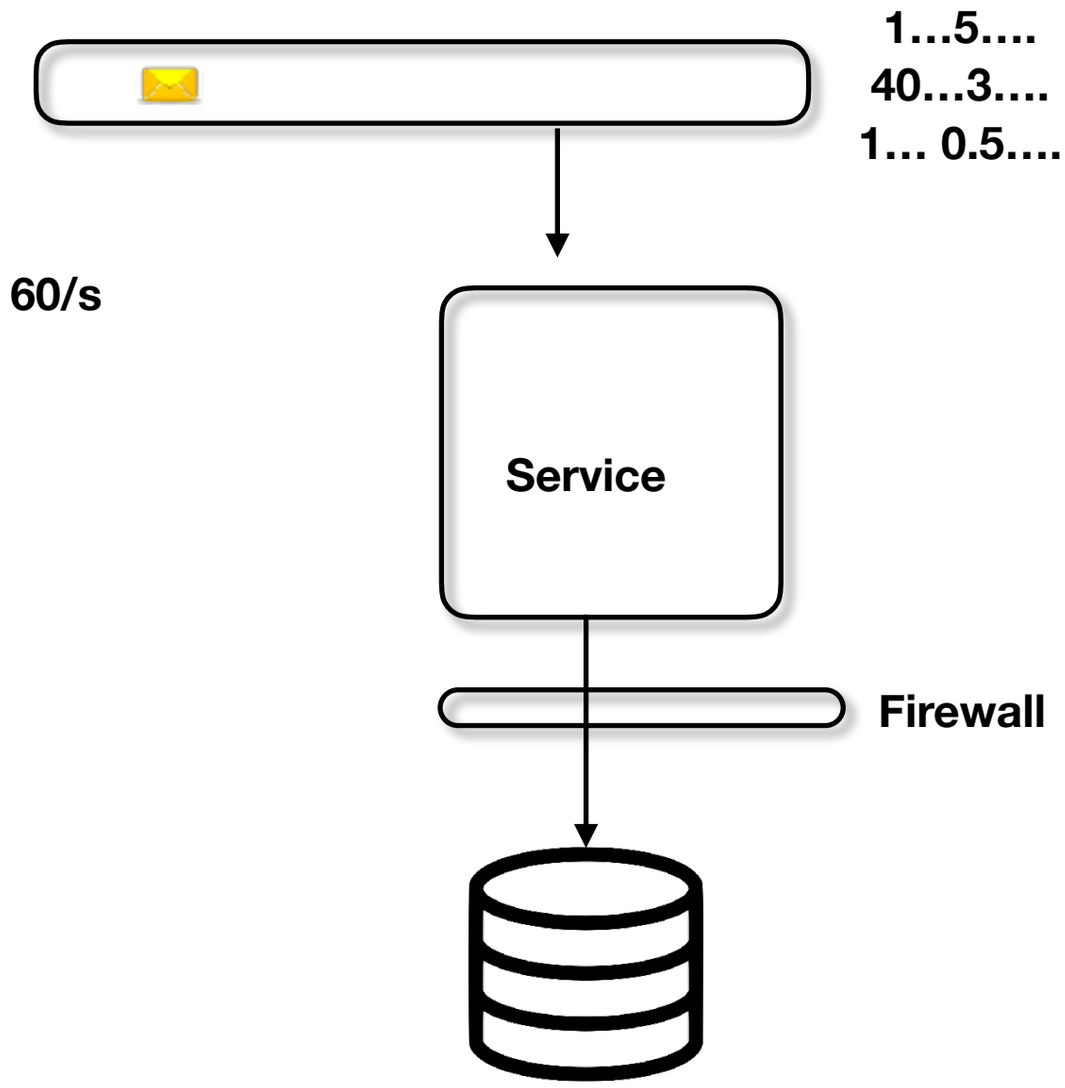
1 million
Users

Full
Batch
Analytics

Short
Batch
Analytics

Stream
Analytics

U1d1,u1d2, u1d3,…..


1…5….
40…3….
1… 0.5….
60/s
Service
Firewall

Web Extension

500

Web Socket

Sync Service

Html
Image Screen Shot
data

Event
user id
device id
time stamp
sequence_id

2 -5 seconds

Web Socket

Ingress Service

OCR

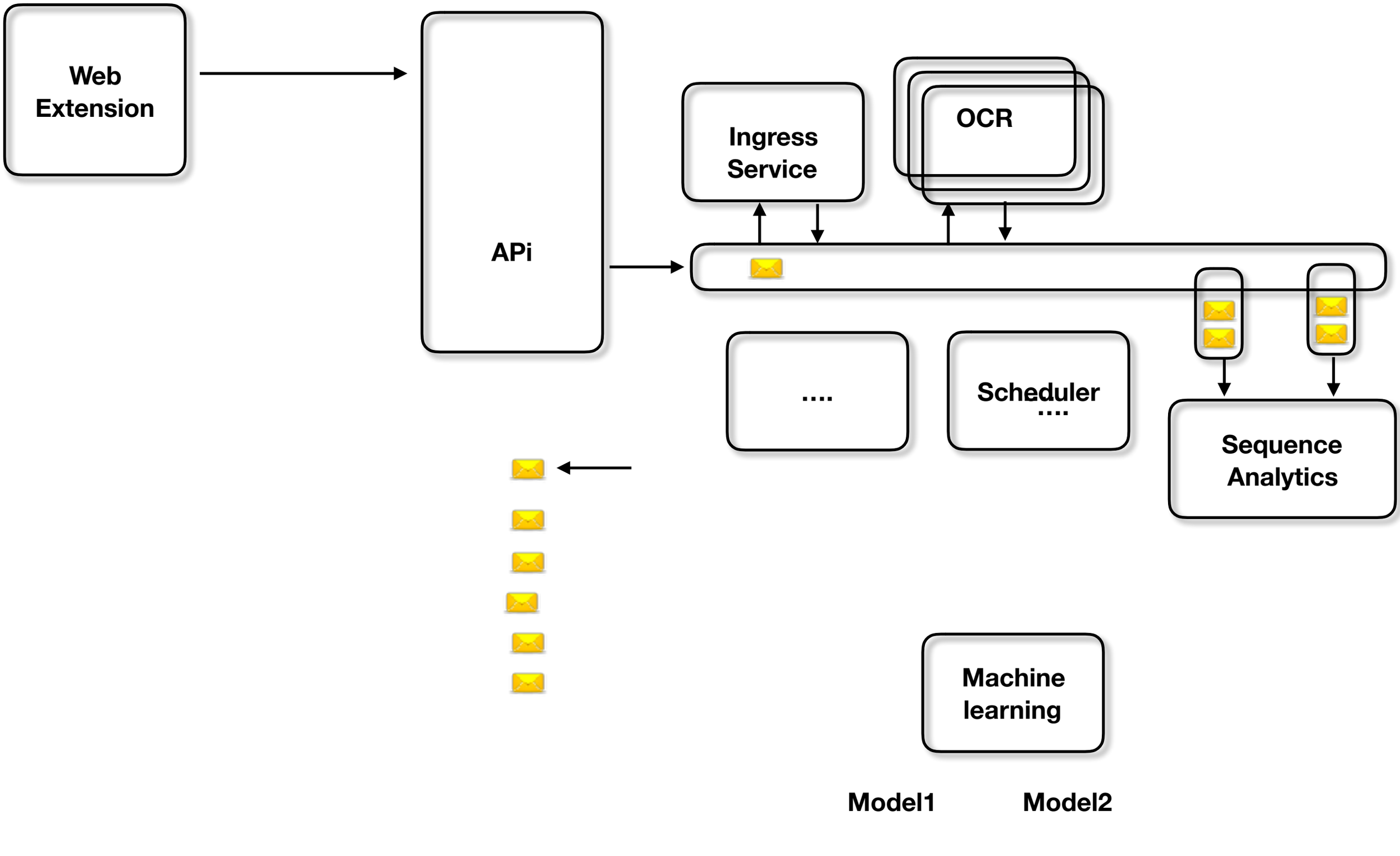
....

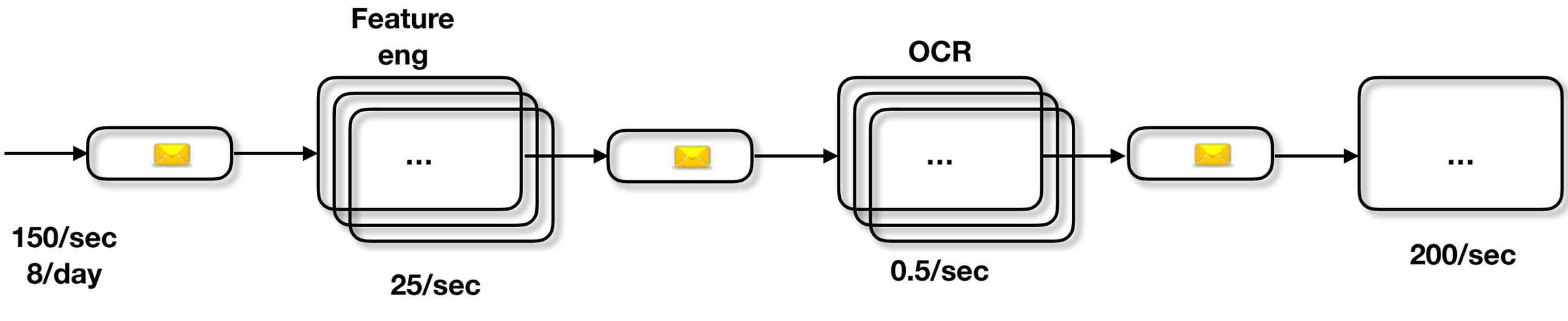
Scheduler

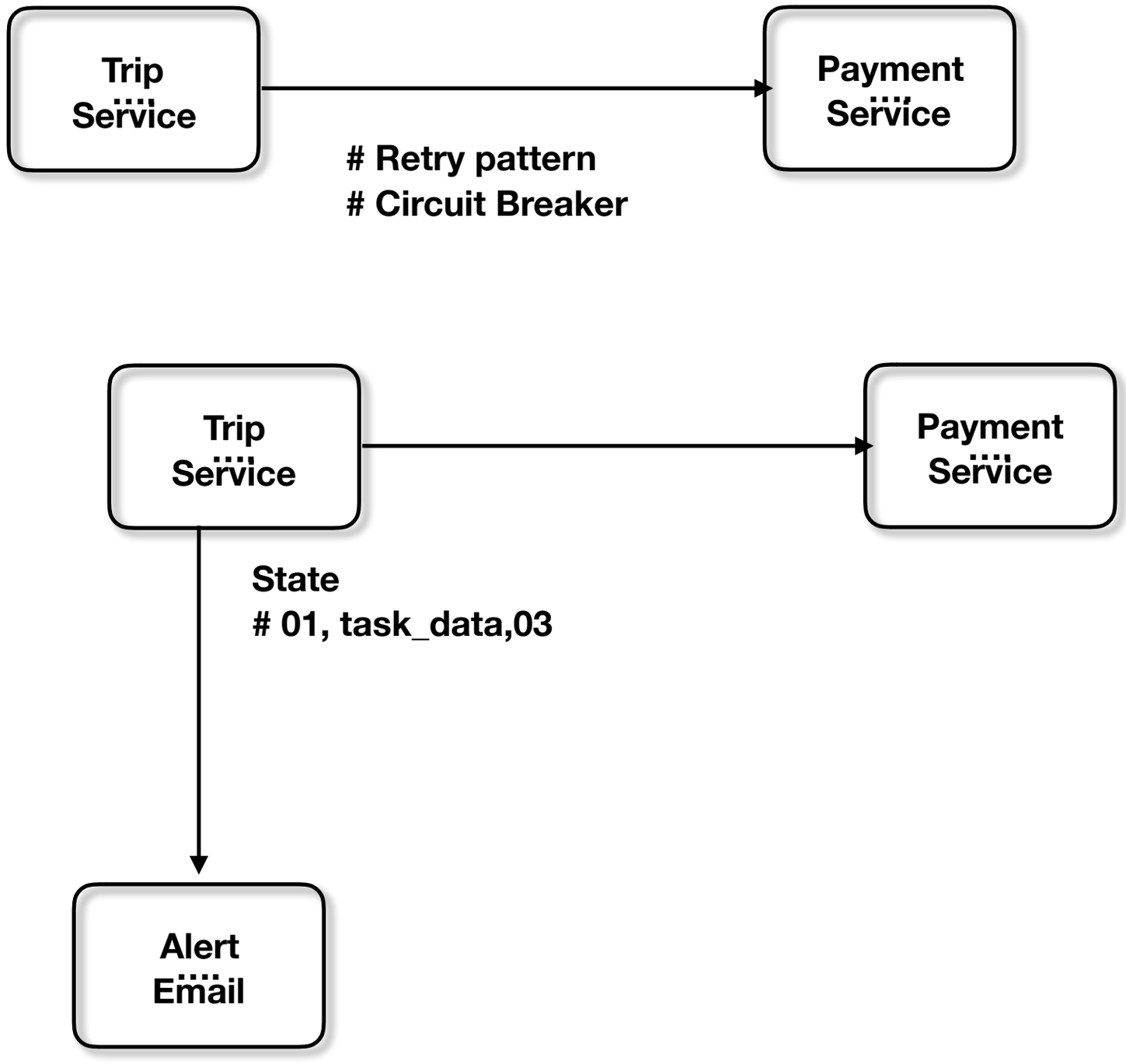
Sequence Analytics

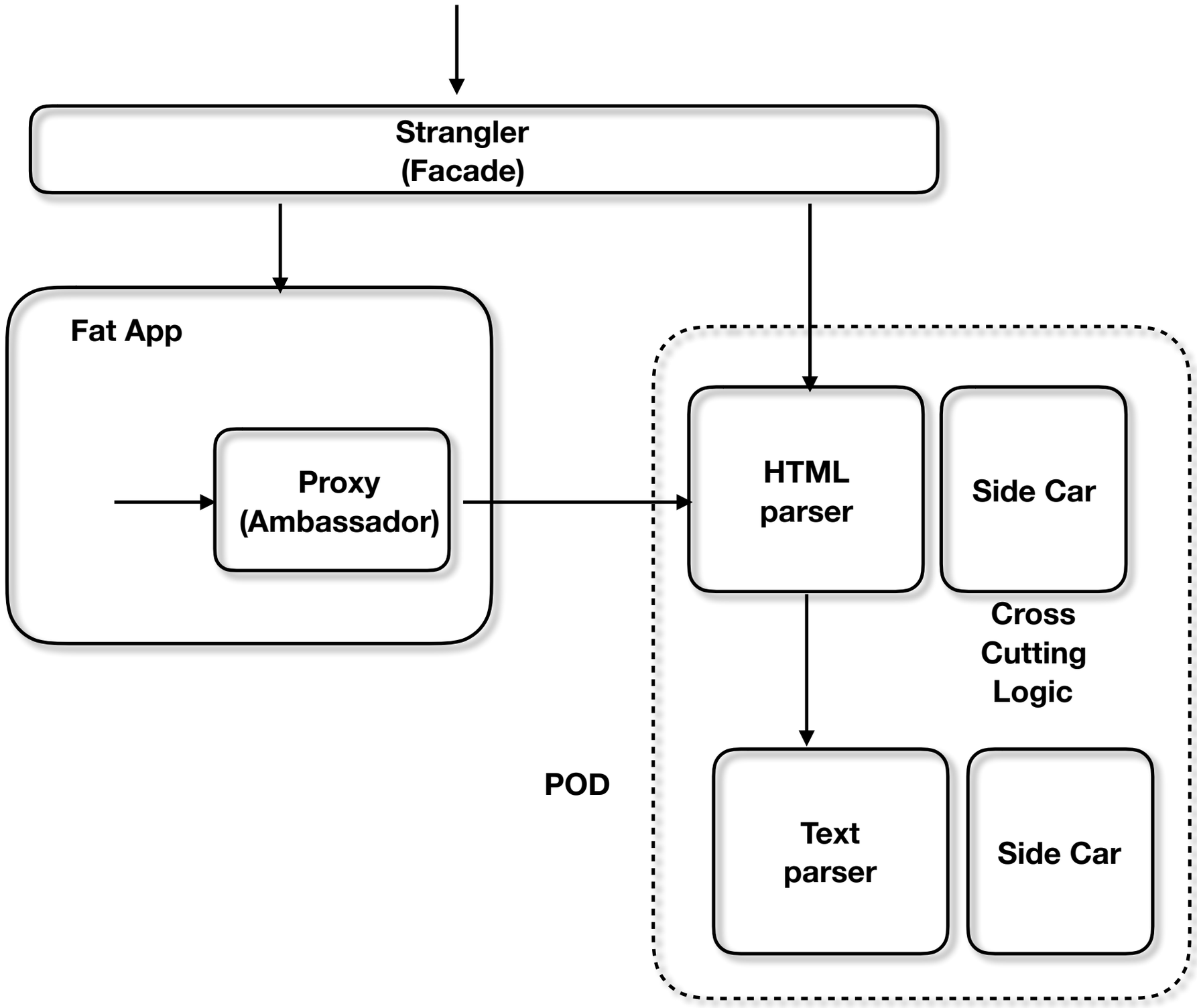
Multiple users
unordered

* Buffer
U1: e1,e3
U2 : e1,e2


Web
Extension
APi
Ingress
Service
OCR
....
Scheduler
....
Sequence
Analytics
Machine
learning
Model1
Model2


Feature
eng
OCR
...
...
...
150/sec
8/day
25/sec
0.5/sec
200/sec


Trip Service
Payment Service
Retry pattern
Circuit Breaker
Trip Service
Payment Service
State
01, task_data,03
Alert Email


Strangler
(Facade)
Fat App
Proxy
(Ambassador)
HTML
parser
Side Car
Cross
Cutting
Logic
POD
Text
parser
Side Car

172.17.0.98

Load balancer port

Node port

Cluster

19.12.24.3 :4000

Node

4001

19.12.24.4 : 4000

Node

4001

10.104.68.18
->10.104.68.21
->10.104.68.20
->10.104.68.19

10.104.68.97
-> 10.104.68.26

Cluster IP

10.104.68.18

10.104.68.97

Service for Service A

10.104.68.21

Pod

Container

Service A:v1

10.104.68.20

Pod

Container

Service A:v2

10.104.68.19

Pod

Container

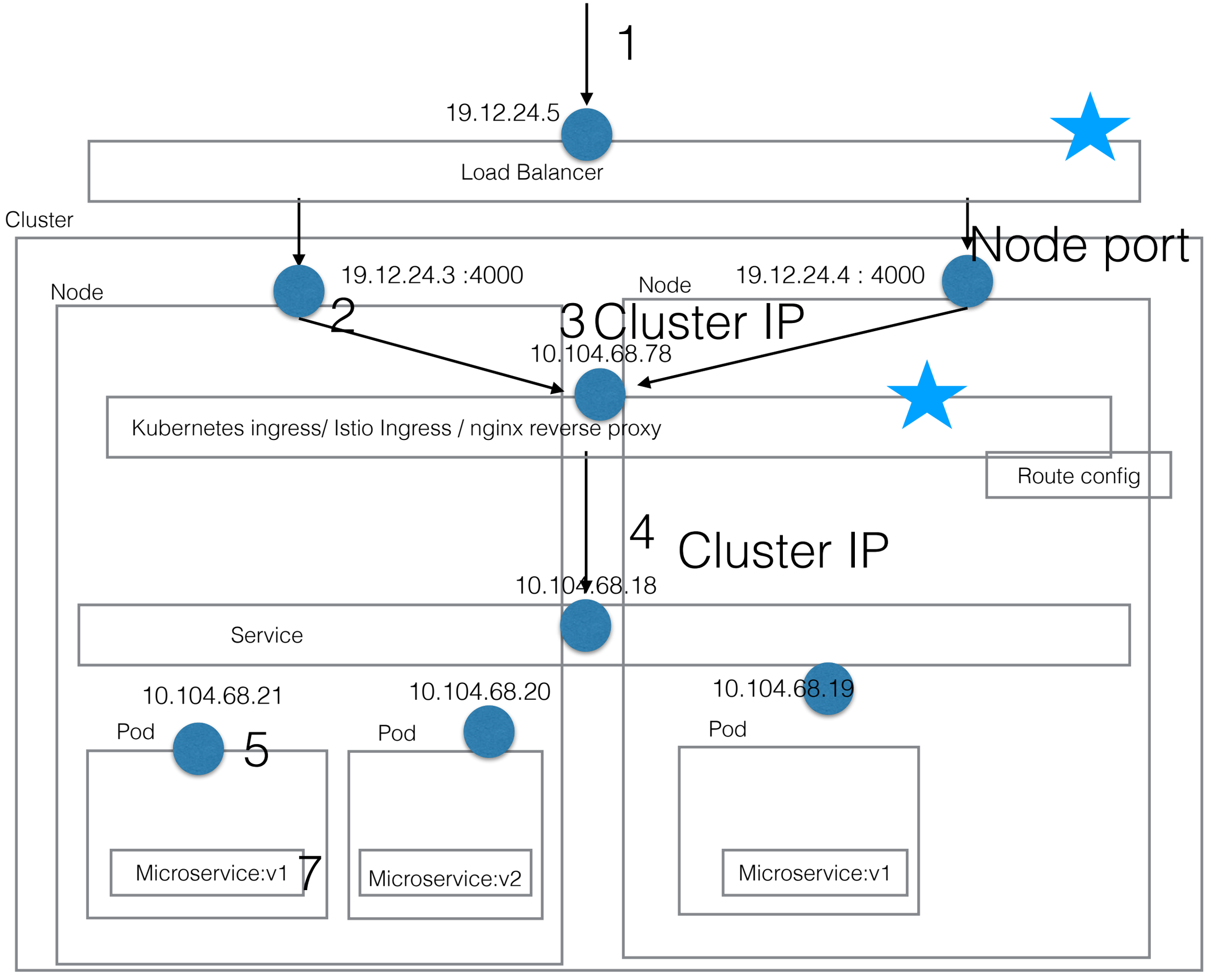
Service A:v1

10.104.68.26

Pod

Container

Service B :v1


1
19.12.24.5
Load Balancer
Cluster
Node port
19.12.24.3 :4000
Node
2
Node
19.12.24.4 : 4000
3 Cluster IP
10.104.68.78
Kubernetes ingress/ Istio Ingress / nginx reverse proxy
Route config
4
Cluster IP
10.104.68.18
Service
10.104.68.21
Pod
5
10.104.68.20
Pod
10.104.68.19
Pod
Microservice:v1
7
Microservice:v2
Microservice:v1

1 API Gateway

19.12.24.5

2 Load Balancer

Cluster

Node port

19.12.24.3 :4000

19.12.24.4 : 4000

Node

Node

Cluster IP

10.104.68.78

3 Kubernetes ingress/ Istio Ingress / nginx reverse proxy

Route config

Cluster IP

10.104.68.18

Service

10.104.68.21

10.104.68.20

10.104.68.19

Pod

Pod

Pod

4

Microservice:v1

Microservice:v2

Microservice:v1

1 API Gateway

Traffic Management
resiliency
security
debugging
throttling

19.12.24.5

2 Load Balancer

Cluster

Node port

19.12.24.3 :4000

Node

19.12.24.4 : 4000

Node

10.104.68.78

3 Kubernetes ingress/ Istio Ingress / nginx reverse proxy

Route config

10.104.68.18

Service

10.104.68.21

Pod

10.104.68.20

Pod

10.104.68.19

Pod

4

Side Car for Ambassador

Microservice:v1

Microservice:v2

Microservice:v1

Traffic Manager (Geode)

East-Us

DMZ

aggregation
offloading
Gatekeeper
routing
- Canary
- Blue green
- Versioning
- AB
- Shadowing

API Management

API Gateway (L7)
Http Traffic

WAF

Metering per tenant

Load Balancer (L4)
Non Traffic

reverse proxy

Traffic Management
...

North South
Traffic

East west
Traffic

Service A

Service

Service Mesh
(Side Car)

Service B

1 API Gateway

2 WAF

19.12.24.5

3 Load Balancer

Cluster

Node port

4

19.12.24.3 :30727

Node

19.12.24.4 : 30727

Node

10.104.68.78

Cluster IP

Kubernetes ingress/ Istio Ingress / nginx reverse proxy

5

Cluster IP

10.104.68.18

Service 6

10.104.68.21

Pod 7

Side Car 8

Microservice:v1 7

10.104.68.20

Pod

Side Car

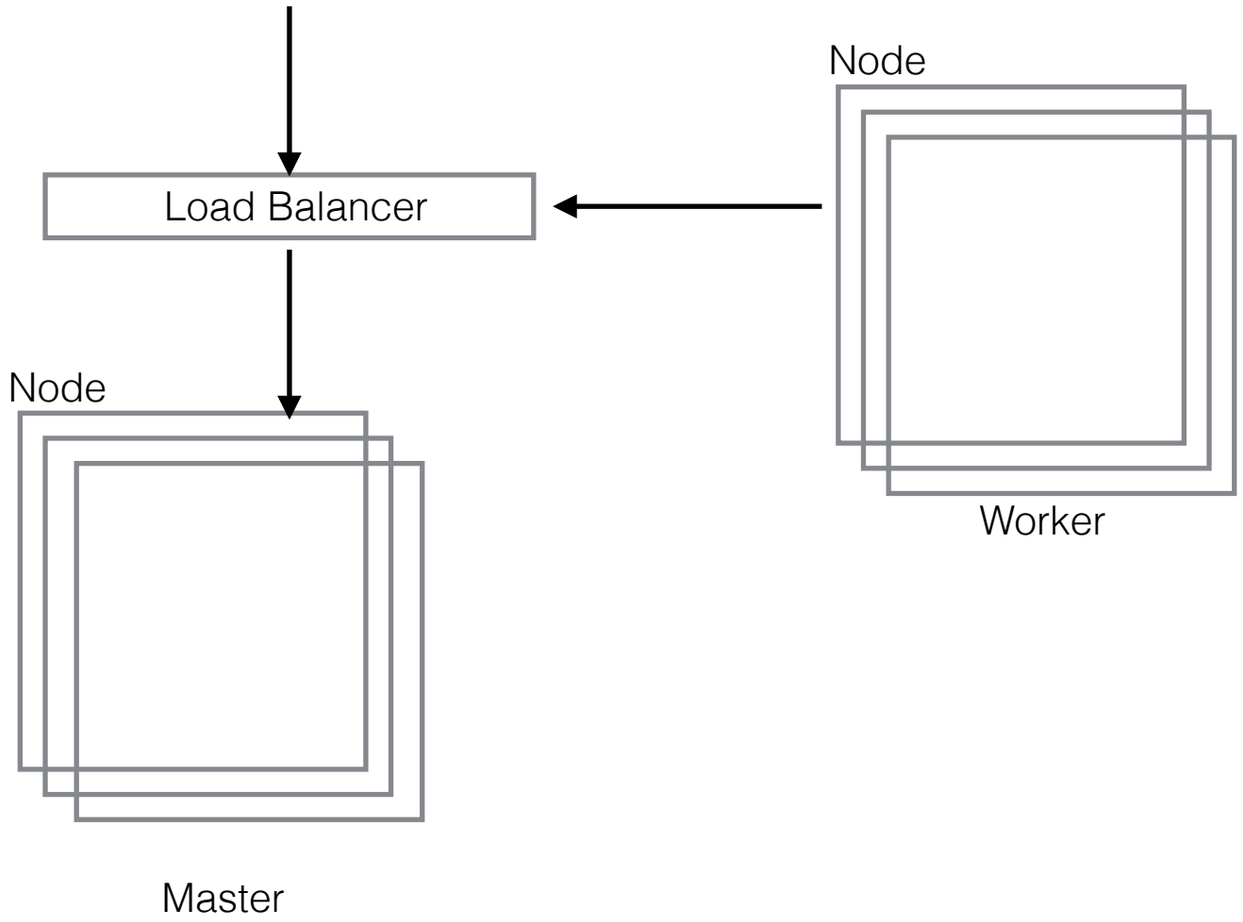
Microservice:v2

10.104.68.19

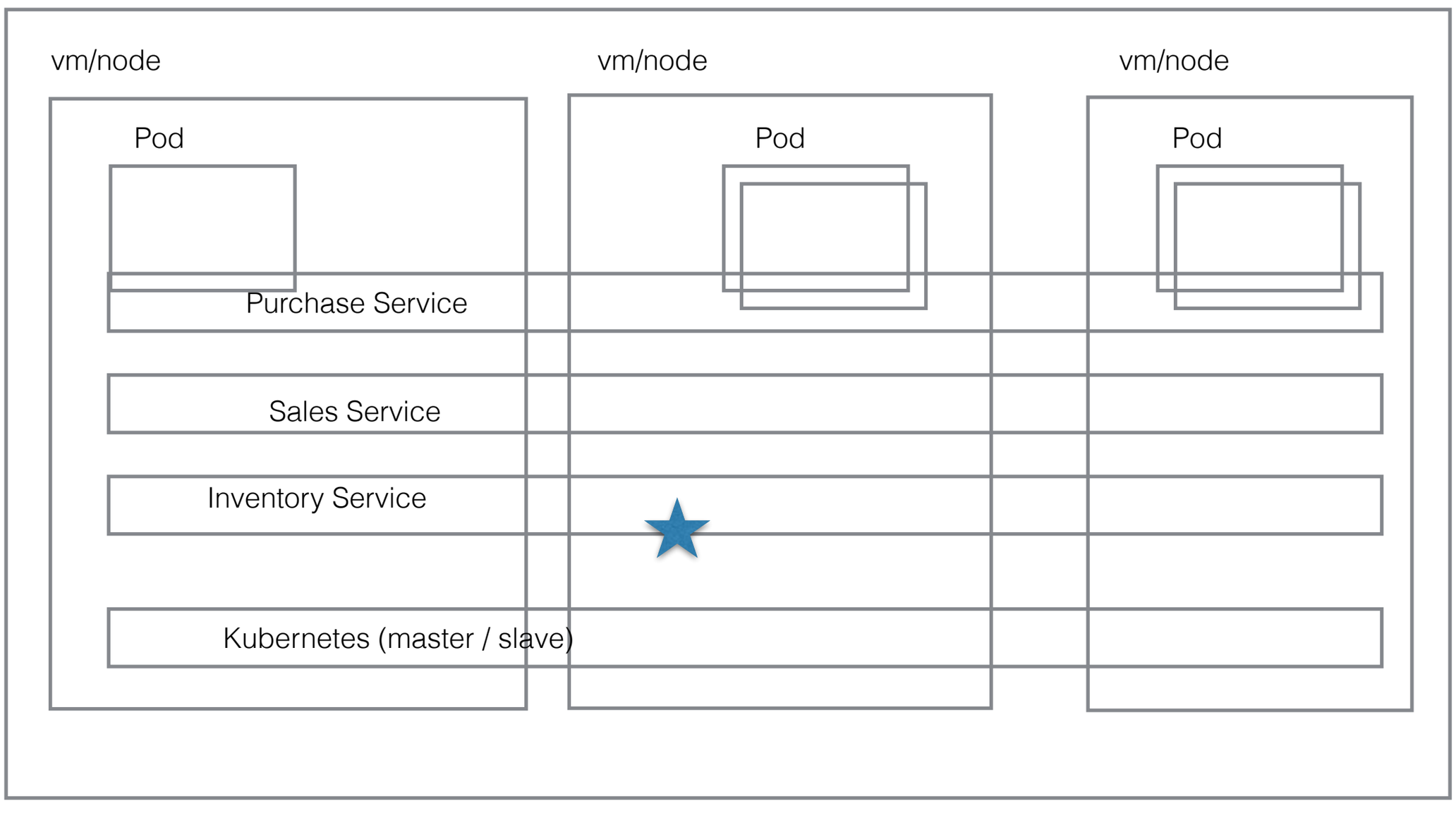
Pod

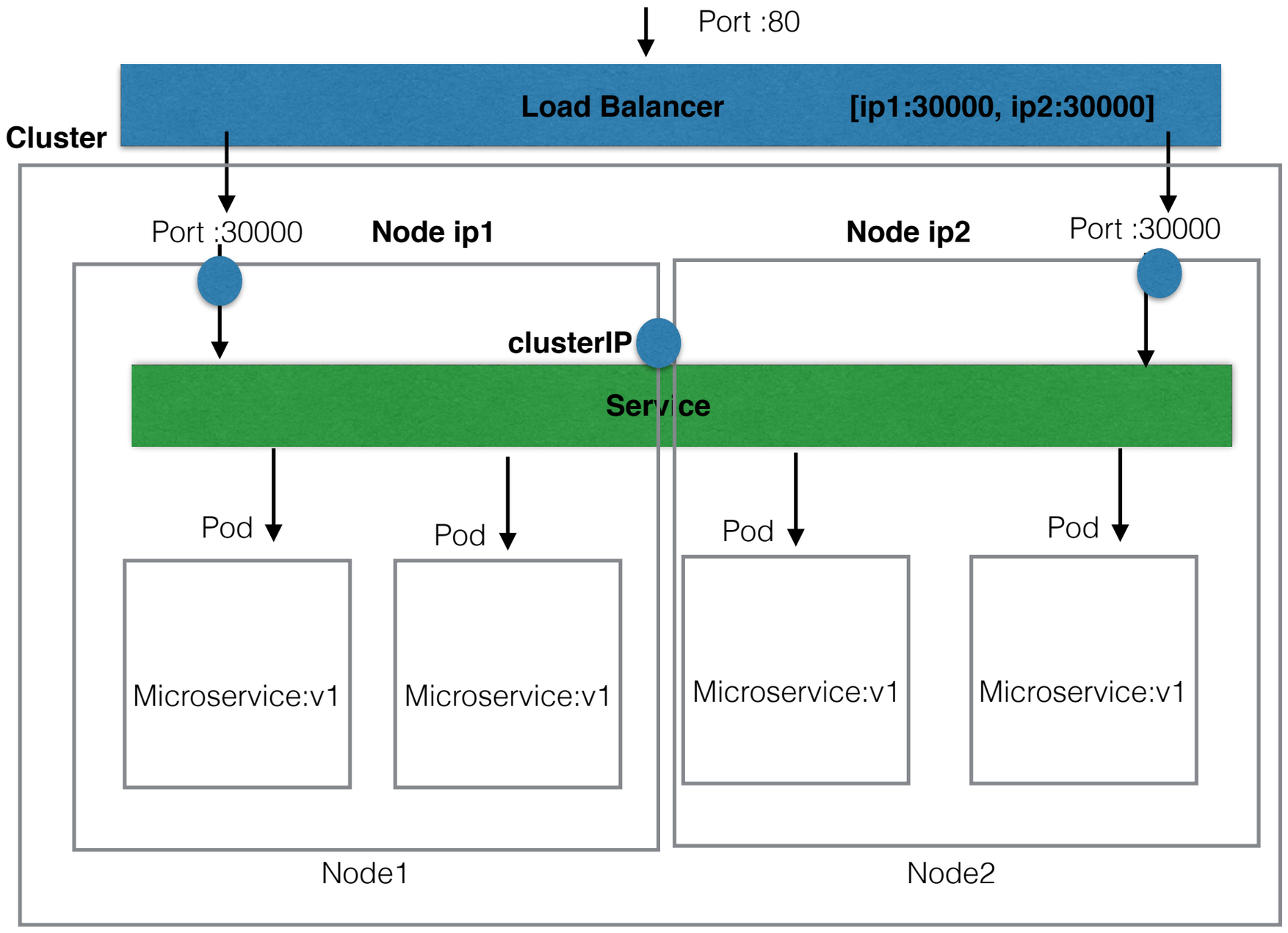
Side Car

Microservice:v1

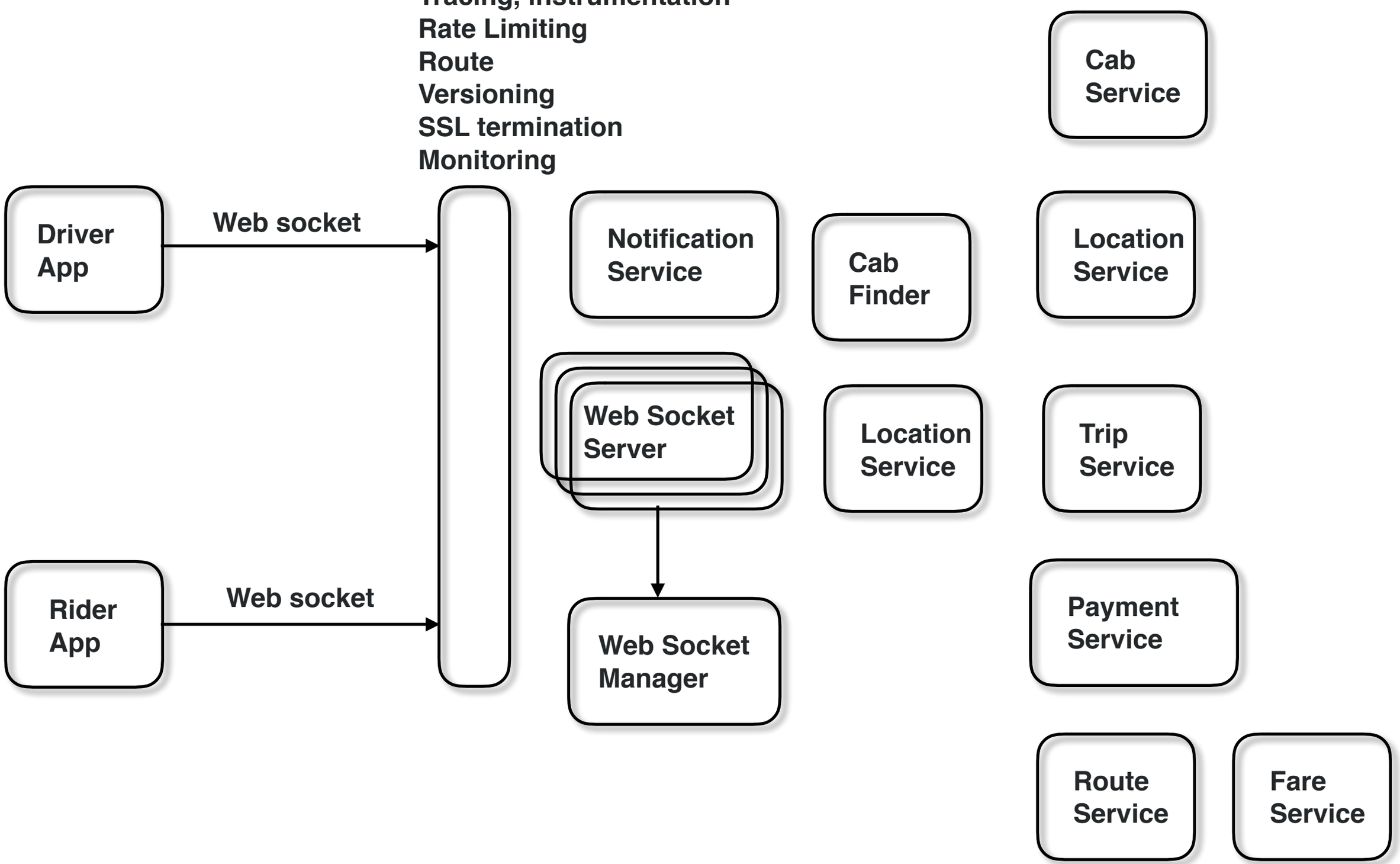

Node
Load Balancer
Node
Worker
Master

Cluster

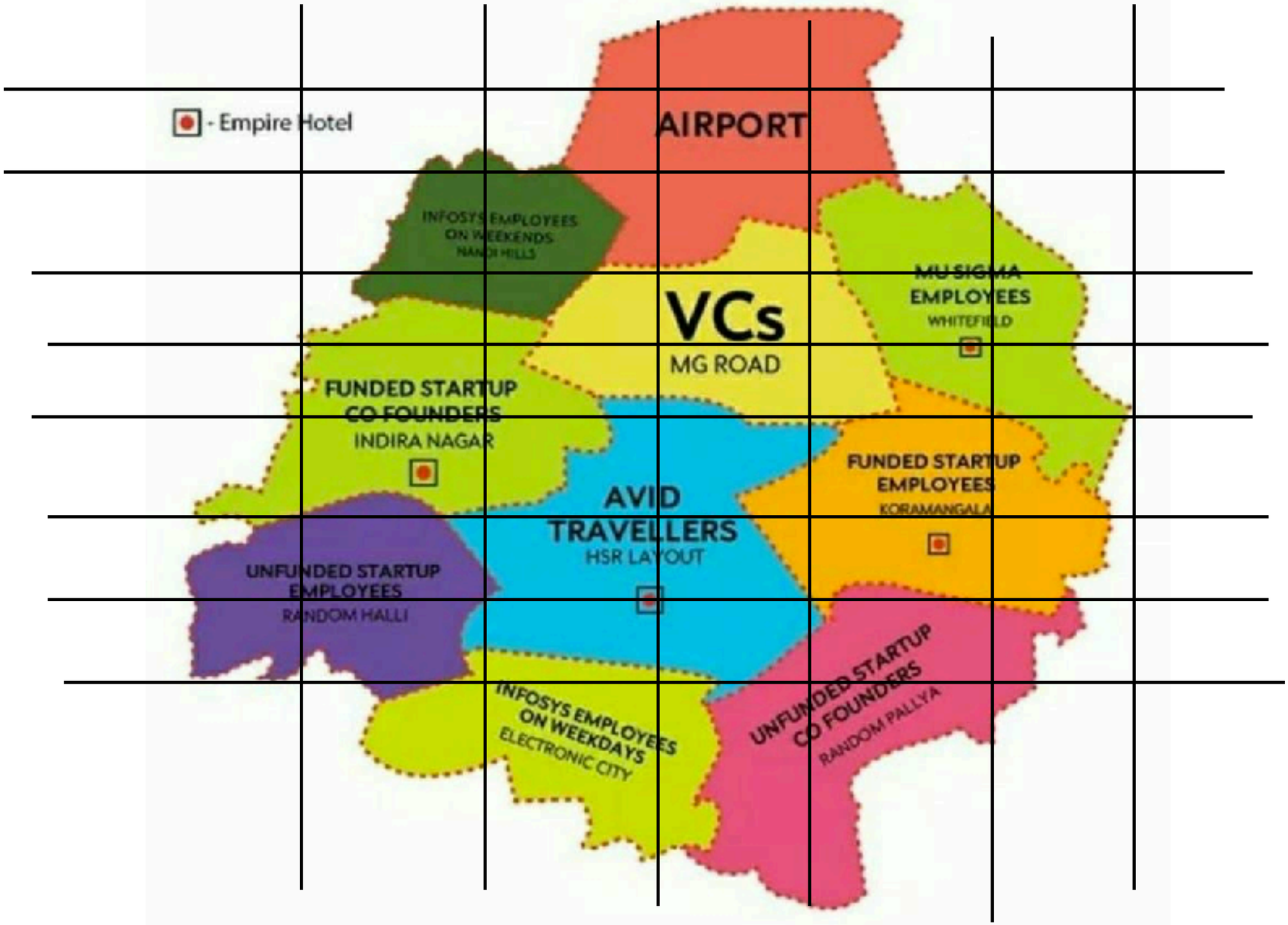

vm/node
vm/node
vm/node
Pod
Pod
Pod
Purchase Service
Sales Service
Inventory Service
Kubernetes (master / slave)

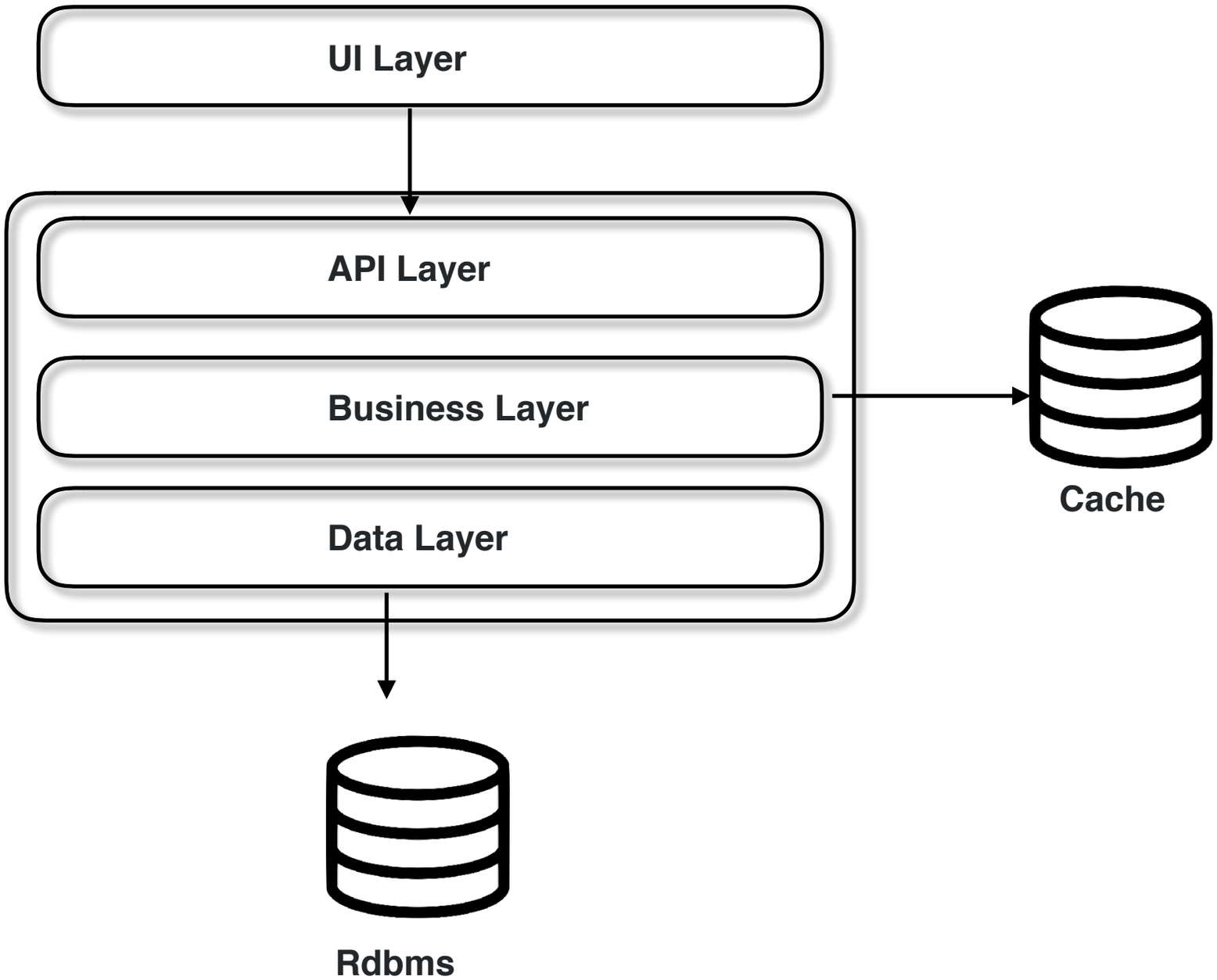

Port :80
Load Balancer
[ip1:30000, ip2:30000]
Cluster
Port :30000
Node ip1
Node ip2
Port :30000
clusterIP
Service
Pod
Pod
Pod
Pod
Microservice:v1
Microservice:v1
Microservice:v1
Microservice:v1
Node1
Node2

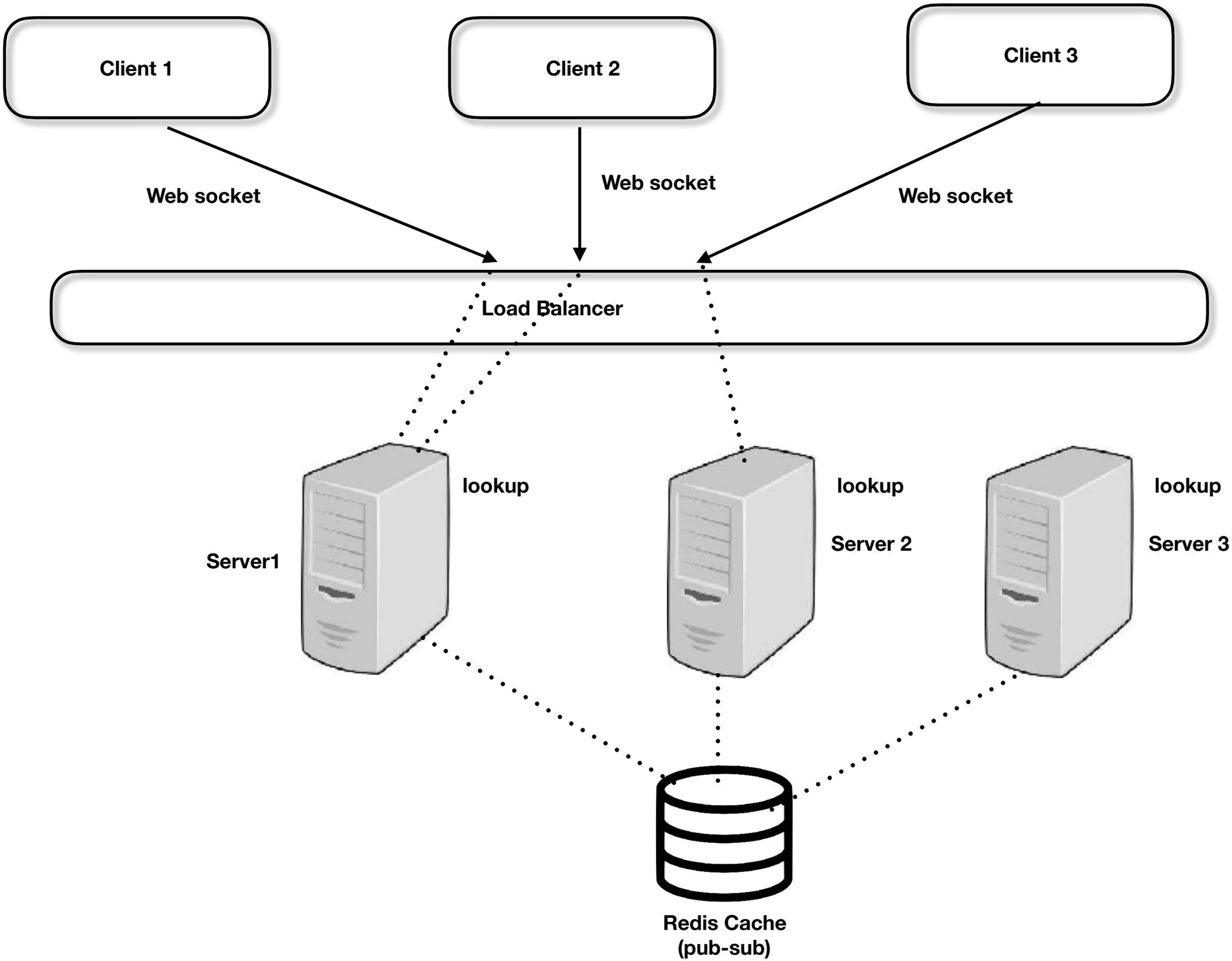
Authentication
Authorization
Transport Security
Tracing, instrumentation
Rate Limiting
Route
Versioning
SSL termination
Monitoring

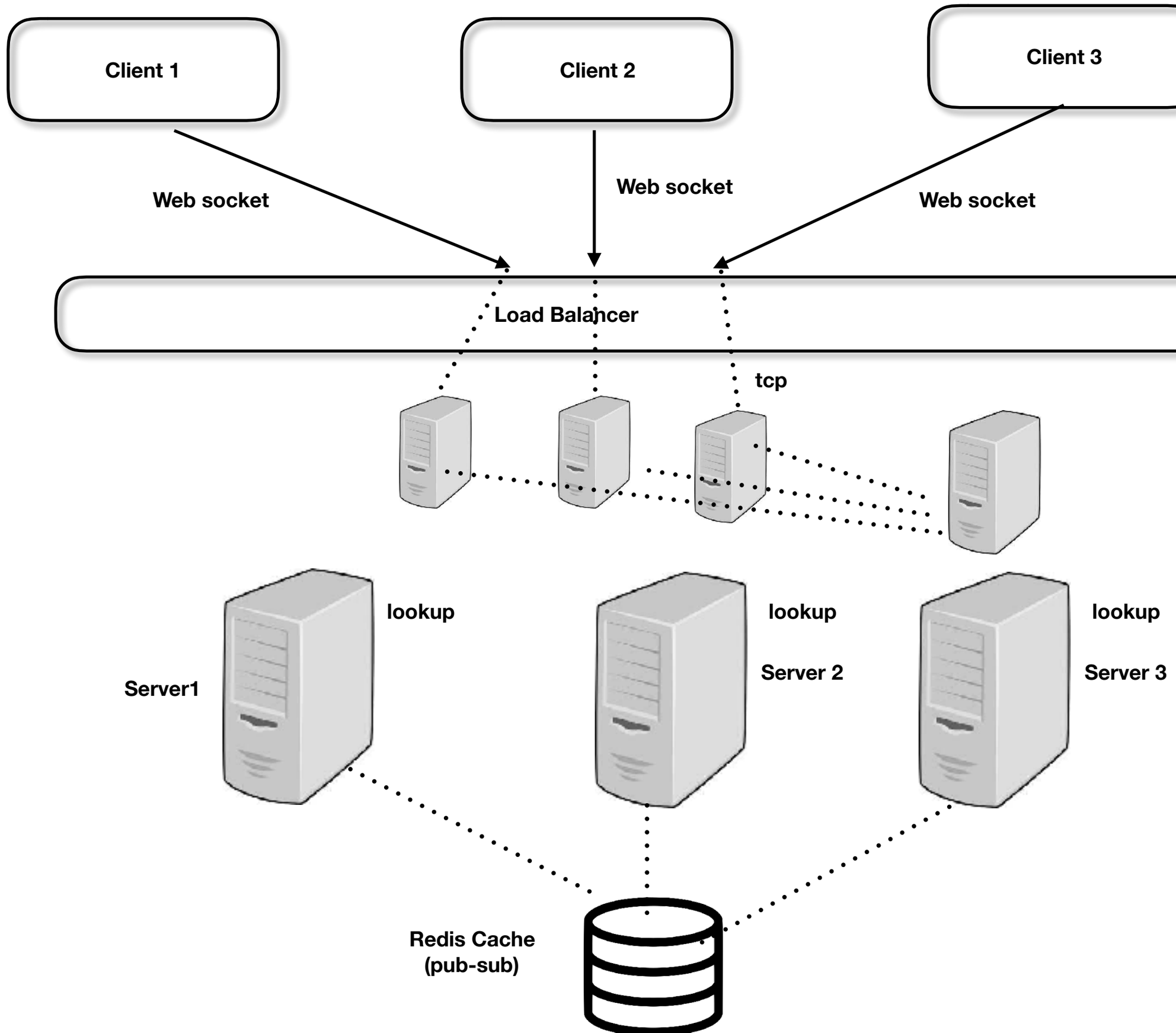


BANGALORE MAP




UI Layer
API Layer
Business Layer
Data Layer
Cache
Rdbms


Client 1
Client 2
Client 3
Web socket
Web socket
Web socket
Load Balancer
lookup
lookup
lookup
Server1
Server 2
Server 3
Redis Cache
(pub-sub)


Client 1
Client 2
Client 3
Web socket
Web socket
Web socket
Load Balancer
tcp
Server1
lookup
lookup
Server 2
lookup
Server 3
Redis Cache
(pub-sub)

File Storage
(object storage)

Sensor

Gateway

ws

Cloud
Ingress

Enricher 1

Enricher 2

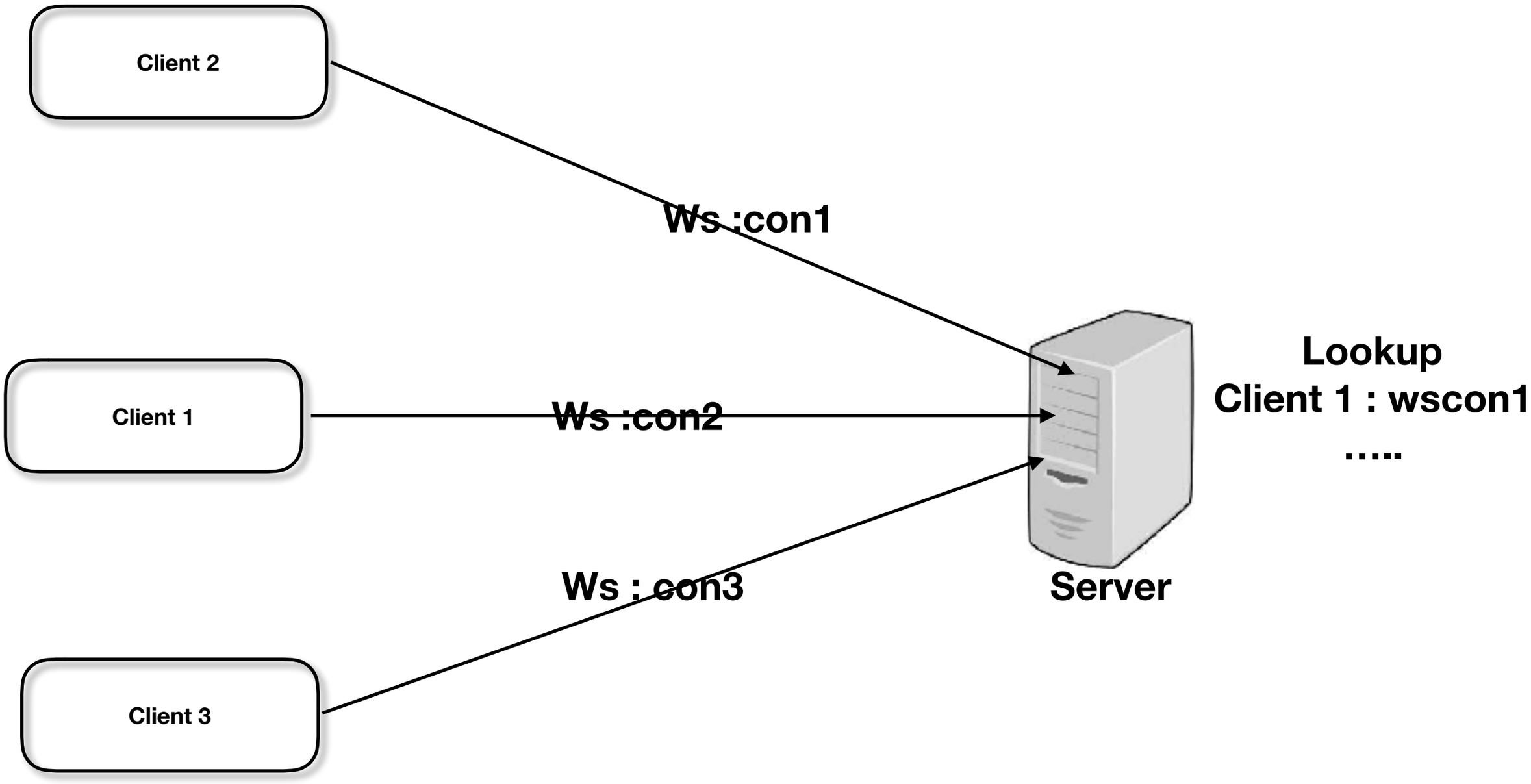
Enricher 3

Egress

Reporting tools

DWH

Spark
Analytics


Client 2
Ws :con1
Client 1
Ws :con2
Lookup
Client 1 : wscon1
…..
Ws : con3
Server
Client 3

Patterns

Scalability Patterns

1. Competing Consumer
2. Queue based load leveling
3. Sharding
4. Claim Check
5. CQRS
6. Geode
7. Deployment stamps

Performance Patterns

1. Async Request Reply (queue + key store)
2. Static Content Hosting
3. Materialized View
4. Backends for front ends
5. Cache Aside
6. Index Table (secondary index, key value store)
7. Throttling (queue, config)
8. Rate Limiting (queue, API management)

Centralize Cross Cutting Concerns

1. Gateway Aggregation(NGINX service using Lua)
2. Gateway Routing (Application gateway, Azure front door)
3. Gateway Offloading (Nginx, Application Gateway, Azure Load Balancer, API management)
4. Gatekeeper {facade} (Application gateway, API management, WAF)
5. Side Car
6. Ambassador
7. Federated Identity
8. External Config store

Legacy modernisation Patterns

1. Strangler
2. Anti Corruption Layer

Reliability / Availability Patterns

1. Retry
2. Circuit breaker
3. Bulk head
4. Compensatable transaction
5. SAGA
6. Schedular Agent supervisor
7. Health endpoint

Communication Patterns

1. Pub sub
2. Priority Queue
3. Event Sourcing
4. Choreography/ Orchestration
5. Leader Election
6. Sequential convoy