MONOLITH

COOL DEV

Let's all give it up for Monolith, who is about to retire. I'd also like to take this opportunity to proudly introduce...

CLAP CLAP CLAP CLAP CLAP CLAP

GET OUT, CHUBBY!

THE MICROSERVICES

COOL DEV

CLAP CLAP CLAP CLAP HUHUUUU CLAP CLAP CLAP CLAP CLAP YEAH CLAP CLAP YEAH HUHUUUU YEAH

Gimme five, buddy.

Talk to my API gateway please...

I'm not sharing a database with you!

Well, they might need some ORCHESTRATION first.

MONOLITH

COOL DEV

Daniel Stori {turnoff.us}

» Skan.ai - chief Architect

» Ai.robotics - chief Architect

» Genpact - solution Architect

» Welldoc - chief Architect

» Microsoft

» Mercedes

» Siemens

» Honeywell

Mubarak

- How to build Microservice
- Patterns
- Anti Patterns
- Hands on Demos
- Case Studies

> Your Technology stack
> Total Years of experience
> what do you know about microservice ?
> Docker ?
> Kubernetes ?

	Application Decomposed into Traditional Distributed Modules	**Application Decomposed into Micro Distributed Application**	**W**	**Score**
Database / Storage	Shared	Not shared *****	2	
Infra (Hosting)	Shared	Not Shared	3	
Sorce Control	Shared	Not Shared *	2	
CI/CD (Build Server)	Shared	Not Shared	3	
Fun Requirements	Shared	Not Shared *	1	
SCRUM Team / Sprint	Shared	Not Shared *	1	
Test Cases	Shared	Not Shared	1	
Architecture	Shared	Not Shared	1	
Technology Stack / Fwks	Shared	Not Shared	1	

	Pros/ Cons	Solution
Development time	- -	
Learning Curve	- -	
Resource Performance (CPU, Memory, I/O)	- -	grpc, thrift, protnbuf
Db Transaction Management	- -	SAGA
Views / Report / Dash board/ join	- -	Materialised View
Infra Cost	- -	Container
Initial Deployment effort	- -	Automation (IAC)
Debugging, Error Handling,	-	Distributed Tracing (e2e flow)
Integration Test	- -	
debug/ error Log Mgmt	- -	Centralize (EFK)
Config Mgmt	- -	Centralize
Authentication	- -	Centralize
Authorization	- -	Centralize
Audit Log mgmt	- -	Centralize
Monitoring / Alerting	- -	Centralize *
Data Security and Privacy	- -	
Build Pipeline (CI)	- -	Automation
Agile Architecture (Agility to change)		
Feature Shipping (Agility to ship)/ CD		
Scalability (volume - request, data,		
Resilence		
Availability		
Ability to do Polygot		
Maintainability (easy to change)		

CQRS
Materialised View
SAGA - compenstable transaction
Eventual Consistency
Event Driven Architecture
DDD
Bounded context
Aggregate
Event Sourcing
ubiquitous language
Command Message
Event Message

Log mngmt
config mngmt
devops

Portal

HTTPs REST + JSON (CRUD)

WAF

Reverse Proxy/ API Gateway

Authentication,
Authorization

Stream
Analytics

do() undo()

Stock
Management
Command

No Stocks

Create
Order

Create
Order
Msg (CUD)

Grpc + protnbuf (R)

Domain Event Bus (command / Events)

Order Created

No Stocks

Create
Order

Order
Management
Command

do() undo()

Create
Order

Order
Created

No
Stocks

Order
Management
Query

do() undo()

Event Service

Data lake
Event Store
Audit
full state

Big Data
Analytics

Data Ware House
snapshot (day/week)

DWH Analytics

weekly/ monthly

RDBMS
3NF
no duplicates, more joins
Write Friendly
not read friendly
current state

Document
DNF
duplicates, no joins
not Write Friendly
read friendly

PUBLIC NETWORK

PROVIDER CLOUD

ENTERPRISE NETWORK

IoT DEVICES

DEVICES

BROWSER

EDGE SERVICES

Container Environment

MICROSERVICE

MICROSERVICE

MICROSERVICE

MICROSERVICE

MICROSERVICE

TRANSFORMATION & CONNECTIVITY

Integration

API MANAGEMENT

ENTERPRISE USER DIRECTORY

ETL

ENTERPRISE DATA

Application Frameworks

LANGUAGE & RUNTIMES

LIBRARIES & FRAMEWORKS

TRACING & LOGGING

Cloud Services

DATA

AI

SECURITY SERVICES

ANALYTICS

MESSAGING

IoT PLATFORM

Microservices Mesh

CIRCUIT BREAKER

SERVICE DISCOVERY

ROUTING SERVICES

CONFIGURATION MANAGEMENT

MICROSERVICES SECURITY

Container Platform

CONTAINER ORCHESTRATION

CONTAINER CLUSTER MANAGEMENT

IMAGE REPOSITORY

LOAD BALANCING

CONTAINER SECURITY

SERVERLESS COMPUTING

Infrastructure

VIRTUAL INFRASTRUCTURE

STORAGE

NETWORK

PROVISION

INFRASTRUCTURE SECURITY

DevOps

CONTINUOUS PROVISION

CONTINUOUS BUILD

DEPENDENCY MANAGEMENT

CONTINUOUS TESTING

CONTINUOUS DEPLOY

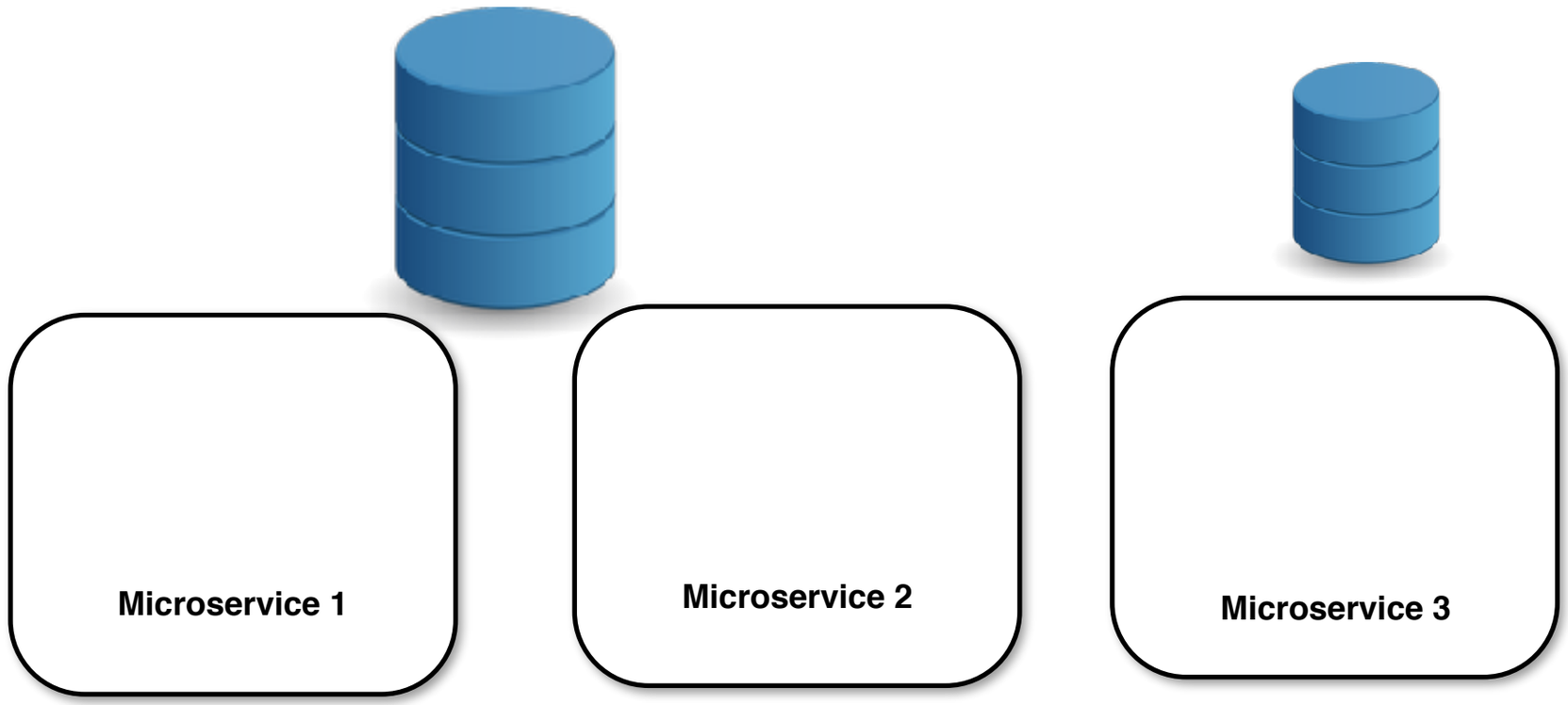
MONITORING & LOGGING

CLOUD MANAGEMENT SERVICES

12 factor App

Codebase	***One codebase per service, tracked in revision control; many deploys.***
Dependencies	***Explicitly declare and isolate dependencies*** • Noncontainerized environments : Chef, Puppet, Ansible,Maven, Gradle,npm • Containerized environment : Dockerfile.
Config	***Store configuration in the environment*** environment variable, application.properties
Backing Services	***Treat backing services as attached resources*** datastores , messaging , SMTP , caching . The current production database could be detached, and the new database attached – all without any code changes.
Build, Release, Run	***Strictly separate build and run stages*** Docker images make it easy to separate the build and run stages.
Processes	***Execute the app in one or more stateless processes*** Store any stateful data, or data that needs to be shared between instances, in a backing service.

Data Isolation	***Each service manages its own data***
Concurrency	***Scale out via the process model*** The Unix process model is largely a predecessor to a true microservices architecture. By leveraging independent deployment feature of microservices, we can individually scale the most needed microservice by using on-demand scaling feature of containers.
Disposability	***Maximize robustness with fast startup and graceful shutdown*** Instances of a service need to be disposable so they can be started, stopped, and redeployed quickly, and with no loss of data.
Dev/Prod Parity	***Keep development, staging, and production as similar as possible***
Logs	***Treat logs as event streams***
Admin Processes	***In a production environment, run administrative and maintenance tasks as a separate microservice.***


Microservice 1
Microservice 2
Microservice 3

Cloud Deployment

< 10 TB

Binary data
(videos, images, audio, ..)

Lob data

Rest API Access
(access anywhere)

Cloud Object Storage

S3, Azure blob

File protocol Support

Cloud File Storage

EFS, Azure Files

Persistent
access from VM

Managed Disk

EBS

Ephemeral (short)
access from VM

Unmanaged Disk

Transactional
(ACID)

RDBMS

Complex
Relationships

Graph

Semi Structured
Data

Document

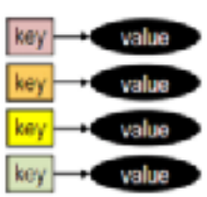
Simple
structure

Persistent

Key Value Store

Transient

Key Value Cache



Is workload compatible with Containers

Yes

No

Is Workload event driven & Short lived

Yes

Serverless Container

Lamda, Azure Function

Many Services To Manage

No

Managed Container

(Few services To manage)

Require Full Control of orchestration Layer

No

Serverless Kubernetes Service

Serverless (PAAS 2.0)

Yes

(PAAS 1.0)

Managed Kubernetes

Work load Requires a web server

Require Full Control of VM, Disk,OS

No

VM (IAAS)

EC2, Azure VM

Is Workload event driven & Short lived

Yes

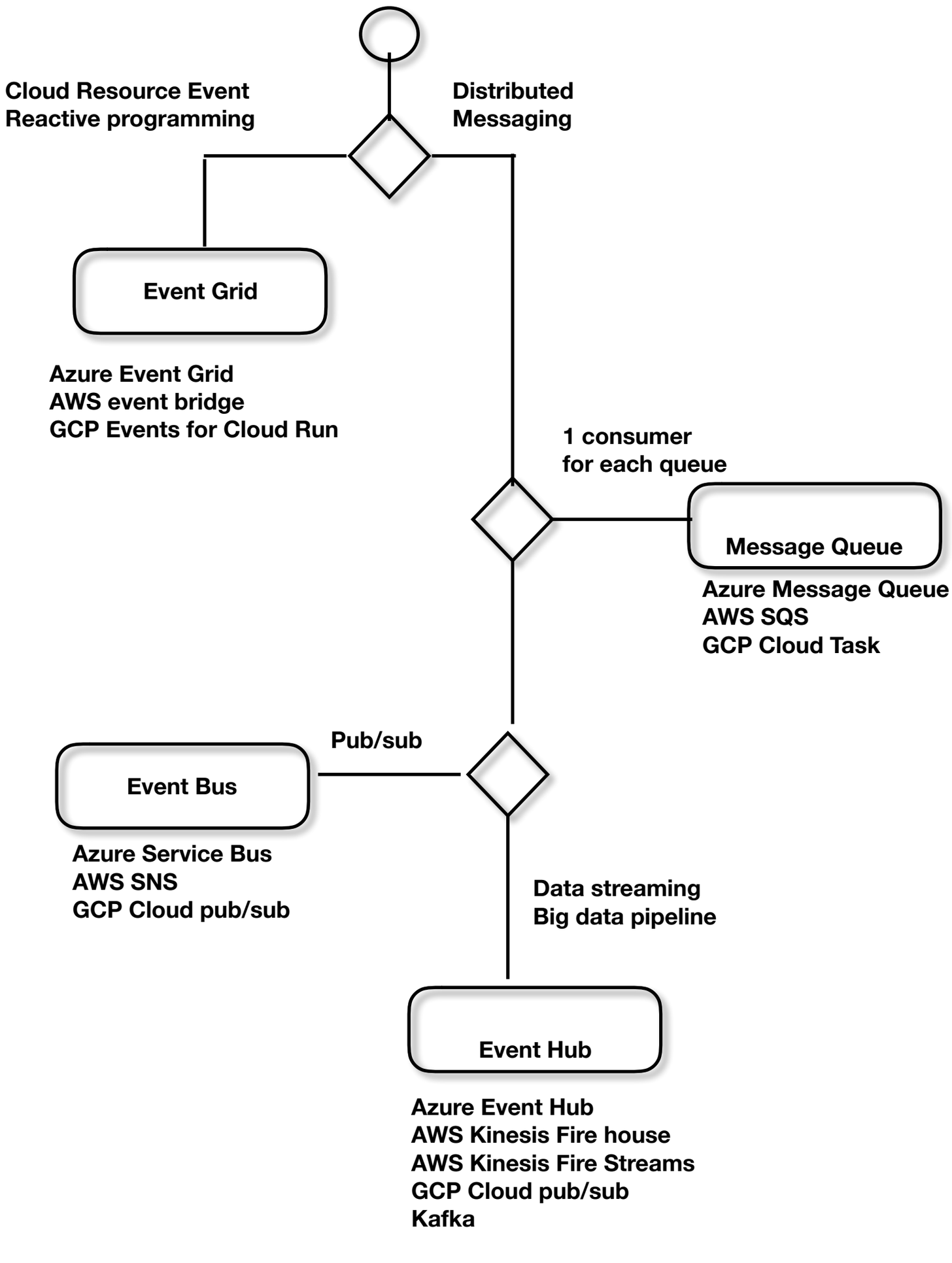
Serverless Webserver

Lamda, Azure Function

No

Managed Web Server

Azure App Service, Elastic Beanstalk


Cloud Resource Event
Reactive programming
Distributed
Messaging
Event Grid
Azure Event Grid
AWS event bridge
GCP Events for Cloud Run
1 consumer
for each queue
Message Queue
Azure Message Queue
AWS SQS
GCP Cloud Task
Pub/sub
Event Bus
Azure Service Bus
AWS SNS
GCP Cloud pub/sub
Data streaming
Big data pipeline
Event Hub
Azure Event Hub
AWS Kinesis Fire house
AWS Kinesis Fire Streams
GCP Cloud pub/sub
Kafka

Data Source Tier

Ingestion Tier

Operation Tier

Insight/Analytics Tier

Structured

unstructured

semi-structured

Real time

Azure Event Hub
AWS Kinesis
GCP Cloud pub/sub
Kafka

Batch

One Time

AWS Snowball,
Azure Data Box
Google Transfer Appliance

Azure Stream Analytics
Azure Kinesis Analytics
Storm
Spark Streaming

Machine Learning

Stream Processing

runs on the incoming data

AWS Elastic MapReduce (EMR)
AZURE HDInsight

Batch Processing

runs on the entire dataset

Storage Tier

Landing Zone

Raw

handling missing values,
filtering outliers,
& Validated data

Staging

Data Lake
AWS S3
Azure Data Lake

Highly refined views of the data

Curated

Analytical data store
Aws Redshift
Azure Synapse Analytics

BI Tools

Applications

Automations

QuickSight
Power BI

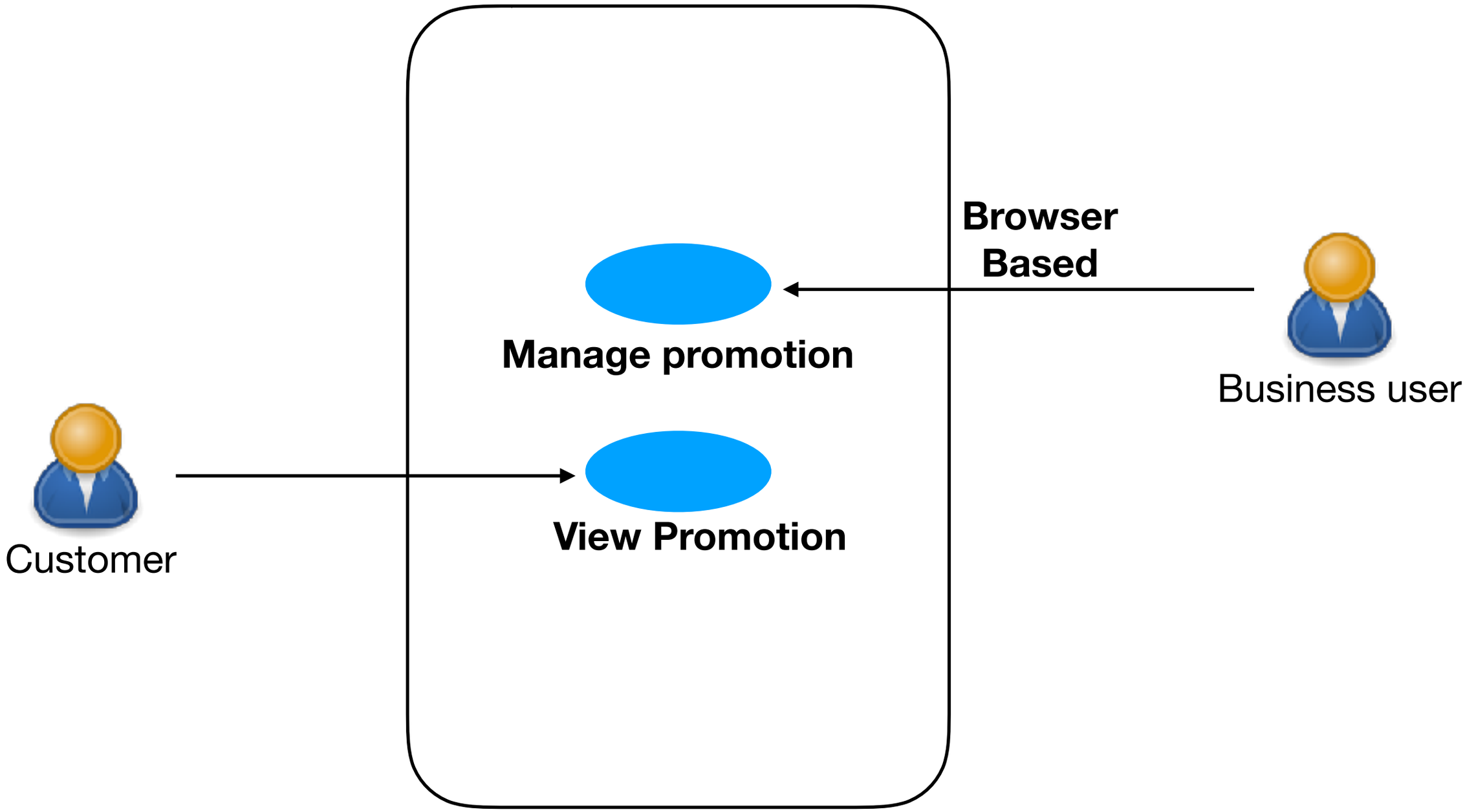
Orchestration Tier

Workflow

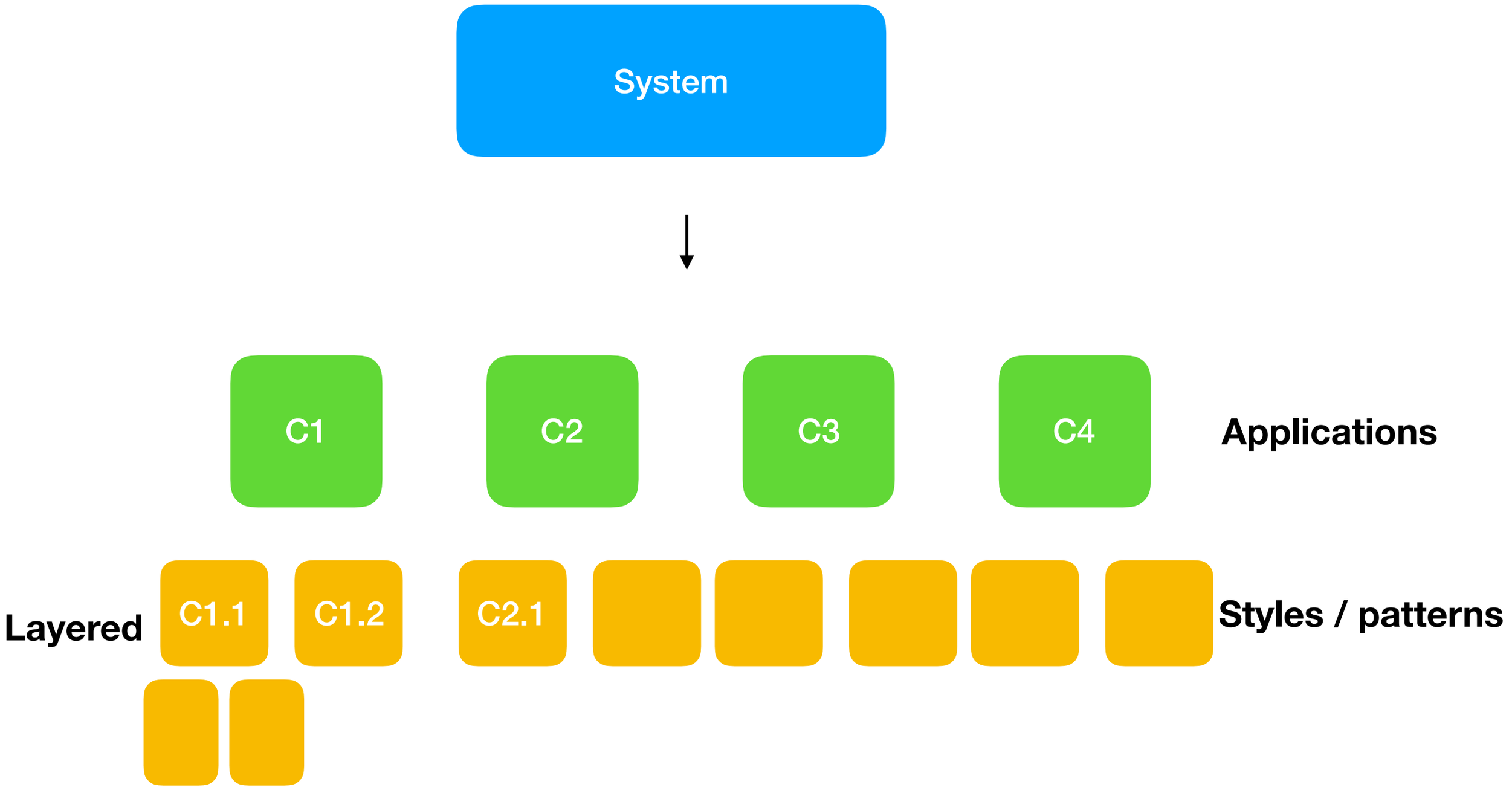
AWS Data Pipeline
Azure Data Factory
Apache Oozie and Sqoop

Context view

Functional view



Logical view



System Decomposition

Client Application

UI Application
Embedded App
* Ras pi

Native Application
* WPF
* Swing
* Android
# Browser - Native Application
* Flash
* Activex
* Applet
# Browser - Server Pages
* JSP
* Php
# Browser - SPA
* Angular
* React

Background Application
* Windows Service
* Daemon

Low Level Application
* Device Driver

Server Application

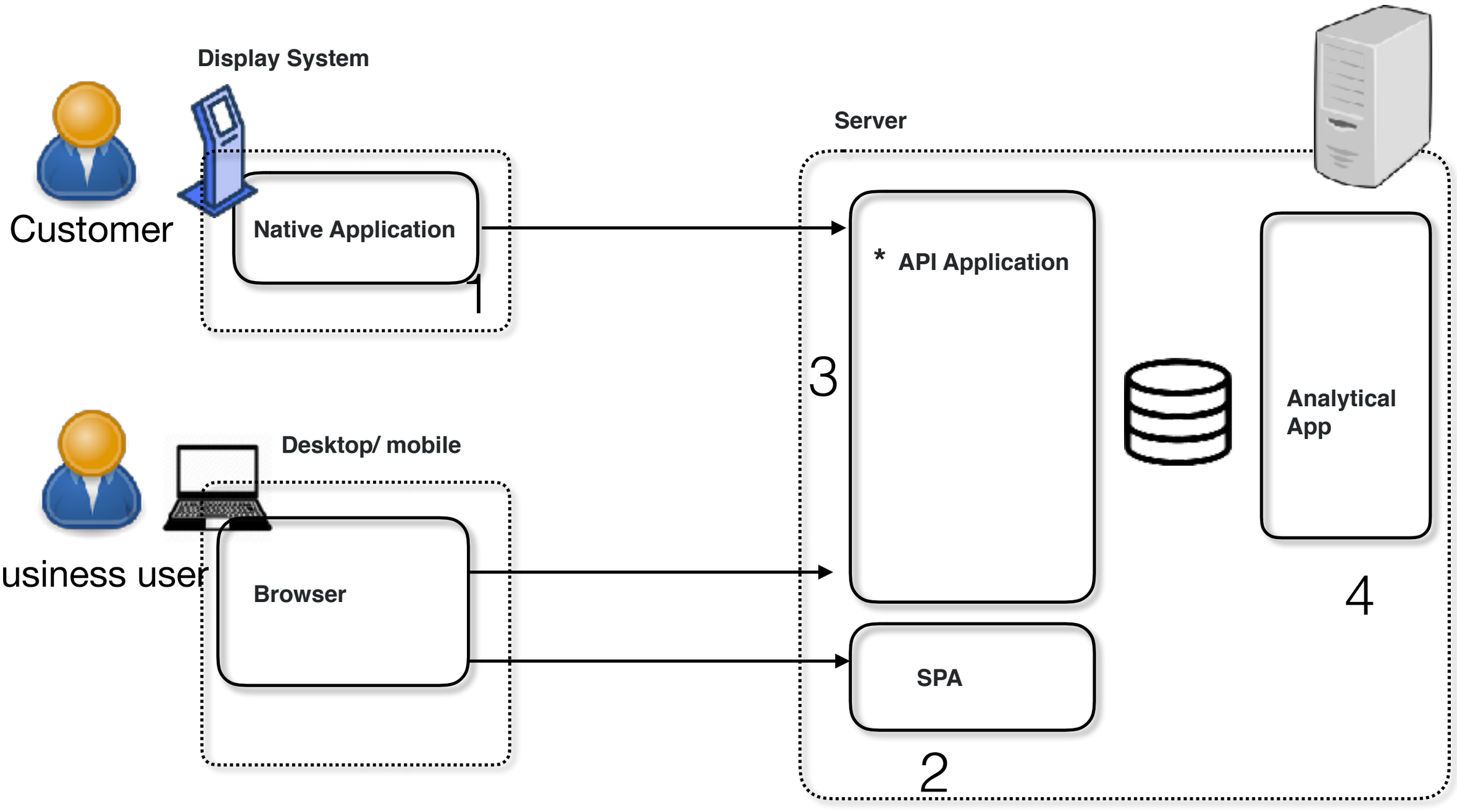
Foreground Application
(Request/response)
* UI App
* API Application

Background Application
Event Based Application
Scheduled Application

Client-Server
Applications

Two-tier Client Server
Multi(N)-tier client-server

Logical view - todo



System Decomposition

Transformations at Similar levels of abstraction

1

Pipes and filter
Batch sequential

subsystems have similar structure

Transformations at Different levels of abstraction

2

Layered
Hexagonal

subsystems have different structure

Decomposition based on core and Variance

3

Micro Kernel
PLA

subsystems are divided into Low level and High level

Interaction oriented Decomposition

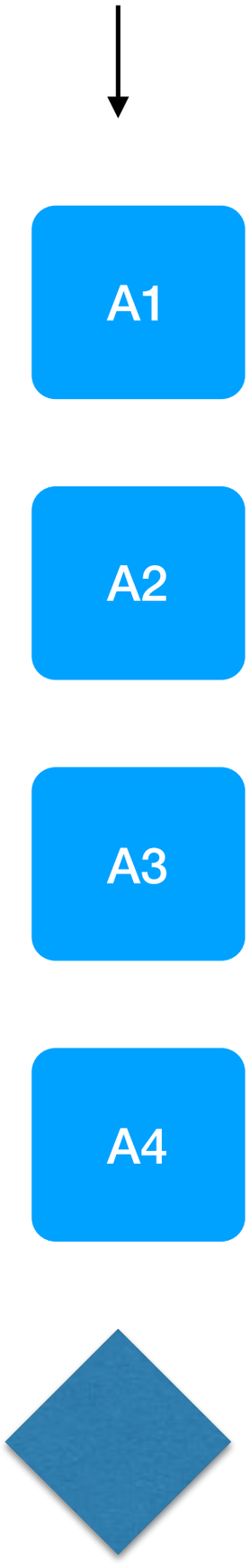
4

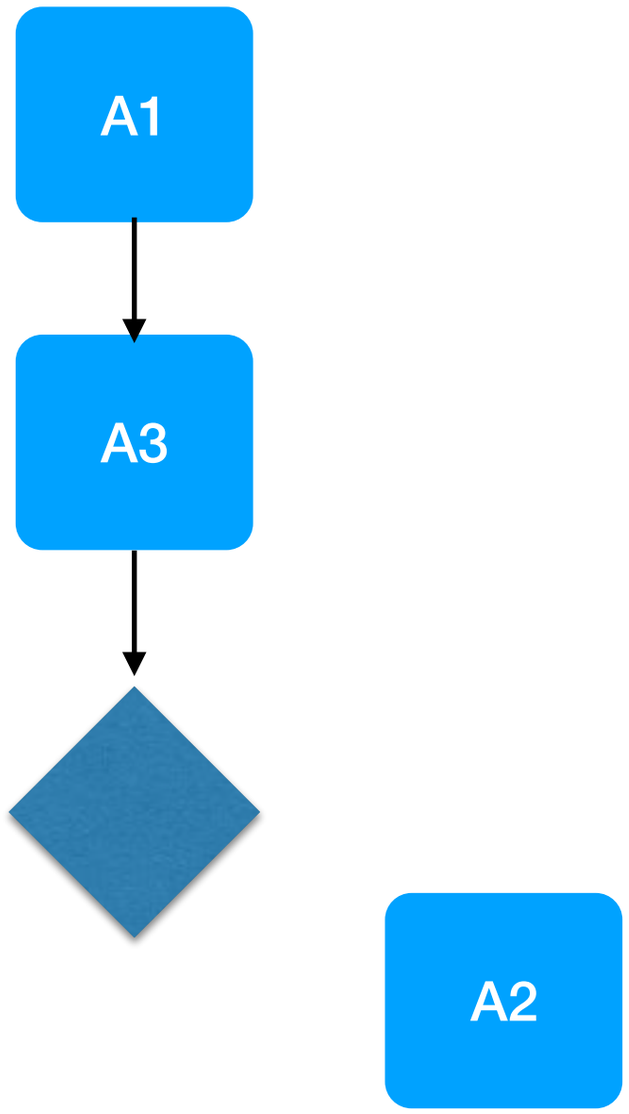
MVC 2
MVP
MVVM

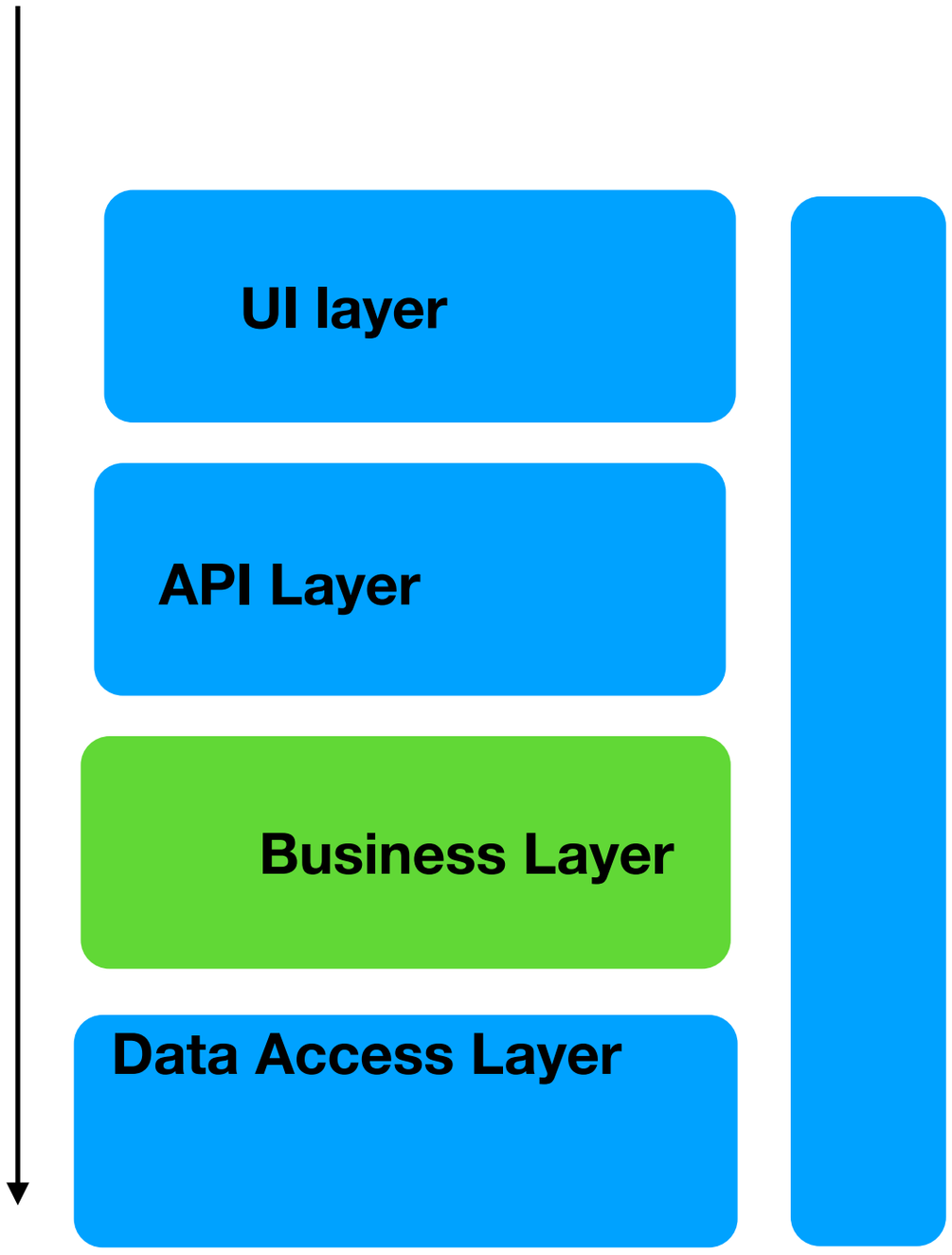
Decompose based on business capabilities (features)

5

Component based
Distributed component
Microservice
SOA


A1
A2
A3
A4


A1
A3
A3
A2


UI layer
API Layer
Business Layer
Data Access Layer

Invoke logic()

Layer 1

f1(int) f2() f3(int,int)

Layer 2

f3(int,int)

f1(bool) f4()

f2(double)

Layer 3

f1() f4()

f2(int) f3() f5(int)

Layer 4

f1(int) f2()

Process Data()

Pipe

Filter 1

f1(int)

Filter 2

f2(int)

Filter 3

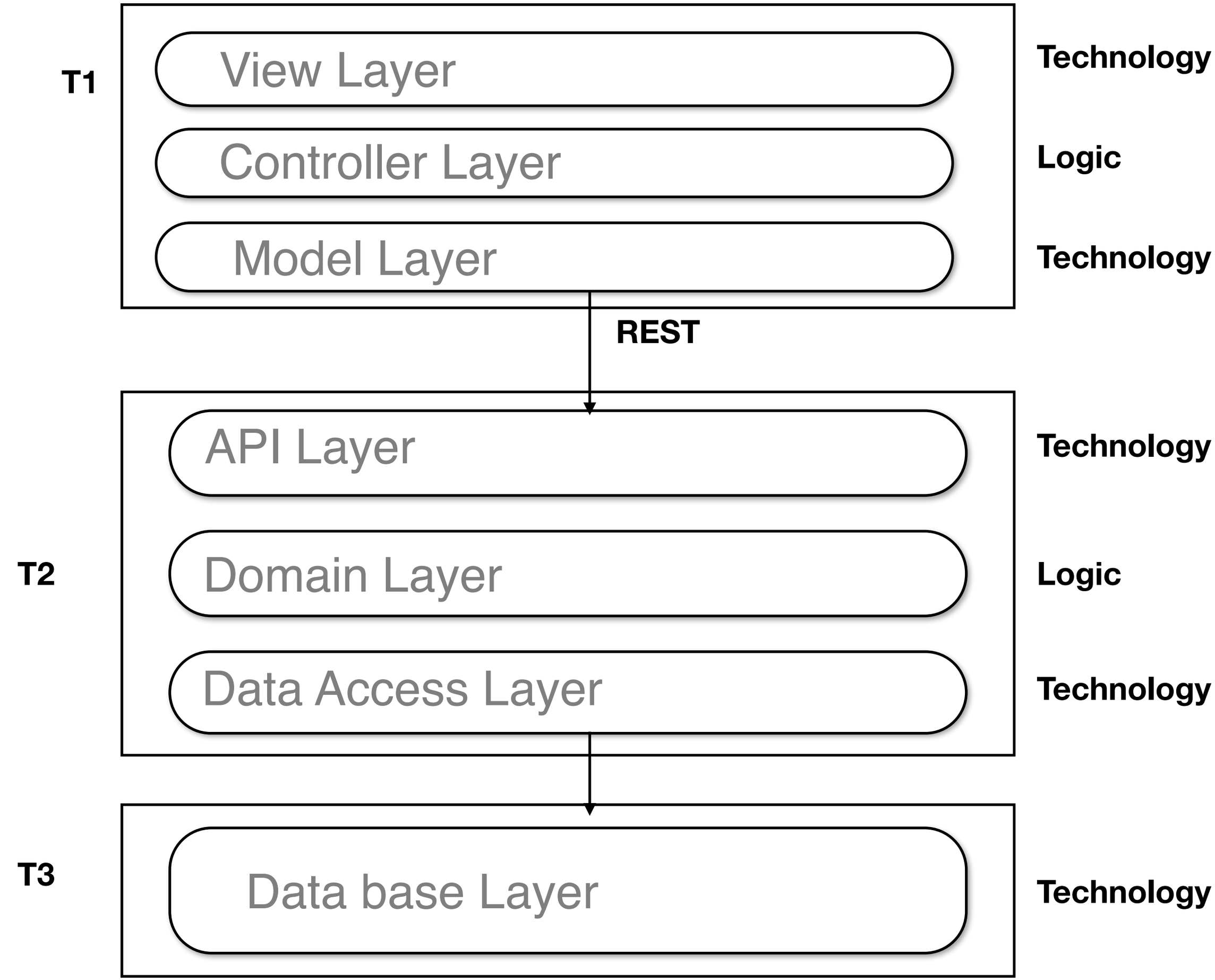
f3(int)

* Can swap filters

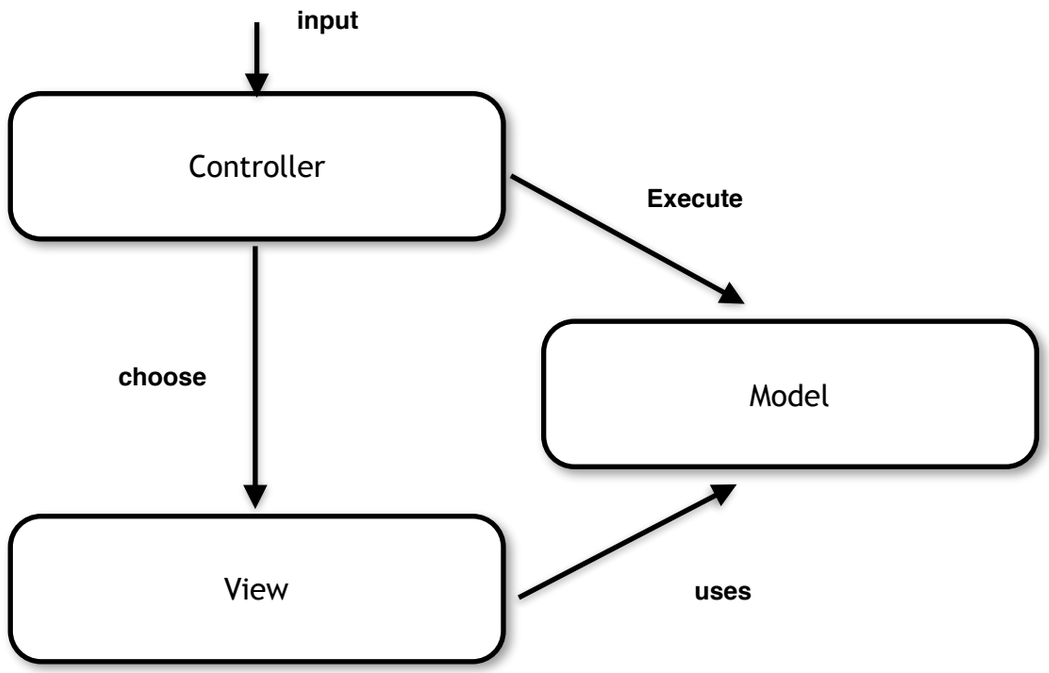
Filter 4

f4(int)

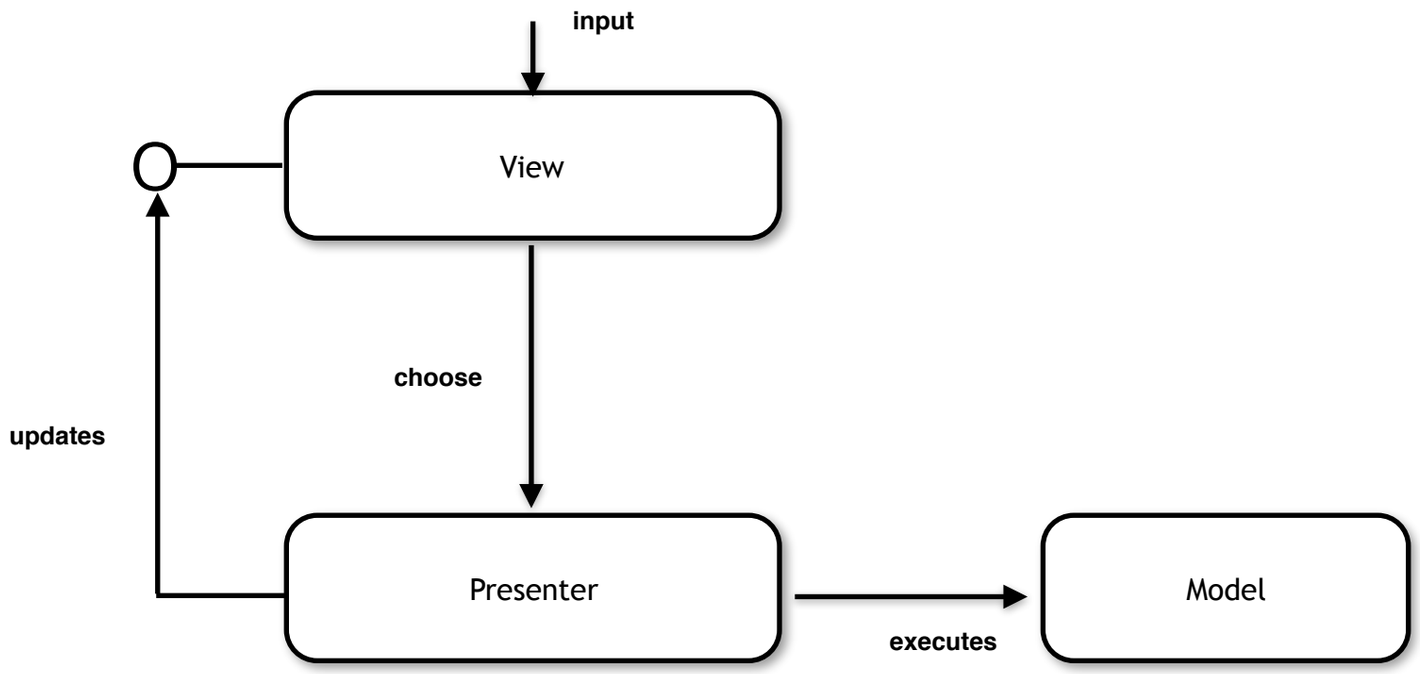
Data


T1
View Layer
Controller Layer
Model Layer
Technology
Logic
Technology
REST
API Layer
Technology
T2
Domain Layer
Logic
Data Access Layer
Technology
T3
Data base Layer
Technology

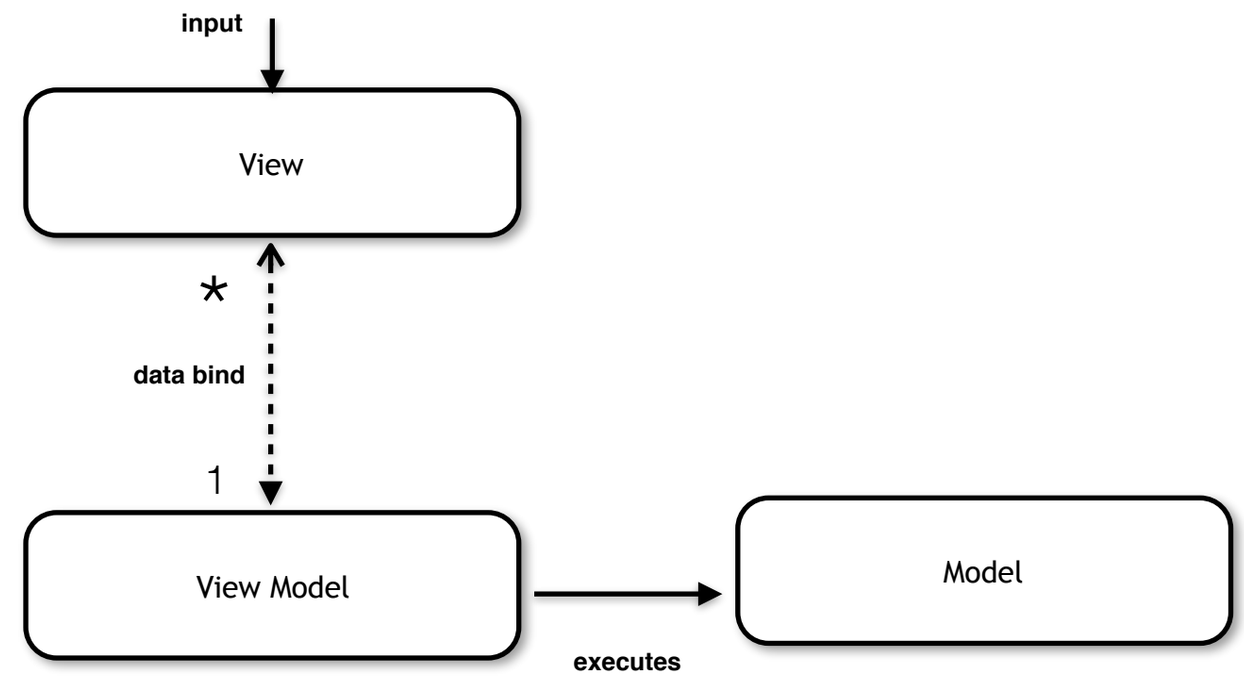
MVC 2



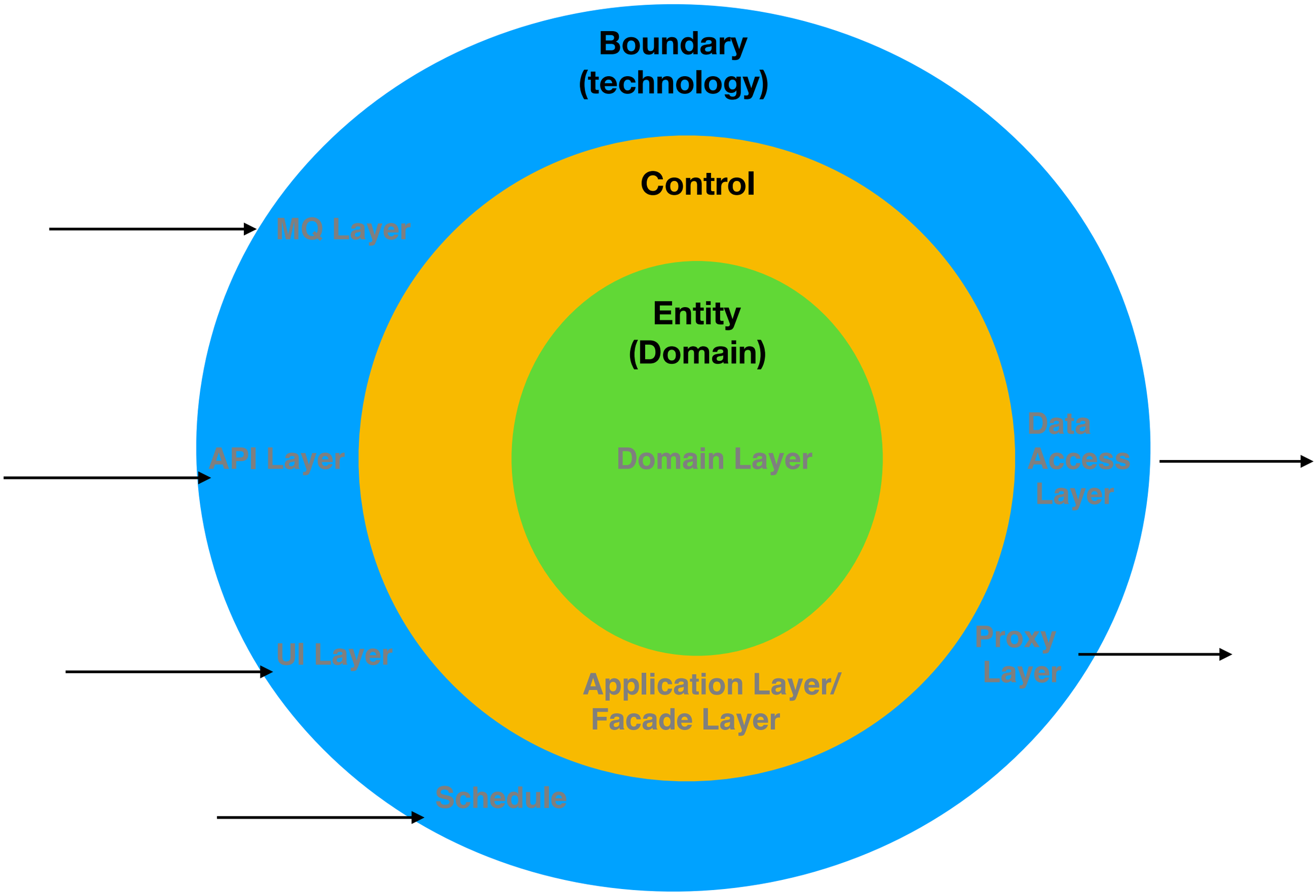
mvp

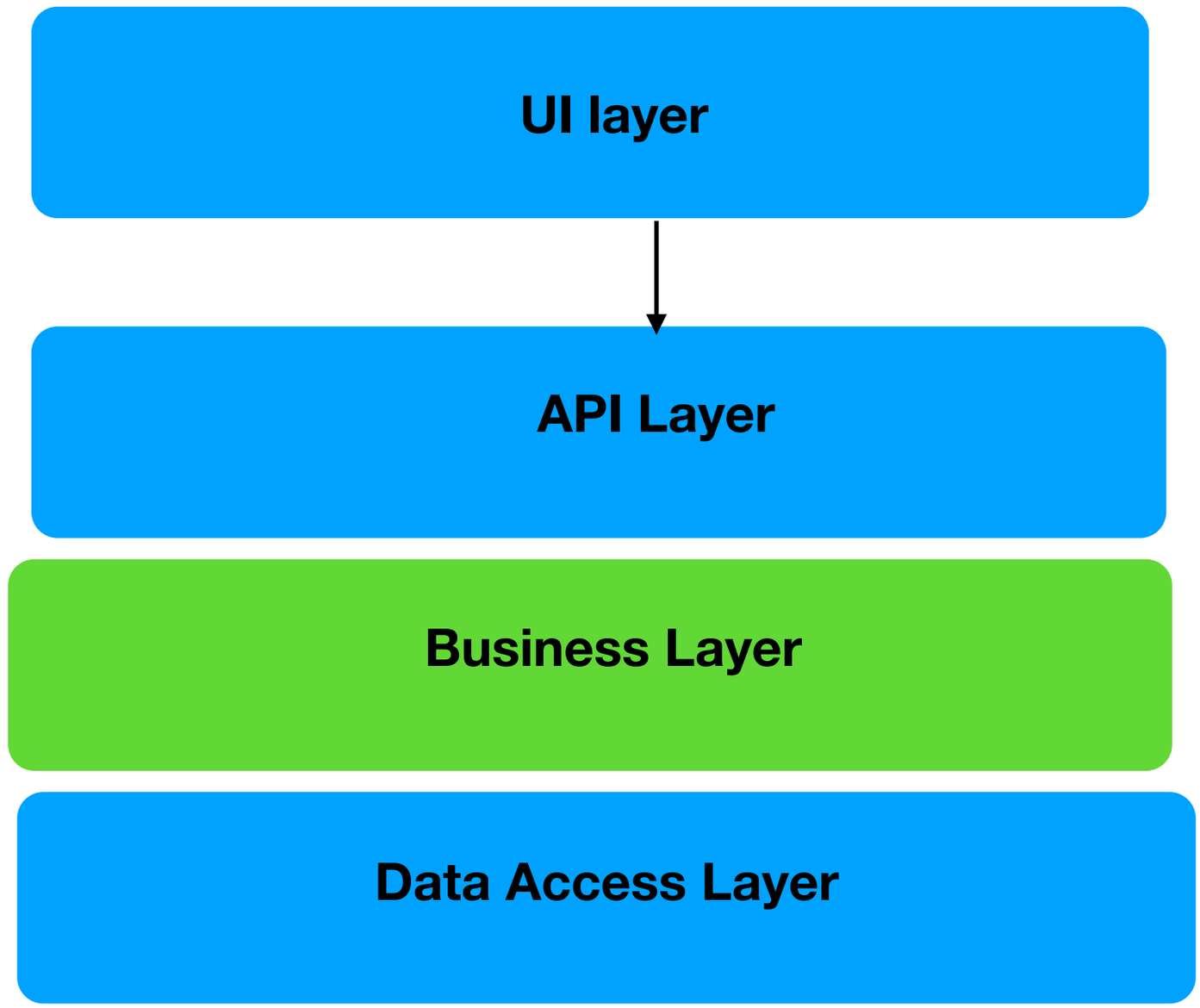


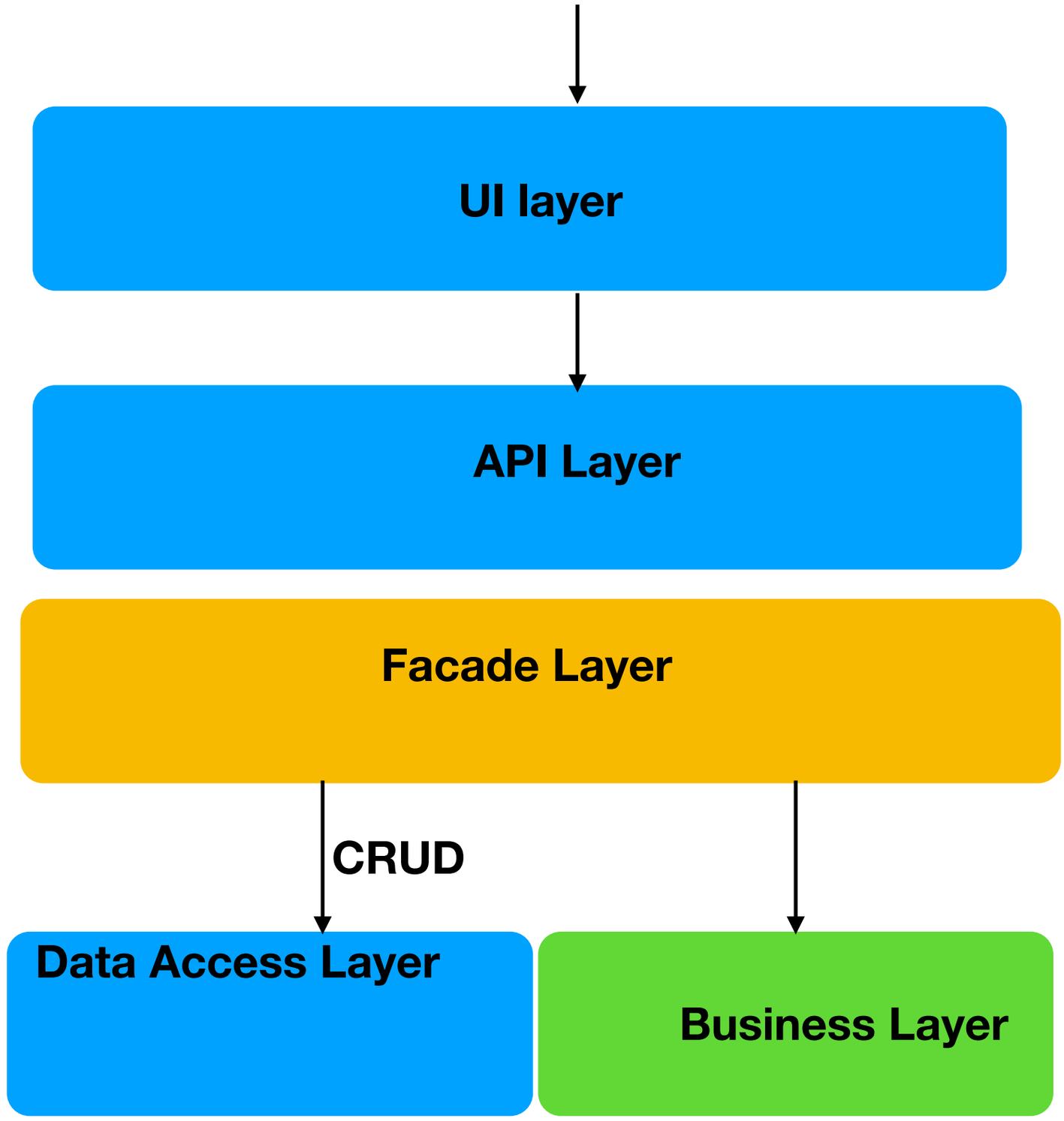
MVVM



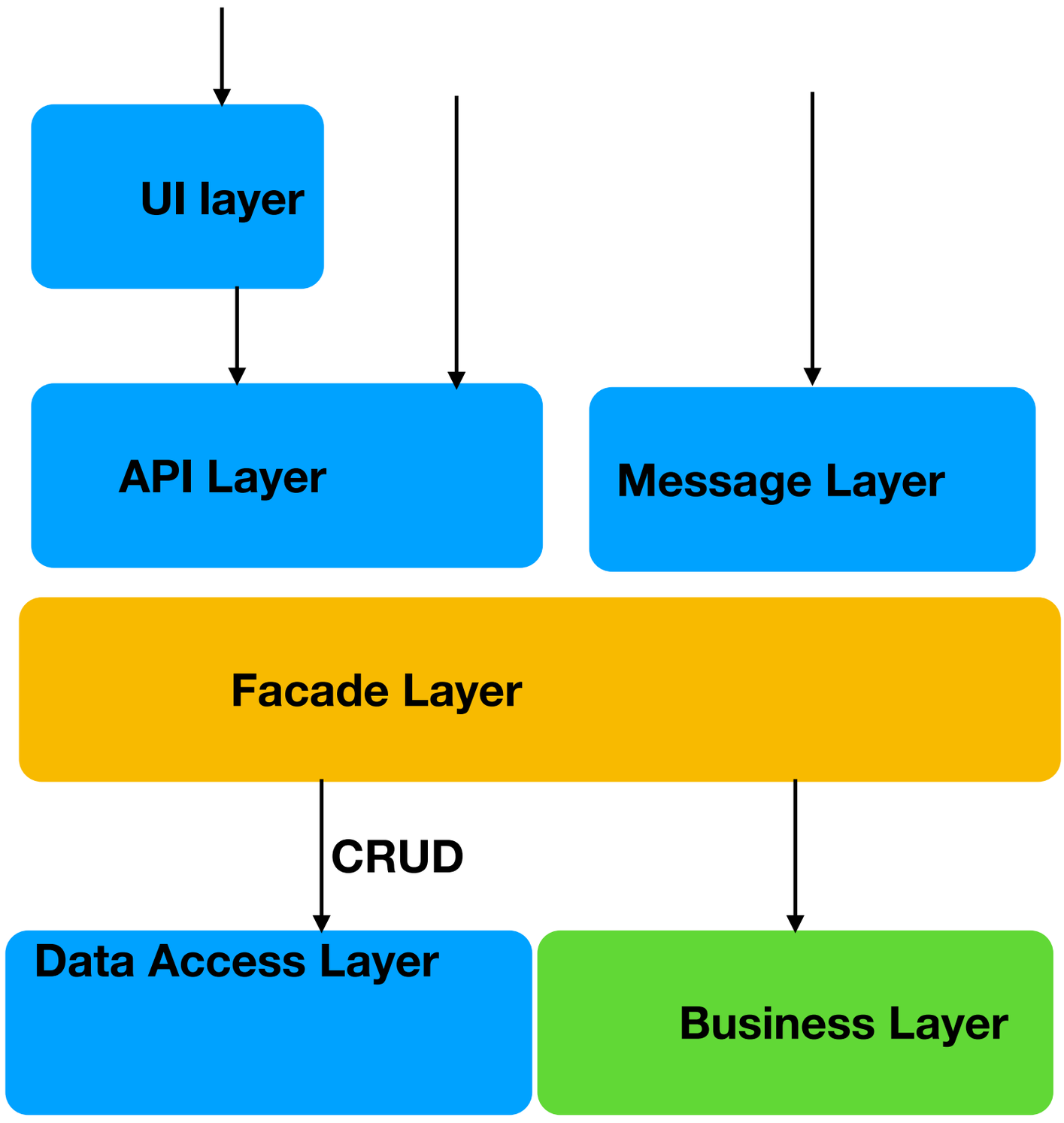
Hexagonal Arch



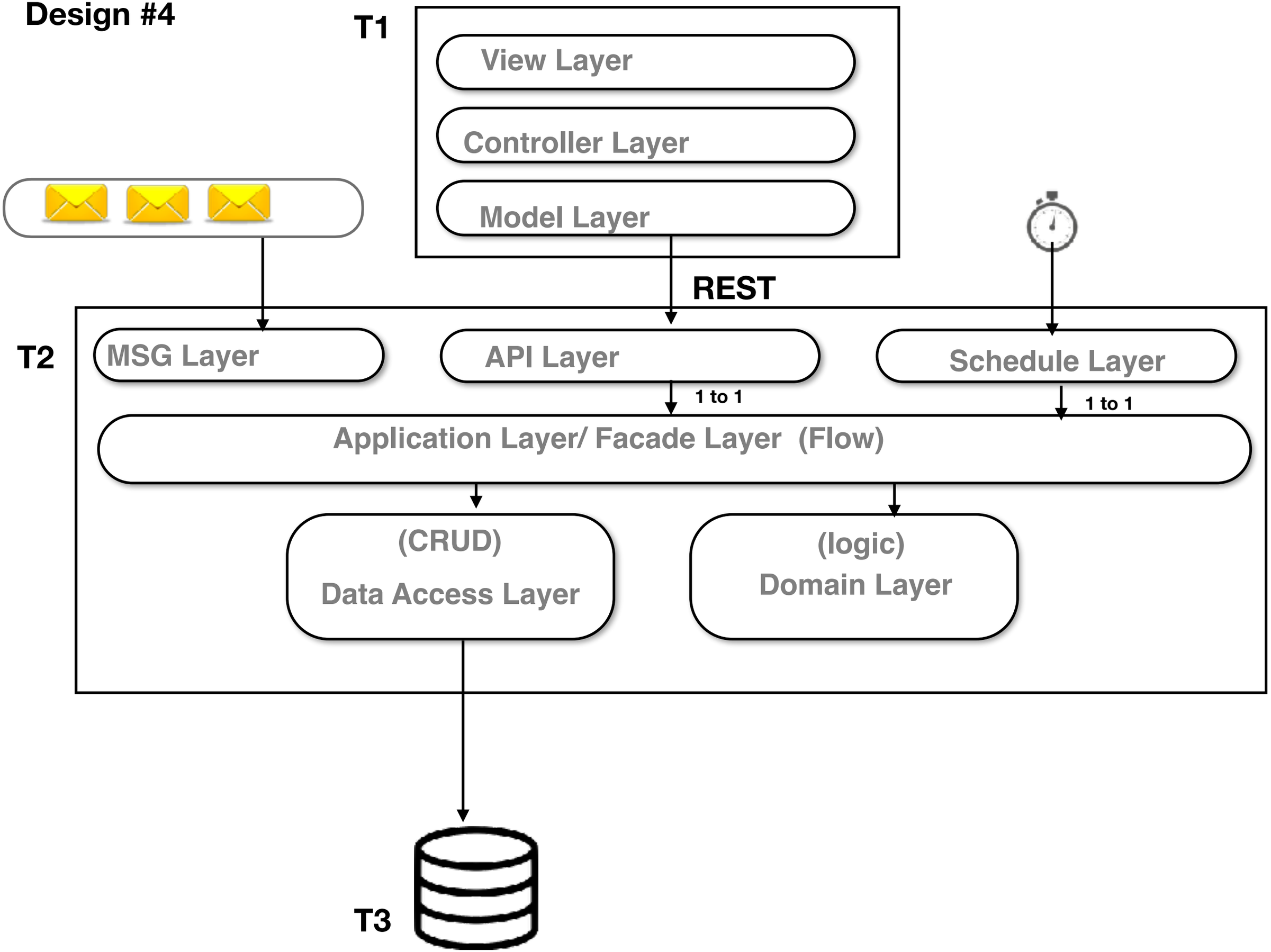

UI layer
API Layer
Business Layer
Data Access Layer


UI layer
API Layer
Facade Layer
CRUD
Data Access Layer
Business Layer

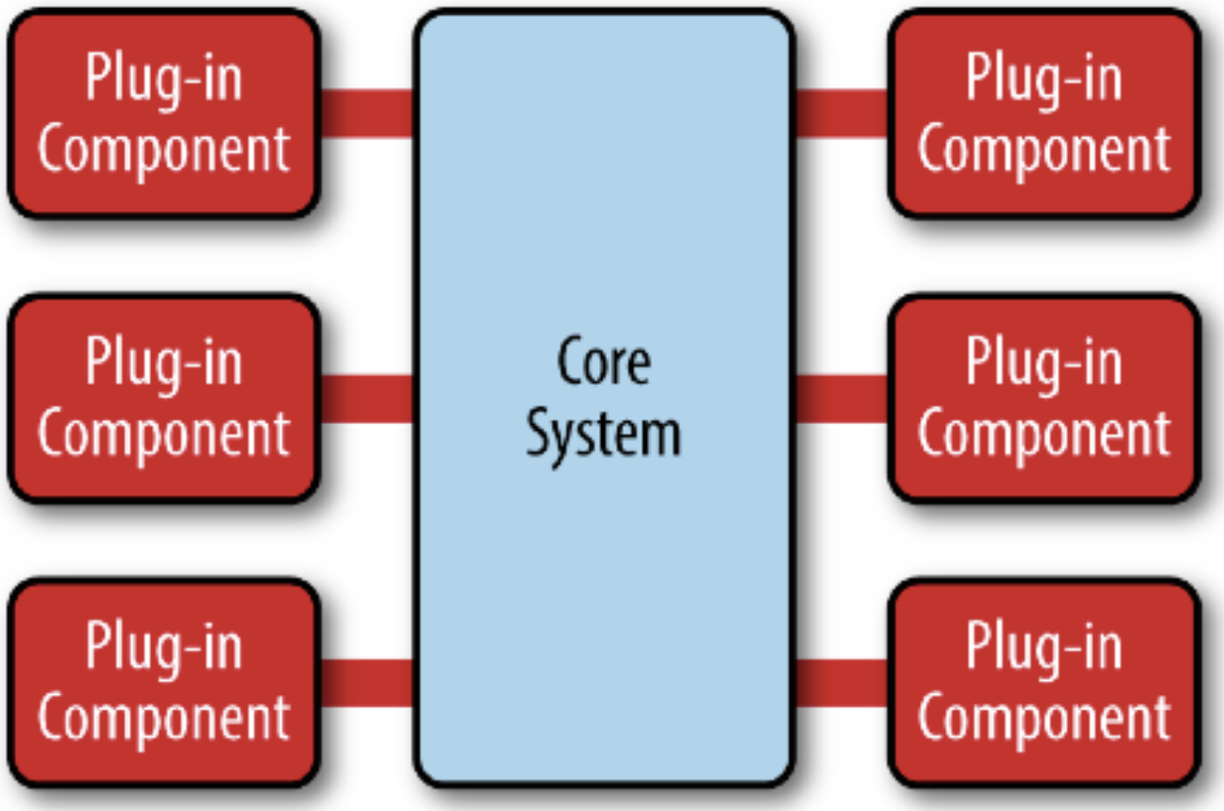
Hexagonal Arch



Design #4


T1
View Layer
Controller Layer
Model Layer
REST
T2
MSG Layer
API Layer
Schedule Layer
1 to 1
1 to 1
Application Layer/ Facade Layer (Flow)
(CRUD)
Data Access Layer
(logic)
Domain Layer
T3

Microkernel Architecture



ToDo App is simple and awesome app to organize your tasks with very easy to use interface. ToDo can help you to make list of your tasks and also you can set Reminder with specific tasks. It reminds you at you specified Time.

Order Processing. Receive an order, Decode an encrypted order, Authenticate the order, then Catch duplicated orders (for example if the order was sent twice) and send to order management system.

A task scheduler. A scheduler contains all the logic for scheduling and triggering tasks

A workflow implementation. The implementation of a workflow contains concepts like the order of the different steps, evaluating the results of steps, deciding what the next step is, etc.

Payment Service receives a data file which consists of,
- Payment information related to the invoices already delivered
- Notes regarding already delivered invoices
- Invoice information,
- And Credit Notes (Invoice Cancellations)

we need to import into the system.
- Invoices
- Payments
- Notes
- Credit Notes

Build two health care applications
1. Diabetes Management
2. Asthma Management.

Diabetes and asthma share a lot in common

1

Application
Exe

Source file

Source file

Compile time
monolithic

2

Application
Exe

Lib

Lib

Link time
monolithic

3

Process

Application
Exe

Dll/so/jar

Dll/so/jar

Dll/so/jar

Runtime
monolithic

In process
comp

4

Process

Exe

Dll/so/jar

Distributed
comp

Process

Exe

Dll/so/jar

Distributed
comp

Distributed Application

5

Process

Exe

Dll/so/jar

Application1

Process

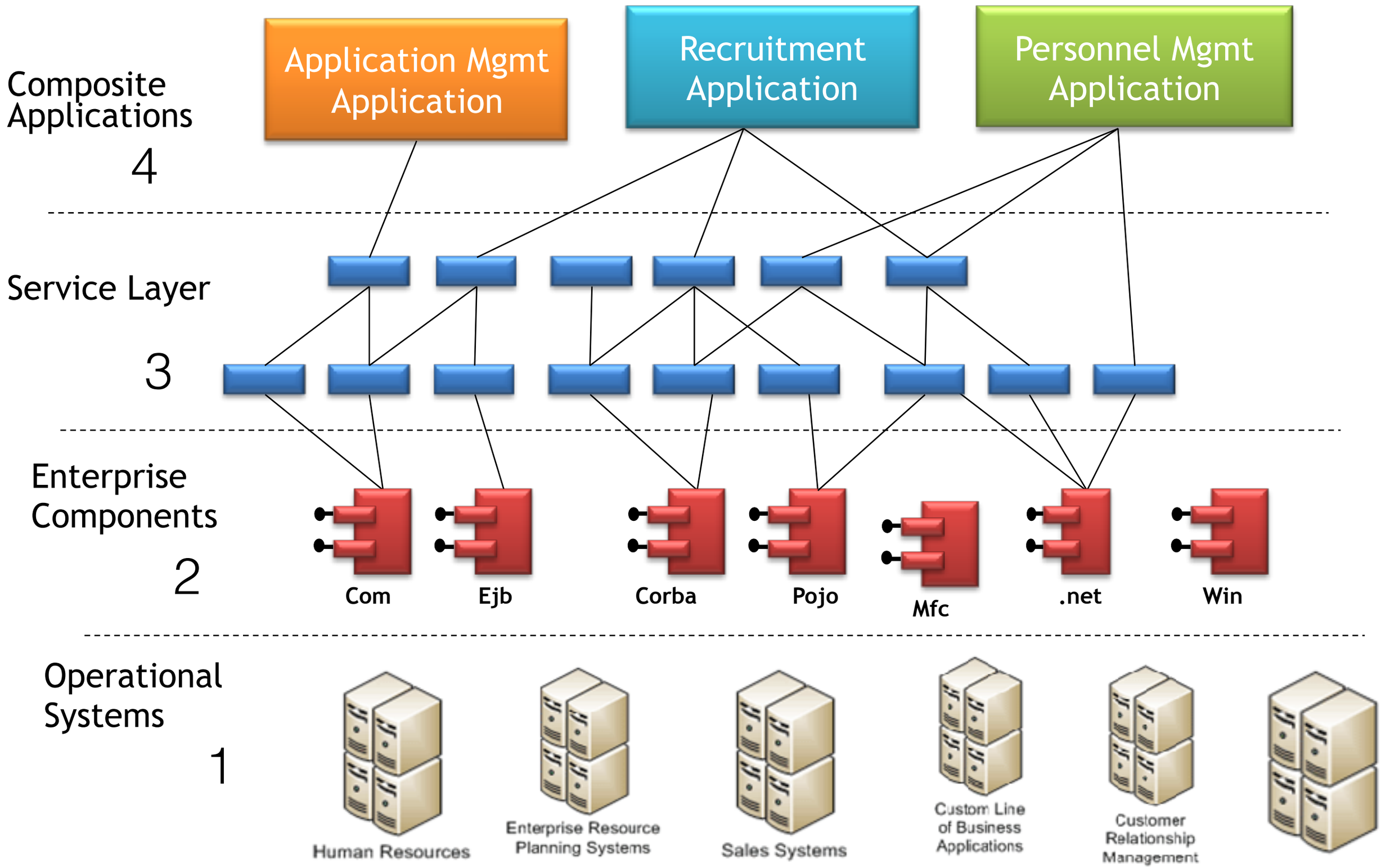
Exe

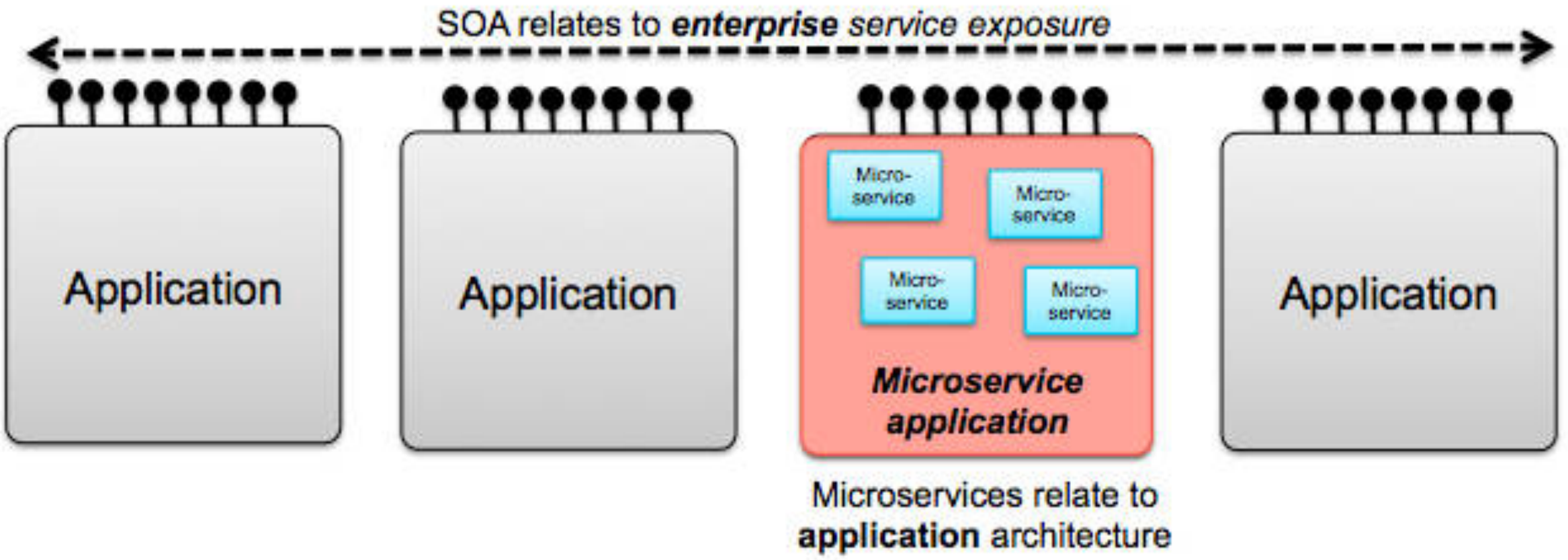
Dll/so/jar

Application2

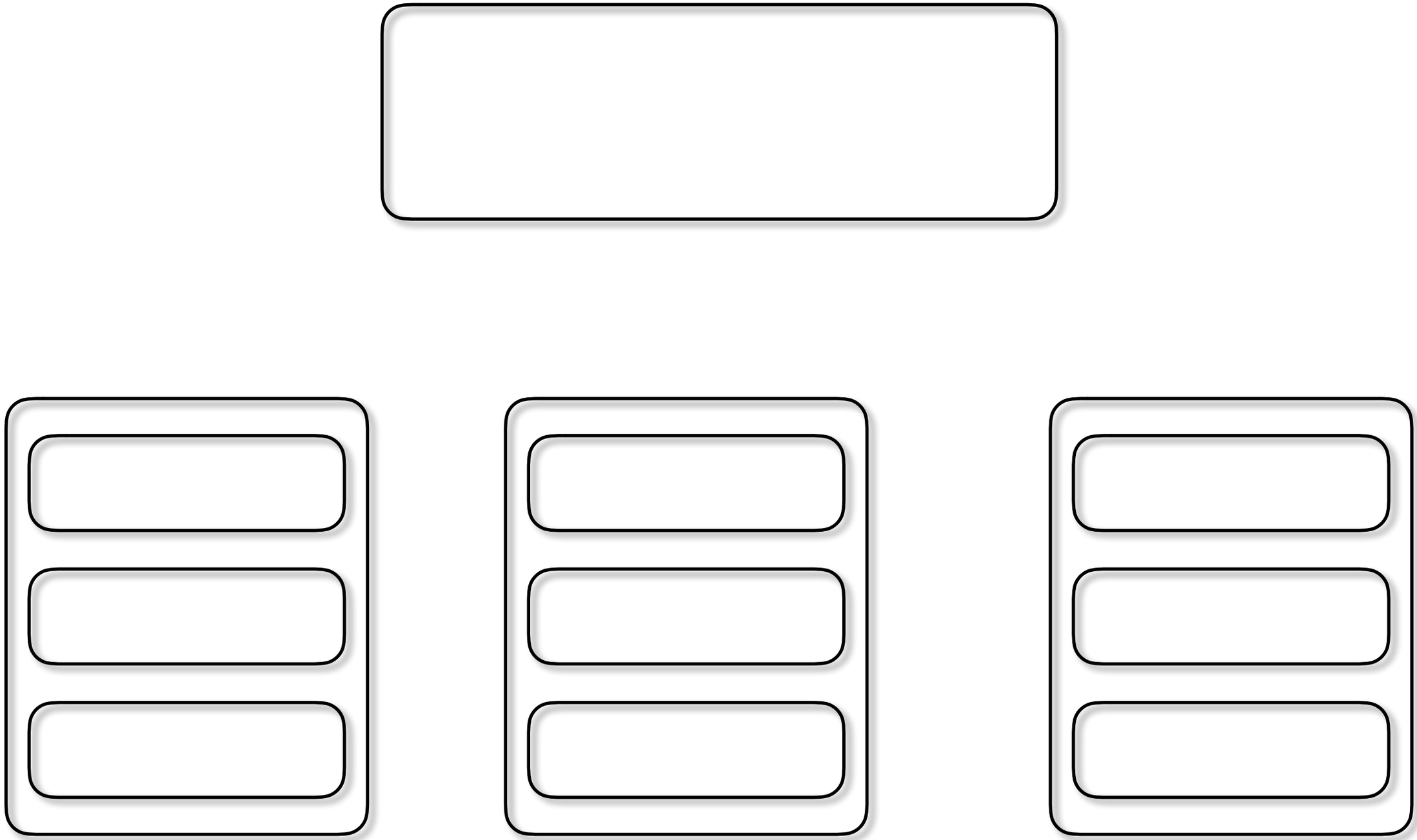
Micro Application

Service Oriented Architecture

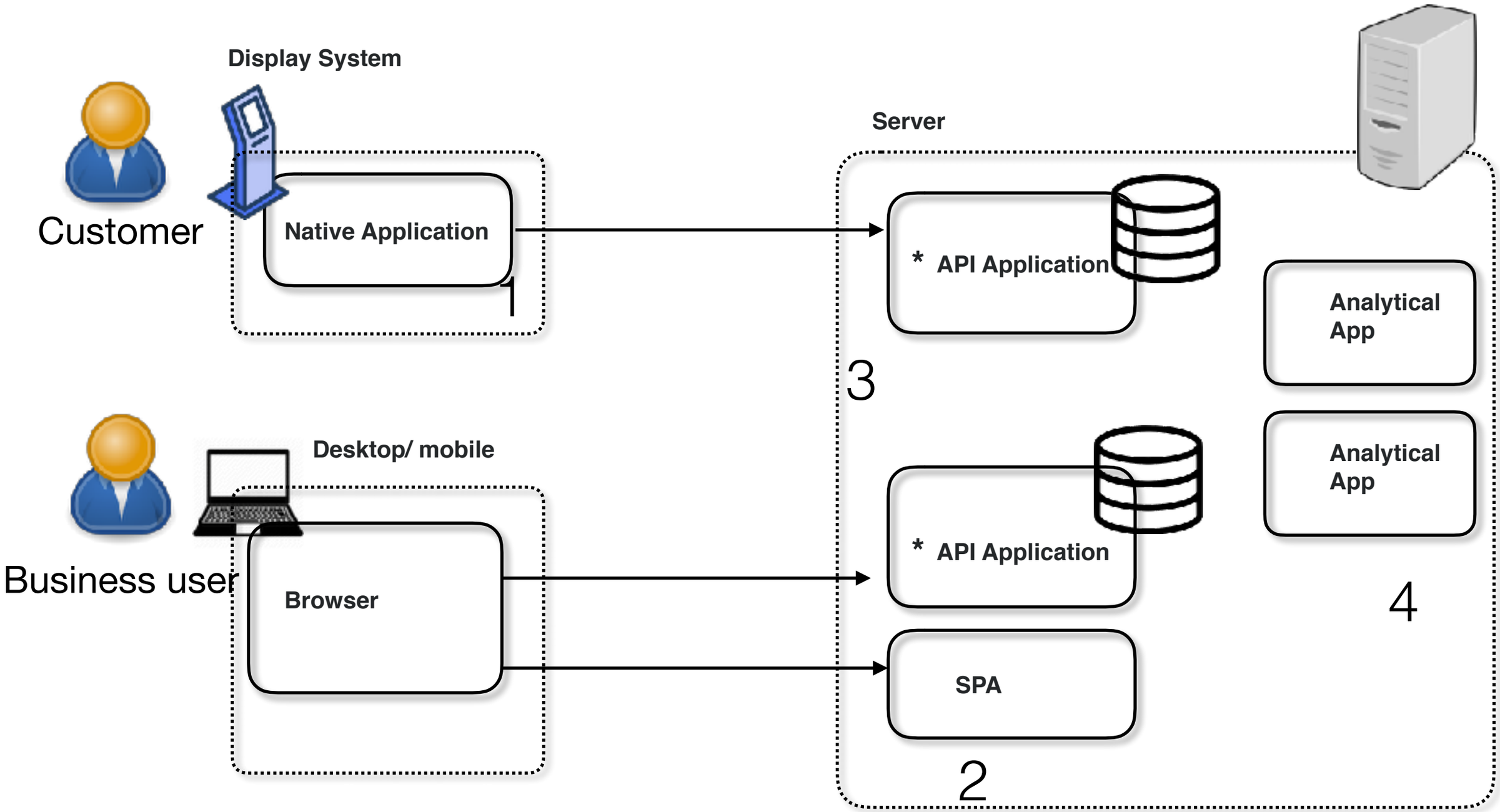




Microservices relate to **application** architecture

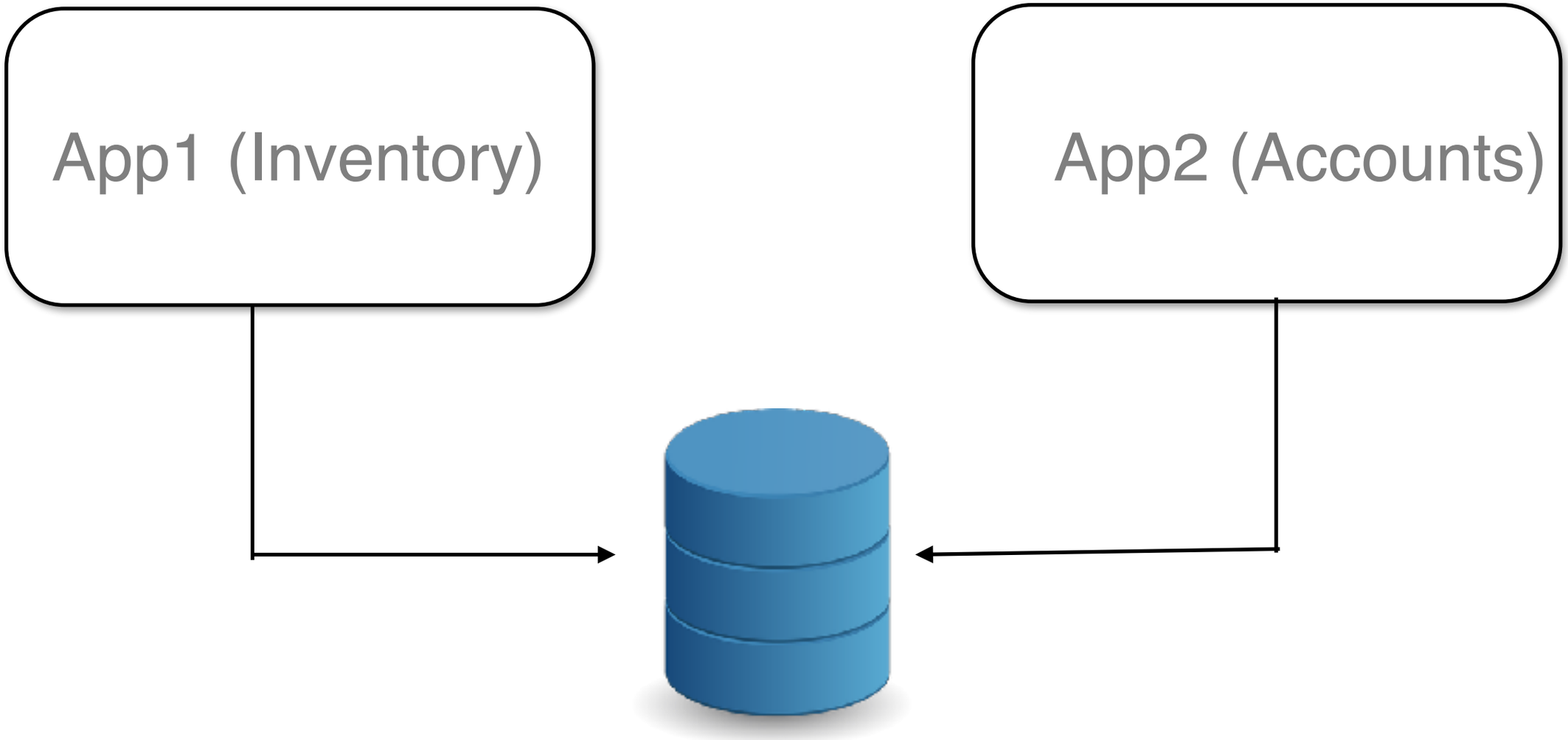


Logical view - todo

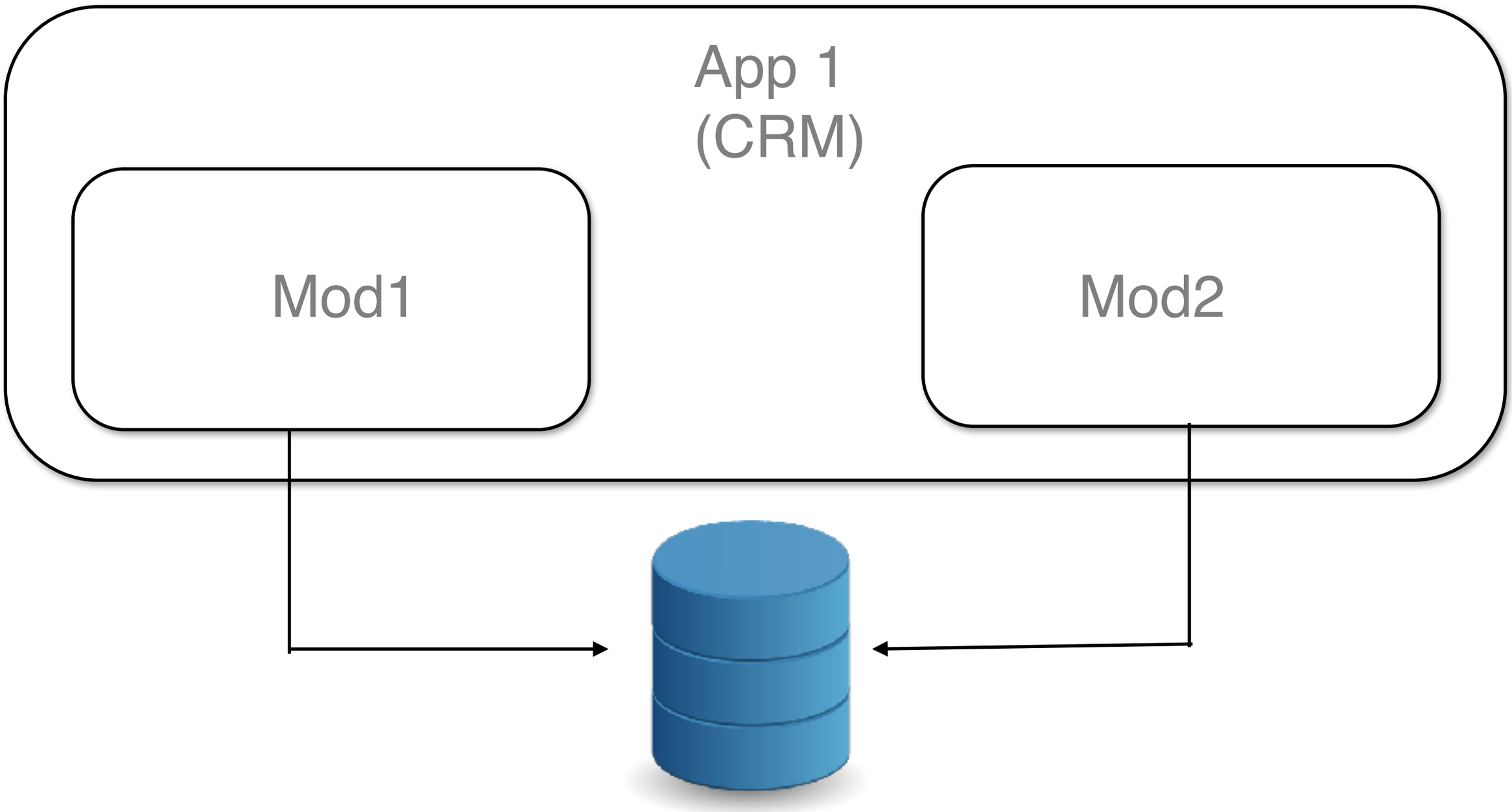


	Application Decomposed into Traditional Distributed Modules	**Application Decomposed into Micro Distributed Application**	**W**	**Score**
Database / Storage	Shared	Not shared	2	
Infra (Hosting)	Shared	Not Shared	3	
Sorce Control	Shared	Not Shared	2	
CI/CD (Build Server)	Shared	Not Shared	3	
Fun Requirements	Shared	Not Shared	1	
SCRUM Team / Sprint	Shared	Not Shared	1	
Test Cases	Shared	Not Shared	1	
Architecture	Shared	Not Shared	1	
Technology Stack / Fwks	Shared	Not Shared	1	

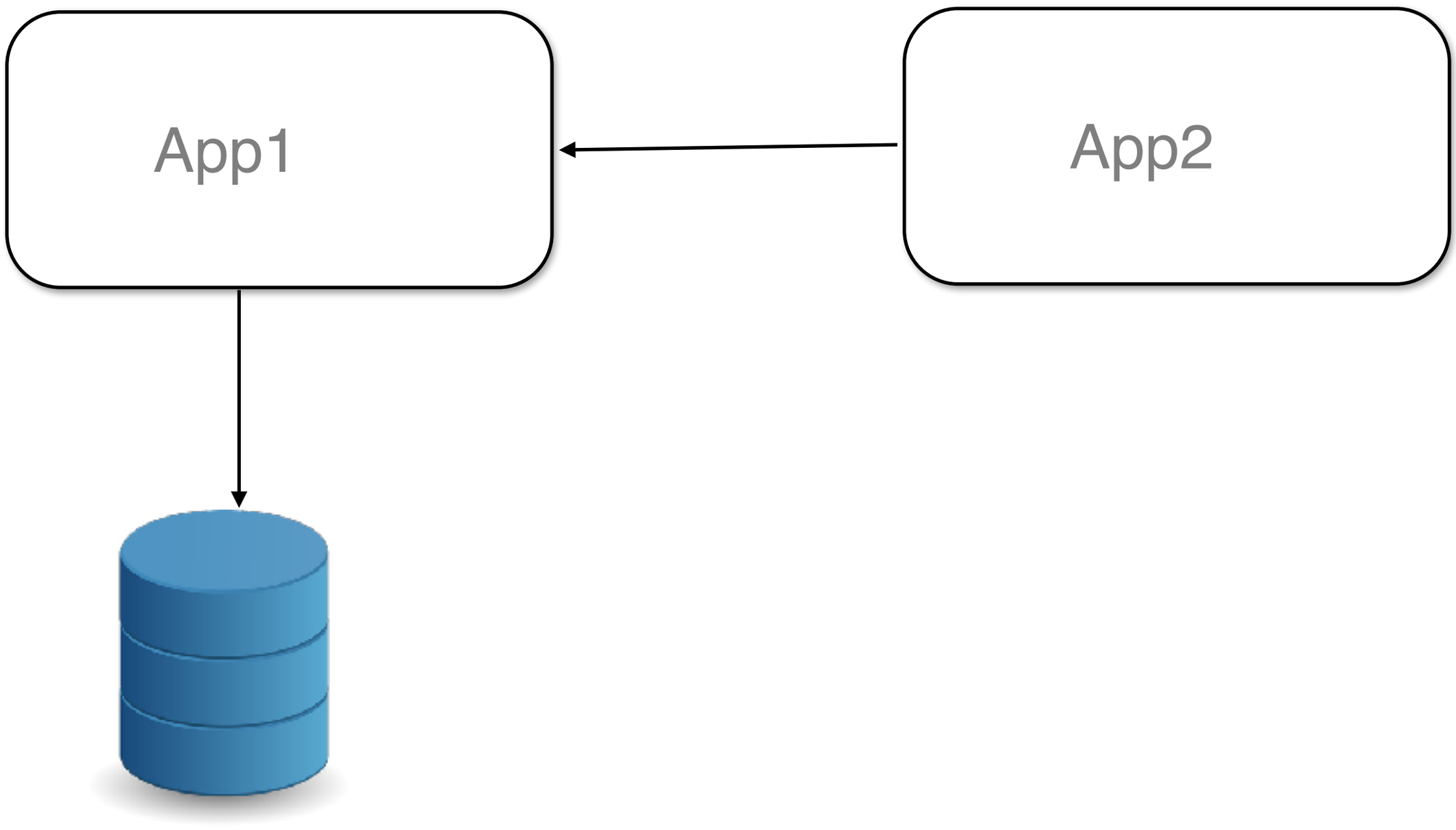
1. Database



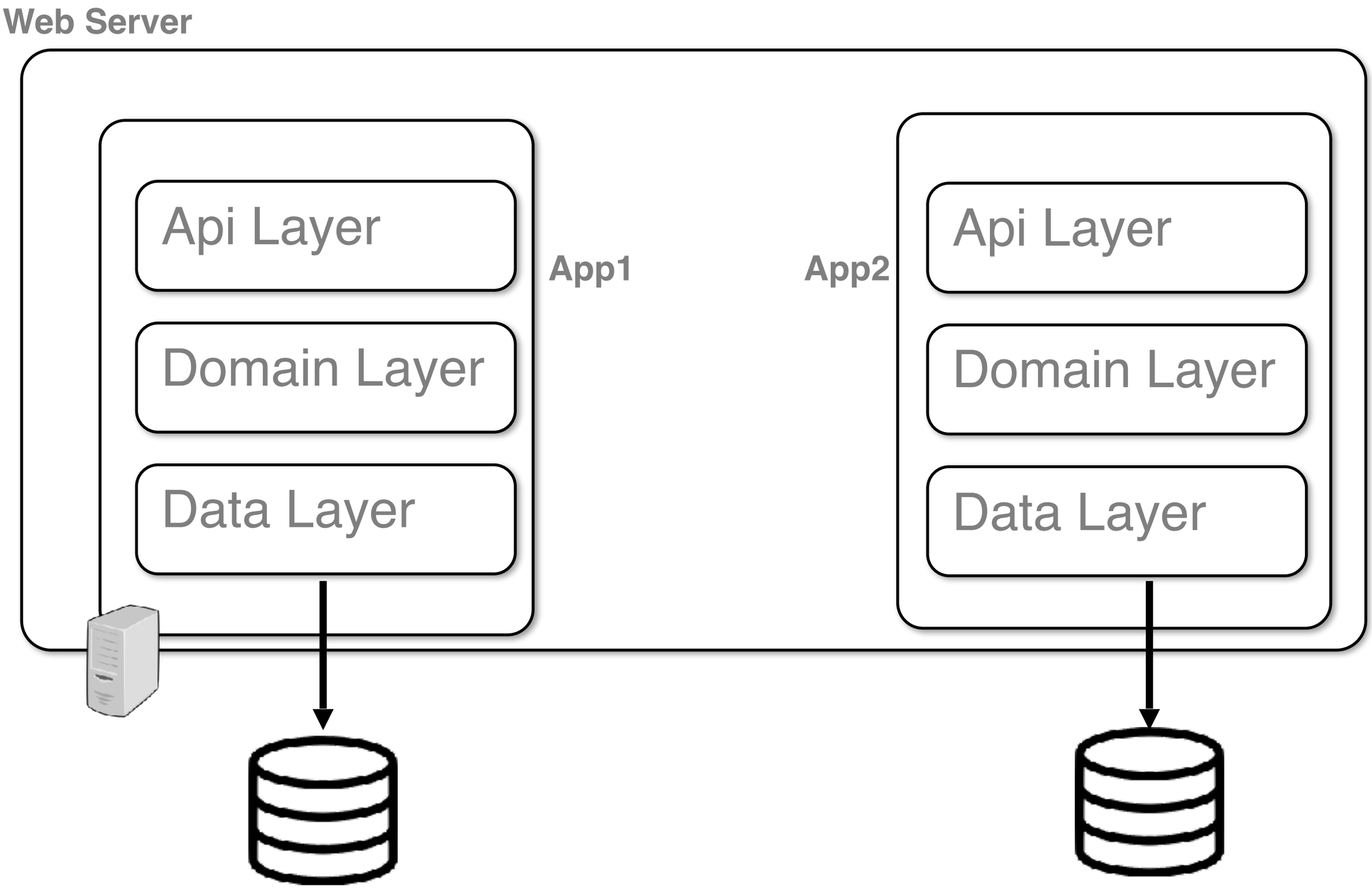
1. Database



1. Database



2. Infrastructure



Types of Microservice

1

UI App (SPA)

Web Server

API Layer

Domain Layer
<<feature1>>

Data Layer

Angular (SPA)

2

Browser

Web Server

UI Layer
(server page)

Domain Layer
<<feature2>>

Data Layer

JSP, Servlet, JSF (Server pages)

3

Desktop

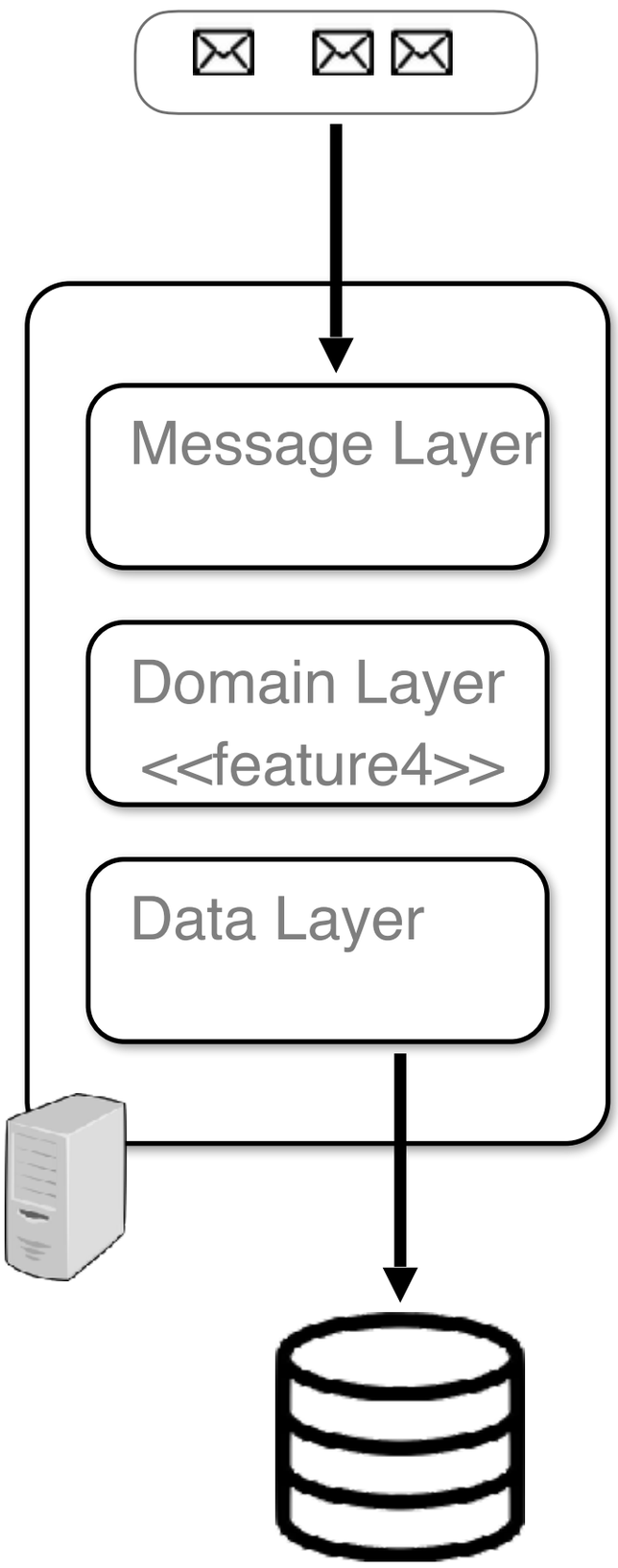
UI Layer
(native)

Domain Layer
<<feature3>>

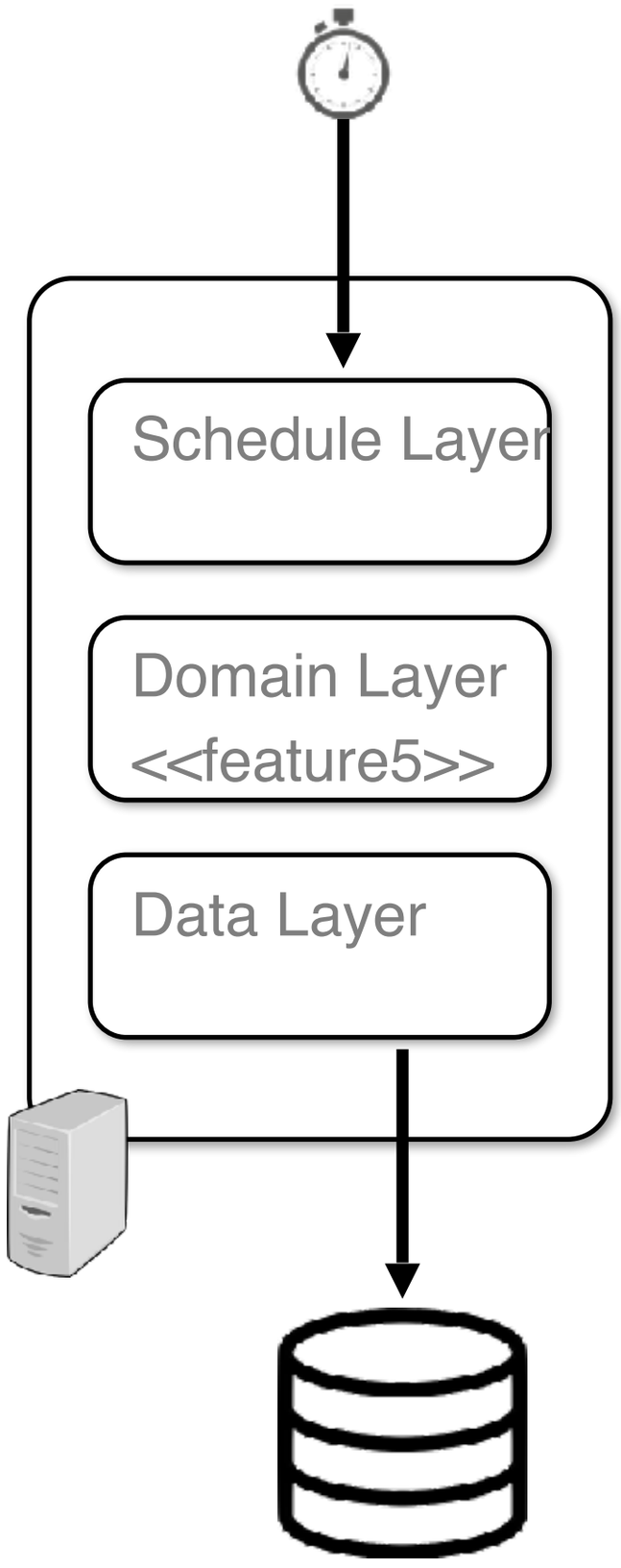
Data Layer

Swing, Android, Ios (Native)

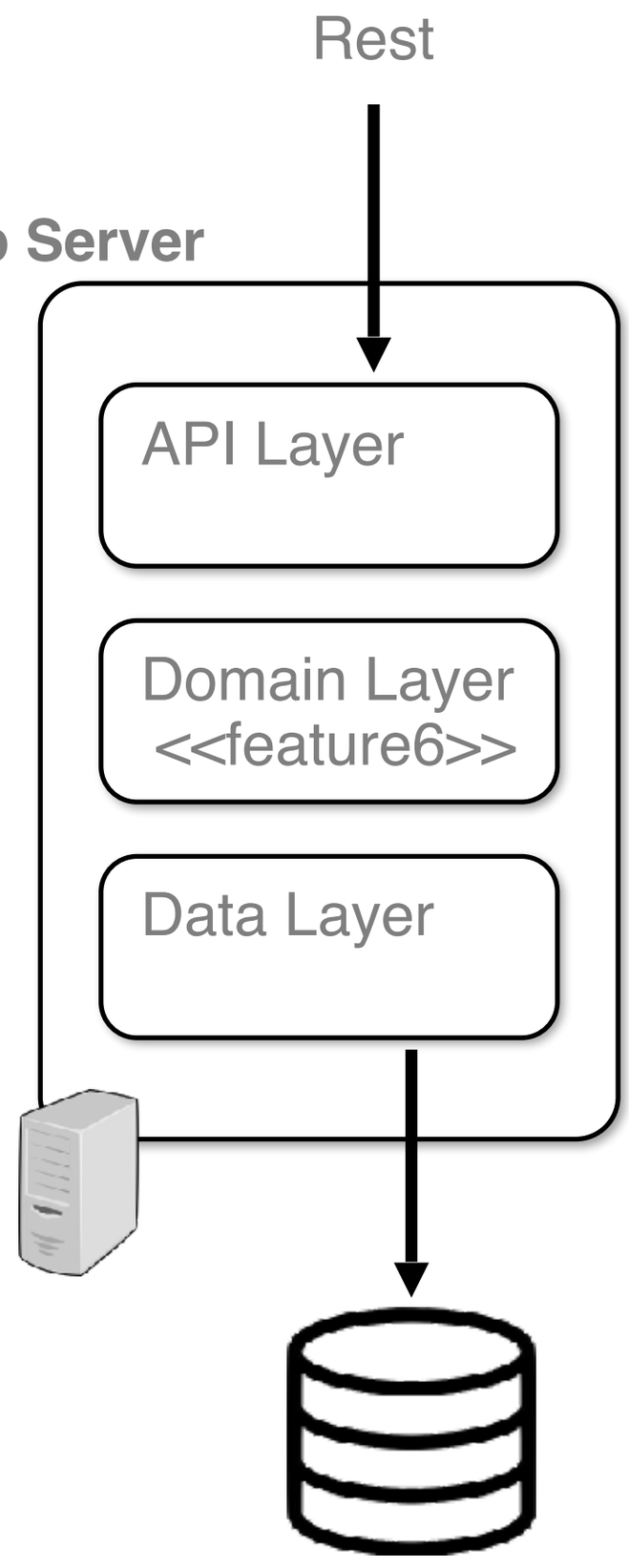
4


Message Layer
Domain Layer
<<feature4>>
Data Layer

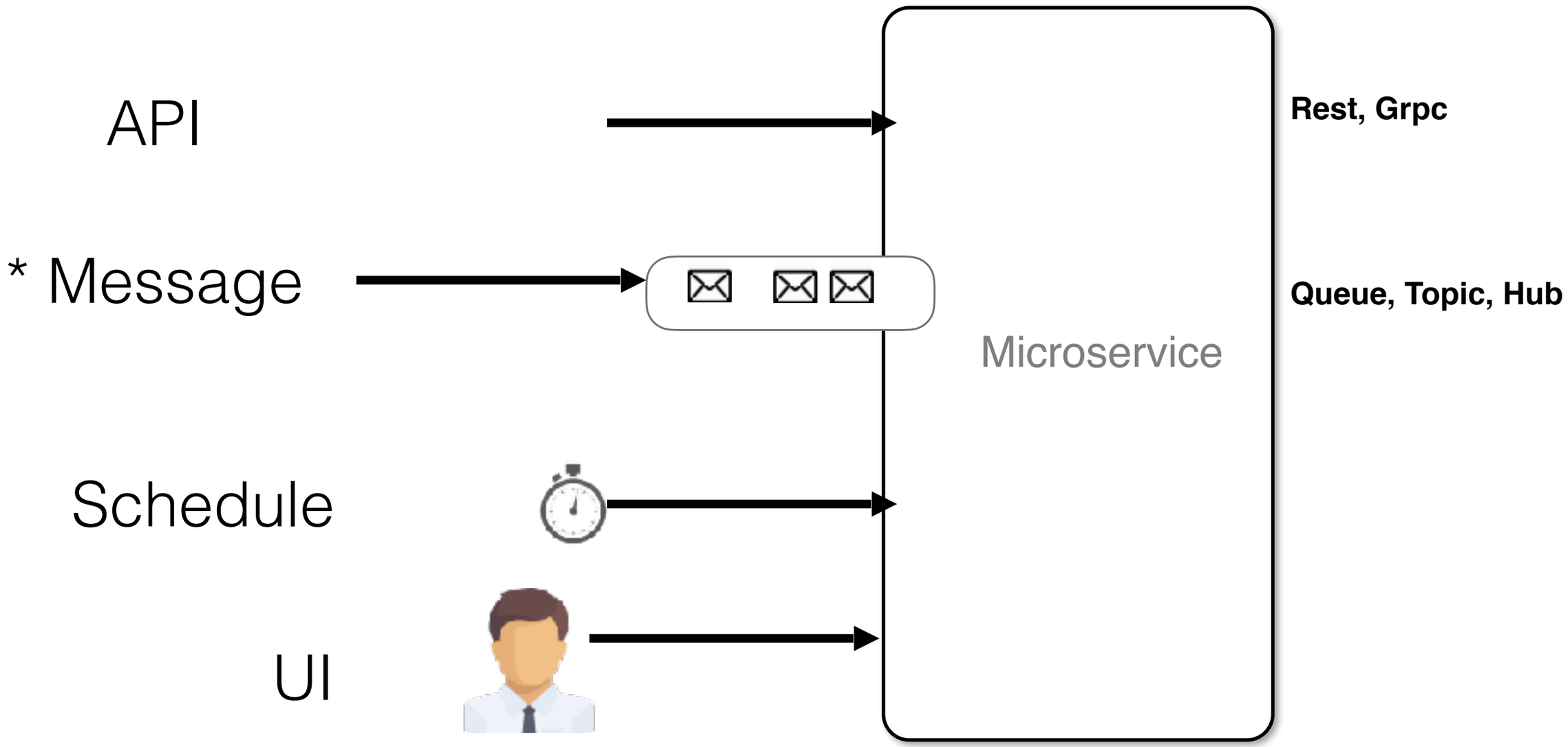
5


Schedule Layer
Domain Layer
<<feature5>>
Data Layer

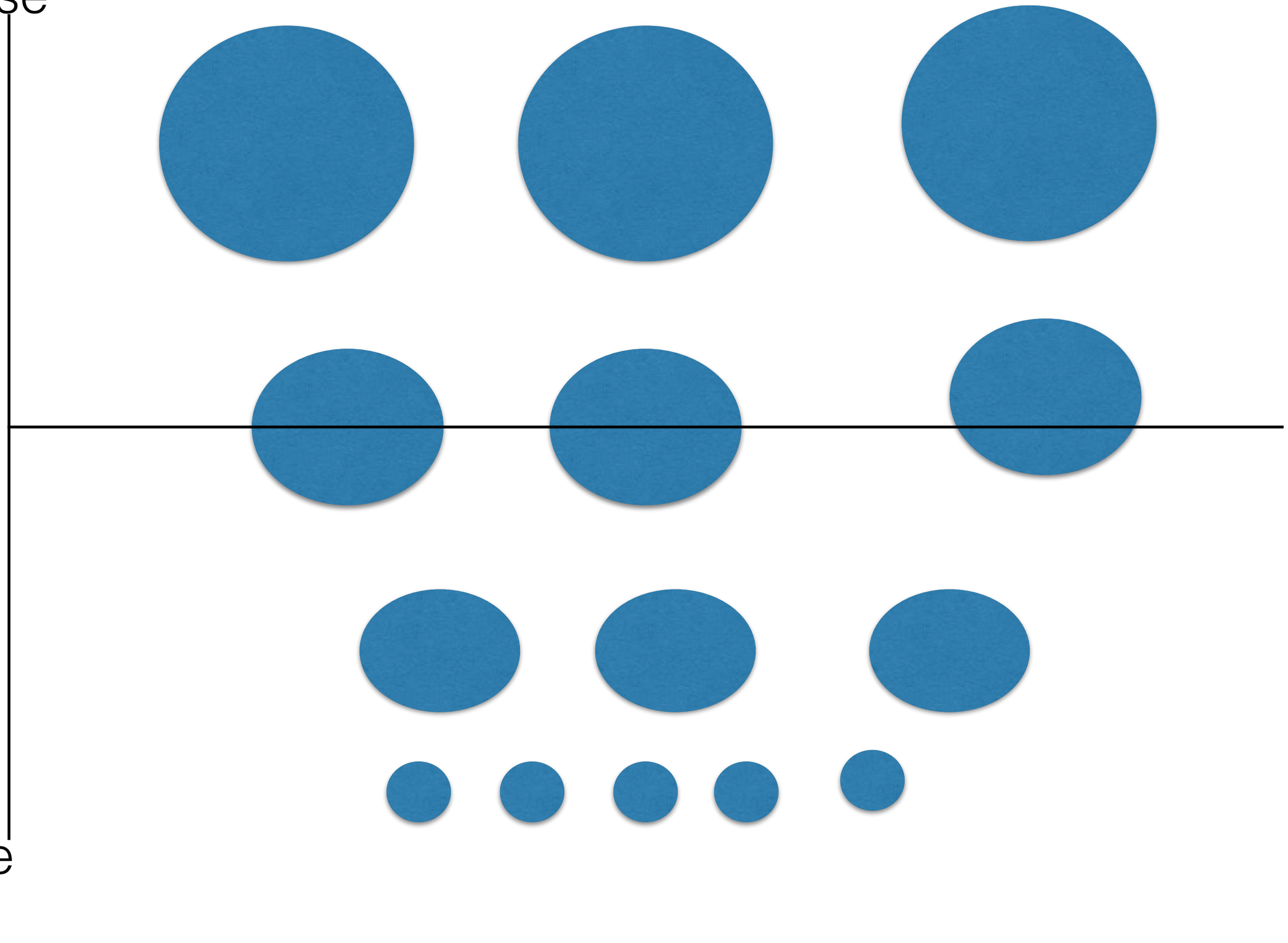
6


Rest
Web Server
API Layer
Domain Layer
<<feature6>>
Data Layer

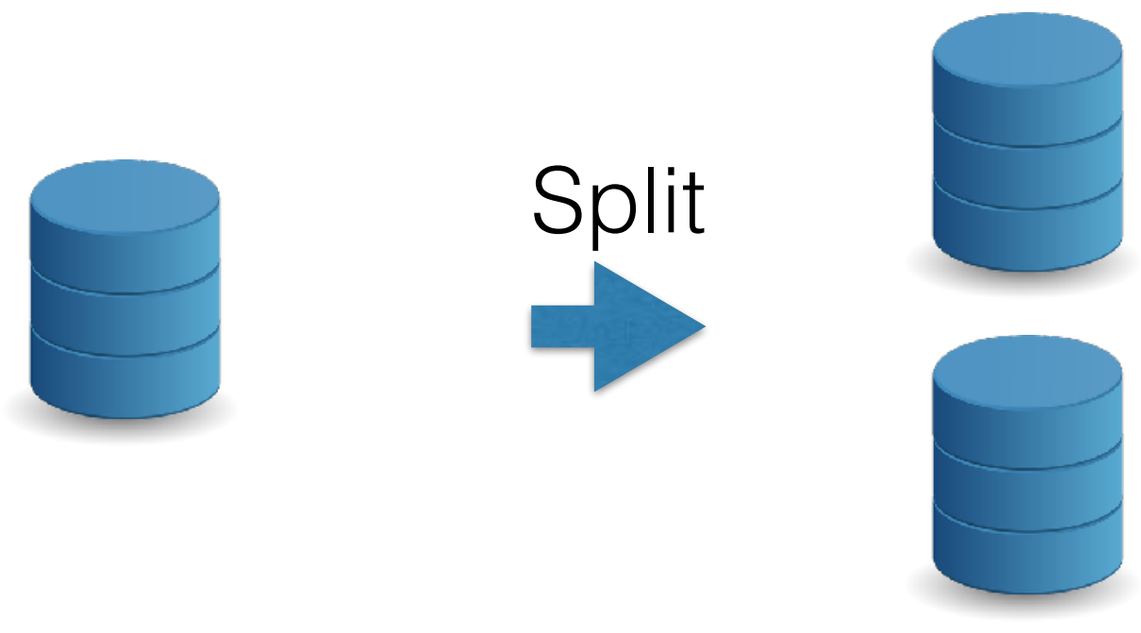
Hexagonal Architecture

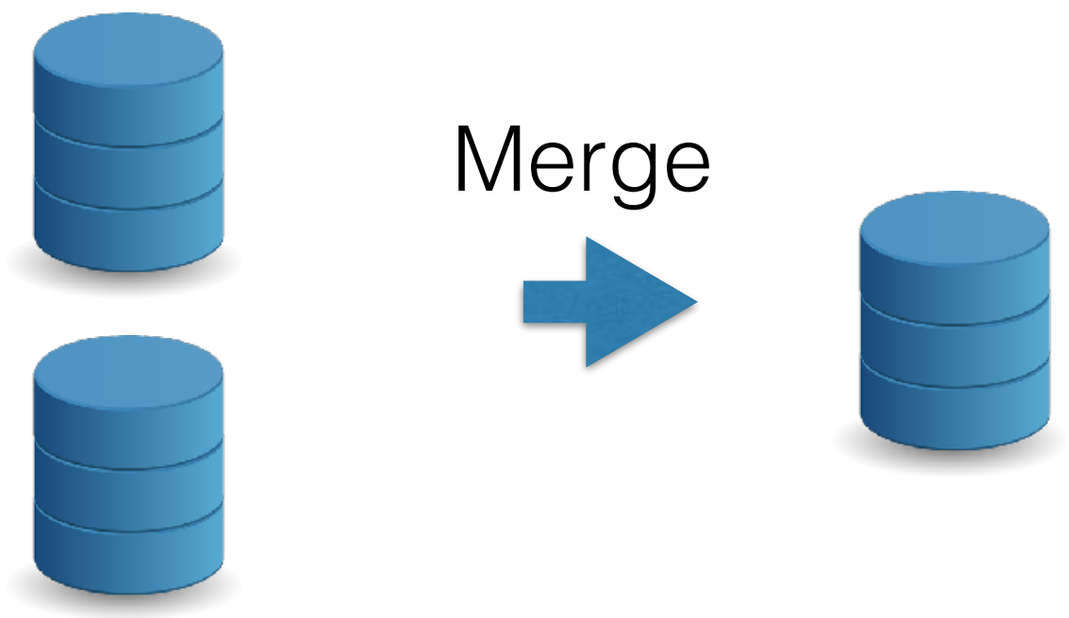


Coarse



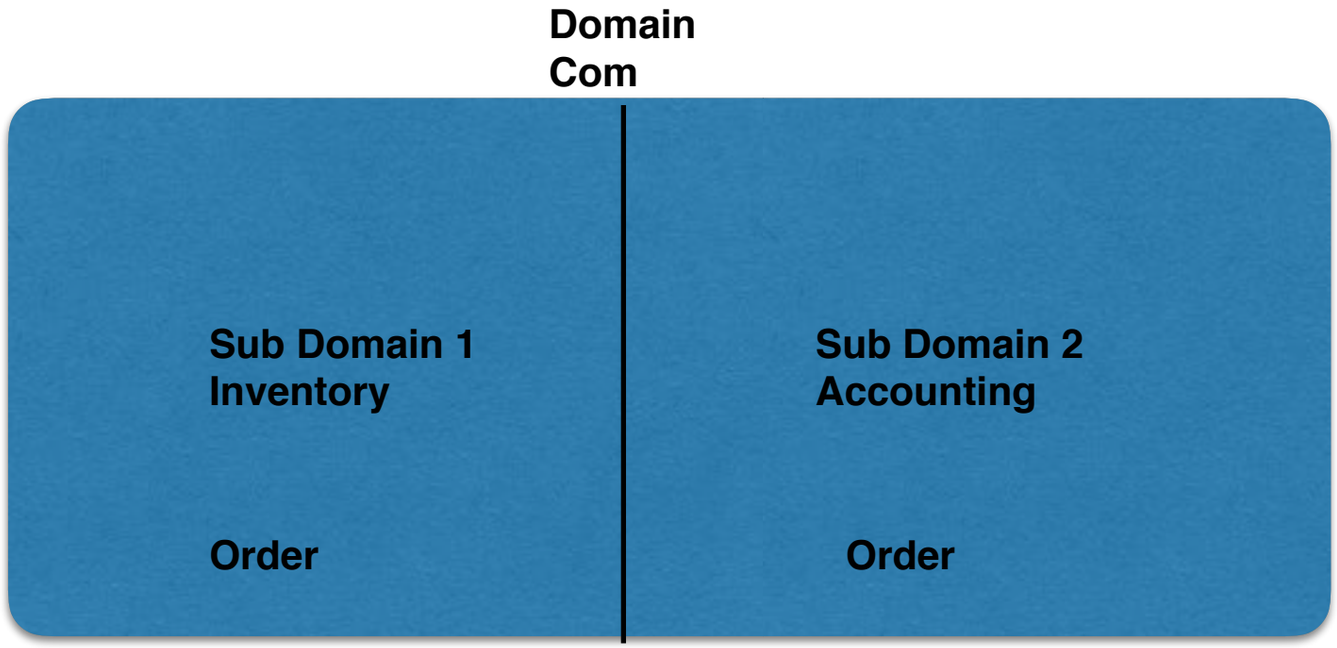
Fine


Split


Merge

Feature

Bounded Context (DDD)

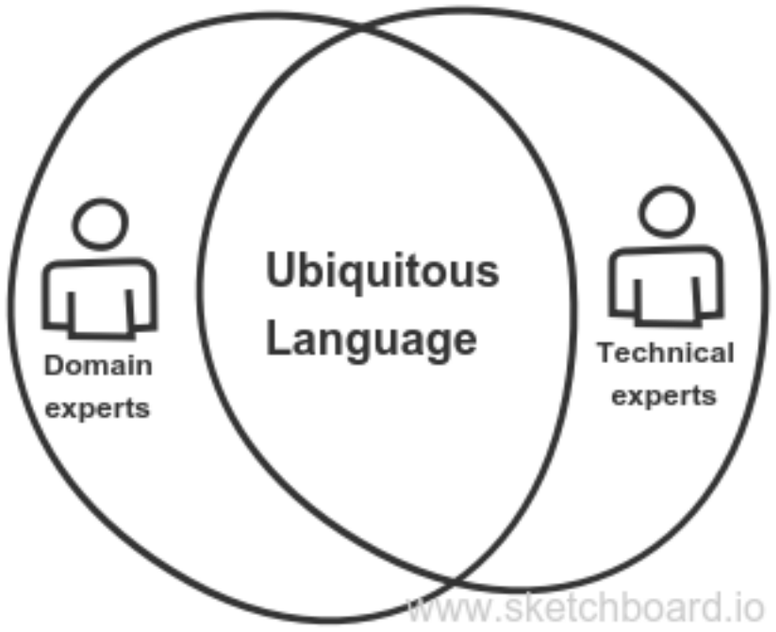


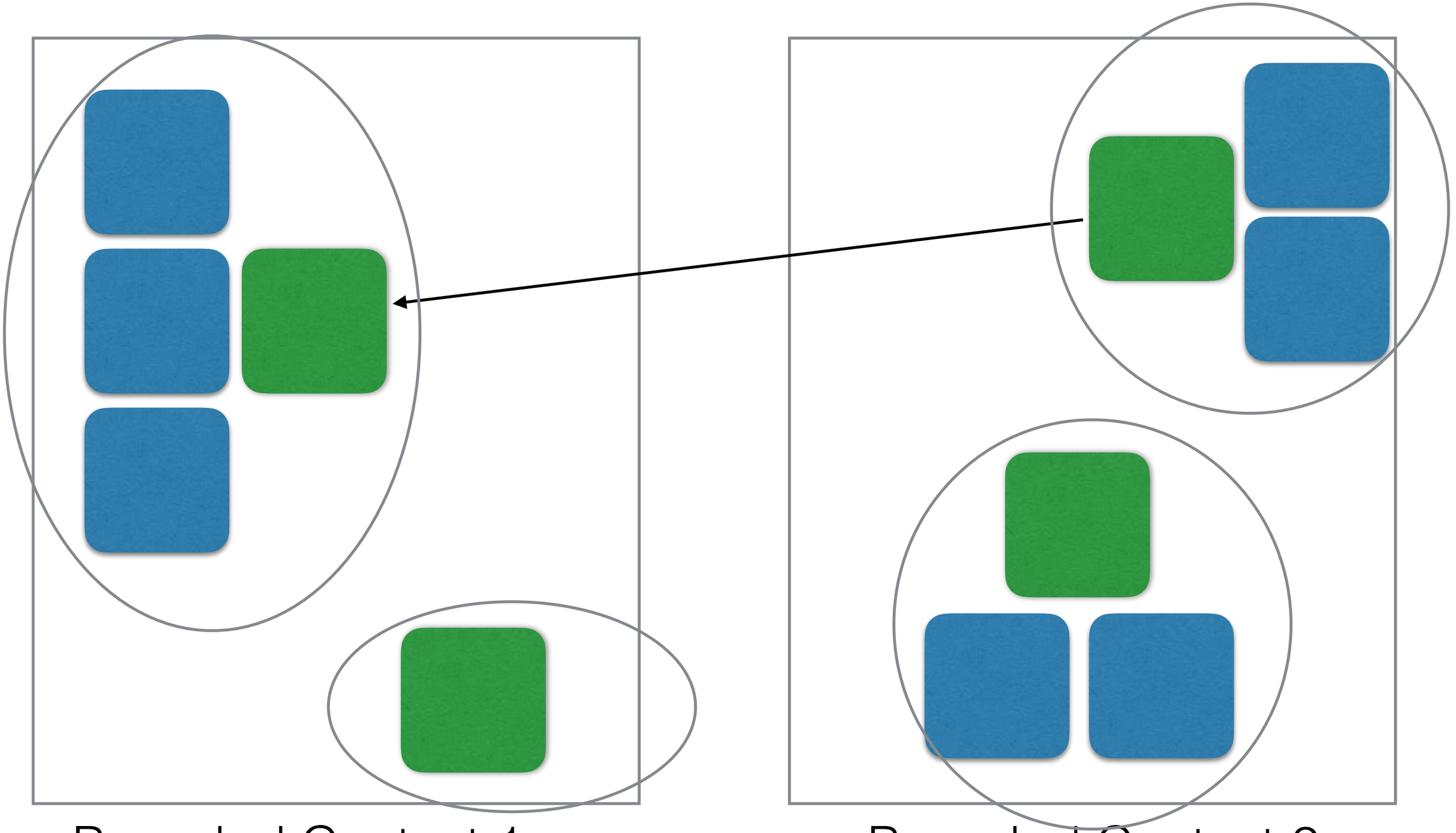
*Transaction Boundary

Aggregate (DDD)

DDD

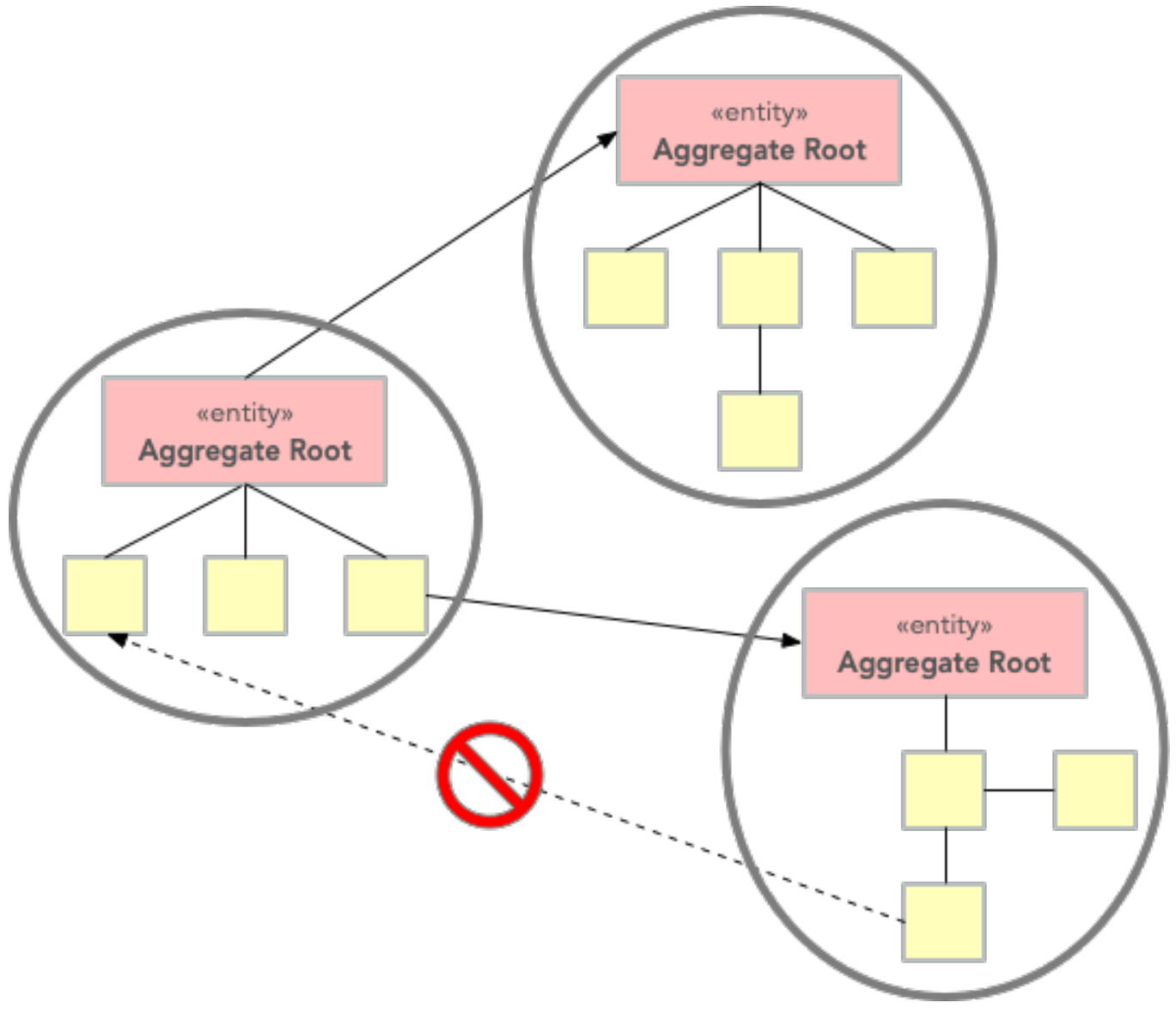
Ubiquitous language
Bounded Context
Aggregates





Bounded Context 1

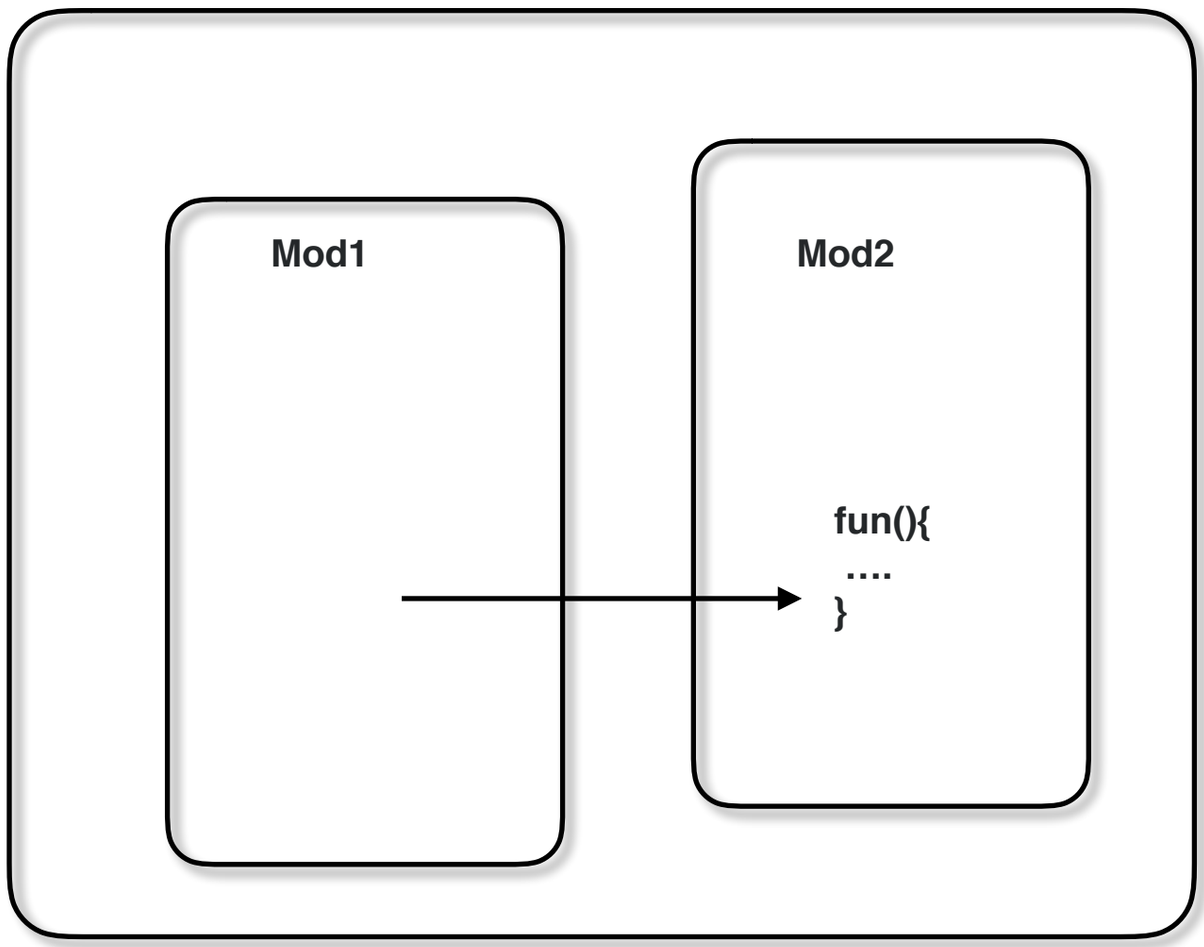
Bounded Context 2


«entity»
Aggregate Root
«entity»
Aggregate Root
«entity»
Aggregate Root

	Pros/ Cons	Solution
Development time	- -	
Learning Curve	- -	
Resource Performance (CPU, Memory, I/O)	- -	Message Q, grpc, thrift, protnbuf
Db Transaction Management	- -	SAGA
Views / Report / Dash board/ join	- -	Materialised View
Infra Cost	- -	Container
Initial Deployment effort	- -	Automation (IAC)
Debugging, Error Handling,	-	
Integration Test	- -	
debug/ error Log Mgmt	- -	Centralize
Config Mgmt	- -	Centralize
Authentication	- -	Centralize
Authorization	- -	Centralize
Audit Log mgmt	- -	Centralize
Monitoring / Alerting	- -	Centralize
Data Security and Privacy	- -	
Build Pipeline (CI)	- -	Automation
Agile Architecture (Agility to change)	++	
Feature Shipping (Agility to ship)/ CD	++	
Scalability (volume - request, data,	++	
Resilence	++	
Availability	++	
Ability to do Polygot	++	
Maintainability (easy to change)	++	

	Cpu Cycles
a + b	3
Fun call	10
CreateThread	2,00,000
Destroy thread	1,00,000
Db call	40,00,000
Api Call	10, 00, 000

Process

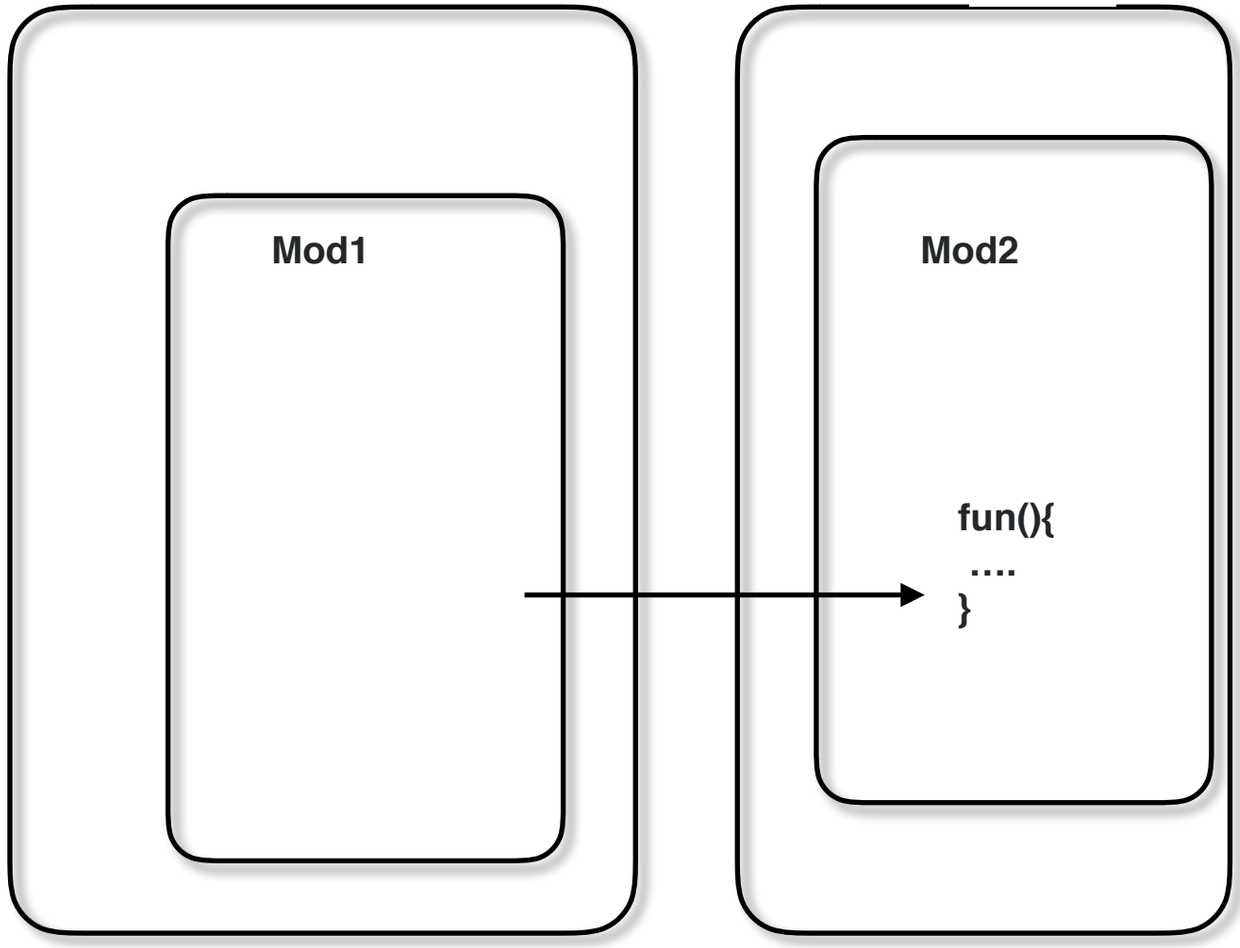


10 cpu cycles

Process

10,00,000 cpu cycles

Process



YAGNI - you aren’t gonna need it

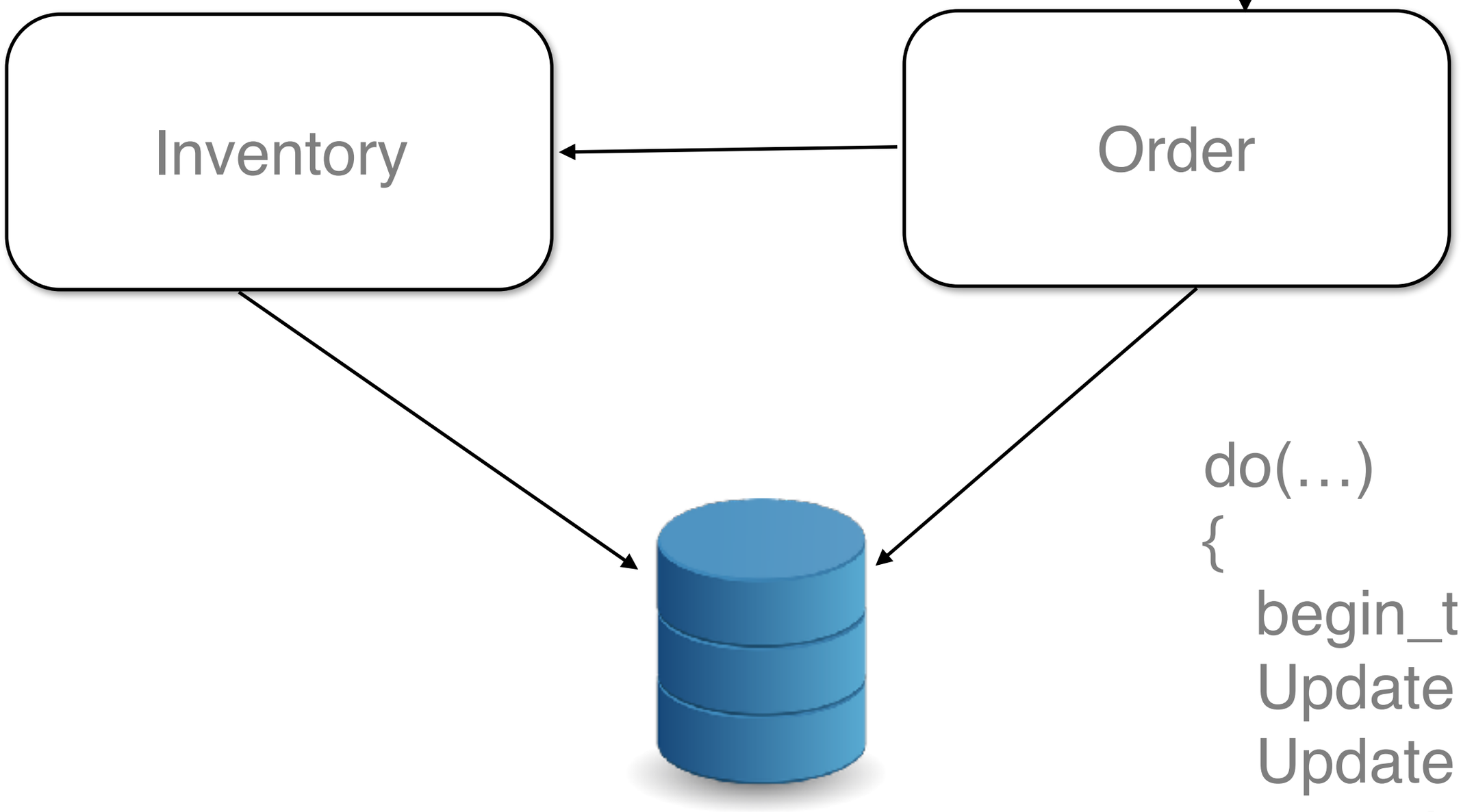
KISS

Solve tomorrow’s problem tomrrow

1. Performance

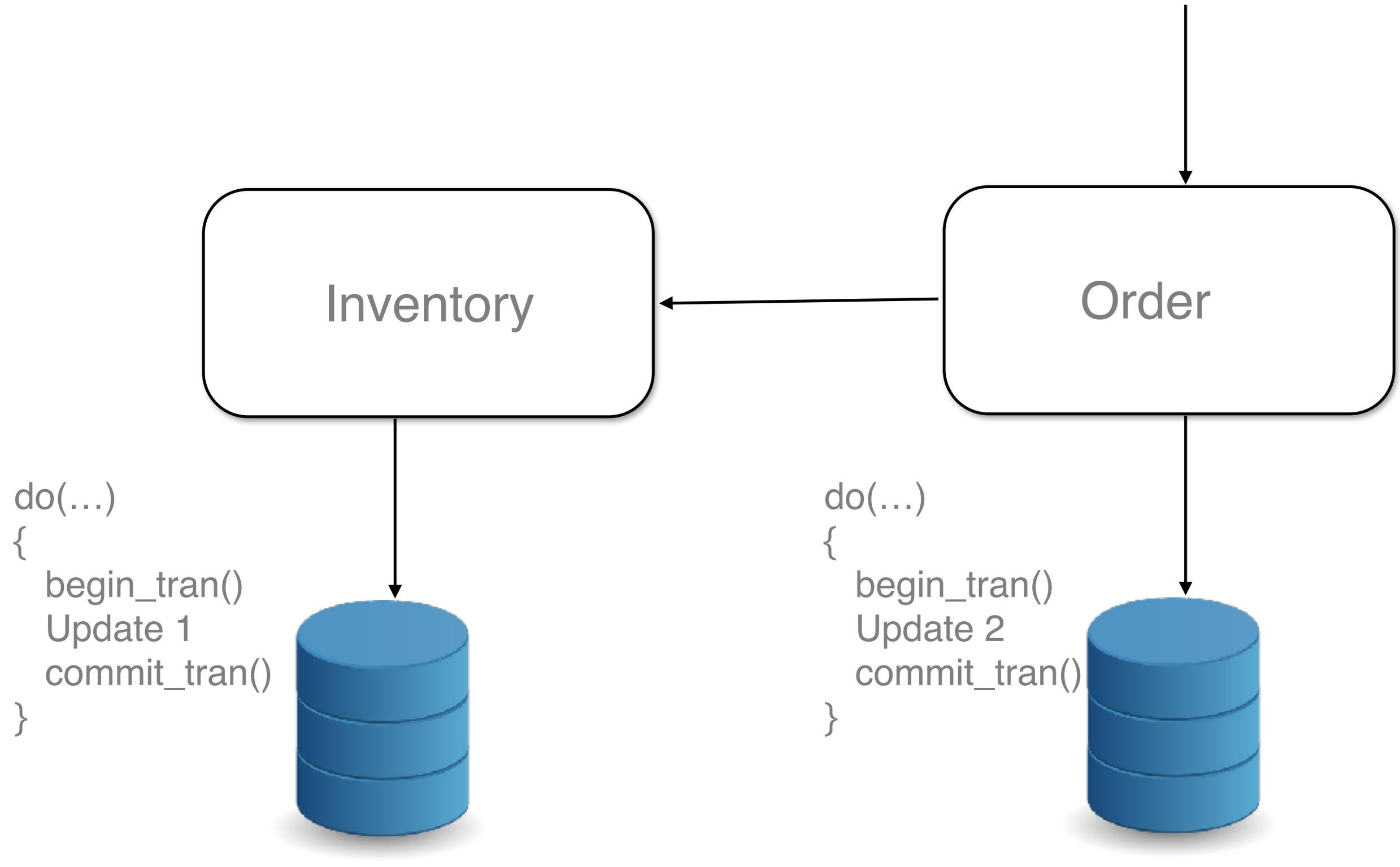
Operation	Cpu Cycles
• 10 + 12	5
• Calling a in-memory Method	10
• Create Thread	2,00,000
• Destroy Thread	1,00,000
• Database Call	40,00,000
• Distributed Fun Call	20,00,000

2. Db transaction



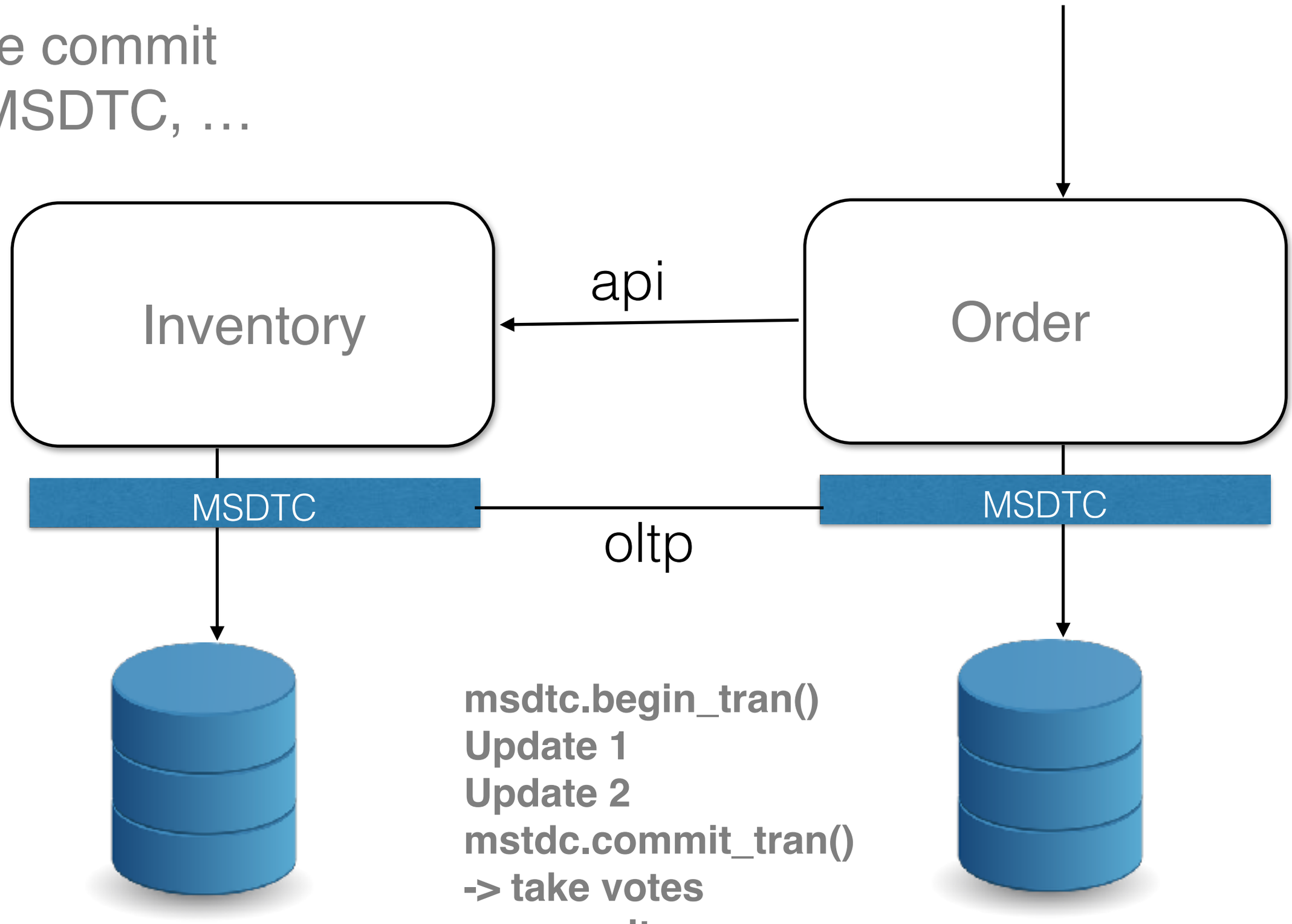

```
do(...)
{
   begin_tran()
   Update 1
   Update 2
   commit_tran()
}
```

2. Db transaction



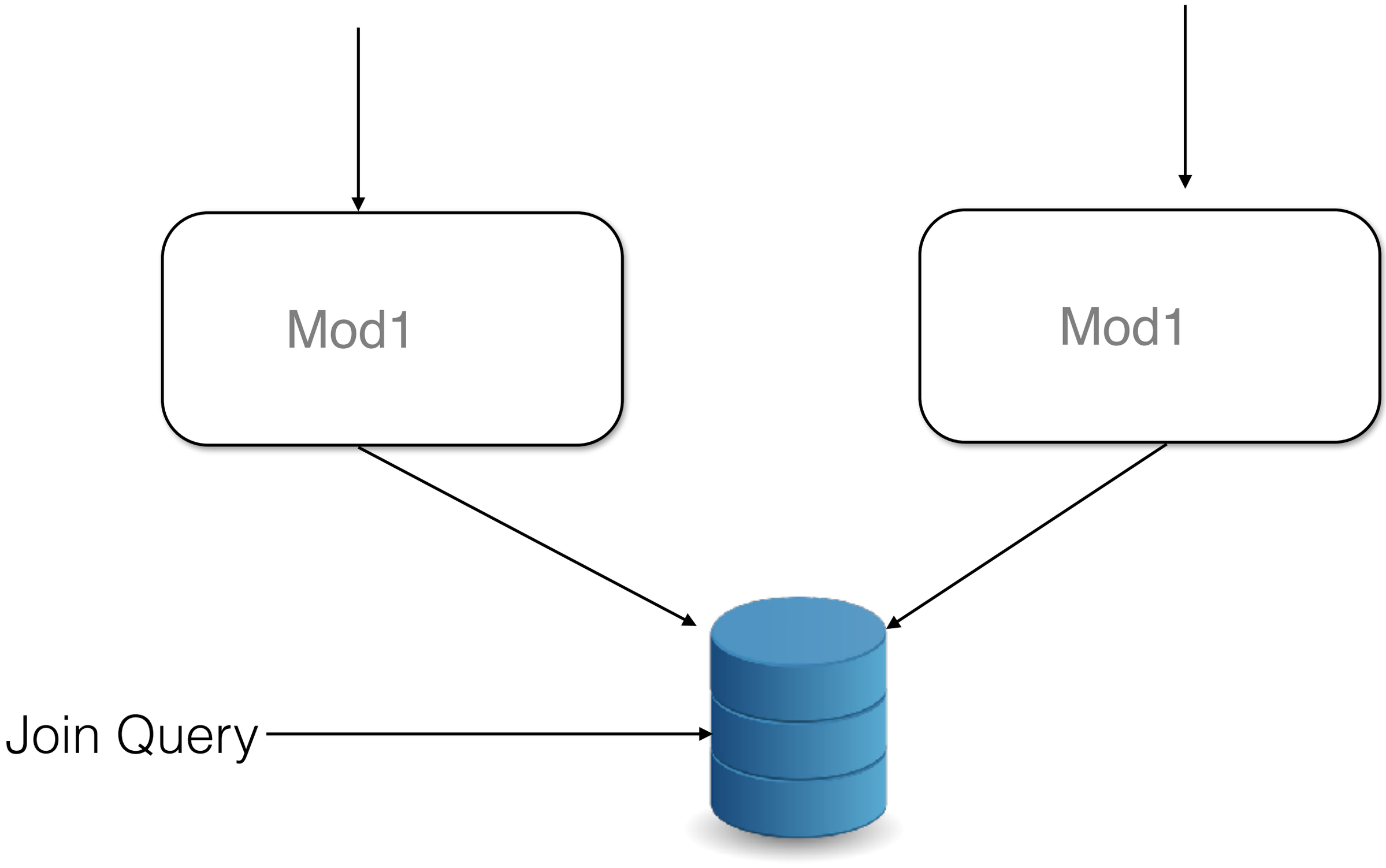
2. Db transaction

2 phase commit
JTX , MSDTC, …

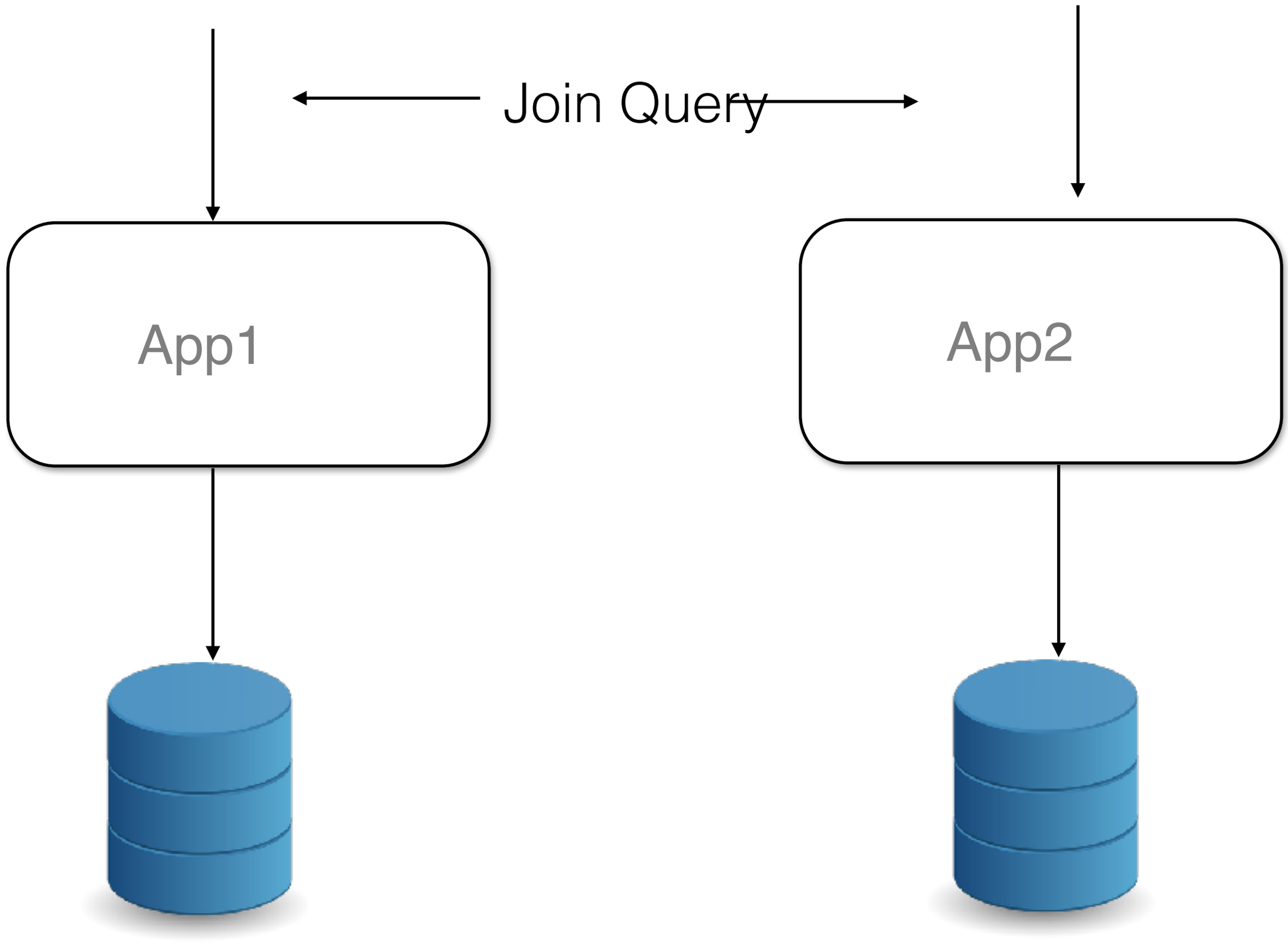


msdtc.begin_tran()
Update 1
Update 2
mstdc.commit_tran()
-> take votes
-> commit

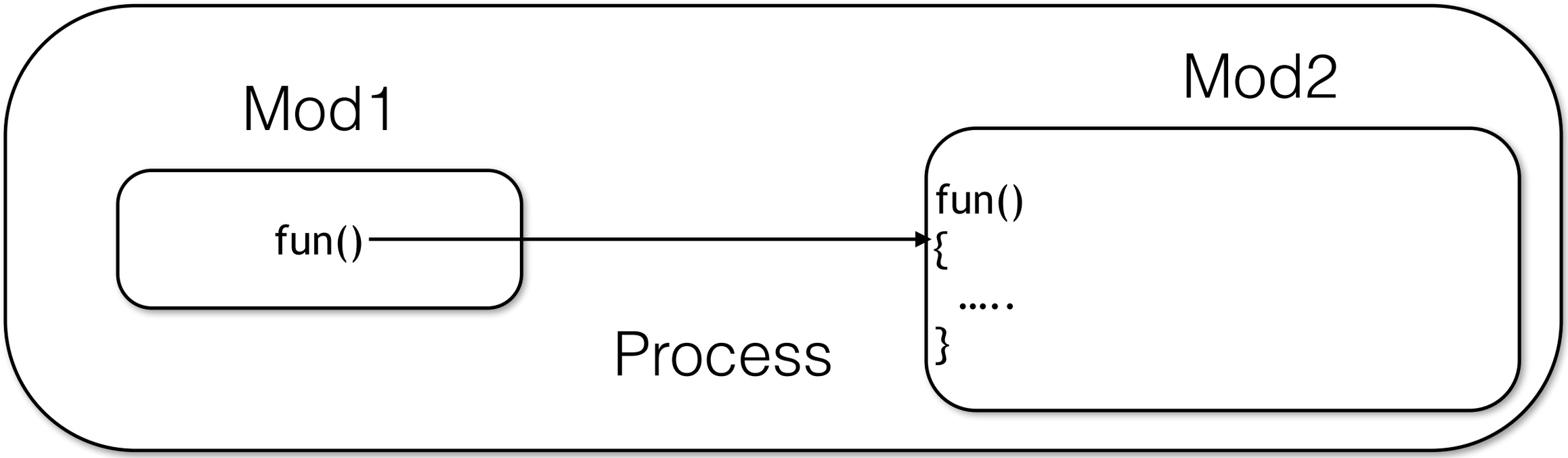
3. Db query

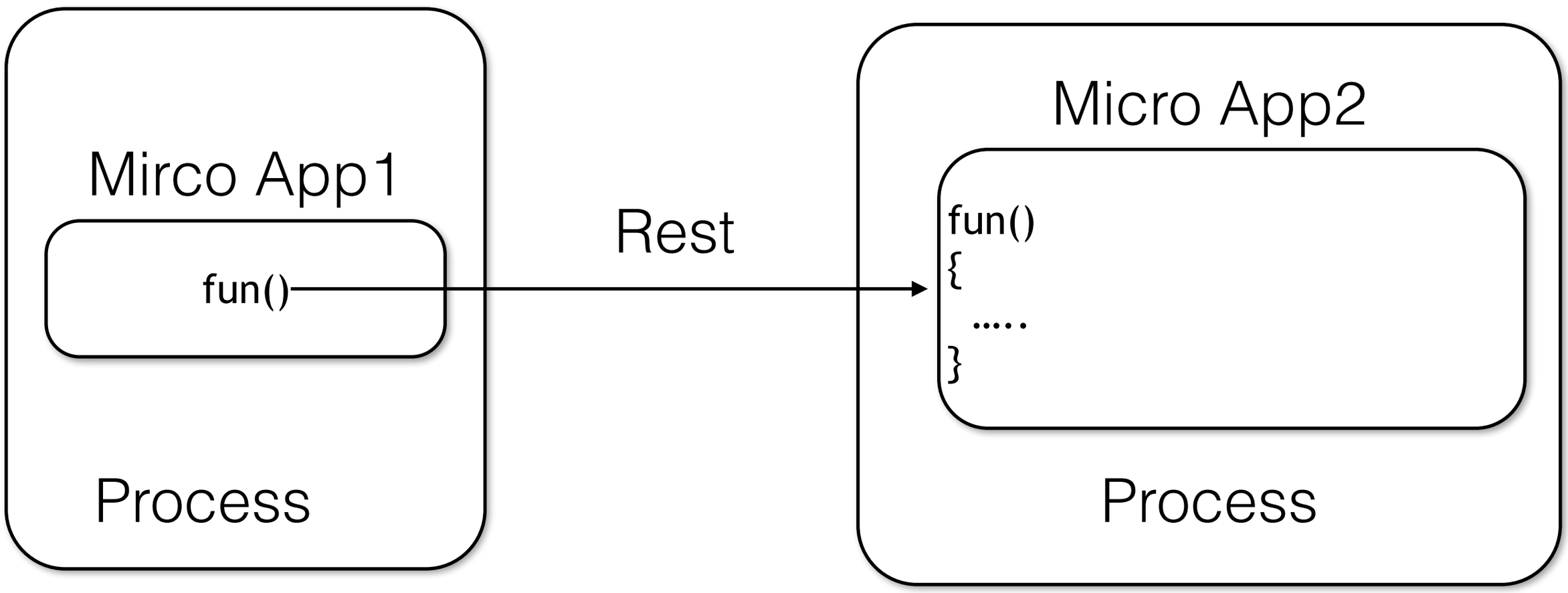


3. Query

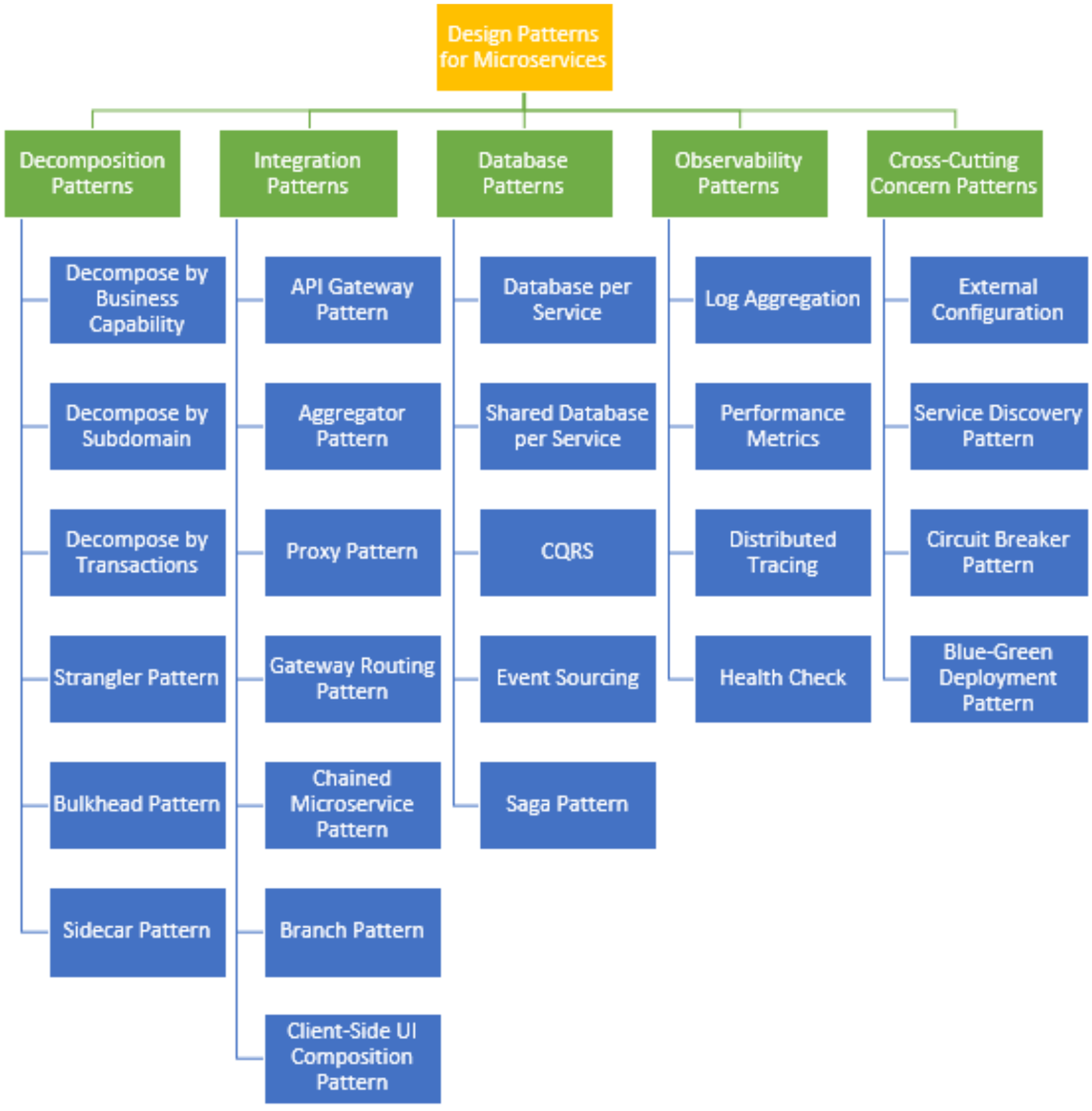


4. Development time

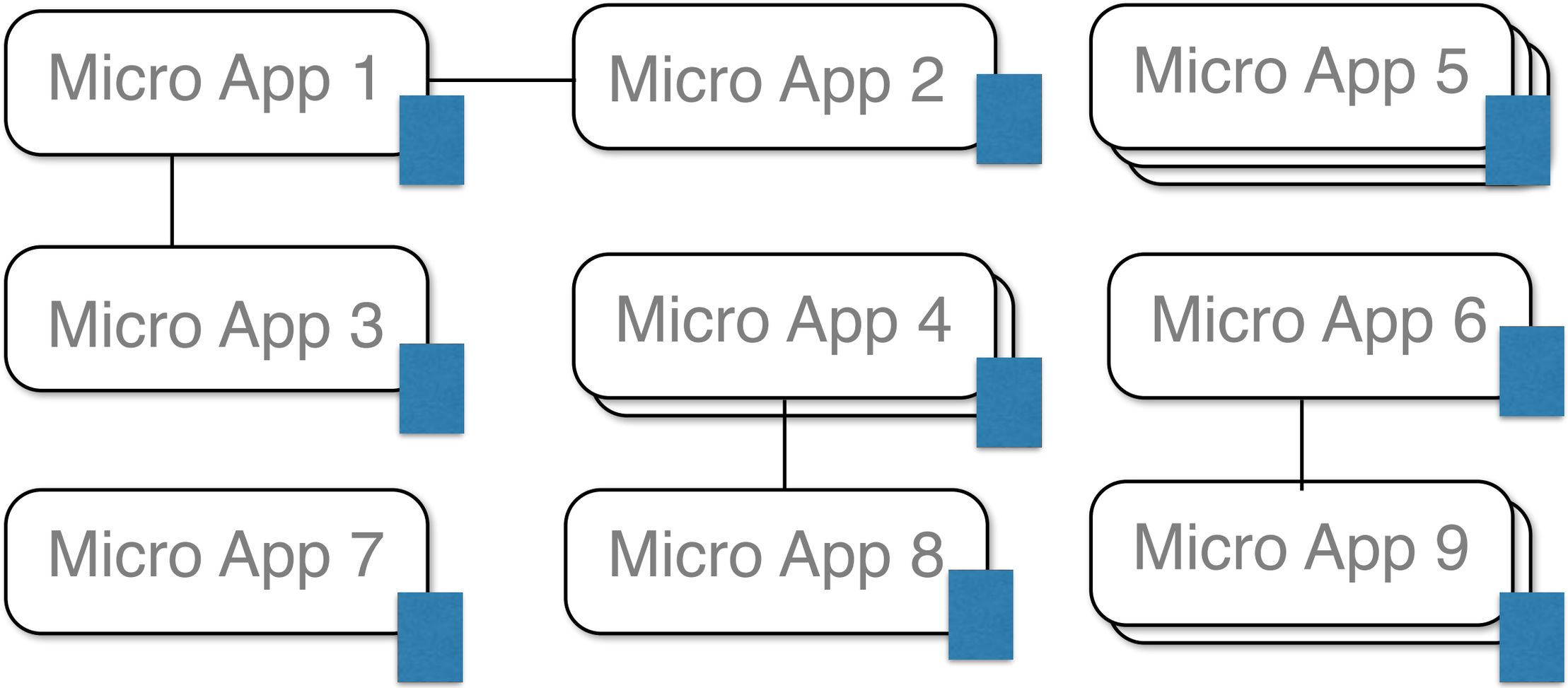




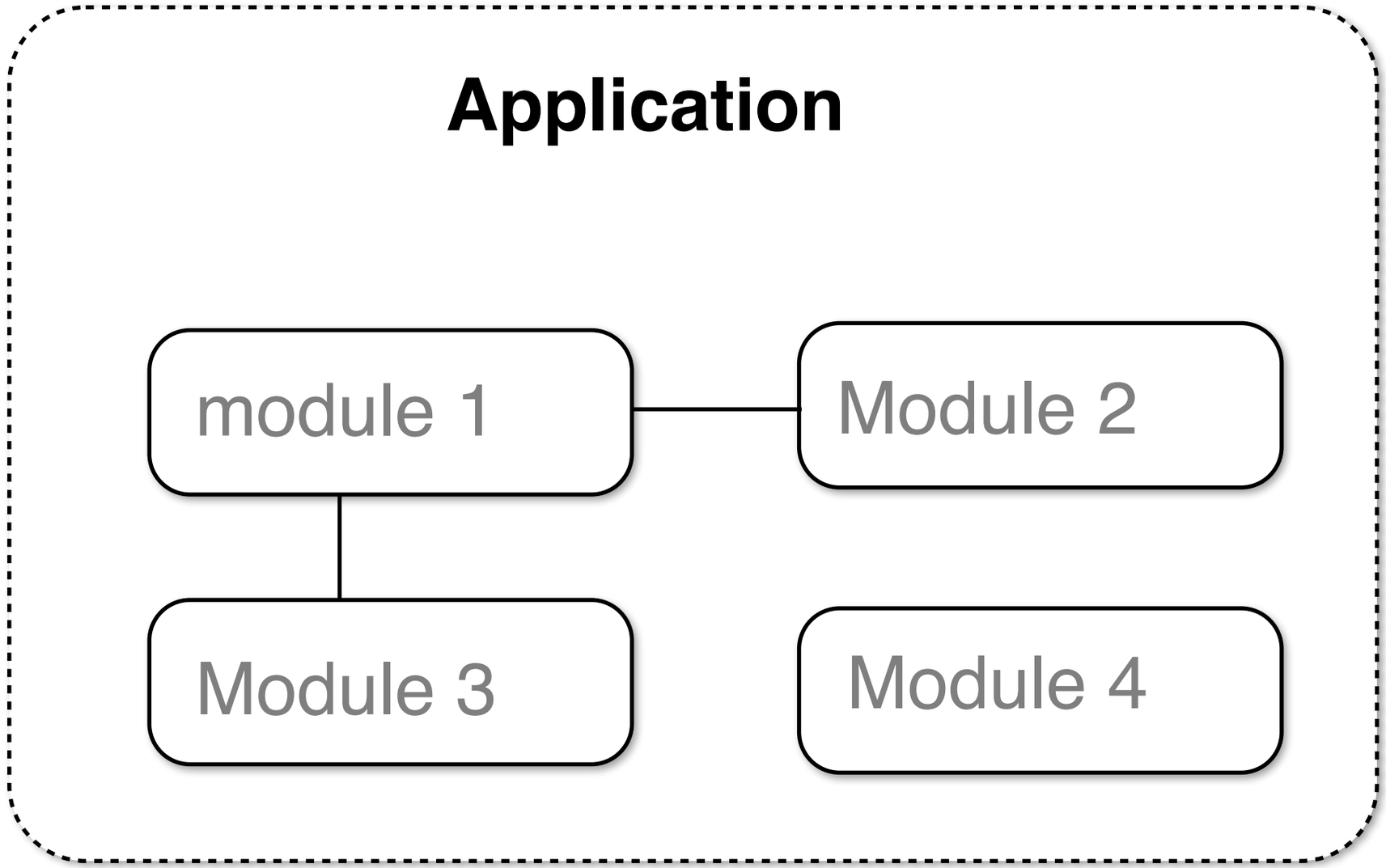
5. Learning Curve



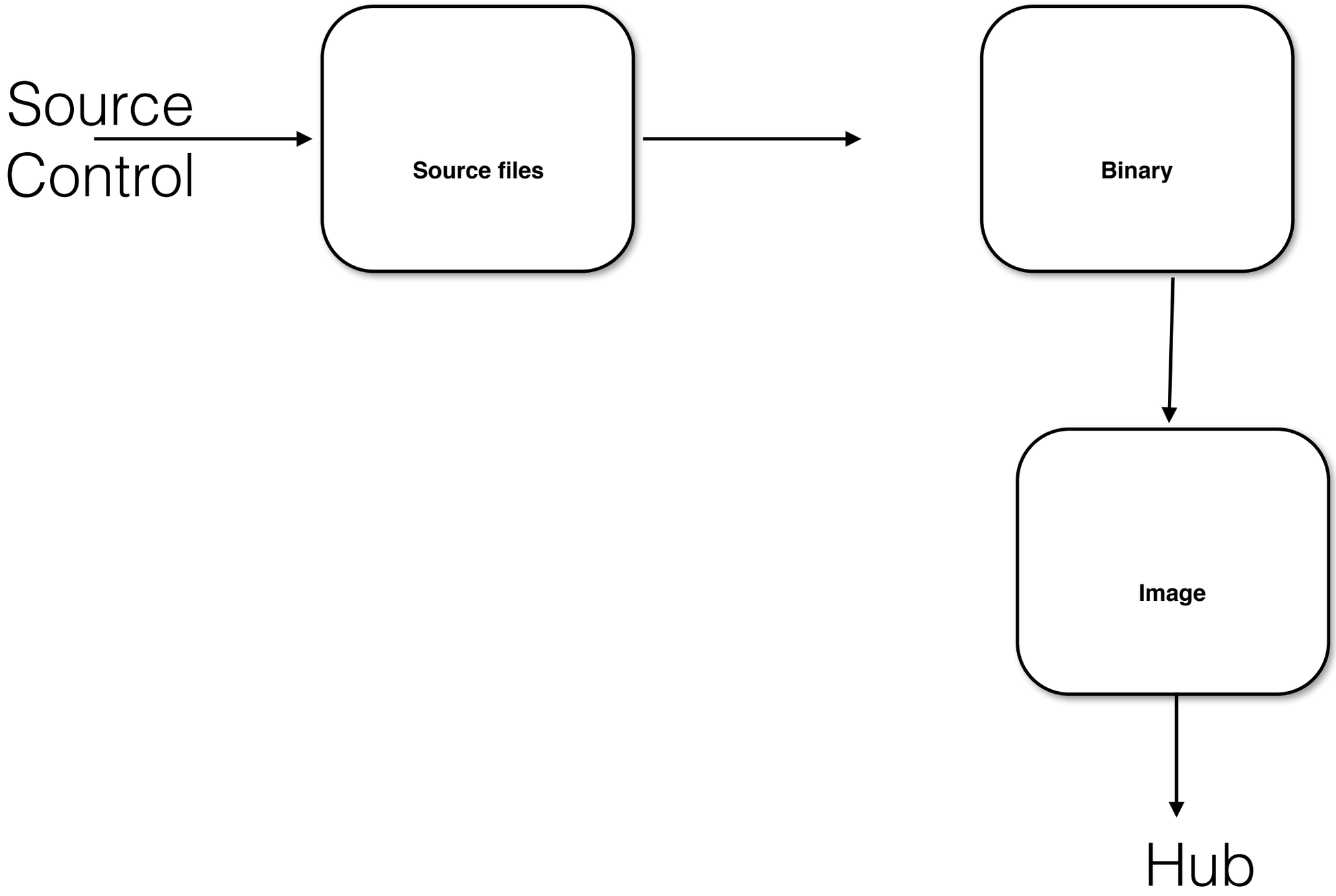
6. Infra Cost


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

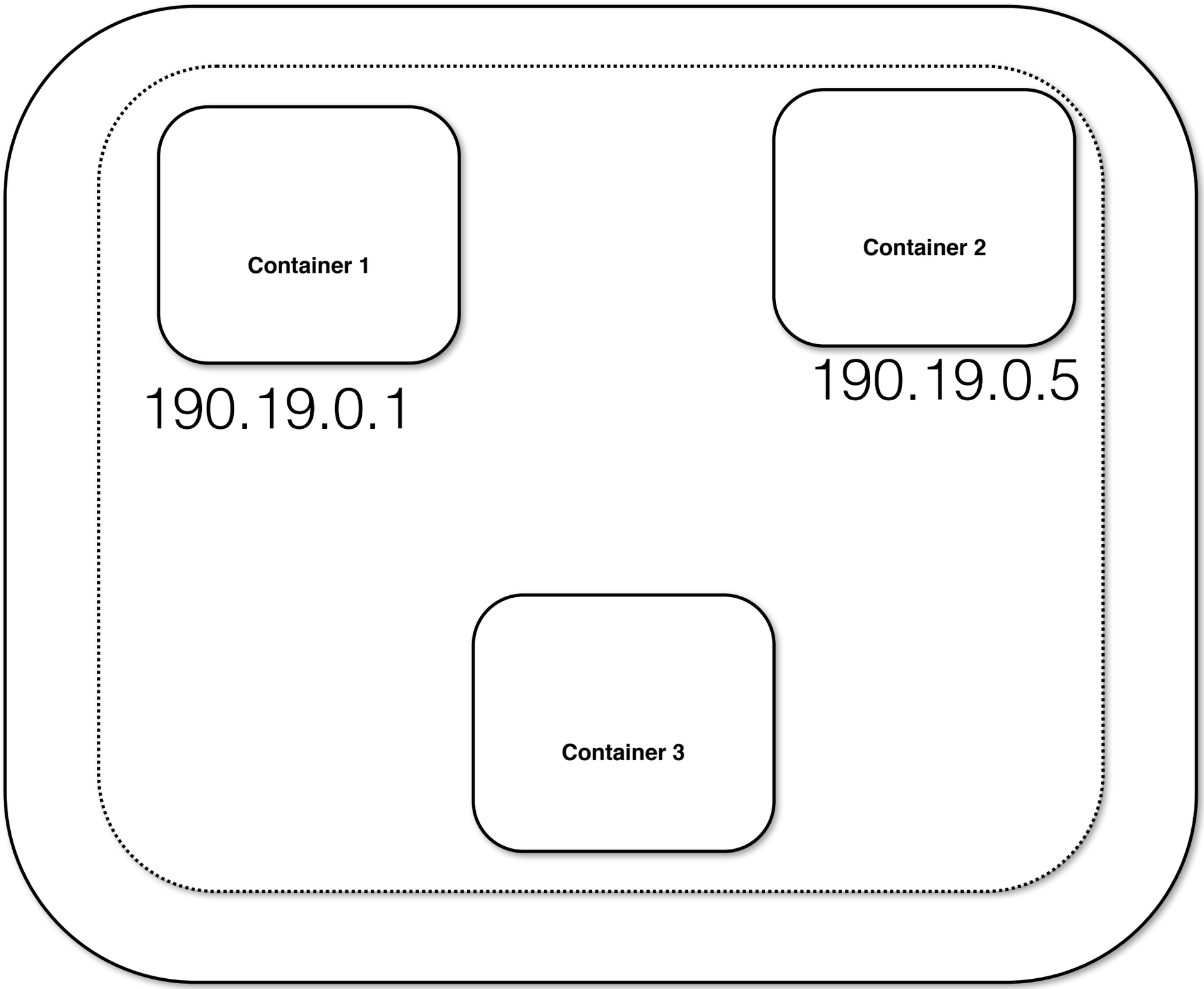
7. Debugging



Reference Arch


Source
Control
Source files
Binary
Image
Hub

VM



CQRS
Materialised View
SAGA - compenstable transaction
Eventual Consistency
Event Driven Architecture
DDD
Bounded context
Aggregate
Event Sourcing
ubiquitous language
Command Message
Event Message

Log mngmt
config mngmt
devops

Portal

HTTPs REST + JSON (CRUD)

WAF

Reverse Proxy/ API Gateway

Authentication, Authorization

Stream Analytics

do() undo()

Stock Management Command

No Stocks

Create Order

Create Order Msg (CUD)

Grpc + protnbuf (R)

Domain Event Bus (command / Events)

Order Created

No Stocks

Create Order

Order Management Command

do() undo()

Create Order

Order Created

No Stocks

Order Management Query

do() undo()

Event Service

Data lake
Event Store
Audit
full state

Big Data Analytics

Data Ware House
snapshot (day/week)

weekly/ monthly

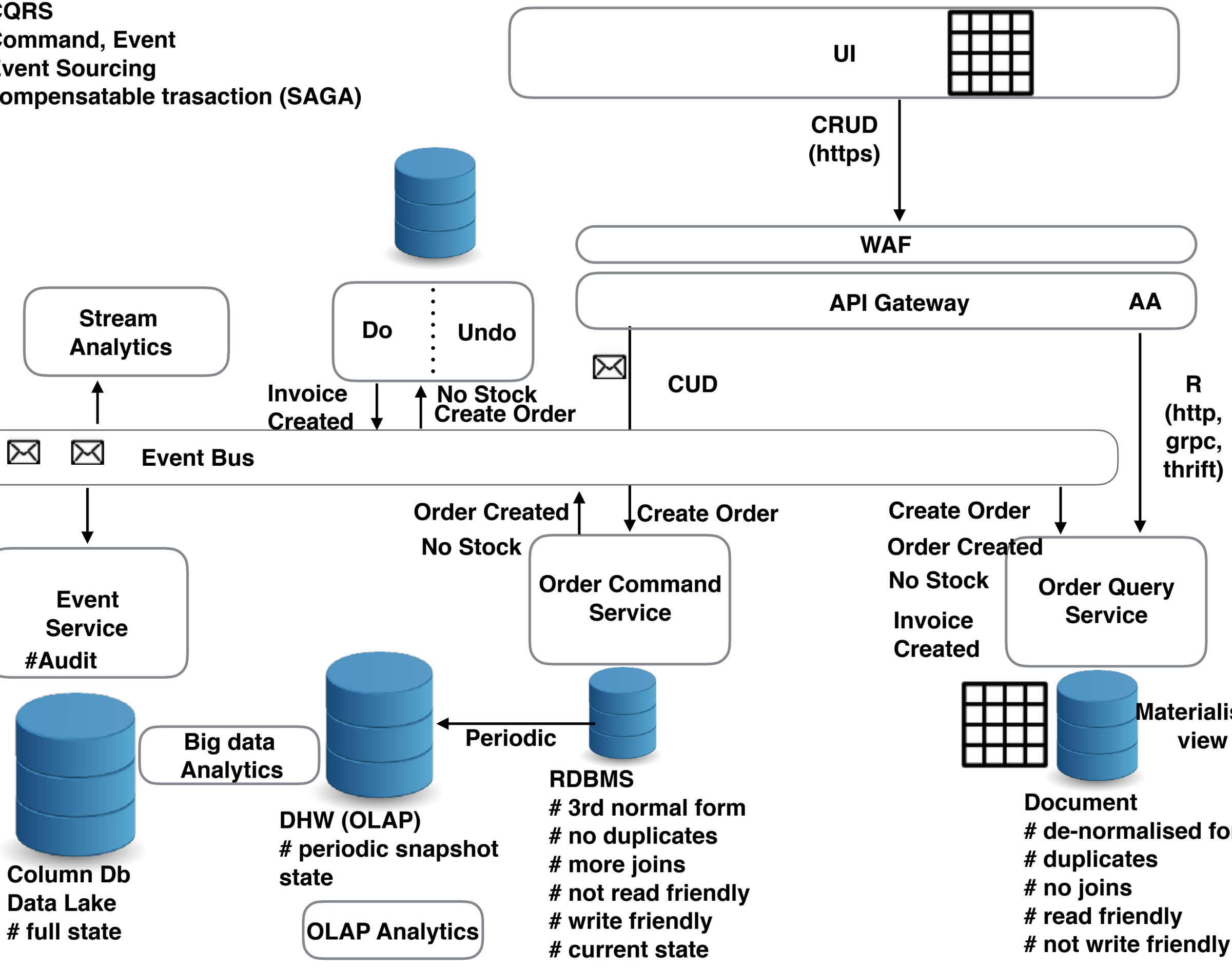
DWH Analytics

RDBMS
3NF
no duplicates, more joins
Write Friendly
not read friendly
current state

Document
DNF
duplicates, no joins
not Write Friendly
read friendly

- Util
- Handler
- Controller
- Manager
- Service
- Helper
- Mediator
- Executor
-

CQRS
Command, Event
Event Sourcing
compensatable trasaction (SAGA)



CQRS
materialised view
Hexagonal Arch
Event Driven Archiecture
Event Sourcing
SAGA
compensatale transaction

Portal

CRUD (rest)

API Gateway

do() & undo()

Account Service

Stream Analytics

CUD

R (grpc)

Cmd Create

Event No Stocks

Domain Event Bus

Event Created

Event No Stocks

Cmd Create

Event Audit Service

Order command Service

Order Query Service

Event Store
(full state of domain)

Big Data

Batch Analytics

Rdbms

write friendly

Document

read friendly
materialised view

Order Query Service

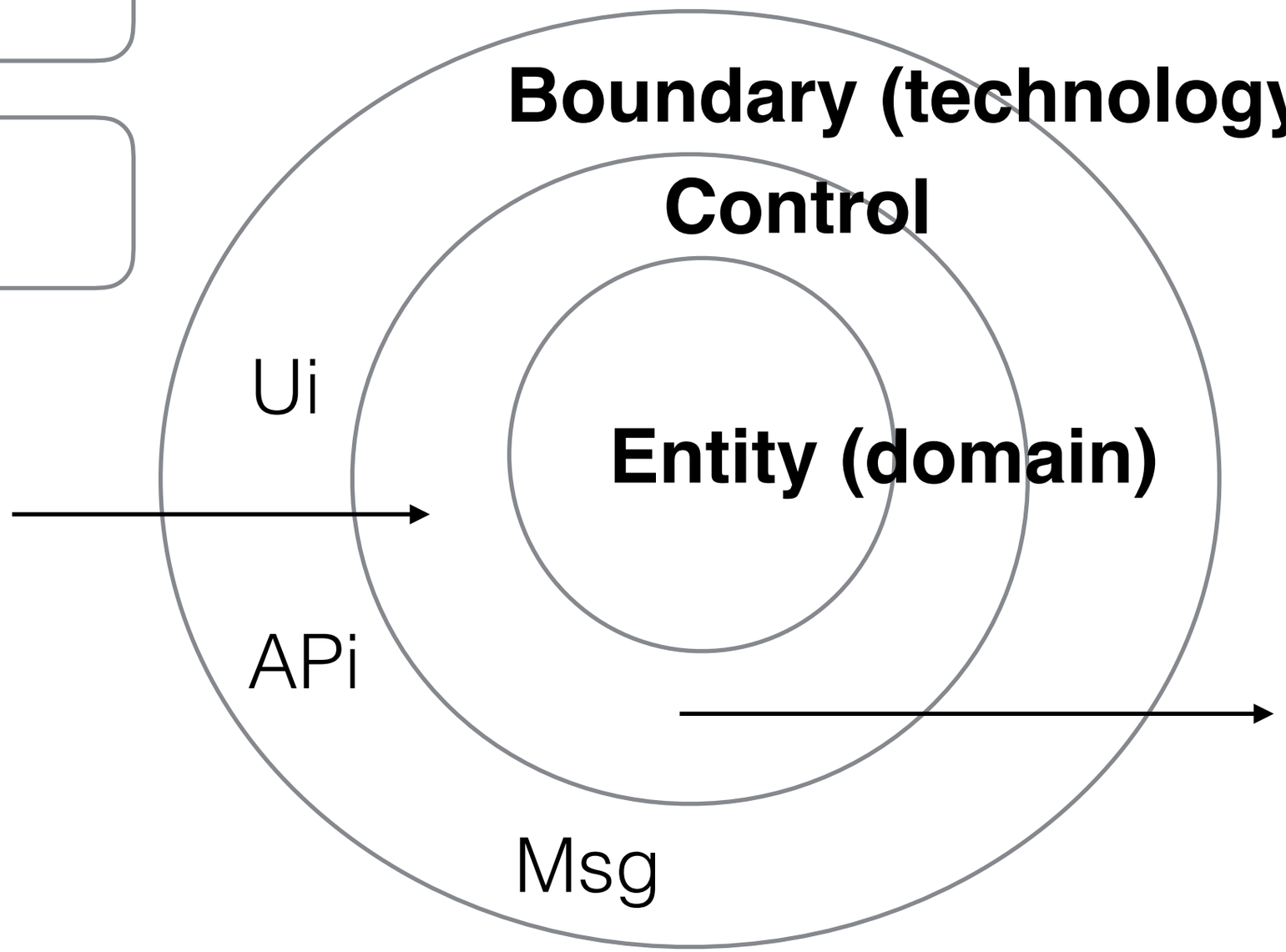
Message Service

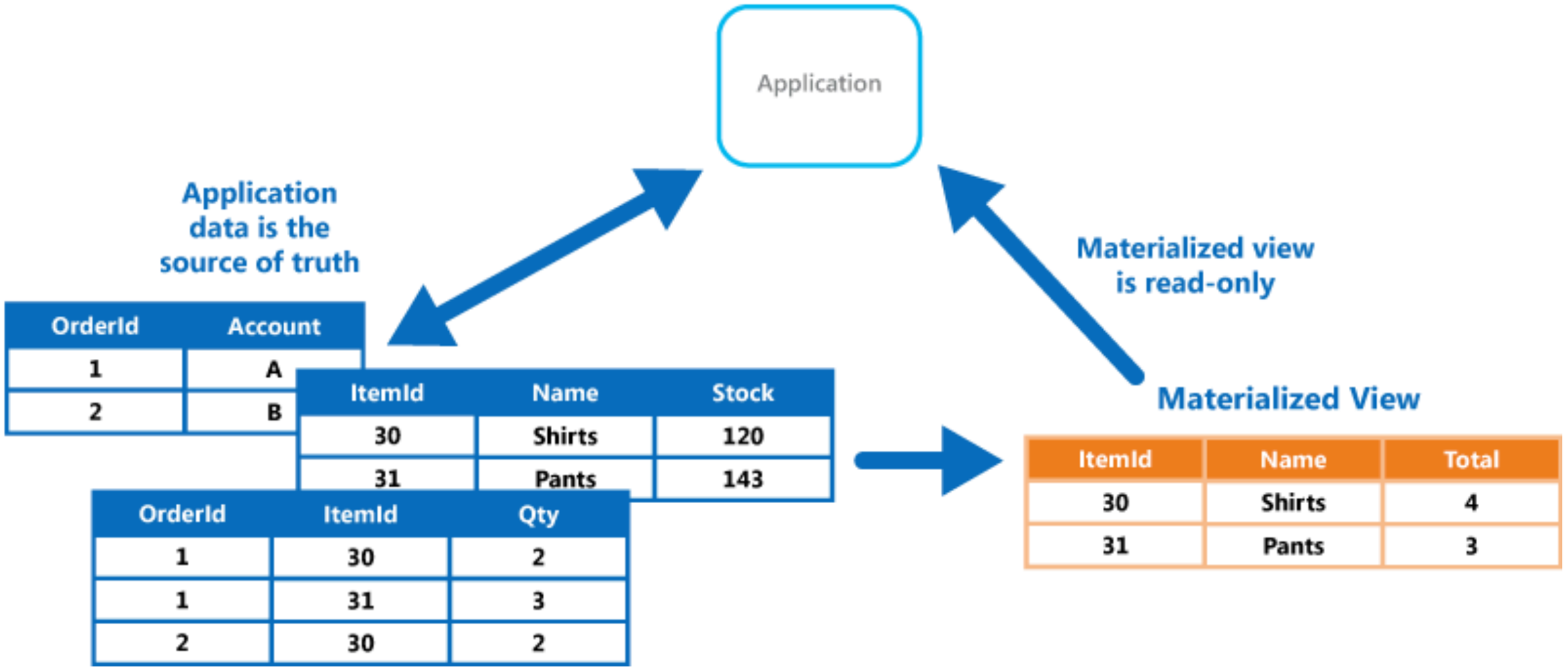
APi Service

Facade/ Control

Repository

Domain




Application
Application data is the source of truth
Materialized view is read-only
Materialized View
OrderId
Account
1
A
2
B
ItemId
Name
Stock
30
Shirts
120
31
Pants
143
OrderId
ItemId
Qty
1
30
2
1
31
3
2
30
2
ItemId
Name
Total
30
Shirts
4
31
Pants
3

DNF

TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

<— duplicates
<— no joins

3NF

TABLE_BOOK

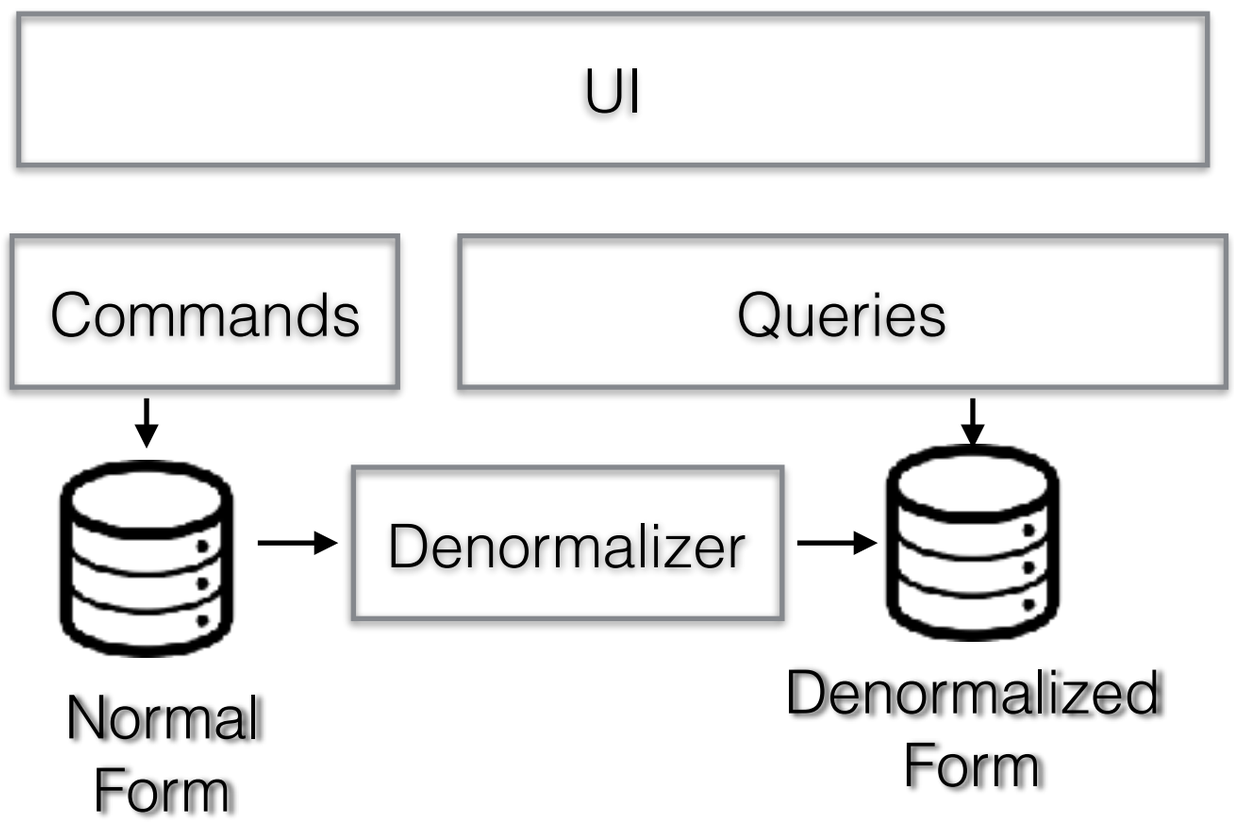
Book ID	Genre ID	Price
1	1	25.99
2	2	14.99
3	1	10.00
4	3	12.99
5	2	17.99

TABLE_GENRE

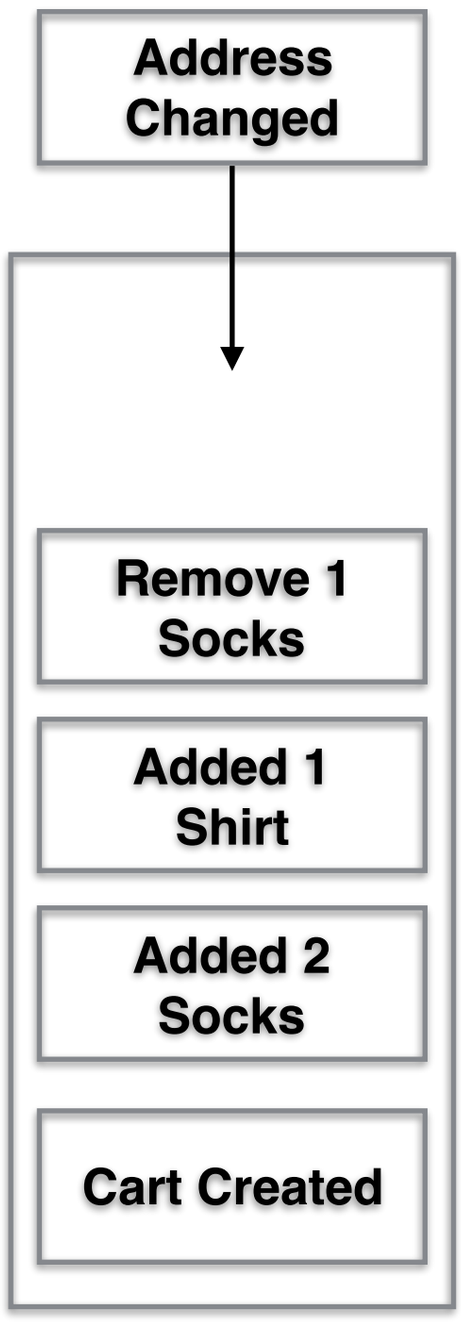
Genre ID	Genre Type
1	Gardening
2	Sports
3	Travel

<— no duplicates
<— more joins

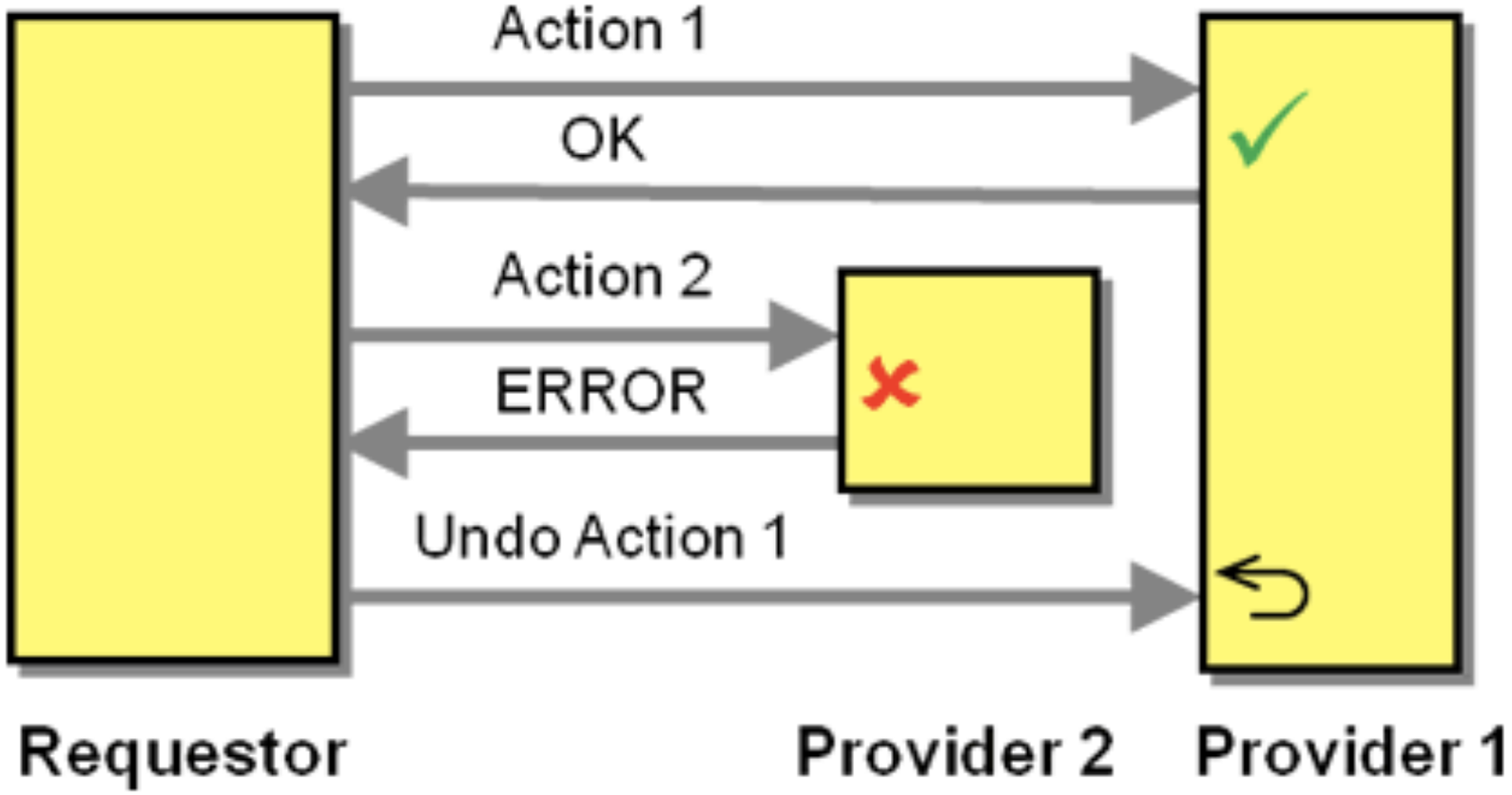
CQRS


UI
Commands
Queries
Denormalizer
Normal Form
Denormalized Form

Event Source

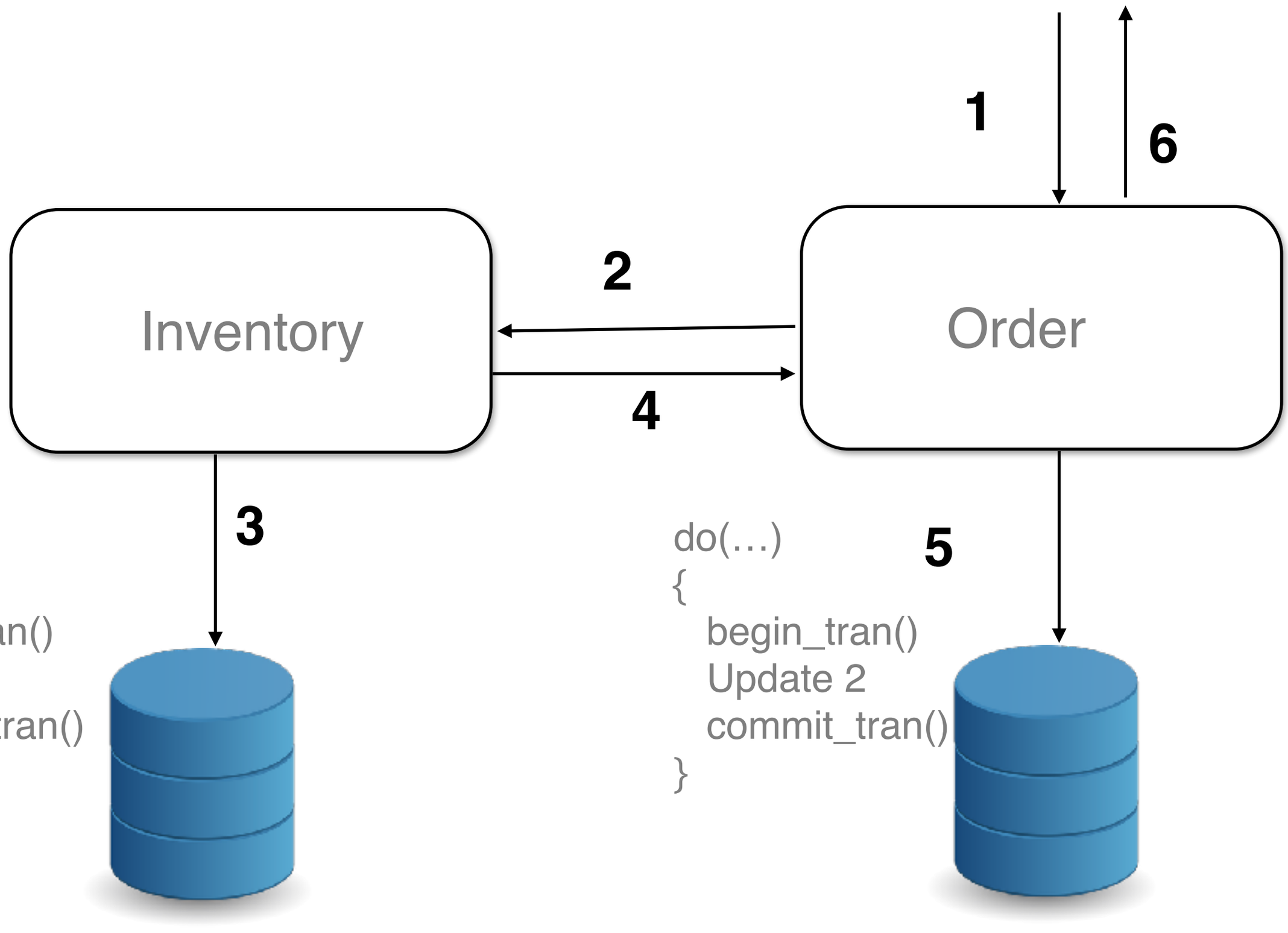


Saga

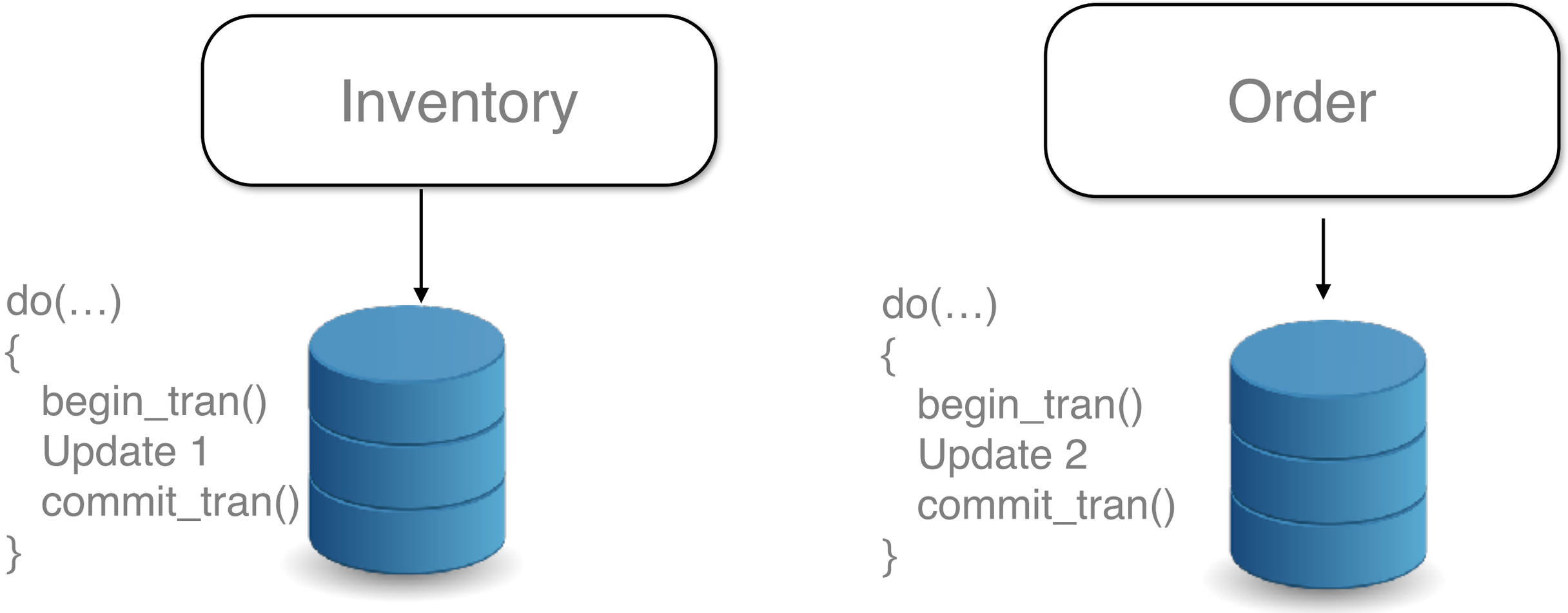


Saga Pattern is a direct solution to implementing distributed transactions in a microservices architecture.

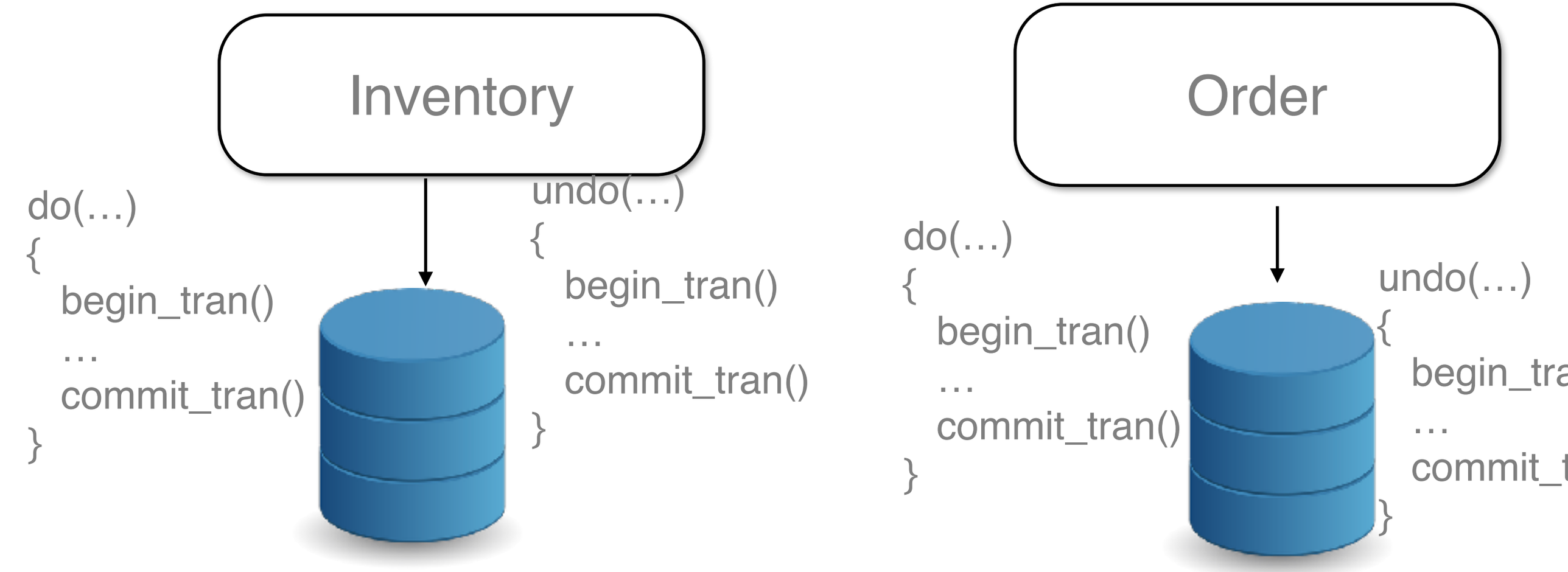
Transaction



SAGA pattern



Compensatable Transaction



Tranid,order,inventory, …
101, y, y, ….

Transaction

2

5

Inventory

Order

do(…)
{
  begin_tran()
  …
  commit_tran()
}

3

undo(…)
{
  begin_tran()
  …
  commit_tran()
}

6

do(…)
{
  begin_tran()
  …
  commit_tran()
}

undo(…)
{
  begin_tra
  …
  commit_t
}

ACID

BASE

Immediate Consistency

Eventual Consistency

UI

CRUD

<<https / wss>>

Load Balancer

WAF

API Gateway

Order API

Create Order

Stream Analytics

Event Sourcing (Domain Events and Commands)

Read

<<grpc / thrift >>

Create Order
No Stocks

Event
Logger

Order
Cmd Service

do() undo()

CQRS

Report
Query Service

do() undo()

Event Store (Big Data)

(Full States)

DWH
(Snap Shot
States)

Rdbms
(Current State)
3rd Nomal Form
joins
no duplicates
write friendly
poor read

Document
(Materialized View)

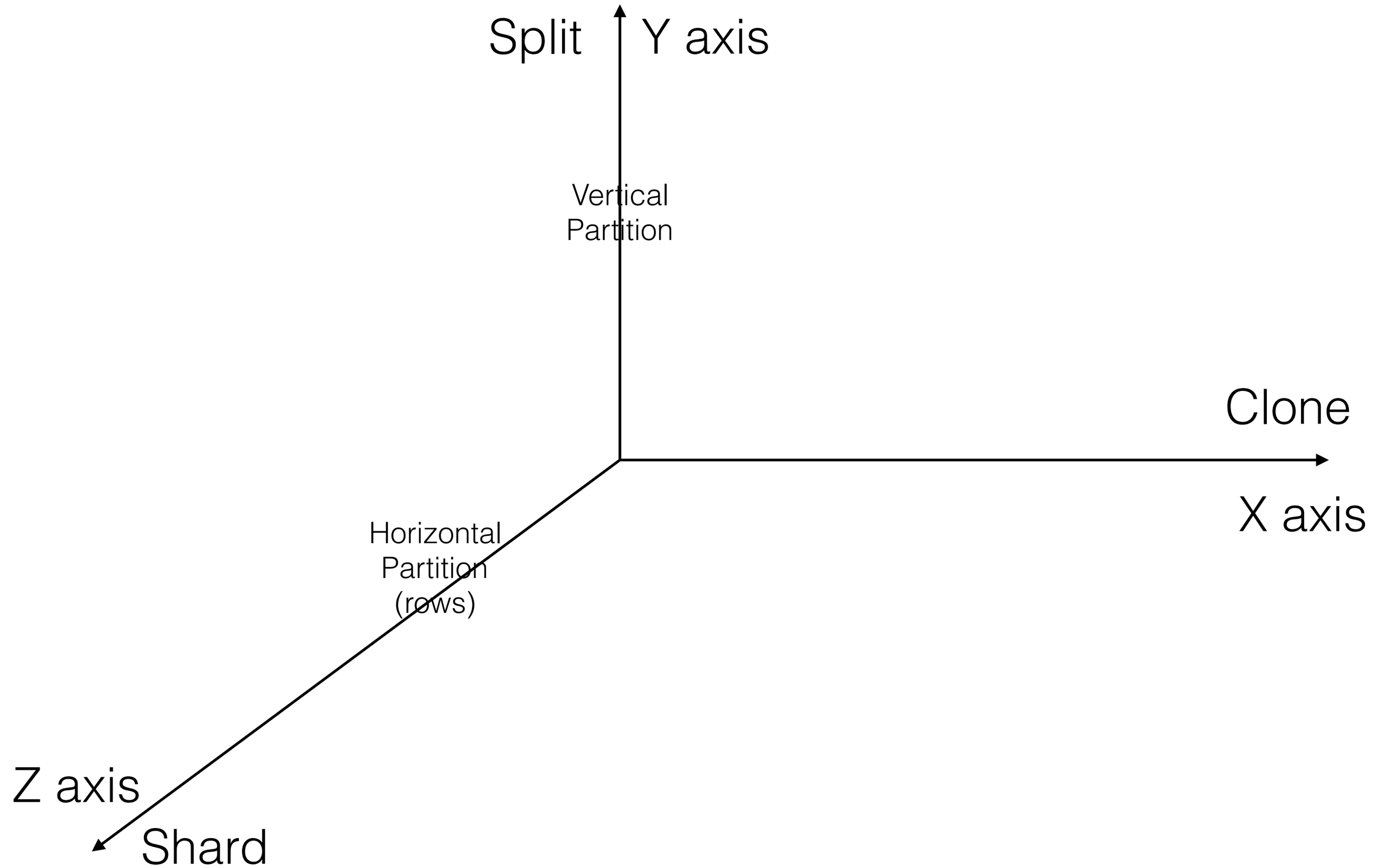
DeNomalized Form
no joins
more duplicates
not write friendly
read friendly

Big Data Analytics

DWH Analytics

Scalability

“Scalability Cube” - 50 rules for high Scalability



“Scalability Cube” - 50 rules for high Scalability

Split

Y axis

Vertical
Partition

Horizontal scaling

Web Server Cluster

Clone

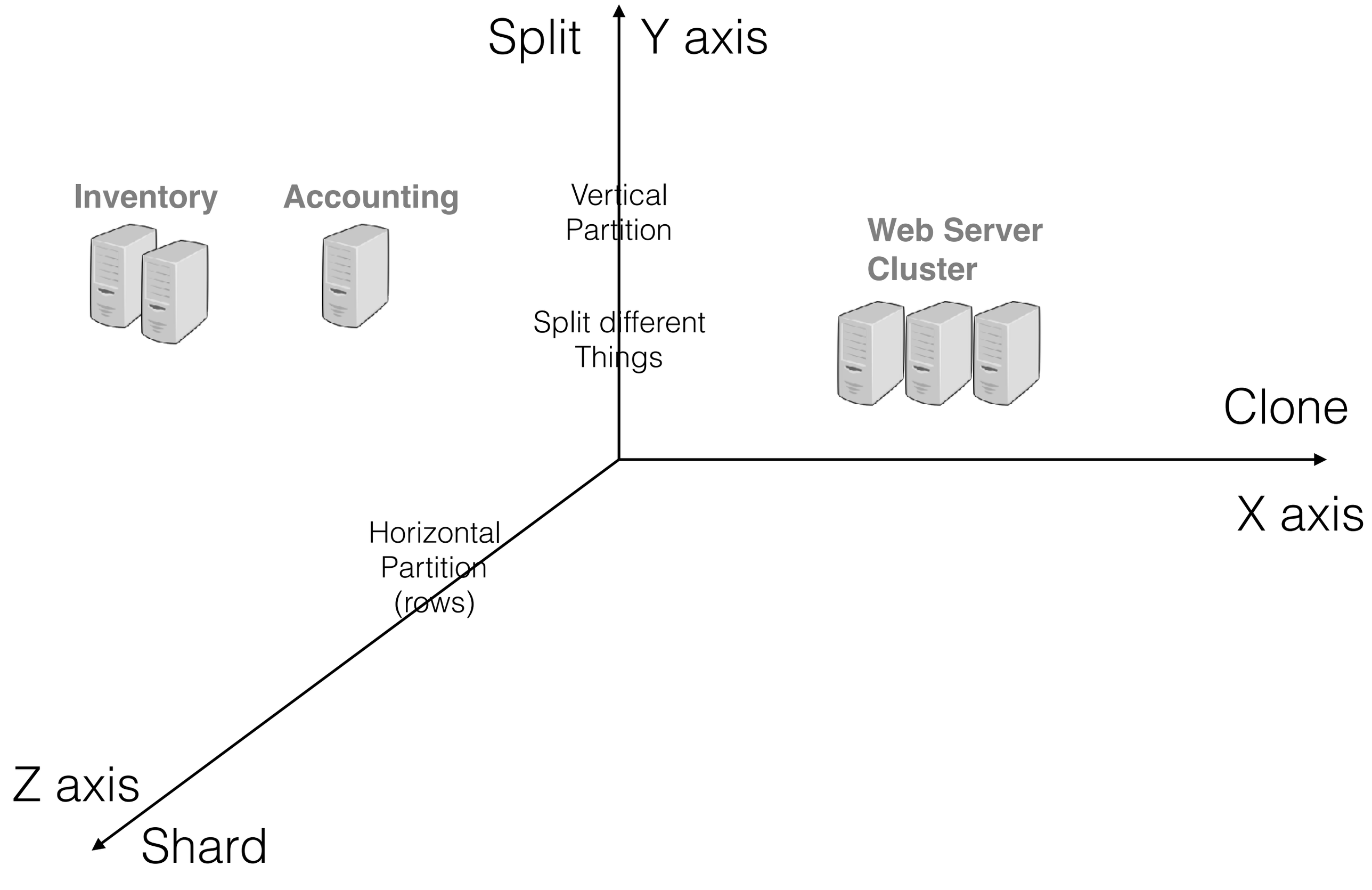
X axis

Horizontal
Partition
(rows)

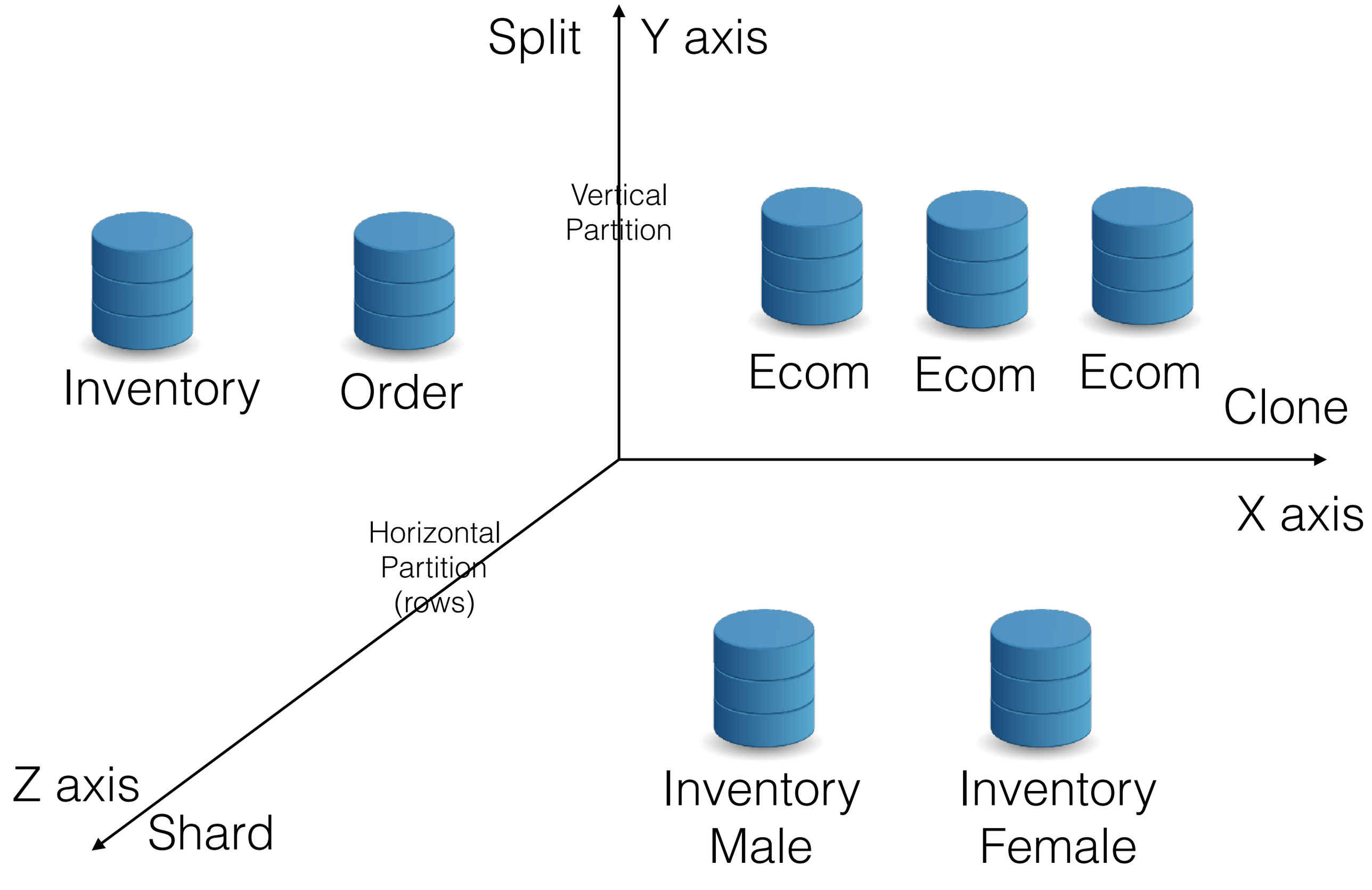
Z axis

Shard

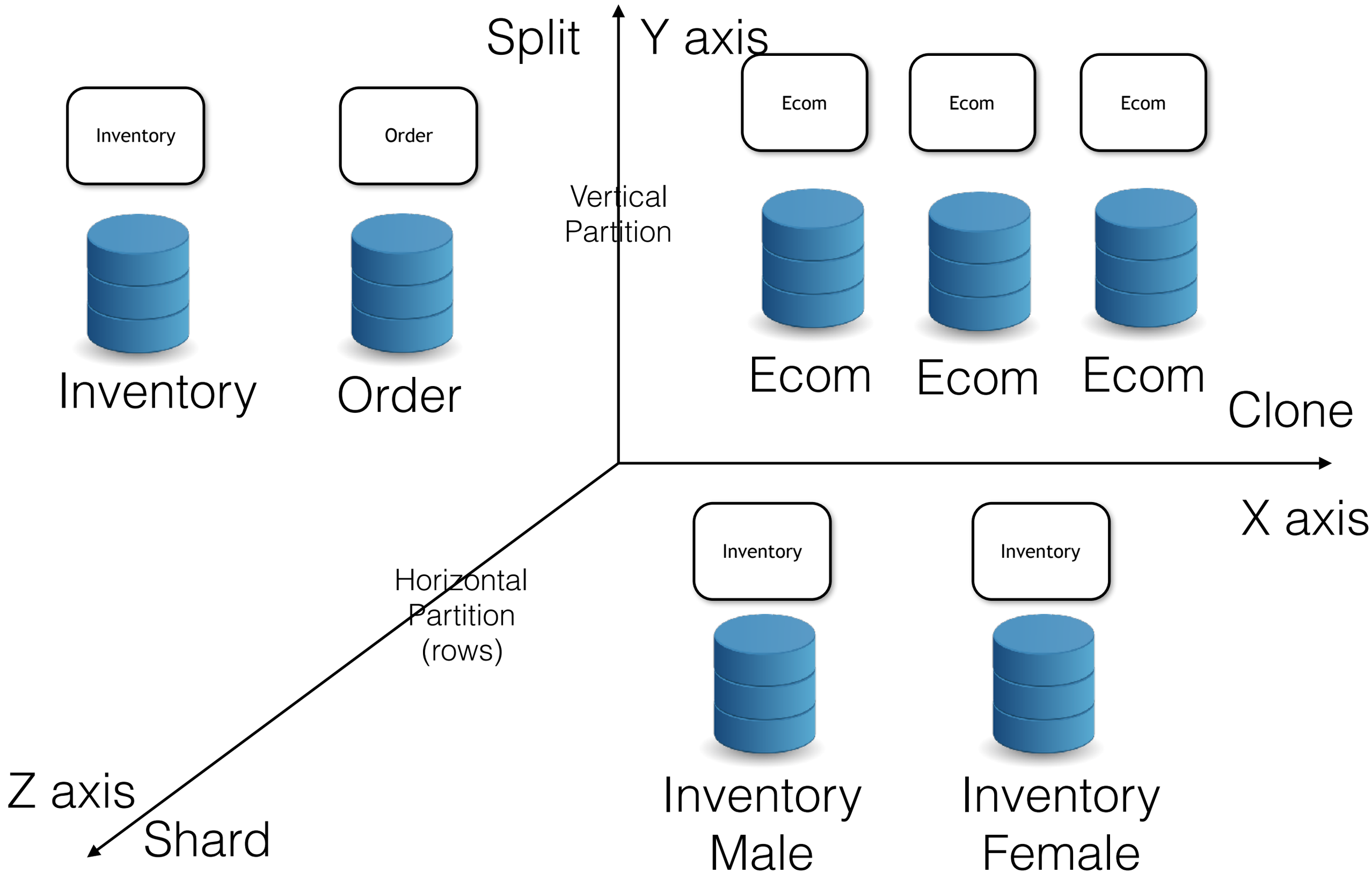
“Scalability Cube” - 50 rules for high Scalability



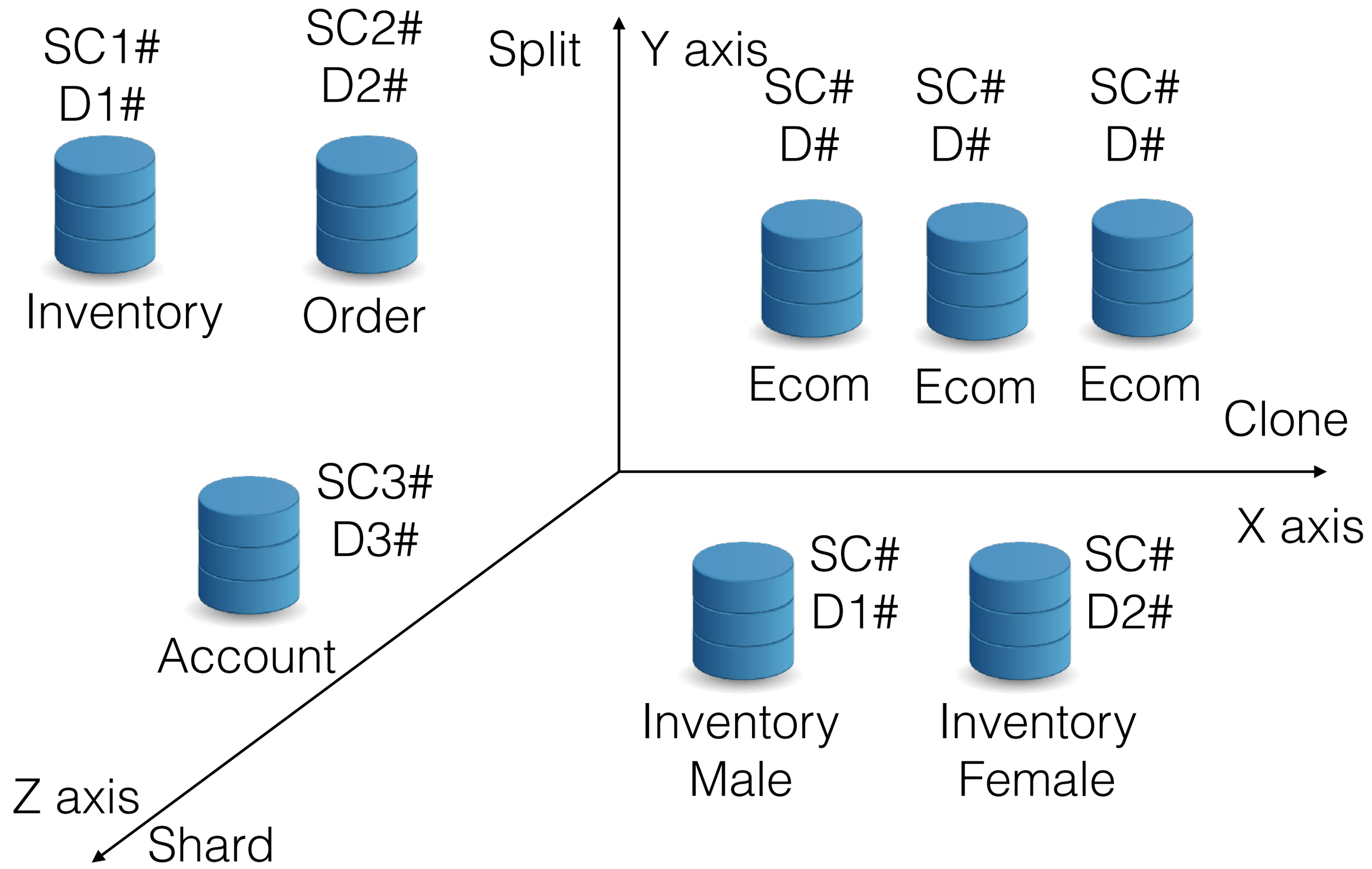
“Scalability Cube” - 50 rules for high Scalability

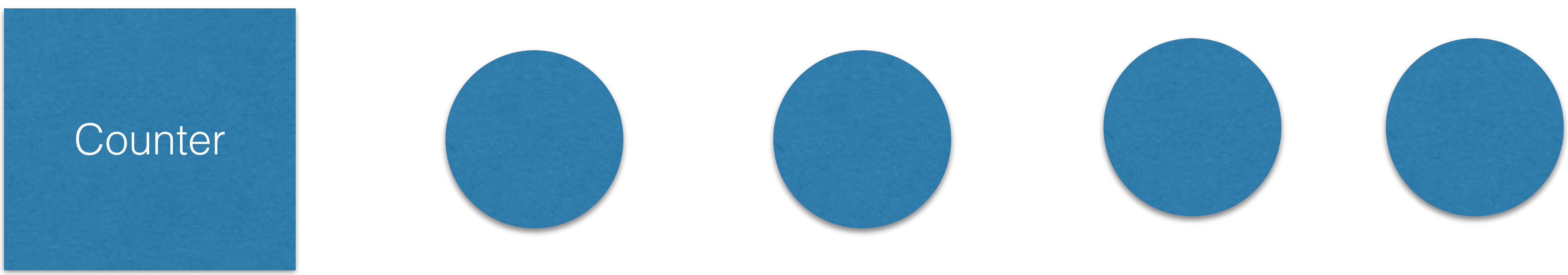


“Scalability Cube” - 50 rules for high Scalability



Scalability Cube - 50 rules for high Scalability




Counter

Counter

C

BP

Counter

E

BP

Counter

?

BP

General

C

E

?

BP

BP

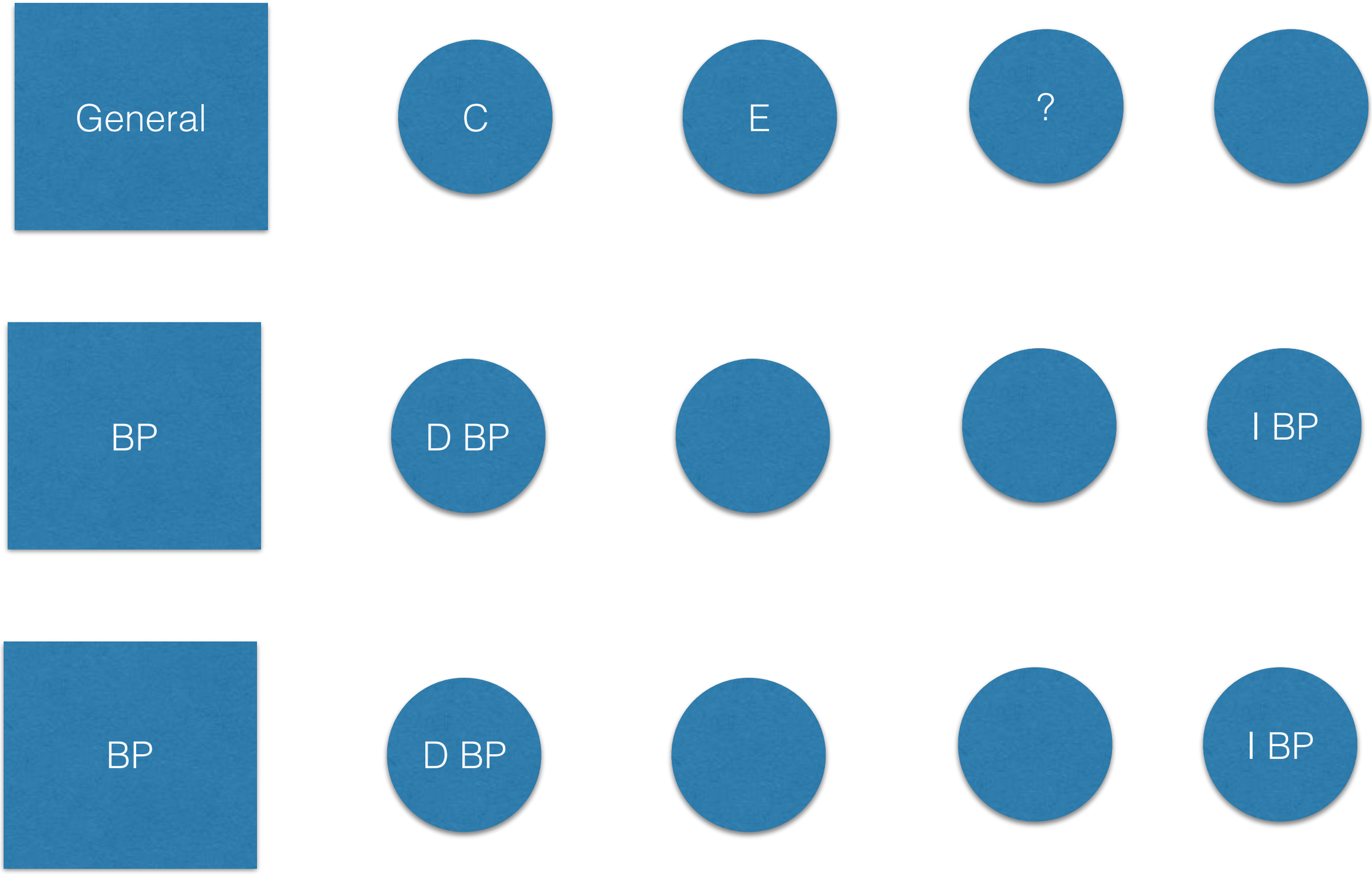
BP

BP

BP

BP

BP


General
C
E
?
BP
D BP
I BP
BP
D BP
I BP

General

C

E

?

D BP

D BP

D BP

D BP

D BP

D BP

D BP

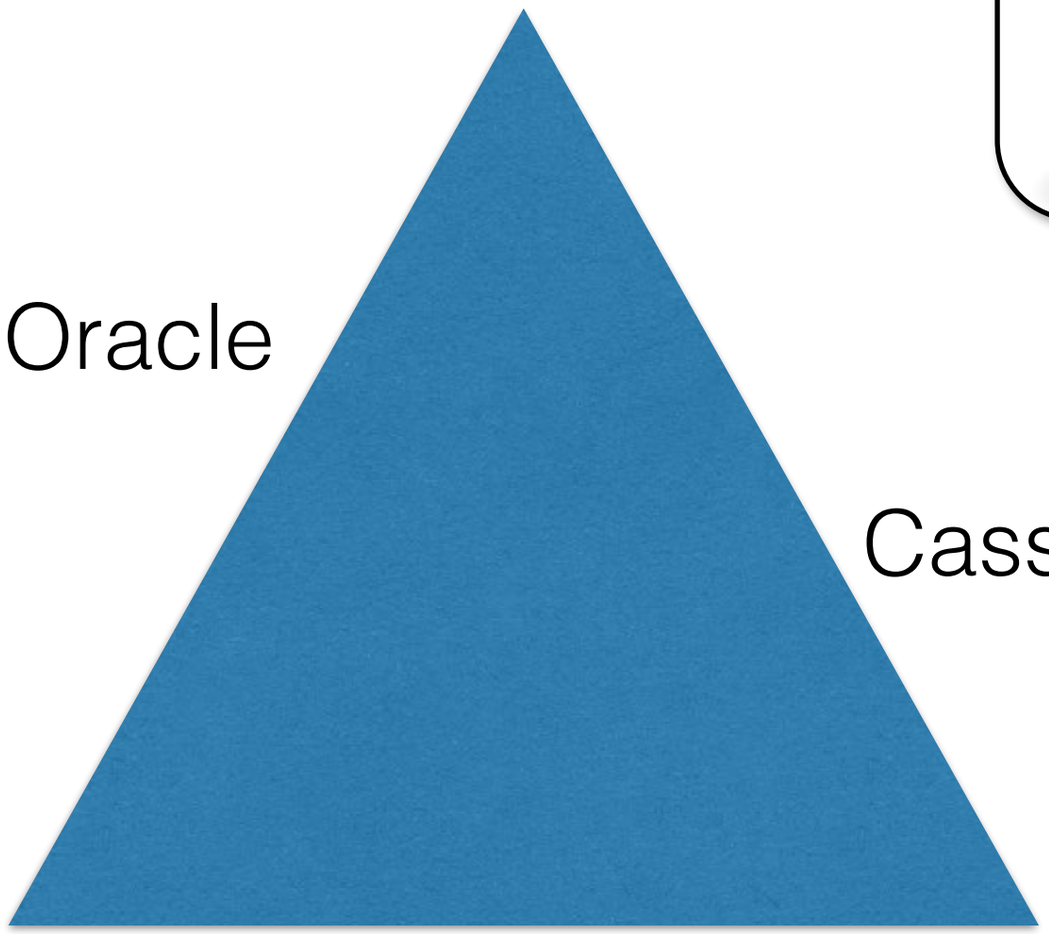
I BP

I BP

I BP

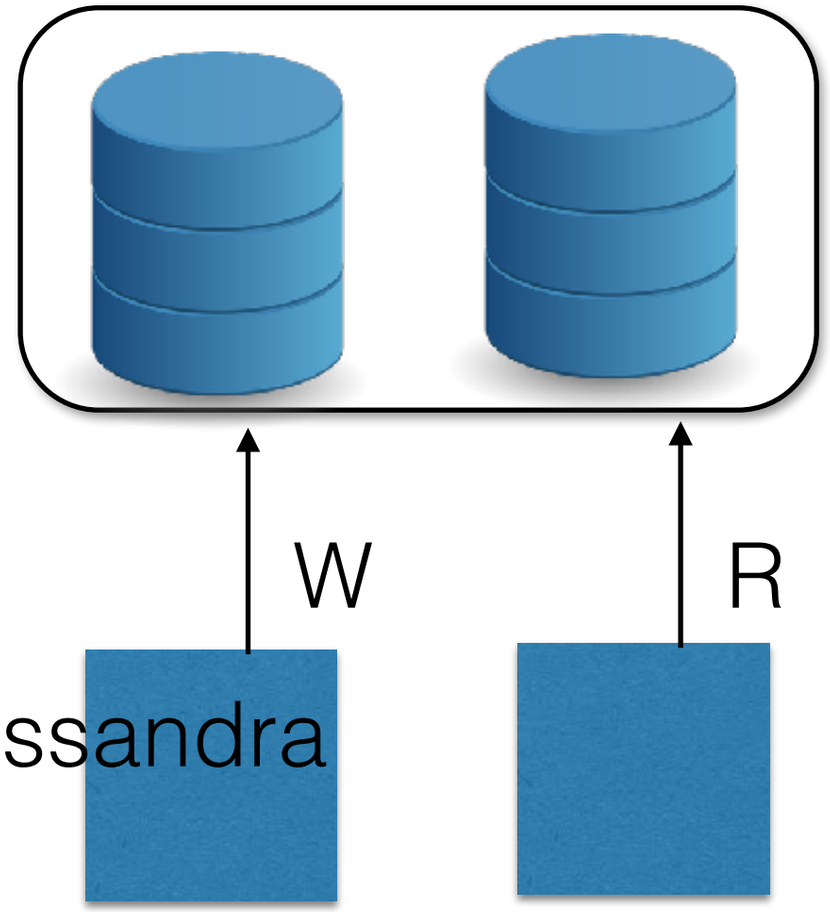
CAP Theorem

A - Availability



mySql, Postgres, Oracle

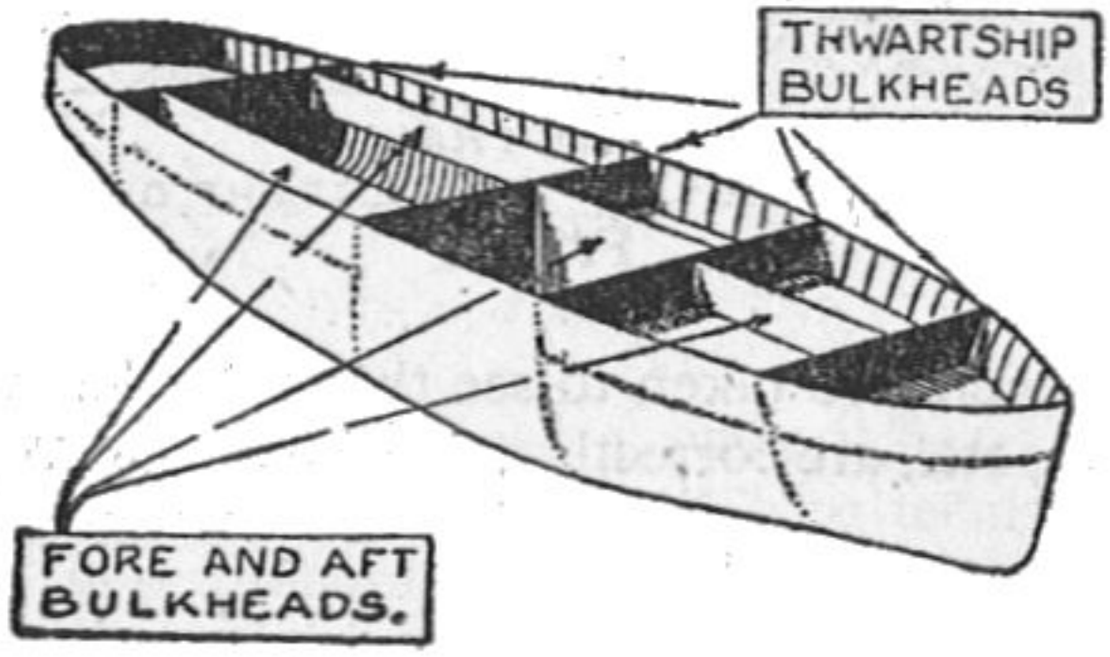
C - Consistency



P - Distributed

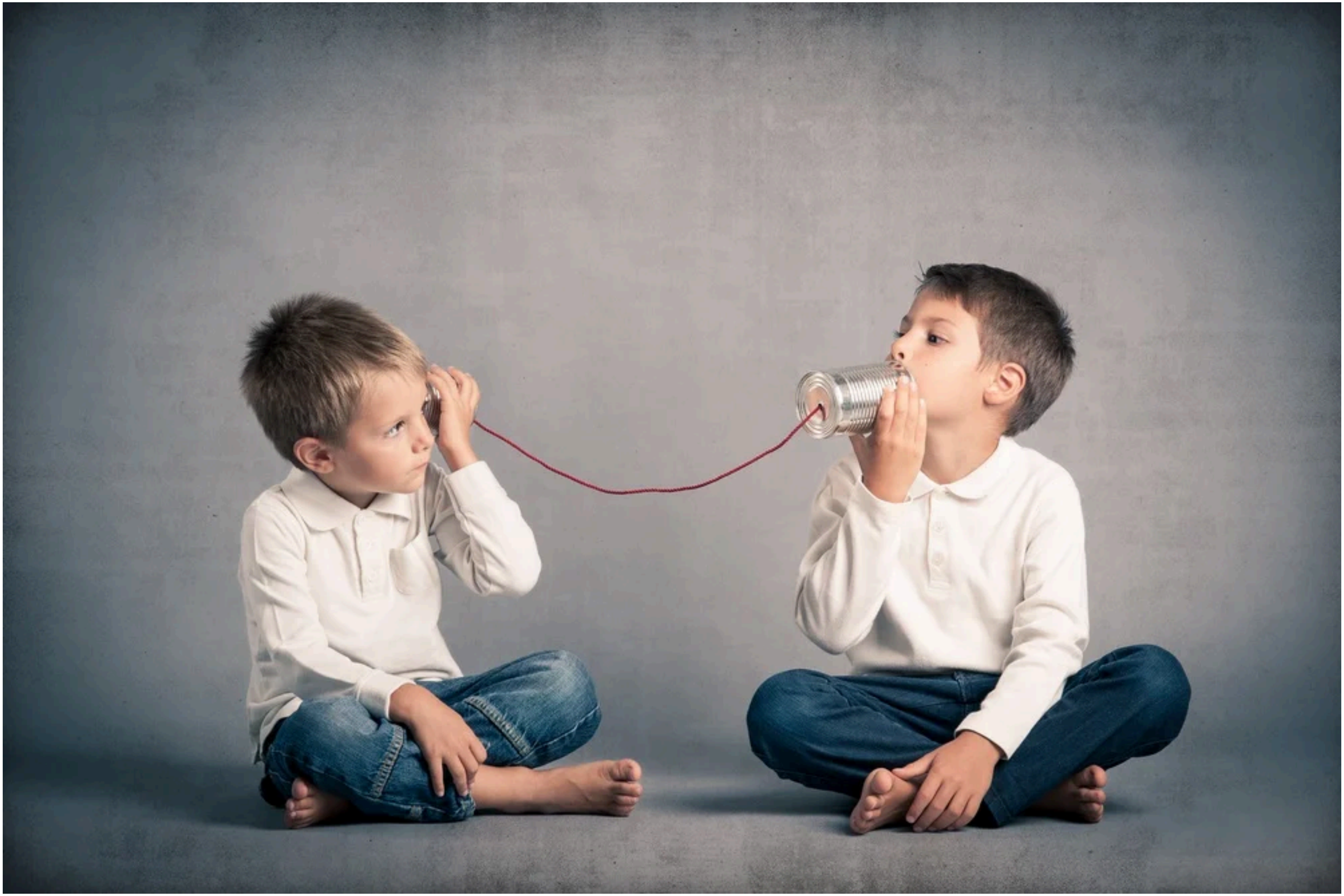
mongodb

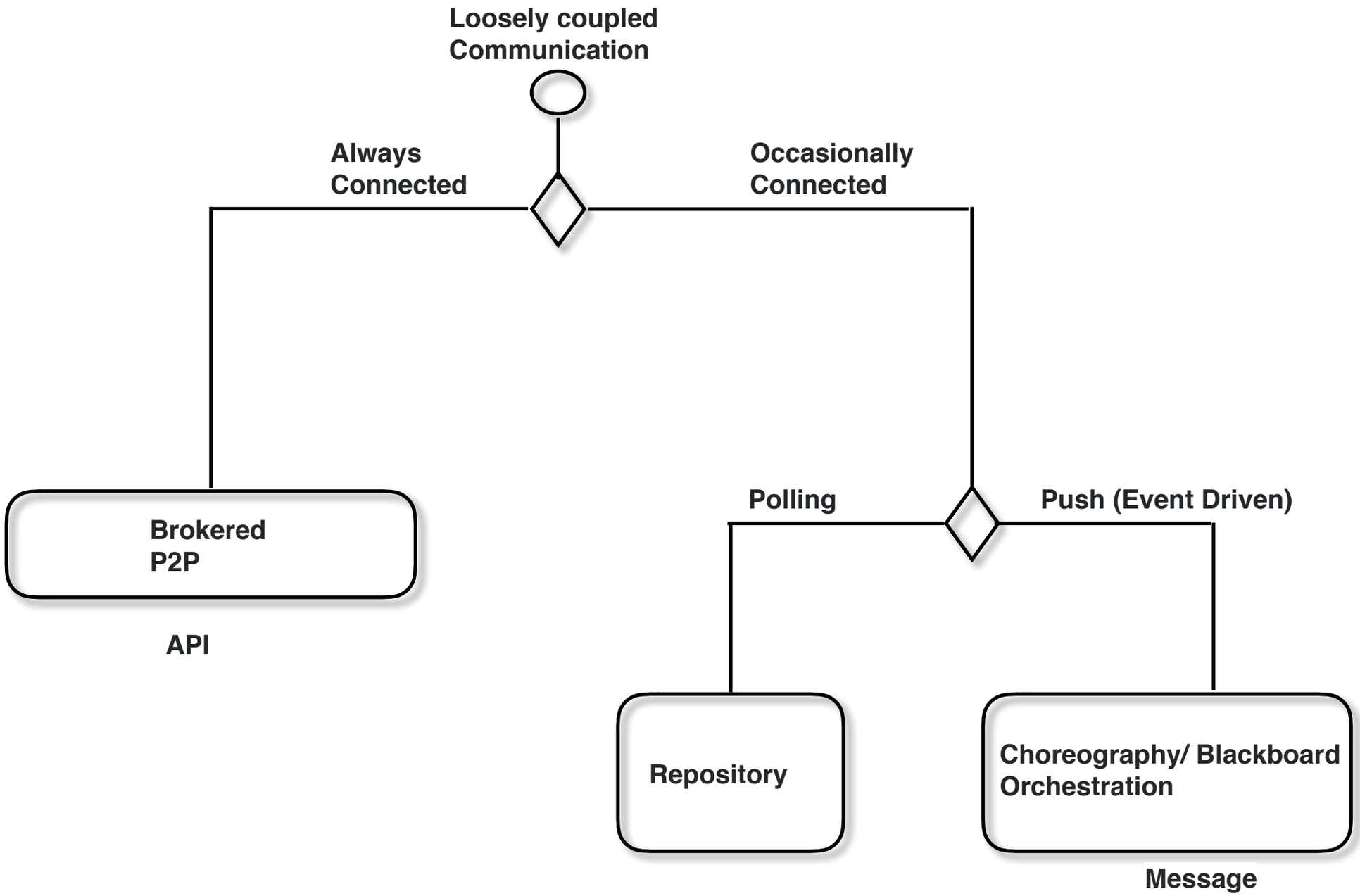
bulkhead

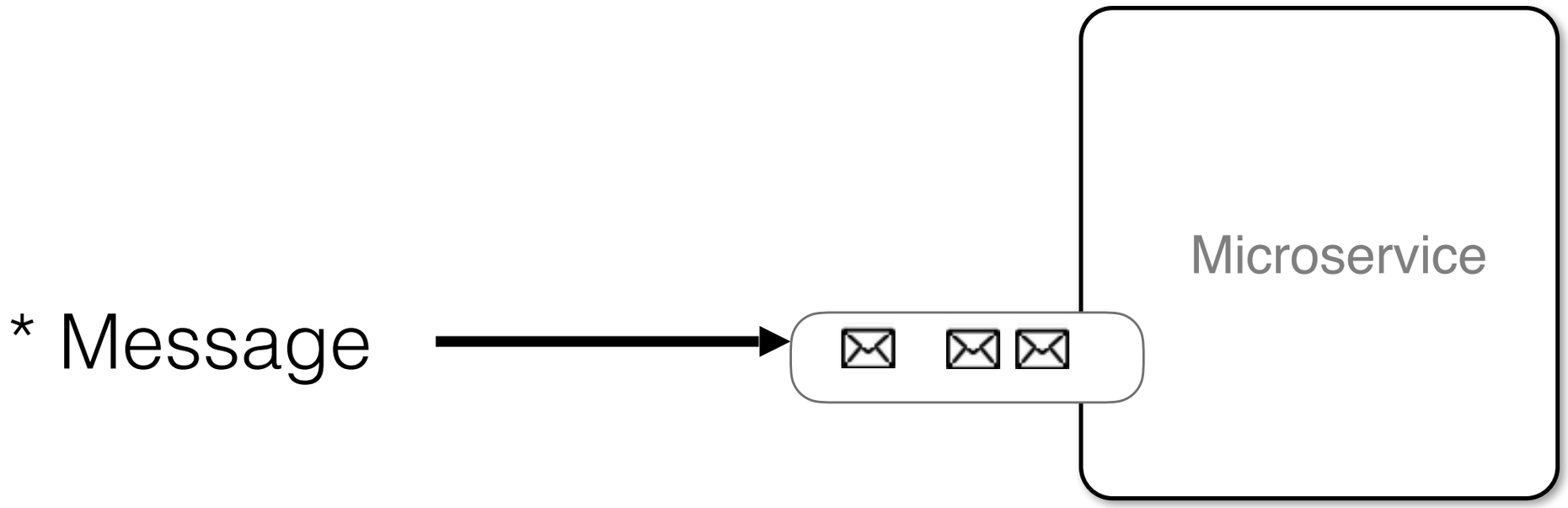


- In general, the goal of the bulkhead pattern is to avoid faults in one part of a system to take the entire system down. The term comes from ships where a ship is divided in separate watertight compartments to avoid a single hull breach to flood the entire ship; it will only flood one bulkhead.cir

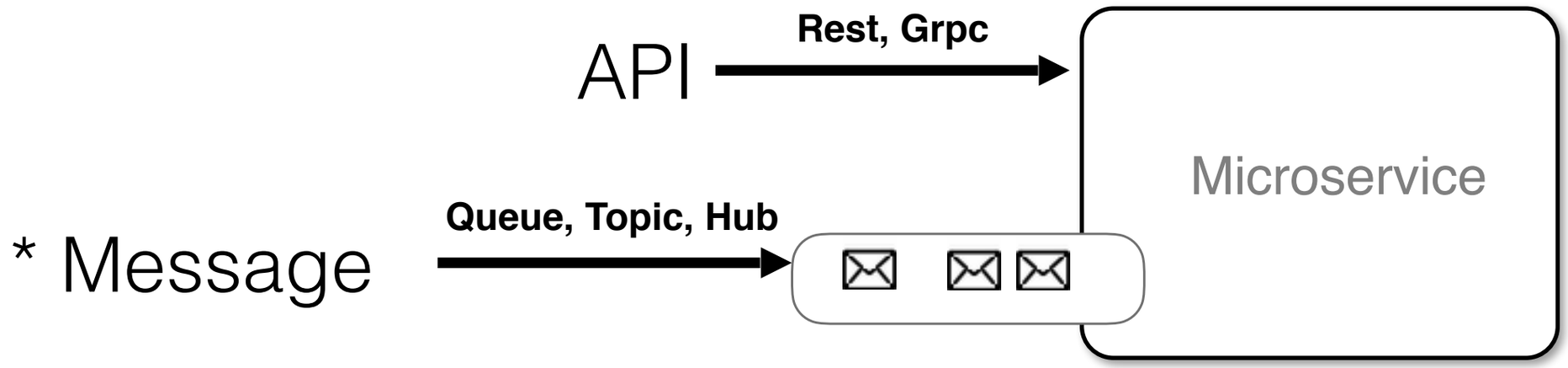
- Partition service instances into different groups, based on consumer load and availability requirements.




Loosely coupled
Communication
Always
Connected
Occasionally
Connected
Brokered
P2P
API
Polling
Push (Event Driven)
Repository
Choreography/ Blackboard
Orchestration
Message




```
fun(){
    while(true){
        Msg m = GetMessage();
        .....
        m.ack();
    }
}
```



Concerns	API	Message	
Ordering	Ordered	Unordered	
Duplicate	No	Yes	Idempotency
Protocol	2 way	One way	
Resilence	No	Yes	
Occousionaly connected	Always connected	Yes	
Scalability	- -	+ +	
Consistency	Immediate (++)	Eventual (- -)	
Load Leveling	- -	+ +	
Low Coupling	- -	+ +	
Distributed Comm Patterns	- -	+ +	

MSMQ

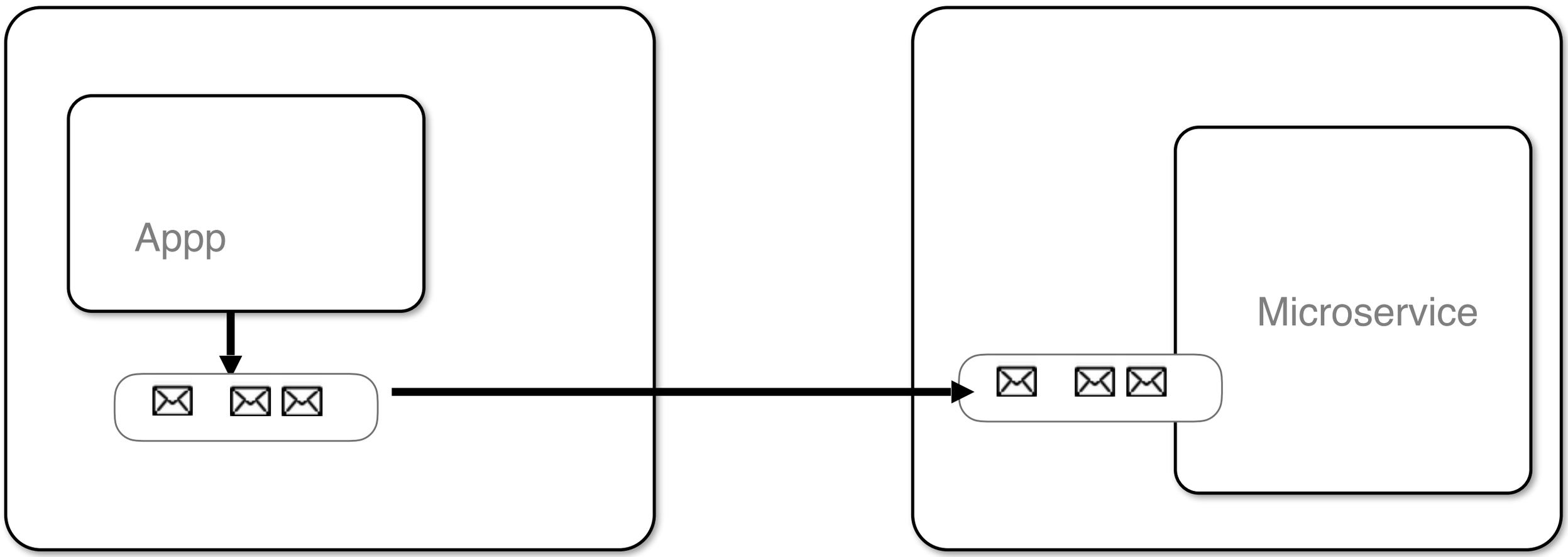
* Message

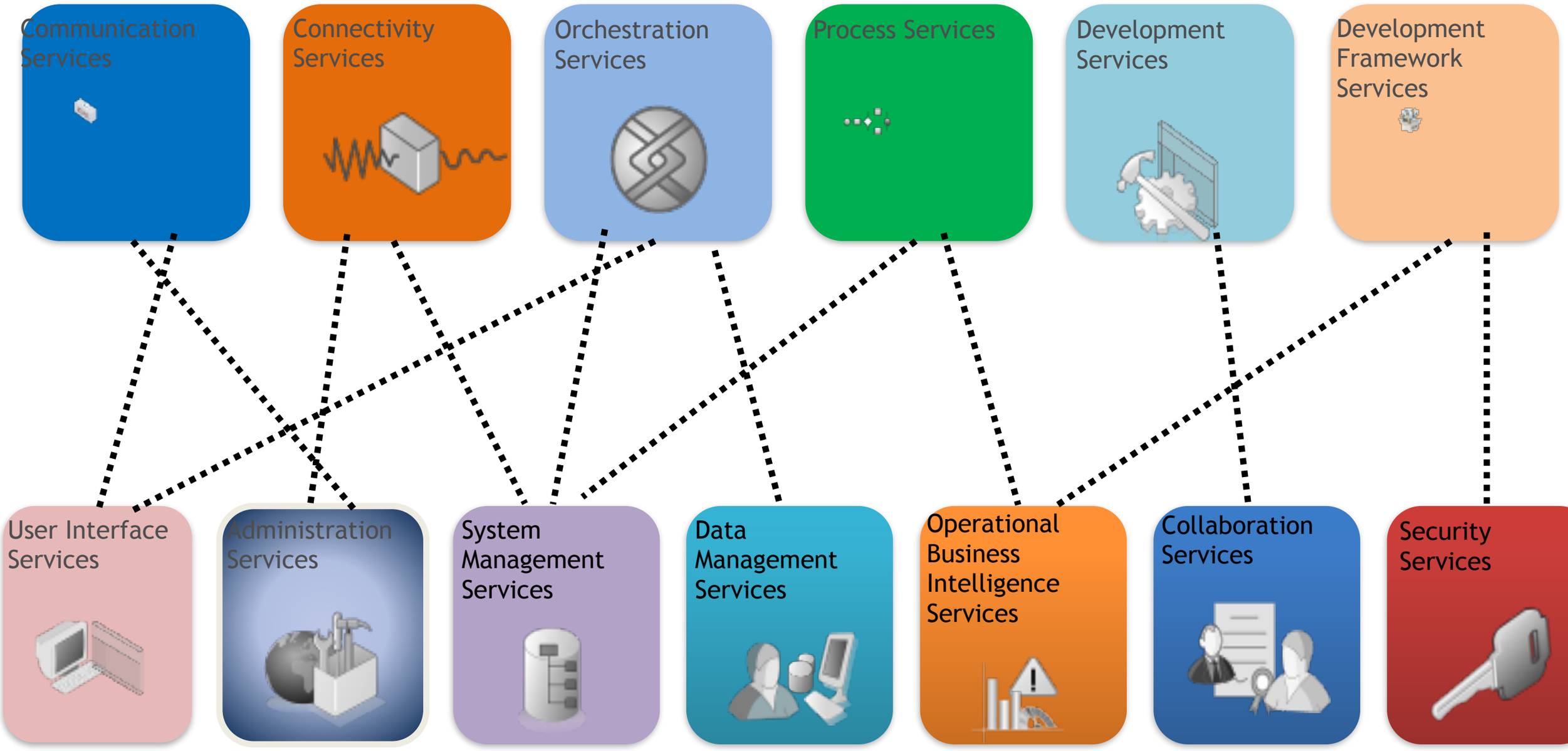
Client

Server

Appp

Microservice




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

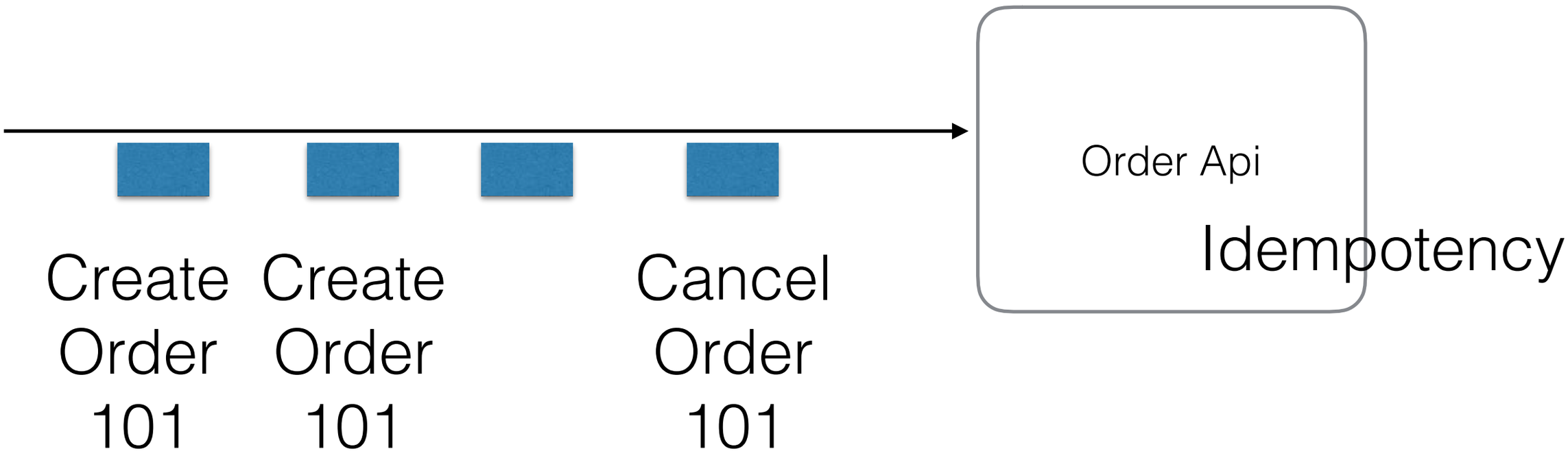
Security Services

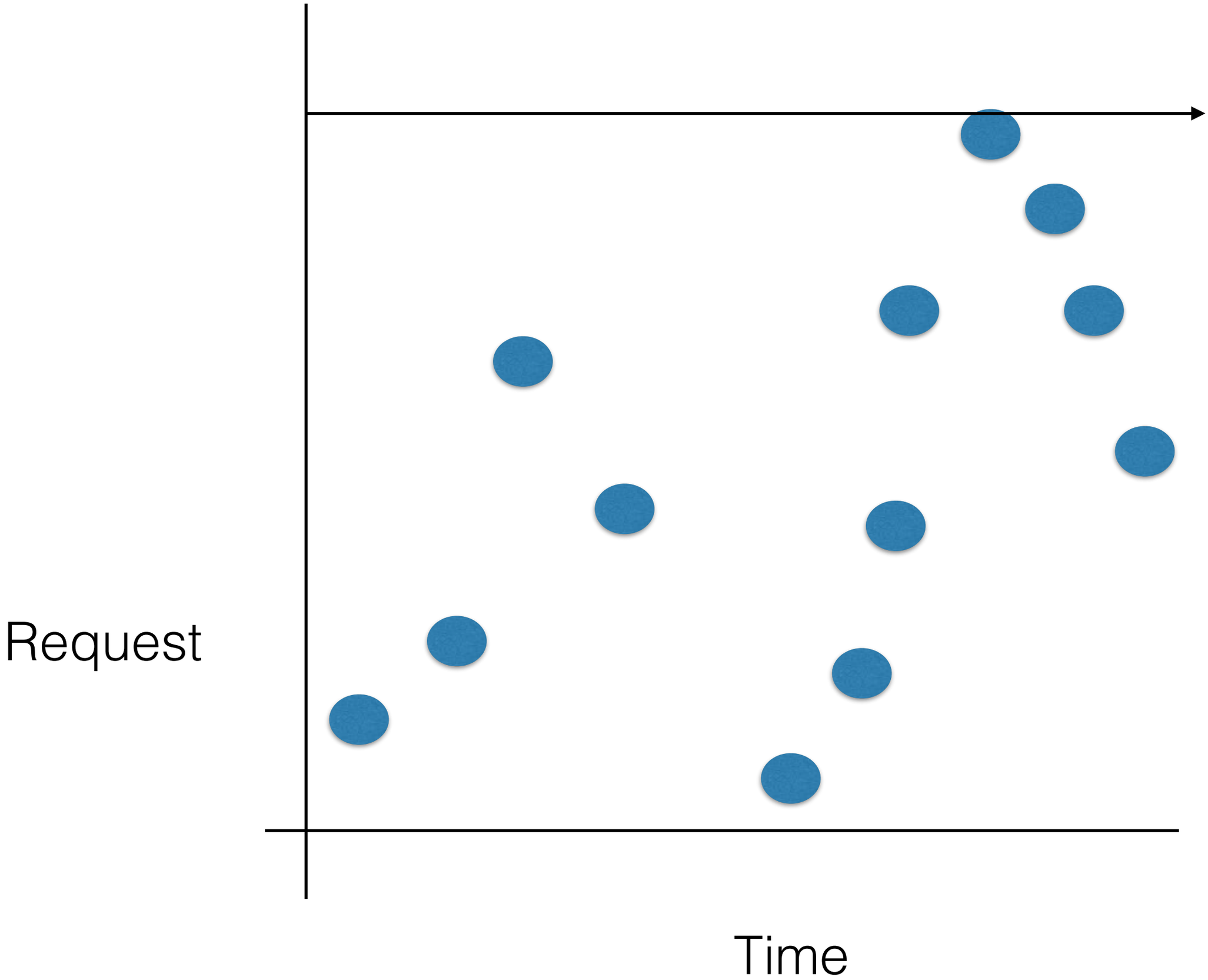

Tasks
Message queue
Service
Requests received
at a variable rate
Messages
processed
at a more
consistent rate

Connected protocol
(REST, WS, GRPC , Thrift)

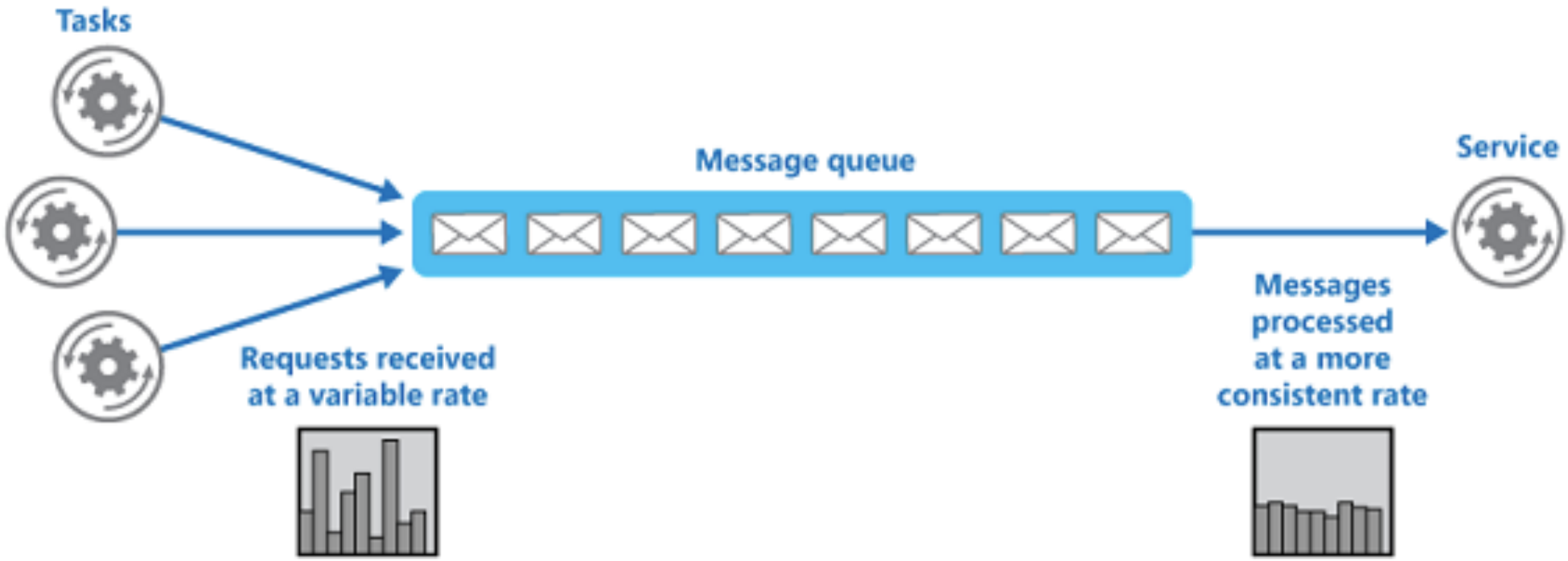
Message protocol
(AMQP, MQTT, …)

	REST	WSS	GRPC
Browser Support	Y	Y	N
Format	TEXT	TEXT	BINARY, HTTP2
Serialization	JSON	JSON	Proto Buf
Request - Response	Y	Y	Y
Performance	- -	+	+ +
Use case	Browser client	Streaming	Service to Service
	North South		East West


Order Api
Idempotency
Create Order 101
Create Order 101
Cancel Order 101

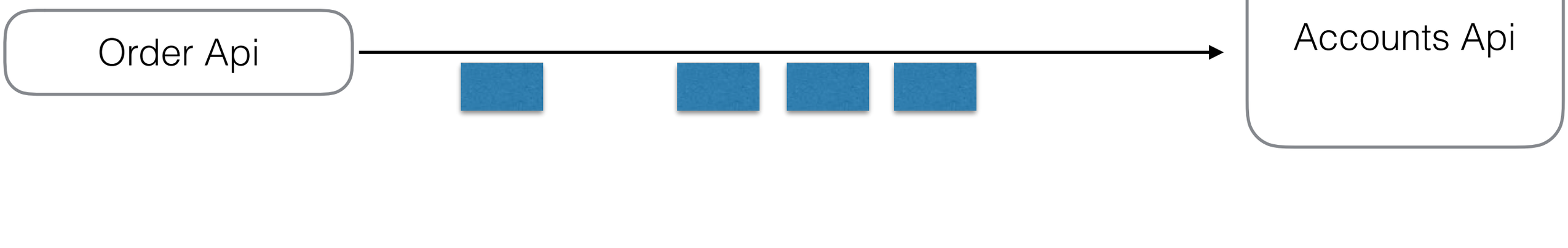

Request
Time

Queue-based Load Levelling



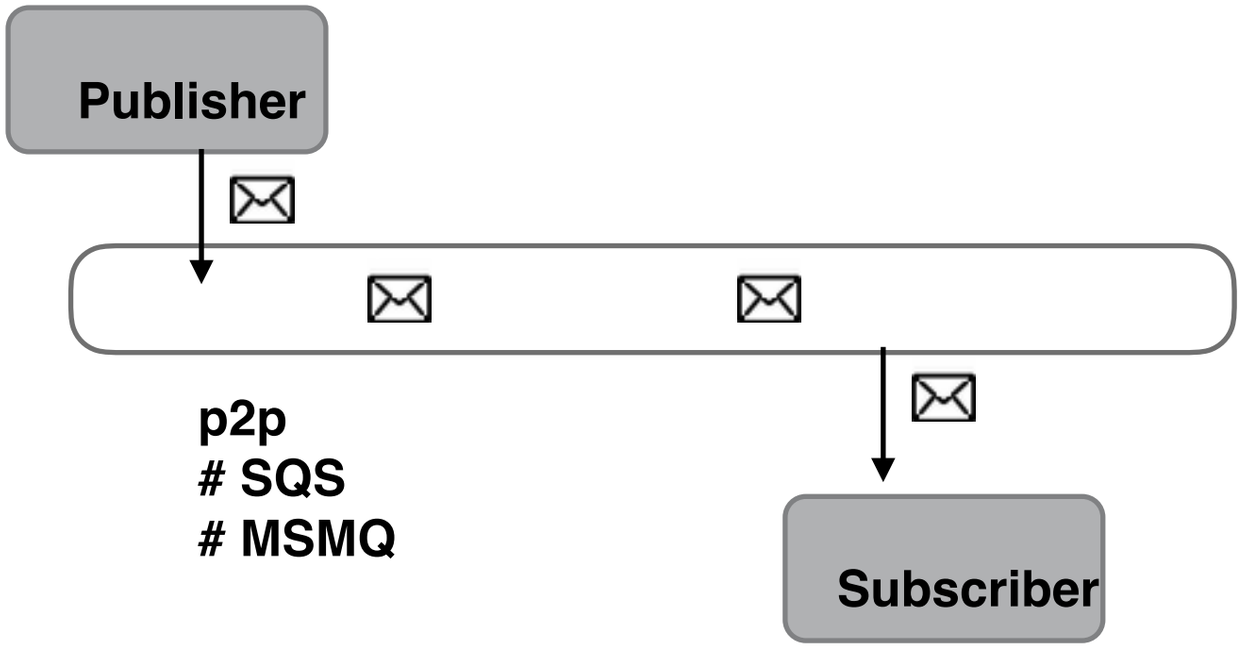
source:msdn

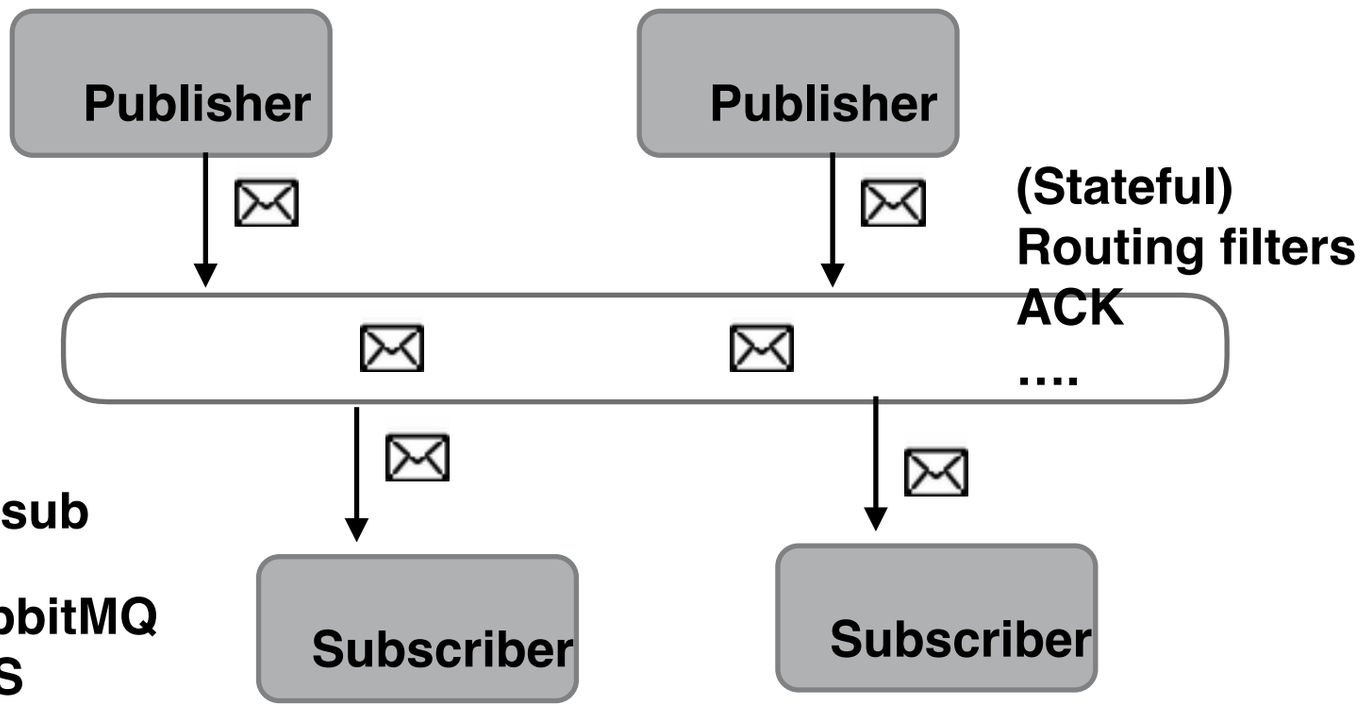
Message db (eventual)



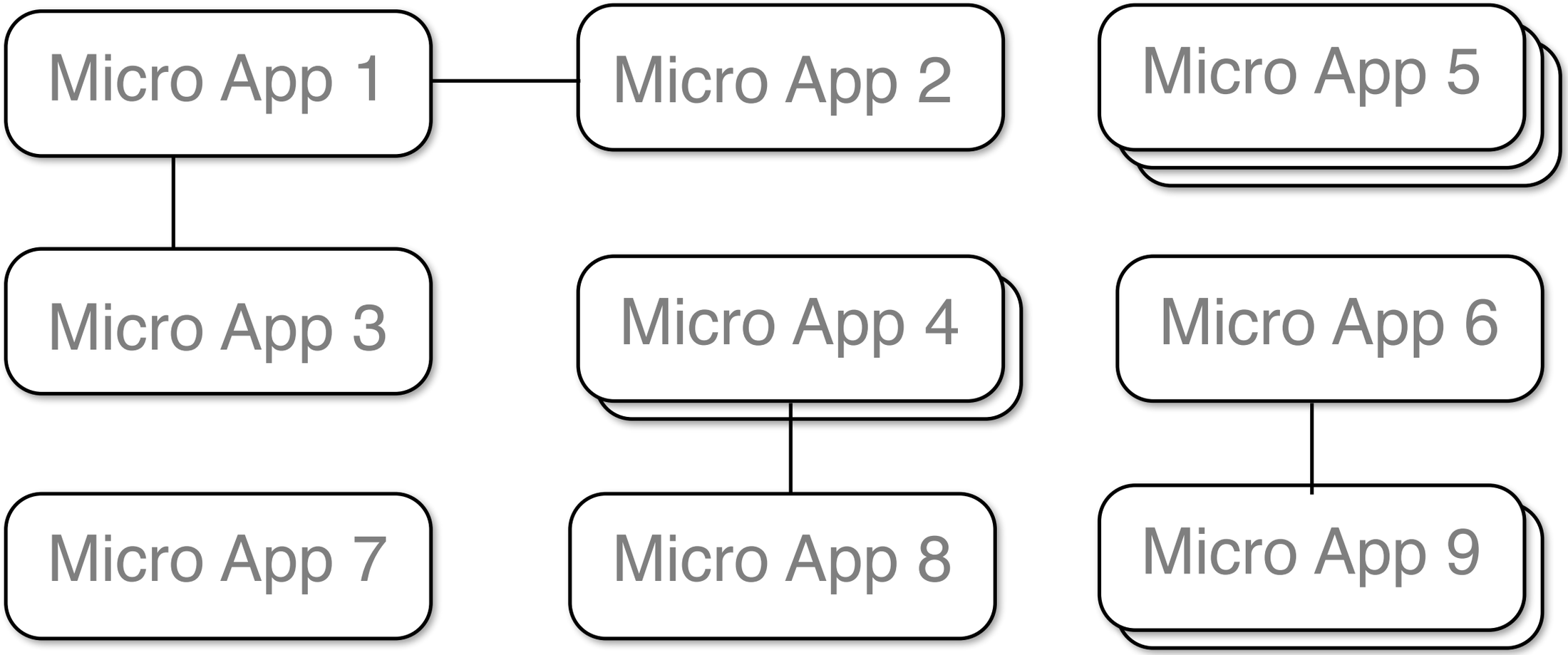

```
msg = getmessage()
...
...
msg.ack()
```

1. Message Queue
2. Message Bus (Topic)
3. Message hub





Isolated Env.


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

My app 1

My app 2

Middlewear (tomcat/ Apache/ …)

Os

Cpu
RAM
Disk
I/o

Machine

VM1

VM2

My app 1

My app 2

Middlewear(tomcat/apache/…)

Middlewear

Unix Guest Os

Win Guest Os

Virtualisation layer

Host Os

Cpu
RAM
Disk
I/o

Cpu
RAM
Disk
I/o

Cpu
RAM
Disk
I/o

Machine

Container 1

Container 2

Cpu
RAM
Disk
I/o

My app 1

Middlewear

My app 2

Middlewear

Cpu
RAM
Disk
I/o

Containerization layer (Docker)

Cpu
RAM
Disk
I/o

Host Os

Machine

VM1

VM2

Container 1

8080

Container 2

My app 1

My app 2

Middlewear

Middlewear

Cpu
RAM
Disk
I/o

8080

8080

Cpu
RAM
Disk
I/o

Containerization layer (Docker)

Unix Guest Os

Win Guest Os

Cpu
RAM
Disk
I/o

Cpu
RAM
Disk
I/o

Virtualisation layer

Host Os (azure)

Machine

VM1

VM2

Container 1

8080

Contai

Todo app

Middlewear

8080

8080

Cpu
RAM
Disk
I/o

Cpu
RAM
Disk
I/o

Containerization layer (Docker)

Unix Guest Os

Win Guest Os

Cpu
RAM
Disk
I/o

Cpu
RAM
Disk
I/o

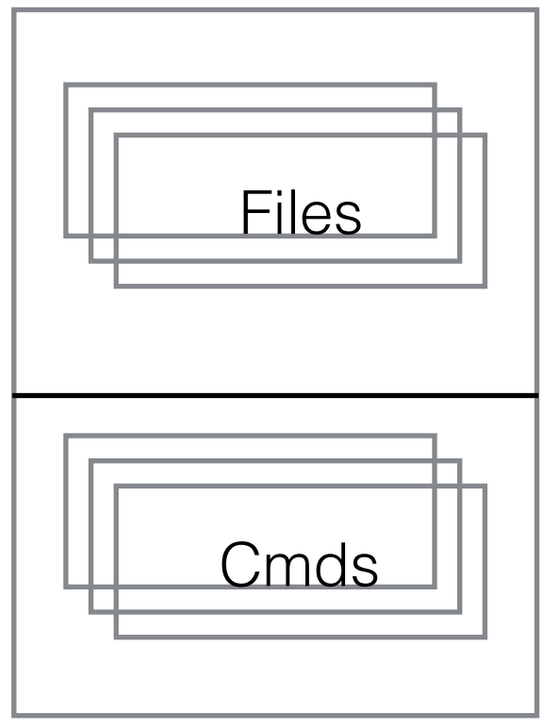
Virtualisation layer

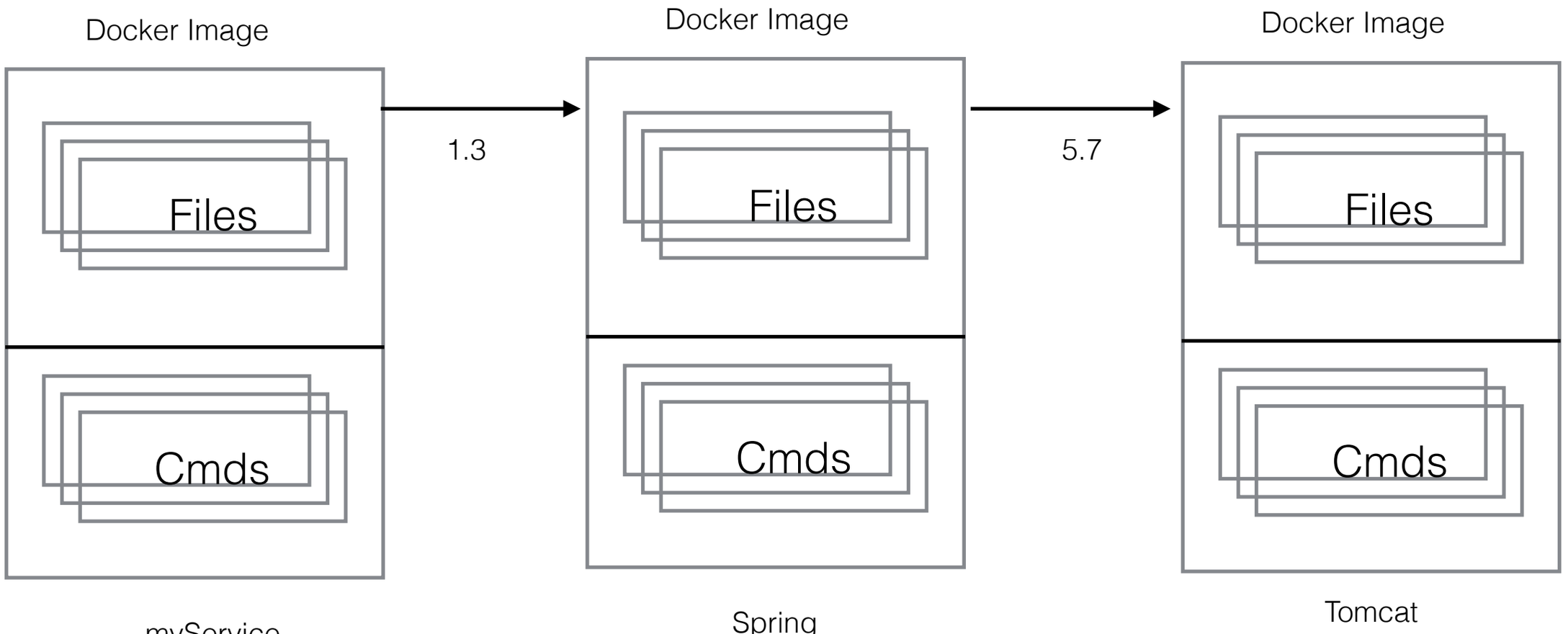
Host Os (azure)

Cpu
RAM
Disk
I/o

Machine

Reproducable Env.


Files
Cmds


Docker Image
Files
Cmds
myService
1.3
Docker Image
Files
Cmds
Spring
5.7
Docker Image
Files
Cmds
Tomcat

Image

Docker file (cmds)

Config Files

Binary Files

Push

hub.docker.com (Repository)

Image

Image

cmds Image

Build

Pull

Config File

Source File

Source File

VM

Container

8080 8080

Nodejs

CPU
RAM
DISK
I/O

copy
Run **cmds** in docker file

Managing Container Life cycle

VM1

VM2

Container 1

Container 2

Container 3

My app 1

My app 2

My app 2

Middlewear

Middlewear

Middlewear

Cpu RAM Disk I/o

Cpu RAM Disk I/o

Containerization layer (Docker)

Unix Guest Os

Cpu RAM Disk I/o

Cpu RAM Disk I/o

Virtualisation layer

Host Os (azure)

Machine

VM1

VM2

Container 1

Container 2

My app 1

Middlewear

Cpu
RAM
Disk
I/o

Kubernetes (slave)

Containerization layer (Docker)

Unix Guest Os

Container 1

Container 2

My app 1

Middlewear

Cpu
RAM
Disk
I/o

Kubernetes (Slave)

Containerization layer (Docker)

Unix Guest Os

Kubernetes (Master)

Run

Cluster

Machine

VM

Pod

Container

Process

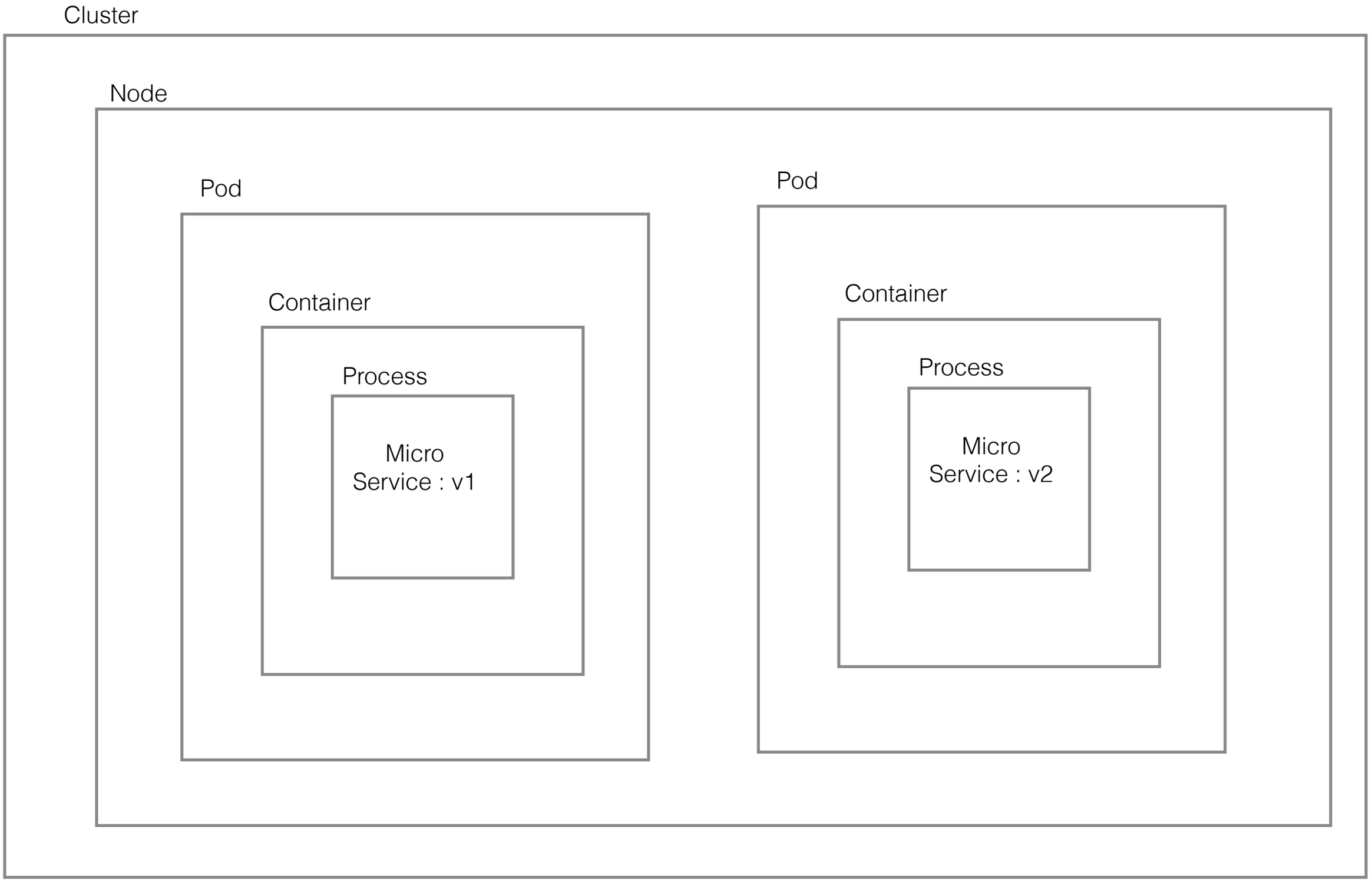
Micro Service

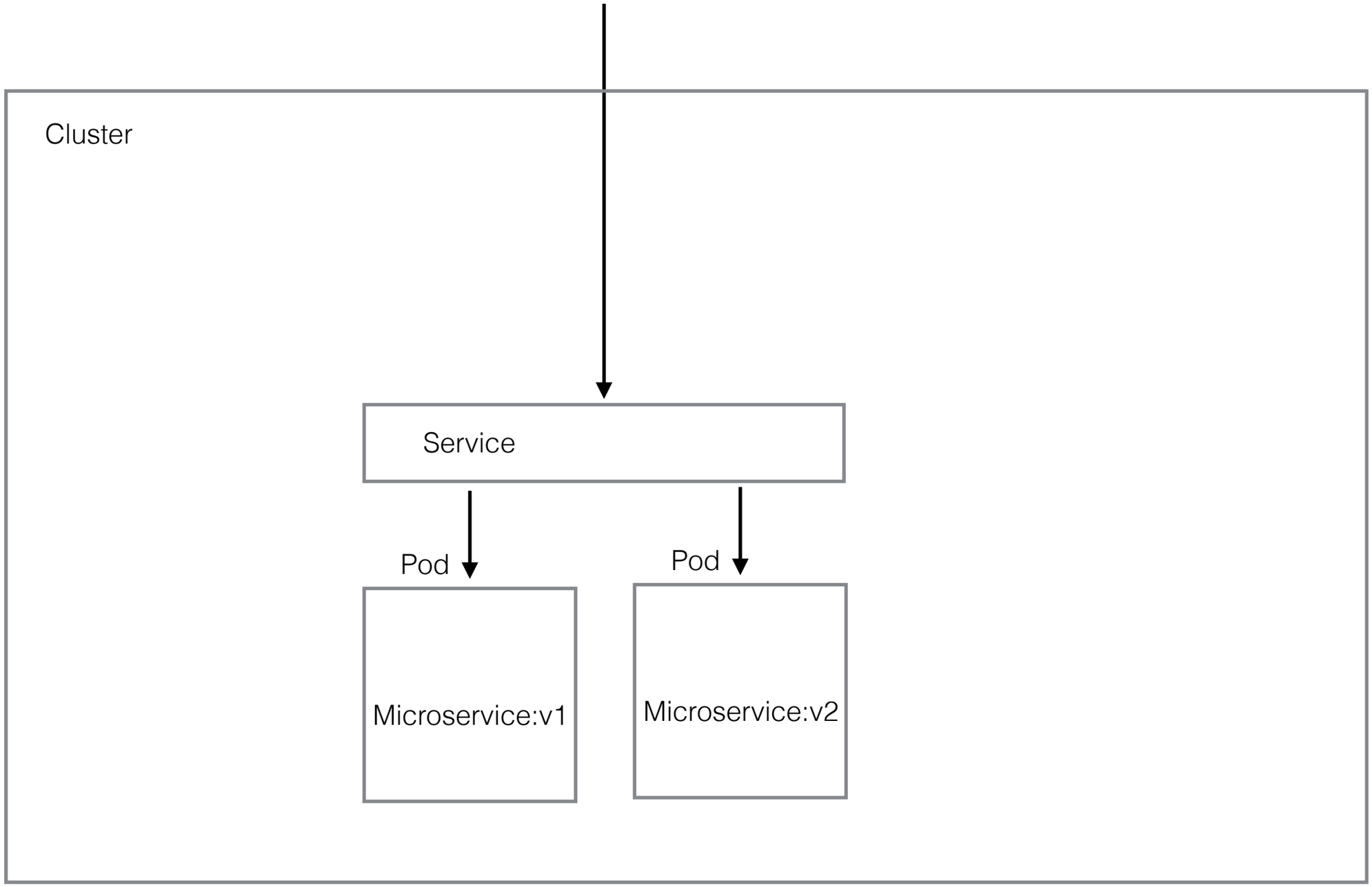
Docker Image

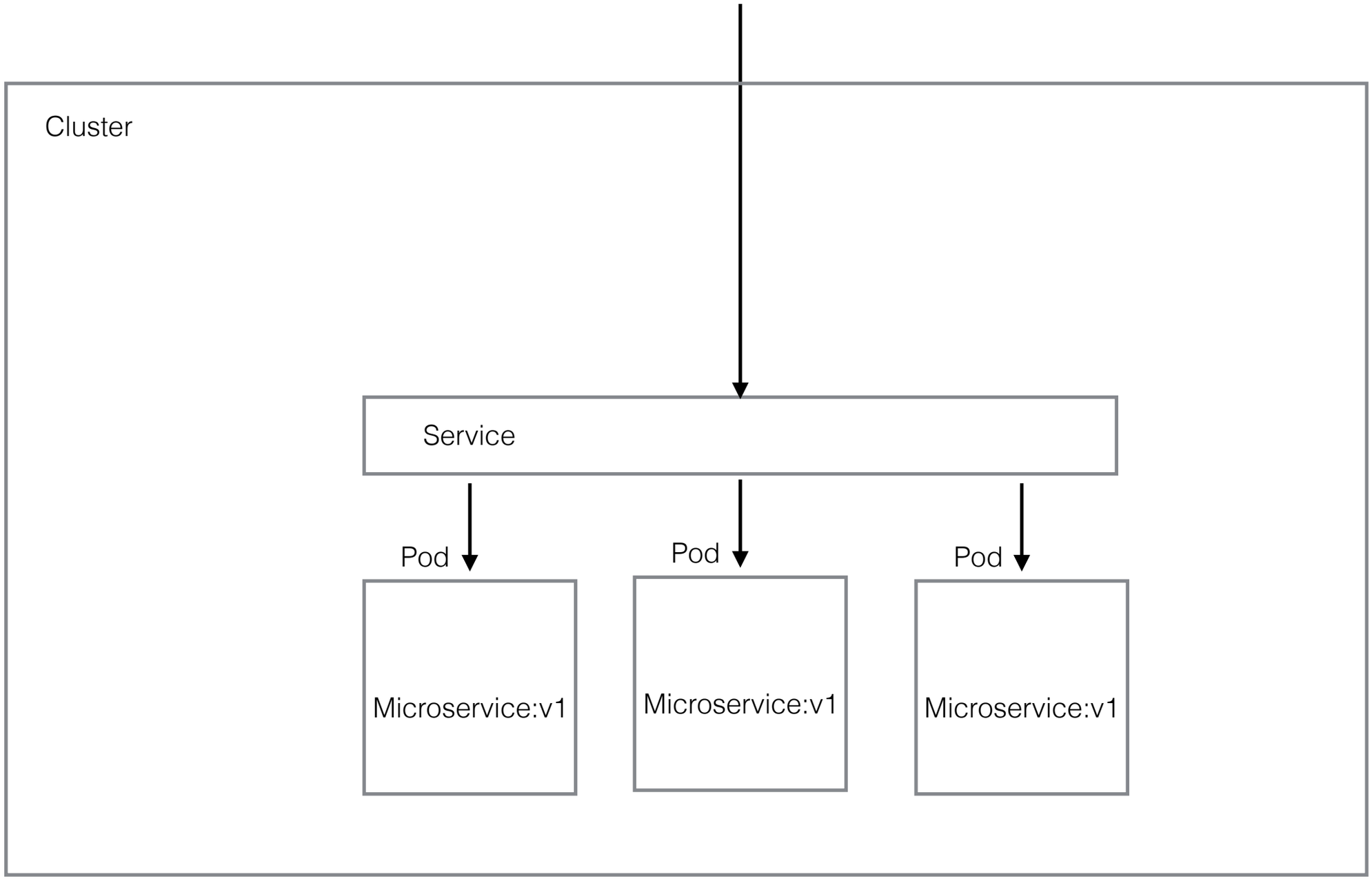
Micro Service

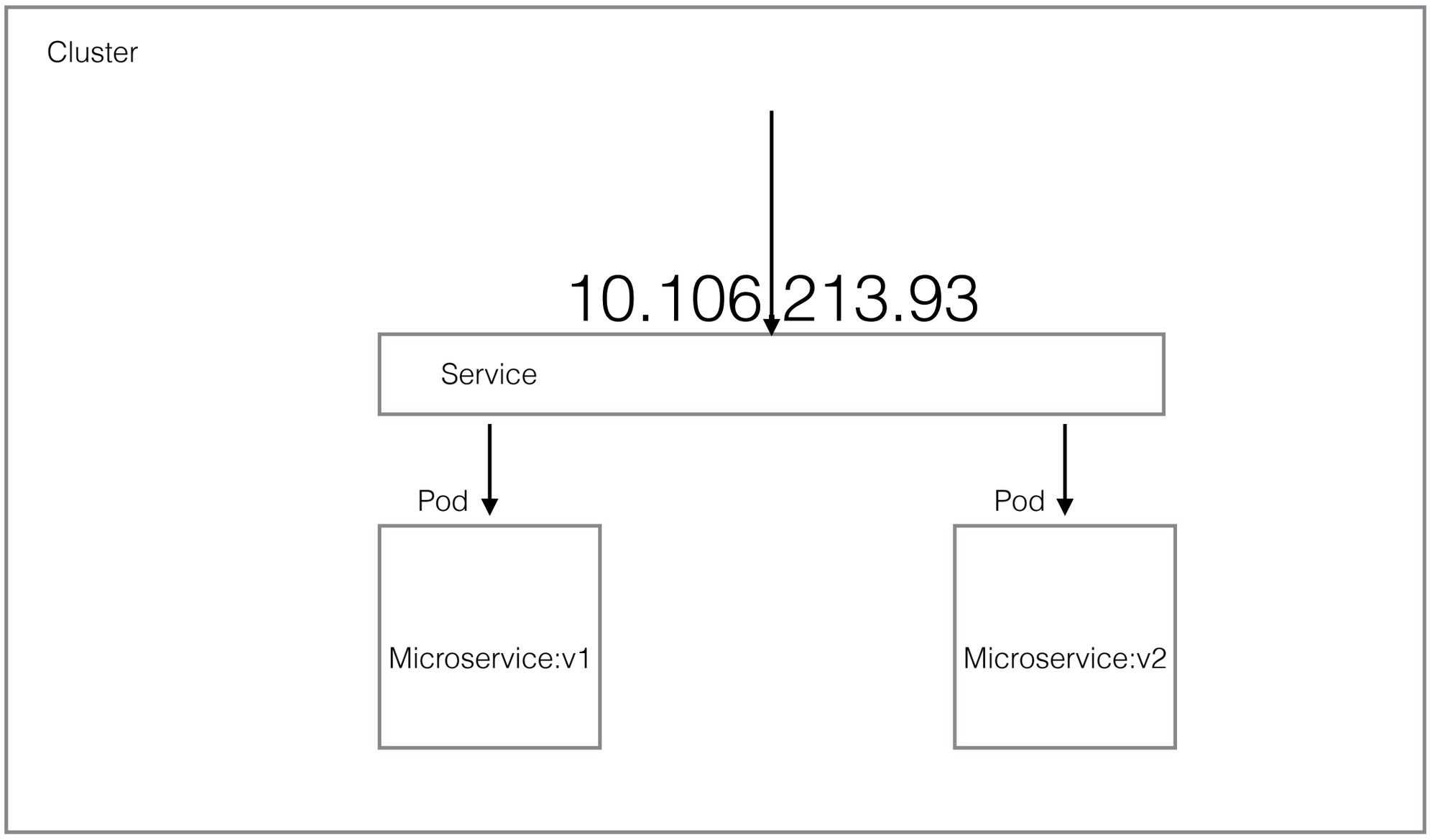
cmds

Docker /k8s


Cluster
Node
Pod
Container
Process
Micro
Service : v1
Pod
Container
Process
Micro
Service : v2


Cluster
Service
Pod
Pod
Microservice:v1
Microservice:v2


Cluster
Service
Pod
Pod
Pod
Microservice:v1
Microservice:v1
Microservice:v1


Cluster
10.106.213.93
Service
Pod
Pod
Microservice:v1
Microservice:v2

Cluster

10.106.213.94

Service B

Pod

Microservice B

Pod

Microservice B

10.106.213.93

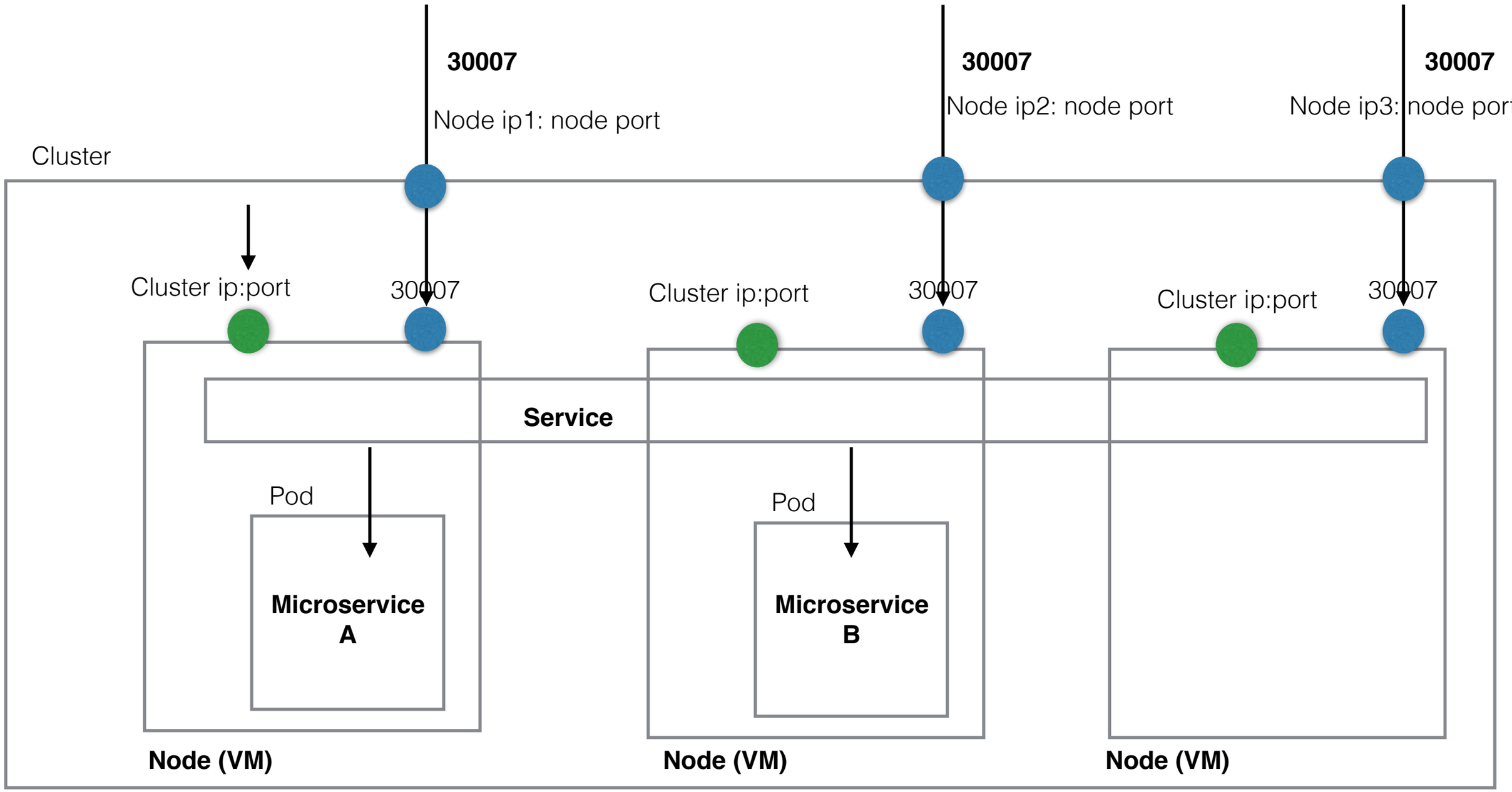
Service A

Pod

Microservice A

Pod

Microservice A


30007
Node ip1: node port
30007
Node ip2: node port
30007
Node ip3: node port
Cluster
Cluster ip:port
30007
Cluster ip:port
30007
Cluster ip:port
30007
Service
Pod
Microservice A
Pod
Microservice B
Node (VM)
Node (VM)
Node (VM)

UI

Network Load Balancer

Node ip1: node port

Node ip2: node port

Node ip3: node port

Cluster

30007

Cluster ip:port

30007

30007

Node

Node

Node

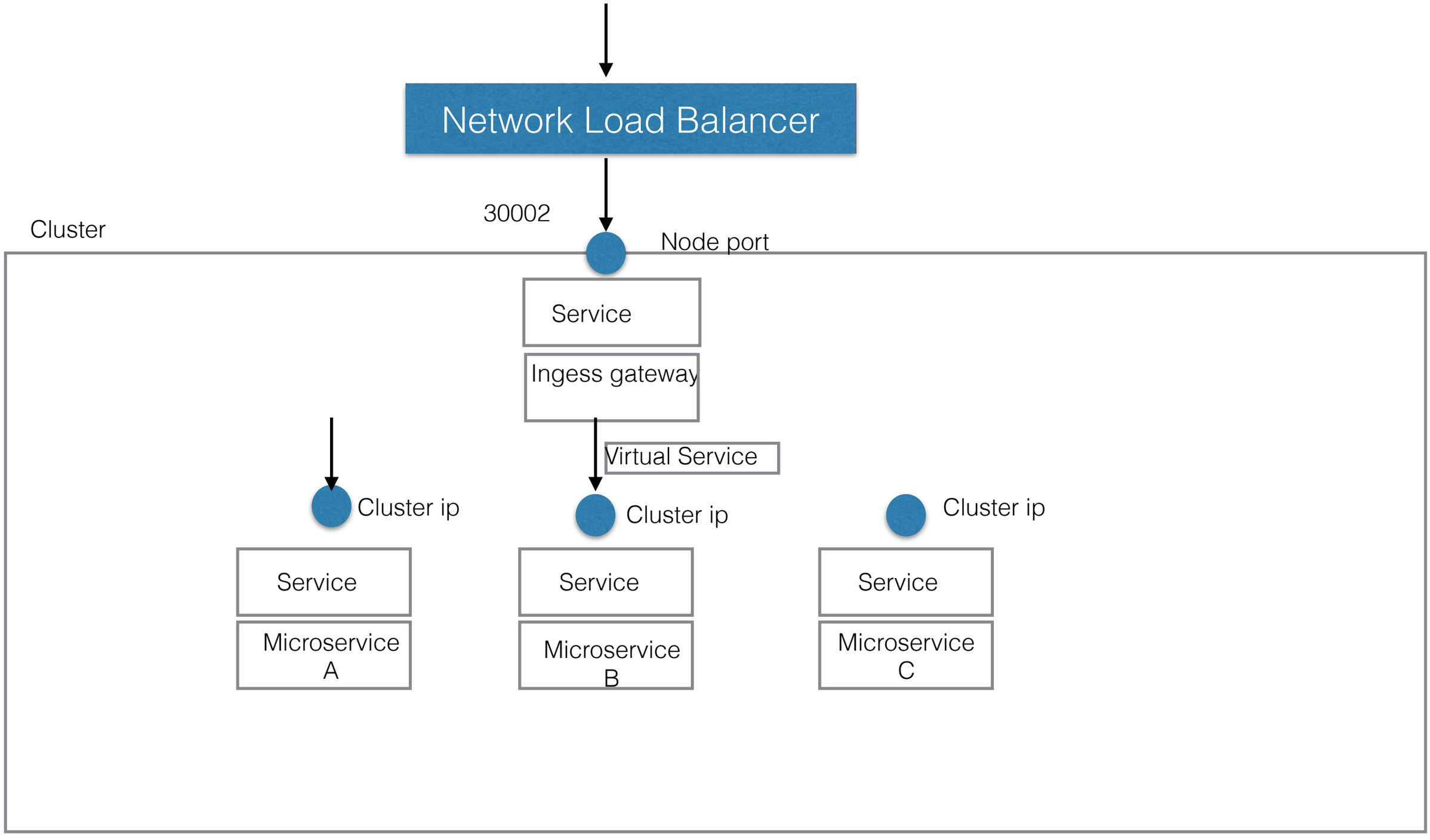
Service

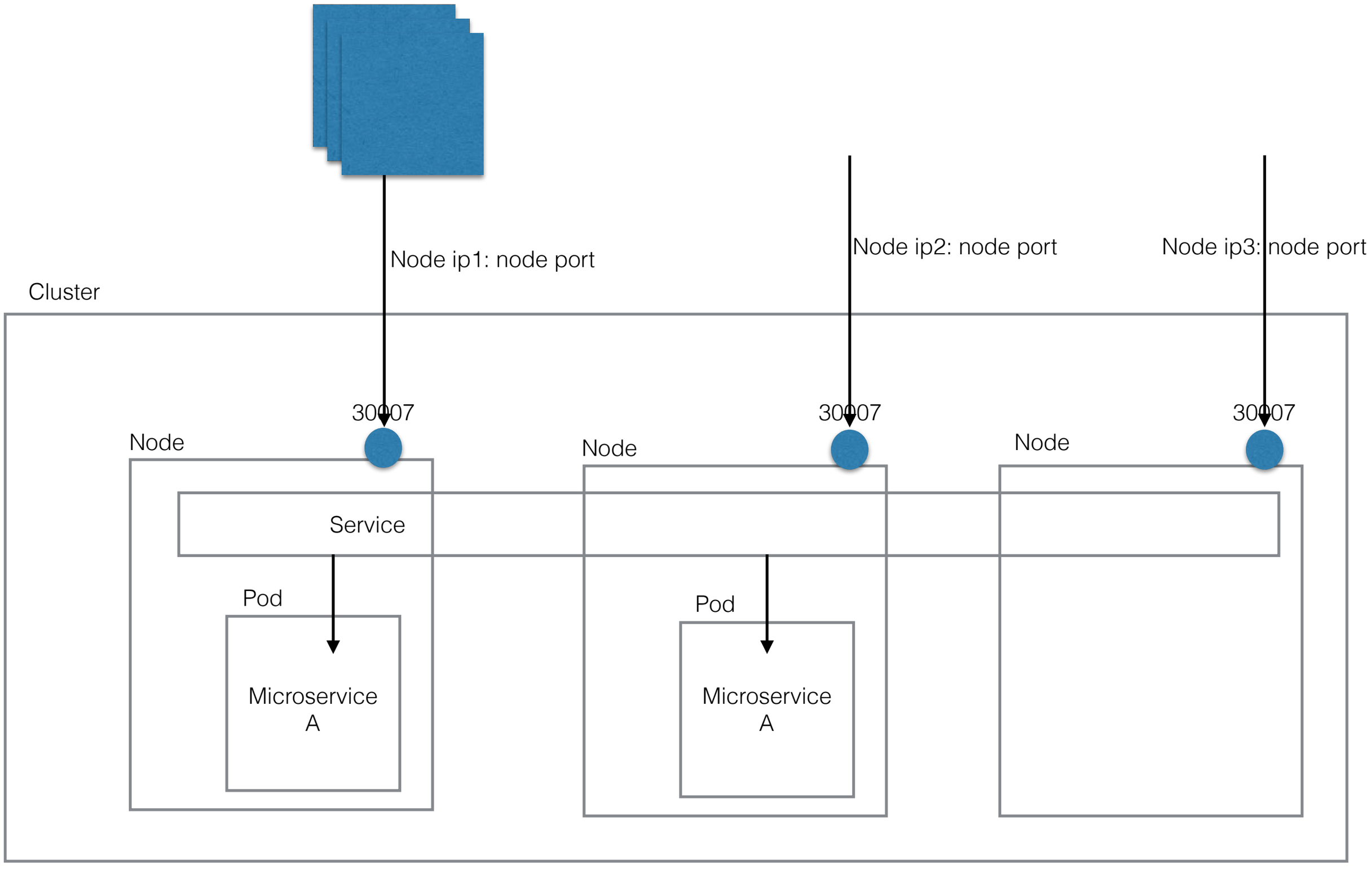
Pod

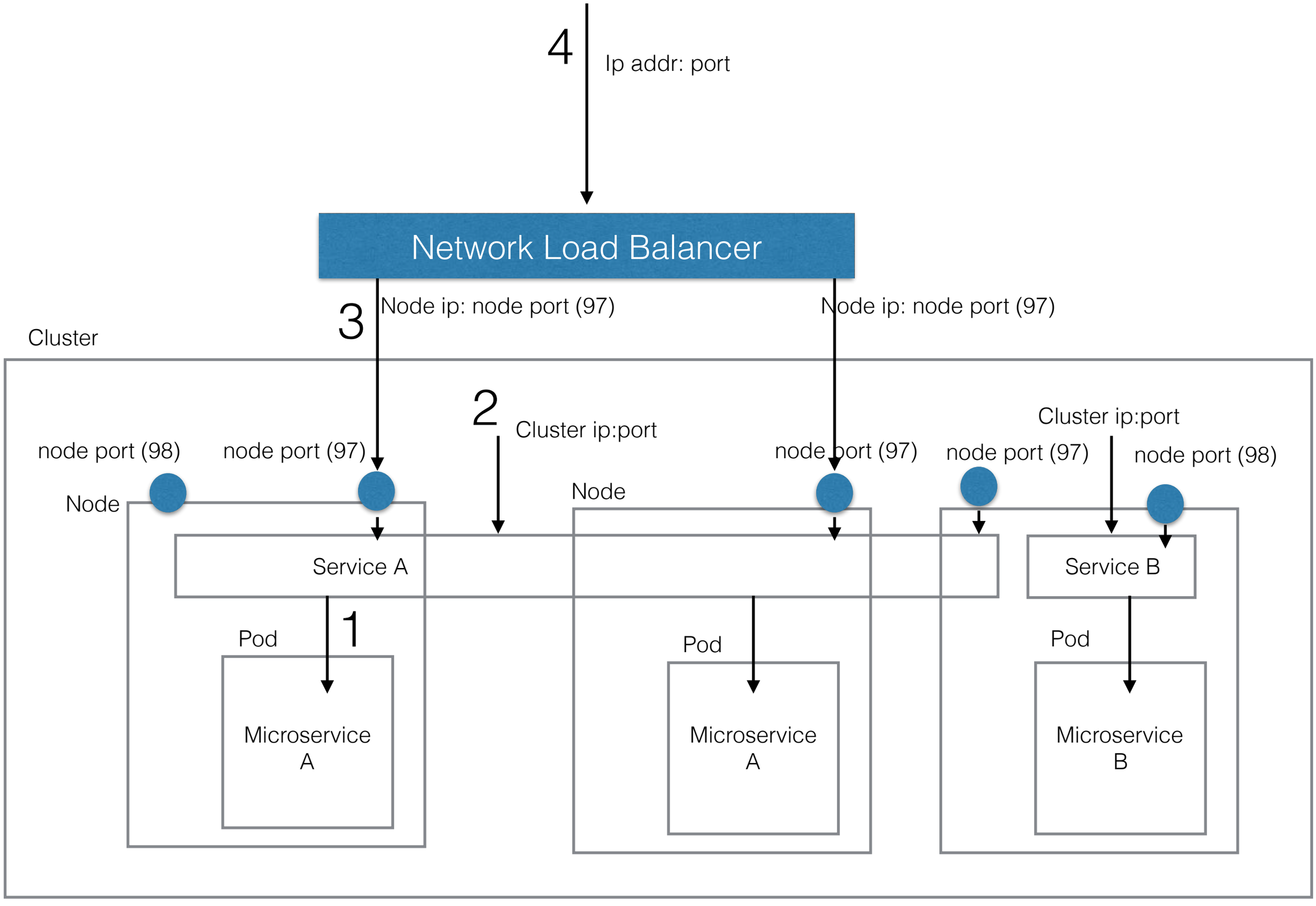
Pod

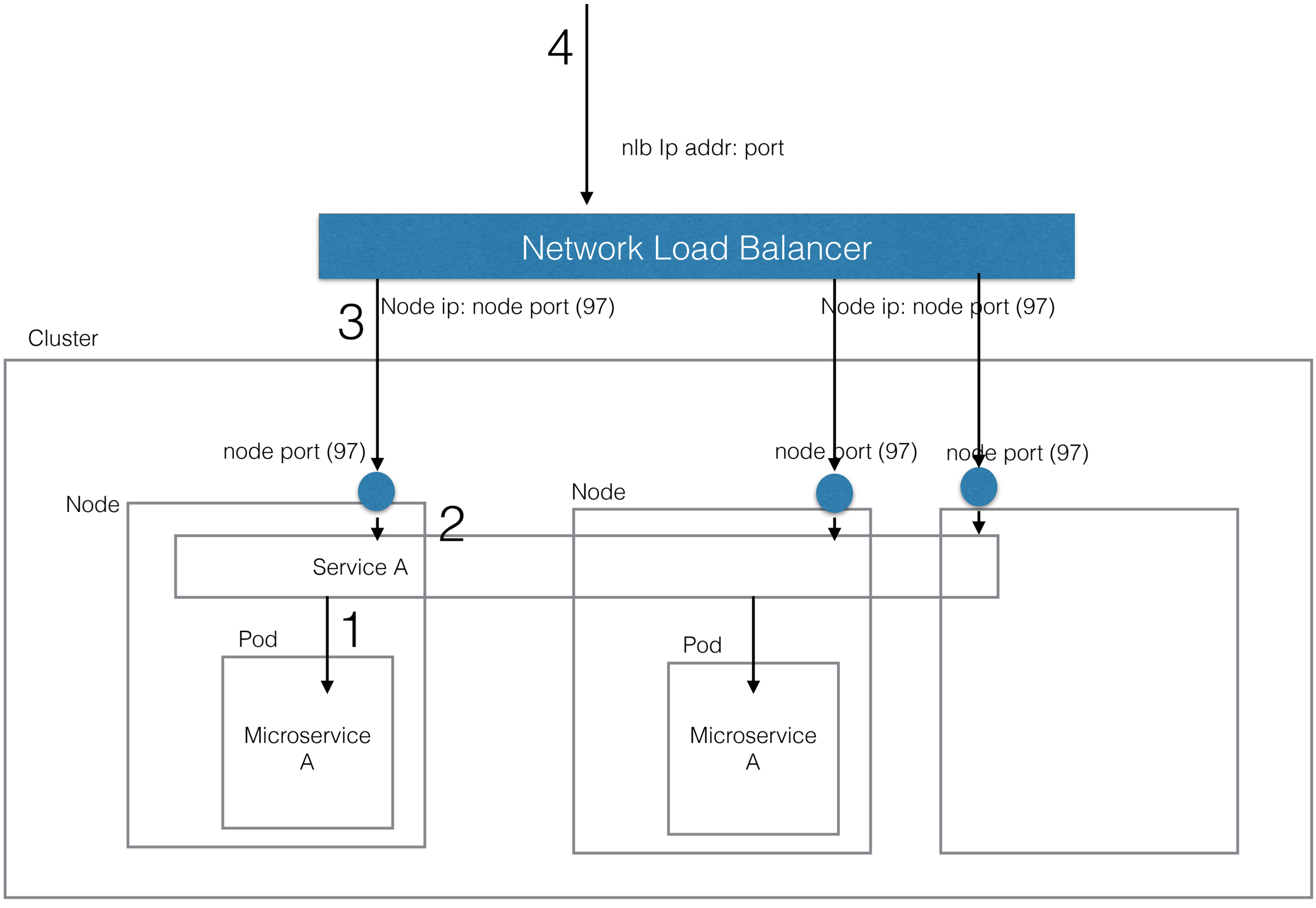
Microservice
A

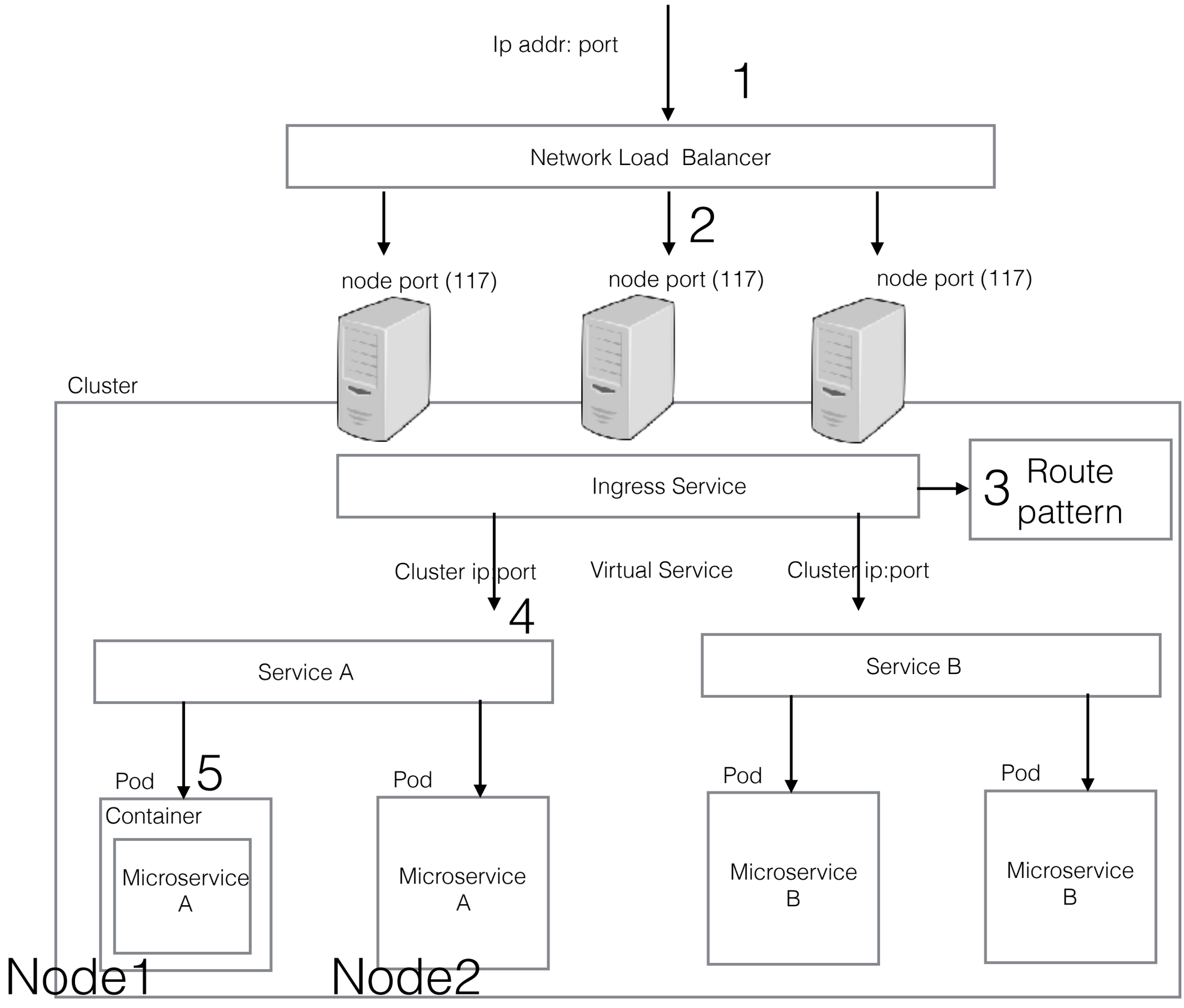
Microservice
A


Network Load Balancer
30002
Cluster
Node port
Service
Ingess gateway
Virtual Service
Cluster ip
Cluster ip
Cluster ip
Service
Service
Service
Microservice A
Microservice B
Microservice C


Node ip1: node port
Node ip2: node port
Node ip3: node port
Cluster
30007
30007
30007
Node
Node
Node
Service
Pod
Pod
Microservice
A
Microservice
A


4
Ip addr: port
Network Load Balancer
3
Node ip: node port (97)
Node ip: node port (97)
Cluster
2
Cluster ip:port
Cluster ip:port
node port (98)
node port (97)
node port (97)
node port (97)
node port (98)
Node
Node
Service A
Service B
1
Pod
Pod
Pod
Microservice A
Microservice A
Microservice B


4
nlb Ip addr: port
Network Load Balancer
3
Node ip: node port (97)
Node ip: node port (97)
Cluster
node port (97)
node port (97)
node port (97)
Node
Node
2
Service A
1
Pod
Pod
Microservice A
Microservice A


Ip addr: port
1
Network Load Balancer
2
node port (117)
node port (117)
node port (117)
Cluster
Ingress Service
3 Route pattern
Cluster ip:port
Virtual Service
Cluster ip:port
4
Service A
Service B
5
Pod
Container
Microservice A
Pod
Microservice A
Pod
Microservice B
Pod
Microservice B
Node1
Node2

1.Docker, k8s, Istio

Outside the cloud (LB Ip addr: port)

Cloud

Load Balancer

Within the cluster(Any Node IPaddr: port)

node port (117)

node port (117)

node port (117)

Cluster

4. Ingress

Ingress Service

Route pattern

2. Enable mesh

Namespace

Virtual Service(Cluster ip:port)

5. Virtual Service

Service A

3. Deploy Pod, service

Pod

Container

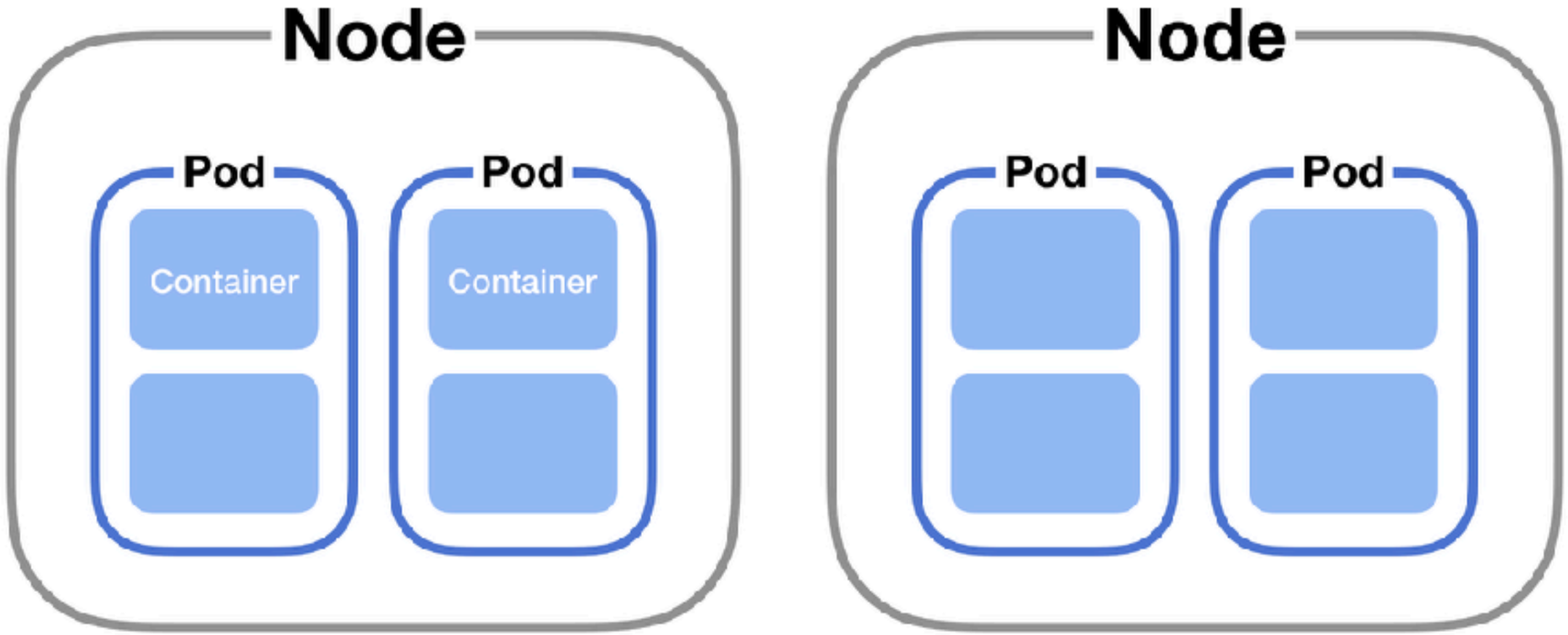
Microservice Date v1

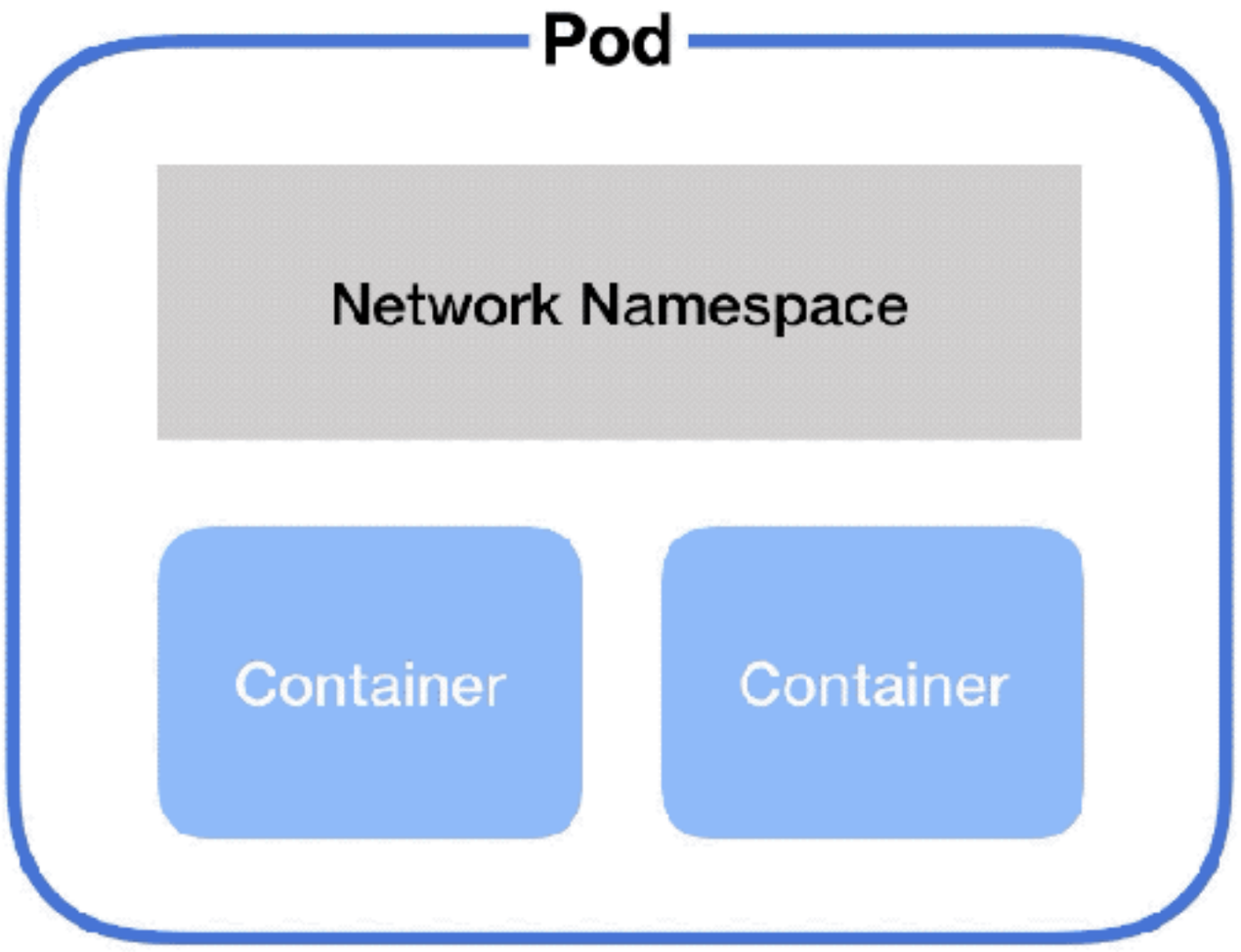
Pod

Microservice Date v2

Service Discovery

Cluster




Pod
Network Namespace
Container
Container

Node

eth0

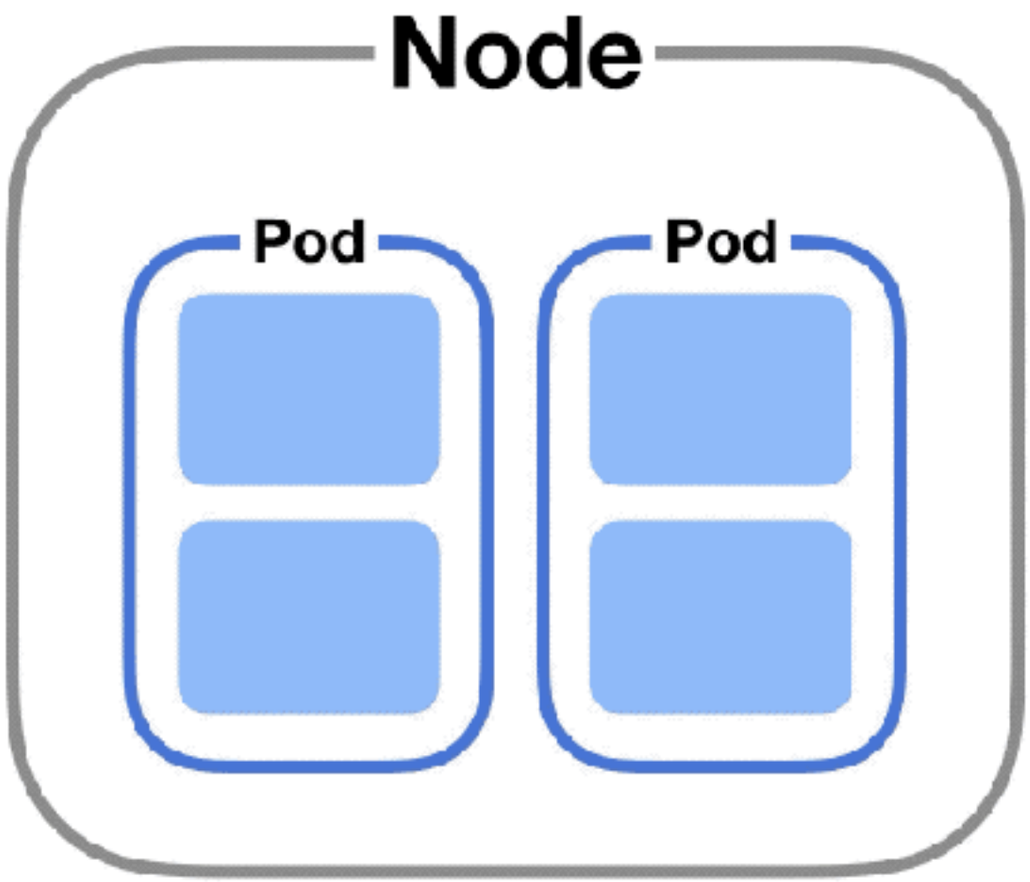
Pod 100.68.1.1

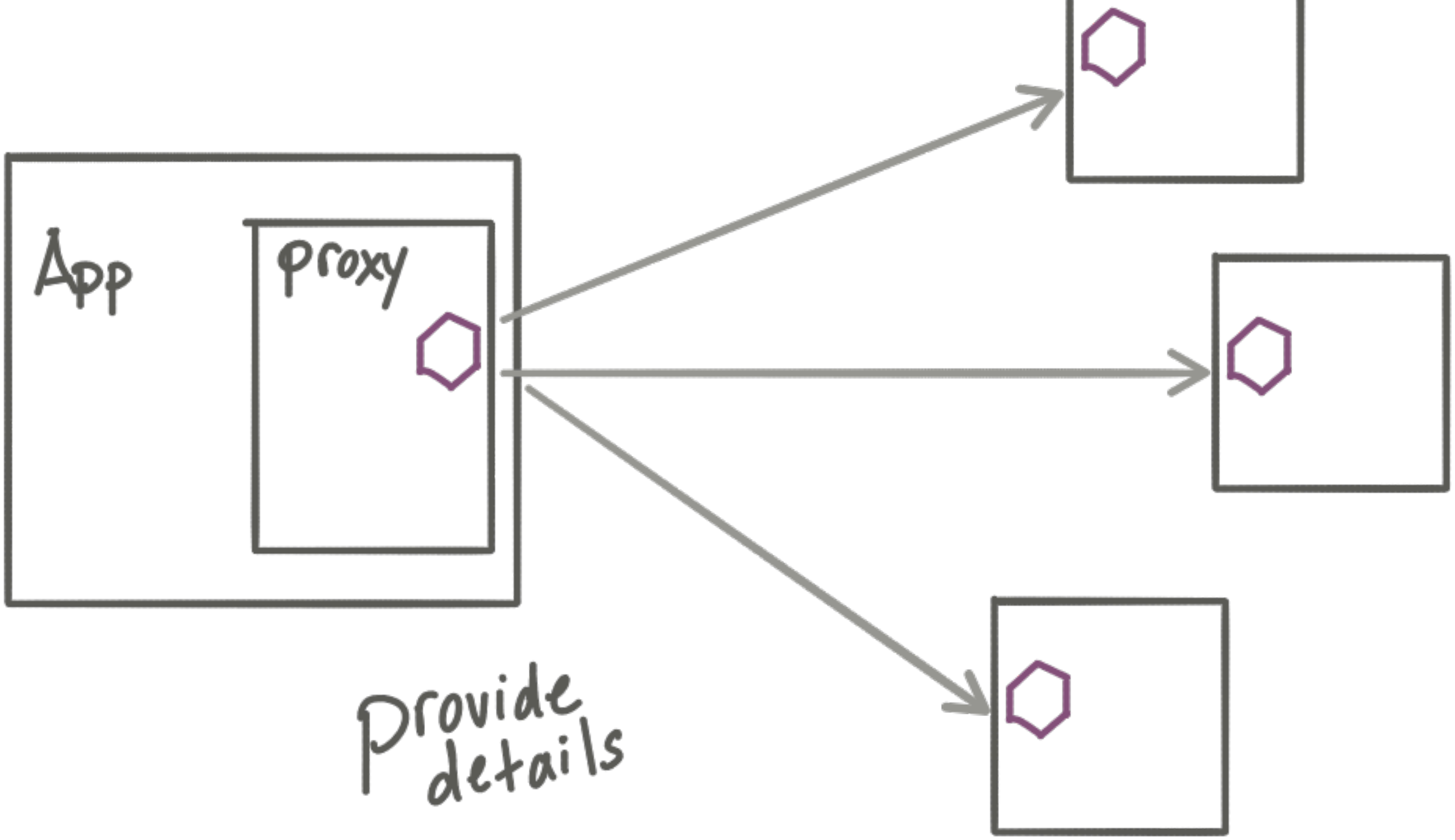
Network Namespace

Container

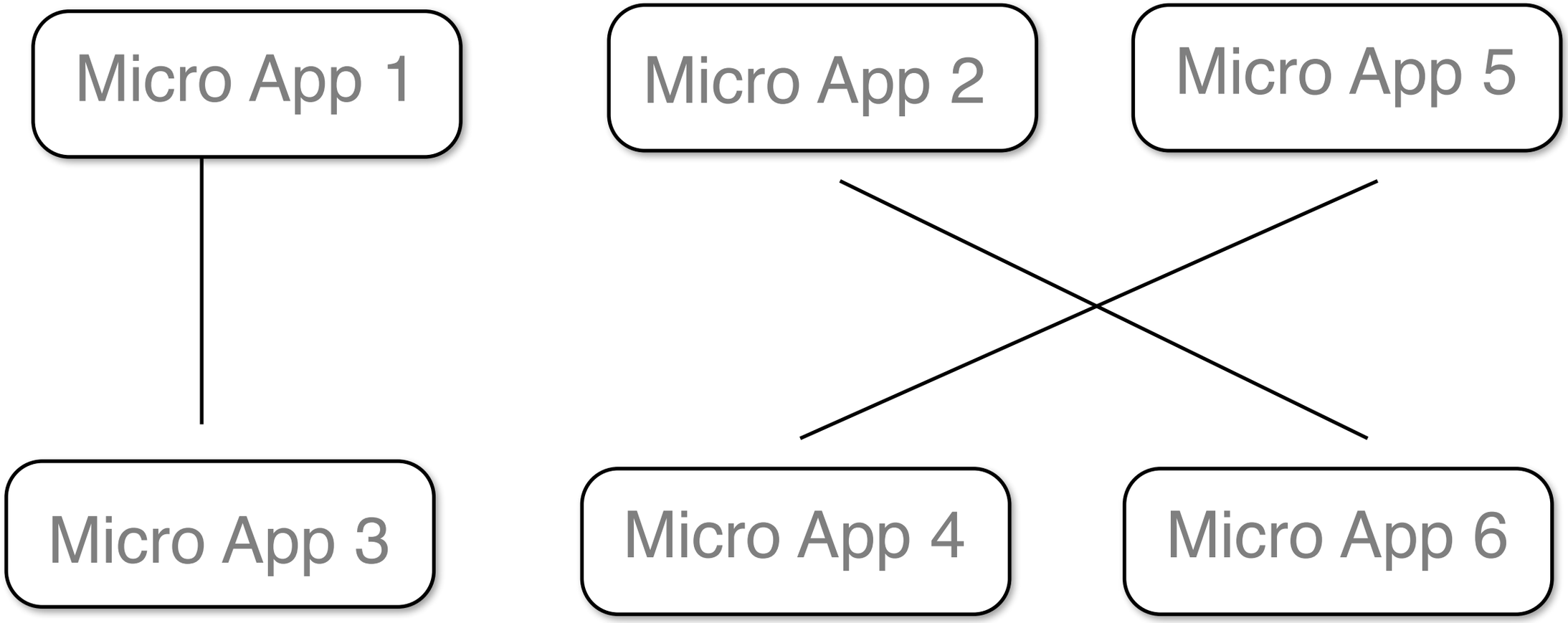
Container

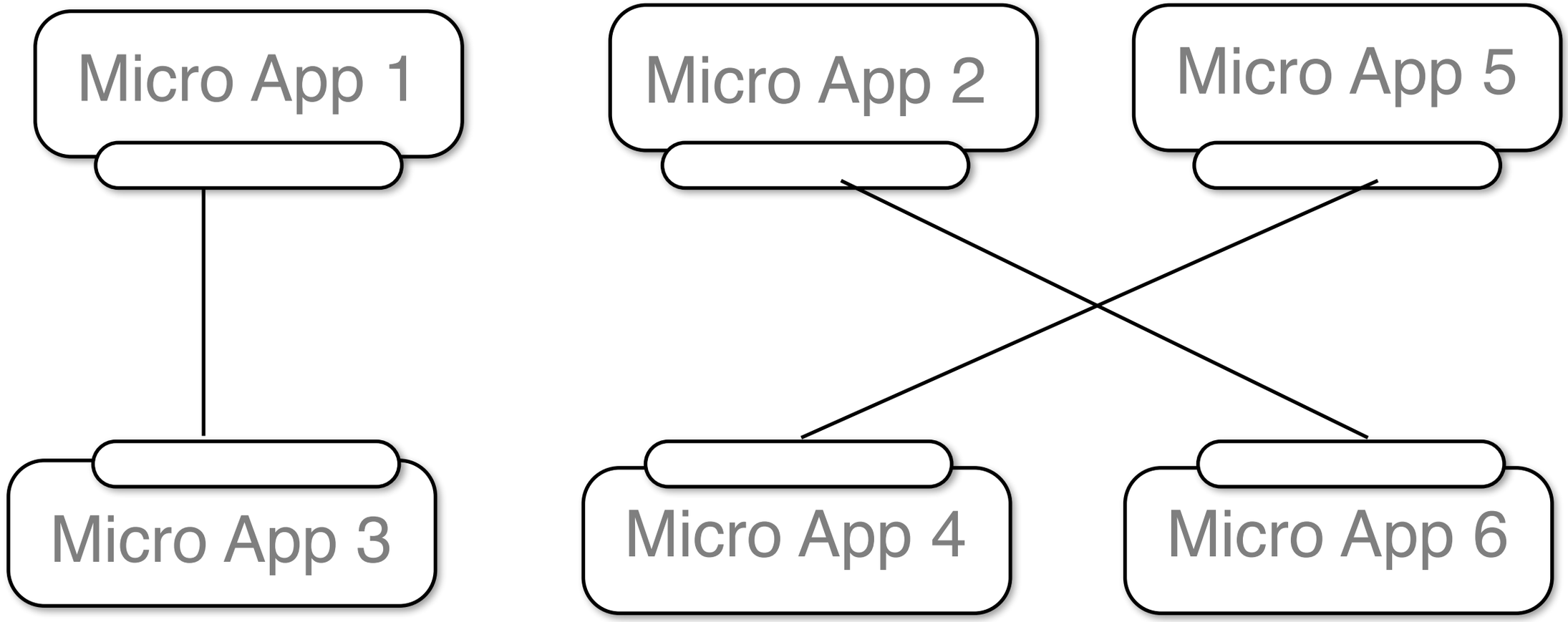
100.68.1.xxx

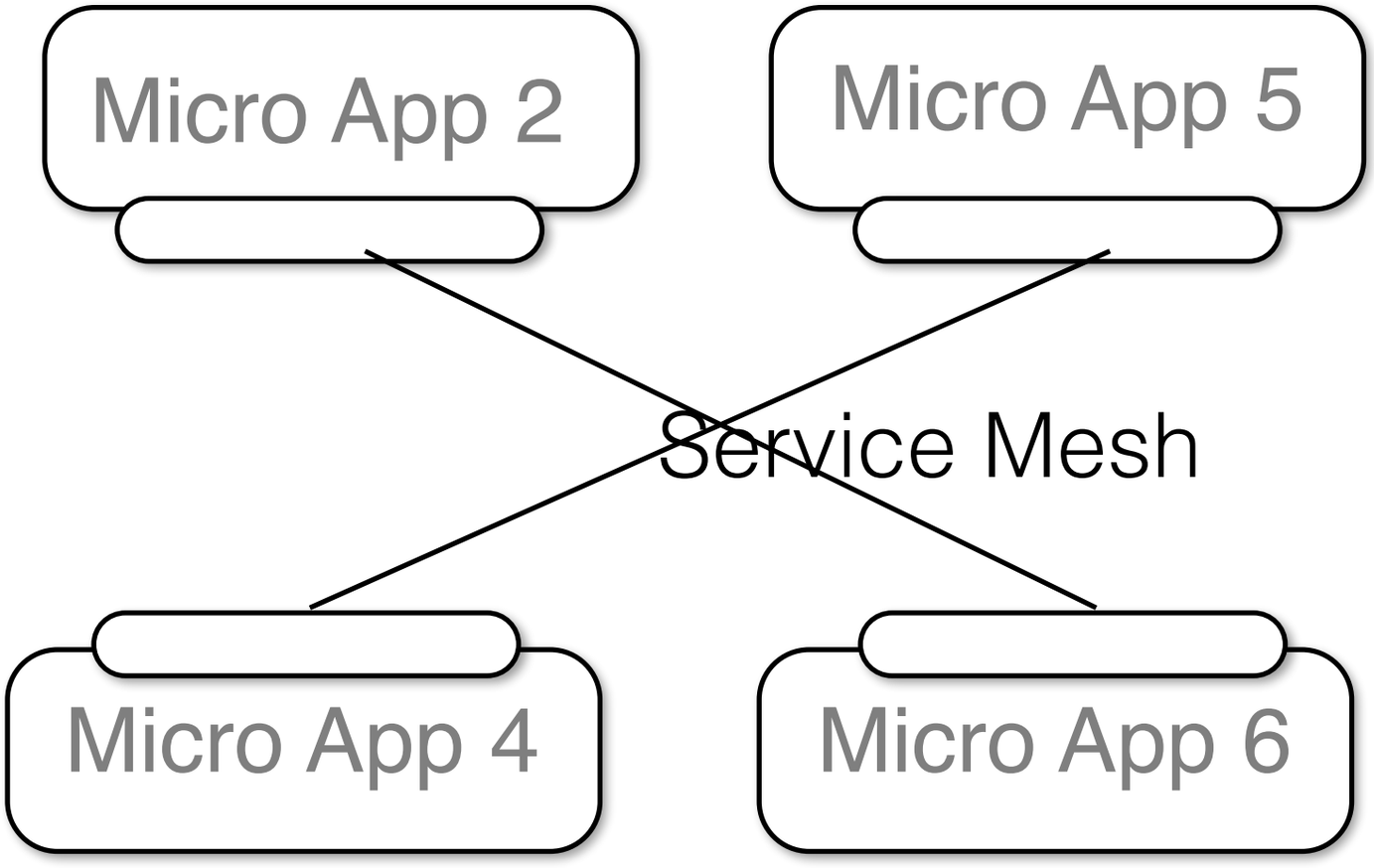


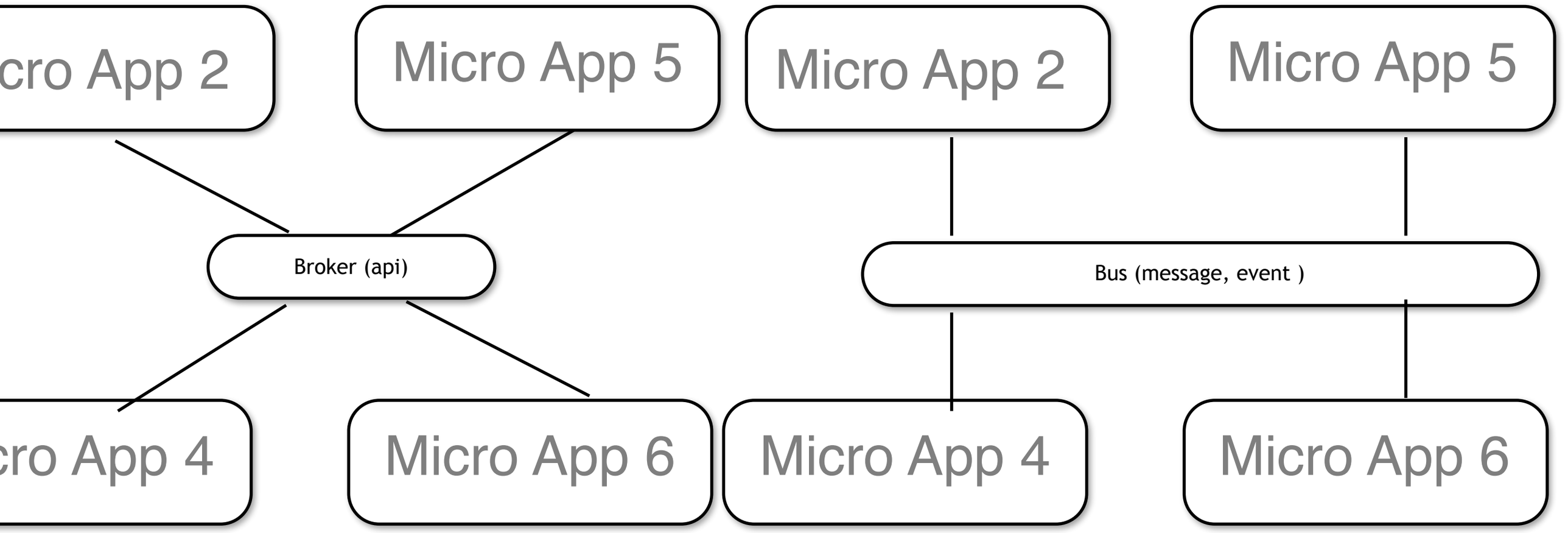


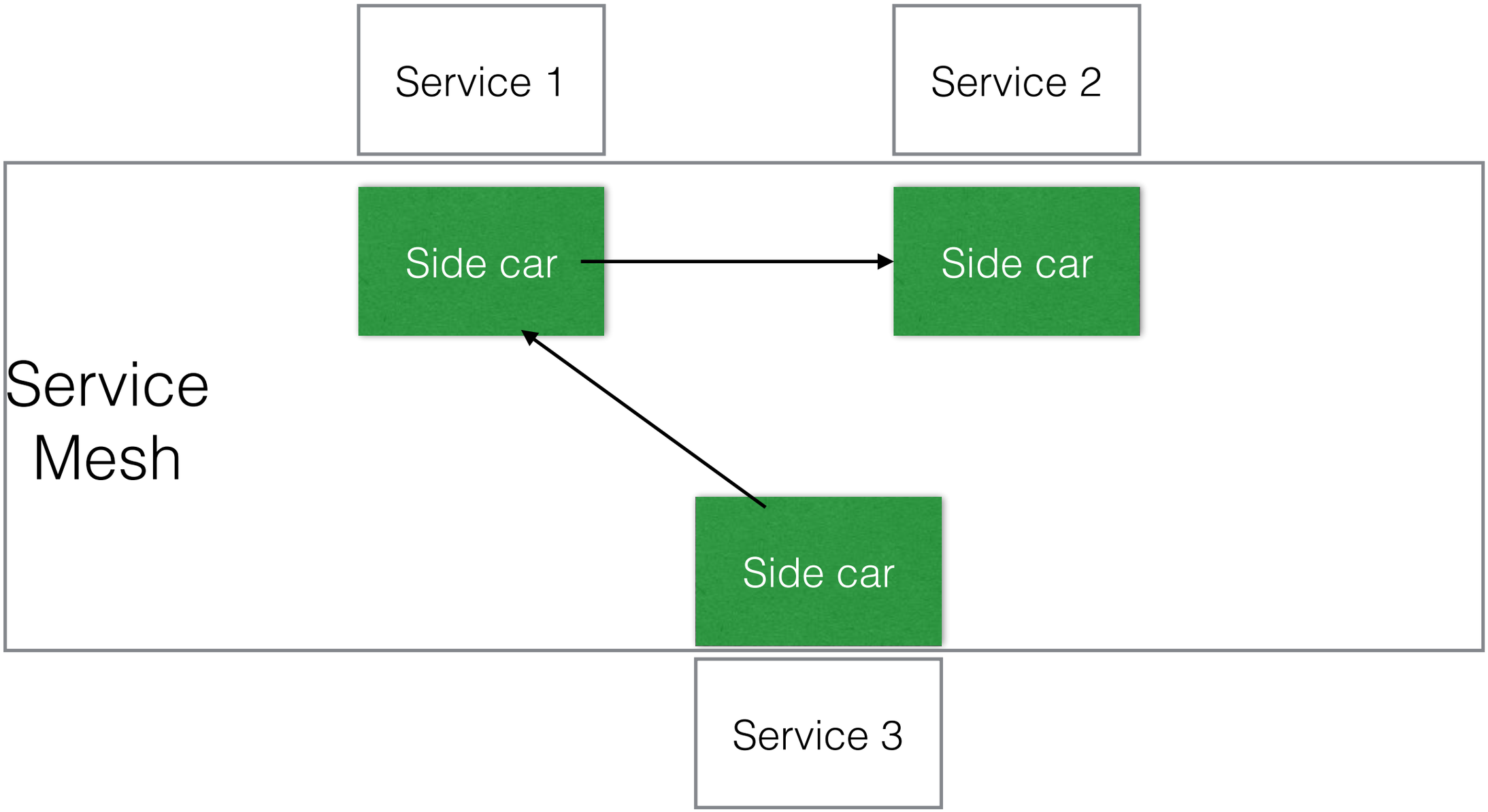
Service Mesh

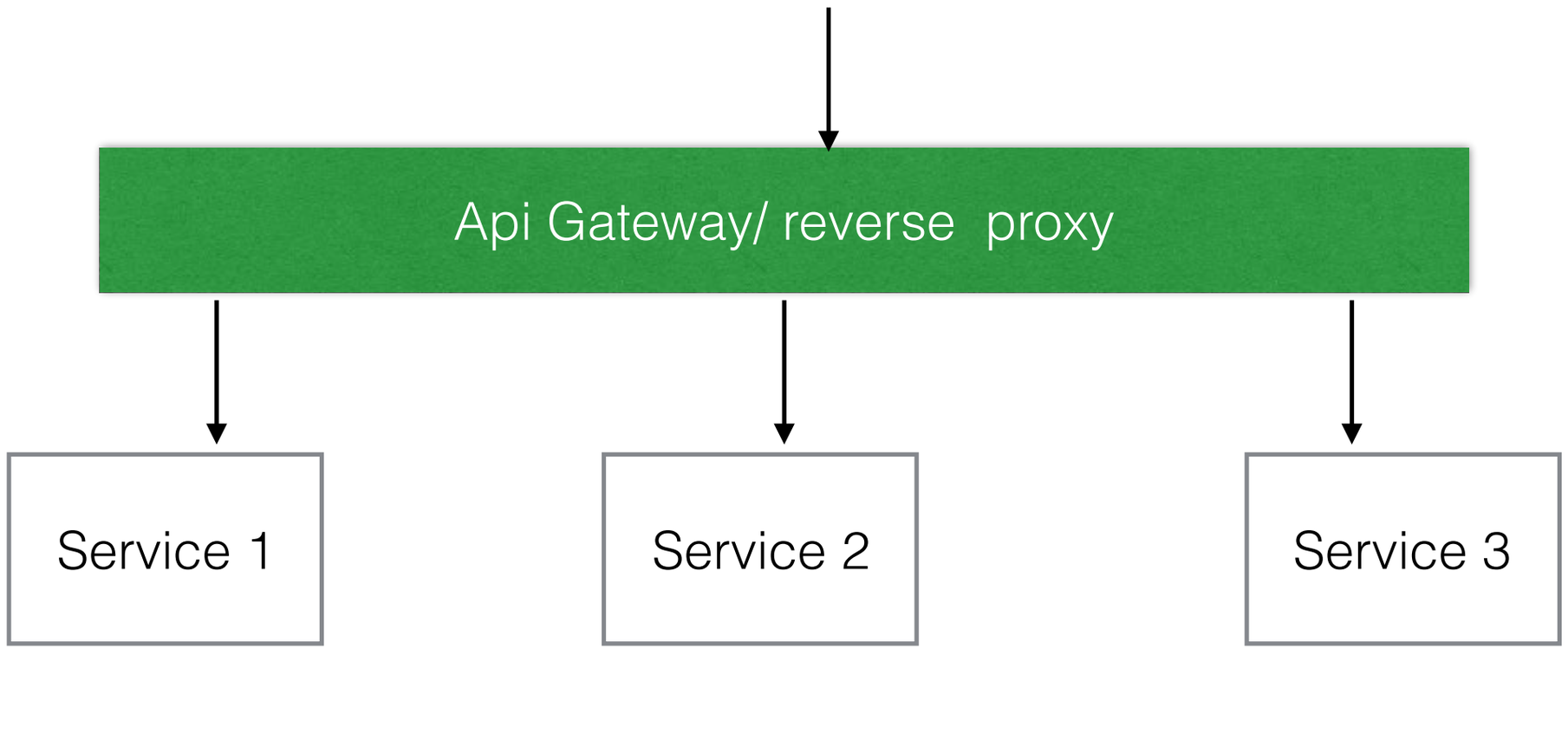

Micro App 1
Micro App 2
Micro App 5
Micro App 3
Micro App 4
Micro App 6


Micro App 1
Micro App 2
Micro App 5
Micro App 3
Micro App 4
Micro App 6


Micro App 2
Micro App 5
Service Mesh
Micro App 4
Micro App 6


cro App 2
Micro App 5
Micro App 2
Micro App 5
Broker (api)
Bus (message, event)
cro App 4
Micro App 6
Micro App 4
Micro App 6


Service 1
Service 2
Side car
Side car
Service
Mesh
Side car
Service 3


Api Gateway/ reverse proxy
Service 1
Service 2
Service 3

Service 1

Service 2

Service 3

API

1

Service 1

Service 2

Message Q

Service 3

2

Event Driven Arch *

3

Service 1

Service 2

Ingress /reverse proxy/ api gateway

Service 3

Broker Arch *

4

Service 1

Service 2

Side car

Side car

Service Mesh

Side car

Service 3

Mesh Arch *

Message Q

Pod

Pod

Pod

Service 1

Service 2

Service 4

Side car

Side car

Service Mesh

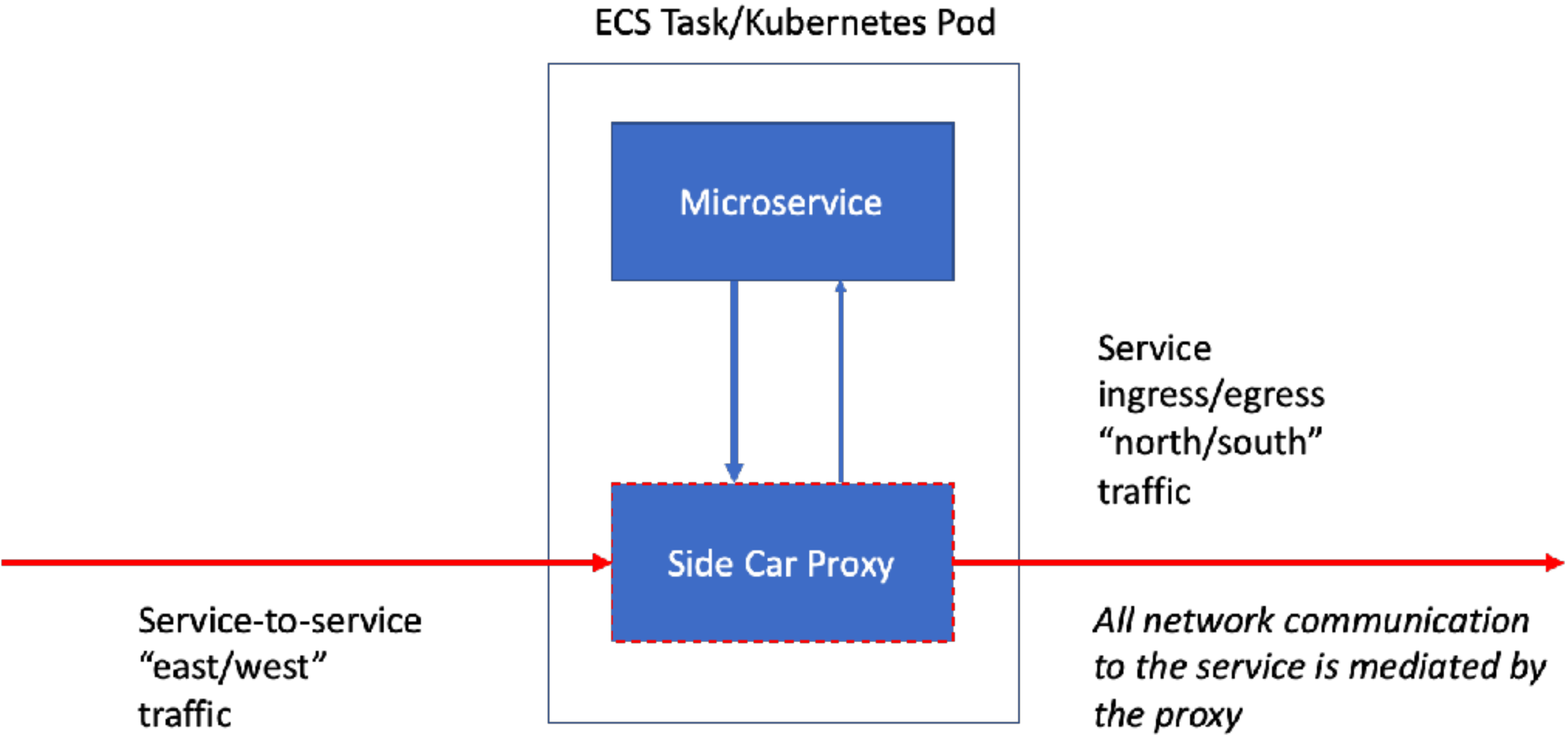
Side car

Service 3

Pod

Ingress/ api gateway/ reverse proxy

Load Balancer

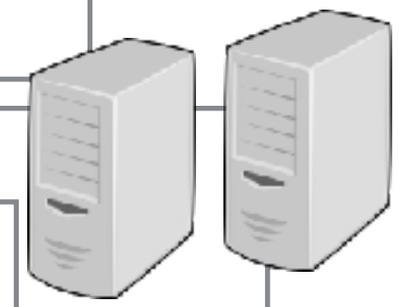

ECS Task/Kubernetes Pod
Microservice
Side Car Proxy
Service
ingress/egress
“north/south”
traffic
Service-to-service
“east/west”
traffic
All network communication
to the service is mediated by
the proxy

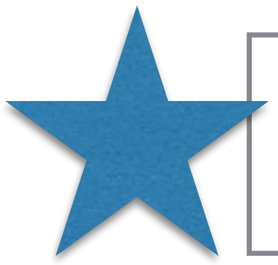
http://172.17.0.22:80/date

North- South

LoadBalancer : L4 (tcp)

Cluster





Ingress : L7 (http)

Virtual Service

http://10.97.33.128:8080/date

East - west

Service

Side car

Microservice
POD

Edge Ingres

Service Mesh Control Plane

Service Mesh data plane

Integration

Integration

Integration

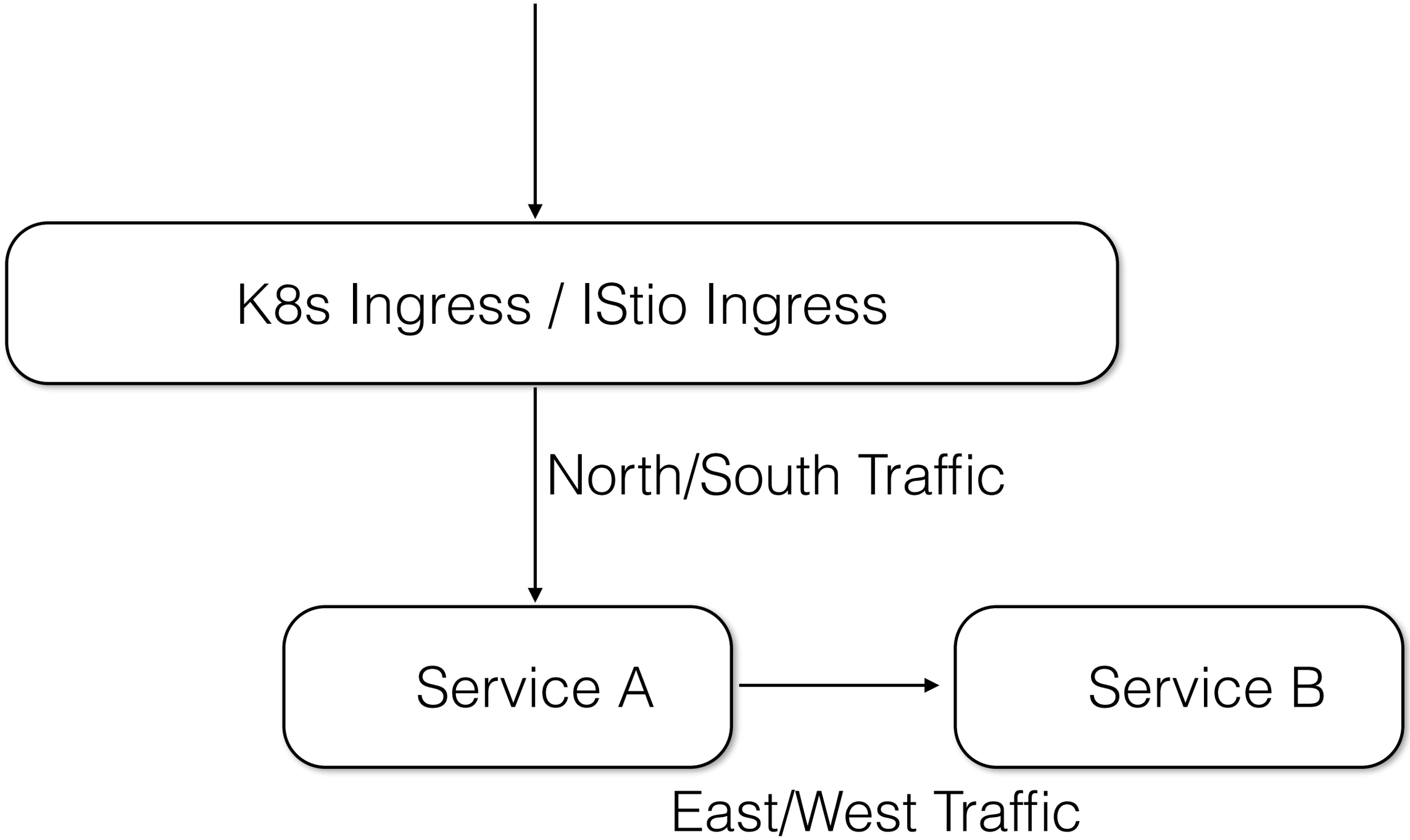
Integration

Integration

North-South Traffic

East-West traffic

Istio


K8s Ingress / IStio Ingress
North/South Traffic
Service A
Service B
East/West Traffic

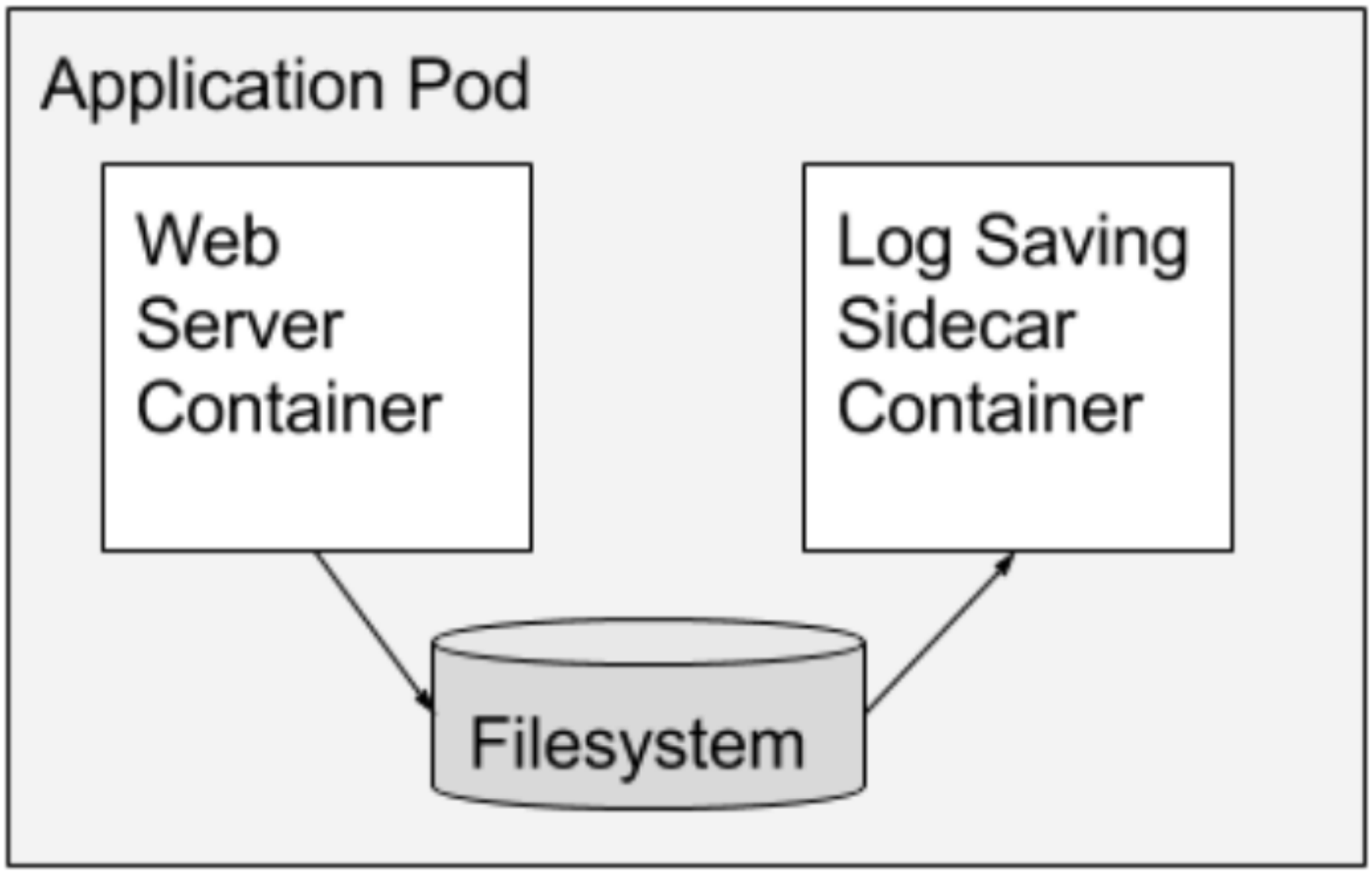
Cross Cutting Concerns

Domain Logic

Authorization
Exception Handling
Versioning
Tracing
Protocol Translation
…

Proxy

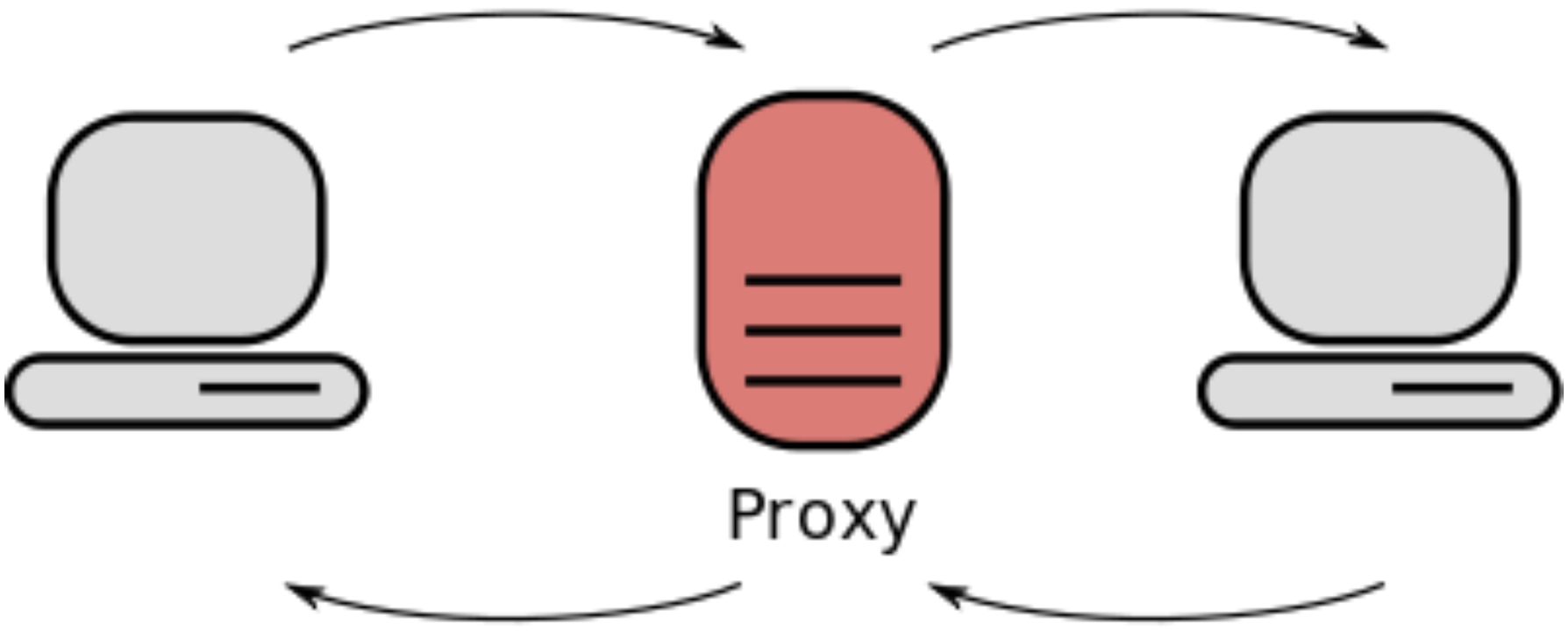
Sidecar


Application Pod
Web
Server
Container
Log Saving
Sidecar
Container
Filesystem

- In a sidecar pattern, the functionality of the main container is extended or enhanced by a sidecar container without strong coupling between two.

- eg. main container is a web server which is paired with a log saver sidecar container that collects the web server’s logs from local disk and streams them to centralized log collector.

Proxy pattern



- Proxy pretends being an object when in fact it's not. There are several types of proxies:

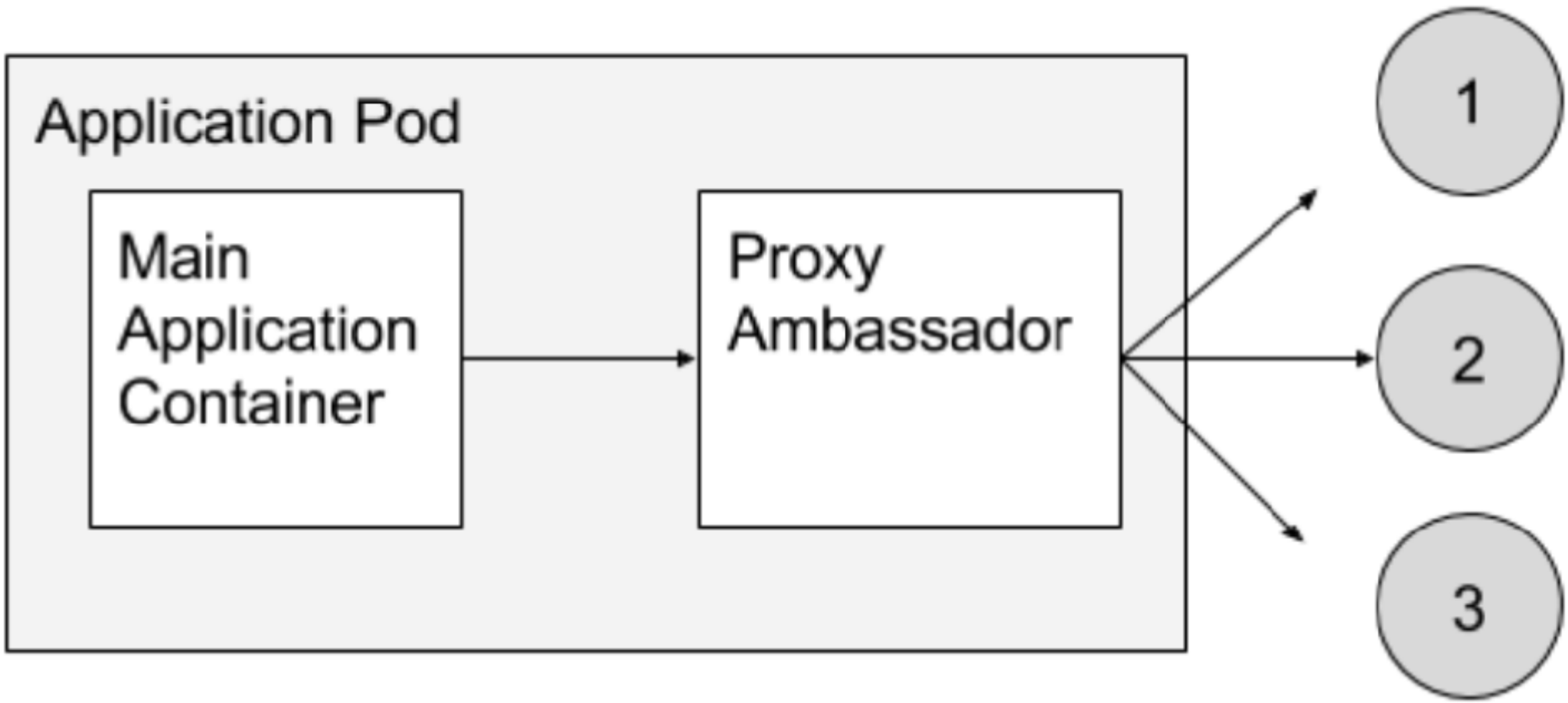
- Remote proxy ("Ambassador") - instead of communicating between server's and client's classes we create a proxy on client side and server side, and only the proxies communicate

- Virtual proxy - creates expensive object on first demand

- Protection proxy - to control access

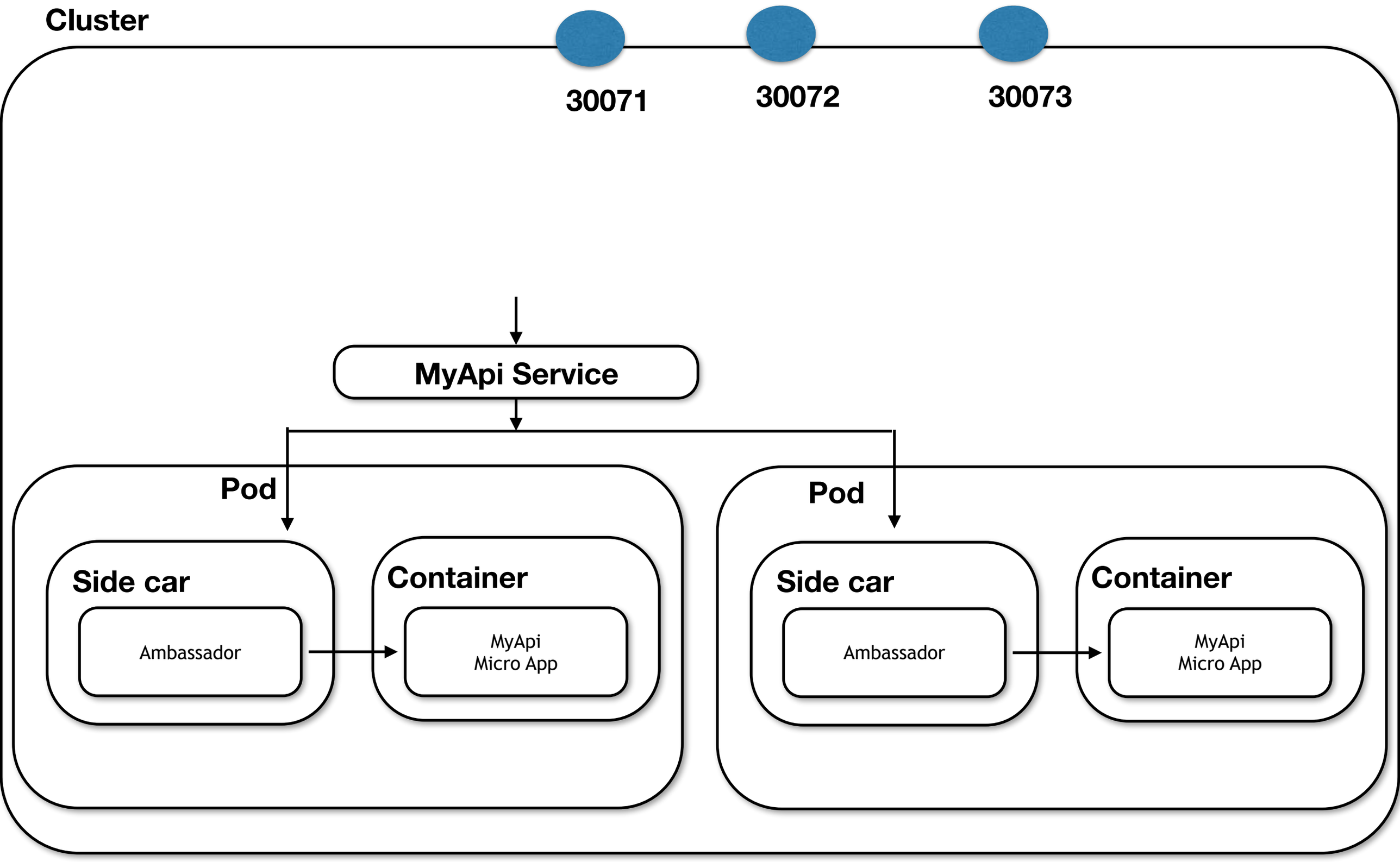
- Smart proxy - enrich an object

Ambassador pattern



- an Ambassador container act as a proxy between two different types of main containers. Typical use case involves proxy communication related to load balancing and/or sharding to hide the complexity from the application.

Deployment Diagram



Deployment Diagram

API Gateway

Network Load Balancer

Node Port

Node Port

Kubernetis Ingress/ Istio Ingress / nginx /...(reverse proxy)

MyApi Service

Pod

Side car

Ambassador

Container

MyApi
Micro App

Pod

Side car

Ambassador

Container

MyApi
Micro App

Service 1

Service 2

Service 3

API

1

Service 1

Service 2

Message Q

Service 3

2

Event Driven Arch *

3

Service 1

Service 2

Ingress /reverse proxy/ api gateway

Service 3

Broker Arch *

4

Service 1

Service 2

Side car

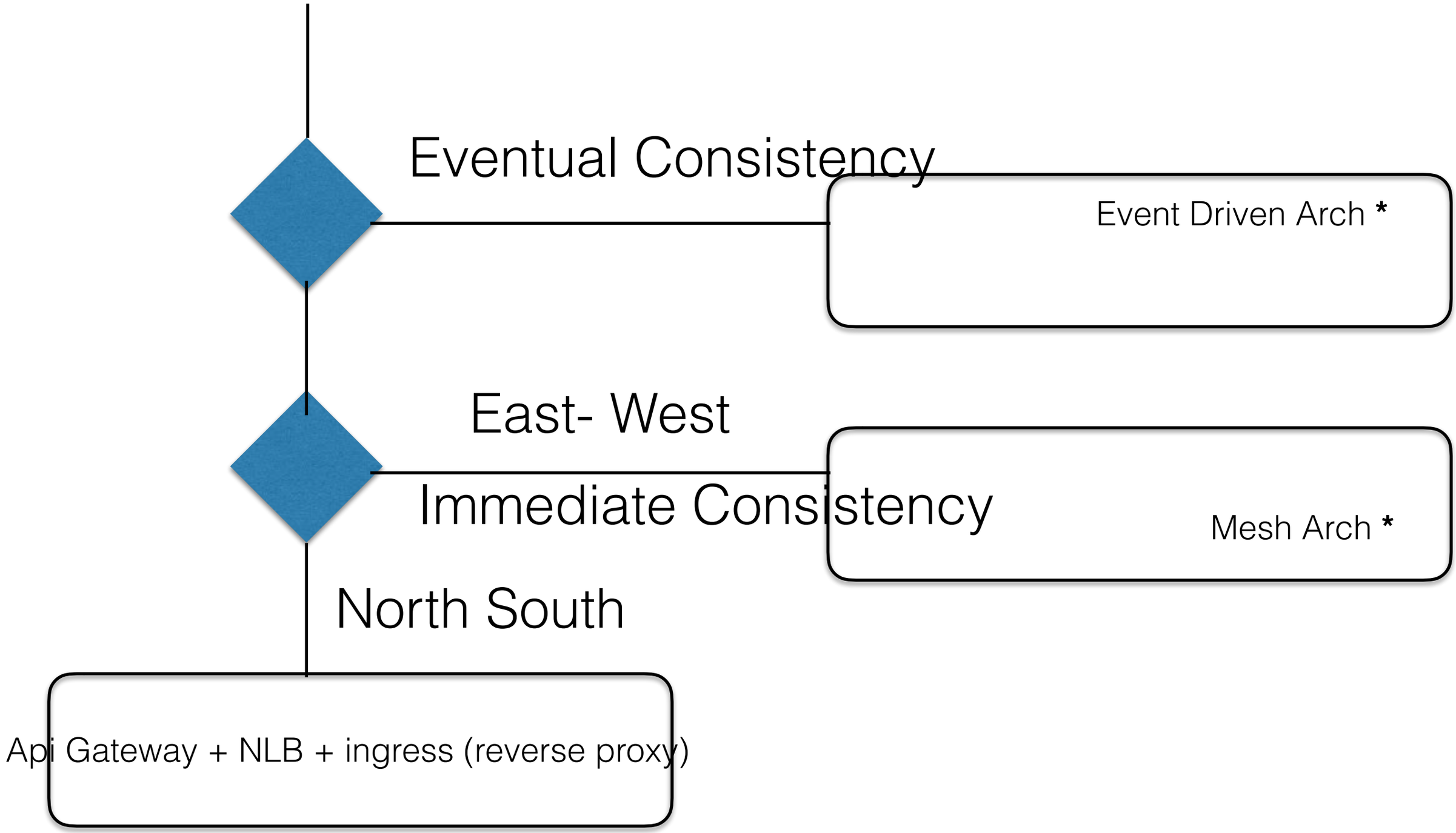
Side car

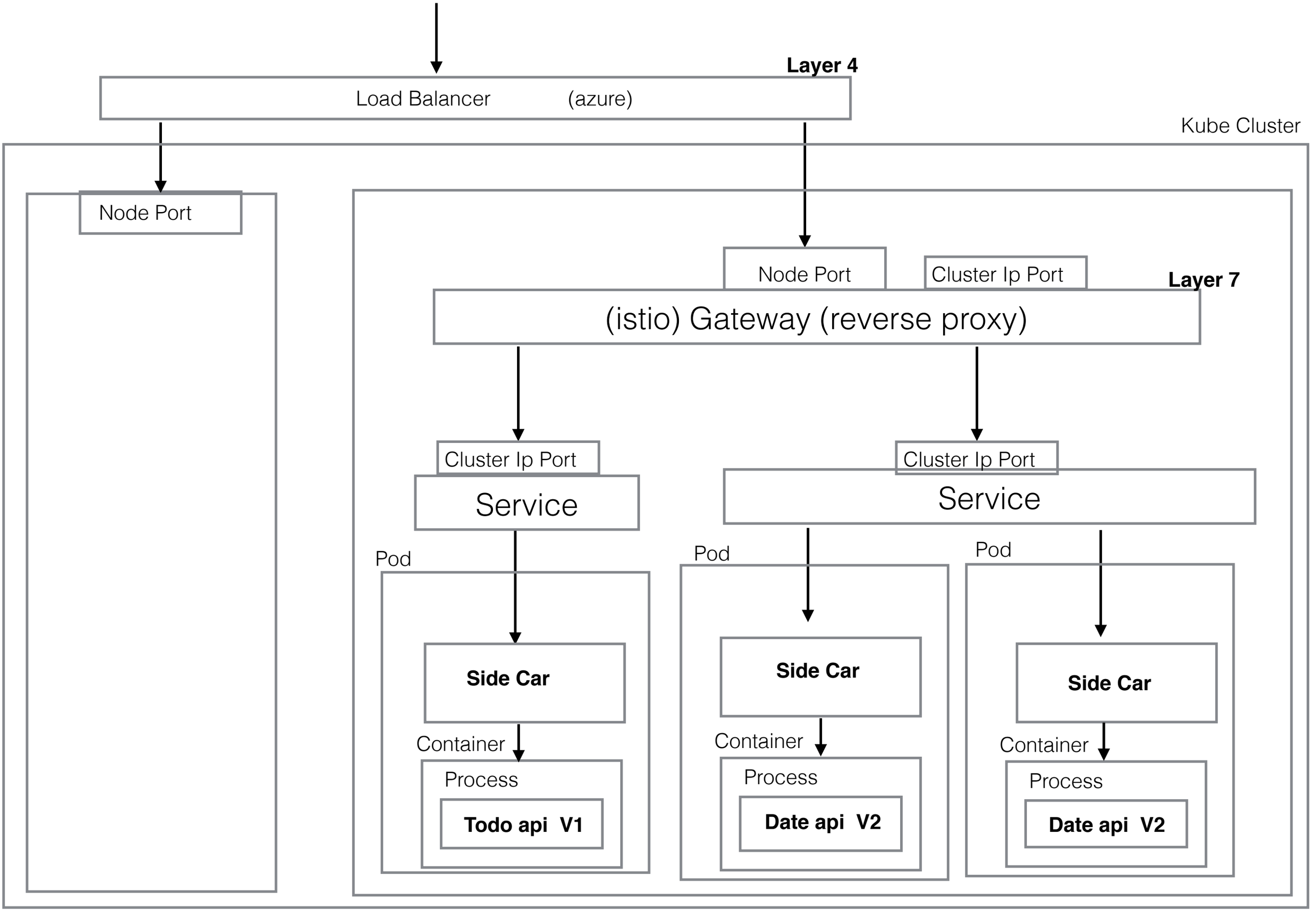
Service Mesh

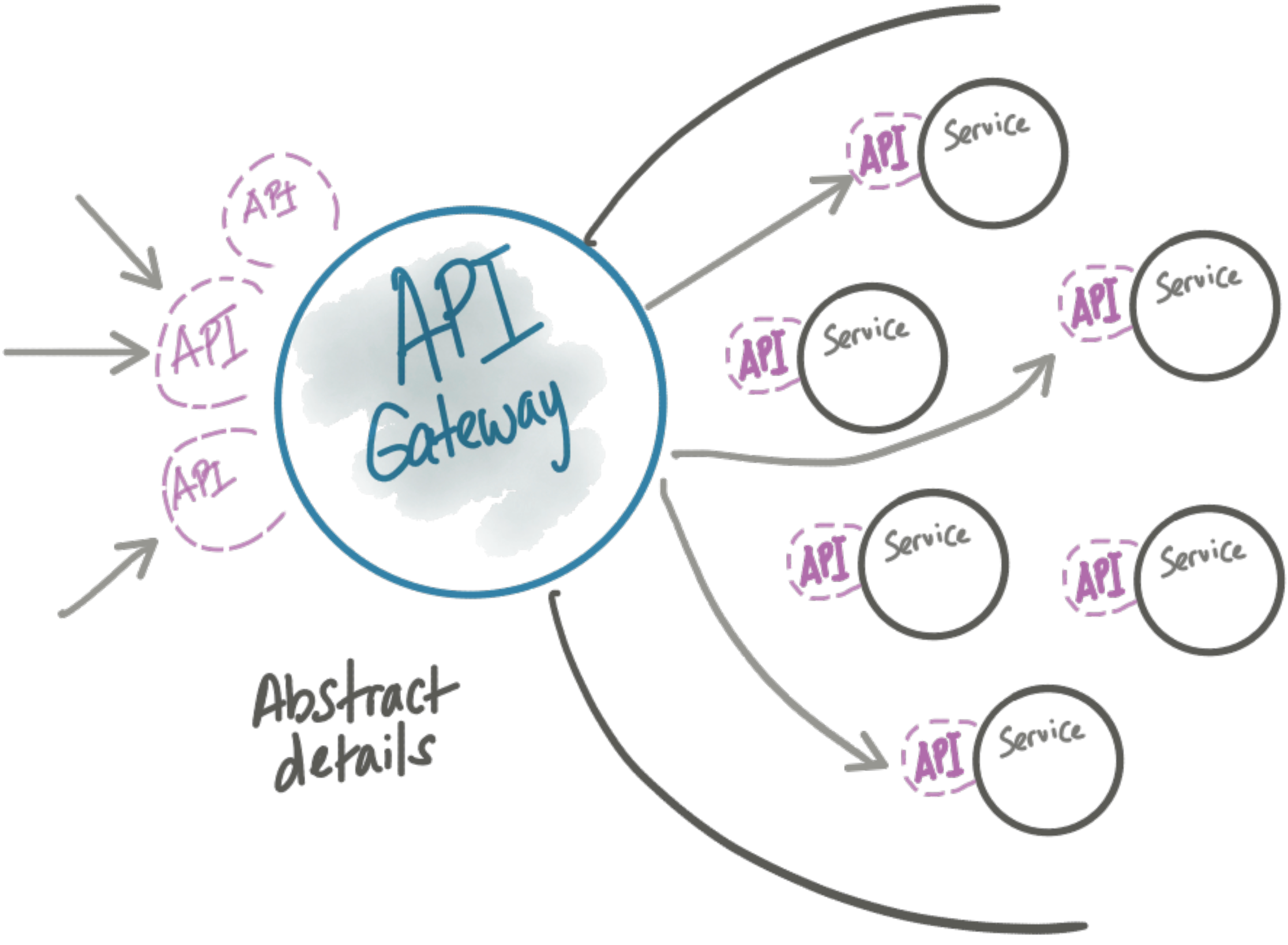
Side car

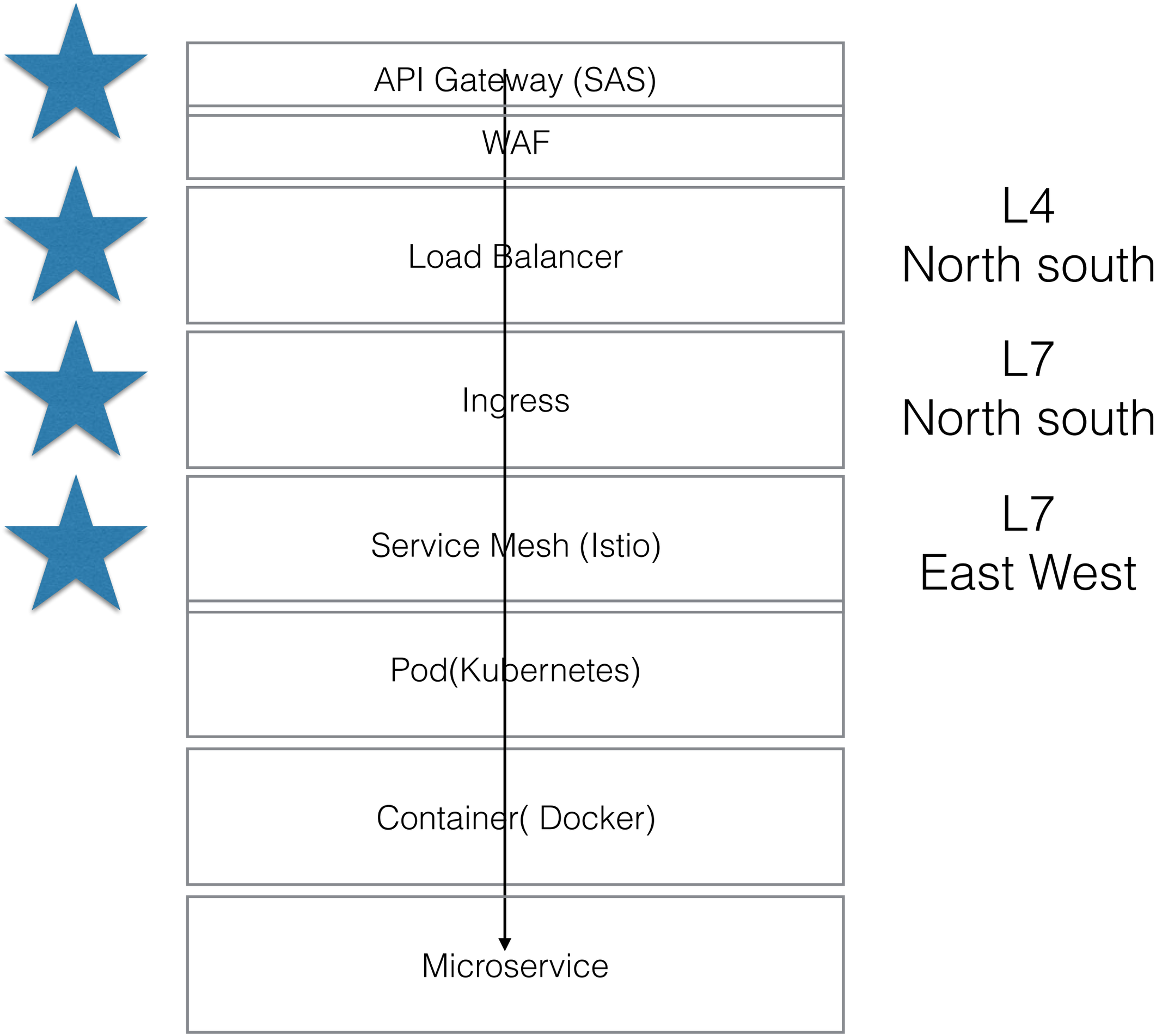
Service 3

Mesh Arch *


Eventual Consistency
Event Driven Arch *
East- West
Immediate Consistency
Mesh Arch *
North South
Api Gateway + NLB + ingress (reverse proxy)


Layer 4
Load Balancer (azure)
Kube Cluster
Node Port
Node Port
Cluster Ip Port
Layer 7
(istio) Gateway (reverse proxy)
Cluster Ip Port
Service
Pod
Side Car
Container
Process
Todo api V1
Cluster Ip Port
Service
Pod
Side Car
Container
Process
Date api V2
Pod
Side Car
Container
Process
Date api V2


API
API
API
API Gateway
API
Service
API
Service
API
Service
API
Service
API
Service
API
Service
Abstract details

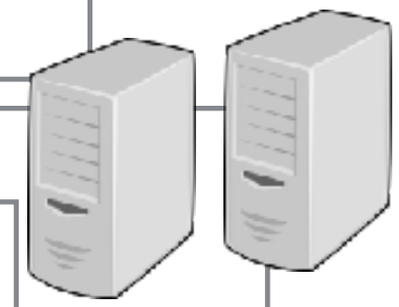

API Gateway (SAS)
WAF
Load Balancer
Ingress
Service Mesh (Istio)
Pod(Kubernetes)
Container(Docker)
Microservice
L4
North south
L7
North south
L7
East West

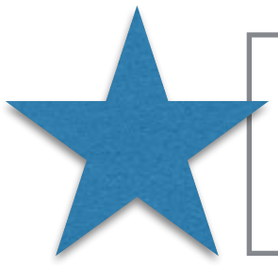
http://172.17.0.22:80/date

North- South

LoadBalancer : L4 (tcp)

Cluster





Ingress : L7 (http)

Virtual Service

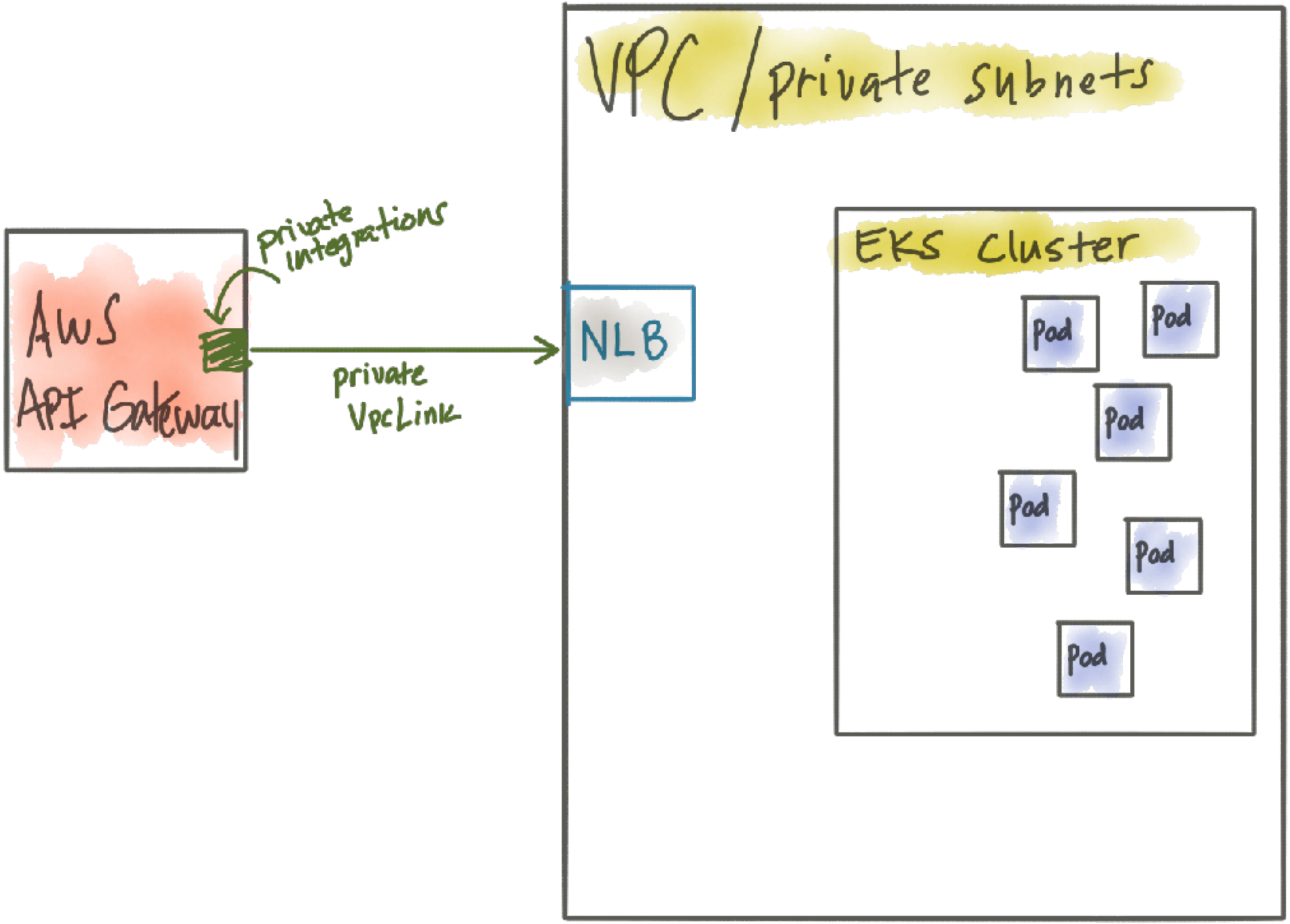
http://10.97.33.128:8080/date

East - west

Service

Side car

Microservice
POD

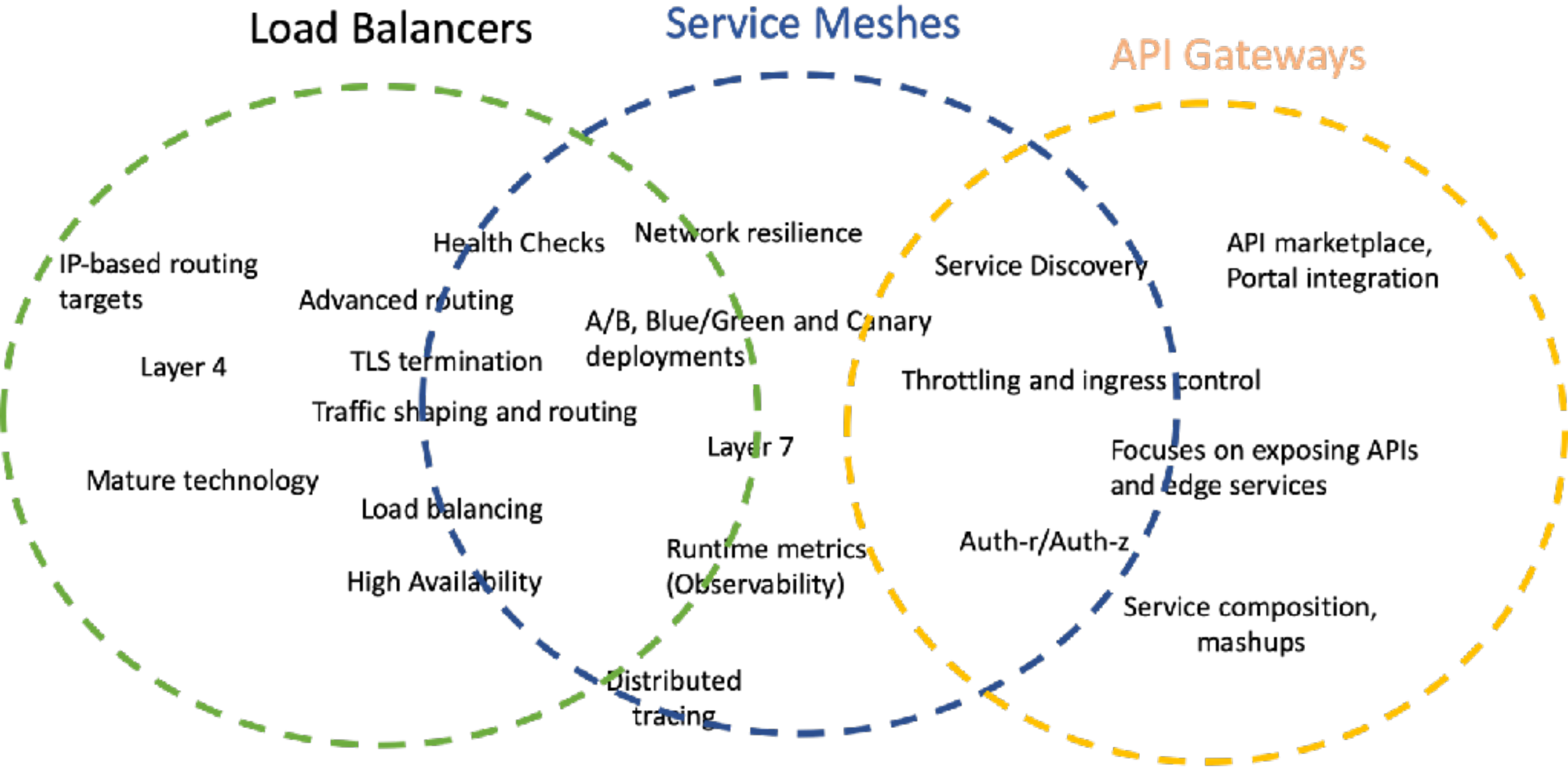


AWS API Gateway runs in it's own VPC and is completely managed so you cannot see any details about its infrastructure.

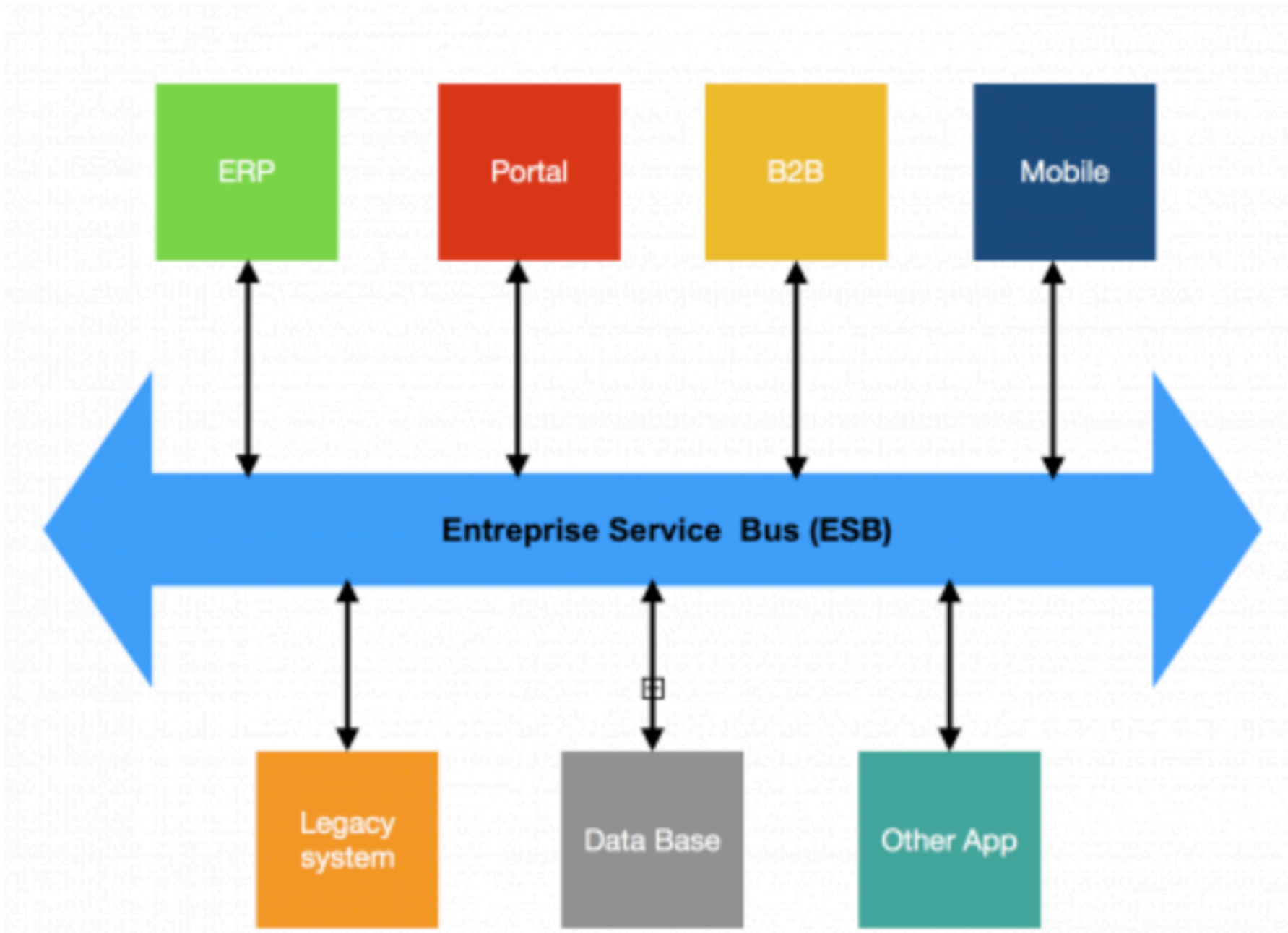
API Gateway: Abstraction, decoupling, edge routing / security

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

API Gateway vs ESB


ERP
Portal
B2B
Mobile
Entreprise Service Bus (ESB)
Legacy system
Data Base
Other App

API Gateway

Corporate network

Firewall

L3

Load Balancer

WAF

L7

Container Cluster

K 8s Ingress/Istio gateway

North south

Service Mesh (Istio)

East West

Container Deployment Platform
(Kubernetes)

Containers(Docker)

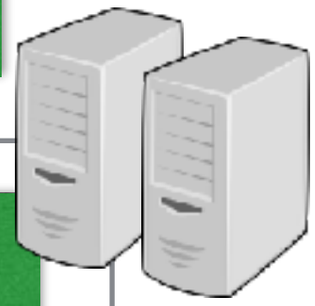
Microservice

http://172.17.0.22:80/date

North- South

API Gateway

LoadBalancer : L4 (tcp)



Cluster

Ingress or ISTIO Gateway : L7 (http)

Virtual Service

http://10.97.33.128:8080/date

East - west

Service

Side car

Microservice
POD

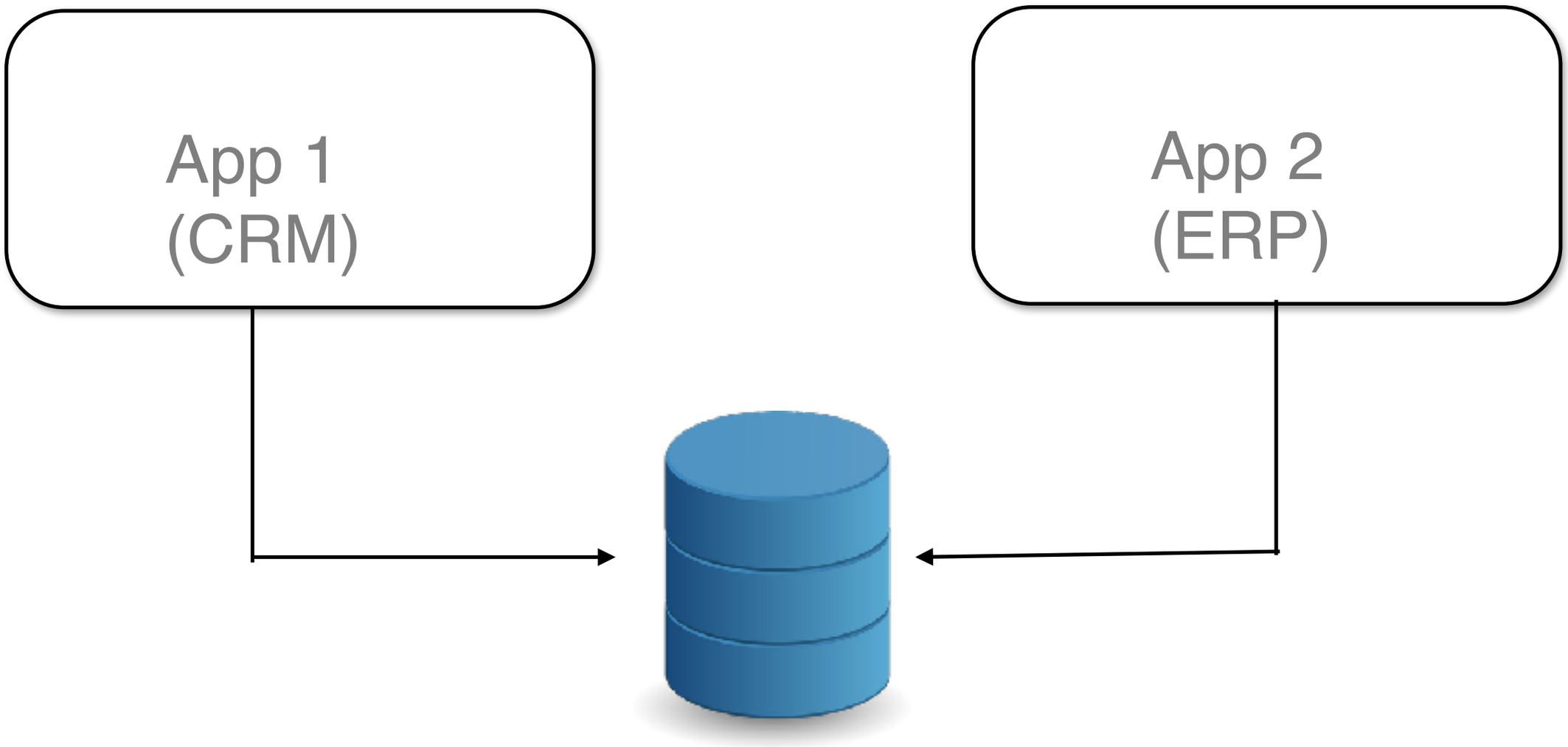
Sharing

Log Agg

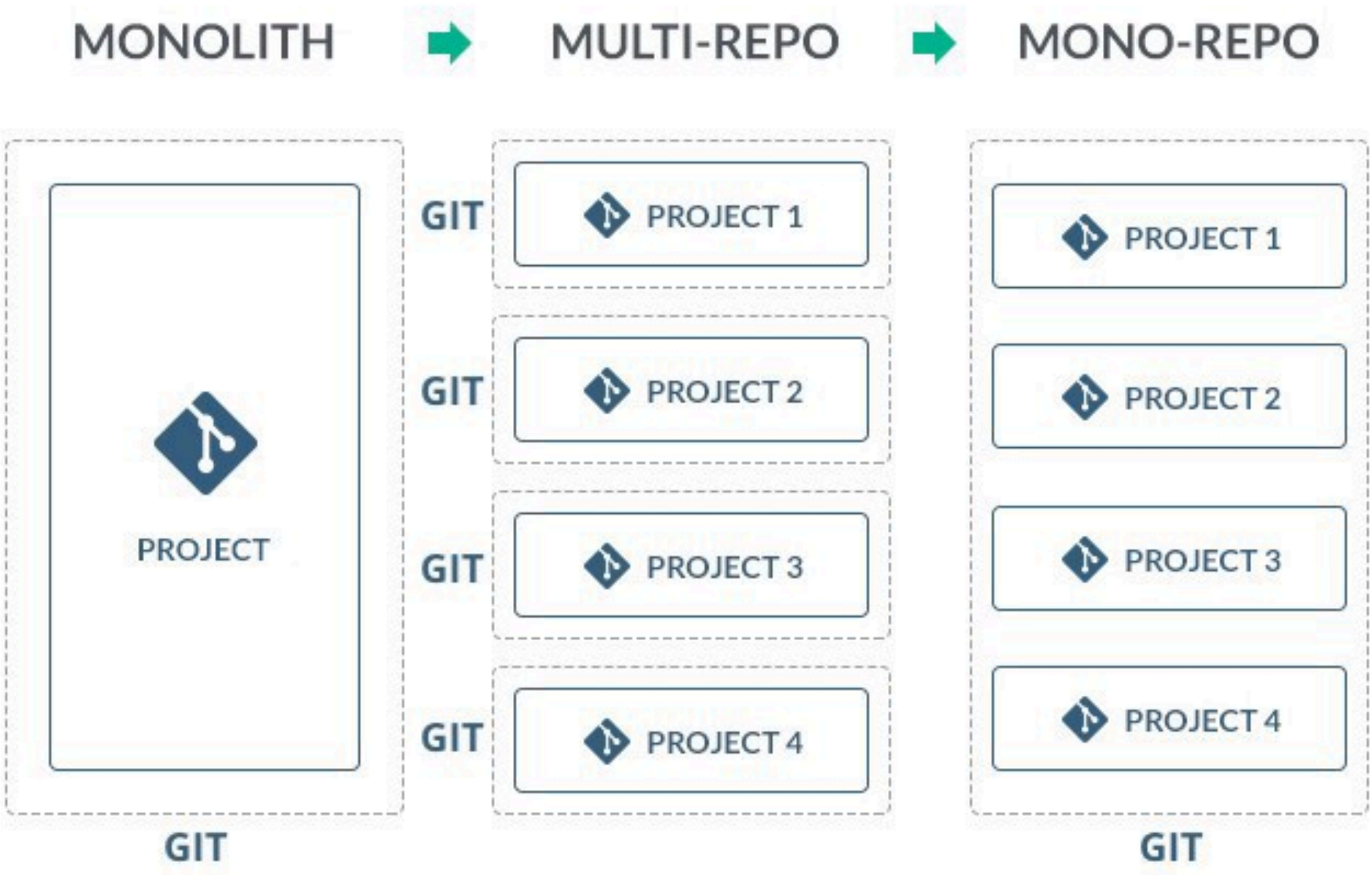
Tracability

Monitoring

1. Database



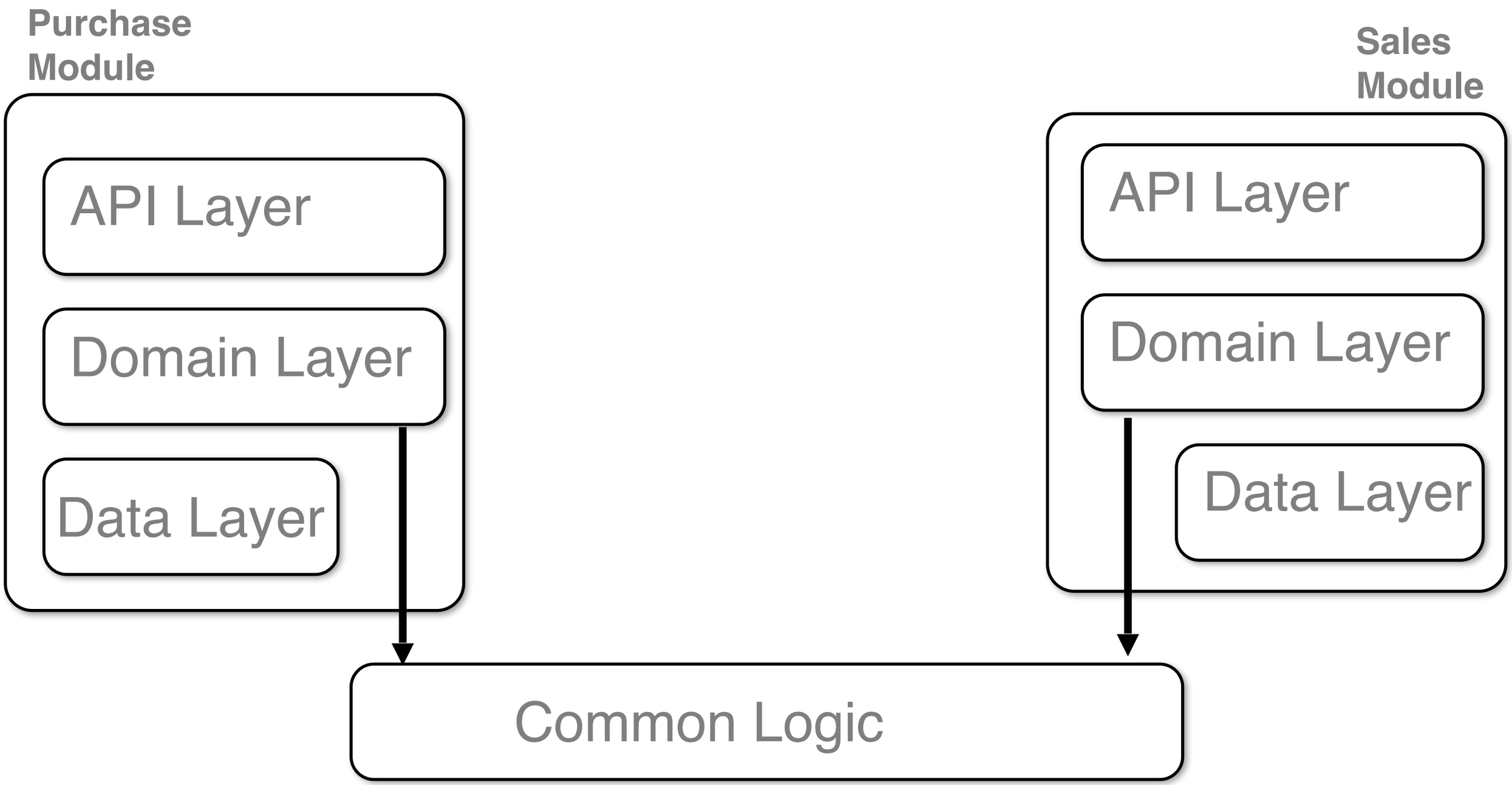
2. Source Control



2. Source Control

	Monorepo	Poly repos
Advantages	Code sharing Easier to standardize code and tooling Easier to refactor code Discoverability - single view of the code	Clear ownership per team Potentially fewer merge conflicts Helps to enforce decoupling of microservices
Challenges	Changes to shared code can affect multiple microservices Greater potential for merge conflicts Tooling must scale to a large code base Access control More complex deployment process	Harder to share code Harder to enforce coding standards Dependency management Diffuse code base, poor discoverability Lack of shared infrastructure

3. Common Logic

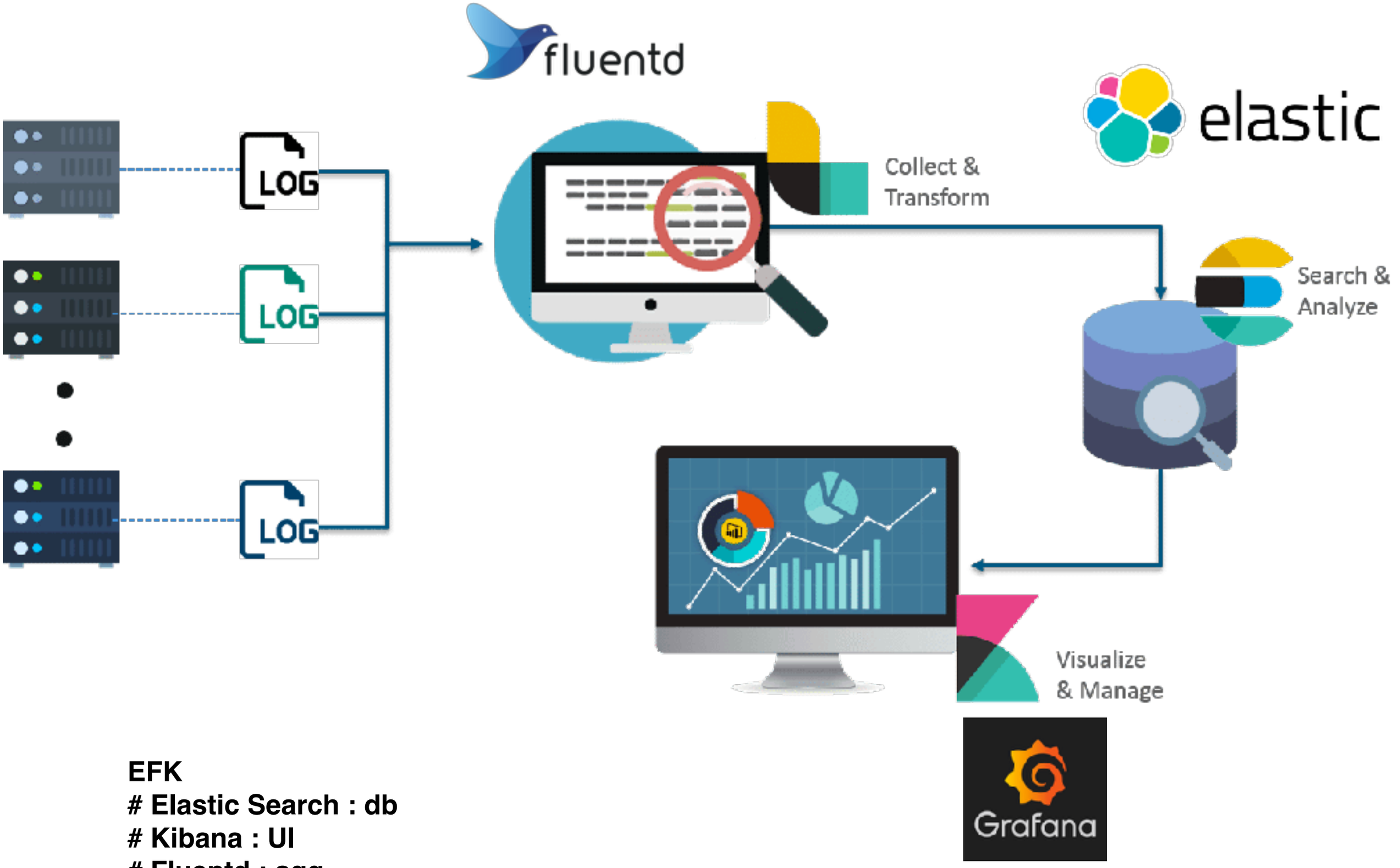


Pure fabrication # stateless, domain independent

Monitoring

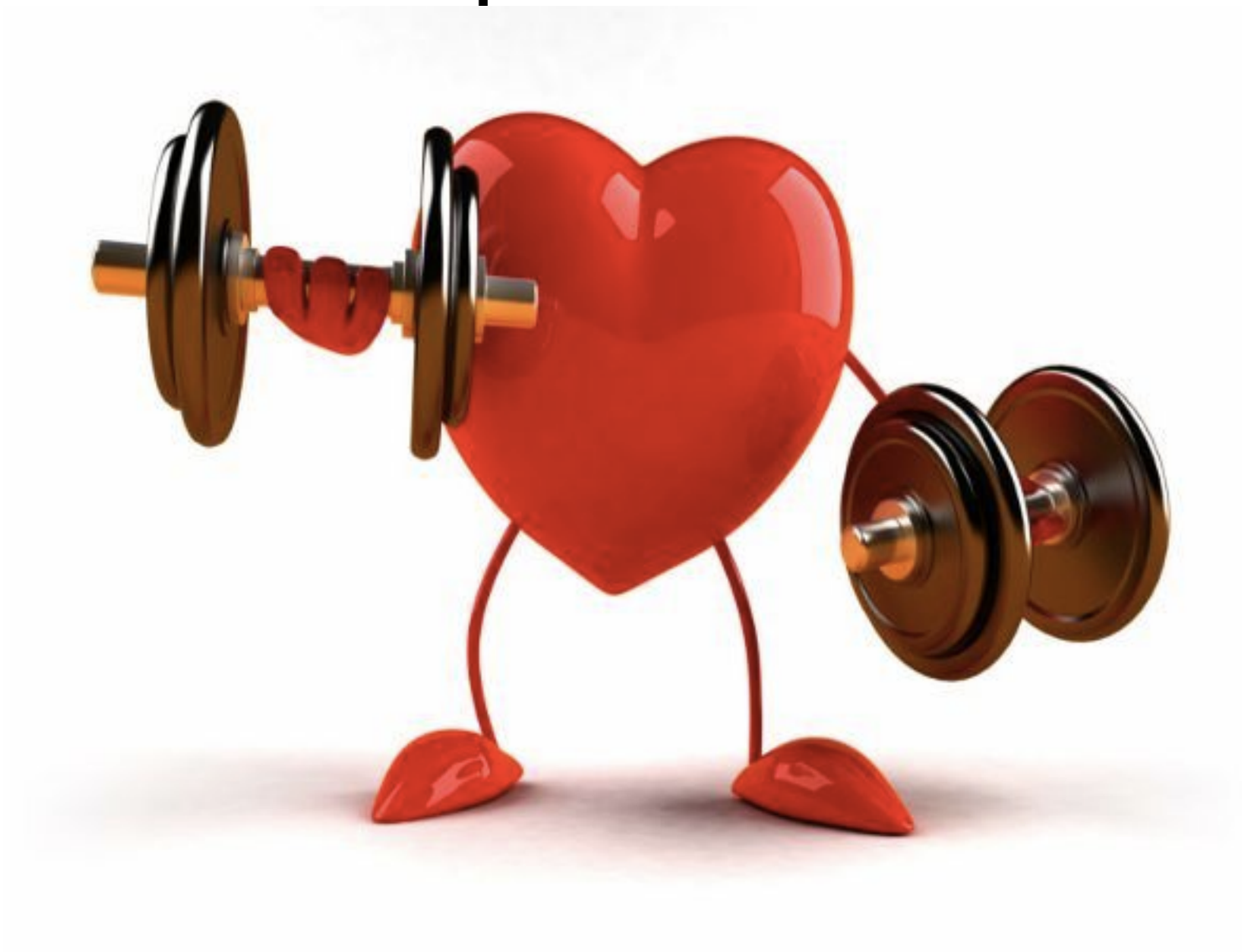
- Kubernetes
    - Kubernetes. Dashboard (metrics)
    - Weavescope (dependancy)
- ISTIO
    - Kiali (dependancy)
    - Grafana (metrics)
- End to End Tracebility
    - Jaeger
- Logs
    - EFK

Log Aggregation



EFK
 # Elastic Search : db
 # Kibana : UI
 # Fluentd : agg

Health Endpoint Monitoring

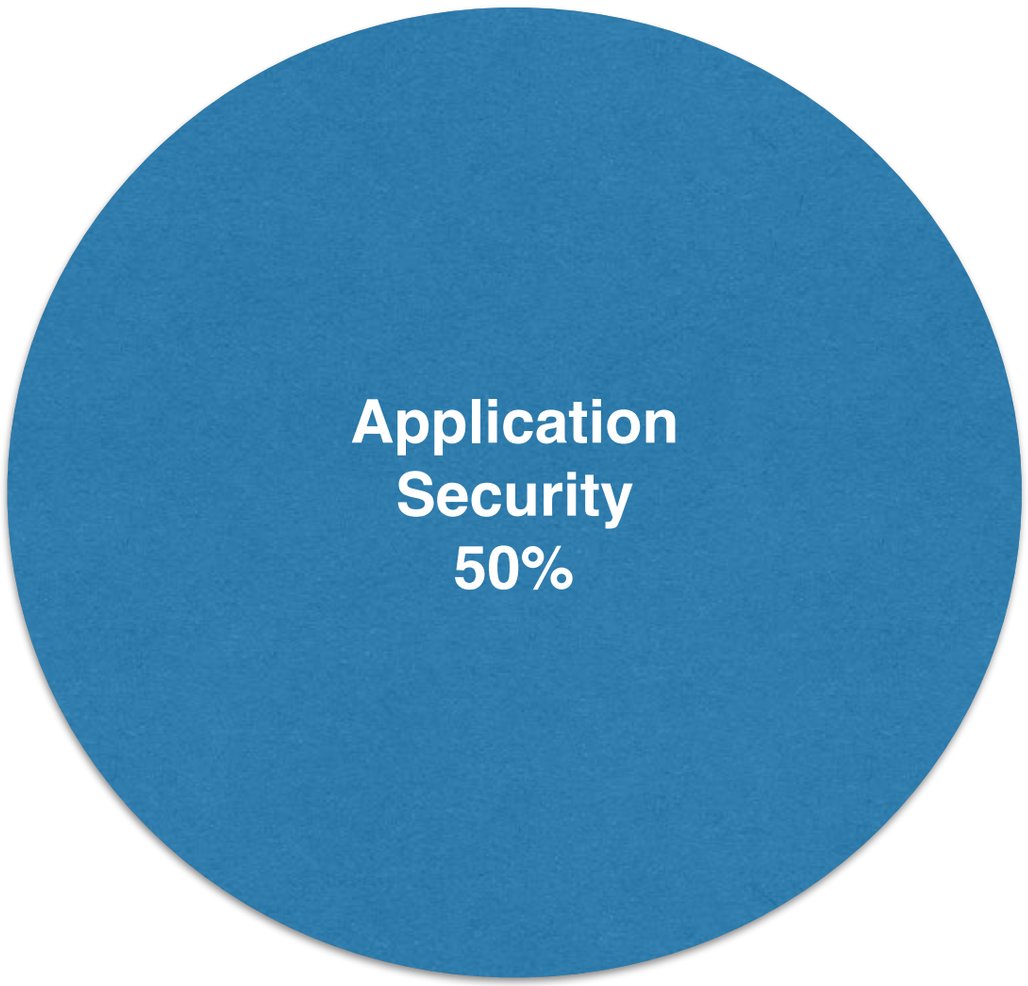


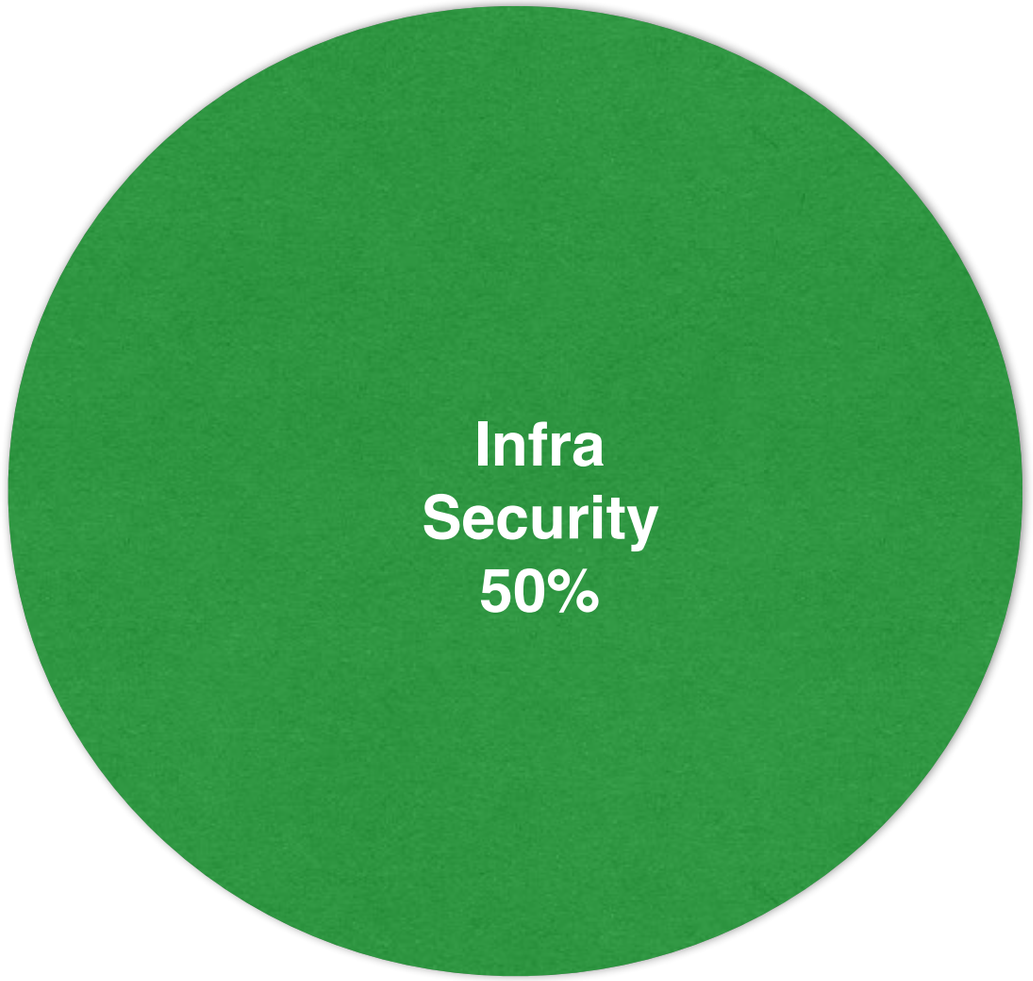
Storage

Compute

Messaging


SECURITY


Application
Security
50%


Infra
Security
50%

Authentication
 # First Defense

Authorization

Auditing - Last Defense

Data in Transit

Data in Rest

Input Validation

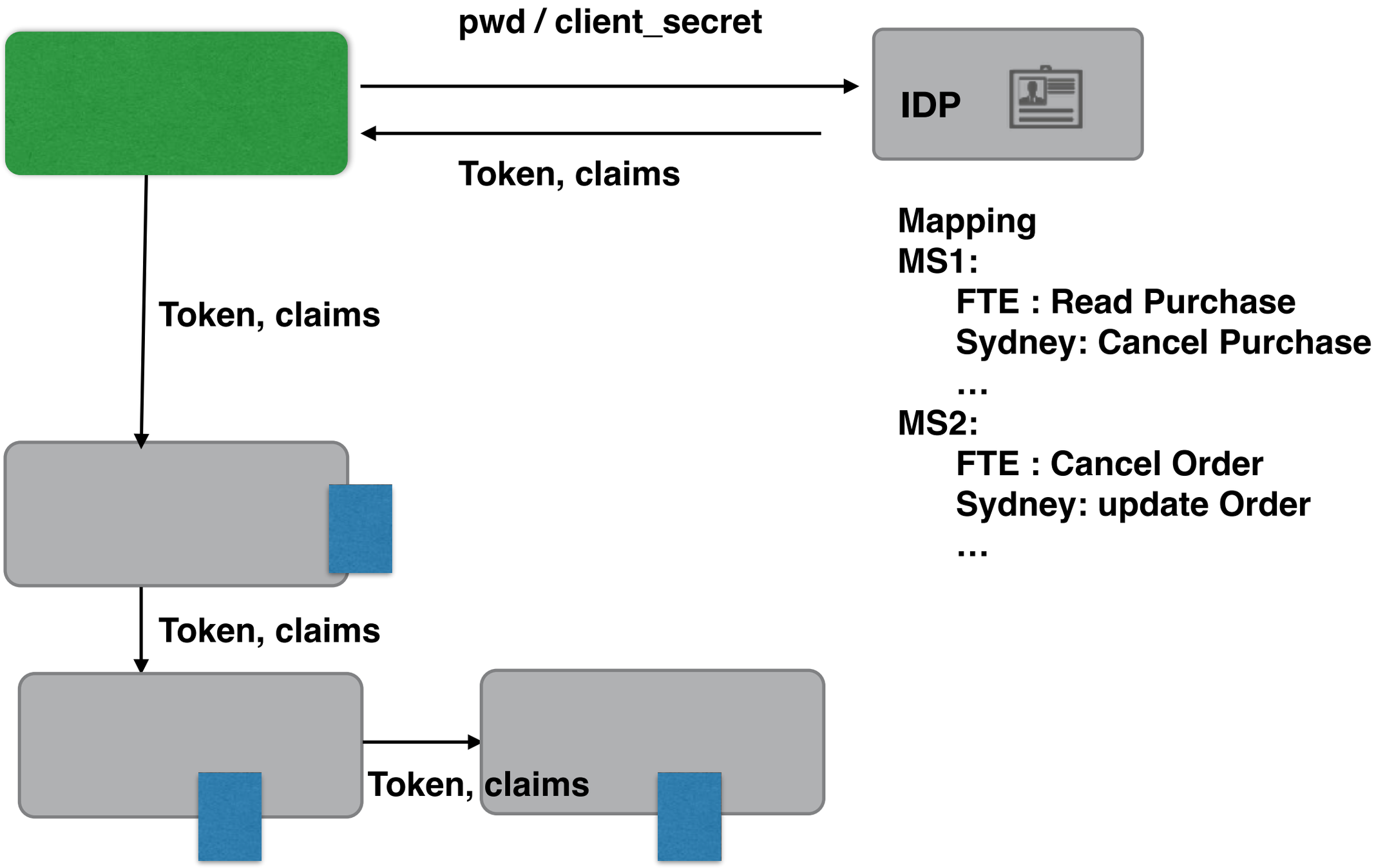
Exception Handling

Session Management

Secret Management

Throttling

L4 Firewall (TCP)	DOS, ..
L7 Firewall (http) - WAF	XSS, sql injection, one click, ….
Vnet	
Subnet	
Route Table	
VPN	



Mapping
MS1:
FTE : Read Purchase
Sydney: Cancel Purchase
…
MS2:
FTE : Cancel Order
Sydney: update Order
…

STRIDE

- S : Spoofing <— Authentication
- T : Tampering <— Data in Transit, Data in Rest
- R : Repudation <— Audit
- I : Information Disclosure <— Authorization
- D : Deniel of Service <— input validation
- E : Elevation of Privileage

Microservice App - 50%

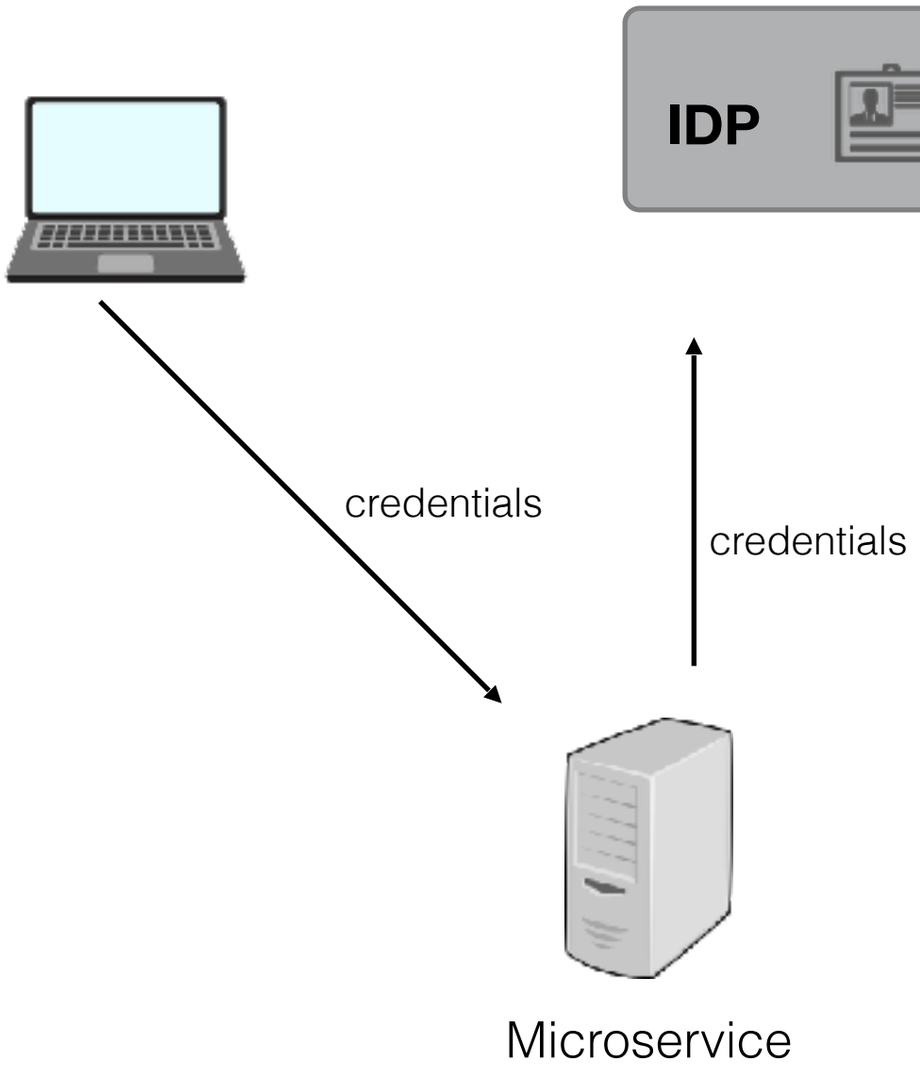
- Authentication - OAuth2 + OpenID Connect + JWT token
- Authorization - RBAC, Claims
- Audit - Event Sourcing
- Data In Transit - https, was
- Data In Rest - Hash, Sym, Asym, cloud storage (*)
- Input Validation - WAF, library (*)
- Exception Handling - fwk
- Session Management- fwk
- Key Management - key vault

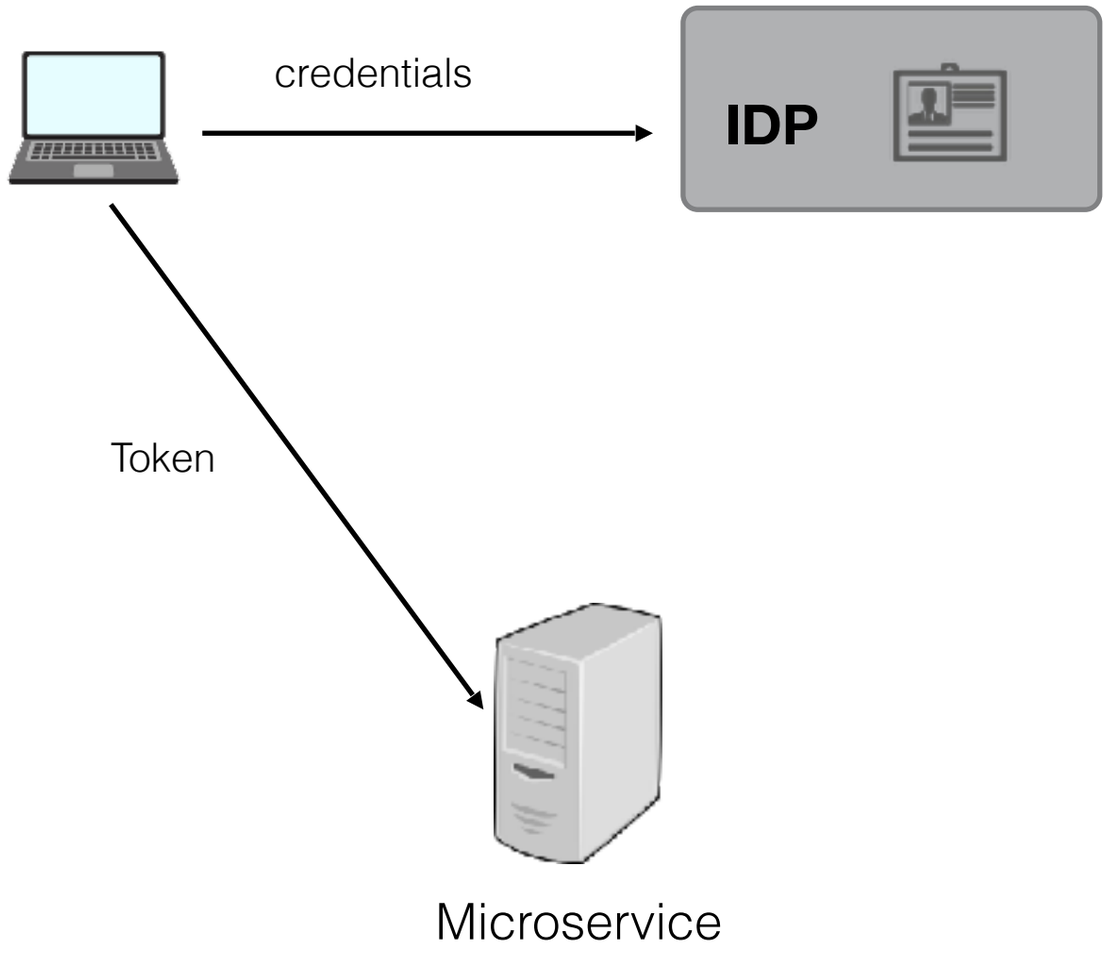
DevSecops - 50%

- L4 Firewall (TCP)
- L7 Firewall (HTTP) - WAF
- vnet, subnet
- Route table
- Scanning tools
- Monitoring & Alerts

- By what you know (Knowledge)
    - Custom (name &pwd), **Oauth2, Openid connect**, Secret question,
- By what you have <— stolen
    - OTP, RSA, Cert, …
- By what you are <— stolen
    - Bio (voice, retina, face, finger print, dna, …)

2 legged vs 3 legged



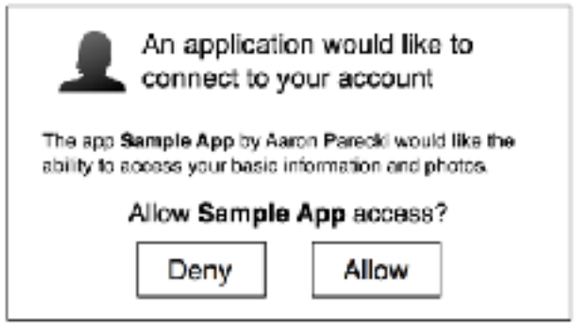


OAuth2: Authorization Code

3

https://oauth2server.com/auth?
response_type=code&
client_id=CLIENT_ID&
redirect_uri=REDIRECT_URI
&scope=photos

4 login page



IDP

Client

5 credentials

6 url?code=AUTH_CODE

App

8

POST https://api.oauth2server.com/token
grant_type=authorization_code&
code=AUTH_CODE_HERE&
redirect_uri=REDIRECT_URI&
client_id=CLIENT_ID&
client_secret=CLIENT_SECRET

9

```
{
   "access_token":"RsT5OjbzRn430zqMLgV3Ia"
}
```

7

url?code=AUTH_CODE

2

redirect

request

1

Server App

Microservice

Implicit

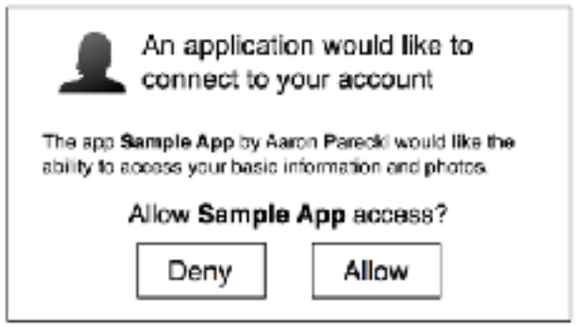
1

https://oauth2server.com/auth?
response_type=token&
client_id=CLIENT_ID&
redirect_uri=REDIRECT_URI&
scope=photos

Client

App

2 login page



3 credentials

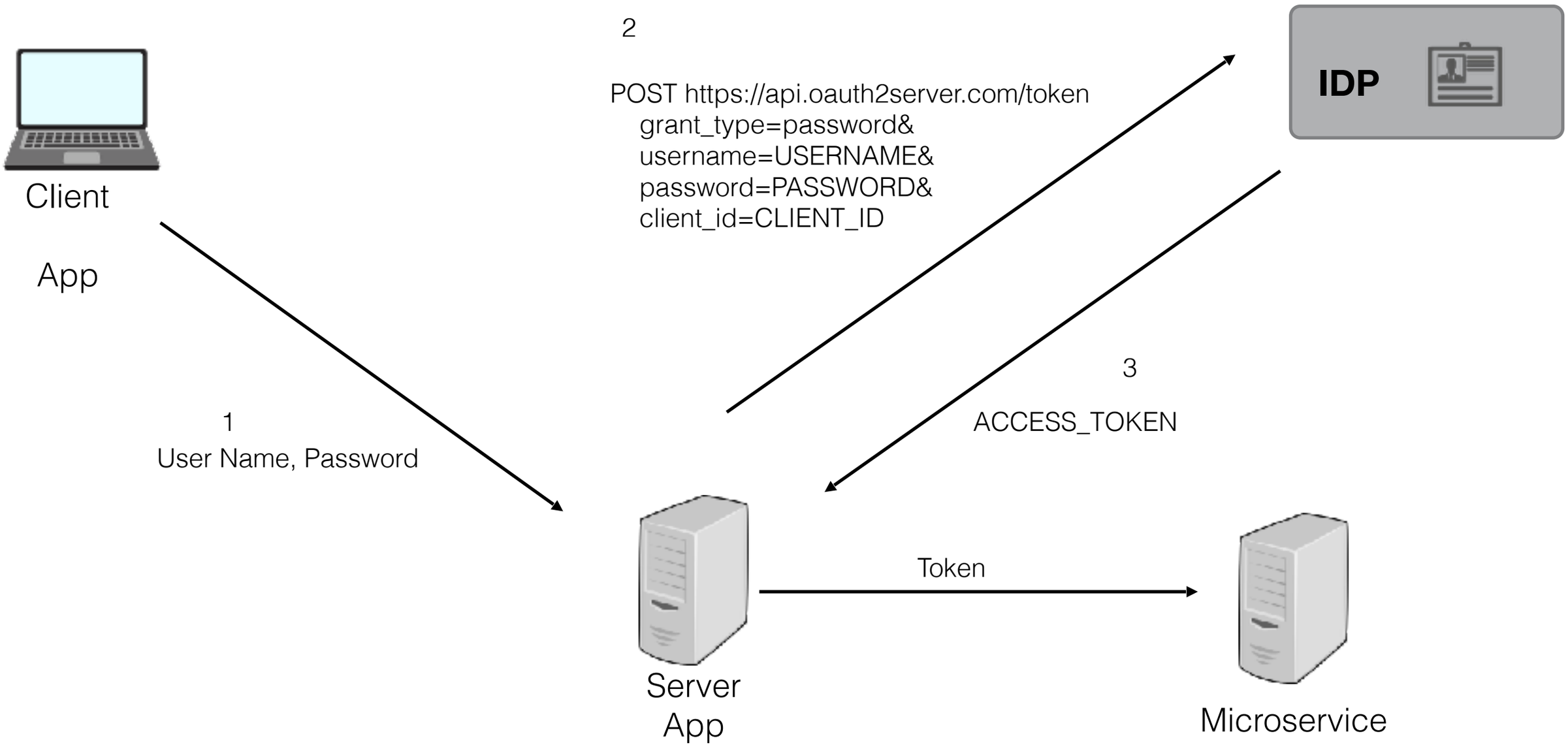
4 https://oauth2client.com/cb#token=ACCESS_TOKEN

5

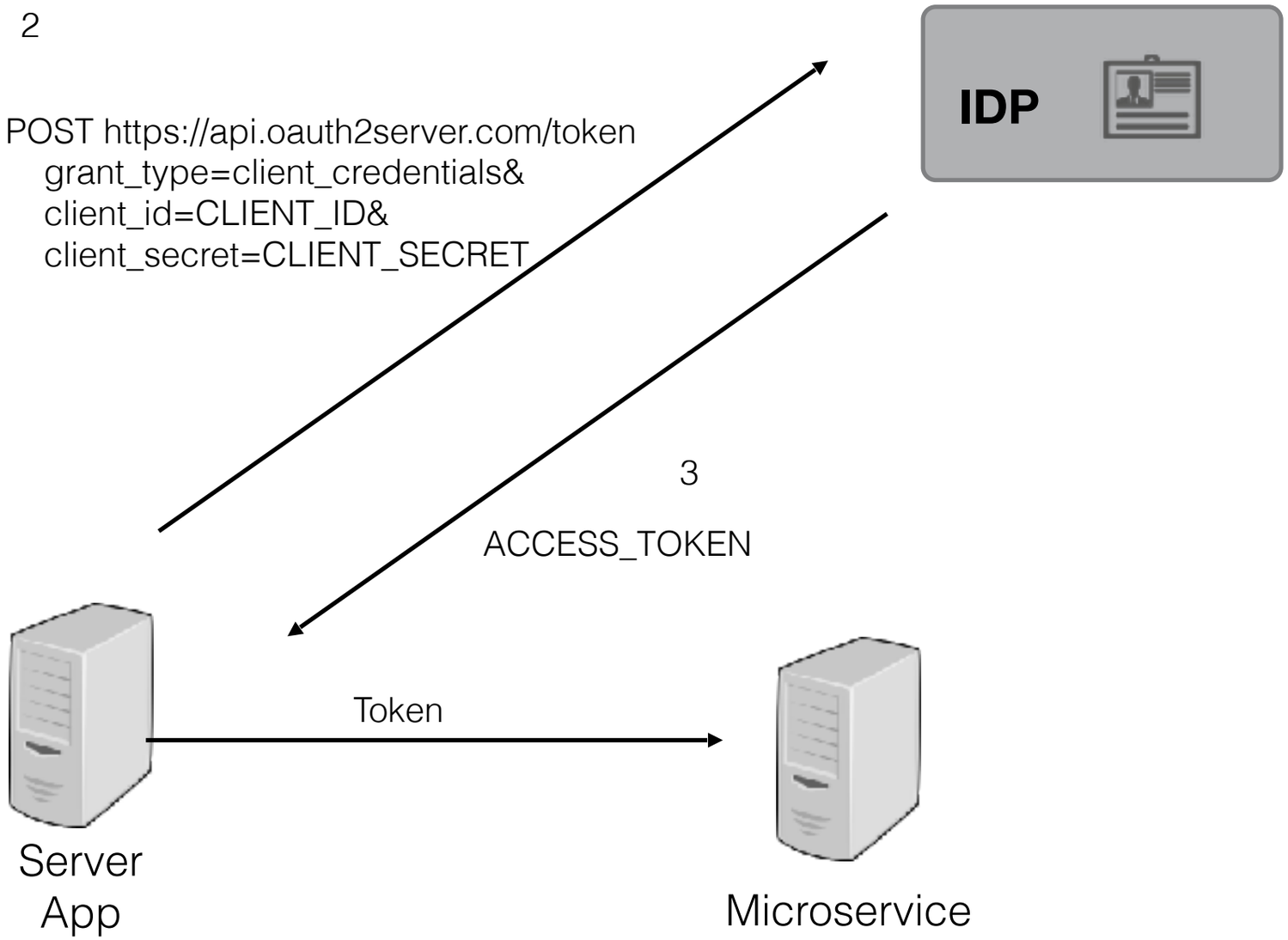
url?token

Microservice

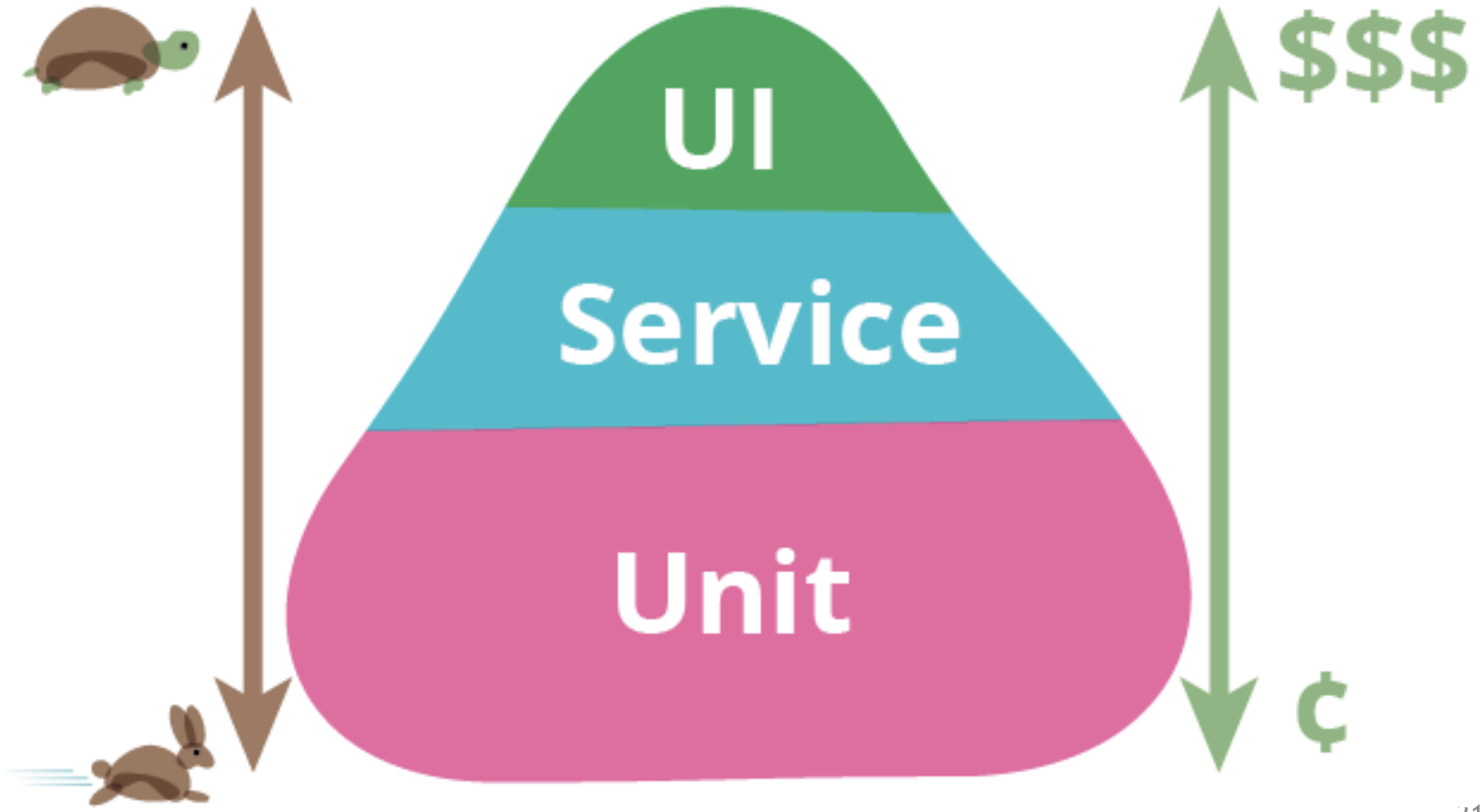
Password


2
POST https://api.oauth2server.com/token
grant_type=password&
username=USERNAME&
password=PASSWORD&
client_id=CLIENT_ID
IDP
Client
App
1
User Name, Password
3
ACCESS_TOKEN
Token
Server
App
Microservice

Clients Credential



Test Pyramid



Number of Tests

13 min

8 min

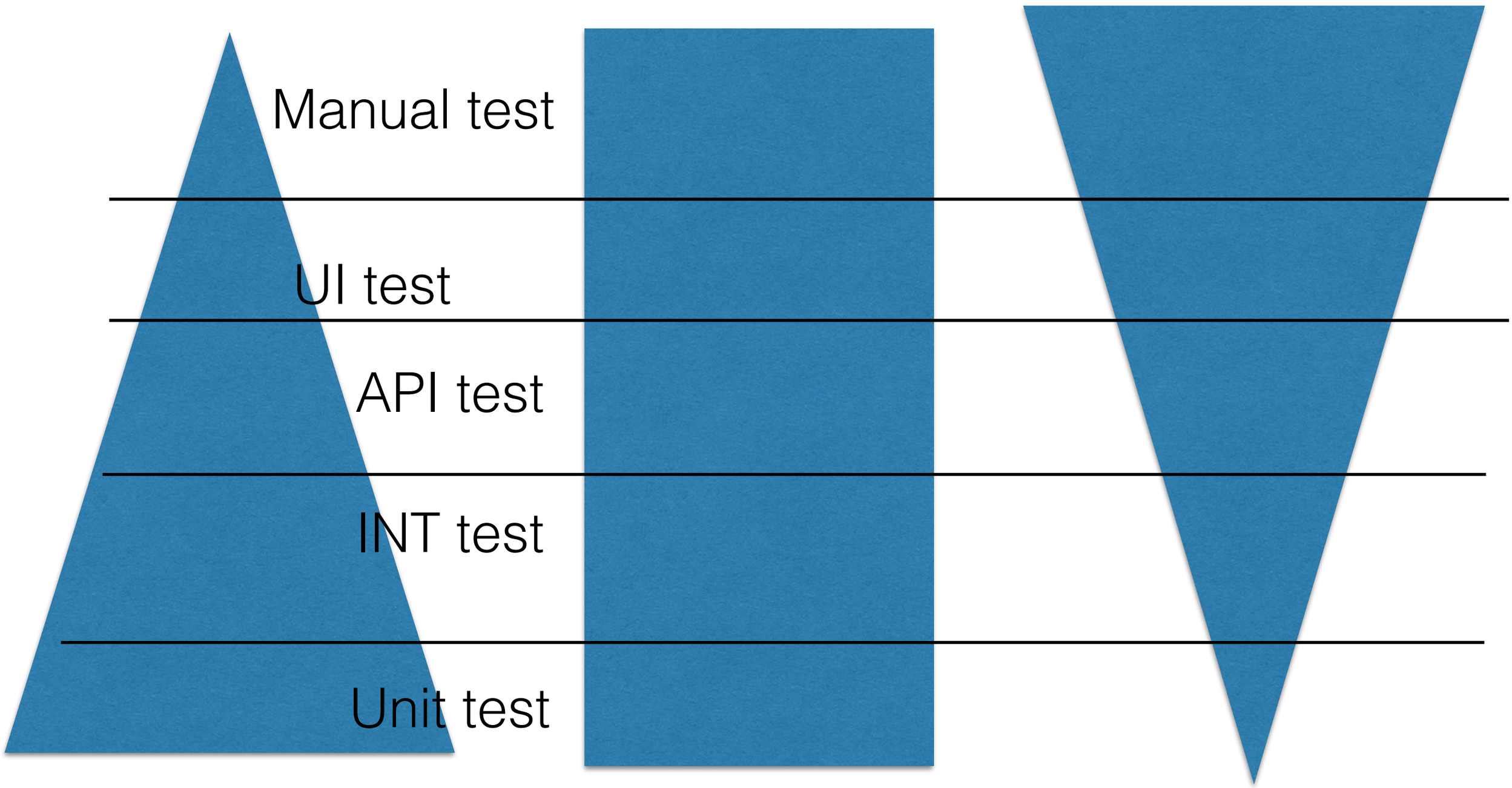
1 second

1 min

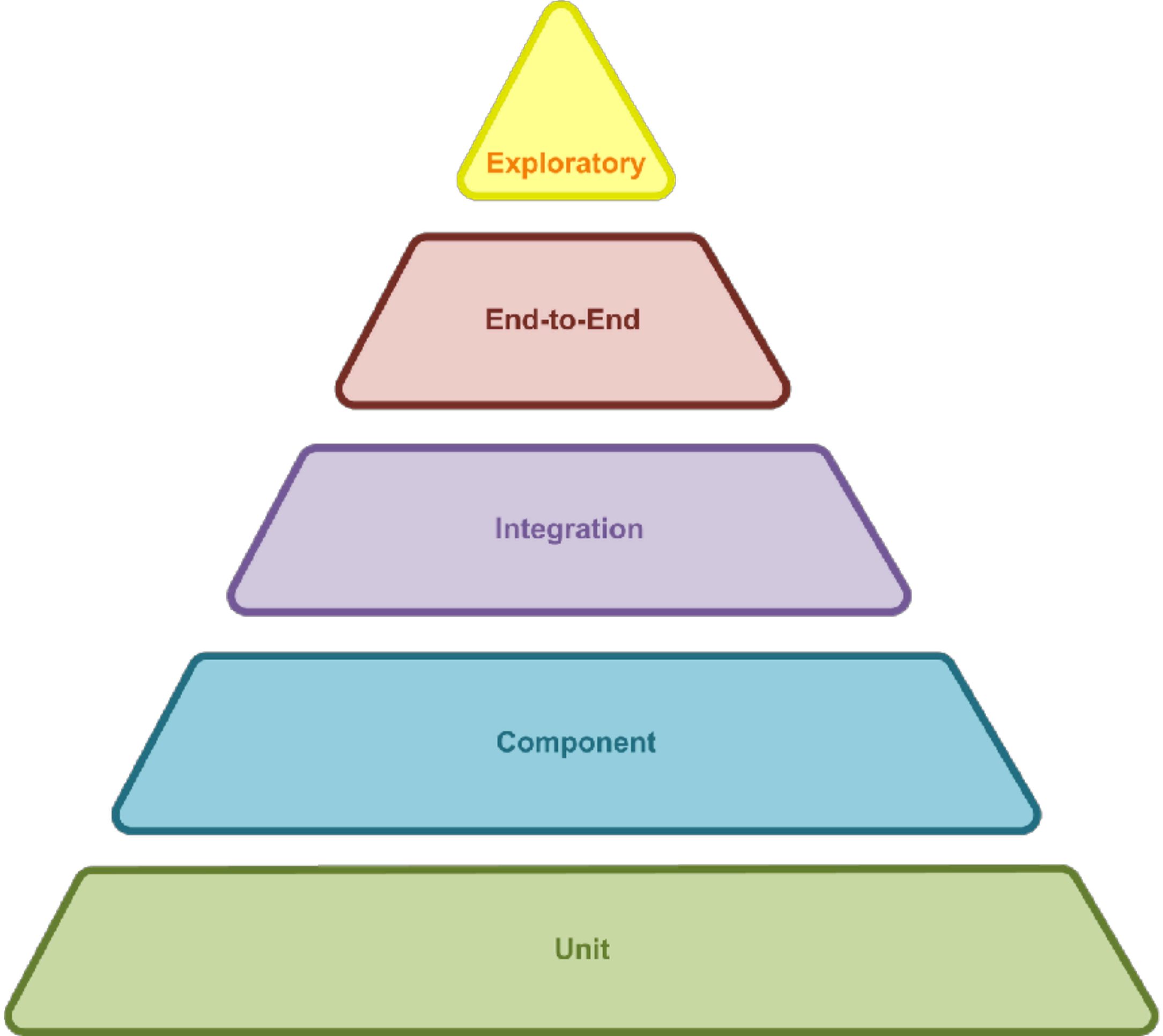
Test Pyramid

www.fabiopereira.me

ThoughtWorks


Manual test
UI test
API test
INT test
Unit test

Cyclomatic Complexity ->


Exploratory
End-to-End
Integration
Component
Unit

App's

Web Server

API Layer

Domain Layer

Data Layer

Web Server

UI Layer

Domain Layer

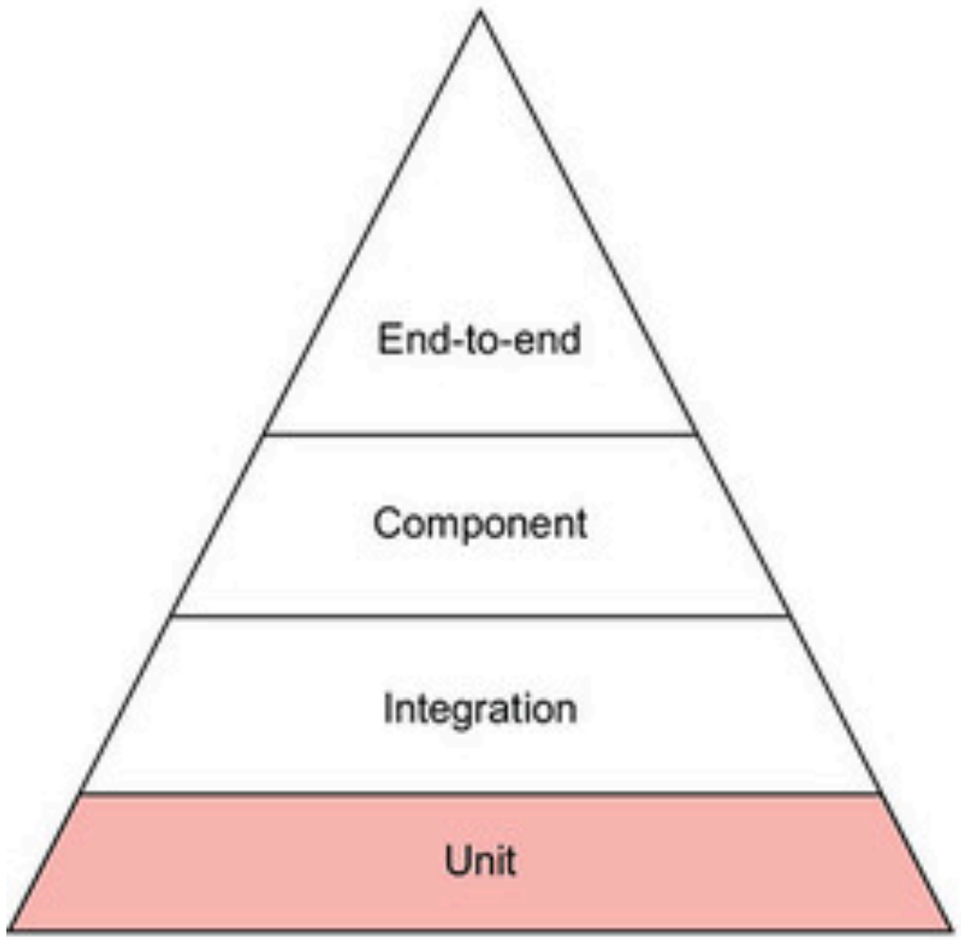
Data Layer

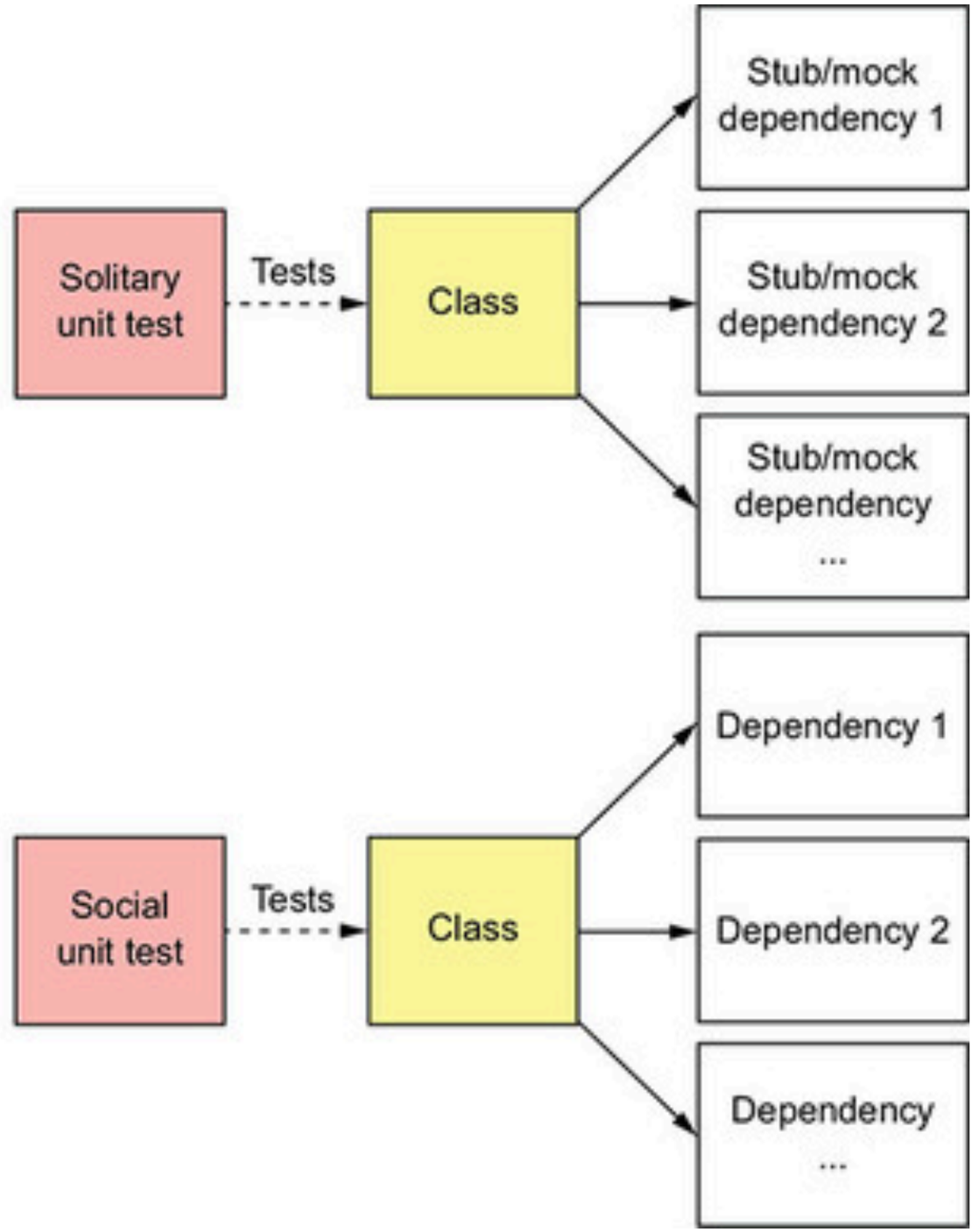
Desktop

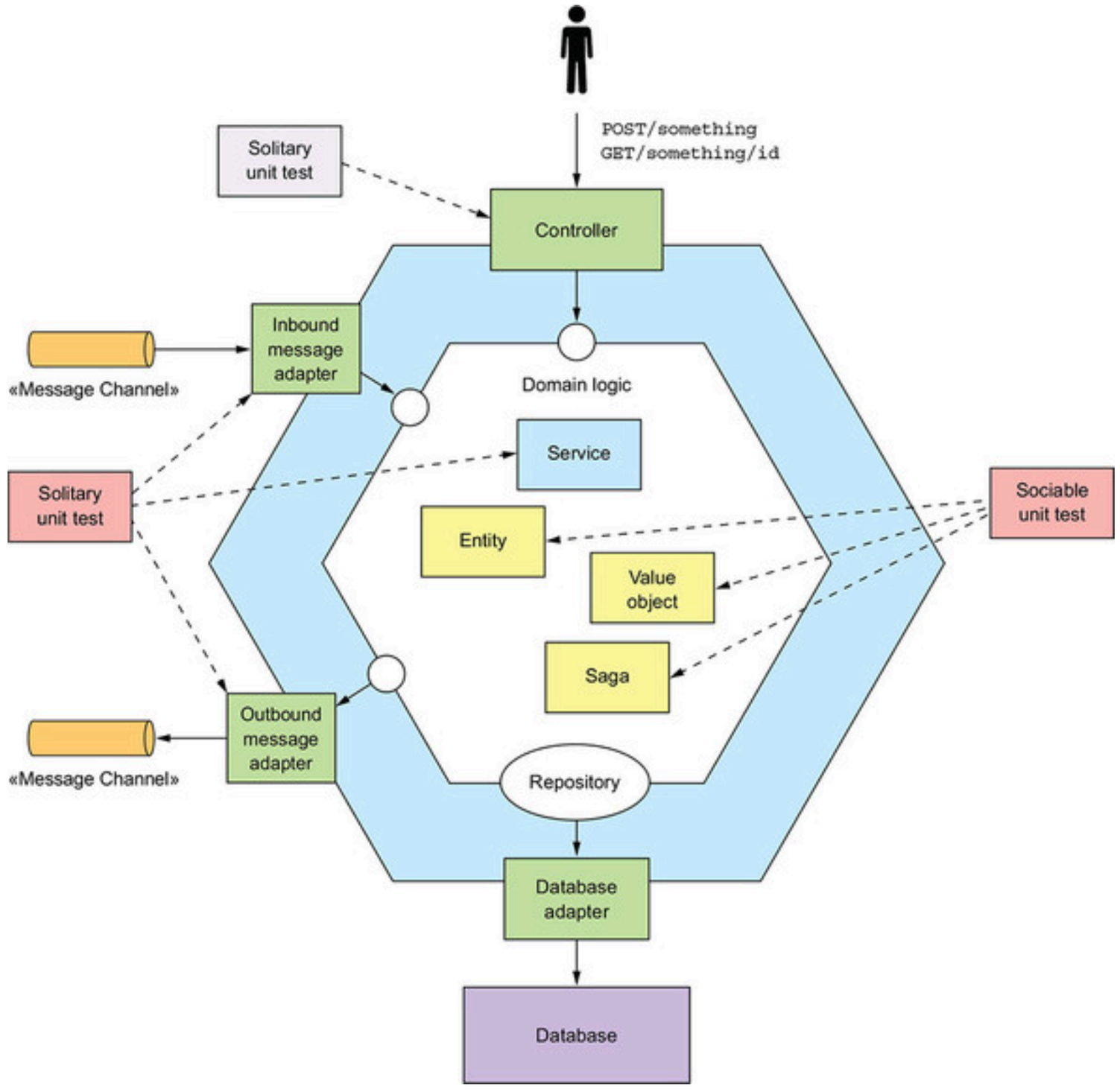
UI Layer

Domain Layer

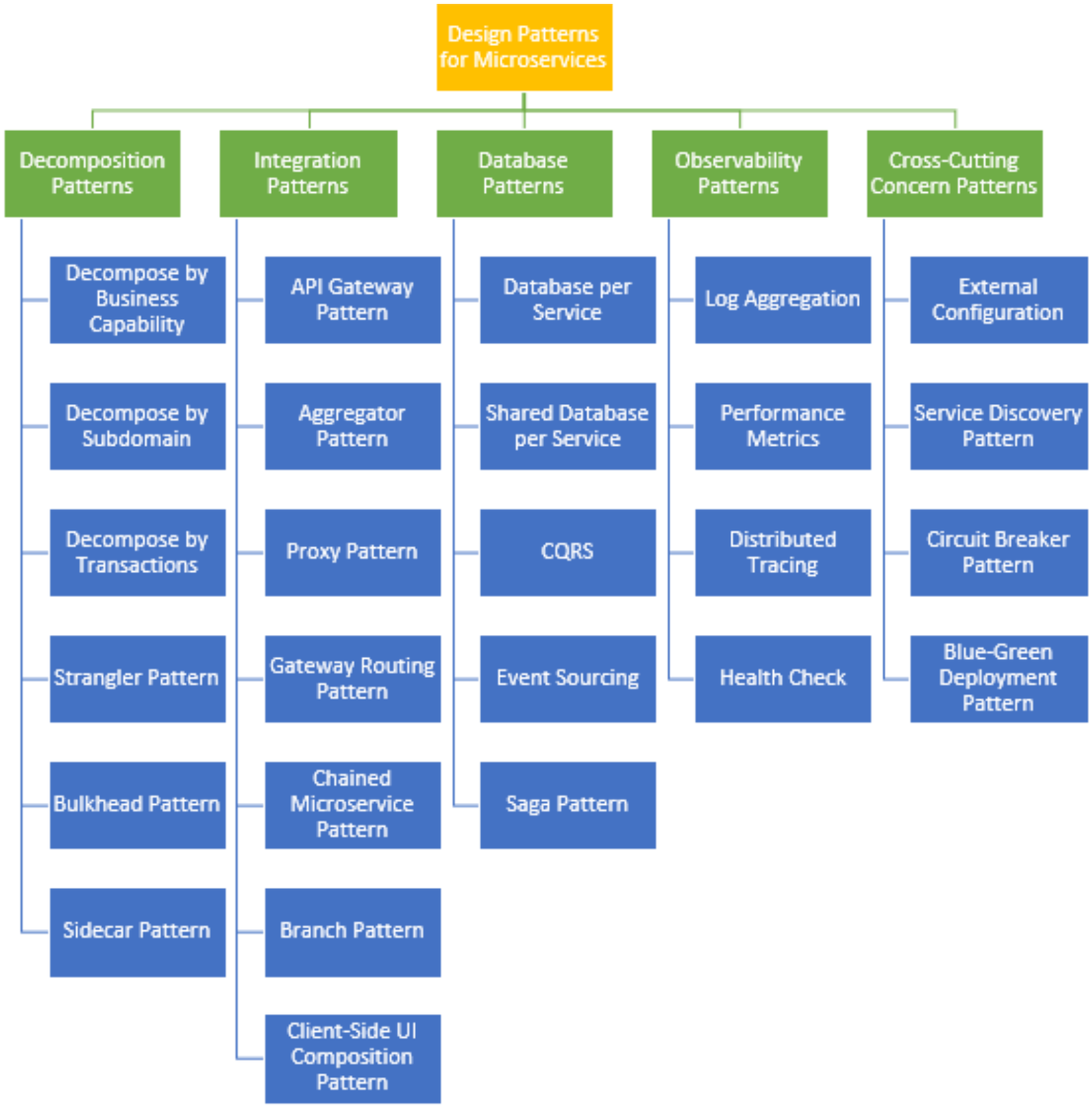
Data Layer


End-to-end
Component
Integration
Unit

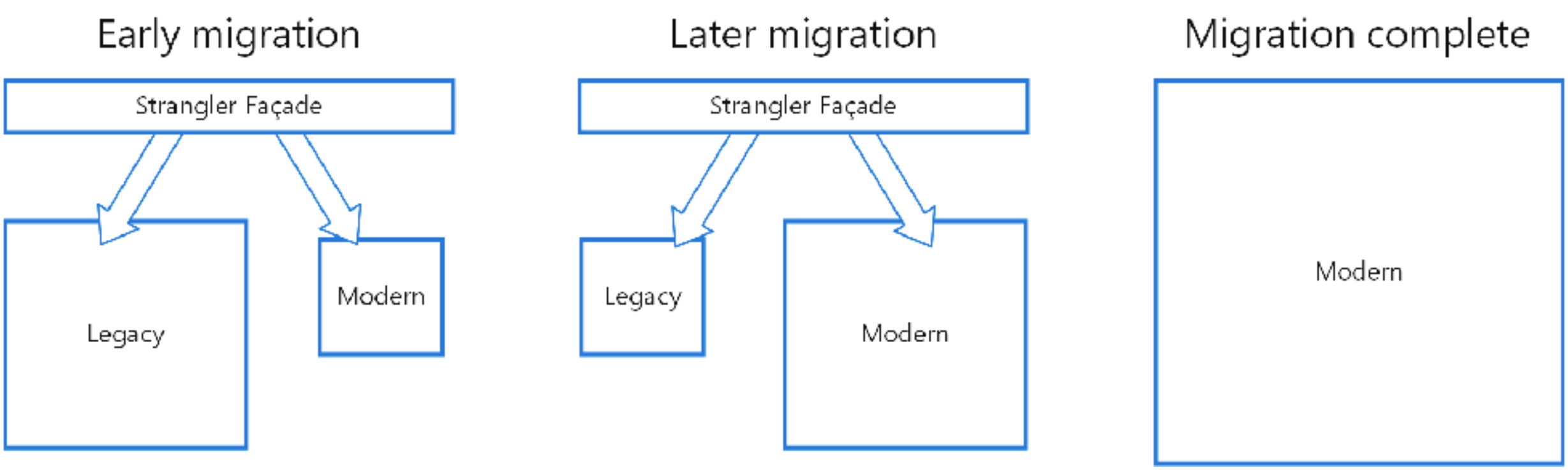

Solitary unit test
Tests
Class
Stub/mock dependency 1
Stub/mock dependency 2
Stub/mock dependency ...
Social unit test
Tests
Class
Dependency 1
Dependency 2
Dependency ...


POST/something
GET/something/id
Solitary unit test
Controller
Inbound message adapter
«Message Channel»
Domain logic
Service
Solitary unit test
Sociable unit test
Entity
Value object
Saga
Outbound message adapter
«Message Channel»
Repository
Database adapter
Database

Patterns

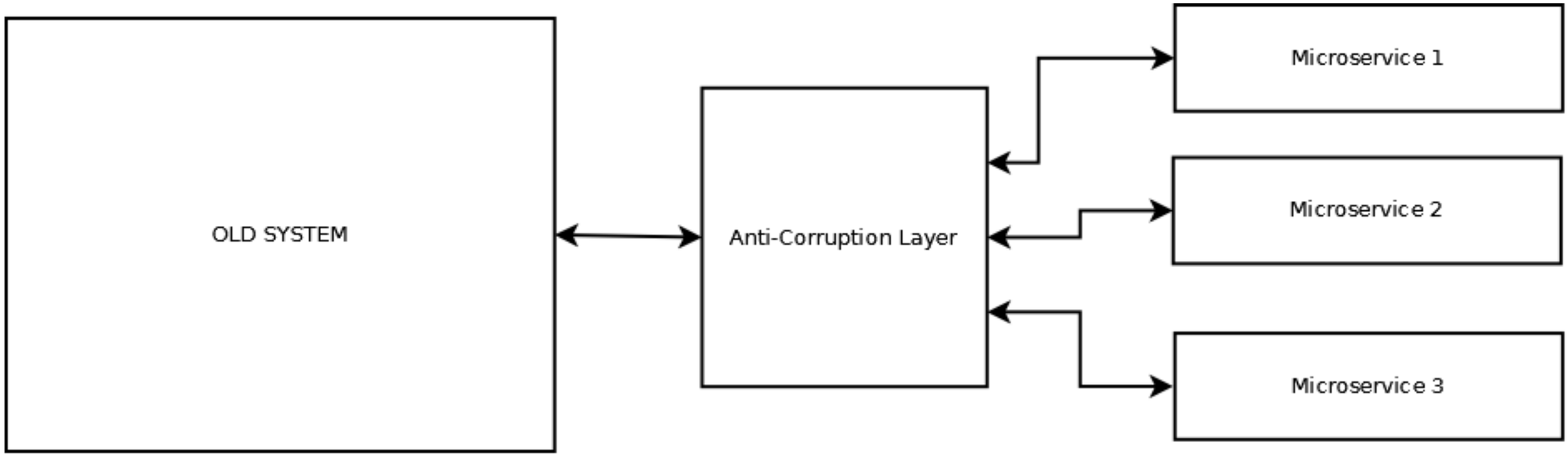

Design Patterns for Microservices
Decomposition Patterns
Integration Patterns
Database Patterns
Observability Patterns
Cross-Cutting Concern Patterns
Decompose by Business Capability
Decompose by Subdomain
Decompose by Transactions
Strangler Pattern
Bulkhead Pattern
Sidecar Pattern
API Gateway Pattern
Aggregator Pattern
Proxy Pattern
Gateway Routing Pattern
Chained Microservice Pattern
Branch Pattern
Client-Side UI Composition Pattern
Database per Service
Shared Database per Service
CQRS
Event Sourcing
Saga Pattern
Log Aggregation
Performance Metrics
Distributed Tracing
Health Check
External Configuration
Service Discovery Pattern
Circuit Breaker Pattern
Blue-Green Deployment Pattern

Strangler pattern



- Create a façade that intercepts requests going to the backend legacy system. Over time, as features are migrated to the new system, the legacy system is eventually "strangled" and is no longer necessary.

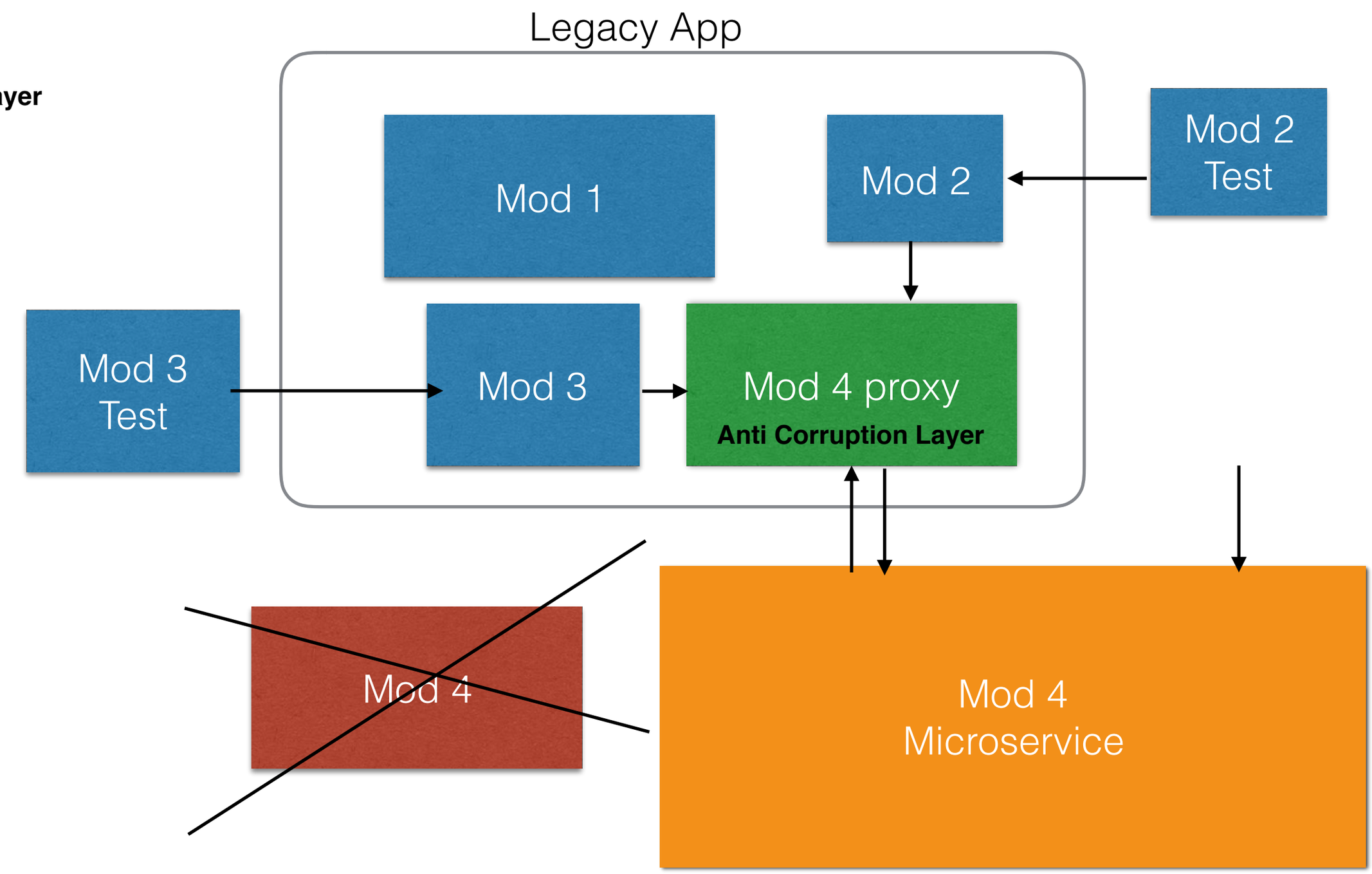
Anti-Corruption Layer

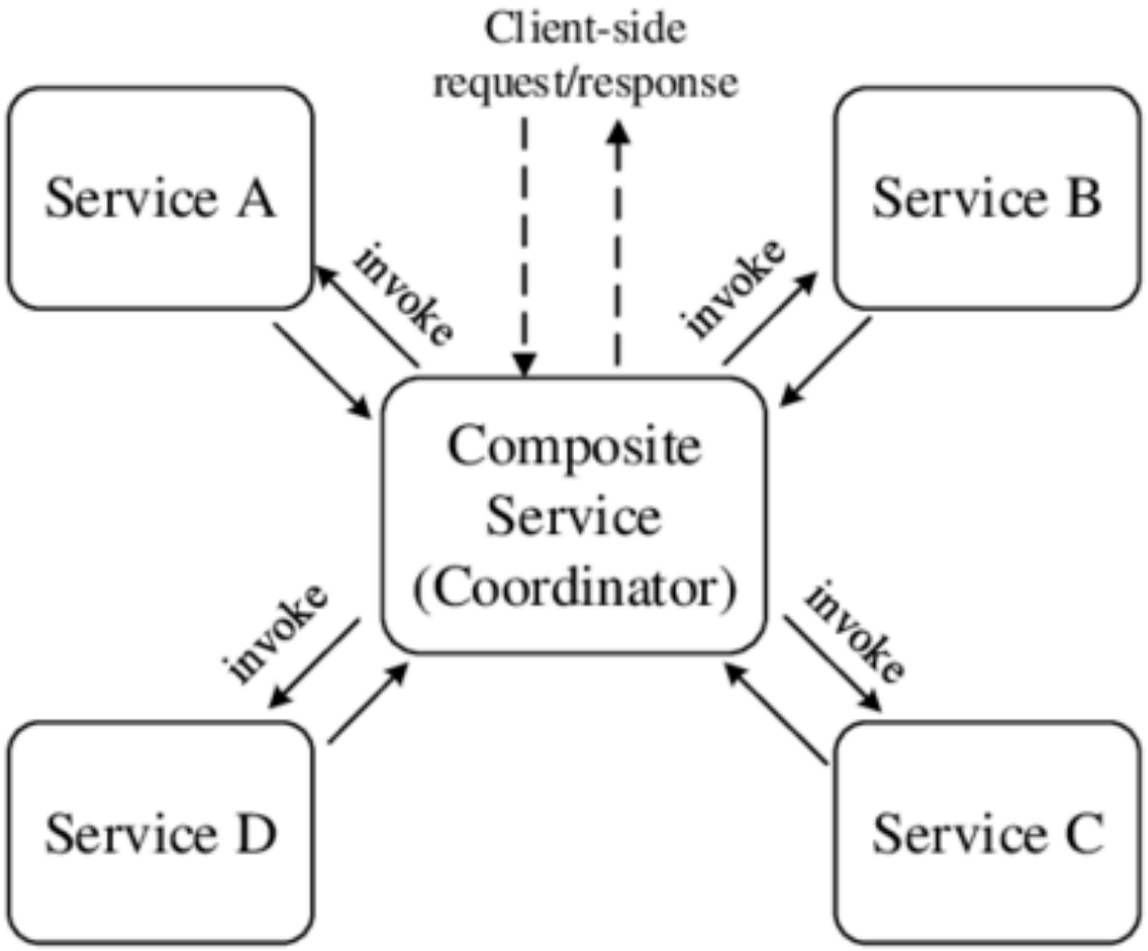


- When you create a new microservice, you notice that some business or technical assumptions or flow should be remodeled. instead of a Big Rewrite ACL maps one domain onto another so that services that use second domain do not have to be "corrupted" by concepts from the first.

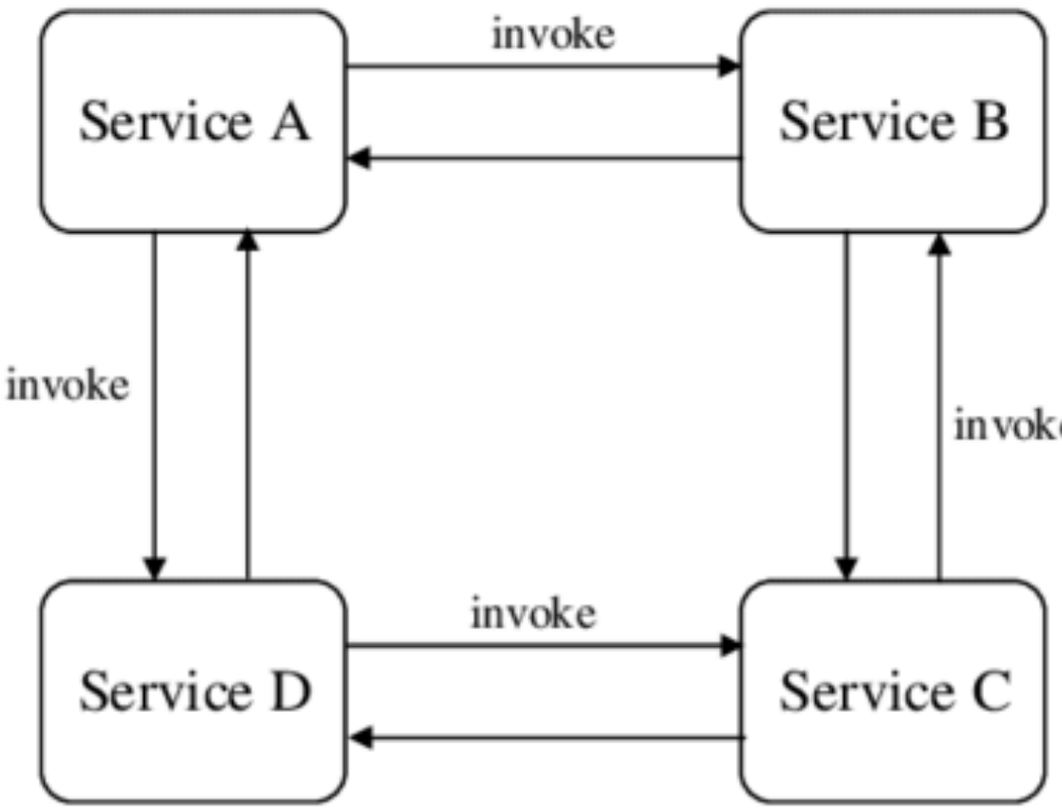
- Such modifications may result in changes to the events that a bounded context publishes.

 # unit test
 # Strangler Pattern
 # Anti Corruption Layer





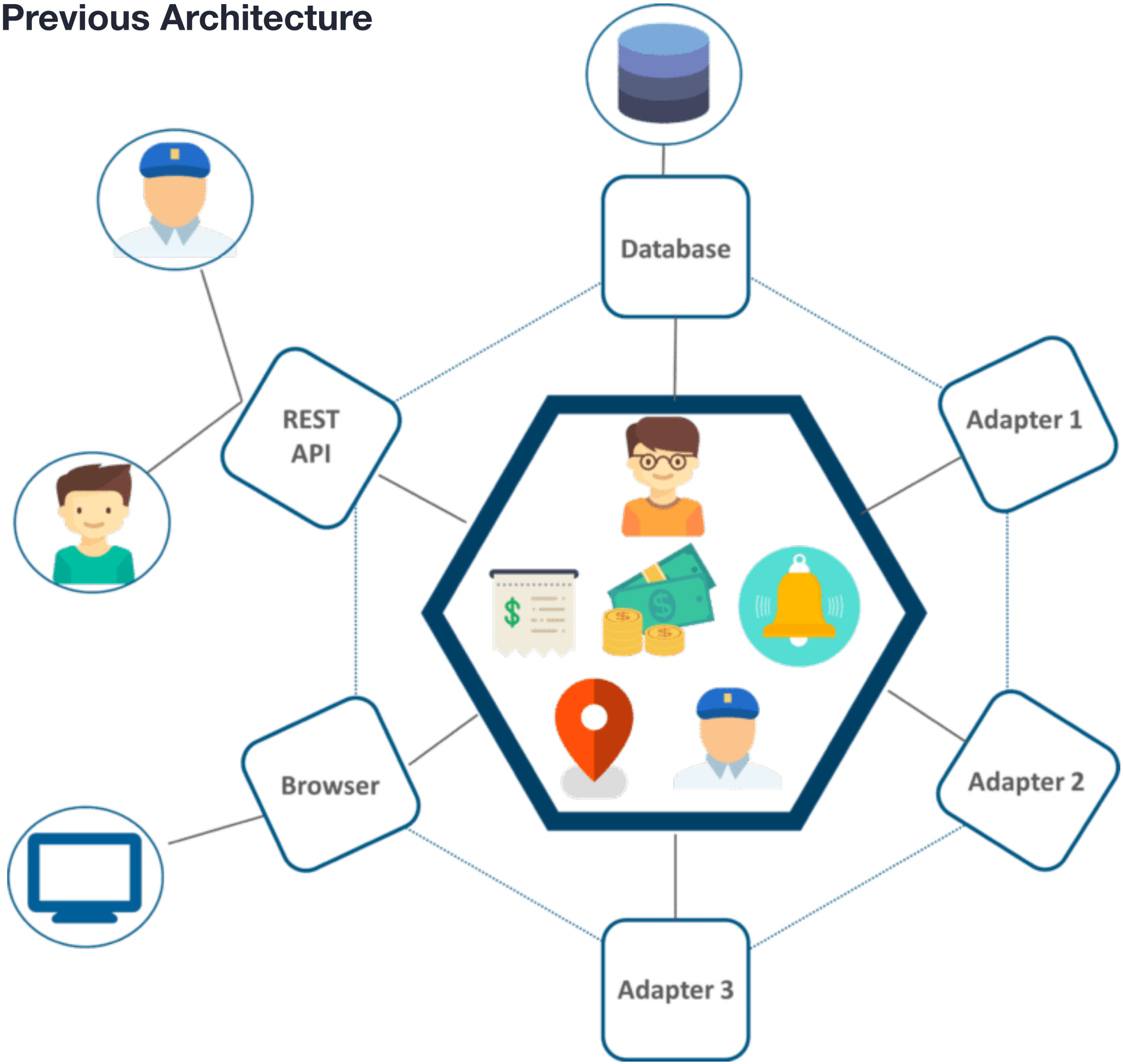
(a) Web Service Orchestration

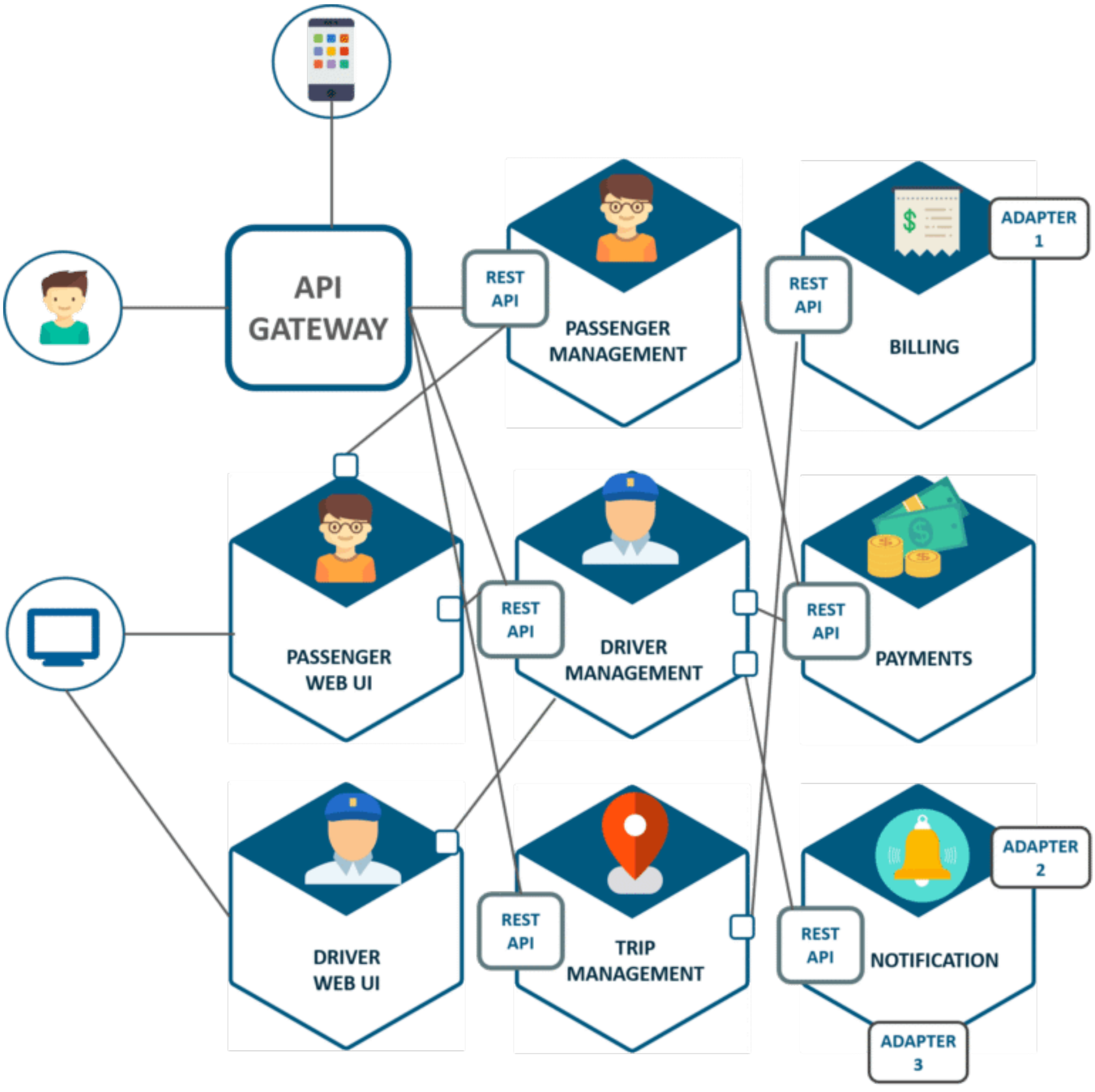


(b) Web Service Choreography

Case Study

Uber's Previous Architecture




API
GATEWAY
REST
API
PASSENGER
MANAGEMENT
REST
API
BILLING
ADAPTER
1
PASSENGER
WEB UI
REST
API
DRIVER
MANAGEMENT
REST
API
PAYMENTS
DRIVER
WEB UI
REST
API
TRIP
MANAGEMENT
REST
API
NOTIFICATION
ADAPTER
2
ADAPTER
3

Netflix turning towards microservices starts in 2009,

Netflix needed 2 years to split their monolith into microservices, and in 2011 announced end of redesigning their structure and organizing it using microservice architecture.

the Netflix application leverages 500+ microservices and API Gateways that handles over 2 billion API edge requests daily.

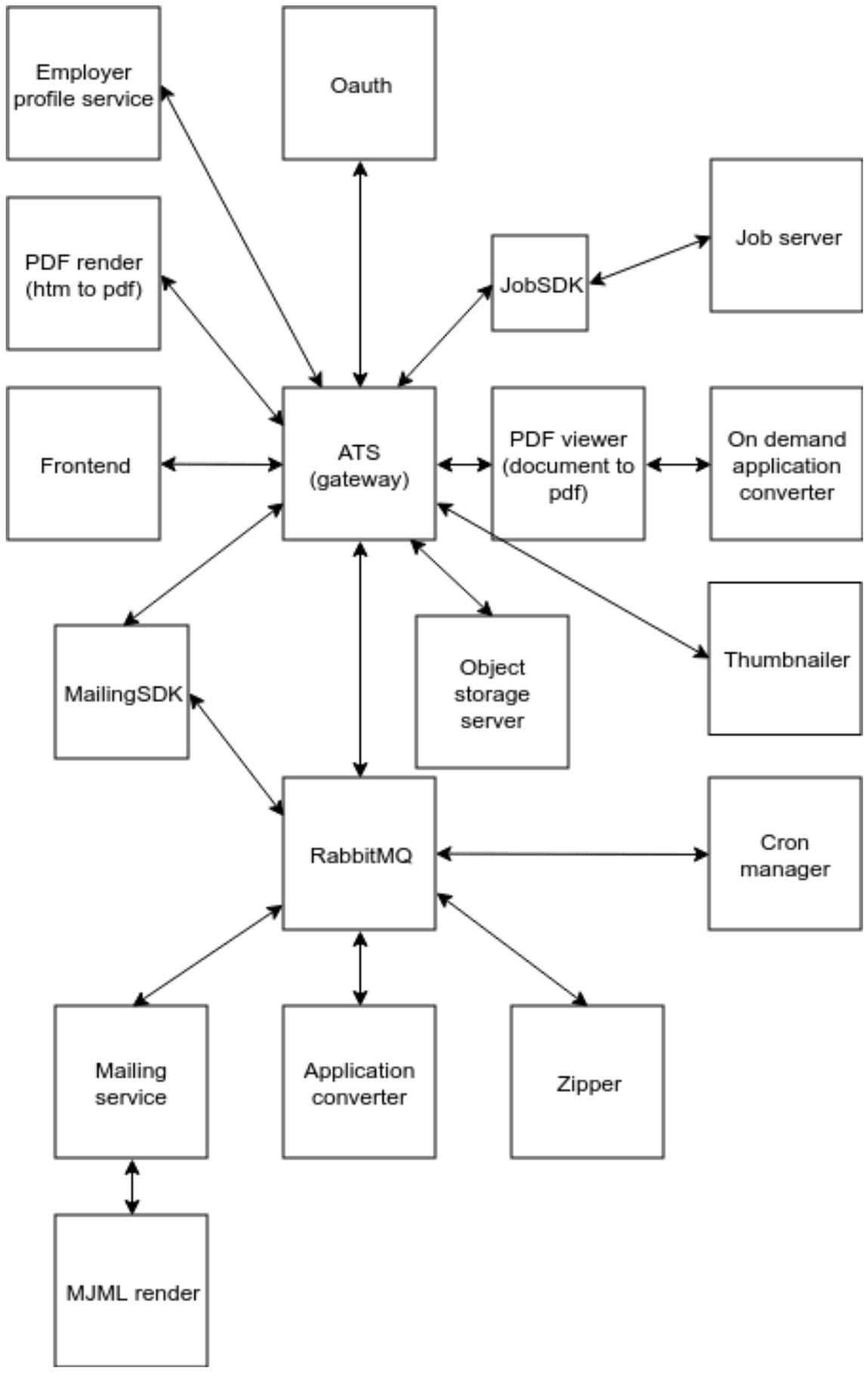
Gilt.com

Gilt reconstructed their monolithic architecture, which was based on Ruby on Rails, to a microservice architecture in 2011. based it on Scala, Docker and AWS technologies and initially created 156 services.

Application Tracking System (ATS)

1. After adding a new feature, it goes into the manual testing phase. Once it's greenlit, you deploy it to production — but what often happens is you miss a testing scenario, and you end up with lots of bugs in production.
2. This problem is tightly connected to the previous one: If you have an older codebase, its foundation starts to slowly break down, so adding new code on top of it ends up relying on other code. With that kind of codebase, sometimes you end up breaking code that should be totally irrelevant to what you're currently writing.
3. When you have 11 people working on the same codebase day in, day out, without any safeties in place whatsoever, problems are bound to arise.

- One Year
- ~50,000 lines of code
- ~2,600 commits
- a team of 3 developers
- consisting of a senior full stack developer and two medior backend developers
- 18 services
- 200 tests


Employer profile service
Oauth
PDF render (htm to pdf)
JobSDK
Job server
Frontend
ATS (gateway)
PDF viewer (document to pdf)
On demand application converter
MailingSDK
Object storage server
Thumbnailer
RabbitMQ
Cron manager
Mailing service
Application converter
Zipper
MJML render

For example, in a hospital domain, a **Patient** being treated in the outpatients department might have a list of **Referrals**, and methods such as *BookAppointment()*. A **Patient** being treated as an Inpatient however, will have a **Ward** property and methods such as *TransferToTheatre()*. Given this, there are two bounded contexts that patients exist in: Outpatients & Inpatients.

In the insurance domain, the sales team put together a **Policy** that has a degree of risk associated to it and therefore cost. But if it reaches the claims department, that information is meaningless to them. They only need to verify whether the policy is valid for the claim. So there is two contexts here: Sales & Claims

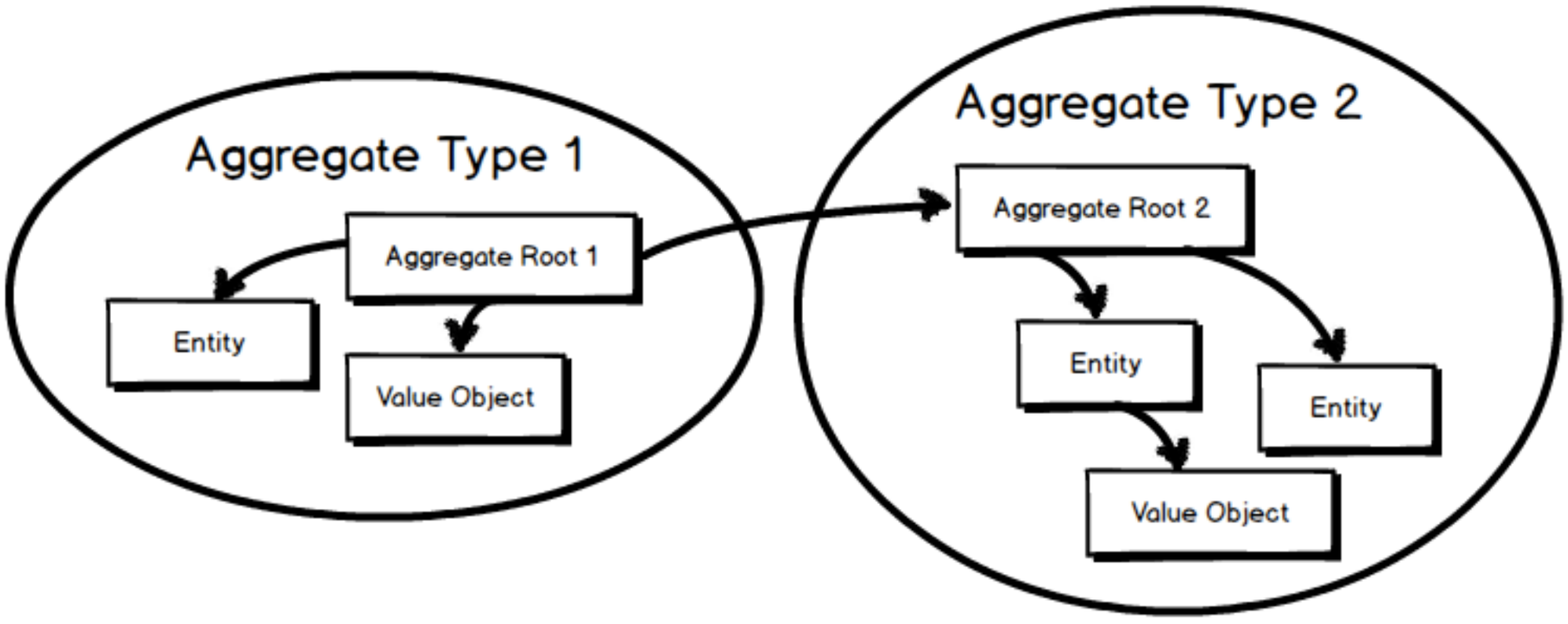
the buyer entity might have most of a person's attributes that are defined in the user entity in the profile or identity microservice, including the identity. But the buyer entity in the ordering microservice might have fewer attributes, because only certain buyer data is related to the order process. The context of each microservice or Bounded Context impacts its domain model.

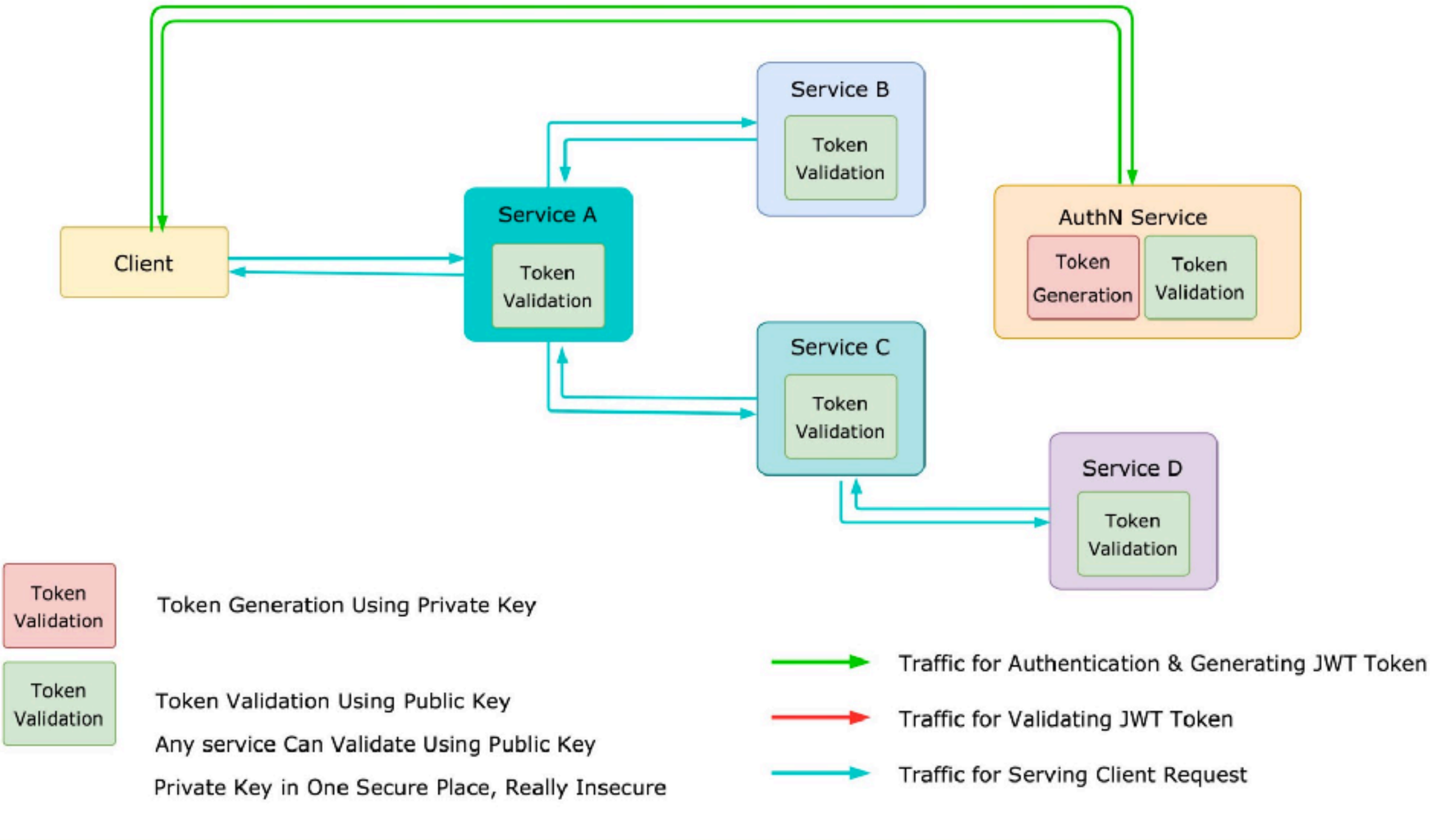
Patterns

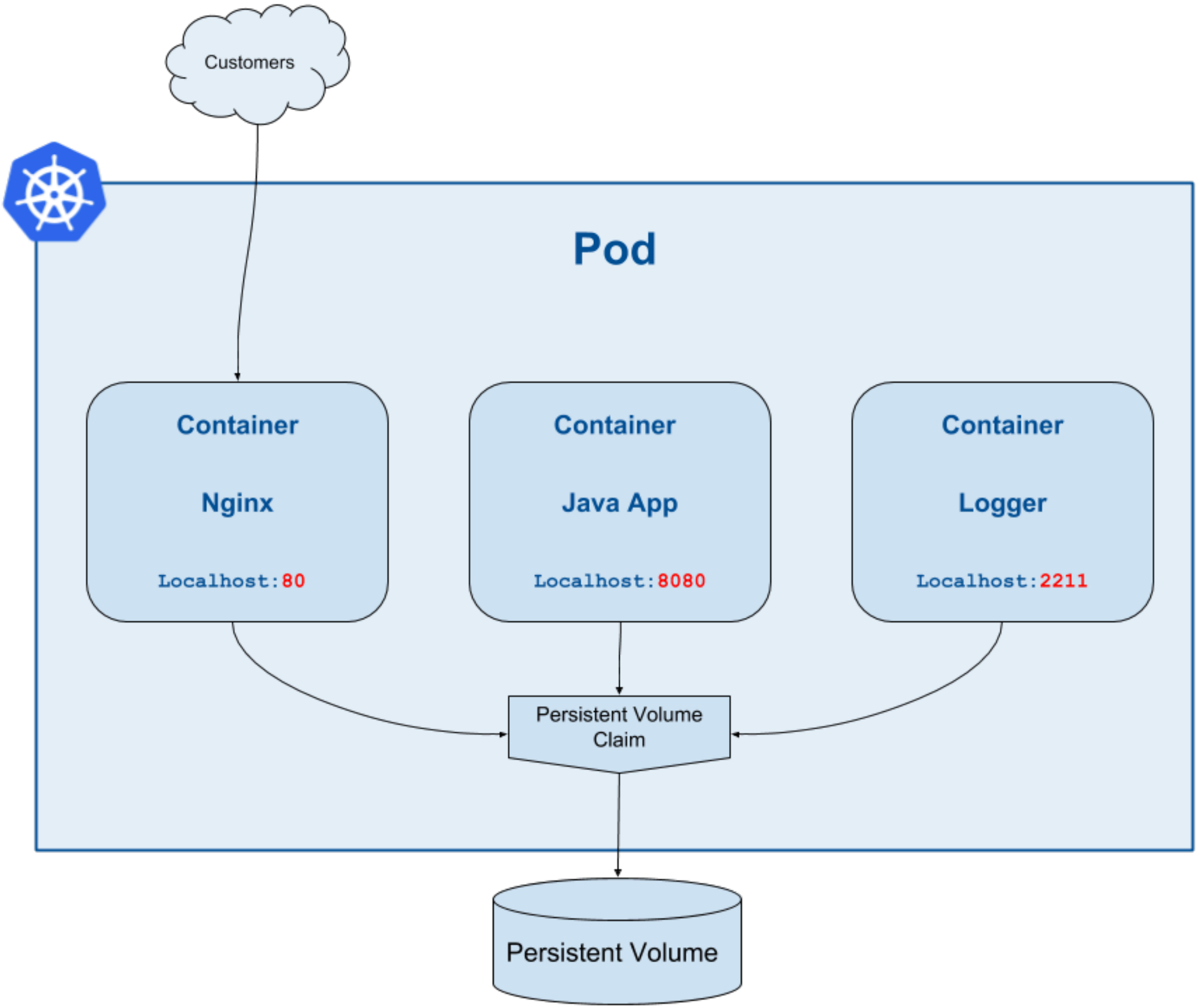
- Implement health monitoring by sending requests to an endpoint on the application. The application should perform the necessary checks, and return an indication of its status.

Bounded Context

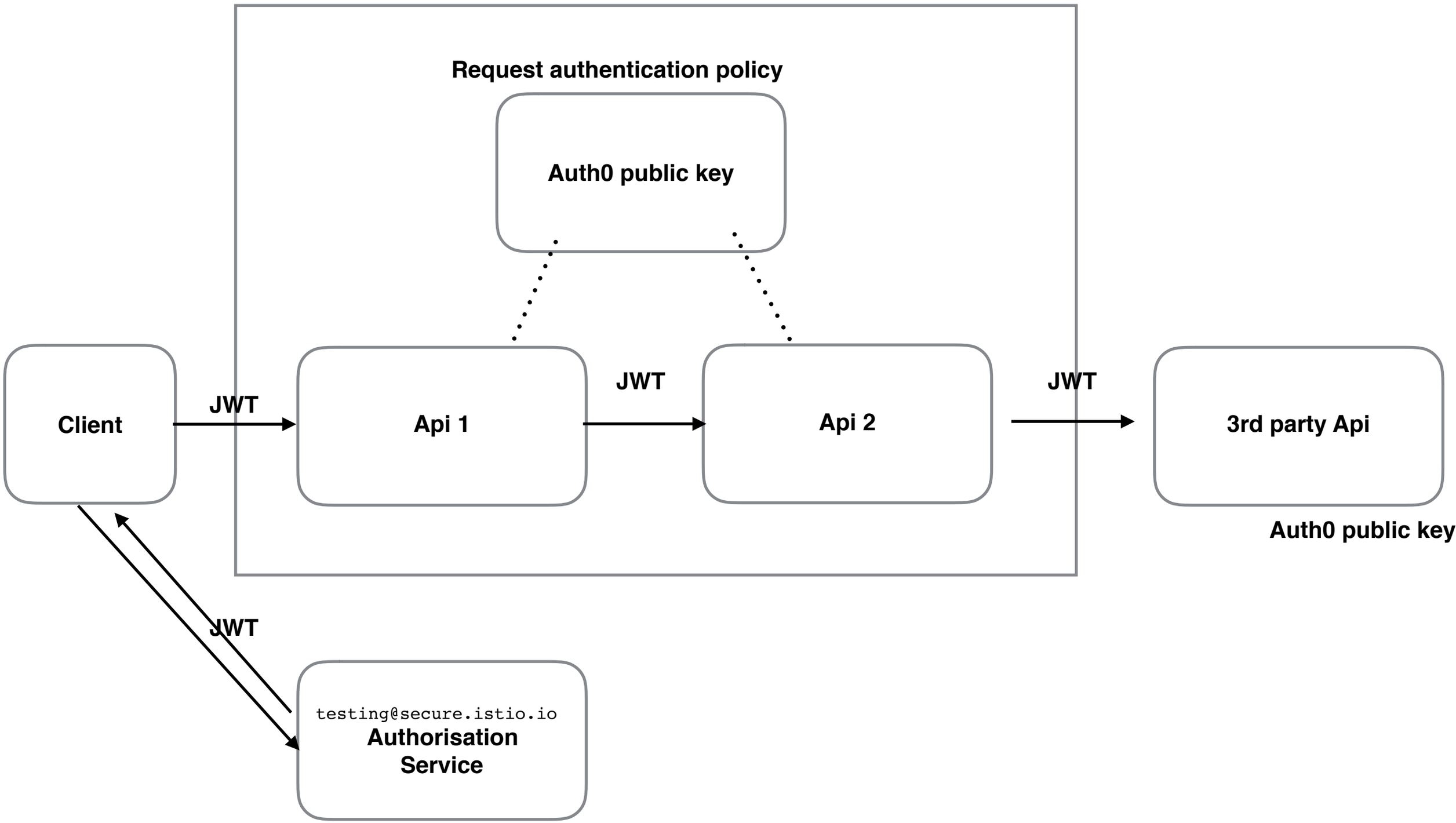
Aggregate Root




Client
Service A
Token Validation
Service B
Token Validation
Service C
Token Validation
Service D
Token Validation
AuthN Service
Token Generation
Token Validation
Token Validation
Token Generation Using Private Key
Token Validation
Token Validation Using Public Key
Any service Can Validate Using Public Key
Private Key in One Secure Place, Really Insecure
Traffic for Authentication & Generating JWT Token
Traffic for Validating JWT Token
Traffic for Serving Client Request


Customers
Pod
Container
Nginx
Localhost:80
Container
Java App
Localhost:8080
Container
Logger
Localhost:2211
Persistent Volume Claim
Persistent Volume

Oath 2


Request authentication policy
Auth0 public key
Client
JWT
Api 1
JWT
Api 2
JWT
3rd party Api
Auth0 public key
JWT
testing@secure.istio.io
Authorisation
Service

- Applications run in *containers*, which in turn run inside of *Pods*.

- All Pods in your Kubernetes cluster have their own IP address and are attached to the same *network*.

- This means all Pods can talk directly to all other Pods.

- However, Pods are unreliable and come and go as scaling operations, rolling updates, rollbacks, failures and other events occur

- Each pod has its own IP address. And each pod thinks it has a totally normal ethernet device called `eth0` to make network requests through.
- When a pod makes a request to the IP address of another node, it makes that request through its own `eth0` interface. This tunnels to the node's respective virtual `vethX` interface.
- A network bridge connects two networks together. When a request hits the bridge, the bridge asks all the connected pods if they have the right IP address to handle the original request.
- In Kubernetes, this bridge is called `cbr0`. Every pod on a node is part of the bridge, and the bridge connects all pods on the same node together.
-

- when the network bridge asks all the pods if they have the right IP address, none of them will say yes.

- After that, the bridge falls back to the default gateway. This goes up to the cluster level and looks for the IP address.

- At the cluster level, there's a table that maps IP address ranges to various nodes. Pods on those nodes will have been assigned IP addresses from those ranges.

- For example, Kubernetes might give pods on node 1 addresses like 100.96.1.1, 100.96.1.2, etc. And Kubernetes gives pods on node 2 addresses like 100.96.2.1, 100.96.2.2, and so on.

- Then this table will store the fact that IP addresses that look like 100.96.1.xxx should go to node 1, and addresses like 100.96.2.xxx need to go to node 2.

- A Service is bound to a ClusterIP, which is a virtual IP address, and no matter what happens to the backend Pods, the ClusterIP never changes, so a client can always send requests to the ClusterIP of the Service.

- A Kubernetes Service object creates a stable network endpoint that sits in front of a set of Pods and load-balances traffic across them.

- You always put a Service in front of a set of Pods that do the same job. For example, you could put a Service in front of your web front-end Pods, and another in front of your authentication Pods. You never put a Service in front of Pods that do different jobs.

- Every service in a cluster is assigned a domain name like my-service.my-namespace.svc.cluster.local.

- when a request is made to a service via its domain name, the DNS service resolves it to the IP address of the service.

- Then kube-proxy converts that service's IP address into a pod IP address.

- You POST a new Service definition to the API Server
- The Service is allocated a ClusterIP (virtual IP address) and persisted to the cluster store
- The cluster's DNS service notices the new Service and creates the necessary DNS A records

```
apiVersion: v1
kind: Service
metadata:
  name: web-svc
  labels:
    blog: svc-discovery
spec:
  type: LoadBalancer
```

It's important to understand that the name registered with DNS is the value of **metadata.name** and that the ClusterIP is dynamically assigned by Kubernetes.

- With service ClusterIP and Kubernetes DNS, service can be easily reached inside a cluster

- If you want more advanced features, such as flexible routing rules, more options for LB, reliable service communication, metrics collection and distributed tracing, etc., then you will need to consider Istio.

- The communication between services is no longer through Kube-proxy but through Istio’s sidecar proxies. Istio sidecar proxy works just like Kube-proxy.

- ClusterIP is only reachable inside a Kubernetes cluster, but what if we need to access some services from outside of the cluster?

- With NodePort, Kubernetes creates a port for a Service on the host, which allows access to the service from the node network.
-

- NodePort is a convenient tool for testing in your local Kubernetes cluster, but it’s not suitable for production because of these limitations.

- A single node is a single point of failure for the system. Once the node is down, clients can’t access the cluster any more.

- A service can be declared as LoadBalancer type to create a layer 4 load balancer in front of multiple nodes. As this layer 4 load balancer is outside of the Kubernetes network, a Cloud Provider Controller is needed for its provision. This Cloud Provider Controller watches the Kubernetes master for the addition and removal of Service resources and configures a layer 4 load balancer in the cloud provider network to proxy the NodePorts on multiple Kubernetes nodes.

if we need to expose multiple services to the outside of a cluster, we must create a LoadBalancer for each service. However, creating multiple LoadBalancers can cause some problems:

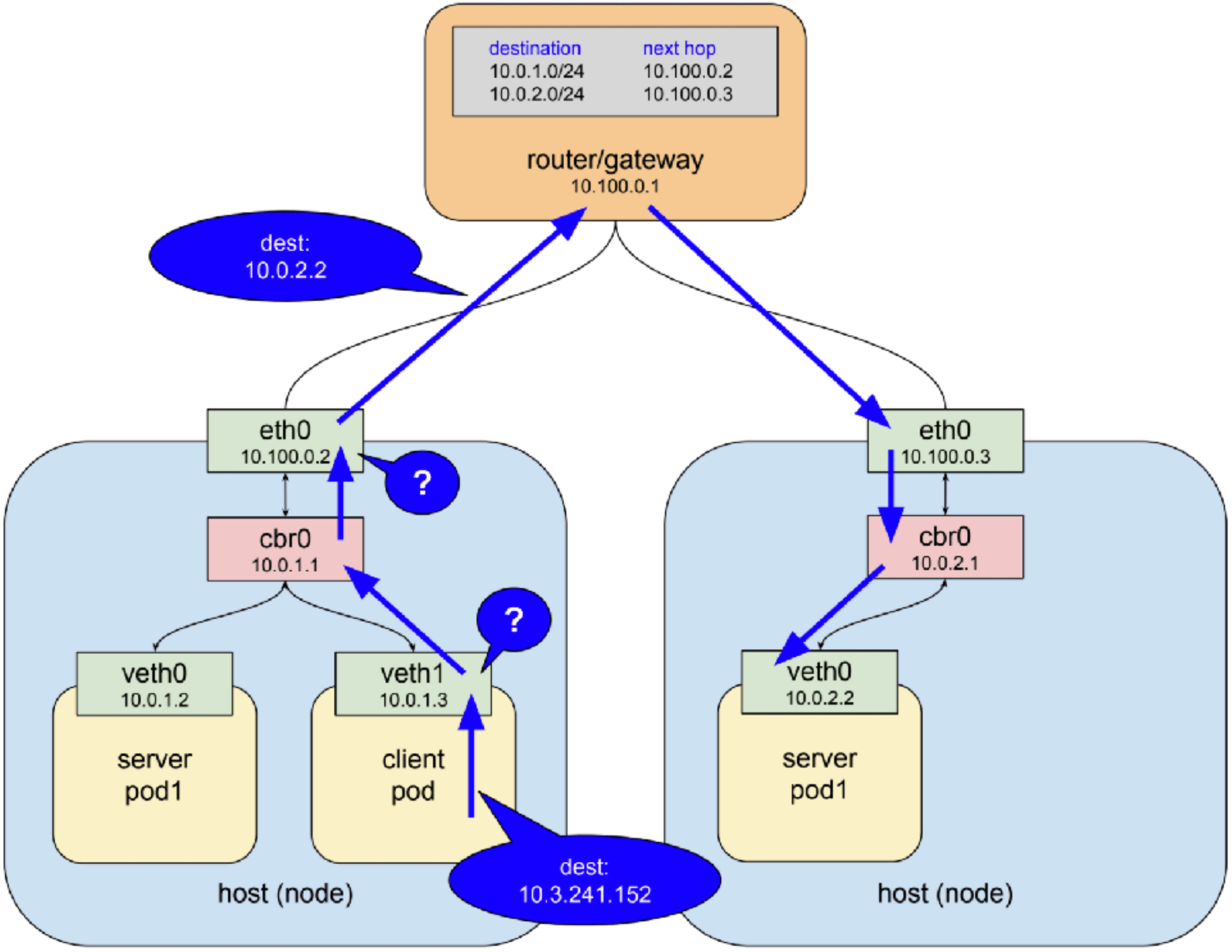
- Needs more public IPs, which normally are limited resources.
- Introduces coupling between the client and the server, making it hard to adjust your backend services when business requirements change.

Kubernetes Ingress resource is used to declare an OSI layer 7 load balancer, which can understand HTTP protocol and dispatch inbound traffic based on the HTTP URL or Host.

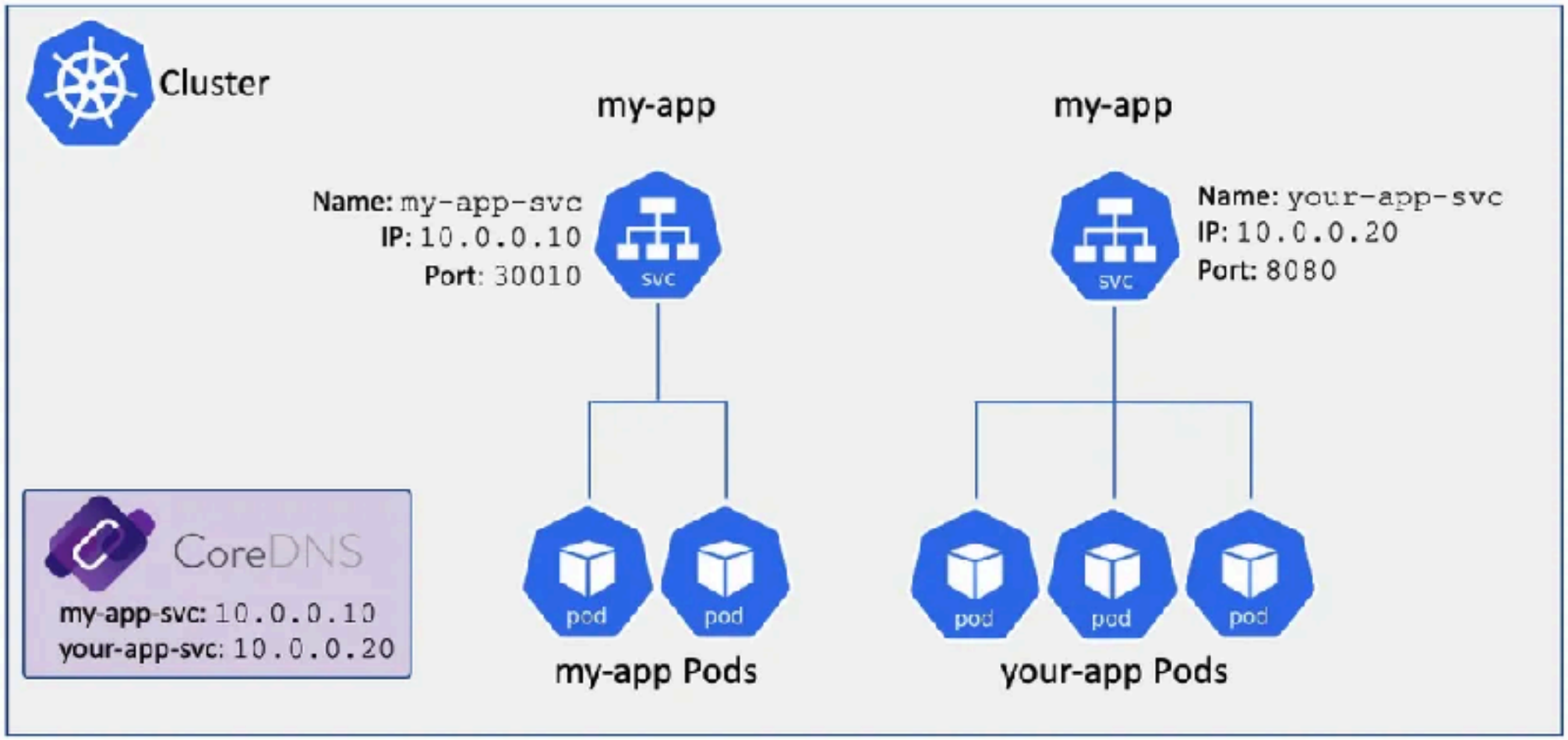
- Ingress controller itself is also deployed as Pods inside the cluster.

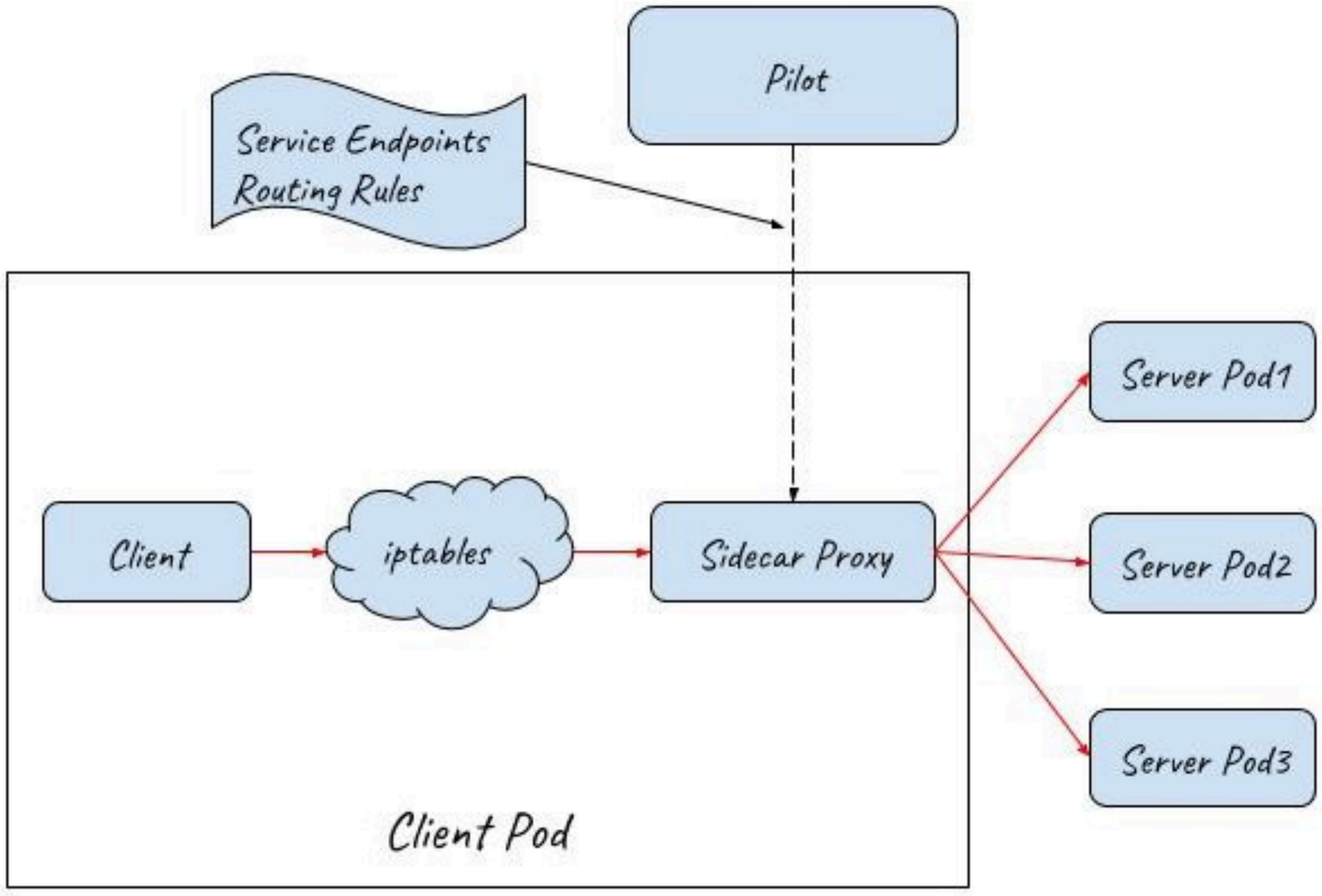
- Ingress controller provides a unified entrance for the HTTP services in a cluster, but it can’t be accessed directly from outside because the ingress controller itself is also deployed as Pods inside the cluster.

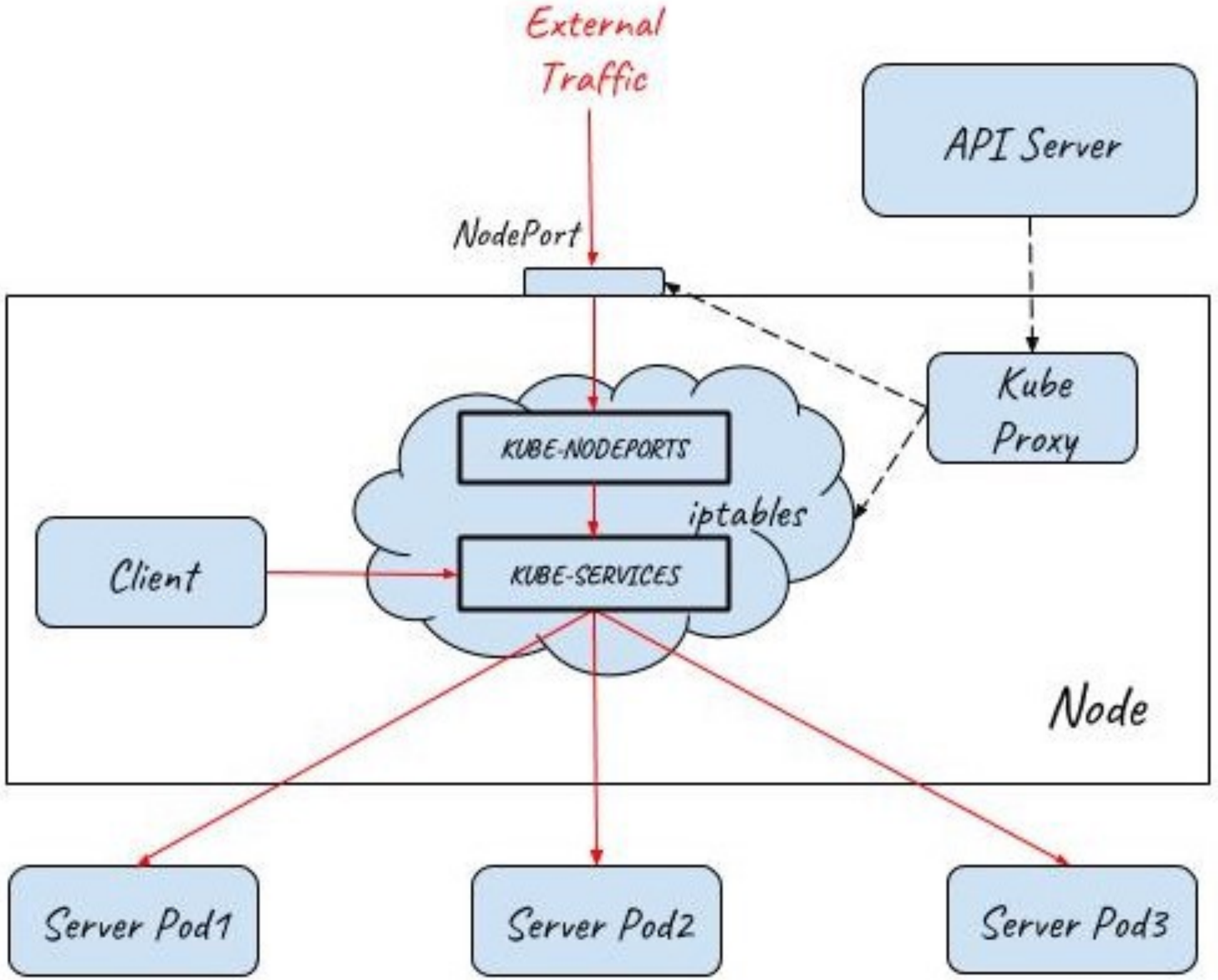
- Ingress controller must work together with NodePort and LoadBalancer to provide the full path for the external traffic to enter the cluster.

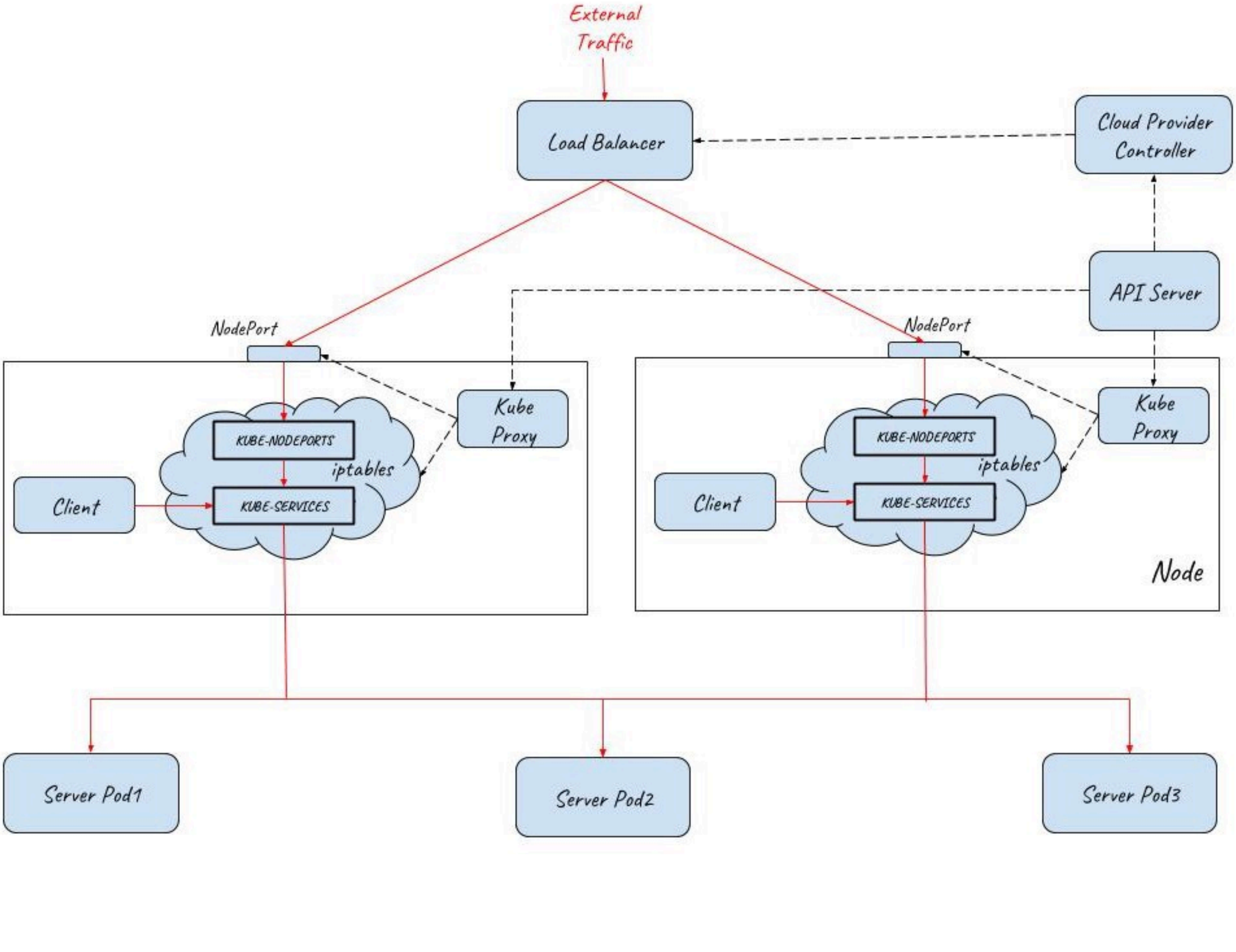

destination
next hop
10.0.1.0/24
10.100.0.2
10.0.2.0/24
10.100.0.3
router/gateway
10.100.0.1
dest:
10.0.2.2
eth0
10.100.0.2
?
cbr0
10.0.1.1
?
veth0
10.0.1.2
veth1
10.0.1.3
server
pod1
client
pod
dest:
10.3.241.152
host (node)
eth0
10.100.0.3
cbr0
10.0.2.1
veth0
10.0.2.2
server
pod1
host (node)

	OSI Layer	TCP/IP	Datagrams are called
Software	**Layer 7** Application	HTTP, SMTP, IMAP, SNMP, POP3, FTP	**Upper Layer Data**
	Layer 6 Presentation	ASCII Characters, MPEG, SSL, TSL, Compression (Encryption & Decryption)	
	Layer 5 Session	NetBIOS, SAP, Handshaking connection	
	Layer 4 Transport	TCP, UDP	**Segment**
	Layer 3 Network	IPv4, IPv6, ICMP, IPSec, MPLS, ARP	**Packet**
Hardware	**Layer 2** Data Link	Ethernet, 802.1x, PPP, ATM, Fiber Channel, MPLS, FDDI, MAC Addresses	**Frame**
	Layer 1 Physical	Cables, Connectors, Hubs (DLS, RS232, 10BaseT, 100BaseTX, ISDN, T1)	**Bits**

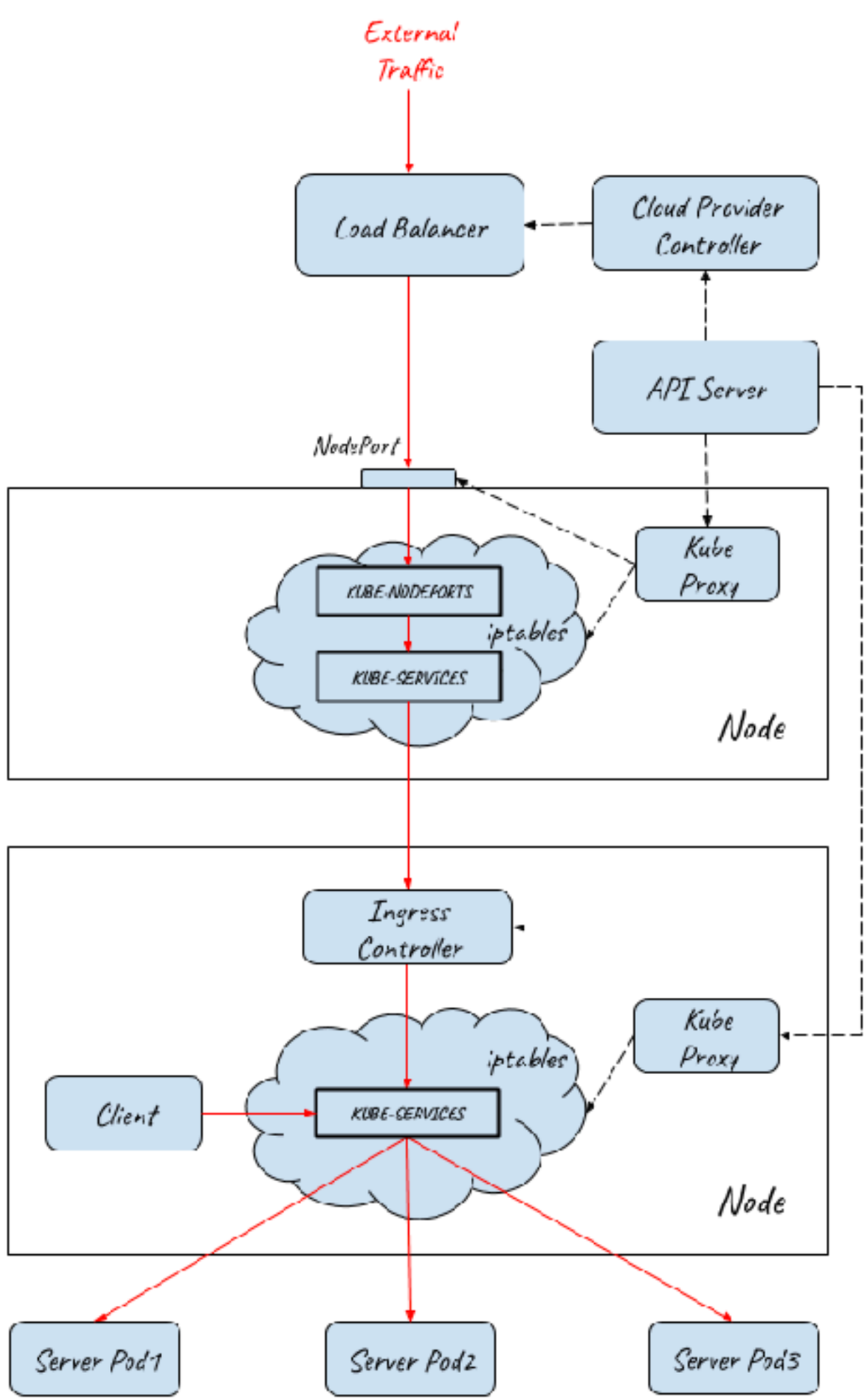

Cluster
my-app
Name: my-app-svc
IP: 10.0.0.10
Port: 30010
svc
my-app
Name: your-app-svc
IP: 10.0.0.20
Port: 8080
svc
CoreDNS
my-app-svc: 10.0.0.10
your-app-svc: 10.0.0.20
pod
pod
my-app Pods
pod
pod
pod
your-app Pods


Pilot
Service Endpoints
Routing Rules
Server Pod1
Client
iptables
Sidecar Proxy
Server Pod2
Server Pod3
Client Pod


External
Traffic
API Server
NodePort
Kube
Proxy
KUBE-NODEPORTS
iptables
Client
KUBE-SERVICES
Node
Server Pod1
Server Pod2
Server Pod3


External Traffic
Load Balancer
Cloud Provider Controller
API Server
NodePort
NodePort
Kube Proxy
Kube Proxy
KUBE-NODEPORTS
KUBE-NODEPORTS
iptables
iptables
KUBE-SERVICES
KUBE-SERVICES
Client
Client
Node
Server Pod1
Server Pod2
Server Pod3

1. Internet/External traffic reaches the layer 4 load balancer.
2. Load balancer dispatches traffic to multiple NodePorts on the Kubernetes.
3. Traffic is captured by iptables and redirected to ingress controller Pods.
4. Ingress controller sends traffic to different Services according to ingress rules.
5. Finally, traffic is redirected to the backend Pods by iptables.



Client Request
→ Load Balancer(External IP)
→ Load Balancer (Node IP)
→ Ingress Controller Service(ClusterIP)
→ Ingress Controller Pod(Pod IP)
→ Backend Service(ClusterIP)
→ Backend Pod(Pod IP)

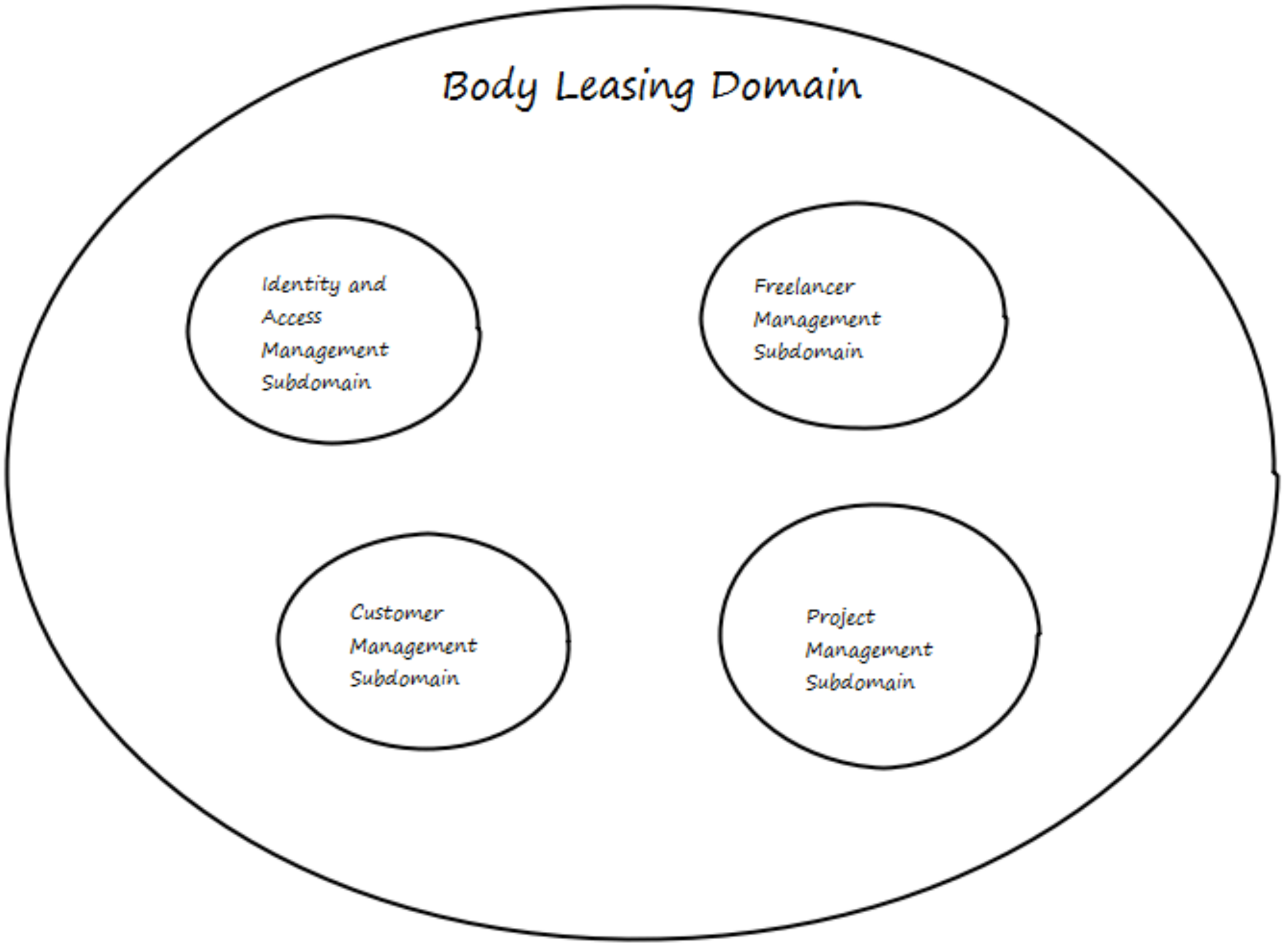
A searchable catalog of Customers

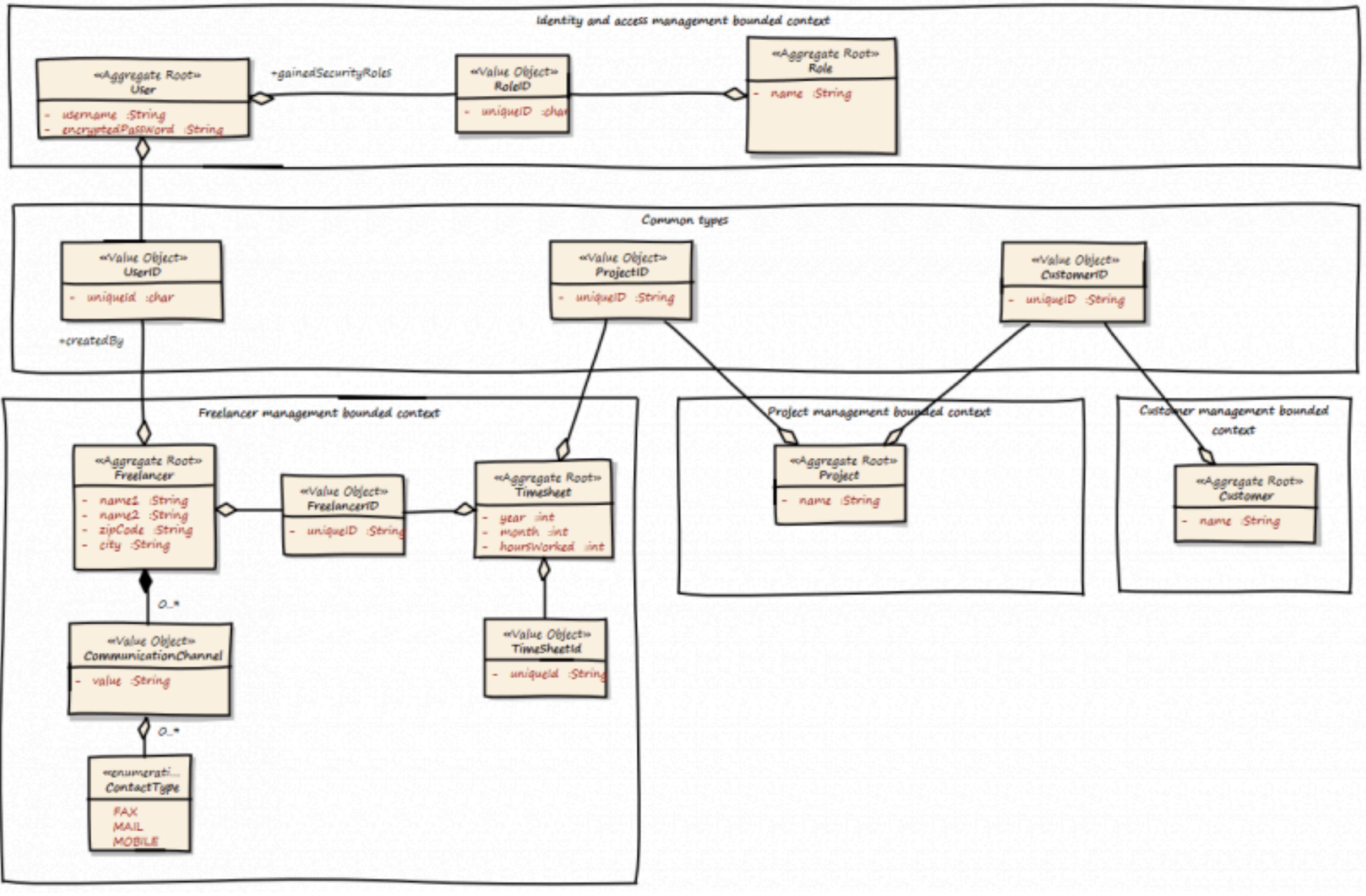
A searchable catalog of Freelancer

A searchable catalog of Projects

Timesheets for the Freelancers

Make Payments to the Freelancers


Body Leasing Domain
Identity and Access Management Subdomain
Freelancer Management Subdomain
Customer Management Subdomain
Project Management Subdomain


Identity and access management bounded context
«Aggregate Root»
User
- username :String
- encryptedPassWord :String
+gainedSecurityRoles
«Value Object»
RoleID
- uniqueID :char
«Aggregate Root»
Role
- name :String
Common types
«Value Object»
UserID
- uniqueId :char
+createdBy
«Value Object»
ProjectID
- uniqueID :String
«Value Object»
CustomerID
- uniqueID :String
Freelancer management bounded context
«Aggregate Root»
Freelancer
- name1 :String
- name2 :String
- zipCode :String
- city :String
«Value Object»
FreelancerID
- uniqueID :String
«Aggregate Root»
Timesheet
- year :int
- month :int
- hoursWorked :int
0..*
«Value Object»
CommunicationChannel
- value :String
«Value Object»
TimeSheetId
- uniqueId :String
0..*
«enumerati...
ContactType
FAX
MAIL
MOBILE
Project management bounded context
«Aggregate Root»
Project
- name :String
Customer management bounded context
«Aggregate Root»
Customer
- name :String

«Presentation»
Client

«use»

Transaction boundary

«Application Service»
FreelancerApplicationService

+ freelancerMoved(FreelancerMovedCommand) :void
+ getPresentationModelFor(FreelancerId) :FreelancerPresentationModel

Invokes usecase on

«Value Object»
Address

- name1 :String
- name2 :String
- zipCode :String
- city :String

+ getName1() :String
+ getName2() :String
+ getCity() :String
+ getZipCode() :String
+ changeName1(String) :Address
+ changeName2(String) :Address
+ changeCity(String) :Address
+ changZipCode(String) :Address

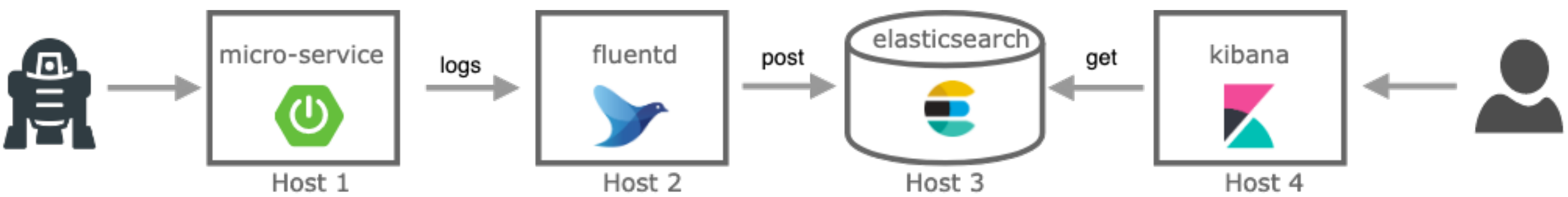
«Aggregate Root»
Freelancer

+ getAddress() :Address
+ movedTo(Address) :void

«Domain Repository»
FreelancerRepository

+ findById(FreelancerId) :Freelancer
+ add(Freelancer) :void
+ remove(Freelancer) :void
+ findByName(String) :Collection<Freelancer>

https://www.mirkosertic.de/blog/2013/04/domain-driven-design-example/


micro-service
logs
fluentd
post
elasticsearch
get
kibana
Host 1
Host 2
Host 3
Host 4

- AAA

- Authentication
- Authorization
- Audit
- Asset Handling (In Rest and Transit)
- Exception Handling
- Session management
- Input Validation
- Key

- STRIDE
- Spoofing
- Tampering
- Repudation
- Information Disclosure
- Deniel of Service
- Elevation of Privilege

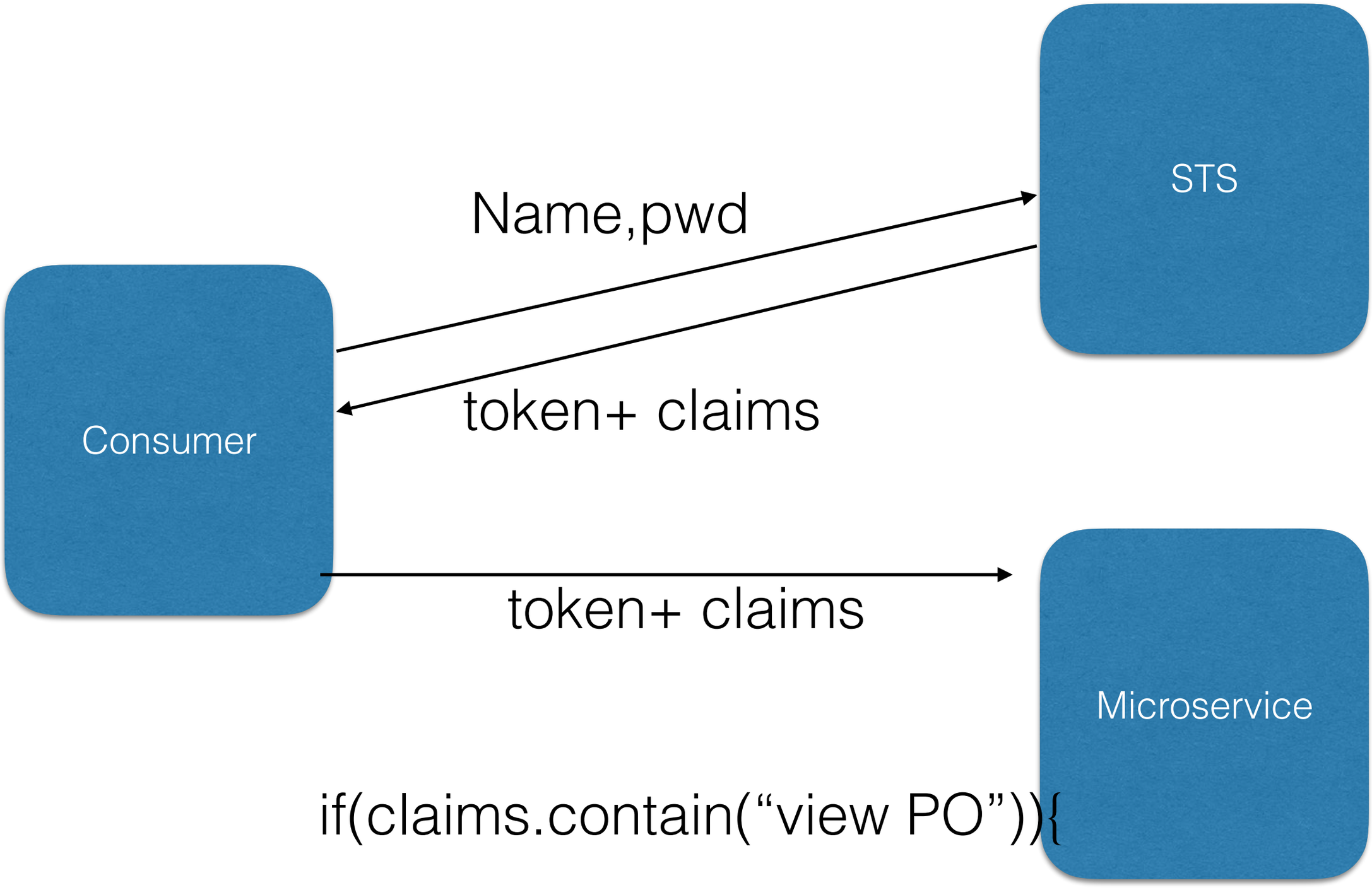
• Authentication	Opened Connect + OAuth2
• Authorization	Claim based
• Audit	Event Store, Event Sourcing
• Asset in Transit	HTTPS/ WSS
• Asset in Rest	Storage Encryption
• Exception Handling	Fwk
• Session management	Fwk
• Input Validation	myCode (fwk) , WAF
• Key	Vault

- By what you know: password, secret <— ***
- By What you have : otp, mail, cert, rsa <—
- By What you Are : Bio (fingerprint, voice, retina, …)
- Your Location:
- Behaviourial
- 2 factor
- multifactor

- By what you know
- Custom Credential Store (not recommended)
- Oauth2 (…)
- OpenIDconnect
- SAML

Claim mapping

Microservice1-> Purchase->c1,c2,c3




```
if(claims.contain("view PO")){
      ....
}
```

token (userid)


```
if(role == “Owner”){
    ....
}
```

userid->role

token, (viewPurchase,Delete sales)


```
if(viewPurchase){
    ....
}
```