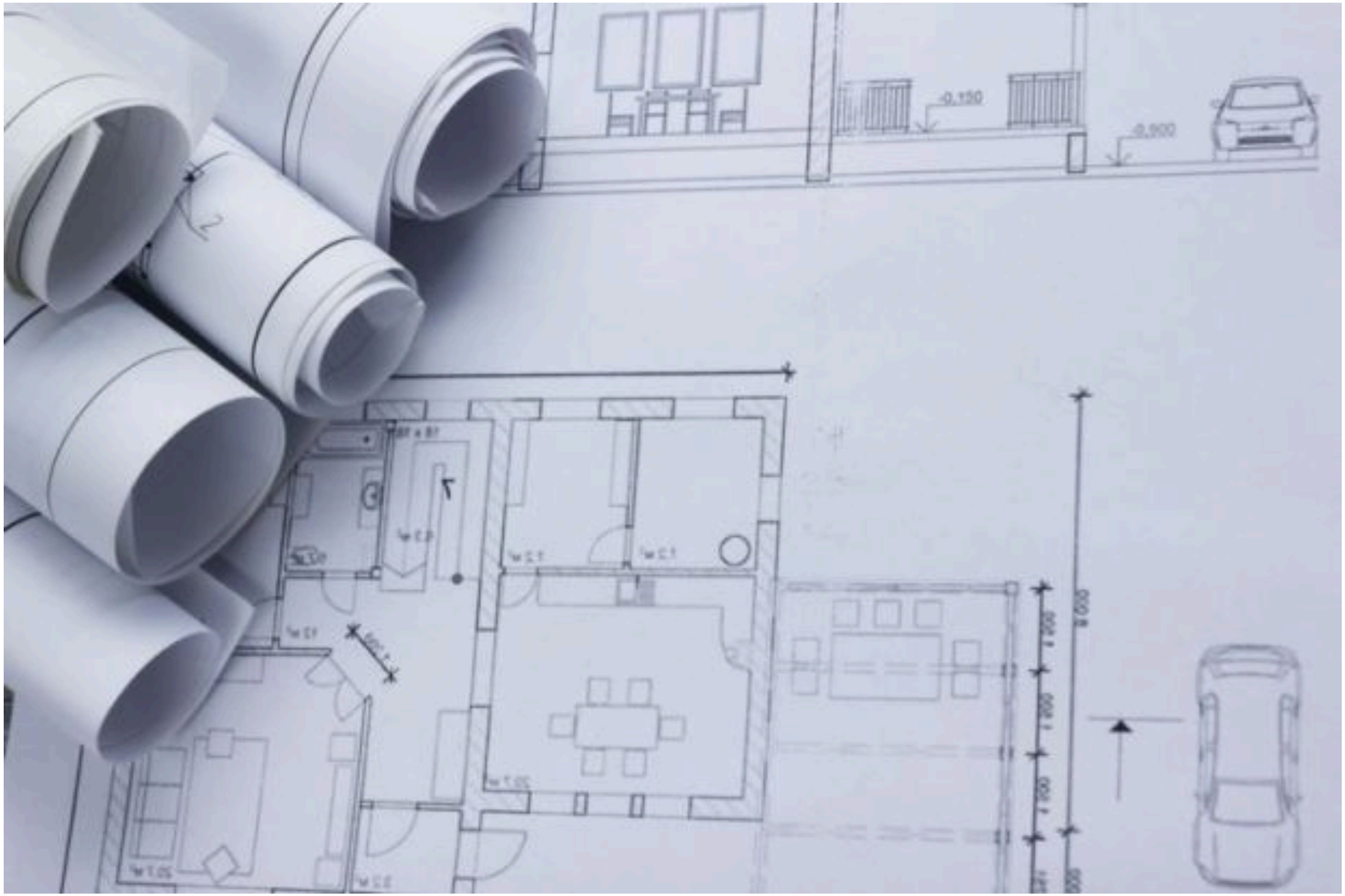


Software Architecture

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

Architecture and Design

Engineering System Quality

Quality

Quality attributes

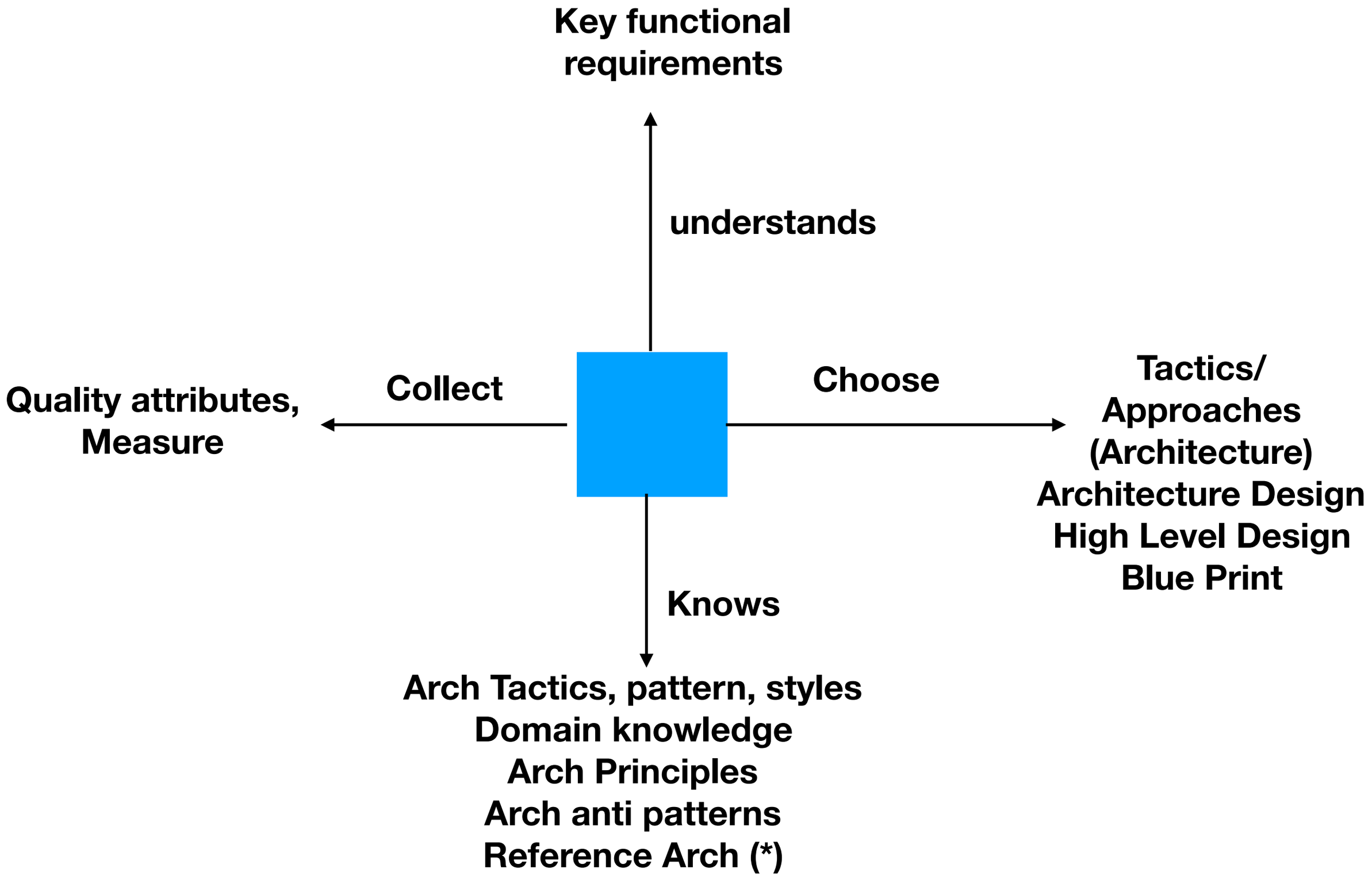
- Availability
- Security (Trust)
- Maintainability
- Reliability (Trust)
- Scalability (volume)
    - CPU, I/O, Memory, Disk
- Performance (resource usage)
    - Cpu, memory, disk, I/O
- Usability
- Robustness (Rugud)
- Portability
- Interoperability
-

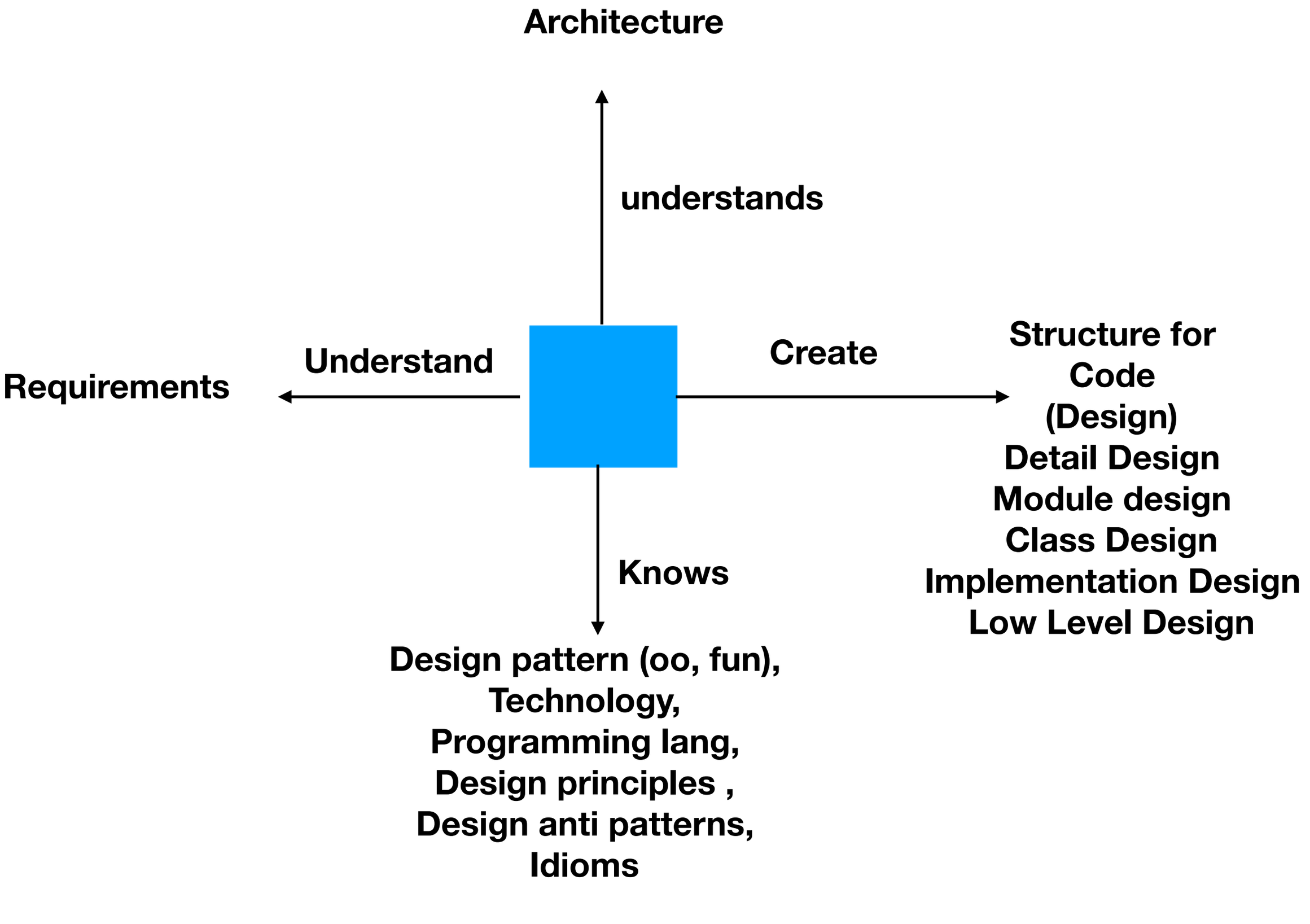
Measure

- % uptime / downtime
- Response time
- tps
- Code coverage
- No of clicks
- Count
- Probability

Engineering Tactics

- Parallel
- Caching
- Chunking
- Lazy Loading
- Throttling
- Monitoring
- Unit test
- Layered pattern
- ACID (transaction)
- Multi tenant
-


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 (*)


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 Design

Code Design

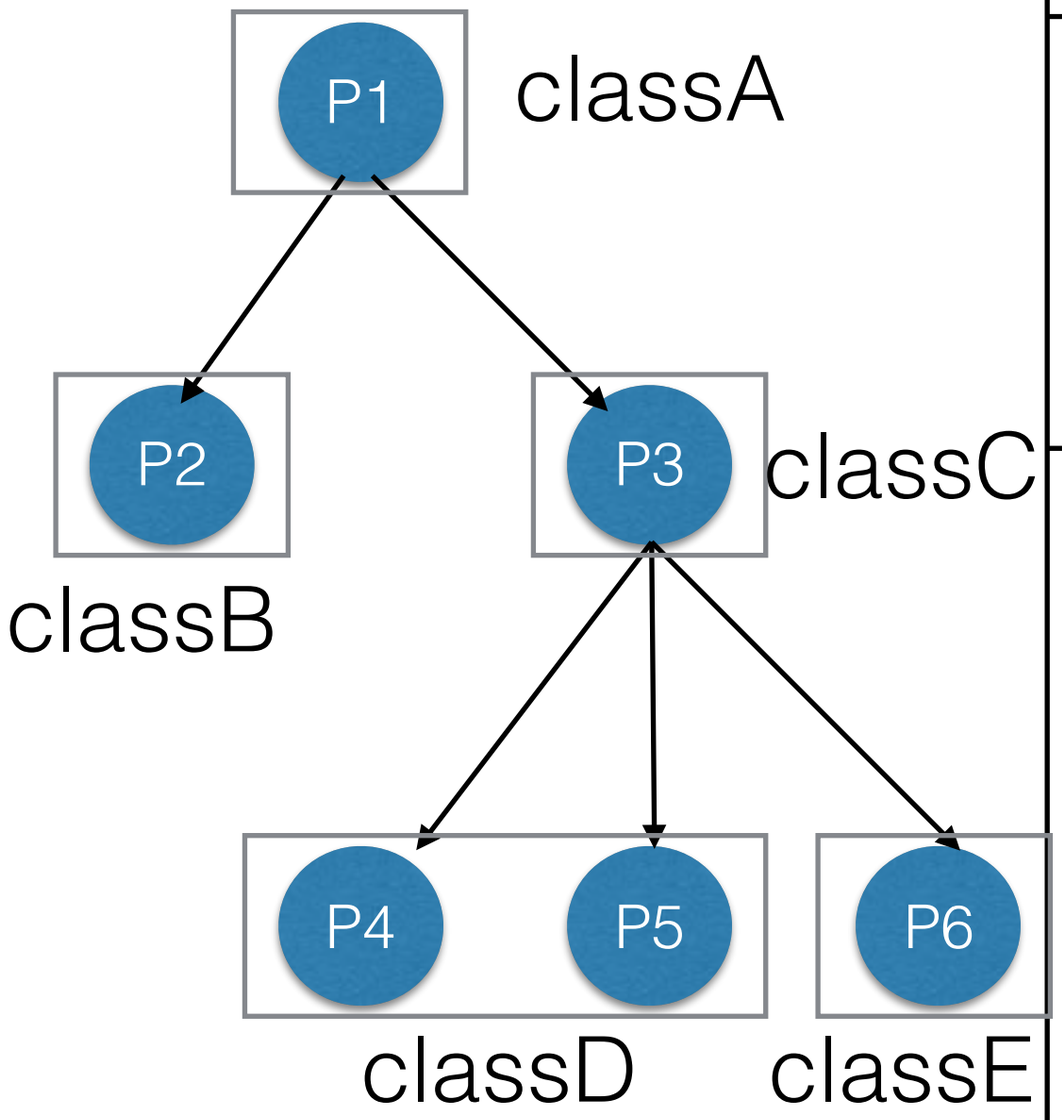
Architecture and Design

“Address System Quality”

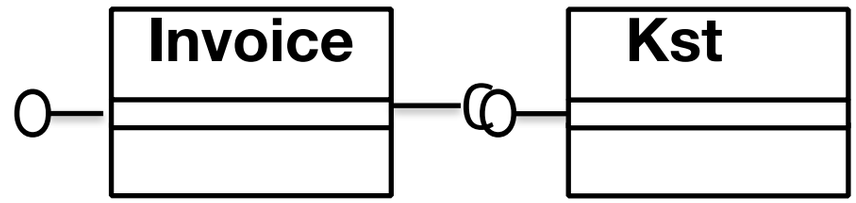
“Code Quality”
 # class design
 # module design
 # low level design

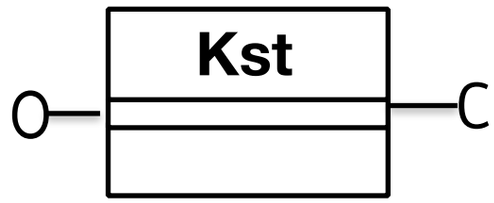
Procedural, OO, functional

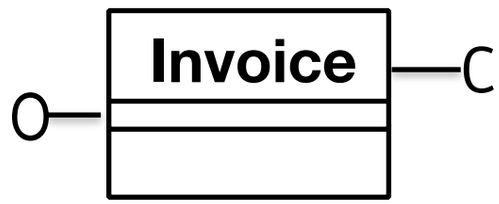
Procedural Prog (tree)

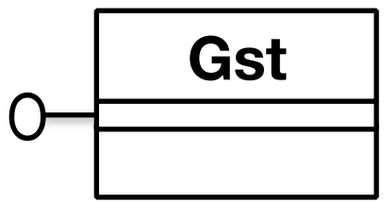


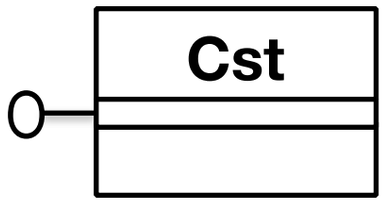
OO Prog (Lego)



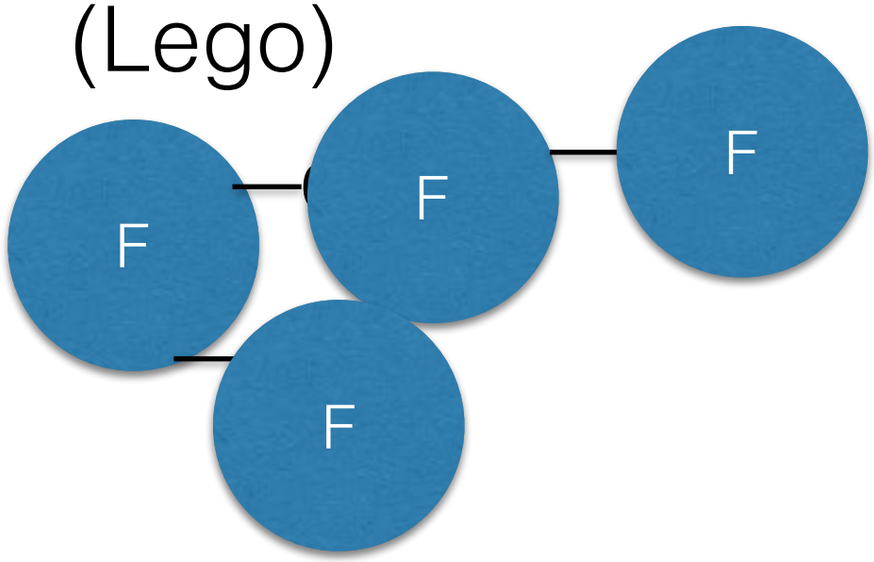


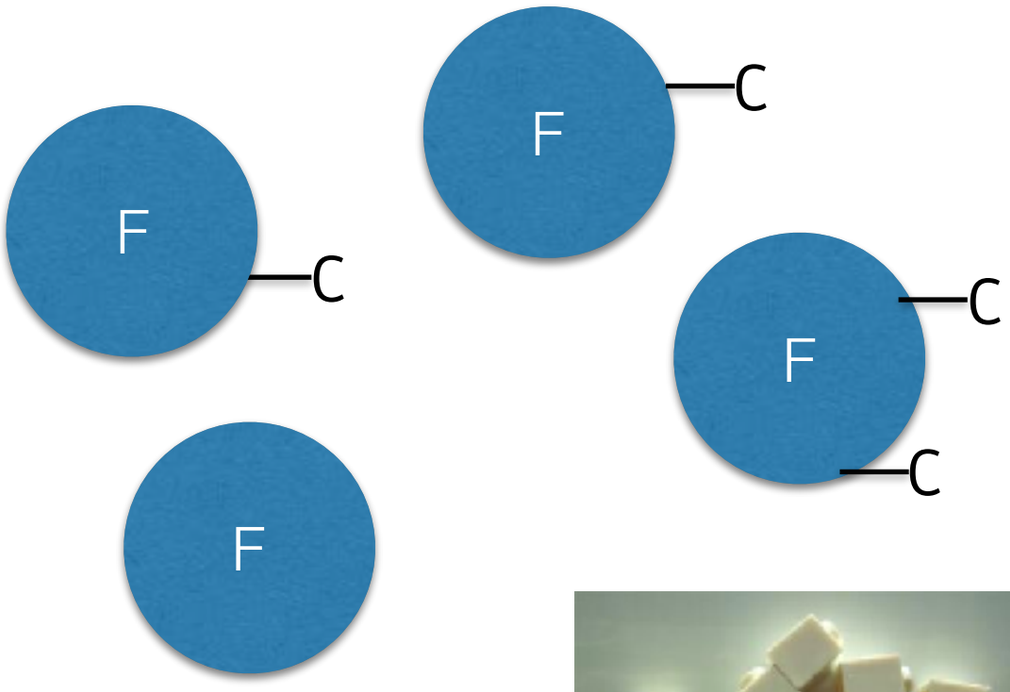




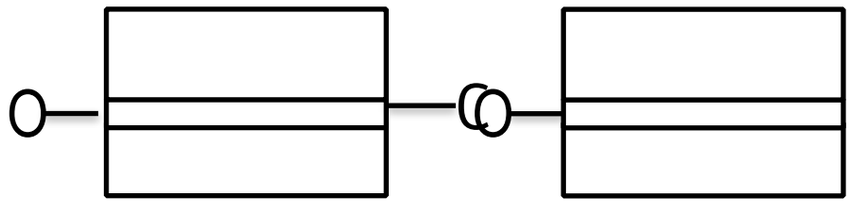


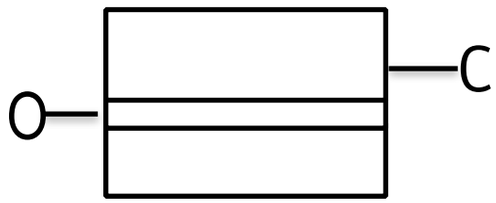
Functional Prog (Lego)

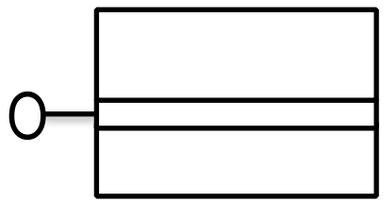


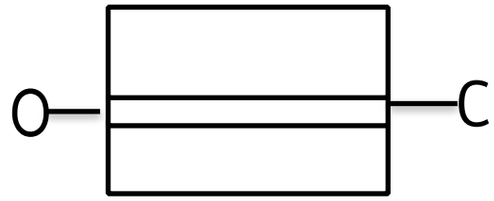


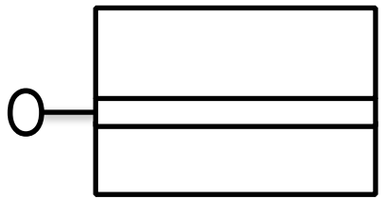
OO Prog (Lego)





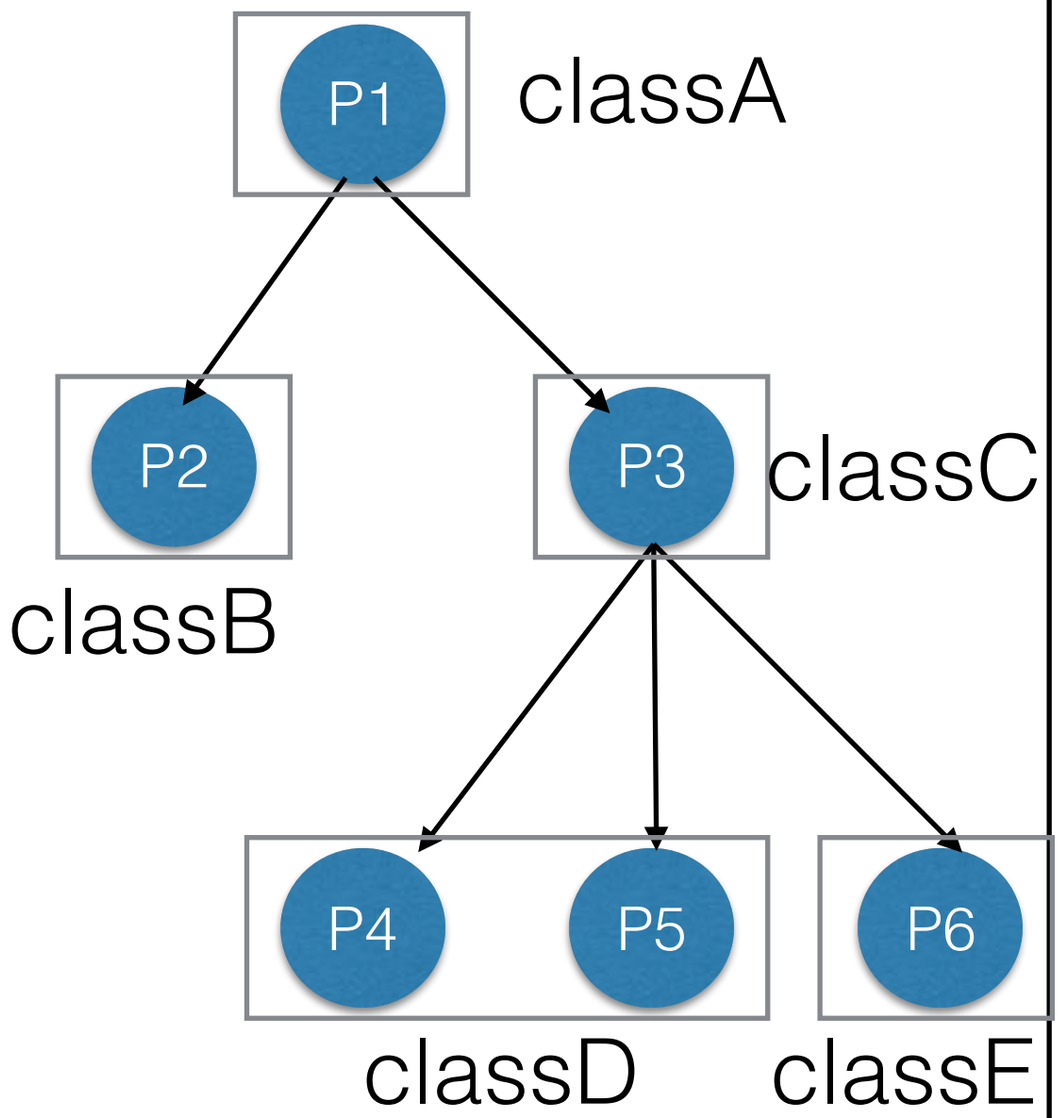






Procedural Prog

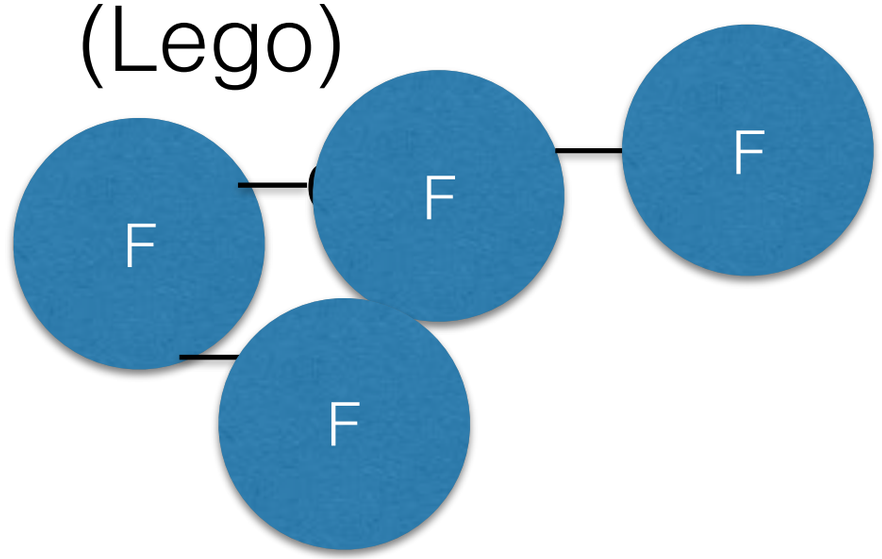
(tree)

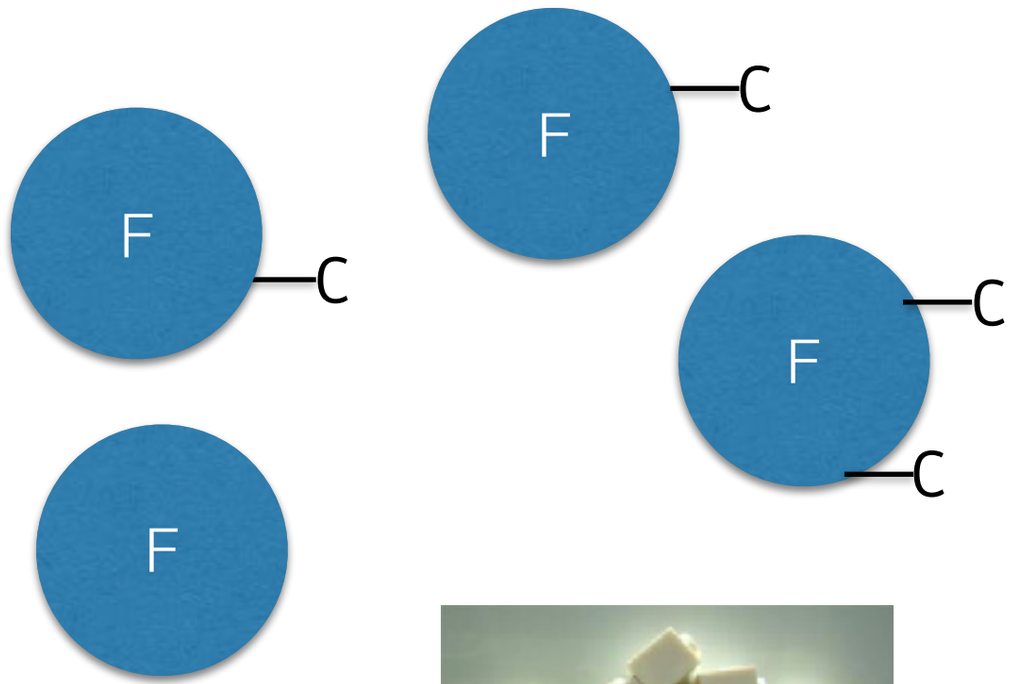


(top down)

Functional Prog

(Lego)



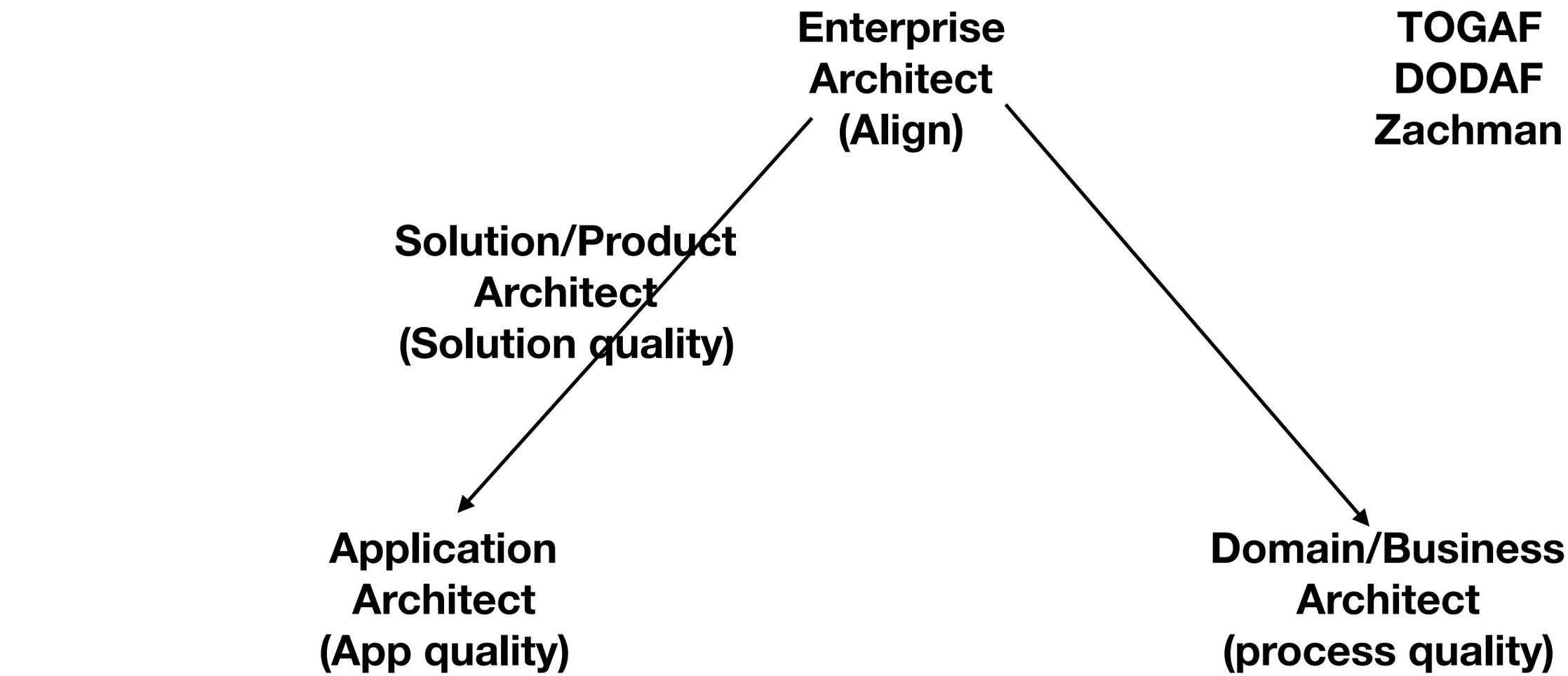


Coding Style	Proc	OO	Fun
Performance	-	-	+ +
Unit Testability	- -	+ +	+ + +
Time to develop/ cost	+ +	- -	+
Manage Large Code	- -	+ +	+
Learning Curve	+ +	- -	-
Language	C, java, js, py	Java, js, py	Java, js, py, scala, kotlin, Haskel

Before

Engineering v/s Tuning

After


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

Sounce: User
Trigger: Enters User Details
Artifact: In the System
Environment: Feeds wrong input and clicks submit
Response: The incorrect details are highlighted
Measure: User is able to correct the errors in data

Source: Developer
Trigger: Runs unit test suite
Artifact: Payment module
Env: After adding a new mode of payment
Response: All tests should be successful
Measure: More than 80 % of code coverage in payment module

Deal of the day

 # one product every day
 # place order
 # cash on delivery

1. Context view

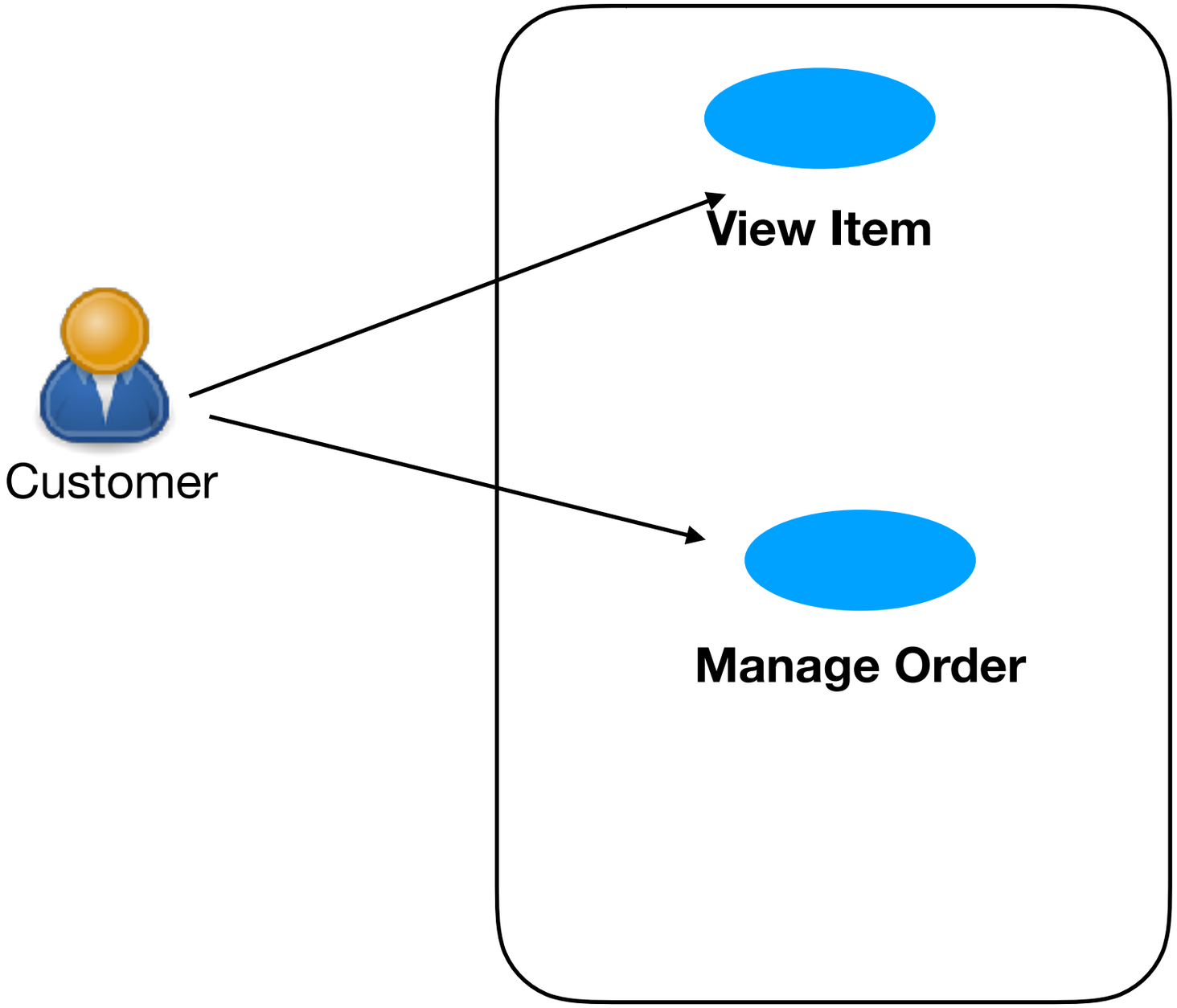
Eagle’s view

Collect Requirement

2. functional View

Understand Domain

Key functionality of the system



3. Quality requirement

Collect NFR

Source (who)	Trigger (action)	Artifact (module)	Environment (context)	Response (output)	Measure (metrics)
As a Customer	I place a order	From the Portal	During peak load (10000)	Order is placed and confirmation is shown	In < 3 sec.
Consumer Web portal	Send a duplicate order request	In the System	Request already sent	Customer is notified that the original Order was successful.	but the customer is not double-charged
Unauthorized user	Place an order	In the system	During normal operations	The system Identifies the request and blocks access	100% probability

4. Constraints

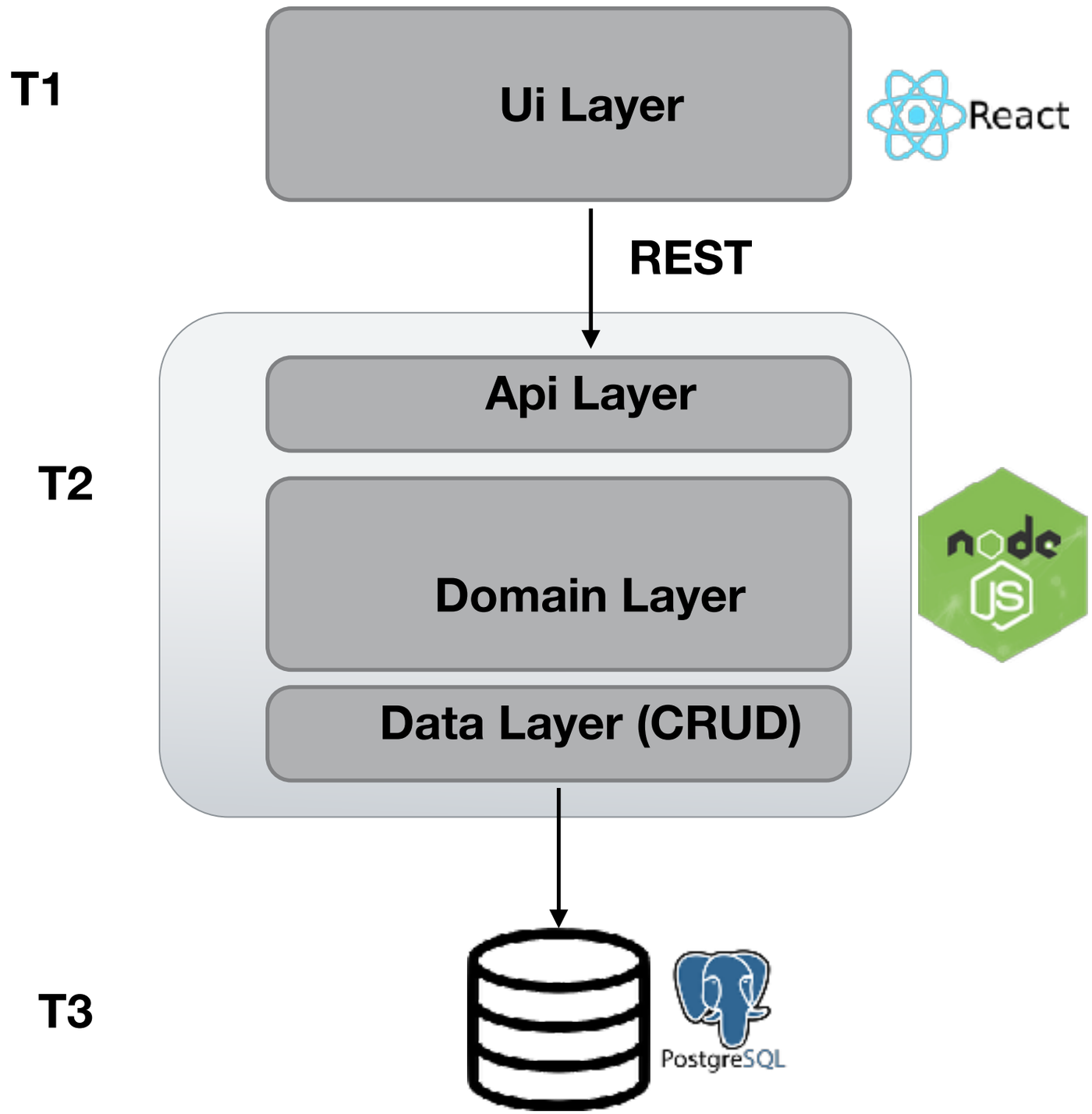
Collect Context

- Should work on IE 11 and chromium browsers
- Should use open source technologies
- Should follow GDR regulations for data management

Define Architecture

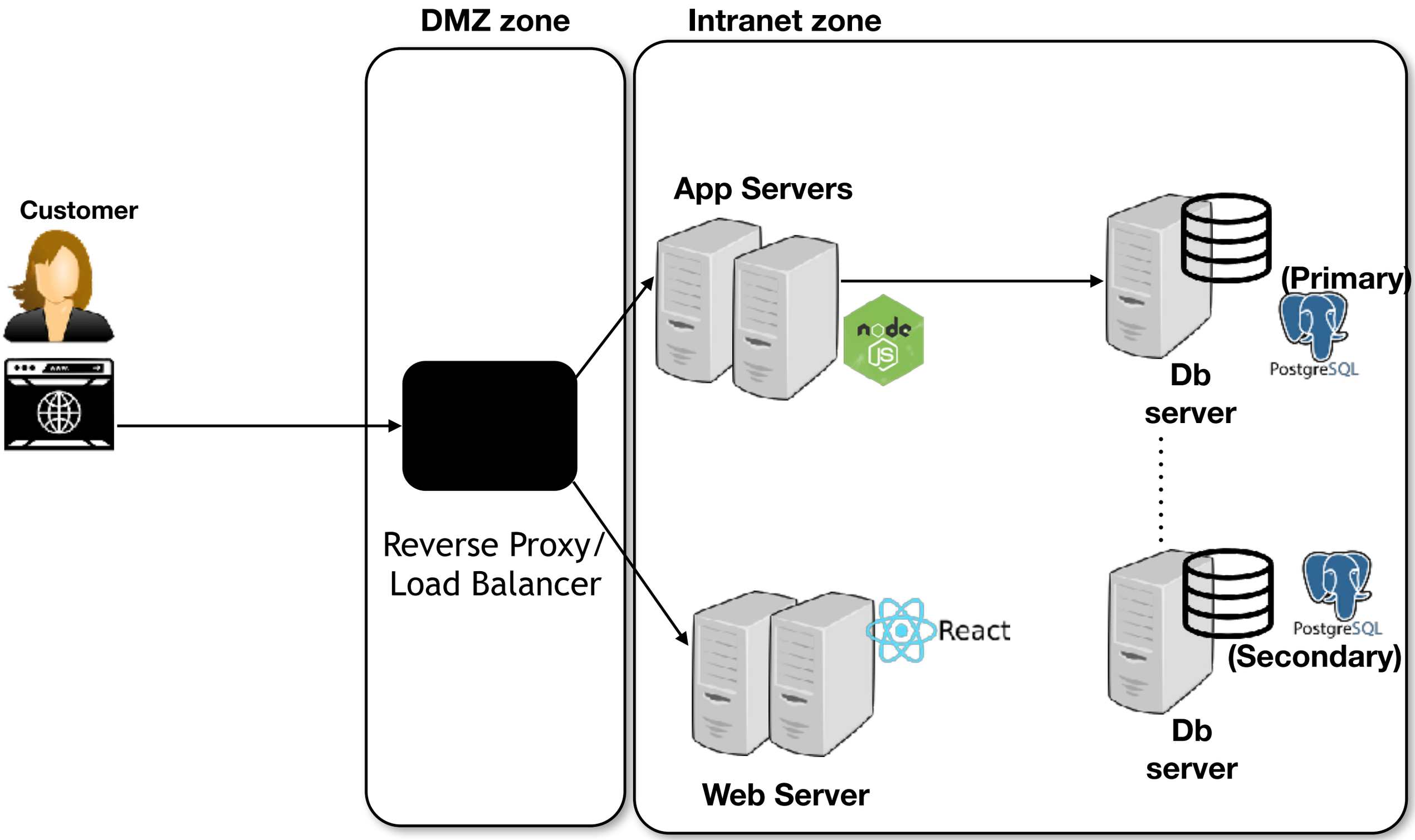
5. Logical View

Decompose system

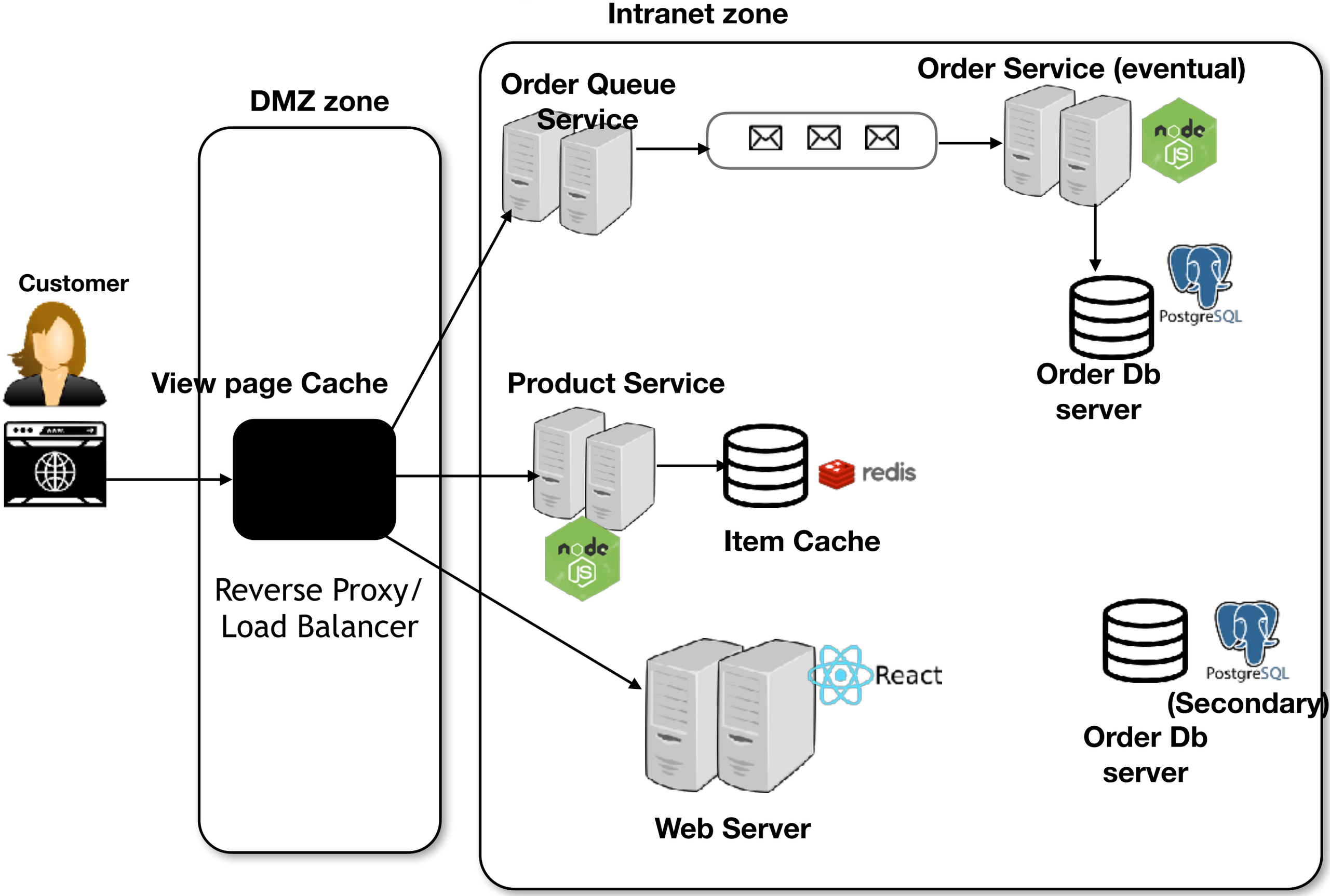


6. Deployment View (IAAS)

Physical view



6. Deployment View (IAAS)



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

Authentication

1. By what you know (*)
    1. Pwd (Oaith2 +openID connect)
    2. pin
    3. secret
2. By what you have
    1. otp
    2. Email
    3. RSA tokens
    4. Cert
3. By What you are
    1. retina
    2. voice
    3. face
    4. finger
    5. dna

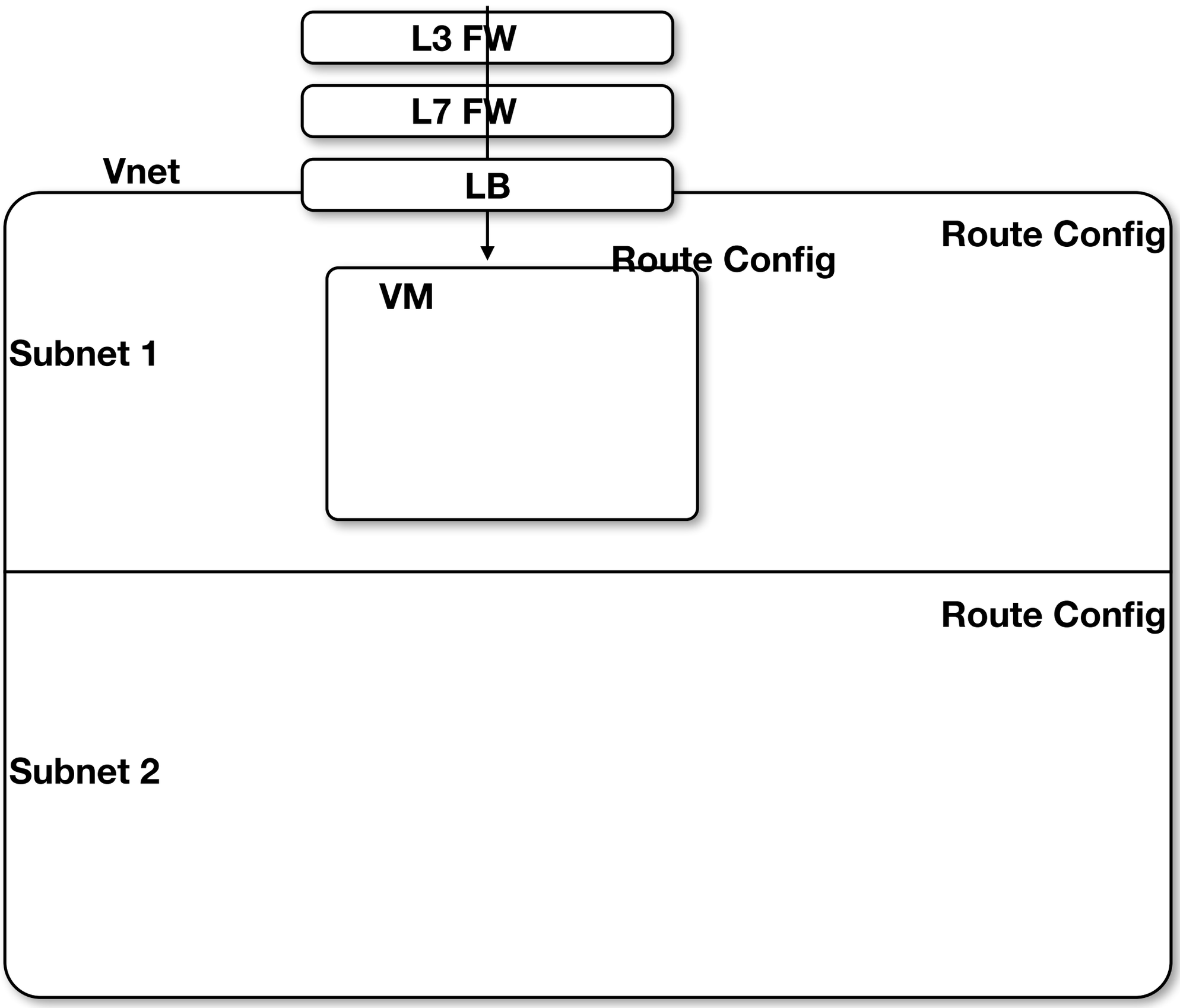
7. Security View

Application Security

Infra Security

Application Security

Authentication (First Defense)	Who are you
Authorization	What can you do
Audit (Last defense)	What did you do
Data Security # In Transit # In Rest	
Input Validation	XSS, CSRF, injection, …
Exception Handling	
Session Handling	
Key management	


L3 FW
L7 FW
Vnet
LB
Route Config
Route Config
VM
Subnet 1
Route Config
Subnet 2

Infra Security

Vnet, subnet	
L3 Firewall (TCP)	Network friewall
L7 Firewall (Http)	Web application Firewall

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

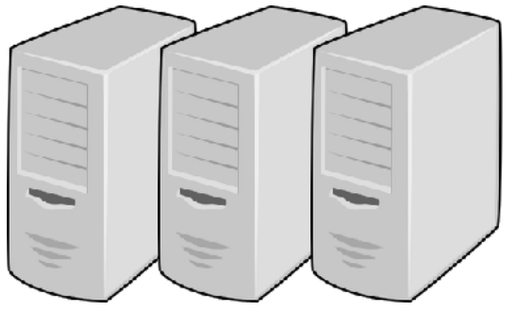
AAA

Scalability

App Servers

Db
server

Horizontal scaling / Scale out



App Servers

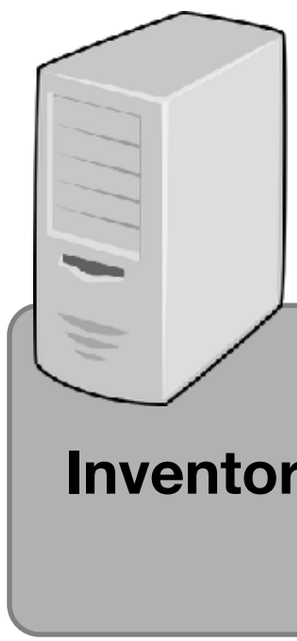
Vertical scaling / scale up

Db
server

Stateful

Customer

Session id



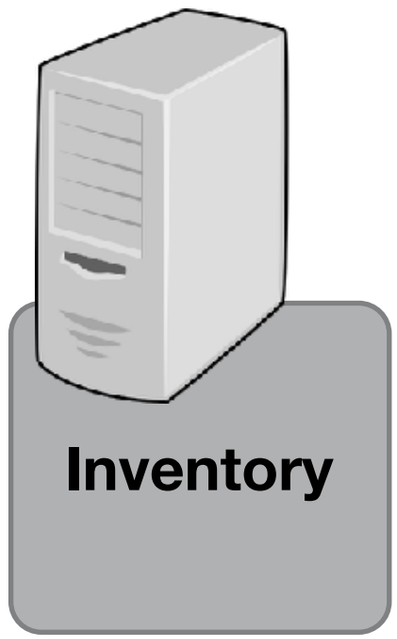
session data :
Selected Item
Qty

Inventory

Stateless

Customer

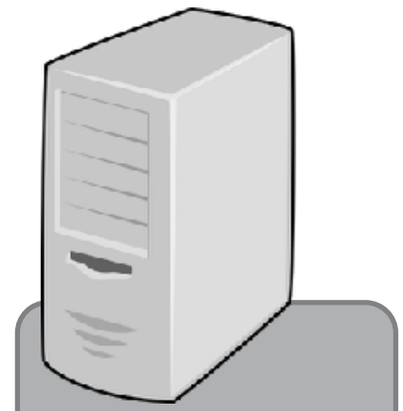
Session data:
Selected Item
Qty



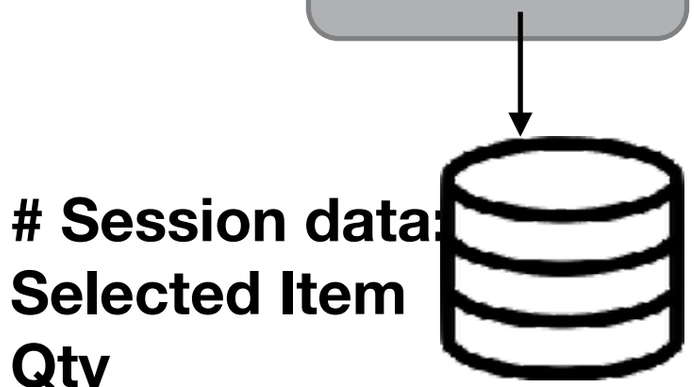
Inventory

Stateless

Customer

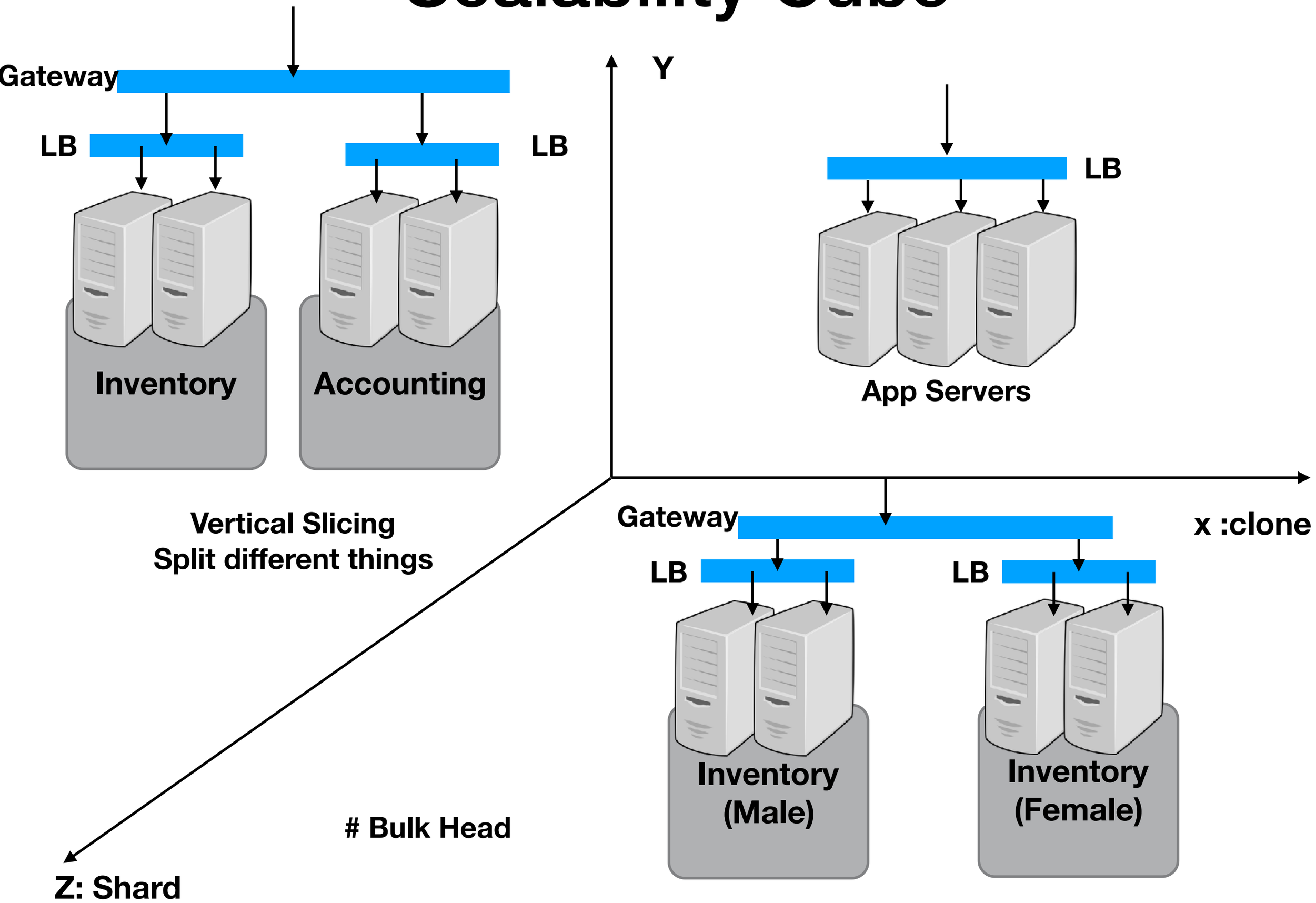


Inventory

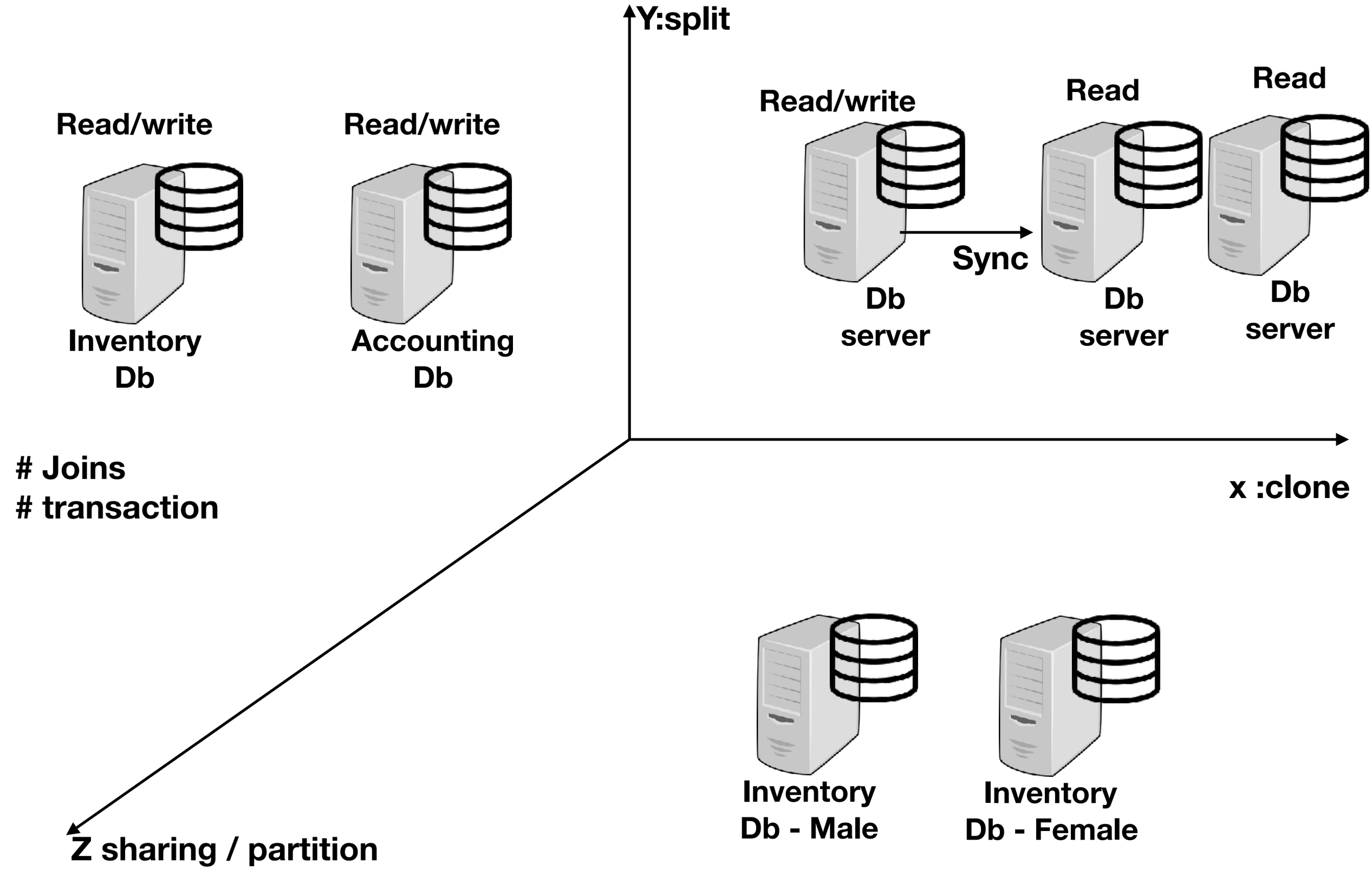


Session data:
Selected Item
Qty

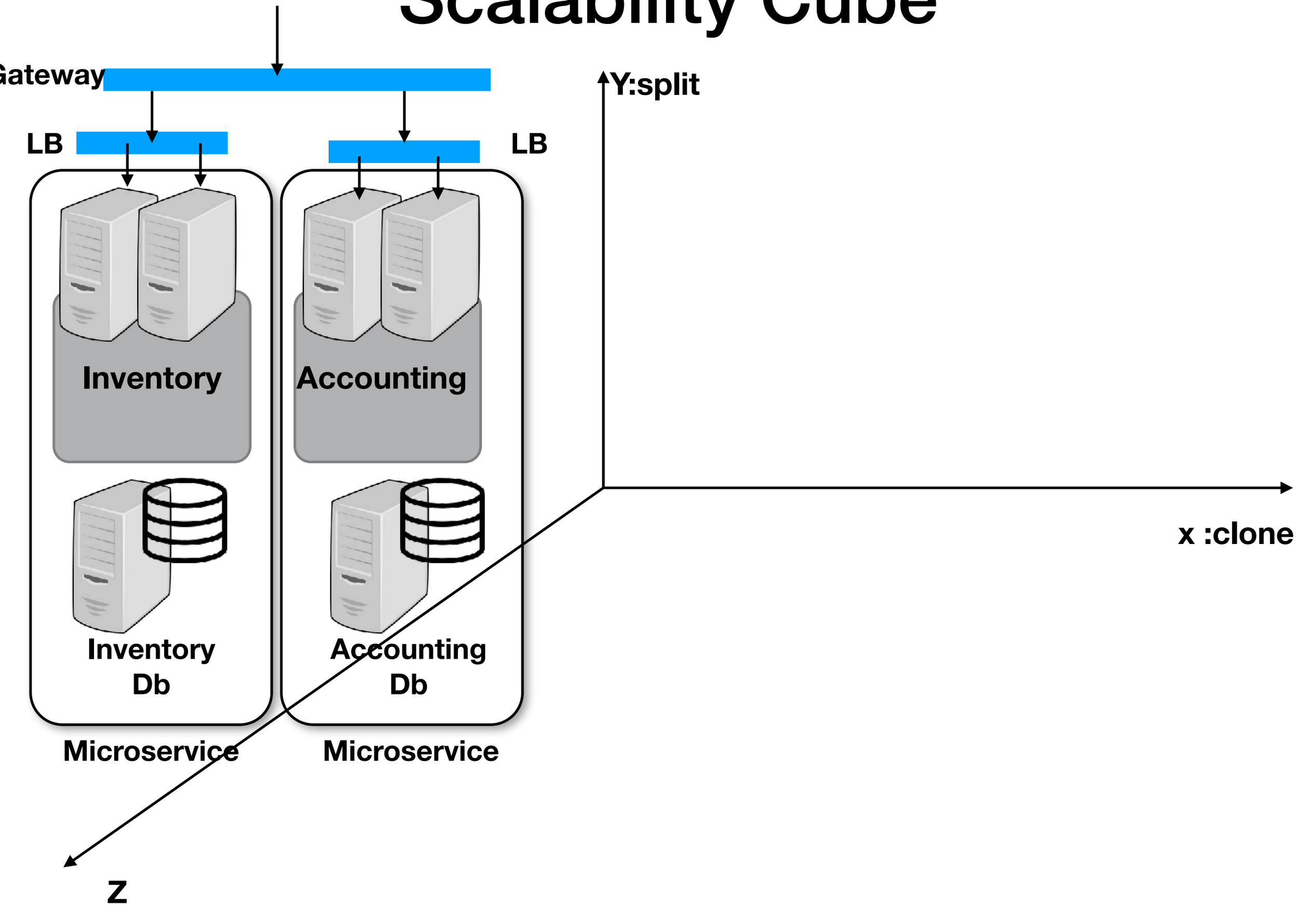
Scalability Cube



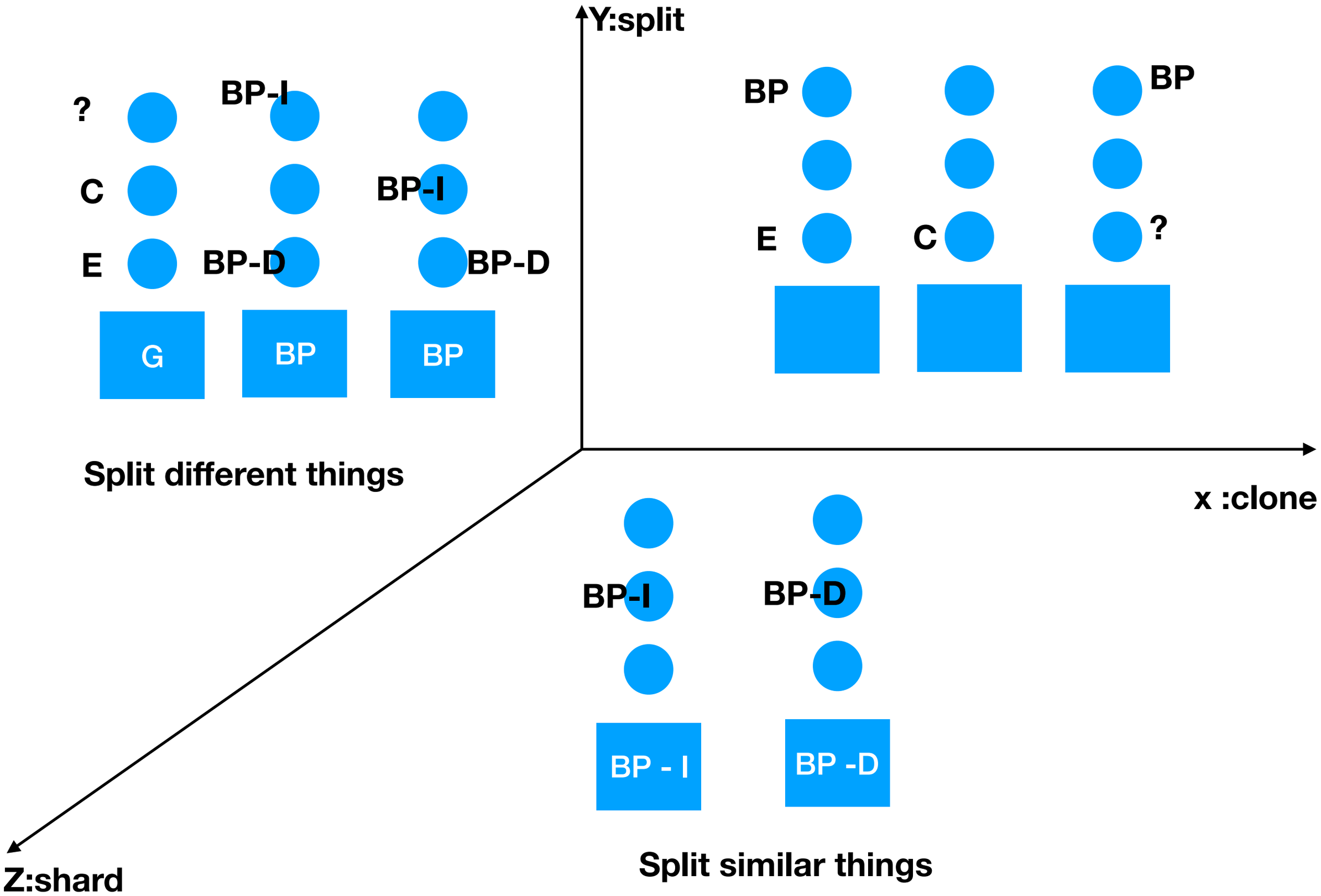
Scalability Cube



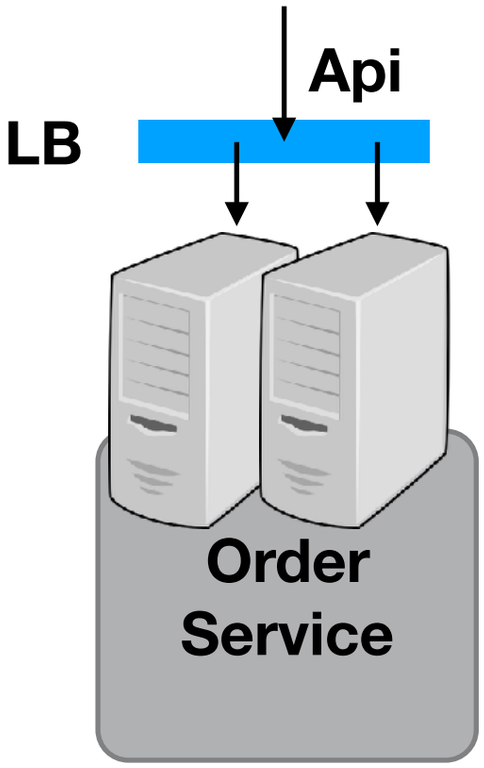
Scalability Cube



Scalability Cube



API call

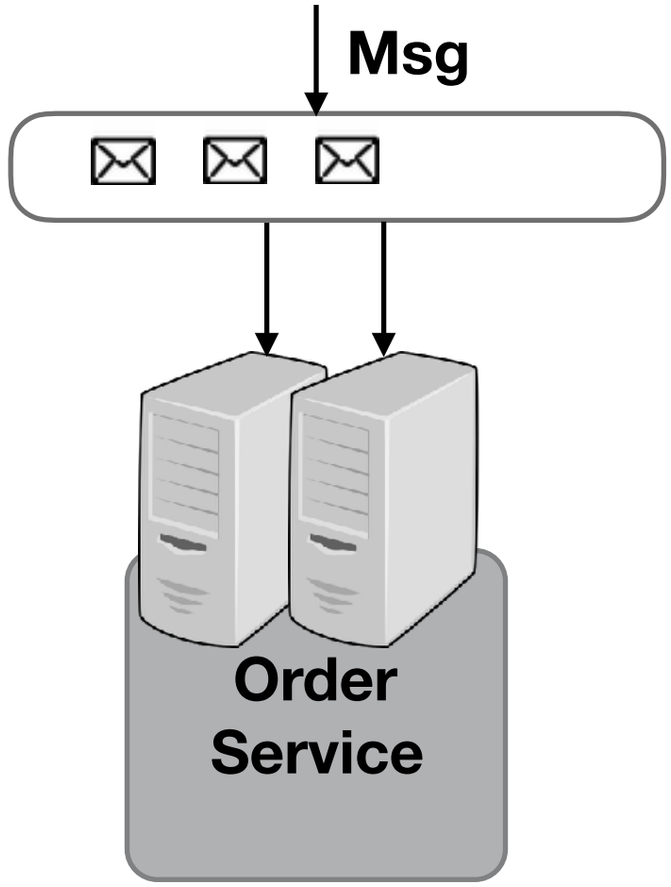


Less Scalable
no of server’s depends on peak load

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

loss of data
Because of throttle

Messaging

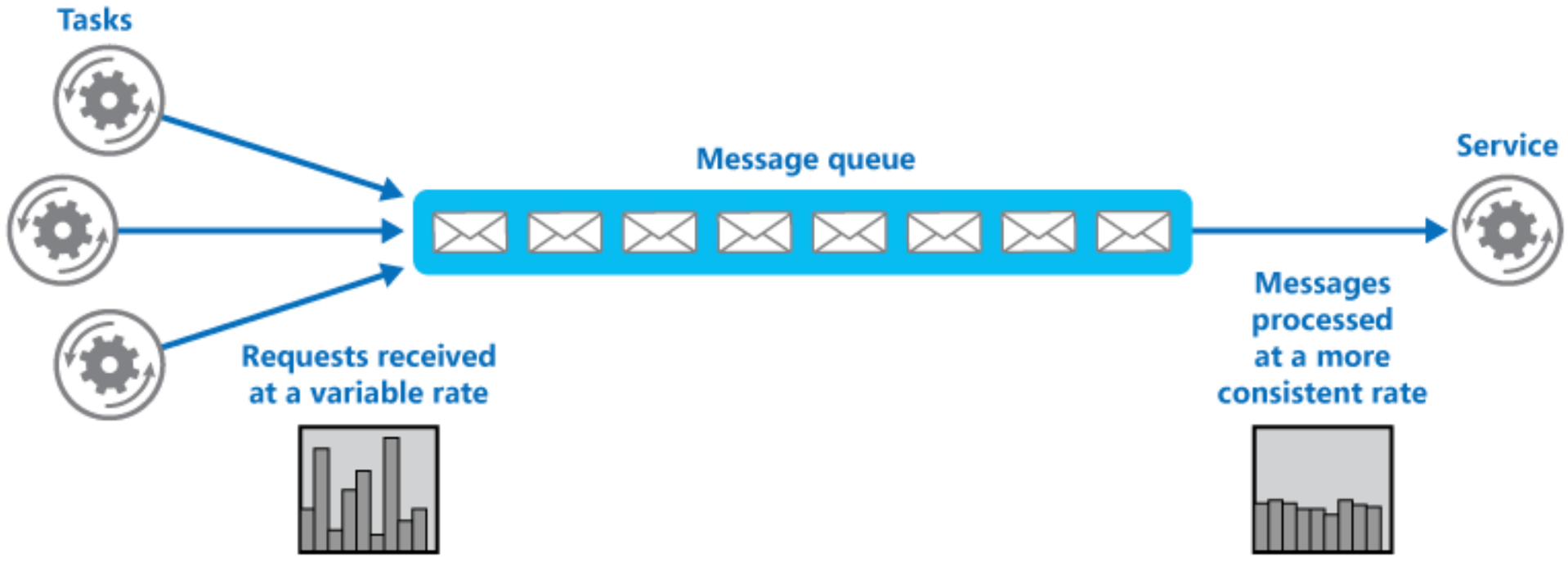


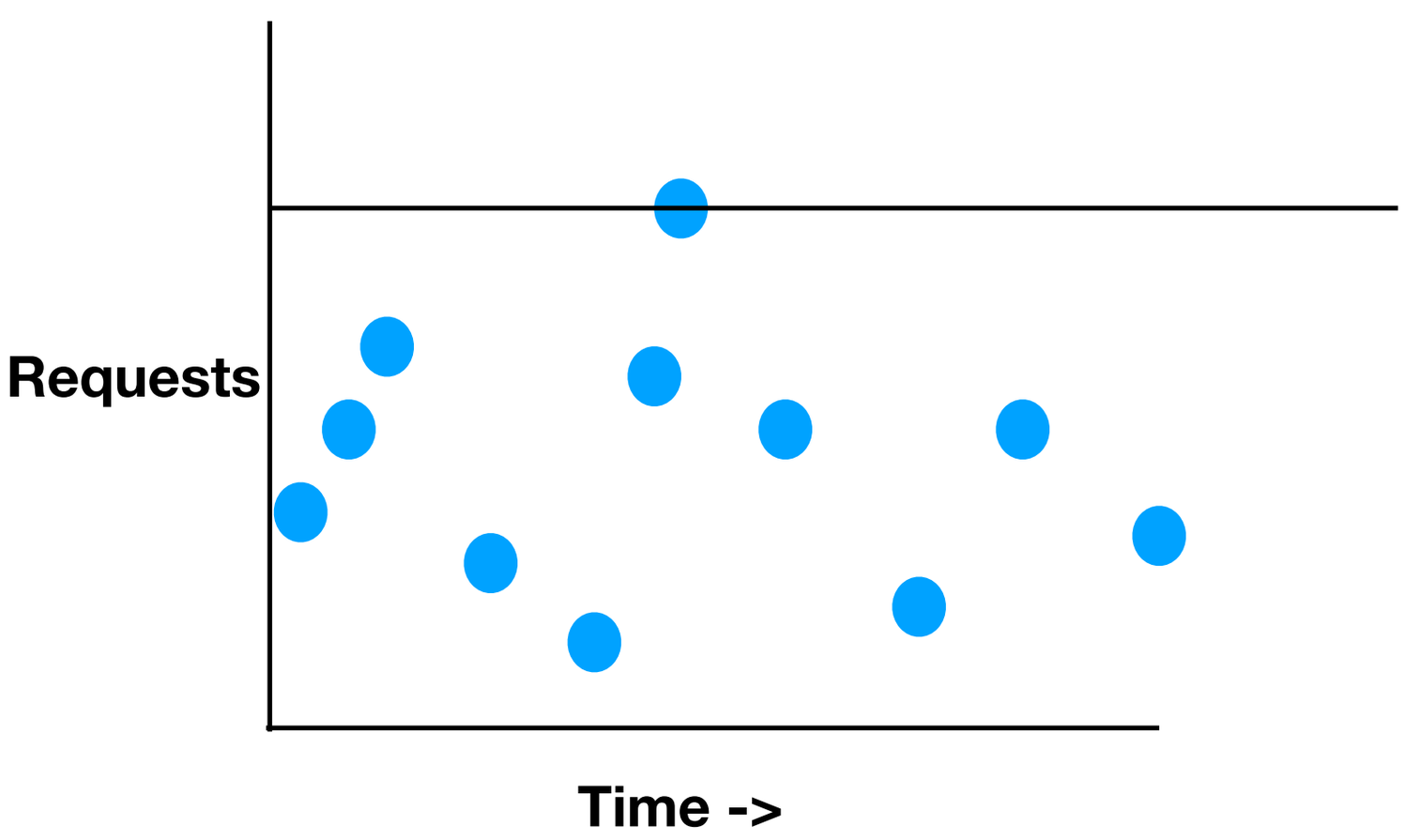
more scalable
no of server’s depends on Avg load
(Load Leveling)

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

no loss of data
Gets queued

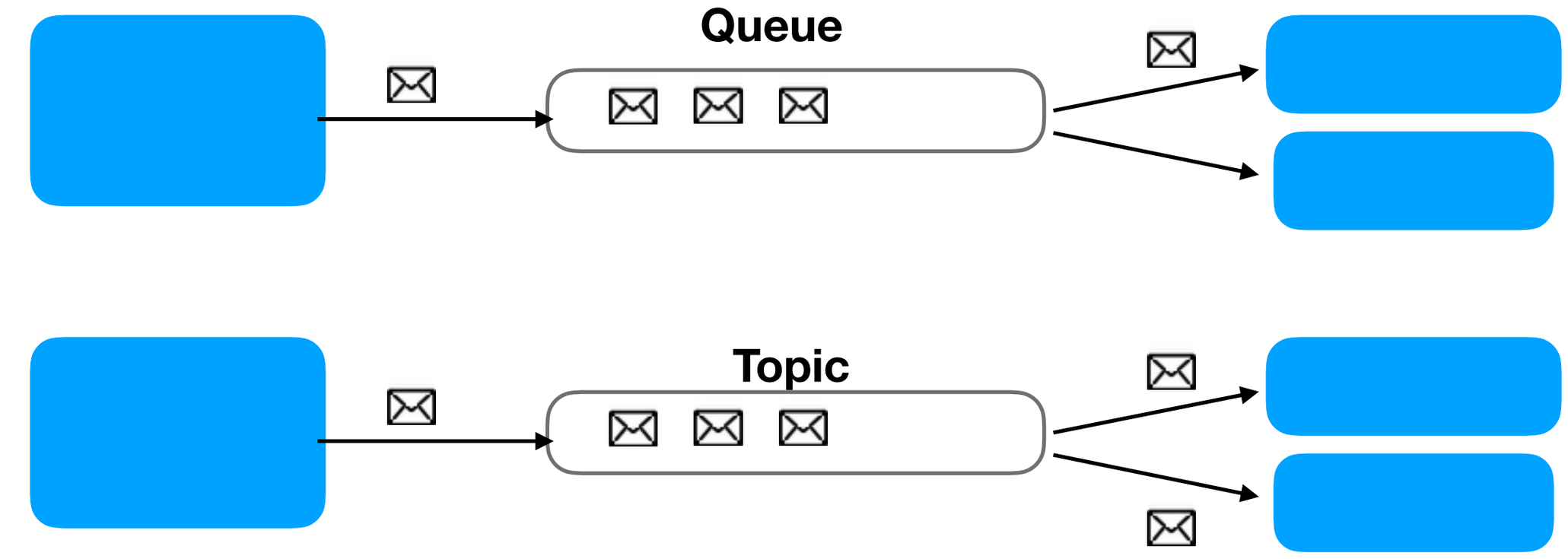
	API	Message	
Development time	++	- - (1:4)	Duplicate Message - Idempotent Unordered delivery
Protocol	Request-response	One Way	
Delivery	Ordered	Unordered	
Duplicate	No	Yes	
Consistency	Immediate (ACID)	Eventual (BASE)	
Learning Curve	++	- -	
Deployment	+ +	- -	


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


Queue
Topic
Hub

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

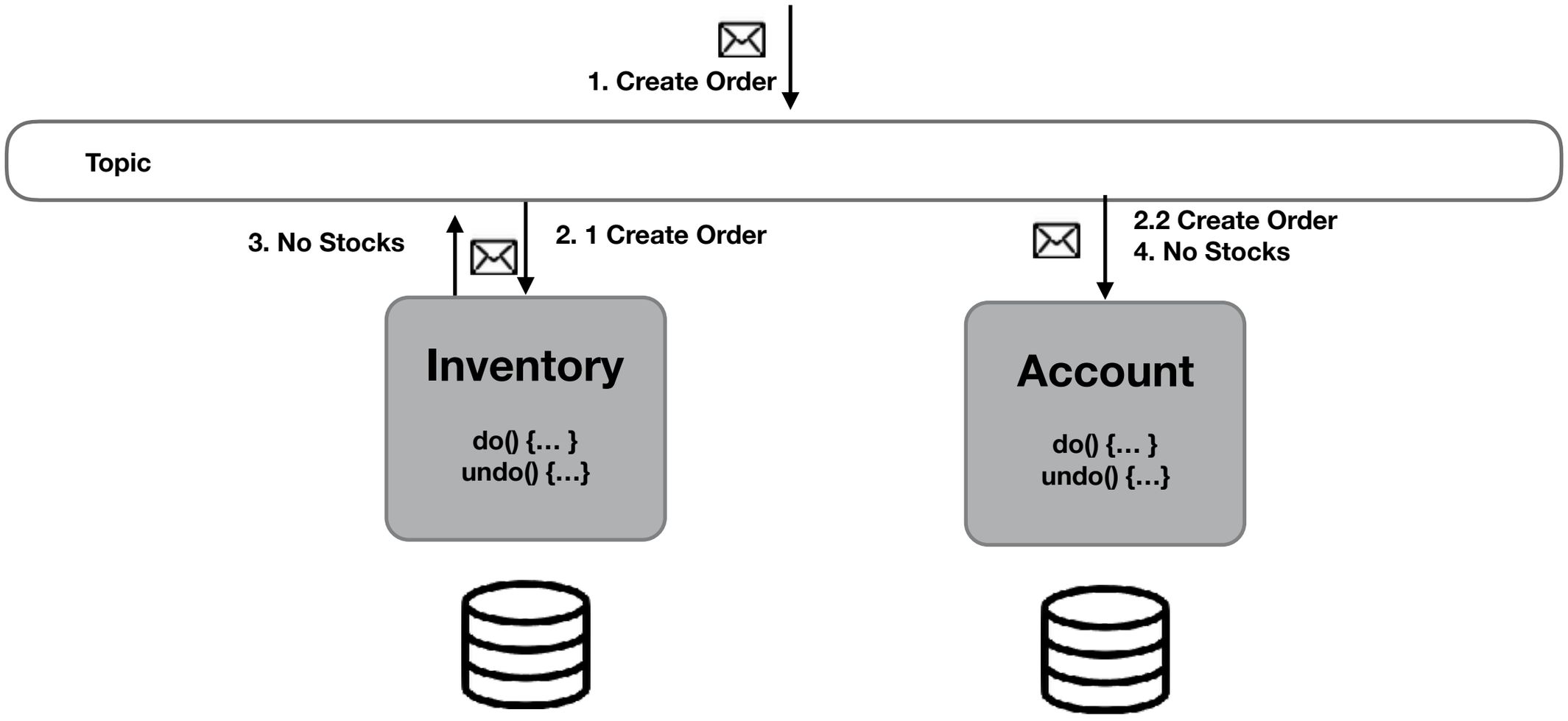
Chunky call (batch)

Caching

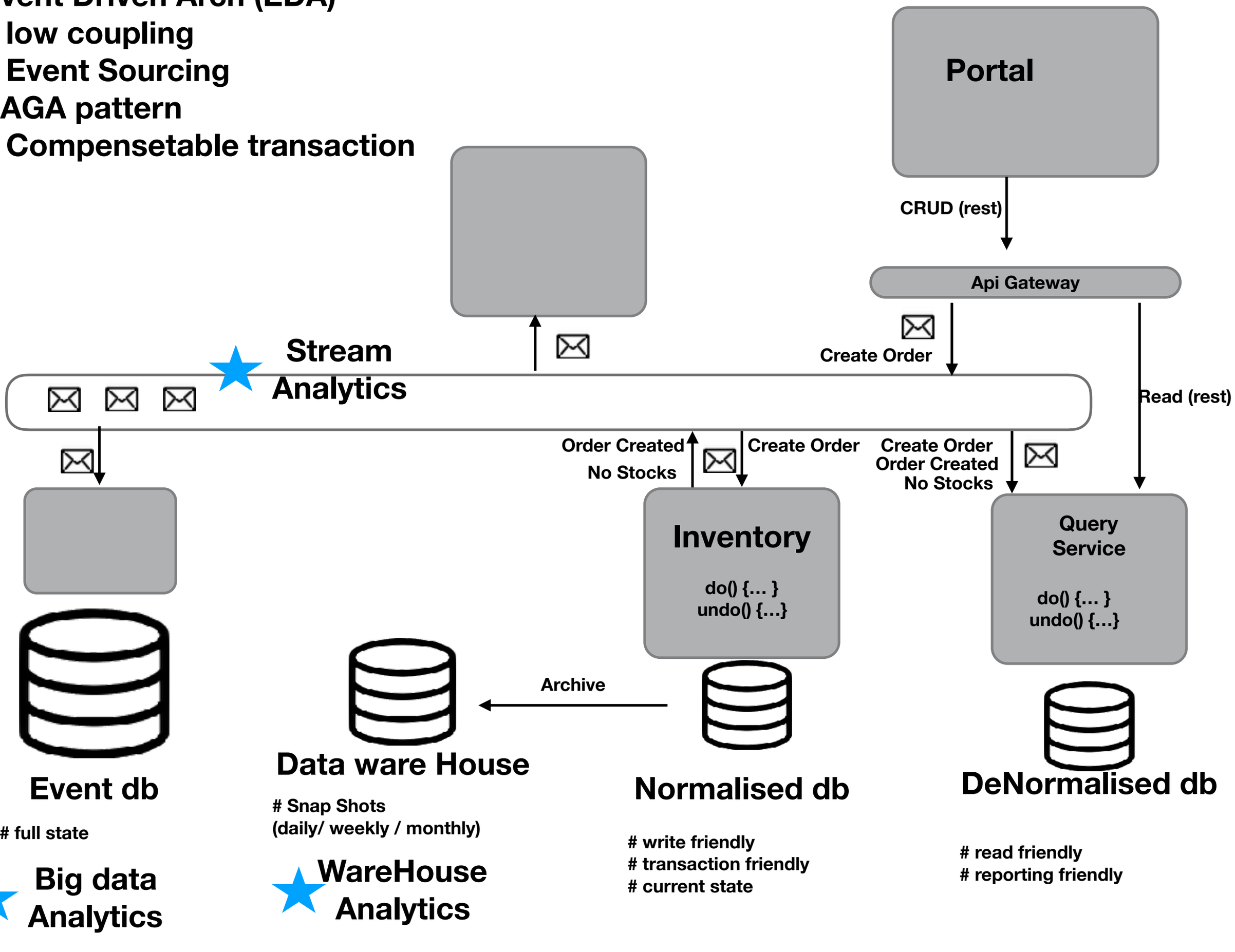
Lazy Loading

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

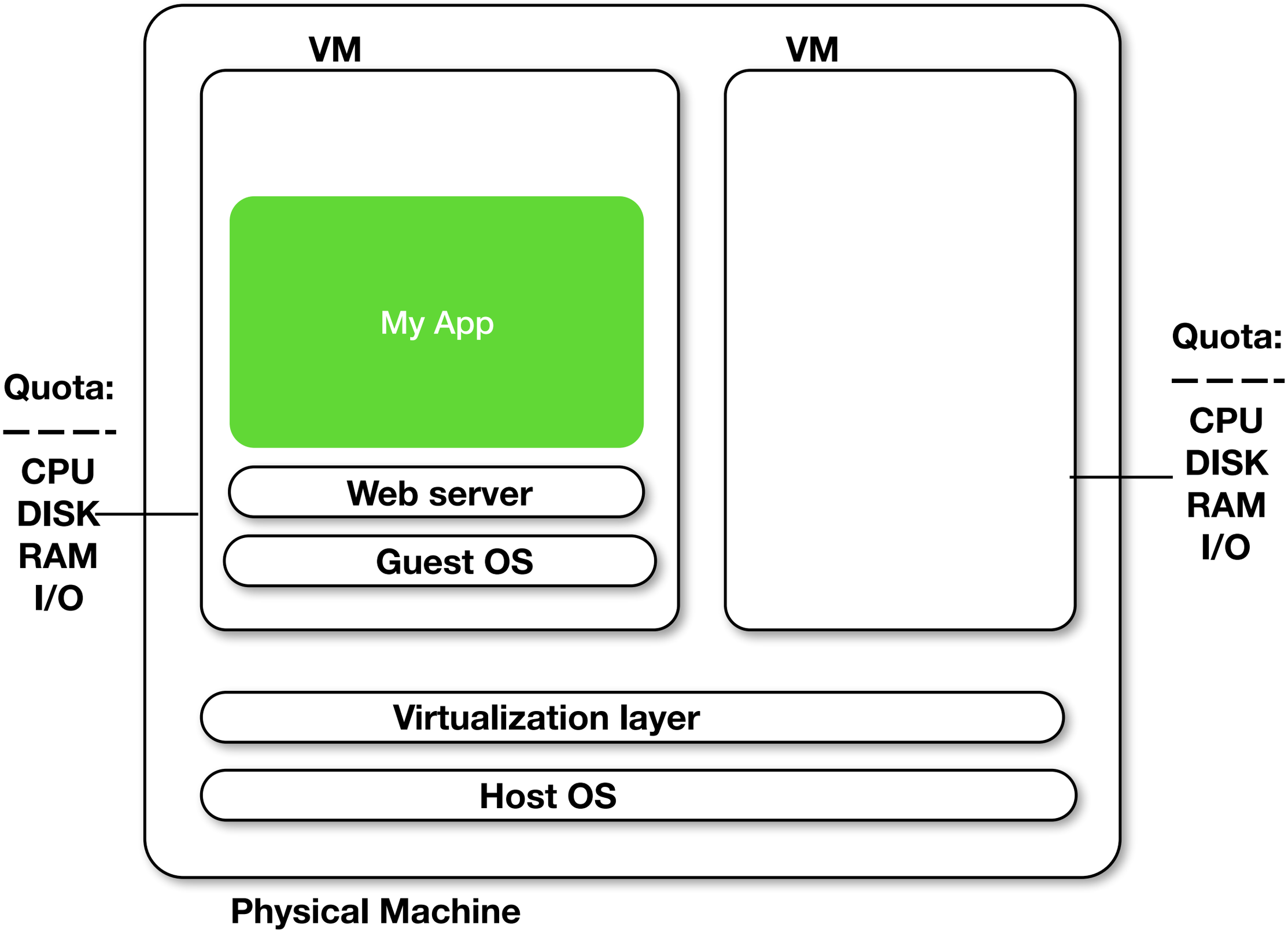
- web server
- Serverless Webserver
- Containerized Server
- Serverless Containerized Server

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)

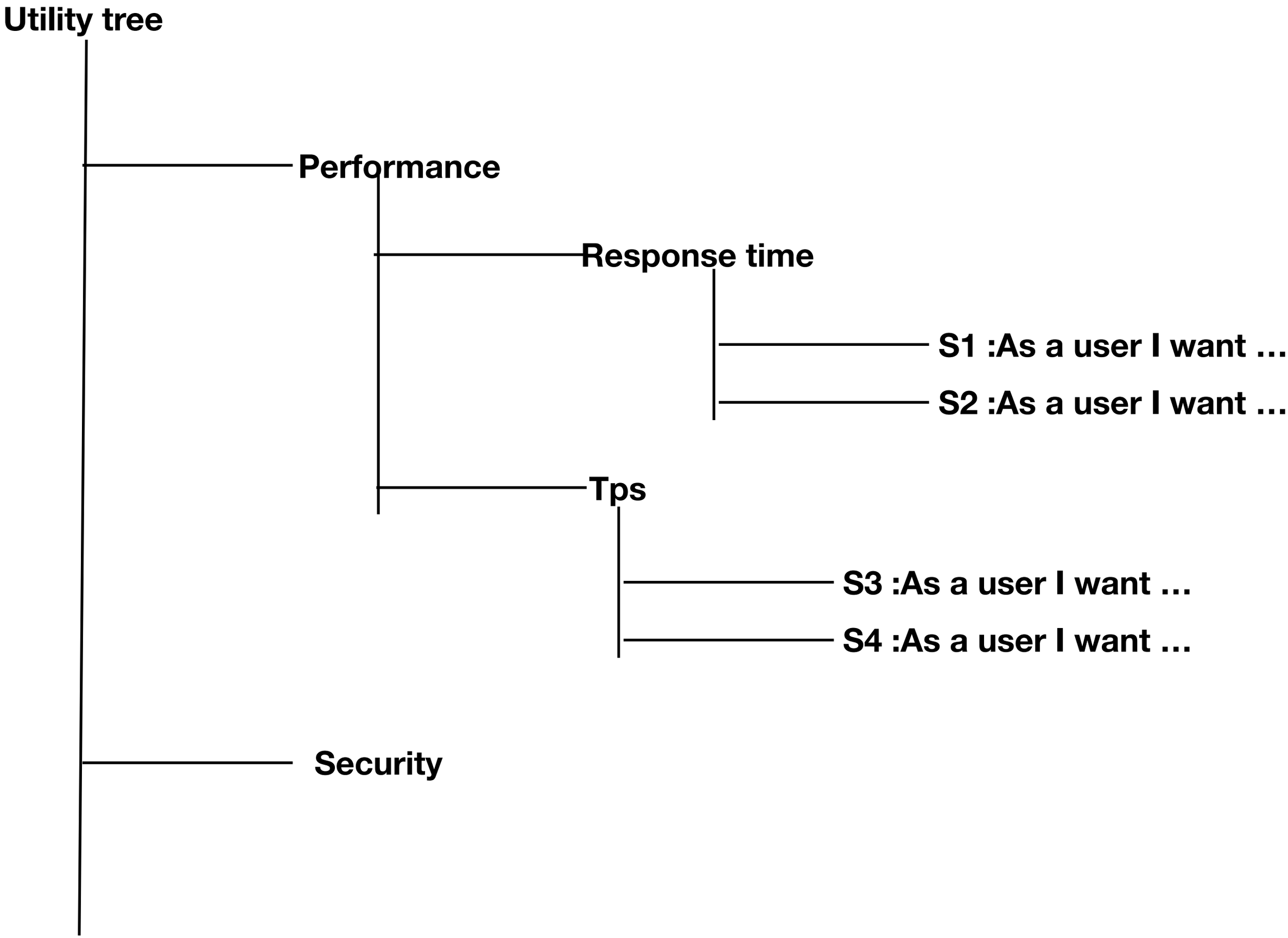
- Analyze Approach and Scenarios
  SC#1 -> A1, A7
  tradeoff:
  SC#2 -> A3
  tradeoff:
  SC#3 -> ?

Review the eval

Evaluator + arch team + all stake holders

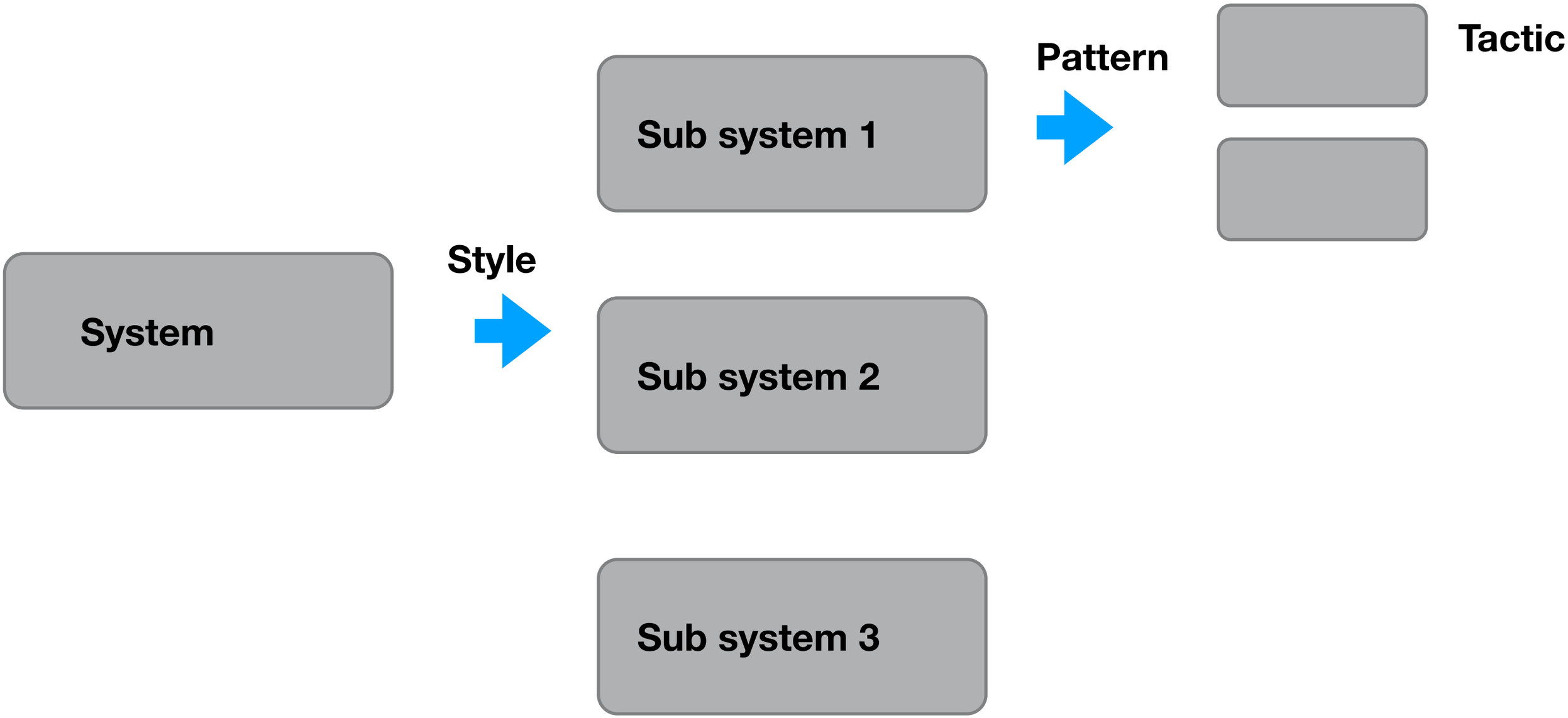
- Brain storm for scenarios (s5,s6,s7)

- 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

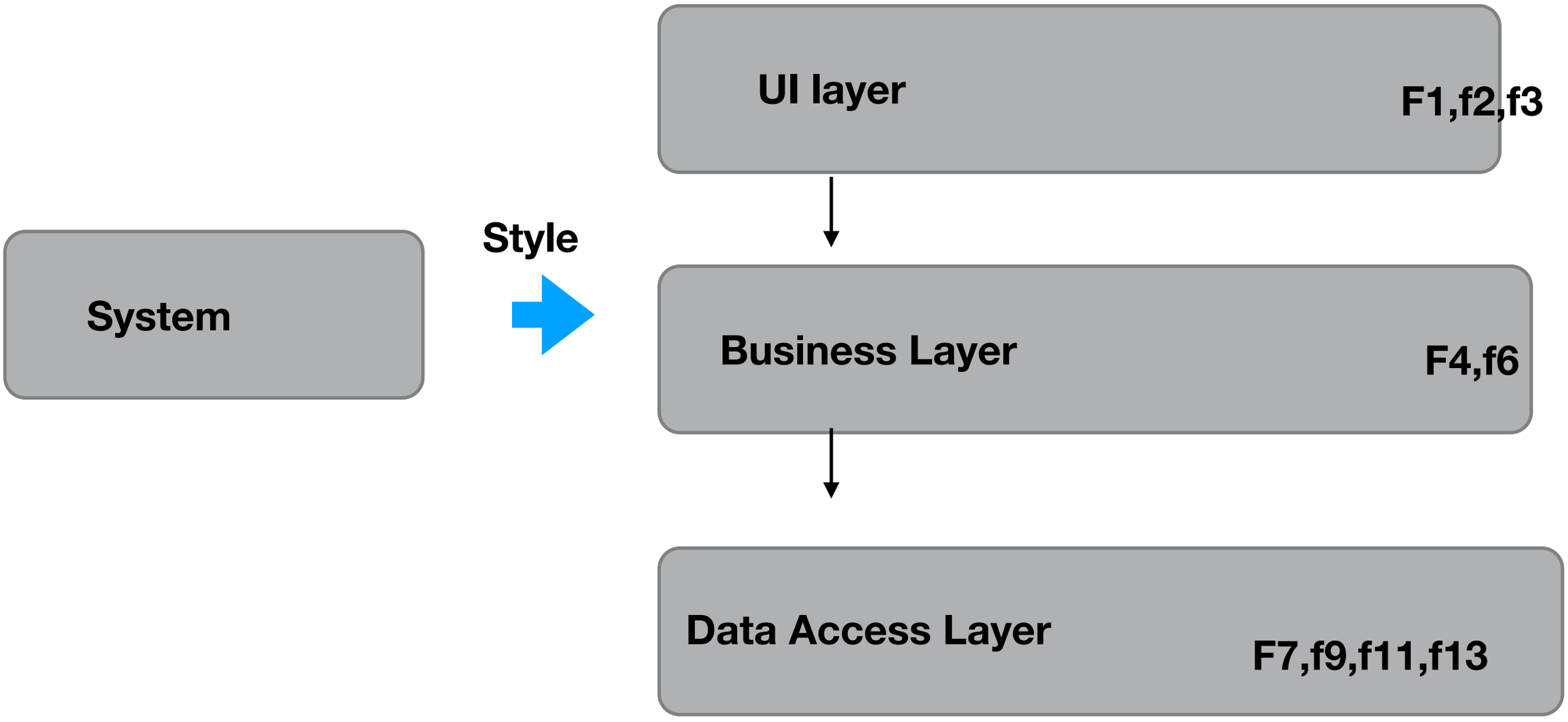
Decompose system



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

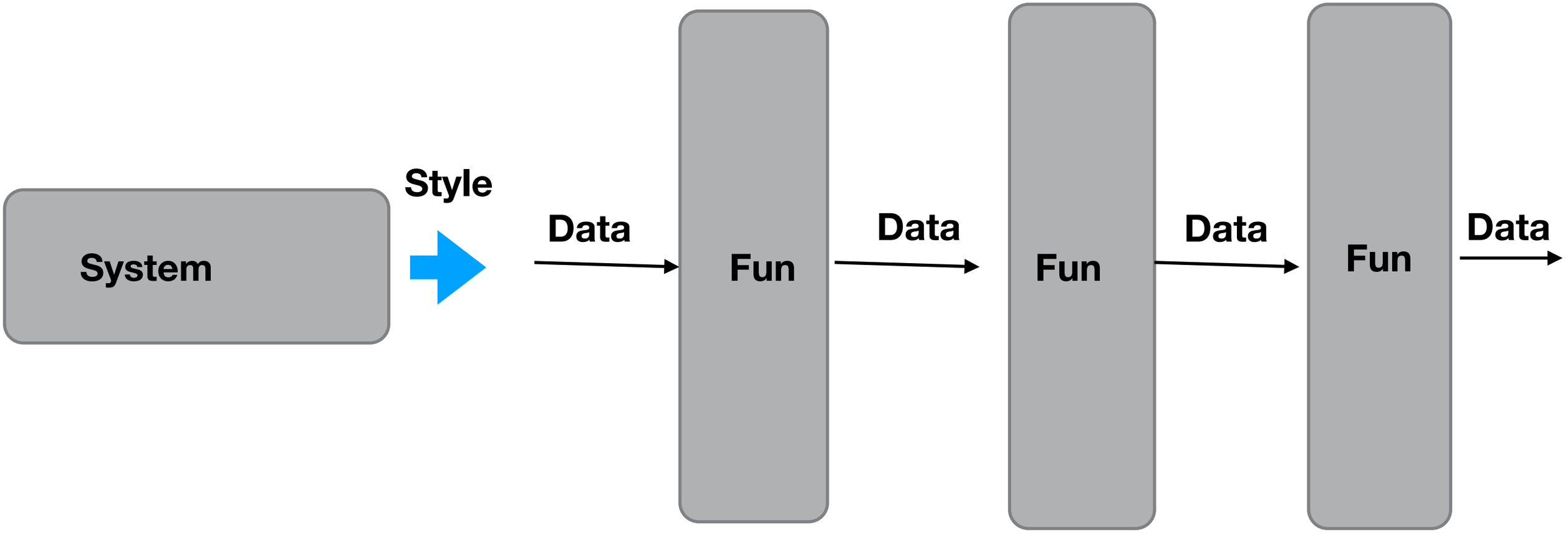
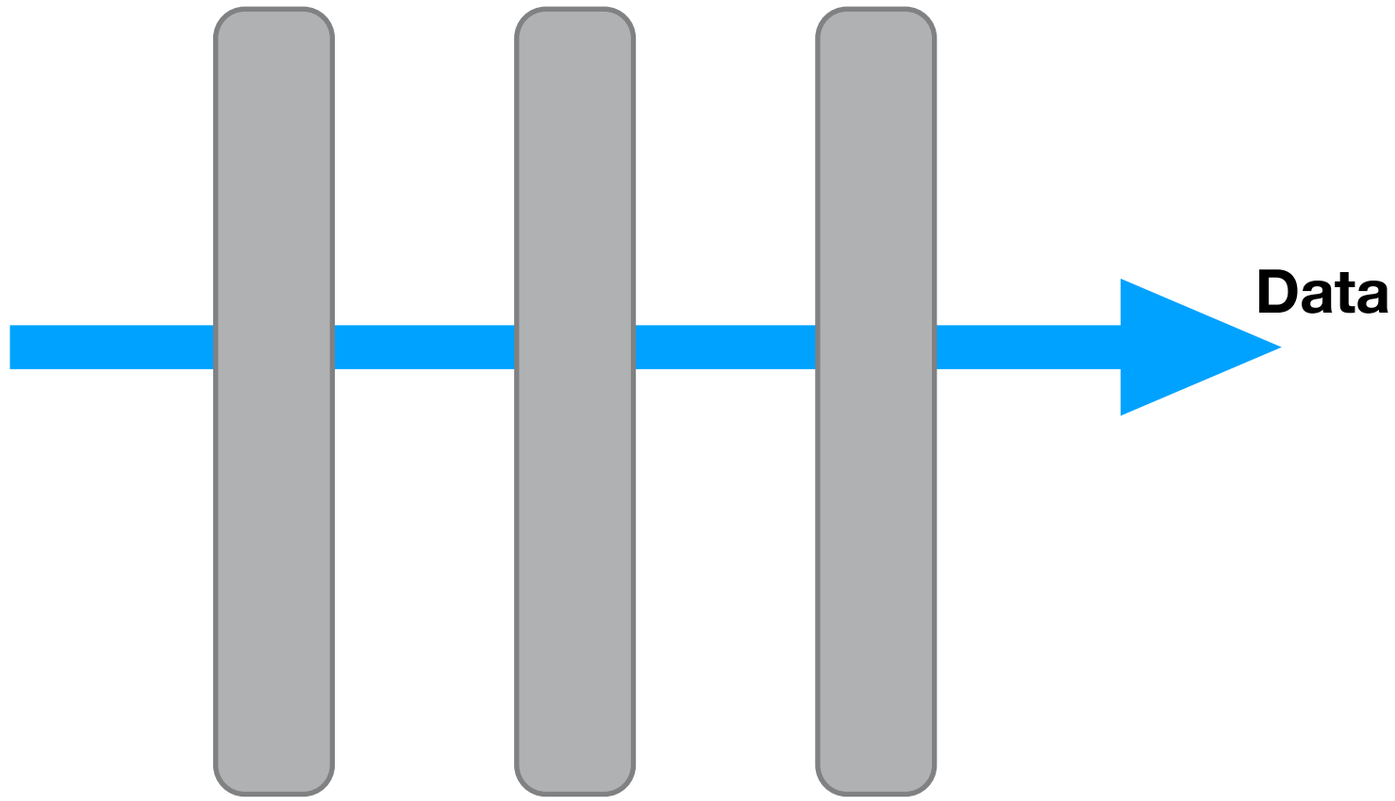


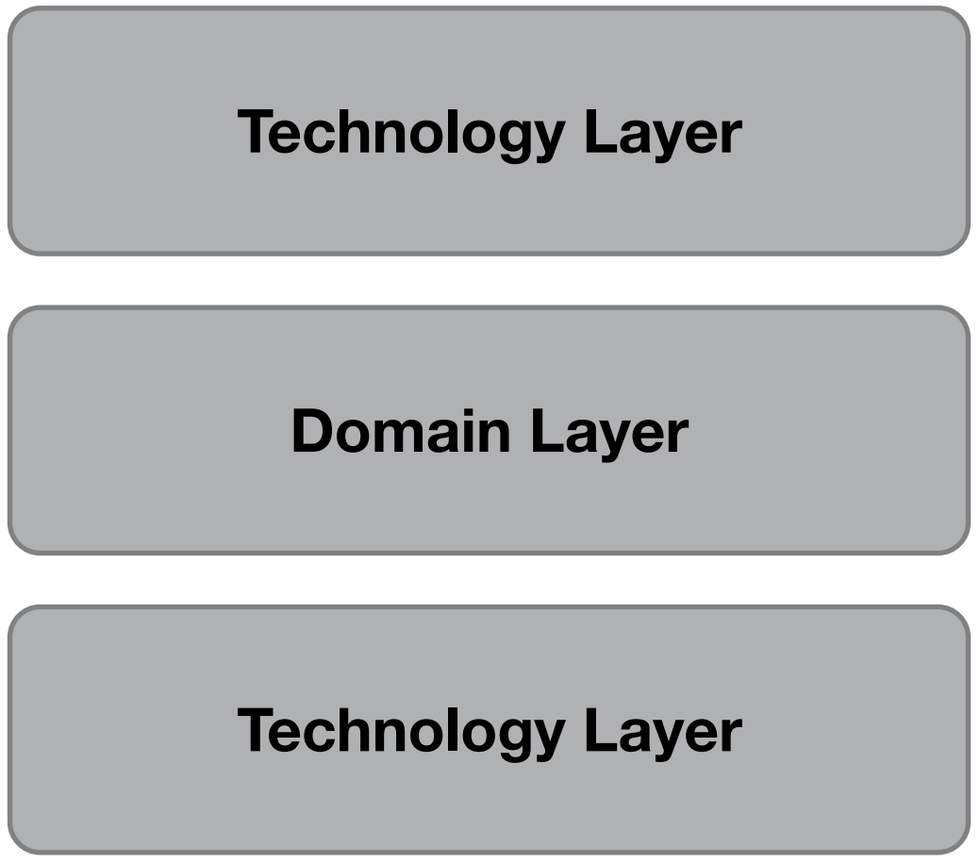
Image processing

- Extract Feature
- OCR
- Mask
-



Seperation concerns

- Separate Technology logic from domain logic



Architectural Pattern

- MVC
- Broker
-

Bid of the day

Bid app

Quality Requirement

Quality attributes

1. Performance
    1. CPU
    2. Memory
    3. Disk
    4. Network
2. Reliability (trust)
3. Availability
4. Maintainability
5. Usability
6. Security (trust)
7. Scalability(volume)
    1. Data
    2. Compute
    3. I/O
8. Robustness (Rugud)
9. Portability
10. Interoperability

Measure (scale)

1. Response time
2. tps
3. Resource utilization
4. Downtime
5. Uptime
6. Code coverage
7. No of clicks
8. Probability
9. ...

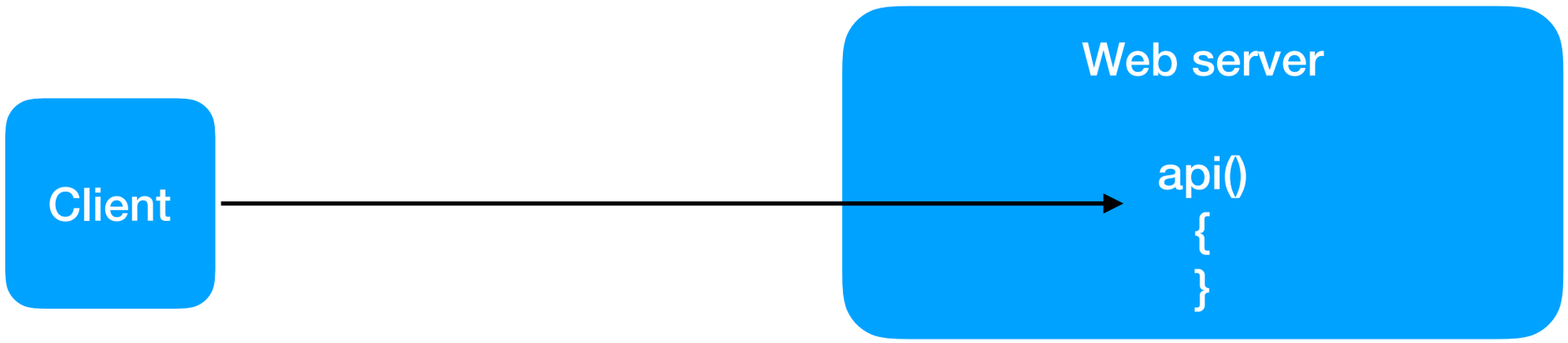
Tactic / Style

1. Caching
2. Chunking
3. Sharding
4. Load balancing
5. Lazy loading
6. Documentation
7. Unit test
8. Monitoring and Alerts
9. Coding guidelines
10. Low Coupling
11. ACID
12. Message Queue
13. Stateless
14. Input validation
15. Scalability cube
16. Micro service

Quality attributes Model

SEI
McCal
Bohem
IEEE
#ISO

< 1 sec
< 3 sec



<<architectural Requirements>>

1. Quality requirements
source
stimulus
artifact
environment
response
measure

2. Constraints (rule)

Collect

Key Functional Requirements

Understand

“Address System Quality”

Choose

Approach

Knows

Technology
Arch patterns
Anti patterns
Principles (rule)
Reference architecture

<<align>>

Enterprise Architecture

 # TOGAF
 # zachman
 # DODAF

Solution/product Architecture

<<process quality>>

Business/ Domain Architecture

 # bpel
 # bmpn

<<s/w quality>>

Application Architecture

 # uml

<<s/w quality specialist>>
Vertical Architecture

Data/ cloud/security/ infra/ UX/ …

Architecture

Quality Engineering

Quality Tuning

Performance Engineering

Performance tuning

Threat Modeling

Ethical hacking

Anti pattern

 # Alice in wonderland

Performance

A +b; <- 3 cpu cycles
fun(); <— 5 cpu cycles

Create thread <— 200,000 cpu cycles

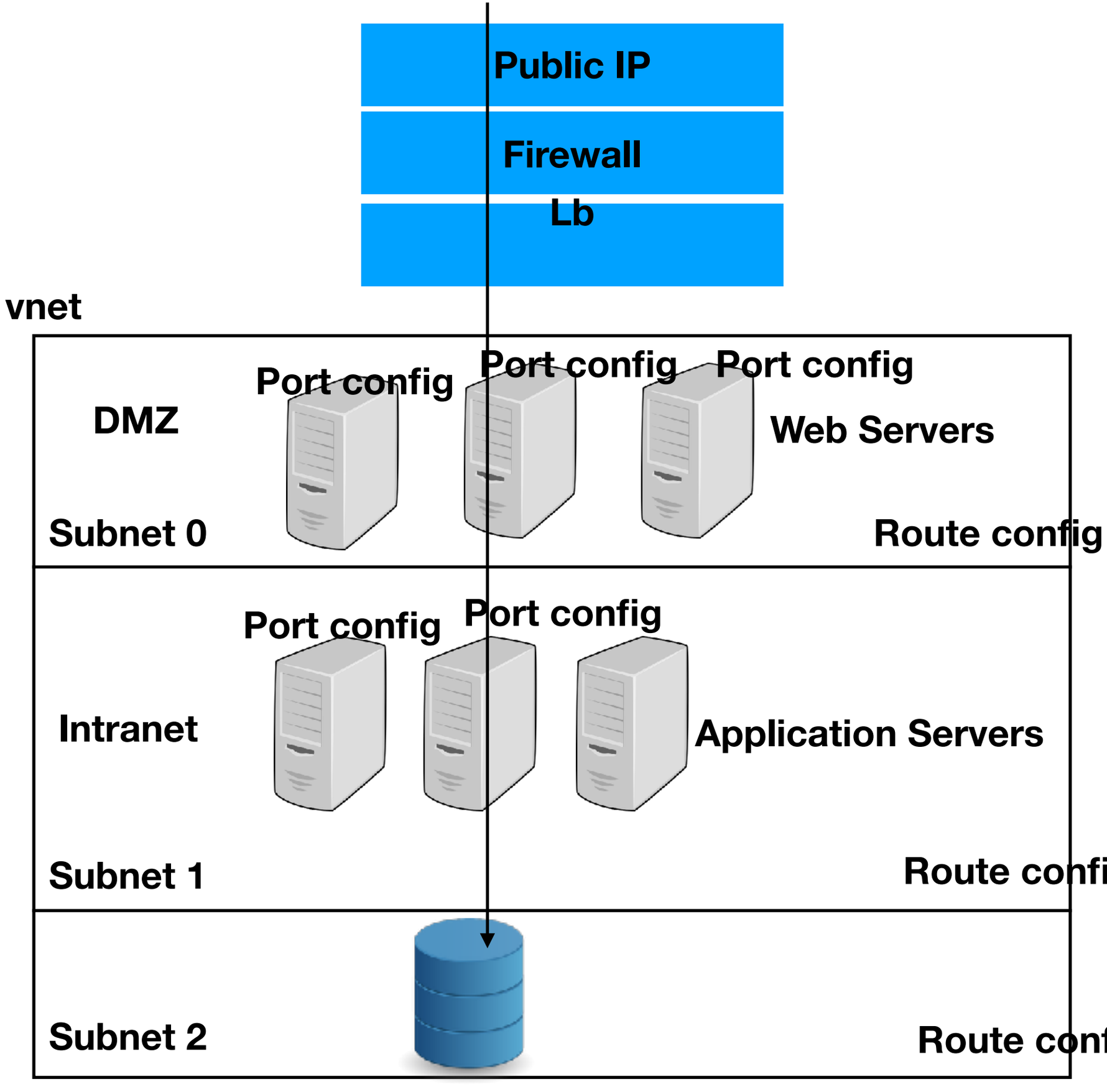
Db call <— 45,00,000 cpu cycles
Remote call (rest) <— 30,00,000 cpu cycles

“COPY”

“Reference Architecture”

- Data processing
- Eda
- Layered
- DDD
- Security
- Big data
-

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			

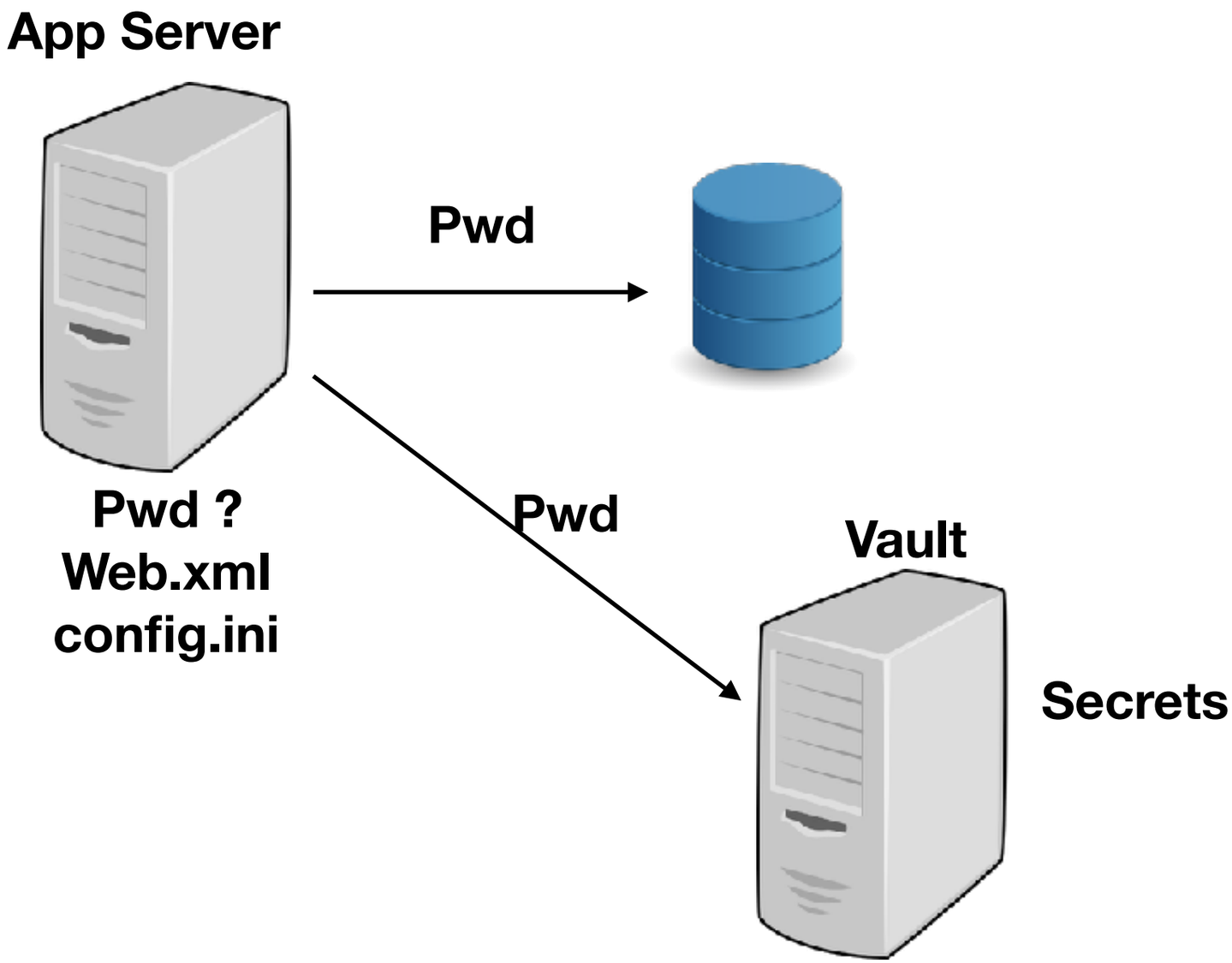

Public IP
Firewall
Lb
vnet
Port config
Port config
Port config
DMZ
Web Servers
Subnet 0
Route config
Port config
Port config
Intranet
Application Servers
Subnet 1
Route config
Subnet 2
Route config

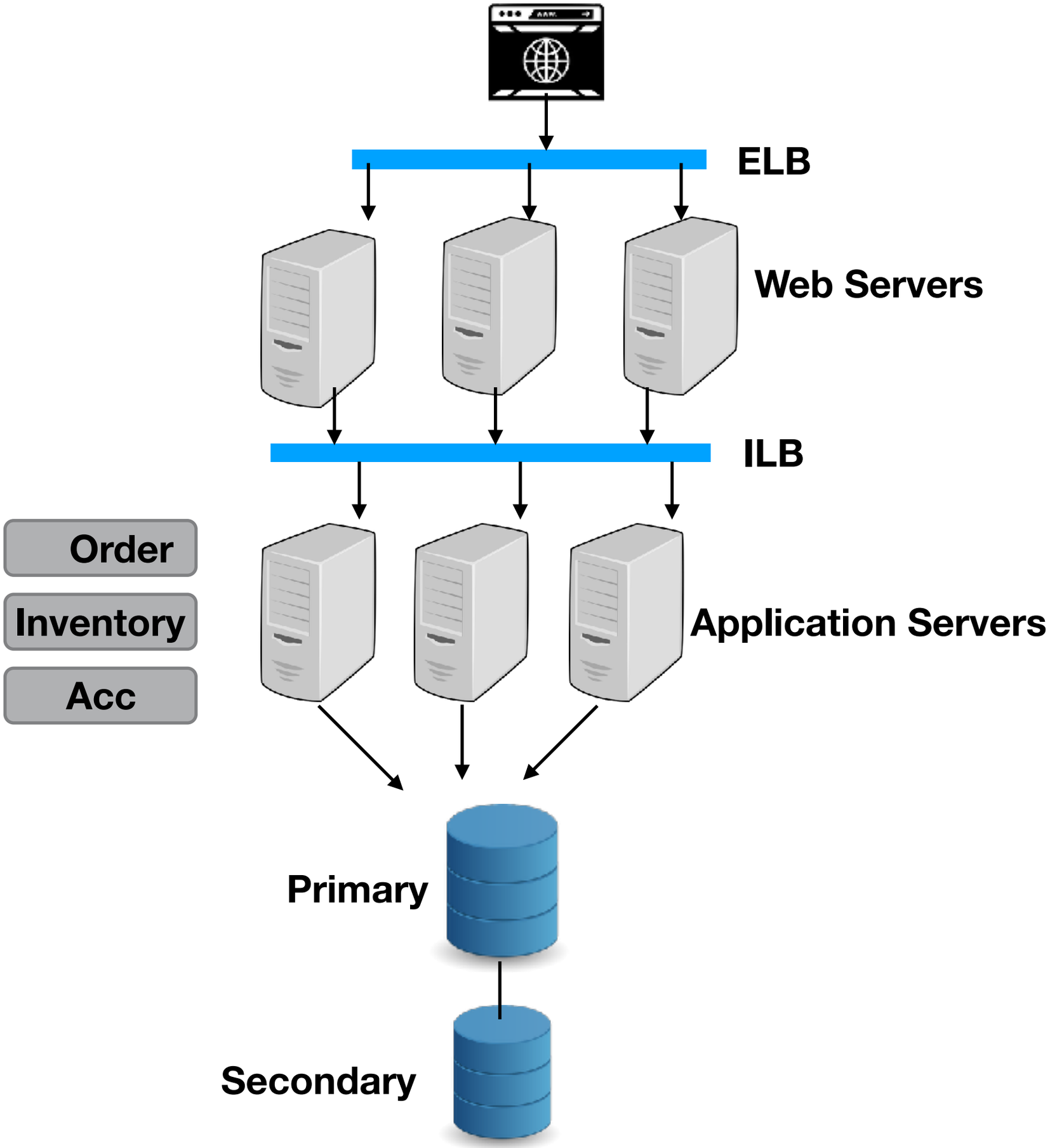
Application Security

- 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)

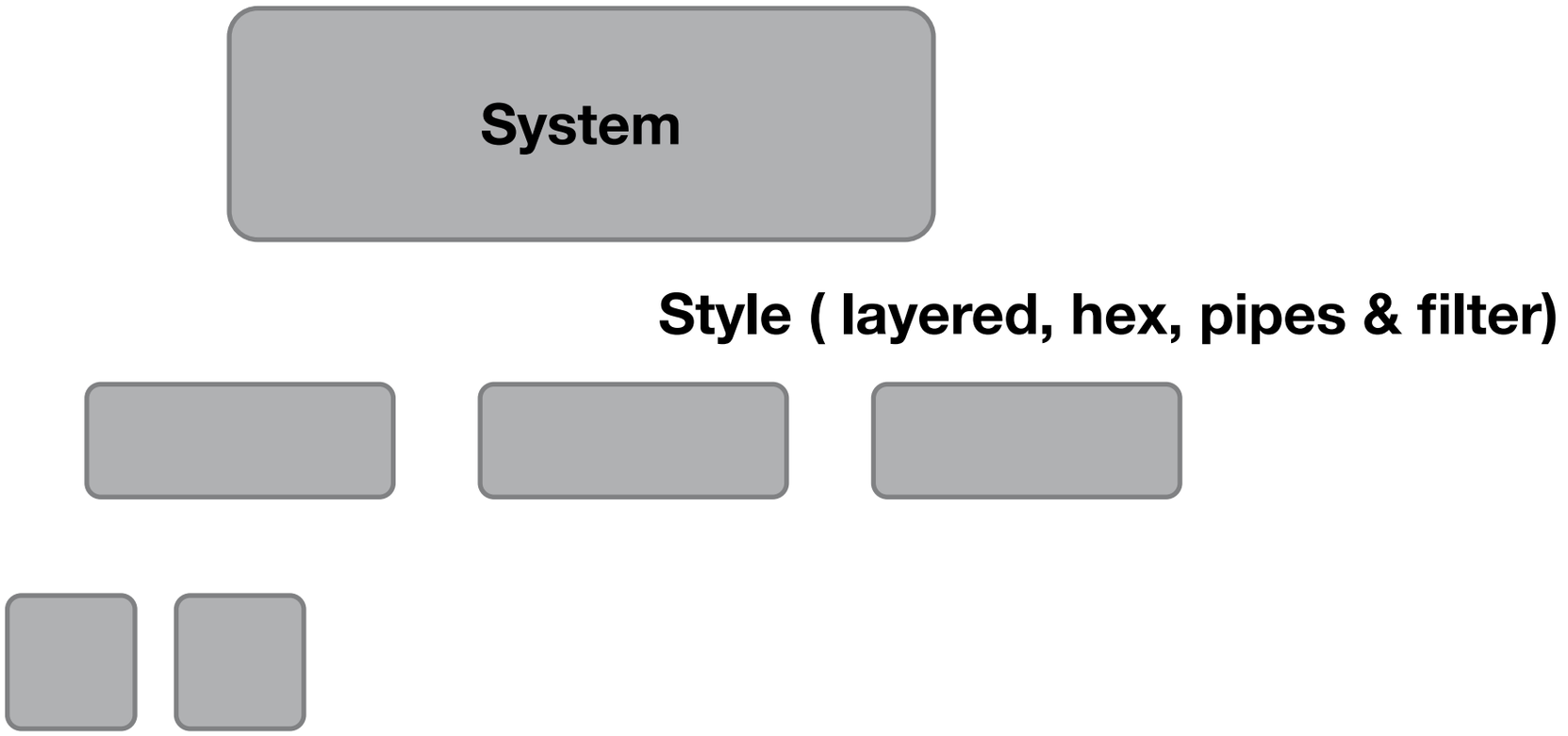
Containerization layer (Docker)

Unix Guest Os

Kubernetes (Master)

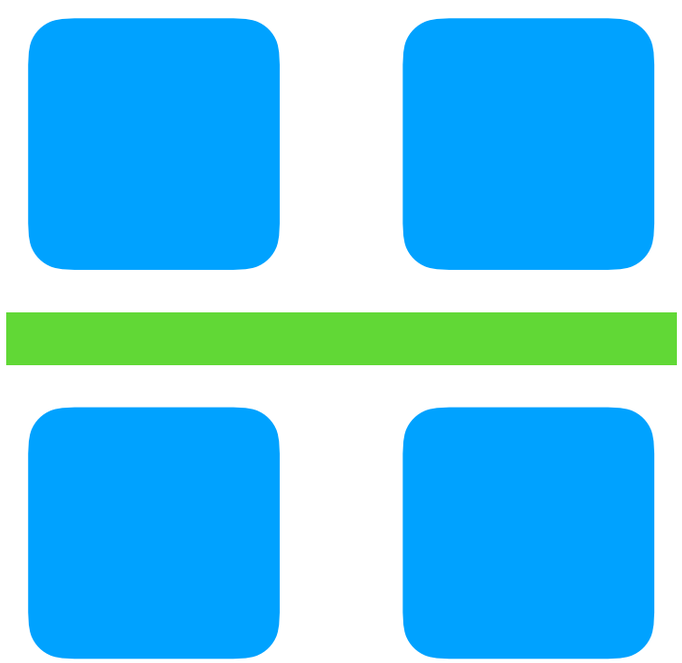
Kubernetis Swarm

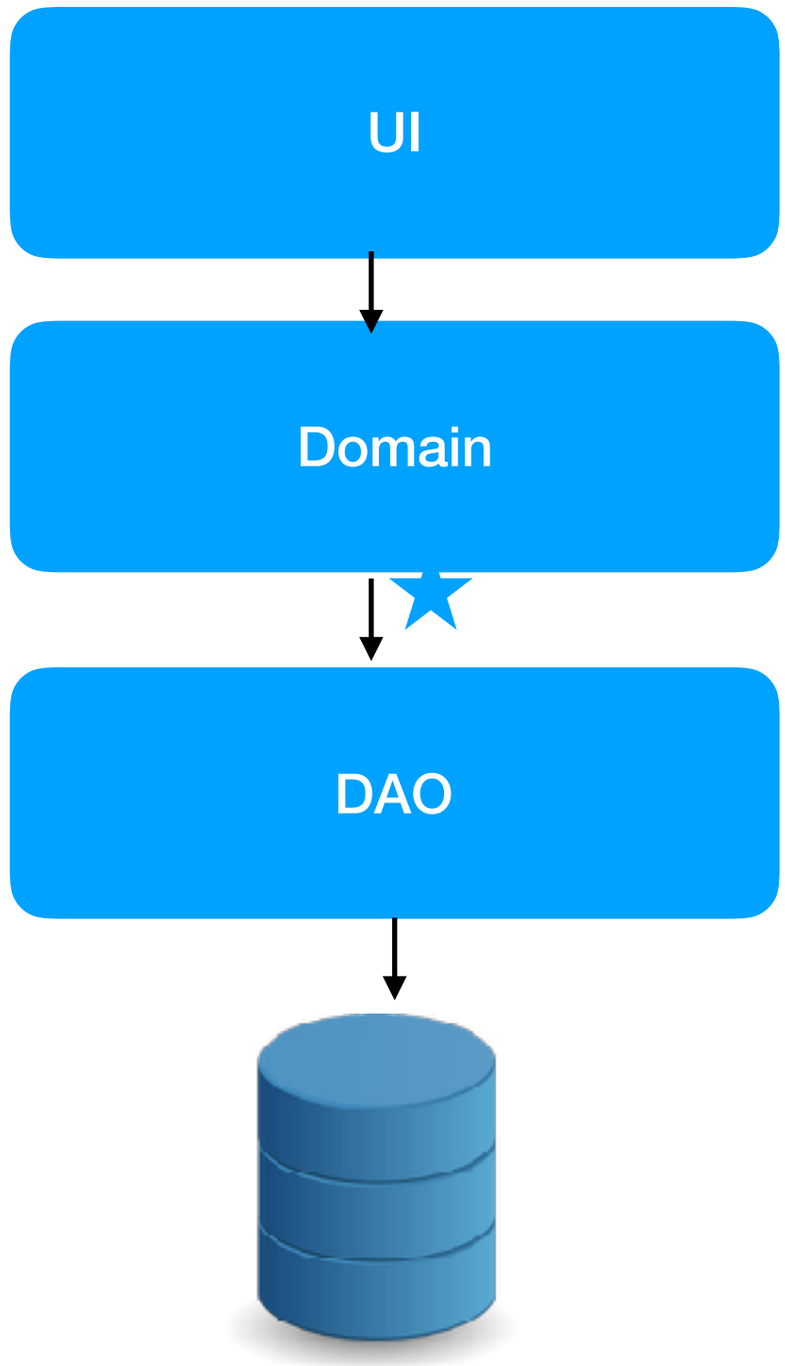
decompose system



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

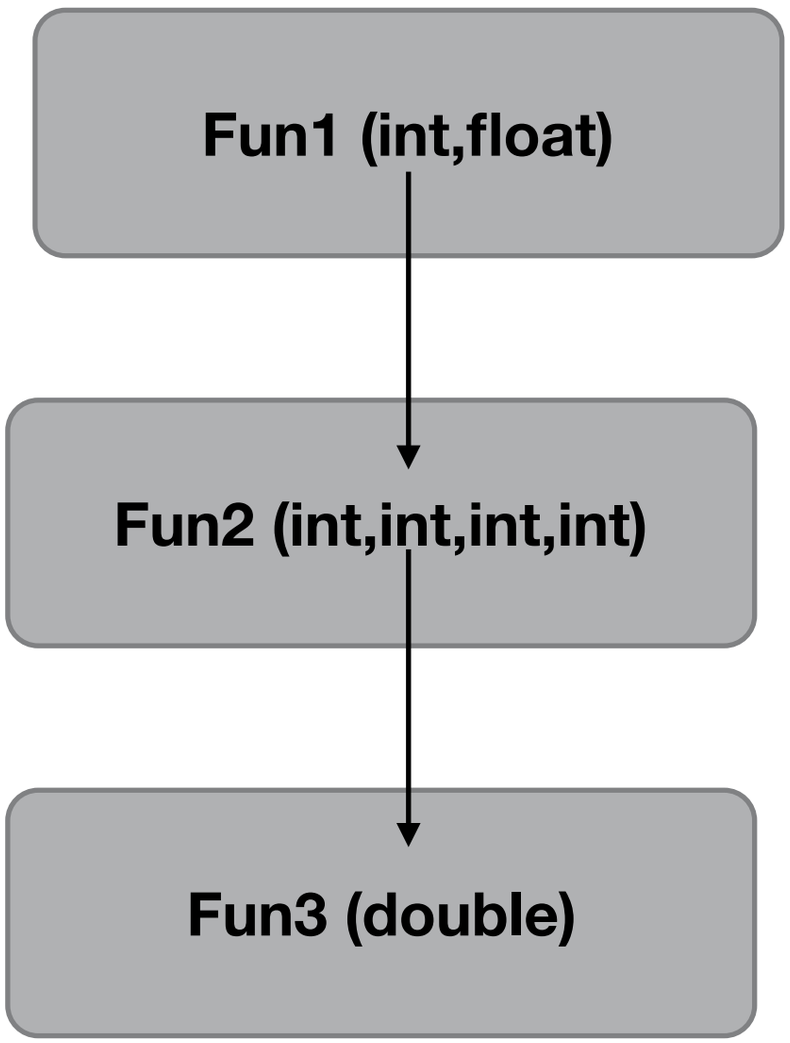
Domain Logic

Proxy Technology

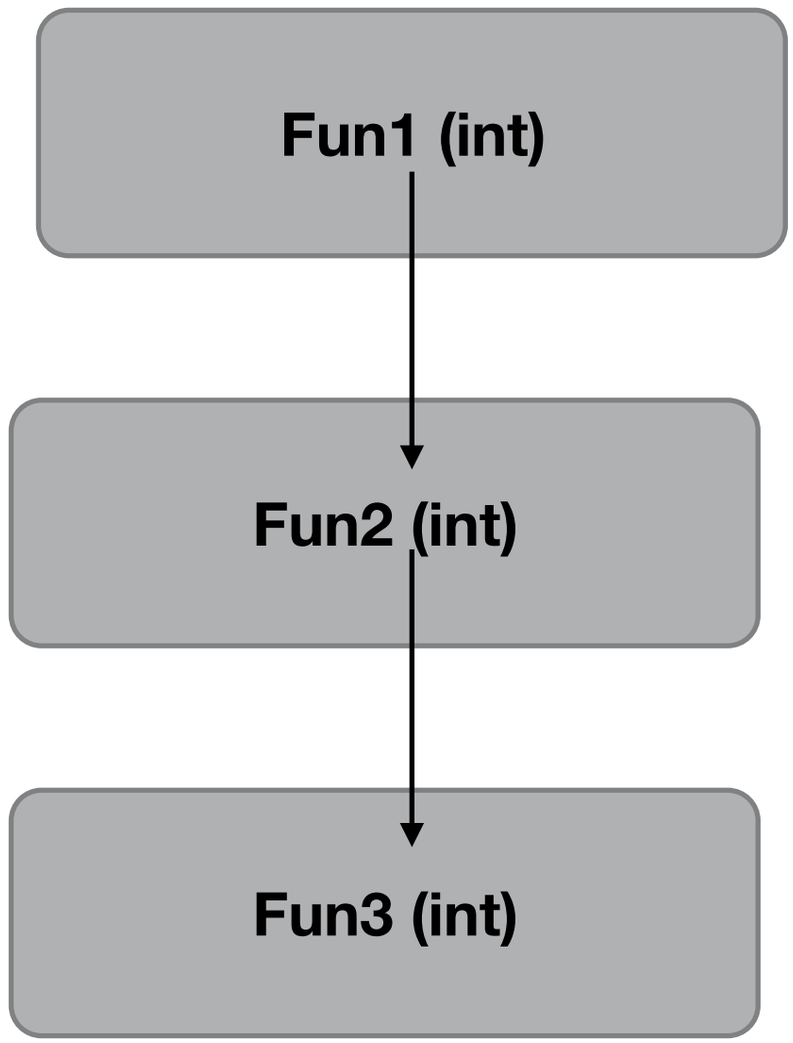
Persistence Technology

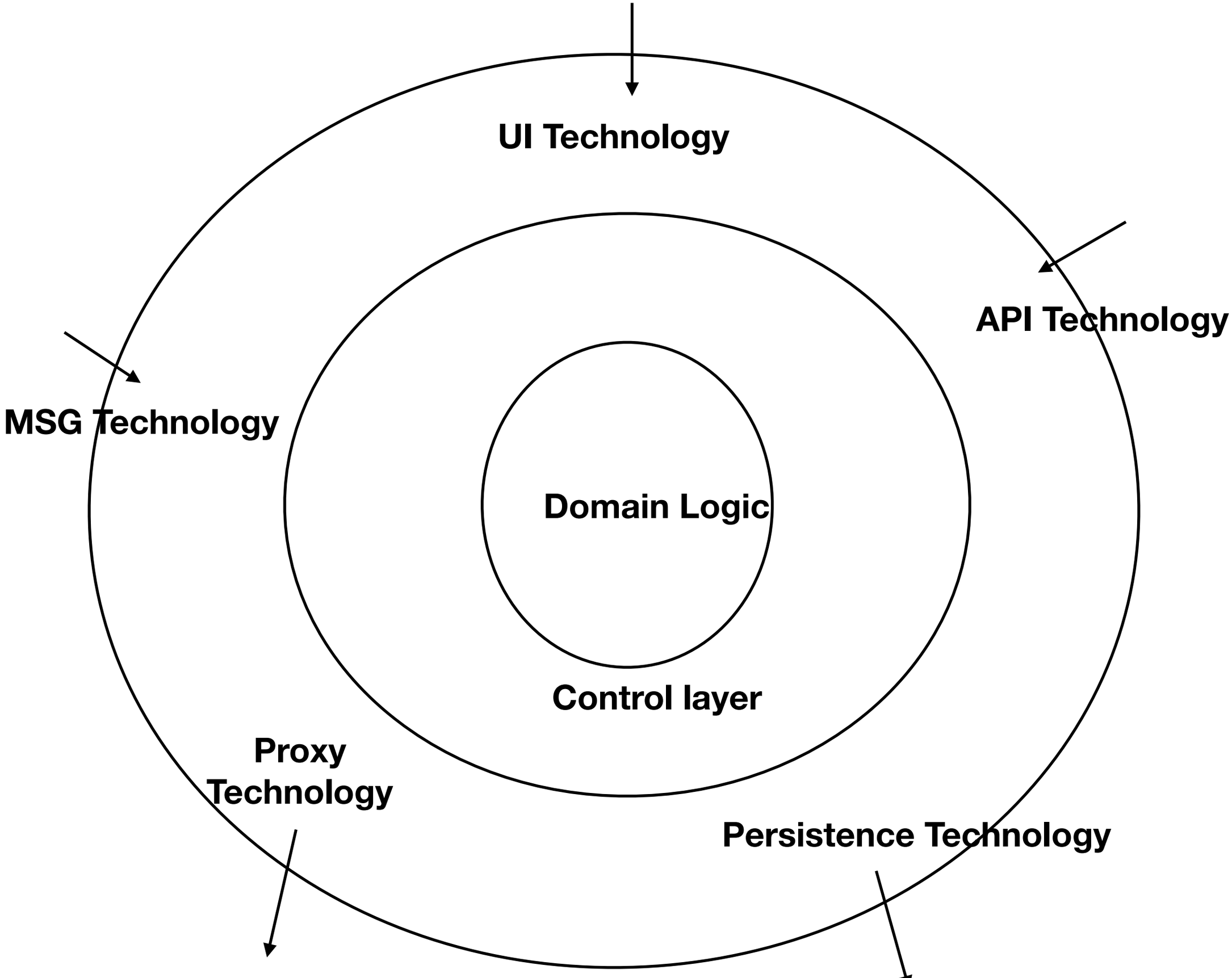
Exit

Layered



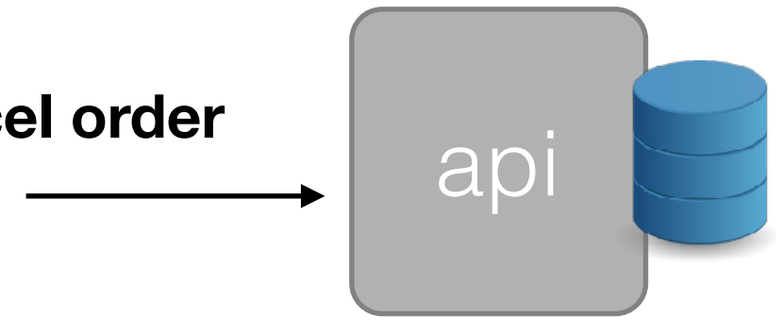
Pipes and filter

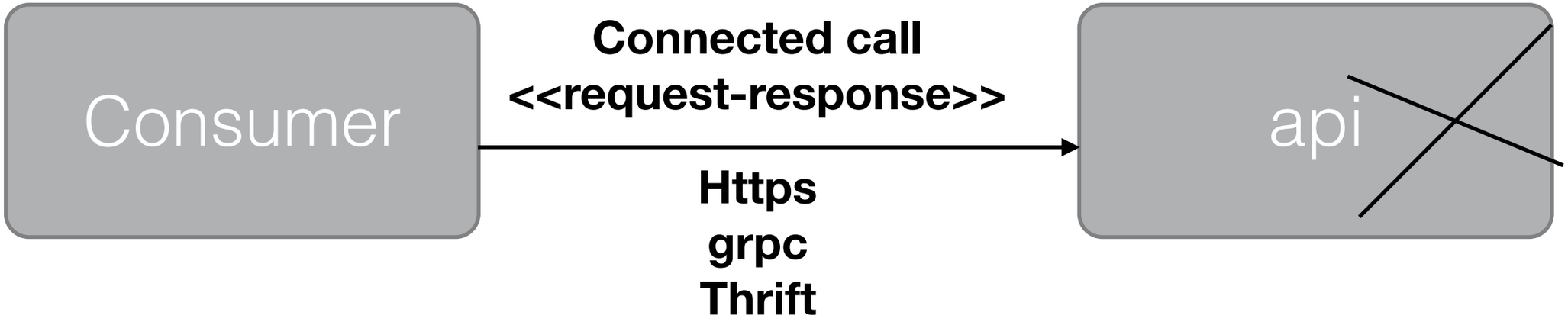


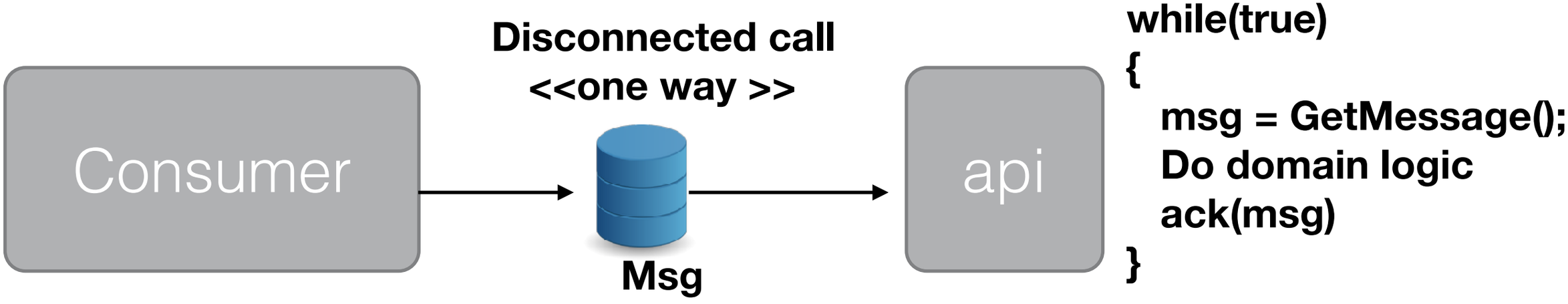

UI Technology
API Technology
MSG Technology
Domain Logic
Control layer
Proxy
Technology
Persistence Technology

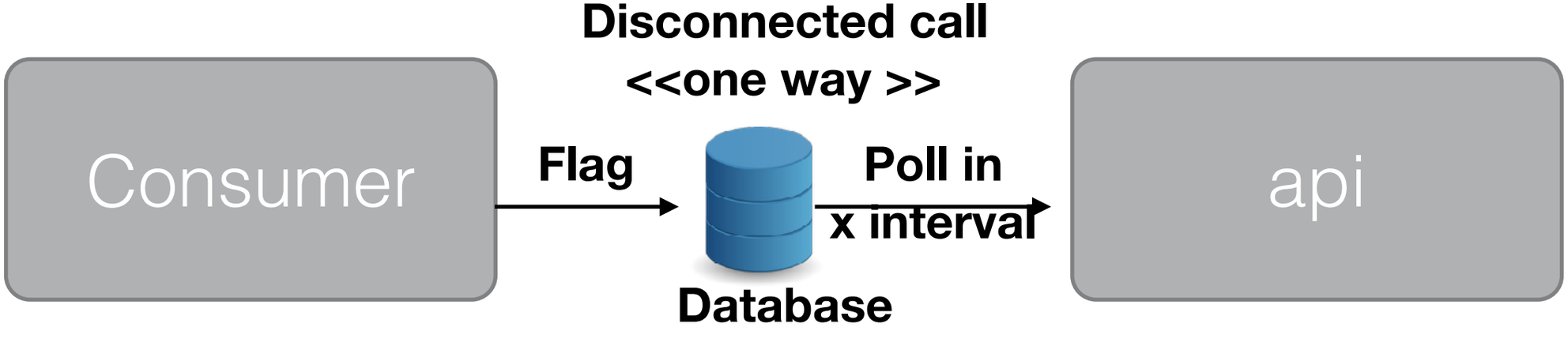
Create order

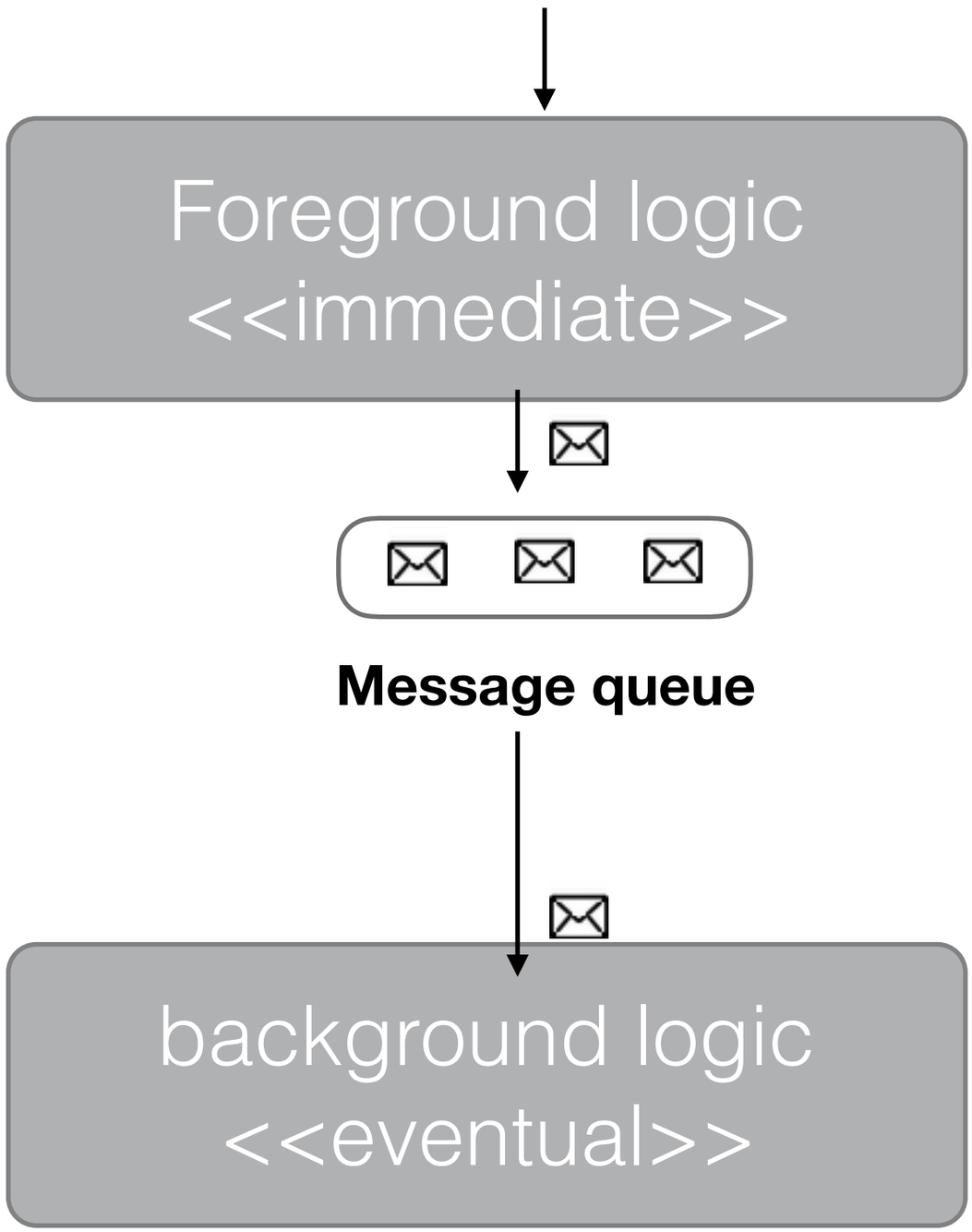
Cancel order



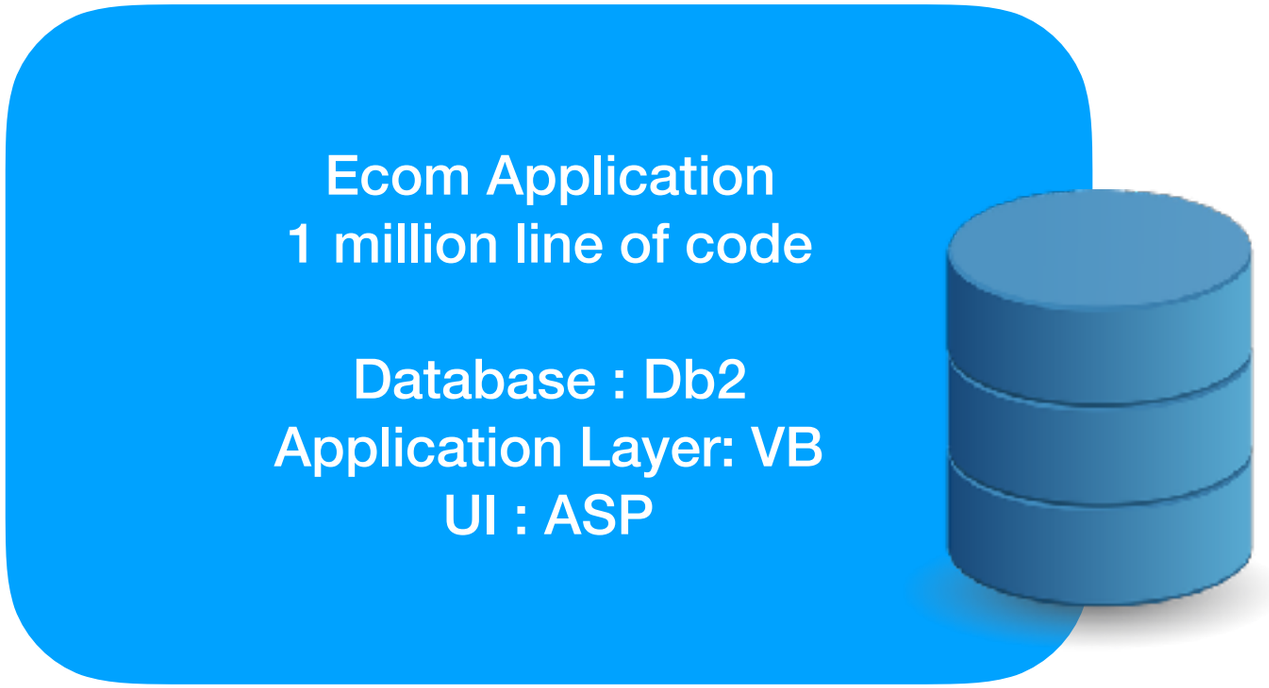

Consumer
Connected call
<<request-response>>
Https
grpc
Thrift
api


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

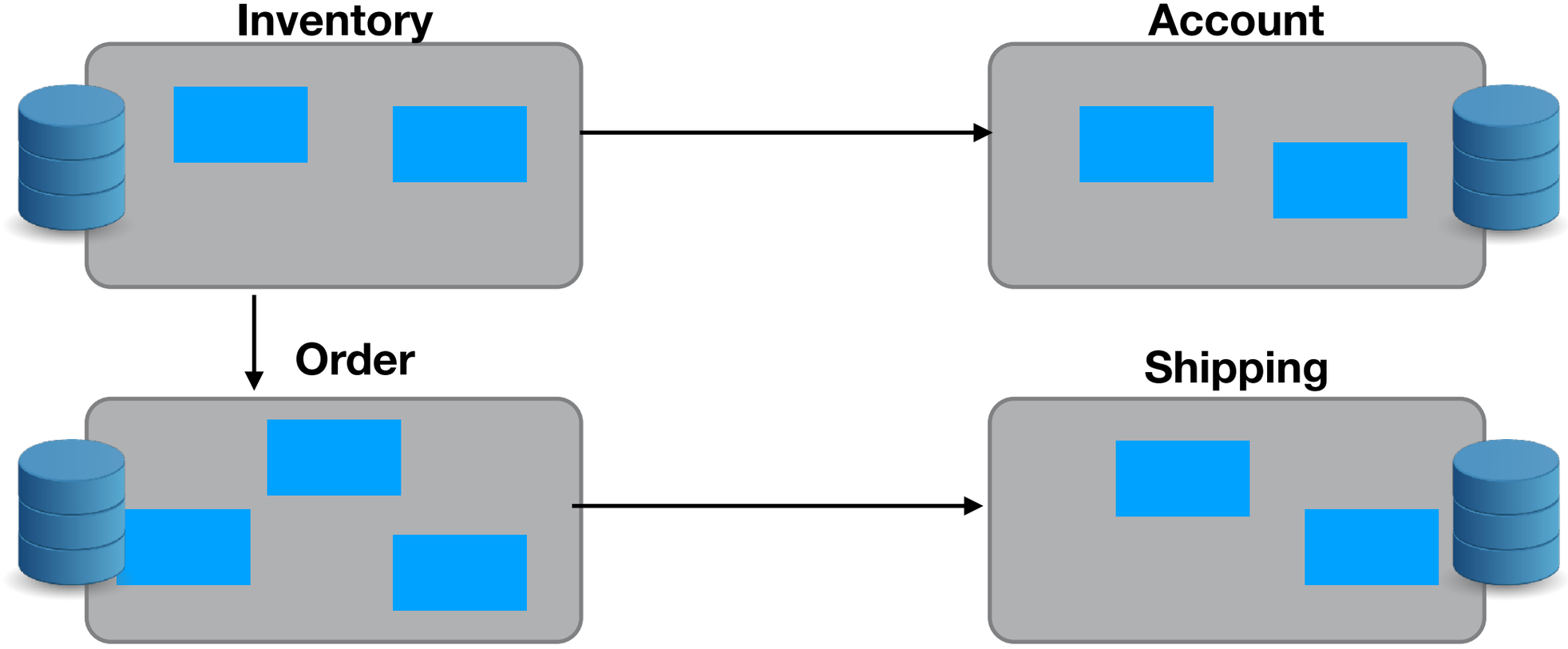

Consumer
Disconnected call
<<one way >>
Flag
Database
Poll in
x interval
api

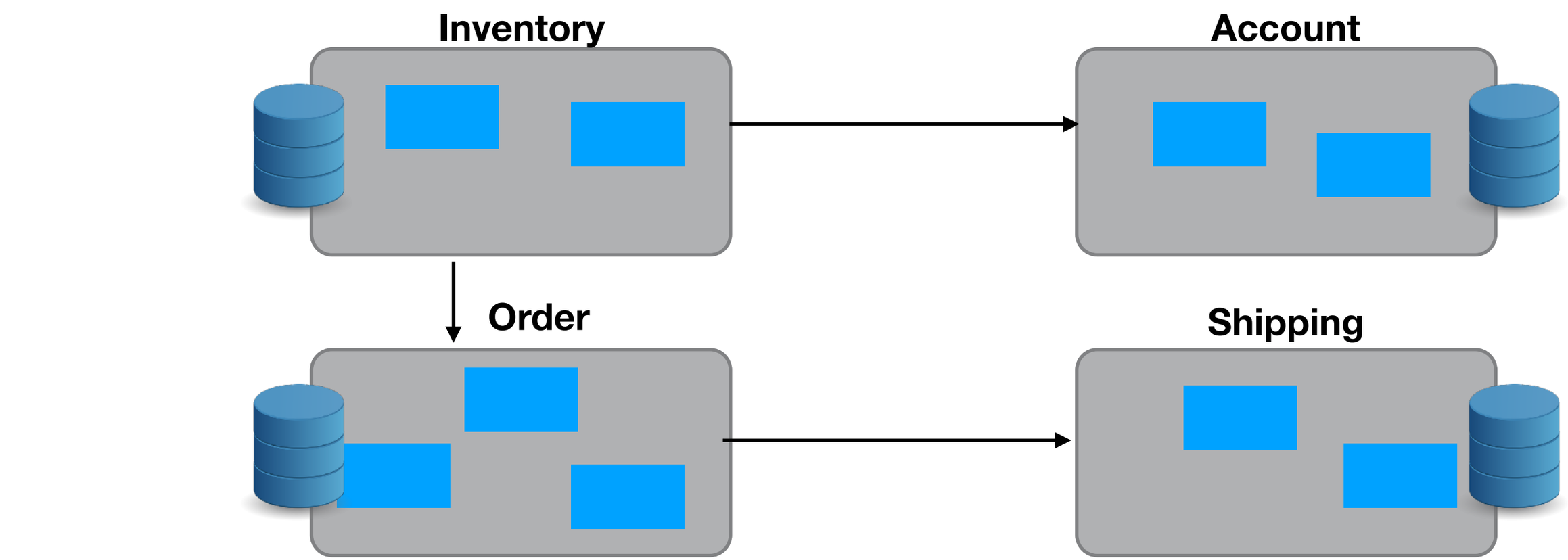

Foreground logic
<<immediate>>
Message queue
background logic
<<eventual>>

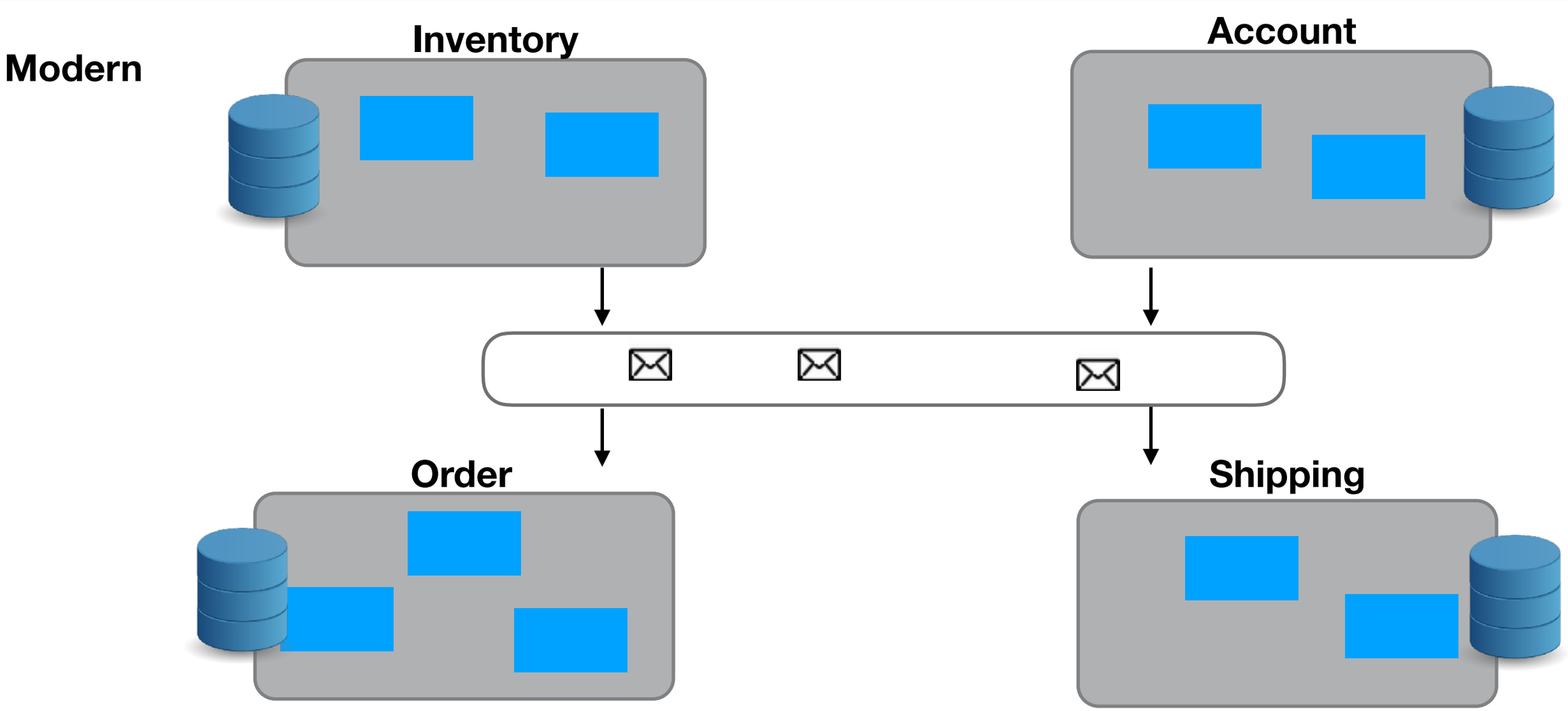
Legacy

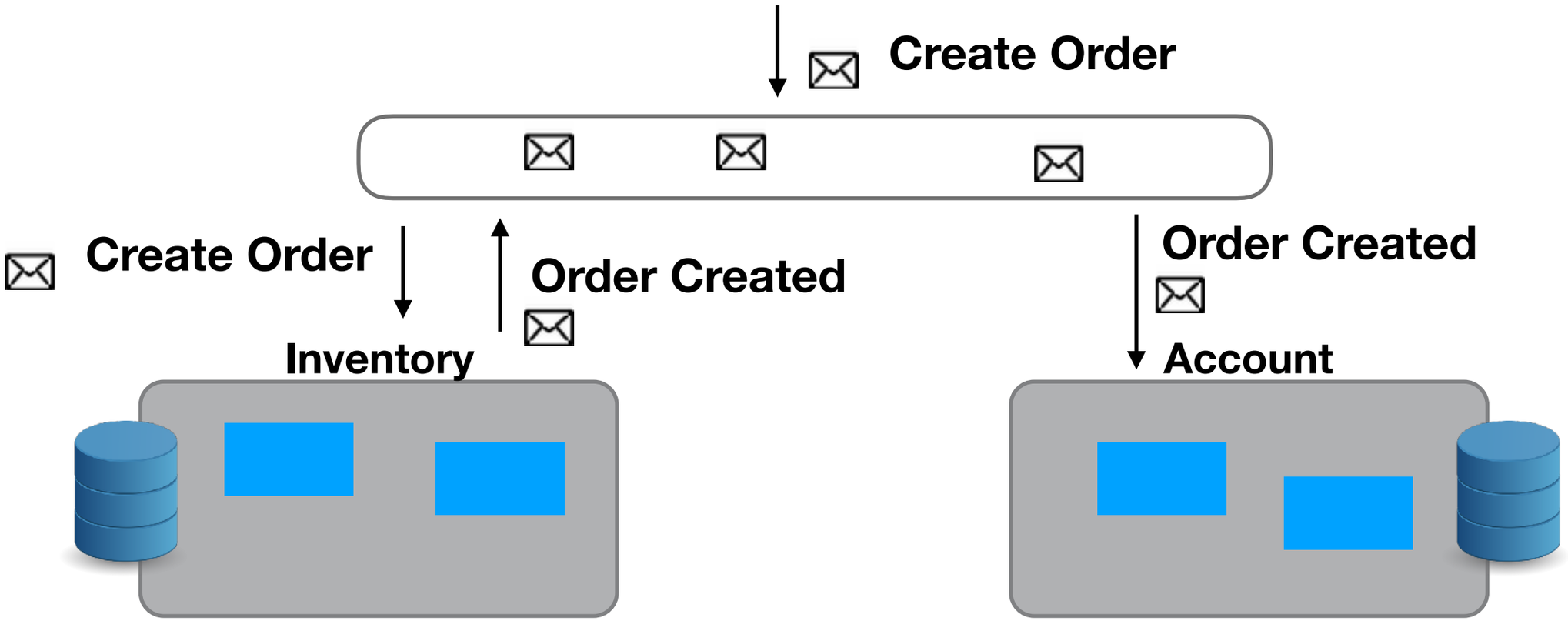


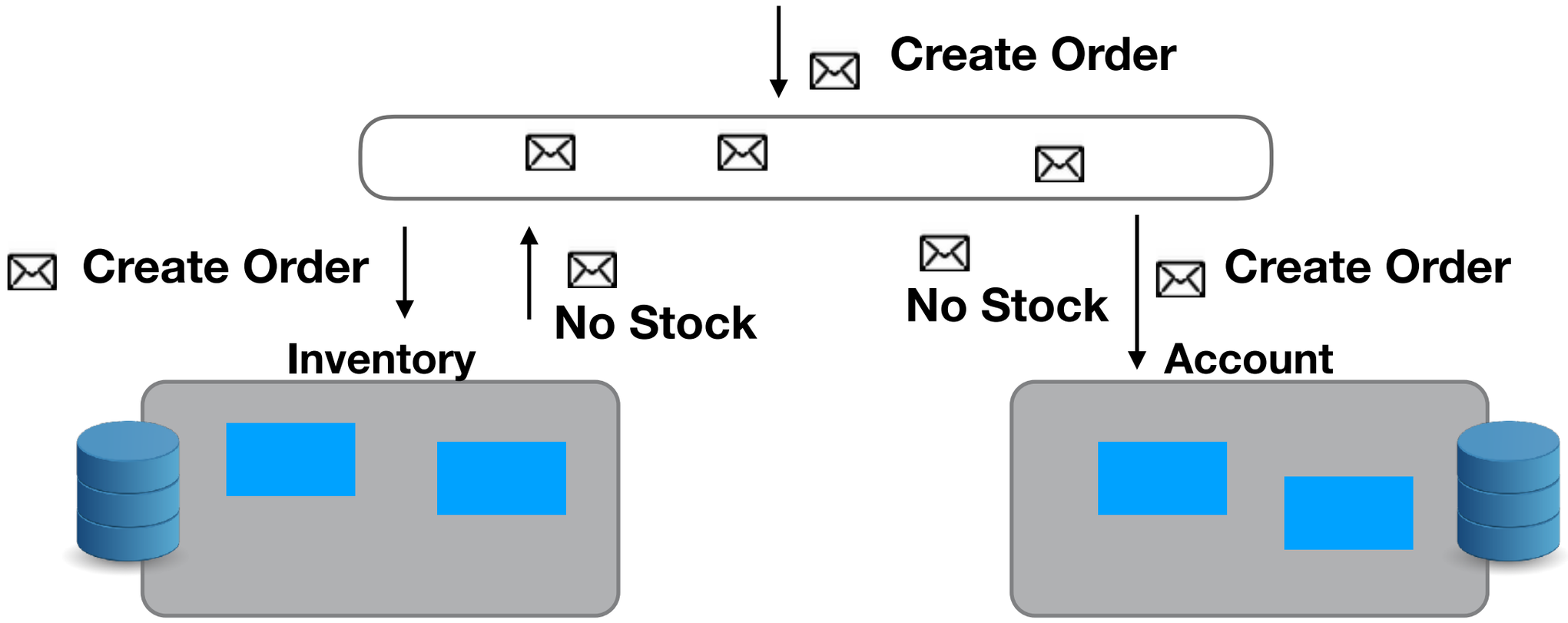
Modern




Inventory
Account
Order
Shipping


Modern
Inventory
Account
Order
Shipping

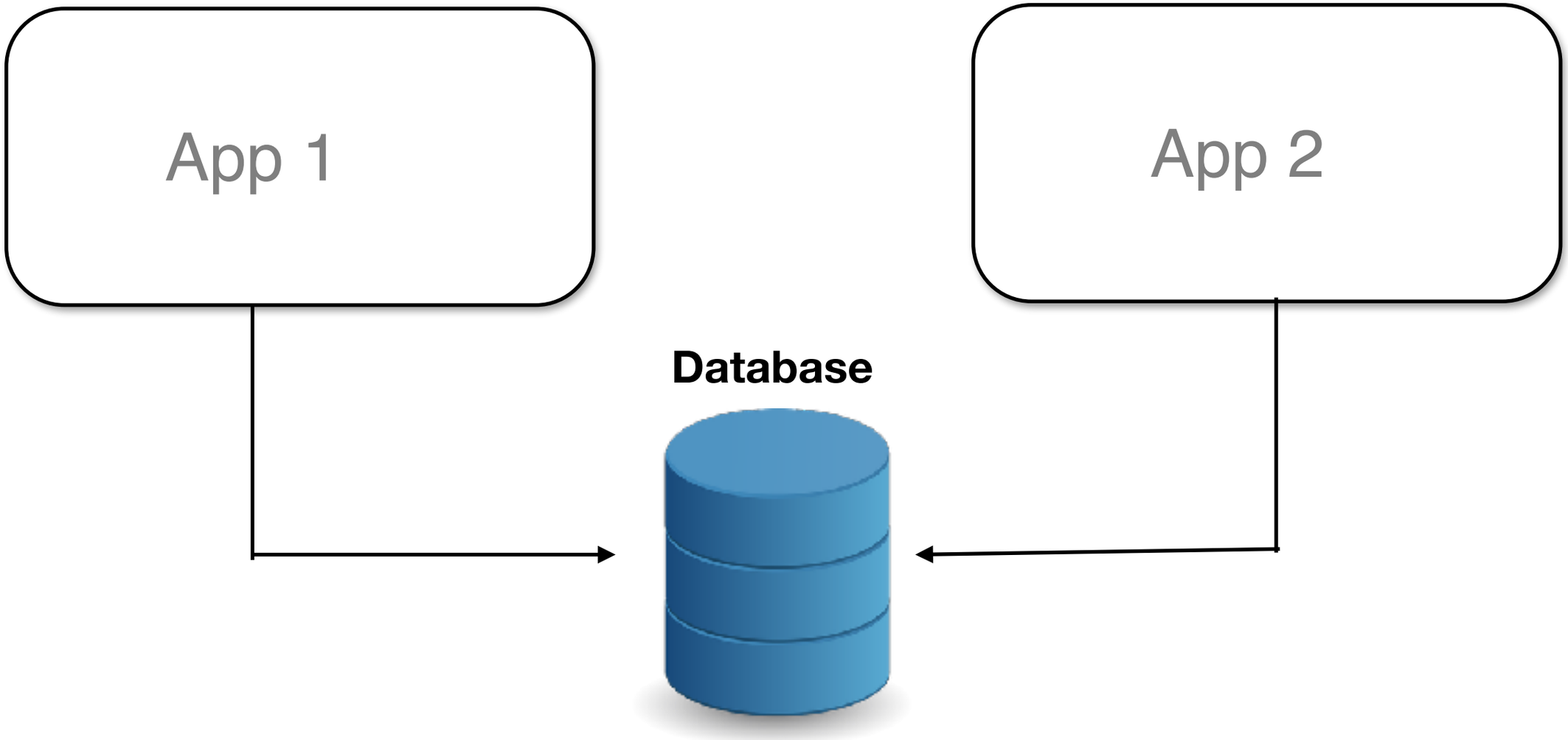

Create Order
Create Order
Order Created
Order Created
Inventory
Account

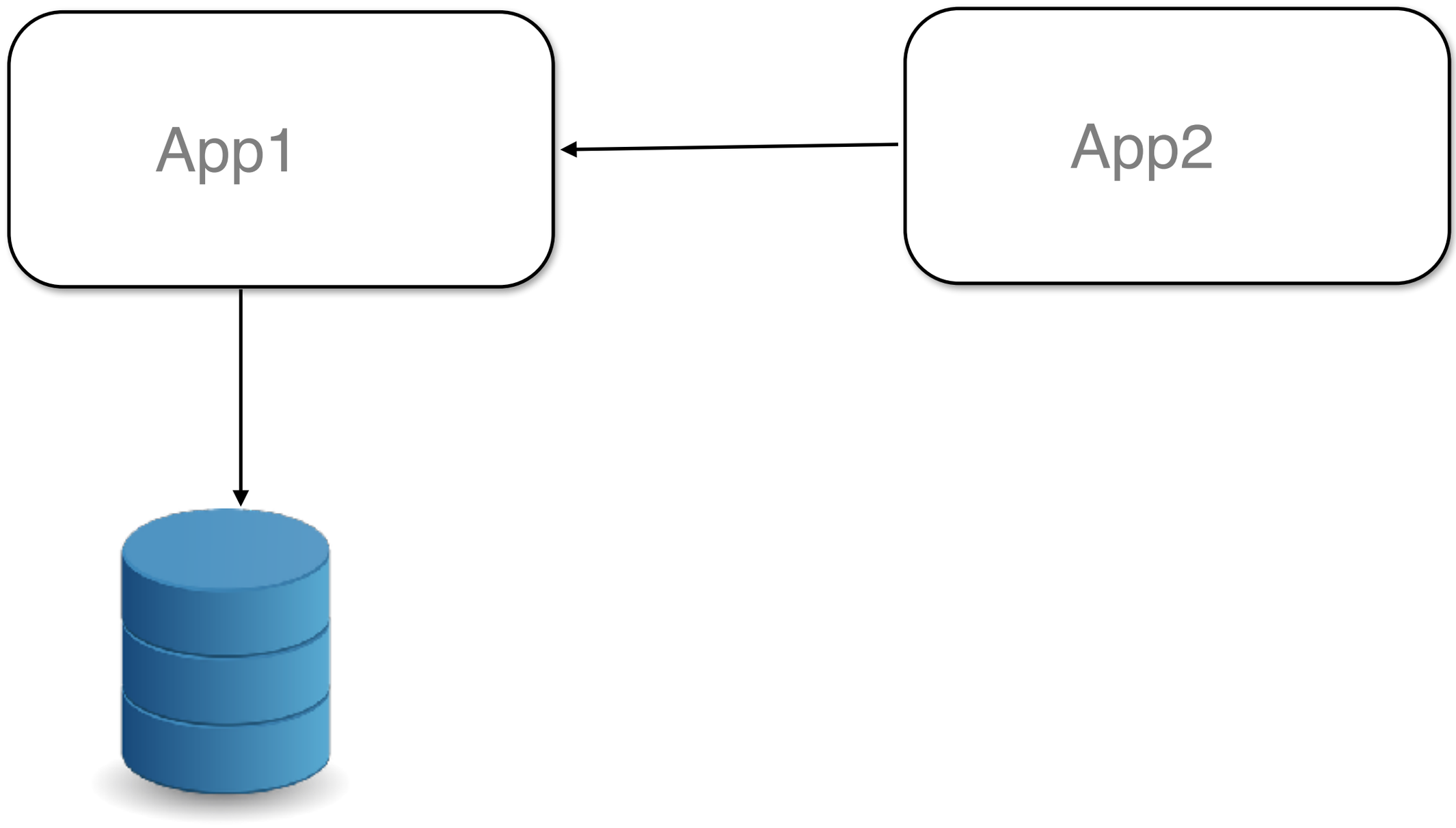


Compensation / undo

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

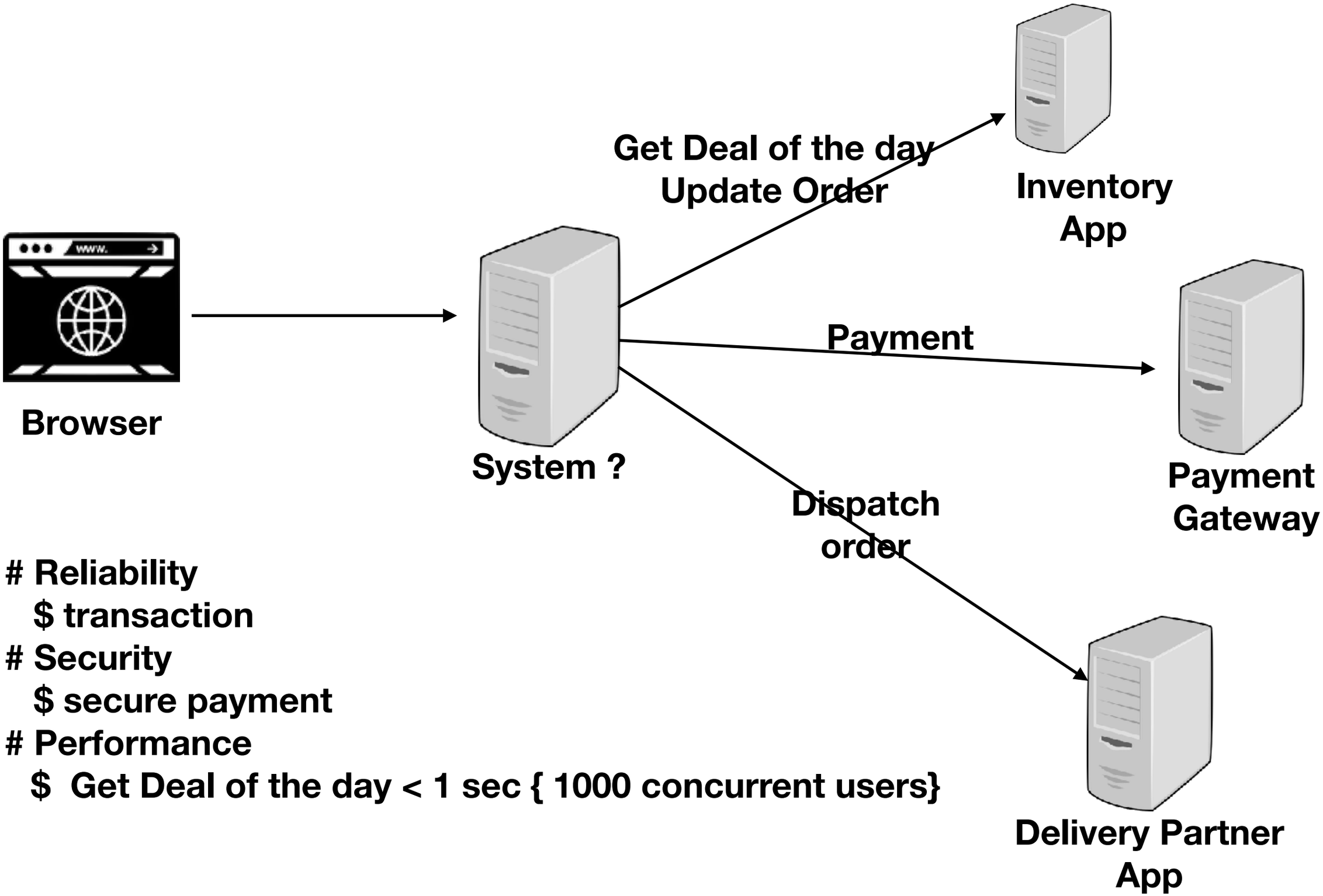
Quality requirements

 # 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

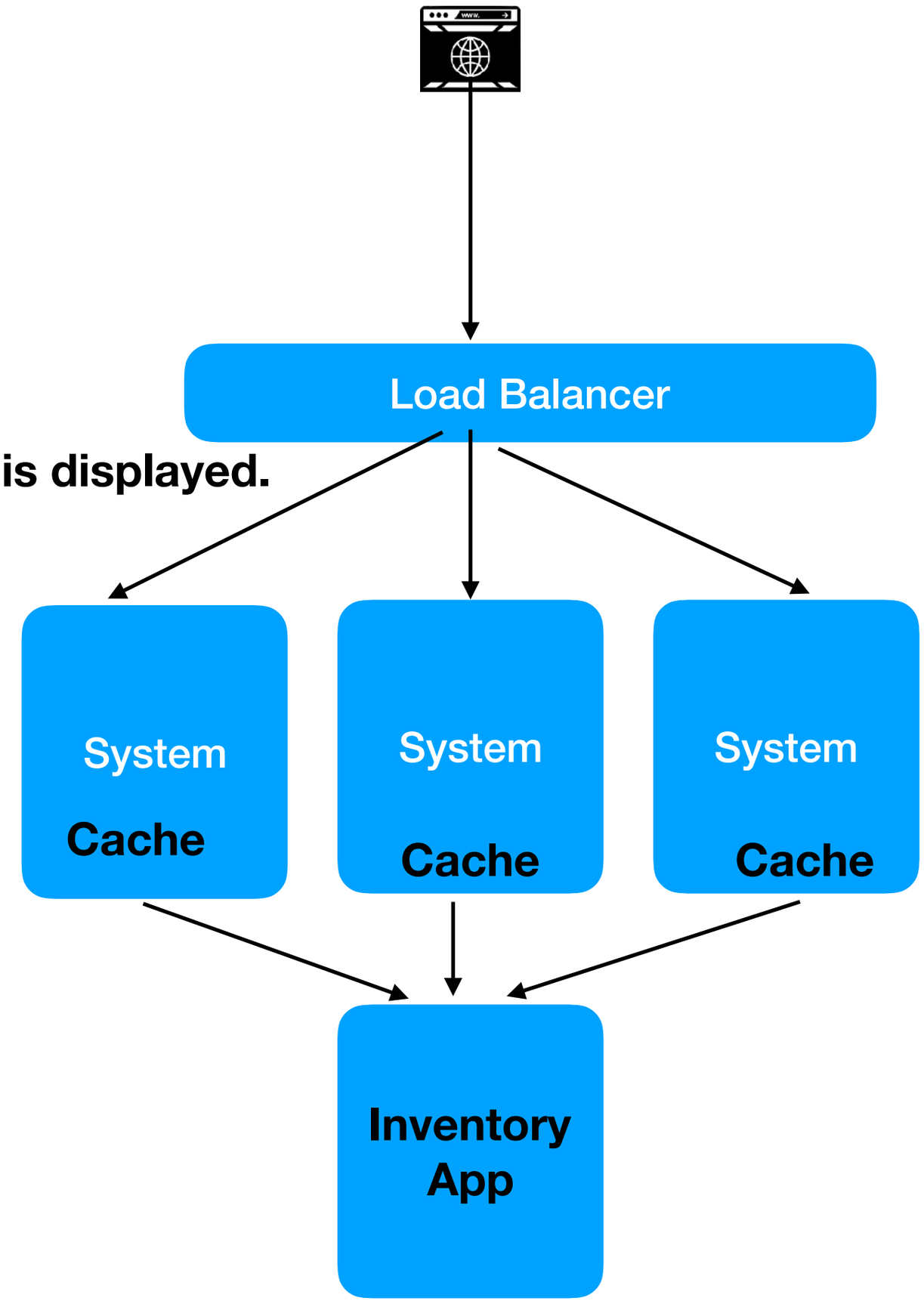
View the Product of the day
Order the product



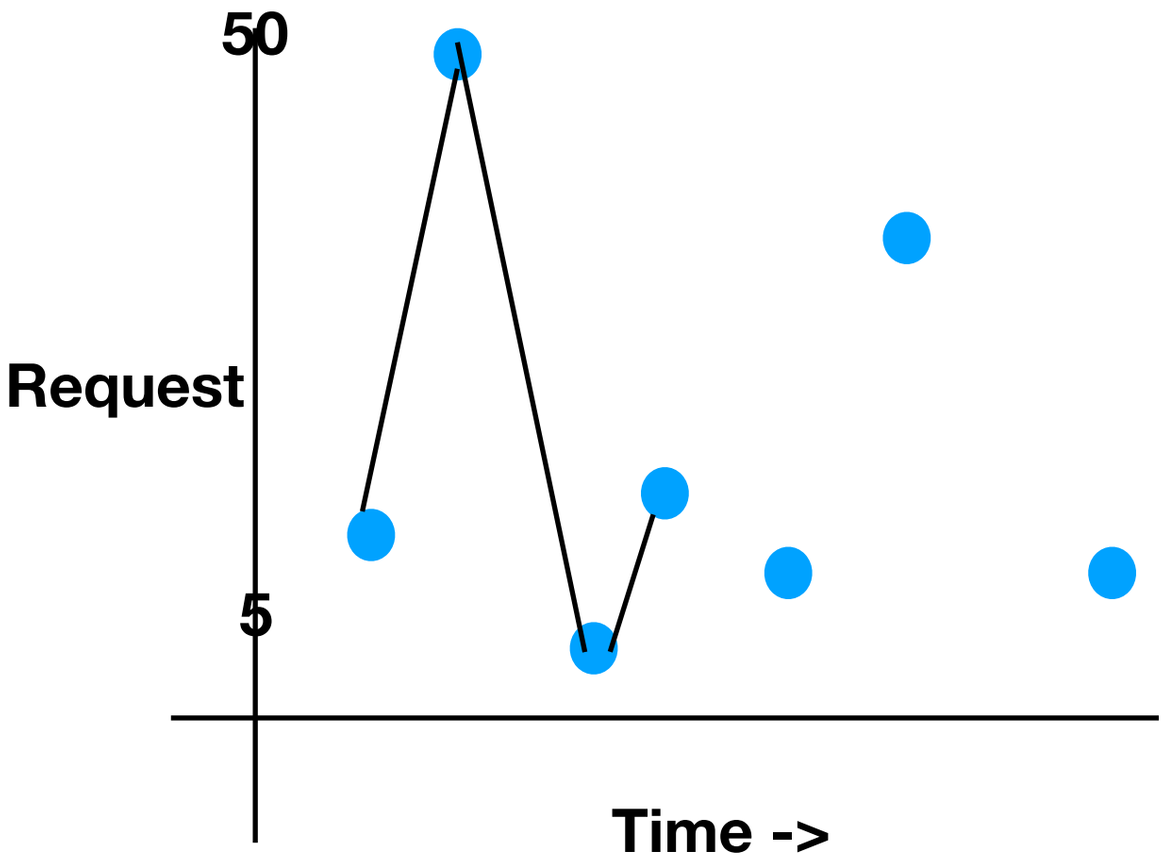
# Reliability
  $ transaction
# Security
  $ secure payment
# Performance
  $ Get Deal of the day < 1 sec { 1000 concurrent users}

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
#

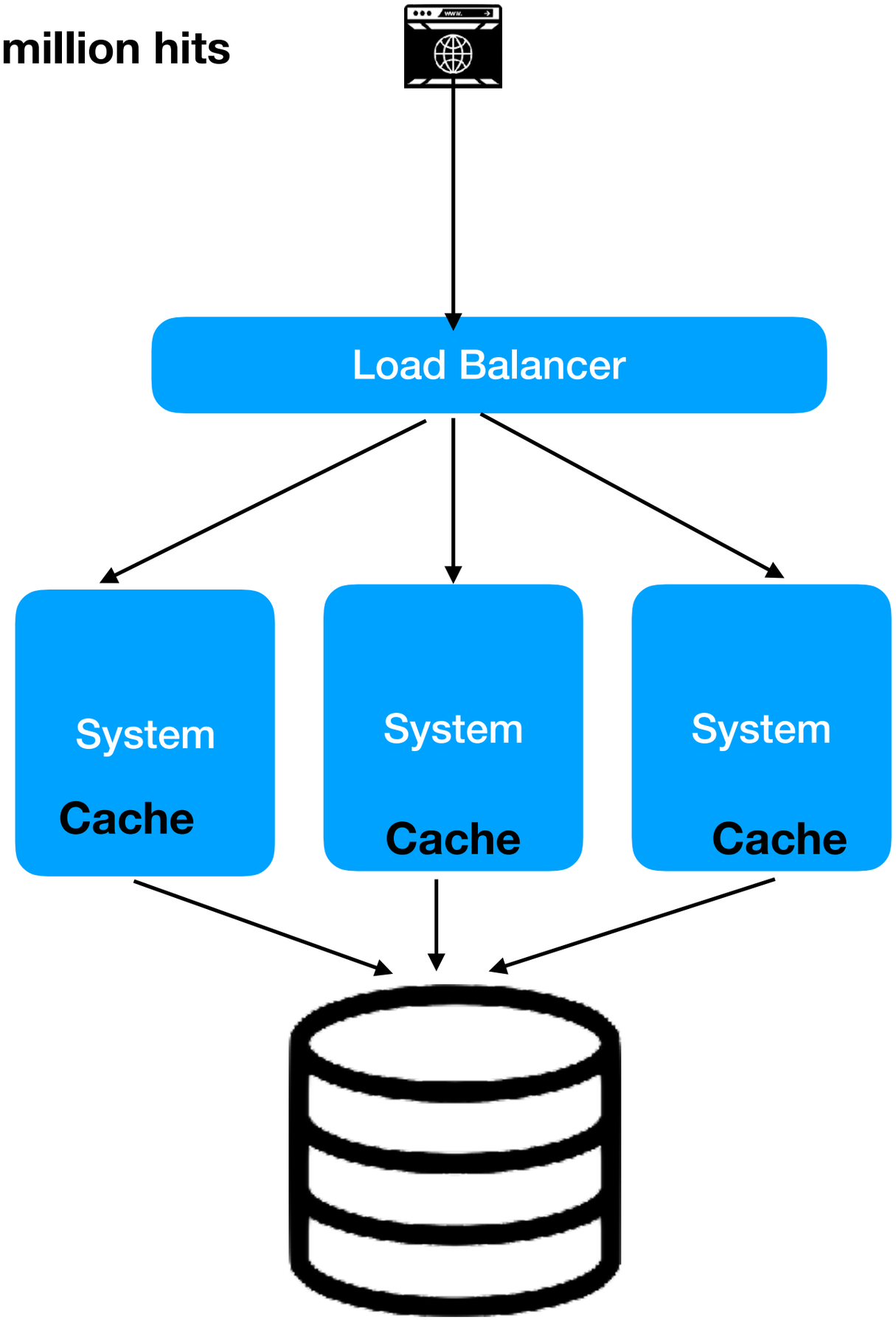


view Deal of the day
create order

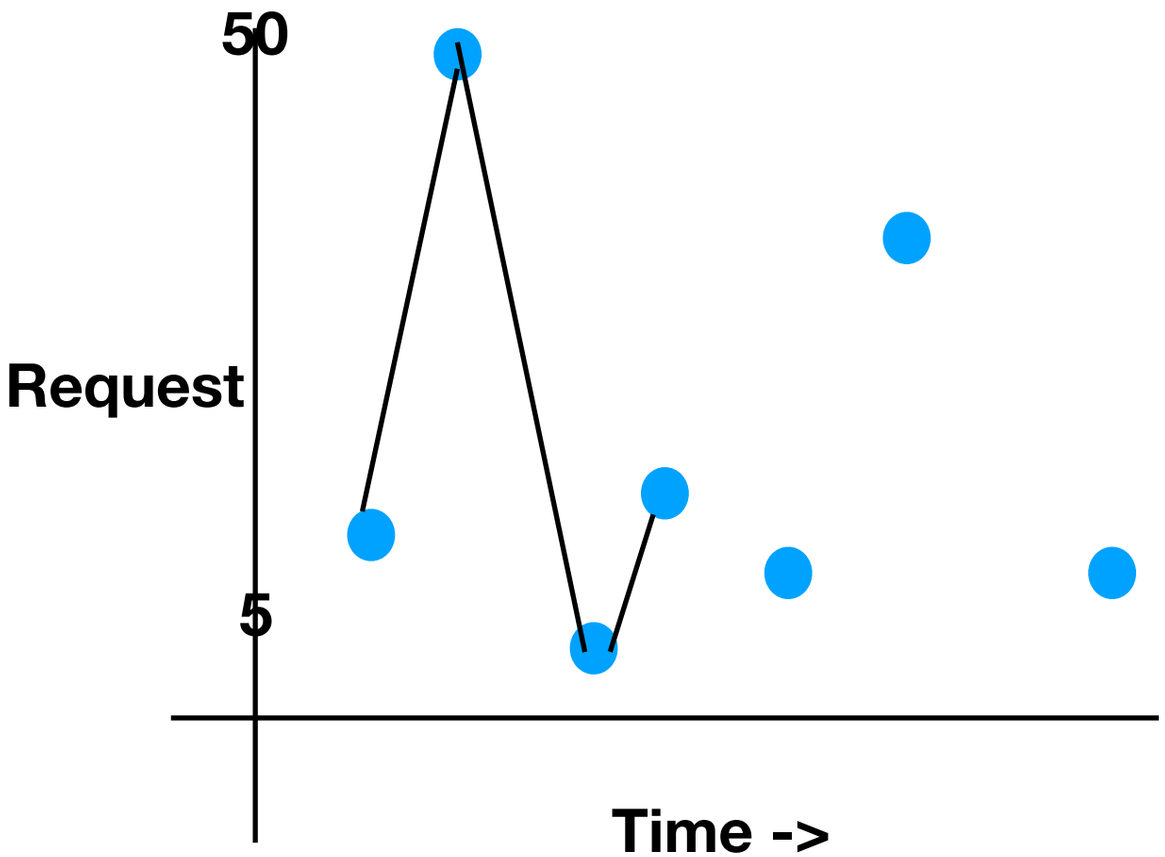


@ eventual consistency
@ stateless

1 million hits



Load Leveling



Browser

<— 4

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

<— 6

CDN

<— 3

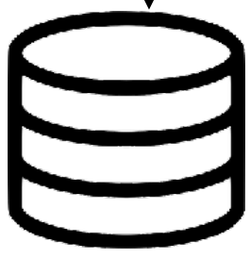
Web server

<— 5

Cache server

<— 2

App server



<— 1

Db server

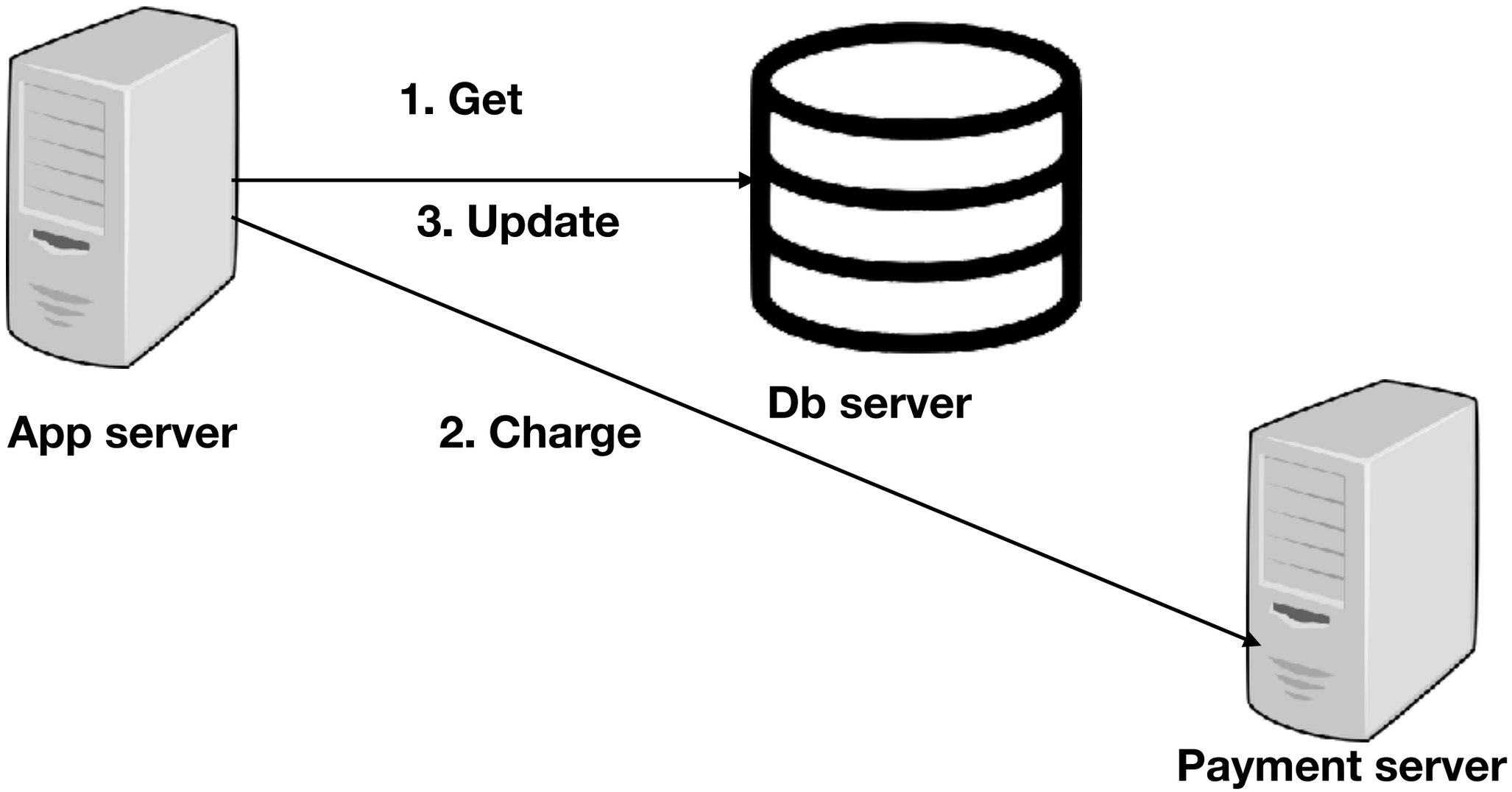
Cache

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

Data cache
Page cache

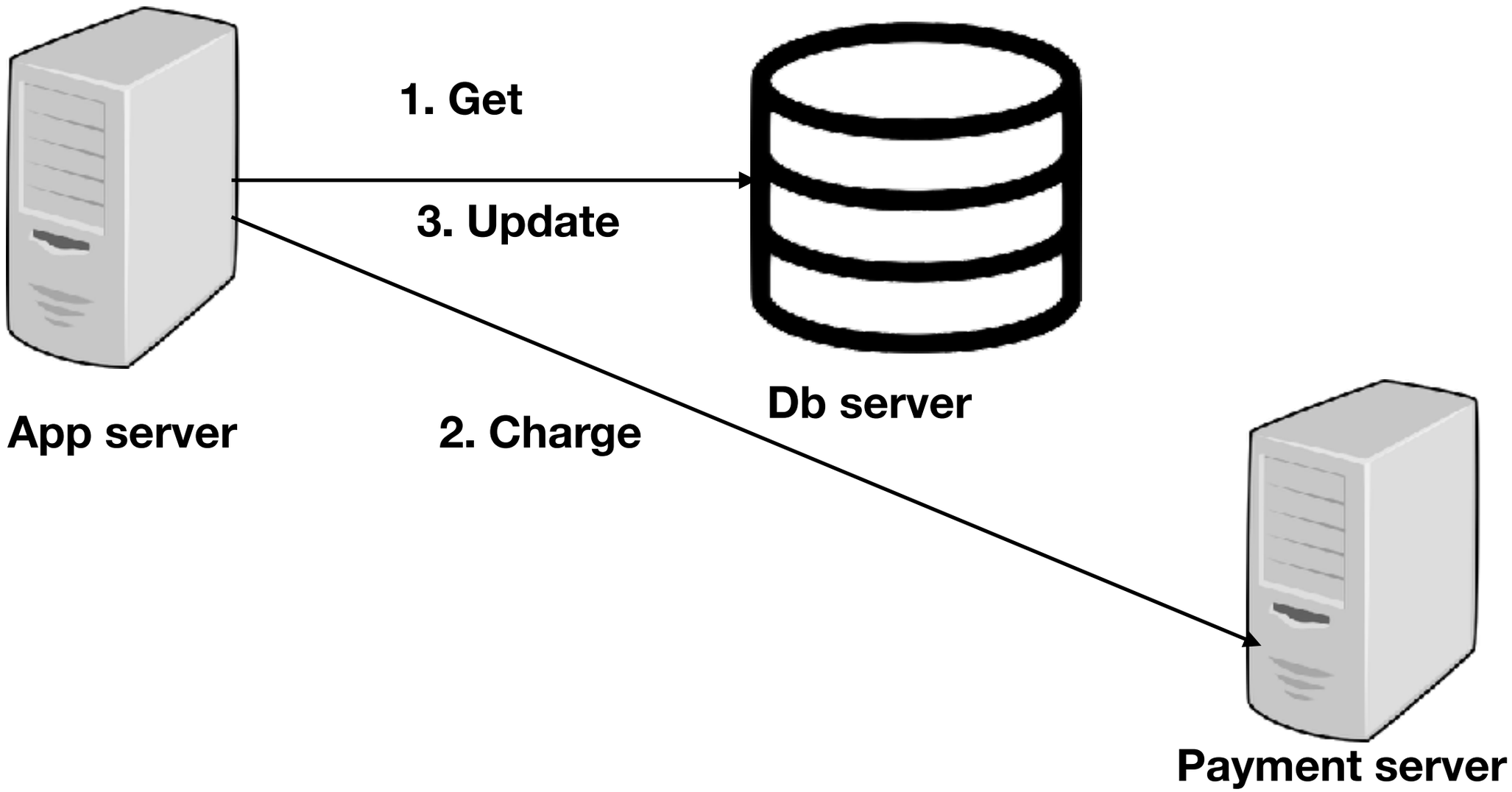
cron job invokes job every 30 days

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
}
```

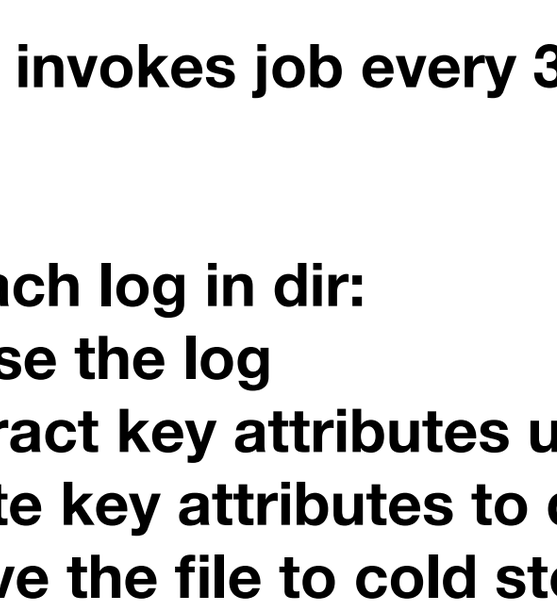
1 million
Devices

1 hour/ 10 mb / core

10 mb/day
Each file < 10 kb

Edge
Device

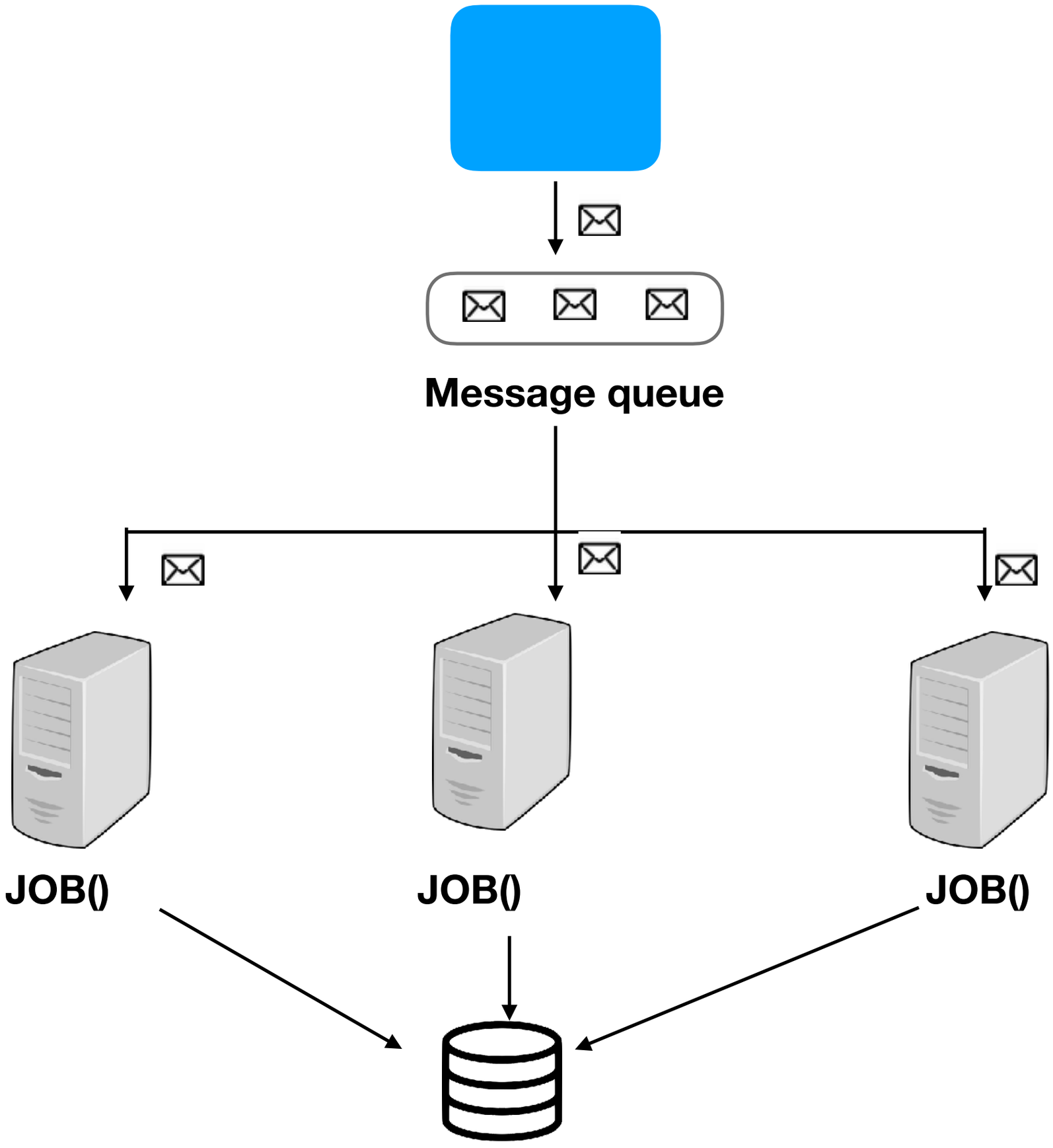
Telemetry logs



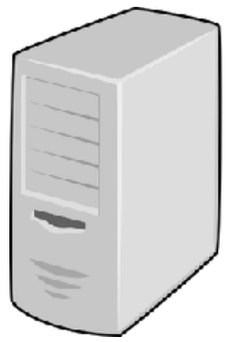
App server

Db server

Disconnected
Connected
Occasionally connected

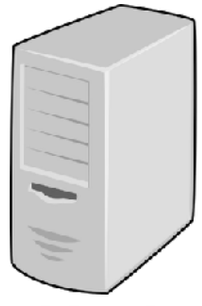

Message queue
JOB()
JOB()
JOB()

Each file < 10 kb



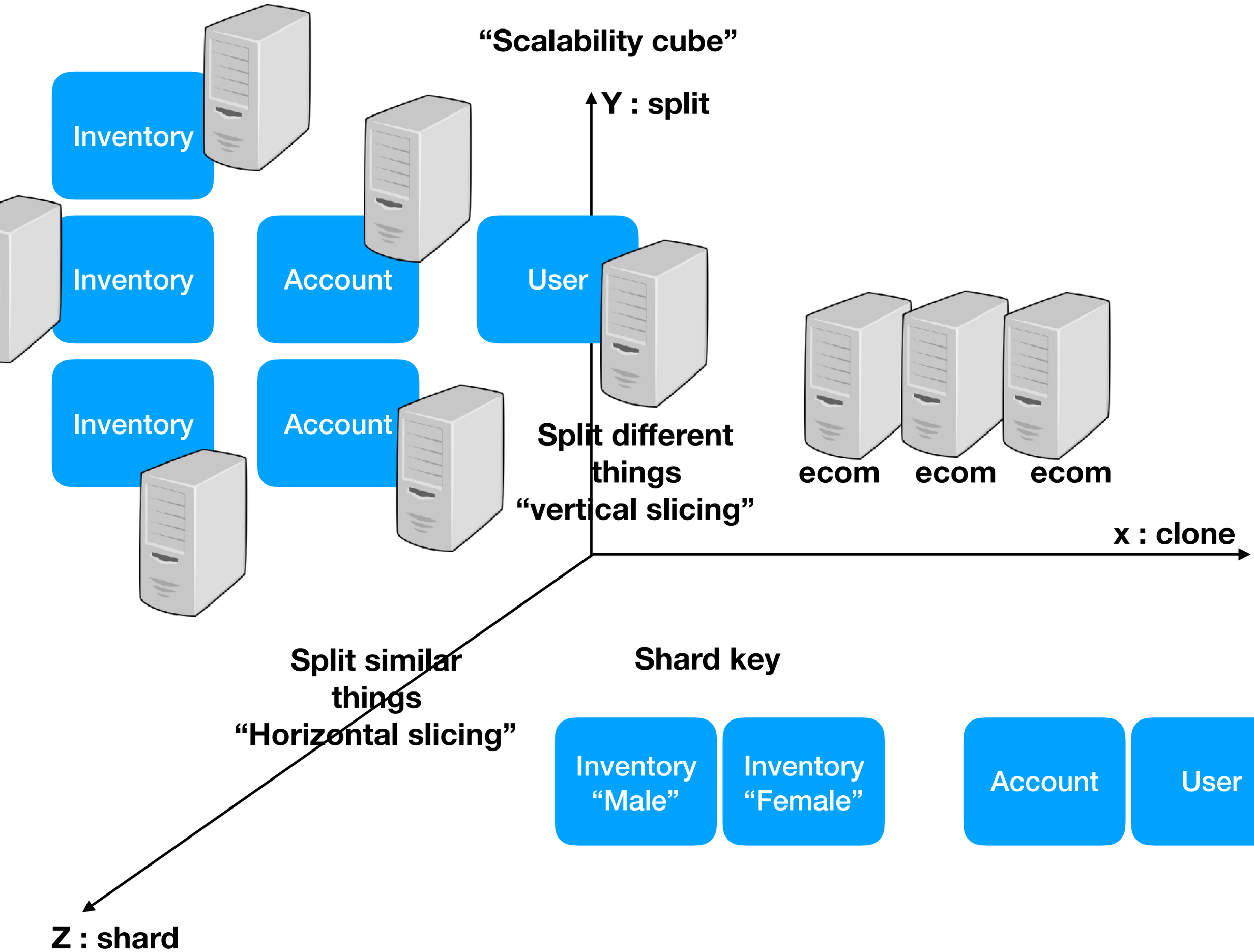
Vm1

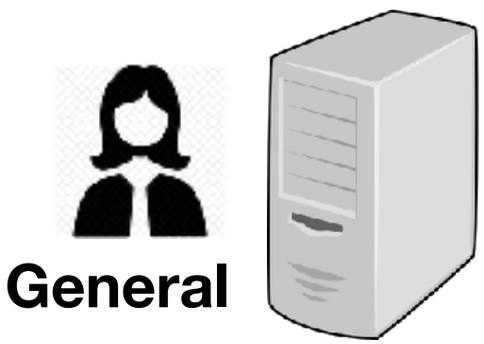
Vm2



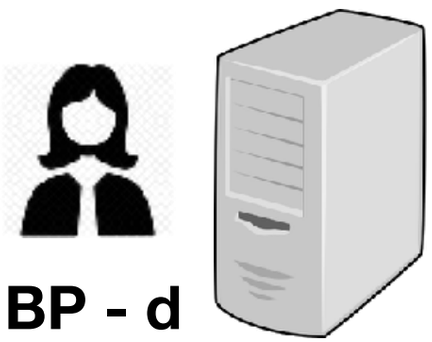
Vm3

```
# 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
}
```

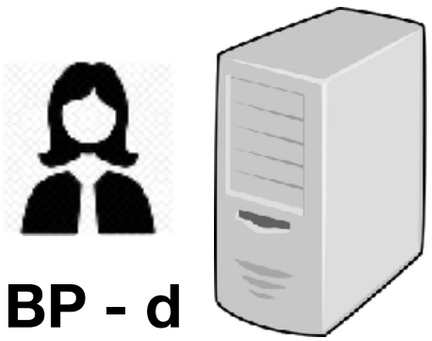

“Scalability cube”
Y : split
Inventory
Inventory
Account
User
Inventory
Account
Split different things “vertical slicing”
ecom
ecom
ecom
x : clone
Split similar things “Horizontal slicing”
Shard key
Inventory “Male”
Inventory “Female”
Account
User
Z : shard

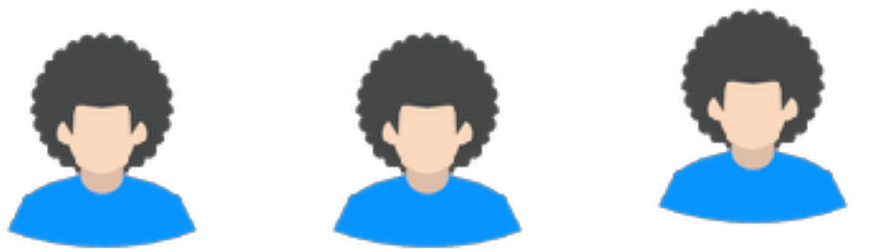

General


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

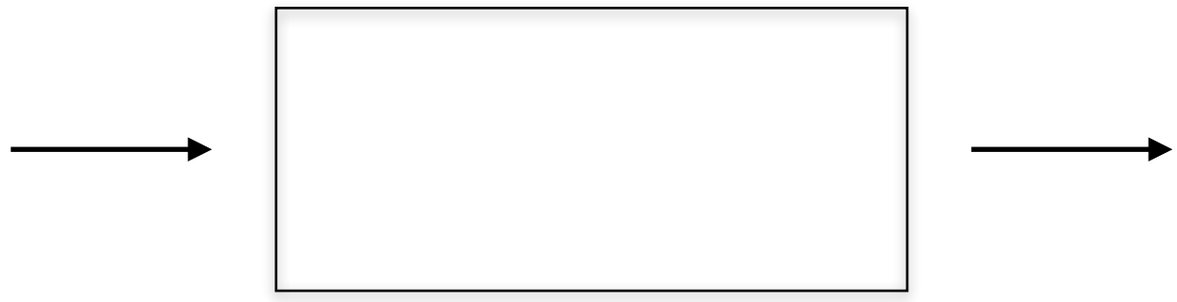
(What) Quality attribute ?	(How) Approach (tactic/style) ?	Measure ?
• Performance	• Caching • Compression • Parallel • Object pooling •	• Latency • Throughput • Response time •

(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 ?

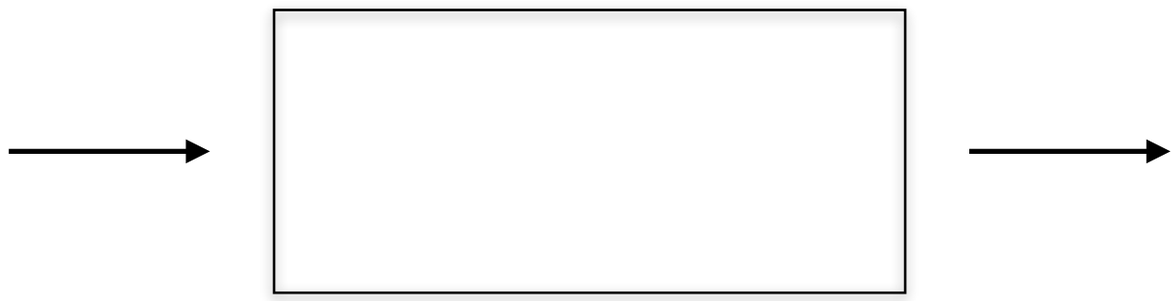
Measure ?



(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

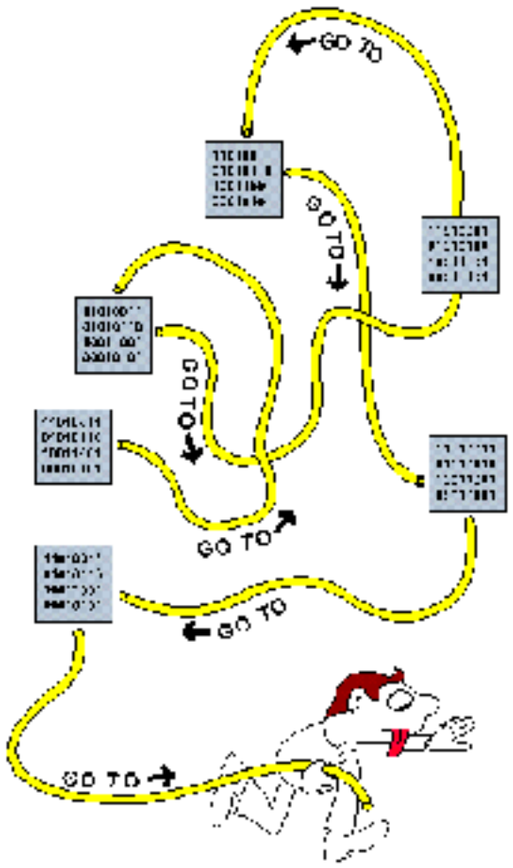
Decide

- 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)

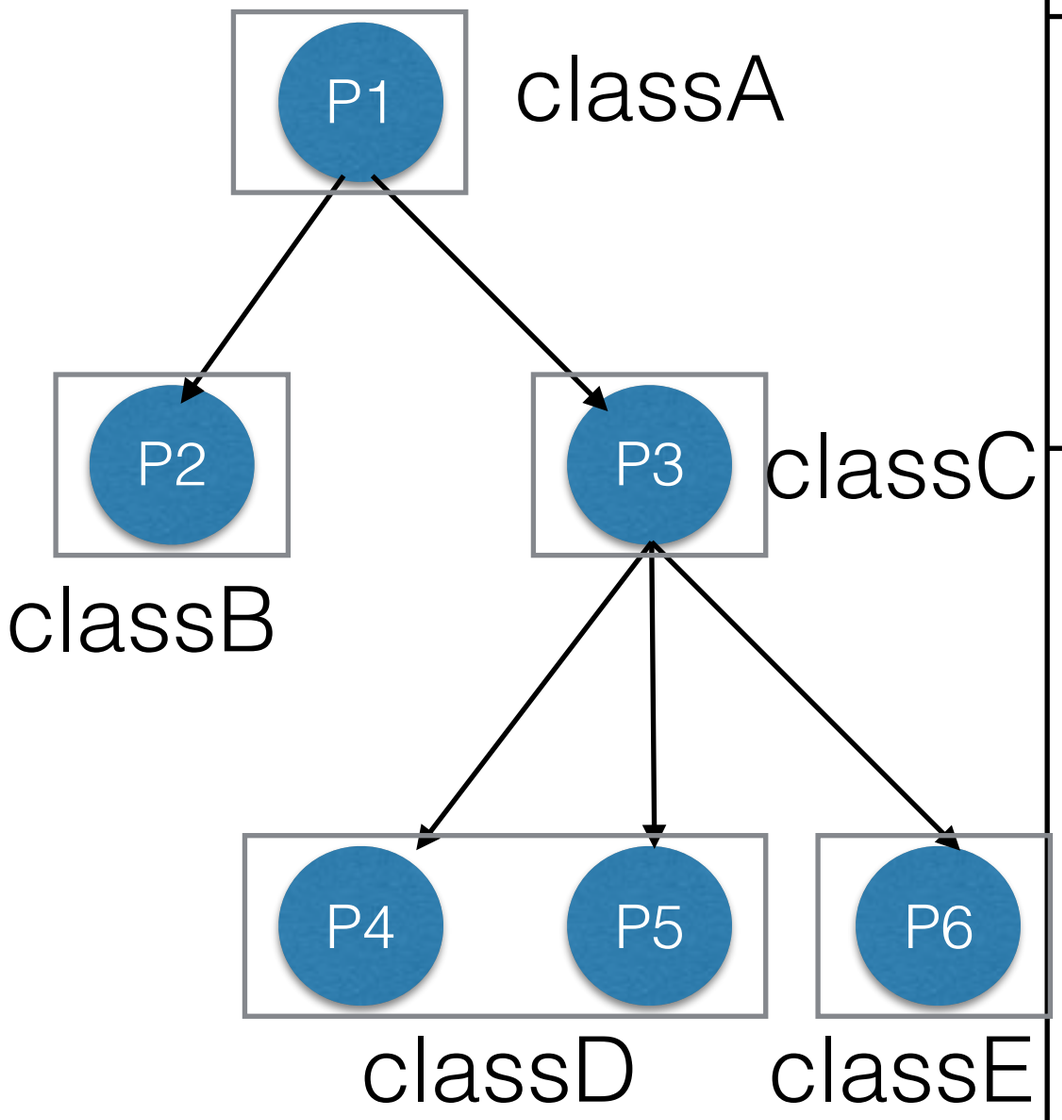
Vertical Architect

Security Architect
UX
Data
Infra
Cloud
Java
…

Proc vs OO vs fun

Procedural Prog

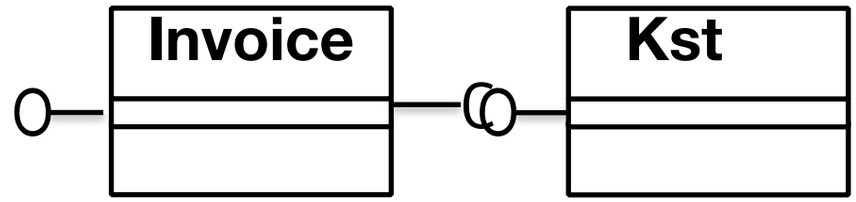
(tree)

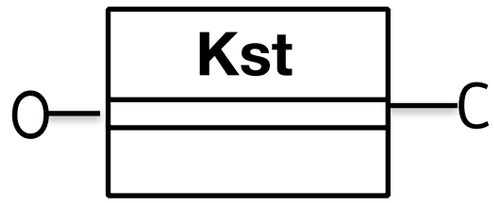


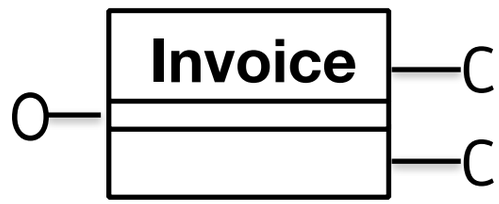
(top down)

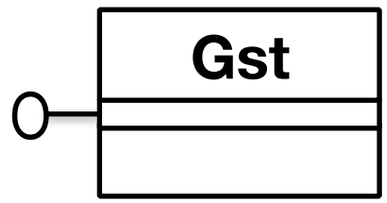
OO Prog

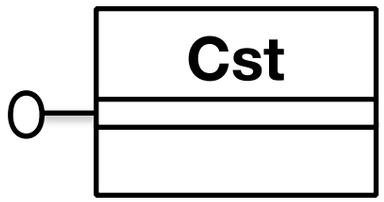
(Lego)





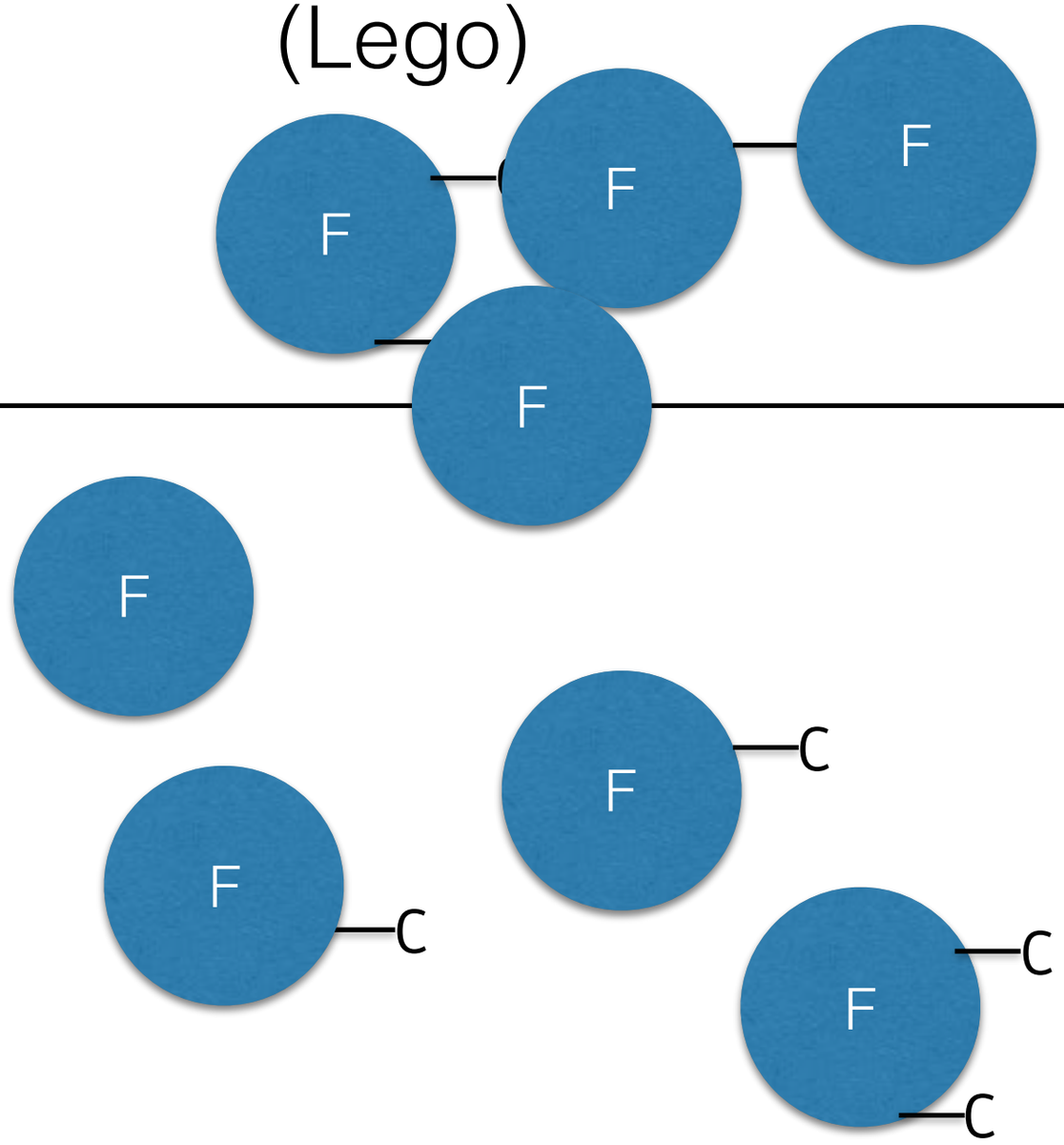




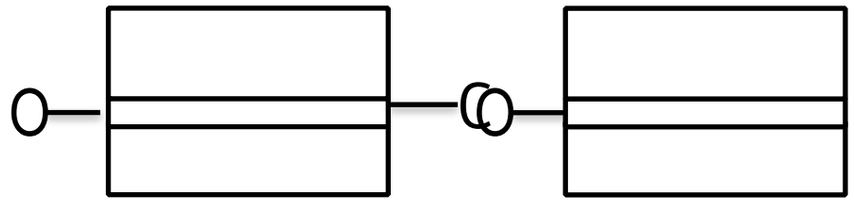


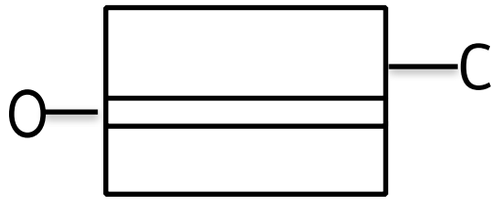
(bottom up)

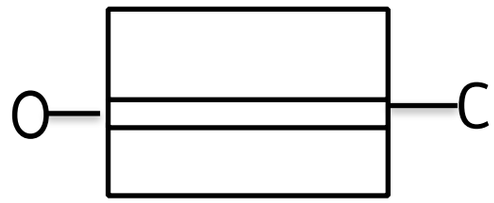
Functional Prog (Lego)

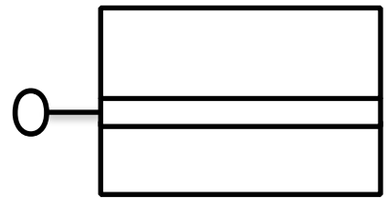


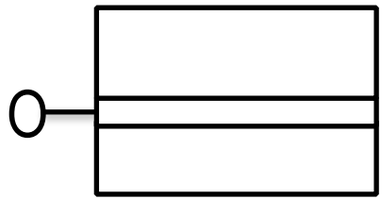
OO Prog (Lego)











	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

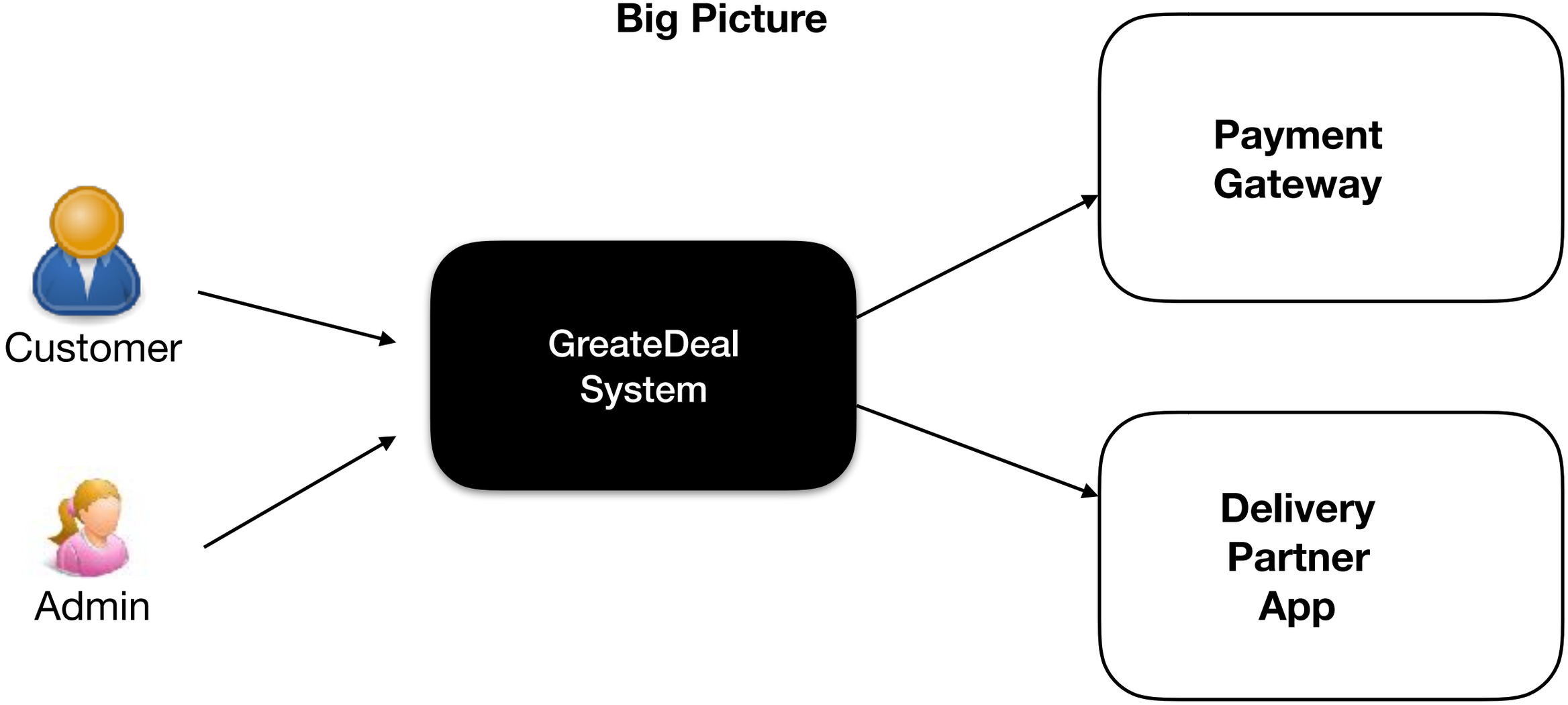
GreatDeal

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)

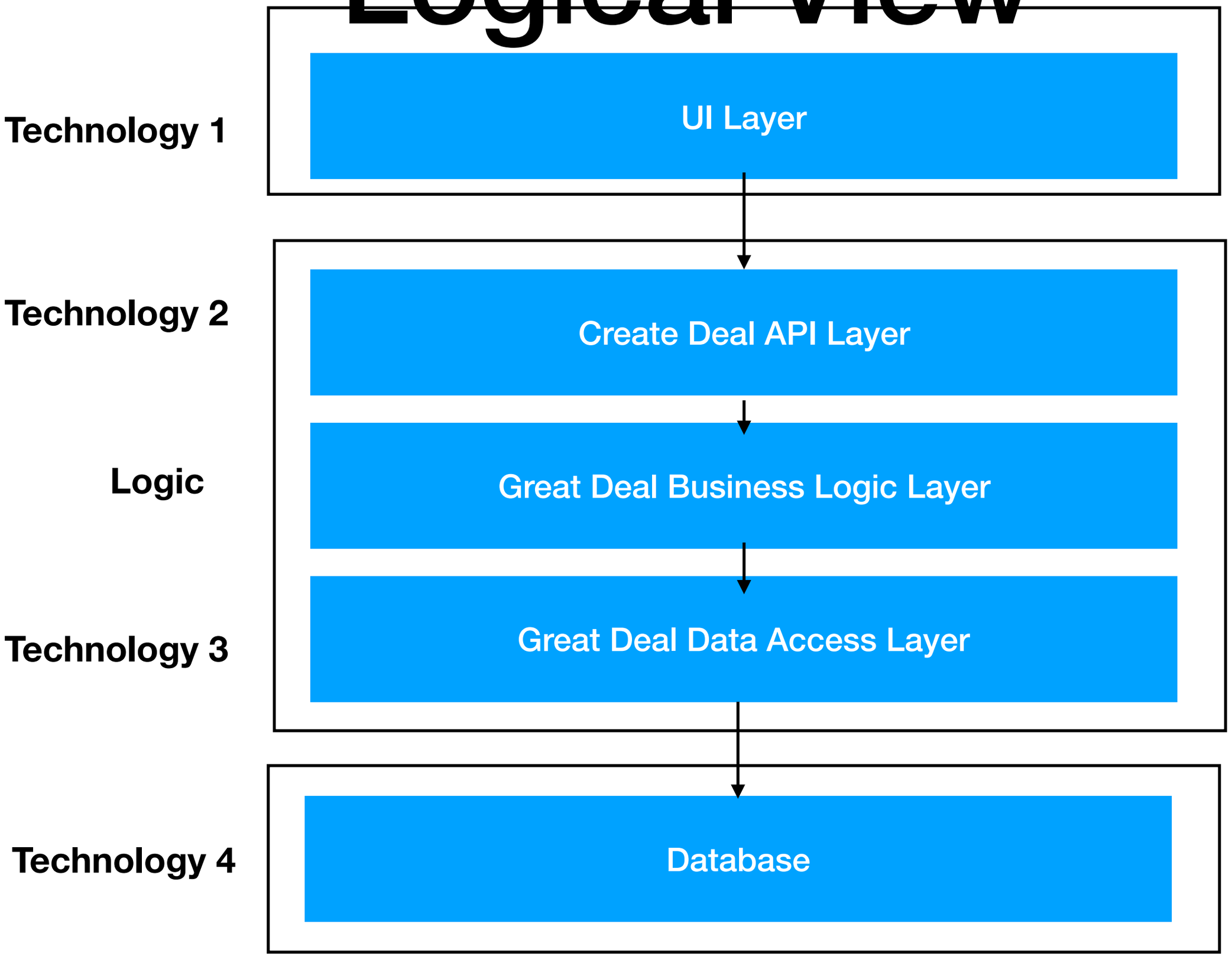
- When a user enters incorrect bidding value into the bidding Web App while product information is displayed. The system prints an error message for the respective user. User is able to bid again with correct value within 30 seconds. (robustness)

Constraints View

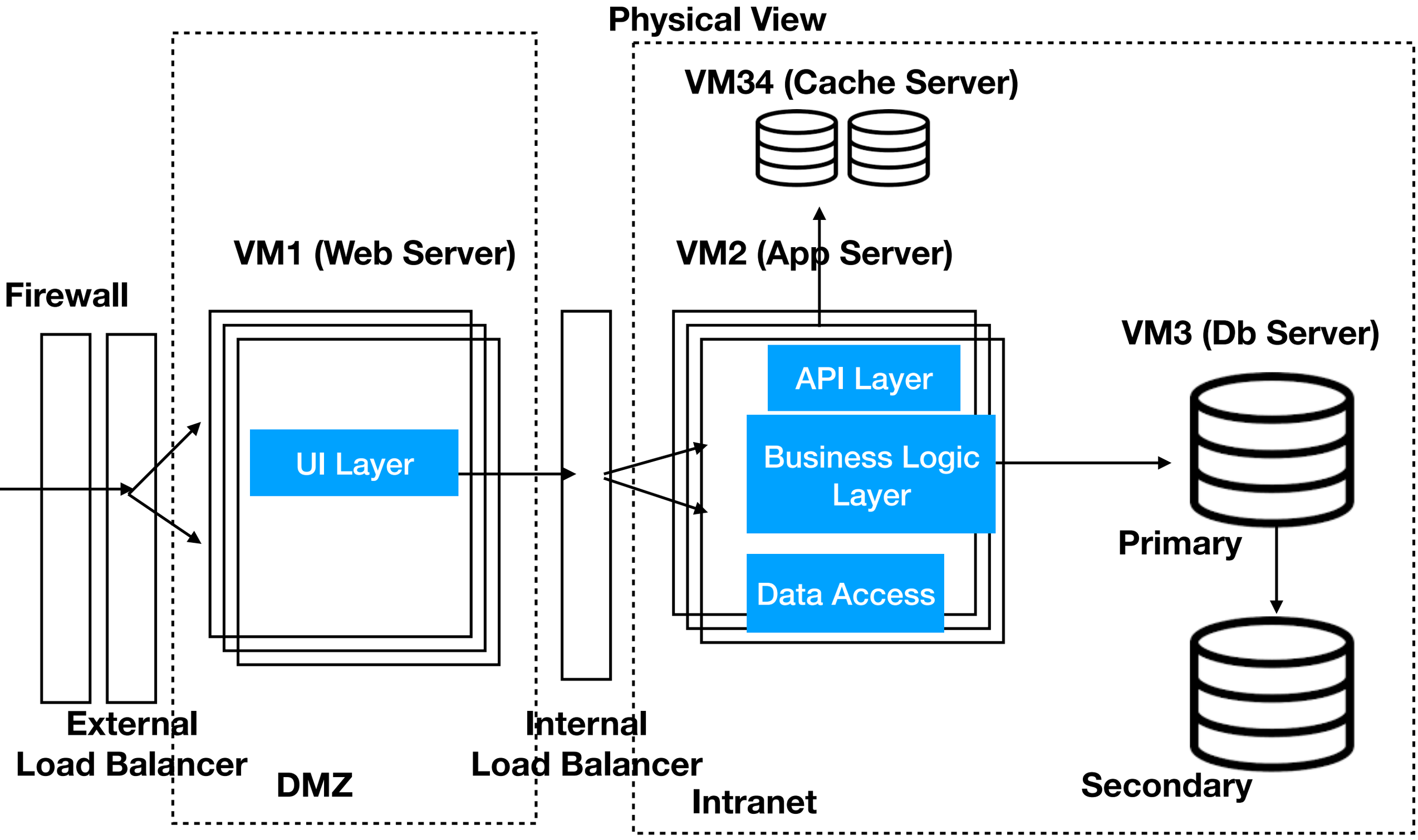
- Should support Internet Explorer 11
- Use open source stack
- API should be built using python

GreatDeal Architecture

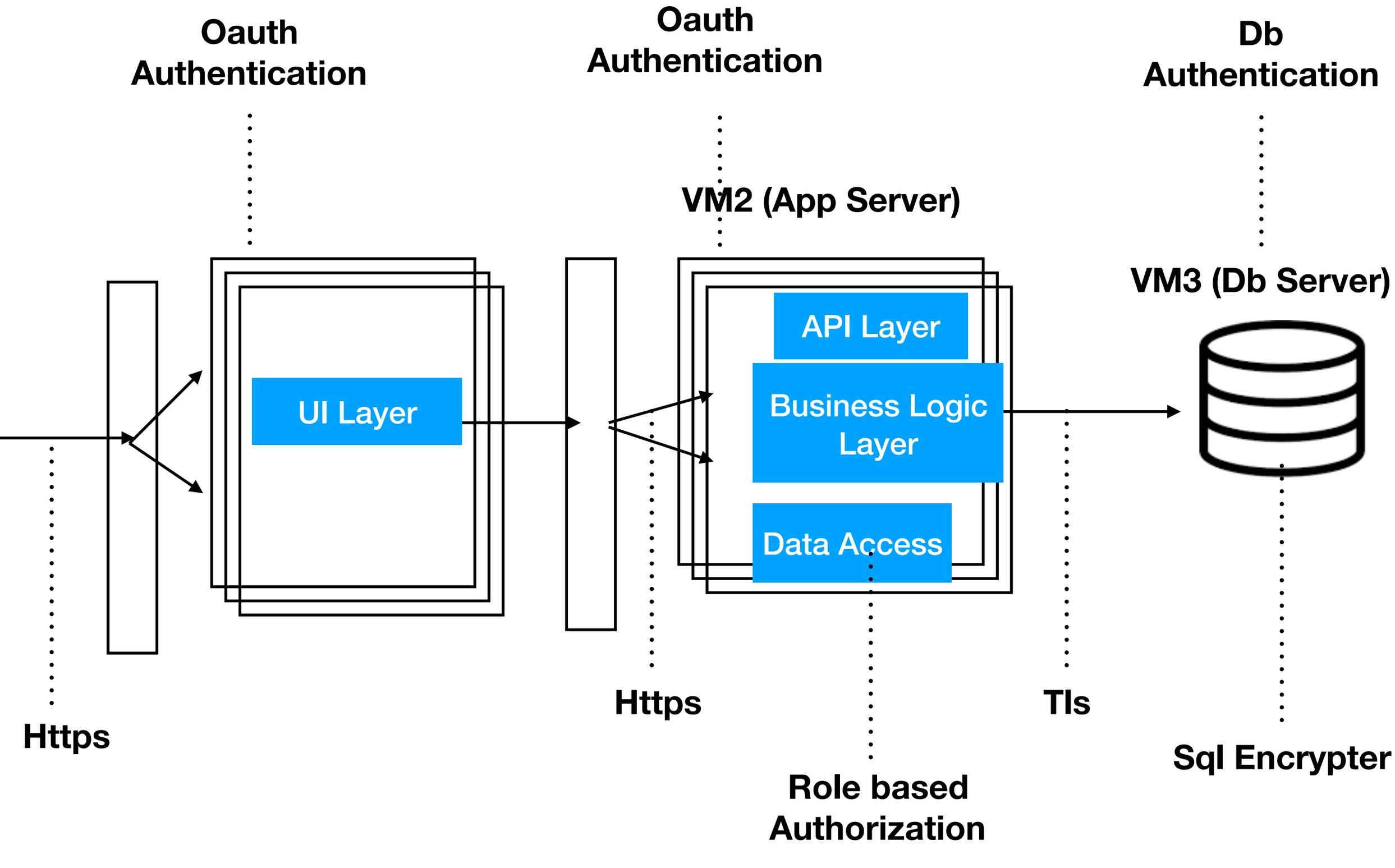
Logical View



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

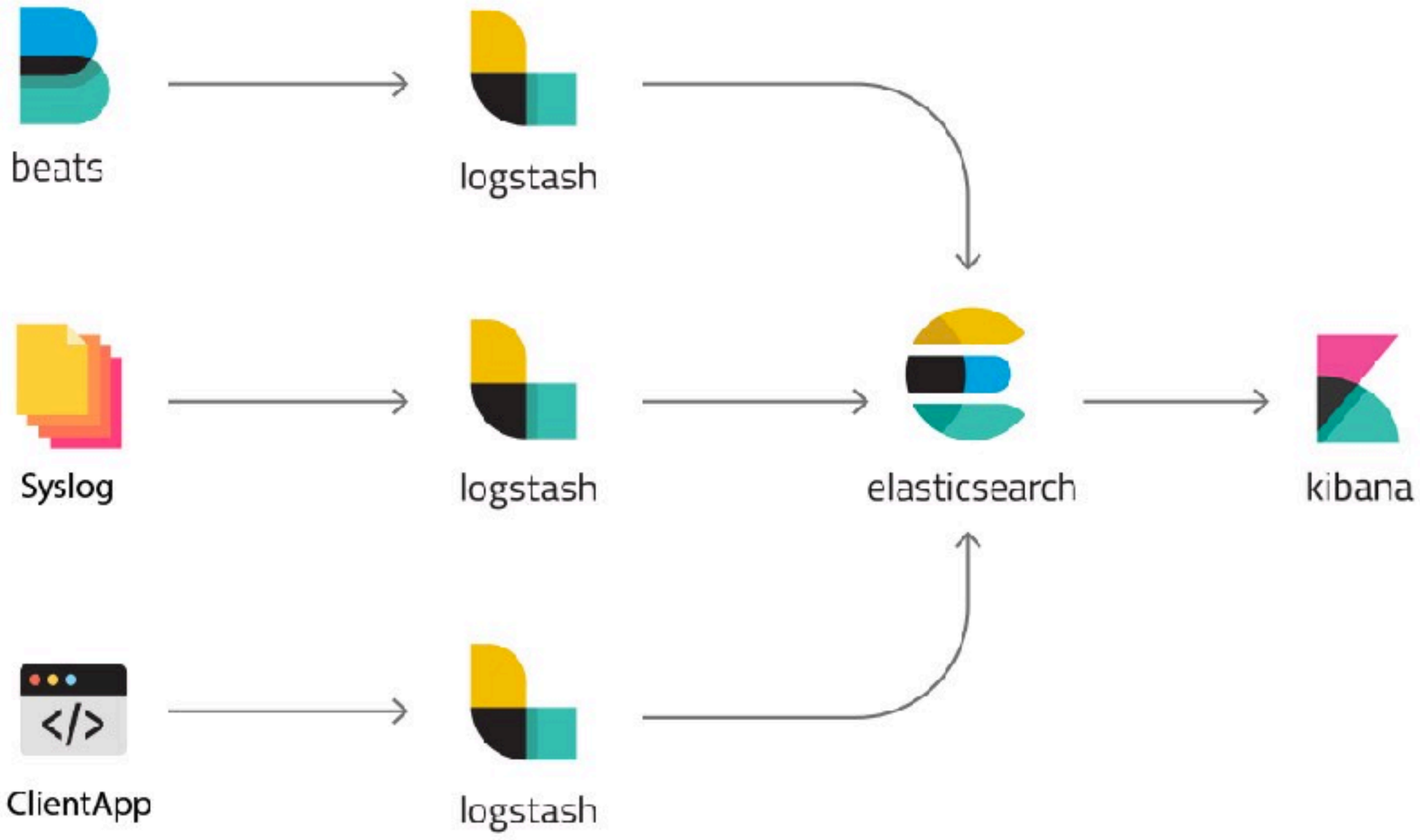
s1 (< 5 sec)
s2(99.99%)
s3 (...)
...

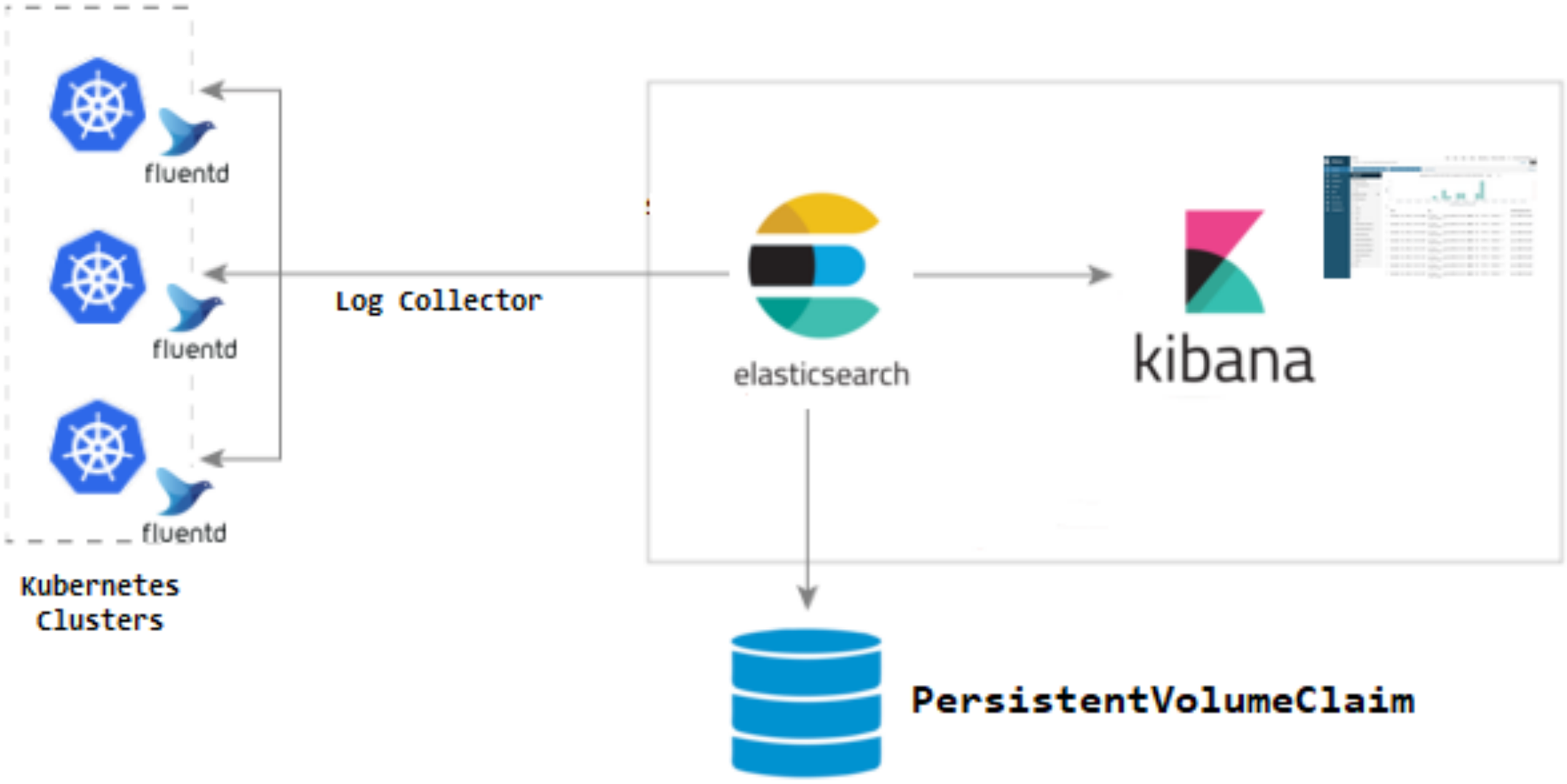
	Ut		
	Performance		**Security**
	S1	**S3**	**S4**

 # analyse Scenario -> Approach

S1 -> A1, A2
S2 -> A6,A8, A9
Sx-> Ay

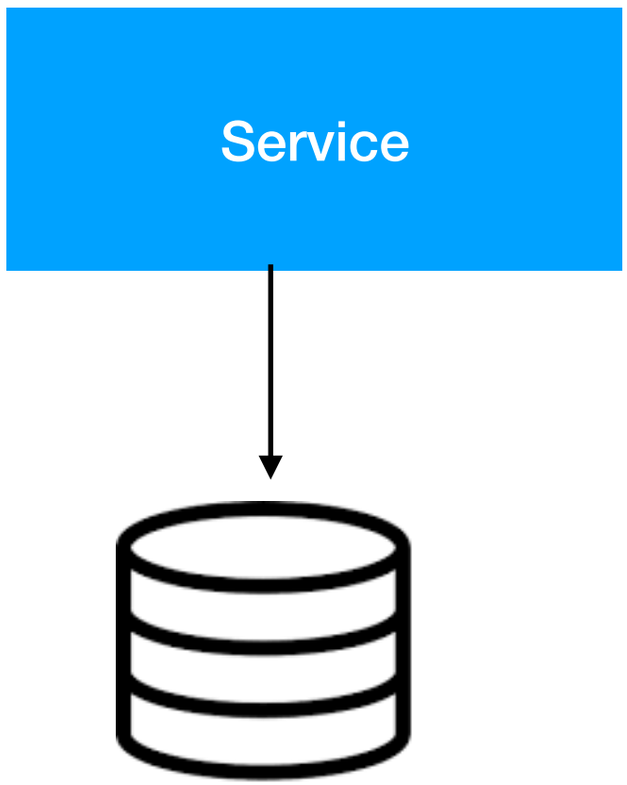
 # brainstorm for scenarios


beats
logstash
Syslog
logstash
elasticsearch
kibana
ClientApp
logstash


fluentd
fluentd
fluentd
Kubernetes
Clusters
Log Collector
elasticsearch
kibana
PersistentVolumeClaim

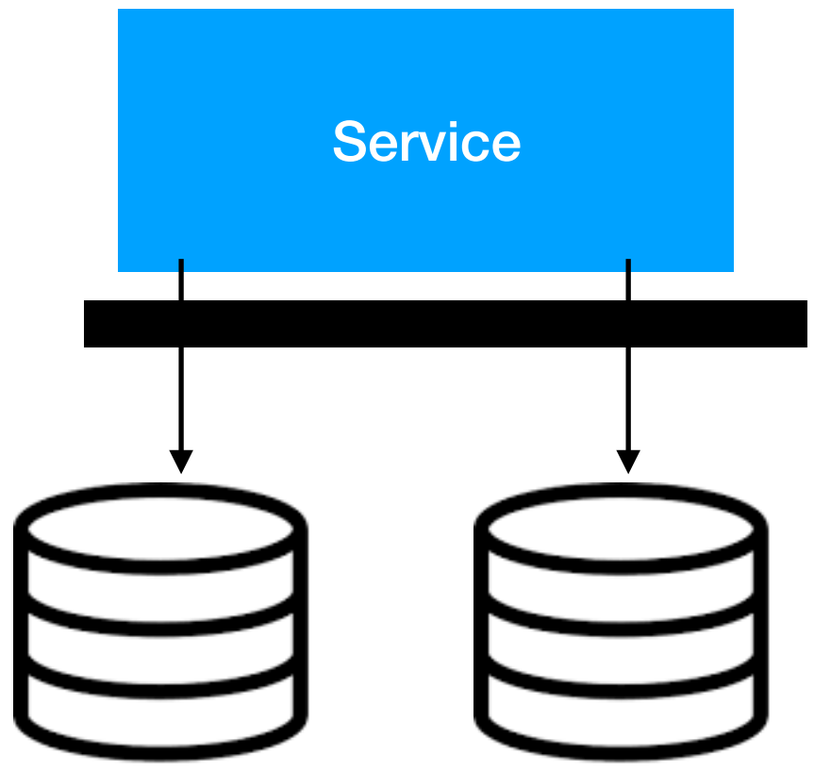
Transaction

Db transition



Begin
Do
Do
Do
Commit

2 phase commit
(JTX, MSDTC, …)



custom logic
(book keeping)

SAGA

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 >>

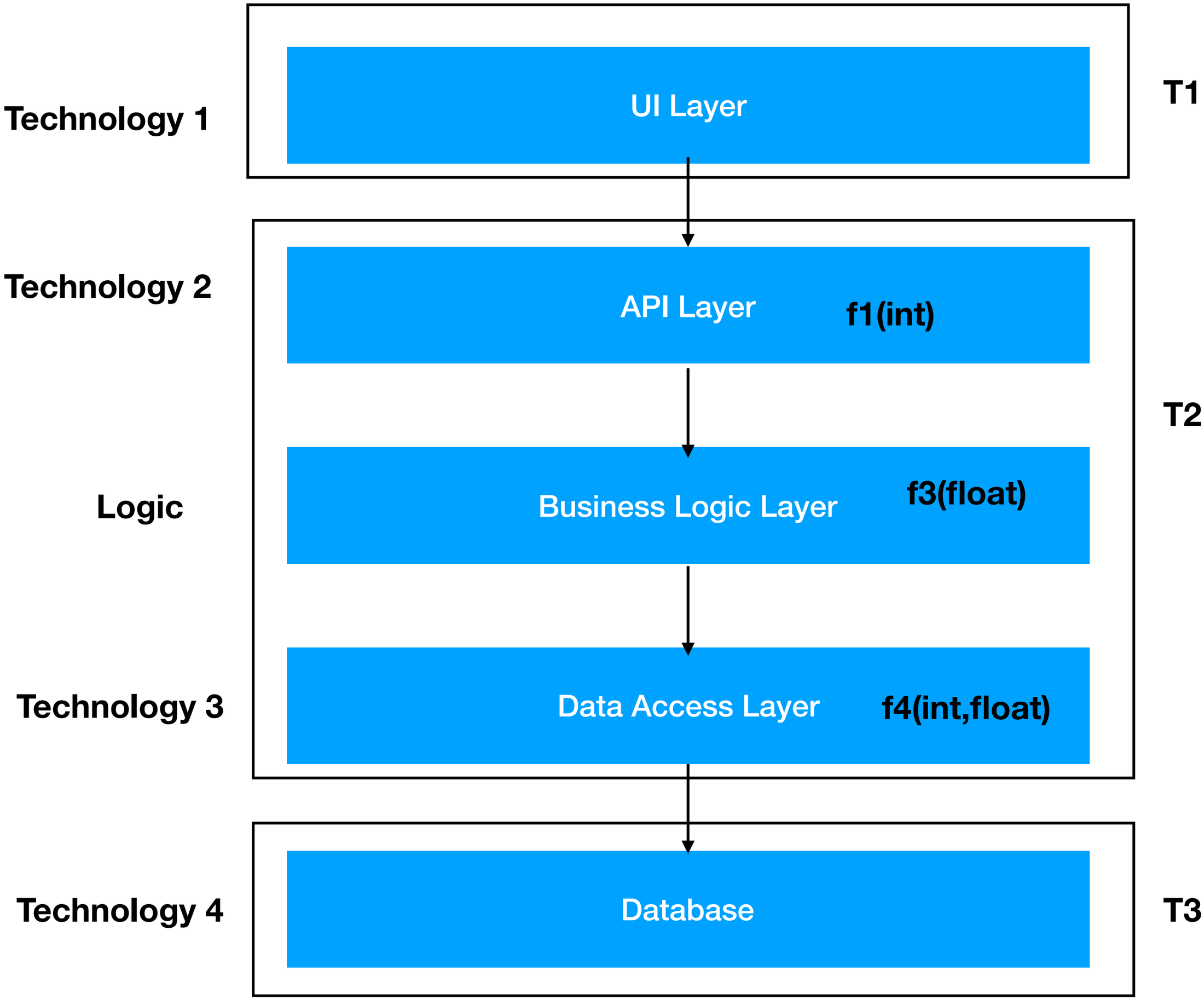
Part1.1

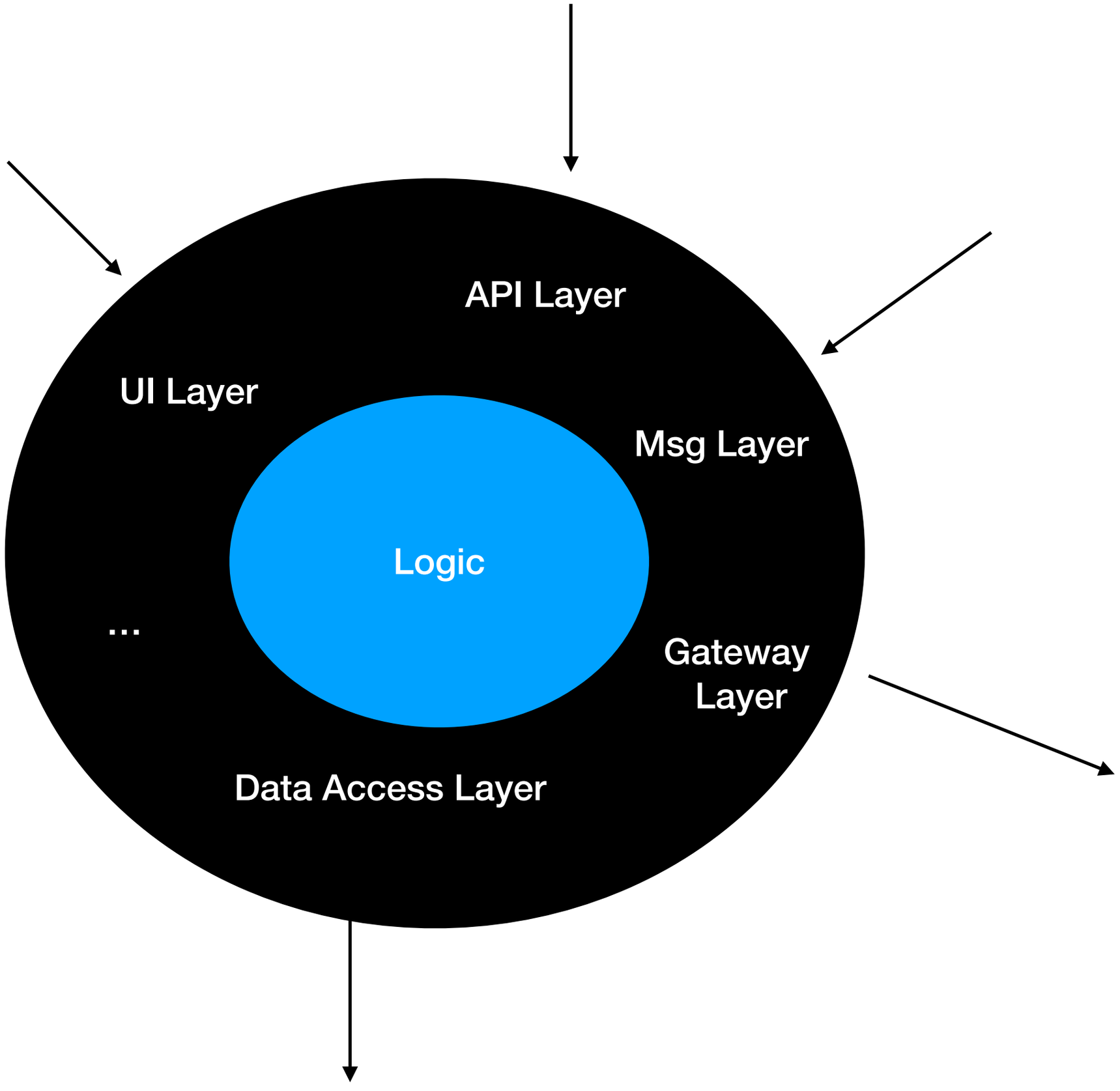
Part1.2

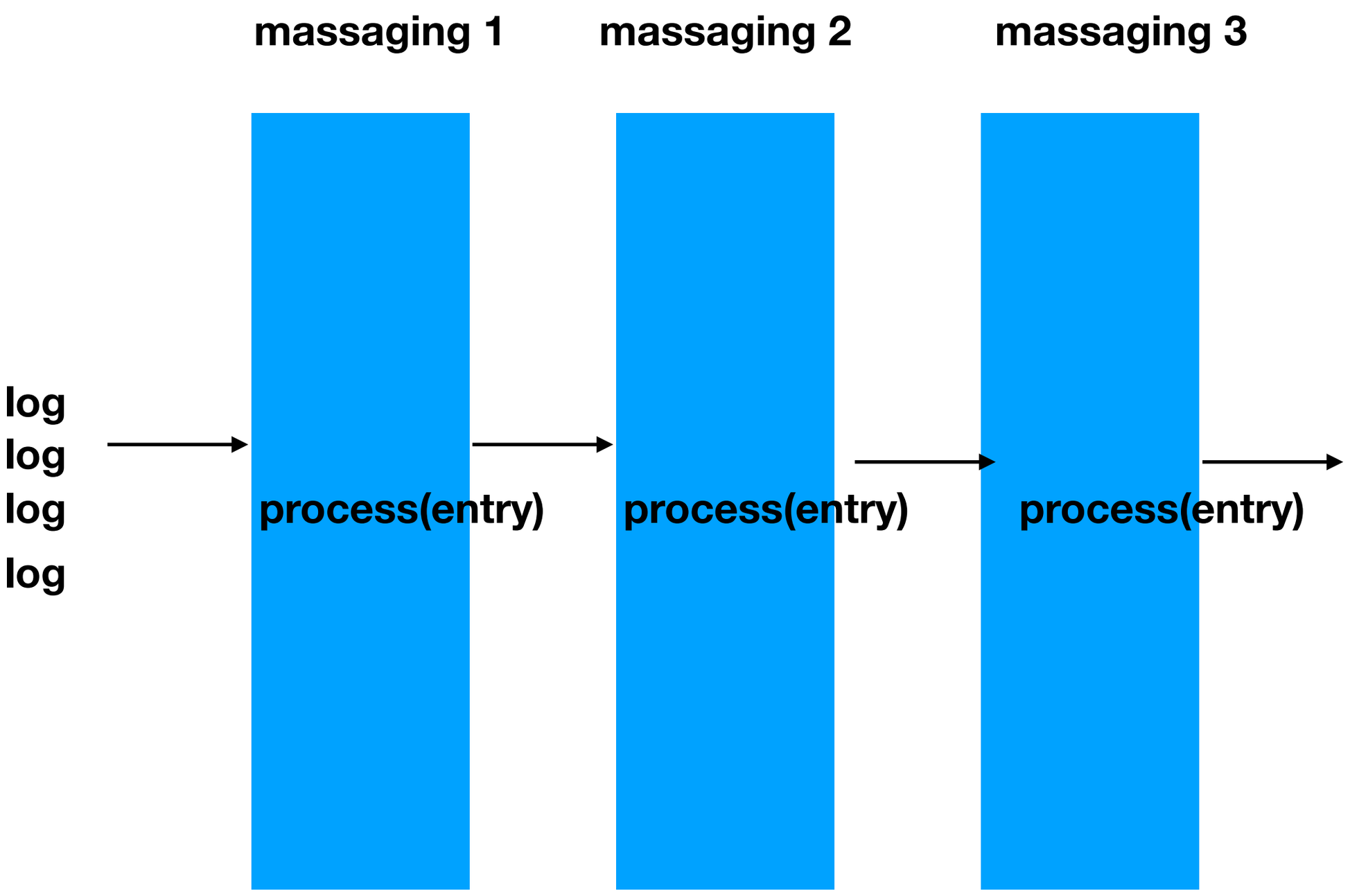
<<apply architectural tactics >>

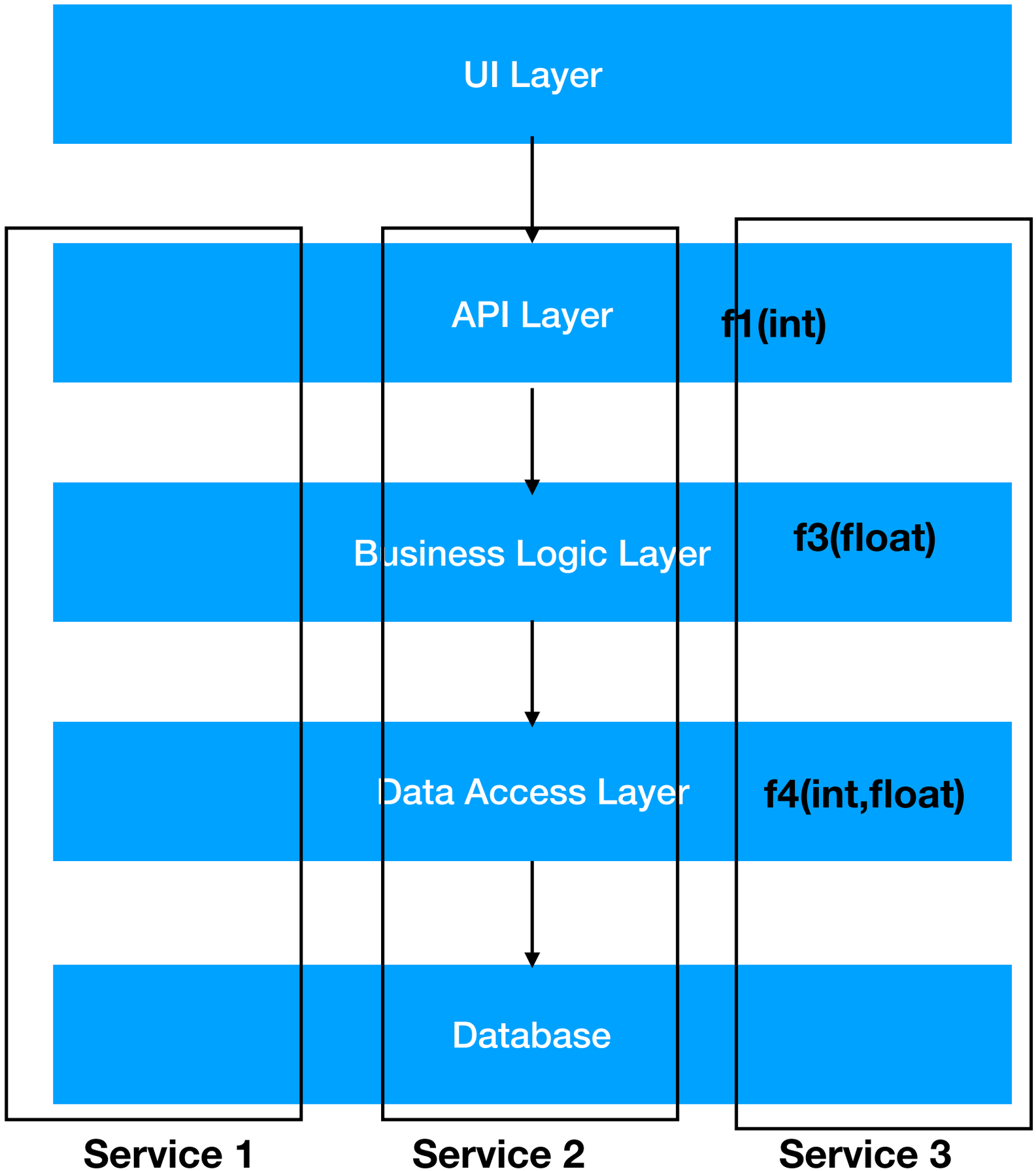
Part1.2.1

Part1.2.2

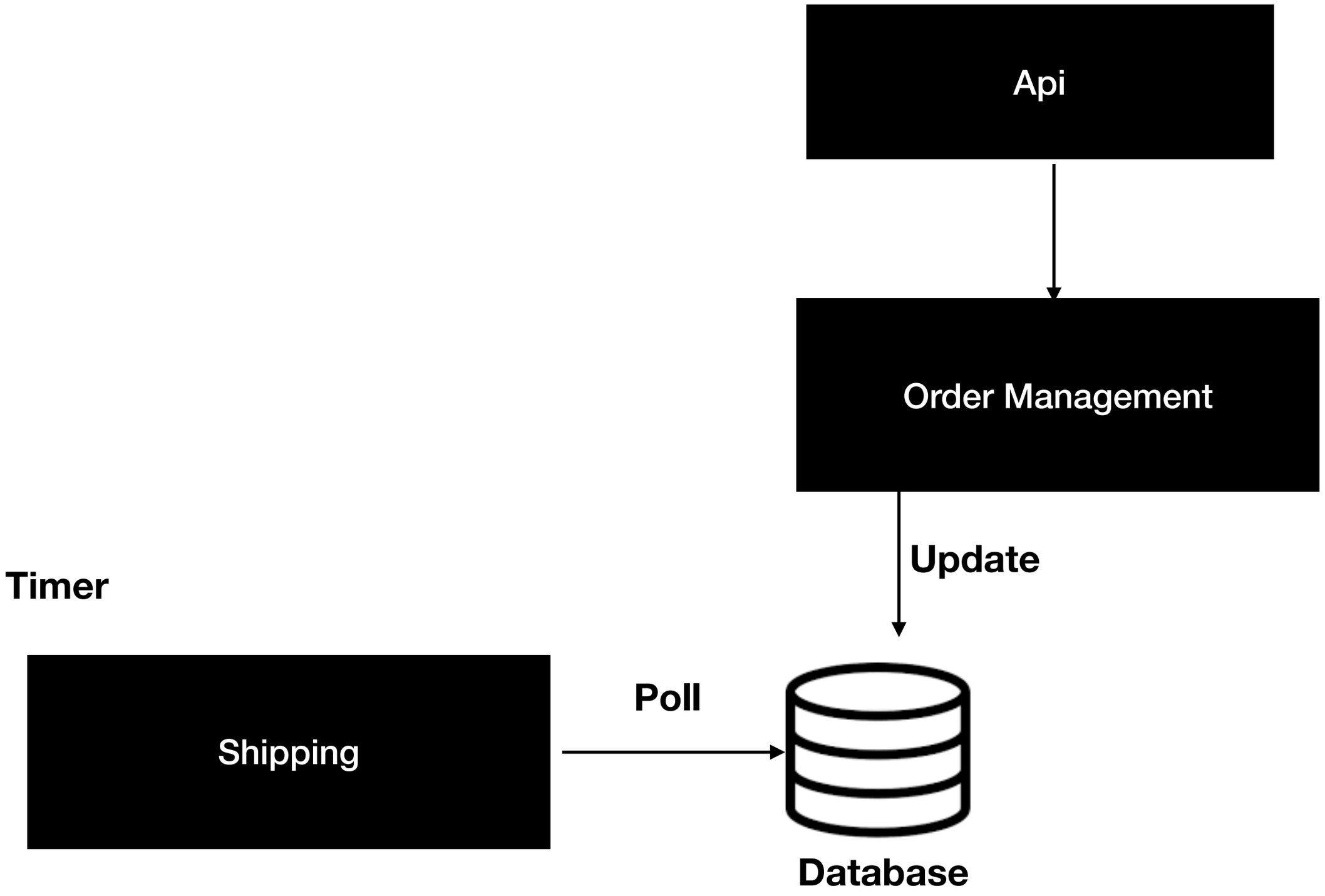

Technology 1
UI Layer
T1
Technology 2
API Layer
f1(int)
T2
Logic
Business Logic Layer
f3(float)
Technology 3
Data Access Layer
f4(int,float)
Technology 4
Database
T3


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

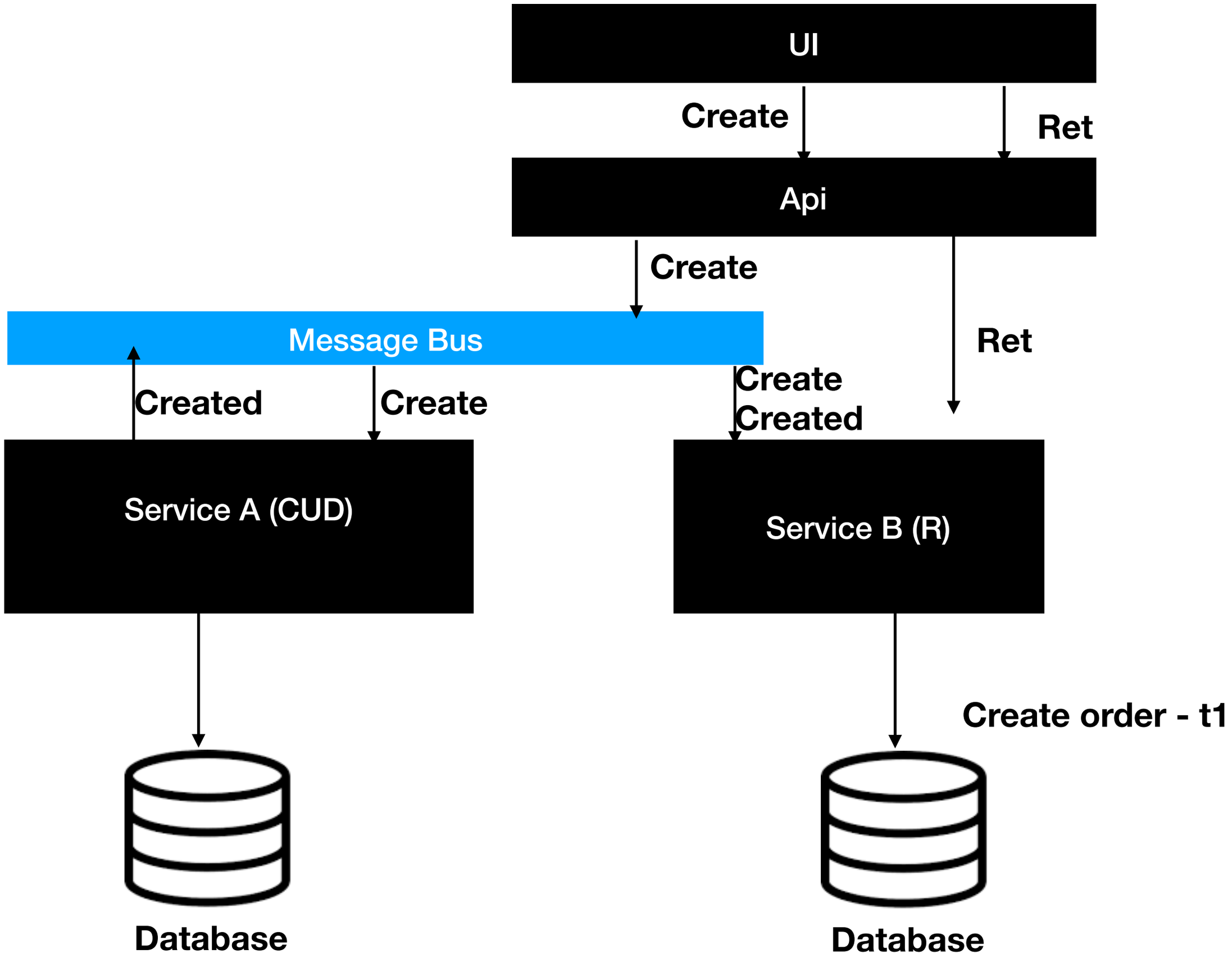
Create order cmd

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

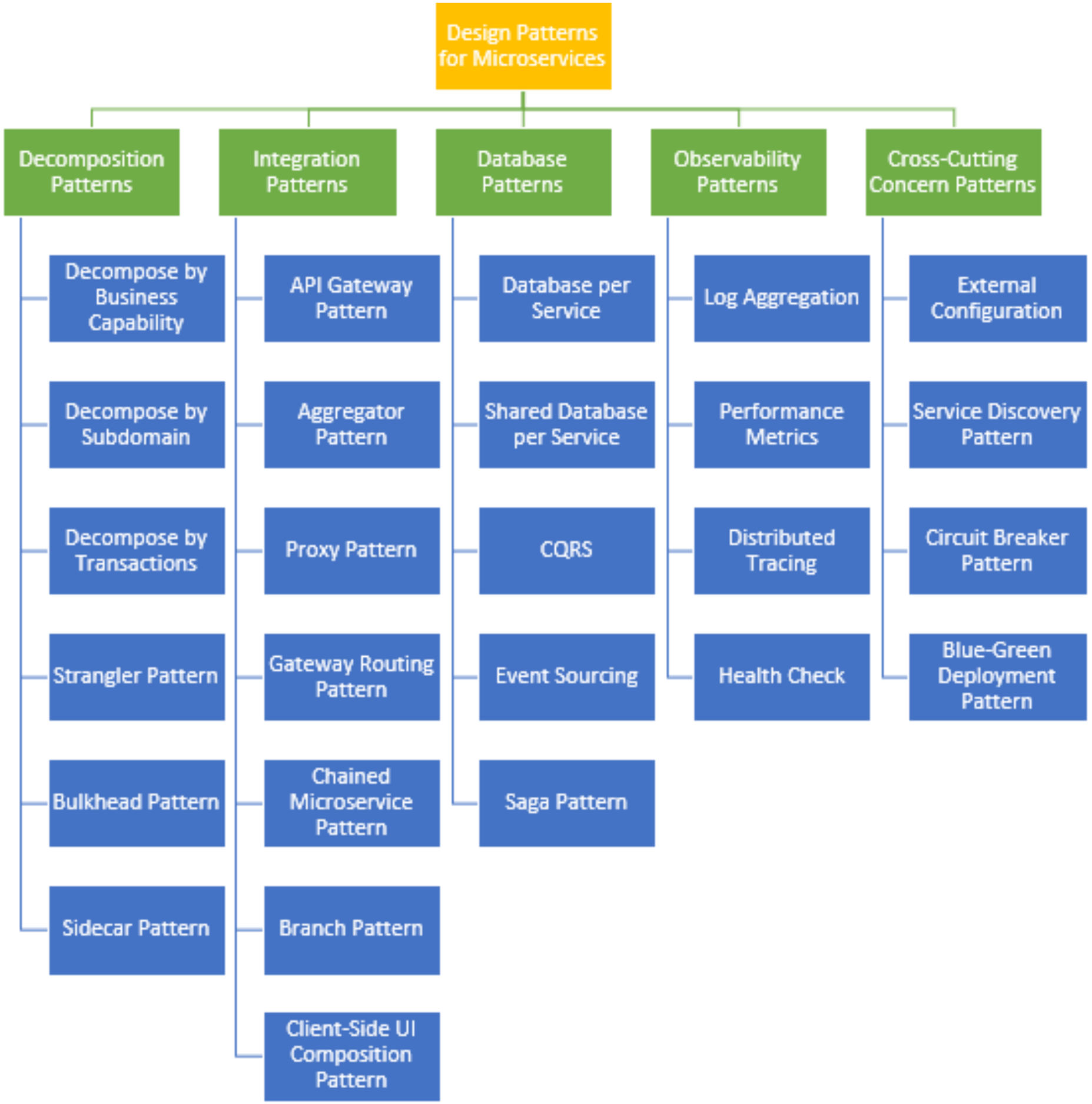
SC#
Data1#

SC#
Data2#

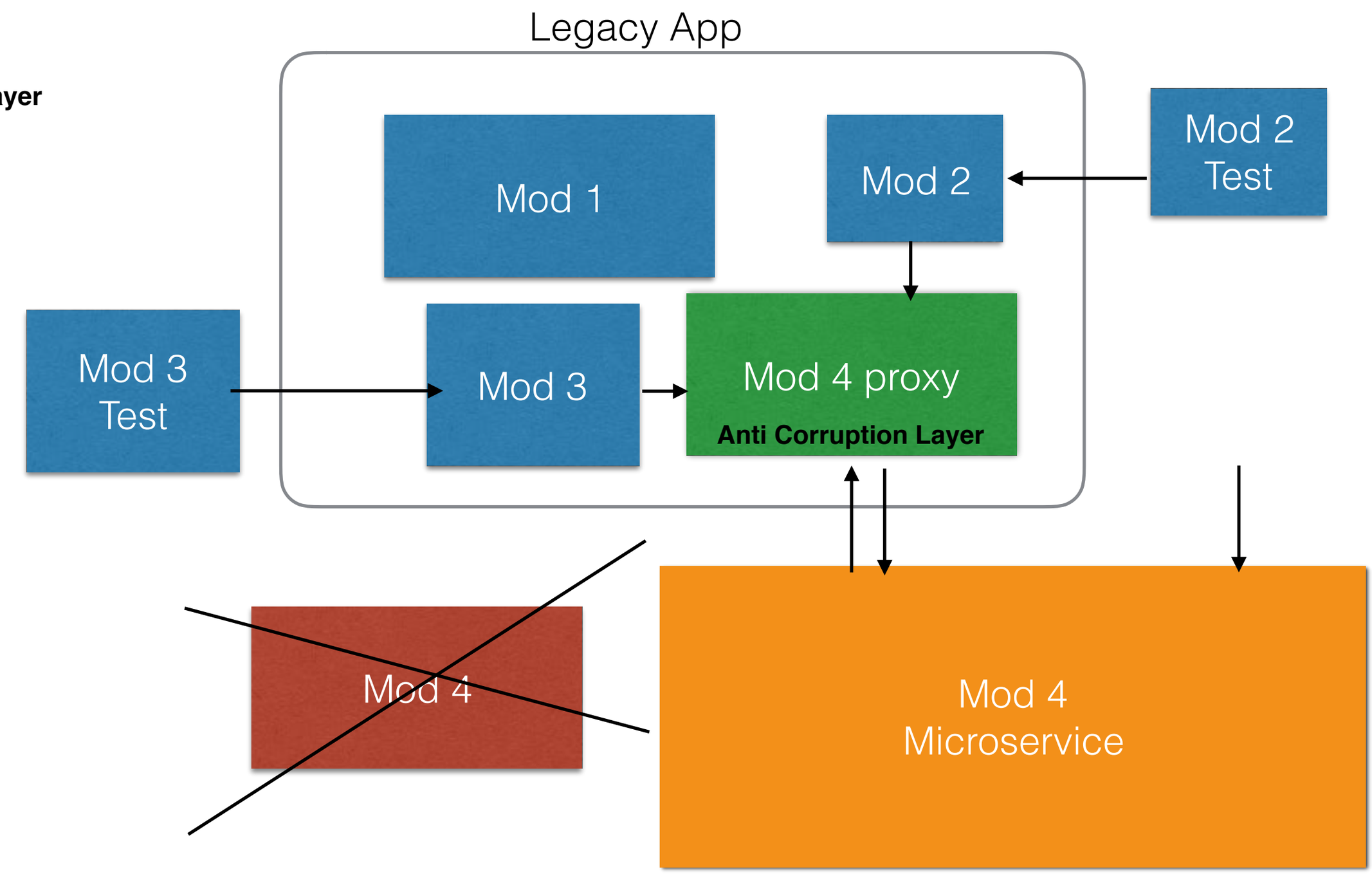
	Monolithic	**Micro**
Fun Requirements	Shared	Not Shared
Source Control	Shared	Not Shared
Build Server	Shared	Not Shared
Database	Shared	Not Shared
Deployment Infra	Shared	Not Shared
Architecture / Technology	Shared	Not Shared
Test Cases	Shared	Not Shared
SCRUM (team) /Sprint	Shared	Not Shared
Platform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

Consistency

Availability

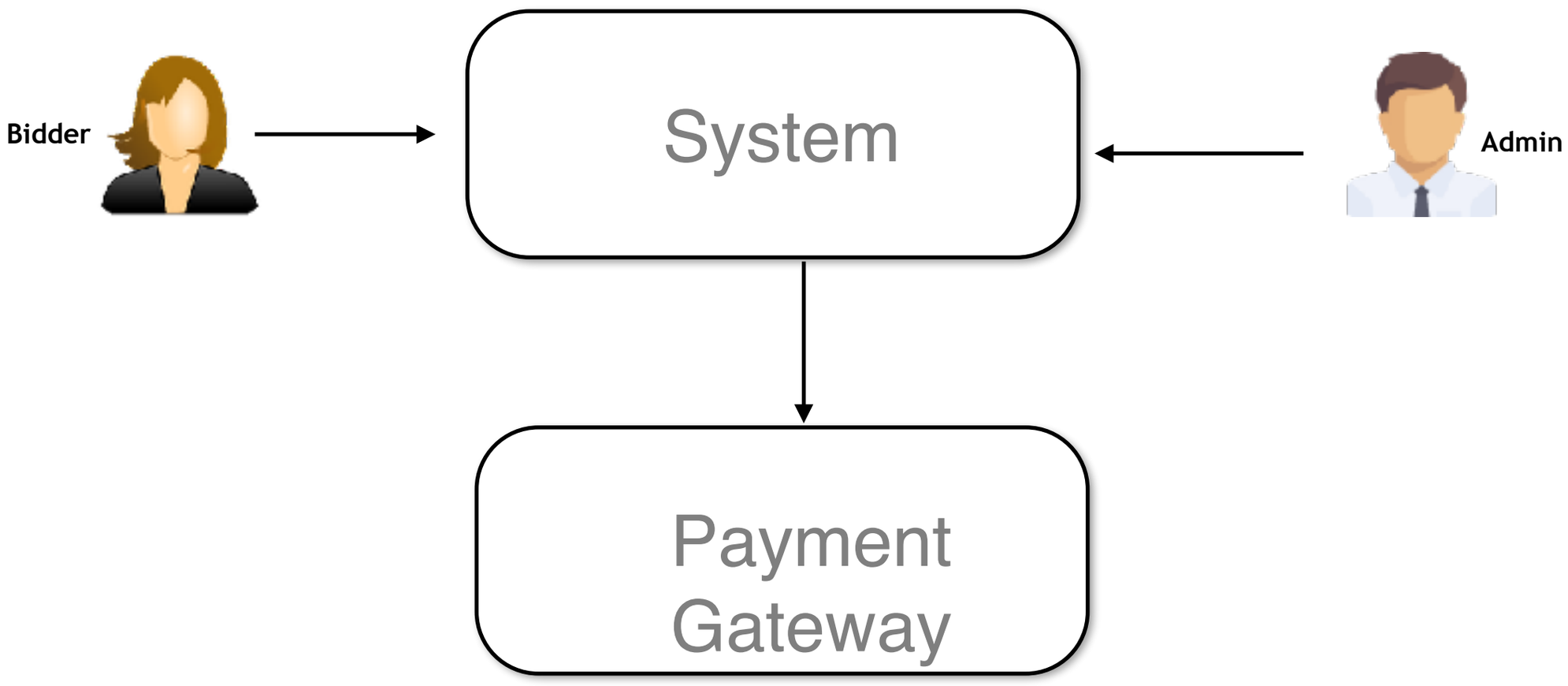
Distributed Database

Bid of the Day

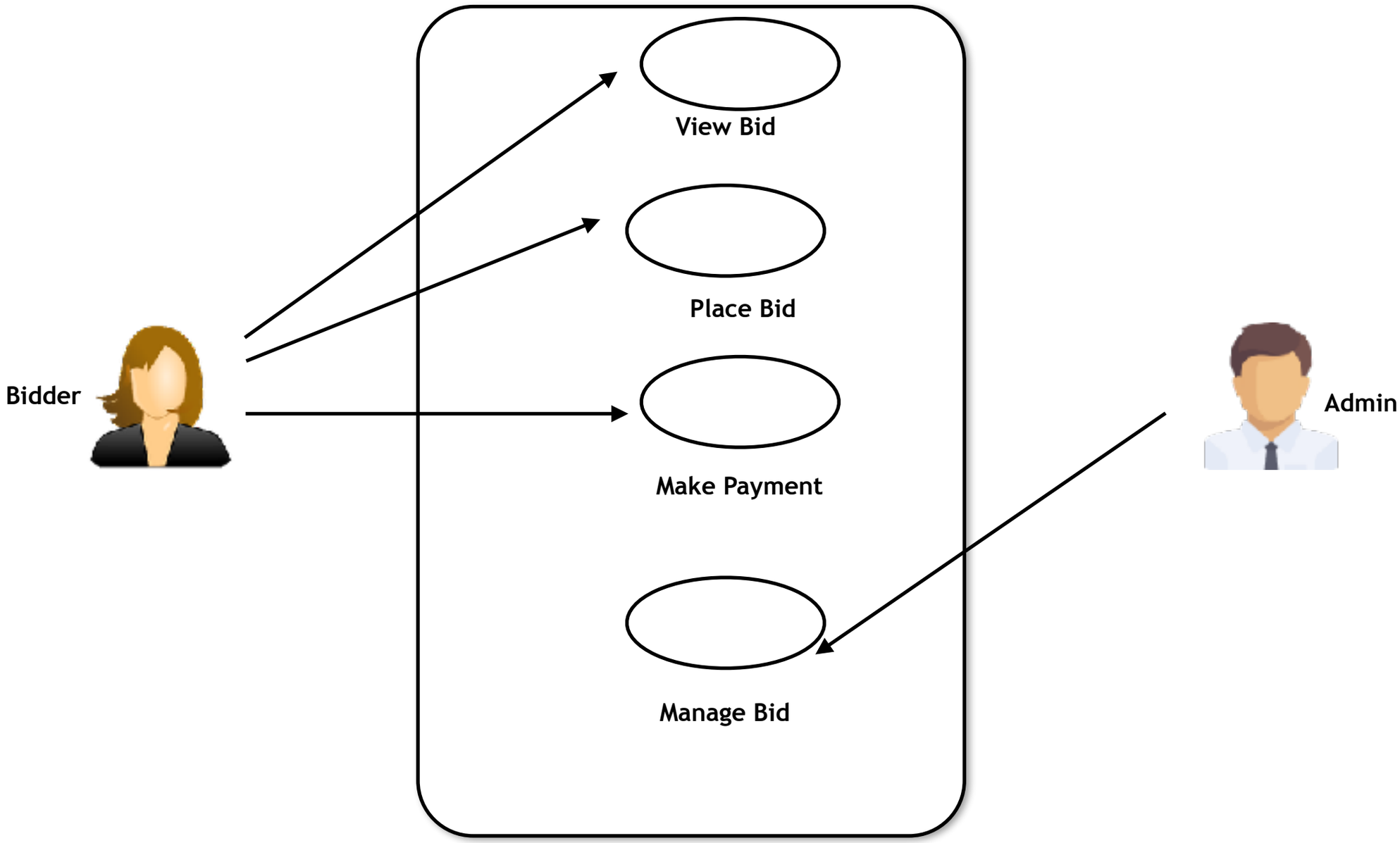
Architectural Requirements

Context View

Black box view



Functional View



Constraints & Assumptions

1. **Use Postgres db**
2. **Use only open source**
3. **GDPR compliance**

Quality

1. As user I want to make payment on the portal after successful bid. The payment is collected is with a PBF 0.0001. (make payment : reliability)

2. 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)

3.

Utility tree

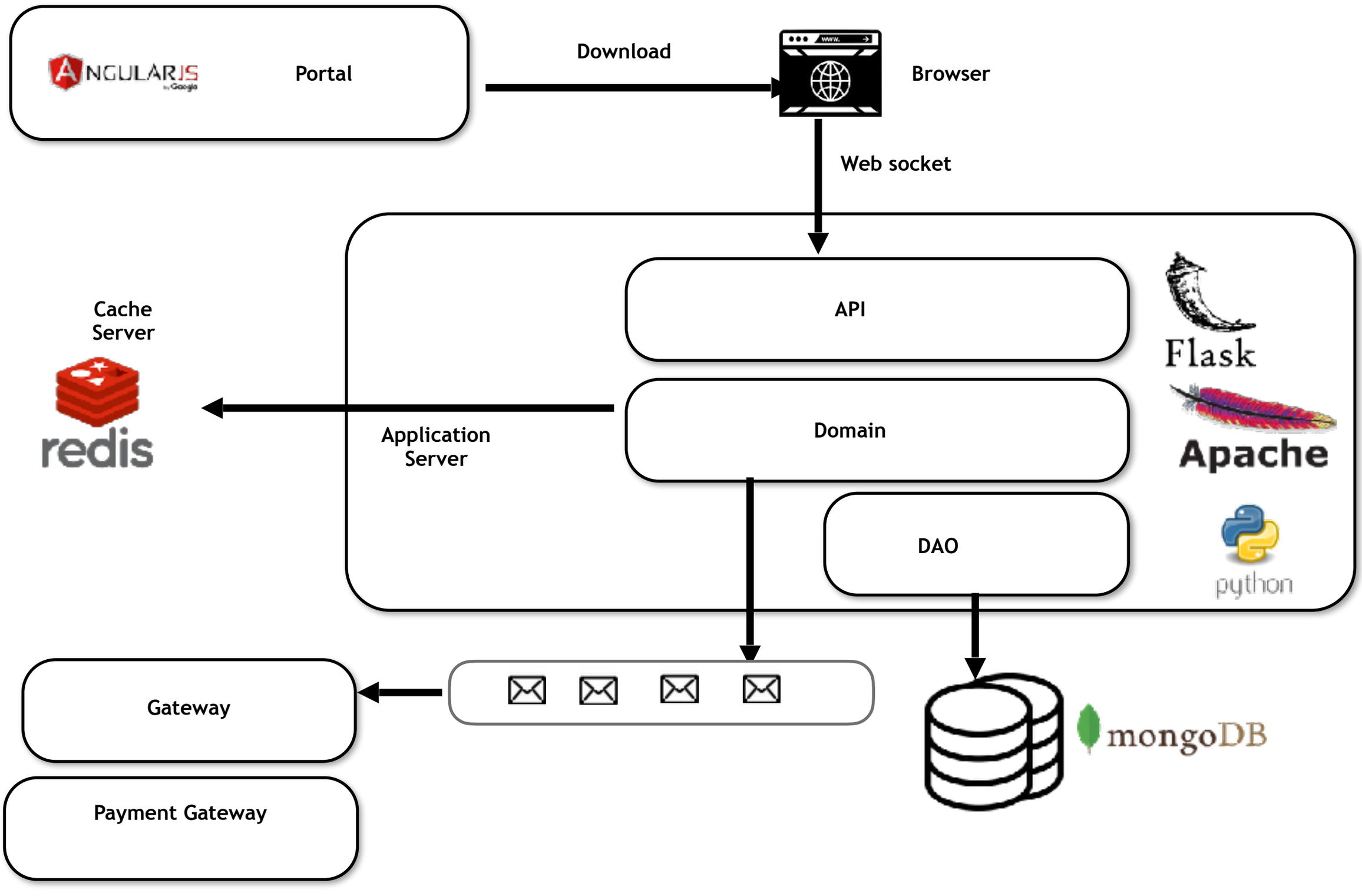
reliability

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

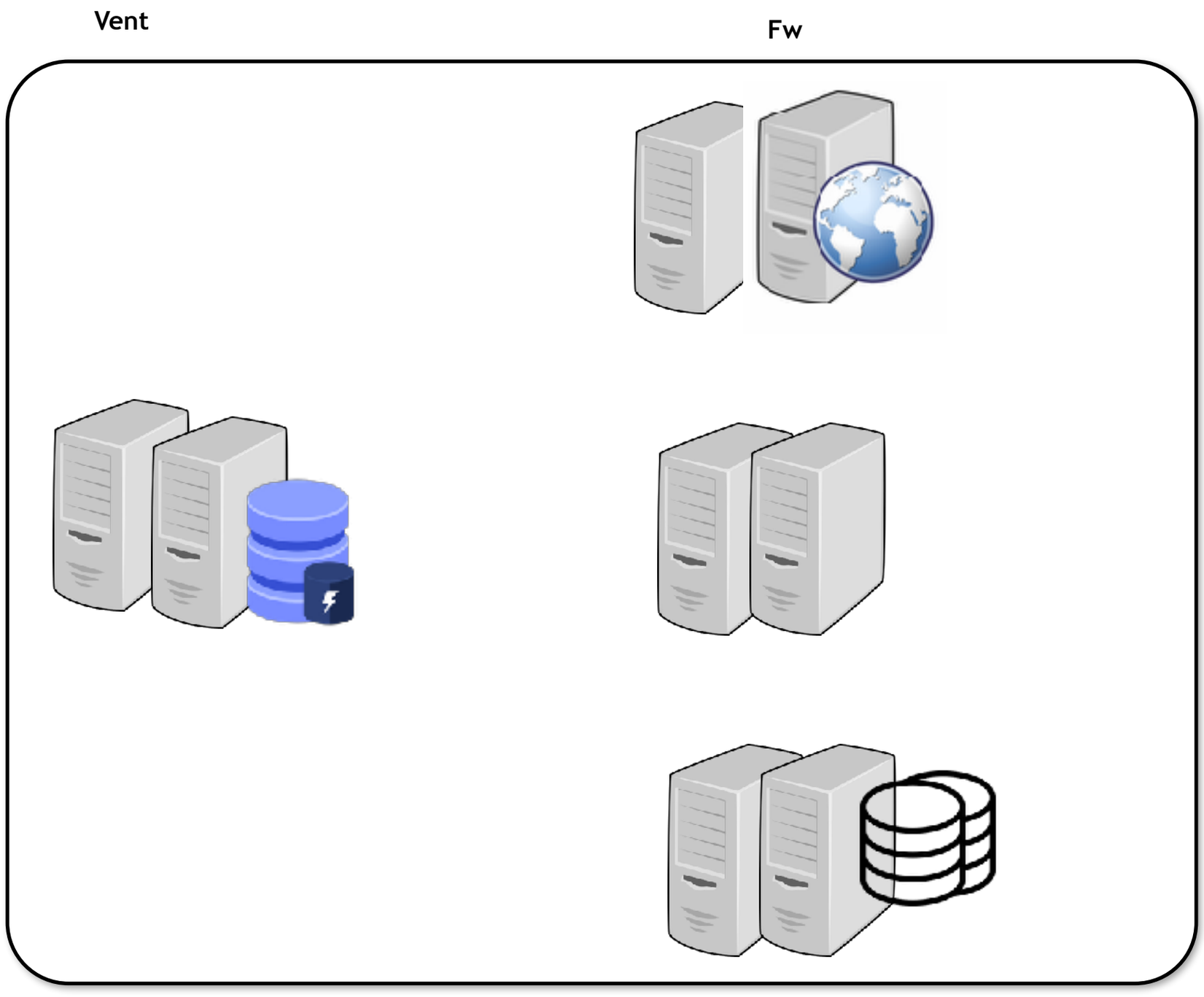
Logical view

White box view

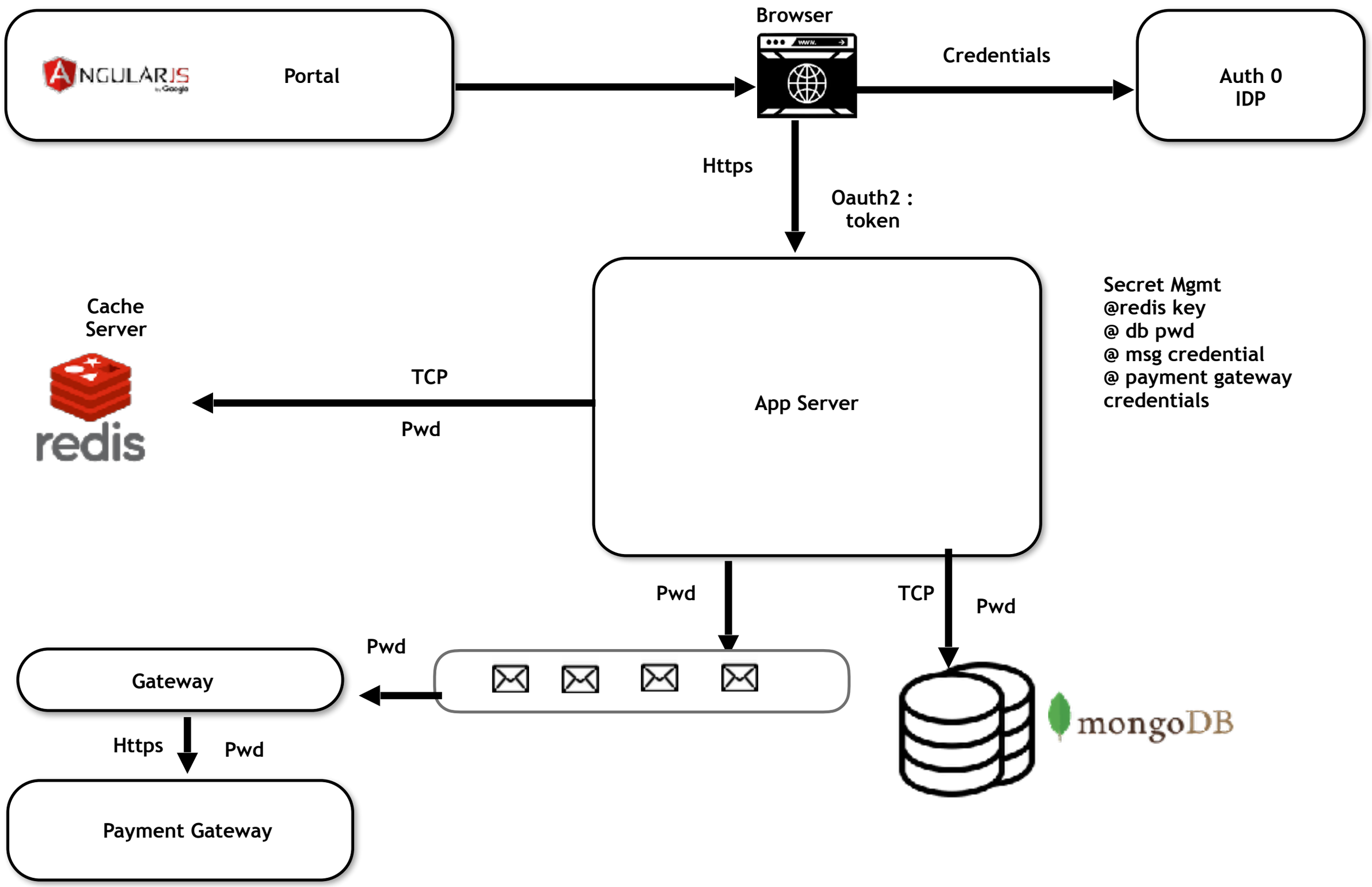


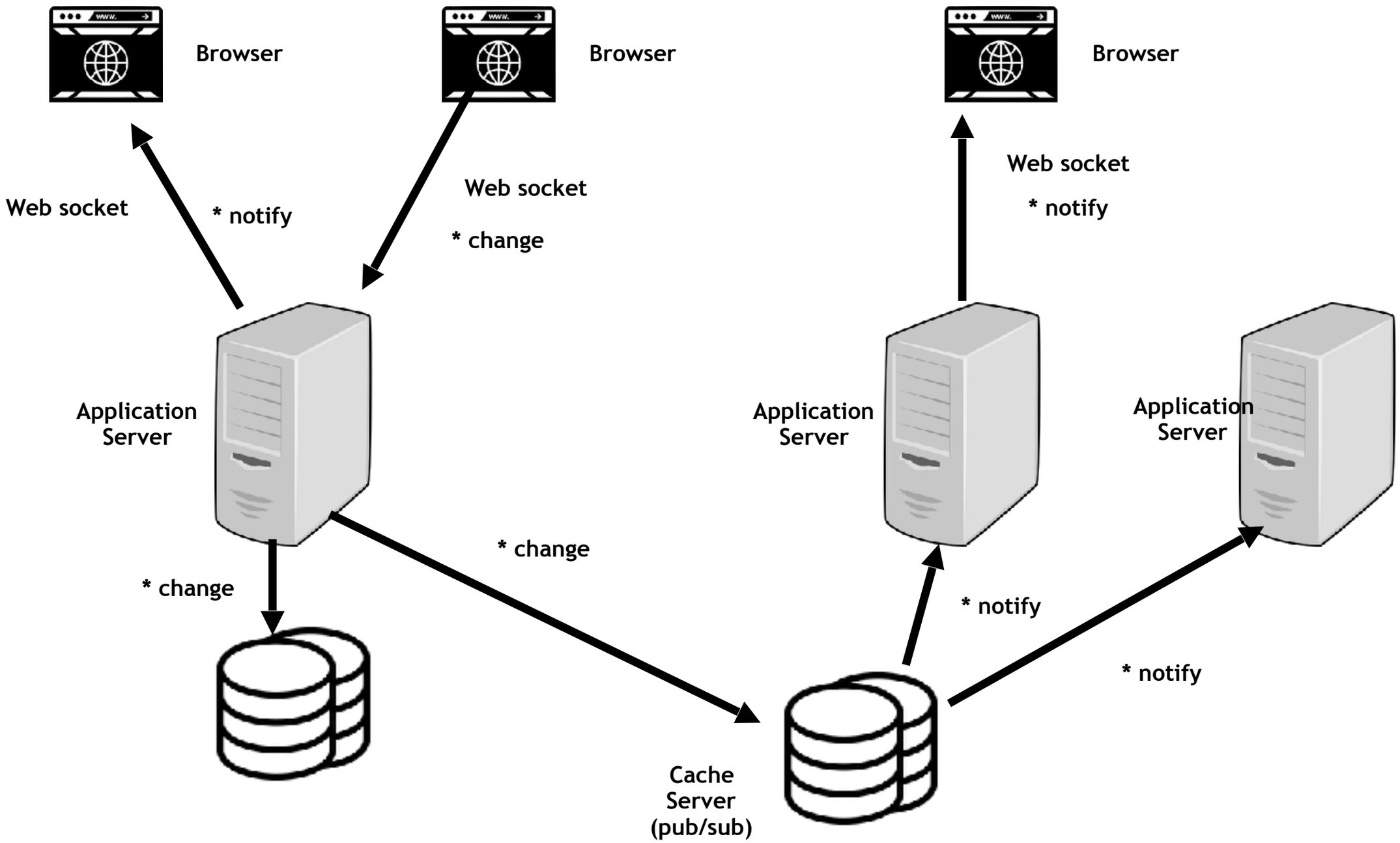
Infrastructure View

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