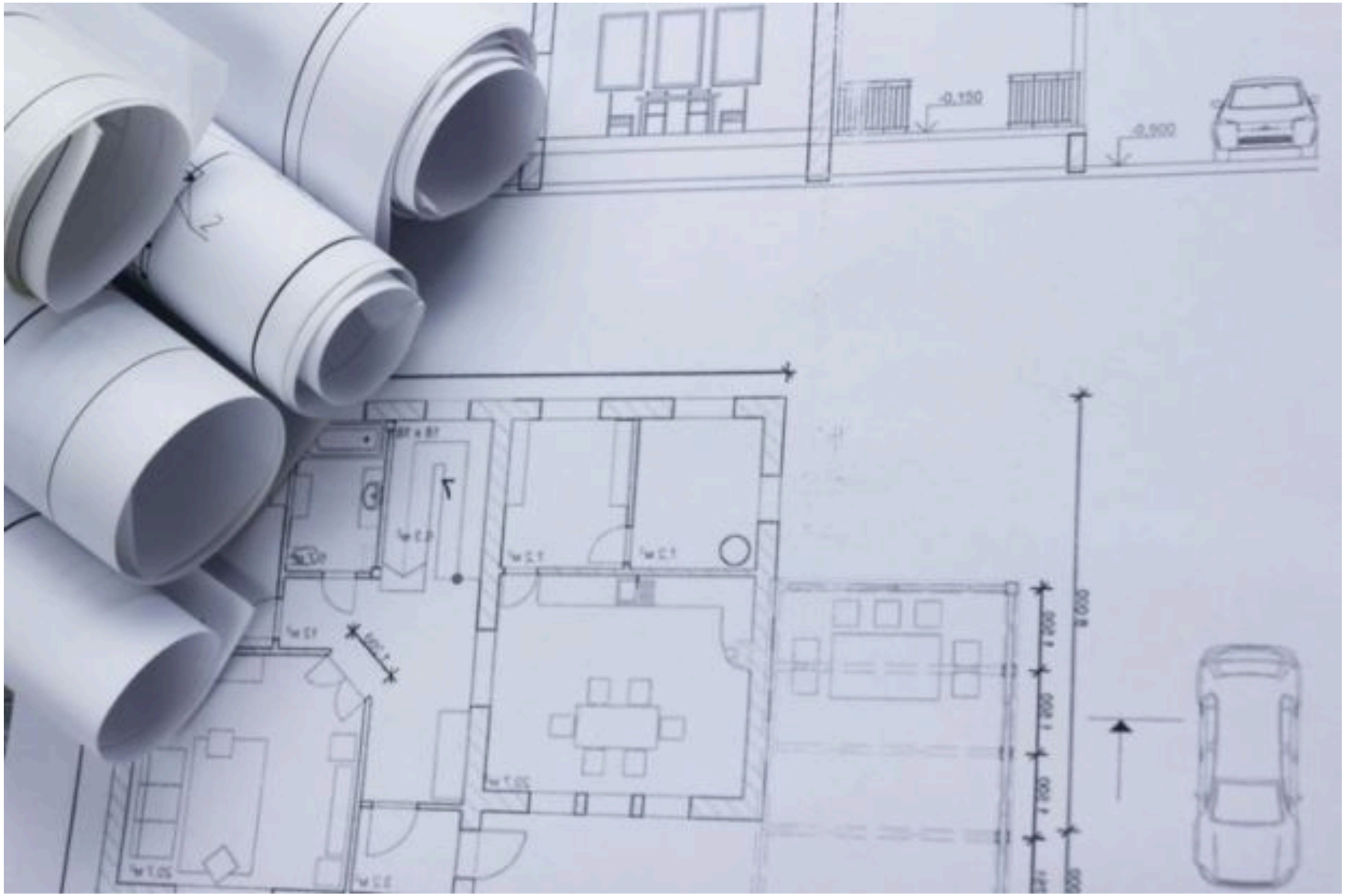


Software Architecture

» Skan.ai - chief Architect

» Ai.robotics - chief Architect

» Genpact - solution Architect

» Welldoc - chief Architect

» Microsoft

» Mercedes

» Siemens

» Honeywell

Mubarak

- Arch Requirements
- Arch Design
- Arch Doc
- Arch Eval

Architecture vs Design

Attribute	Measure	Approach
• Performance <— • Maintainability <— • Security (Trust) <— • Scalability (Volume - I/O, CPU, Memory) <— • Usability x • Availability <— • Reliability (Trust) <— • Robustness (Rugud) •	• Response time • Memory • CPU • I/O • Latency • TPS • …	• Caching • Parallel • Pooling • Lazy Loading • Compression • Chunking • Reusability/ Extensible • Modular /Component • Low Coupling • EDA • ACID - transaction • Input Validation

5 Communicate

Key functional requirements

2

Understands Domain

NFR

3

Quality attributes, Measure (Architectural Requirements)

Collect

4

Choose

Approaches (Architectural Decisions)

1

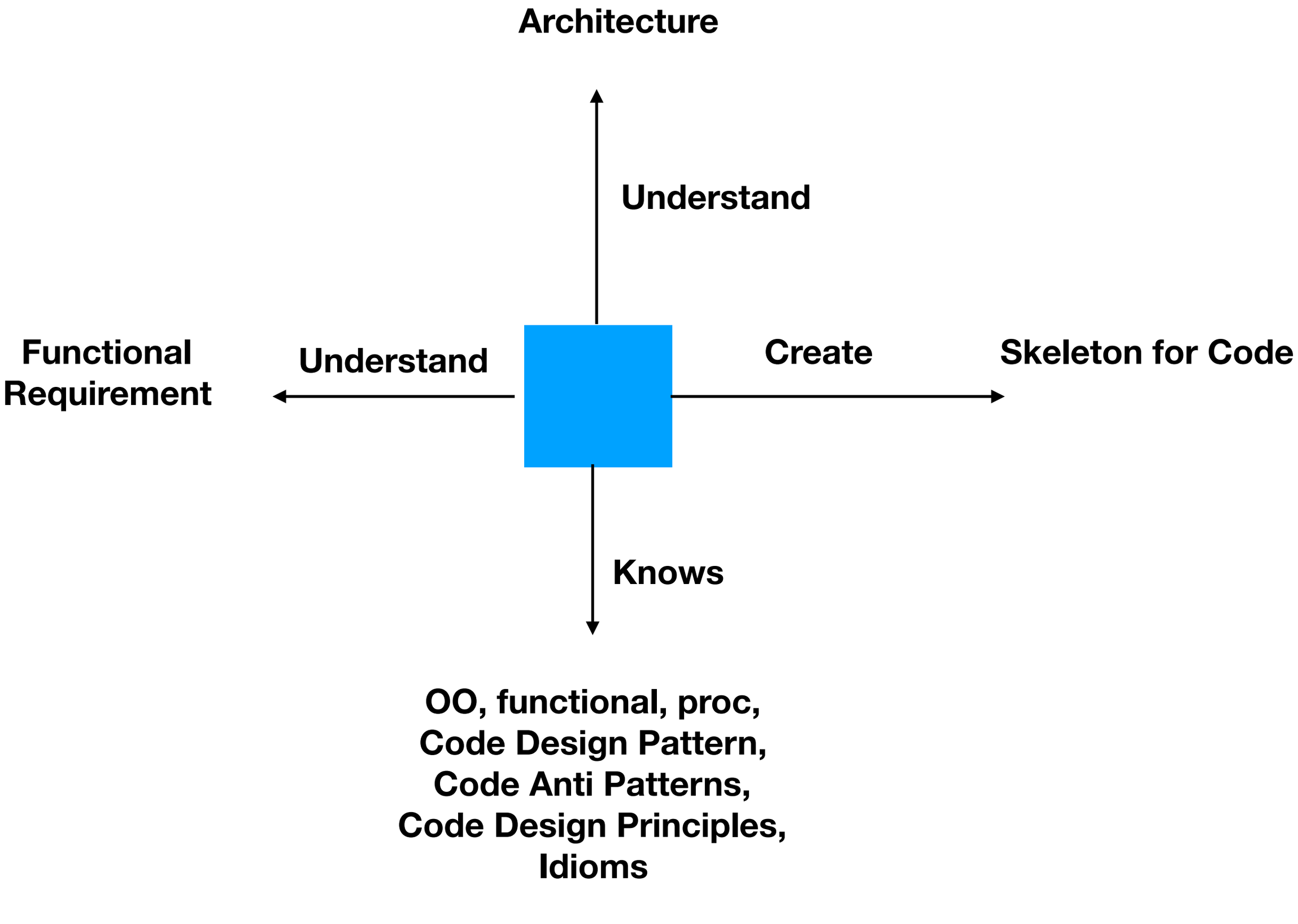
Knows

Arch Tactics, pattern, styles
Arch Principles
Arch anti patterns
Reference Arch (*)

Pre

Engineering vs Tuning

Post


Architecture
Understand
Functional Requirement
Understand
Create
Skeleton for Code
Knows
OO, functional, proc,
Code Design Pattern,
Code Anti Patterns,
Code Design Principles,
Idioms

Architecture Design
UI Design
Test Design
Module Design
Class Design
Code Design

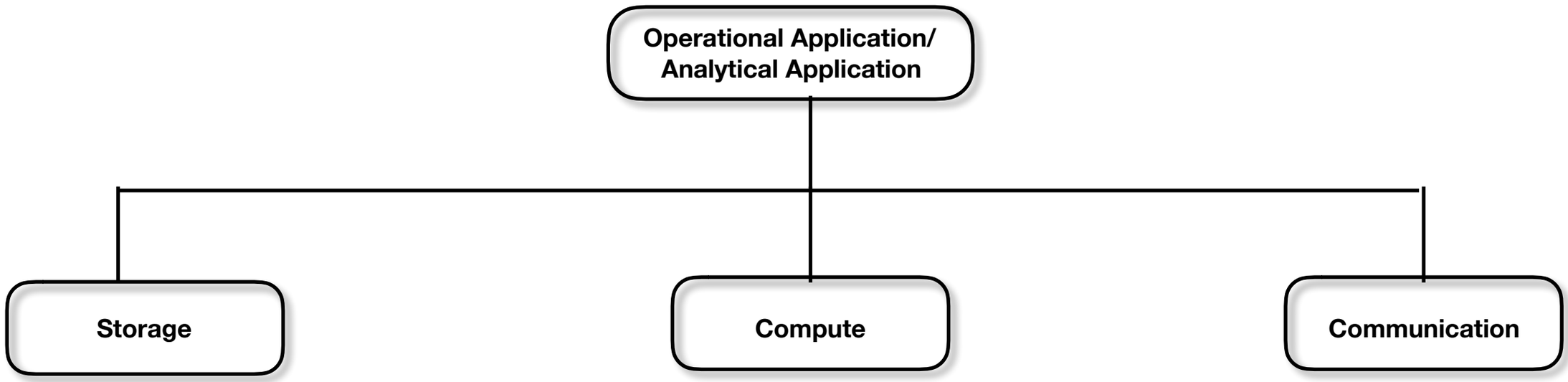
Architecture vs Design

Code Maintainability

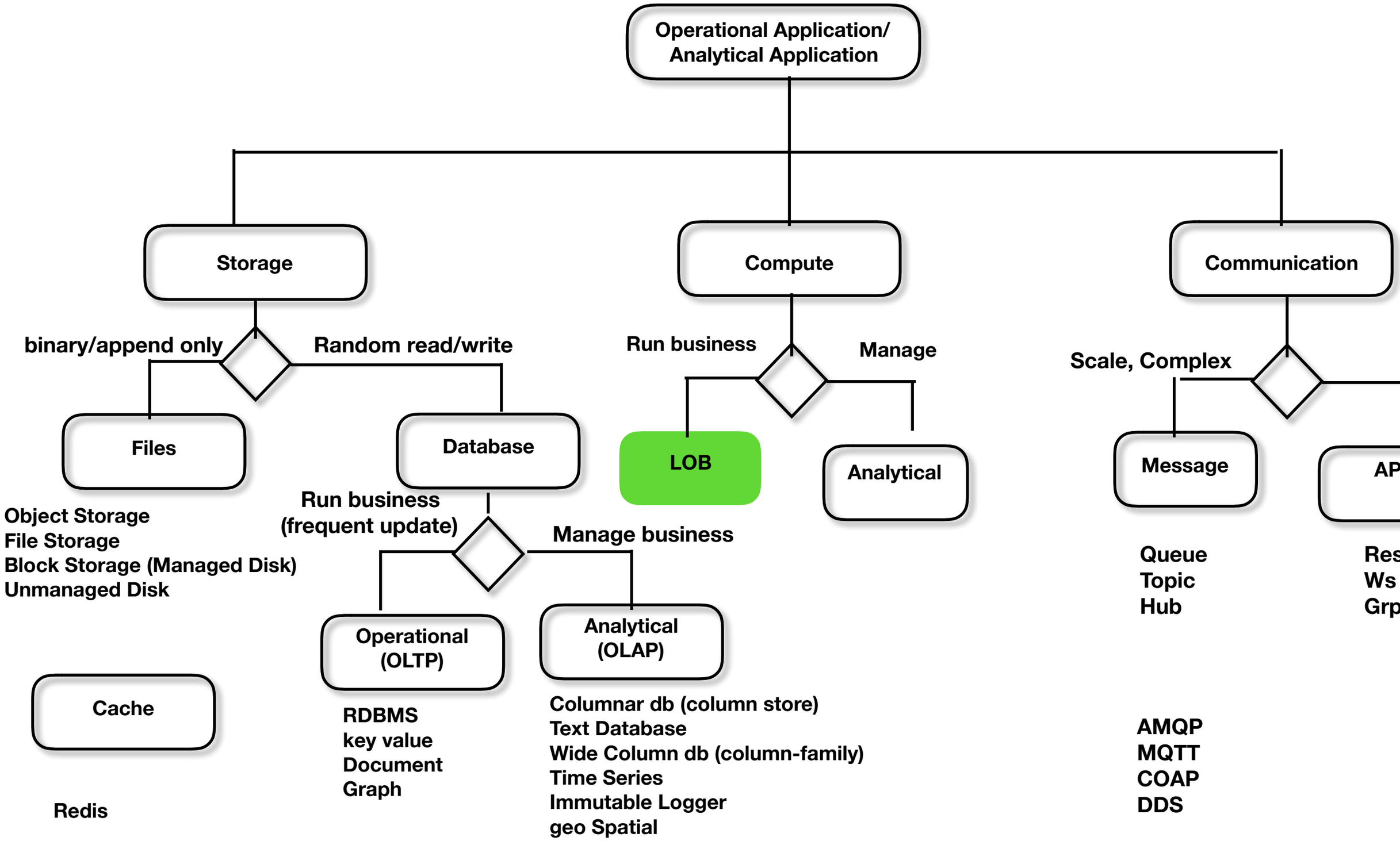
System quality

Detail Design
Module Design
Class Design
Low Level Design
Code design
Implementation Design

Architect an Cloud App ?


Operational Application/
Analytical Application
Storage
Compute
Communication

- Binary (images, documents, videos,…)
- Append only
- Archive
- Scale
- Static content
- No random read



System

Compute

Compute

Compute

Binary Storage

Object Storage

Immutable
$
Low Perf
High Volume
0.01$/GB

- Azure Blobs
- AWS S3
- google Cloud Storage

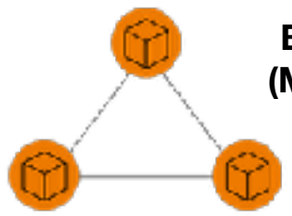
4

File Storage

Mutable
$$$$
Medium Perf (ex: 500 ms)
Medium Voulme

- Azure Files
- AWS EFS
- Google Filestore
- NFS

3



Block Storage
(Managed Disk)

Mutable
$$$
High Perf (10 ms)
Low Volume
1$/GB

- Azure Disks
- AWS EBS
- Google Persistent Disk

2

Unmanaged Disk

Mutable
$
High Perf

- Local Disk

1

Access across VM
(Network Storage)

Single VM Storage

Rest API /
SOAP protocol

Network File System (NFS) Protocol /
Server Message Block (SMB) protocol

Internet

Lan

Persistent

Temporary

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

Operational (OLTP)

Structured

**[ACID - transaction,
LIMITED SCALABLE - not distributed
Fixed Schema]**

RDBMS (OLTP)

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

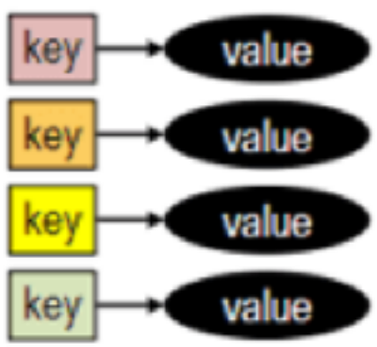
1

Semi Structured

**[* BASE,
SCALE- shareded (distributed),
* Flexible Schema]**

Dictionary (key value)

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



2

Nested Objects

Document

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

3

Entity Relationship

Graph

- AWS Neptune
- Azure Cosmos -> Graph
- 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

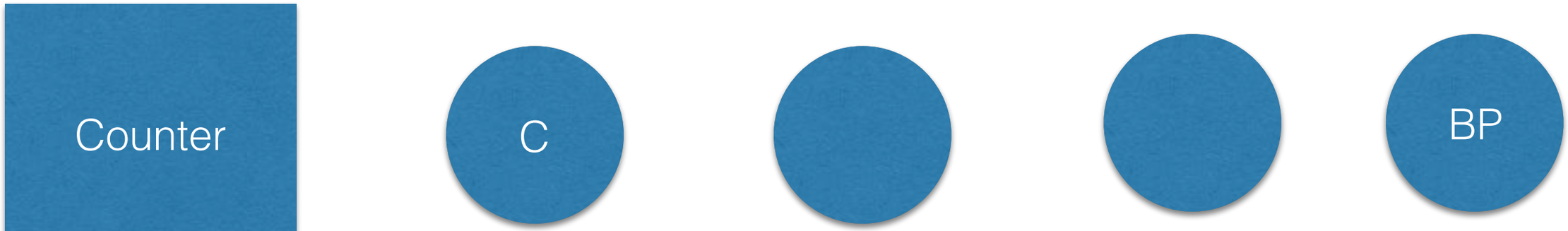
UserID	Name	Dob
1	John Smith	8/30/1964

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" }
   ]
}
```


Counter
C
BP

Counter

C

BP

Counter

E

BP

Counter

?

BP

General

C

E

?

BP

BP

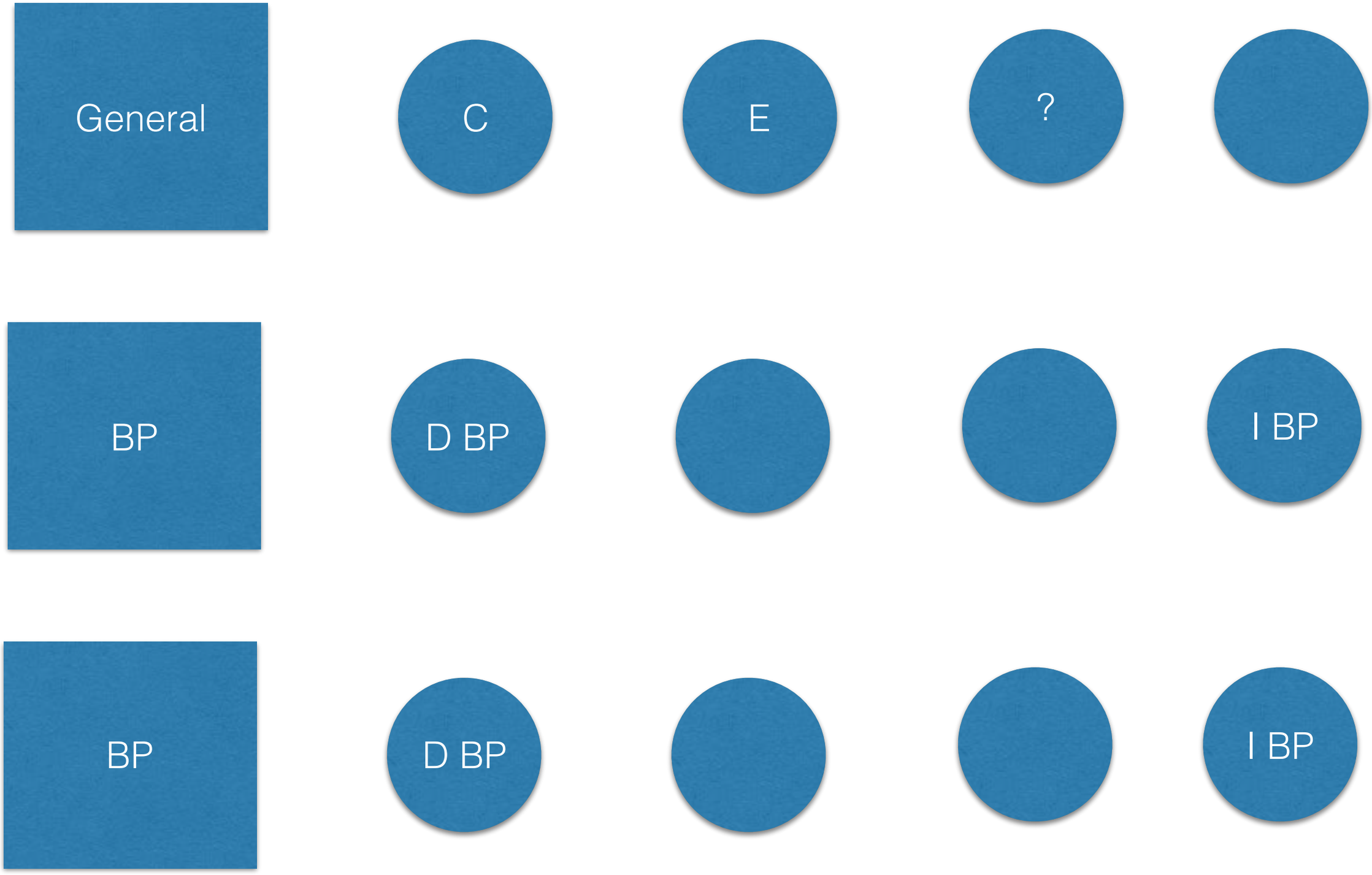
BP

BP

BP

BP

BP


General
C
E
?
BP
D BP
I BP
BP
D BP
I BP

General

C

E

?

D BP

D BP

D BP

D BP

D BP

D BP

D BP

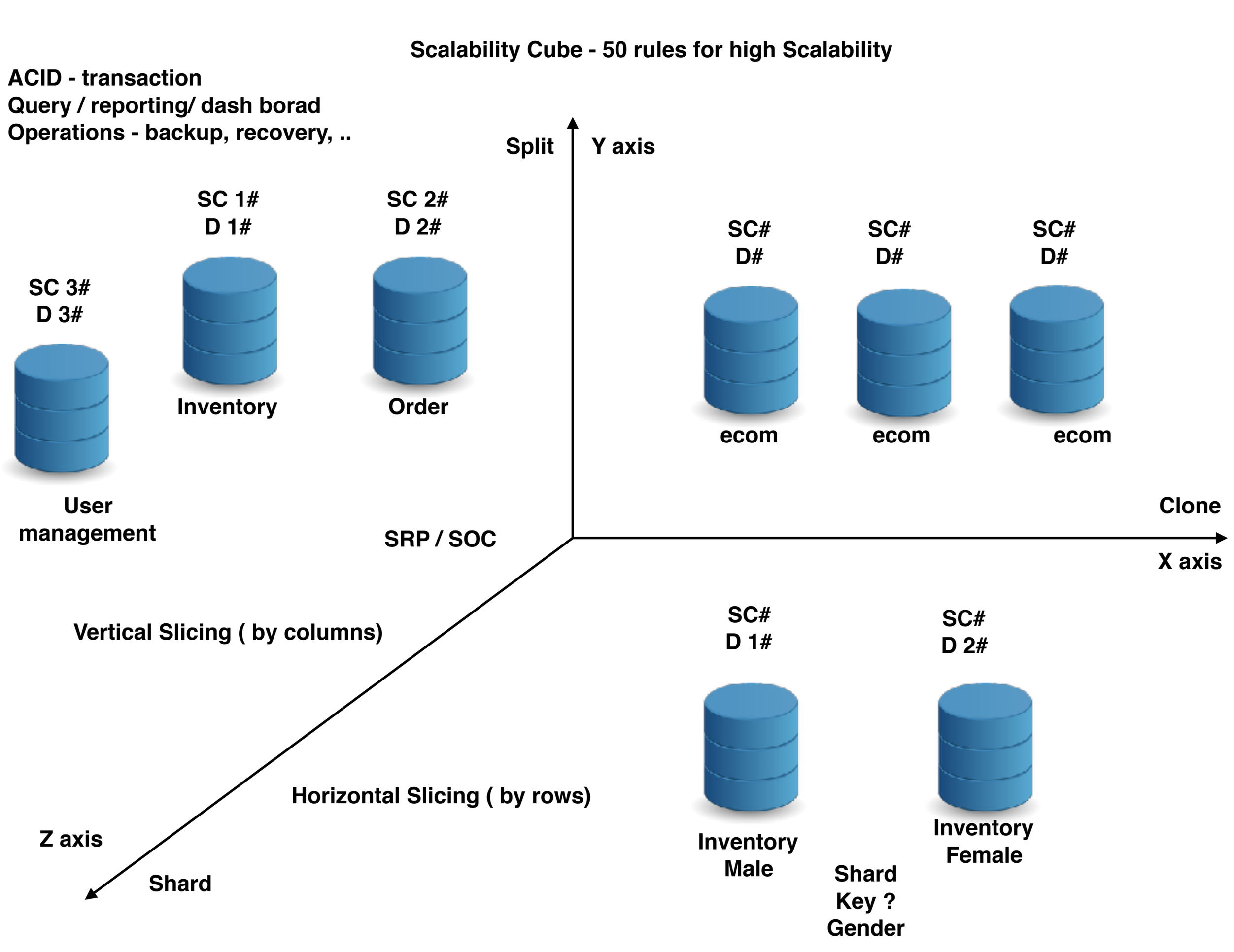
I BP

I BP

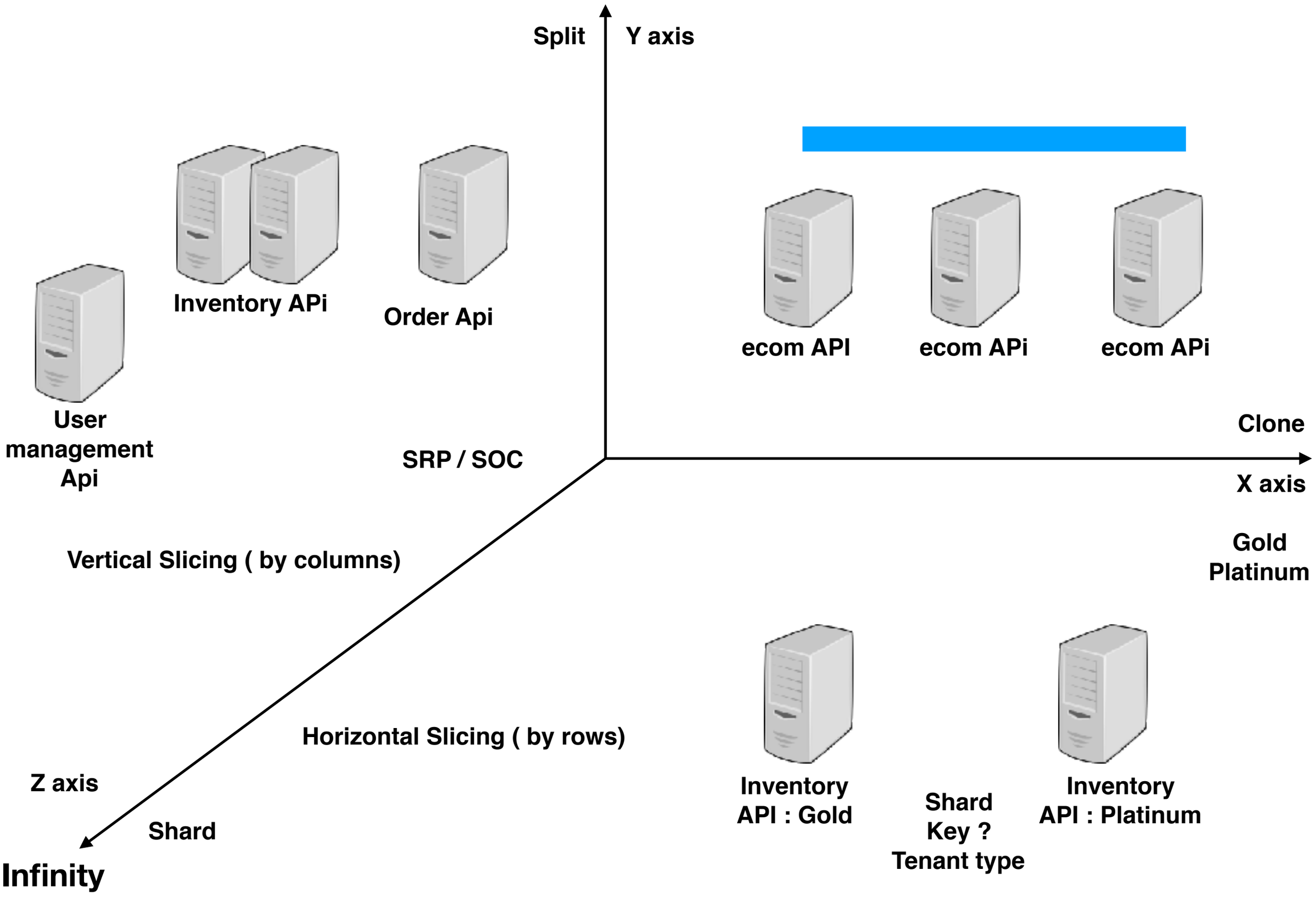
I BP

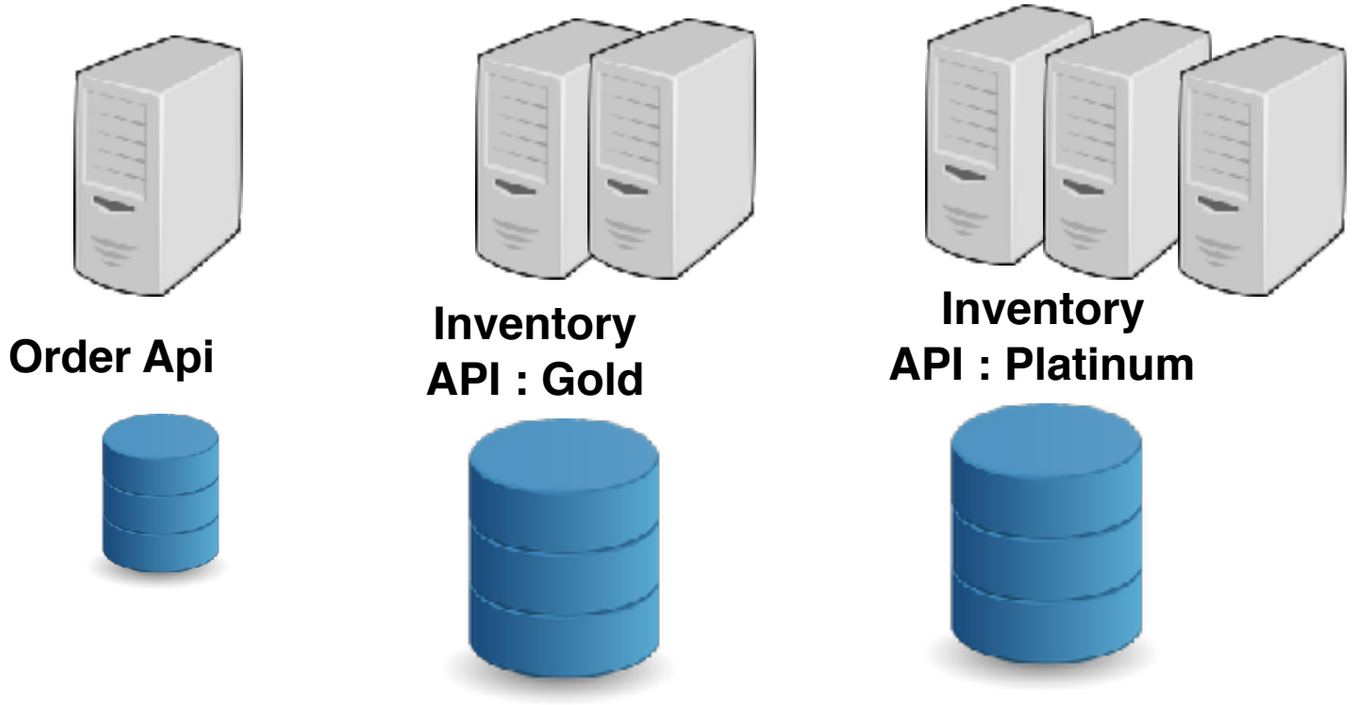
Scalability Cube - 50 rules for high Scalability

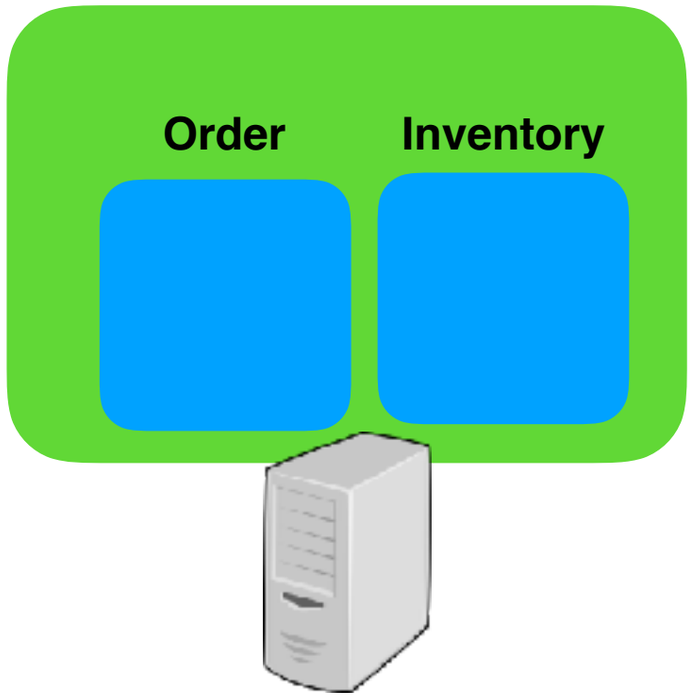
ACID - transaction
Query / reporting/ dash borad
Operations - backup, recovery, ..



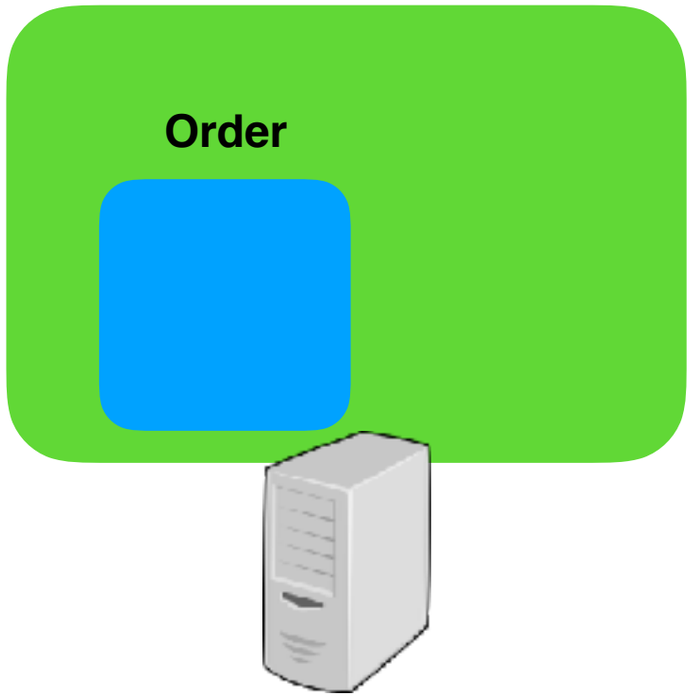
Scalability Cube - 50 rules for high Scalability



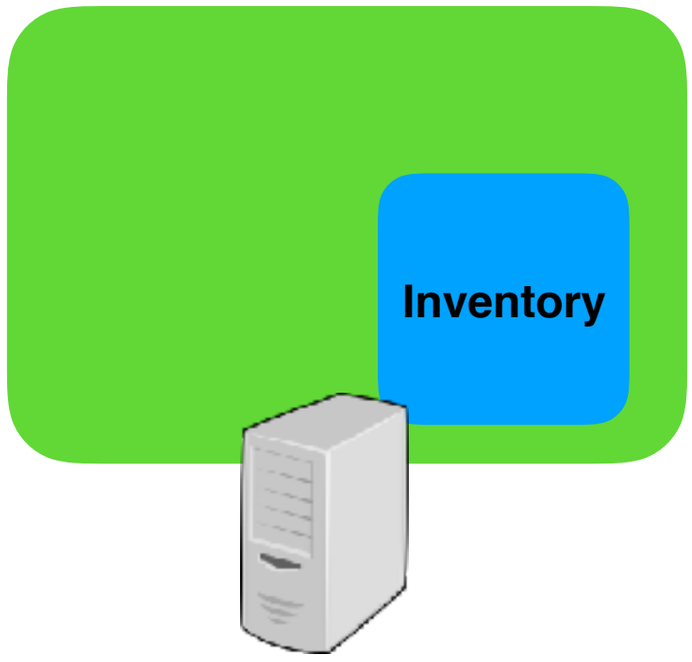

Order Api
Inventory
API : Gold
Inventory
API : Platinum



ecom

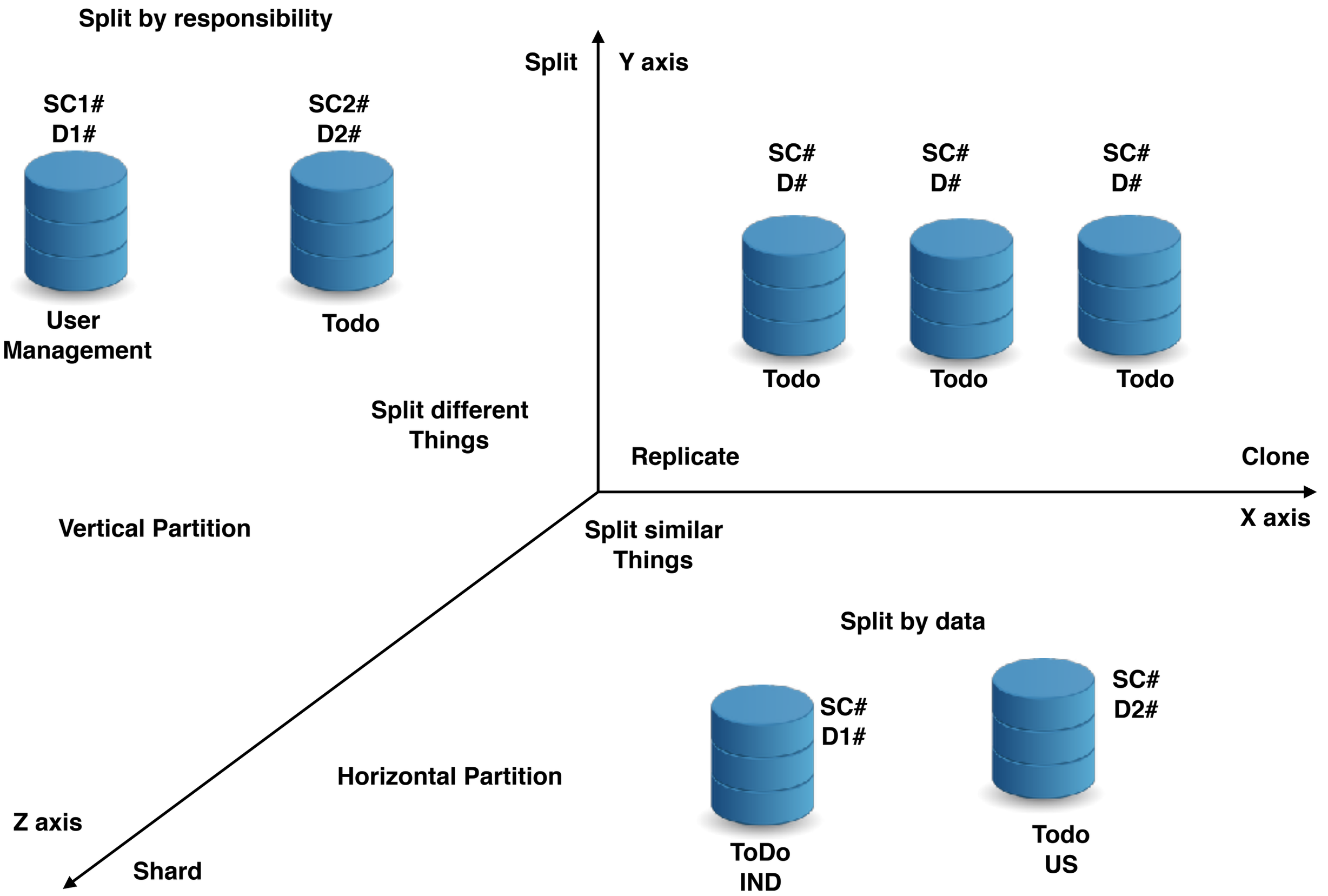


Order



Inventory

Scalability Cube - 50 rules for high Scalability



Ecom

Split different Things

Vertical Partition

User

Inventory

Accounts

Order

Horizontal Partition

Shard

Inventory: M

Inventory : F

Order: AUS

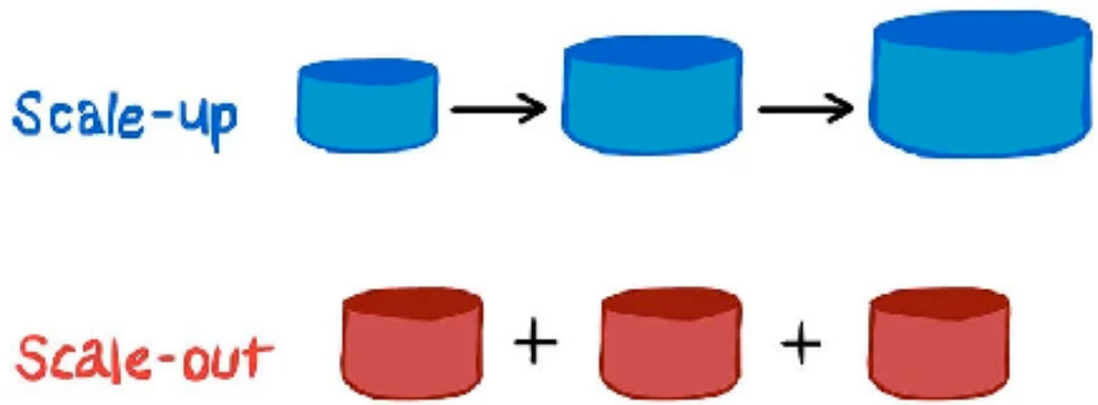
Order: USD

Clone

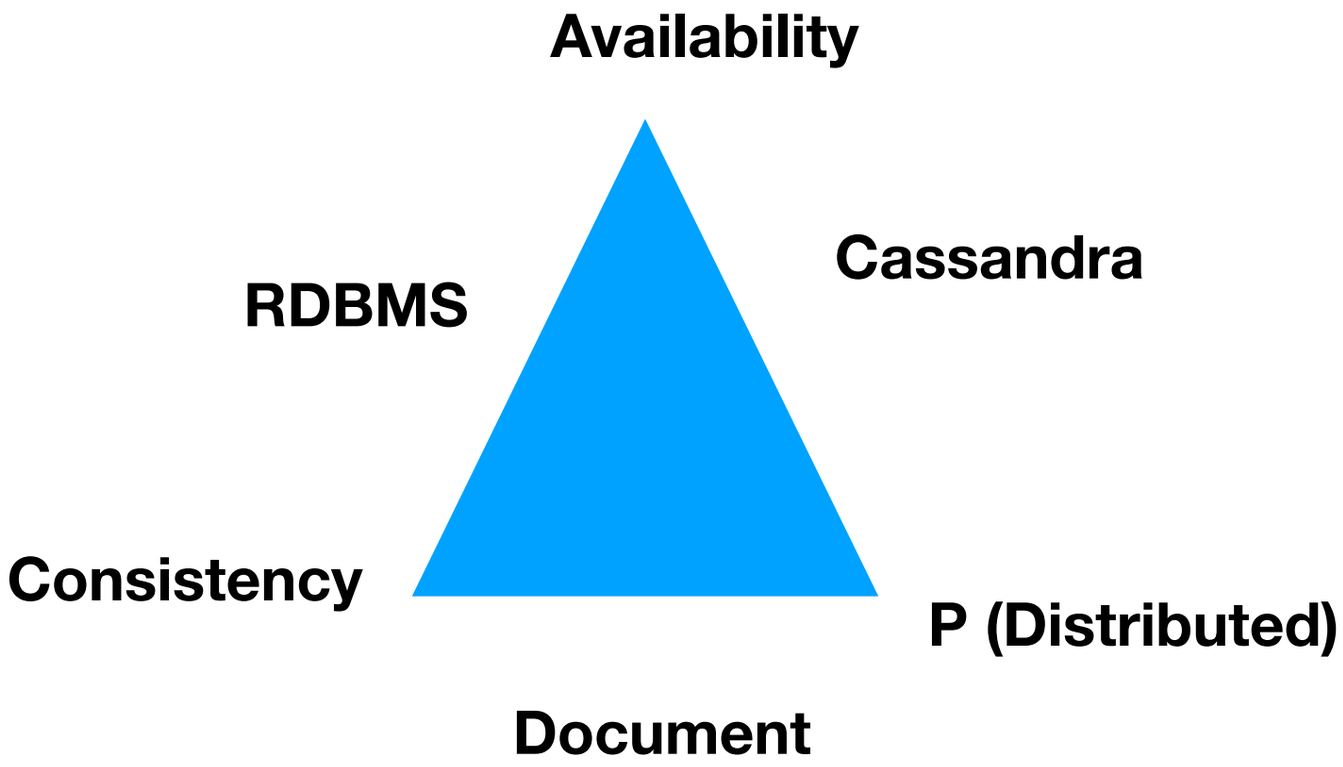
Inventory: M

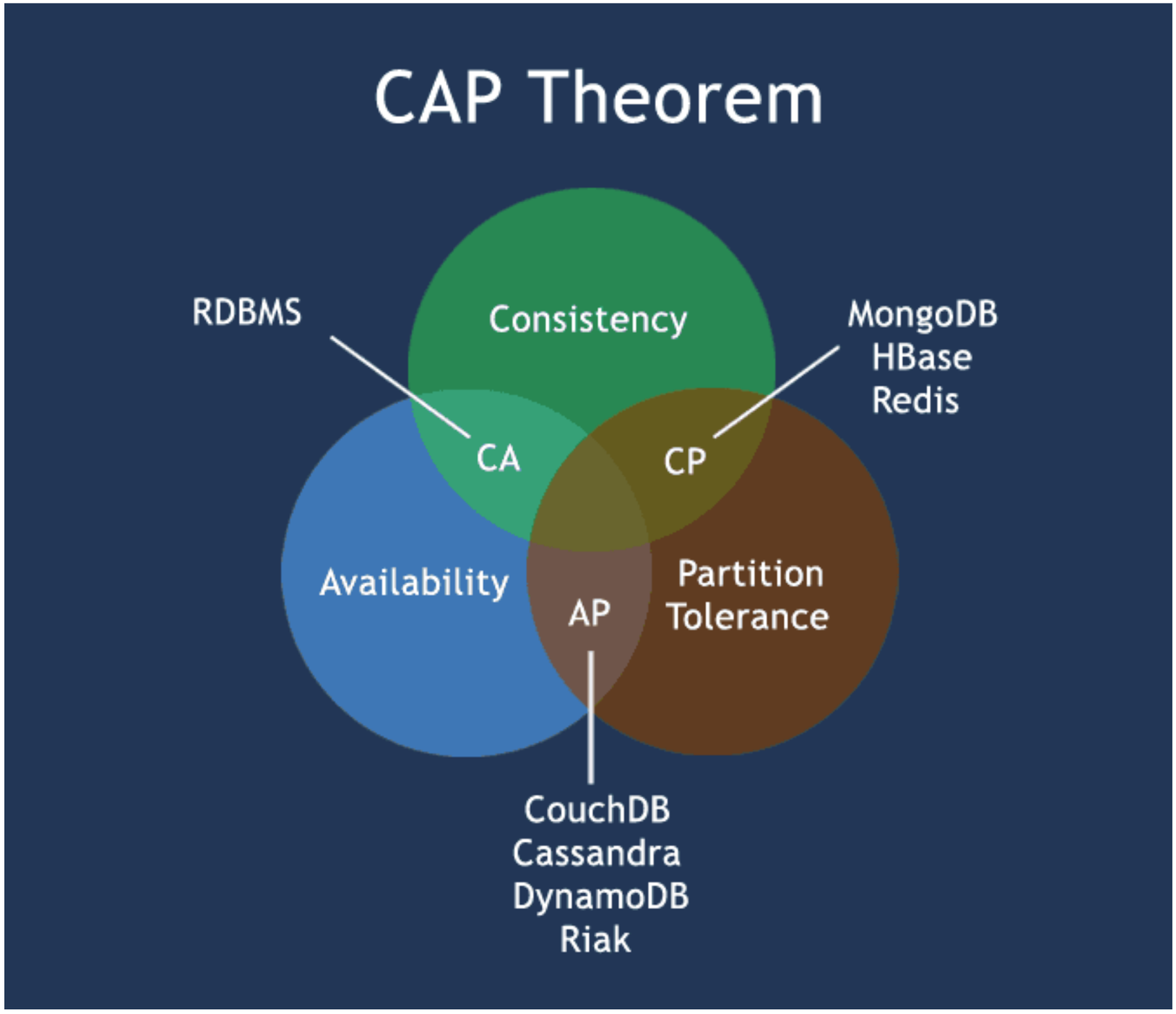
Inventory: M

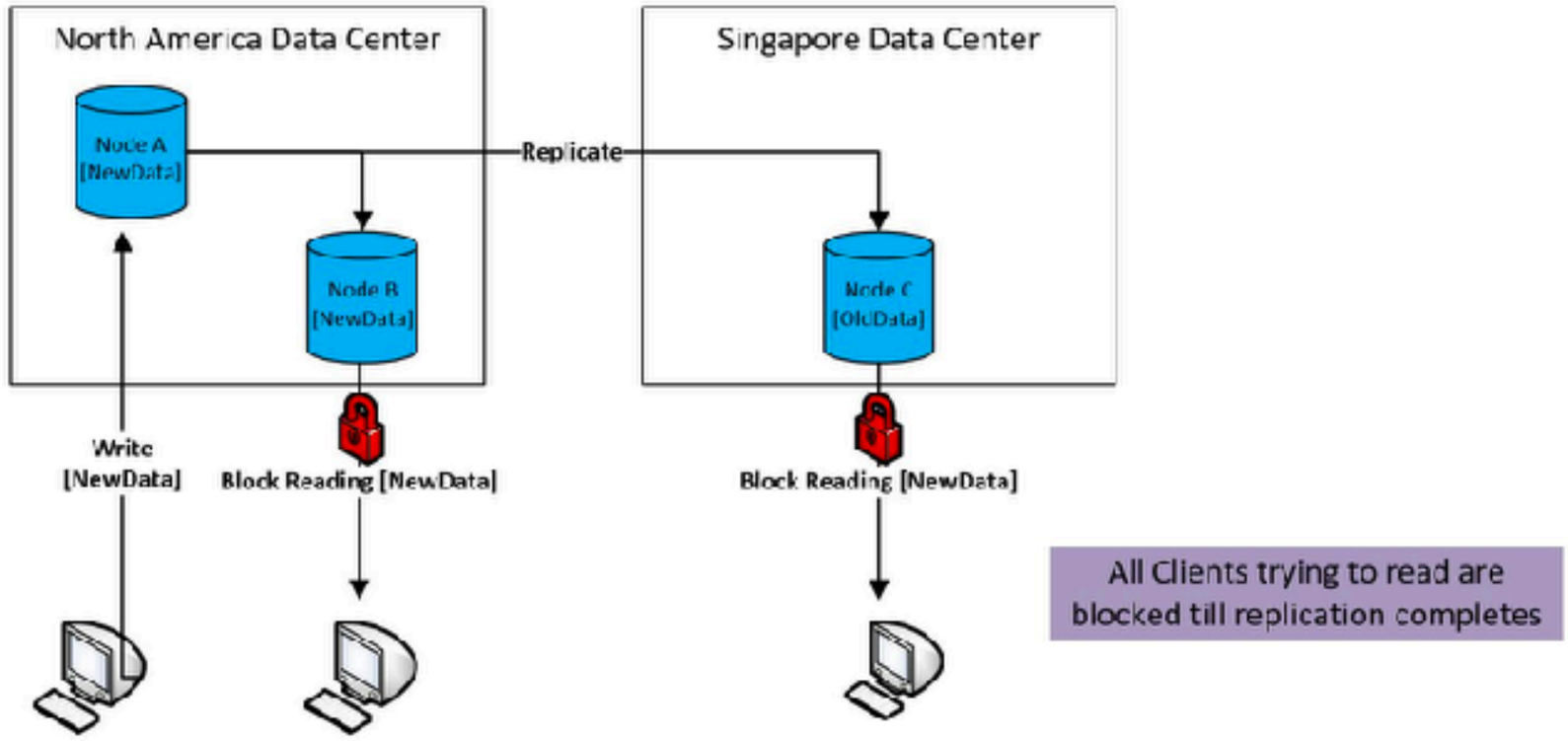
Scale-Up vs. Scale-Out

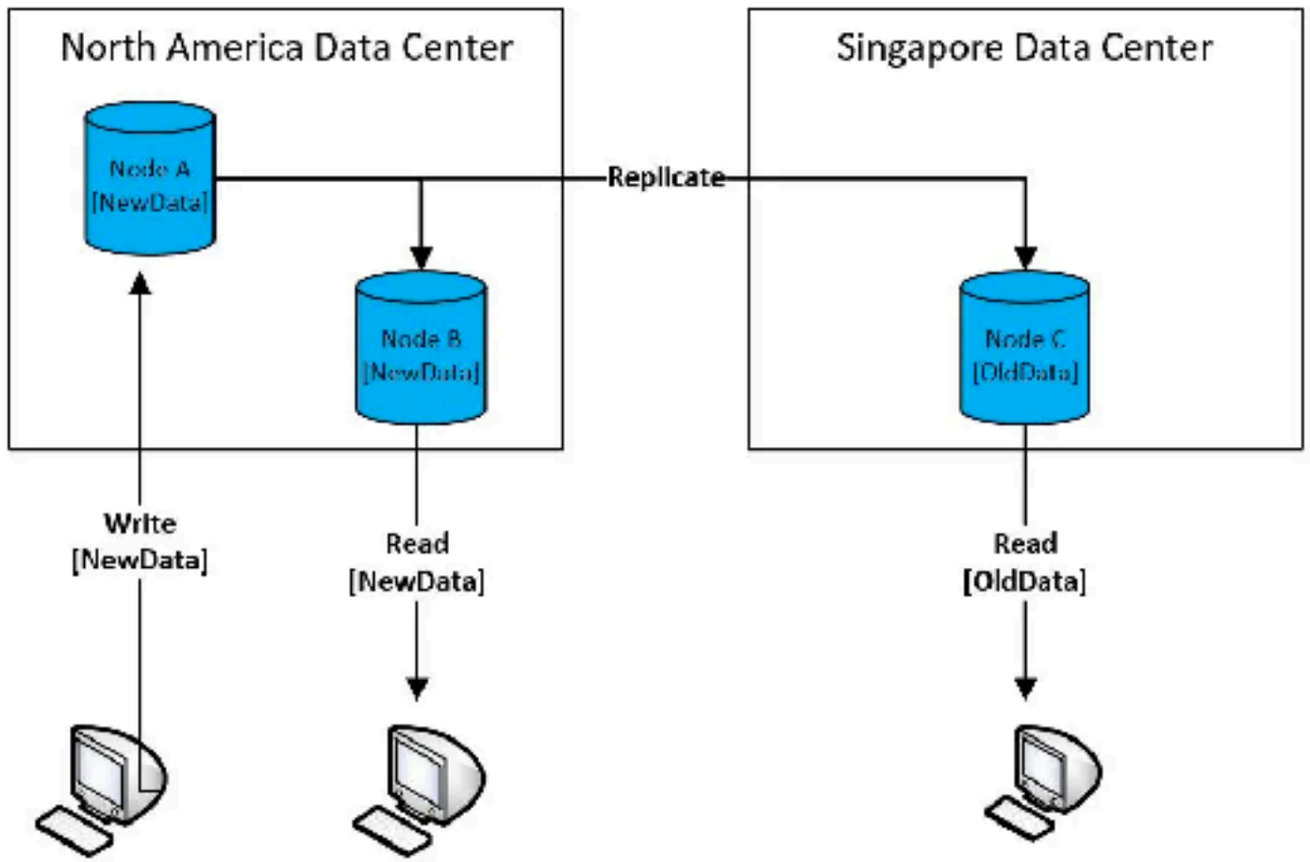


Strong Consistency vs. Eventual Consistency




CAP Theorem
RDBMS
Consistency
MongoDB
HBase
Redis
CA
CP
Availability
AP
Partition
Tolerance
CouchDB
Cassandra
DynamoDB
Riak


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

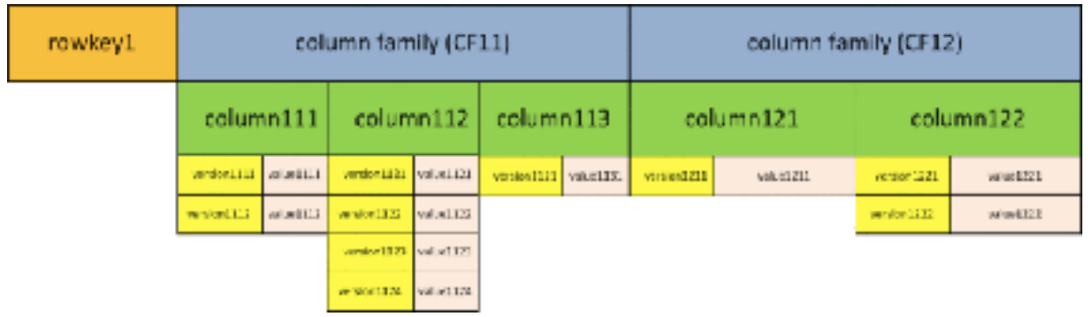
MYSQL & NOSQL

	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



Read

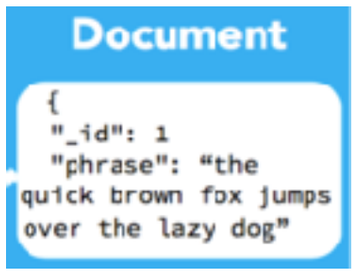
Columnar db (column store)

Cloud DWH (OLAP)

- AWS Redshit (max: 8 PB)
- **Azure Synapse**
- GCP BigQuery
- Snow flakes
- Kudu, Druid, Pinot
- Click house

Read

Text Database



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

Read/write

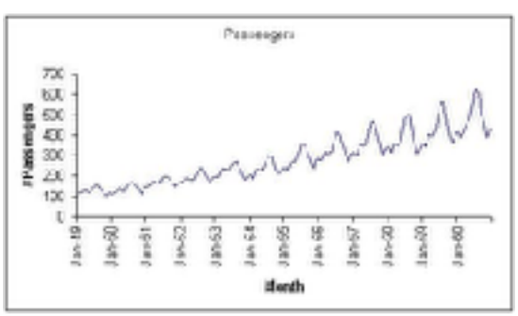
Wide Column db (column-family)

- AWS Keyspaces (unlimited)
- Cosmos
- Big Table
- **Hbase, cassandra**
- Scylla

KairosDB (cassandra)

Time Series

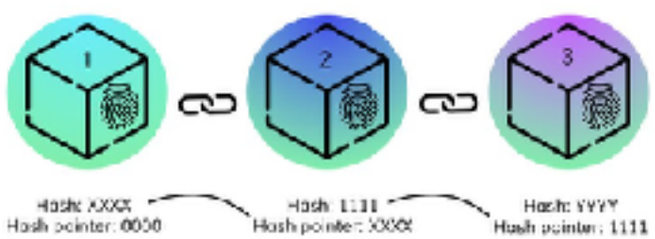
- AWS Timestream
- InfluxDB, Scylla
- OpenTSDB,
- KairosDB



Audit Trial

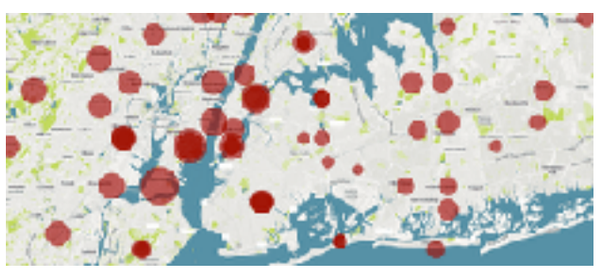
Immutable Logger

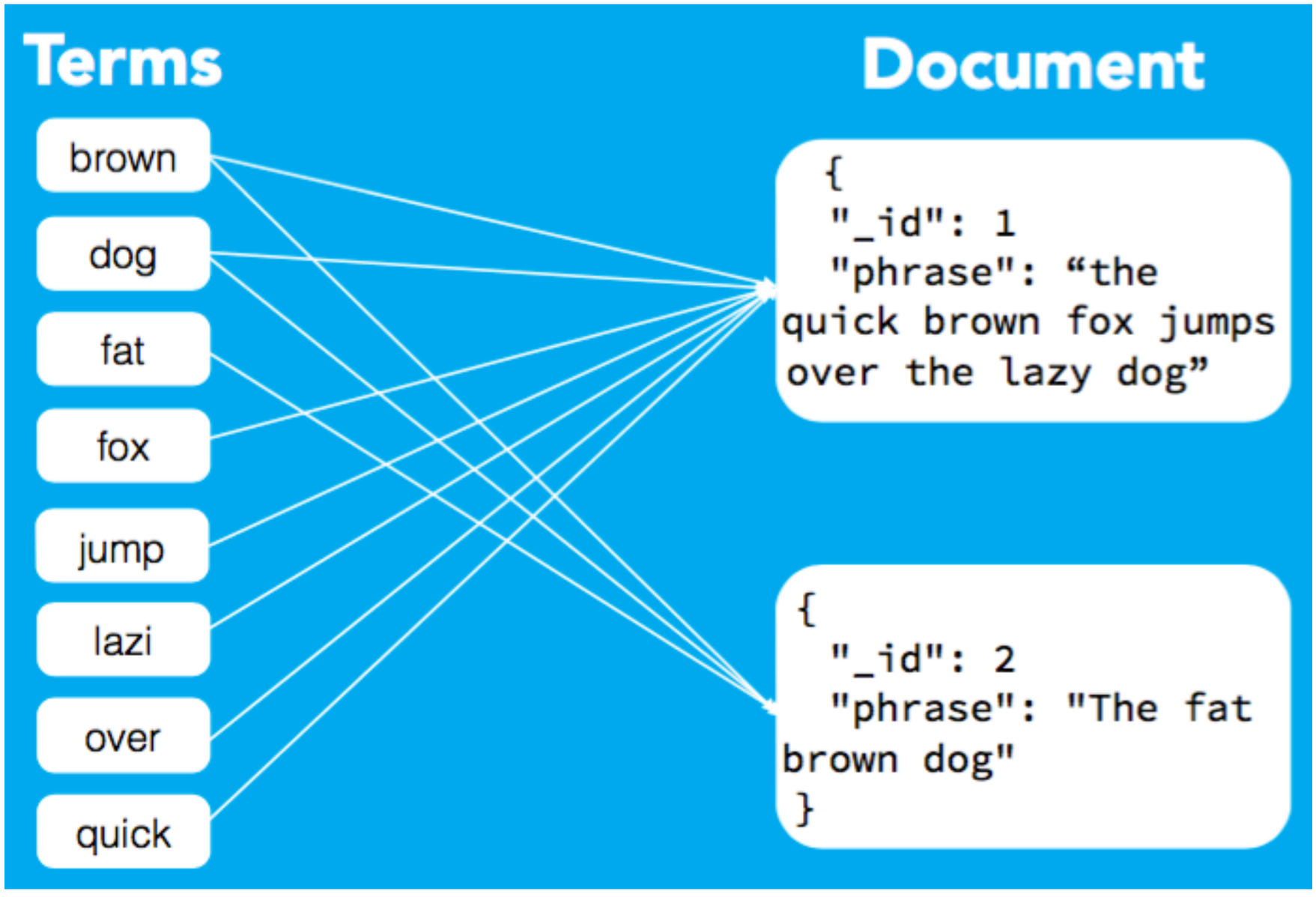
- AWS Quantum Ledger
- Azure SQL Database Ledger
- Google ?
- Hyper ledger Fabric

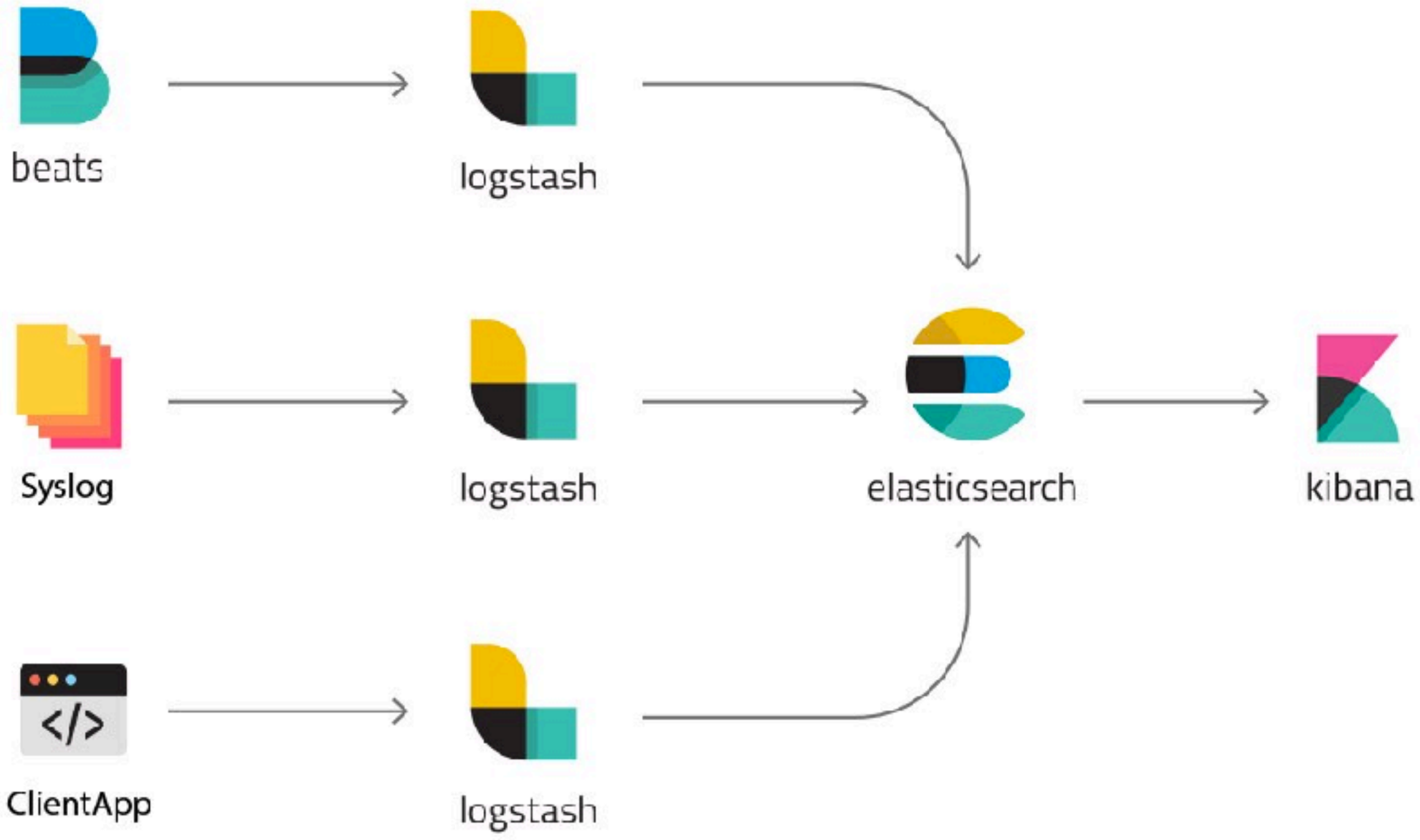


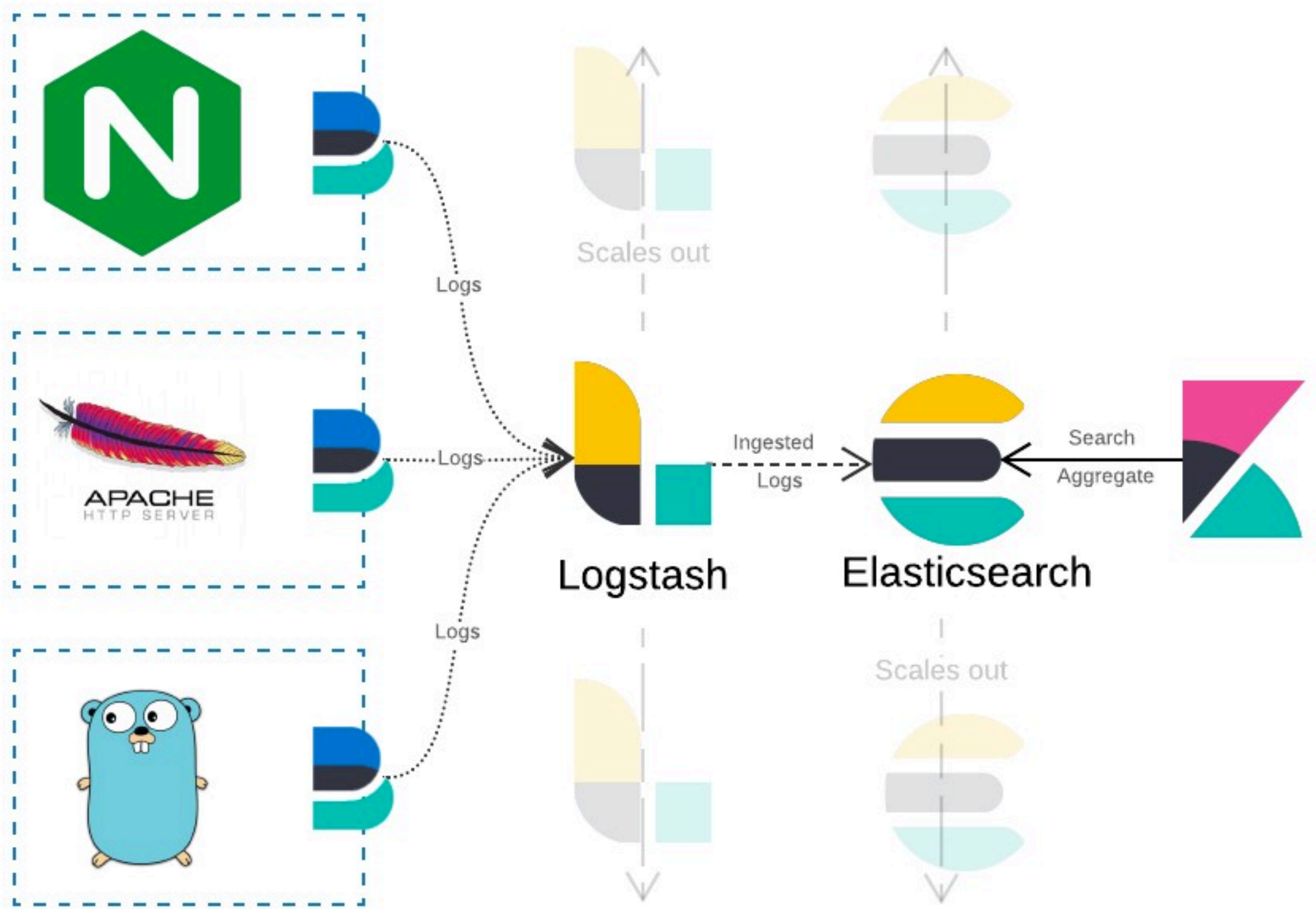
geo Spatial

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

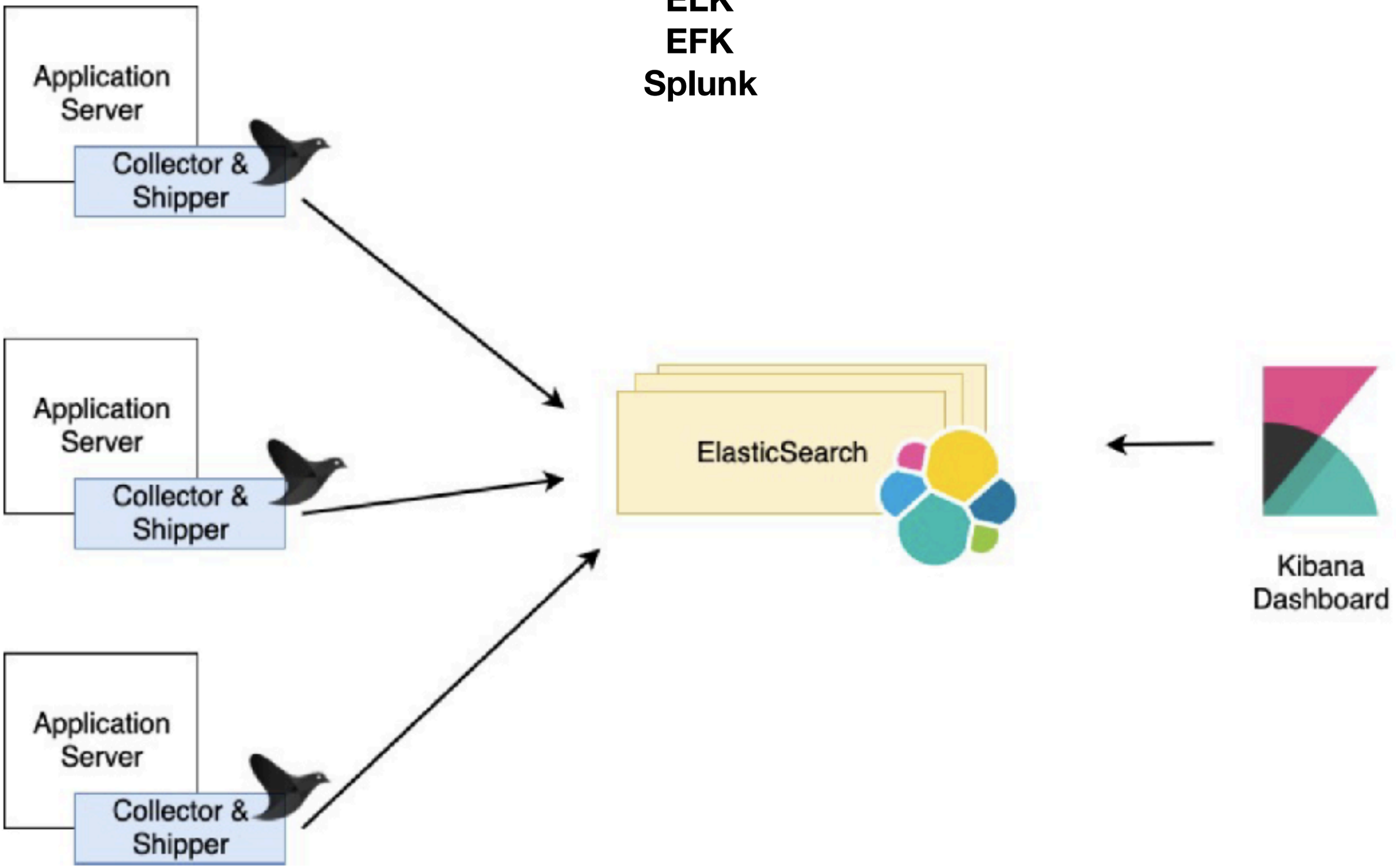



Terms
Document
brown
dog
fat
fox
jump
lazi
over
quick
{
"_id": 1
"phrase": “the quick brown fox jumps over the lazy dog”
{
"_id": 2
"phrase": "The fat brown dog"
}


beats
logstash
Syslog
logstash
elasticsearch
kibana
ClientApp
logstash


Logs
Scales out
APACHE
HTTP SERVER
Logs
Ingested
Logs
Search
Aggregate
Logstash
Elasticsearch
Logs
Scales out

ELK
EFK
Splunk



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

Row Store

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

Column Store

Country	India
	India
	Germany
	US
Product	Chocolate
	Ice-cream
	Chocolate
	Noodle
Sales	1000
	2000
	4000
	500

Row-oriented

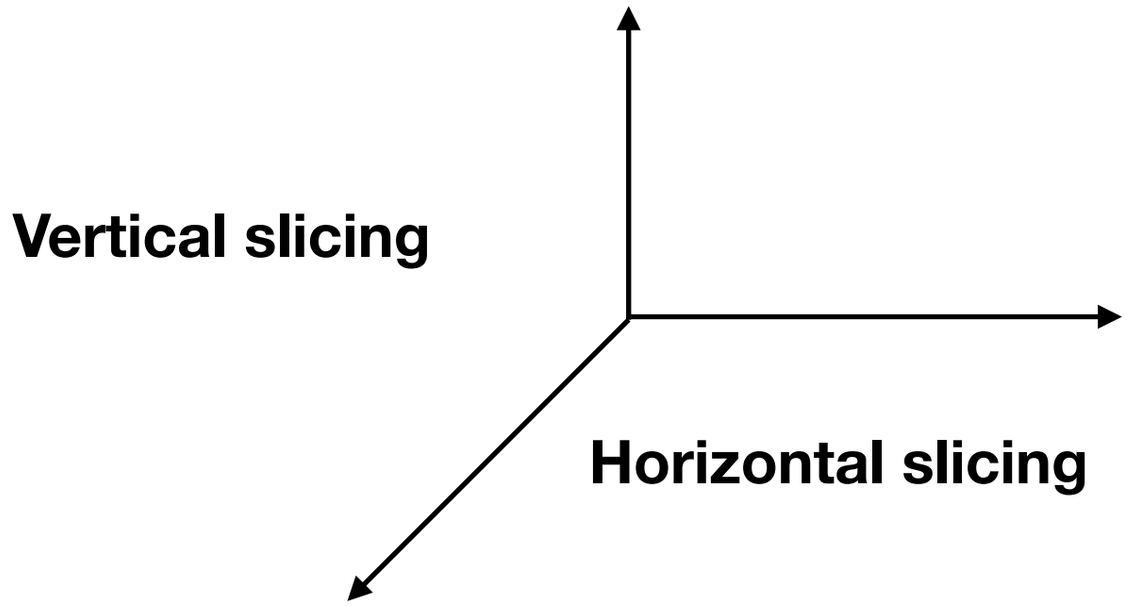
ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

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



rowkey1	column family (CF11)			column family (CF12)	
	column111	column112	column113	column121	column122
	version1111 value1111	version1121 value1121	version1121 value1131	version1211 value1211	version1221 value1221
	version1112 value1112	version1122 value1122			version1222 value1222
		version1123 value1123			
		version1124 value1124			

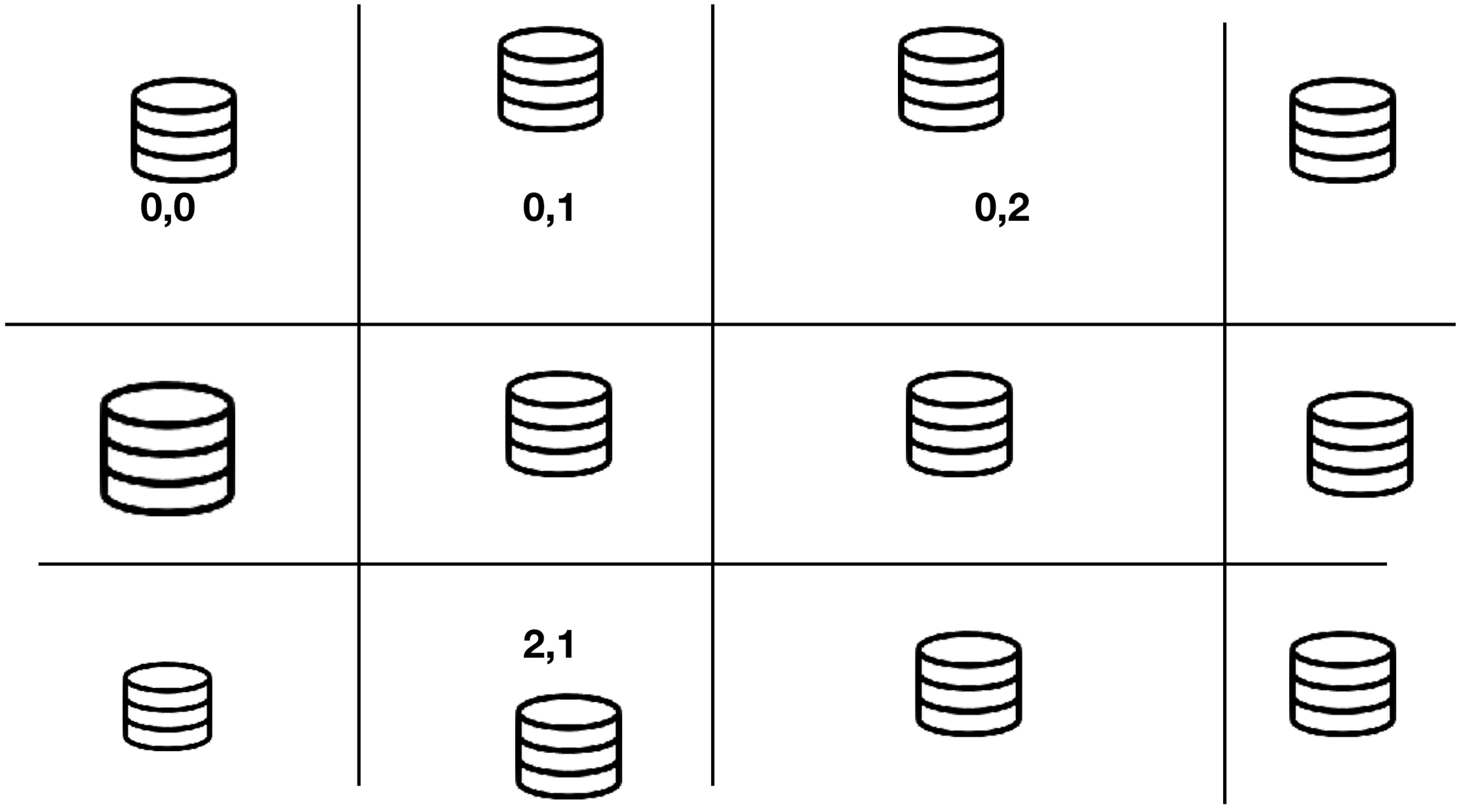
{Ordered, cust name, amount} , {itemcode, qty,price}

(Row partition id (shard key), column family id, row id)

multidimensional map (map of maps)

```
{
  "rowkey1": {"cf11": {"column111": {"version1111": value1111,
                                     "version1112": value1112},
                       "column112": {"version1121": value1121,
                                     "version1122": value1122,
                                     "version1123": value1123,
                                     "version1124": value1124},
                       "column113": {"version1131": value1131}
                      },
             "cf12": {"column121": {"version1211": value1211},
                       "column122": {"version1221": value1221},
                                     "version1222": value1222}
                      }
             },
   "rowkey2": {"cf11": {"column111": {"version2111": value2111,
                                      "version2112": value2112},
                        "column112": {"version2121": value2121,
                                      "version2122": value2122,
                                      "version2123": value 2123,
                                      "version2124": value2124}
                       },
              "cf12": {"column121": {"version2211": value2211},
                       "column122": {"version2221": value2221}
                       }
             }
}
```

Row partition id,
Col partition id



Column Family 1

Column Family 2

	cf1:col-A	cf1:col-B	cf2:col-Foo	cf2:col-XYZ	cf2:foobar
Region 1					
row-1					
row-10					
row-18	A18 - v1	B18 - v3	Foo18 - v1	XYZ18 - v2	foobar18 - v1
Region 2					
row-2					
row-5					
row-6					
row-7					

Physical Coordinates for a Cell: *Region Directory* → *Column Family Directory* → *Row Key* → *Column Family Name* → *Column Qualifier* → *Version*

	CF1:colA	CF1:colB	CF1:colC
Row1	@time7: value3		
Row10	@time2: value1	@time2: value1	
Row11	@time6: value2		
Row2	@time4: value1		@time4: value1

Row-oriented

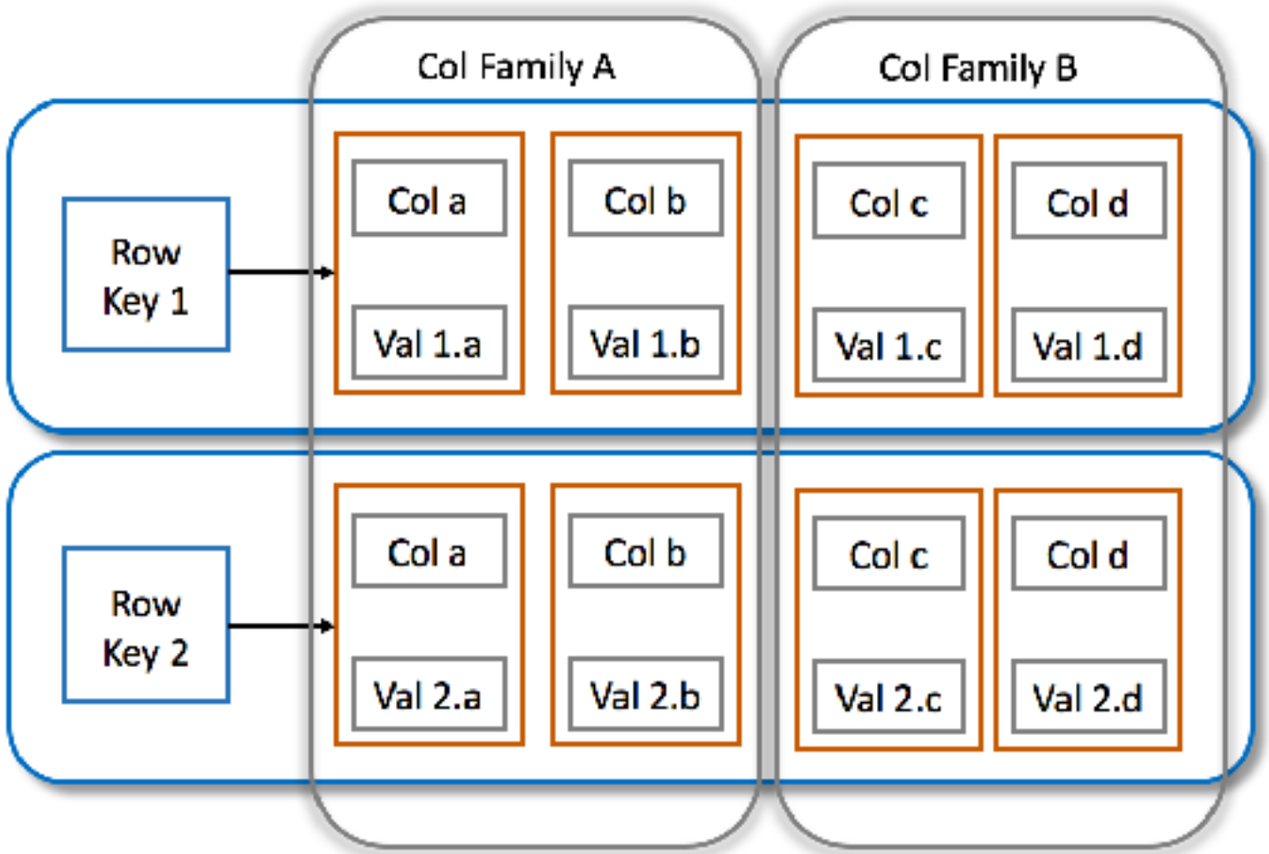
ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

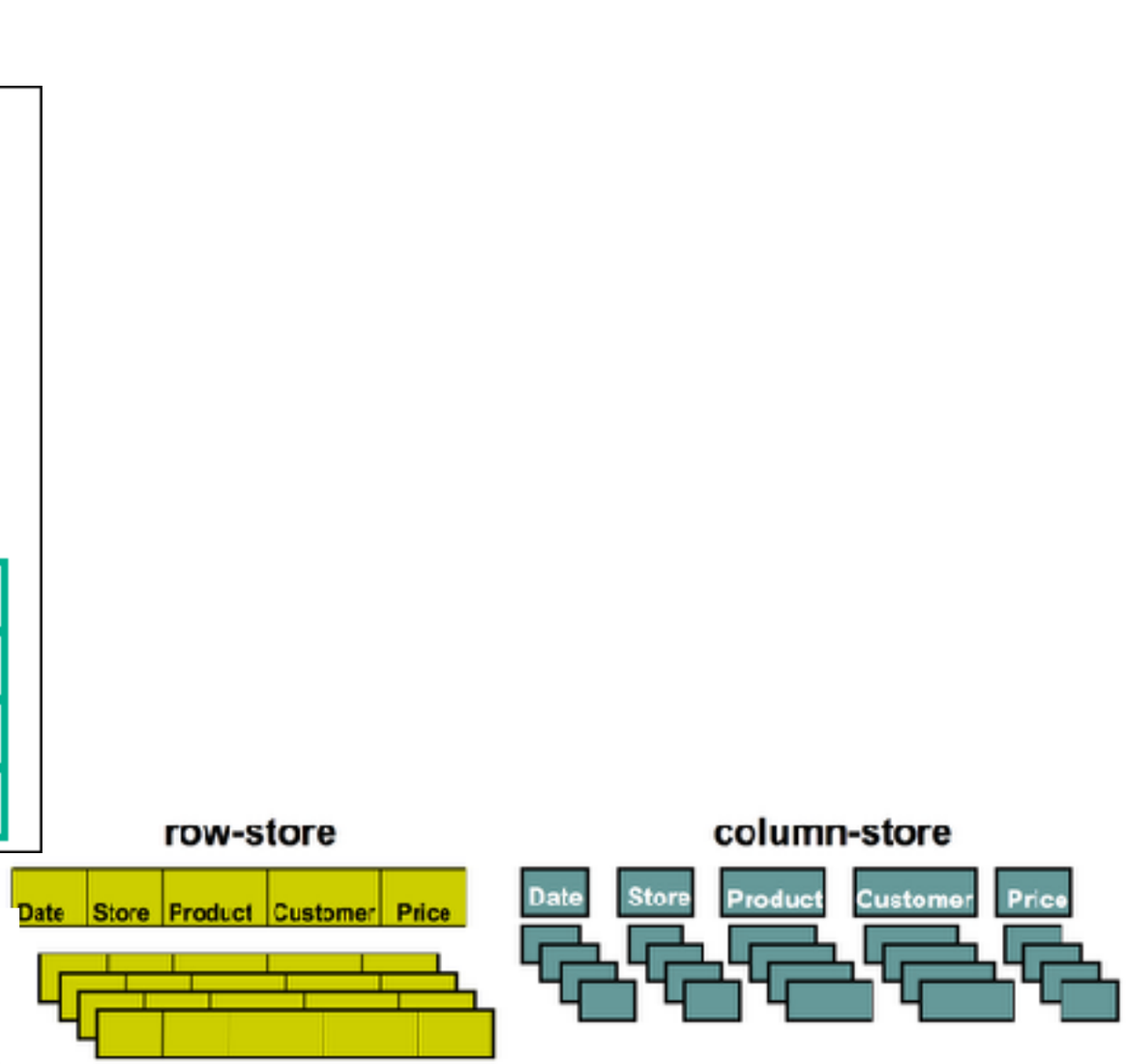
Column-oriented

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





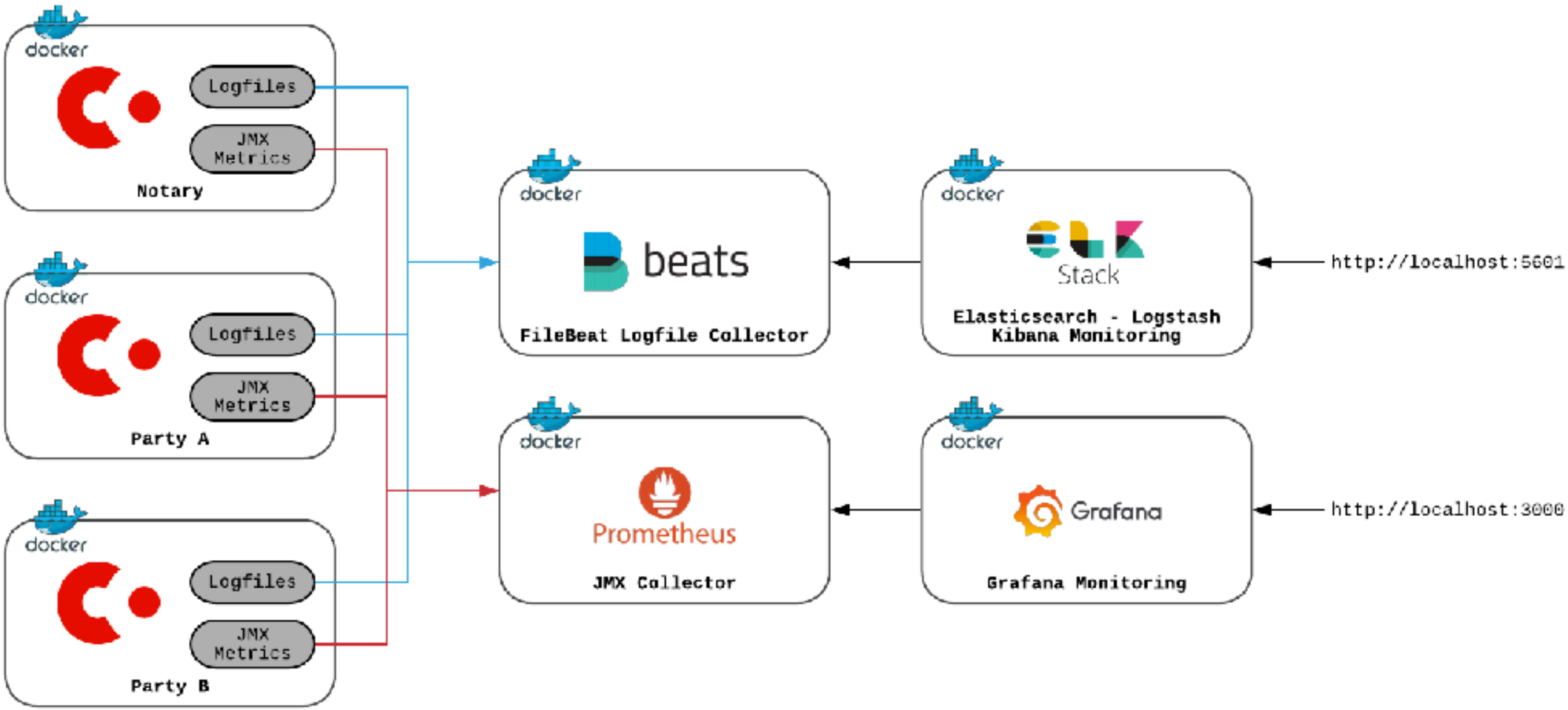
column-family	Columnar
sparse data model	
multi-dimensional map	
column-families are not independently accessible.	every column is stored separately
NoSQL	SQL interface
Reads that use the partition key are incredibly fast	optimized for read-mostly analytical workloads
high throughput writes	very slow writes
	data warehouses

	MongoDB	Cassandra
Expressive object model	Yes	No
Secondary indexes	Yes	No
No downtime on node failure	No	Yes
High write throughput	No	Yes

- Cassandra is optimized for really high throughput on writes. **If your use case is read-heavy (like cache) then Cassandra might not be an ideal choice.**
- It does not support complete transaction management across the tables. Not ACID compliant system.
- Secondary Index not supported. Have to rely on Elastic search /Solr for Secondary index and the custom sync component has to be written.

not possible to aggregate data in a query. Sums and averages like information have to be done by the client-side application.

Use Cassandra as the primary data store for capturing information as it arrives into the system; but then build “query-optimised views” of subsets of that data in other databases specialised to the kinds of access users require. Use an indexing engine such as SOLR to provide full-text search across records, or a graph database such as Neo4j to store metadata in a way that supports “traversal-heavy” queries that join together many different kinds of entity. Use an RDBMS when you need the full power and flexibility of SQL to express ad hoc queries that explore the data in multiple dimensions.


docker
Logfiles
JMX Metrics
Notary
docker
Logfiles
JMX Metrics
Party A
docker
Logfiles
JMX Metrics
Party B
docker
beats
FileBeat Logfile Collector
docker
Prometheus
JMX Collector
docker
Stack
Elasticsearch - Logstash
Kibana Monitoring
http://localhost:5601
docker
Grafana
Grafana Monitoring
http://localhost:3000

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	☑

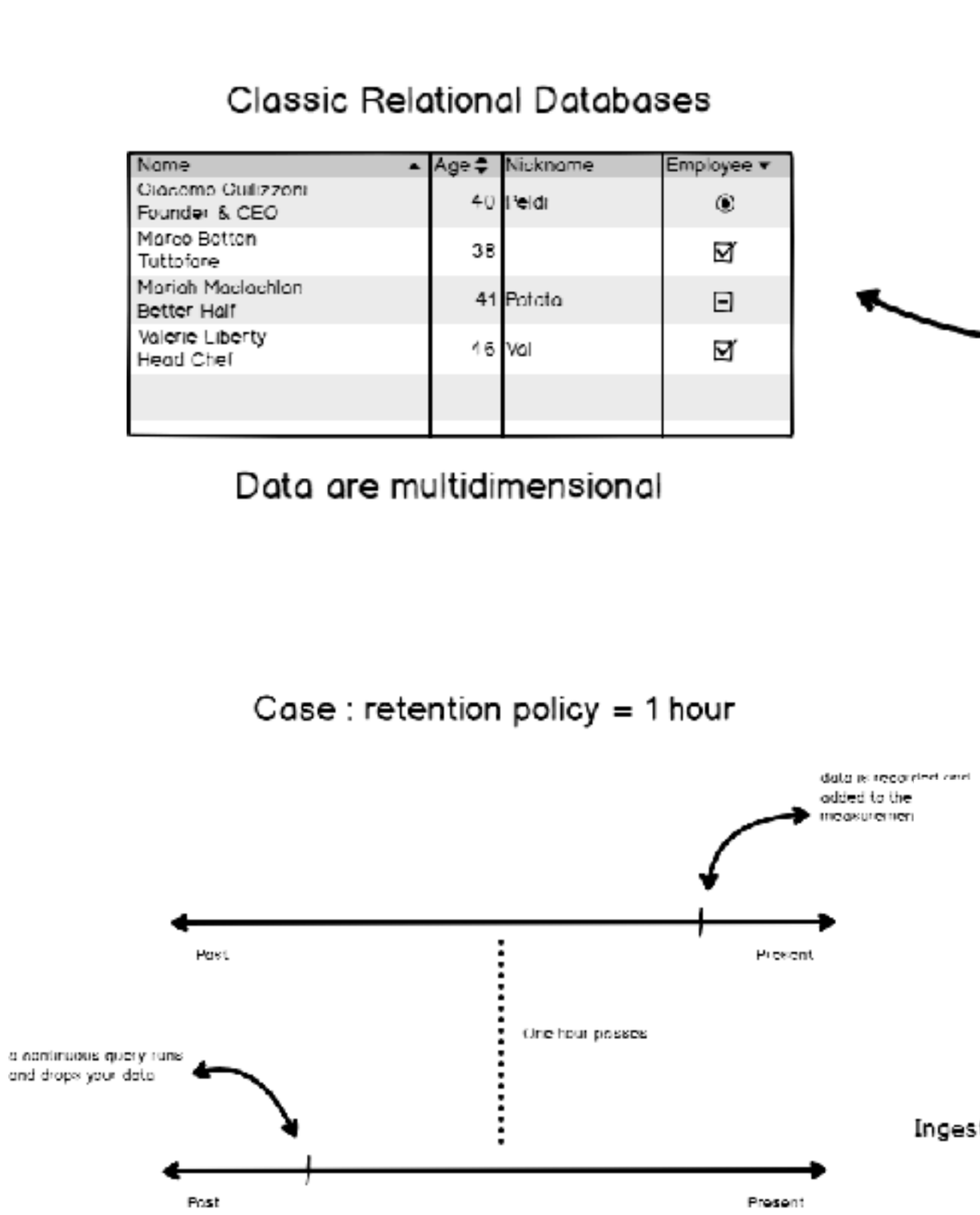
Data are multidimensional

Time Series Databases

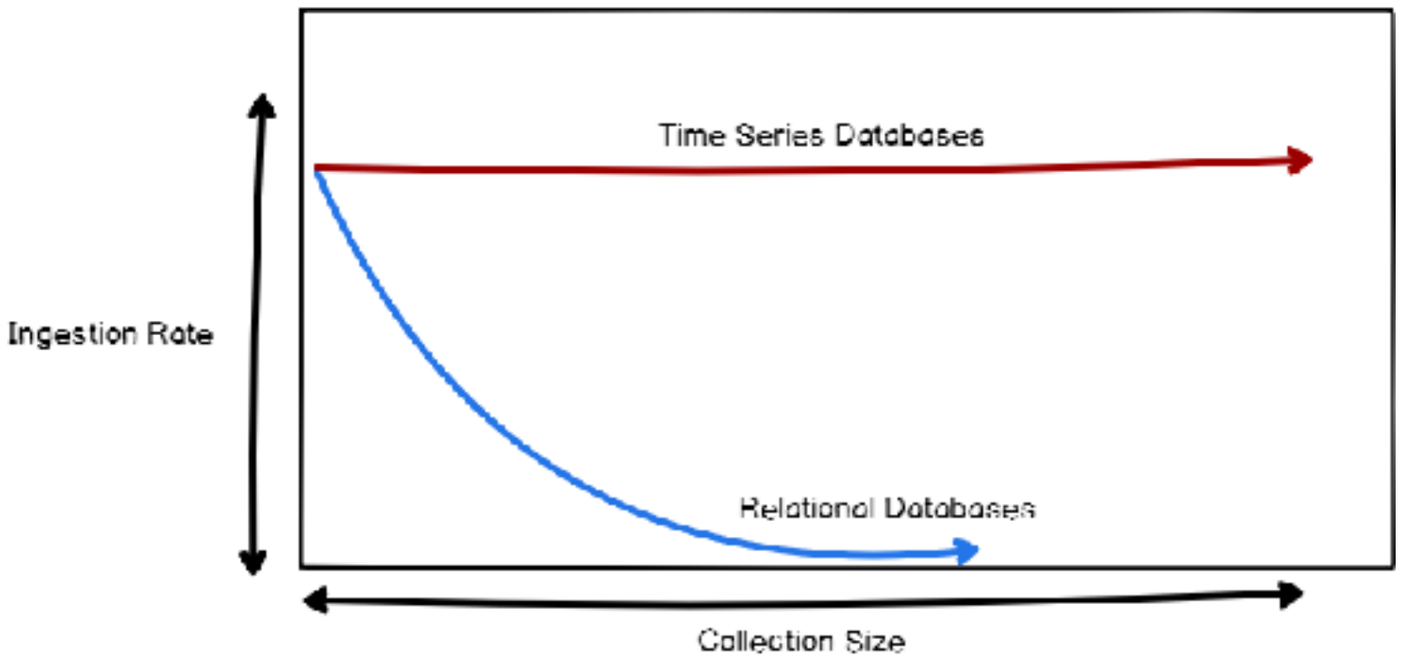
Sensor Temperature	Time
39.6	12/04/19 @ 14:12
41.2	12/04/19 @ 14:13
12.4	14/04/19 @ 12:15
18.5	16/04/19 @ 10:05

Data are aggregated over time

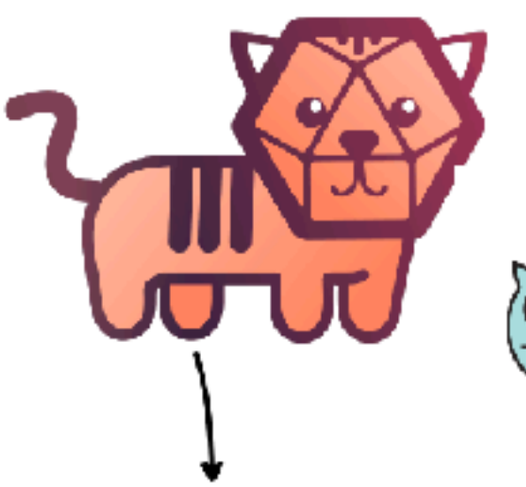
Case : retention policy = 1 hour



DBMS & TSDB Difference

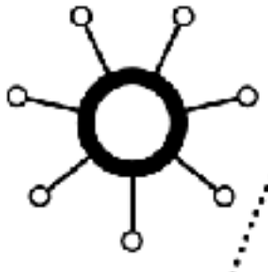


Tools that 'produce' TSDB data



monitoring done via specific query agents

collectd



StatsD

monitoring done via node exporters

Prometheus

Tools that 'consume' TSDB

chronograf

DATADOG

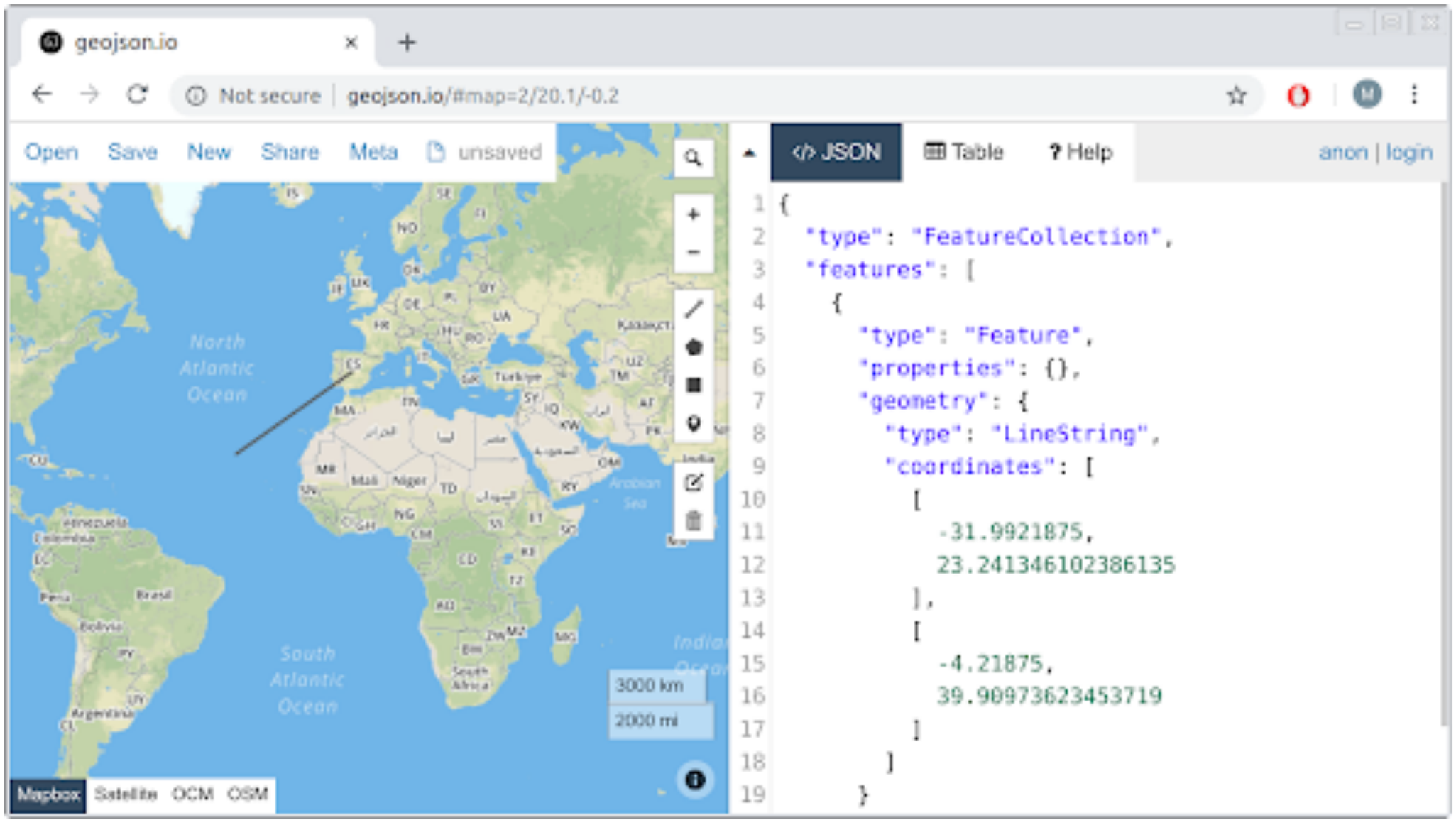
Prometheus

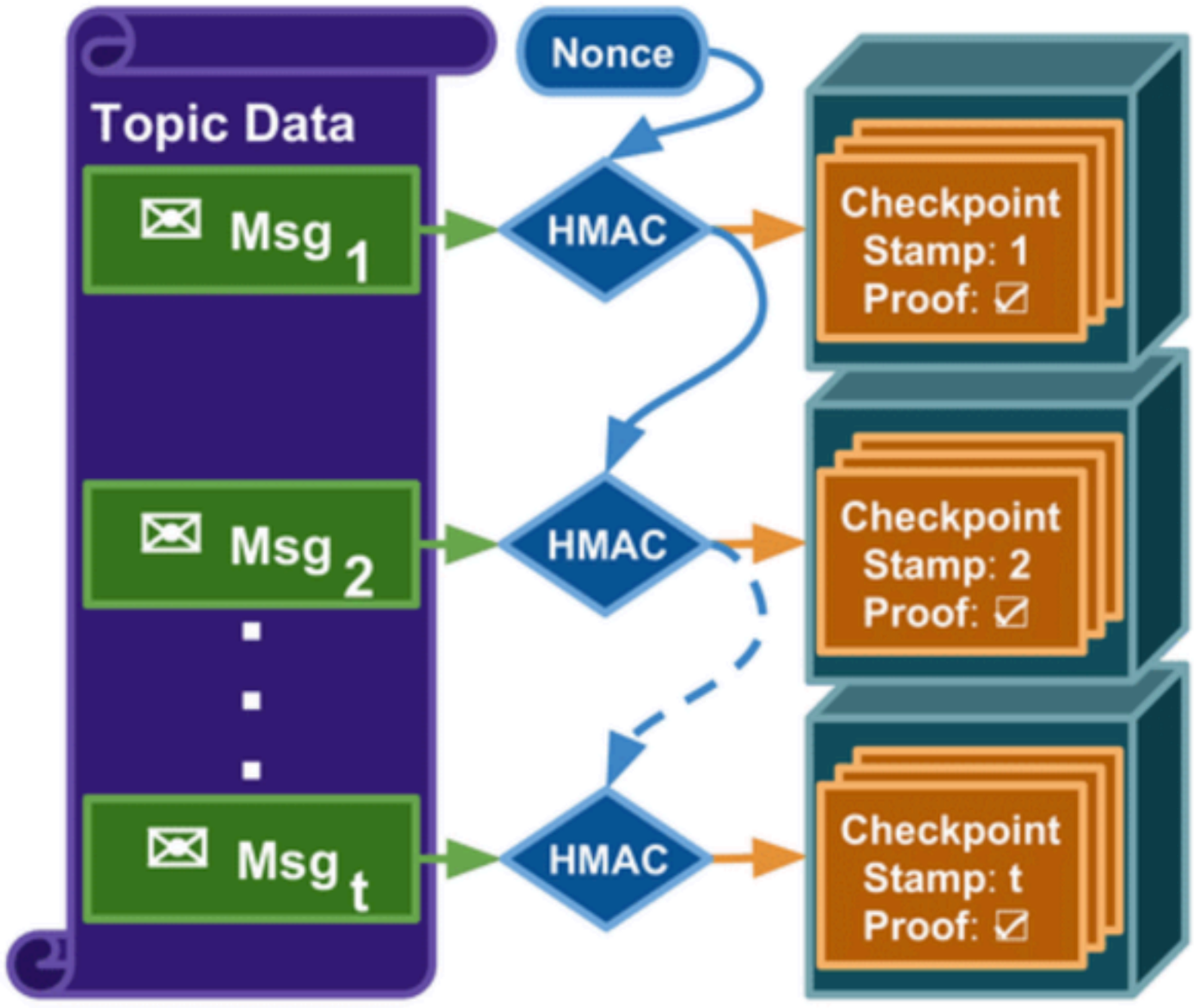
Grafana

kibana

*Non-exhaustive list

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



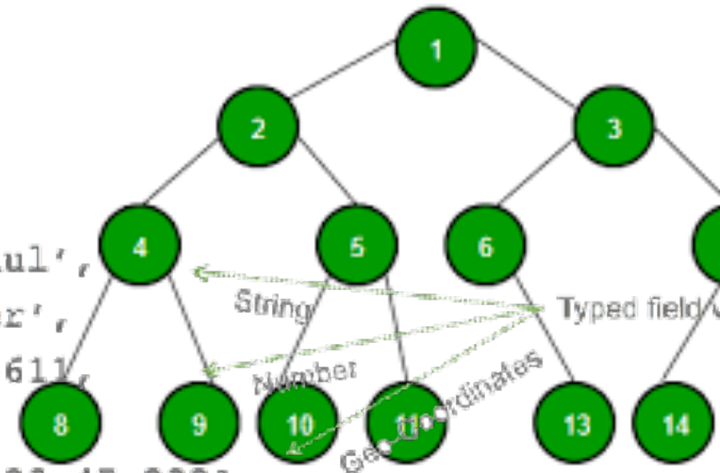

Topic Data
Msg 1
Msg 2
Msg t
Nonce
HMAC
HMAC
HMAC
Checkpoint
Stamp: 1
Proof: ☑
Checkpoint
Stamp: 2
Proof: ☑
Checkpoint
Stamp: t
Proof: ☑

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

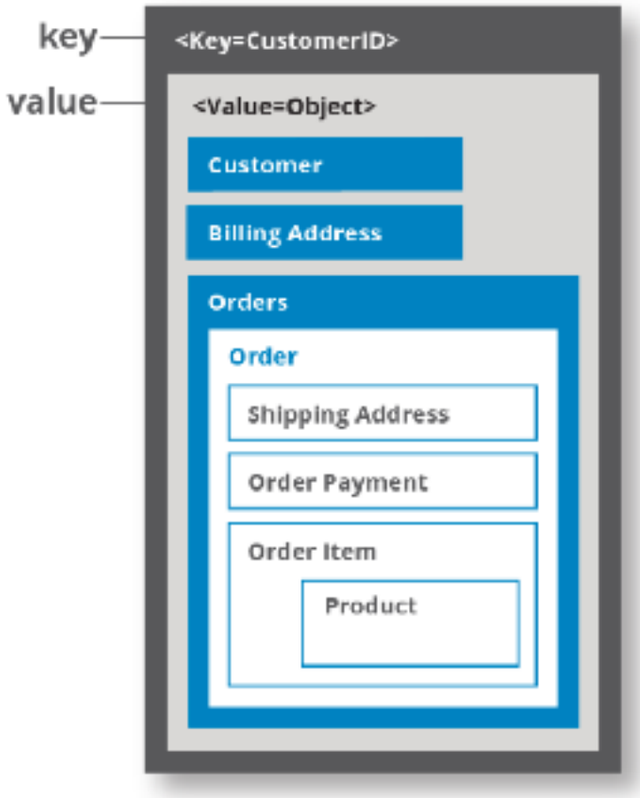
Fields

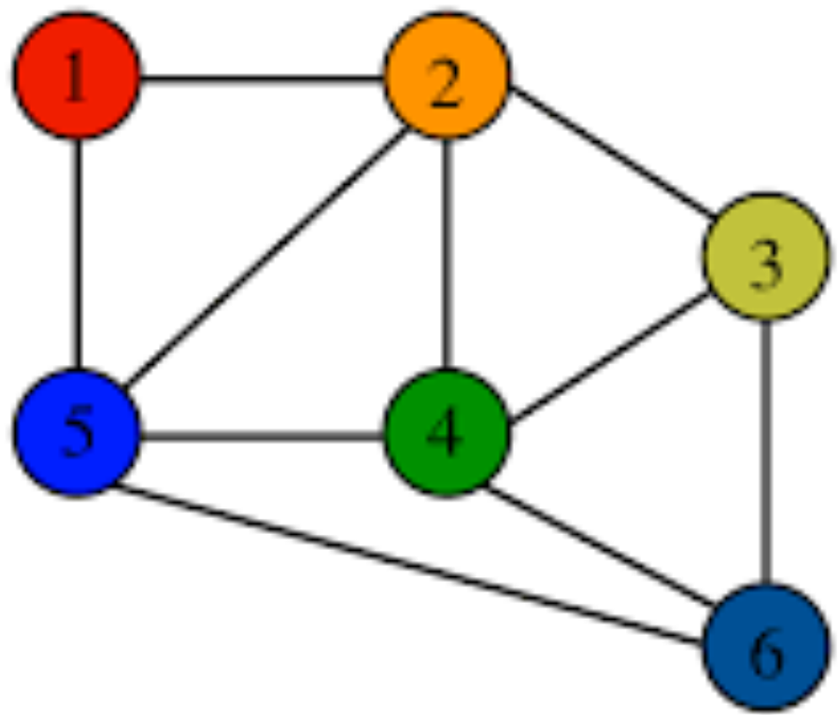



```
{
  first_name: 'Paul',
  surname: 'Miller',
  cell: 447557505611,
  city: 'London',
  location: [45.123,47.232],
  Profession: ['banking', 'finance', 'trader'],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: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

Fields can contain an array of documents

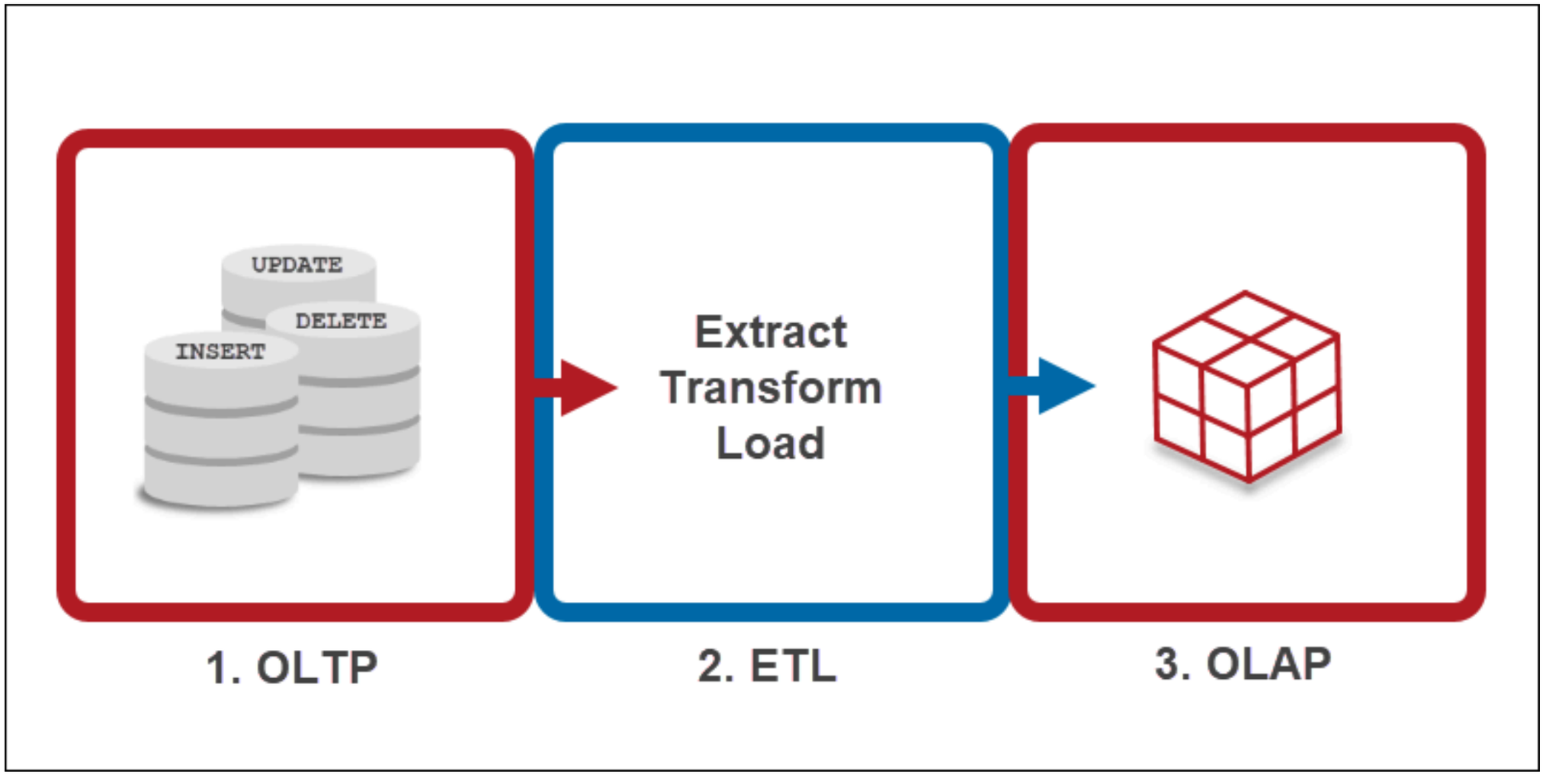
key	value
123	123 Main St.
126	(805) 477-3900





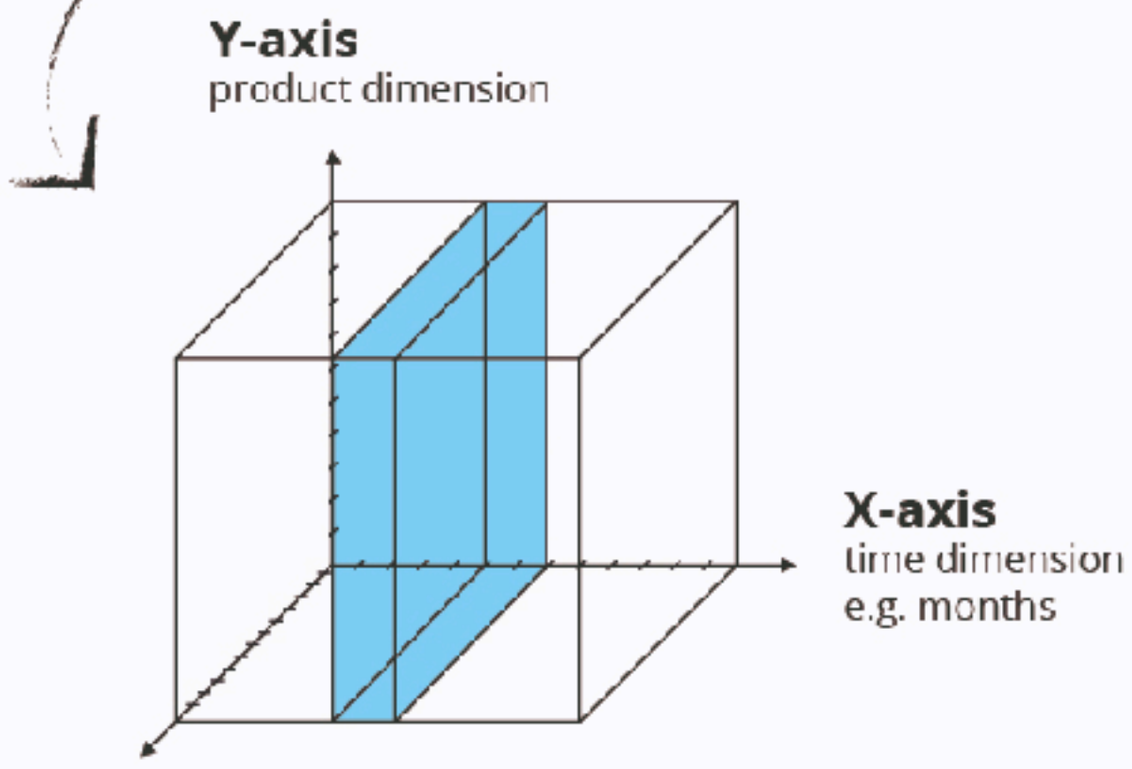
Wide Column

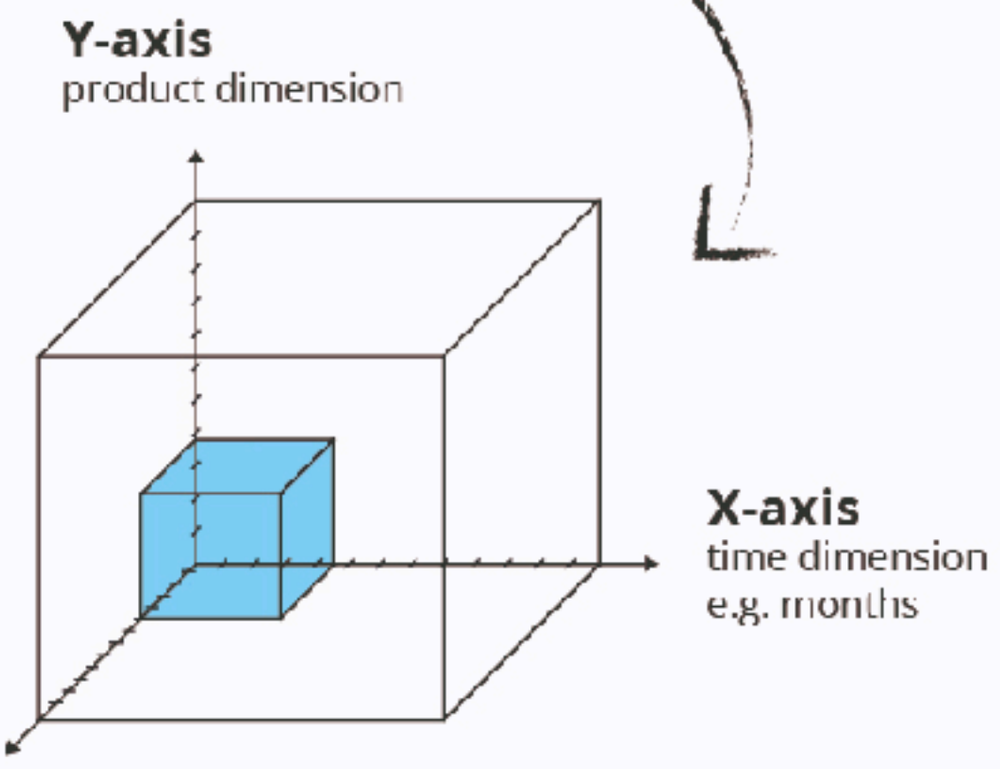
- Spotify uses Cassandra to store user profile attributes and metadata about artists, songs, etc. for better personalization
- Facebook initially built its revamped Messages on top of HBase, but is now also used for other Facebook services like the Nearby Friends feature and search indexing
- Outbrain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

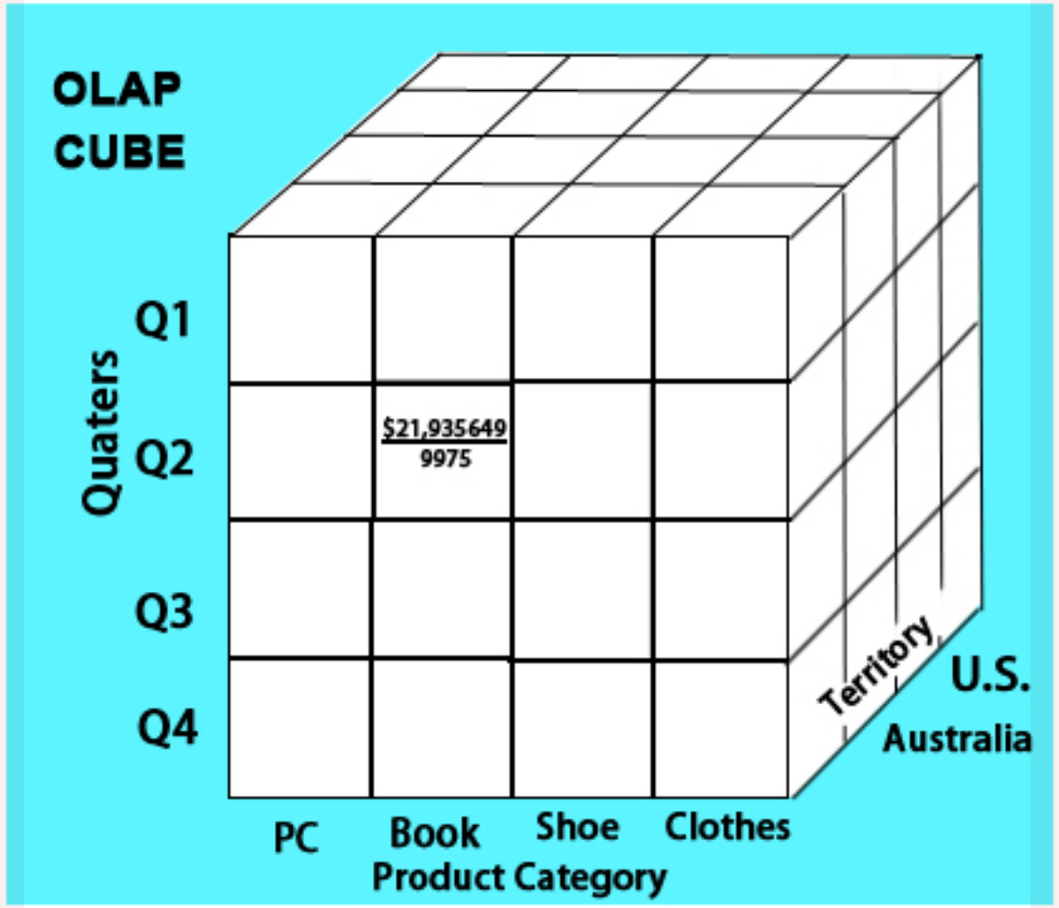

UPDATE
DELETE
INSERT
Extract
Transform
Load
1. OLTP
2. ETL
3. OLAP

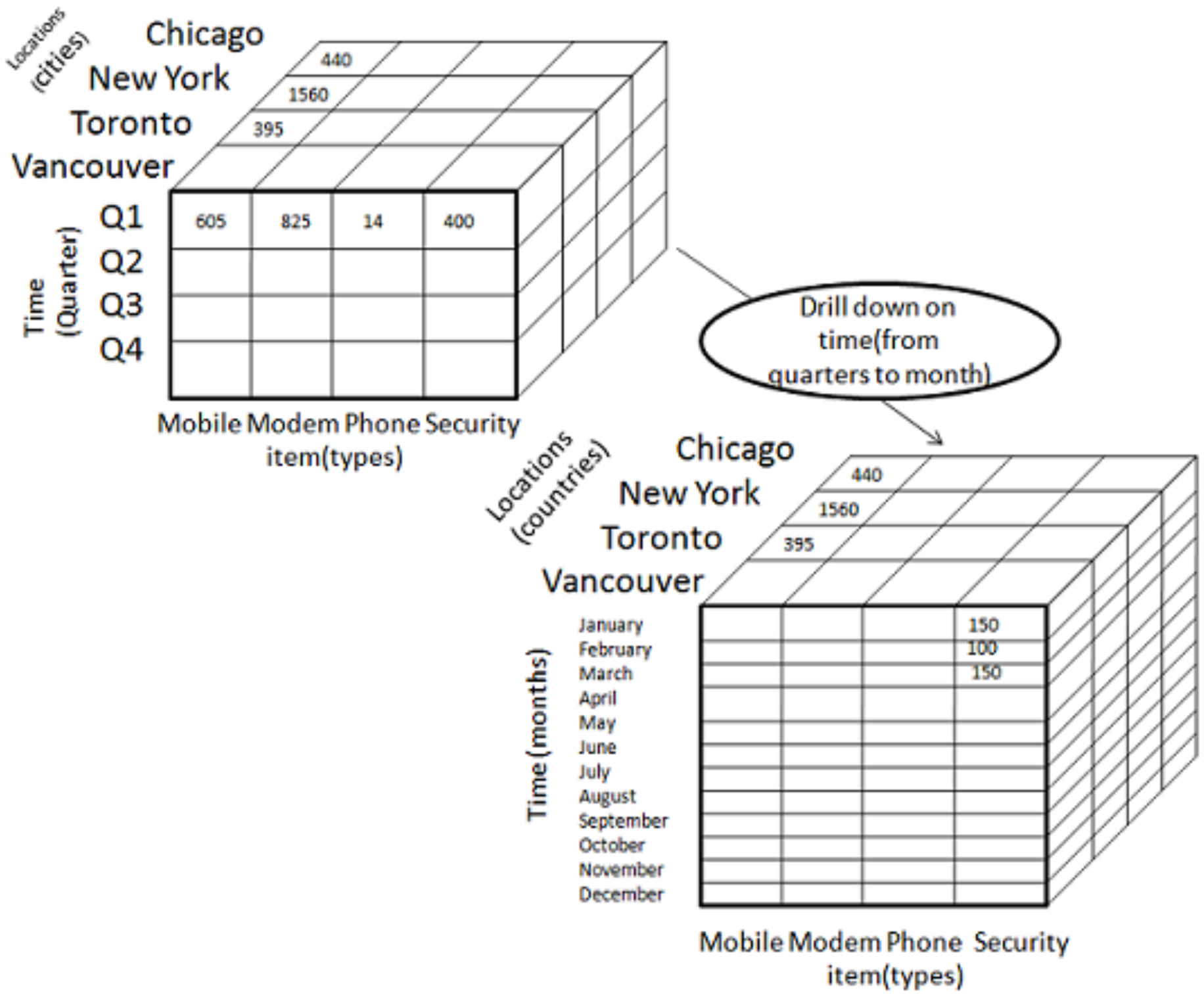
Slicing & Dicing

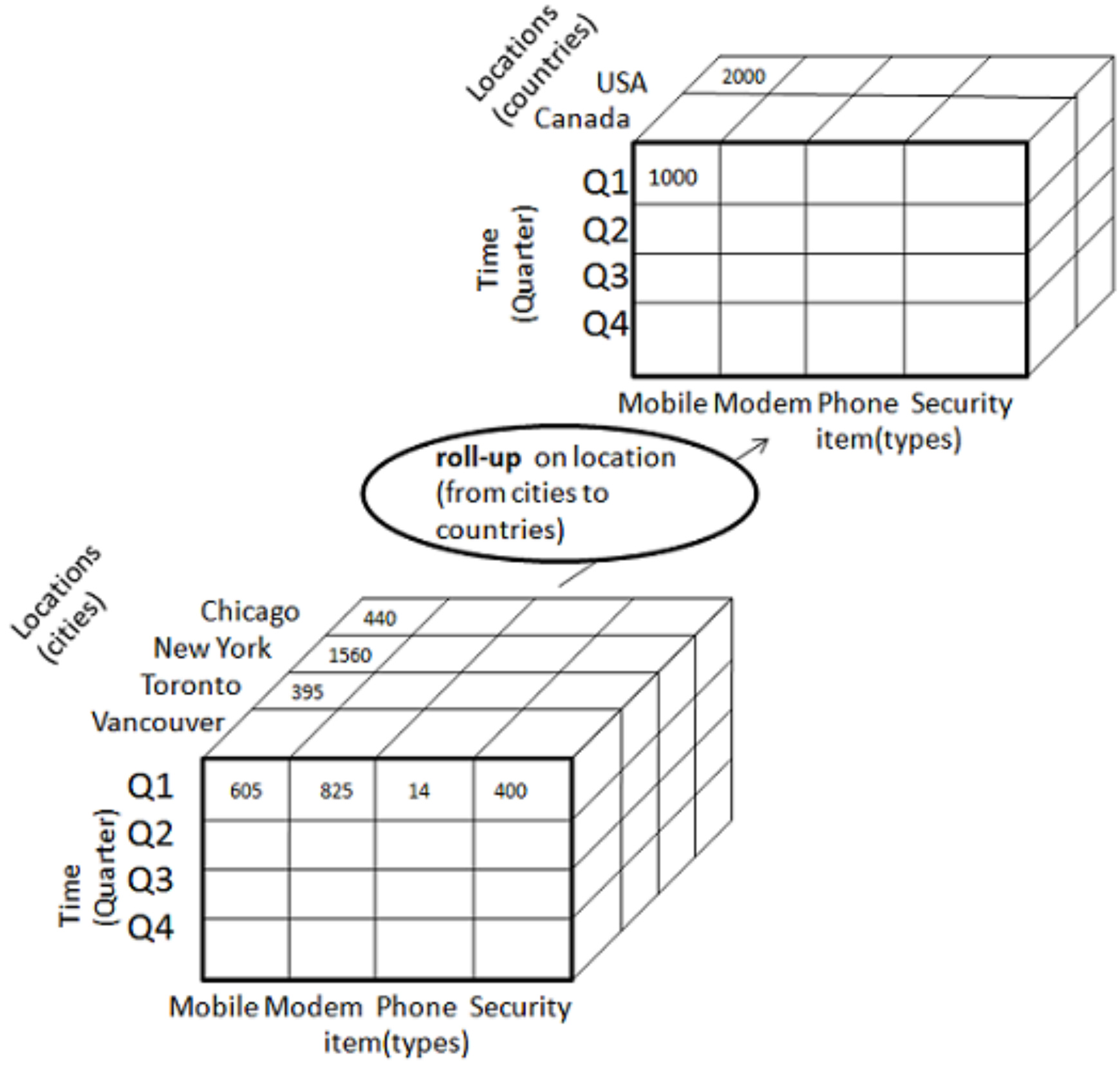
OLAP cube operations

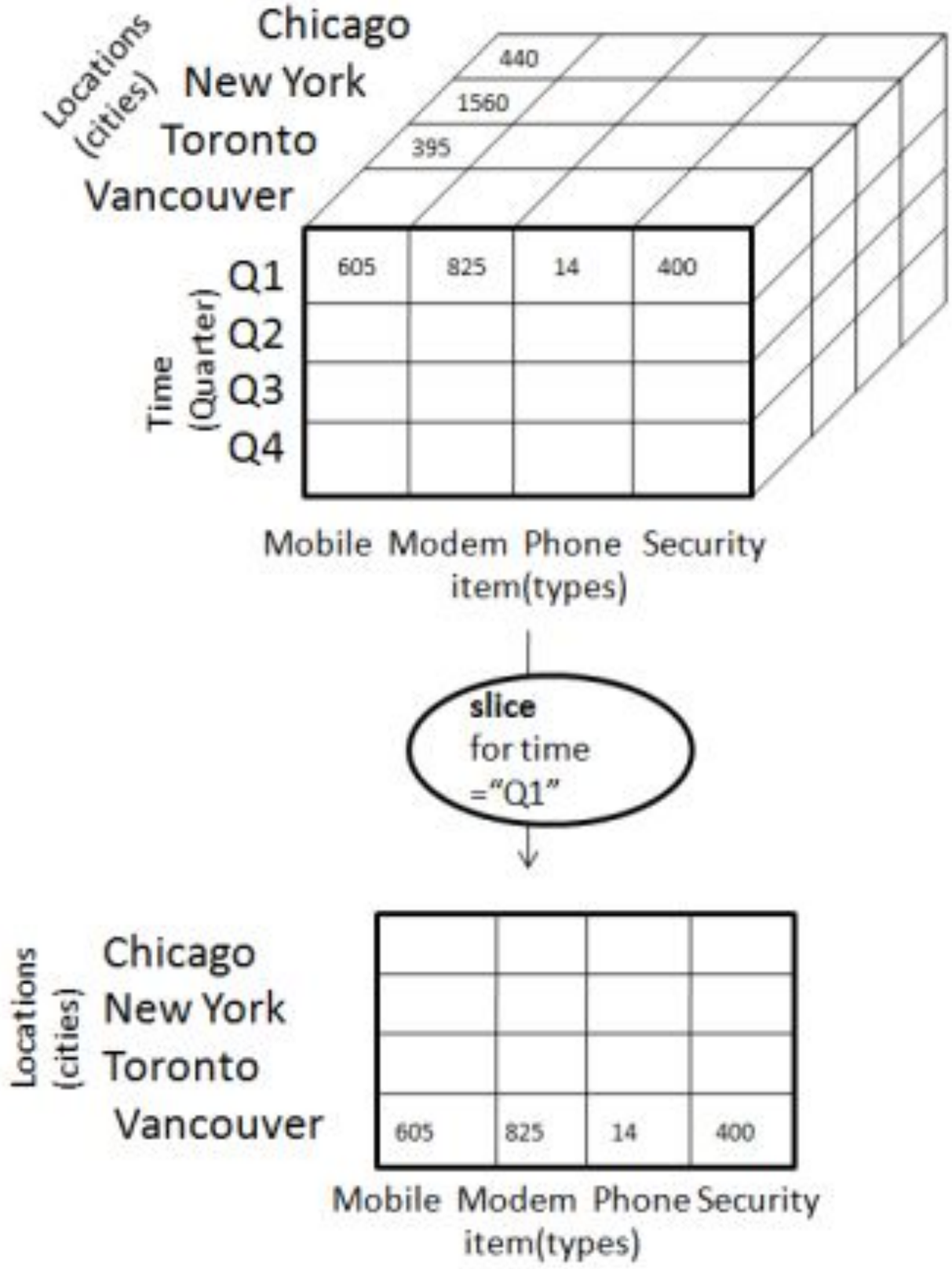


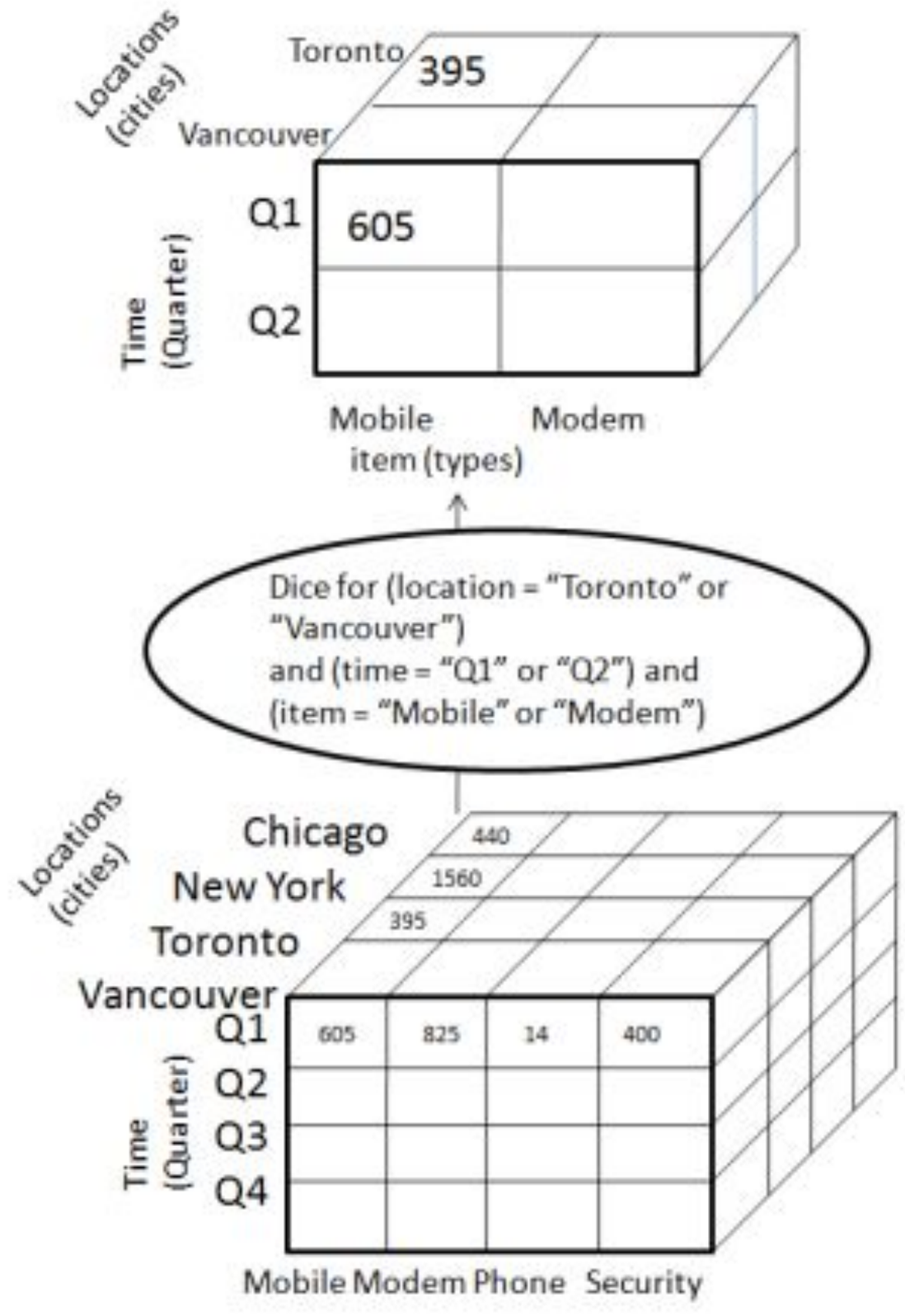


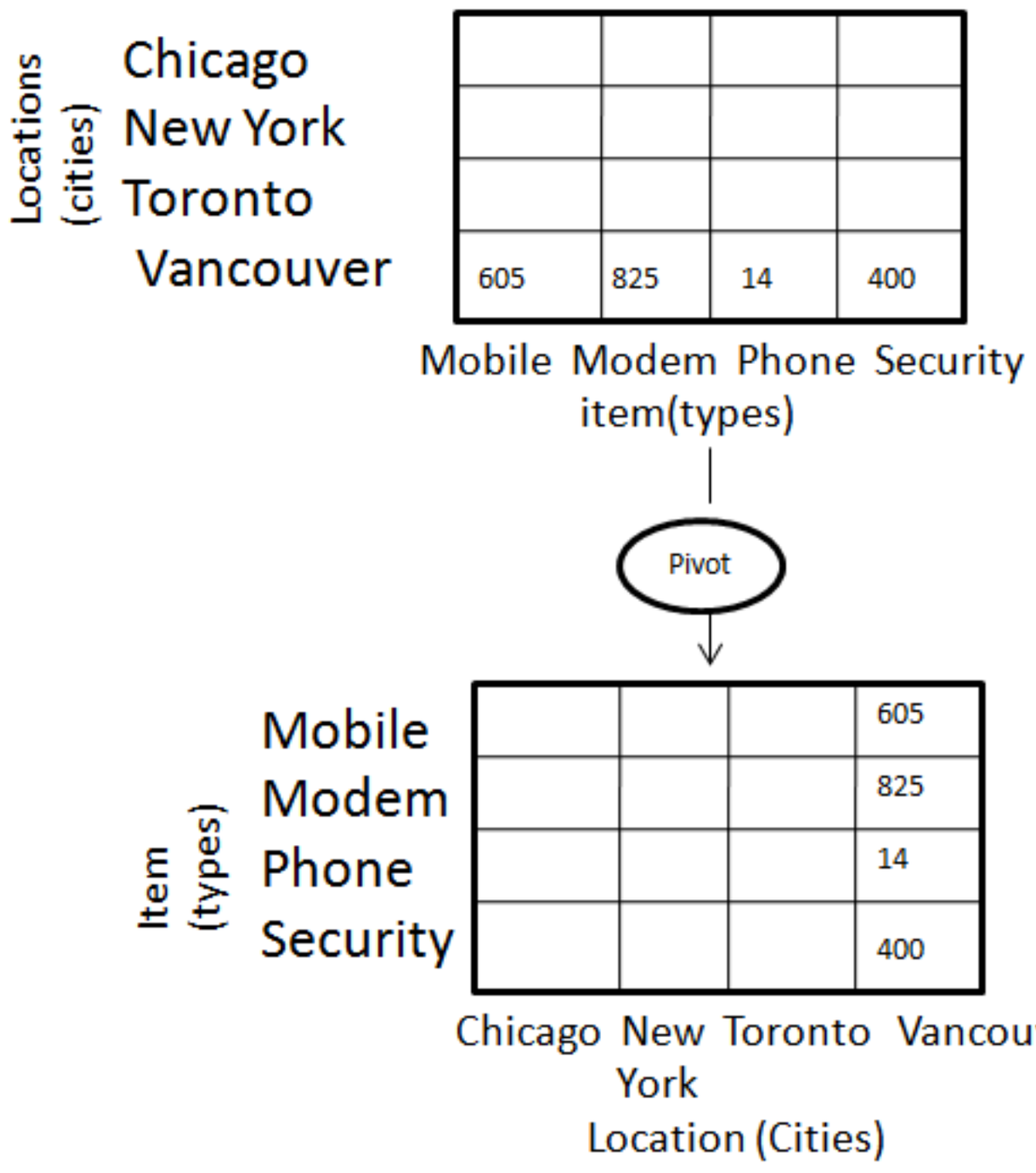

OLAP
CUBE
Quaters
Q1
Q2
Q3
Q4
$21,935649
9975
PC
Book
Shoe
Clothes
Product Category
Territory
U.S.
Australia


Locations (cities)
Chicago
New York
Toronto
Vancouver
440
1560
395
Time (Quarter)
Q1
Q2
Q3
Q4
605
825
14
400
Mobile Modem Phone Security
item(types)
Drill down on time(from quarters to month)
Locations (countries)
Chicago
New York
Toronto
Vancouver
440
1560
395
Time (months)
January
February
March
April
May
June
July
August
September
October
November
December
150
100
150
Mobile Modem Phone Security
item(types)

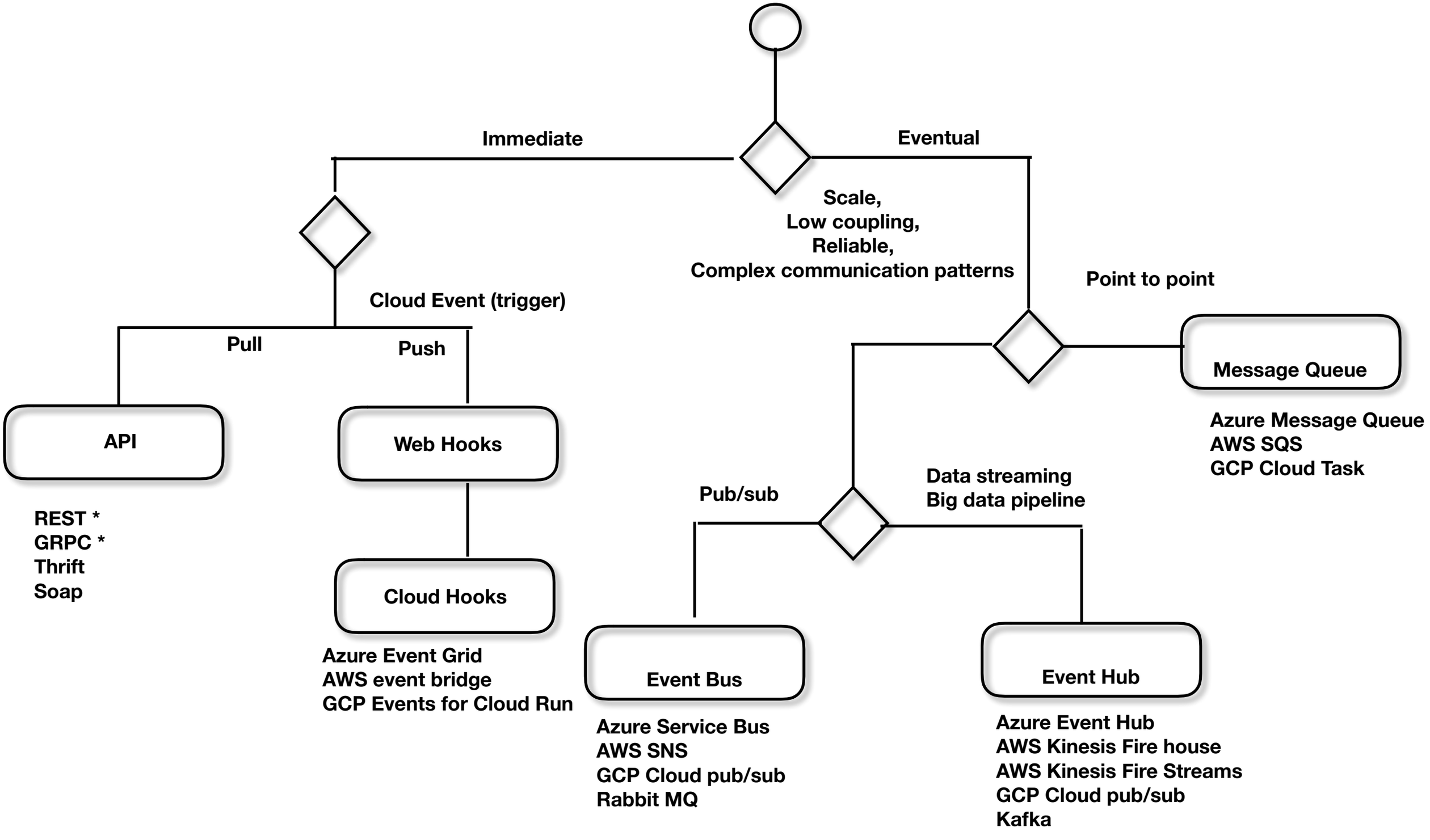

Locations (countries)
USA
Canada
2000
1000
Q1
Q2
Q3
Q4
Time (Quarter)
Mobile Modem Phone Security
item(types)
roll-up on location (from cities to countries)
Locations (cities)
Chicago
New York
Toronto
Vancouver
440
1560
395
605
825
14
400
Q1
Q2
Q3
Q4
Time (Quarter)
Mobile Modem Phone Security
item(types)


Locations (cities)
Chicago
New York
Toronto
Vancouver
440
1560
395
Time (Quarter)
Q1
Q2
Q3
Q4
605
825
14
400
Mobile
Modem
Phone
Security
item(types)
slice
for time
="Q1"
Locations (cities)
Chicago
New York
Toronto
Vancouver
605
825
14
400
Mobile
Modem
Phone
Security
item(types)


Locations (cities)
Toronto
395
Vancouver
Q1
605
Q2
Time (Quarter)
Mobile
Modem
item (types)
Dice for (location = "Toronto" or "Vancouver")
and (time = "Q1" or "Q2") and
(item = "Mobile" or "Modem")
Locations (cities)
Chicago
440
New York
1560
Toronto
395
Vancouver
Q1
605
825
14
400
Q2
Q3
Q4
Time (Quarter)
Mobile
Modem
Phone
Security


Locations (cities)
Chicago
New York
Toronto
Vancouver
605
825
14
400
Mobile
Modem
Phone
Security
item(types)
Pivot
Item (types)
Mobile
Modem
Phone
Security
605
825
14
400
Chicago
New York
Toronto
Vancouver
Location (Cities)

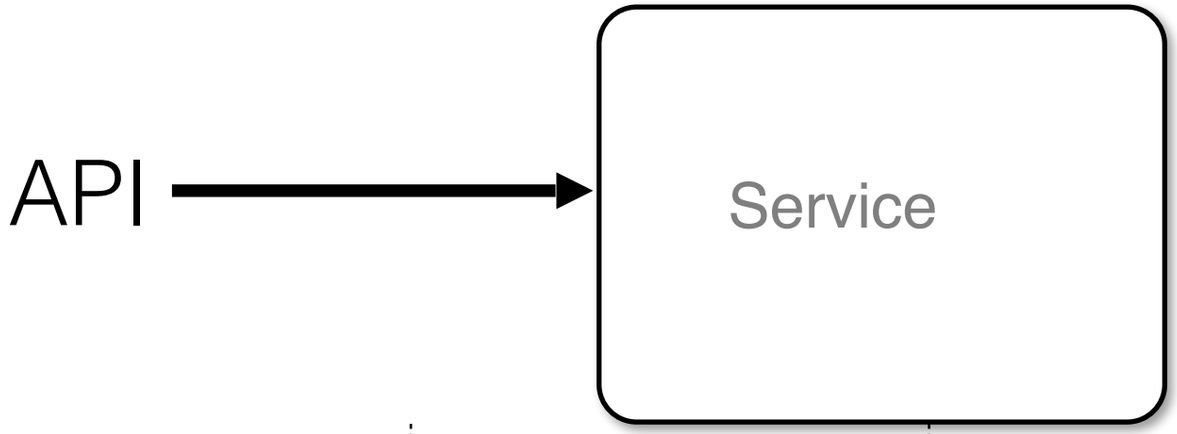
Choose Communication protocol



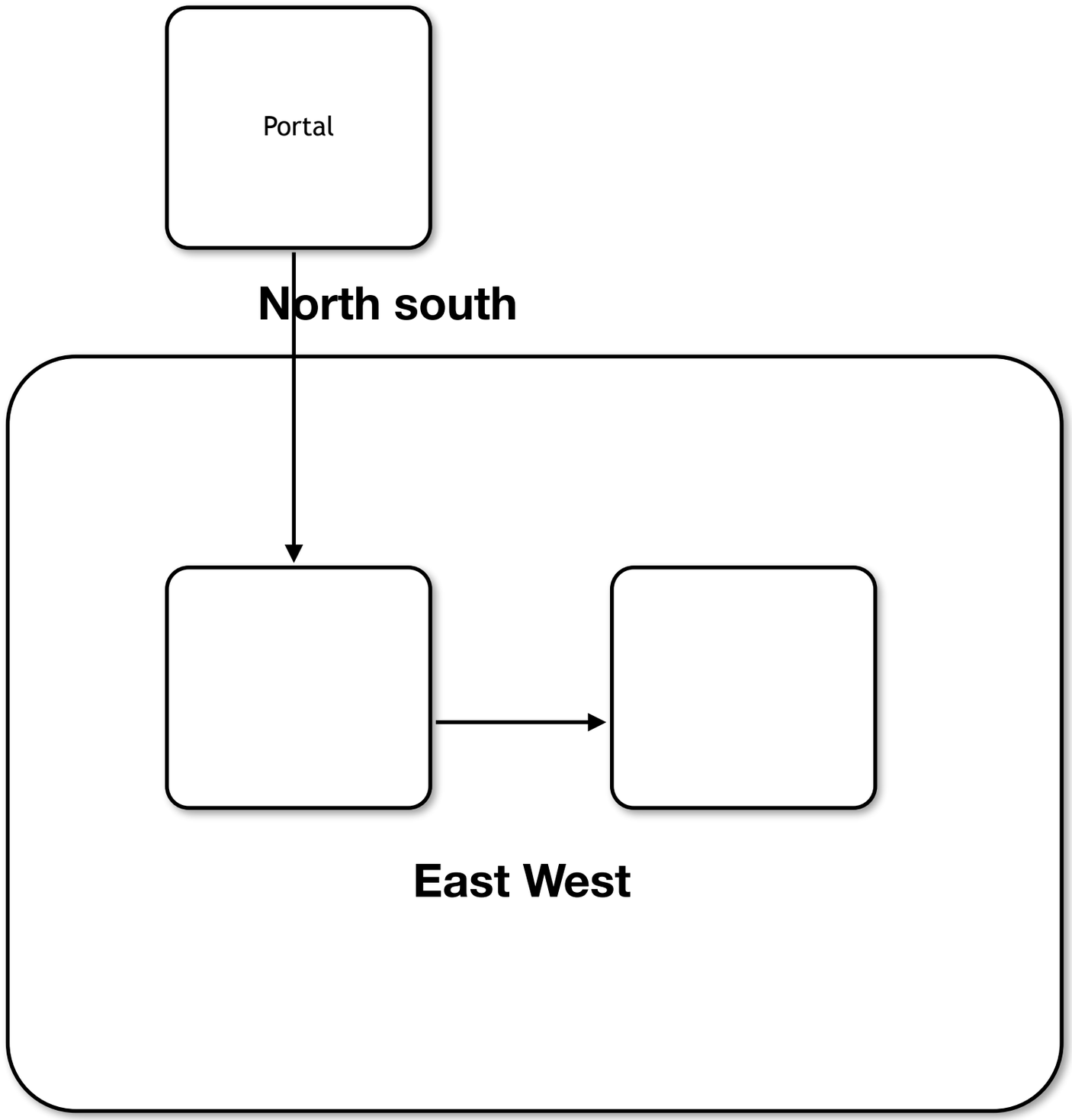
Connected protocol
(REST, WS, GRPC , Thrift)

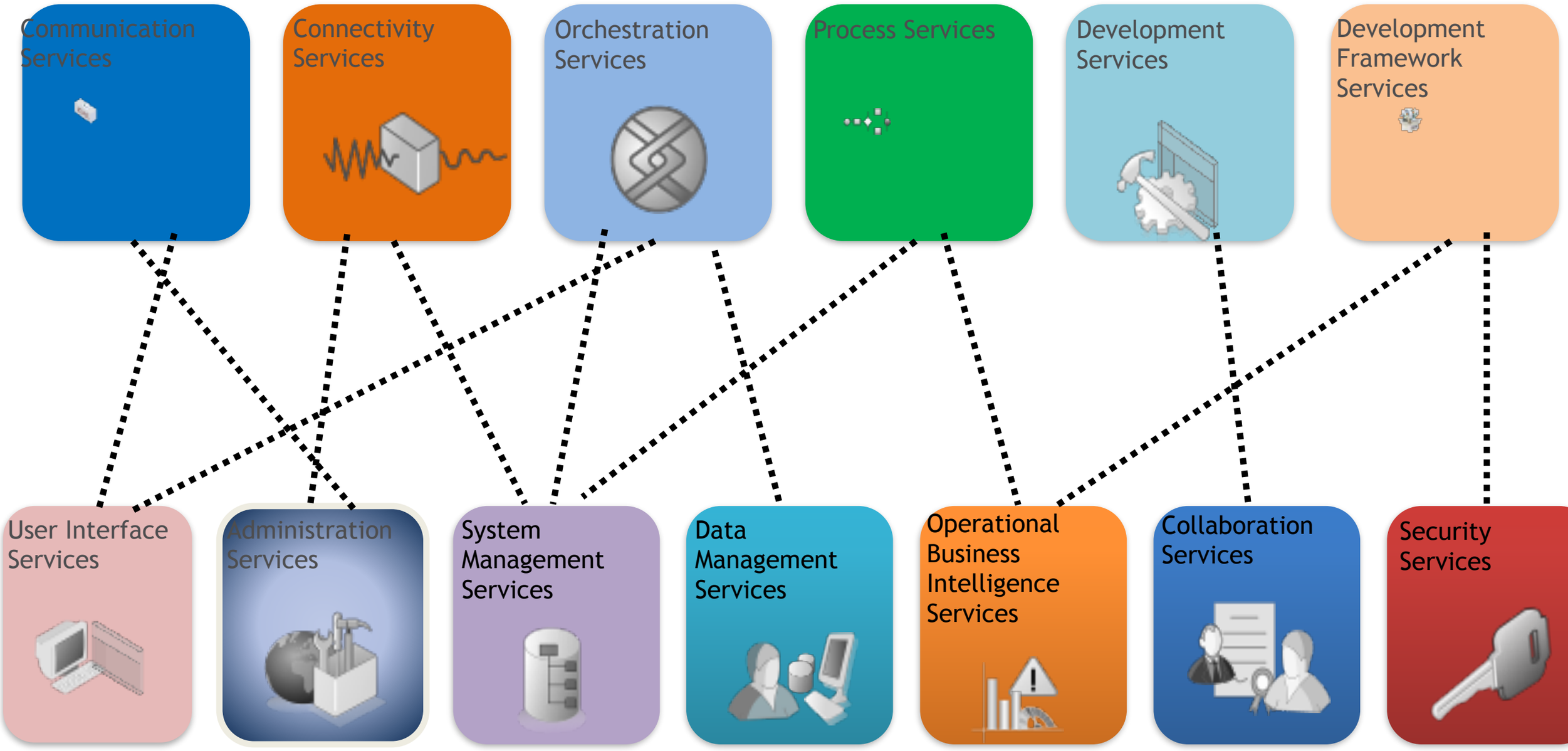
Message protocol
(AMQP, MQTT, …)

	Connected	Msg database (ACID)
Scale	- -	+ +
Coupling	- -	+ +
Reliability	- -	+ +
Developer effort		
Consistency	Immediate	Eventual
Debugging		
Delivery Order	Ordered ++	Unordered * - -
Duplicate	- -	- -
Dev Ops Effort		
Exception Handling	+ +	- - ?



	REST	WSS	GRPC
Browser Support	**Y**	**Y**	**N**
Format	**TEXT (HTTP 1.x)**	**TEXT (HTTP 1.x)**	**BINARY, HTTP2**
Serialization	**JSON**	**JSON / XML / Any**	**Proto Buf**
Performance	**- -**	**+**	**+ +**
Duplex communication	**Pull**	**Pull & Push**	**Pull**
Use case	**North South**	**Streaming & North South**	**East West**


Portal
North south
East West


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

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

Event vs Command

Post **Pre**

OrderCreated **InvoiceCreated** **OrderCanceled**

Communication Services

Connectivity Services

Orchestration Services

Process Services

Development Services

Development Framework Services

CreateOrder

OderCreated **OderCanceled** **OderCreated**

User Interface Services

Administration Services

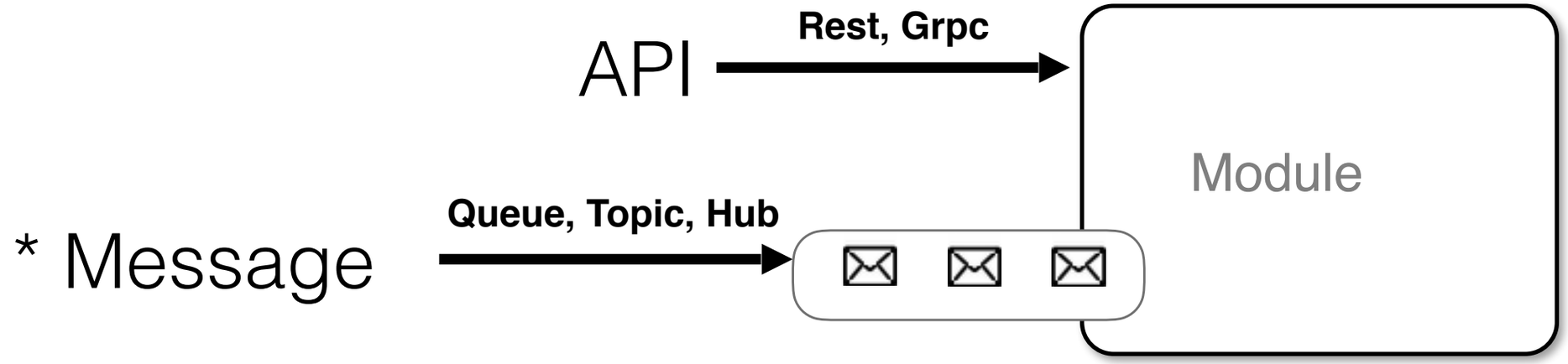
System Management Services

Data Management Services

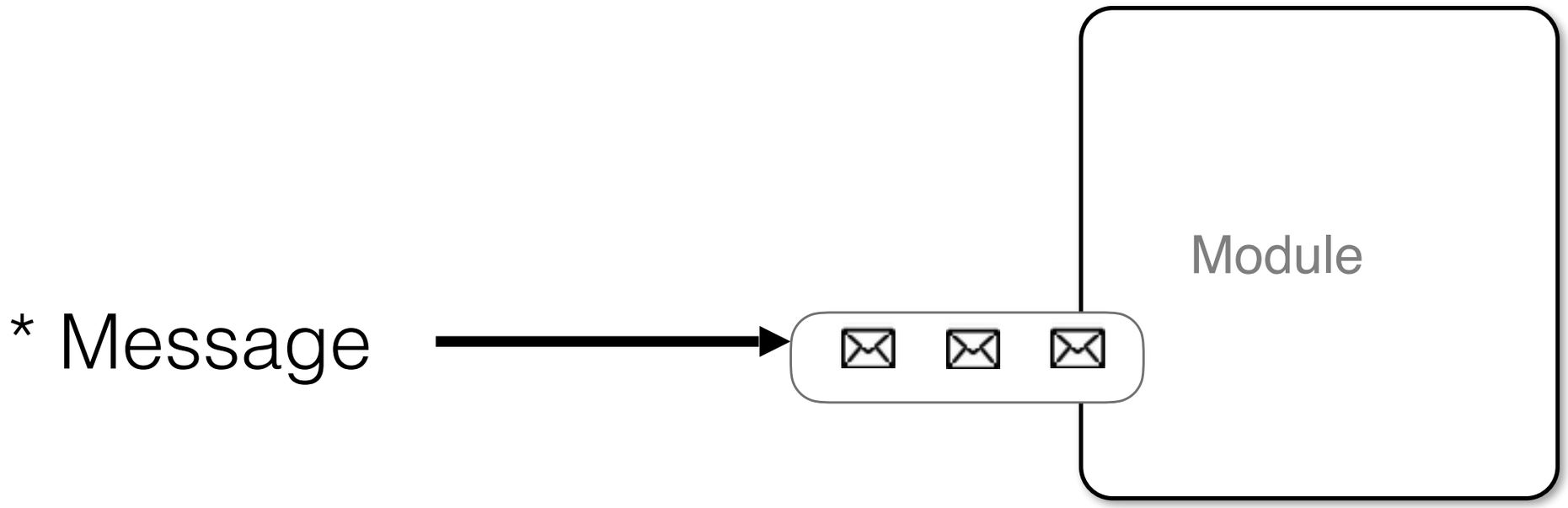
Operational Business Intelligence Services

Collaboration Services

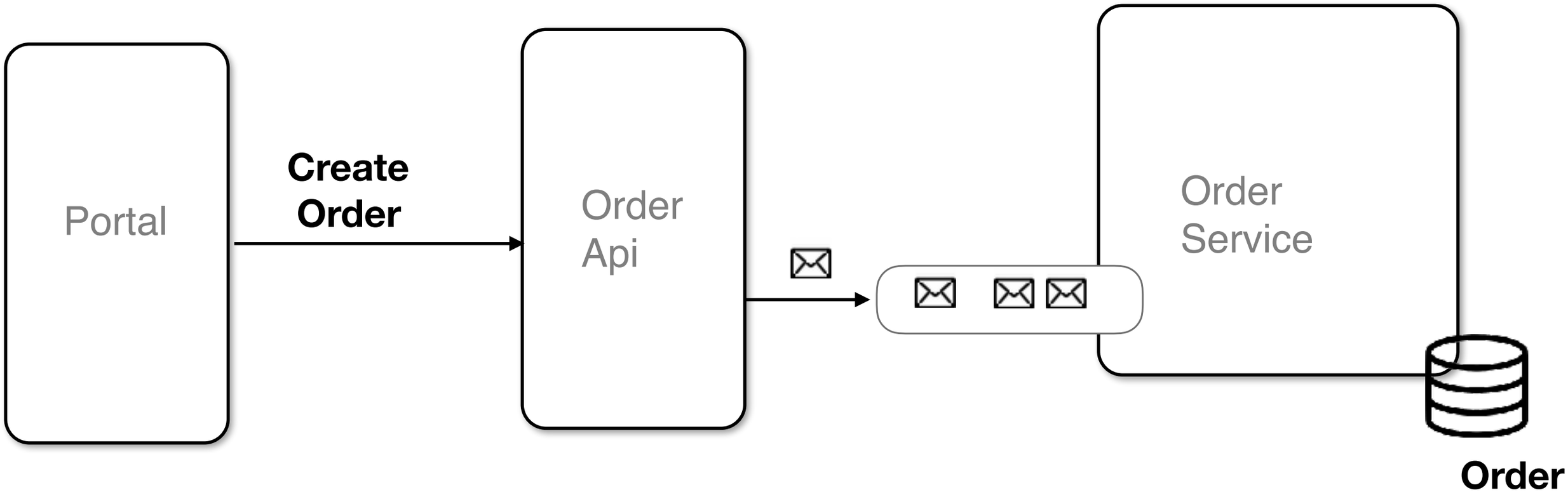
Security Services



	<< API >>	<< Message >>	
Ordering	Ordered (+)	Unordered(-)	
Duplicate	Yes (-)	Yes(-)	Idempotency
Protocol	2 way (+)	One way (-)	
Resilence (recover)	No (-) retry logic	Yes (+)	
Connection	Always connected	Occousionaly connected	
Scalability	- -	+ + +	
Consistency	Immediate (++)	Eventual (- -)	
Load Leveling	- -	+ +	
Low Coupling (maintainability)	- -	+ +	
Distributed Comm Patterns	- -	+ +	SAGA, Materialized View
Internet	Yes	Yes *	
Browser support	Yes	No	
Dev effort	++	- -	
Operational effort	+ +	- -	




```
fun(){
    while(true){
        Msg m = GetMessage();
        .....
        m.ack();
    }
}
```

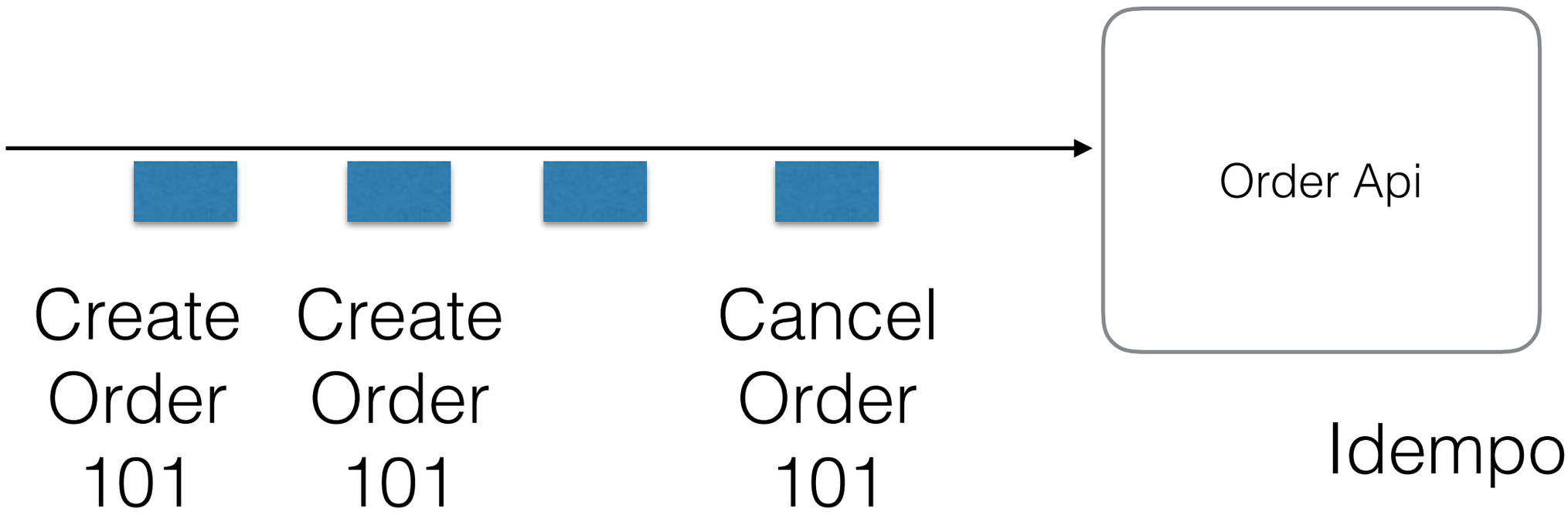

Portal
Create
Order
Order
Api
Order
Service
Order
Order
Service
Account
Service
Order
Account

Client

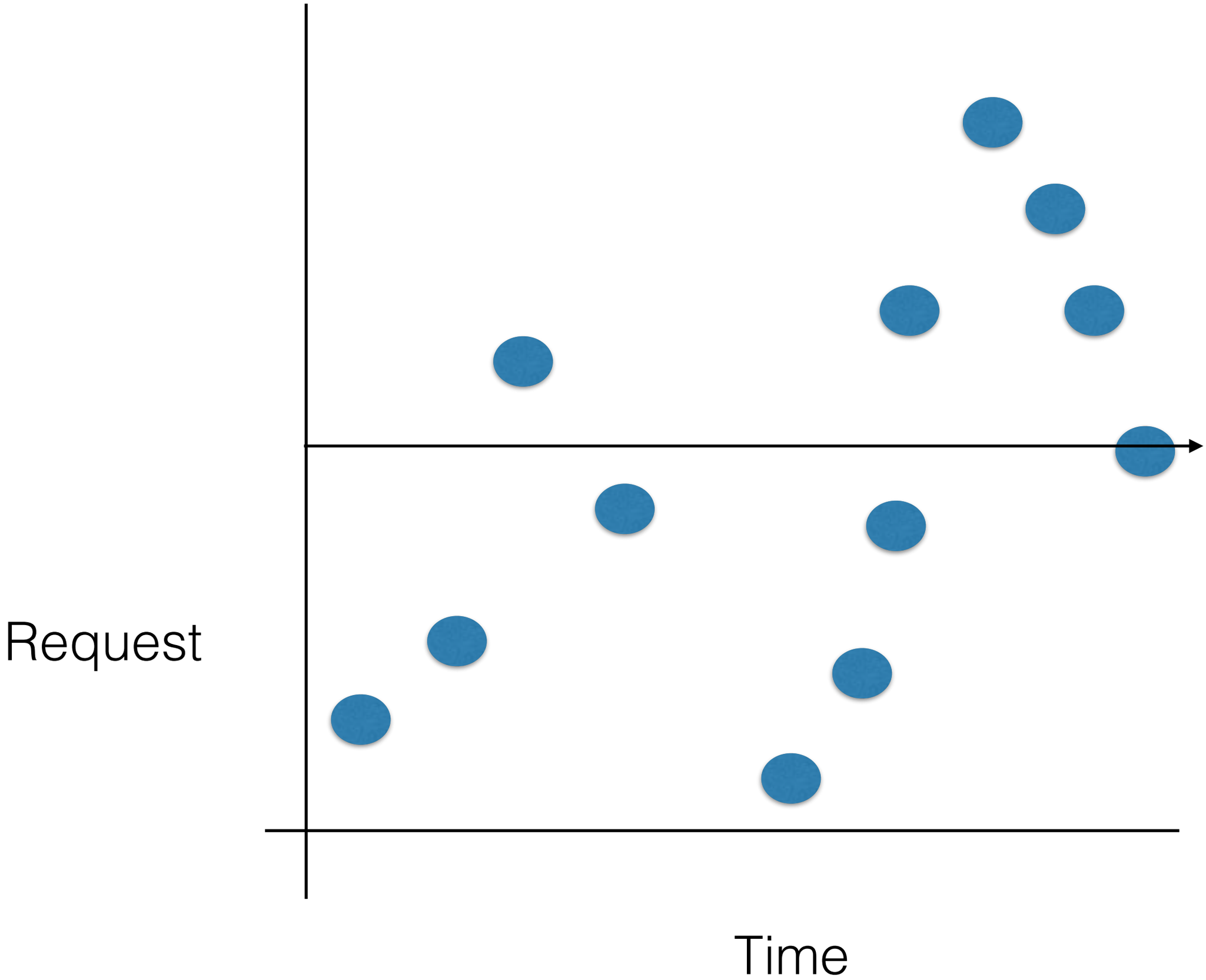
Server

Appp

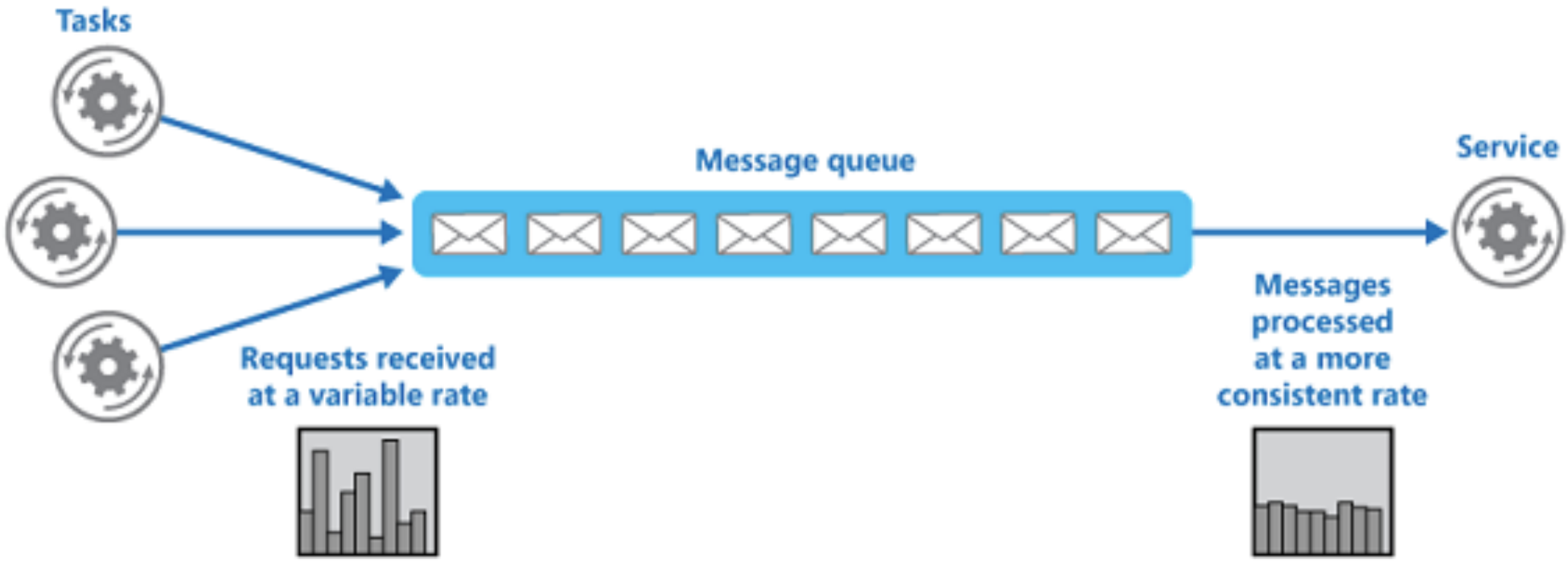
Service



Idempotency

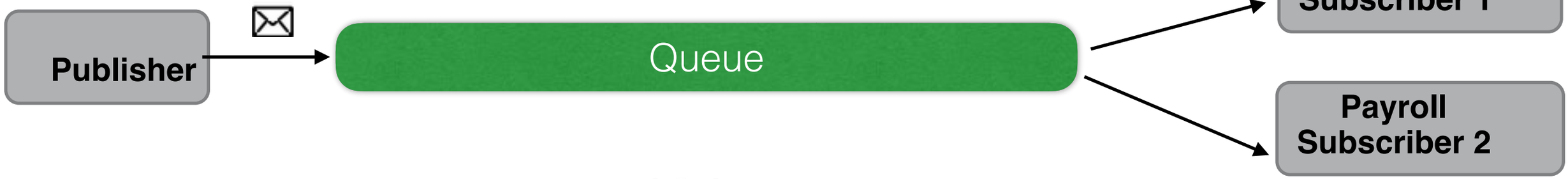

Request
Time

Queue-based Load Levelling

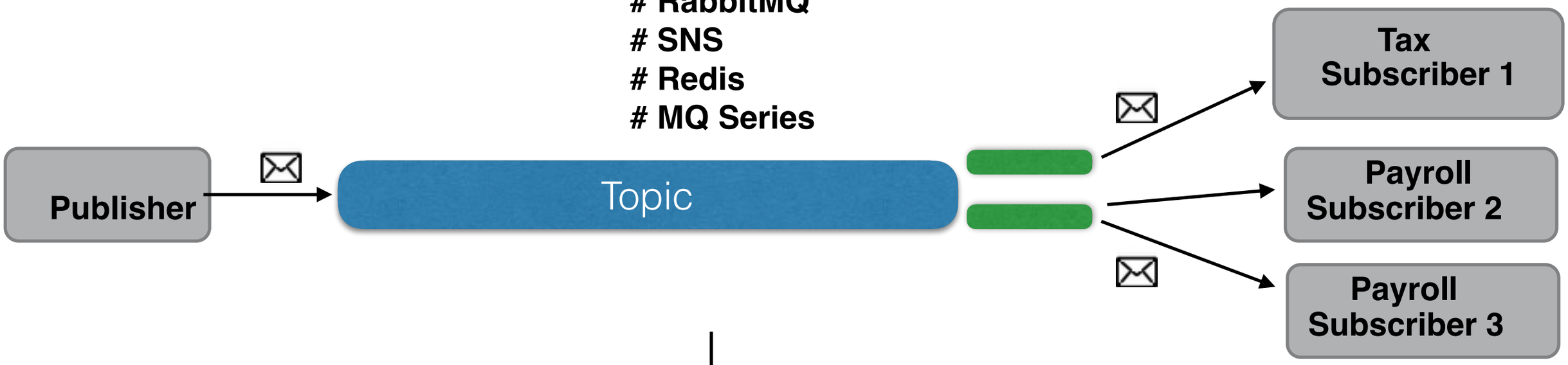


source:msdn

p2p
 # SQS
 # MSMQ
 # …



pubSub
 # RabbitMQ
 # SNS
 # Redis
 # MQ Series



<<Topic>>

Image

Task

Parallelism

Logic 1(data) , Logic 2(data) , Logic 3(data)

OCR(data) , Feature(data) ,Masking(data)

<<queue>>

Event1
Event 2
Event 3

Data

Parallelism

Logic(data1) , Logic(data2) , Logic(data3)

Anonyimization (Event1) ,
Anonyimization(Event2) ,
Anonyimization(Event3)

	Kafka	Pulsar	RabbitMQ (Mirrored)
Peak Throughput (MB/s)	605 MB/s	305 MB/s	38 MB/s
p99 Latency (ms)	5 ms (200 MB/s load)	25 ms (200 MB/s load)	1 ms* (reduced 30 MB/s load)
Number of Programming Languages Supported	17	6	22
Stack Overflow Questions	21,233	134	11,430
Pub/sub	Yes	Yes	Yes
Message routing	Medium	Medium	High
Queueing	Low	Medium	High
Event Streaming	High	Medium	No
Mission-critical	High	Low	High
Slack Community Size	23,057	2,332	7,492
Meetups	486	1	1
Message replay, time travel	+++	+++	-
Exactly-once processing	+++	+	-
consumption	Pull	Push	Push
Permanent storage	Yes	Partial	No

	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	

Table 1

	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	
Potential data loss	Yes	Yes	
Community and vendor support	Good	Good	Good
Building an event store system (used as a store)	No	Yes	No
Ordering	not guaranteed	Guaranteed	
Replay events	No	Yes	No
Transactions	No	Yes	No

API VS WEBHOOKS

API

An API stands for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.

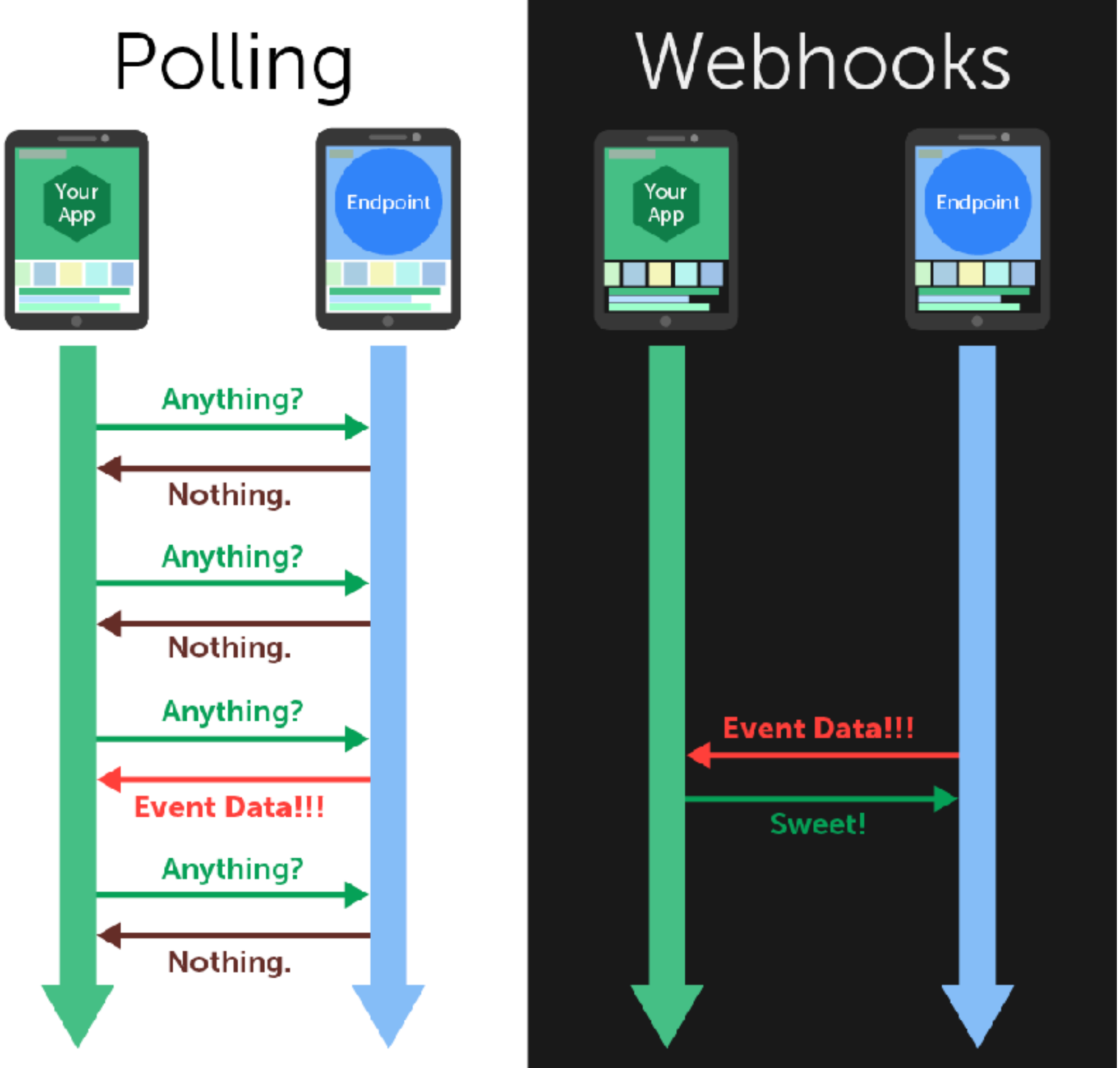
APIs are request-based, meaning that they operate when requests come from 3rd party apps.

To use a real-world analogy, APIs would be likened to you repeatedly calling a retailer to ask if they've stocked up on a brand of shoes you like.

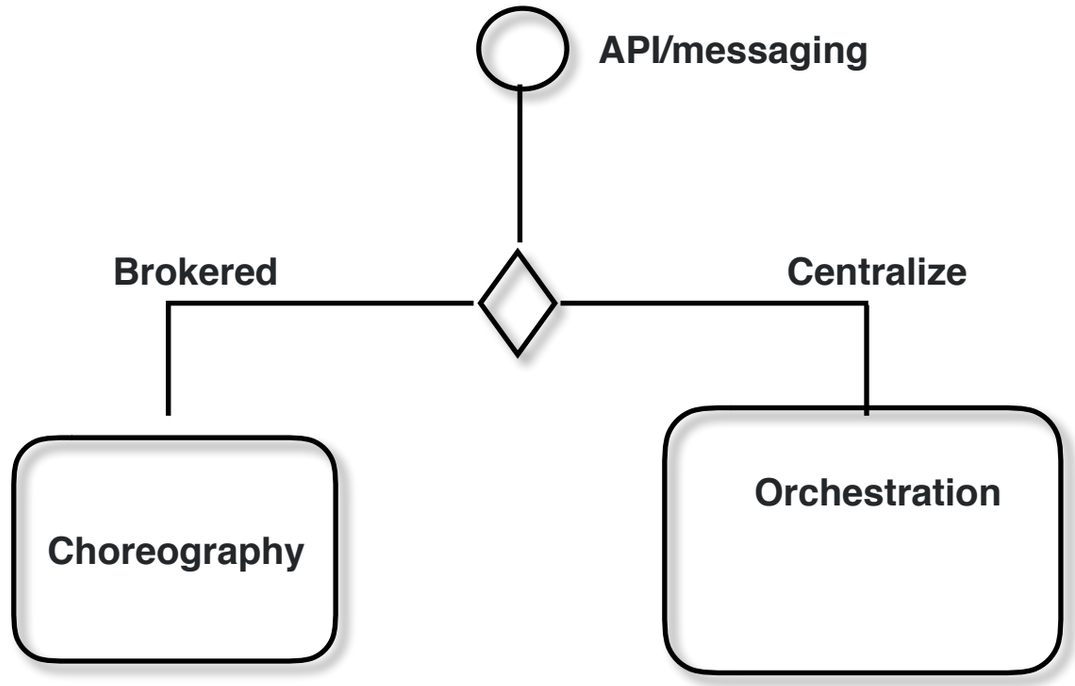
Webhook is also called reverse API, web callback, or an HTTP push API. It delivers data as an event happens or almost immediately.

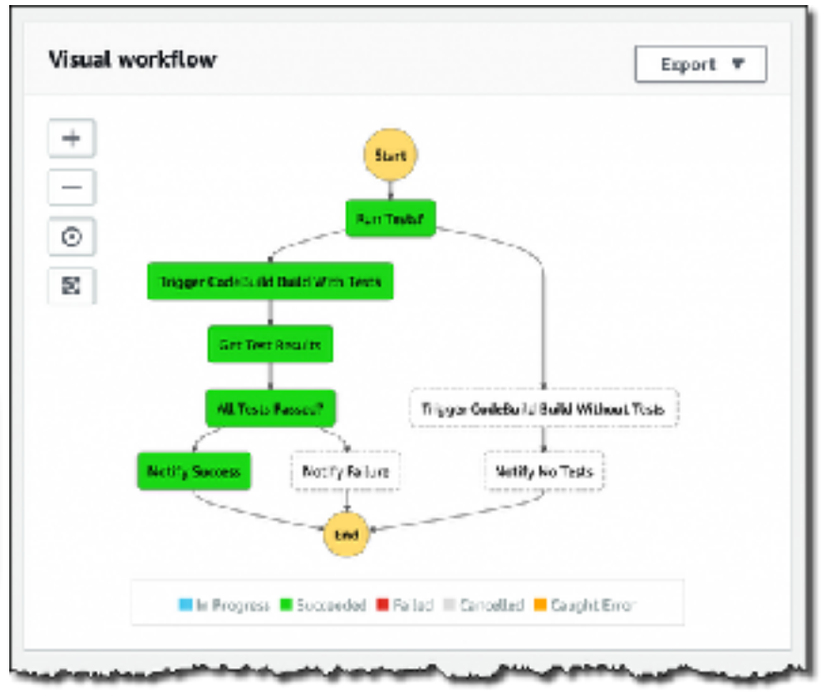
Webhooks are event-based, meaning that they will run when a specific event occurs in the source app.

Webhooks would then be like asking the retailer to call you whenever they have the shoes in stock, which frees up time on both sides.

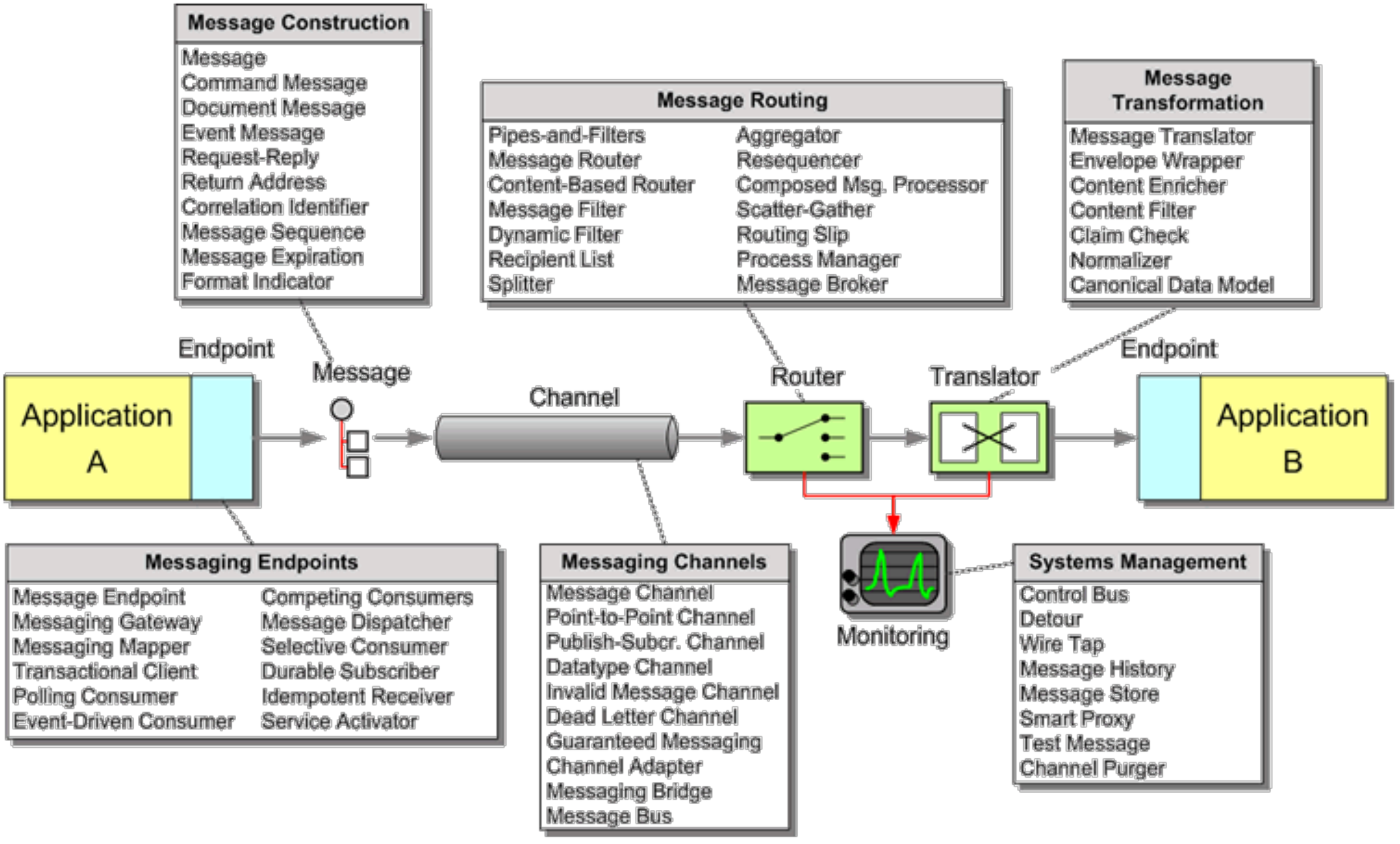

Polling
Your App
Endpoint
Anything?
Nothing.
Anything?
Nothing.
Anything?
Event Data!!!
Anything?
Nothing.
Webhooks
Your App
Endpoint
Event Data!!!
Sweet!

Choose Communication Patterns

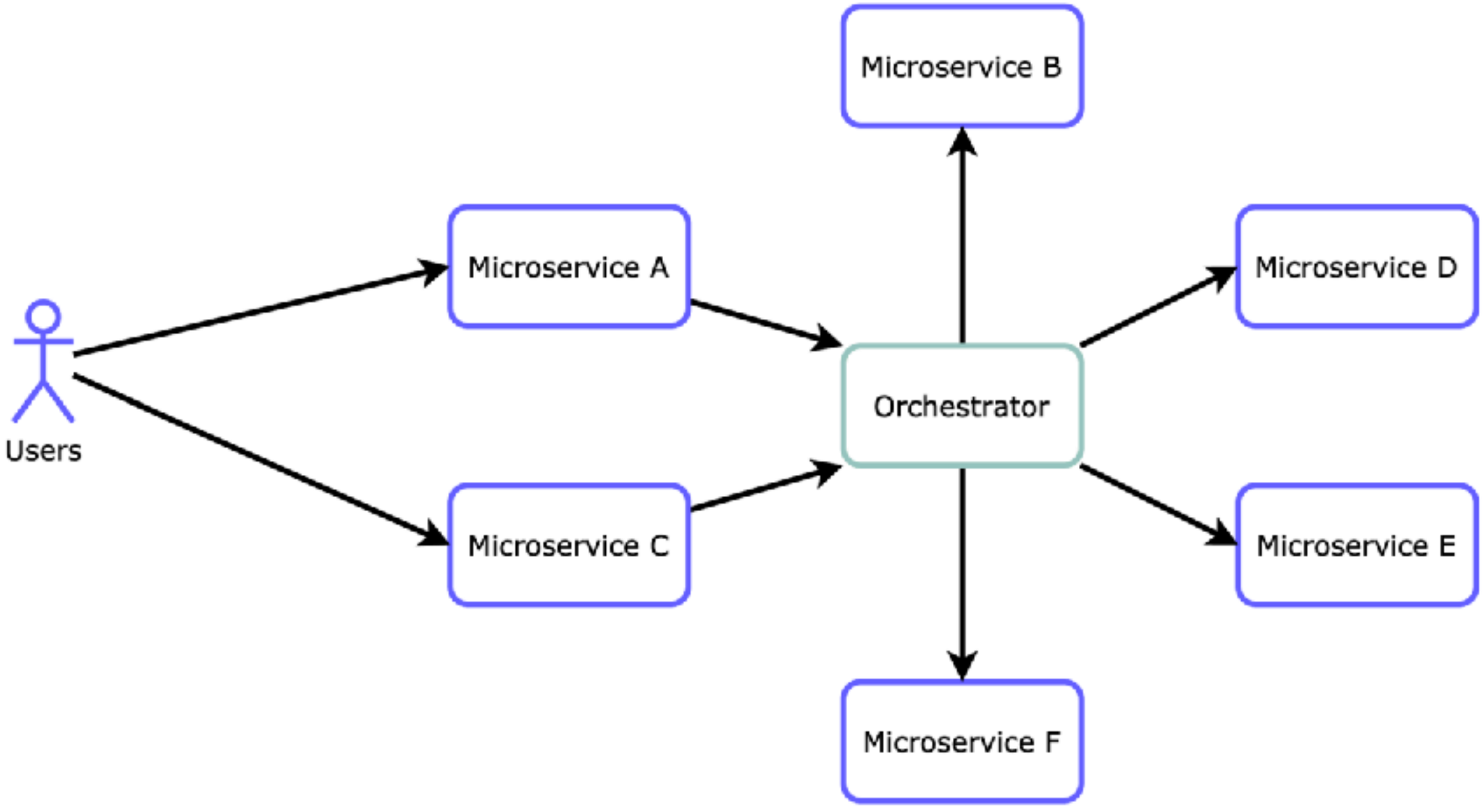


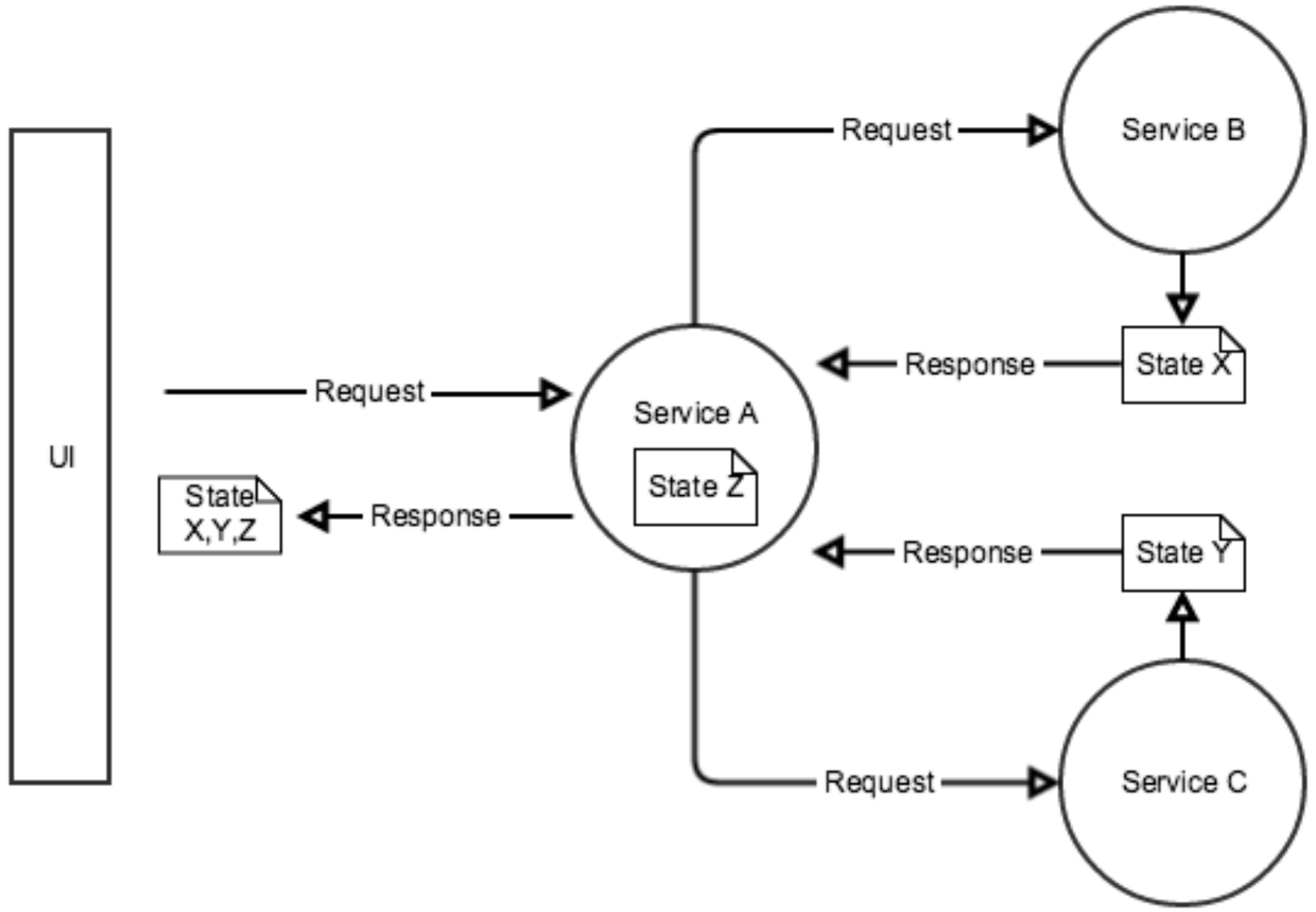


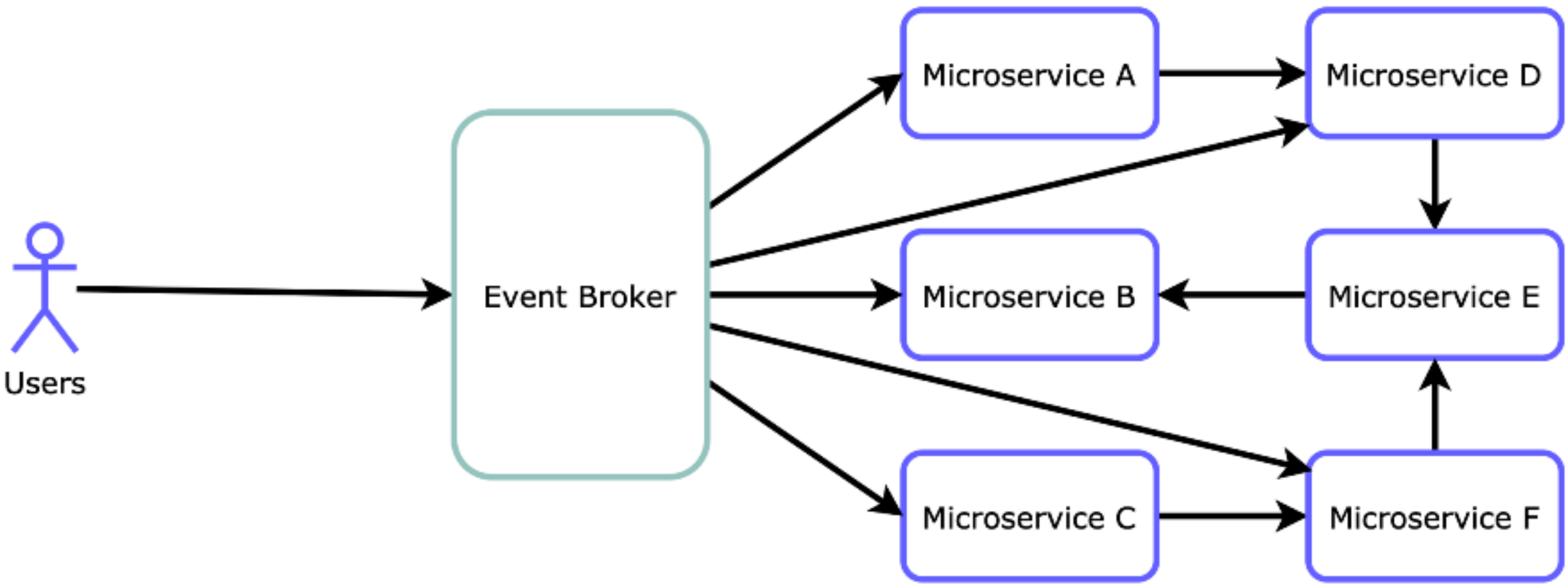
 # AWS Step
 # Azure Logic App
 # Biztalk
 # Webmethods
 # Mule
 # ServiceNow
#



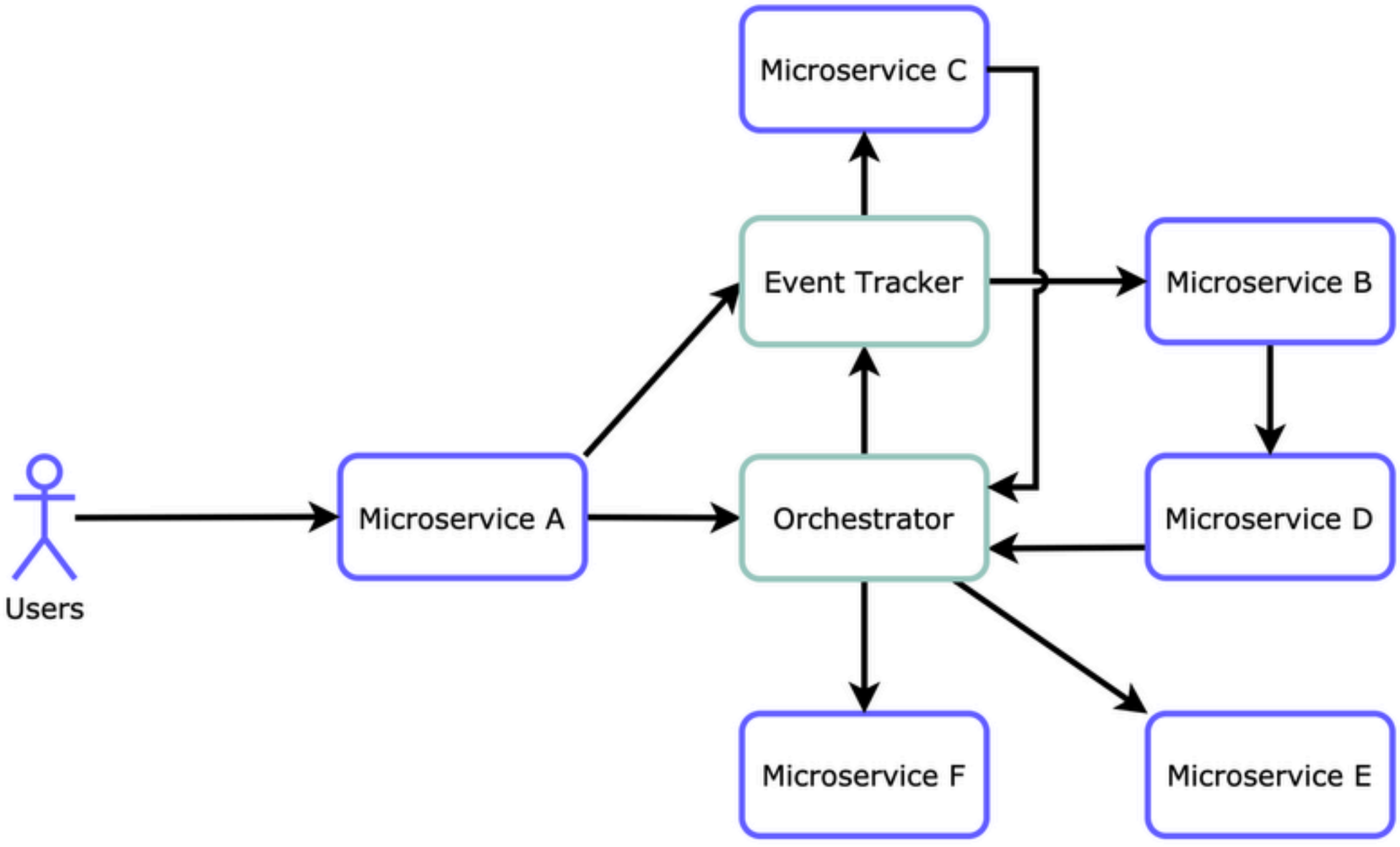
https://www.enterpriseintegrationpatterns.com/patterns/messaging/

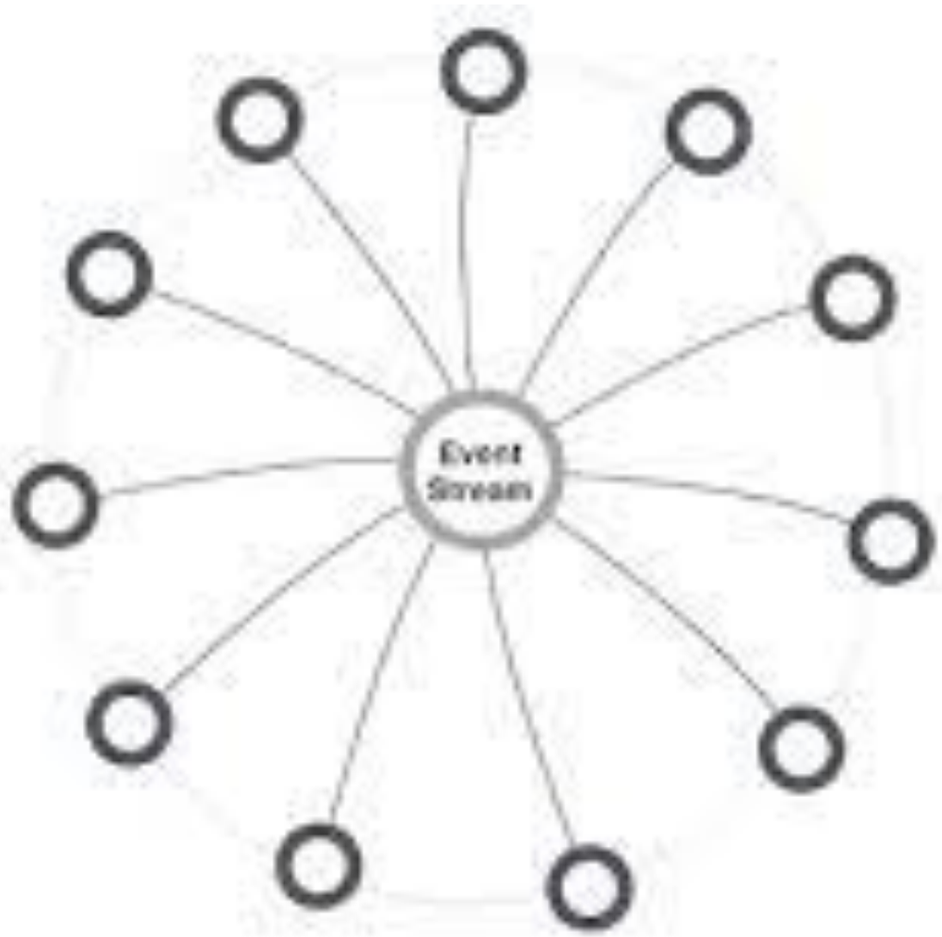

Microservice B
Microservice A
Microservice D
Users
Orchestrator
Microservice C
Microservice E
Microservice F


UI
Request
Service A
State Z
Request
Service B
Response
State X
State X,Y,Z
Response
Response
State Y
Request
Service C

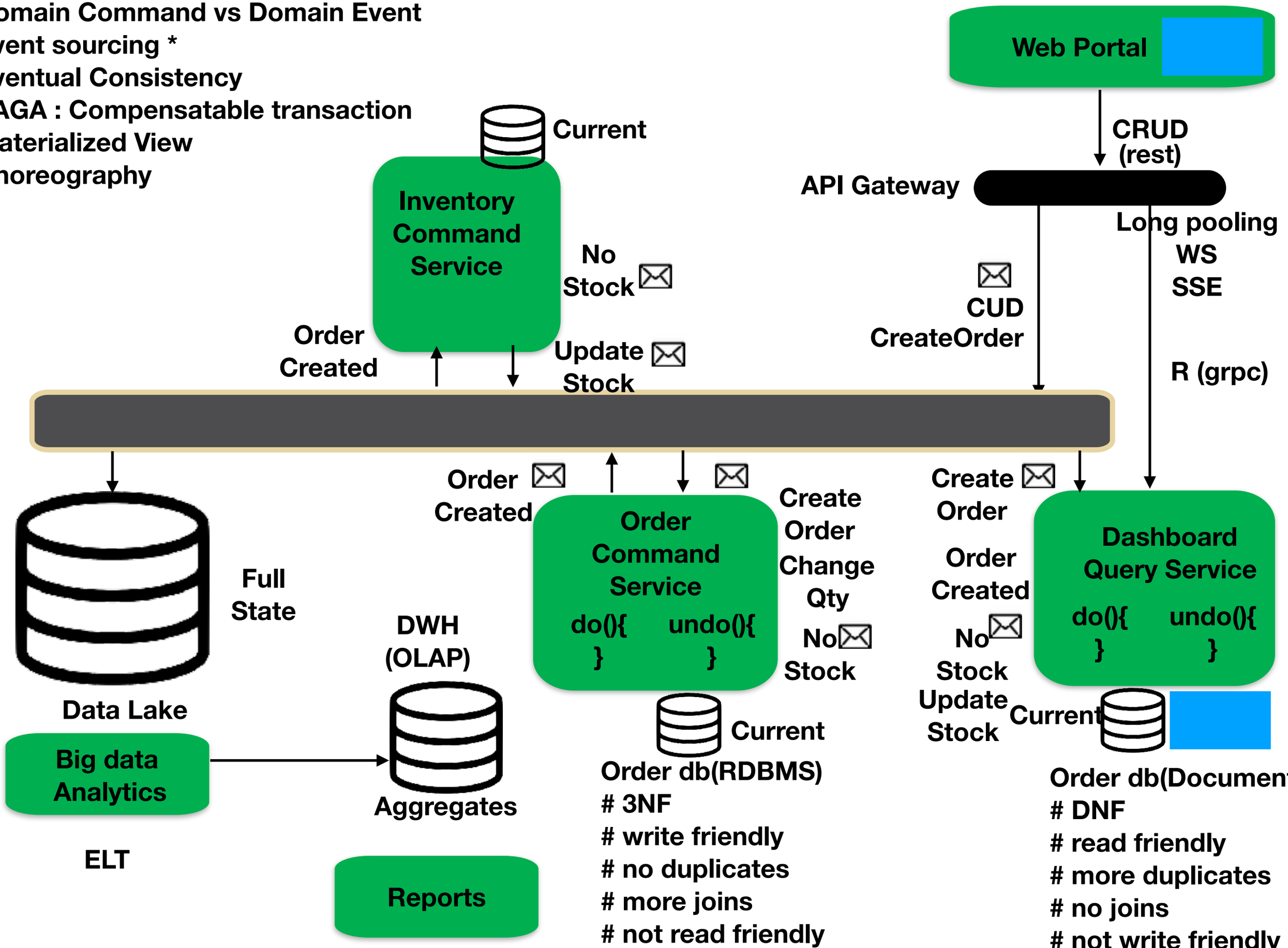

Microservice A
Microservice D
Users
Event Broker
Microservice B
Microservice E
Microservice C
Microservice F

Hybrid

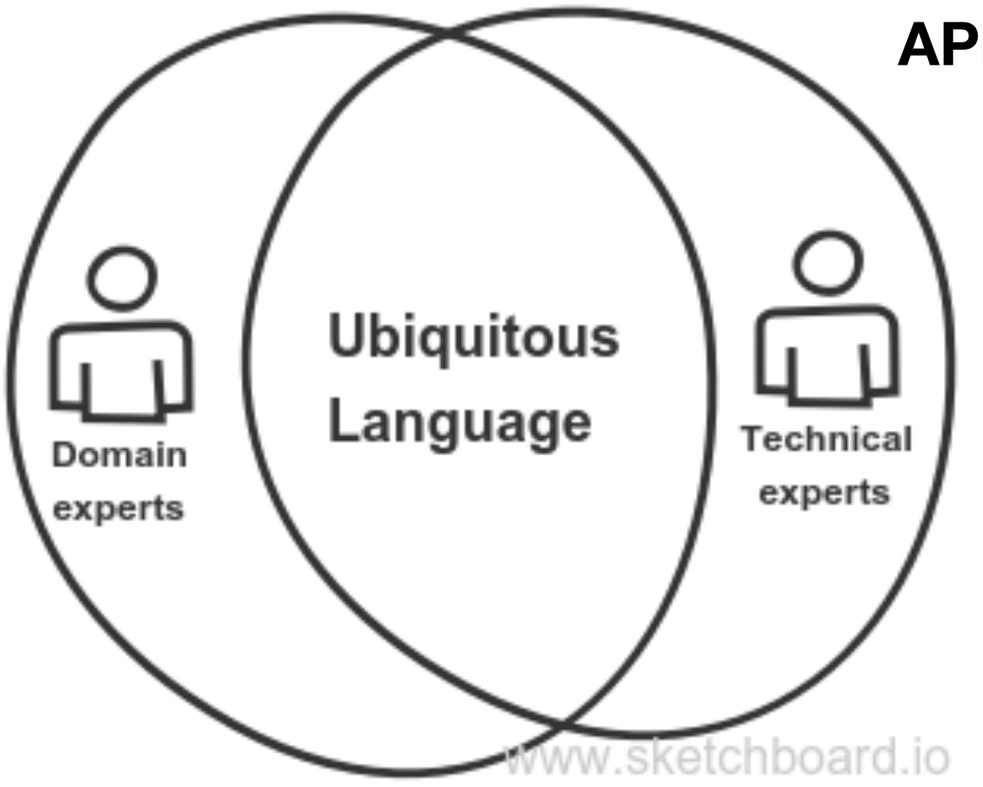

Microservice C
Event Tracker
Microservice B
Users
Microservice A
Orchestrator
Microservice D
Microservice F
Microservice E


Event
Stream

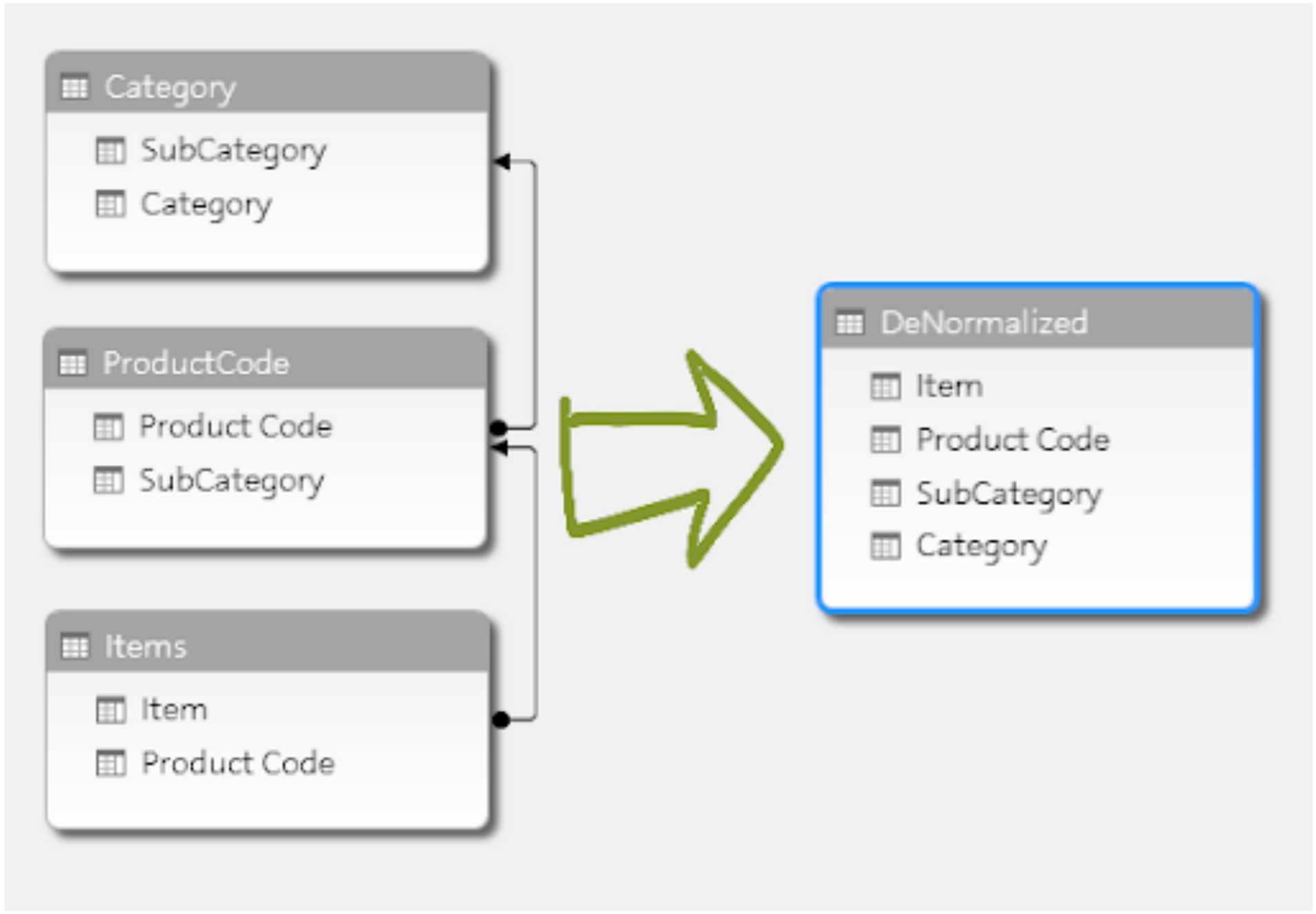
CQRS (Command Query Responsibility)
Domain Command vs Domain Event
Event sourcing *
Eventual Consistency
SAGA : Compensatable transaction
Materialized View
Choreography



DDD



Domain Events


Category
SubCategory
Category
ProductCode
Product Code
SubCategory
Items
Item
Product Code
DeNormalized
Item
Product Code
SubCategory
Category

EDA
SAGA
Event sourcing
Eventual Consistency

Portal

CRUD (rest + json)

API

Account Command Services

Stream Analytics

CUD

CreateOrder

R (grpc)

OrderCreated

NoSufficient Stock

Order Command Services

CreateOrder

OrderCreated

NoSufficient Stock

Query Services

Data lake
full state

DWH
Snapshot state

RDBMS
write friendly
Current state
* NF

Materialized View

Document / RDBMS
read friendly
Current state
* DNF

Batch (Big Data) Analytics

DWH Analytics

Files

Databases (Cloud / On Premises)

Cloud Sources

Audio/ Video/Documents

Data Lake

WebApps, Sensors, IoT

Social Feeds

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

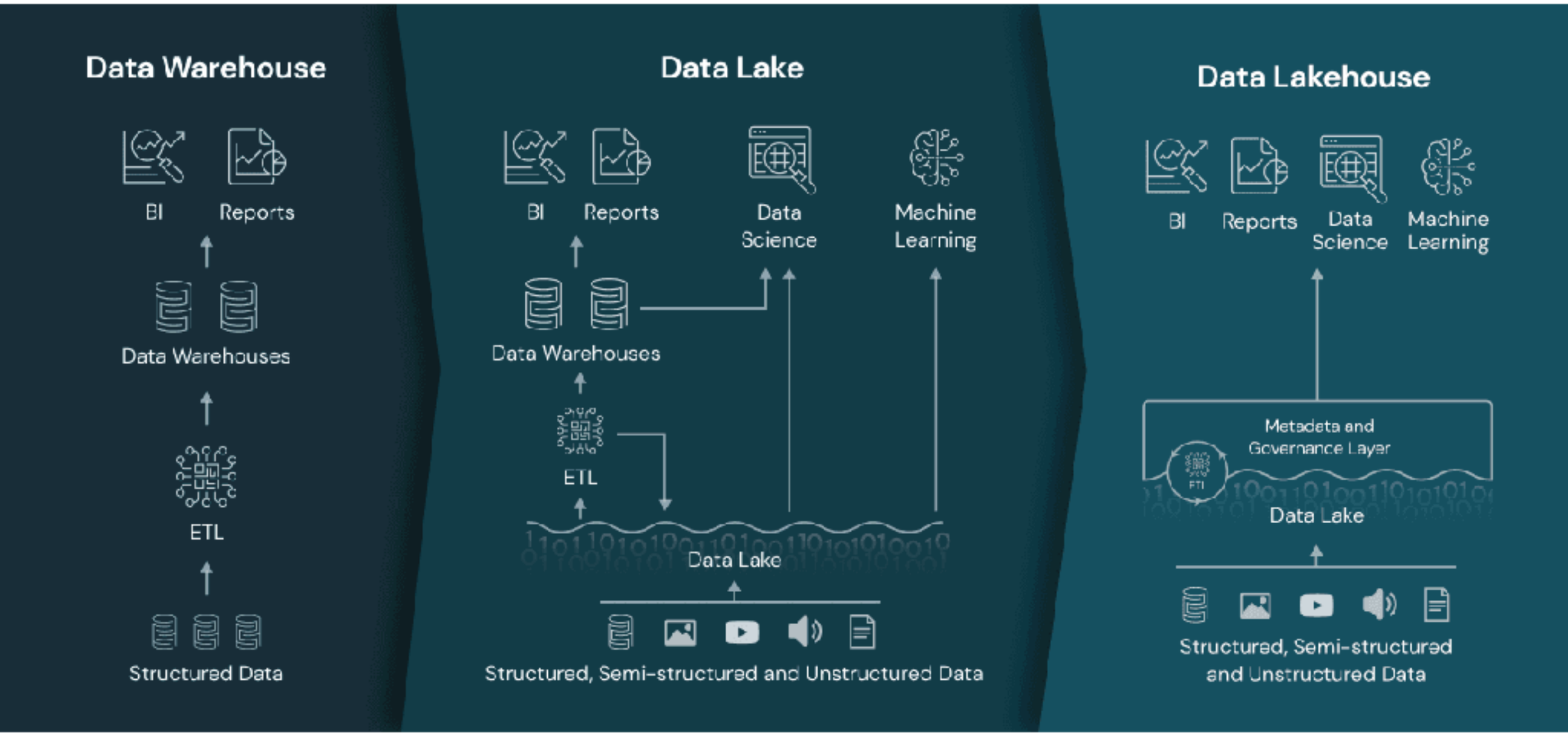
Insight/Analytics Tier

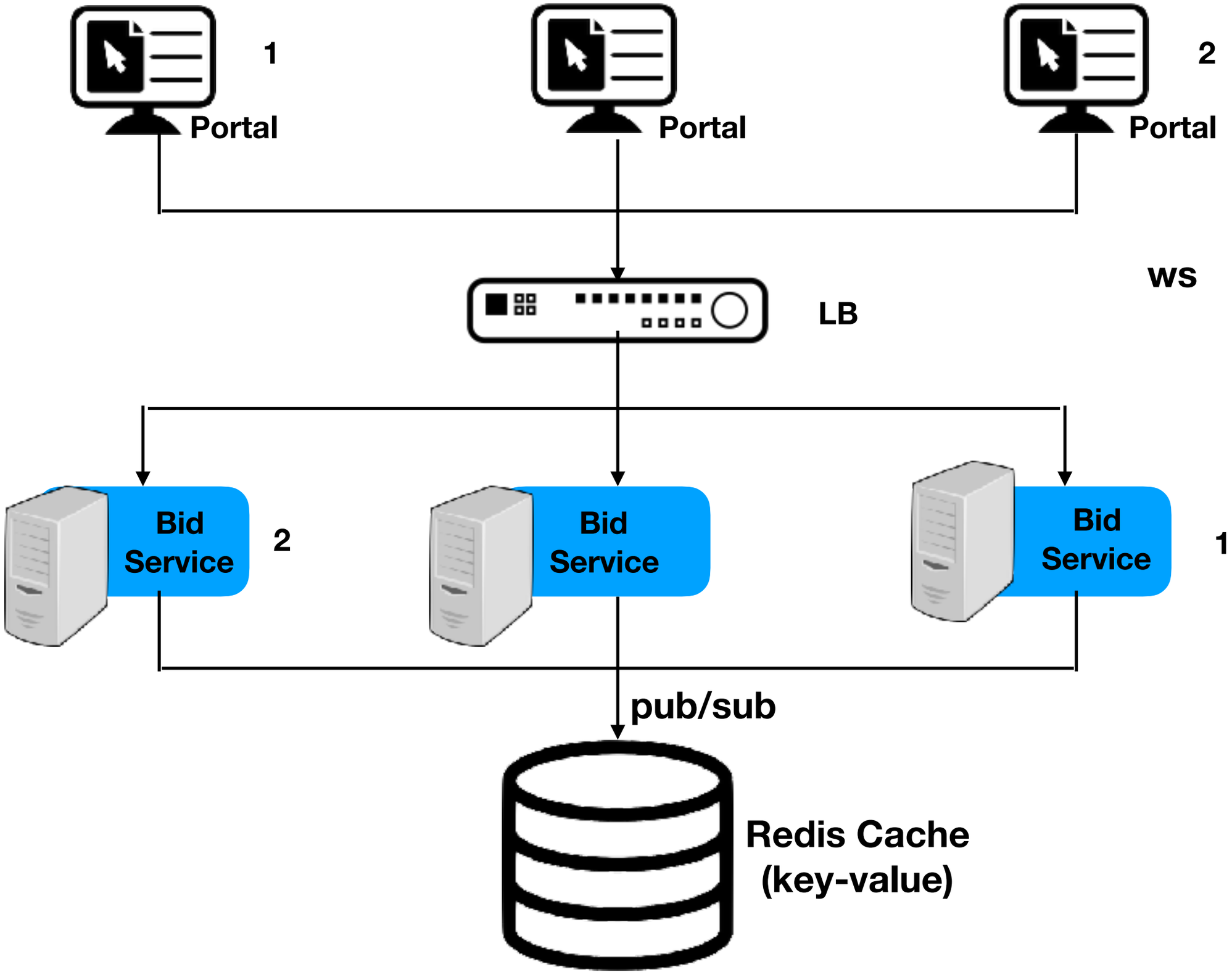
BI Tools

Applications

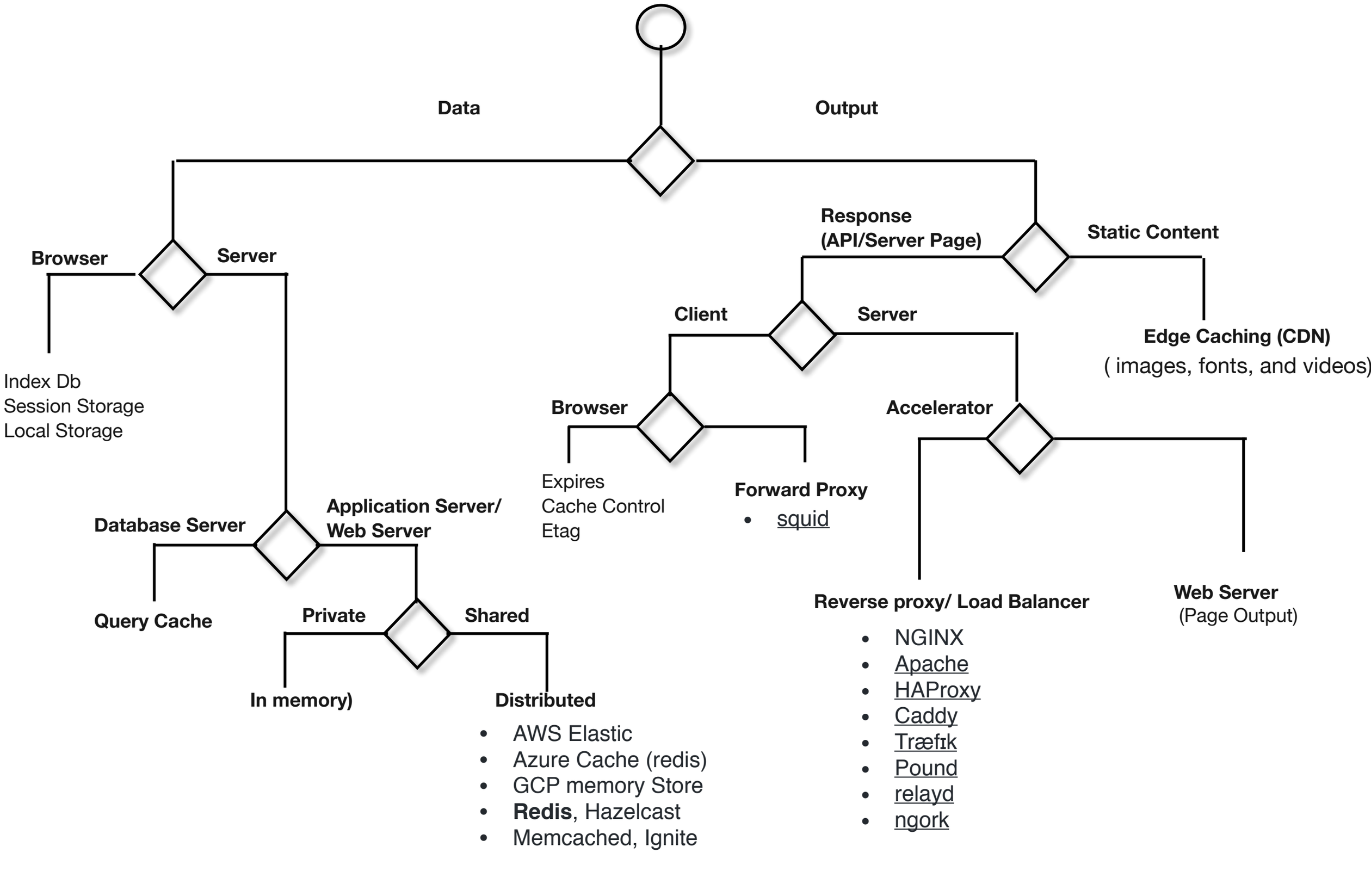
Automations

QuickSight
Power BI


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


1
Portal
Portal
2
Portal
WS
LB
Bid Service
2
Bid Service
Bid Service
1
pub/sub
Redis Cache
(key-value)

Choose Cache



nginx
#

Web Server/
Application Server

Browser

APi
Gateway

Load
balancer

Code

Client
App

Forward
Proxy

Server
App

Reverse
proxy

tomcat
IIS
gunicorn
jobs
web sphere
Apache

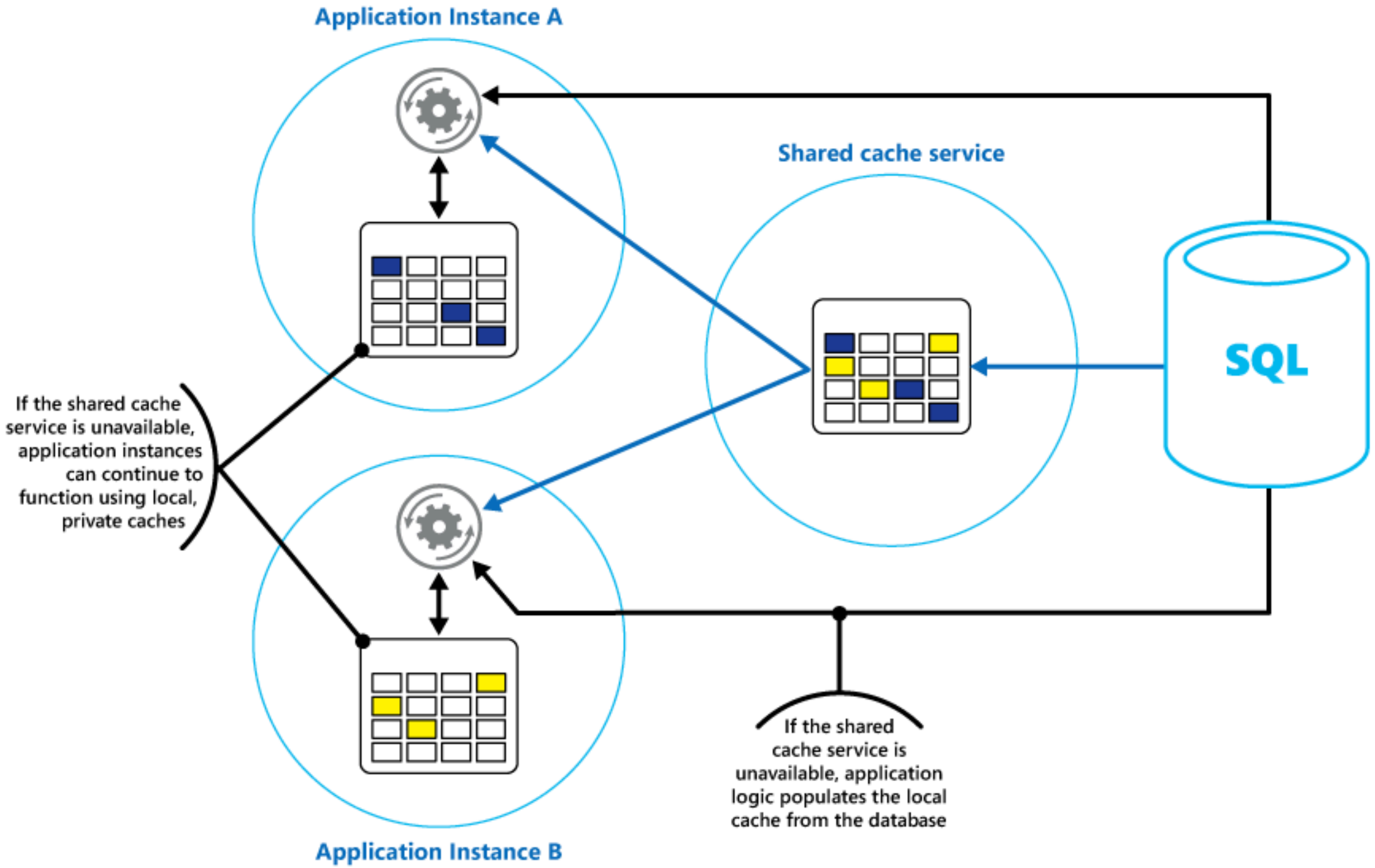
Web acceleration r

SSL/TLS Processing

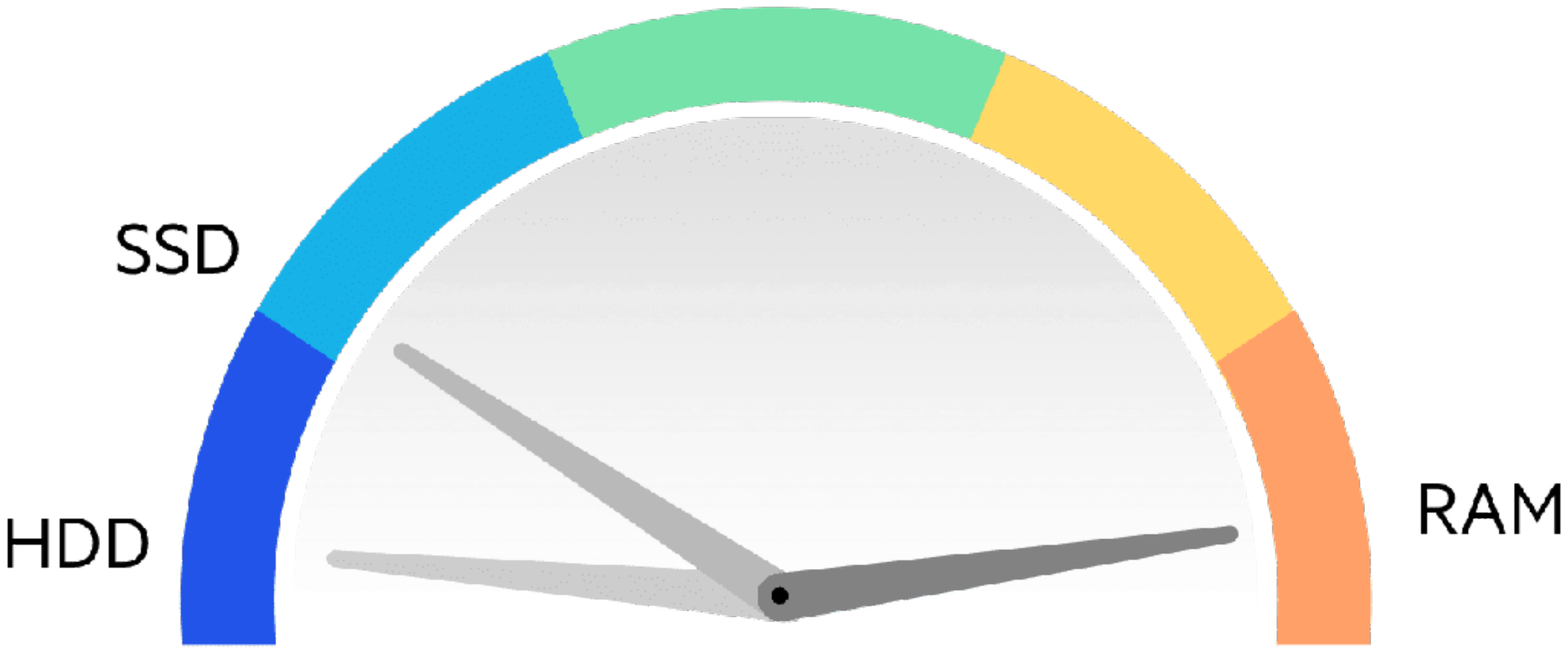
Compression

Caching and Prefetching

Using a local private cache with a shared cache.

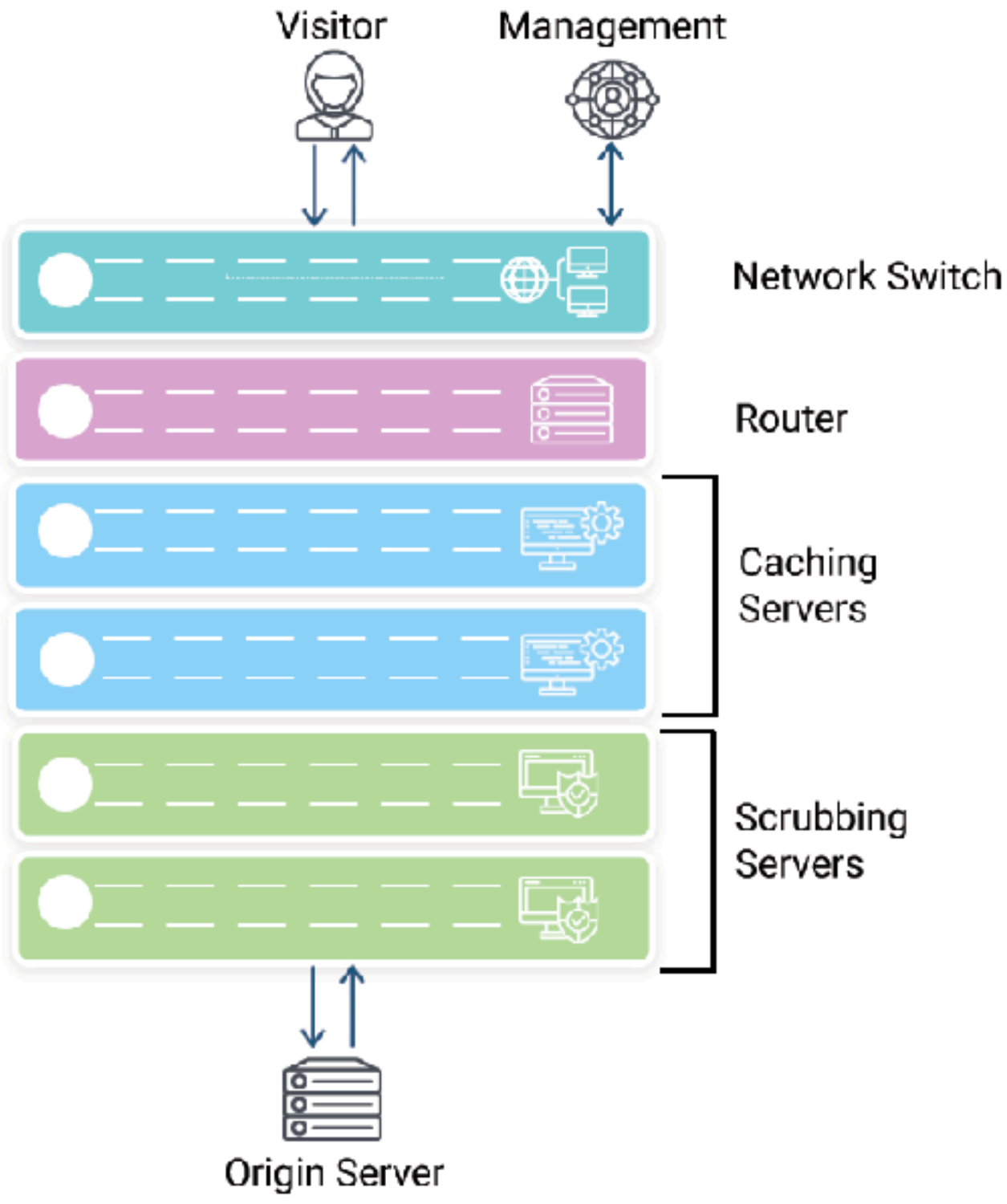


Speed Comparison

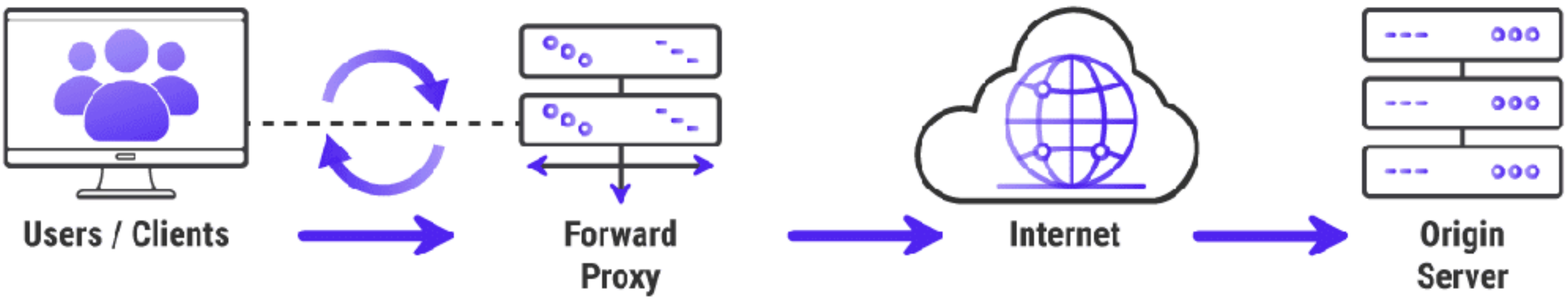


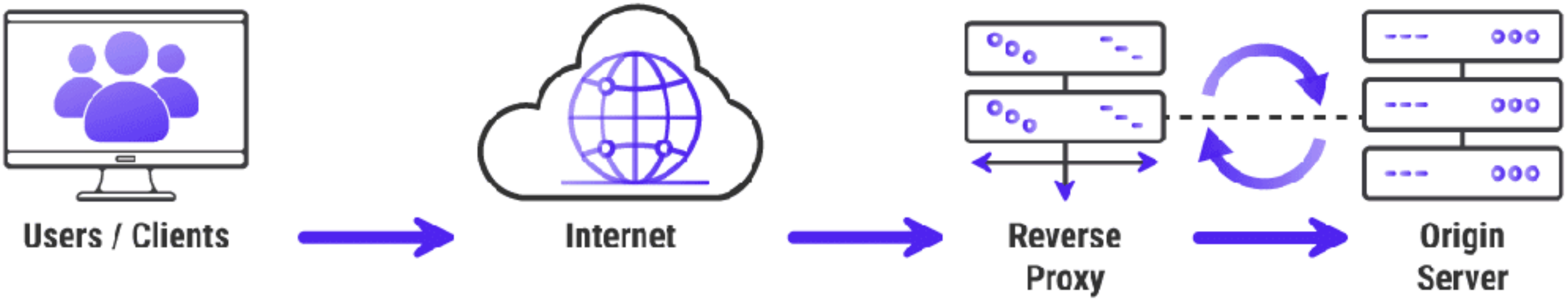
HOW CONTENT DELIVERY NETWORK FUNCTIONS

TOOLBOX



Forward Proxy vs Reverse Proxy





Device registration
Device communication
Protocol translation

Stream Processing

Datalake house

Cloud Gateway

Dataware house

Reports

Sensor

Field Gateway

Event Stream

Data lake

DataMart

Connected vehicles, Fleet management

Connected manufacturing

Facilities management

Batch Processing

Choose Operational Compute

API App/ Web App / Background App

Containers ?

Yes

No

Distributed ?

Yes

No

High Frequency of usage

Simple Use case

Yes

No

Serverless Kubernetes

- Azure Container Instance
- AWS Fargate for Amazon ECS
- Google Cloud Run

7

High Frequency of usage

Yes

No

Serverless Container (PAAS 2.0)

- Azure Functions
- AWS Lambda
- Google Cloud Function

5

Managed Kubernetes (PAAS 1.0)

- Azure kubernetes Service
- Amazon Elastic Kubernetes Service (Amazon EKS)
- Amazon Elastic Container Service
- Google Kubernetes Engine (GKE)

6

Paas ?

Yes

No

High Frequency of usage

No

Yes

Serverless Webserver (PAAS 2.0)

- Azure Functions
- AWS Lambda
- Google Cloud Function

3

Managed Container (PAAS 1.0)

- Azure App Service
- AWS Elastic Beanstalk
- Google App Engine

4

IAAS

- Azure VM
- EC2
- Google Compute Engine

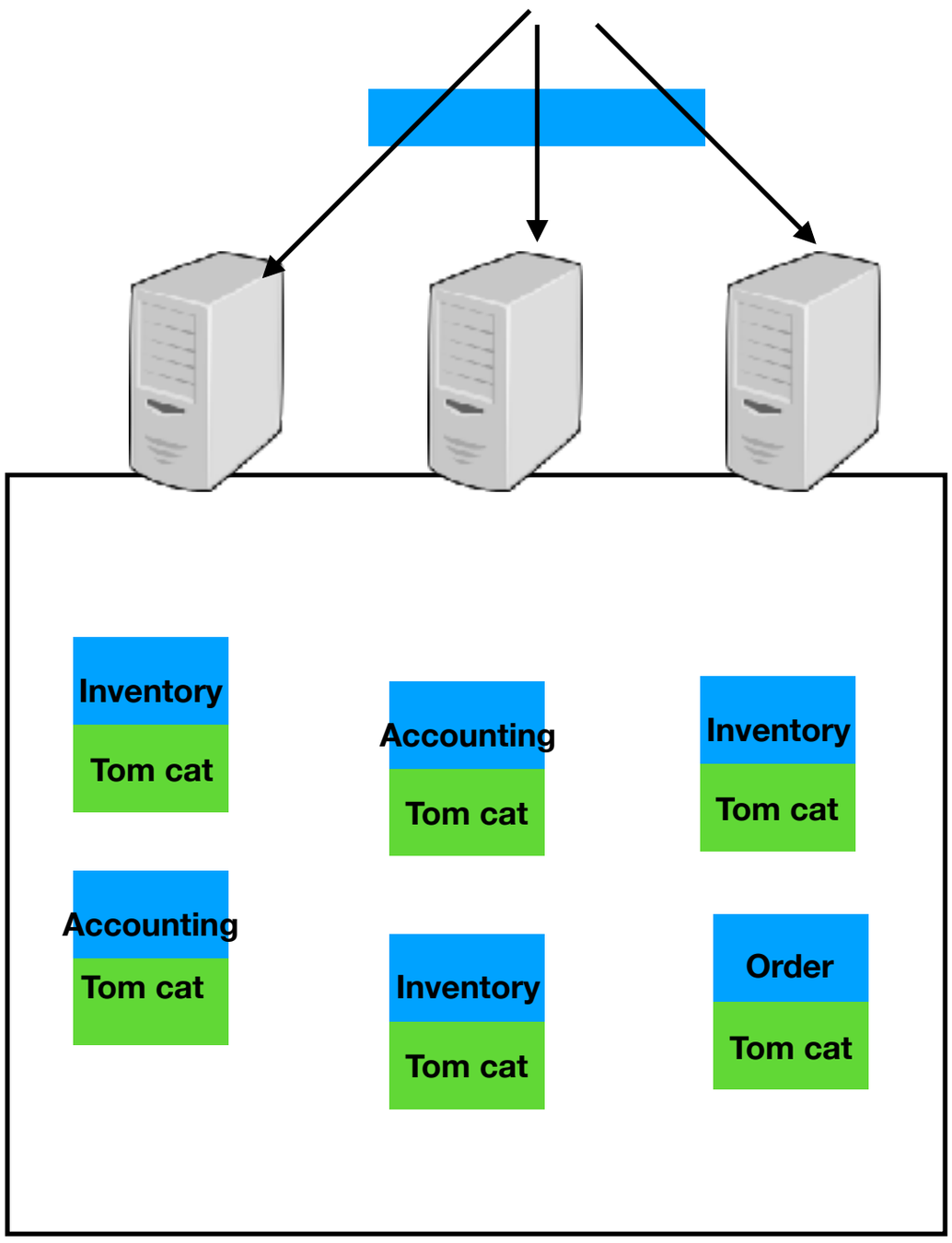
1

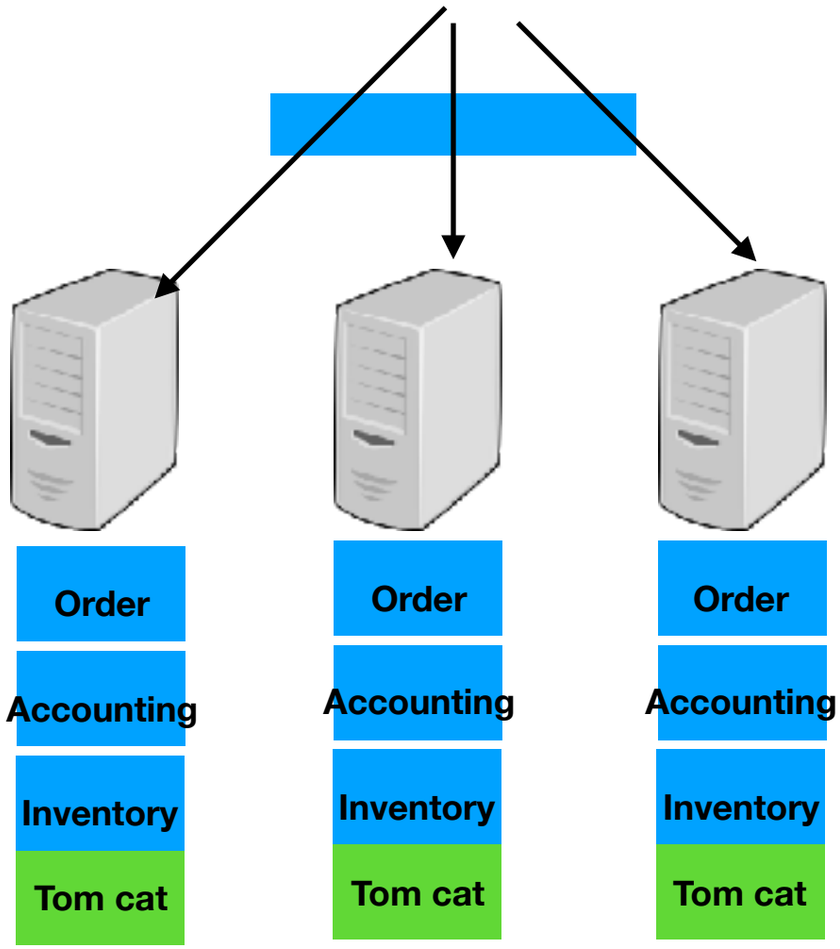
Managed Web Server (PAAS 1.0)

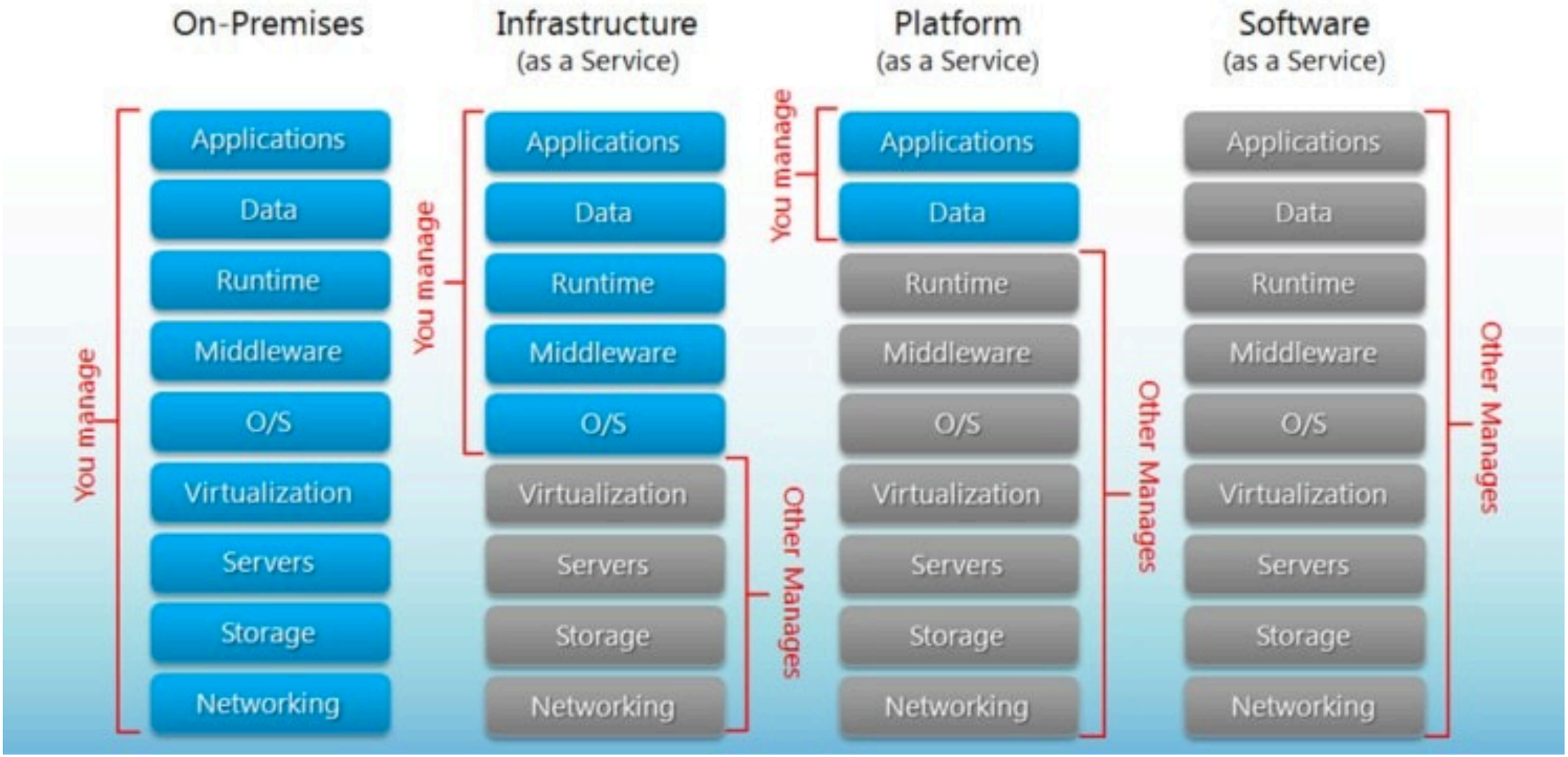
- Azure App Service
- AWS Elastic Beanstalk
- Google App Engine

2


Vm
<<my app>>
Middle wear
Docker
OS
<<my image>>
Vm
<<my app>>
Middle wear
OS


Inventory
Tom cat
Accounting
Tom cat
Inventory
Tom cat
Accounting
Tom cat
Inventory
Tom cat
Order
Tom cat


Order
Accounting
Inventory
Tom cat
Order
Accounting
Inventory
Tom cat
Order
Accounting
Inventory
Tom cat


On-Premises
Infrastructure
(as a Service)
Platform
(as a Service)
Software
(as a Service)
You manage
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Other Manages

Isolated Env.

My app 1

My app 2

Middlewear (tomcat/ Apache/ IIS / Nodejs…)

Os

Cpu
RAM
Disk
I/o

Machine

VM1

VM2

My app 1

My app 3

Middlewear(tomcat/apache/…)

My app 2

Unix Guest Os

Win Guest Os

Virtualisation layer

Host Os

Cpu
RAM
Disk
I/o

Cpu
RAM
Disk
I/o

Cpu
RAM
Disk
I/o

Machine

Container 1

Container 2

Cpu
RAM
Disk
I/o

My app 1

Middlewear

My app 2

Cpu
RAM
Disk
I/o

Containerization layer (Docker)

Cpu
RAM
Disk
I/o

Host Os

Machine

VM1

VM2

Container 1

8080

Container 2

My app 1

My app 2

Middlewear

Middlewear

Cpu
RAM
Disk
I/o

8080

8080

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 (Hyperviser)

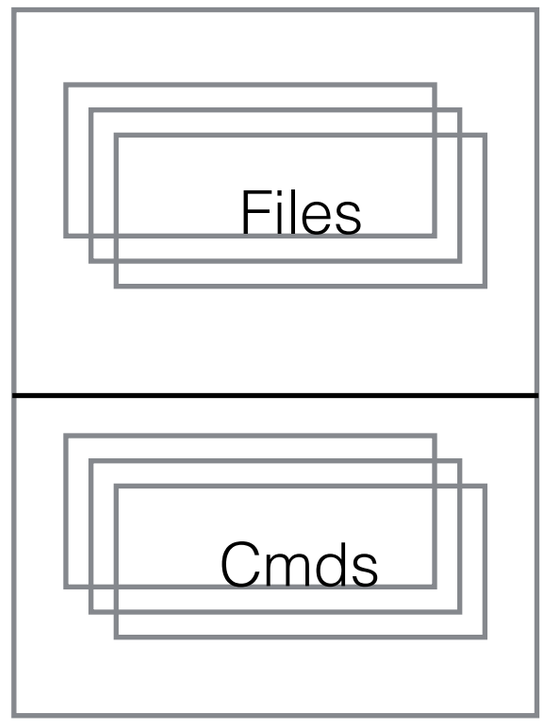
Host Os (azure)

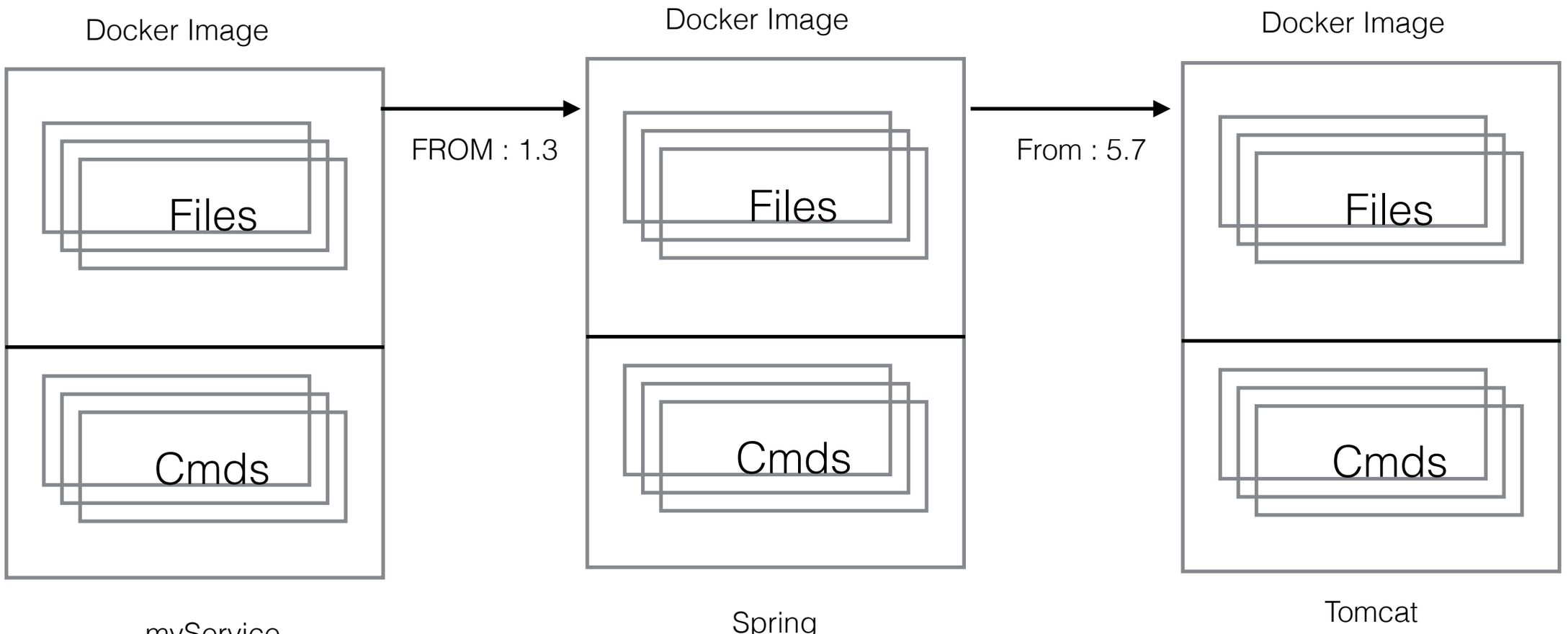
Cpu
RAM
Disk
I/o

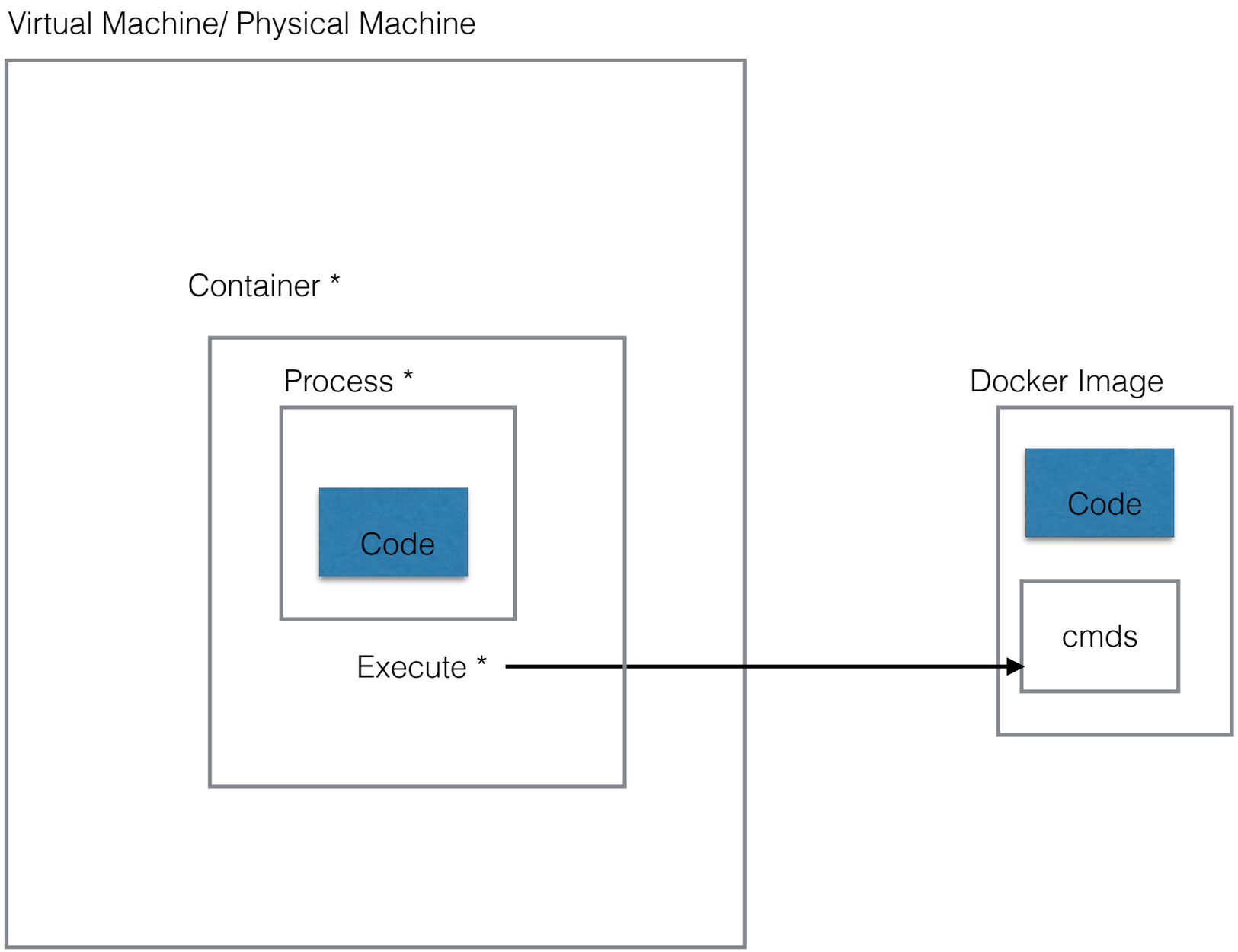
Machine

Reproducable Env.

setup/msi/war/..




Docker Image
Files
Cmds
myService
FROM : 1.3
Docker Image
Files
Cmds
Spring
From : 5.7
Docker Image
Files
Cmds
Tomcat


Virtual Machine/ Physical Machine
Container *
Process *
Code
Execute *
Docker Image
Code
cmds

Image

cmds

Binary Files
(js, pyc, jar, exe, dll, …)

Docker file

Package file

Source File

Project

Build

Push

htsl.docker.com (Repository)

hub.docker.com (Repository)

Image

Python 3.7 Image

My App Image

Pull

VM

Image

Container

Nodejs

copy
Run **cmds** in docker file

CPU
RAM
DISK
I/O

Account Service Image

Inventory Service Image

Order Service Image

VM

Account Service Image

C1

Inventory Service Image

C2

Order Service Image

C3

Db Servers VM

Account

Inventory

Order

Managing Container Life cycle

VM1

VM2

Container 1

Container 2

Container 3

My app 1

My app 2

My app 3

Middlewear

Middlewear

Middlewear

Cpu
RAM
Disk
I/o

Cpu
RAM
Disk
I/o

Containerization layer (Docker)

..

Unix Guest Os

…

Cpu
RAM
Disk
I/o

Cpu
RAM
Disk
I/o

Virtualisation layer

Host Os (azure)

Machine

Kubernetes (slave)

Kubernetes (Slave)

Kubernetes (Master)

VM1

Container 1

Container 2

My app 1

Middlewear

Cpu
RAM
Disk
I/o

Kubernetes (slave)

Containerization layer (Docker)

Unix Guest Os

VM2

Container 1

Container 2

My app 1

Middlewear

Cpu
RAM
Disk
I/o

Kubernetes (Slave)

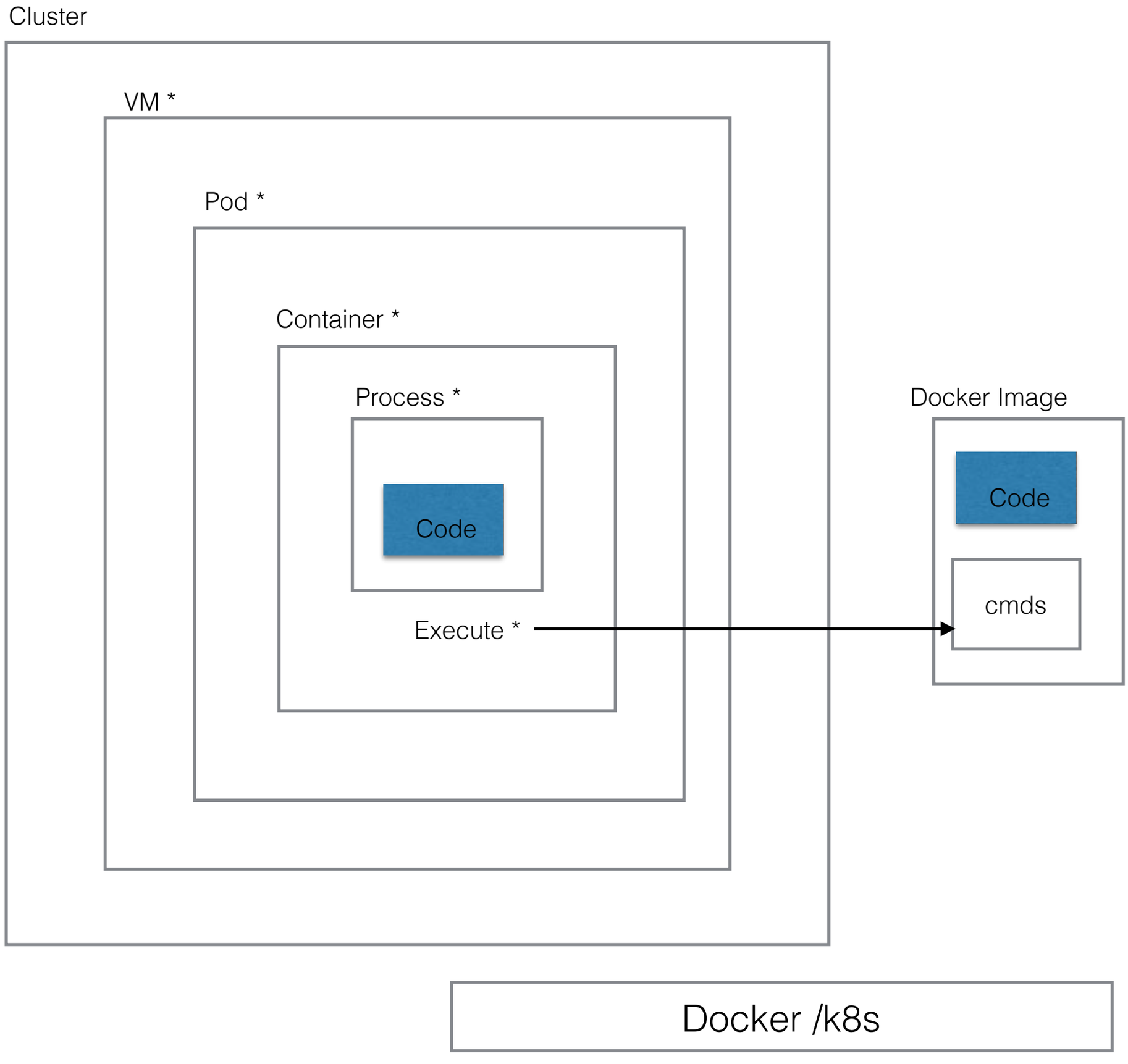
Containerization layer (Docker)

Unix Guest Os

VM3

Kubernetes (Master)

Cmds


Cluster
VM *
Pod *
Container *
Process *
Code
Execute *
Docker Image
Code
cmds
Docker /k8s

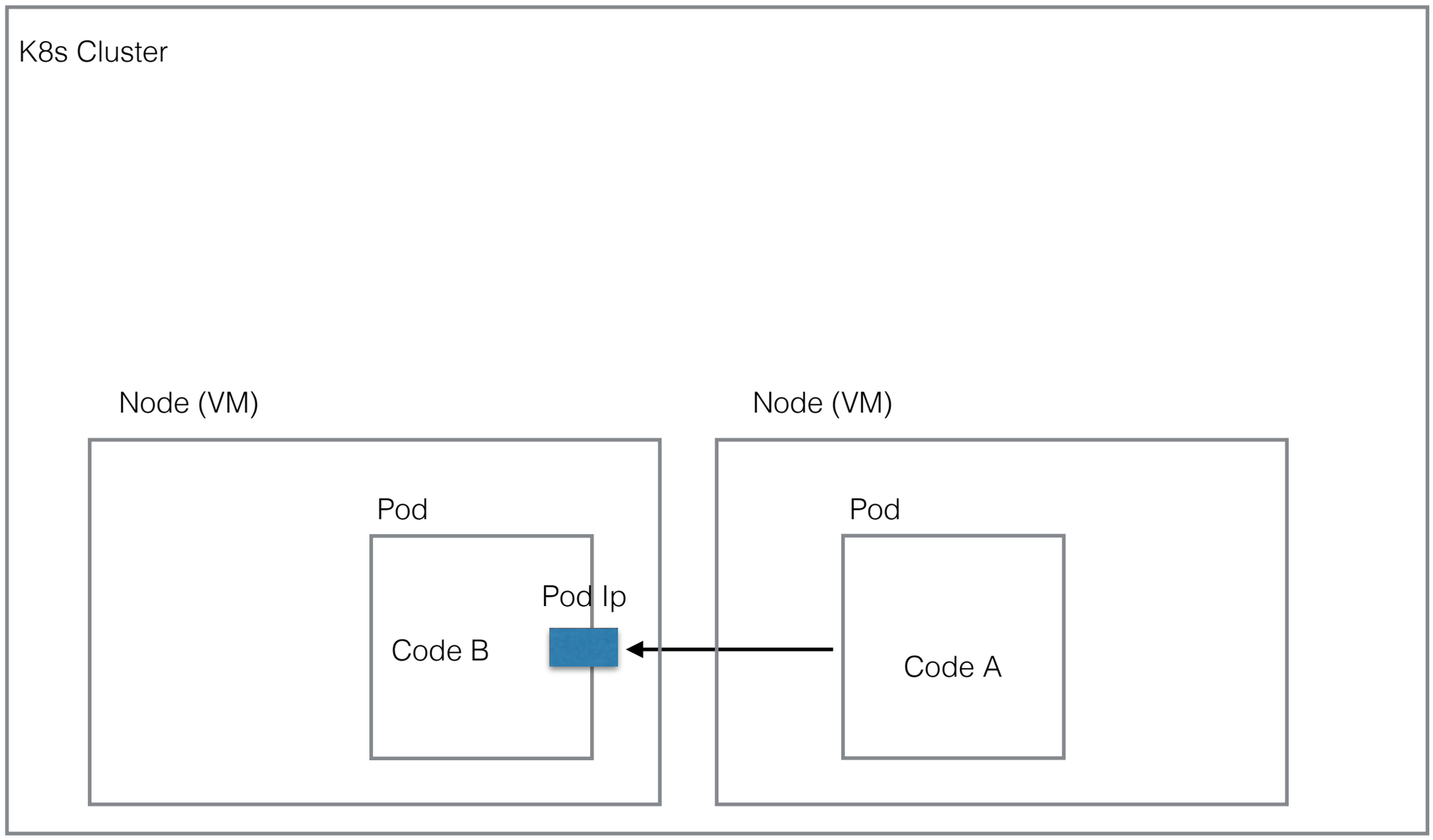
K8s Cluster

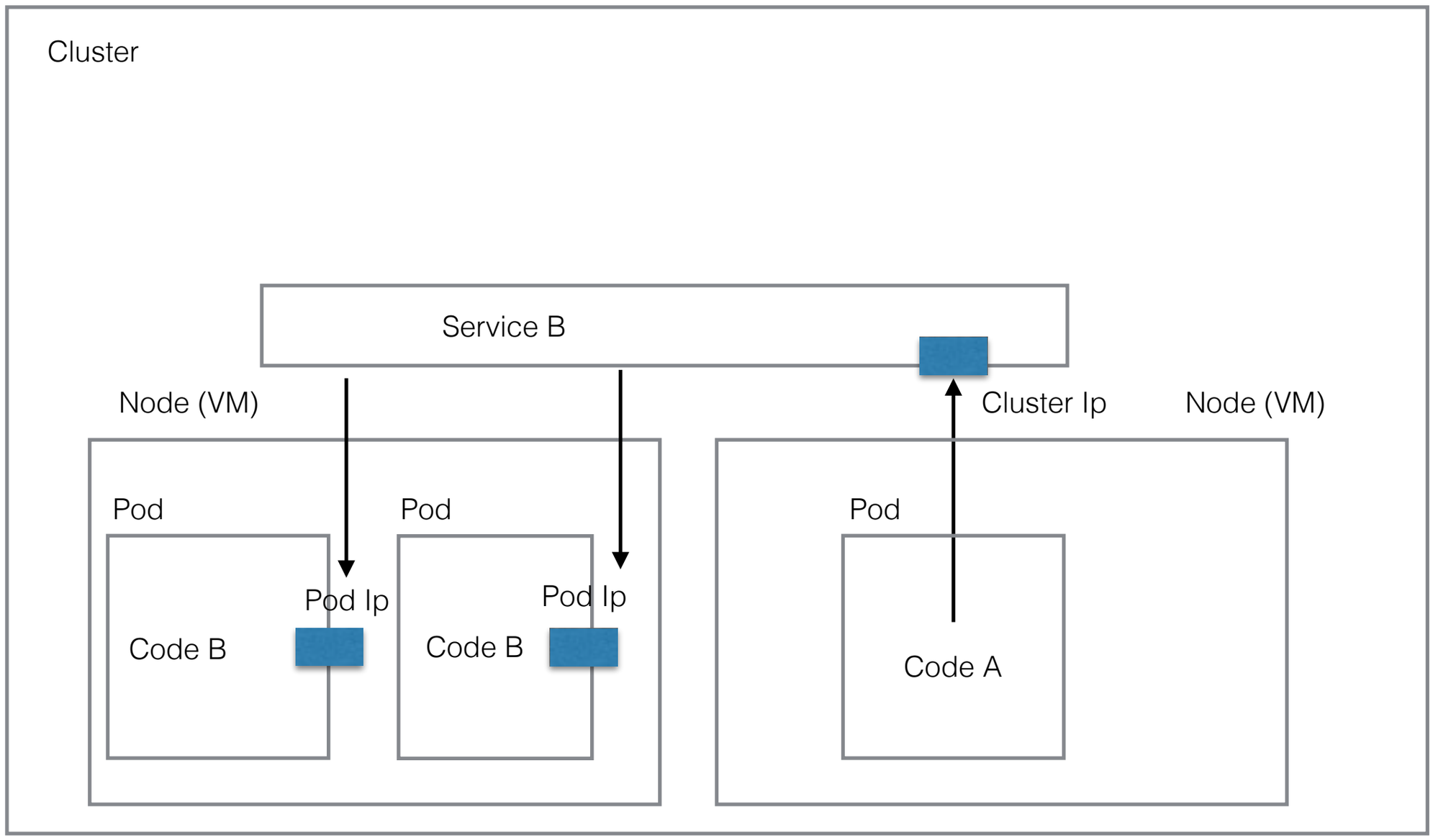
Node (VM)

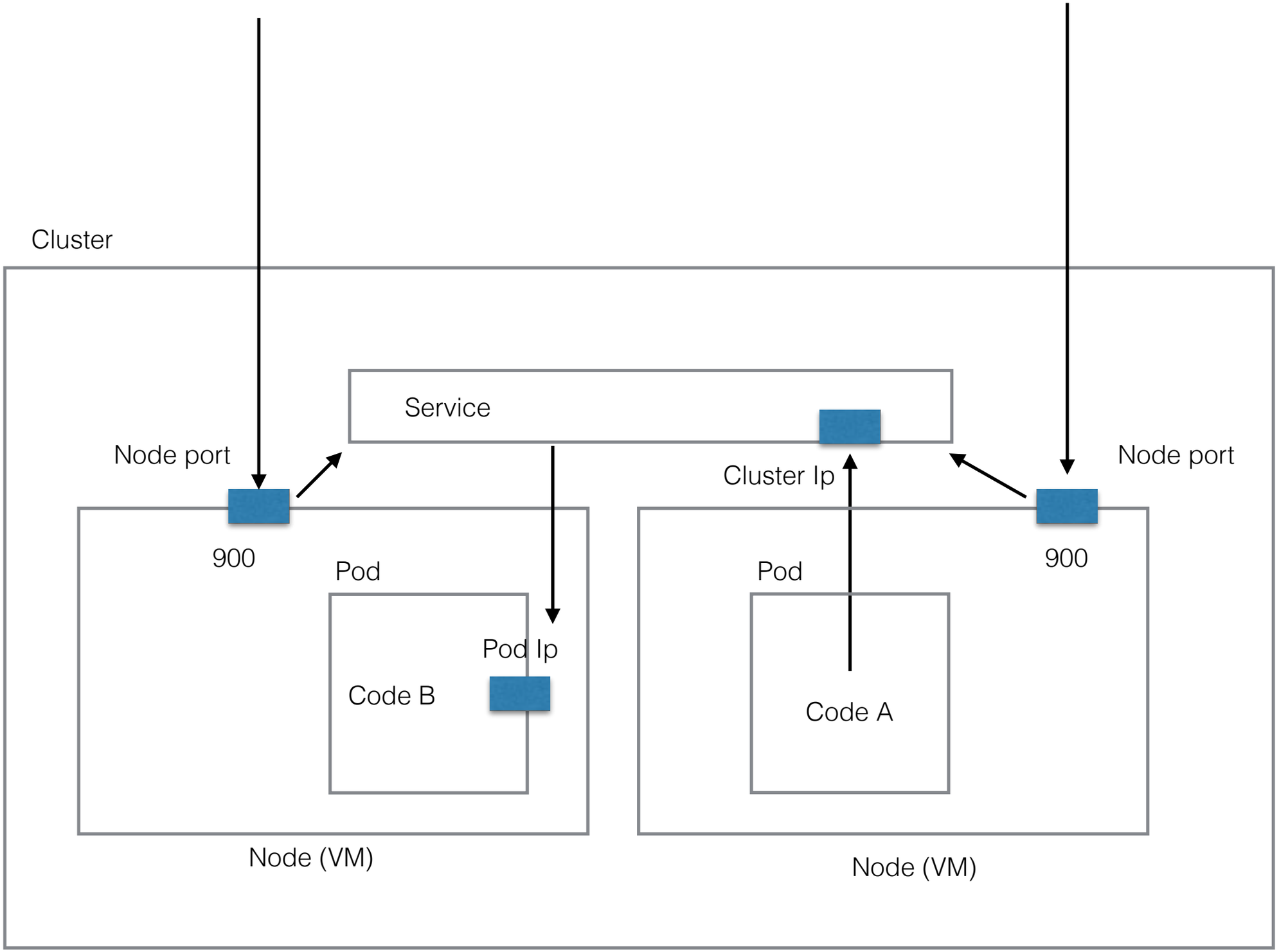
Pod

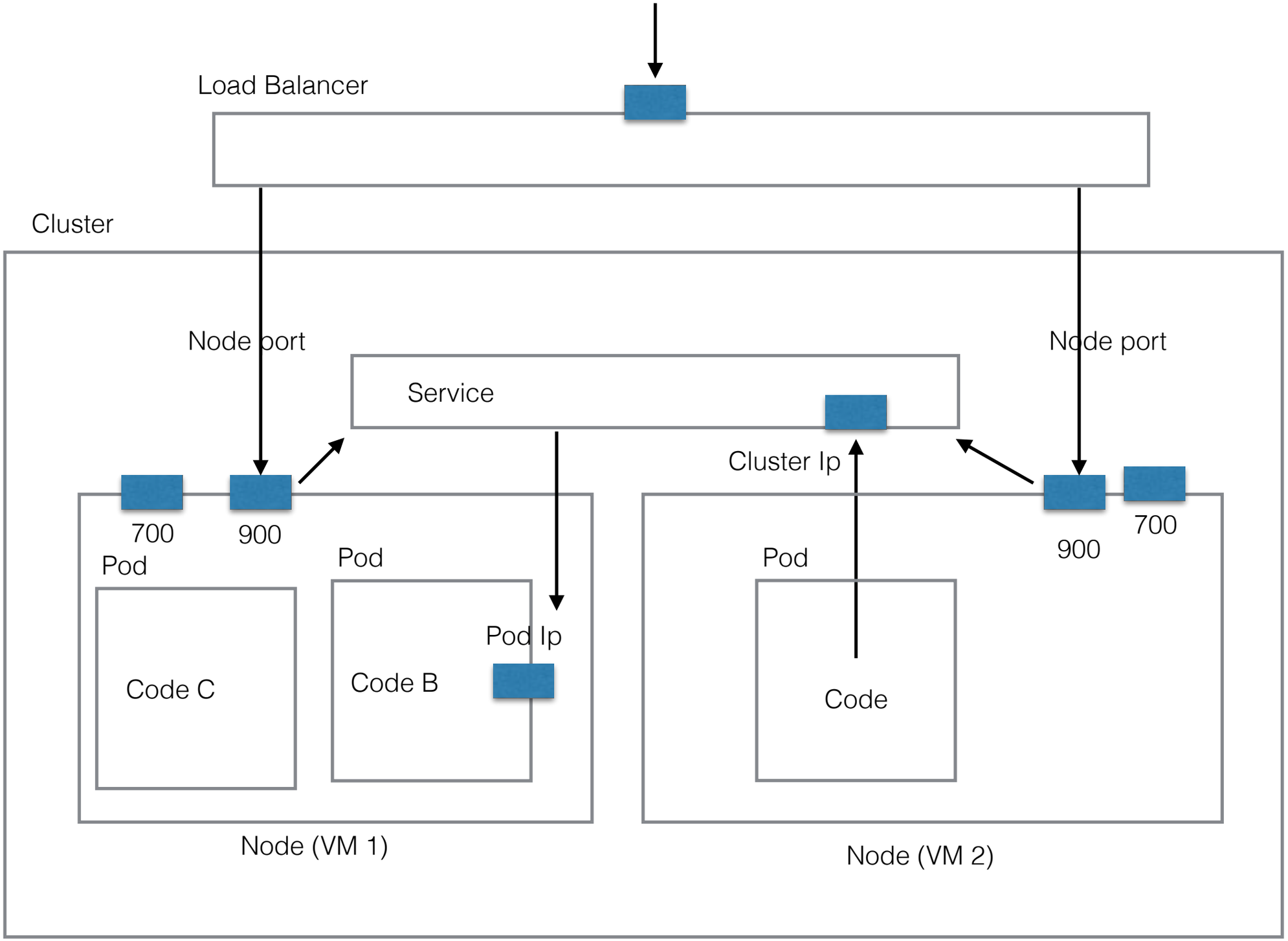
Pod Ip

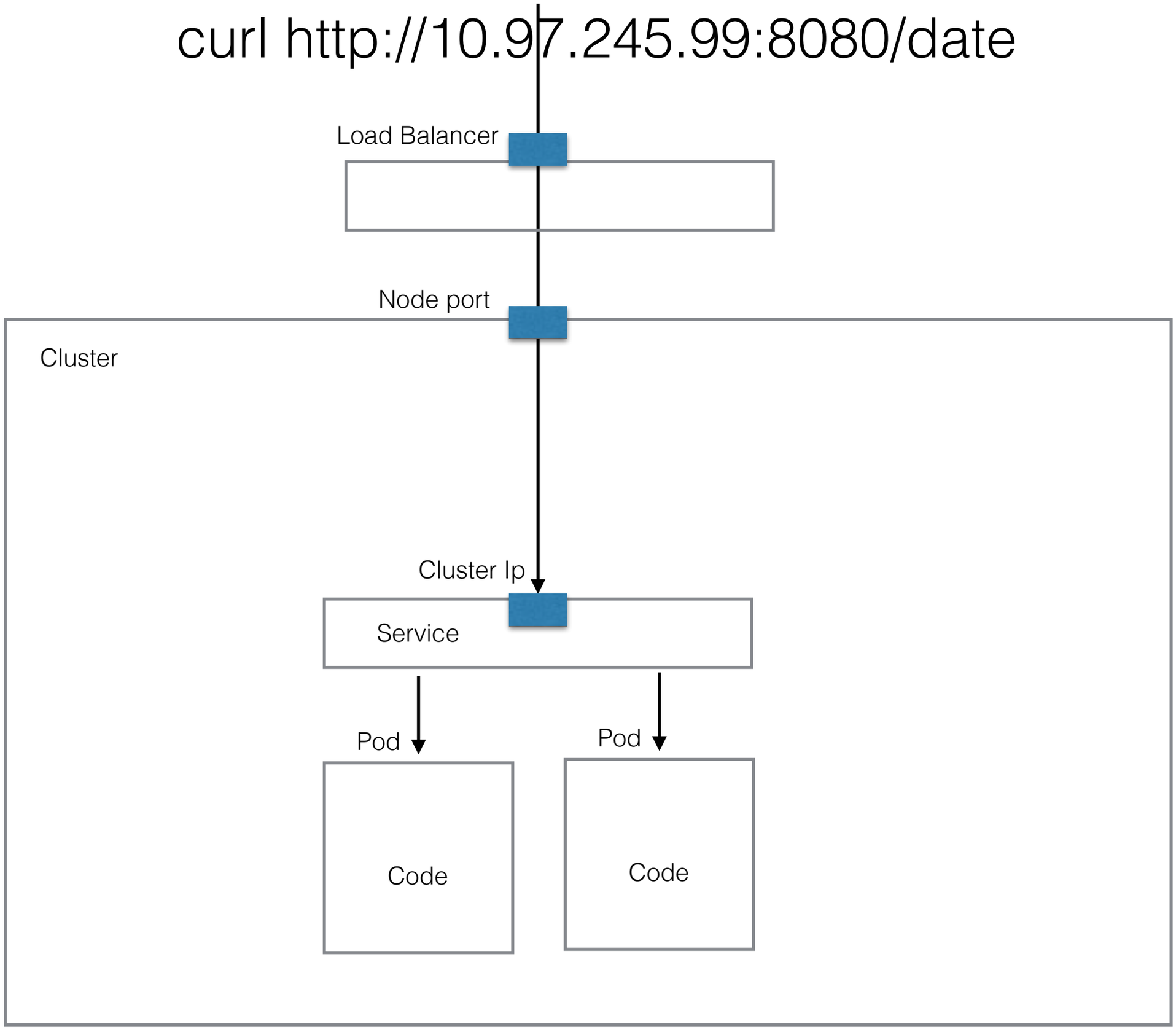
Code


K8s Cluster
Node (VM)
Pod
Pod Ip
Code B
Node (VM)
Pod
Code A

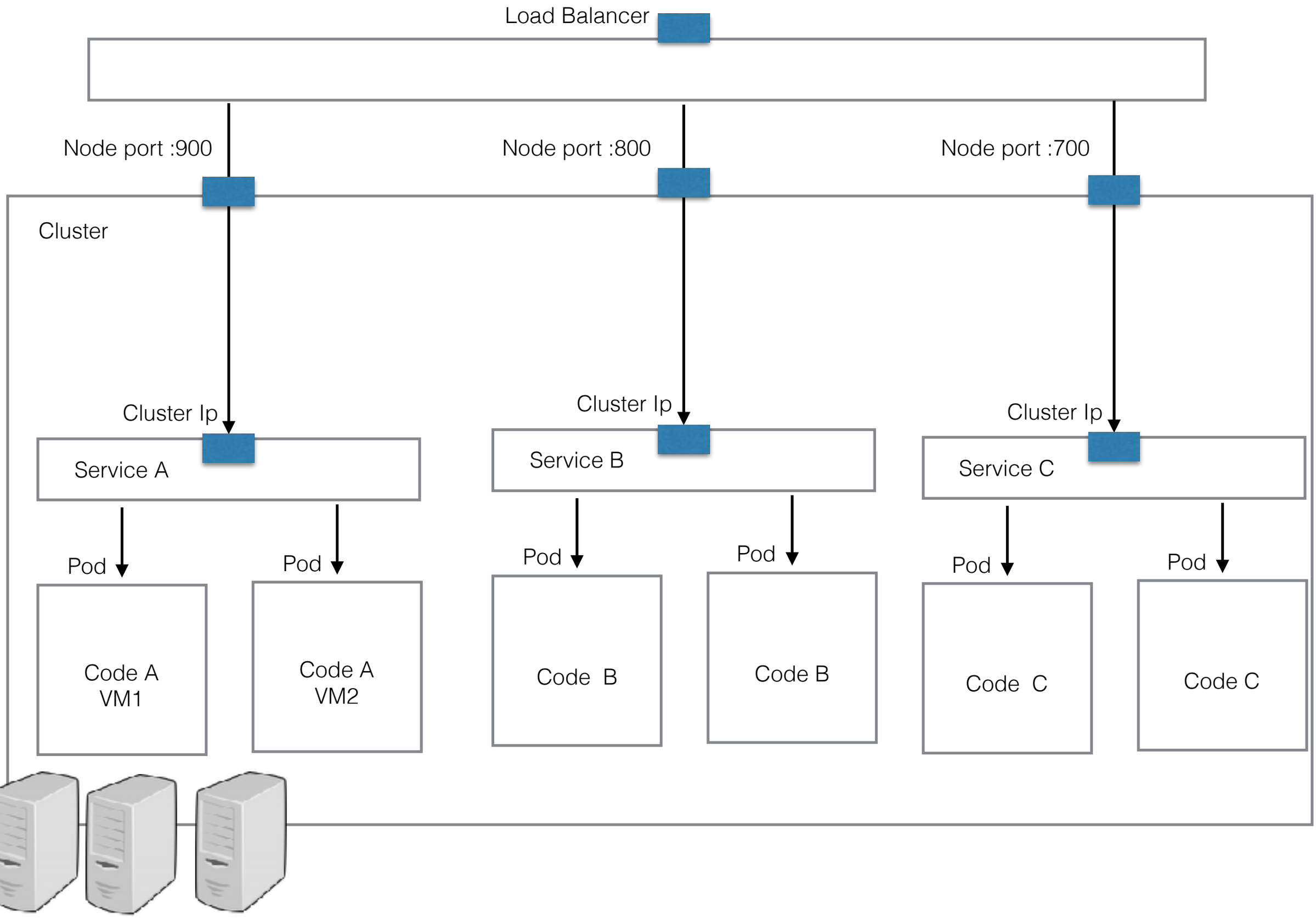

Cluster
Service B
Node (VM)
Cluster Ip
Node (VM)
Pod
Pod
Pod
Pod Ip
Pod Ip
Code B
Code B
Code A

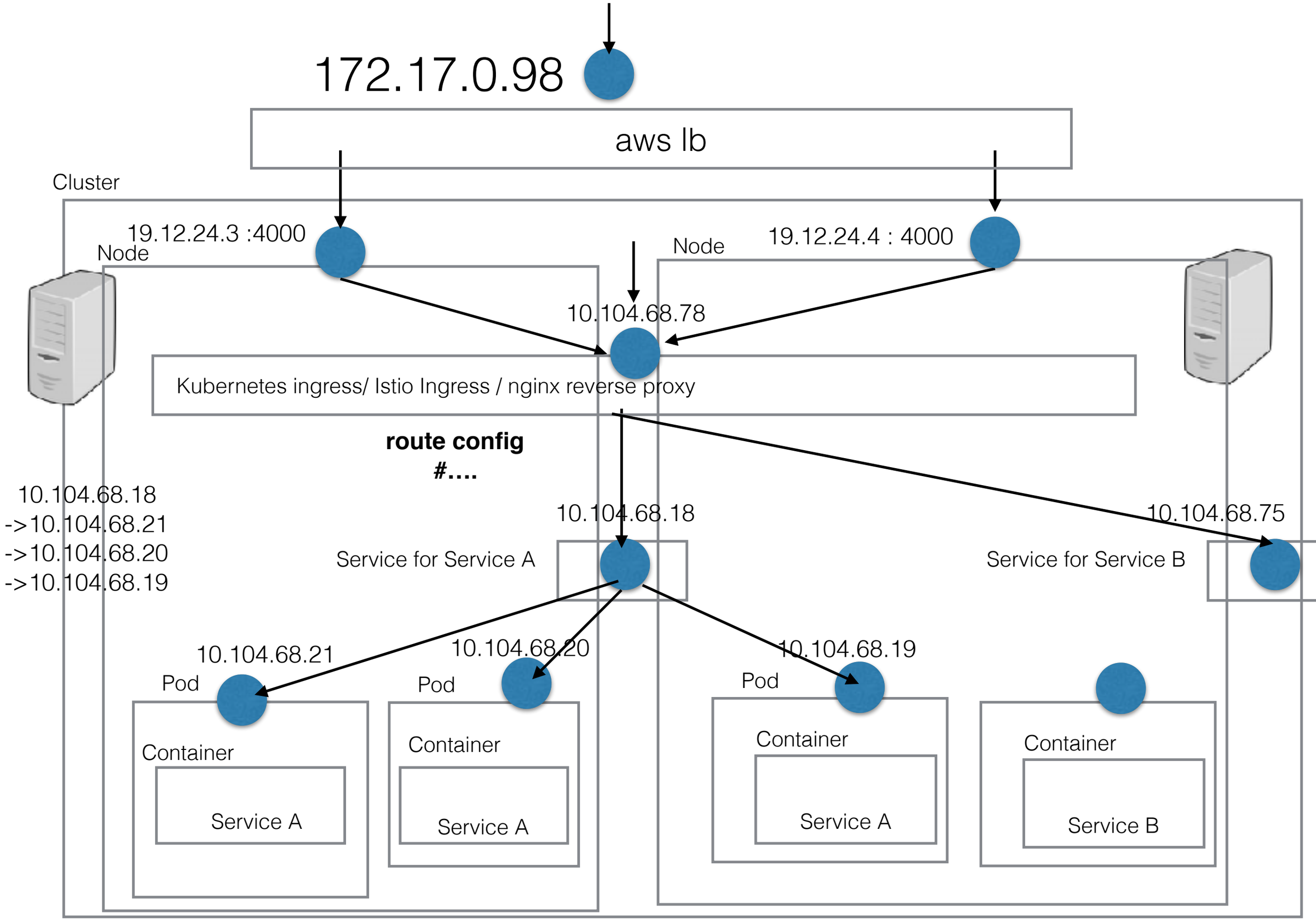

Cluster
Service
Node port
Cluster Ip
Node port
900
Pod
Pod Ip
Code B
Node (VM)
900
Pod
Code A
Node (VM)


Load Balancer
Cluster
Node port
Node port
Service
Cluster Ip
700
900
700
900
Pod
Pod
Pod
Code C
Code B
Pod Ip
Code
Node (VM 1)
Node (VM 2)


curl http://10.97.245.99:8080/date
Load Balancer
Node port
Cluster
Cluster Ip
Service
Pod
Pod
Code
Code

curl http://10.97.245.99:8080/date




172.17.0.98
aws lb
Cluster
19.12.24.3 :4000
Node
19.12.24.4 : 4000
Node
10.104.68.78
Kubernetes ingress/ Istio Ingress / nginx reverse proxy
route config
#....
10.104.68.18
->10.104.68.21
->10.104.68.20
->10.104.68.19
10.104.68.18
Service for Service A
10.104.68.75
Service for Service B
10.104.68.21
Pod
Container
Service A
10.104.68.20
Pod
Container
Service A
10.104.68.19
Pod
Container
Service A
Container
Service B

APi Gateway

172.17.0.98

aws lb

Cluster

19.12.24.3 :4000

Node

19.12.24.4 : 4000

Node

10.104.68.78

Kubernetes ingress/ **Istio Ingress** / nginx reverse proxy

route config
#....

10.104.68.18
->10.104.68.21
->10.104.68.20
->10.104.68.19

10.104.68.18

Service for Service A

10.104.68.75

Service for Service B

10.104.68.21

Pod

Container

Service A

10.104.68.20

Pod

Container

Service A

10.104.68.19

Pod

Container

Service A

Container

Service B

API management platform : API gateway plus a whole lot more features to create an entire API marketplace

VM1

8080

Vnet

Container 1

8080

Container 2

todoDB

Todo
app(py)

Flask

8080

8080

Cpu
RAM
Disk
I/o

Disk

Cpu
RAM
Disk
I/o

Containerization layer (Docker)

Disk

Unix Guest Os

Cpu
RAM
Disk
I/o

VM2

Win Guest Os

Cpu
RAM
Disk
I/o

Virtualisation layer

Host Os (azure)

Cpu
RAM
Disk
I/o

Machine

27017:27017
/root/myapi/mydata:/data/db

INFRASTRUCTURE SETUP

AWS ELB for All Website traffic(eg:www.redbus.in, www.redbus.sg, www.redbus.pe)

NginX Server to segregatel traffic for different geographies to separate AWS ELBs

AWS ELB
(Domestic Traffic)

AWS ELB
(International Traffic)

Amazon EC2 Instances
(ubuntu OS/ Kestrel servers)

Amazon Instances(ubuntu OS/ Kestrel servers)

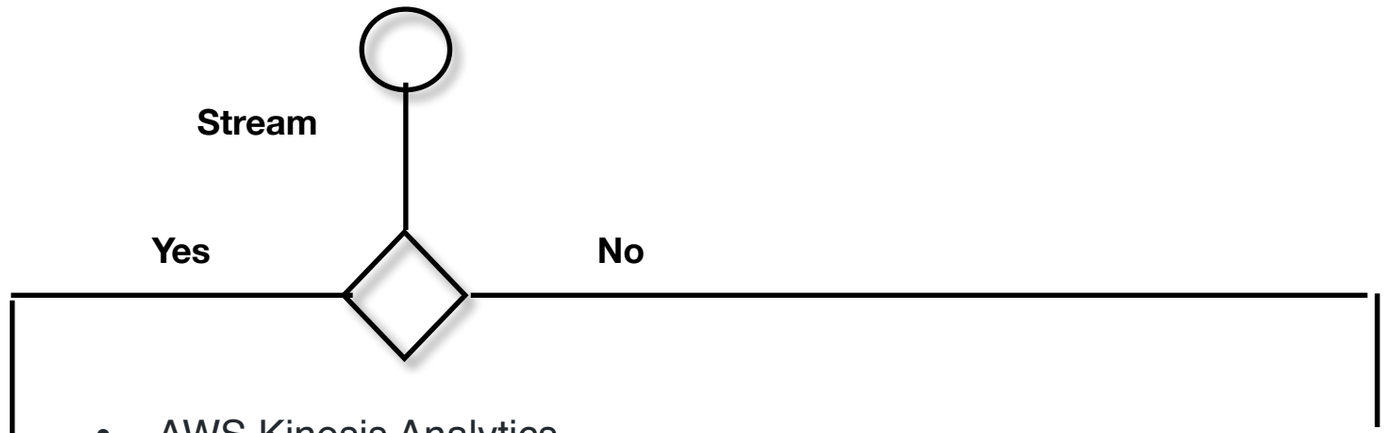
.NET framework
MVC 4
IIS
windows
[$0.192/hr for Windows OS]
http://
XXXX.redbus.pe/, Throughput
was 60.5/Sec.

2–6 EC2s of C4 family
depending on the traffic.

.NET CORE
[$0.1/hr for Linux] 45% of cost

http://XXXX.redbus.pe/ ,
Throughput was 142.9/Sec.

Choose Analytical Compute



Stream Analytics

generates a “complex event”

- AWS Kinesis Analytics
- Apache Flink
- Apache Samba
- Apache Spark Streaming
- Apache Storm
- Apache Beam
- Azure Stream Analytics
- Hazelcast Jet
- Kafka Streams
- Onyx
- Siddhi

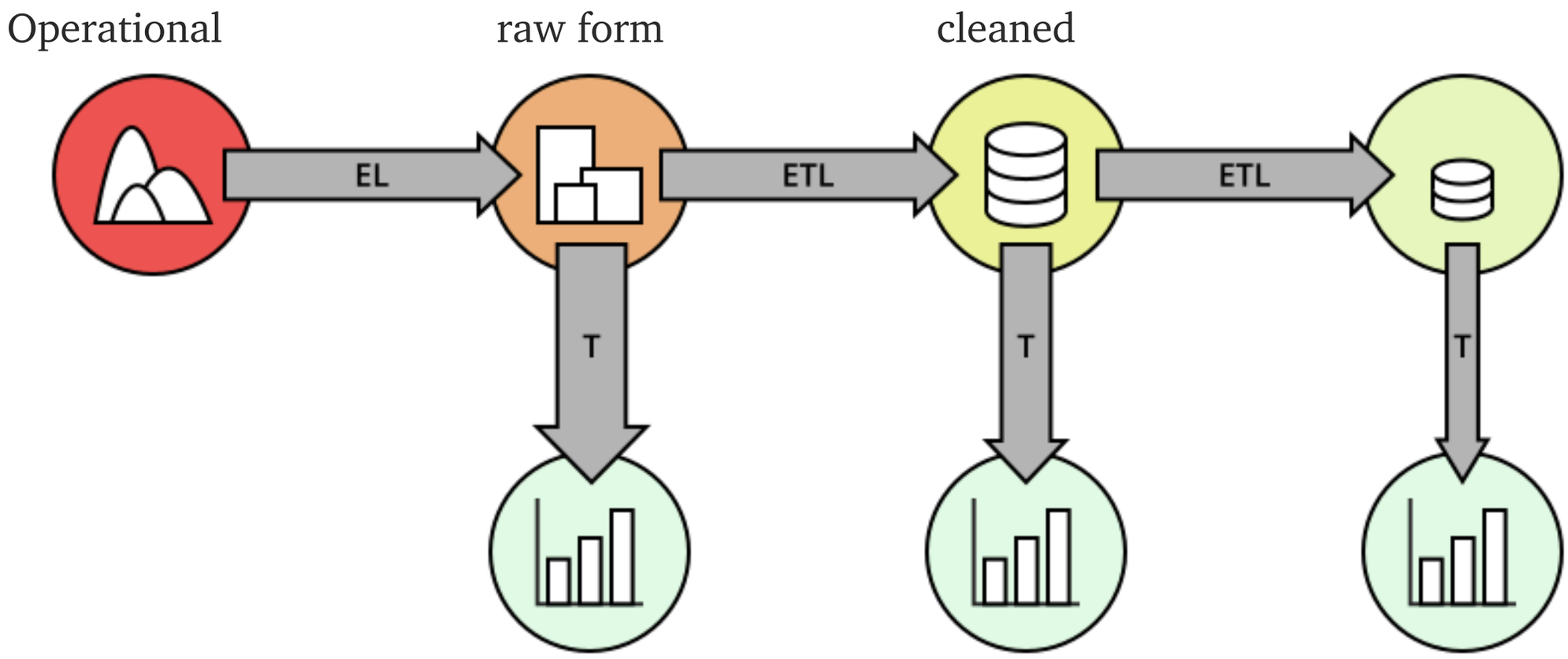
Complex Event Processing (CEP)

- Drools

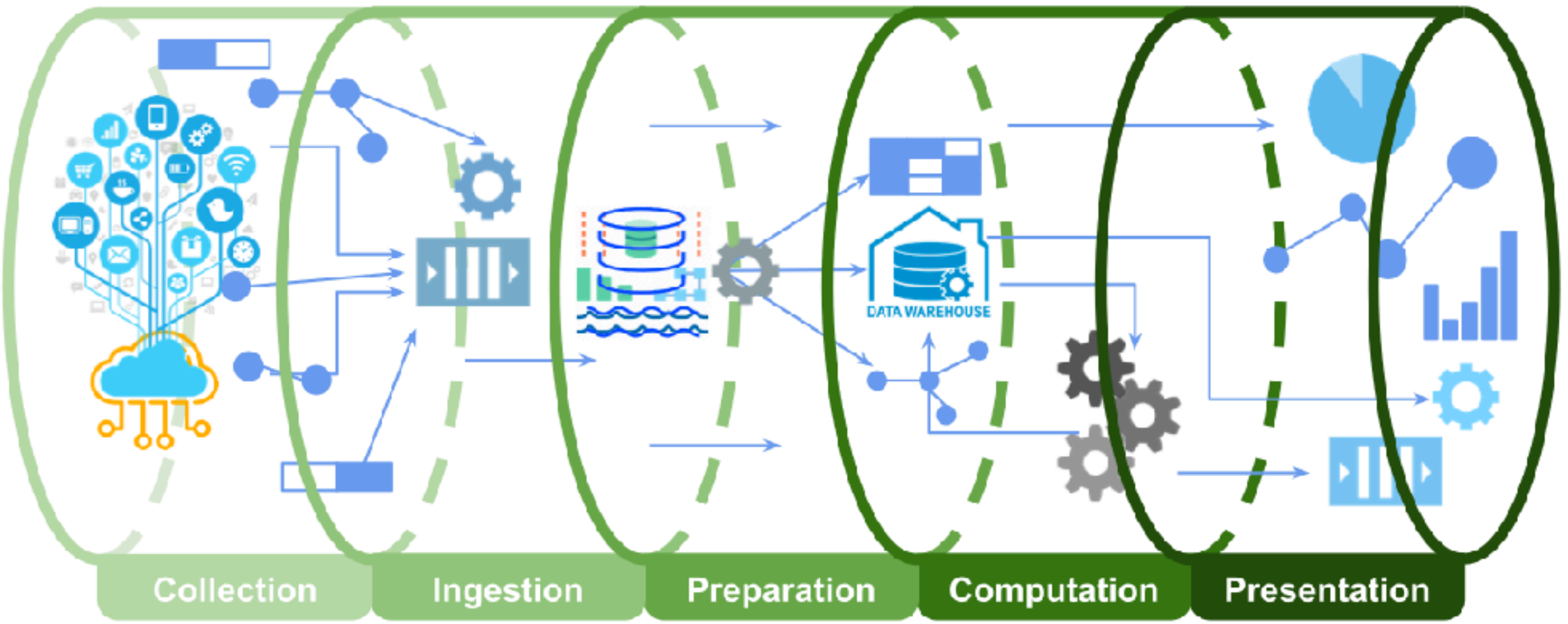
Batch Analytics

- Map Reduce
- Apache Spark
- Apache Flink
- Hive
- Pig
- U-Sql

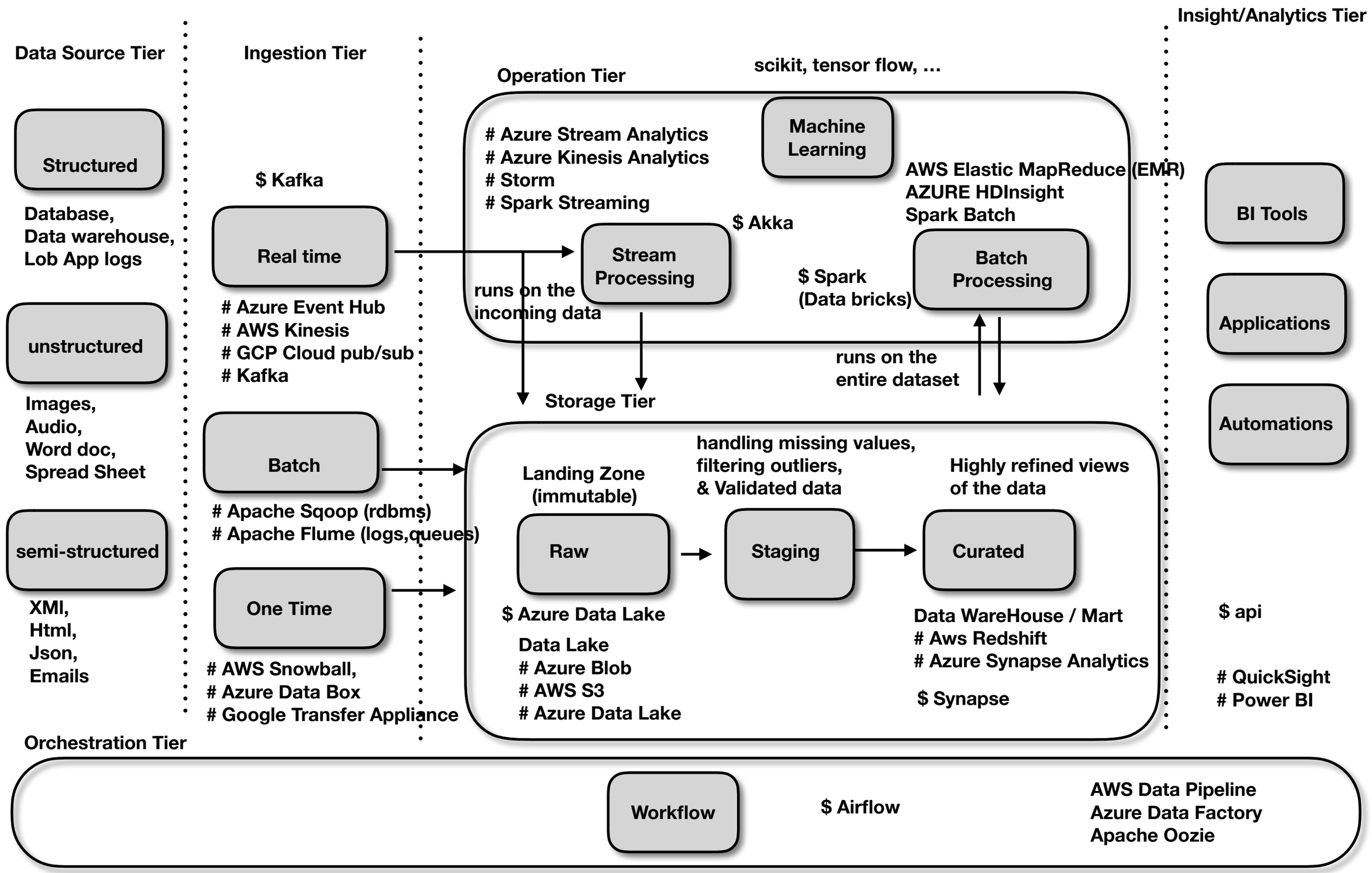

Data Lake
Data Warehouse
Data Mart


Operational
raw form
cleaned
EL
ETL
ETL
T
T
T

Data Pipeline



Reference Architecture
Analytical Application

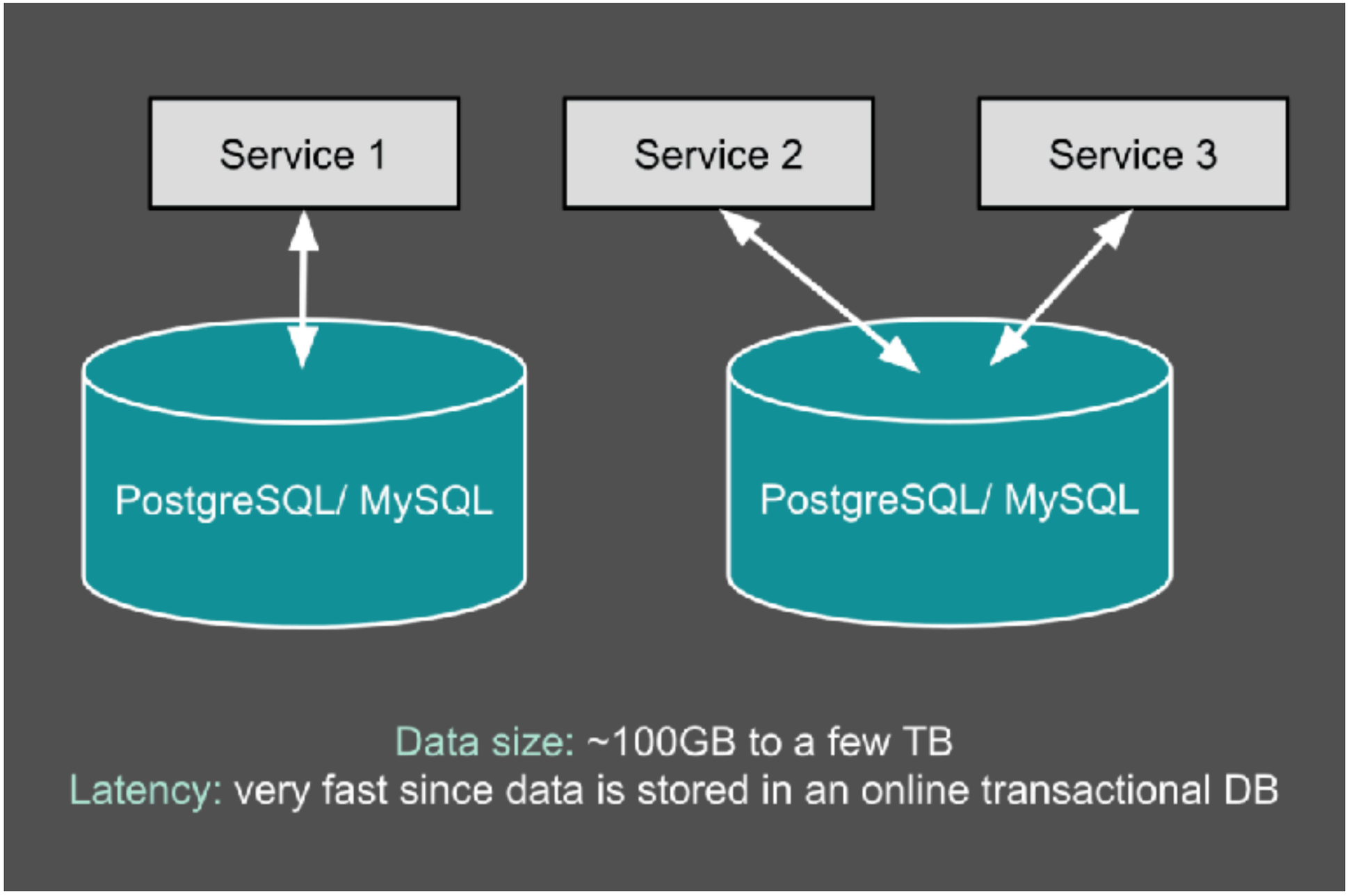



e1,e2,e3,e4,e5,e6,e7,
7 days
a1,a2,a3,a4,…


e1,e2,e3,e4,e5,e6,e7,
Customer Interactions
Clicked
Bounced
Delivered
…
Live
Reporting

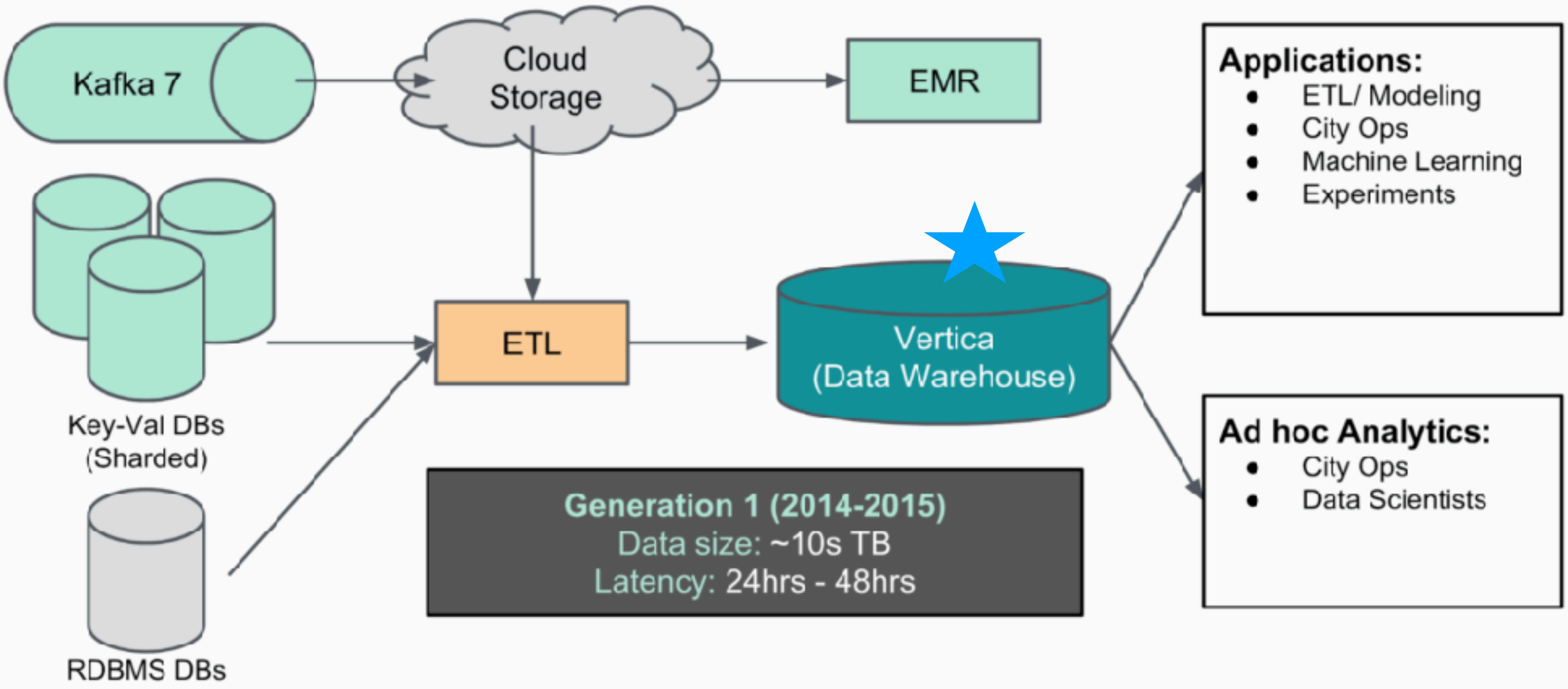

Reporting

Generation 1: The beginning of Big Data at Uber (Before 2014)



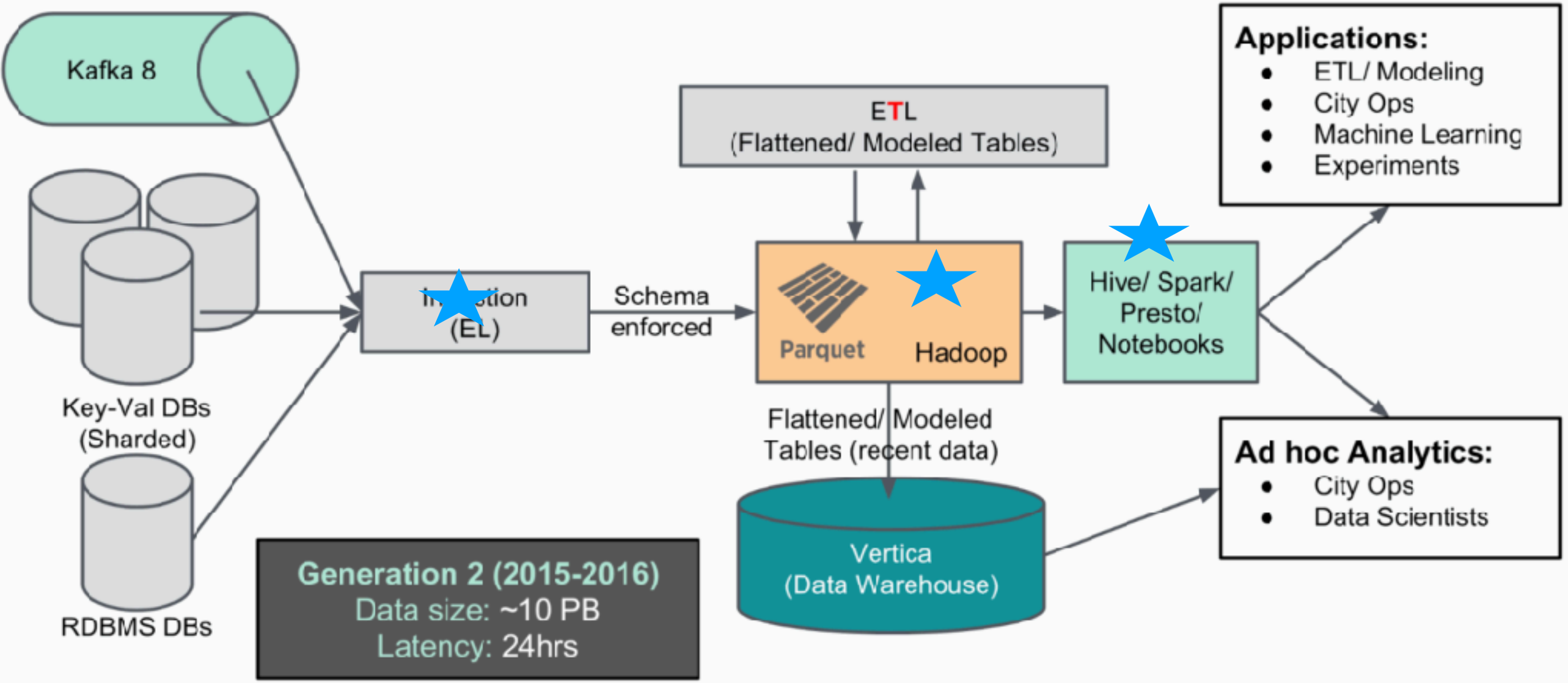
Data was scattered across different OLTP databases.

Generation 1 (2014-2015) - The beginning of Big Data at Uber



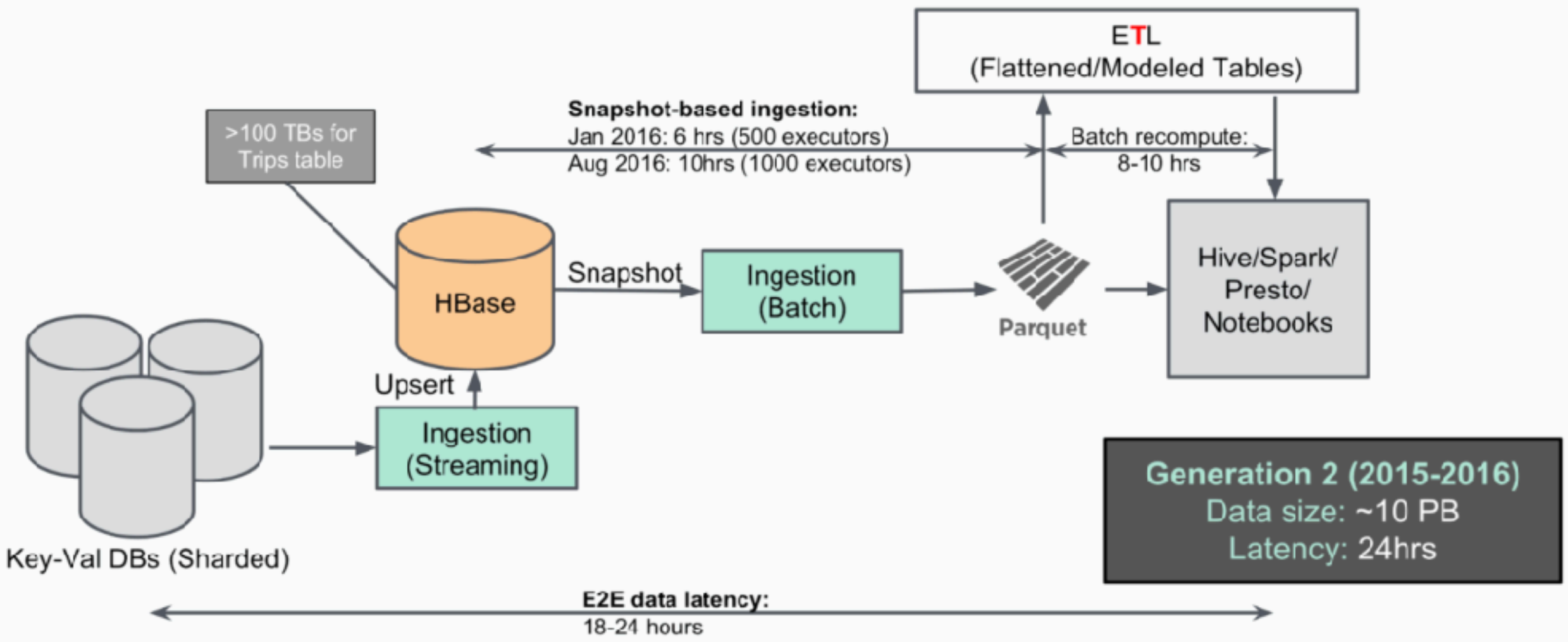
Aggregating all of Uber's data in one place. Developed multiple ad hoc ETL jobs that copied data from different sources (i.e. AWS S3, OLTP databases, service logs, etc.) into Vertica.

Generation 2 (2015-2016) - The arrival of Hadoop



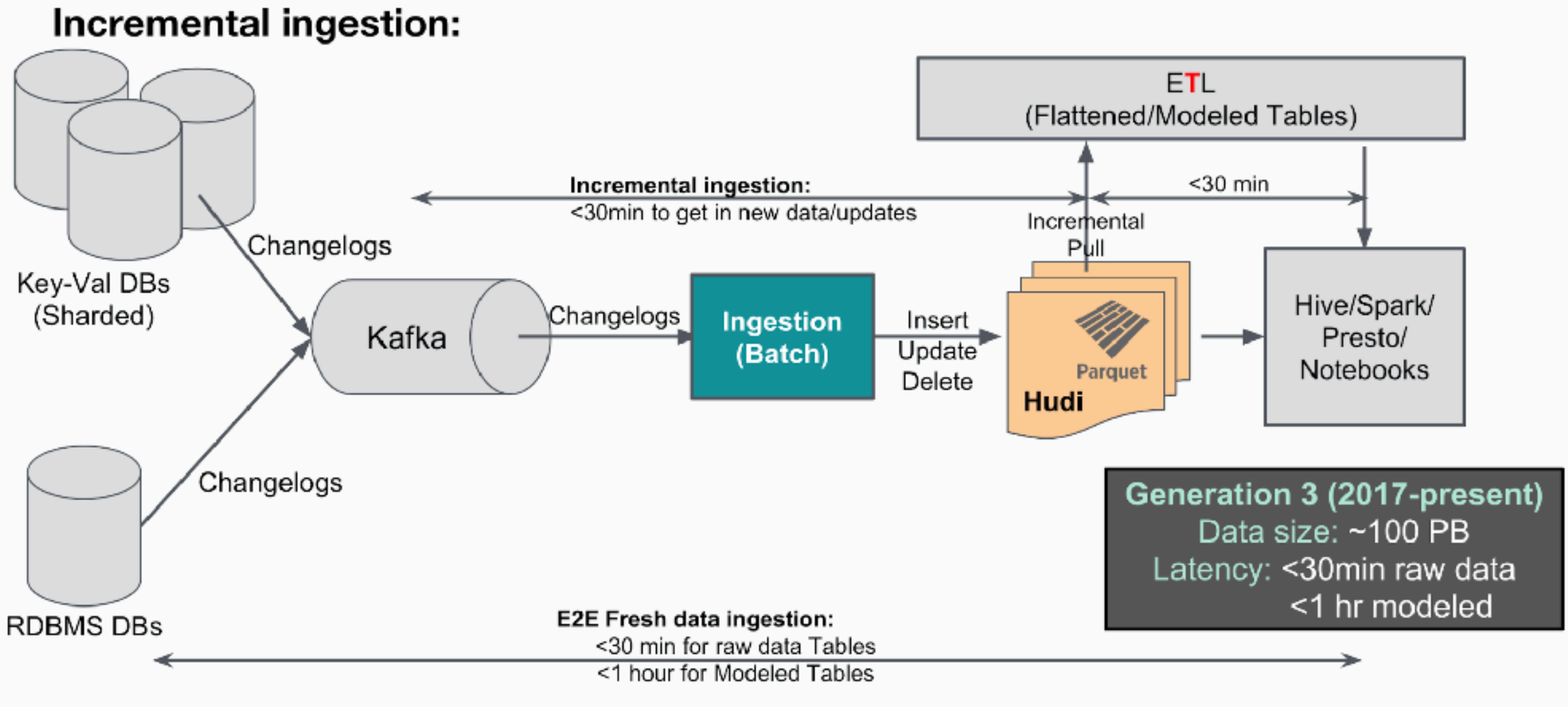
Data was ingested with no transformation during ingestion.

Generation 2 (2015-2016) - The arrival of Hadoop



only over 100 gb of new data was added every day. Since HDFS and Parquet do not support data updates, each run of the ingestion job had to convert the entire, over 100 tb dataset for that specific table.

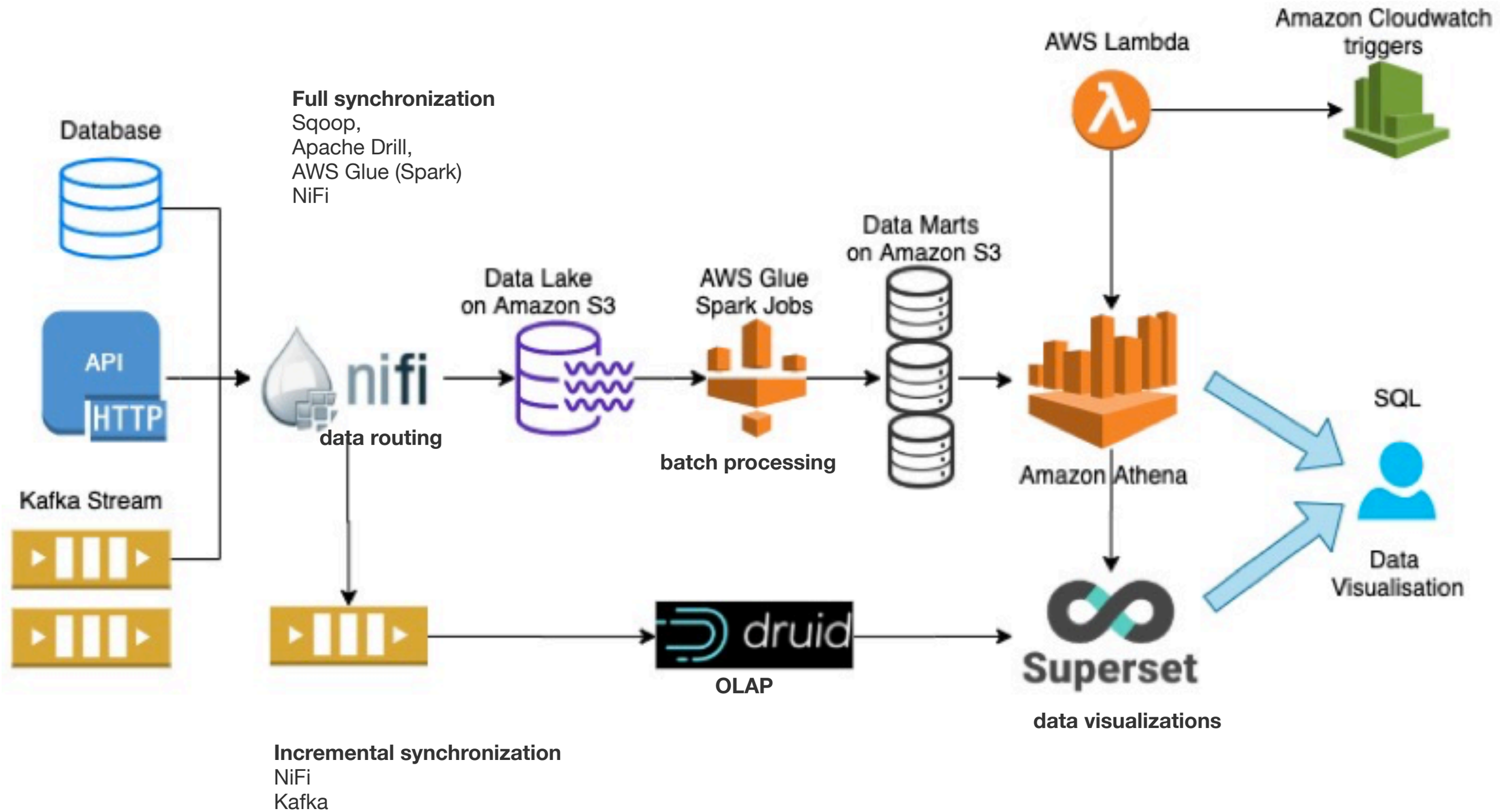
Generation 3 (2017-present) - Let's rebuild for long term



Redbus

700 GB/ week

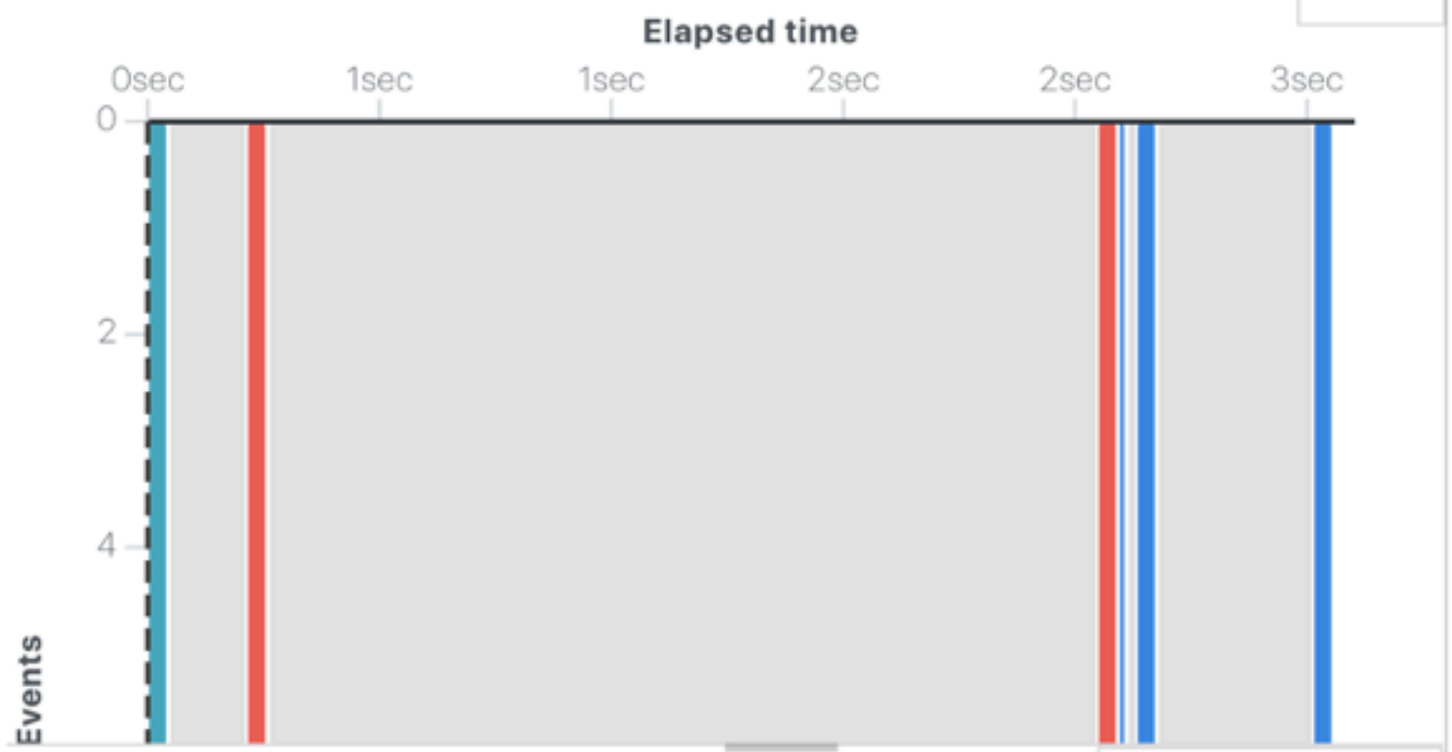
Full synchronization
Sqoop,
Apache Drill,
AWS Glue (Spark)
NiFi



Incremental synchronization
NiFi
Kafka

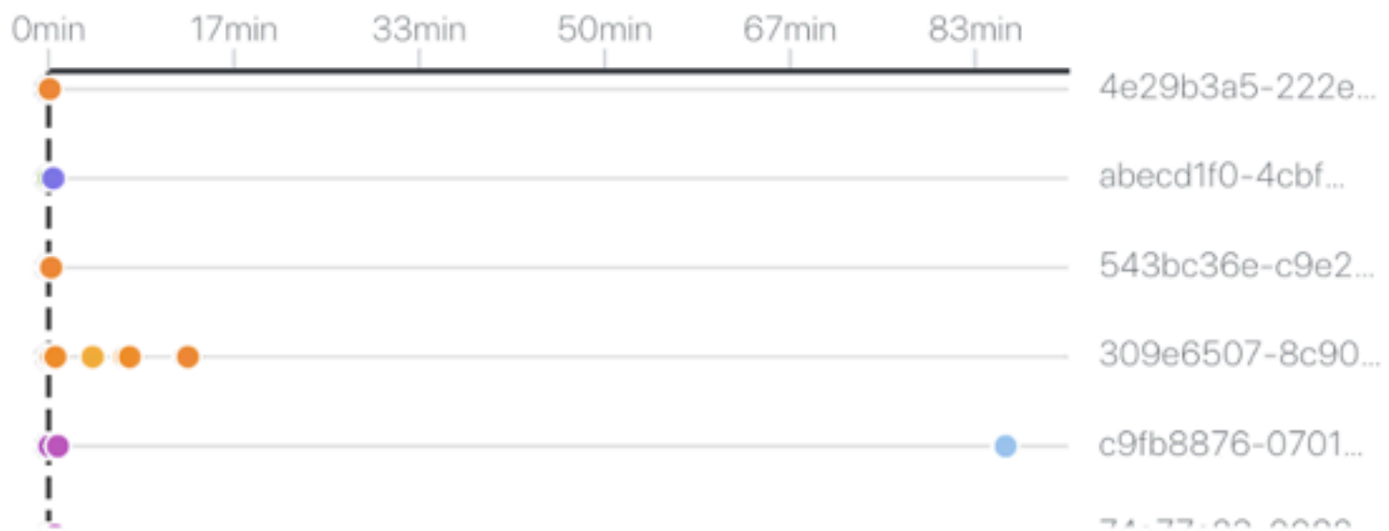
Event Sequence

Hide >



Clear Selection (11)

ALLCONFIG



Align sequences by

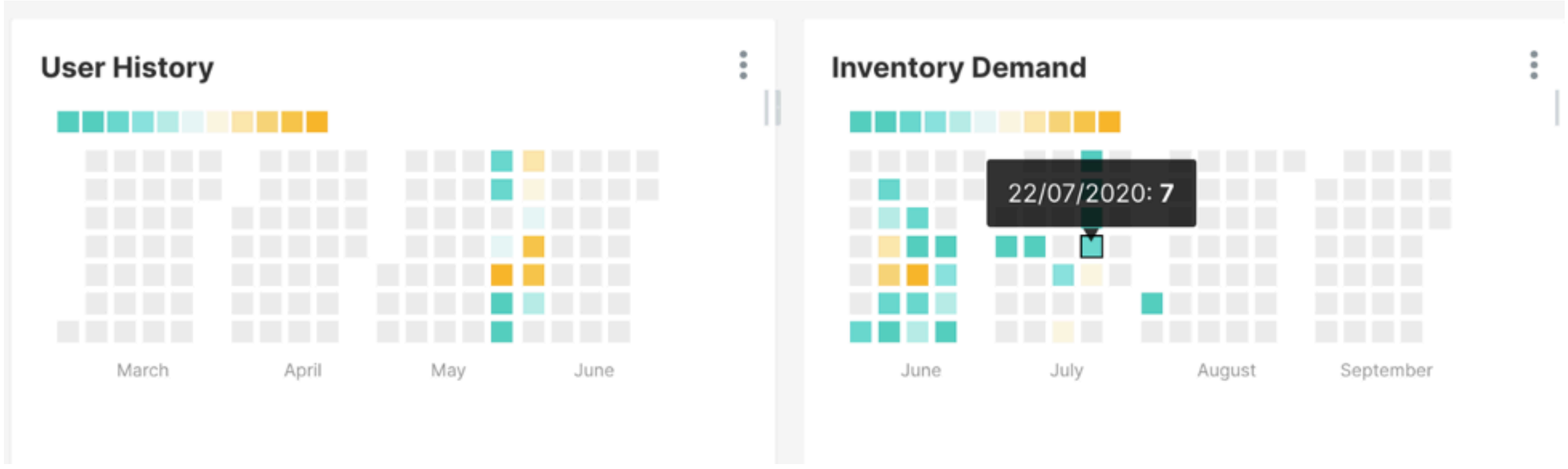
1st - + event

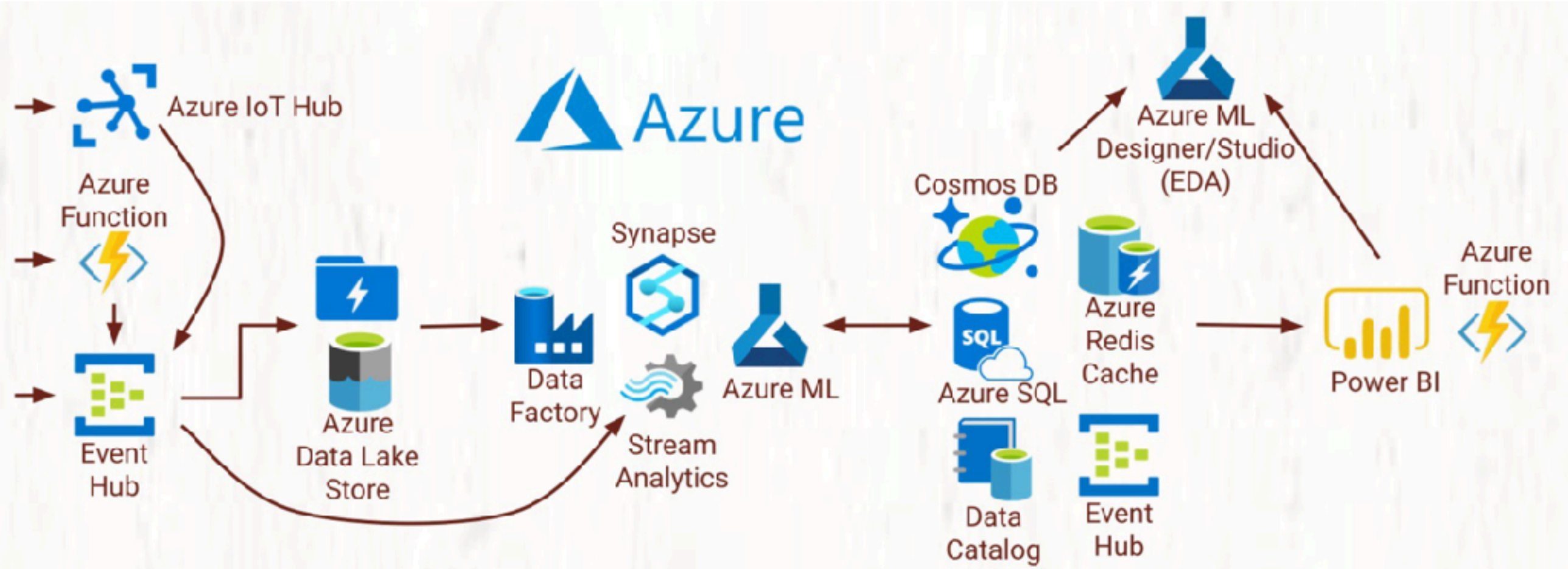
Event type summary

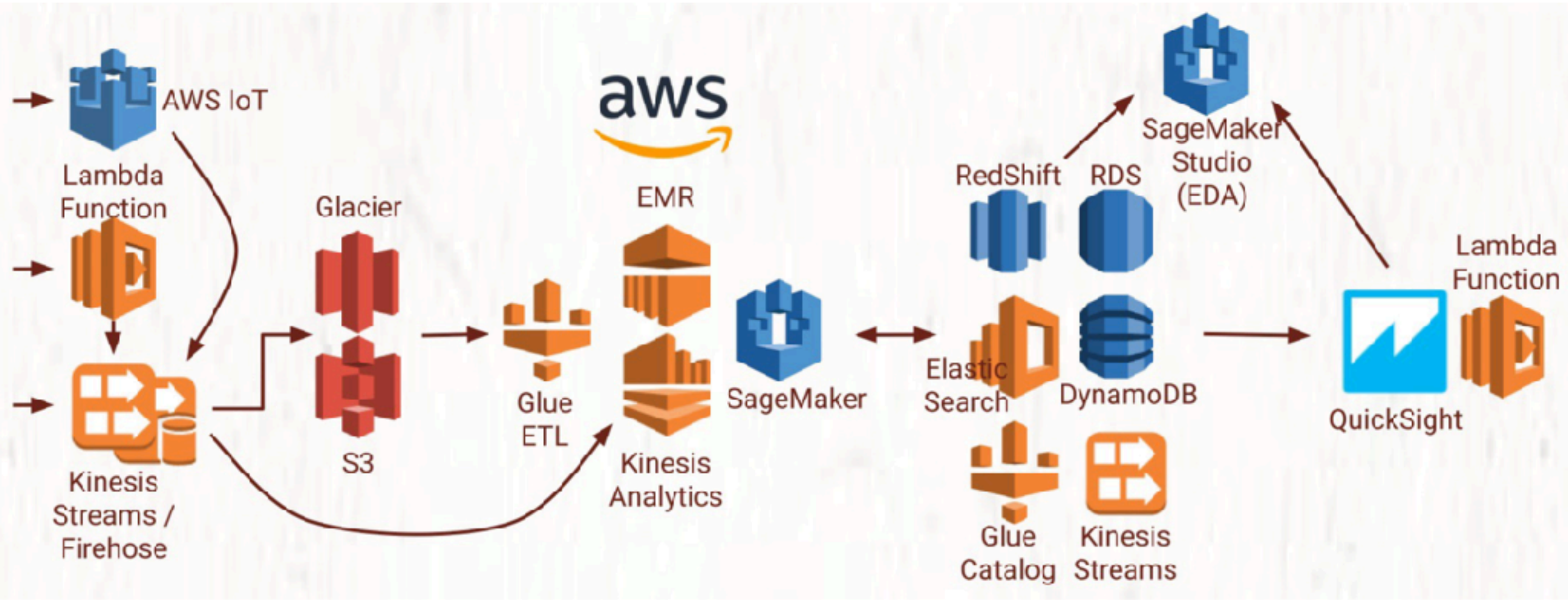
78 events (427 hidden [84.6%])

- allconfig (14.1%)
- busdetails
- busroutes
- filters
- offers (35.9%)
- offerscovid (23.1%)
- persuasion
- perz (26.9%)
- rides
- seatlayout
- signin

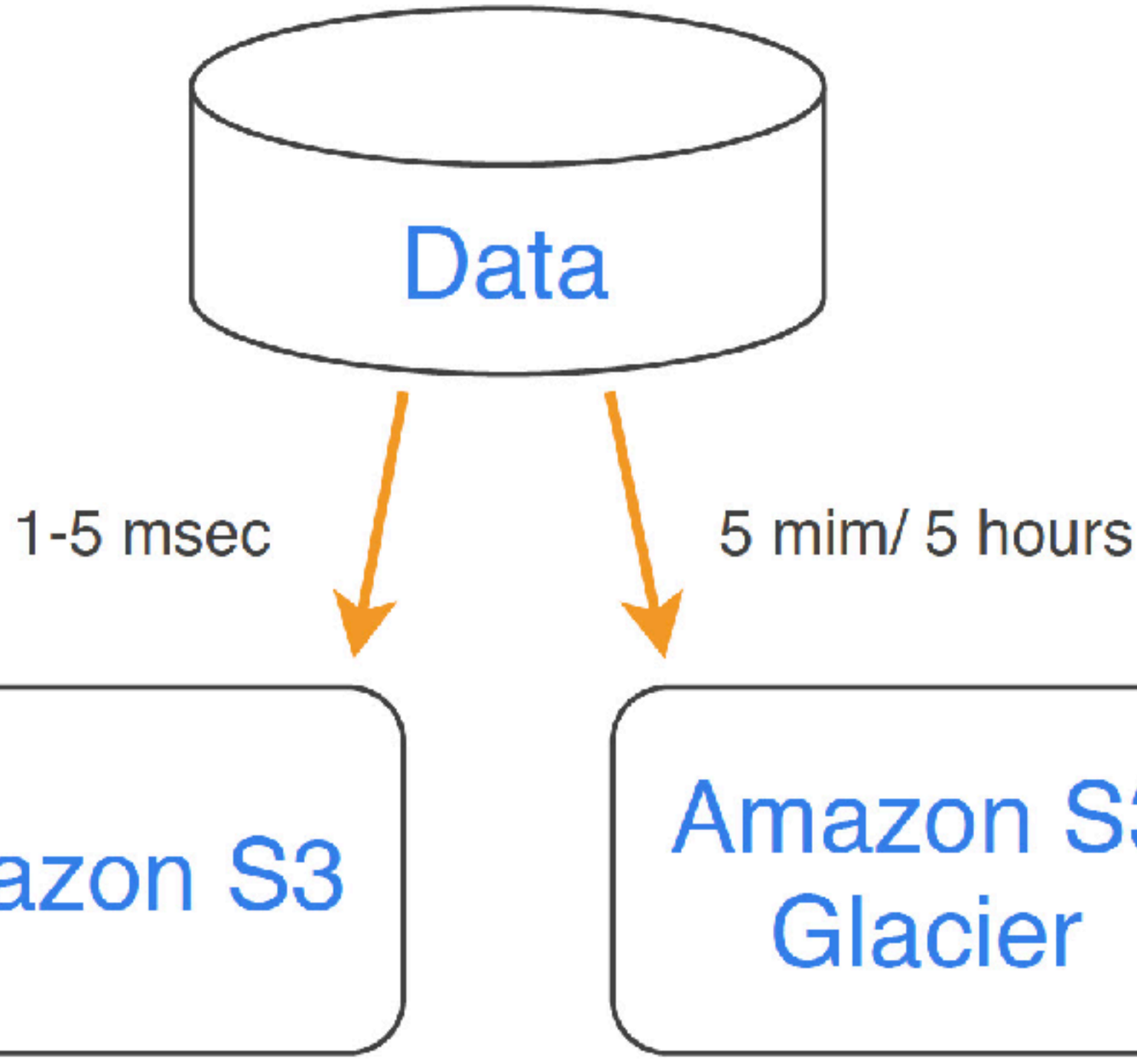
Calendar heat-map




Azure IoT Hub
Azure
Function
Event
Hub
Azure
Data Lake
Store
Synapse
Data
Factory
Stream
Analytics
Azure ML
Azure ML
Designer/Studio
(EDA)
Cosmos DB
SQL
Azure SQL
Data
Catalog
Azure
Redis
Cache
Event
Hub
Power BI
Azure
Function


AWS IoT
Lambda Function
Glacier
EMR
aws
RedShift
RDS
SageMaker Studio (EDA)
Lambda Function
Glue ETL
S3
SageMaker
Elastic Search
DynamoDB
QuickSight
Kinesis Analytics
Kinesis Streams / Firehose
Glue Catalog
Kinesis Streams

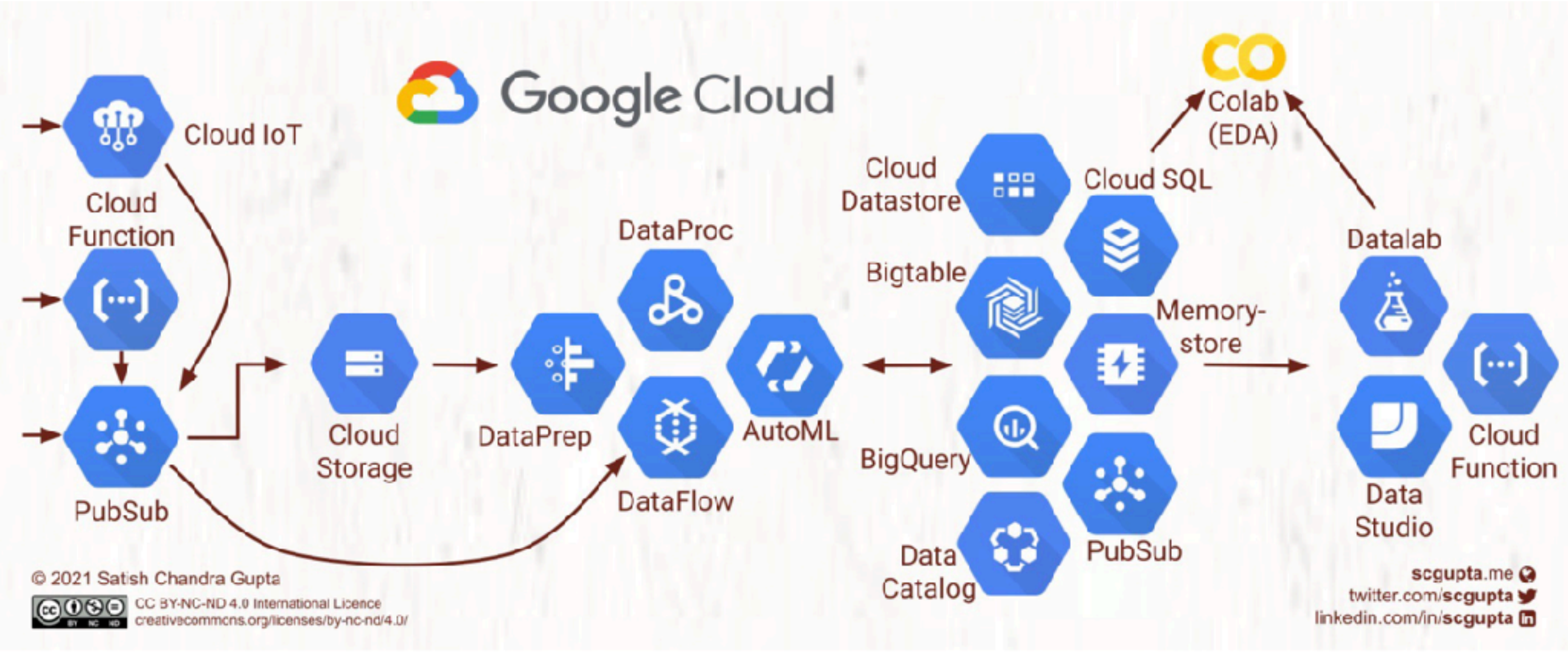
	Kinesis Data Streams	Kinesis Firehose
Purpose	Low latency streaming service for ingest at scale	Data transfer service to load streaming data into Amazon S3, Redshift, Elasticsearch & Splunk
Provisioning	Managed service but needs configuration for shards.	Fully managed service, no administration.
Processing	Real Time (~200ms latency for classic, ~70ms for enhanced fan out)	Near real time (depends on buffer size OR buffer time min. 60 secs)
Scaling	Must manage scaling (configure shards)	Automated Scaling – as per the demand
Data Storage	Configurable from 1 to 7 days	Does not provide data storage
Replay capability	Supports replay capability	Does not support replay capability
Producers	Need to write code for producer. Supports SDK, Kinesis Agent, KPL, CloudWatch, IoT	Need to write code for producer. Supports KPL, Kinesis Agent, Data Streams, CloudWatch, IoT
Consumers	Open ended. Supports multiple consumers and destinations. Supports KCL and Spark.	Closed ended. Handled by Firehose. Does not support KCL or Spark.


Data
1-5 msec
5 mim/ 5 hours
Amazon S3
Amazon S3 Glacier
$0.025 per GB per month
$0.004 per GB per month + extraction cost

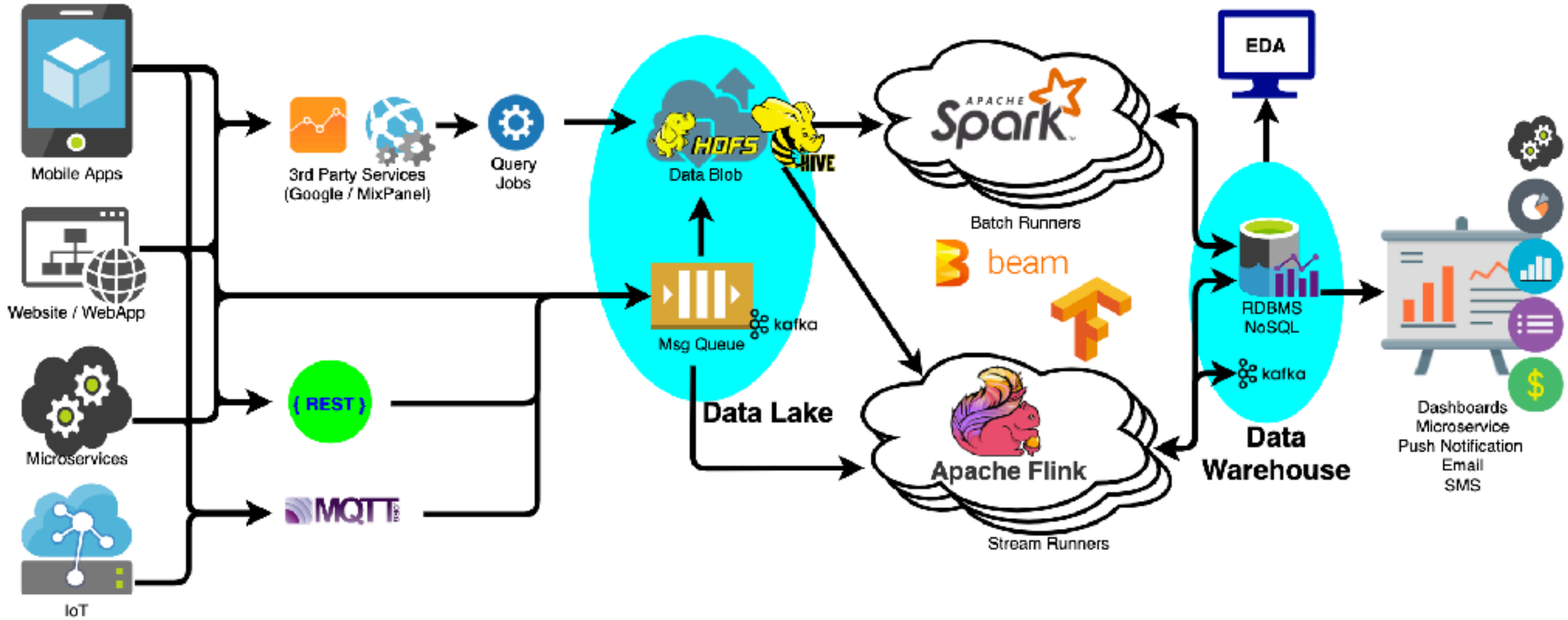
	S3	S3 Glacier
Access speed	Hi	Low
API availability	Yes	Yes
Data storage cost	Relatively high	Very low
Object size	Up to 5 Tb	40 Tb
Accommodation in regions	Many regions	Average
Static Web Content	Yes	No
Supporting Versioning	Yes	No

	Data Streams	Data Firehose	Data Analytics	Video Streams
Short definition	Scalable and durable real-time data streaming service.	Capture, transform, and deliver streaming data into data lakes, data stores, and analytics services.	Transform and analyze streaming data in real time with Apache Flink.	Stream video from connected devices to AWS for analytics, machine learning, playback, and other processing.
Data sources	Any data source (servers, mobile devices, IoT devices, etc) that can call the Kinesis API to send data.	Any data source (servers, mobile devices, IoT devices, etc) that can call the Kinesis API to send data.	Amazon MSK, Amazon Kinesis Data Streams, servers, mobile devices, IoT devices, etc.	Any streaming device that supports Kinesis Video Streams SDK.
Data consumers	Kinesis Data Analytics, Amazon EMR, Amazon EC2, AWS Lambda	Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, generic HTTP endpoints, Datadog, New Relic, MongoDB, and Splunk	Analysis results can be sent to another Kinesis stream, a Kinesis Data Firehose delivery stream, or a Lambda function	Amazon Rekognition, Amazon SageMaker, MxNet, TensorFlow, HLS-based media playback, custom media processing application
Use cases	– Log and event data collection – Real-time analytics – Mobile data capture – Gaming data feed	– IoT Analytics – Clickstream Analytics – Log Analytics – Security monitoring	– Streaming ETL – Real-time analytics – Stateful event processing	– Smart technologies – Video-related AI/ML – Video processing

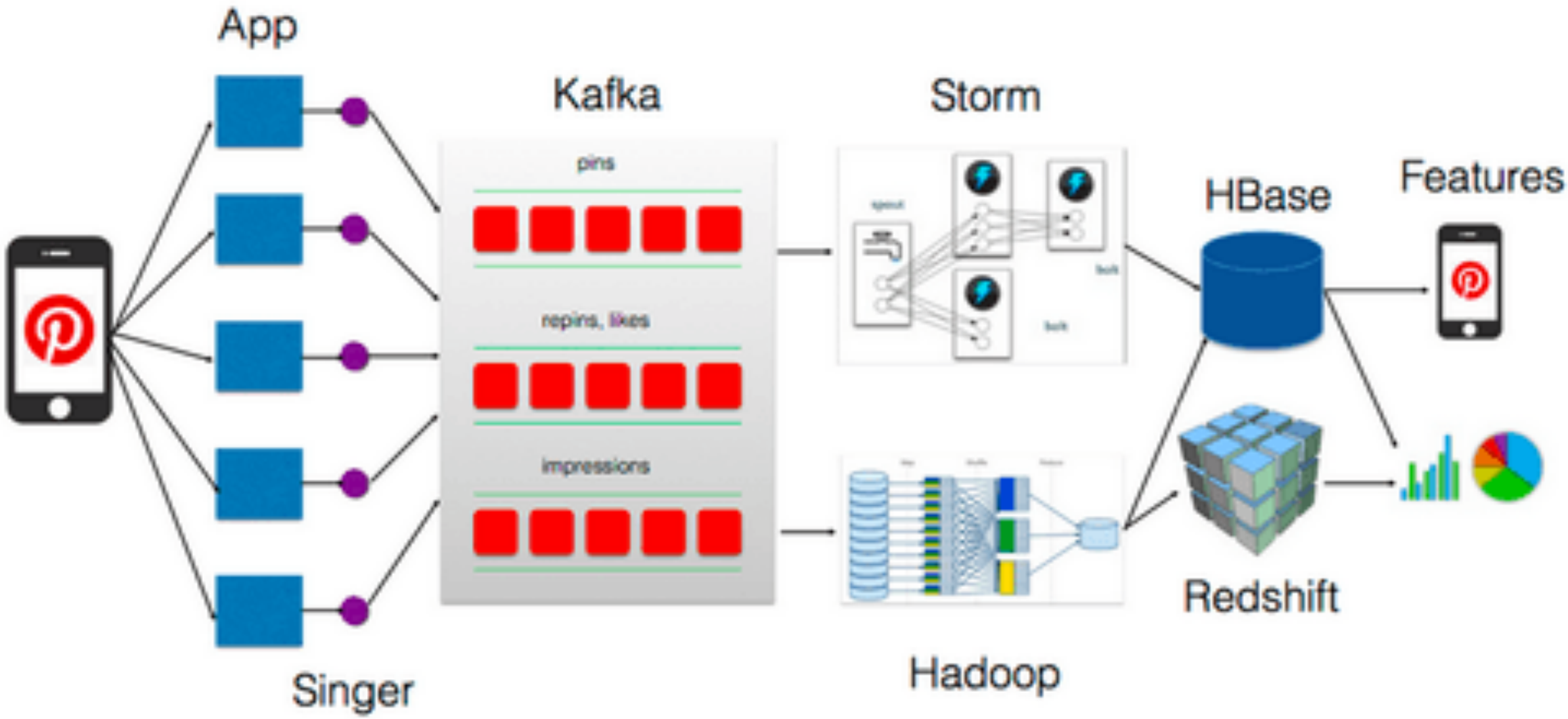
S. No.	Amazon DynamoDB	Amazon Redshift
1	It was developed by Amazon in 2012.	It was developed by Amazon in 2012.
2	It is hosted, scalable database service by Amazon with data stored in Amazon cloud.	It is large scale data warehouse service for use with business intelligence tools.
3	It does not support SQL query language.	It supports SQL query language. But it does not fully support an SQL-standard.
4	It does not provide concept of Referential Integrity. Hence, no Foreign Keys.	It provides concept of Referential Integrity. Hence, there are Foreign Keys.
5	Its Primary database models are Document store and Key-value store.	Its primary database model is Relational DBMS.
6	It does not support Server-side scripting.	It supports user-defined functions for Server-side scripting in python.
7	Eventual Consistency and Immediate Consistency are used to ensure consistency in distributed system.	Immediate Consistency is used to ensure consistency in distributed system.
8	It does not offers API for user-defined Map/Reduce methods. But maybe implemented via Amazon Elastic MapReduce.	It does not offer API for user-defined Map/Reduce methods.
9	It supports secondary indexes.	It supports restricted secondary indexes.


Google Cloud
Cloud IoT
Cloud Function
PubSub
Cloud Storage
DataProc
DataPrep
DataFlow
AutoML
BigQuery
Cloud Datastore
Bigtable
Data Catalog
Cloud SQL
Memory-store
PubSub
Colab (EDA)
Datalab
Data Studio
Cloud Function
© 2021 Satish Chandra Gupta
CC BY-NC-ND 4.0 International Licence
creativecommons.org/licenses/by-nc-nd/4.0/
scgupta.me
twitter.com/scgupta
linkedin.com/in/scgupta

Open Source Data analytics Architecture

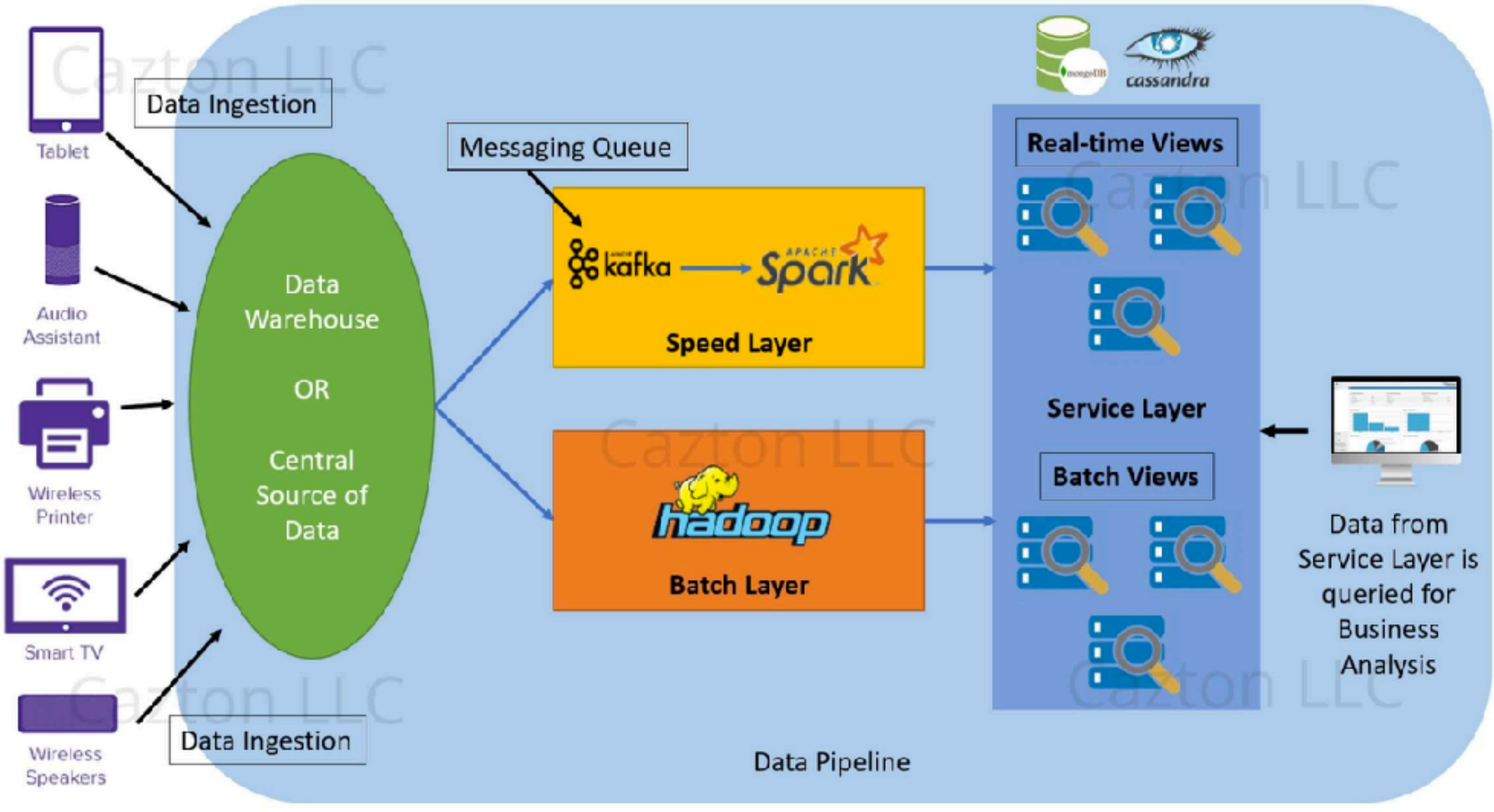


Data analytics Architecture adopted by Pinterest



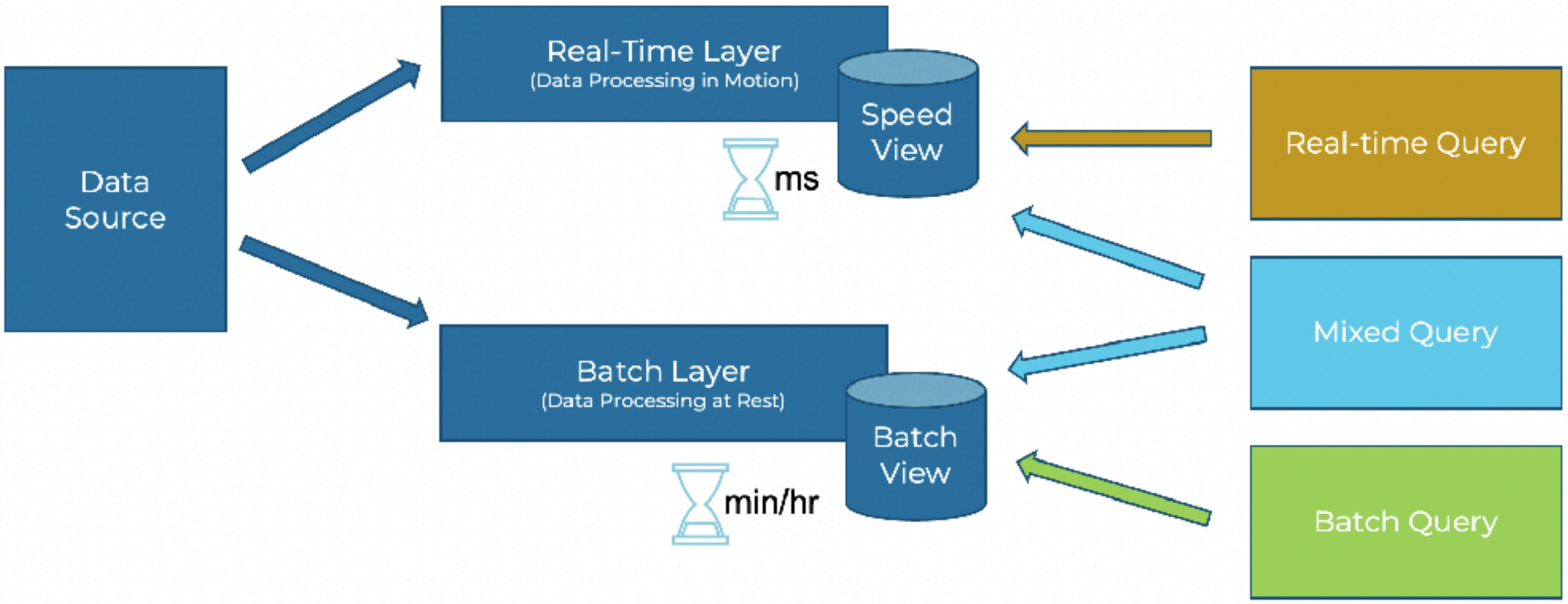
Data Architecture overview

Lambda Architecture



Lambda Architecture

Option 2: Separate serving layers



RAW DATA

DATA STORAGE

BATCH PROCESSING

PROCESSING DATA

BATCH LAYER

QUERY

DATA

VIEW 1

VIEW 2

VIEW 3

BATCH VIEWS

MERGED VIEW

PARTIAL AGGREGATIONS

REAL-TIME VIEWS

VIEW 1

VIEW 2

VIEW 3

SERVING LAYER

STREAM

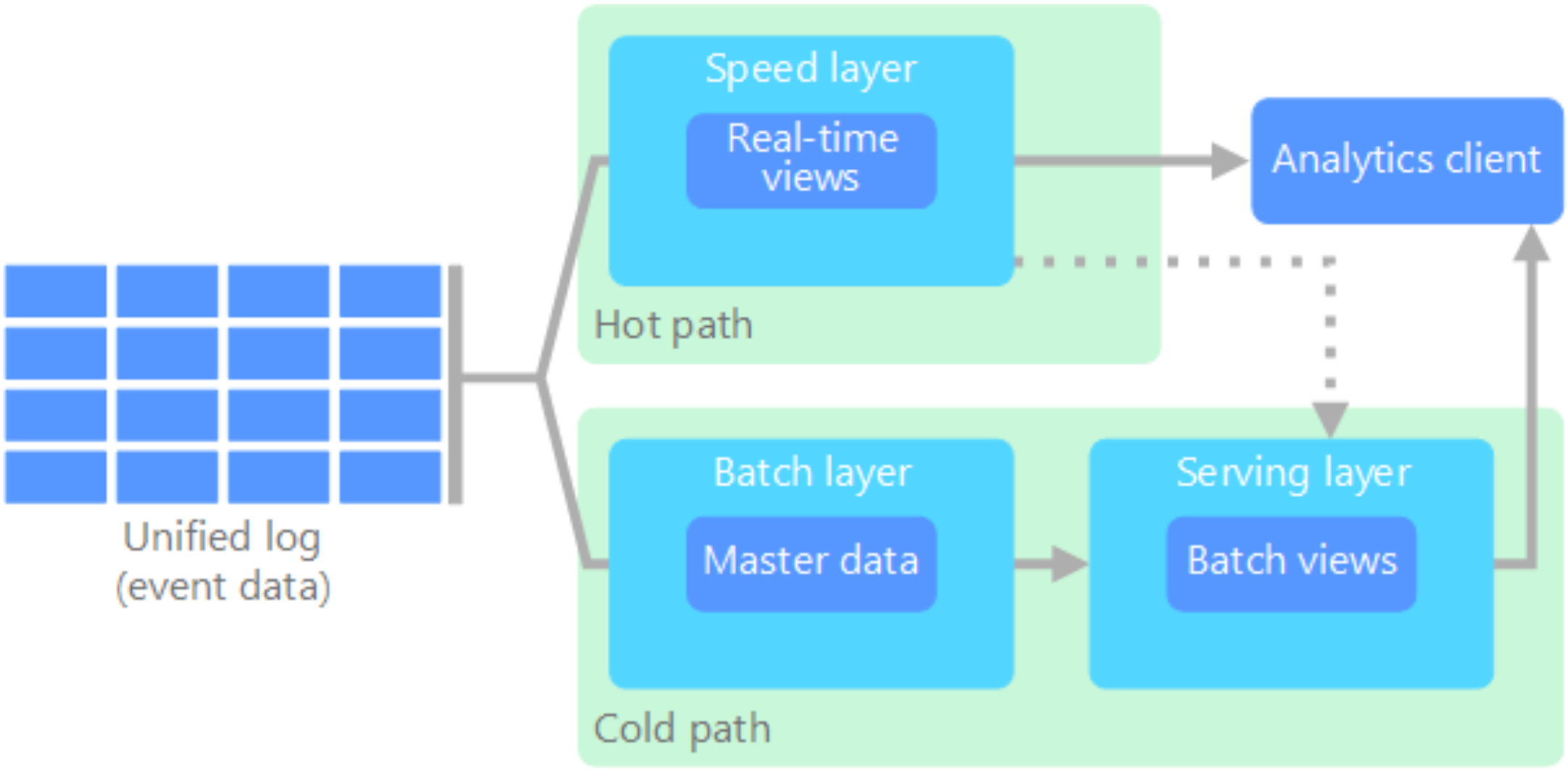
DATA INCREMENT

REAL-TIME PROCESSING

PROCESSED INCREMENT

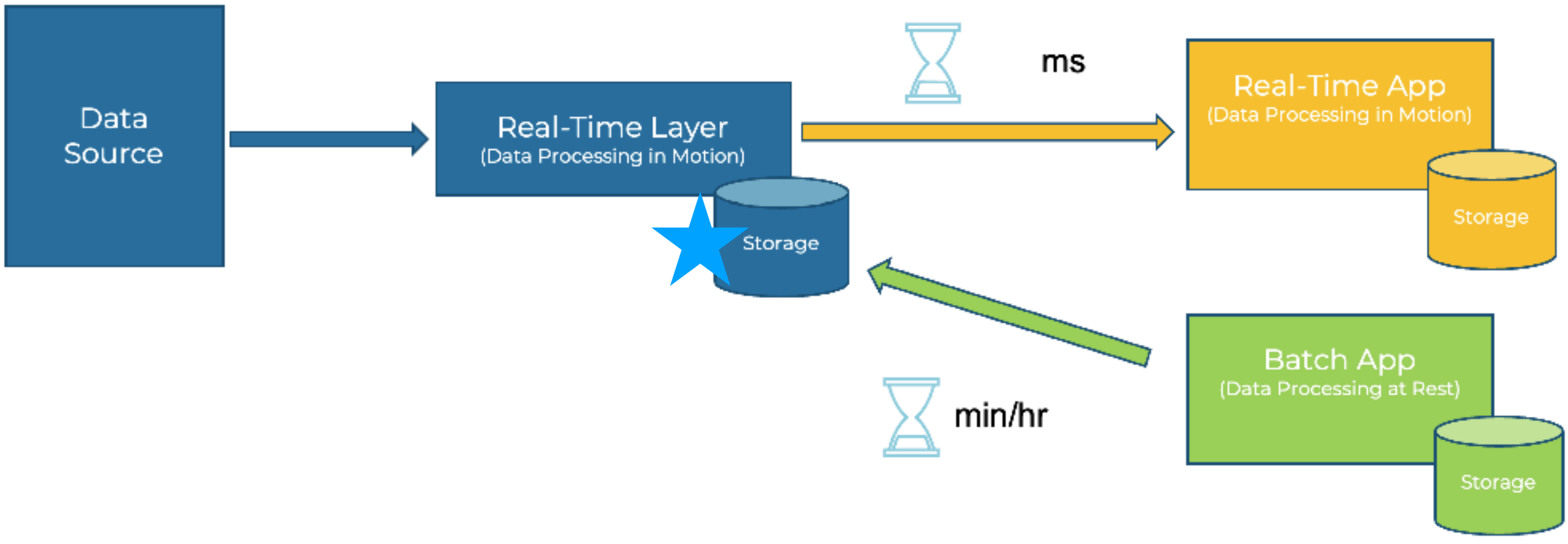
SPEED LAYER

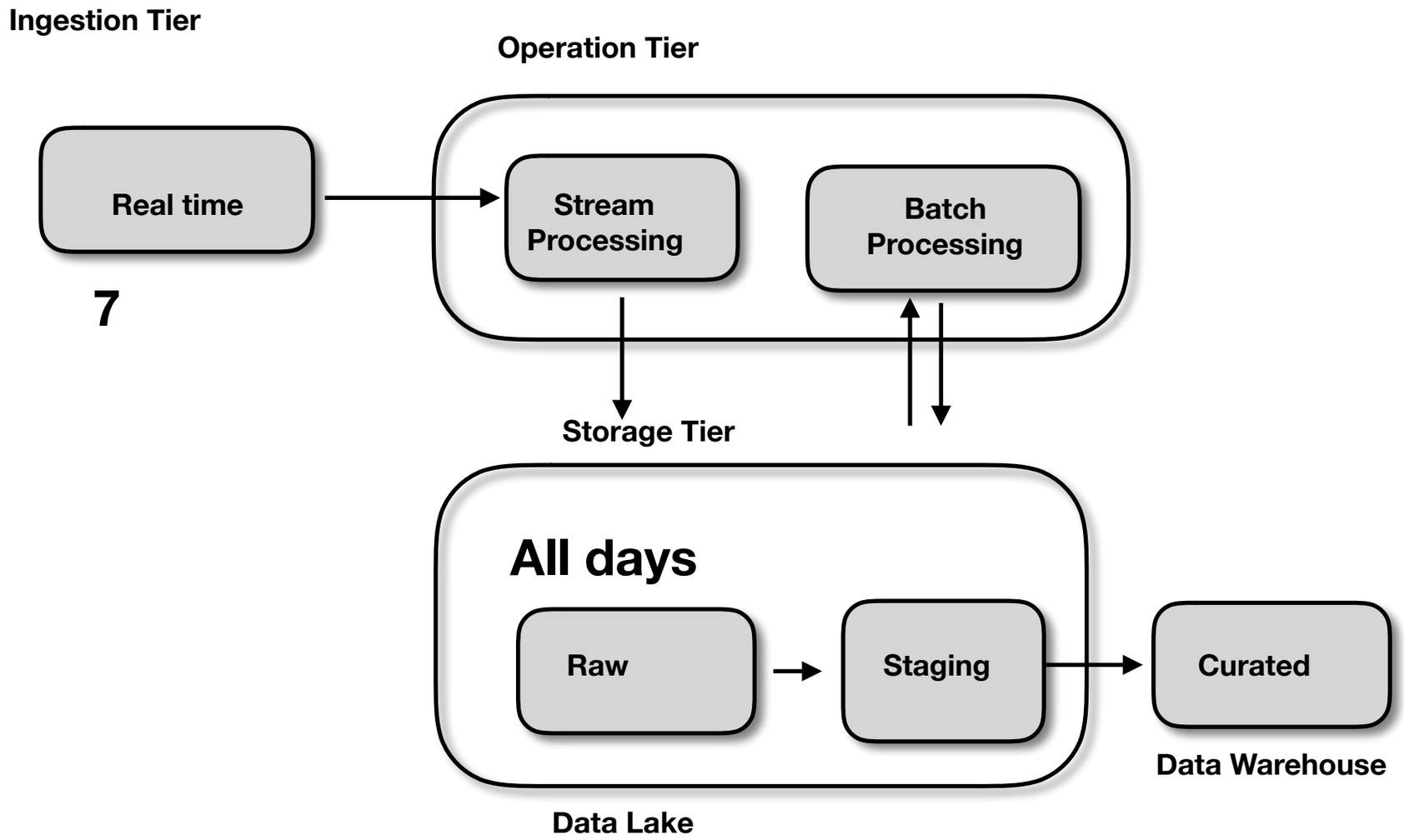
Lambda Architecture

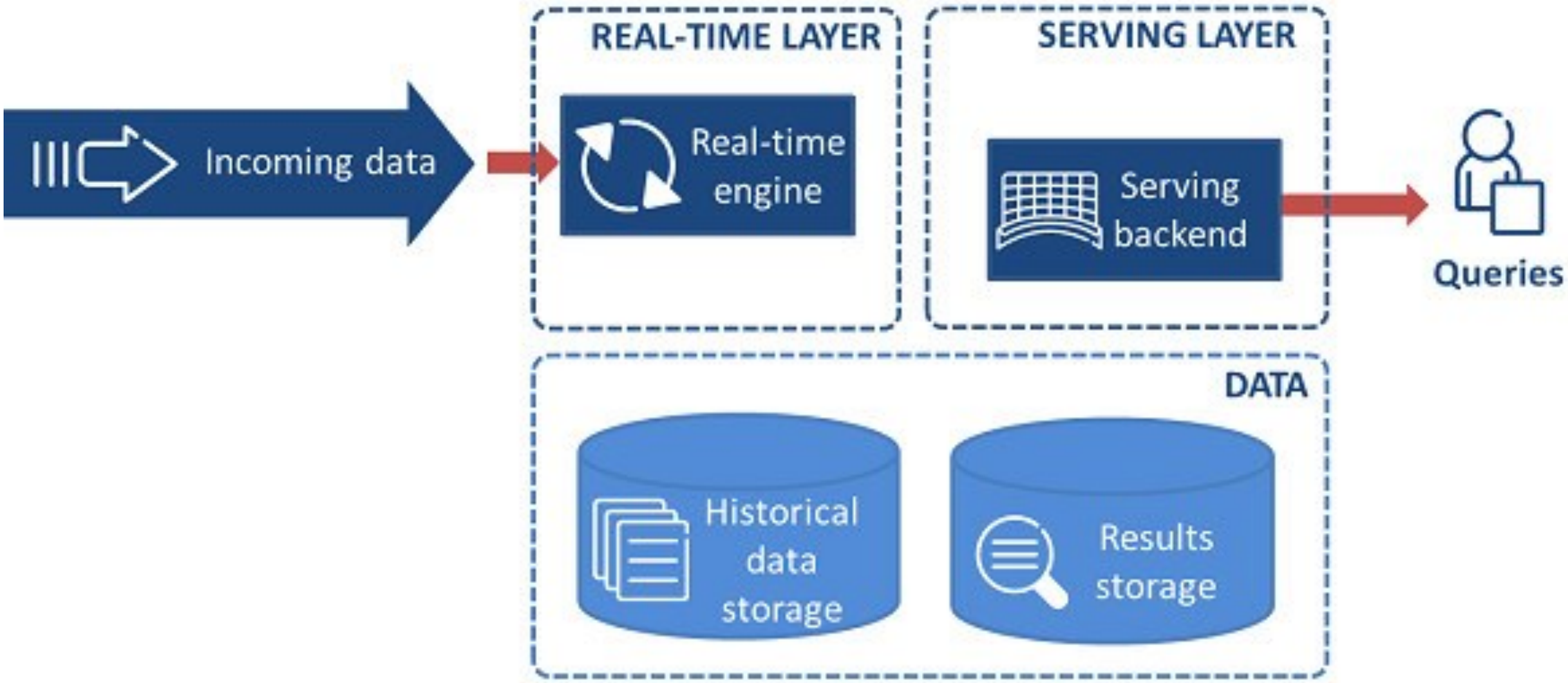


Kappa Architecture

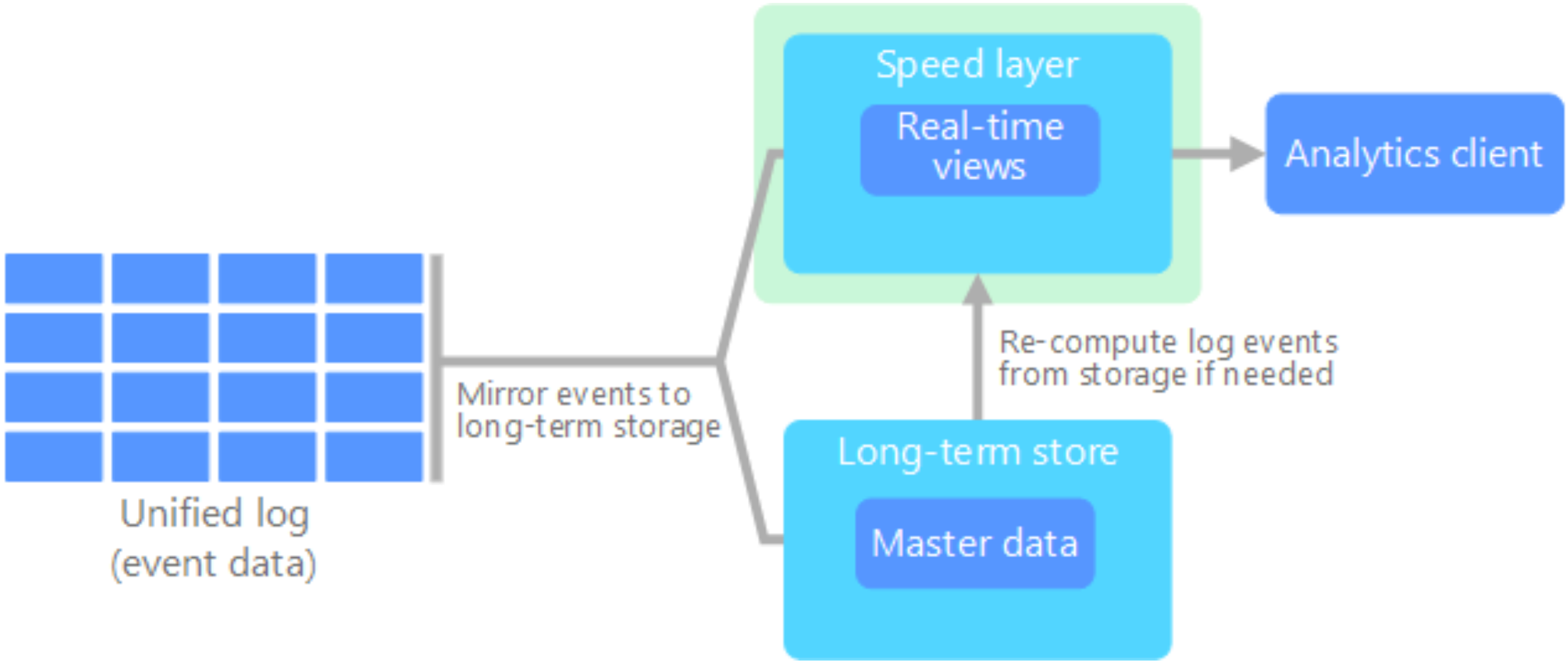
One pipeline for real-time and batch consumers




Ingestion Tier
Operation Tier
Real time
Stream Processing
Batch Processing
Storage Tier
All days
Raw
Staging
Curated
Data Warehouse
Data Lake


REAL-TIME LAYER
SERVING LAYER
Incoming data
Real-time engine
Serving backend
Queries
DATA
Historical data storage
Results storage

Kappa Architecture



SERVING LAYER

PARTIAL AGGREGATES

QUERY

REAL-TIME VIEWS

VIEW 1

VIEW 2

VIEW 3

DATA

STREAM

RAW DATA

IMMUTABLE DATA

LOG

REAL-TIME PROCESSING

PROCESSED DATA

SPEED LAYER

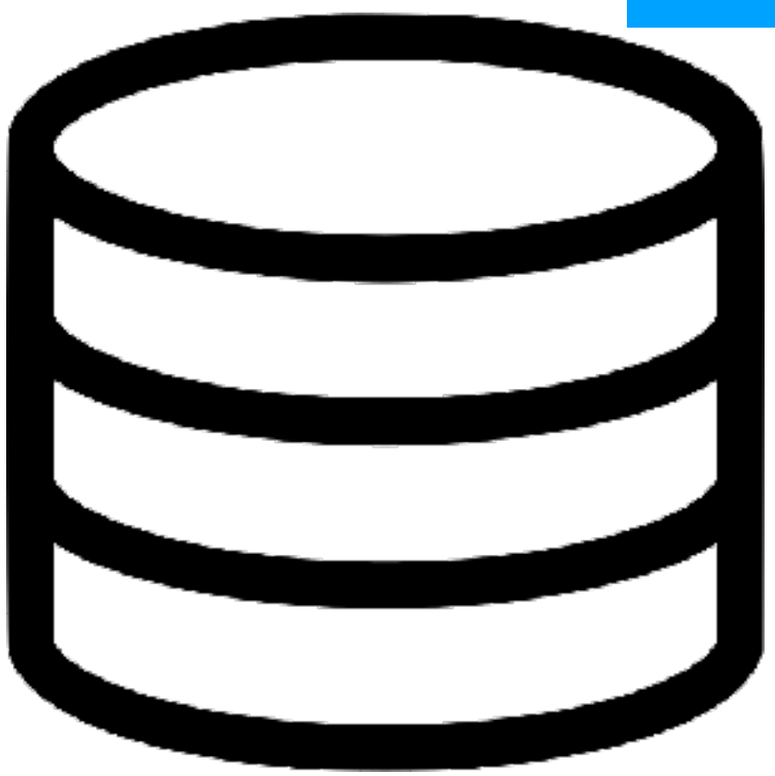
Stream

Analytical Code

Storage for last 7 days data

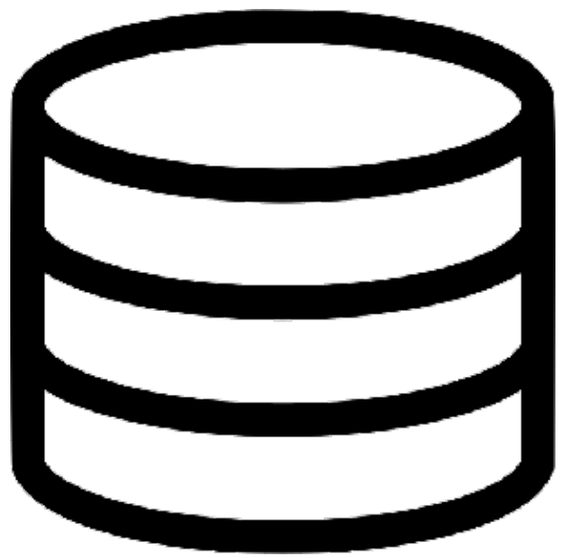
Analytical Code

Stream



Storage for last 300 days data

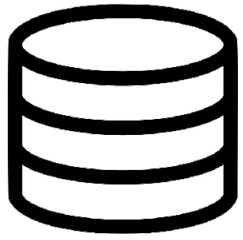
Data Lake



Analytical Code

Storage for last 300 days data

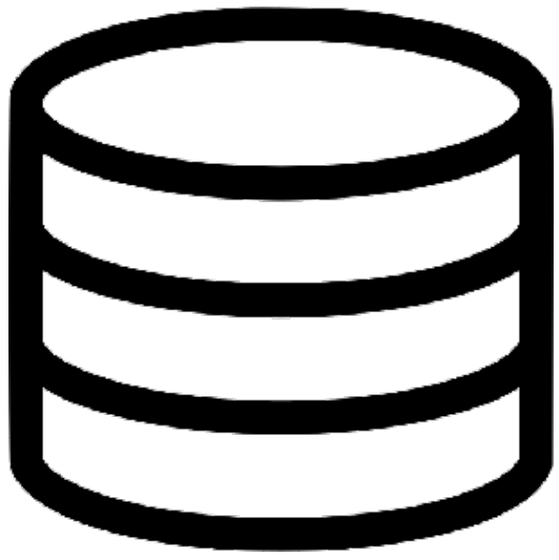
Stream



Analytical Code

Storage for last 7 days data

Data Lake



Analytical Code

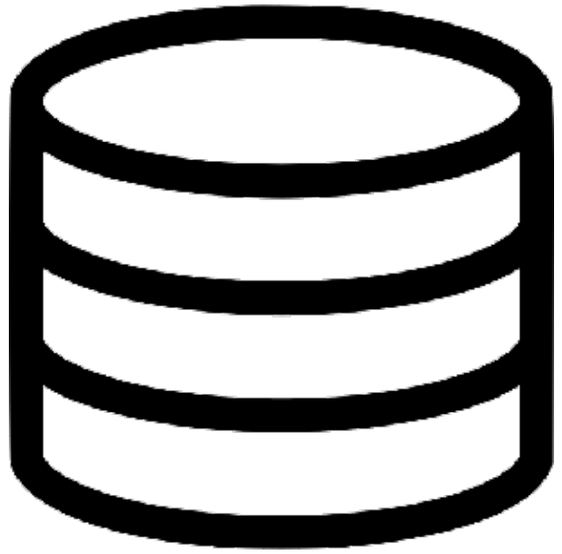
Storage for last 300 days data

Stream

Analytical Code

Storage for last 7 days data

Data Lake



Analytical Code

Storage for last 300 days data

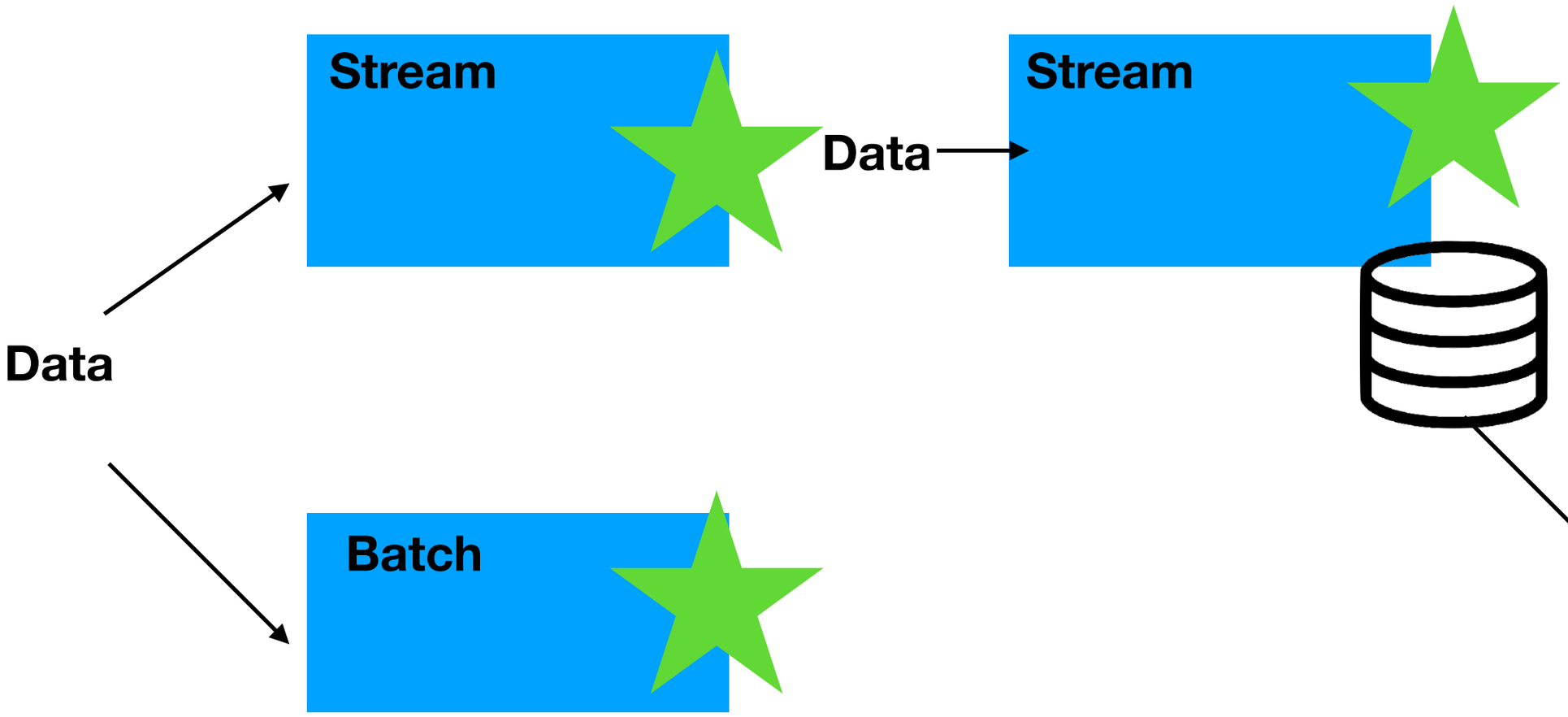
What Architecture to Choose?

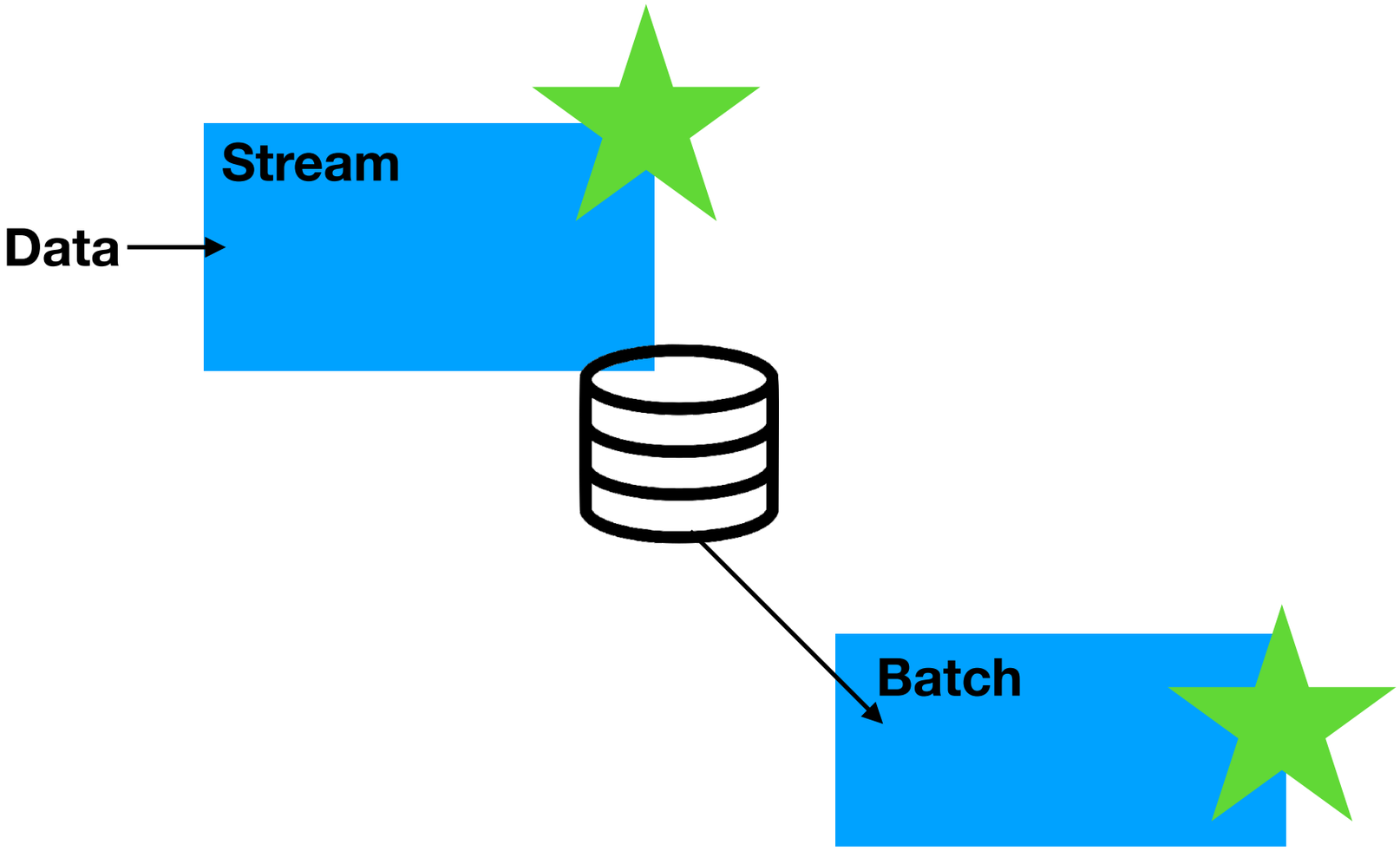
Kappa
Real-Time
Data in Motion

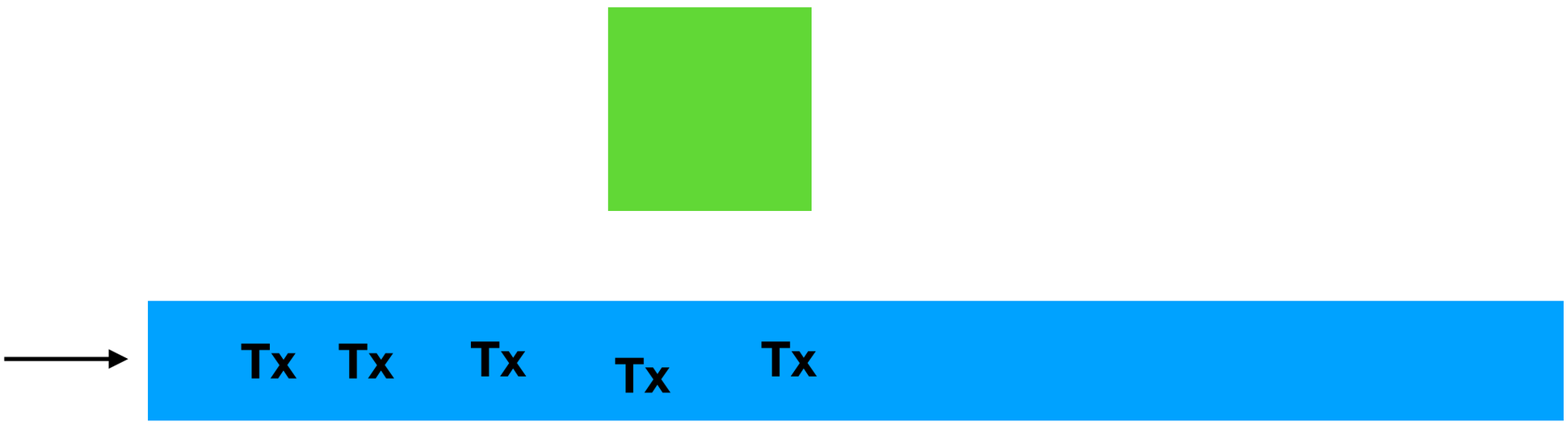
VS

Lambda
Batch
Data at Rest

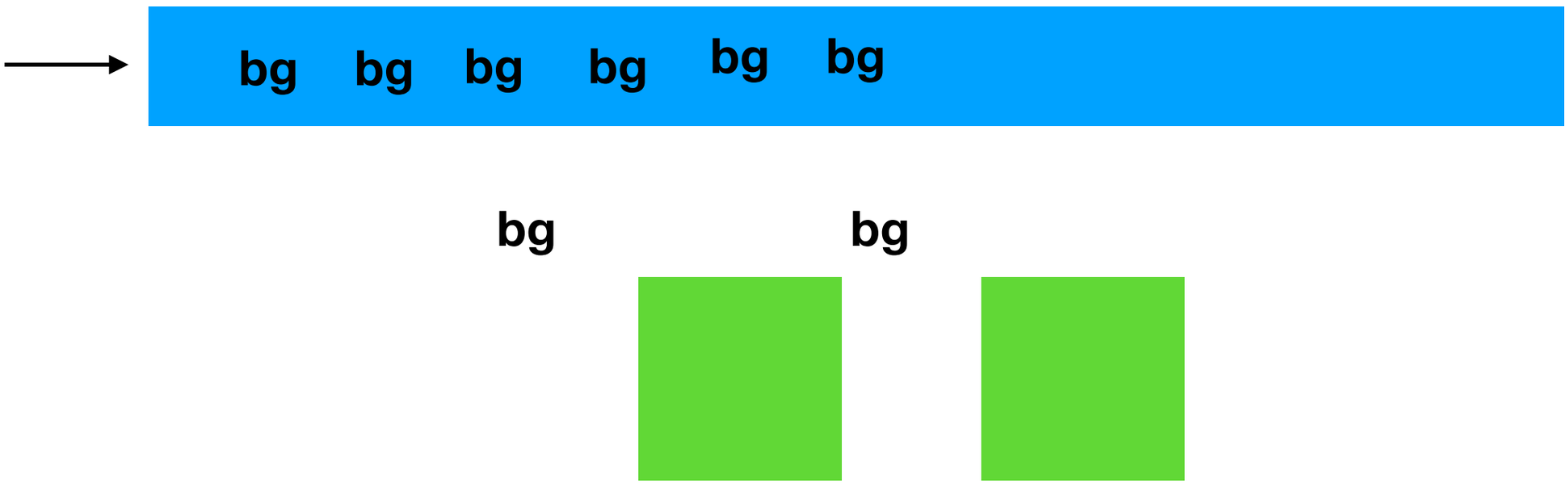
KAI WAEHNER

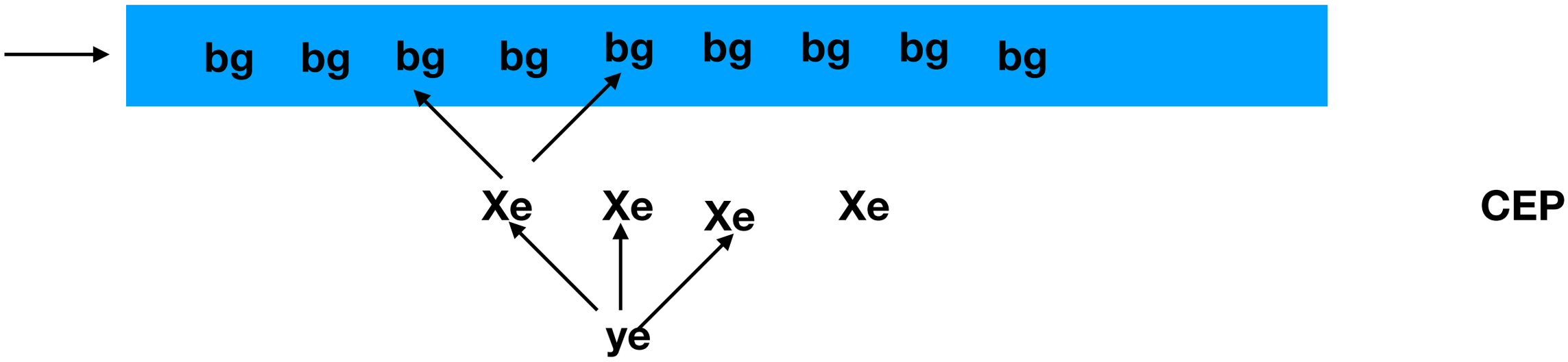

Stream
Data
Stream
Data
Batch
Batch

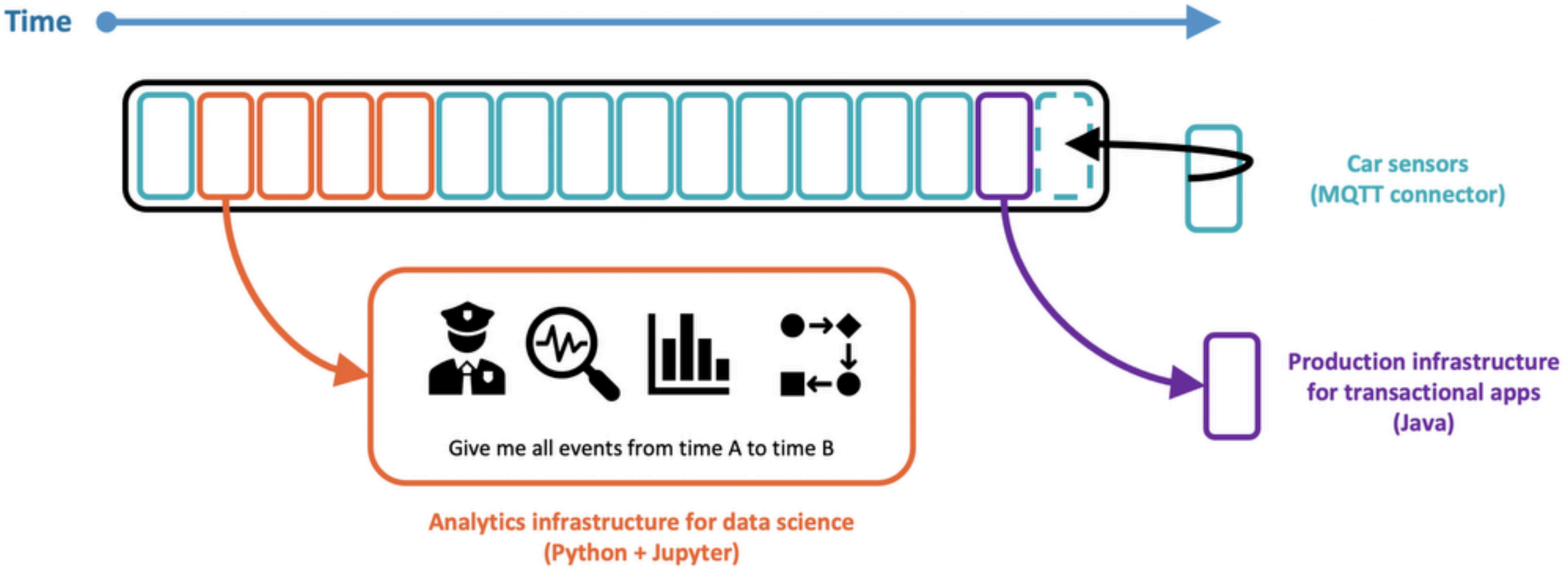

Stream
Data
Batch



2 tx, same employee, in < 5 minutes on same account
Tx by an employee on an acc from a different regions




bg bg bg bg bg bg bg bg bg
Xe Xe Xe
Xe
ye
CEP
DROOLS
ne



Kappa architectures enable transactional workloads in addition to analytical workloads too.
A single pipeline for everything. No need for a Lambda architecture! Kappa enables transactional and analytical workloads.

Stream

LOB

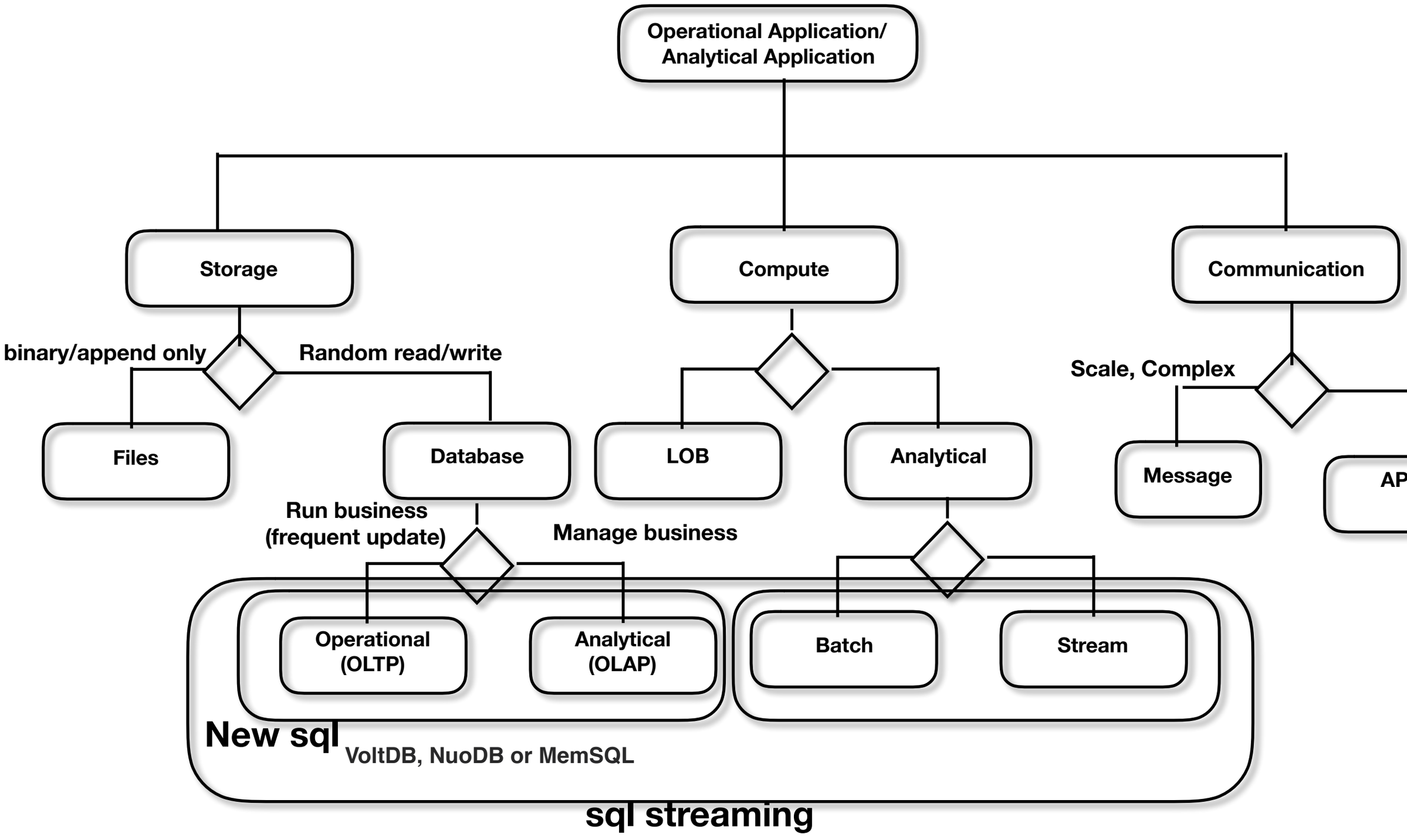
Analytics

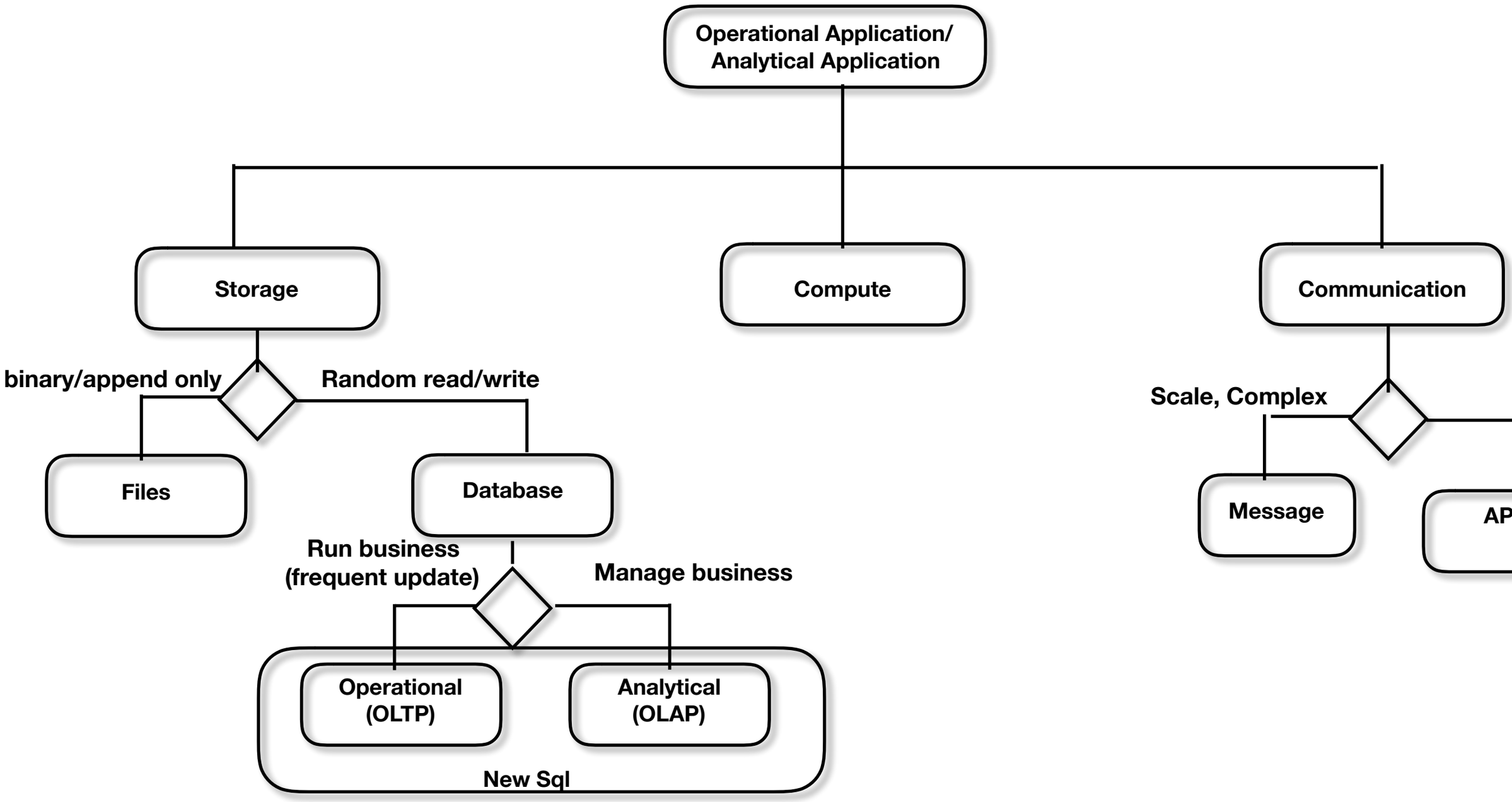
Stream

Batch

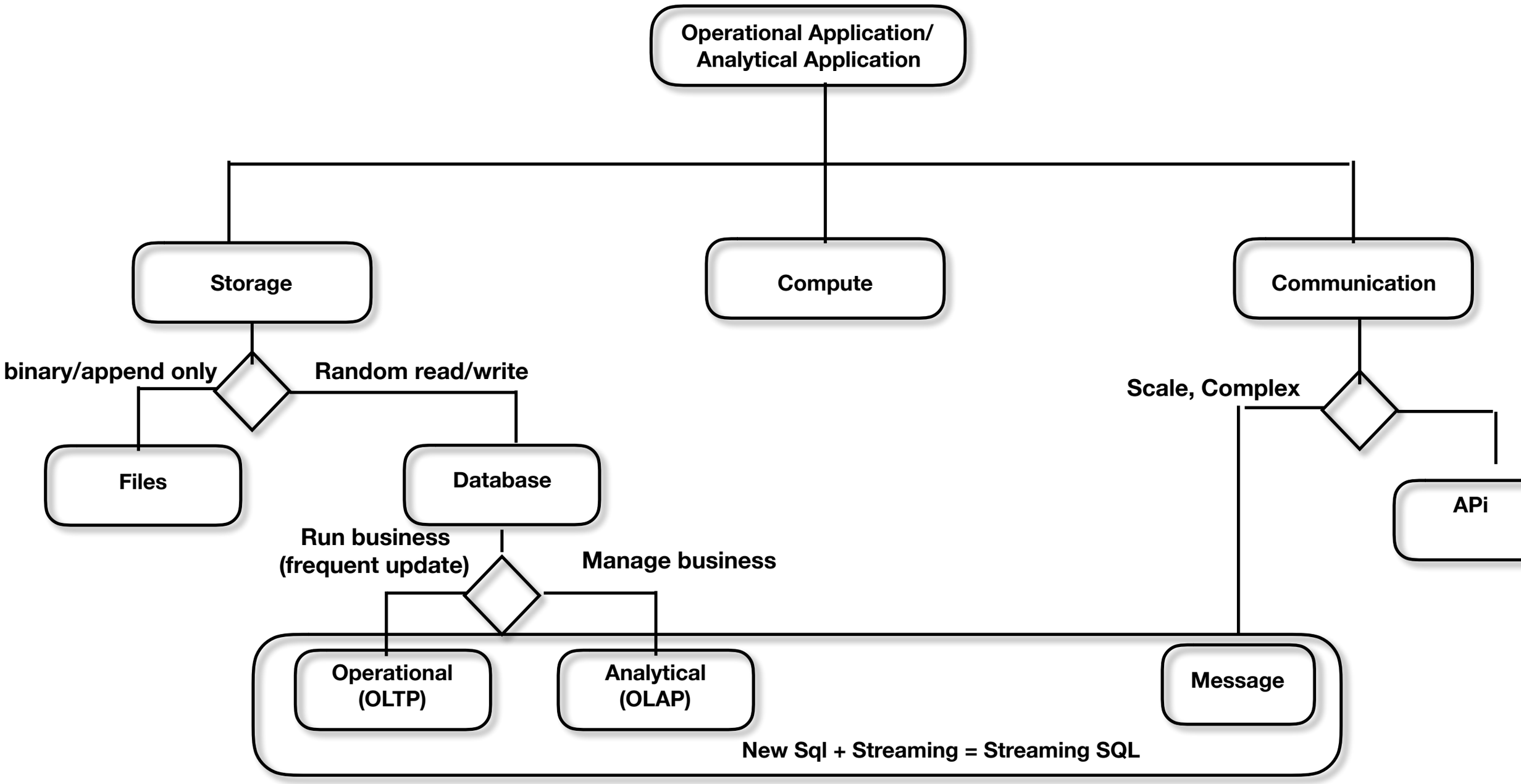
Spark itself doesn't manage these machines. It needs a cluster manager (also sometimes called *scheduler*)

- Standalone: Simple cluster-manager, limited in features, incorporated with Spark.
- Apache Mesos.
- Hadoop YARN
- Kubernetes

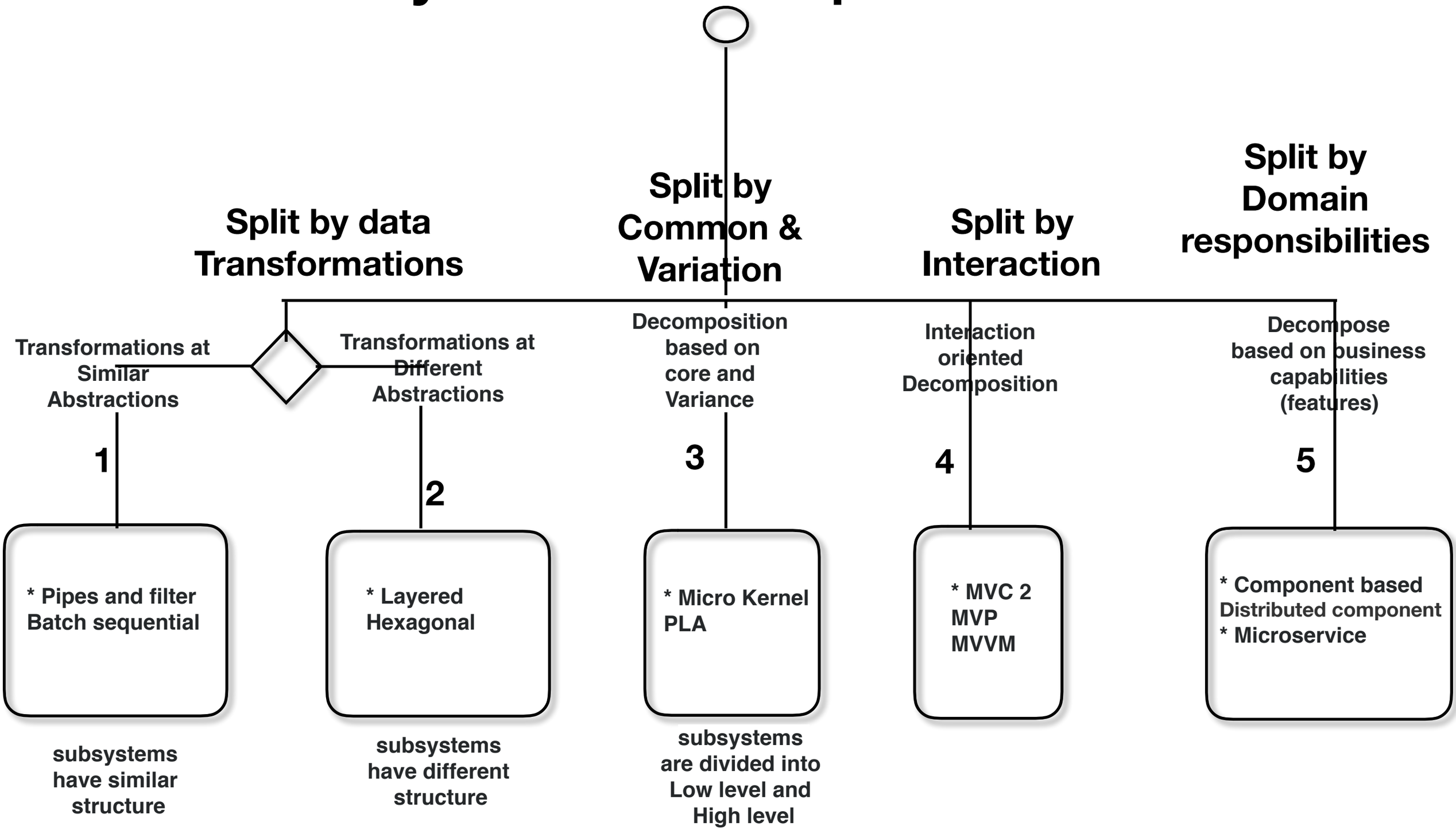

Operational Application/
Analytical Application
Storage
Compute
Communication
binary/append only
Random read/write
Files
Database
LOB
Analytical
Scale, Complex
Message
APi
Run business
(frequent update)
Manage business
Operational
(OLTP)
Analytical
(OLAP)
Batch
Stream
New sql
VoltDB, NuoDB or MemSQL
sql streaming



VoltDB, NuoDB or MemSQL


Operational Application/
Analytical Application
Storage
Compute
Communication
binary/append only
Random read/write
Files
Database
Run business
(frequent update)
Manage business
Operational
(OLTP)
Analytical
(OLAP)
Scale, Complex
APi
Message
New Sql + Streaming = Streaming SQL

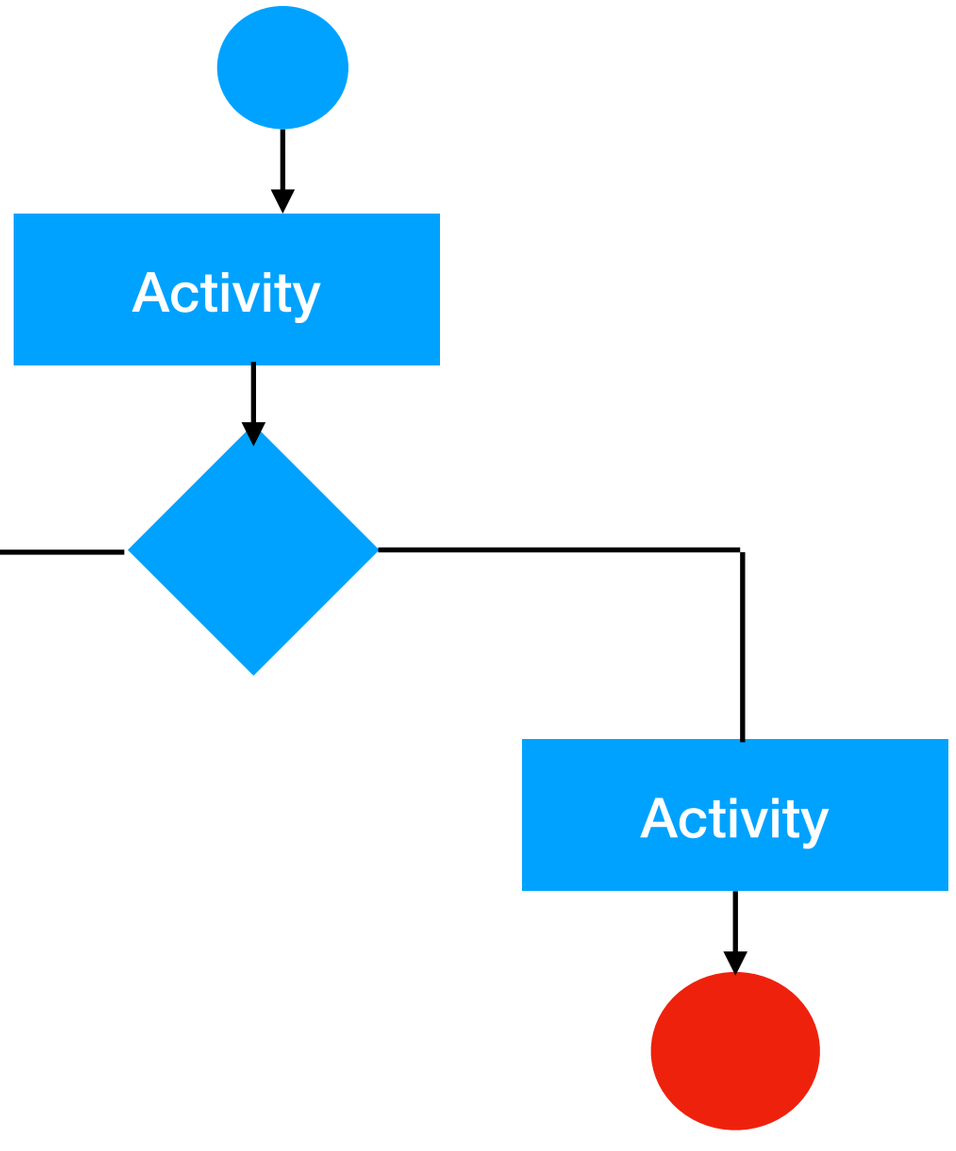
Choose System Decomposition Patterns

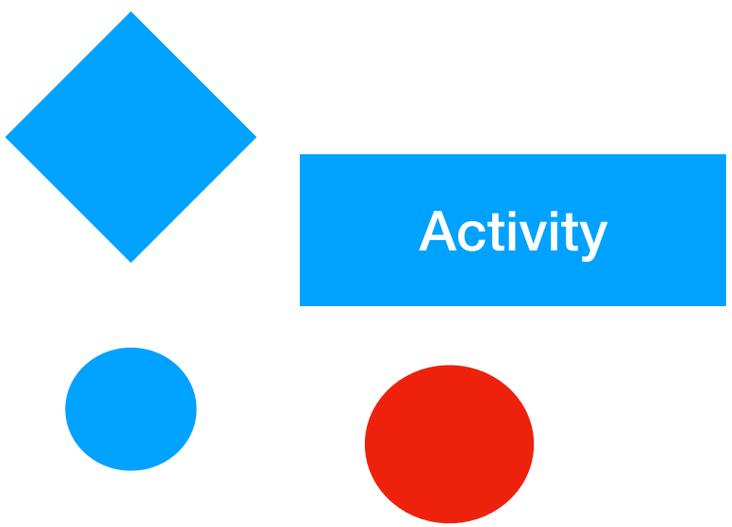


Eclipse IDE. Downloading the basic Eclipse product provides you little more than a fancy editor. However, once you start adding plug-ins, it becomes a highly customizable and useful product.

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.

A workflow implementation. The implementation of a workflow contains concepts like the order of the different steps, evaluating the results of steps, deciding what the next step is, etc.





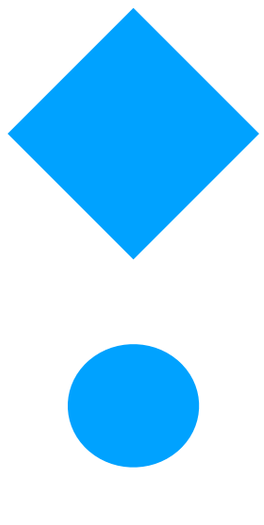
A task scheduler. A scheduler contains all the logic for scheduling and triggering tasks

Internet browses plug-ins add additional capabilities that are not otherwise found in the basic browser

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.

Core

Pipes and filter

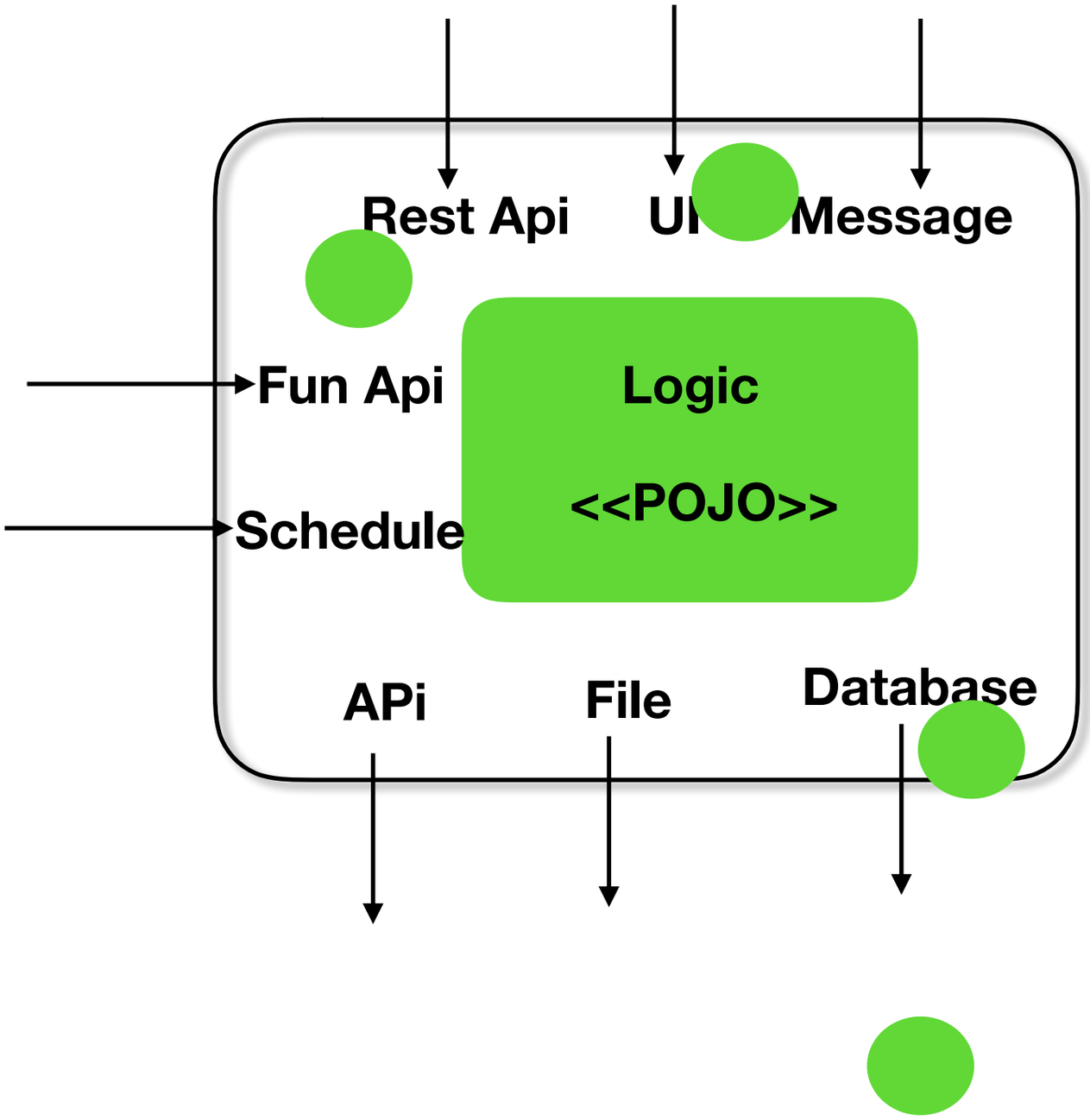


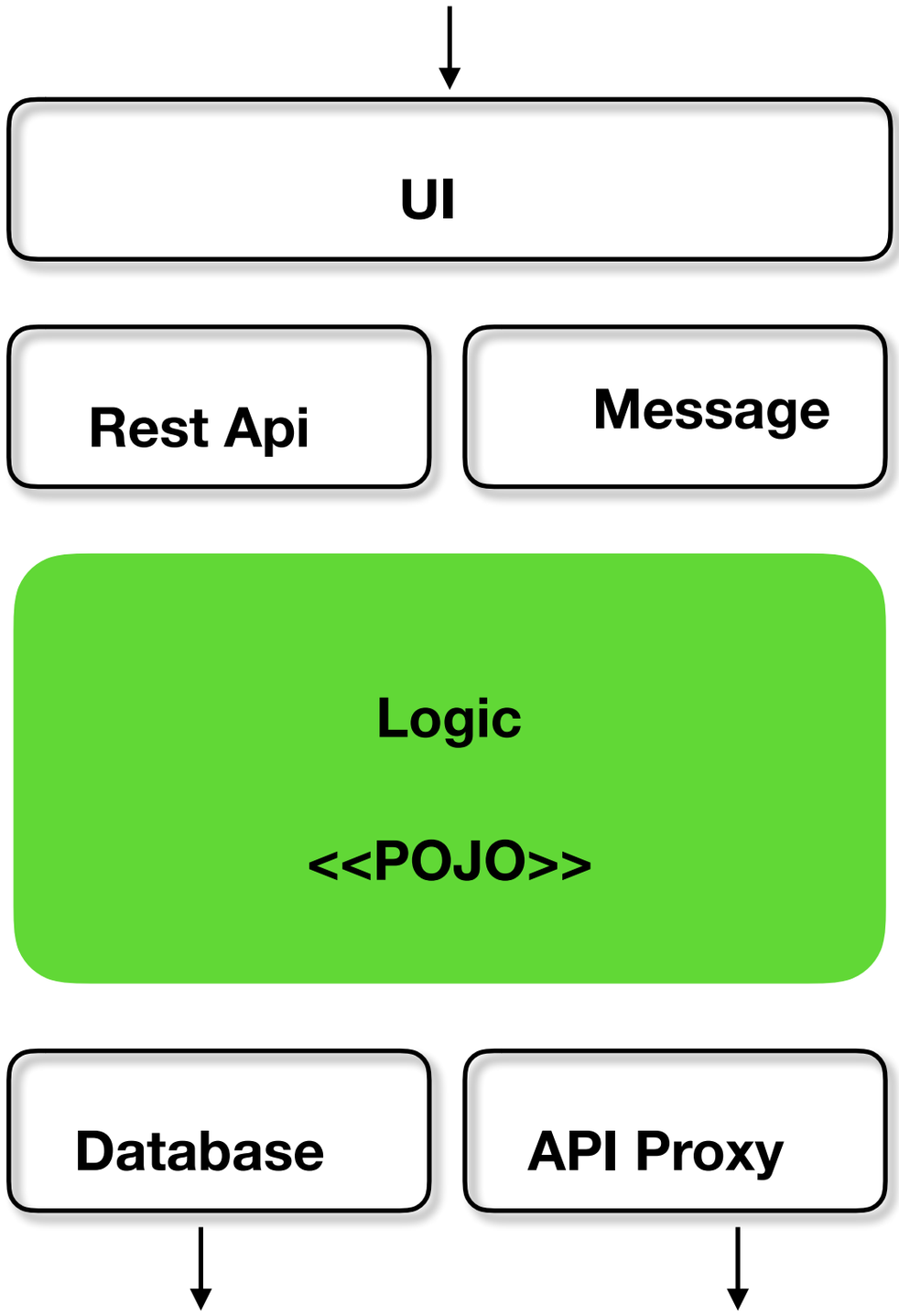
Activity holder

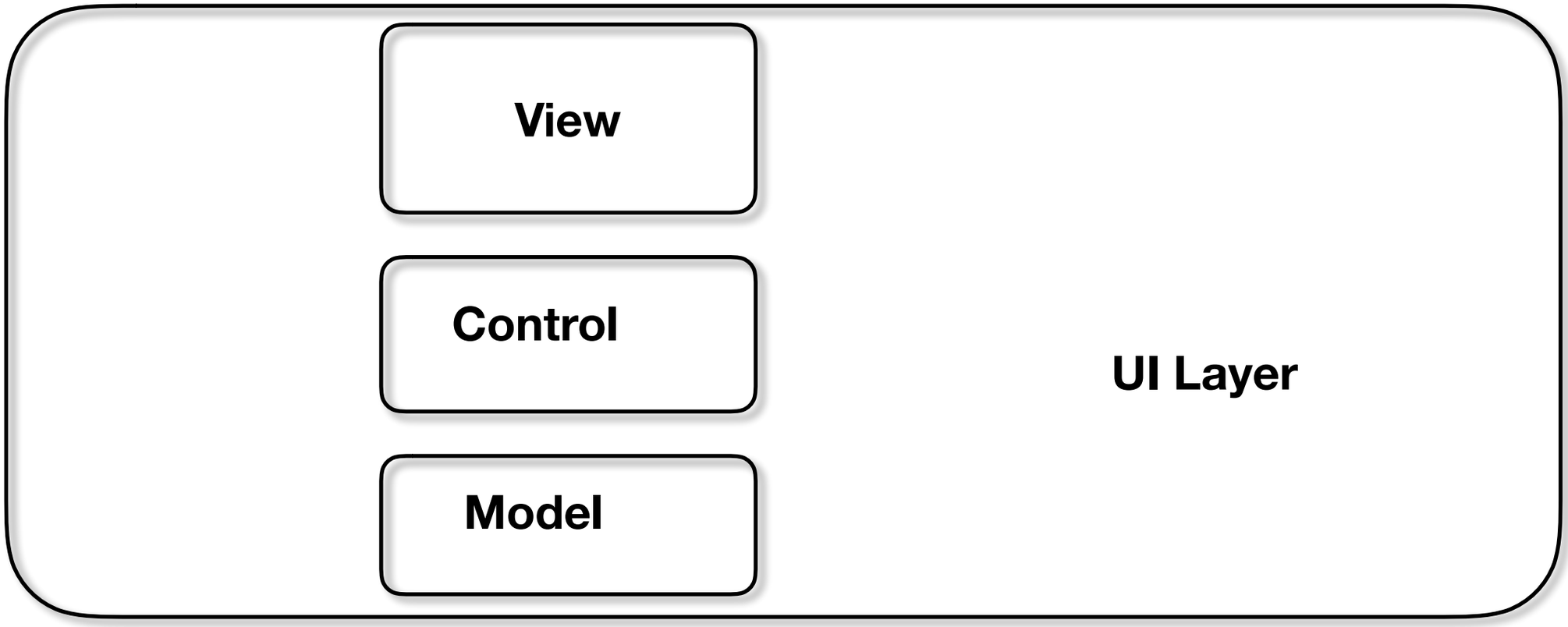
Activity1

Activity2

Activity3


Rest Api
UI
Message
Fun Api
Logic
<<POJO>>
Schedule
APi
File
Database


UI
Rest Api
Message
Logic
<<POJO>>
Database
API Proxy



Api

UI

Rest Api

Message

Facade

Logic

<<POJO>>

DAL

API Proxy

Layered

UI

Update order

Schedule

Rest Api

Message

Delegate

flow Logic , orchestration Logic

Application

DAL

API Proxy

Domain Logic

<<POJO>>

Hexagonal

Invoke logic()

Data1

Layer 1

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

Data2

Layer 2

f3(int,int)

f1(bool) f4()

f2(double)

Data3

Layer 3

f1() f4()

f2(int) f3() f5(int)

Data4

Layer 4

f1(int) f2()

Data5

Process Data()

Data1

Pipe

Filter 1

f1(int)

Data2

Filter 2

f2(int)

Data3

Filter 3

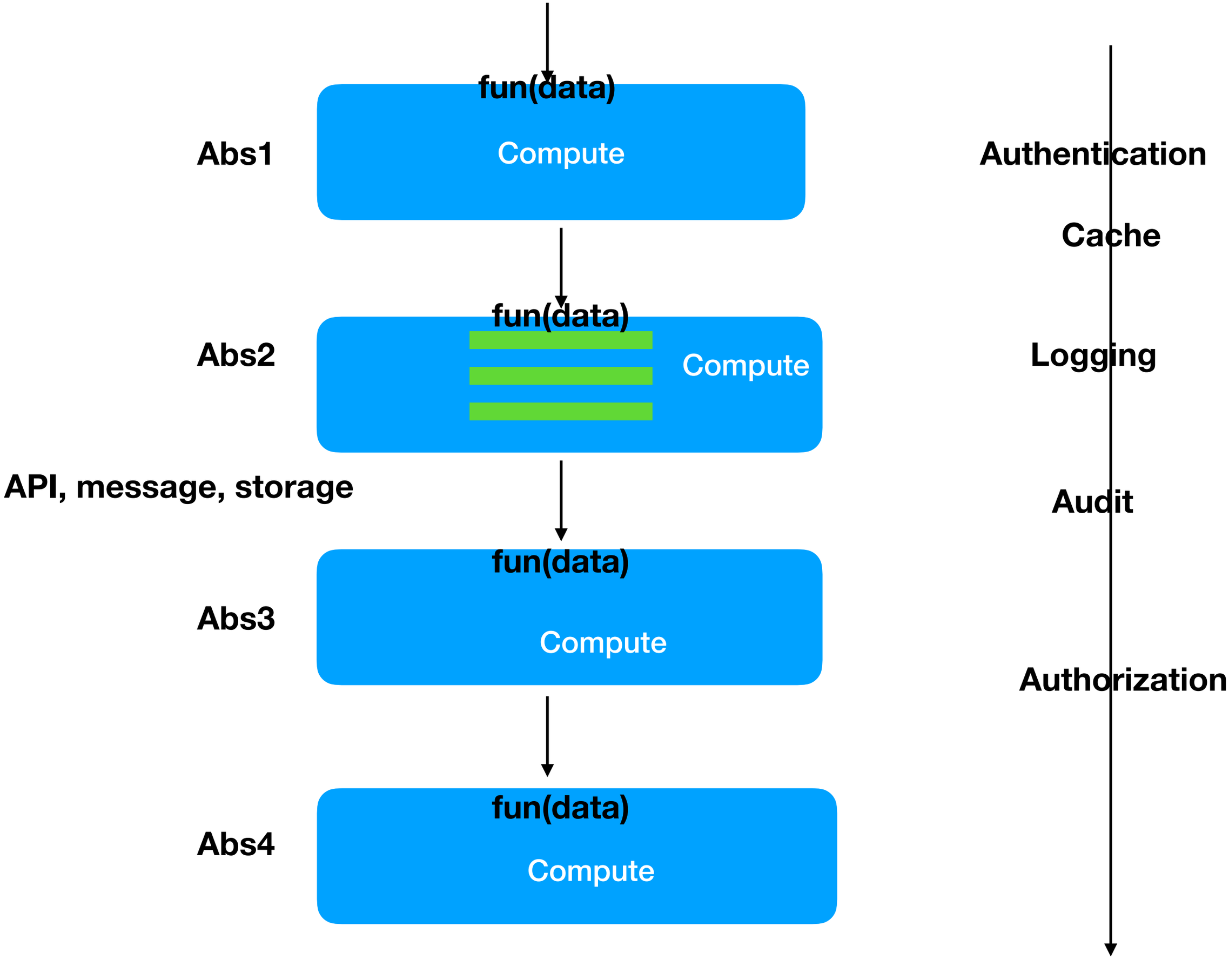
f3(int)

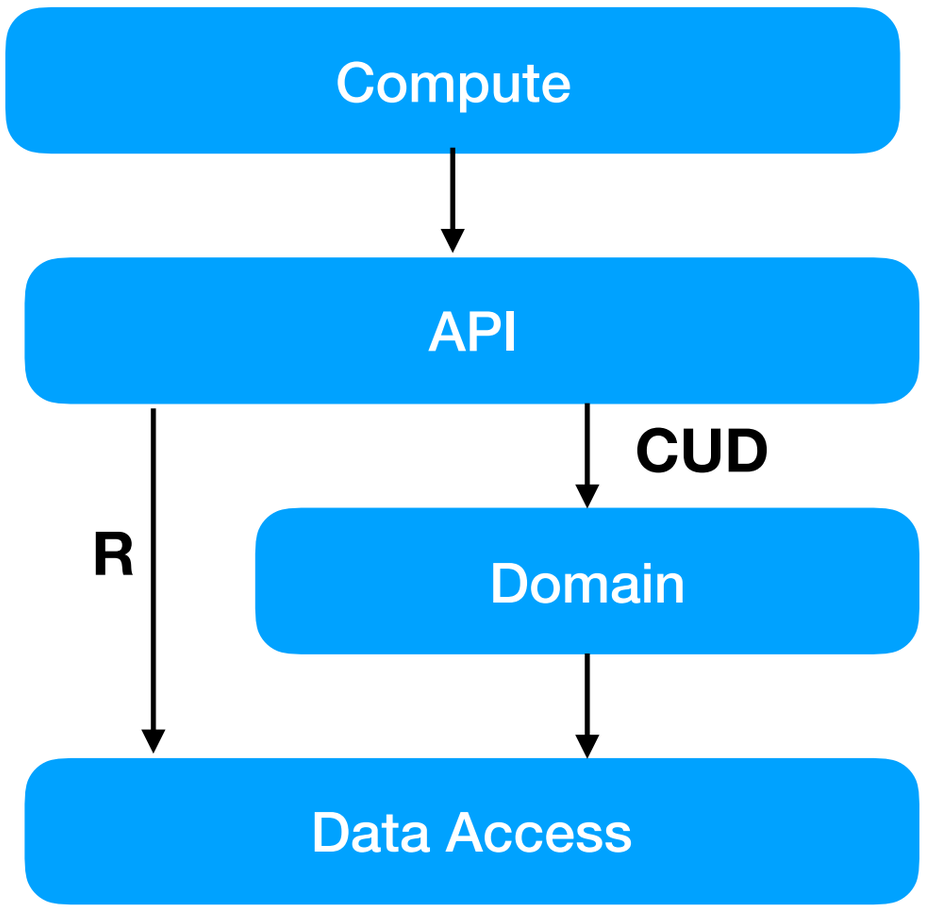
Data4

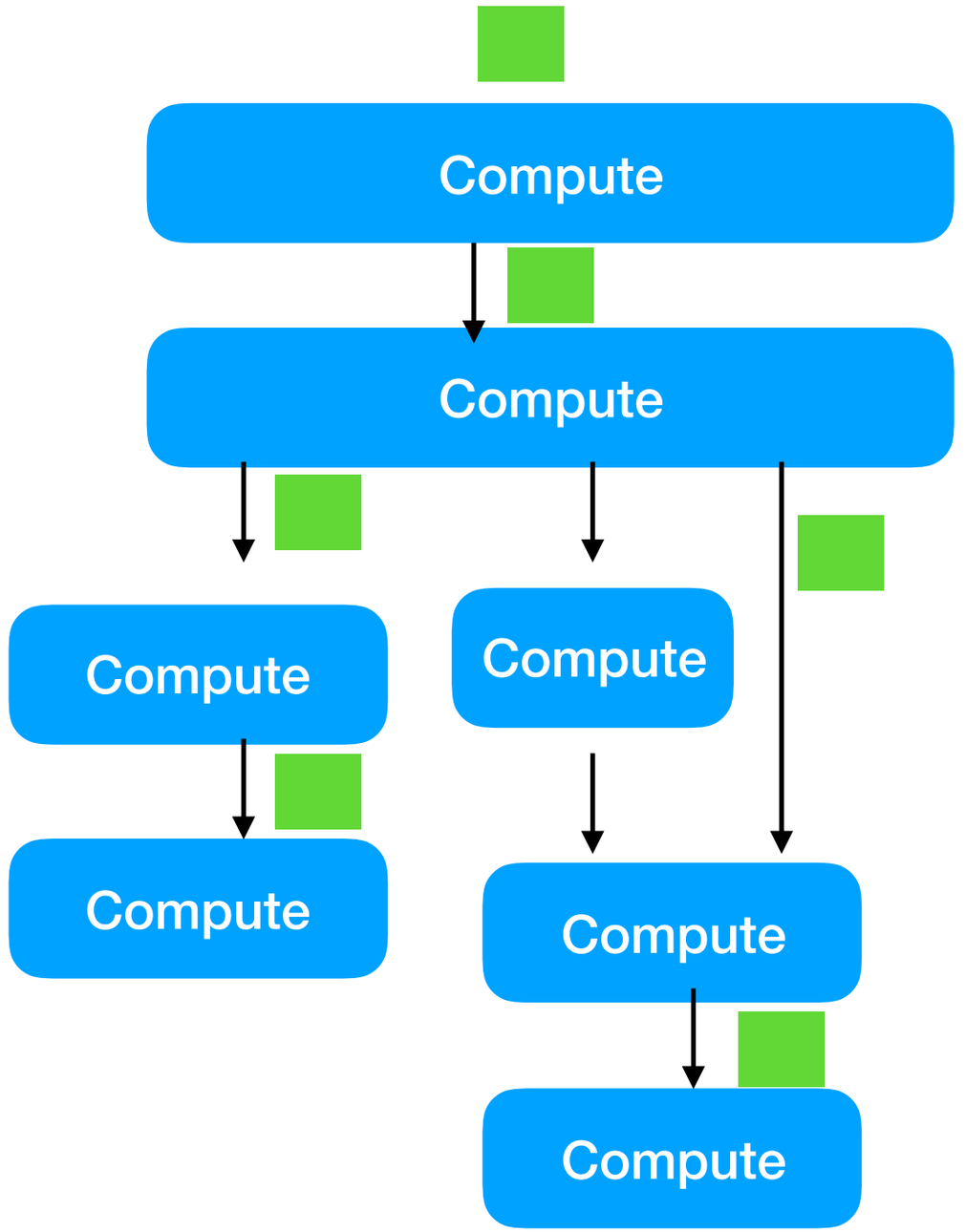
Filter 4

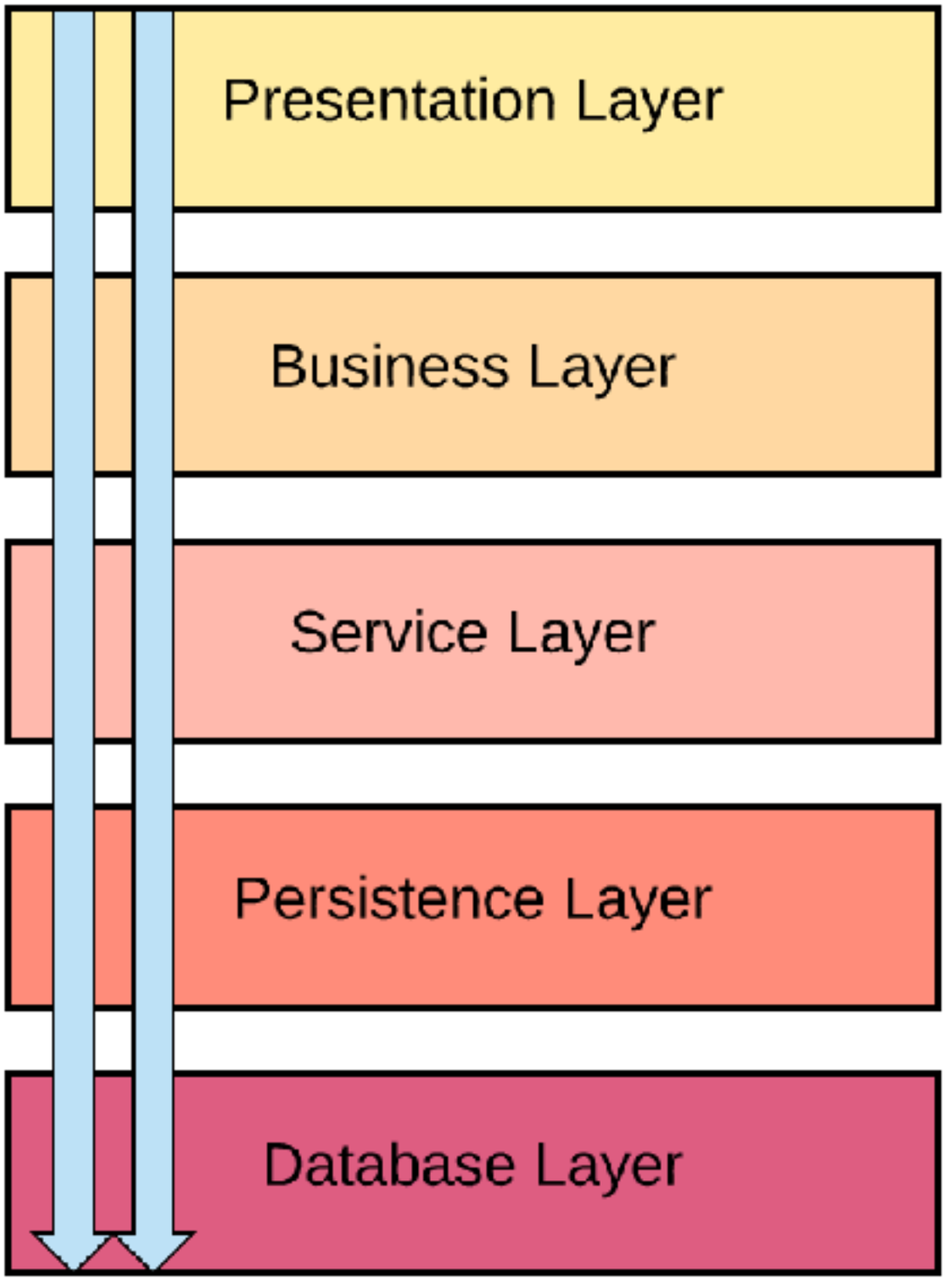
f4(int)

Data5

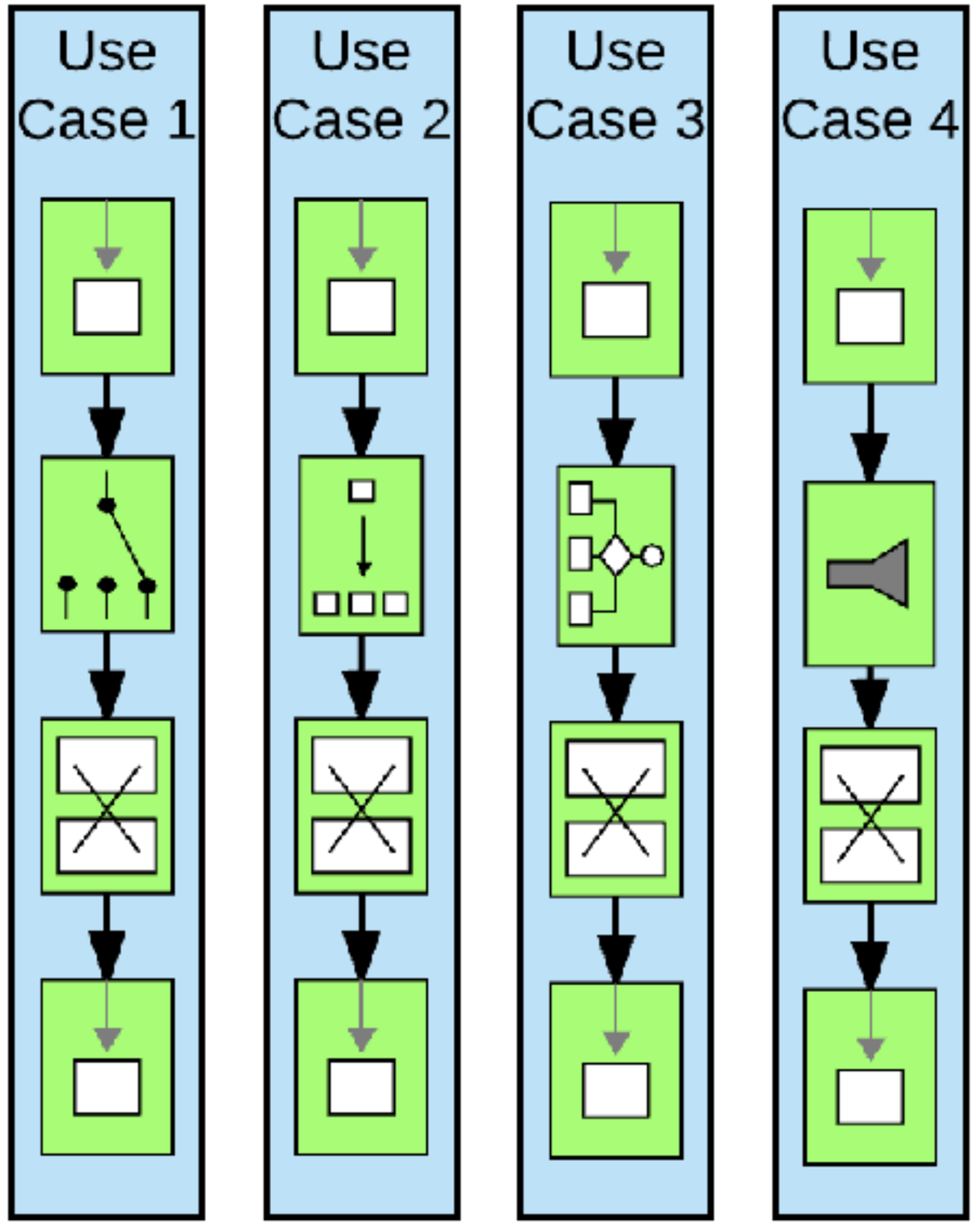

fun(data)
Abs1
Compute
fun(data)
Abs2
Compute
API, message, storage
fun(data)
Abs3
Compute
fun(data)
Abs4
Compute
Authentication
Cache
Logging
Audit
Authorization


Compute
API
CUD
R
Domain
Data Access

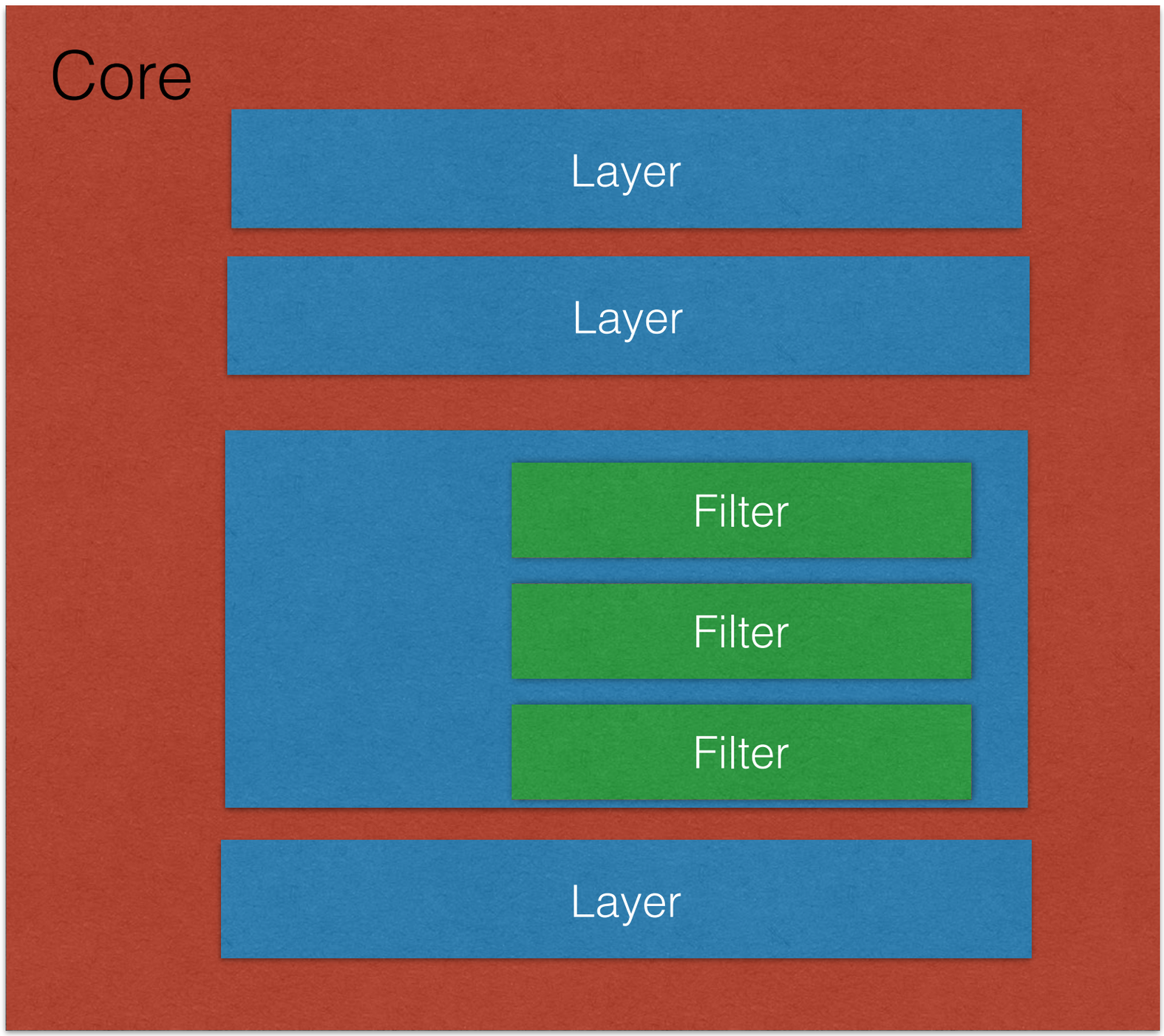

Compute
Compute
Compute
Compute
Compute
Compute
Compute



Layered Architecture



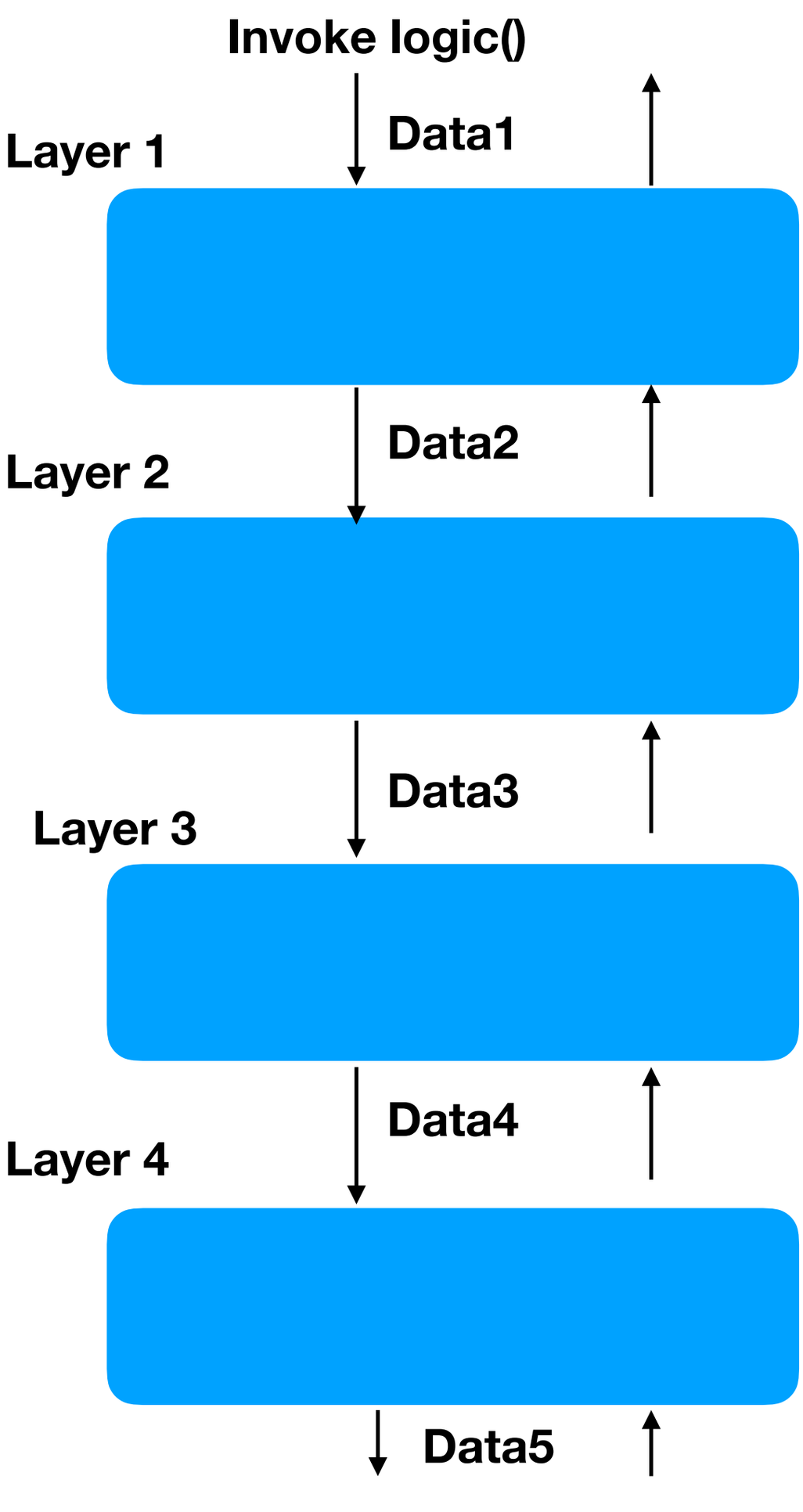
Pipes and Filters

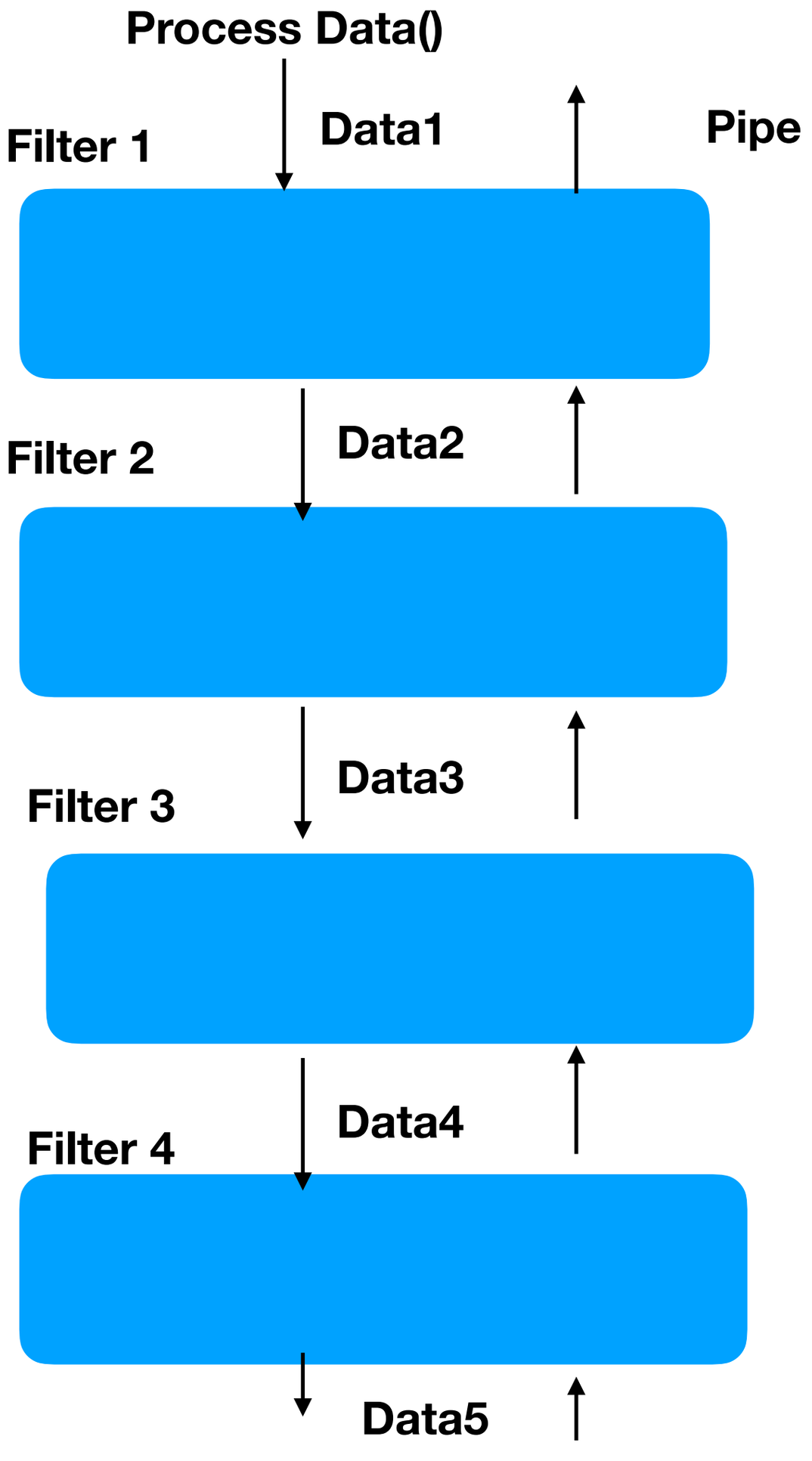


Plugins

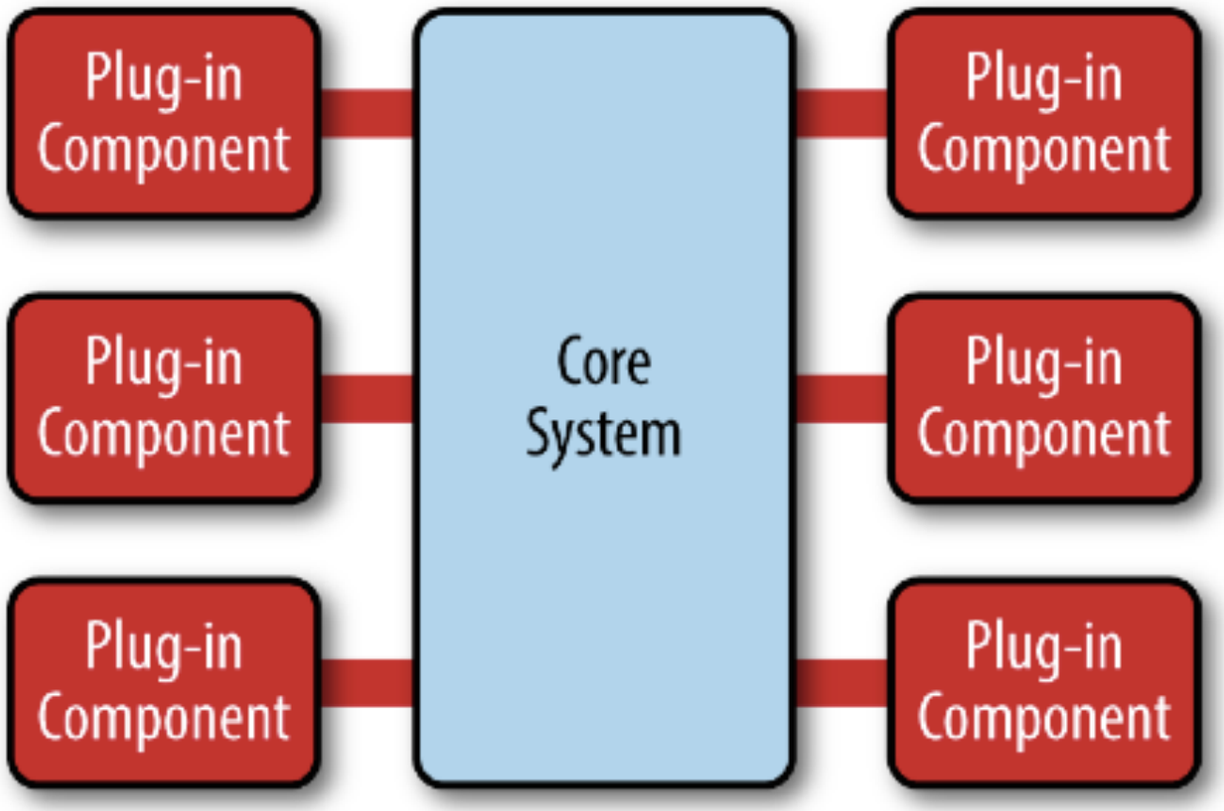
Plugins

Plugins


Invoke logic()
Layer 1
Data1
Data2
Layer 2
Data3
Layer 3
Data4
Layer 4
Data5


Process Data()
Filter 1
Data1
Pipe
Filter 2
Data2
Filter 3
Data3
Filter 4
Data4
Data5

Microkernel Architecture



1

Ship one binary

Application
Exe

Source file

Source file

Compile time
monolithic

2

Ship one binary

Application
Exe

Lib binaries

Lib binaries

Source file

Link time
monolithic

3

Ship multiple binary

Process

Application
Exe

Dll/so/jar

Dll/so/jar

Dll/so/jar

Runtime
monolithic

In process
comp

Ship multiple binary & host in multiple process

4

Process

Exe

Dll/so/jar

Distributed
comp

Process

Exe

Dll/so/jar

Distributed
comp

Distributed Application

5

Process

Exe

Dll/so/jar

Application1

Process

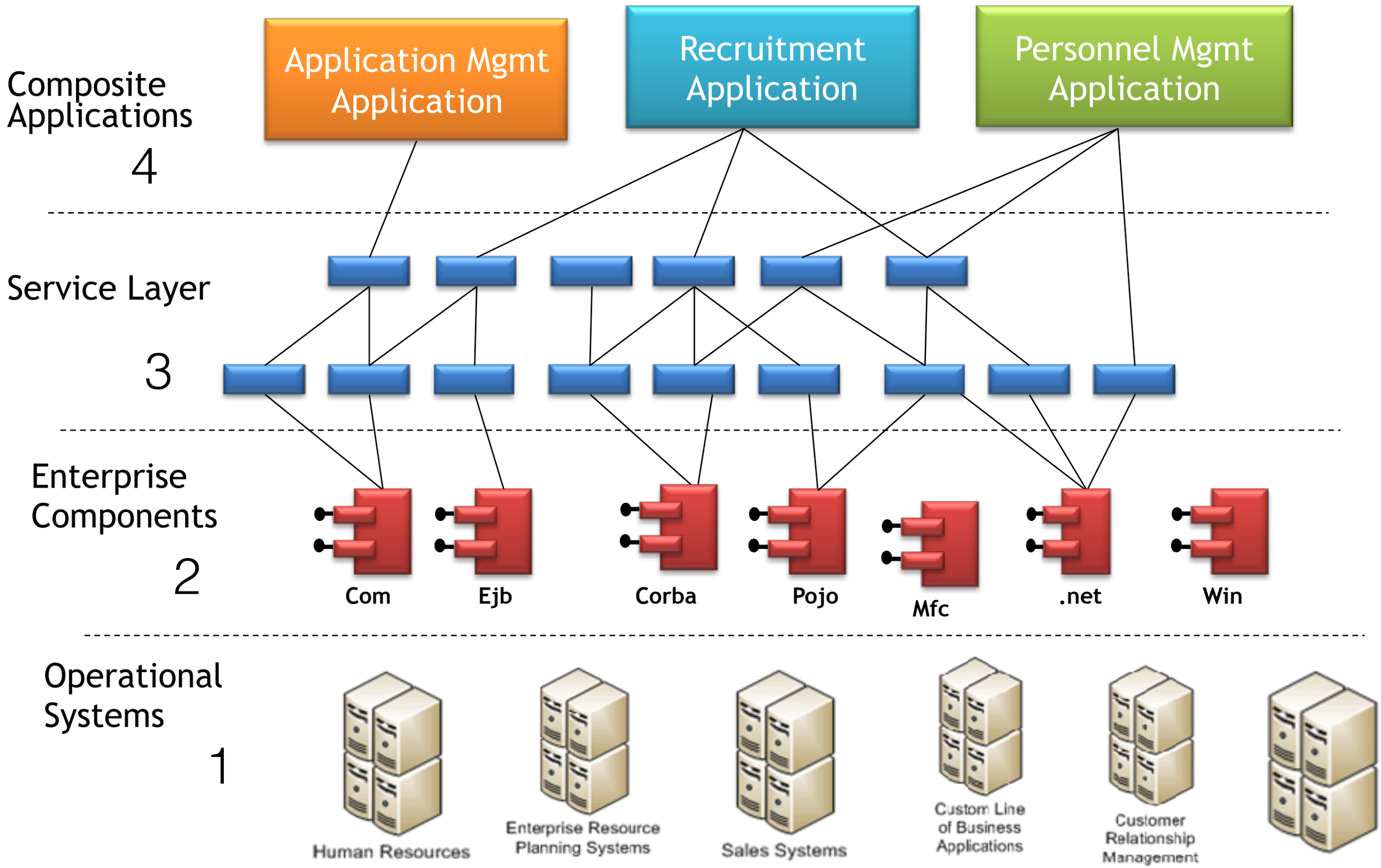
Exe

Dll/so/jar

Application2

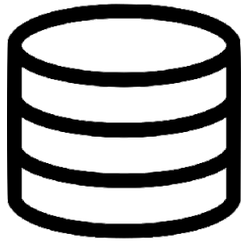
Micro Application

Service Oriented Architecture



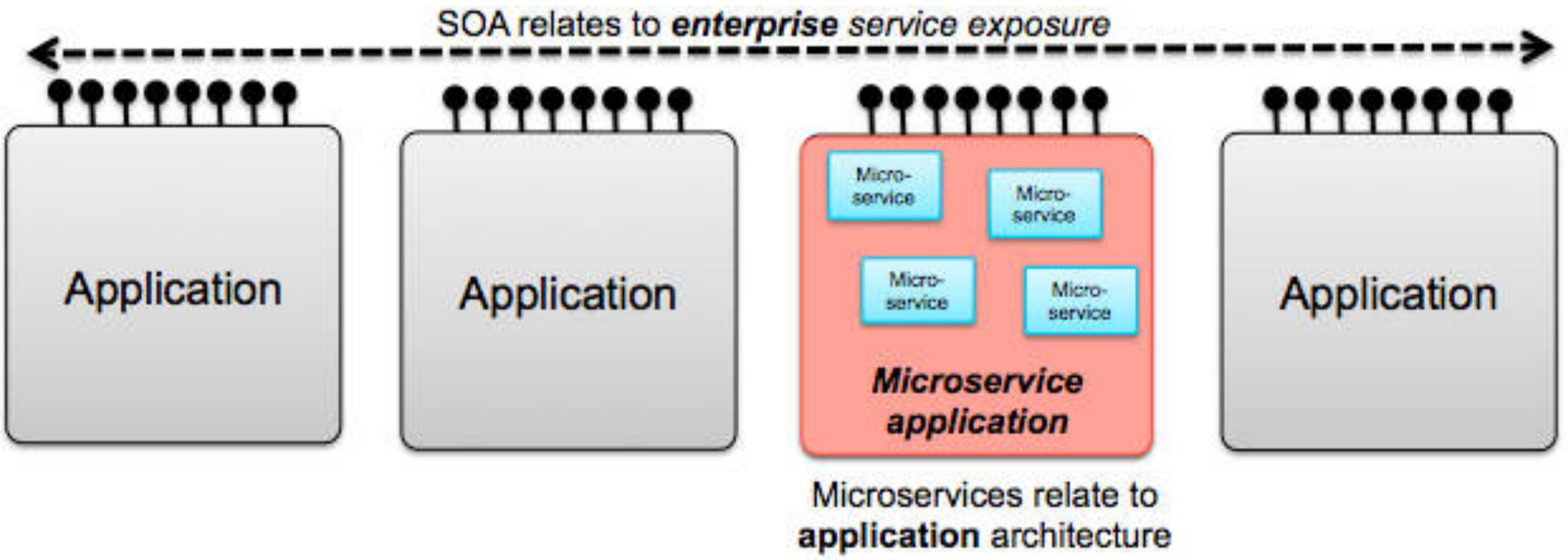
ToDo Portal
(Single Page Application)

ToDo API Service
(Api Application)



ToDo Calendar Service
(Background Application)

ToDo Prediction Service
(Background Application)



Microservices relate to **application** architecture

Break an app into smaller manageable piece

	2 Modules	**2 Applications**	**W**	**Score**
Share Database / Storage	Yes	No	2	
Share Infra (Hosting)	Yes	No	3	
Share Sorce Control	Yes	No	2	
Share CI/CD (Build Server)	Yes or No	No	3	
Share functional logic (same feature)		No		
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	

Application

Modules

Modules

Modules

3 or 4

Application

Small App

Small App

Small App

Modules

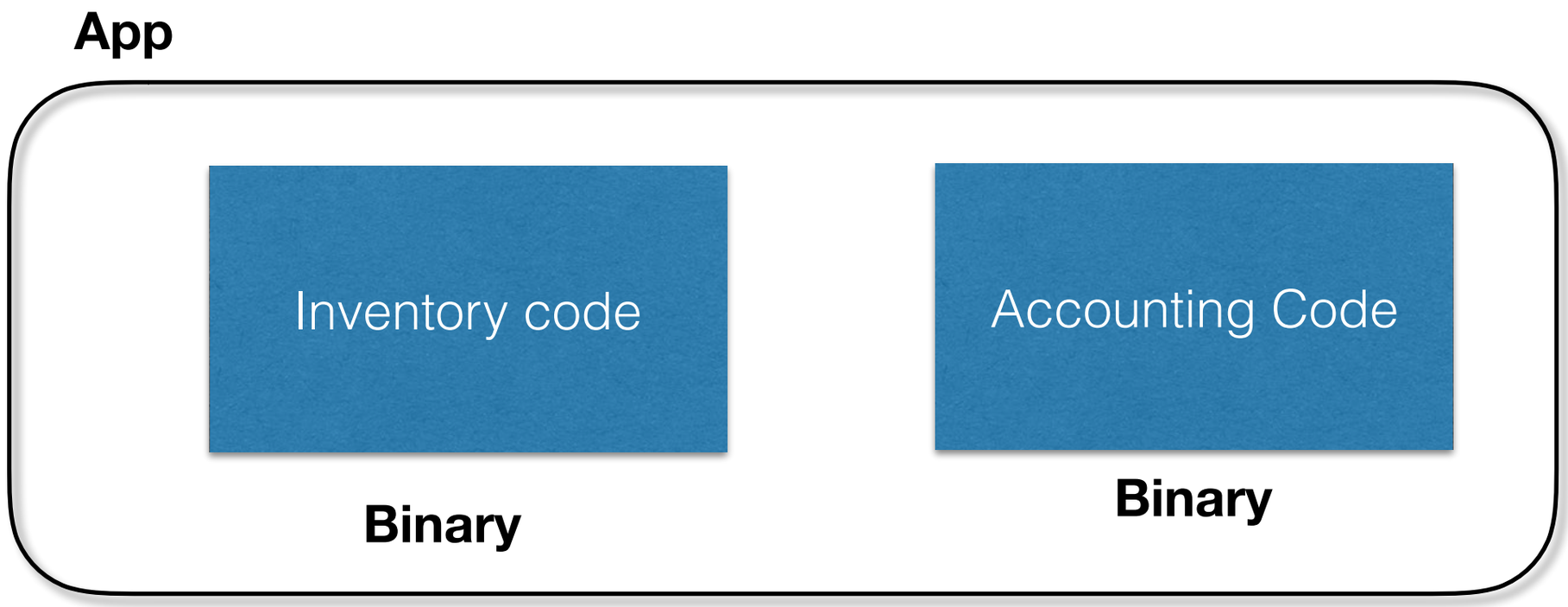
Modules

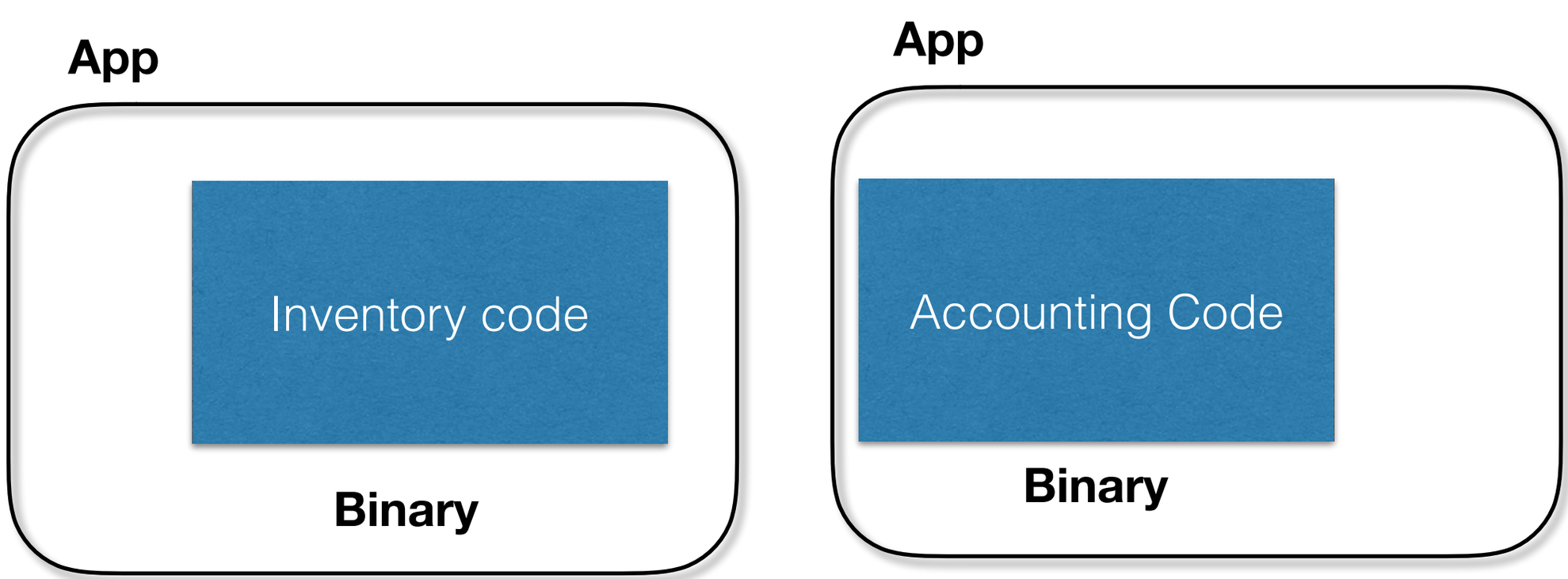
Modules

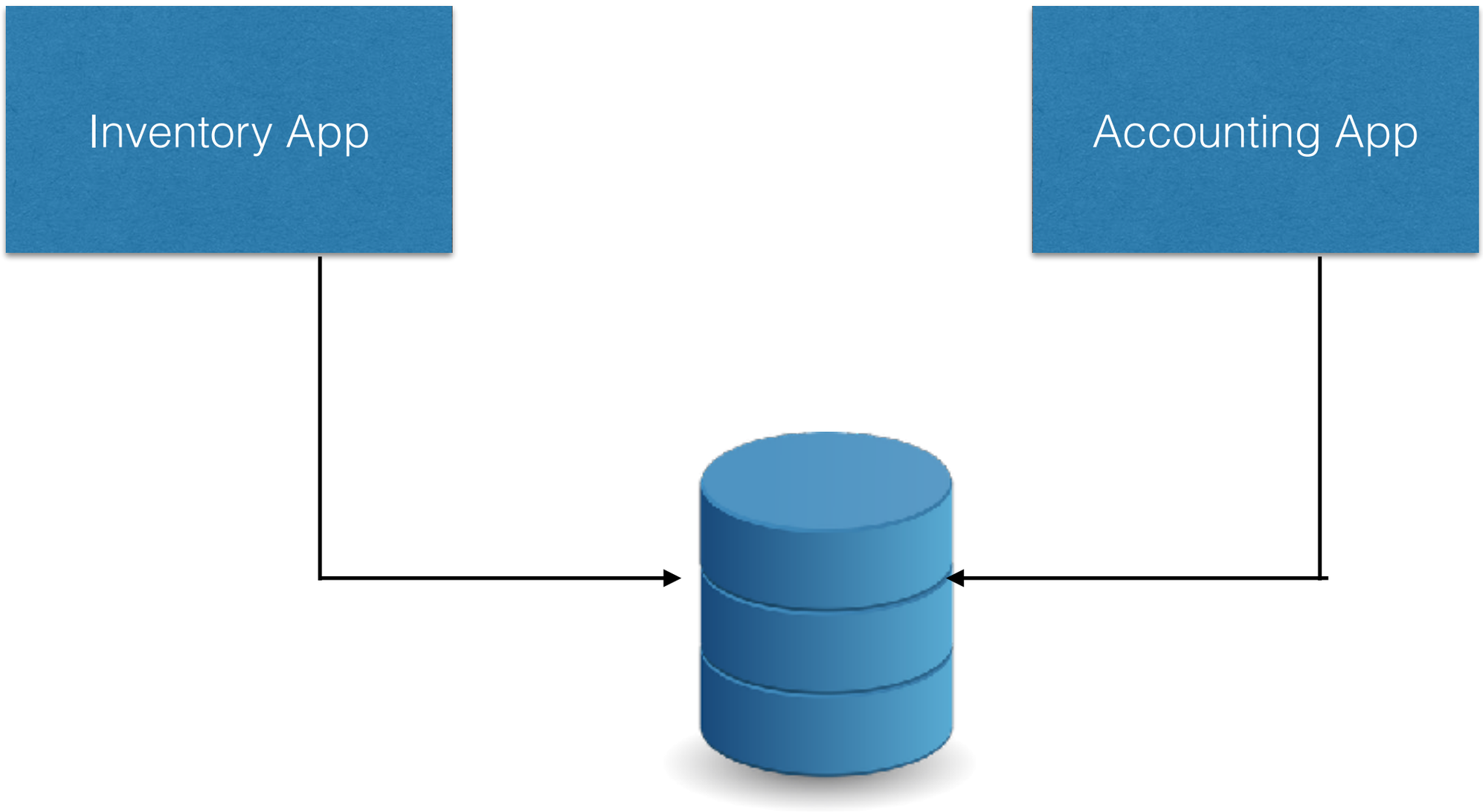
Modules

Modules

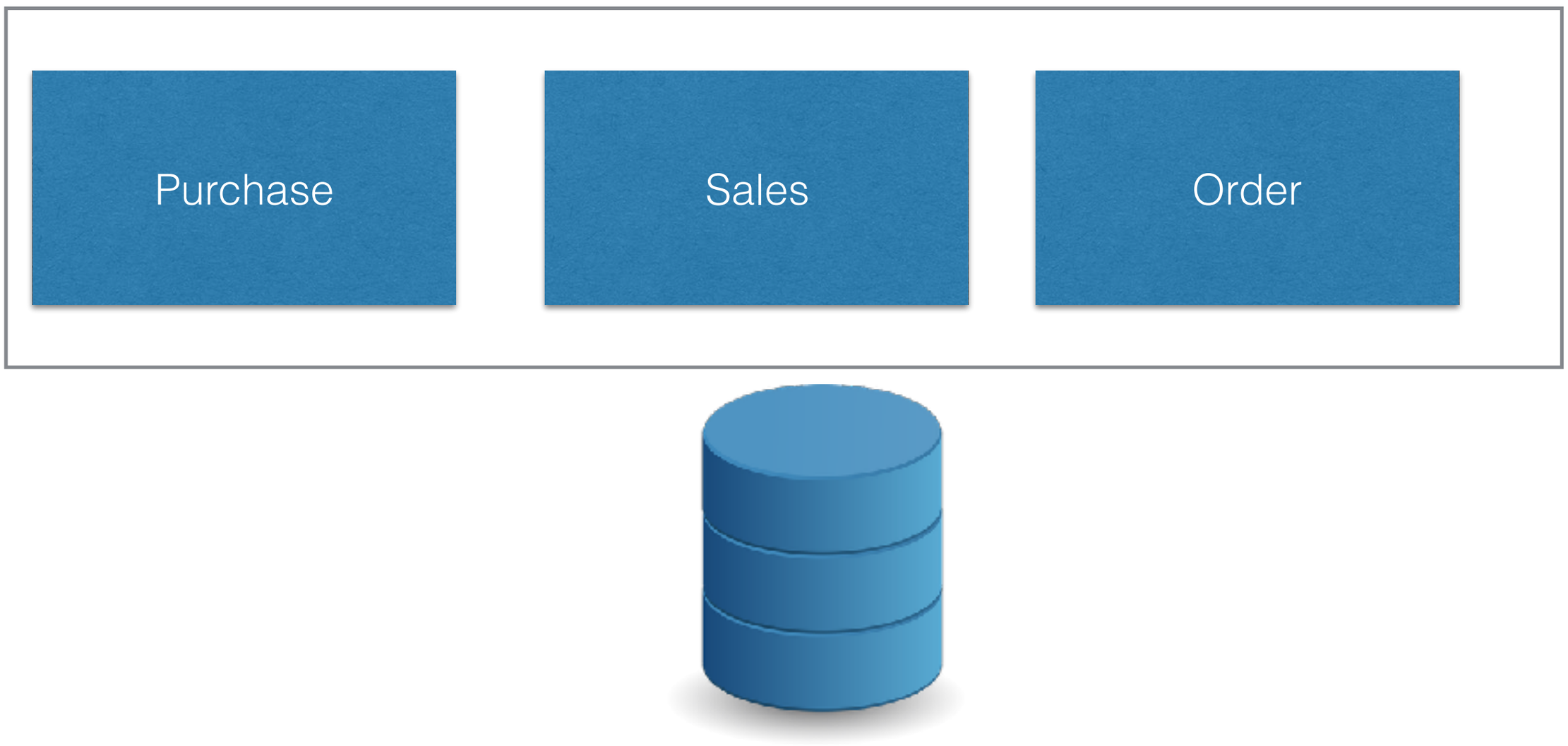
Modules

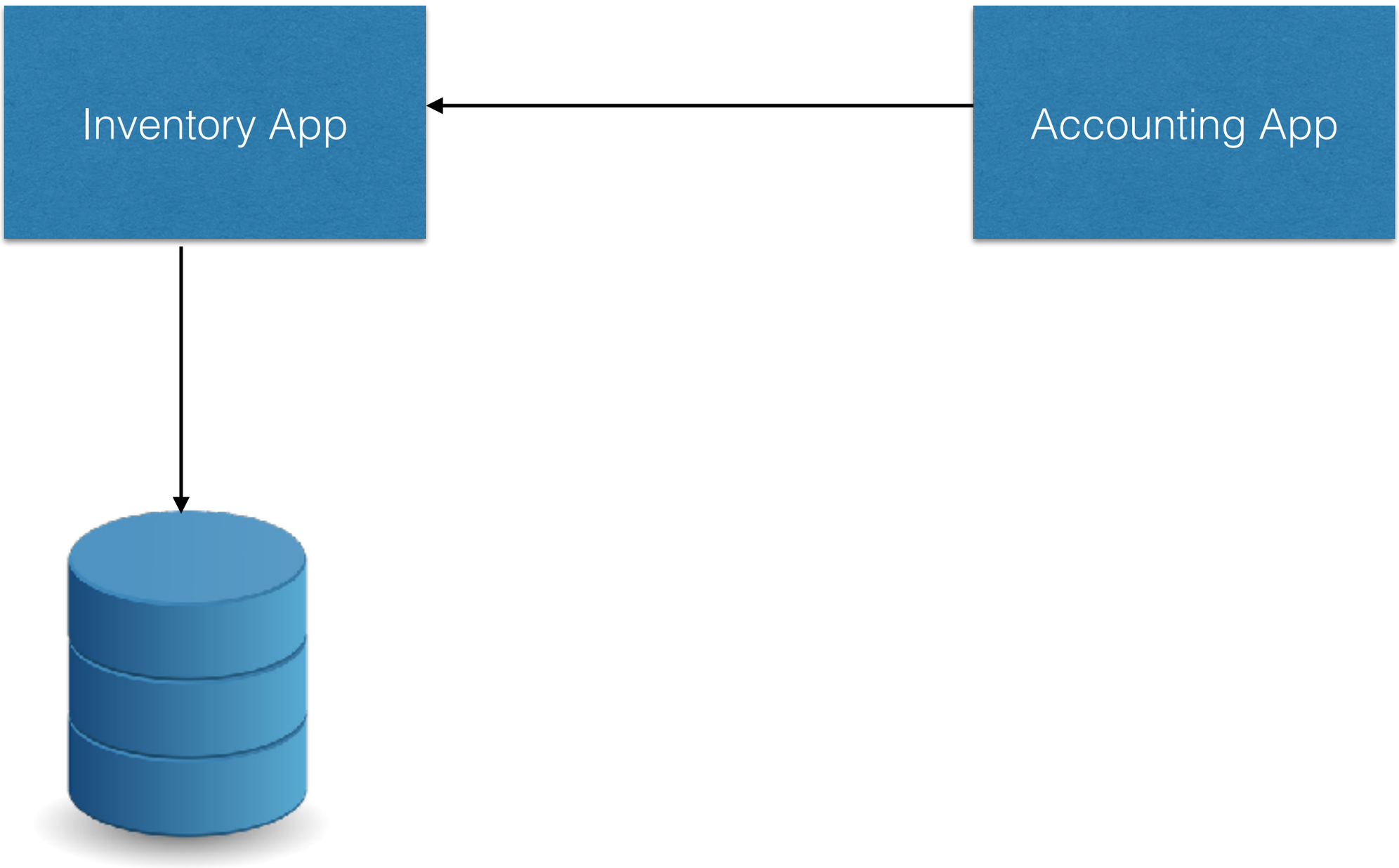

App
Inventory code
Binary
Accounting Code
Binary


App
Inventory code
Binary
App
Accounting Code
Binary


Inventory App
Accounting App

Inventory Application




Inventory App
Accounting App

VM

Inventory App

Accounting App

APP

APP

APP

APP

APP

Types of Microservice

1
Web Server

UI App (SPA)

API Layer

Domain Layer
<<feature1>>

Data Layer

Angular (SPA)

2
Web Server

Browser

UI Layer
(server page)

Domain Layer
<<feature2>>

Data Layer

JSP, Servlet, JSF (Server pages)

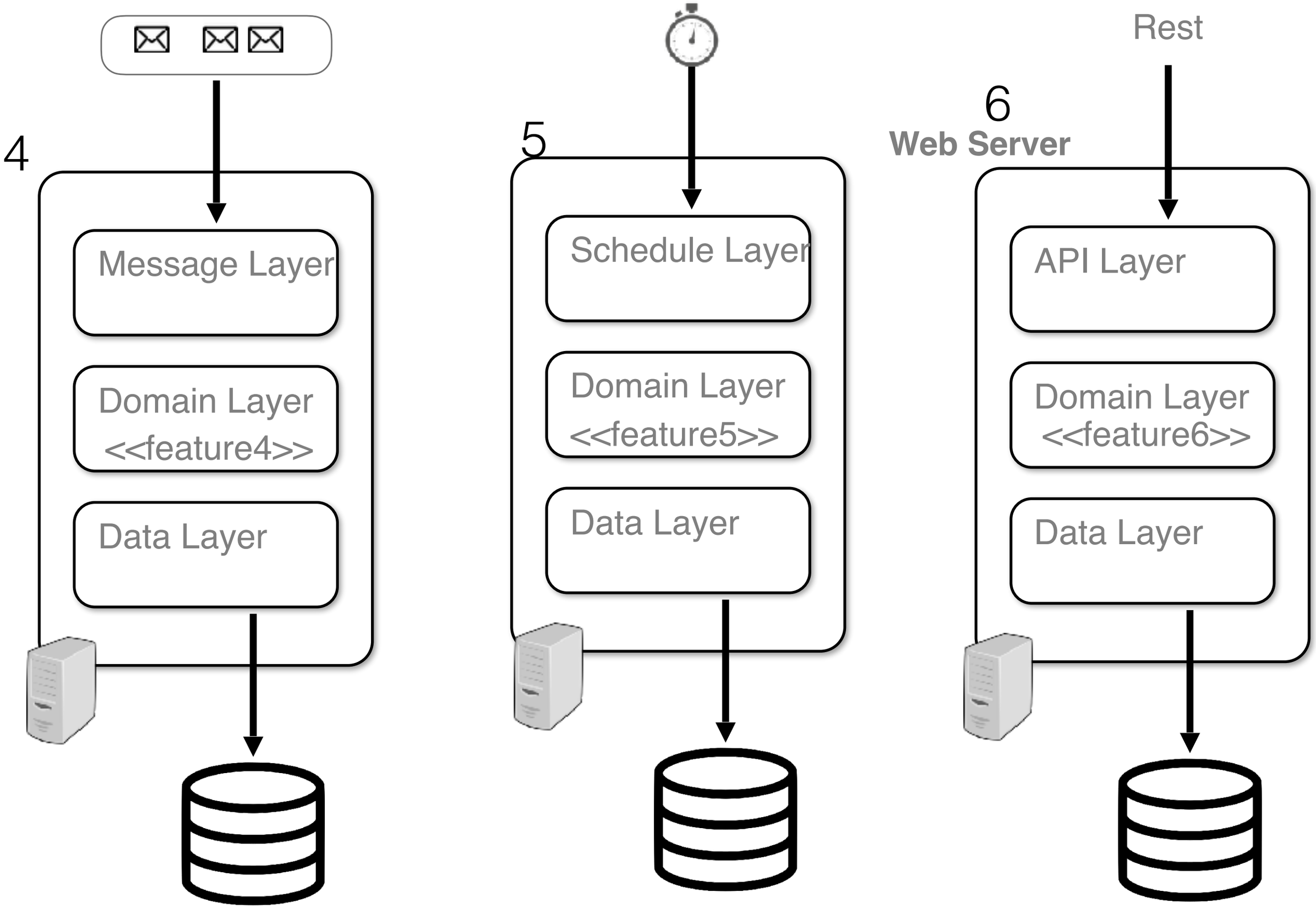
3
Desktop

UI Layer
(native)

Domain Layer
<<feature3>>

Data Layer

Swing, Android, Ios (Native)

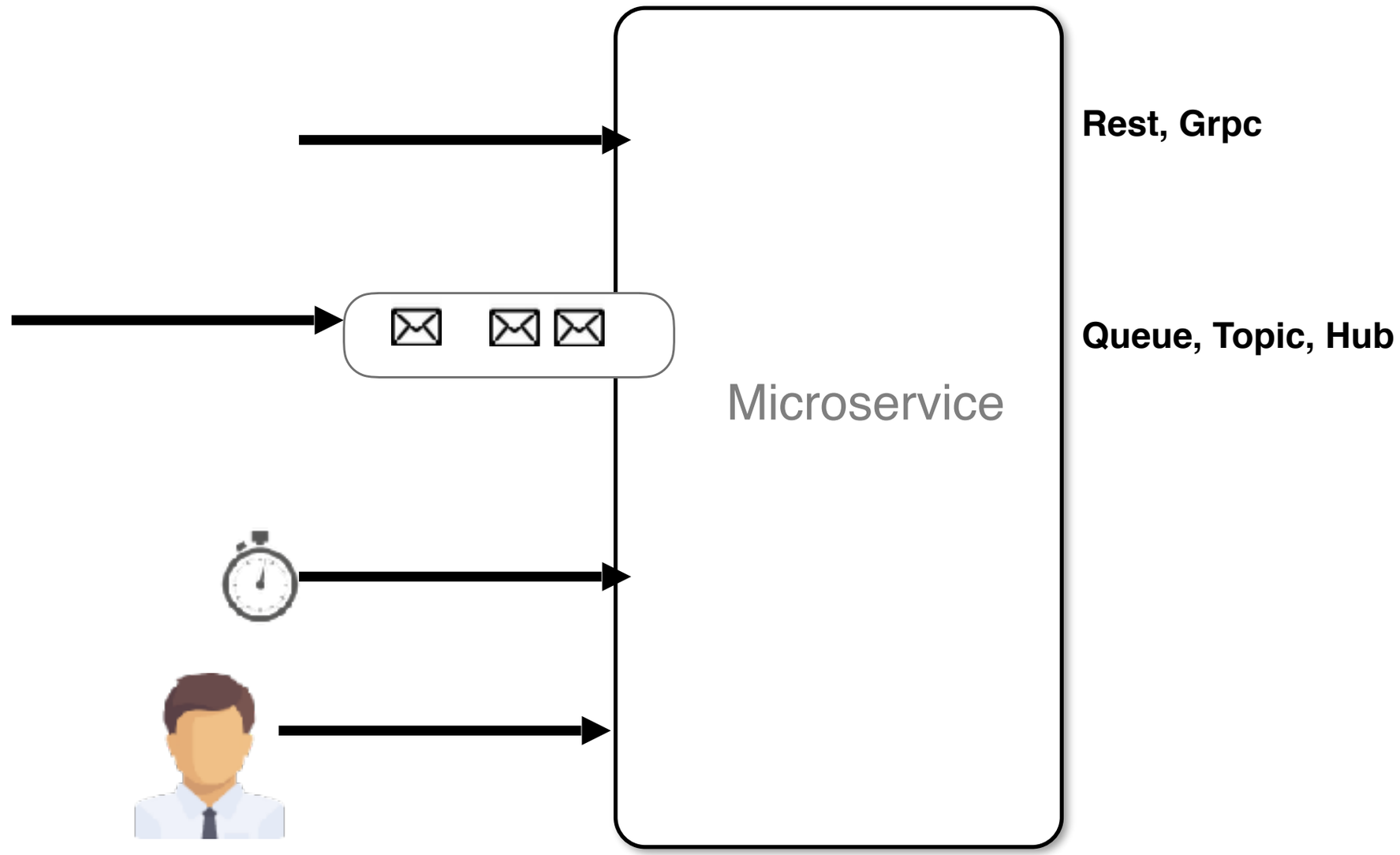

4
Message Layer
Domain Layer
<<feature4>>
Data Layer
5
Schedule Layer
Domain Layer
<<feature5>>
Data Layer
6
Web Server
Rest
API Layer
Domain Layer
<<feature6>>
Data Layer

API

* Message

Schedule

UI

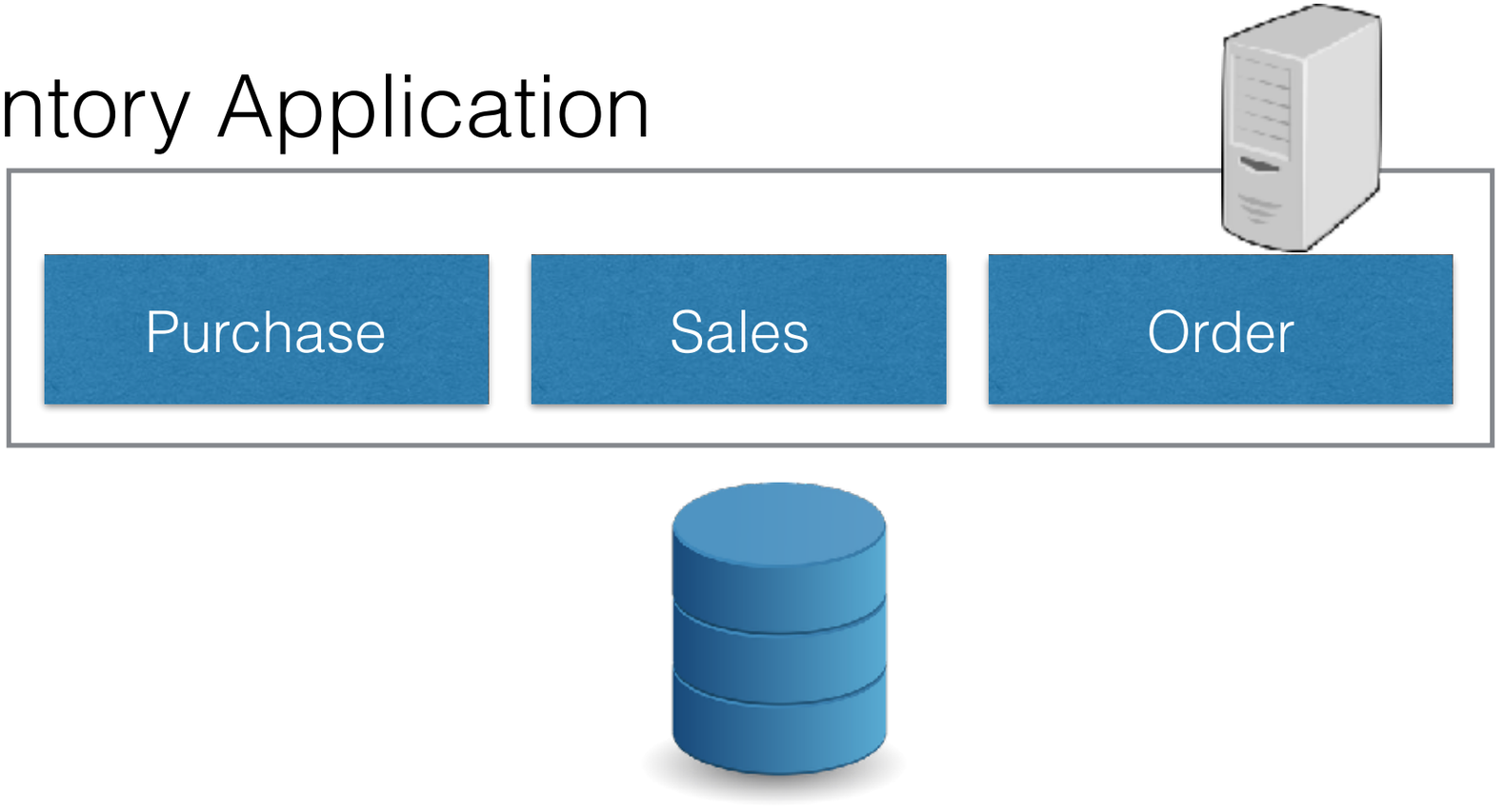


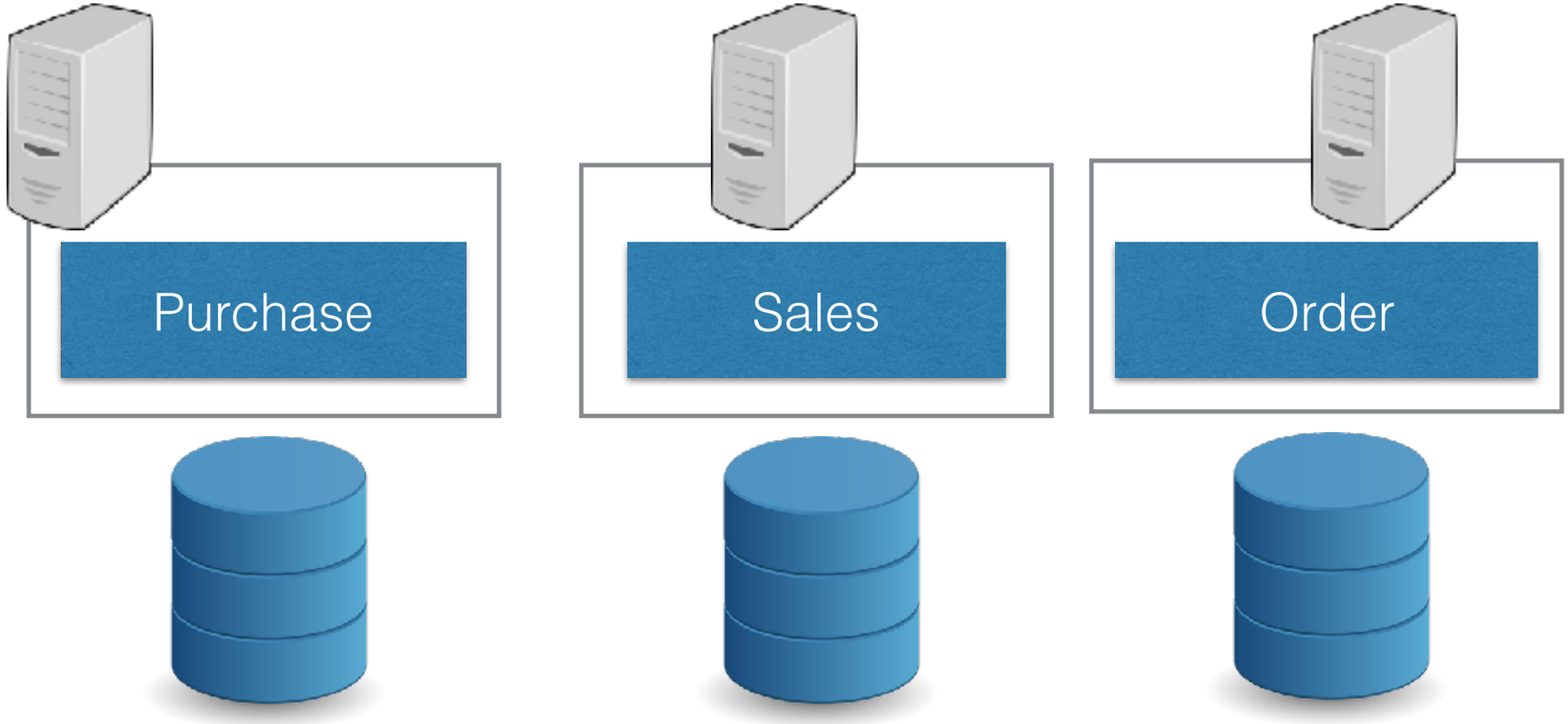
Rest, Grpc

Queue, Topic, Hub

	Pros/ Cons	Solution
Development time		
Micro-service practices Learning Curve		
Resource Performance (CPU, Memory, I/O)		
Db Transaction Management		
Views / Report / Dash board/ join		
Infra Cost		
First time Deployment effort		
Debugging, Error Handling (End to End)		
Integration Test		
Log Mgmt (debug/ error)		
Config Mgmt		
Authentication (who)		
Authorization (what can they do)		
Audit Log mgmt (who accessed what)		
Monitoring / Alerting		
Data Security and Privacy (transit, storage)		
Build Pipeline (CI)		
Agile Architecture (Agility to change)		
Feature Shipping (Agility to ship)/ CD		
Scalability (volume - request, data,		
Availability		
Ability to do Polygot		

Inventory Application

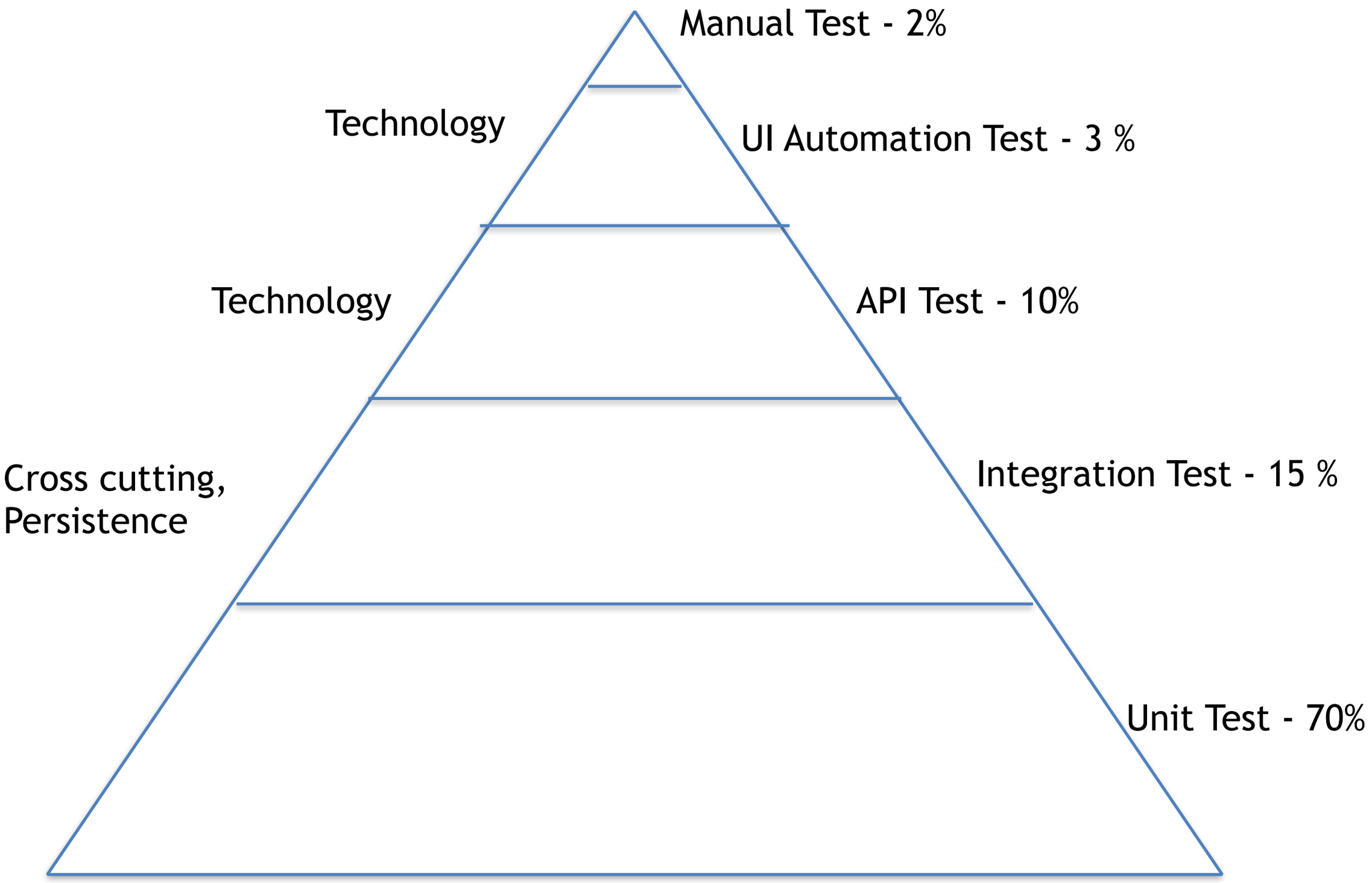




Decentrailze Domain

Centrailze Tech

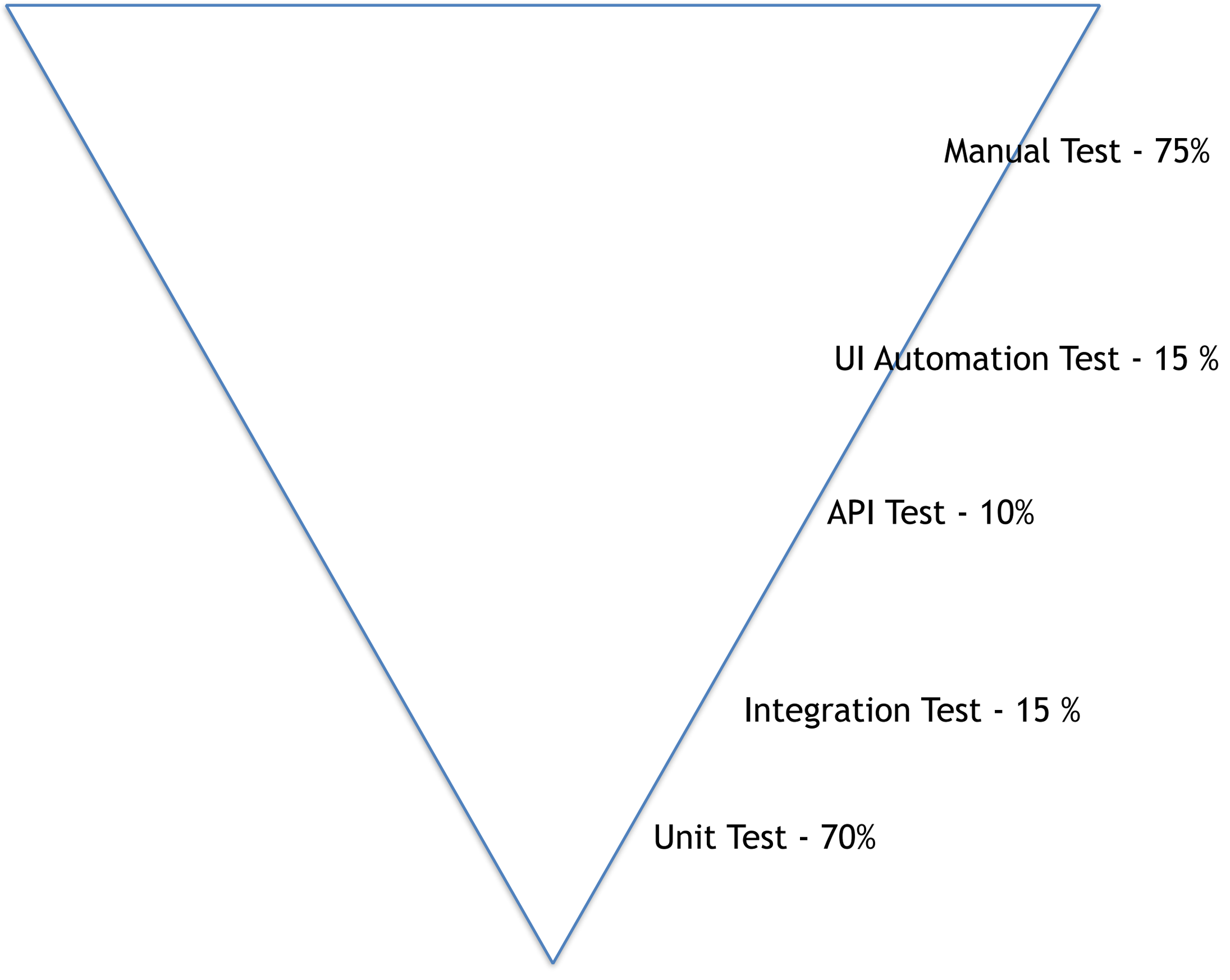
Pyramid test

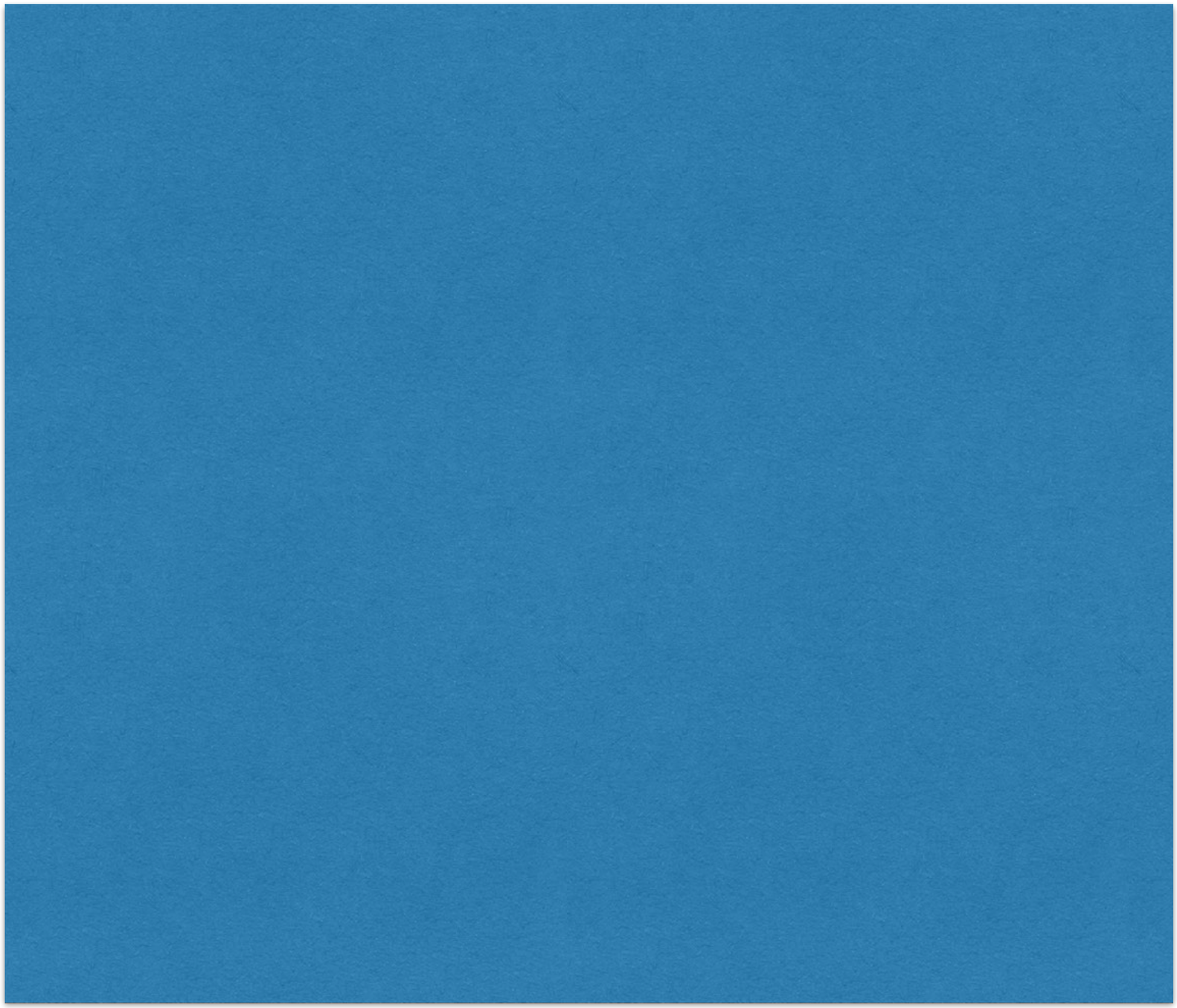


Unit Test

- Documentation for Code
- Design Code
- Regression
- Find Bugs

Its working don’t touch it


Manual Test - 75%
UI Automation Test - 15 %
API Test - 10%
Integration Test - 15 %
Unit Test - 70%



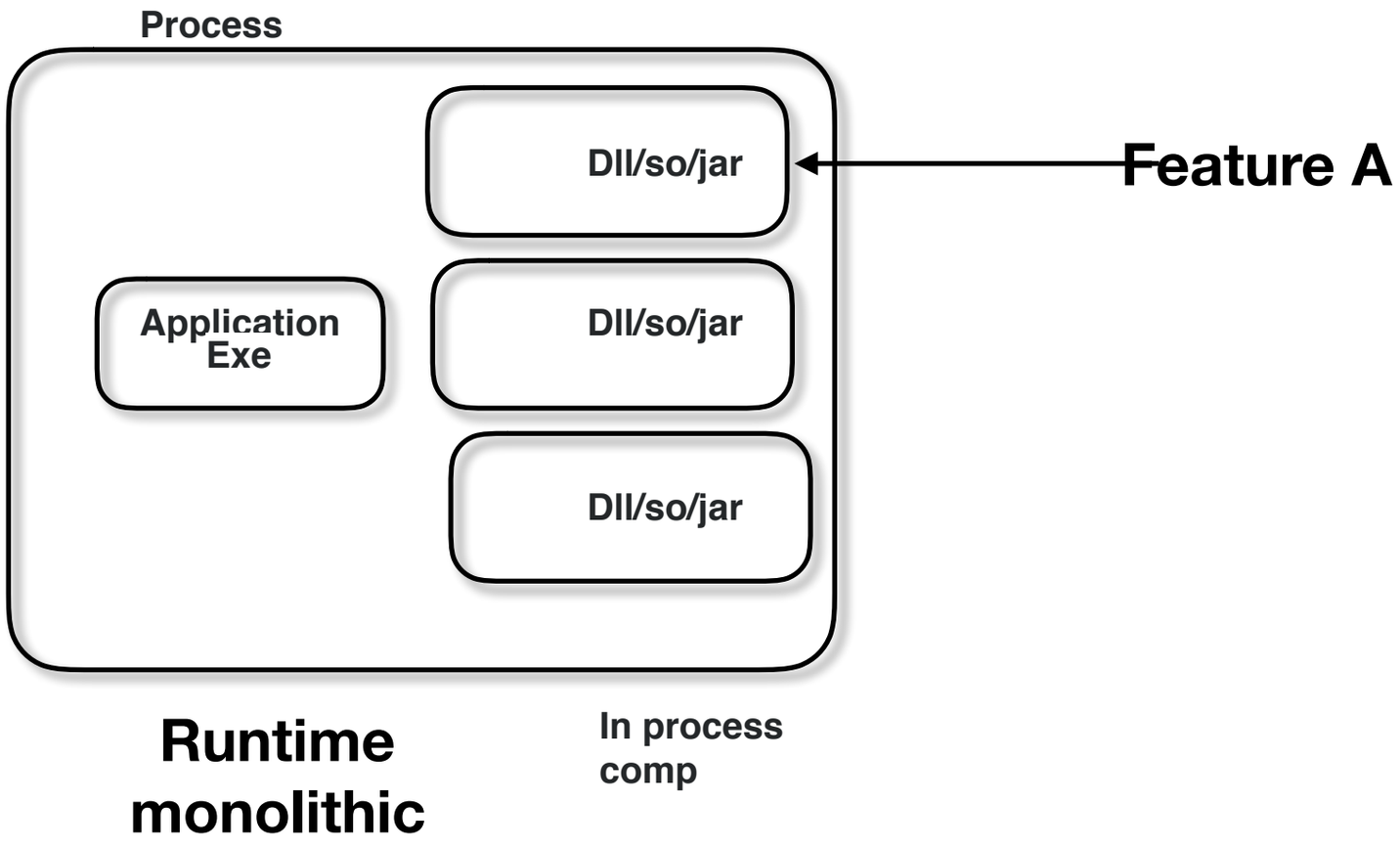
Manual Test - 2%

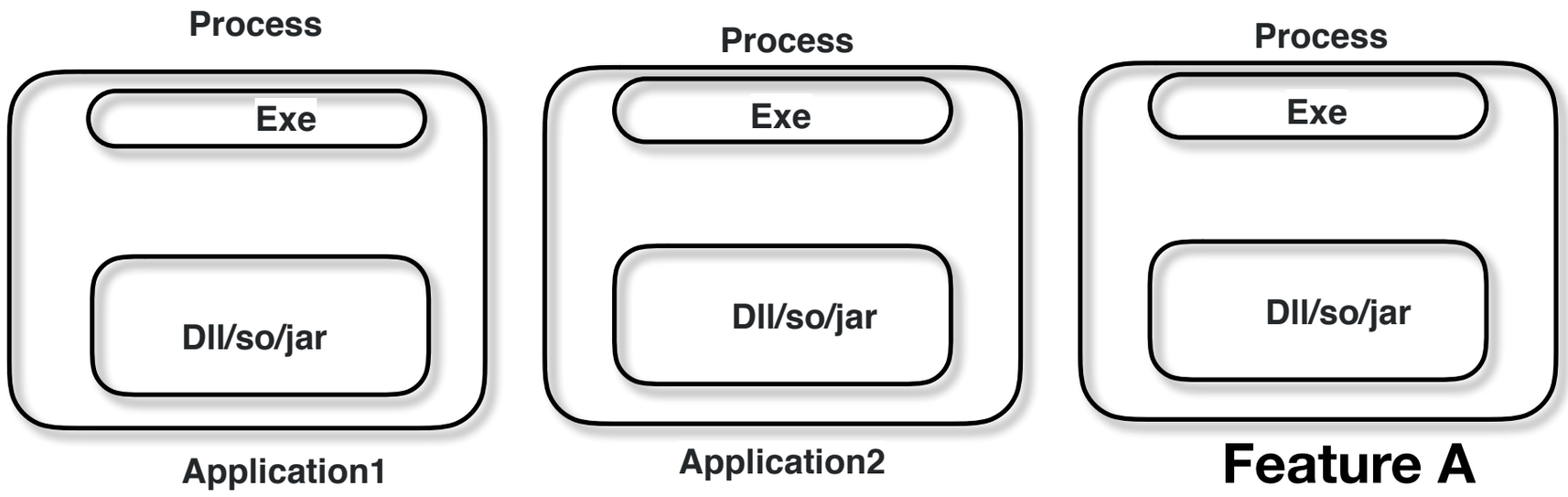
UI Automation Test - 3 %

API Test - 10%

Integration Test - 15 %

Unit Test - 70%

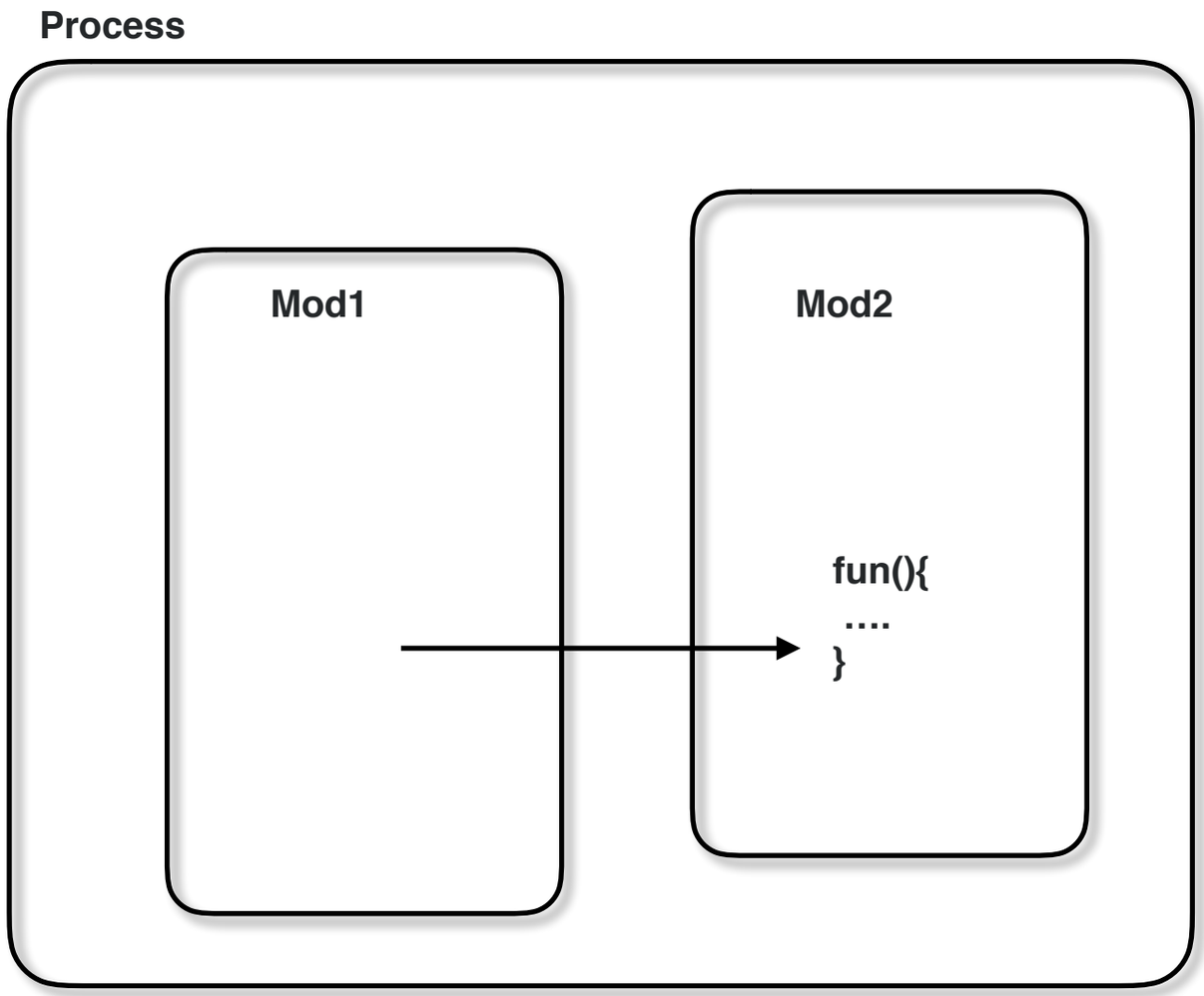




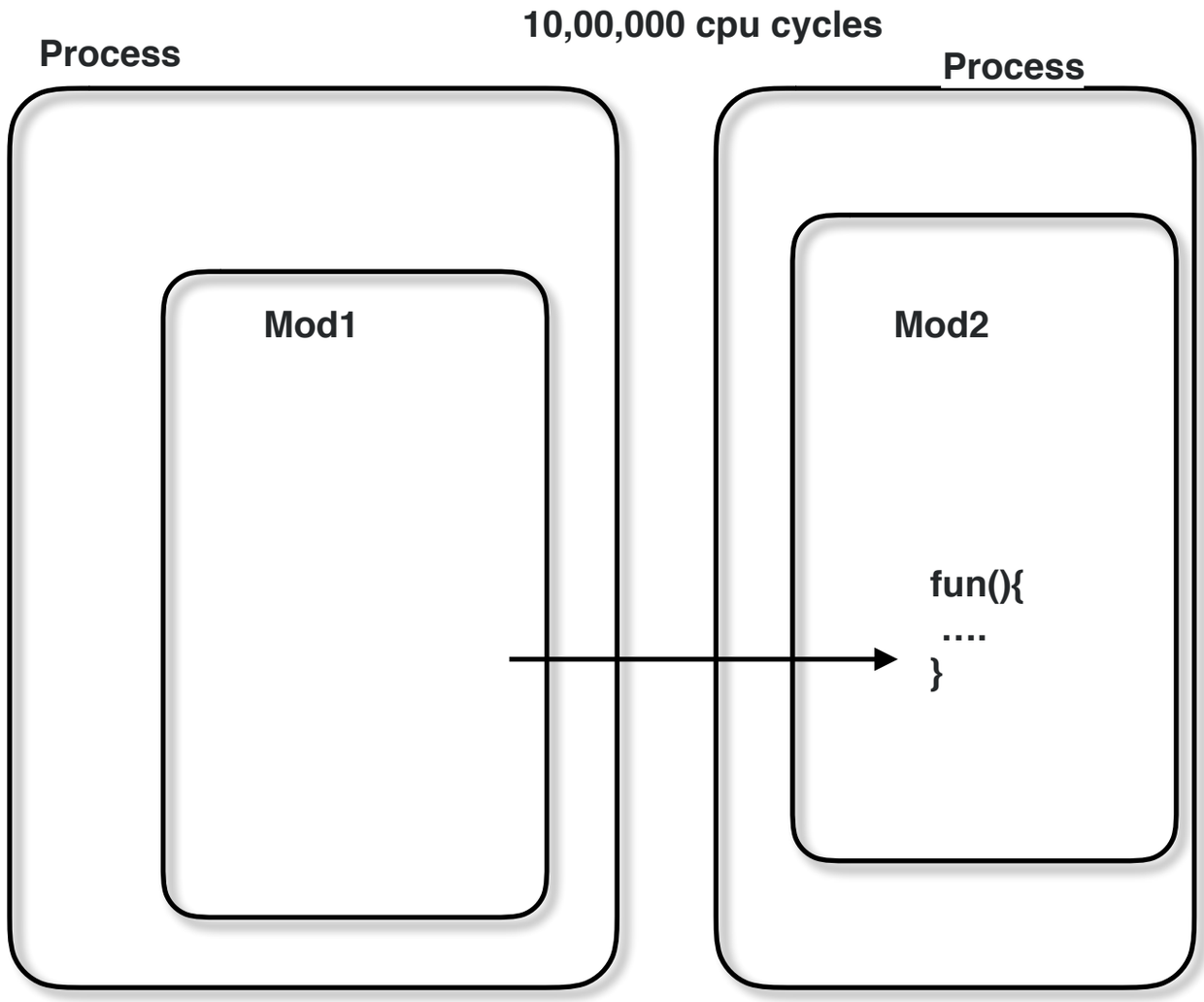
Micro Application

1. Performance

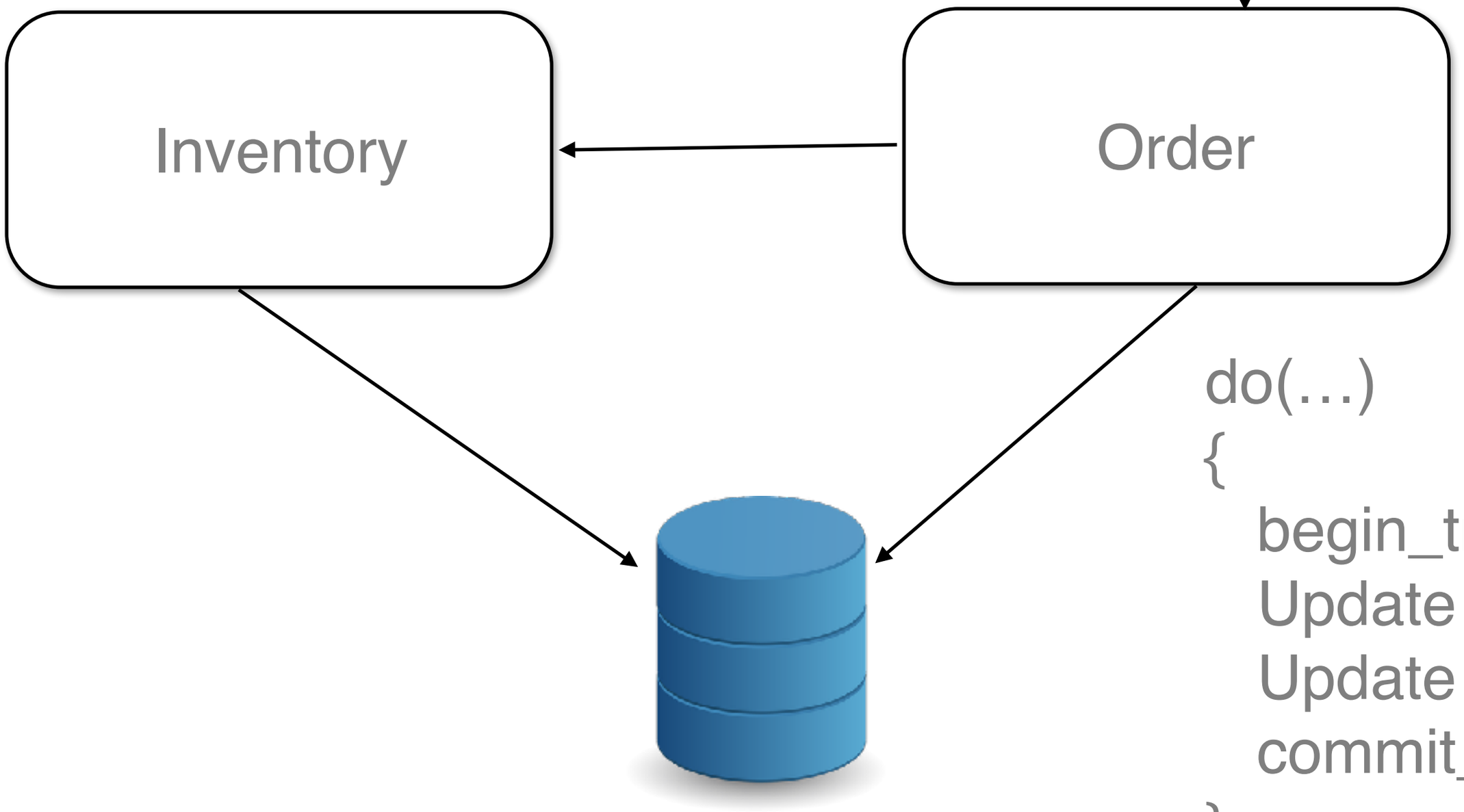
Operation	Cpu Cycles
• 10 + 12	3
• Calling a in-memory Method	10
• Create Thread	2,00,000
• Destroy Thread	1,00,000
• Database Call	40,00,000
• Distributed Fun Call	20,00,000
Write to disk	10,00,000



10 cpu cycles

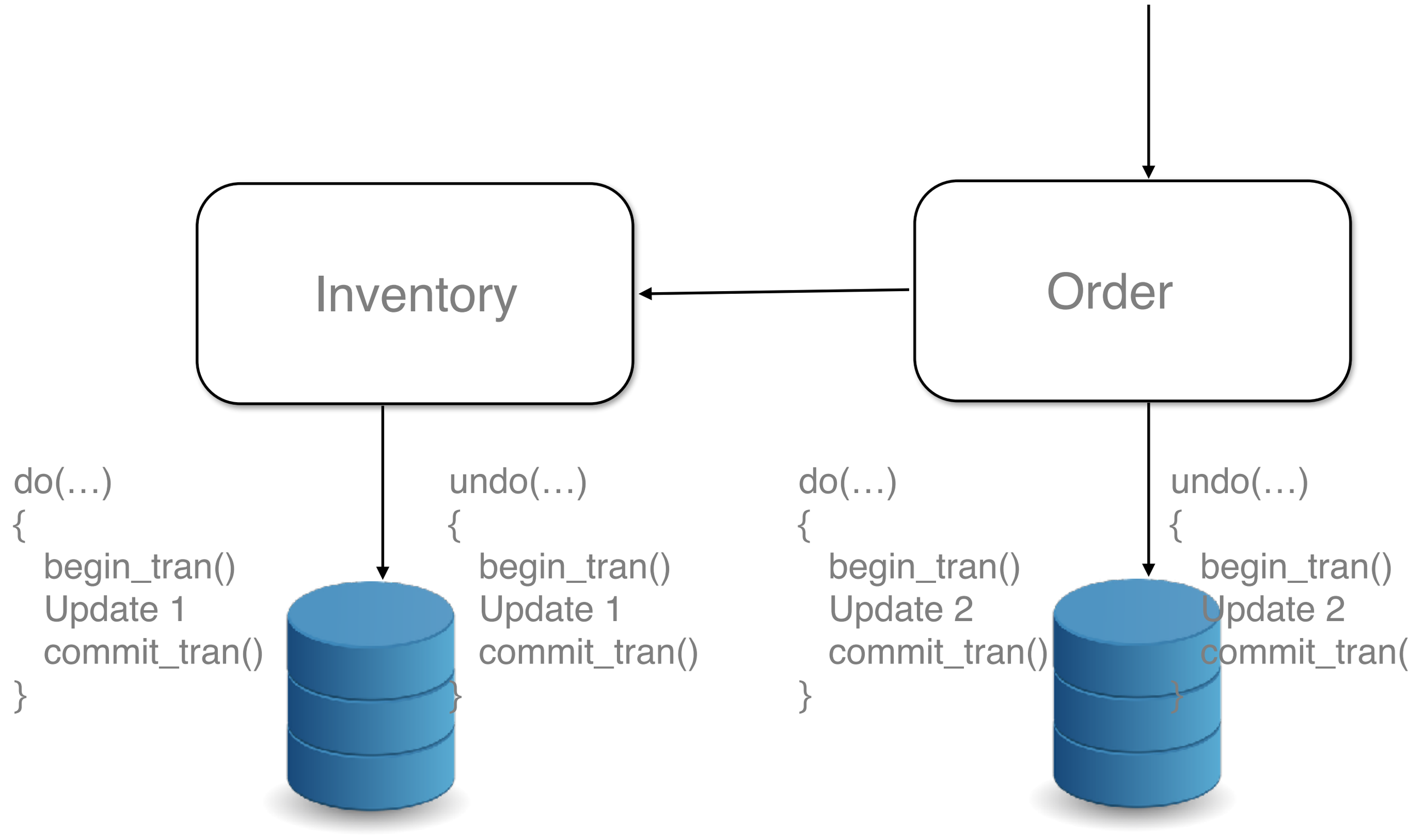


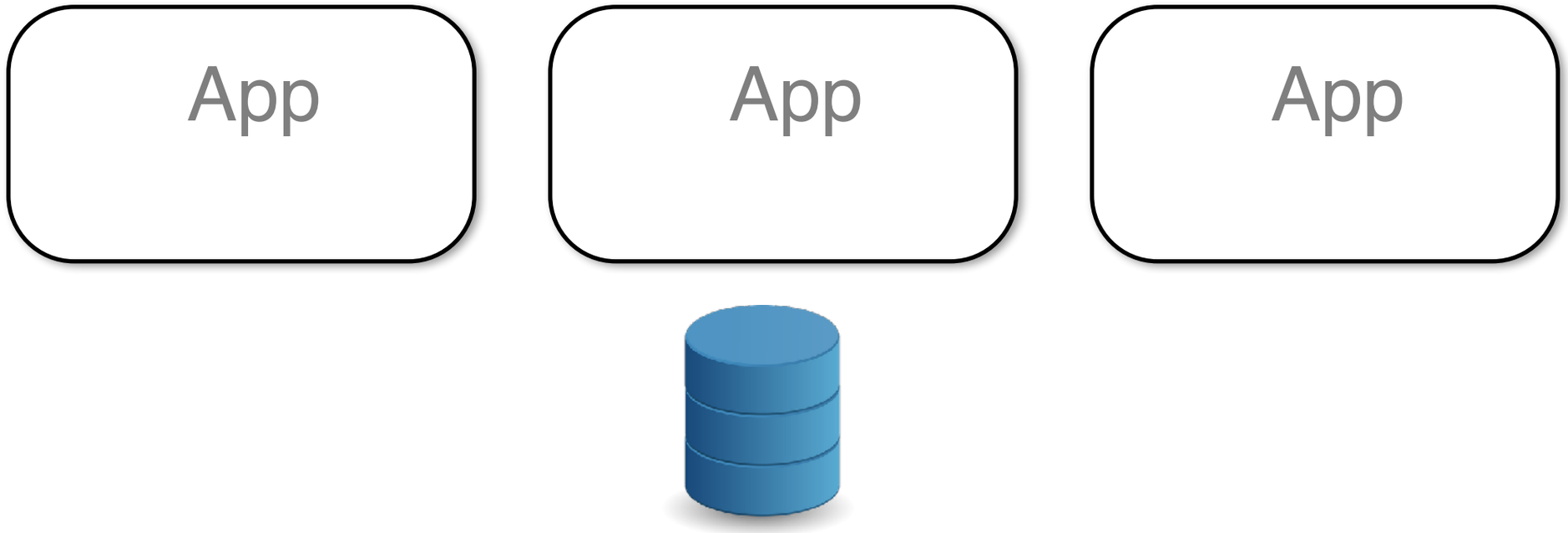
2. Db transaction



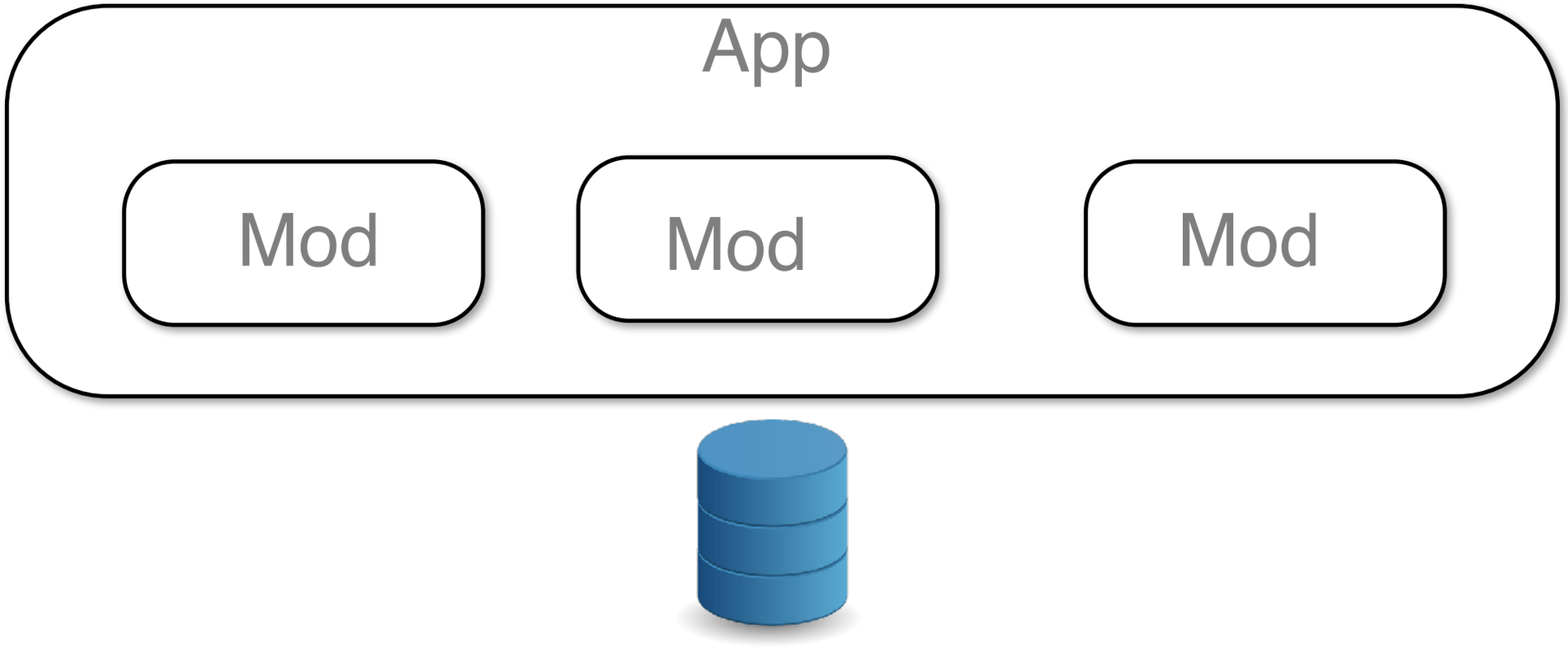

```
do(…)
{
   begin_tran()
   Update 1
   Update 2
   commit_tran()
}
```

2. Db transaction



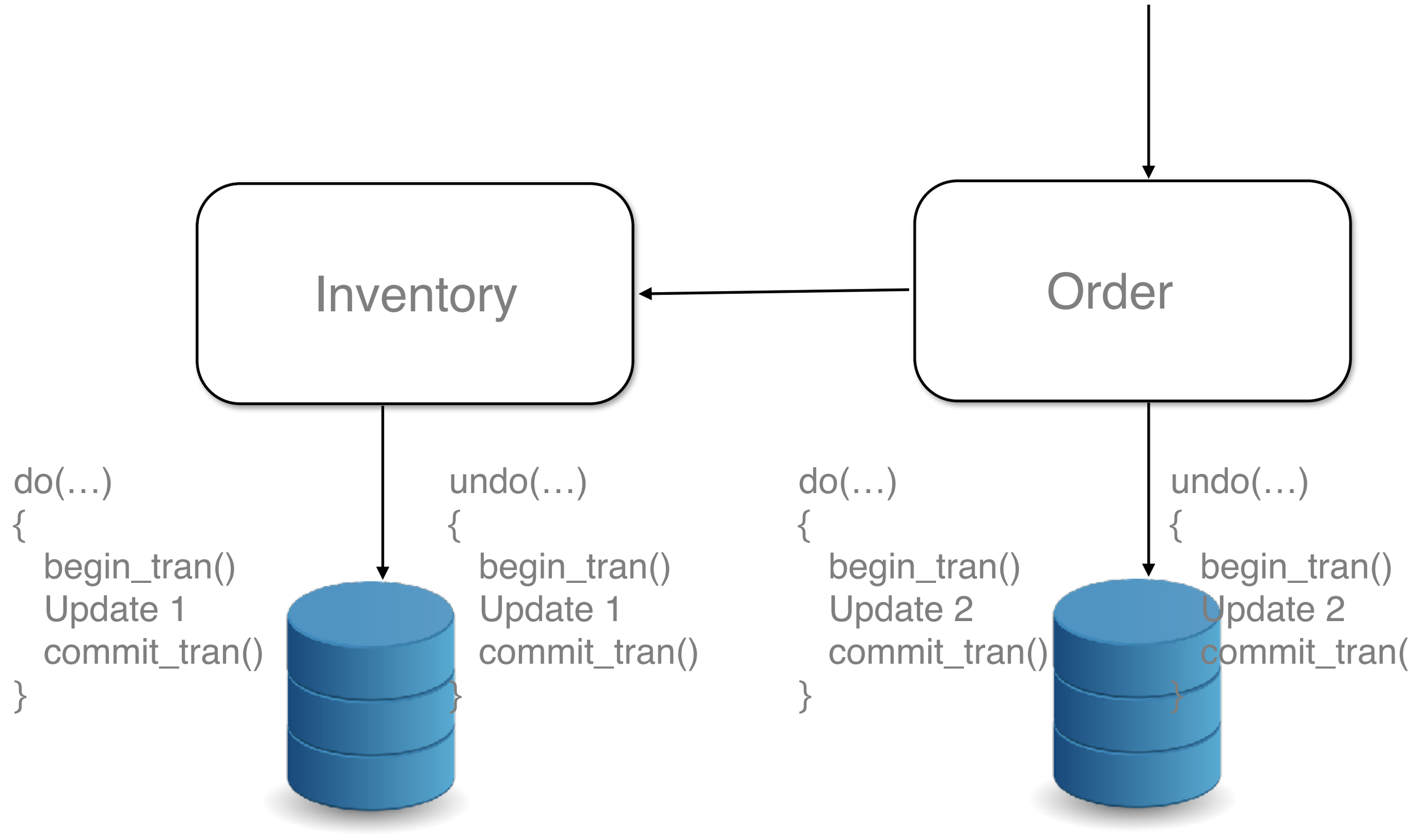

App
App
App

App
App
App


App
Mod
Mod
Mod

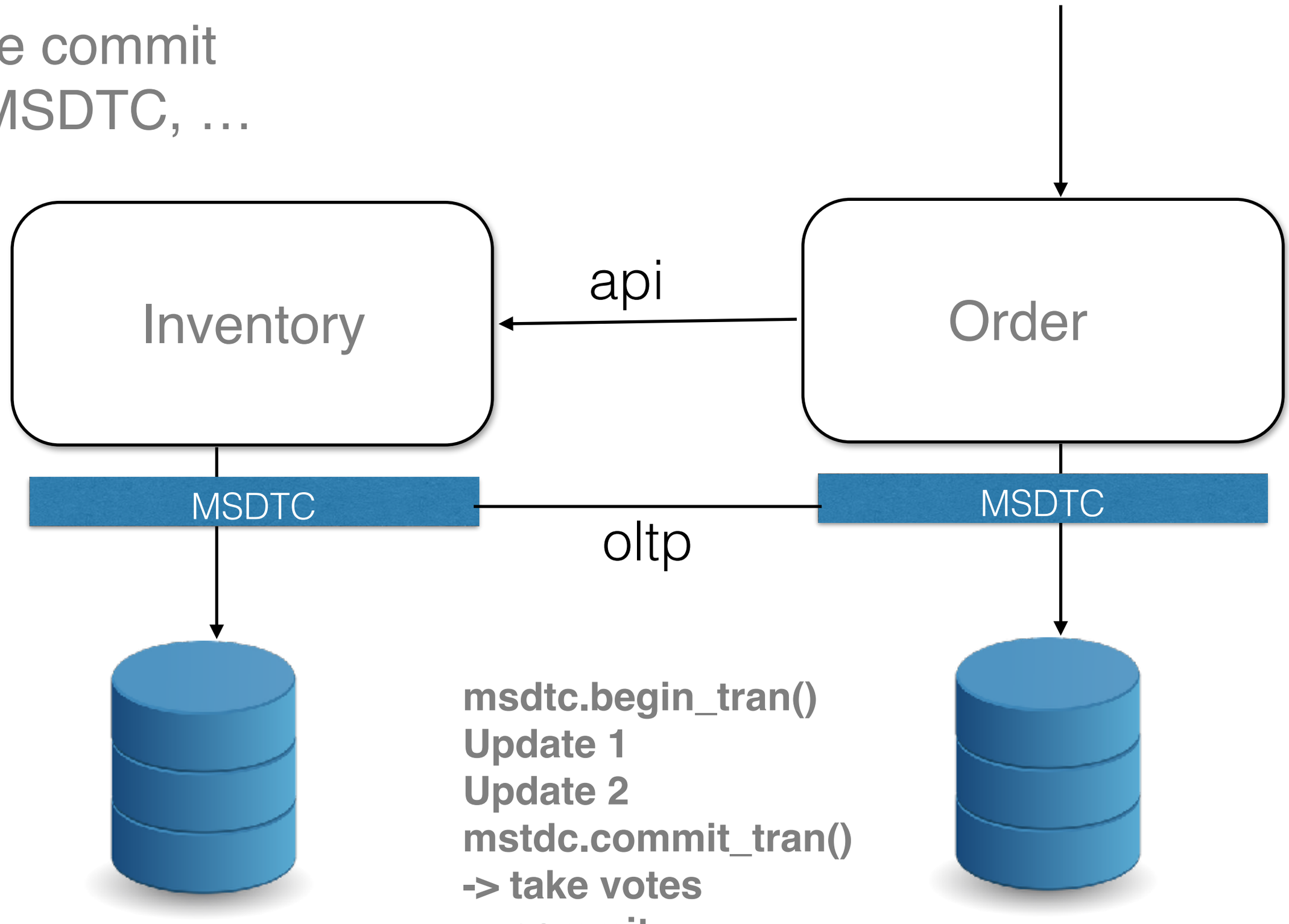

App
App
App

2. Db transaction



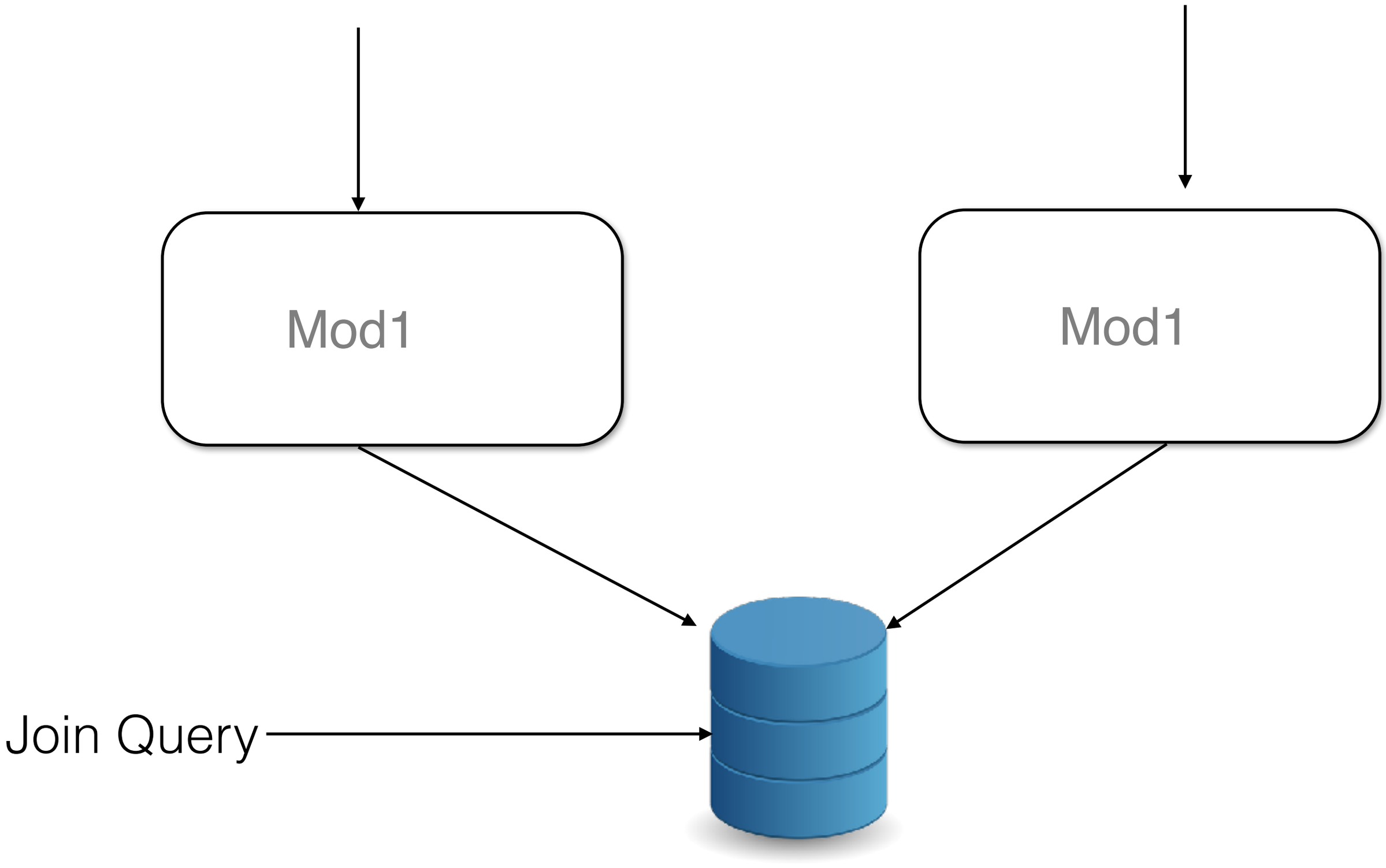
2. Db transaction

2 phase commit
JTX , MSDTC, …

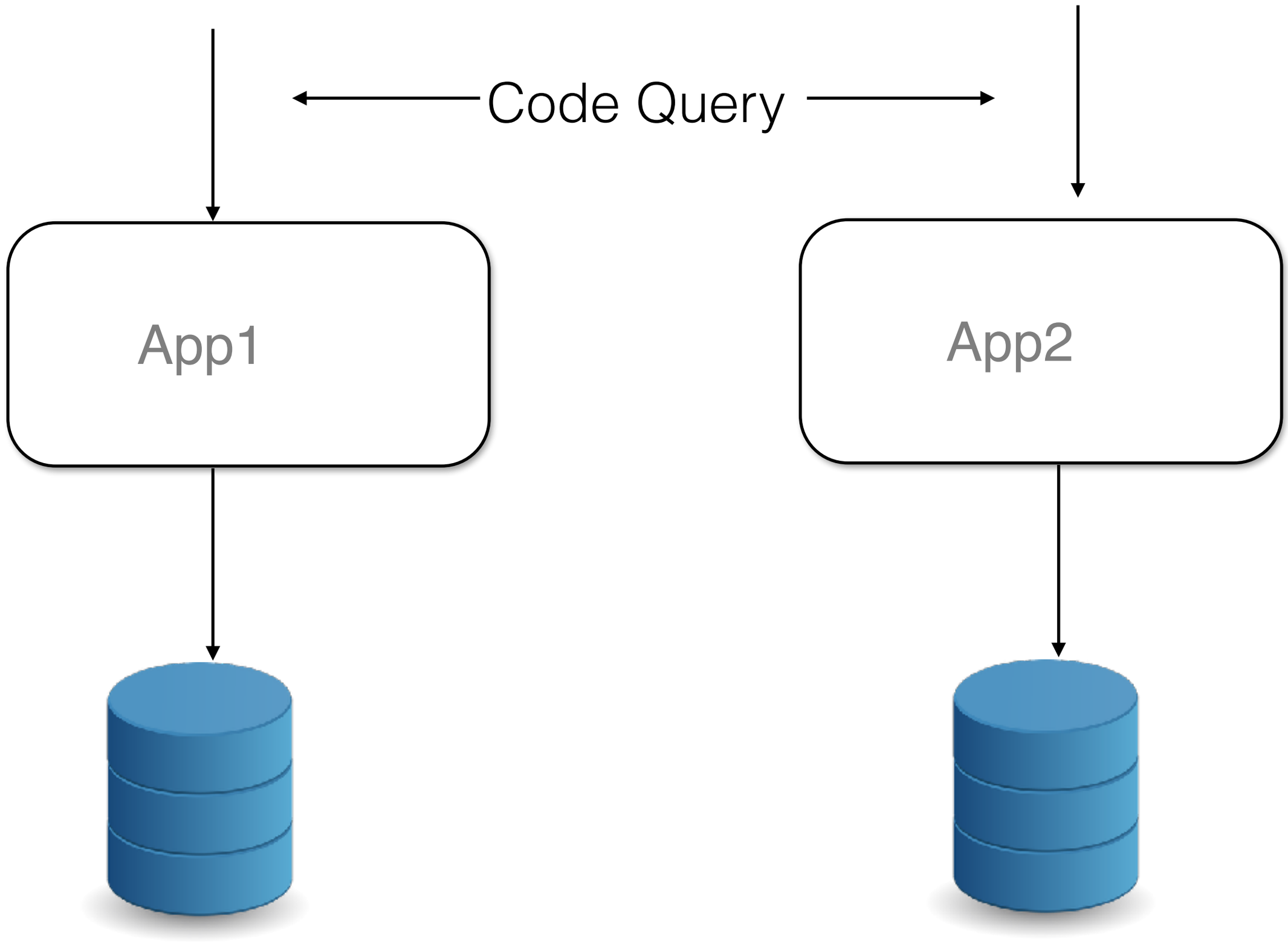


msdtc.begin_tran()
Update 1
Update 2
mstdc.commit_tran()
-> take votes
-> commit

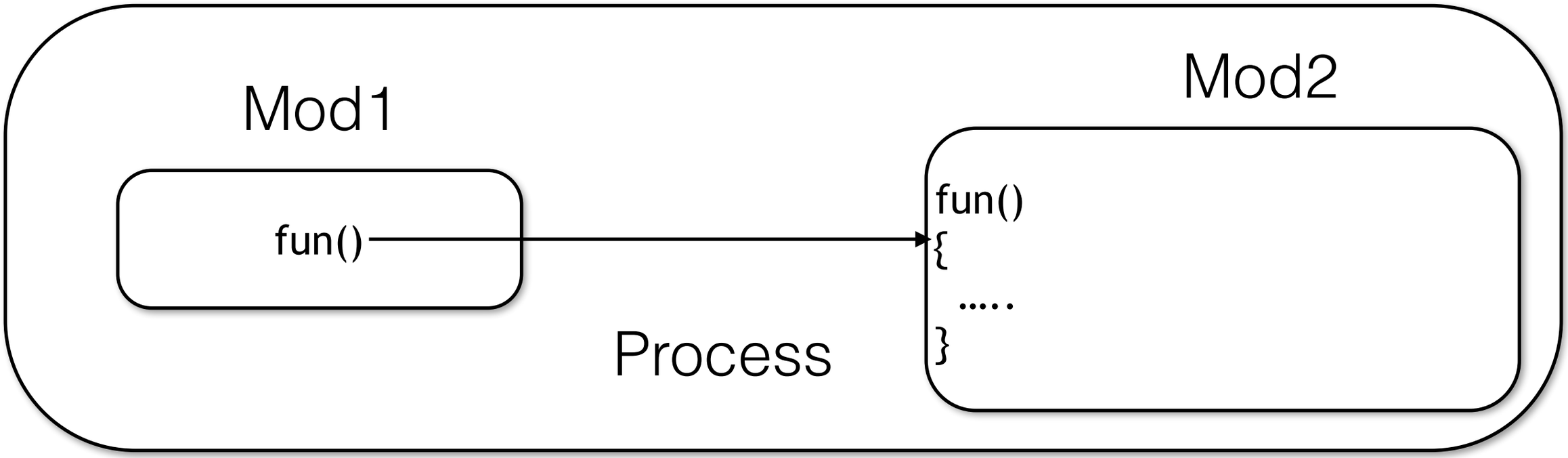
3. Db query

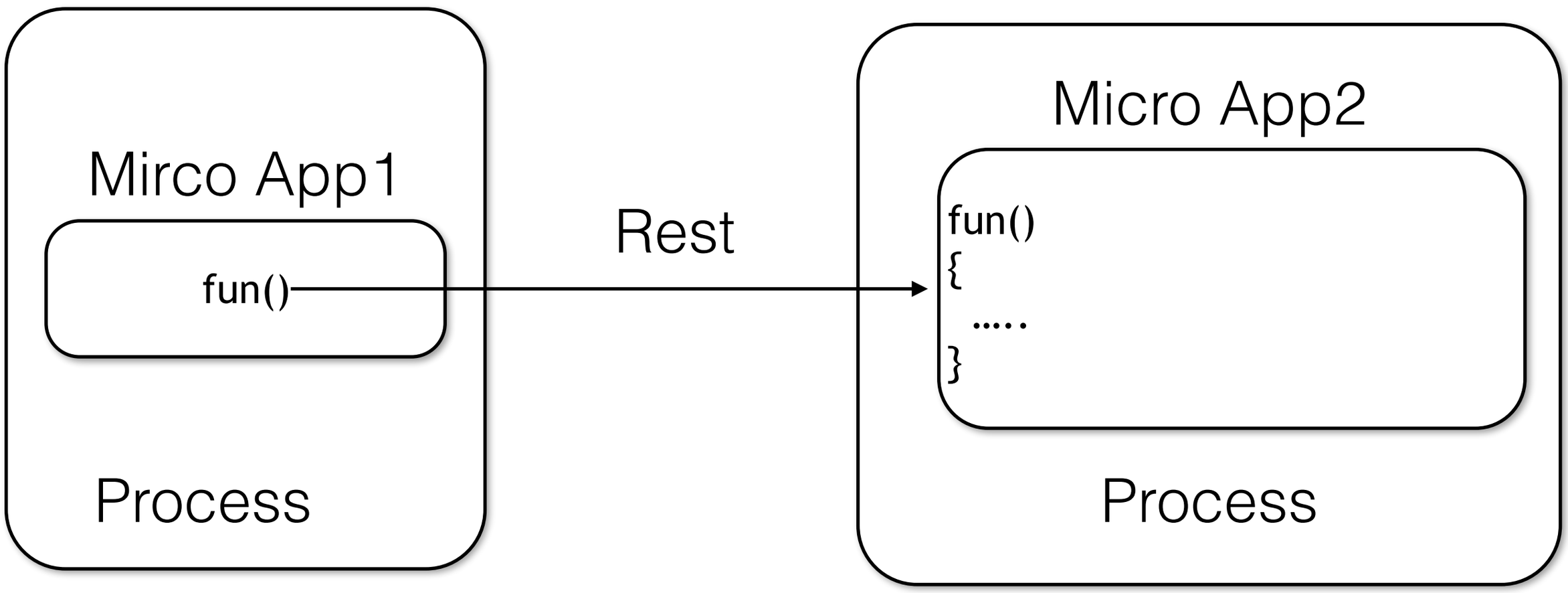


3. Query

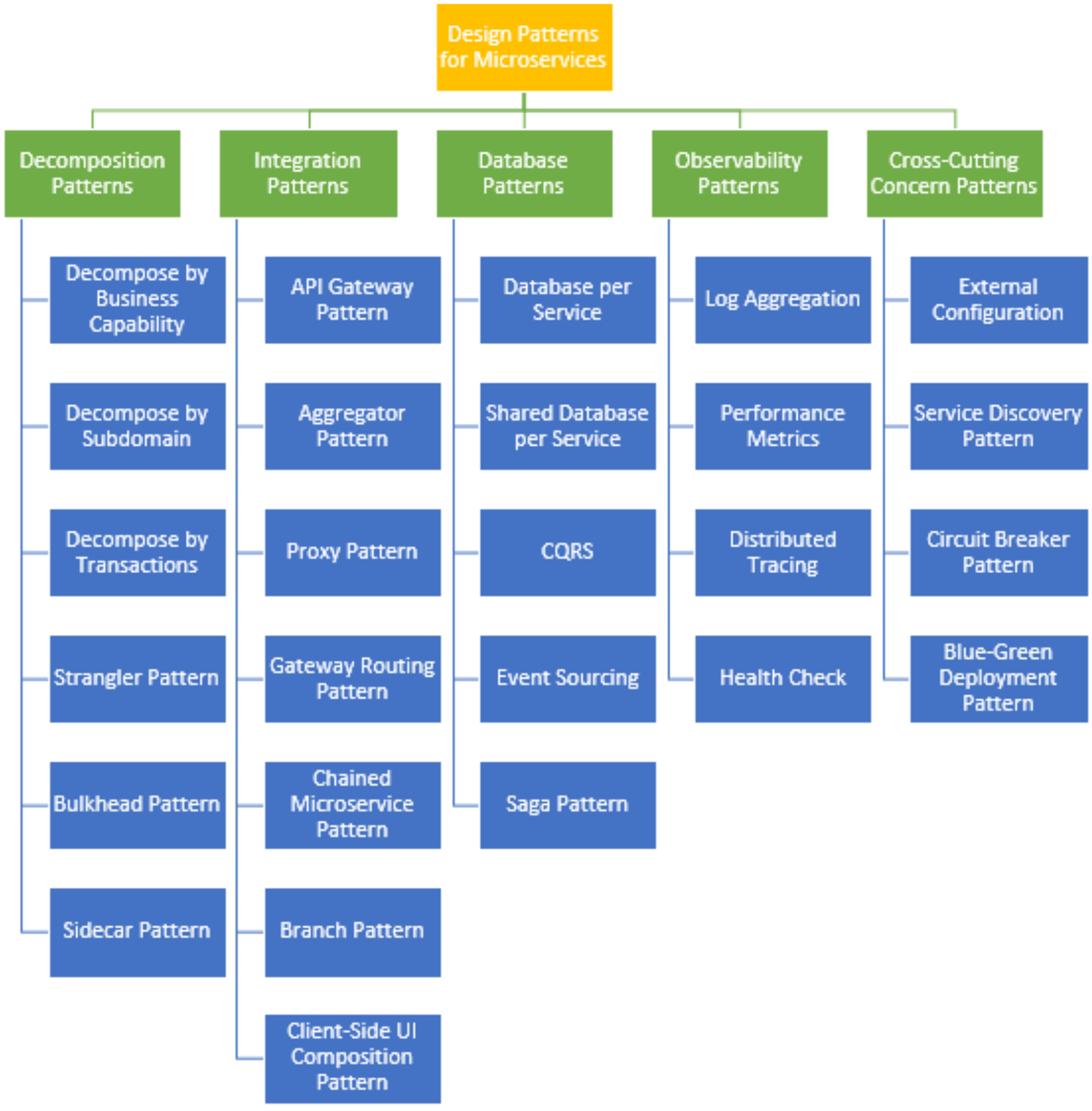


4. Development time

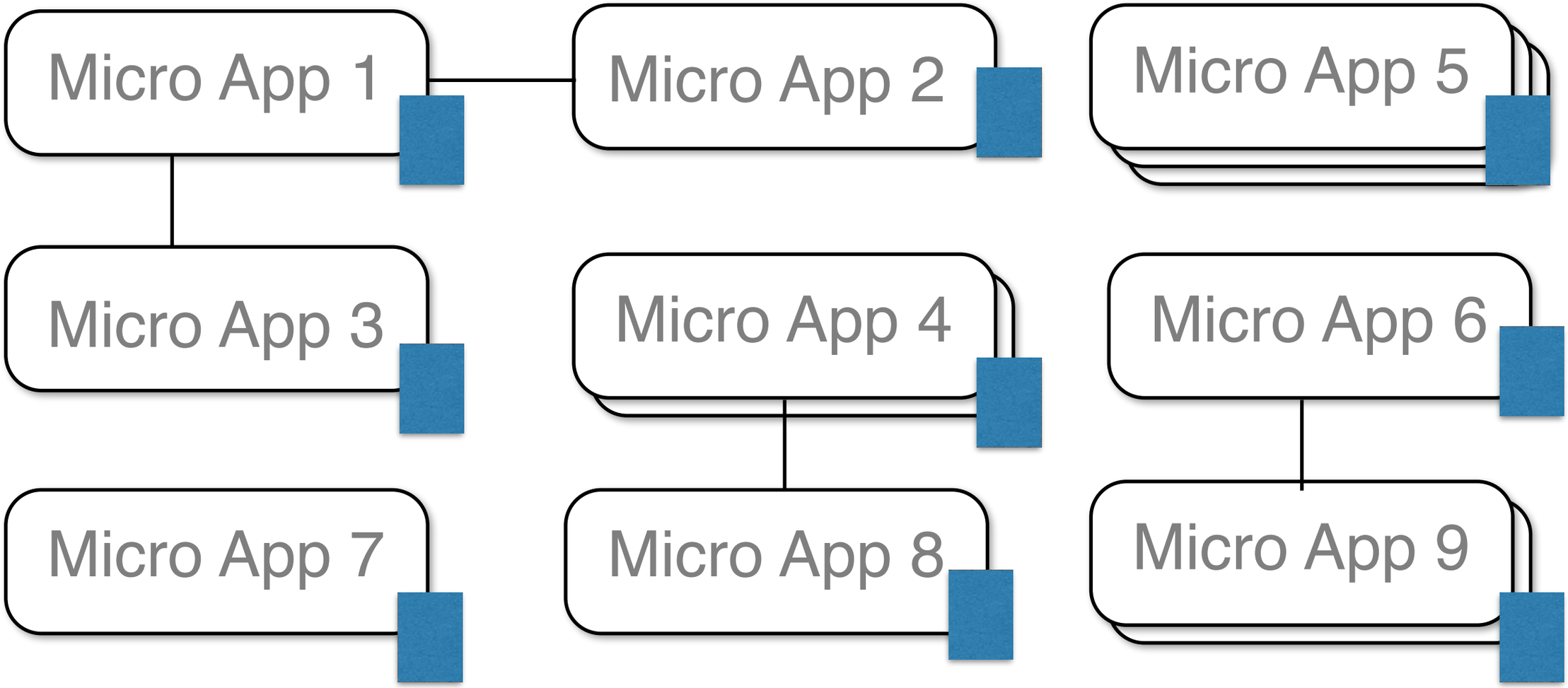




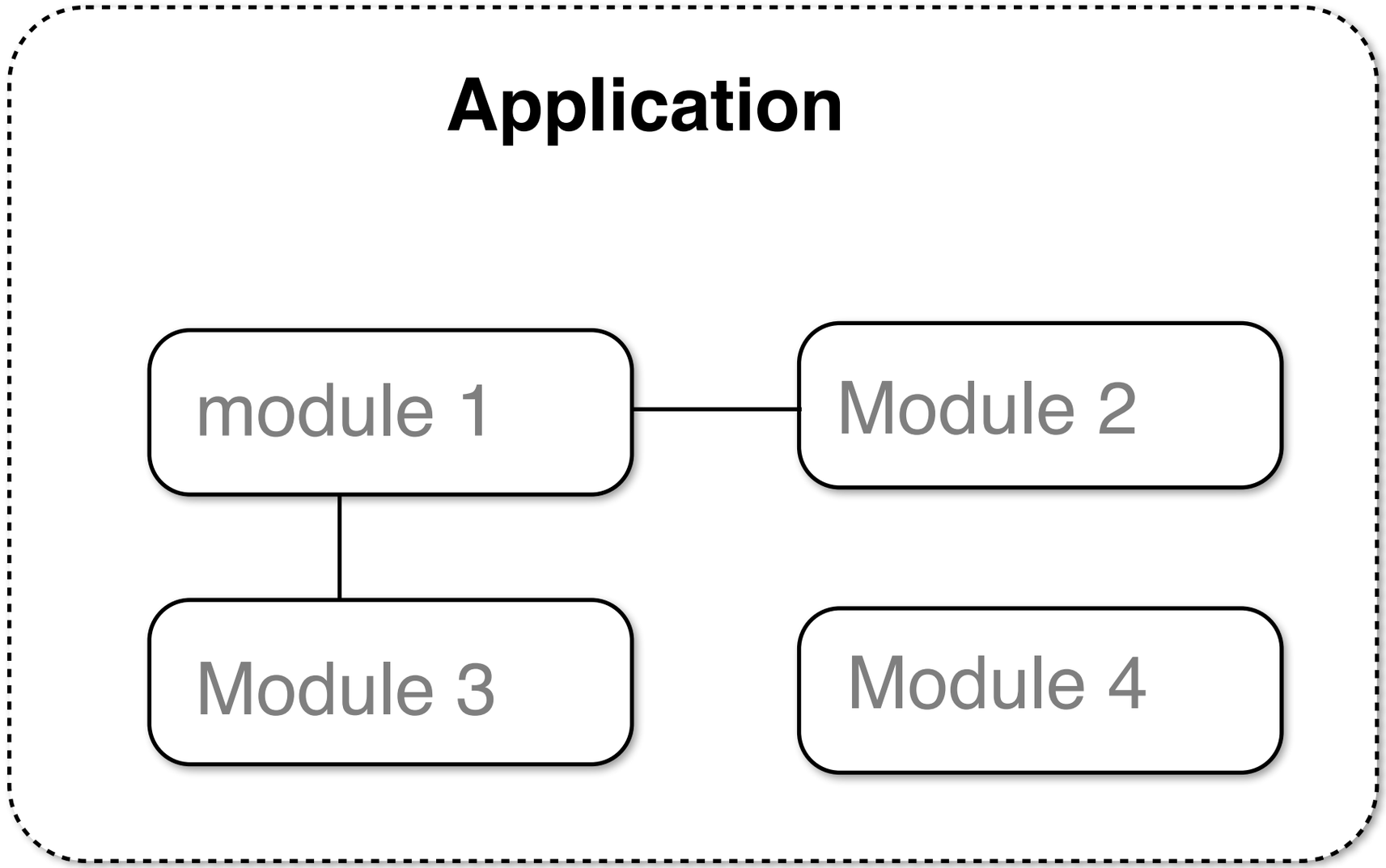
5. Learning Curve

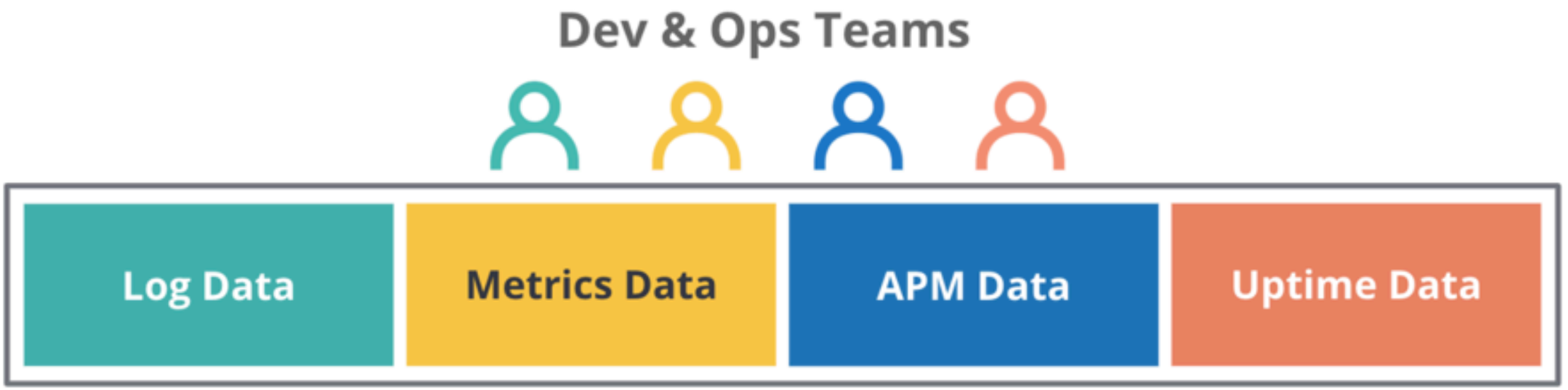


6. Infra Cost

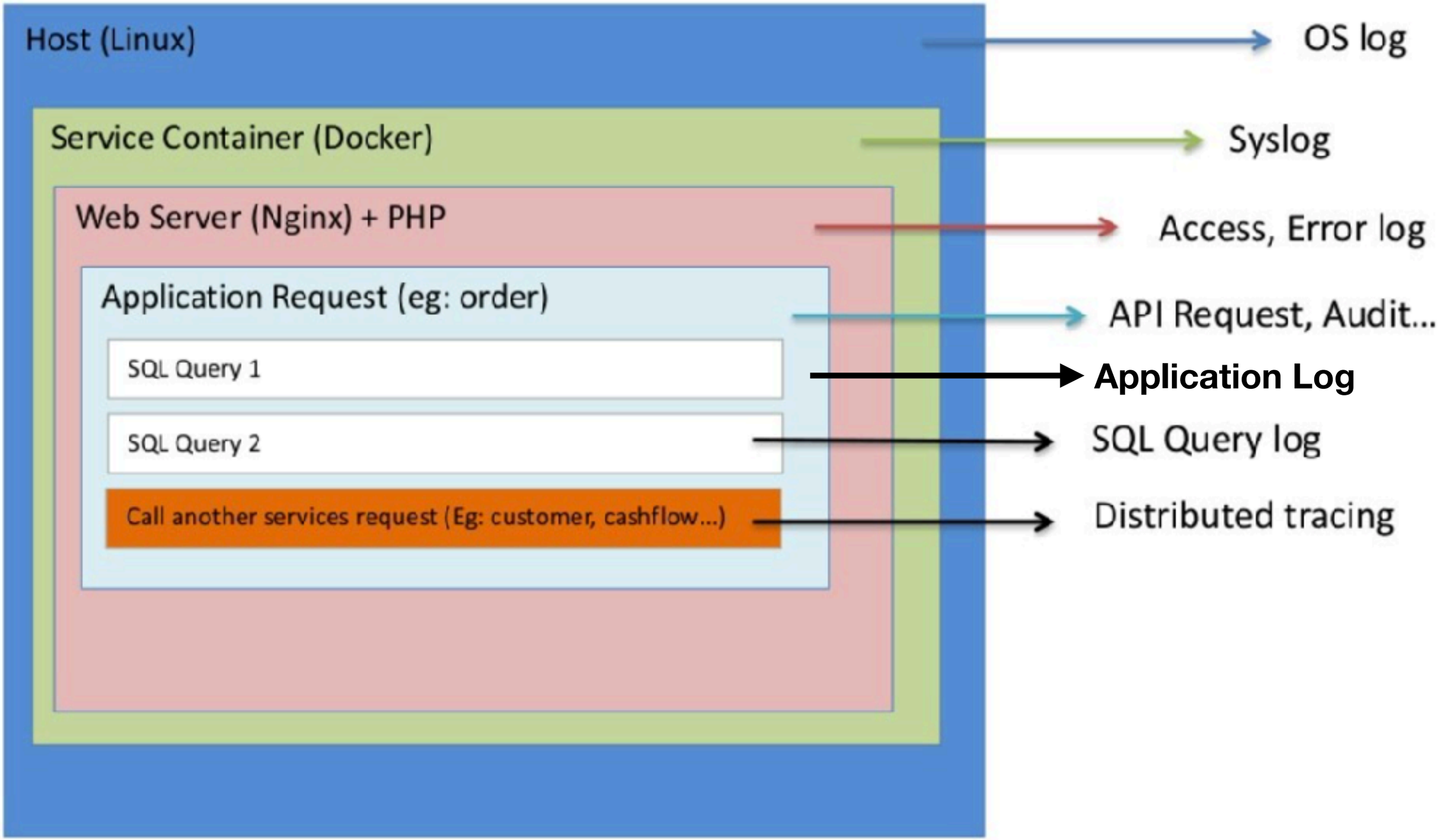

Micro App 1
Micro App 2
Micro App 5
Micro App 3
Micro App 4
Micro App 6
Micro App 7
Micro App 8
Micro App 9

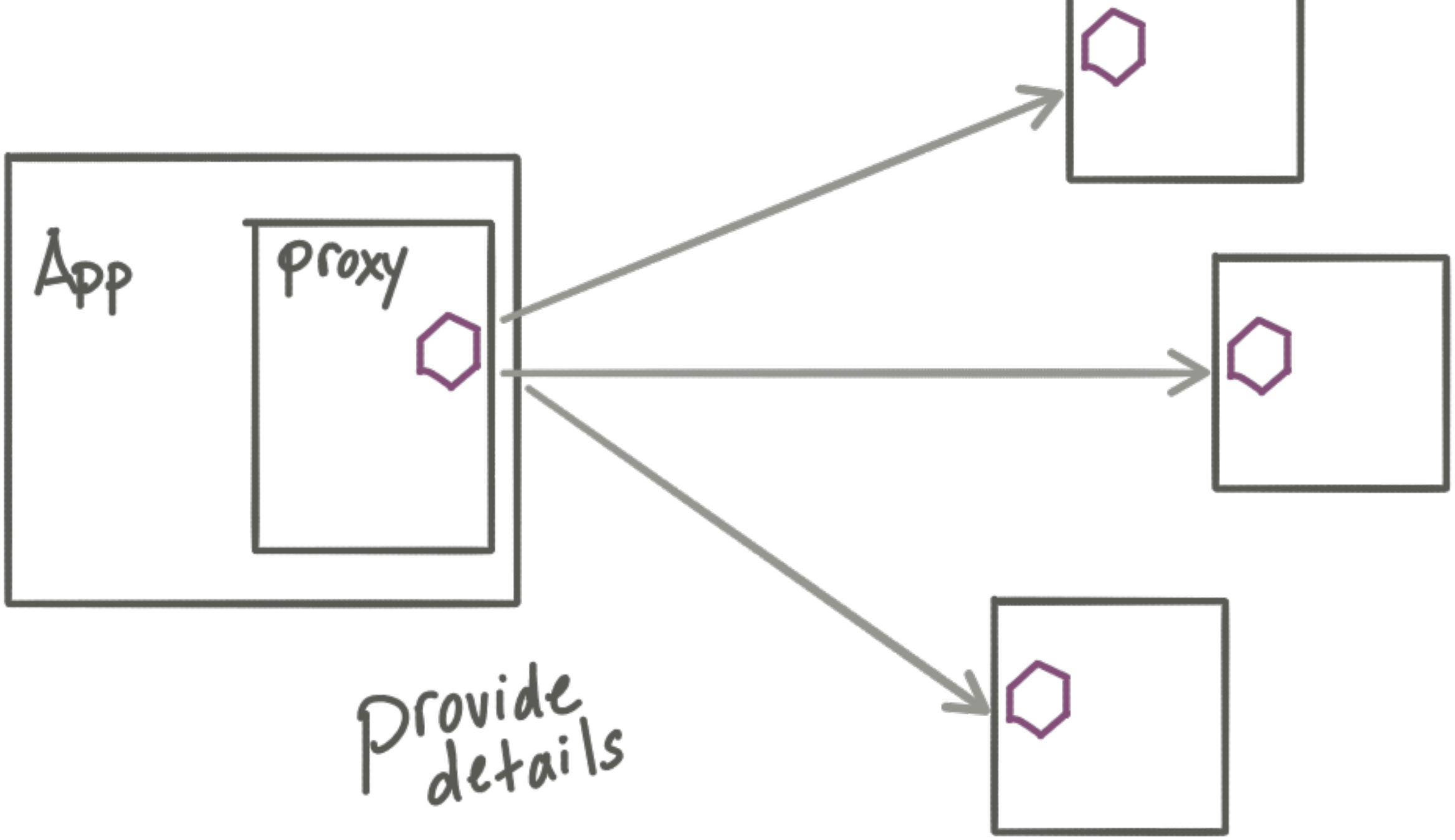
7. Debugging



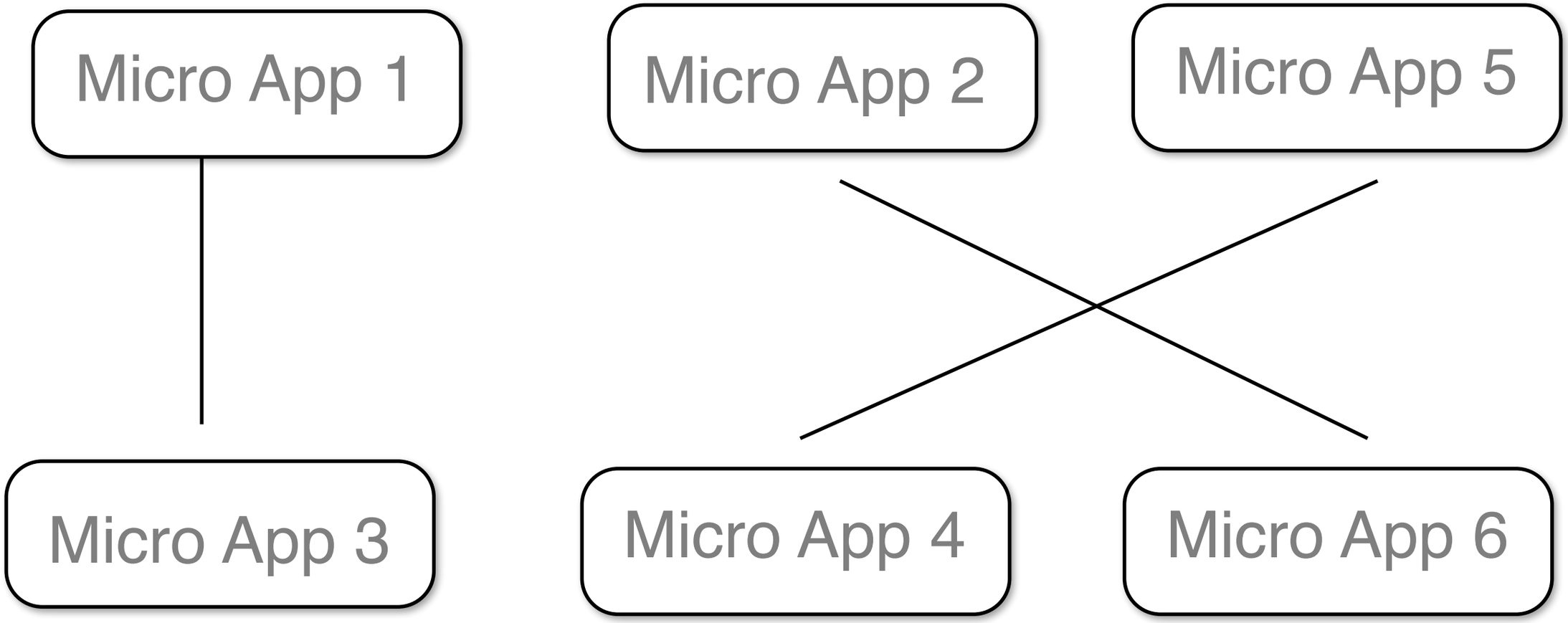


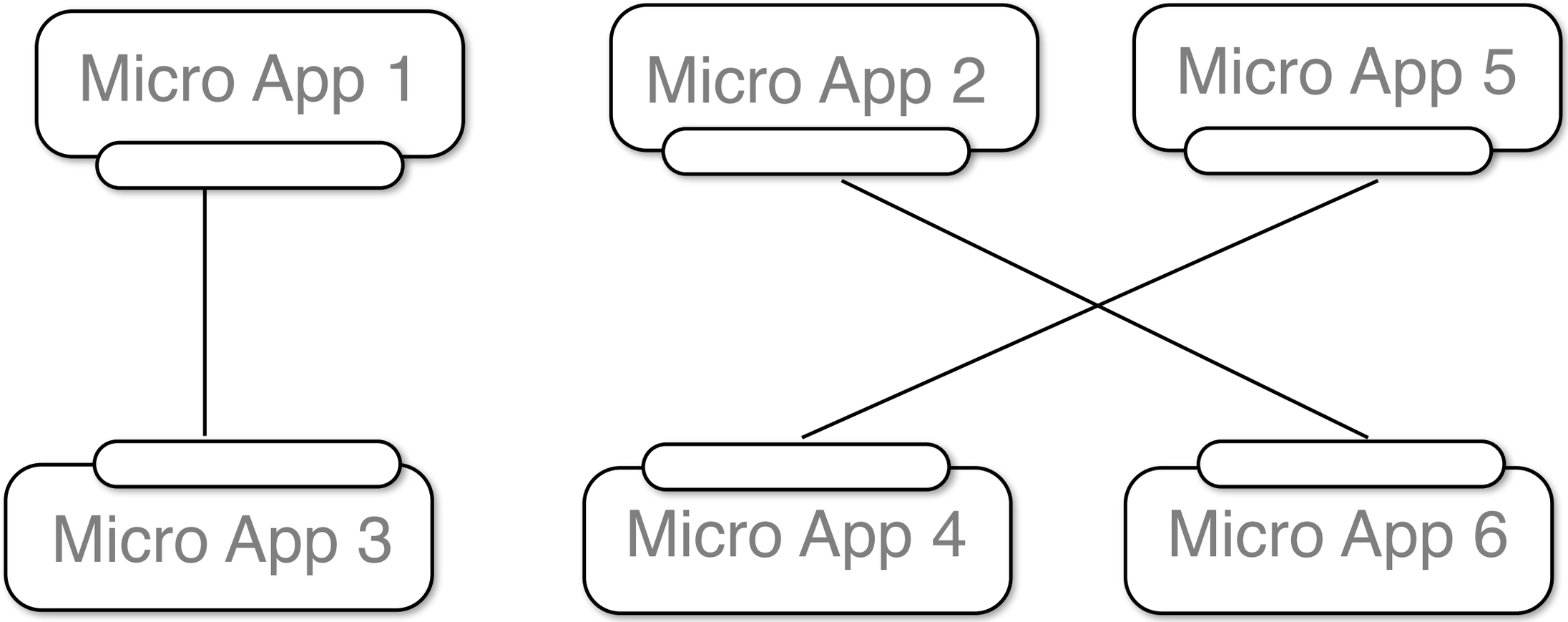
Web Logs	Container Metrics	Real User Monitoring	Uptime
App Logs	Host Metrics	Txn Perf Monitoring	Response Time
Database Logs	Database Metics	Distributed Tracing	
Container Logs	Network Metrics		
	Storage Metrics		

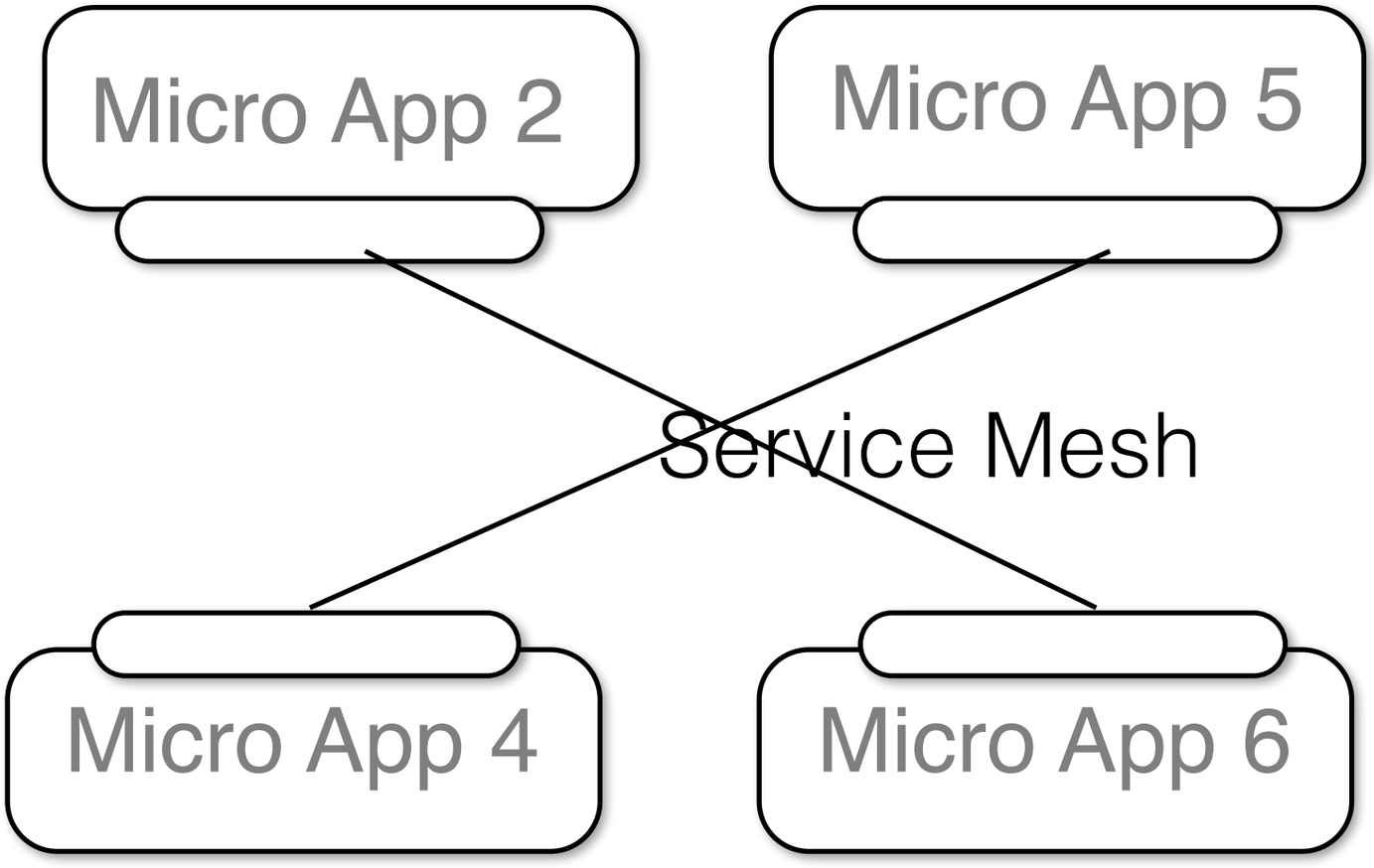

Host (Linux)
Service Container (Docker)
Web Server (Nginx) + PHP
Application Request (eg: order)
SQL Query 1
SQL Query 2
Call another services request (Eg: customer, cashflow...)
OS log
Syslog
Access, Error log
API Request, Audit...
Application Log
SQL Query log
Distributed tracing

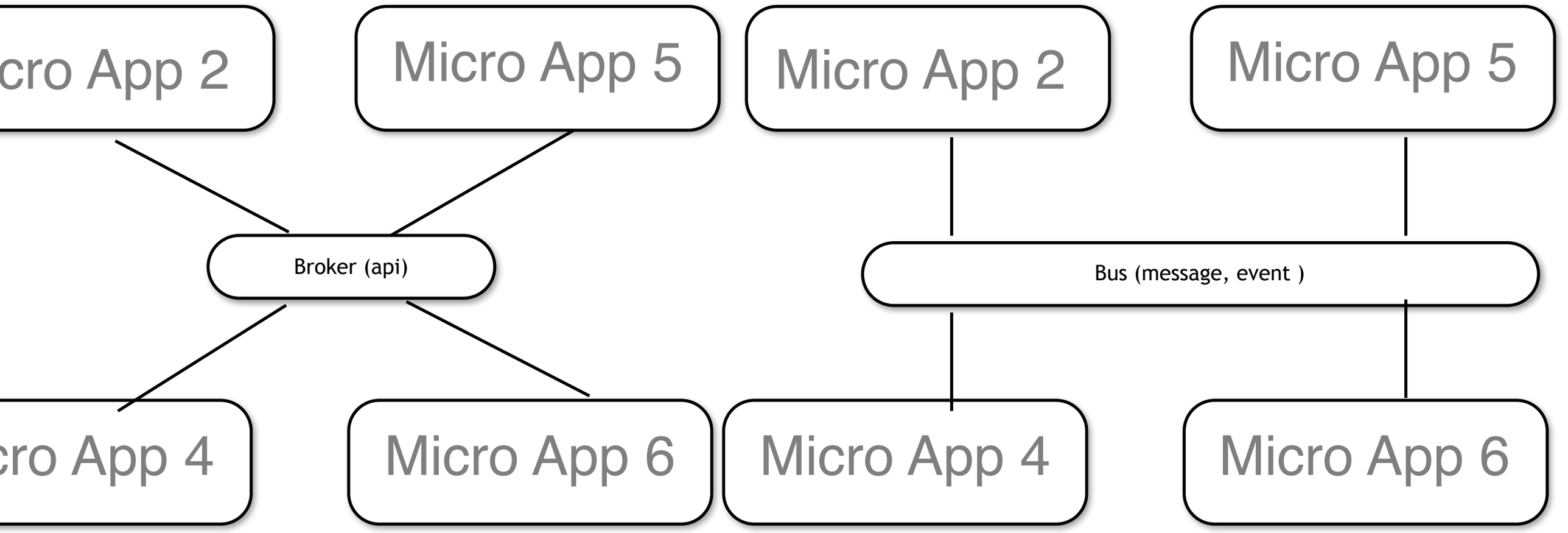


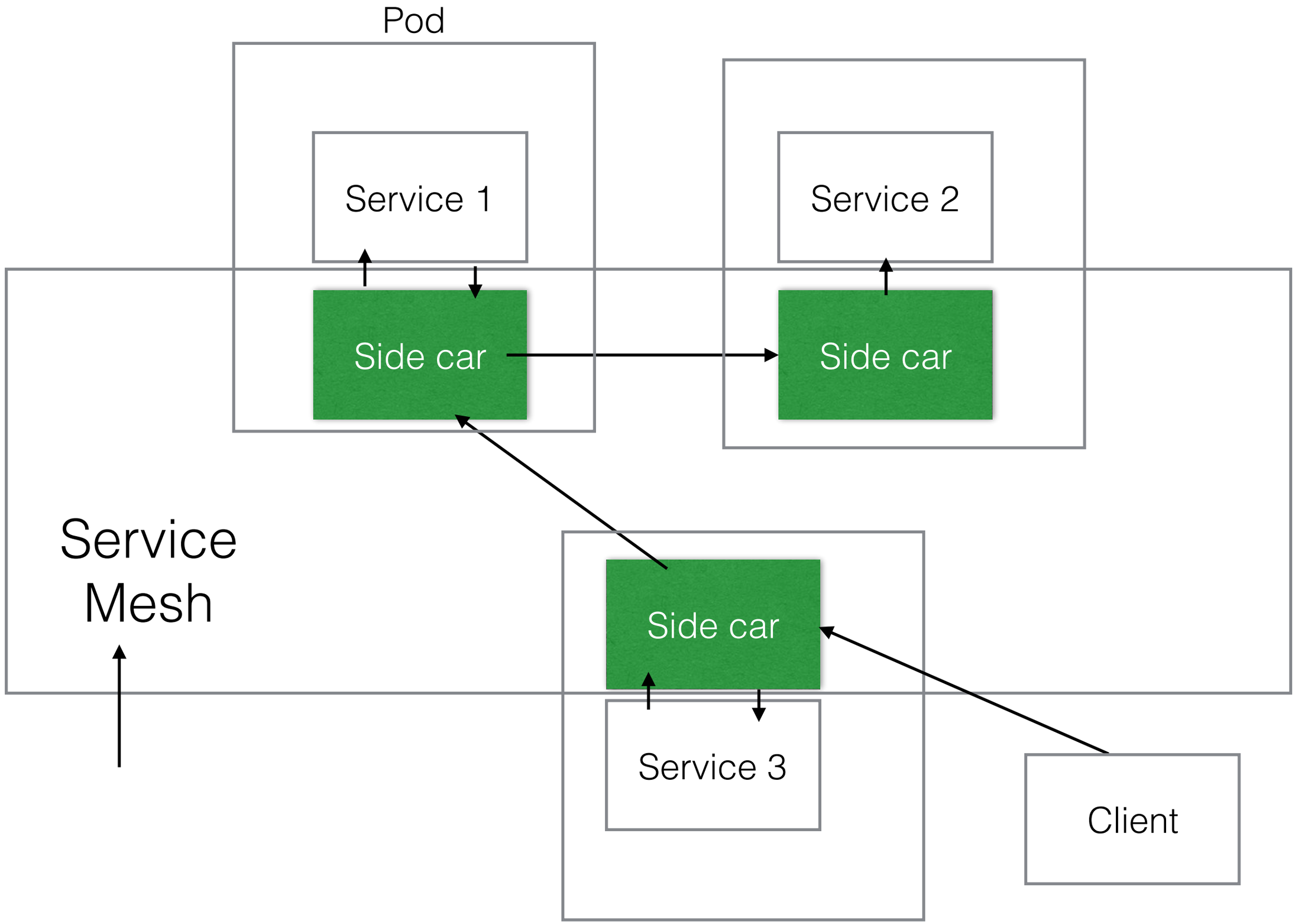
Service Mesh

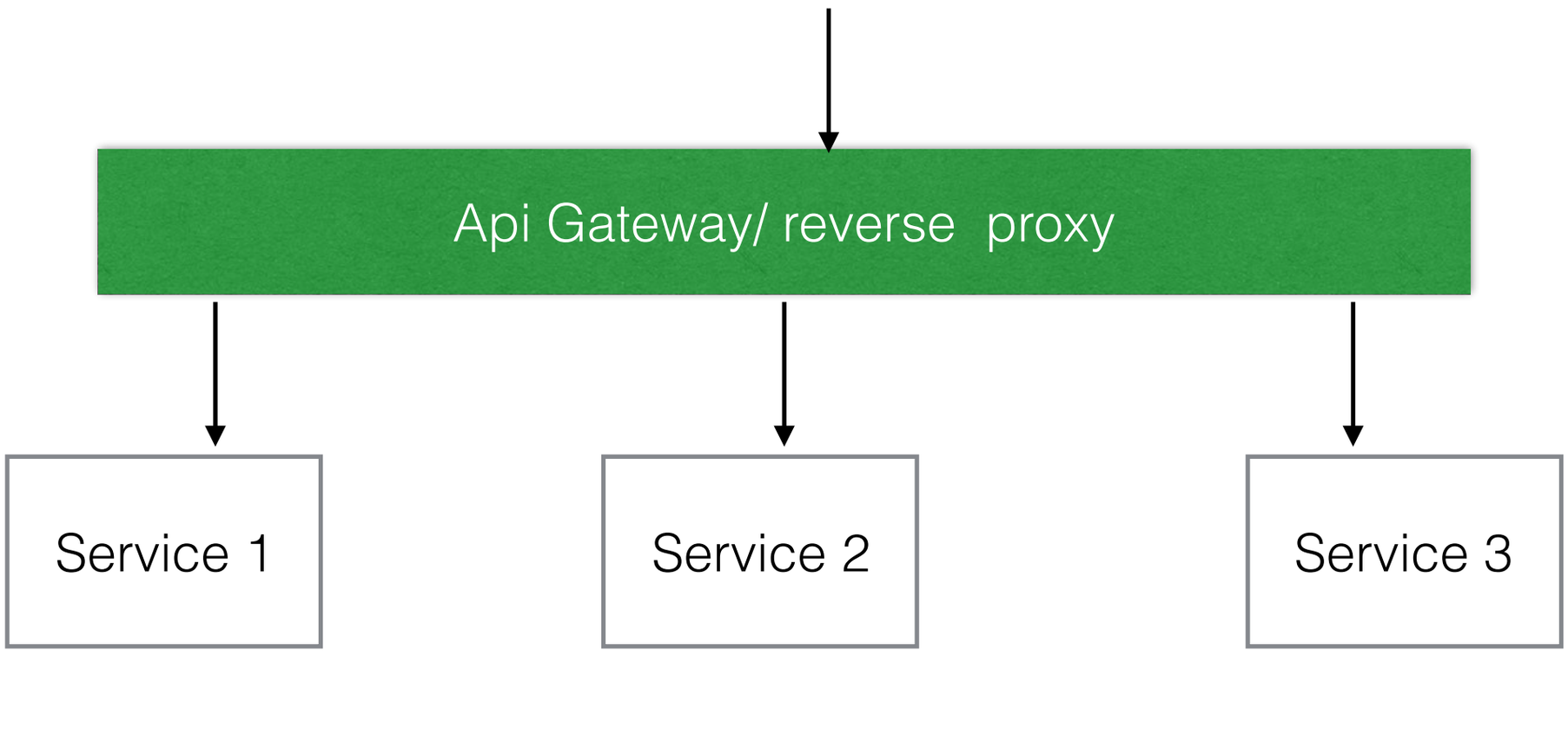

Micro App 1
Micro App 2
Micro App 5
Micro App 3
Micro App 4
Micro App 6


Micro App 1
Micro App 2
Micro App 5
Micro App 3
Micro App 4
Micro App 6


Micro App 2
Micro App 5
Service Mesh
Micro App 4
Micro App 6


cro App 2
Micro App 5
Micro App 2
Micro App 5
Broker (api)
Bus (message, event)
cro App 4
Micro App 6
Micro App 4
Micro App 6


Pod
Service 1
Service 2
Side car
Side car
Service
Mesh
Side car
Service 3
Client


Api Gateway/ reverse proxy
Service 1
Service 2
Service 3

Service 1

Service 2

Service 3

API

2

1

Service 1

Service 2

Message Q

Service 3

Event Driven Arch *

3

4

Service 1

Service 2

Ingress /reverse proxy/ api gateway

North-south

Service 3

Broker Arch *

Side car

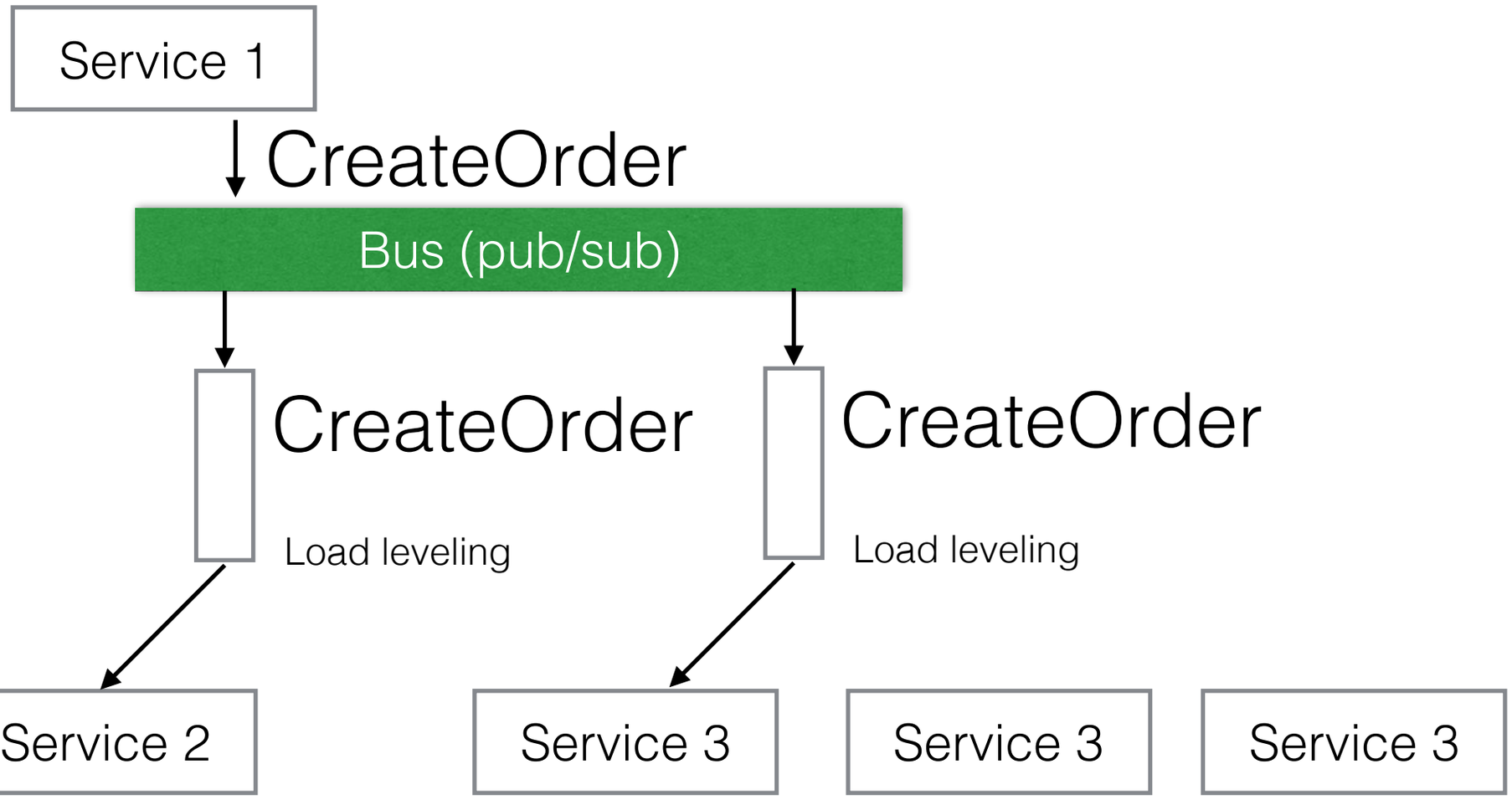
Side car

Service Mesh

Side car

Service 3

Mesh Arch *


Service 1
CreateOrder
Bus (pub/sub)
CreateOrder
Load leveling
CreateOrder
Load leveling
Service 2
Service 3
Service 3
Service 3

Message Q

Pod

Pod

J Pod

Service 1

Service 2

Service 4

Side car

J

Side car

J

Service Mesh

Side car

J

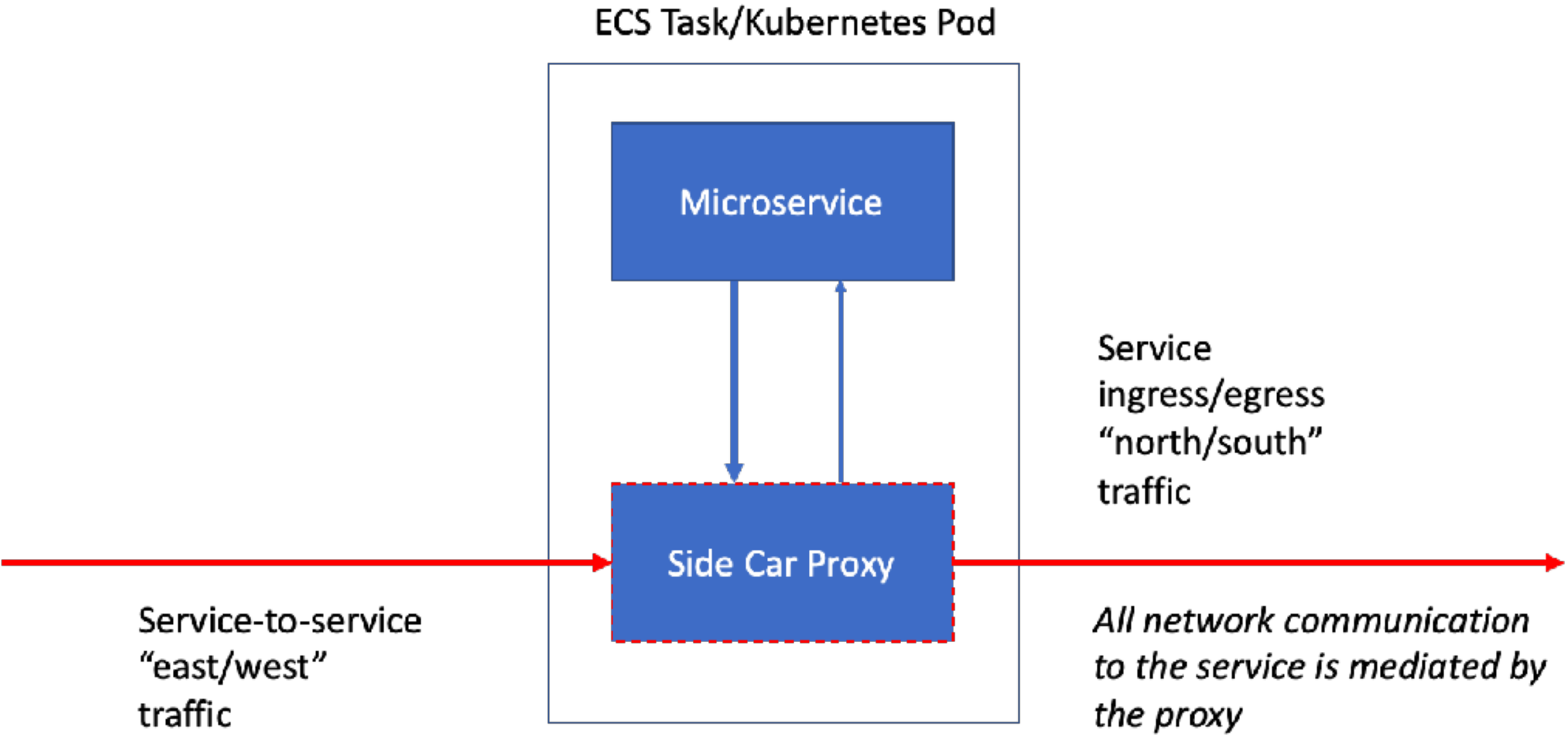
Service 3

Pod

J

Ingress/reverse proxy

Load Balancer

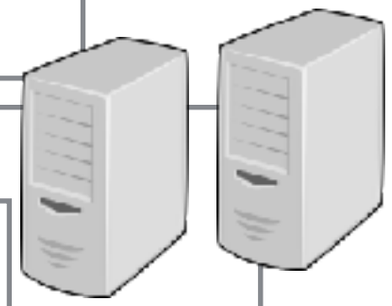

ECS Task/Kubernetes Pod
Microservice
Side Car Proxy
Service
ingress/egress
“north/south”
traffic
Service-to-service
“east/west”
traffic
All network communication
to the service is mediated by
the proxy

http://172.17.0.22:80/date

North- South

LoadBalancer : L4 (tcp)

Cluster



Ingress : L7 (http)

Virtual Service

http://10.97.33.128:8080/date

Service

East - west

Side car

Microservice
POD

Edge Ingres

Service Mesh Control Plane

Service Mesh data plane

Integration

Integration

Integration

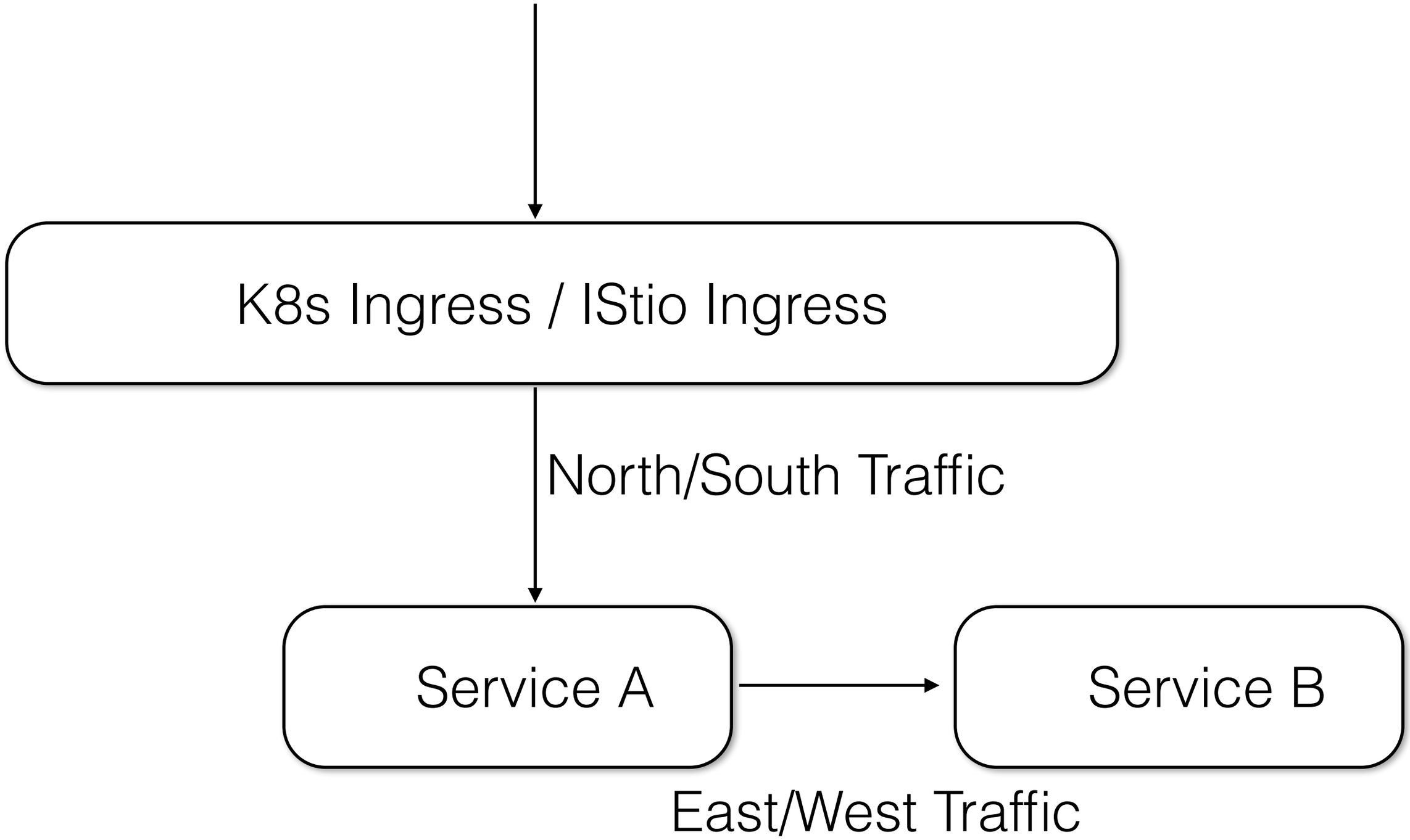
Integration

Integration

North-South Traffic

East-West traffic

Istio


K8s Ingress / IStio Ingress
North/South Traffic
Service A
Service B
East/West Traffic

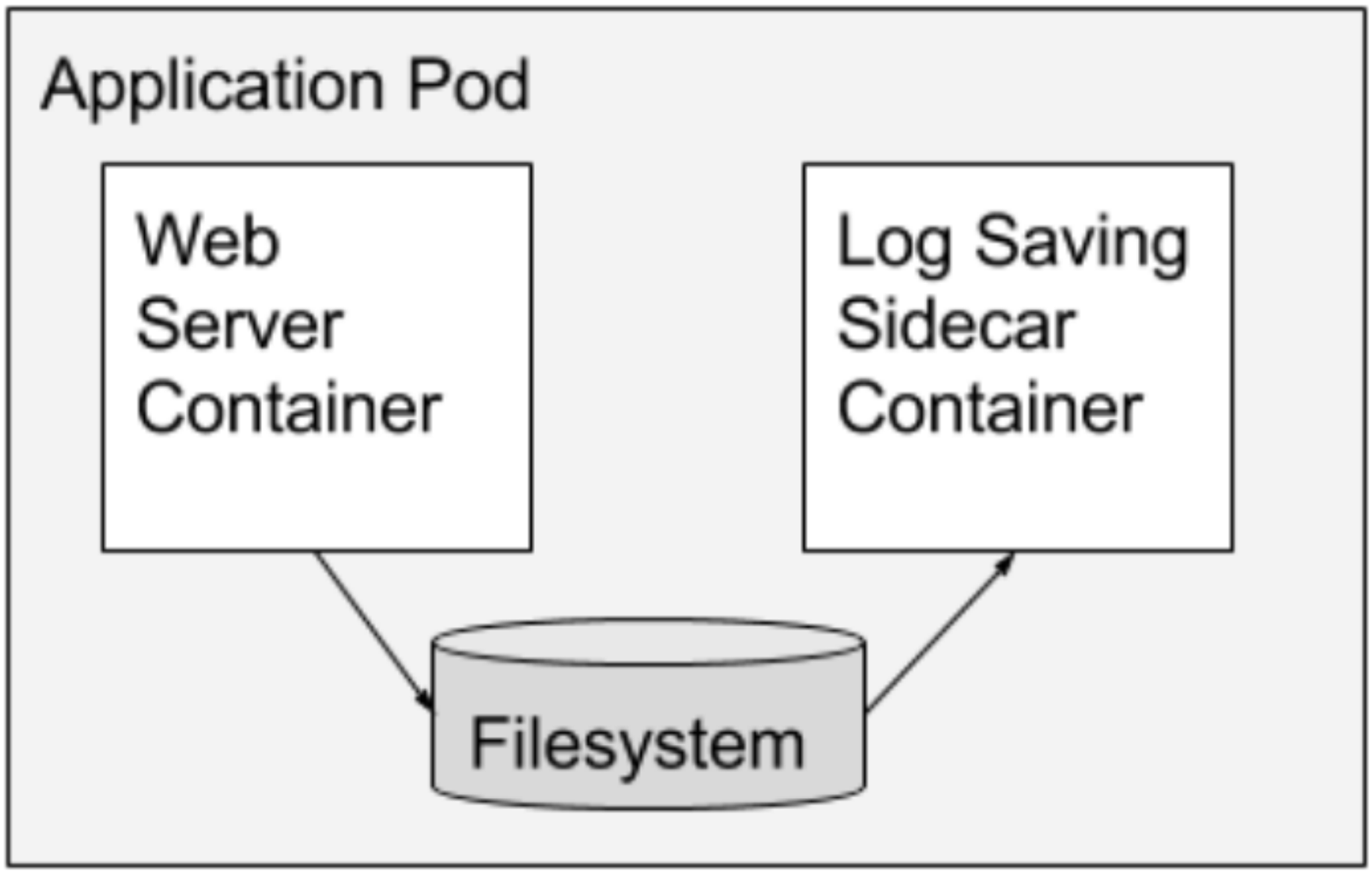
Cross Cutting Concerns

Domain Logic

Authorization
Exception Handling
Versioning
Tracing
Protocol Translation
…

Proxy

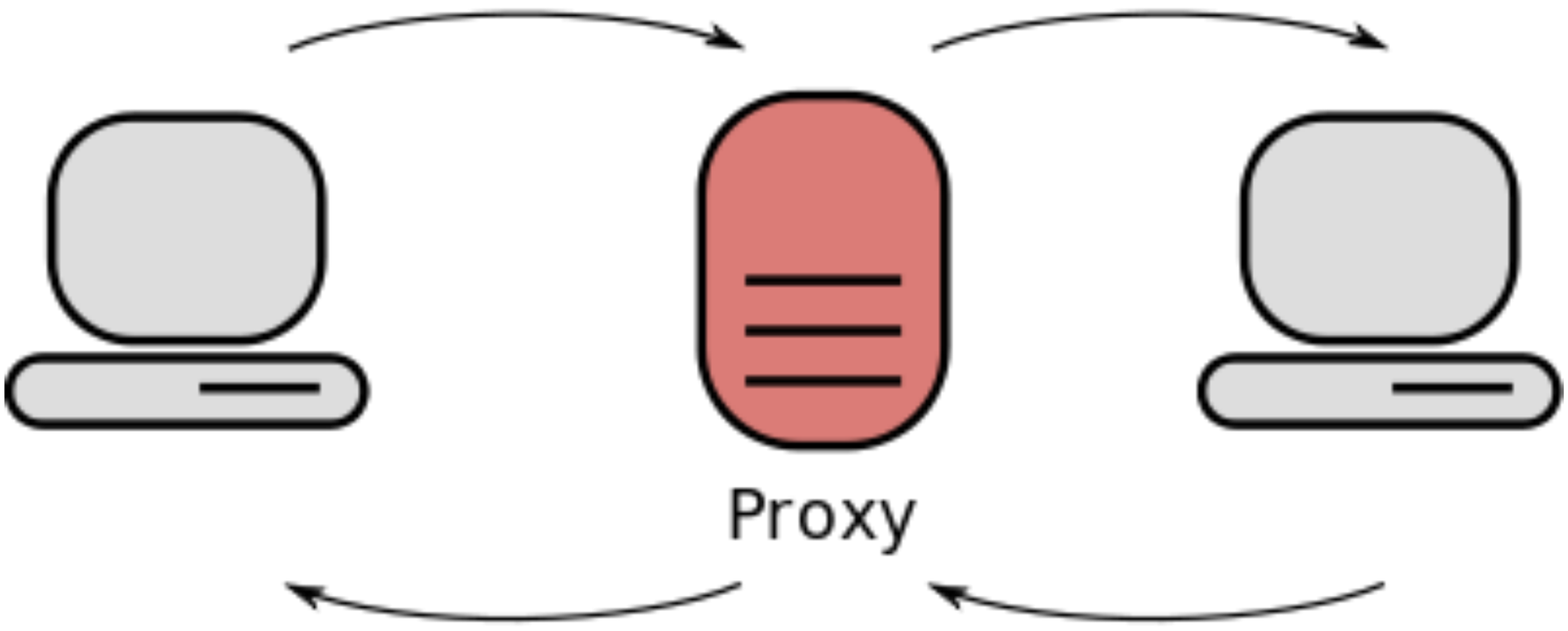
Sidecar


Application Pod
Web
Server
Container
Log Saving
Sidecar
Container
Filesystem

- In a sidecar pattern, the functionality of the main container is extended or enhanced by a sidecar container without strong coupling between two.

- eg. main container is a web server which is paired with a log saver sidecar container that collects the web server’s logs from local disk and streams them to centralized log collector.

Proxy pattern



- Proxy pretends being an object when in fact it's not. There are several types of proxies:

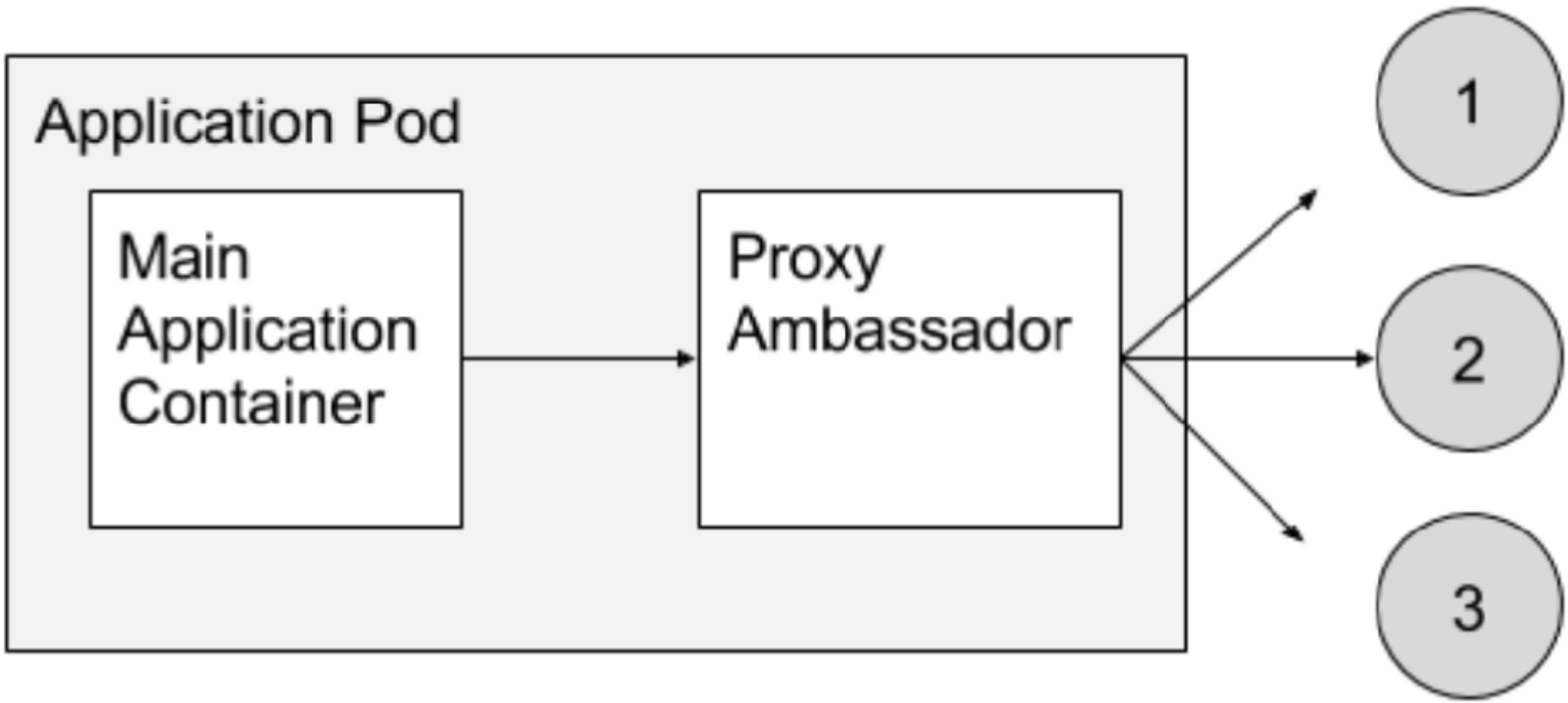
- Remote proxy ("Ambassador") - instead of communicating between server's and client's classes we create a proxy on client side and server side, and only the proxies communicate

- Virtual proxy - creates expensive object on first demand

- Protection proxy - to control access

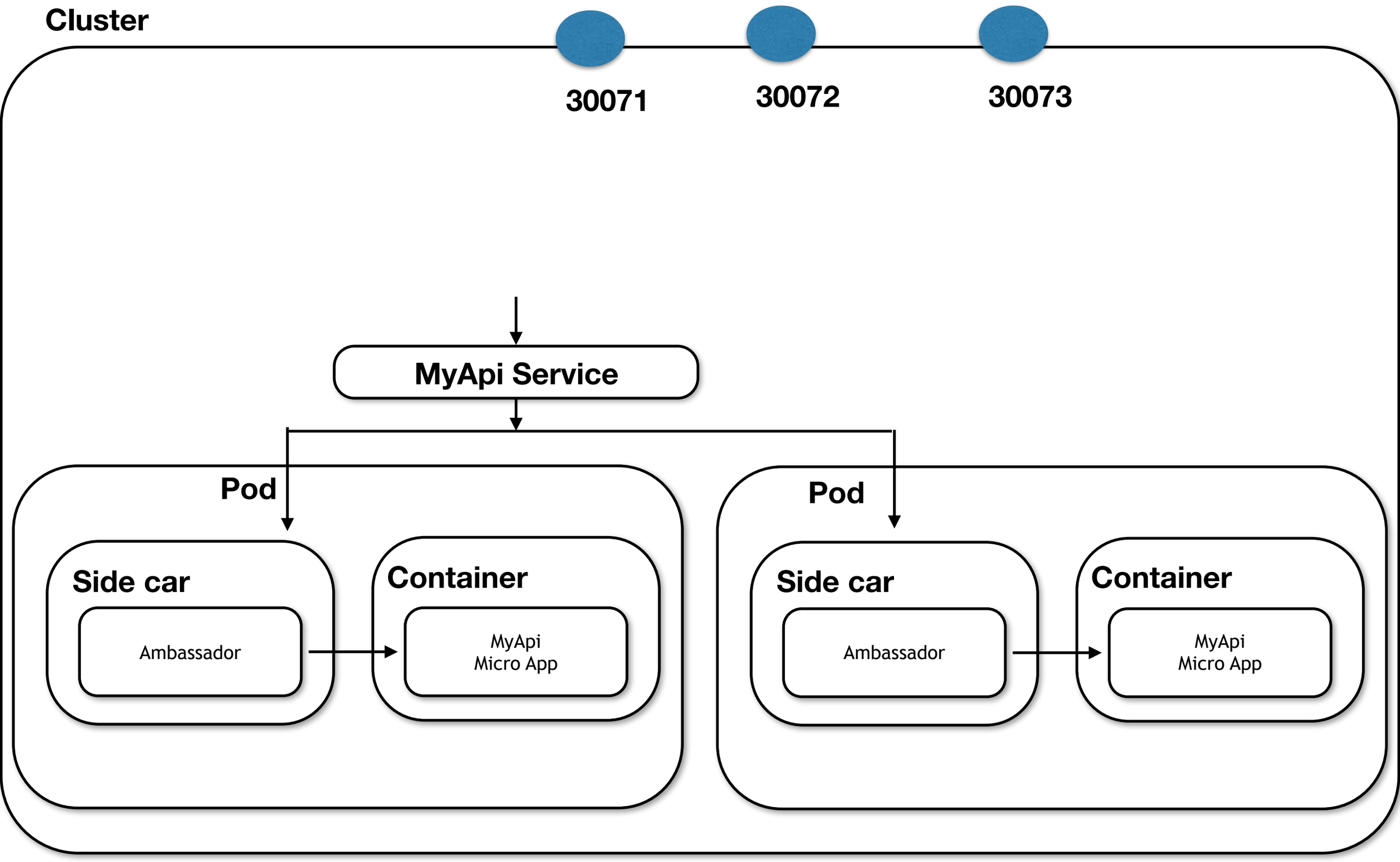
- Smart proxy - enrich an object

Ambassador pattern



- an Ambassador container act as a proxy between two different types of main containers. Typical use case involves proxy communication related to load balancing and/or sharding to hide the complexity from the application.

Deployment Diagram



Deployment Diagram

API Gateway

Network Load Balancer

Node Port

Node Port

Kubernetis Ingress/ Istio Ingress / nginx /...(reverse proxy)

MyApi Service

Pod

Side car

Ambassador

Container

MyApi
Micro App

Pod

Side car

Ambassador

Container

MyApi
Micro App

Service 1

Service 2

Service 3

API

1

Service 1

Service 2

Message Q

Service 3

2

Event Driven Arch *

3

Service 1

Service 2

Ingress /reverse proxy/ api gateway

Service 3

Broker Arch *

4

Service 1

Service 2

Side car

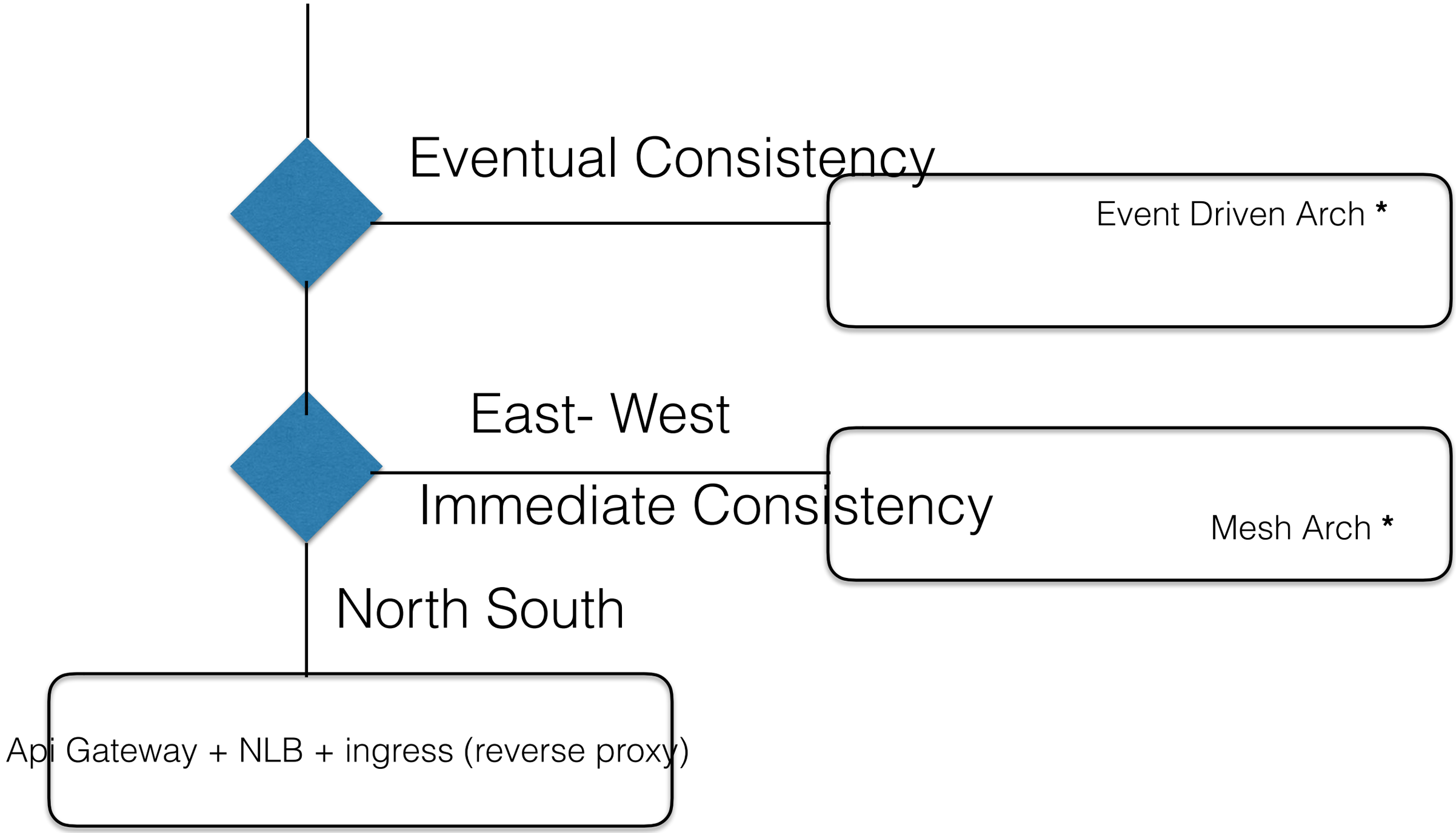
Side car

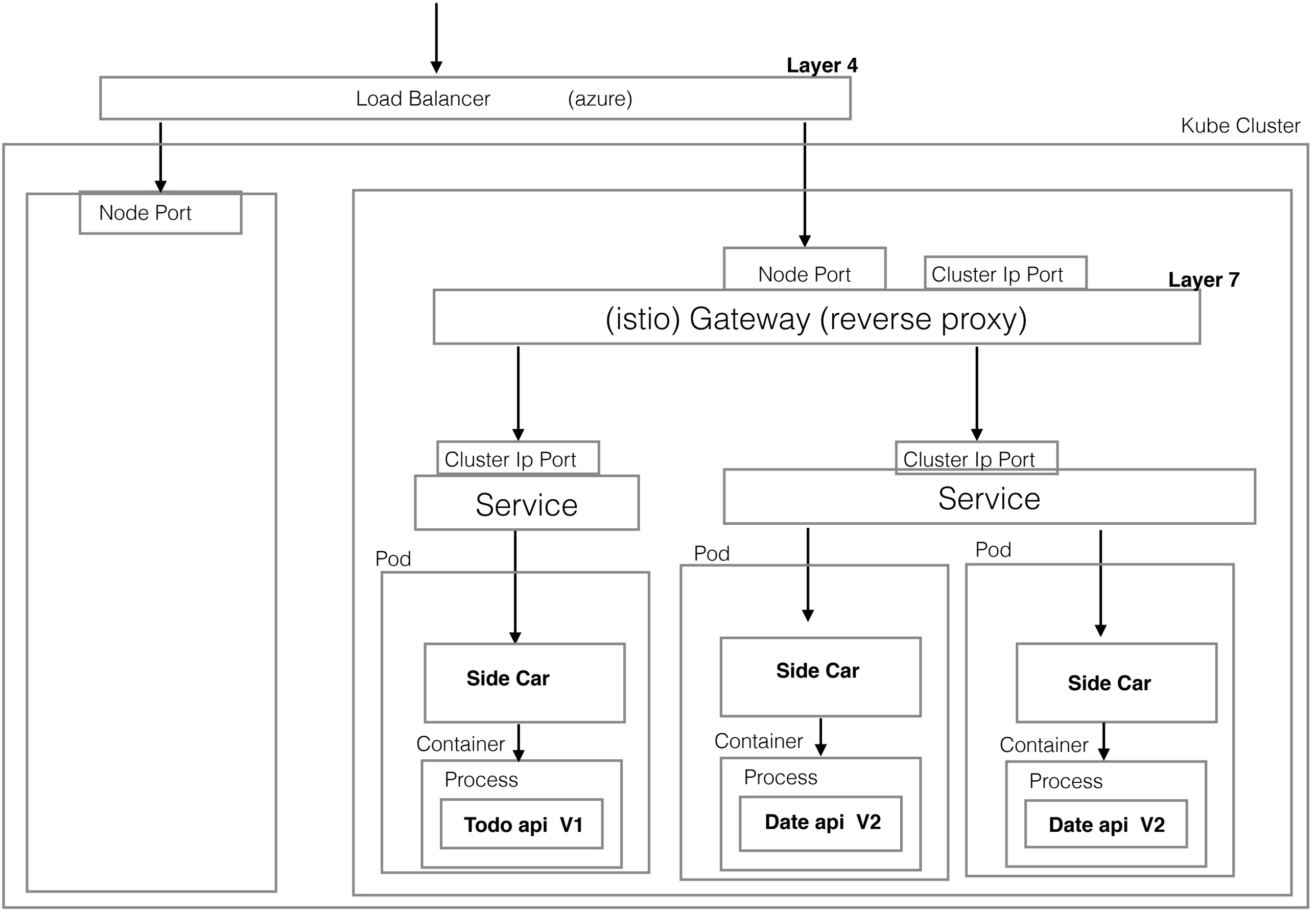
Service Mesh

Side car

Service 3

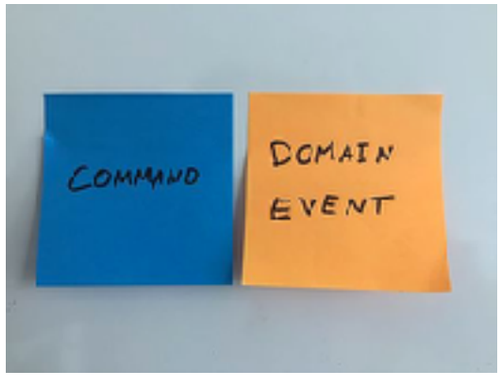
Mesh Arch *


Eventual Consistency
Event Driven Arch *
East- West
Immediate Consistency
Mesh Arch *
North South
Api Gateway + NLB + ingress (reverse proxy)


Layer 4
Load Balancer (azure)
Kube Cluster
Node Port
Node Port
Cluster Ip Port
Layer 7
(istio) Gateway (reverse proxy)
Cluster Ip Port
Service
Pod
Side Car
Container
Process
Todo api V1
Cluster Ip Port
Service
Pod
Side Car
Container
Process
Date api V2
Pod
Side Car
Container
Process
Date api V2

Event storming

Step 1: Create domain events



Step 2: Add the commands that caused the domain event

Step 2b: Add the actor that executes the command

Step 3: Add corresponding aggregate

Domain Event

Command

Aggregate

Issue

User Role

Claim Submission Self-Service

Submit claim (online or via mail)

Customer

Claim submitted

Claim

Check documentation (completeness)

Administrator in charge

Claim registered

Claim

Check insurance

Responsible person in claims department

Assessment performed

Accept claim and schedule payment

Responsible person in claims department

Payment scheduled

Claim

Reject claim

Responsible person in claims department

Claim rejected

Perform payment

Payment performed

Payment

Notification

Notify customer (by mail)

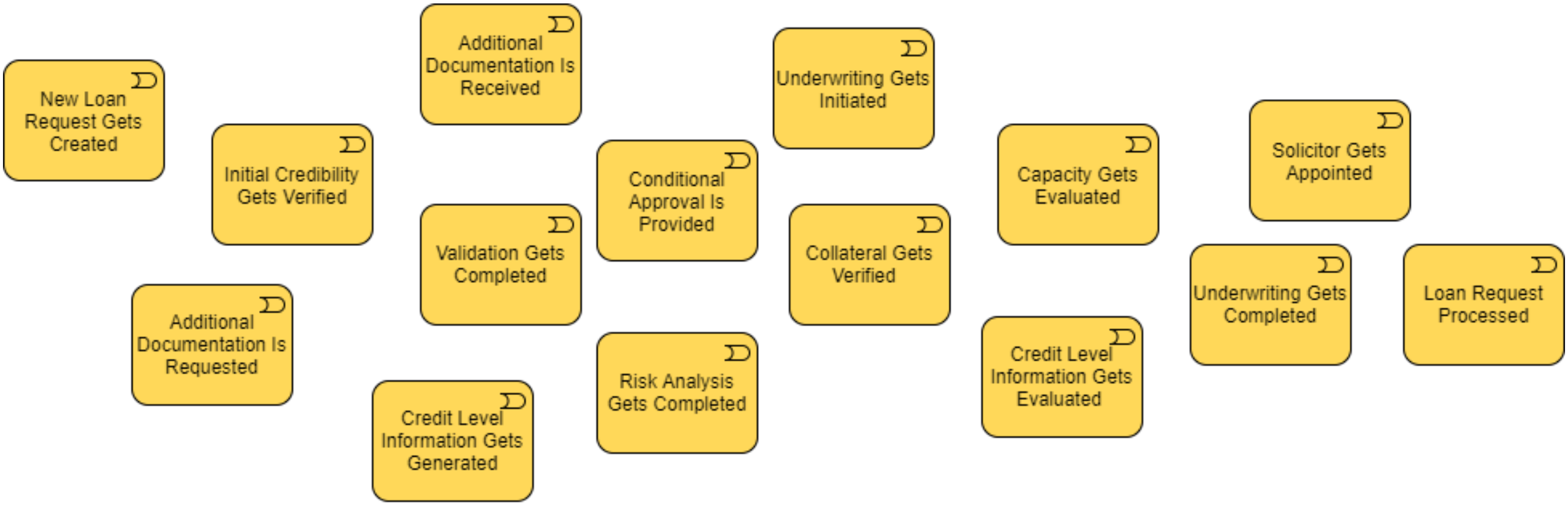
Customer notified

Realization in the Policy Management Context?

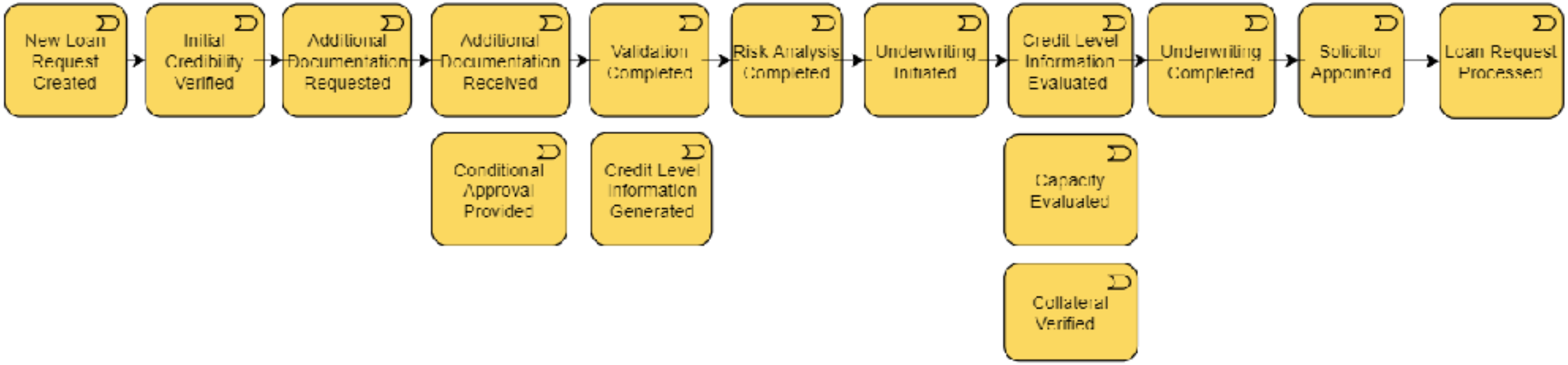
Or new Bounded Context?

https://github.com/getmubarak/Microservice/blob/master/case%20study/Loan%20Approval%20Process.md

Identification of Domain Events

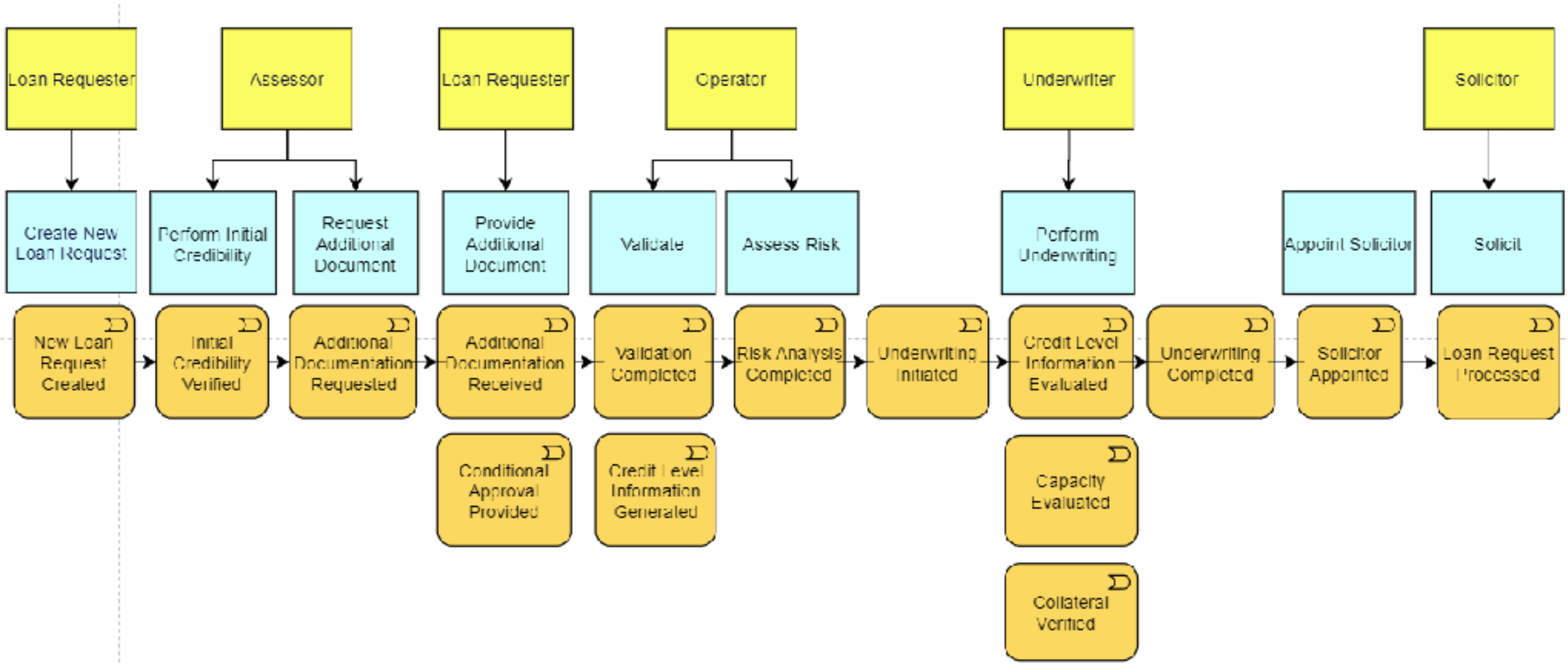


Time Sequencing of Domain Events



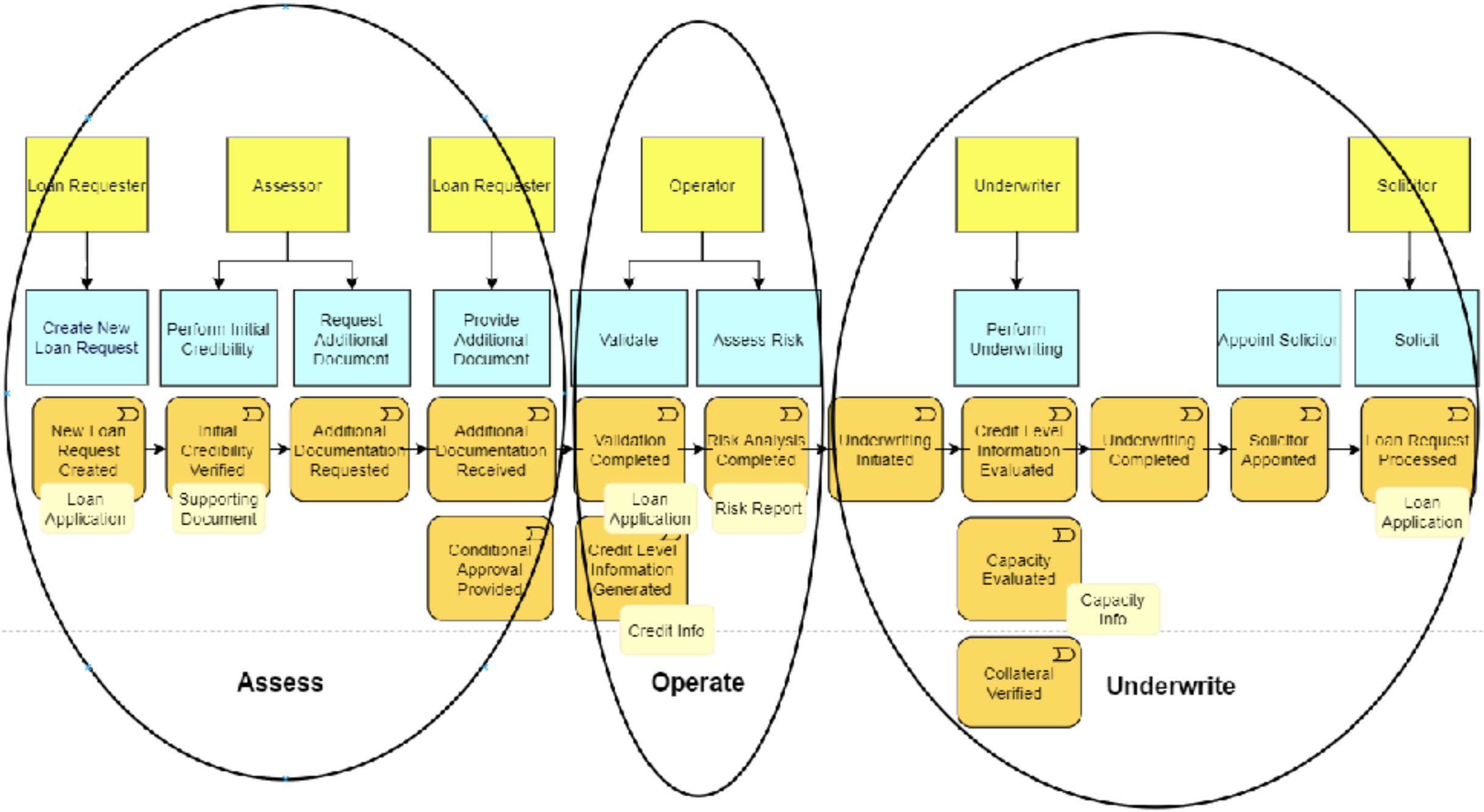
parallel have been stacked up vertically

Identification of Triggers/Commands

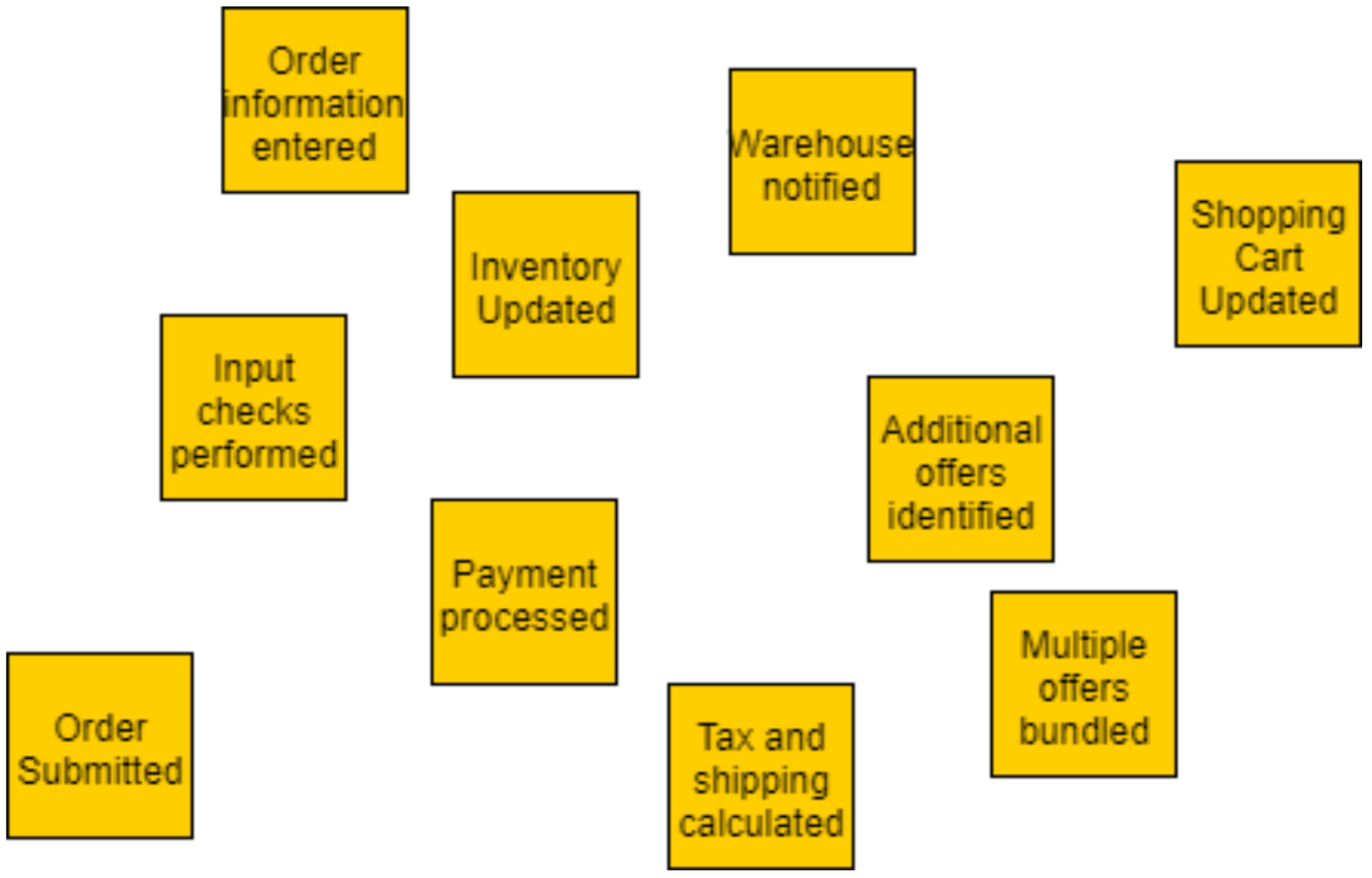


- Yellow represents the actors
- Blue represents the commands issued by the actors
- Orange are the events we started with

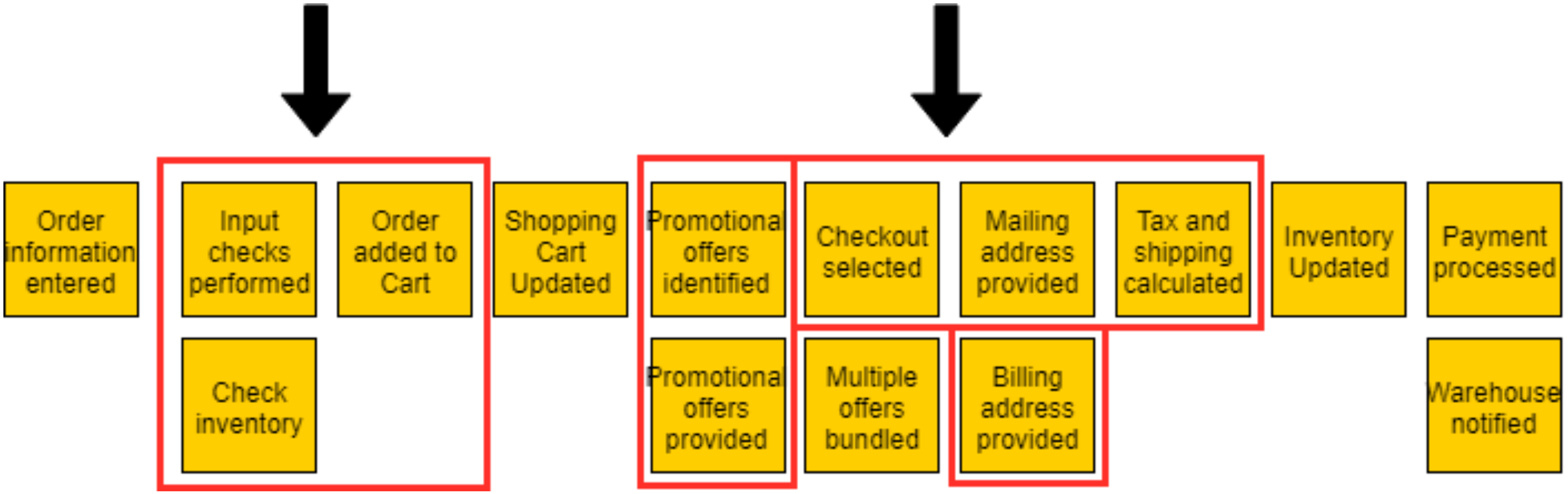
Identification of Aggregates



Step #1 — Event Discovery

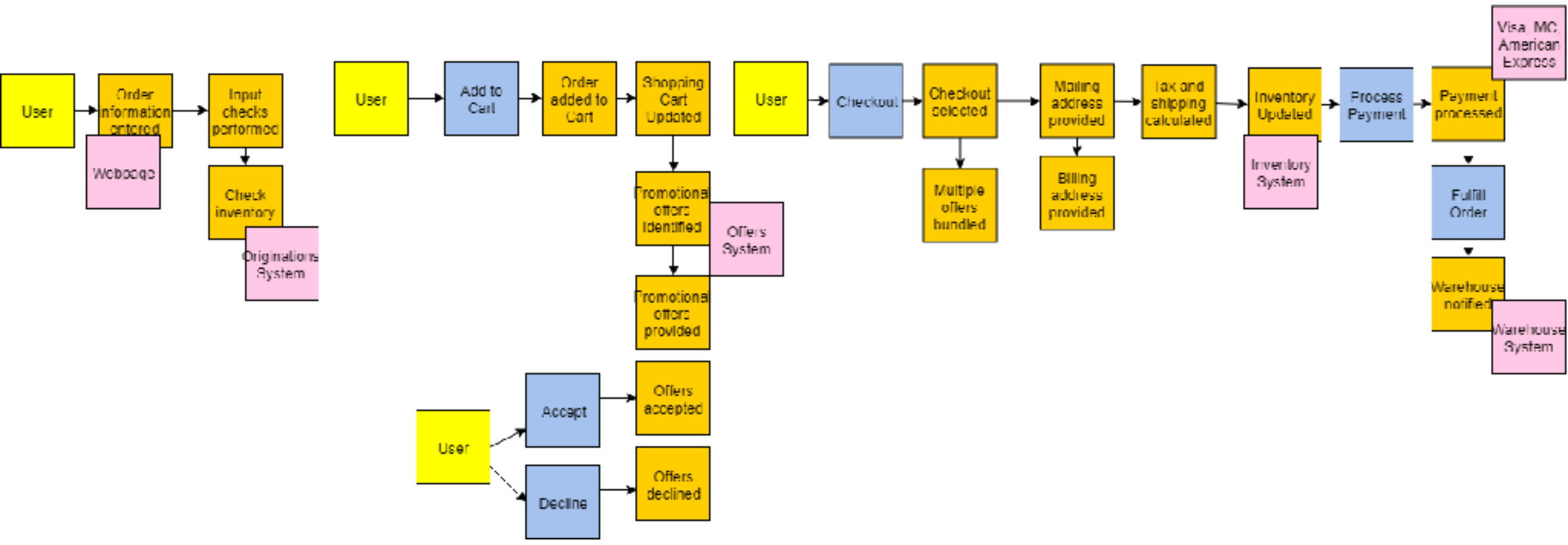


Step #2 — Placing the Events in Sequence

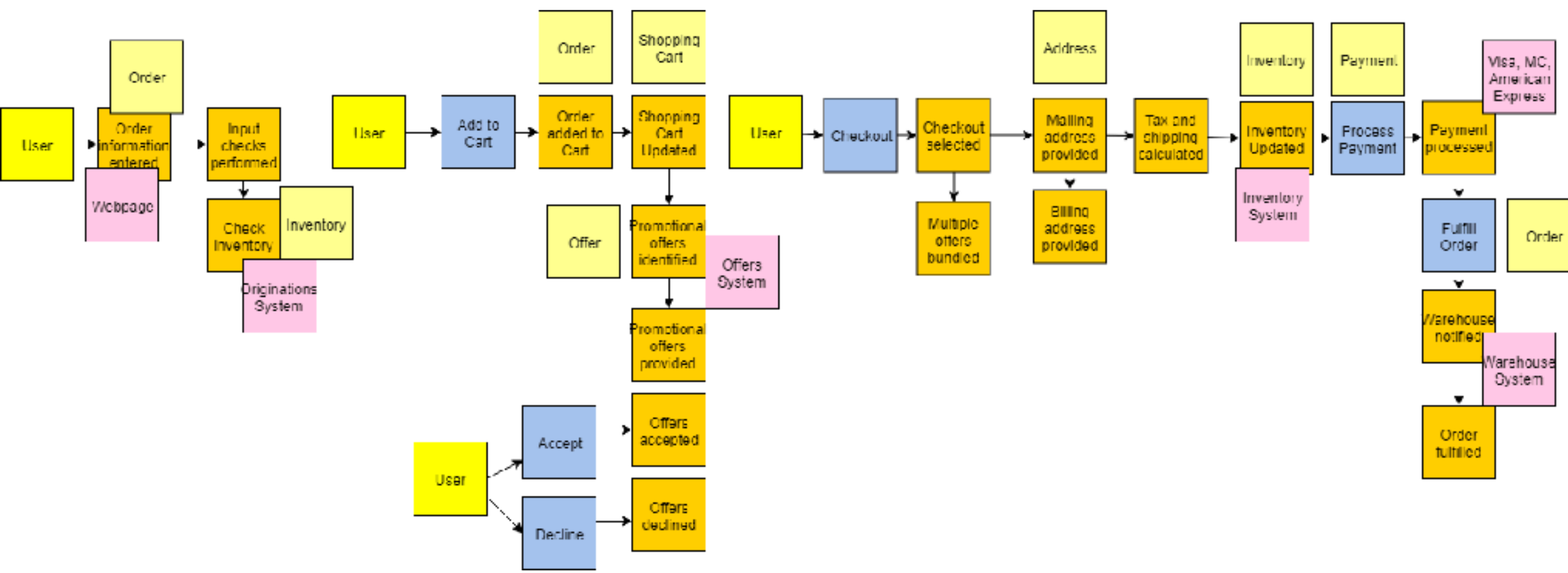


missing events (outlined in red)

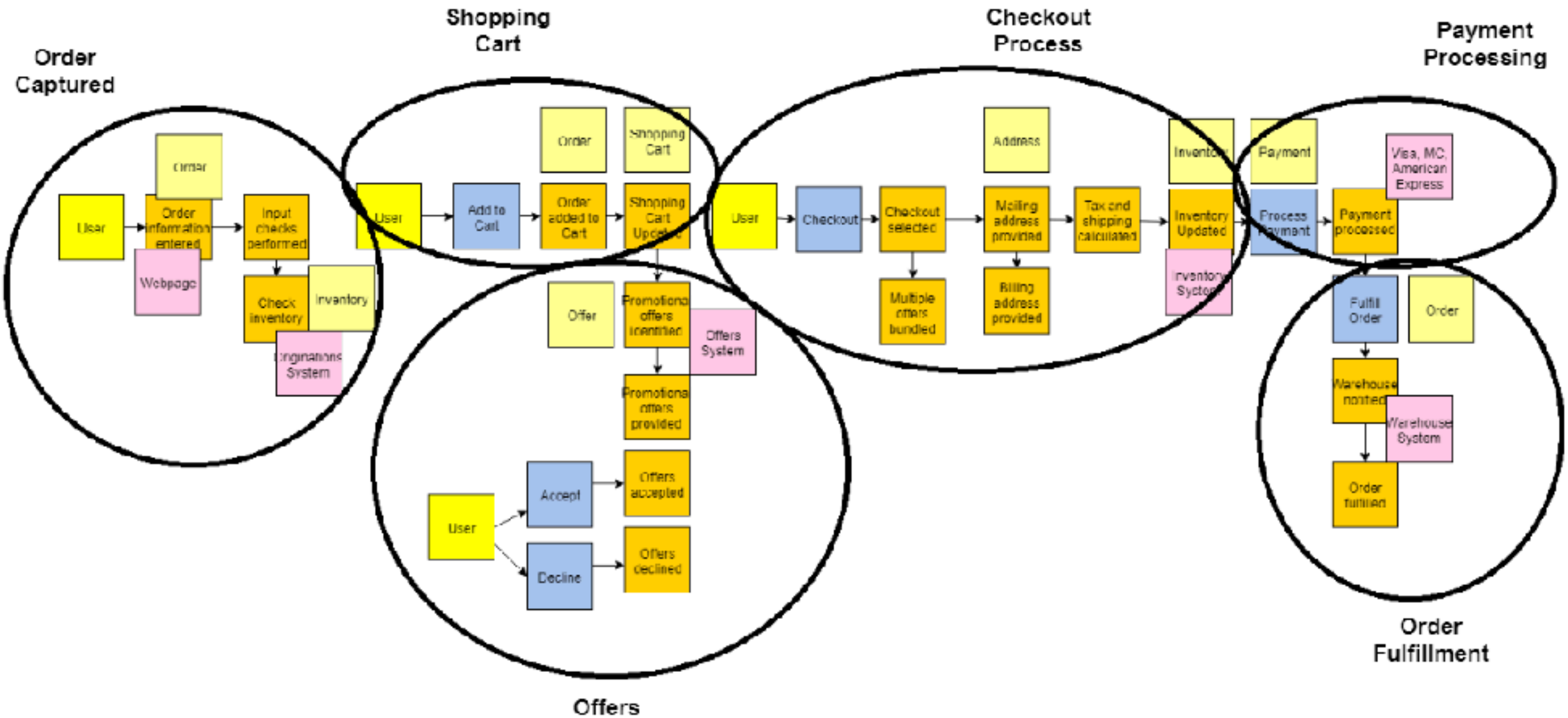
Step #3 — Modeling Out the Broader Ecosystem



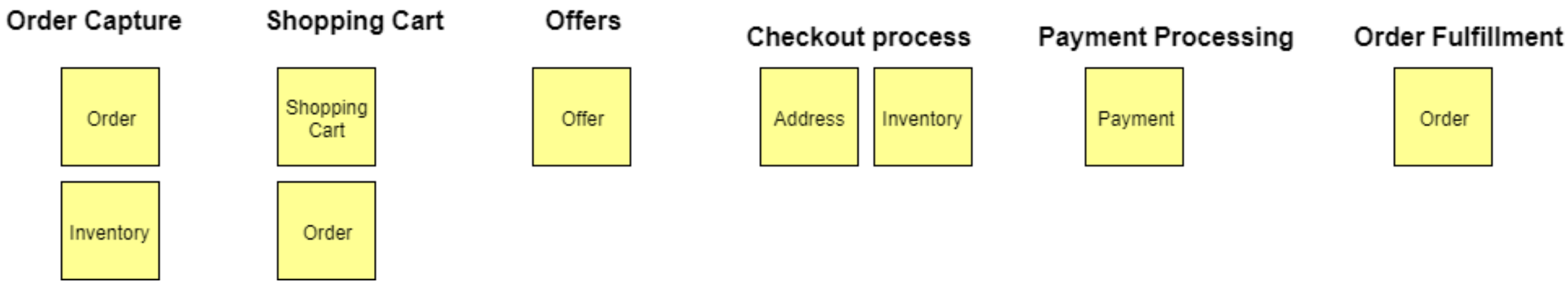
Step #4 — Identify Aggregates



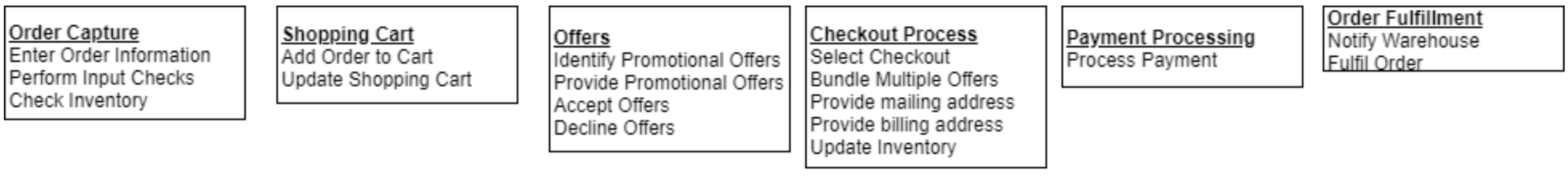
Step #5 — Bounded Context Categorization of Events

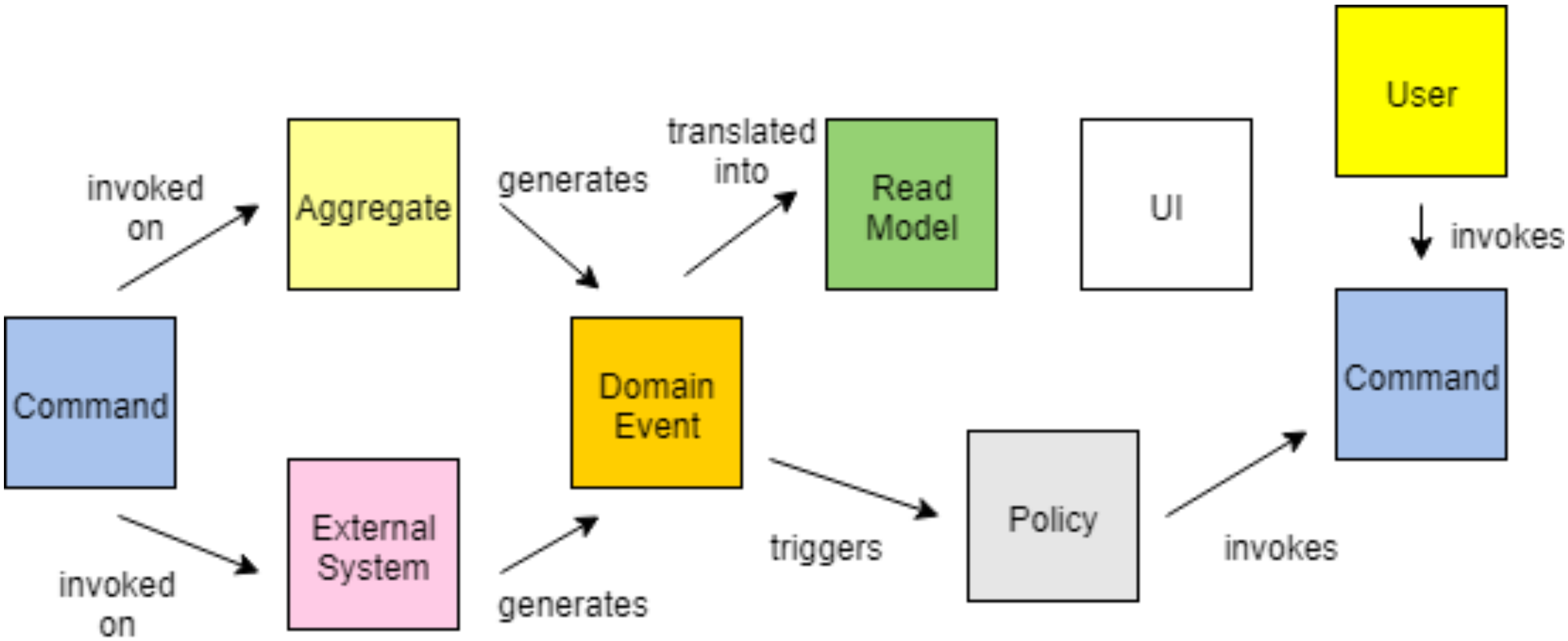


Step #6: Create Services



Step #7: Create Capabilities

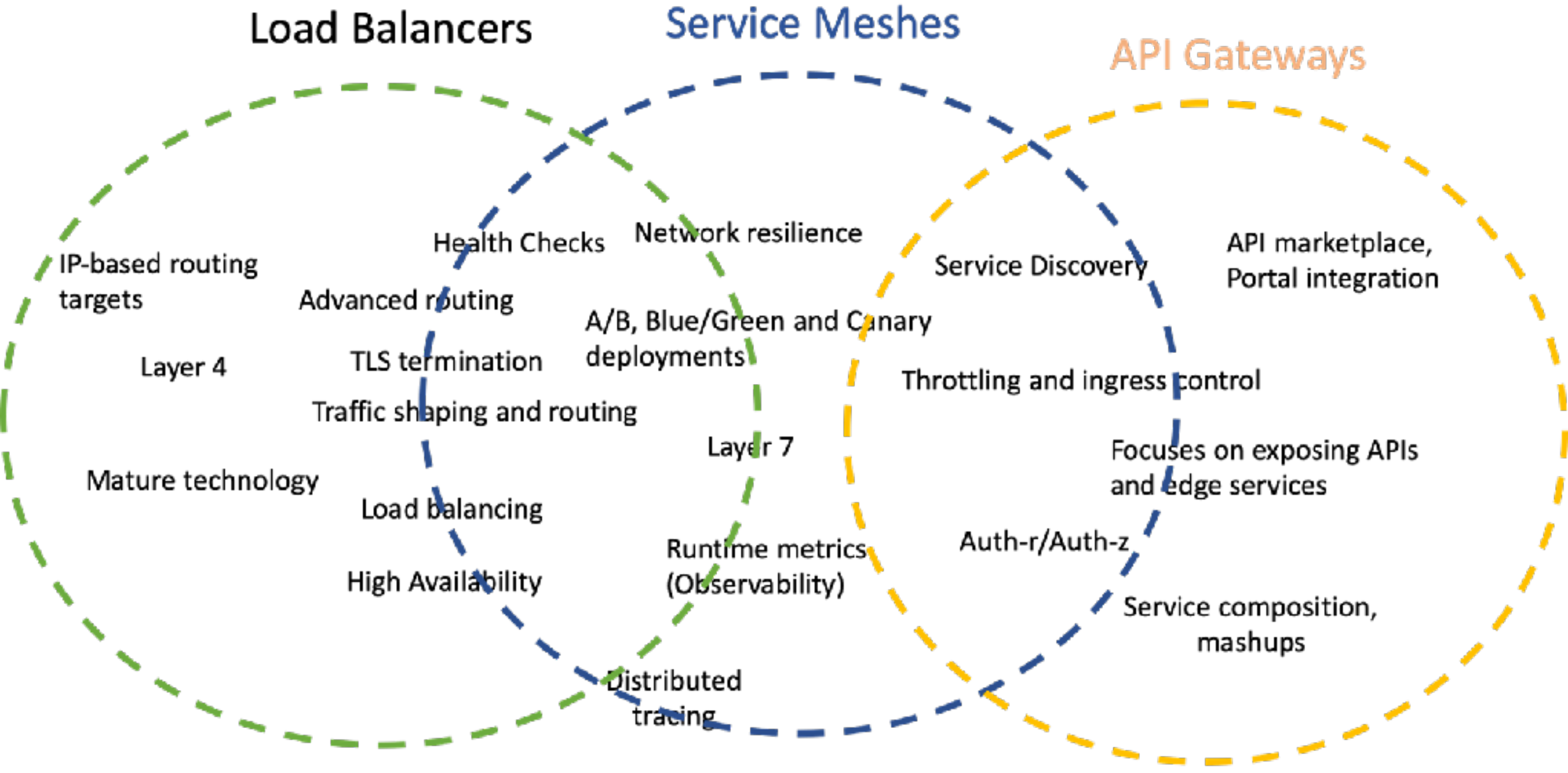



User
invokes
Command
invoked on
Aggregate
generates
translated into
Read Model
UI
Command
Domain Event
triggers
Policy
invokes
invoked on
External System
generates

API Gateway: Abstraction, decoupling, edge routing / security

Service Mesh: endpoints, hosts, ports, metric collection, traffic routing, security

Deployment Platform: Cluster nodes, container schedule, resource management


Load Balancers
Service Meshes
API Gateways
IP-based routing targets
Layer 4
Mature technology
Load balancing
High Availability
Health Checks
Network resilience
Advanced routing
TLS termination
Traffic shaping and routing
A/B, Blue/Green and Canary deployments
Layer 7
Runtime metrics (Observability)
Distributed tracing
Service Discovery
Throttling and ingress control
Auth-r/Auth-z
API marketplace, Portal integration
Focuses on exposing APIs and edge services
Service composition, mashups

- Test , stage, prod
- Api gateway
- Database