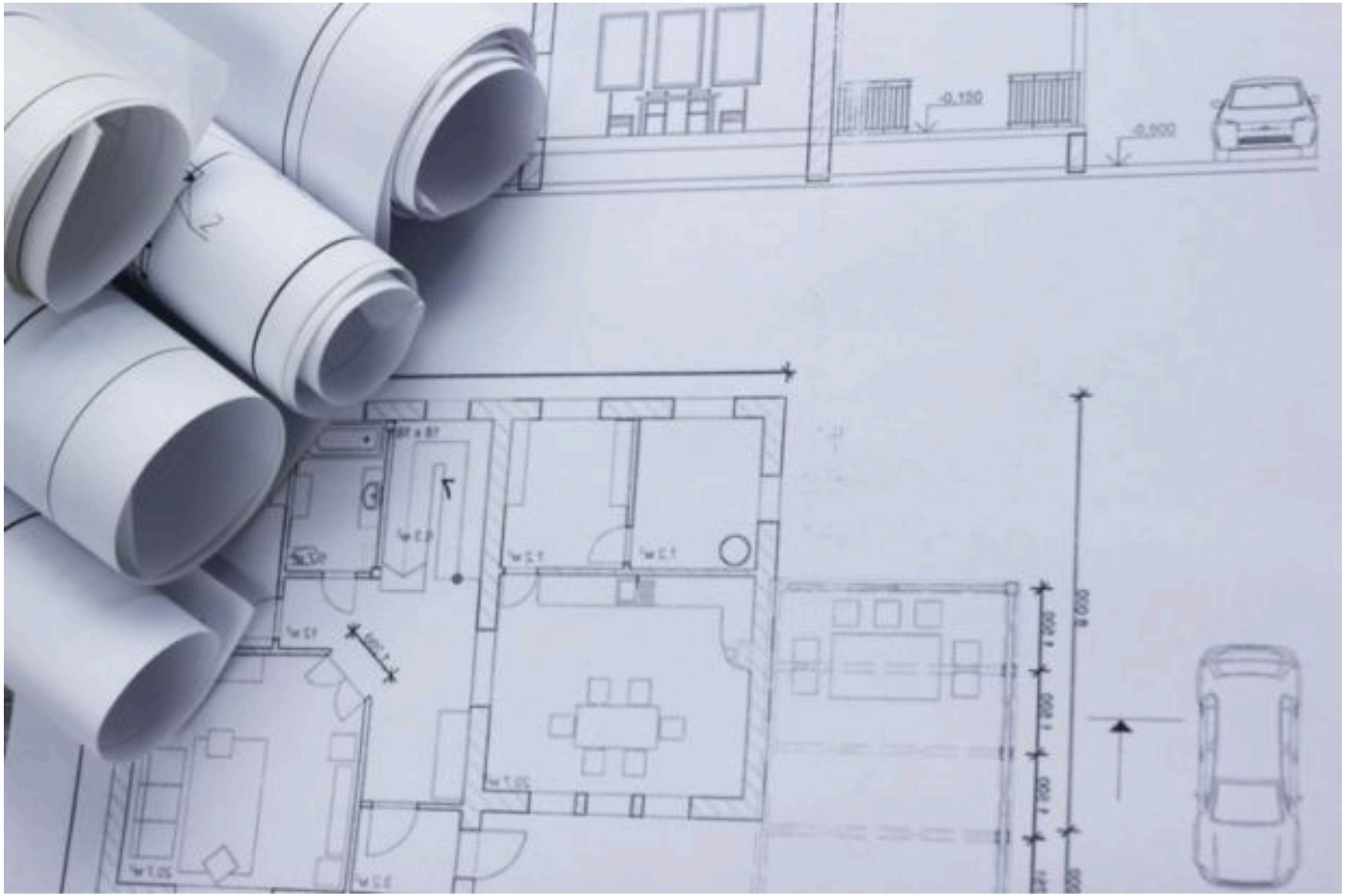


Software Architecture

» Skan.ai - chief Architect

» Ai.robotics - chief Architect

» Genpact - solution Architect

» Welldoc - chief Architect

» Microsoft

» Mercedes

» Siemens

» Honeywell

Mubarak

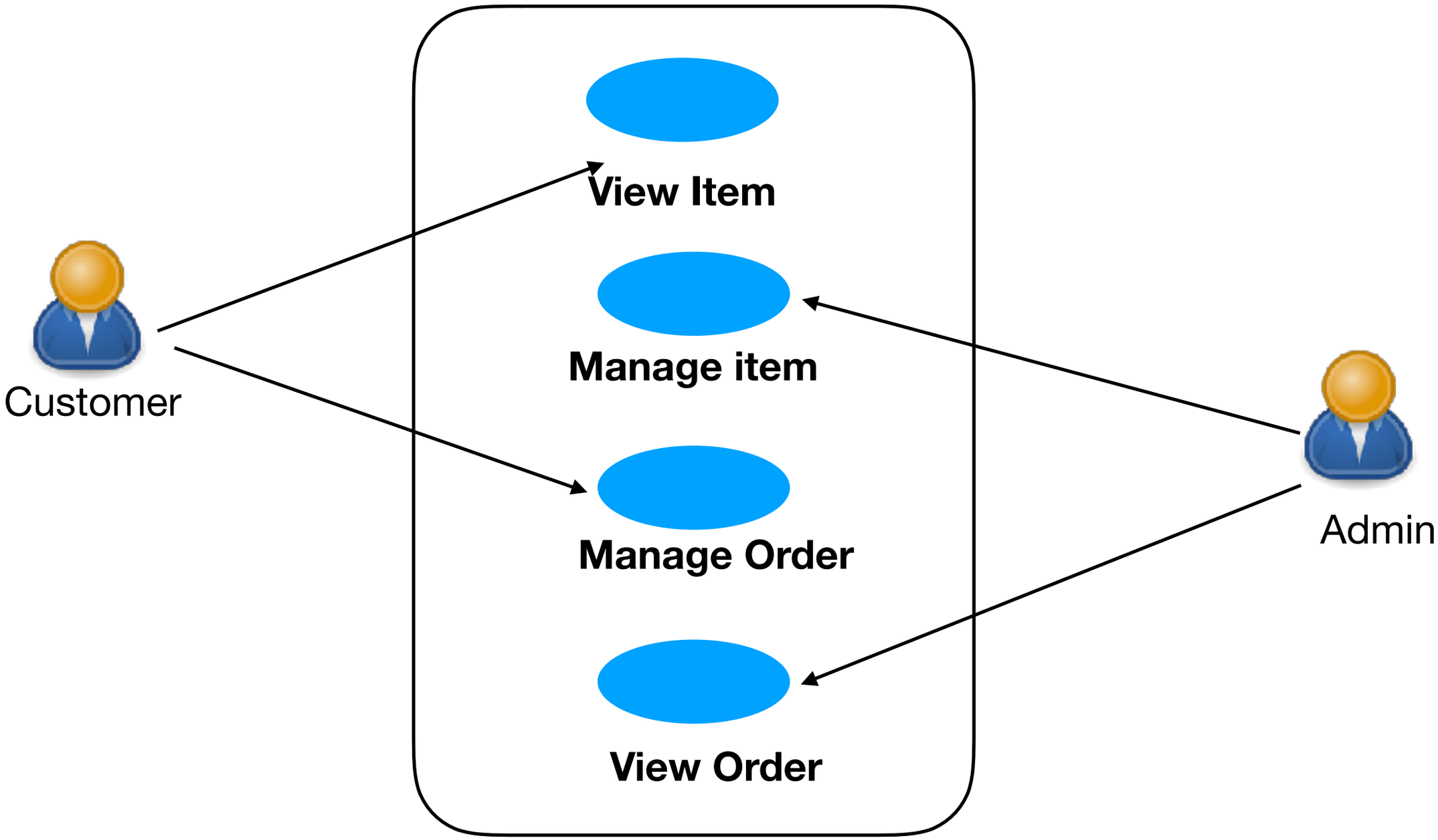
- Application Architecture Scope/ Role
- Arch Requirements
- Arch Design
- Arch Doc
- Arch Eval

- Years of experience
- Technology stack
- Business Domain
- Expectations

Books

- Software Architecture in Practice - SEI
- POSA (Pattern oriented Software Architecture v1,2,3,4)
- Beautiful Architecture
- 97 things every architect should know
- Architecture Boot camp
-

Great Deal


View Item
Manage item
Manage Order
View Order
Customer
Admin

Architecture and Design

Quality

1. Correctness
2. Maintainability
   # dev
   # test
   # ops
   # deployment
3. Availability
4. Reliability (trust)
5. Security (trust)
6. Performance (includes scalability)
   # CPU
   # Memory
   # Network
   # disk
7. Robustness (rugud)
8. Usability (UX)
9. Portability
10. Interoperability

Measure

* response time
* tps
* Uptime
* Build time
* Change request time
* No of clicks
* PF of failure
* mtbf

Approach / tactic

1. Caching
2. Lazy loading
3. Sharding
4. Parllel
5. Compression
6. Modular
7. Log
8. Abstraction
9. Doc
10. Reusability
11. ACID (transaction)
12. Auth
13. Input validation
14.

1. Maintainability
2. Availability
3. Reliability (trust)
4. Security (trust)
5. Performance (includes scalability)
6. Robustness (rugud)

Quality & Measure

1. Collect

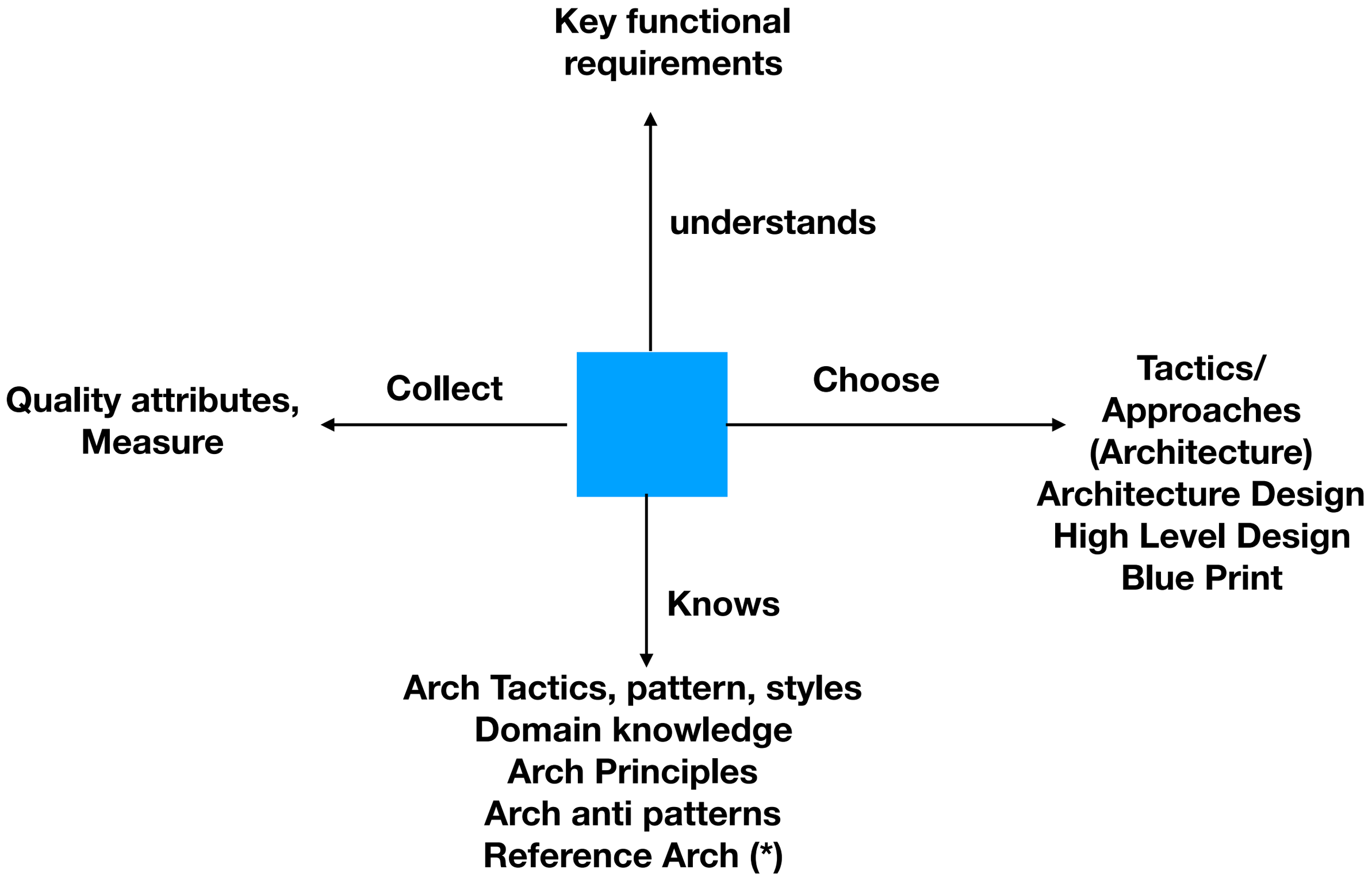
1. Context View
2. Functional View (high level)
3. Quality View (important / significant fun)
4. Constraints
5. Assumptions

2. Choose

Approach / tactic

<<Architecture>>

Architecture Design
Blue print
System Design
Skeleton
Eagles view
HLD


Key functional requirements
understands
Quality attributes, Measure
Collect
Choose
Tactics/ Approaches (Architecture) Architecture Design High Level Design Blue Print
Knows
Arch Tactics, pattern, styles
Domain knowledge
Arch Principles
Arch anti patterns
Reference Arch (*)

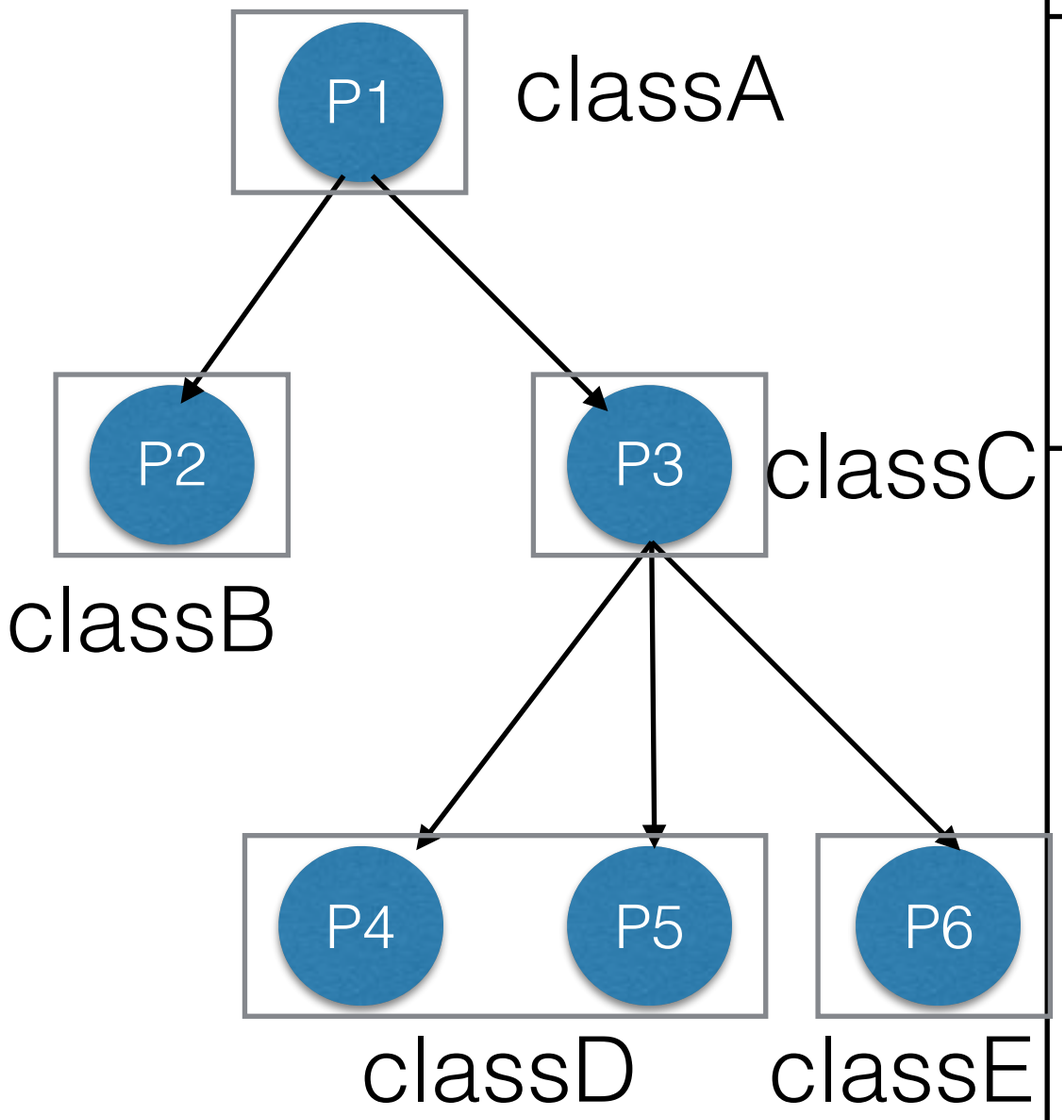
Before

Engineering v/s Tuning

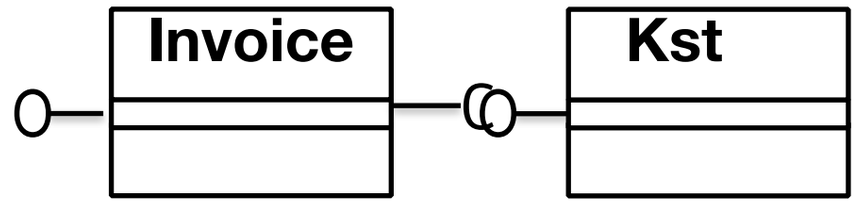
After

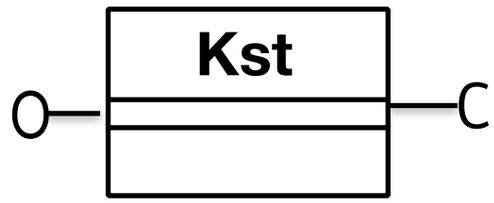
Procedural, OO, functional

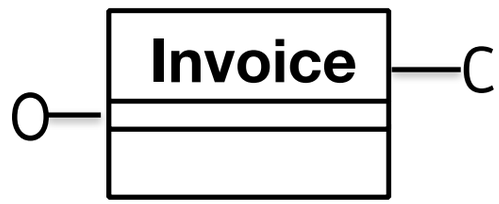
Procedural Prog (tree)

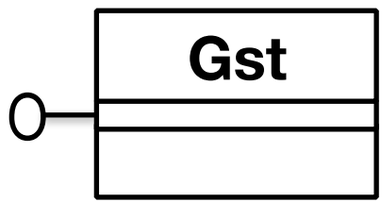


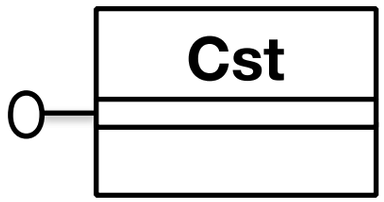
OO Prog (Lego)



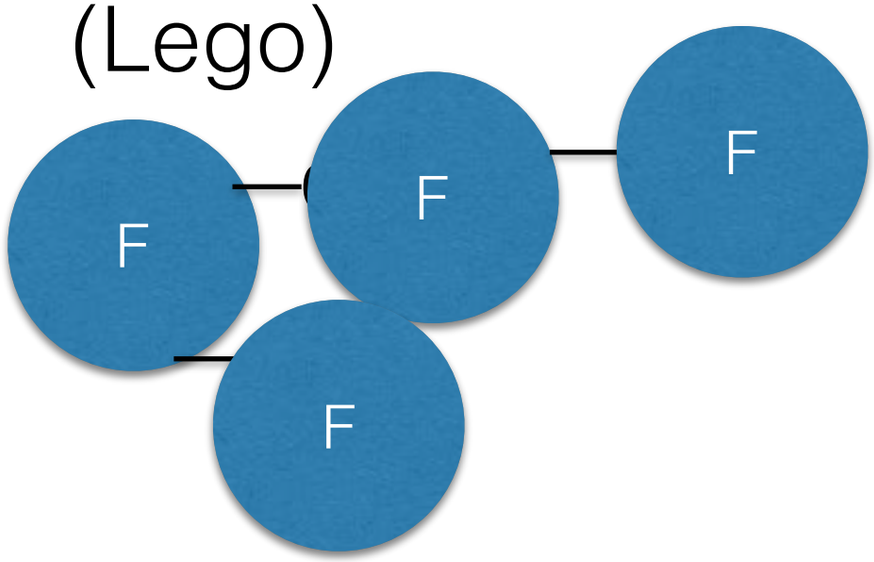


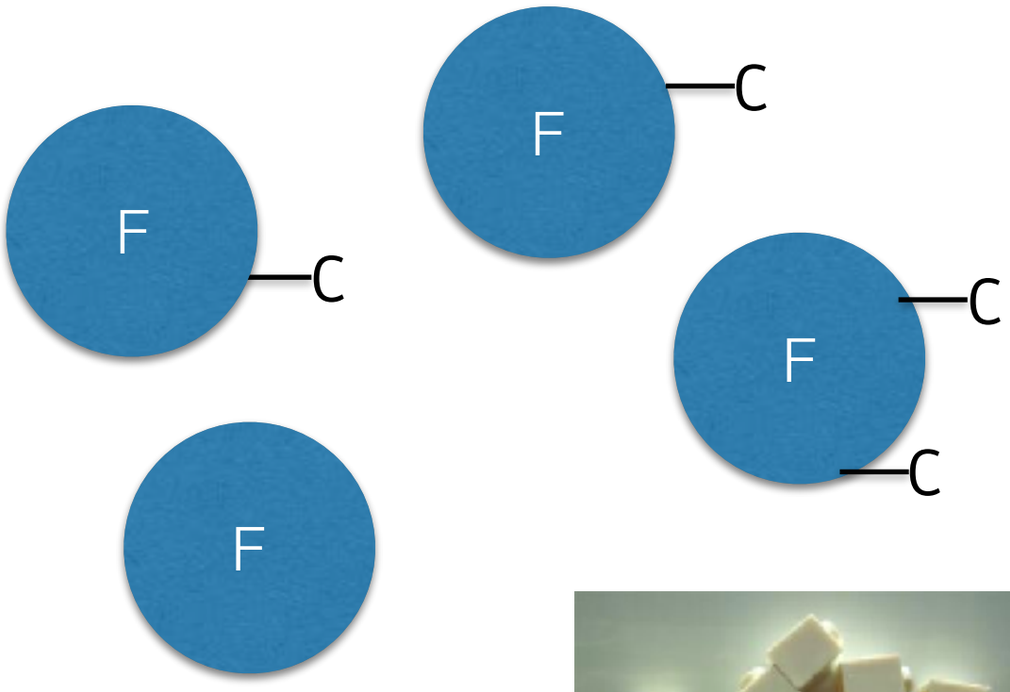




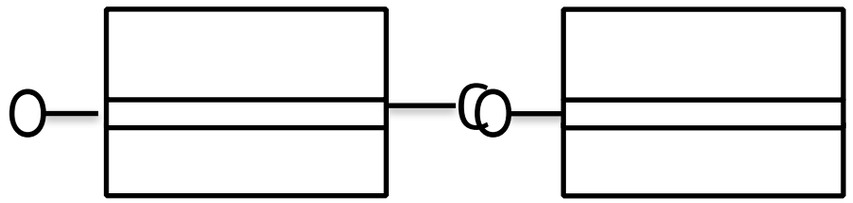


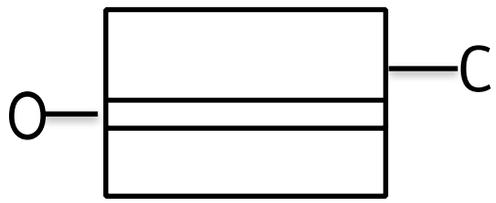
Functional Prog (Lego)

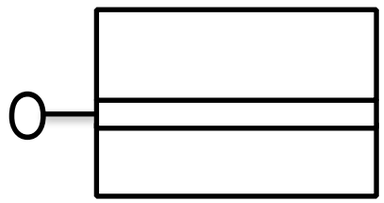


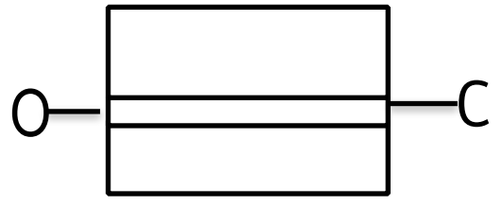


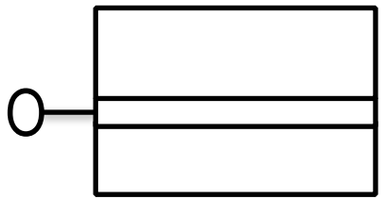
OO Prog (Lego)





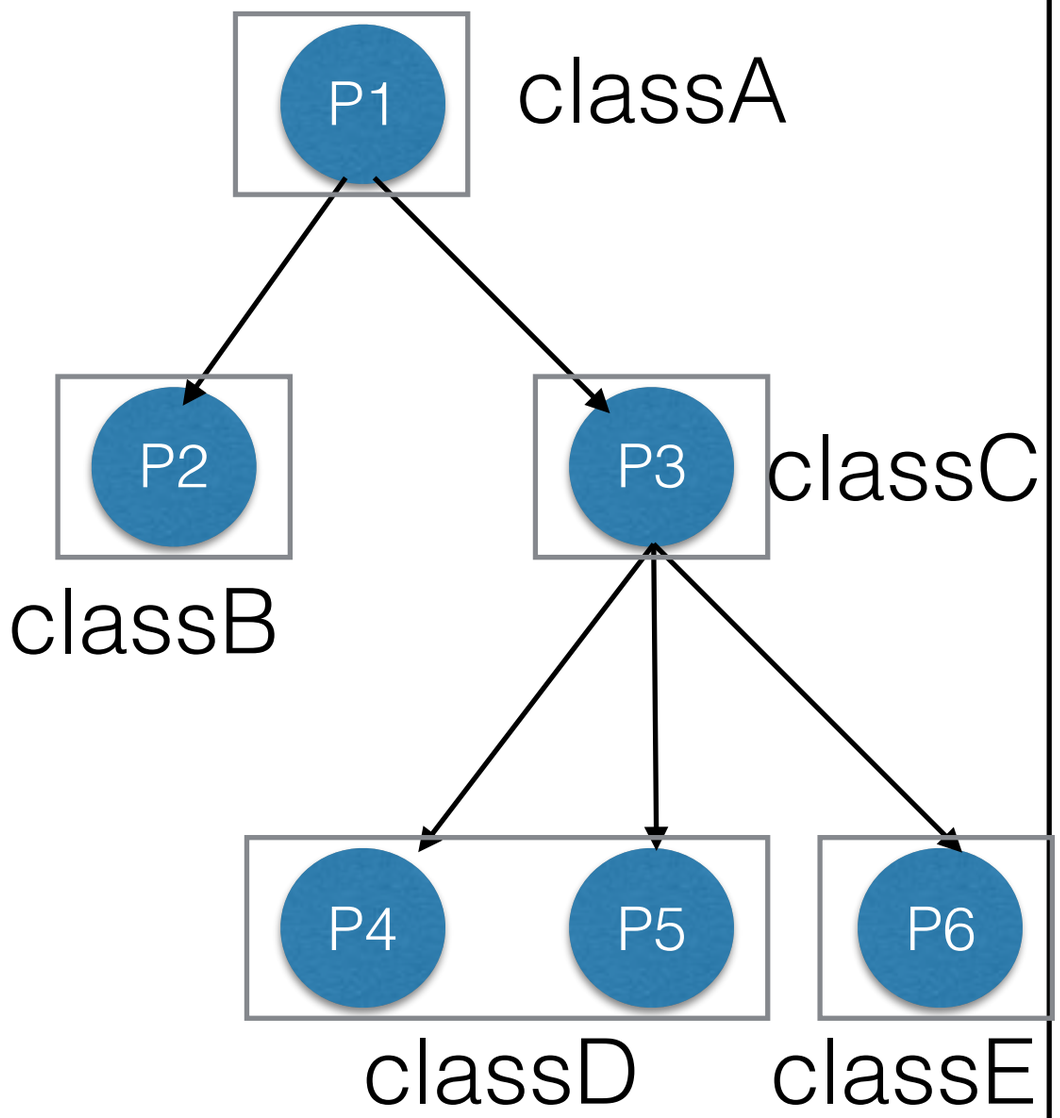






Procedural Prog

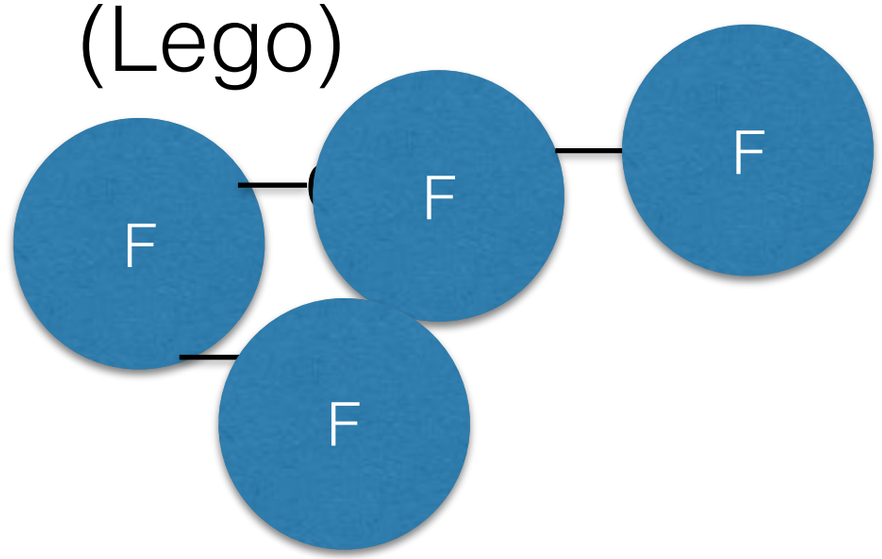
(tree)

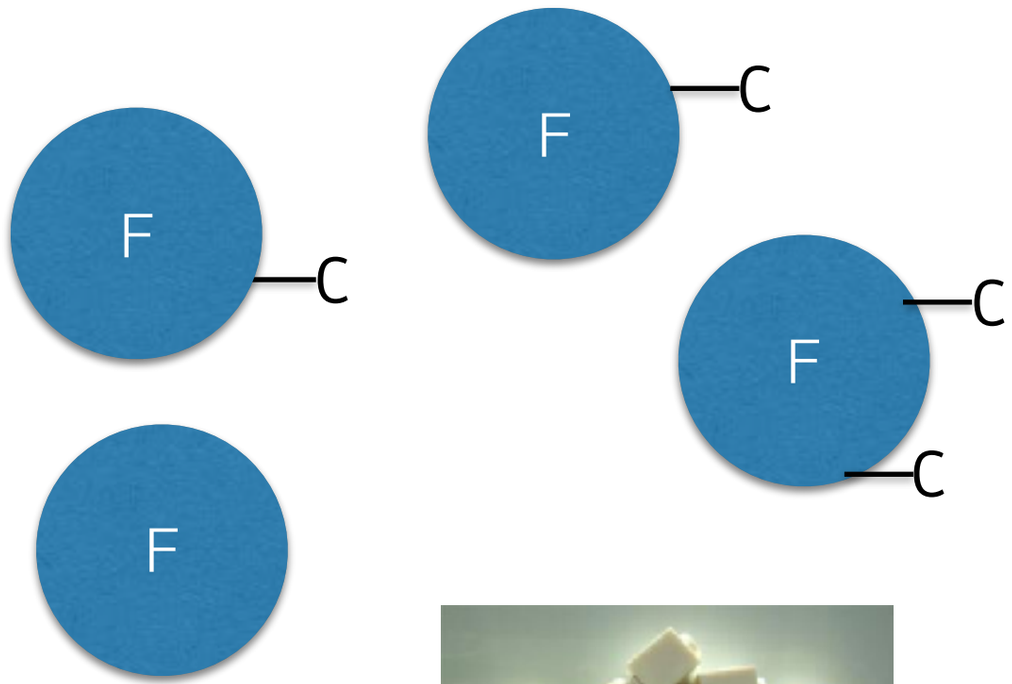


(top down)

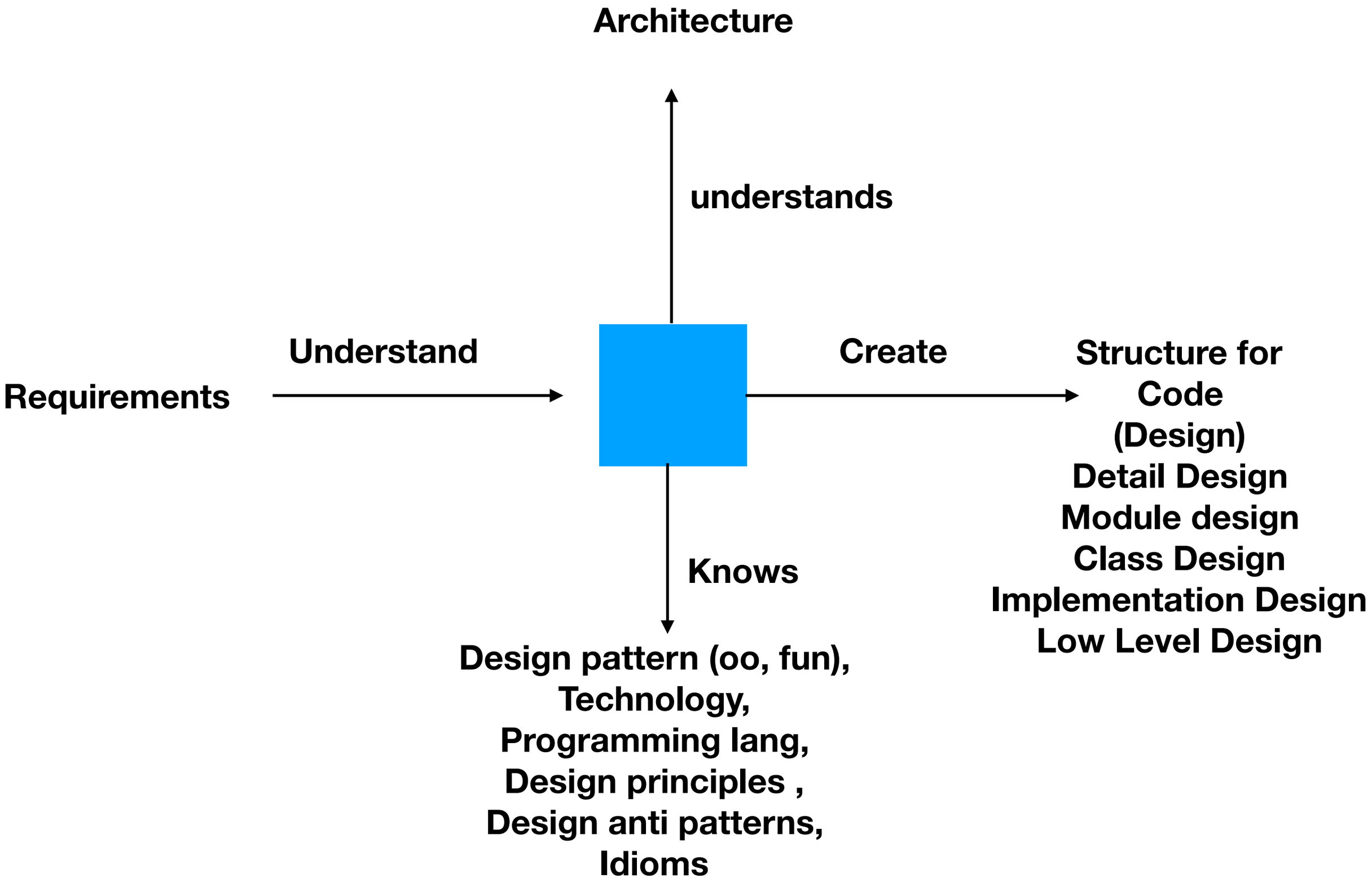
Functional Prog

(Lego)

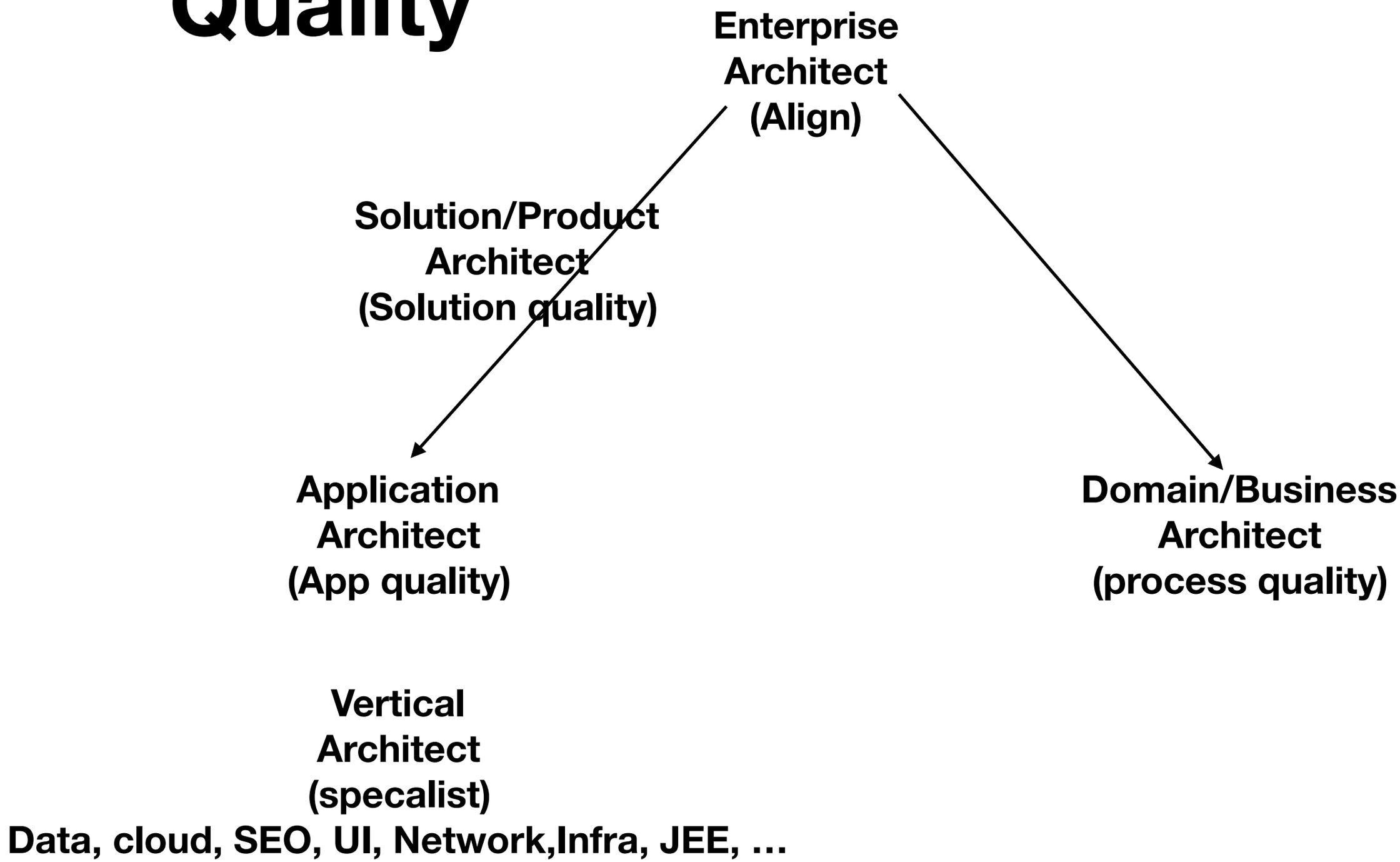




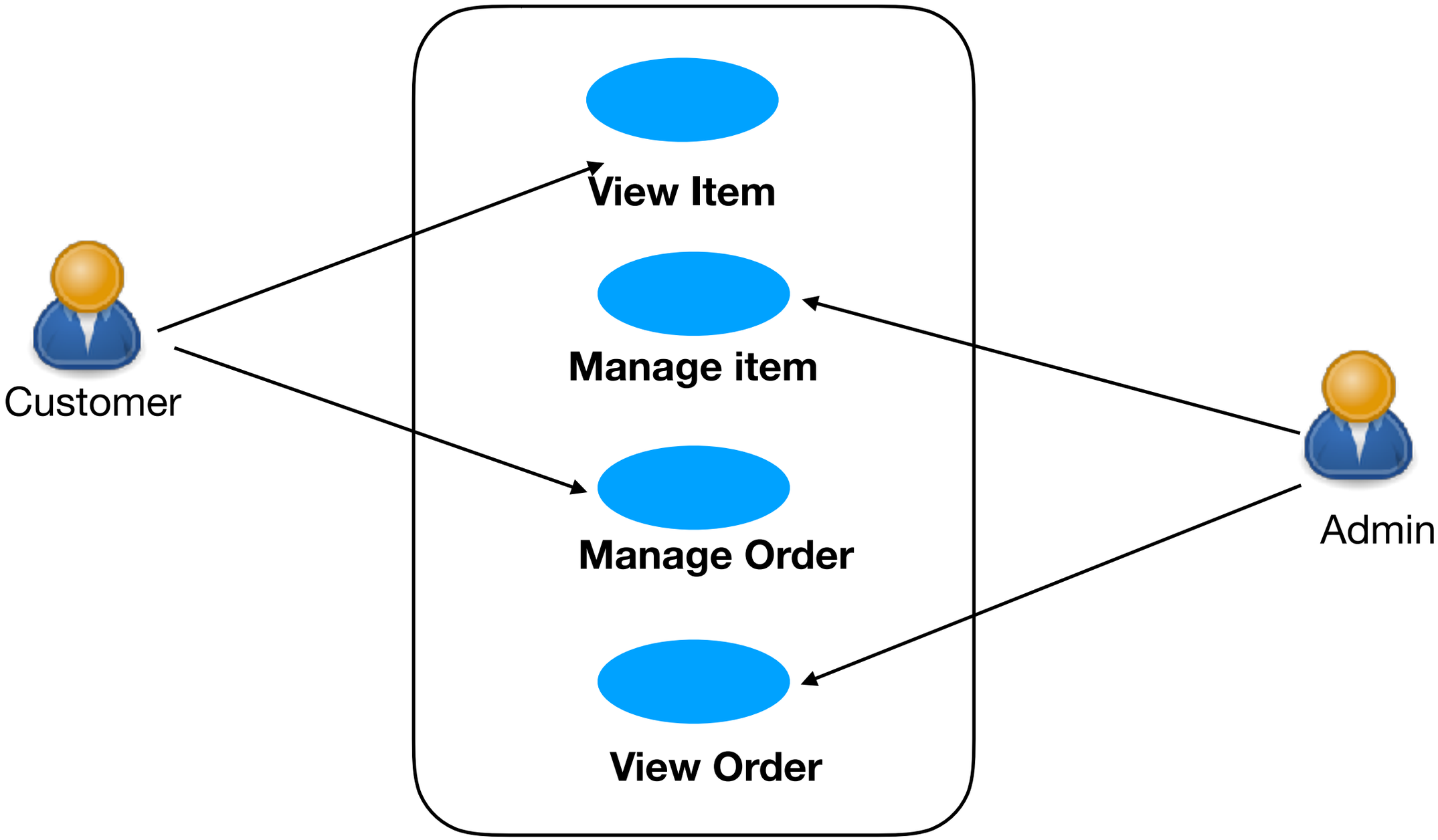
Coding Style	Proc	OO	Fun
Performance	-	-	+ +
Unit Testability	- -	+ +	+ + +
Time to develop/ cost	+ +	- -	+
Manage Large Code	- -	+ +	+
Learning Curve	+ +	- -	-
Language	C, java, js, py	Java, js, py	Java, js, py, scala, kotlin, Haskel


Architecture
understands
Understand
Requirements
Create
Structure for
Code
(Design)
Detail Design
Module design
Class Design
Implementation Design
Low Level Design
Knows
Design pattern (oo, fun),
Technology,
Programming lang,
Design principles ,
Design anti patterns,
Idioms

Quality


Enterprise
Architect
(Align)
Solution/Product
Architect
(Solution quality)
Application
Architect
(App quality)
Domain/Business
Architect
(process quality)
Vertical
Architect
(specalist)
Data, cloud, SEO, UI, Network,Infra, JEE, …

Great Deal


View Item
Manage item
Manage Order
View Order
Customer
Admin

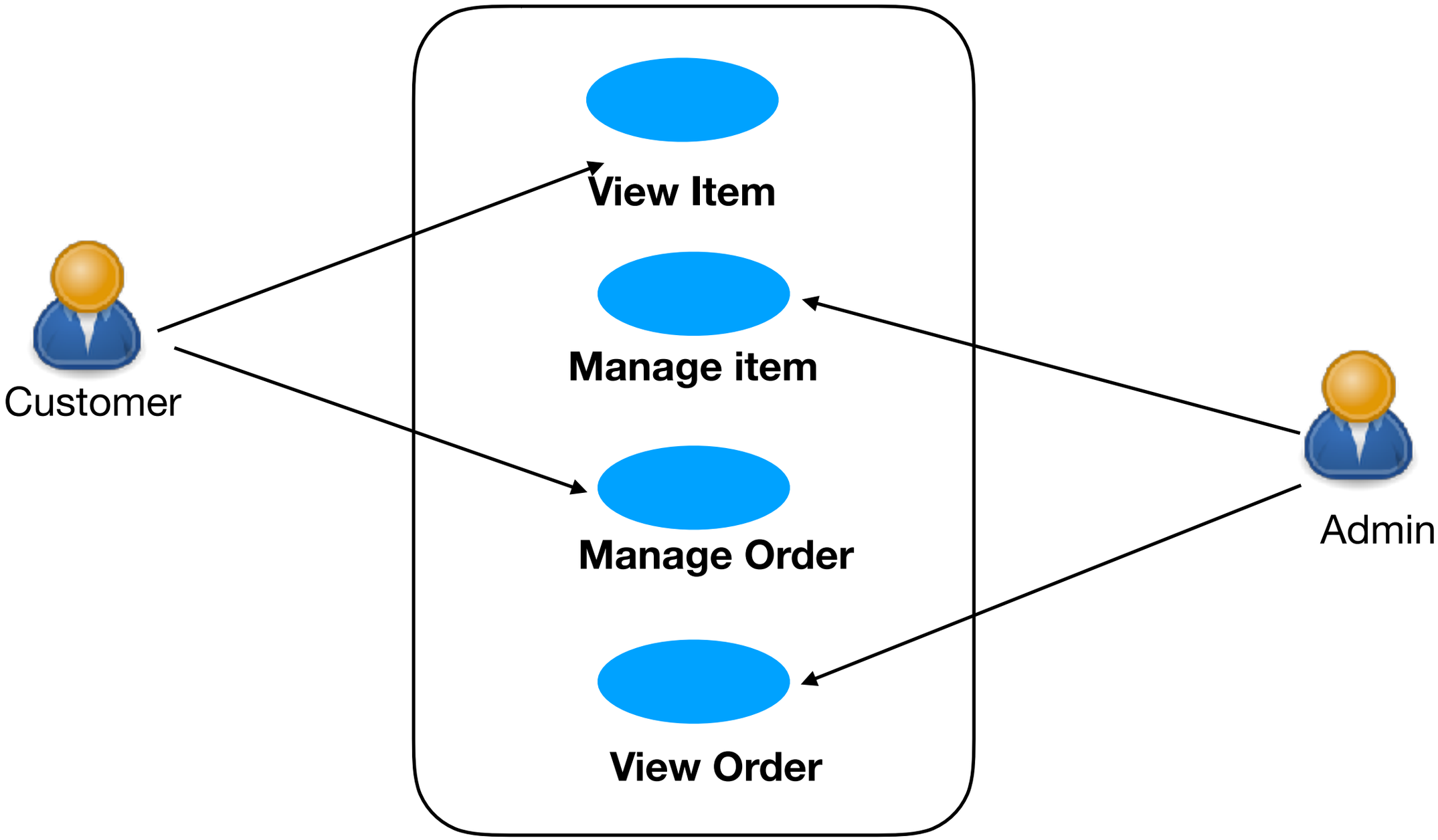
Which Quality ?

1. Maintainability <— —
2. Availability
3. Reliability (trust)
4. Security (trust)
5. Performance (includes scalability)
6. Robustness (rugud)

Arch Req

1. Context view

2. Functional view



3. Quality View

Quality Attribute Scenarios

Source (who)	Stimulus (action)	Artifact (which)	Environment (context)	Response (output)	Measure (scale)
As a user	I want to buy An item	From the portal	During peak load.	The order Confirmation page is displayed to the user	In < 3 secs
Consumer Web site	sent a purchase order request	to the XYZ Application	Duplicate request	not double-charged	In 100% of cases
a processor	stops working	in the “central system”	during peak traffic hours	Notify and start providing “degraded mode”	The time spent in degraded mode should be no more than 5 minutes

3. Quality requirement

Collect NFR

Source (who)	Trigger (action)	Artifact (module)	Environment (context)	Response (output)	Measure (metrics)
Develo per	Change the persistence	Portal	During maintenance	The data persistence mechanism is integrated	In < 1 month
Develo per	Change the UI	In the Portal	During maintenance	The UI technology is replaced	In < 15 days

4. Constraints

1	It should work on IE 7
2	Should work on windows 7
3	Should use Azure Cloud

5. Assumption

1	Will use open source Technologies only
2	
3	

Arch Approach

1. Principles

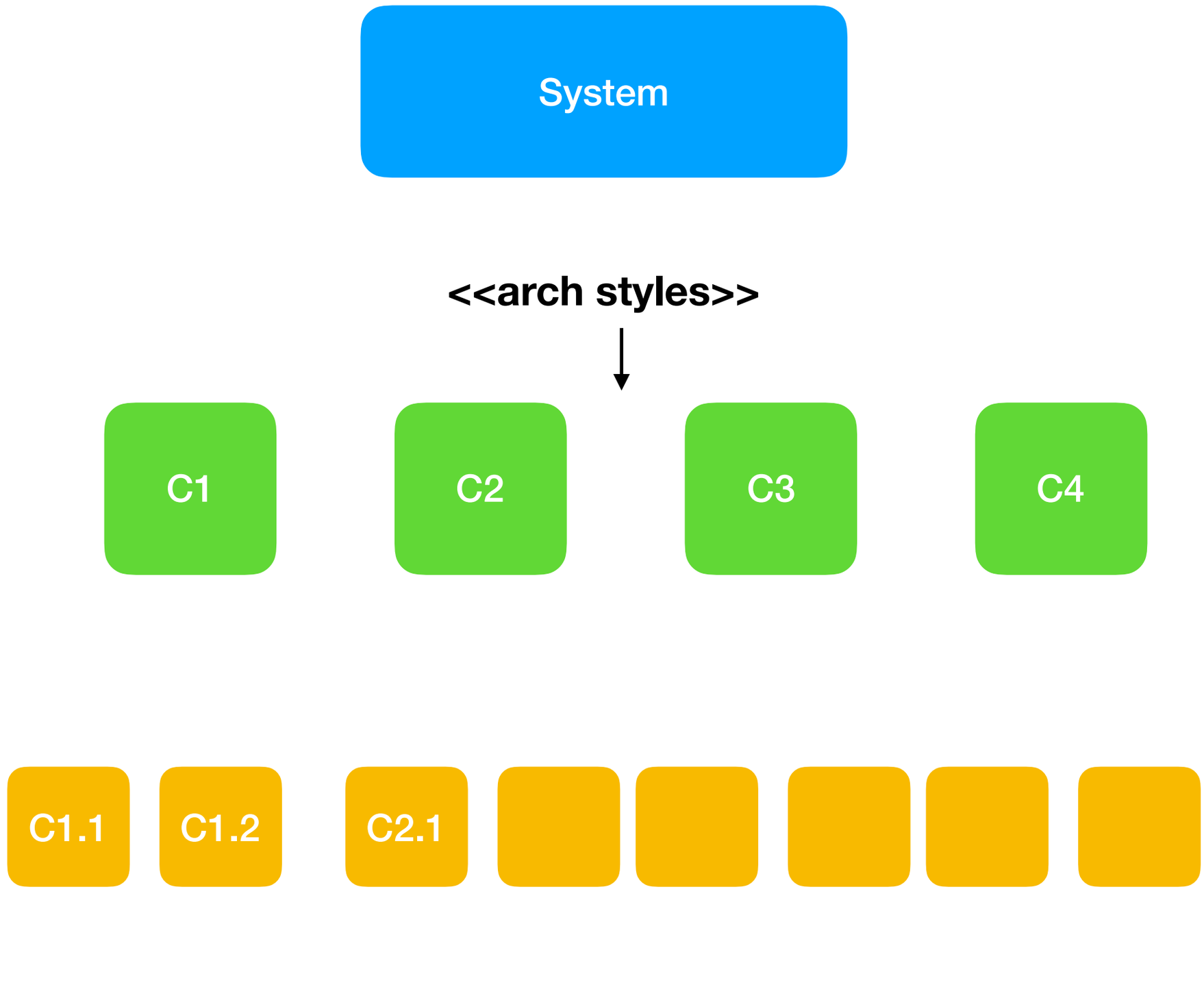
1	Seperation of Concerns
2	CQS - Command Query Seperation
3	KISS
4	YAGNI

2. Styles/Patterns

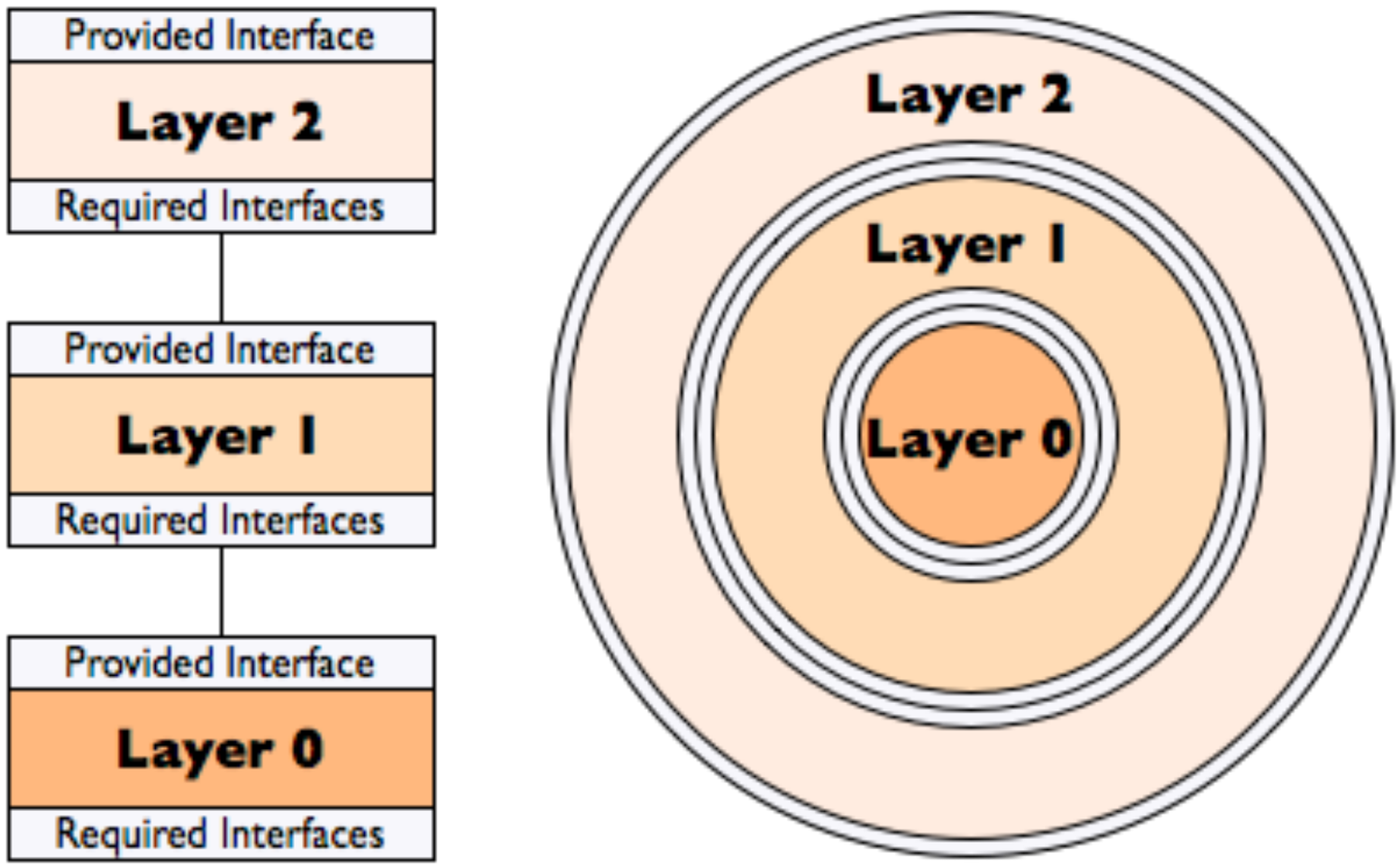
1	Hexagonal Arch/ Boundary Control Entity
2	Pipes and Filter
3	

Arch Design

Decompose the system

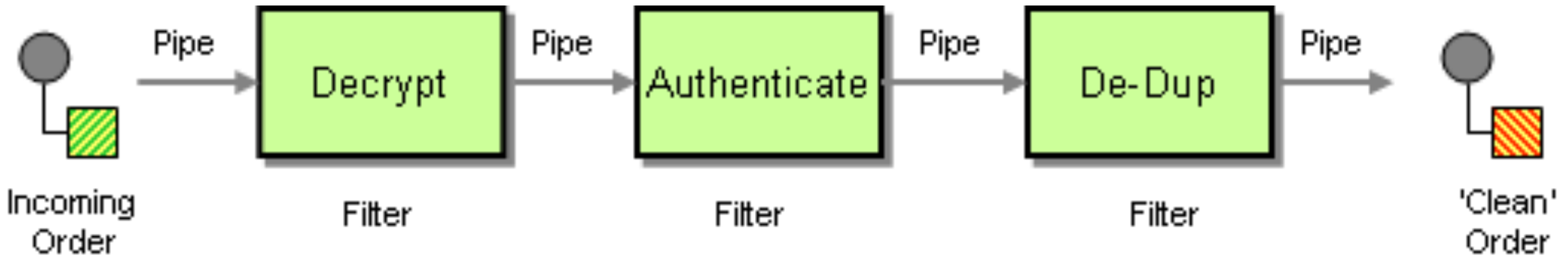


Layered Pattern



In a layered architecture, components are arranged in layers, ordered from lowest to highest, such that each layer only interacts with the one below it.

Pipes and Filter Pattern



Pipeline consists of a chain of processing elements , arranged so that the output of each element is the input of the next.

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

f1(int)

Can swap filters

Filter 3

f1(int)

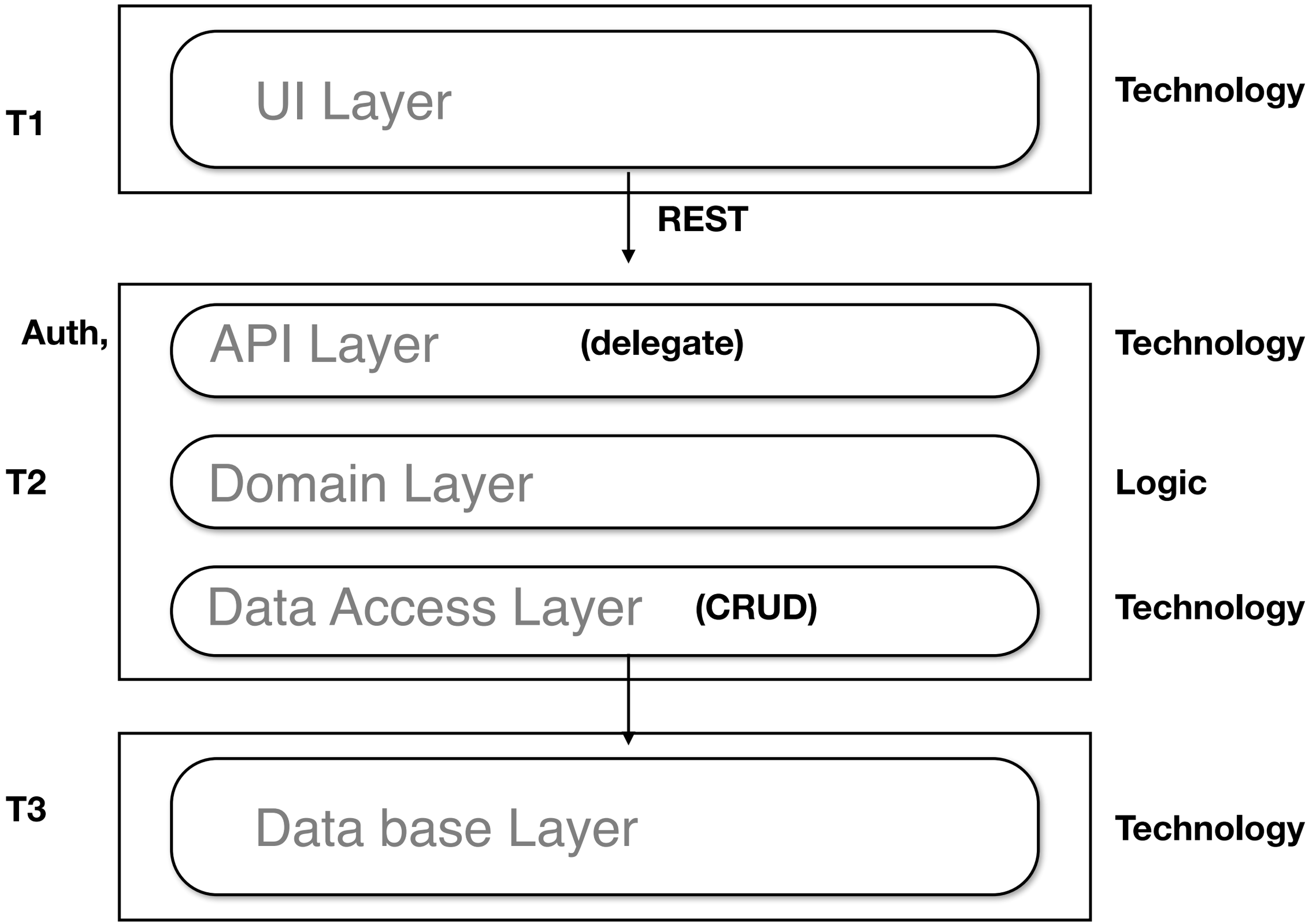
Filter 4

f1(int)

Data

Design #1

Logical View



Design #2

T1

validation, formatting, Routing, State mgmt

View Layer — **Technology**

Controller Layer — **Logic**

Model Layer — **Technology**

REST

API Layer — **Technology**

T2

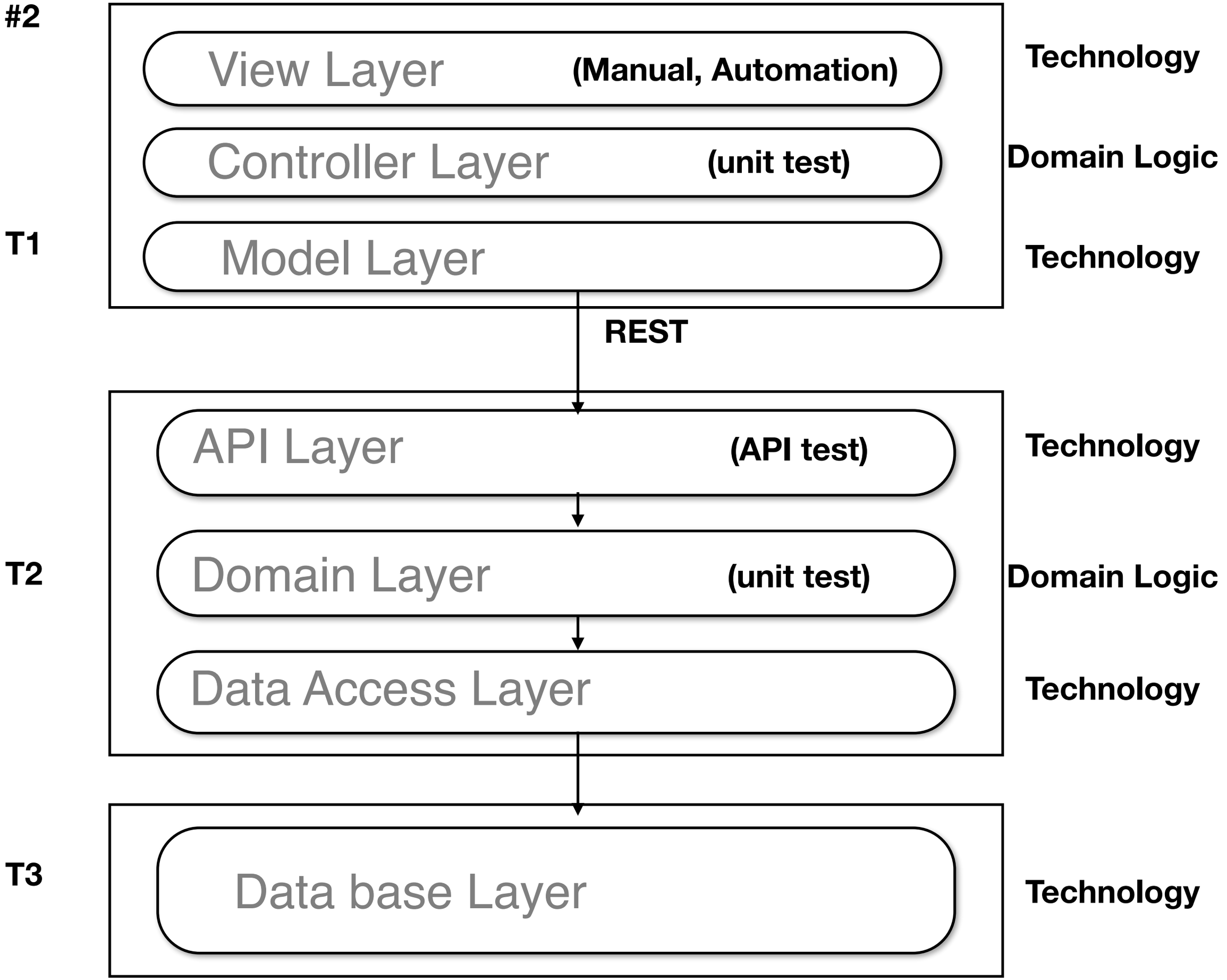
Domain Layer — **Logic**

Data Access Layer — **Technology**

T3

Data base Layer — **Technology**

Design #2


View Layer
(Manual, Automation)
Technology
Controller Layer
(unit test)
Domain Logic
T1
Model Layer
Technology
REST
API Layer
(API test)
Technology
T2
Domain Layer
(unit test)
Domain Logic
Data Access Layer
Technology
T3
Data base Layer
Technology

Design #3

T1

View Layer

Controller Layer

Model Layer

Technology

Domain Logic

Technology

REST

T2

API Layer (delegate)

1 to 1

Application Layer/ Facade Layer

(CRUD)
Data Access Layer

(logic)
Domain Layer

API Technology

Cross Cutting Concern Technology

Domain Logic

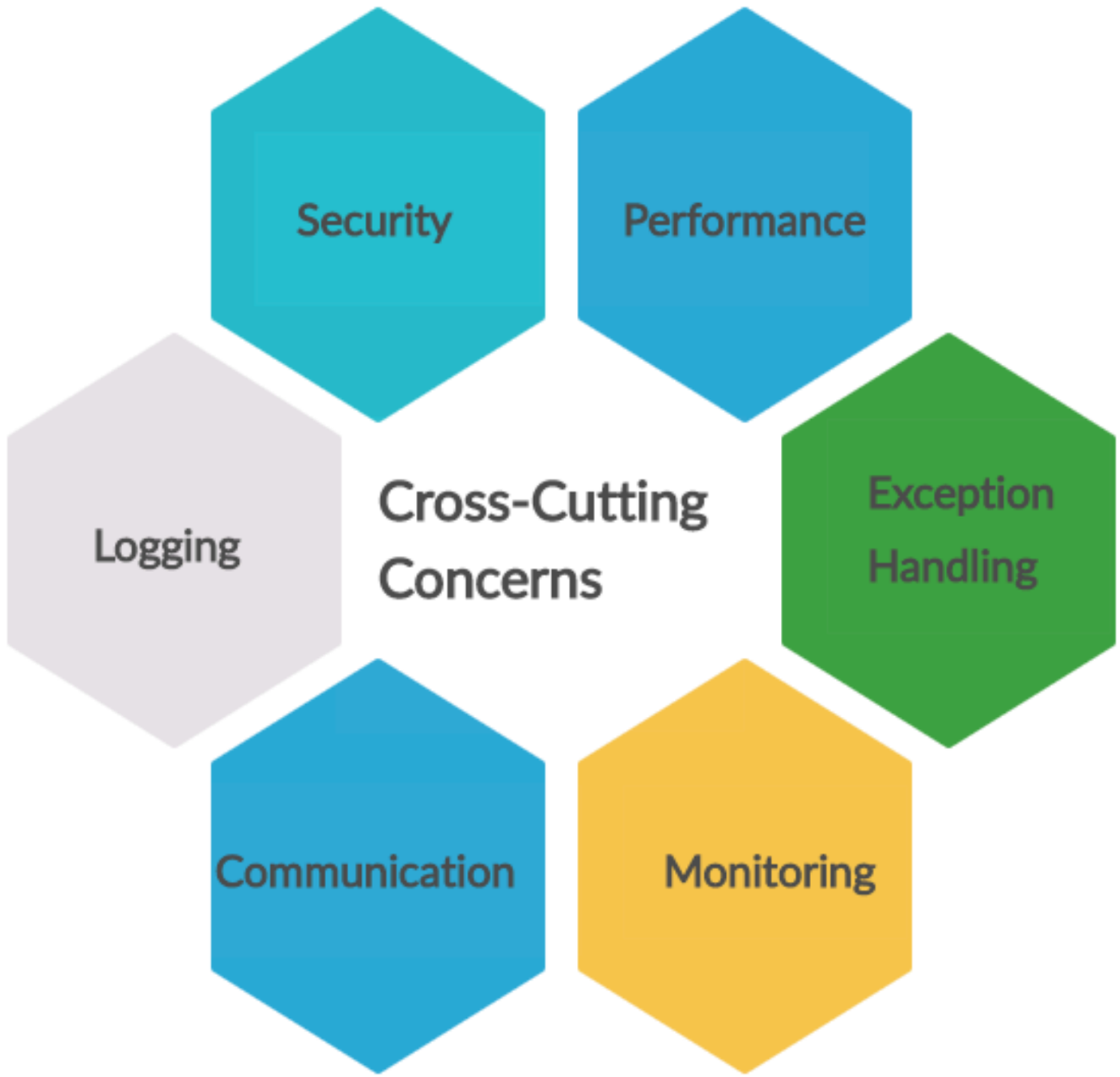
Technology

flow, log, Exception Handling, Authorization, Caching

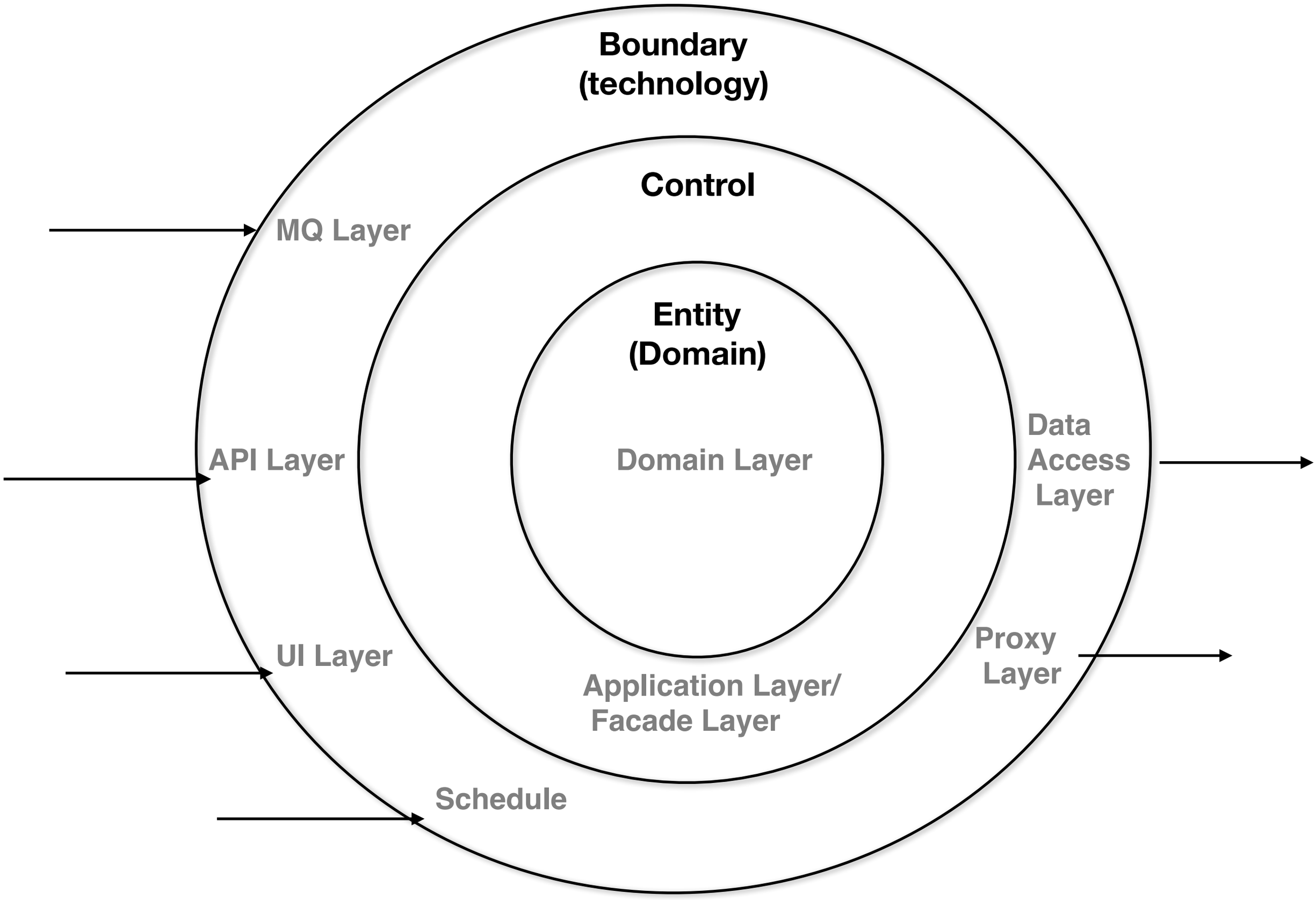
T3

Data base Layer

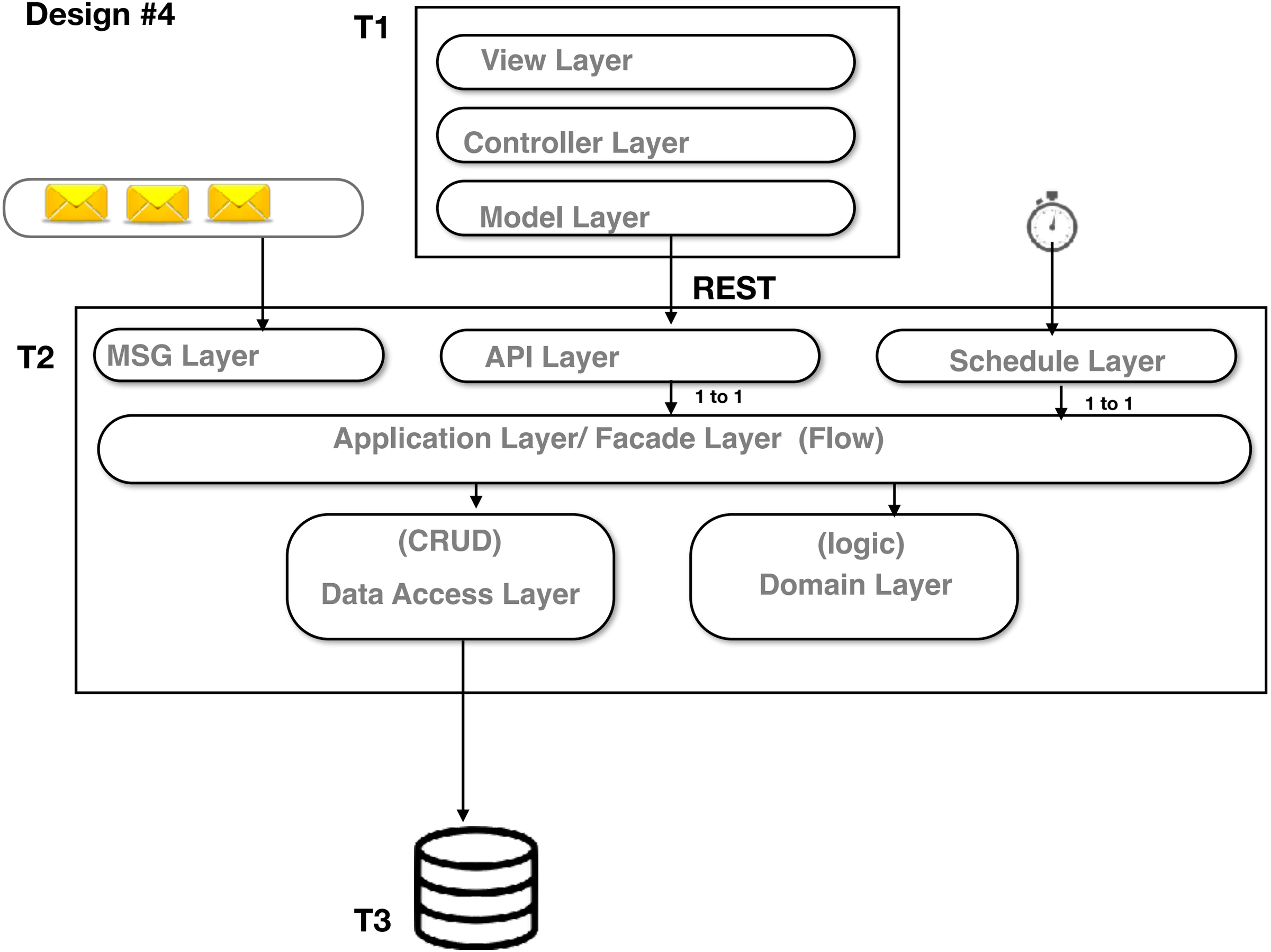
Technology


Security
Performance
Logging
Cross-Cutting Concerns
Exception Handling
Communication
Monitoring

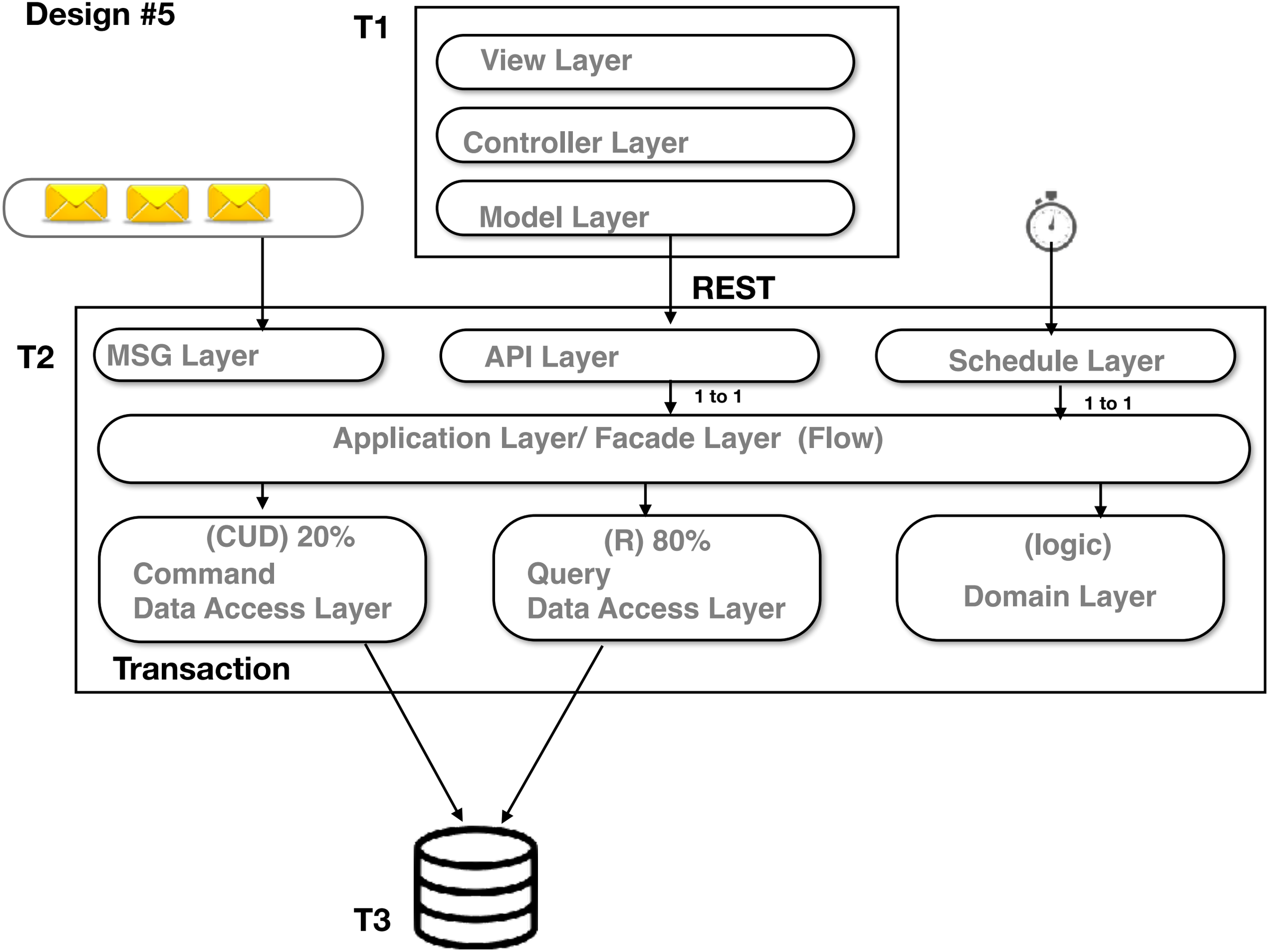
Hexagonal Arch



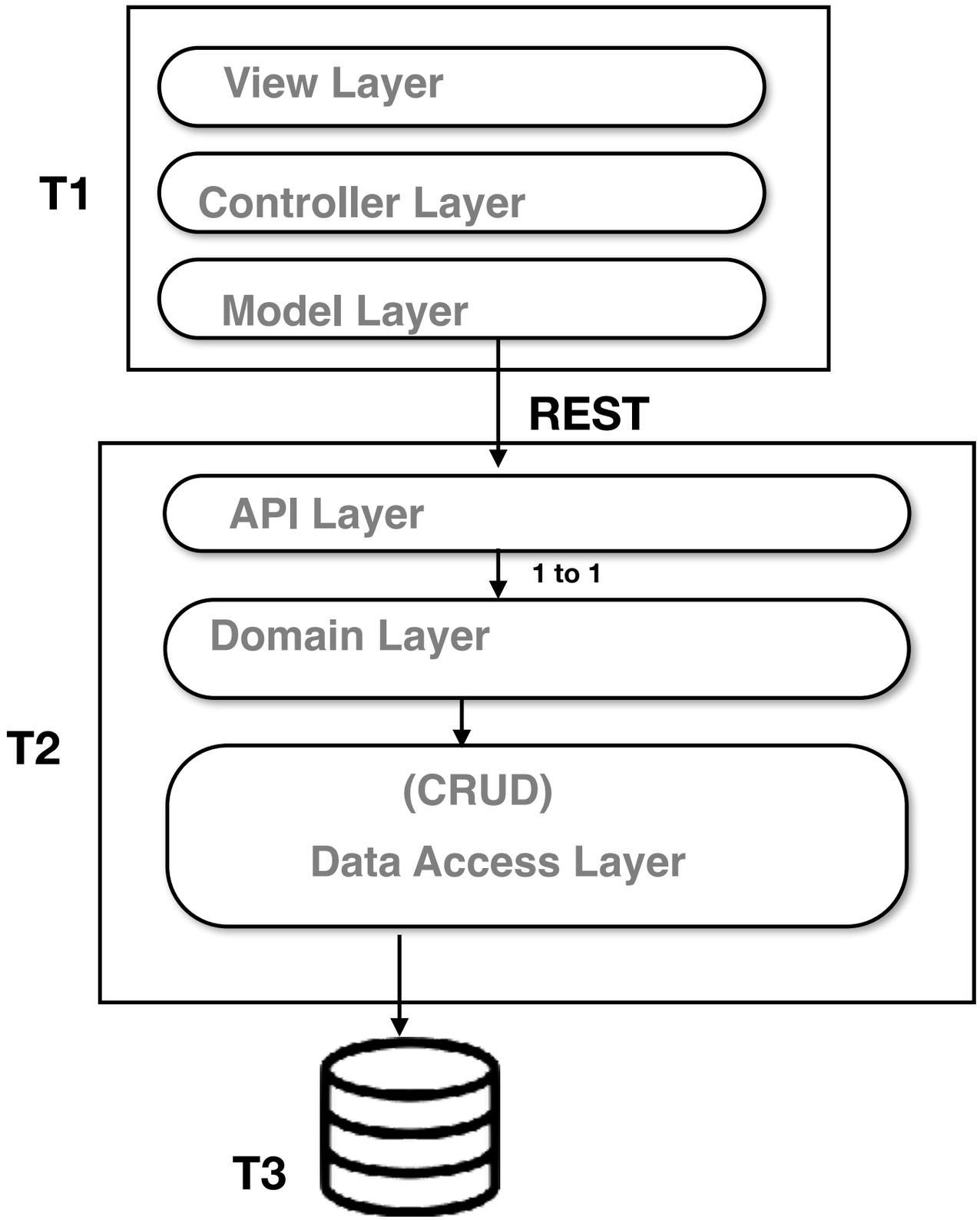
Design #4


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

Design #5


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)
(CUD) 20%
Command
Data Access Layer
(R) 80%
Query
Data Access Layer
(logic)
Domain Layer
Transaction
T3

6. Logical View



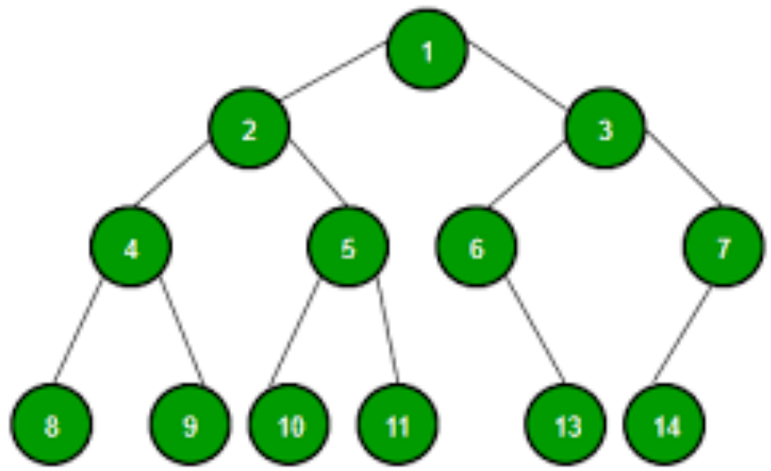
Persistence Matrix

File Storage	Rdbms	Document	Key-Valued	Graph	Column DB	Time Series	Distributed File Storage
Disk	MySQL Oracle Postgres	Mongo Dynamo	Redis Couch	Neo4j	HBase Cassandra	Influx TSDB	S3 Azure Blob
Binary	Referential Integrity	Hierarchy (Tree)	KEY = VALUE	Graph	Big Data		Binary
	No Duplicates	Tree queries	Performance	Graph queries	TB's	Time series queries	Multiple copies
	Transaction (ACID)			Transaction (ACID)			

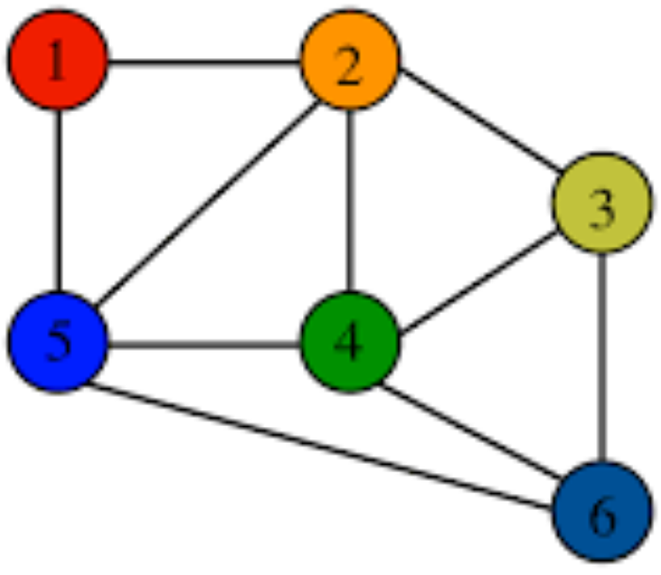
Rdbms (Referential Integrity)

tblPerson			
ID	Name	Email	GenderID
1	Jade	j@j.com	2
2	Mary	m@m.com	3
3	Martin	ma@ma.com	1
4	Rob	r@r.com	NULL
5	May	may@may.com	2
6	Kristy	k@k.com	NULL

tblGender	
ID	Gender
1	Male
2	Female
3	Unknown



K1 - > value1
K2 - > value2
K3 - > value3
K4 - > value4



Column based storage

Row id	Column	Value
1	First Name	Mickey
1	Last Name	Mouse
1	…	123 …
1	..	Anaheim

t1,t2,t3,t4, … t5,t6….

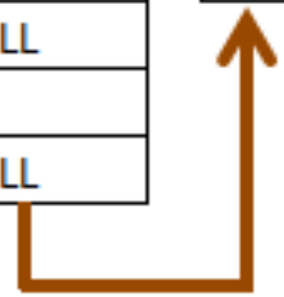
3NF

tblPerson

ID	Name	Email	GenderID
1	Jade	j@j.com	2
2	Mary	m@m.com	3
3	Martin	ma@ma.com	1
4	Rob	r@r.com	NULL
5	May	may@may.com	2
6	Kristy	k@k.com	NULL

tblGender

ID	Gender
1	Male
2	Female
3	Unknown



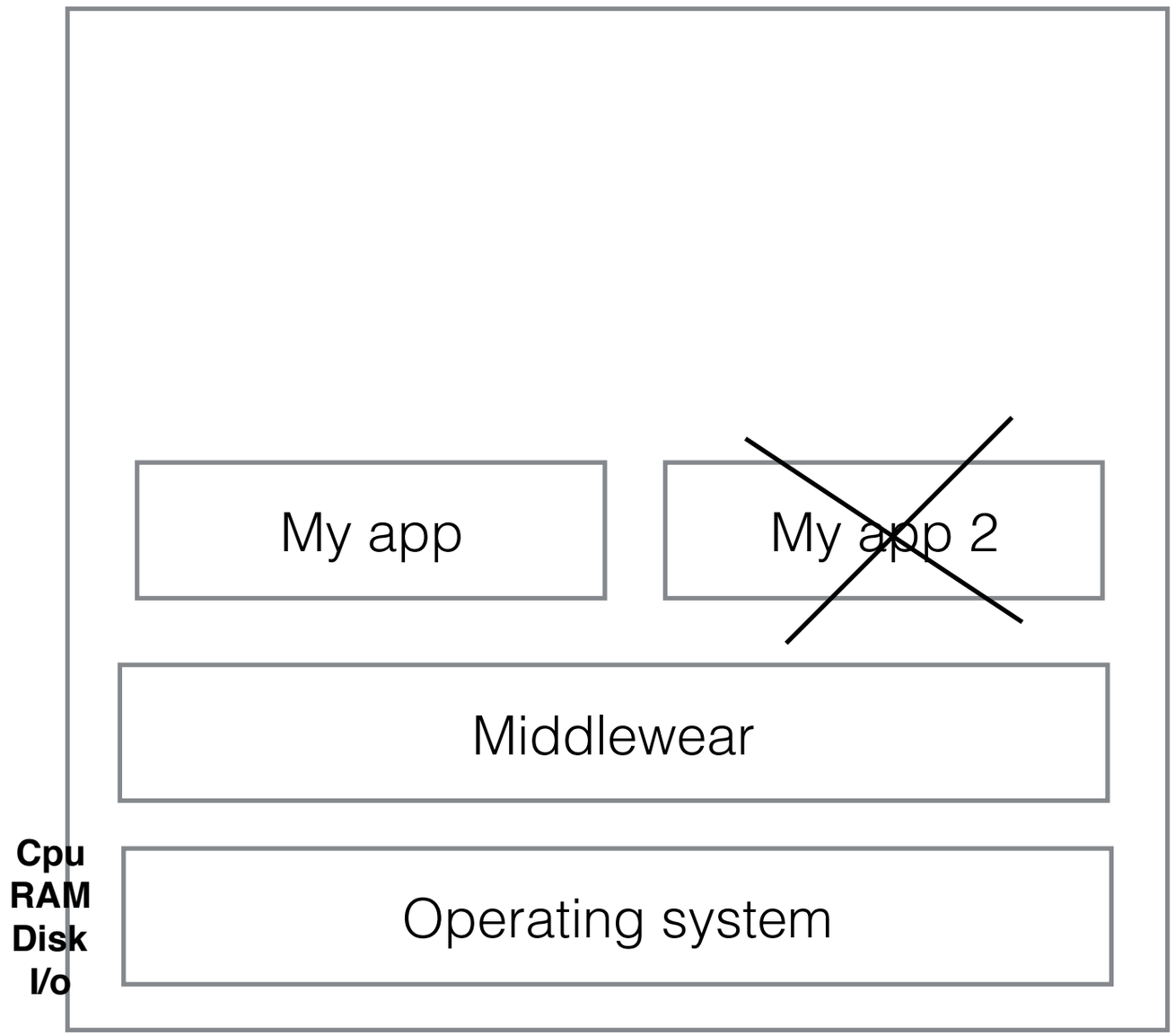
DNF

ID	Name	Email	Gender
			Female
			Unknown
			Male

Application Hosting Matrix

IAAS	Middlewear PAAS	Serverless Middlewear PAAS	Kubernetes PAAS	Serverless Kubernetes PAAS
VM (1 or *)	VM (1 or *) with Webserver	VM (1 or *) with Webserver	VM (1 or *) With Kuberenetes	VM (1 or *) With Kuberenetes
Charged Base on count of VMS provisioned	Charged Base on count of VMS provisioned	Charged Based on execution (cpu, memory, io ,…)	Charged Base on count of VMS provisioned	Charged Based on execution (cpu, memory, io ,
Win Unix …	Tomcat, Jboss, Nginx, …			
Ec2	Beanstalk	aws Lamda	aws	aws
Azure VM	Azure App service	Azure functions	Azure	Azure

Design #1



Flask + apache
Flask + gunicorn,
tomcat,
jboss,
Iis,
Express + nodejs,

Design #2

VM1

VM2

My app 1

My app 3

My app 2

Middlewear

Middlewear

Unix Guest Os

Win Guest Os

Cpu
RAM
Disk
I/o

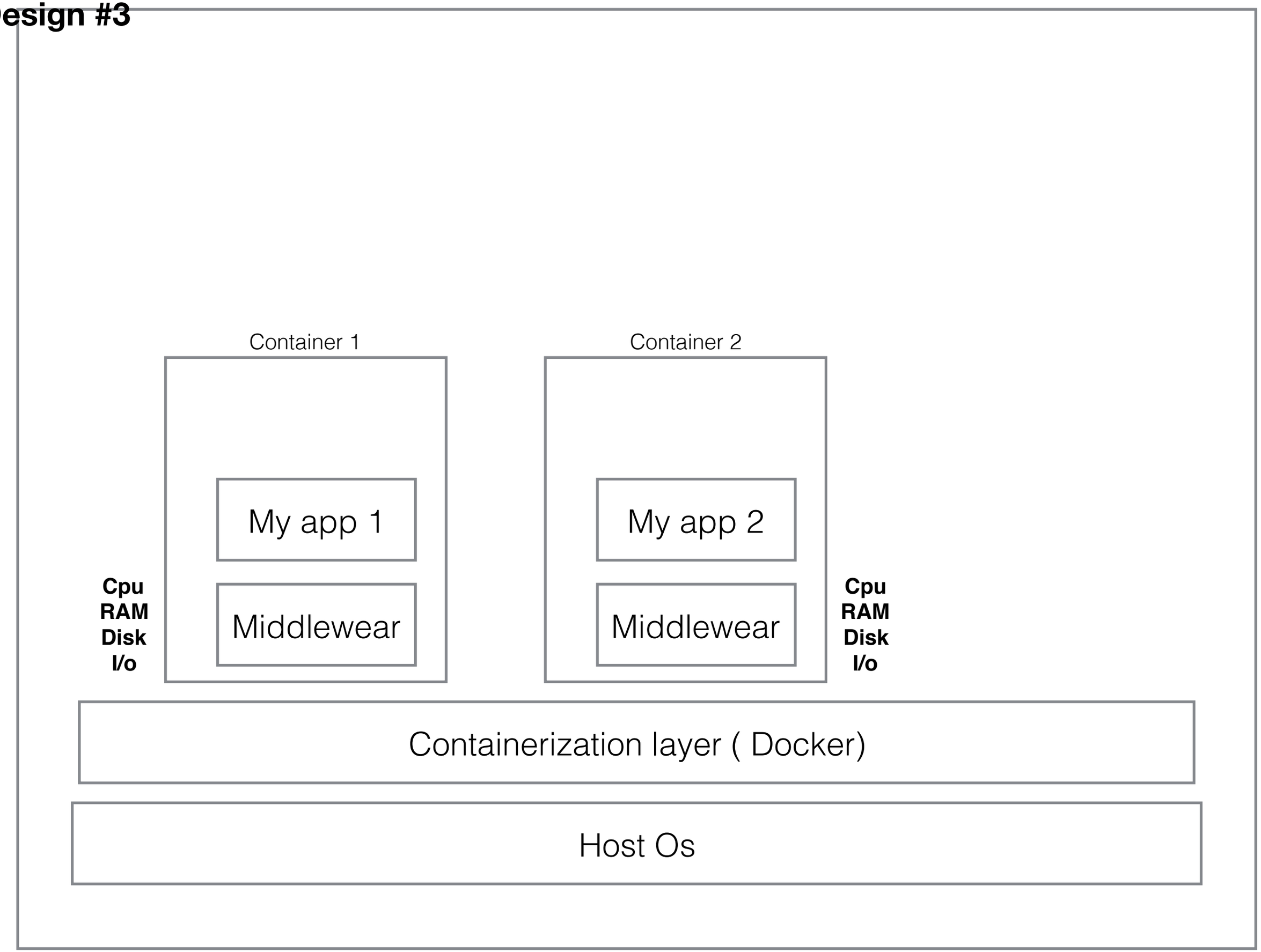
Cpu
RAM
Disk
I/o

Virtualisation layer

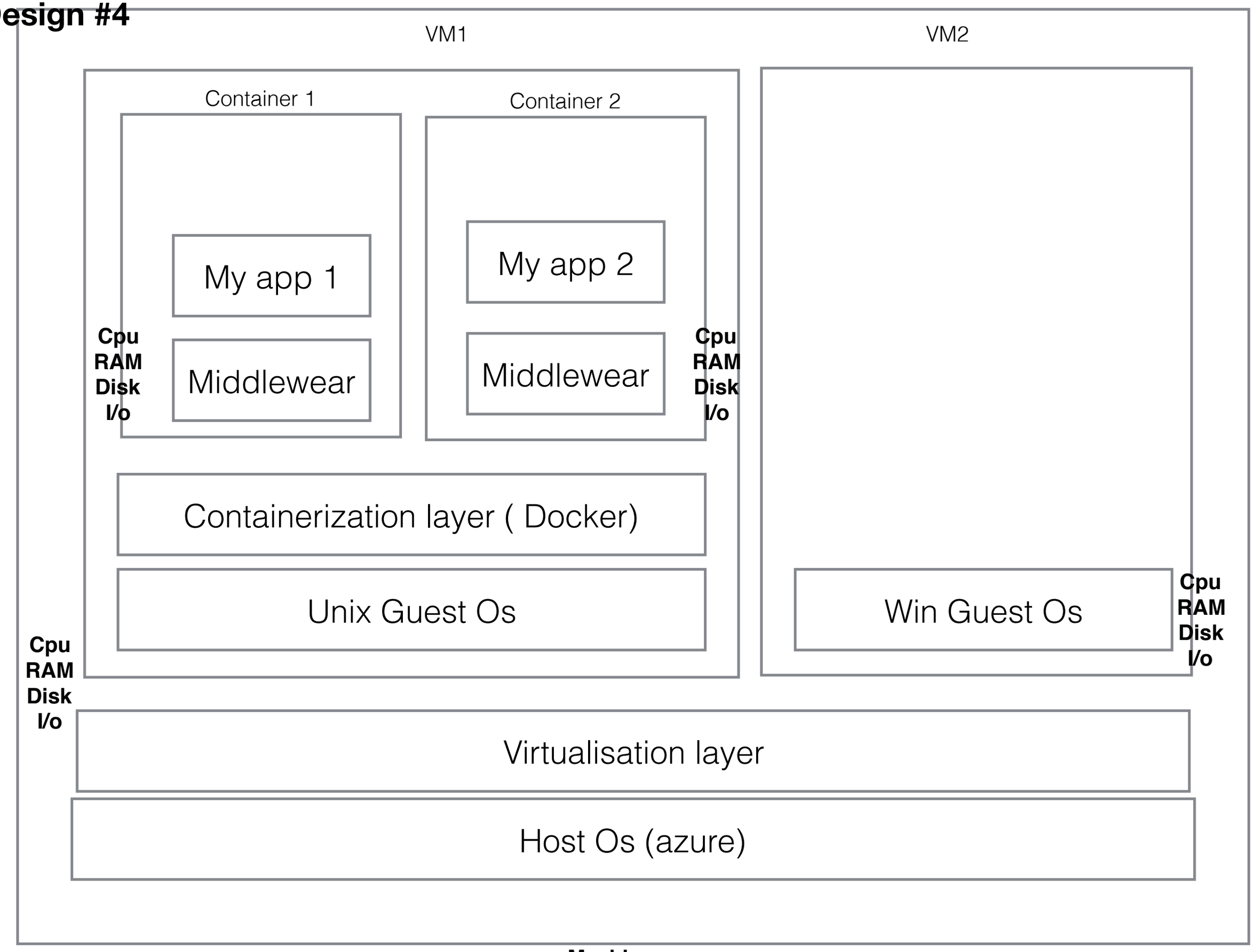
Host Os

Node

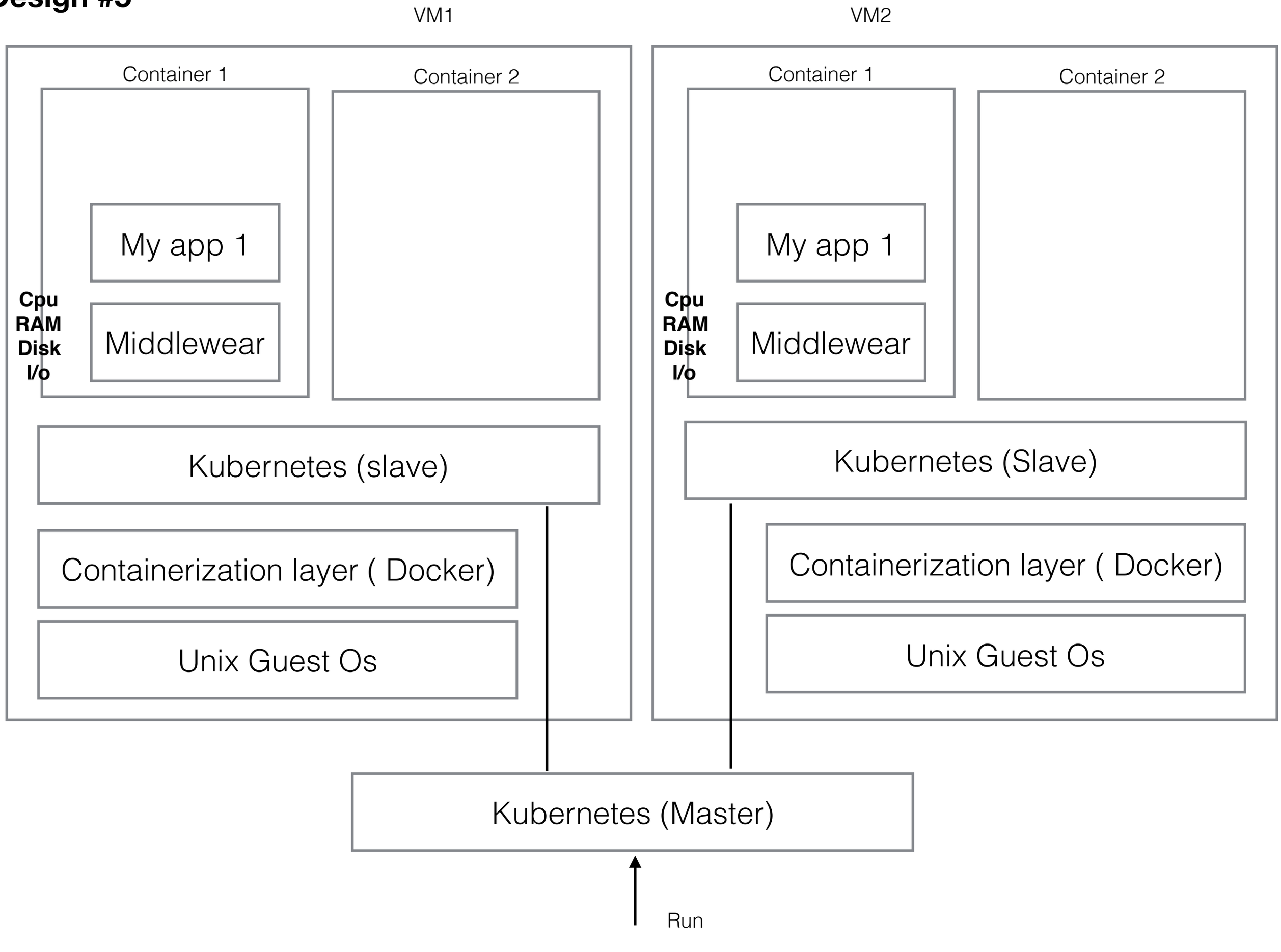
Design #3

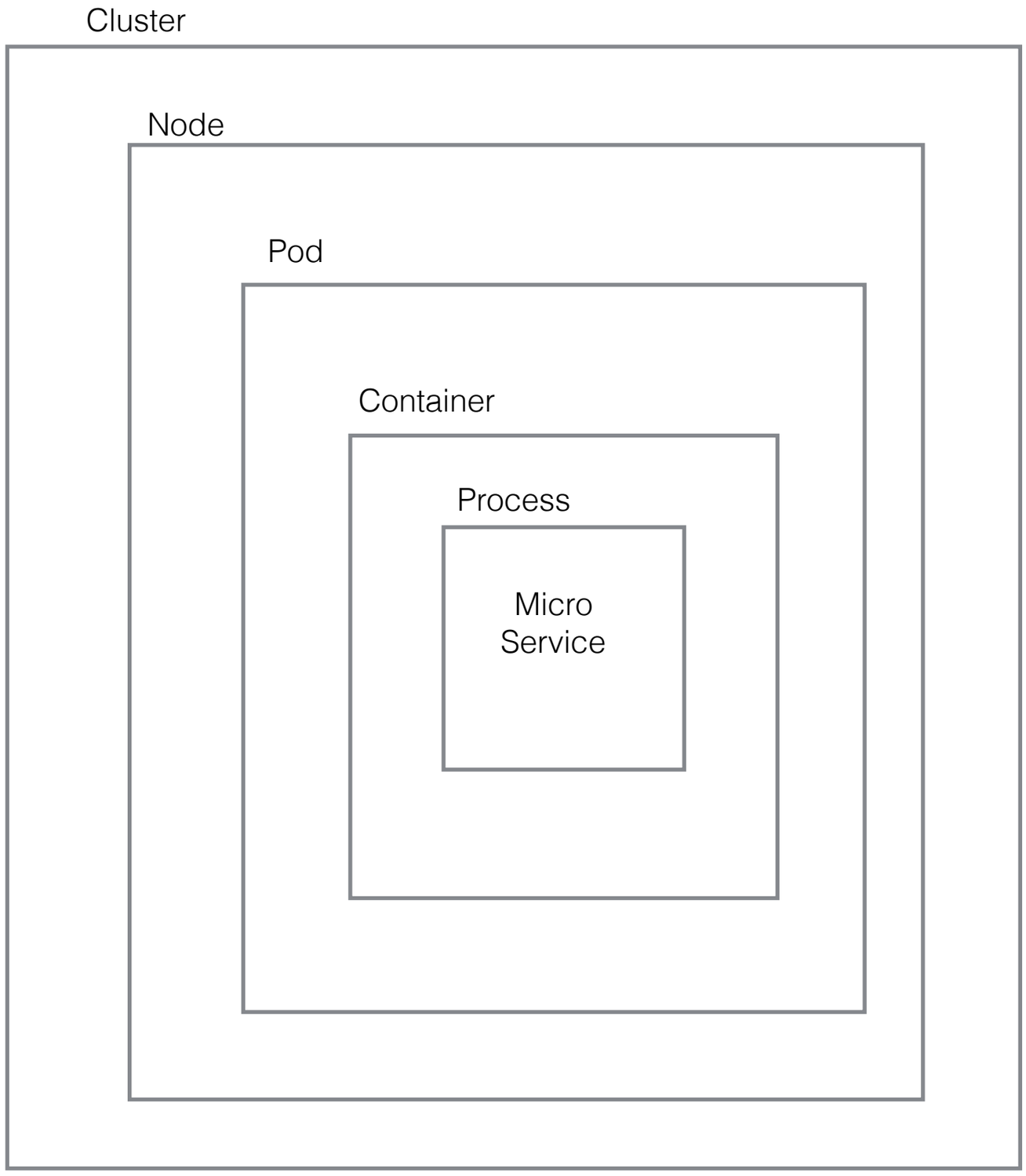

Container 1
My app 1
Middlewear
Container 2
My app 2
Middlewear
Cpu
RAM
Disk
I/o
Cpu
RAM
Disk
I/o
Containerization layer (Docker)
Host Os
Node

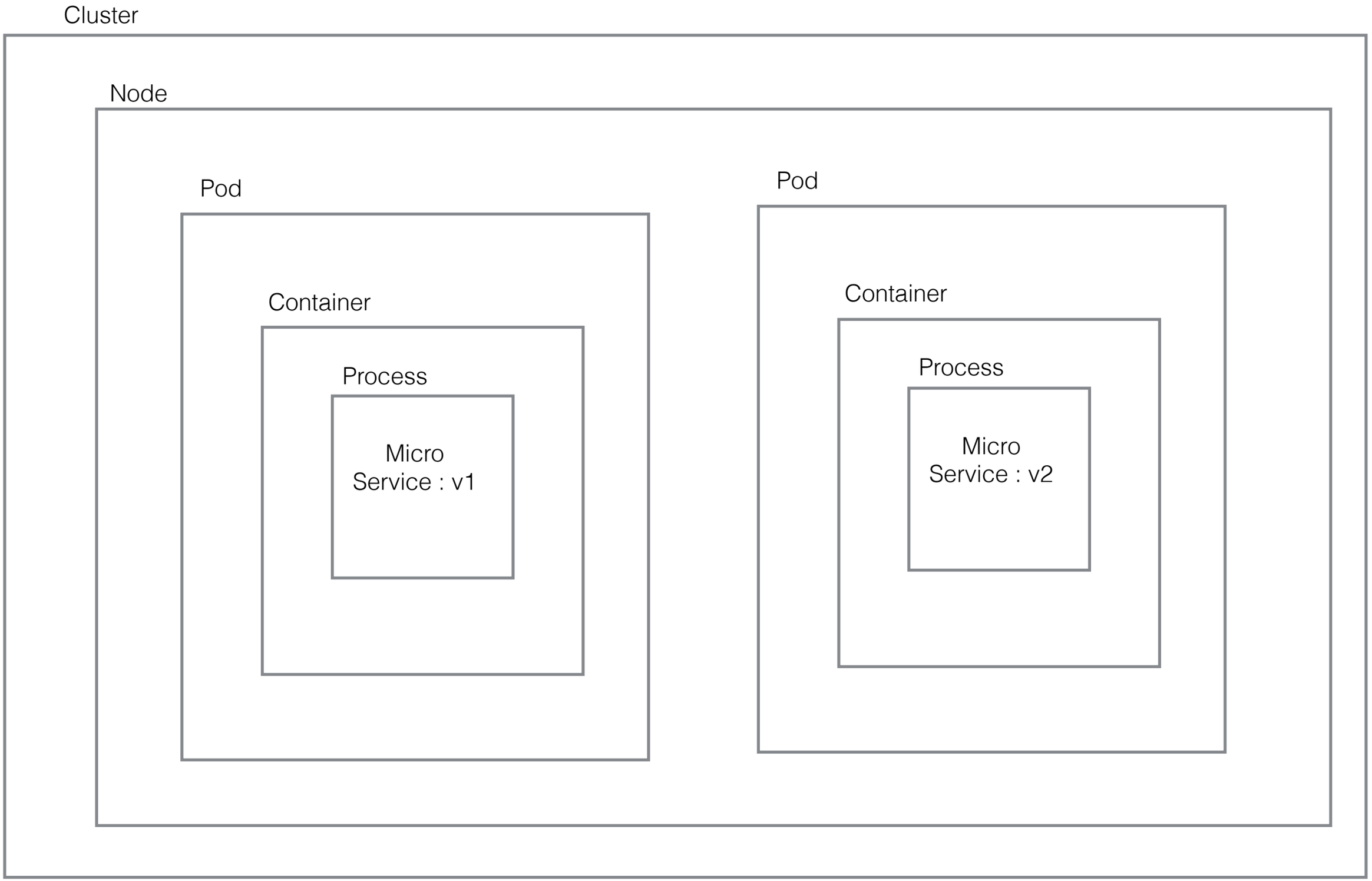
Design #4

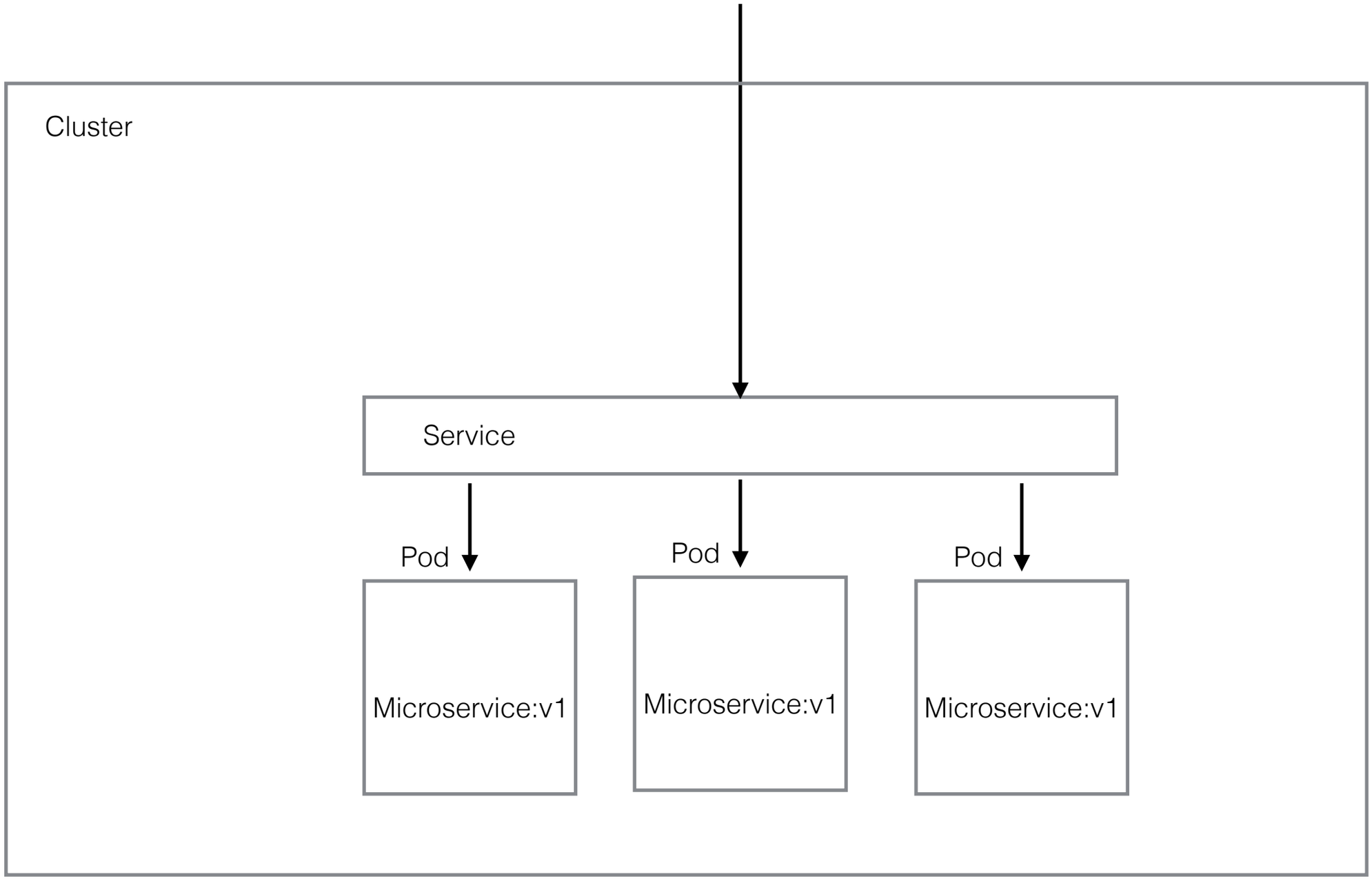

VM1
VM2
Container 1
Container 2
My app 1
My app 2
Middlewear
Middlewear
Cpu
RAM
Disk
I/o
Cpu
RAM
Disk
I/o
Containerization layer (Docker)
Unix Guest Os
Win Guest Os
Cpu
RAM
Disk
I/o
Cpu
RAM
Disk
I/o
Virtualisation layer
Host Os (azure)
Machine

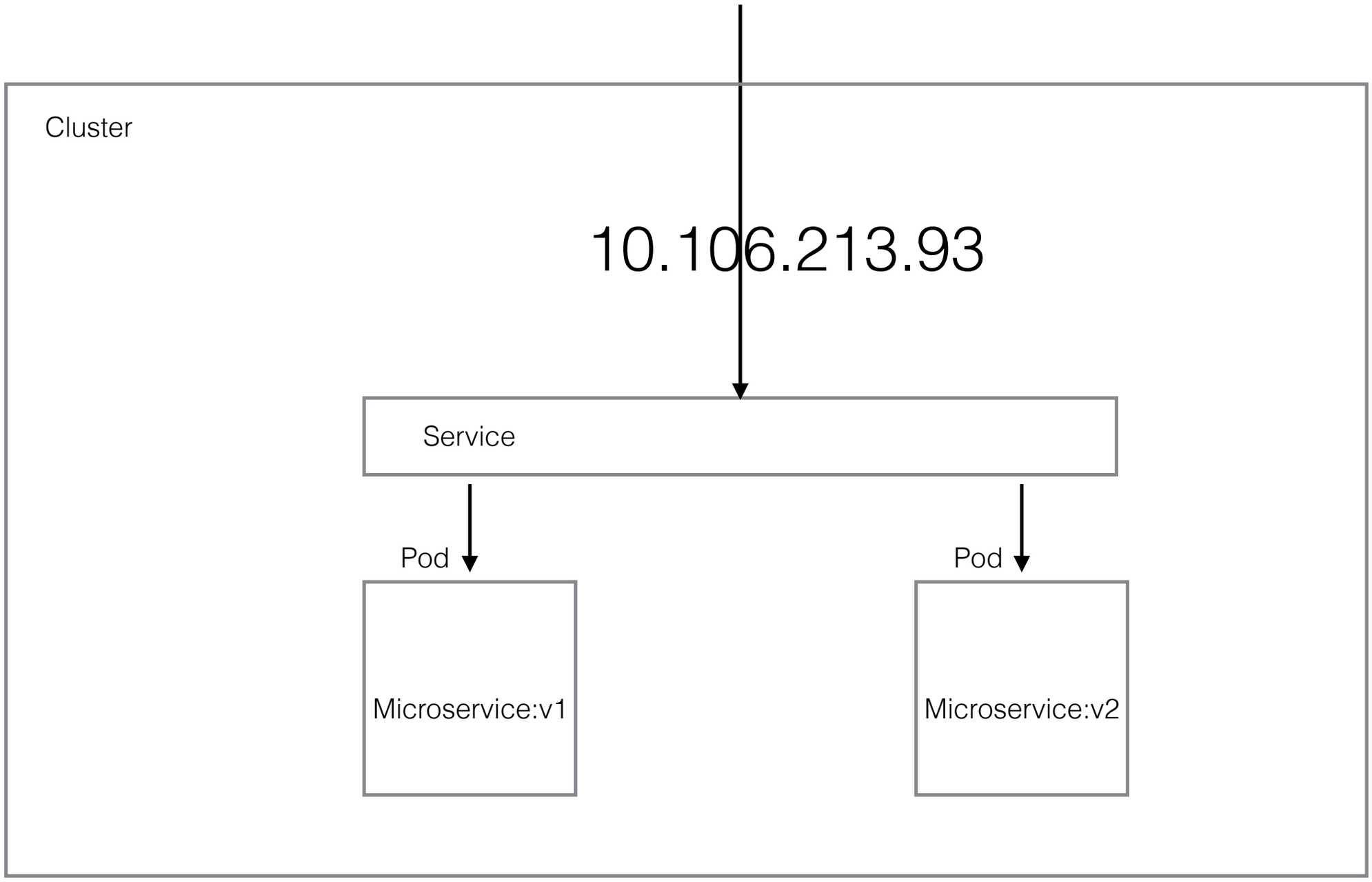
Design #5


VM1
VM2
Container 1
Container 2
My app 1
Middlewear
Cpu
RAM
Disk
I/o
Kubernetes (slave)
Containerization layer (Docker)
Unix Guest Os
Kubernetes (Slave)
Kubernetes (Master)
Run


Cluster
Node
Pod
Container
Process
Micro
Service


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

XX.XX.XX.XX

Pod

Microservice B

XX.XX.XX.XX

Pod

Microservice B

10.106.213.93

Service A

XX.XX.XX.XX

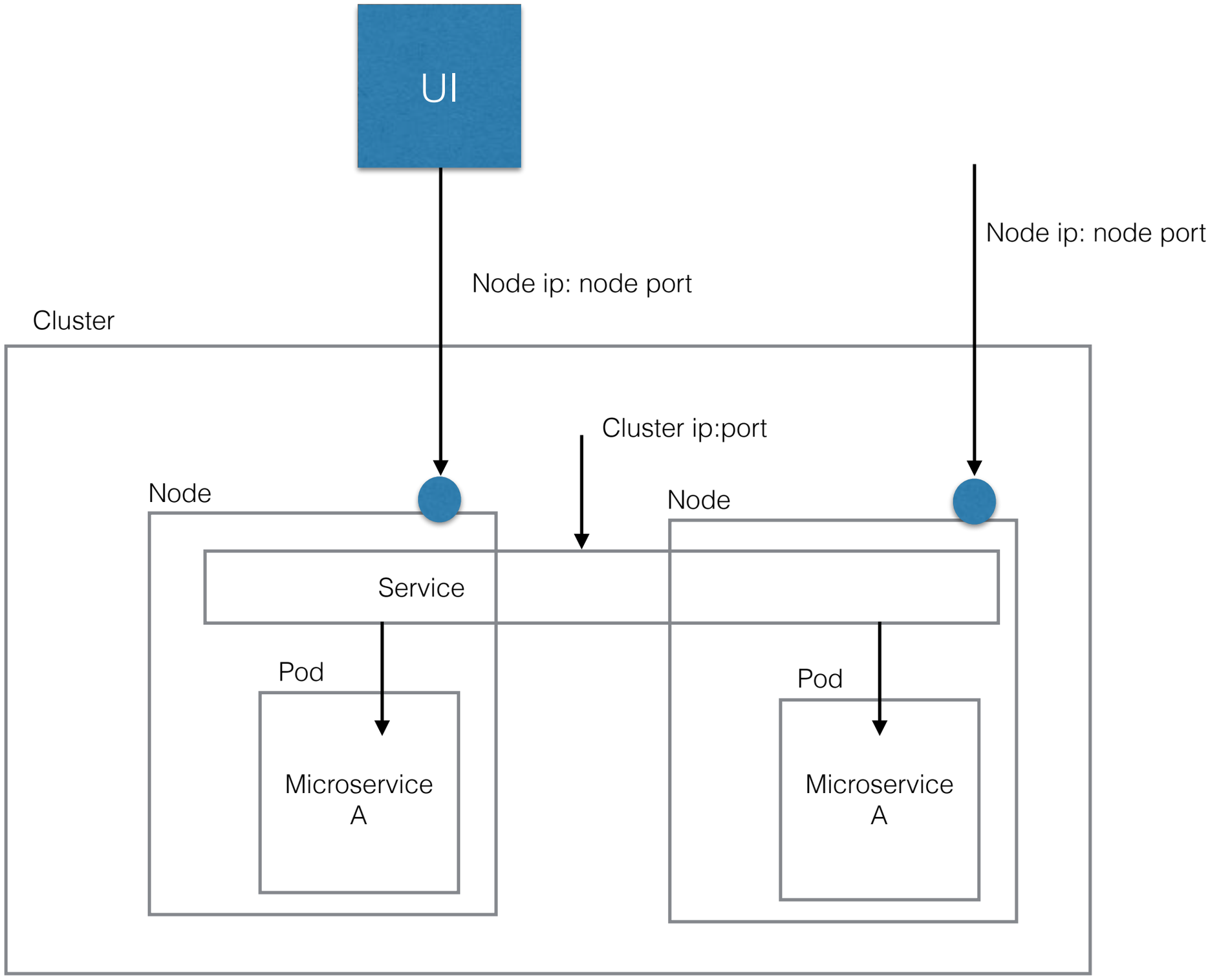
Pod

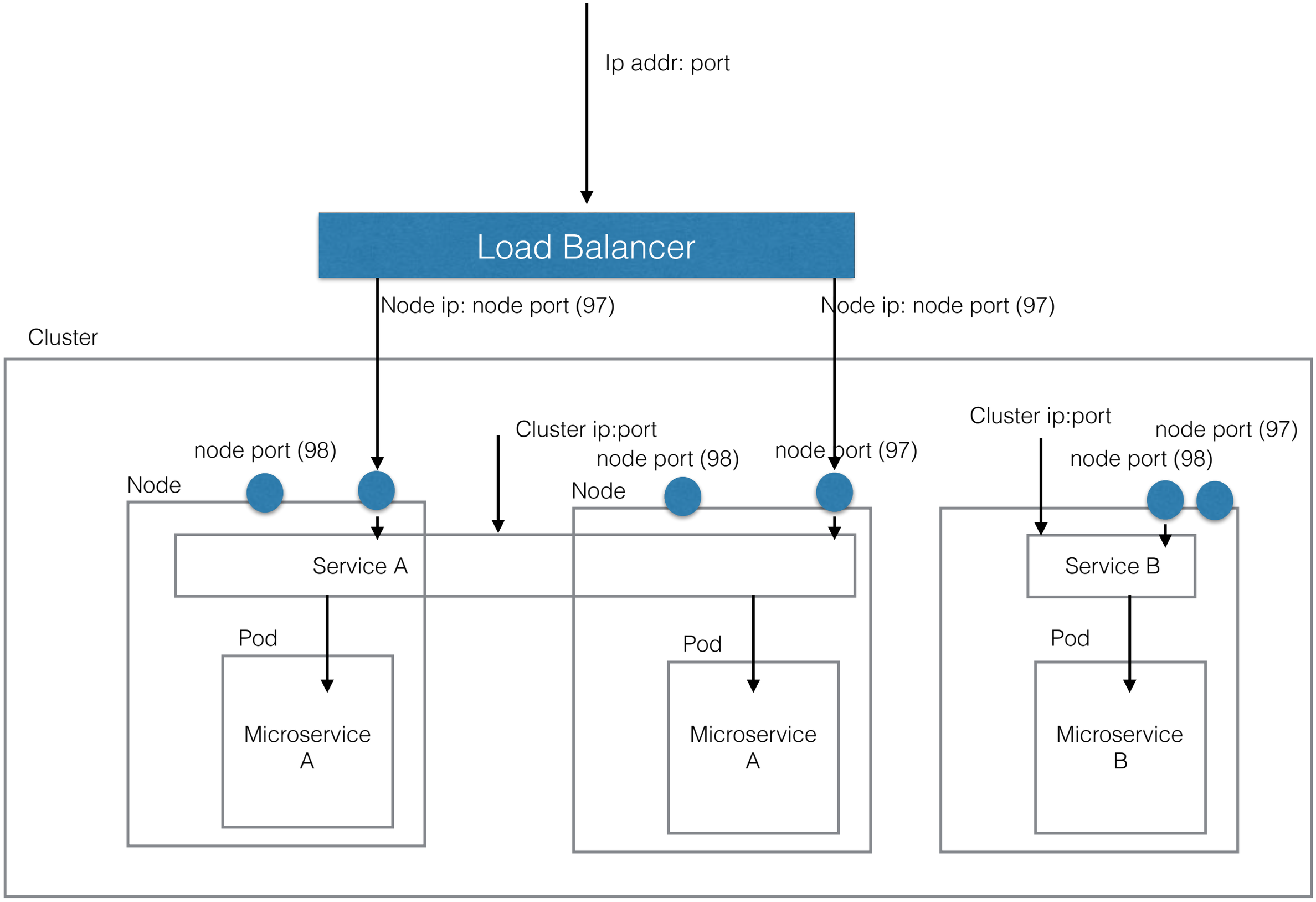
Microservice A

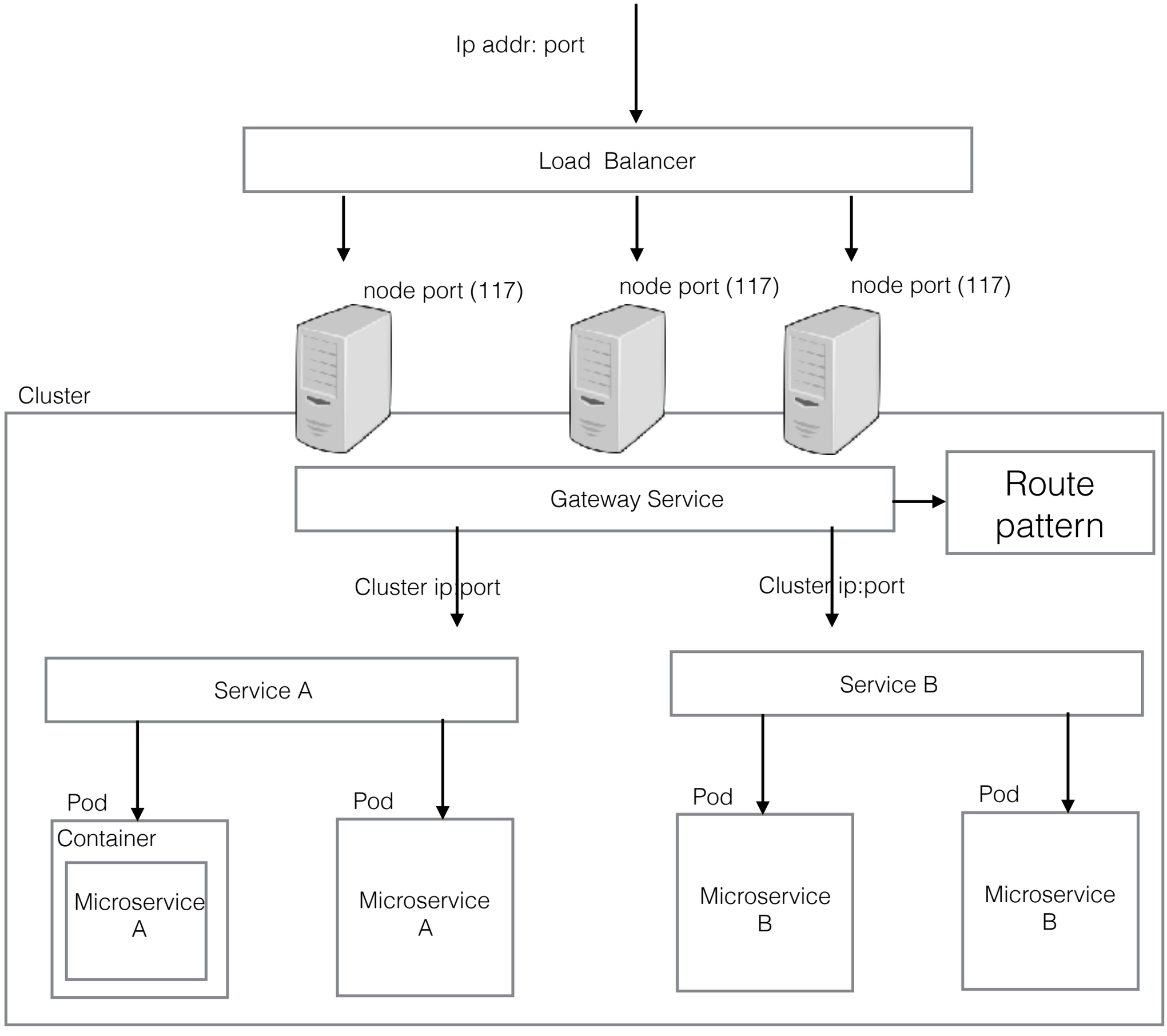
XX.XX.XX.XX

Pod

Microservice A


UI
Node ip: node port
Node ip: node port
Cluster
Cluster ip:port
Node
Node
Service
Pod
Pod
Microservice
A
Microservice
A


Ip addr: port
Load Balancer
Node ip: node port (97)
Node ip: node port (97)
Cluster
node port (98)
Node
Service A
Pod
Microservice
A
Cluster ip:port
node port (98)
Node
Pod
Microservice
A
node port (97)
Cluster ip:port
node port (97)
node port (98)
Service B
Pod
Microservice
B


Ip addr: port
Load Balancer
node port (117)
node port (117)
node port (117)
Cluster
Gateway Service
Route pattern
Cluster ip:port
Cluster ip:port
Service A
Service B
Pod
Container
Microservice A
Pod
Microservice A
Pod
Microservice B
Pod
Microservice B

Kubernetes Demo

My app

OS

Machine

My app

Middlewear

OS

Machine

My app

Middlewear

Guest OS

VM

Machine

My app

Middlewear

Container

Guest OS

VM

Machine

Kubernetes

Cluster

My app

Middlewear

Container

Guest OS

VM

Machine

My app

Middlewear

Container

Guest OS

VM

Machine

Kubernetes

Cluster

My app

Middlewear

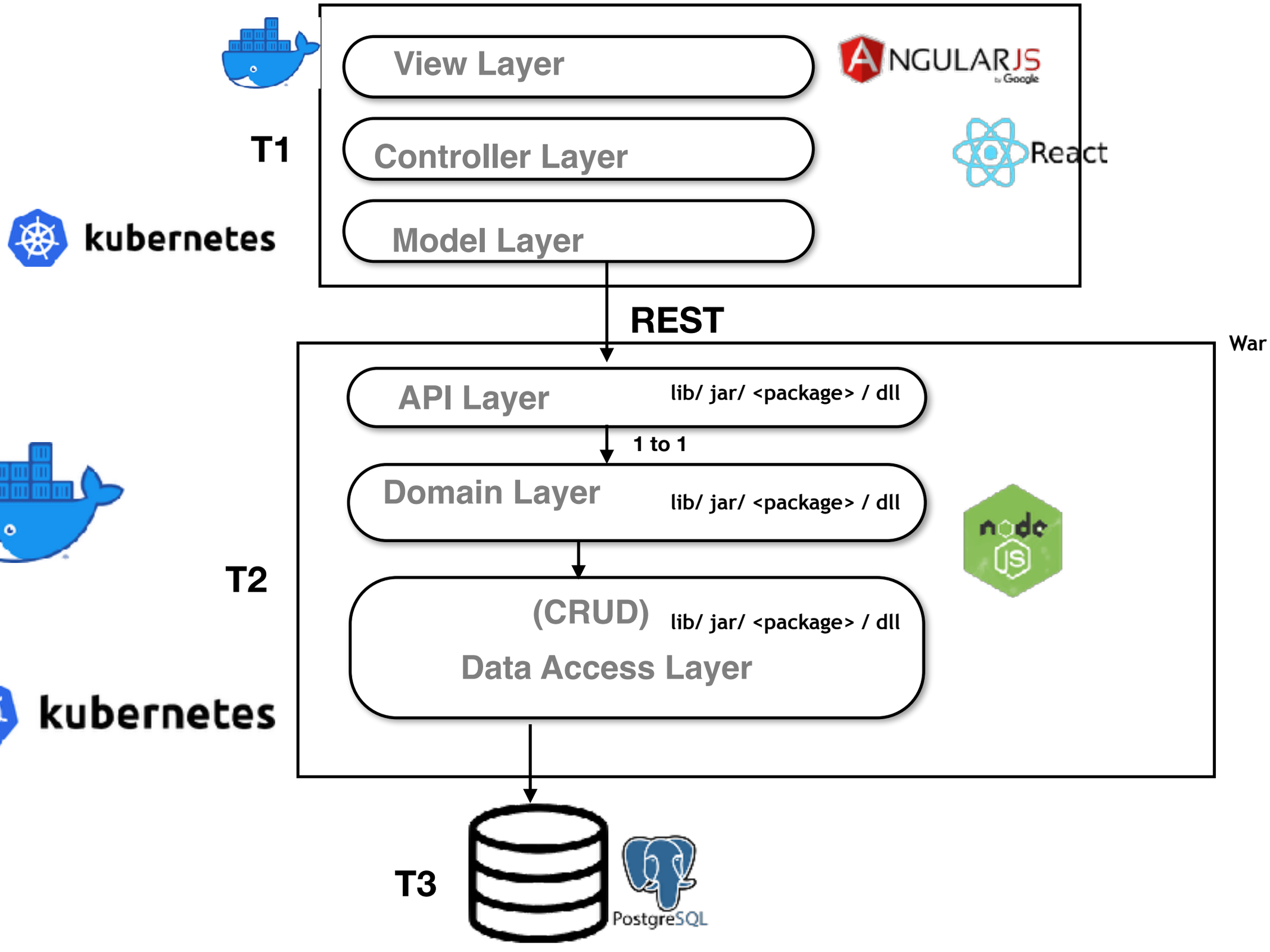
Guest OS

VM

Machine

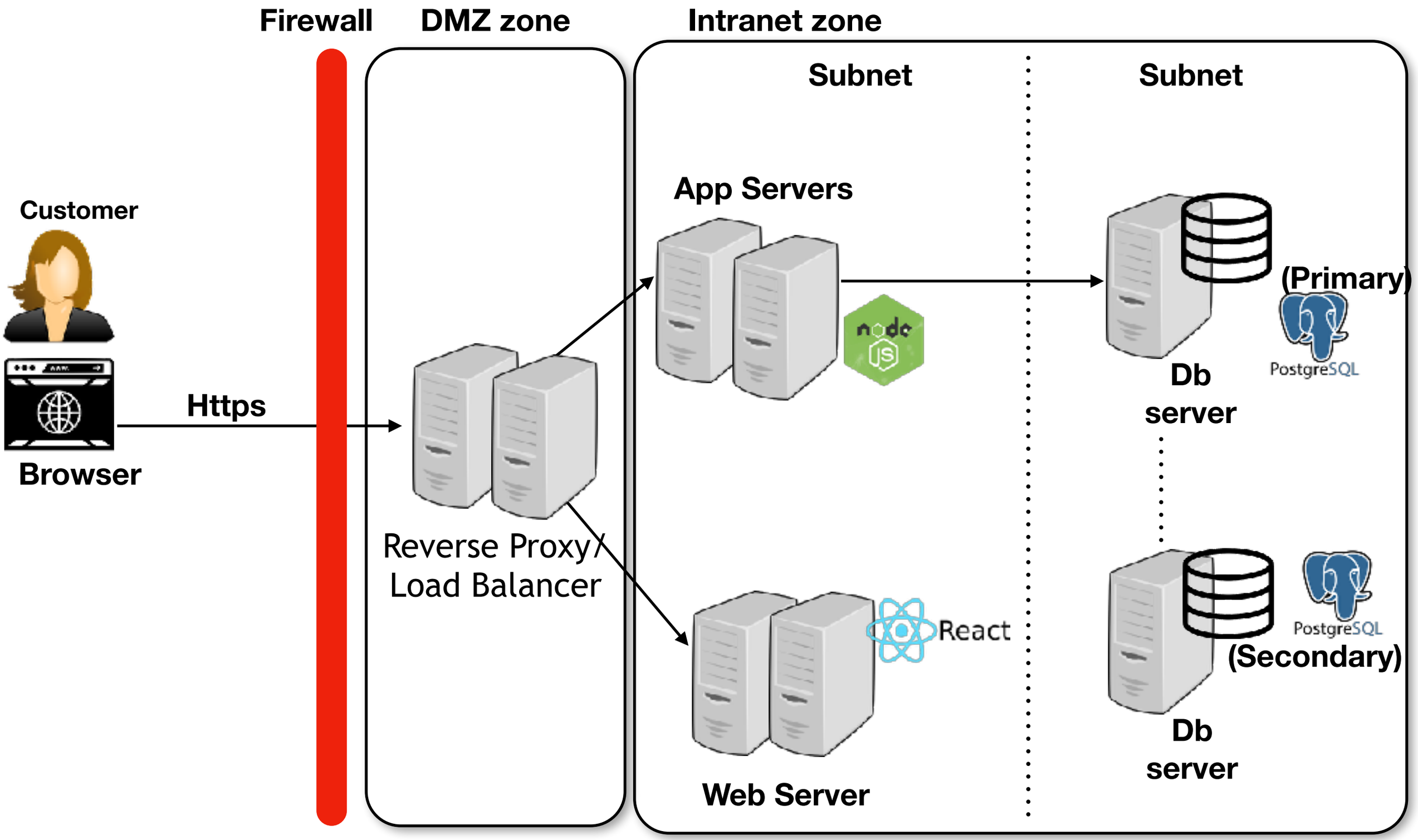
Cluster

7. Technology view



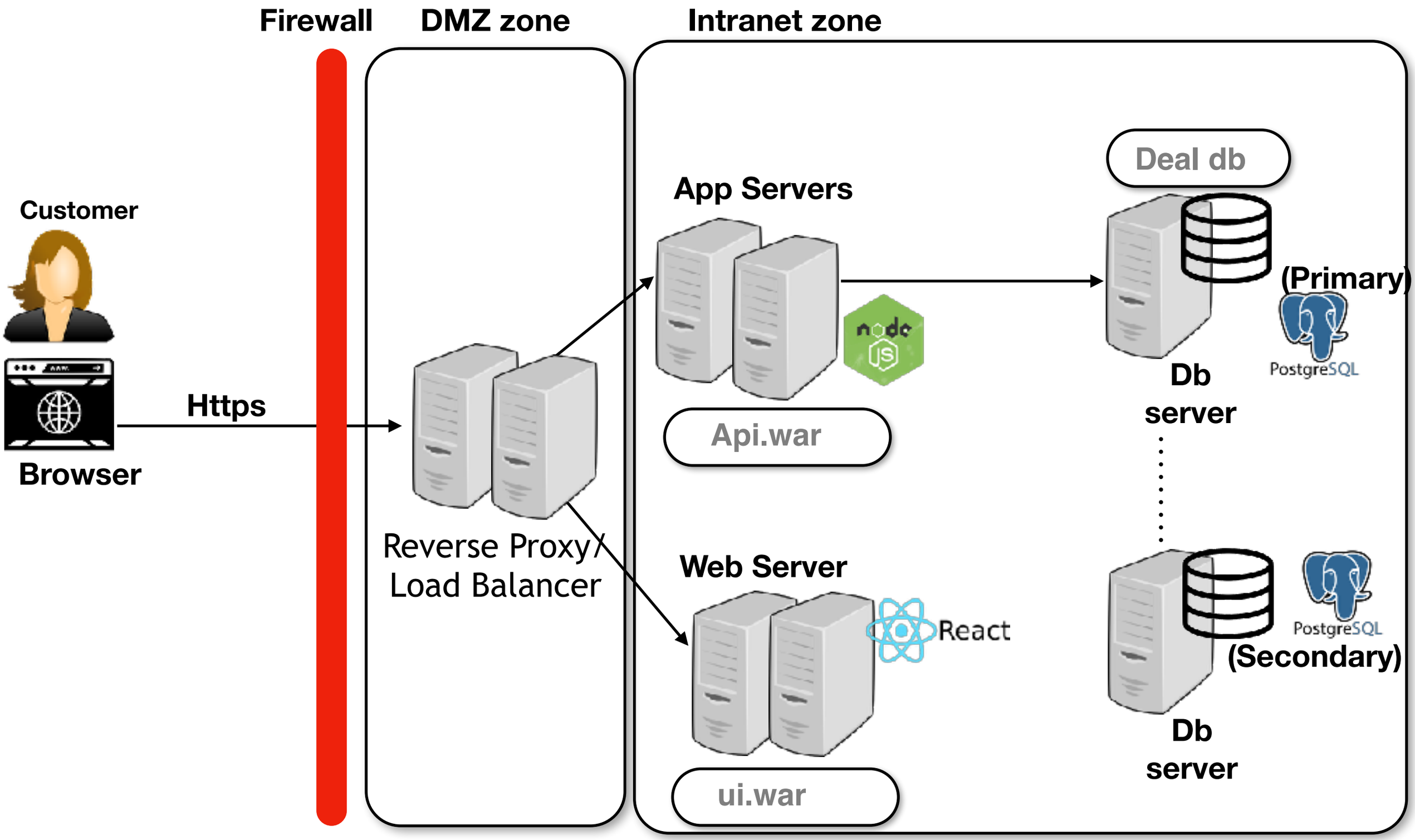
8. Infrastructure View

Physical view

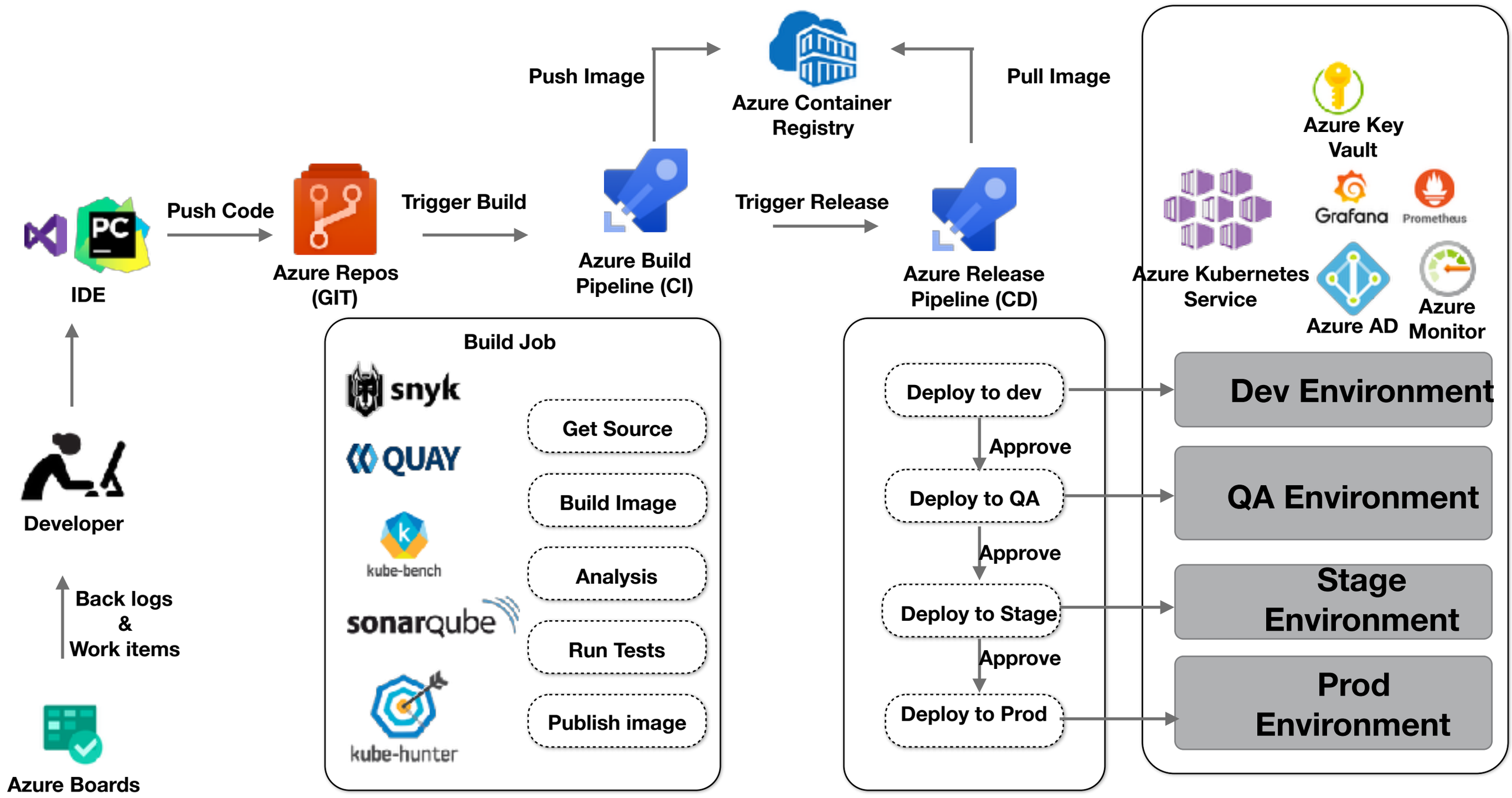


9. Deployment View

Physical view

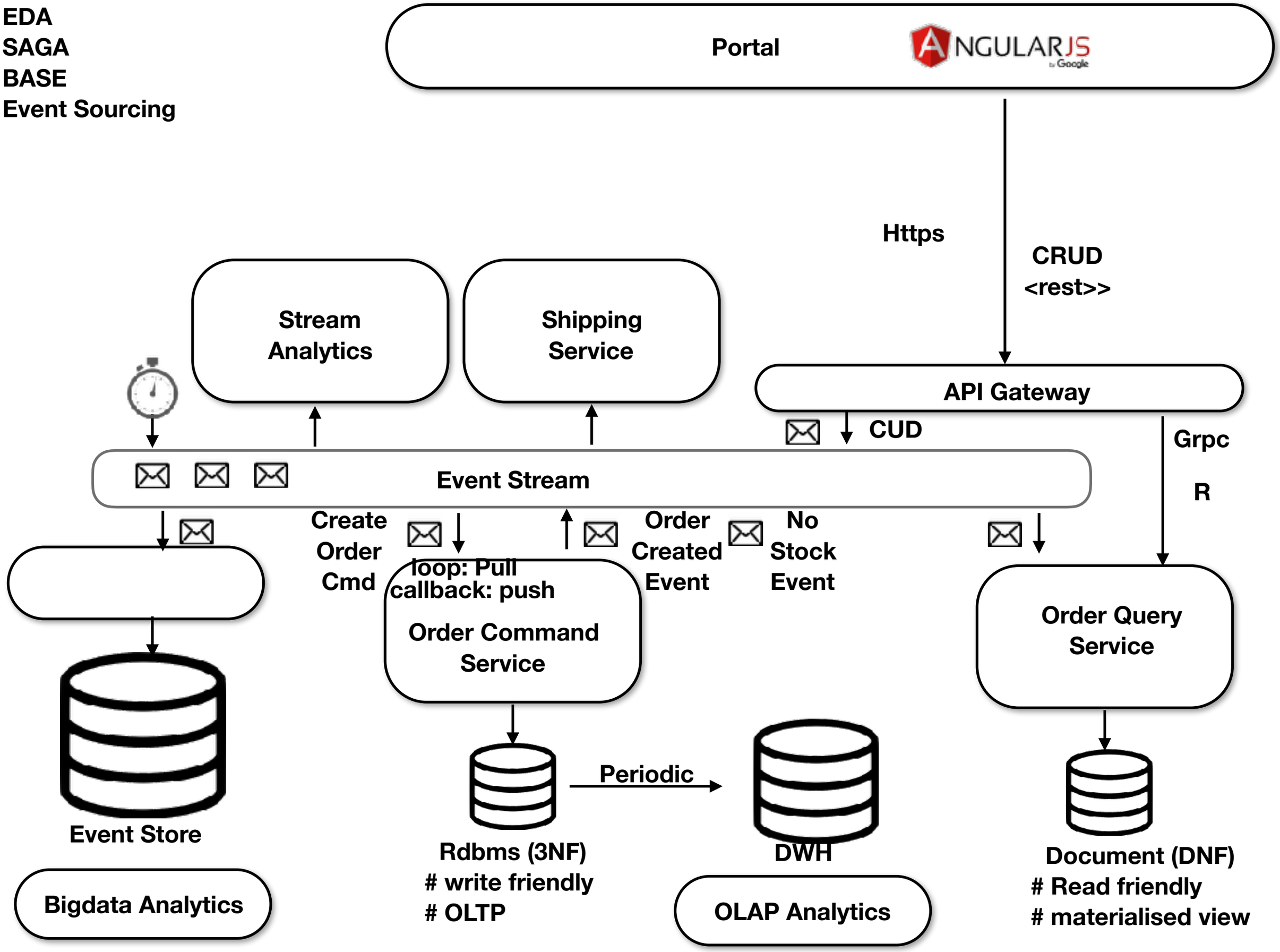


10. DevOps View

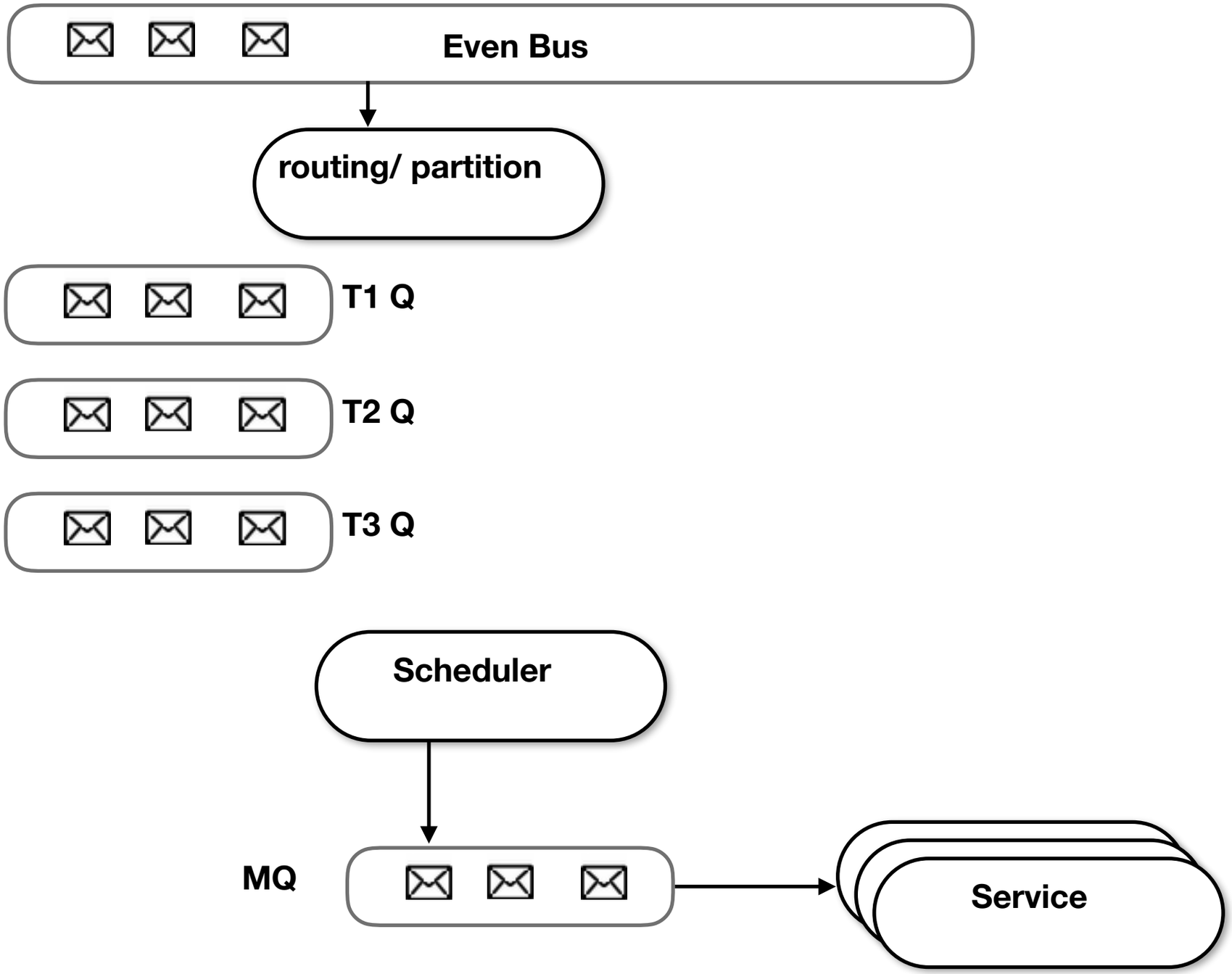


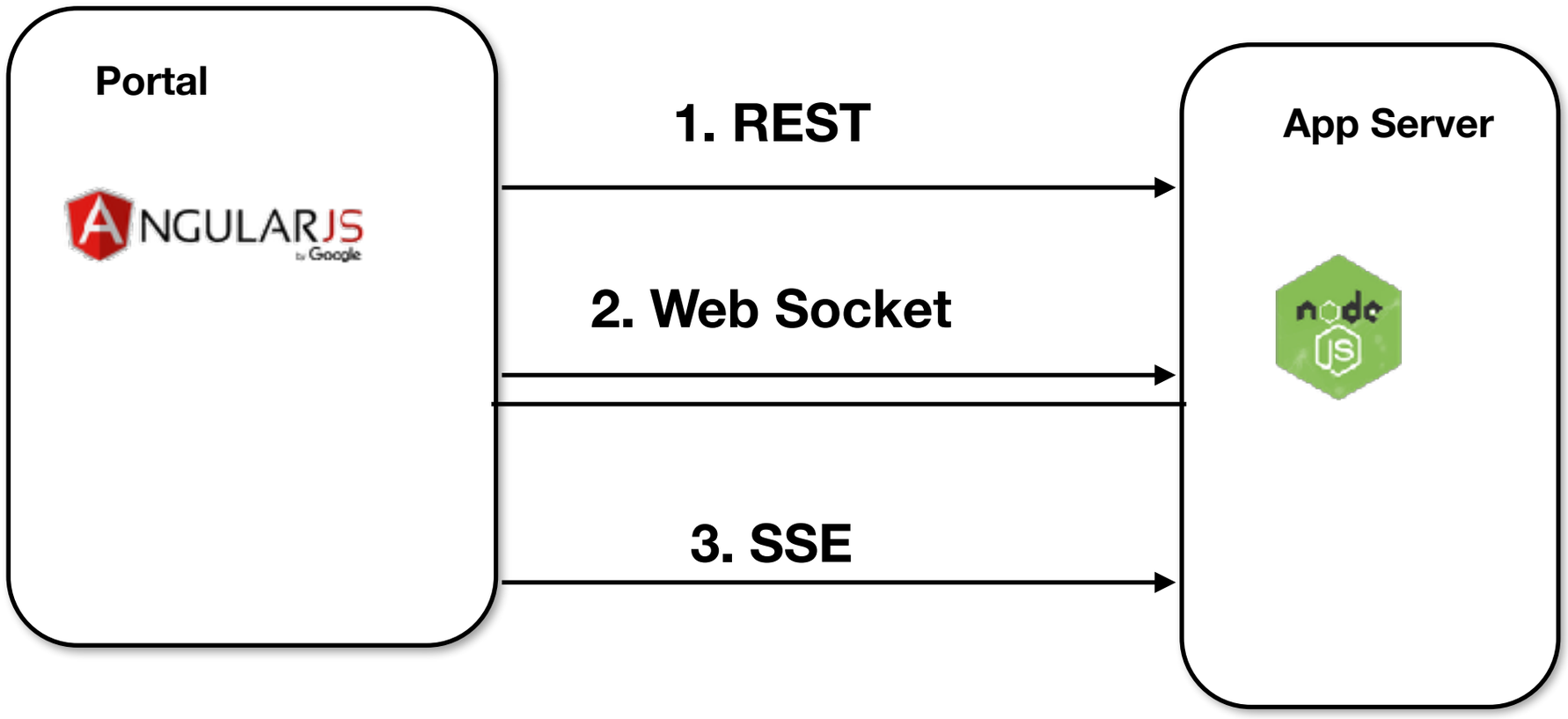
CQRS
EDA
SAGA
BASE
Event Sourcing

“Reference Arch”

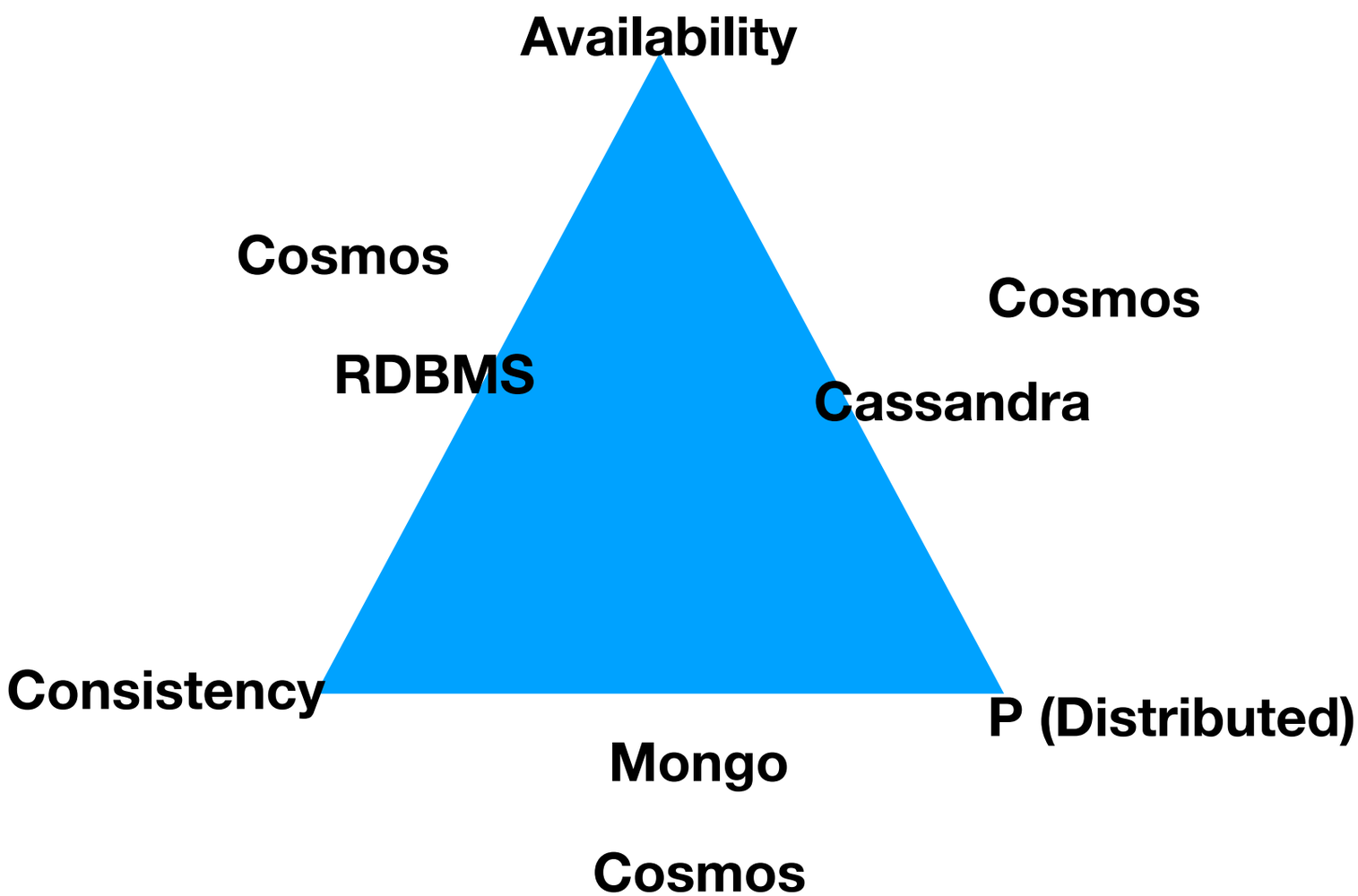


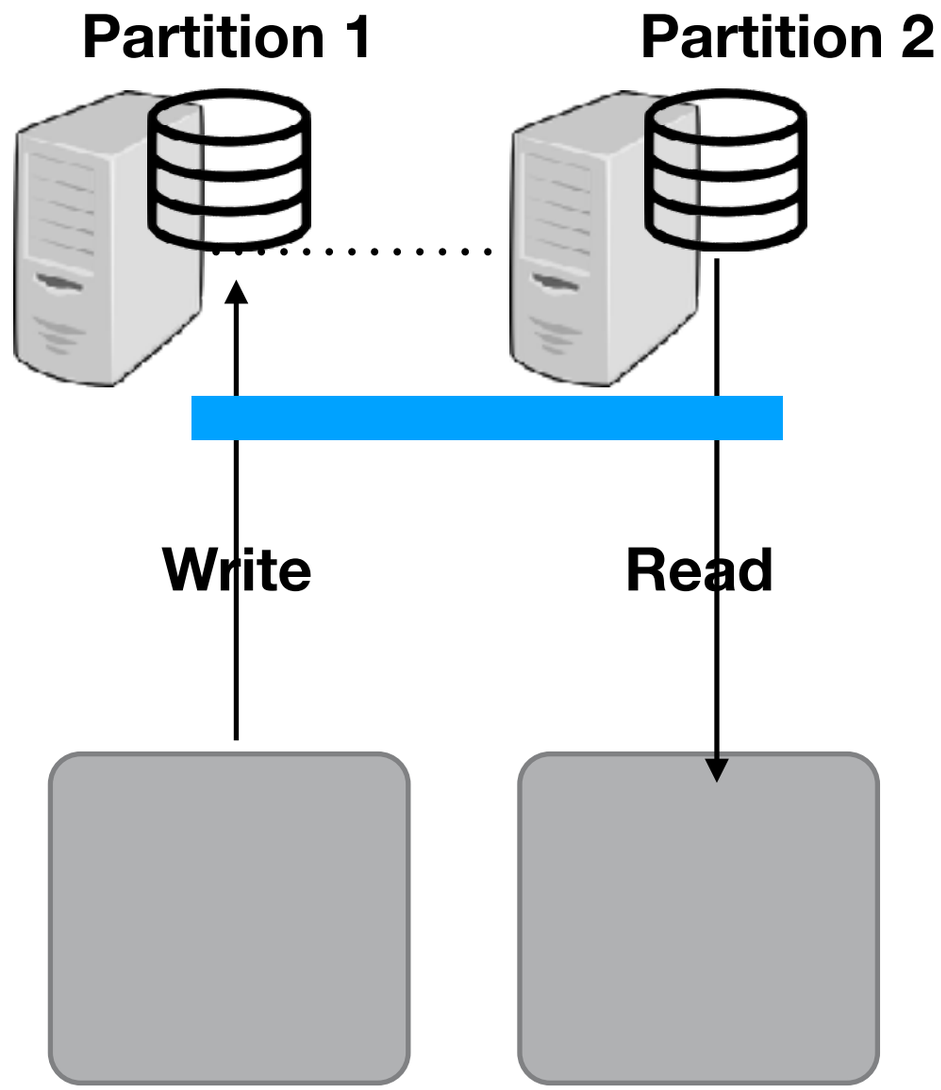
Fun call -> 10 cpu cycle
Network Api call -> 20,00,000 cpu cycles


Even Bus
routing/ partition
T1 Q
T2 Q
T3 Q
Scheduler
MQ
Service


Portal
ANGULARJS
by Google
1. REST
2. Web Socket
3. SSE
App Server
node
JS

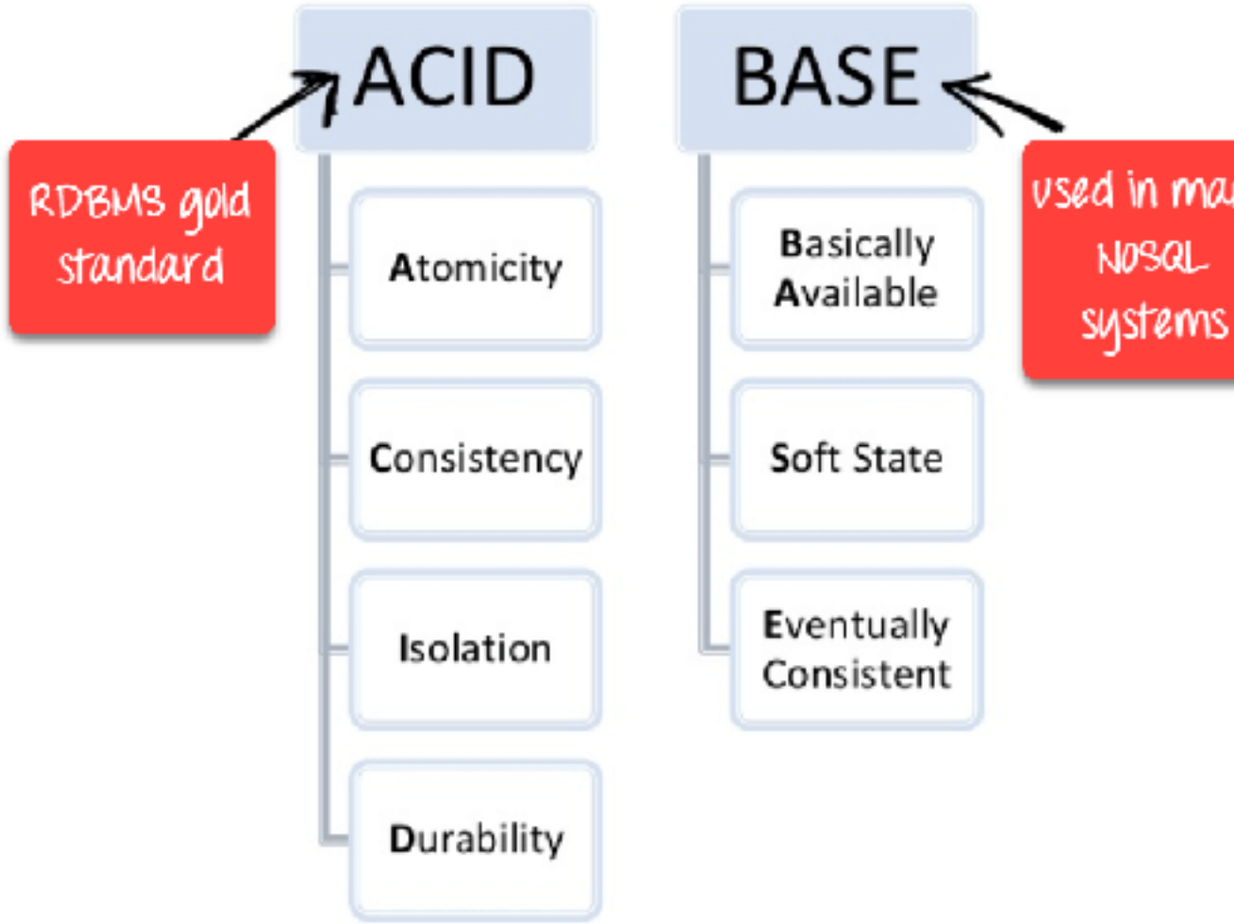
CAP Theorem


Availability
Cosmos
RDBMS
Cosmos
Cassandra
Consistency
P (Distributed)
Mongo
Cosmos


Partition 1
Partition 2
Write
Read

Rdbms
 # 3rd normal Form
 # referential integrity
 # no duplicates
 # write friendly
 # more joins
 # not read friendly

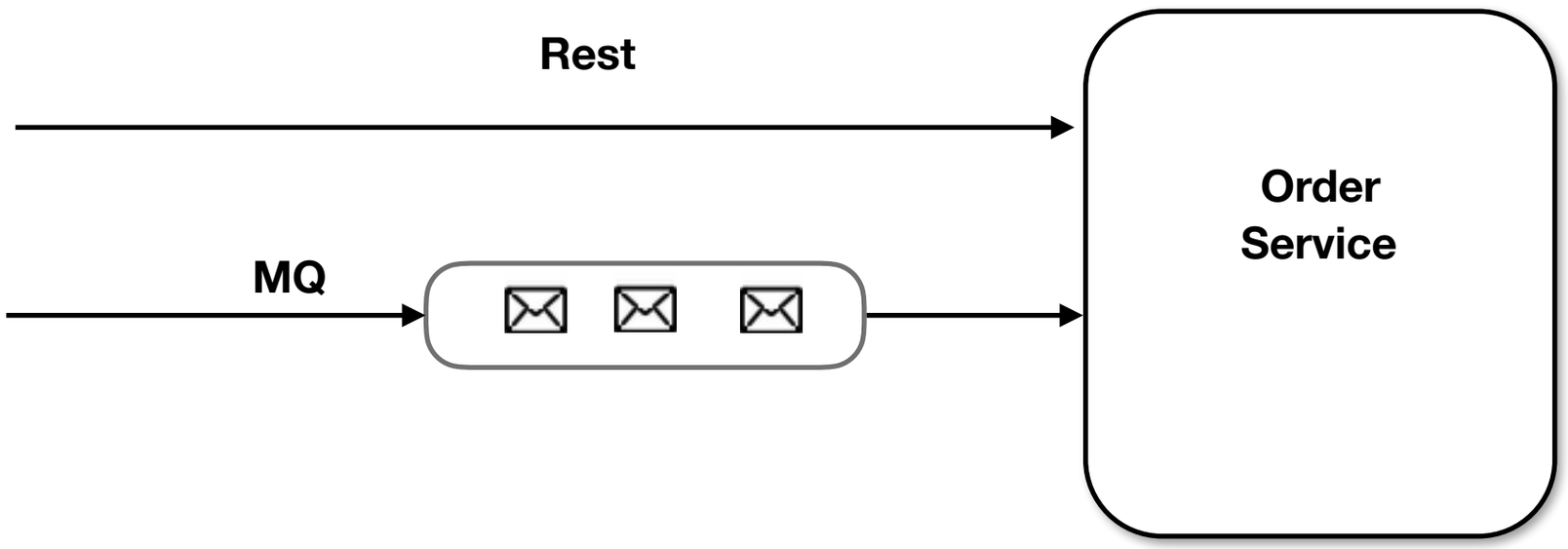
Document
 # DNF
 # no referential integrity
 # more duplicates
 # not write friendly
 # no joins
 # read friendly



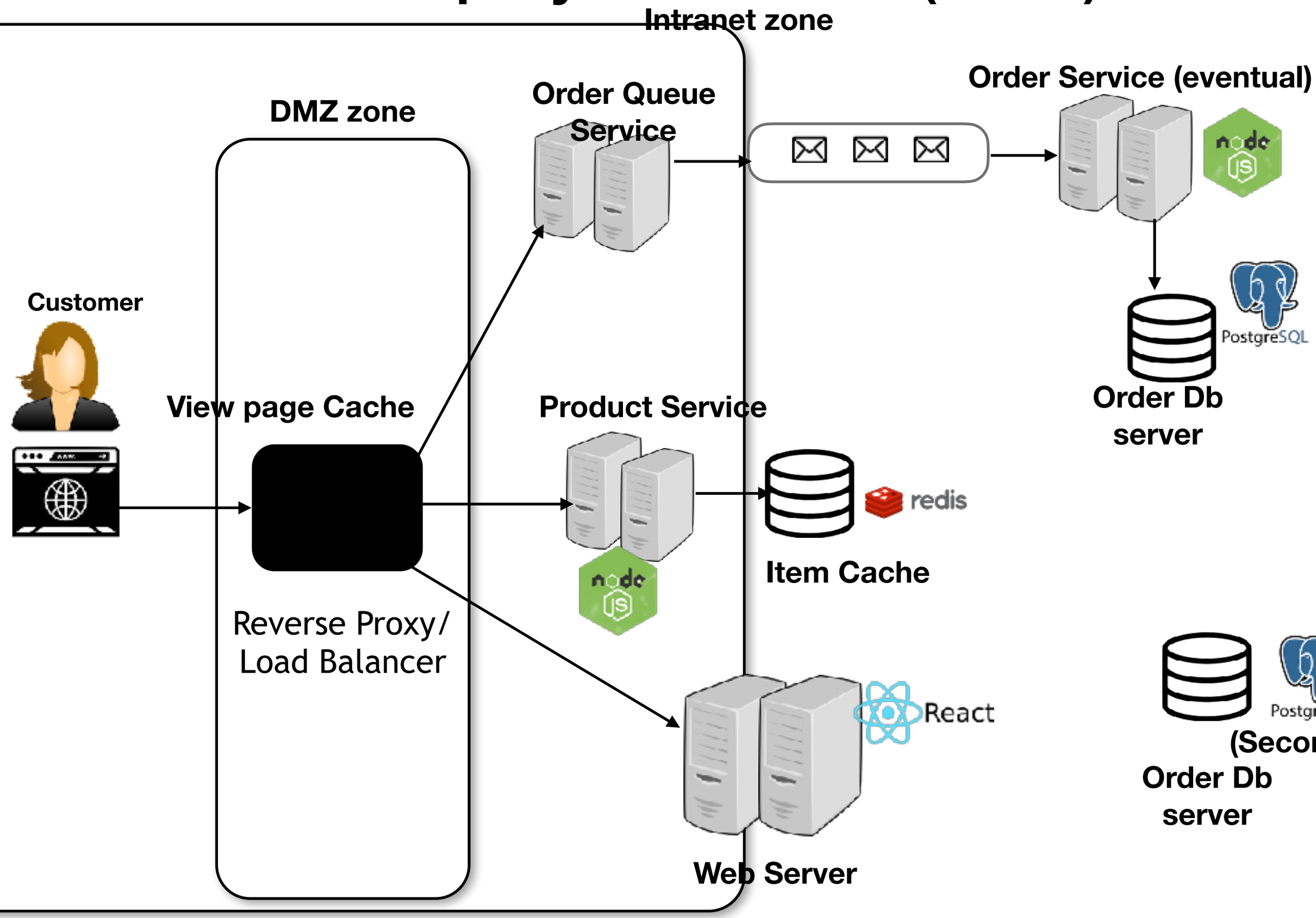
API Gateway
 # Centeralize Authentication
 # Centeralize Autorization
 # Protocol translation
 # SSL termination
 # Load Balancer
#

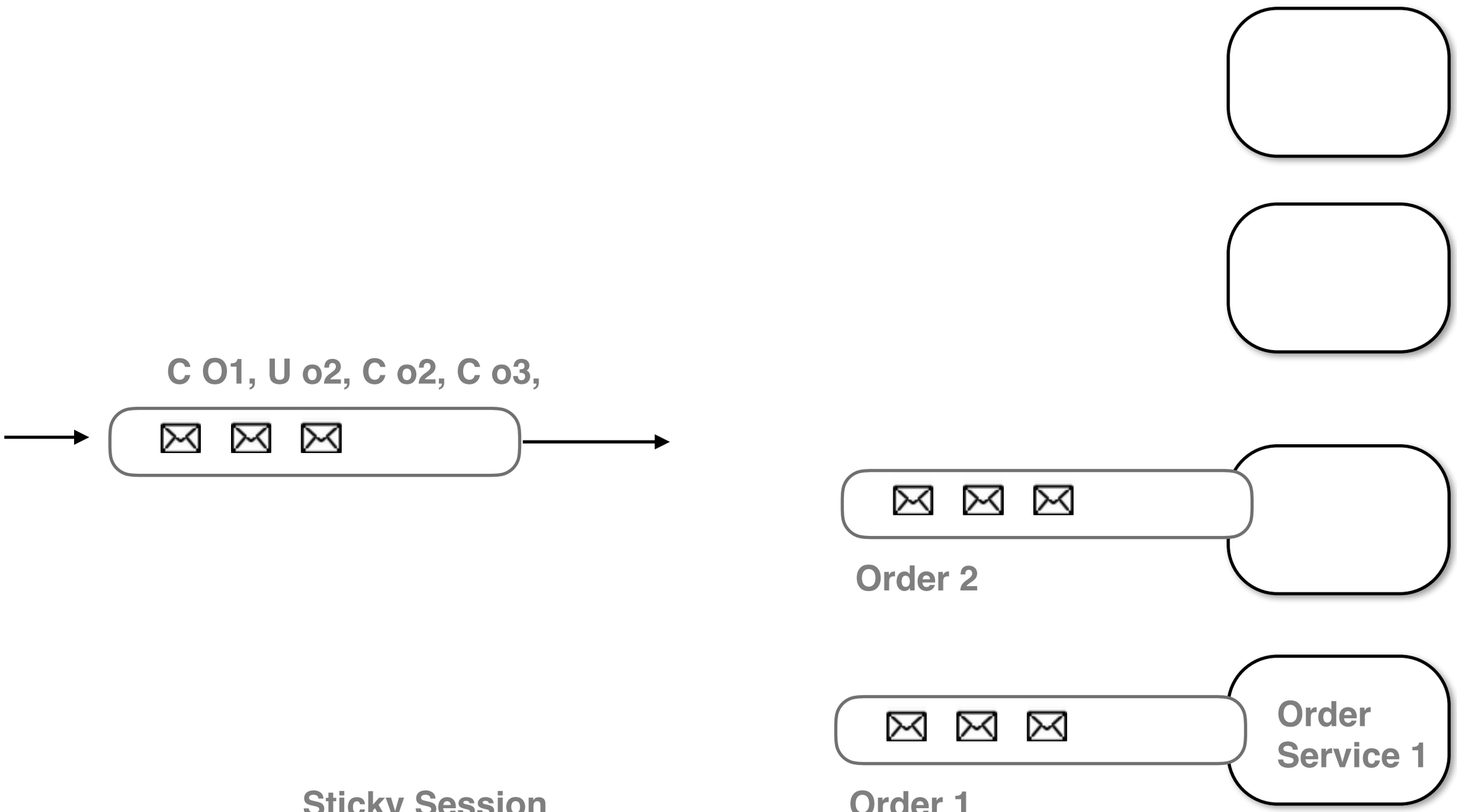
1, MQ is not a queue (not FIFO)
   Unordered delivery
2. Duplicate Msg
   # Idempotent
 3. One way
   # success
   # failure

1. EDA
2. Distributed Transaction
3. Sync
4. Pub/ sub
5. Reliability
6. Scale



6. Deployment View (IAAS)





Sticky Session
Actor Pattern

7. Security View

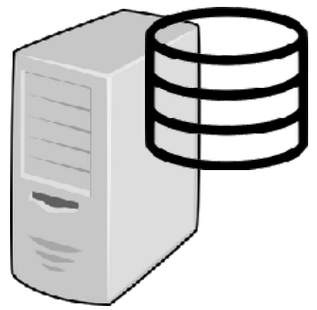
Authentication (First Defense)	Oauth2 + openid connect
Authorization	Role based
Audit (Last defense)	
Data Security # In Transit # In Rest	Fwk
Input Validation	Fwk, WAF
Exception Handling	Fwk
Session Handling	fwk
Key management	Vault

STRIDE

Spoofing	Authentication
Tampering	Data security
Repudation	Lack of audit
Information disclosure	Authorization Exception Handling Input validation
Deniel of Service	Input validation Fire wall
Elevation of Privilege	Input validation Authorization

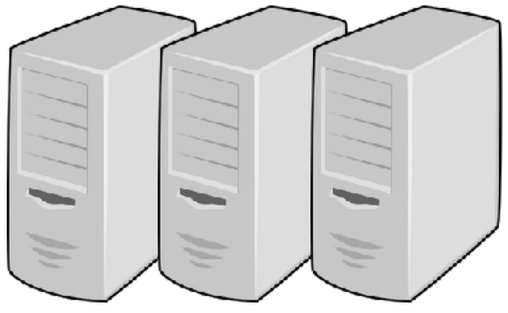
Scalability

App Servers



Db
server

Horizontal scaling / Scale out



App Servers

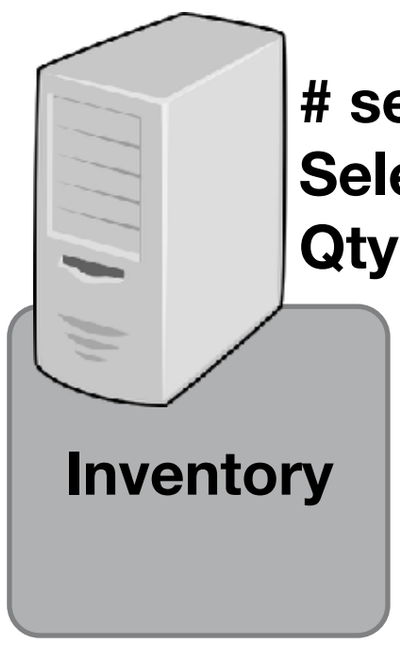
Vertical scaling / scale up

Db
server

Stateful

Customer

Session id

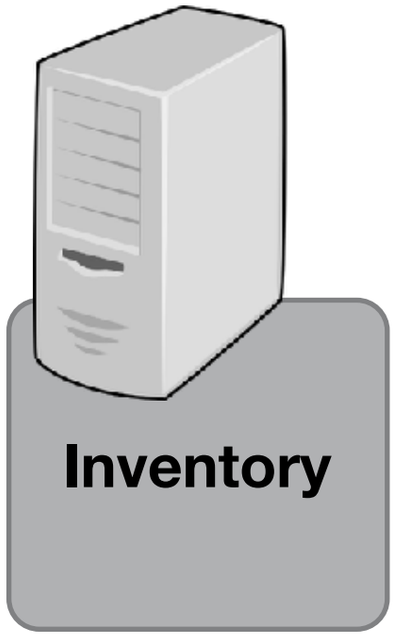


session data :
Selected Item
Qty

Inventory

Stateless

Customer

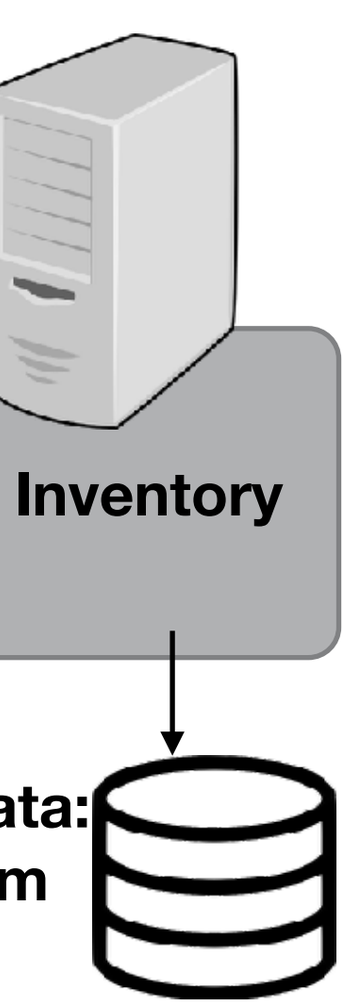


Inventory

Session data:
Selected Item
Qty

Stateless

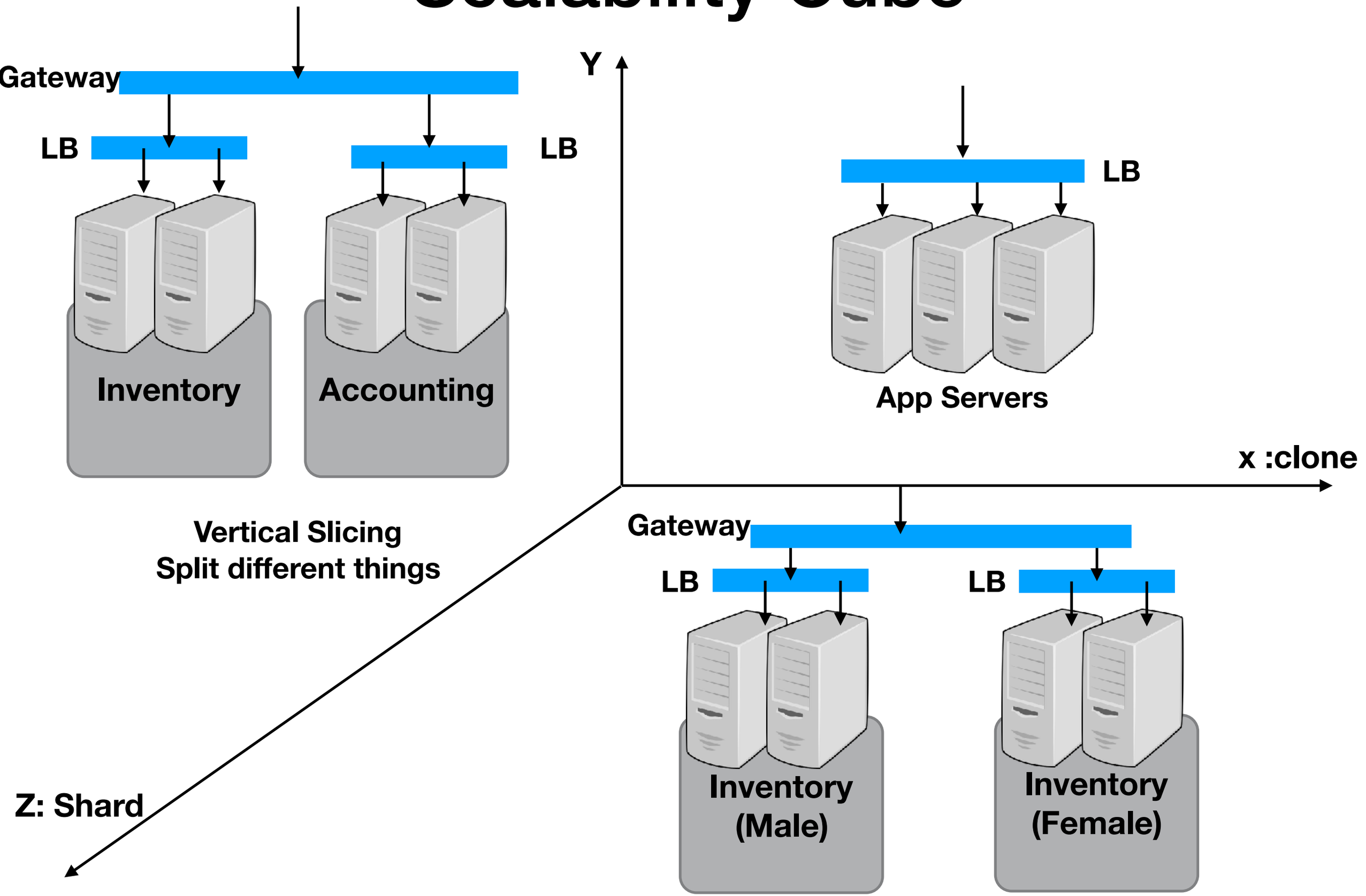
Customer



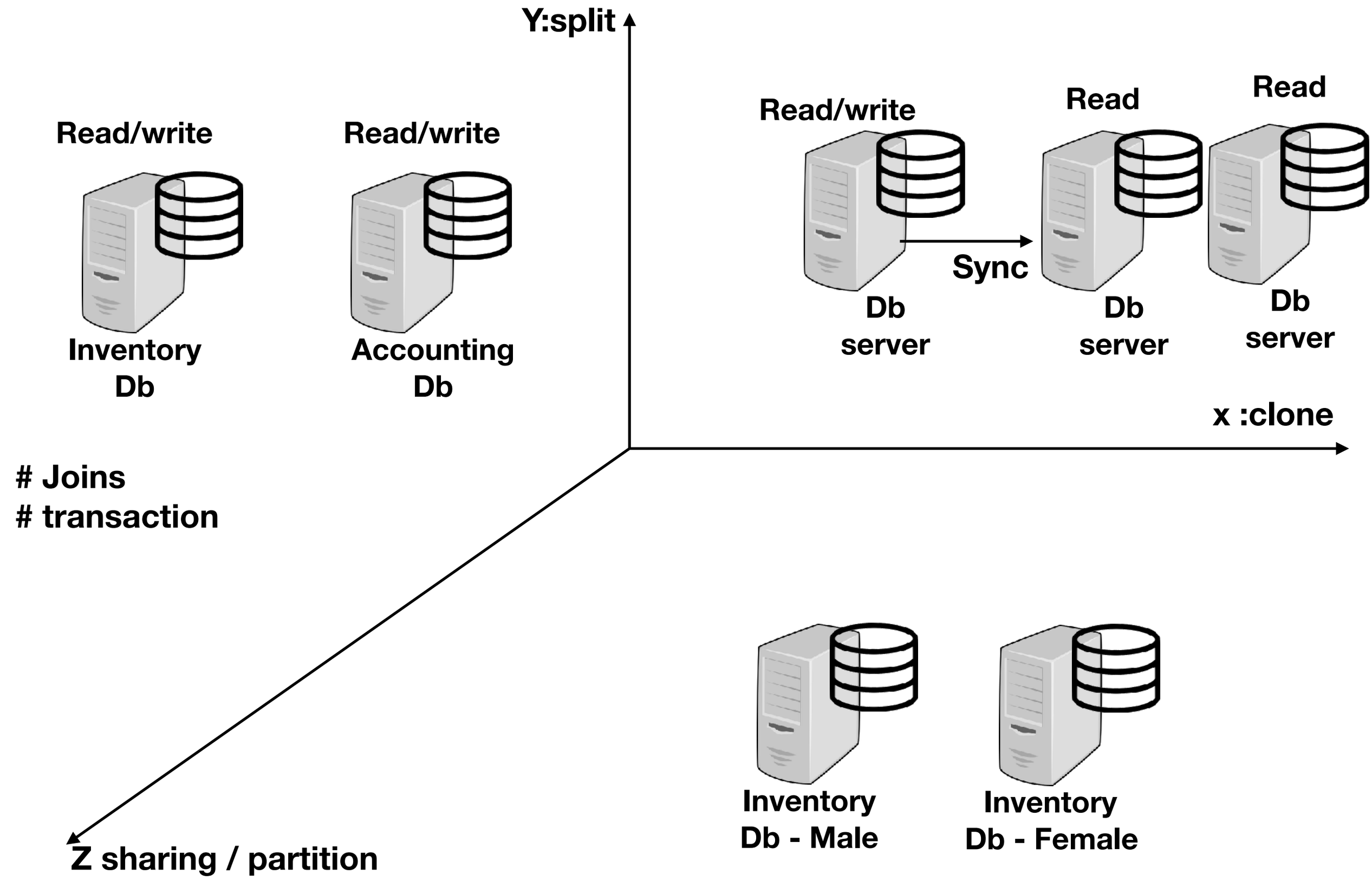
Inventory

Session data:
Selected Item
Qty

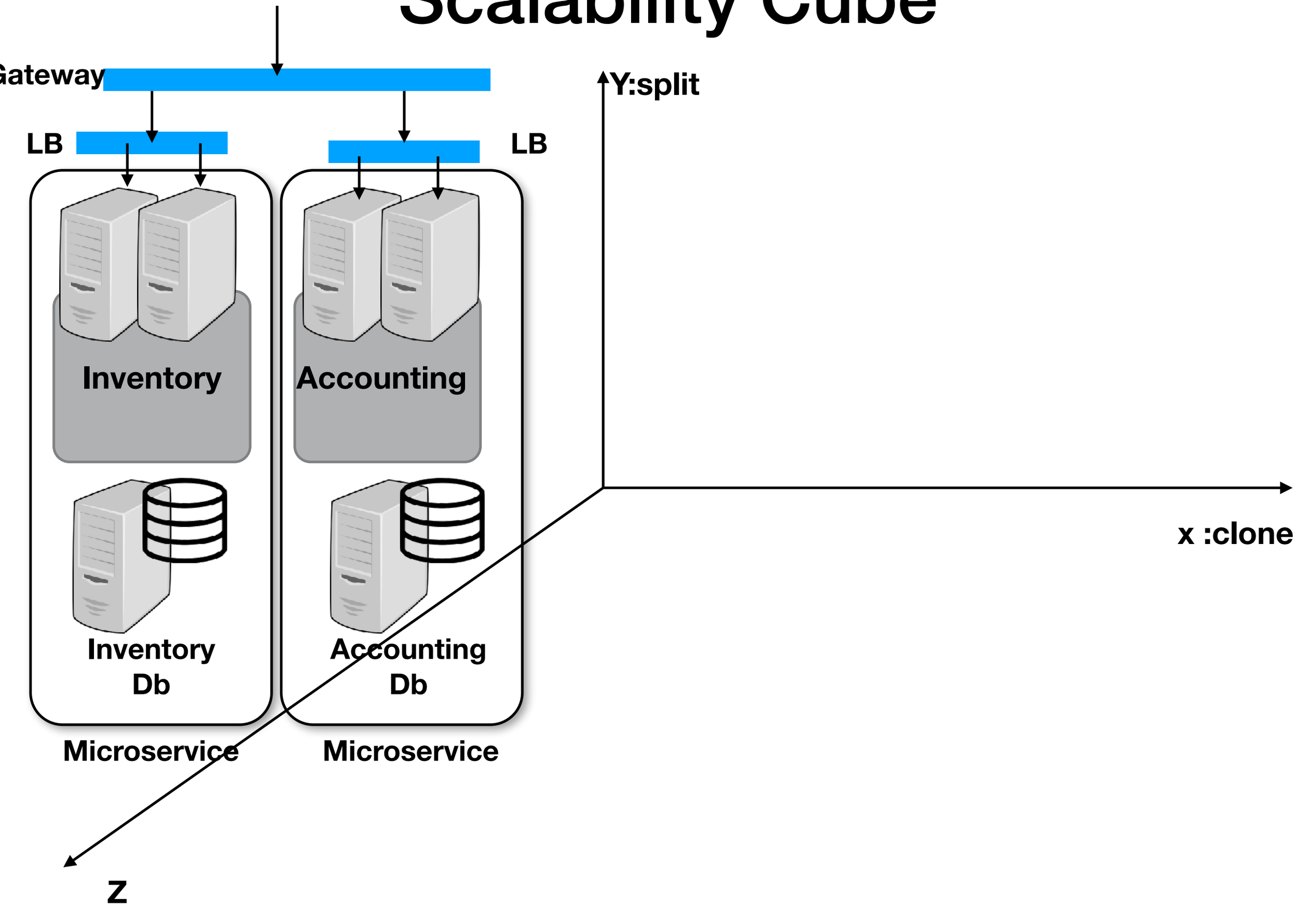
Scalability Cube



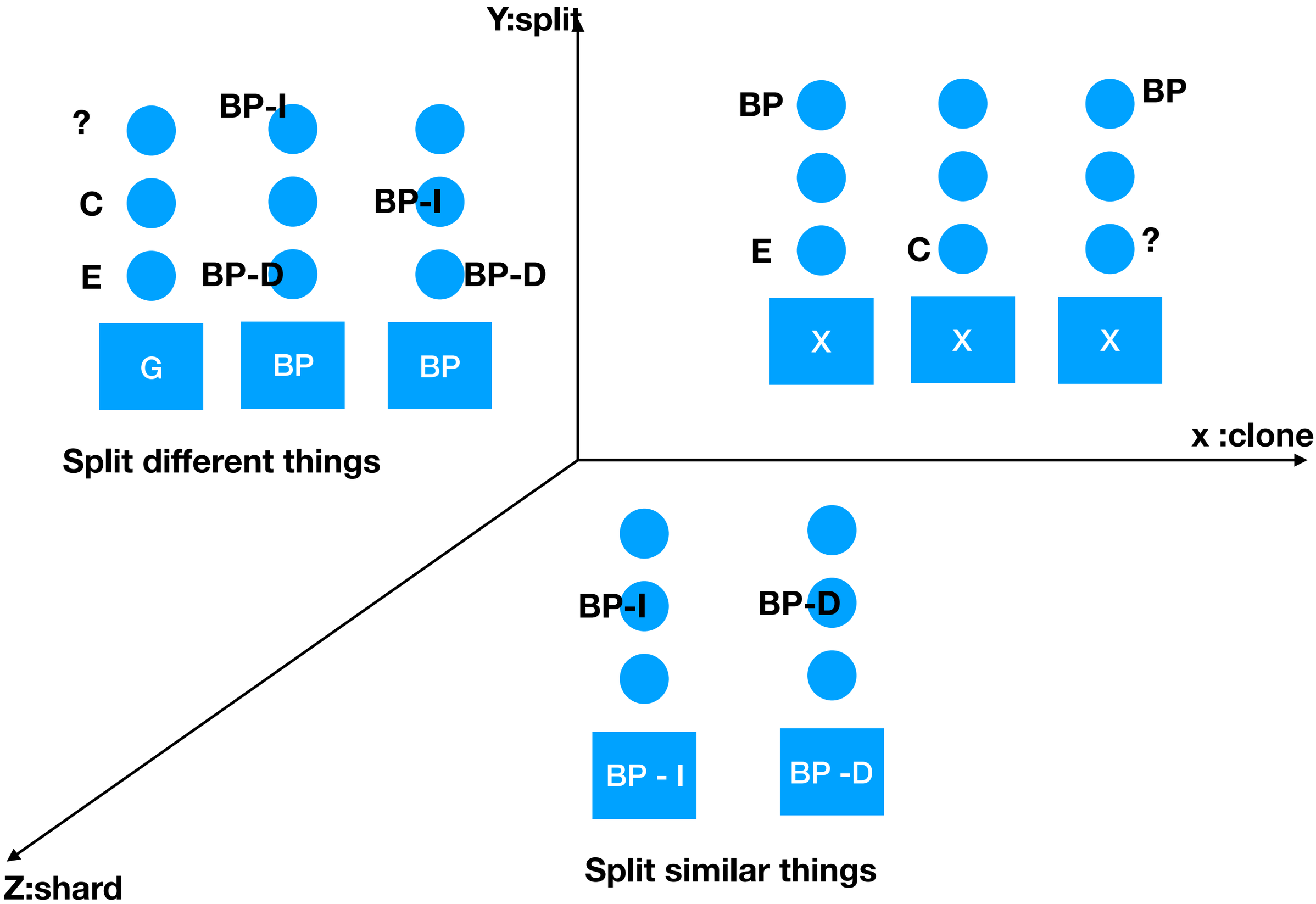
Scalability Cube



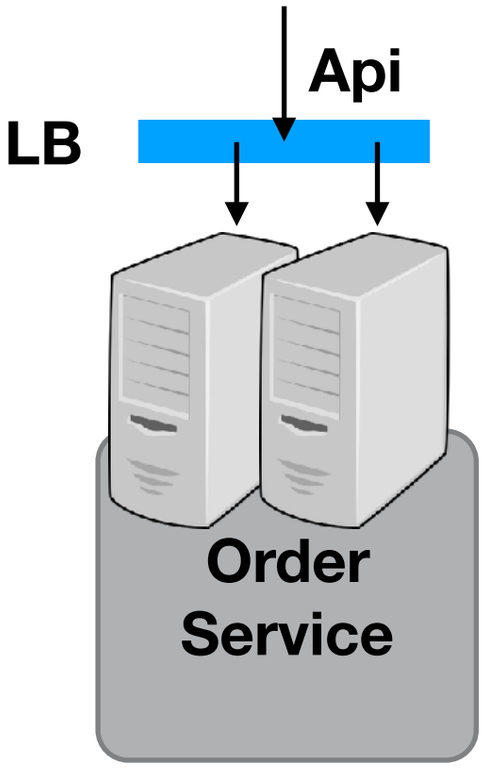
Scalability Cube



Scalability Cube



API call

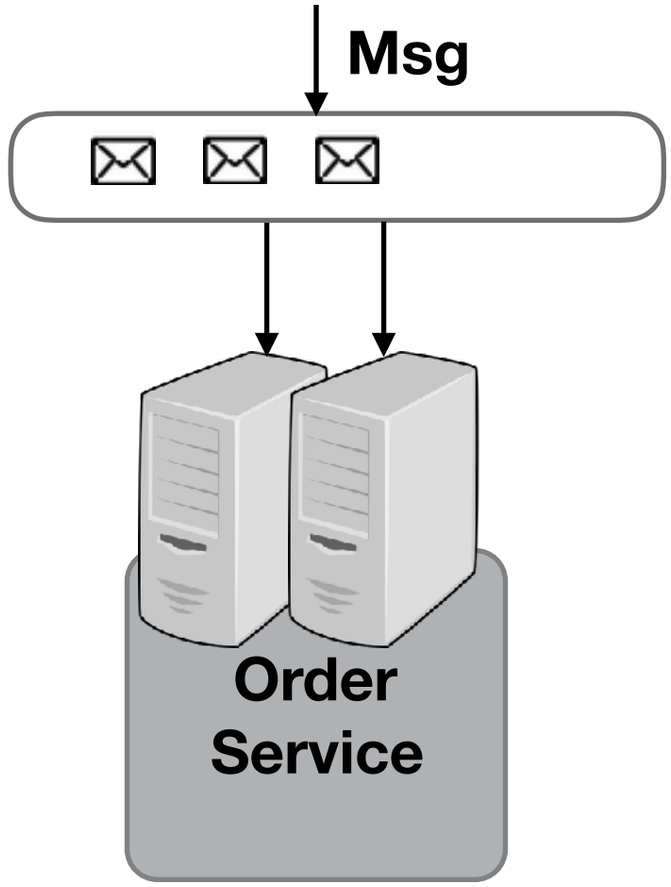


Less Scalable
no of server’s depends on peak load

less reliable
Will not remember last execution context
After recovery during a crash

loss of data
Because of throttle

Messaging

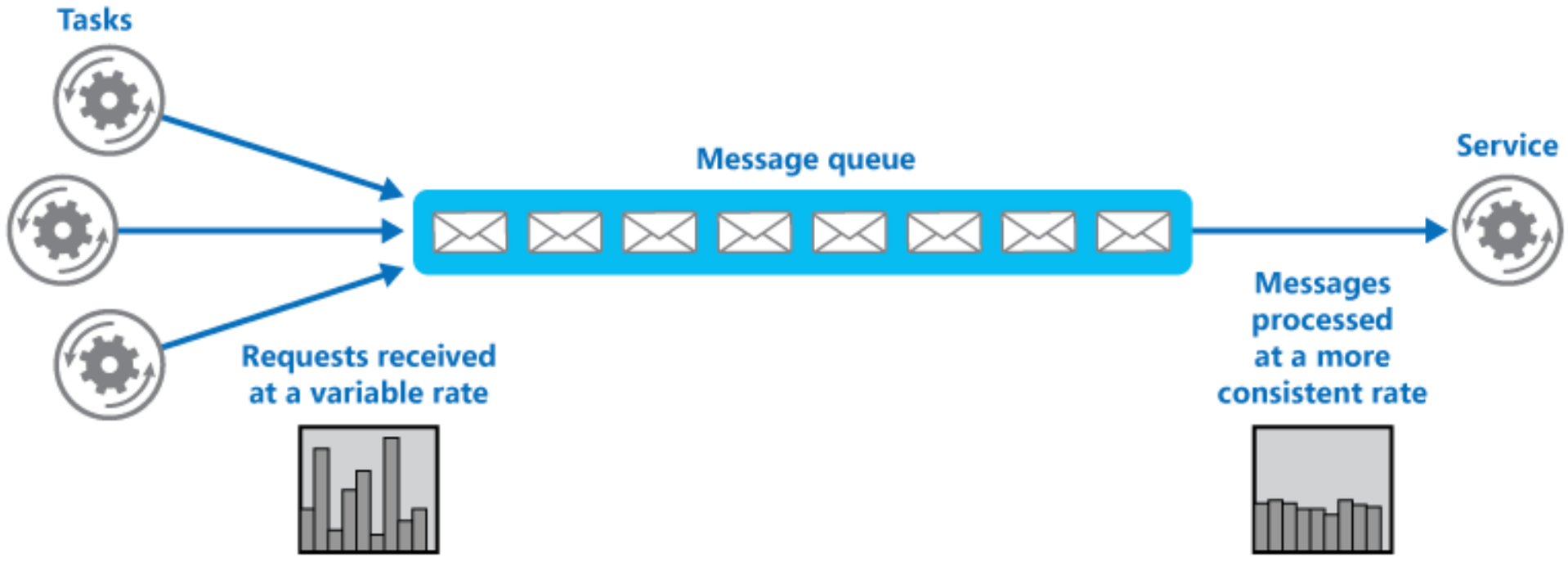


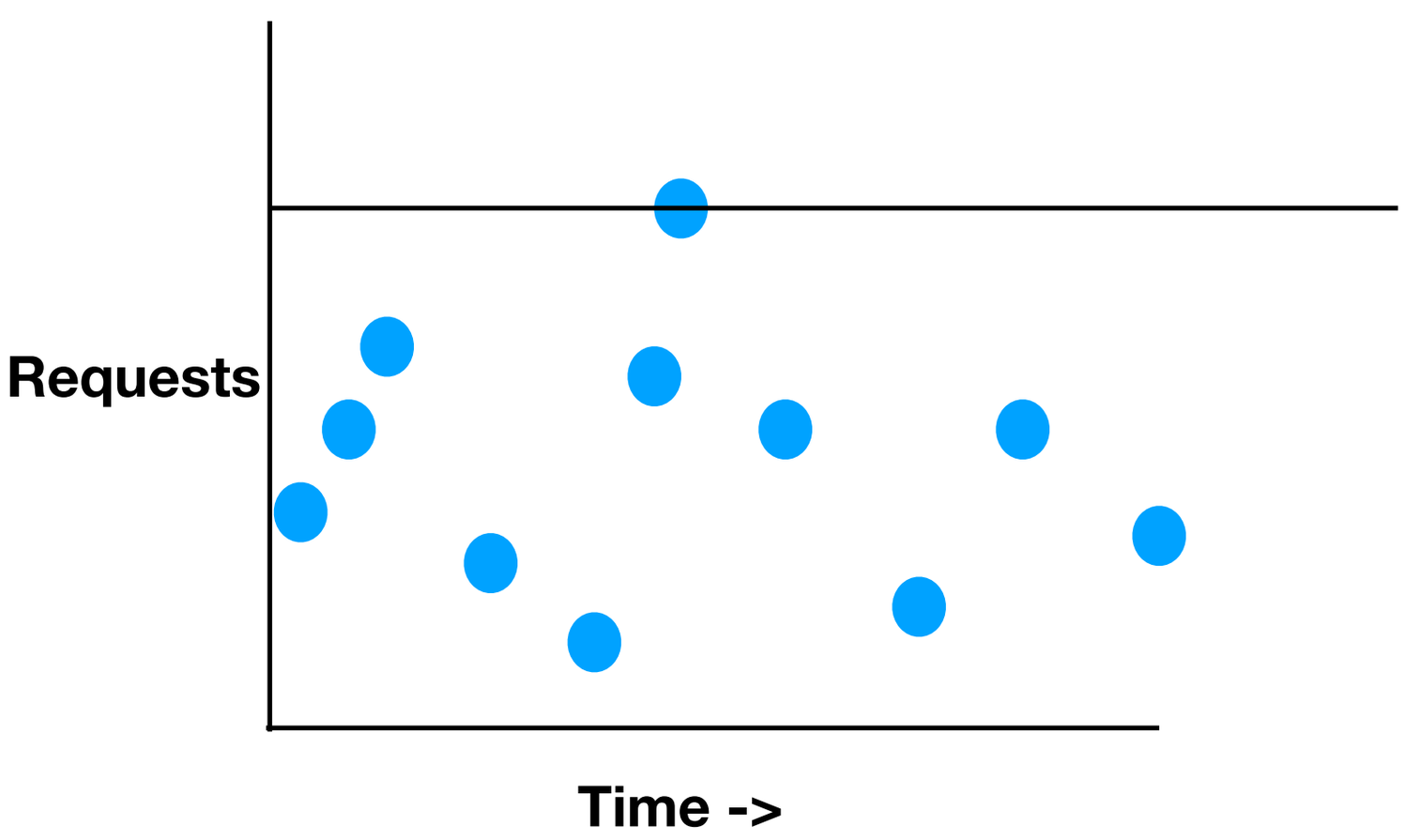
more scalable
no of server’s depends on Avg load
(Load Leveling)

more reliable
If message are not ACK, messages
reappear in Queue

no loss of data
Gets queued

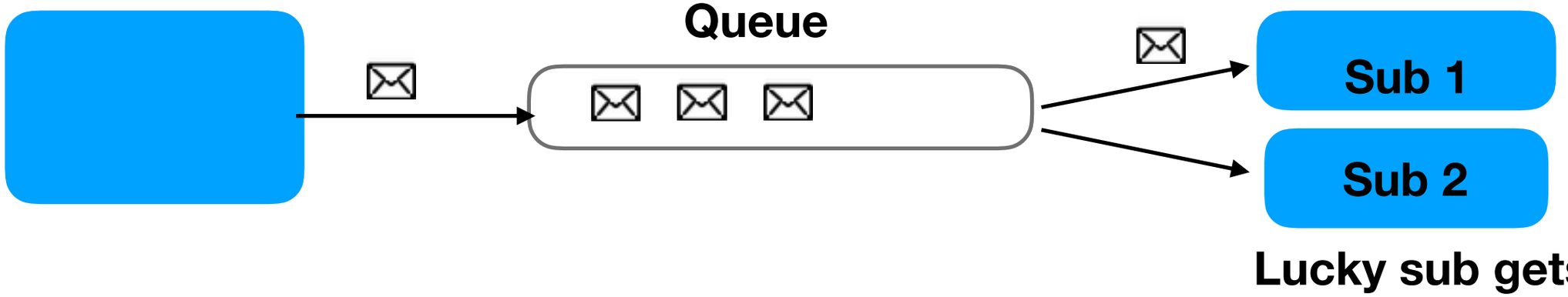
	API	Message	
Development time	++	- - (1:4)	Duplicate Message - Idempotent Unordered delivery
Protocol	Request-response	One Way	
Delivery	Ordered	Unordered	
Duplicate	No	Yes	
Consistency	Immediate (ACID)	Eventual (BASE)	
Learning Curve	++	- -	
Deployment	+ +	- -	

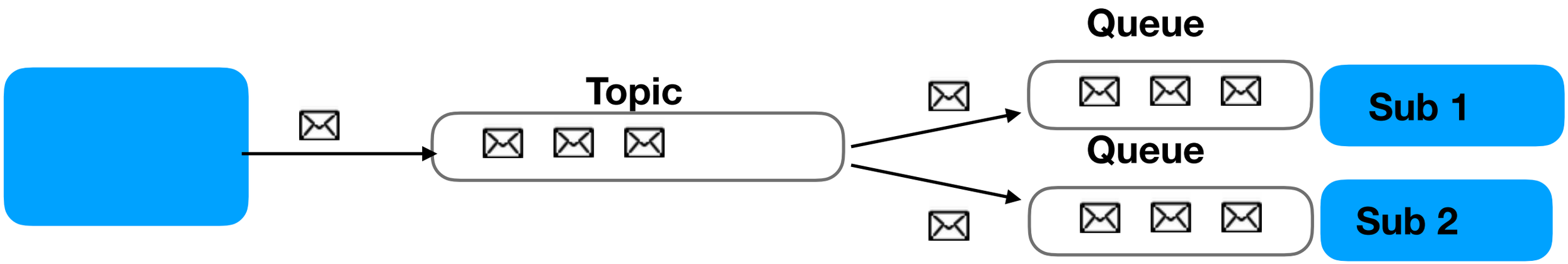

Tasks
Message queue
Service
Requests received at a variable rate
Messages processed at a more consistent rate


Requests
Time ->

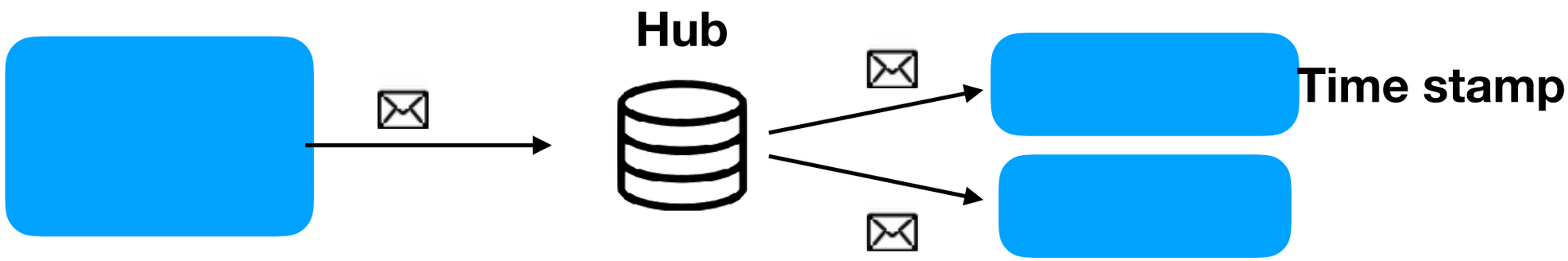
Types of Messaging

Queue	Topic	Hub
Point to Point	Pub Sub	Database





All sub gets it



sub can read msg n times
until TTL

Scalability tactics Check list

Stateless

Vertical Scaling / Scale up

Clone / Scale Out (Honrizonal Scaling)

Splitting (Vertical Slicing)

Sharding / Parititioning (Horizontal Slicing)

Messaging

Caching

Performance tactics Check list

Reduce network I/O

Reduce disk I/O

Compression

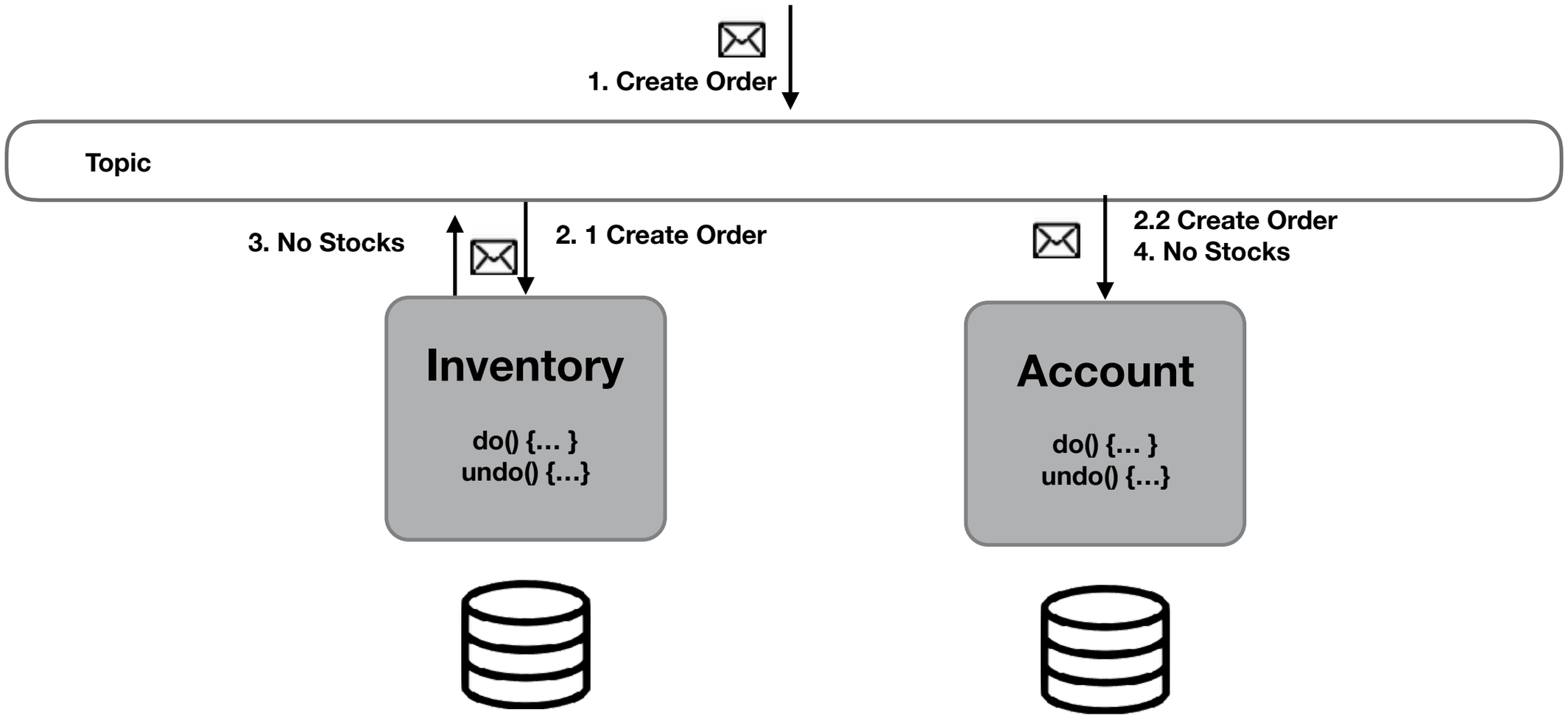
Chunky call (batch)

Caching

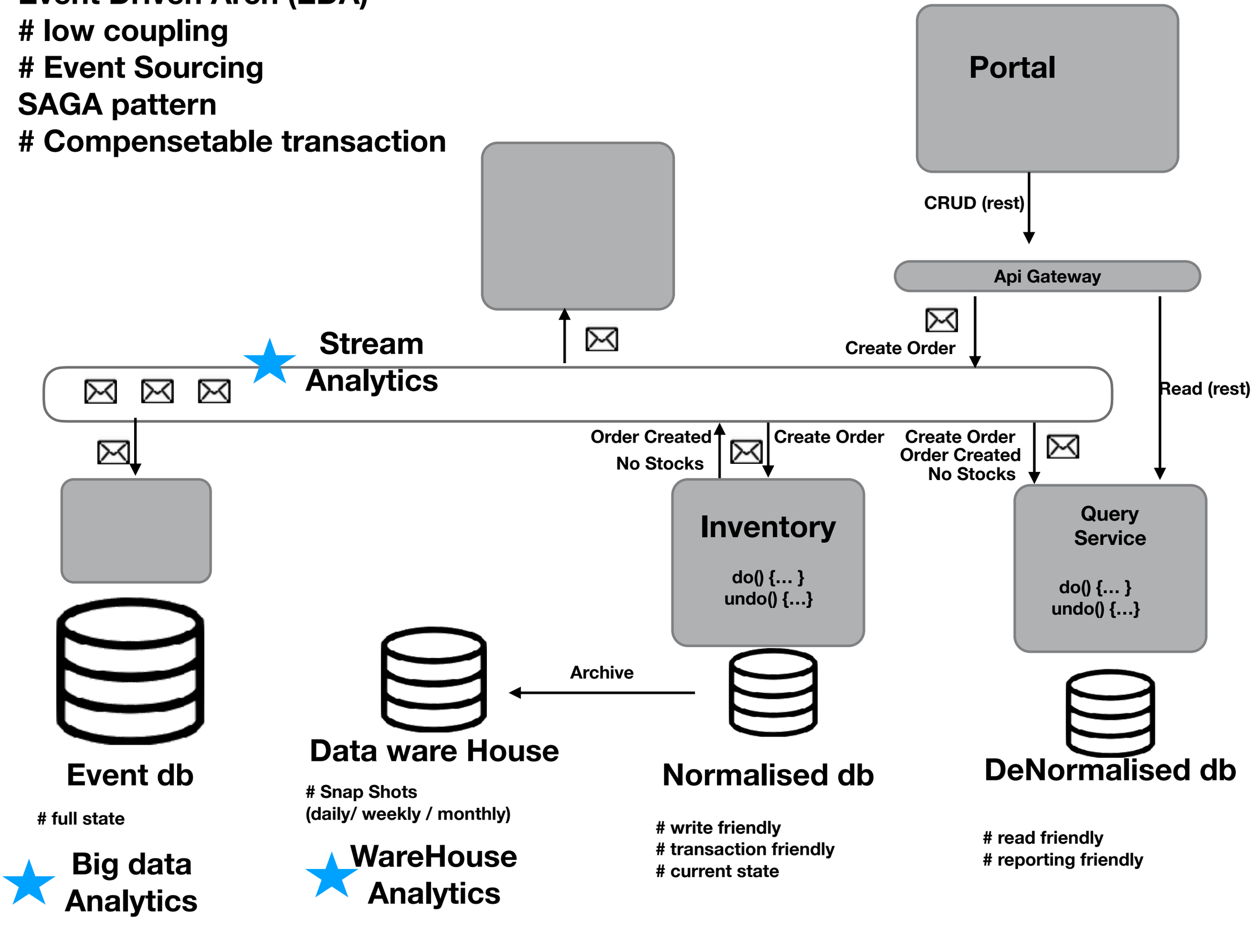
Lazy Loading

Eager Loading

Parallel


1. Create Order
Topic
3. No Stocks
2. 1 Create Order
2.2 Create Order
4. No Stocks
Inventory
do() {… }
undo() {…}
Account
do() {… }
undo() {…}

Event Driven Arch (EDA)
 # low coupling
 # Event Sourcing
SAGA pattern
 # Compensetable transaction



6. Deployment View (PAAS)

Persistence

- File Storage
- RDBMS db
- Document db
- Graph db
- Key Value db
- Column db
- Data Lake Storage

Compute

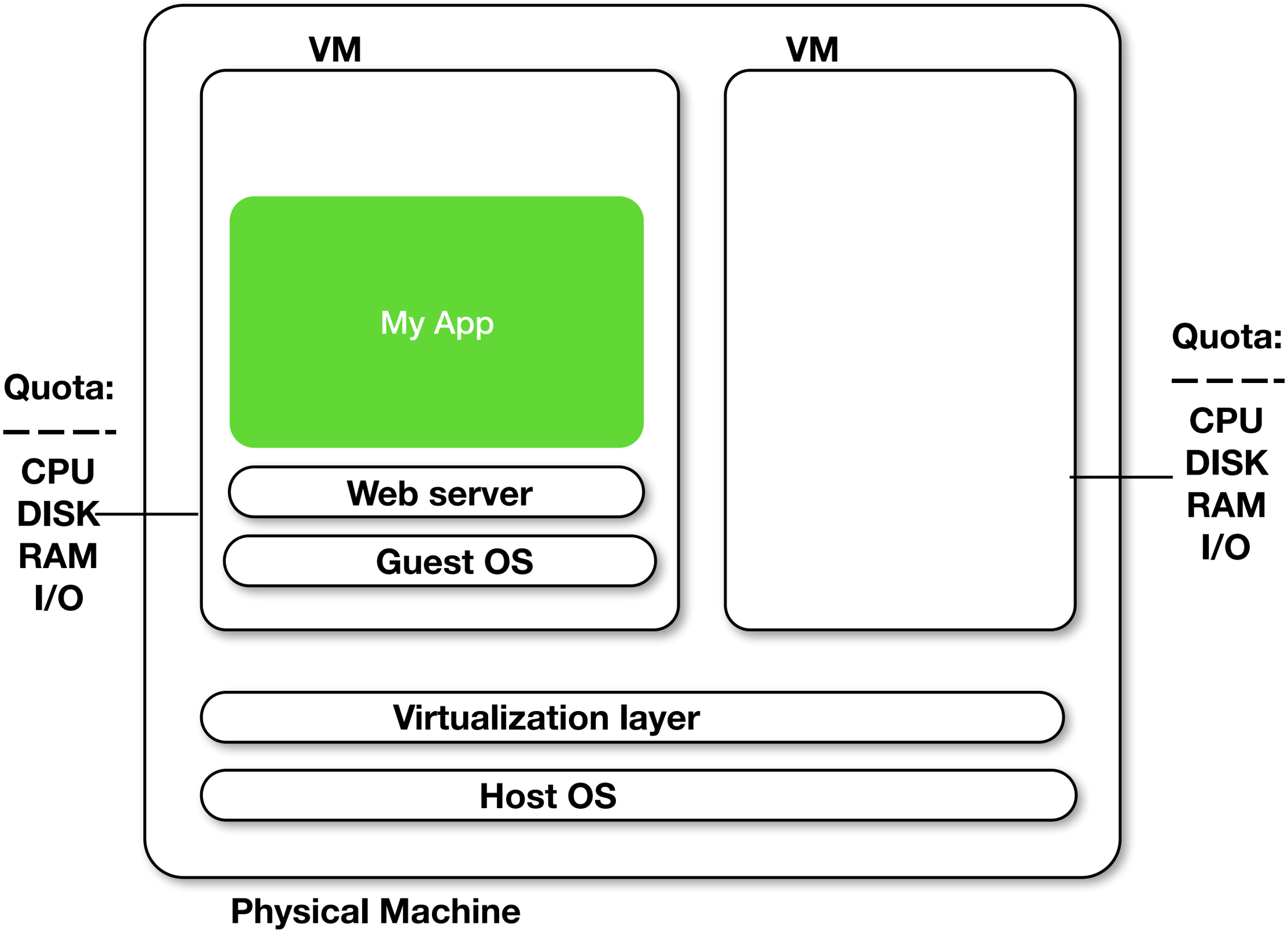
- web server
- Serverless Webserver
- Containerized Server
- Serverless Containerized Server

Messaging

- Queue
- Topic
- Hub

	Distributed	Transaction (ACID)	Binary Data (pic, videos,)	Referential integrity	Hierarchical	
• File Storage	Yes	n	Y	N	N (binary)	S3, HDFS, Blob
• RDBMS db	No (gb)	y	N	Y	N (tabular)	Postgres
• Document db	Yes	n	N	N (denormalized)	Y (tree)	Mongo
• Graph db	No (gb)	y	N	Y	Y (graph)	Neo4j
• Key Value db	Yes	n	N	N	N (key:value)	Redis
• Column db	Yes+	n	N	N	N (tabular)	HBase
• Data Lake Storage	Yes++	n	Y	N	N (binary)	Azure, aws, Google

• web server	Pay for VM’s		
• Serverless Webserver	Pay by use (cpu, memory)		
• Containerized Server	Pay for VM’s		
• Serverless Containerized Server	Pay by use (cpu, memory)		


VM
VM
My App
Web server
Guest OS
Quota:
CPU
DISK
RAM
I/O
Quota:
CPU
DISK
RAM
I/O
Virtualization layer
Host OS
Physical Machine

Physical Machine

VM

Container

Container

Quota:

CPU
DISK
RAM
I/O

My App

Web server

Containerisation layer

Guest OS

Quota:

CPU
DISK
RAM
I/O

Virtualization layer

Host OS

Steps

Collect

- Context view
- Functional View
- Quality requirements
- Constraints

Define

- Logical view
- deployments View

No joins
More duplicates
Not Write friendly
read friendly

Denormalized form

Weather			
city	**state**	**high**	**low**
Phoenix	Arizona	105	90
Tucson	Arizona	101	92
Flagstaff	Arizona	88	69
San Diego	California	77	60
Albuquerque	New Mexico	80	72

3rd normal form

Primary Table

CompanyId	CompanyName
1	Apple
2	Samsung

?

Related Table

CompanyId	ProductId	ProductName
1	1	iPhone
15	2	Mustang

Associated Record ✓

Orphaned Record ✗

More joins
No duplicates
Write friendly
Not read friendly

Bad

- Alice in wonderland

- Though shall have Quality

Good

- SEI Quality Attribute Scenario - collecting requirements
- SEI ATAM (Architecture trade off analysis method) - eval
- Views - doc
-

Eval

Review the arch

Evaluator + arch team

- Identify all architectural approaches (A1, A2, A3, A4, A5)

- Identify all Quality Attribute Scenarios (S1, s2, s3)

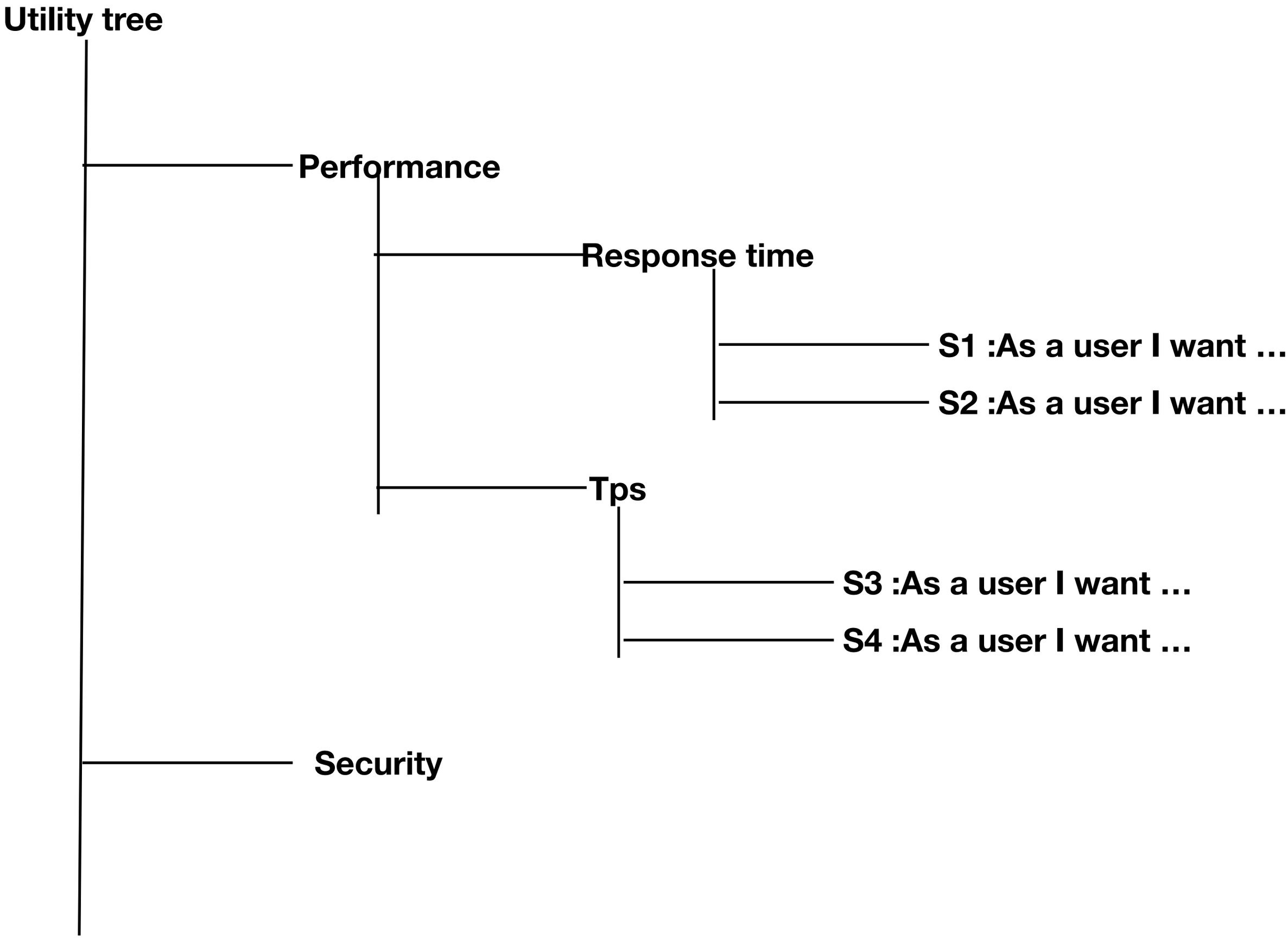
- Analyze Approach and Scenarios
  SC#1 -> A1, A7
  tradeoff:
  SC#2 -> A3
  tradeoff:
  SC#3 -> ?

Review the eval

Evaluator + arch team + all stake holders

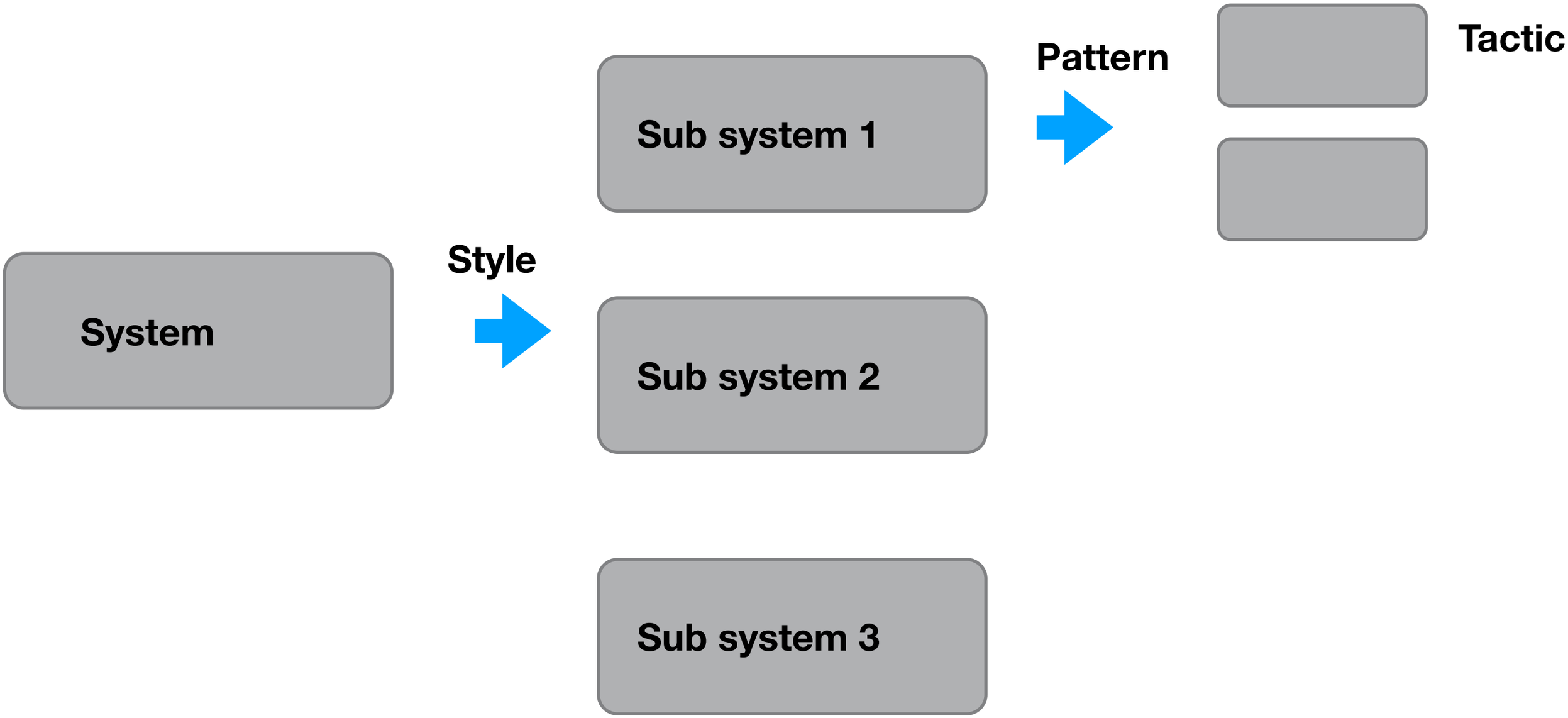
- Brain storm for scenarios (s5,s6,s7)

- Analyze Approach and Scenarios
  SC#5 -> A1, A7
  tradeoff:
  SC#6 -> A3
  tradeoff:
  SC#7 -> ?


Utility tree
Performance
Response time
S1 :As a user I want …
S2 :As a user I want …
Tps
S3 :As a user I want …
S4 :As a user I want …
Security

5. Logical View

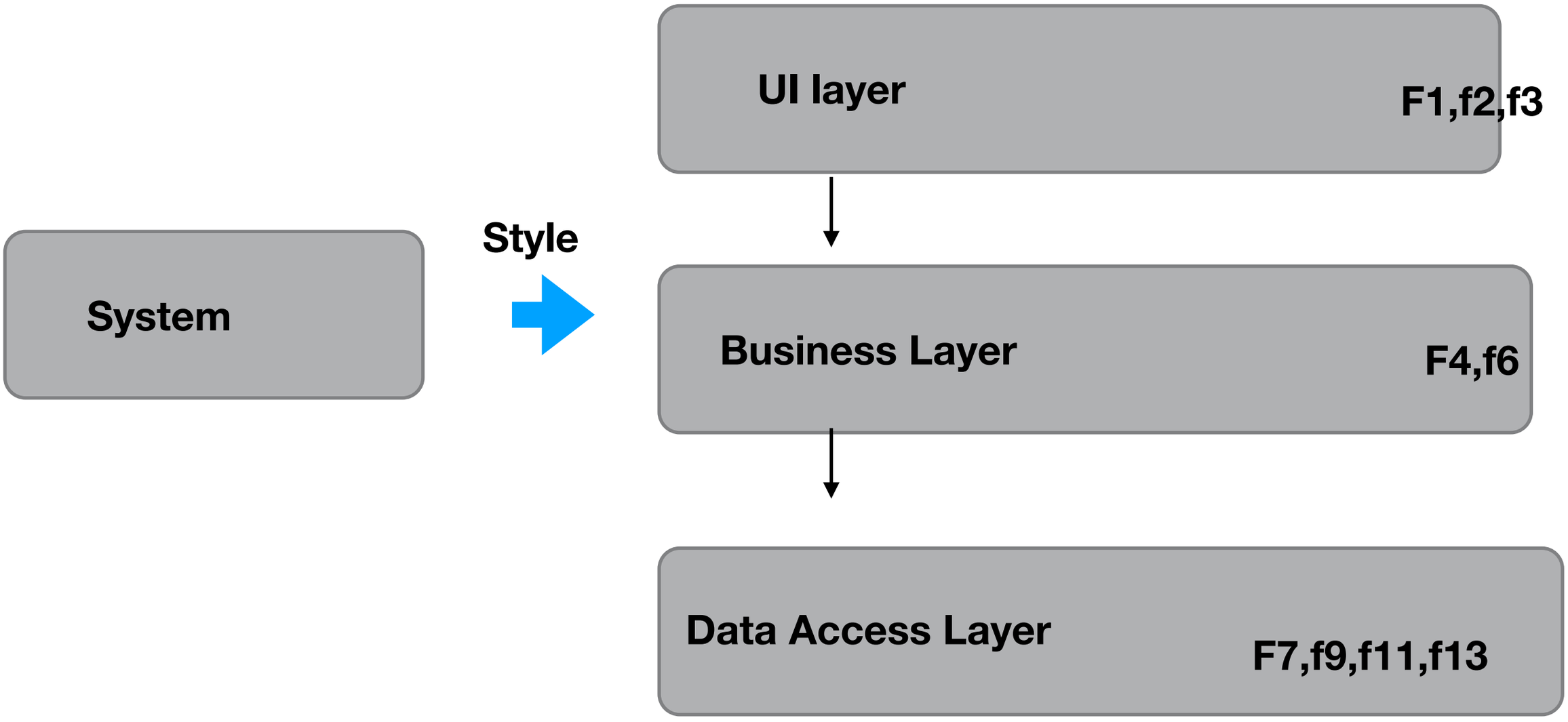
Decompose system



Architectural Styles

- Layered
- Pipes and Filter
- Event Driven Architecture
- Microservice
- CQRS
- Cloud
- Service Oriented Architecture
- Product Line Architecture
- Model Driven Architecture

Layered


UI layer
F1,f2,f3
Style
System
Business Layer
F4,f6
Data Access Layer
F7,f9,f11,f13

Pipes And Filter

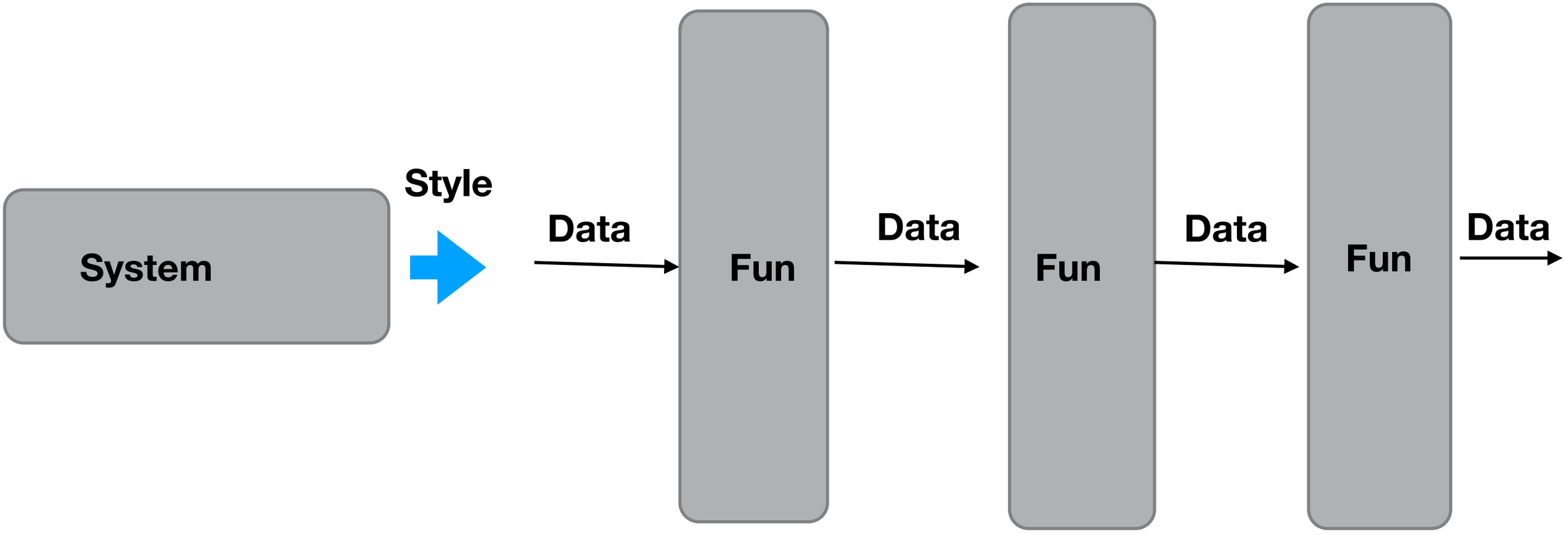
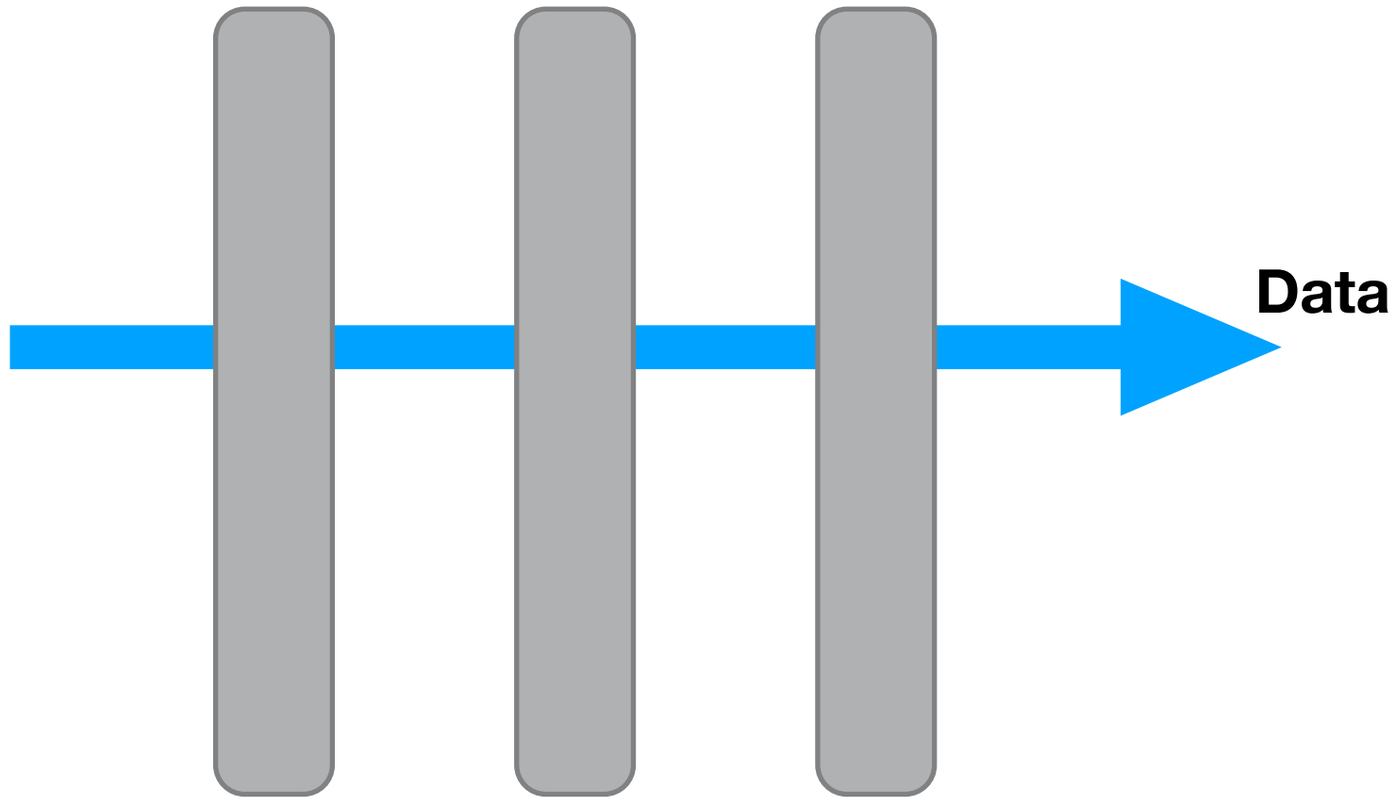


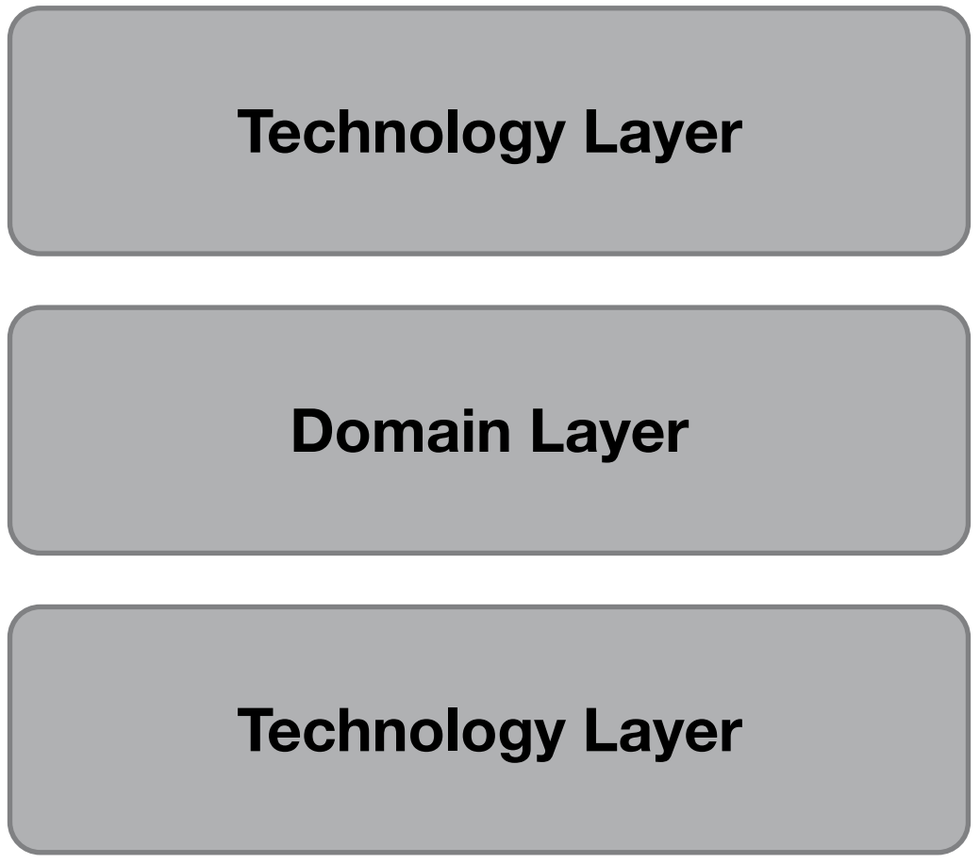
Image processing

- Extract Feature
- OCR
- Mask
-



Seperation concerns

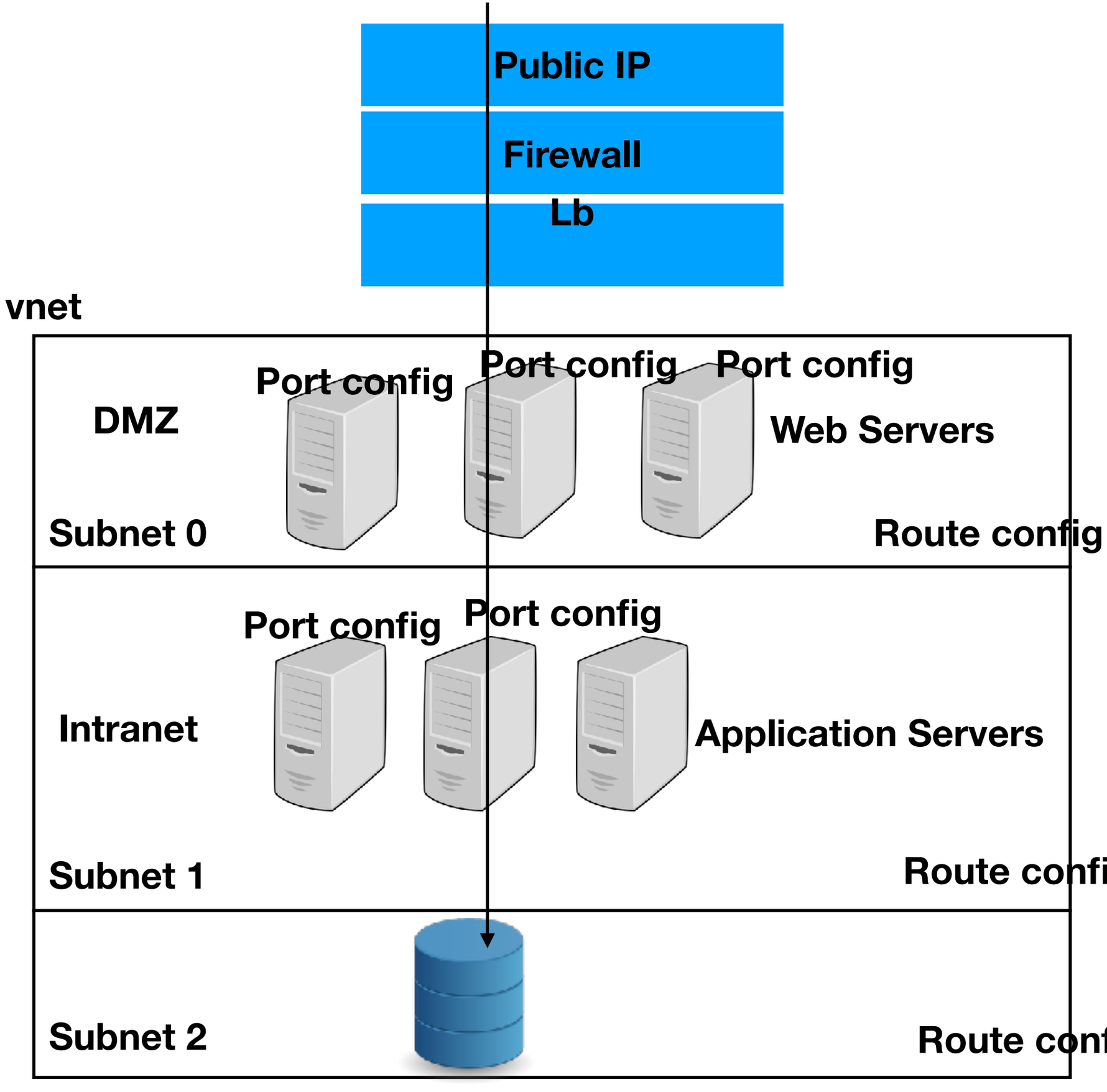
- Separate Technology logic from domain logic



Architectural Pattern

- MVC
- Broker
-

Performance	Maintainability	Security	Reliability
Caching	Message q (low coupling)	AAA	Message q
Clone (load balance)	Layered	Input validation	ACID
Split	Hexagonal Onion Ring Boundary Control Entity	Exception handling	
Sharding		Session mgmt	
Message q	Pipe and Filter	Secret mgmt	
Stateless	Containers # cheaper isolation # reproducable env # Monitoring # easy deployment	Data Security	
Object pooling		Vent, subnet	
Lazy loading		Firewall	
Chunking (batch)		Load balancer	
Eager Loading		Route config	
Compression			
Containers #scale			

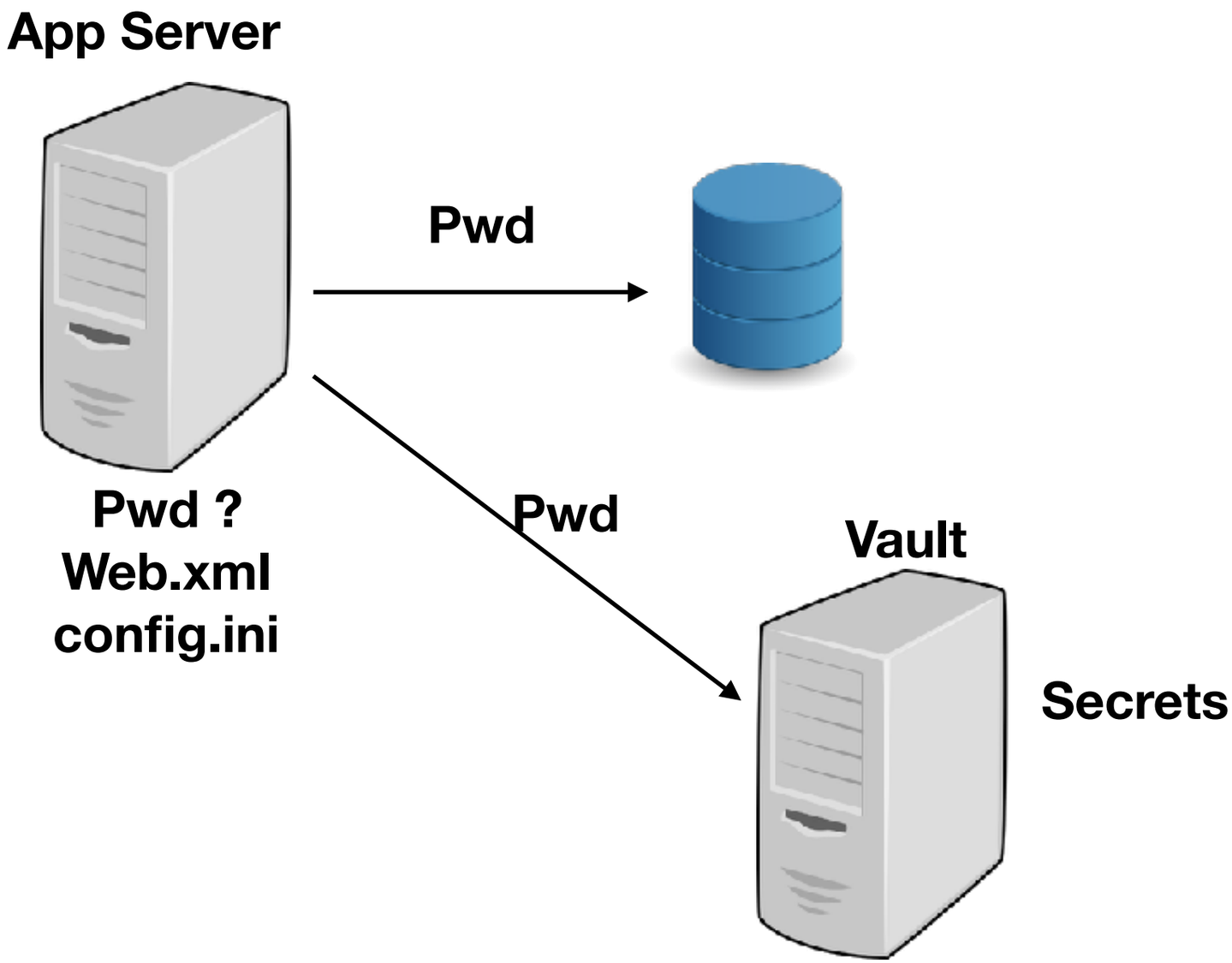

Public IP
Firewall
Lb
vnet
Port config
Port config
Port config
DMZ
Web Servers
Subnet 0
Route config
Port config
Port config
Intranet
Application Servers
Subnet 1
Route config
Subnet 2
Route config

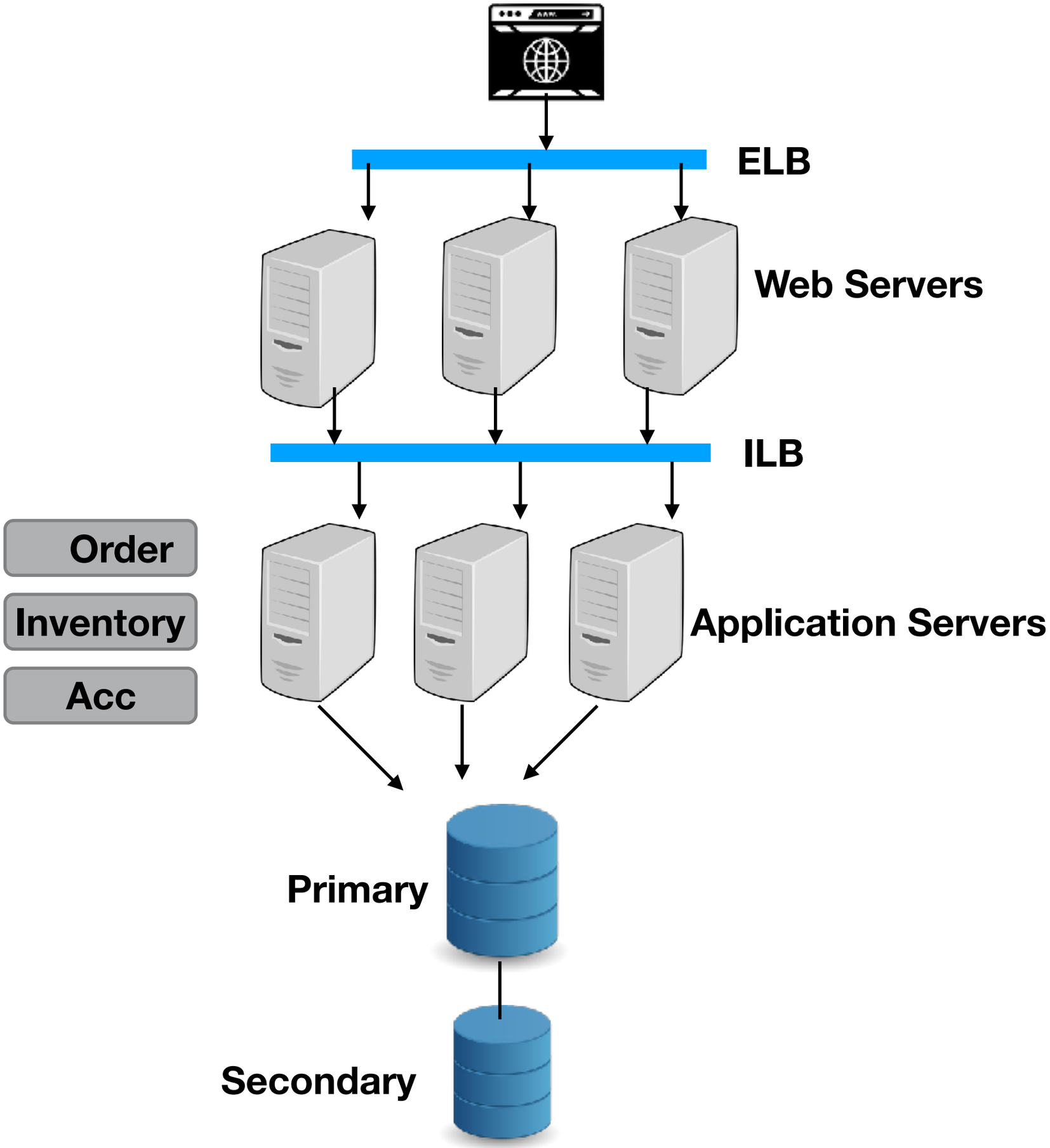
Application Security

- Authentication (First Defense) “who are you”
    - By what you know (pwd, secret, api key)
    - By what you have (otp, email, cert, rsa)
    - By what you are (face, voice, retina, …)
    - 2 factor
    - Multi factor
- Authorization “what can you do” (RBAC)
- Audit (Last Defense) “what did you do”
- Data Security
    - In Transit (SSL)
    - In Rest (Hash, Symmetric, Asymmetric)
- Input Validations
- Exception management
- Session management
- Secret management

Authentication

- Pwd
- otp
- API key
- Cert
- Secret question
- Bio metric (face, retina, voice, finger print)


App Server
Pwd
Pwd ?
Web.xml
config.ini
Pwd
Vault
Secrets


ELB
Web Servers
ILB
Order
Inventory
Acc
Application Servers
Primary
Secondary

ELB

Cluster

Web

Web

Order

Order

Acc

Acc

Acc

Inventory

Inventory

Inventory

Inventory

Primary

Secondary

VM1

VM2

Container 1

Container 2

My app 1

Middlewear

Cpu
RAM
Disk
I/o

Kubernetes (slave)

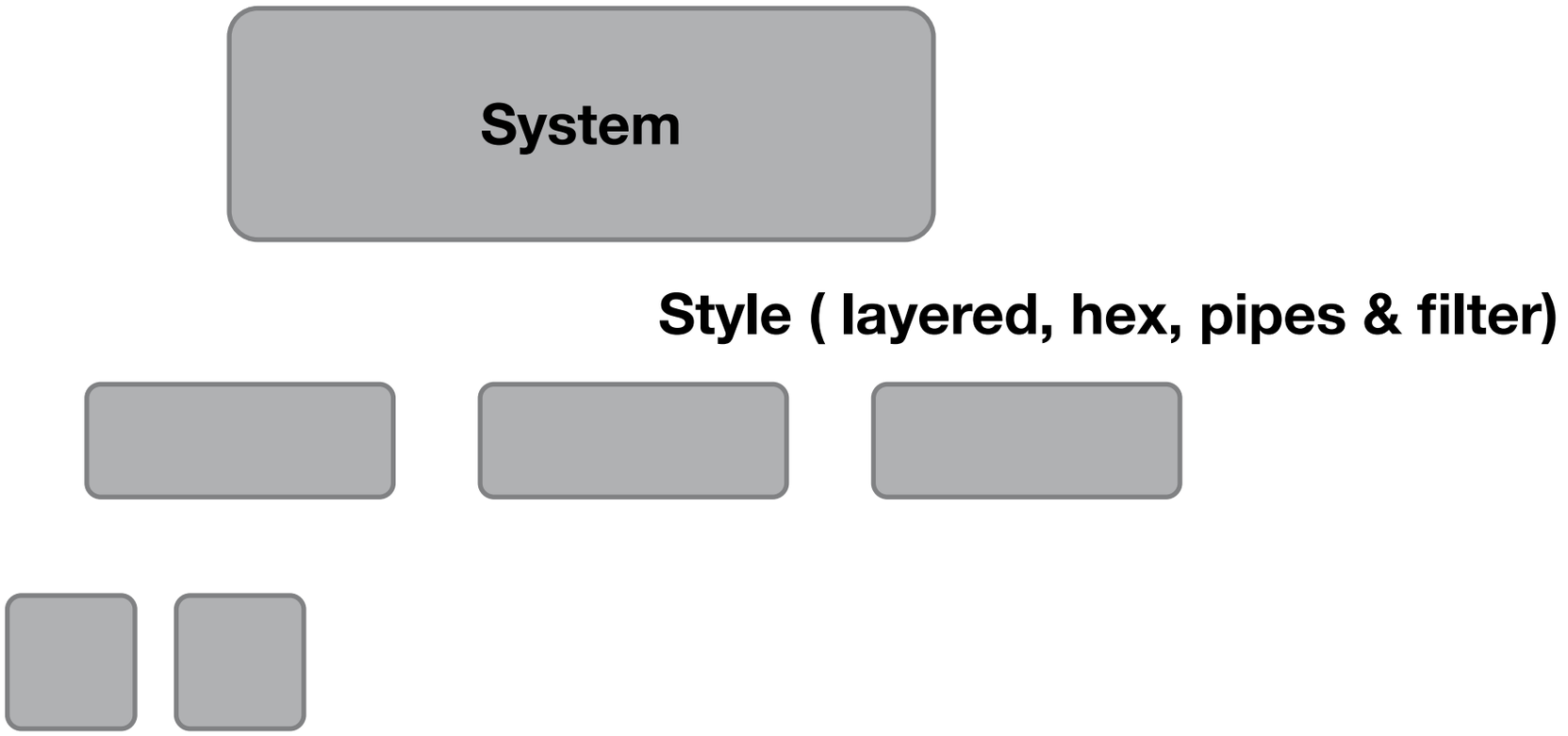
Containerization layer (Docker)

Unix Guest Os

Kubernetes (Master)

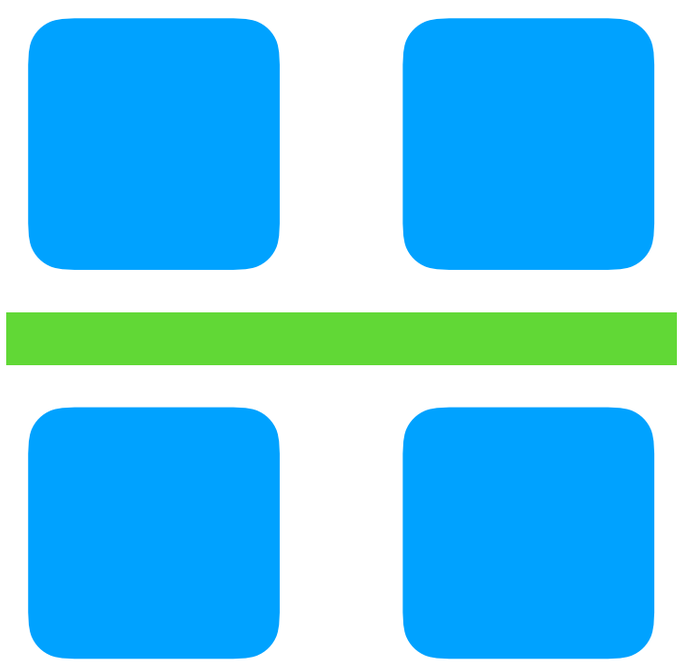
Kubernetis Swarm

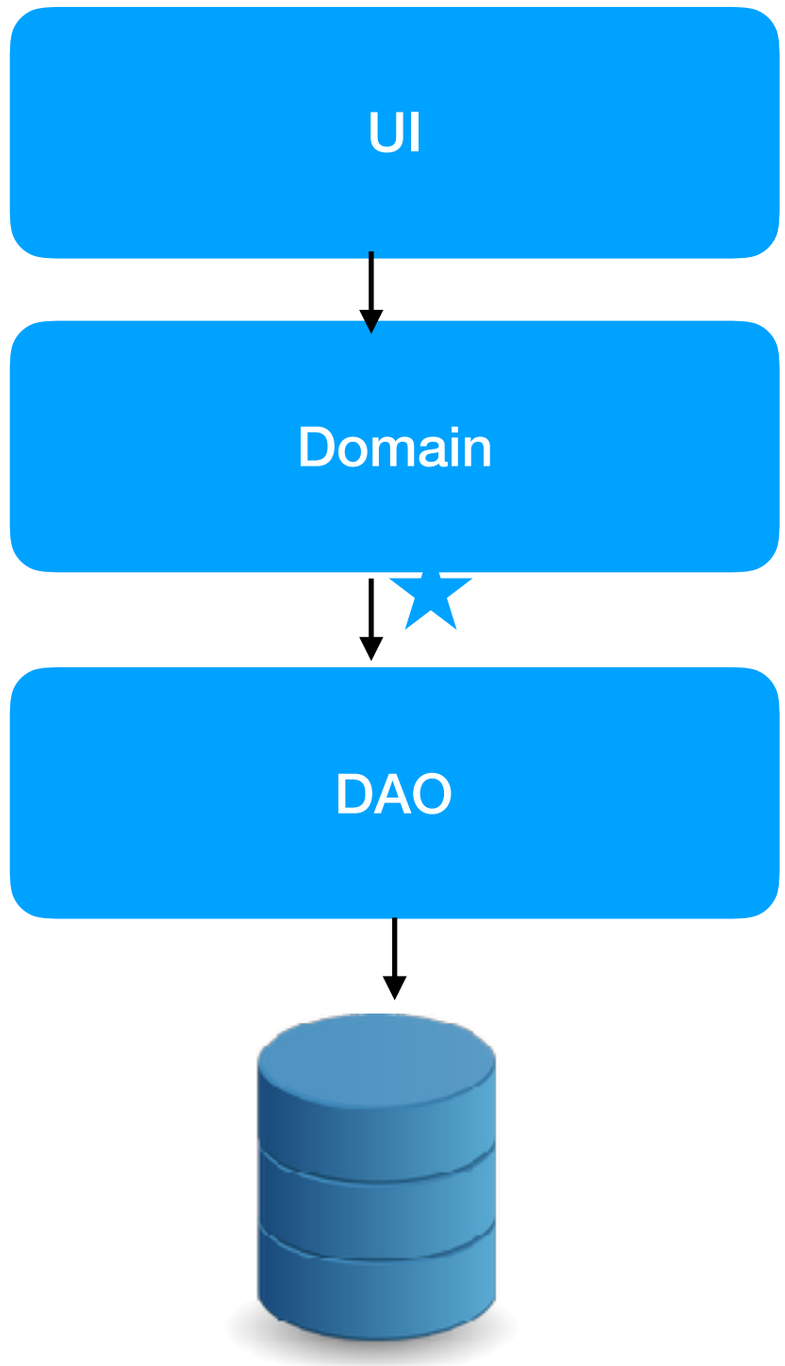
decompose system



Message Q (db)

Gain	Pain	
Load Leveling (Scalability) # eventual consistency	Duplicate msg #Idempotency	
Reliable Call # ack	Unordered delivery	
Event Driven Architecture (Maintainability) # low coupling	One way # error # result	
Distributed transaction	backup/ recovery	
	Compensation logic	




UI
Domain
DAO

Entry

UI Technology

MSG Technology

API Technology

Control (Application) Layer

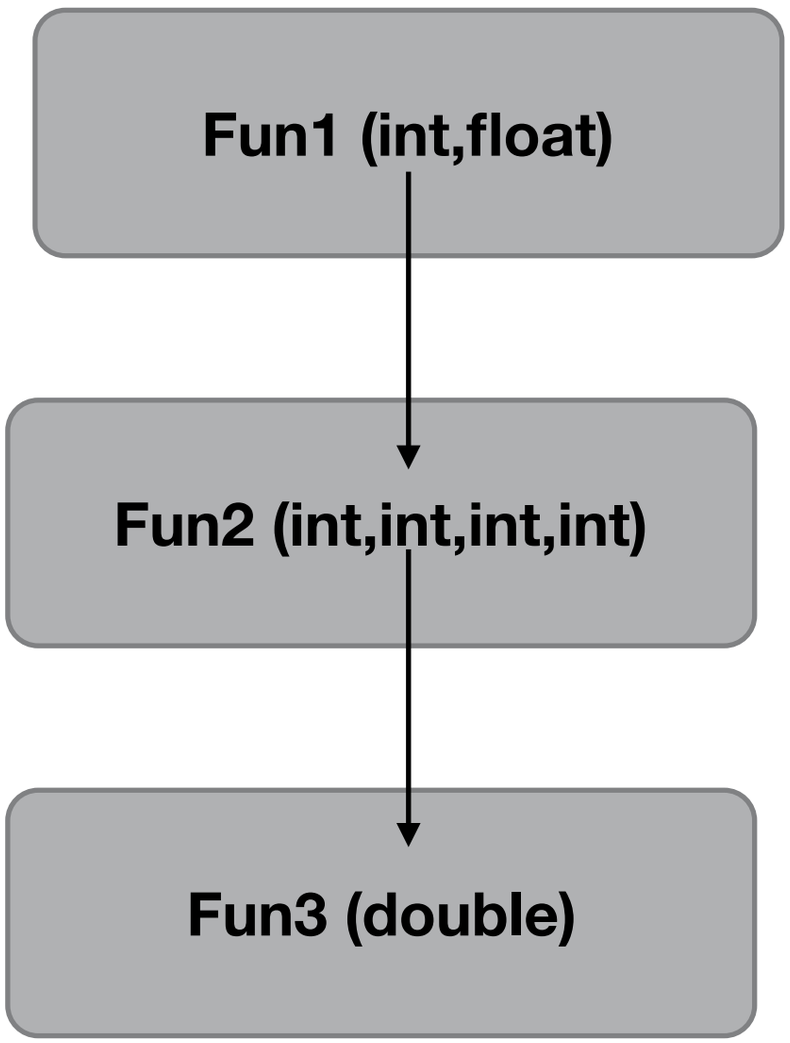
Domain Logic

Proxy Technology

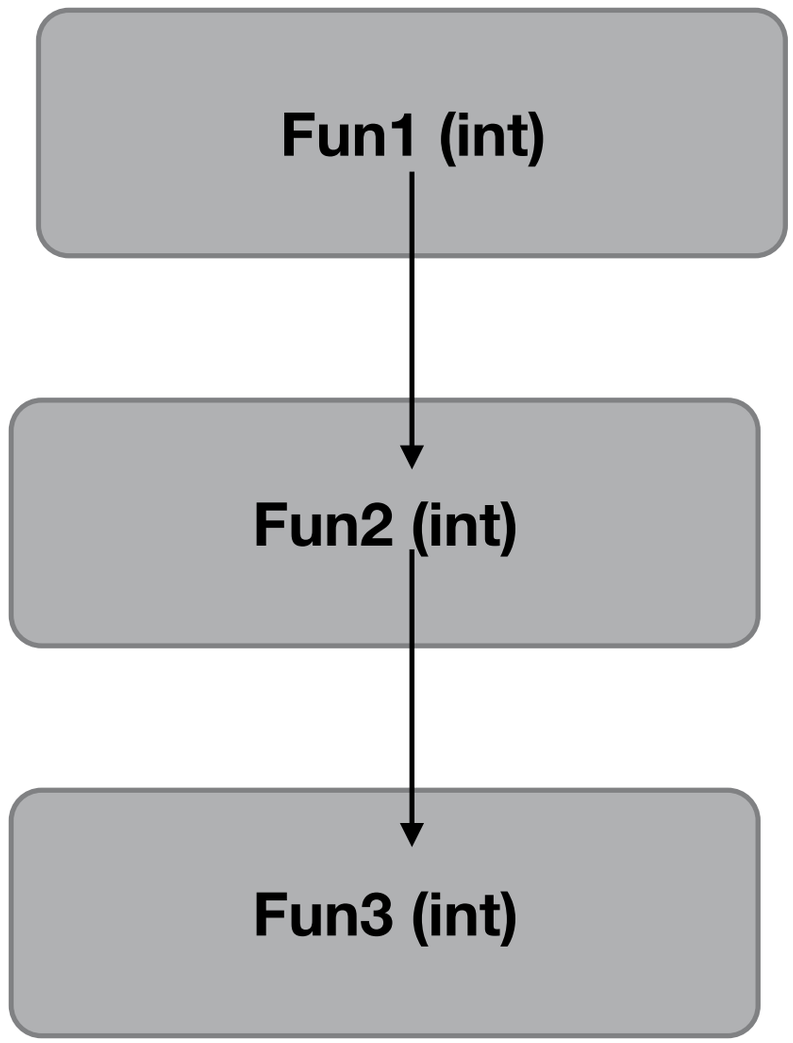
Persistence Technology

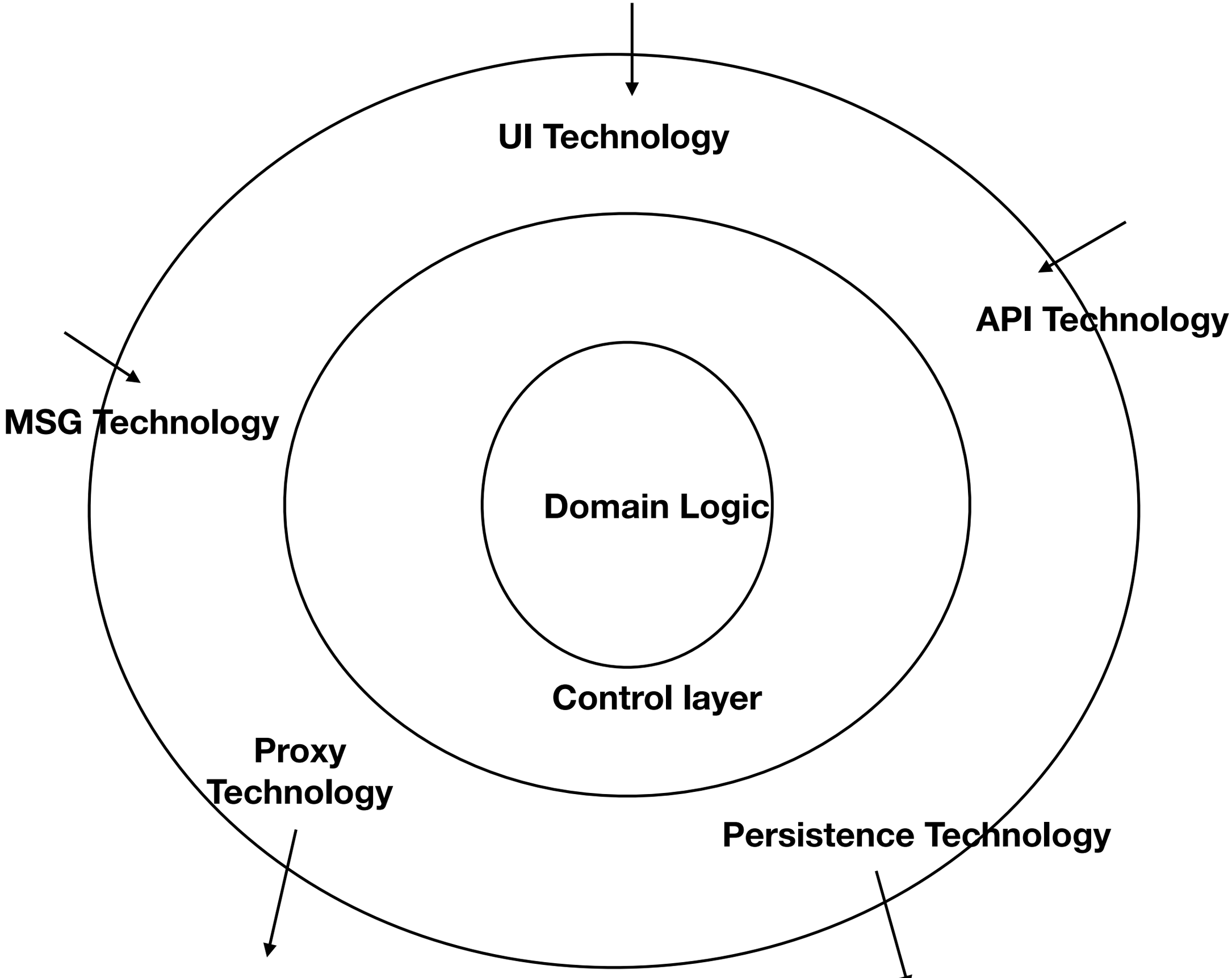
Exit

Layered



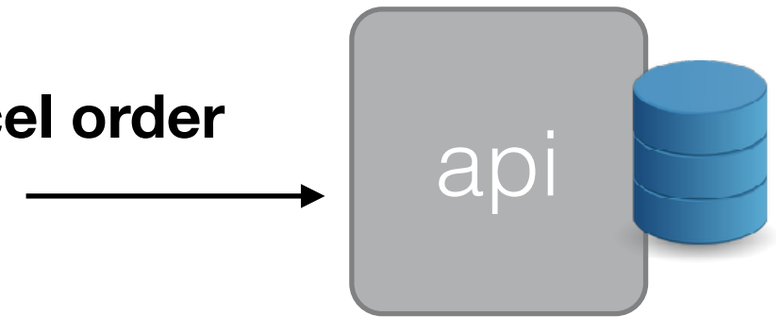
Pipes and filter

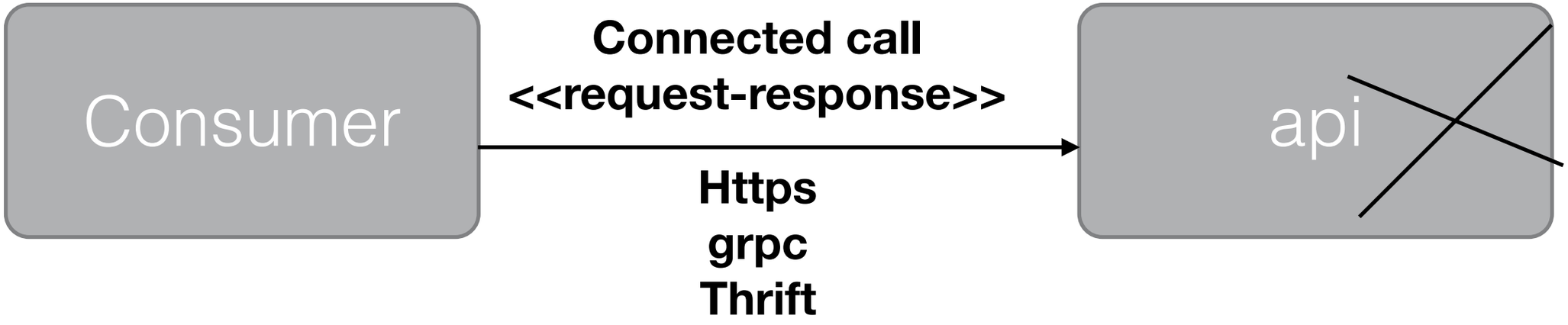


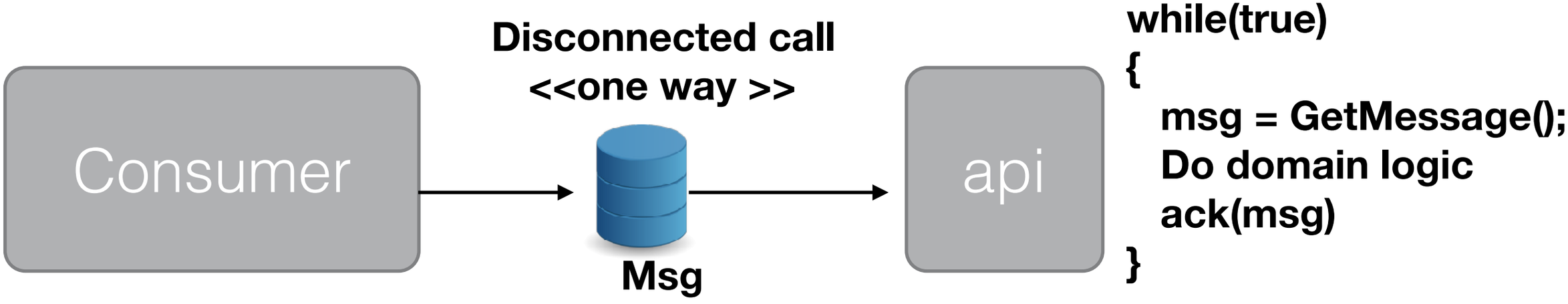

UI Technology
API Technology
MSG Technology
Domain Logic
Control layer
Proxy
Technology
Persistence Technology

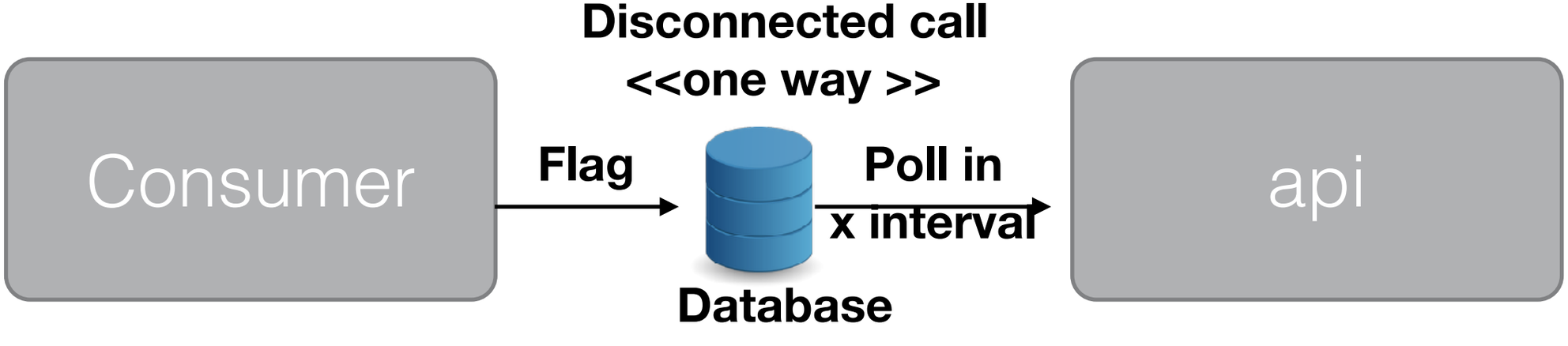
Create order

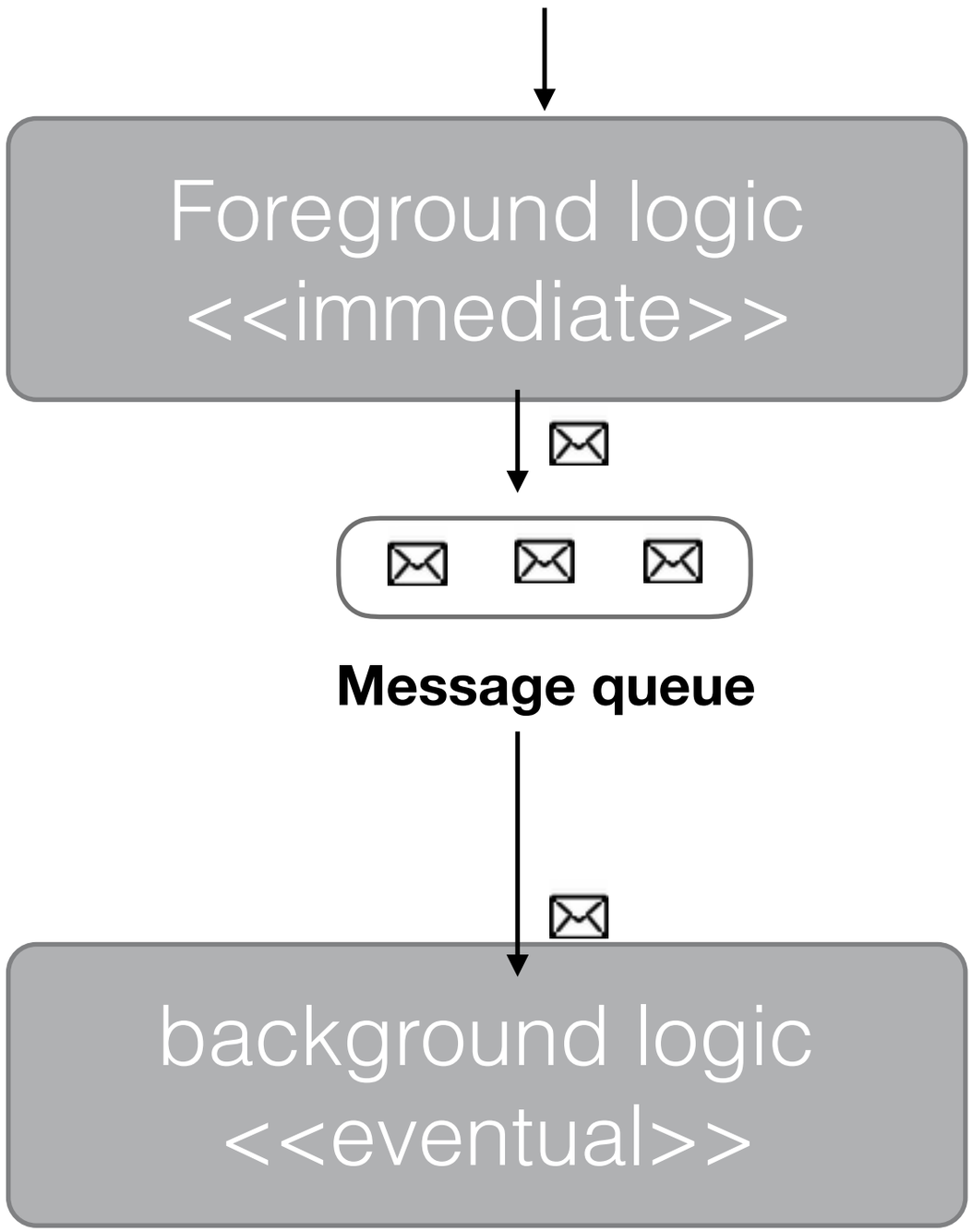
Cancel order



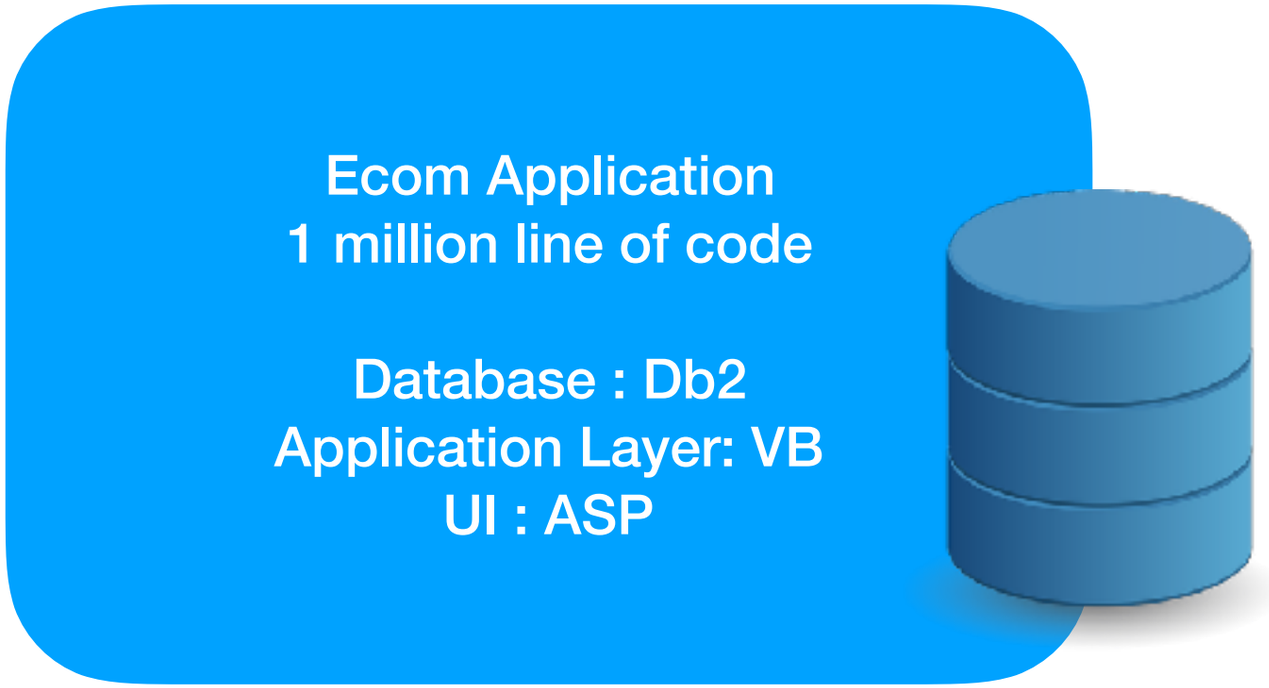

Consumer
Connected call
<<request-response>>
Https
grpc
Thrift
api


Consumer
Disconnected call
<<one way >>
Msg
api
while(true)
{
msg = GetMessage();
Do domain logic
ack(msg)
}

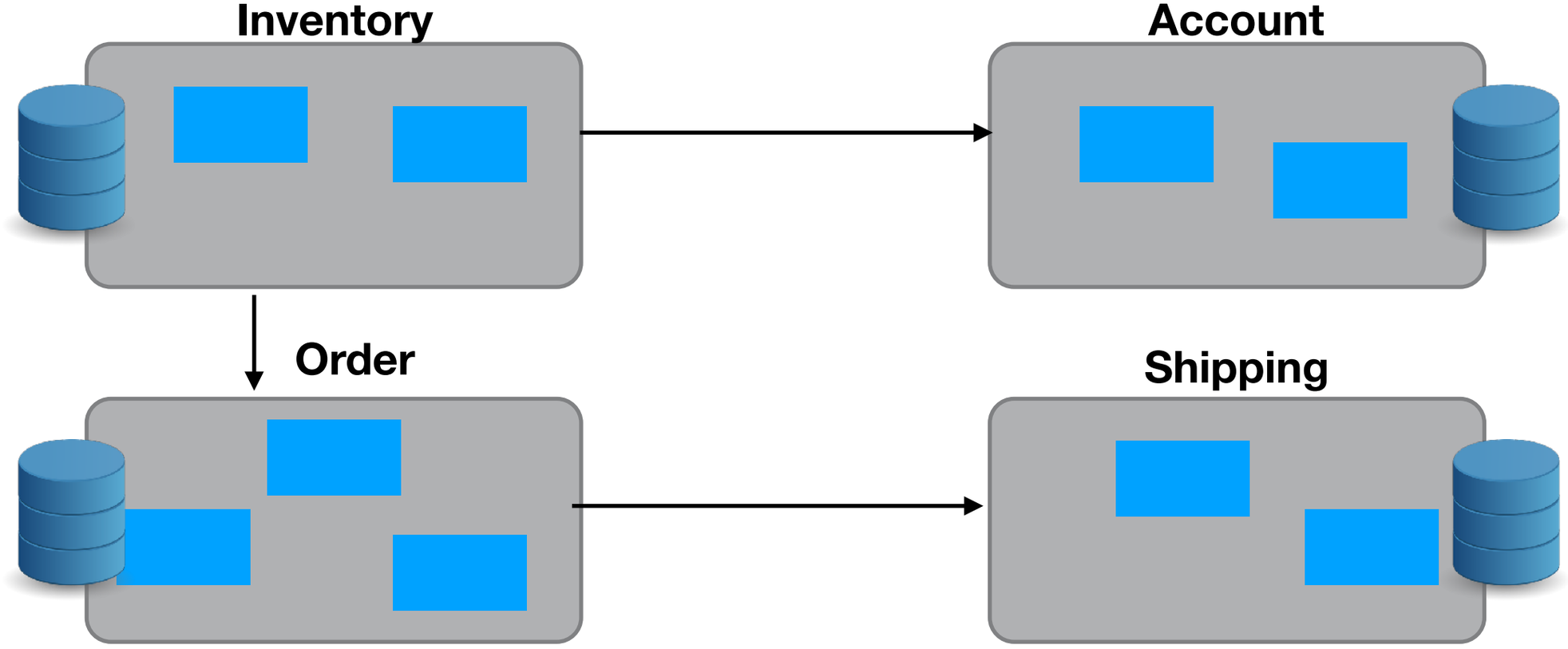

Consumer
Disconnected call
<<one way >>
Flag
Database
Poll in
x interval
api

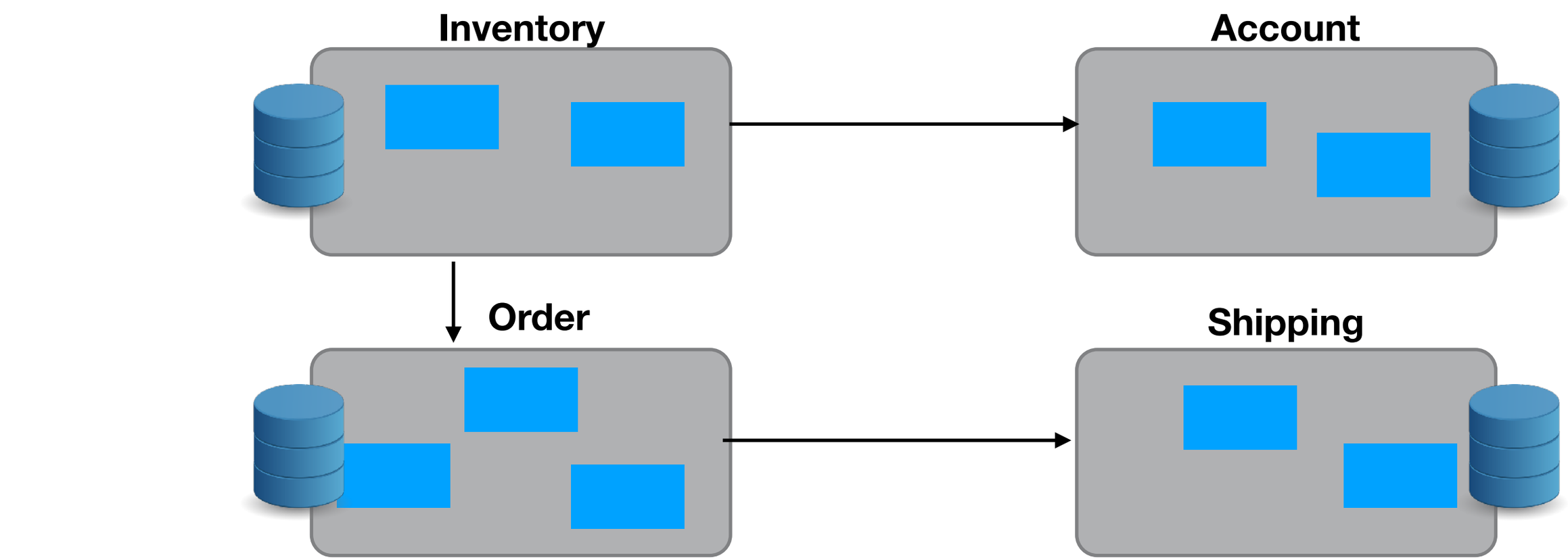

Foreground logic
<<immediate>>
Message queue
background logic
<<eventual>>

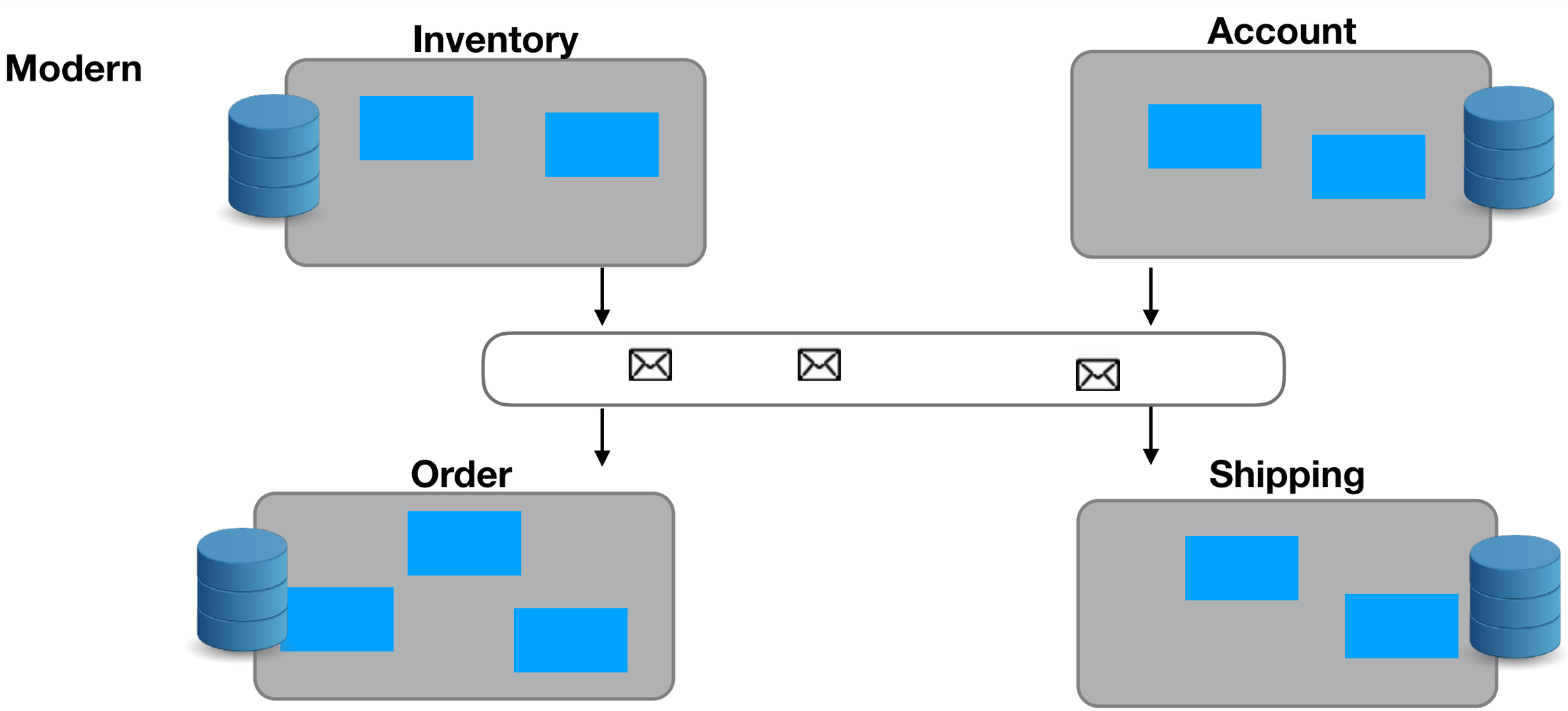
Legacy

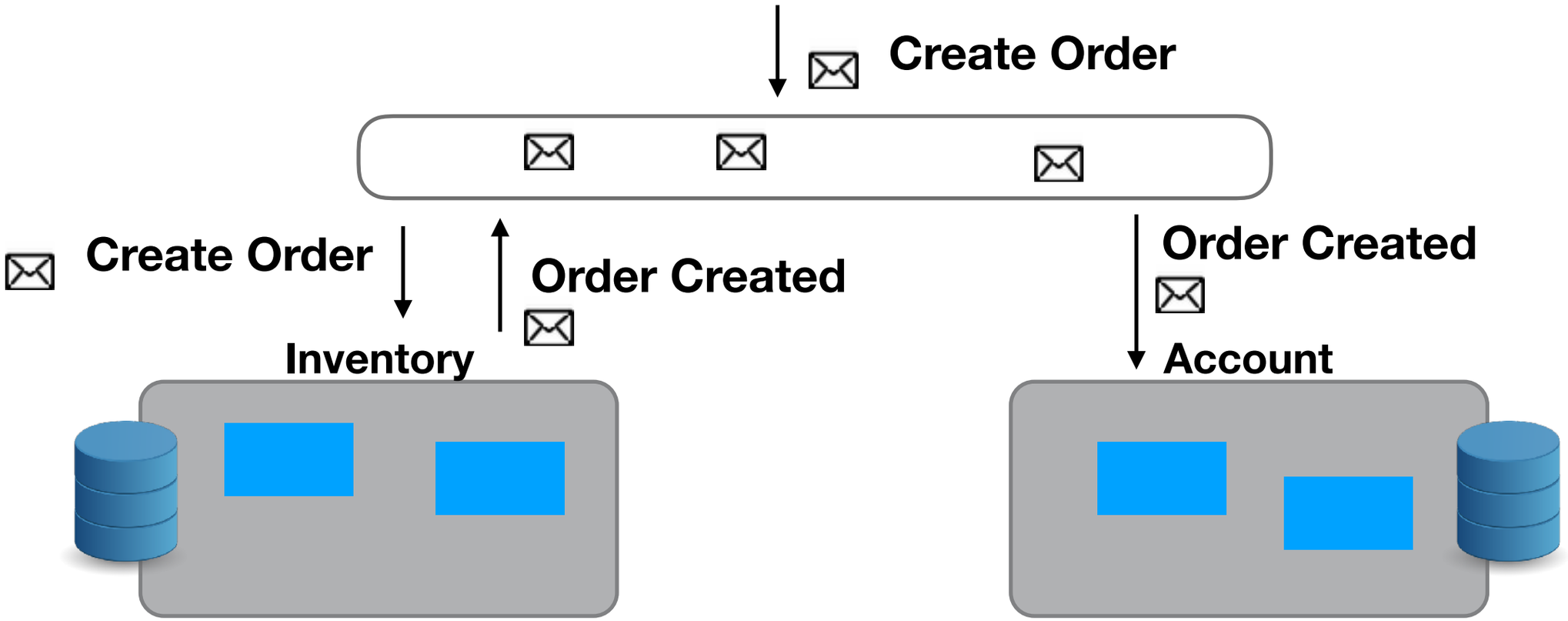


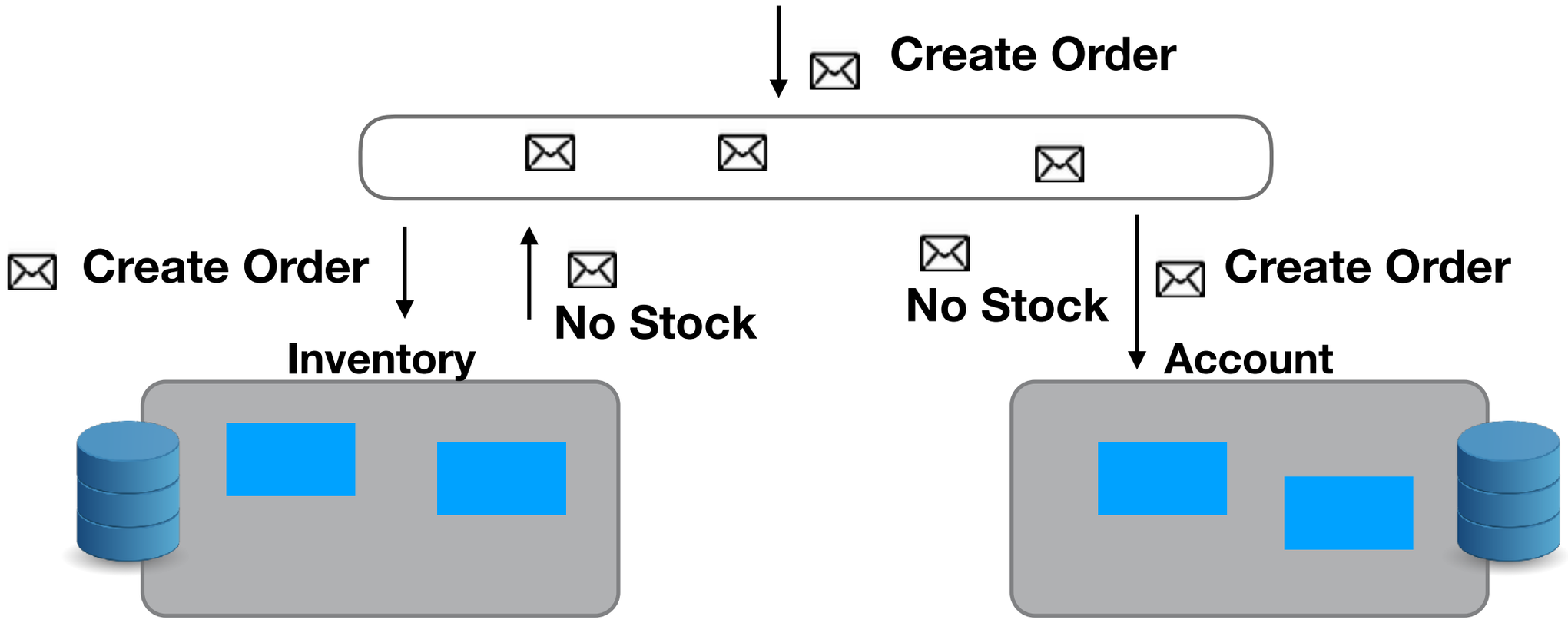
Modern




Inventory
Account
Order
Shipping


Modern
Inventory
Account
Order
Shipping

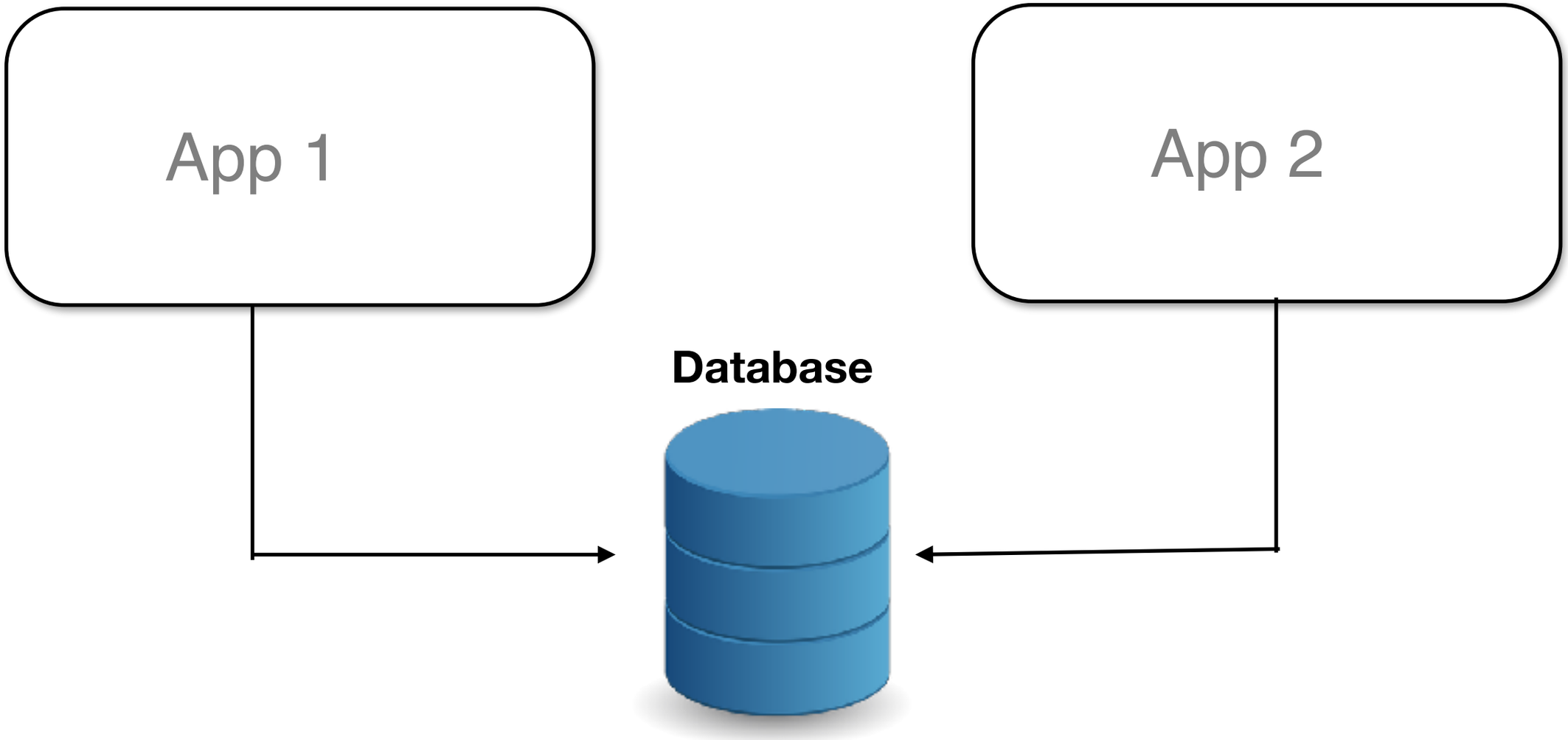

Create Order
Create Order
Order Created
Order Created
Inventory
Account

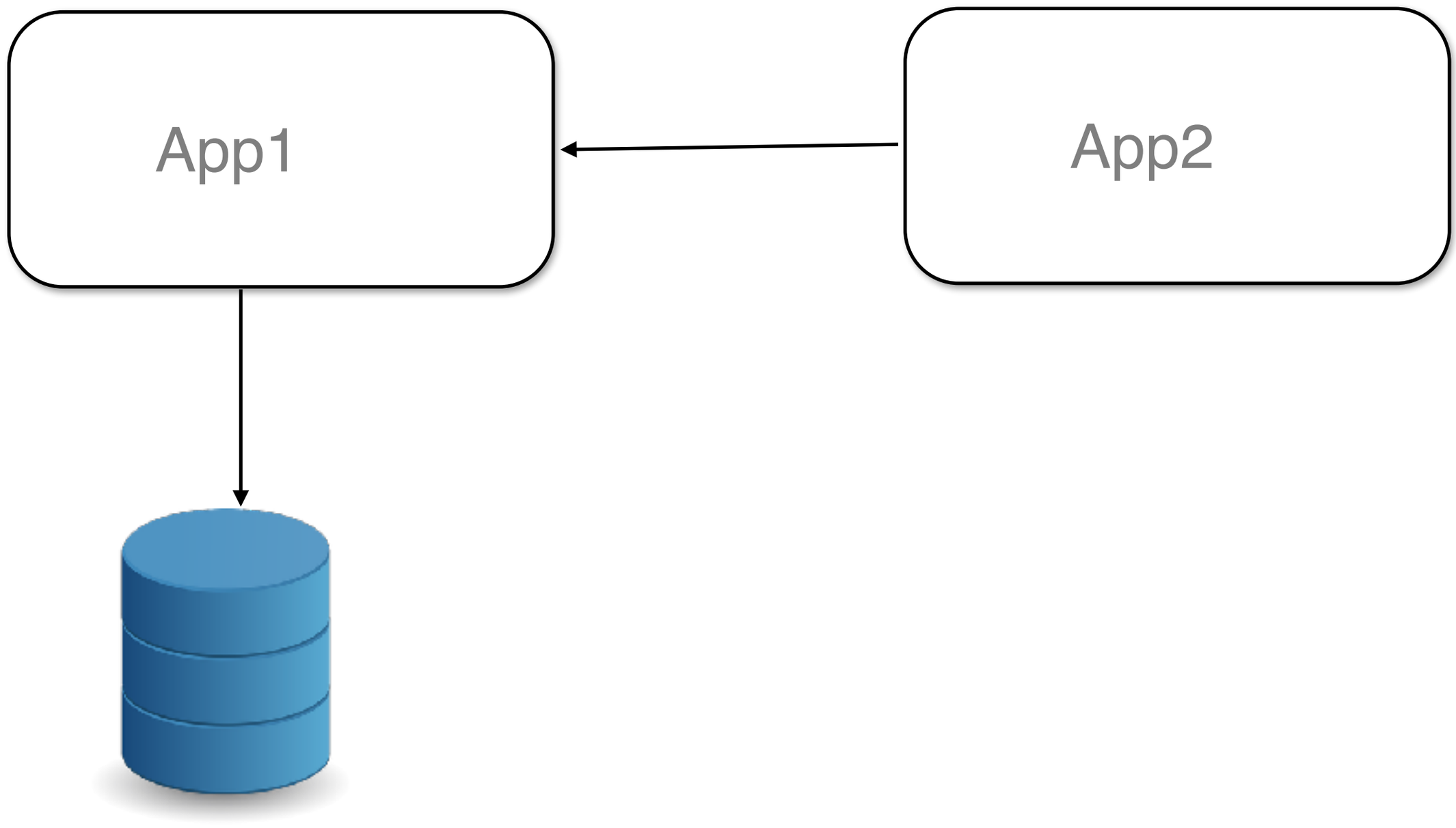


Compensation / undo

Concerns	Legacy	Modern Microservice
Performance	++	- - (high impact)
Scalability	- -	+ +
Availability	- -	+ +
Security	++	- -
Deployment	+ +	- -
Environment Cost/Setup	+ +	- -
Reliability	+ +	- -
Debugging	+ +	- -
Integration test	+ +	- -
Unit test	- -	++
Agility to change a part (incremental change)	- -	+ +
Feature shipping	- -	+ +

Maintainability (change) :




App1
App2

Modernising a legacy system

Banking Application
1 million line of code

Database : Db2
Application Layer: VB
UI : ASP

Database : Oracle
Application : Python
UI : React

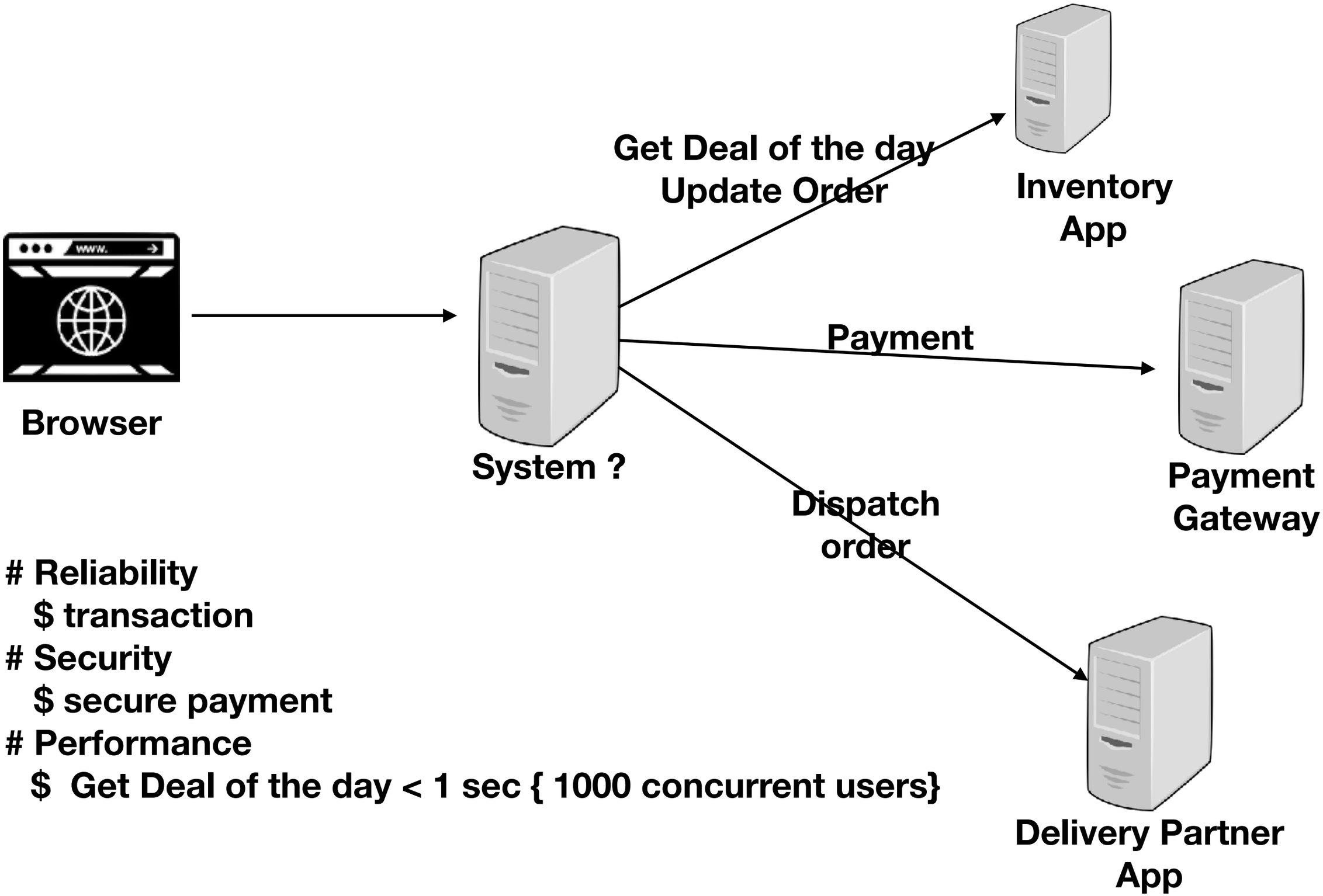
Quality requirements

 # quality : Reliability
 # Source : Consumer Web site
 # Stimulus : purchase order request
 # Artifact : to the XYZ Application
 # Environment : Duplicate request
 # Response: The XYZ receives the duplicate request, but the consumer is not double-charged, data remains in a consistent state, and the Consumer Web site is notified that the original request was successful.
 # measure : PBF 0.001

The Consumer Web site sent a purchase order request to the XYZ Application. The XYZ processed that request but didn’t reply to Consumer Website within five seconds, so the Consumer Web site resends the request to the XYZ. The XYZ receives the duplicate request, but the consumer is not double-charged, data remains in a consistent state, and the Consumer Web site is notified that the original request was successful.

Deal of the day

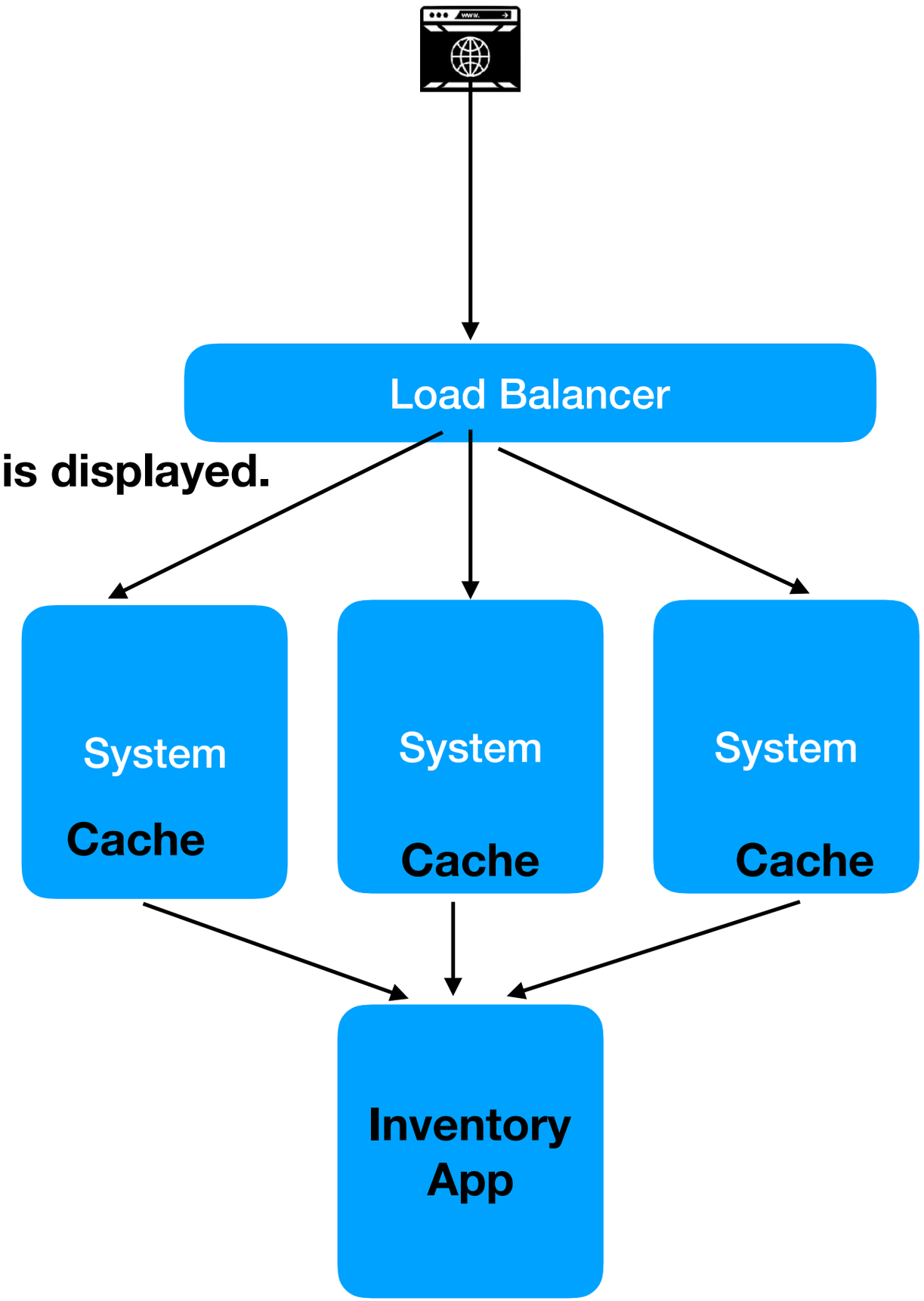
View the Product of the day
Order the product



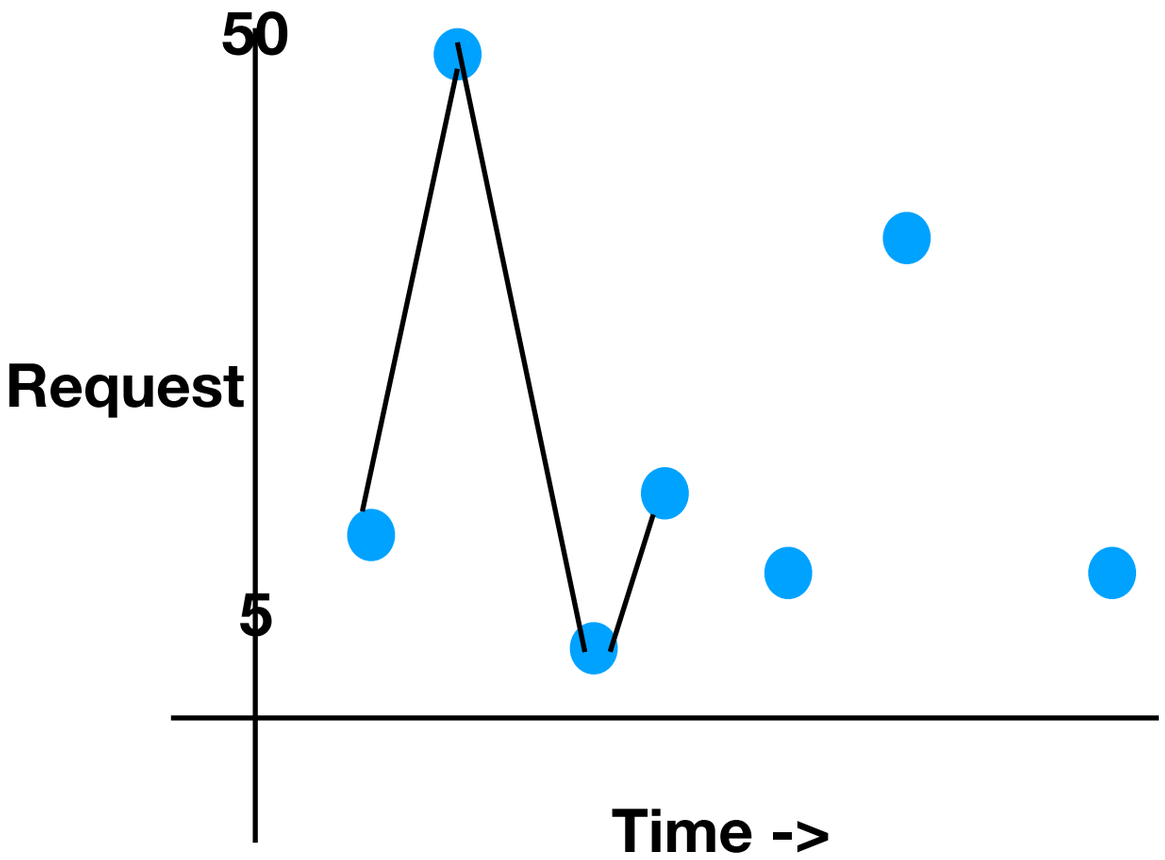
# Reliability
  $ transaction
# Security
  $ secure payment
# Performance
  $ Get Deal of the day < 1 sec { 1000 concurrent users}

Architectural Requirement
 # quality : Performance
 # Source : customer
 # Stimulus : request view product page
 # Artifact : Deal of the day Web App
 # Environment : 1000 concurrent users
 # Response: The deal of the day product is displayed.
 # measure : < 1 sec

Tapproach-> Quality
 # Cache -> performance
 # Clone + LB -> Performance, Availability
 # L3 FW, L7 FW -> Security
#

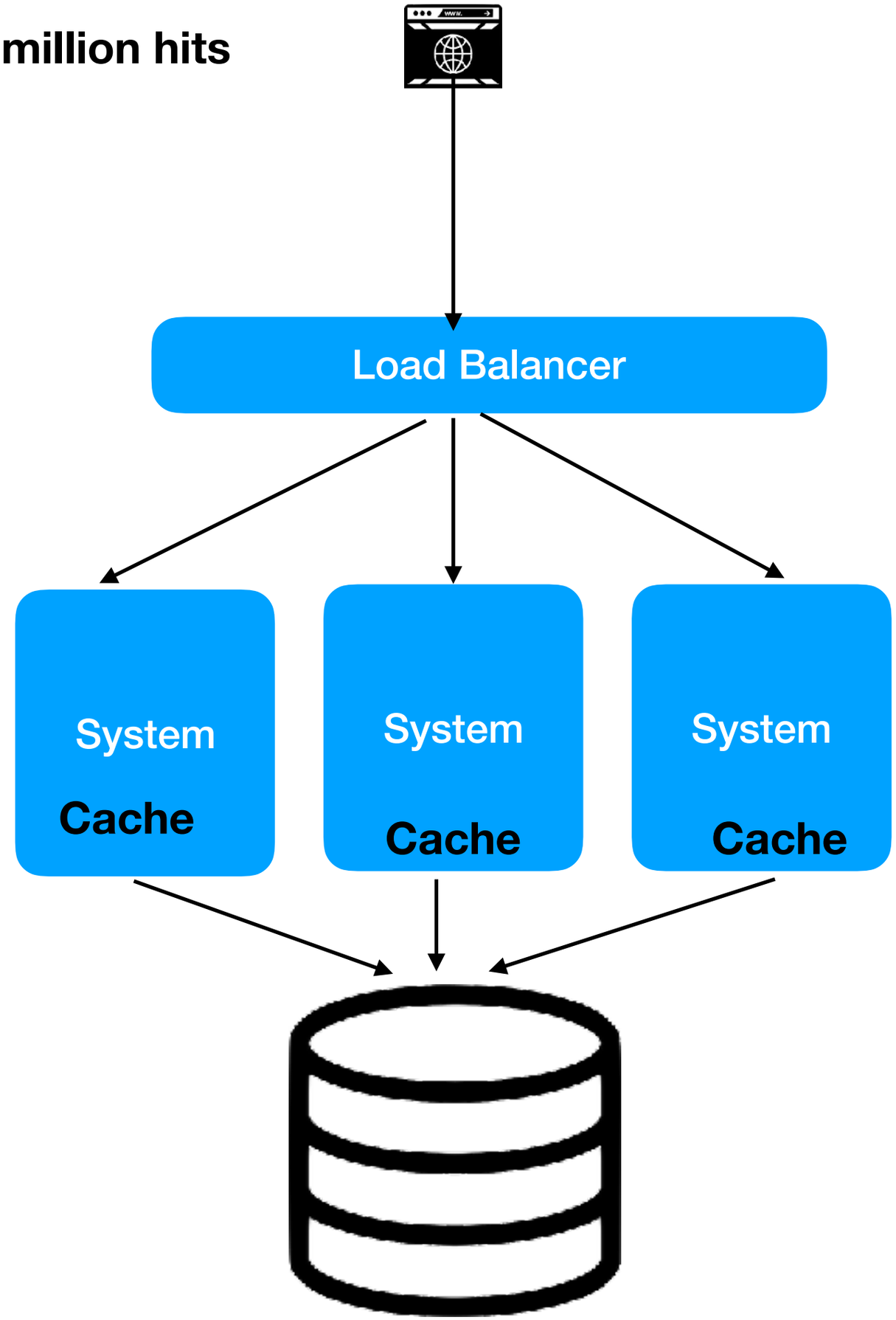


view Deal of the day
create order

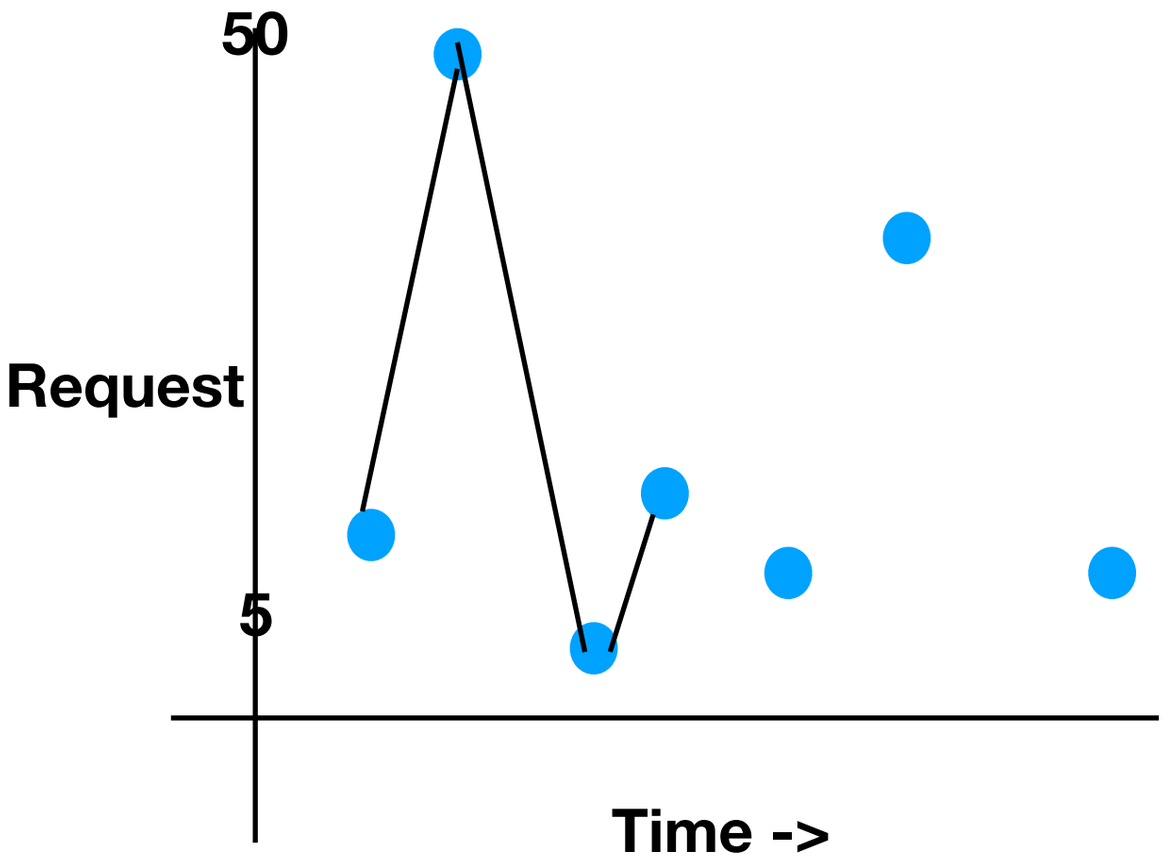


@ eventual consistency
@ stateless

1 million hits



Load Leveling



Browser

<— 4

expires header, E tag
Indexdb
Local storage (key -value)
Session storage (key -value)

<— 6

CDN

<— 3

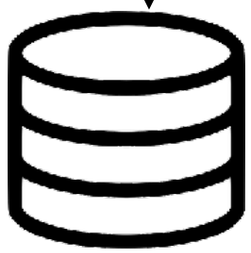
Web server

<— 5

Cache server

<— 2

App server



<— 1

Db server

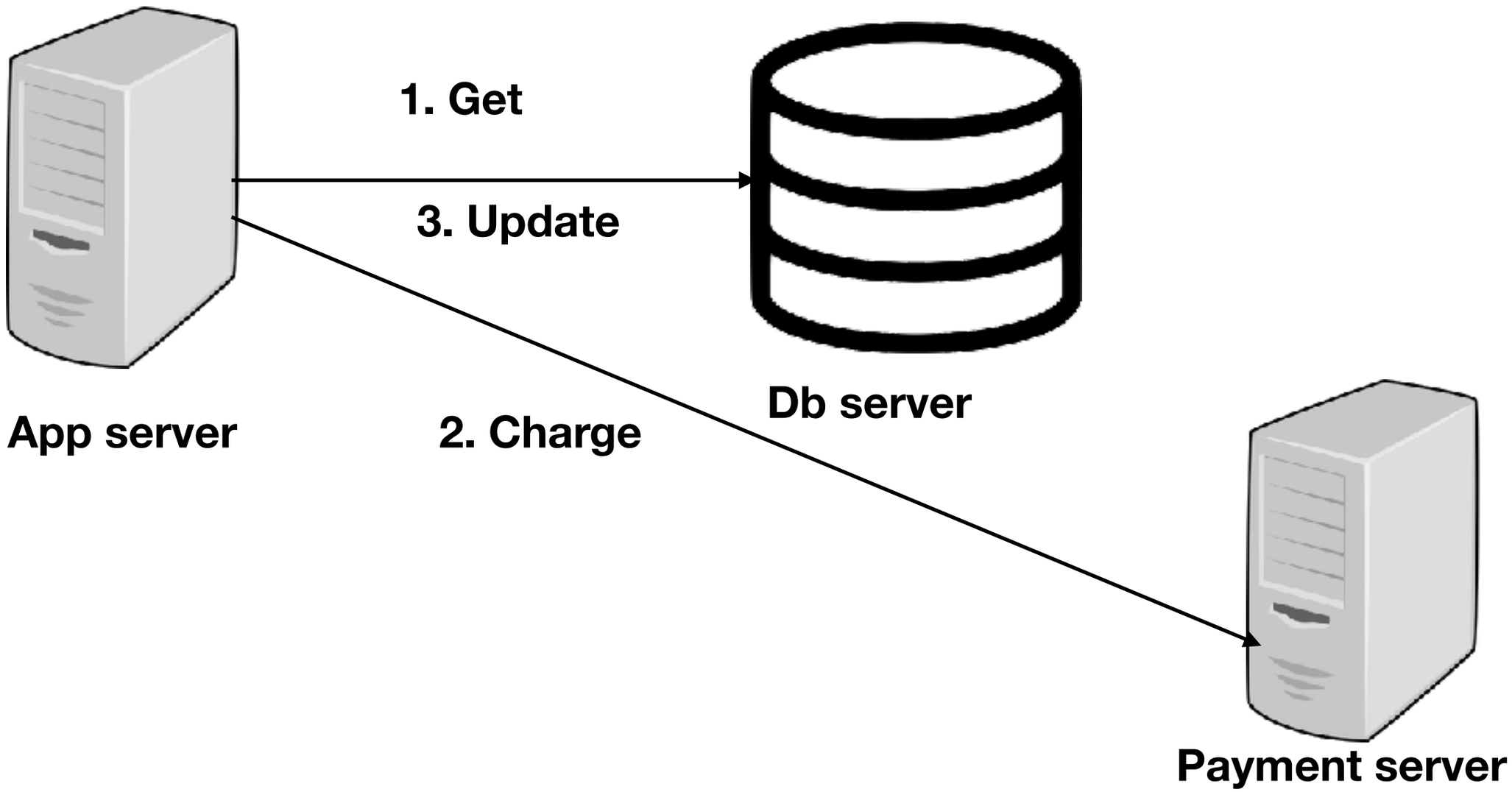
Cache

When to expire cache ?
“one of the most difficult problems in computer science”

Data cache
Page cache

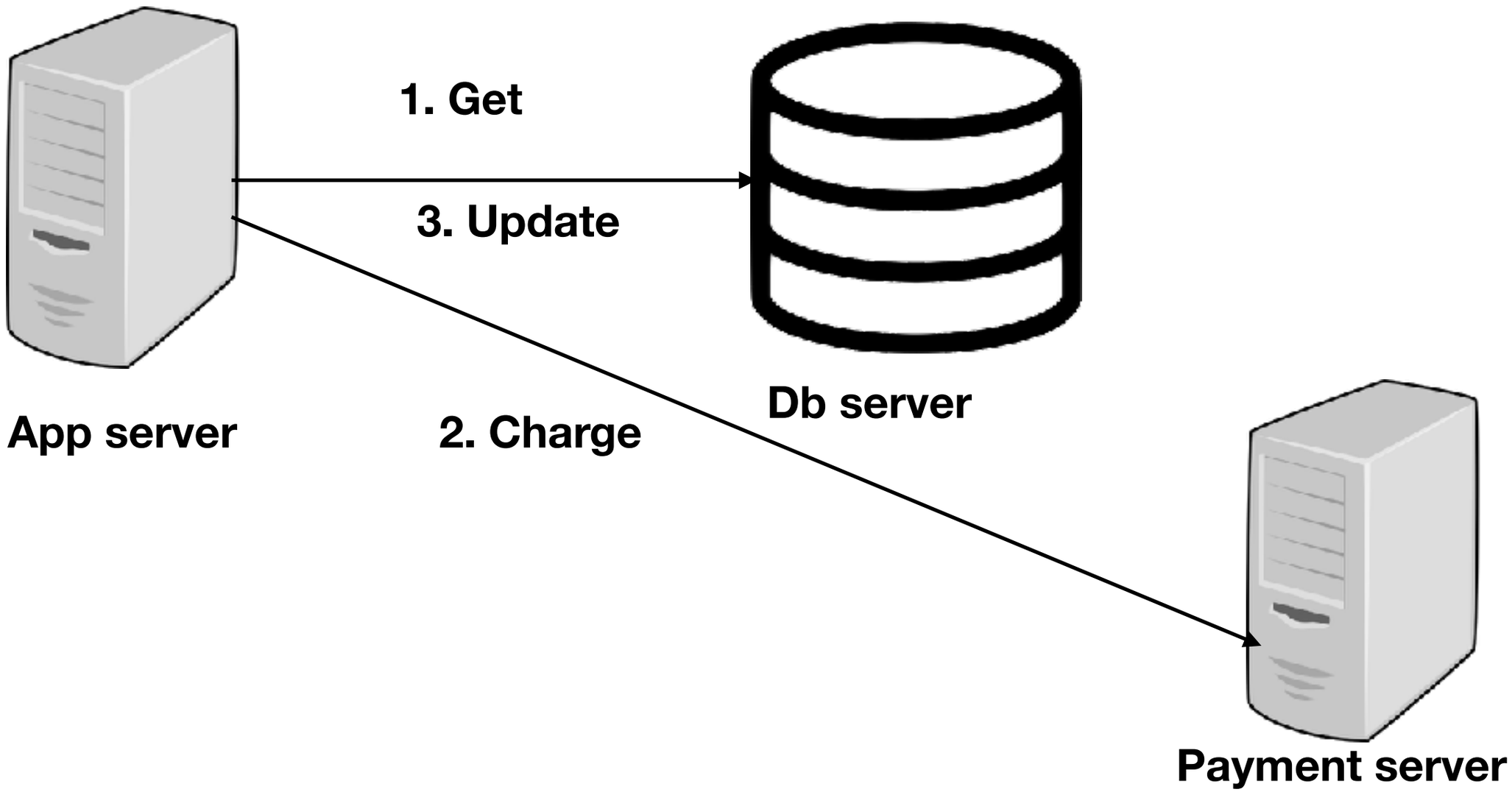
cron job invokes job every 30 days

job()
{
  1. For each Credit card details (get from your db)
      2.Charge card (call 3rd path payment gateway)
      3.Activate Account (update some column in your db)
}



cron job invokes job every 30 days

job()
{
1. For each Credit card details (get from your db where not started)
2. Set flag in transaction (acc, amount, time, status:started)
3.Charge card (call 3rd path payment gateway)
4. Set flag in transaction acc, amount, time, status:completed)
5. Activate Account (update some column in your db)
}



```
# cron job invokes job every 30 seconds
job()
{
        Foreach log in dir:
          Parse the log
          Extract key attributes using algo
          Write key attributes to db
          Move the file to cold storage
}
```

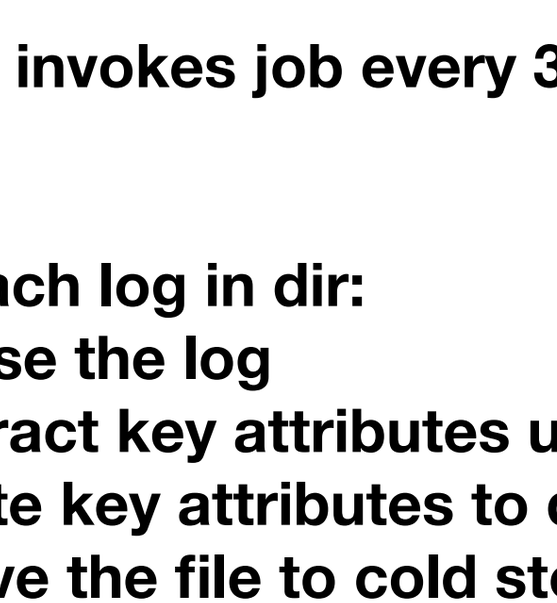
1 million
Devices

1 hour/ 10 mb / core

10 mb/day
Each file < 10 kb

Edge
Device

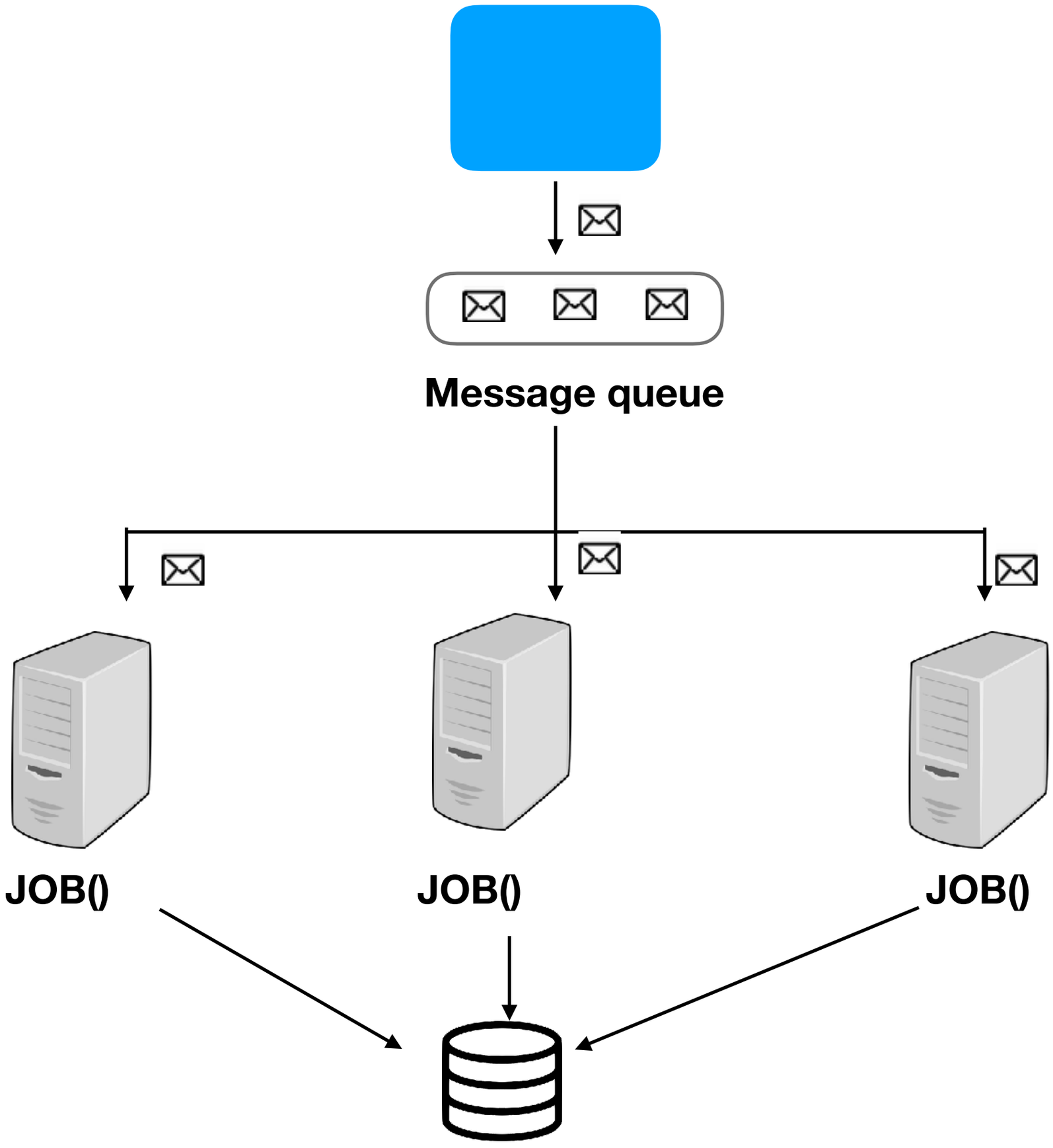
Telemetry logs



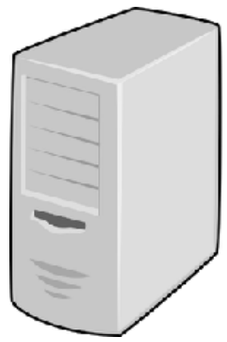
App server

Db server

Disconnected
Connected
Occasionally connected

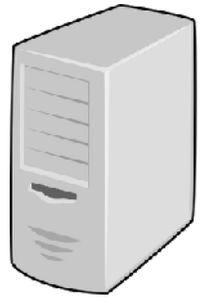

Message queue
JOB()
JOB()
JOB()

Each file < 10 kb



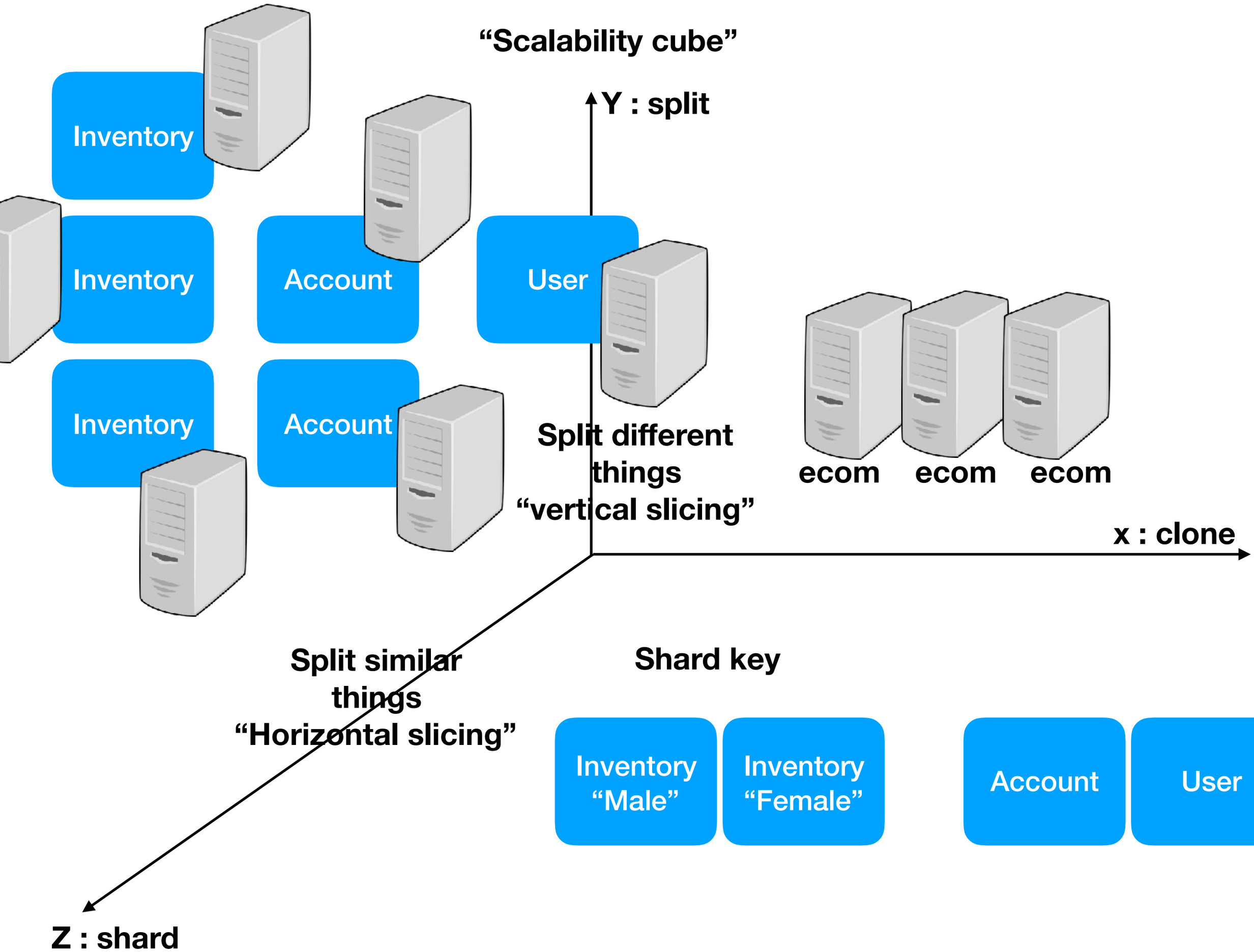
Vm1

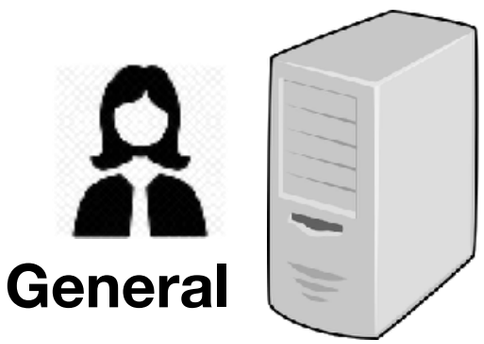
Vm2



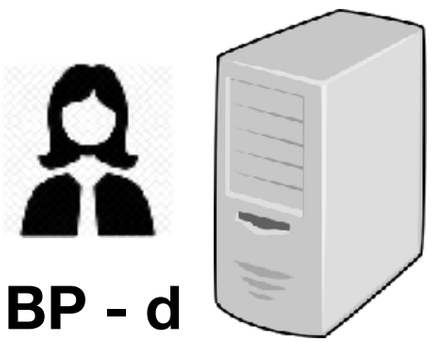
Vm3

```
# cron job invokes job every 30 seconds
job()
{
        Foreach log in dir:
           Parse the log
           Extract key attributes using algo
           Write key attributes to db
           Move the file to cold storage
}
```

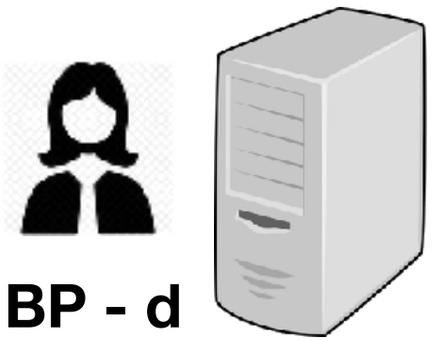

“Scalability cube”
Y : split
Inventory
Inventory
Account
User
Inventory
Account
Split different things “vertical slicing”
ecom
ecom
ecom
x : clone
Split similar things “Horizontal slicing”
Shard key
Inventory “Male”
Inventory “Female”
Account
User
Z : shard

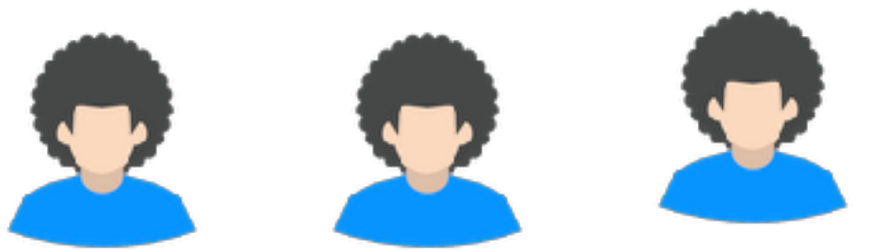

General


E
?
C


BP - d


BP : d
BP
BP : d


BP - d


BP : d
BP
BP : d


BP - I


BP : i

Quality attribute ?	Approach (tactic/style) ?	Measure ?
• Maintainability	• Modularization • Health monitoring • Config • Documentation • Styling • Automated tested • CI/CD • …	• Cyclomatic complexity • Code coverage • Low Coupling

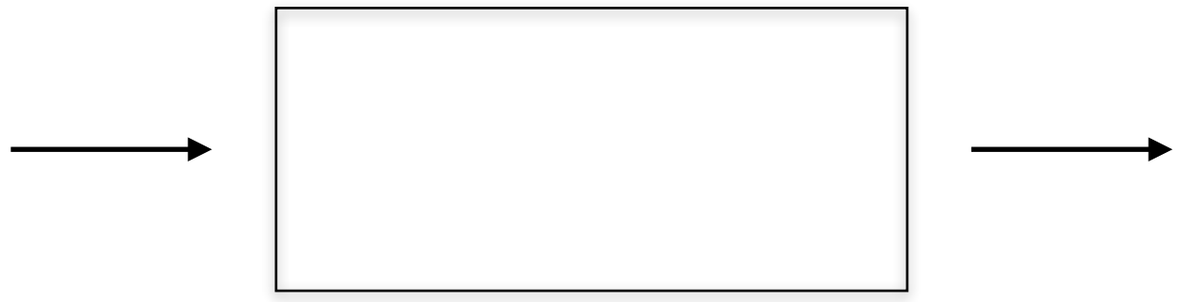
(What) Quality attribute ?	(How) Approach (tactic/style) ?	Measure ?
• Performance	• Caching • Compression • Parallel • Object pooling •	• Latency • Throughput • Response time •

(What) Quality attribute ?	(How) Approach (tactic/style) ?	Measure ?
• Correctness • Maintainability • Performance(cpu, memory, disk, network, …) • Scalability (volume of resource: cpu, memory, disk, network, …) • Availability • Security (trust) • Reliability (trust) • Robustness (rugud) • Usability • Interoprability • Portability	• Caching • Compression • Parallel • Object pooling • Reusability • Authorization • Input validation • Throttling (max) • ACID (transaction) • Testability	• Latency • Throughput • Response time • % of down time • % of uptime • tps • Probability • Mtf, mtbf, pf, … • No of clicks •

Quality Models(McCall model, the Boehm's model, the IEEE, SEI)

(What) Quality attribute ?

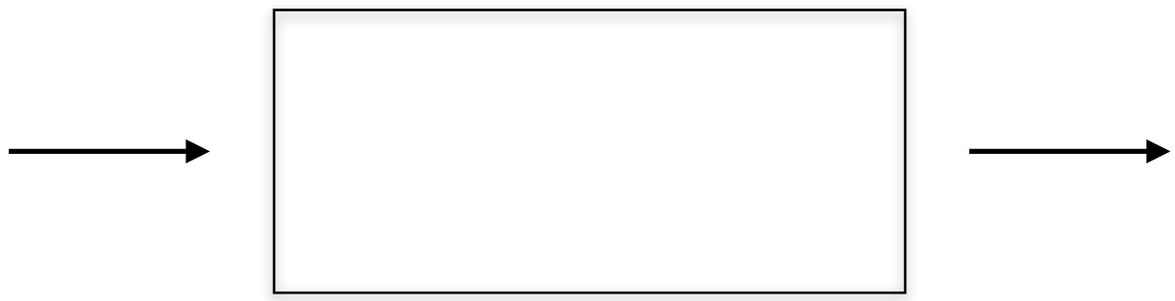
Measure ?



(How) Approach (tactic/style) ?

Knowledge

“Quality requirements”



Architecture

“Approaches”

Implicit Architecture	Explicit Architecture
Performance tuning **(after)**	**Performance engineering** **(before)**
Hacking **(after)**	**Threat Modeling** **(before)**

“functional requirements”

Understand

“Quality requirements”

Collect

Quality engineering

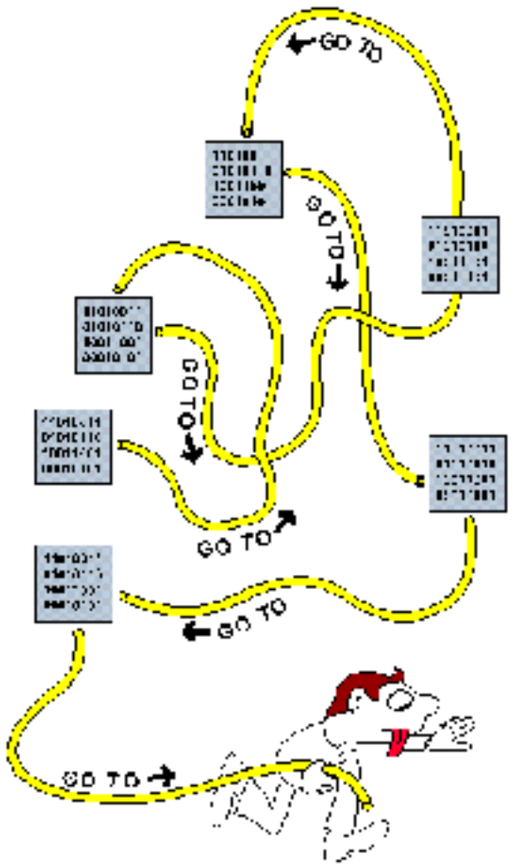
Decide

- Architecture
- High Level Design
- System Design
- Arch Design

“Approaches”

Knows

Styles, arch patterns, tactics,
Arch anti patterns.



“guidelines”

Follow

“fun requirement”

Understand

Manage
Code
Complexity

Create

- Detail Design
- Implementation Design
- Code design
- Low level design

“Skeleton for Code”

Knows

OO patterns, fun pattern,
lang idioms

Togaf, dodaf, zachman fwk

Enterprise Architect

(align)

Product/Solution Architect
(Quality of the product)

Application Architect

(Quality of the application)

Domain Architect

(Quality of the process)

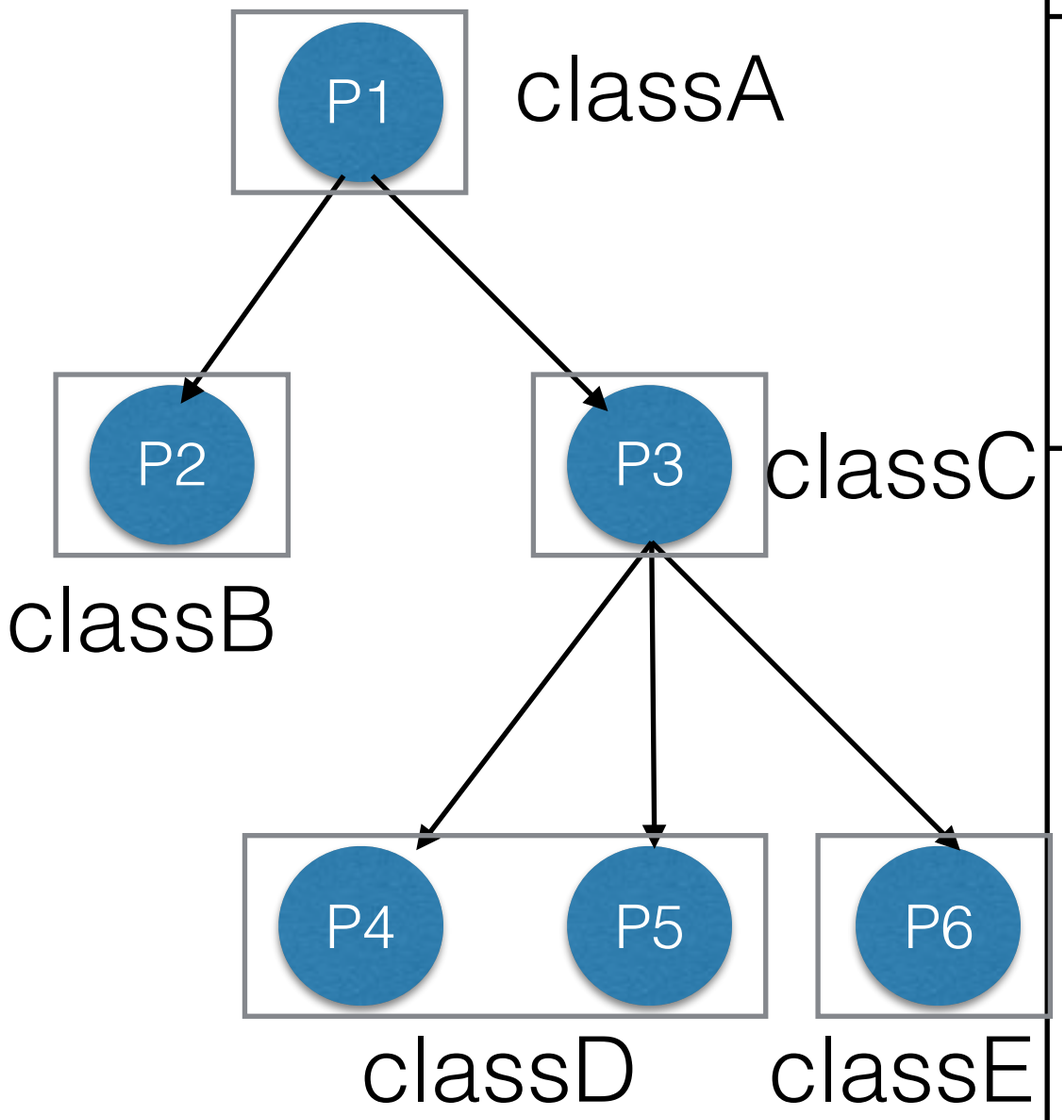
Vertical Architect

Security Architect
UX
Data
Infra
Cloud
Java
…

Proc vs OO vs fun

Procedural Prog

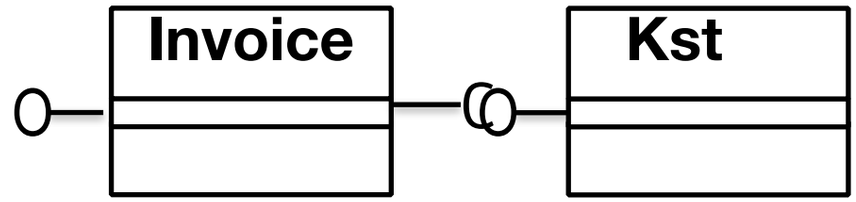
(tree)

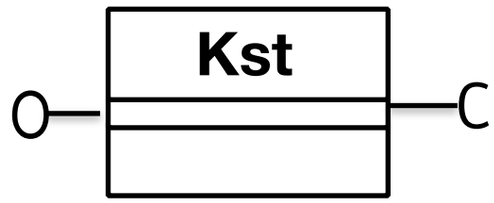


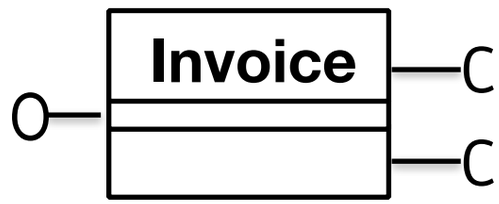
(top down)

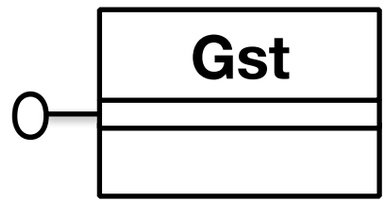
OO Prog

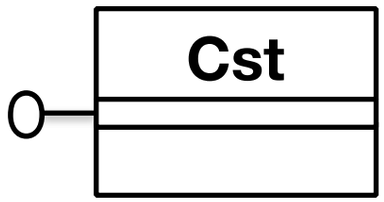
(Lego)





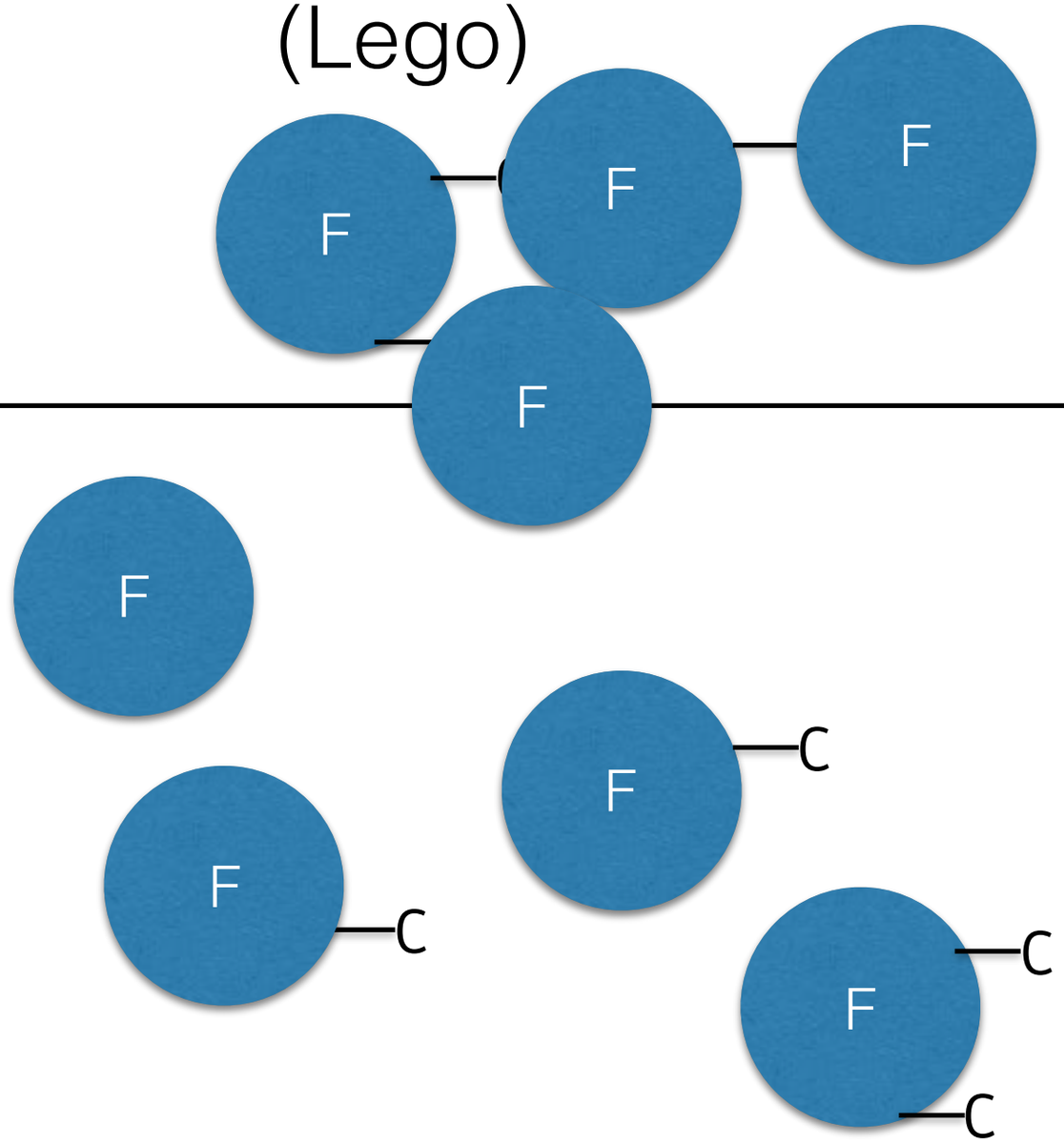




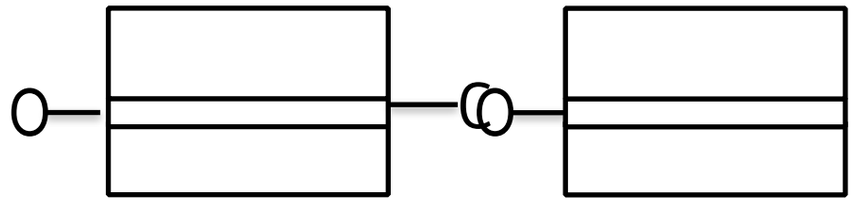


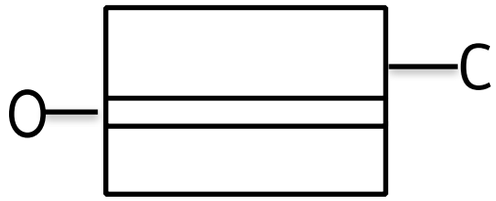
(bottom up)

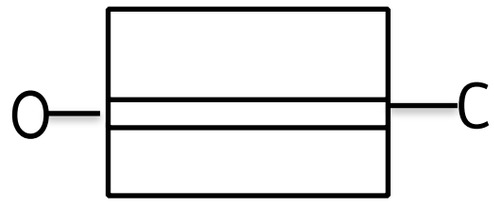
Functional Prog (Lego)

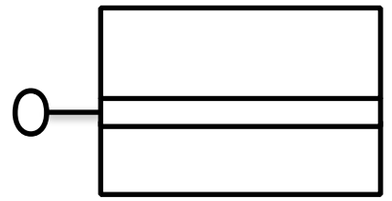


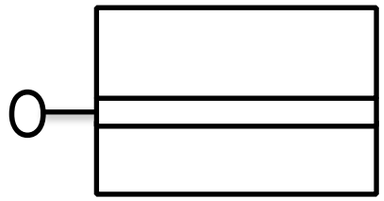
OO Prog (Lego)











	Proc (tree)	OO (lego)	Fun
Lang	C, py, java, C#, JS, c++	Java, C#, C++, py, js	py,js, J8,c#
Constructs	if/switch/goto/ Static methods	Polymorphism/ Exceptions	High order fun/ recursion/ closure
Performance	-	-	+ +
Security	-	-	-
Learning Curve	+ +	- -	-
Development Time	+ +	- -	+
Unit Test	- -	+	+ +
Code Maintainability/ Support Time	- - -	+ +	+

Todo

- 5 most important quality attributes for you domain
- At least 10 approach for each quality
- At least 3 measures for each quality
- Software Architecture in Practice -SEI practices
-

Anti patterns

- Alice in Wonderland

-

patterns

Case study

todo.com

- <todo.com>
- CRUD todo
- Web App
- Single user

GreatDeal.com

- <GreatDeal.com>
- Single product with n qty for a day
- Web App
- Collect payment if stock exist
- Send the product through delivery partner

bidder.com

- <bidder.com>
- Single product (1 qty) for a day
- Web App
- Collect payment from highest bidder
- Send the product through delivery partner

Telmon

- System Monitoring
- CPu utilization
- Memory
- Disk
- H/w failures
- Notify

App1 / Device

log
log

App2/ Device

log
log

Log agg

log
log
log
log

Analytics

Metrics

QAW process

- Quality attribute workshop
- Process to collect arch requirements
- Process to collect Quality Attribute scenario(NFR) | user story (FR)

1. Prepare seed Quality Attribute scenarios (NFR)
2. Get all stake holders in to a 1/2 day brainstorming session for NFR.
3. Collect Scenarios
4. Prioritise Scenarios

As a User I want to add a todo In the web portal when 100,000 users are using the portal. The portal displays a success message In < 3 sec time.

Source (who)	As a User
Stimulus (action)	I want to add a todo
Artifact (module)	In the web portal
Environment (context)	when 100,000 users are using the portal.
Response	The portal displays a success message
Measure	In < 3 sec time.

Source (who)	processor
Stimulus (action)	stops working
Artifact (module)	in the “central system”
Environment (context)	during peak traffic hours
Response (output)	start providing “degraded mode” service
Measure	The time spent in degraded mode should be no more than 5 minutes.

App

Piece1

Piece2

Piece3

Architecture

Collect Arch Requirements

1. Context view
2. Functional View
3. Quality View
4. Constraints

Build Arch

1. Logical View
2. Deployment View

Eval Arch

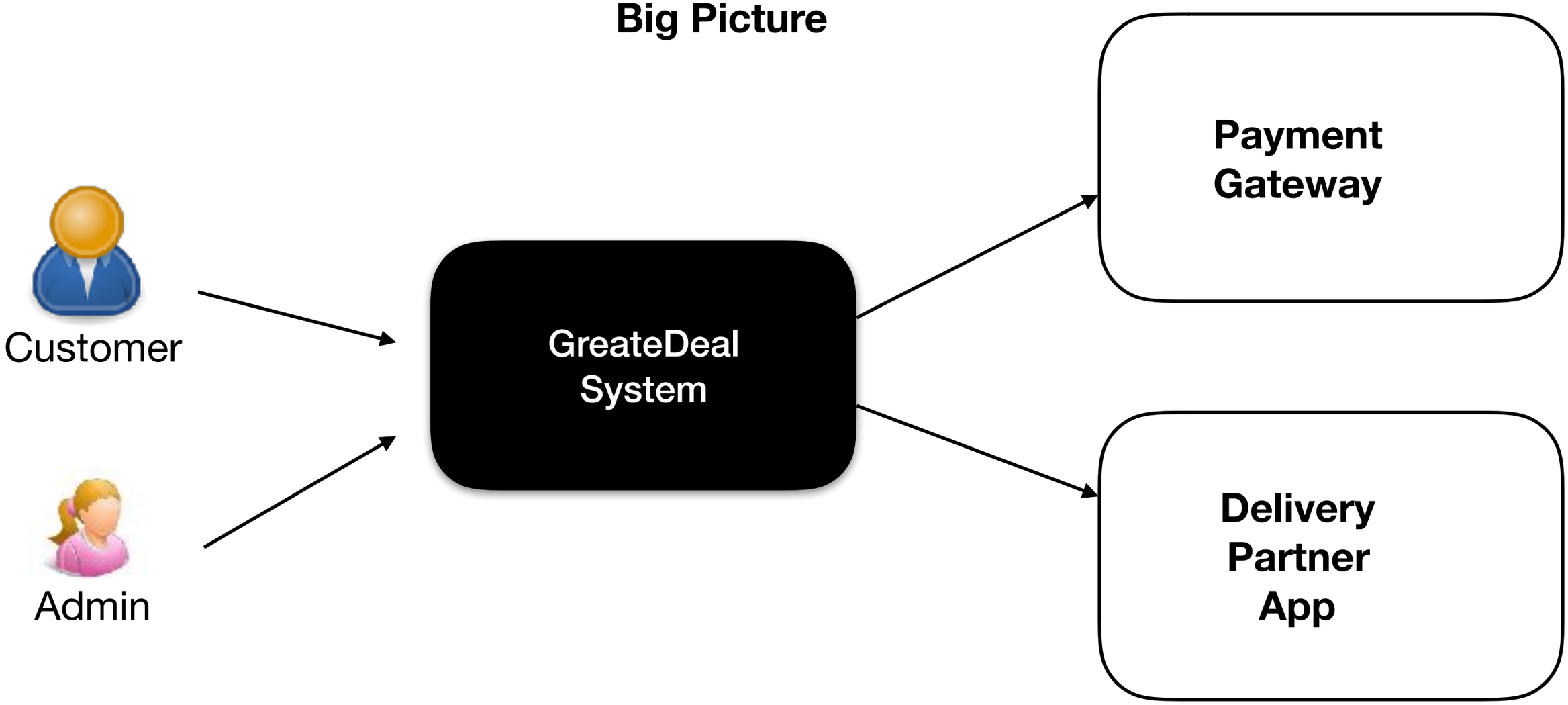
GreatDeal

Arch

Requirements Gathering

Context View

Black Box View
Big Picture



- Does it set the scene ?
- What is it that's being built?
- How does it fit into the surrounding environment ?
- Does it show relationship with the existing System ?

- 80:20 rule (20% is important)
- Does it Identifies key users ?
- Does it identify the architecturally significant use cases ?
  - **Business Critical**. The use case has a high usage level or is particularly important to users or other stakeholders when compared to other features, or it implies high risk.
  - **High Impact**. The use case intersects with both functionality and quality attributes, or represents a crosscutting concern that has an end-to-end impact across the layer and tiers of your application. An example might be a Create, Read, Update, Delete (CRUD) operation that is security-sensitive.
  - Include a summary to highlight why are they architecturally significant.

Quality View

- As a User I want to view the Deal of the Day when 100,000 users are using the portal. The portal displays the Item In < 1 sec time. (performance)

- When a user places an order, the payment fails in the server during peak hours and the order is cancelled and money is refunded within 2 hours. (reliability)

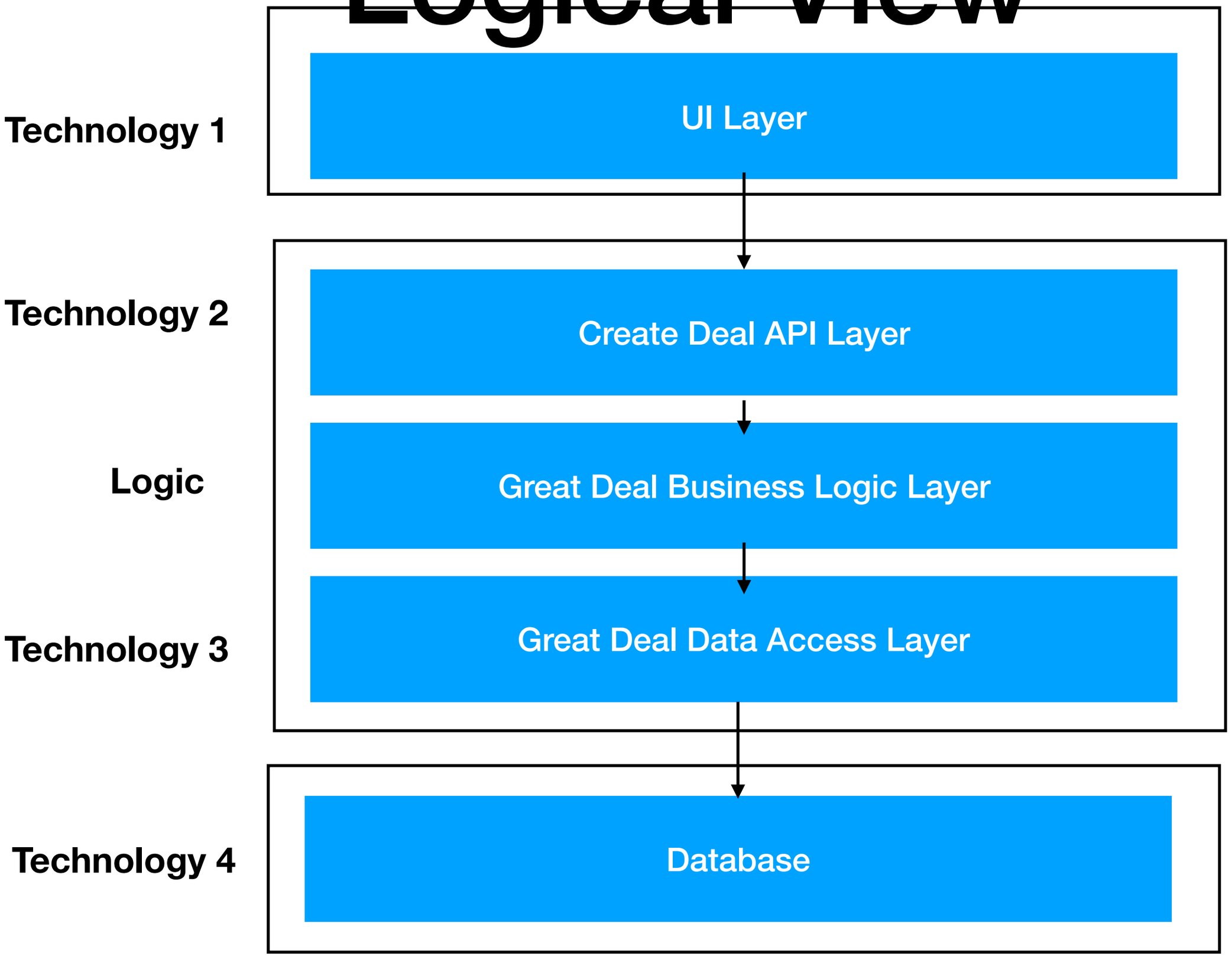
- When a user enters incorrect bidding value into the bidding Web App while product information is displayed. The system prints an error message for the respective user. User is able to bid again with correct value within 30 seconds. (robustness)

Constraints View

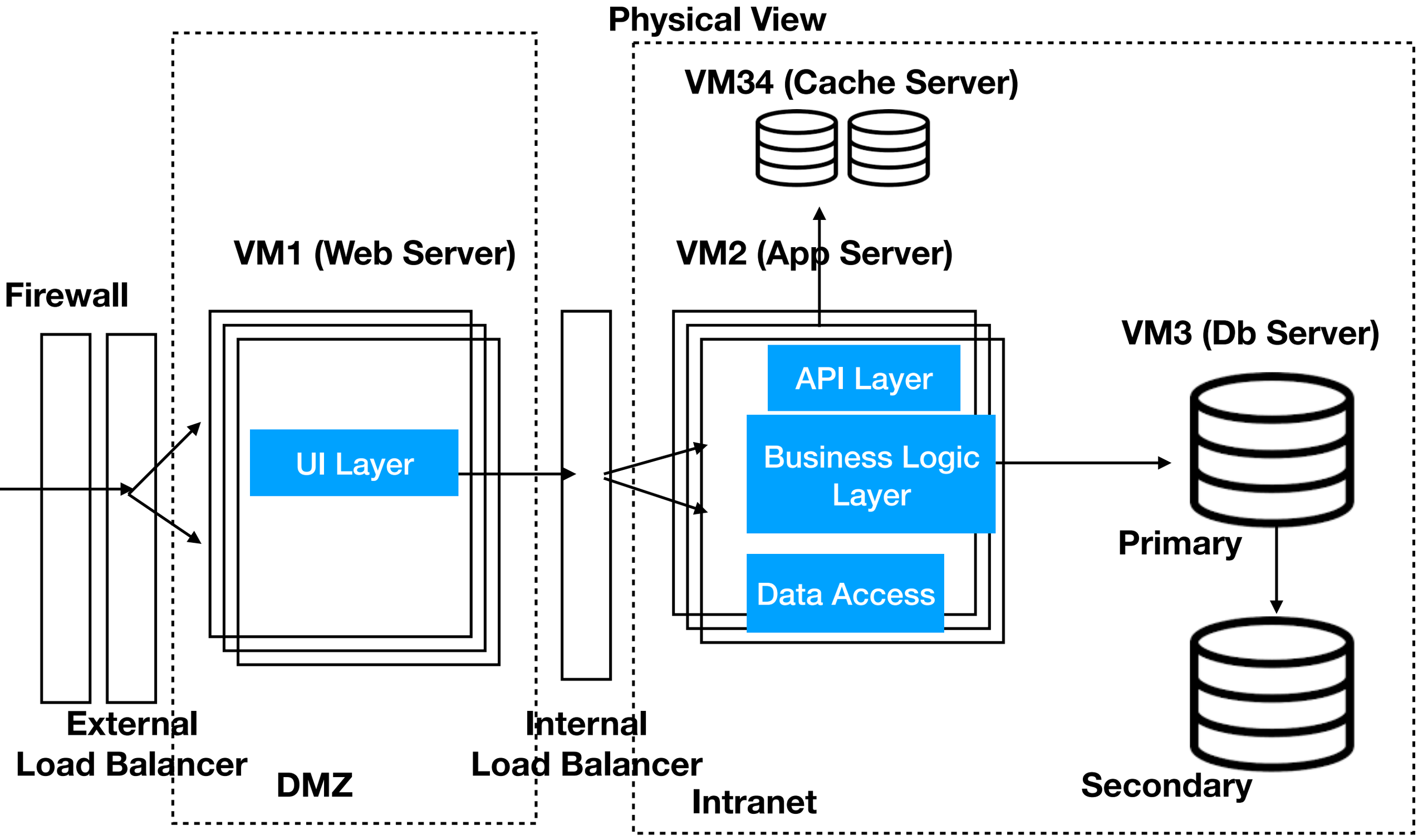
- Should support Internet Explorer 11
- Use open source stack
- API should be built using python

GreatDeal Architecture

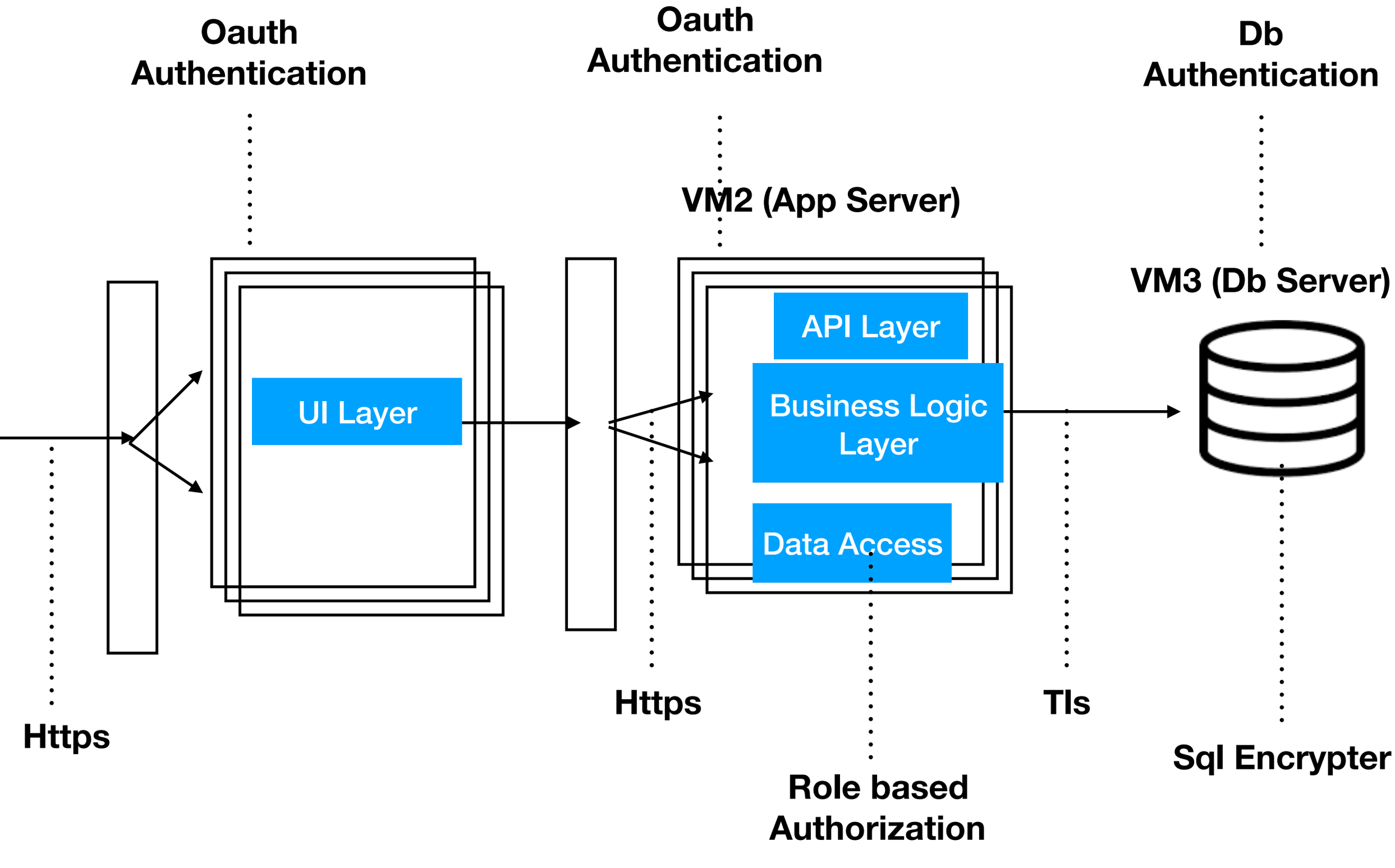
Logical View



Deployment View



Security View



Data View

 # transaction - reliability
 # scalability
- **volume (split, shard, clone)**
- **request (split, clone)**
- **aggregation (materiallized view)**

 # Data Security
 # Data Retention (archive)
 # Data Recovery
 # Data quality (integrity)
 # Data authorisation

Operation View

 # Infrastructure

- **Monitoring (CPU, Memory, I/O, Disk, LB, ..) <— Nagios**
- **Alerting**

 # Application Monitoring

- **Application logs (ELK, Splunk, EFK, data dog, new relic)**
- **API Monitoring (APi Gateway)**
- **Alert**

Evaluate Architecture (ATAM)

 # identify all Architectural approaches

 # A1 (CQRS)
 # A2(Cube)
 # A3 (Caching)
…

 # identify all quality requirements

 # s1 (< 5 sec)
 # s2(99.99%)
 # s3 (…)
 # …

 # analyse Scenario -> Approach

S1 -> A1, A2
S2 -> A6,A8, A9
S3-> ?
S4 -> A6, ?

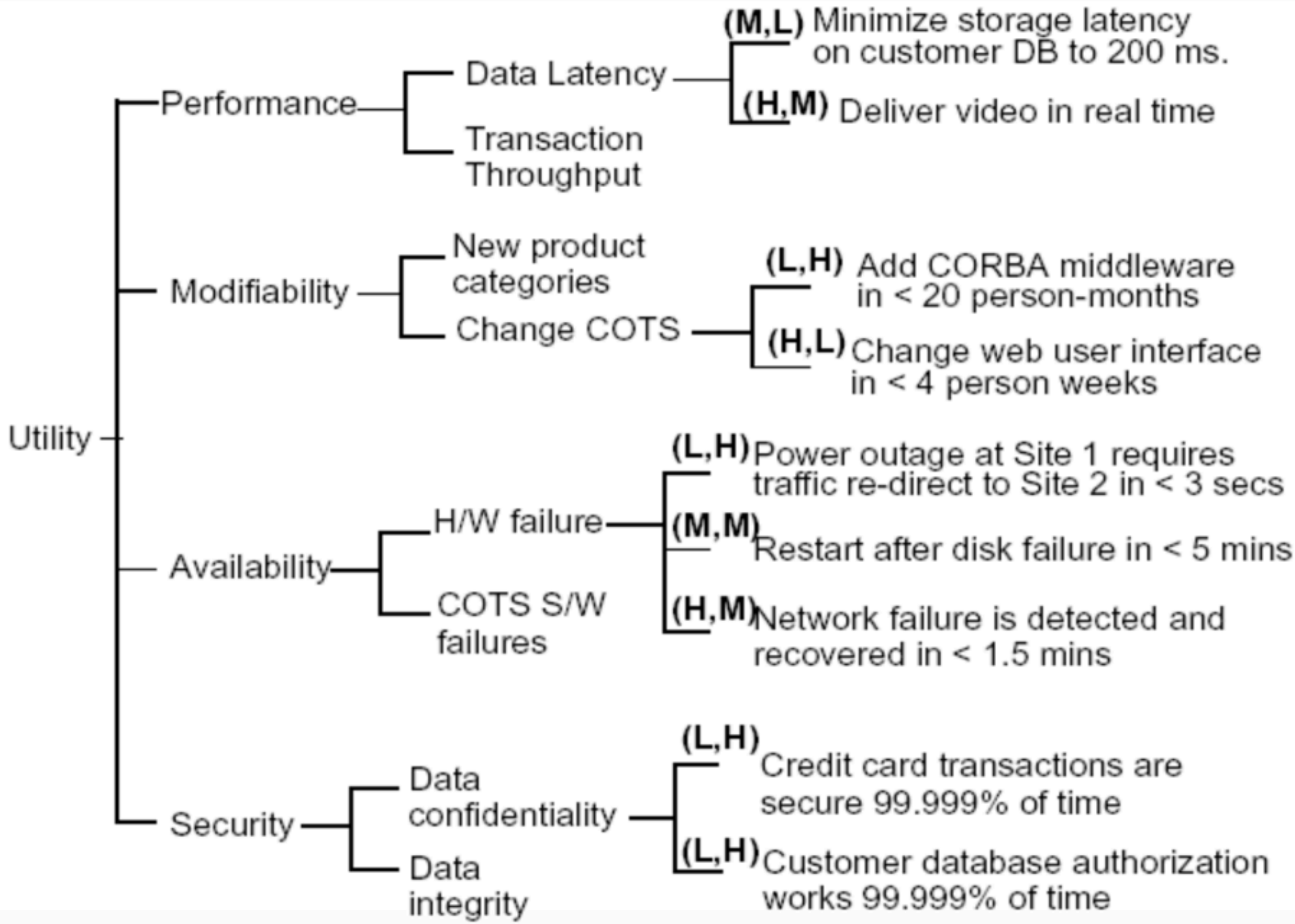
=> Risk & trade off's

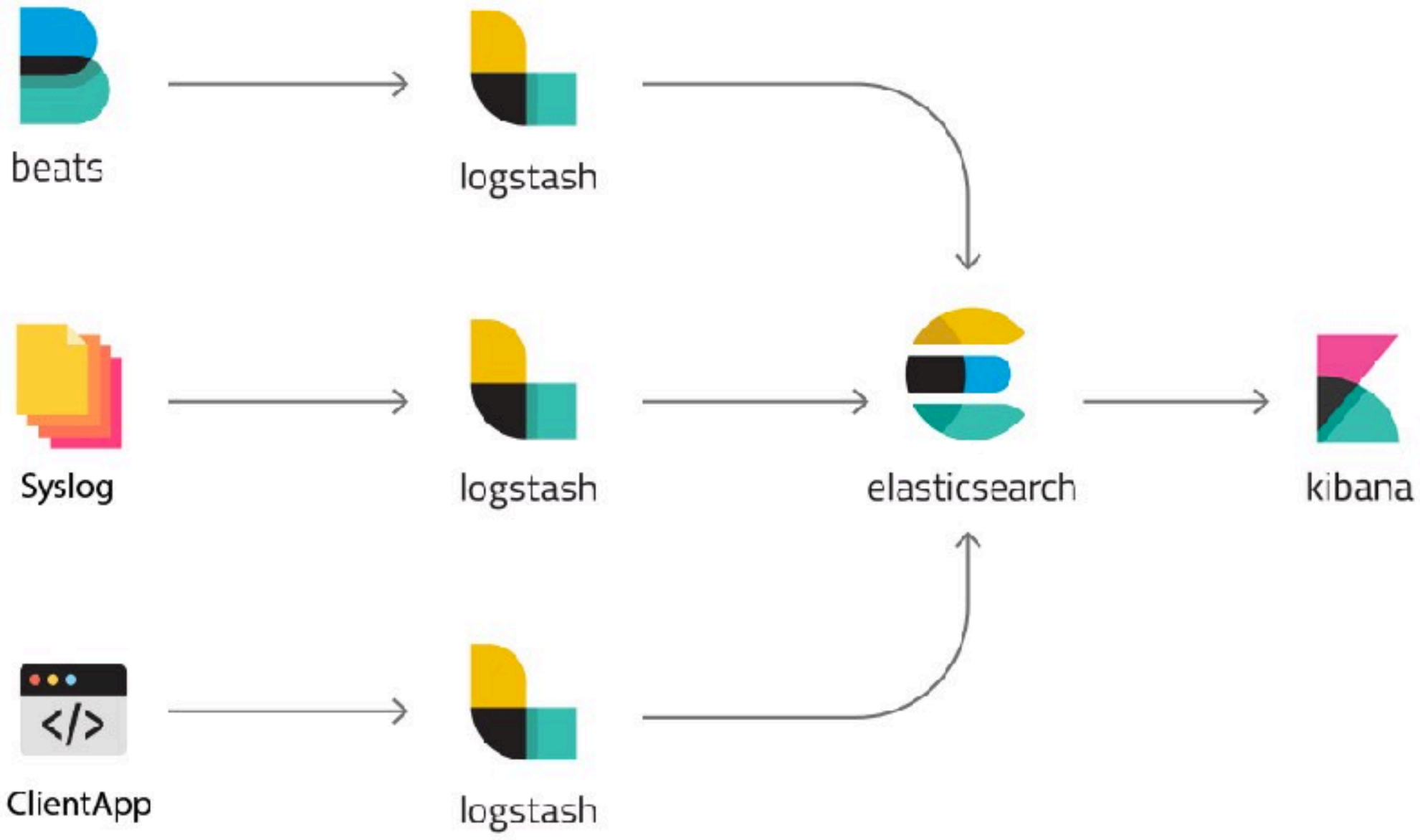
 # brainstorm for scenarios

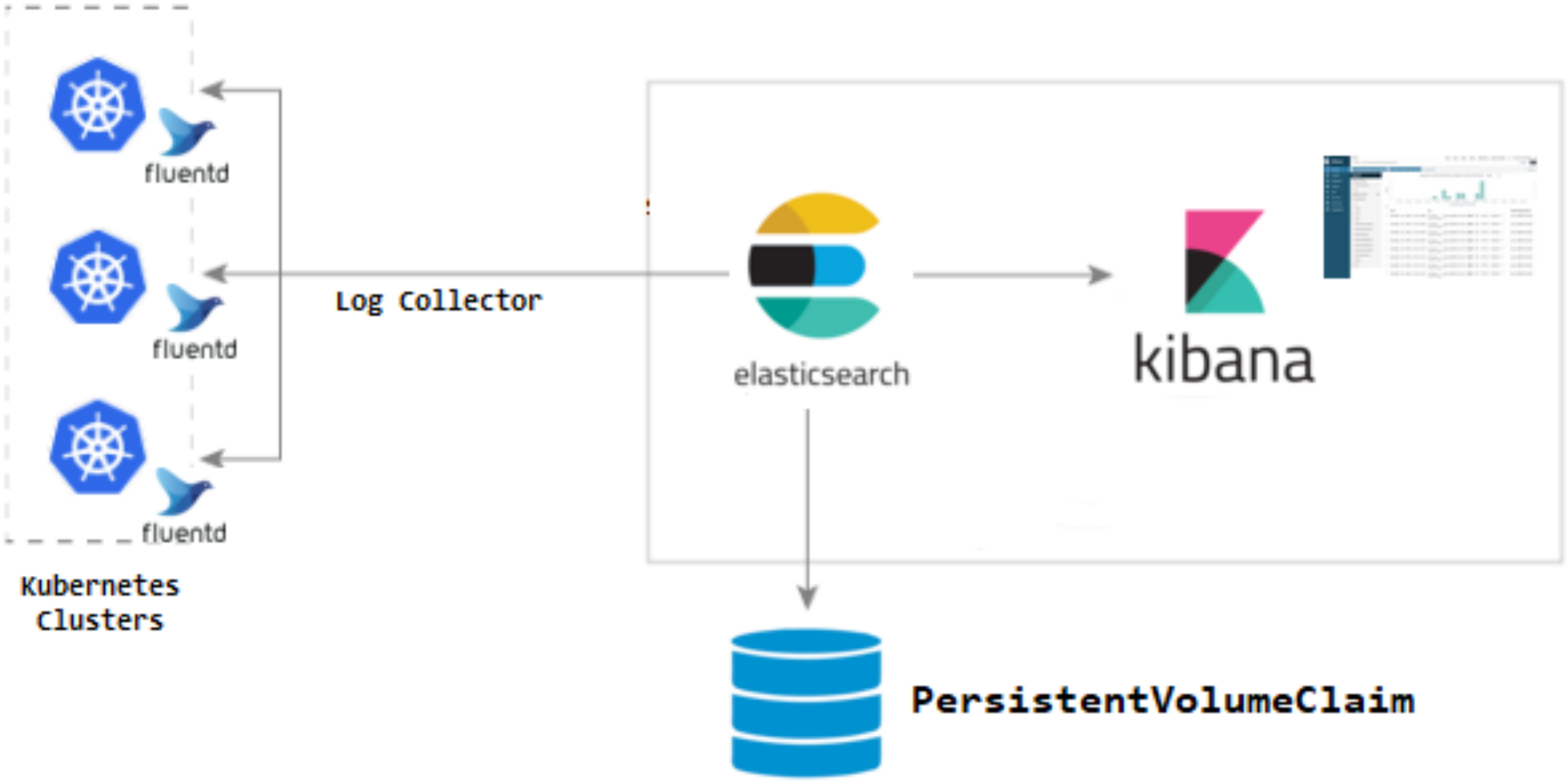
 # s1 (< 5 sec)
 # s2(99.99%)
 # s3 (…)
 # …

 # analyse Scenario -> Approach

=> Risk & trade off's

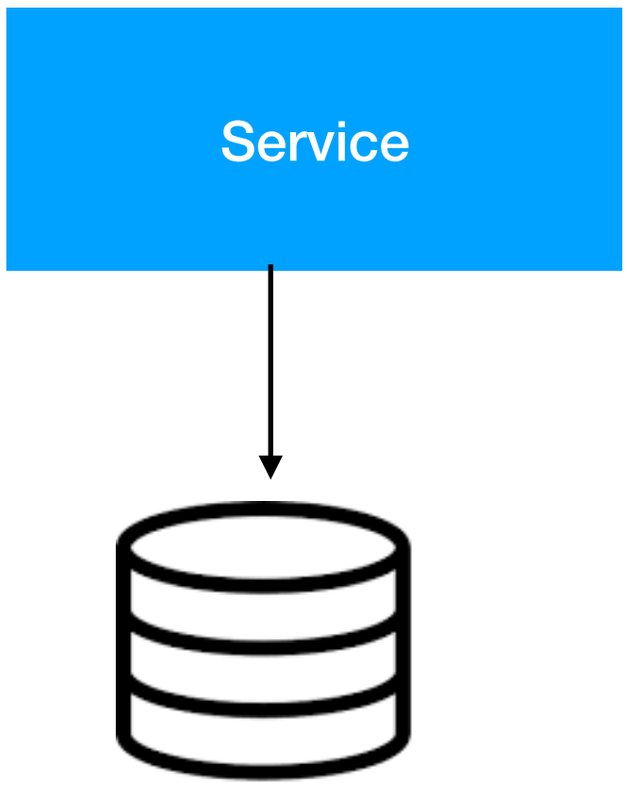

Utility
Performance
Data Latency
(M,L) Minimize storage latency on customer DB to 200 ms.
Transaction Throughput
(H,M) Deliver video in real time
Modifiability
New product categories
Change COTS
(L,H) Add CORBA middleware in < 20 person-months
(H,L) Change web user interface in < 4 person weeks
Availability
H/W failure
(L,H) Power outage at Site 1 requires traffic re-direct to Site 2 in < 3 secs
(M,M) Restart after disk failure in < 5 mins
COTS S/W failures
(H,M) Network failure is detected and recovered in < 1.5 mins
Security
Data confidentiality
(L,H) Credit card transactions are secure 99.999% of time
Data integrity
(L,H) Customer database authorization works 99.999% of time


beats
logstash
Syslog
logstash
elasticsearch
kibana
ClientApp
logstash


fluentd
fluentd
fluentd
Kubernetes
Clusters
Log Collector
elasticsearch
kibana
PersistentVolumeClaim

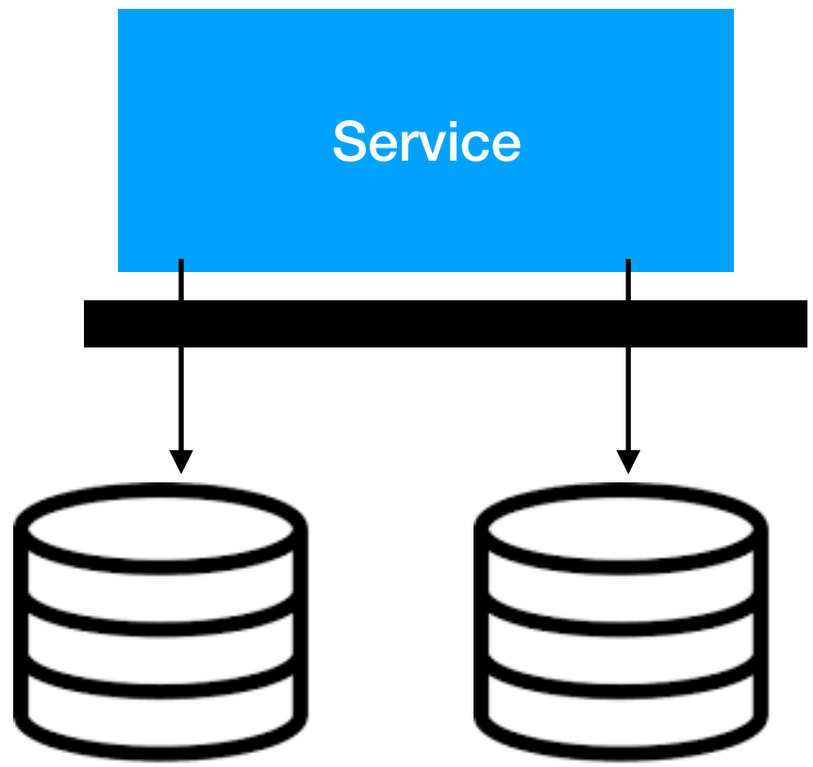
Transaction

Db transition



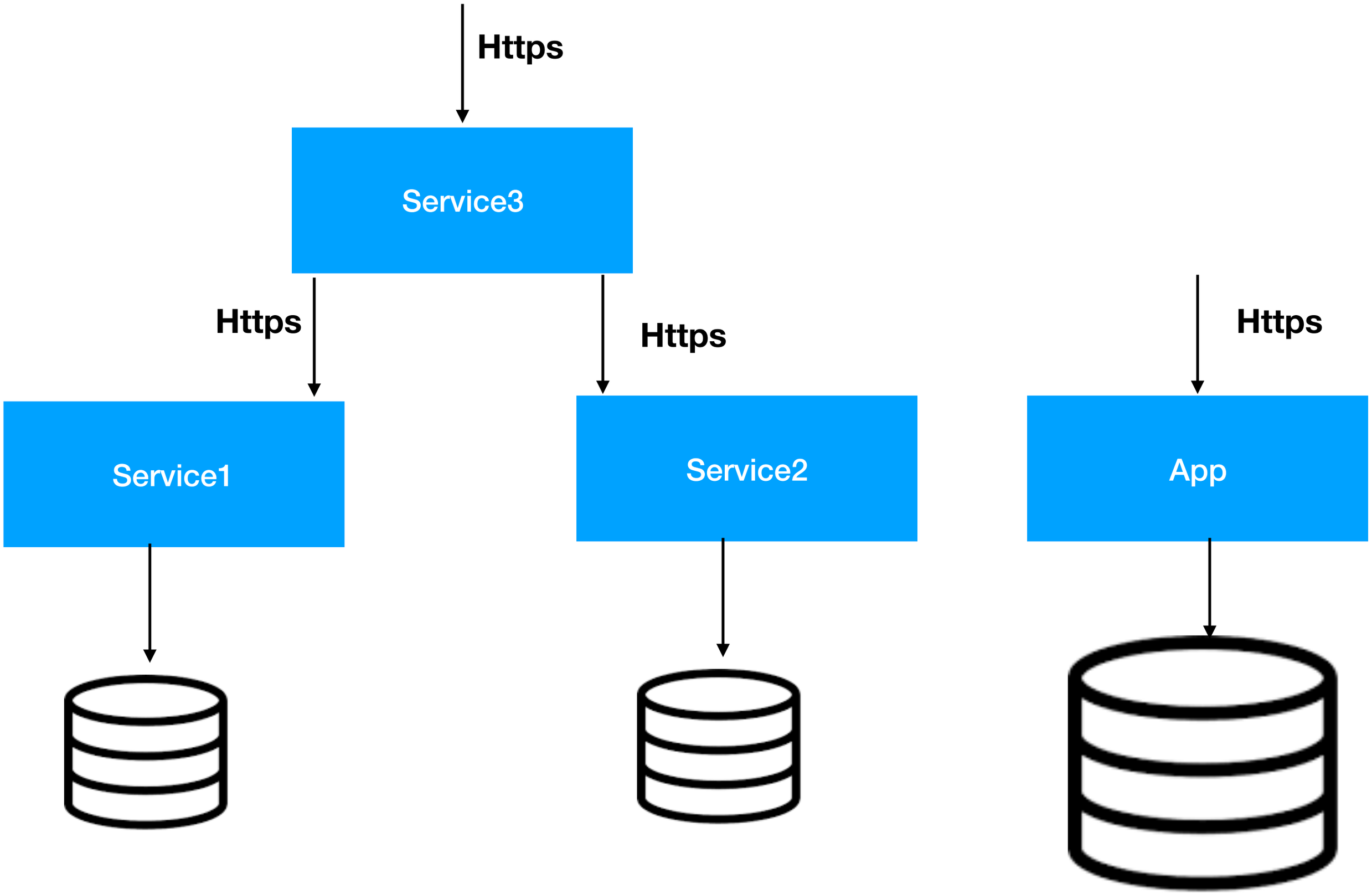
Begin
Do
Do
Do
Commit

2 phase commit
(JTX, MSDTC, …)



custom logic
(book keeping)

SAGA


Https
Service3
Https
Https
Https
Service1
Service2
App

Service1

Service1

Service1

Security View

- Authentication (first defence, who am i)
- Authorization (what can I do/see)
- Audit (last defence, what did I do)
- Input validation
- Session mgmt
- Exception mgmt
- Asset mgmt (rest, transit)
- Configuration mgmt

- AAA
- STRIDE
  - Spoofing
  - Tampering
  - Repudation
  - Info disclosure
  - Denial of service
  - Elevation of prove
- Authentication
  - By what you know (pwd,key,scret)
  - By what you have (otp,rsa,email,..)
  - By what you are (face,retina,voice,
-

Security View

List all entry points (webserver, app server, db server)
List all exit points
Identify Assets
Identify all data flow

Architectural Style

- Layered style
- Tiered Style
- Pipes and filter style
- Micro Service (Micro Apps)
- EDA (Event Driven Architecture)
- Hexagonal
- CQRS

GreateDeal
System

<<apply architectural style>>

Part1

Part1

Part1

<<apply architectural patterns >>

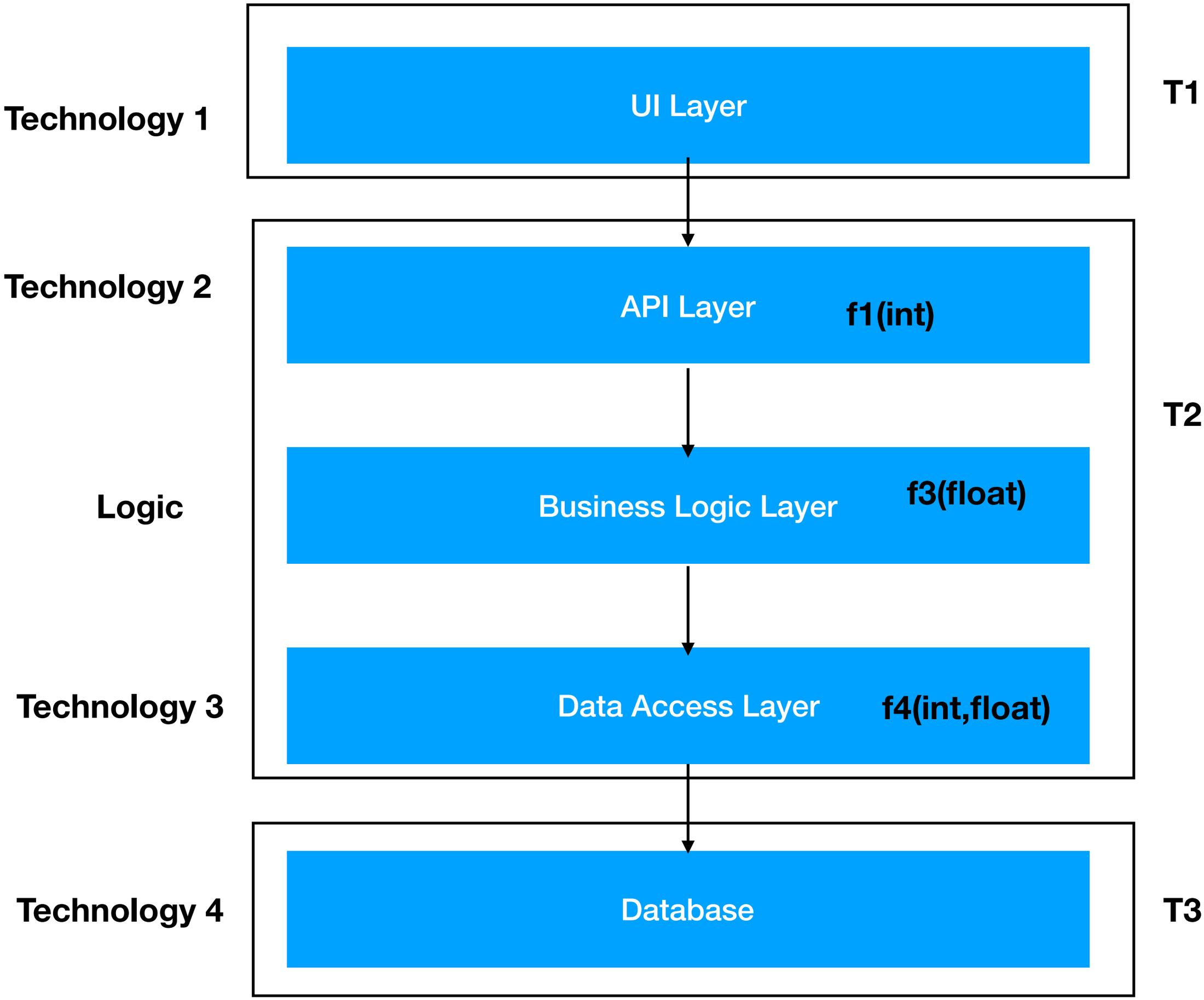
Part1.1

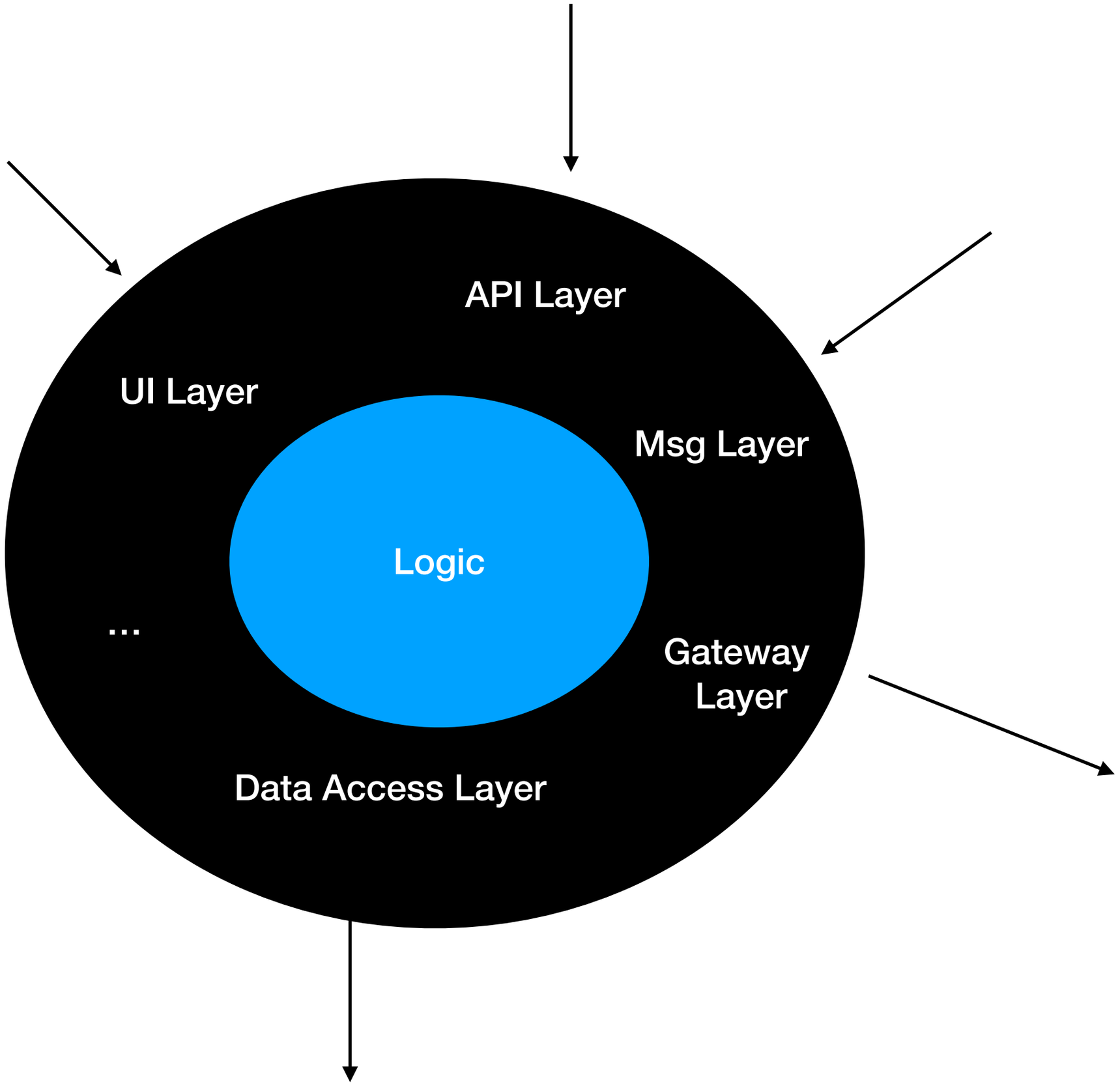
Part1.2

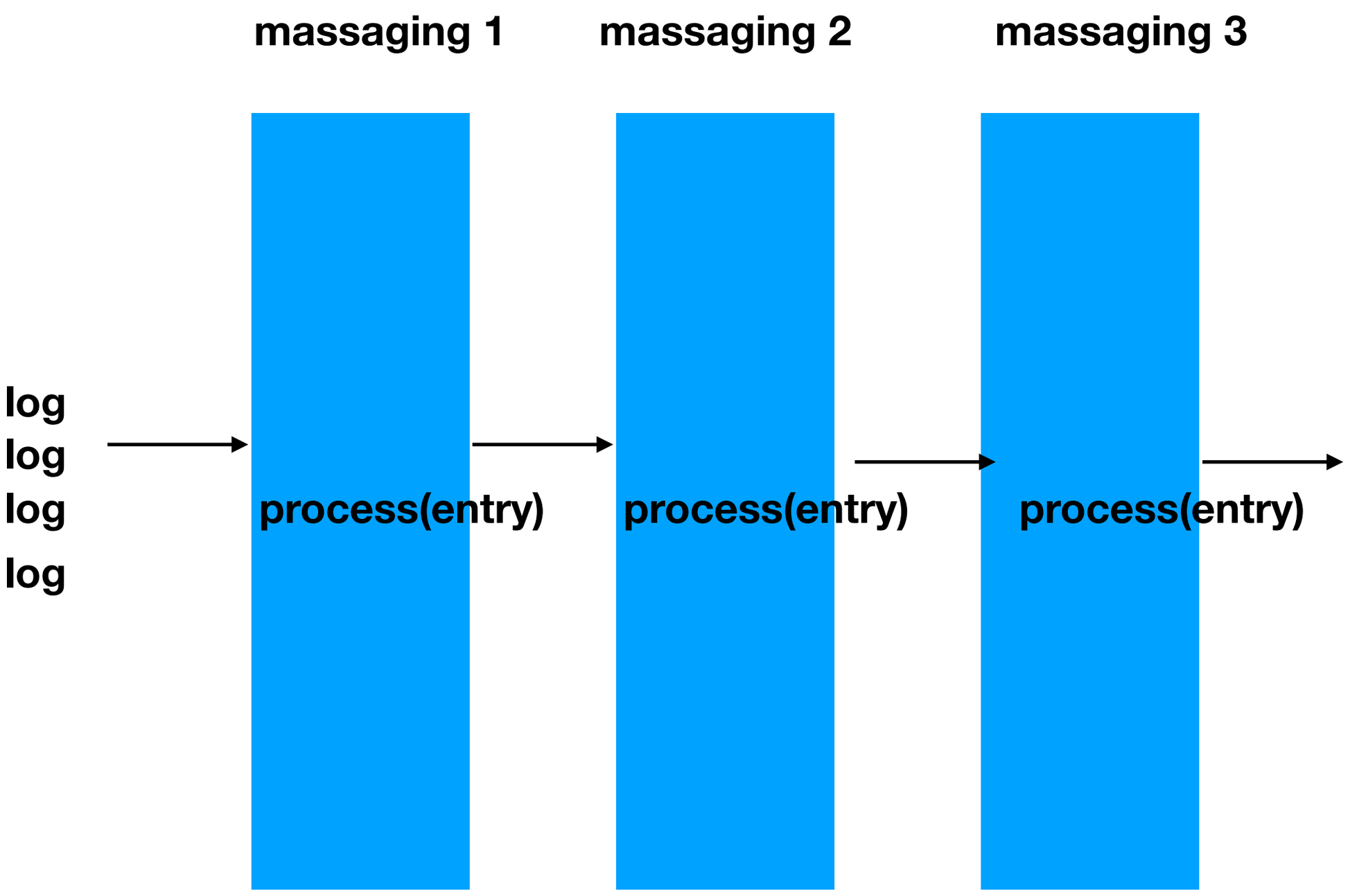
<<apply architectural tactics >>

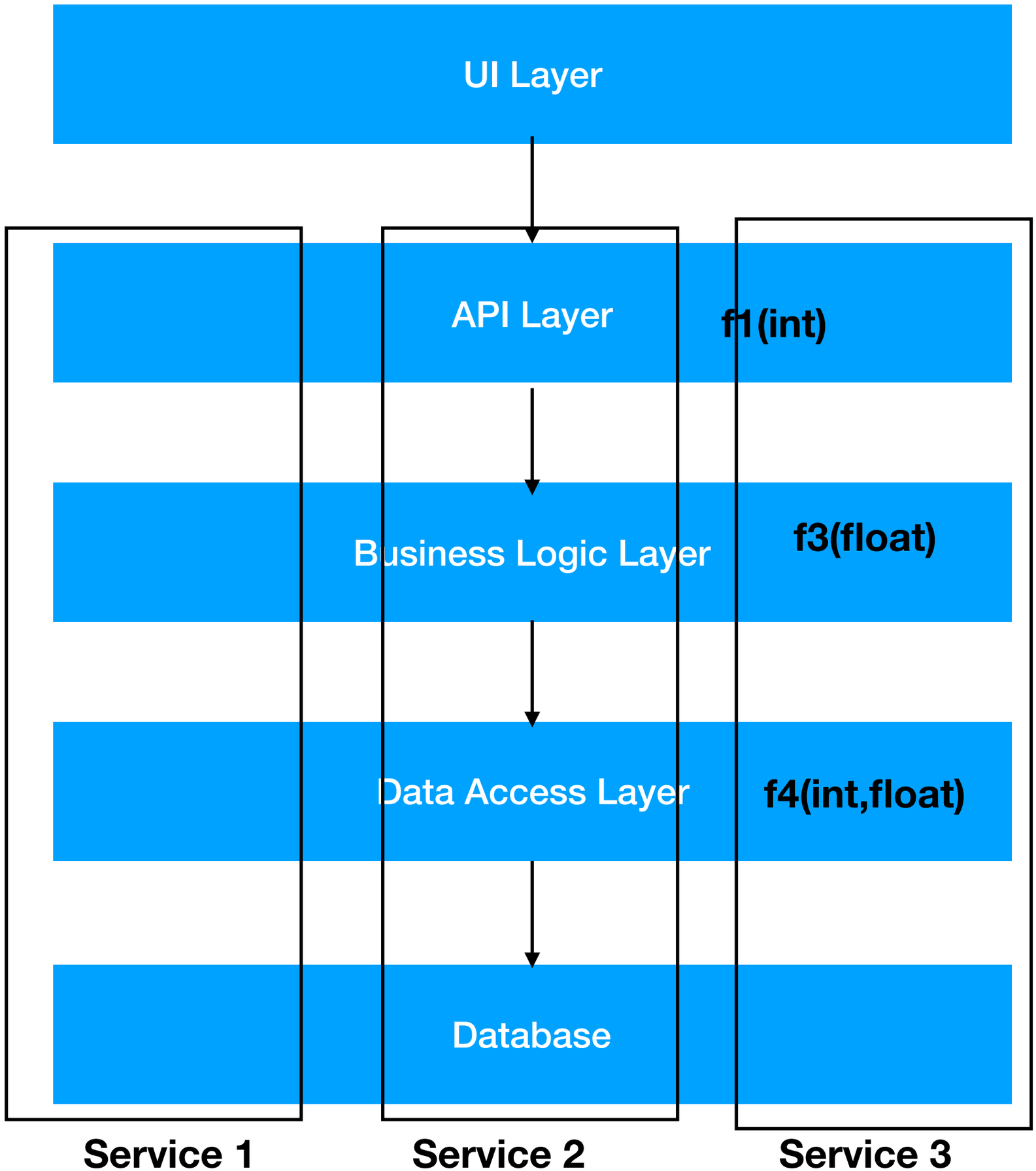
Part1.2.1

Part1.2.2

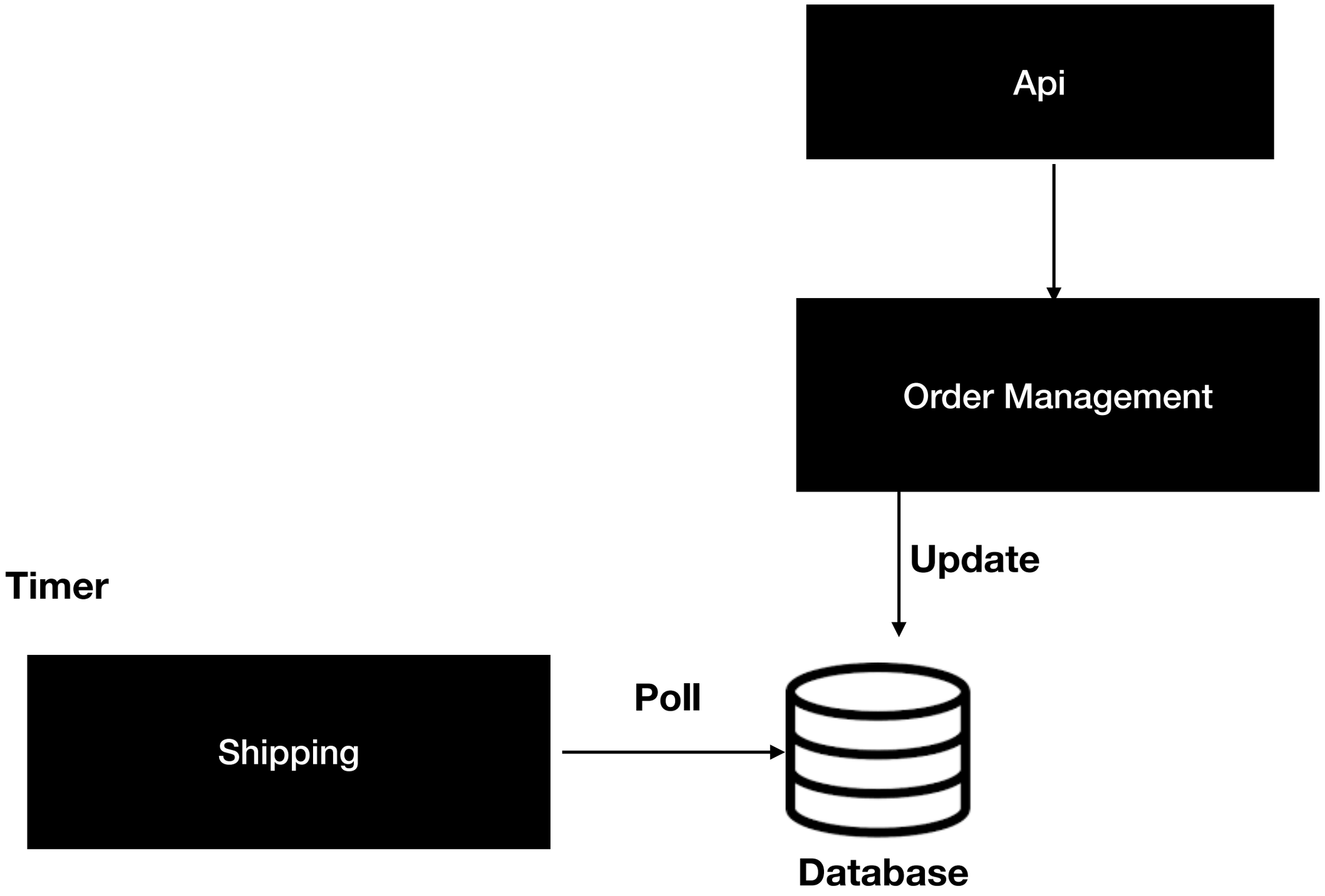

Technology 1
UI Layer
T1
Technology 2
API Layer
f1(int)
T2
Logic
Business Logic Layer
f3(float)
Technology 3
Data Access Layer
f4(int,float)
Technology 4
Database
T3


API Layer
UI Layer
Msg Layer
Logic
...
Gateway
Layer
Data Access Layer


massaging 1
massaging 2
massaging 3
log
log
log
log
process(entry)
process(entry)
process(entry)


UI Layer
API Layer
f1(int)
Business Logic Layer
f3(float)
Data Access Layer
f4(int,float)
Database
Service 1
Service 2
Service 3

Trigger for a logic
Timer -> Time (wish for birthday, backup, marketing mails)
Timer + poll -> Domain Event (create order, cancel order, …)
Human -> UI Click



“Push Design”

Api

Create Order cmd

Activemq
Msmq
MQSeries
Rabbitmq
Kafka

Message Bus

Created order event
Created order event
Cancel order event

Created order event
Cancel order event

Create order cmd

Shipping
> Idempotent
> unordered

Order Management

Update

Database

duplicate msg
unordered msg
One way (sucess/failure, return result)
Eventual Consistency
transaction

SAGA pattern

Api

Create Order cmd

Message Bus

NO Stock Event

Create Order cmd

NO Stock Event

Create Order cmd

Inventory Management

Update

Database

Order Management
undo logic
Idempotency
Unordered
send result

Update

Database

UI Layer

CUD

API Layer

Business Logic Layer

Data Access Layer

3 rd normal form

R

API Layer

Reporting Logic Layer

Data Access Layer

De normalized form

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

Bookid	Genere Type	Price

 # 3 rd normal form
 # no duplicates
 # write friendly
 # more joins
 # not read friendly

 # De normalized form
 # duplicate data
 # not write friendly
 # no joins
 # read friendly

Api GATEWAY

Api

“Push Design”

Create Order cmd

Message Bus

Created order event

Created order event

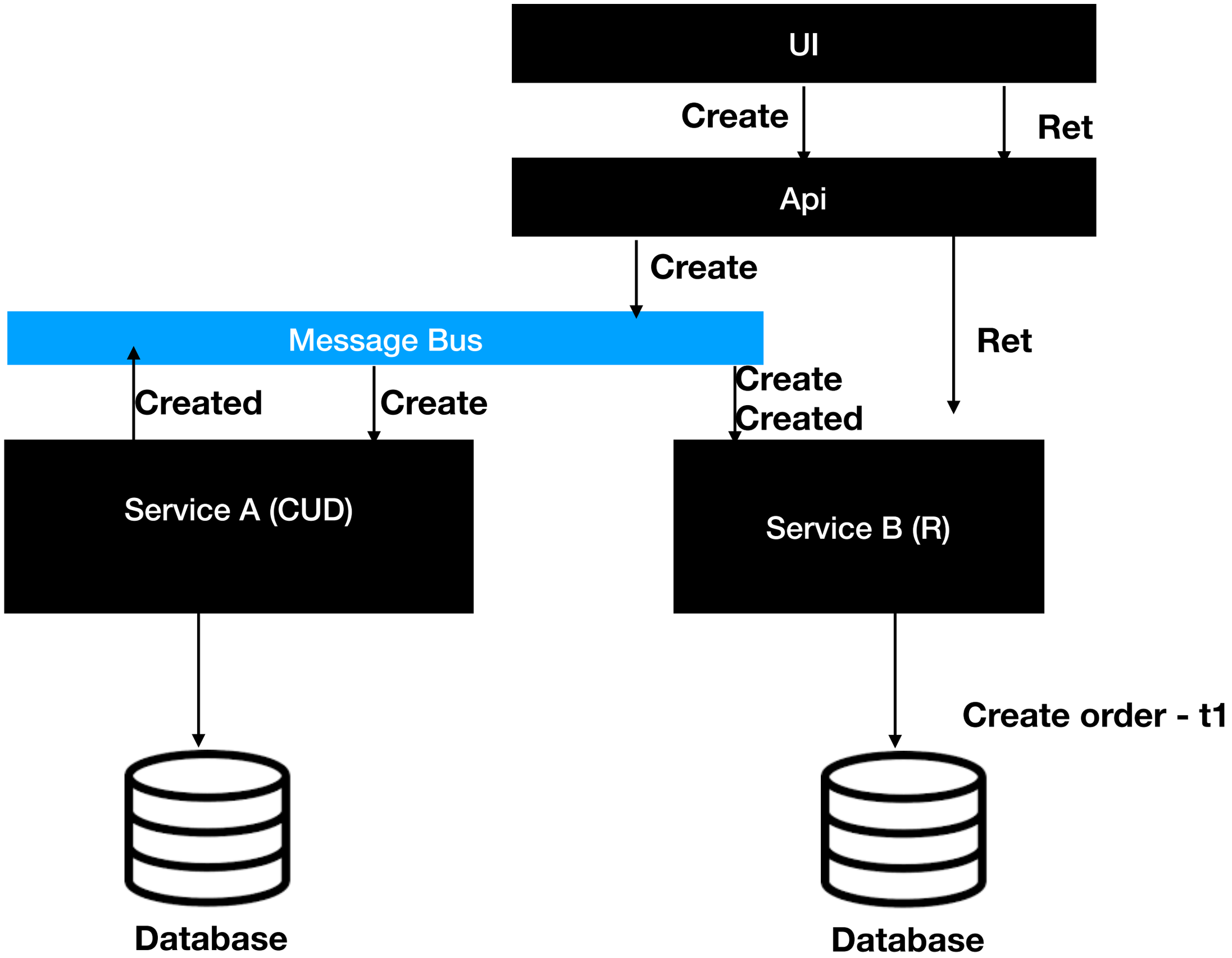
Create order cmd

Shipping

Order Management

Update

Database


UI
Create
Ret
Api
Create
Ret
Message Bus
Created
Create
Create
Created
Service A (CUD)
Service B (R)
Create order - t1
Database
Database

Scalability Cube pattern
50 rules for high scalability

Split

Vertical slicing

User

Inv

Acc

SC1#
Data1#

SC2#
Data2#

SC3#
Data3#

ecom

ecom

ecom

Write

Read

Read

SC#
Data#

SC#
Data#

SC#
Data#

Clone

Shard

Horizontal slicing

Male
Inv

Female
Inv

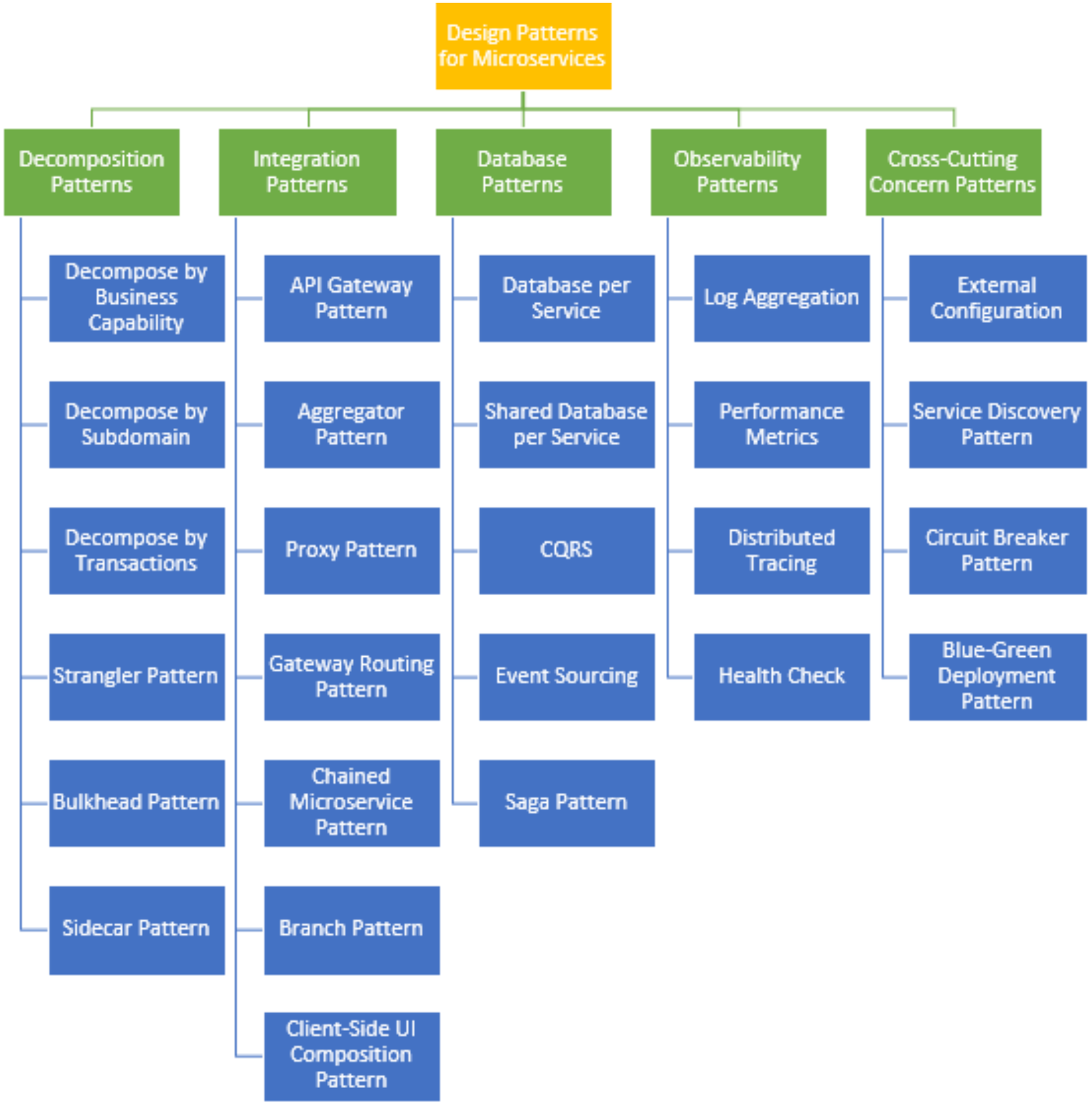
SC#
Data1#

SC#
Data2#

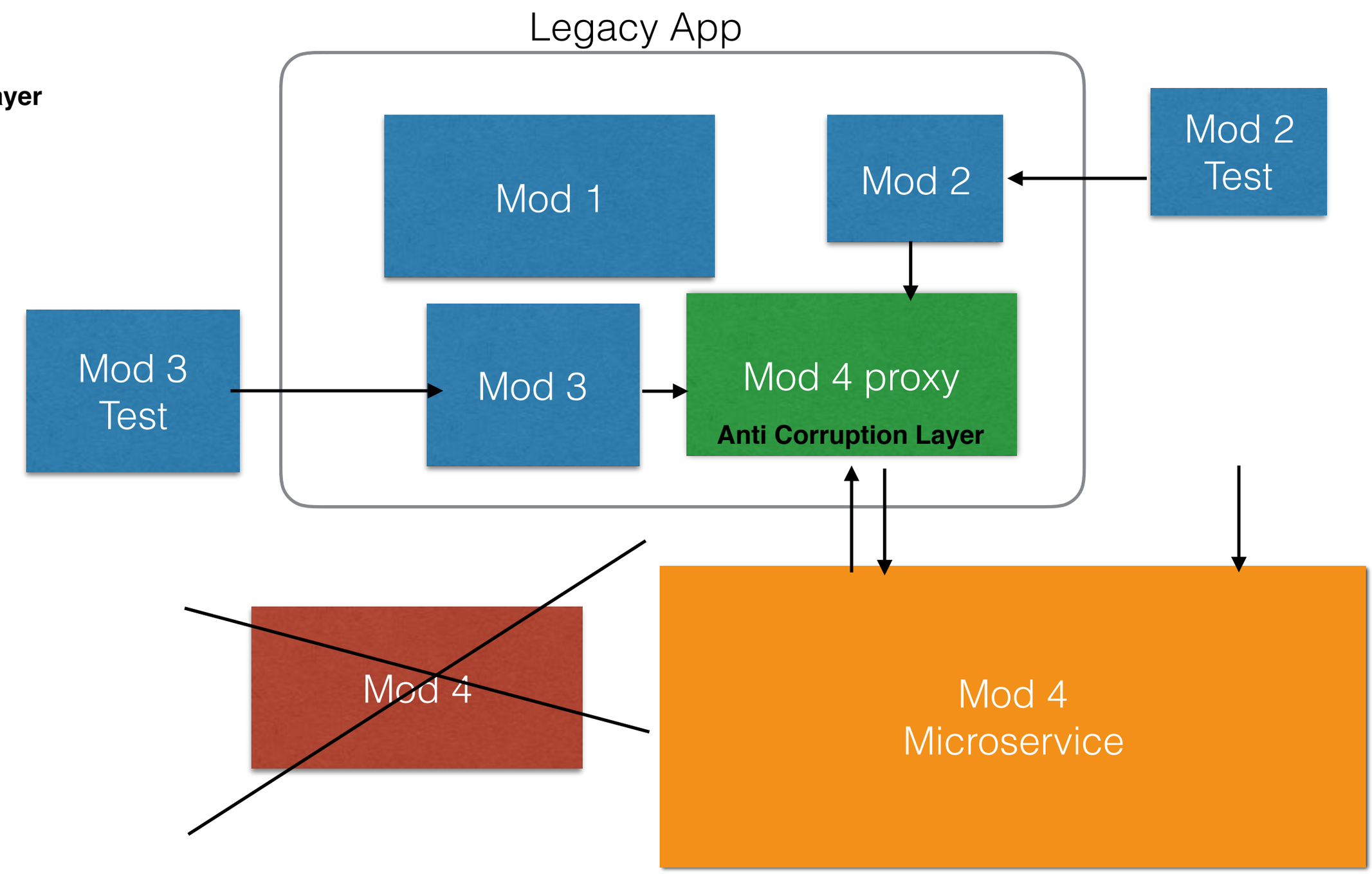
	Monolithic	**Micro**
Fun Requirements	Shared	Not Shared
Source Control	Shared	Not Shared
Build Server	Shared	Not Shared
Database	Shared	Not Shared
Deployment Infra	Shared	Not Shared
Architecture / Technology	Shared	Not Shared
Test Cases	Shared	Not Shared
SCRUM (team) /Sprint	Shared	Not Shared
Platform / Frameworks	Shared	Not Shared

Microservice

	Pros/ Cons
Performance	- - -
ACID (Transaction)	- - -
Time To Develop	- - -
Learning Curve	- - -
End to End Testing	- - -
Infrastructure cost	- - -
Devops	
Debug	- - -
Monitoring	- - -
Reproducable Environment	- - -
Configuration mgmt	
Log mgmt	
Resilancy (Bulk Head)	+++
Maintenability	+++
Scalability	+++
Polygot	+++
Agile Architecture	+++
Feature Shipping	+++


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

 # unit test
 # Strangler Pattern
 # Anti Corruption Layer



- A + b (3 cpu cycles)
- Fun (10 cpu cycles)
- Create Thread (100,000 cpu cycles)
- Remote db call (45,00,000 cpu cycles)

DHL

Requirement for a new App

App

Expose gems as Service

S1 S2 S3 S4

Identify All Gems in the Application

Dll Jar Py Bean

Identify All Applications in the enterprise

Cam Inv Hr

Utility tree

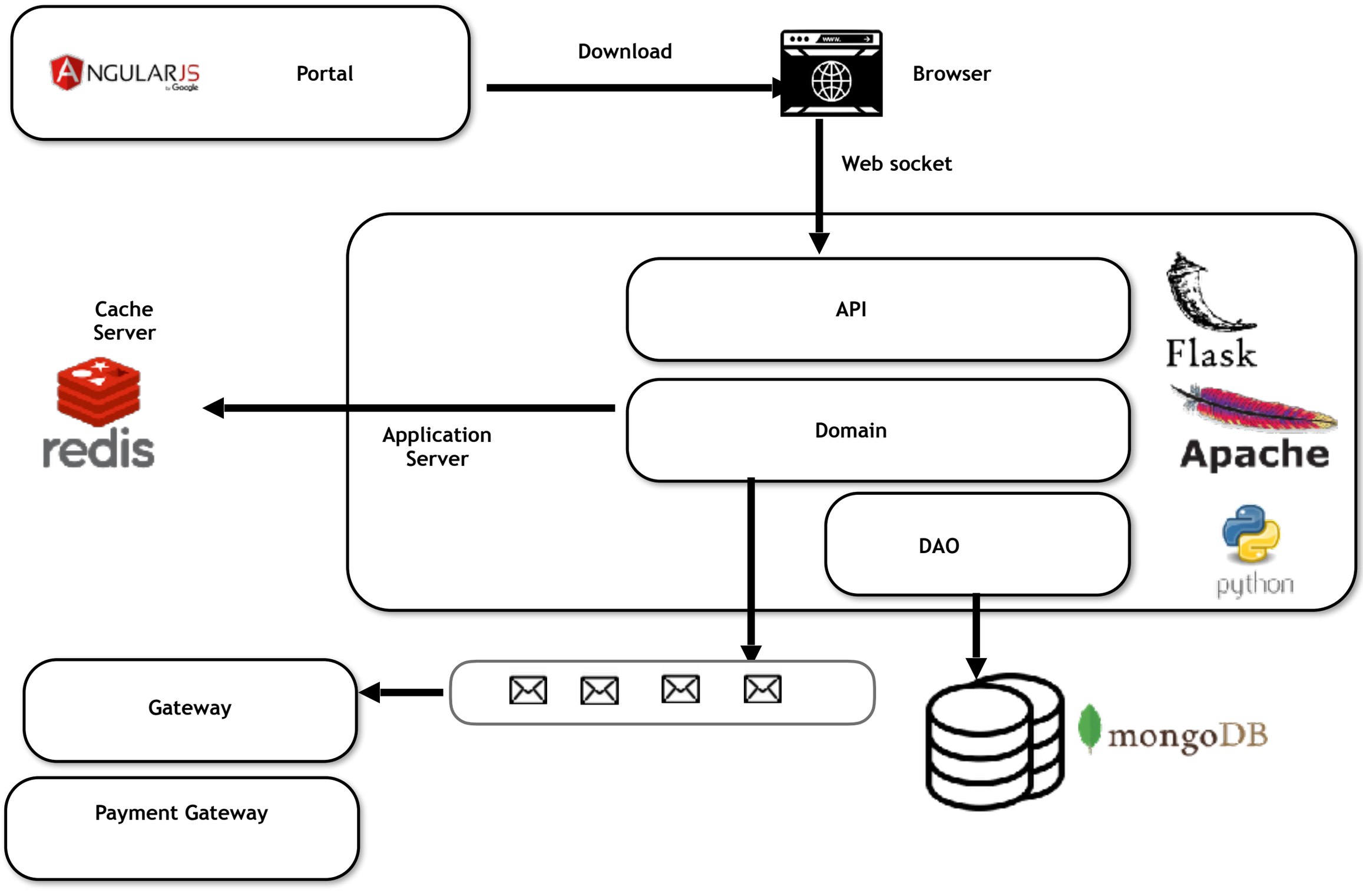
reliability

As a user i want to view the current on the portal during peak load. The updated bids are displayed in < 1 sec. (view bid : performance)

Architectural Definition

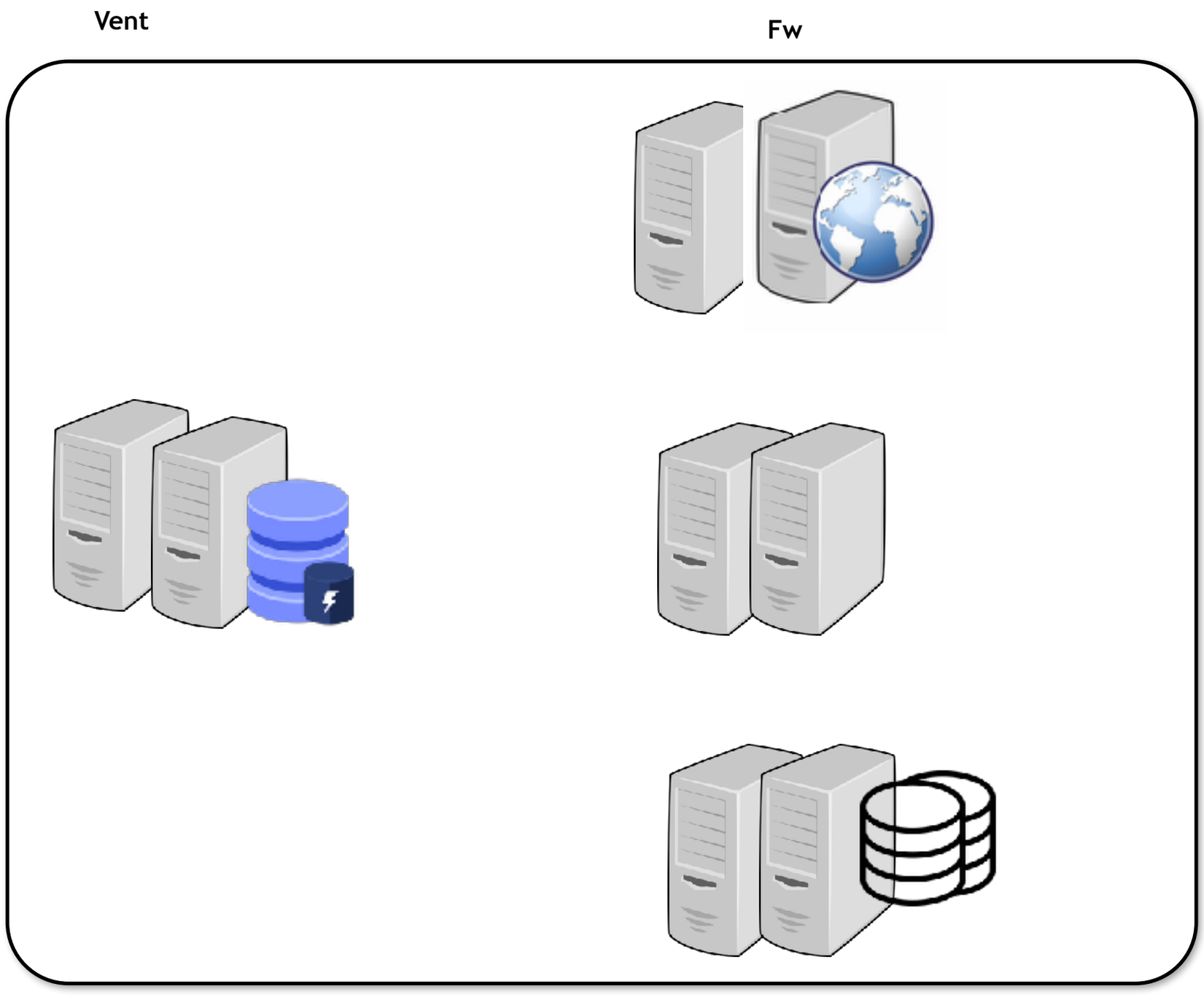
Logical view

White box view

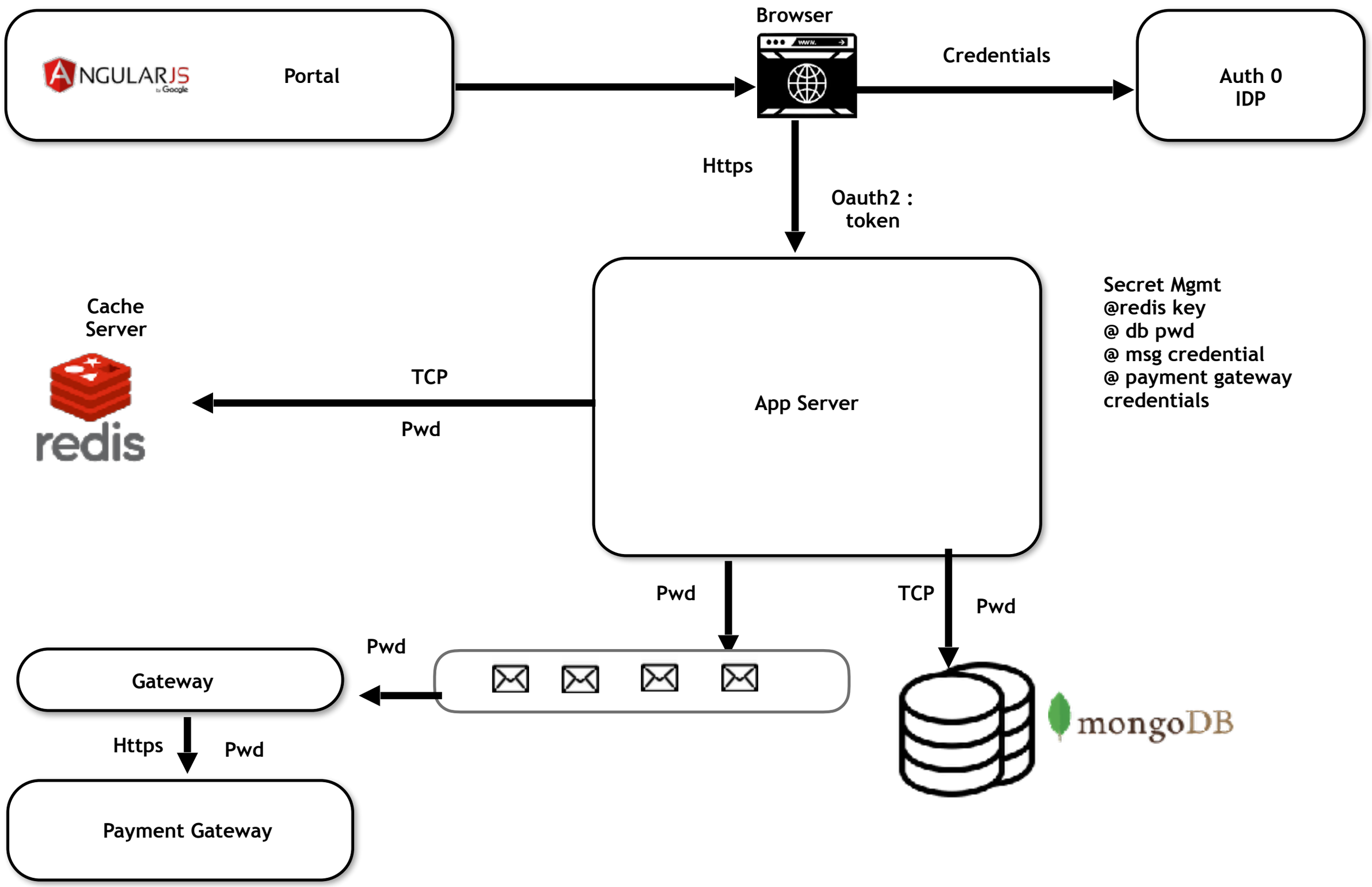


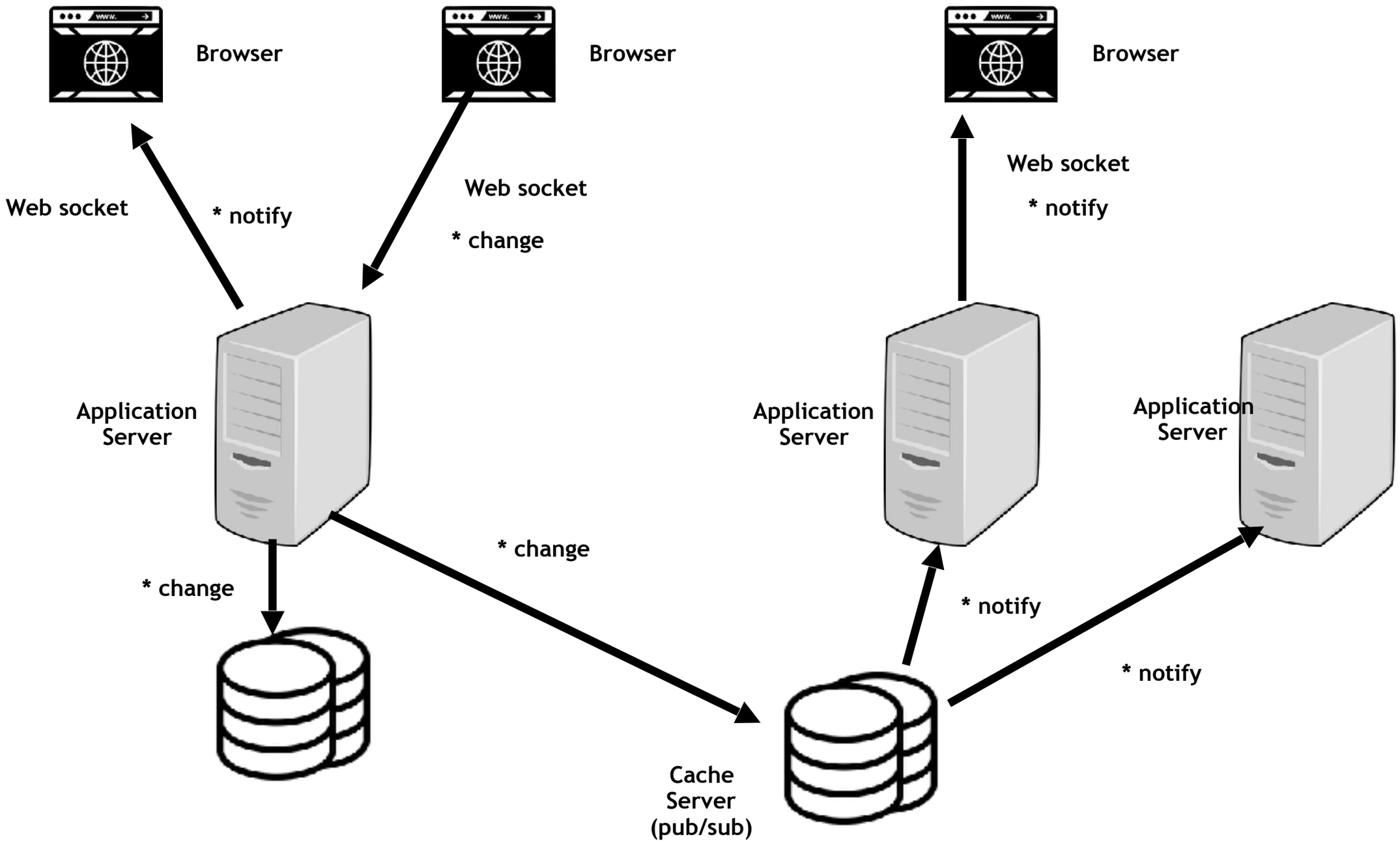
Infrastructure View

Physical view



Security View




Browser
Browser
Web socket
* notify
Web socket
* change
Application
Server
* change
* change
Cache
Server
(pub/sub)
Browser
Web socket
* notify
Application
Server
Application
Server
* notify
* notify

Eval

- ATAM (architecture trade-off analysis method)

- Identify all approaches (a1,a2,a3, …)
- Identify all scenarios - utility tree (s1, s2, s3, s4, …)
- Analyze

S1 -> a2,a3 +
S2 -> a3 +
S3 -> ?
S4 -> a1 ?

- Identify all scenarios - brain storm (us1, us2, us4, …)
- Analyze

US1 -> ?
US2 -> a3