» Skan.ai - chief Architect

» Ai.robotics - chief Architect

» Genpact - solution Architect

» Welldoc - chief Architect

» Microsoft

» Mercedes

» Siemens

» Honeywell

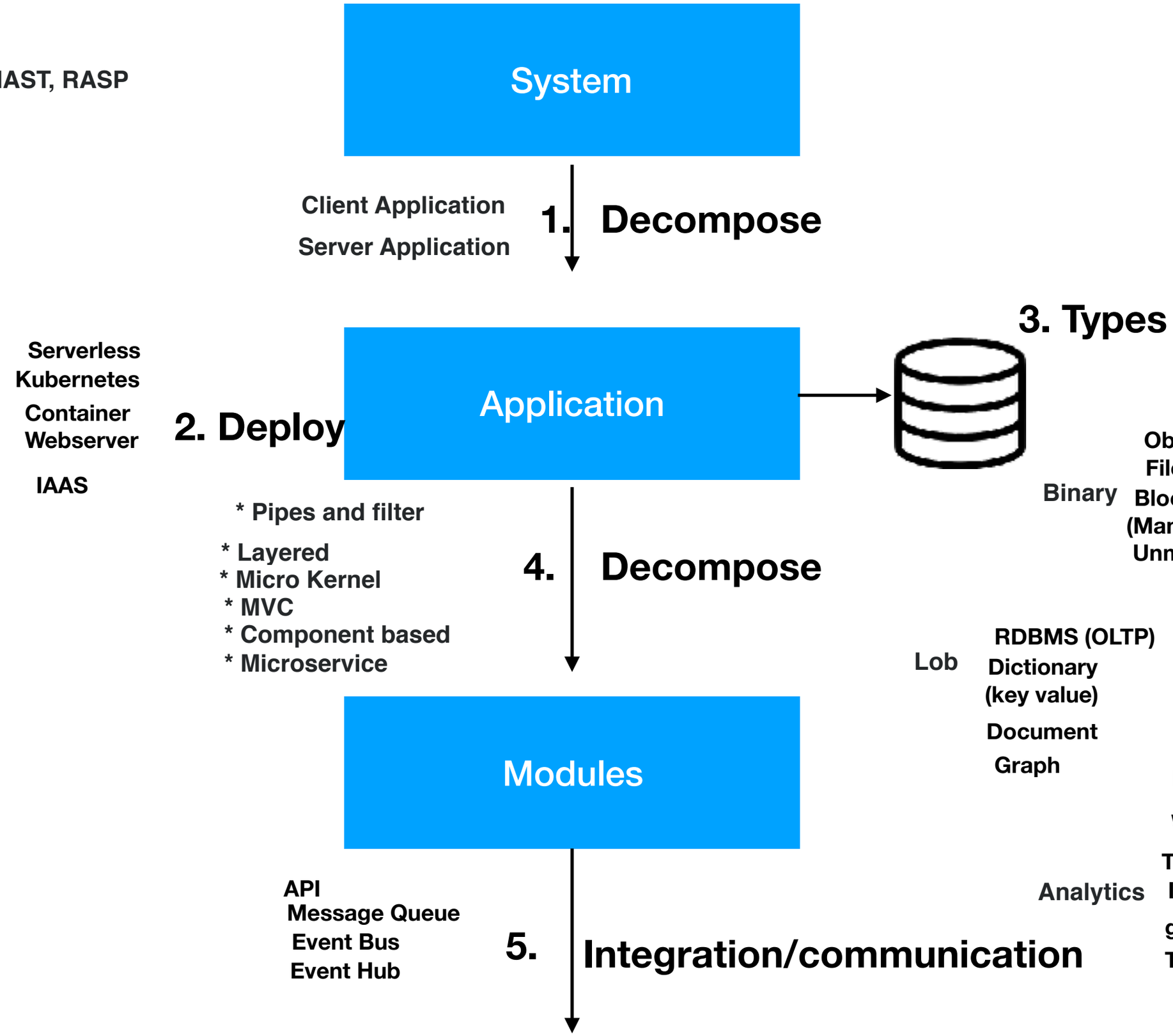
Mubarak

- **Cloud Computing**
- **Micro services**
- **ML & Data Science**
- **DevSecOps**

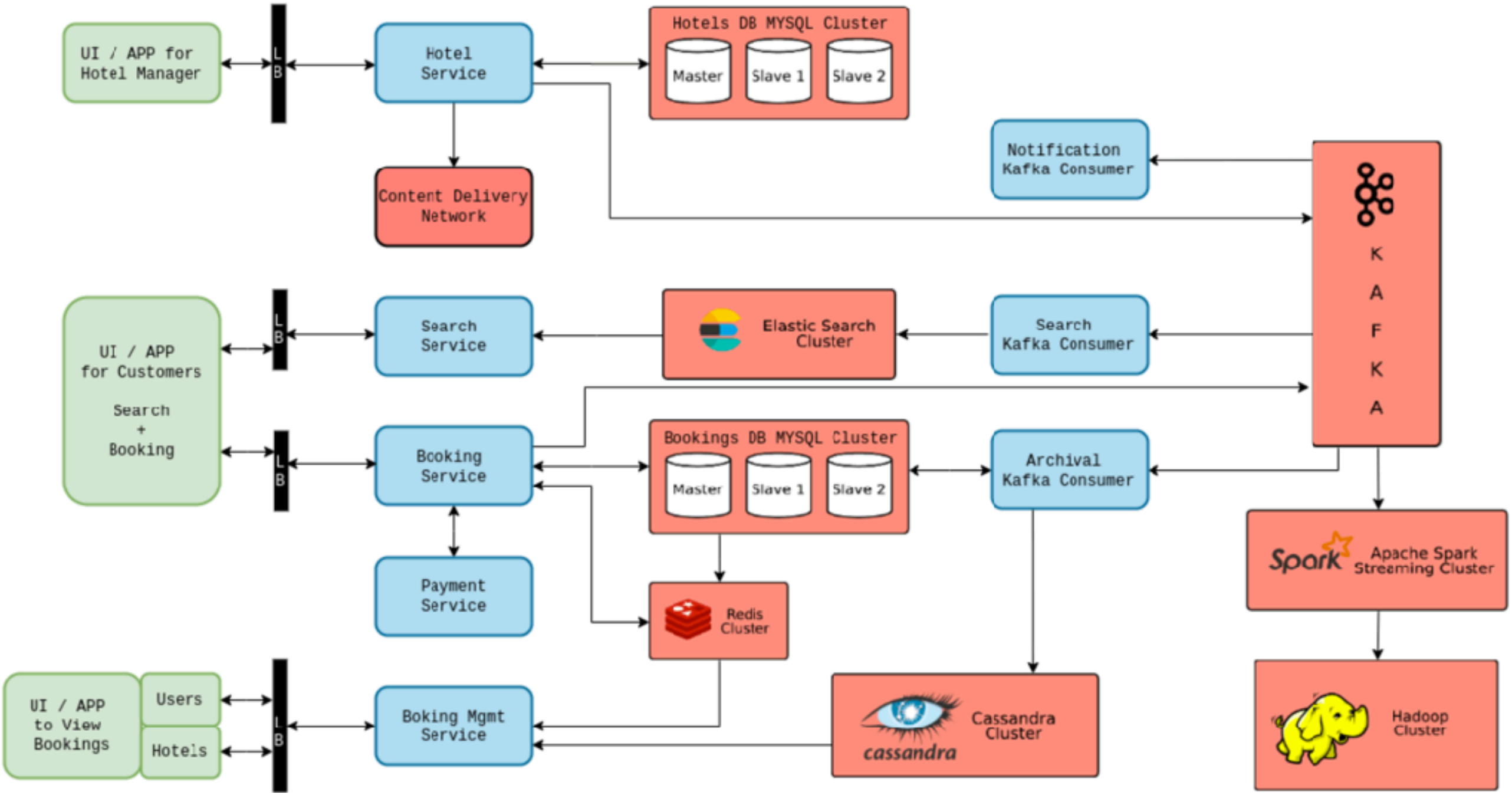
- Years of experience
- Role
- Expectations

6. DevSecOps (Dev to Prod)

Containeres
CI/CD
Testing
Monitoring
SAST, DAST, IAST, RASP

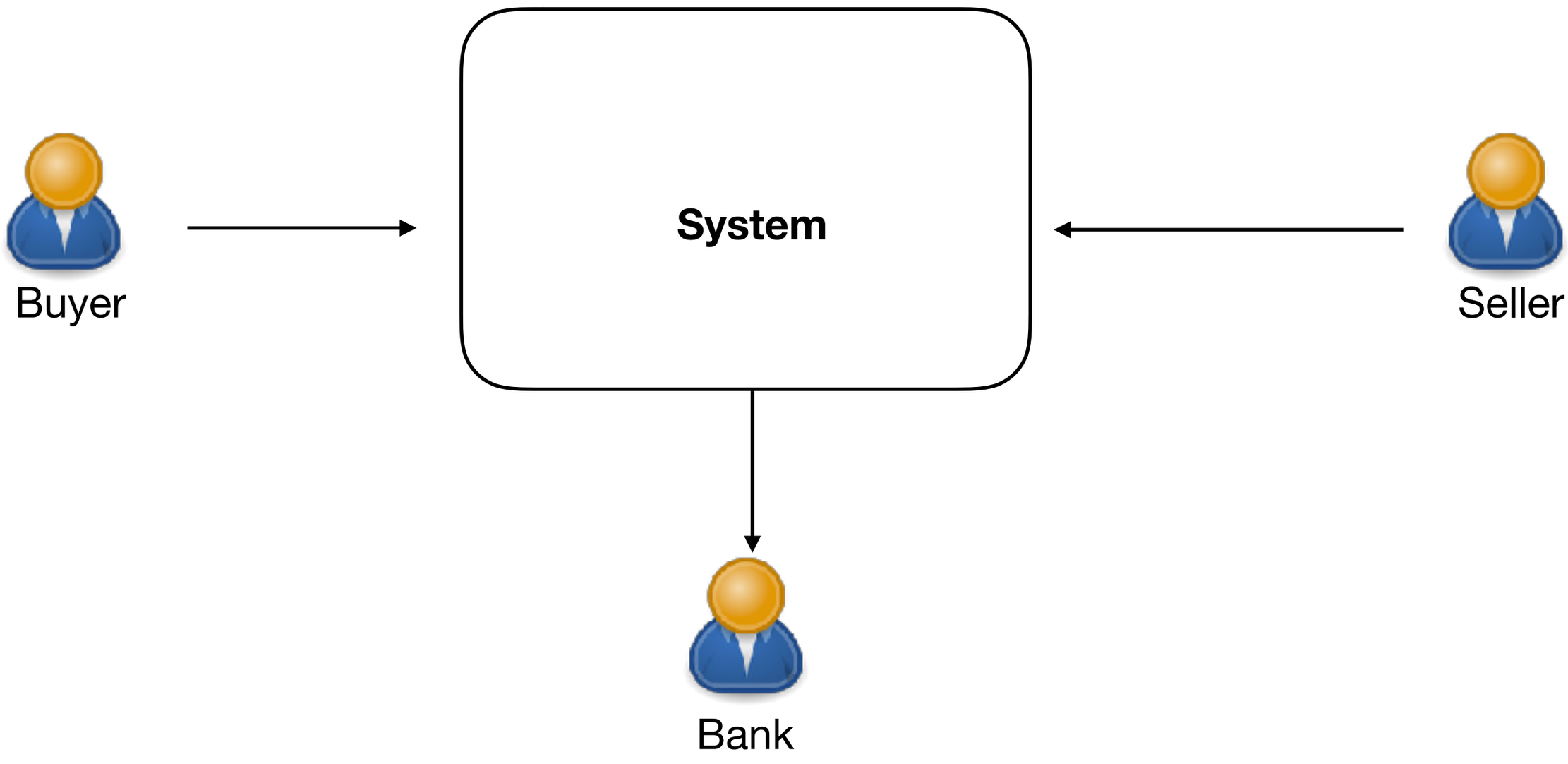


7. Analytics

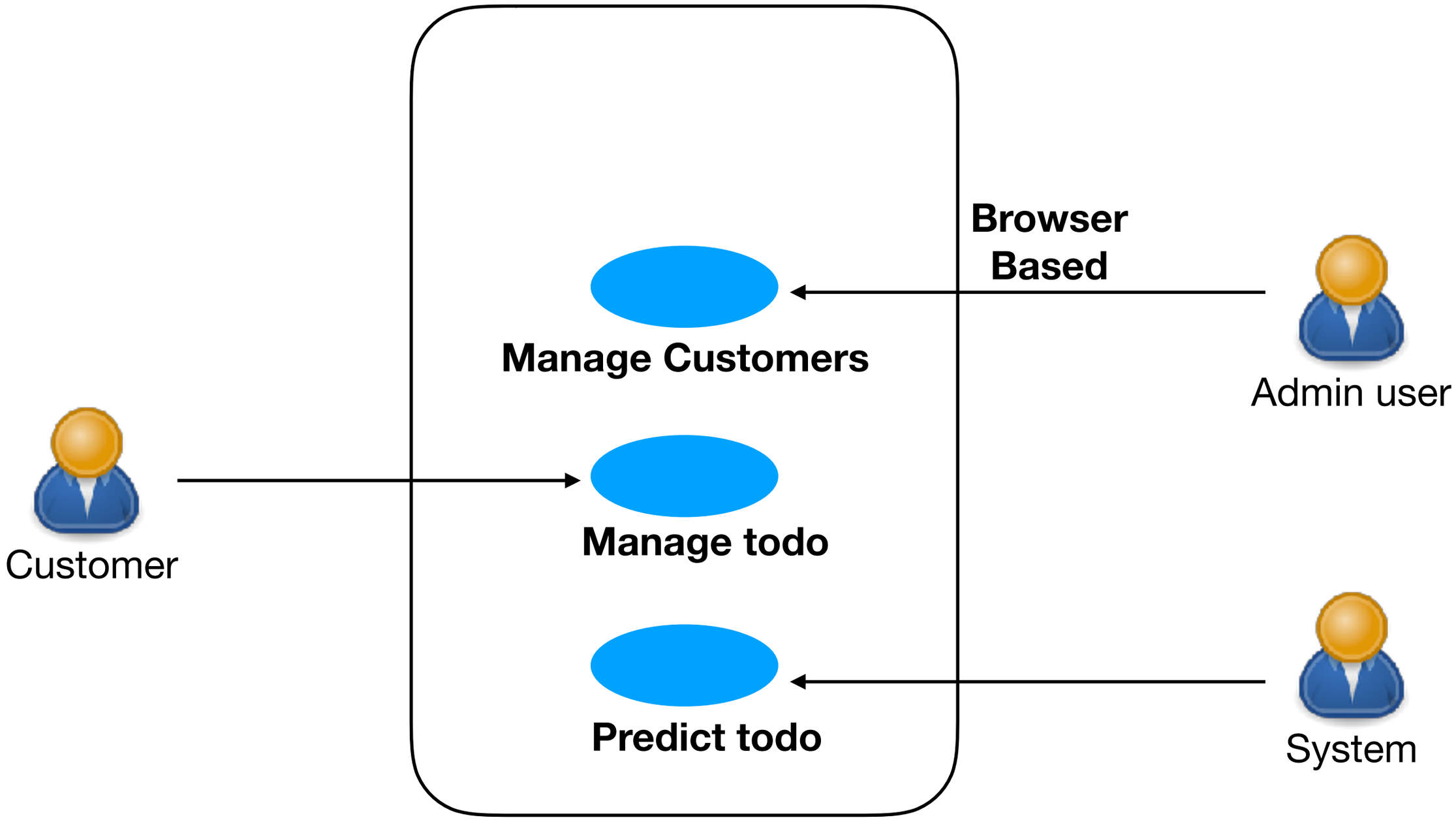

UI / APP for Hotel Manager
LB
Hotel Service
Hotels DB MYSQL Cluster
Master
Slave 1
Slave 2
Content Delivery Network
Notification Kafka Consumer
K A F K A
UI / APP for Customers
Search + Booking
LB
Search Service
Elastic Search Cluster
Search Kafka Consumer
LB
Bcoking Service
Bookings DB MYSQL Cluster
Master
Slave 1
Slave 2
Archival Kafka Consumer
Spark
Apache Spark Streaming Cluster
Payment Service
Redis Cluster
UI / APP to View Bookings
Users
Hotels
LB
Boking Mgmt Service
cassandra
Cassandra Cluster
Hadoop Cluster

Context View

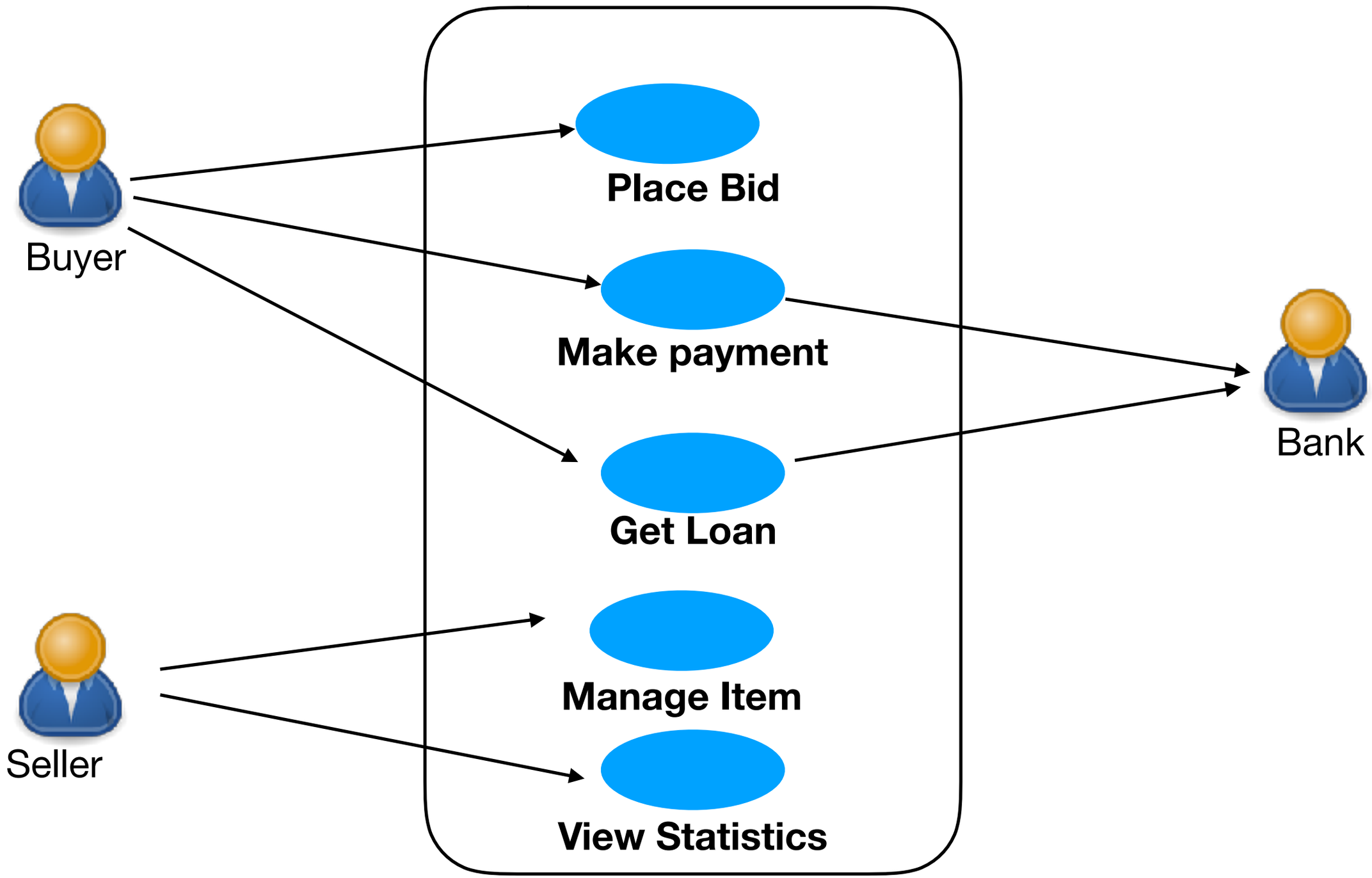
Context view



Functional view



Functional view



Compute

Storage

Message

System Decomposition

Compute

Client Application

Server Application

Client-Server Applications

UI Application
Native Application (1)
* WPF
* Swing
* Android
# Browser - Native Application (2)
* Flash
* Activex
* Applet
# Browser - SPA(4)
* Angular
* React

Background Application
* Windows Service
* Daemon
* Exe

Foreground Application

* Server Pages (JSP, ASP)
* API Application

Background Application
Event Based Application
Scheduled Application

Two-tier Client Server

Multi(N)-tier client-server

Browser

Native

Request

Native

Static Webserver

Flash File

Flash
Silverlight
Applet
Activex

Binary

Browser

Html / js

Request

Html / js

Dynamic Webserver

Engine

File

Jsp,
asp,
php,
Servlet,
JSF,
Cold Fusion,
Python,
Nodes

Browser

Html / js

Angular
React
...

Request

Html / js

Static Webserver

File

html/js

Desktop/ mobile

Customer

Browser

Crud todo

Server

* API Application

API App

* Prediction Application

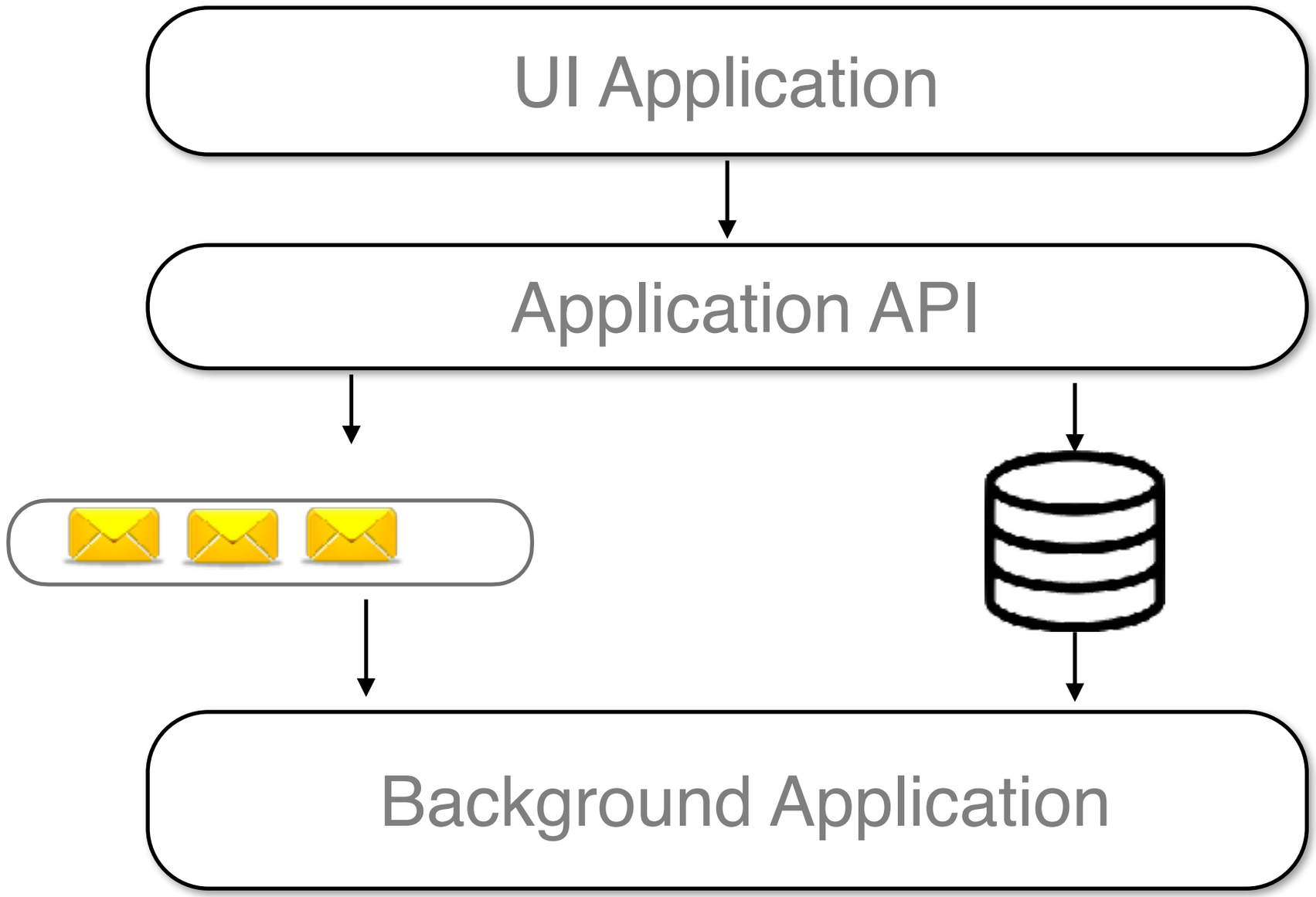
Analytical App

www.todo.com

Business user

Browser - SPA

UI App


UI Application
Application API
Background Application

Buyer

Seller

Web Portal

API Service

Statistics Report

Data Store

Event Stream

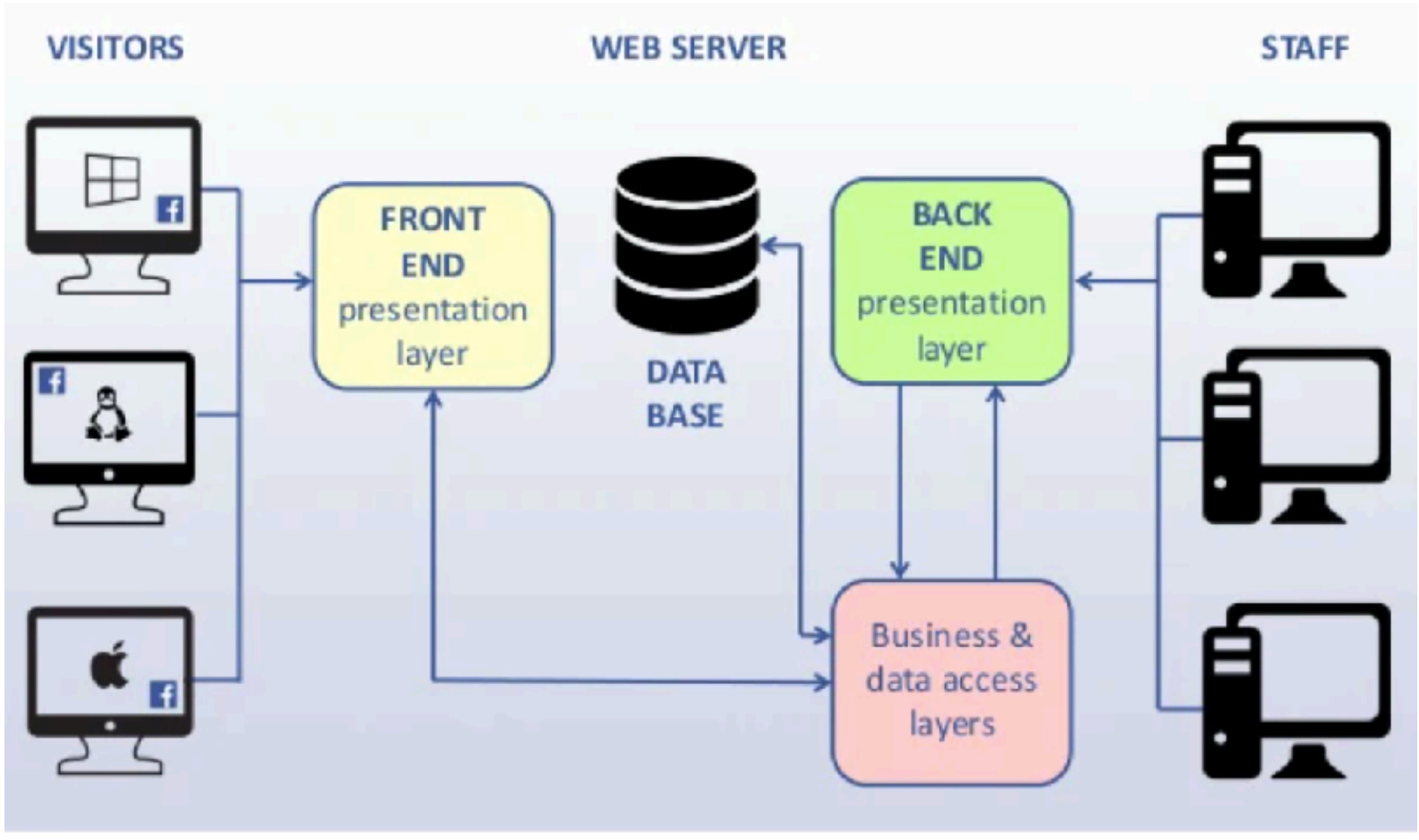
Data Lake

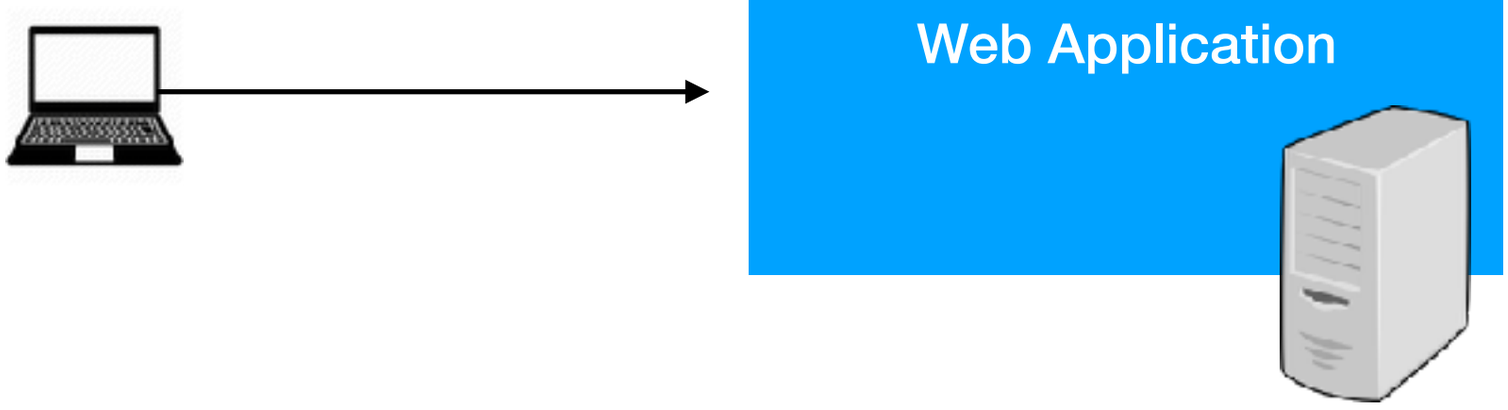
Loan Service

Analytic Service

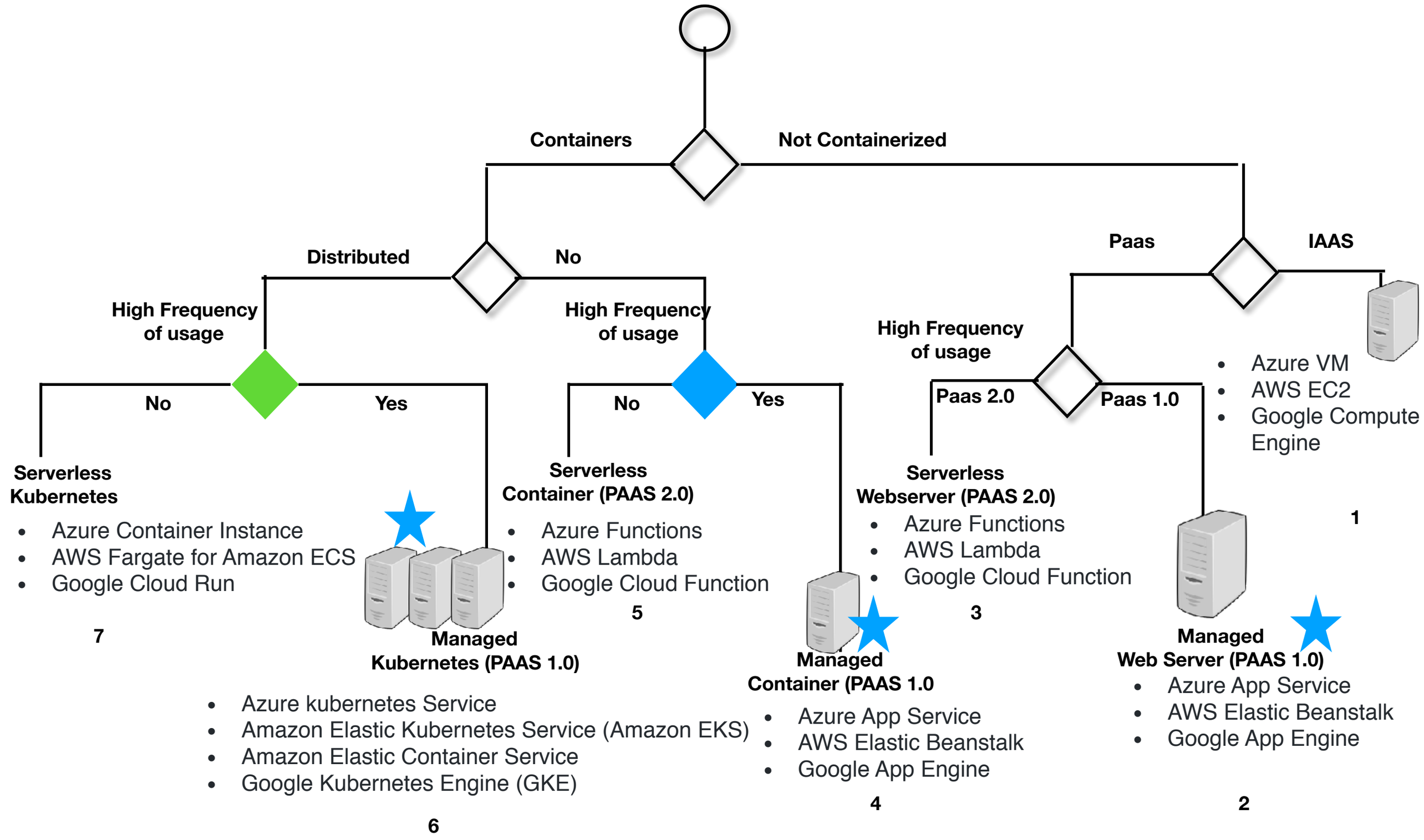
Third Party Bank Services

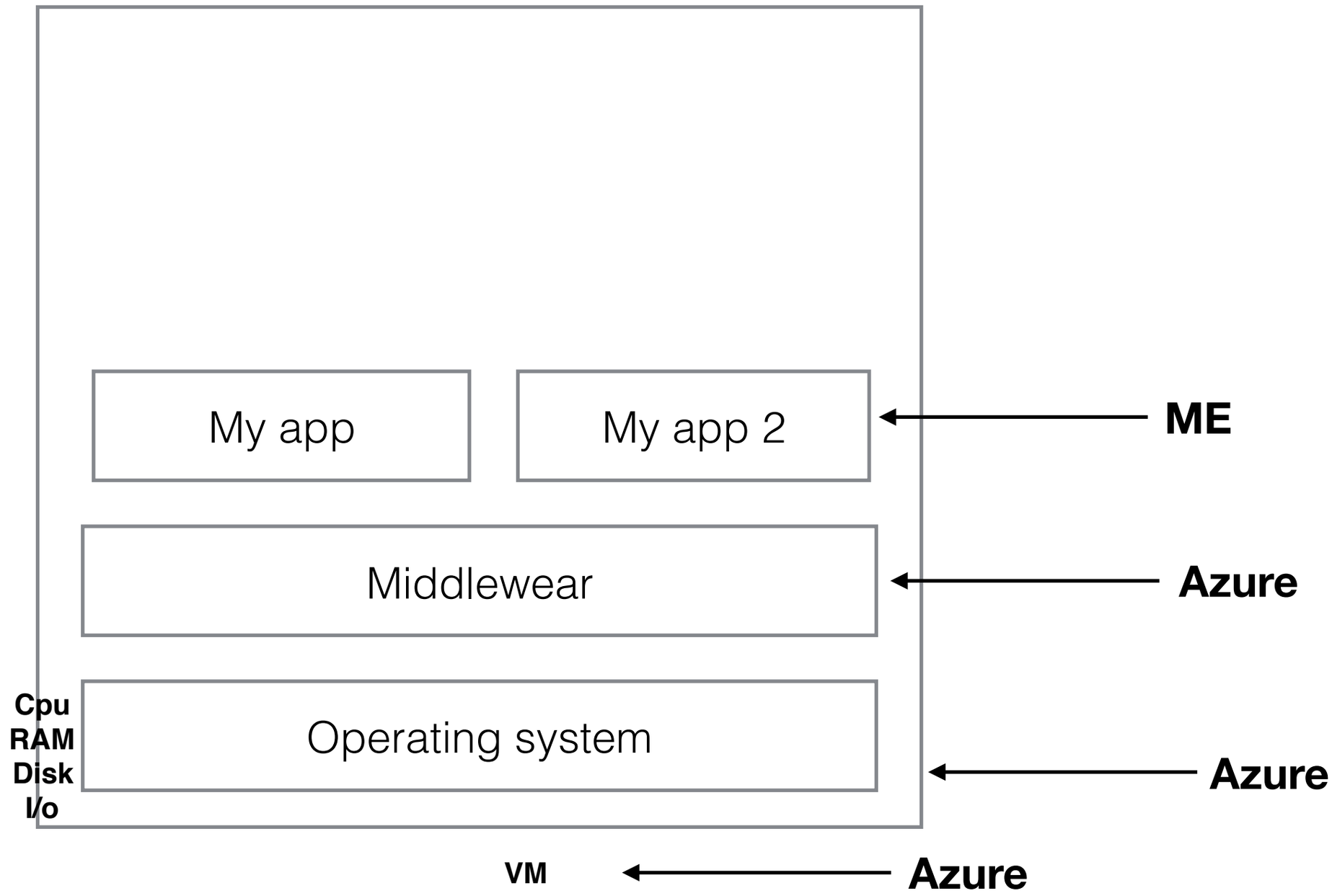
Facebook Reference Architecture

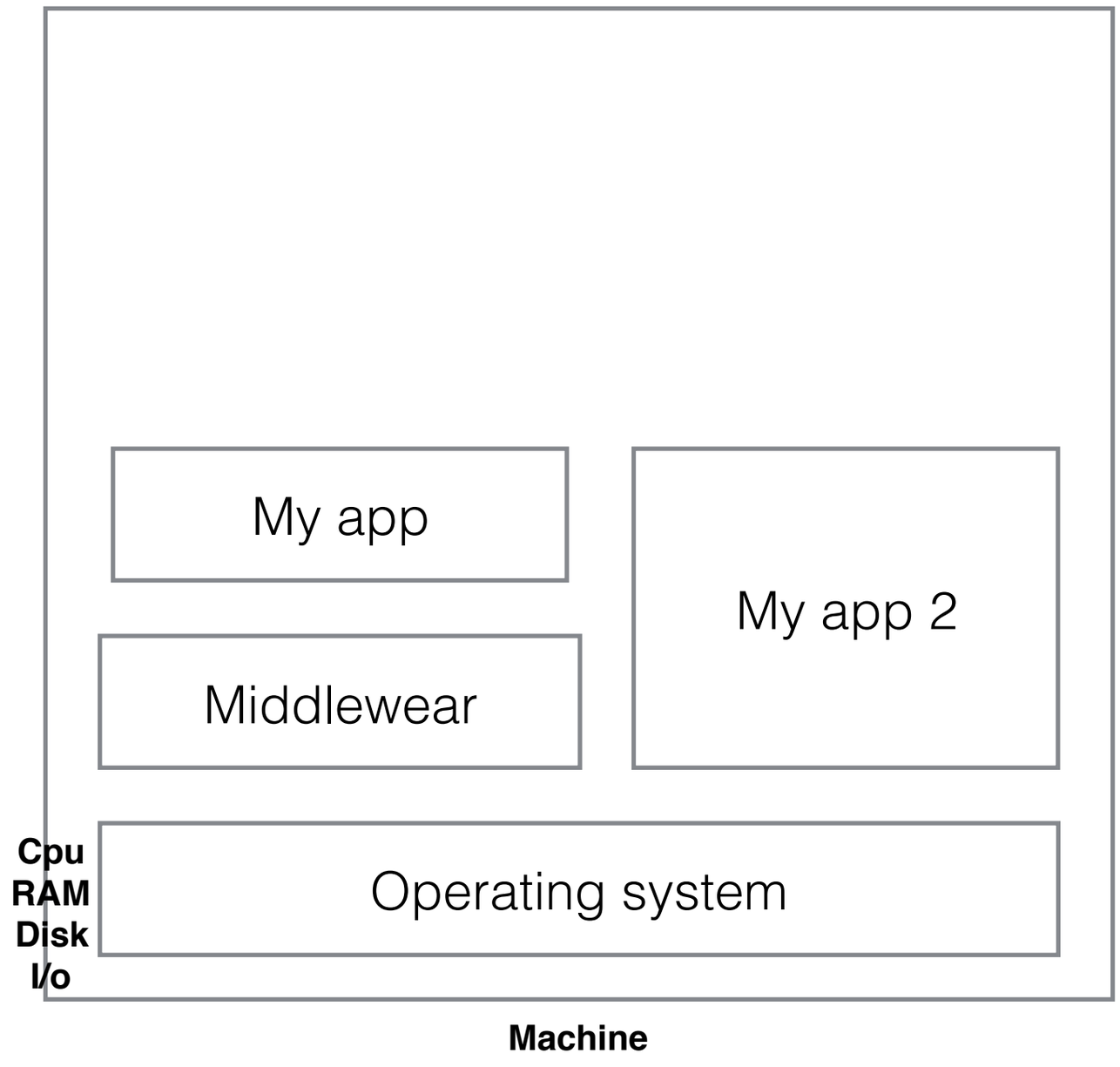


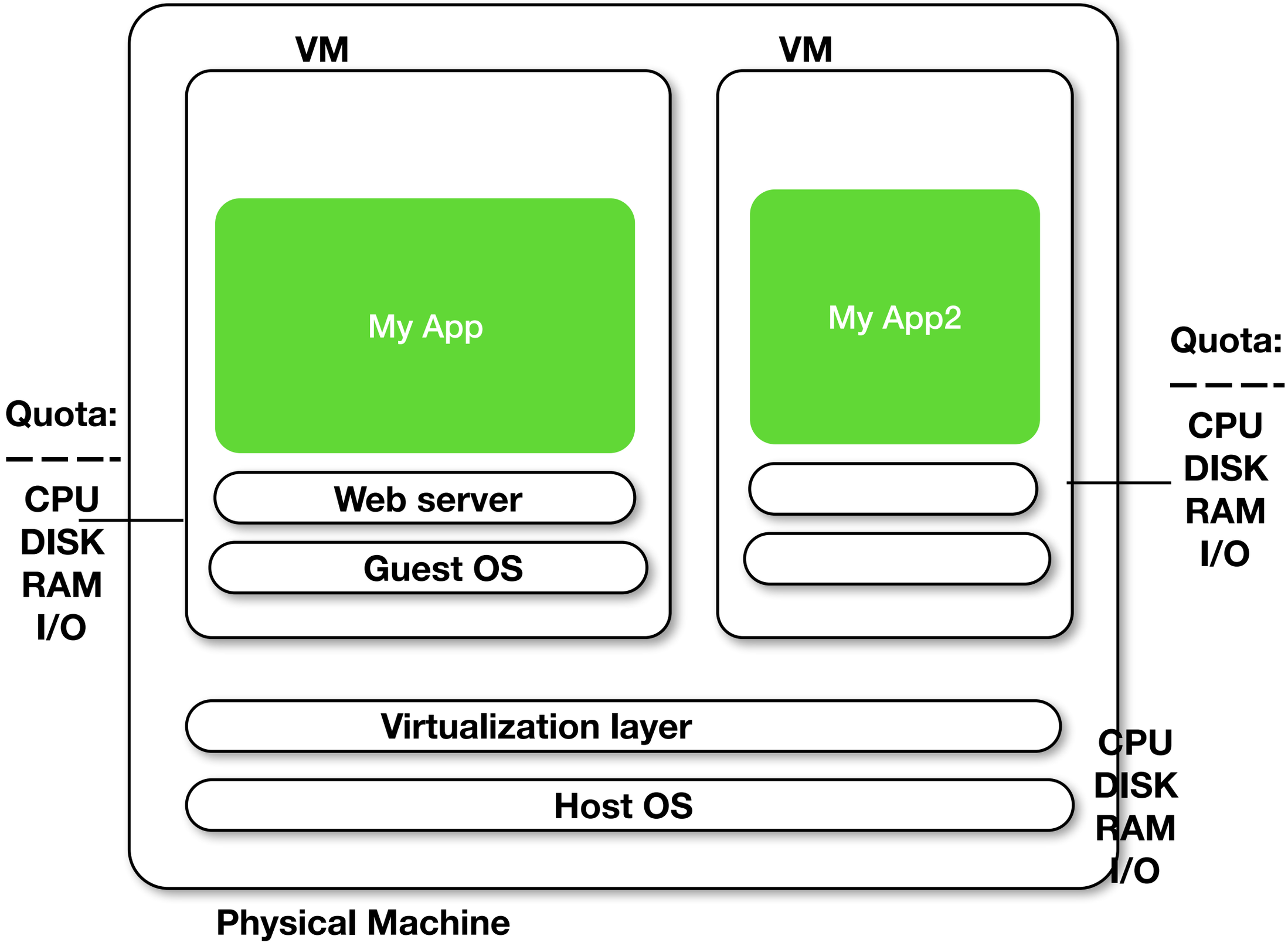

Web Application

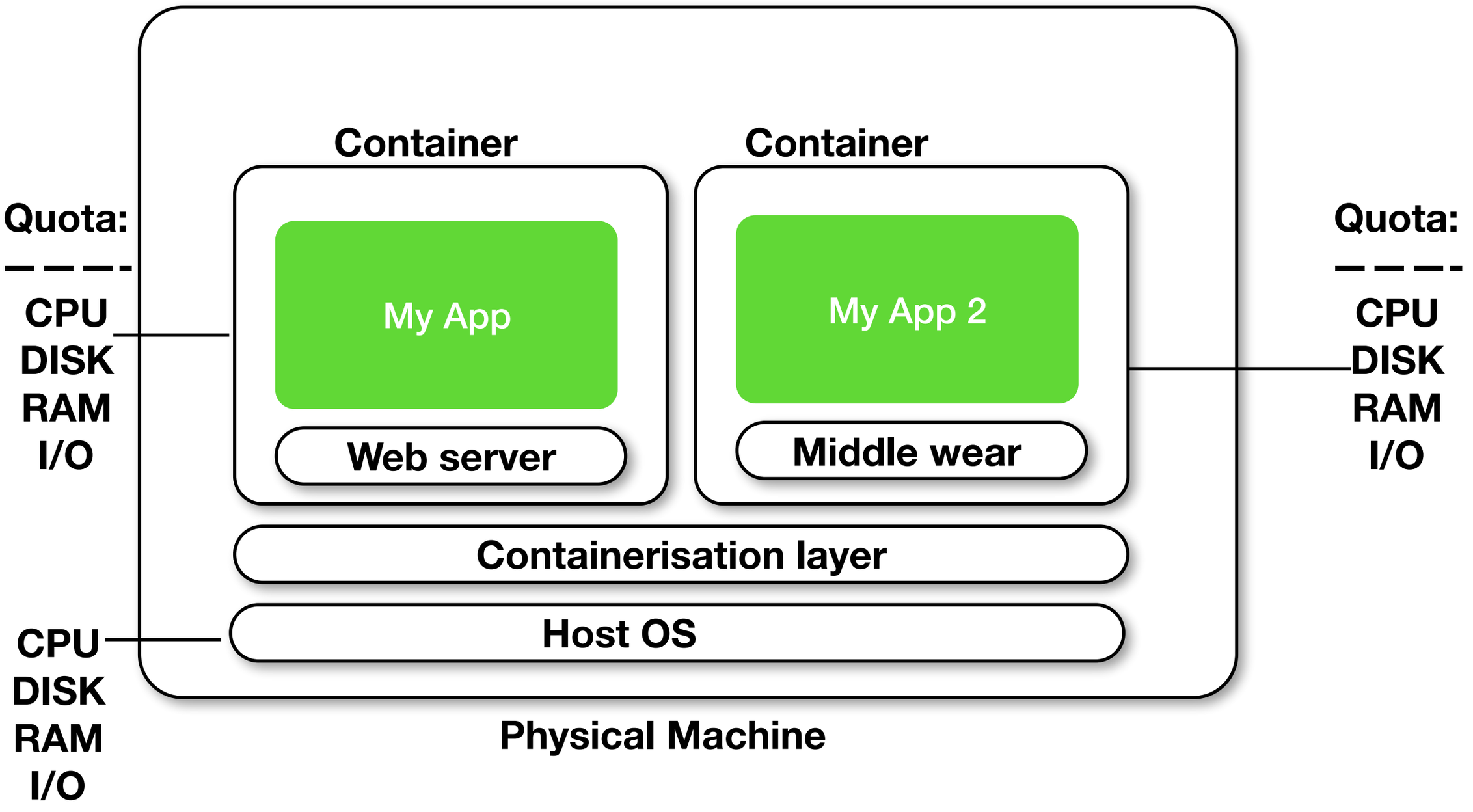
Deployment View

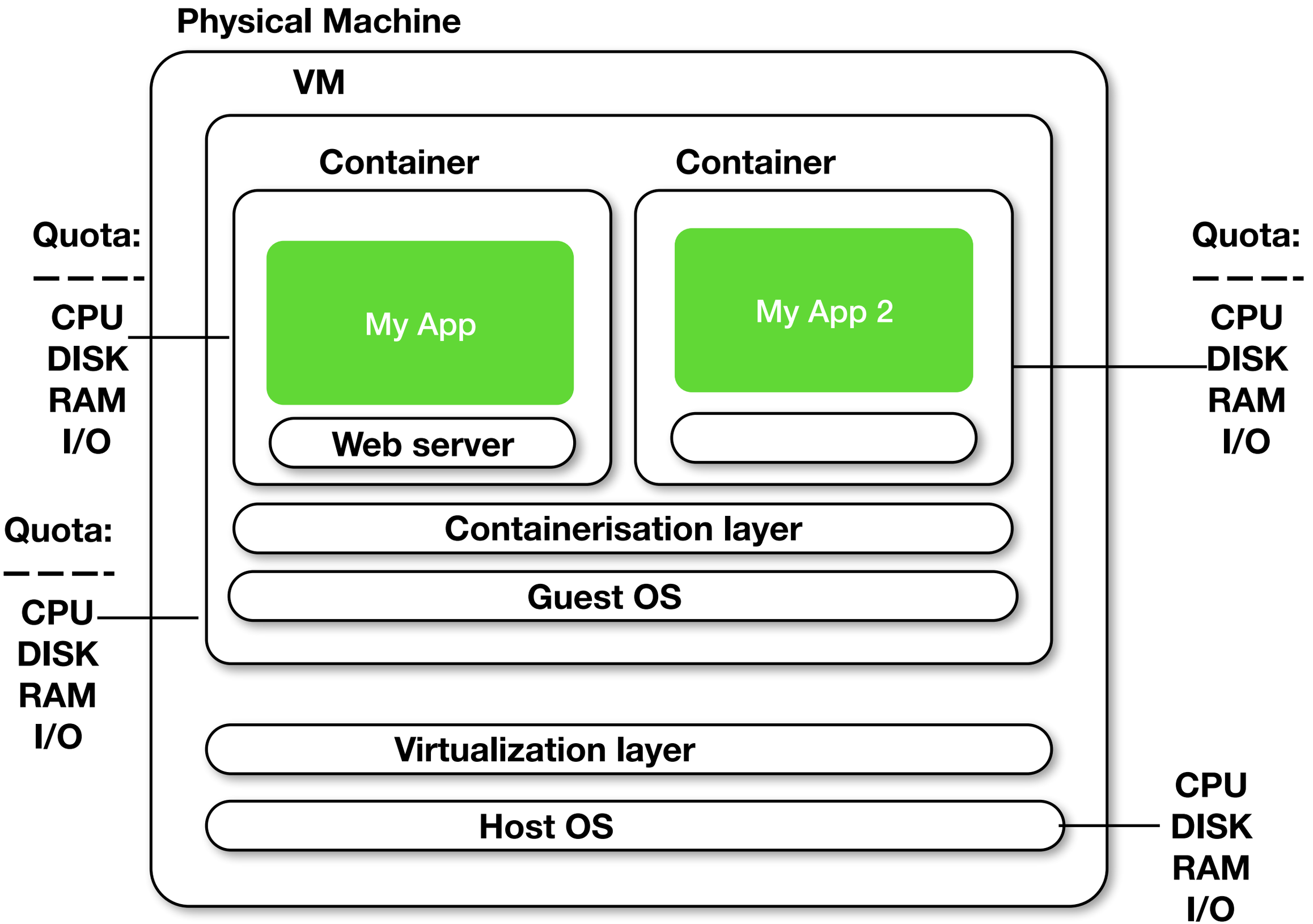



My app
My app 2
ME
Middlewear
Azure
Cpu
RAM
Disk
I/o
Operating system
Azure
VM
Azure

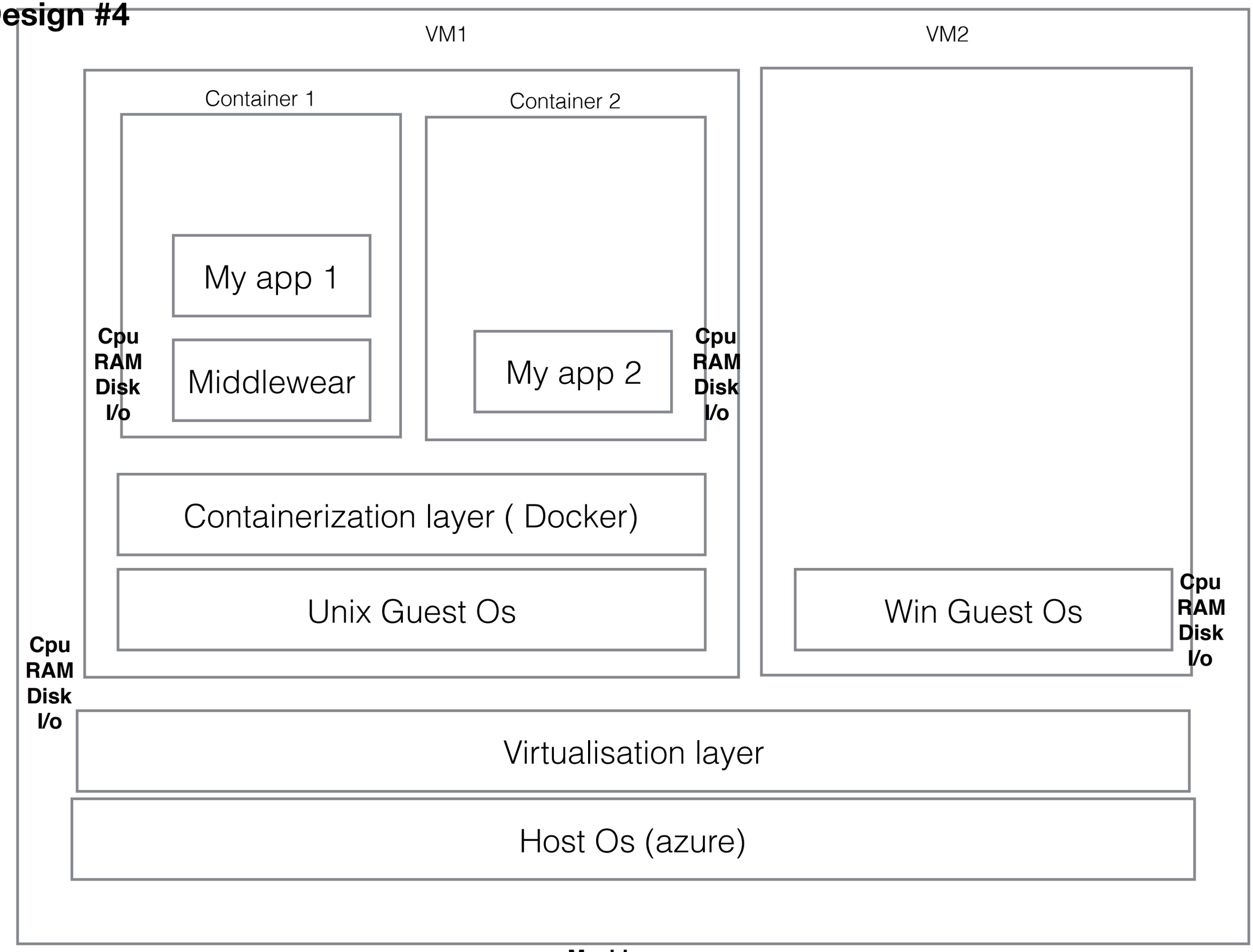

My app
My app 2
Middlewear
Operating system
Cpu
RAM
Disk
I/o
Machine


VM
VM
My App
My App2
Web server
Guest OS
Virtualization layer
Host OS
Physical Machine
Quota:
CPU
DISK
RAM
I/O
Quota:
CPU
DISK
RAM
I/O
CPU
DISK
RAM
I/O


Container
Container
Quota:
My App
My App 2
CPU
DISK
RAM
I/O
Web server
Middle wear
Containerisation layer
Host OS
CPU
DISK
RAM
I/O
Quota:
CPU
DISK
RAM
I/O
Physical Machine


Physical Machine
VM
Container
Container
My App
My App 2
Web server
Containerisation layer
Guest OS
Virtualization layer
Host OS
Quota:
CPU
DISK
RAM
I/O
Quota:
CPU
DISK
RAM
I/O
Quota:
CPU
DISK
RAM
I/O
CPU
DISK
RAM
I/O

Design #4


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

VM1

VM2

Container 1

Container 2

My app 1

Middlewear

My app 2

Container 2

My app 3

Containerization layer (Docker)

Containerization layer

Unix Guest Os

Unix Guest Os

Virtualisation layer

Host Os (azure)

Cpu RAM Disk I/o

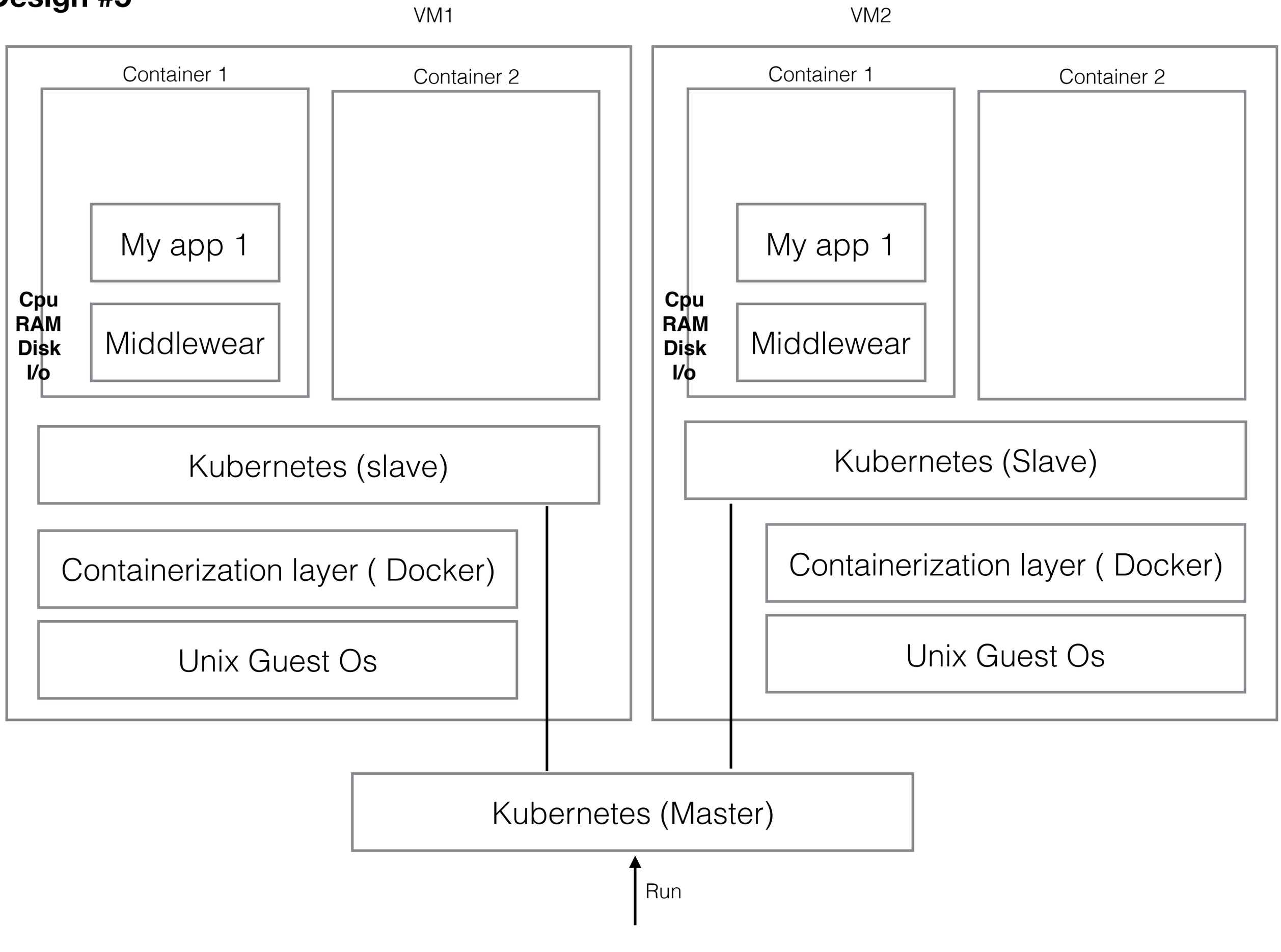
Cpu RAM Disk I/o

Cpu RAM Disk I/o

Cpu RAM Disk I/o

Machine

Design #5


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

My app

OS

Machine

My app

Middlewear

OS

Machine

My app

Middlewear

Guest OS

VM

Machine

My app

Middlewear

Container

Guest OS

VM

Machine

Kubernetes

Cluster

My app

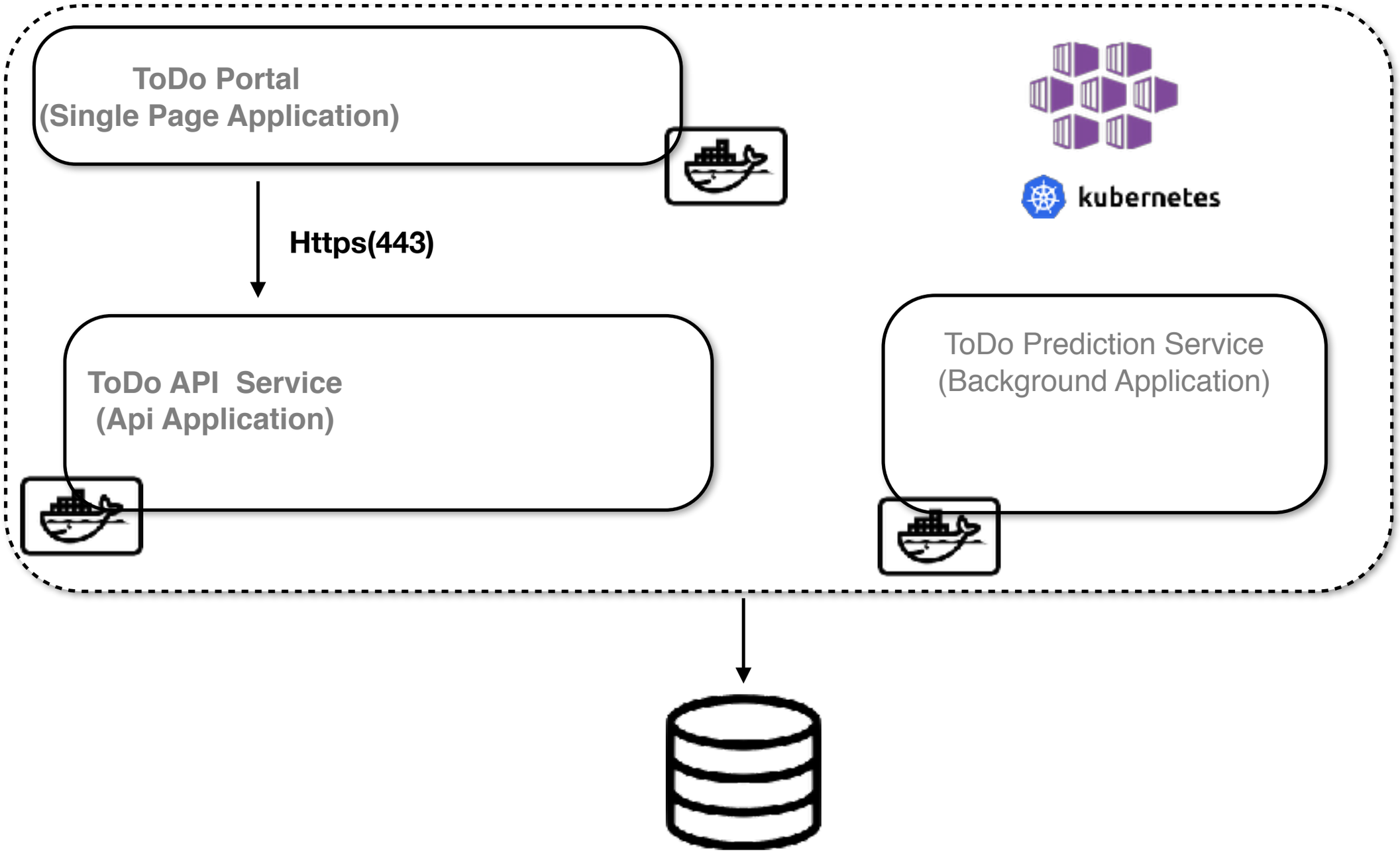
Middlewear

Container

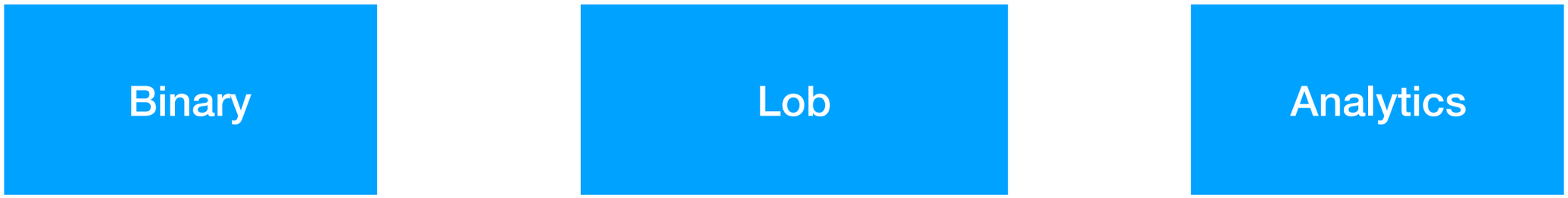
Guest OS

VM

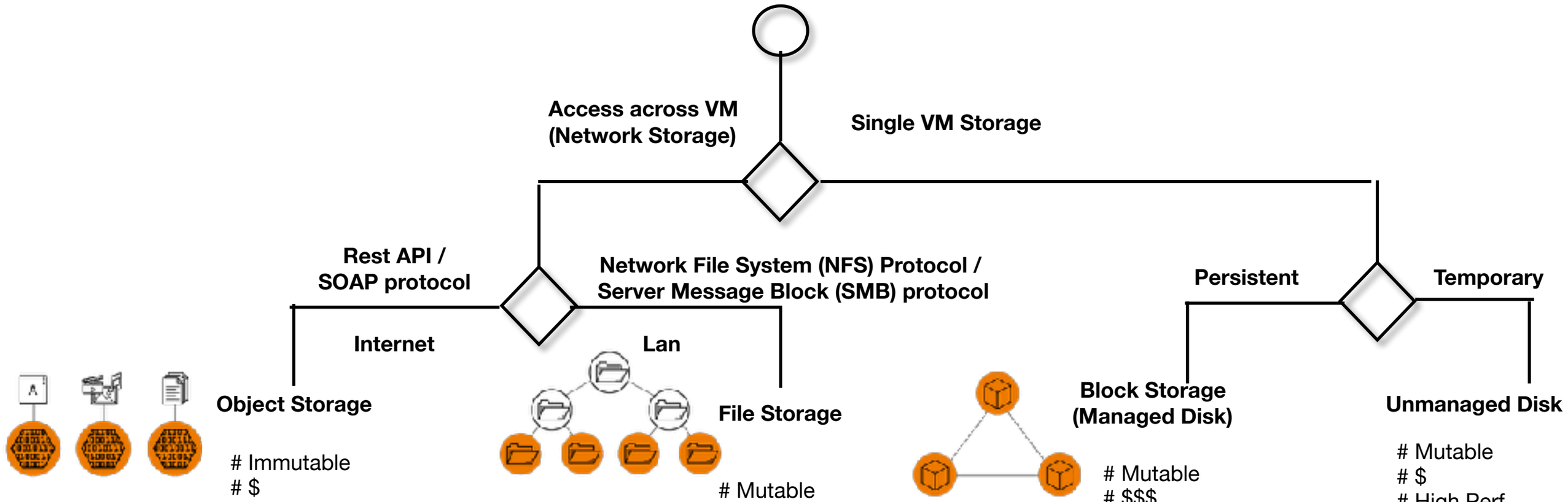
Machine


ToDo Portal
(Single Page Application)
kubernetes
Https(443)
ToDo API Service
(Api Application)
ToDo Prediction Service
(Background Application)

System Storage



Binary Storage



Object Storage

Immutable
$
Low Perf
High Volume
0.01$/GB

- Azure Blobs
- AWS S3
- google Cloud Storage

4

File Storage

Mutable
$$$$
Medium Perf
Medium Voulme

- Azure Files
- EFS
- Google Filestore
- NFS

3

Block Storage
(Managed Disk)

Mutable
$$$
High Perf
Low Volume
1$/GB

- Azure Disks
- AWS EBS
- Google Persistent Disk

2

Unmanaged Disk

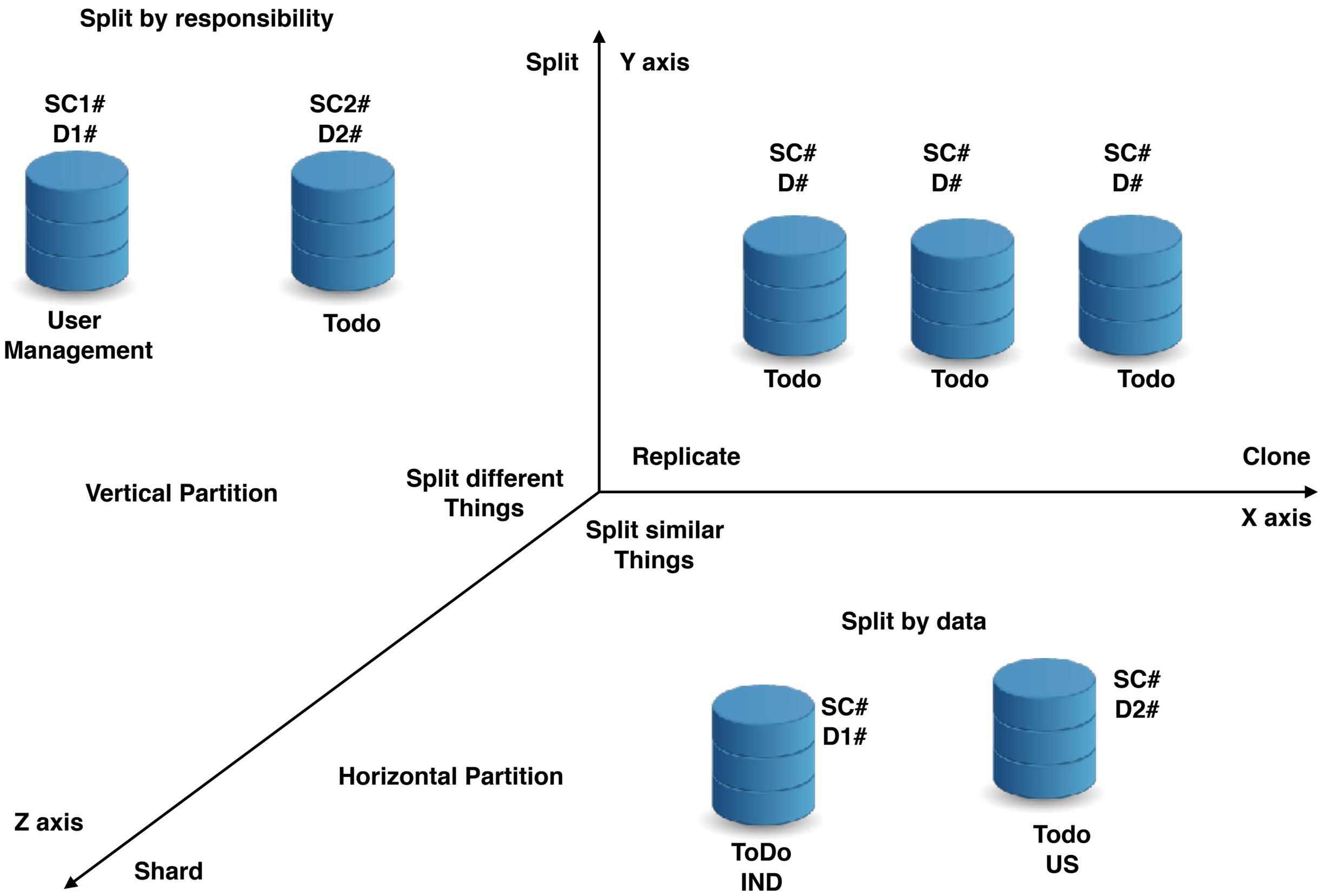
Mutable
$
High Perf

- Local Disk

1

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

Scalability Cube - 50 rules for high Scalability



LOB

Structured

[ACID - transaction]

RDBMS (OLTP)

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

1

Semi Structured

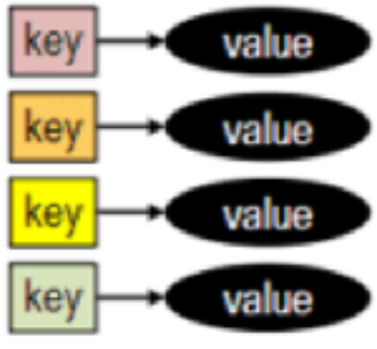
**[* BASE,
SCALE- shareded,
* Flexible Schema]**

Dictionary (key value)

Persistent | **Cache**

- AWS Elastic
- Azure Cache (redis)
- GCP memory Store
- **Redis**, Hazelcast
- Memcached, Ignite

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



2

Nested Objects

Document

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

3

Entity Relationship

Graph

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

4

Rows vs. Documents

	Table
Row 1	Data
Row 2	Data
Row 3	Data
...	:

Document 1

```
{
  data
]
```

Document 2

```
{
  data
}
```

Document 3

```
{
  data
]
```

...

Columns vs. Properties

Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data

Document 1

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 2

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 3

```
{
```

Schema vs. Schema-Free

ID	Name	IsActive	Dob
1	John Smith	True	8/30/1964
2	Sarah Jones	False	2/18/2002
3	Adam Stark	True	7/13/1987

Document 1

```
[
  "id": "1",
  "name": "John Smith",
  "isActive": true,
  "dob": "1964-30-08"
]
```

Document 2

```
{
  "id": "2",
  "fullName": "Sarah Jones",
  "isActive": false,
  "dob": "2002-02-18"
}
```

Document 3

```
{
  "id": "3",
  "fullName":
  {
    "first": "Adam",
    "last": "Stark"
  },
  "isActive": true,
  "dob": "2015-04-19"
}
```

Normalized vs. Denormalized

User Table

UserID	Name	Dob
1	John Smith	8/30/1964

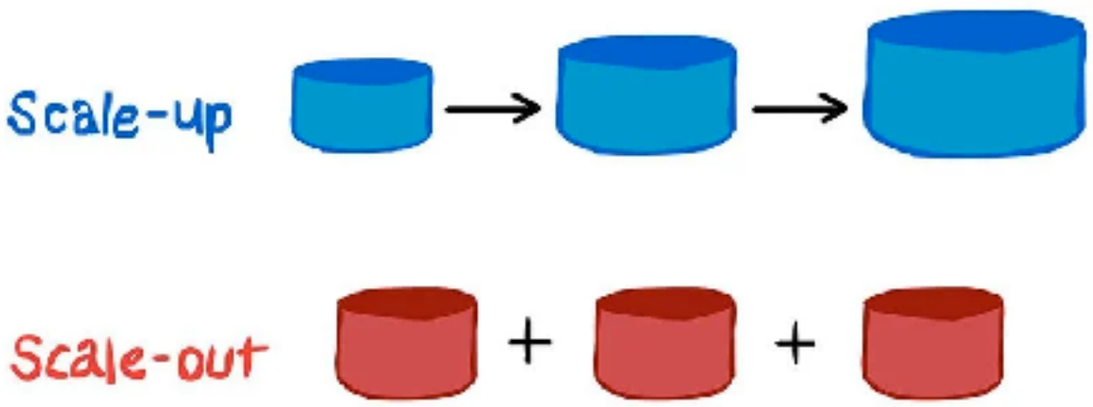
Holdings Table

StockID	UserID	Qty	Symbol
1	1	100	MSFT
2	1	75	WMT

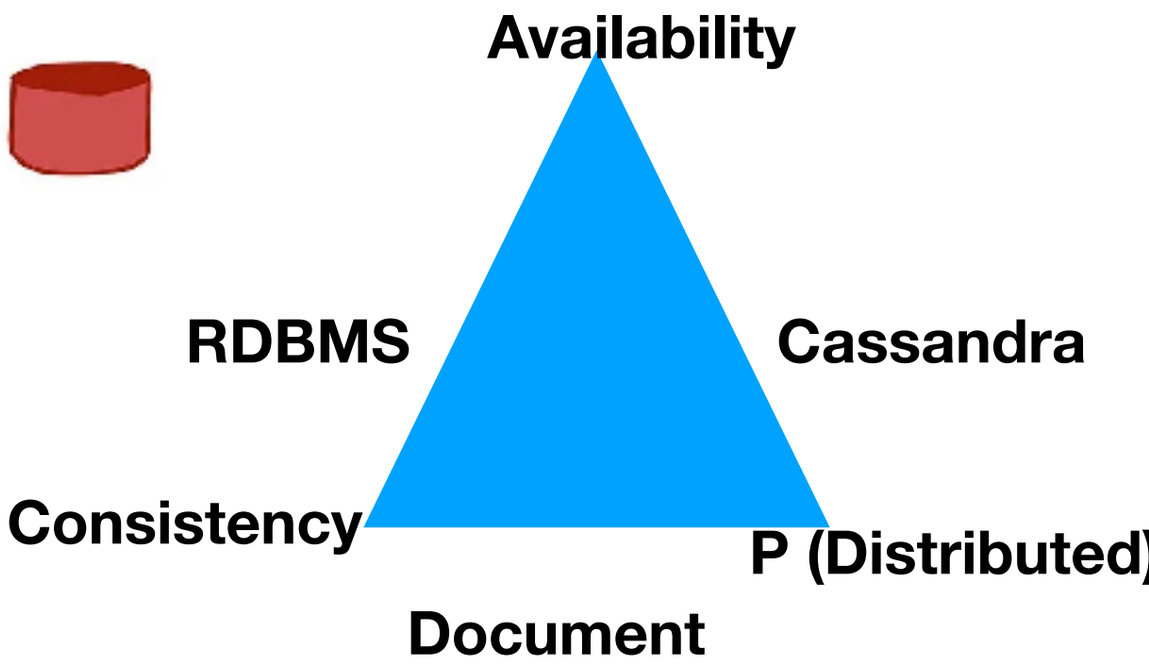
Document

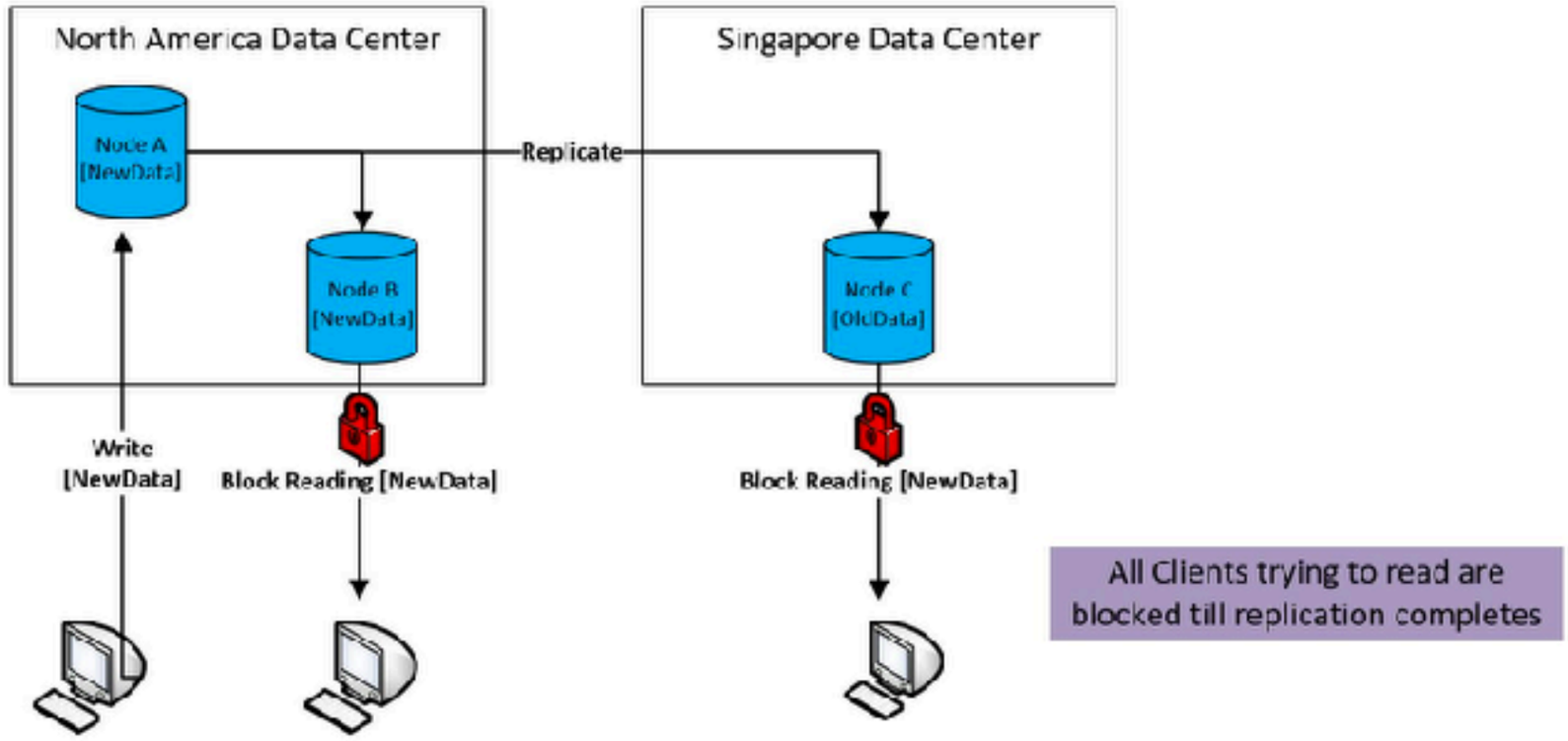
```
{
  "id": "1",
  "name": "John Smith",
  "dob": "1964-30-08",
  "holdings": [
     { "qty": 100, "symbol": "MSFT" },
     { "qty": 75, "symbol": "WMT" }
   ]
}
```

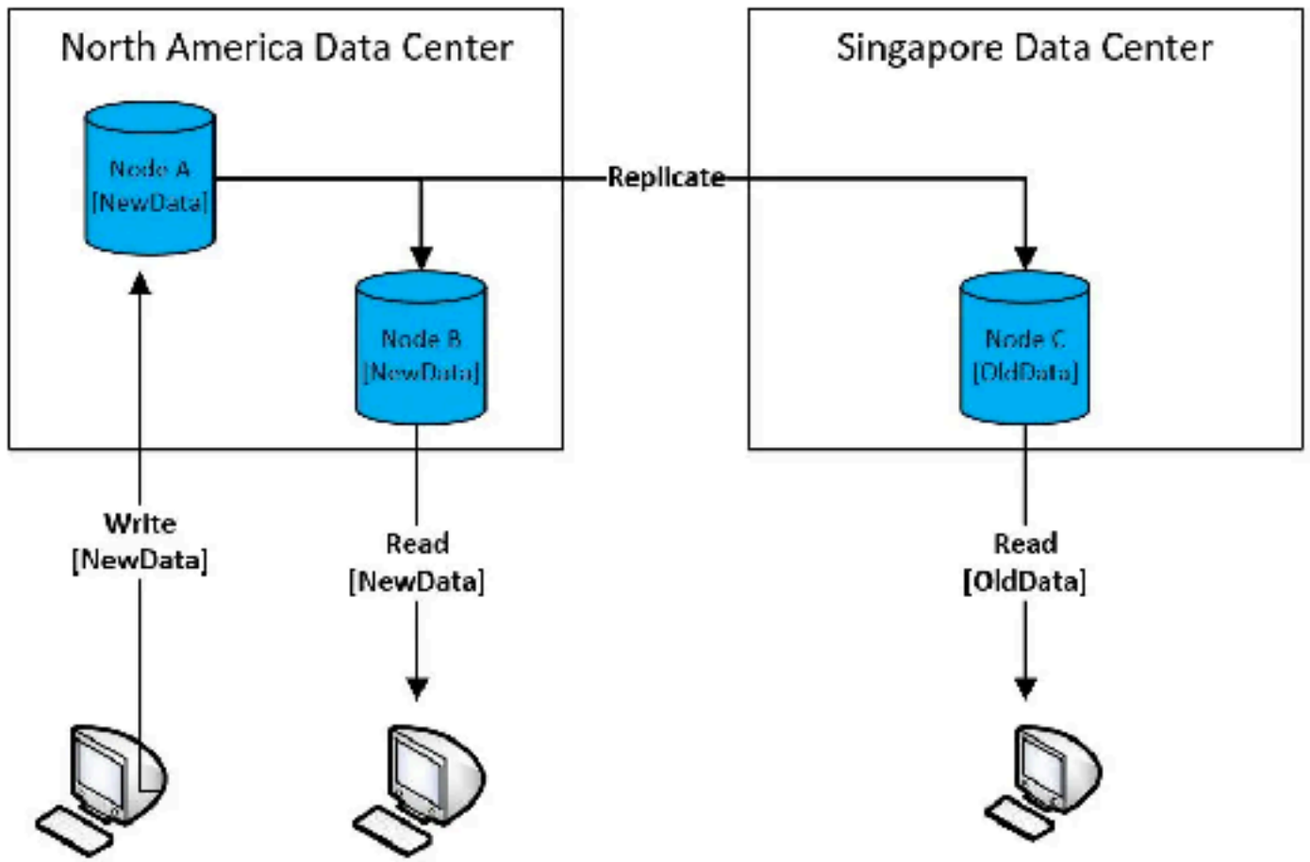
Scale-Up vs. Scale-Out



Strong Consistency vs. Eventual Consistency




North America Data Center
Singapore Data Center
Node A [NewData]
Replicate
Node B [NewData]
Node C [OldData]
Write [NewData]
Block Reading [NewData]
Block Reading [NewData]
All Clients trying to read are blocked till replication completes


North America Data Center
Singapore Data Center
Node A [NewData]
Replicate
Node B [NewData]
Node C [OldData]
Write [NewData]
Read [NewData]
Read [OldData]

COMMON COMPARISONS BETWEEN

MYSQL & NOSQL

	MySQL	NoSQL
Nature	Relational Database	Non-Relational Database
Design	Based on the concept of tables	Based on the concept of documents
Scalable	Tough to scale due to its relational nature	Easily scalable big data compared to relational
Model	Detailed database model is needed before creation	No need of a detailed database model
Community	Vast community available	Community is growing rapidly, but still smaller compared to MySQL
Standardization	SQL is standard language	Lacks standard query language
Schema	The Schema is rigid	The Schema is dynamic
Flexibility	Not very flexible in terms of design	Very flexible in terms of design
Insertions	Inserting new columns or fields affect the design	No effect on the design with the insertion of new columns or fields

Facebook, Wikipedia, Quora, Flickr	MySQL
Twitter	MySQL for tweets and users their own special kind of graph database, FlockDB, built on top of MySQL their own version of Memcached
LinkedIn	Oracle Database and Voldemort
YouTube	MySQL -> BigTable
Microsoft, Myspace	***SQL Server***
Yahoo	***PostgreSQL***

Key Value	○ Twitter uses Redis to deliver your Twitter timeline
	○ Pinterest uses Redis to store lists of users, followers, unfollowers, boards, and more
	○ Coinbase uses Redis to enforce rate limits and guarantee correctness of Bitcoin transactions
	○ Quora uses Memcached to cache results from slower, persistent databases
Graph	○ Walmart uses Neo4j to provide customers personalized, relevant product recommendations and promotions
	○ Medium uses Neo4j to build their social graph to enhance content personalization
	○ Cisco uses Neo4j to mine customer support cases to anticipate bugs
Document	○ SEGA uses MongoDB for handling 11 million in-game accounts
	○ Cisco moved its VSRM (video session and research manager) platform to Couchbase to achieve greater scalability
	○ Aer Lingus uses MongoDB with Studio 3T to handle ticketing and internal apps
	○ Built on MongoDB, The Weather Channel's iOS and Android apps deliver weather alerts to 40 million users in real-time

Analytical

Structured

Semi Structured

Big data Db

DWH (OLAP)

- AWS Redshit
- **Azure Synapse**
- GCP BigQuery
- Snow flakes
- Kudu, Druid, Pinot
- Click house

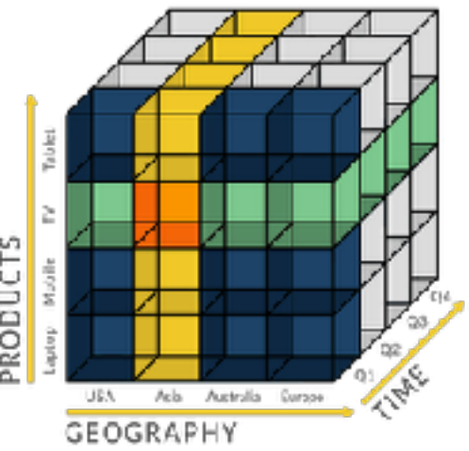
Text Database

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

{…{..{ .. } …}..}
{…{..{ .. } …}..}

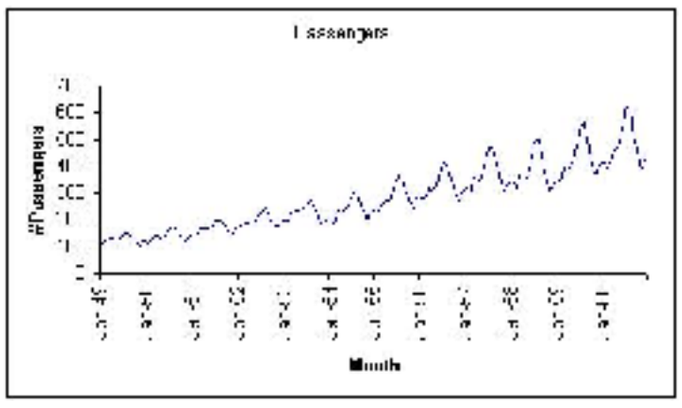
Wide Column (Column db)

- AWS Keyspaces
- Cosmos
- Big Table
- **Hbase, cassandra**
- Scylla



Time Series

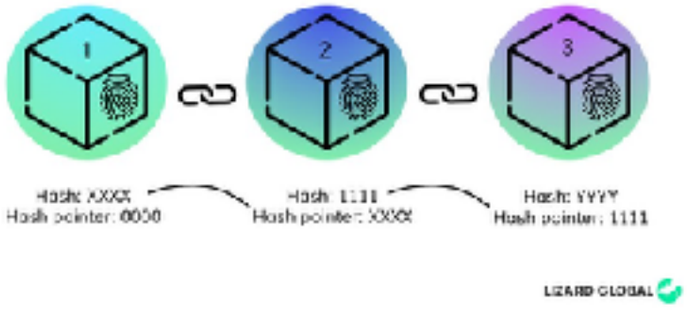
- AWS Timestream
- Cosmos
- Big Table, Big Query
- InfluxDB, Scylla
- OpenTSDB



Audit Trial

Immutable Logger

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



geo Spatial

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

Classic Relational Databases

Name	Age	Nickname	Employee
Gianarno Cuttizzoni Founder & CEO	40	Heidi	◉
Marco Botton Tuttofare	38		☑
Mariah Maclachlan Better Half	41	Potato	⊟
Valerie Liberty Head Chef	16	Val	☑

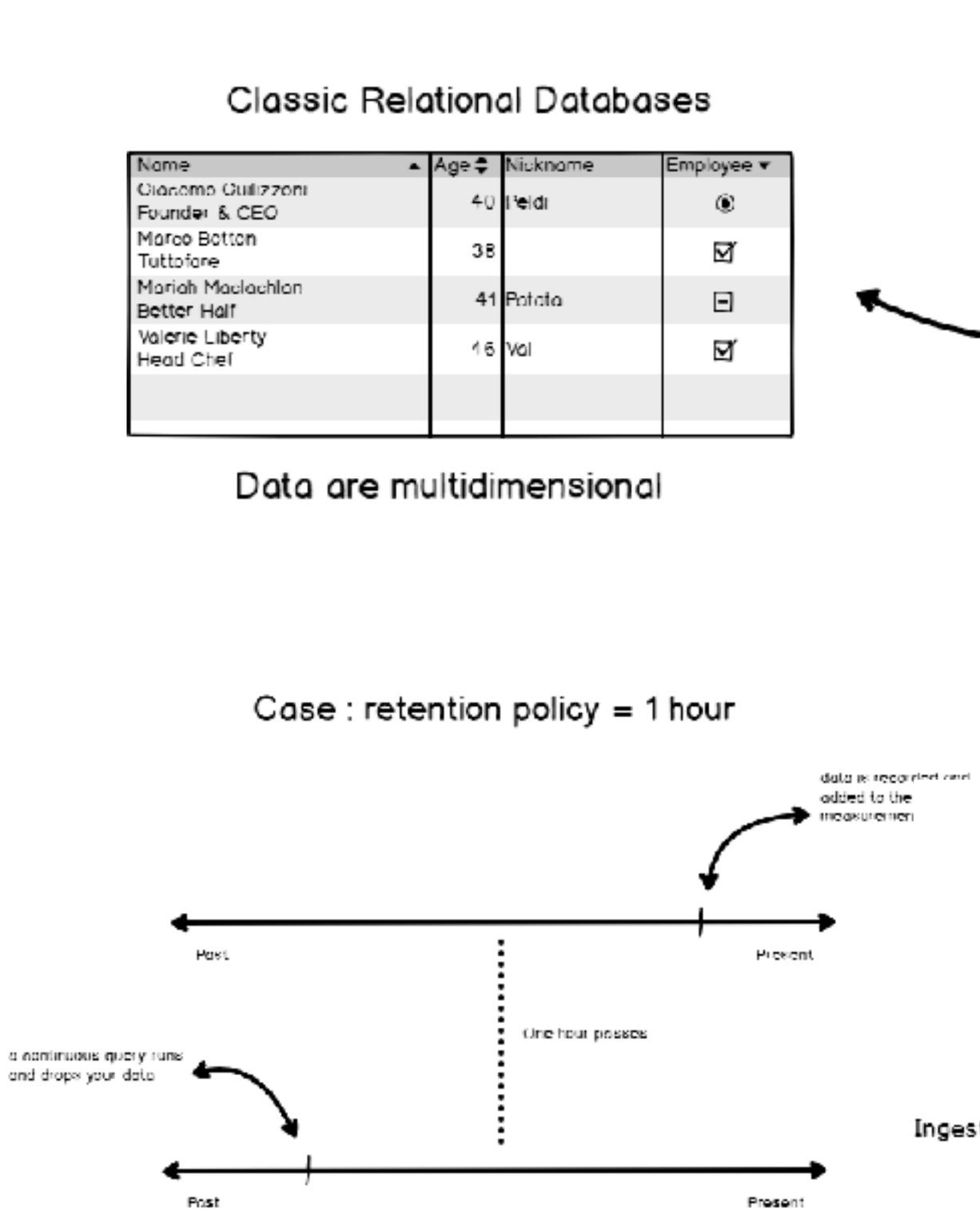
Data are multidimensional

Time Series Databases

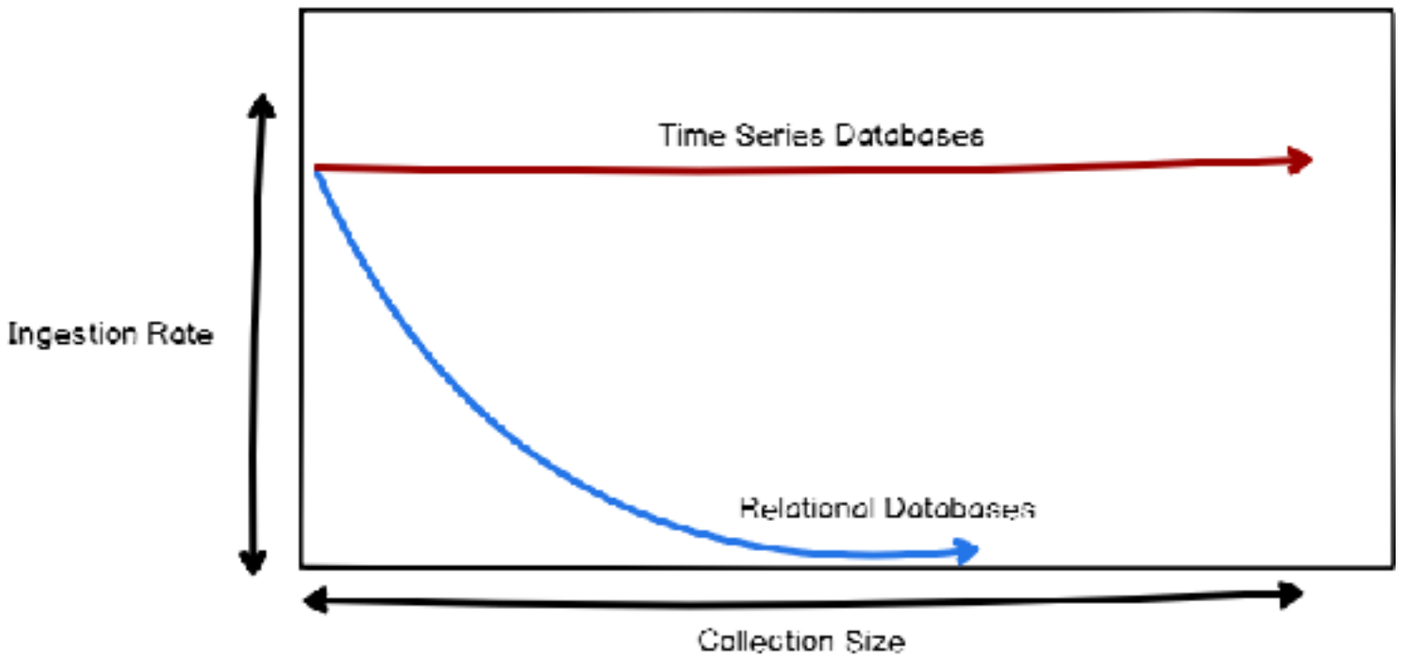
Sensor Temperature	Time
39.6	12/04/19 @ 14:12
41.2	12/04/19 @ 14:13
12.4	14/04/19 @ 12:15
18.5	16/04/19 @ 10:05

Data are aggregated over time

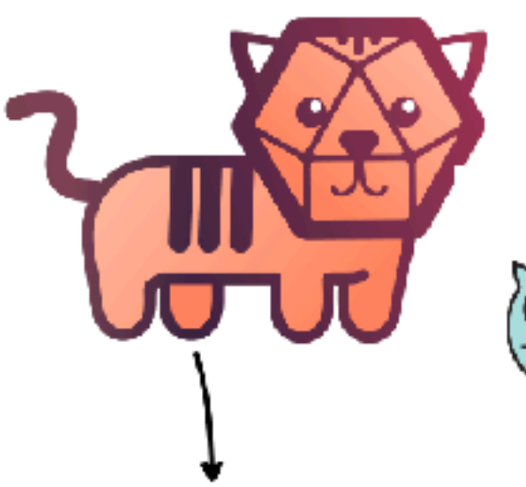
Case : retention policy = 1 hour



DBMS & TSDB Difference

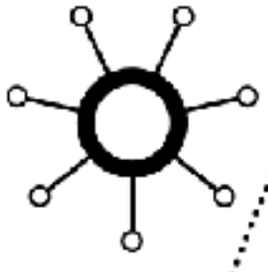


Tools that 'produce' TSDB data



monitoring done via specific query agents

collectd



StatsD

monitoring done via node exporters

Prometheus

Tools that 'consume' TSDB

chronograf

DATADOG

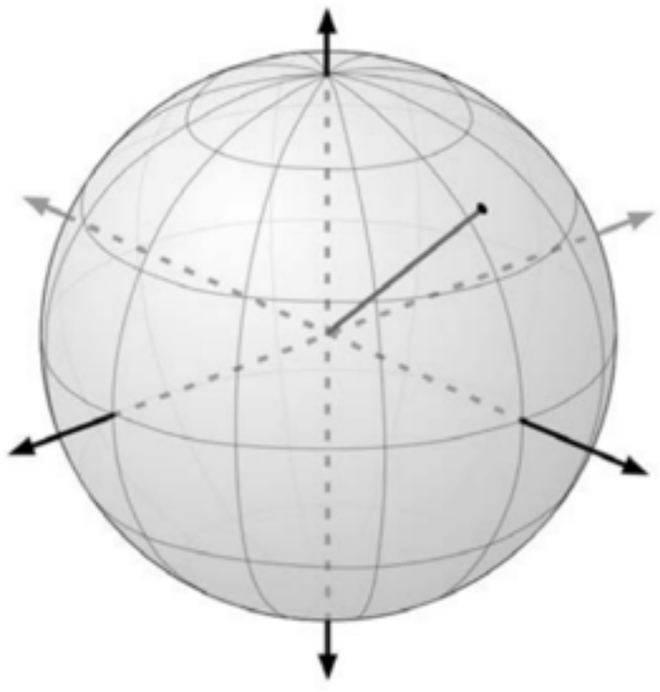
Prometheus

Grafana

kibana

*Non-exhaustive list

Car ID	Departure			Arrival		
	Date-Time	Latitude	Longitude	Date-Time	Latitude	Longitude
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...



Row-oriented

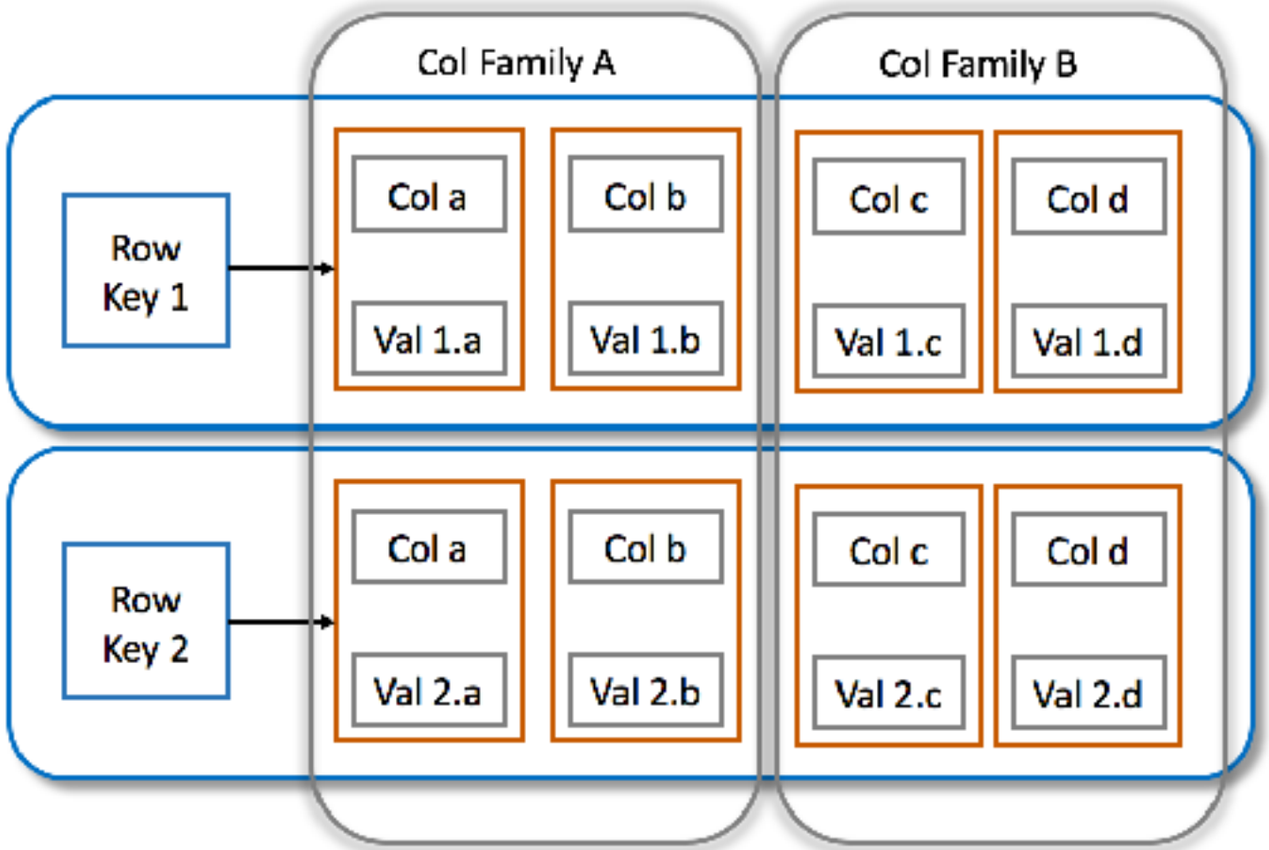
ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

Name	ID
John	001
Karen	002
Bill	003

Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003



RDBMS vs. Columnar: OLTP vs. OLAP

scgupta.link/datastores

OLTP

- Rows are stored together
- Word-aligned heterogeneous values take more space
- Fast row reads and updates
- High ACID transaction throughput
- GBs of data, scale vertically
- Suitable for typical CRUD applications

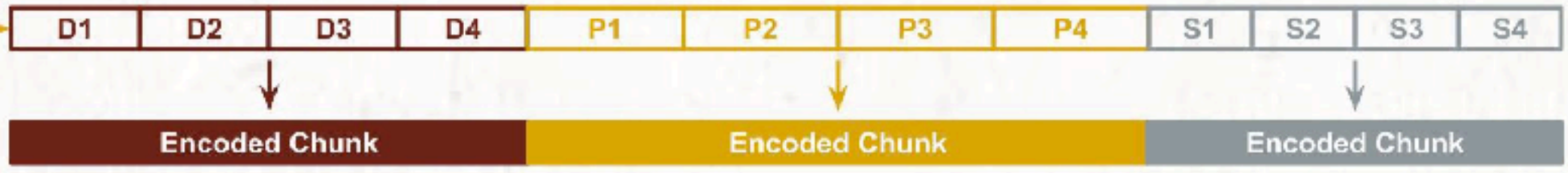
Logical Table

Date	Product	Sale
D1	P1	S1
D2	P2	S2
D3	P3	S3
D4	P4	S4

RDBMS: Row-Oriented Datastore

Columnar: Column-Oriented Datastore

OLAP



- Columns are stored together
- Type aware encoding aids compression
- Fast column summerizations
- High aggregation query throughput
- TBs of data, scale horizontally
- Suitable for analytics and machine learning

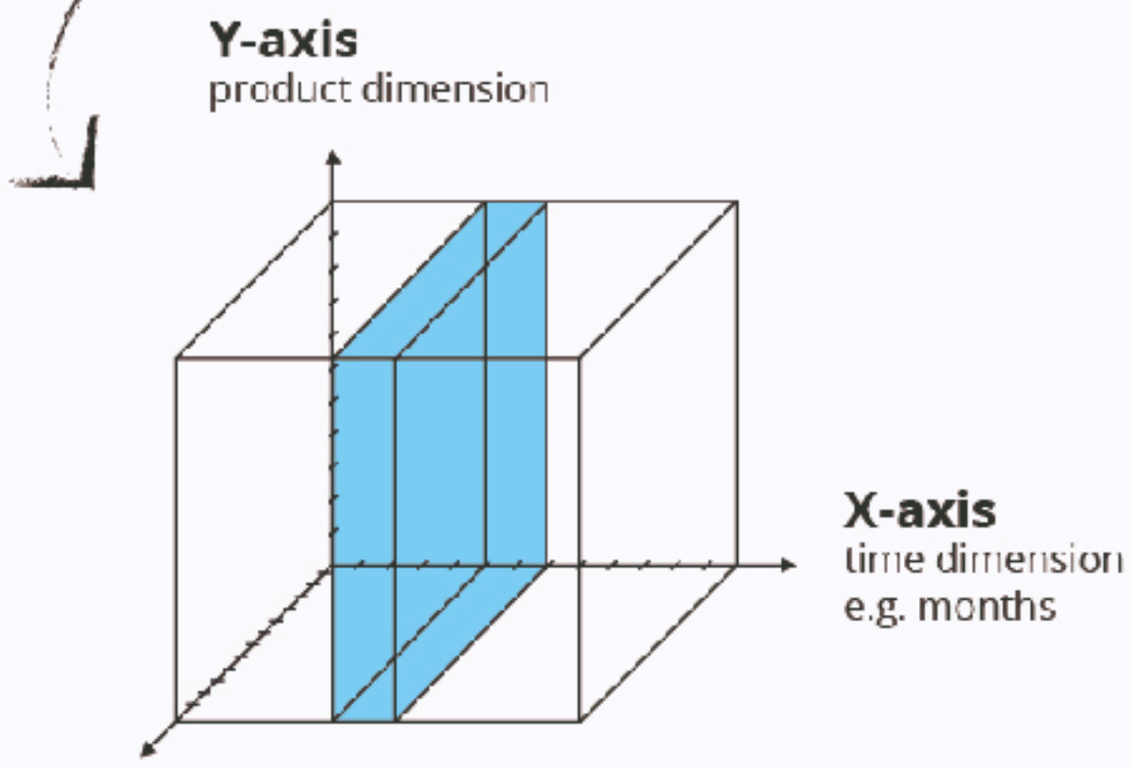
scgupta.me
twitter.com/**scgupta**
linkedin.com/in/**scgupta**

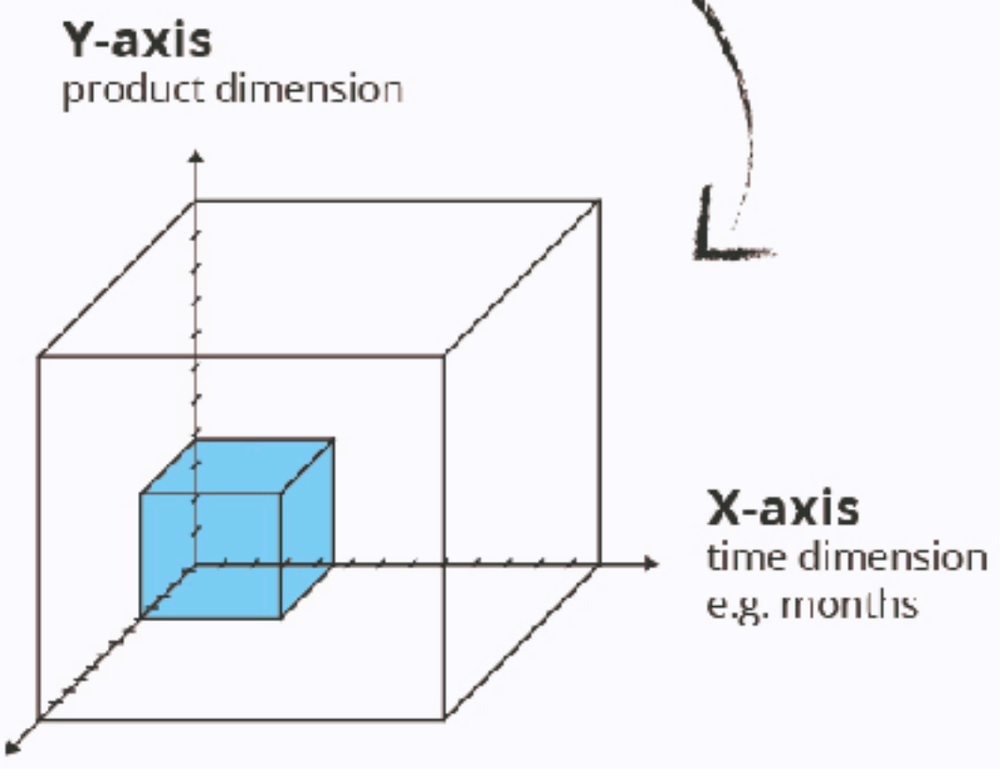
DATABASE
VERSUS
DATA WAREHOUSE

DATABASE	DATA WAREHOUSE
An organized collection of related data which stores data in a tabular format	A central location which stores consolidated data from multiple databases
Contains detailed data	Contains summarized data
Uses Online Transactional Processing (OLTP)	Uses Online Analytical Processing (OLAP)
Helps to perform fundamental operations of a business	Helps to analyze the business
Less fast and less accurate	Faster and accurate
Application oriented	Subject oriented
Tables and joins are complex because they are normalized	Tables and joins are simple because they are denormalized
Design is helped by entity relationship modelling	Design is helped by data modelling technique

Slicing & Dicing

OLAP cube operations



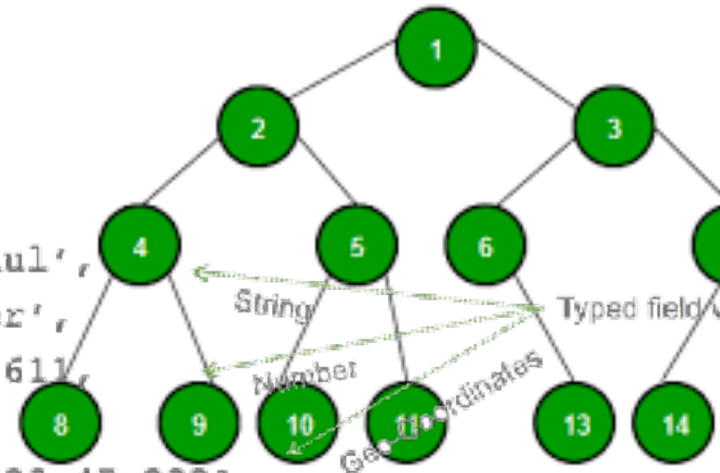


Rdbms (Referential Integrity)

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

tblGender	
ID	Gender
1	Male
2	Female
3	Unknown

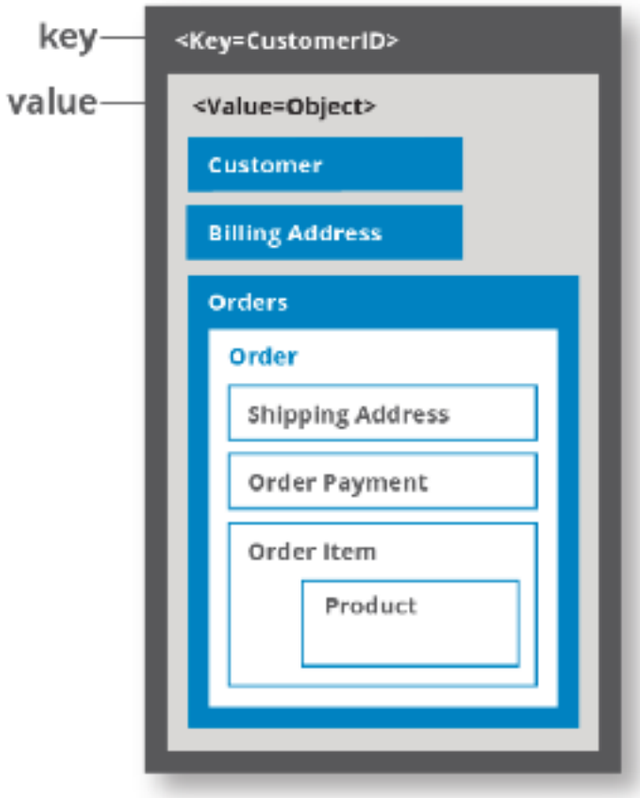
Fields

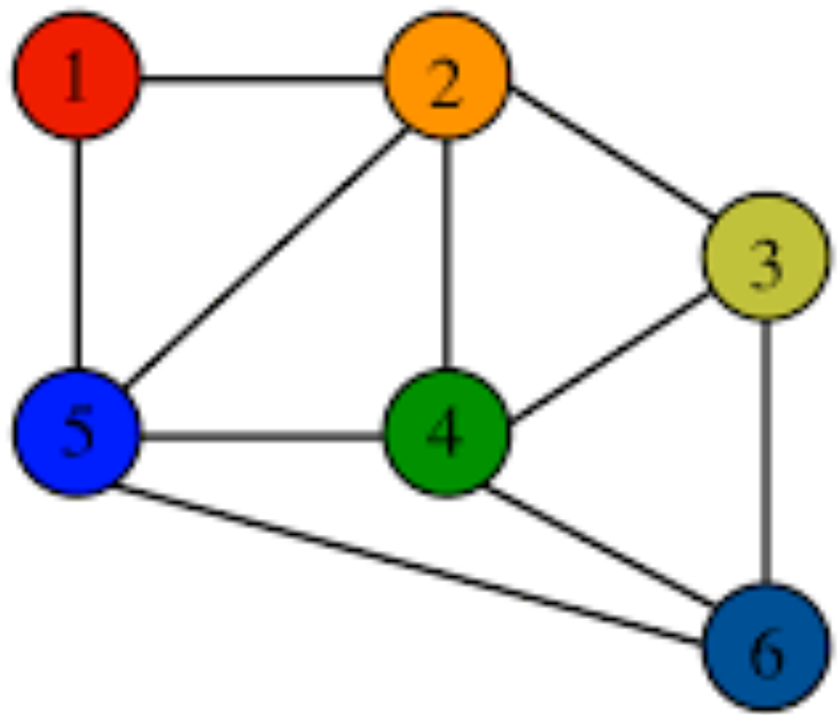



```
{
  first_name: 'Paul',
  surname: 'Miller',
  cell: 447557505611,
  city: 'London',
  location: [45.123,47.232],
  Profession: ['banking', 'finance', 'trader'],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  ]
}
```

Fields can contain an array of documents

key	value
123	123 Main St.
126	(805) 477-3900





Wide Column

- Spotify uses Cassandra to store user profile attributes and metadata about artists, songs, etc. for better personalization
- Facebook initially built its revamped Messages on top of HBase, but is now also used for other Facebook services like the Nearby Friends feature and search indexing
- Outbrain uses Cassandra to serve over 190 billion personalized content recommendations each month

Files

Databases (Cloud / On Premises)

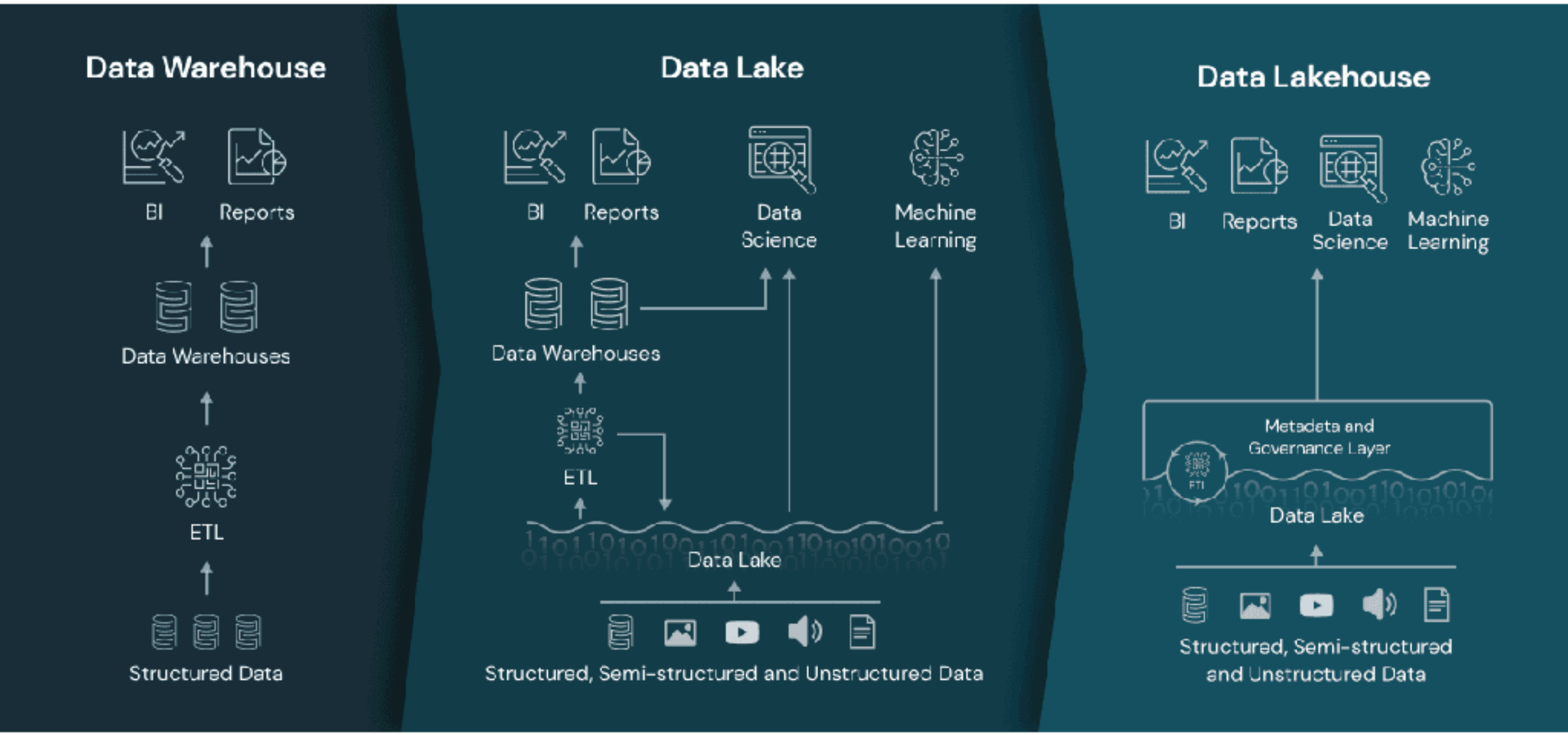
Cloud Sources

Audio/ Video/Documents

Data Lake

WebApps, Sensors, IoT

Social Feeds


Data Warehouse
BI
Reports
Data Warehouses
ETL
Structured Data
Data Lake
BI
Reports
Data Science
Machine Learning
Data Warehouses
ETL
Data Lake
Structured, Semi-structured and Unstructured Data
Data Lakehouse
BI
Reports
Data Science
Machine Learning
Metadata and Governance Layer
ETL
Data Lake
Structured, Semi-structured and Unstructured Data

Data Source Tier

Structured

Database,
Data warehouse,
Lob App logs

unstructured

Images,
Audio,
Word doc,
Spread Sheet

semi-structured

XMl,
Html,
Json,
Emails

Landing Zone
(immutable)

Raw

handling missing values,
filtering outliers,
& Validated data

Stage X

Stage Y

Highly refined views
of the data

Curated

Data Lake
Azure Blob
AWS S3
Azure Data Lake

Data WareHouse / Mart
Aws Redshift
Azure Synapse Analytics

Insight/Analytics Tier

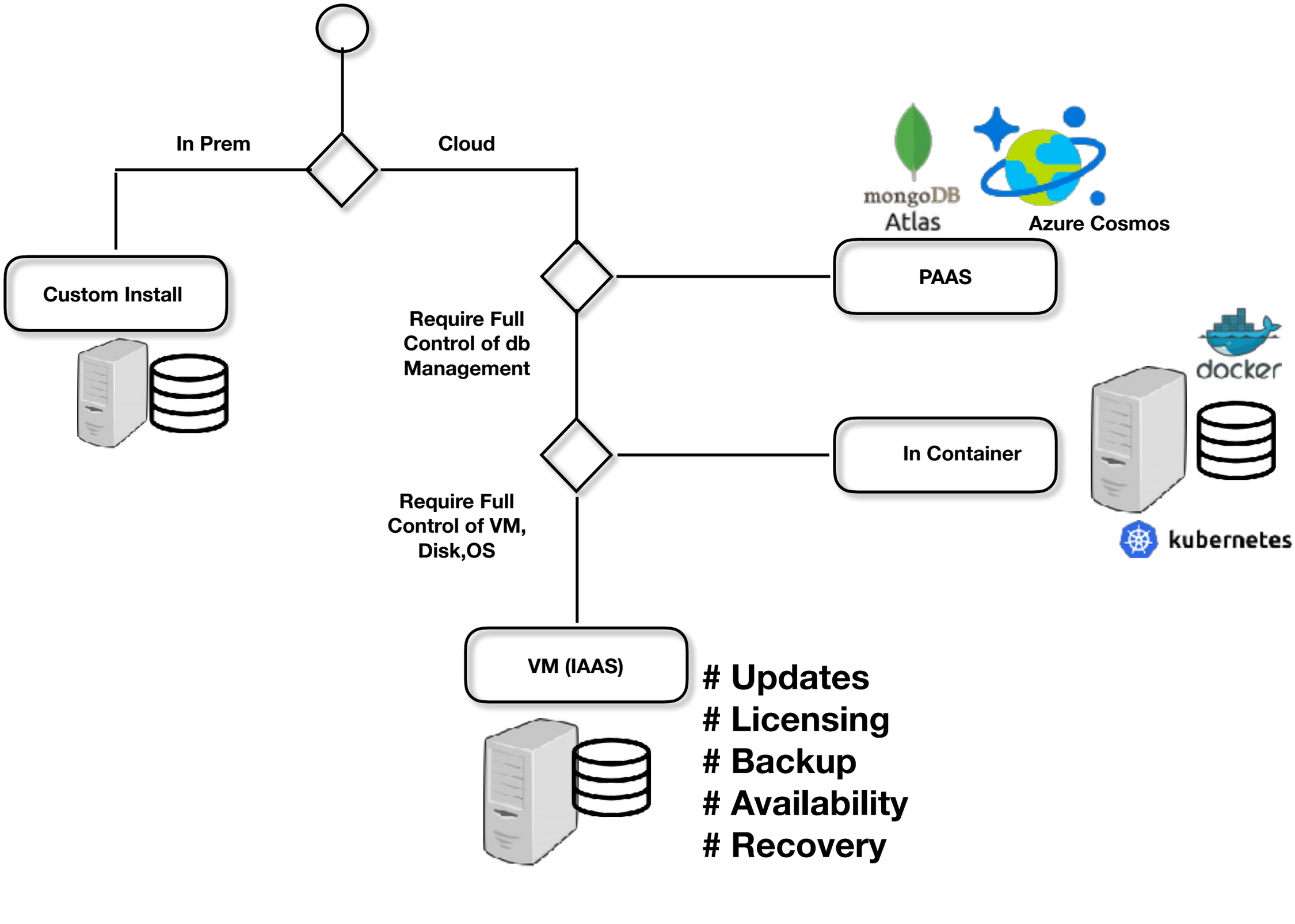
BI Tools

Applications

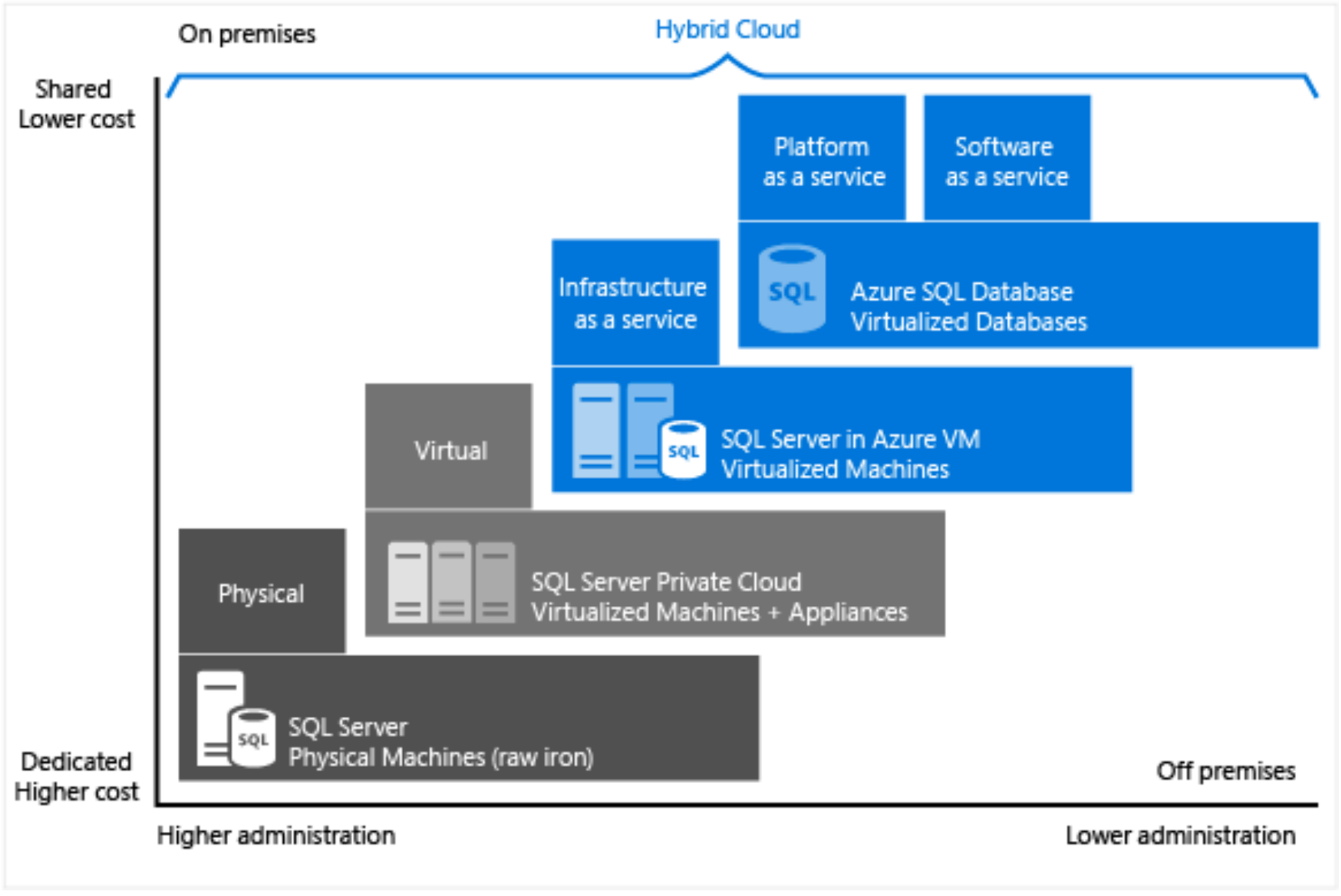
Automations

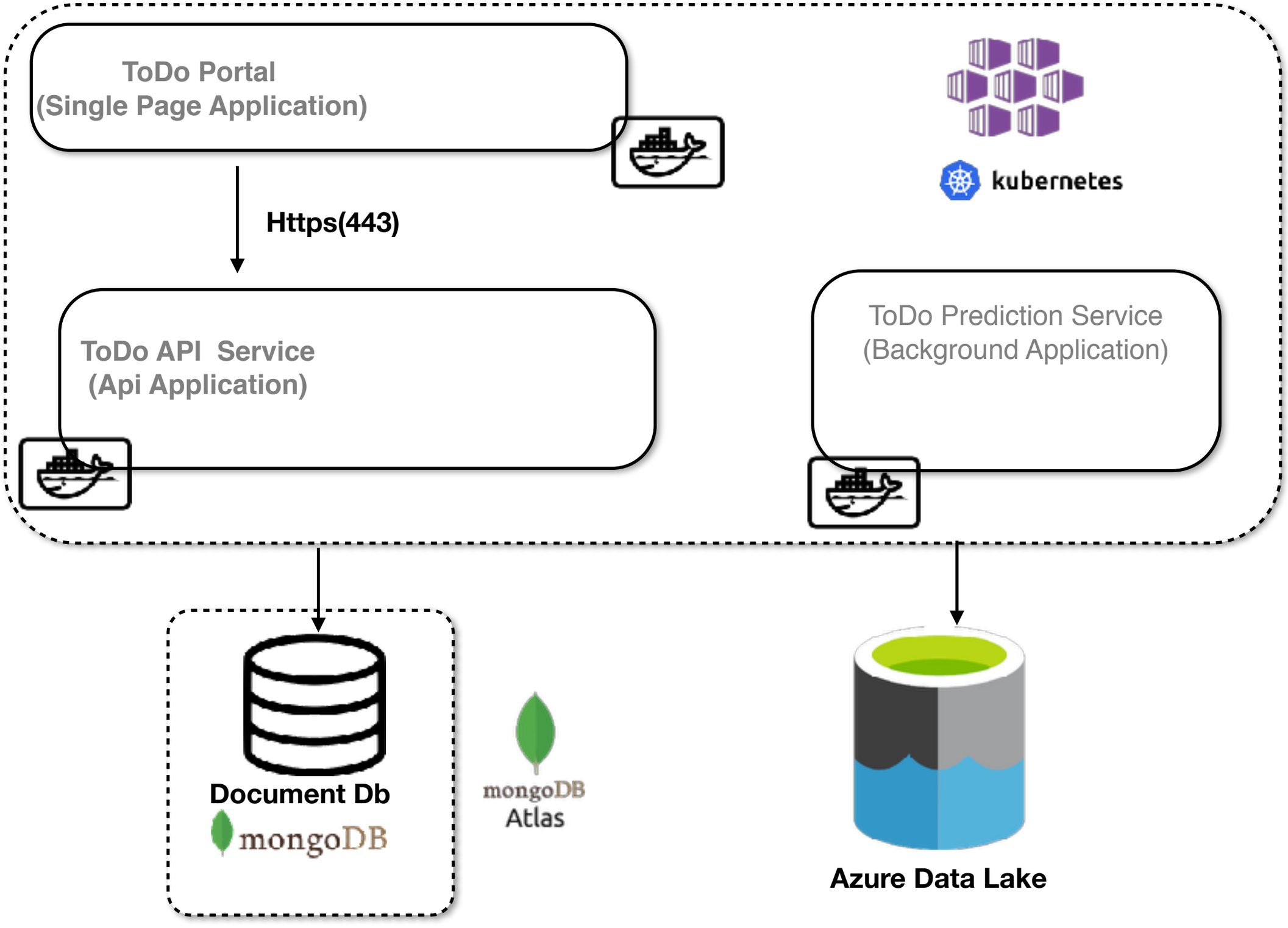
QuickSight
Power BI

Deploying Data Tier

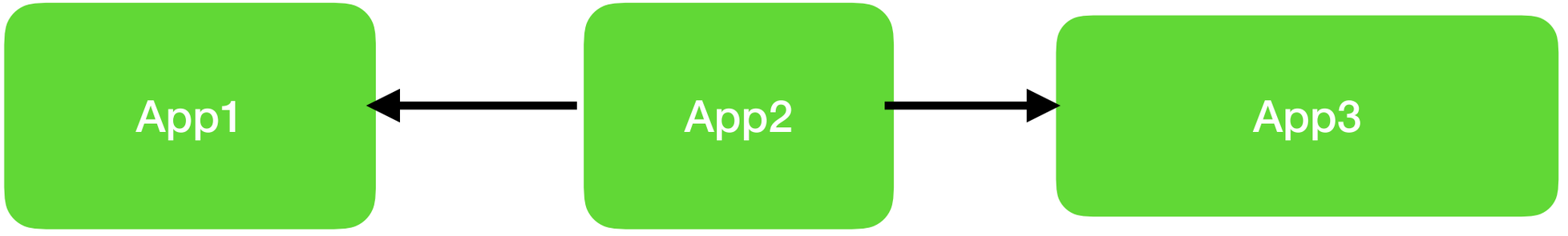


Deploying Data Tier

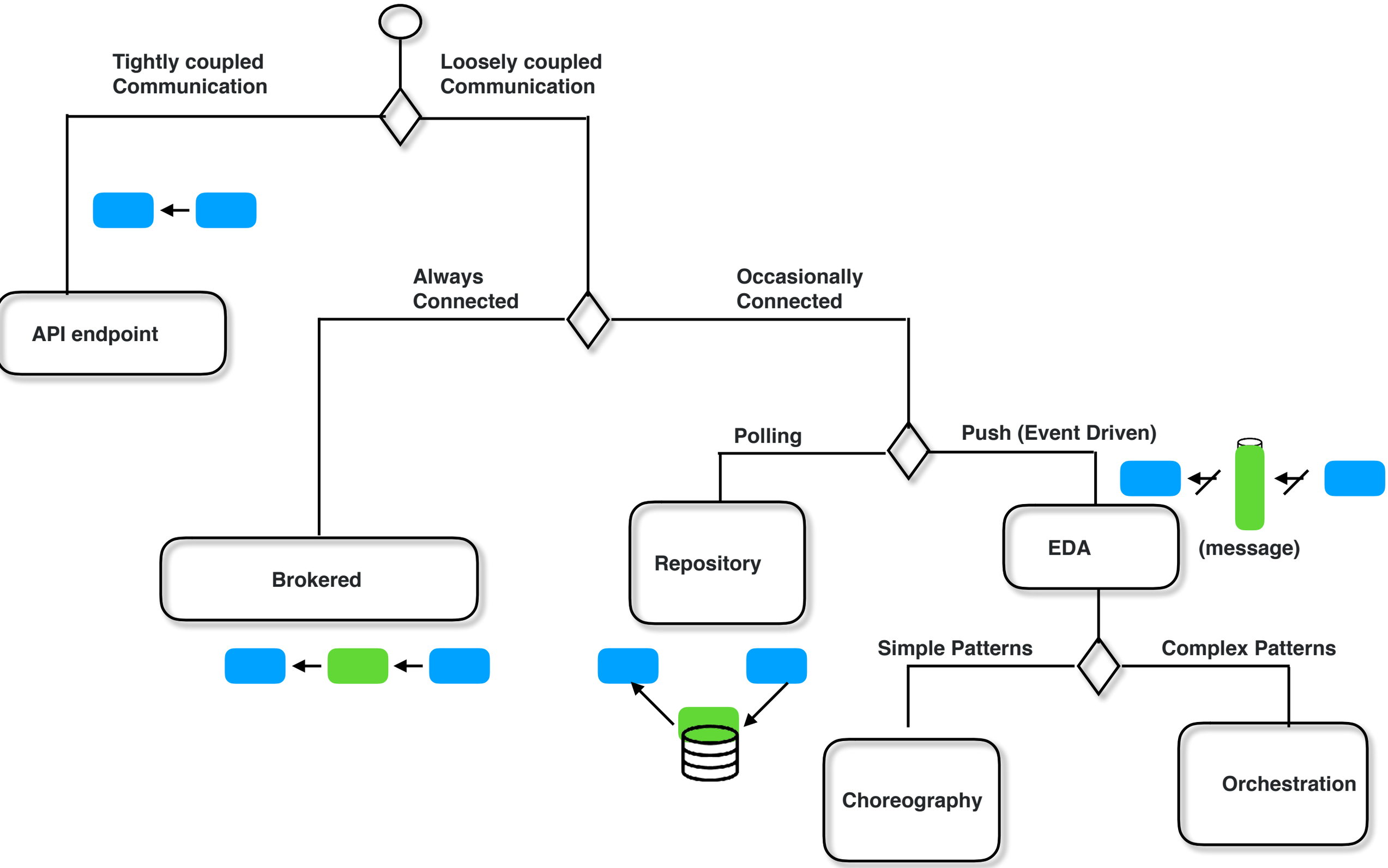


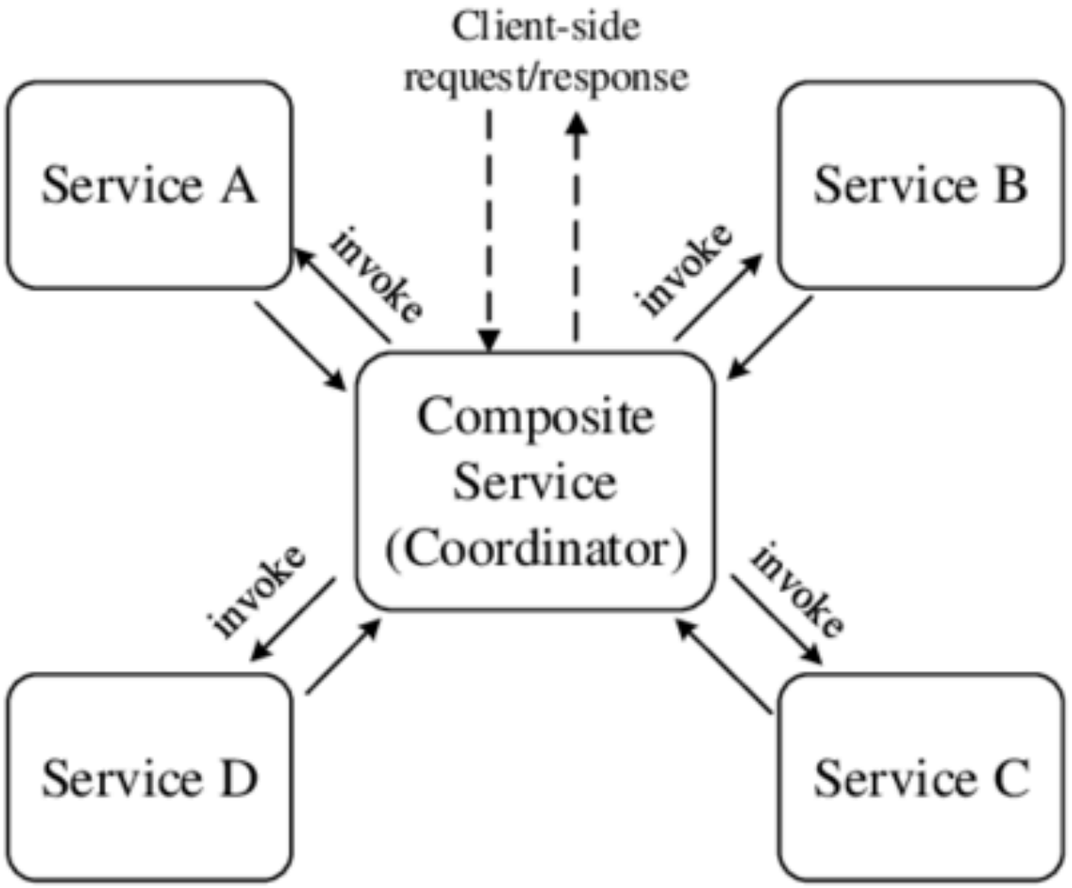

ToDo Portal
(Single Page Application)
Https(443)
ToDo API Service
(Api Application)
ToDo Prediction Service
(Background Application)
kubernetes
Document Db
mongoDB
mongoDB
Atlas
Azure Data Lake

System Integration

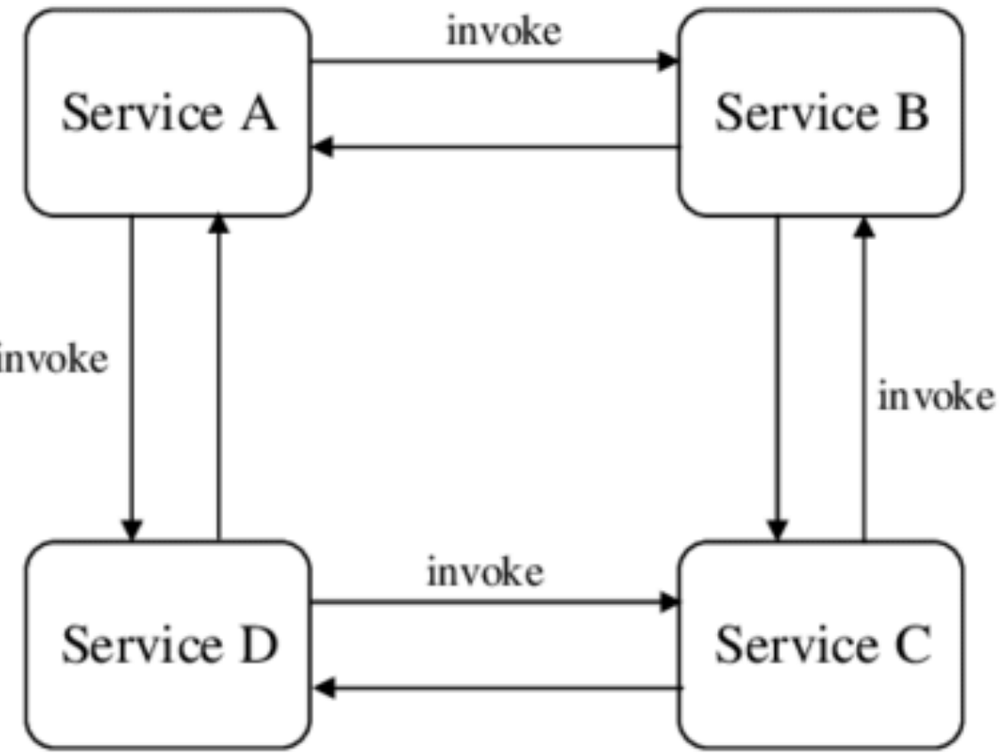


Choose Communication Mechanism

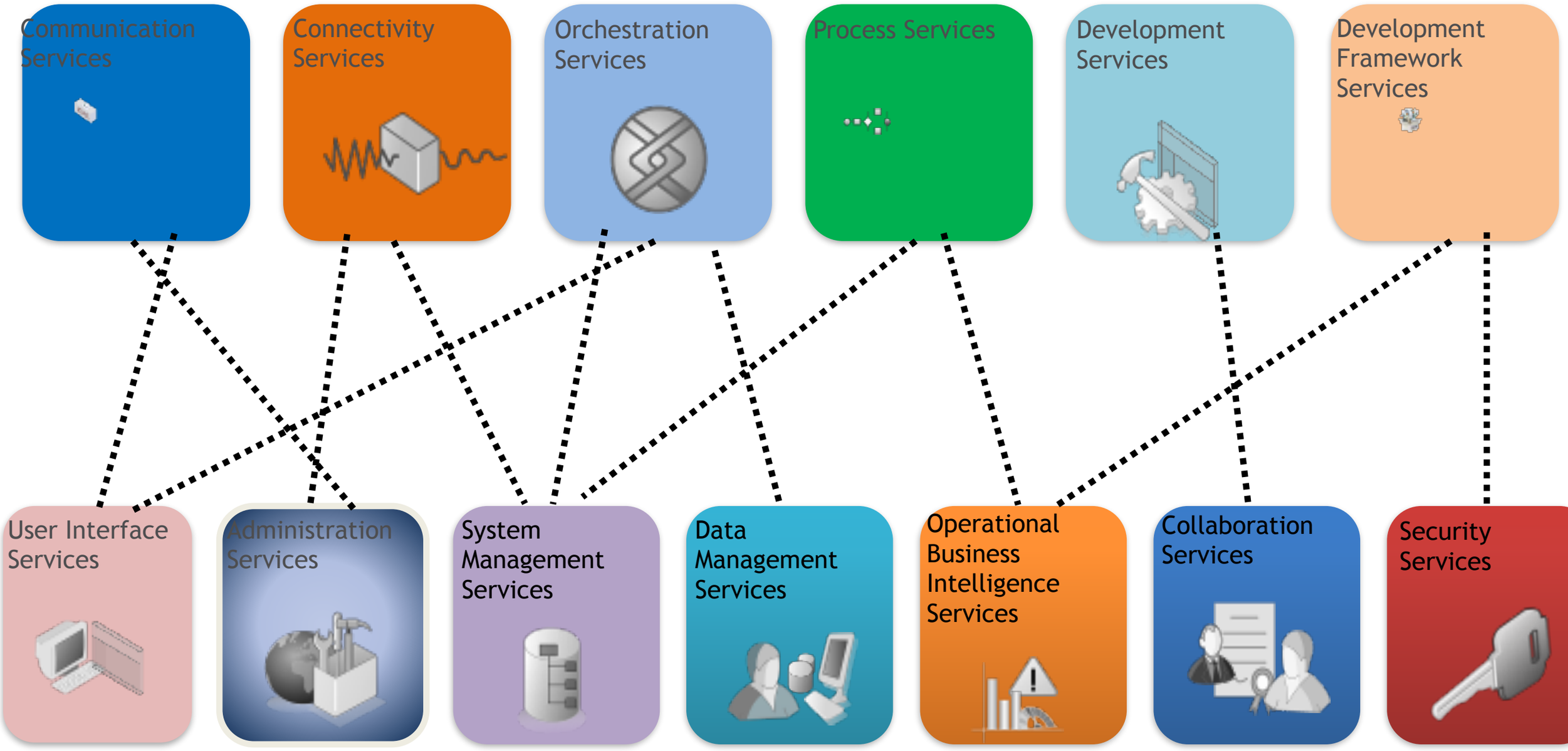




(a) Web Service Orchestration



(b) Web Service Choreography


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

Communication Services

Connectivity Services

Orchestration Services

Process Services

Development Services

Development Framework Services

User Interface Services

Administration Services

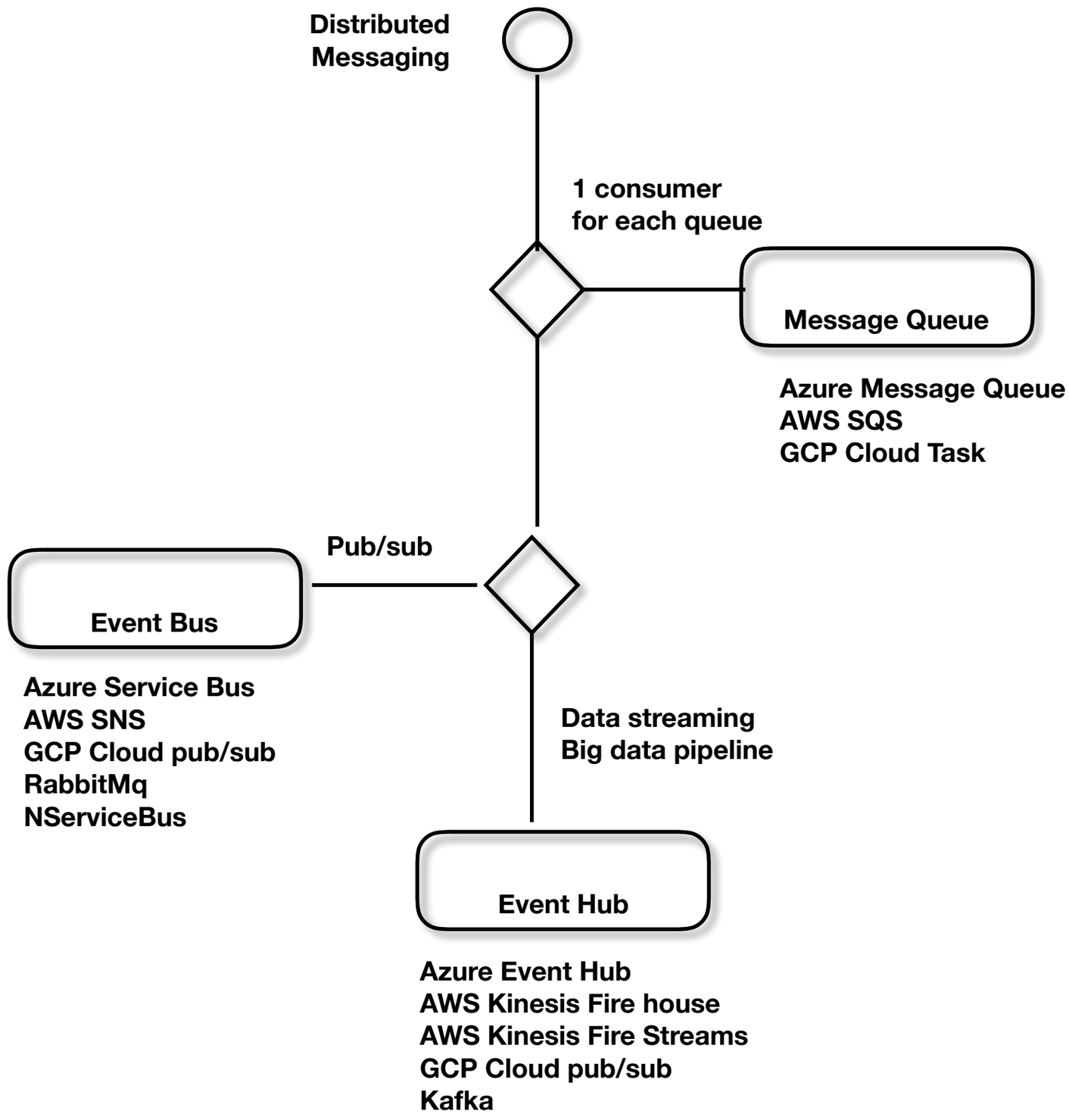
System Management Services

Data Management Services

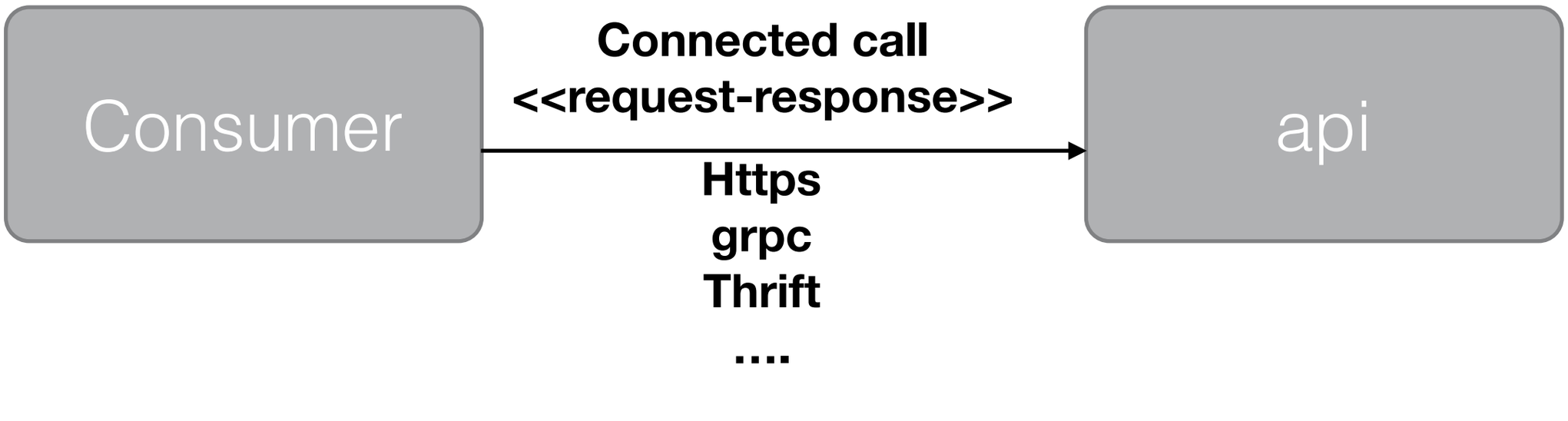
Operational Business Intelligence Services

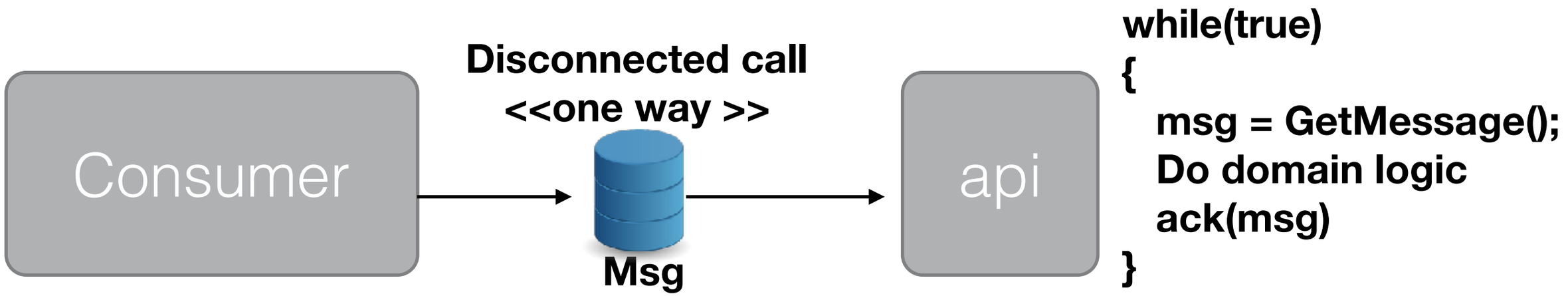
Collaboration Services

Security Services

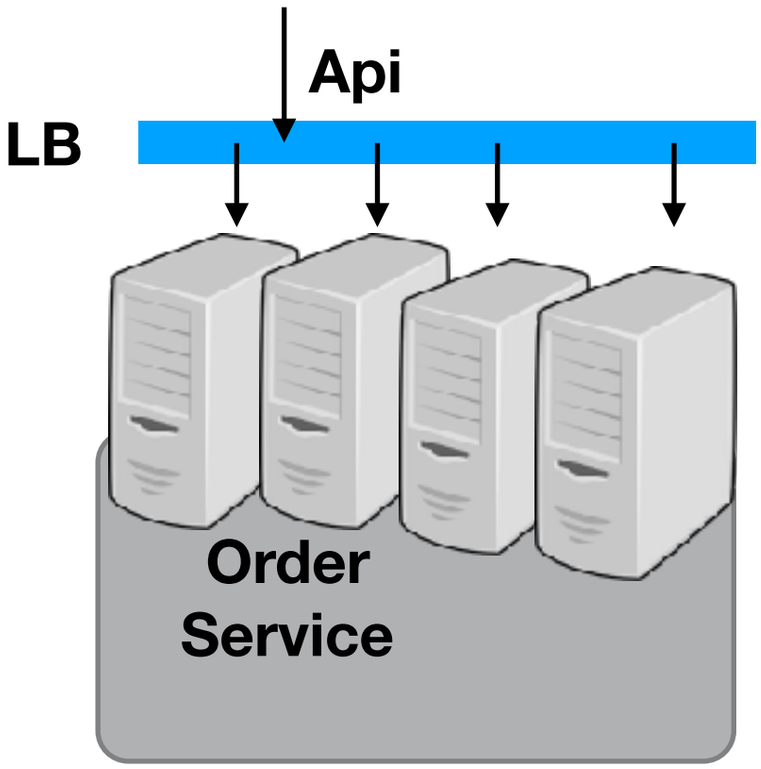

Distributed
Messaging
1 consumer
for each queue
Message Queue
Azure Message Queue
AWS SQS
GCP Cloud Task
Pub/sub
Event Bus
Azure Service Bus
AWS SNS
GCP Cloud pub/sub
RabbitMq
NServiceBus
Data streaming
Big data pipeline
Event Hub
Azure Event Hub
AWS Kinesis Fire house
AWS Kinesis Fire Streams
GCP Cloud pub/sub
Kafka

Logical view
Communication





API call



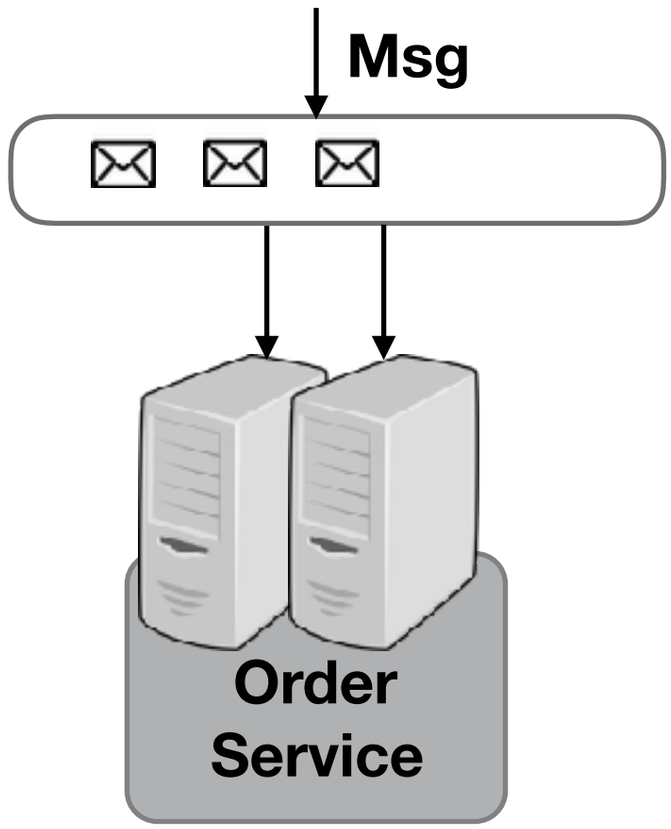
immediate Consistency

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



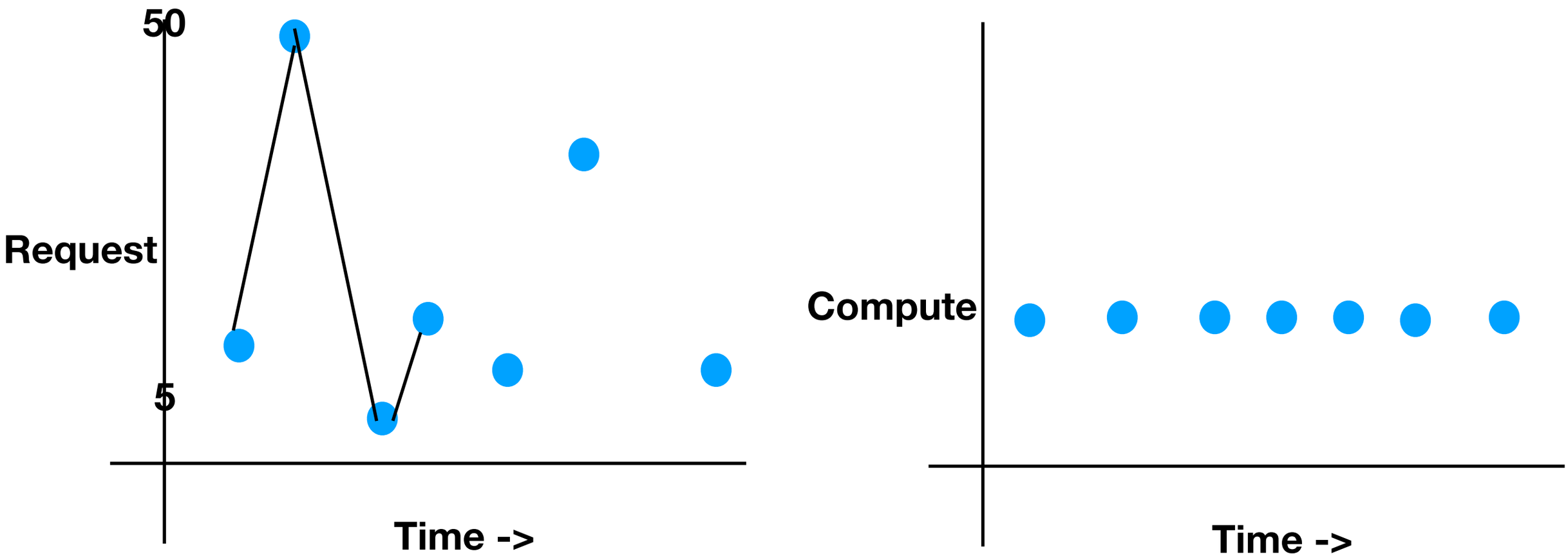
Eventual Consistency

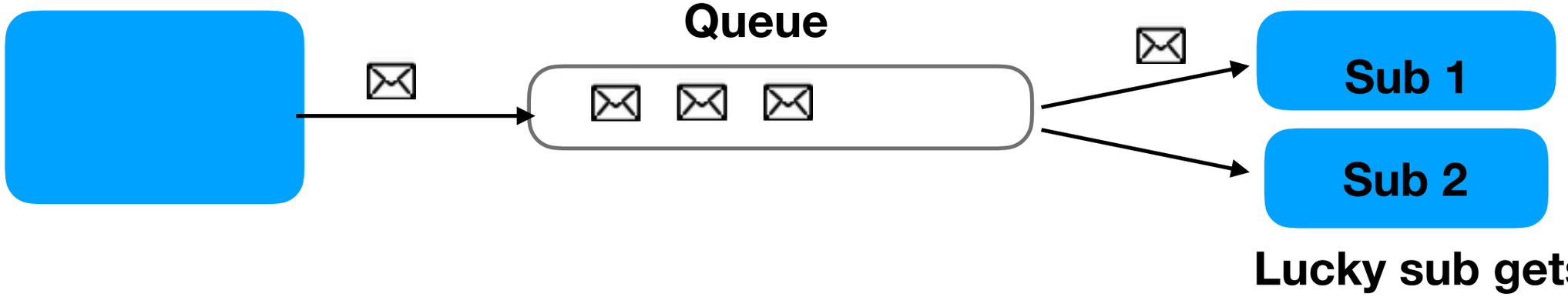
more scalable
no of server’s depends on Avg load
(Load Leveling)

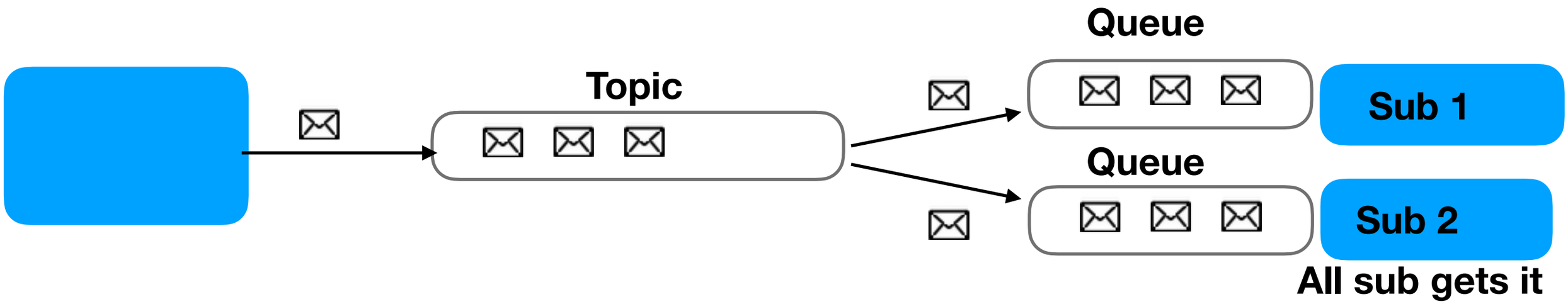
more reliable
If message are not ACK, messages
reappear in Queue

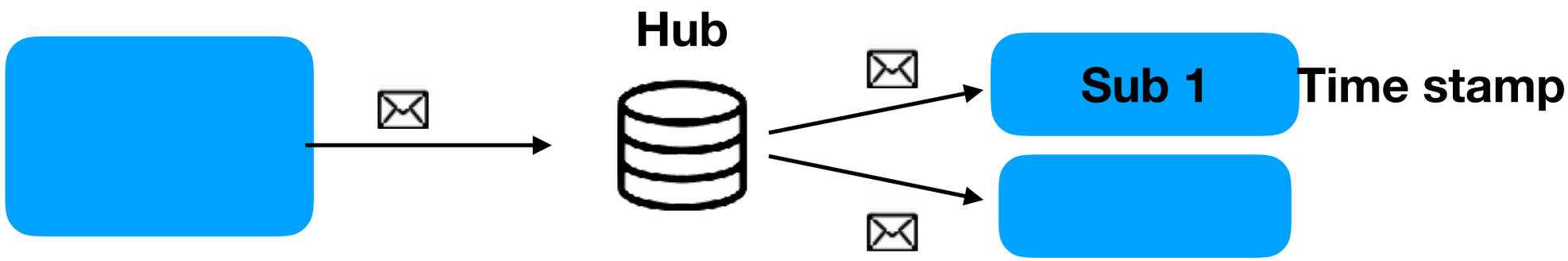
no loss of data
Gets queued

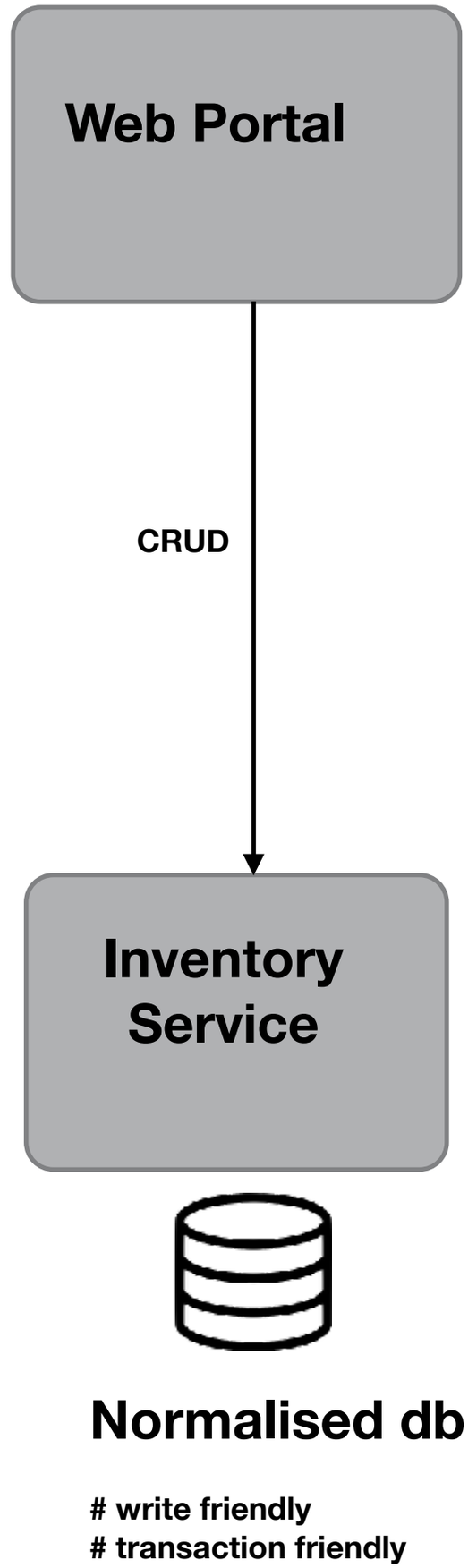
Load Leveling




Queue
Sub 1
Sub 2
Lucky sub gets it


Topic
Queue
Sub 1
Queue
Sub 2
All sub gets it


Hub
Sub 1
Time stamp
sub can read msg n times
until TTL


Web Portal
CRUD
Inventory Service
Normalised db
write friendly
transaction friendly

3rd normal

Pupil Table			
Pupil ID	**First Name**	**Last Name**	**ClassID**
1	Bob	Jones	1
2	Bill	Jones	2
3	Fred	Jones	1

ClassTable		
Class ID	**Class Name**	**Room**
1	CompSci 101	S16
2	English 101	M42

 # write friendly
 # transaction 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

 # read friendly

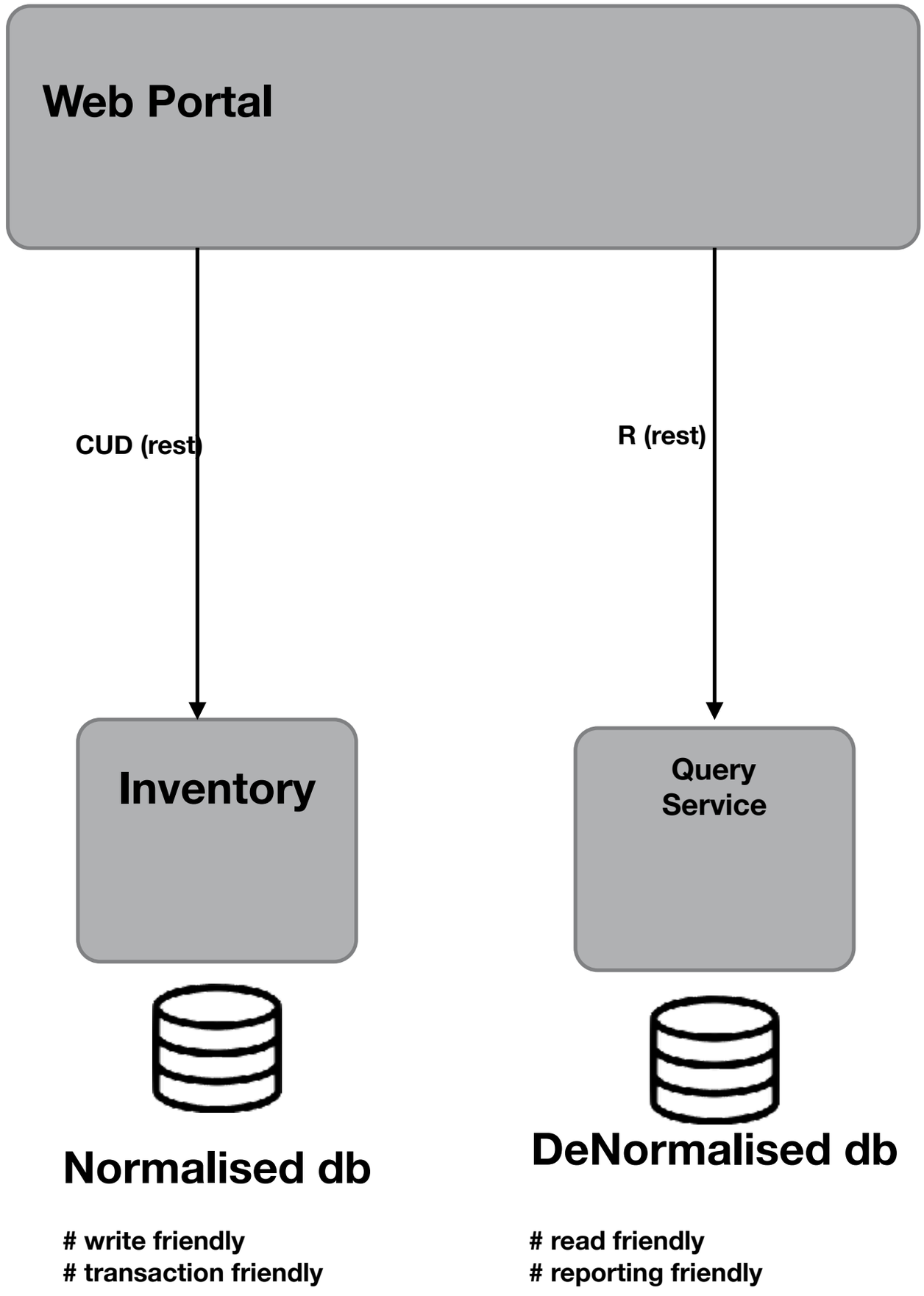
Web Portal

CRUD

Inventory Service

DeNormalised db

 # read friendly
 # reporting friendly


Web Portal
CUD (rest)
R (rest)
Inventory
Query
Service
Normalised db
DeNormalised db
write friendly
transaction friendly
read friendly
reporting friendly

Portal

Create Order

Read (rest)

Create Order

Create Order

Inventory

do() {… }
undo() {…}

Query Service

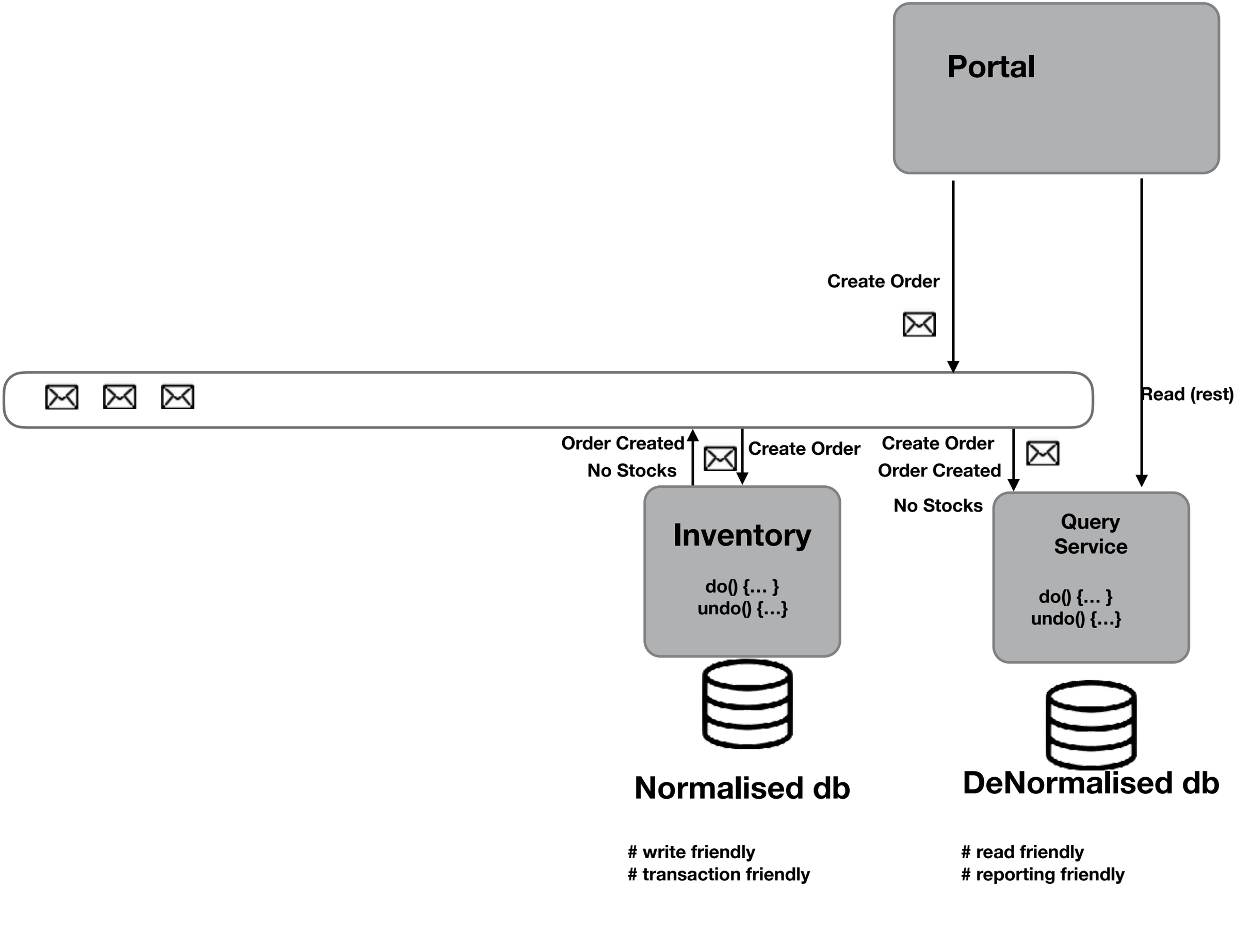
do() {… }
undo() {…}

Normalised db

write friendly
transaction friendly

DeNormalised db

read friendly
reporting friendly


Portal
Create Order
Read (rest)
Order Created
No Stocks
Create Order
Create Order
Order Created
No Stocks
Inventory
do() {… }
undo() {…}
Query
Service
do() {… }
undo() {…}
Normalised db
DeNormalised db
write friendly
transaction friendly
read friendly
reporting friendly

Portal

Accounting

Create Order

Order Created

Read (rest)

Order Created
No Stocks

Create Order

Create Order
Order Created
No Stocks

Inventory

do() {… }
undo() {…}

Query
Service

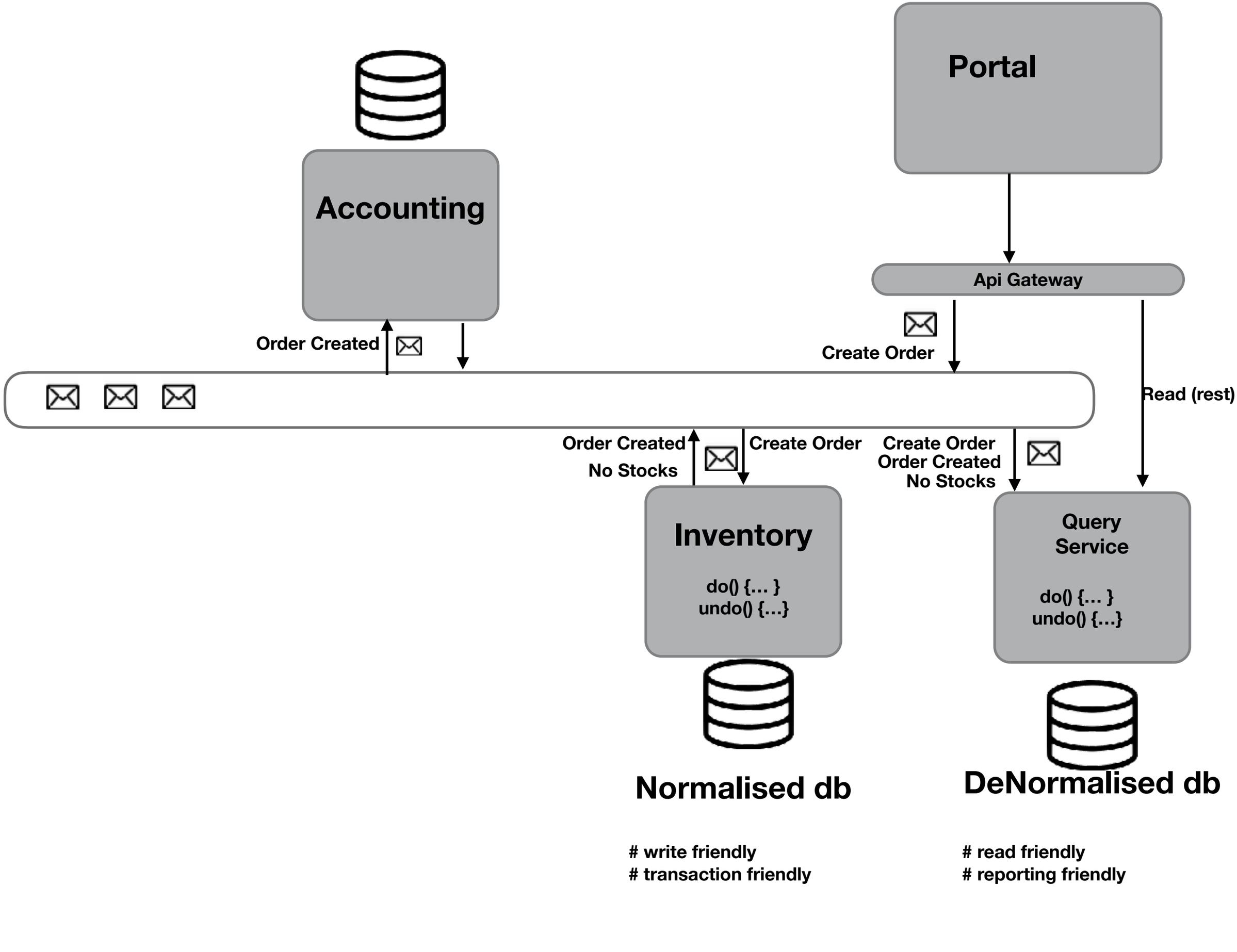
do() {… }
undo() {…}

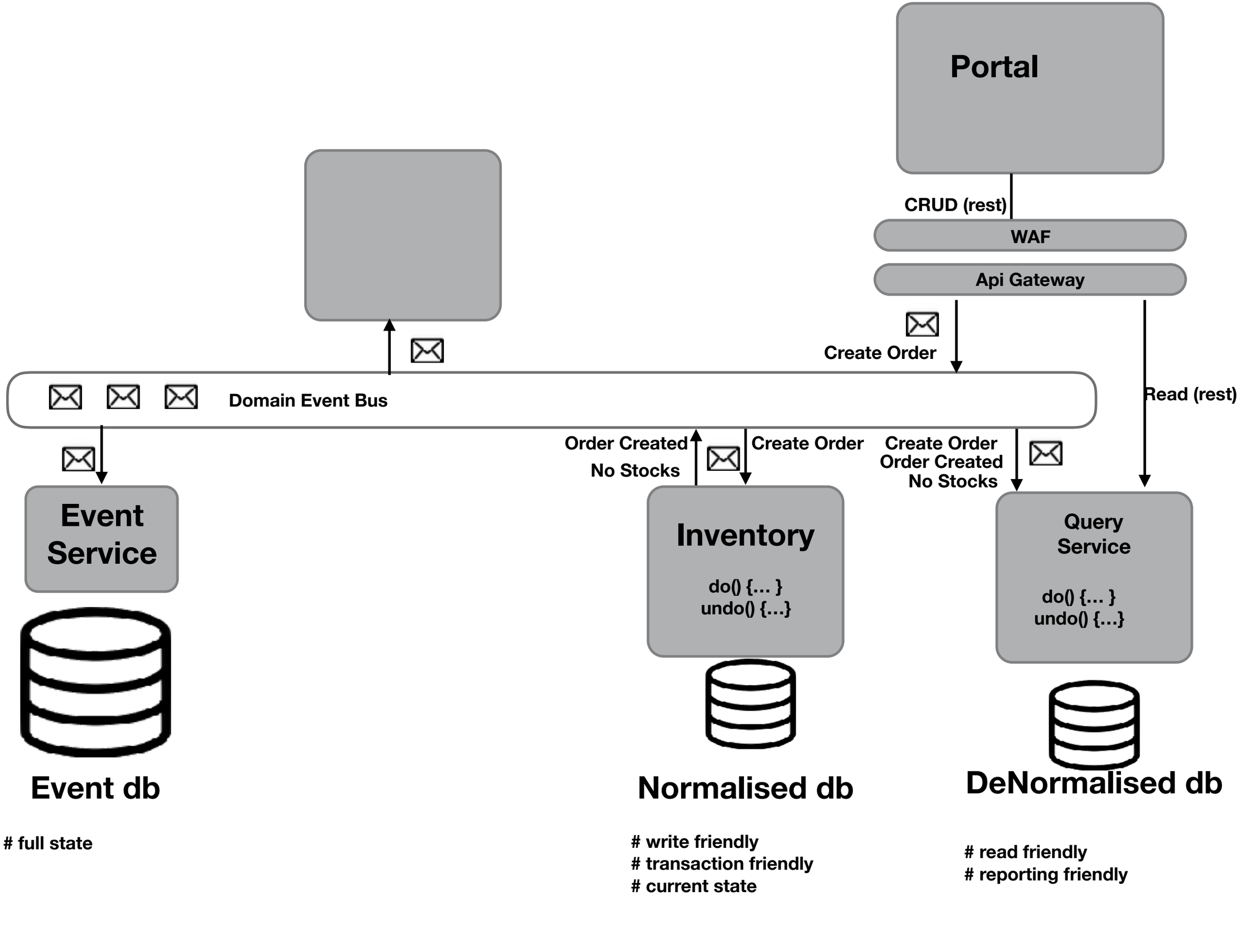
Normalised db

write friendly
transaction friendly

DeNormalised db

read friendly
reporting friendly


Portal
Accounting
Api Gateway
Order Created
Create Order
Read (rest)
Order Created
No Stocks
Create Order
Create Order
Order Created
No Stocks
Inventory
do() {… }
undo() {…}
Query
Service
do() {… }
undo() {…}
Normalised db
DeNormalised db
write friendly
transaction friendly
read friendly
reporting friendly


Portal
CRUD (rest)
WAF
Api Gateway
Create Order
Read (rest)
Domain Event Bus
Order Created
No Stocks
Create Order
Create Order
Order Created
No Stocks
Event Service
Inventory
do() {... }
undo() {...}
Query Service
do() {... }
undo() {...}
Event db
Normalised db
DeNormalised db
full state
write friendly
transaction friendly
current state
read friendly
reporting friendly

Portal

CRUD (rest)

WAF

Api Gateway

Create Order

Read (rest)

Stream Analytics

Order Created
No Stocks

Create Order

Create Order
Order Created
No Stocks

Event Service

Inventory

do() {... }
undo() {...}

Query Service

do() {... }
undo() {...}

Event db

full state

Archive

Data ware House

Snap Shots
(daily/ weekly / monthly)

Normalised db

write friendly
transaction friendly
current state

DeNormalised db

read friendly
reporting friendly

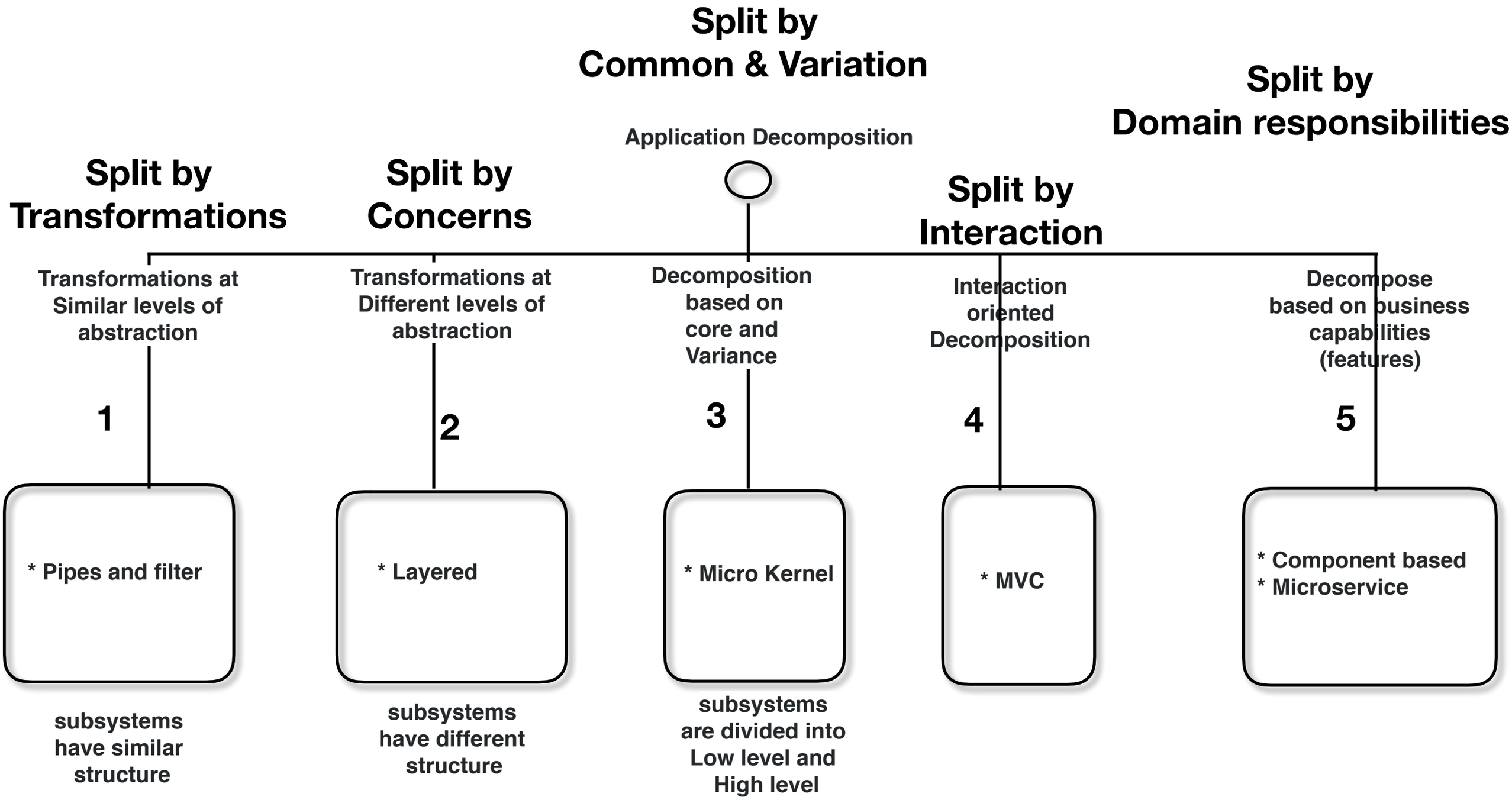
Big data / Batch Analytics

WareHouse Analytics

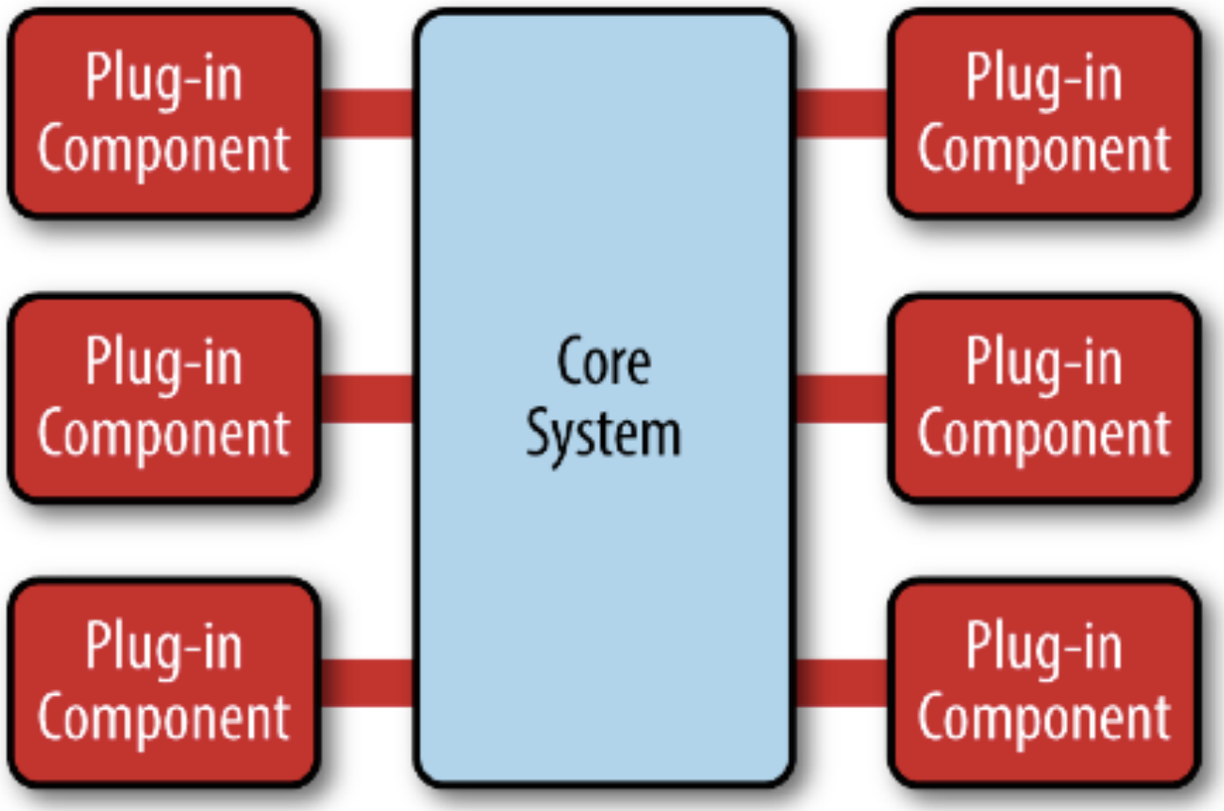
	Rabbitmq	Kafka	Redis
Scale	50K msg per second	1 million/ sec	1 million/ sec
Transient messages	Yes	No	Y
Persistent yes	Yes	Yes	Y
Protocol	AMQP		
One to one	Y	N	Y
One to many	Y	Y	Y
Advanced Message Queueing Protocol-based routing	Y	N	N
consumption pattern	Pull+Push	Pull	
Client Libraries	Py, java, php, .net, js,…	Py, java, php, .net, js,…	
Managed Service	yes, but not native in aws	Azure, aws, Confluent	Azure, aws
License	MPL	Apache	
Message Priority	Yes	No	
Monitoring	yes	yes	
Authentication	OAuth2, Std Auth	Kerberos, Oauth2, std Auth	
Message delay	microsecond	Millisecond	

Application Decomposition

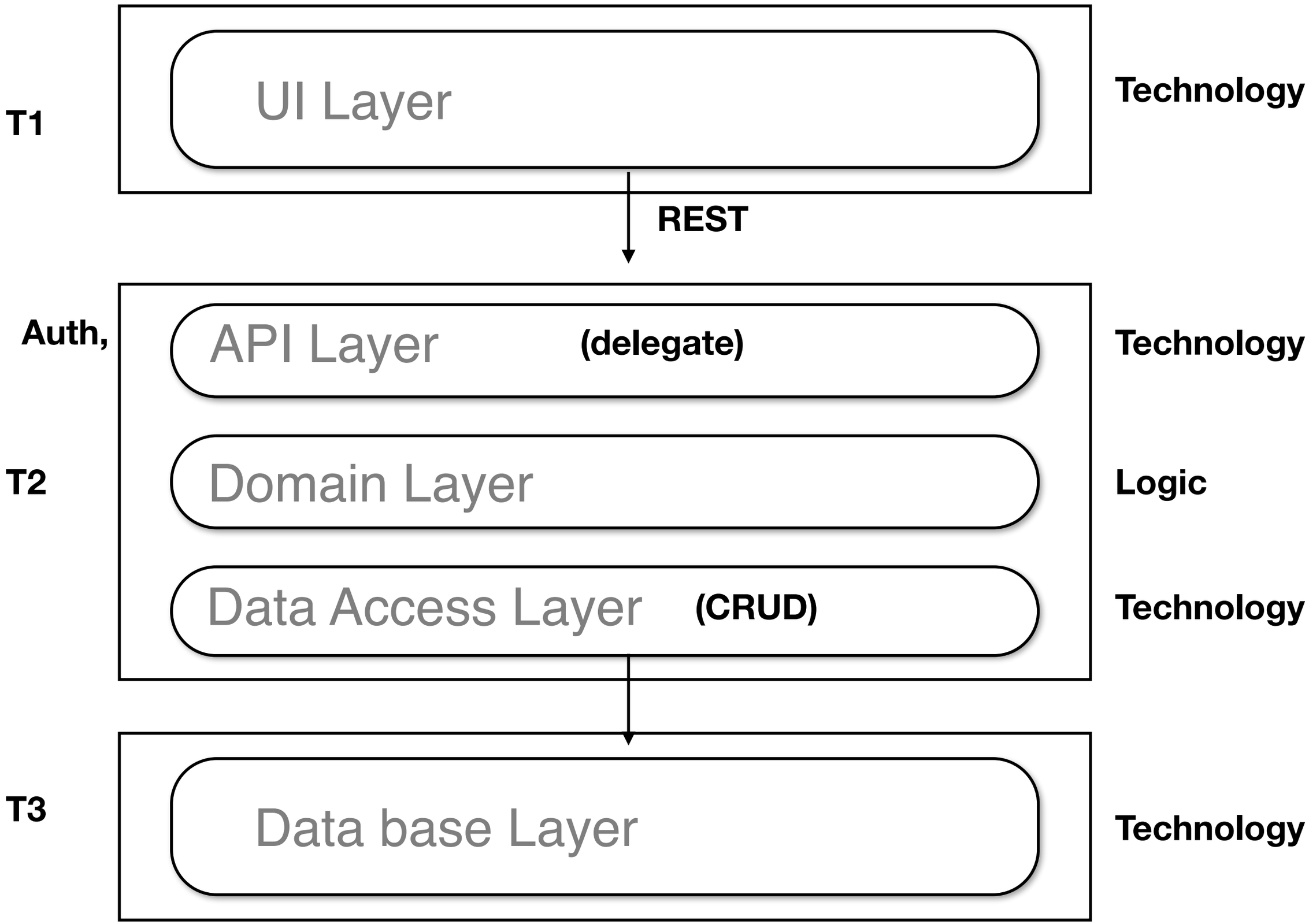
Decompose application



Microkernel Architecture



Logical View



Invoke logic()

Layer 1

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

Layer 2

f3(int,int)

f1(bool) f4()

f2(double)

Layer 3

f1() f4()

f2(int) f3() f5(int)

Layer 4

f1(int) f2()

Process Data()

Pipe

Filter 1

f1(int)

Filter 2

f2(int)

* Can swap filters

Filter 3

f3(int)

Filter 4

f4(int)

Data

Break up the User Interface layout and Logic into separate components. This way, it's much easier to manage and make changes to either side without them interfering with each other.

Eclipse IDE. Downloading the basic Eclipse product provides you little more than a fancy editor. However, once you start adding plug-ins, it becomes a highly customizable and useful product.

An application is required to perform a variety of tasks of varying complexity on the information that it processes. The processing tasks performed by each module, or the deployment requirements for each task, could change as business requirements are updated. Also, additional processing might be required in the future, or the order in which the tasks performed by the processing could change. A solution is required that addresses these issues, and increases the possibilities for code reuse.

Networking engineering is a complicated task, which involves software, firmware, chip level engineering, hardware, and electric pulses. To ease network engineering, the whole networking concept is decomposed into more manageable parts.

claims processing system contains the basic business logic required by the insurance company to process a claim, except without any custom processing. most insurance claims applications leverage large and complex rules engines to handle much of this complexity.

However, these rules engines can grow into a complex big ball of mud where changing one rule impacts other rules, or making a simple rule change requires an army of analysts, developers, and testers. State-specific rules and processing is separate from the claims system and can be added, removed, and changed with little or no effect on the rest of the system or other modules.

1

Application
Exe

Compile time
monolithic

2

Application
Exe

Lib

Lib

Link time
monolithic

3

Process

Application
Exe

Dll/so/jar

Dll/so/jar

Dll/so/jar

In process
comp

Runtime
monolithic

4

Process

Exe

Dll/so/jar

Distributed
comp

Pipes
Sockets
http
Roc
…

Process

Exe

Dll/so/jar

Distributed
comp

Distributed Application

5

Process

Exe

Dll/so/jar

Application1

Process

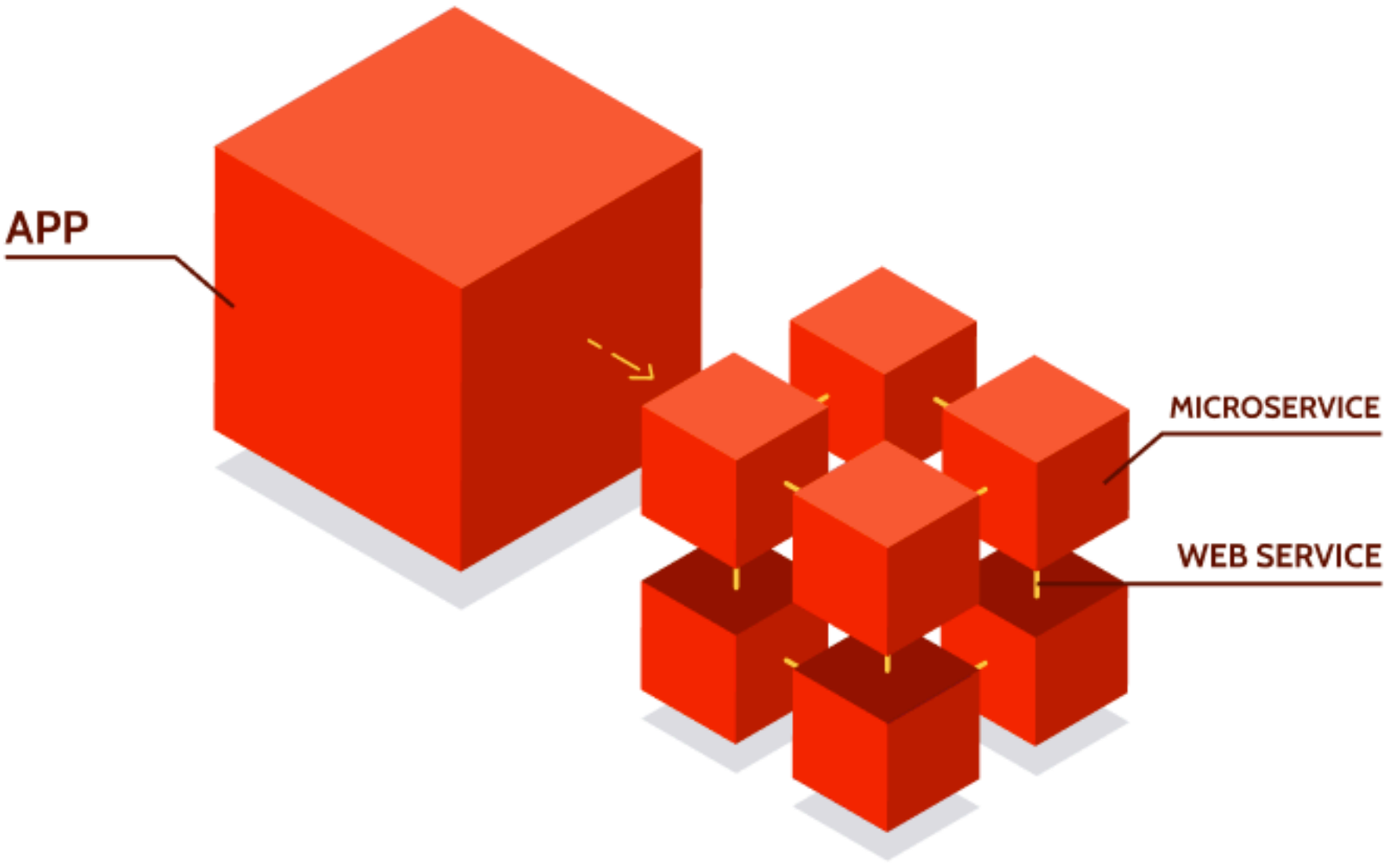
Exe

Dll/so/jar

Application2

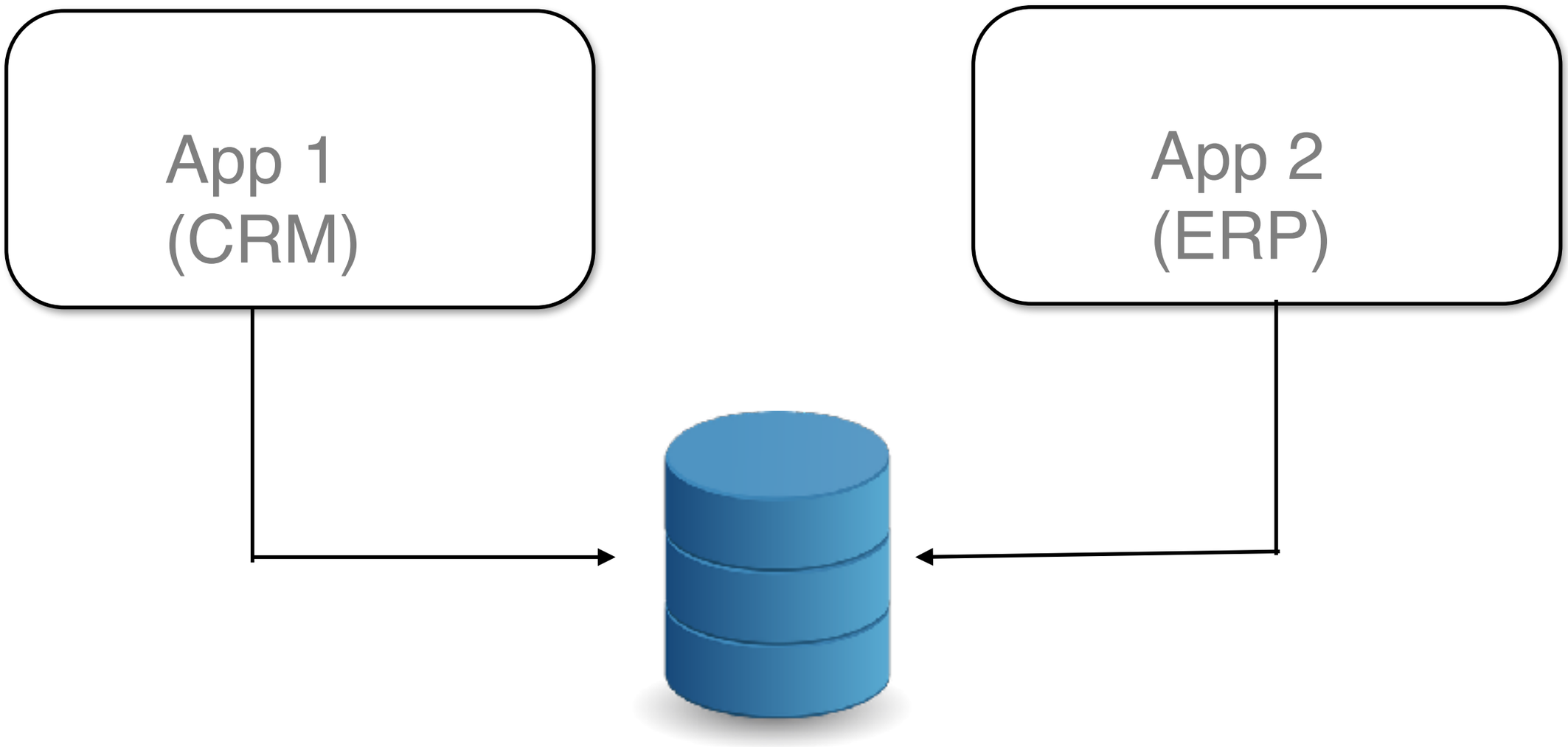
Micro Application

Microservice

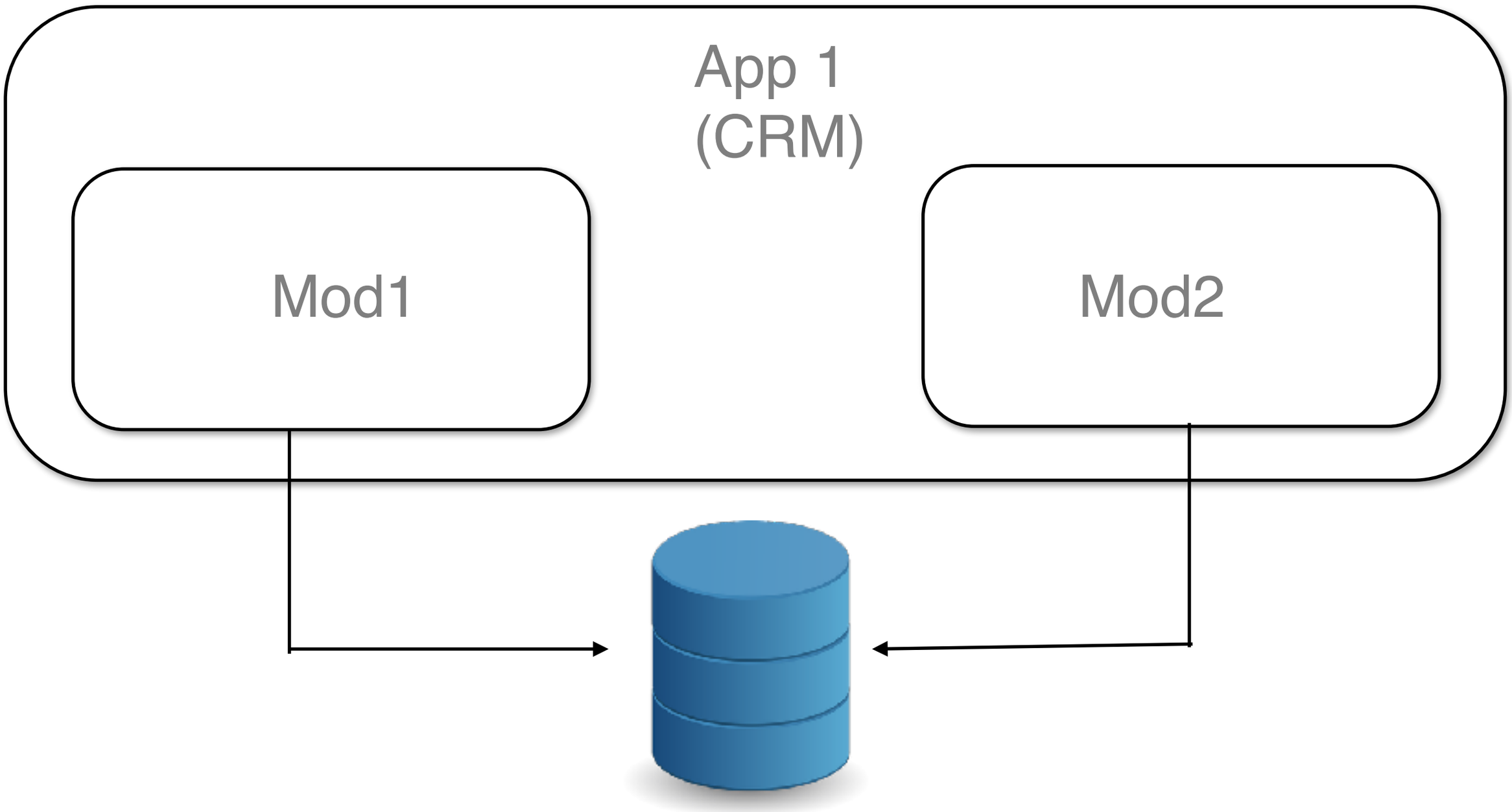


	2 modules	2 Applications	W	Score
Database / Storage	Shared	Not Shared	2	
Infra (Hosting)	Shared	Not Shared	3	
Sorce Control	Shared	Not Shared	2	
CI/CD (Build Server)	Shared	Not Shared	3	
Fun Requirements	Shared	Its Own	1	
SCRUM Team / Sprint	Shared	Its own	1	
Test Cases	Shared	Its own	1	
Architecture	Shared	Its own	1	
Technology Stack / Fwks	Shared	Its own	1	

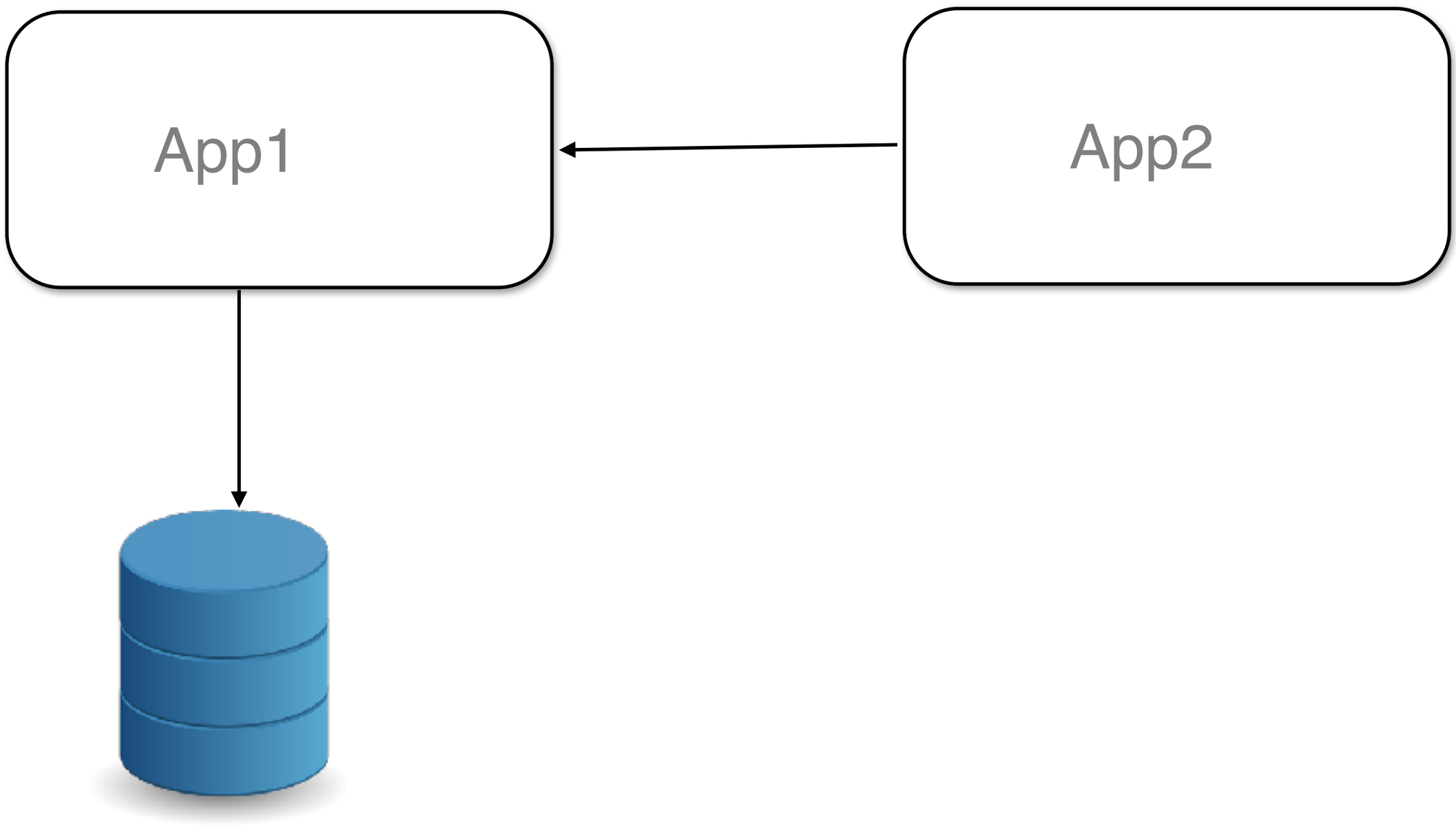
1. Database



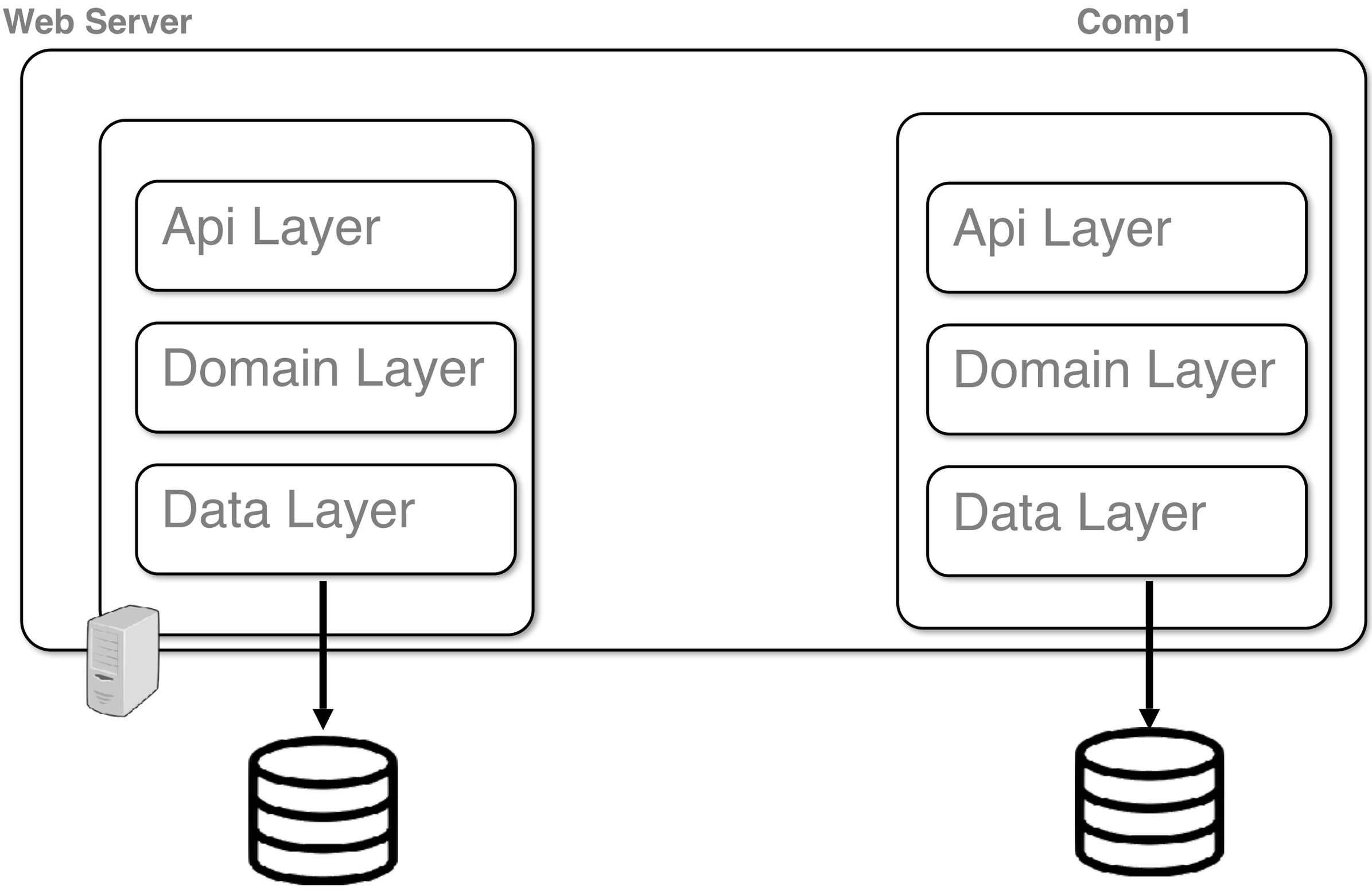
1. Database



1. Database



2. Infrastructure



1

UI App - SPA

Web Server

API Layer

Domain Layer
<<feature1>>

Data Layer

Angular (SPA)

2

Browser

Web Server

UI Layer -
Server pages

Domain Layer
<<feature2>>

Data Layer

JSP, Servlet, JSF (Server pages)

3

Desktop

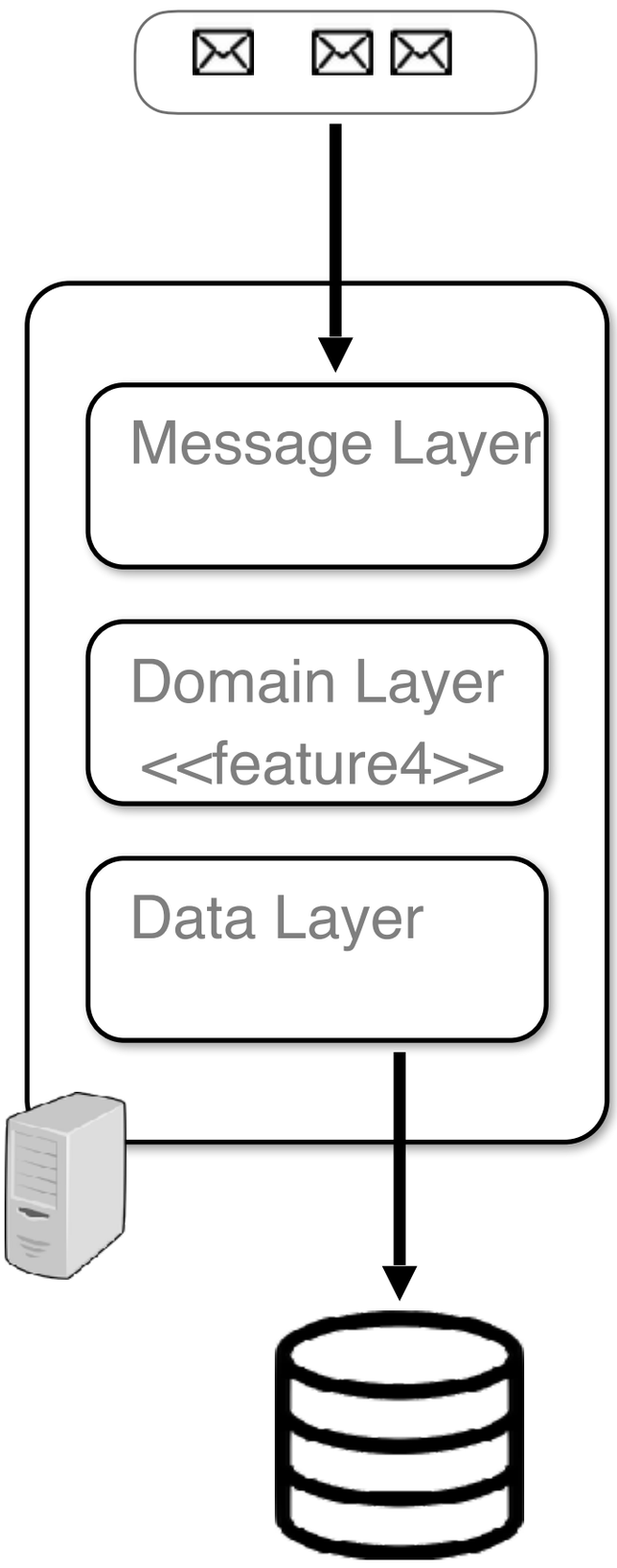
Native UI Layer

Domain Layer
<<feature3>>

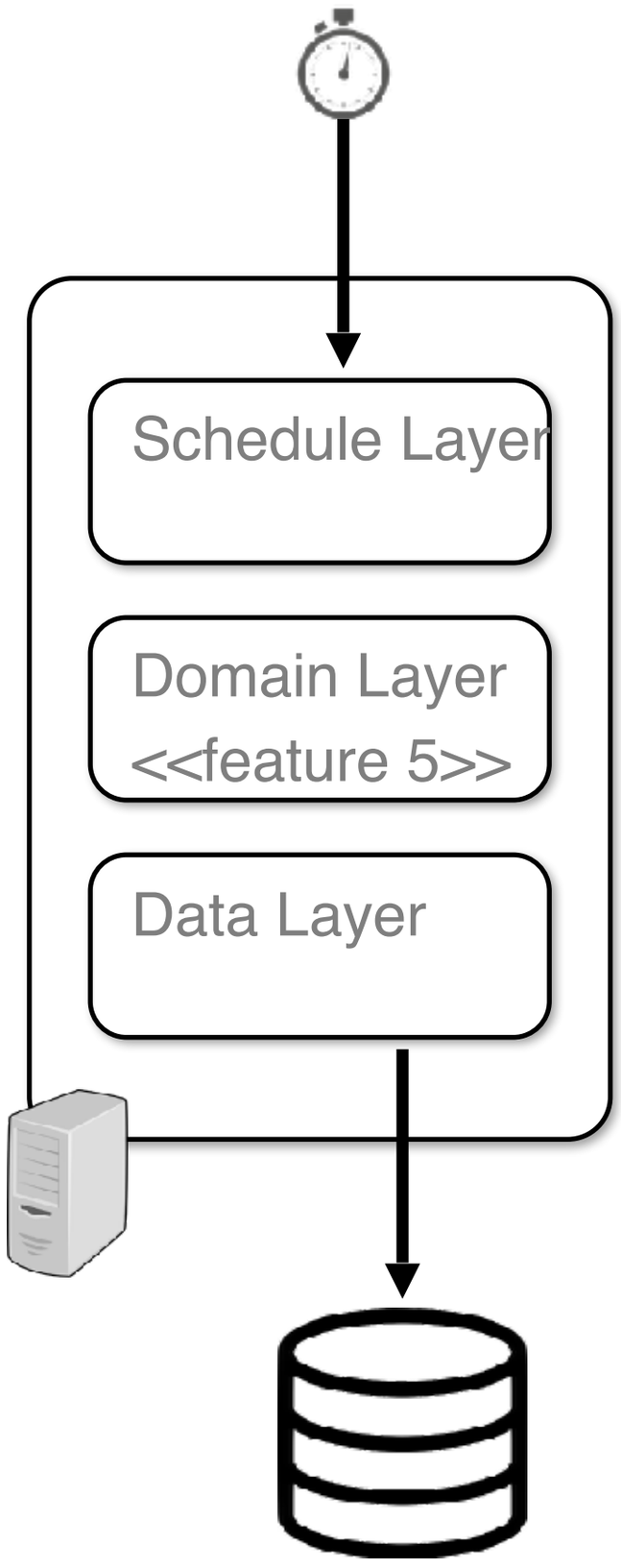
Data Layer

Swing, Android, Ios (Native)

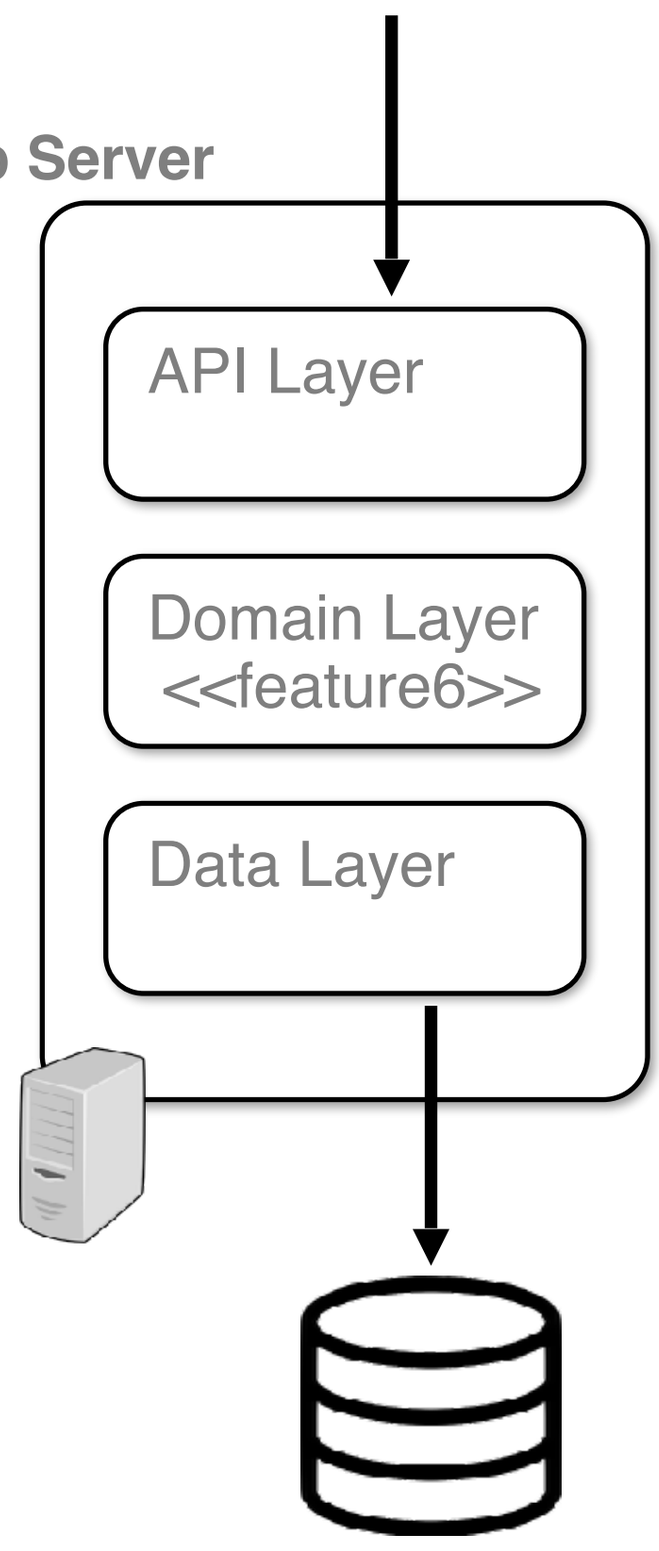
1


Message Layer
Domain Layer
<<feature4>>
Data Layer

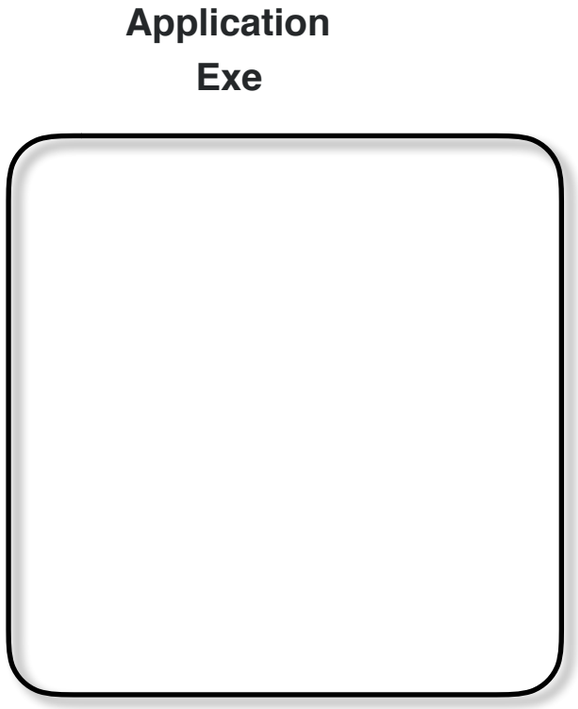
2


Schedule Layer
Domain Layer
<<feature 5>>
Data Layer

3

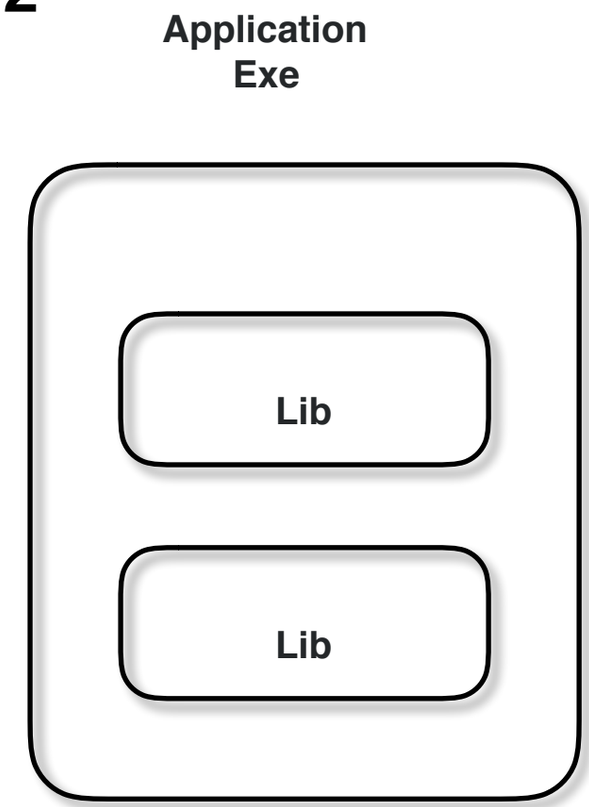

Web Server
API Layer
Domain Layer
<<feature6>>
Data Layer

1



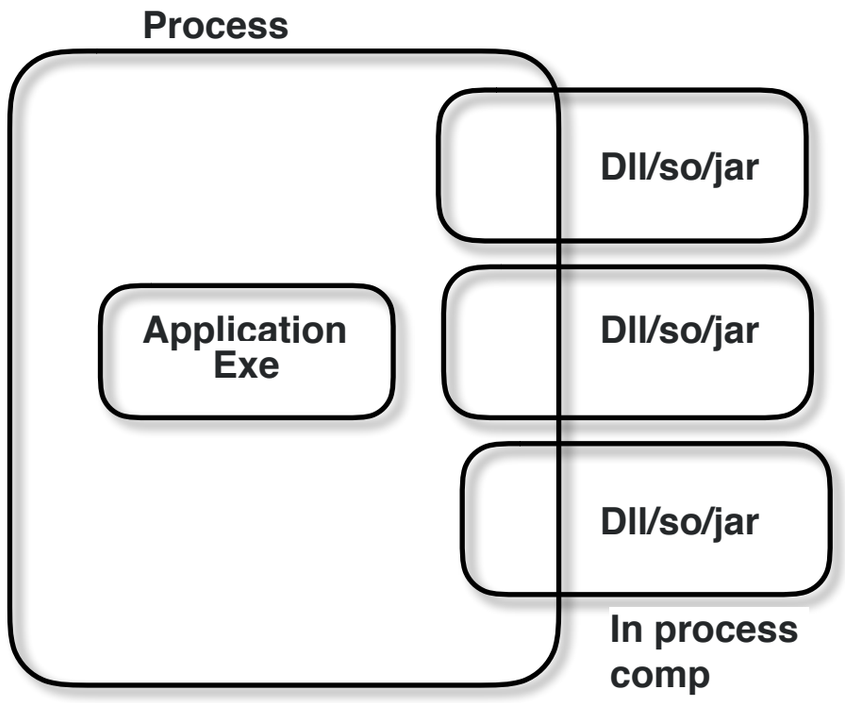
Compile time
monolithic

2



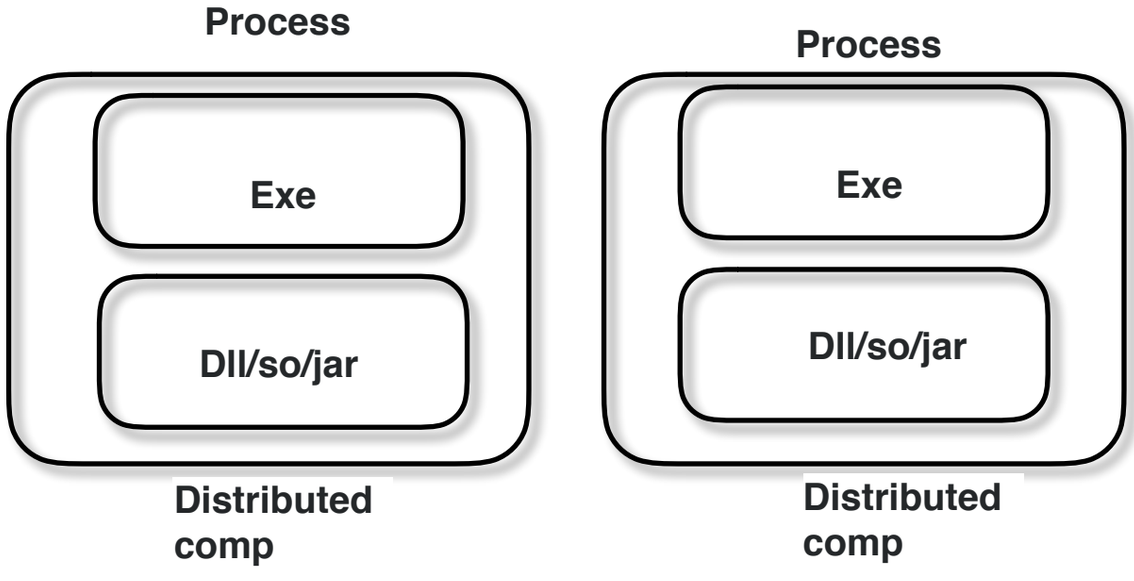
Link time
monolithic

3



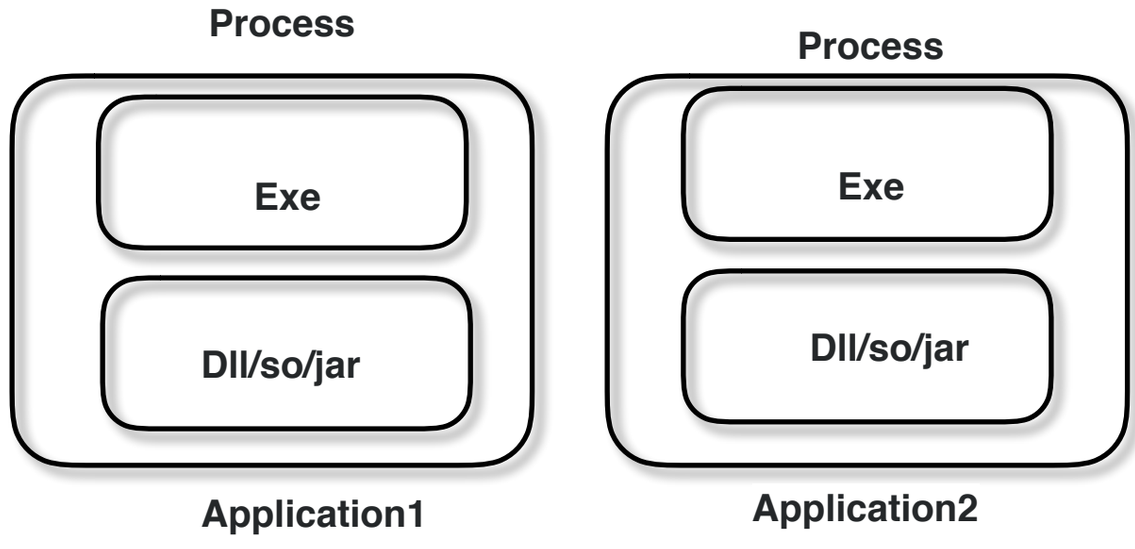
Runtime
monolithic

4



Distributed Application

5

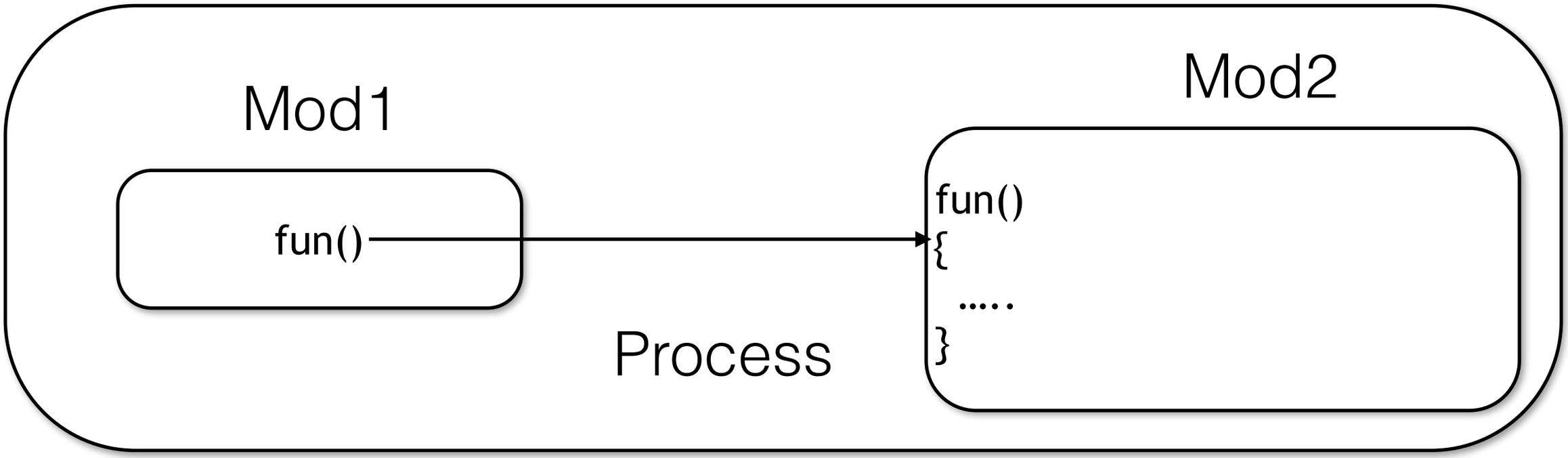


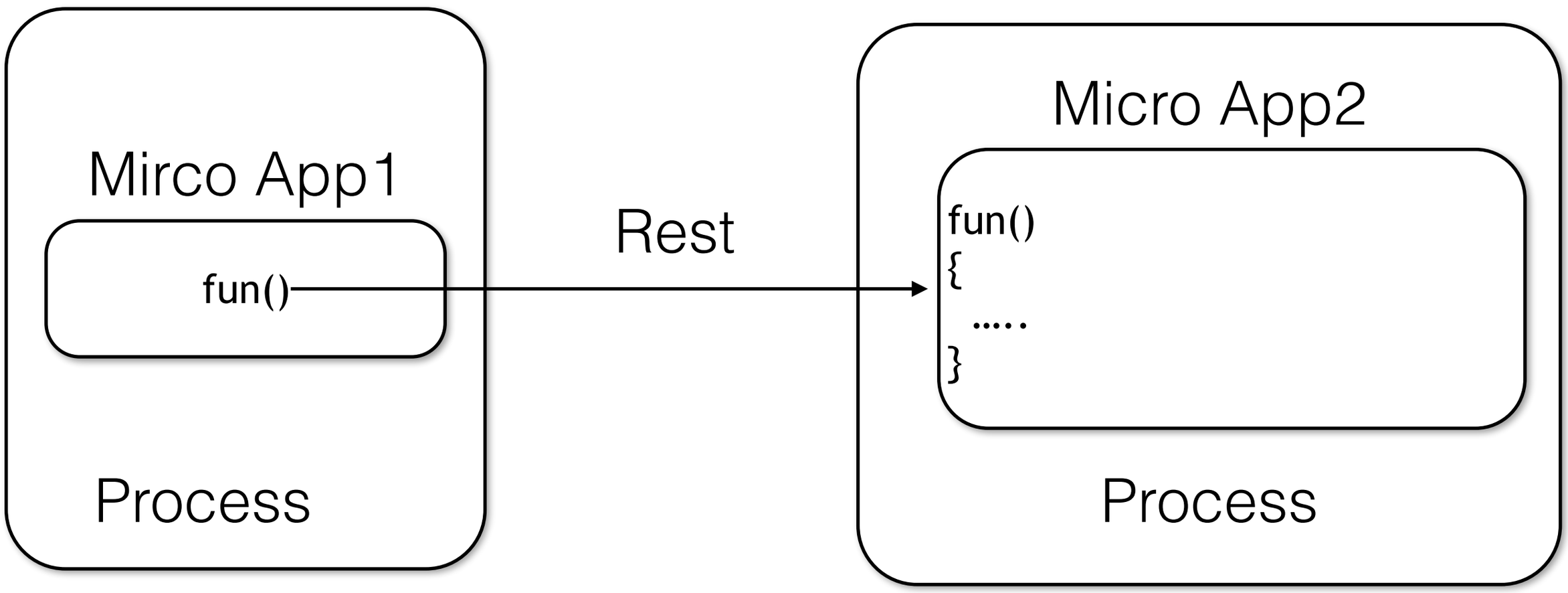
Micro Application

	Pros/ Cons	Solution
Development time	**- - (3 times)**	
Learning Curve	**- -**	
Resource Performance (CPU, Memory, I/O)	**- - (I/O)**	Grpc, proto buf, …
Db Transaction Management	**- - (multiple)**	
Views / Report / Dash board/ join	**- - (multiple)**	
Infra Cost	**- -**	Containers
Deployment effort	**- -**	Automation
Debugging, Error Handling,	**- -**	
Integration Test	**- -**	
debug/ error Log Mgmt	**- -**	Centralize - ELK, EFK, Splunk, ..
Config Mgmt	**- -**	Centralize - Consul, …
Authentication (who are you)	**- -**	Centralize - OAuth2, OpenID connect
Authorization (what can you do)	**- -**	Centralize
Audit Log mgmt (what did you do)	**- -**	Centralize
Monitoring / Alerting	**- -**	Centralize - Prometheus, App Insight
Data Security and Privacy (In Rest, In Transit)	**- -**	
Build Pipeline (CI)	**- -**	Automation
Agile Architecture (Agility to change)	**+ + +**	
Feature Shipping (Agility to ship)/ CD	**+ + +**	
Scalability (volume - request, data, compute)	**+**	
Resilence	**+**	
Availability	**+**	
Ability to do Polygot	**+**	

	CPU Cycles
a + b	3 cpu
Fun call	10 cpu cycles
Create Thread	200,000 cpu cycles
Destroy thread	100,000 cpu cycles
Write file	10,00,000
API Call	20,00,000
Db call	45,00,000

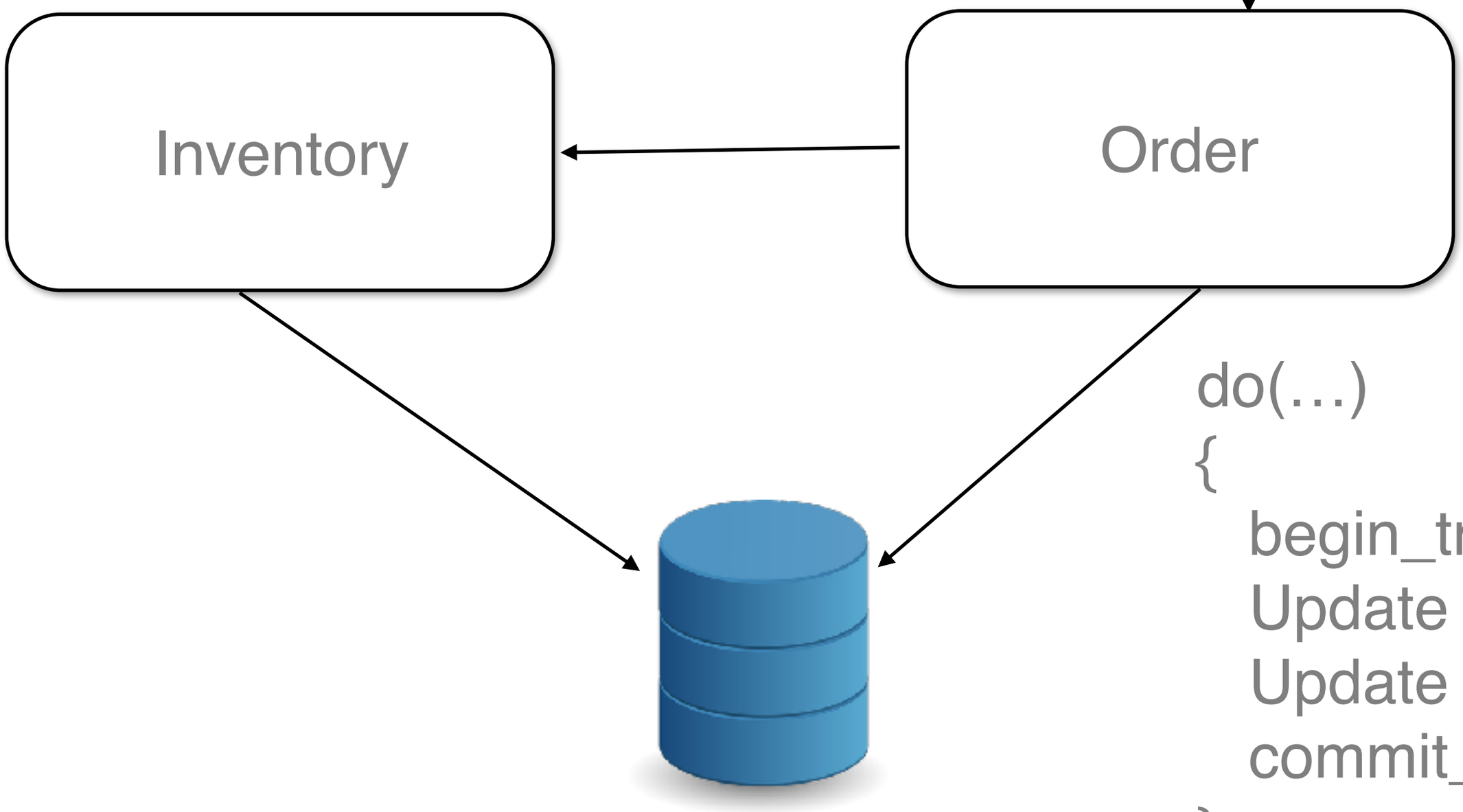
4. Development time



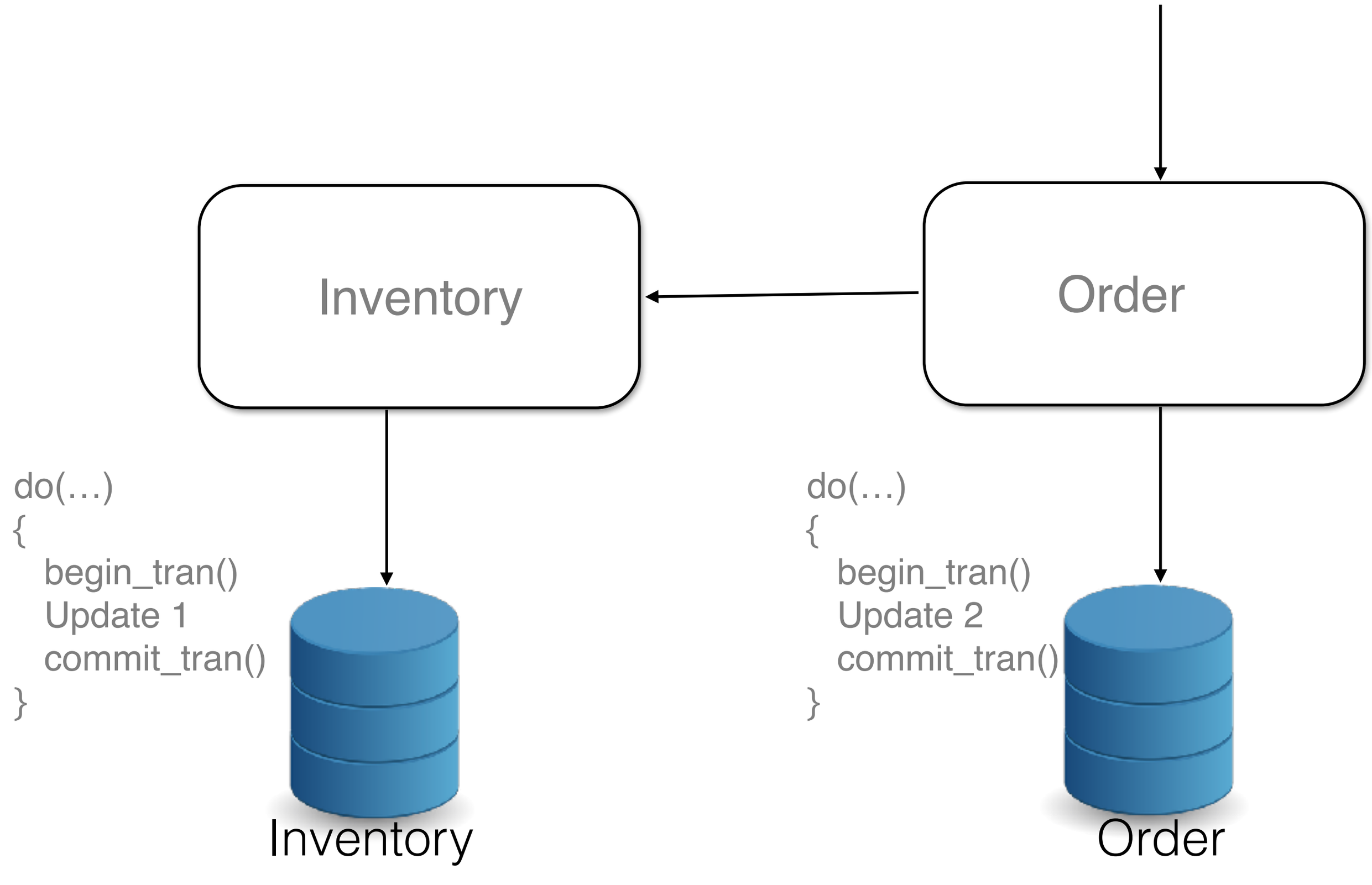


A + B	3
Fun call	10 - 20
Create Thread	2,00,000
Write File (I/O)	10,00,000
Network call	10,00,000 - 50,00,000

2. Db transaction

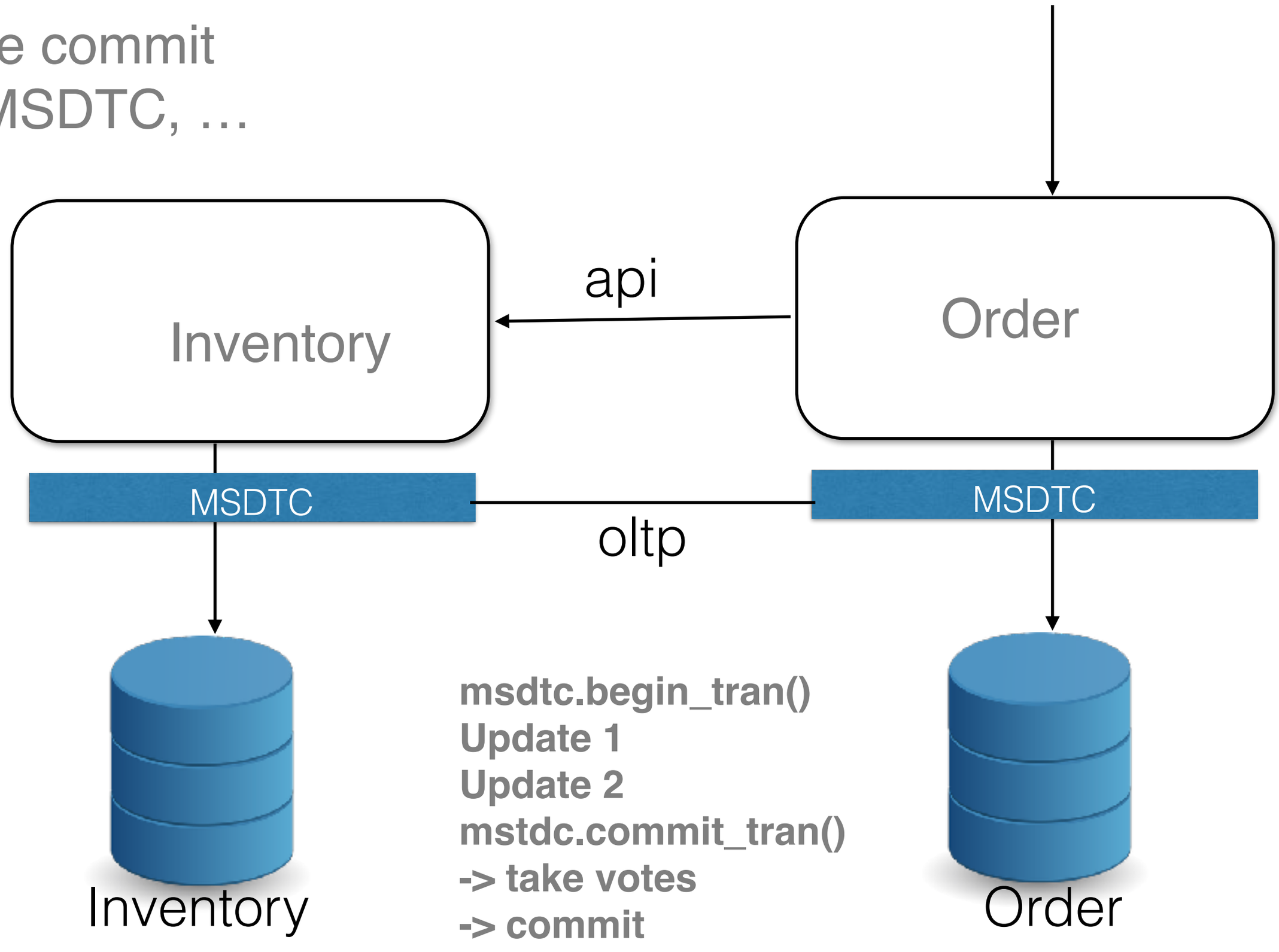


2. Db transaction



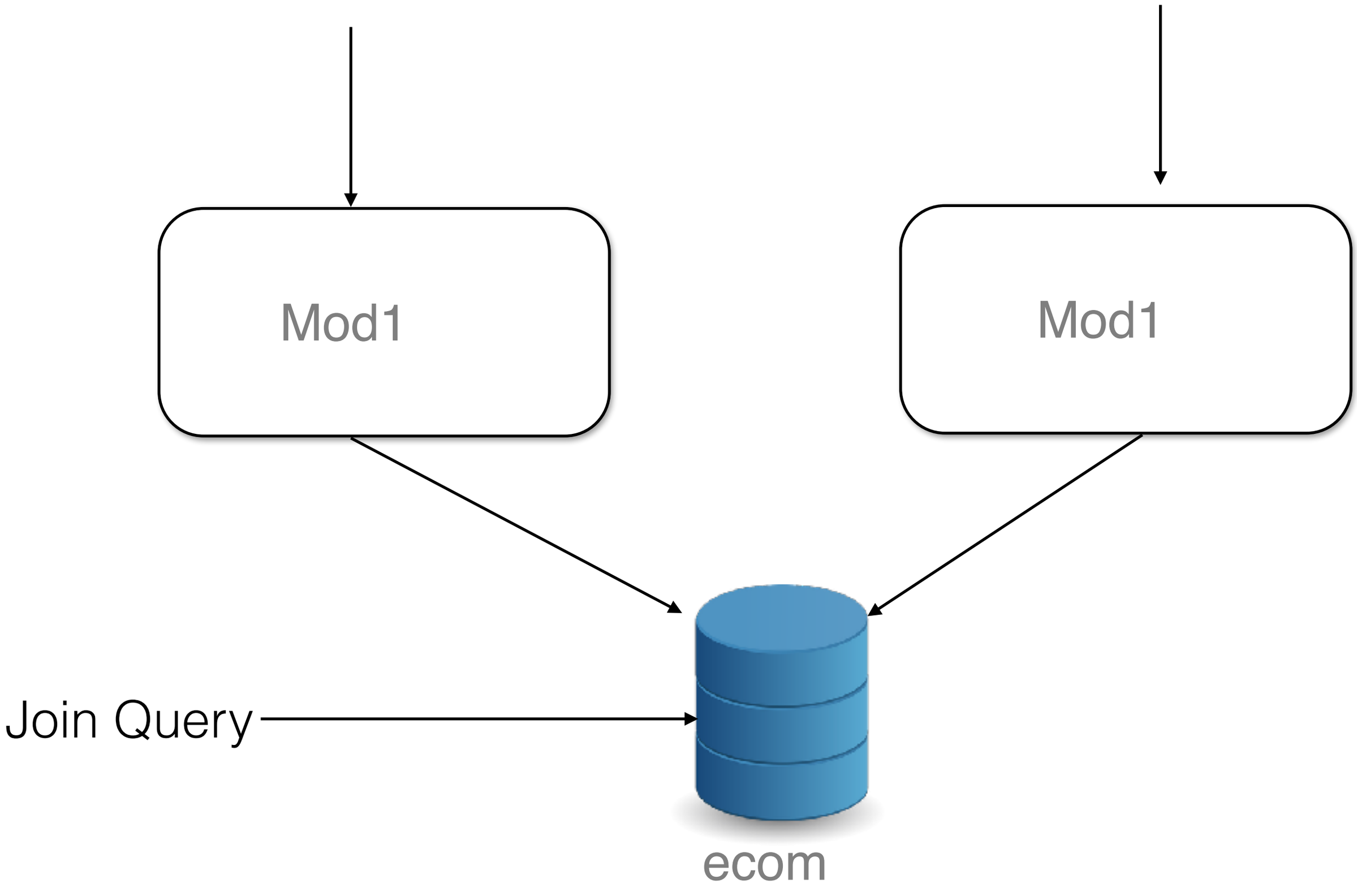
2. Db transaction

2 phase commit
JTX , MSDTC, …

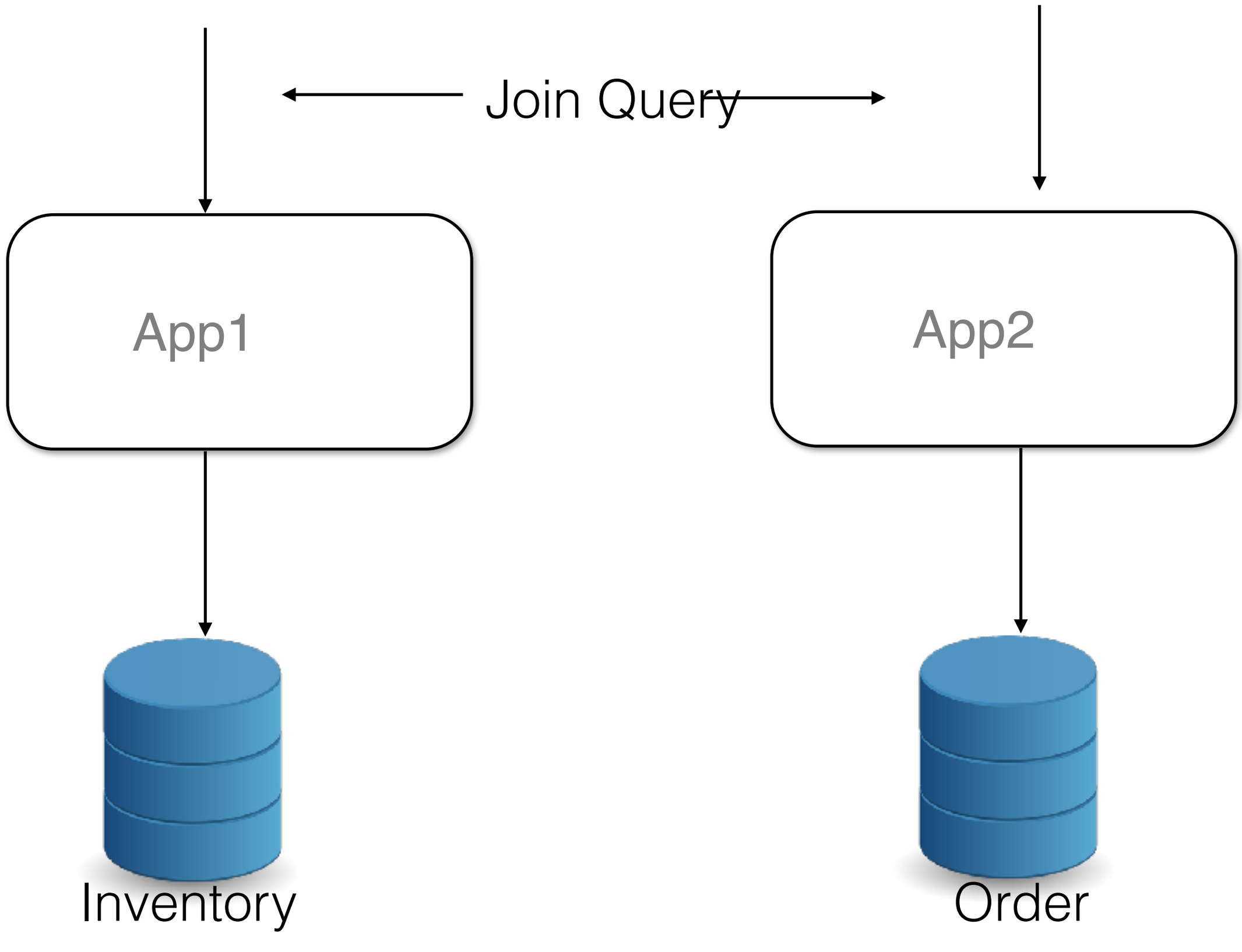


msdtc.begin_tran()
Update 1
Update 2
mstdc.commit_tran()
-> take votes
-> commit

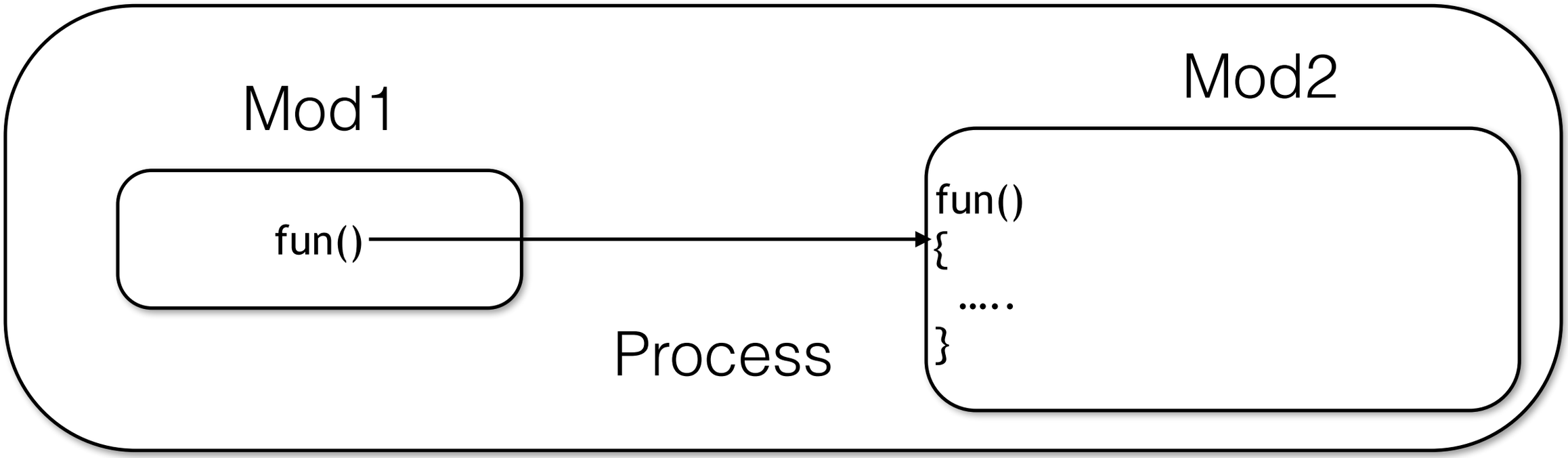
3. Db query

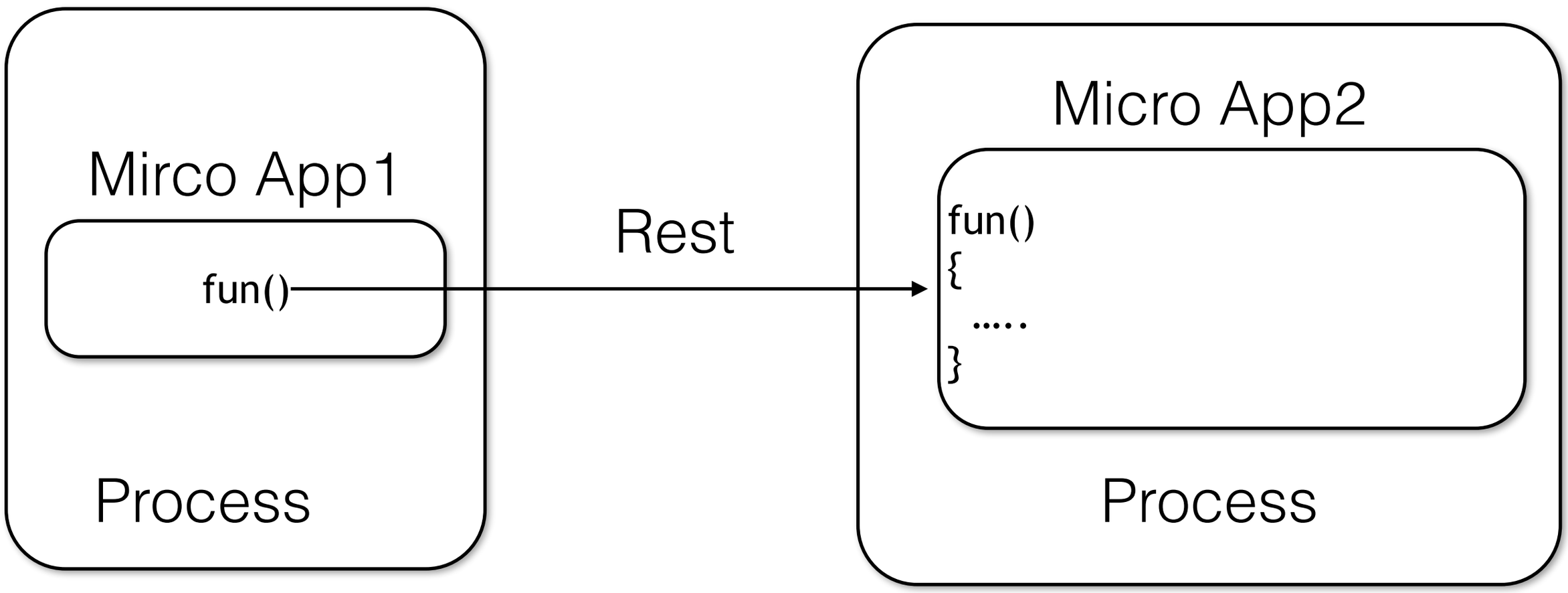


3. Query

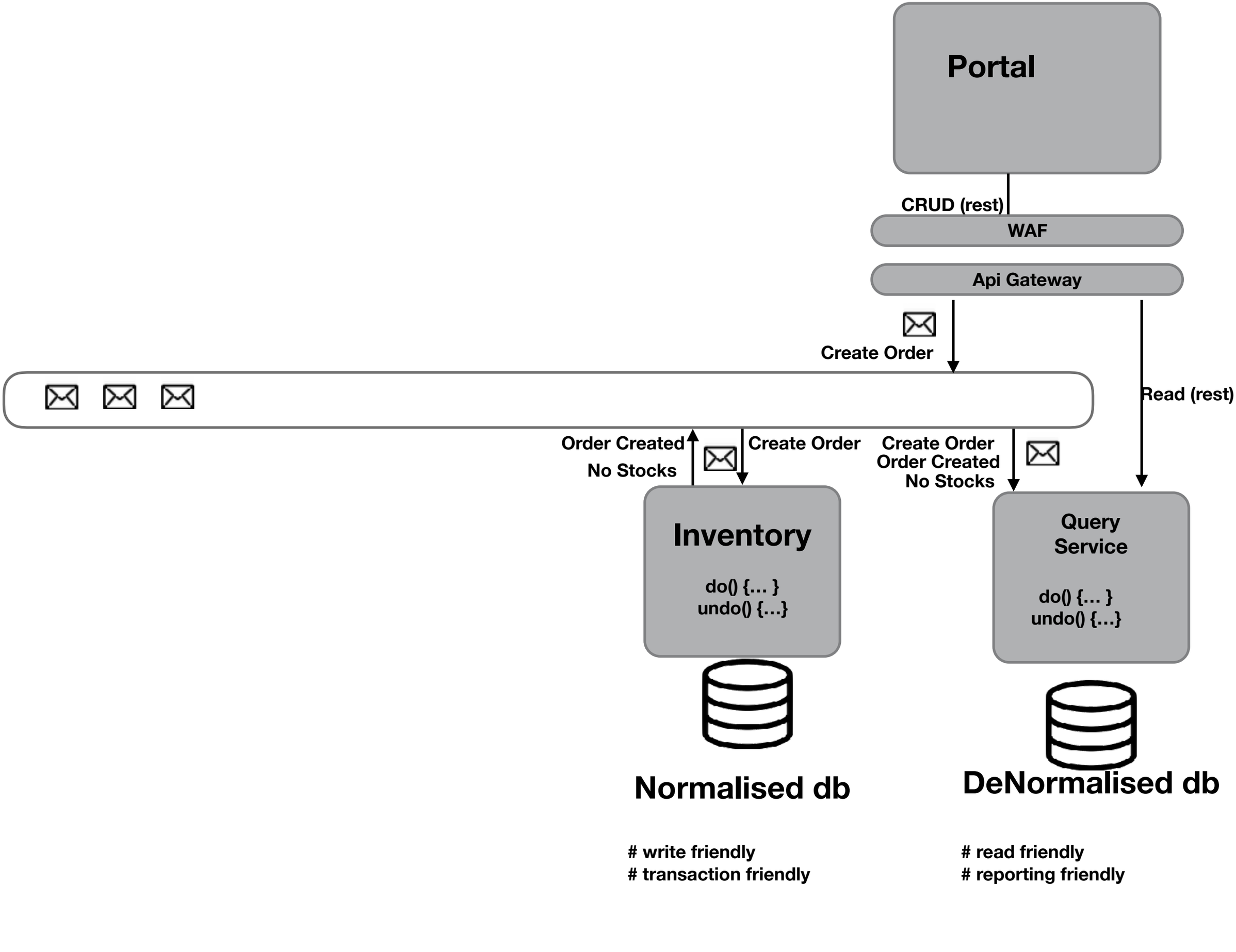


4. Development time

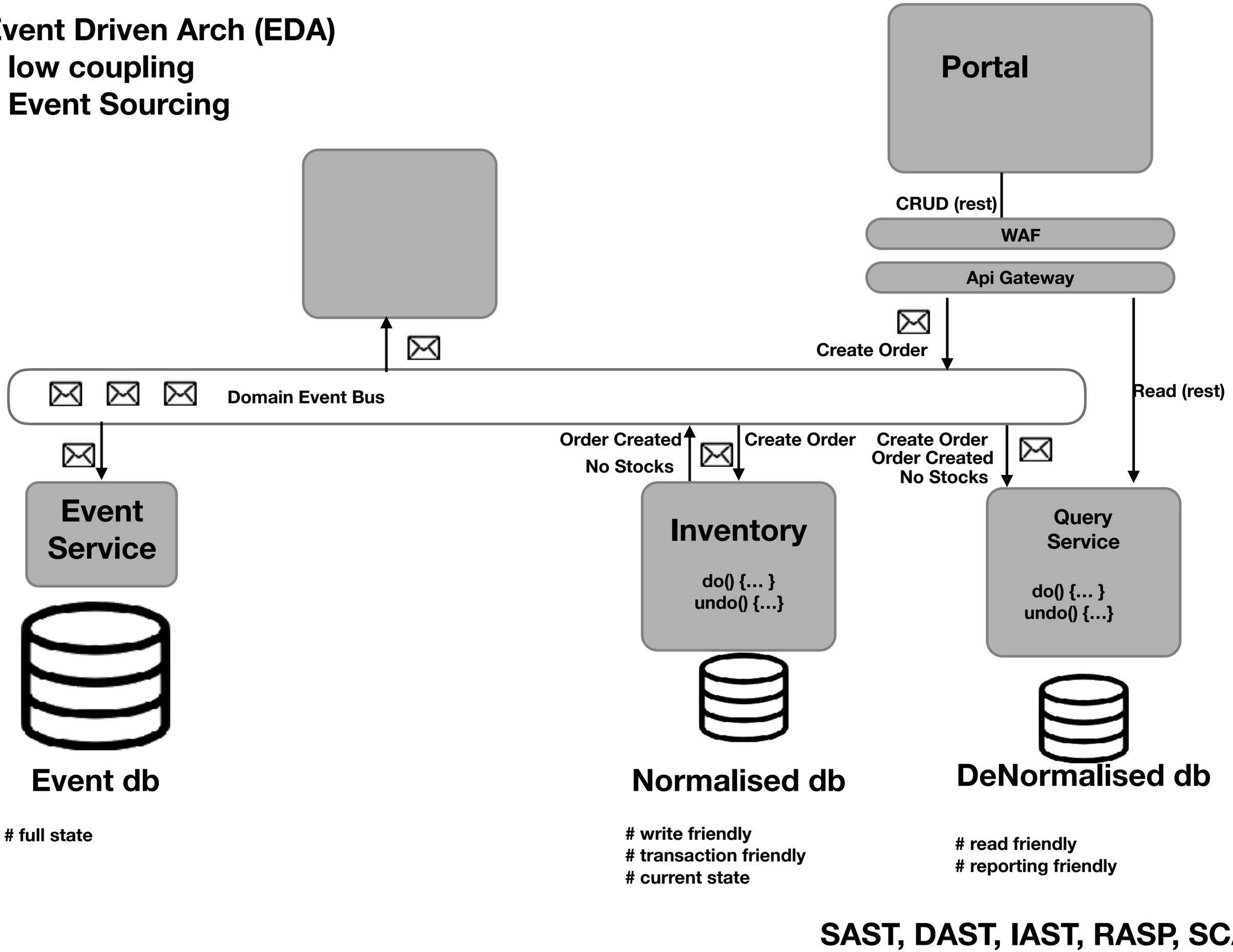




Analytics App


Portal
CRUD (rest)
WAF
Api Gateway
Create Order
Read (rest)
Order Created
No Stocks
Create Order
Create Order
Order Created
No Stocks
Inventory
do() {… }
undo() {…}
Query
Service
do() {… }
undo() {…}
Normalised db
DeNormalised db
write friendly
transaction friendly
read friendly
reporting friendly

Event Driven Arch (EDA)
low coupling
Event Sourcing



full state

write friendly
transaction friendly
current state

read friendly
reporting friendly

SAST, DAST, IAST, RASP, SCA

Portal

CRUD (rest)

WAF

Api Gateway

Create Order

Read (rest)

Stream Analytics

Order Created
No Stocks

Create Order

Create Order
Order Created
No Stocks

Event Service

Inventory

do() {... }
undo() {...}

Query Service

do() {... }
undo() {...}

Event db

full state

Archive

Data ware House

Snap Shots
(daily/ weekly / monthly)

Normalised db

write friendly
transaction friendly
current state

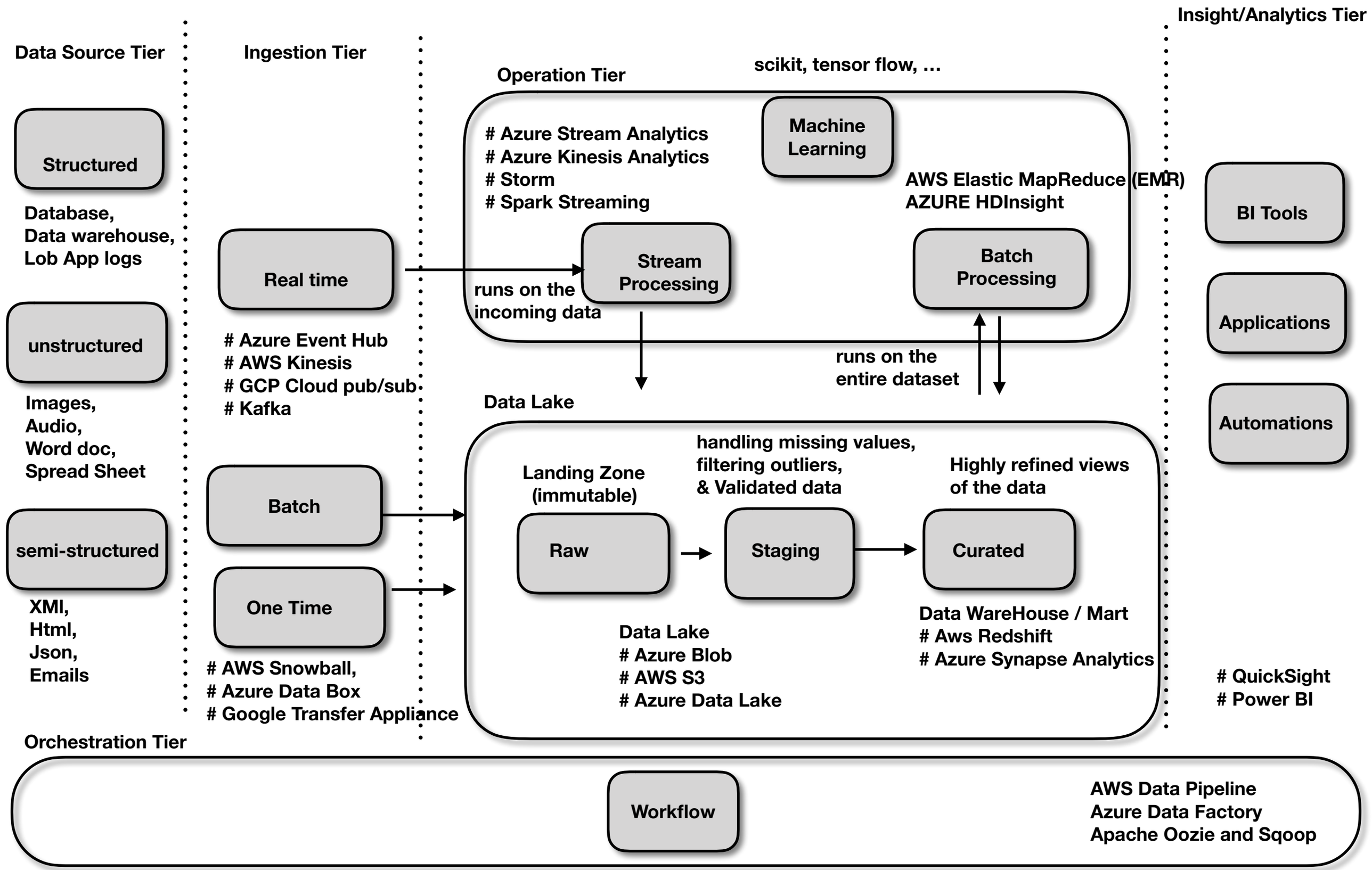
DeNormalised db

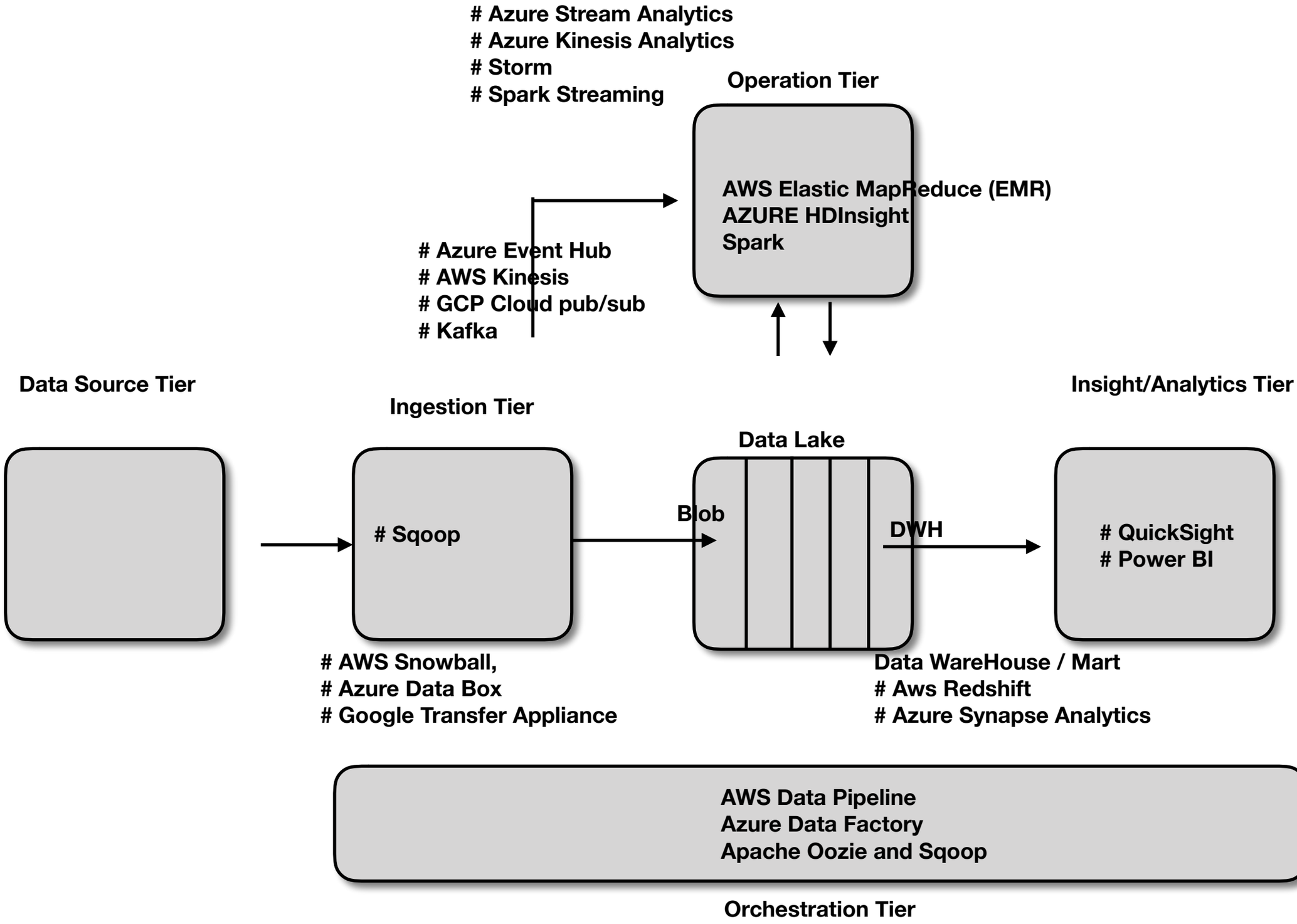
read friendly
reporting friendly

Big data / Batch Analytics

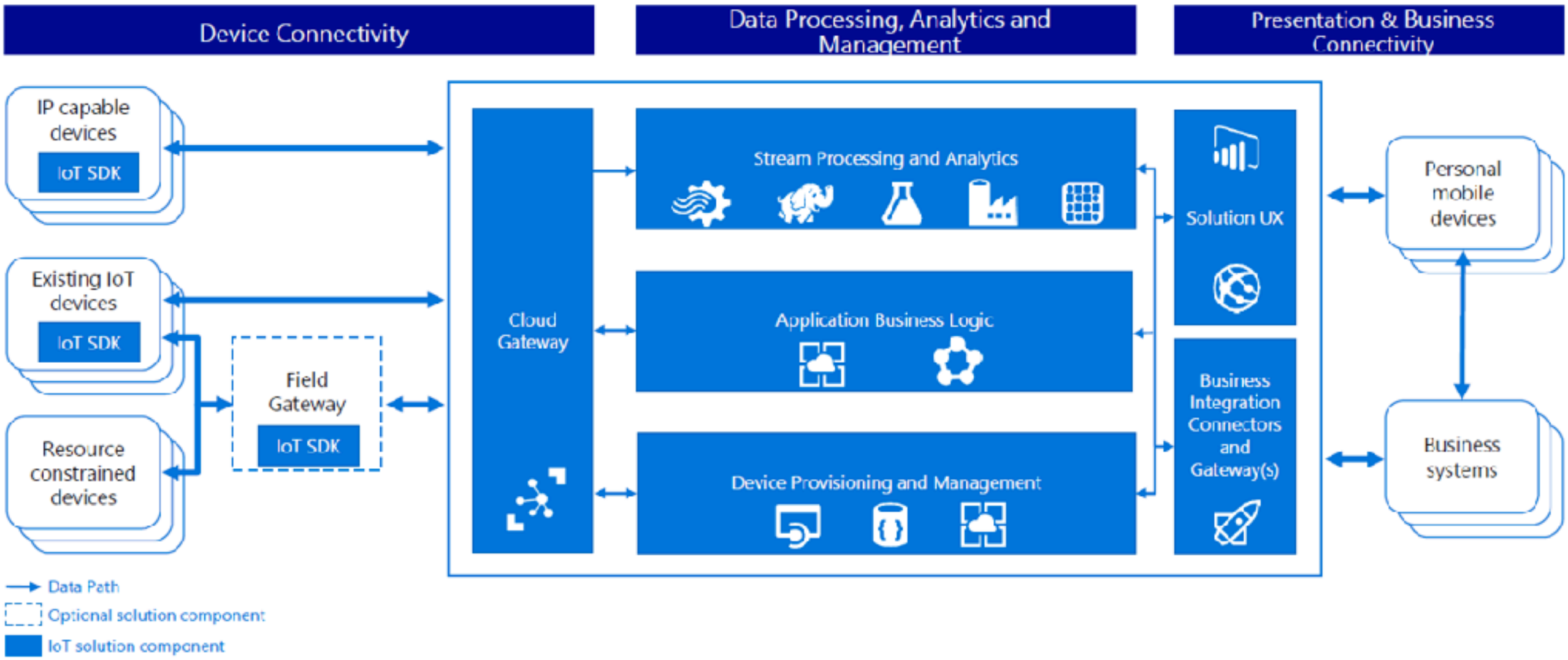
WareHouse Analytics

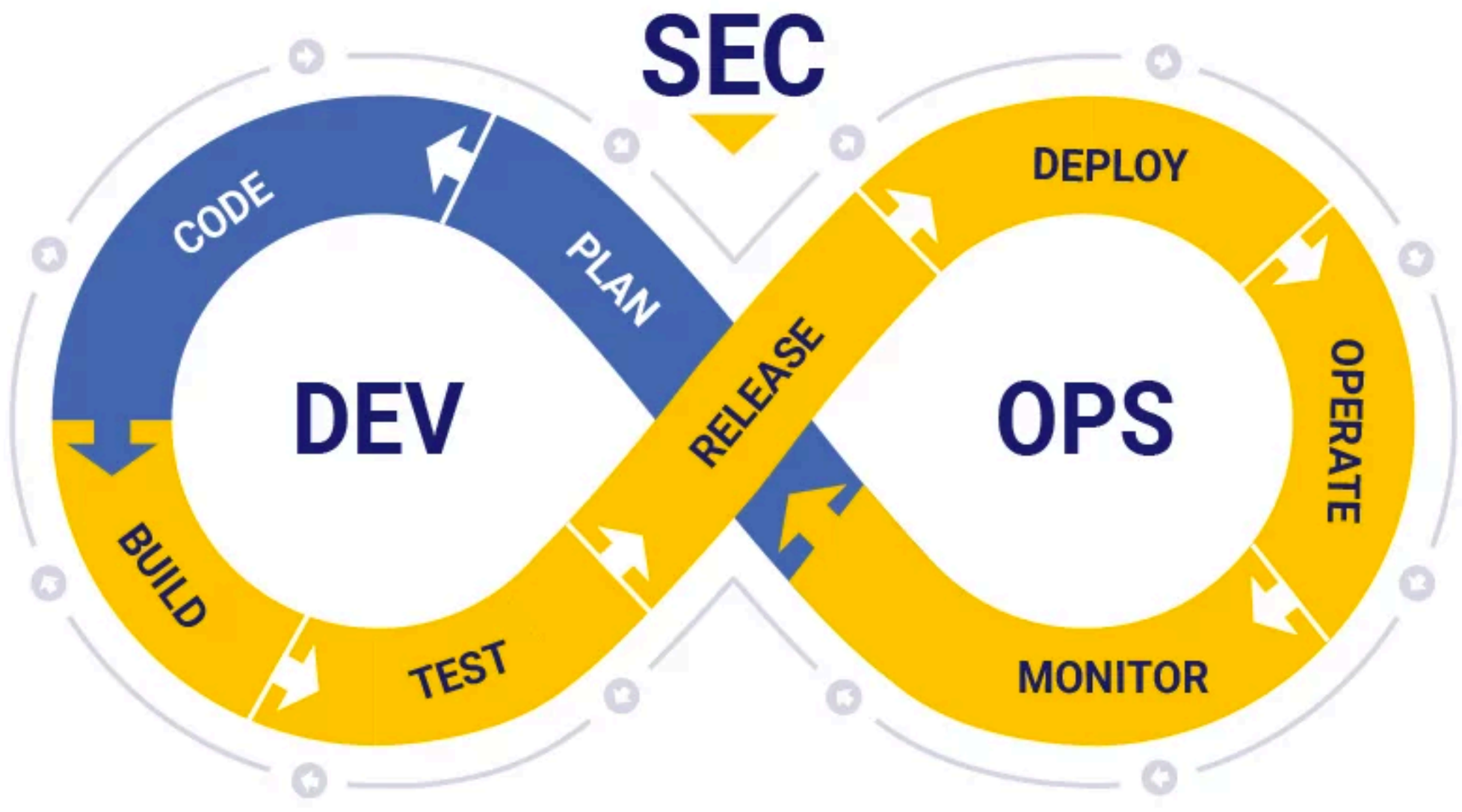
Reference Architecture
Analytical Application




Azure Stream Analytics
Azure Kinesis Analytics
Storm
Spark Streaming
Operation Tier
AWS Elastic MapReduce (EMR)
AZURE HDInsight
Spark
Azure Event Hub
AWS Kinesis
GCP Cloud pub/sub
Kafka
Data Source Tier
Ingestion Tier
Data Lake
Insight/Analytics Tier
Sqoop
Blob
DWH
QuickSight
Power BI
AWS Snowball,
Azure Data Box
Google Transfer Appliance
Data WareHouse / Mart
Aws Redshift
Azure Synapse Analytics
AWS Data Pipeline
Azure Data Factory
Apache Oozie and Sqoop
Orchestration Tier

Reference Architecture
IOT Application




SEC
CODE
PLAN
BUILD
TEST
RELEASE
DEPLOY
OPERATE
MONITOR
DEV
OPS

Code

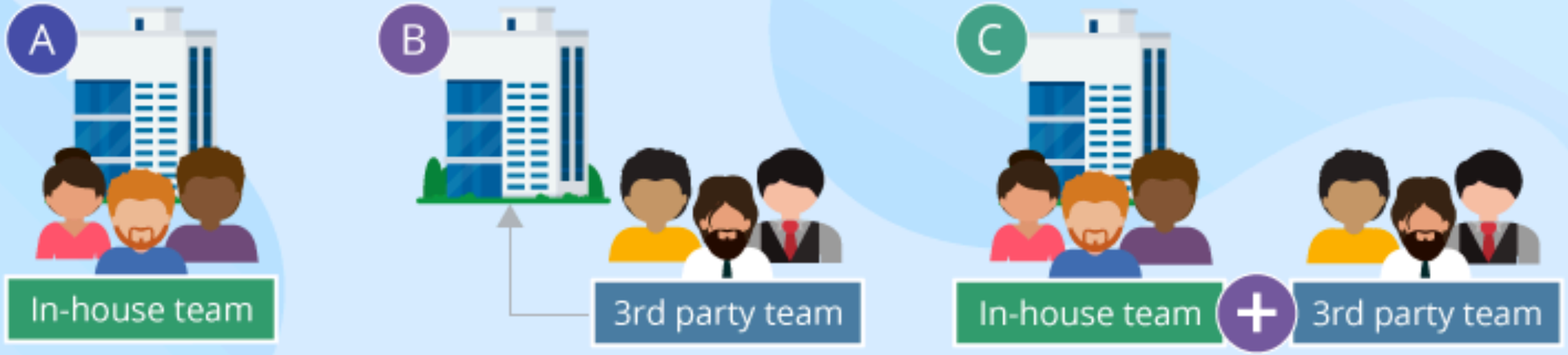
Developers

QA

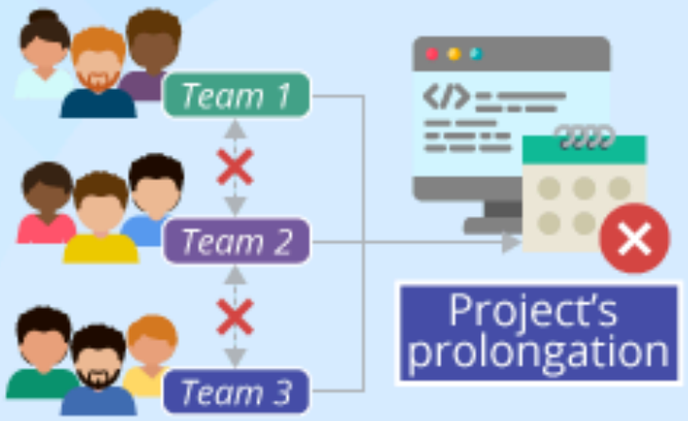
Deployment/
Operations

pre-DevOps

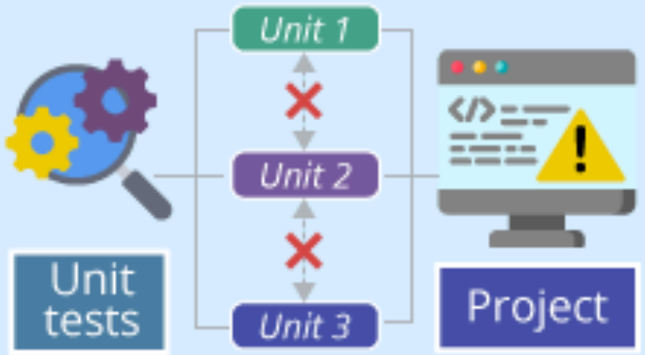
1. In-house, outsourced, or partially outsourced software development



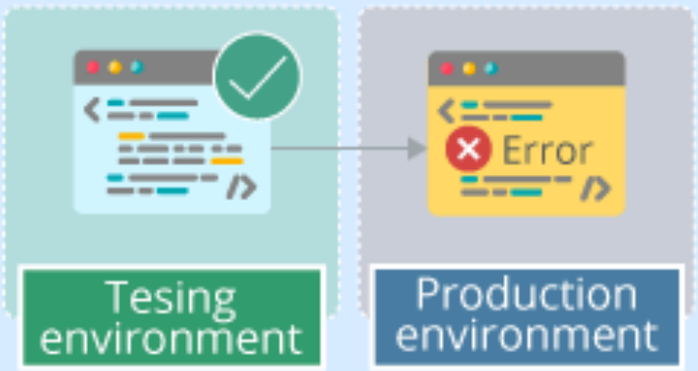
2. Strict segregation of duties between the departments



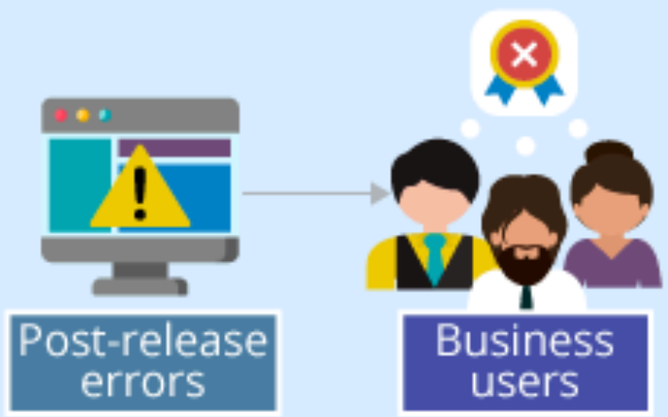
3. Insufficient coverage by tests



4. High probability of post-release errors

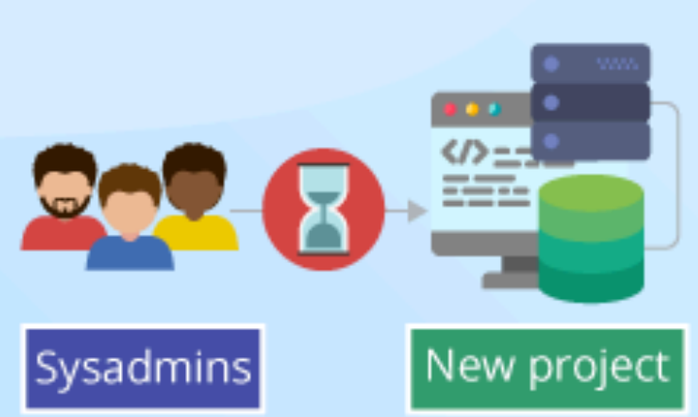


5. Lack of users' trust in software quality



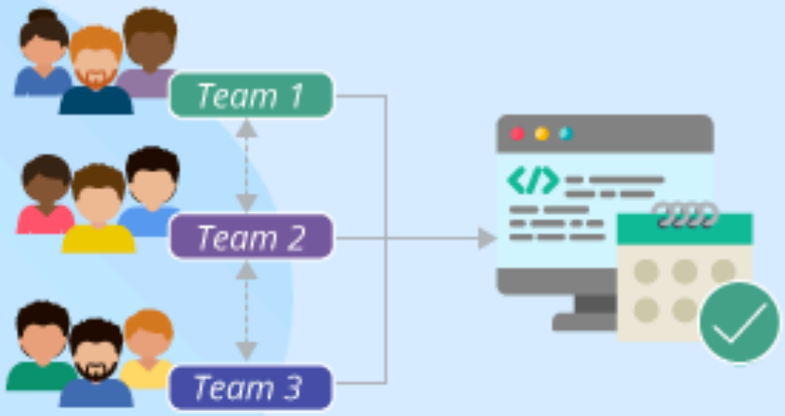
6. Weeks for updates and fixes

7. Time-consuming deployment of the infrastructure



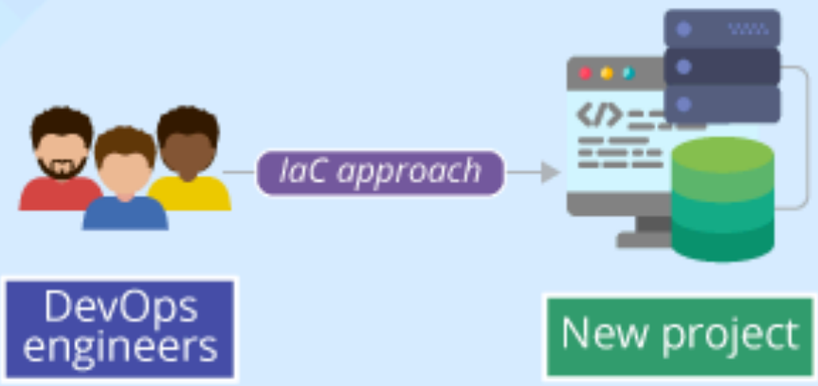
DevOps

1. Constant communication between the teams engaged in software development

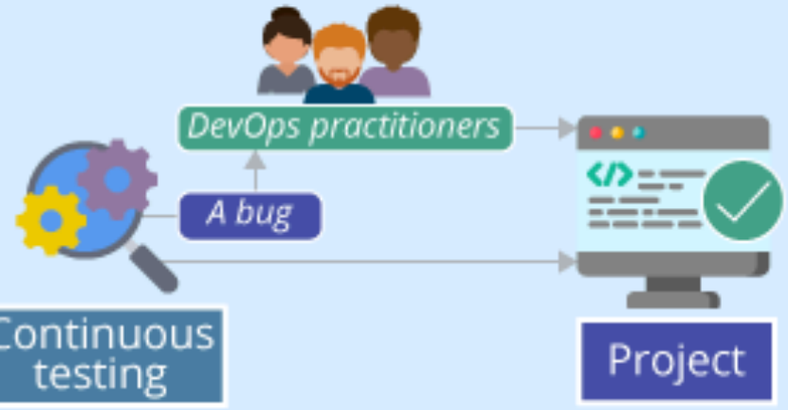


2. Fewer software failures caused by the differences in infrastructure configurations

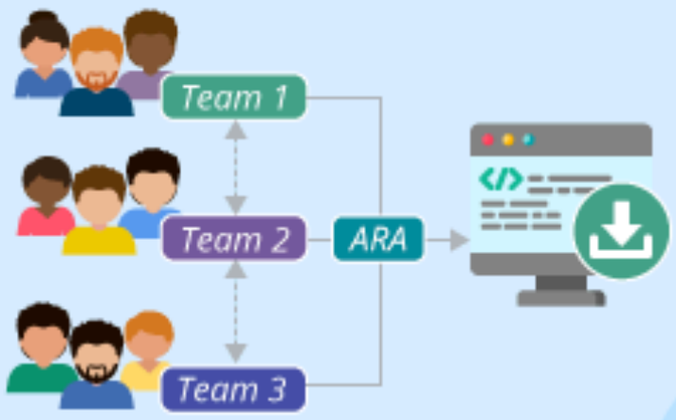
3. Fast provision of new infrastructure



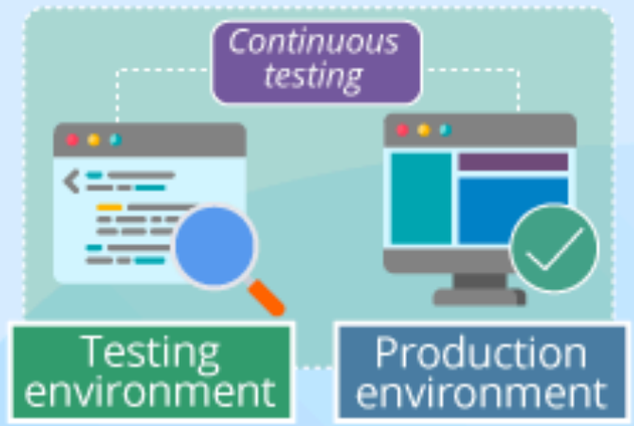
4. The increased amount of test automation



5.Quick and reliable delivery of application updates

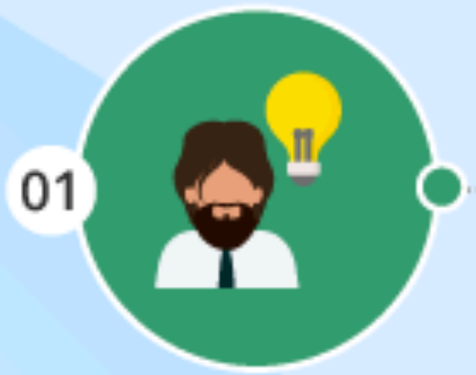


6. Fewer post-release errors



7. Improved users' trust

DEVOPS IMPLEMENTATION ROADMAP



01

Organizing a DevOps initiative

02

Building the DevOps strategy

03

Containerizing

04

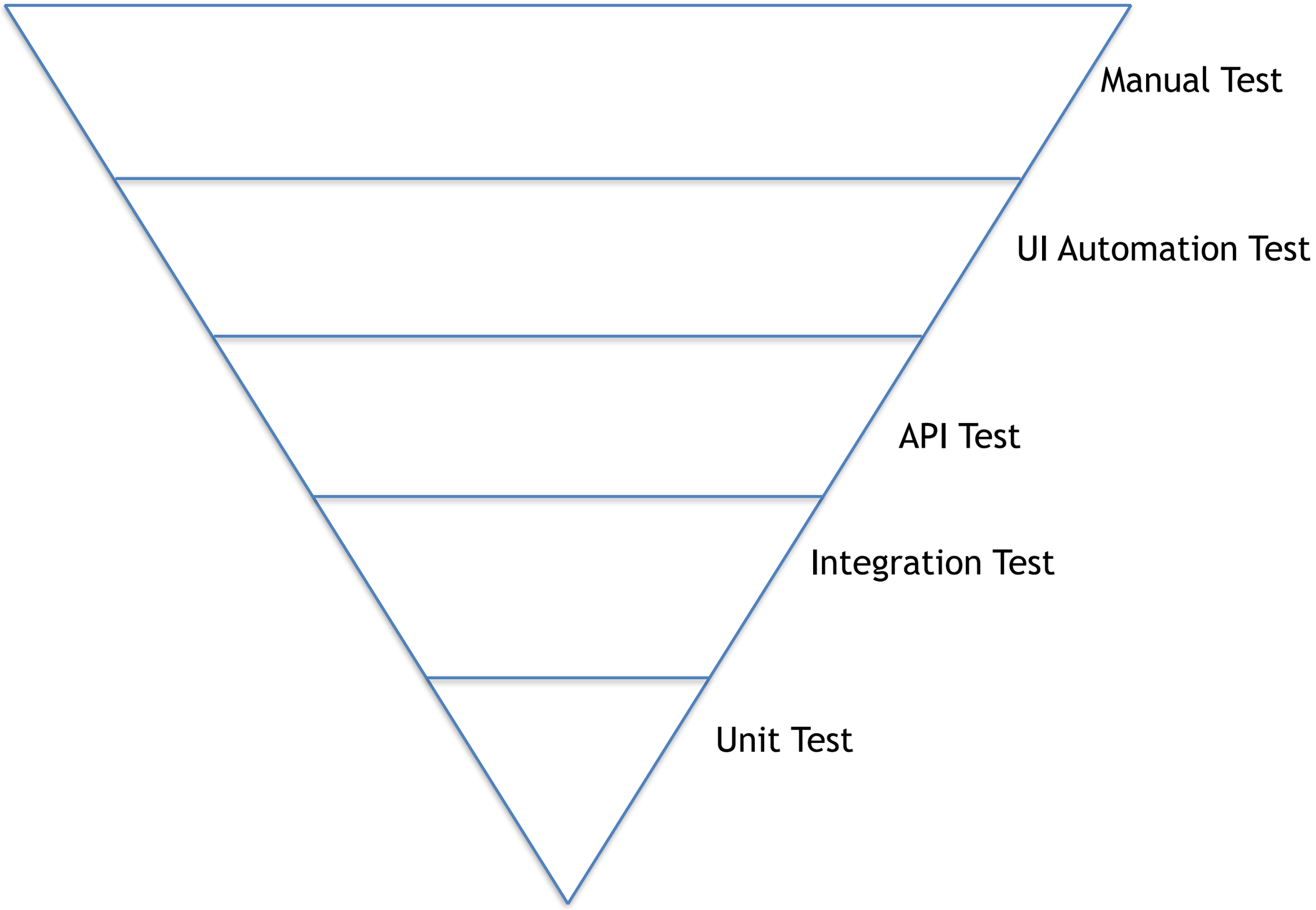
Integrating infrastructure automation with CI/CD tools

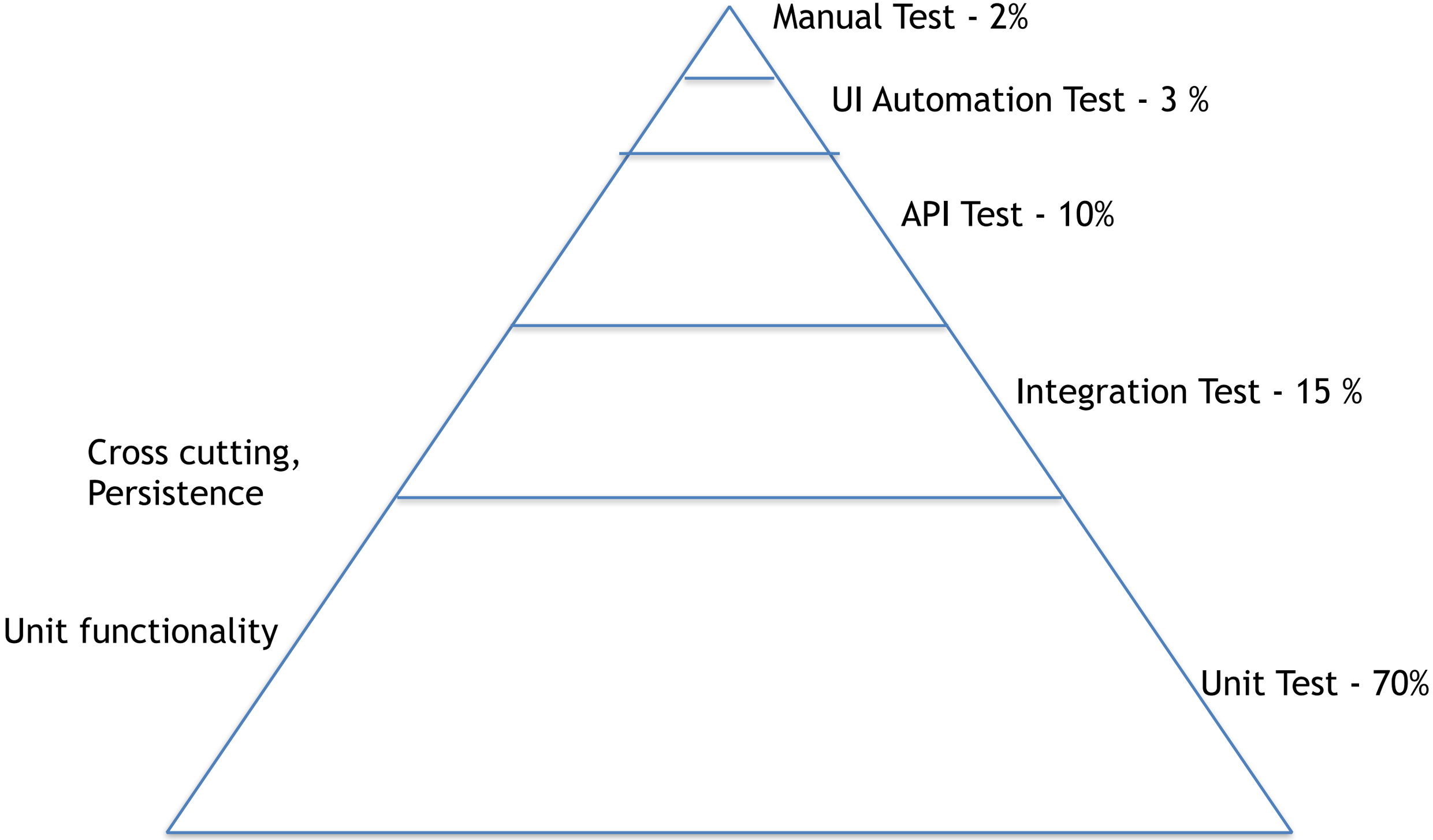
05

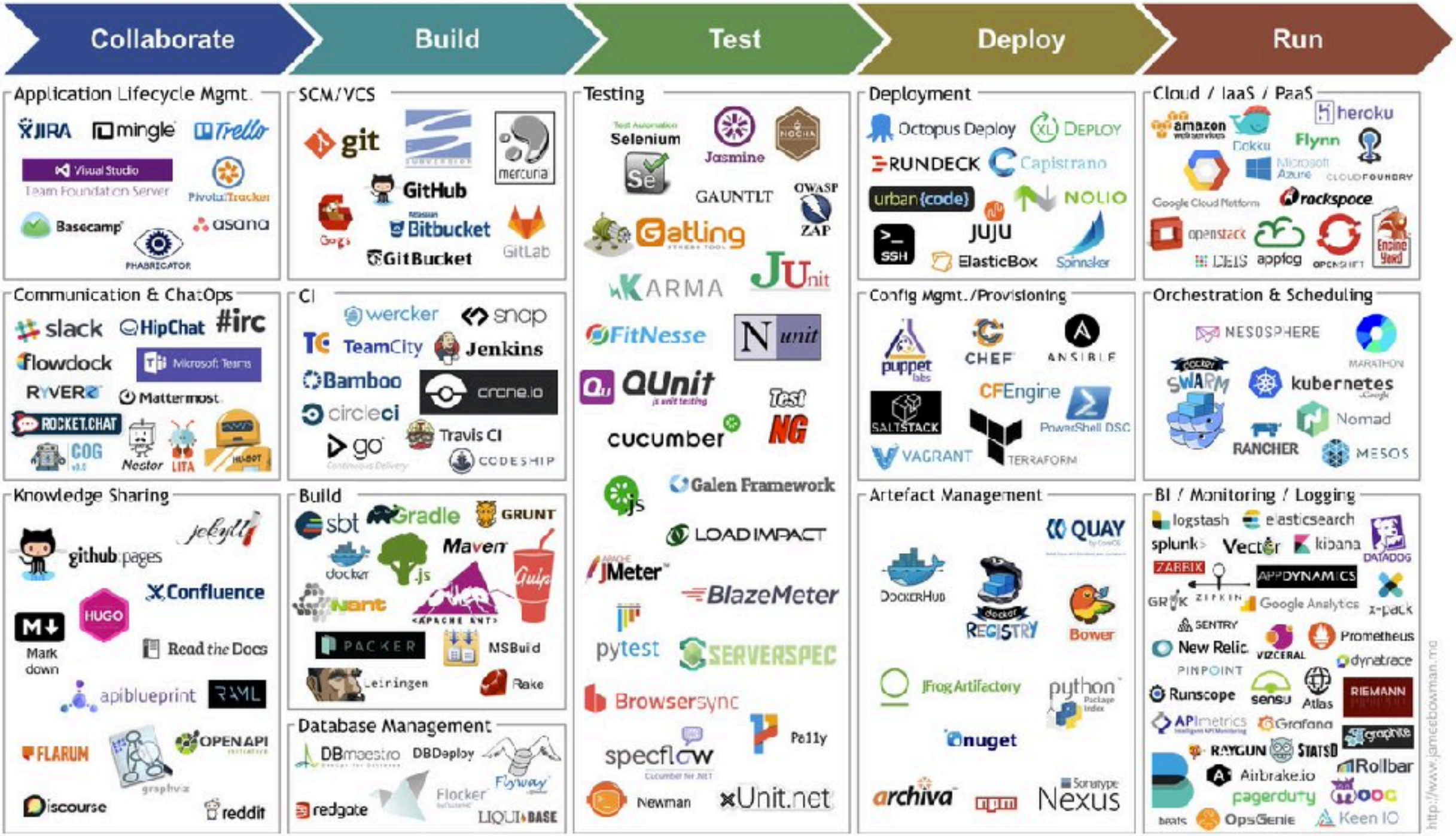
Increasing the amount of test automation and aligning QA with development

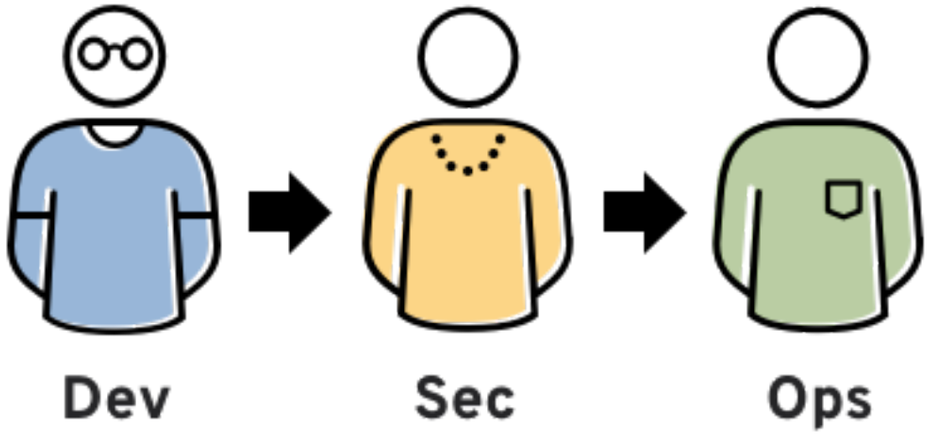
06

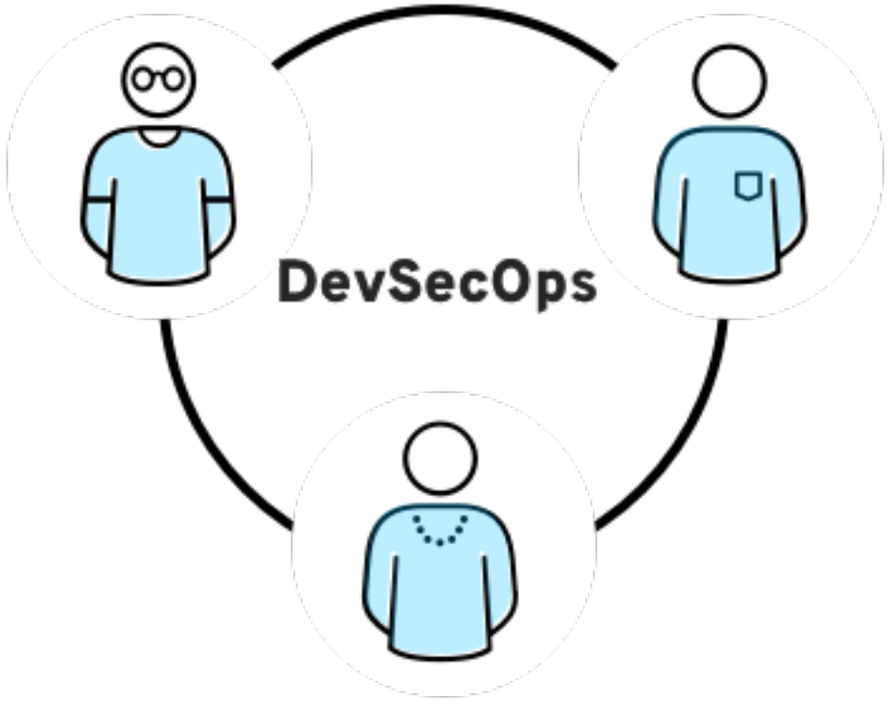
Ensuring total application performance monitoring


Manual Test
UI Automation Test
API Test
Integration Test
Unit Test


Manual Test - 2%
UI Automation Test - 3 %
API Test - 10%
Integration Test - 15 %
Cross cutting,
Persistence
Unit functionality
Unit Test - 70%

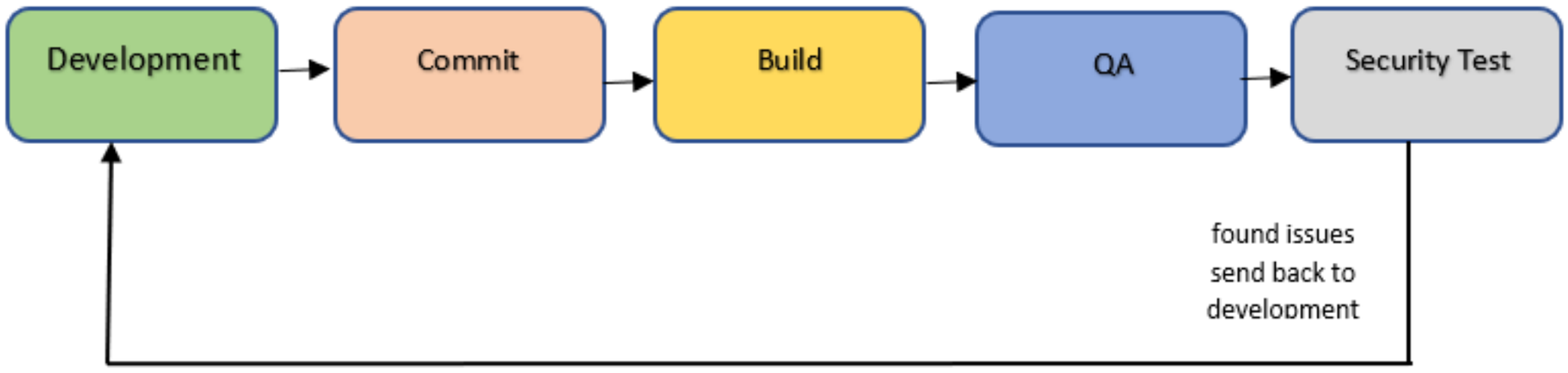

Collaborate
Build
Test
Deploy
Run
Application Lifecycle Mgmt.
JIRA
mingle
Trello
Visual Studio
Team Foundation Server
PivotalTracker
Basecamp
asana
PHABRICATOR
SCM/VCS
git
subversion
mercurial
GitHub
Gogs
Bitbucket
GitBucket
GitLab
Testing
Test Automation
Selenium
Jasmine
NOCHA
GAUNTLT
OWASP
ZAP
Gatling
JUnit
KARMA
FitNesse
N unit
QUnit
js unit testing
cucumber
Test
NG
Galen Framework
LOAD IMPACT
JMeter
BlazeMeter
pytest
SERVERSPEC
Browsersync
Pa11y
specflow
Cucumber for .NET
Newman
xUnit.net
Deployment
Octopus Deploy
DEPLOY
RUNDECK
Capistrano
urban{code}
NOLIO
SSH
JUJU
ElasticBox
Spinnaker
Cloud / IaaS / PaaS
amazon web services
Deis
heroku
Flynn
Microsoft Azure
CLOUDFOUNDRY
Google Cloud Platform
rackspace
openstack
DEIS
appfog
OPENSHIFT
Engine Yard
Communication & ChatOps
slack
HipChat
#irc
flowdock
Microsoft Teams
RYVER
Mattermost
ROCKET.CHAT
COG
Nectar
LITA
HUBOT
CI
wercker
snap
TeamCity
Jenkins
Bamboo
circleci
cruise.io
go
Travis CI
CODESHIP
Config Mgmt./Provisioning
puppet labs
CHEF
ANSIBLE
CFEngine
SALTSTACK
PowerShell DSC
VAGRANT
TERRAFORM
Orchestration & Scheduling
MESOSPHERE
MARATHON
SWARM
kubernetes
Nomad
RANCHER
MESOS
Knowledge Sharing
github pages
jekyll
Confluence
HUGO
Mark down
Read the Docs
apiblueprint
RAML
FLARUM
OPEN API
graphviz
Discourse
reddit
Build
sbt
Gradle
GRUNT
docker
Maven
.js
Gulp
nant
<APACHE ANT>
PACKER
MSBuild
Leiningen
Rake
Database Management
DBmaestro
DBDeploy
Flyway
redgate
Flocker
LIQUIBASE
Galen Framework
LOAD IMPACT
BlazeMeter
SERVERSPEC
Artefact Management
QUAY
DockerHub
docker REGISTRY
Bower
JFrog Artifactory
python Package Index
nuget
archiva
npm
Sonatype Nexus
BI / Monitoring / Logging
logstash
elasticsearch
splunk
Vector
kibana
DATADOG
ZABBIX
APPDYNAMICS
GROK
ZIPKIN
Google Analytics
x-pack
SENTRY
New Relic
VIZCERAL
Prometheus
dynatrace
PINPOINT
Runscope
sensu
Atlas
RIEMANN
APImetrics
Grafana
graphite
RAYGUN
STATSD
Rollbar
Airbrake.io
pagerduty
OOOC
beats
OpsGenie
Keen IO
http://www.jamescbowman.net



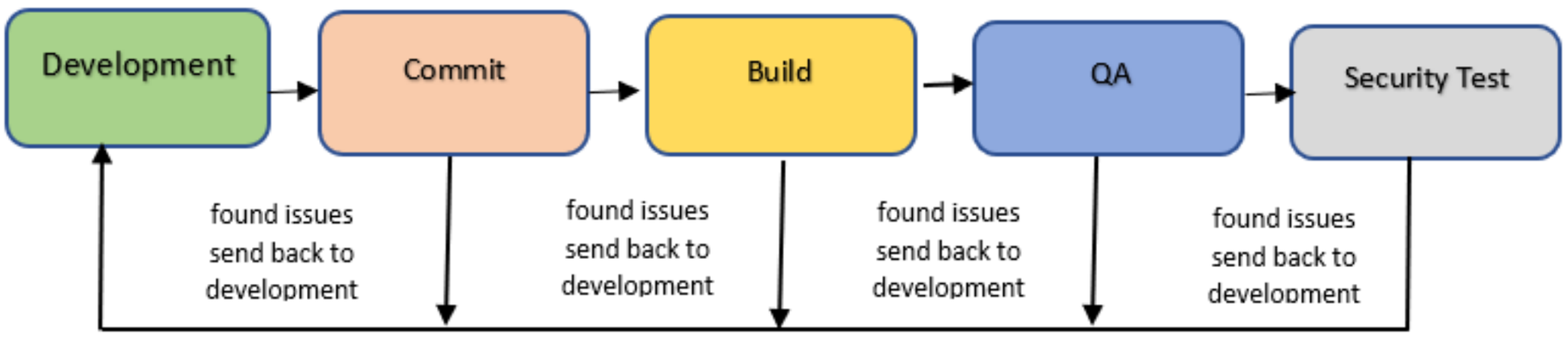


DevSecOps is about injecting security into the DevOps lifecycle.

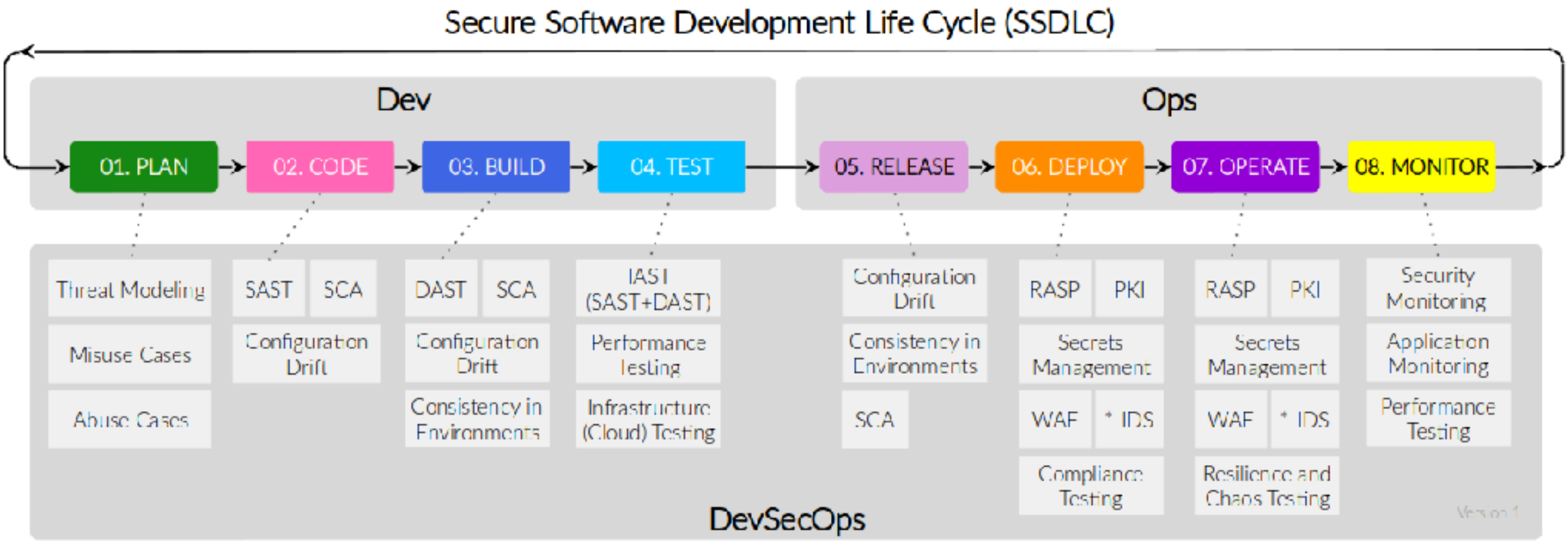
"Security in traditional way"



"Security in DevSecOps way"



SAST **DAST** **IAST (SAST + DAST)** **RASP (SAST + DAST)**



DAST vs. SAST

Vulnerability Coverage

SAST

+ Null Pointer Dereference
+ Threading Issues
+ Code Quality Issues
+ Insecure Crypto Issues
+ Issues in Non Web application Code

- Higher number of FP
- Run time Code generation
- Dynamic Languages (Ruby + Python)

+ SQL Injection
+ Cross Site Scripting (XSS)
+ OS Commanding
+ HTTP Response Splitting
+ LDAP Injection
+ XPATH Injection
+ Path Traversal
+ Buffer Overflows
+ Format String Issues

DAST

+ Runtime Privilege Issues
+ Authentication Issues
+ Session Management Issues
+ Insecure 3rd Party Libraries
+ Business Logic Vulnerabilities
+ Protocol Parser Issues

- Web2.0, JSON, Flash, HTML 5.0,
- Integrity and Availability violations
- Long Execution Times

SAST vs. DAST

Static application security testing (SAST) and dynamic application security testing (DAST) are both methods of testing for security vulnerabilities, but they're used very differently.

Here are some key differences between the two:

White box security testing

- The tester has access to the underlying framework, design, and implementation.
- The application is tested from the inside out.
- This type of testing represents the developer approach.

Black box security testing

- The tester has no knowledge of the technologies or frameworks that the application is built on.
- The application is tested from the outside in.
- This type of testing represents the hacker approach.

Requires source code

- SAST doesn't require a deployed application.
- It analyzes the source code or binary without executing the application.

Requires a running application

- DAST doesn't require source code or binaries.
- It analyzes by executing the application.

Finds vulnerabilities earlier in the SDLC

- The scan can be executed as soon as code is deemed feature-complete.

Finds vulnerabilities toward the end of the SDLC

- Vulnerabilities can be discovered after the development cycle is complete.

Less expensive to fix vulnerabilities

- Since vulnerabilities are found earlier in the SDLC, it's easier and faster to remediate them.
- Findings can often be fixed before the code enters the QA cycle.

More expensive to fix vulnerabilities

- Since vulnerabilities are found toward the end of the SDLC, remediation often gets pushed into the next cycle.
- Critical vulnerabilities may be fixed as an emergency release.

Can't discover run-time and environment-related issues

- Since the tool scans static code, it can't discover run-time vulnerabilities.

Can discover run-time and environment-related issues

- Since the tool uses dynamic analysis on an application, it is able to find run-time vulnerabilities.

Typically supports all kinds of software

- Examples include web applications, web services, and thick clients.

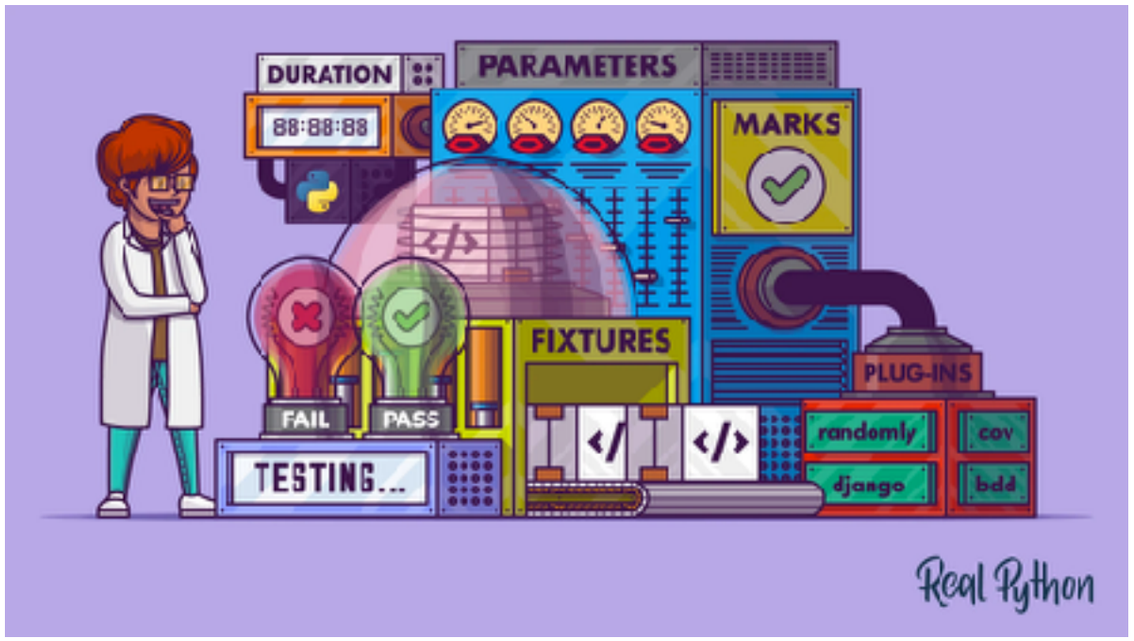
Typically scans only apps like web applications and web services

- DAST is not useful for other types of software.

SAST and DAST techniques complement each other.

Both need to be carried out for comprehensive testing.

IAST: Interactive application security testing. Monitoring an application for security vulnerabilities while it is running — at testing time.



RASP: Runtime application self protection. Monitoring an application to detect attacks while it is running — at production time.

IAST tools look for security vulnerabilities, whereas **RASP** monitors the application for attacks, **and** protects the application against them when it senses an attack happening.

DevSecOps Pipeline Techstack

Legend: DevOps DevSecOps

INOVO | VENTURE PARTNERS

CI/CD Pipeline

Commit code to repository

Build

Deploy

IDE Plugins

JFrog, snyk, Skim

Pre-Commit Hooks

Talisman, git-secrets, HOUND

Secrets Management

HashiCorp Vault, Secret Manager, AWS Secrets Manager

SAST

Checkmarx, DerScanner, ECG

SCA

GitHub, BLACKDUCK, snyk

DAST

RAPID7, acunetix, CONTRAST SECURITY

IAST

CONTRAST SECURITY, SYNOPSYS, VERACODE

Secure Infrastructure as code

clair, anchore, DOCKSCAN

Compliance as code

KitchenCI, SERVERSPEC, CHEF INSPEC

RASP

Signal Sciences, TREND MICRO, imperva

Web Application Firewall

imperva, CLOUDFLARE, wallarm

Monitoring Tools

DATADOG, Nagios, PerfOps

Chaos Engineering

Gremlin, ChaosToolkit

Vulnerability management

Hacker, ARCHERY