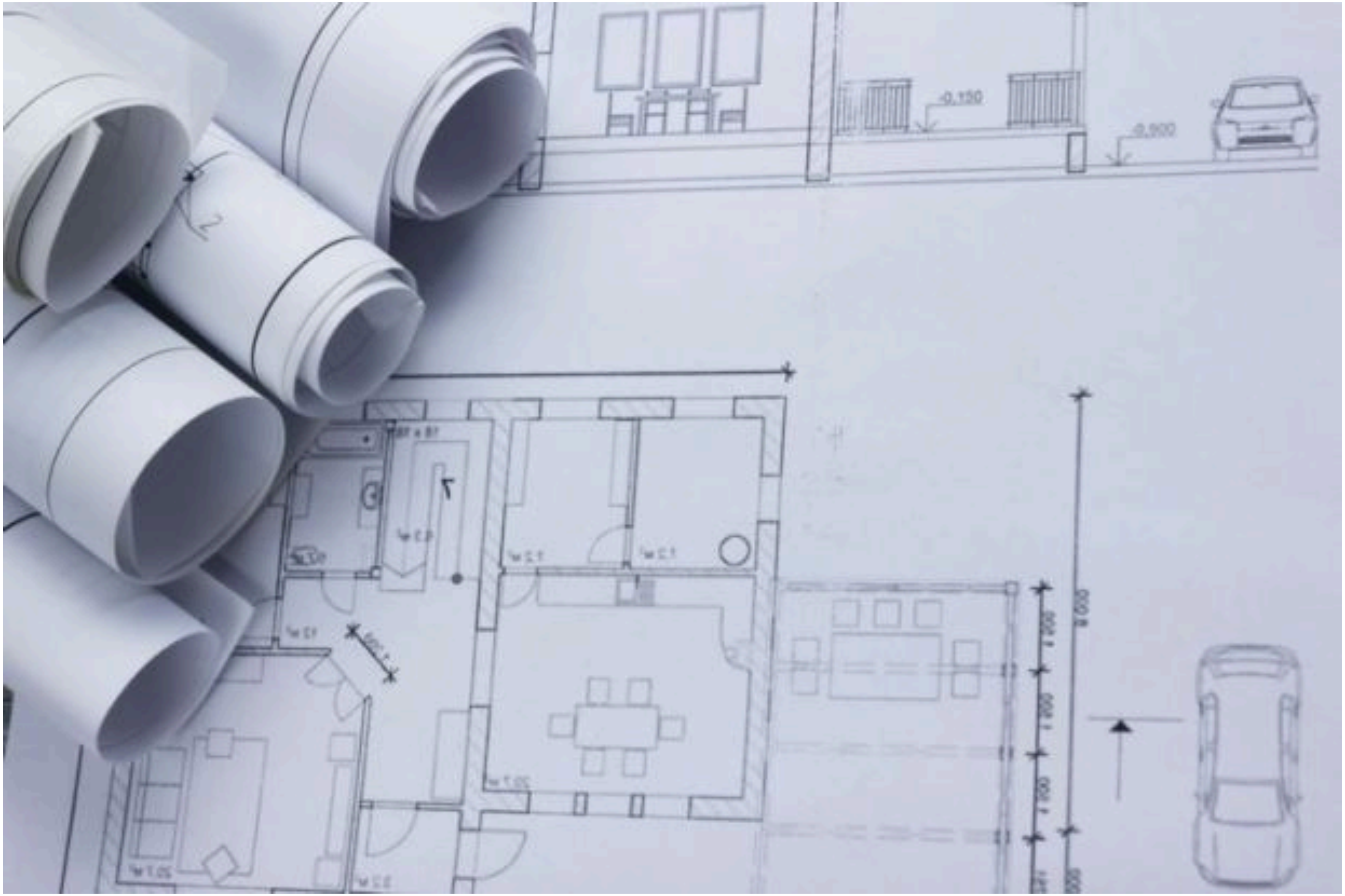


Software Architecture

A LITTLE BIT ABOUT me...

- Skan.ai - chief Architect
- Ai.robotics - chief Architect
- Genpact - solution Architect
- Welldoc - chief Architect
- Microsoft
- Mercedes
- Siemens
- Honeywell

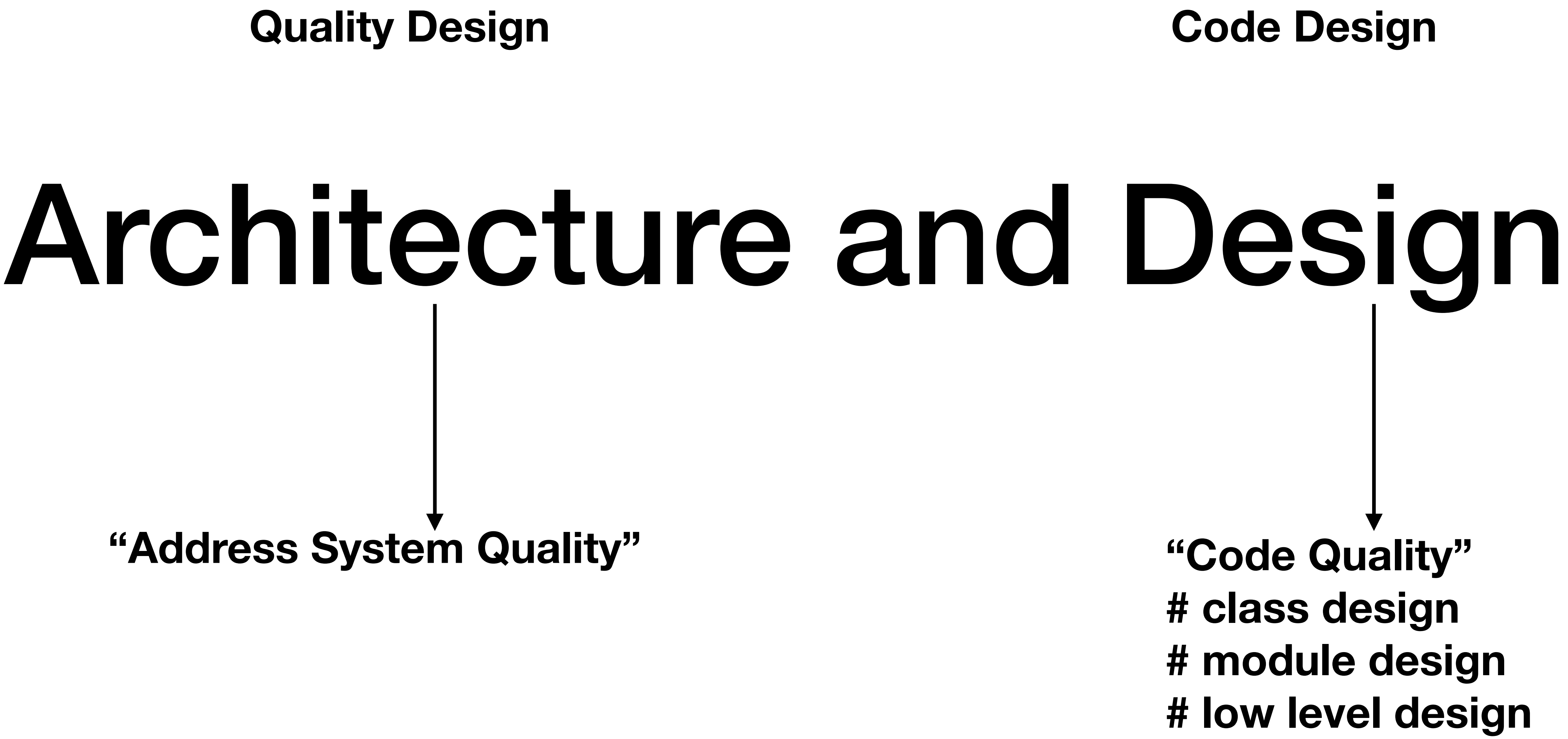
Mubarak

- 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
-



Quality Design

Code Design

Architecture and Design

"Address System Quality"

"Code Quality"
# class design
# module design
# low level design

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

Quality attributes Model

# SEI
# McCal
# Bohem
# IEEE
#ISO

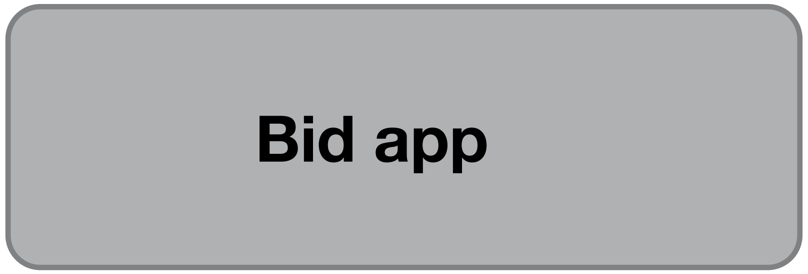
Measure (scale)

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

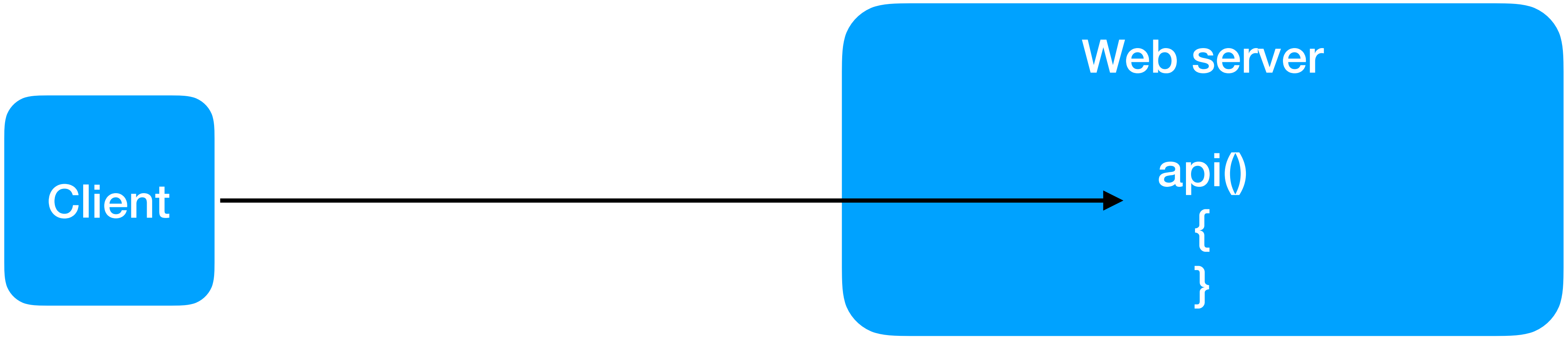
Bid of the day



Bid app

< 1 sec

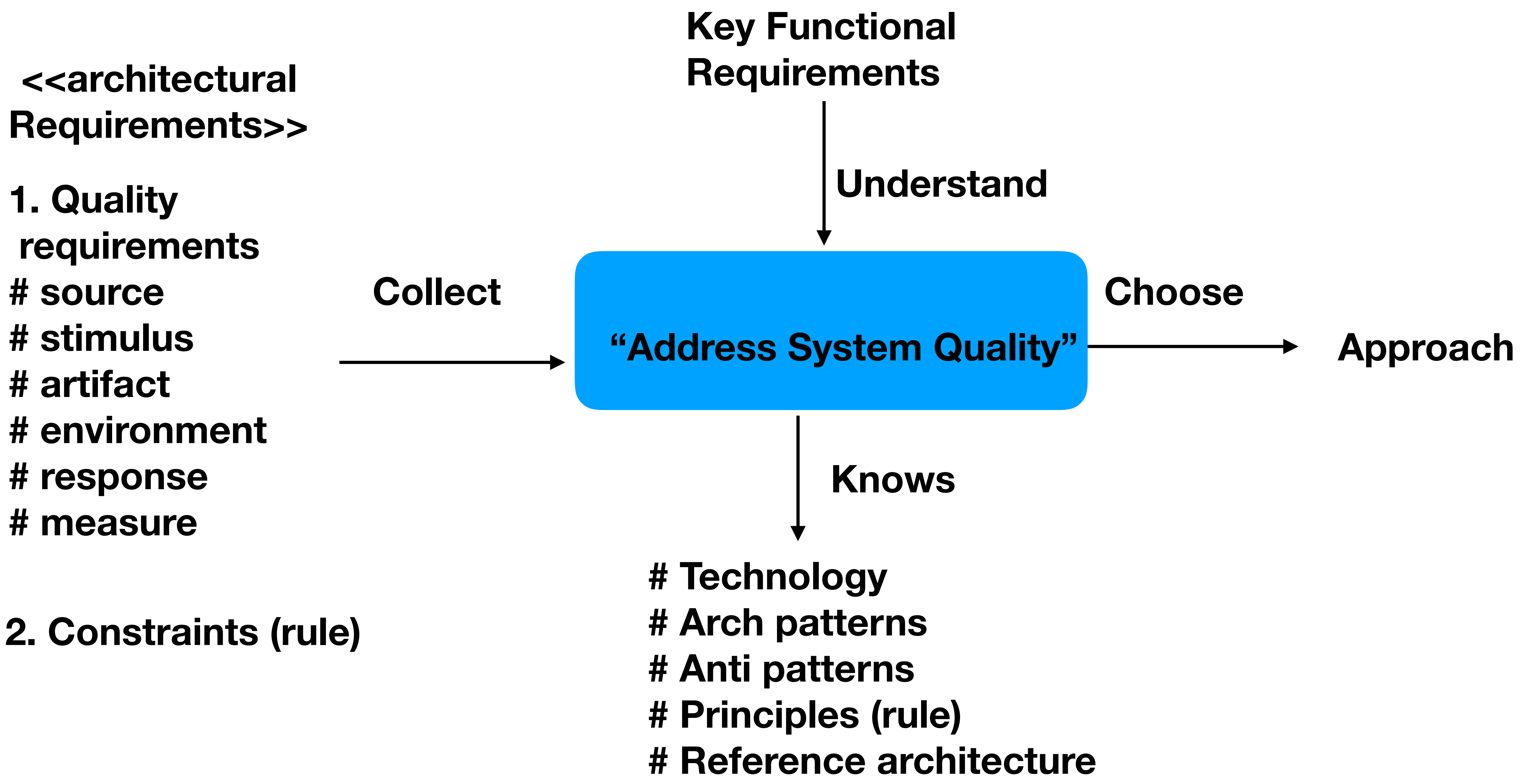
< 3 sec



Client

Web server

```
api()
  {
  }
```



<<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

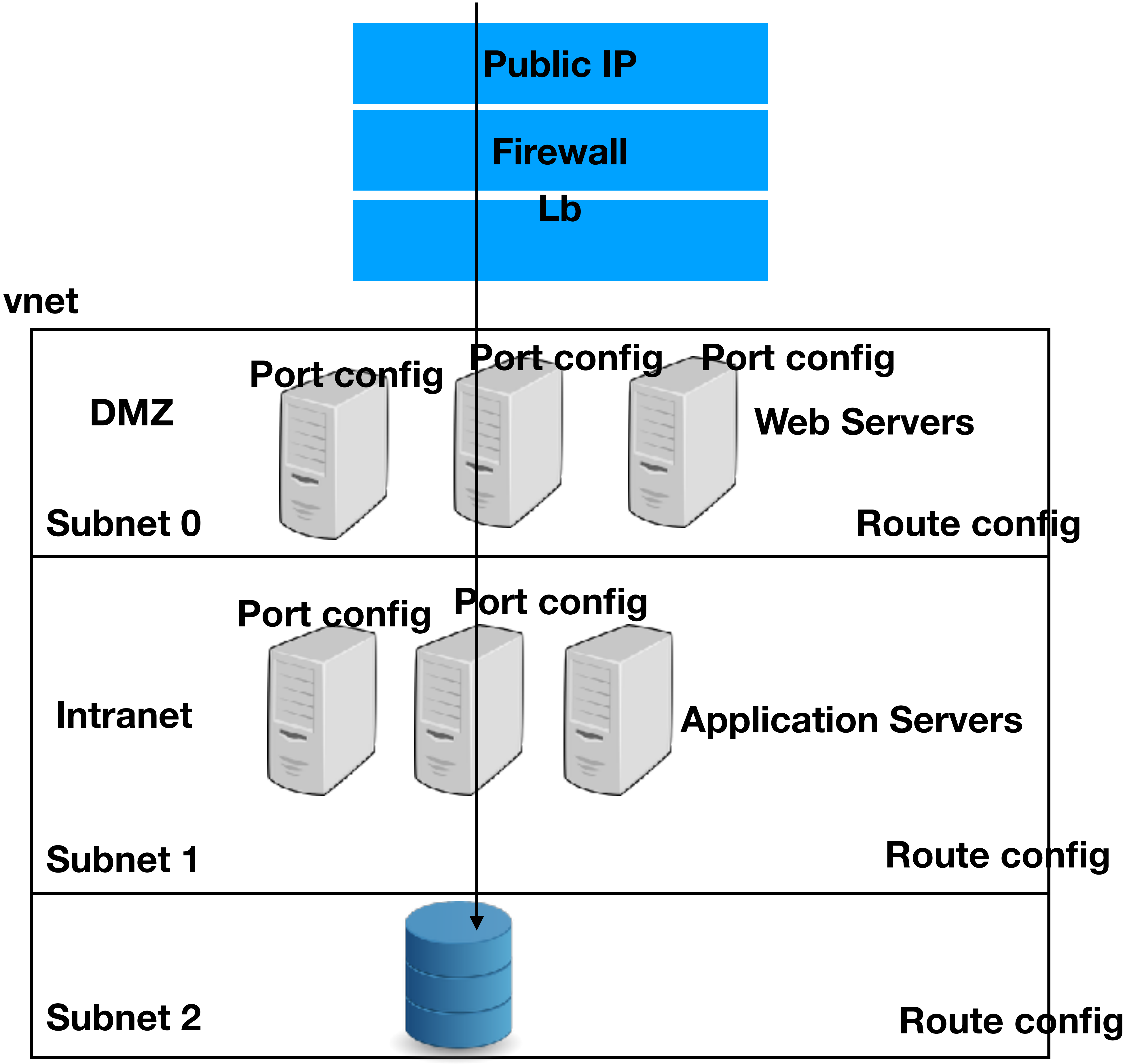
Db call <— 45,00,000 cpu cycles
Remote call (rest) <— 30,00,000 cpu cycles

“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

DMZ

Port config

Port config

Port config

Web Servers

Subnet 0

Route config

Port config

Port config

Intranet

Application Servers

Subnet 1

Route config

Subnet 2

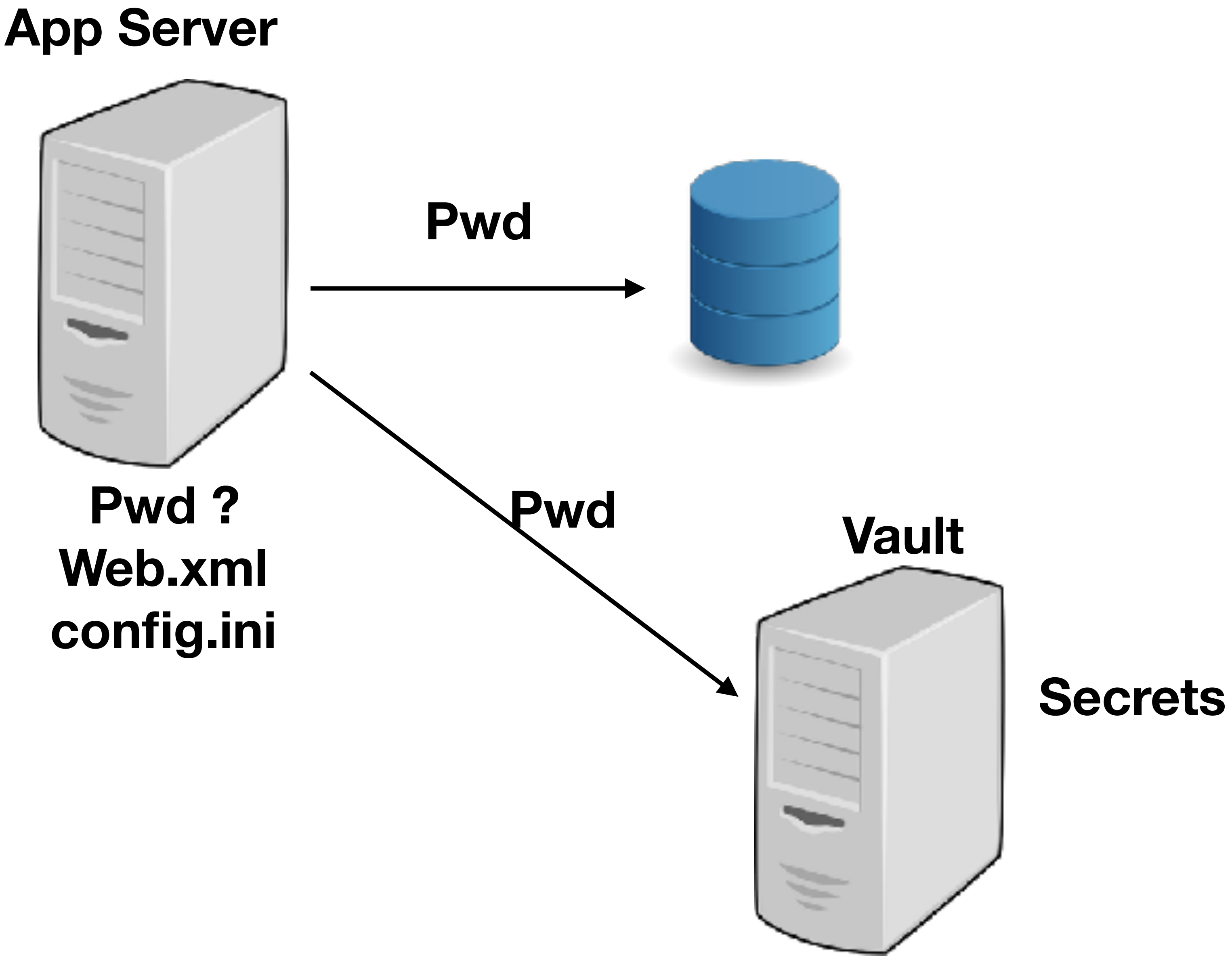
Route config

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

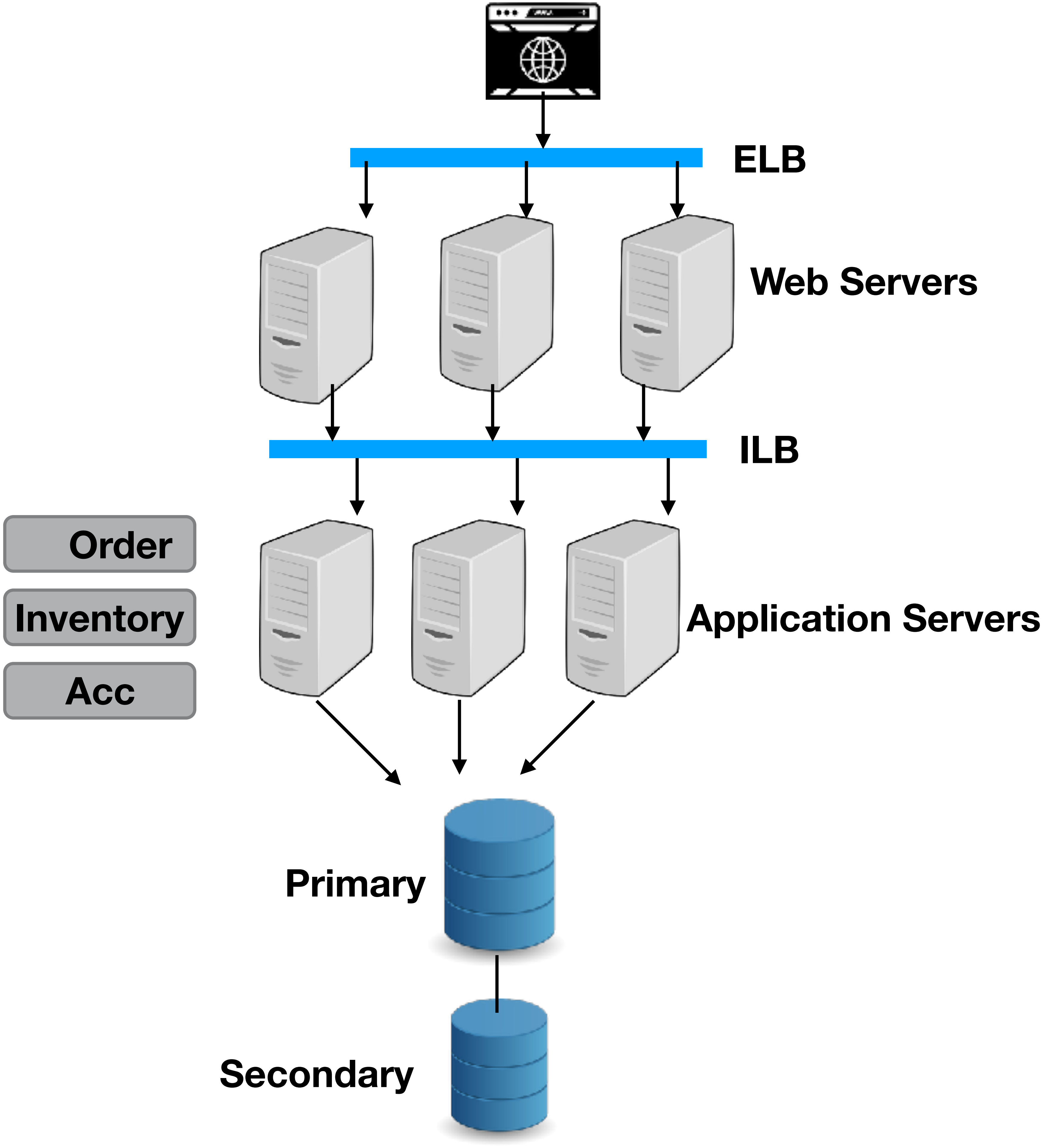
Pwd

Pwd ?
Web.xml
config.ini

Pwd

Vault

Secrets



ELB

Web Servers

ILB

Order

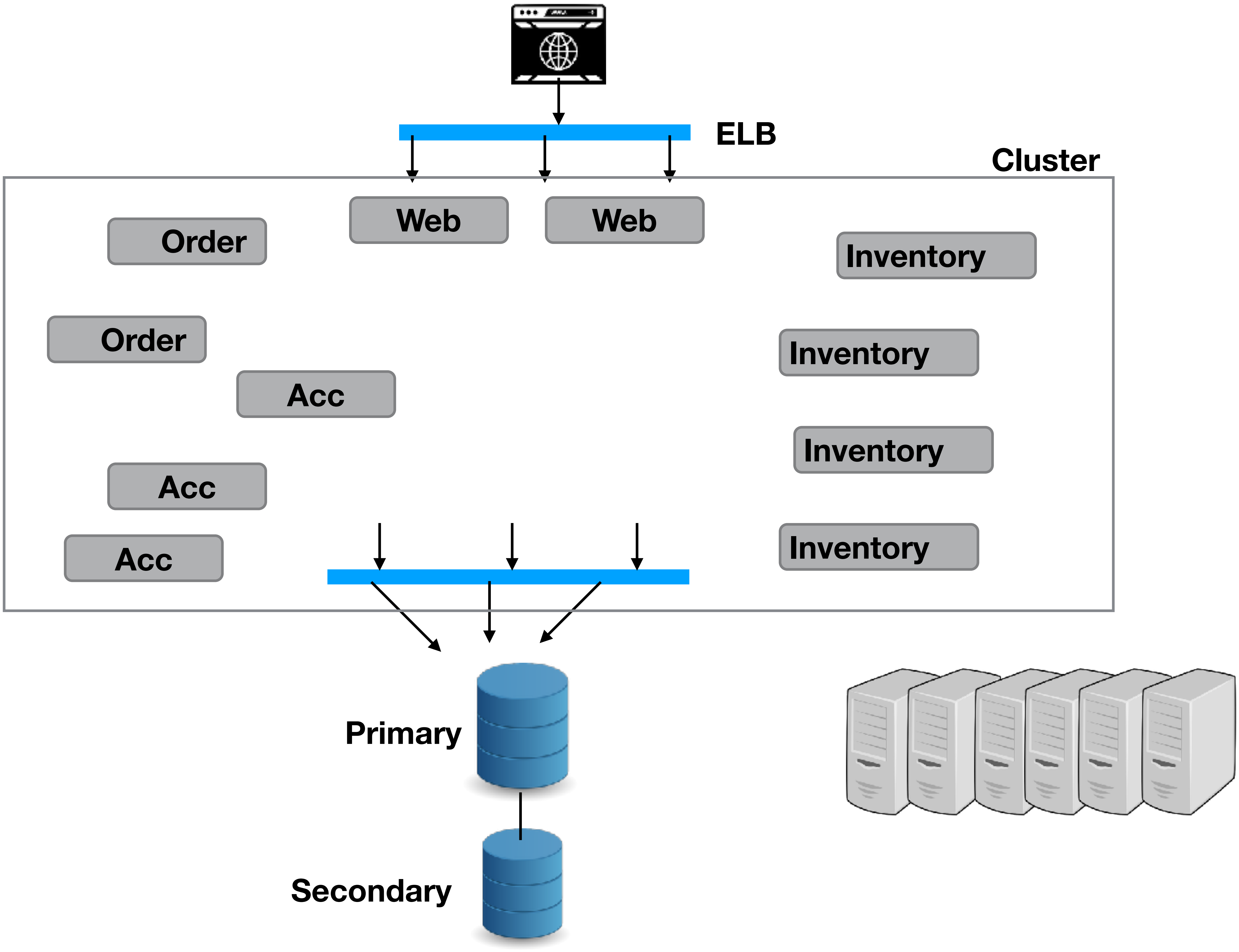
Inventory

Acc

Application Servers

Primary

Secondary



ELB

Cluster

Order

Web

Web

Inventory

Order

Inventory

Acc

Inventory

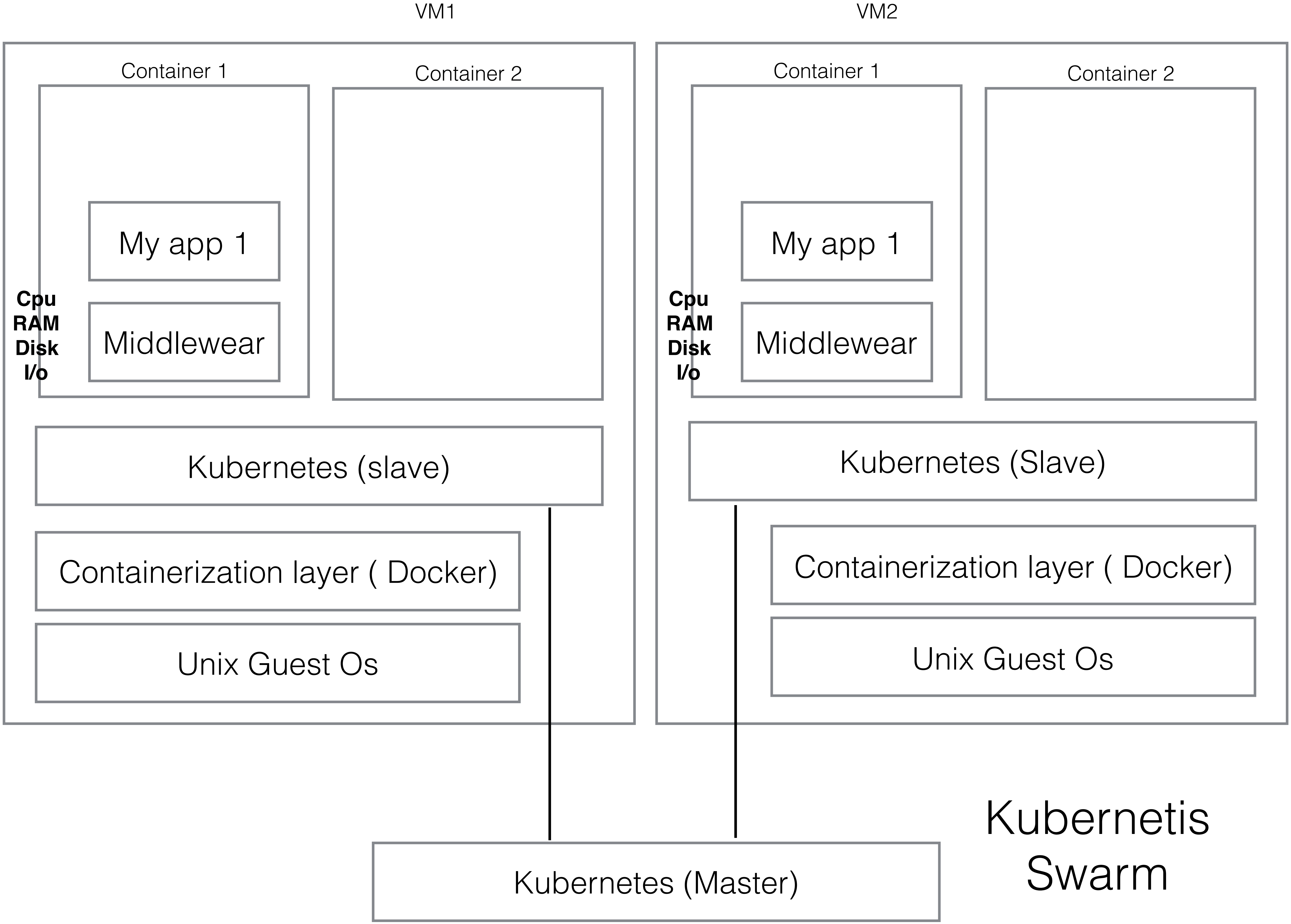
Acc

Acc

Inventory

Primary

Secondary


VM1

VM2

Container 1

Container 2

Container 1

Container 2

My app 1

Middlewear

Cpu
RAM
Disk
I/o

My app 1

Middlewear

Cpu
RAM
Disk
I/o

Kubernetes (slave)

Kubernetes (Slave)

Containerization layer (Docker)

Containerization layer (Docker)

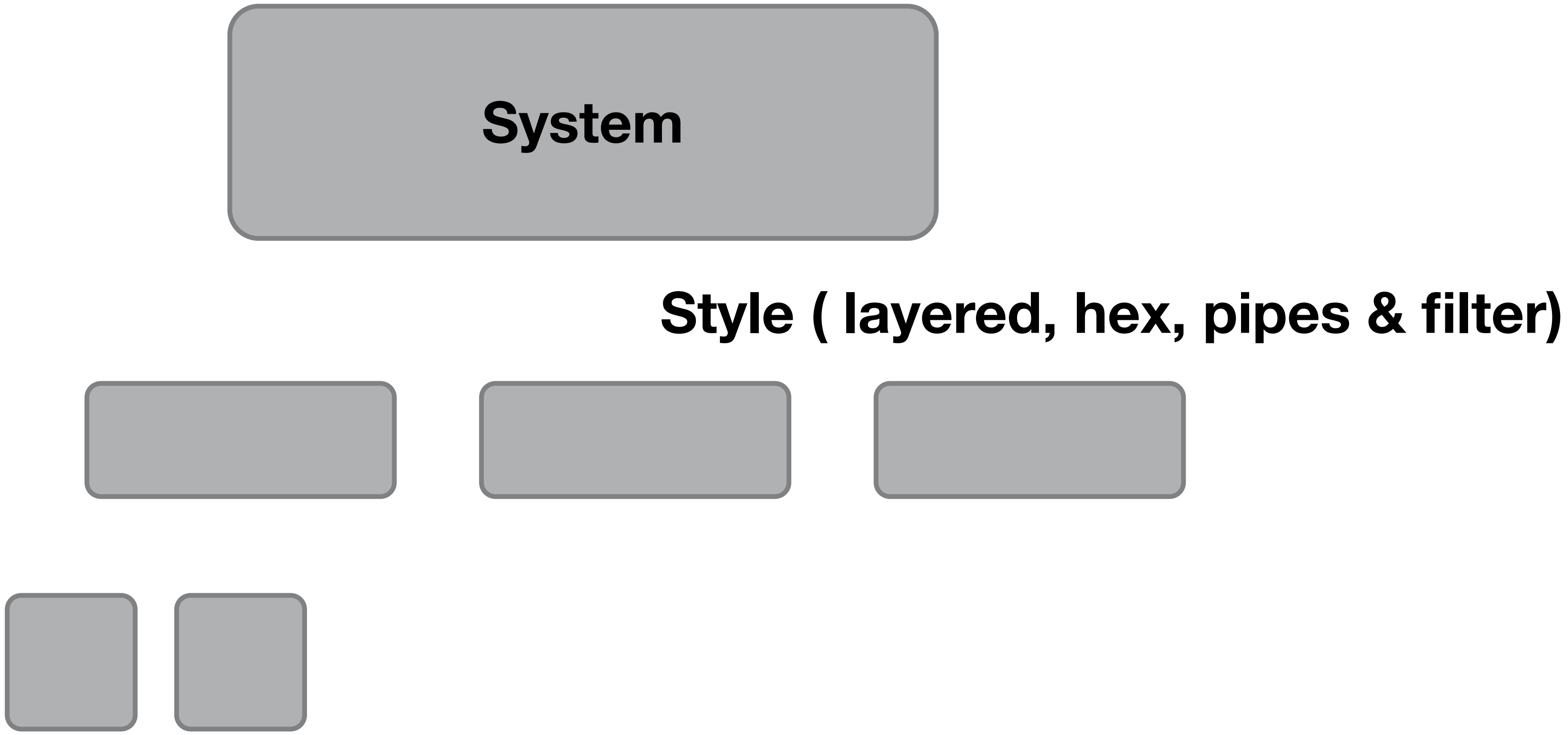
Unix Guest Os

Unix Guest Os

Kubernetes (Master)

Kubernetis
Swarm

# decompose system

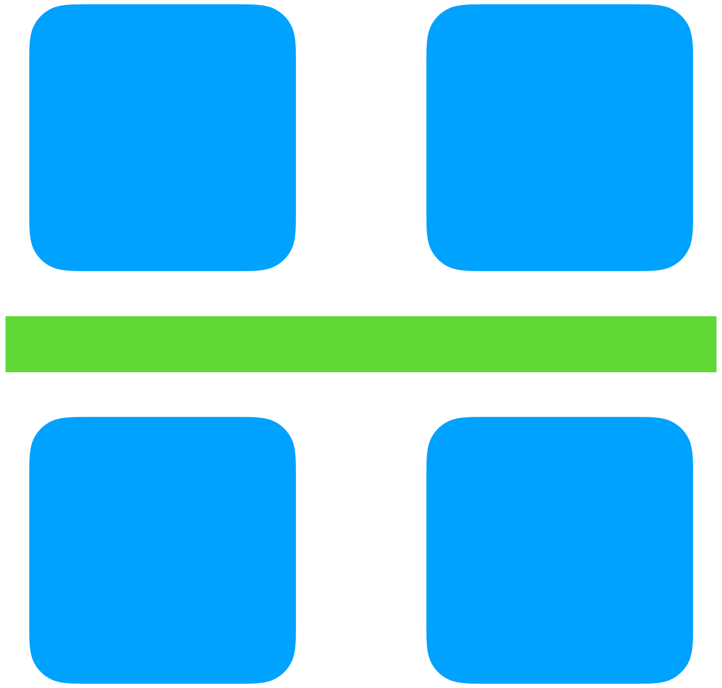


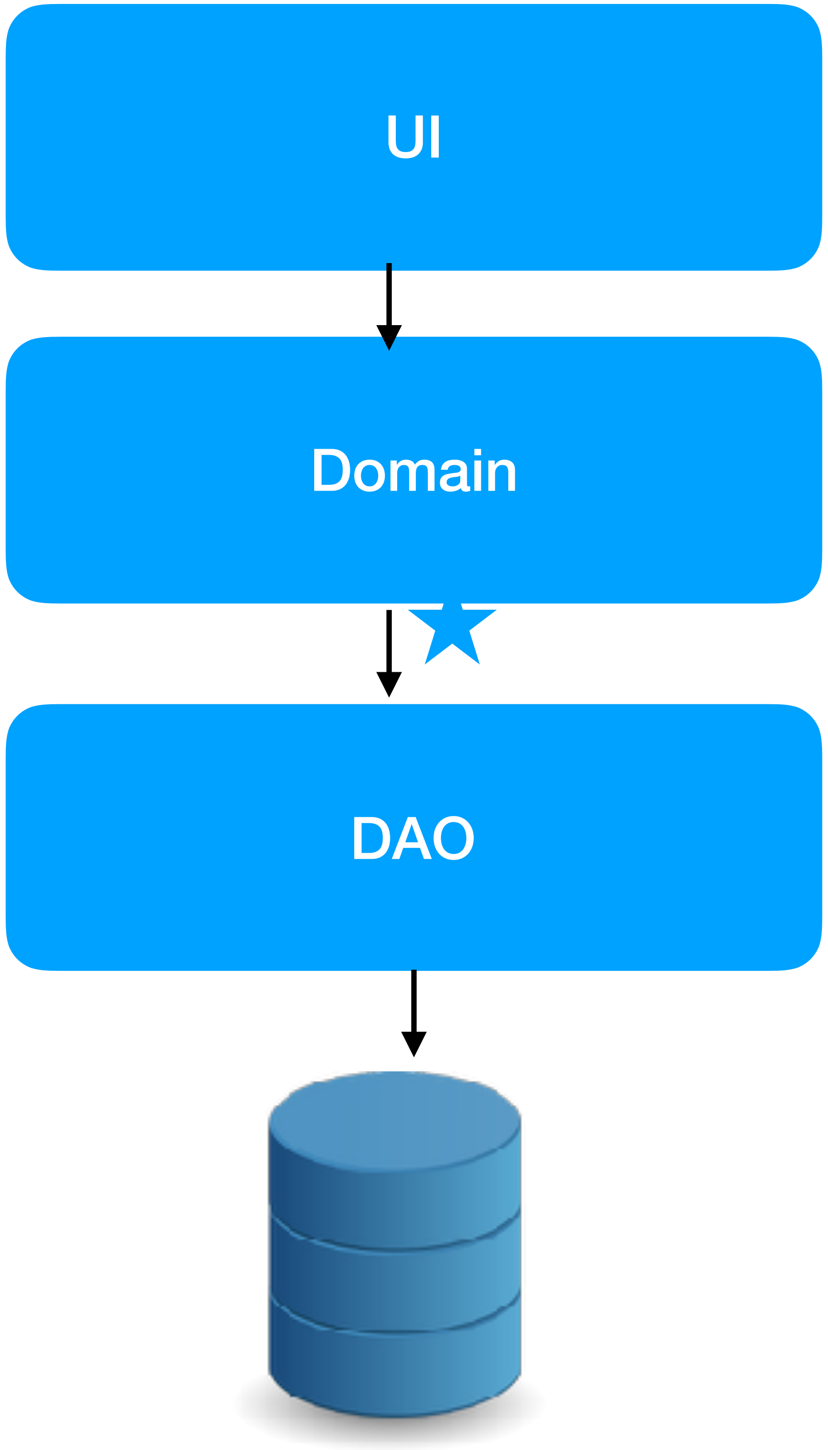
System

Style (layered, hex, pipes & filter)

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	

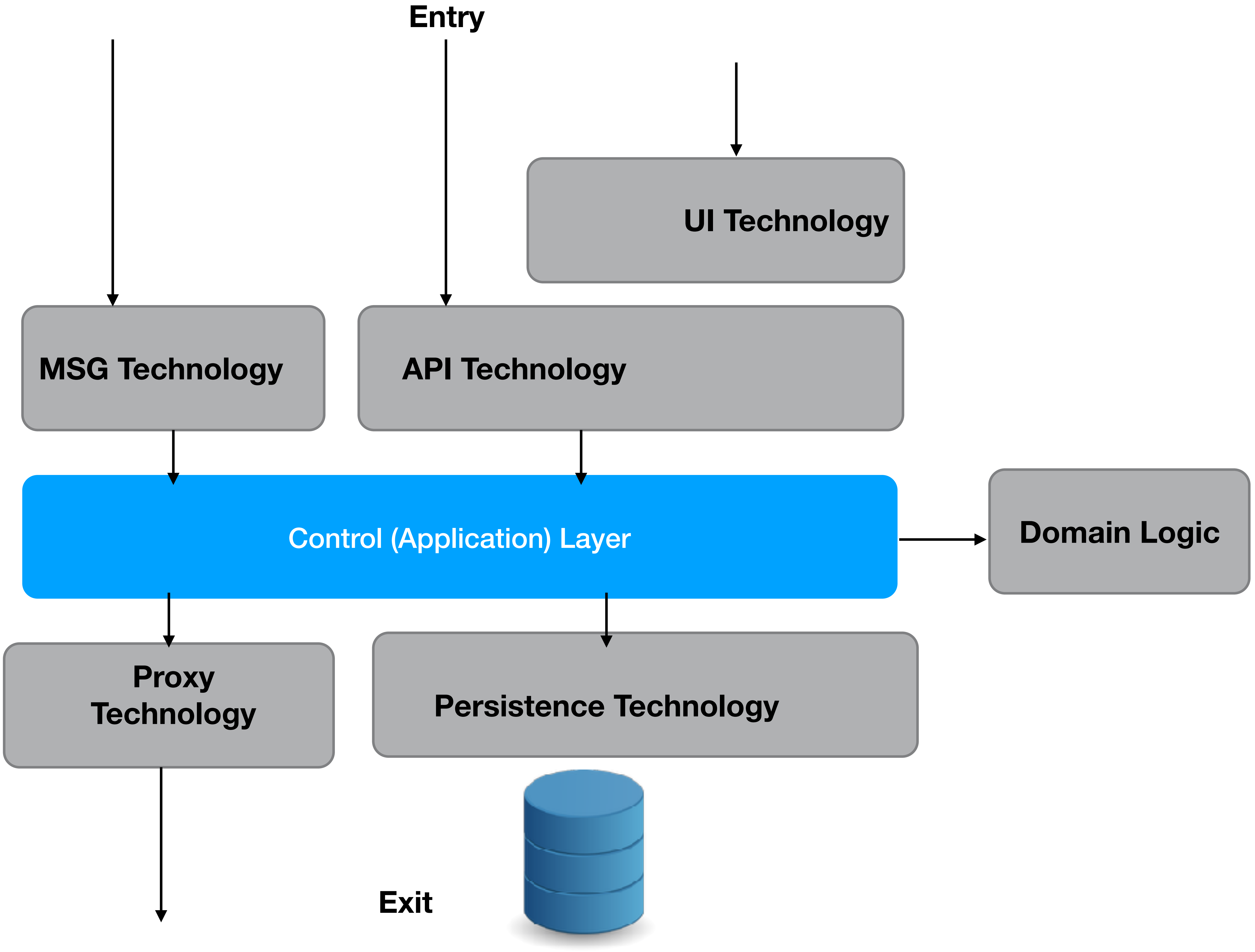




UI

Domain

DAO



Entry

UI Technology

MSG Technology

API Technology

Control (Application) Layer

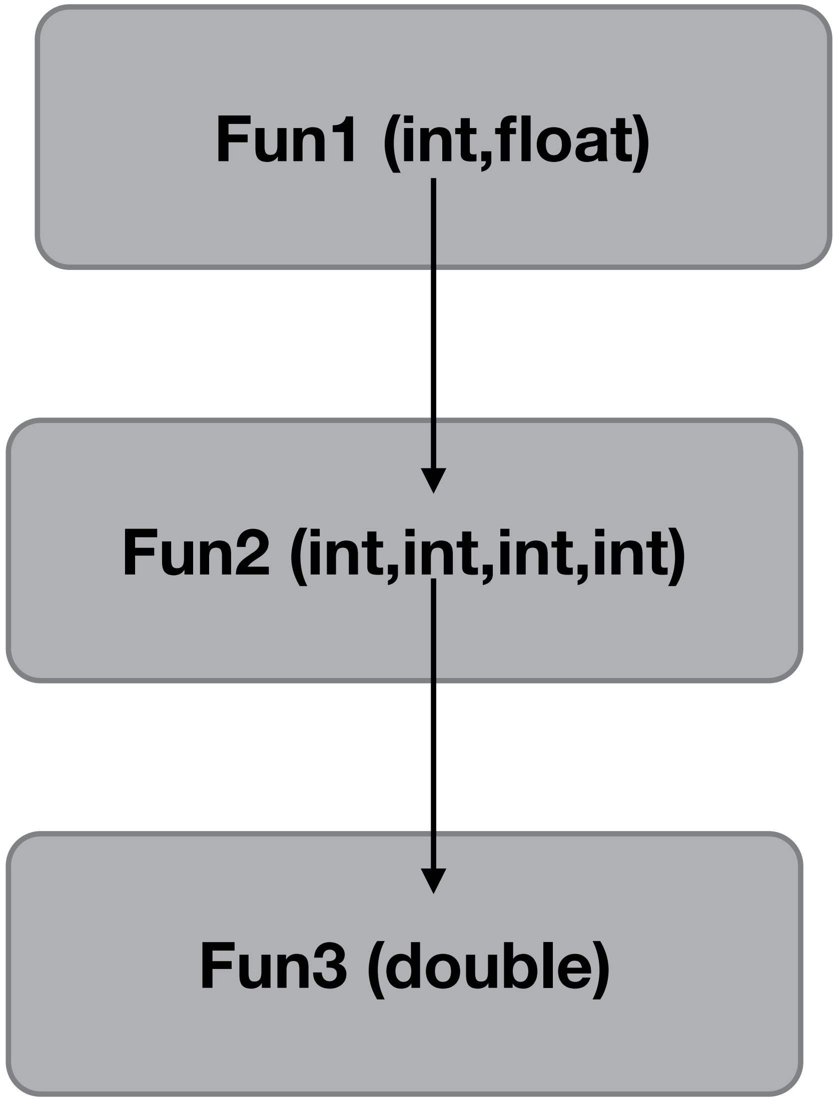
Domain Logic

Proxy Technology

Persistence Technology

Exit

Layered

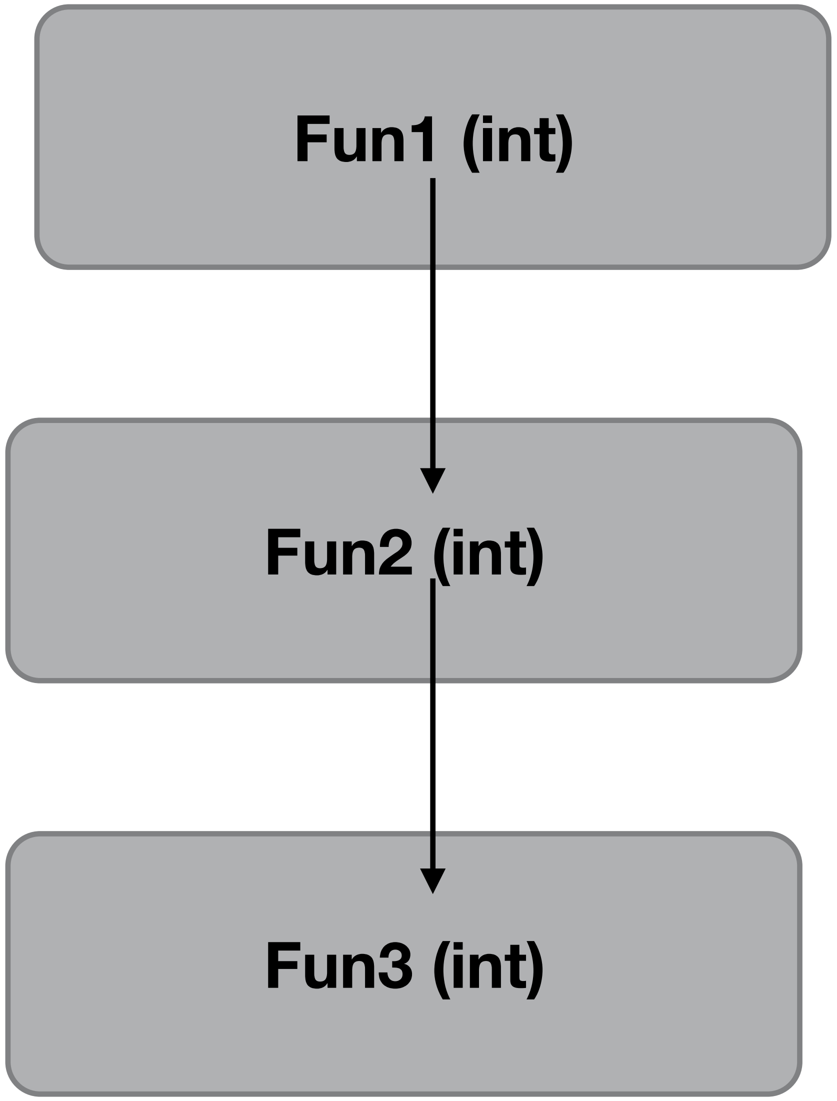


Fun1 (int,float)

Fun2 (int,int,int,int)

Fun3 (double)

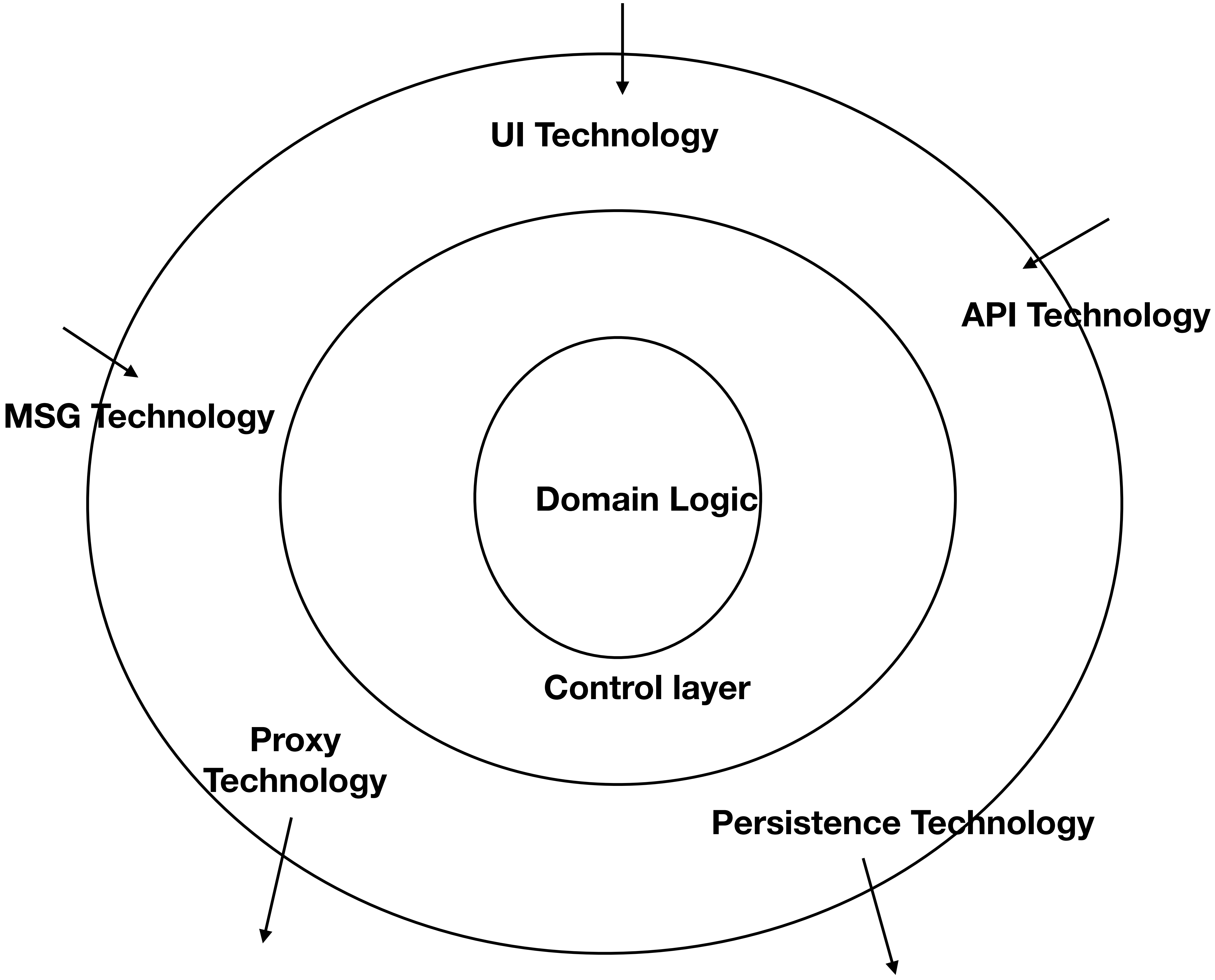
Pipes and filter



Fun1 (int)

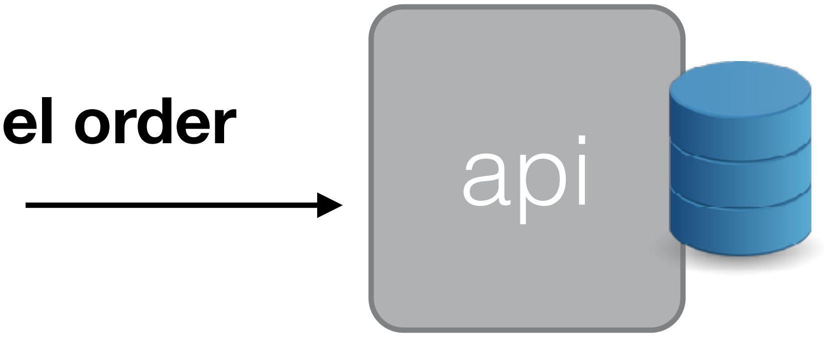
Fun2 (int)

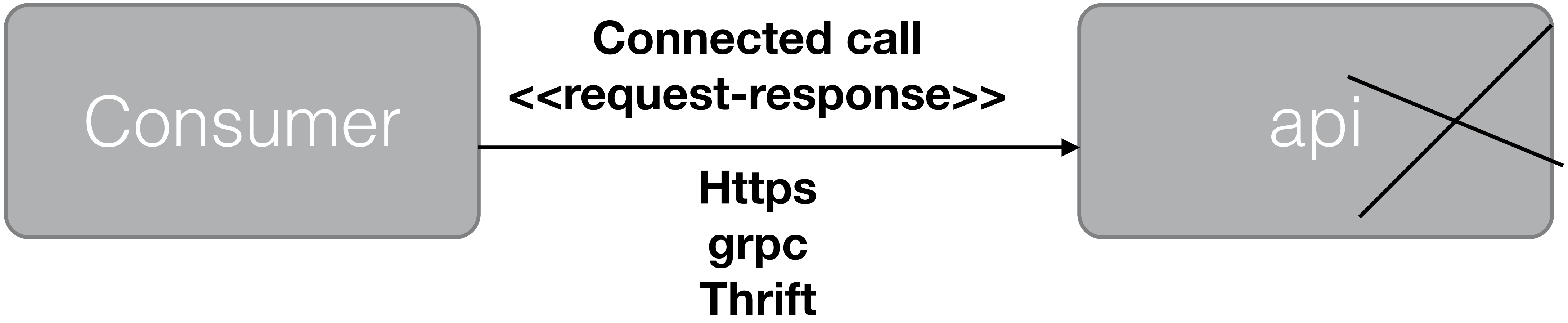
Fun3 (int)


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

Create order

Cancel order



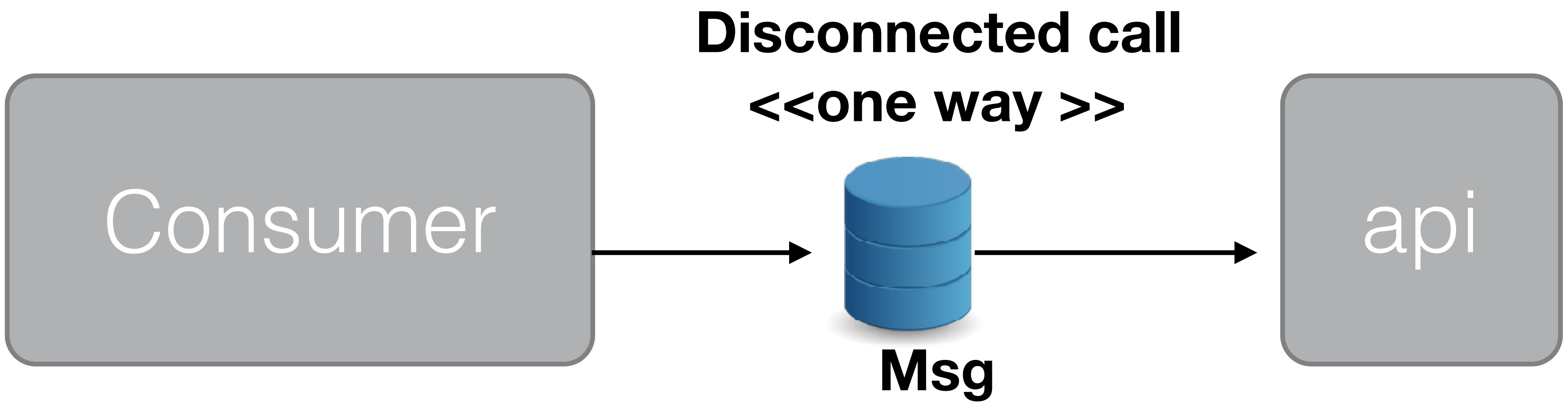


Consumer

Connected call
<<request-response>>

Https
grpc
Thrift

api



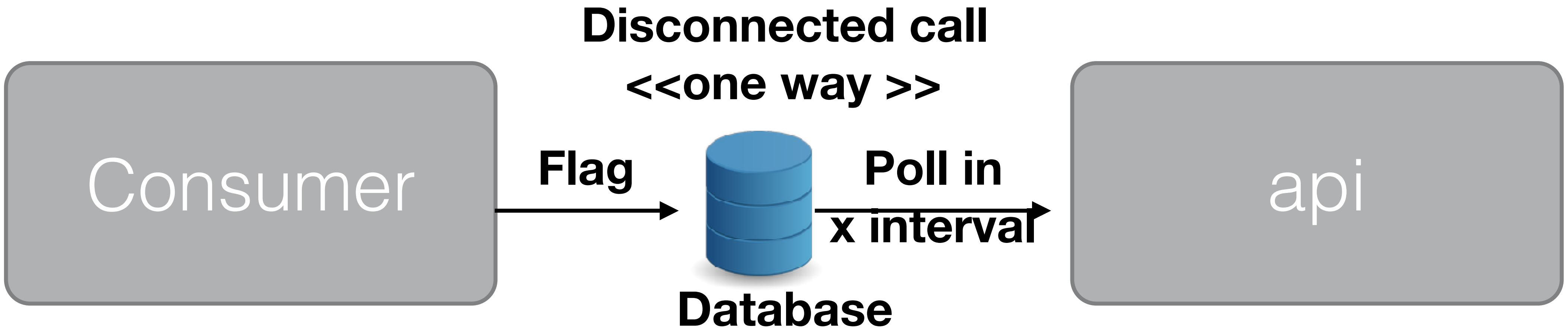
Consumer

Disconnected call
<<one way >>

Msg

api

```
while(true)
{
   msg = GetMessage();
   Do domain logic
   ack(msg)
}
```



Consumer

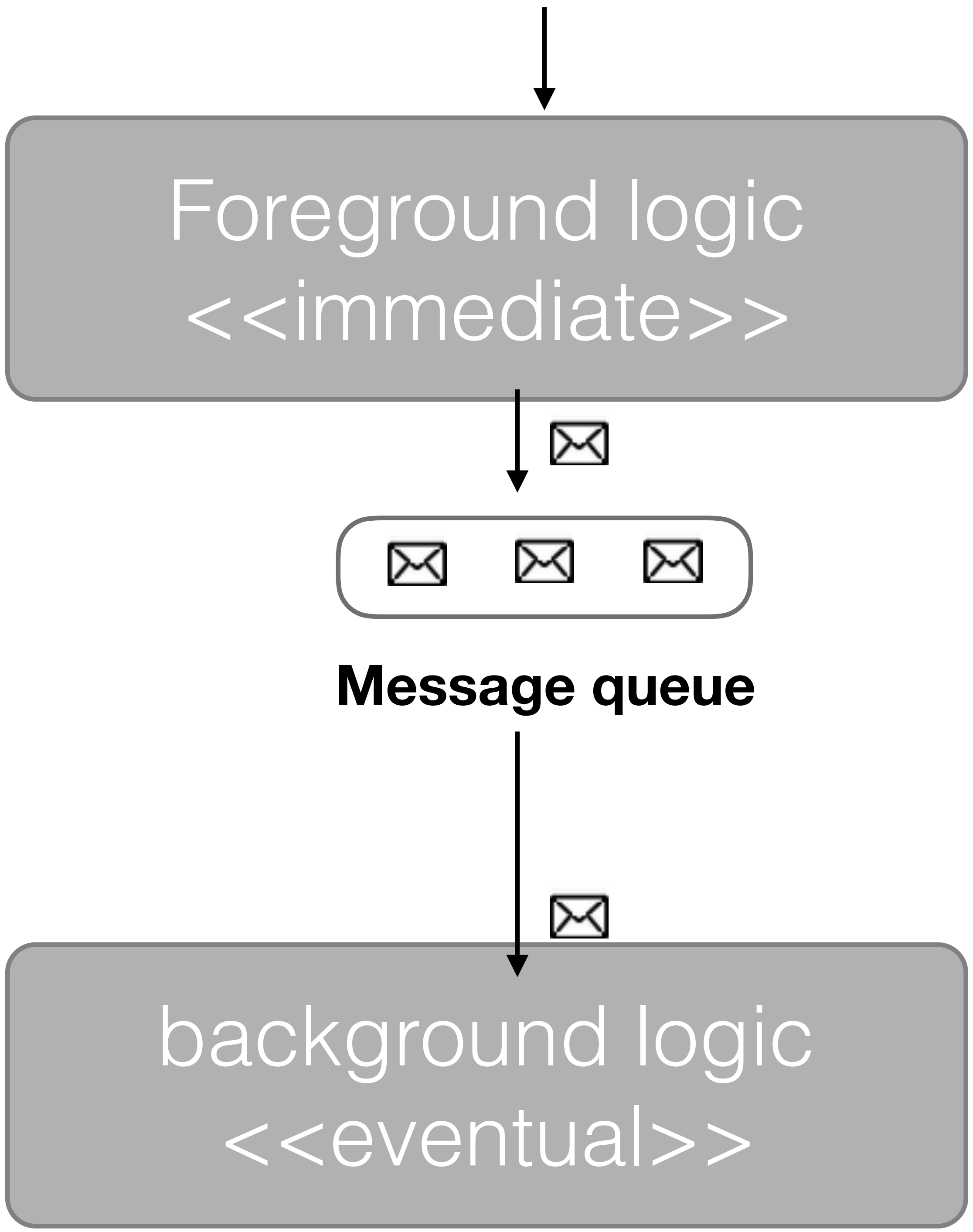
Disconnected call
<<one way >>

Flag

Poll in
x interval

Database

api

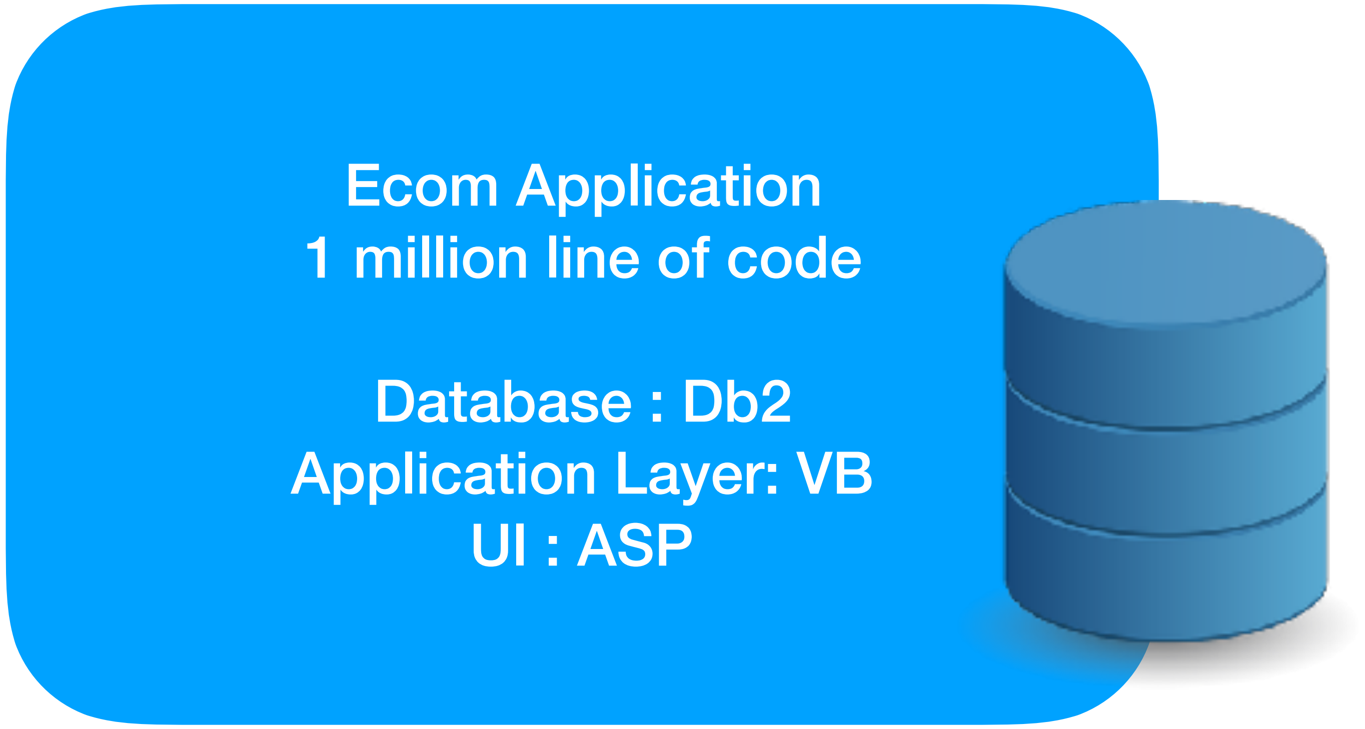


Foreground logic
<<immediate>>

Message queue

background logic
<<eventual>>

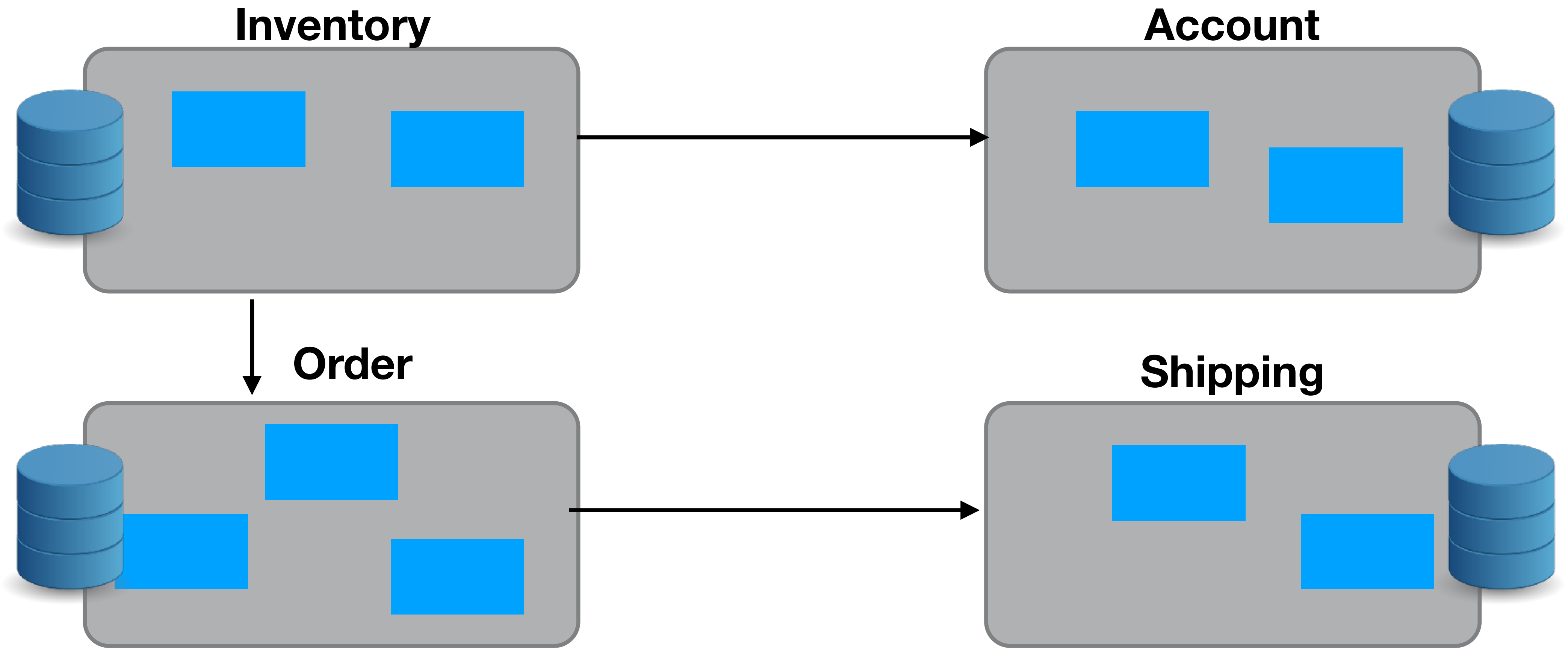
Legacy



Ecom Application
1 million line of code

Database : Db2
Application Layer: VB
UI : ASP

Modern

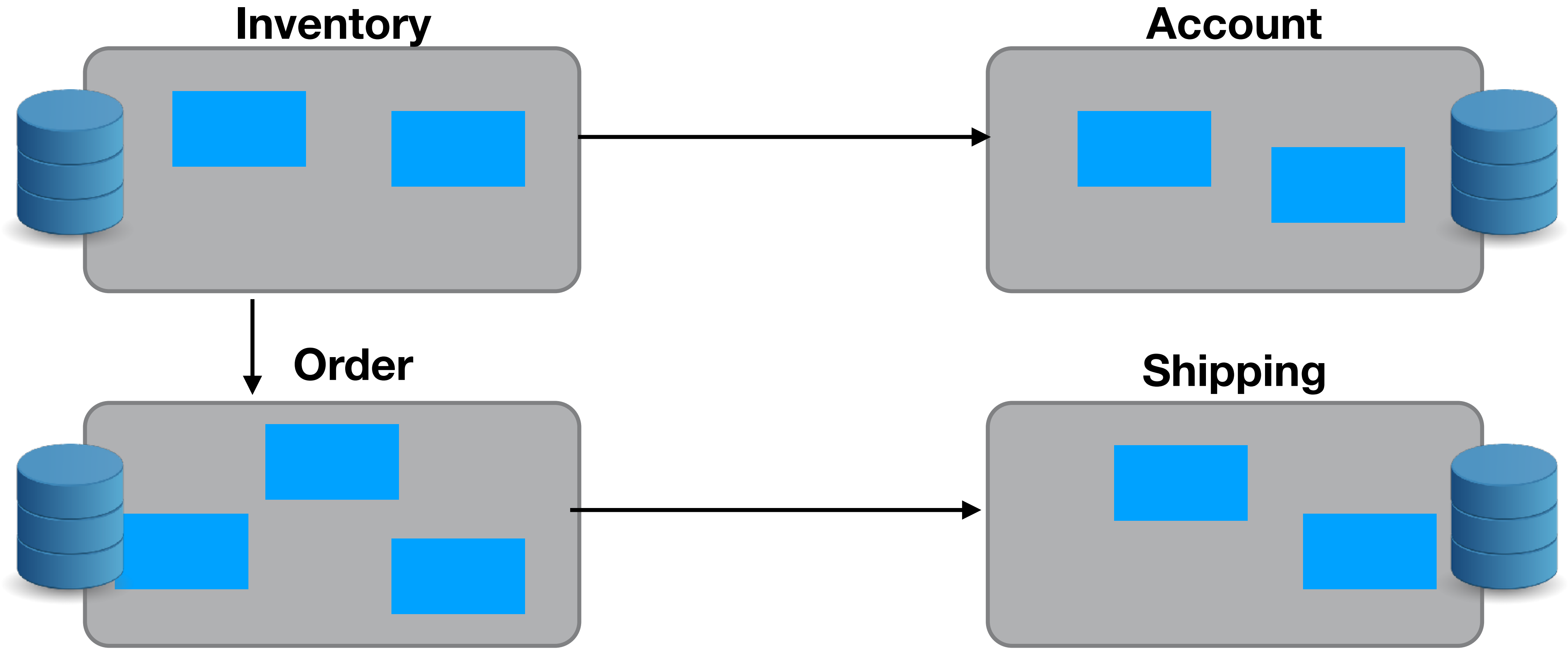


Inventory

Account

Order

Shipping



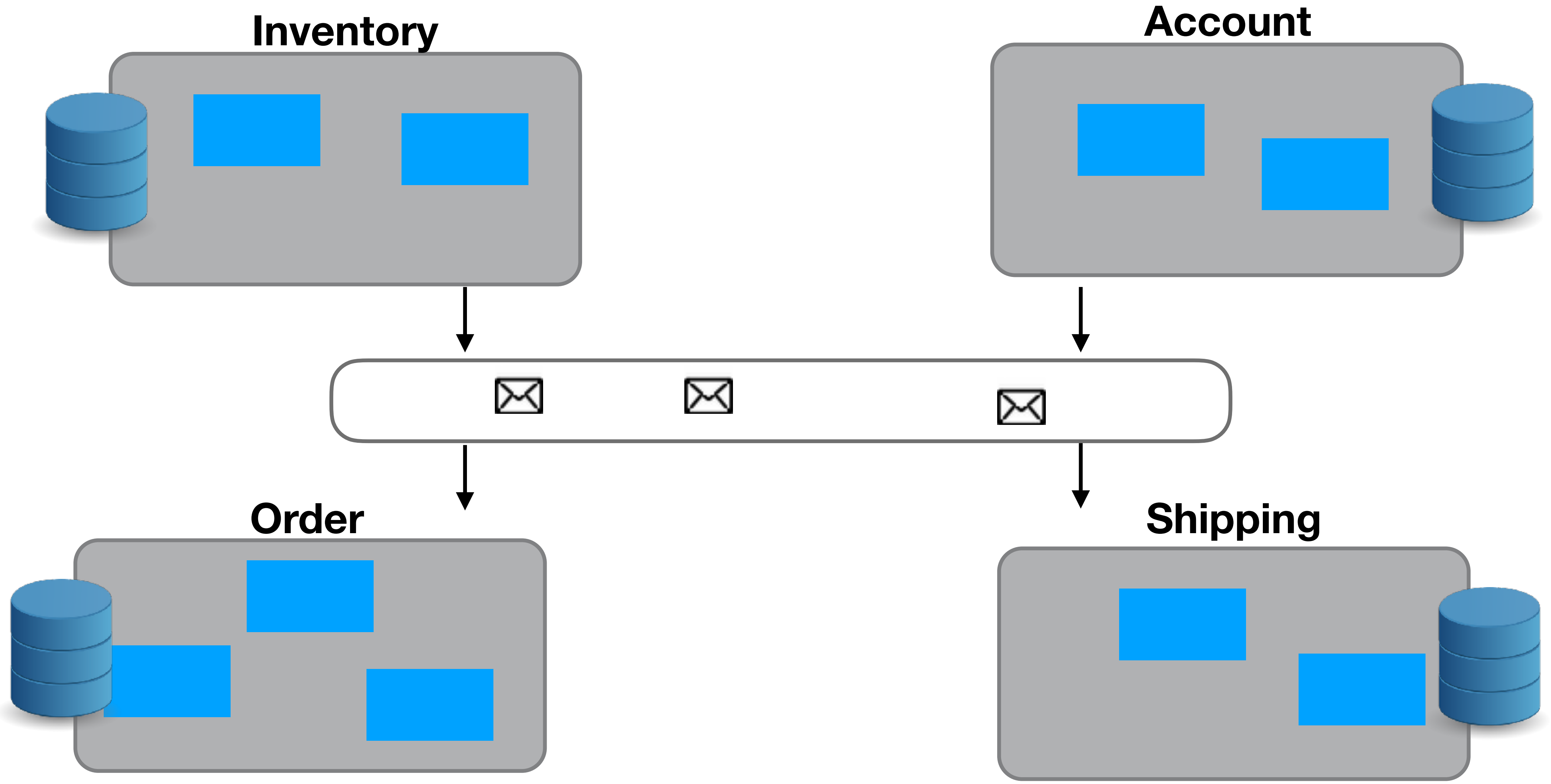
Inventory

Account

Order

Shipping

Modern

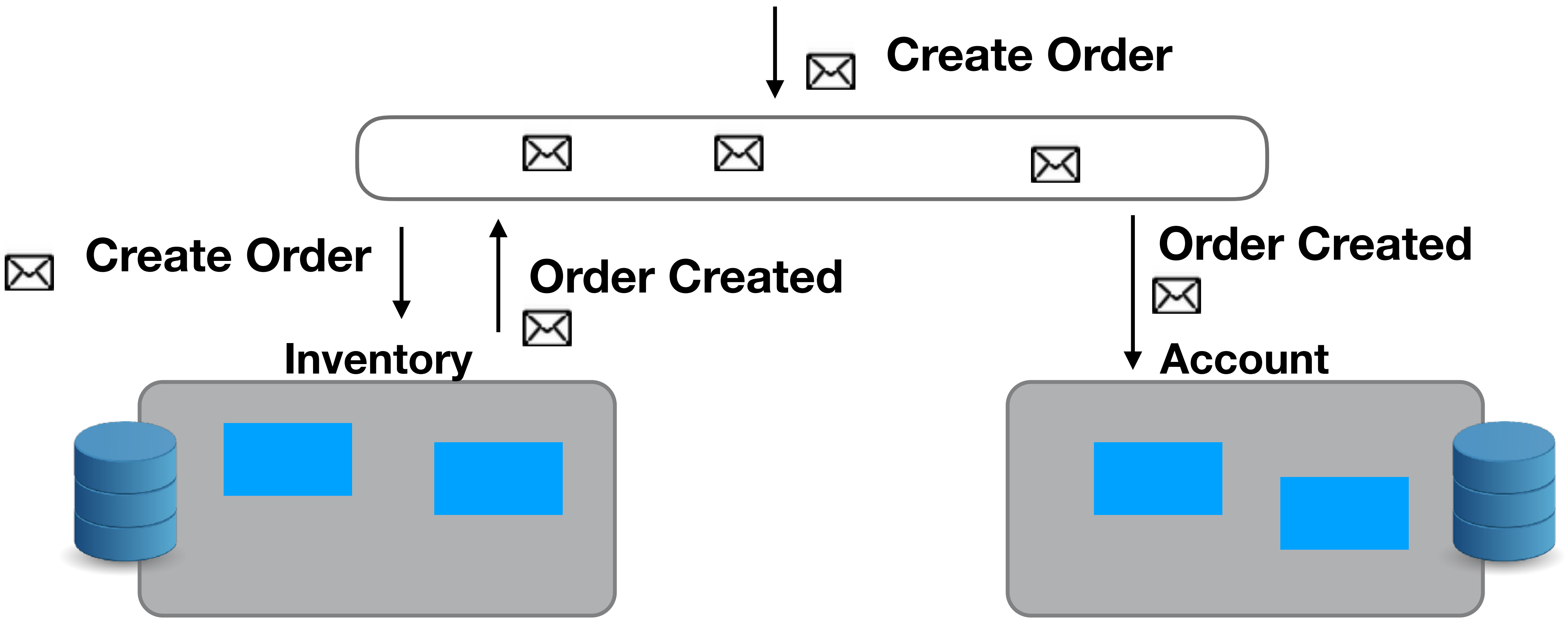


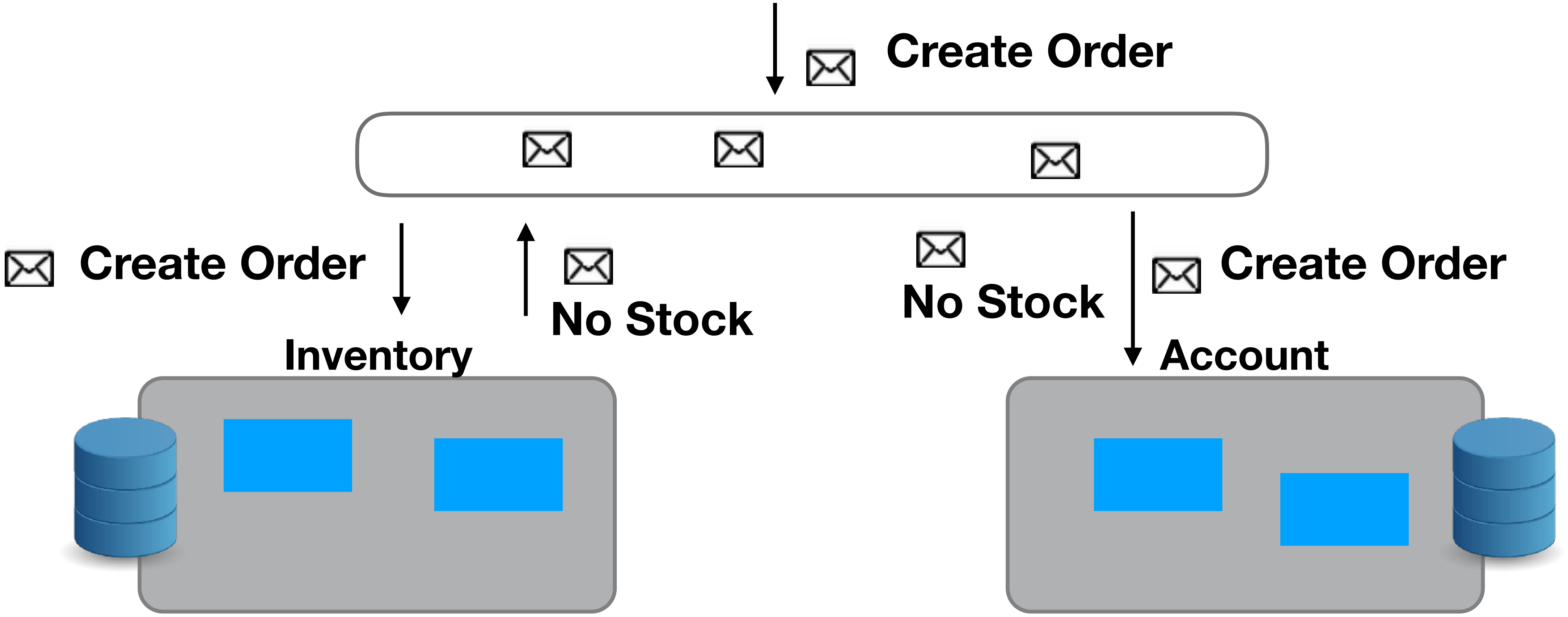
Inventory

Account

Order

Shipping


Create Order
Create Order
Order Created
Order Created
Inventory
Account



Create Order

Create Order

No Stock

No Stock

Create Order

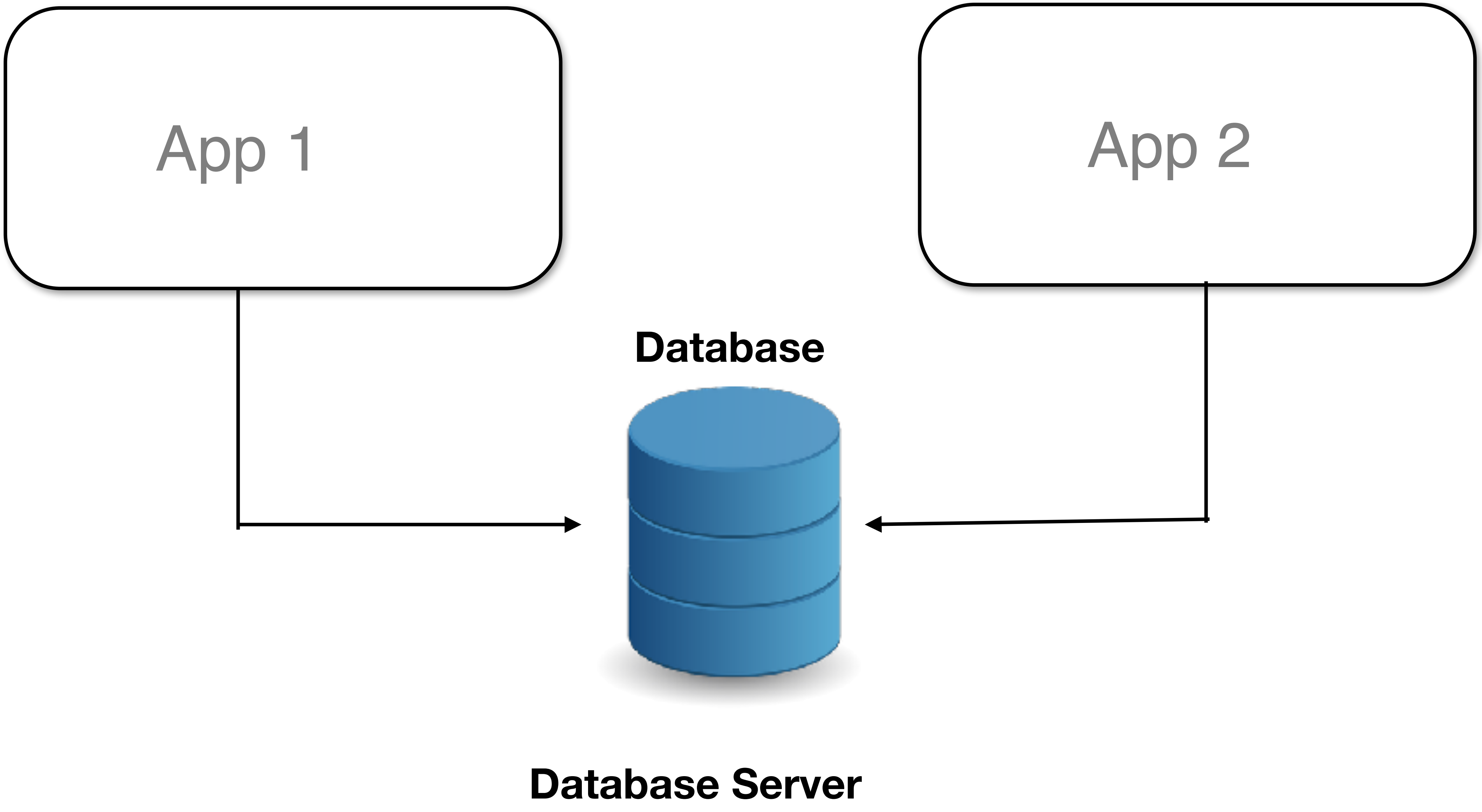
Inventory

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) :

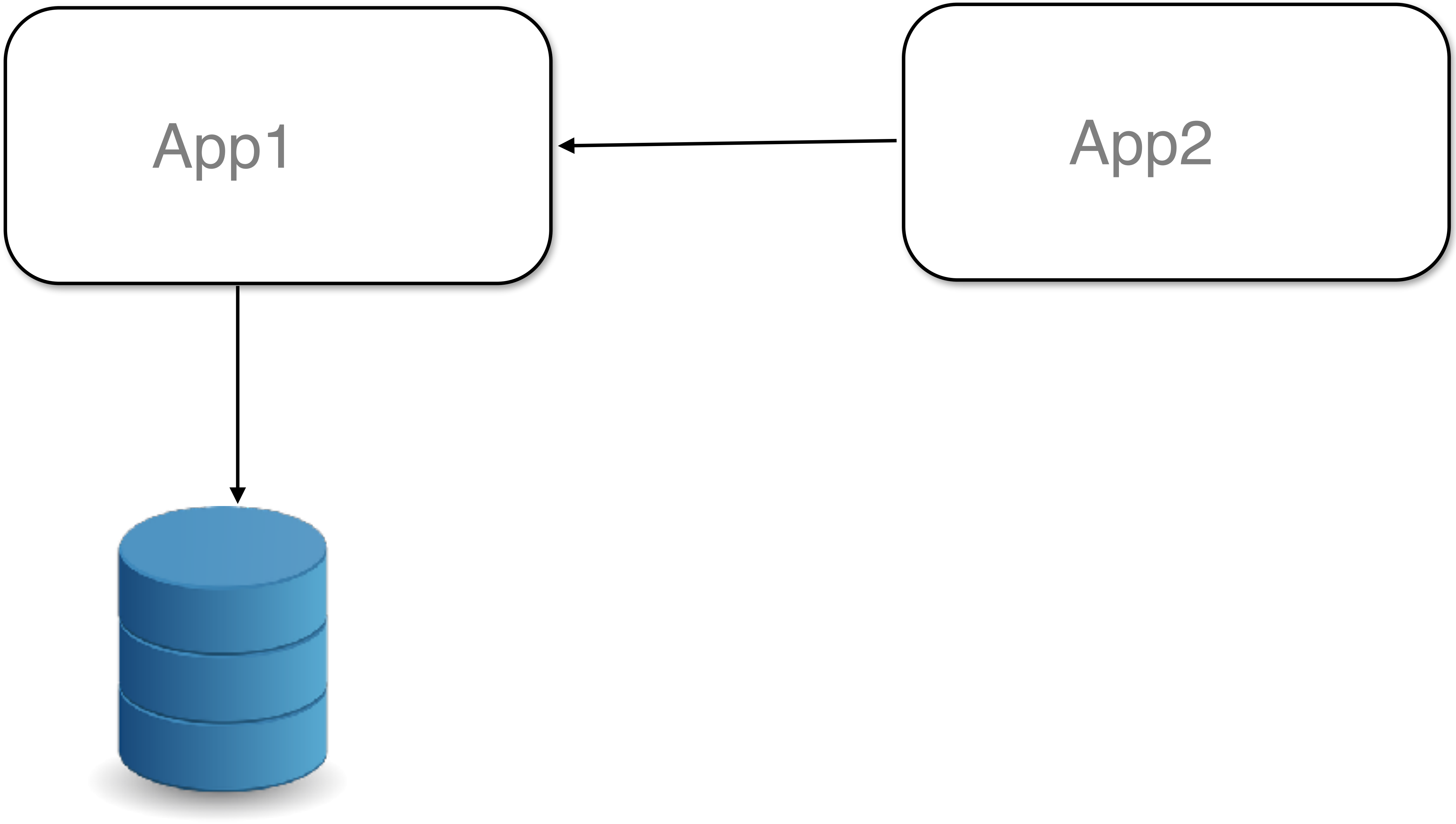


App 1

App 2

Database

Database Server


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

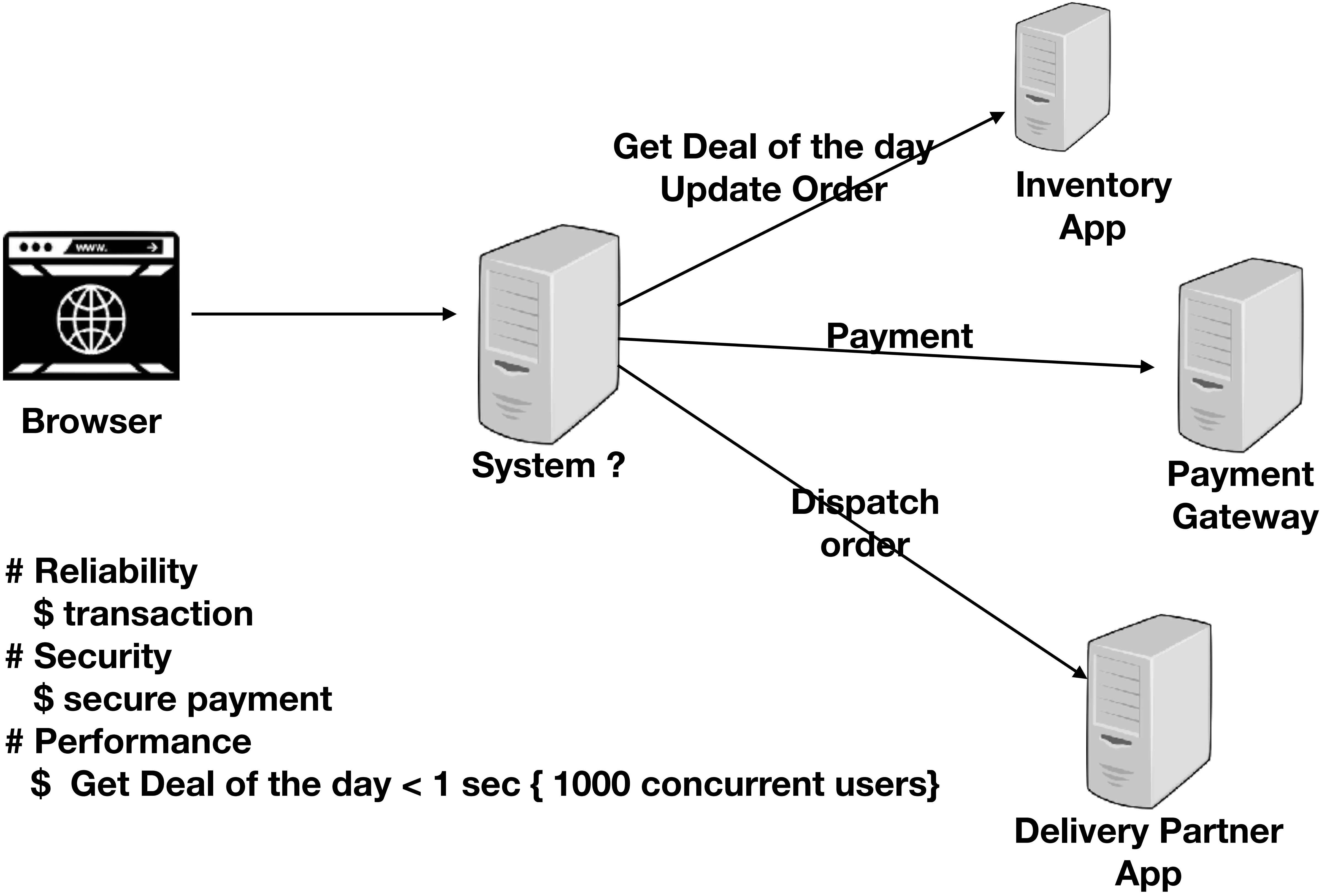
Quality requirements

# 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.

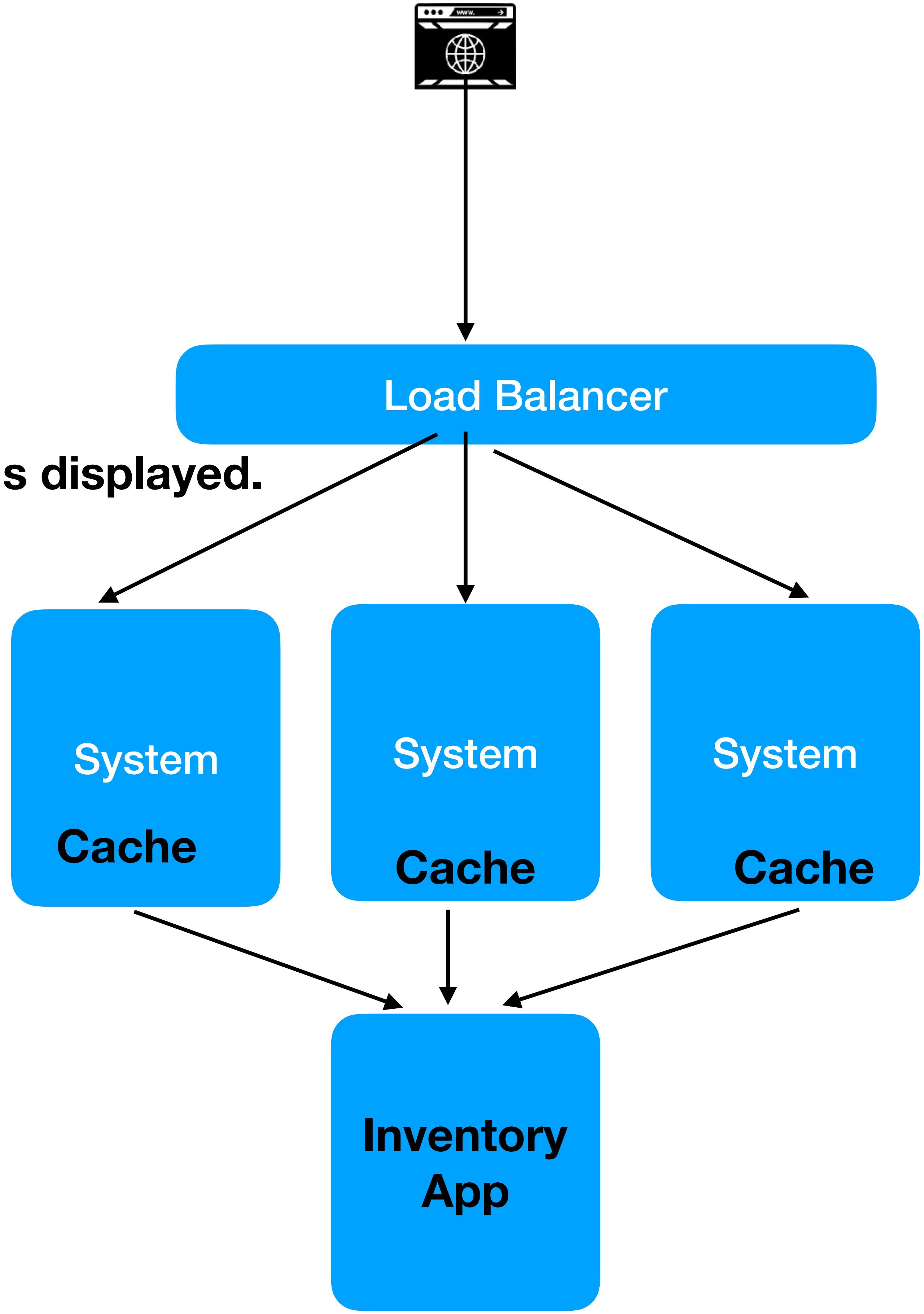
Deal of the day

# View the Product of the day
# Order the product



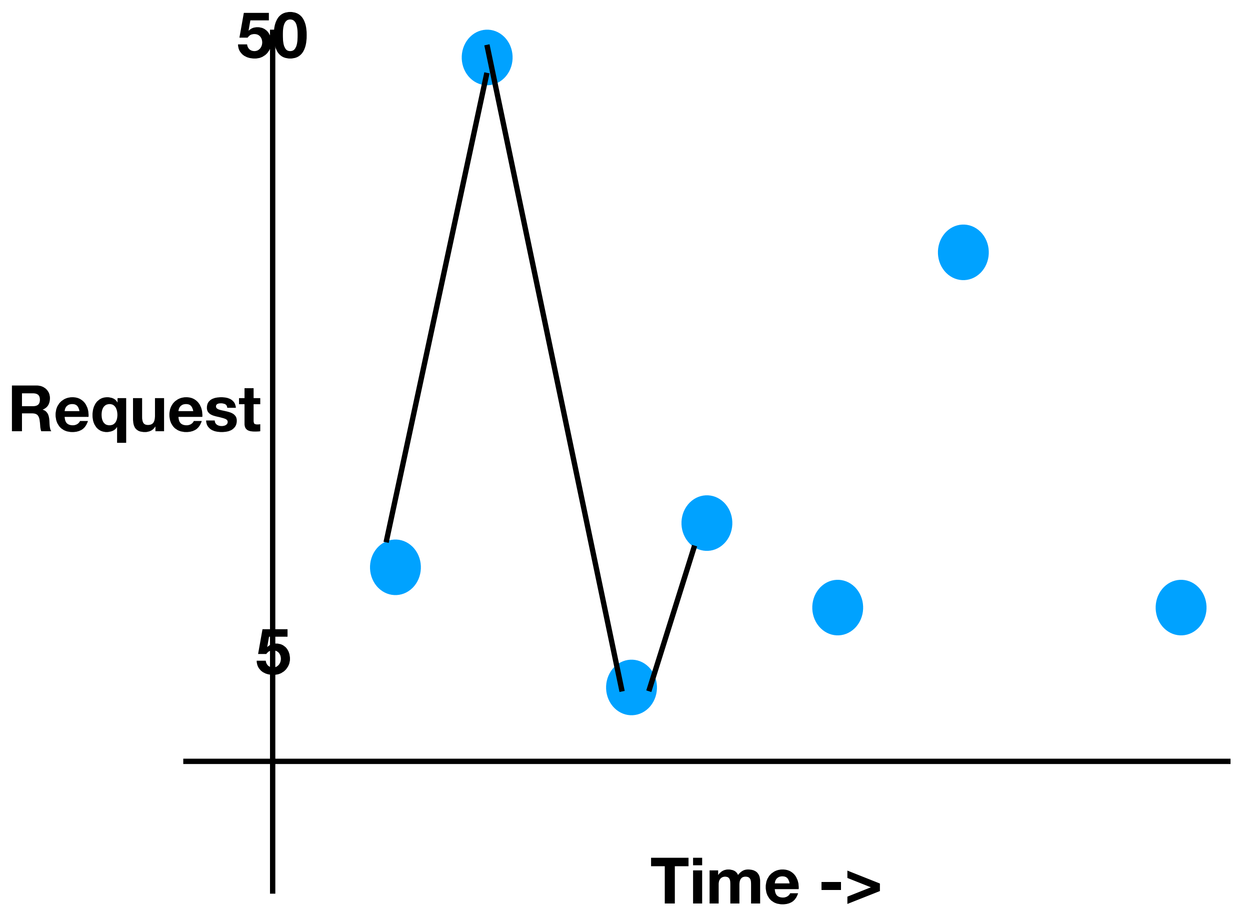
# Reliability
  $ transaction
# Security
  $ secure payment
# Performance
  $ Get Deal of the day < 1 sec { 1000 concurrent users}

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



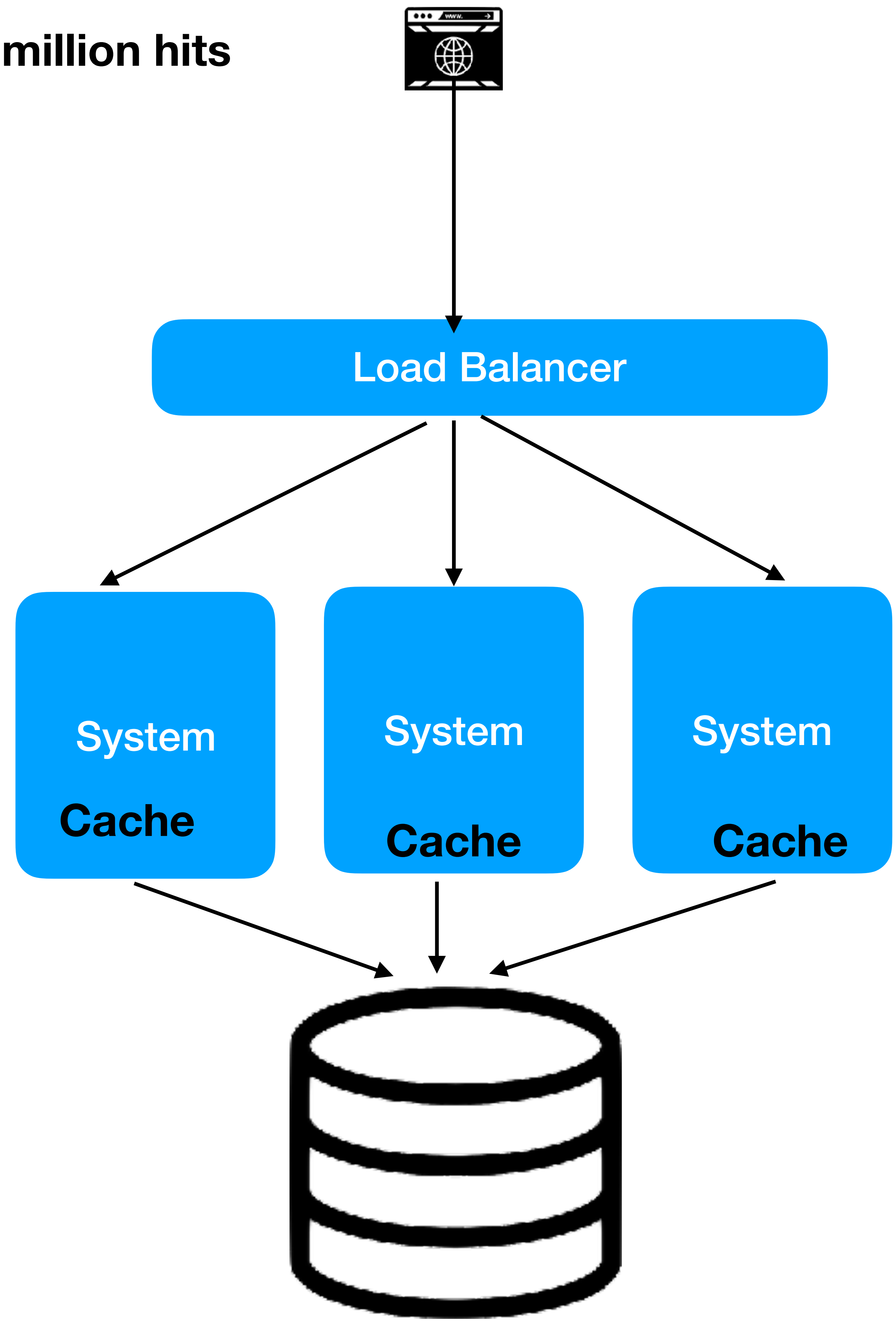
Tapproach-> Quality
# Cache -> performance
# Clone + LB -> Performance, Availability
# L3 FW, L7 FW -> Security
#

# view Deal of the day
# create order

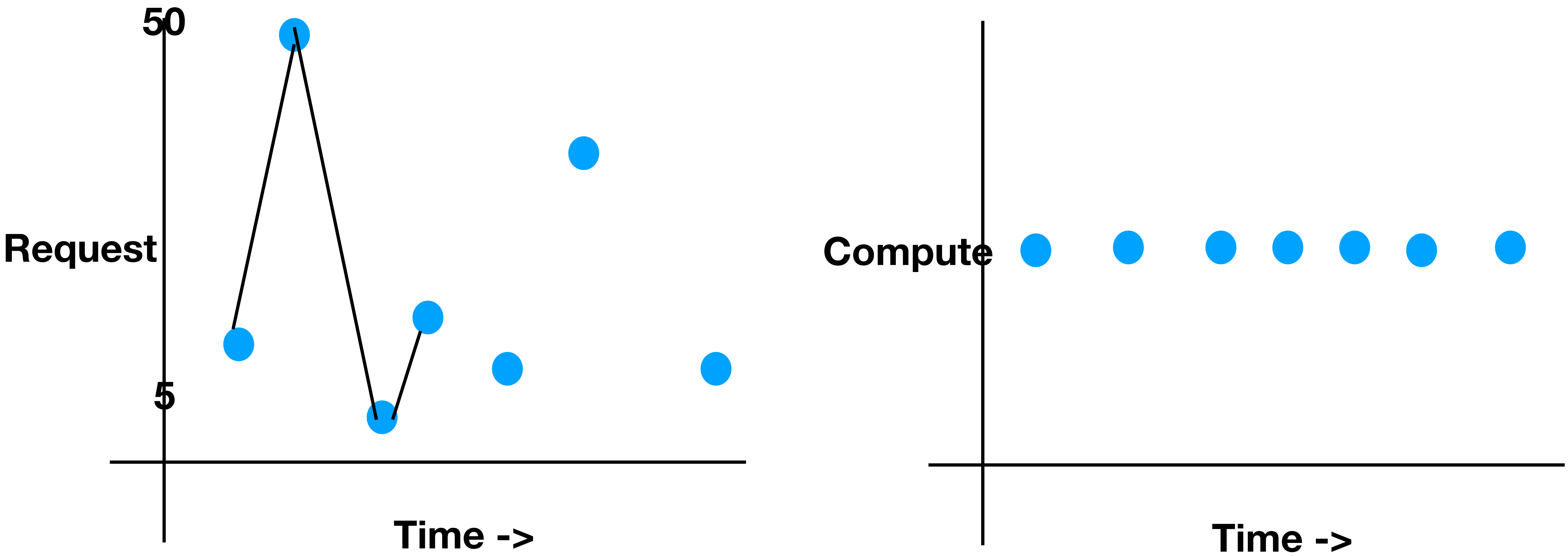


@ eventual consistency
@ stateless

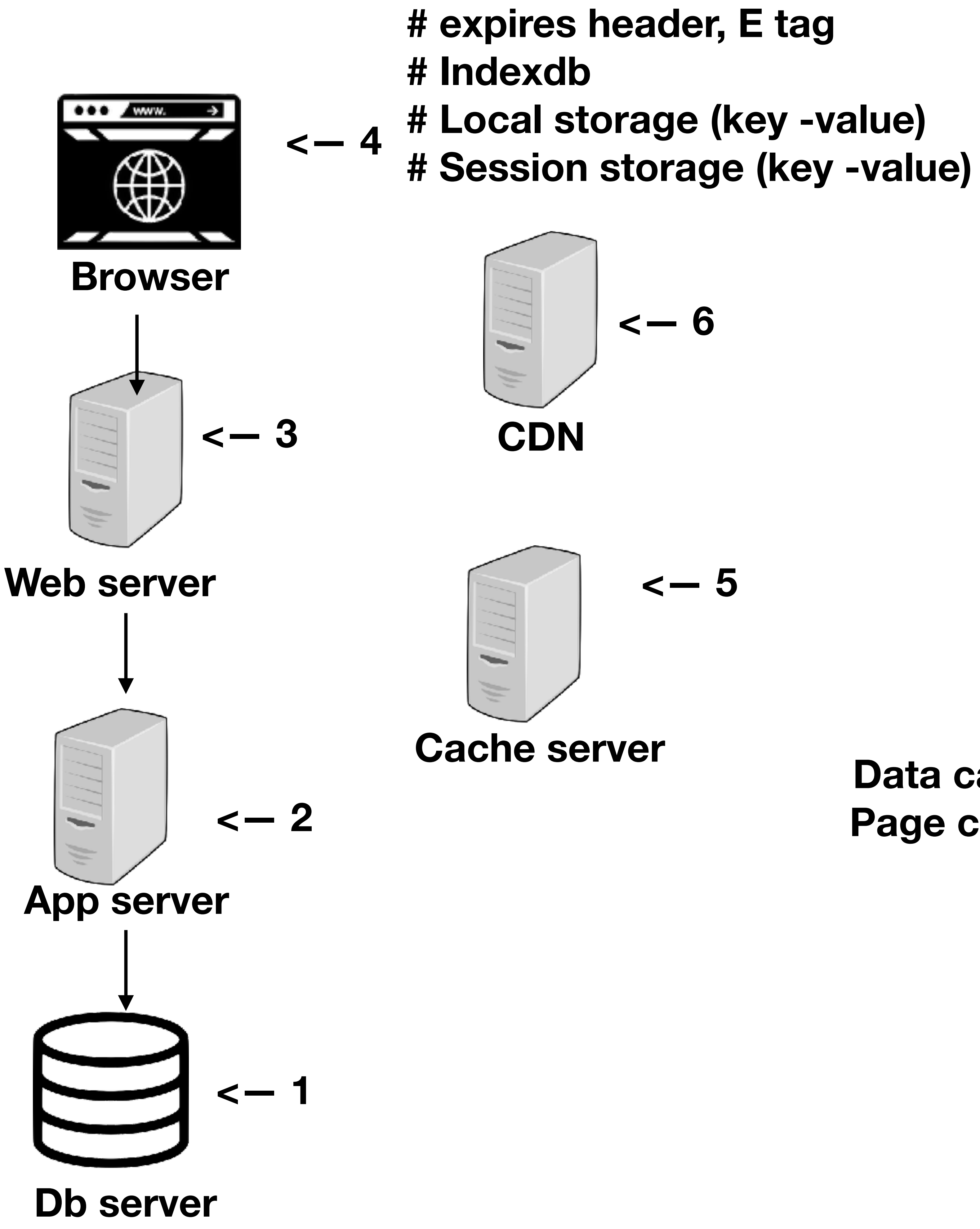
1 million hits



Load Leveling



Cache



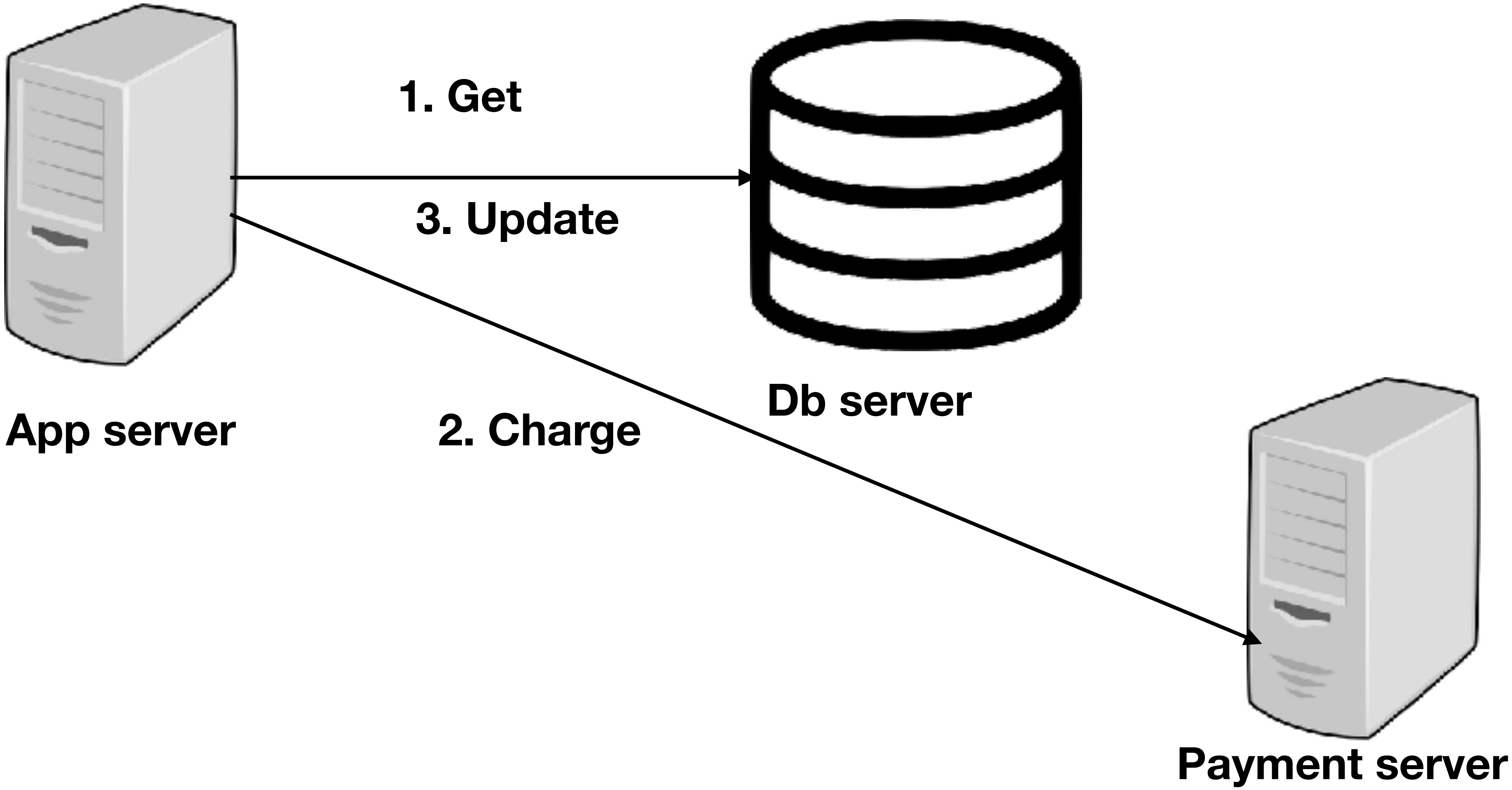
# expires header, E tag
# Indexdb
# Local storage (key -value)
# Session storage (key -value)

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

Data cache
Page cache

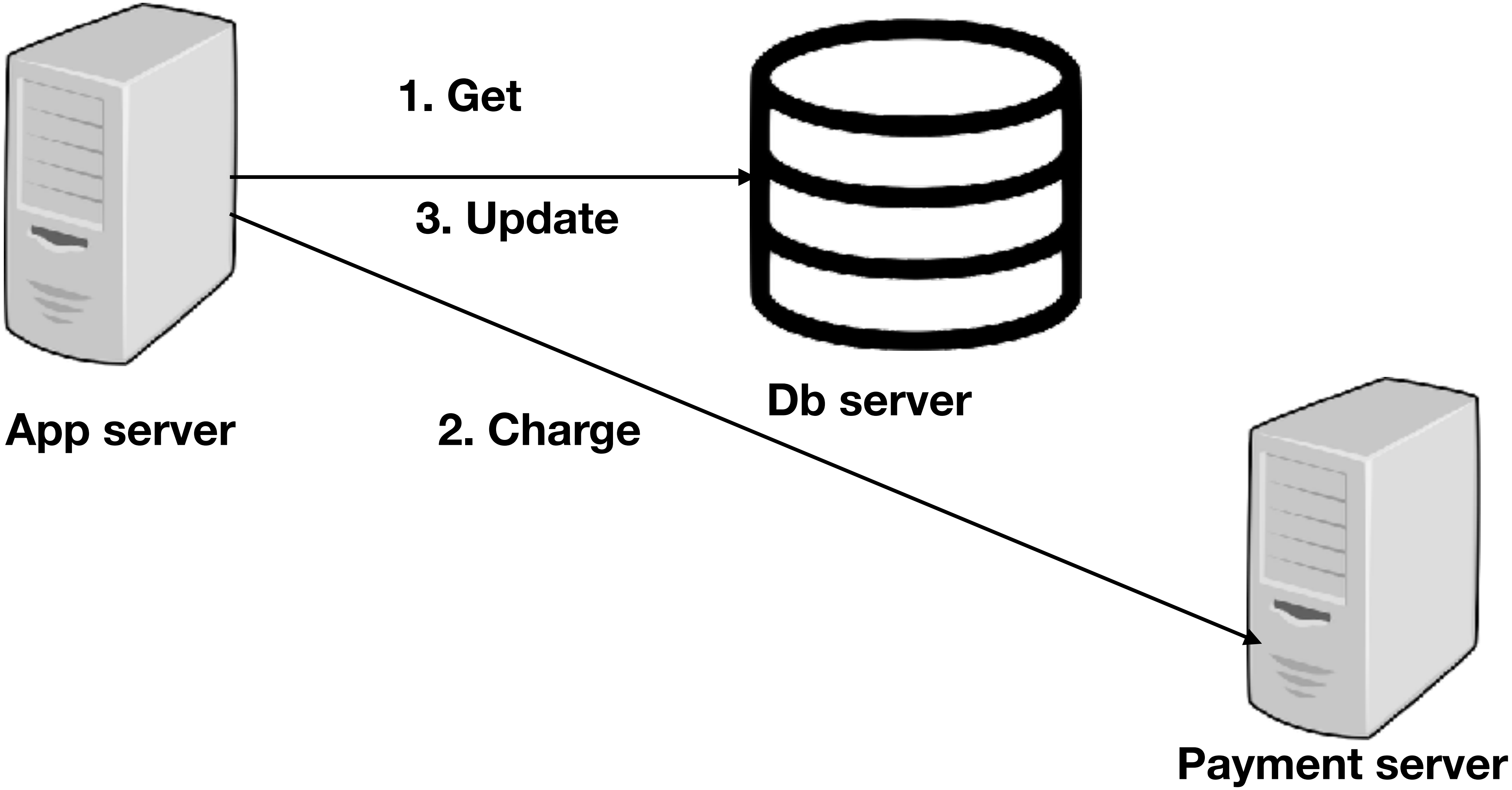
```
# cron job invokes job every 30 days

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)
}
```

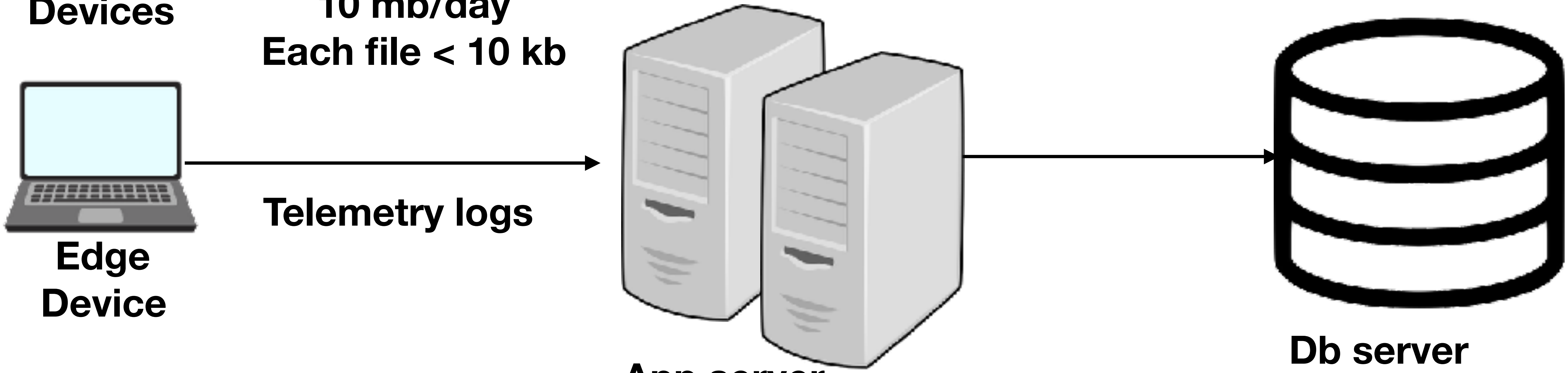


```
# cron job invokes job every 30 days

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

Edge
Device

10 mb/day
Each file < 10 kb

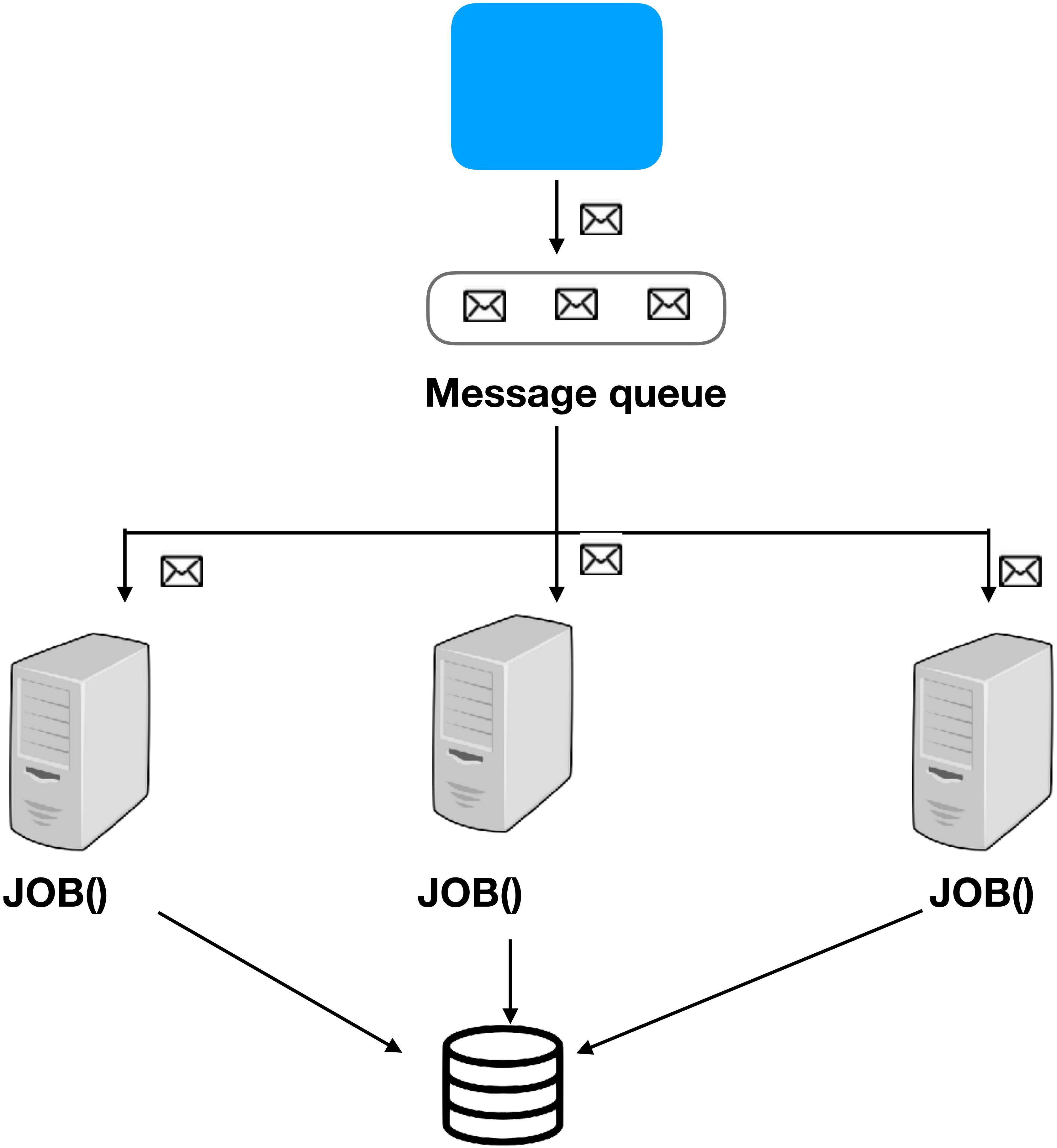
Telemetry logs

1 hour/ 10 mb / core

App server

Db server

# Disconnected
# Connected
# Occasionally connected

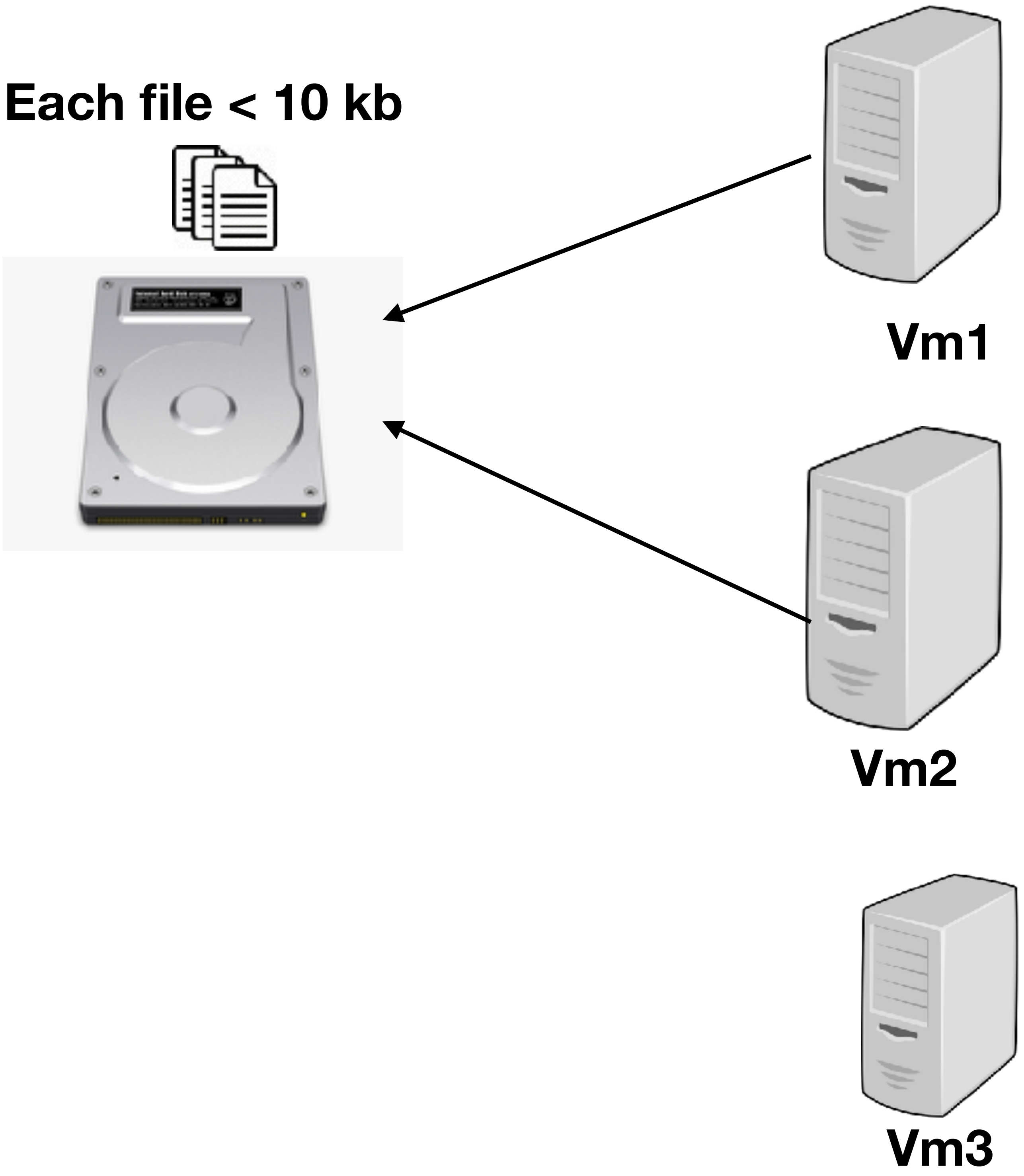


Message queue

JOB()

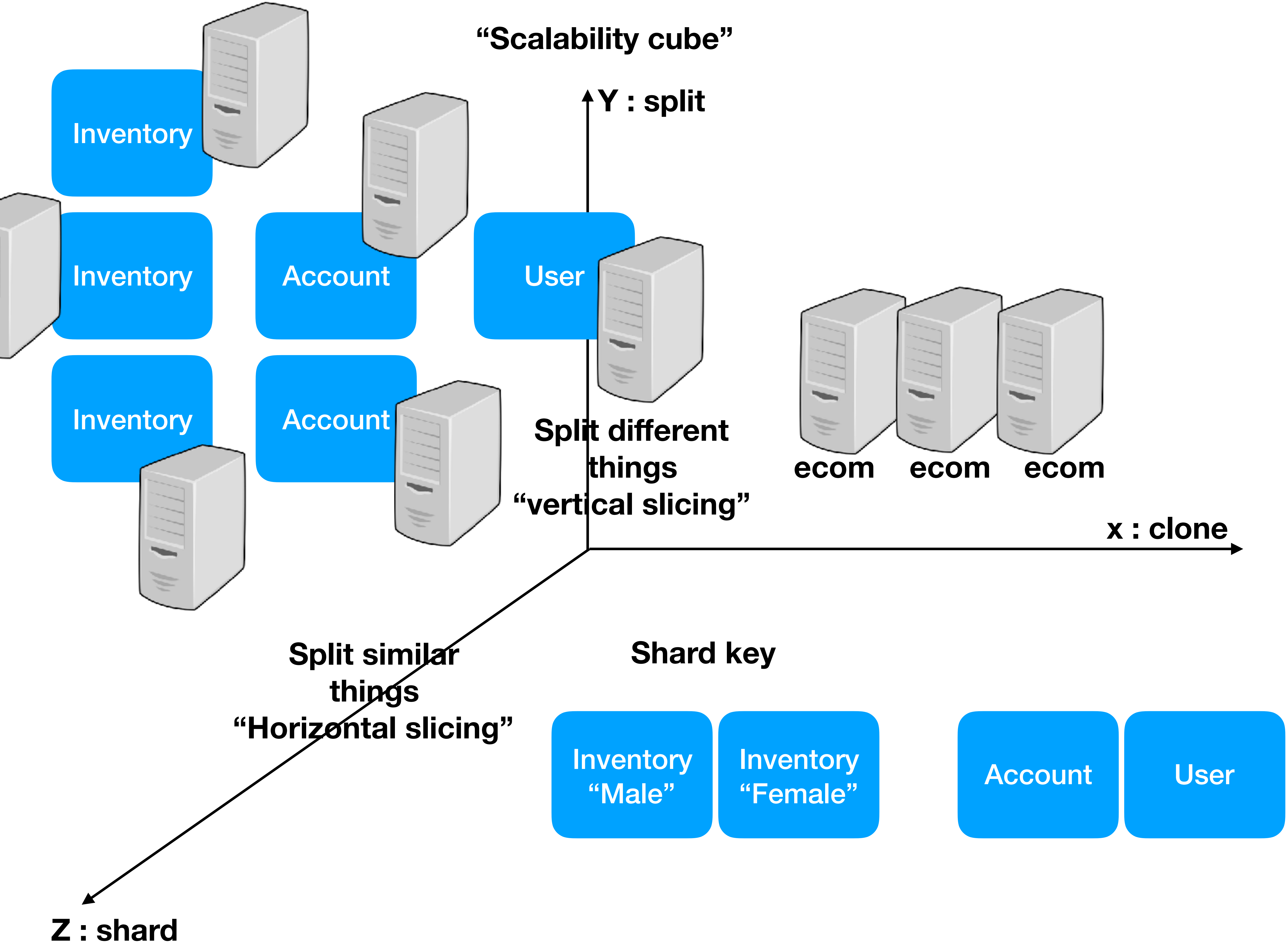
JOB()

JOB()



```
# 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

Split different things “vertical slicing”

Inventory

Inventory

Inventory

Account

Account

User

x : clone

ecom

ecom

ecom

Z : shard

Split similar things “Horizontal slicing”

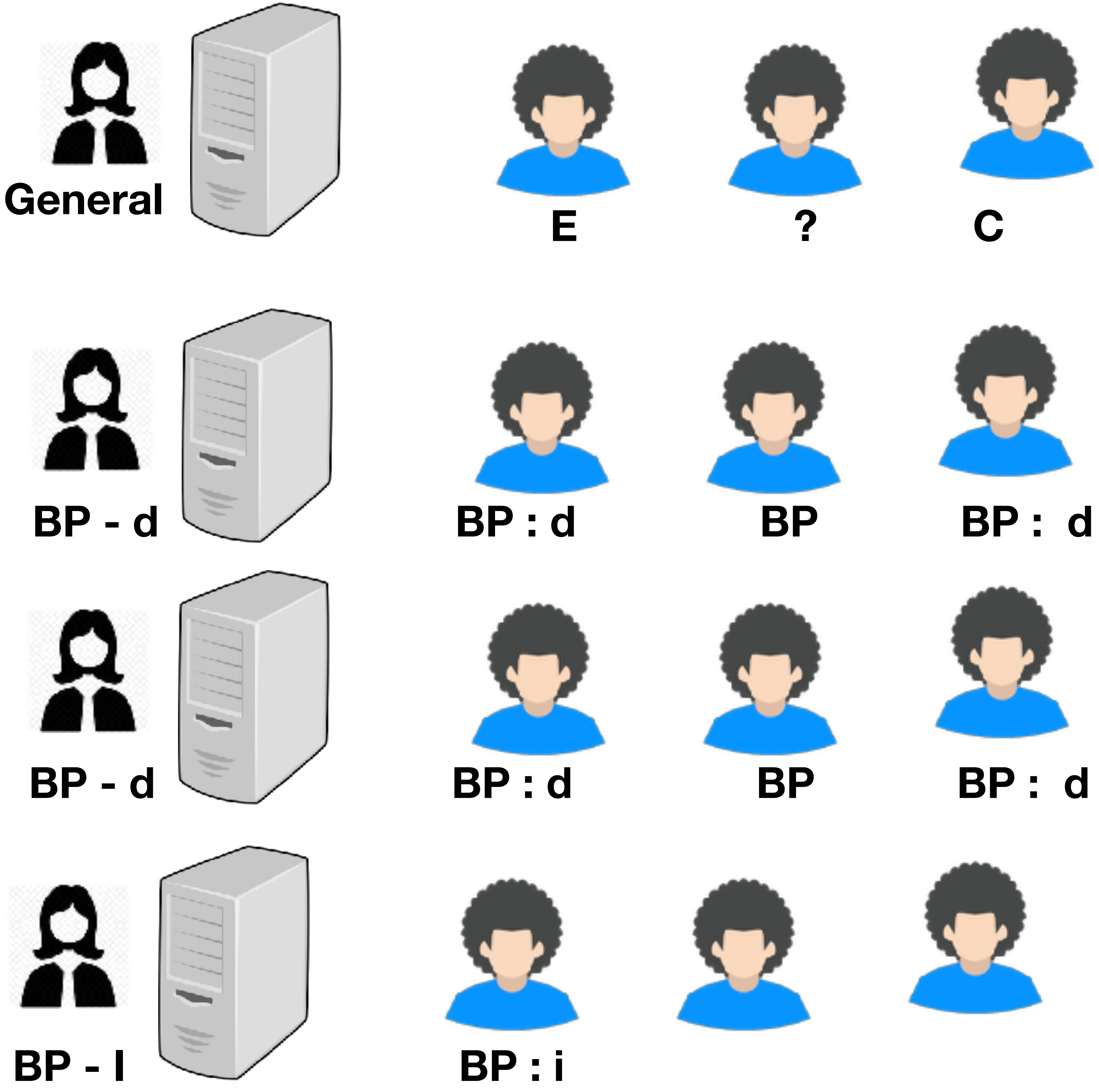
Shard key

Inventory “Male”

Inventory “Female”

Account

User



General E ? C

BP - d BP : d BP BP : d

BP - d BP : d BP BP : d

BP - I BP : i

Quality attribute ?

- Maintainability

Approach (tactic/style) ?

- Modularization
- Health monitoring
- Config
- Documentation
- Styling
- Automated tested
- CI/CD
- …

Measure ?

- Cyclomatic complexity
- Code coverage
- Low Coupling

(What) Quality attribute ?

- Performance

(How) Approach (tactic/style) ?

- Caching
- Compression
- Parallel
- Object pooling
-

Measure ?

- Latency
- Throughput
- Response time
-

(What) Quality attribute ?

- 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

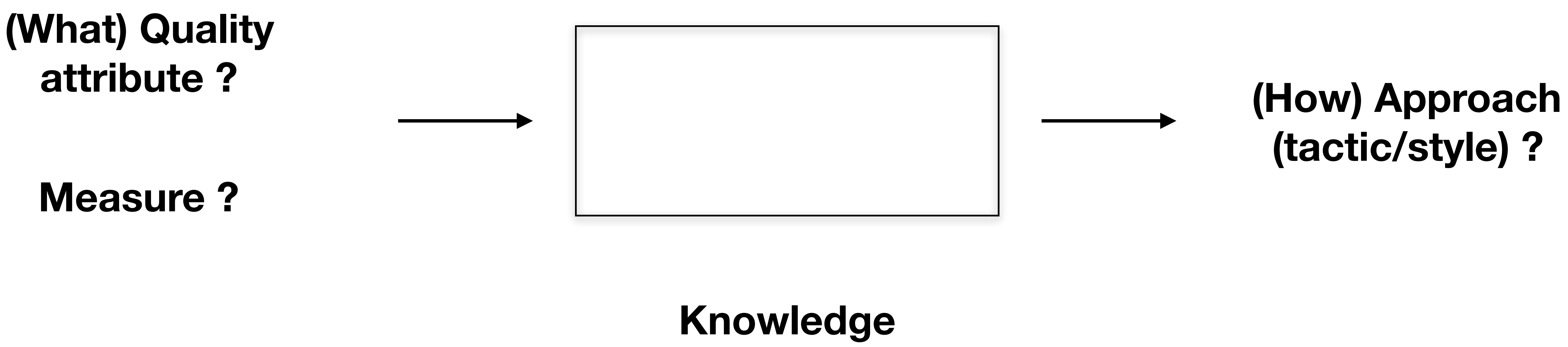
(How) Approach (tactic/style) ?

- Caching
- Compression
- Parallel
- Object pooling
- Reusability
- Authorization
- Input validation
- Throttling (max)
- ACID (transaction)
- Testability

Measure ?

- 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)

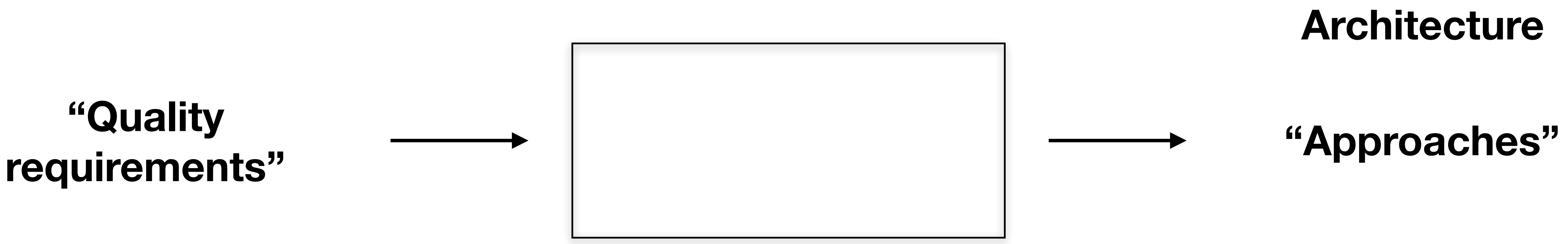


(What) Quality attribute ?

Measure ?

(How) Approach (tactic/style) ?

Knowledge

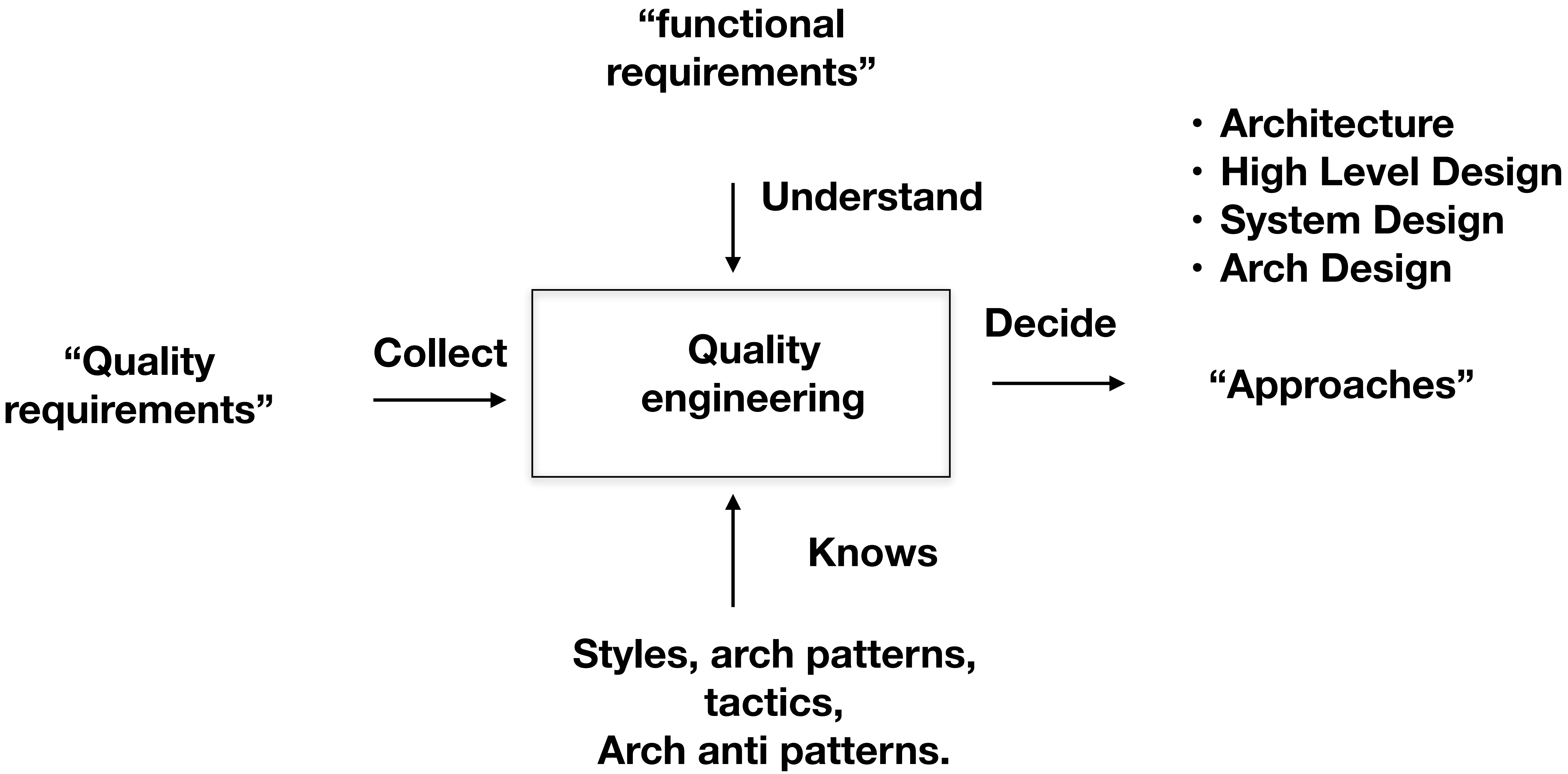


“Quality requirements”

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

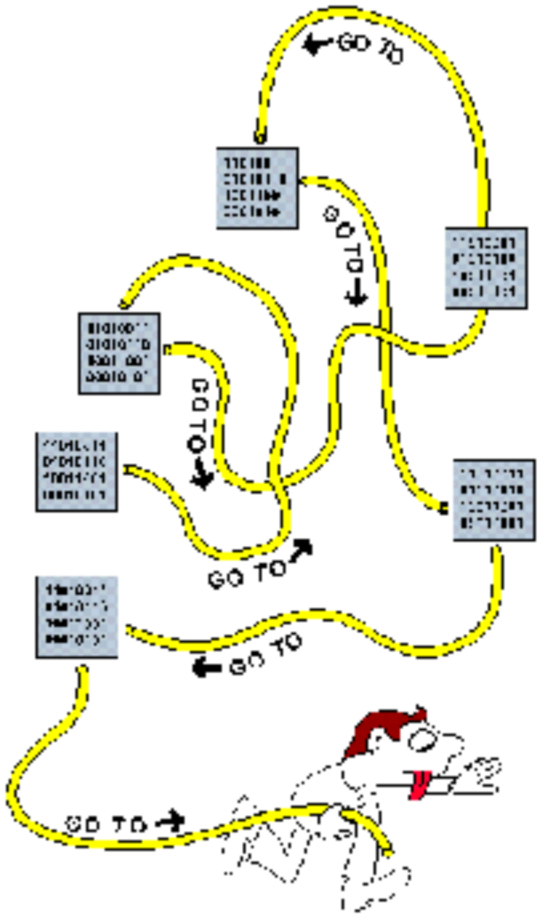
Decide

“Approaches”

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

Knows

Styles, arch patterns,
tactics,
Arch anti patterns.



“guidelines”

Follow

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

“fun requirement”

Understand

Manage Code Complexity

Create

“Skeleton for Code”

Knows

OO patterns, fun pattern, lang idioms

Togaf, dodaf, zachman fwk

Enterprise Architect

(align)

Product/Solution Architect
(Quality of the product)

Domain Architect

(Quality of the process)

Application Architect

(Quality of the application)

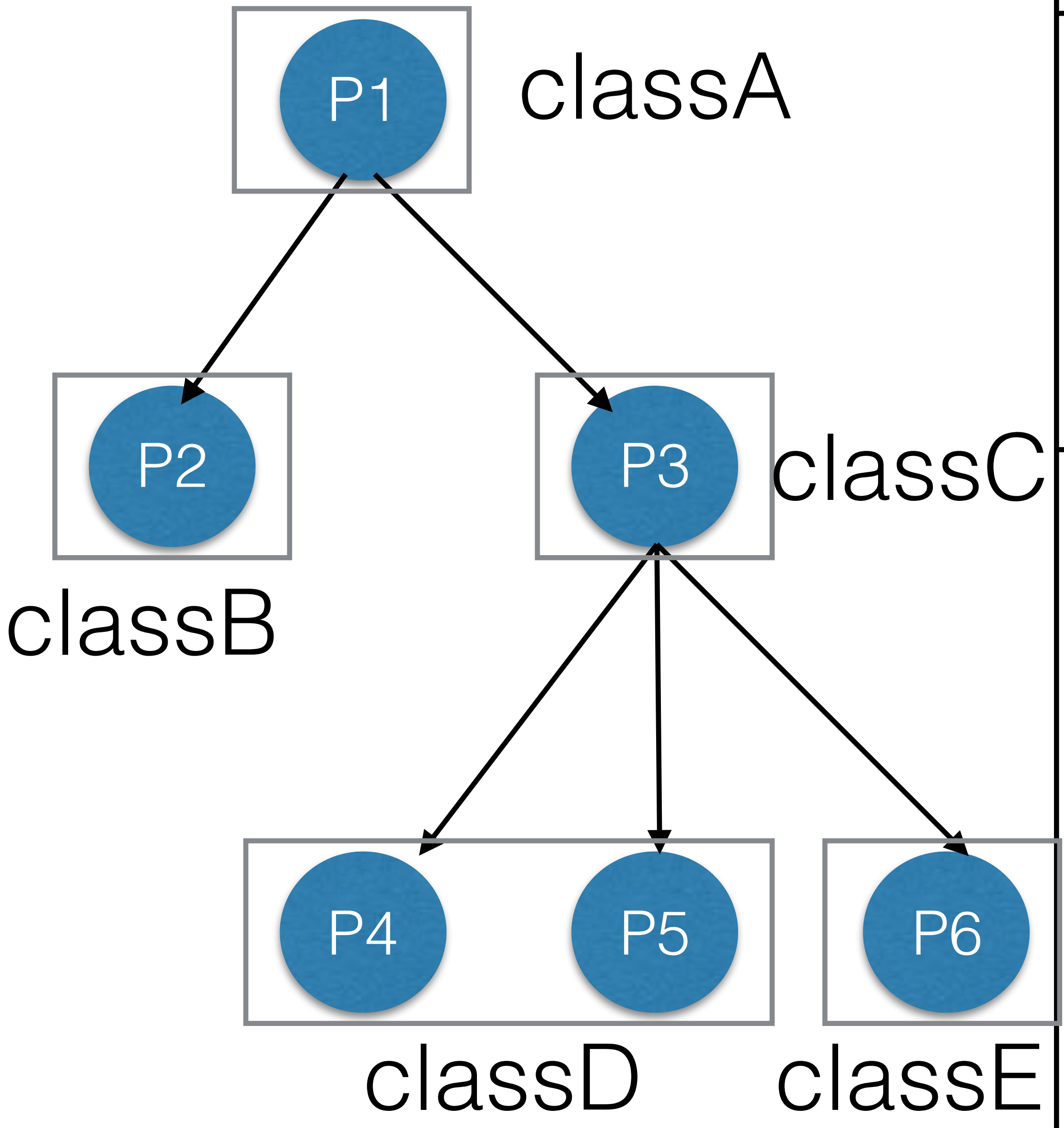
Vertical Architect

Security Architect
UX
Data
Infra
Cloud
Java
…

Proc vs OO vs fun

Procedural Prog

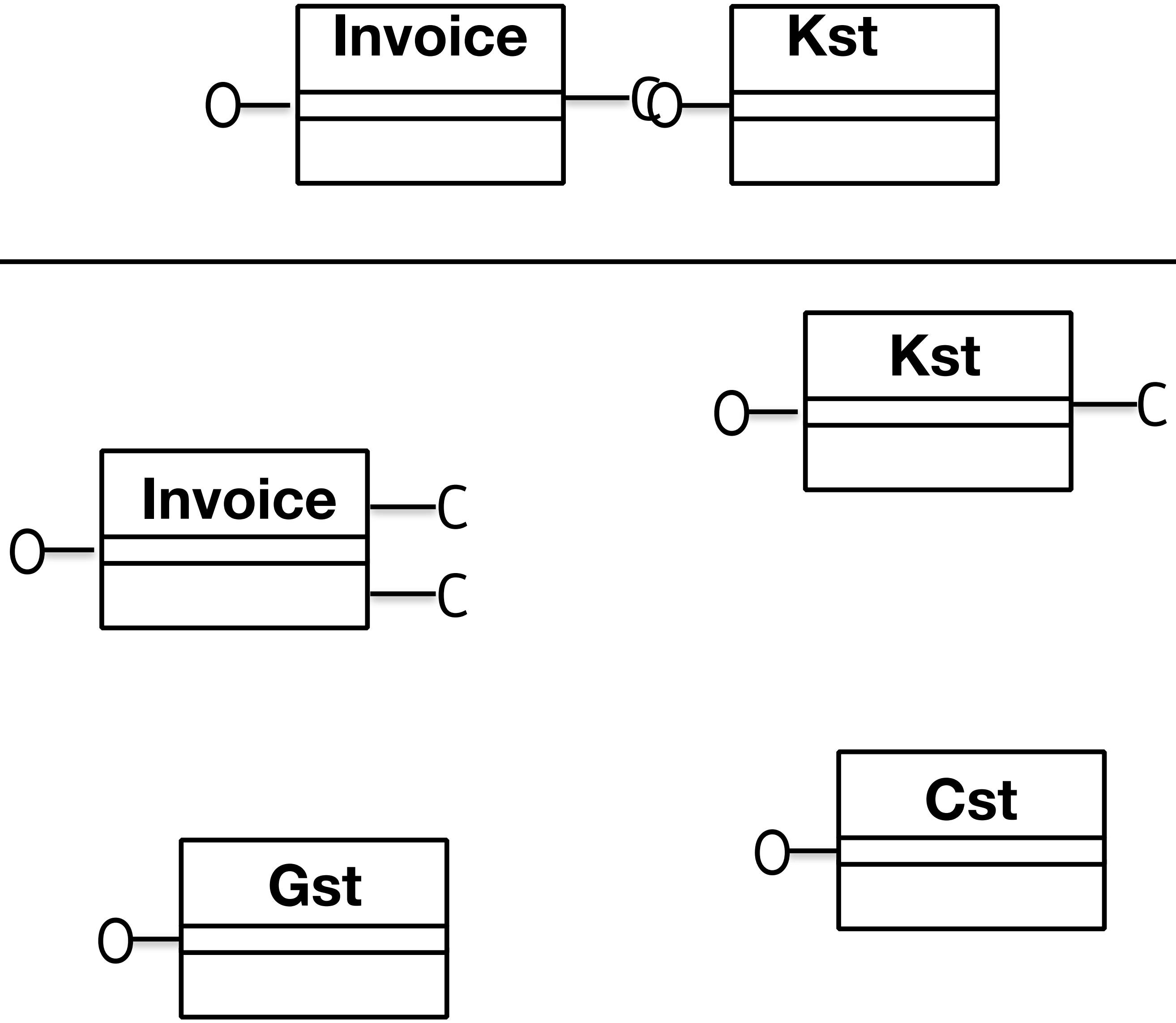
(tree)



(top down)

OO Prog

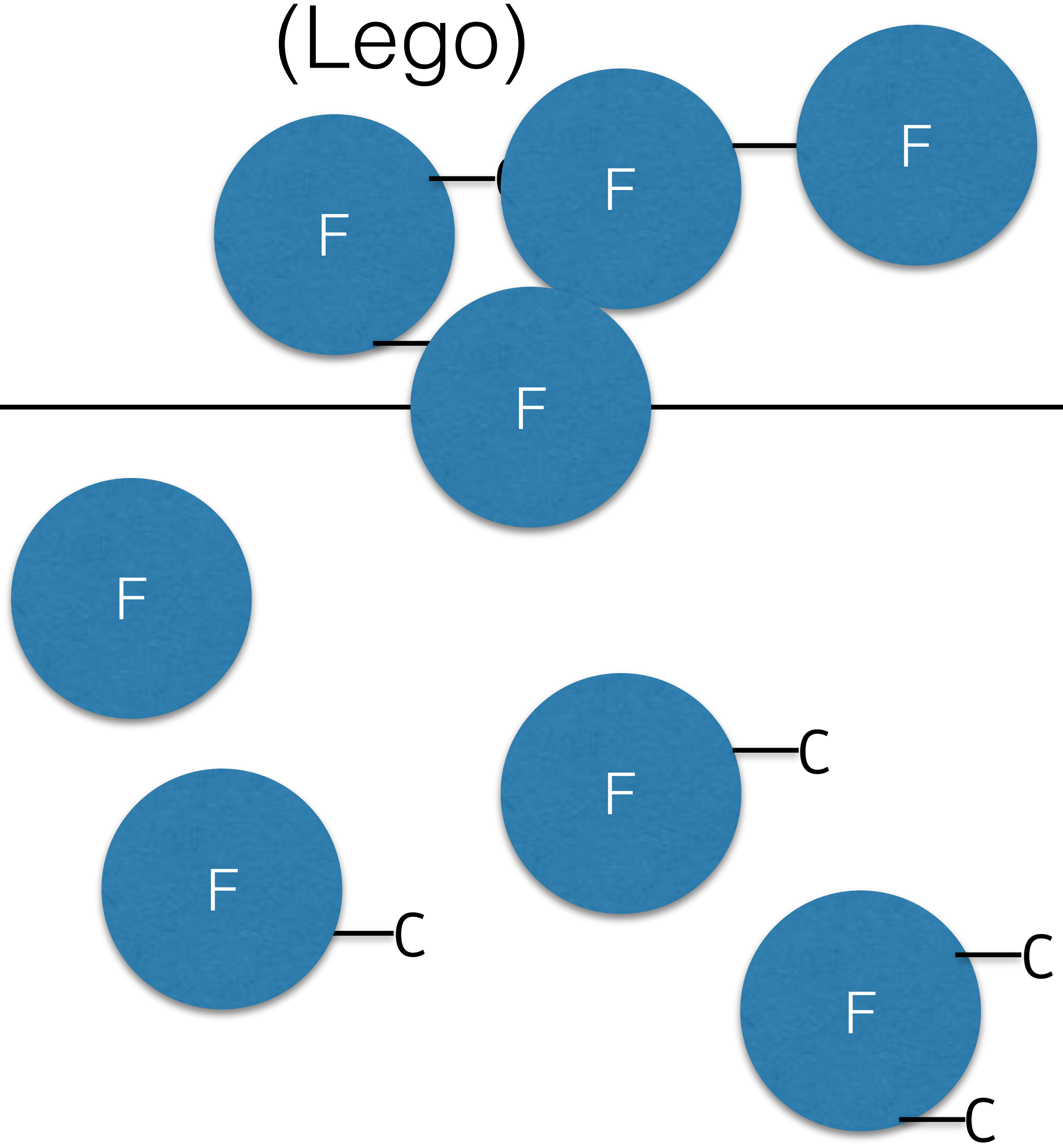
(Lego)



(bottom up)

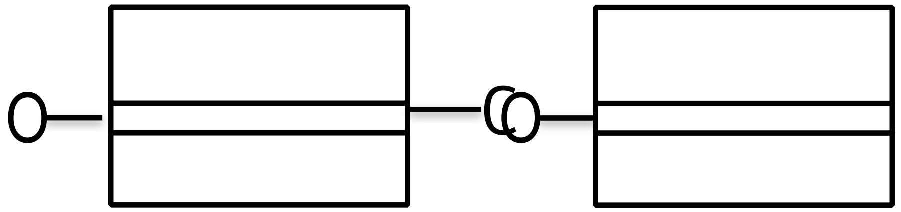
Functional Prog

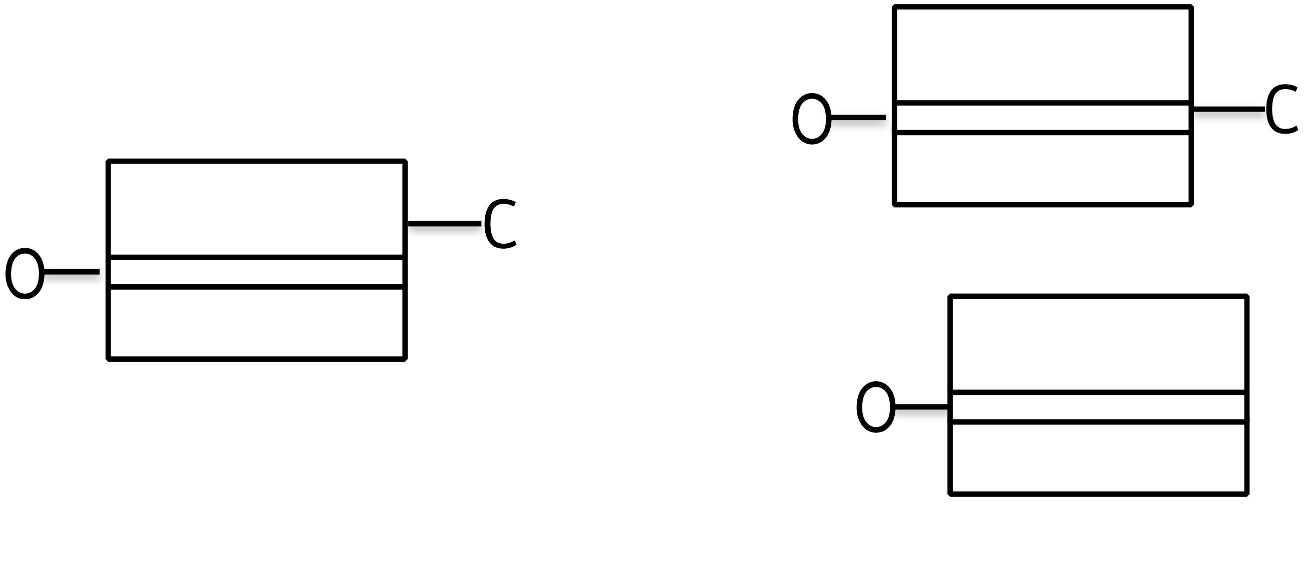
(Lego)

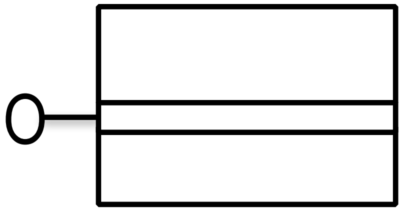


OO Prog

(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

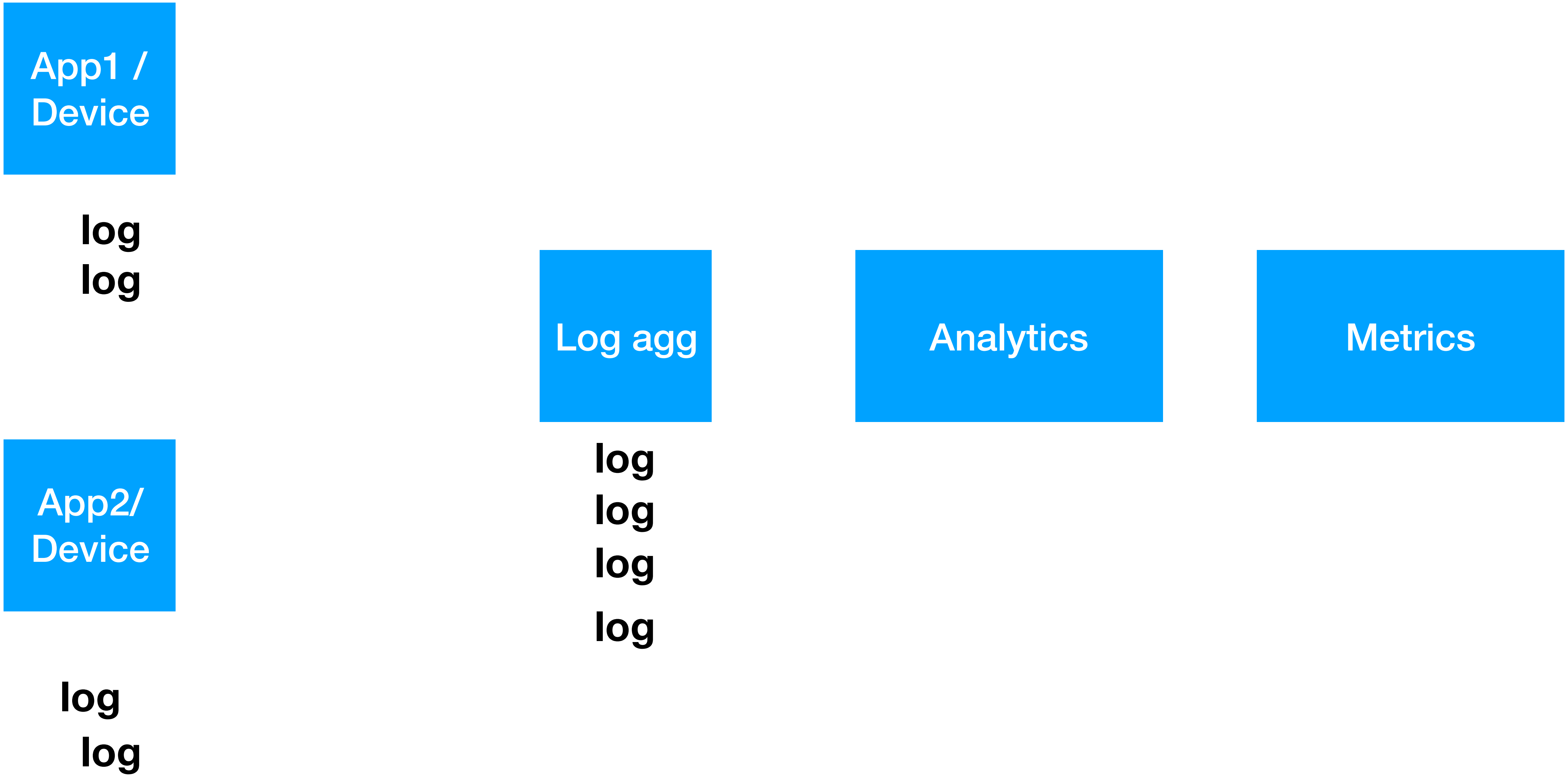
- <GreatDeal.com >
- Single product with n qty for a day
- Web App
- Collect payment if stock exist
- Send the product through delivery partner

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

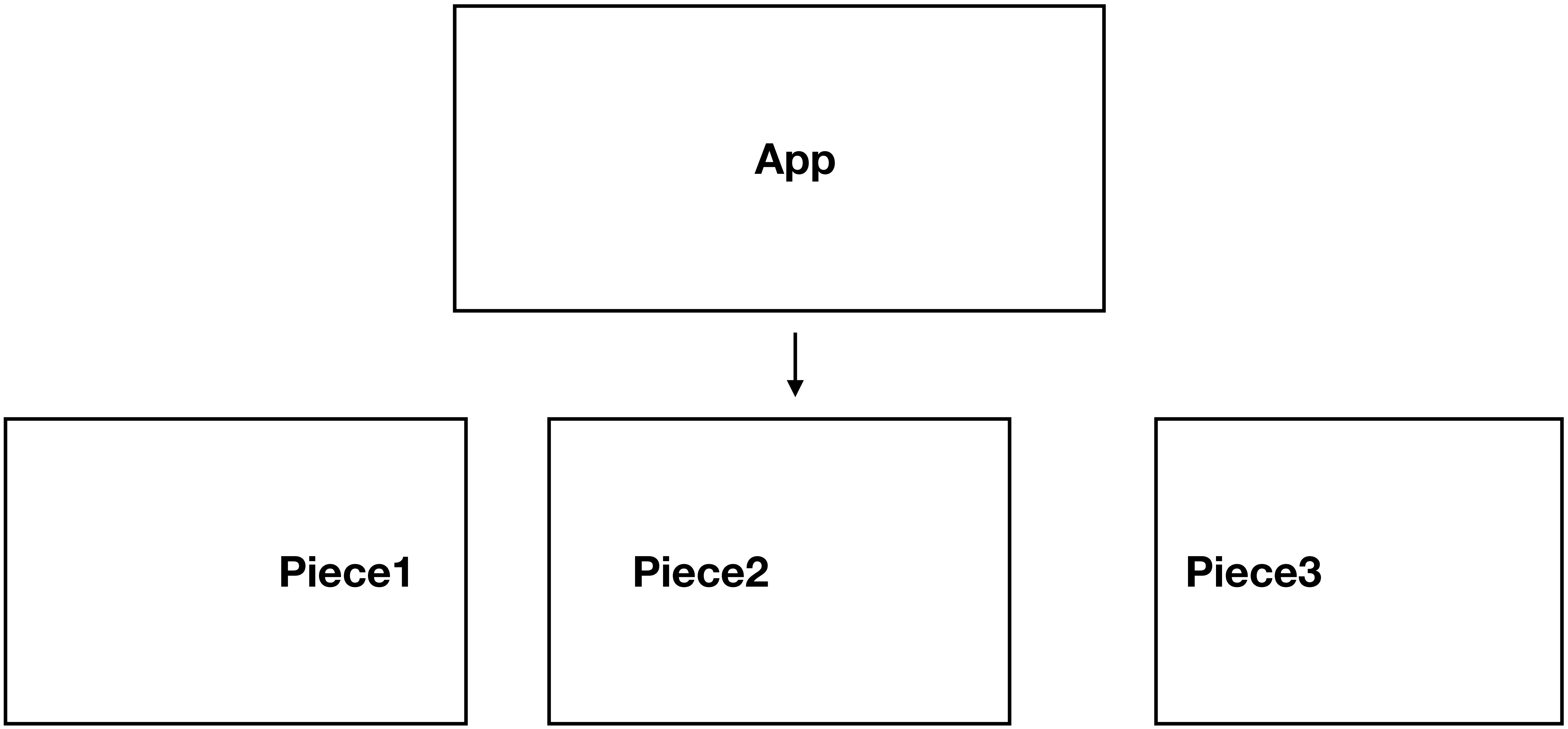
- Quality attribute workshop
- Process to collect arch requirements
- Process to collect Quality Attribute scenario(NFR) | user story (FR)

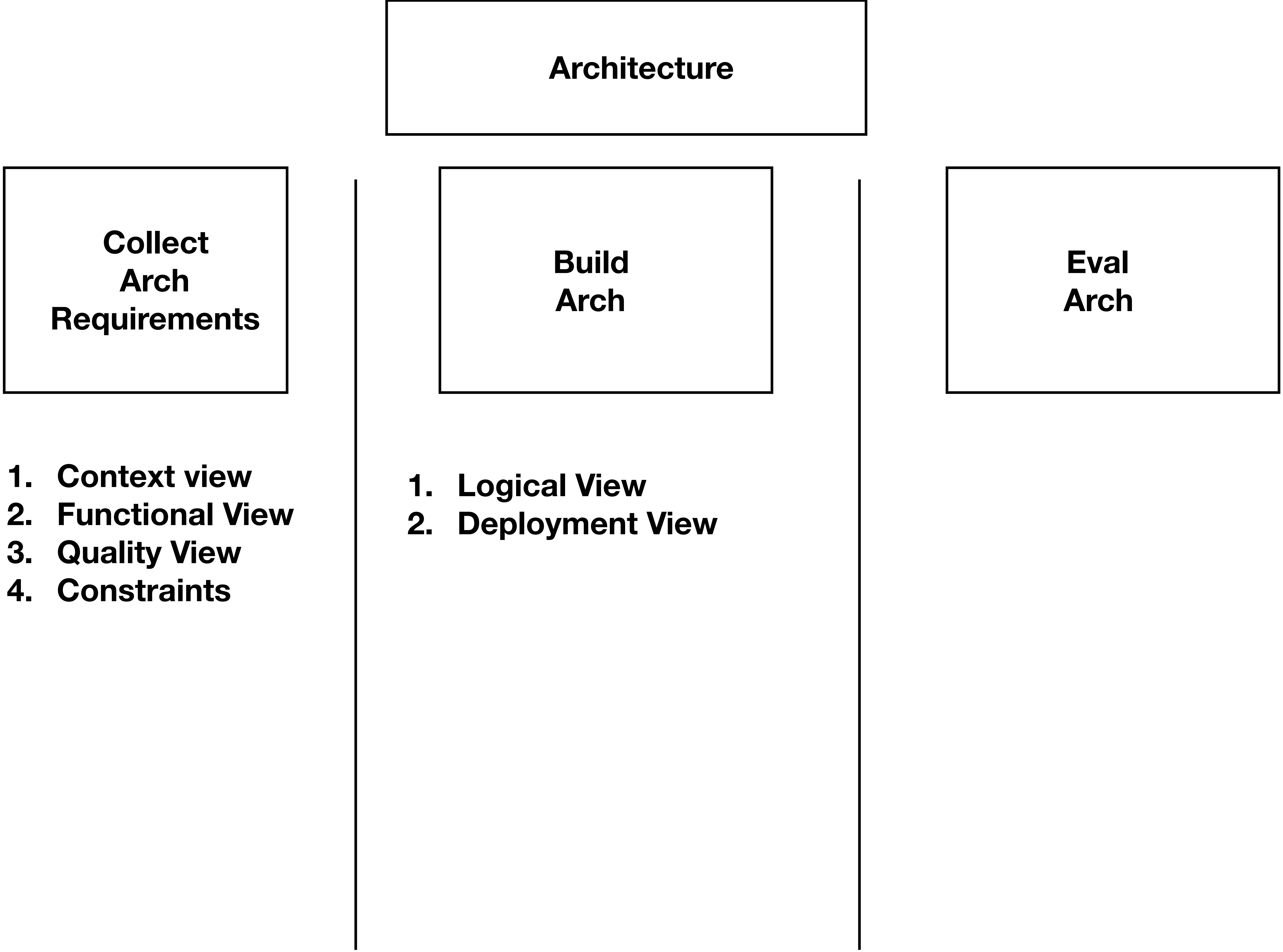
1. Prepare seed Quality Attribute scenarios (NFR)
2. Get all stake holders in to a 1/2 day brainstorming session for NFR.
3. Collect Scenarios
4. Prioritise Scenarios

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

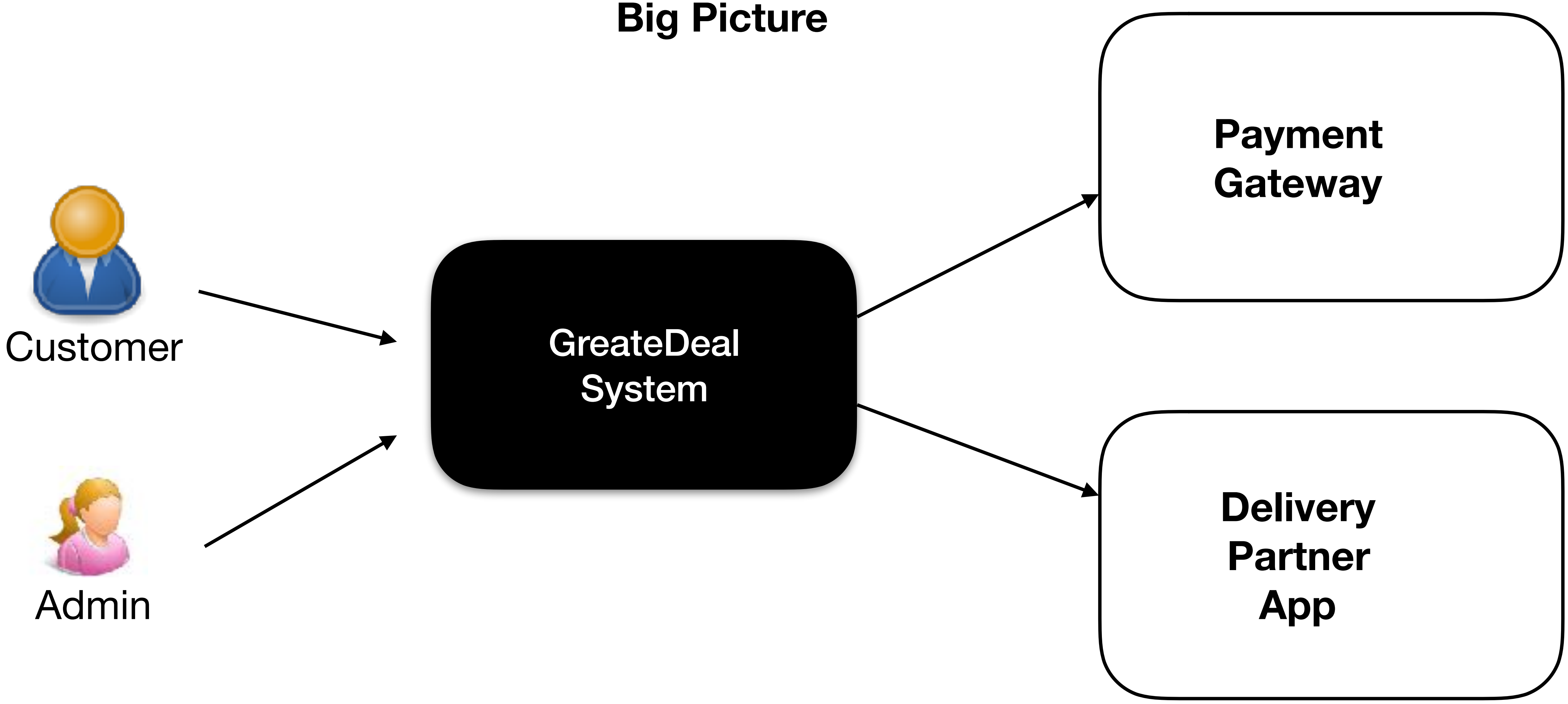
1. Logical View
2. Deployment View

Eval Arch

GreatDeal
Arch
Requirements Gathering

Context View

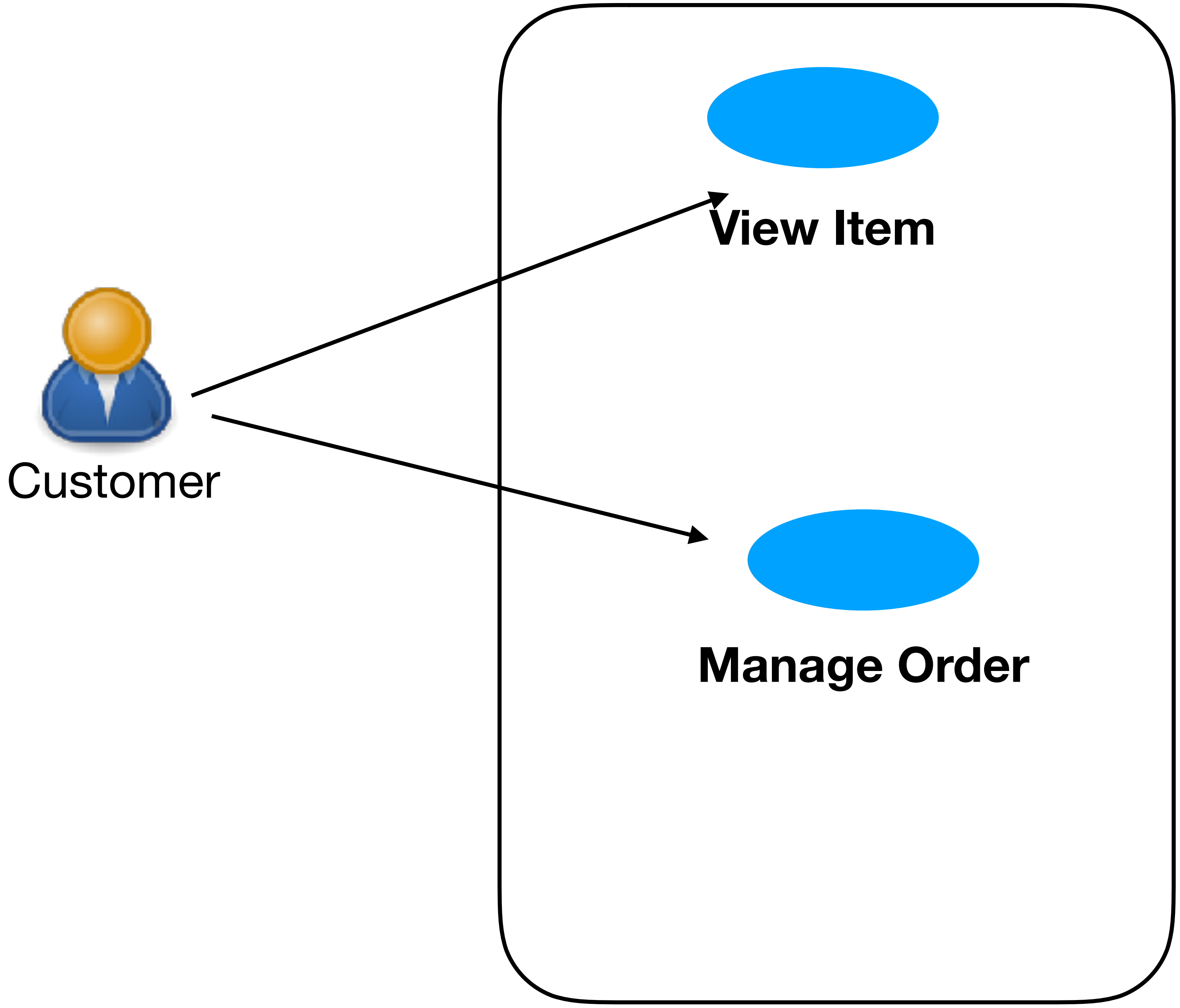
Black Box View
Big Picture



- 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 ?

Functional View

Key functionality of the 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

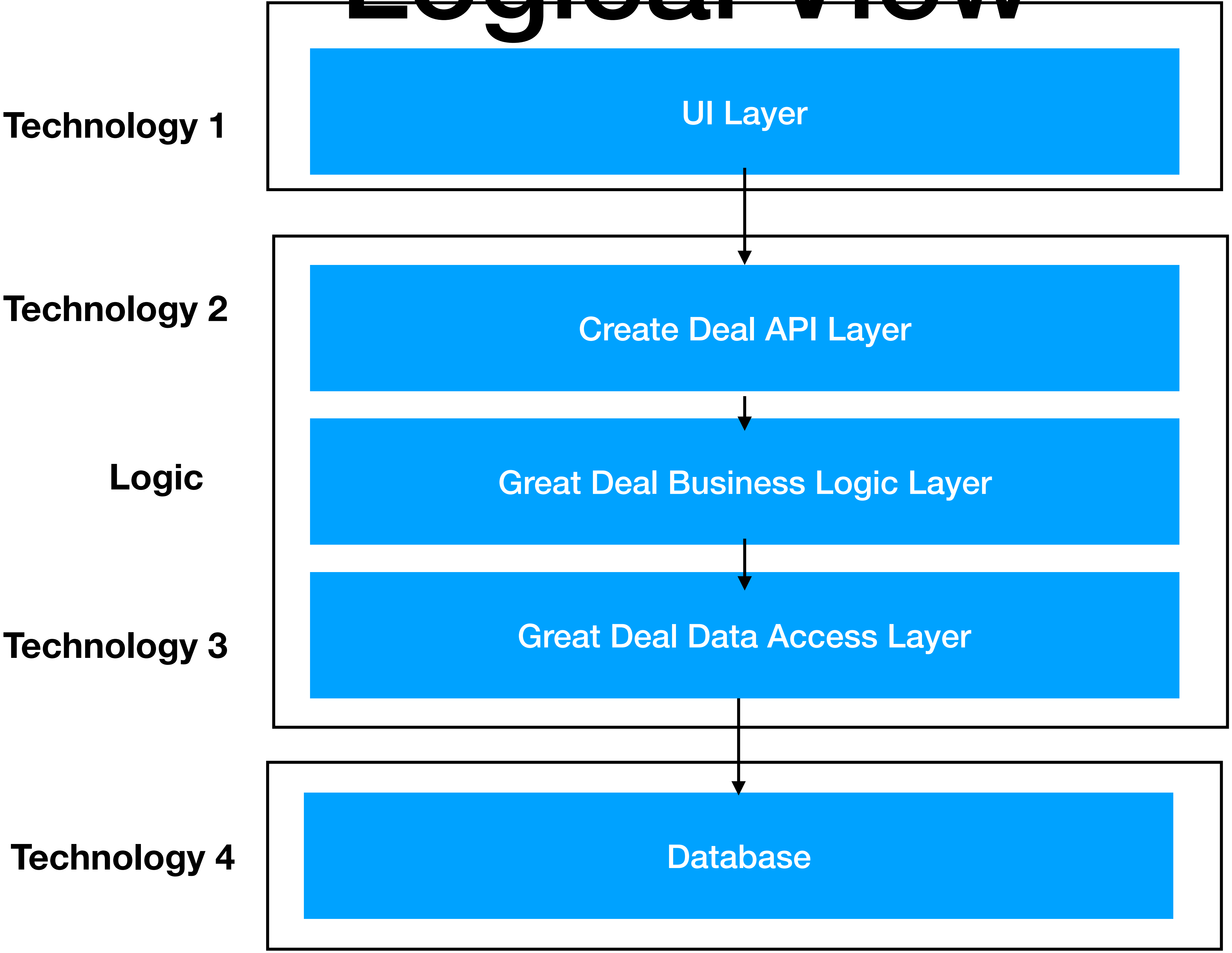
- As a User I want to view the Deal of the Day when 100,000 users are using the portal. The portal displays the Item In < 1 sec time. (performance)
- When a user places an order, the payment fails in the server during peak hours and the order is cancelled and money is refunded within 2 hours. (reliability)
- 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



Technology 1

UI Layer

Technology 2

Create Deal API Layer

Logic

Great Deal Business Logic Layer

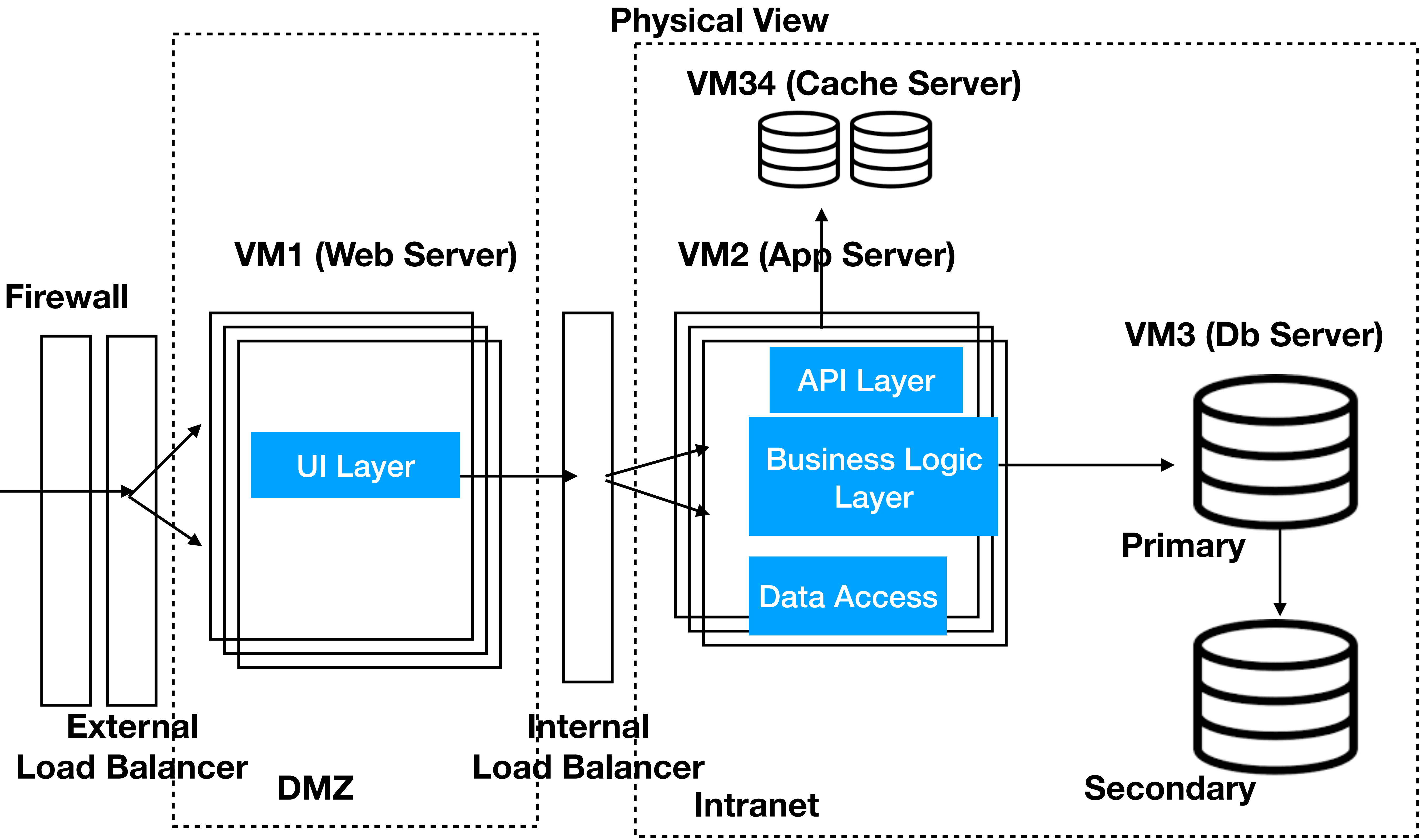
Technology 3

Great Deal Data Access Layer

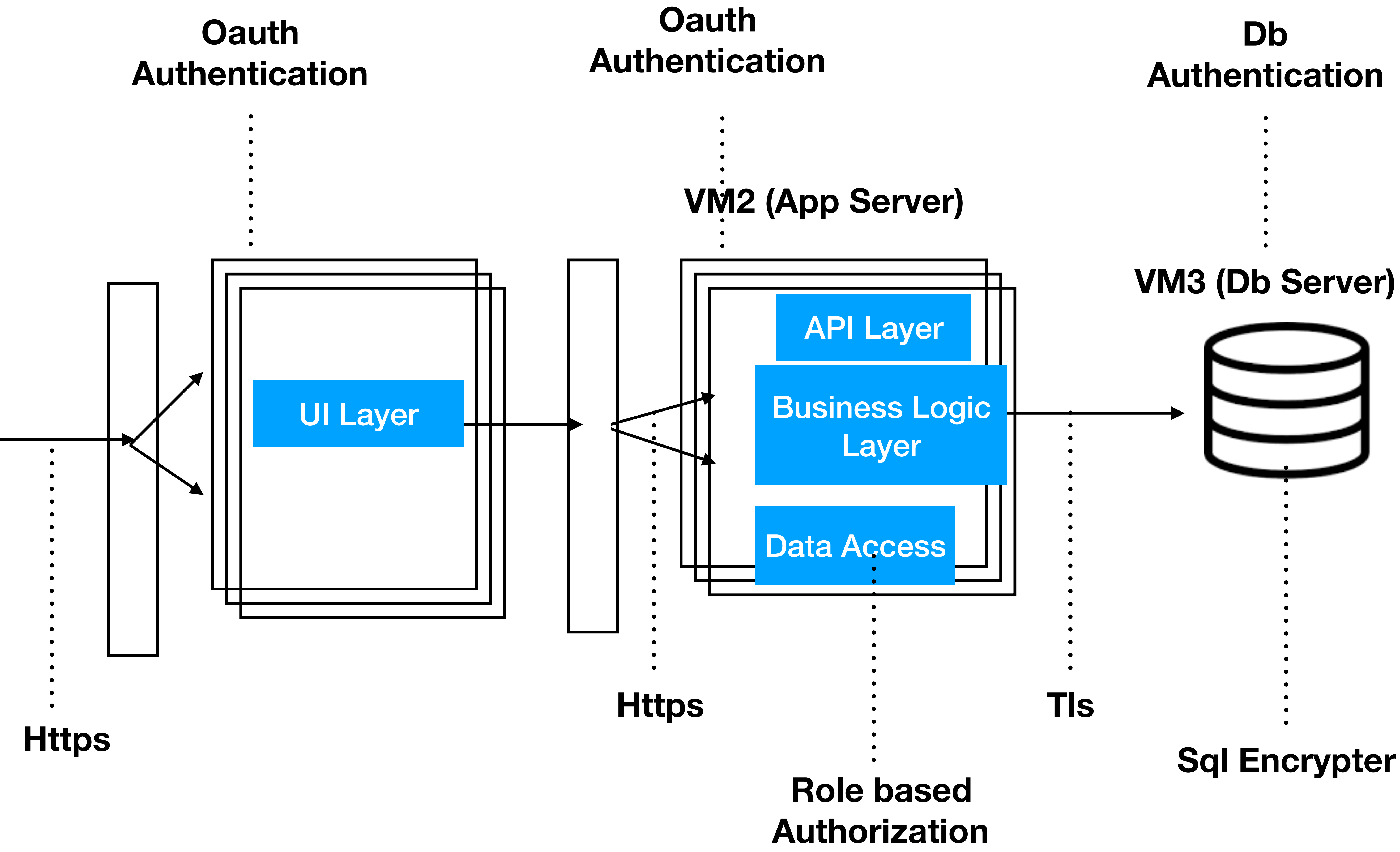
Technology 4

Database

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

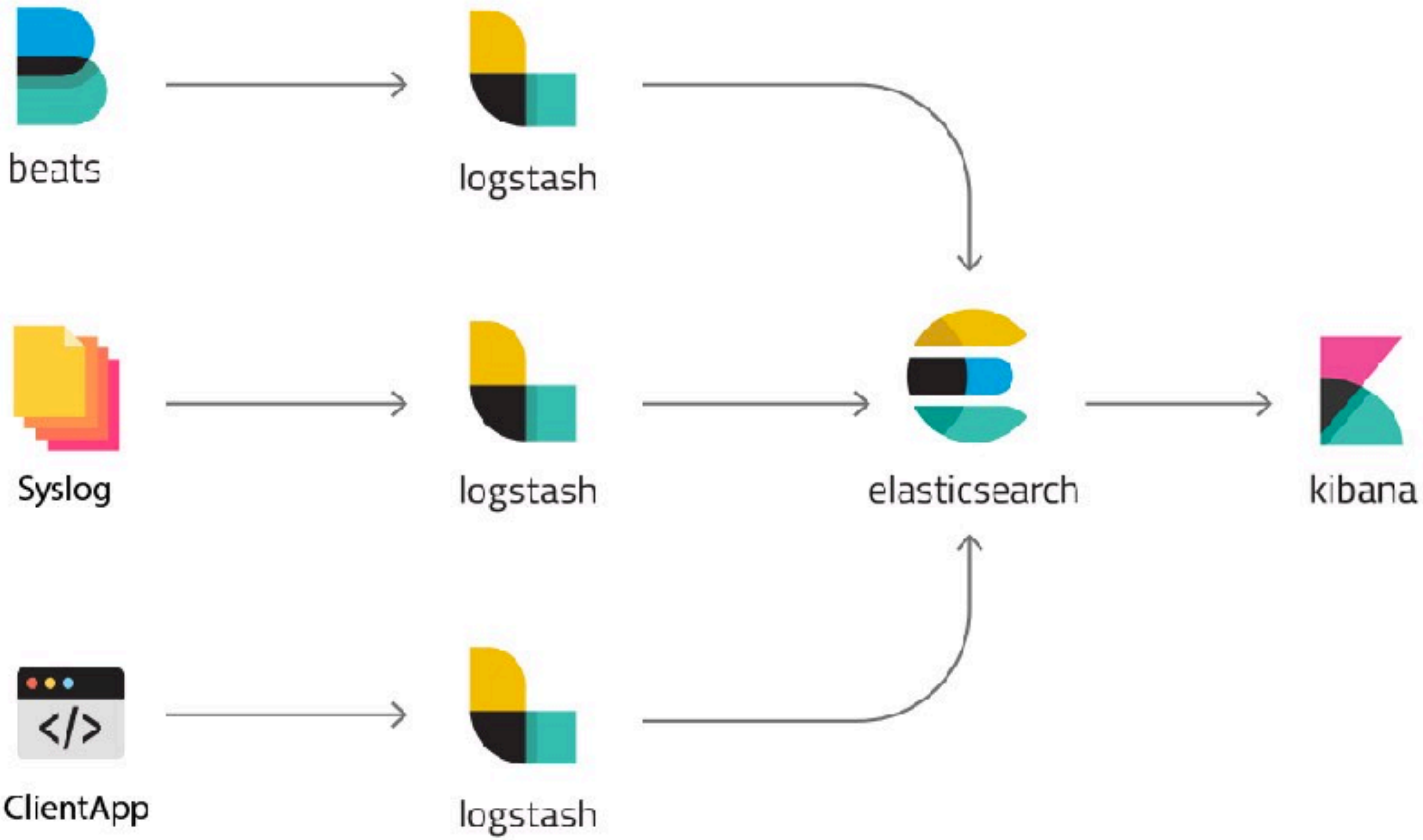
Performance Security

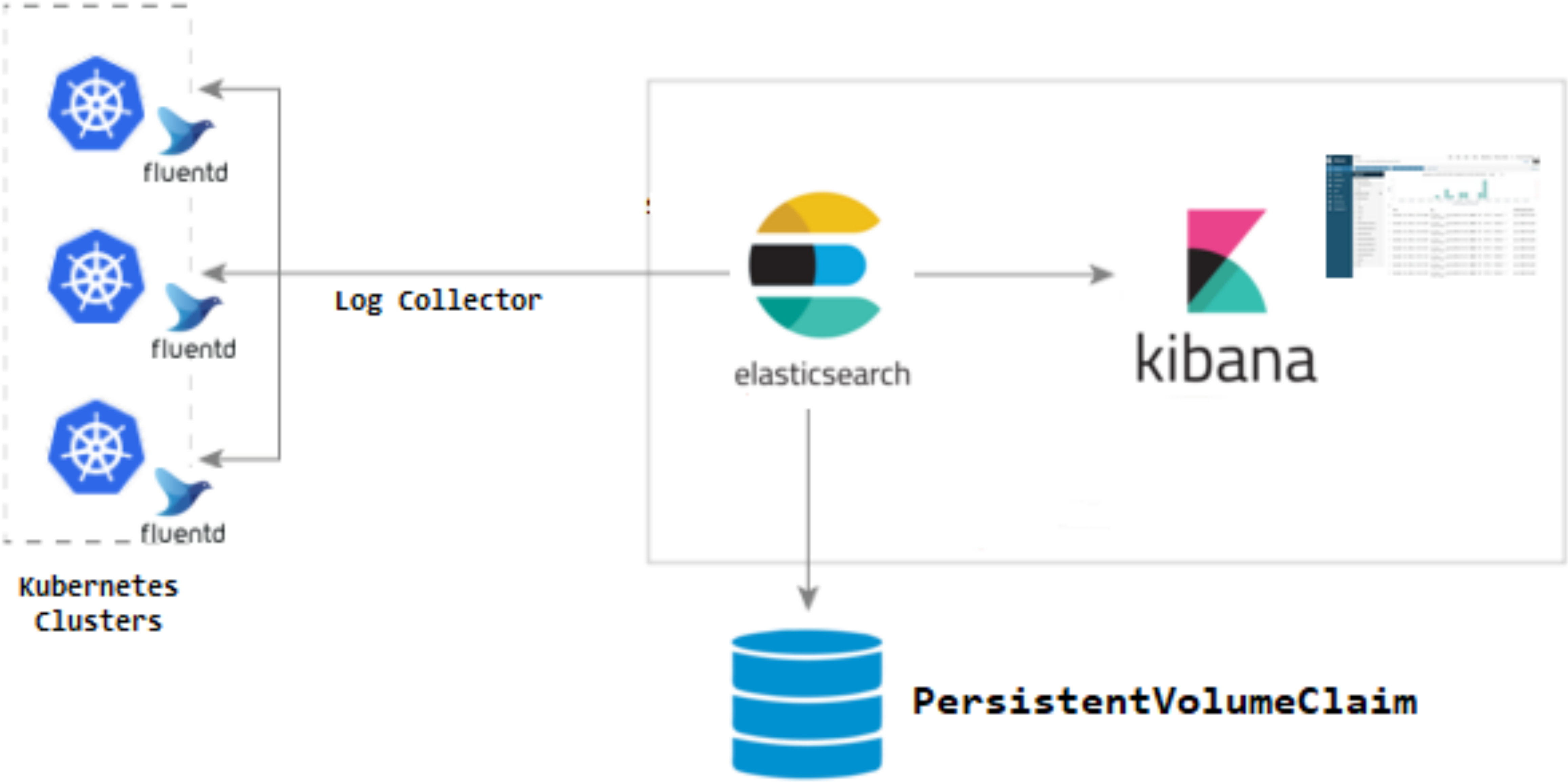
S1 S3 S4

# analyse Scenario -> Approach

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

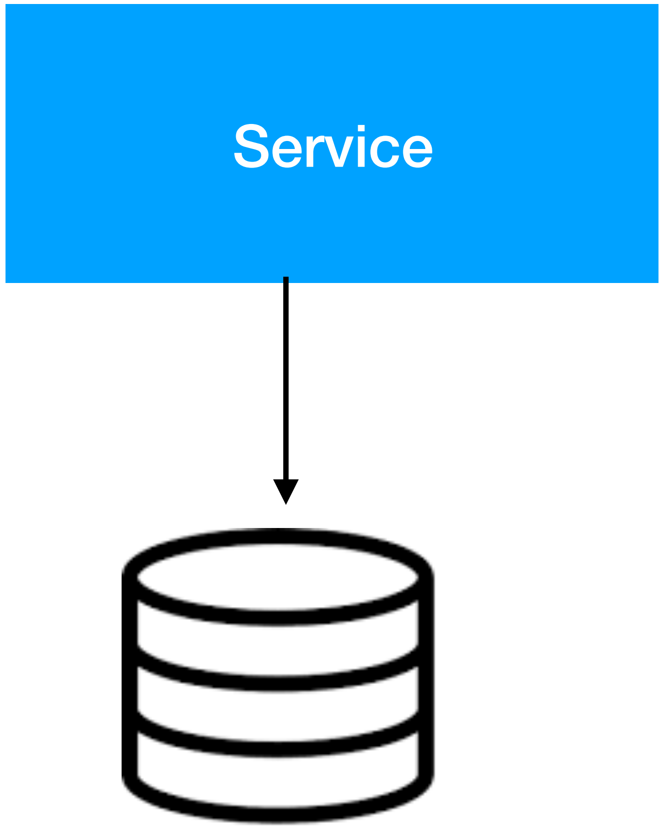
# brainstorm for scenarios


beats
logstash
Syslog
logstash
elasticsearch
kibana
ClientApp
logstash


fluentd
fluentd
fluentd
Kubernetes Clusters
Log Collector
elasticsearch
kibana
PersistentVolumeClaim

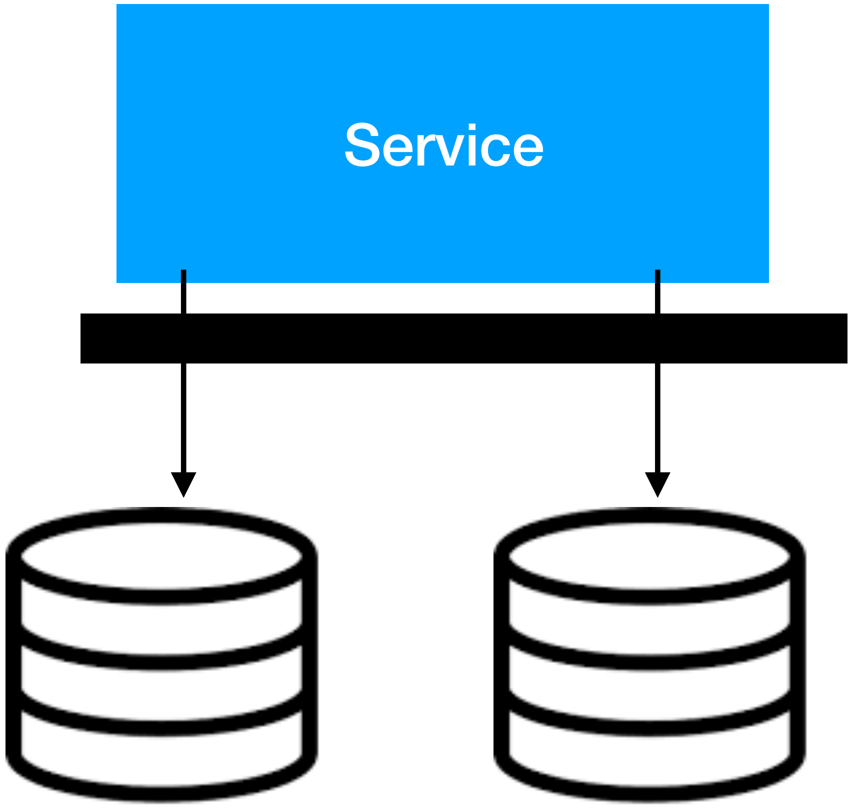
Transaction

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

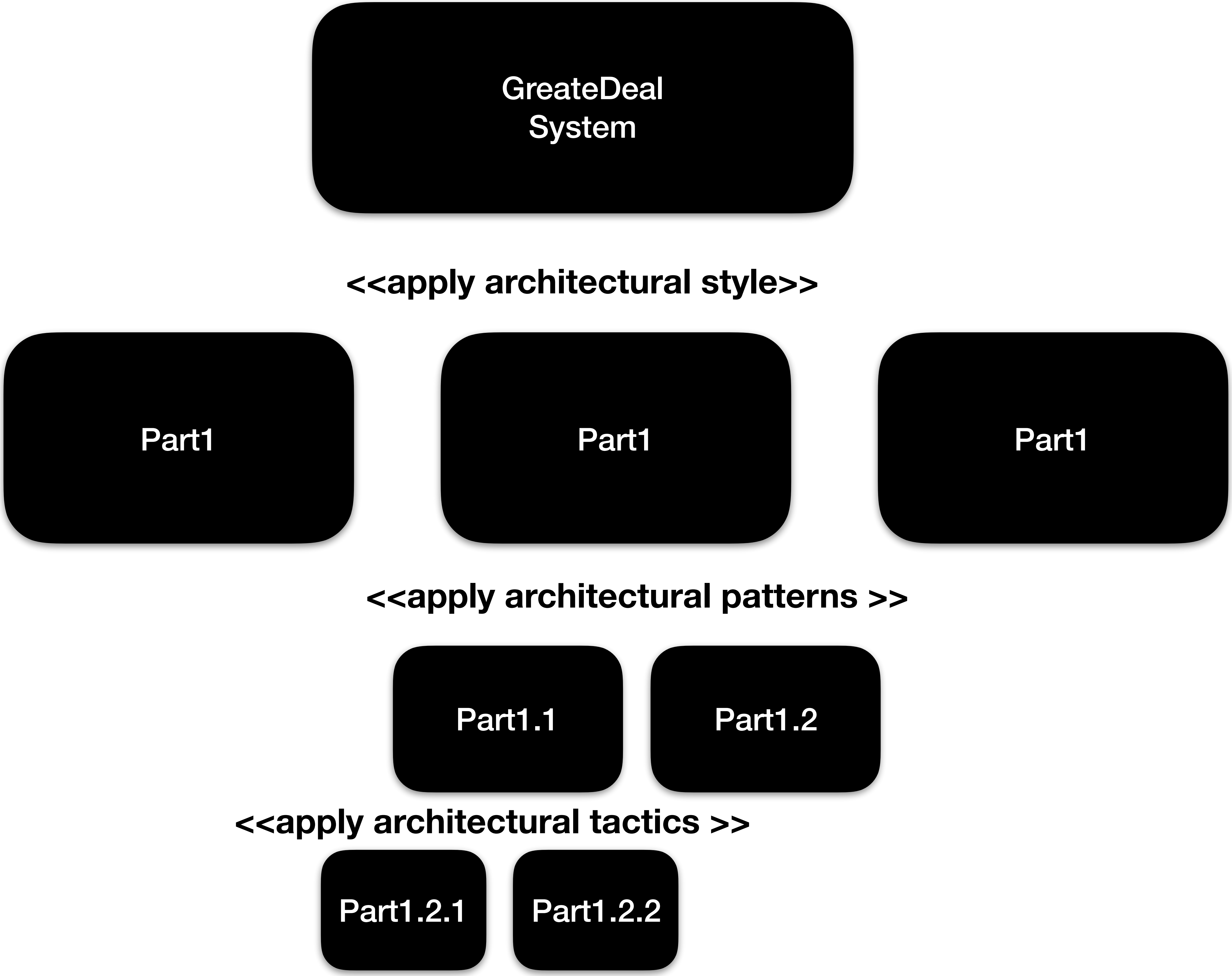
- AAA
- STRIDE
    - Spoofing
    - Tampering
    - Repudation
    - Info disclosure
    - Denial of service
    - Elevation of prove
- Authentication
    - By what you know (pwd,key,scret)
    - By what you have (otp,rsa,email,..)
    - By what you are (face,retina,voice,
-

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 >>

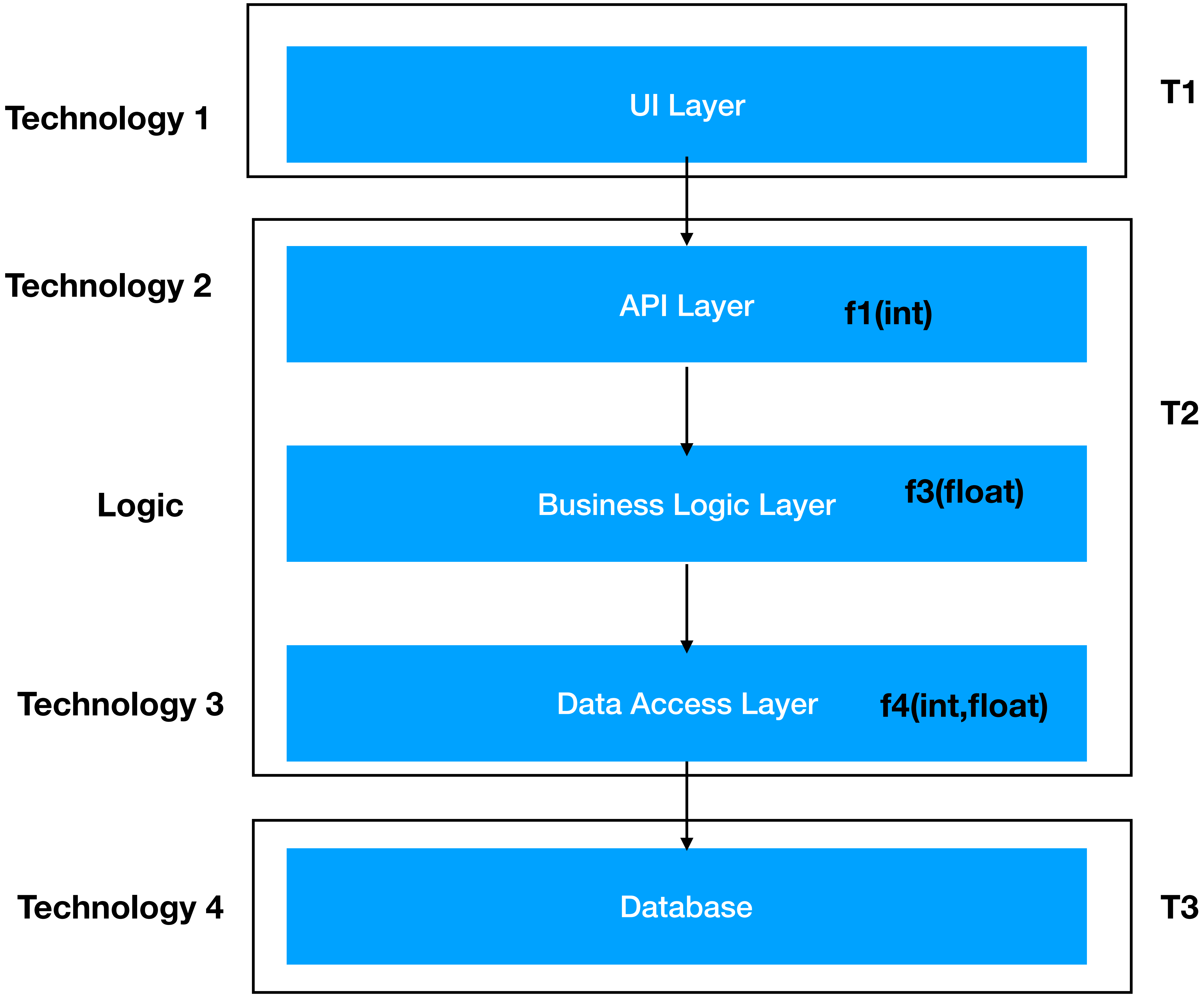
Part1.1

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

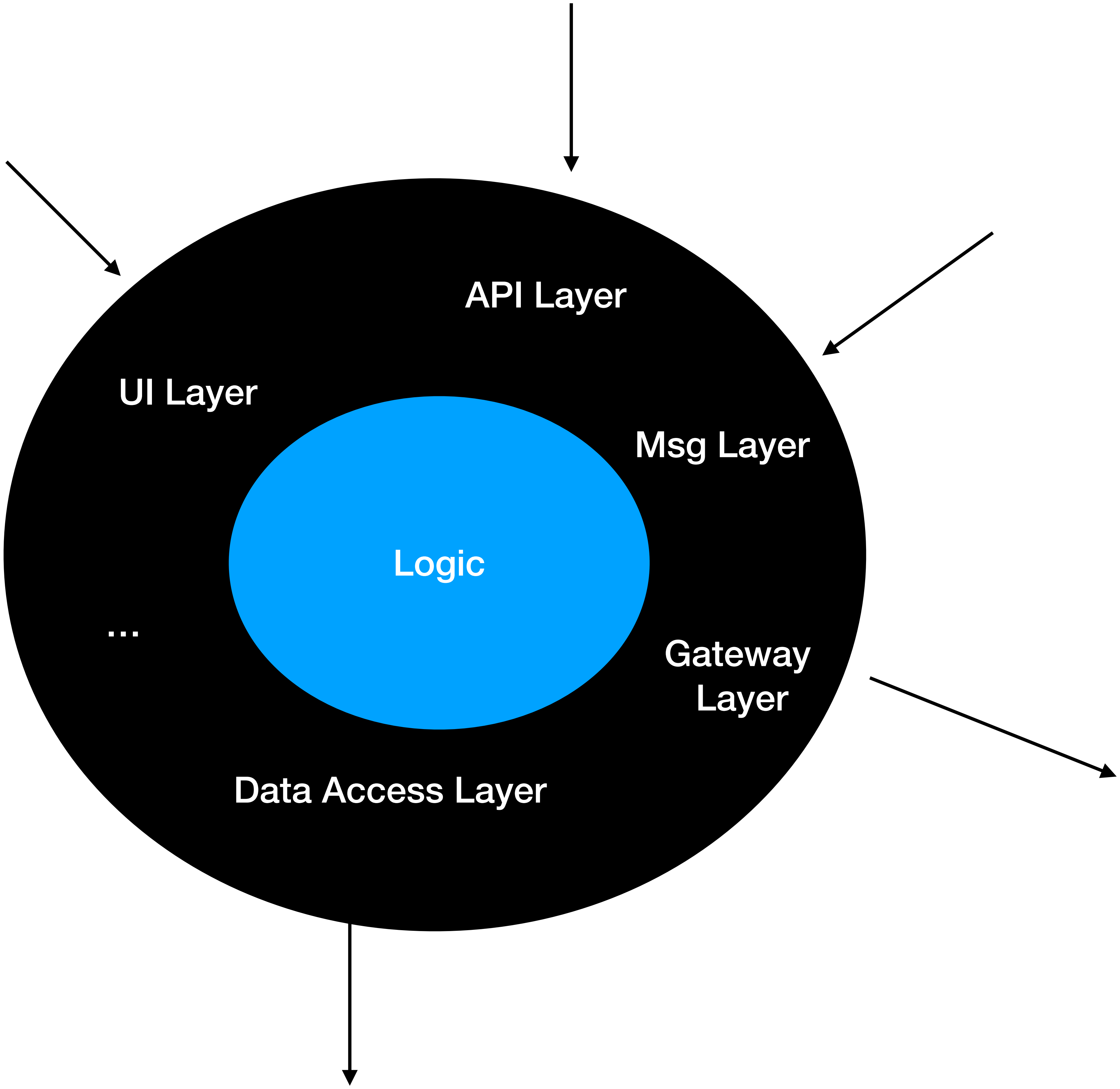
Data Access Layer

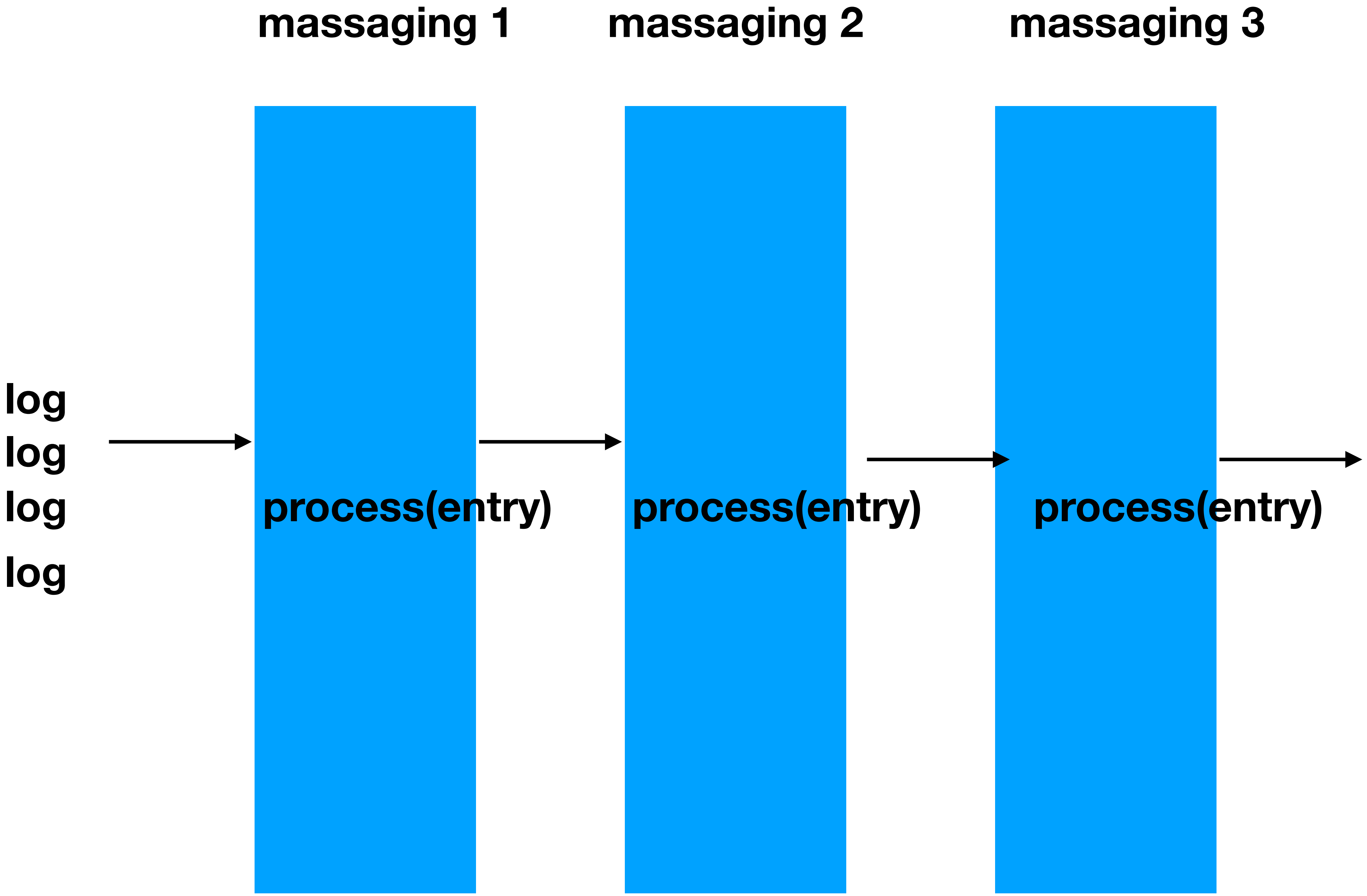
f4(int,float)

Technology 4

Database

T3


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



massaging 1

massaging 2

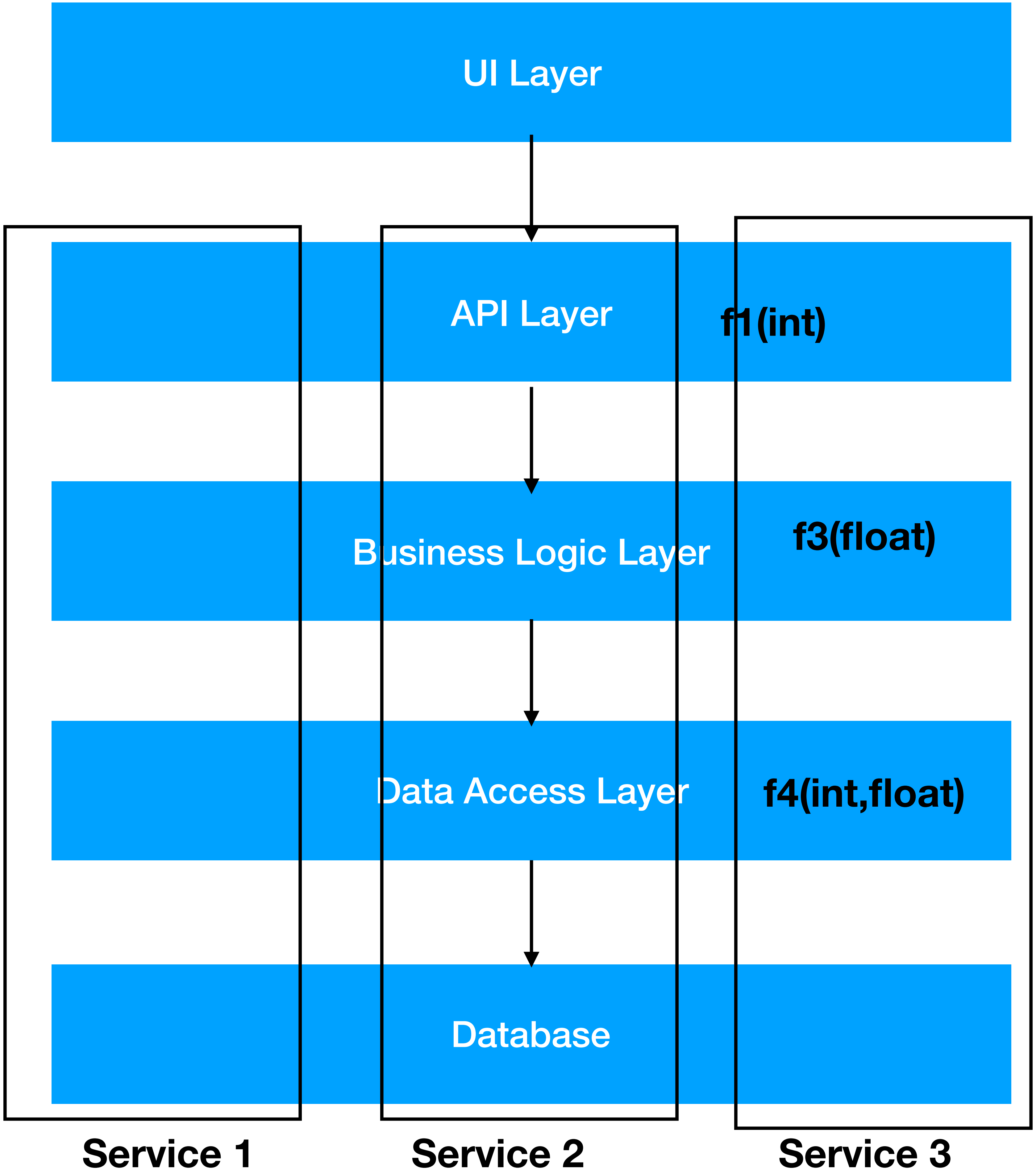
massaging 3

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

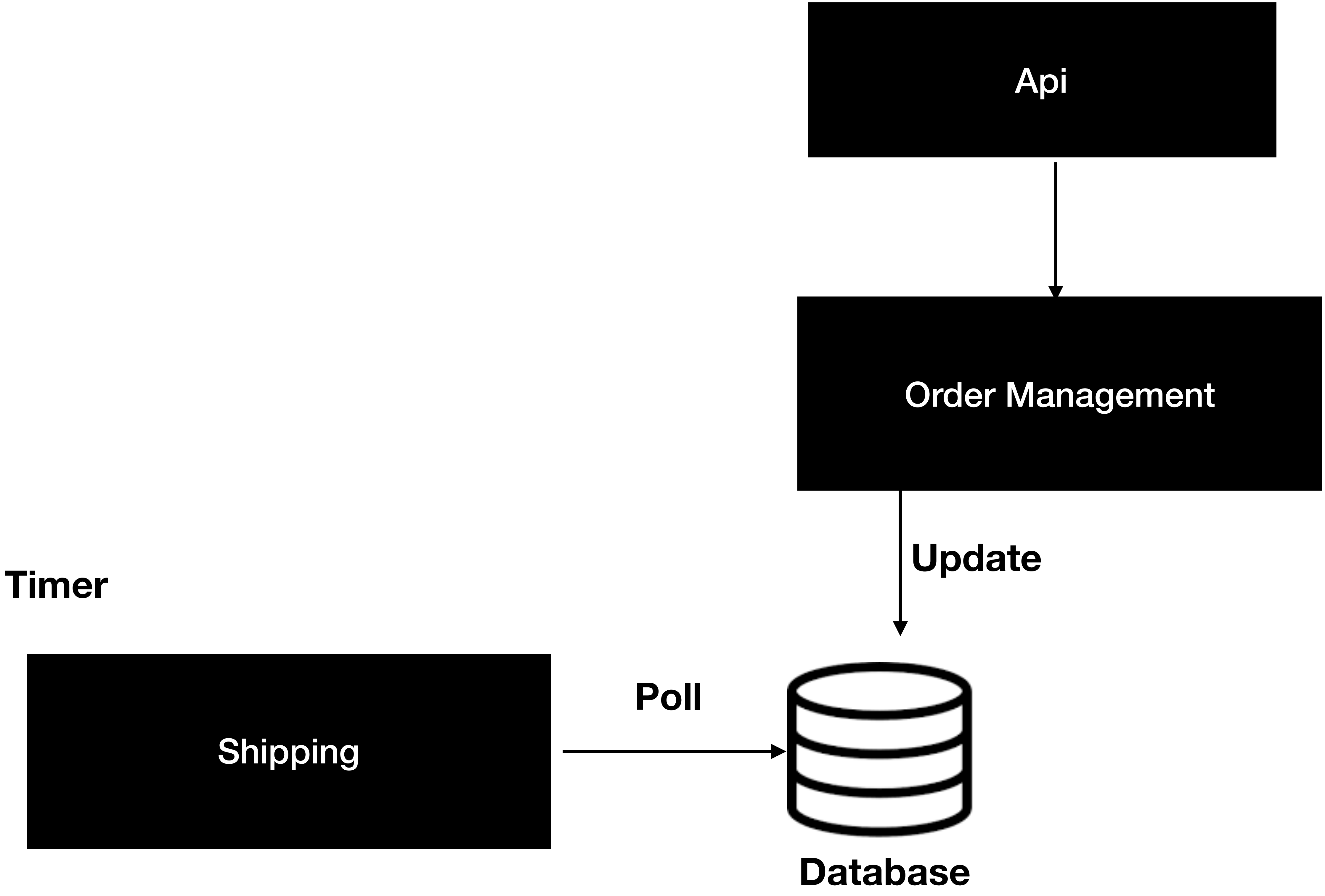
Service 1

Service 2

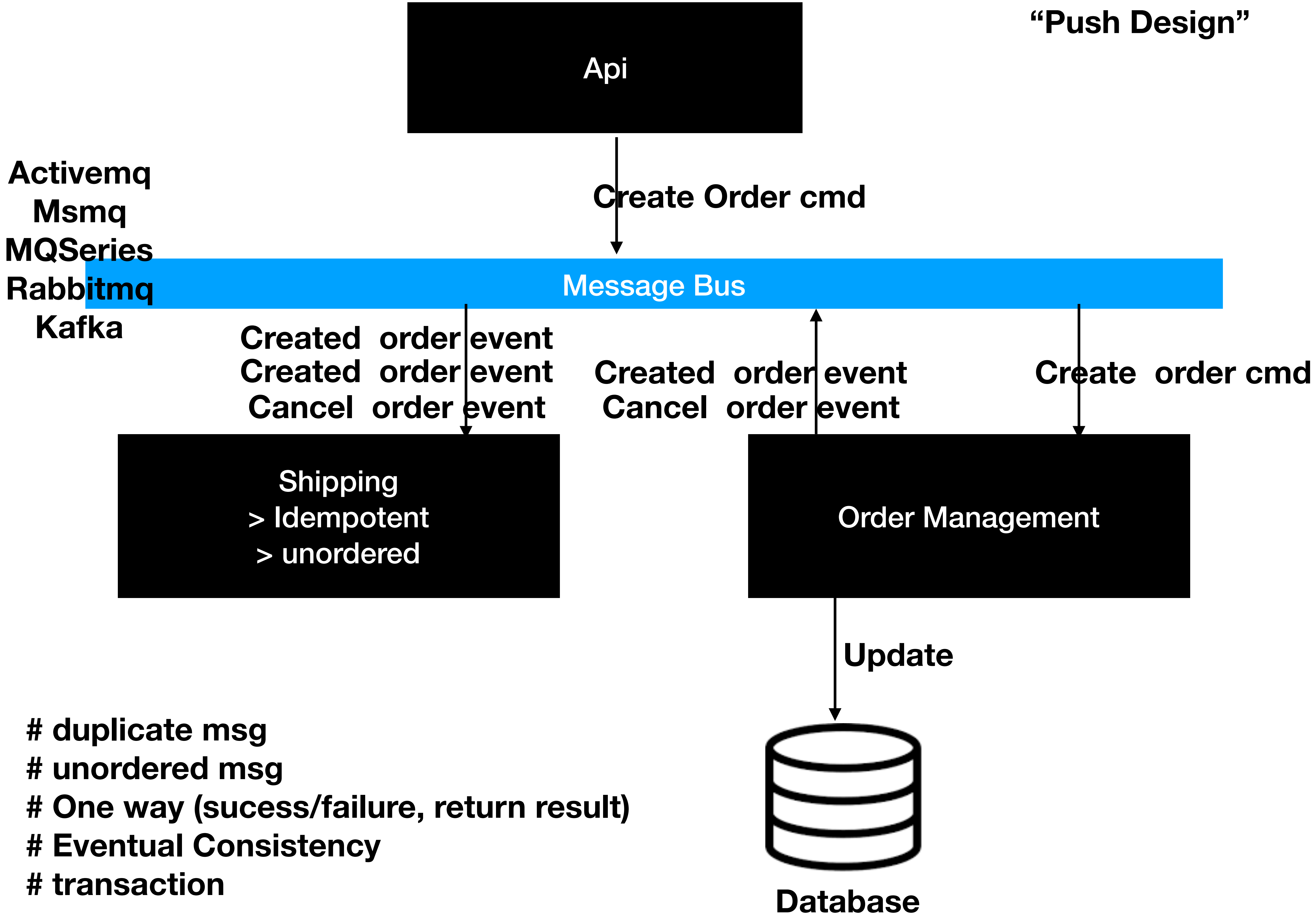
Service 3

“Pull Design”

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”



Activemq
Msmq
MQSeries
Rabbitmq
Kafka

Api

Create Order cmd

Message Bus

Created order event
Created order event
Cancel order event

Shipping
> Idempotent
> unordered

Created order event
Cancel order event

Create order cmd

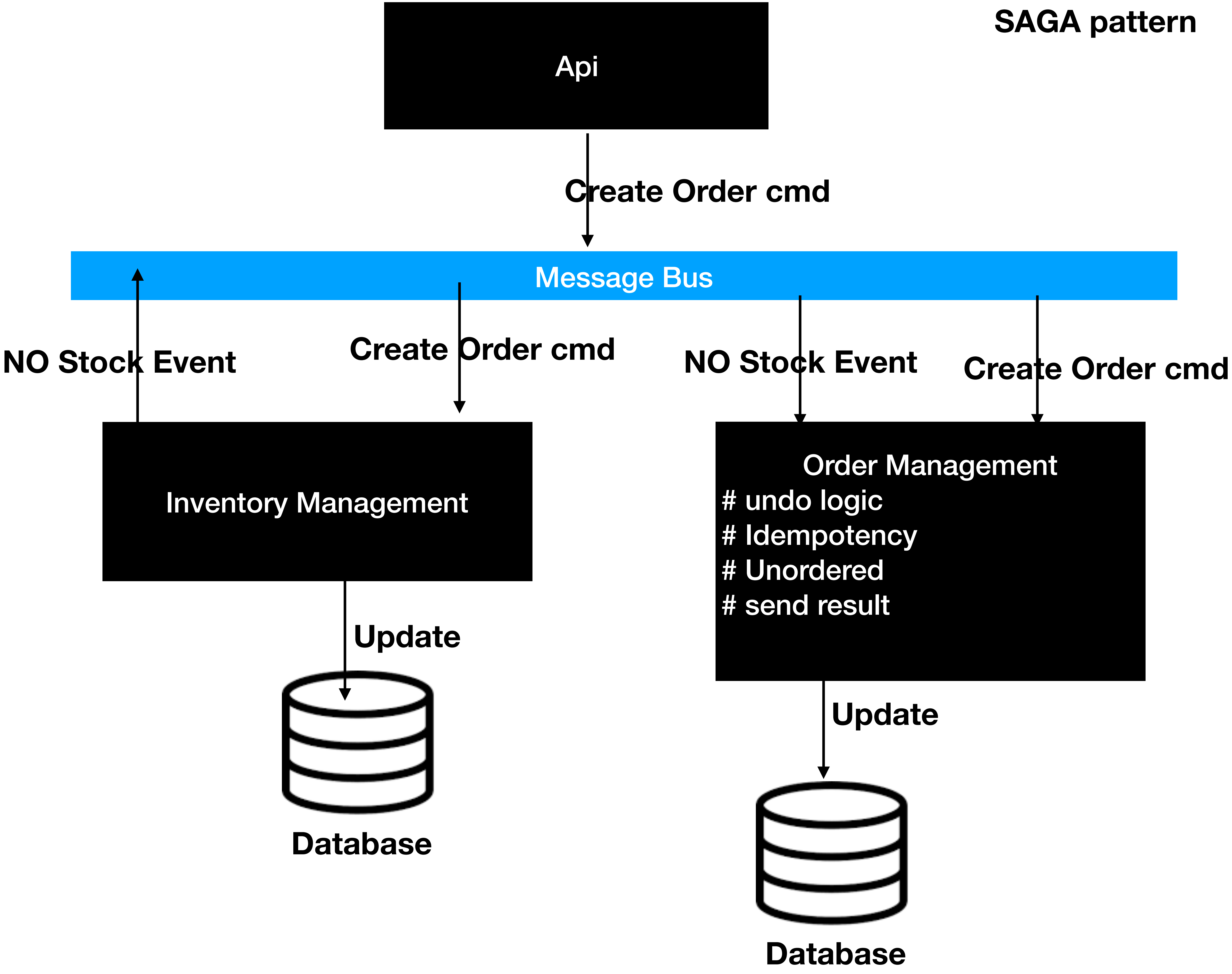
Order Management

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

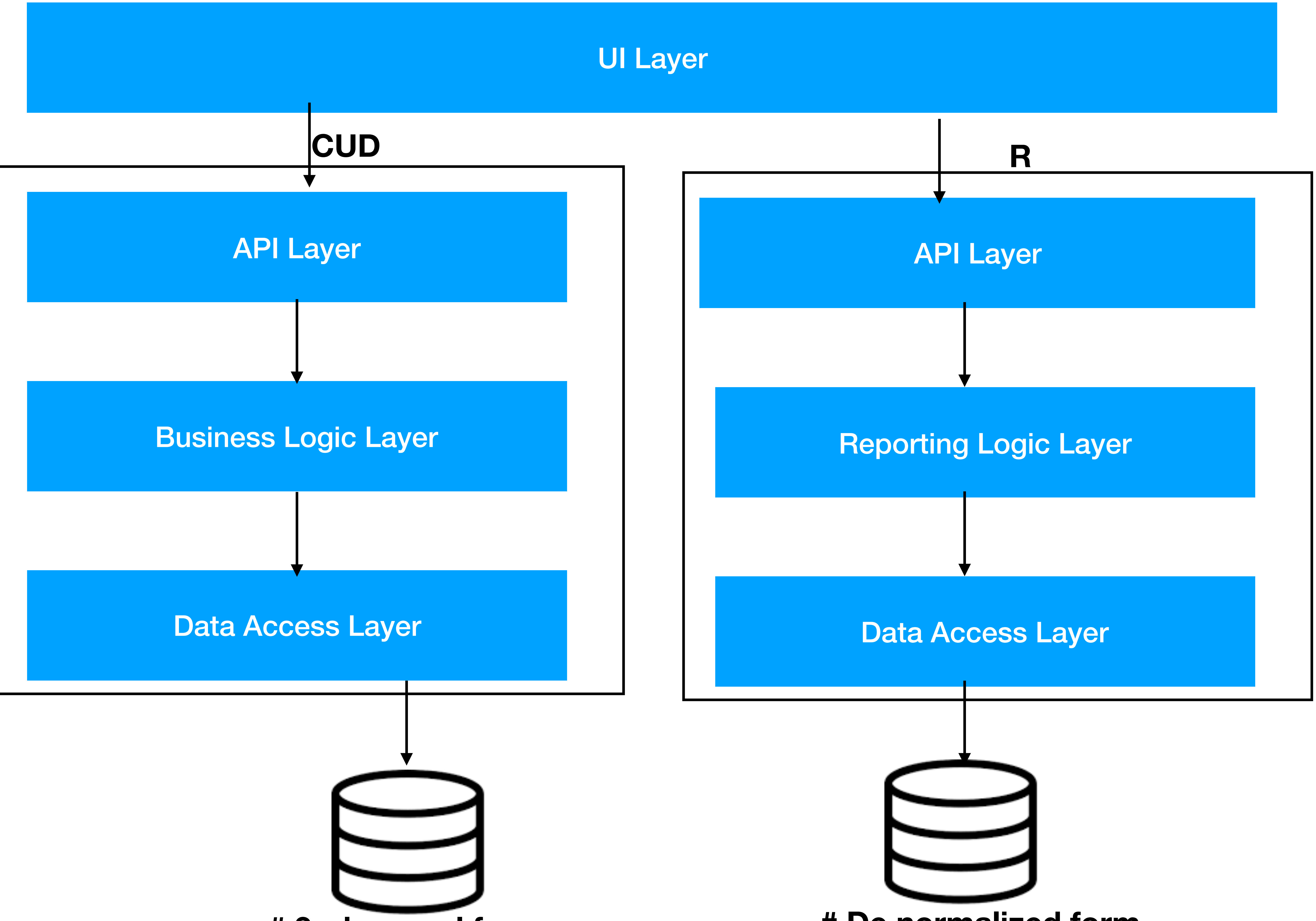
Update

Database

Order Management
# undo logic
# Idempotency
# Unordered
# send result

Update

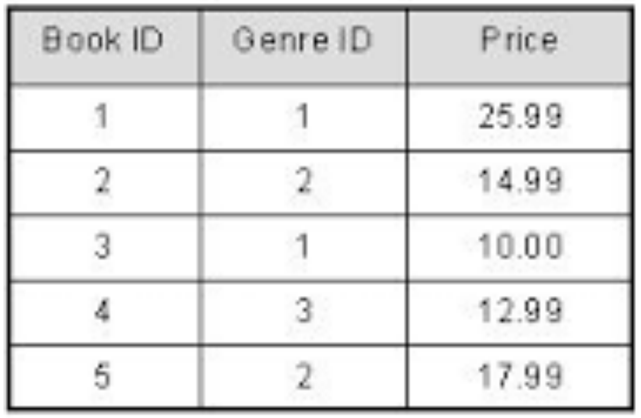
Database



# 3 rd normal form

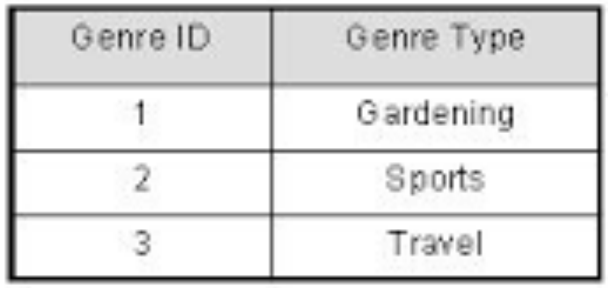
De normalized form

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



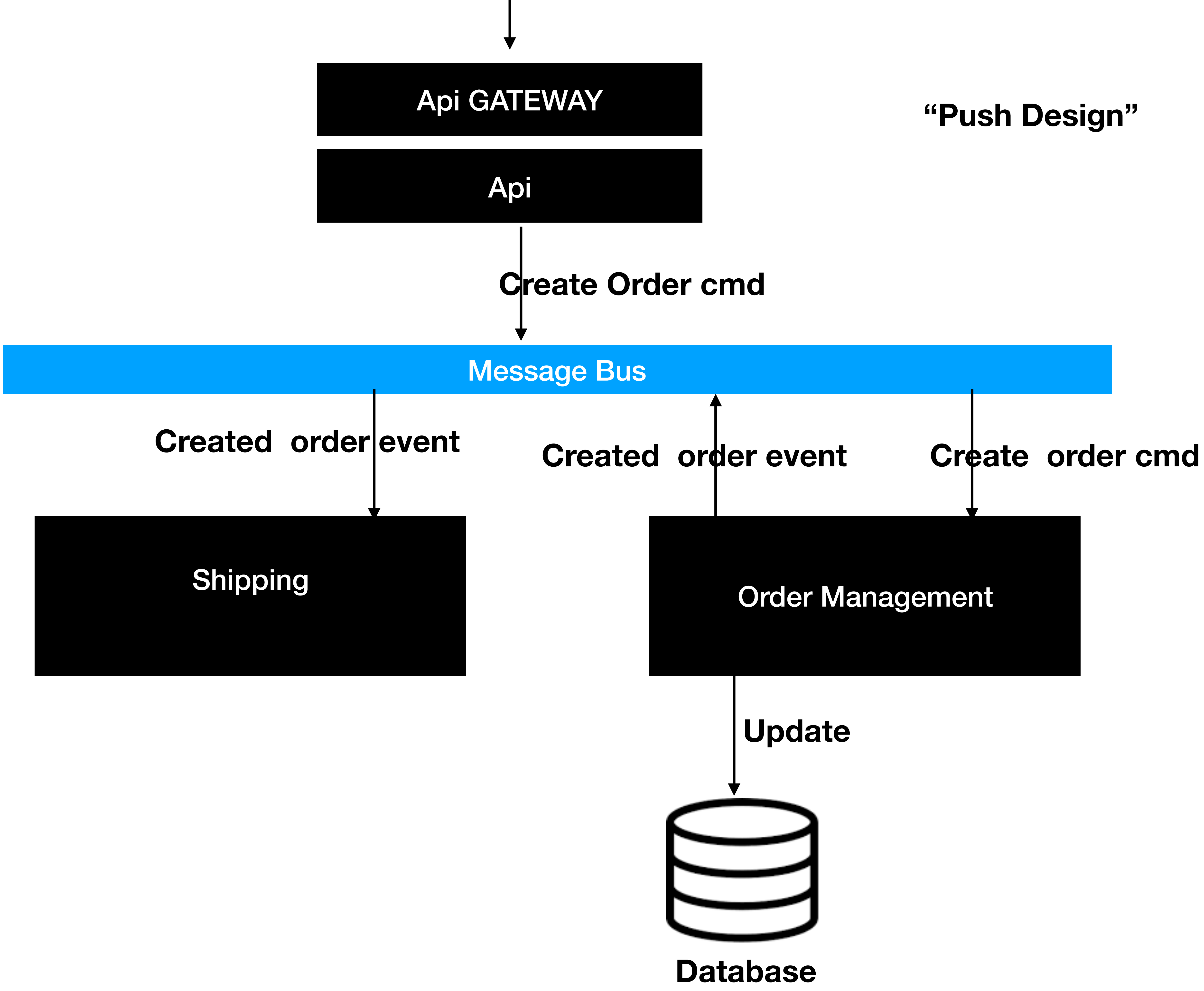
Genre ID	Genre Type
1	Gardening
2	Sports
3	Travel

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

“Push Design”



Api GATEWAY

Api

Create Order cmd

Message Bus

Created order event

Created order event

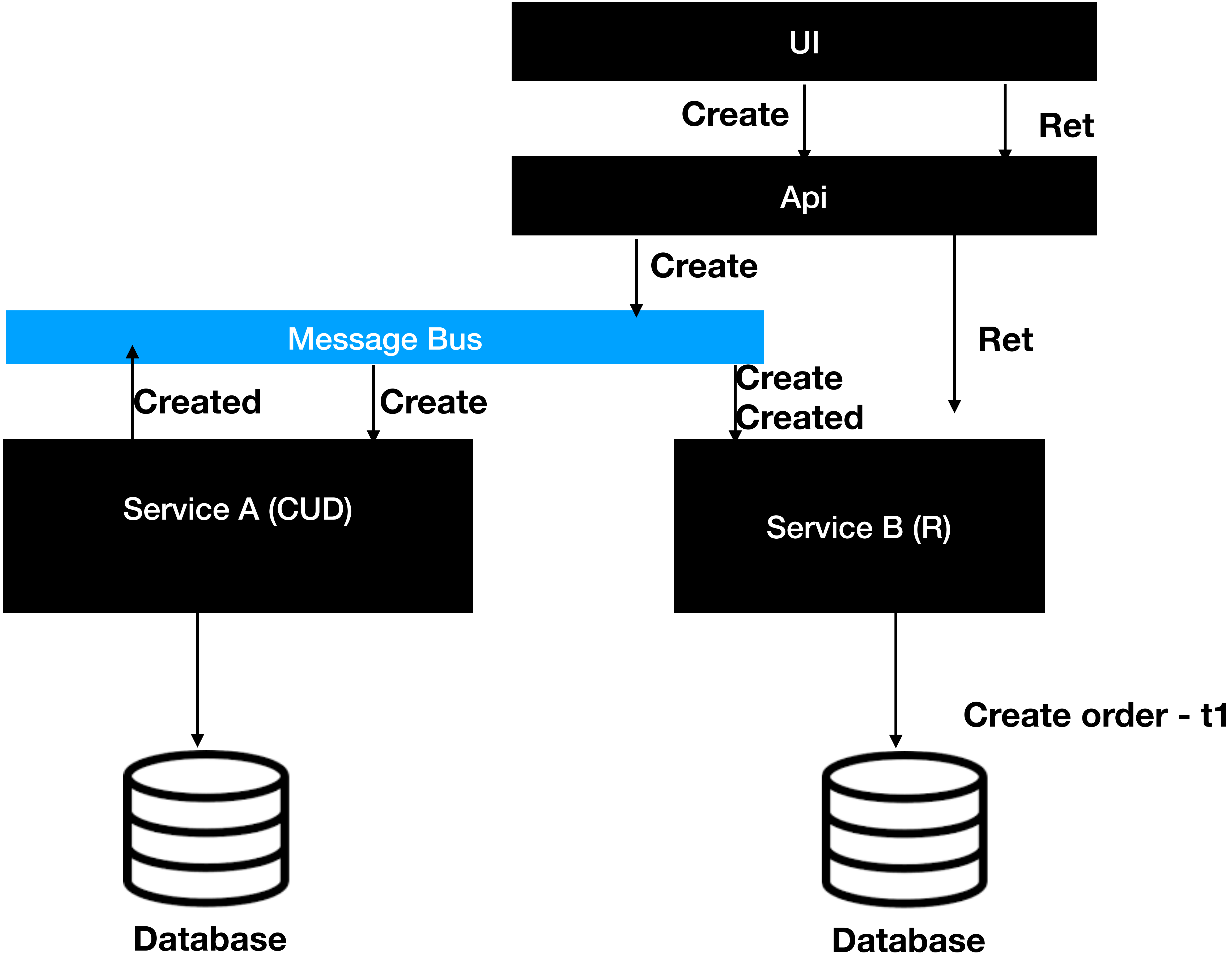
Create order cmd

Shipping

Order Management

Update

Database



UI

Create

Ret

Api

Create

Message Bus

Ret

Created

Create

Create

Created

Service A (CUD)

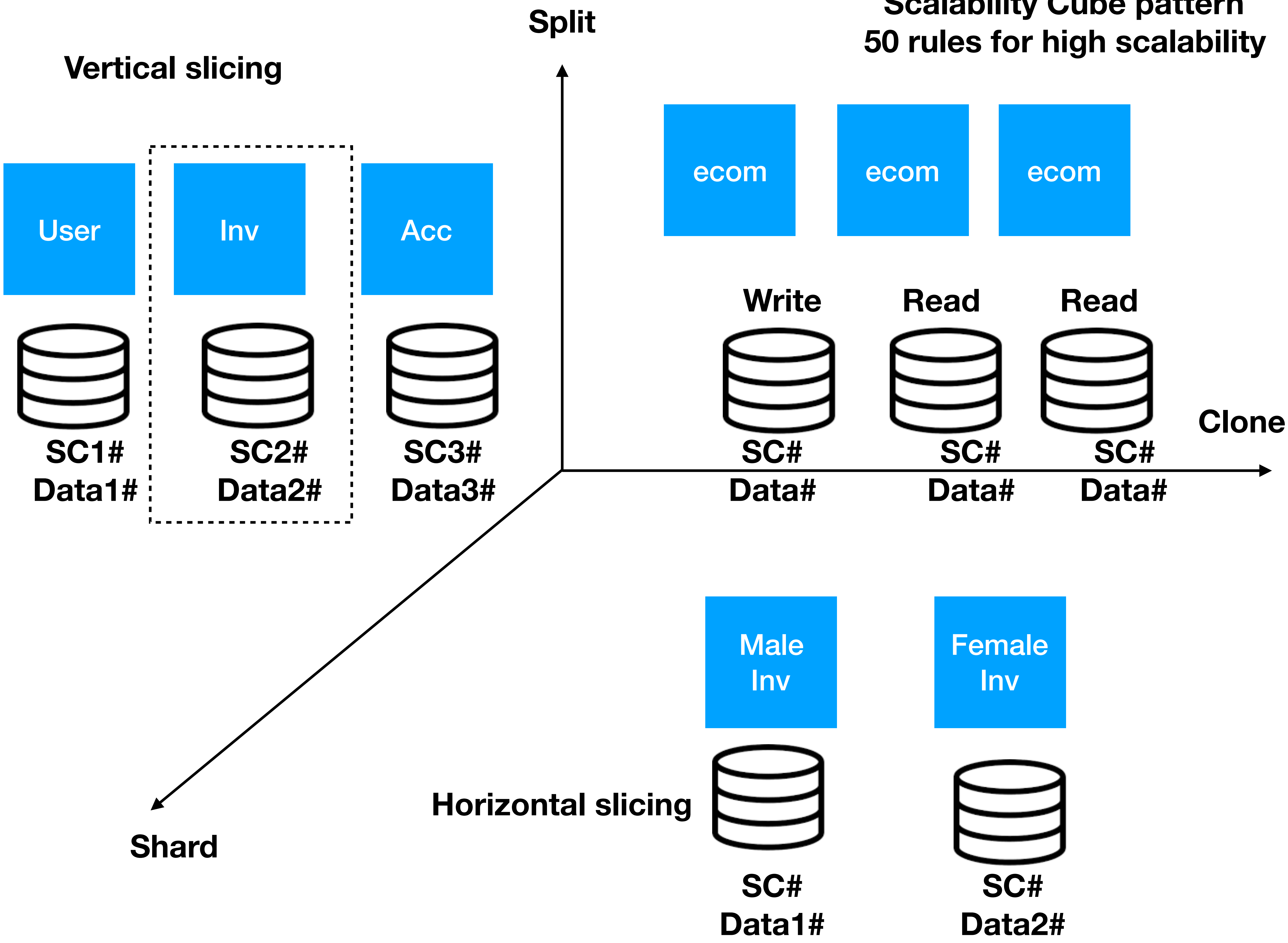
Service B (R)

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

Male
Inv

Female
Inv

Horizontal slicing

SC#
Data1#

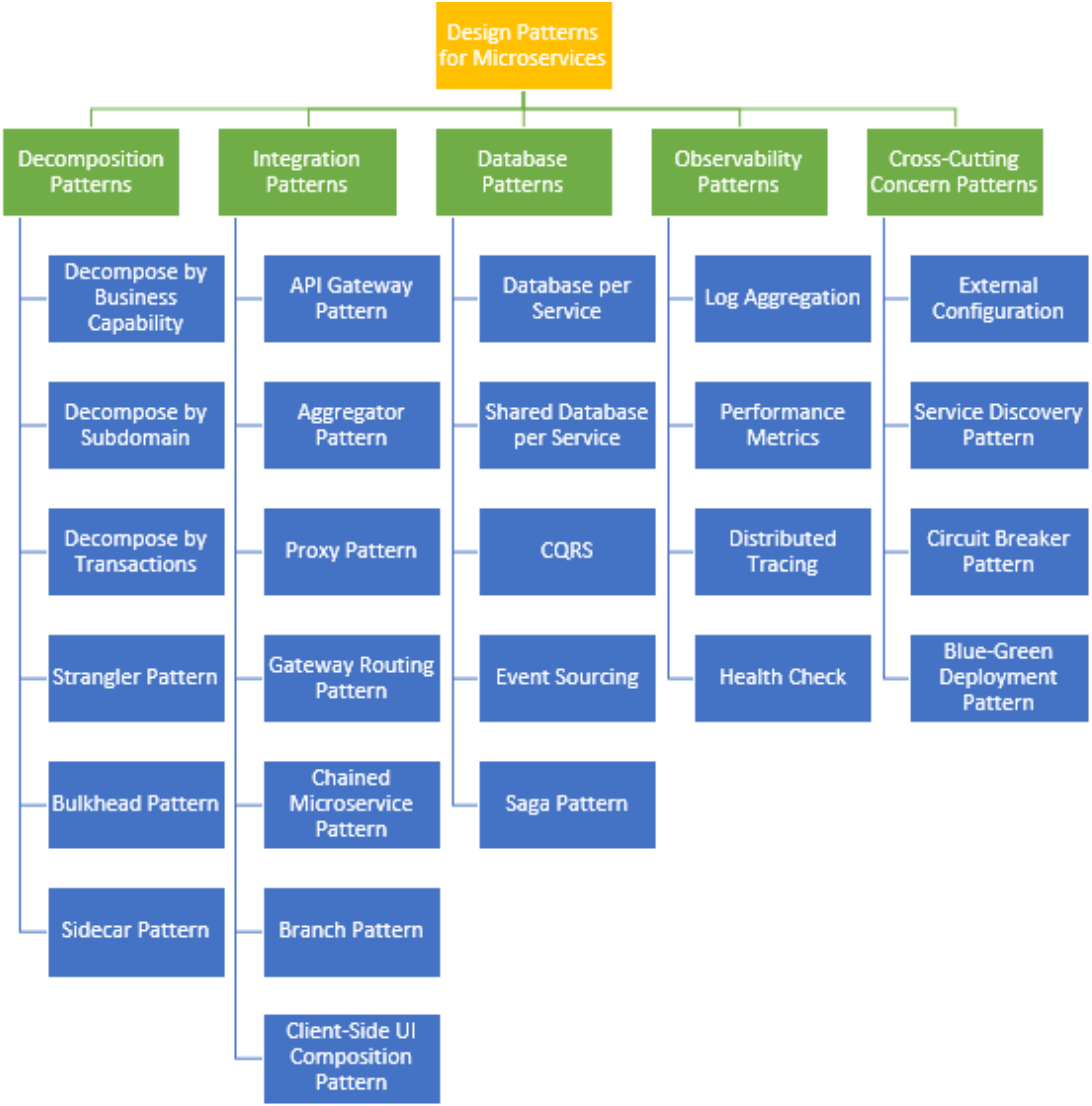
SC#
Data2#

Shard

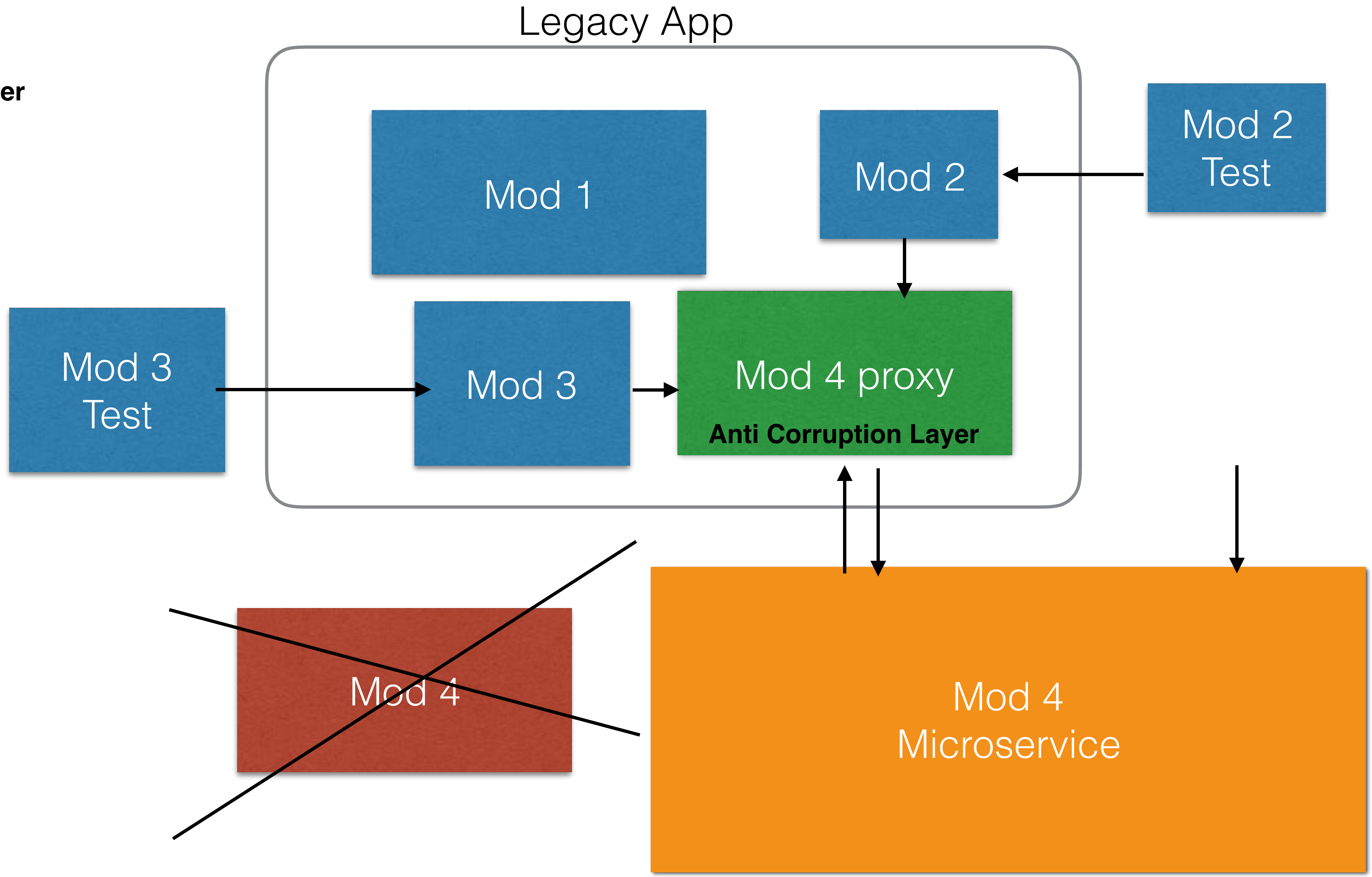
	Monolithic	Micro
Fun Requirements	Shared	Not Shared
Source Control	Shared	Not Shared
Build Server	Shared	Not Shared
Database	Shared	Not Shared
Deployment Infra	Shared	Not Shared
Architecture / Technology	Shared	Not Shared
Test Cases	Shared	Not Shared
SCRUM (team) /Sprint	Shared	Not Shared
Platform /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

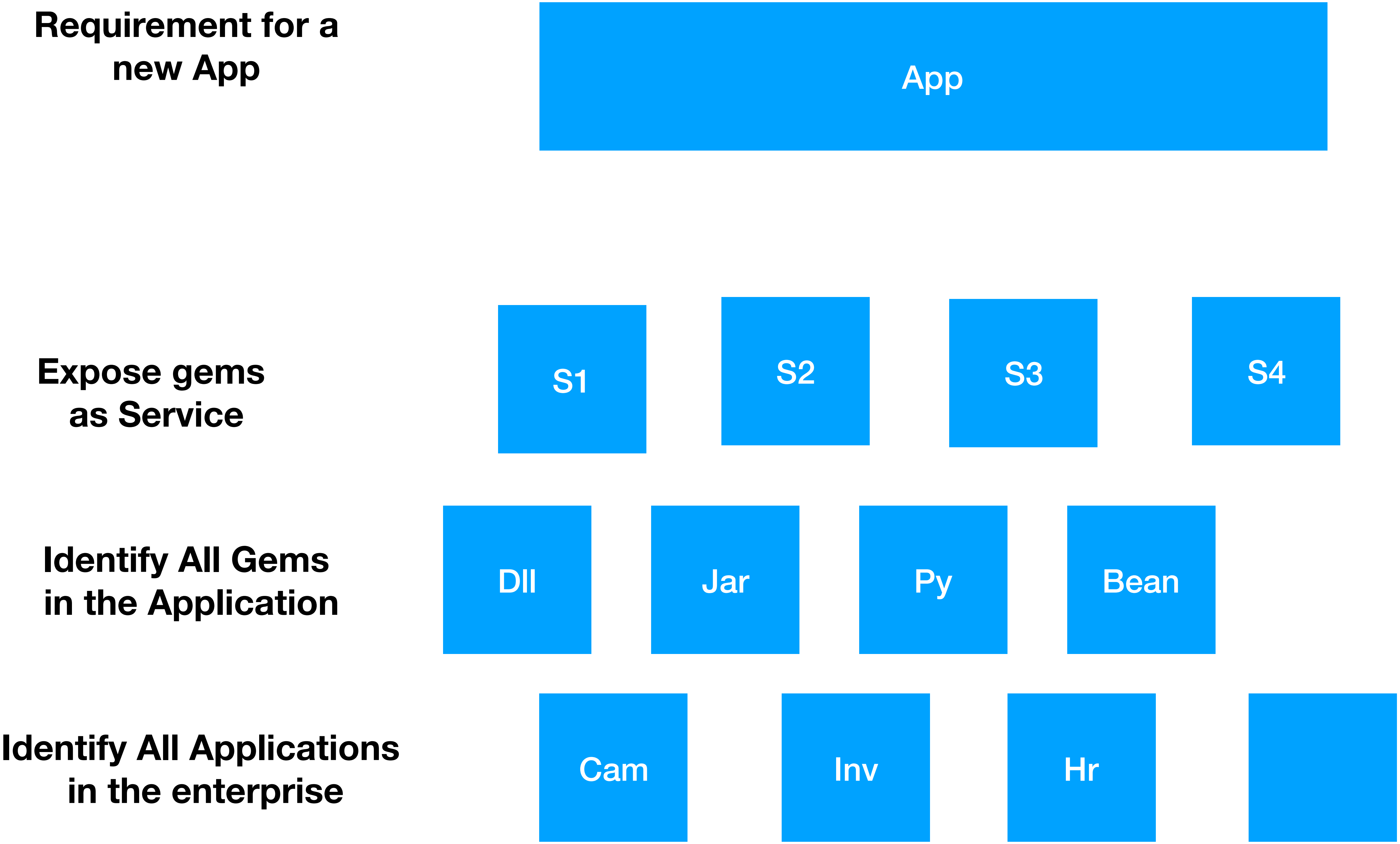

Design Patterns for Microservices
Decomposition Patterns
Integration Patterns
Database Patterns
Observability Patterns
Cross-Cutting Concern Patterns
Decompose by Business Capability
Decompose by Subdomain
Decompose by Transactions
Strangler Pattern
Bulkhead Pattern
Sidecar Pattern
API Gateway Pattern
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

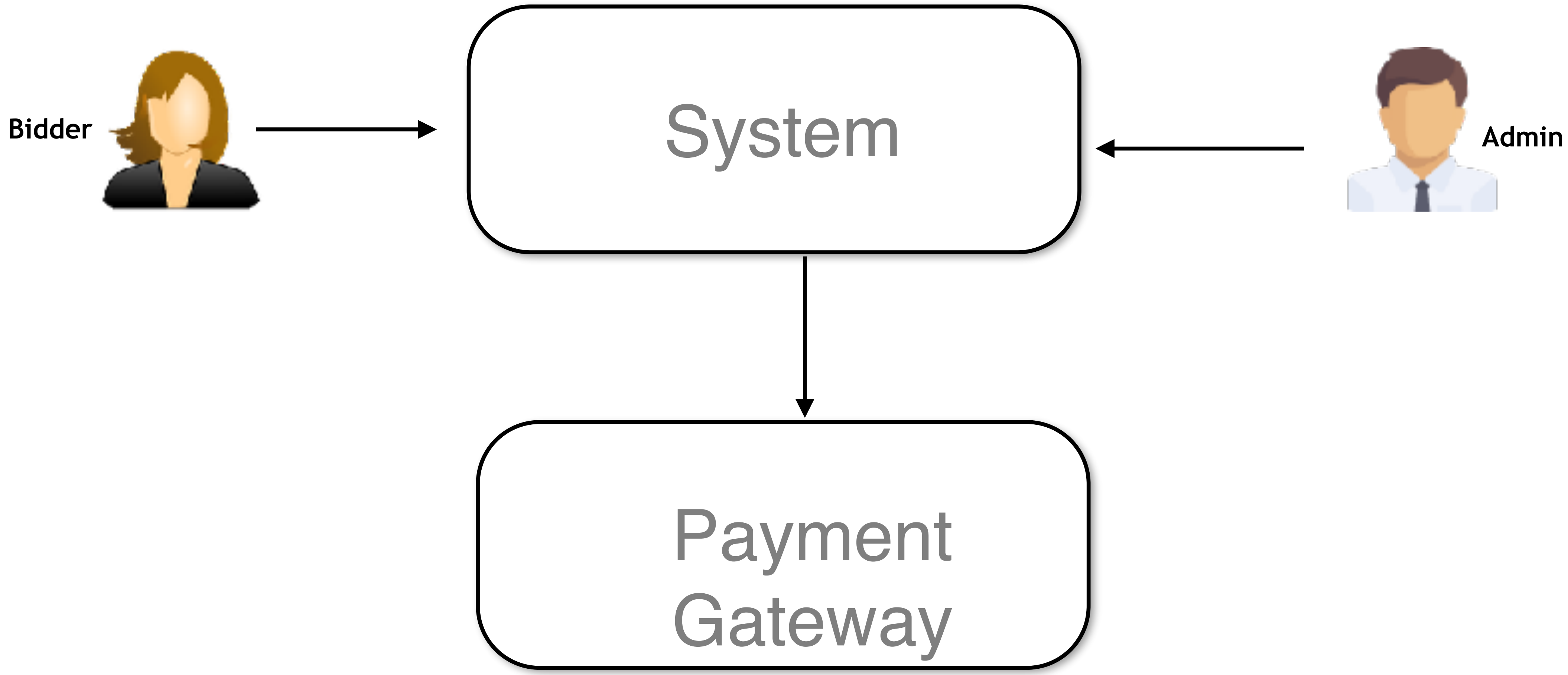
Distributed Database

Bid of the Day

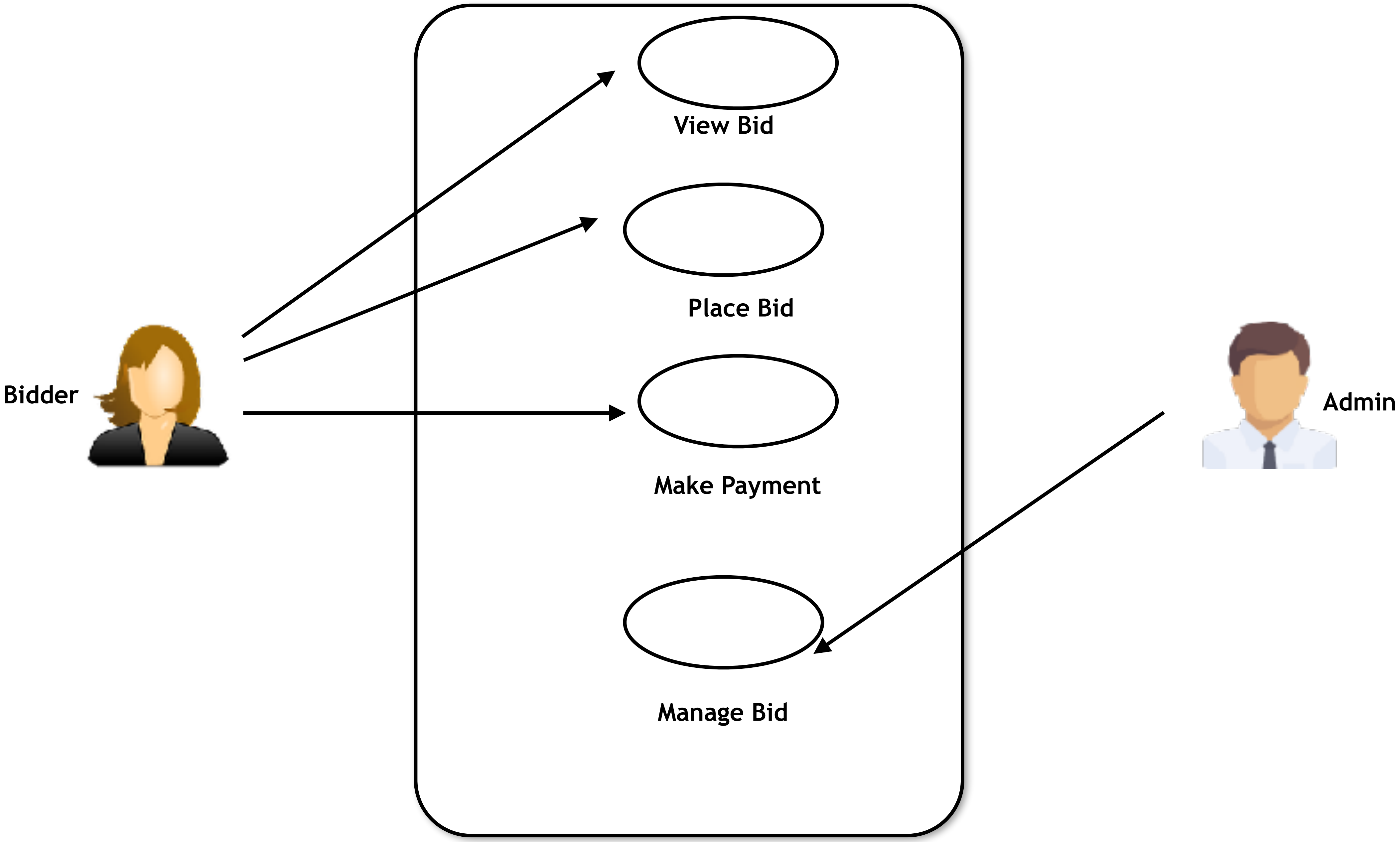
Architectural Requirements

Context View

Black box view



Functional View



Constraints & Assumptions

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

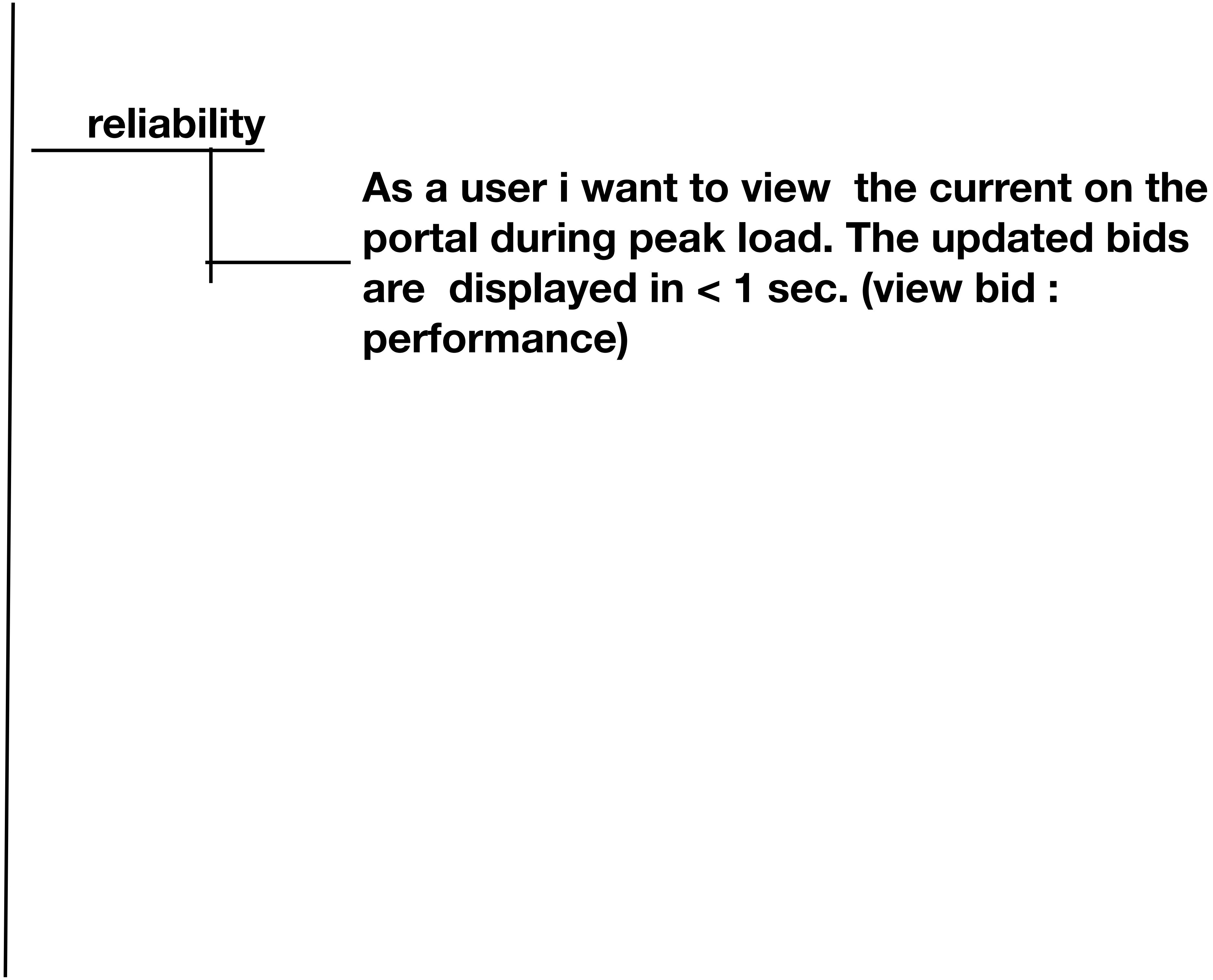
Quality

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



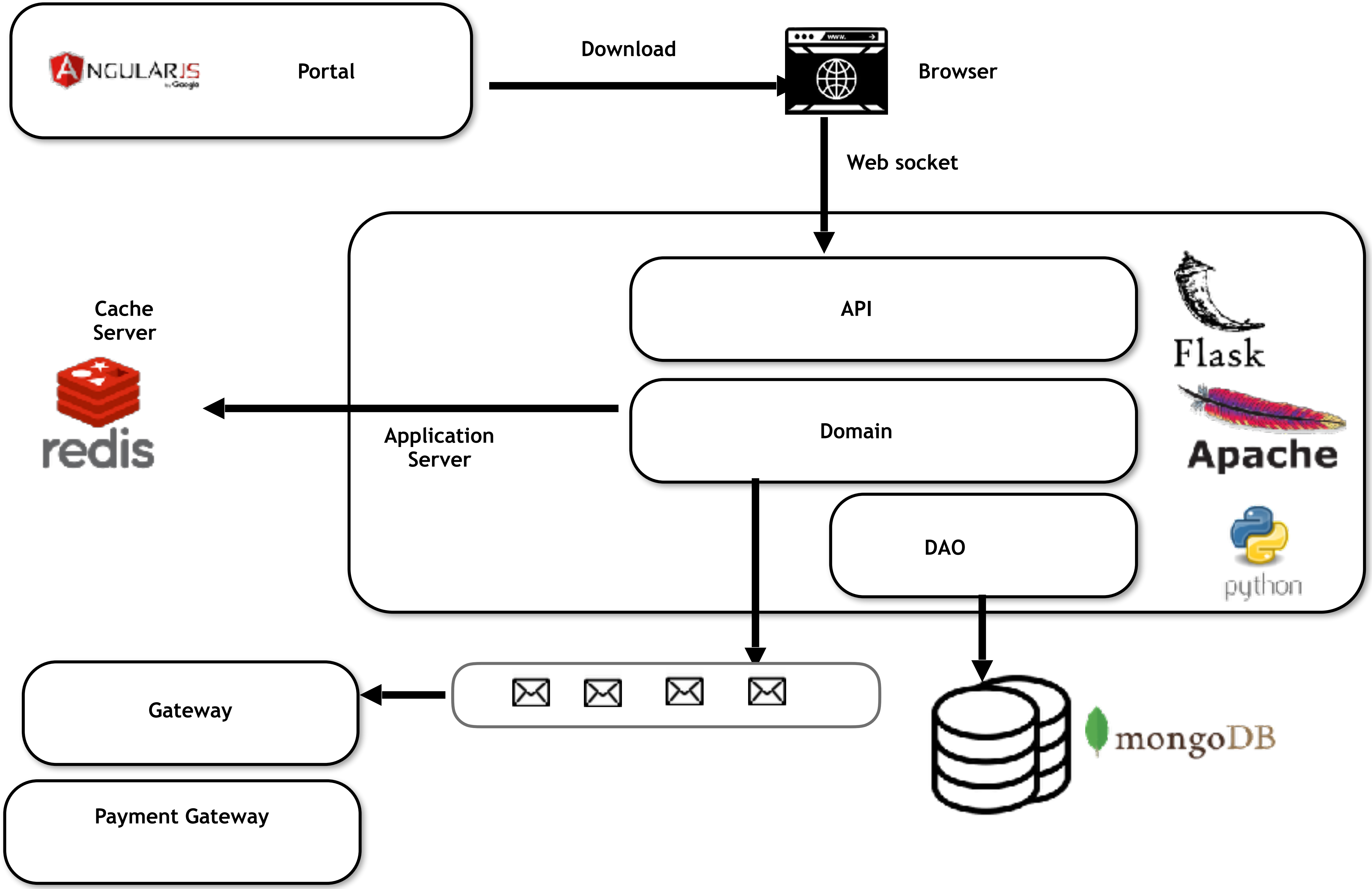
reliability

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

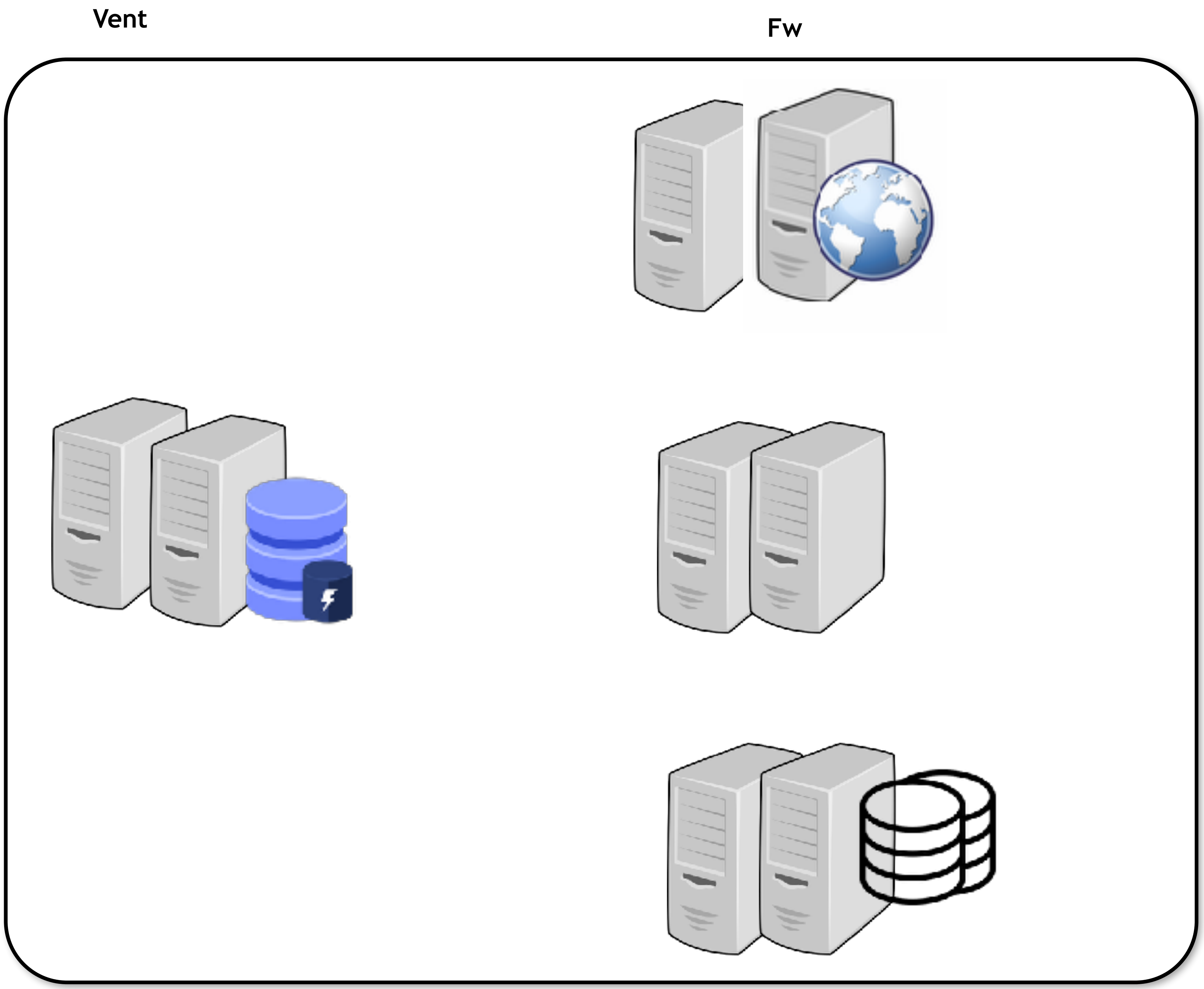
Logical view

White box view



Infrastructure View

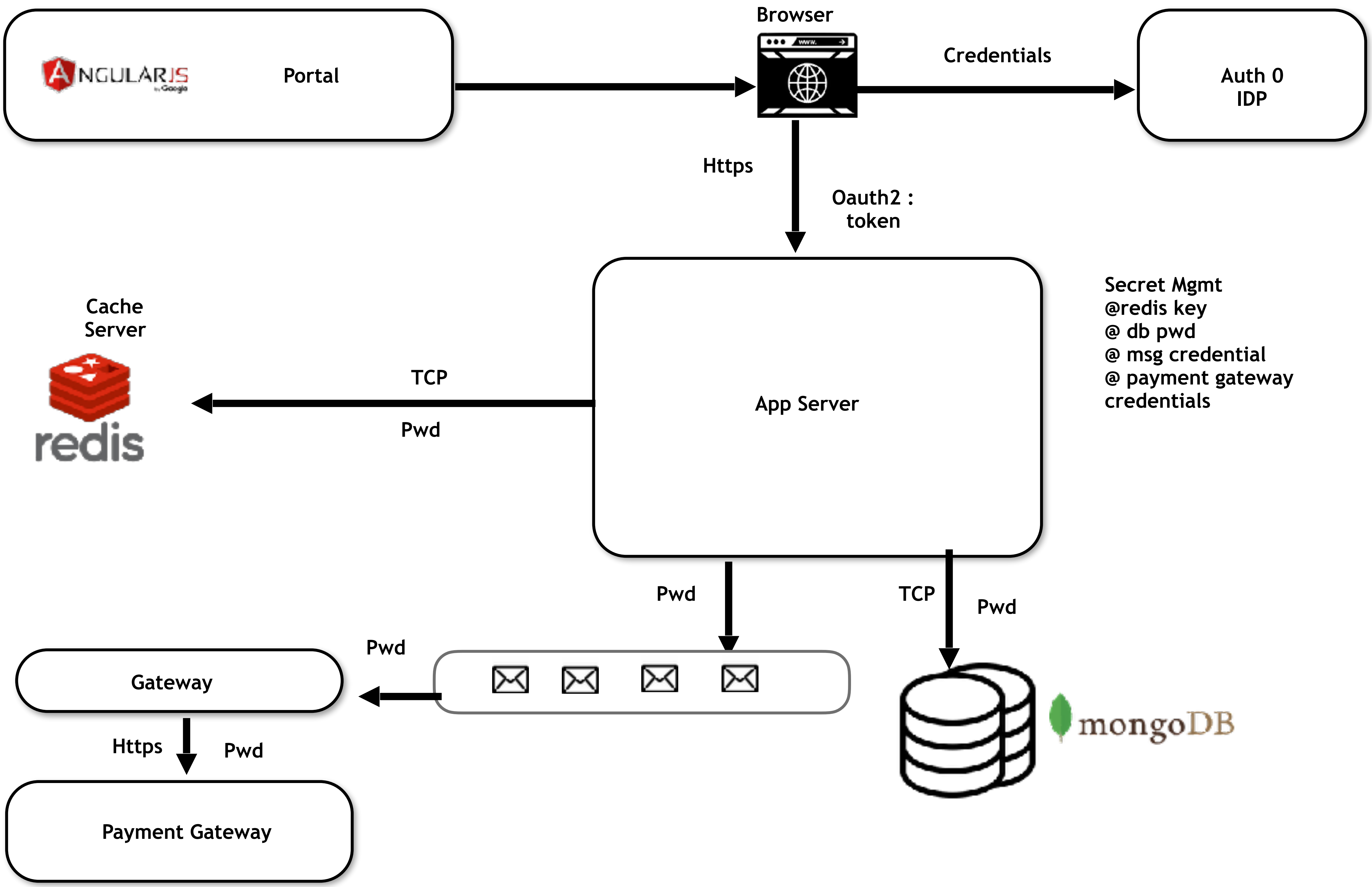
Physical view

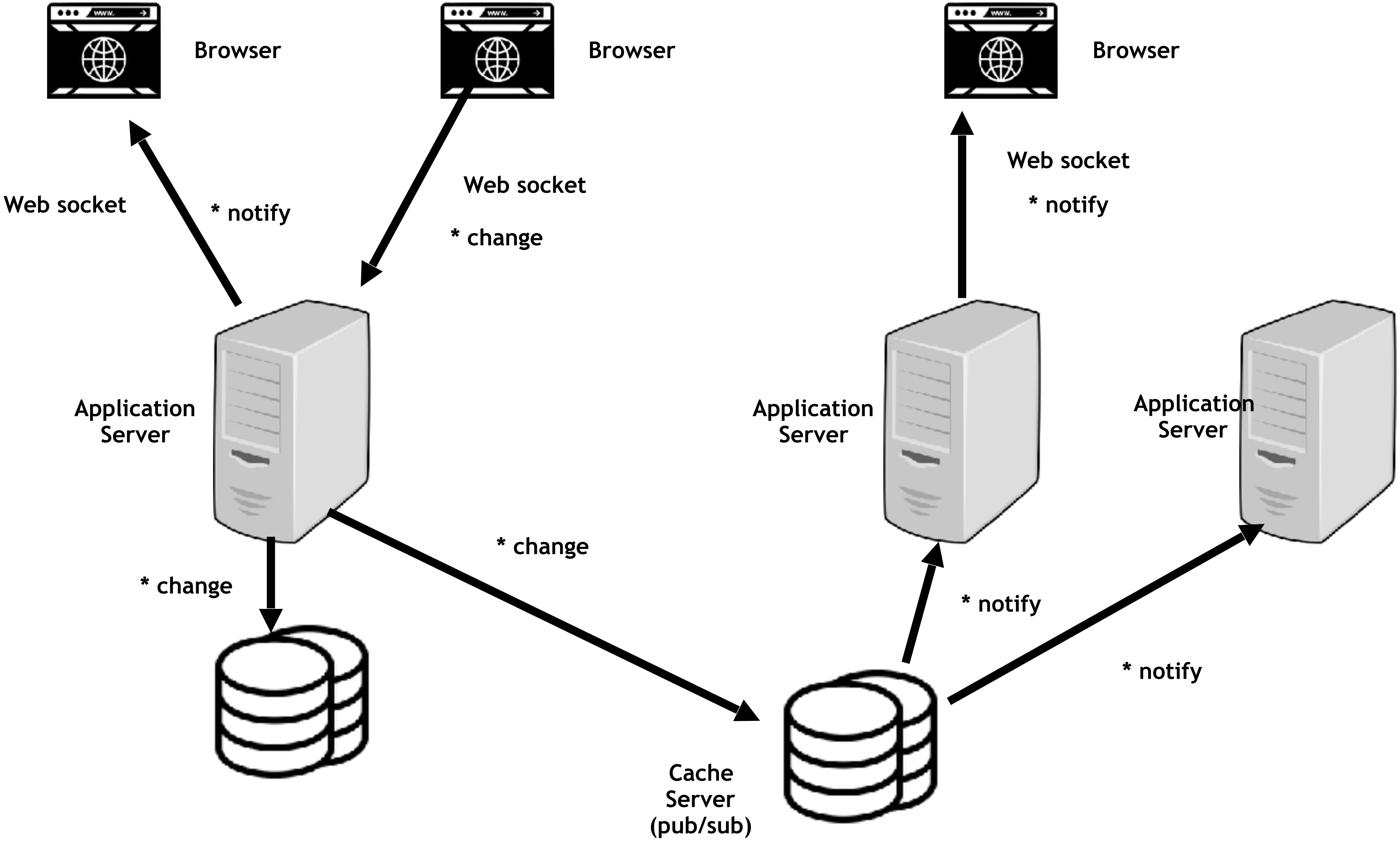


Vent

Fw

Security View




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

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