





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



Mubarak



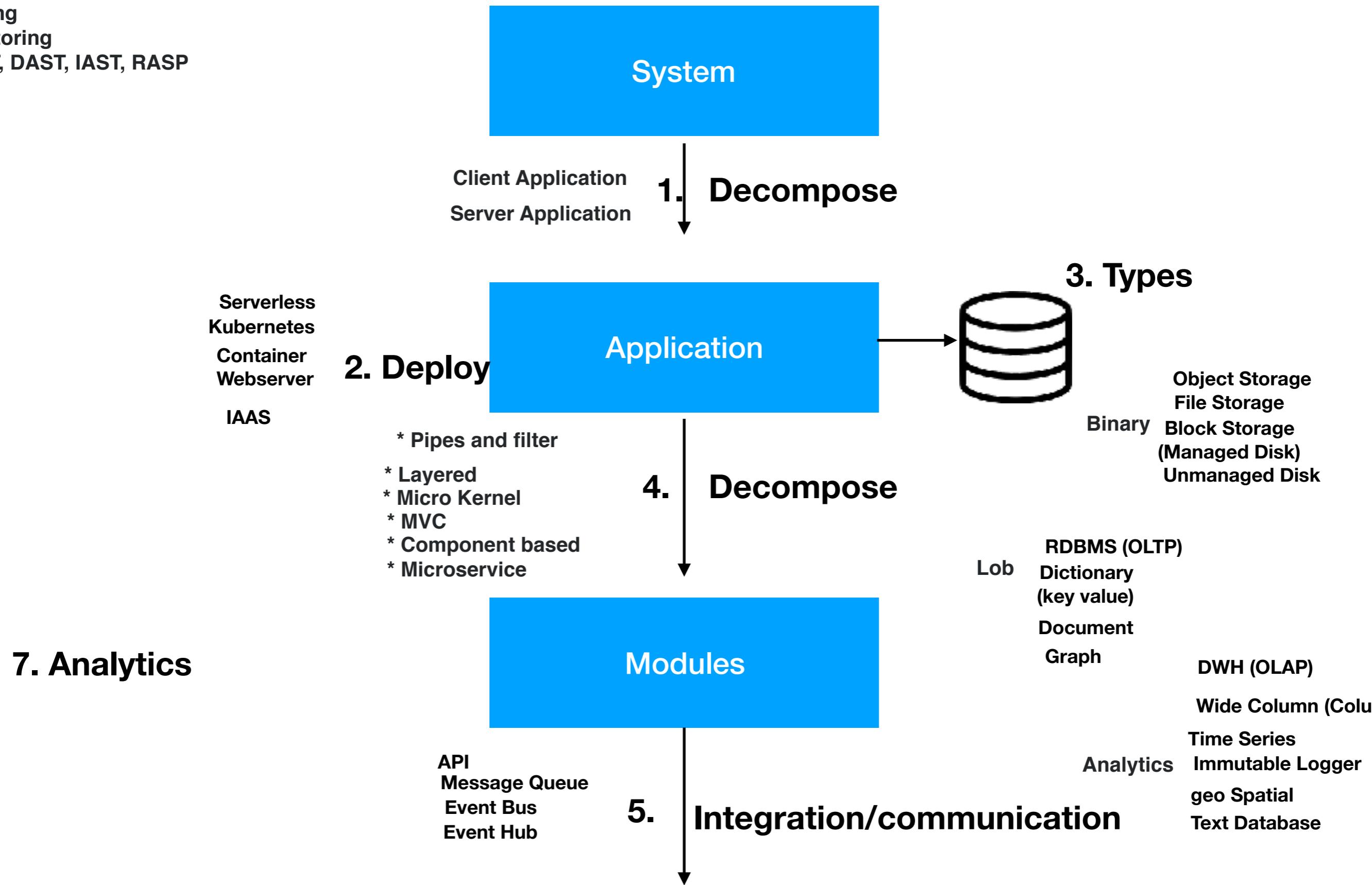
what do  
**YOU**?  
expect.

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

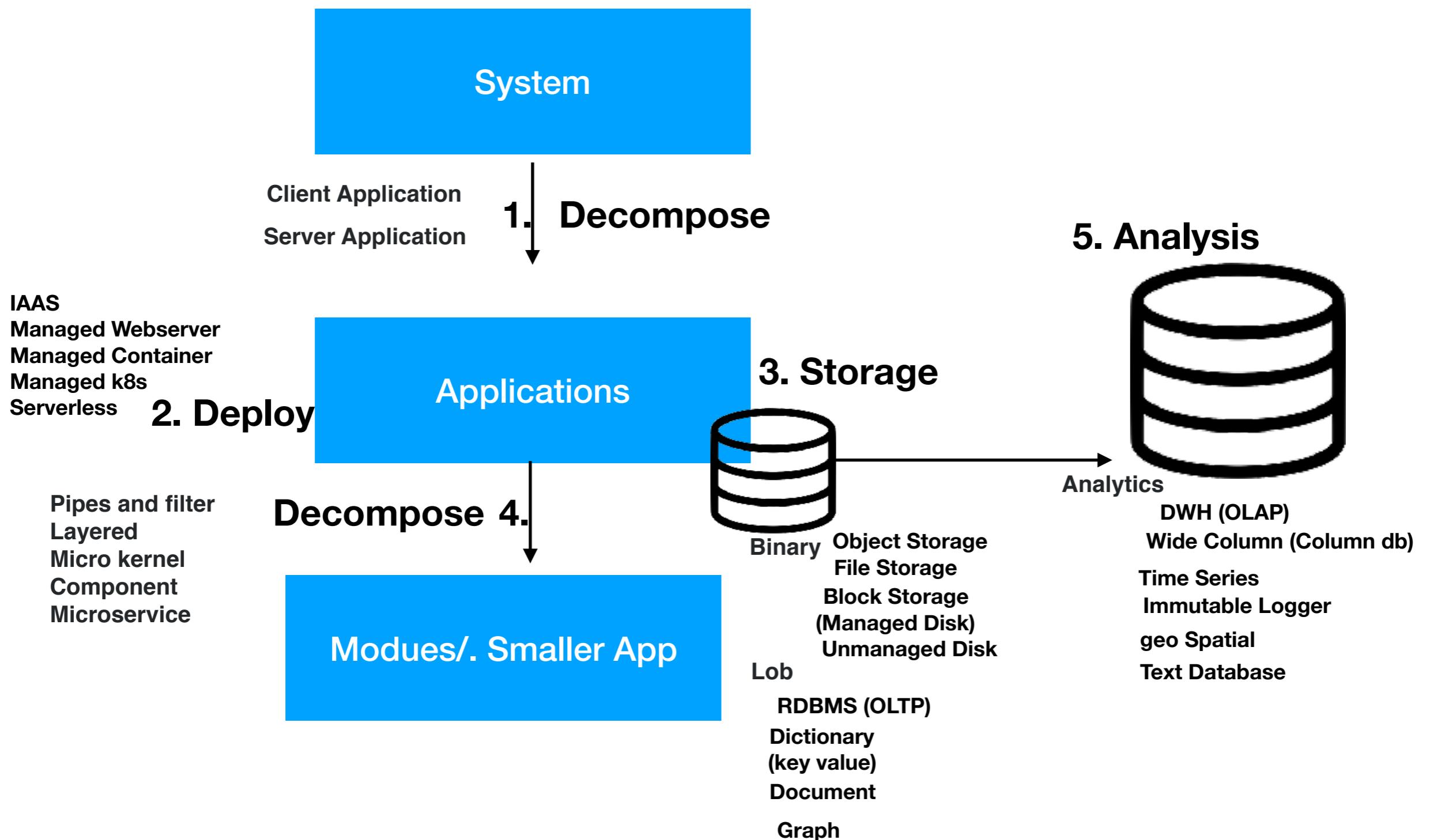
- Years of experience
- Role
- Expectations

## 6. DevSecOps (Dev to Prod)

Containere  
/CD  
Testing  
Monitoring  
AST, DAST, IAST, RASP



# 6. DevOps



System

Client Application  
Server Application

1. Decompose

Applications

Decompose 2.

Pipes and filter  
Layered  
Micro kernel  
Component

Modues

Decompose 2.

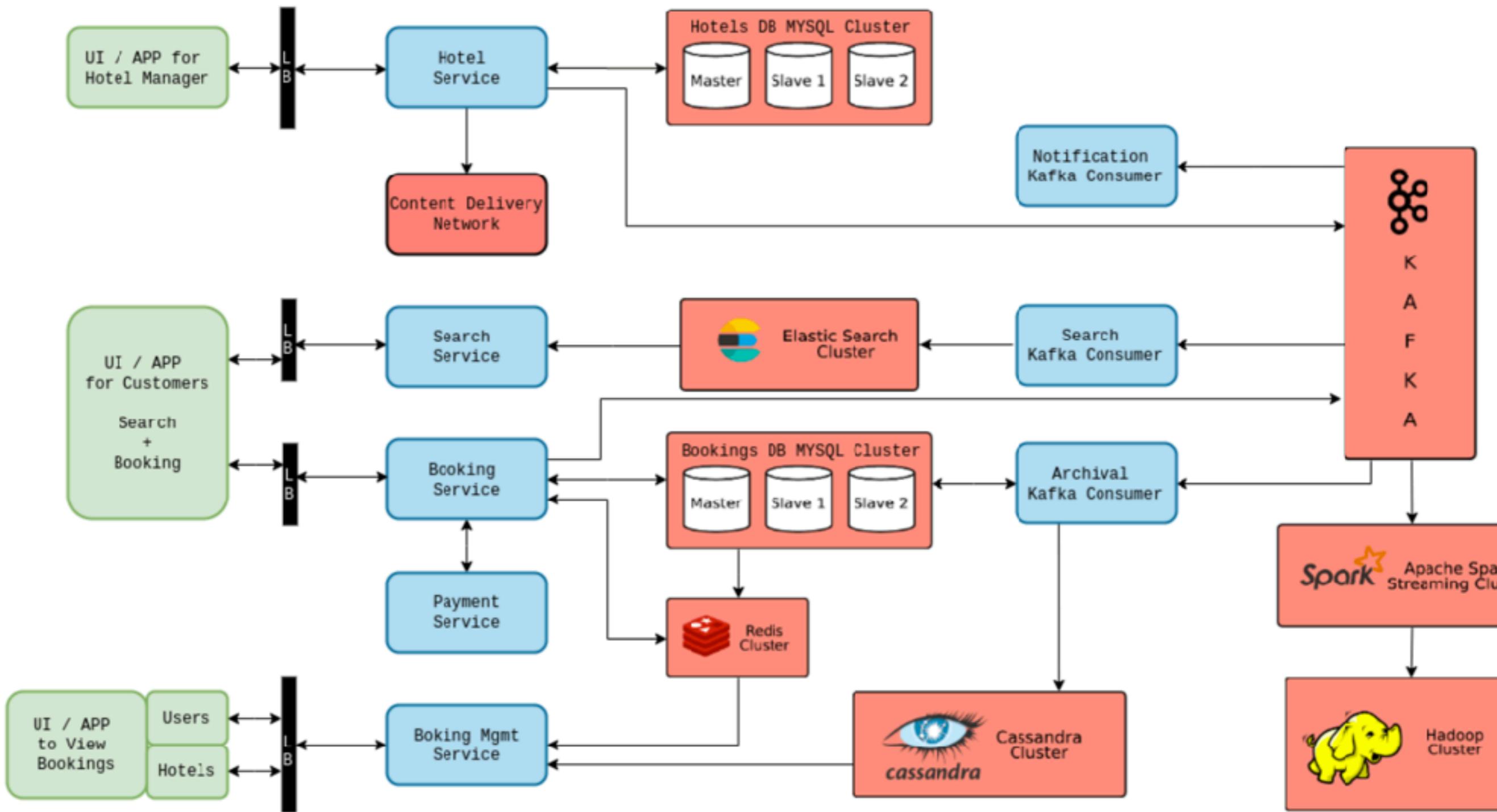
Micro service

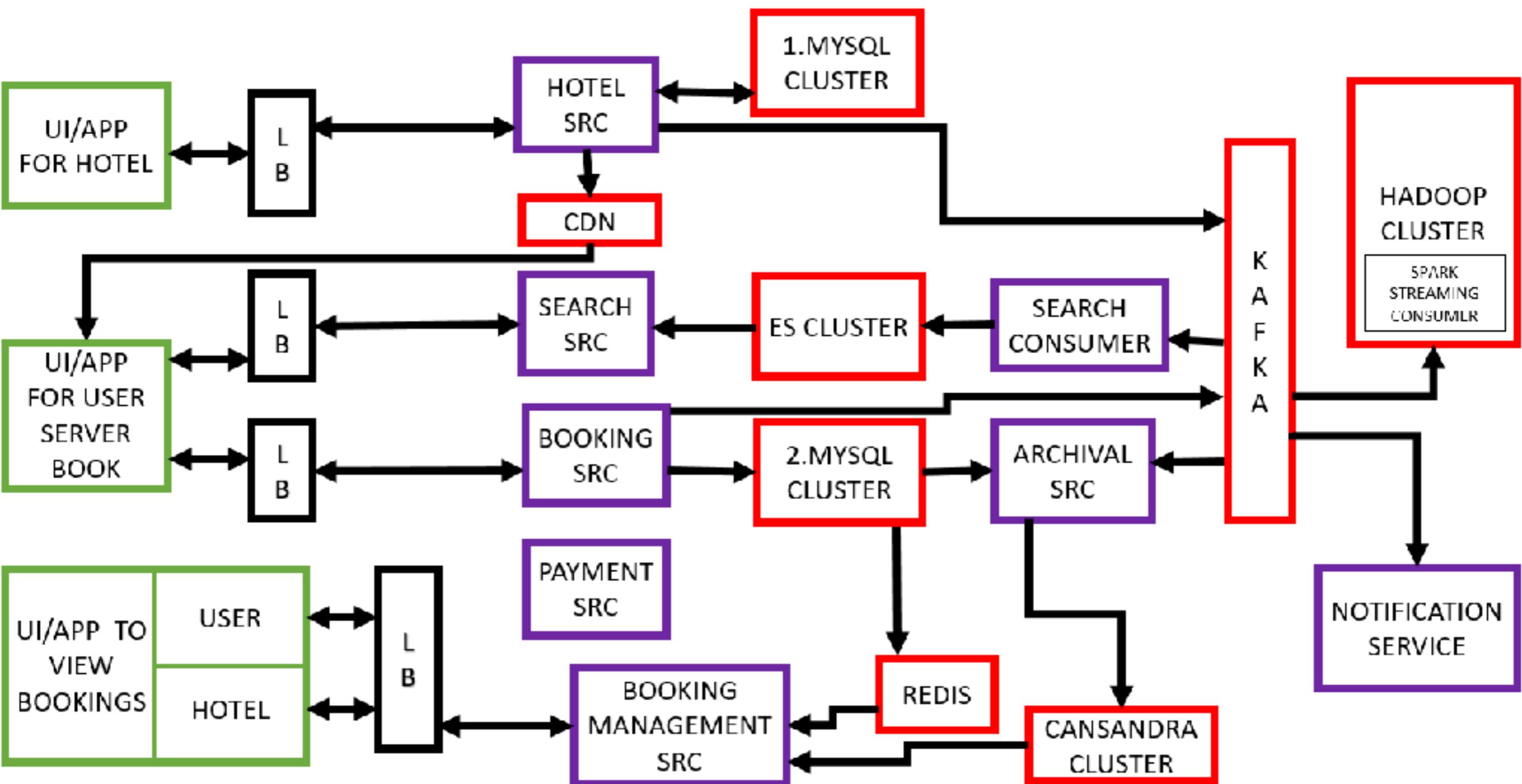
Small Apps

Decompose 3.

Pipes and filter  
Layered  
Micro kernel  
Component

Modues

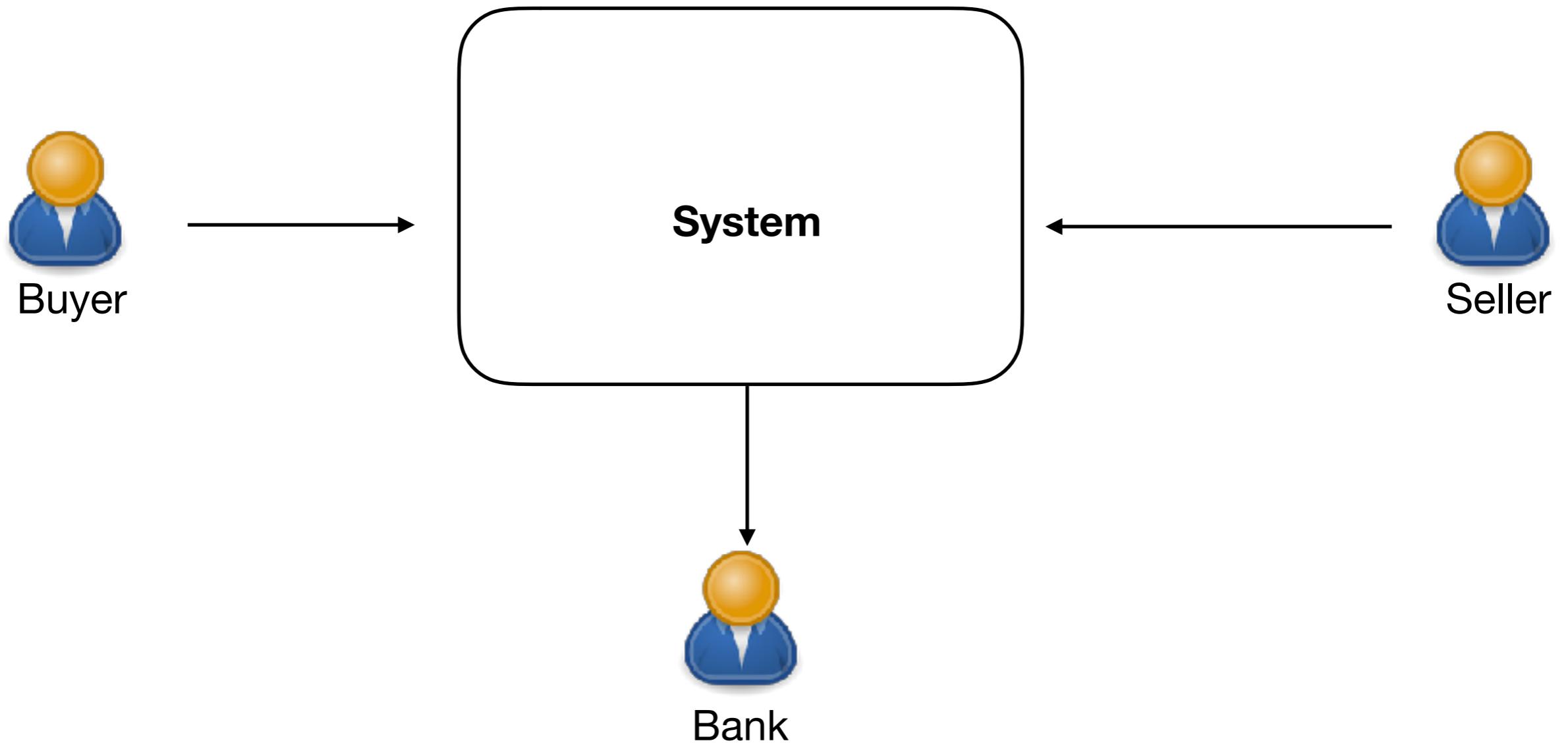




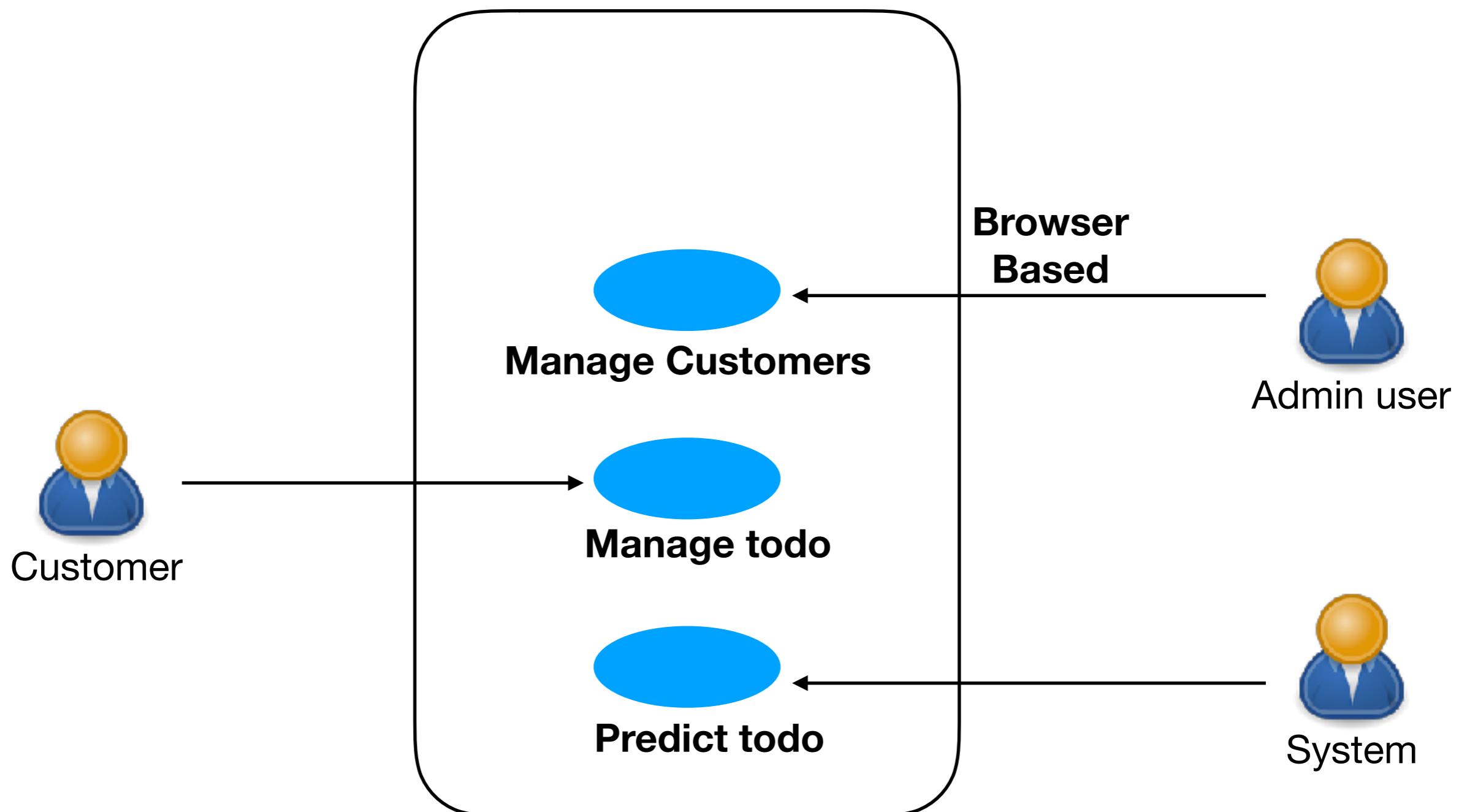
# Context View



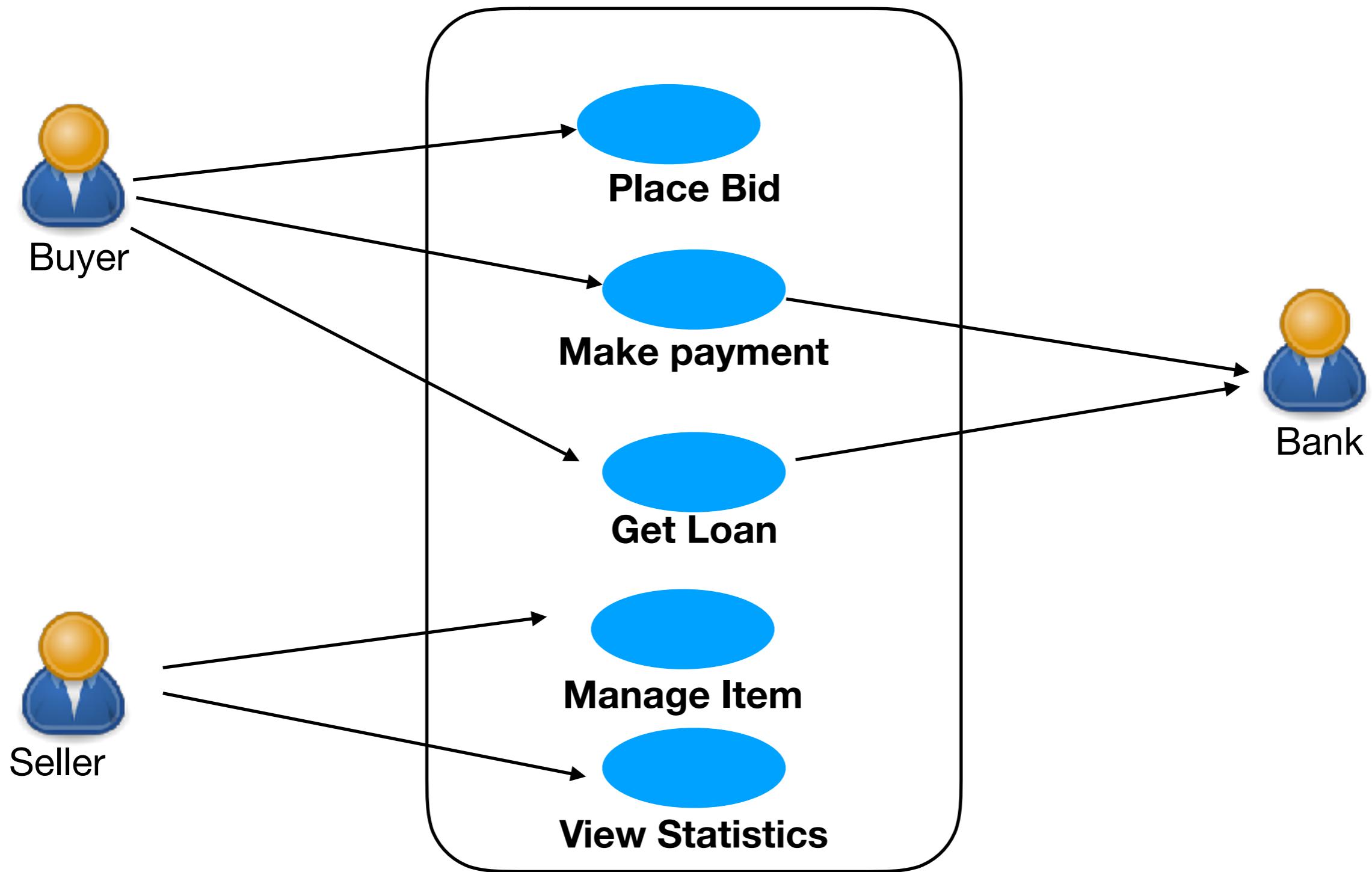
# Context view



# Functional view



# Functional view



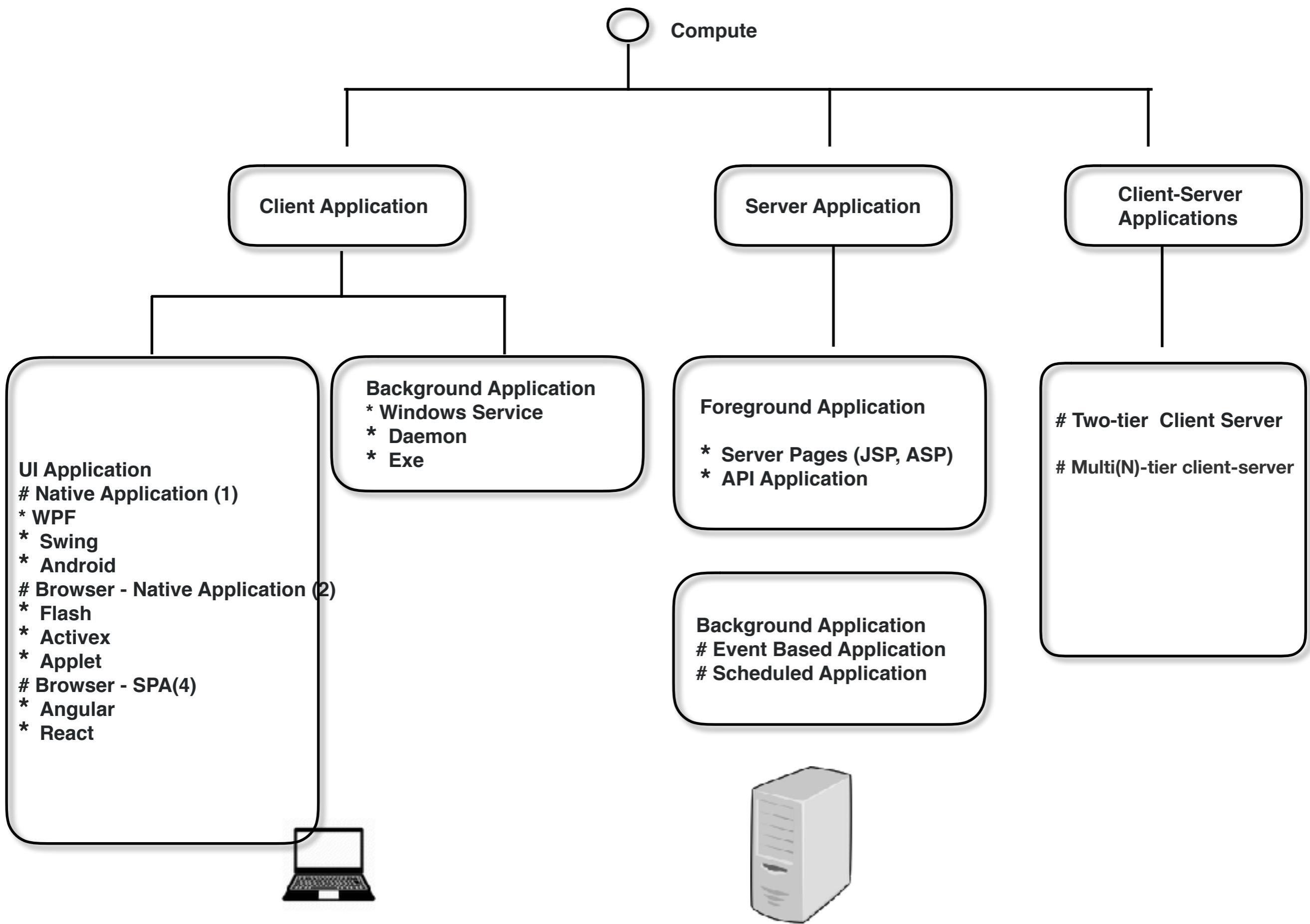
Compute

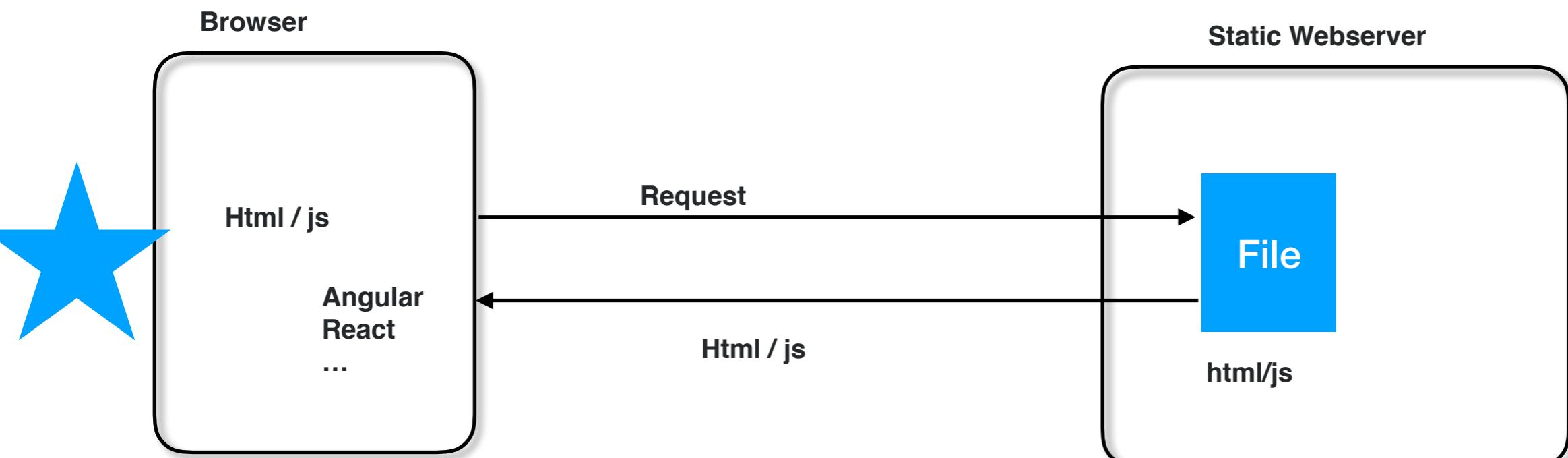
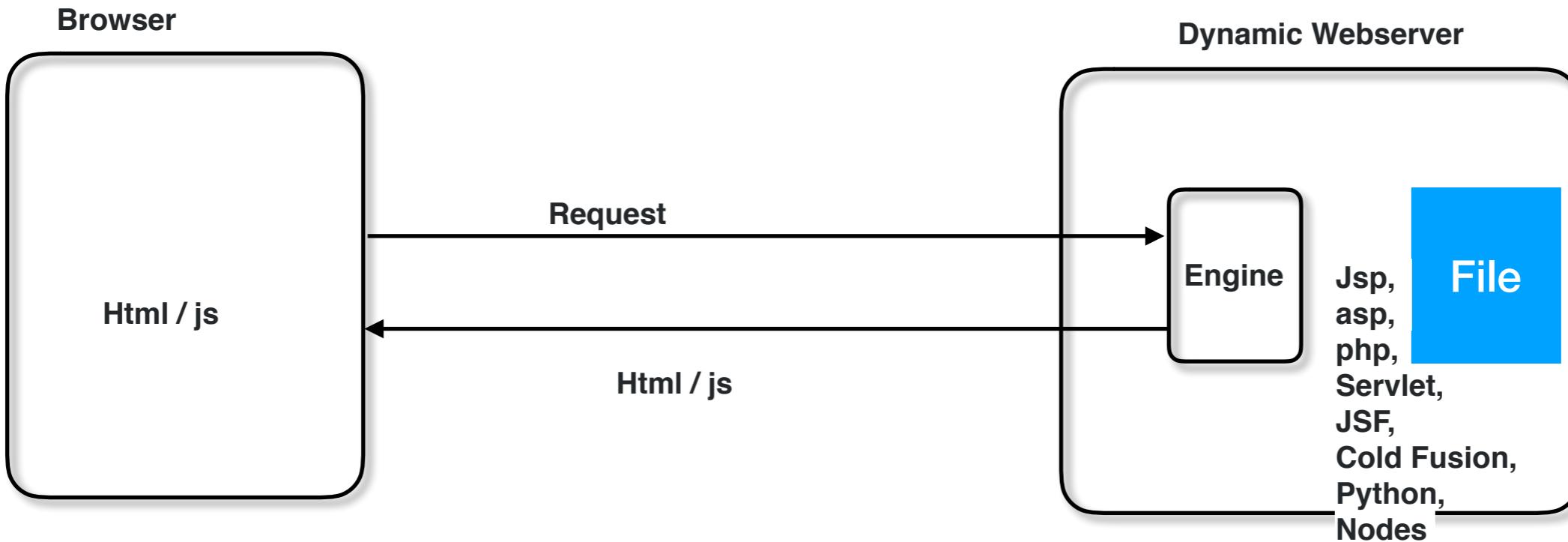
Storage

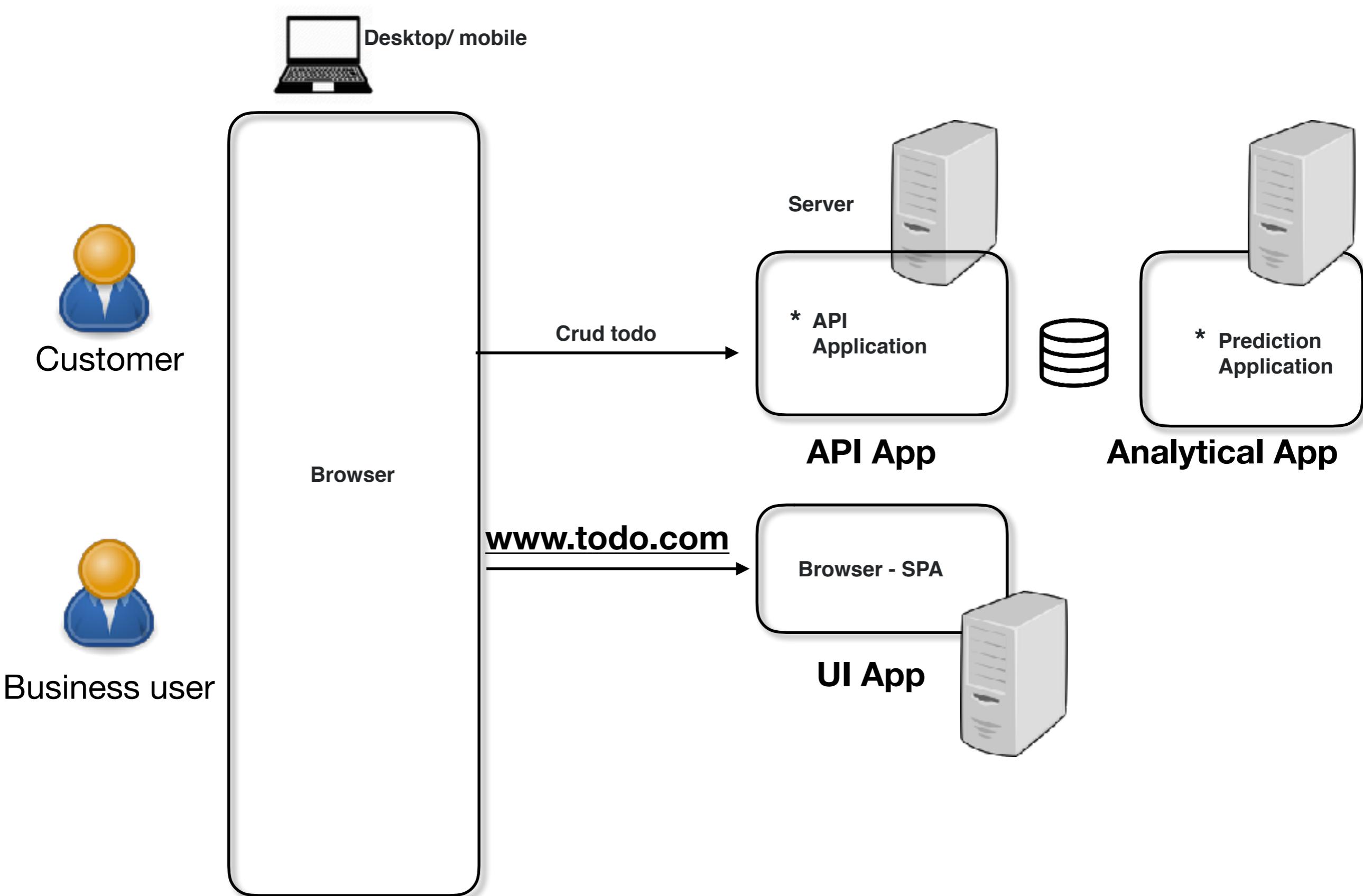
Message

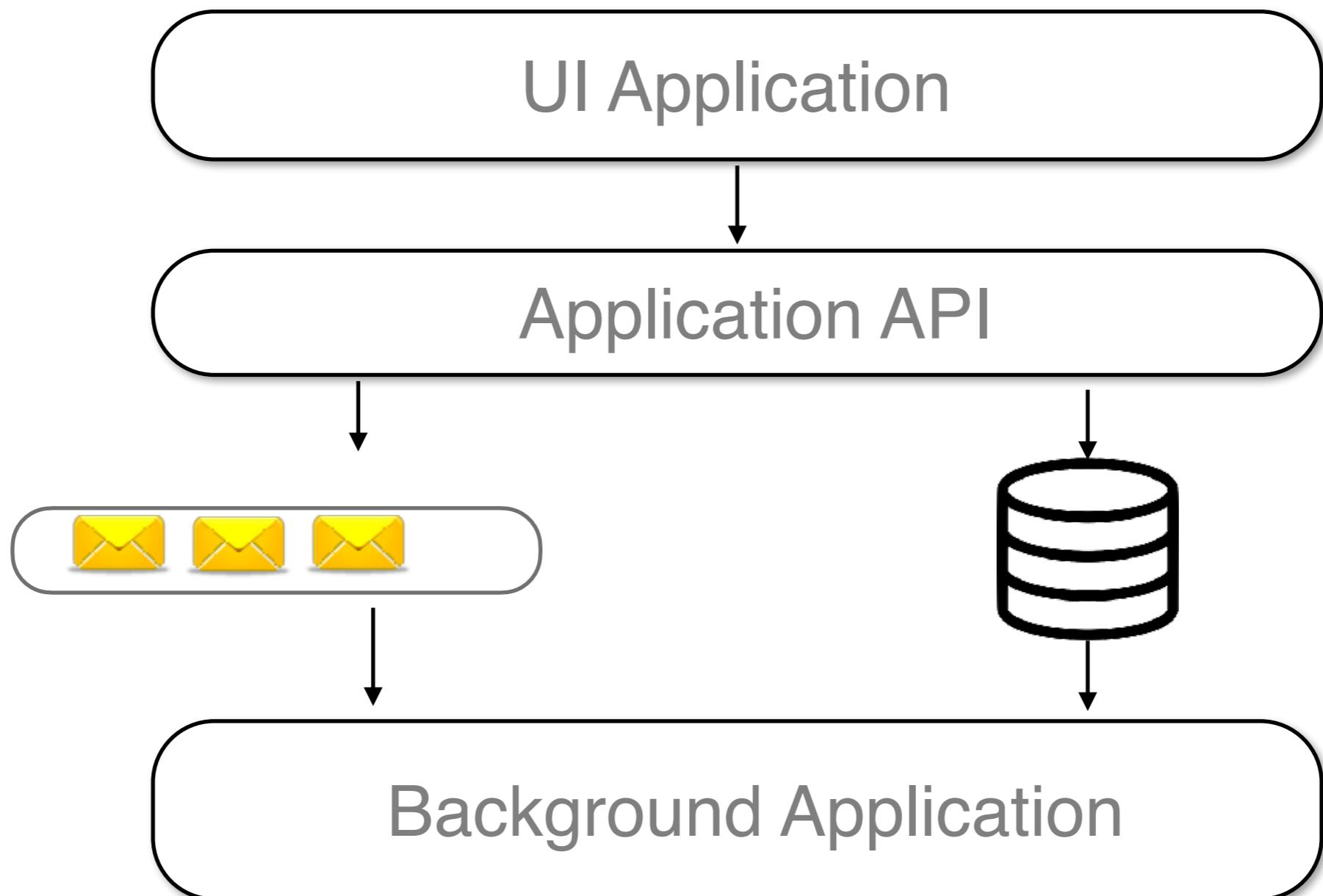
# System Decomposition

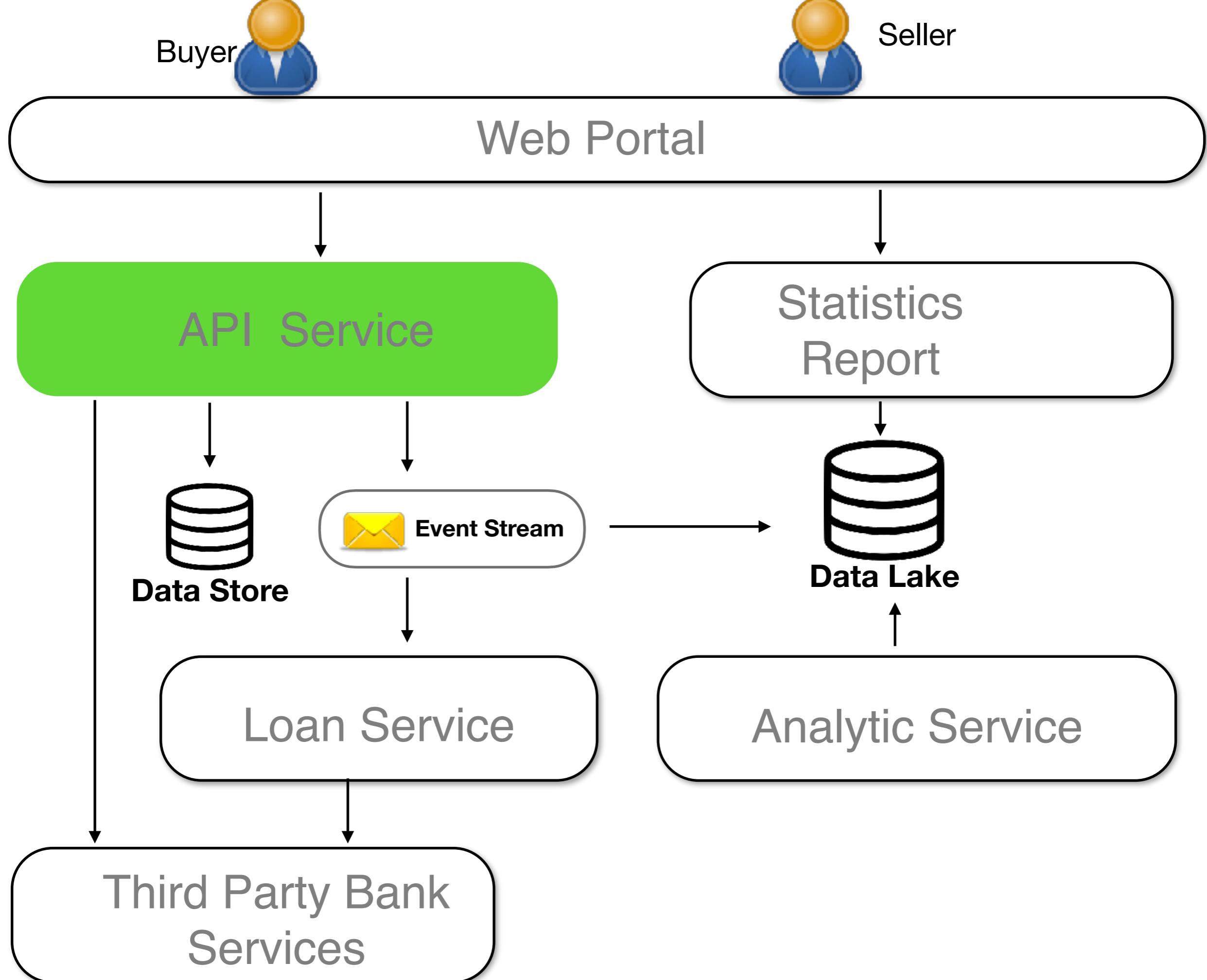




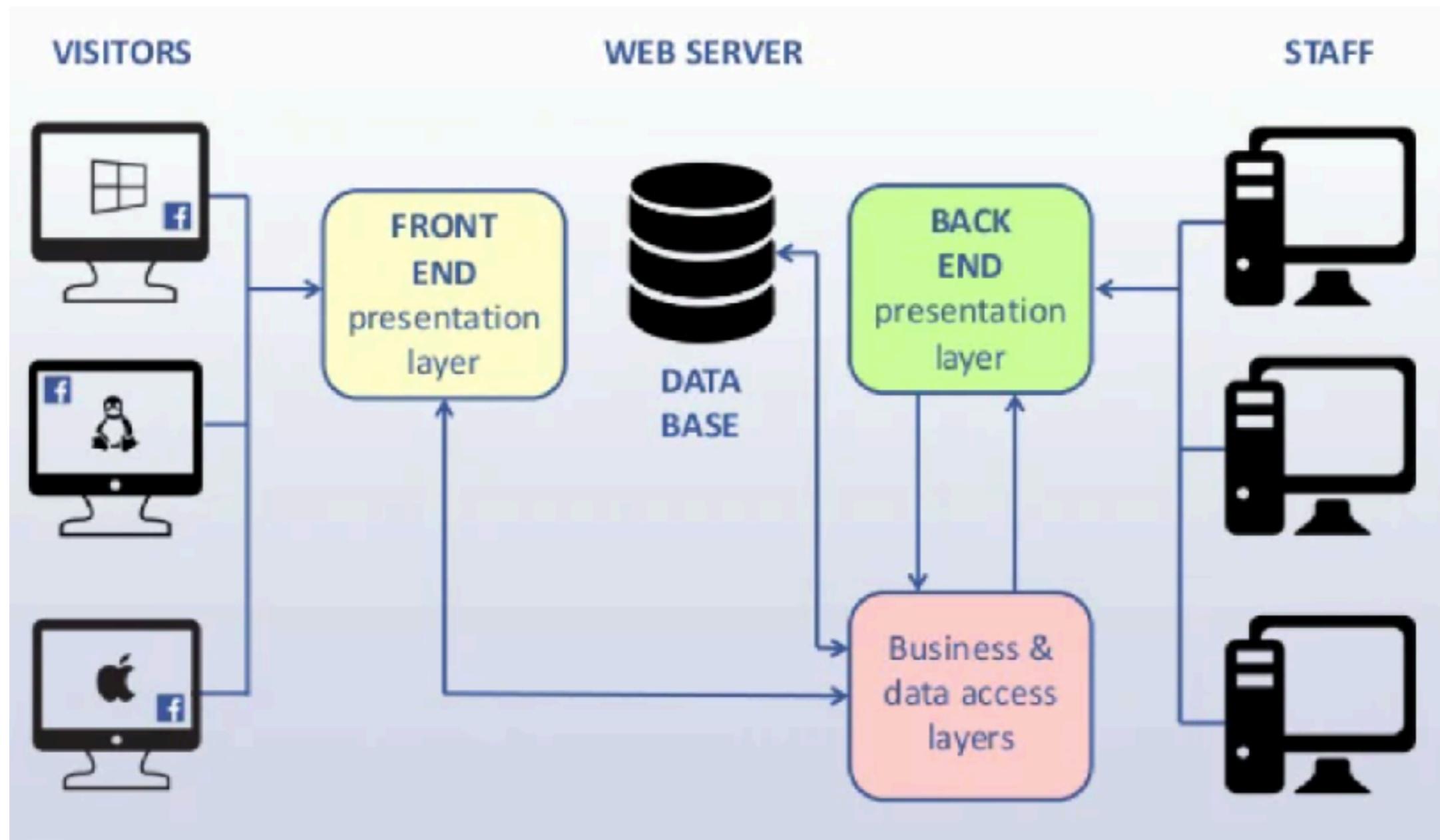




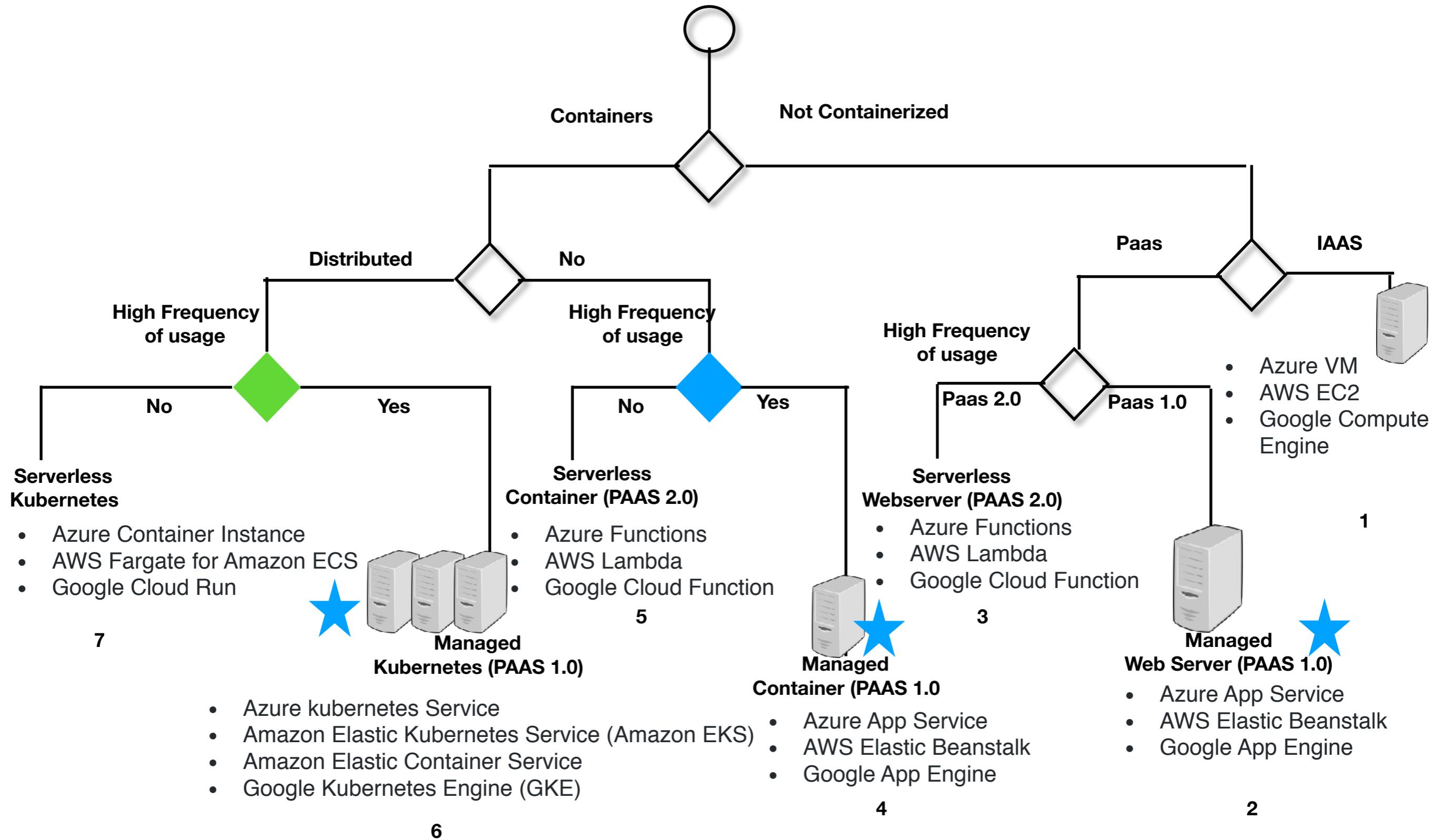


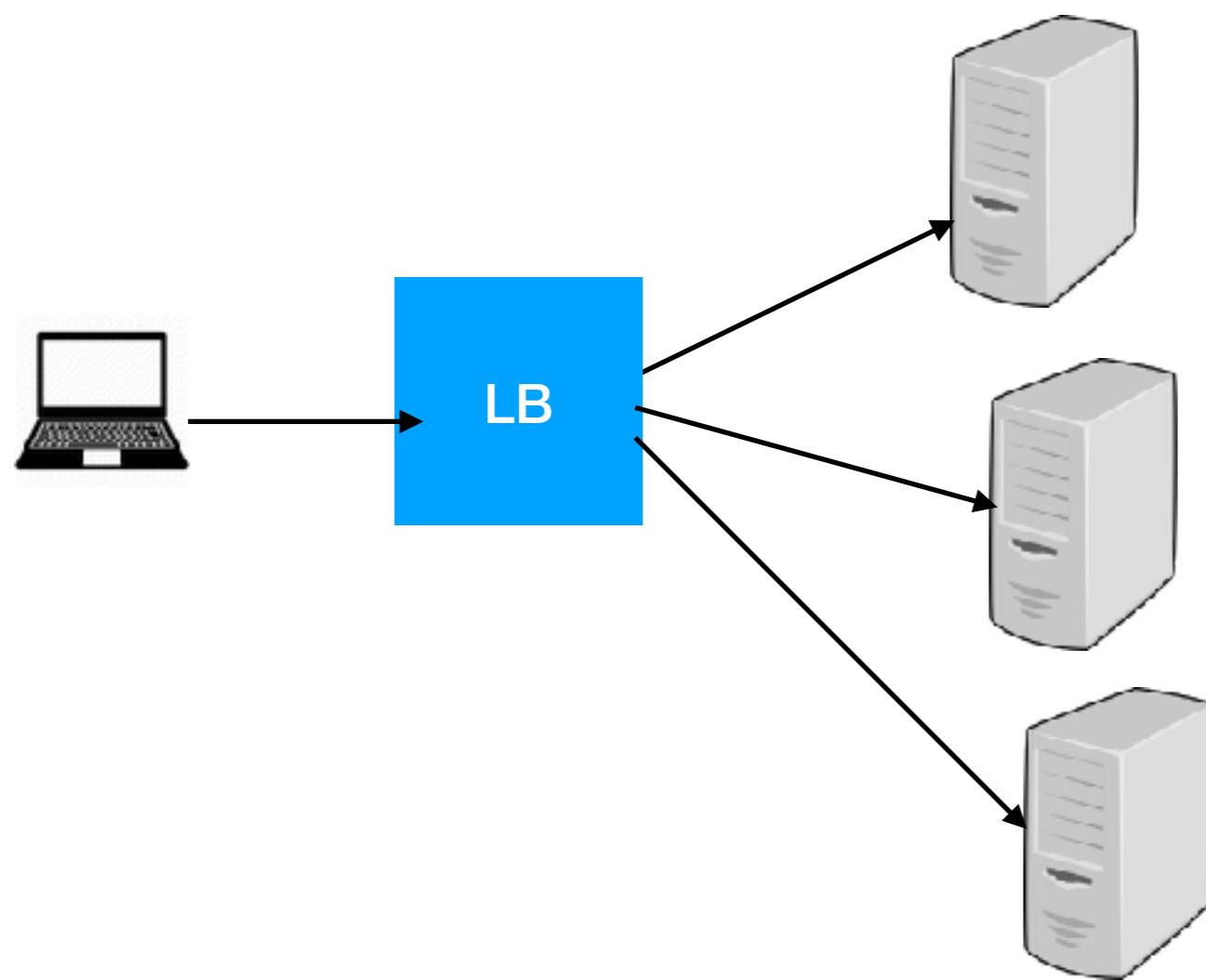


# Facebook Reference Architecture



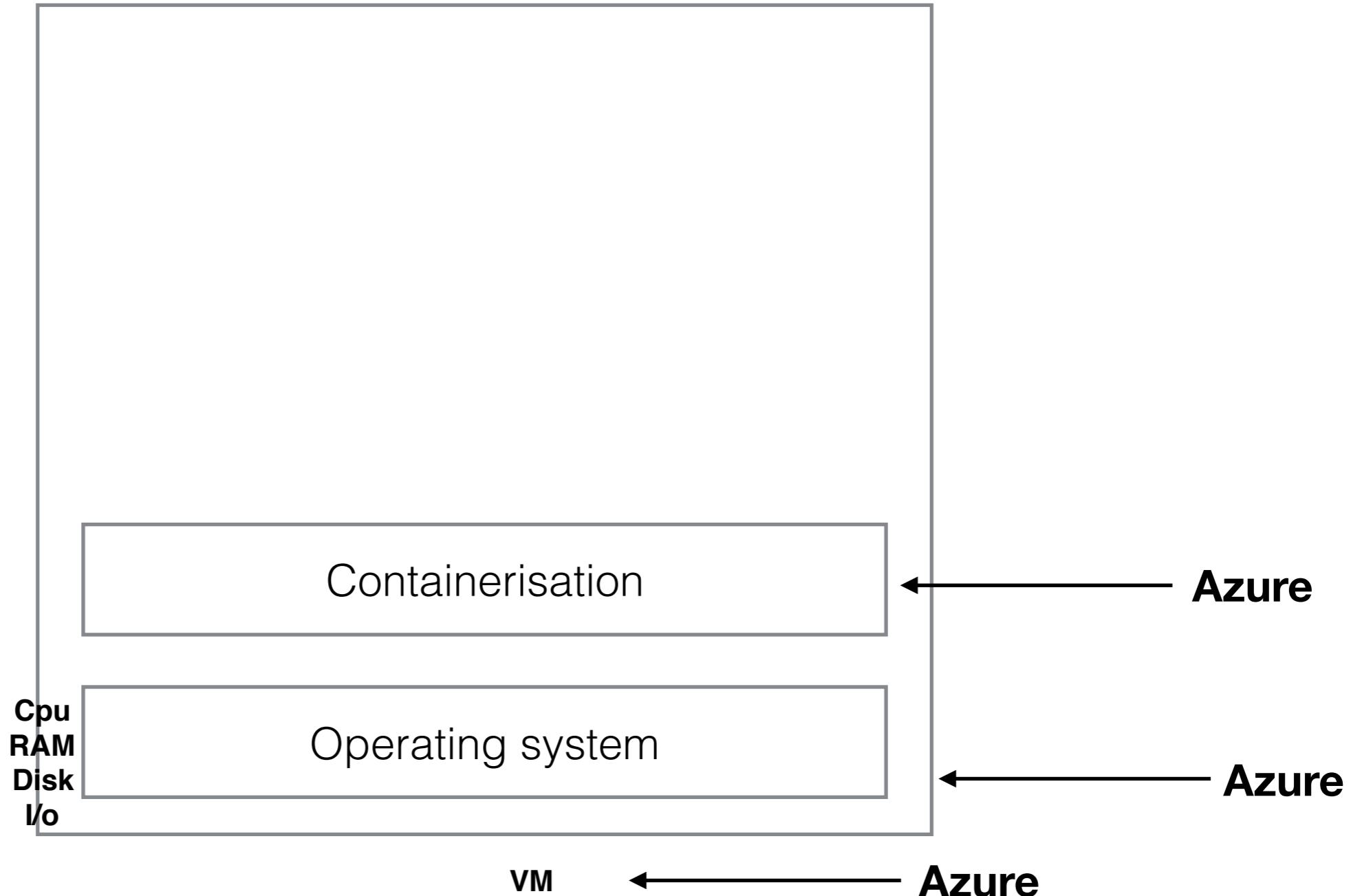
# Deployment View

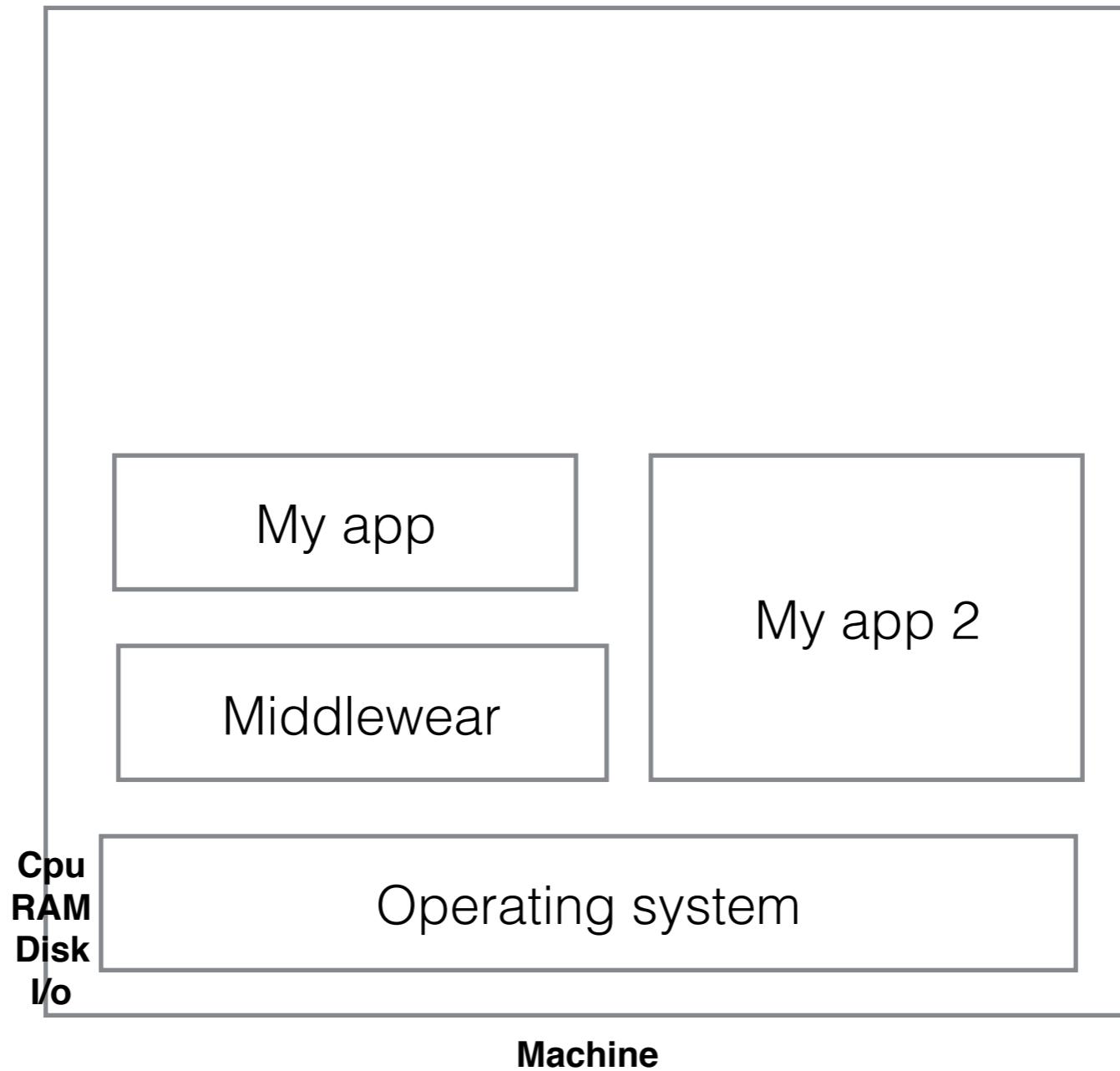


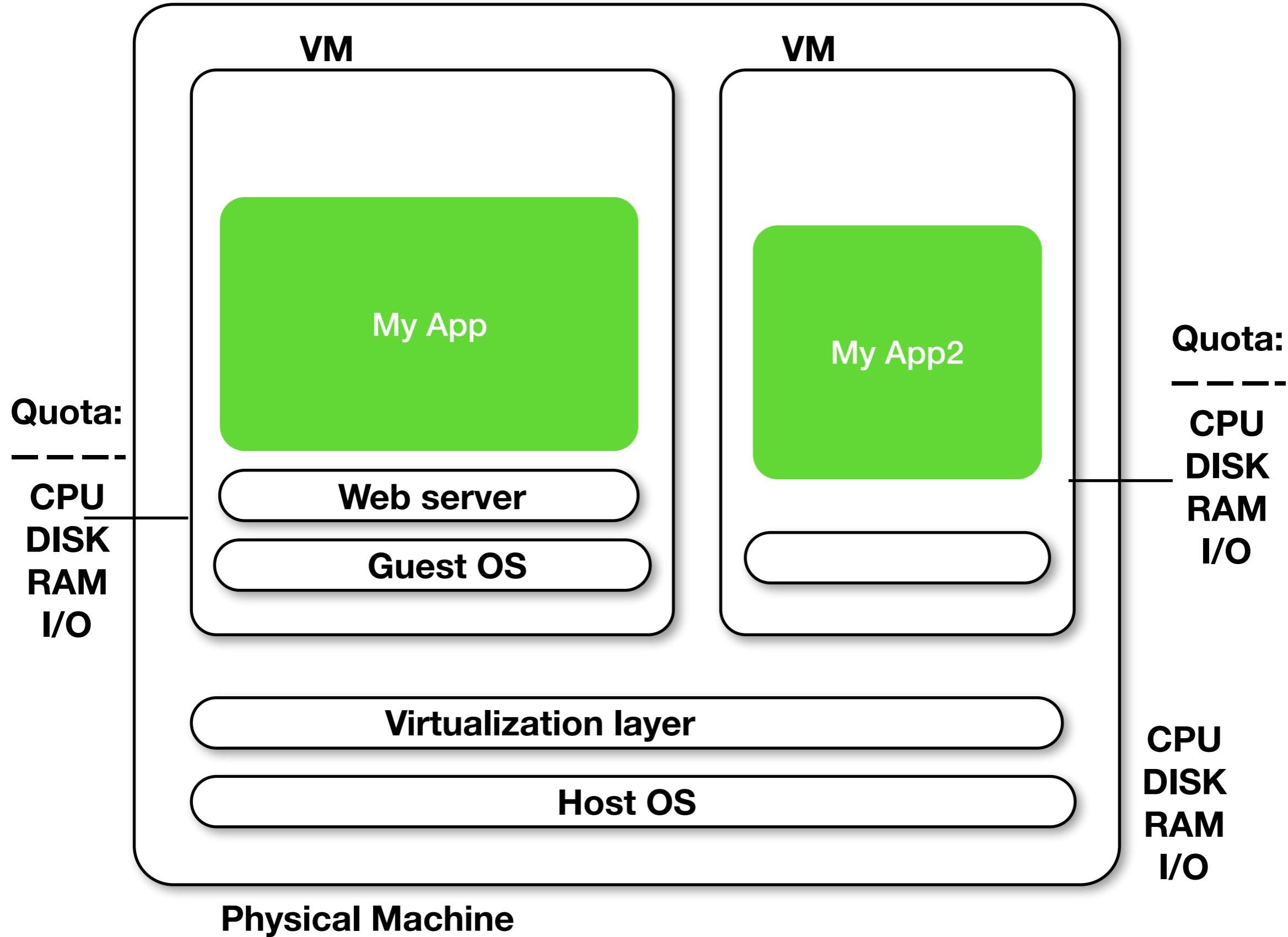


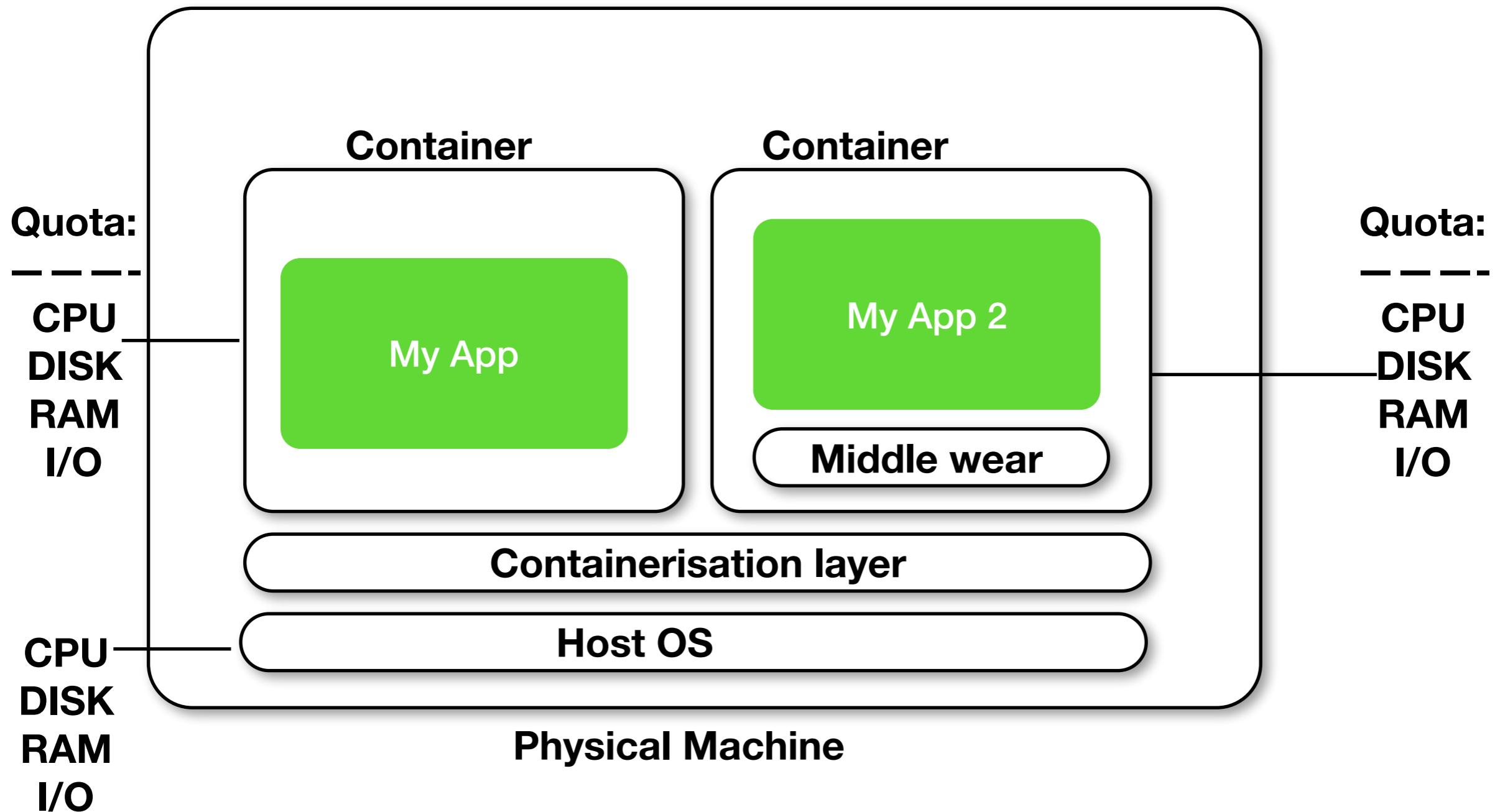
**Registry  
OS API  
File System  
Network Calls**

...

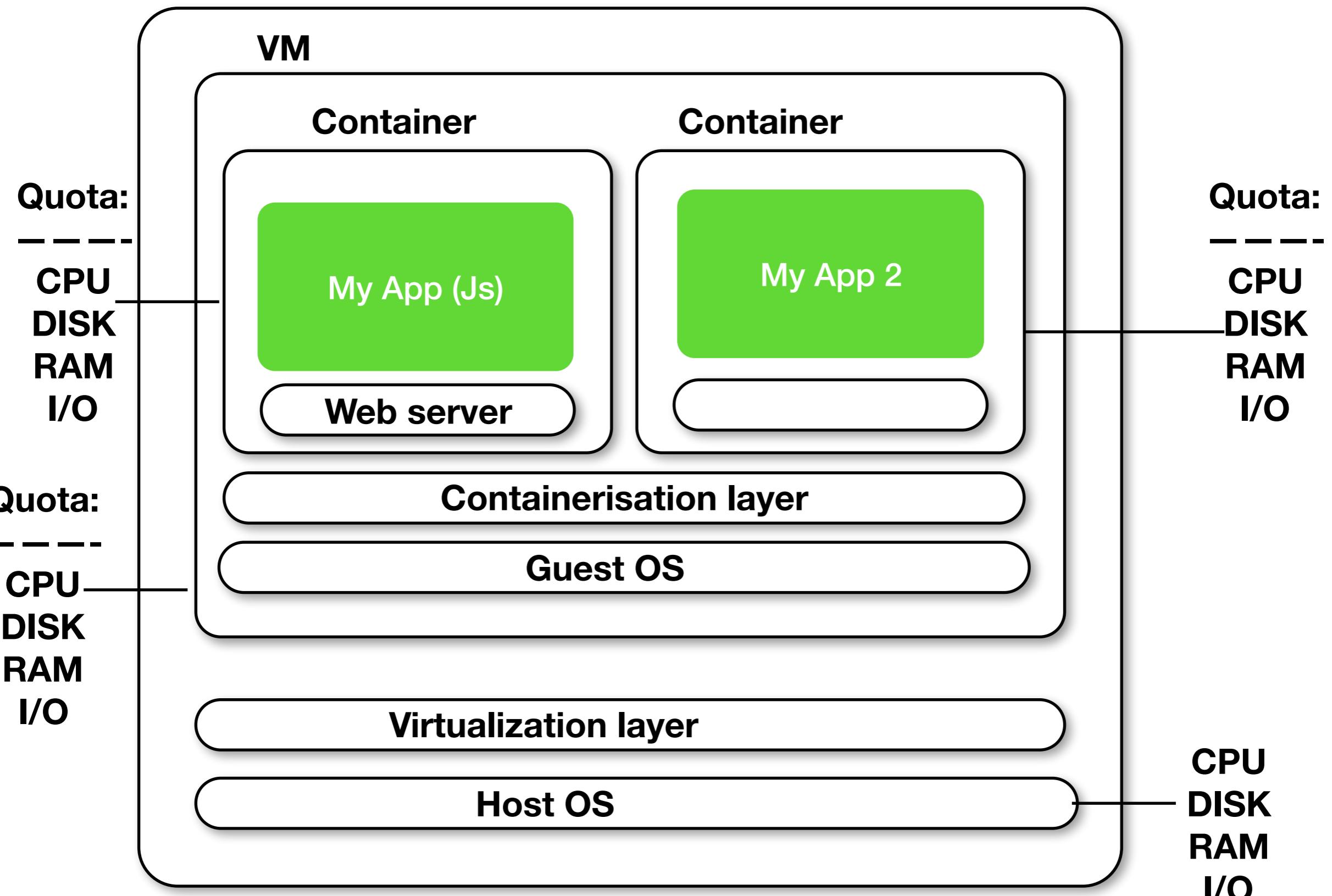


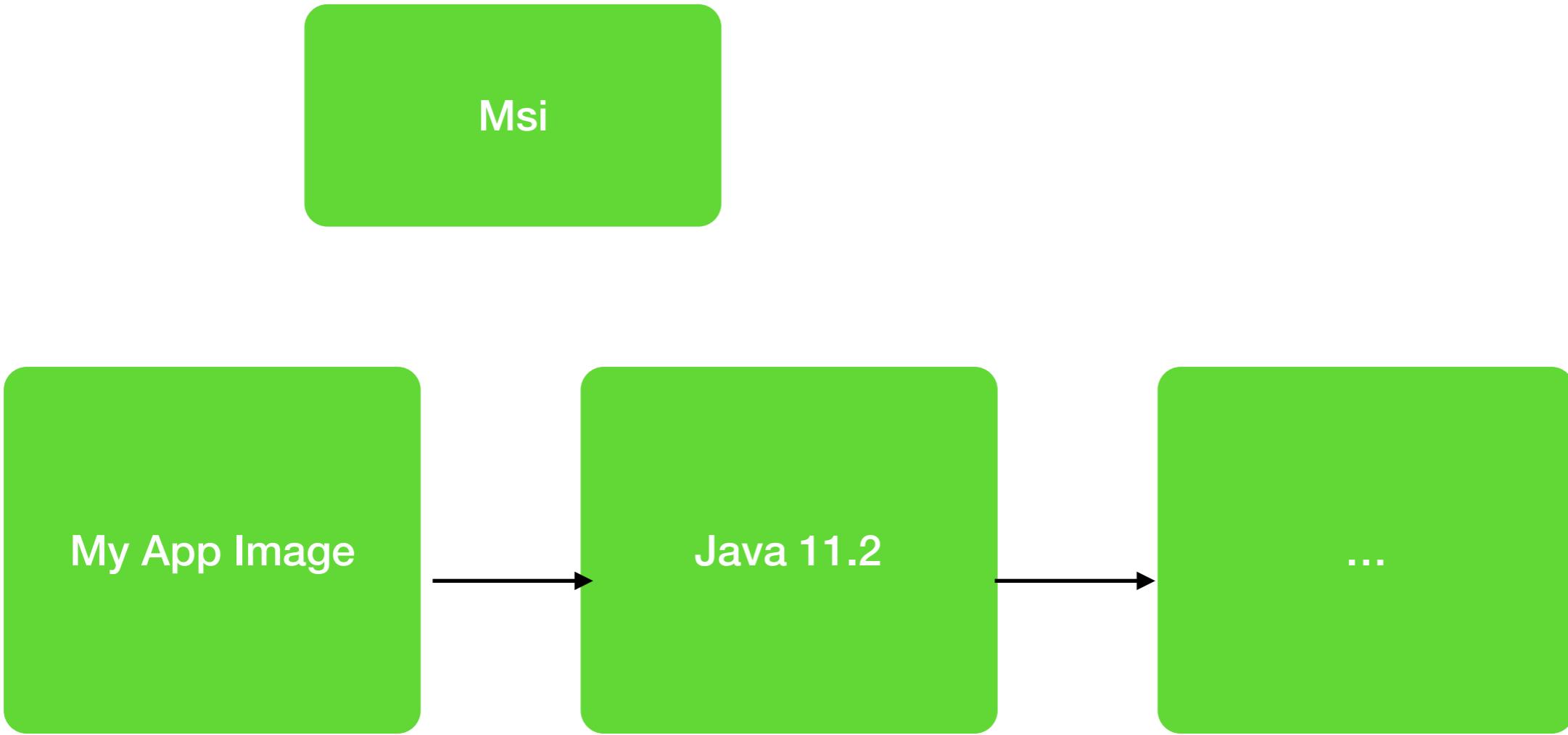






# Physical Machine





Msi

My App Image

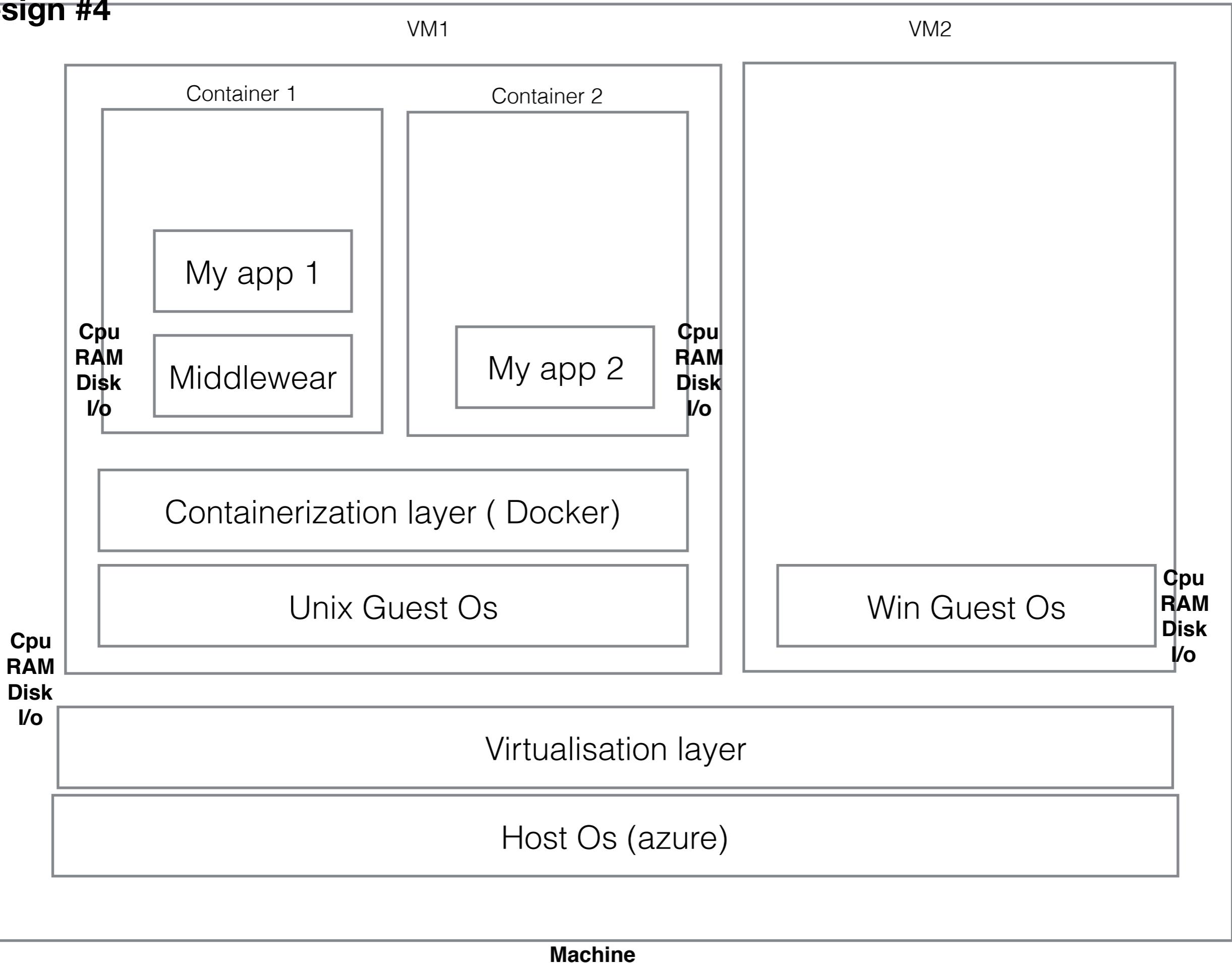
Java 11.2

...

## Design #4

VM1

VM2



VM1

VM2

Container 1

Container 2

Cpu  
RAM  
Disk  
I/o

My app 1

Middlewear

Cpu  
RAM  
Disk  
I/o

My app 2

Containerization layer ( Docker)

Cpu  
RAM  
Disk  
I/o

Unix Guest Os

My app 3

Containerization layer

Cpu  
RAM  
Disk  
I/o

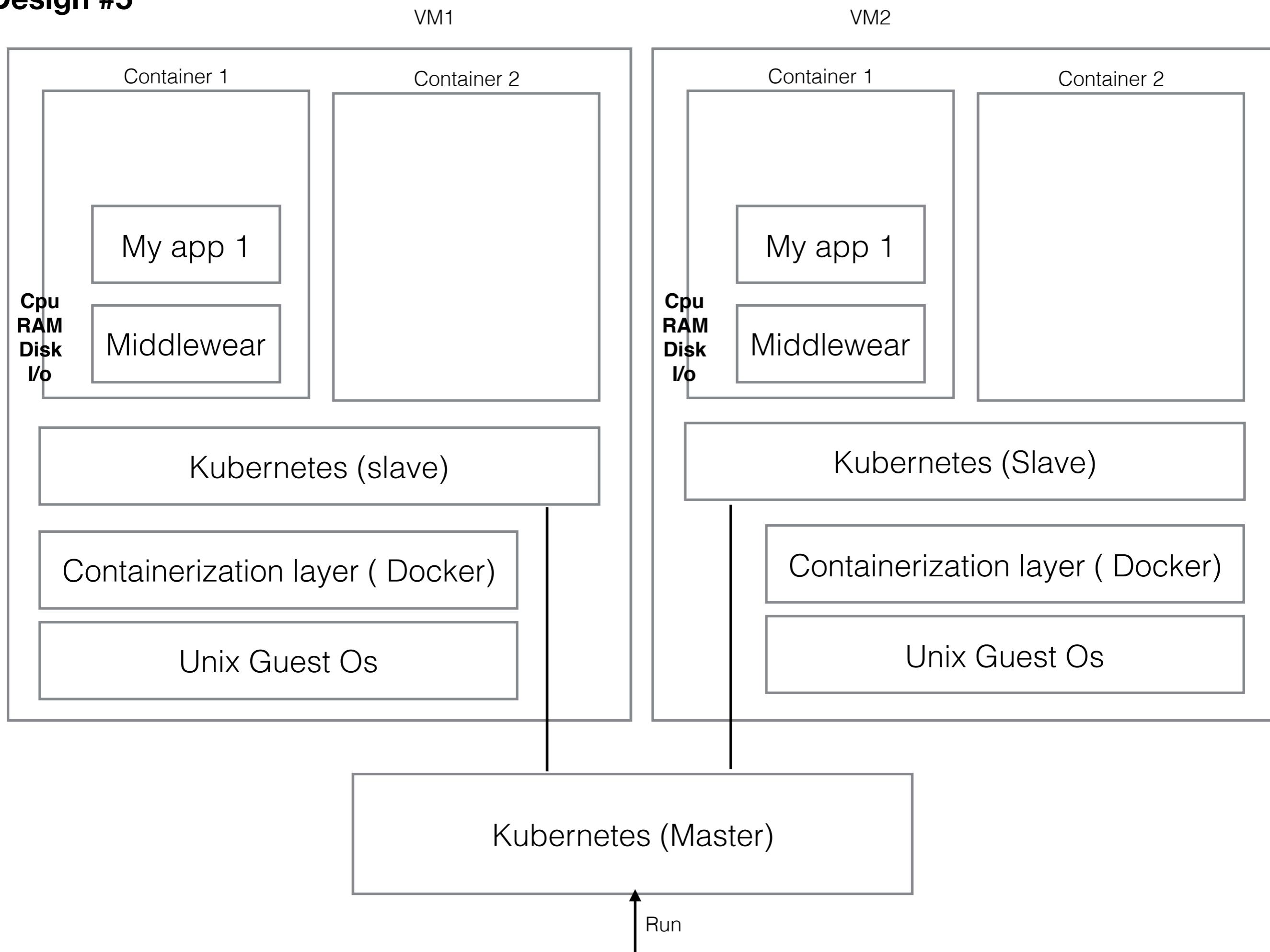
Unix Guest Os

Virtualisation layer

Host Os (azure)

**Machine**

## Design #5



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

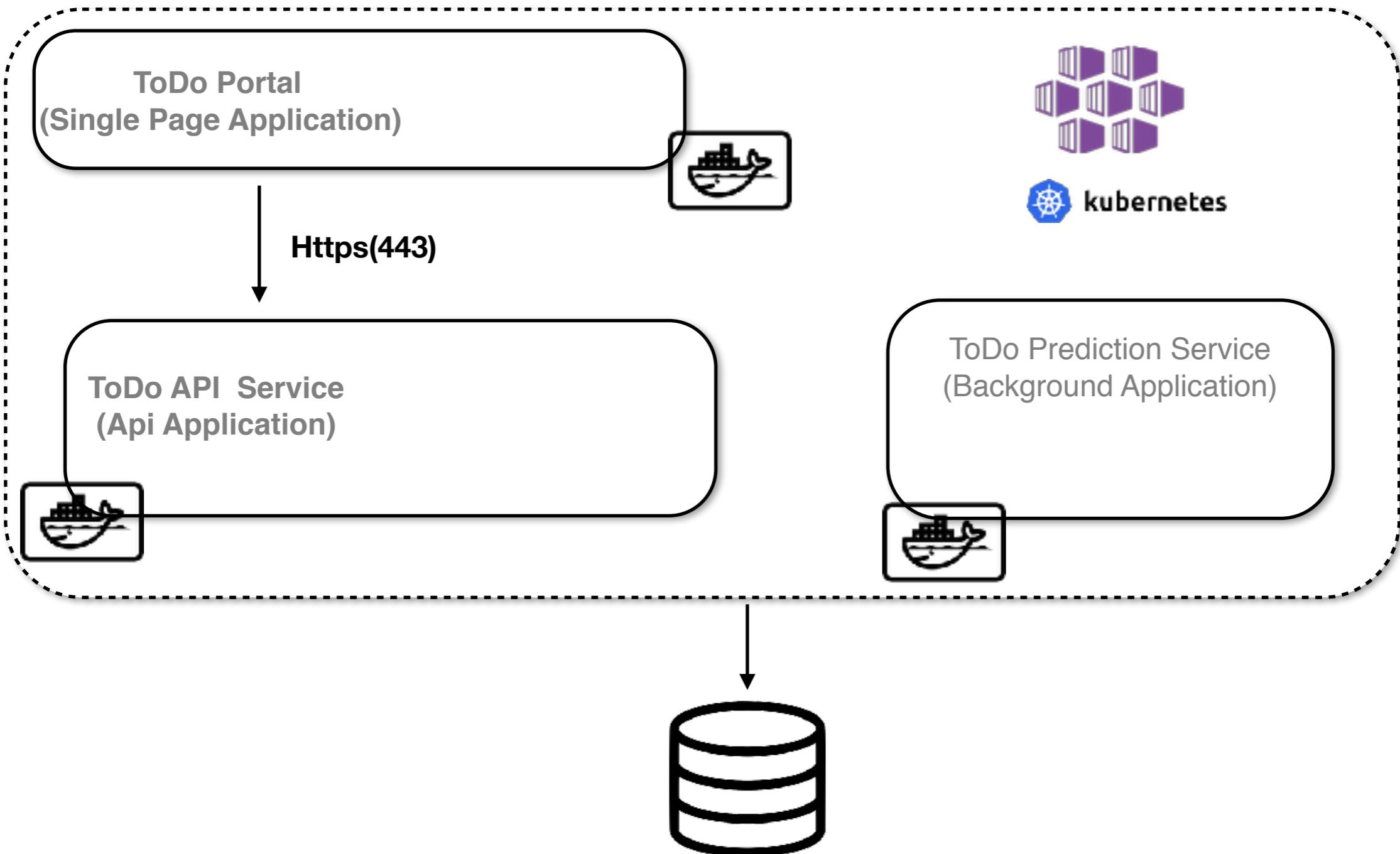
Middlewear

**Container**

Guest OS

**VM**

**Machine**



# System Storage

**Files**

Binary

Audio  
Video  
Images  
Logs

**Database**

Lob

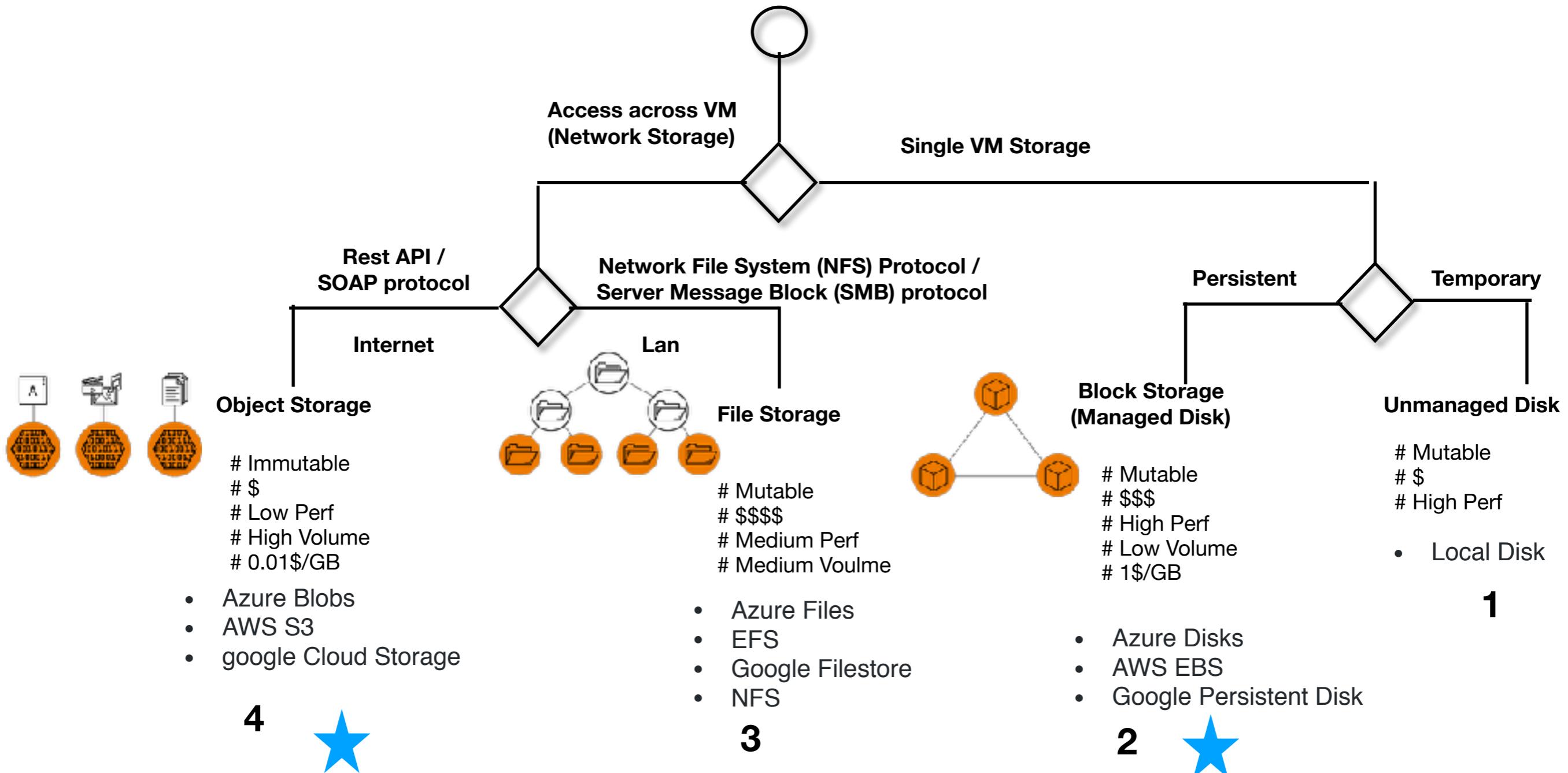
Product  
Sale  
Invoice  
....

**Analytical**

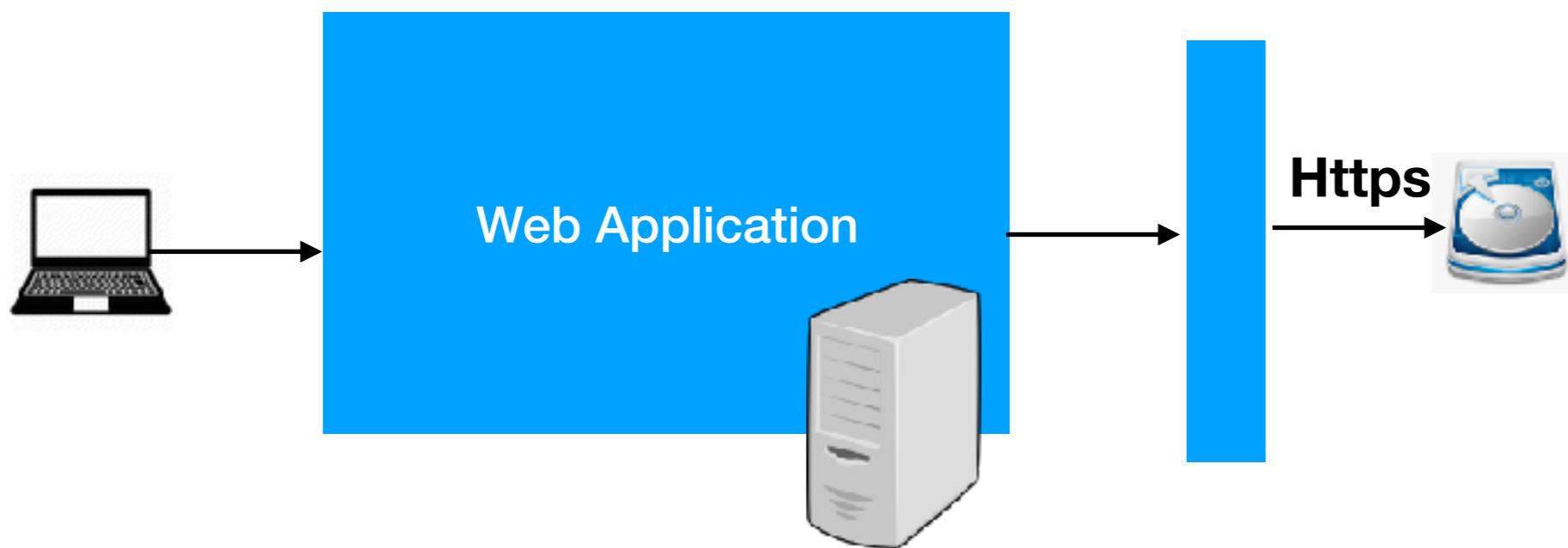
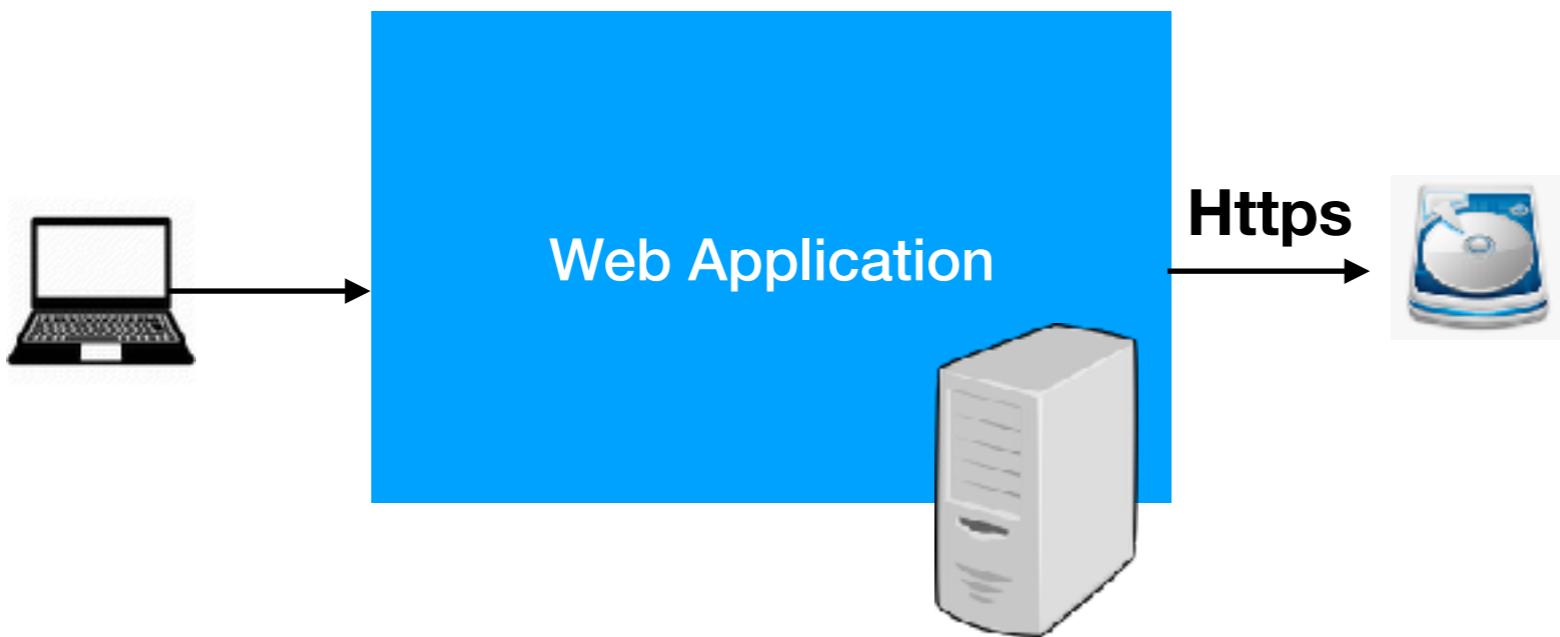
Analytics

Historical /past

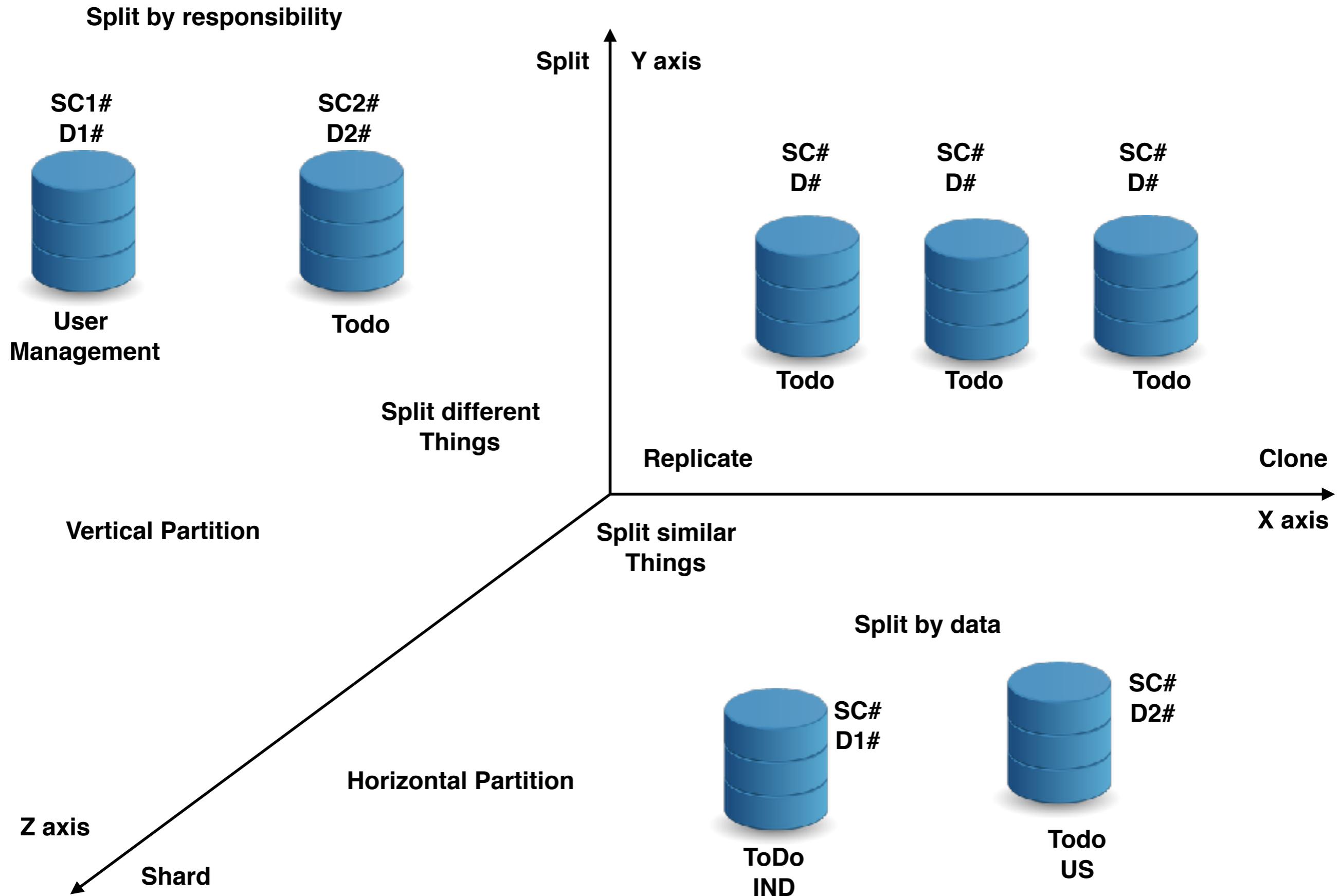
# Binary Storage



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



## Scalability Cube - 50 rules for high Scalability



**Split different  
Things**



**Ecom**

**Vertical Partition**



**User**



**Inventory**



**Accounts**



**Order**

**Horizontal Partition**



**Shard**



**Inventory:**  
**M**



**Order:**  
**AUS**



**Order:**  
**USD**



**Inventory:** **Inventory:**  
**M** **M**

**Clone**

# LOB

Structured

[ACID - transaction ]

RDBMS (OLTP)

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



1

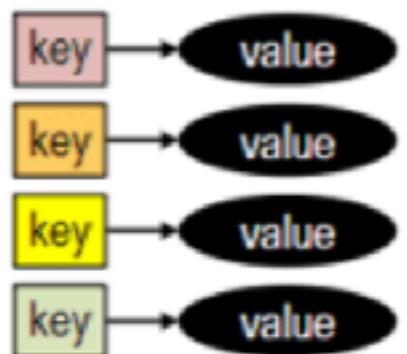
Dictionary  
(key value)

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

Semi Structured

[\* BASE,  
SCALE- shared,  
\* Flexible Schema]

Nested  
Objects

Document

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

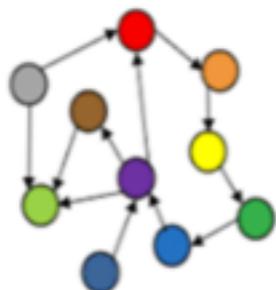


3

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
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

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

## 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": "1964-08-30"
  }
]
```

Document 2

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

Document 3

```
[
  {
    "id": "3",
    "name": "Adam Stark",
    "isActive": true,
    "dob": "1987-07-13"
  }
]
```

## Normalized vs. Denormalized

User Table		
User ID	Name	Dob
1	John Smith	8/30/1964

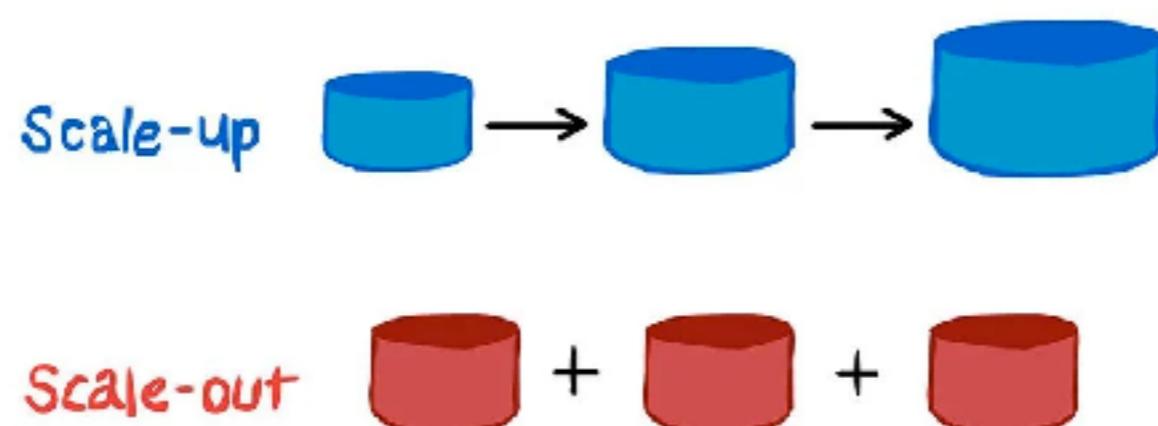
Holdings Table

StockID	User ID	Qty	Symbol
1	1	100	MSFT
2	1	75	WMT

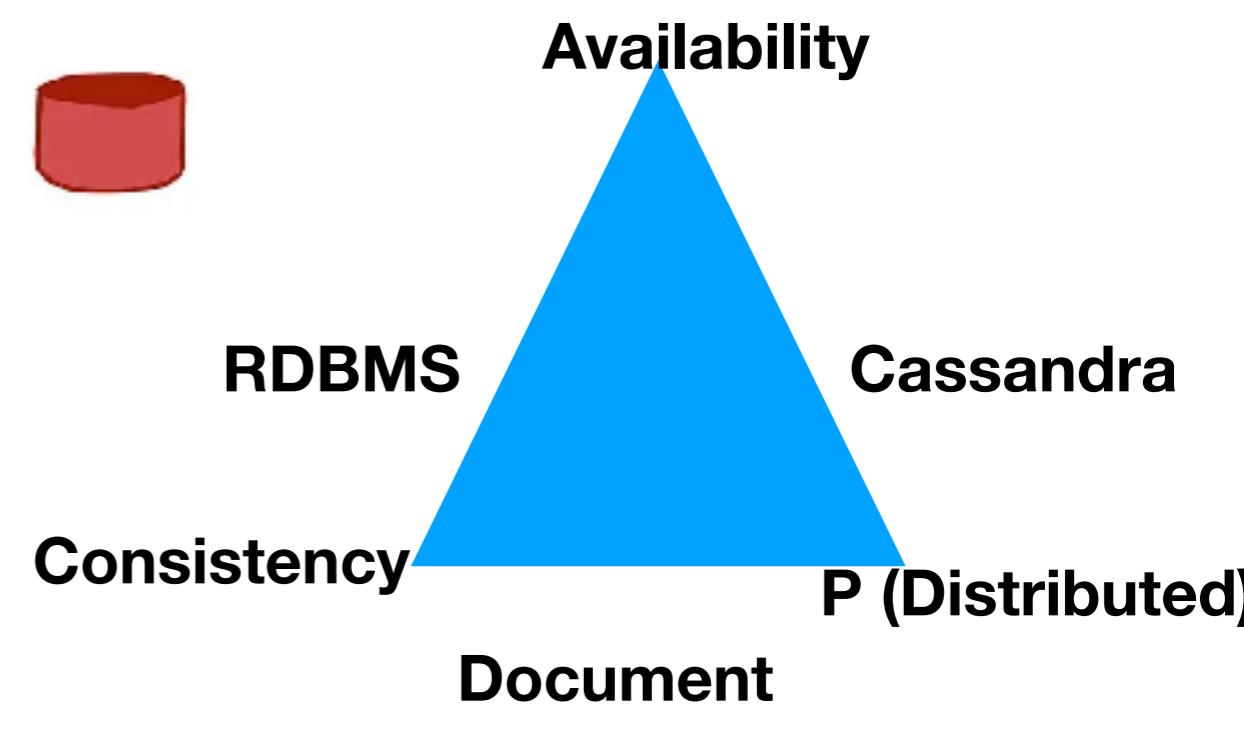
Document

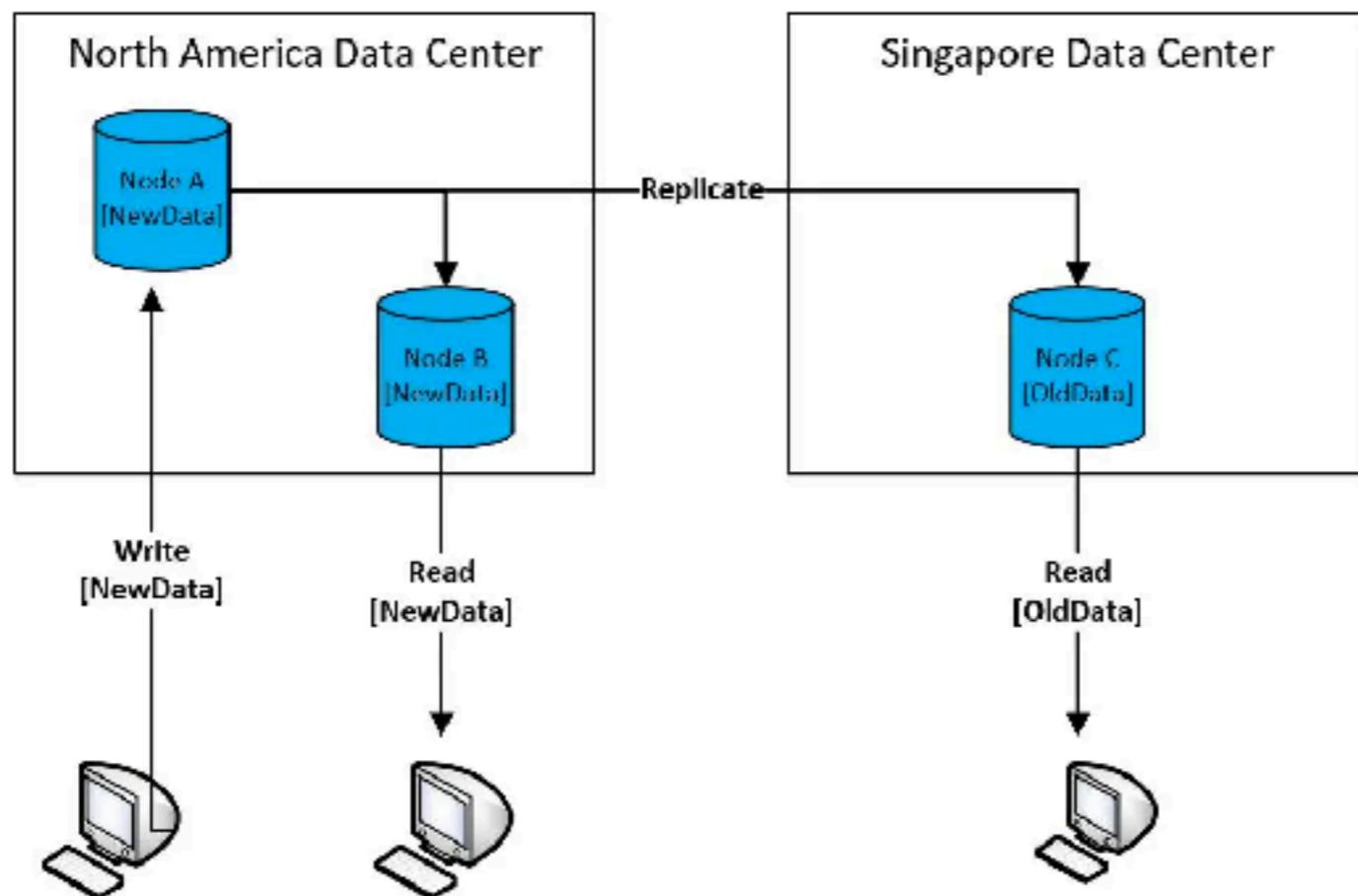
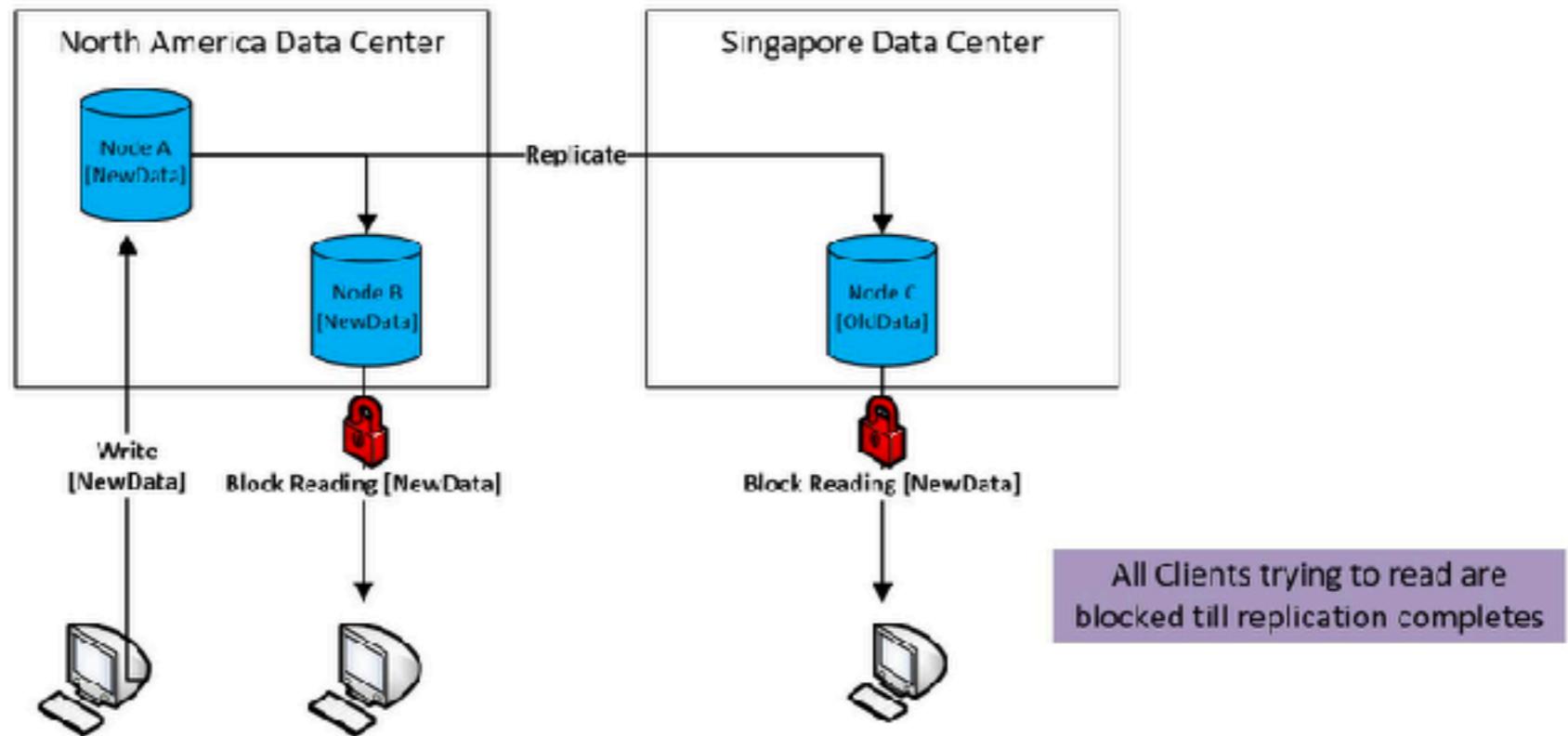
```
{  
  "id": "1",  
  "name": "John Smith",  
  "dob": "1964-30-08",  
  "holdings": [  
    { "qty": 100, "symbol": "MSFT" },  
    { "qty": 75, "symbol": "WMT" }  
  ]  
}
```

## Scale-Up vs. Scale-Out



## Strong Consistency vs. Eventual Consistency





# 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*

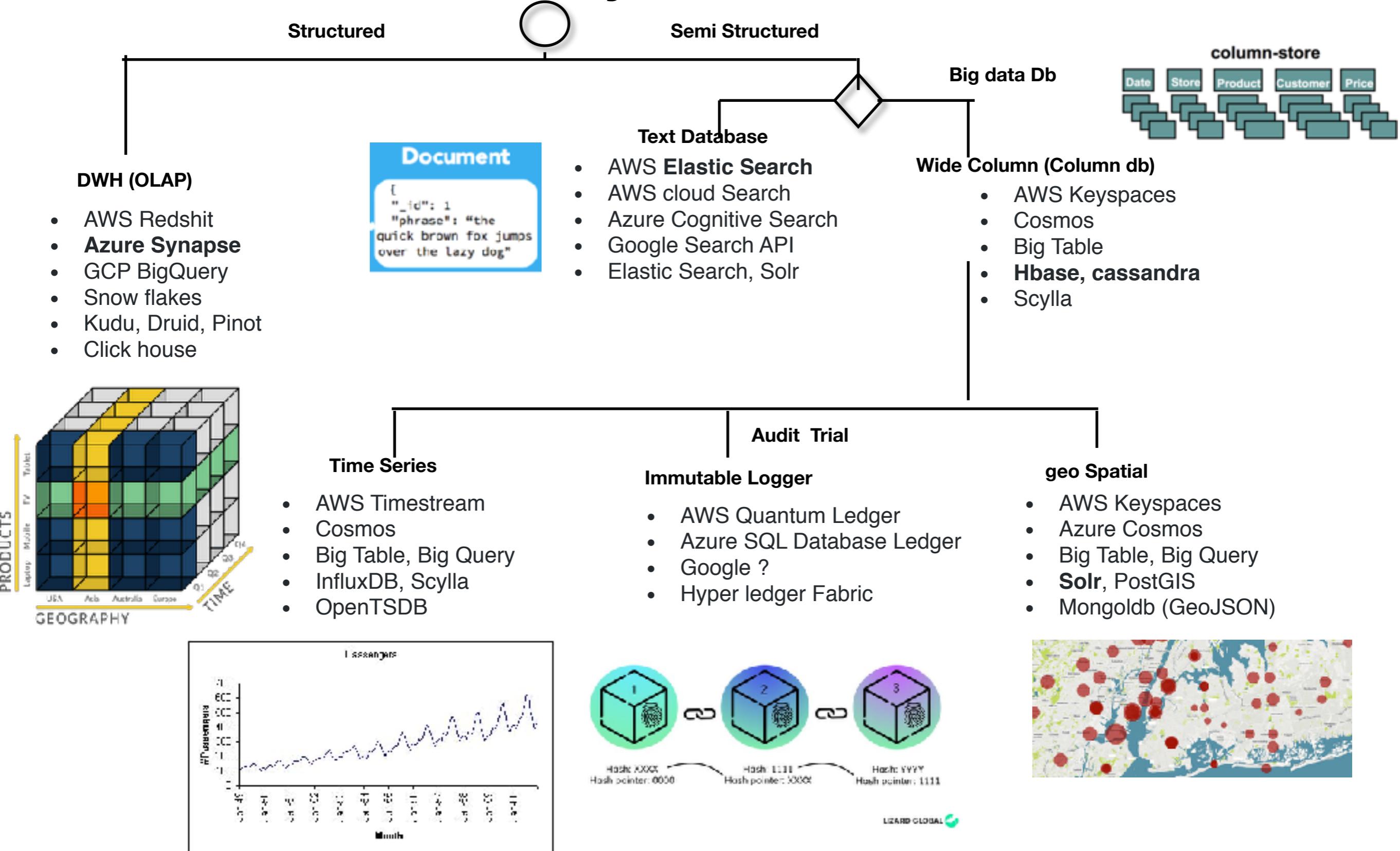
**SQL Server**

*Yahoo*

**PostgreSQL**

	<ul style="list-style-type: none"><li>○ Twitter uses Redis to deliver <a href="#">your Twitter timeline</a></li></ul>
Key Value	<ul style="list-style-type: none"><li>○ Pinterest uses Redis to store lists of users, followers, unfollowers, boards, <a href="#">and more</a></li><li>○ <a href="#">Coinbase</a> uses Redis to enforce rate limits and guarantee correctness of Bitcoin transactions</li></ul>
Graph	<ul style="list-style-type: none"><li>○ <a href="#">Quora</a> uses Memcached to cache results from slower, persistent databases</li><li>○ <a href="#">Walmart</a> uses Neo4j to provide customers personalized, relevant product recommendations and promotions</li></ul>
Document	<ul style="list-style-type: none"><li>○ <a href="#">Medium</a> uses Neo4j to build their social graph to enhance content personalization</li><li>○ <a href="#">Cisco</a> uses Neo4j to mine customer support cases to anticipate bugs</li></ul>
	<ul style="list-style-type: none"><li>○ SEGA uses MongoDB for handling 11 million in-game accounts</li><li>○ Cisco moved its VSRM (video session and research manager) platform to Couchbase to <a href="#">achieve greater scalability</a></li></ul>
	<ul style="list-style-type: none"><li>○ Aer Lingus uses MongoDB with <a href="#">Studio 3T</a> to handle ticketing and internal apps</li><li>○ Built on MongoDB, The Weather Channel's iOS and Android apps <a href="#">deliver weather alerts</a> to 40 million users in real-time</li></ul>

# Analytical



## Terms

brown

dog

fat

fox

jump

lazi

over

quick

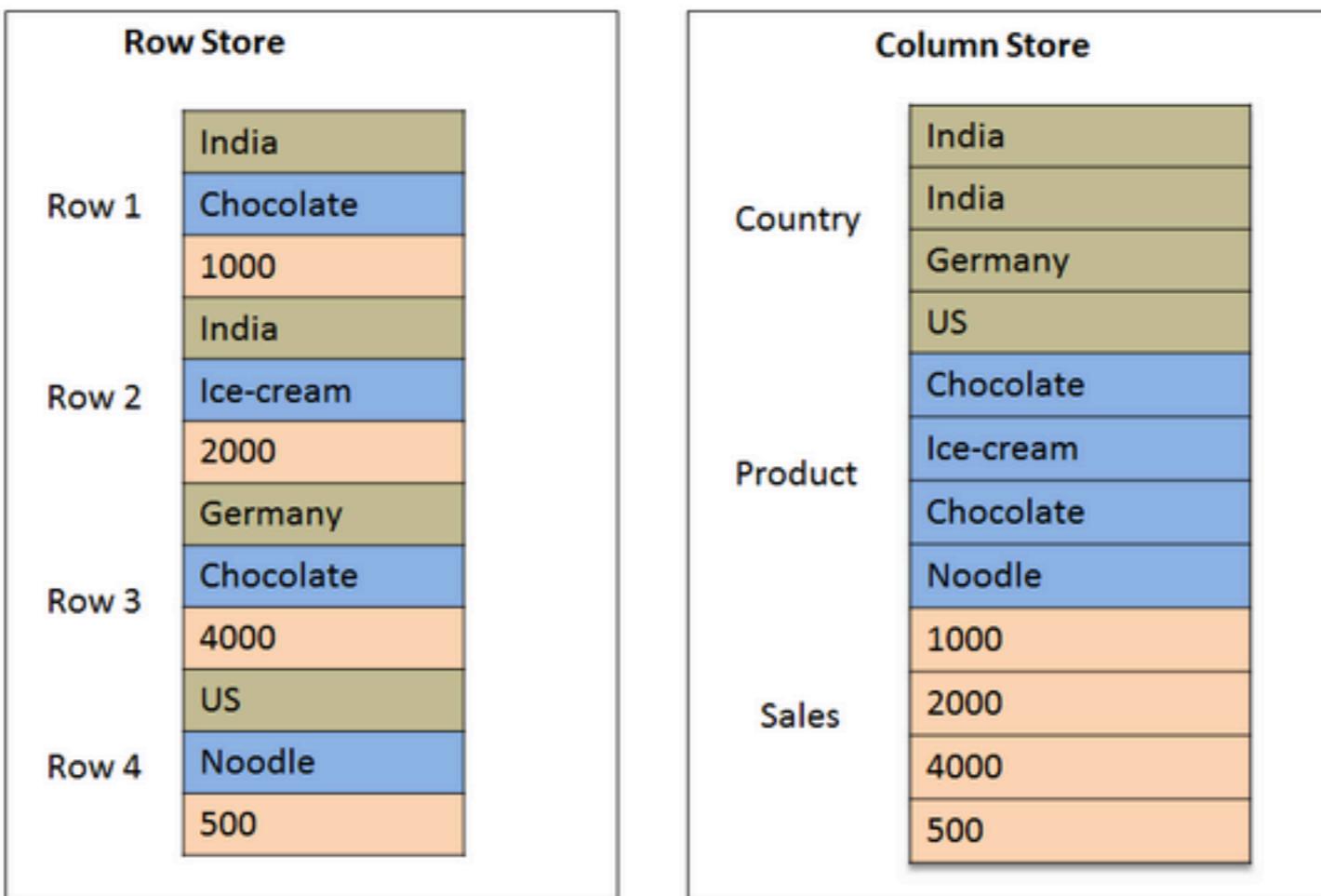
## Document

```
{  
  "_id": 1  
  "phrase": "the  
  quick brown fox jumps  
  over the lazy dog"
```

```
{  
  "_id": 2  
  "phrase": "The fat  
  brown dog"  
}
```

**Table**

	Country	Product	Sales
Row 1	India	Chocolate	1000
Row 2	India	Ice-cream	2000
Row 3	Germany	Chocolate	4000
Row 4	US	Noodle	500



## Classic Relational Databases

Name	Age	Nickname	Employee
Gianfranco Quilizzoni Founder & CEO	40	Heldi	<input checked="" type="radio"/>
Marco Botton Tuttofare	38		<input checked="" type="checkbox"/>
Mariah MacLachlan Better Half	41	Potato	<input type="checkbox"/>
Valerie Liberty Head Chef	16	Val	<input checked="" type="checkbox"/>

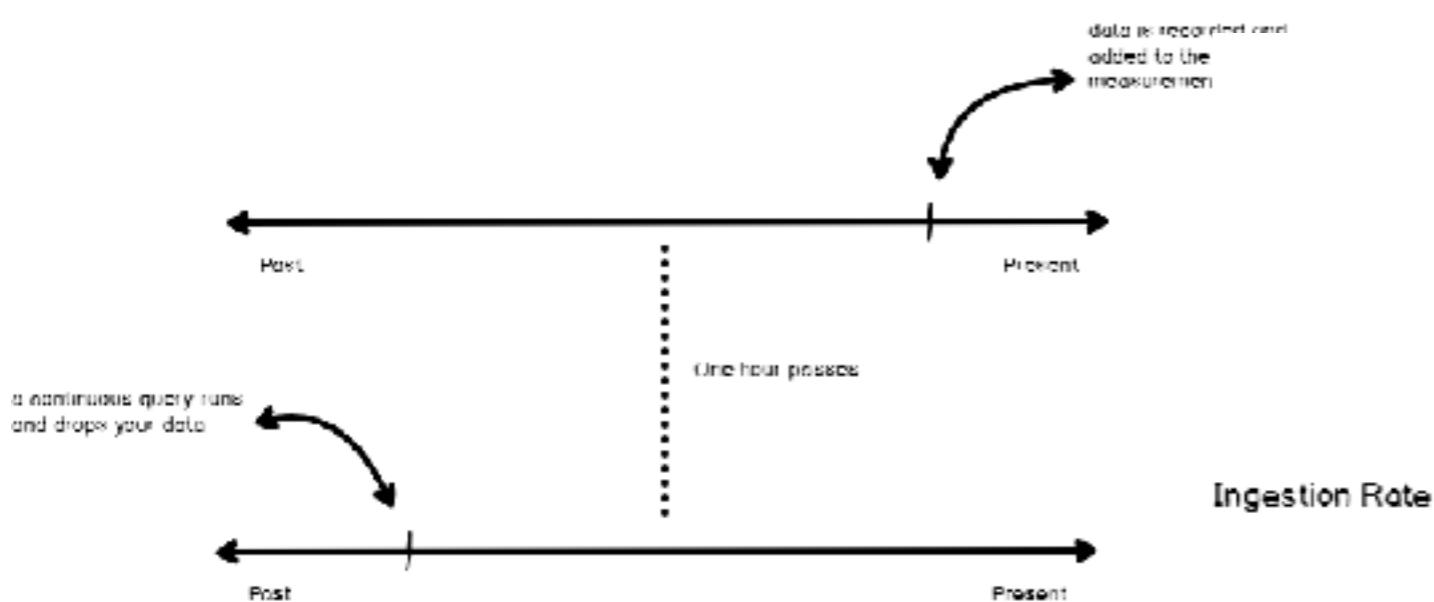
Data are multidimensional

## Time Series Databases

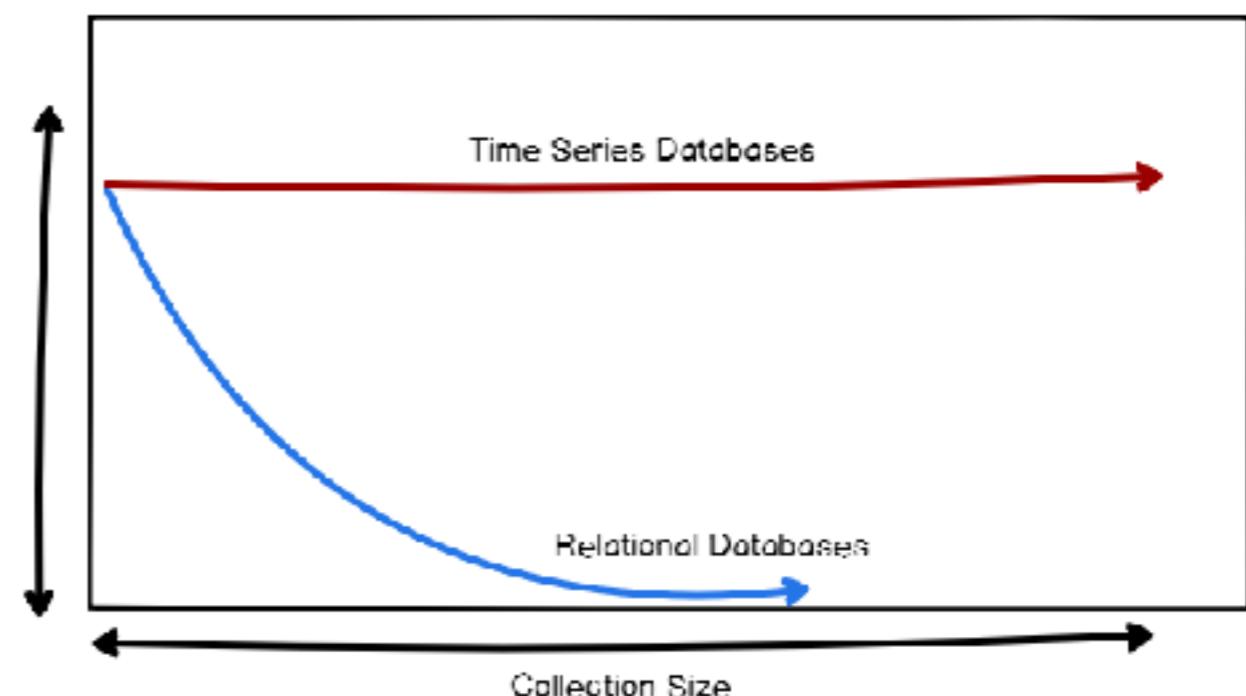
Sensor Temperature	Time
39.6	12/01/19 @ 11:12
11.2	12/01/19 @ 11:13
12.4	14/04/19 @ 12:15
18.5	16/04/19 @ 10:05

Data are aggregated over time

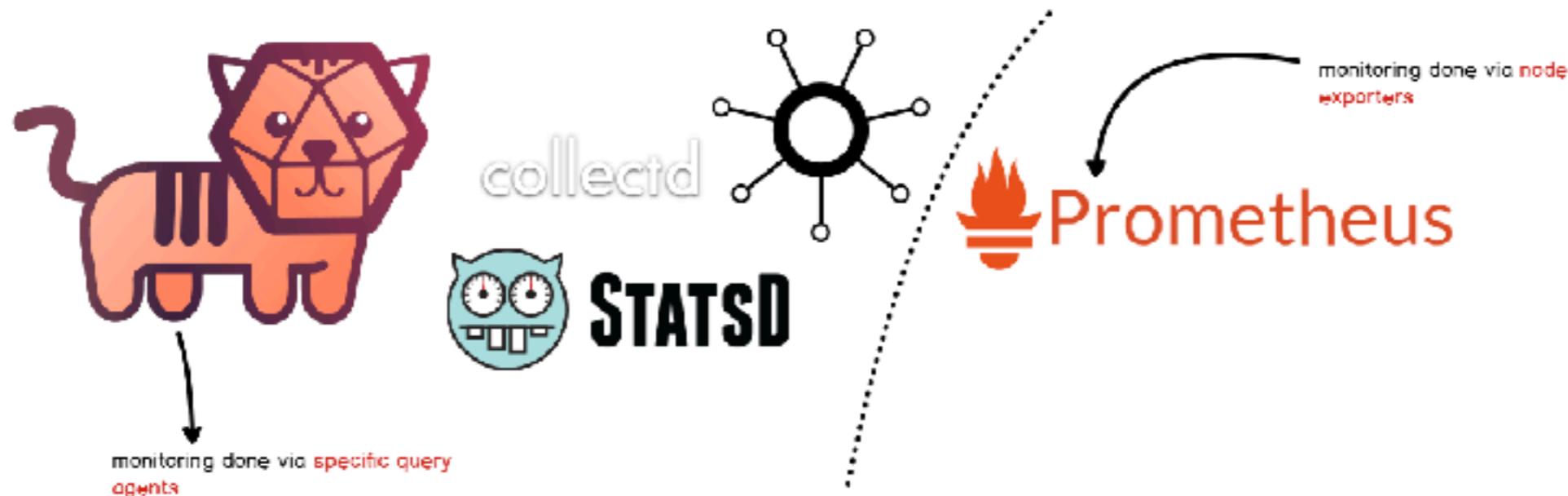
Case : retention policy = 1 hour



## DBMS & TSDB Difference



## Tools that 'produce' TSDB data

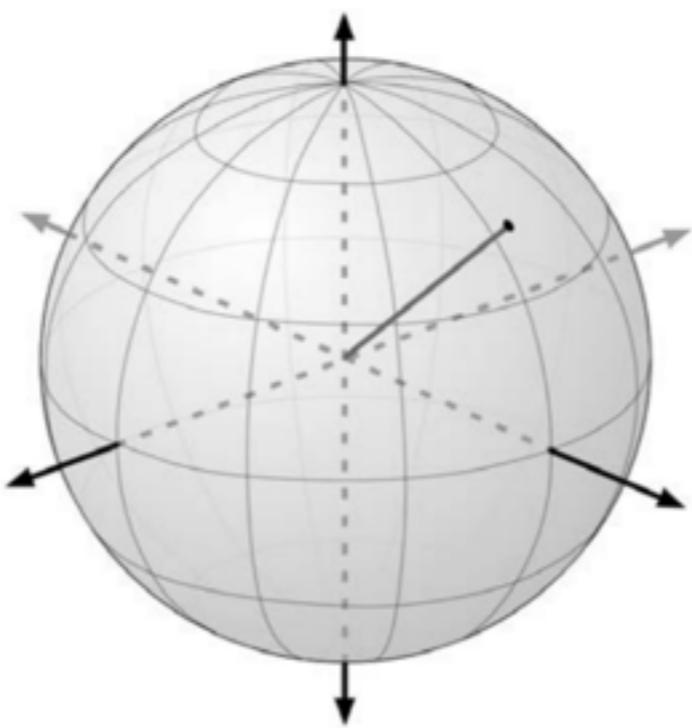


## Tools that 'consume' TSDB



\*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

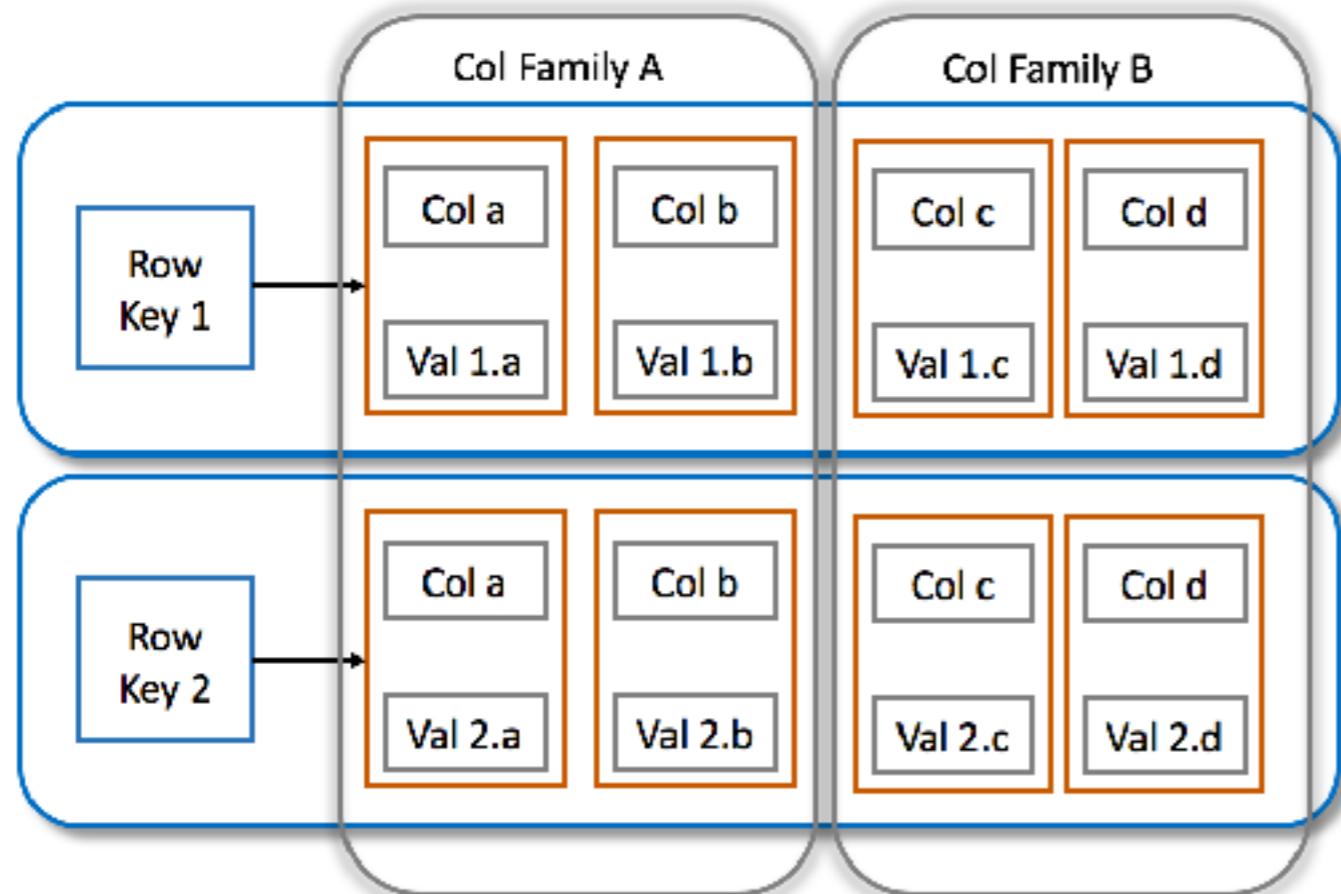
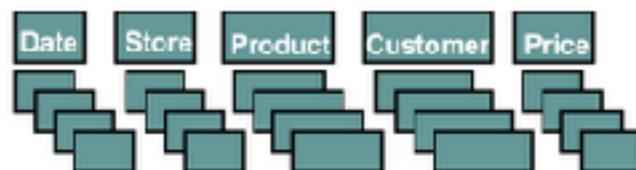
Grade	ID
Senior	001
Freshman	002
Junior	003

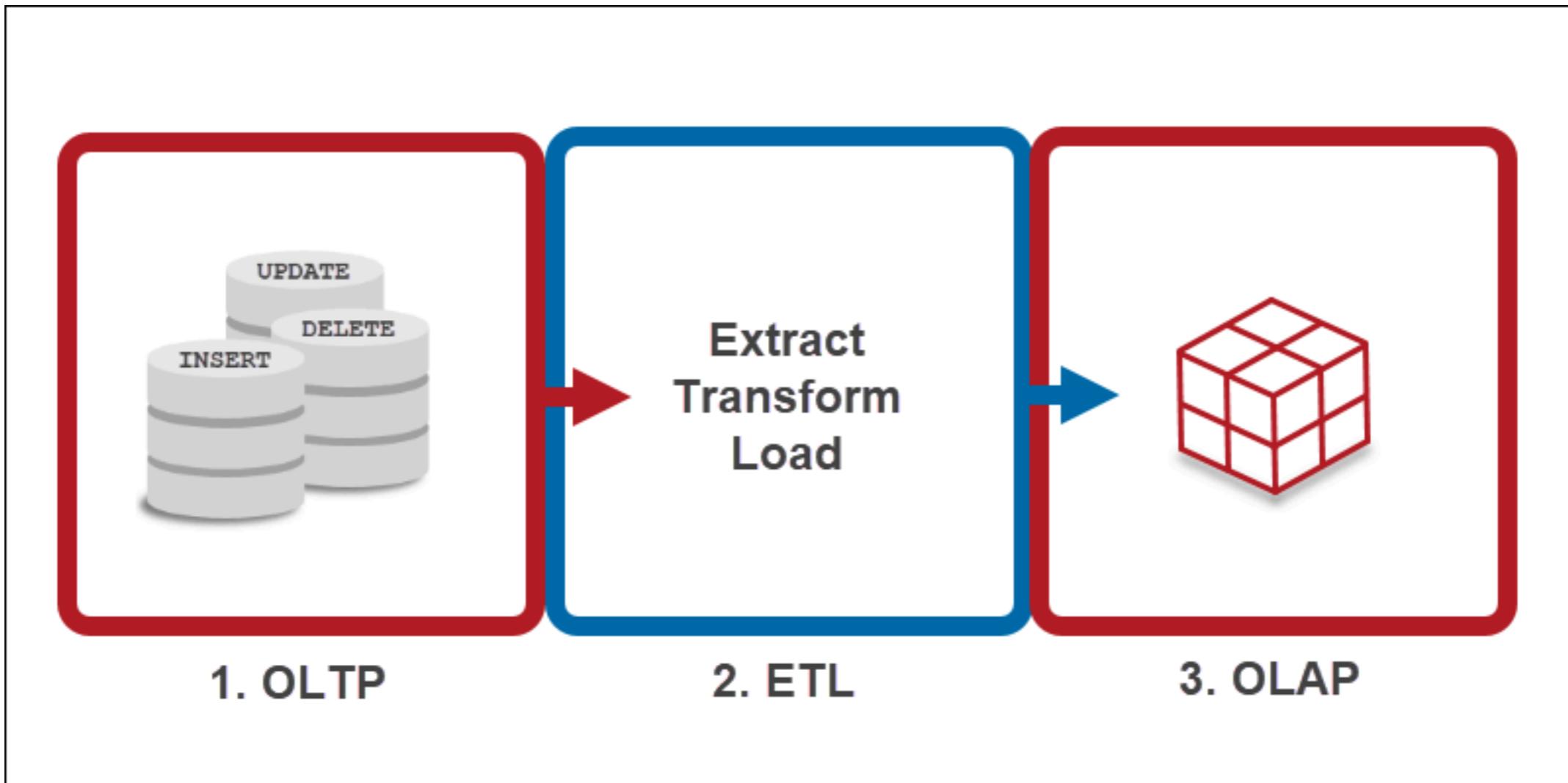
GPA	ID
4.00	001
3.67	002
3.33	003

row-store



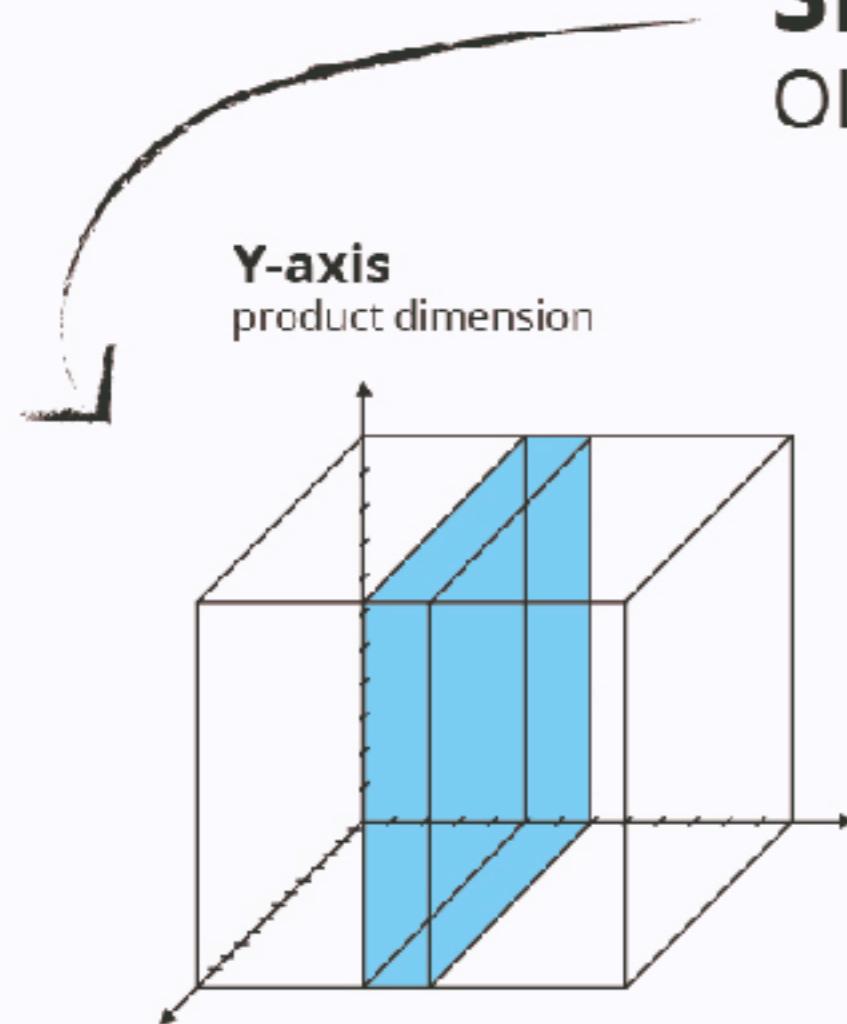
column-store



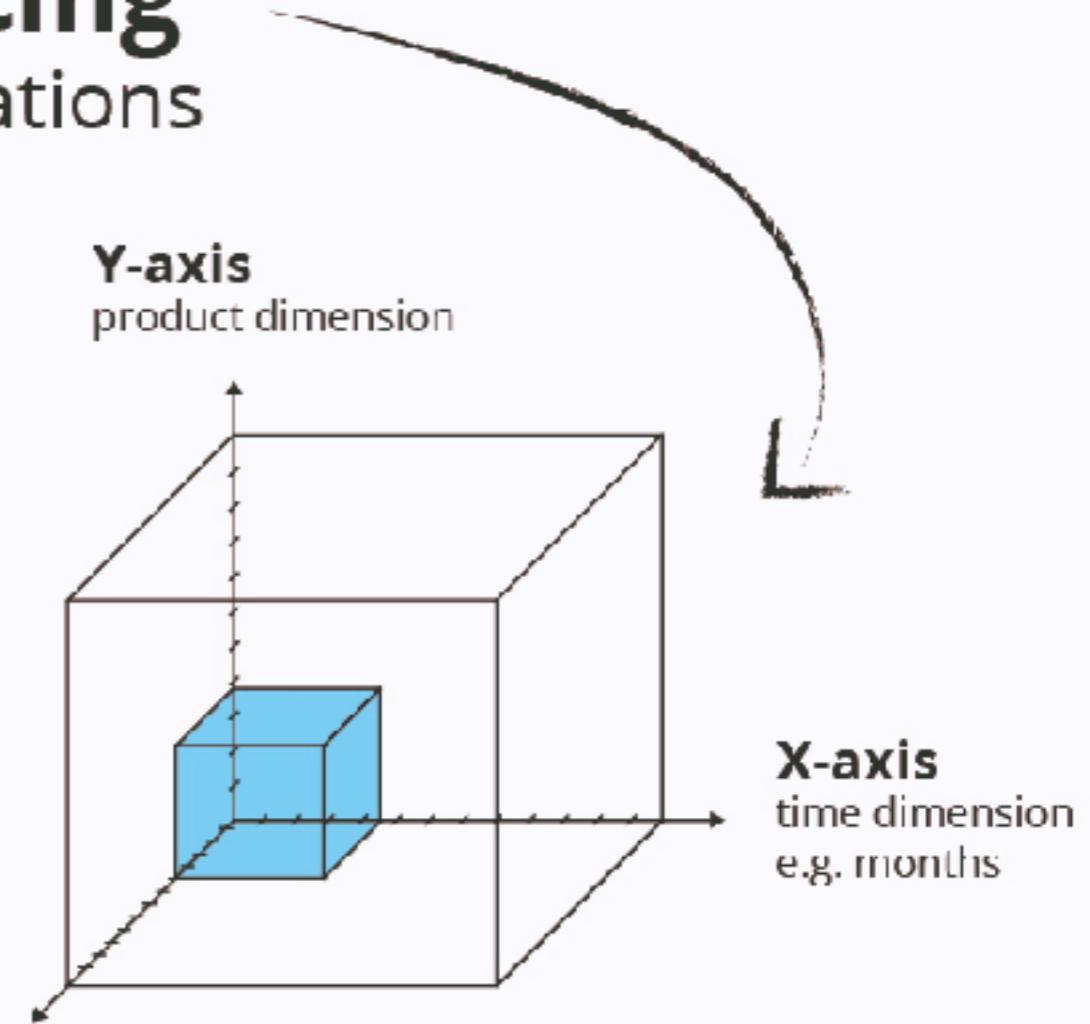


# Slicing & Dicing

OLAP cube operations



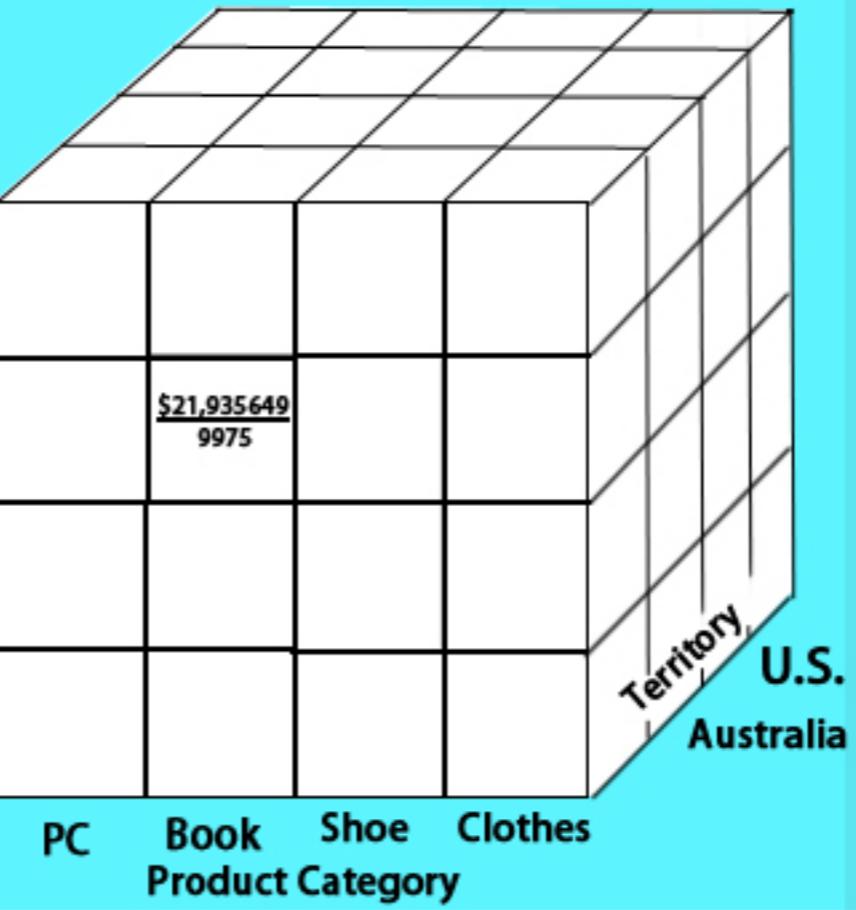
**Z-axis**  
customer dimension

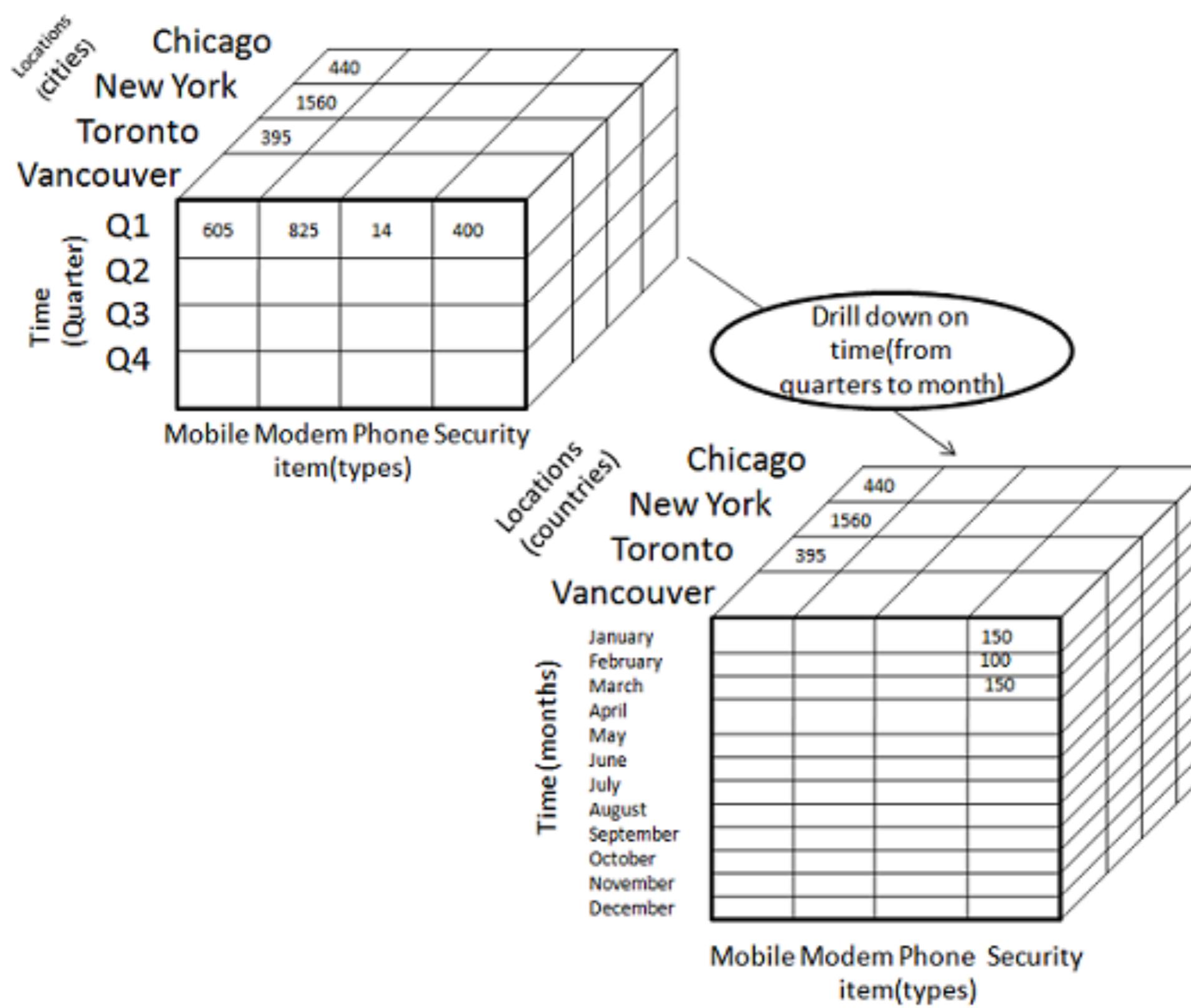


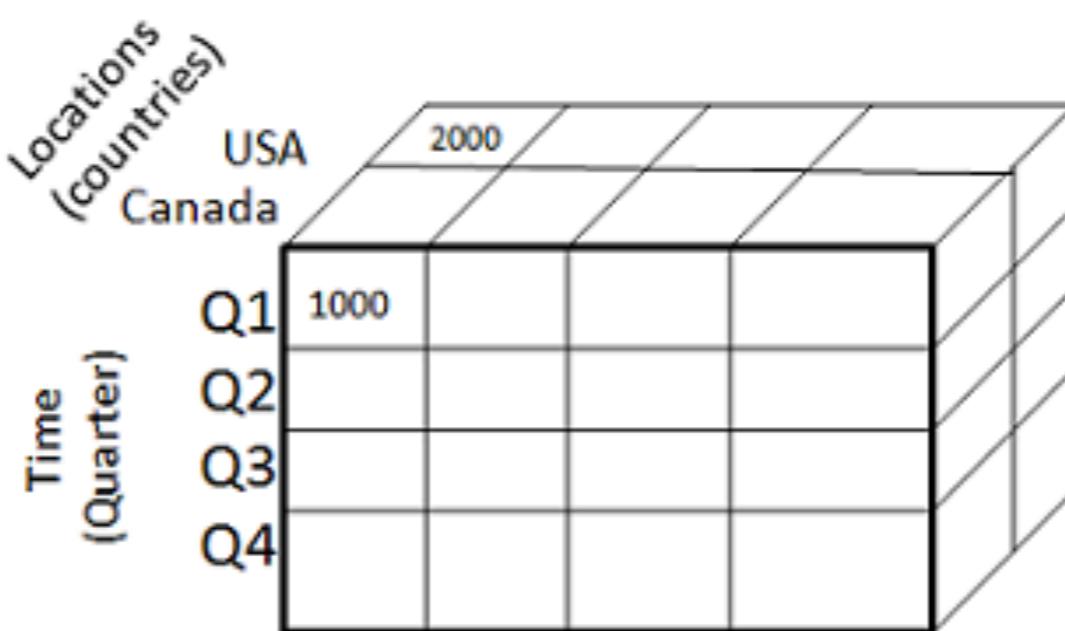
**Z-axis**  
customer dimension

## **OLAP CUBE**

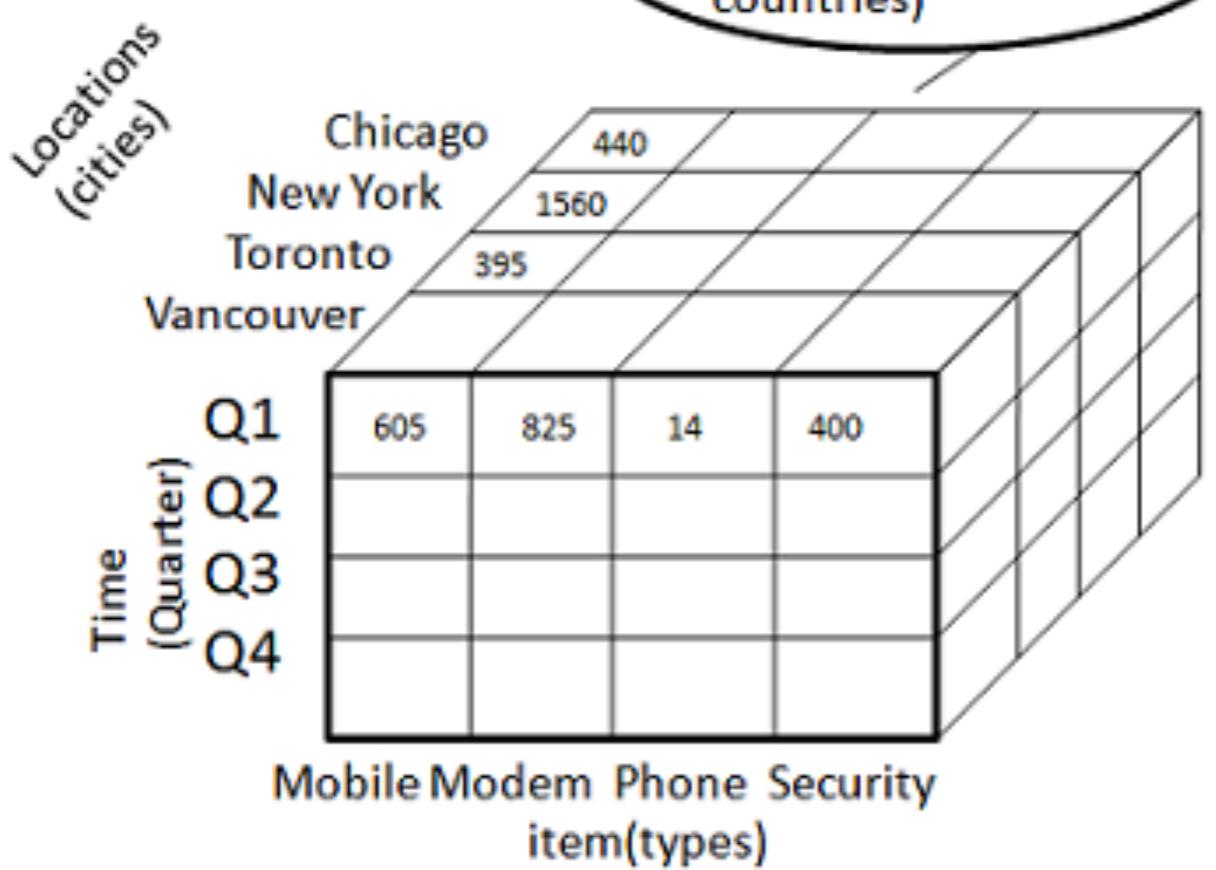
Quarters  
Q1  
Q2  
Q3  
Q4





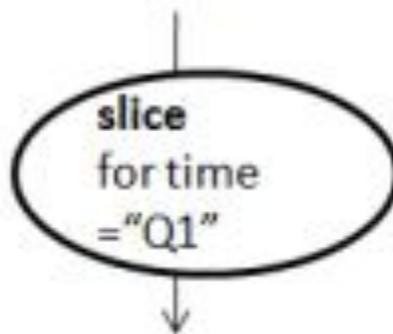


**roll-up** on location  
(from cities to  
countries)



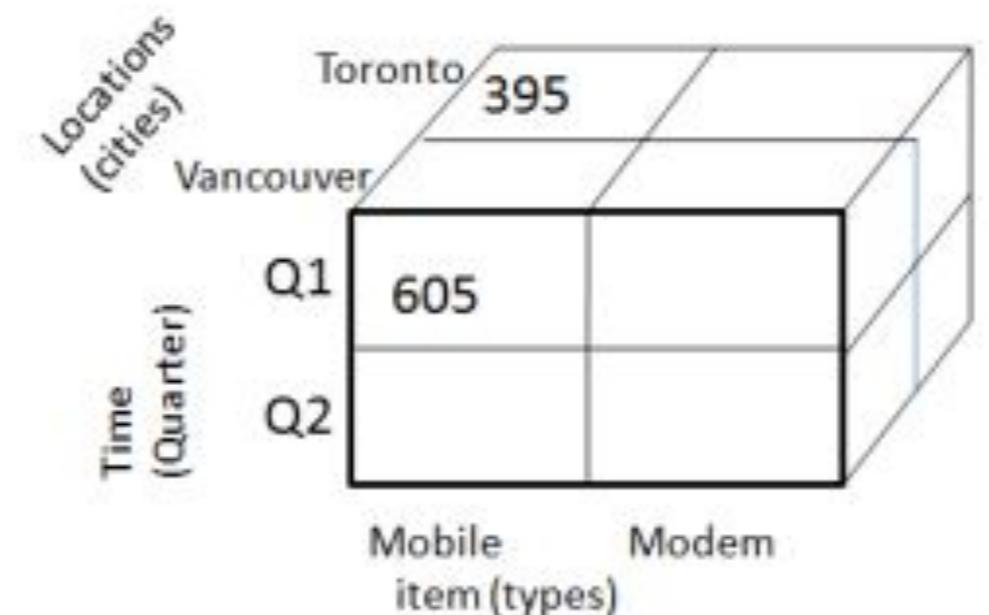
		Mobile Modem Phone Security				
		item(types)		item(types)		
Locations (cities)	Time (Quarter)	Chicago		New York		Toronto
		440		1560		395
		605		825		14
		Q1		400		
		Q2				

Mobile Modem Phone Security  
item(types)



		Mobile Modem Phone Security				
		item(types)		item(types)		
Locations (cities)	Time (Quarter)	Chicago		New York		Toronto
		440		1560		395
		605		825		14
		Q1		400		
		Q2				

Mobile Modem Phone Security  
item(types)



Dice for (location = "Toronto" or  
"Vancouver")  
and (time = "Q1" or "Q2") and  
(item = "Mobile" or "Modem")



The diagram illustrates the Pivot operation on a 4x4 matrix. The original matrix has "Locations (cities)" on the rows and "Item (types)" on the columns.

	605	825	14	400

Item (types)

Mobile Modem Phone Security

Location (Cities)

Chicago New York Toronto Vancouver

Item (types)

Mobile Modem Phone Security

Pivot

Location (Cities)

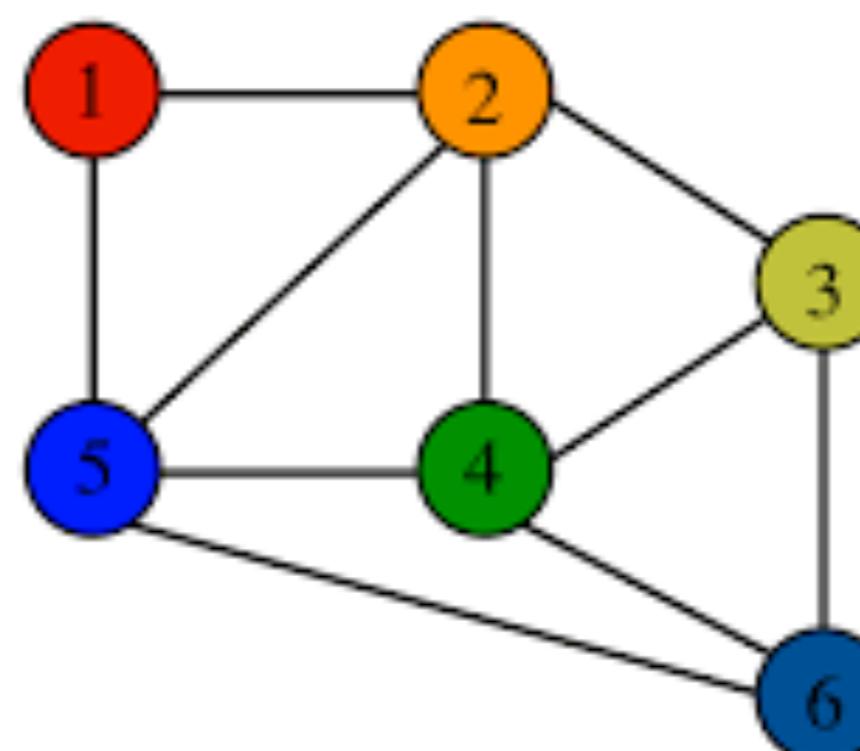
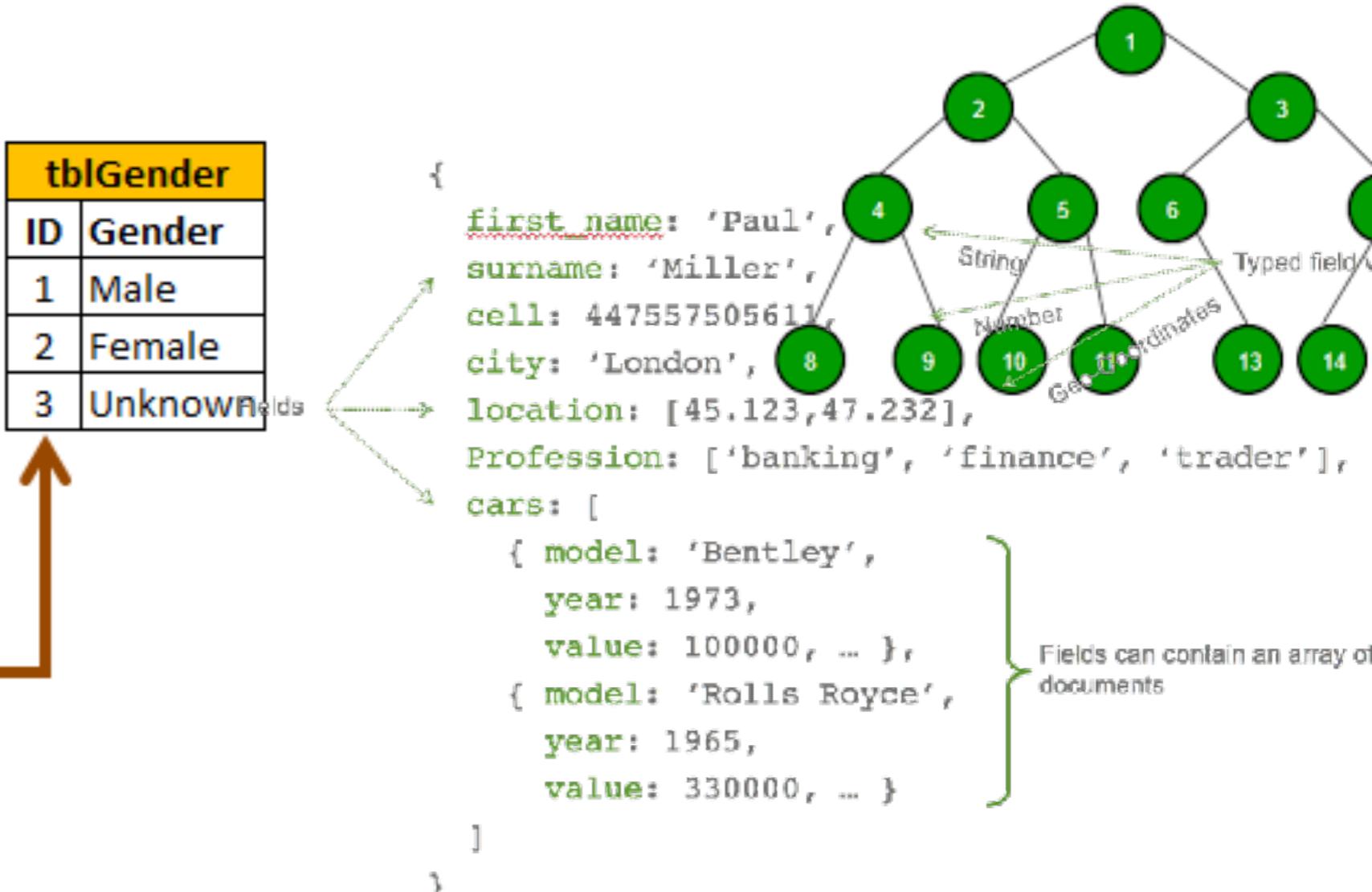
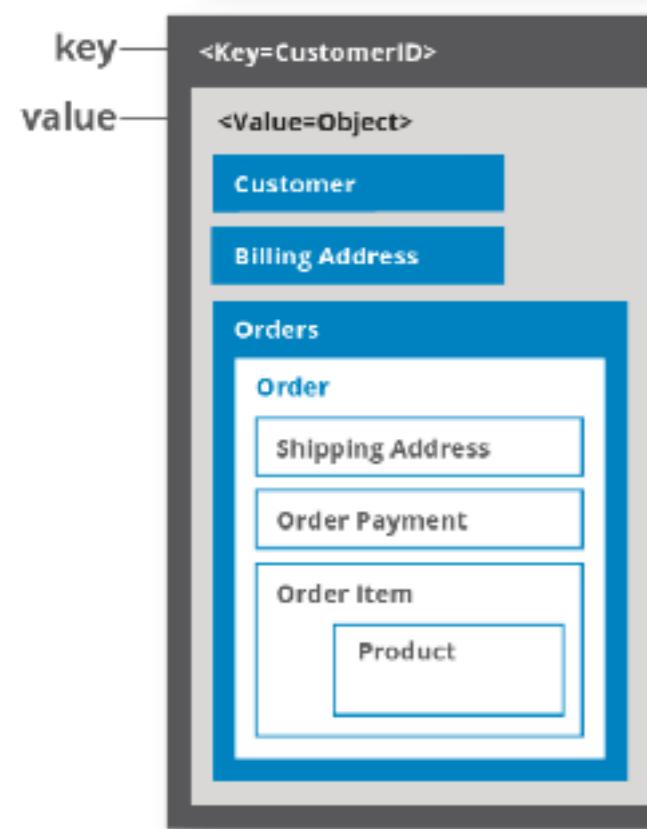
Chicago New York Toronto Vancouver

# 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

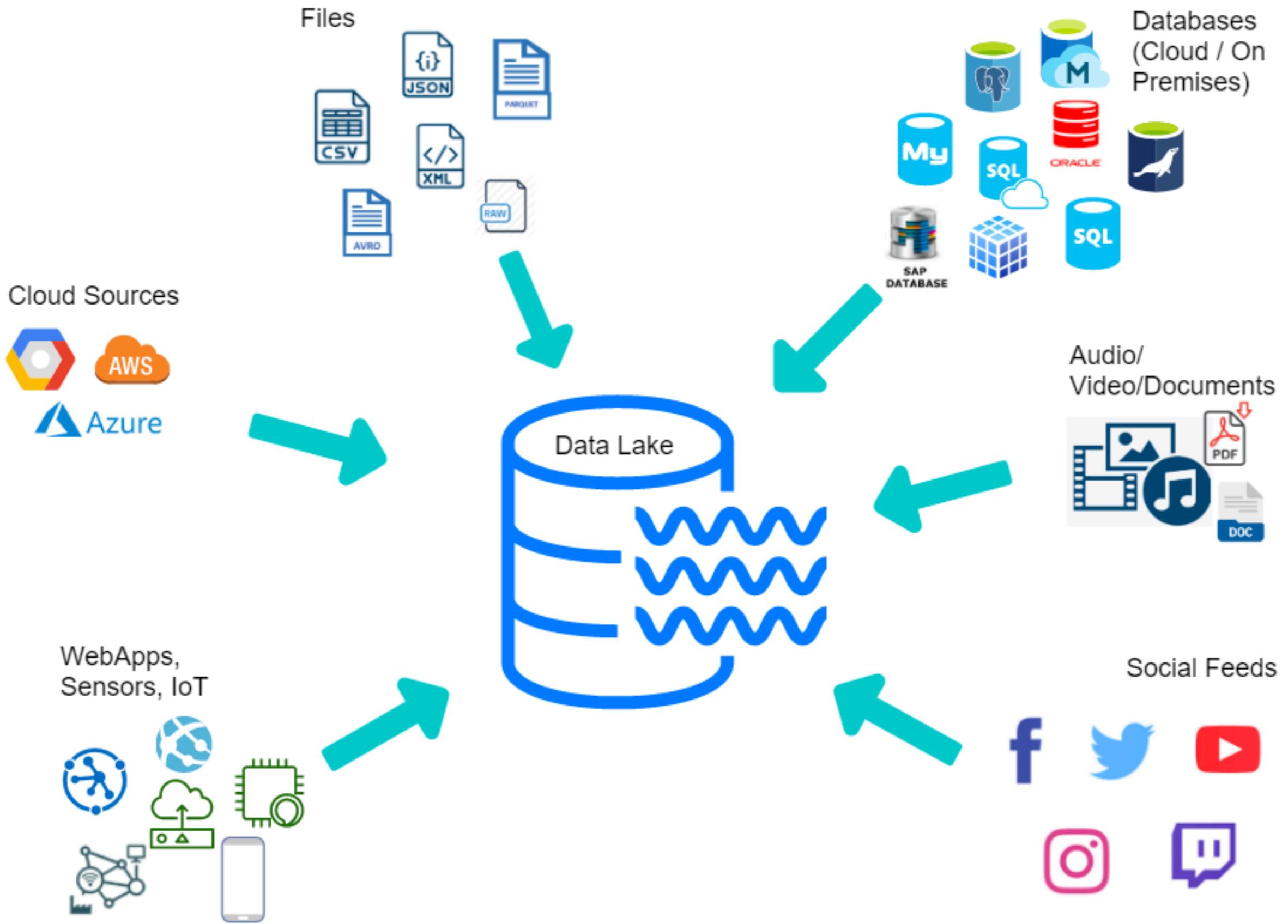
tblGender	
ID	Gender
1	Male
2	Female
3	Unknown

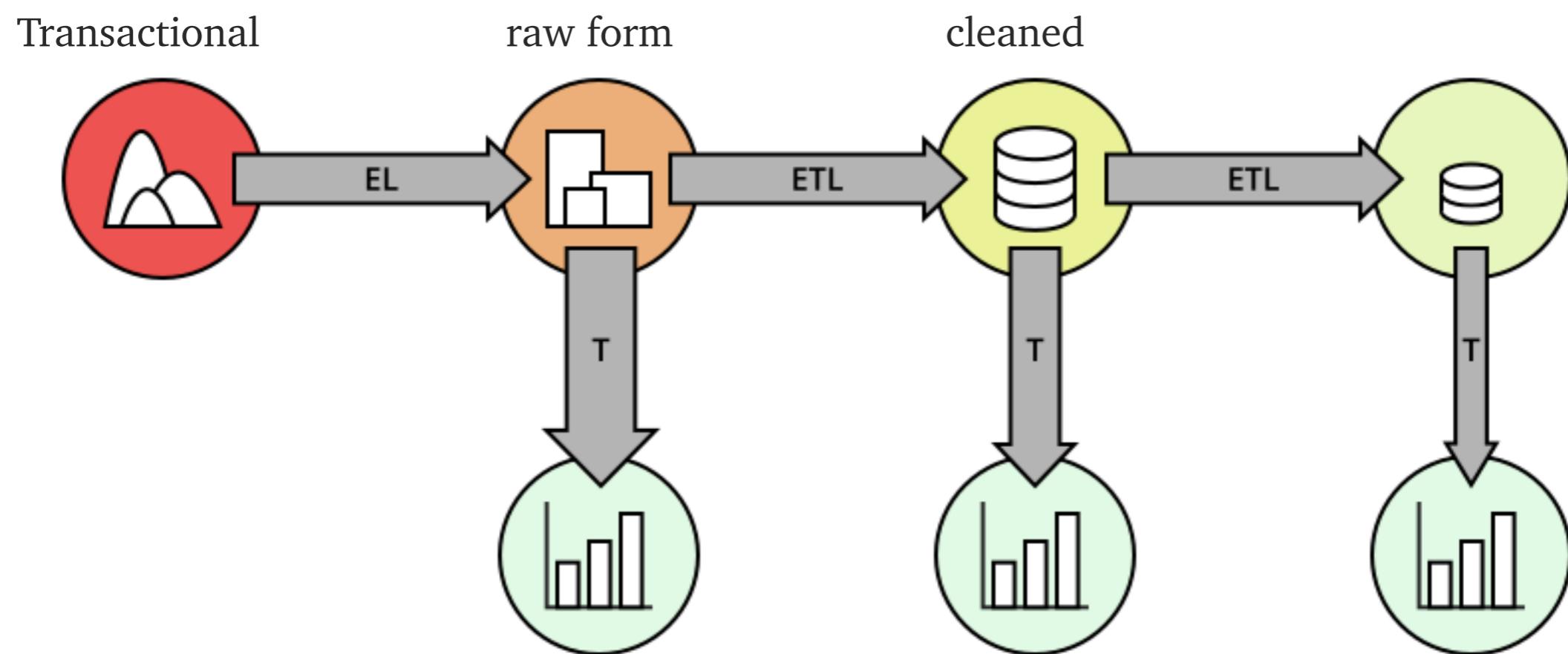
key	value
123	123 Main St.
126	(805) 477-3900



## 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

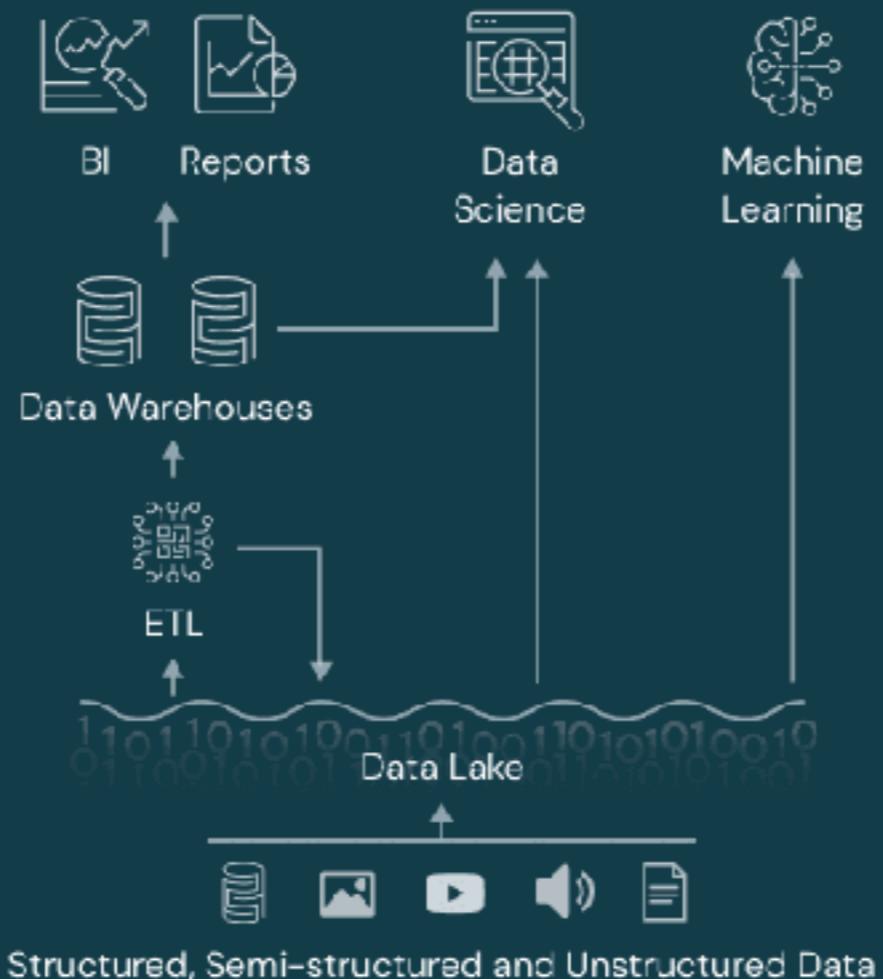




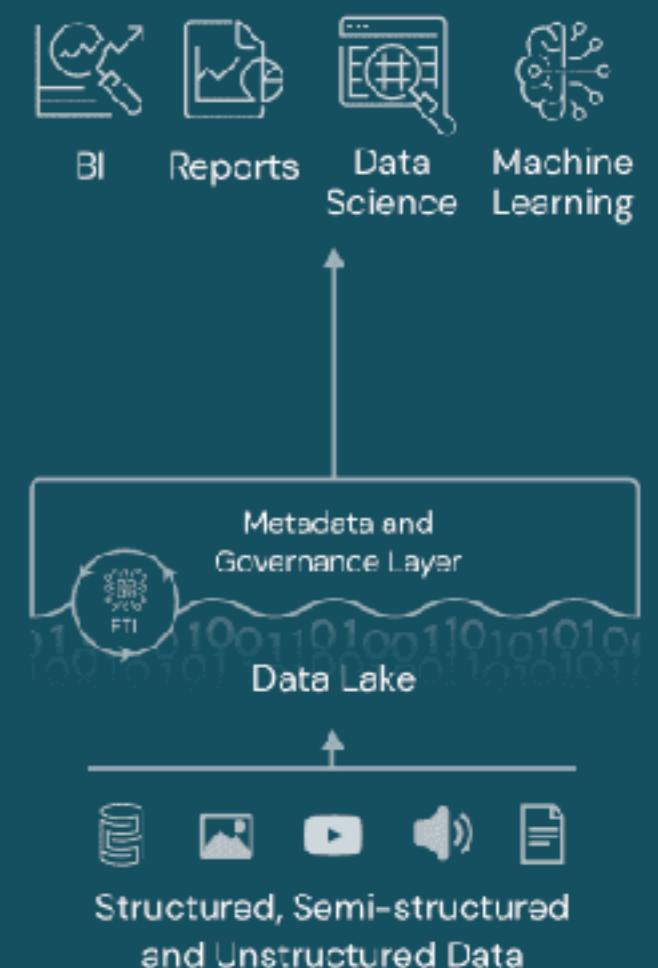
## Data Warehouse

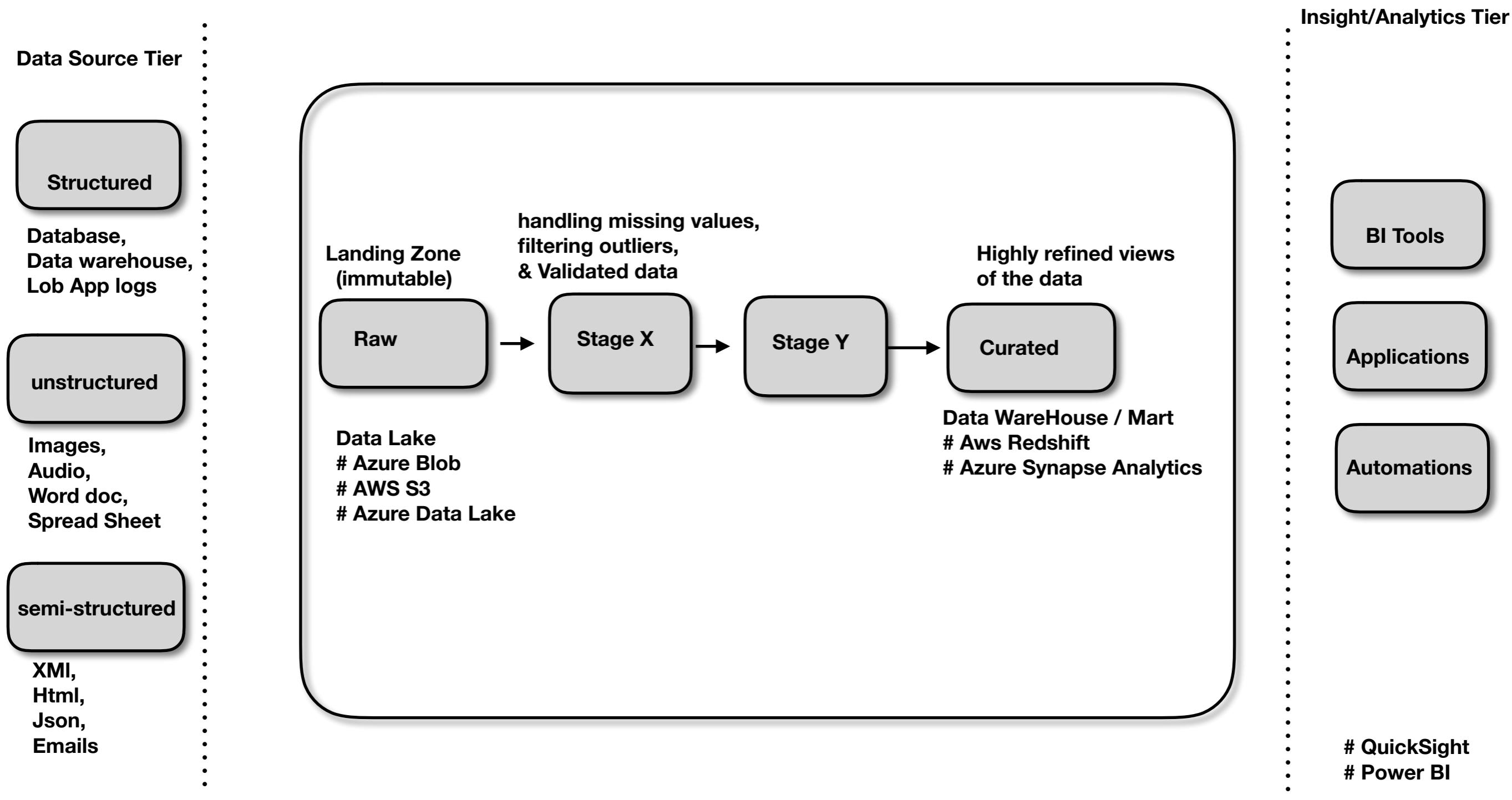


## Data Lake

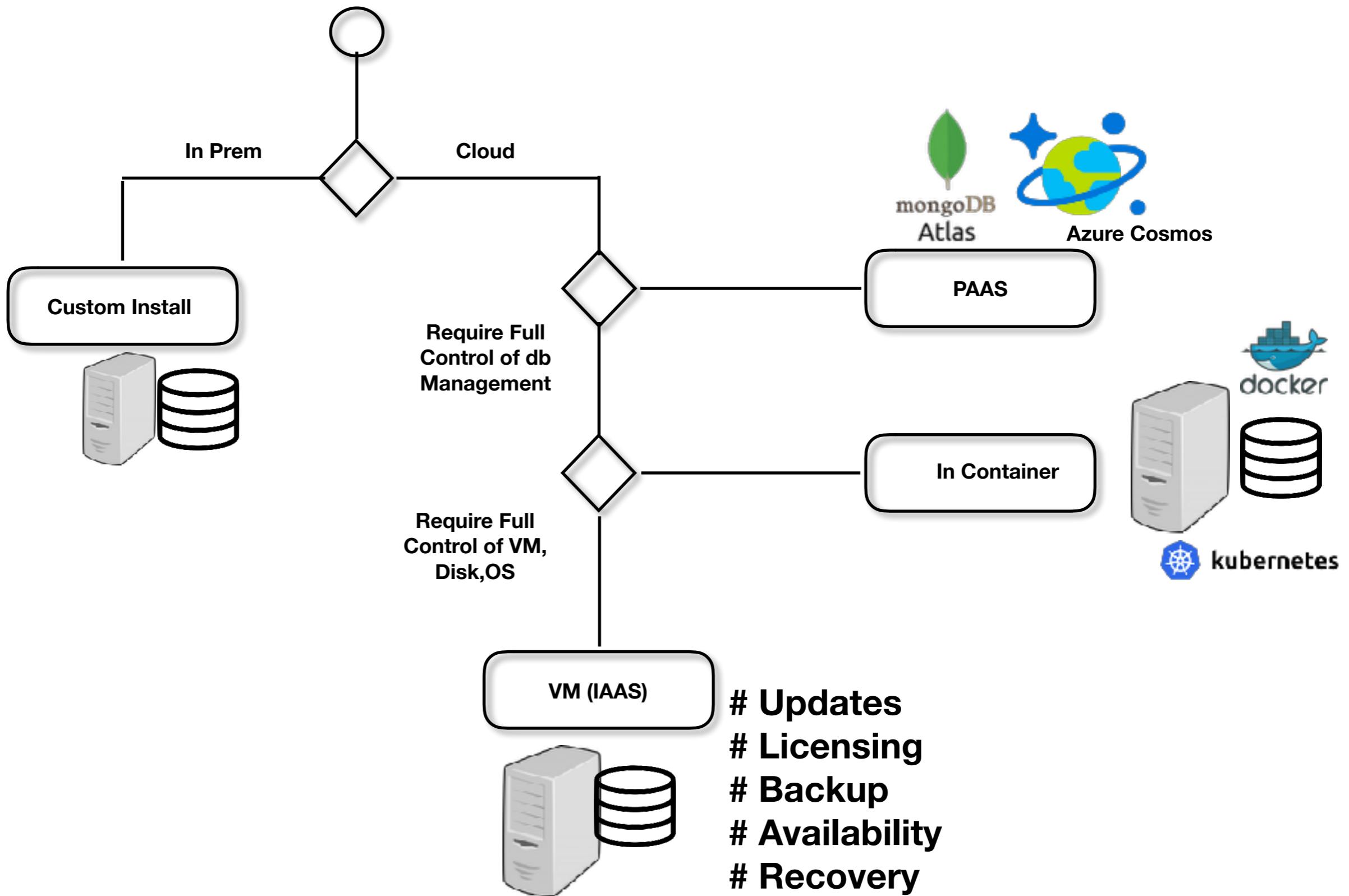


## Data Lakehouse

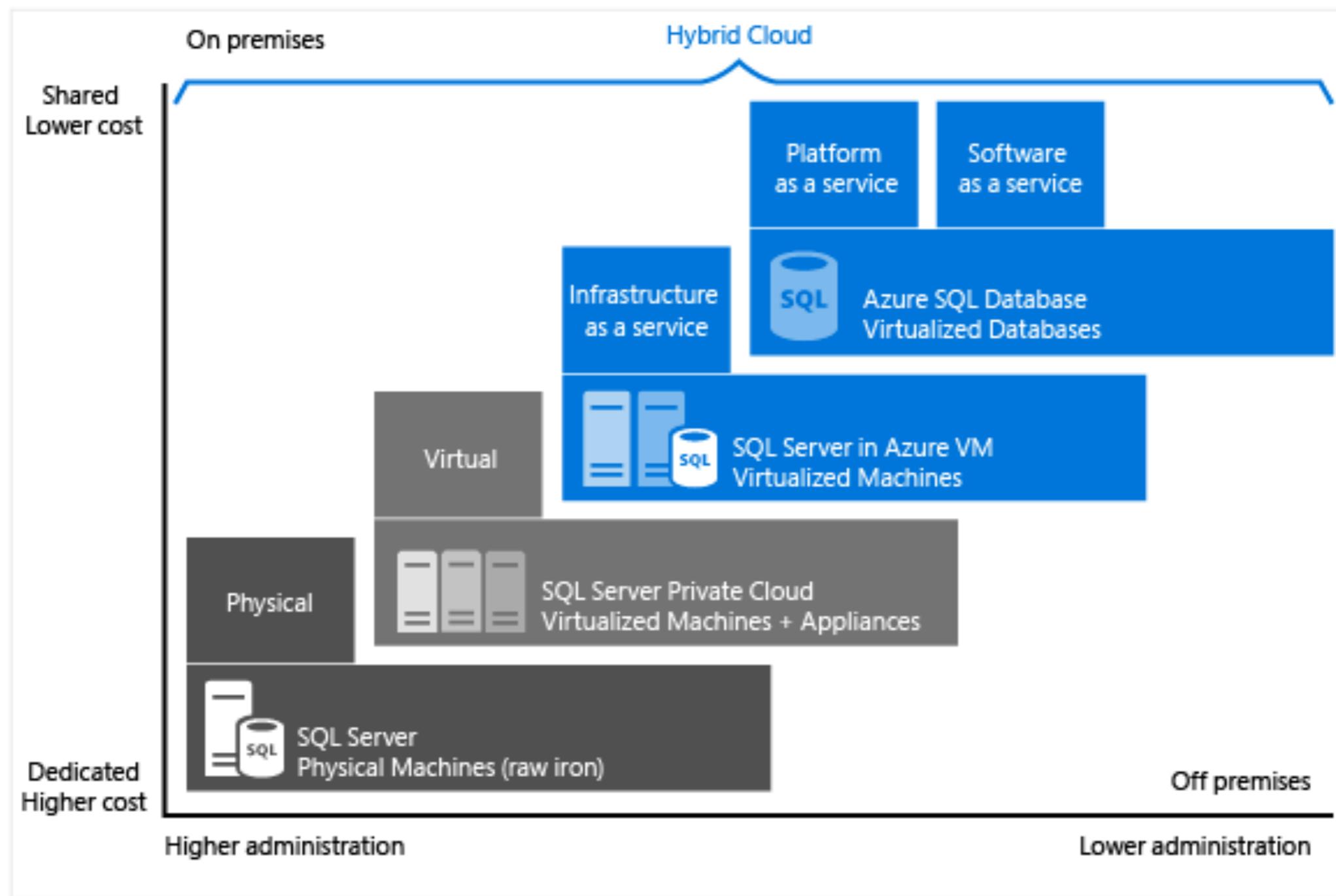


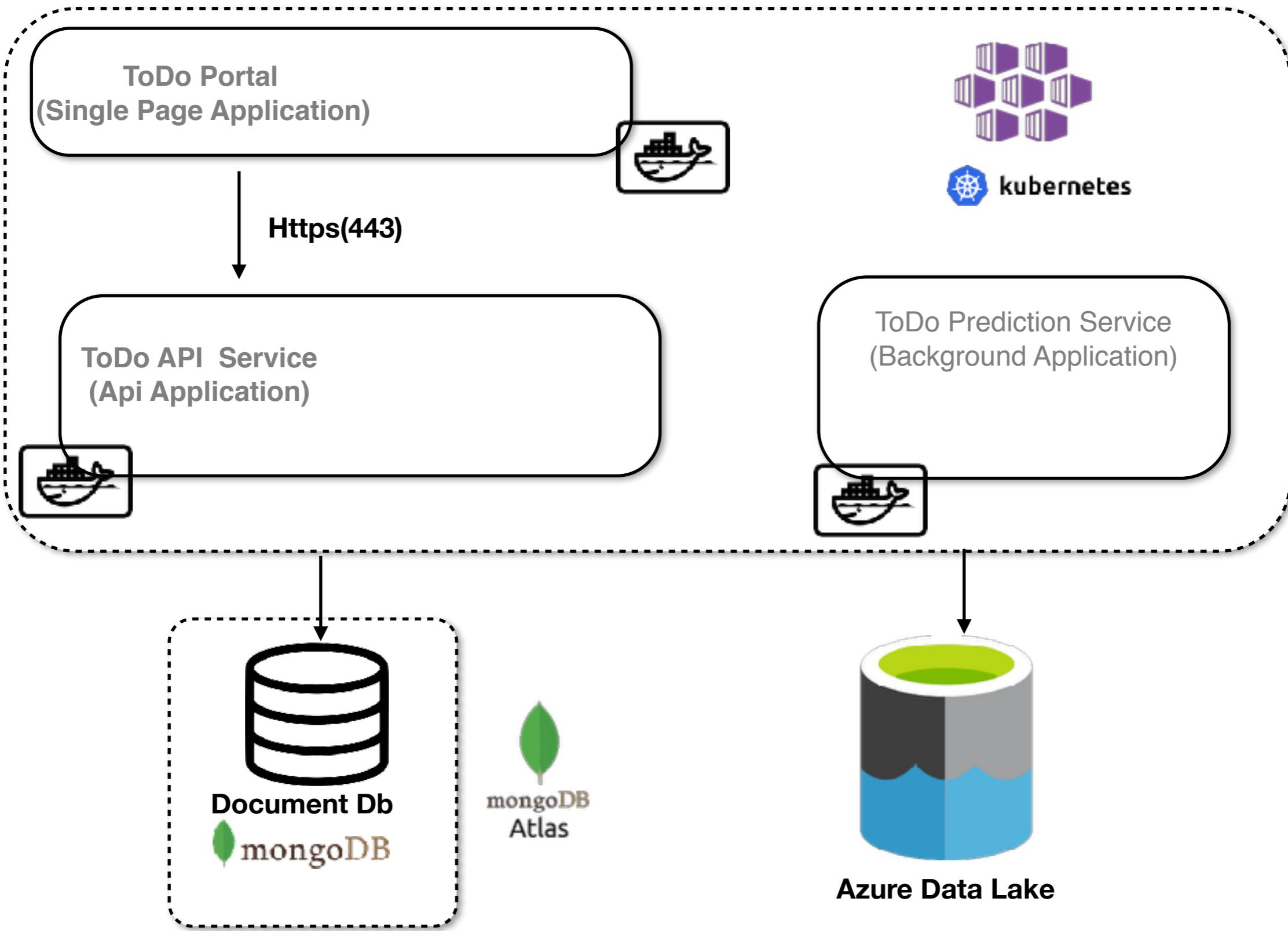


# Deploying Data Tier

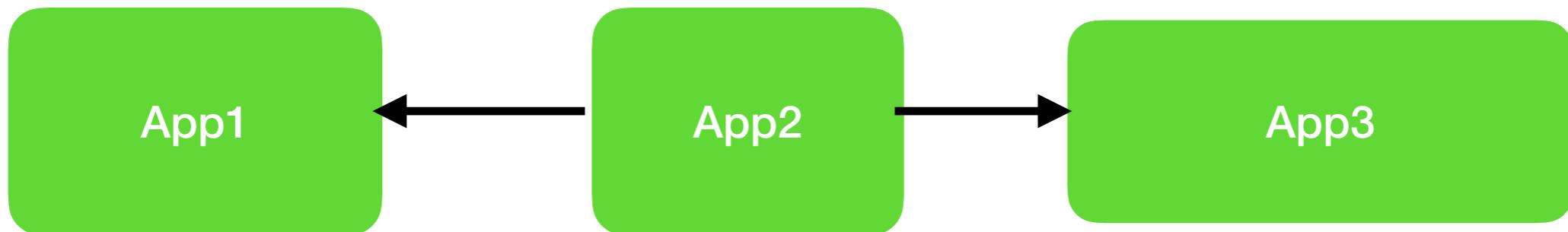


# Deploying Data Tier

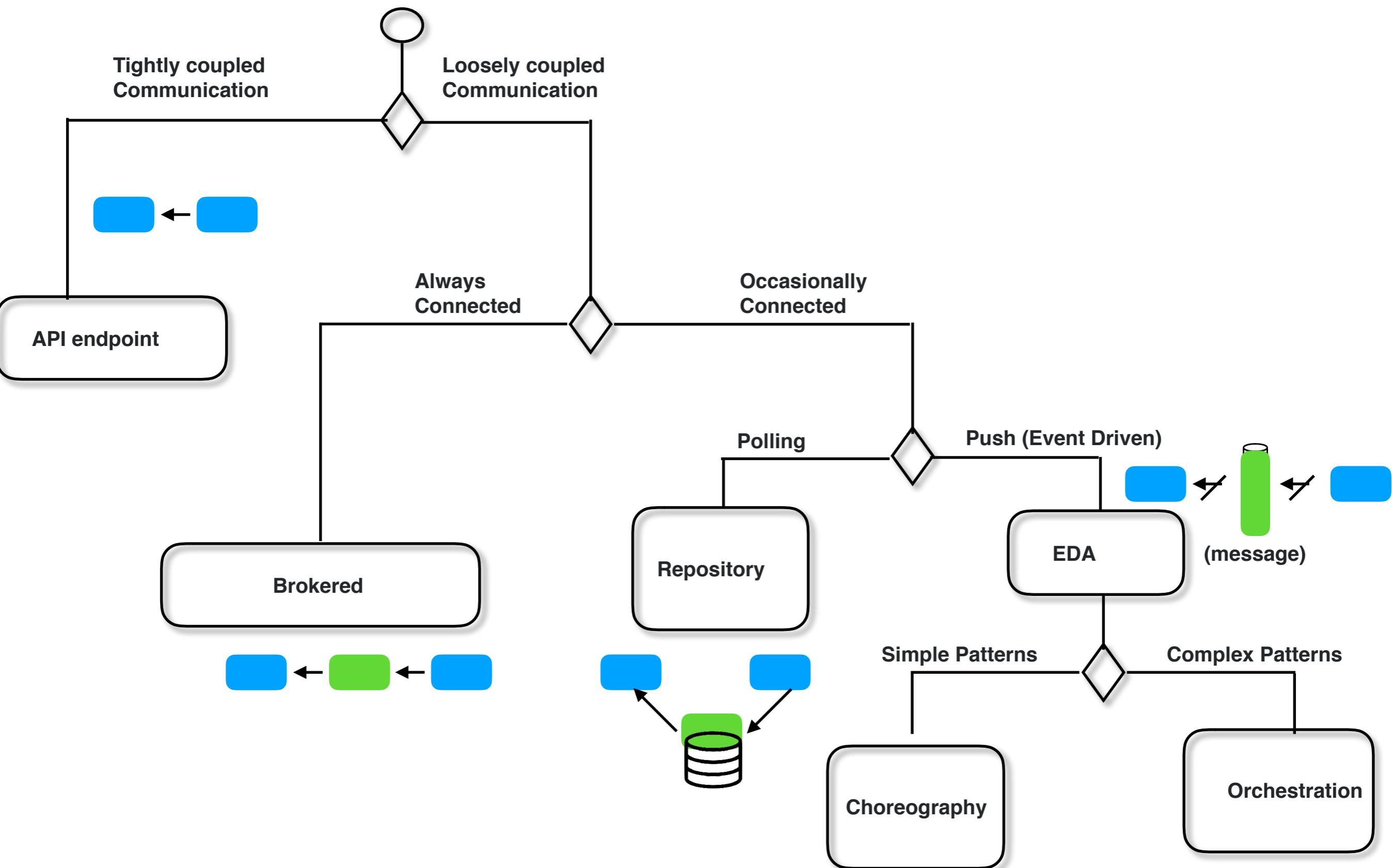


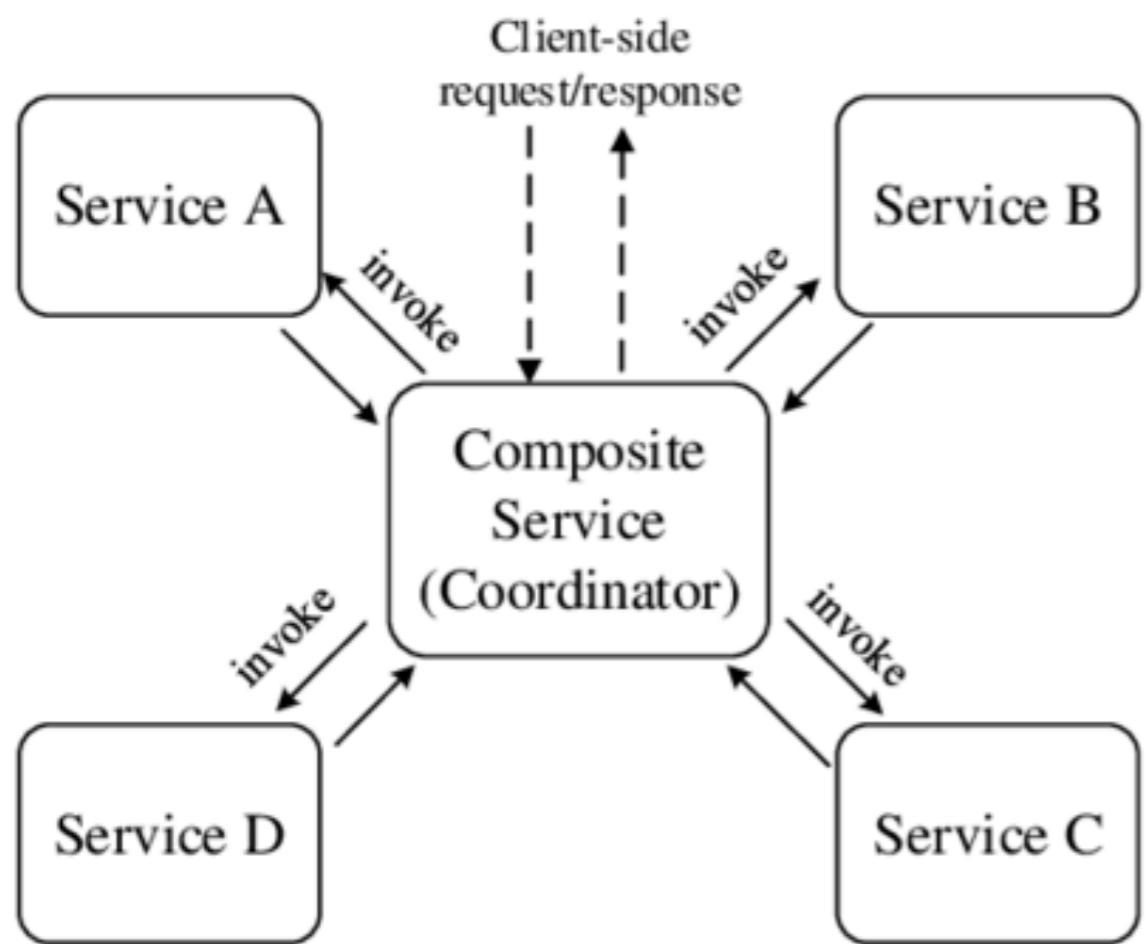


# System Integration

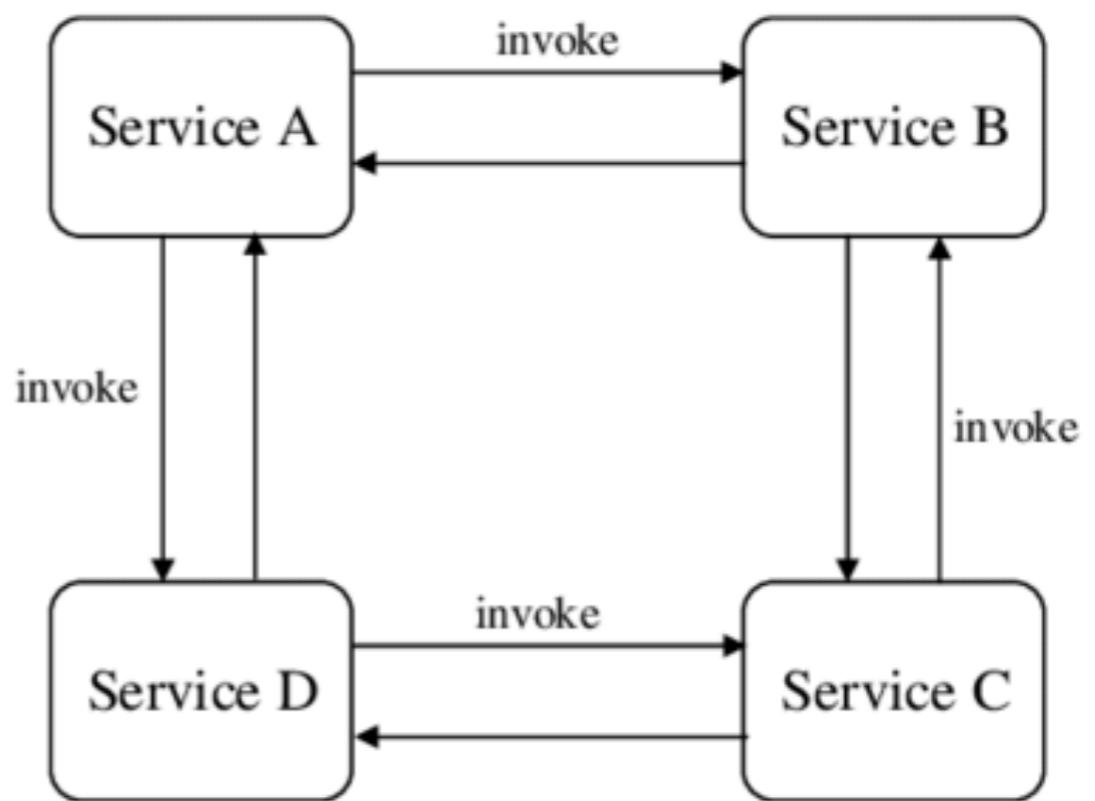


# Choose Communication Mechanism

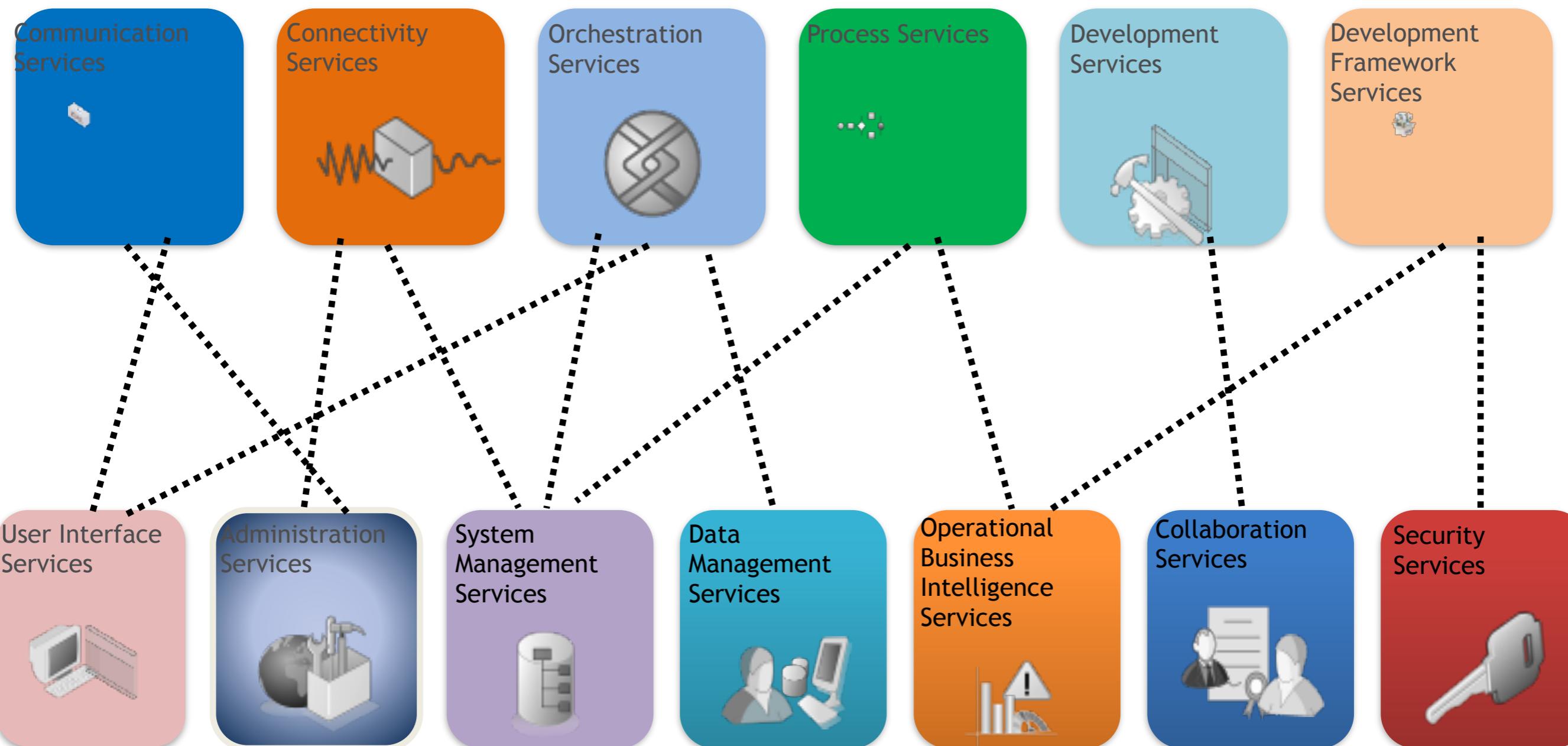


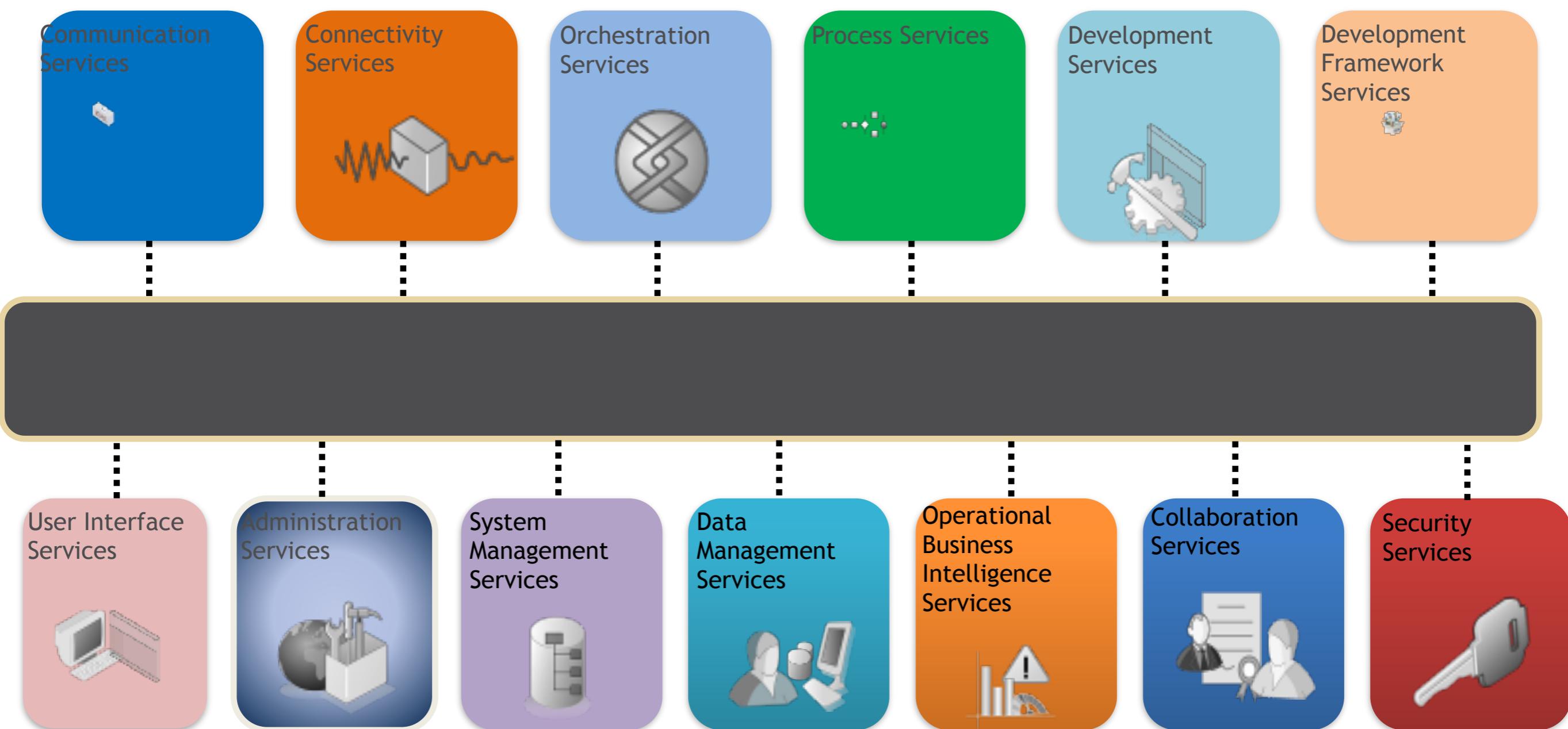


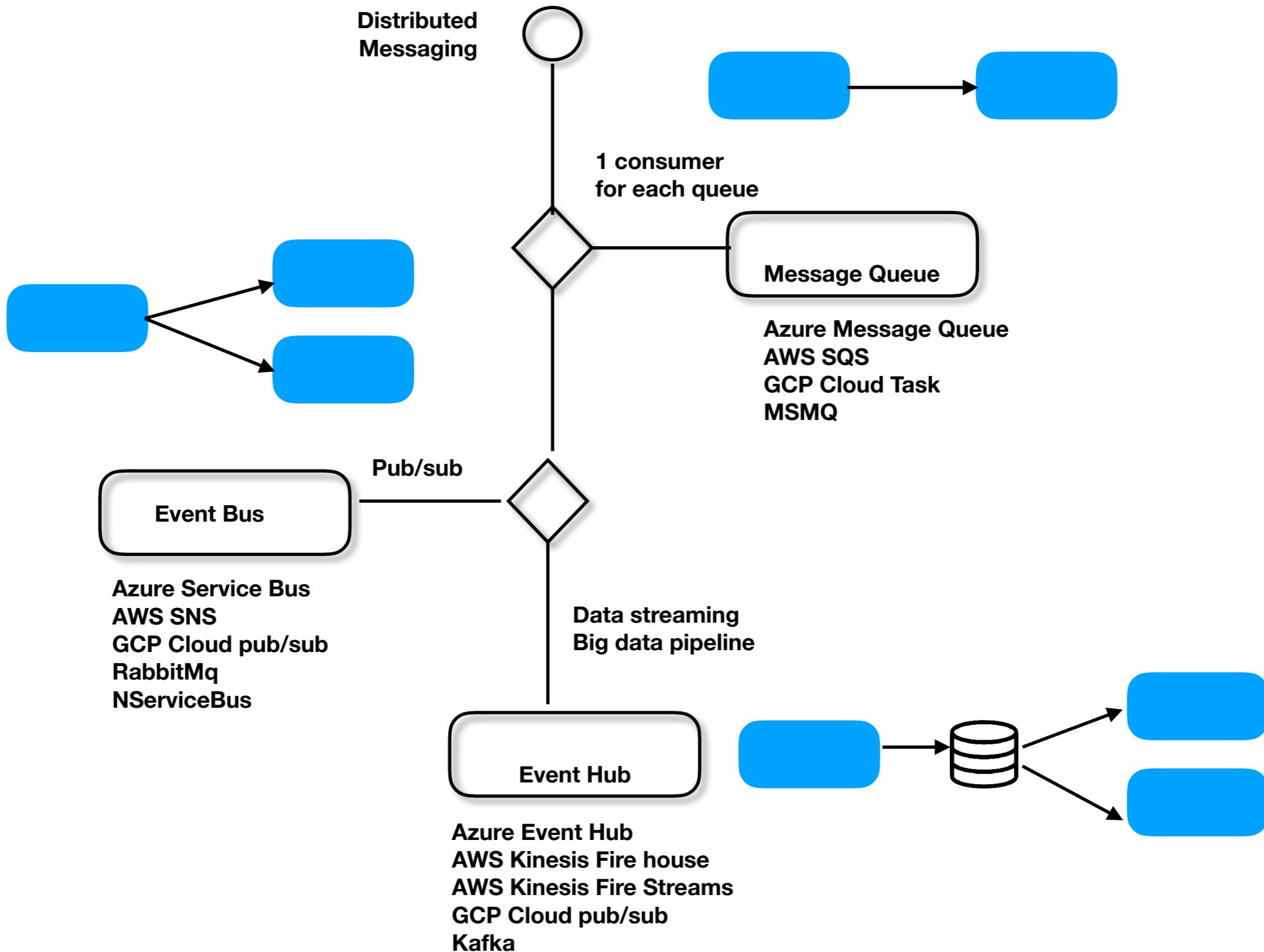
(a) Web Service Orchestration



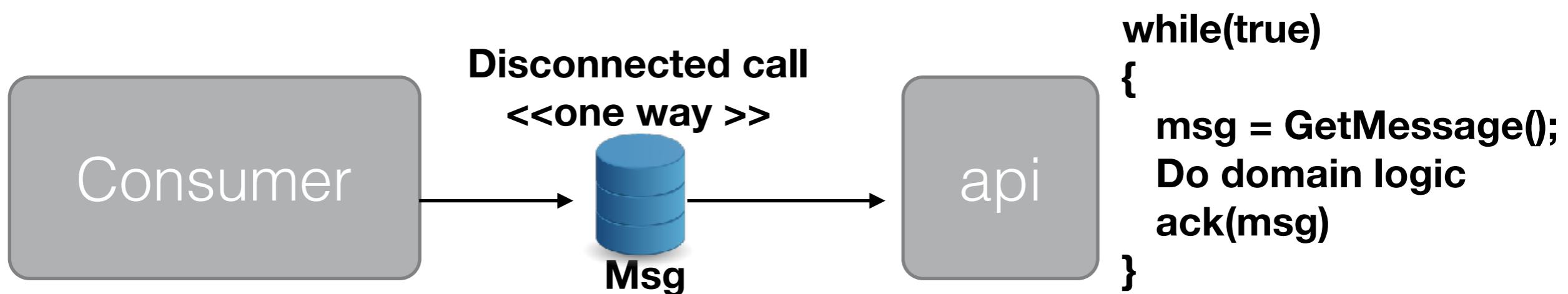
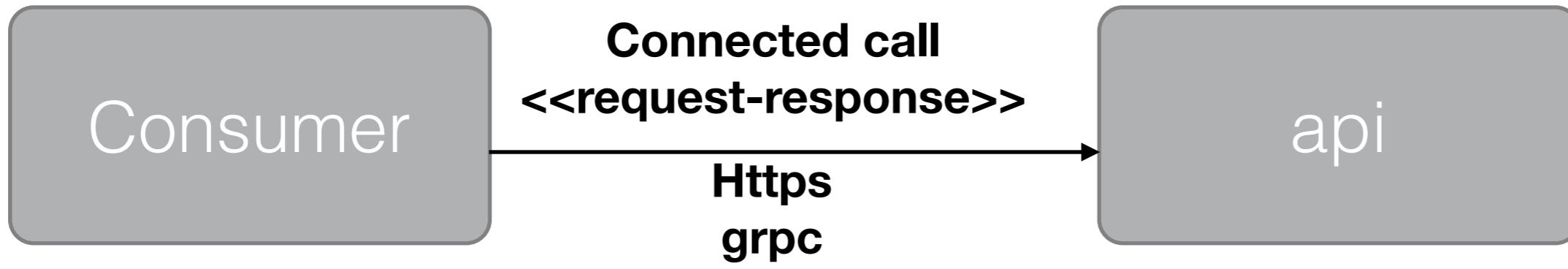
(b) Web Service Choreography



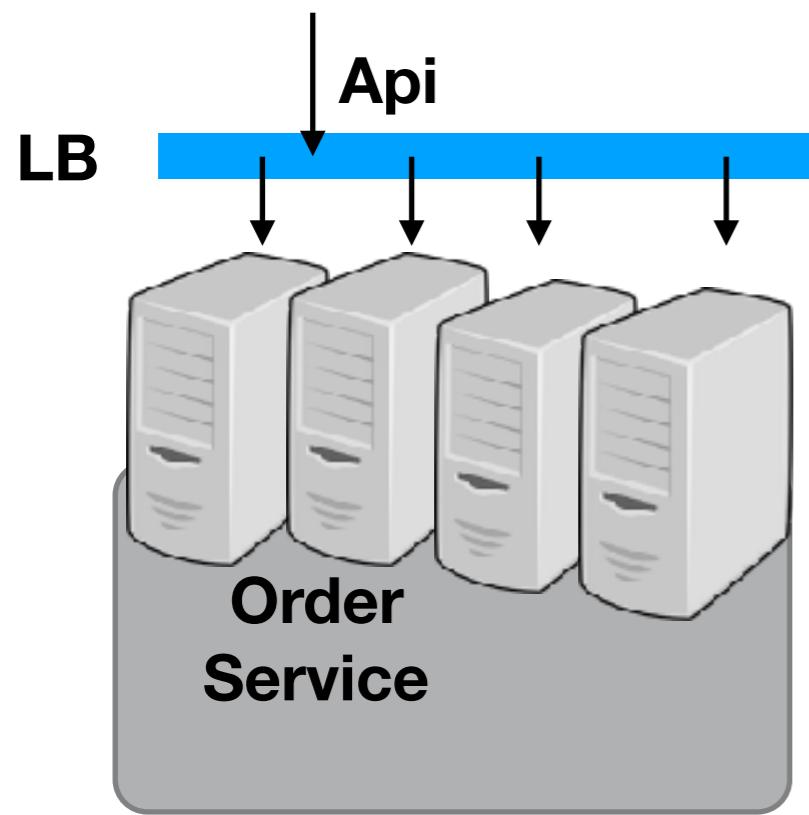




# Logical view Communication



# API call



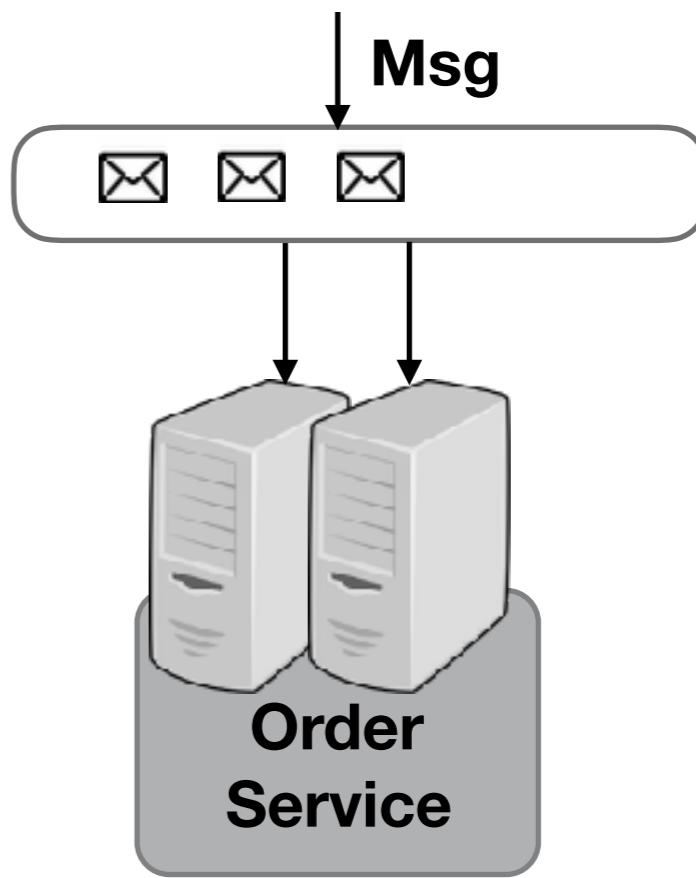
# immediate Consistency

# 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



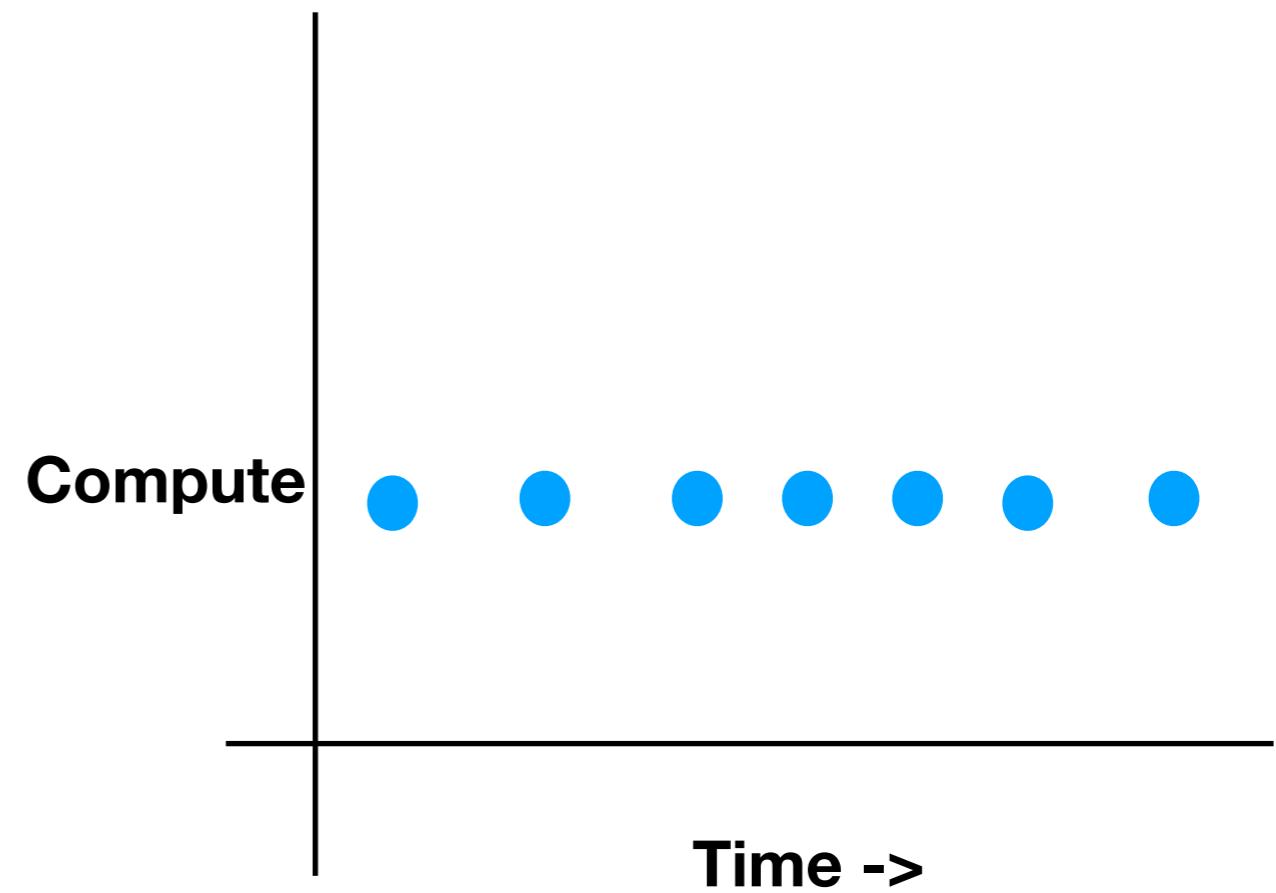
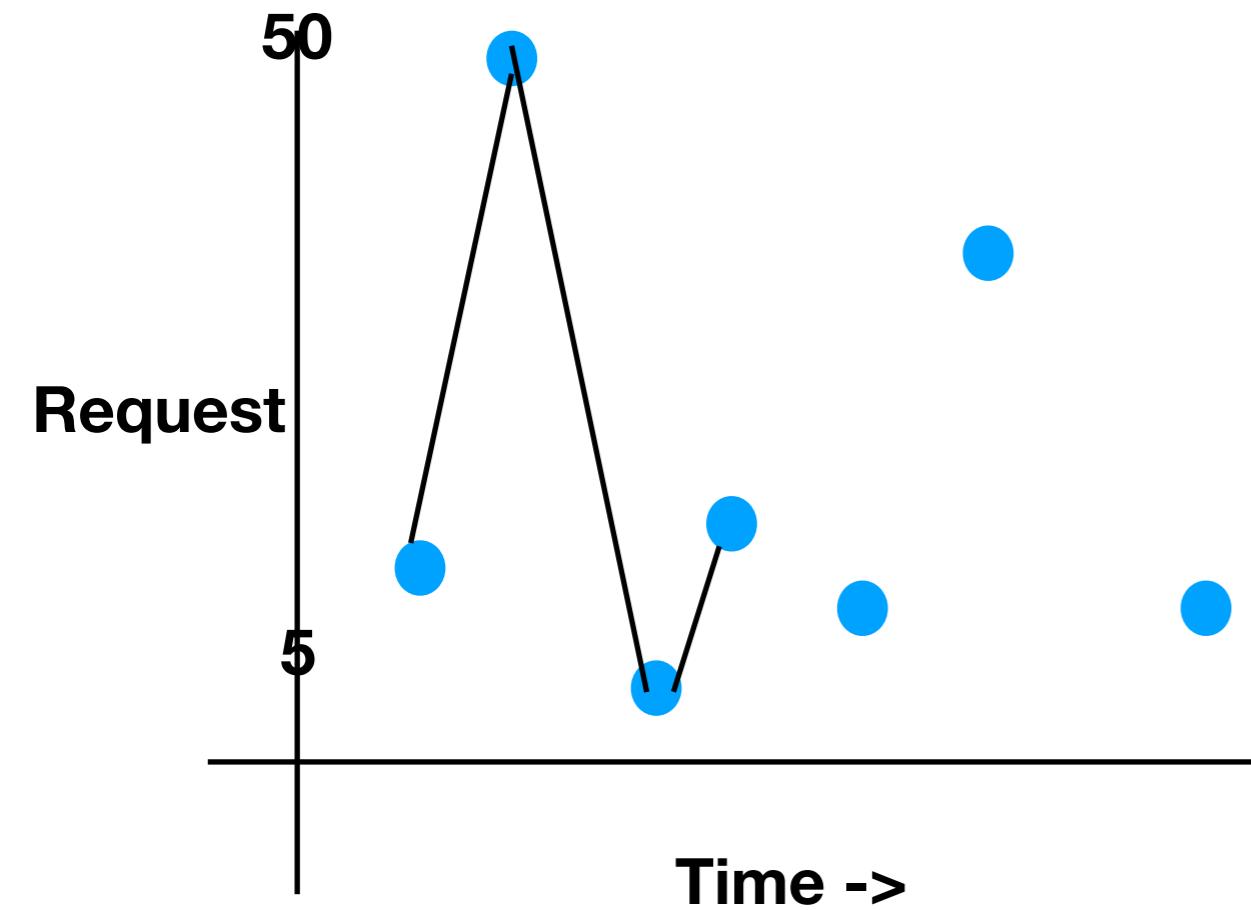
# Eventual Consistency

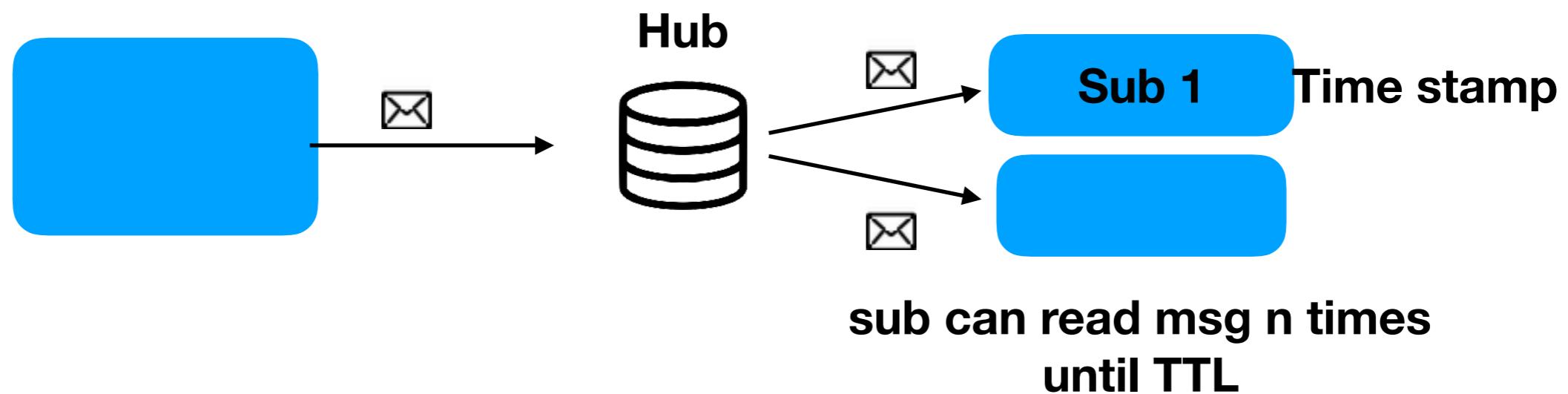
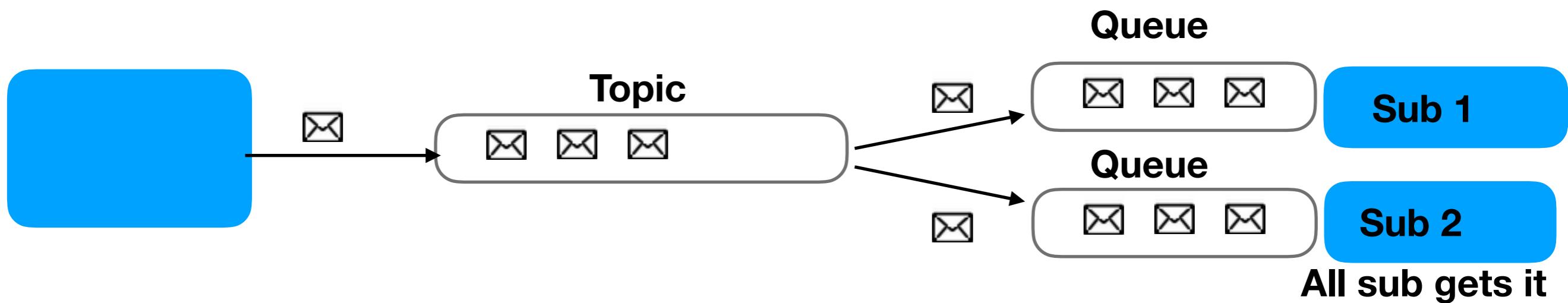
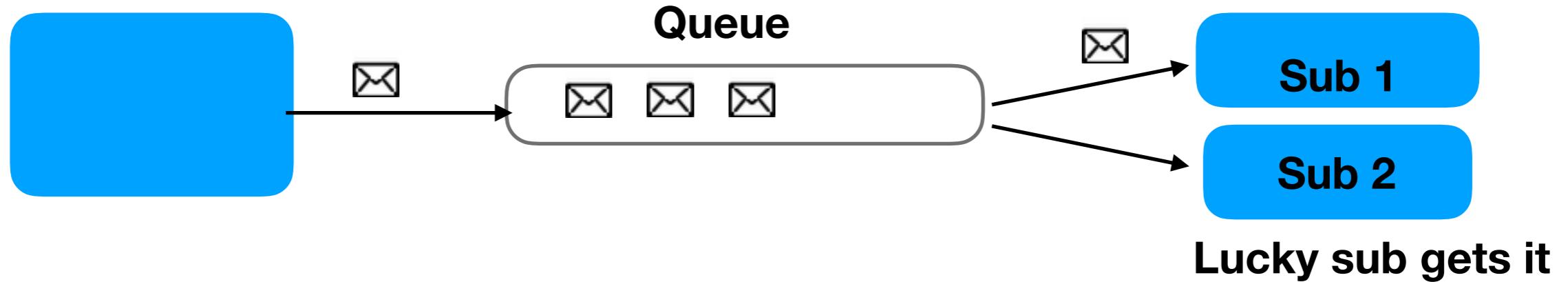
# more scalable  
no of server's depends on Avg load  
(Load Leveling)

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

# no loss of data  
Gets queued

# Load Leveling





**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

Class Table

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



## Normalised db

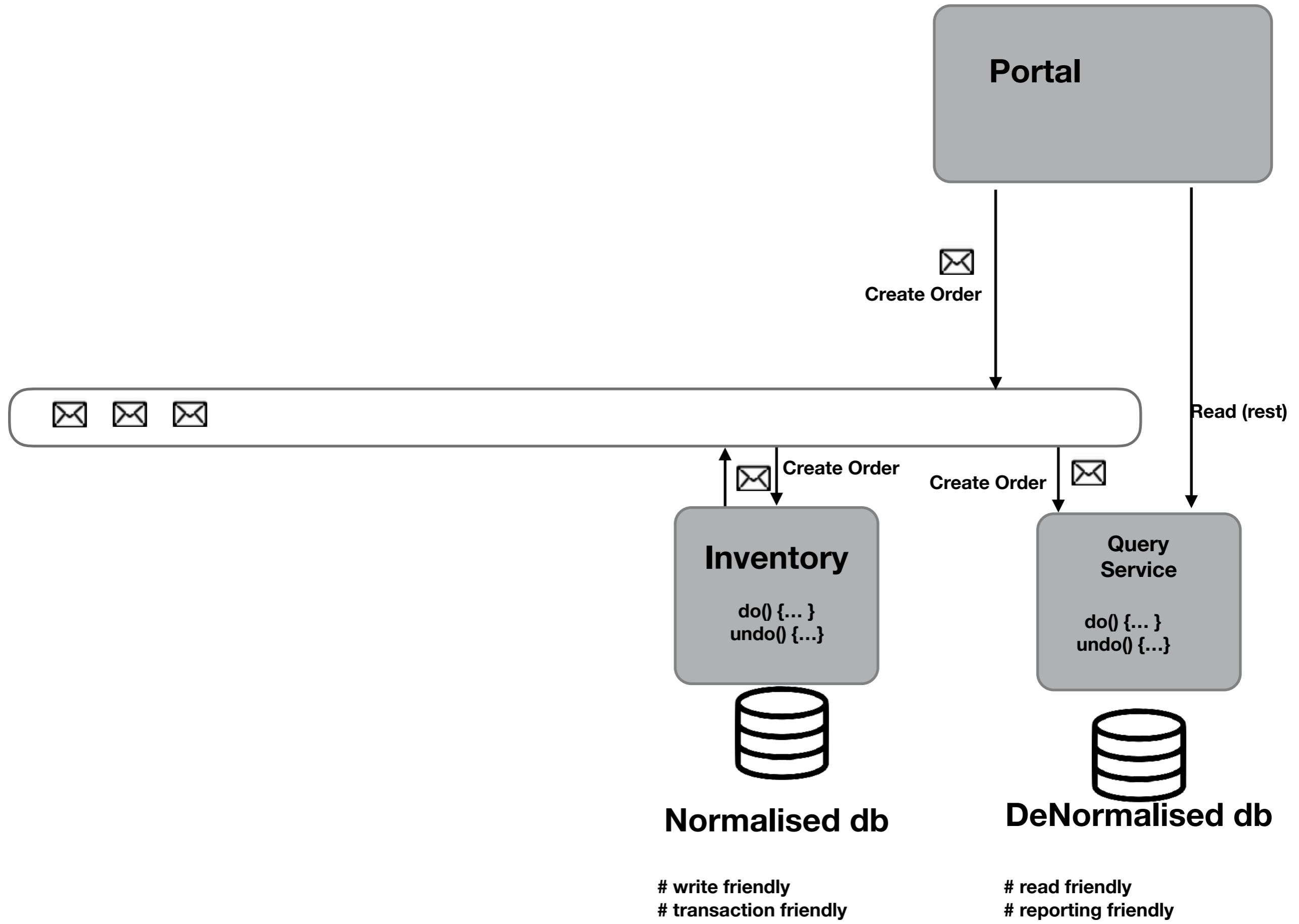
# write friendly  
# transaction friendly

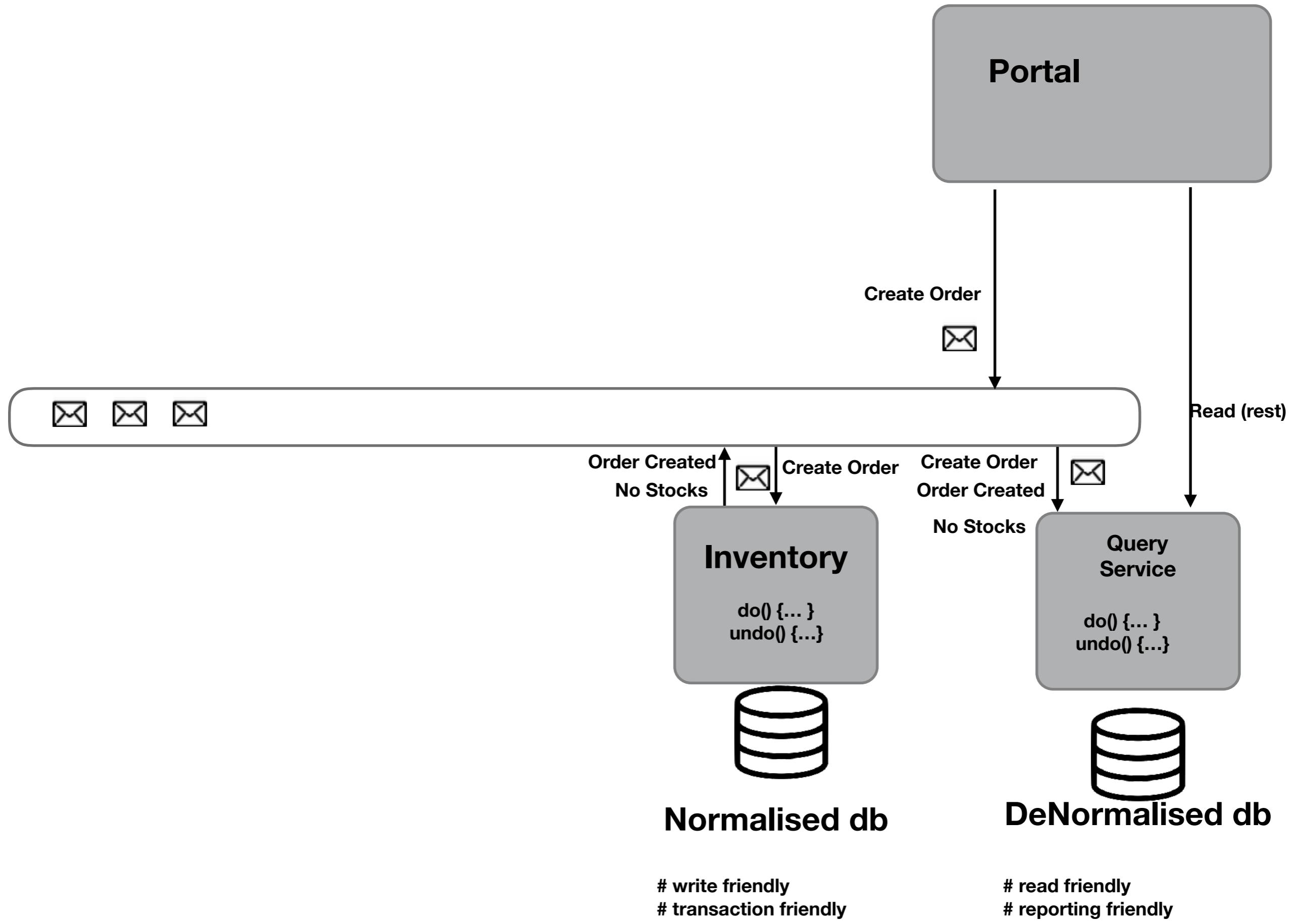
## Query Service

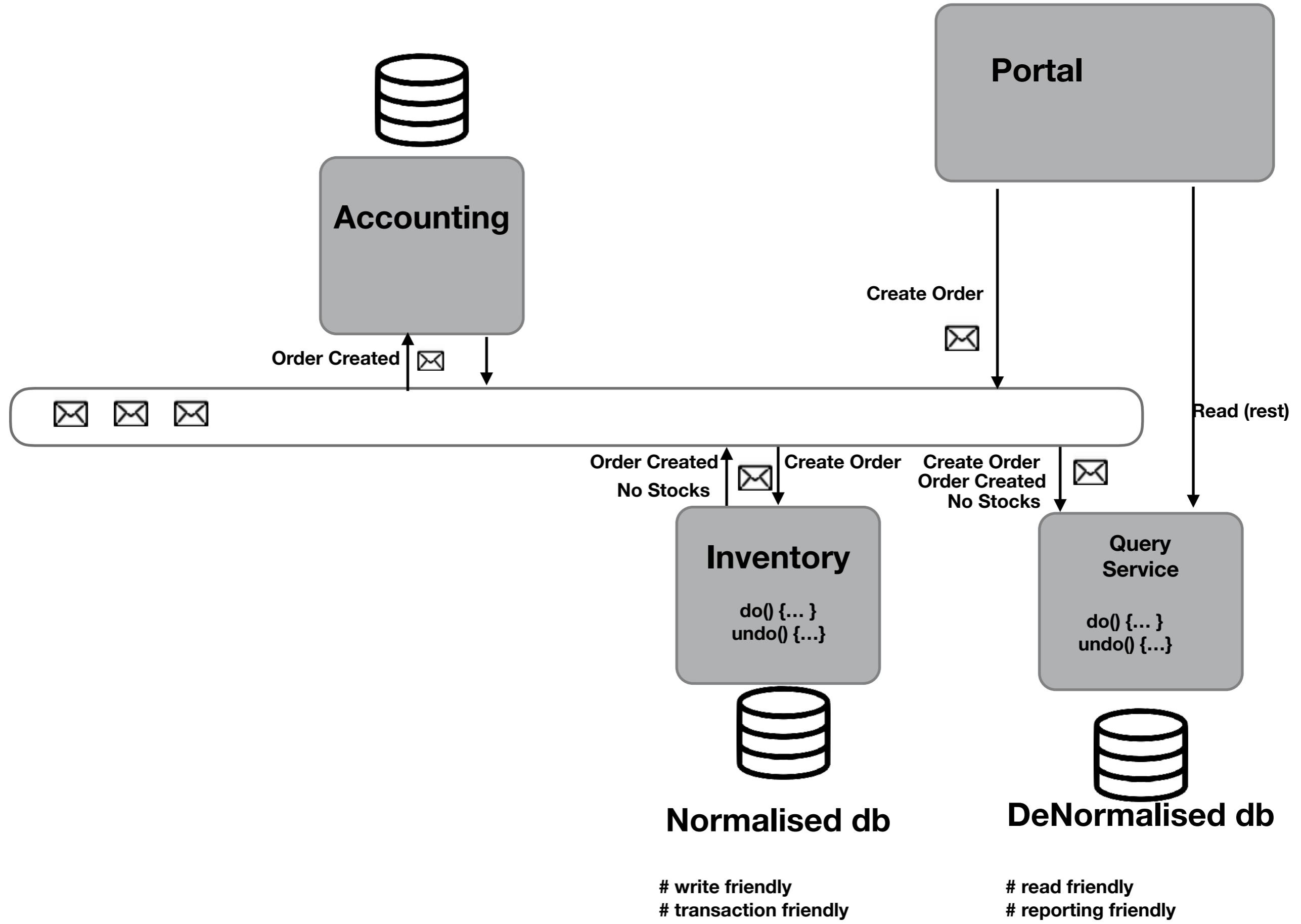


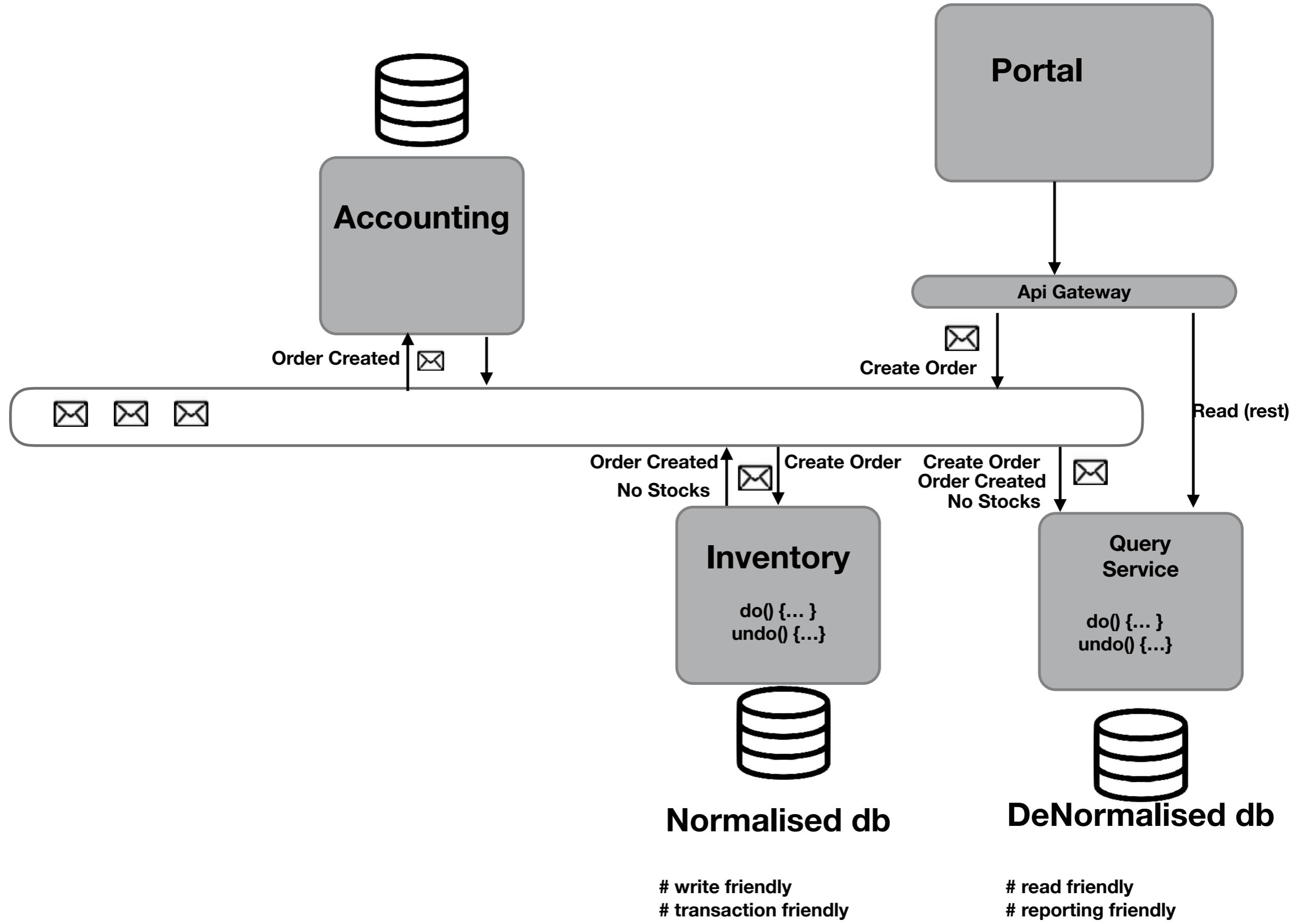
## DeNormalised db

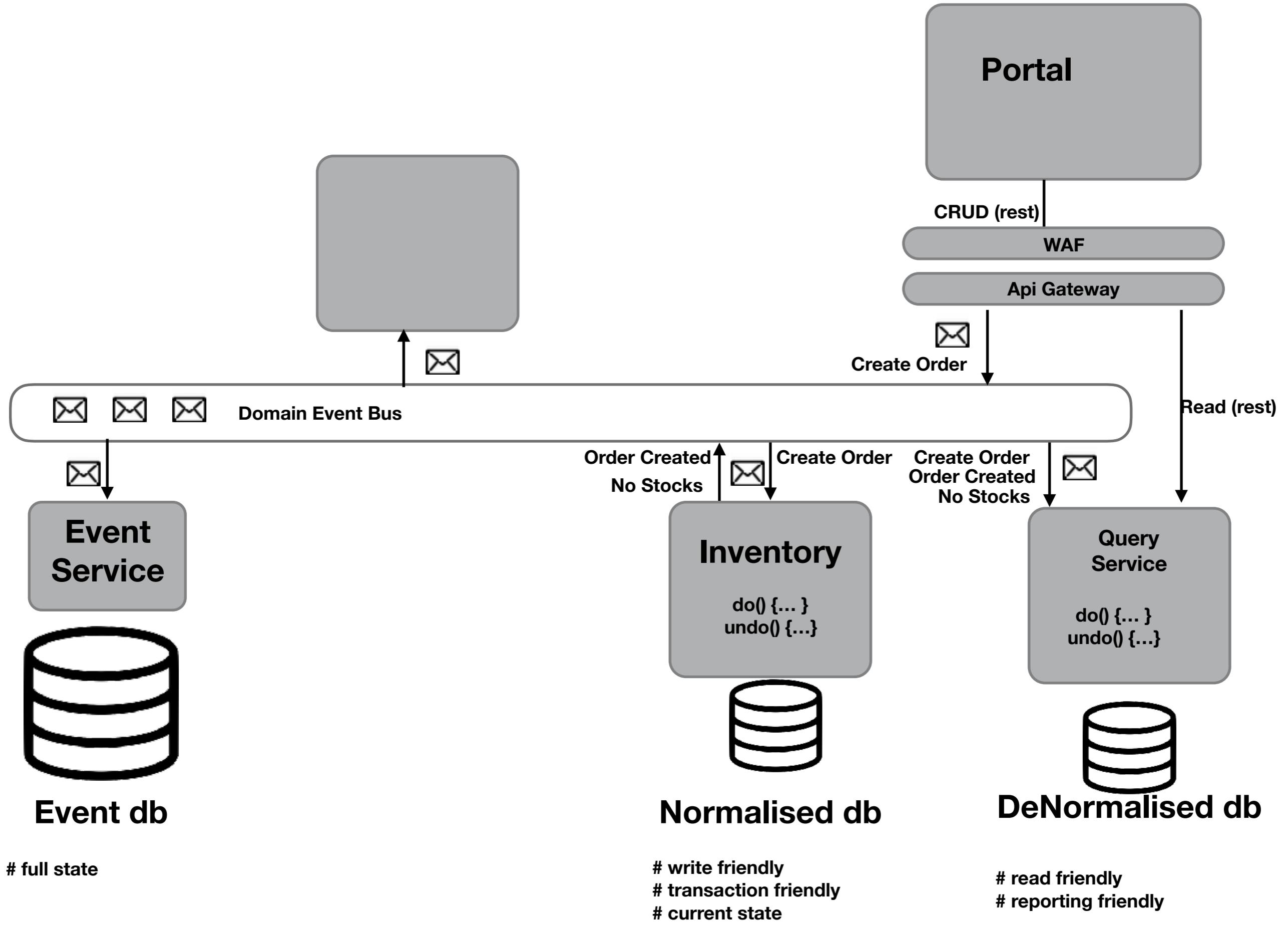
# read friendly  
# reporting friendly

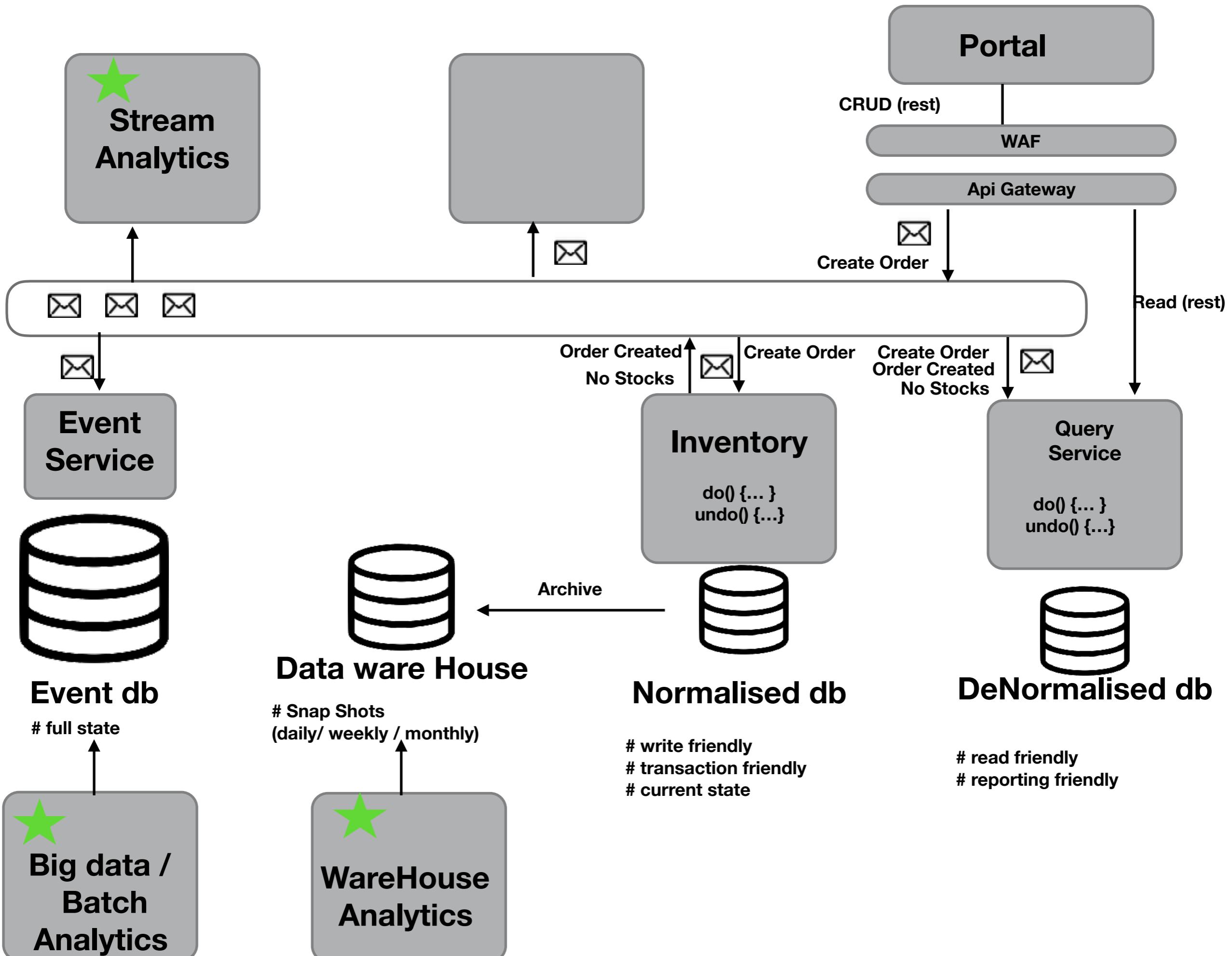










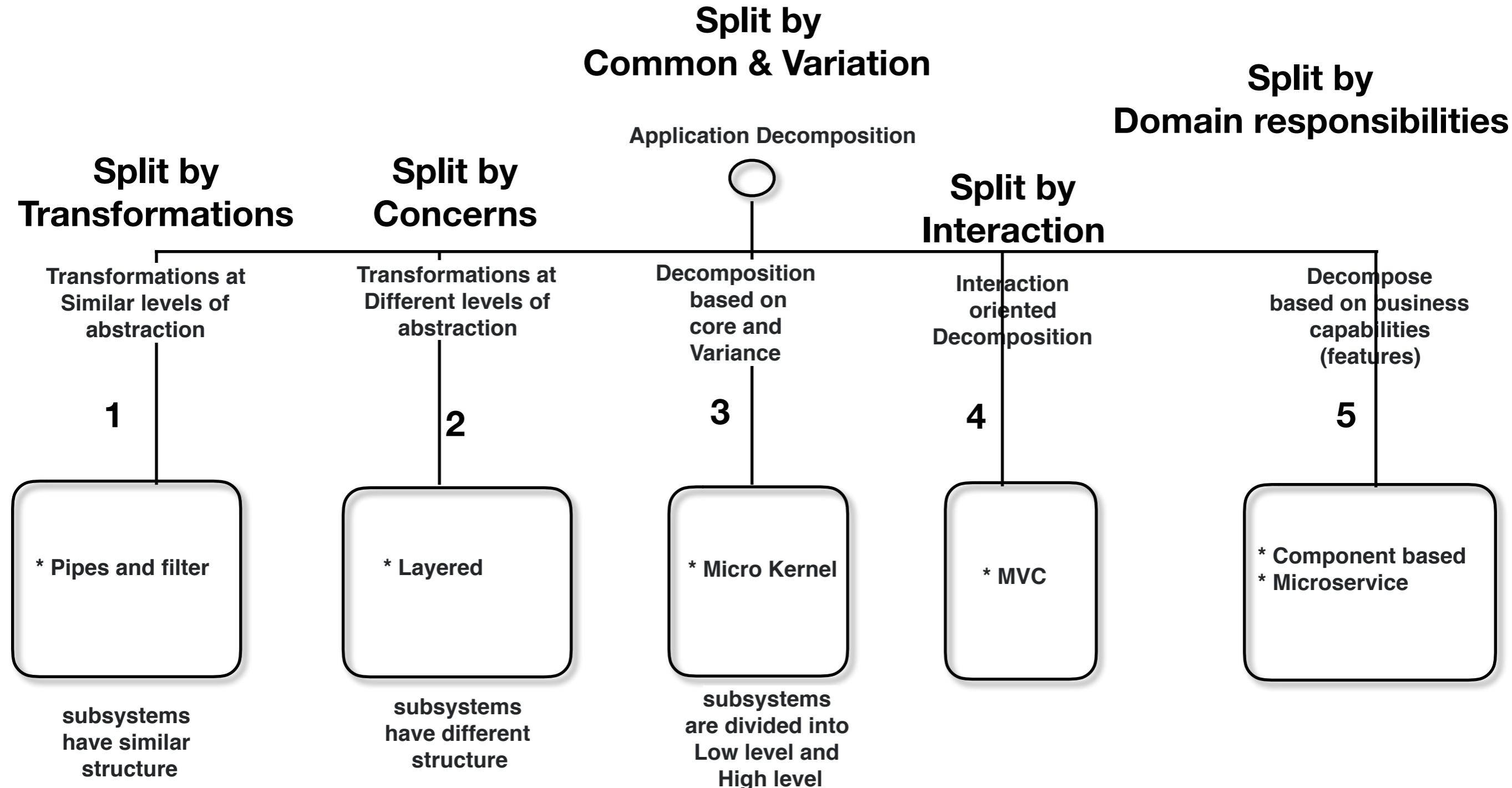


	Rabbitmq	Kafka	Redis
Scale	50K msg per second	1 million/ sec	1 million/ sec
Transient messages	Yes	No	Y
Persistent yes	Yes	Yes	Y
Protocol	AMQP		
One to one	Y	N	Y
One to many	Y	Y	Y
Advanced Message Queueing Protocol-based routing	Y	N	N
consumption pattern	Pull+Push	Pull	
Client Libraries	Py, java, php, .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	

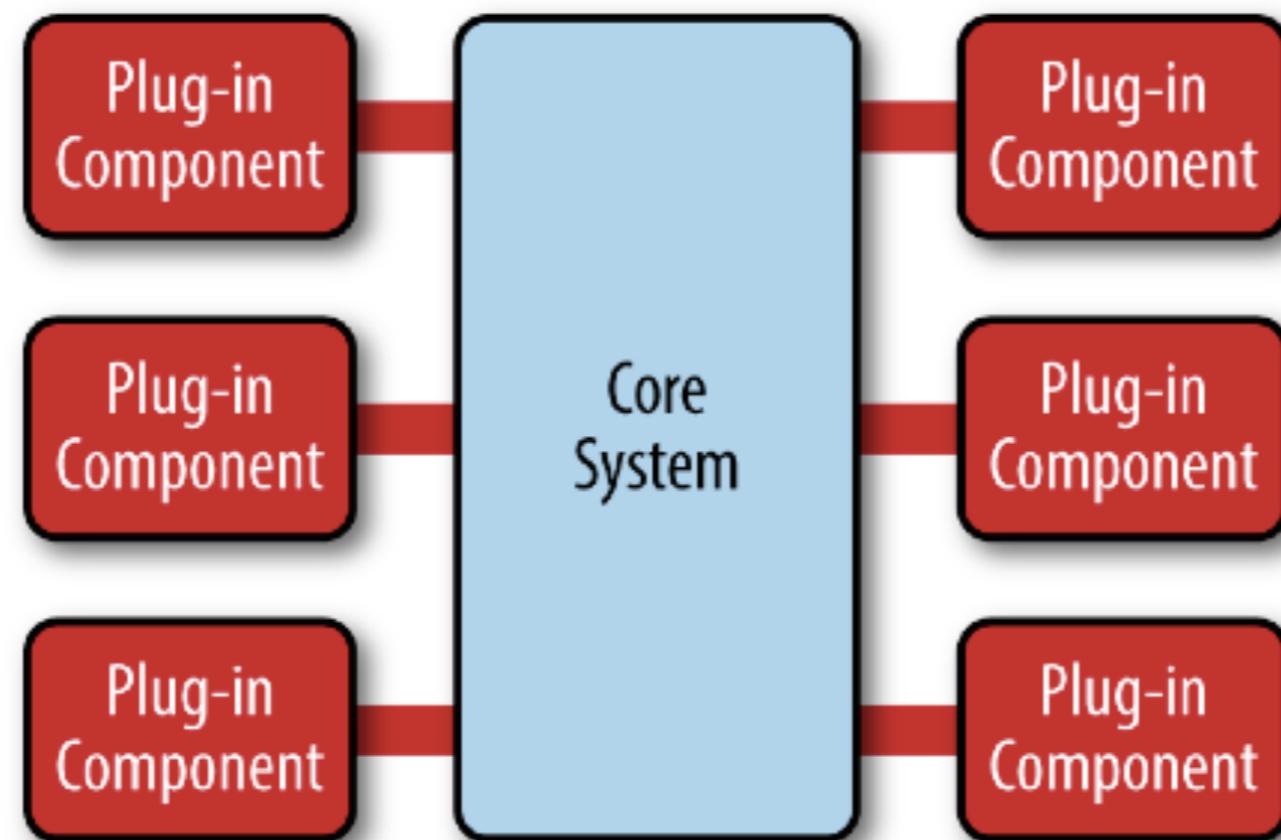
# Application Decomposition



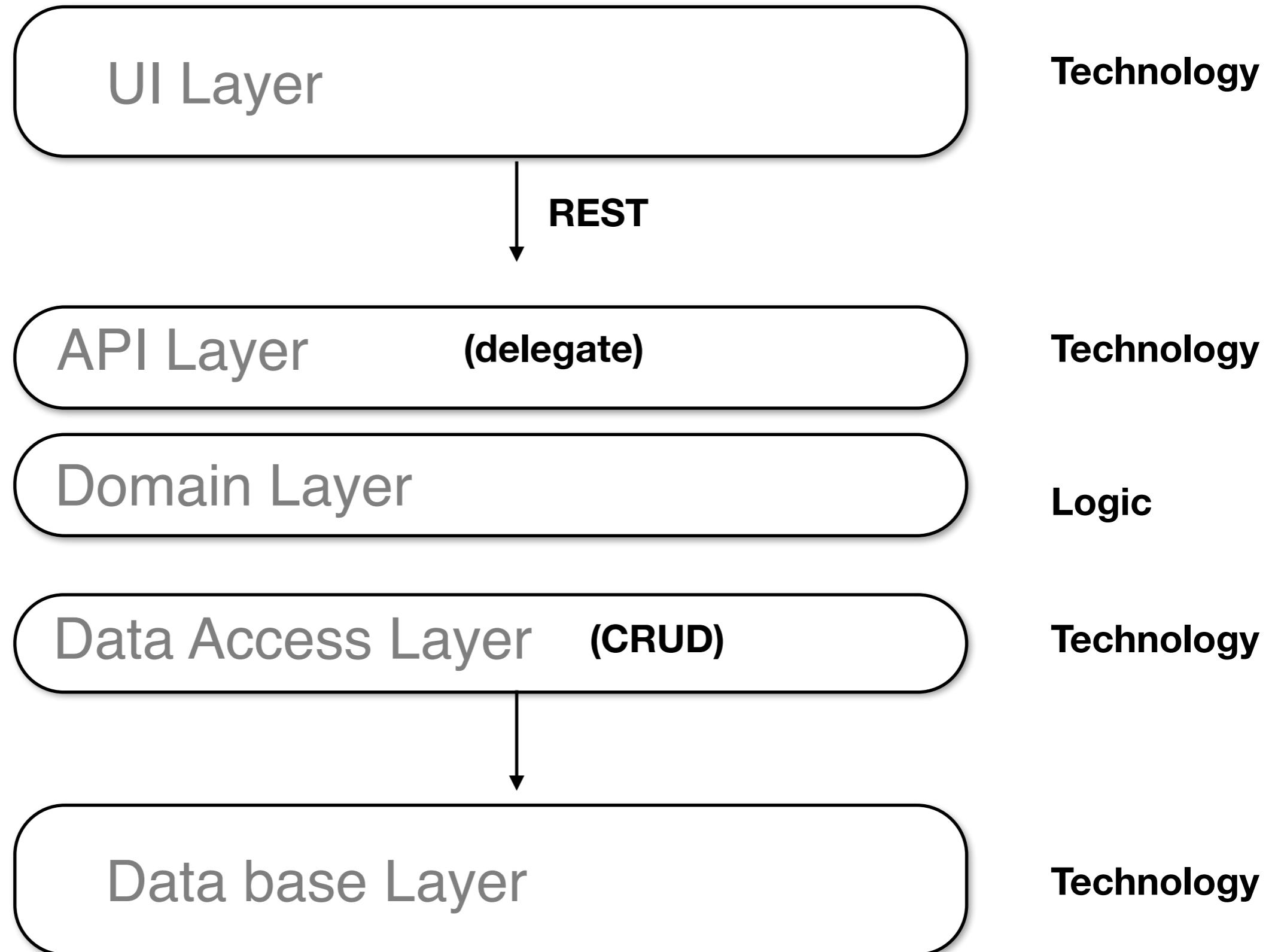
# Decompose application

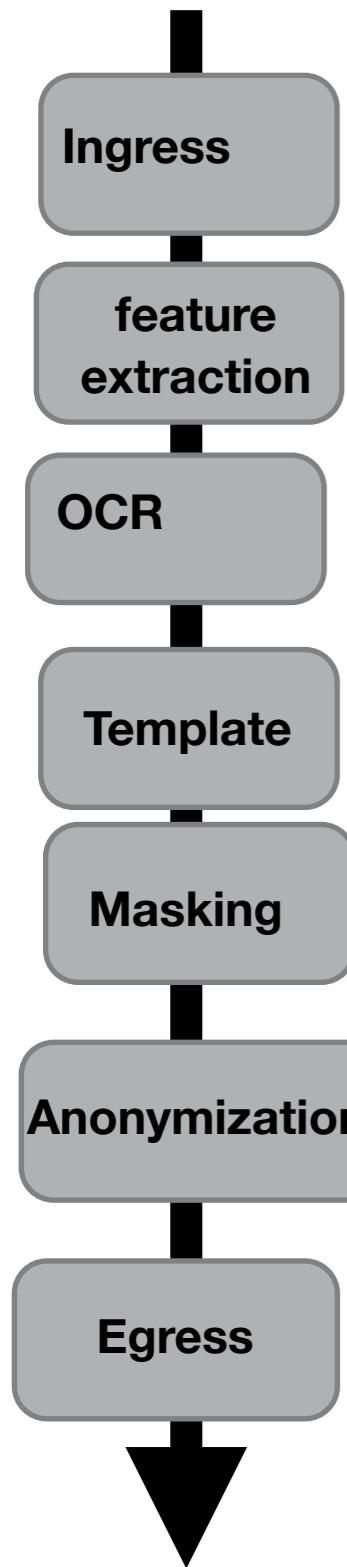


# Microkernel Architecture



# Logical View





## Invoke logic()

Layer 1

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

Layer 2

f1(bool) f3(int,int)  
f2(double) f4()

Layer 3

f1() f4()  
f2(int) f3()  
f5(int)

Layer 4

f1(int) f2()

## Process Data()

Filter 1

f1(int)

Filter 2

f2(int)

Filter 3

f3(int)

Filter 4

f4(int)

Data

Pipe

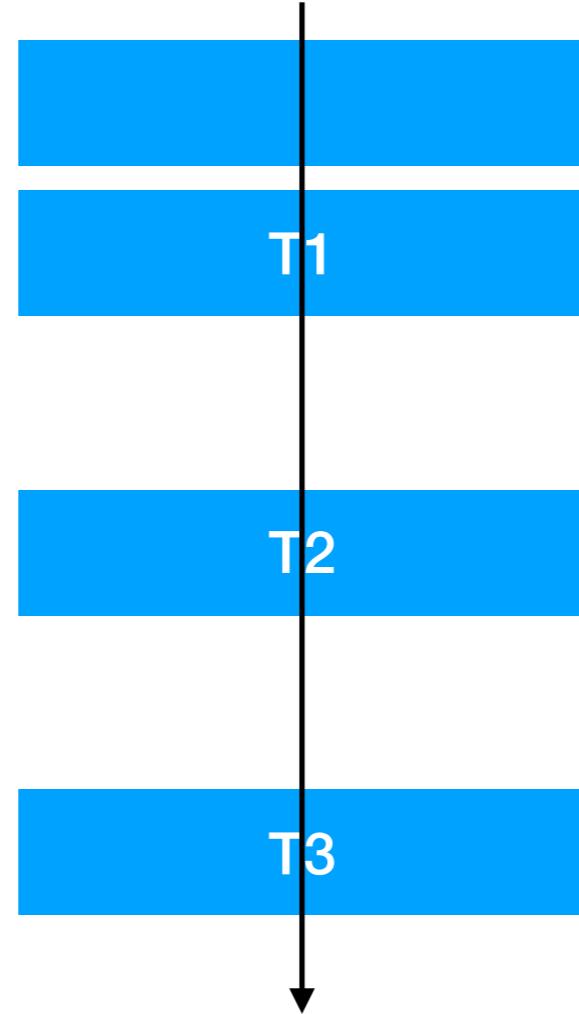
\* Can swap filters

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.

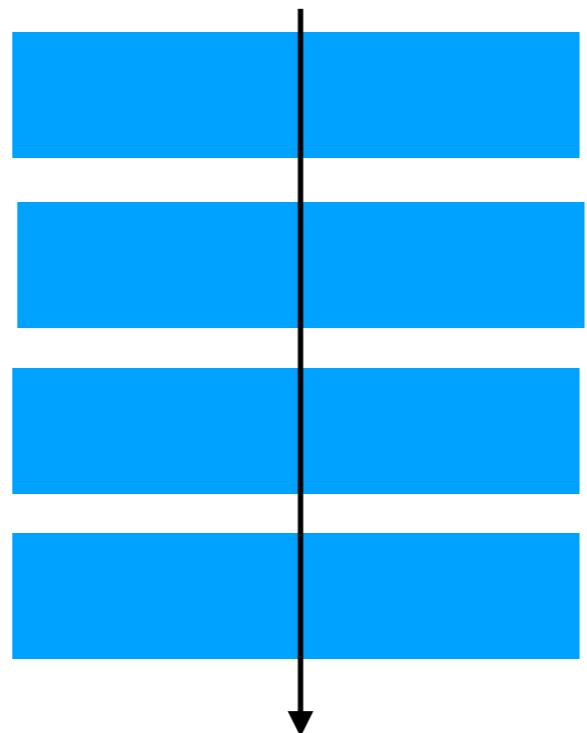
Eclipse IDE. Downloading the basic Eclipse product provides you little more than a fancy editor. However, once you start adding plug-ins, it becomes a highly customizable and useful product.



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

2

Application  
Exe

3

Process

In process  
comp

**Compile time  
monolithic**

**Link time  
monolithic**

**Runtime  
monolithic**

4

Process

Pipes  
Sockets  
http  
Roc  
...

Distributed  
comp

Process

Distributed  
comp

5

Process

Application1

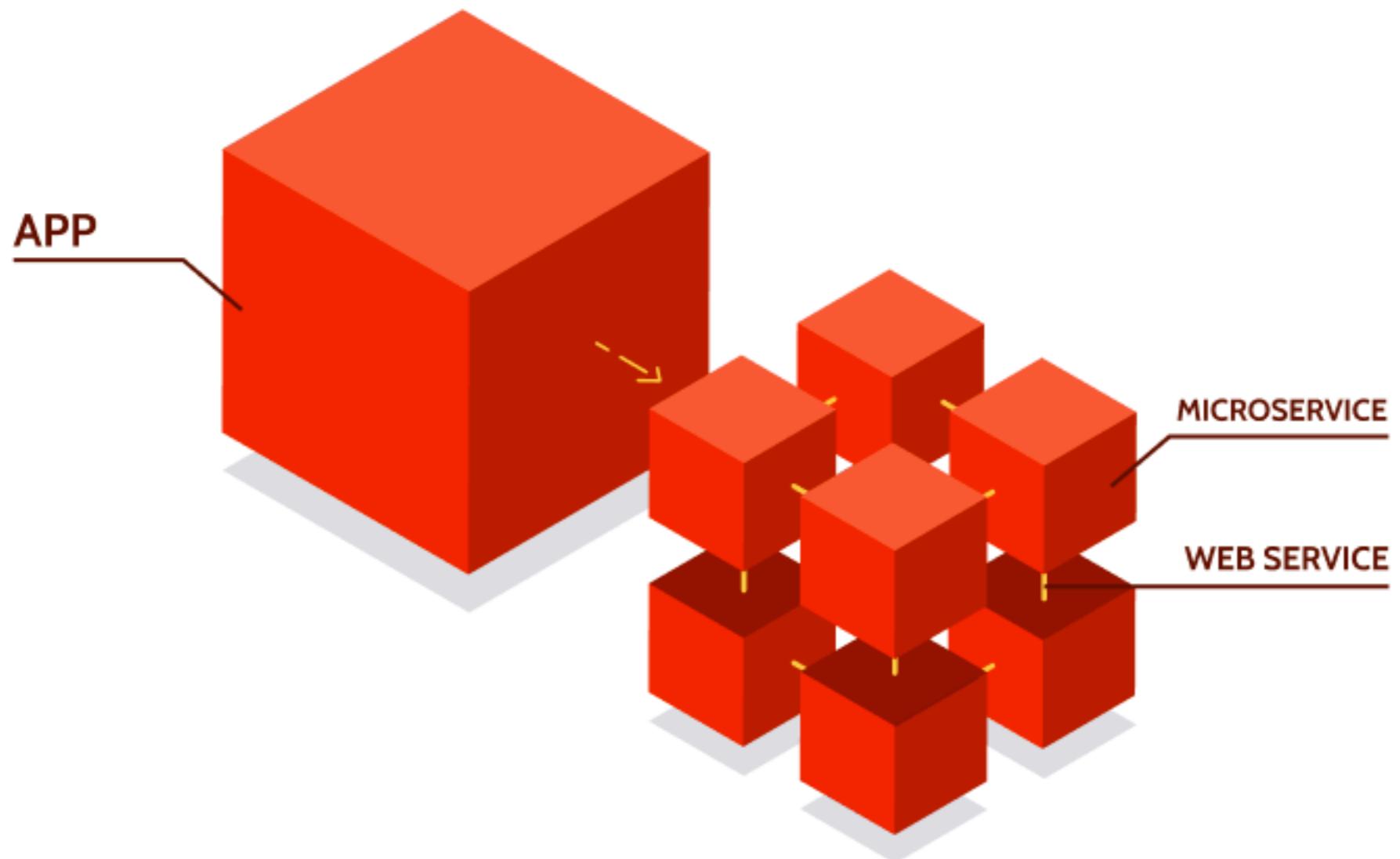
Process

Application2

**Distributed Application**

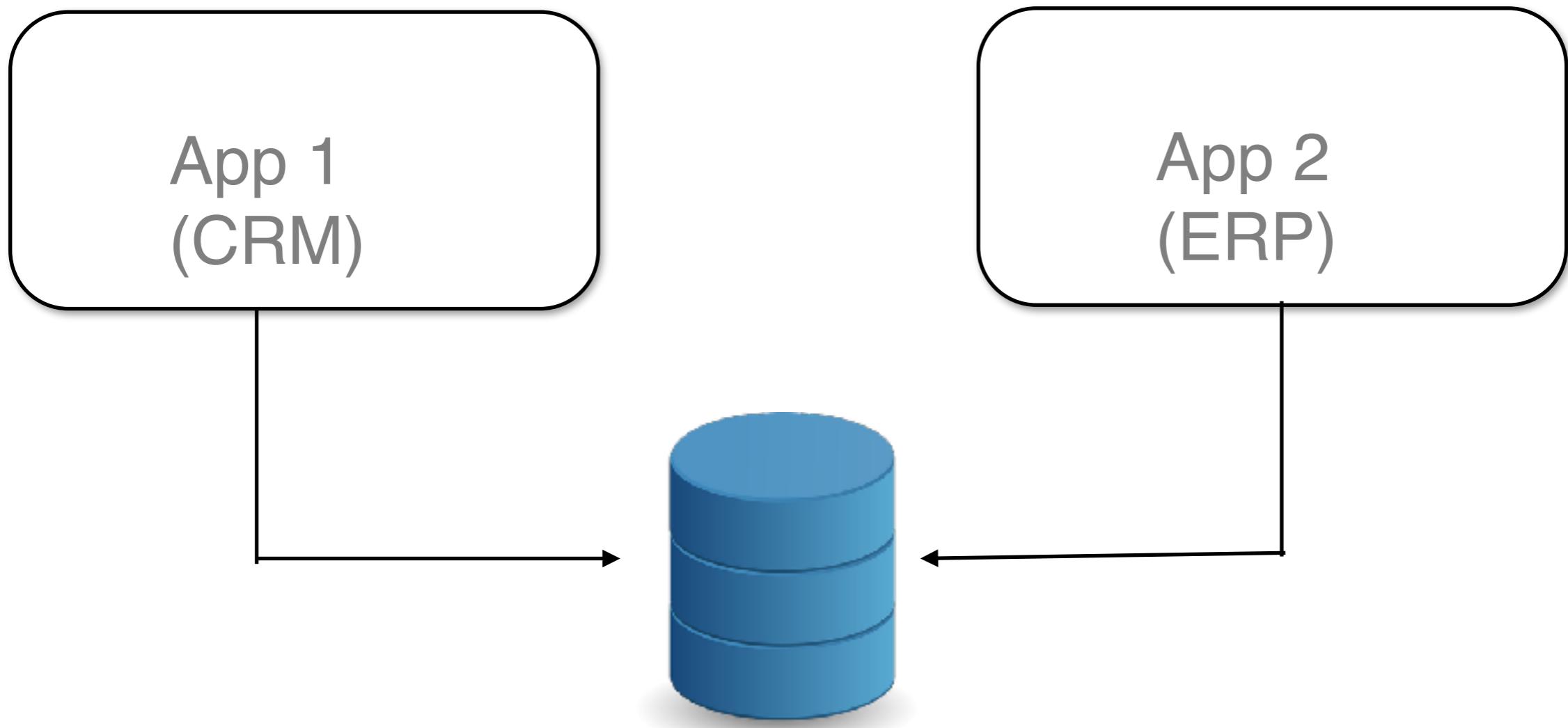
**Micro Application**

# Microservice

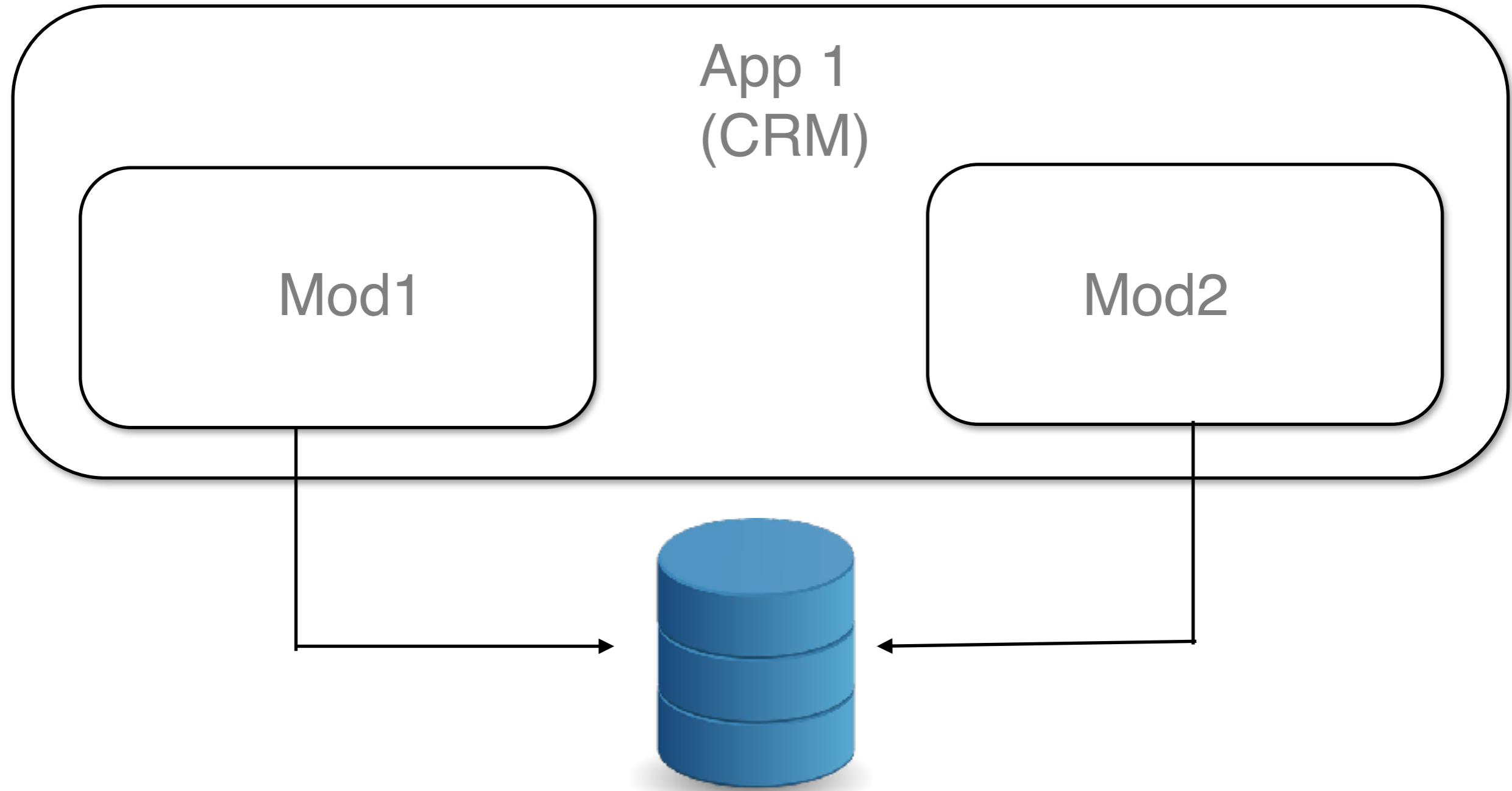


	<b>2 modules</b>	<b>2 Applications</b>	<b>W</b>	<b>Score</b>
<b>Database / Storage</b>	Shared	Not shared	2	
<b>Infra (Hosting)</b>	Share	Not Shared	3	
<b>Sorce Control</b>	Shared	Its Own	2	
<b>CI/CD (Build Server)</b>	Shared	Its Own	3	
<b>Fun Requirements</b>	Shared	Its Own	1	
<b>SCRUM Team / Sprint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

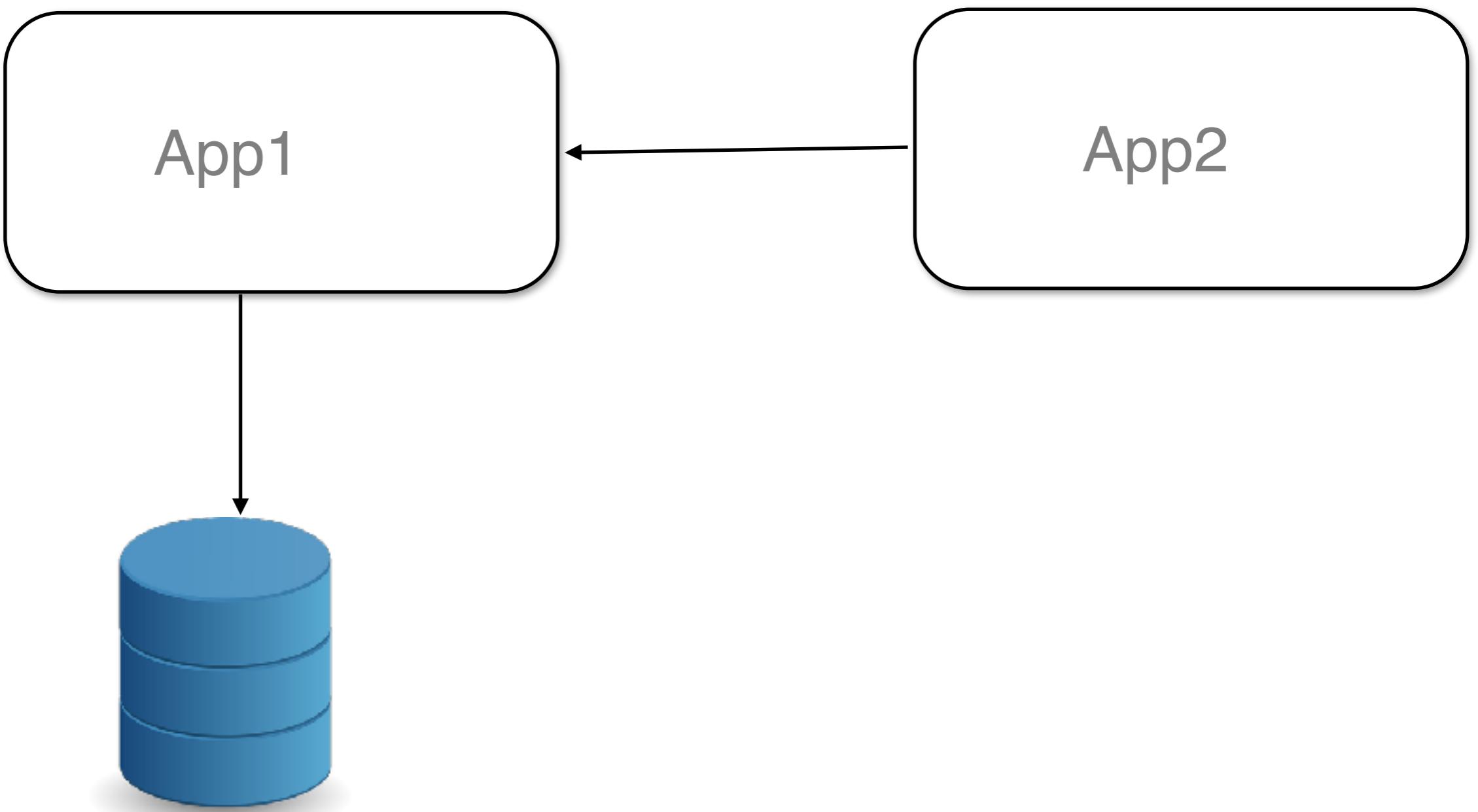
# 1. Database



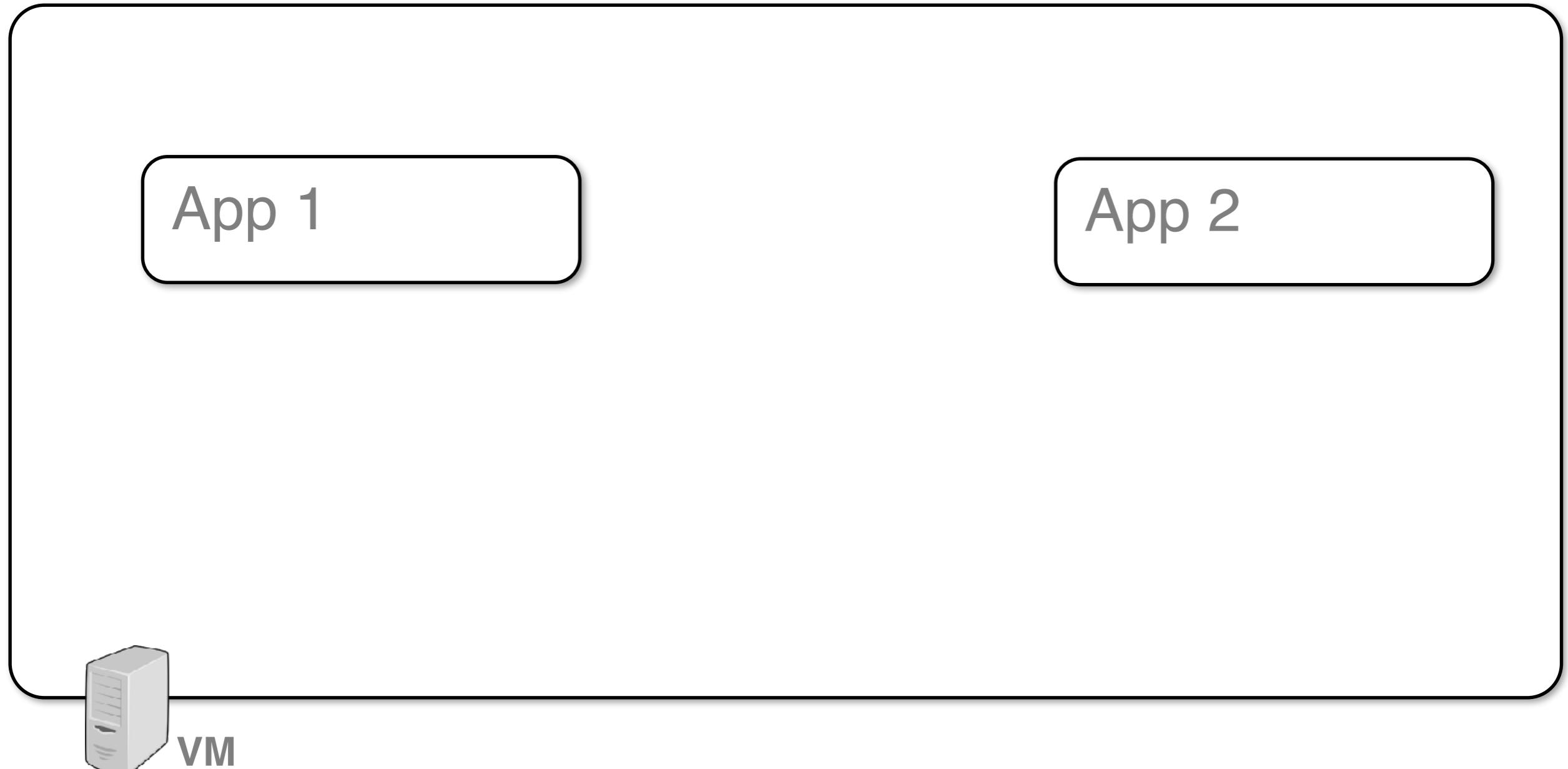
# 1. Database



# 1. Database



## 2. Infrastructure



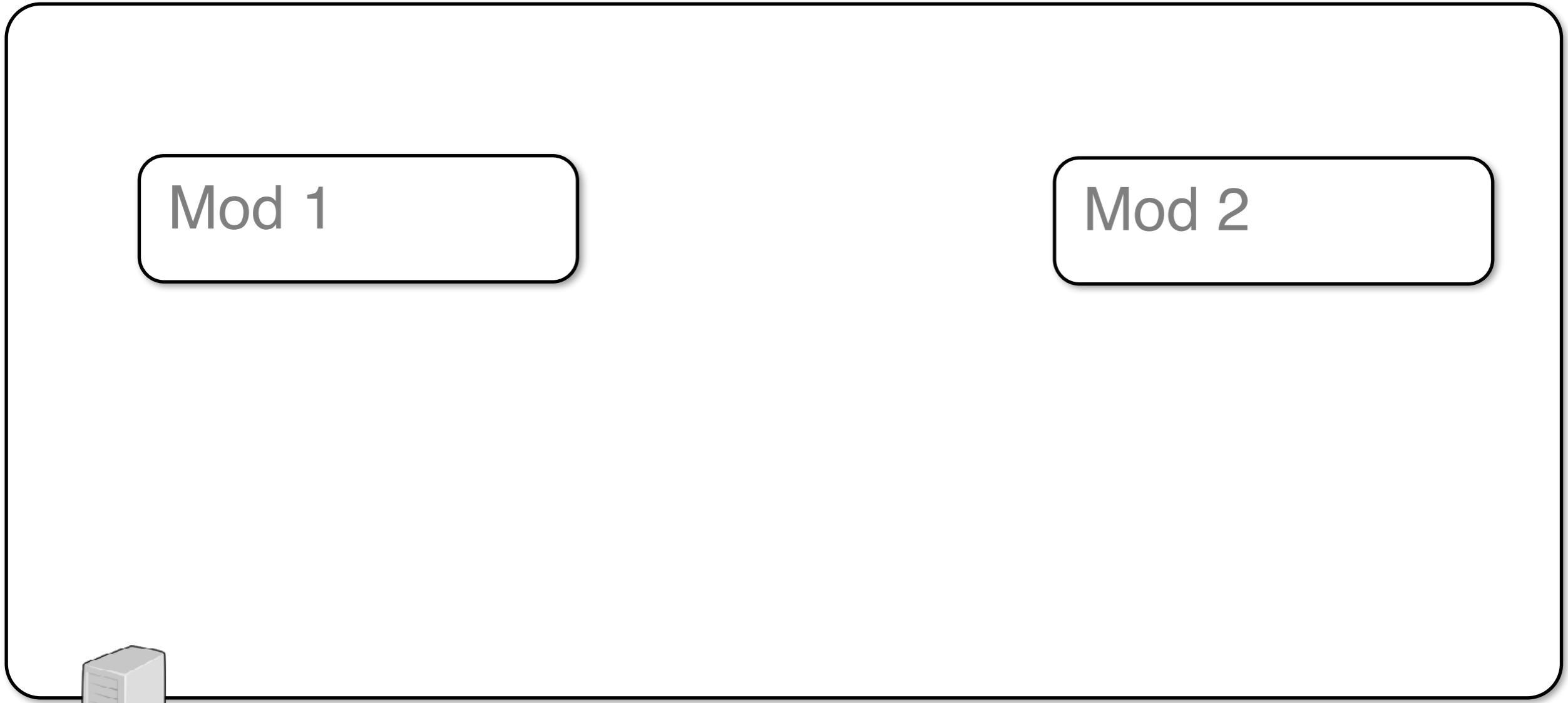
## 2. Infrastructure

Mod 1

Mod 2



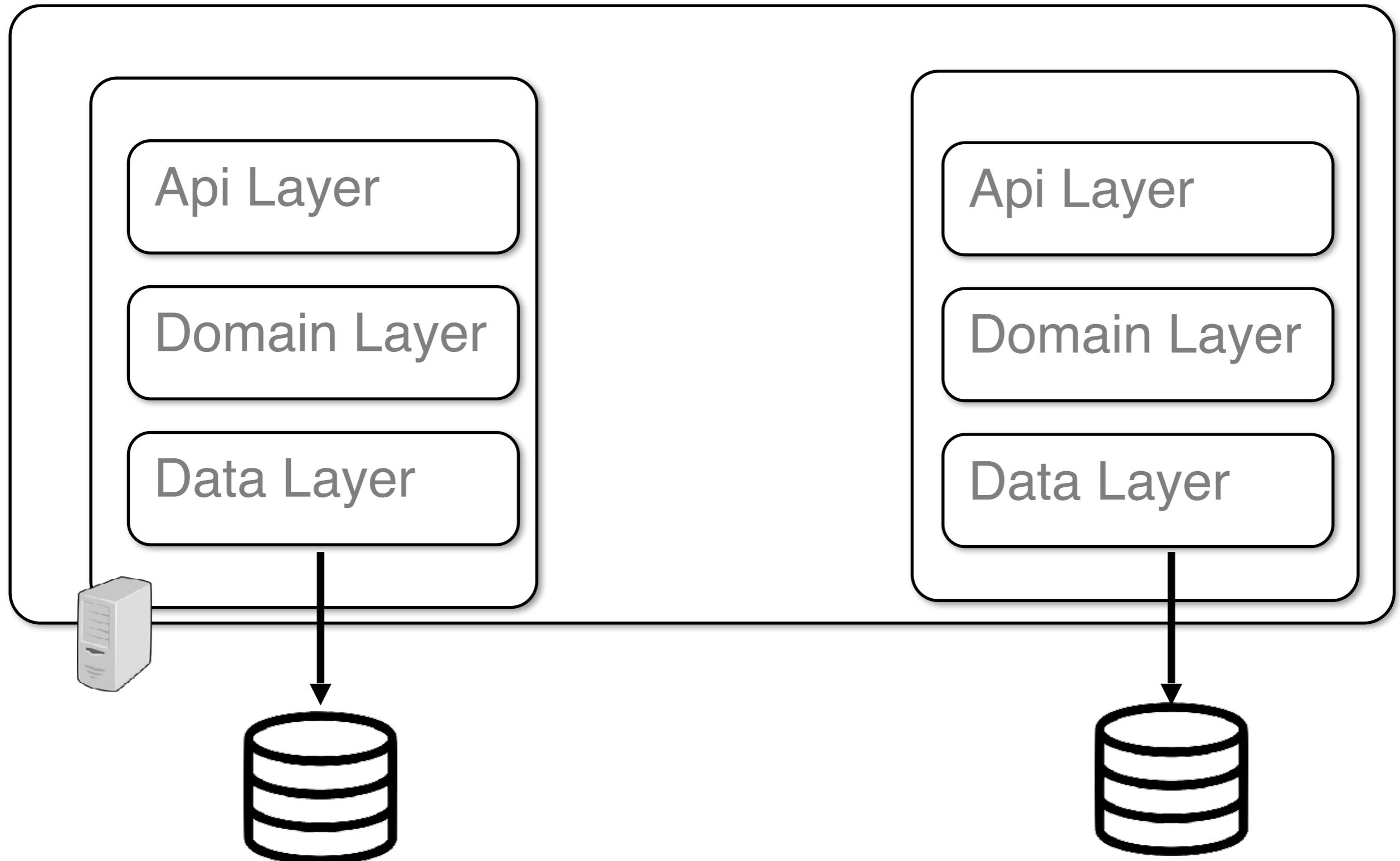
VM



## 2. Infrastructure

Web Server

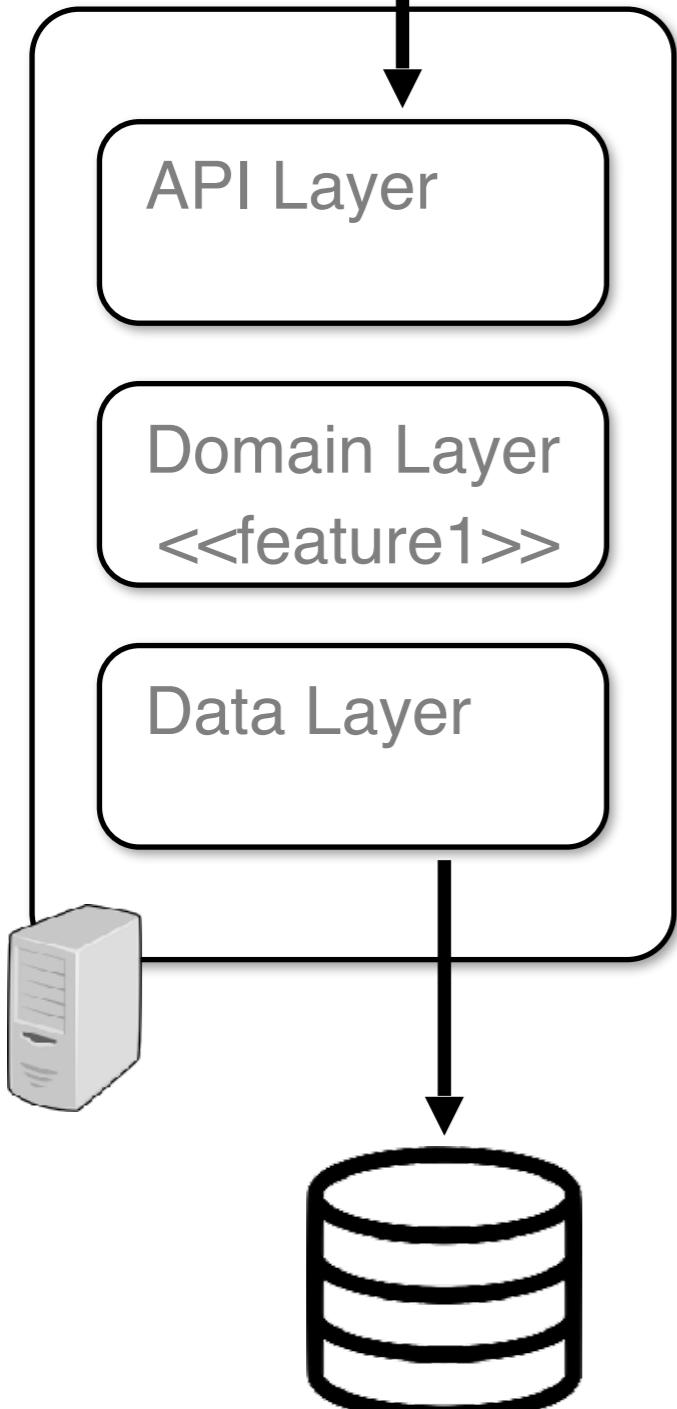
Comp1



1

UI App - SPA

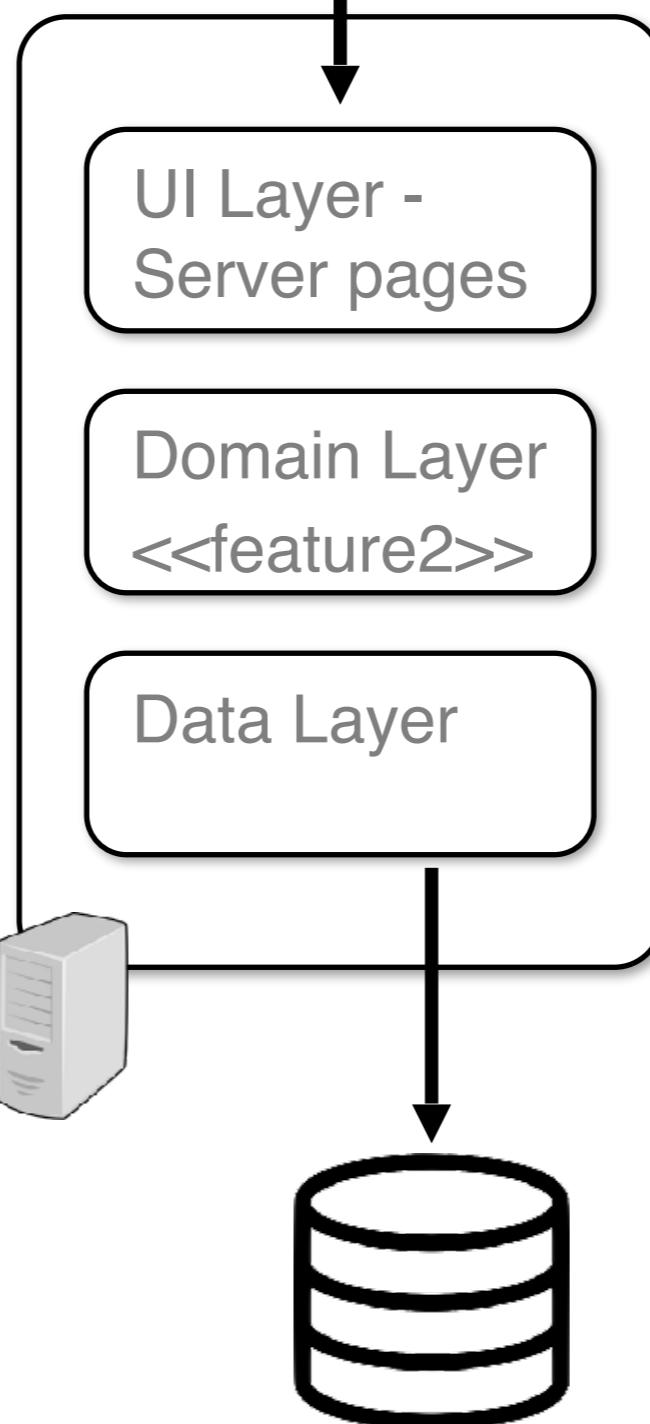
Web Server



2

Browser

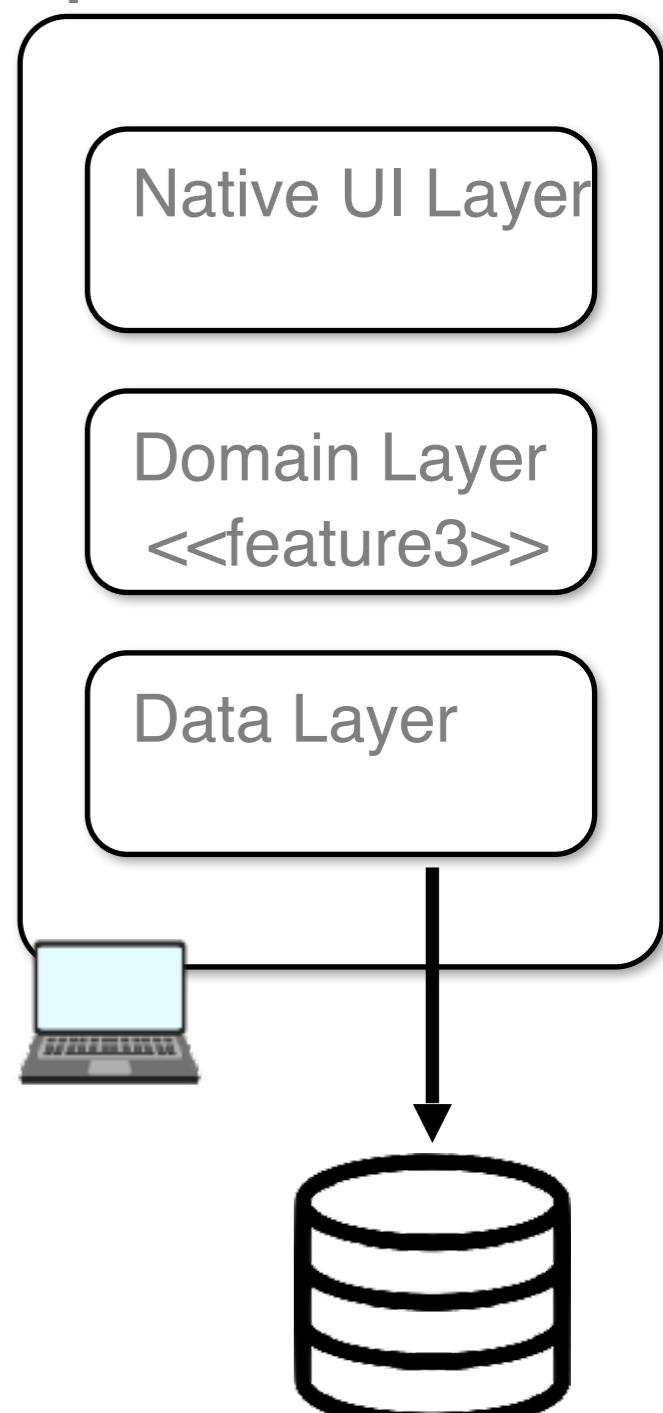
Web Server



3



Desktop

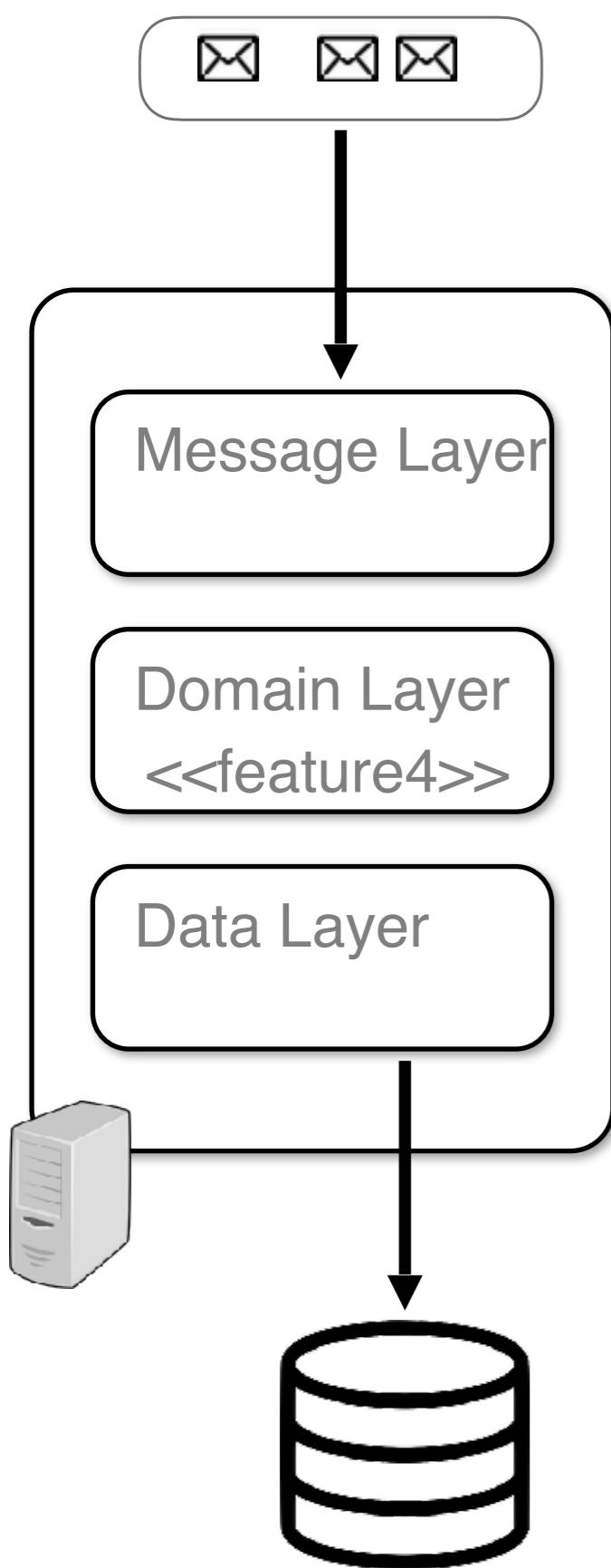


Angular (SPA)

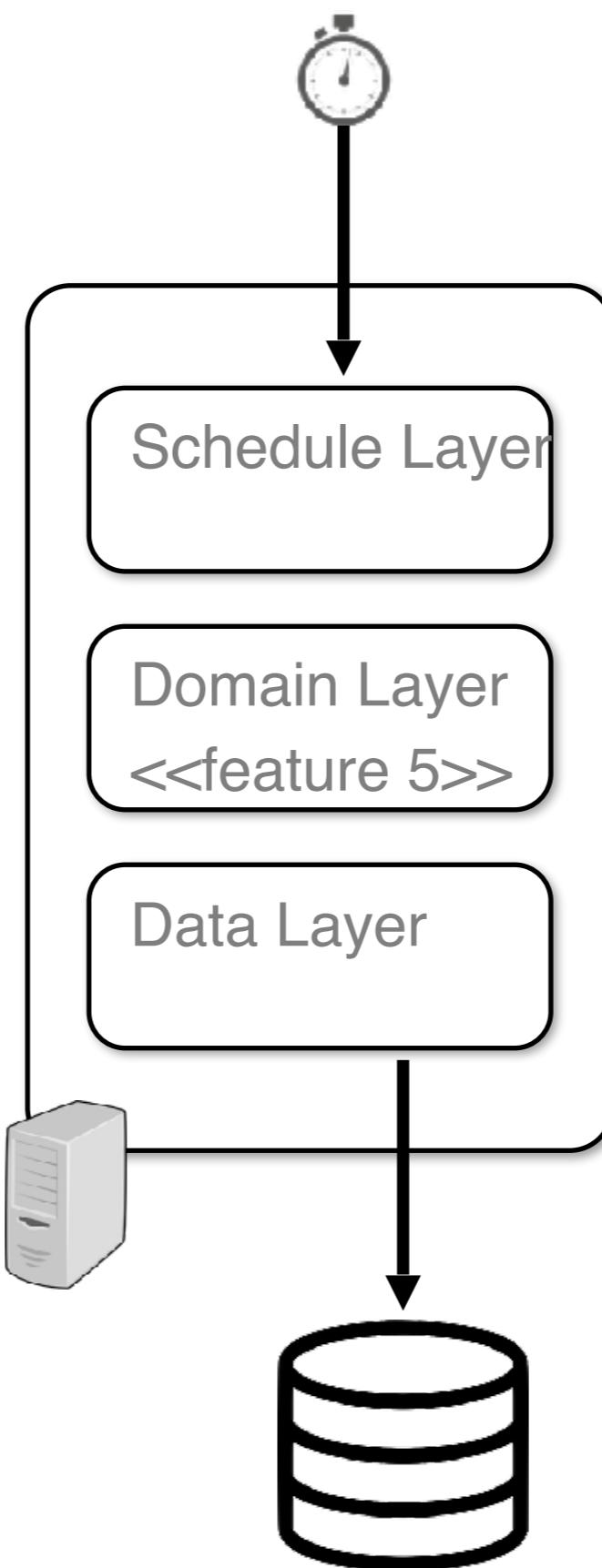
JSP, Servlet, JSF (Server pages)

Swing, Android, Ios (Native)

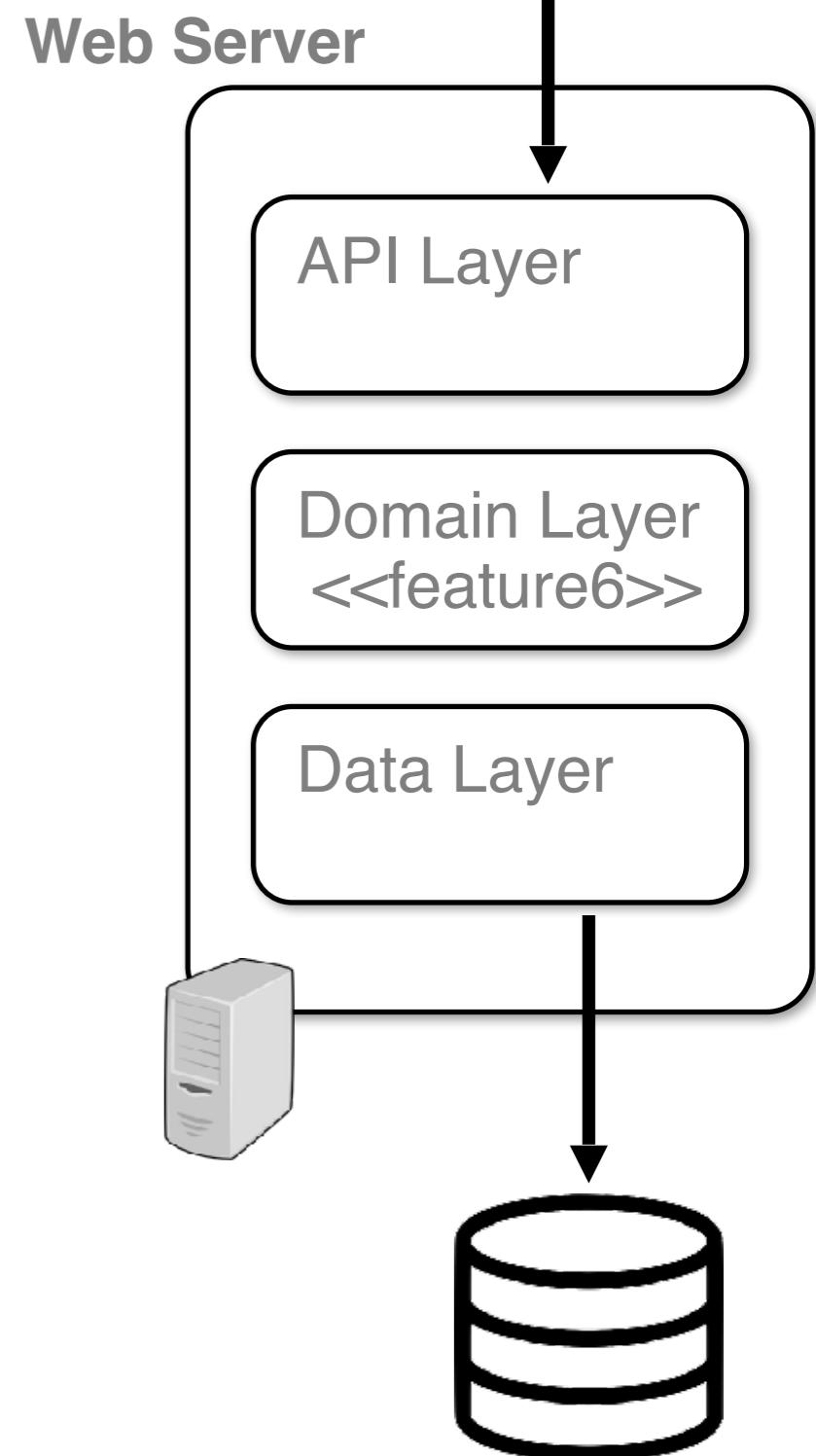
1



2



3



1

Application  
Exe

2

Application  
Exe

3

Process

In process  
comp

**Compile time  
monolithic**

**Link time  
monolithic**

**Runtime  
monolithic**

4

Process

Process

Exe

DII/so/jar

Distributed  
comp

5

Process

Process

Exe

Exe

DII/so/jar

Application1

Application2

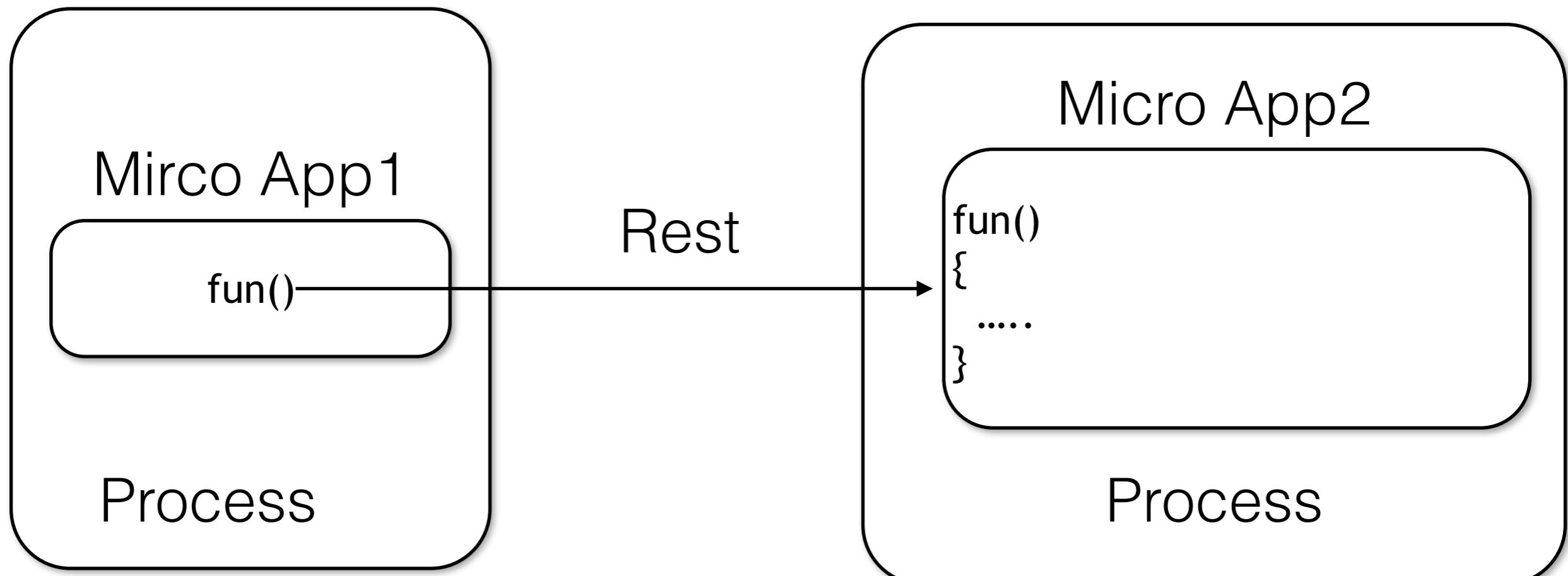
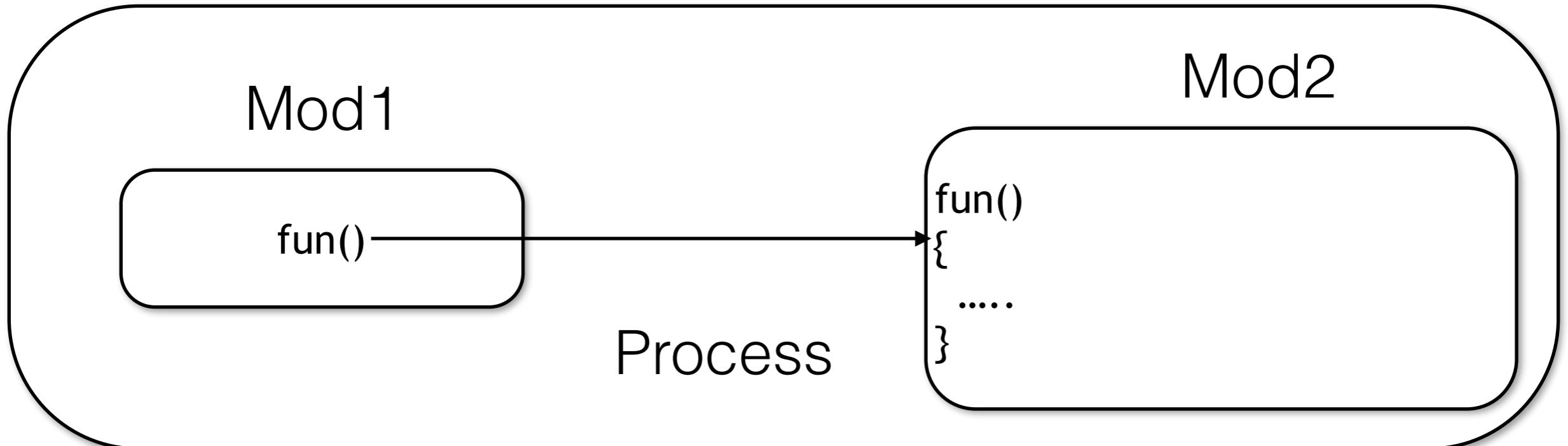
**Distributed Application**

**Micro Application**

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

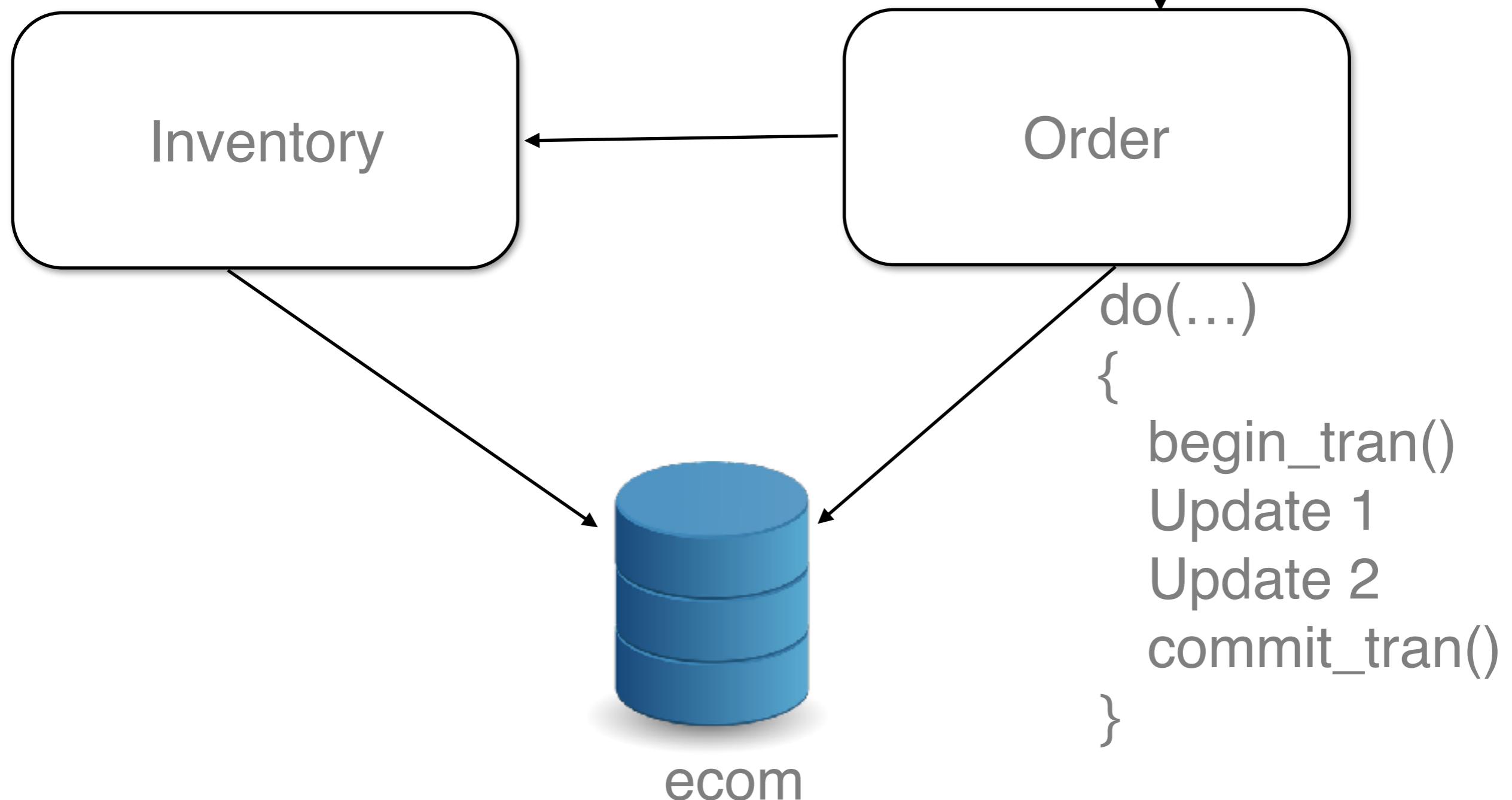
	CPU Cycles
a + b	3 cpu
Fun call	10 cpu cycles
Create Thread	200,000 cpu cycles
Destroy thread	100,000 cpu cycles
Write file	10,00,000
API Call	20,00,000
Db call	45,00,000

# 4. Development time

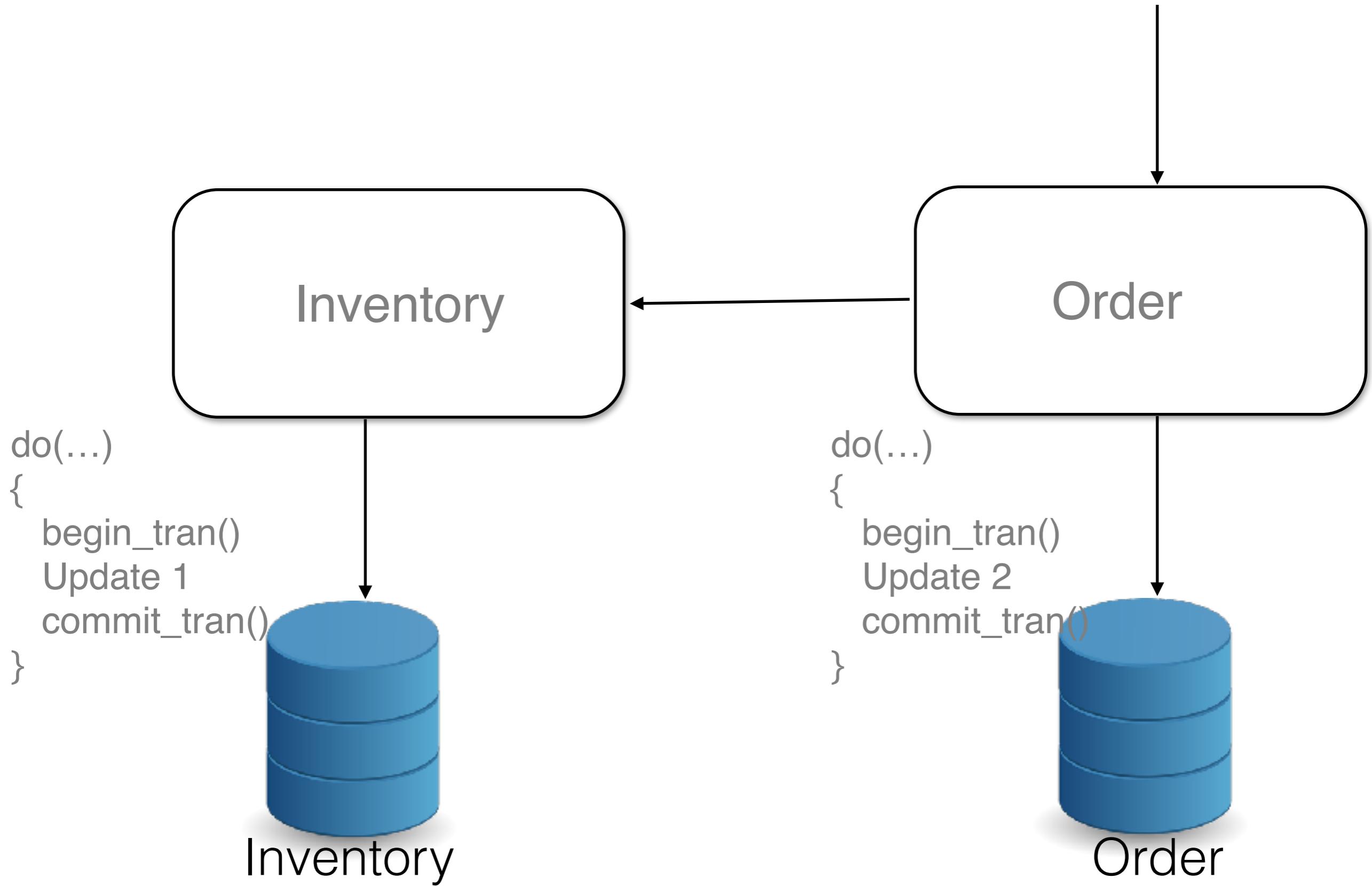


A + B	3
Fun call	10 - 20
Create Thread	2,00,000
Write File (I/O)	10,00,000
Network call	10,00,000 - 50,00,000

## 2. Db transaction

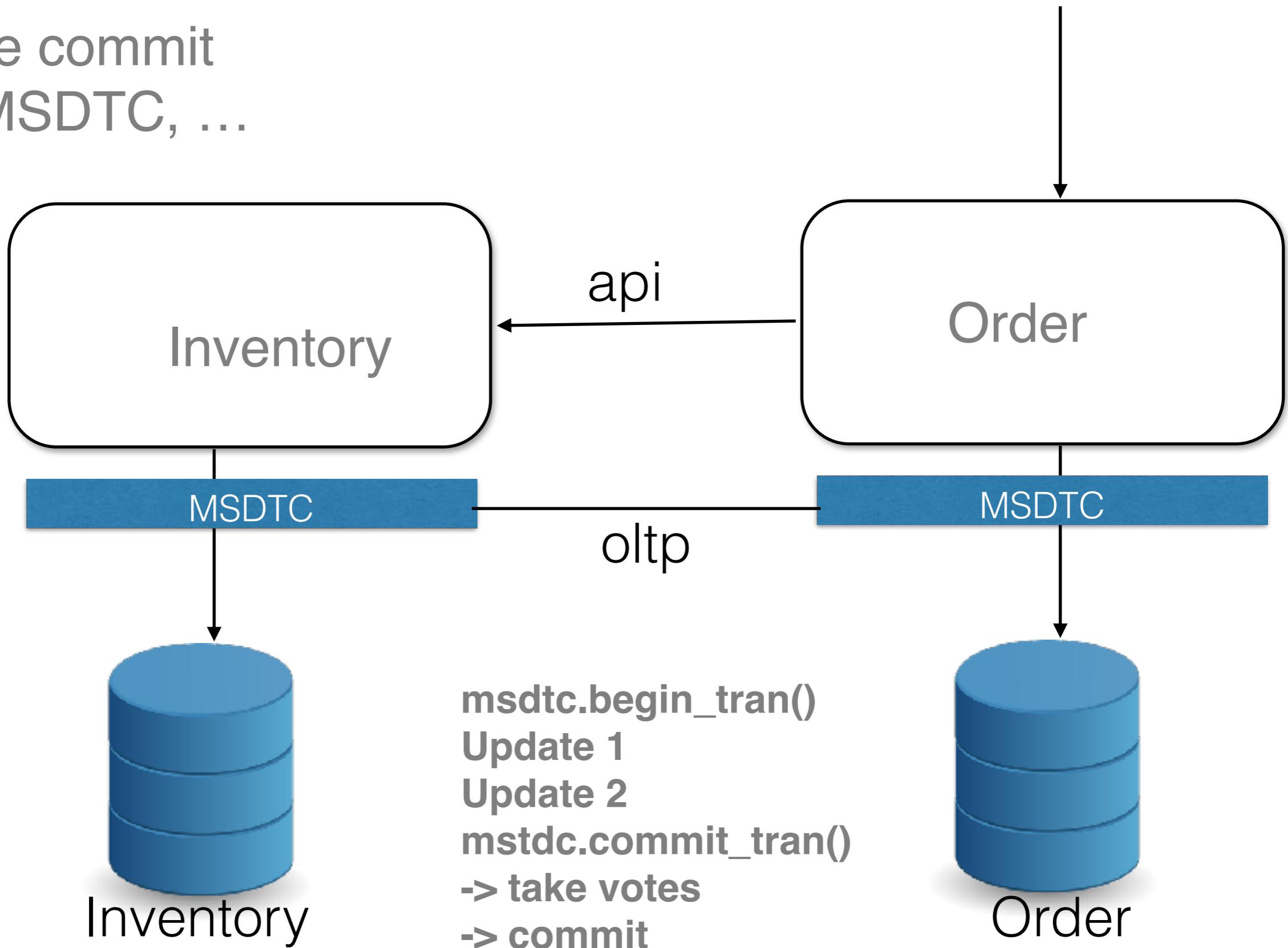


## 2. Db transaction

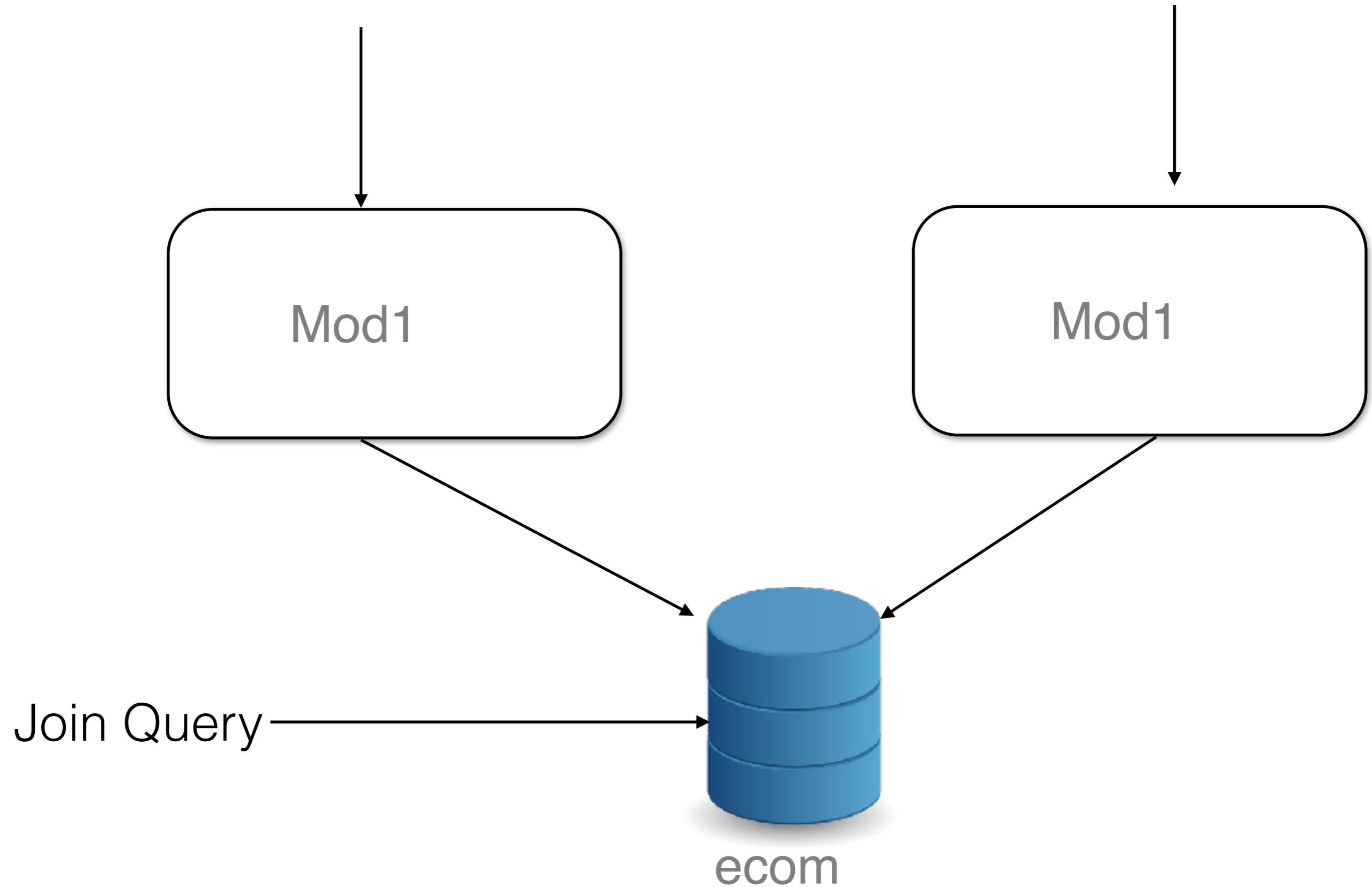


## 2. Db transaction

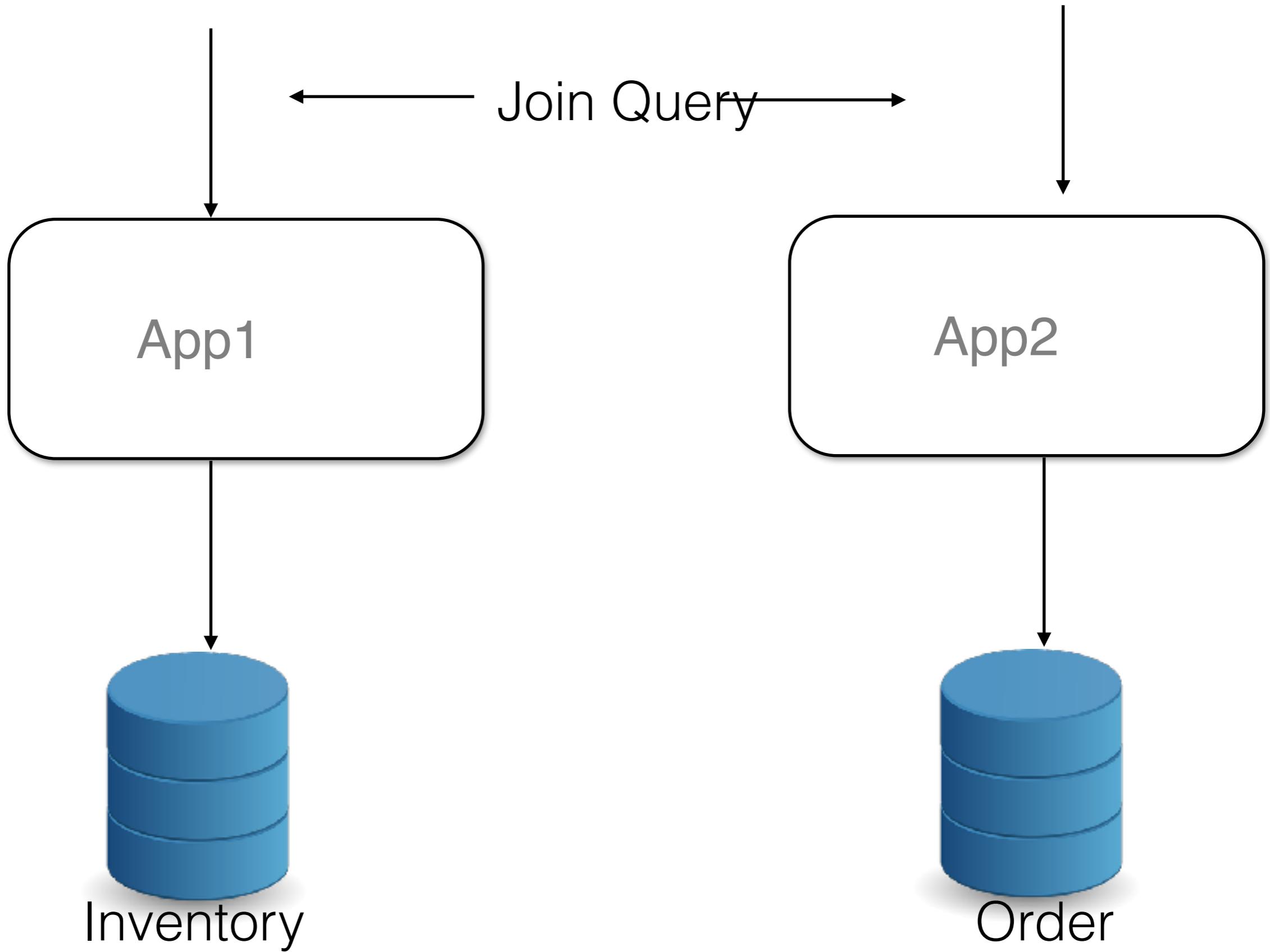
2 phase commit  
JTX , MSDTC, ...



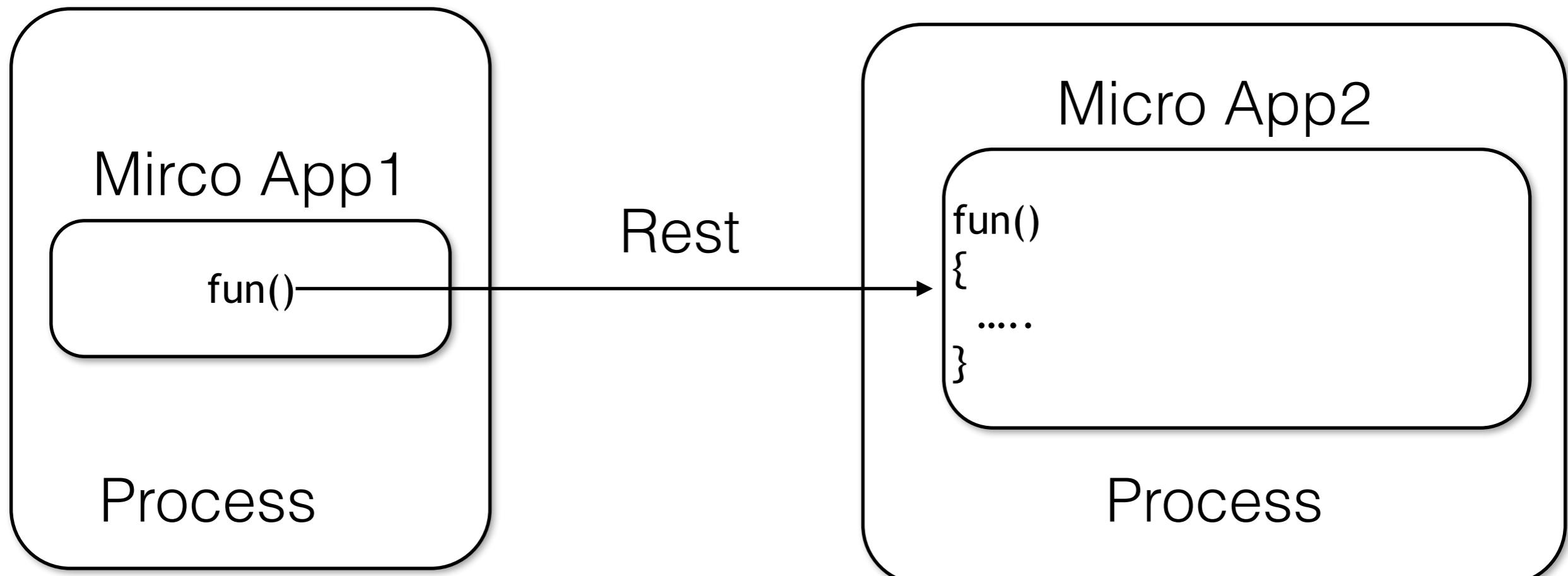
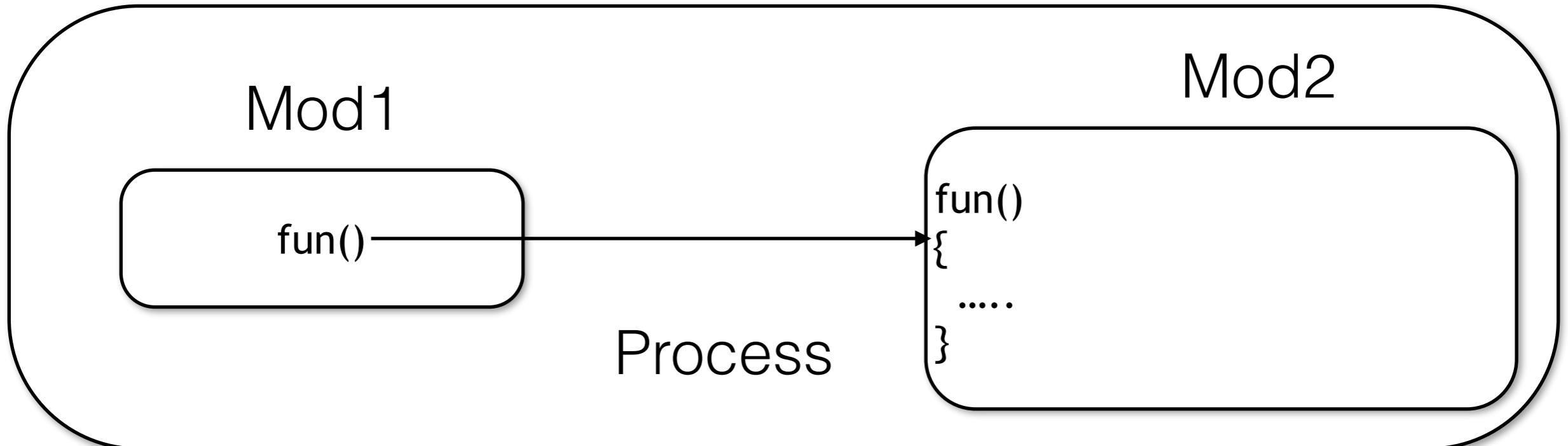
### 3. Db query



# 3. Query

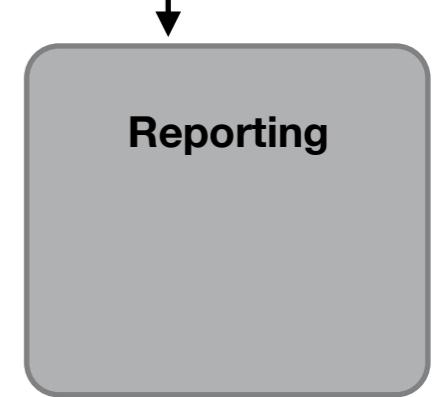
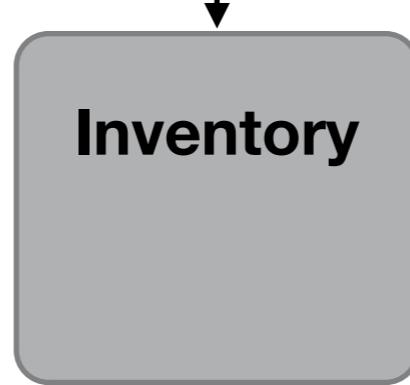
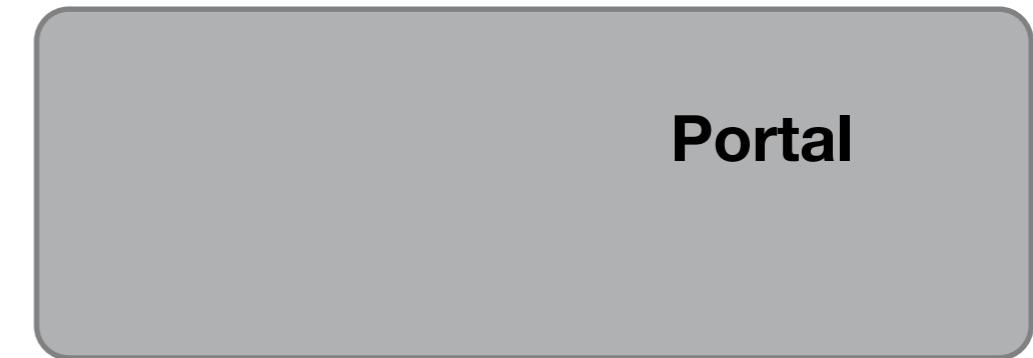


# 4. Development time

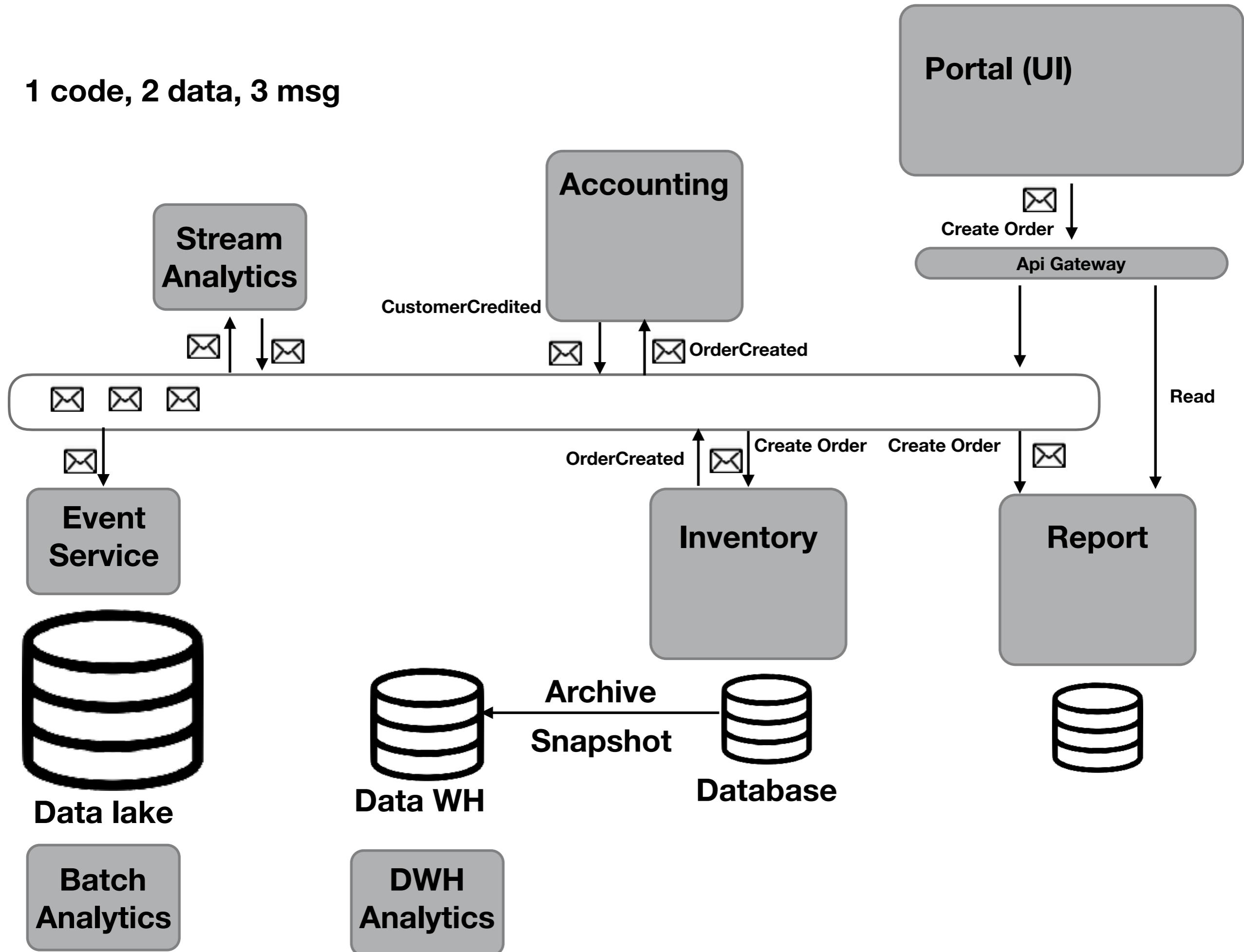


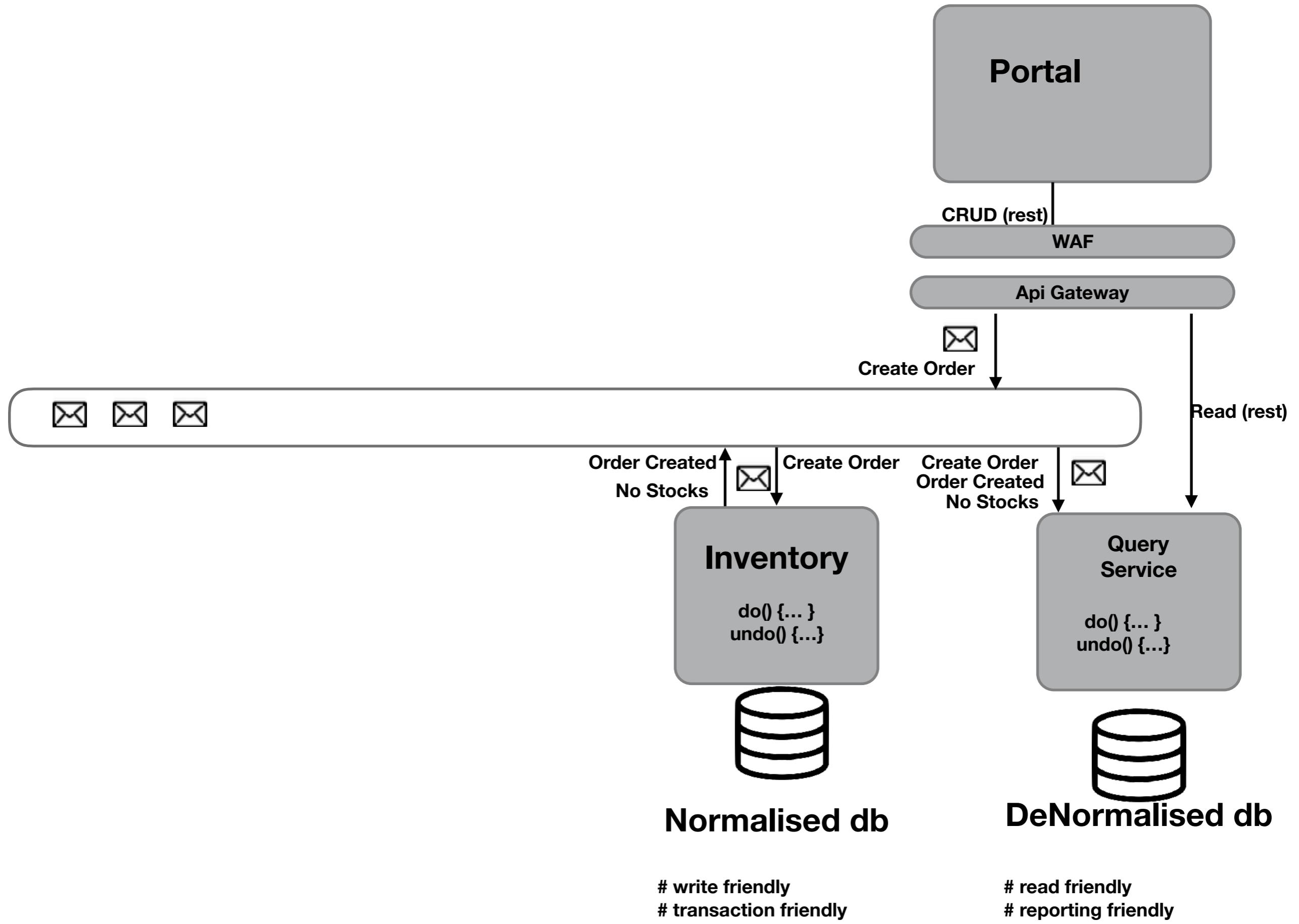
# Analytics App





1 code, 2 data, 3 msg

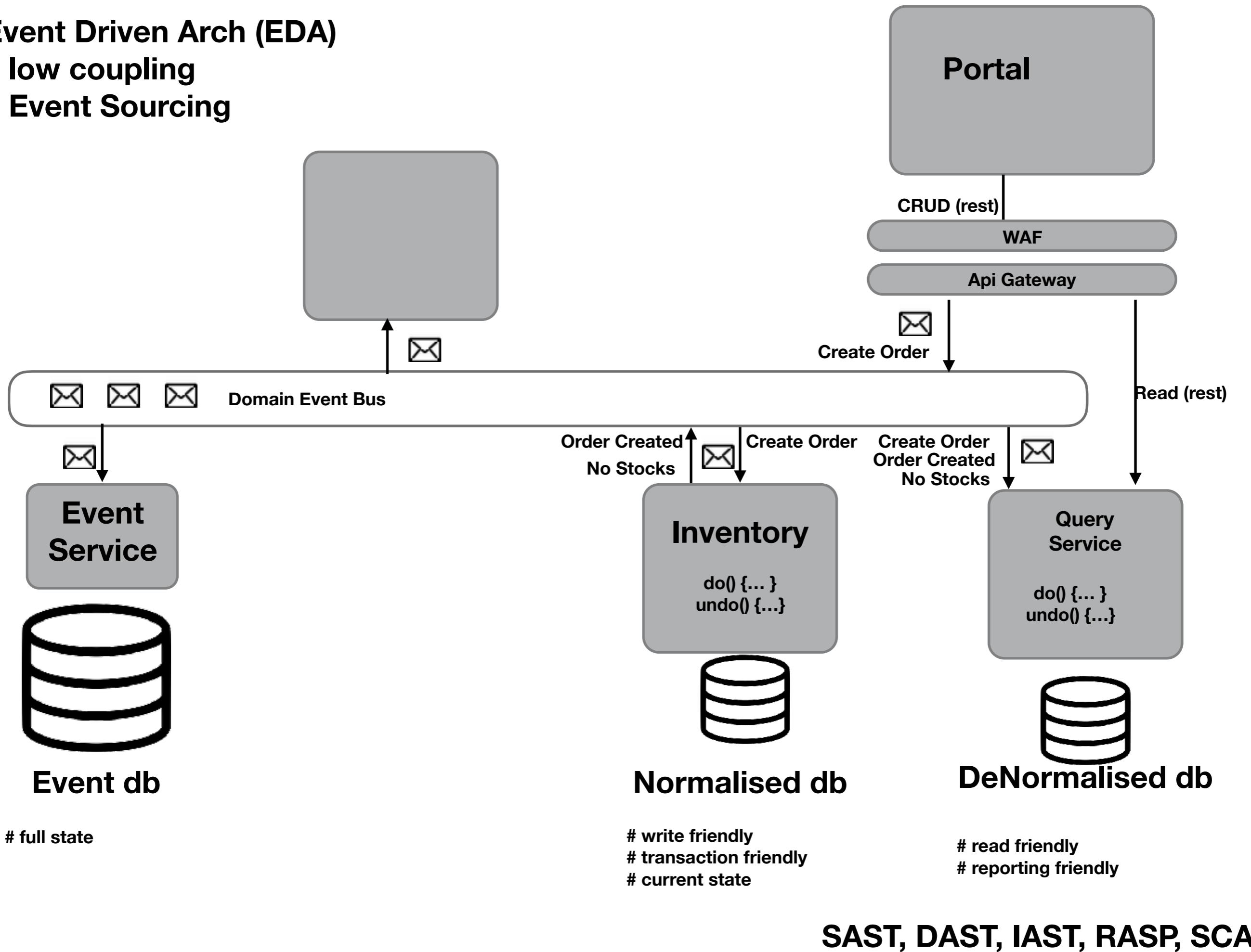


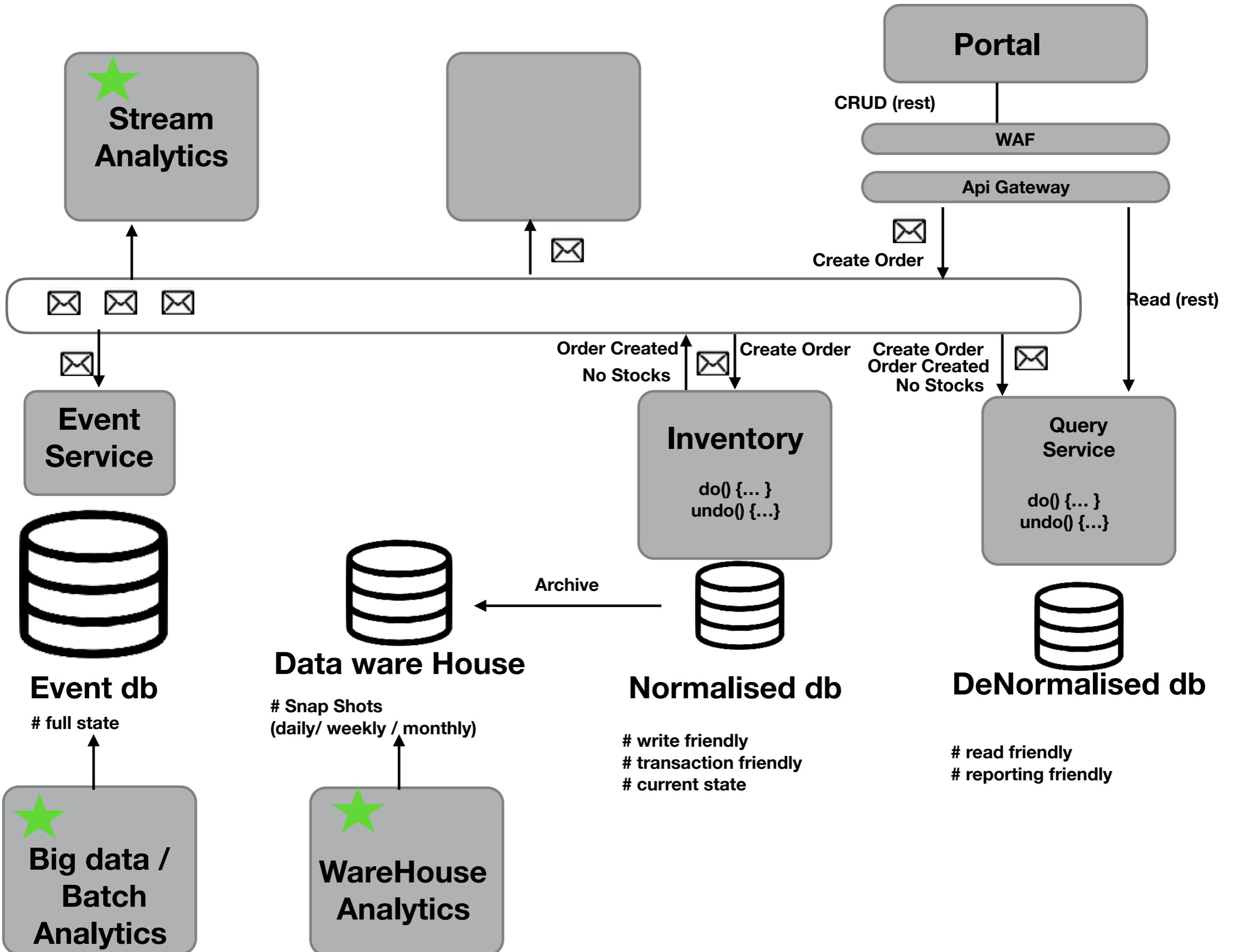


# Event Driven Arch (EDA)

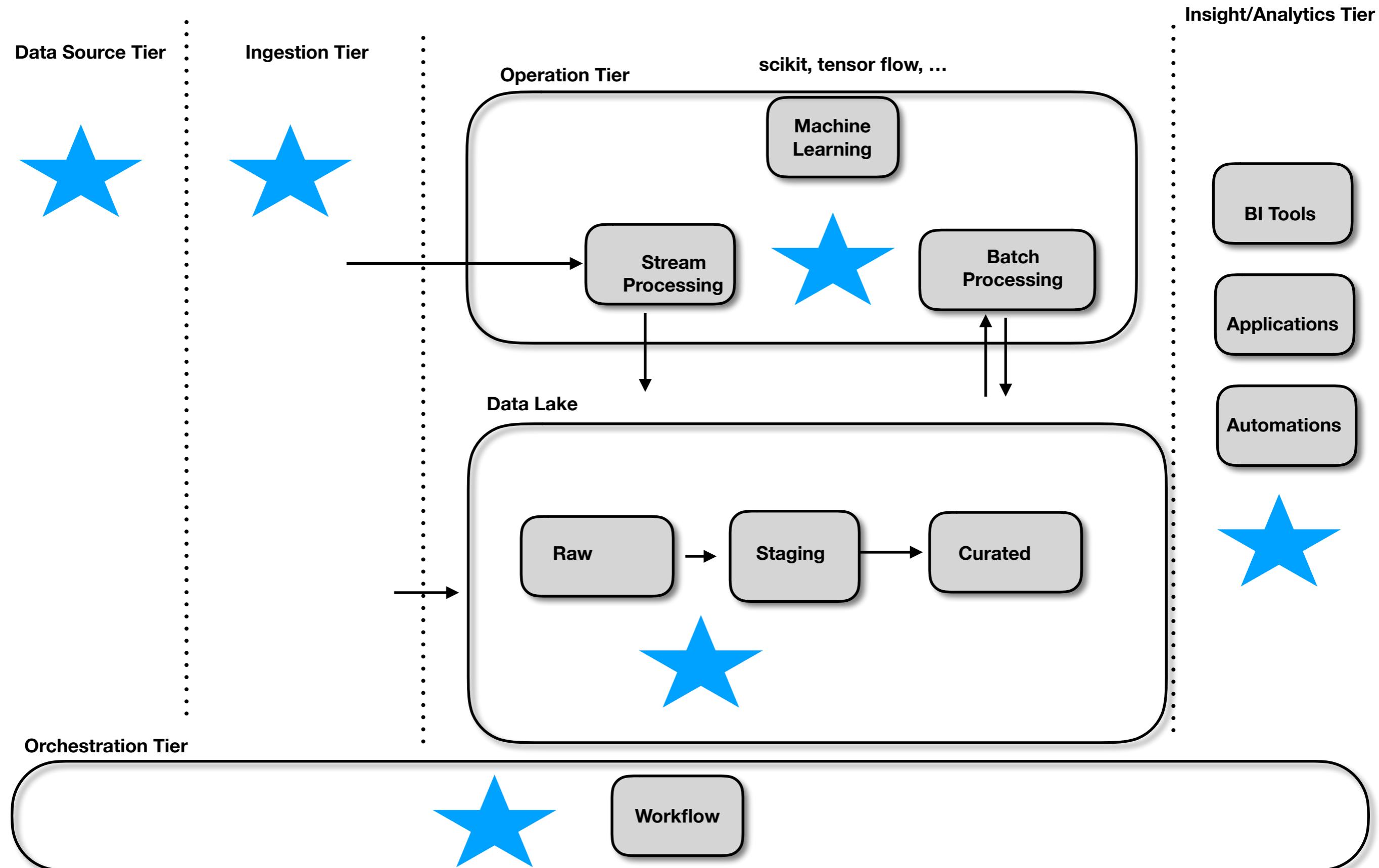
# low coupling

# Event Sourcing



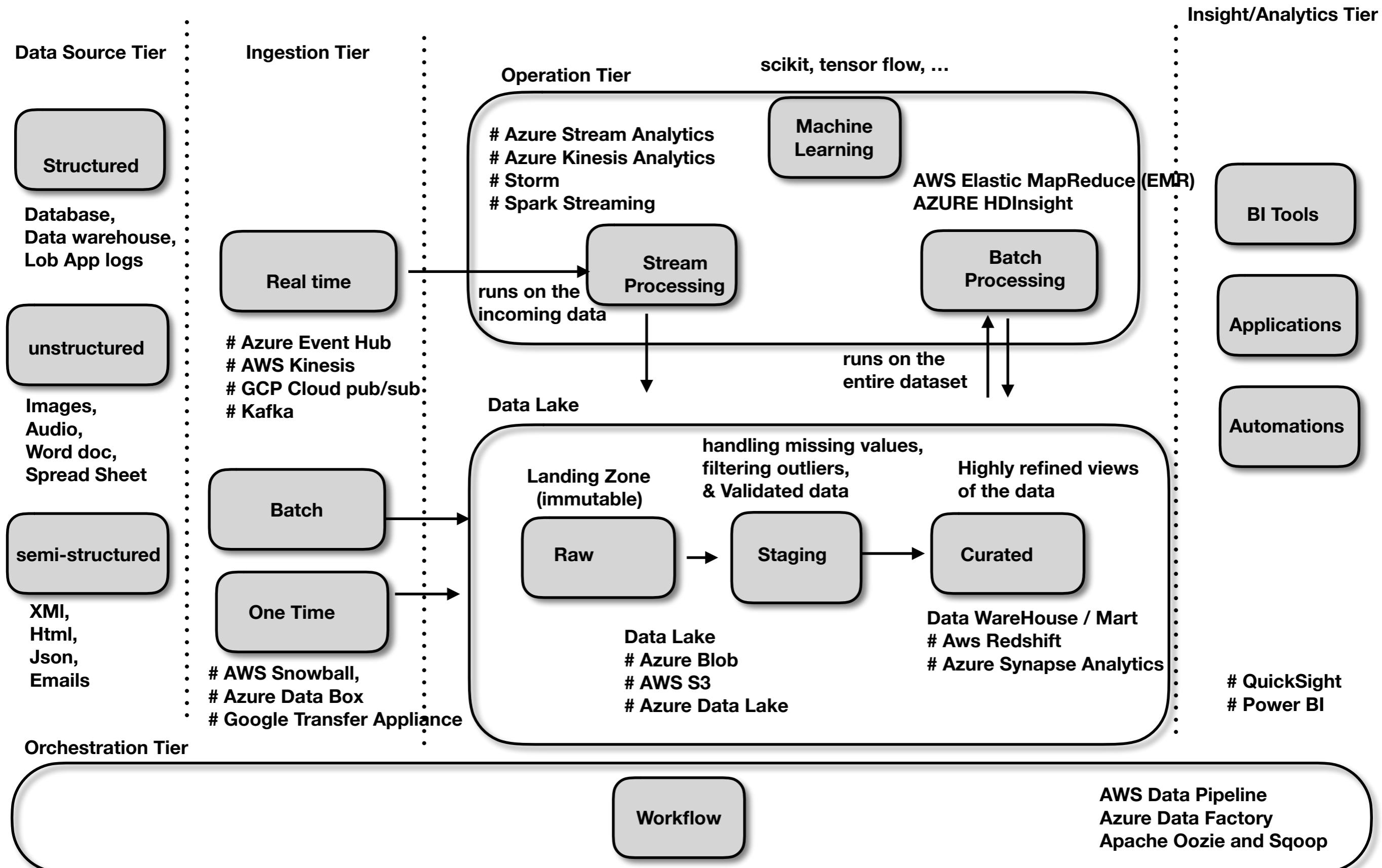


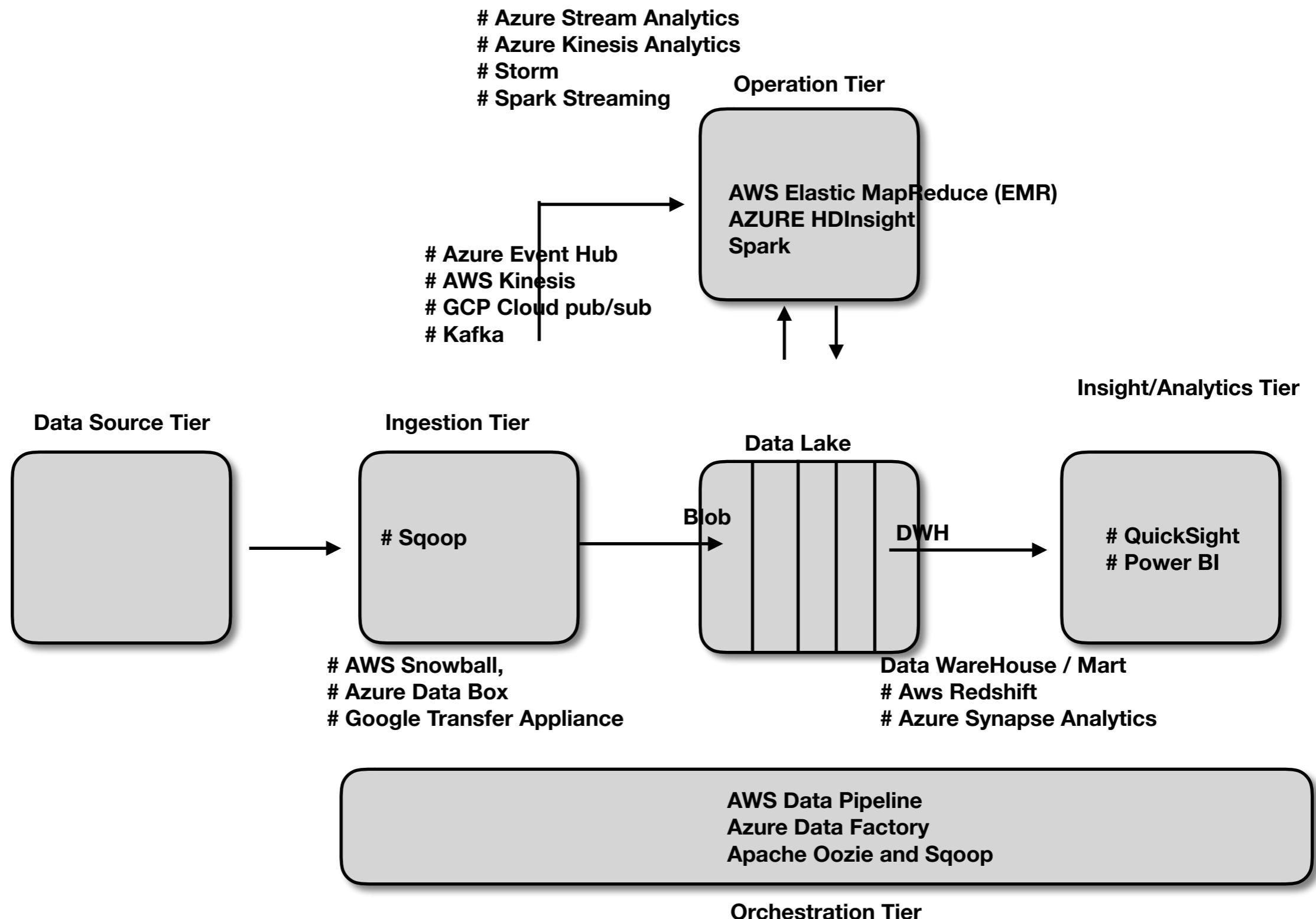
# Reference Architecture Analytical Application



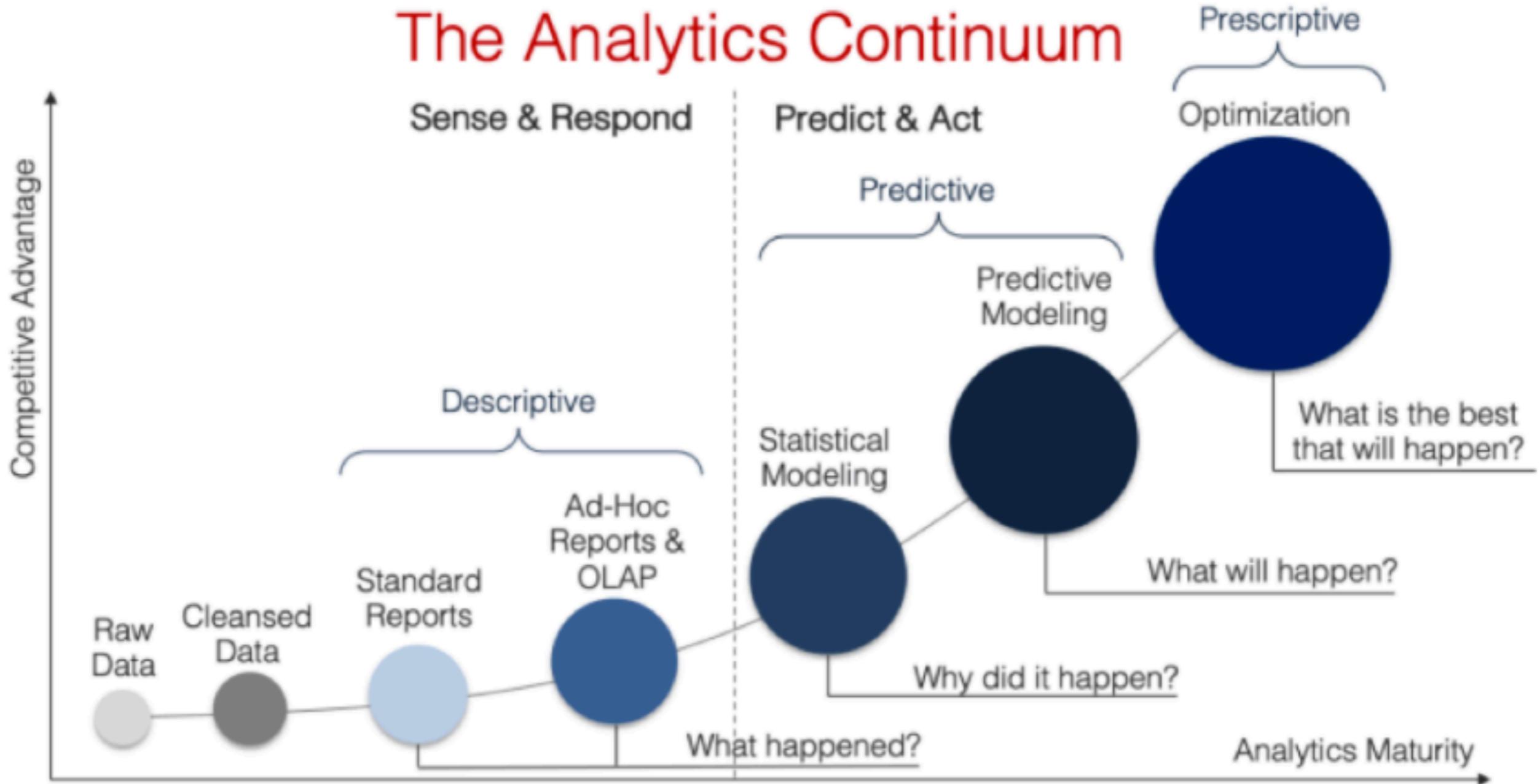
# Reference Architecture

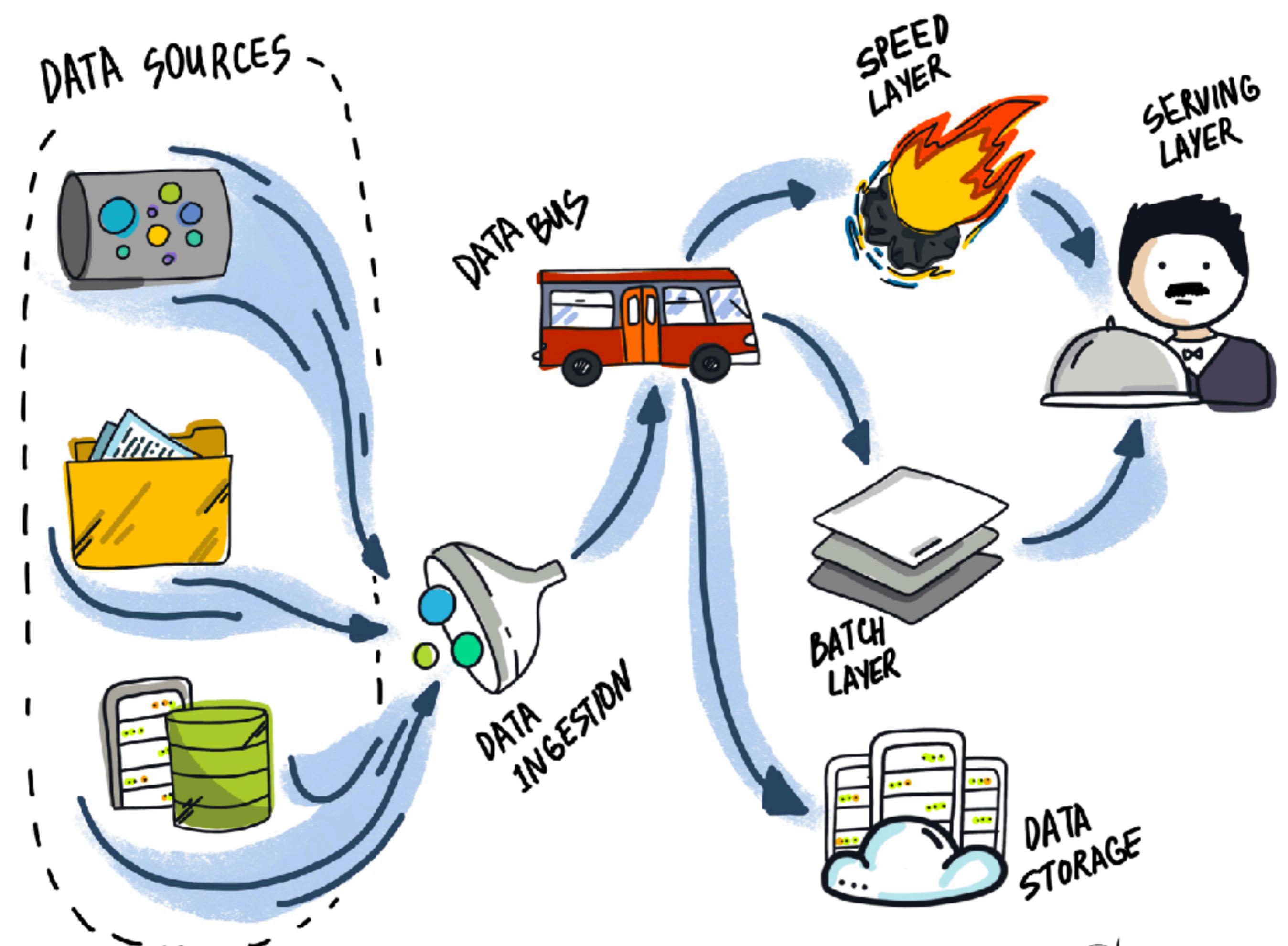
## Analytical Application

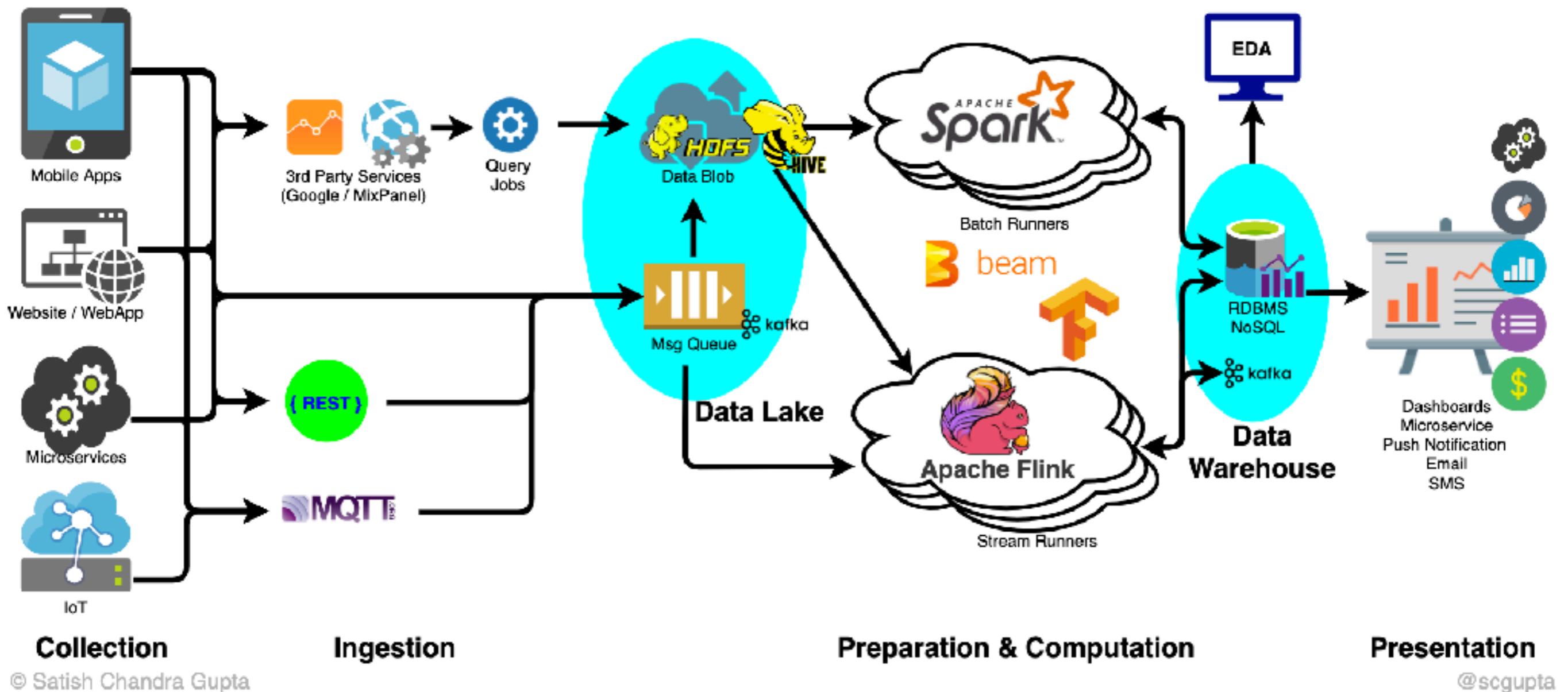


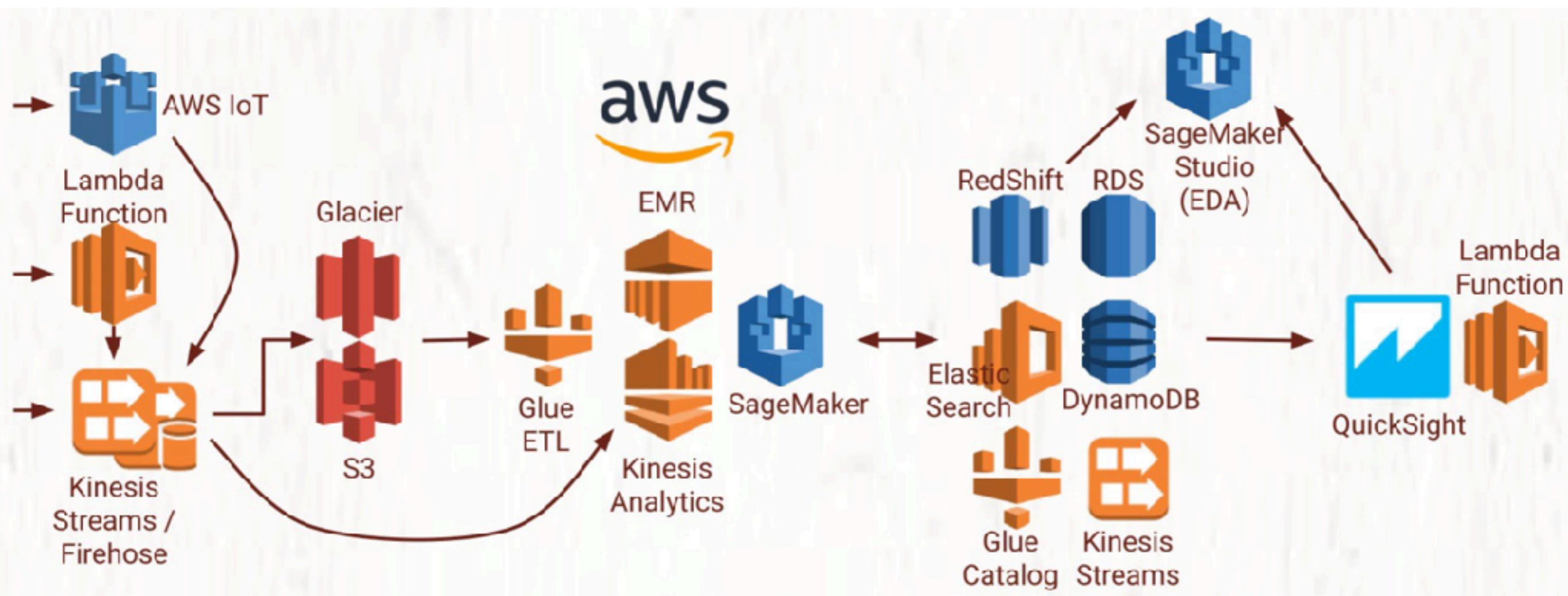


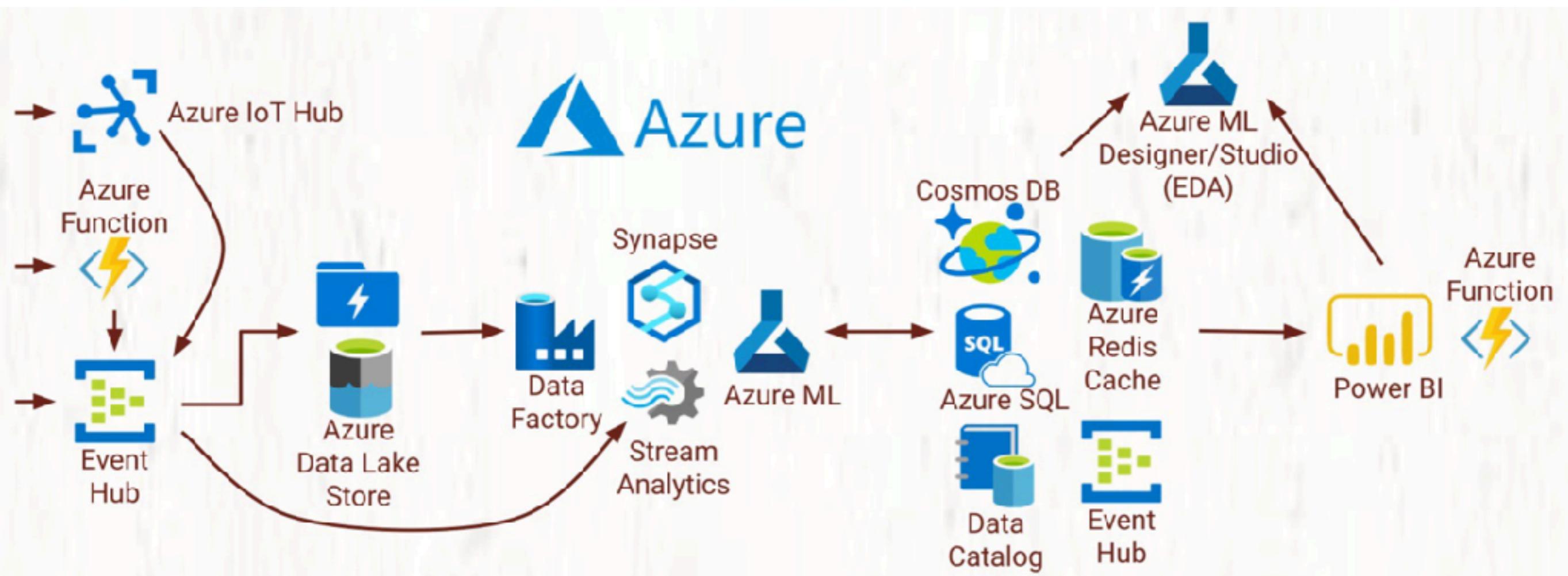
# The Analytics Continuum

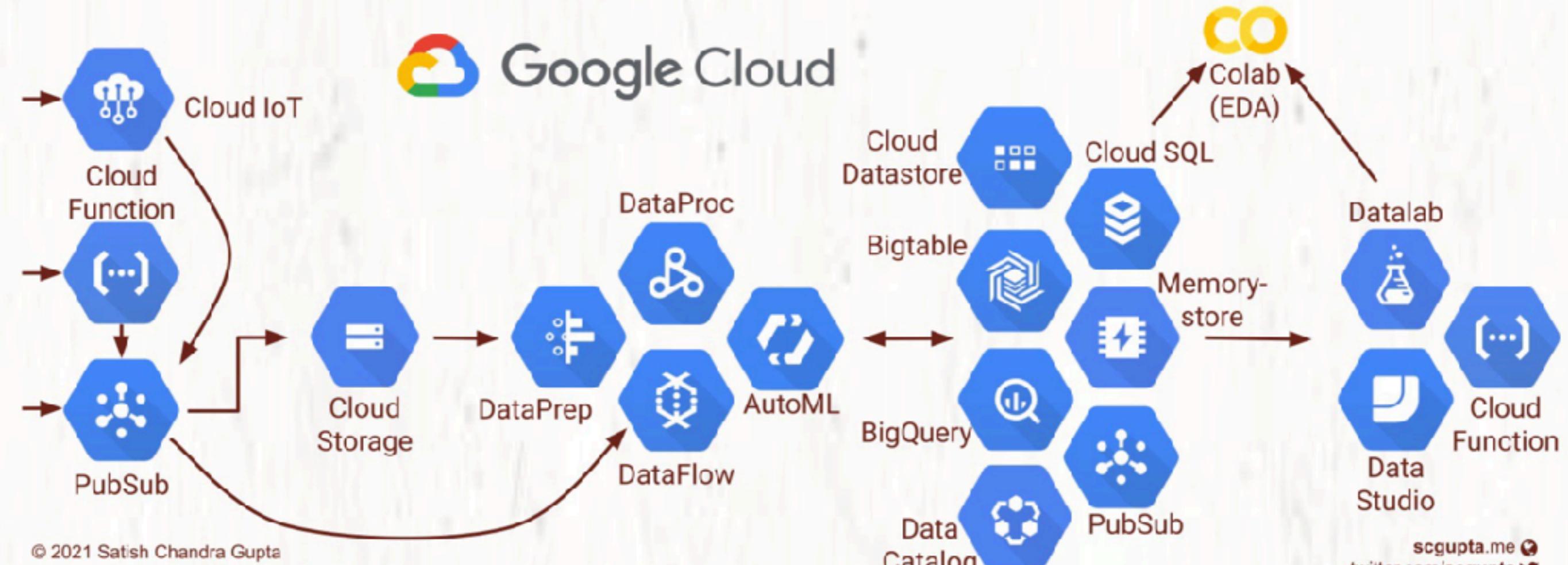












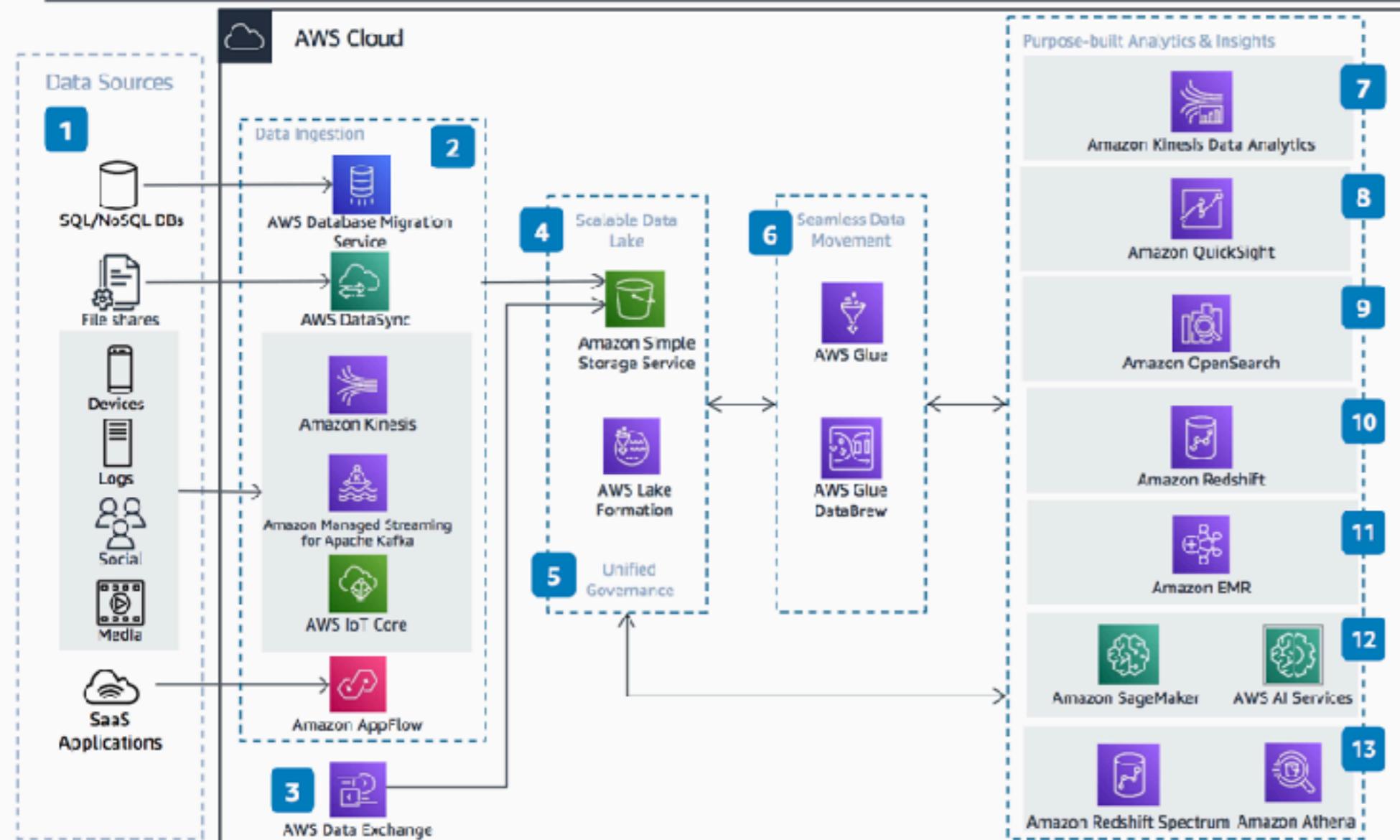
© 2021 Satish Chandra Gupta

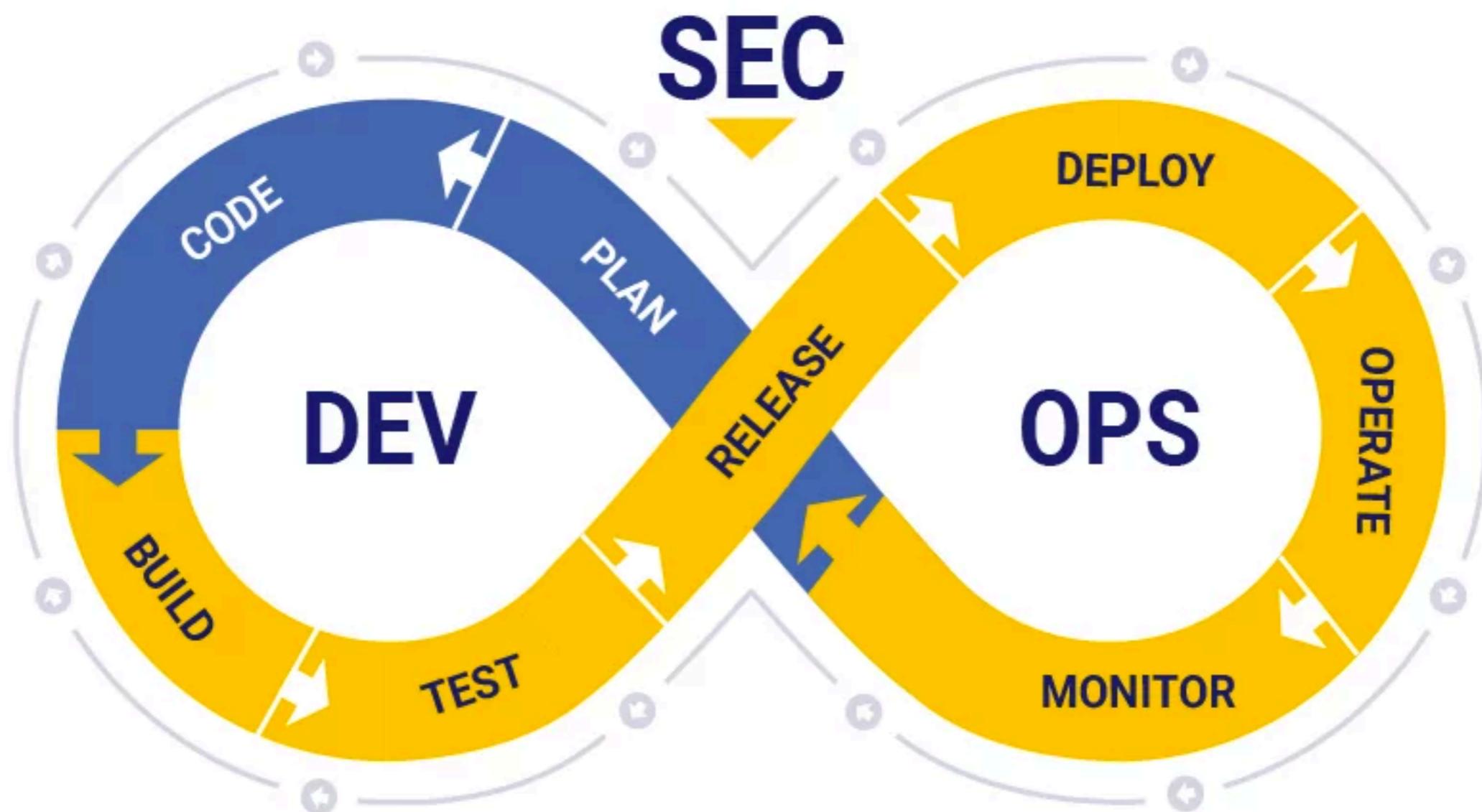
CC BY-NC-ND 4.0 International Licence  
[creativecommons.org/licenses/by-nc-nd/4.0/](https://creativecommons.org/licenses/by-nc-nd/4.0/)

[scgupta.me](http://scgupta.me)   
[twitter.com/scgupta](https://twitter.com/scgupta)   
[linkedin.com/in/scgupta](https://linkedin.com/in/scgupta)

# Modern Data Analytics Reference Architecture on AWS

This architecture enables customers to build modern data analytics pipeline using the [Lake House](#) approach to derive insights from the data.





**Developers**

**Code**

**QA**

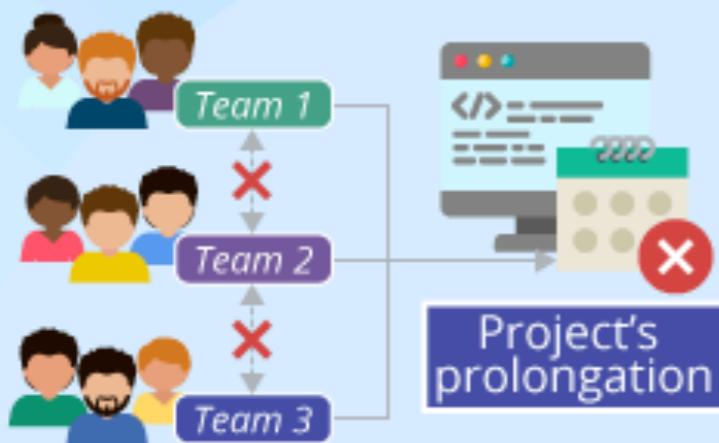
**Deployment/  
Operations**

# pre-DevOps

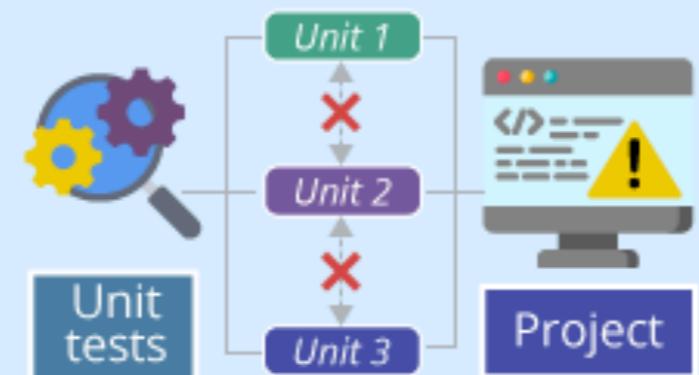
## 1. In-house, outsourced, or partially outsourced software development



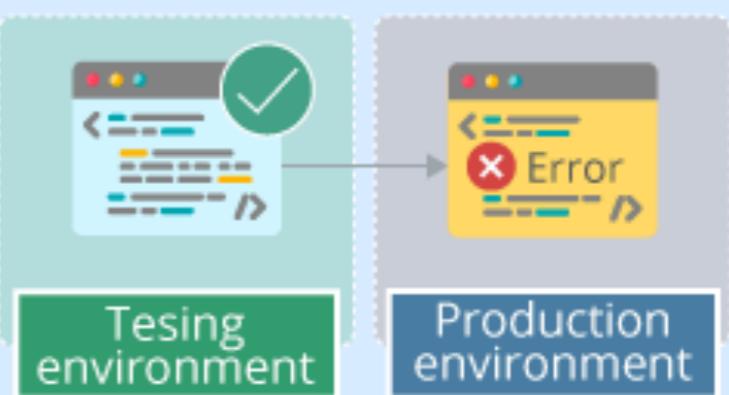
## 2. Strict segregation of duties between the departments



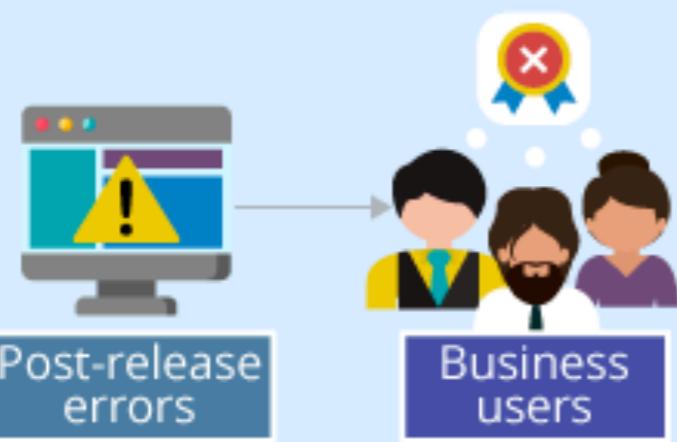
## 3. Insufficient coverage by tests



## 4. High probability of post-release errors



## 5. Lack of users' trust in software quality



## 6. Weeks for updates and fixes

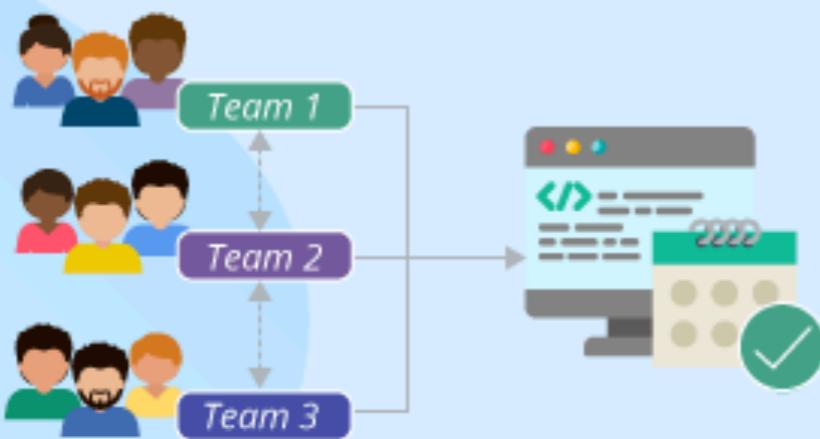


## 7. Time-consuming deployment of the infrastructure



# DevOps

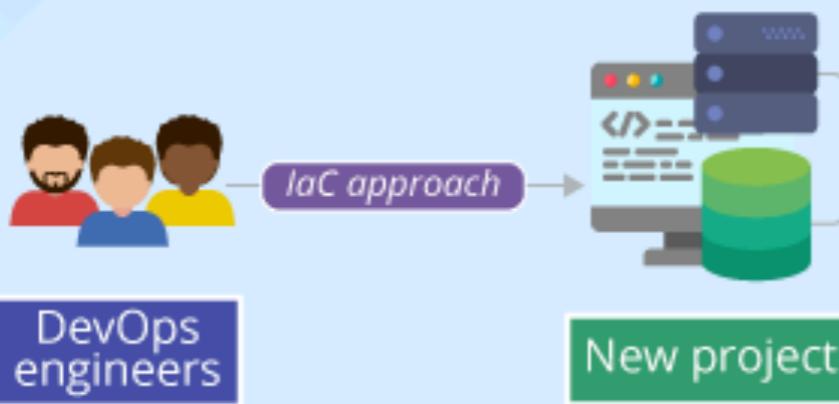
1. Constant communication between the teams engaged in software development



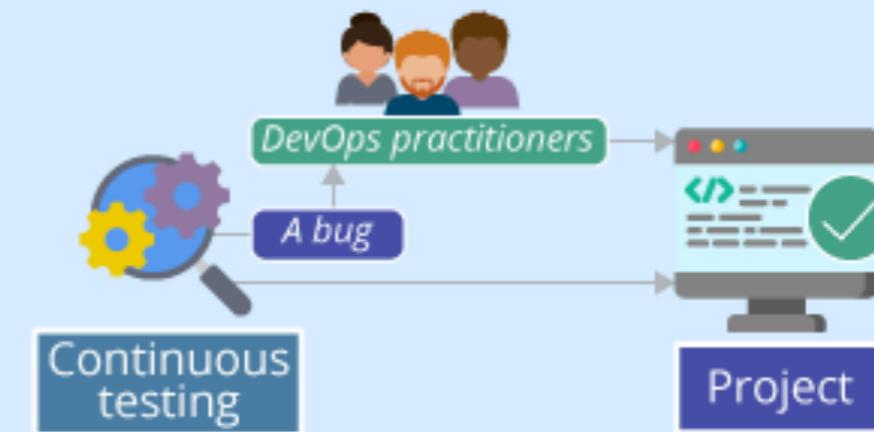
2. Fewer software failures caused by the differences in infrastructure configurations



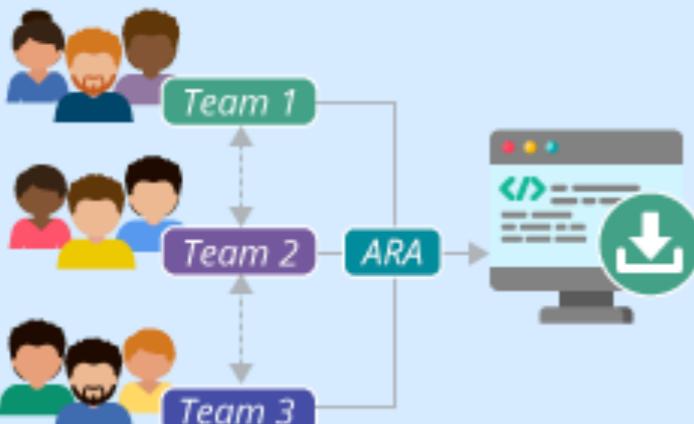
3. Fast provision of new infrastructure



4. The increased amount of test automation



5. Quick and reliable delivery of application updates



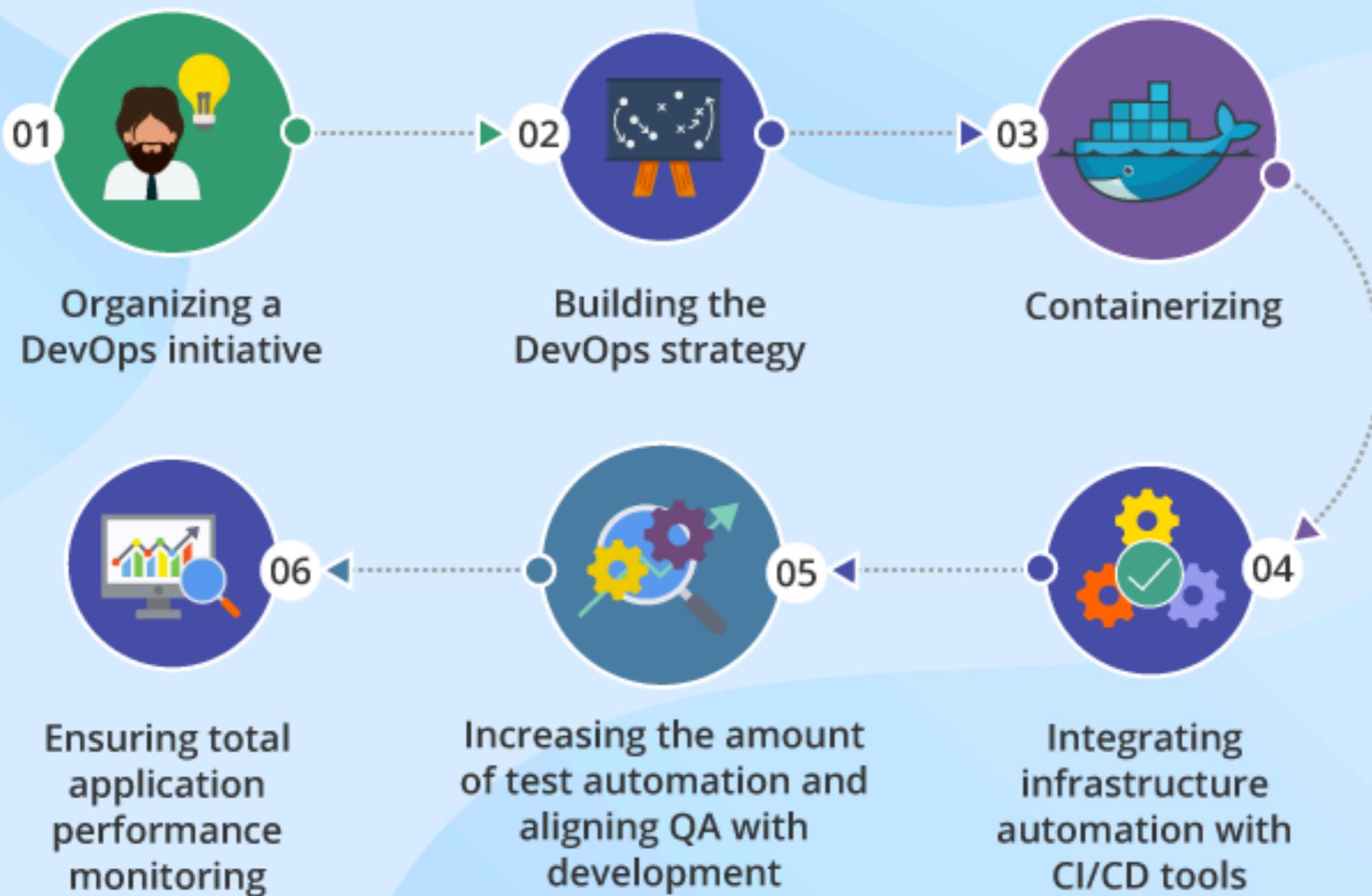
6. Fewer post-release errors

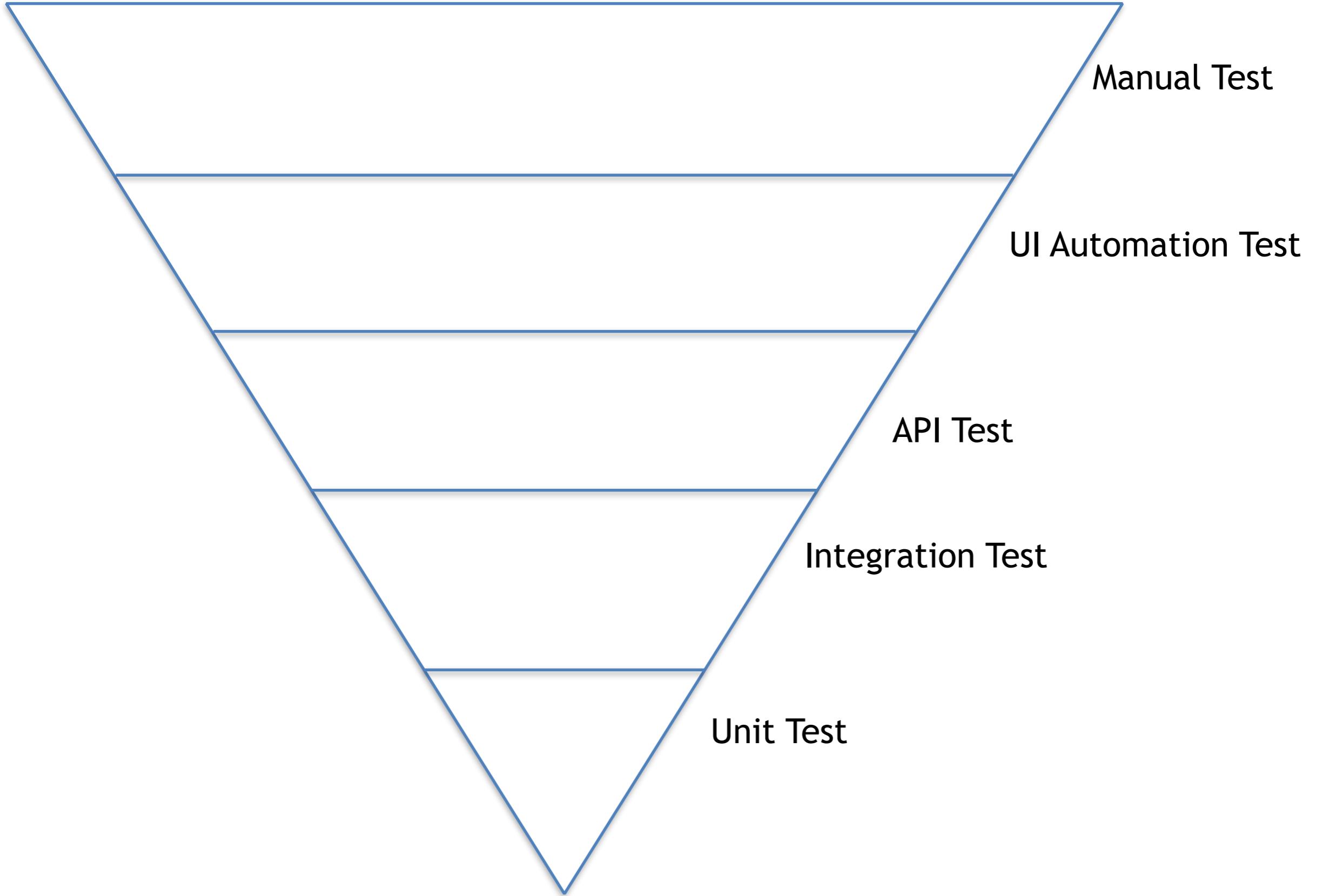


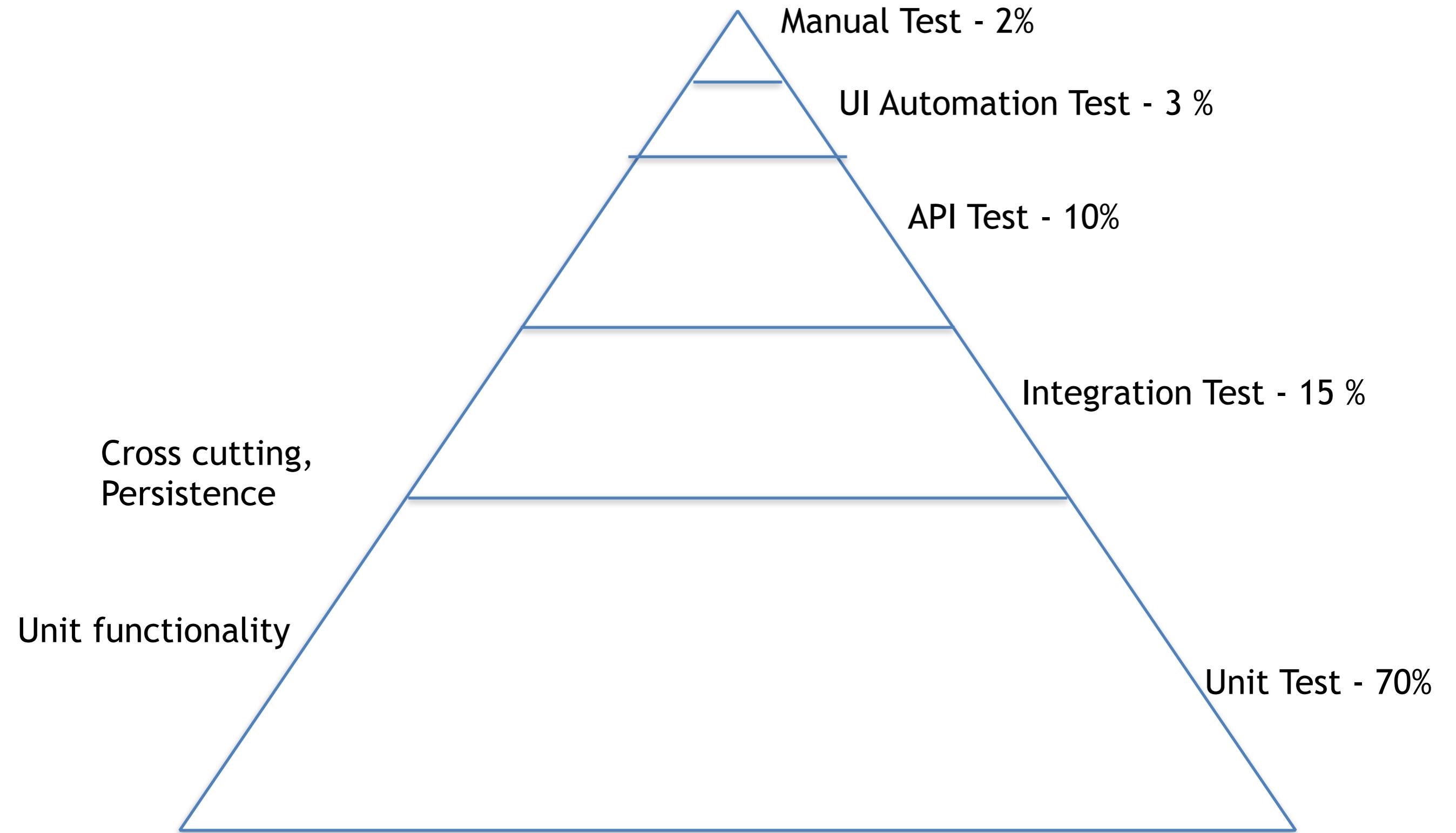
7. Improved users' trust



## DEVOPS IMPLEMENTATION ROADMAP







## Collaborate

### Application Lifecycle Mgmt.



### Communication & ChatOps



### Knowledge Sharing



## Build

### SCM/VCS



### Testing



## Test

### Deployment



## Deploy

### Cloud / IaaS / PaaS



## Run

### Config Mgmt./Provisioning



### Orchestration & Scheduling

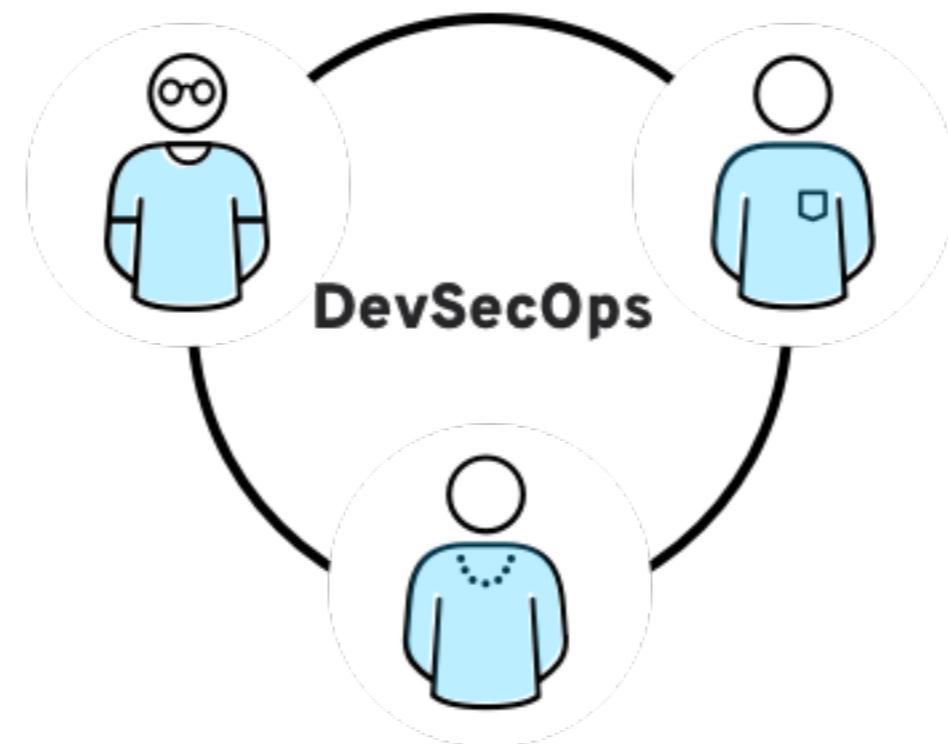
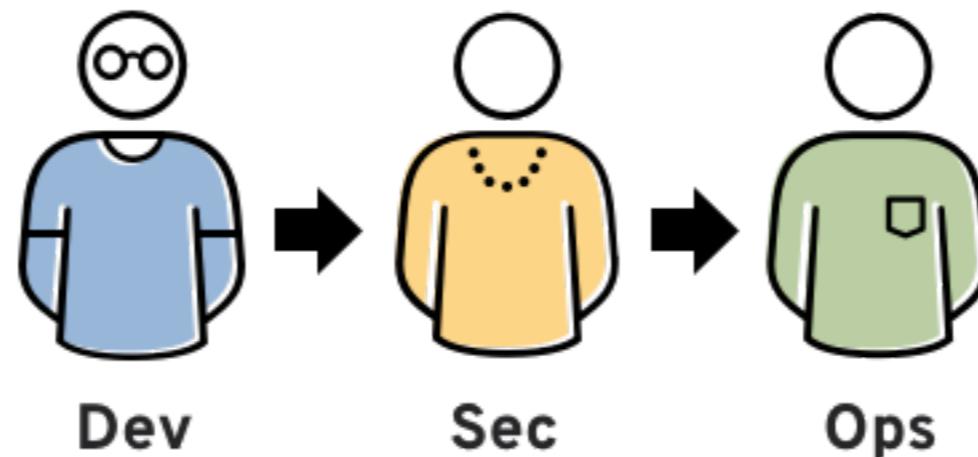


### Artefact Management



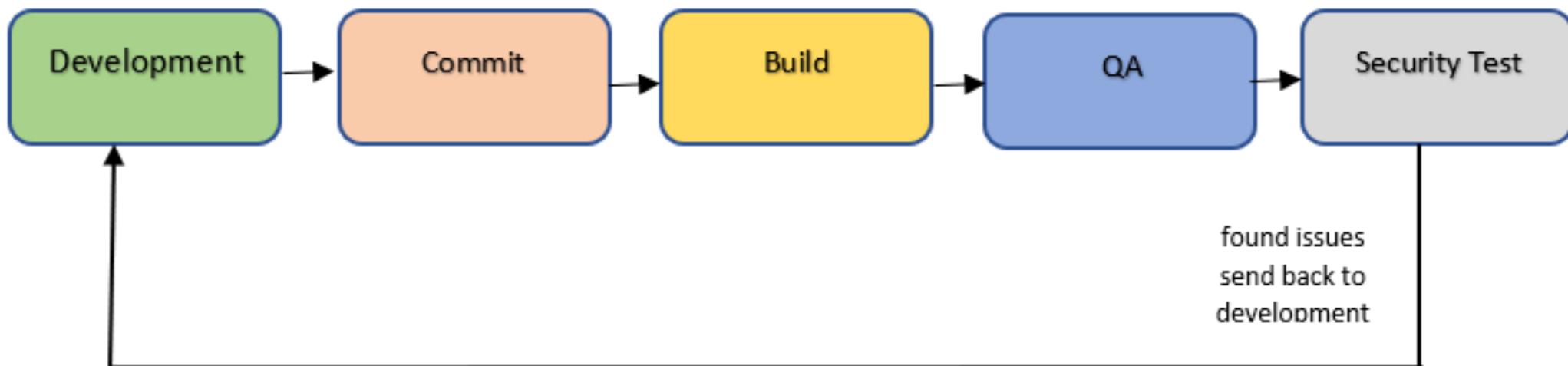
### BI / Monitoring / Logging



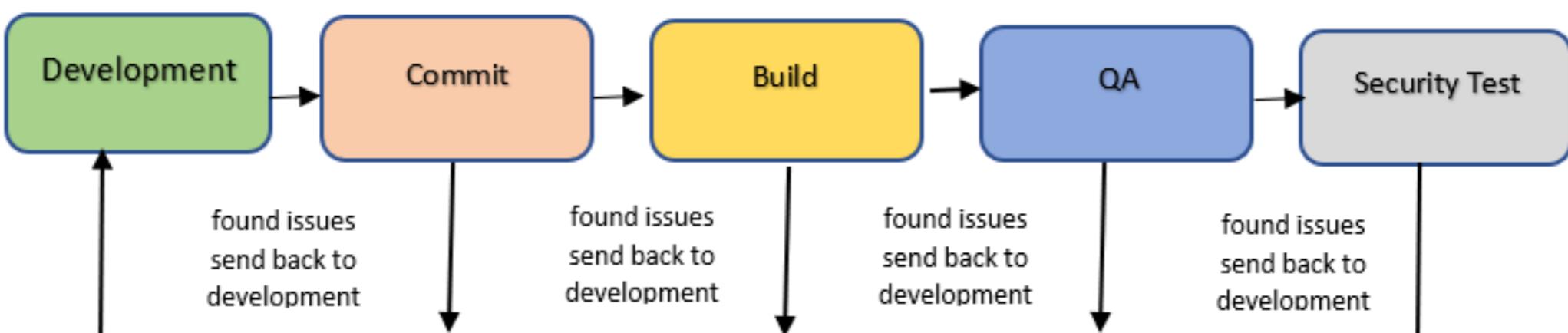


DevSecOps is about injecting security into the DevOps lifecycle.

*"Security in traditional way"*



*"Security in DevSecOps way"*



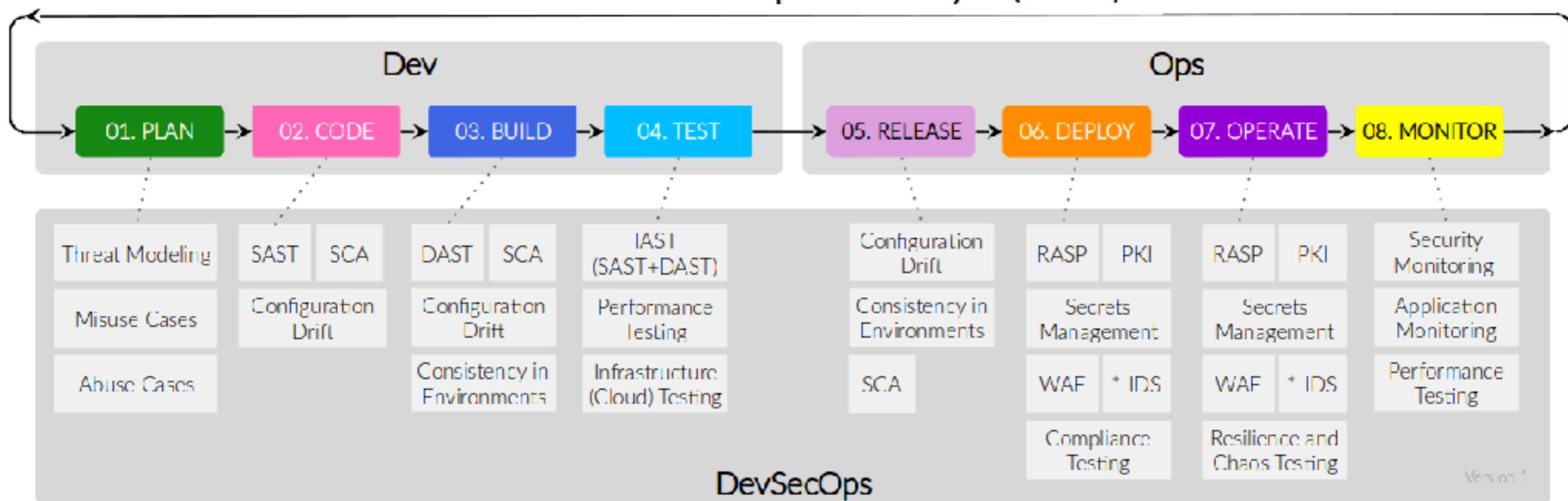
## IAST (SAST + DAST)

## RASP (SAST + DAST)

### DAST

### SAST

### Secure Software Development Life Cycle (SSDLC)



# 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)

### DAST

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

- + Runtime Privilege Issues
- + Authentication Issues
- + Session Management Issues
- + Insecure 3<sup>rd</sup> 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.



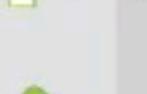
## Typically supports all kinds of software

- 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.

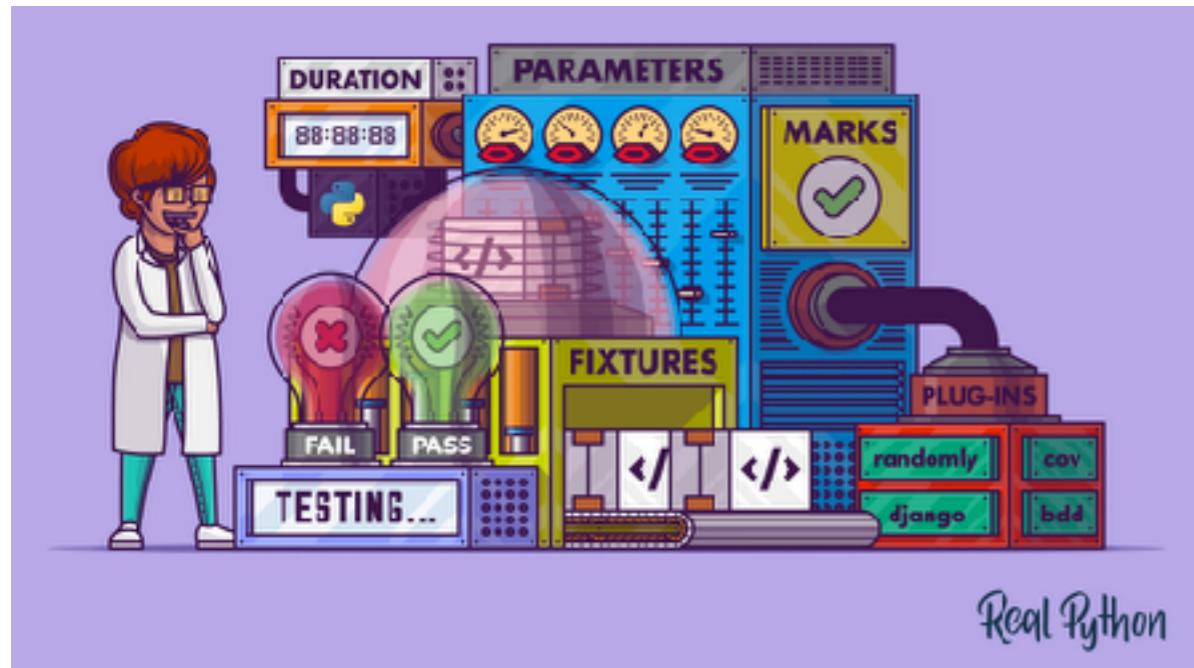


SAST and DAST techniques complement each other.



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

