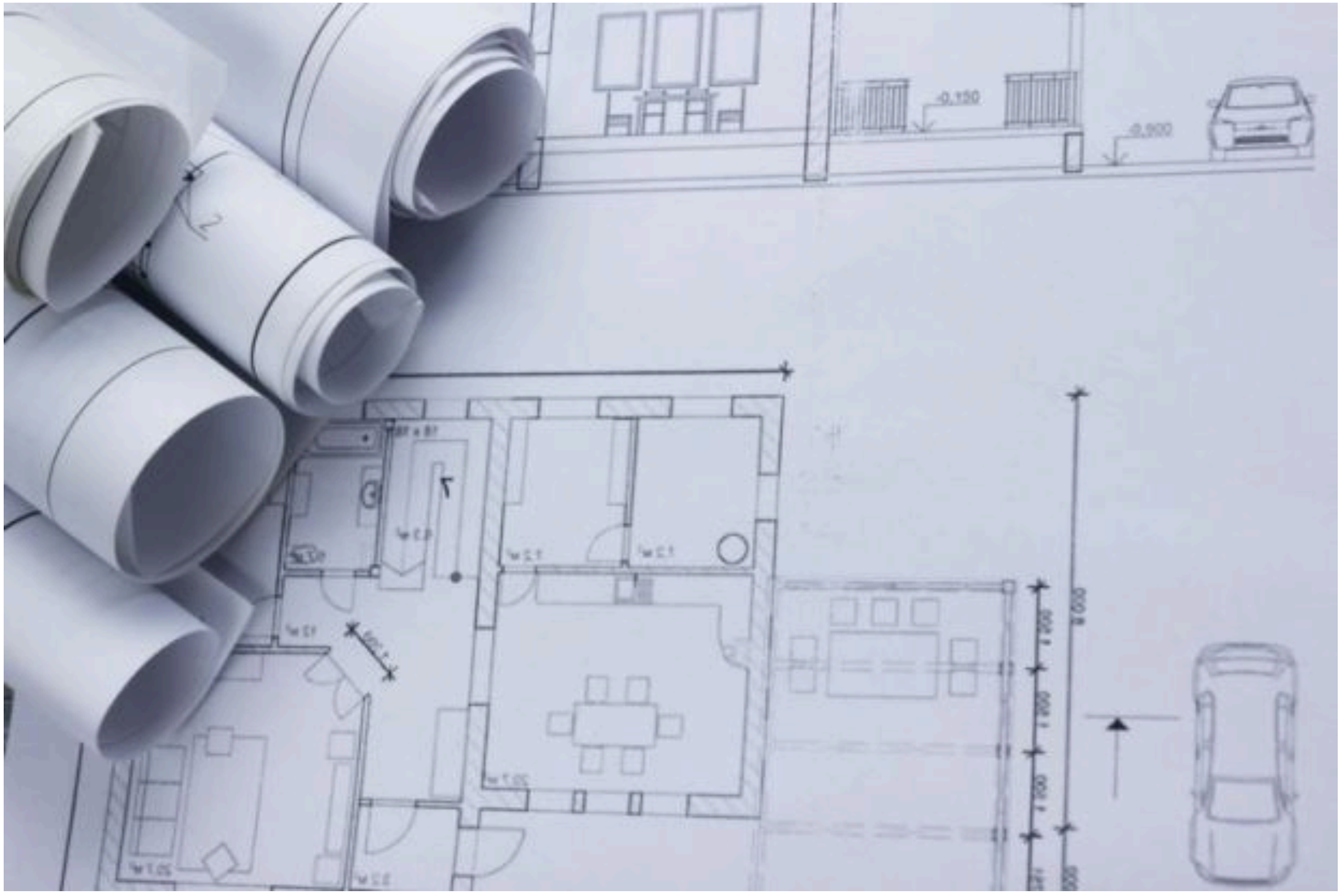


Software Architecture

» 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

- Years of experience
- Technology stack
- Expectations

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

Architecture vs Design

System quality

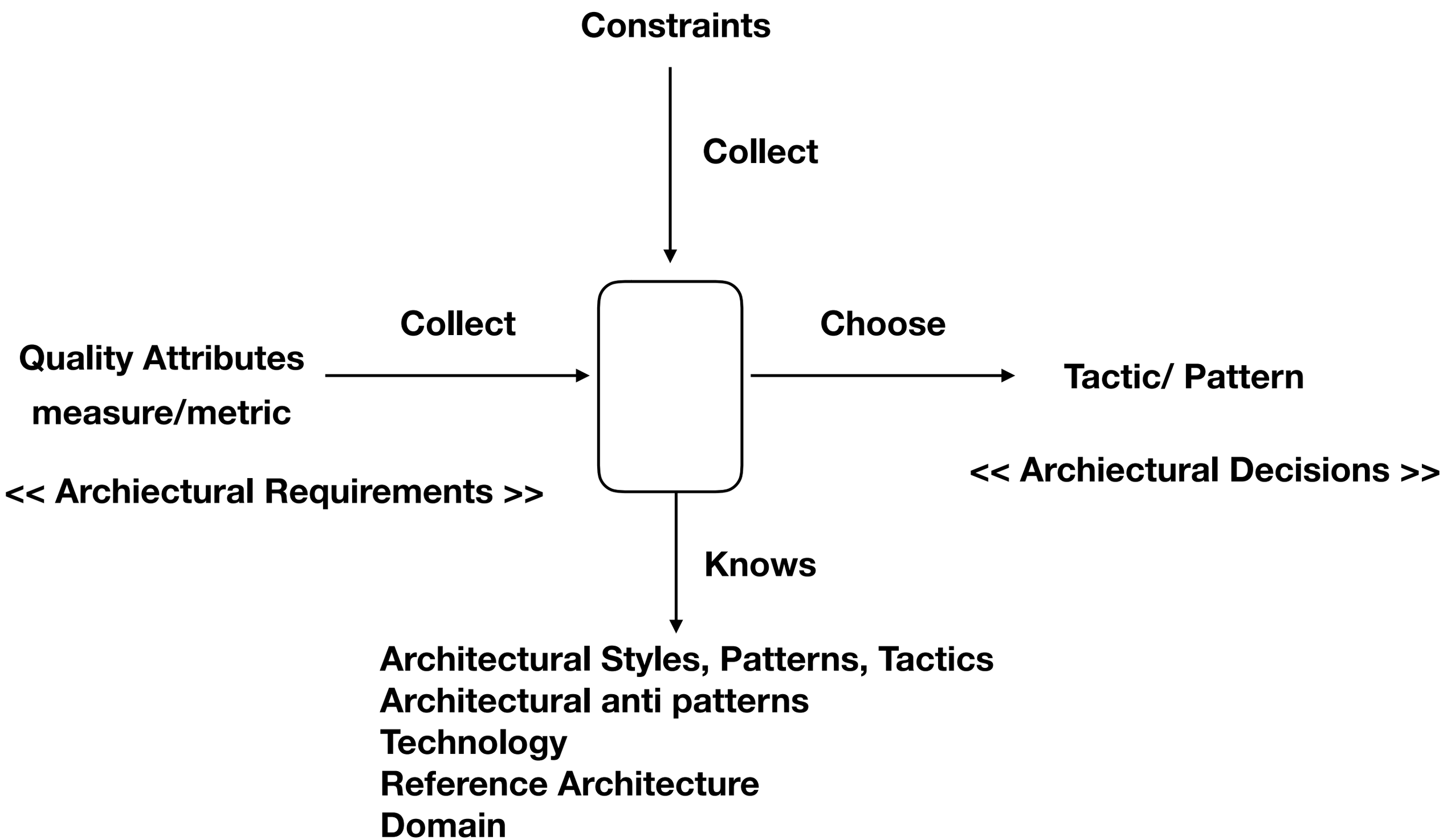
Code Maintainability

Architecture Design
UI Design
Test Design

Quality Attributes	measure/metric	Tactic/ Pattern

• Scalability (volume)
• Performance
  • Cpu
  • I/O
  • Memory
• Security - Trust
• Maintainability
	• Response time
	• Latency
	• DREAD
	• probability
		• Caching
		• Concurrency
		• Master and Slave
		• Stateless
		• Object Pooling
		• Lazy loading
		• Authorization
		• POLP
		• Authentication
		• Testable
		• Monitoring
		• Doc
		• Transaction - ACID

• Robustness - Rugud • Reliability - Trust • Availability • Usability • …		

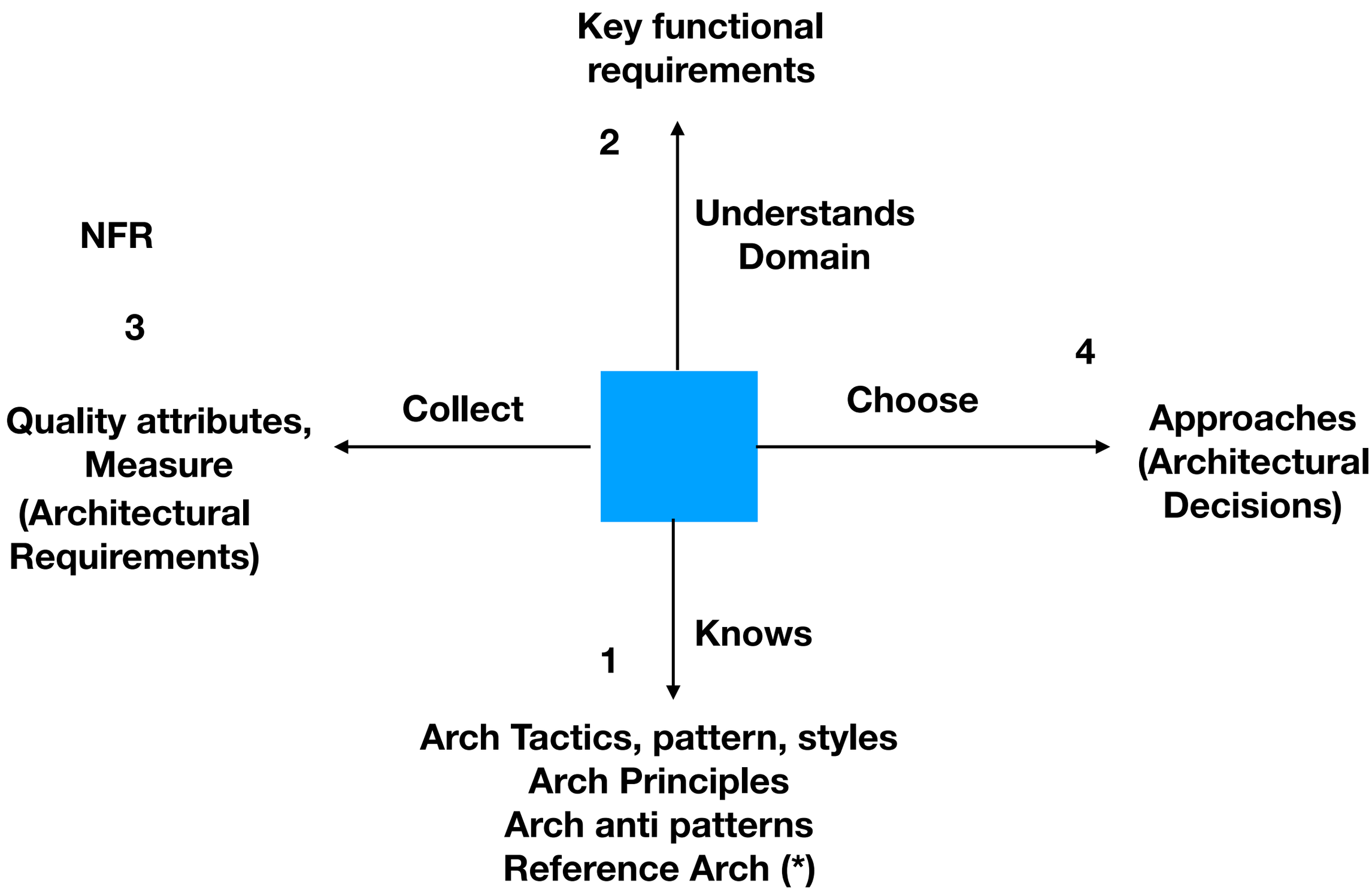

Constraints
Collect
Collect
Quality Attributes
measure/metric
<< Archiectural Requirements >>
Choose
Tactic/ Pattern
<< Archiectural Decisions >>
Knows
Architectural Styles, Patterns, Tactics
Architectural anti patterns
Technology
Reference Architecture
Domain

Pre

Engineering vs Tuning

Post

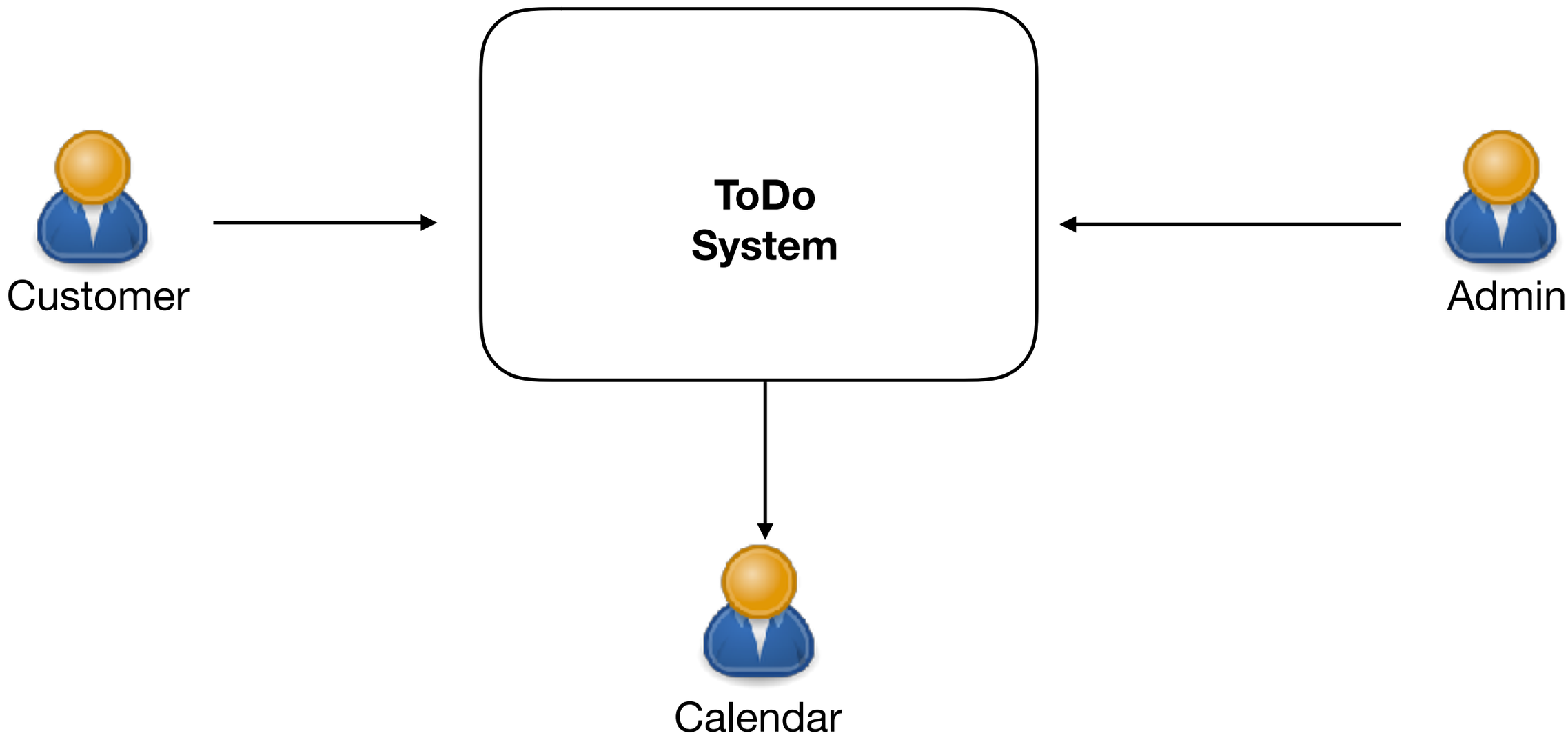
5 Communicate

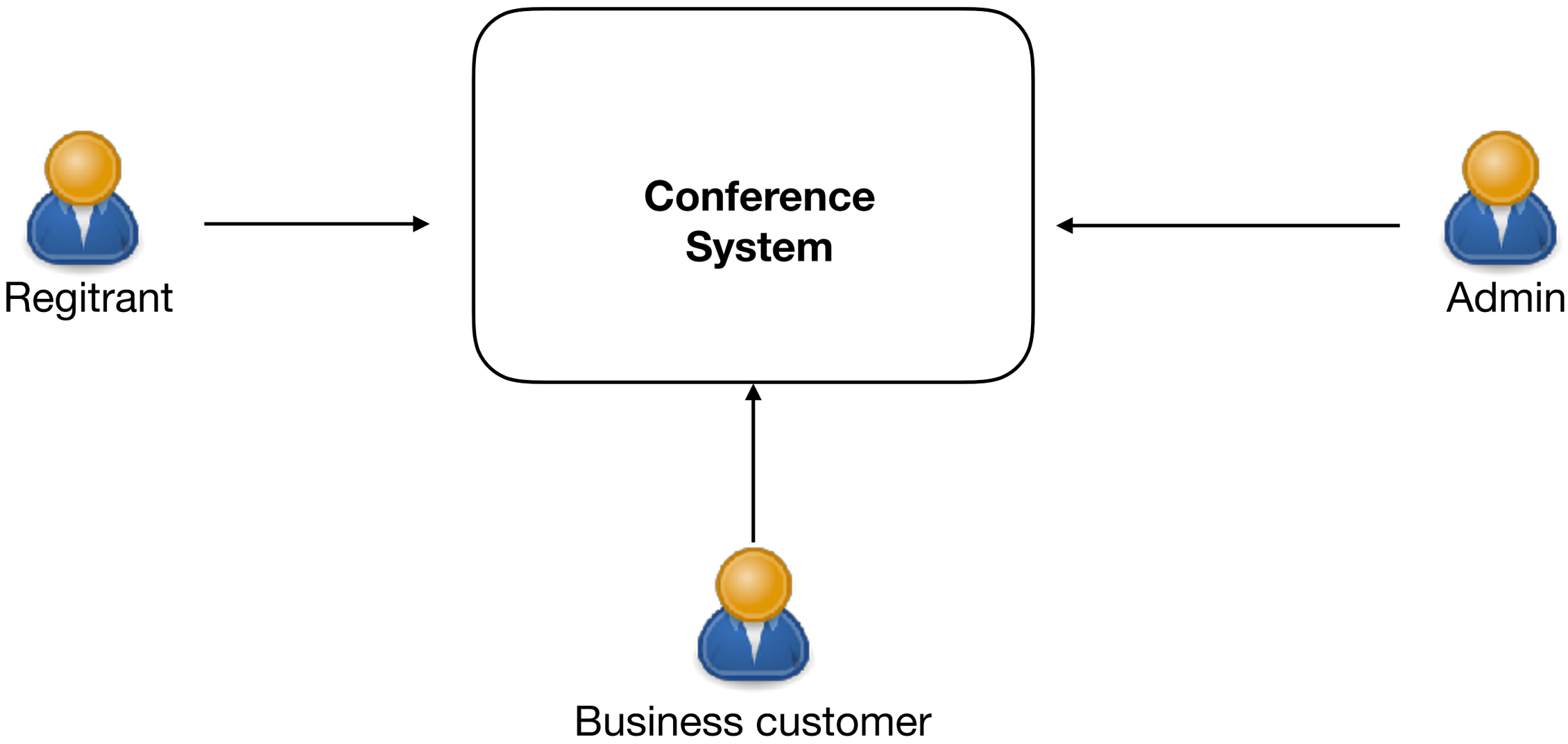


Part 1

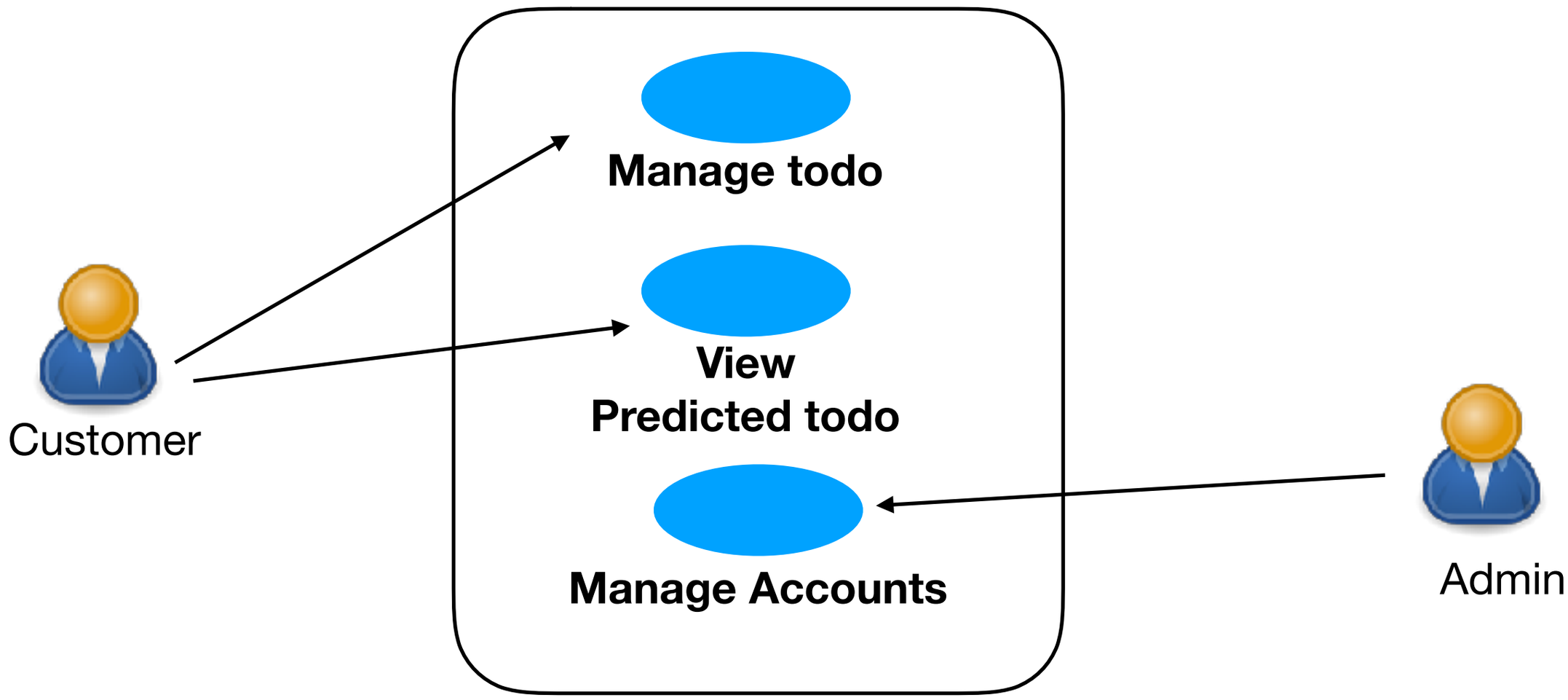
Arch Req

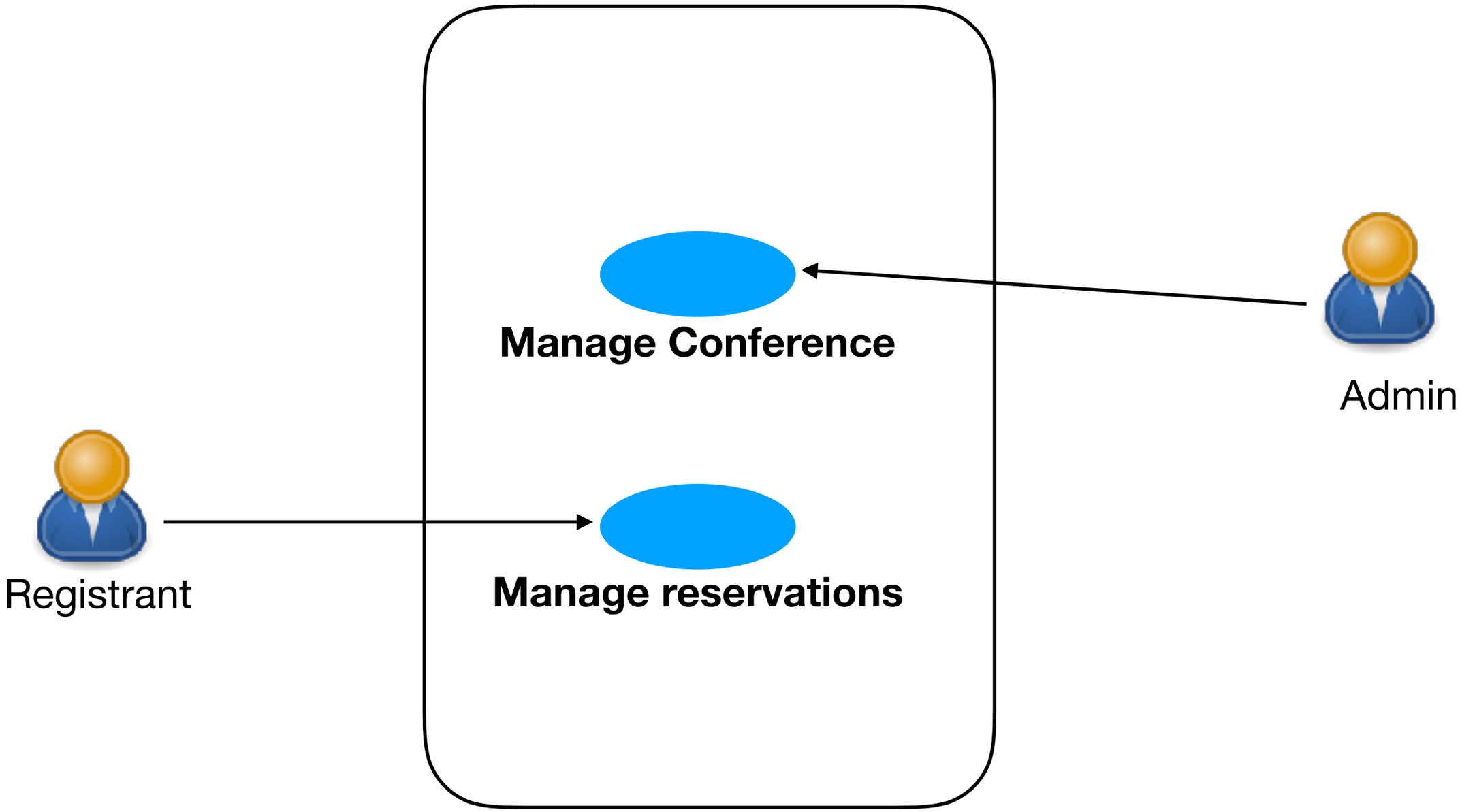
1. Context view




Conference System
Regitrant
Admin
Business customer

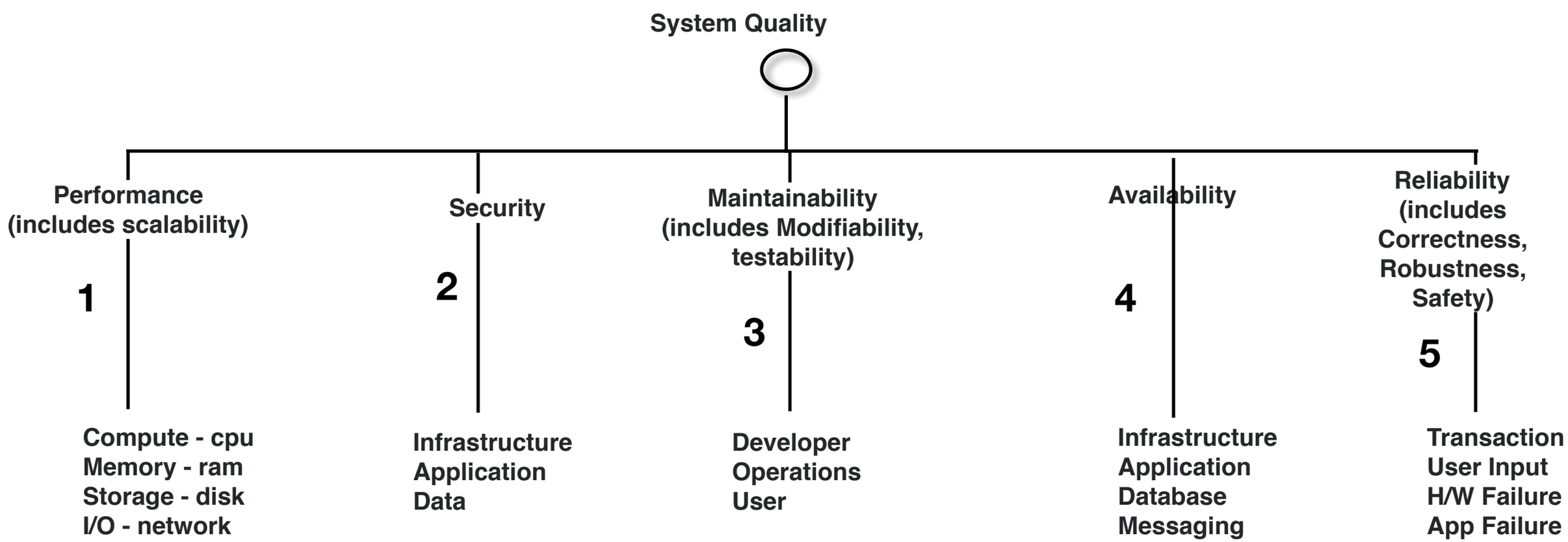
2. Functional view




Manage Conference
Admin
Registrant
Manage reservations

3. Quality View

Usability
Portability
Interoperability



Quality Model

- IEEE
- ISO
- MacCal
- Bohem
- ..

Quality Attribute Scenarios

Quality	Source (who)	Stimulus (action)	Artifact (which)	Environment (context)	Response (output)	Measure (scale)
Performance	As a user	I want to add a todo	In the portal	During peak load.	The todo is added	In < 3 secs
Reliability	As a user	I want to add a todo	From the portal	Duplicate request	Duplicate entry is detected	In 100% of cases
Maintainability	Developer	Change the Configuration engine	In the Frame work	During maintenance	The configuration mechanism is replaced	In < 1 week
Security	unknown identity	requests to add a new todo	In the portal	During Normal load.	block access to the data and record the access attempts	100% probability of detecting the attack, 100% probability of denying access
Availability	The processor	Failed	In the db server	During Operational Hours	Secondary is made Primary	In < 5 minutes

As a user I want to **add a todo** In the portal

As a user I want to **add a todo** In the portal During **peak load.** The todo is added In **< 3 secs**

4. Constraints view

It should work on IE 11, Chrome and Firefox

Should work on windows, unix, Mac platforms

Should deploy on Azure Cloud

---

5. Assumptions View

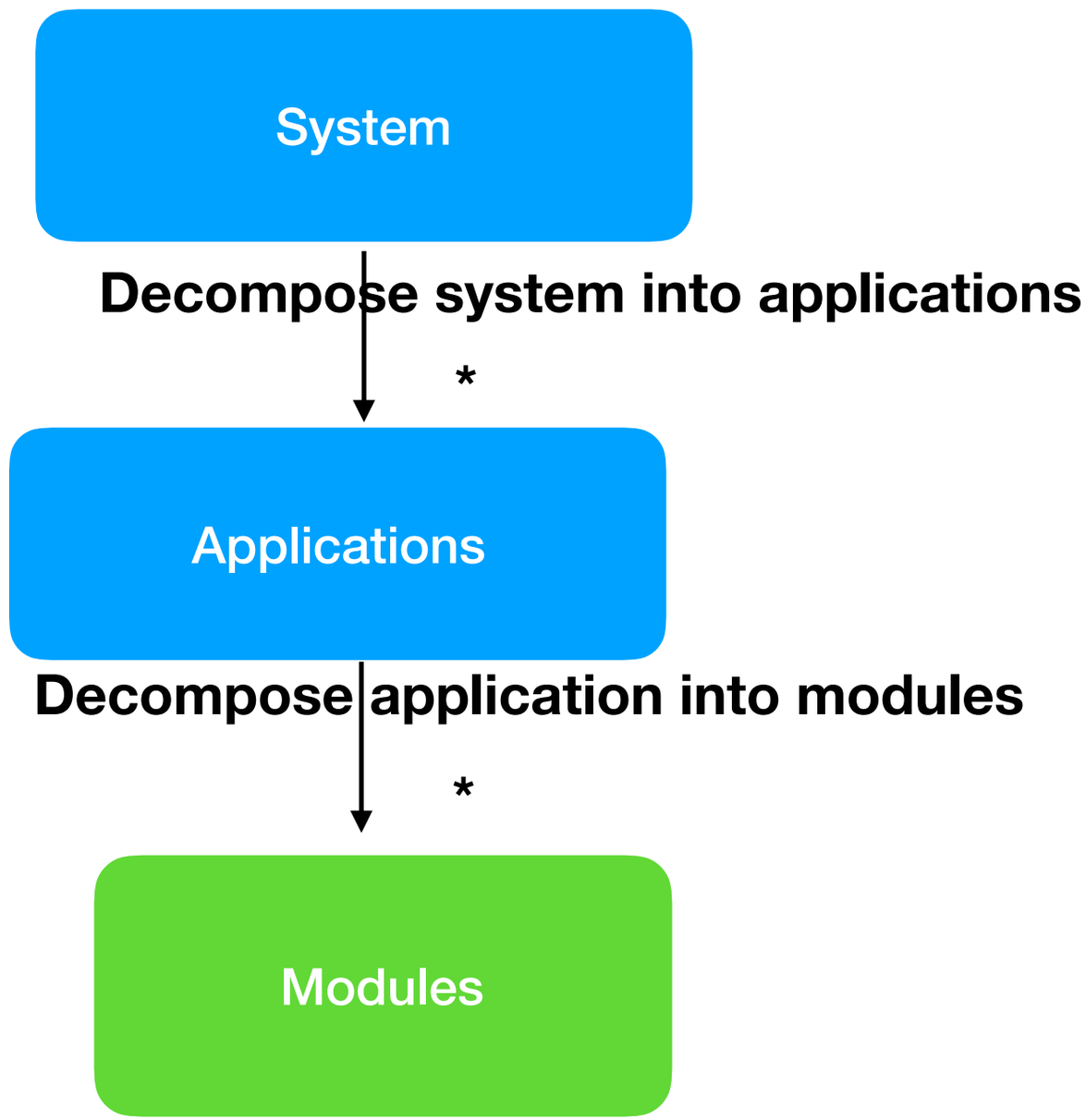
1	Will use open source Technologies only
2	
3	

Part 2

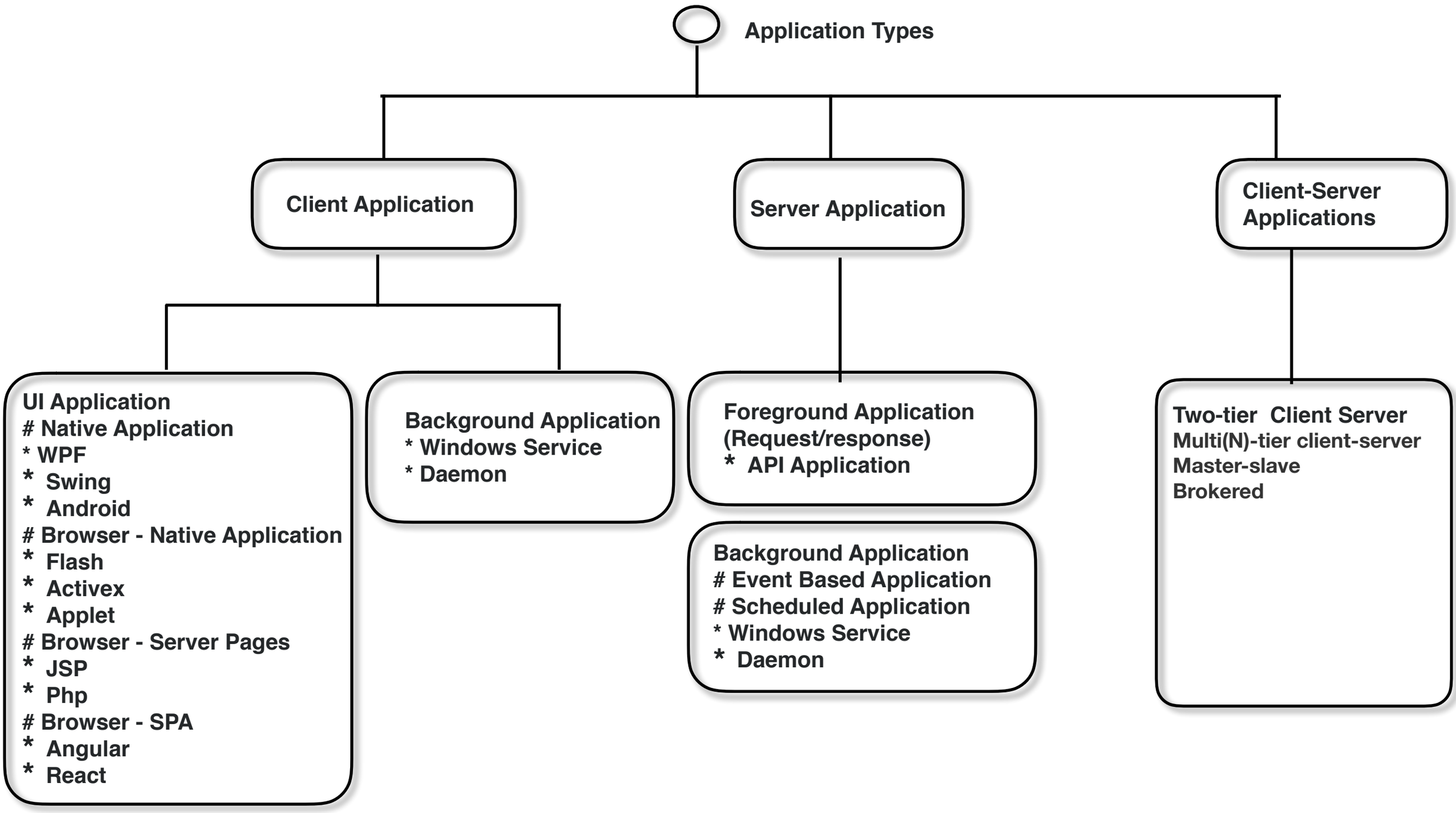
Arch Design

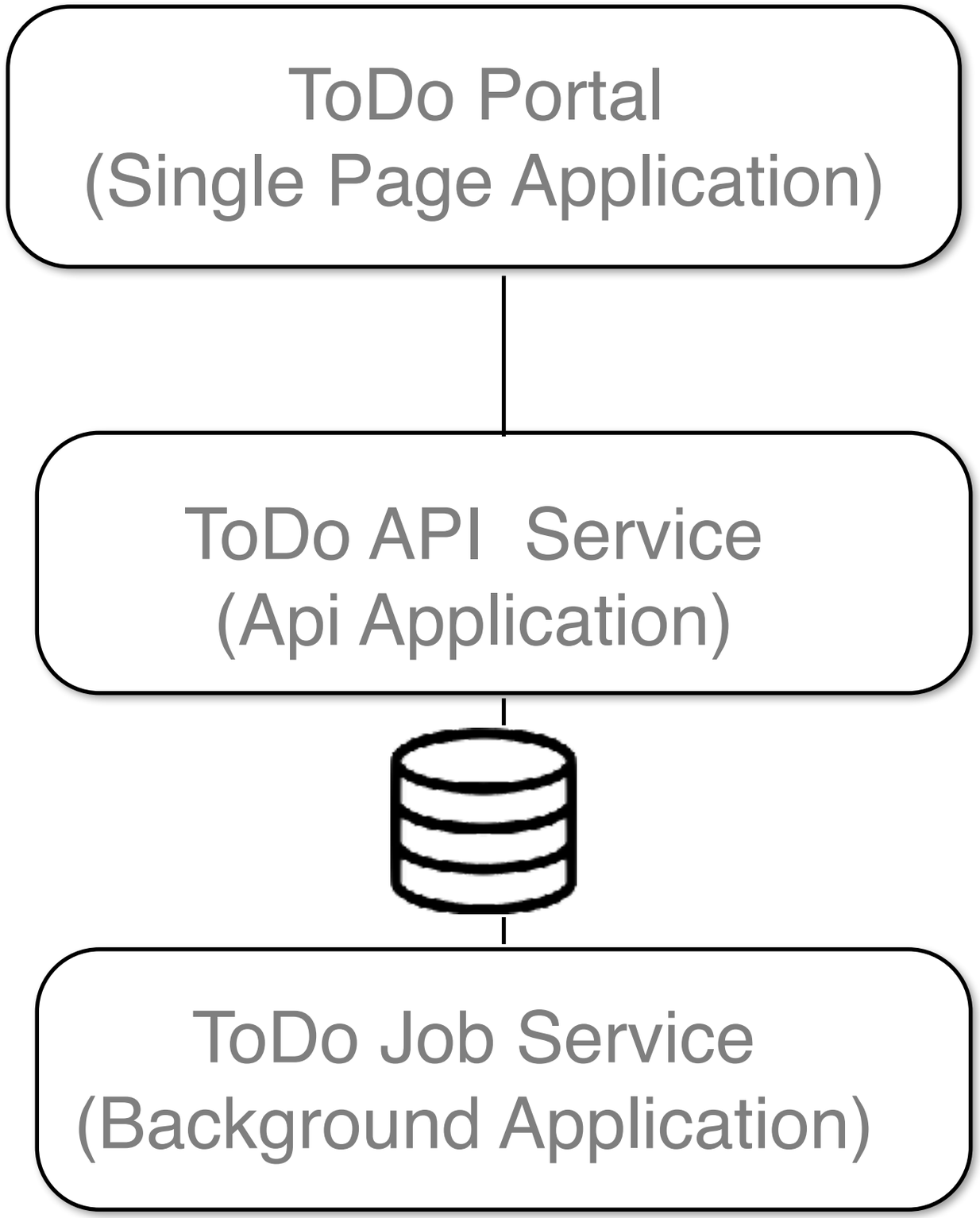
Decisions

1. Logical View

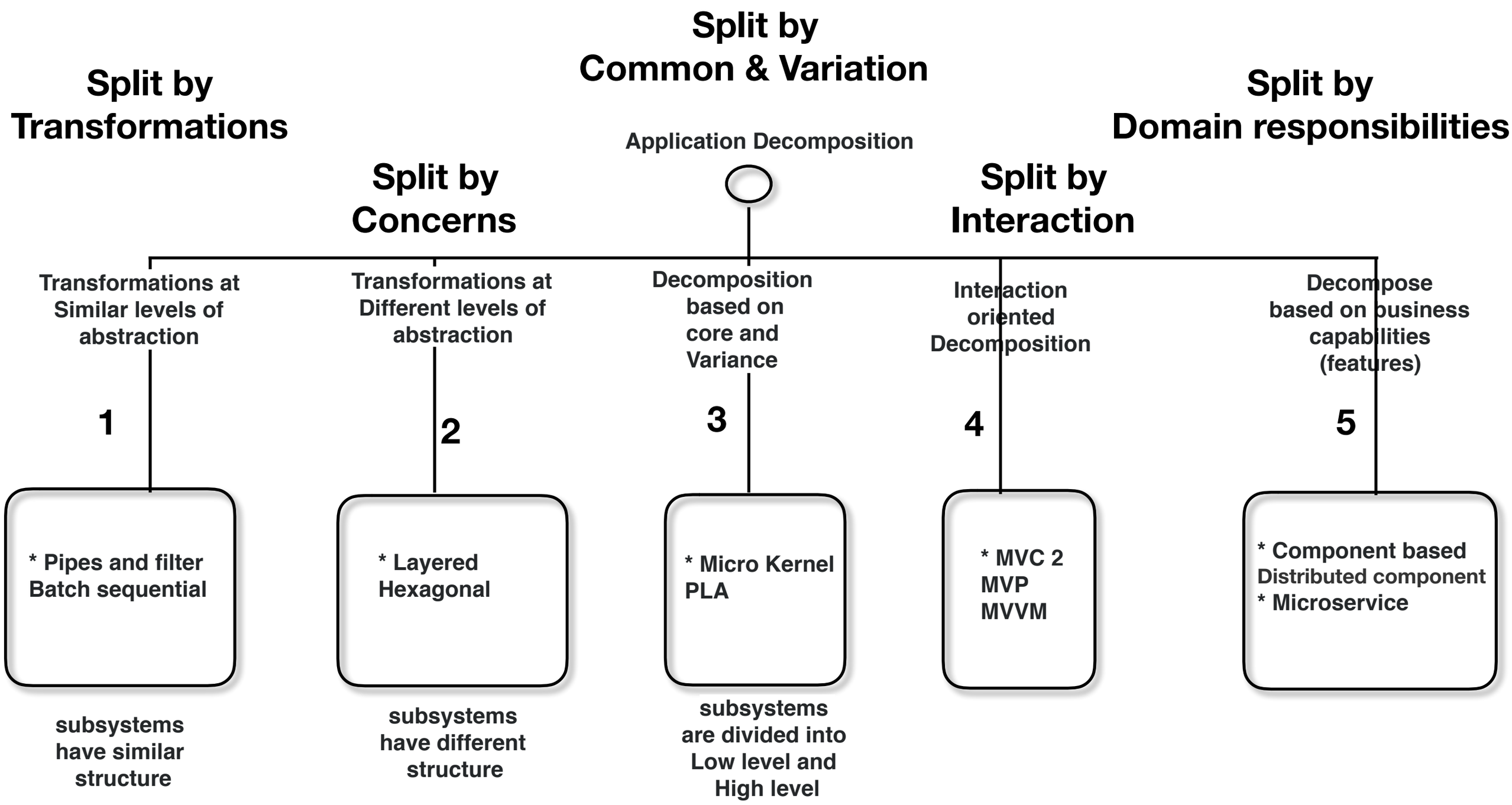


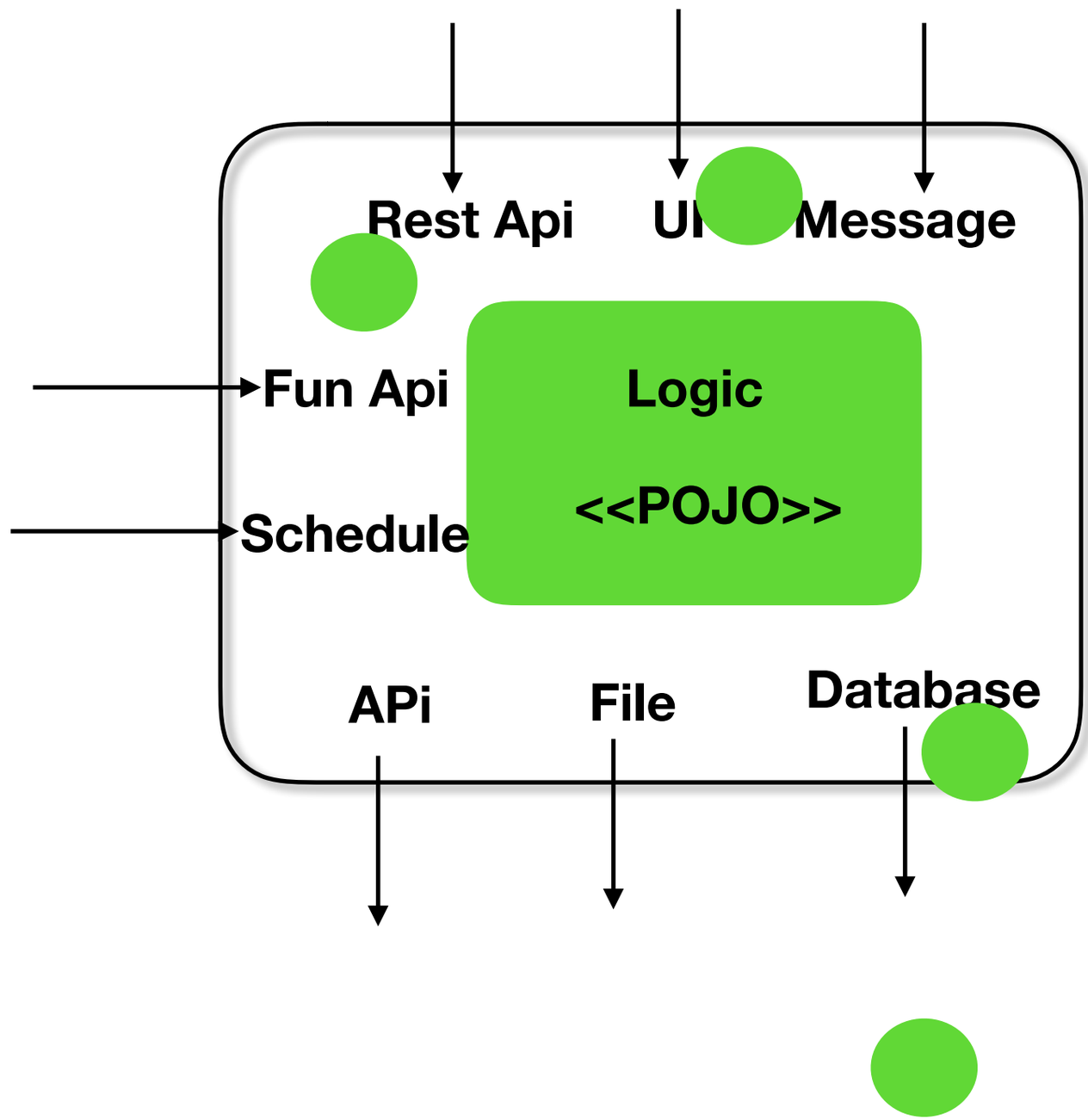
Step 1. Decompose the system into Applications/Tiers

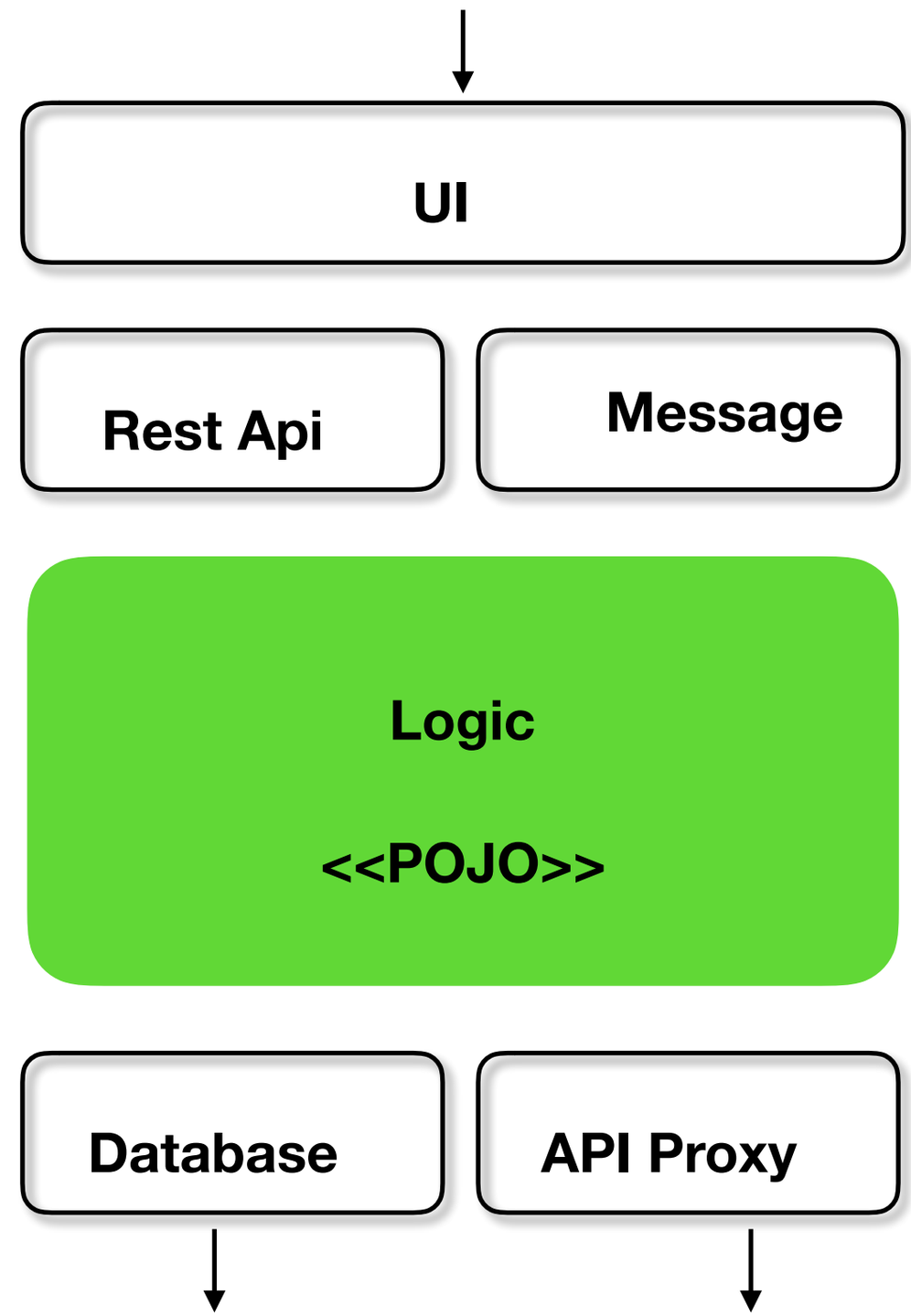


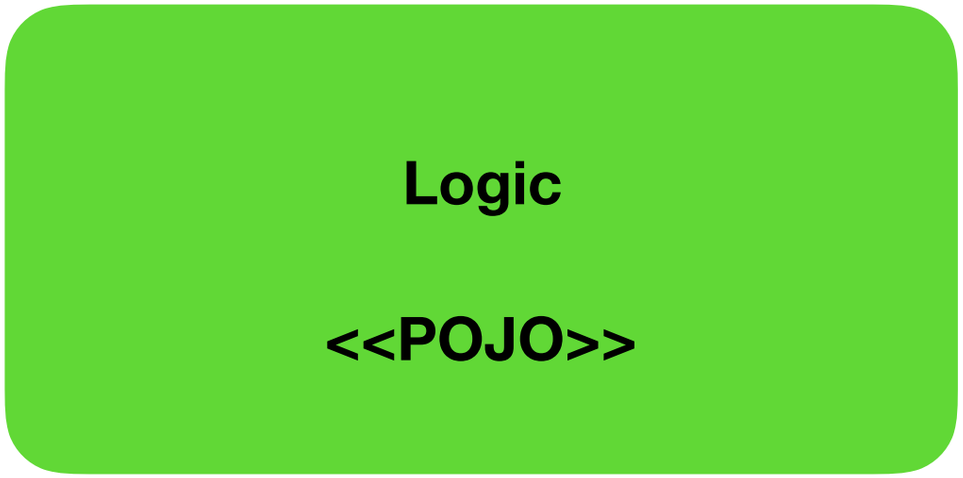

ToDo Portal
(Single Page Application)
ToDo API Service
(Api Application)
ToDo Job Service
(Background Application)

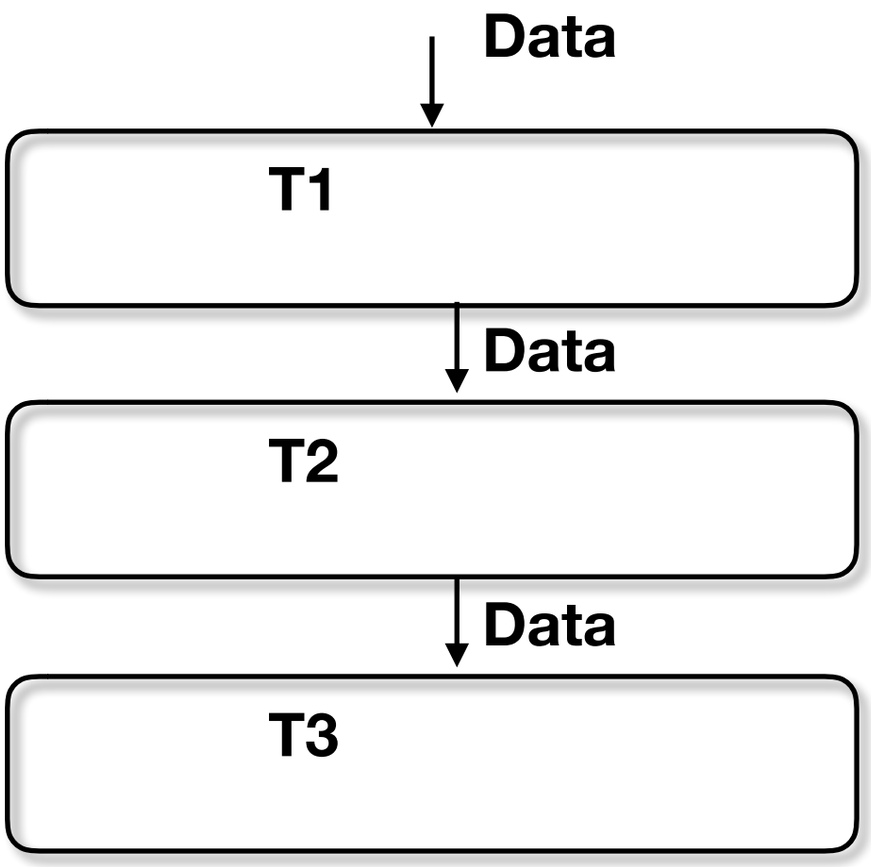
Step 1. Decompose application/ Tiers

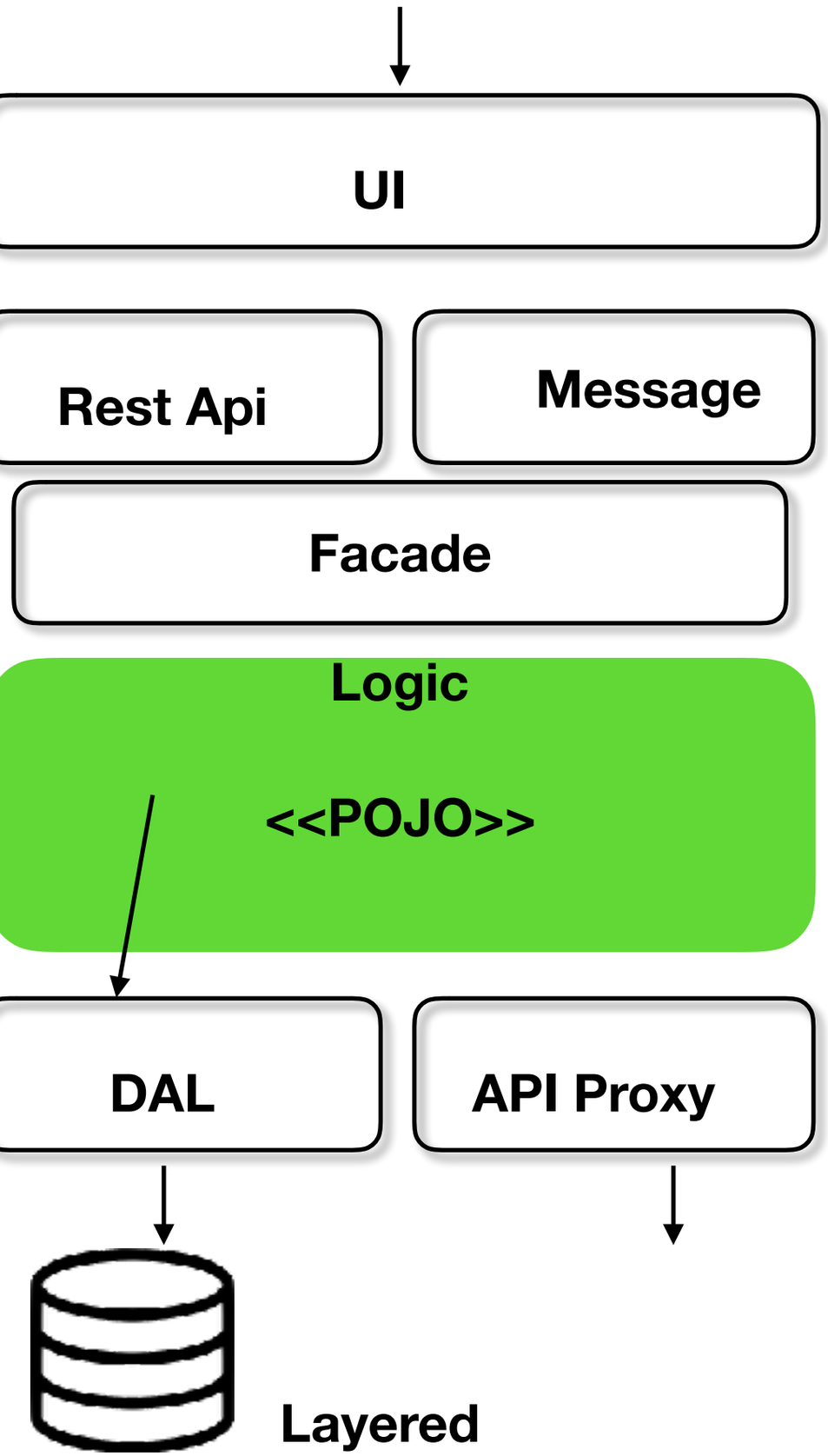


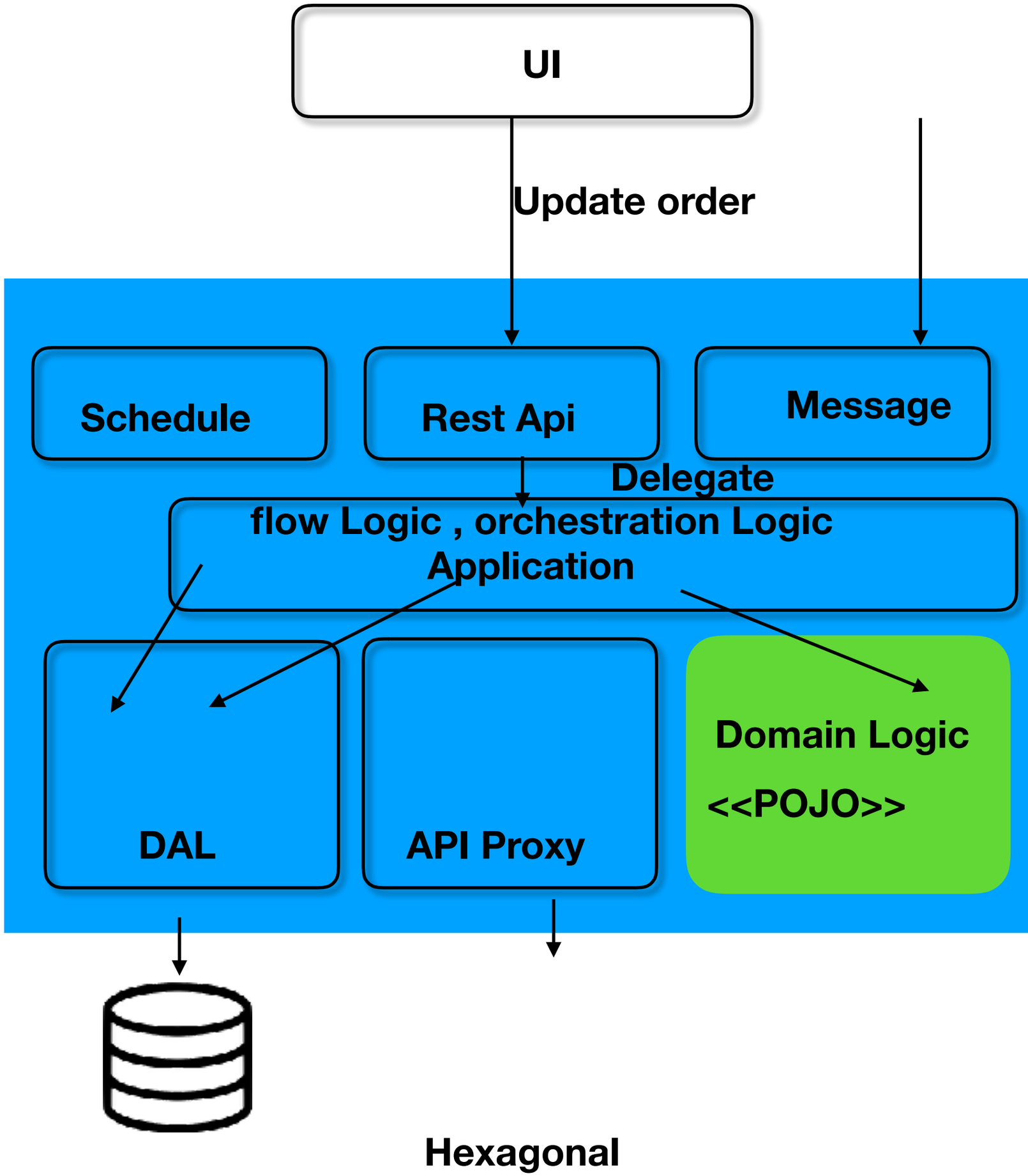

Rest Api
UI
Message
Fun Api
Logic
<<POJO>>
Schedule
APi
File
Database

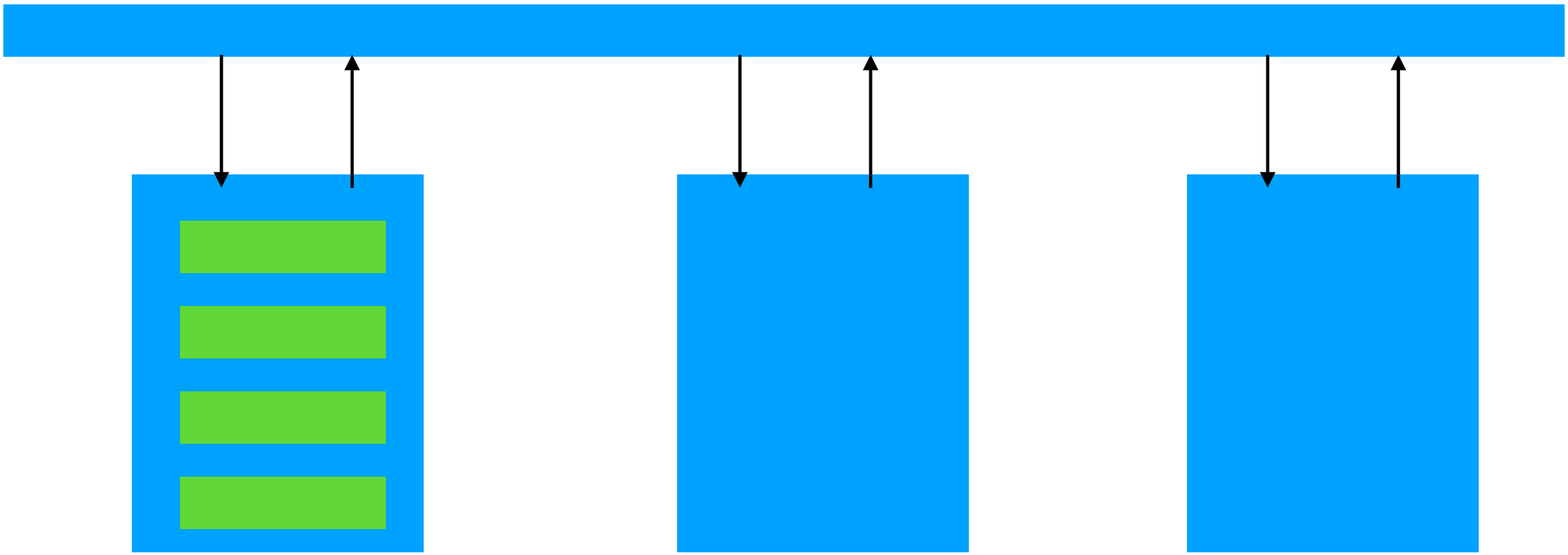

UI
Rest Api
Message
Logic
<<POJO>>
Database
API Proxy

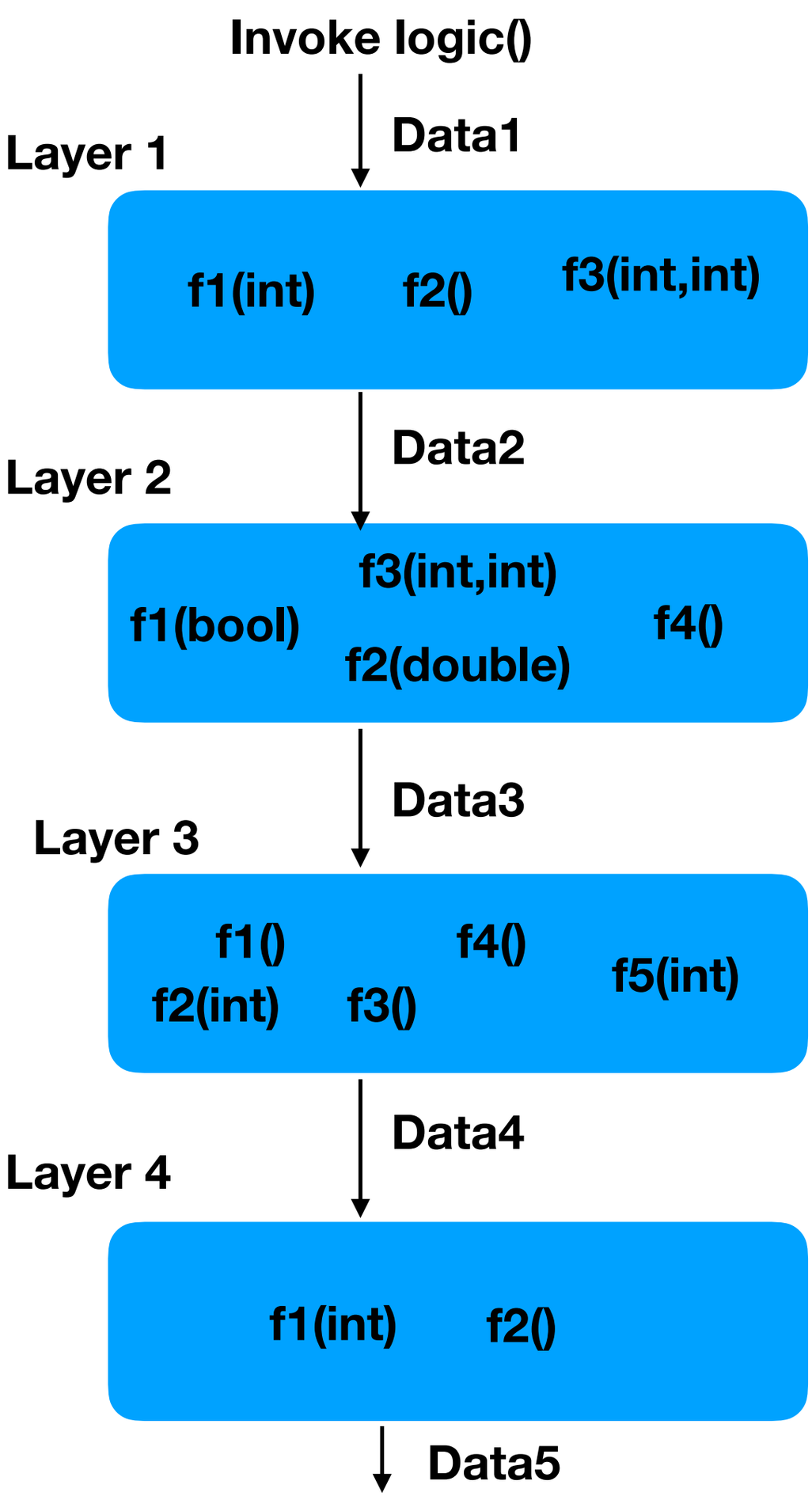

Logic
<<POJO>>

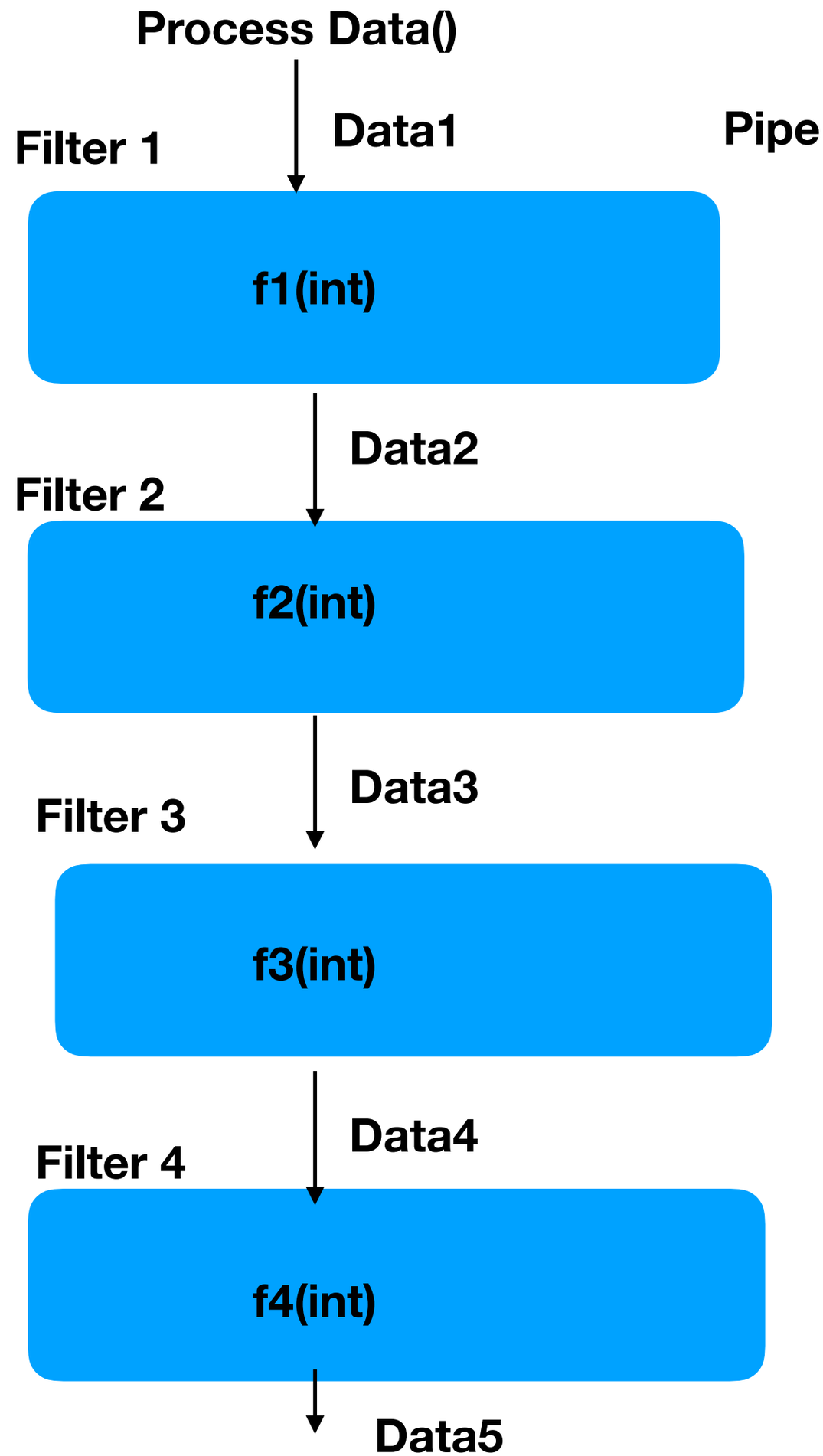

Data
T1
Data
T2
Data
T3

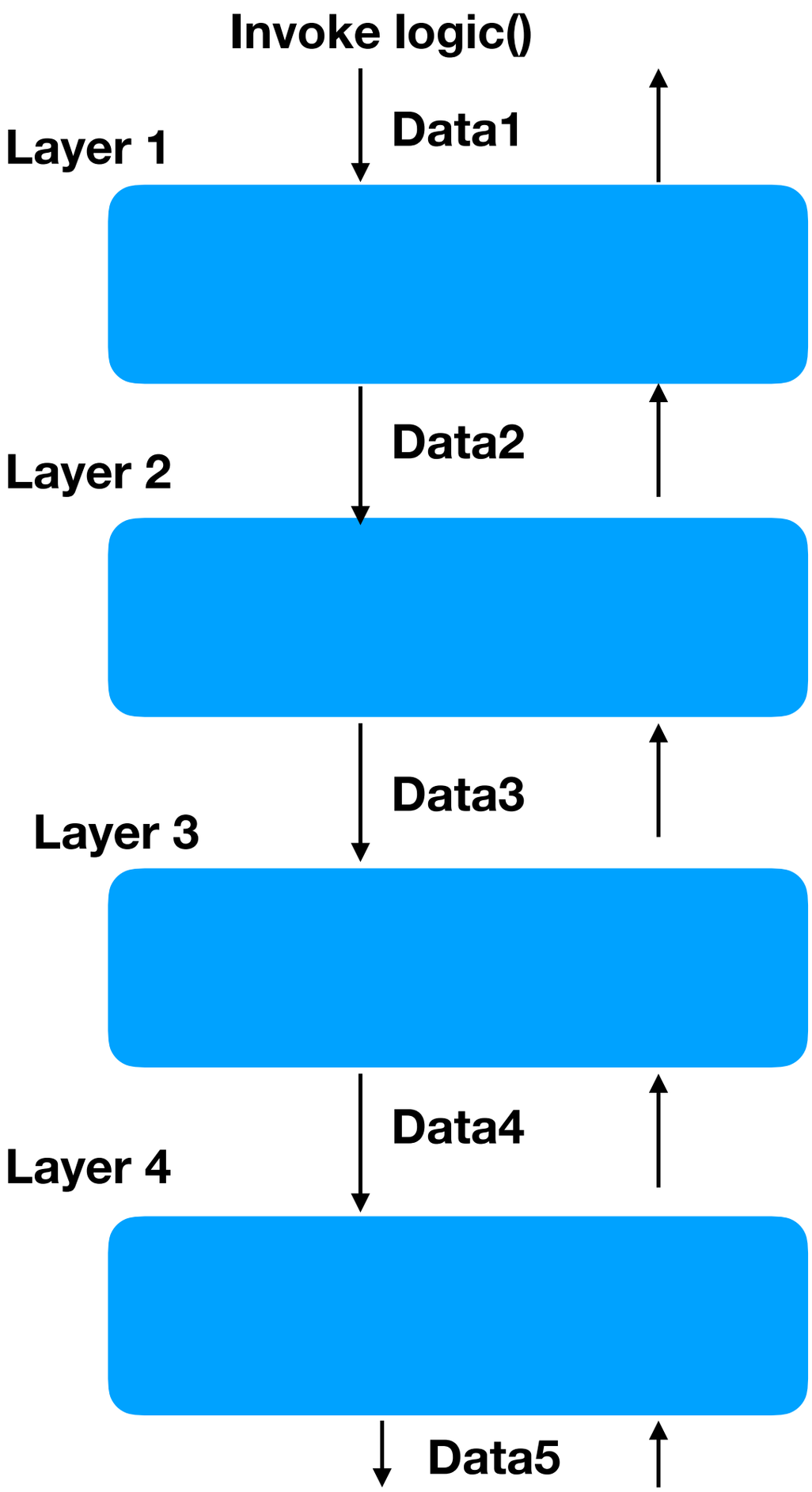

UI
Rest Api
Message
Facade
Logic
<<POJO>>
DAL
API Proxy
Layered

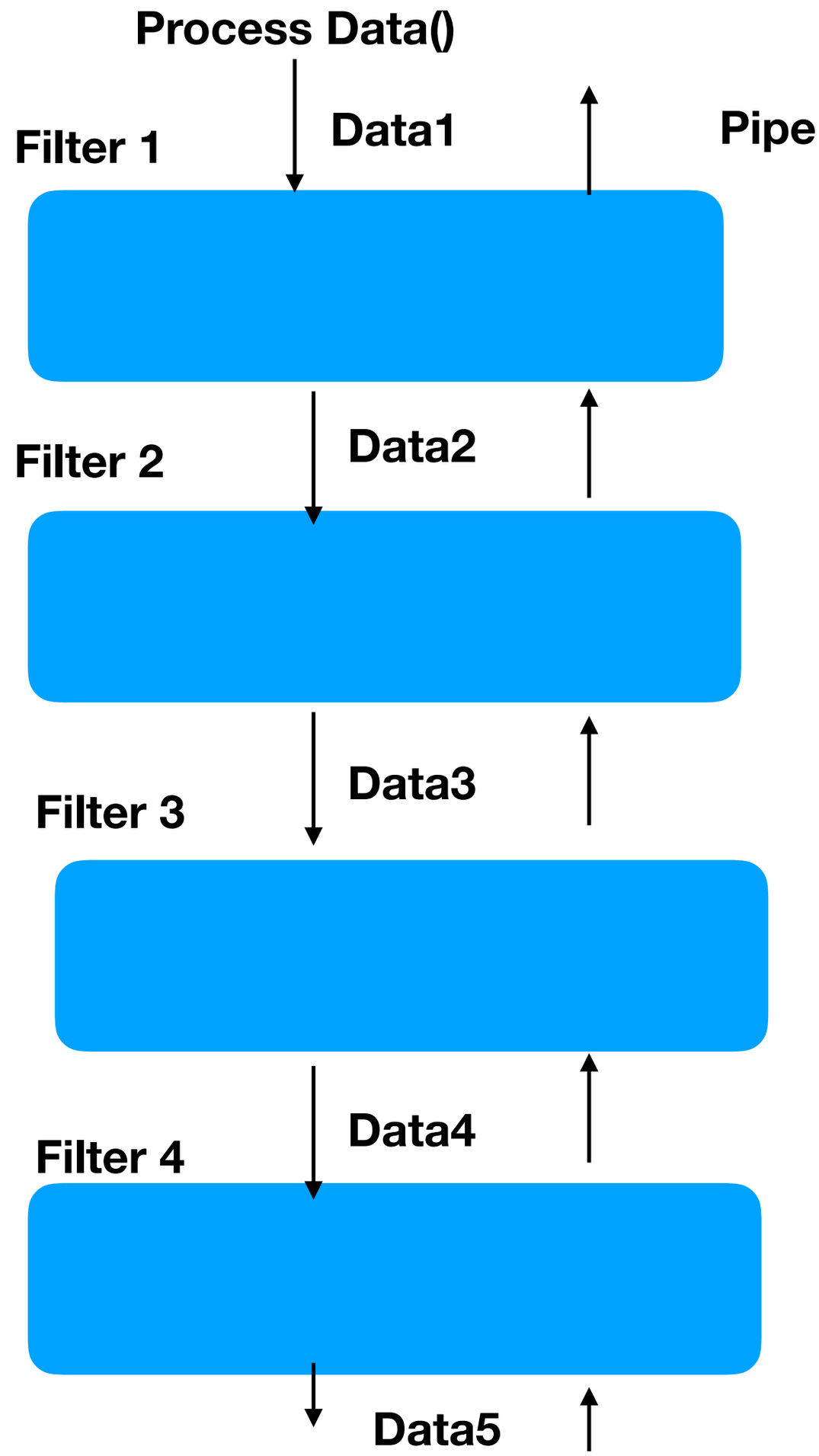

UI
Update order
Schedule
Rest Api
Message
Delegate
flow Logic , orchestration Logic
Application
DAL
API Proxy
Domain Logic
<<POJO>>
Hexagonal



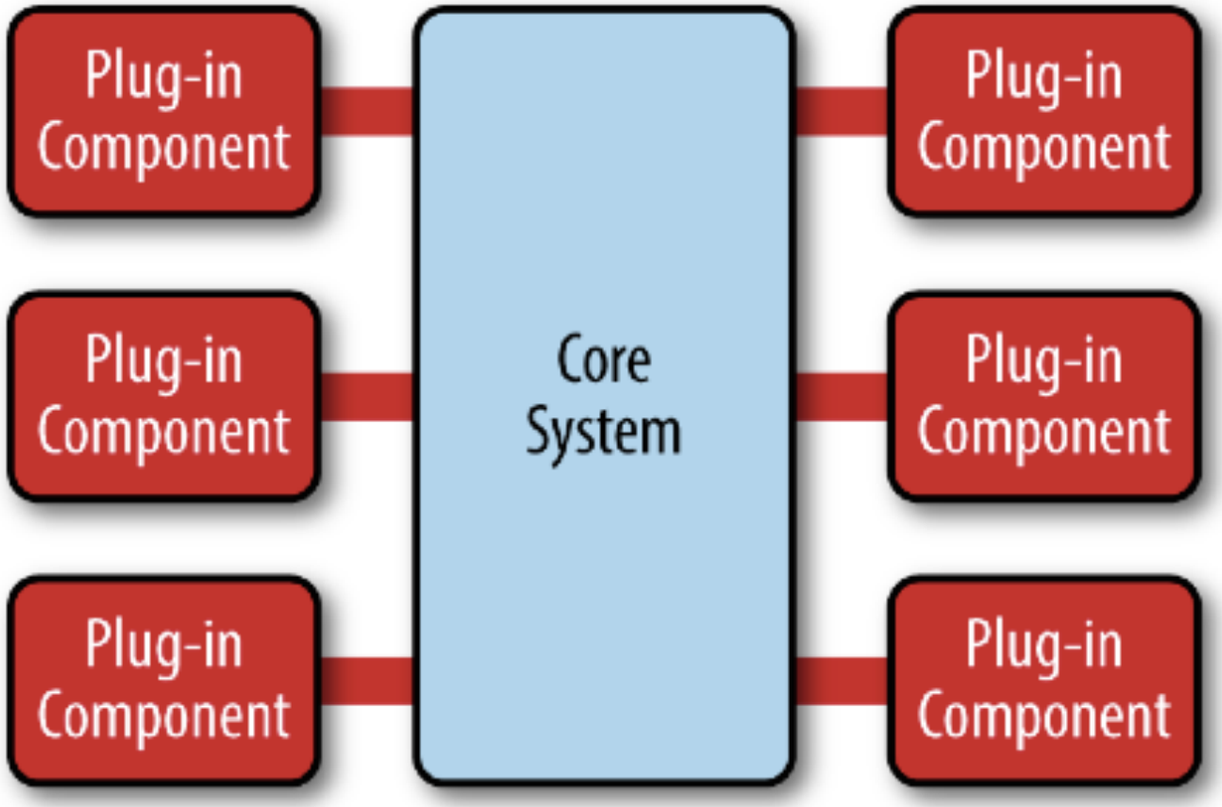

Invoke logic()
Layer 1
Data1
f1(int)
f2()
f3(int,int)
Layer 2
Data2
f3(int,int)
f1(bool)
f4()
f2(double)
Layer 3
Data3
f1()
f4()
f5(int)
f2(int)
f3()
Layer 4
Data4
f1(int)
f2()
Data5

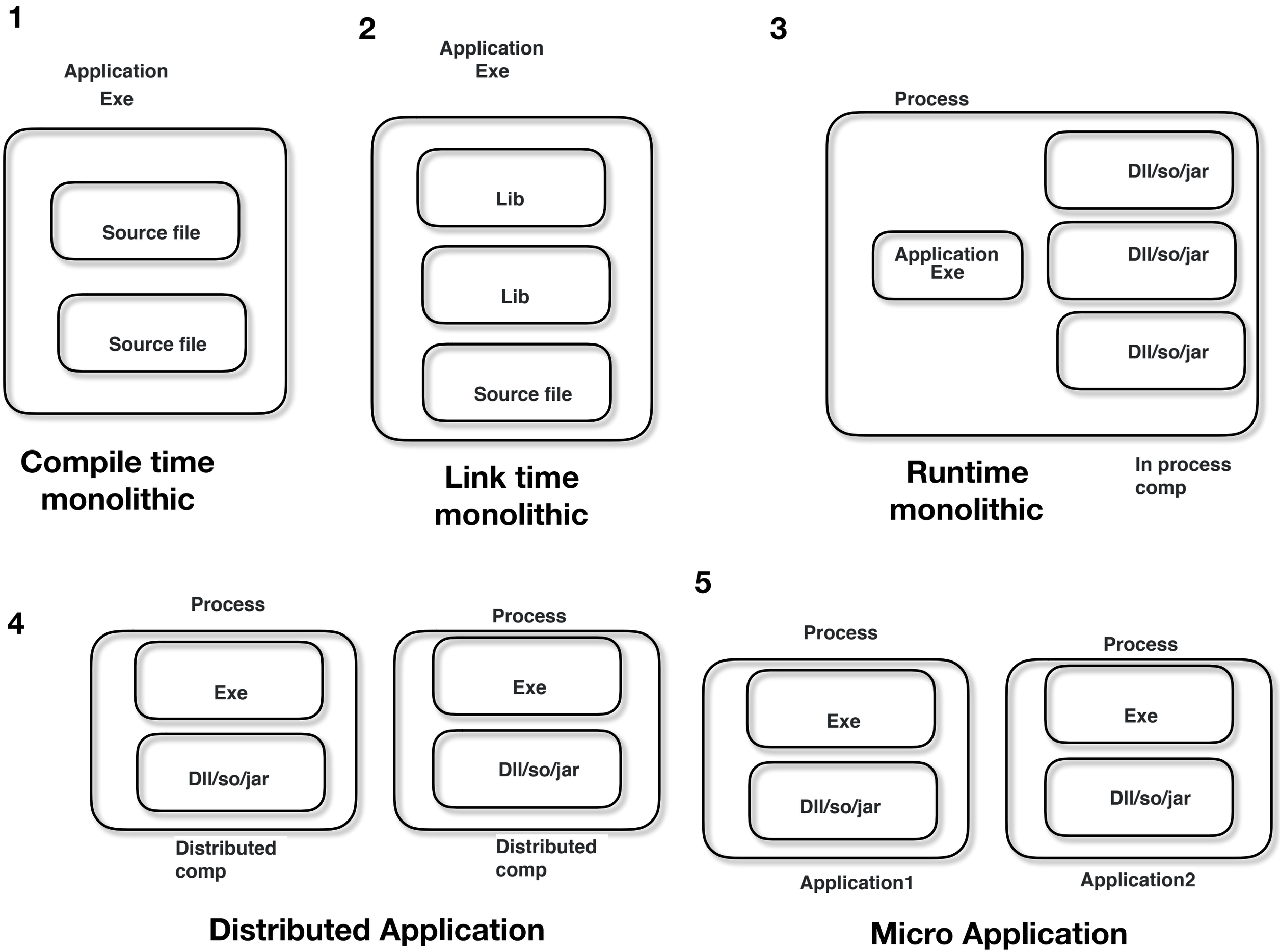

Process Data()
Pipe
Filter 1
Data1
f1(int)
Data2
Filter 2
f2(int)
Data3
Filter 3
f3(int)
Data4
Filter 4
f4(int)
Data5

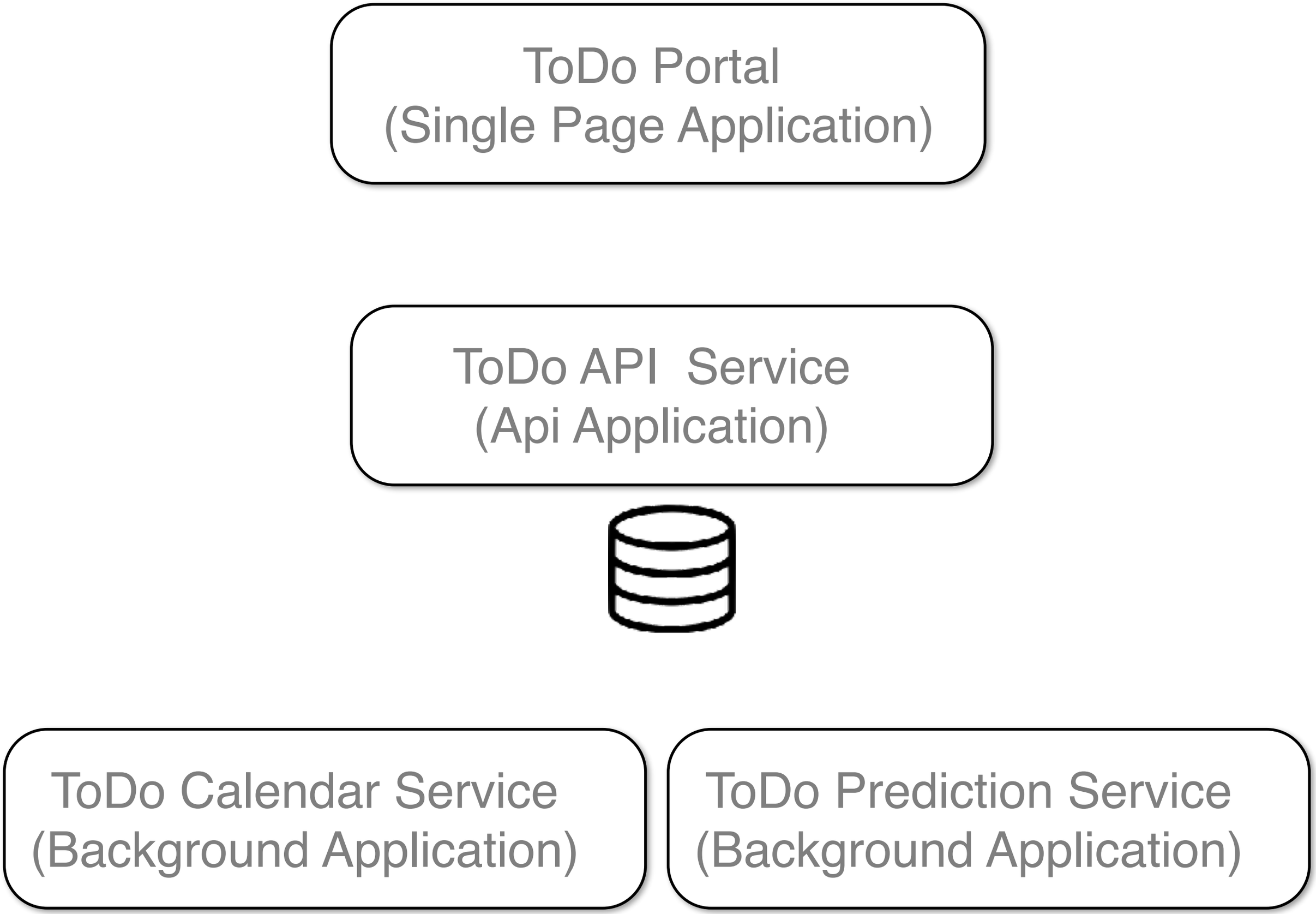

Invoke logic()
Layer 1
Data1
Layer 2
Data2
Layer 3
Data3
Layer 4
Data4
Data5

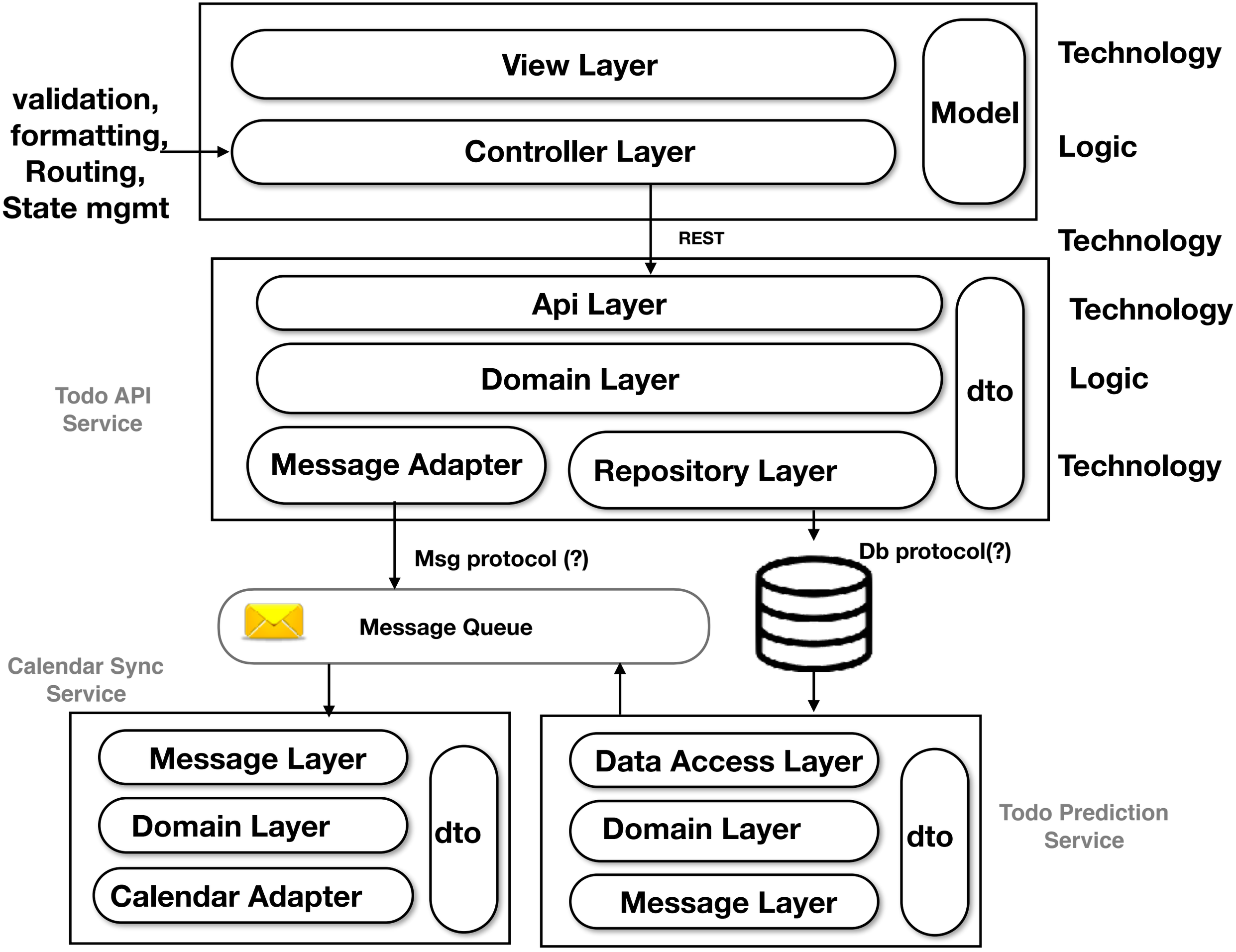

Process Data()
Filter 1
Data1
Pipe
Filter 2
Data2
Filter 3
Data3
Filter 4
Data4
Data5

Microkernel Architecture

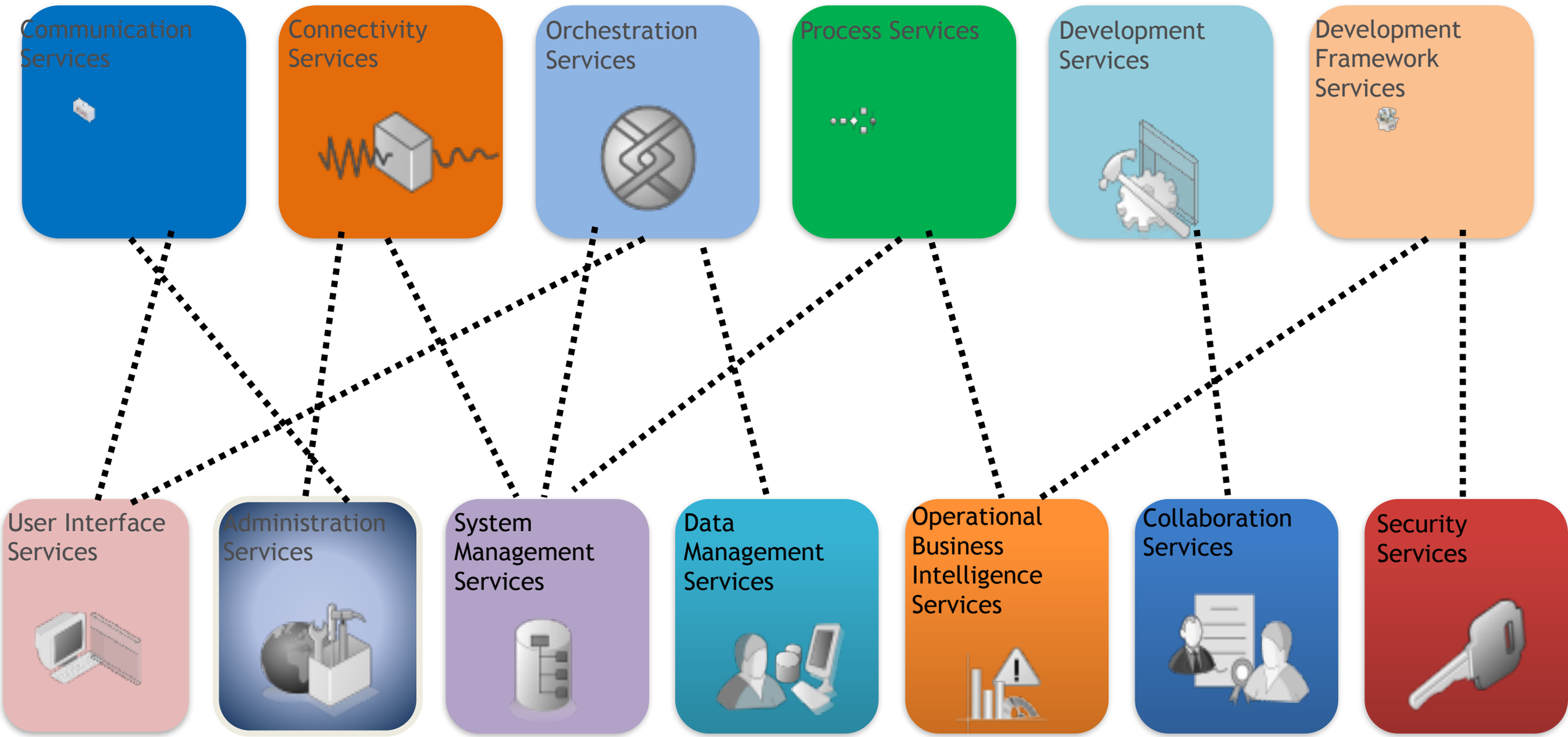


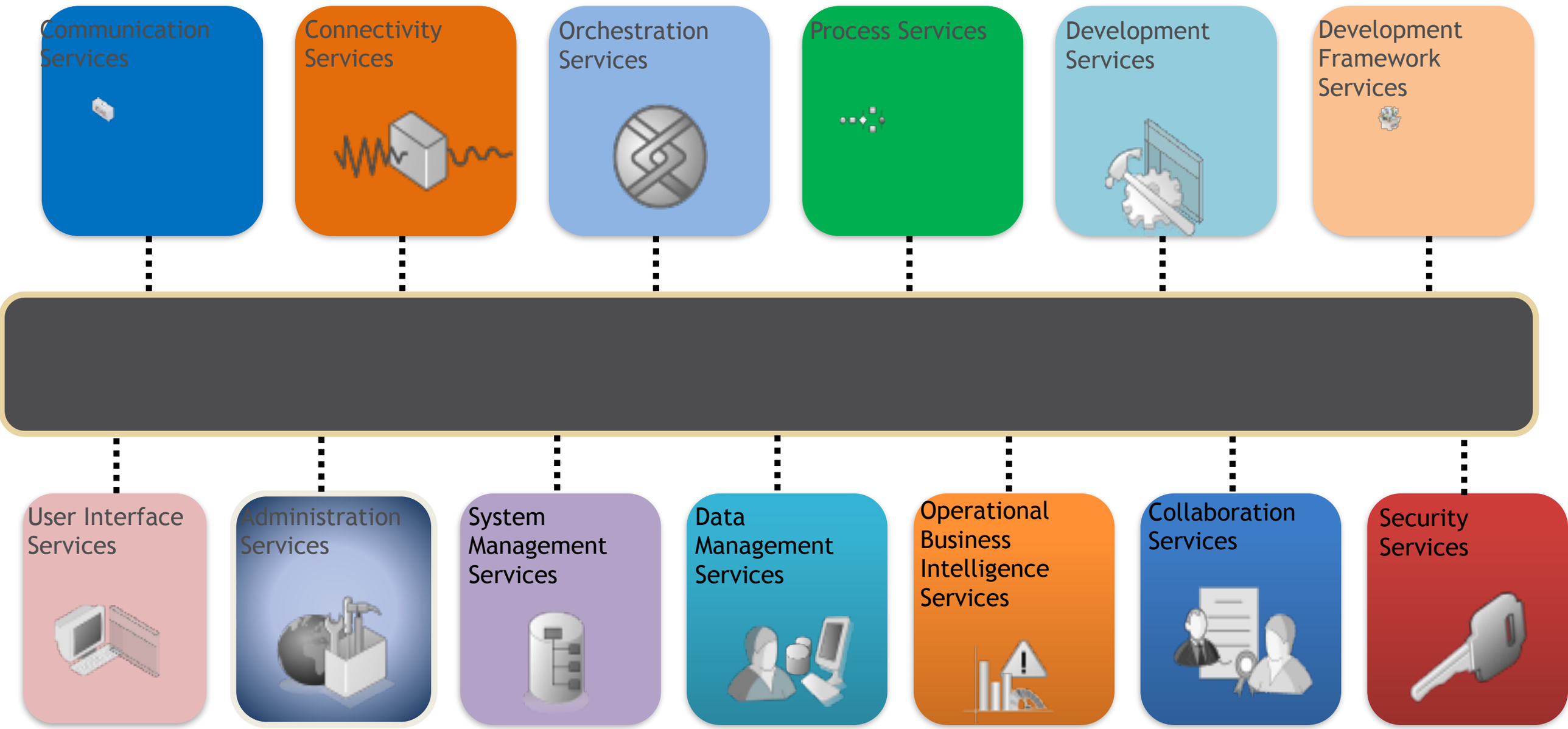

1
Application
Exe
Source file
Source file
Compile time
monolithic
2
Application
Exe
Lib
Lib
Source file
Link time
monolithic
3
Process
DII/so/jar
Application
Exe
DII/so/jar
DII/so/jar
Runtime
monolithic
In process
comp
4
Process
Exe
DII/so/jar
Distributed
comp
Process
Exe
DII/so/jar
Distributed
comp
Distributed Application
5
Process
Exe
DII/so/jar
Application1
Process
Exe
DII/so/jar
Application2
Micro Application


ToDo Portal
(Single Page Application)
ToDo API Service
(Api Application)
ToDo Calendar Service
(Background Application)
ToDo Prediction Service
(Background Application)

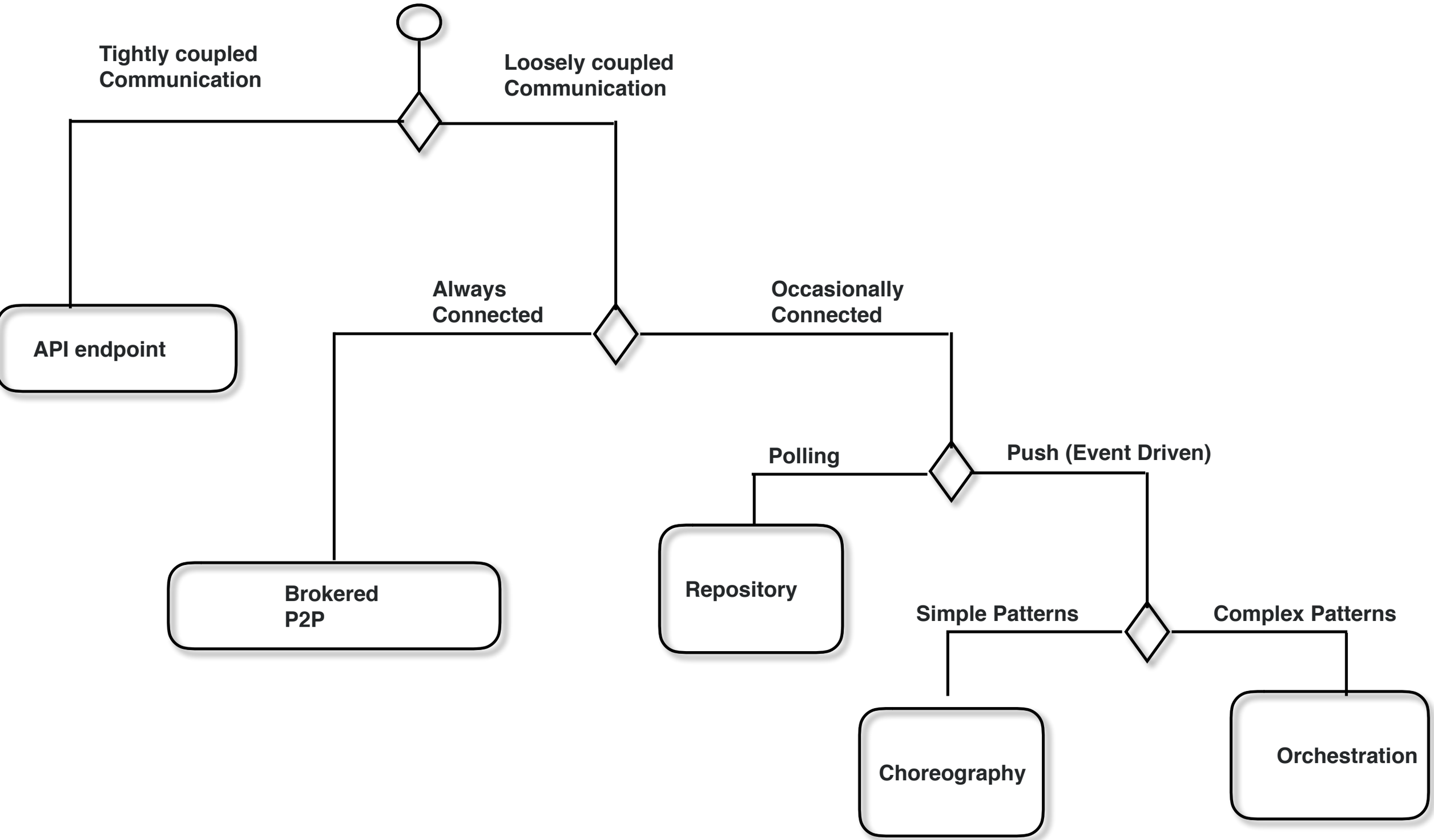

View Layer
Controller Layer
Model
Technology
Logic
validation,
formatting,
Routing,
State mgmt
REST
Technology
Api Layer
Domain Layer
Message Adapter
Repository Layer
dto
Technology
Logic
Technology
Todo API
Service
Msg protocol (?)
Db protocol(?)
Message Queue
Calendar Sync
Service
Message Layer
Domain Layer
Calendar Adapter
dto
Data Access Layer
Domain Layer
Message Layer
dto
Todo Prediction
Service

2. Integration View


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

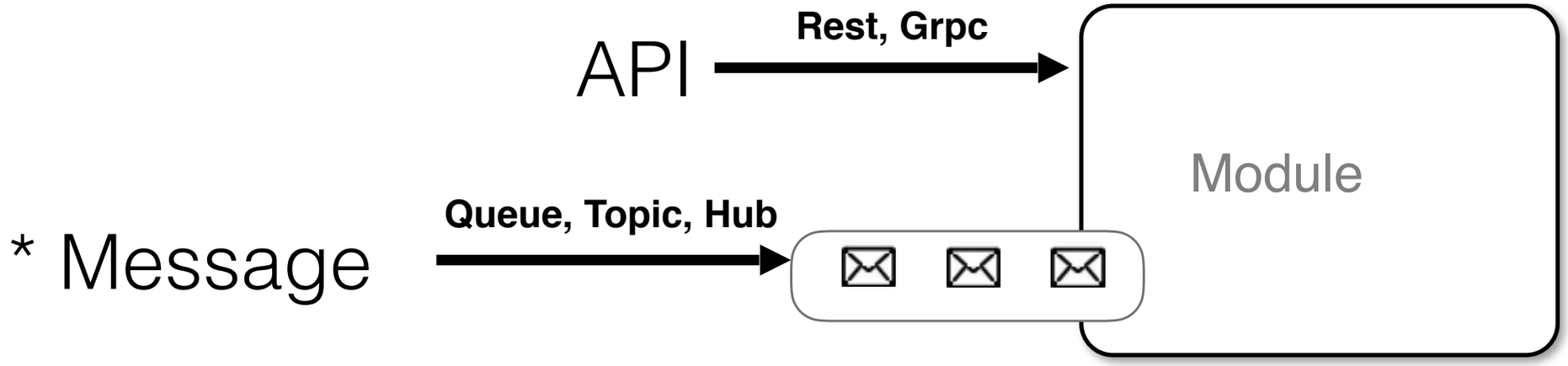

Communication Services
Connectivity Services
Orchestration Services
Process Services
Development Services
Development Framework Services
User Interface Services
Administration Services
System Management Services
Data Management Services
Operational Business Intelligence Services
Collaboration Services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

Choose Communication Mechanism

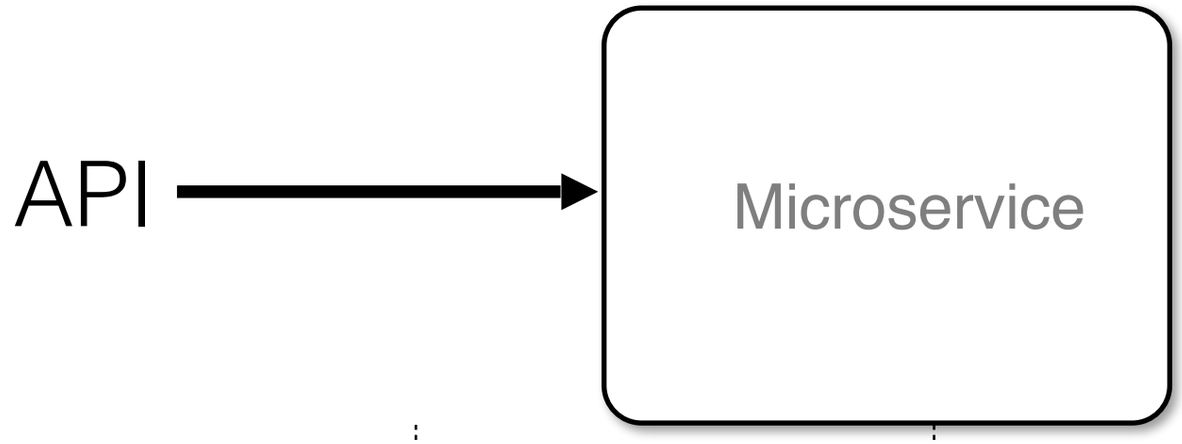


Connected protocol
(REST, WS, GRPC , Thrift)

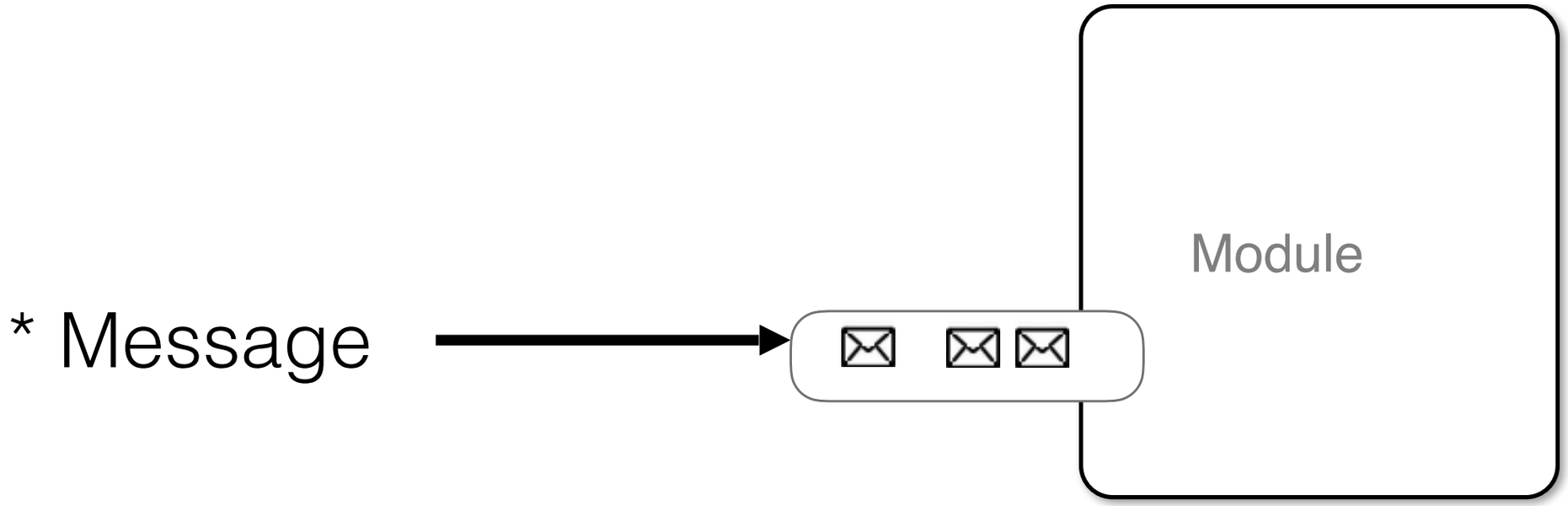
Message protocol
(AMQP, MQTT, …)



	<< 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	No	
Browser support	Yes	No	
Dev effort	++	- -	
Operational effort	+ +	- -	



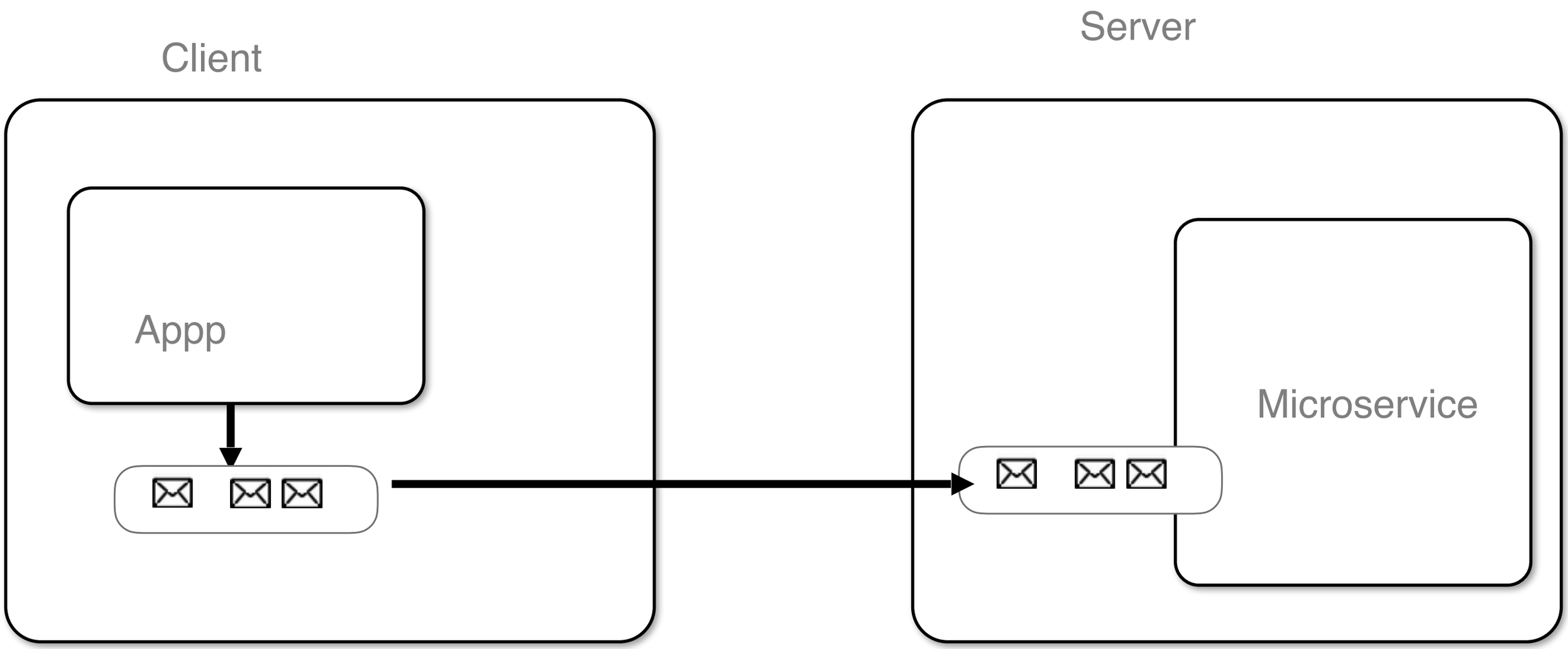
	REST	WSS	GRPC
Browser Support	**Y**	**Y**	**N**
Format	**TEXT**	**TEXT**	**BINARY, HTTP2**
Serialization	**JSON**	**JSON**	**Proto Buf**
Performance	**- -**	**+**	**+ +**
Use case	**North South**	**Streaming**	**East West**

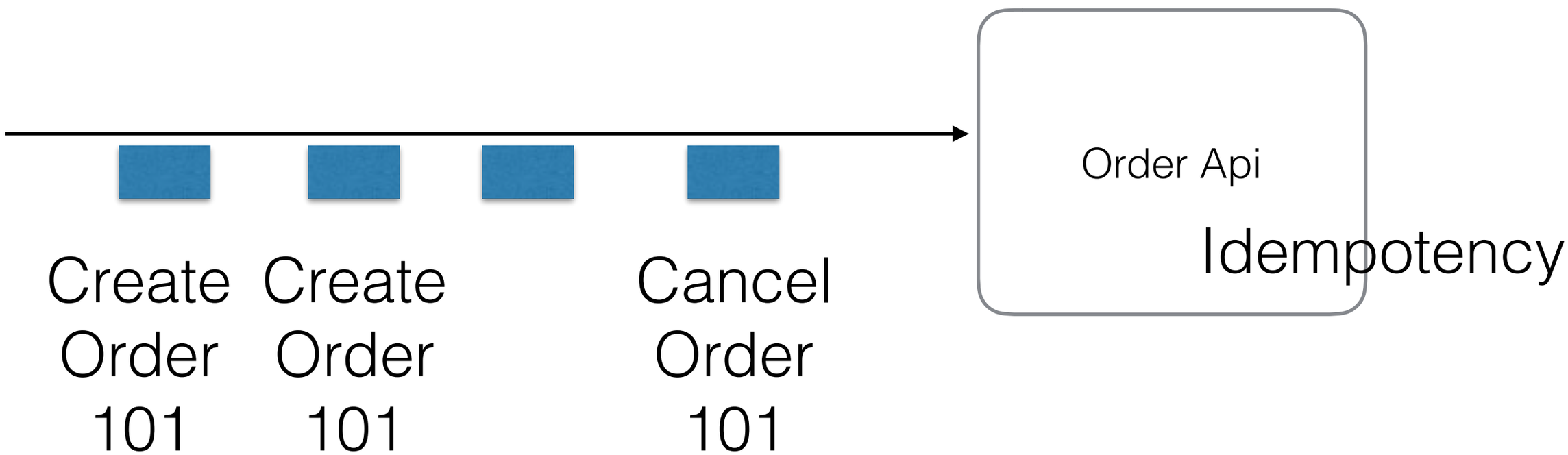


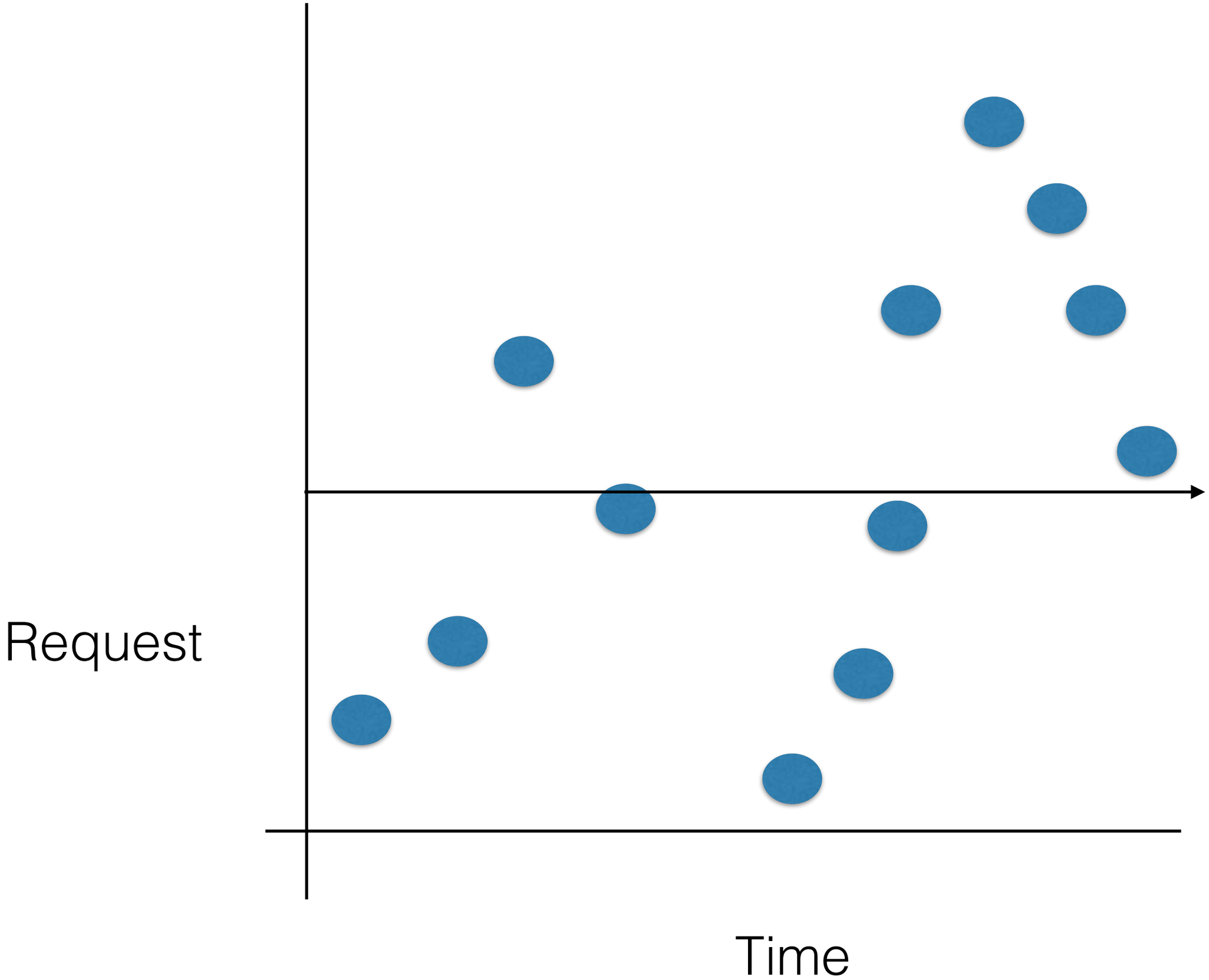

```
fun(){
    while(true){
        Msg m = GetMessage();
        .....
        m.ack();
    }
}
```

MSMQ

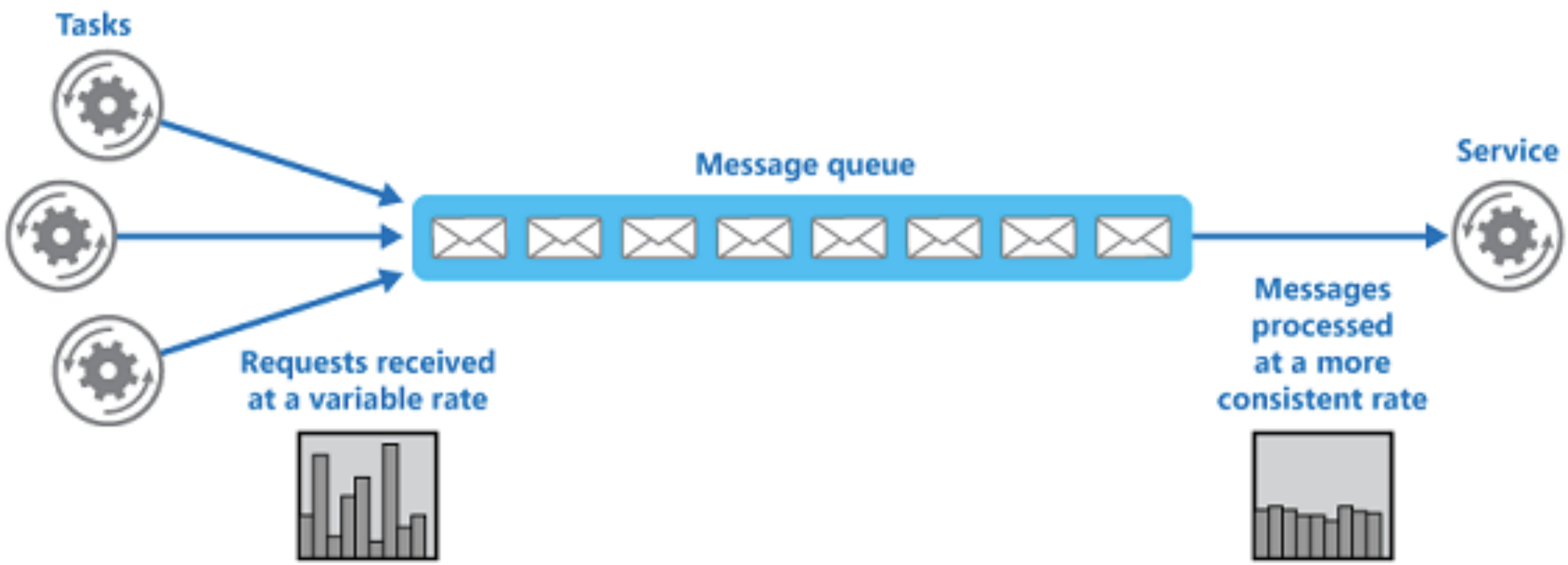
* Message




Order Api
Idempotency
Create Order 101
Create Order 101
Cancel Order 101


Request
Time

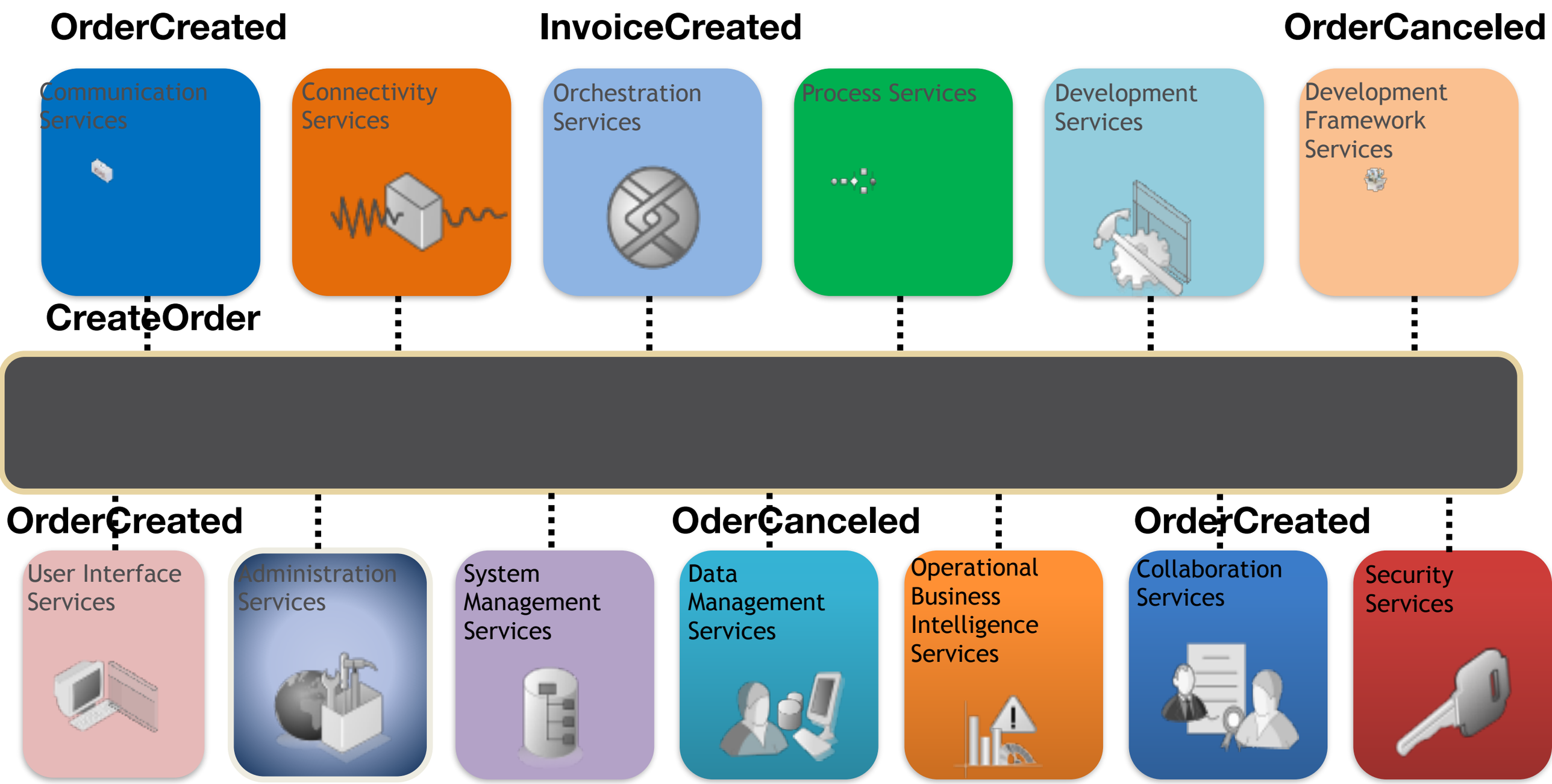
Queue-based Load Levelling



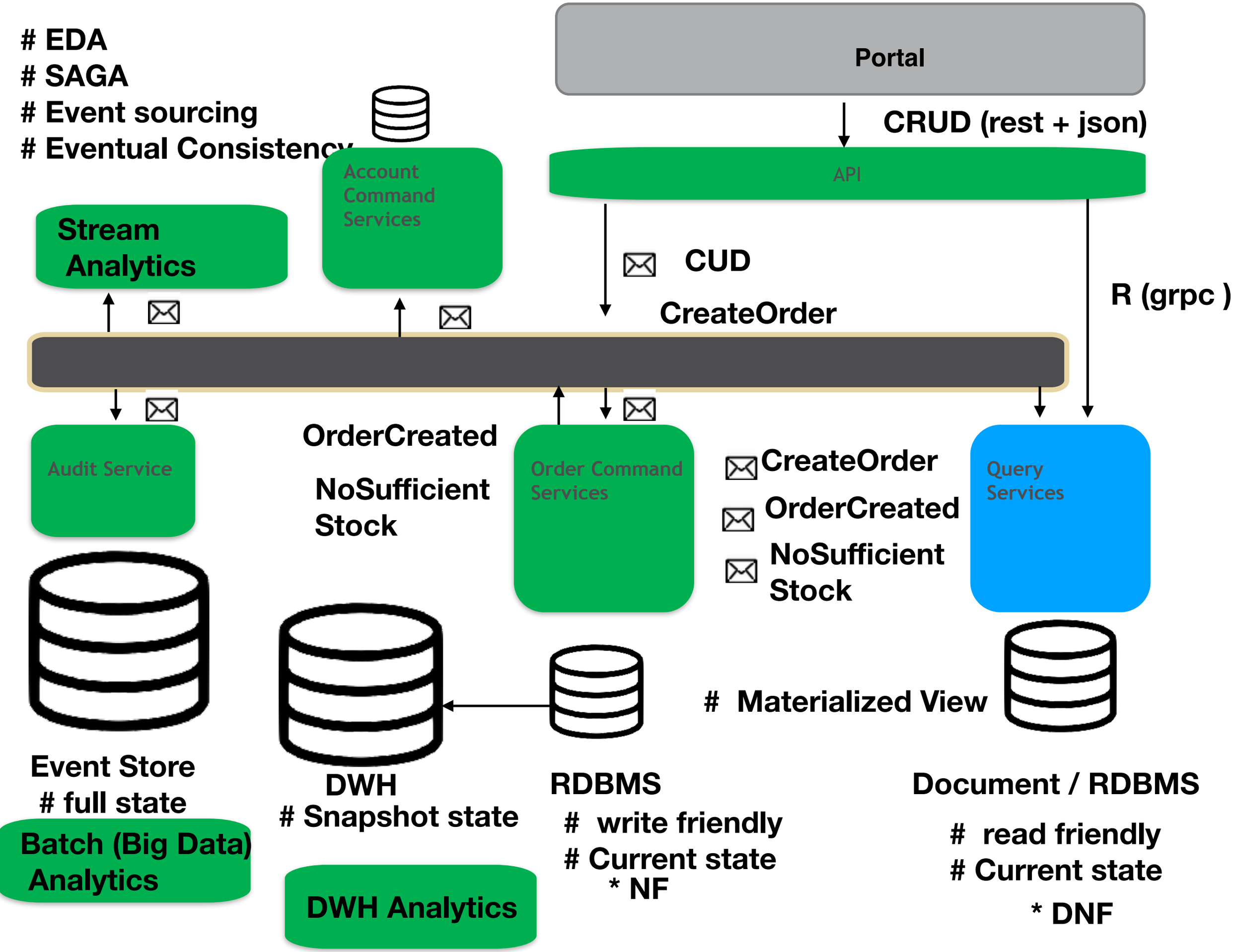
source:msdn

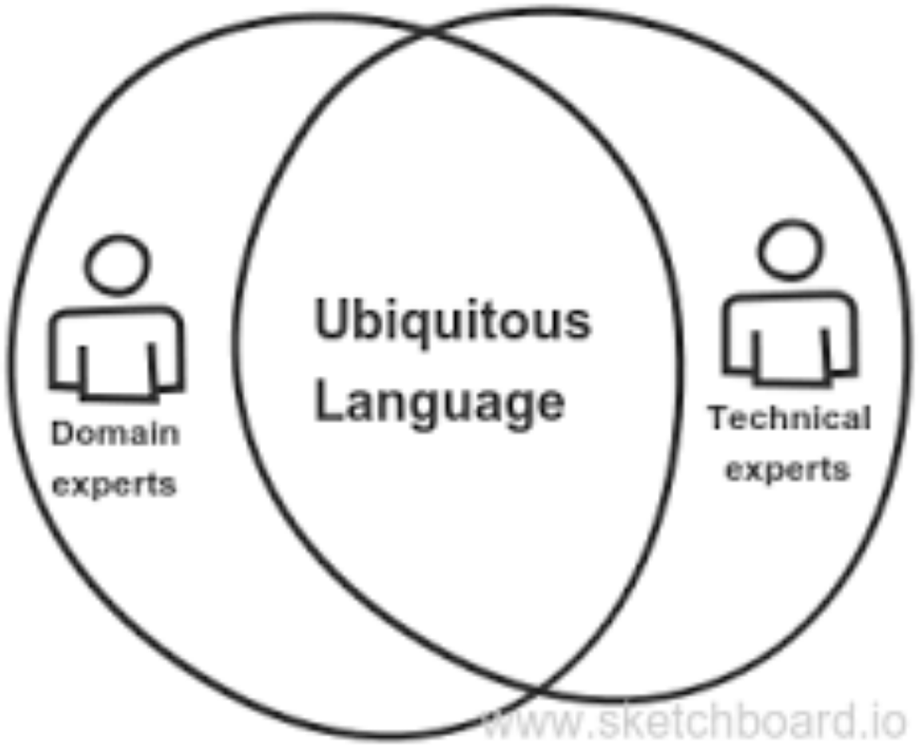
Event vs Command

Post **Pre**



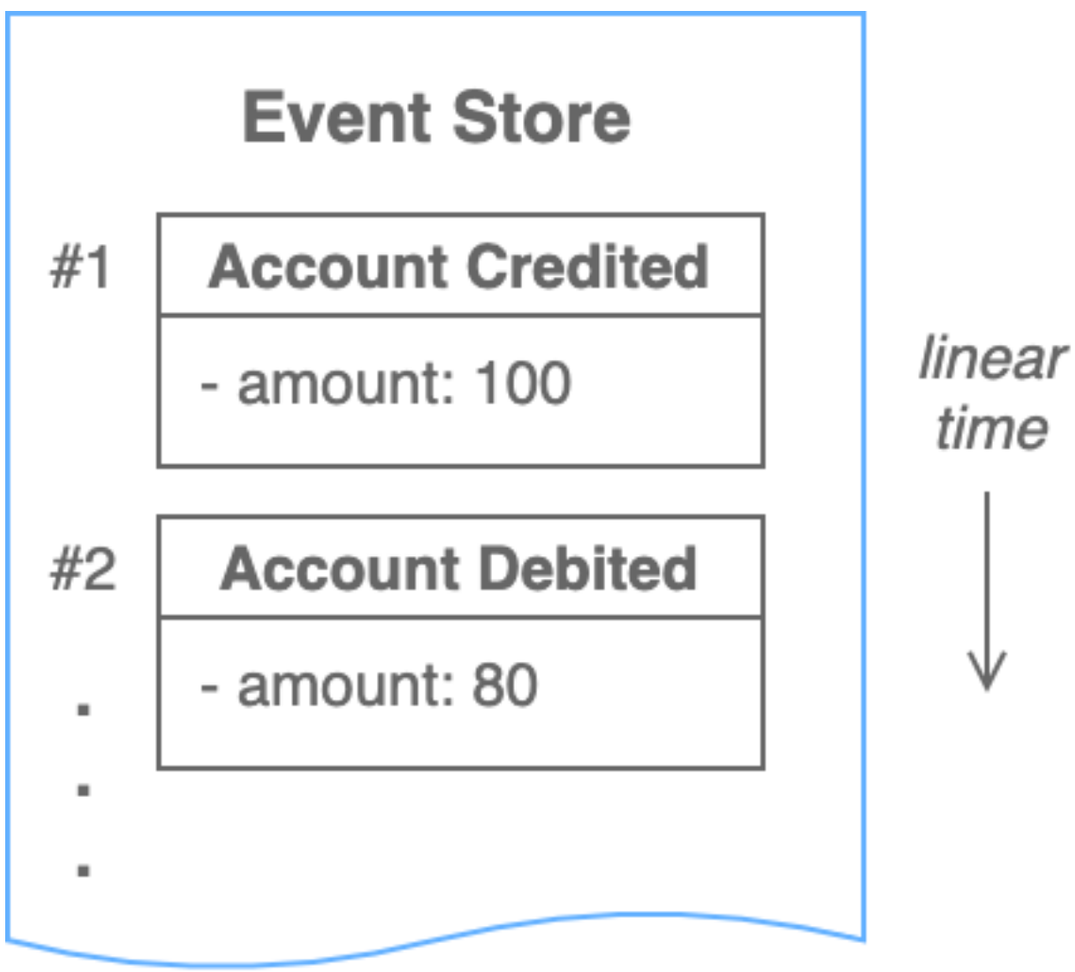
EDA
SAGA
Event sourcing
Eventual Consistency



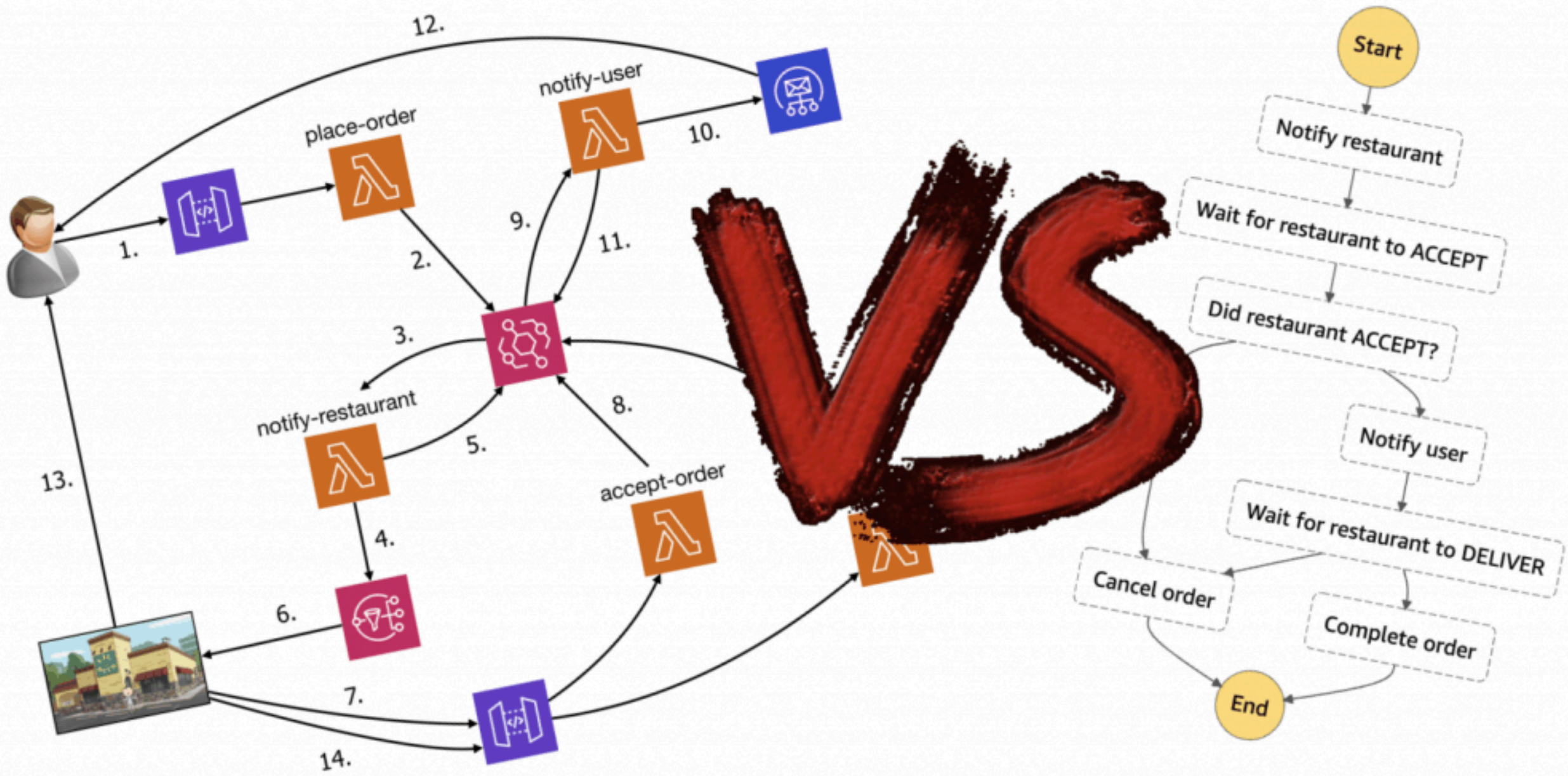


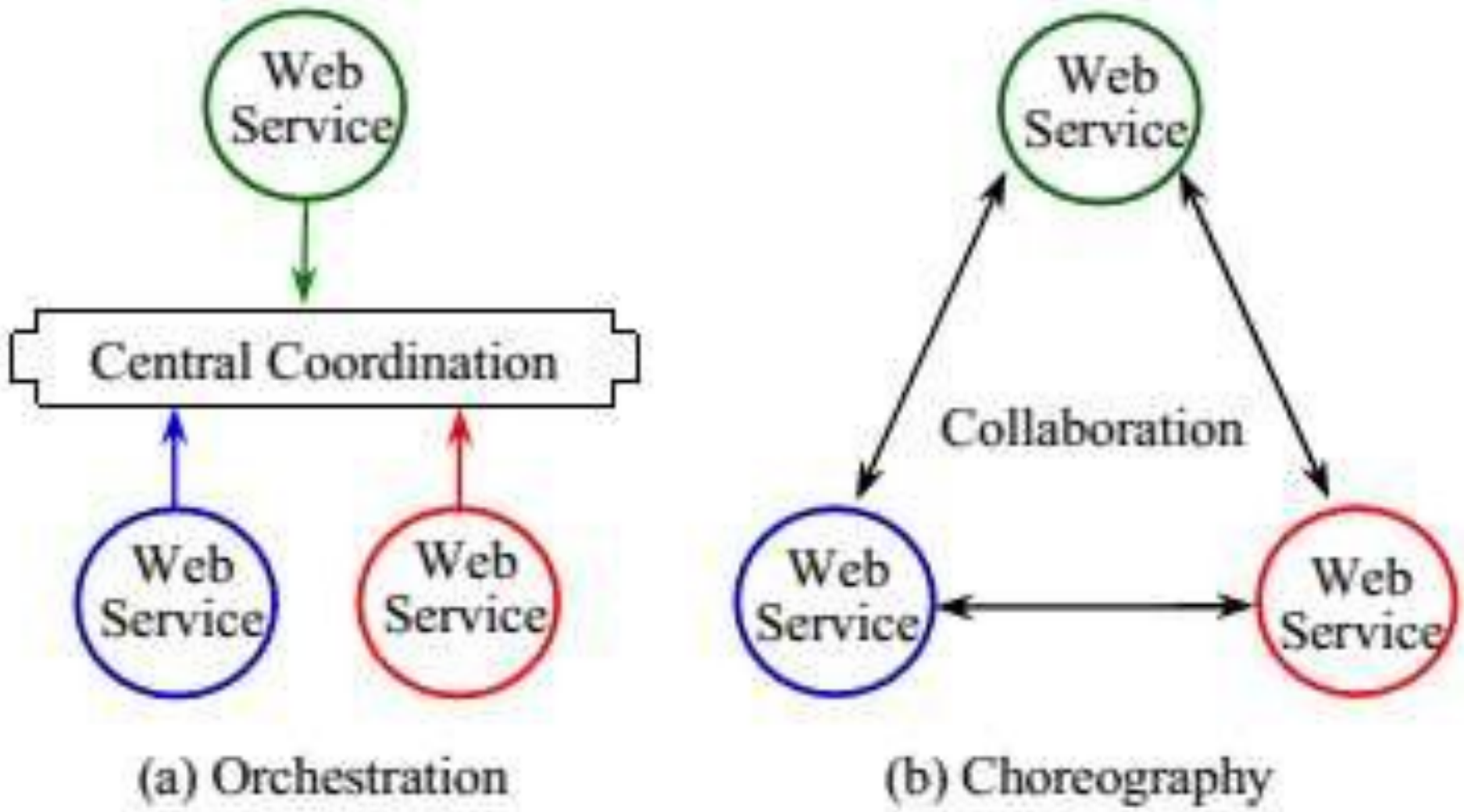
ubiquitous language

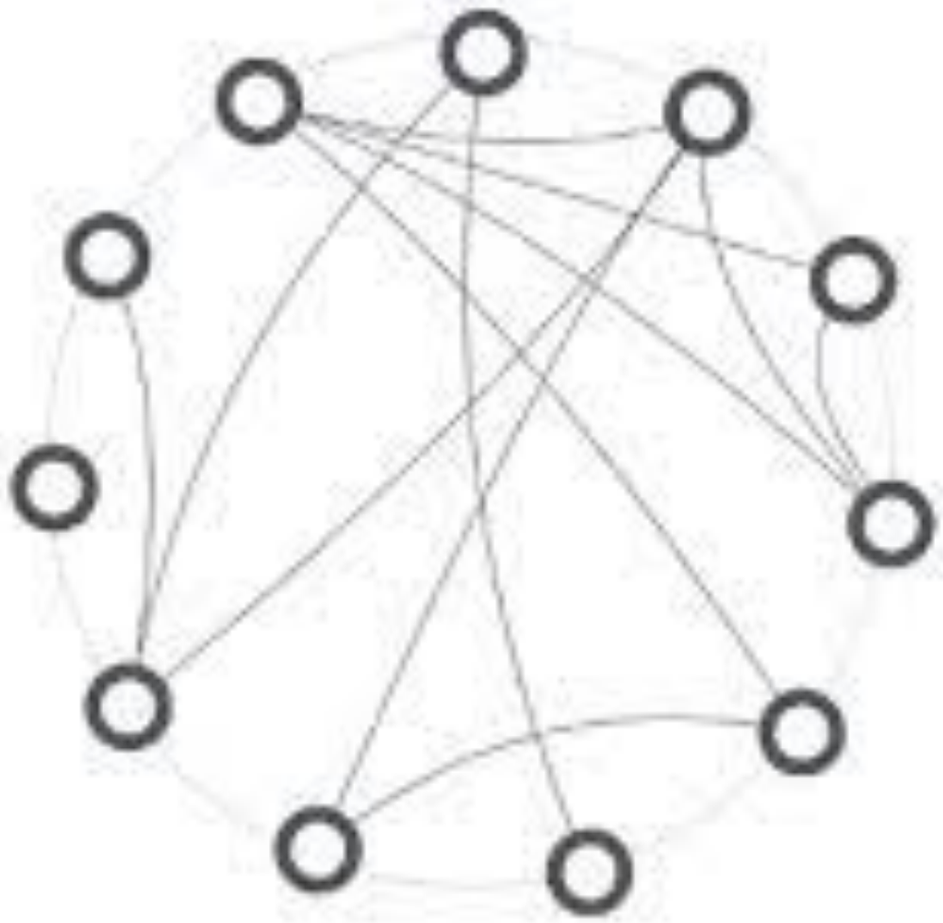
DDD

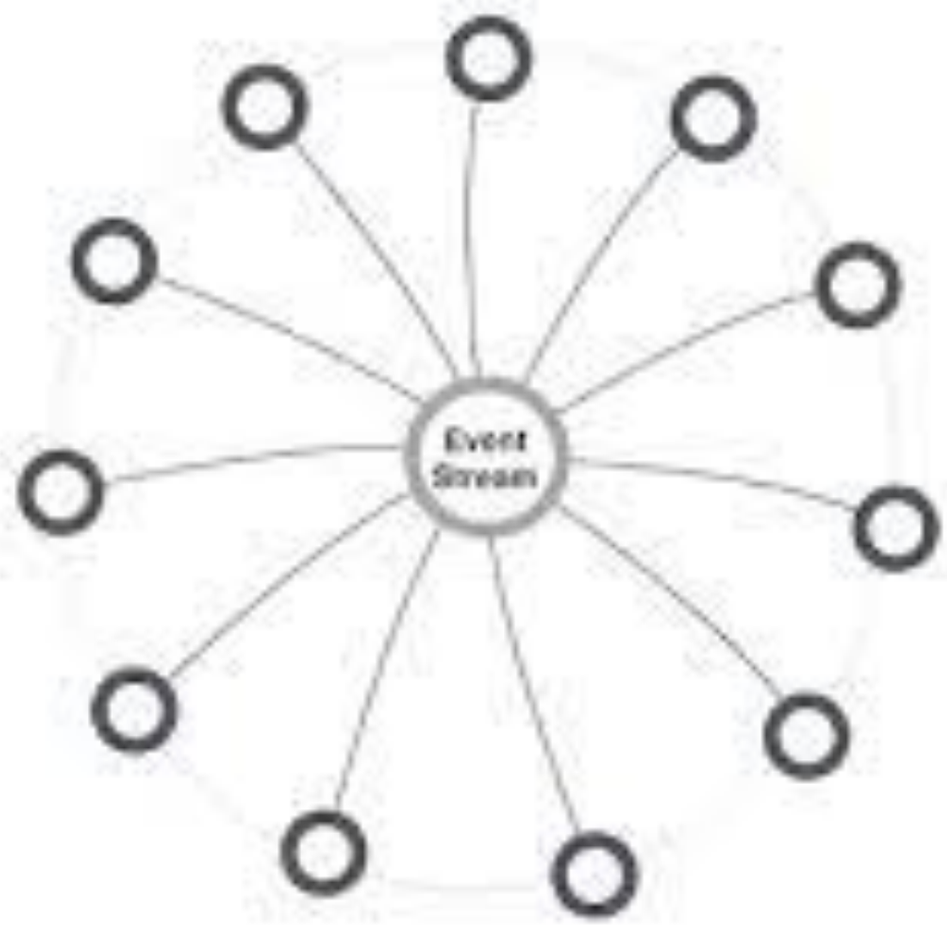


Event Sourcing

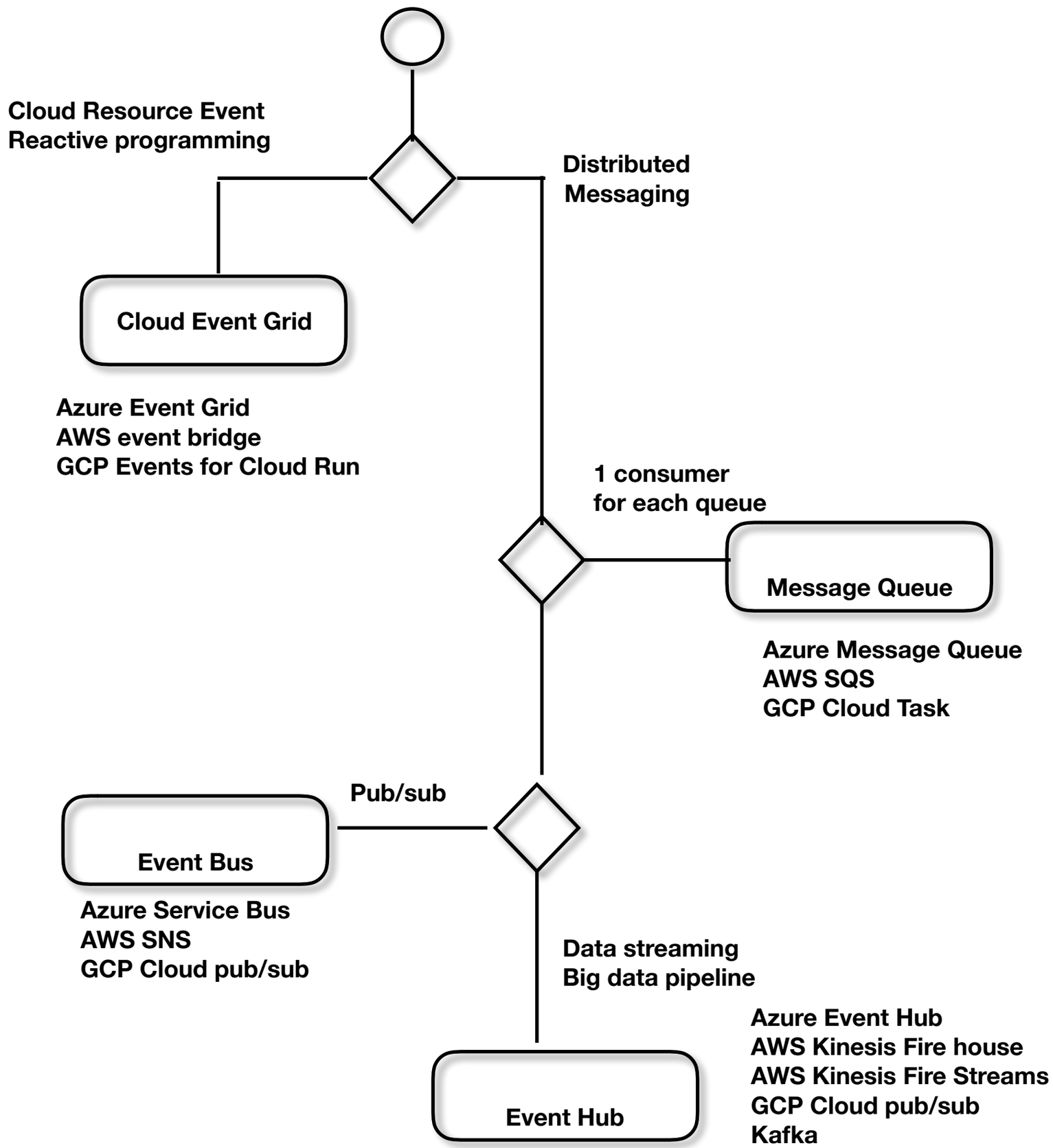

12.
notify-user
place-order
10.
1.
2.
9.
11.
3.
notify-restaurant
8.
5.
13.
accept-order
4.
6.
7.
14.
VS
Start
Notify restaurant
Wait for restaurant to ACCEPT
Did restaurant ACCEPT?
Notify user
Wait for restaurant to DELIVER
Cancel order
Complete order
End

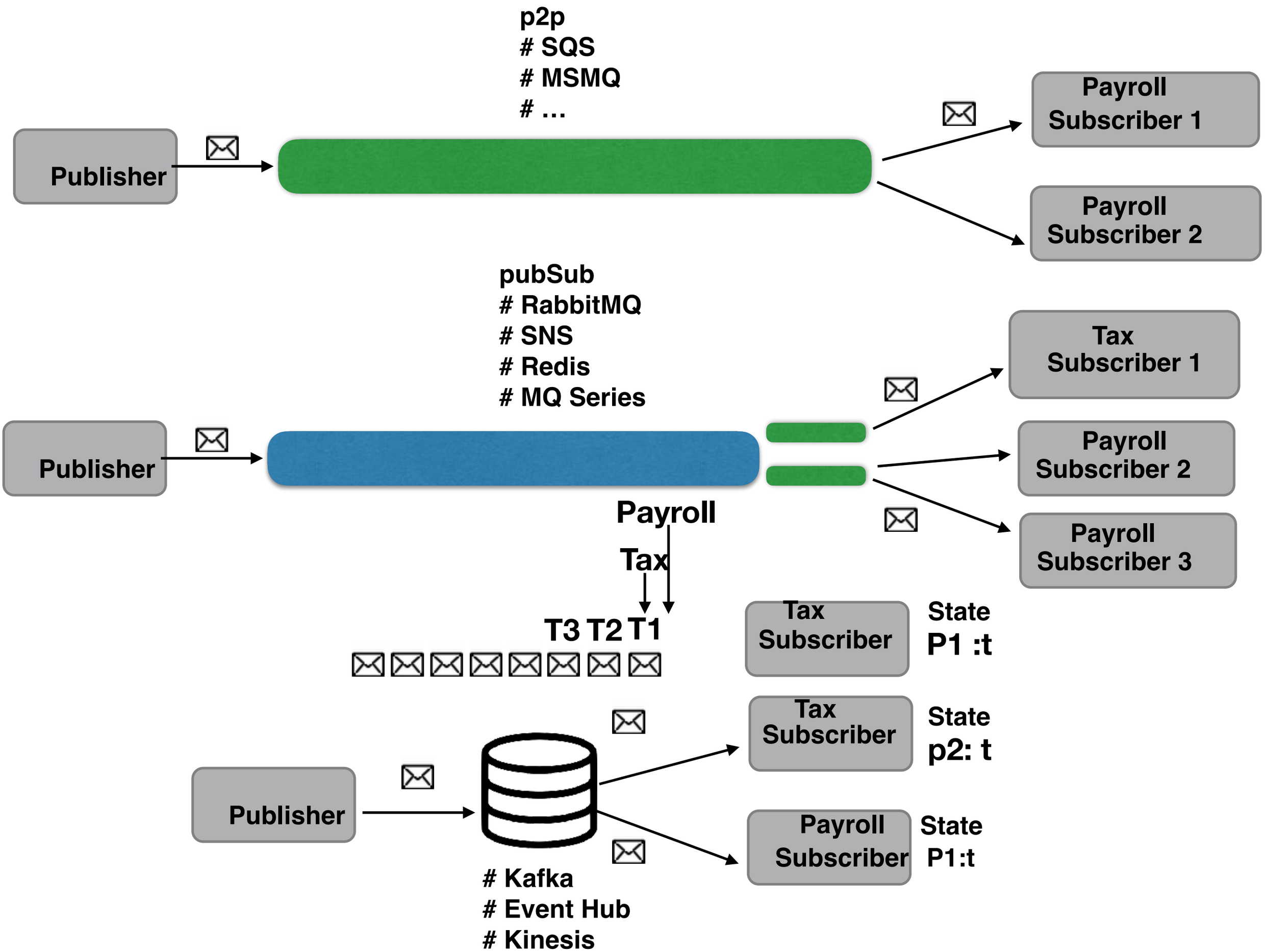

Web Service
Central Coordination
Web Service
Web Service
(a) Orchestration
Web Service
Collaboration
Web Service
Web Service
(b) Choreography

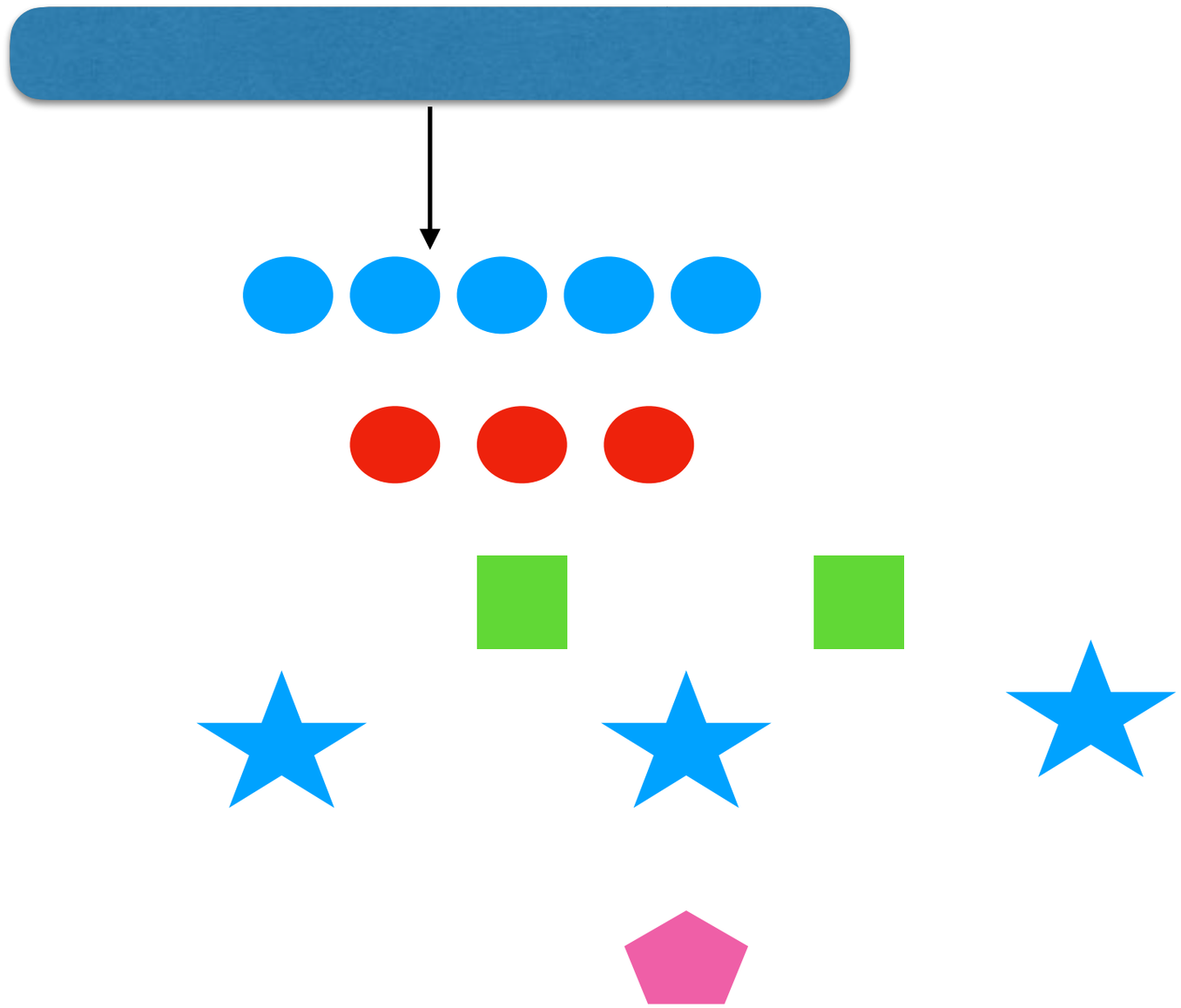



Event
Stream

Choose Message Engine



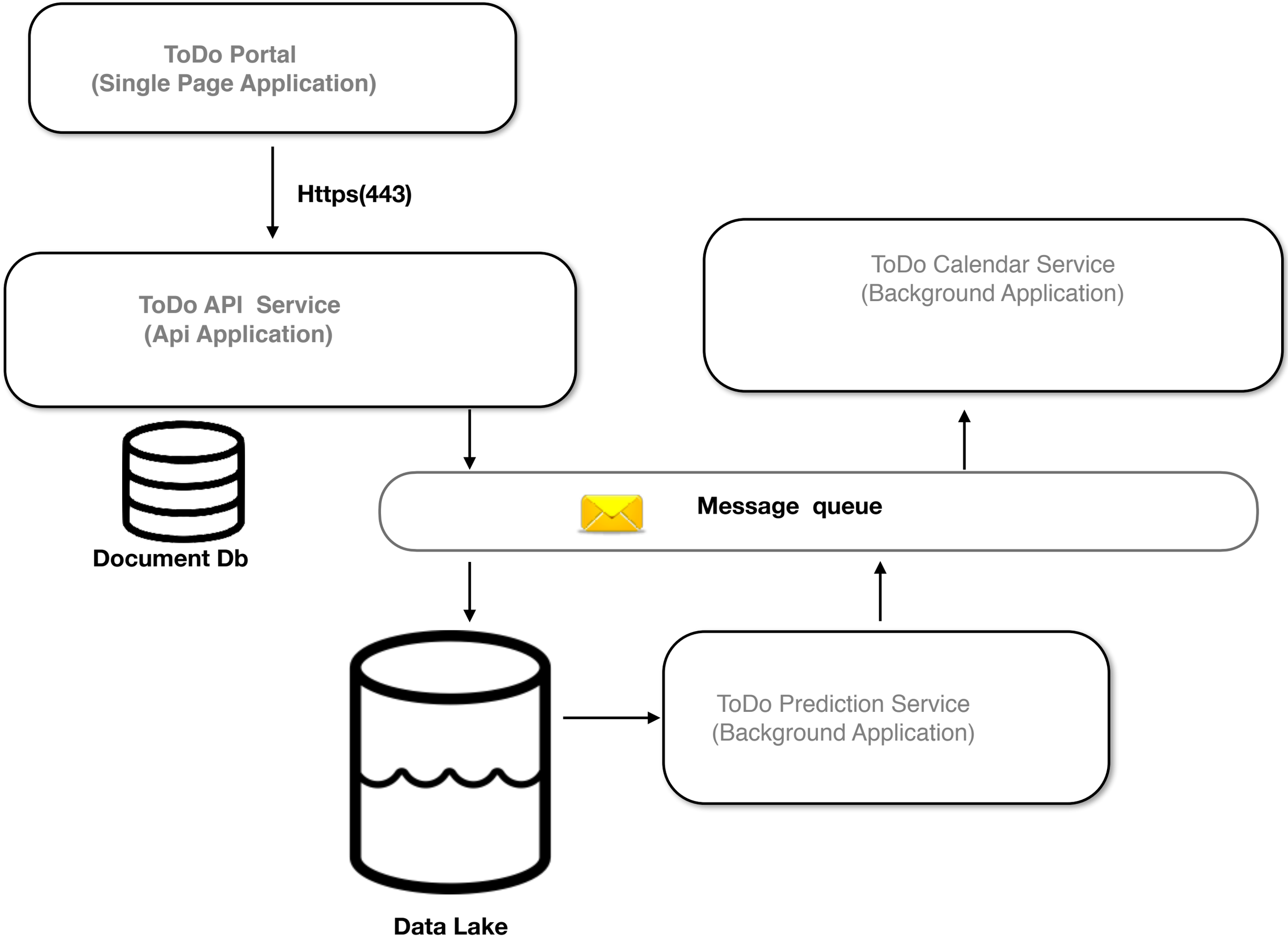

p2p
SQS
MSMQ
...
Publisher
Payroll Subscriber 1
Payroll Subscriber 2
pubSub
RabbitMQ
SNS
Redis
MQ Series
Publisher
Tax Subscriber 1
Payroll Subscriber 2
Payroll Subscriber 3
Payroll
Tax
T3 T2 T1
Tax Subscriber
State P1 :t
Tax Subscriber
State p2: t
Publisher
Payroll Subscriber
State P1:t
Kafka
Event Hub
Kinesis



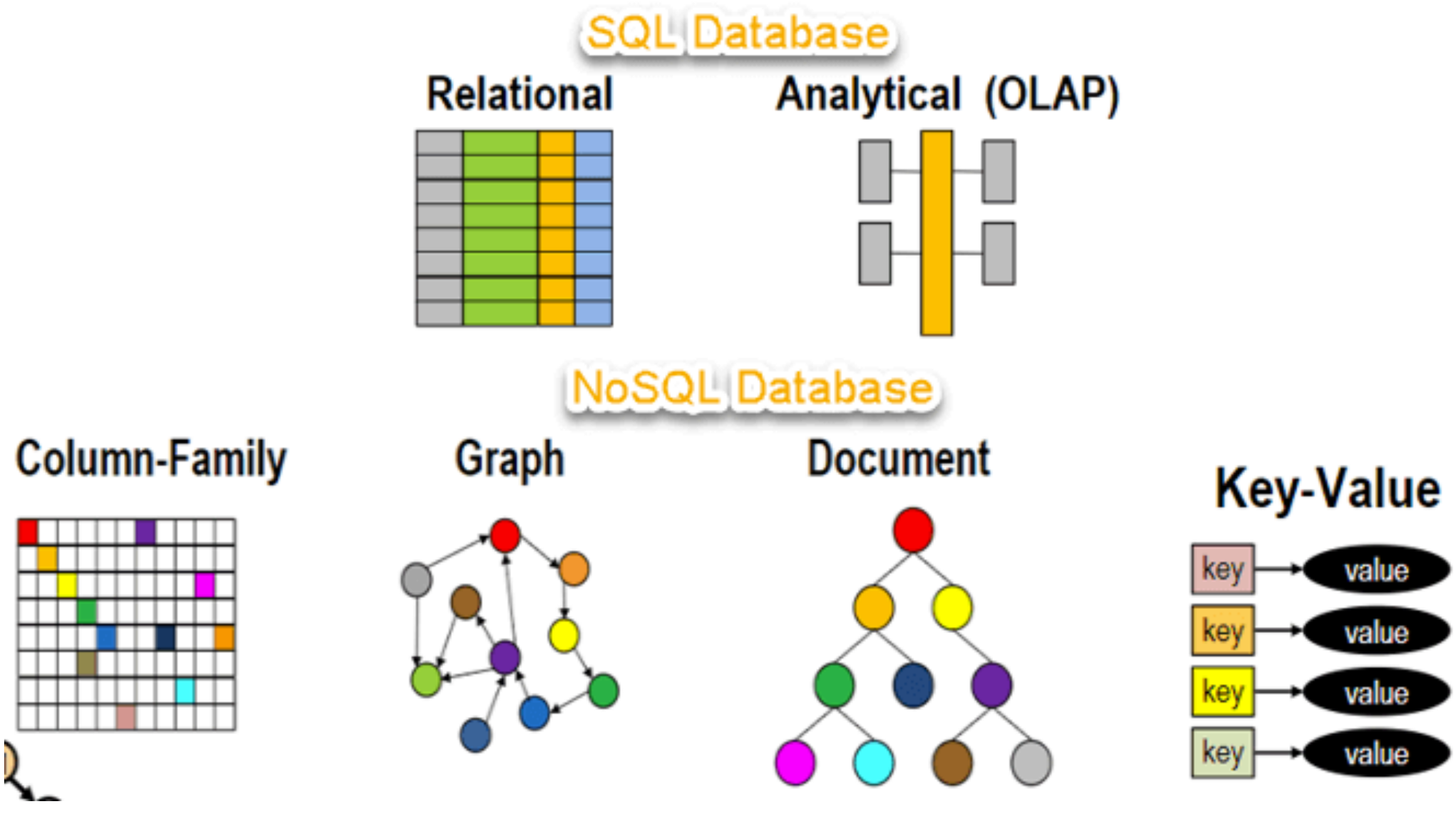
	Rabbitmq	Kafka	Redis
Scale	50K msg per second	1 million/ sec	1 million/ sec
Transient messages	Yes	No	Y
Persistent yes	Yes	Yes	Y
Protocol	AMQP		
One to one	Y	N	Y
One to many	Y	Y	Y
Advanced Message Queueing Protocol-based routing	Y	N	N
consumption pattern	Pull+Push	Pull	
Client Libraries	Py, java, php,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


ToDo Portal
(Single Page Application)
Https(443)
ToDo API Service
(Api Application)
ToDo Calendar Service
(Background Application)
Document Db
Message queue
ToDo Prediction Service
(Background Application)
Data Lake

3. Data View


SQL Database
Relational
Analytical (OLAP)
NoSQL Database
Column-Family
Graph
Document
Key-Value
key
value
key
value
key
value
key
value

Db	IAAS	PAAS
Manage db server (install, upgrade, ..)	Me	Vendor
backup/recovery	Me	Vendor (3 copies of backup)
Availability	Me	Vendor
Monitoring	Me	Vendor

LOB

Structured

RDBMS

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

Semi Structured

Un Structured

- Binary storage

Dictionary (key value)

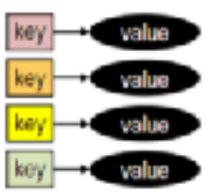
Persistent

Cache

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

Key Value

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



k1->{.....}
k2->{.....}
k3->{.....}
k4->{.....}

Full Text Search

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

{...{..{ .. } ...}..}
{...{..{ .. } ...}..}
{...{..{ .. } ...}..}

Nested Objects

data store

Document

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

Entity Relationship

Graph

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

Rows vs. Documents

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

Document 1
```
{
 data
}
```

Document 2
```
{
 data
}
```

Document 3
```
{
 data
}
```

...

Columns vs. Properties

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

Document 1
```
{
 "prop1": data,
 "prop2": data,
 "prop3": data,
 "prop4": data
}
```

Document 2
```
{
 "prop1": data,
 "prop2": data,
 "prop3": data,
 "prop4": data
}
```

Document 3
```
{
 "prop1": data,
 "prop2": data,
 "prop3": data,
 "prop4": data
}
```

Schema vs. Schema-Free

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

Document 1
```
{
 "id": "1",
 "name": "John Smith",
 "isActive": true,
 "dob": "1964-30-08"
}
```

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

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

Normalized vs. Denormalized

User Table

UserID	Name	Dob
1	John Smith	8/30/1964

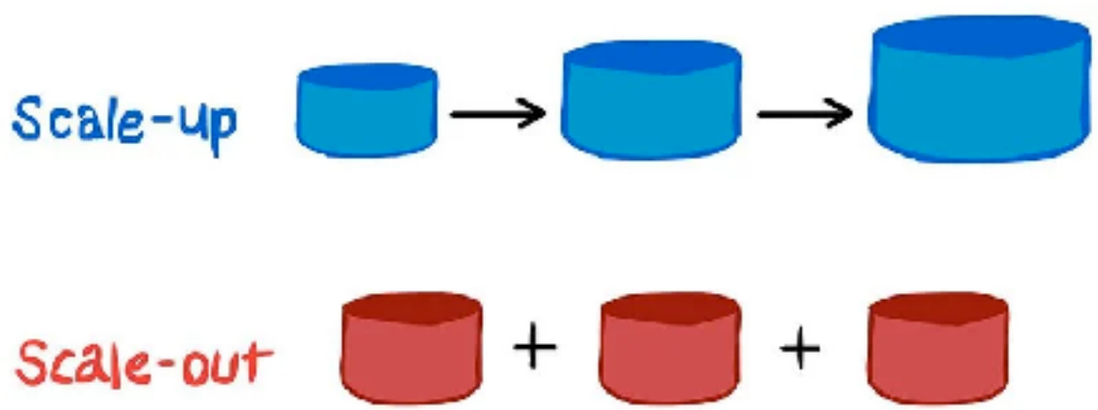
Holdings Table

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

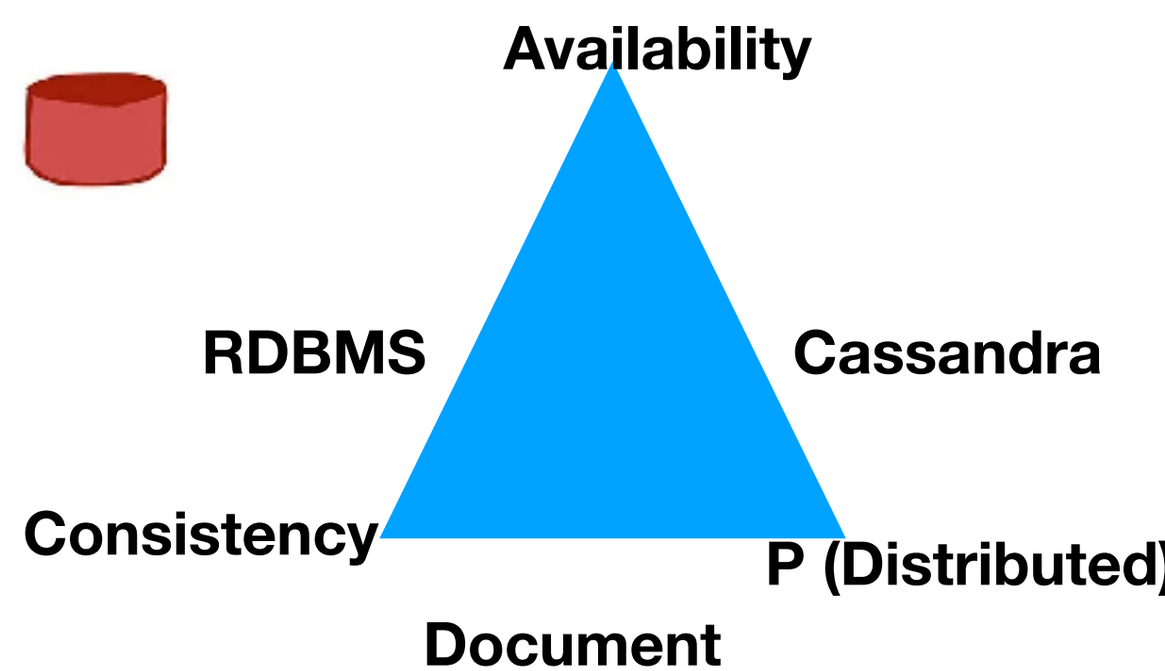
Document

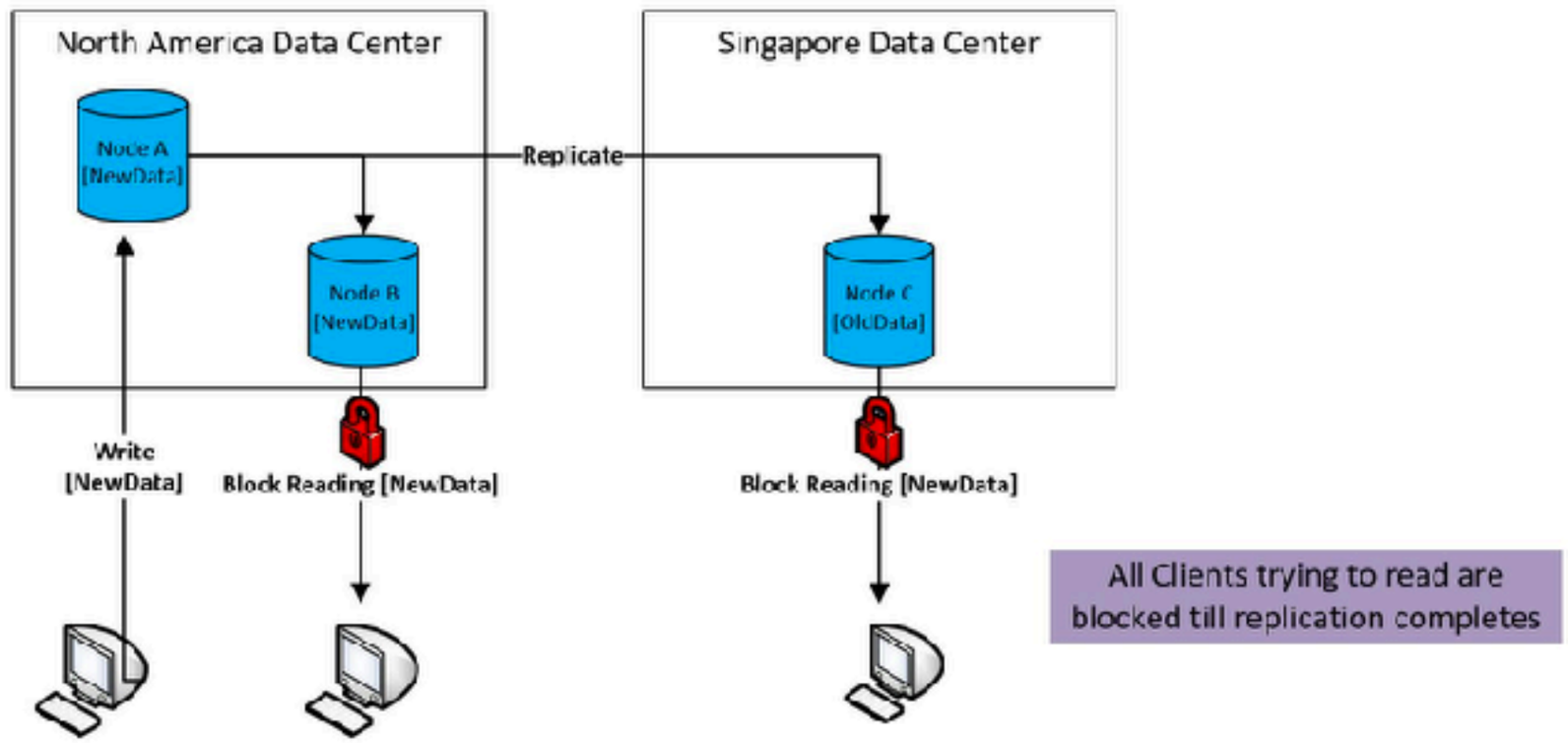
```
{
  "id": "1",
  "name": "John Smith",
  "dob": "1964-30-08",
  "holdings": [
    { "qty": 100, "symbol": "MSFT" },
    { "qty": 75, "symbol": "WMT" }
  ]
}
```

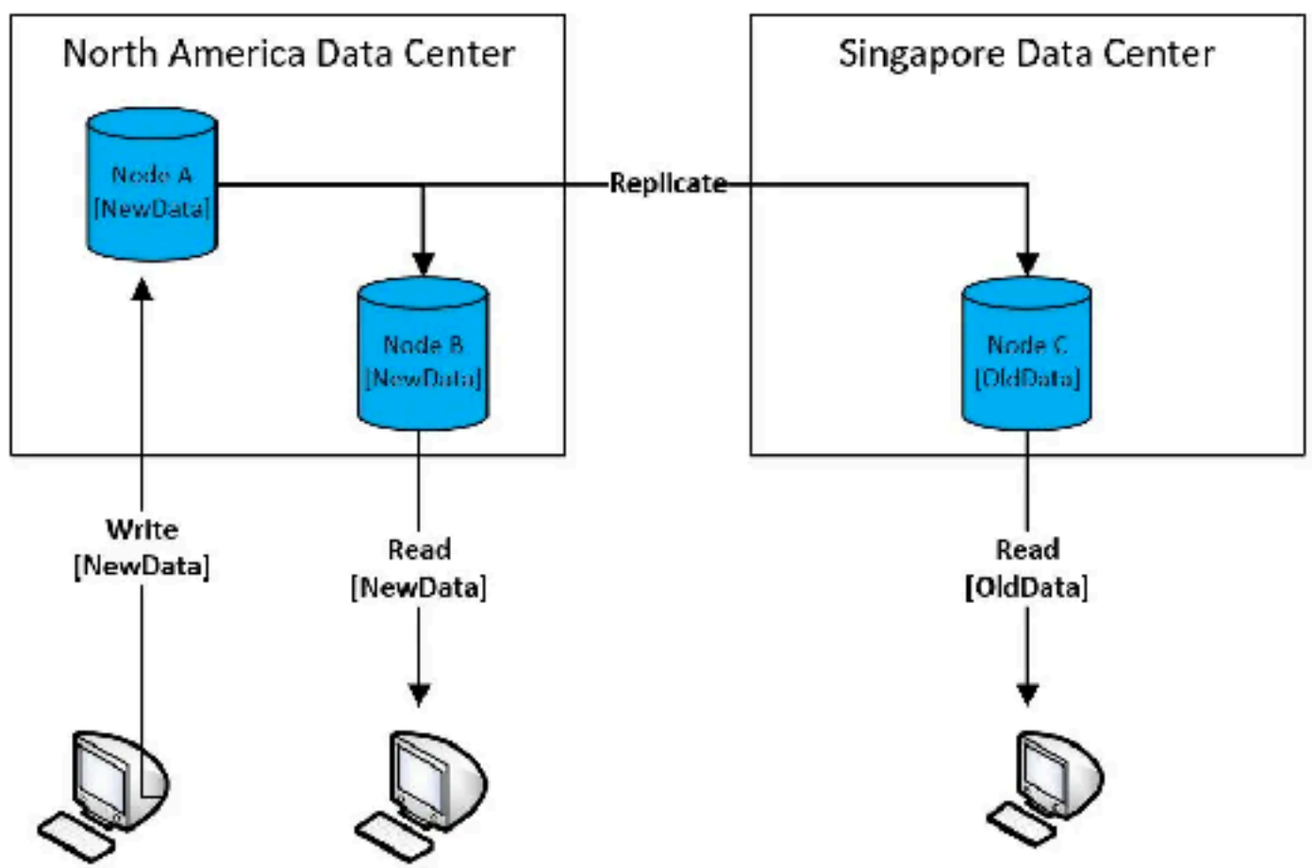
Scale-Up vs. Scale-Out



Strong Consistency vs. Eventual Consistency




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

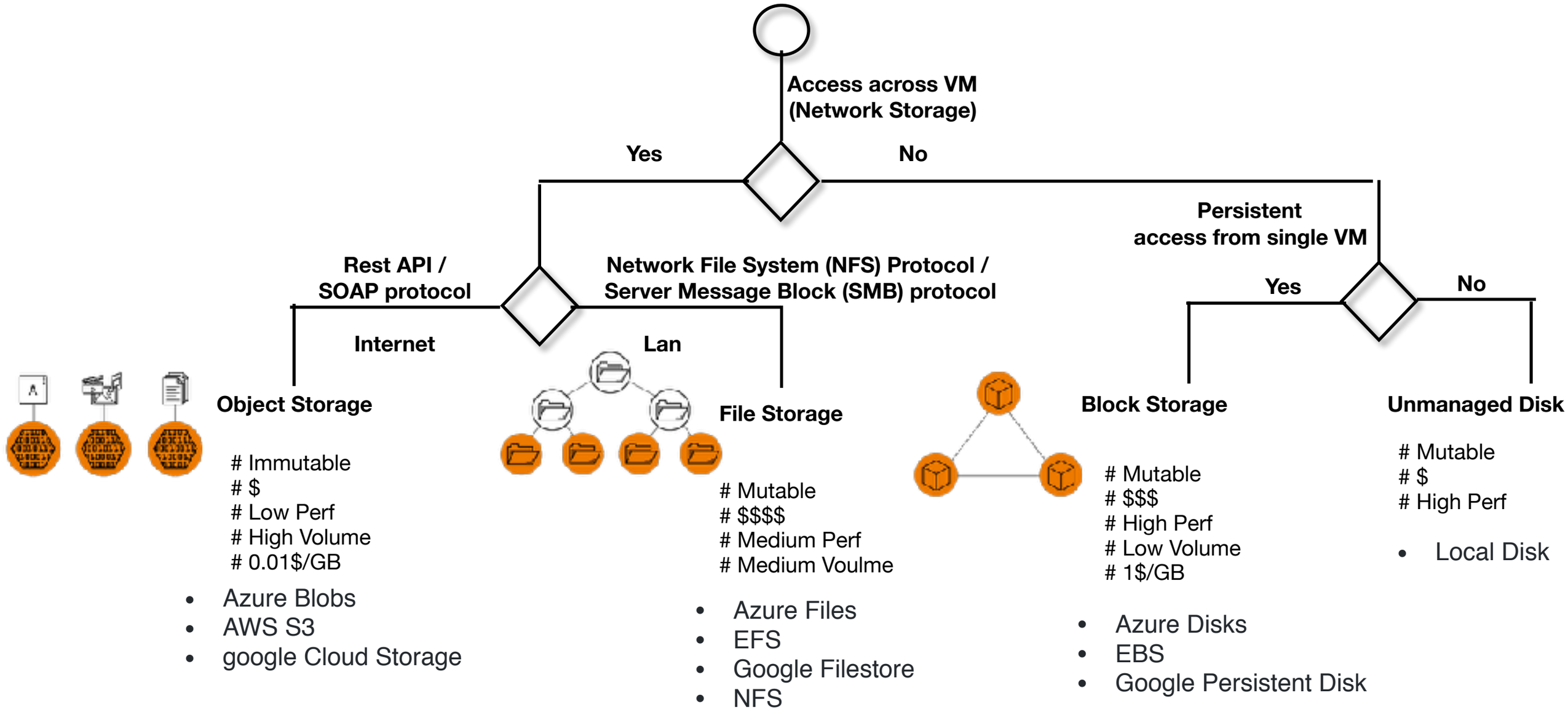

North America Data Center
Singapore Data Center
Node A
[NewData]
Replicate
Node B
[NewData]
Node C
[OldData]
Write
[NewData]
Read
[NewData]
Read
[OldData]

COMMON COMPARISONS BETWEEN

MYSQL & NOSQL

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

Binary Storage



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

Analytical

Structured

DWH

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

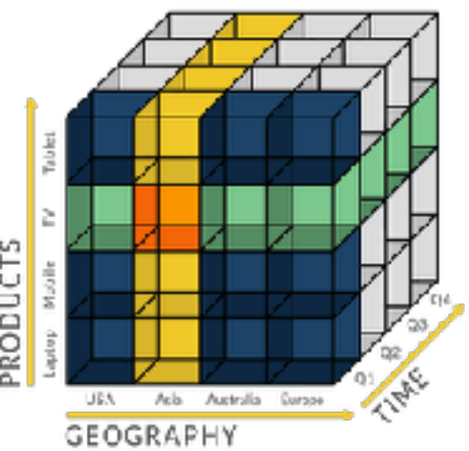
Semi Structured

Wide Column

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

Un Structured

- Binary storage



Time Series

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

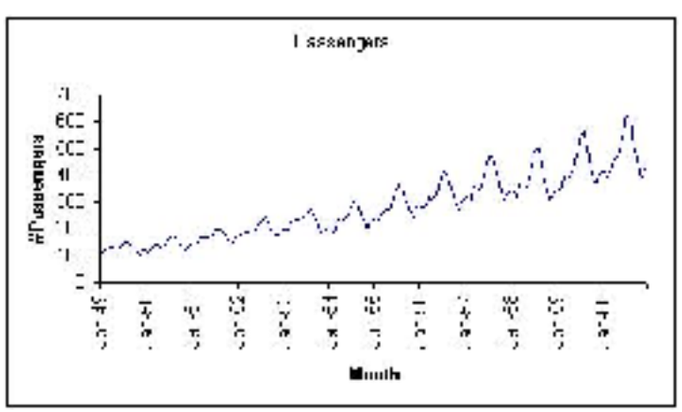
Audit Trial

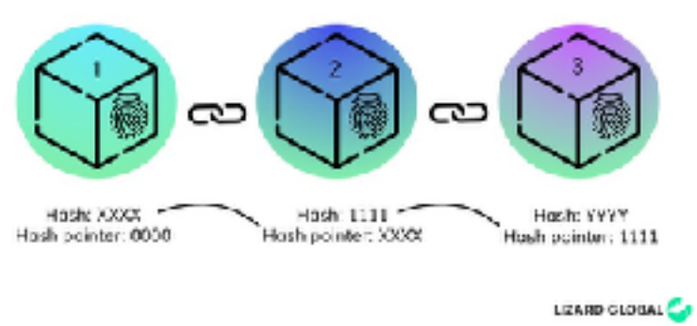
Immutable Logger

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

geo Spatial

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





Classic Relational Databases

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

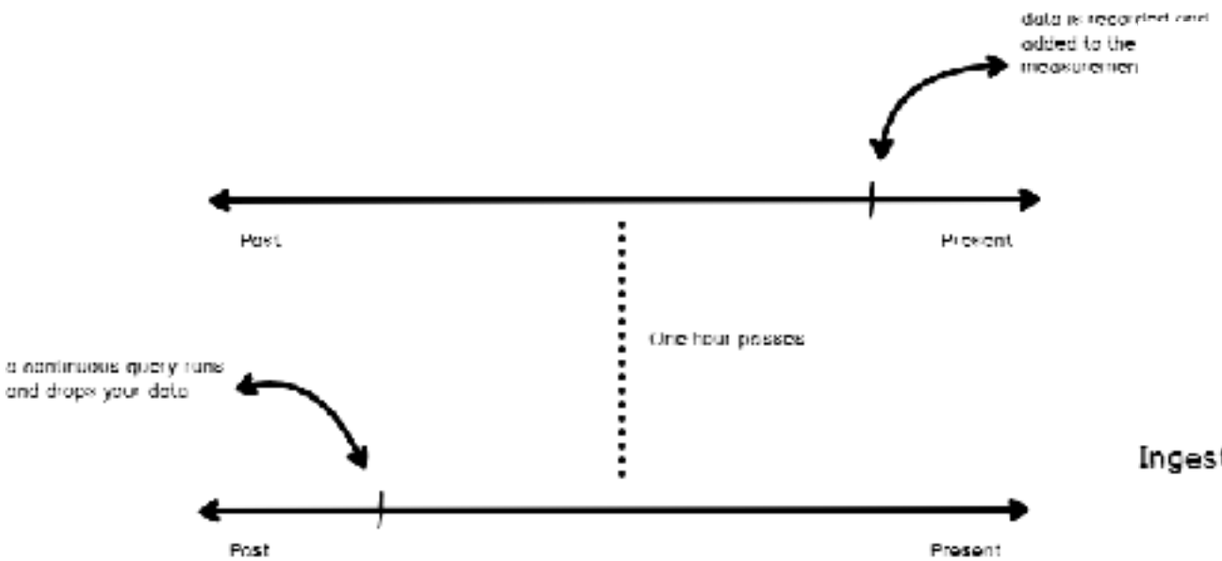
Data are multidimensional

Time Series Databases

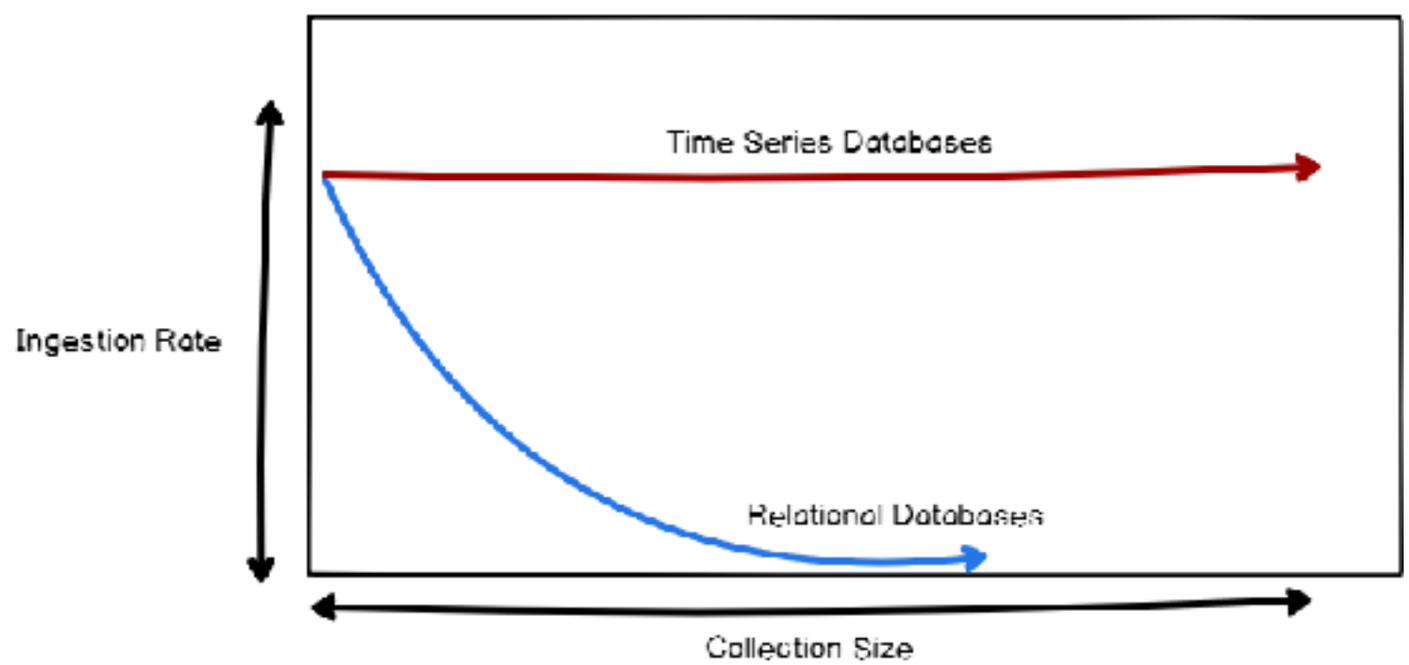
Sensor Temperature	Time
39.6	12/04/19 @ 14:12
41.2	12/04/19 @ 14:13
12.4	14/04/19 @ 12:15
18.5	16/04/19 @ 10:05

Data are aggregated over time

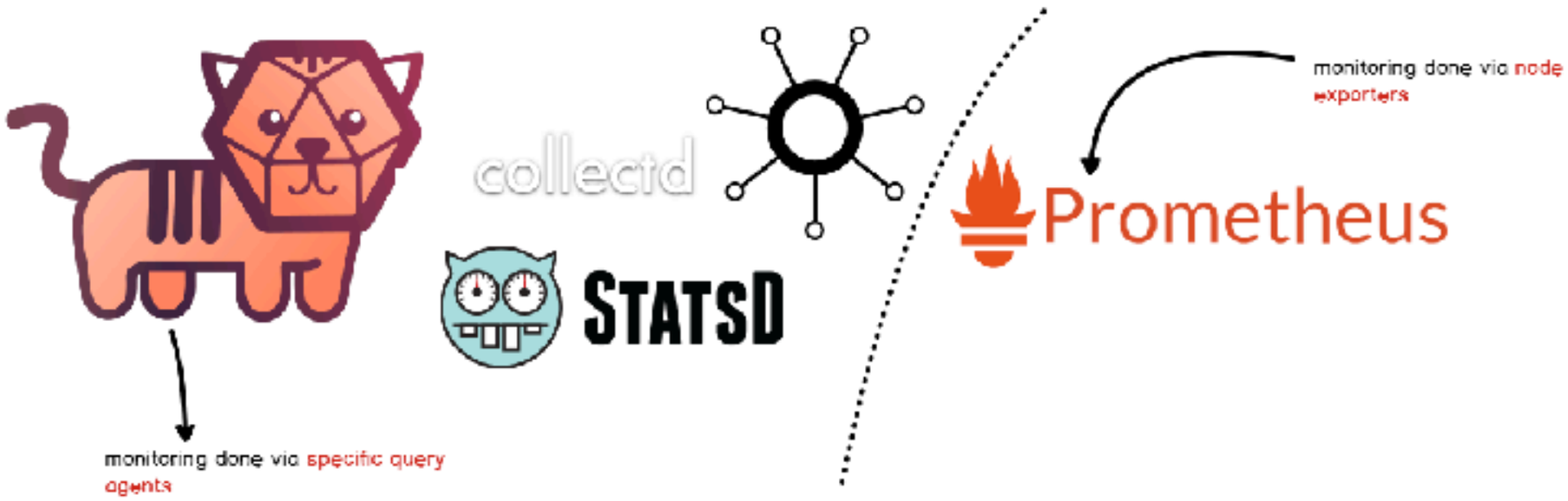
Case : retention policy = 1 hour



DBMS & TSDB Difference



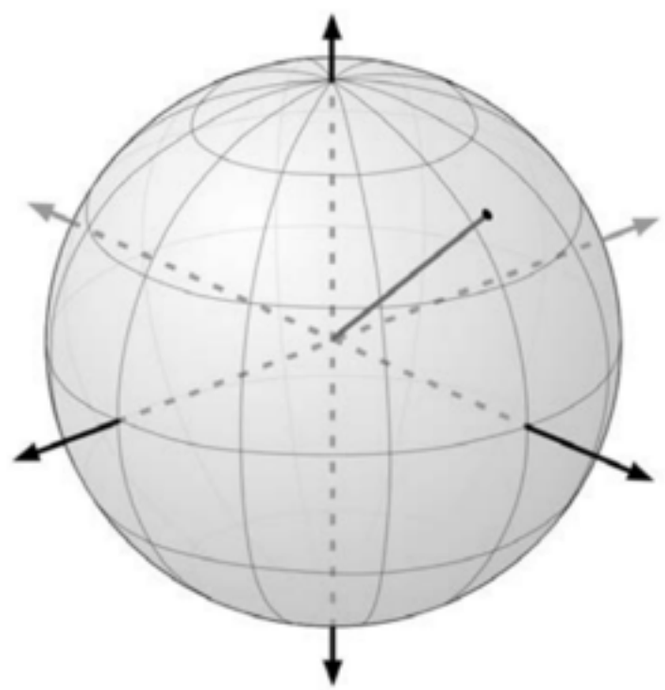
Tools that 'produce' TSDB data



Tools that 'consume' TSDB

*Non-exhaustive list

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



Row-oriented

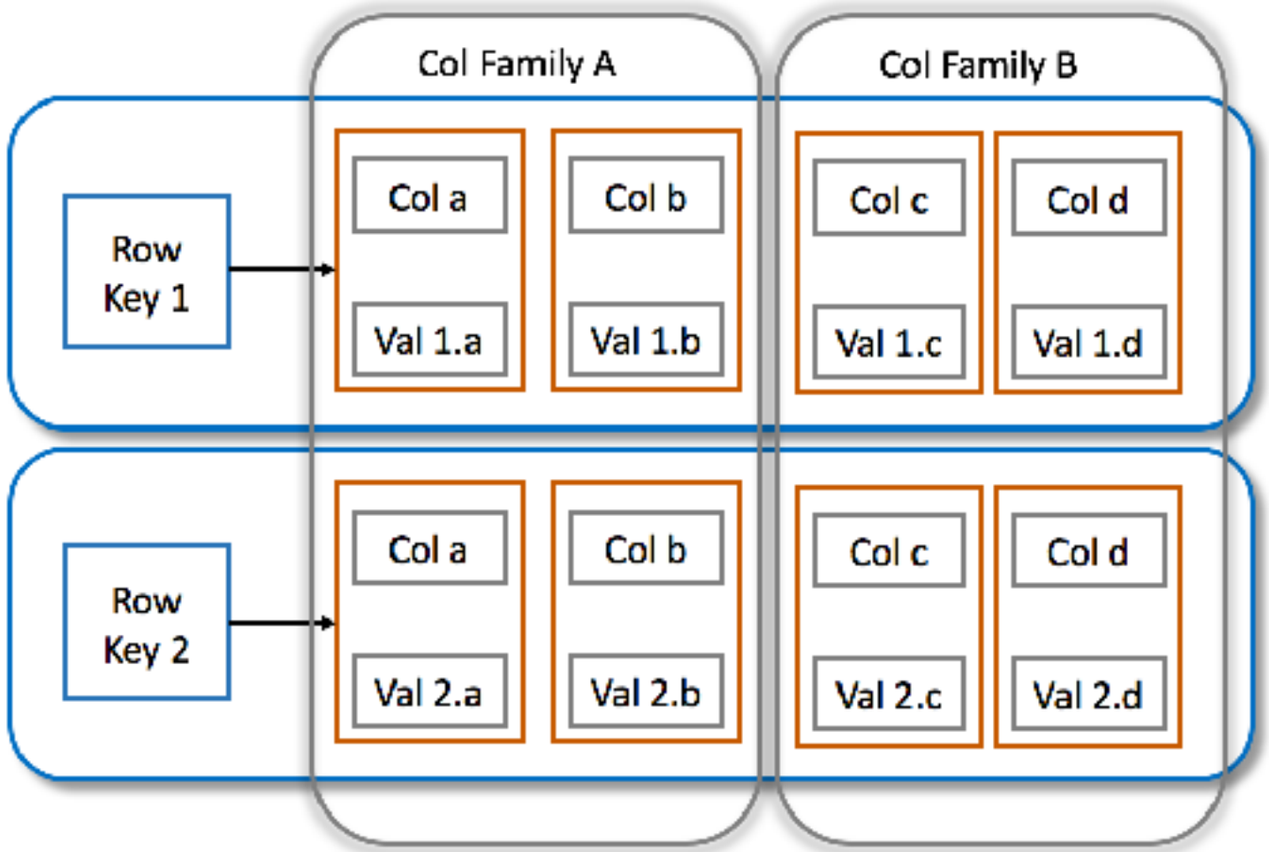
ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

Name	ID
John	001
Karen	002
Bill	003

Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003



RDBMS vs. Columnar: OLTP vs. OLAP

scgupta.link/datastores

OLTP

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

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

Logical Table

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

RDBMS: Row-Oriented Datastore

Columnar: Column-Oriented Datastore

OLAP

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

Encoded Chunk	Encoded Chunk	Encoded Chunk

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

© 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

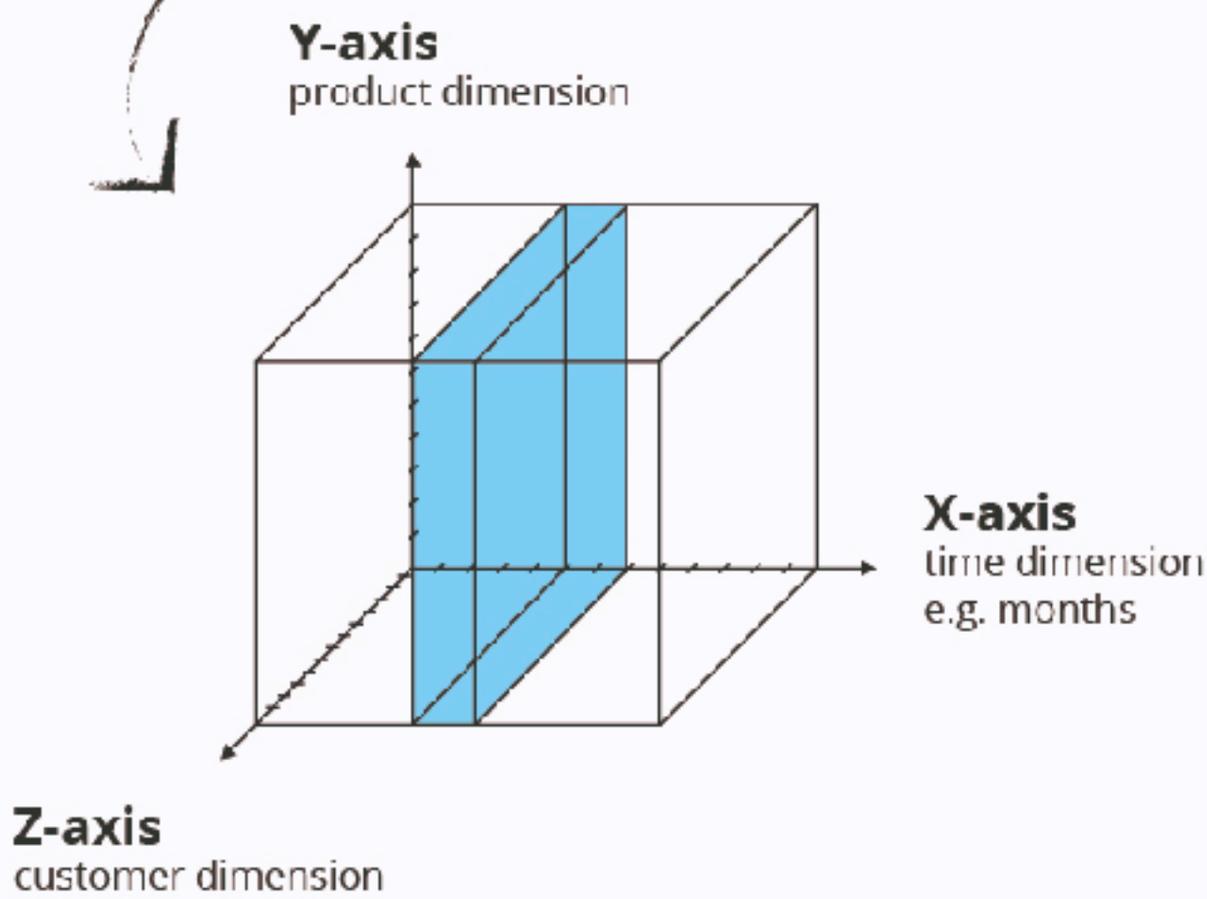
DATABASE
VERSUS
DATA WAREHOUSE

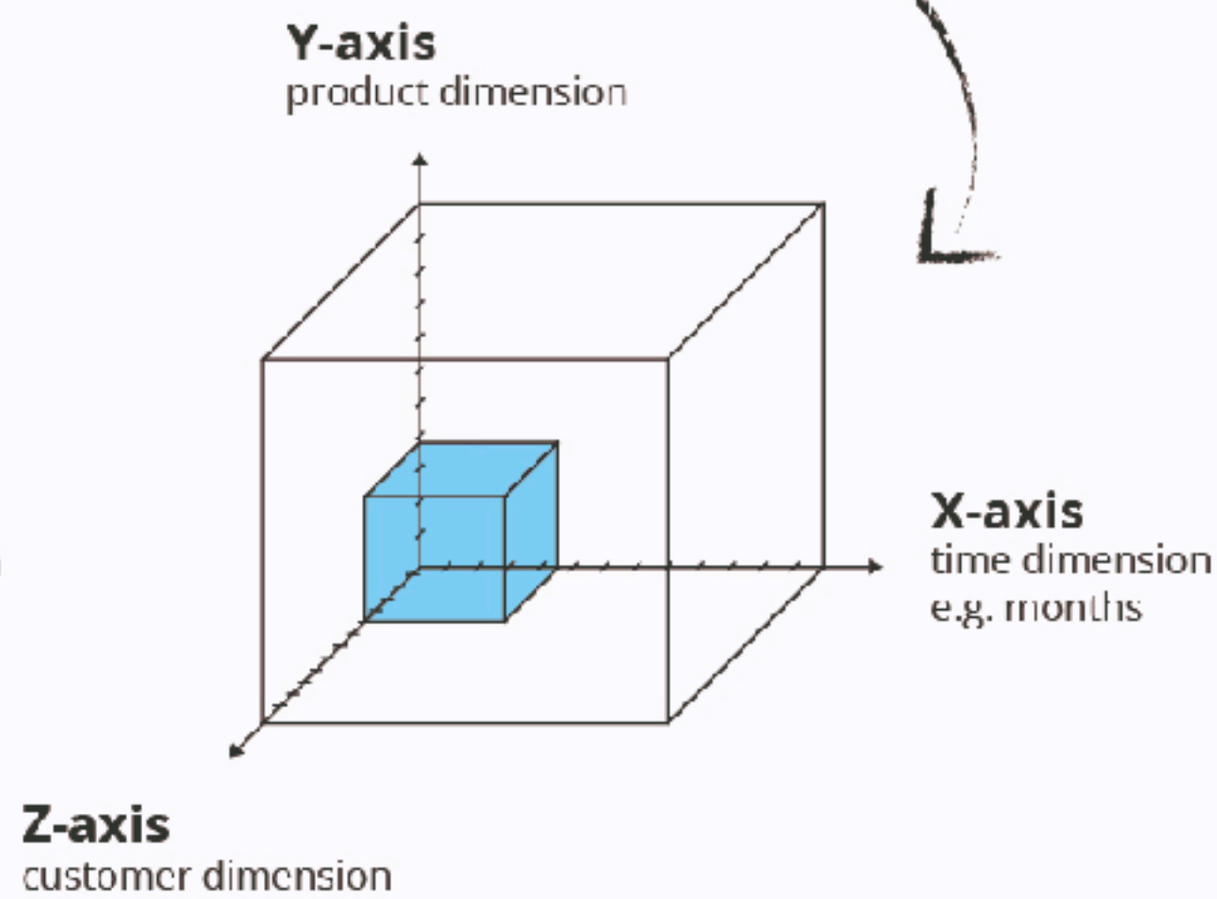
DATABASE	DATA WAREHOUSE
An organized collection of related data which stores data in a tabular format	A central location which stores consolidated data from multiple databases
Contains detailed data	Contains summarized data
Uses Online Transactional Processing (OLTP)	Uses Online Analytical Processing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

Visit www.PEDIAA.com

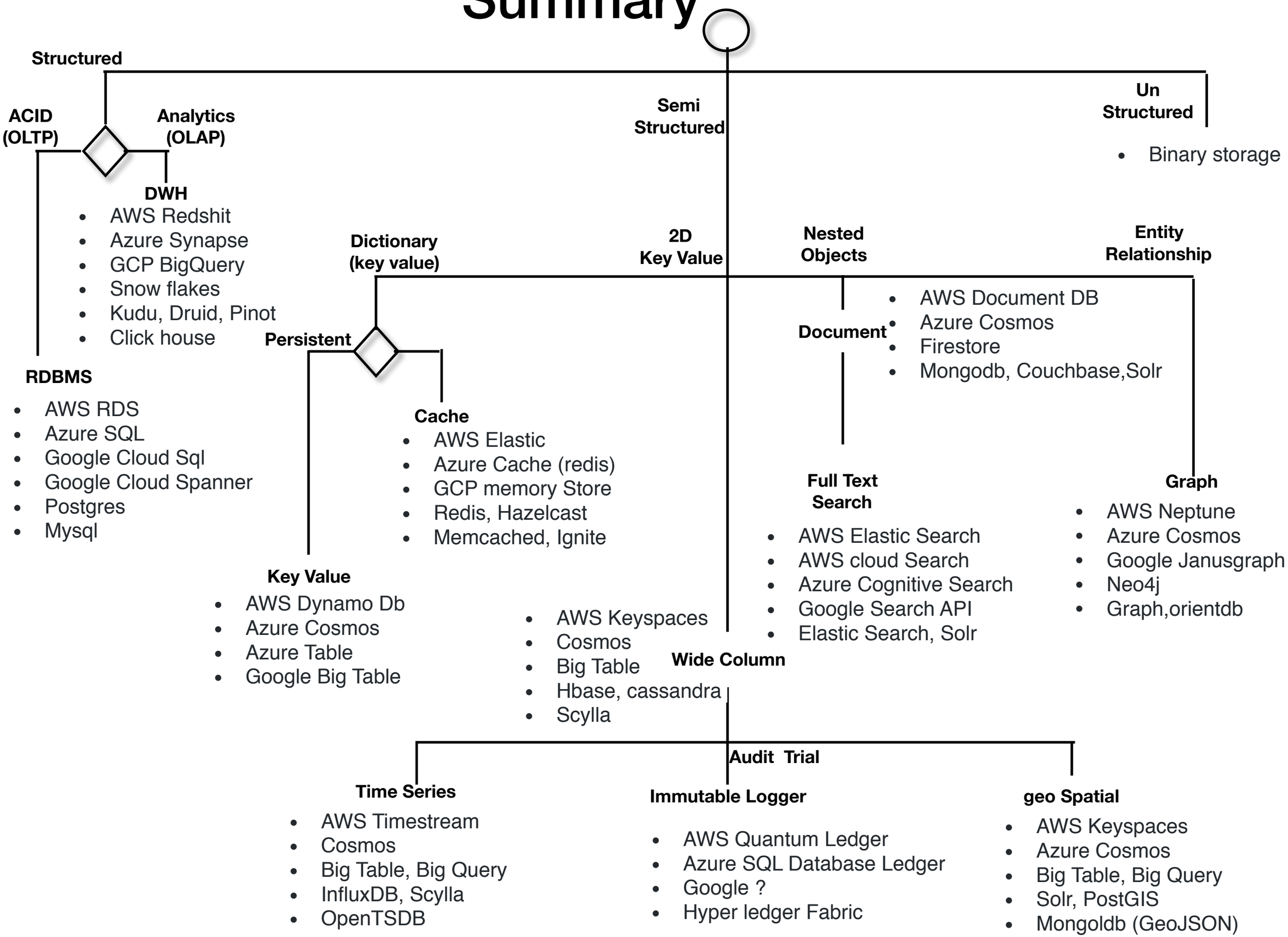
Slicing & Dicing

OLAP cube operations



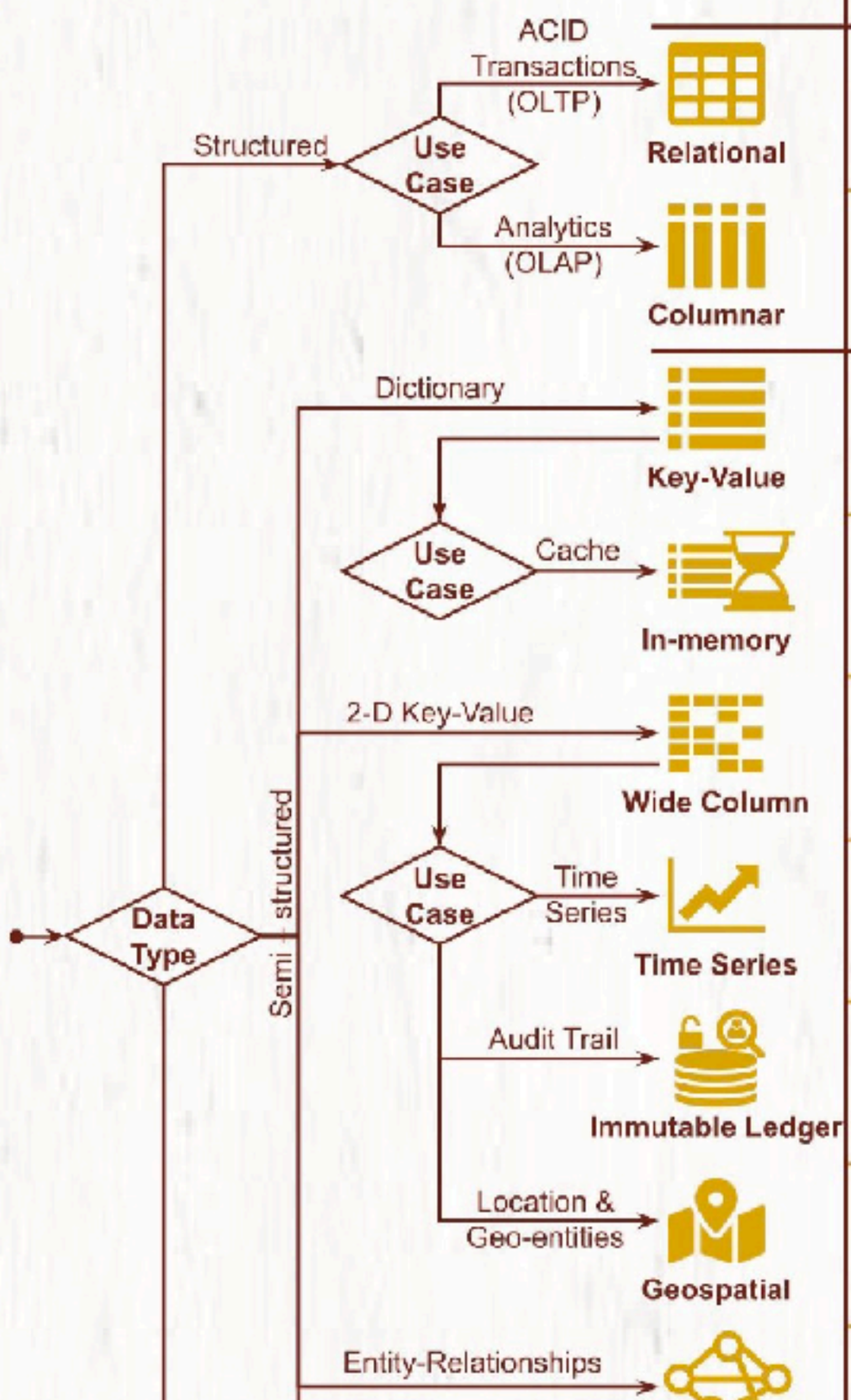


Summary


Structured
ACID (OLTP)
Analytics (OLAP)
DWH
AWS Redshit
Azure Synapse
GCP BigQuery
Snow flakes
Kudu, Druid, Pinot
Click house
RDBMS
AWS RDS
Azure SQL
Google Cloud Sql
Google Cloud Spanner
Postgres
Mysql
Semi Structured
Un Structured
Binary storage
Dictionary (key value)
2D Key Value
Nested Objects
Entity Relationship
Persistent
Cache
AWS Elastic
Azure Cache (redis)
GCP memory Store
Redis, Hazelcast
Memcached, Ignite
Key Value
AWS Dynamo Db
Azure Cosmos
Azure Table
Google Big Table
Document
AWS Document DB
Azure Cosmos
Firestore
Mongodb, Couchbase,Solr
Full Text Search
AWS Elastic Search
AWS cloud Search
Azure Cognitive Search
Google Search API
Elastic Search, Solr
Graph
AWS Neptune
Azure Cosmos
Google Janusgraph
Neo4j
Graph,orientdb
Wide Column
AWS Keyspaces
Cosmos
Big Table
Hbase, cassandra
Scylla
Audit Trial
Time Series
AWS Timestream
Cosmos
Big Table, Big Query
InfluxDB, Scylla
OpenTSDB
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)

SQL vs. NoSQL: Cheatsheet for AWS, Azure, and Google Cloud

scgupta.link/datastores



	aws	Azure	Google Cloud	Cloud Agnostic
Relational	RDS	Azure SQL Database	Cloud SQL, Cloud Spanner	SQL Server, Oracle, DB2, MySQL, PostgreSQL
Columnar	RedShift	Azure Synapse	BigQuery	Snowflake, ClickHouse, Druid, Kudu, Pinot
Key-Value	DynamoDB	Cosmos DB	BigTable	Redis, ScyllaDB, Ignite
In-memory	ElastiCache	Azure Cache for Redis	Memory-store	Redis, Memcached, Hazelcast, Ignite
Wide Column	Keyspaces	Cosmos DB	BigTable	HBase, Cassandra, ScyllaDB
Time Series	Timestream	Cosmos DB	BigTable, BigQuery	OpenTSDB, InfluxDB, ScyllaDB
Immutable Ledger	Quantum Ledger Database (QLDB)	Azure SQL Database Ledger	✗	Hyperledger Fabric
Geospatial	Keyspaces	Cosmos DB	BigTable, BigQuery	Solr, PostGIS, MongoDB (GeoJSON)
Entity-Relationships	Neptune	Cosmos DB	JanusGraph +	OrientDB, Neo4J,

NoSQL

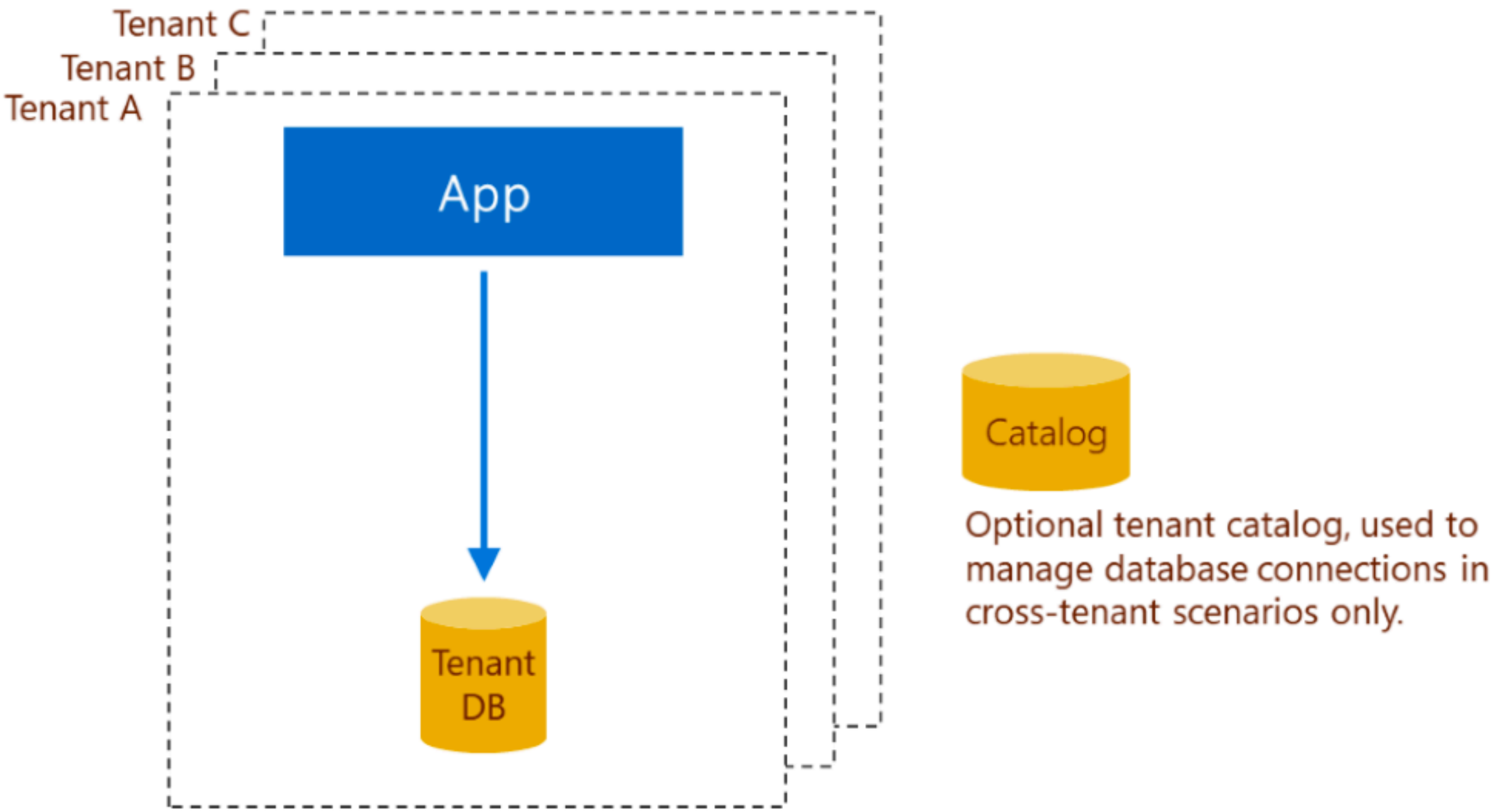
- Twitter uses Redis to deliver your Twitter timeline
- Pinterest uses Redis to store lists of users, followers, unfollowers, boards, and more
- Coinbase uses Redis to enforce rate limits and guarantee correctness of Bitcoin transactions
- Quora uses Memcached to cache results from slower, persistent databases
- 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
- 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

- 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

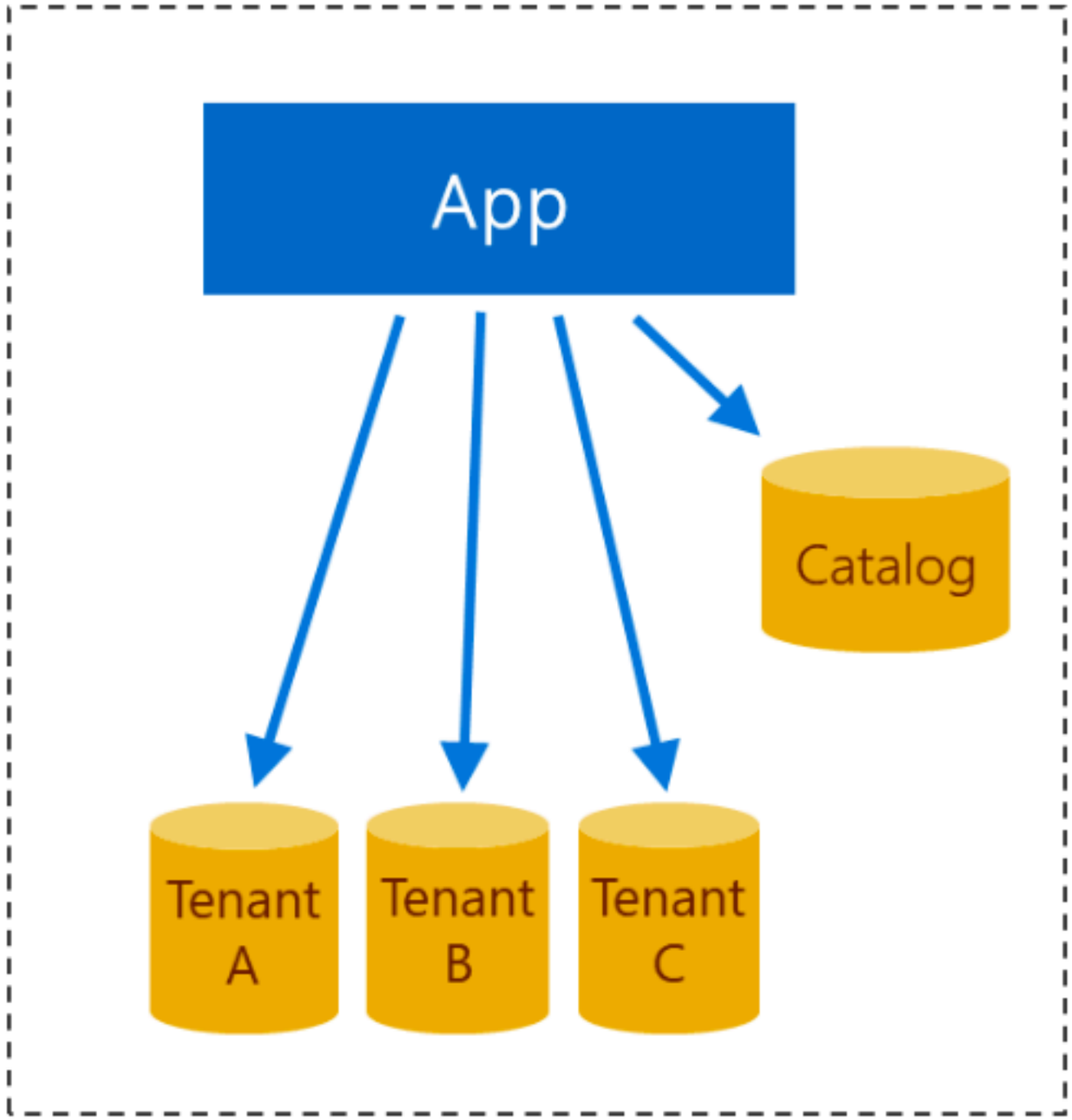
Multi Tenancy

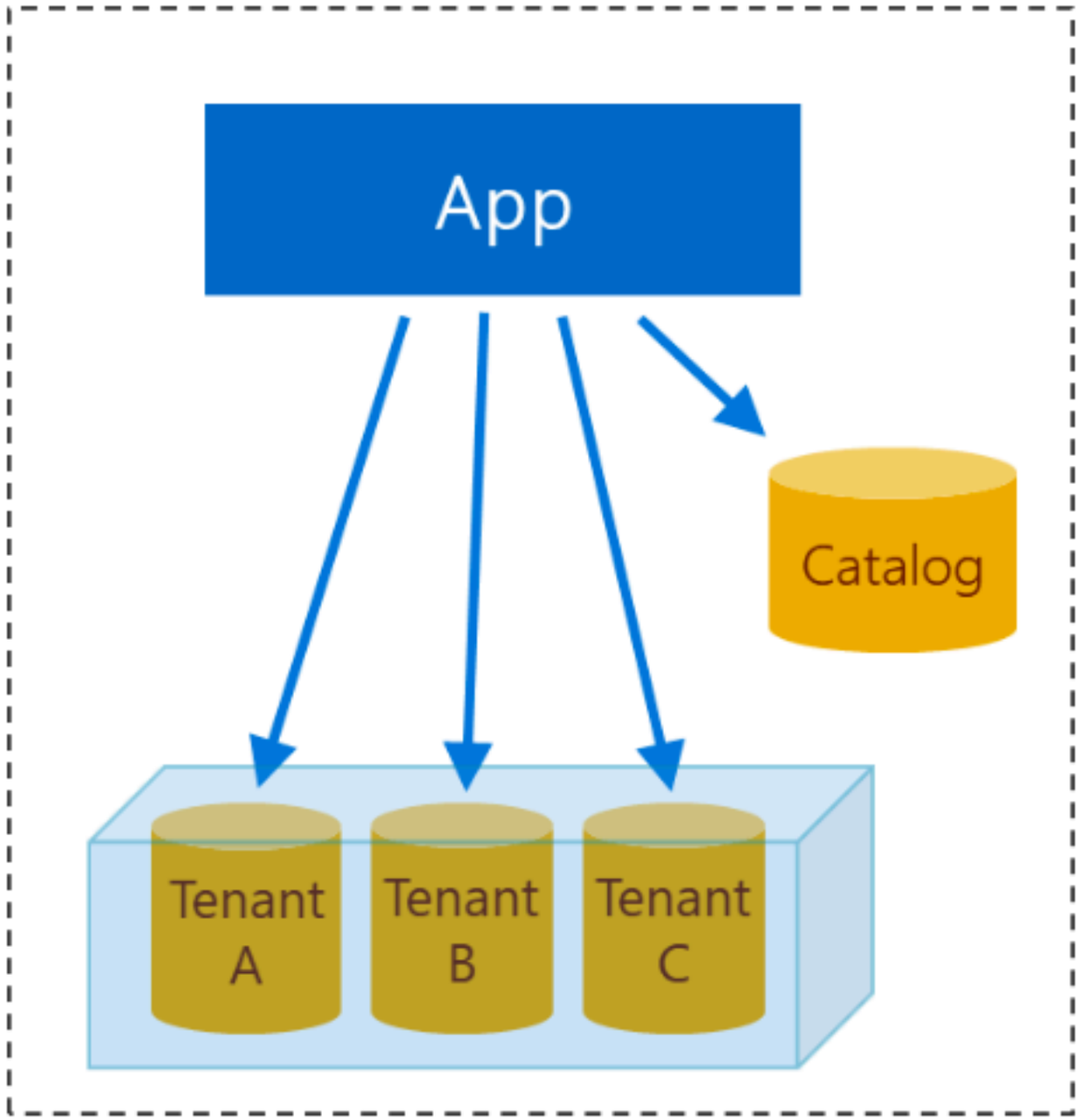
Measurement	Standalone app	Database-per-tenant	Sharded multi-tenant
Scale	Medium 1-100s	Very high 1-100,000s	Unlimited 1-1,000,000s
Tenant isolation	Very high	High	Low; except for any single tenant (that is alone in an MT db).
Database cost per tenant	High; is sized for peaks.	Low; pools used.	Lowest, for small tenants in MT DBs.
Performance monitoring and management	Per-tenant only	Aggregate + per-tenant	Aggregate; although is per-tenant only for singles.
Development complexity	Low	Low	Medium; due to sharding.
Operational complexity	Low-High. Individually simple, complex at scale.	Low-Medium. Patterns address complexity at scale.	Low-High. Individual tenant management is complex.

Standalone single-tenant app with single-tenant database



Multi-tenant app with database-per-tenant

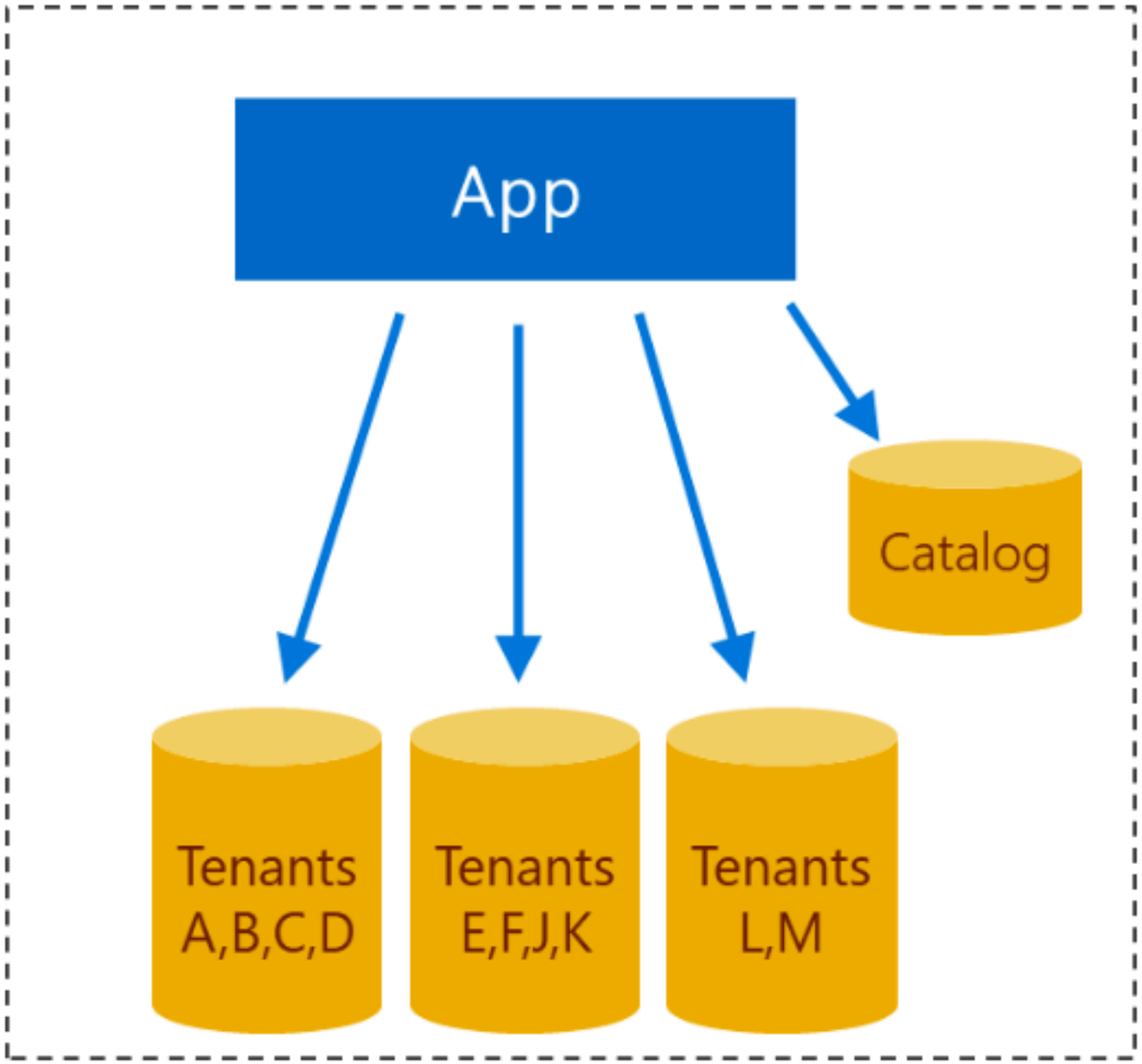




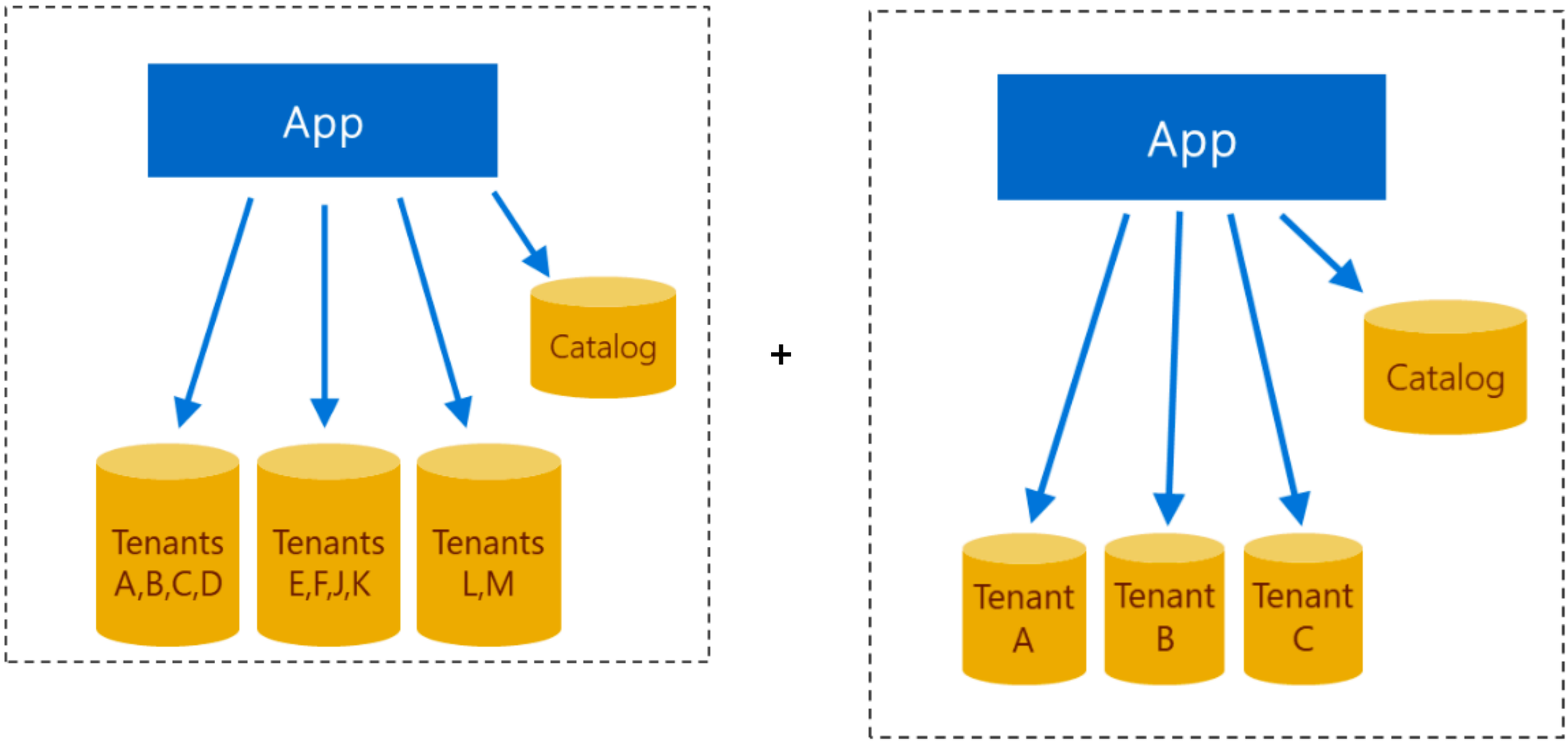
When databases are deployed in the same resource group, they can be grouped into elastic pools. The pools provide a cost-effective way of sharing resources across many databases. This pool option is cheaper than requiring each database to be large enough to accommodate the usage peaks that it experiences.

Multi-tenant app with single multi-tenant databases

Multi-tenant app with single sharded multi-tenant databases



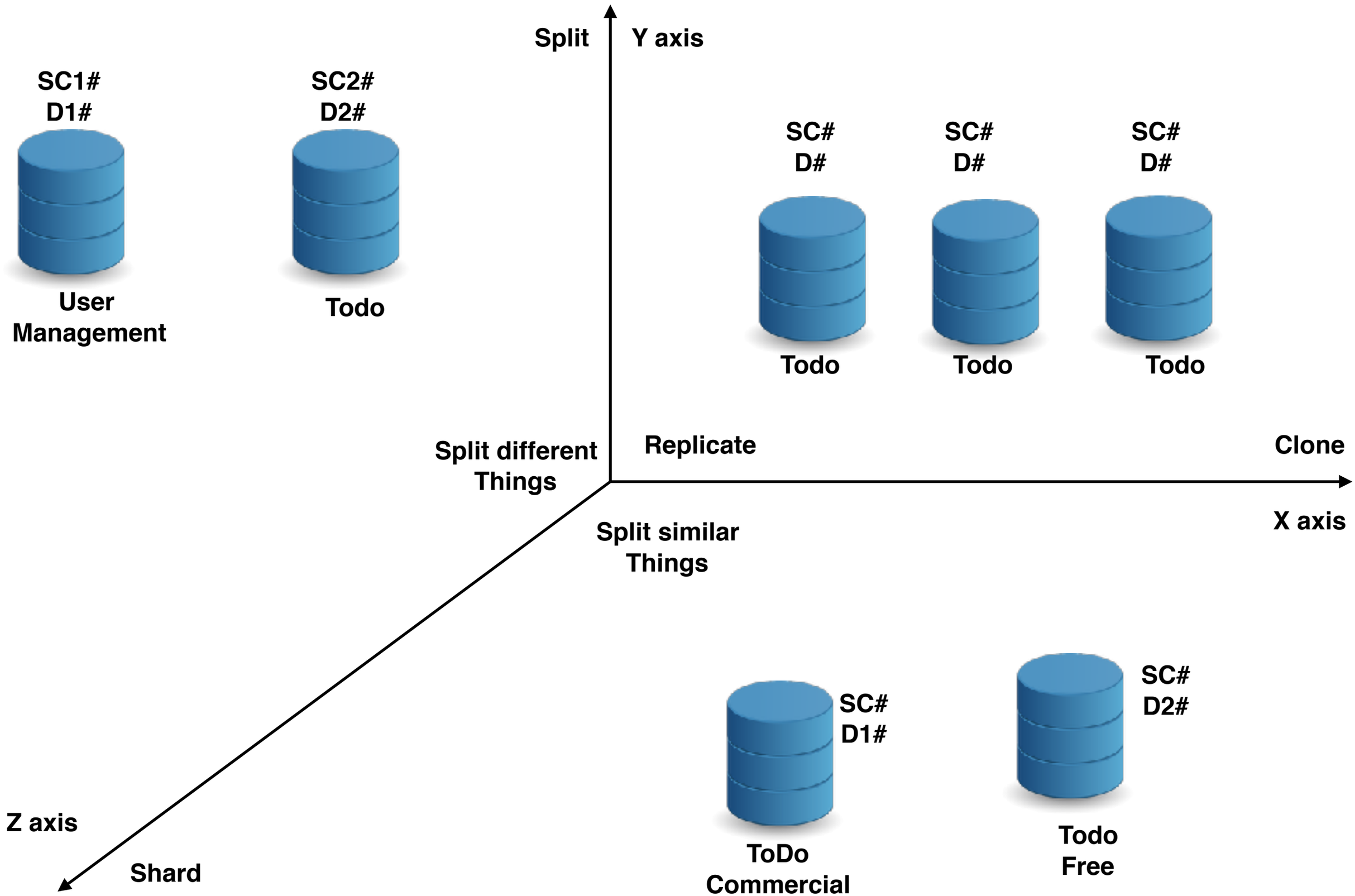
Hybrid sharded multi-tenant database model



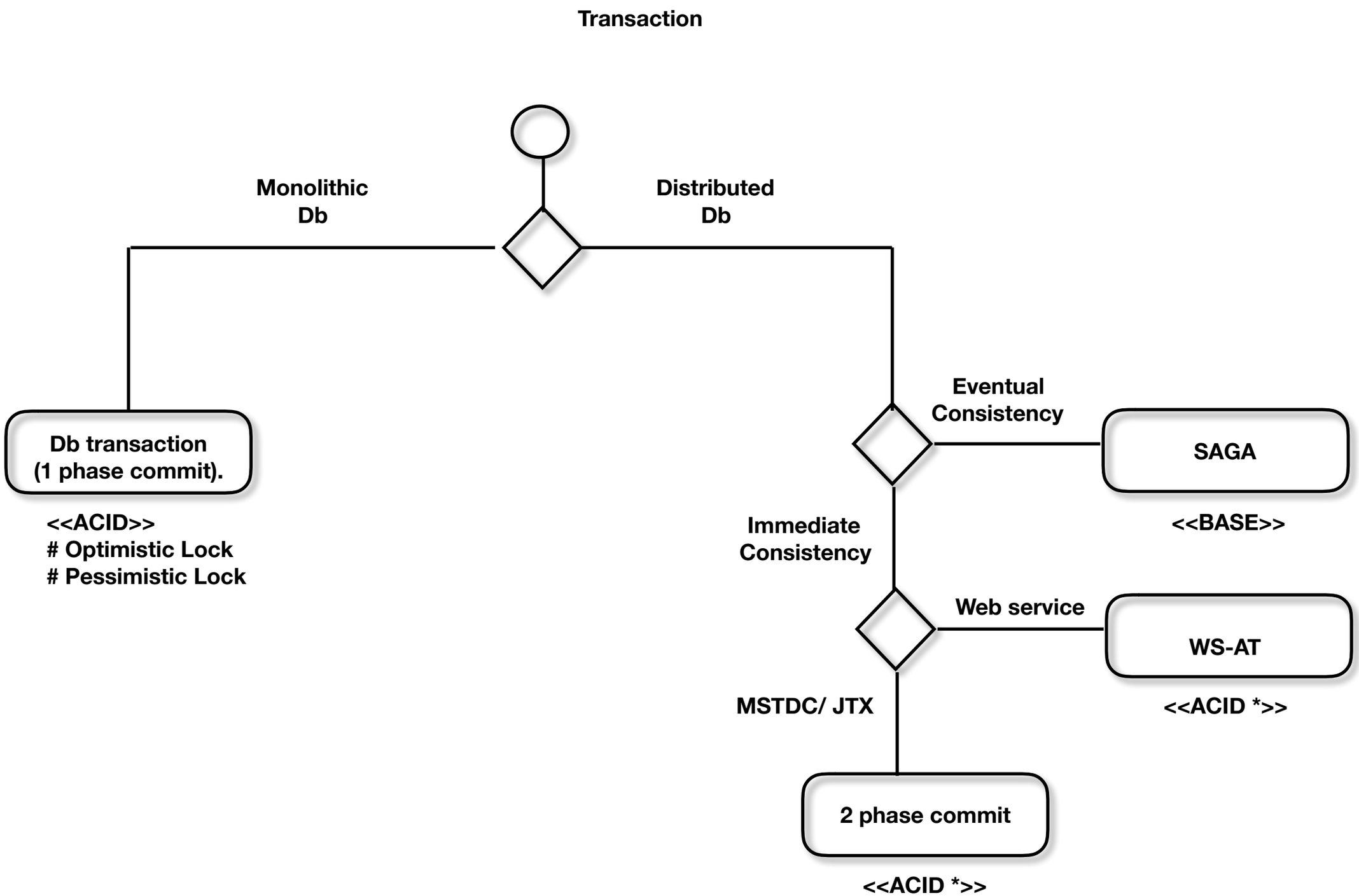
Size the storage

Use Case	**Maximum Response Time**	Start of Window	End Of Window	Transactions/ Hour	List of **Database Activity (CRUD)**	Size of Storage Per transaction

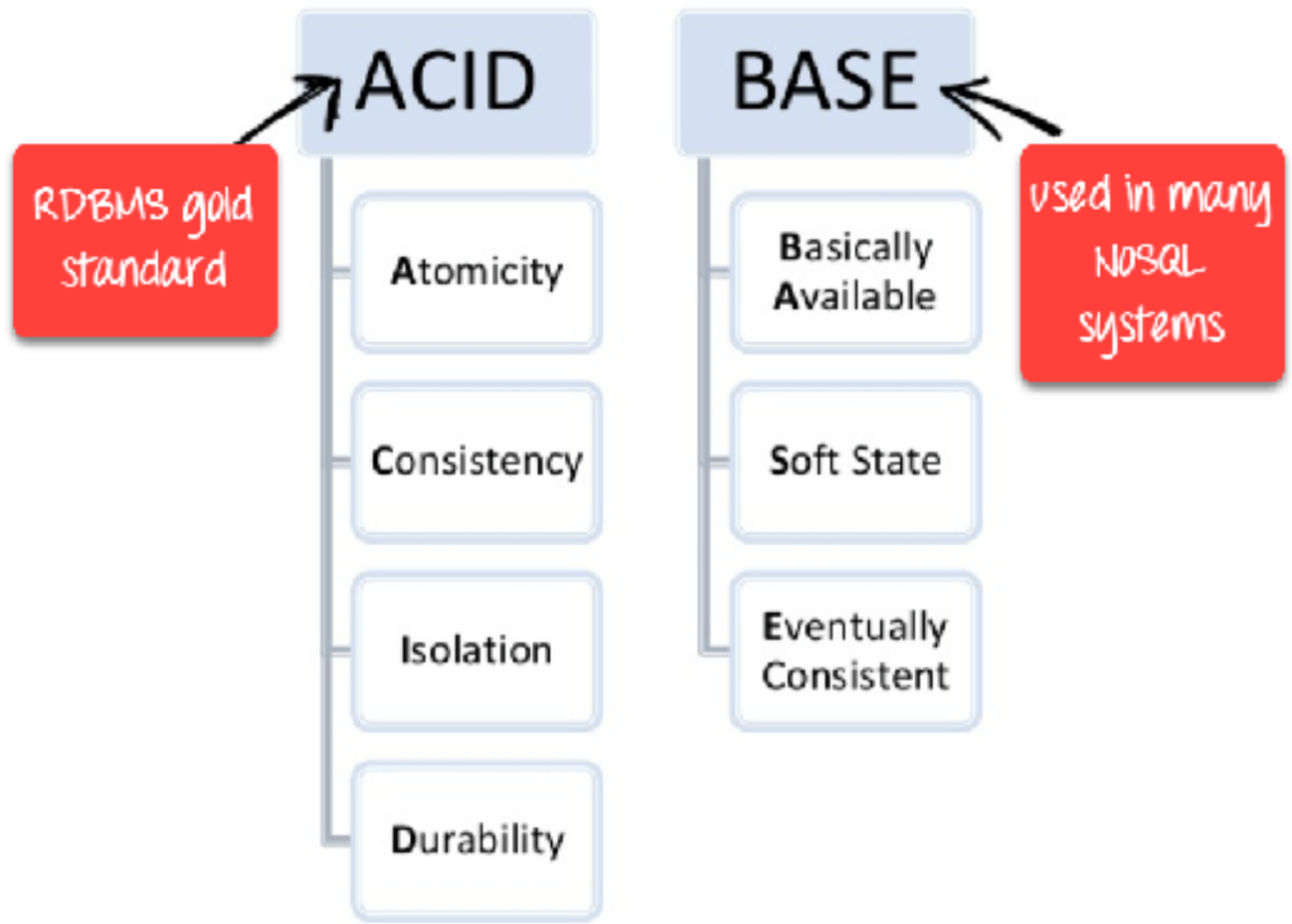
Scalability Cube - 50 rules for high Scalability



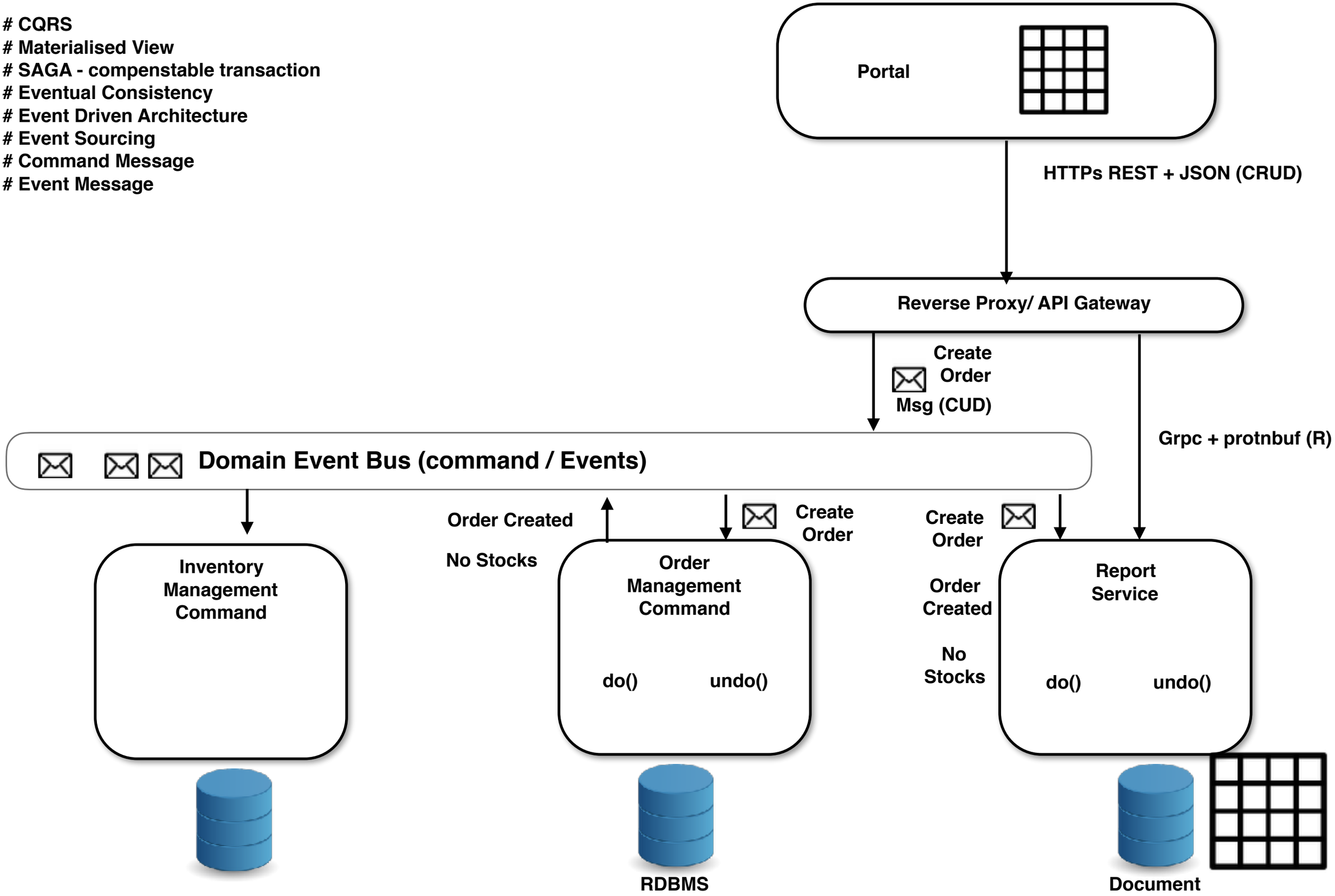
Transaction and Concurrency



* 2PC **is not resilient to all possible failure configurations**.


ACID
RDBMS gold standard
Atomicity
Consistency
Isolation
Durability
BASE
used in many NOSQL systems
Basically Available
Soft State
Eventually Consistent

CQRS
Materialised View
SAGA - compenstable transaction
Eventual Consistency
Event Driven Architecture
Event Sourcing
Command Message
Event Message



3NF
no duplicates, more joins
Write Friendly
not read friendly
current state

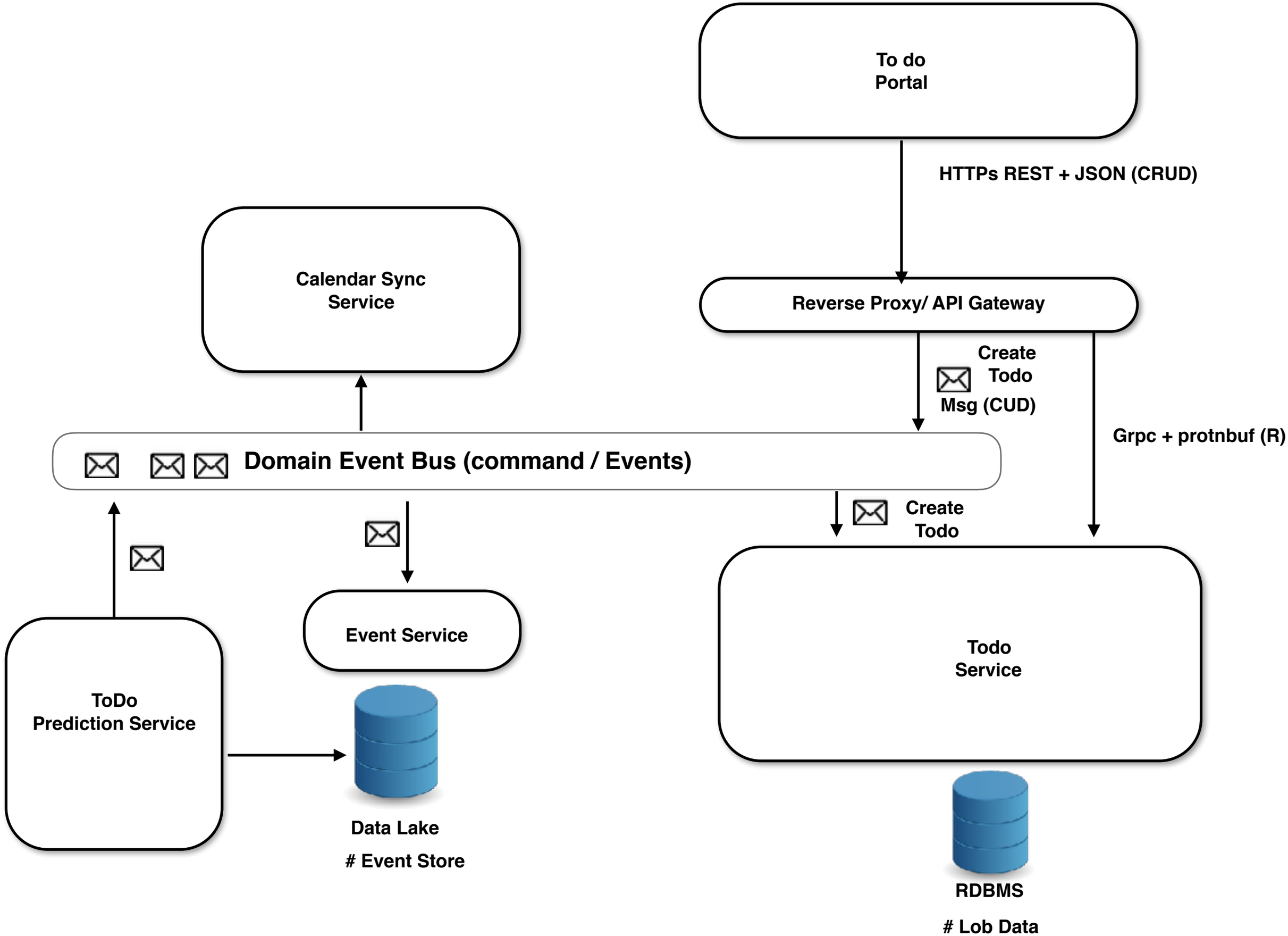
DNF
duplicates, no joins
not Write Friendly
read friendly

All transactions are encapsulated within a call from the BusinessLayer to a stored procedure. Any SQL errors within a stored procedure force a rollback and the error is raised to the BusinessLayer. Account data integrity is maintained through optimistic concurrency. A unique version timestamp is returned with each *head* Account object. During an update, the saveAccount stored procedure verifies the timestamp of the modified account is the same as the one in the database. If not, another user had previously updated the Account, so the update is rejected. The user is prompted to refresh a current version of the account before making edits.

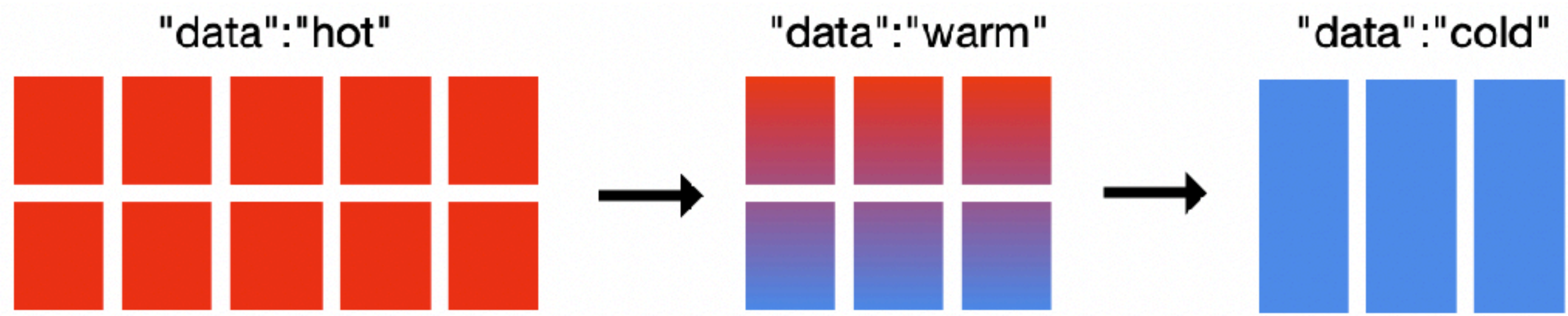
```
    if @version <> (select version from Account where account_id =
@AccountPK)
    begin
      rollback transaction
      raiserror (60001, 11, 1, 'Account')
    end
```

Add Metadata

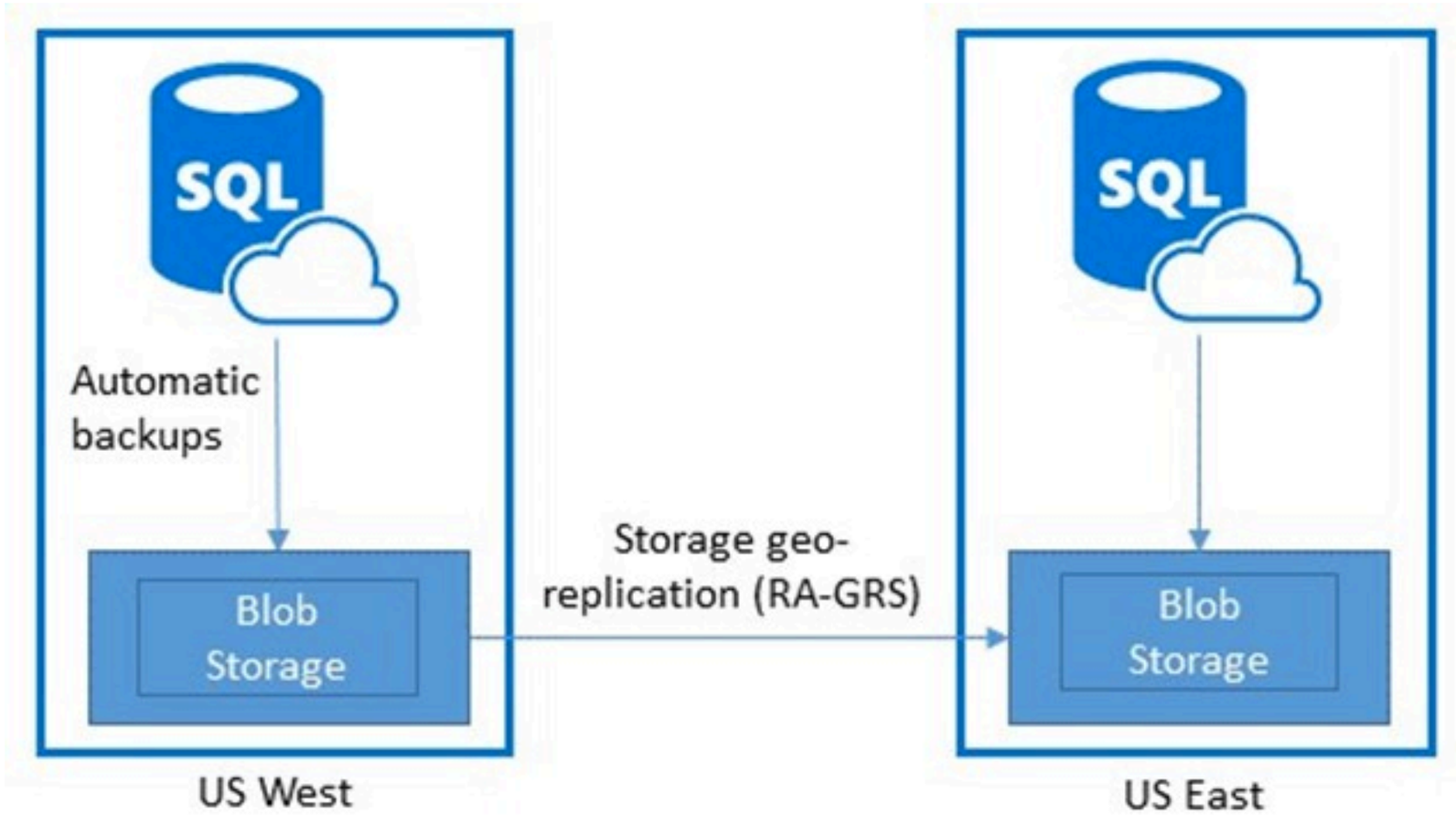
Data Language
Data Timezone
Data Version
Tenant
Created By
Created Date
Last Modified Date
…


To do
Portal
HTTPs REST + JSON (CRUD)
Calendar Sync
Service
Reverse Proxy/ API Gateway
Create
Todo
Msg (CUD)
Grpc + protnbuf (R)
Domain Event Bus (command / Events)
Create
Todo
Event Service
Todo
Service
ToDo
Prediction Service
Data Lake
Event Store
RDBMS
Lob Data

Archive



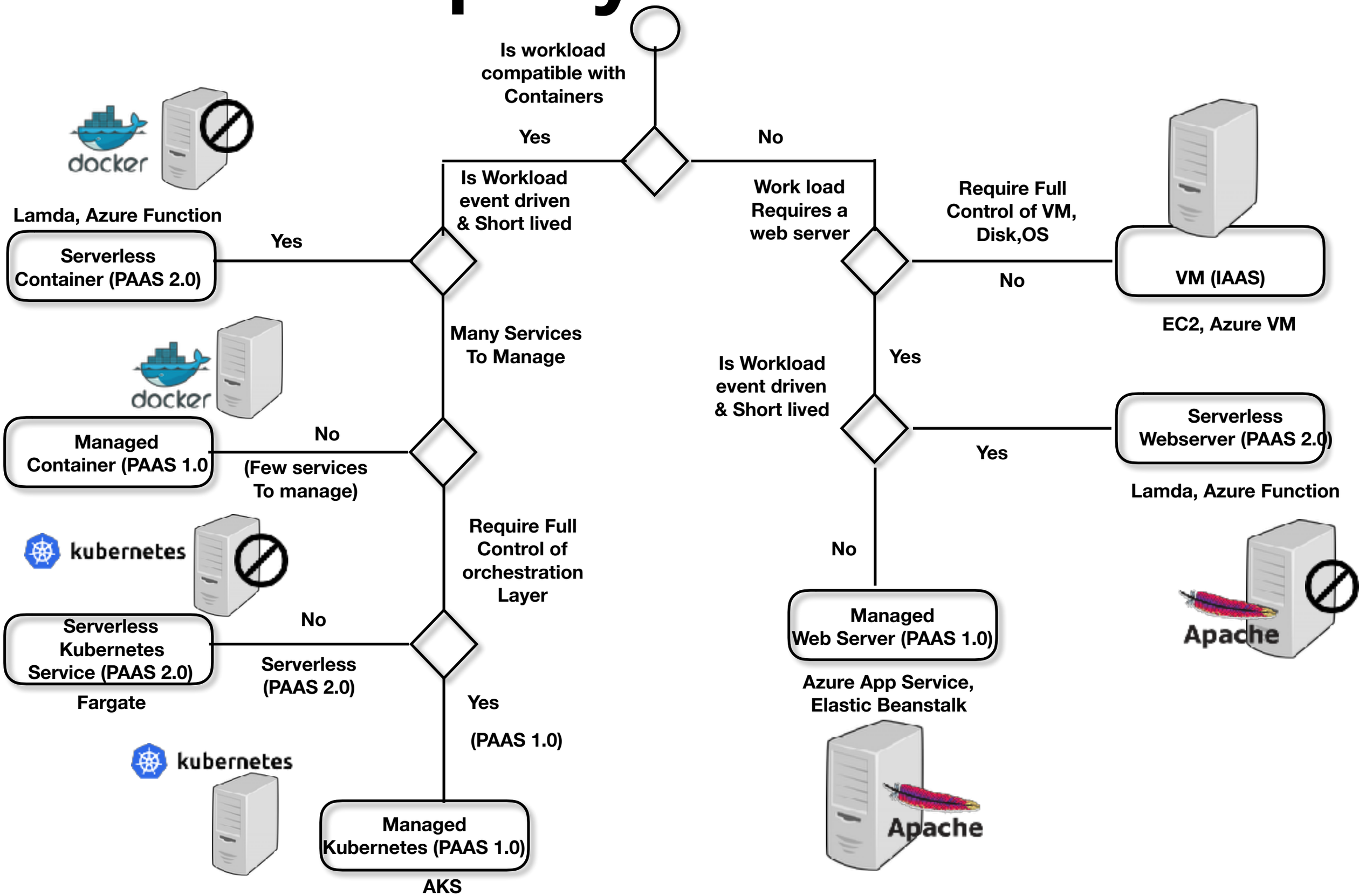
Backup and Recovery



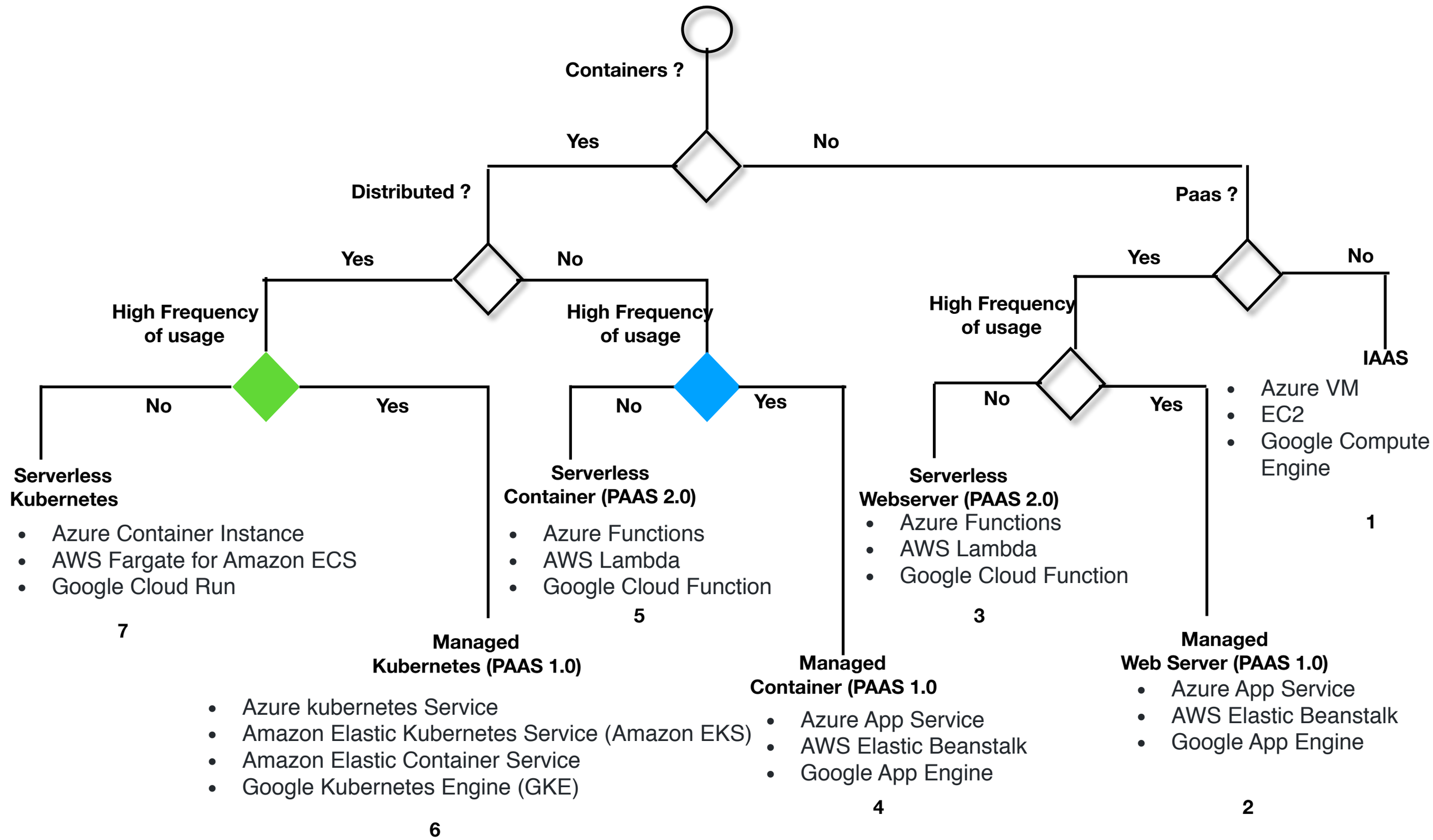
* Full backups
* differential backups
* transaction log backups

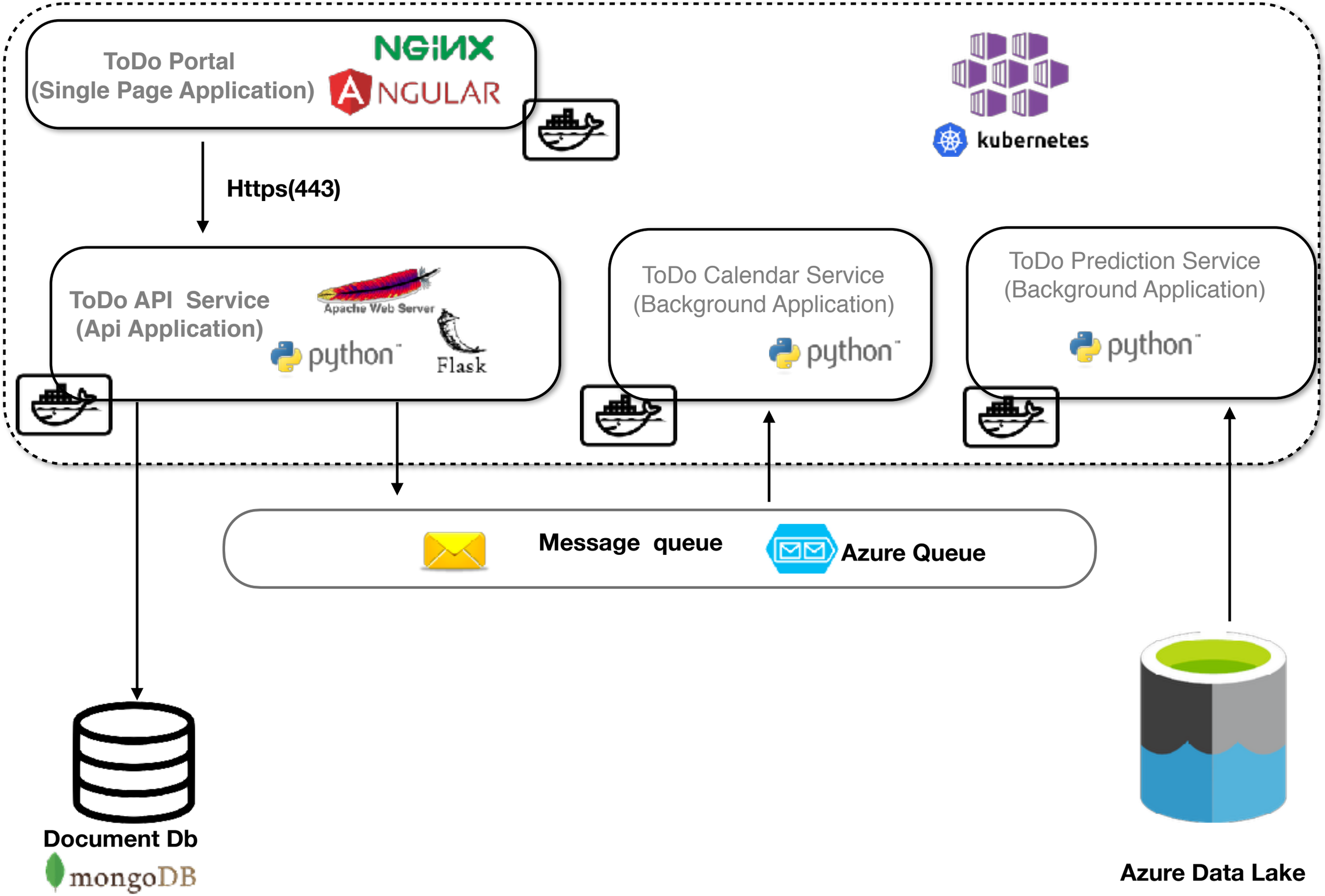
* Standard Geo-Replication
* Active Geo-Replication

4. Deployment View

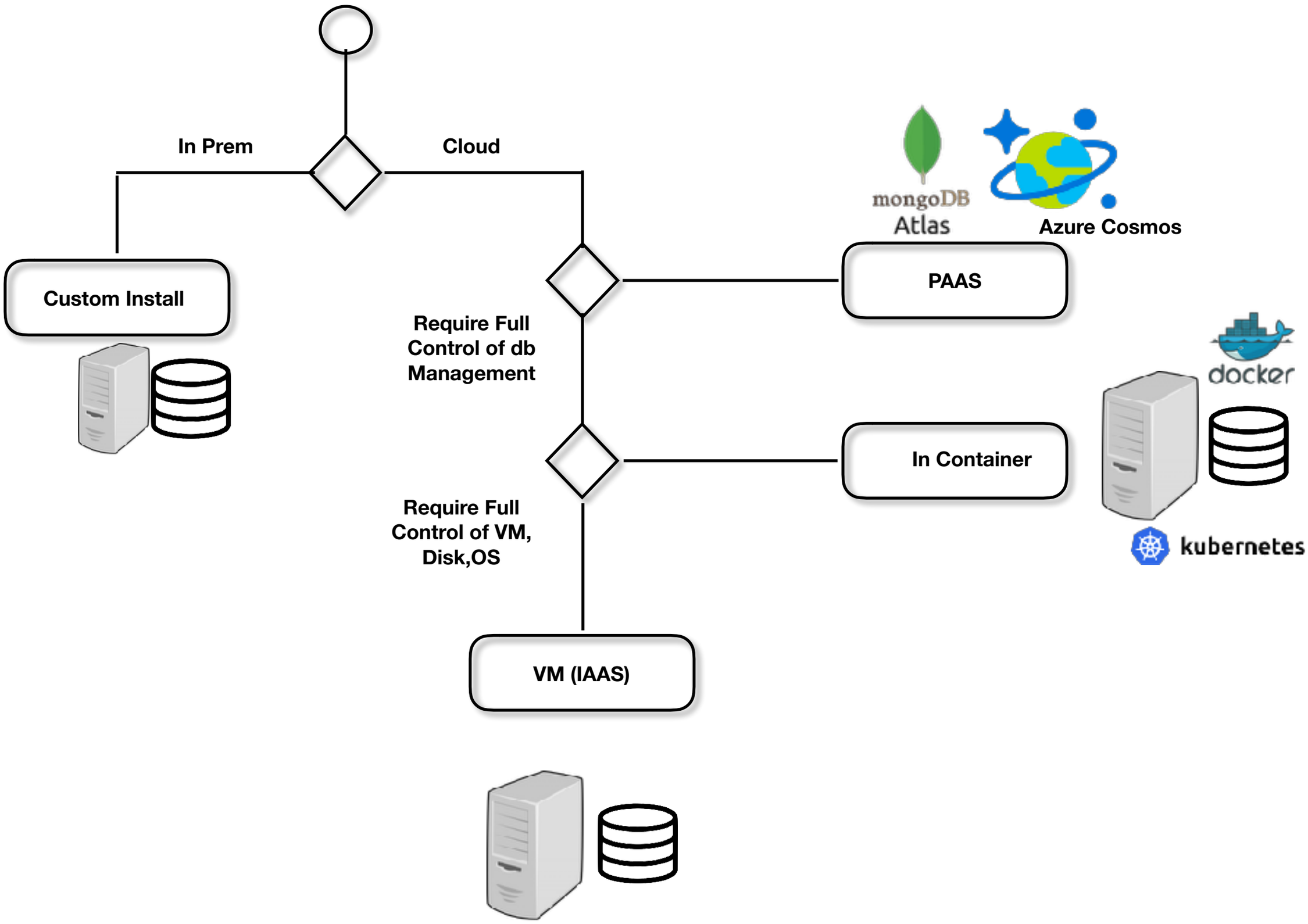


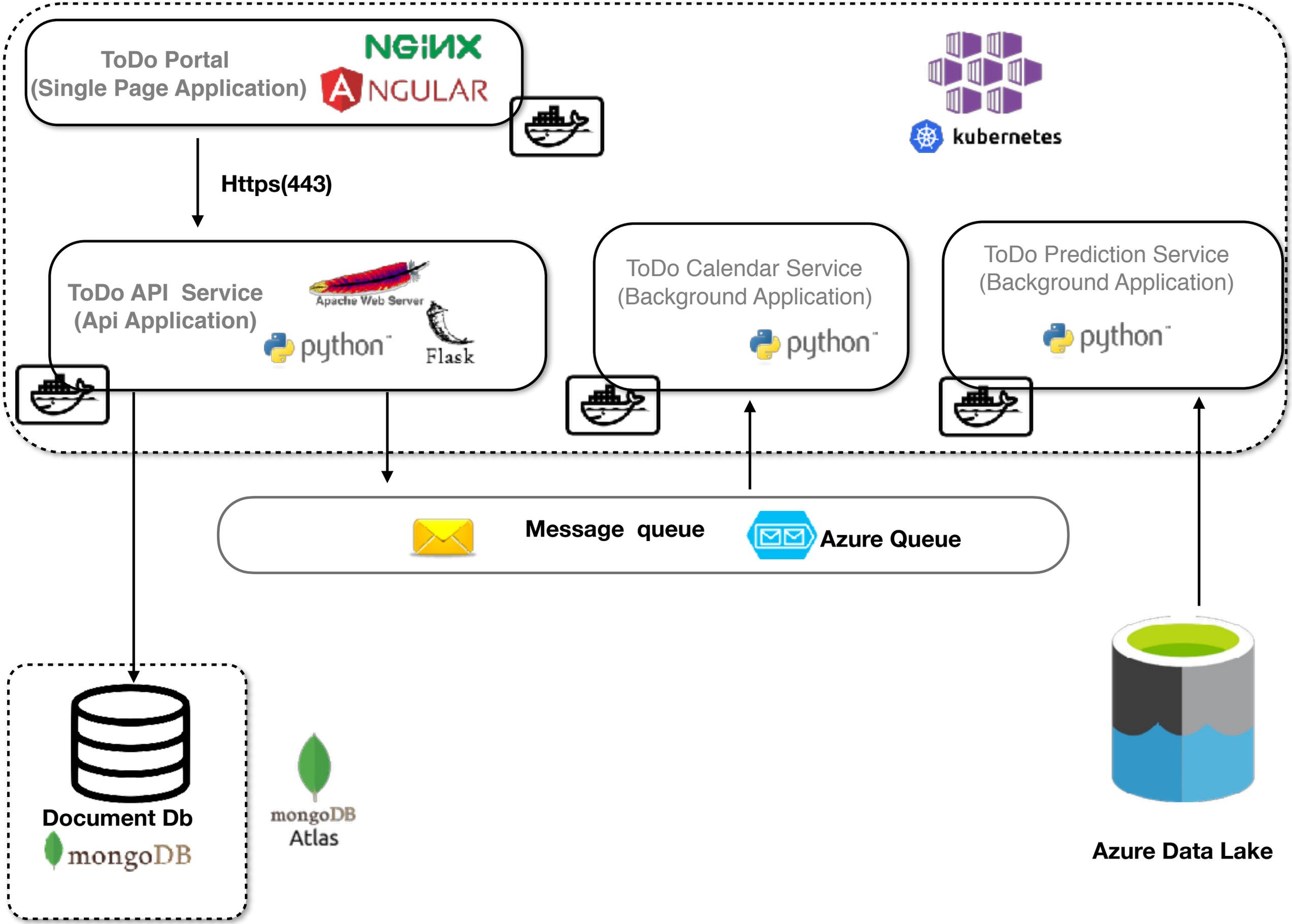
Deployment View




ToDo Portal
(Single Page Application)
NGINX
ANGULAR
kubernetes
Https(443)
ToDo API Service
(Api Application)
Apache Web Server
python
Flask
ToDo Calendar Service
(Background Application)
python
ToDo Prediction Service
(Background Application)
python
Message queue
Azure Queue
Document Db
mongoDB
Azure Data Lake

Deploying Data Tier




ToDo Portal
(Single Page Application)
NGINX
ANGULAR
kubernetes
Https(443)
ToDo API Service
(Api Application)
Apache Web Server
python
Flask
ToDo Calendar Service
(Background Application)
python
ToDo Prediction Service
(Background Application)
python
Message queue
Azure Queue
Document Db
mongoDB
mongoDB
Atlas
Azure Data Lake

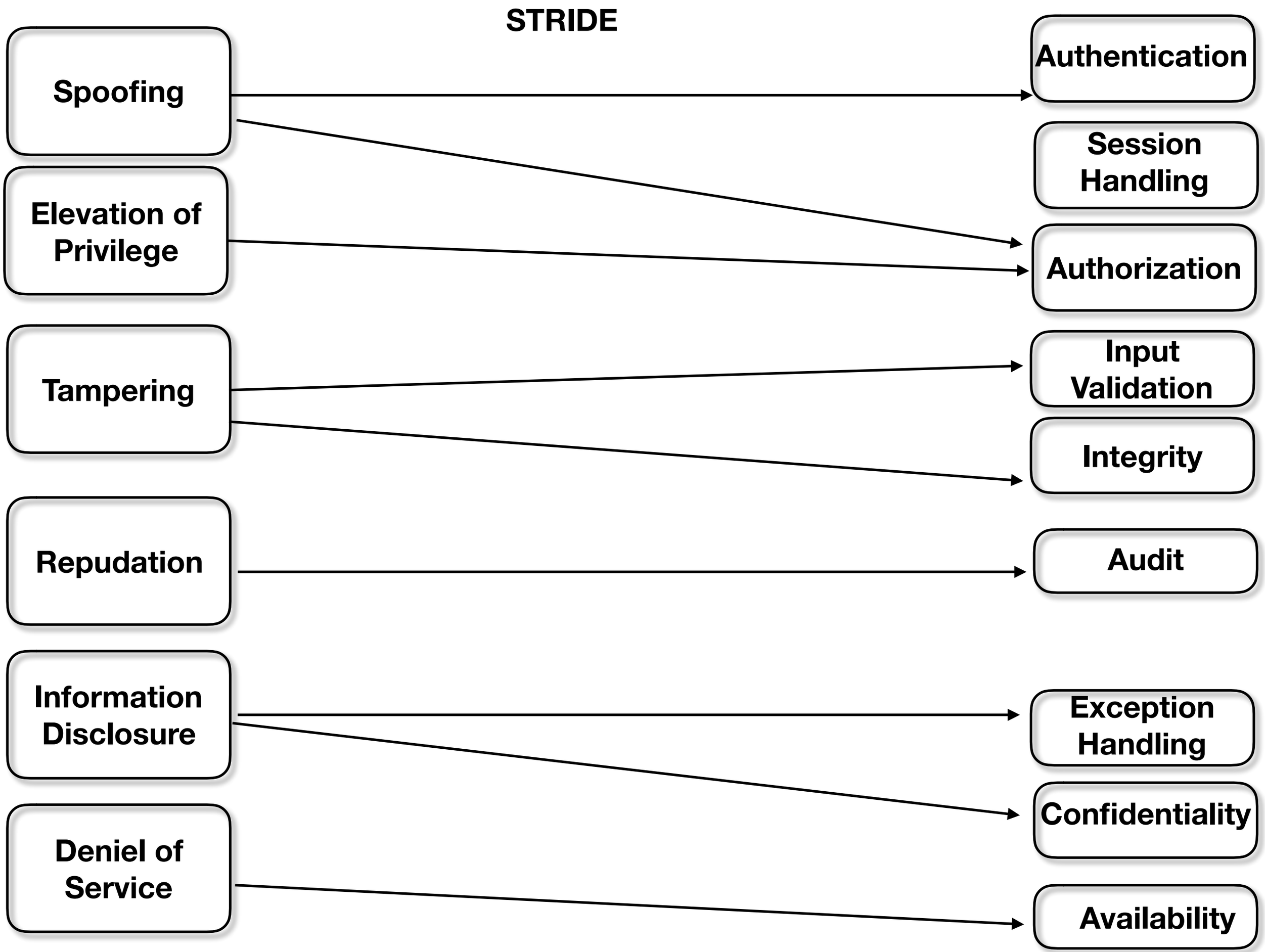
5. Application Security View

Threat Modeling

Threat Modeling Method	Features
STRIDE	• Helps identify relevant mitigating techniques • Is the most mature • Is easy to use but is time consuming
PASTA	• Helps identify relevant mitigating techniques • Directly contributes to risk management • Encourages collaboration among stakeholders • Contains built-in prioritization of threat mitigation • Is laborious but has rich documentation
LINDDUN	• Helps identify relevant mitigation techniques • Contains built-in prioritization of threat mitigation • Can be labor intensive and time consuming
CVSS	• Contains built-in prioritization of threat mitigation • Has consistent results when repeated • Has automated components • Has score calculations that are not transparent
Attack Trees	• Helps identify relevant mitigation techniques • Has consistent results when repeated • Is easy to use if you already have a thorough understanding of the system
Persona non Grata	• Helps identify relevant mitigation techniques • Directly contributes to risk management • Has consistent results when repeated • Tends to detect only some subsets of threats
Security Cards	• Encourages collaboration among stakeholders • Targets out-of-the-ordinary threats • Leads to many false positives
hTMM	• Contains built-in prioritization of threat mitigation • Encourages collaboration among stakeholders • Has consistent results when repeated
Quantitative TMM	• Contains built-in prioritization of threat mitigation • Has automated components • Has consistent results when repeated
Trike	• Helps identify relevant mitigation techniques • Directly contributes to risk management • Contains built-in prioritization of threat mitigation • Encourages collaboration among stakeholders • Has automated components • Has vague, insufficient documentation
VAST Modeling	• Helps identify relevant mitigation techniques • Directly contributes to risk management • Contains built-in prioritization of threat mitigation • Encourages collaboration among stakeholders • Has consistent results when repeated • Has automated components • Is explicitly designed to be scalable • Has little publicly available documentation
OCTAVE	• Helps identify relevant mitigation techniques • Directly contributes to risk management • Contains built-in prioritization of threat mitigation • Encourages collaboration among stakeholders • Has consistent results when repeated • Is explicitly designed to be scalable • Is time consuming and has vague documentation

PASTA

STAGE I - Definition of the Objectives (DO)	• DO 1.1 - Document the business requirements • DO 1.2 – Define the security/compliance requirements • DO 1.3 – Define the business impact • DO 1.4 – Determine the risk profile
Stage II - Definition of the Technical Scope (DTS)	• DTS 2.1 – Enumerate Software components • DTS 2.2 – Identify Actors & Data Sinks/Source • DTS 2.3 – Enumerate System-Level services • DTS 2.4 – Enumerate 3rd Party infrastructure. • DTS 2.5 – Assert completeness of secure design.
Stage III - Application Decomposition and Analysis (ADA)	• ADA 3.1 – Enumerate all application use cases • ADA 3.2 – Document Data Flow Diagrams (DFDs) • ADA 3.3 – Security functional analysis & the use of trust boundaries
Stage IV - Threat Analysis (TA)	• TA 4.1 – Analyze the overall threat scenario • TA 4.2 – Gather threat information from internal threat sources • TA 4.3 – Gather threat information from External threat sources • TA 4.4 – Update the threat libraries • TA 4.5 – Threat agents to assets mapping. • TA 4.6 – Assignment of the probabilistic values for identified threats
Stage V - Weakness and Vulnerability Analysis (WVA)	• WVA 5.1 – Review/correlate existing vulnerabilities • WVA 5.2 – Identify weak design patterns in the architecture • WVA 5.3 – Map threats to vulnerabilities • WVA 5.4 – Provide Context risk Analysis based upon Threat-Vulnerability • WVA 5.5 – Conduct targeted vulnerability testing
Stage VI - Attack Modeling & Simulation (AMS)	• AMS 6.1 – Analyze the attack scenarios • AMS 6.2 – Update the attack library/vectors and the control framework • AMS 6.3 – Identify the attack surface and enumerate the attack vectors • AMS 6.4 – Assess the probability and impact of each attack scenario. • AMS 6.5 – Derive a set of cases to test existing countermeasures. • AMS 6.6 – Conduct attack driven security tests and simulations
STAGE VII - Risk Analysis & Management (RAM)	• RAM 7.1 – Calculate the risk of each threat • RAM 7.2 – Identify countermeasures and risk mitigations measures • RAM 7.3 – Calculate the residual risks • RAM 7.4 – Recommend strategies to manage risks


STRIDE
Spoofing
Elevation of Privilege
Tampering
Repudation
Information Disclosure
Deniel of Service
Authentication
Session Handling
Authorization
Input Validation
Integrity
Audit
Exception Handling
Confidentiality
Availability

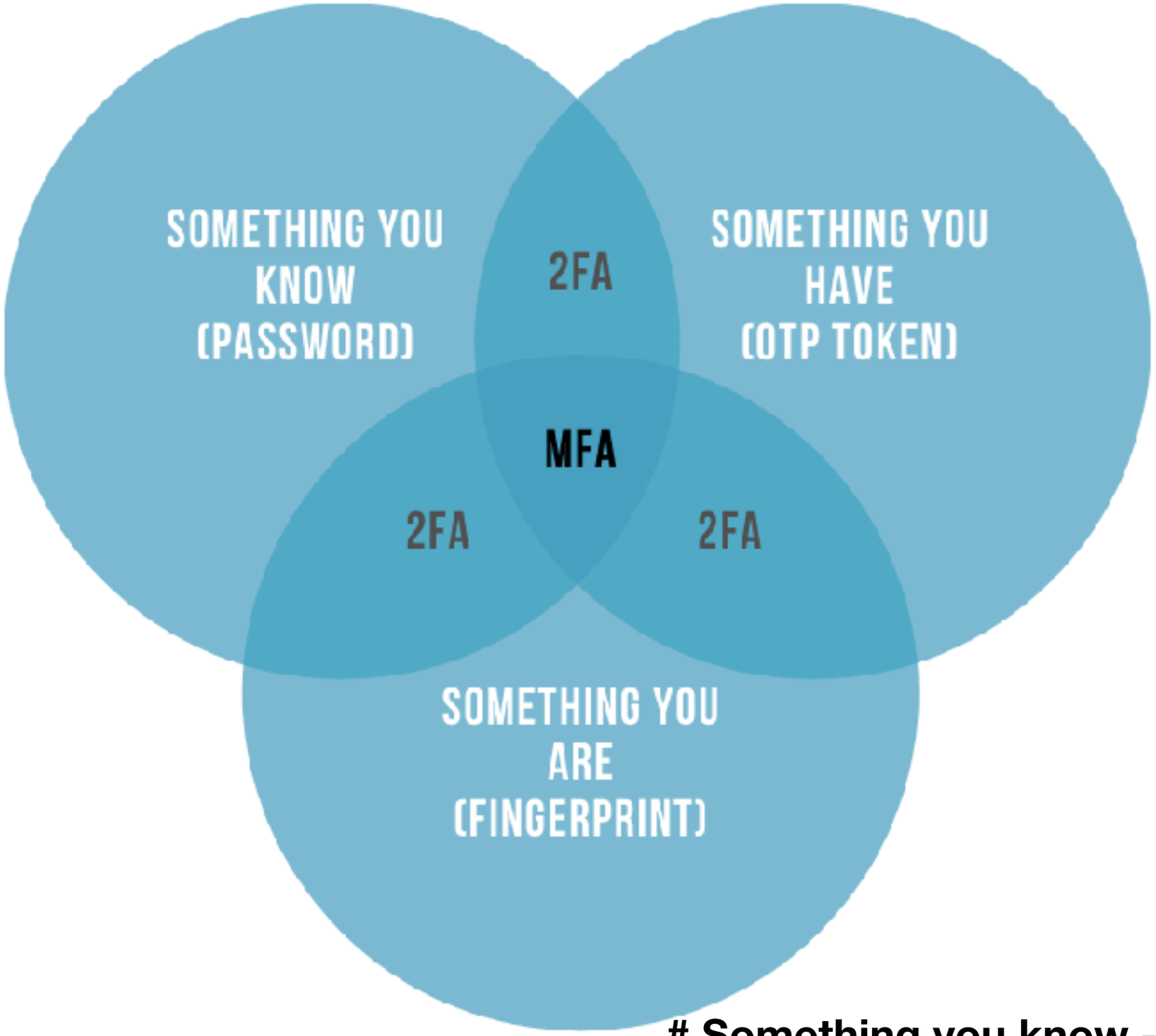
Concerns

- Authentication
- Authorization
- Audit
- Data Security
  - In Rest
  - In Transit
- Input Validation/ Output Sanitization
- Exception Handling
- Session Management
- Key Management

Approach

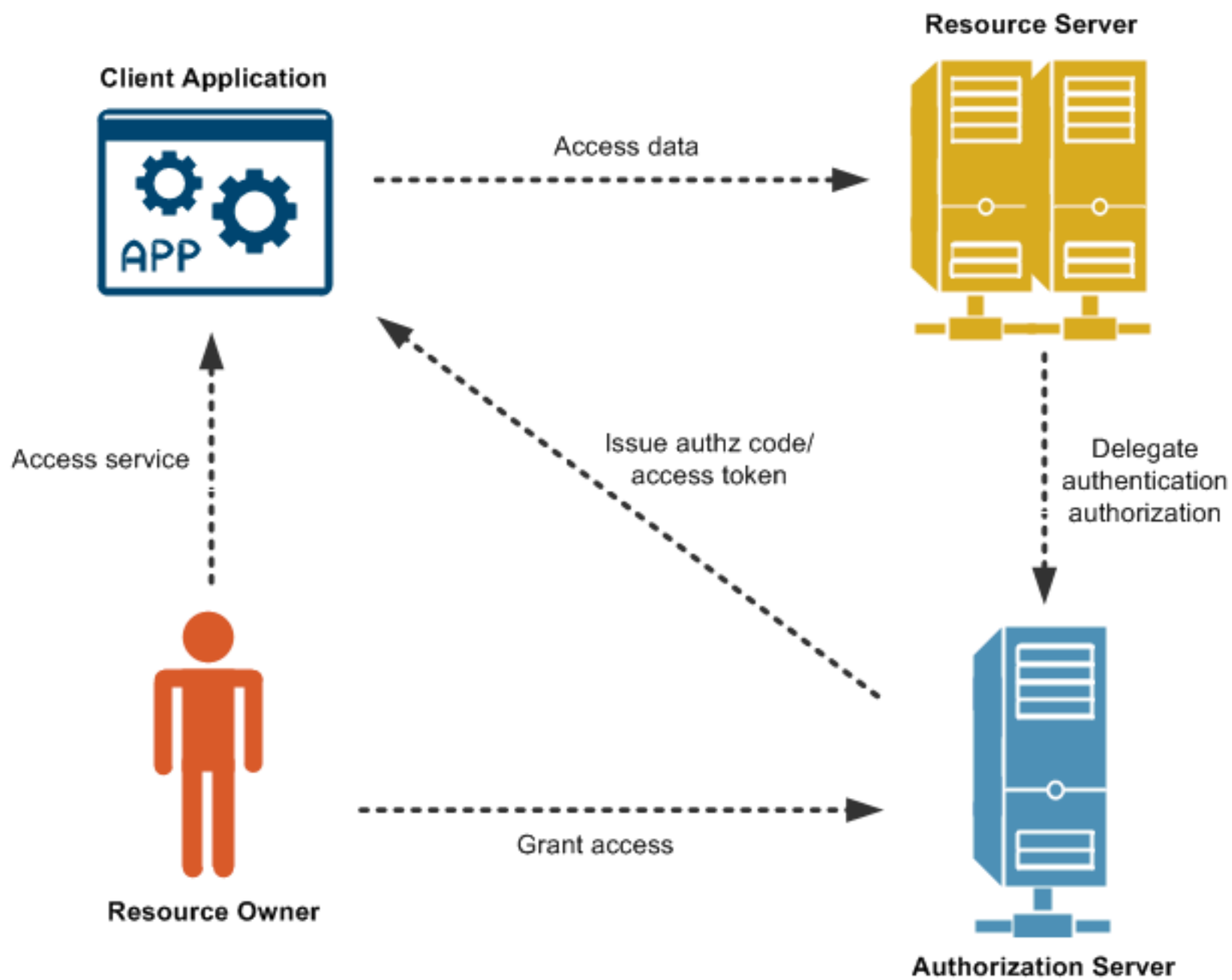
- SAML
- SSO
- Federated
- OAuth2
- Kerberos
-

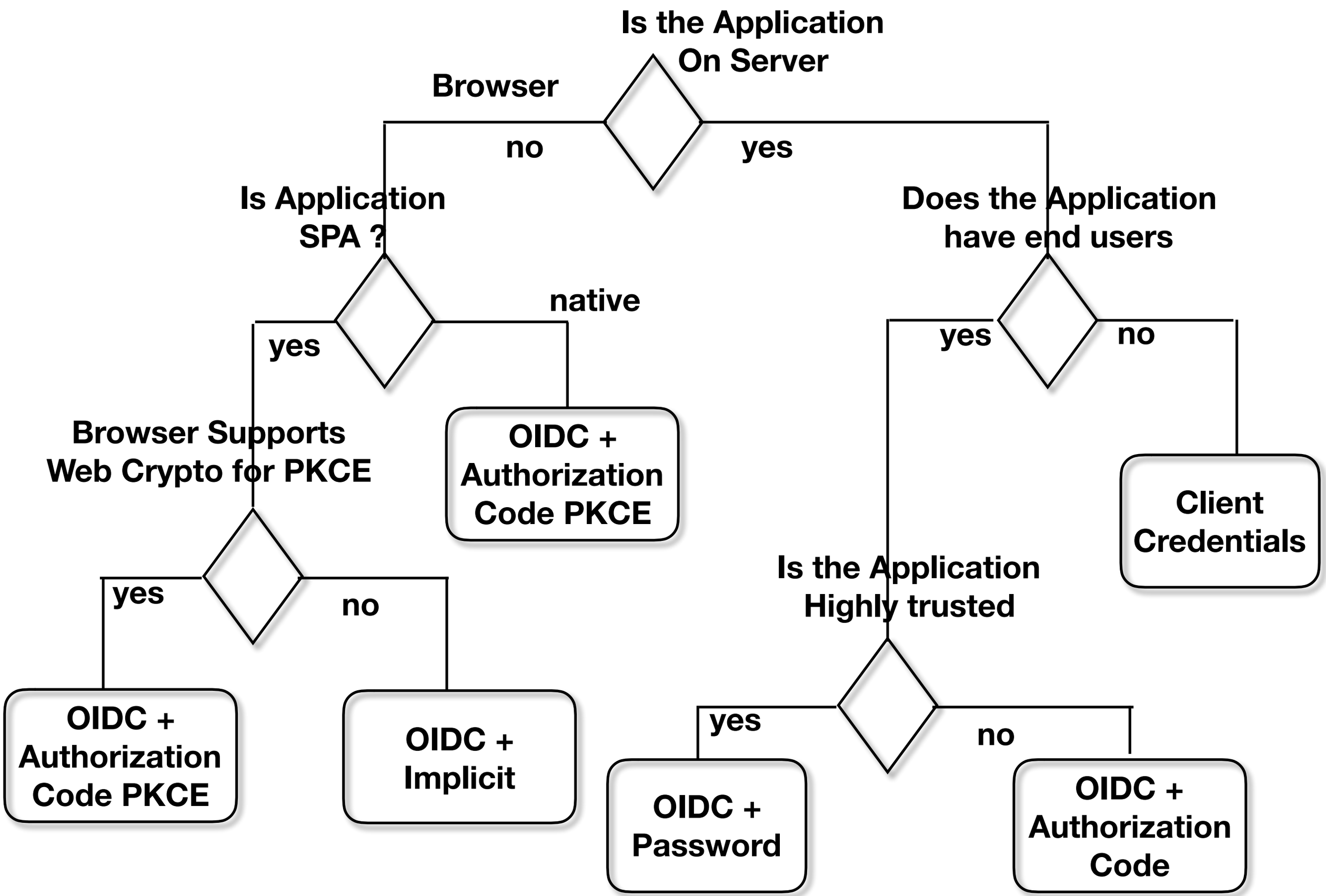
Step 1. Authentication



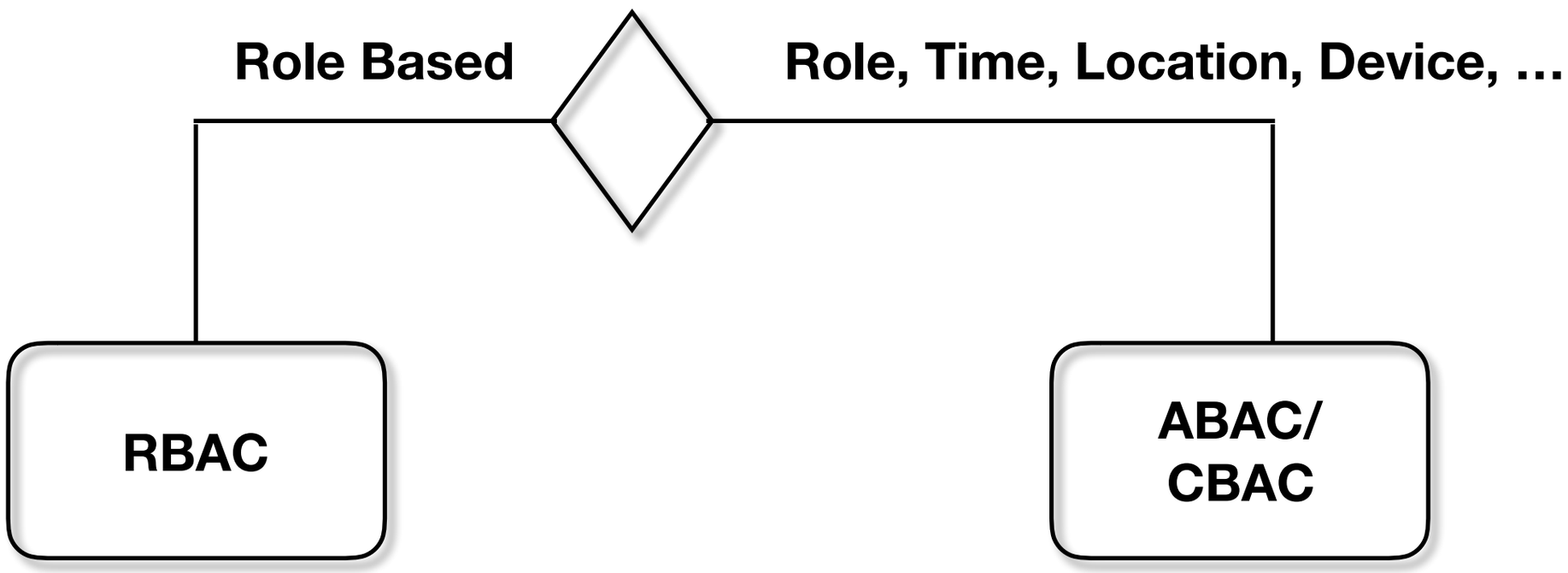
 # Something you know - pwd, security question
 # Something you Have - otp, cert, rsa
 # Where you are - country, office, …
 # Something you are - voice, face, dna, ..
 # Something you do - behaviour

Centralize

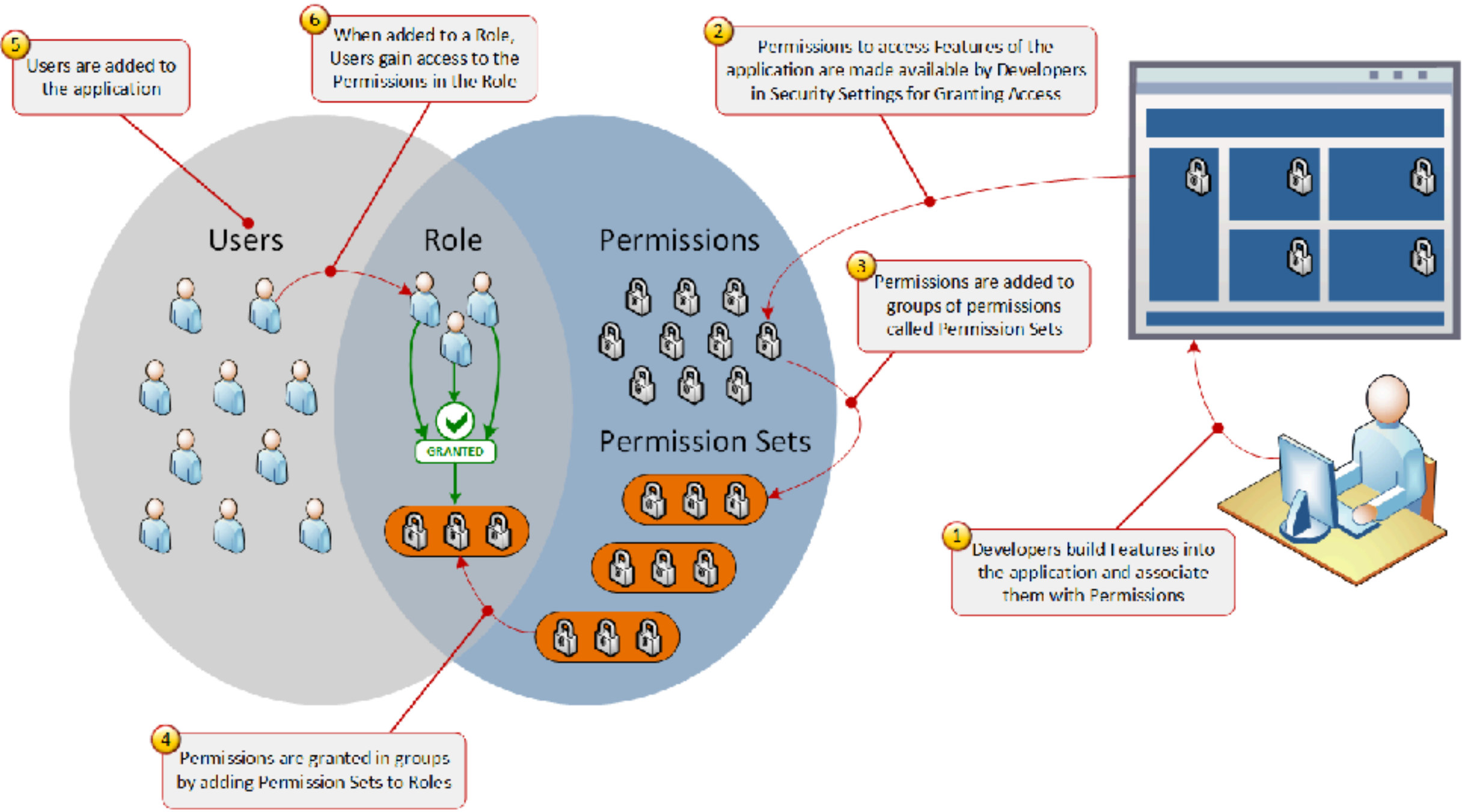

Resource Server
Client Application
APP
Access data
Access service
Issue authz code/
access token
Delegate
authentication
authorization
Grant access
Resource Owner
Authorization Server


Is the Application On Server
Browser
no
yes
Is Application SPA ?
Does the Application have end users
native
yes
yes
no
Browser Supports Web Crypto for PKCE
OIDC + Authorization Code PKCE
Client Credentials
Is the Application Highly trusted
yes
no
yes
no
OIDC + Authorization Code PKCE
OIDC + Implicit
OIDC + Password
OIDC + Authorization Code

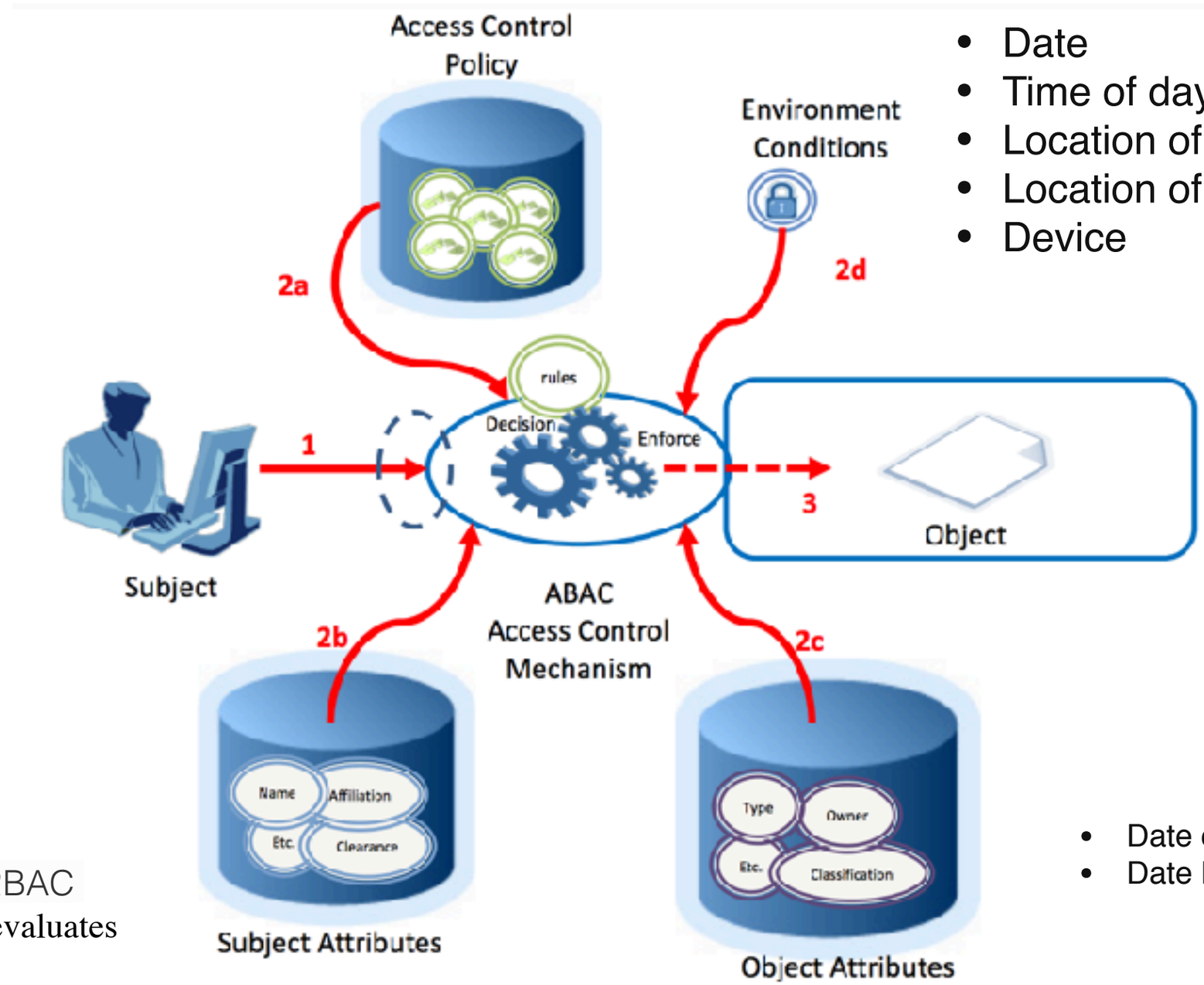
Step 2. Authorization



Role Based Access Control (RBAC)



Attribute Based Access Control (ABAC) / Claim Based Access Control (CBAC)



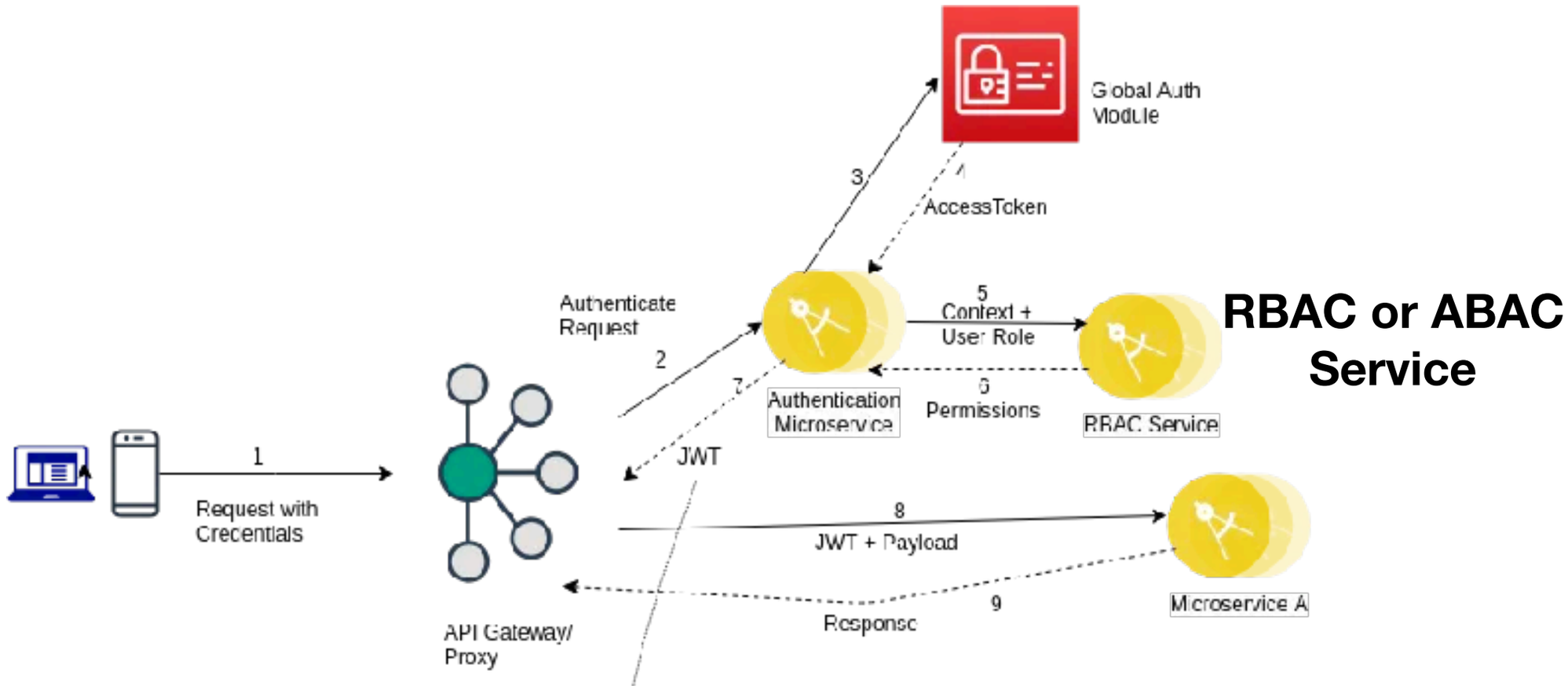
- Date
- Time of day
- Location of data
- Location of user
- Device

- Date created
- Date Modified

ABAC is an extension of RBAC

Access Control Mechanism evaluates

a) Rules (RBAC),
b) Subject Attributes,
c) Object Attributes
d) Environment Conditions
to compute a decision




```
{
"UserData": {
"userId":"#{USER_ID}",
"email":'#{USER_EMAIL}",
"first_name':"#{USER_FIRST_NAME}",
"last_name":'#{USER_LAST_NAME}',
"name":"#{USER_NAME}",
"roles": {
"#{ROLE_NAME}":"#{ROLE_ID}"
},
"permissions": {
"#{PERMISSION_NAME}":'#{PERMISSION_ID}",
}
},
"exp':148761231
}
```

Step 3. Audit

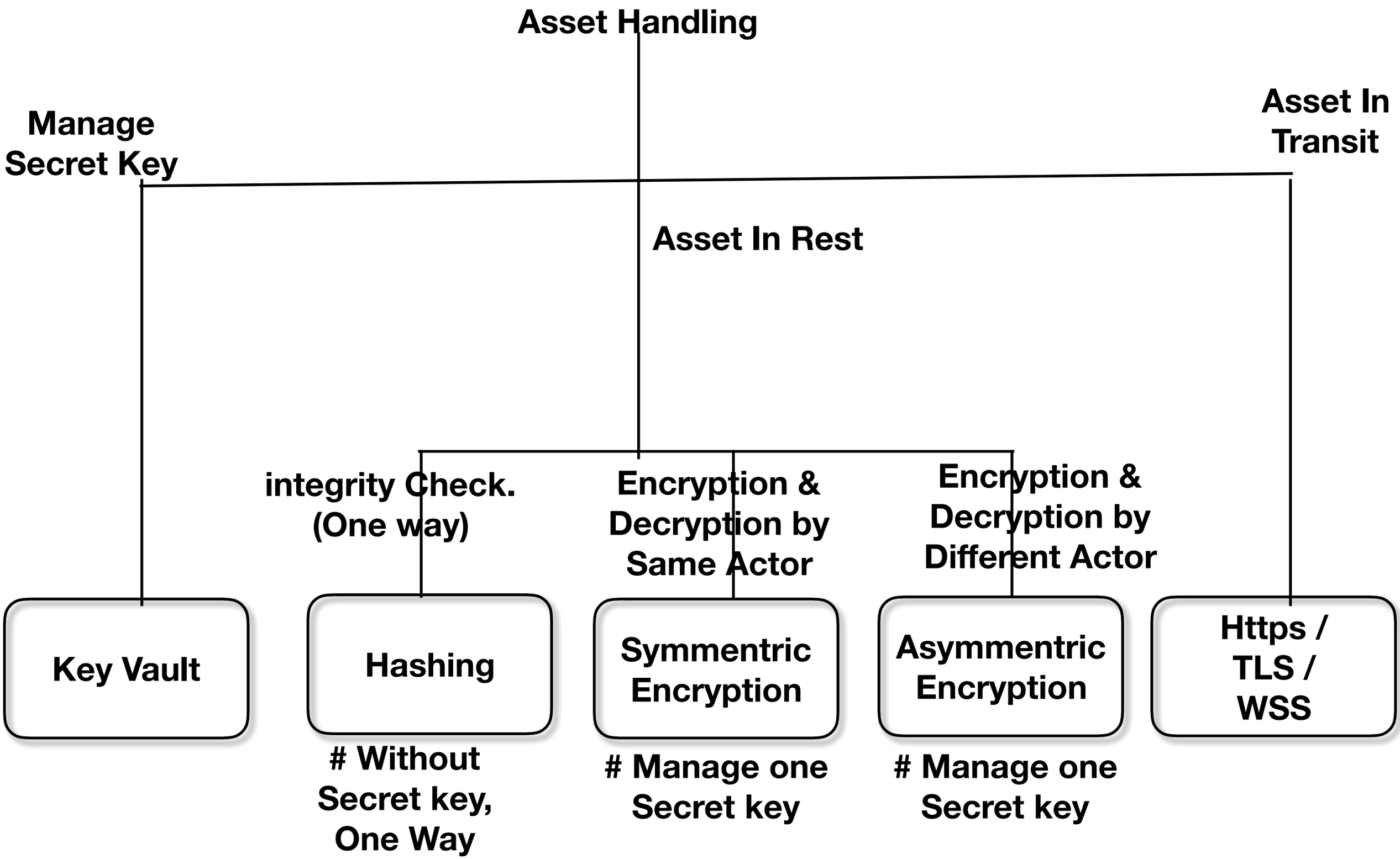
who - Identity

what - Action

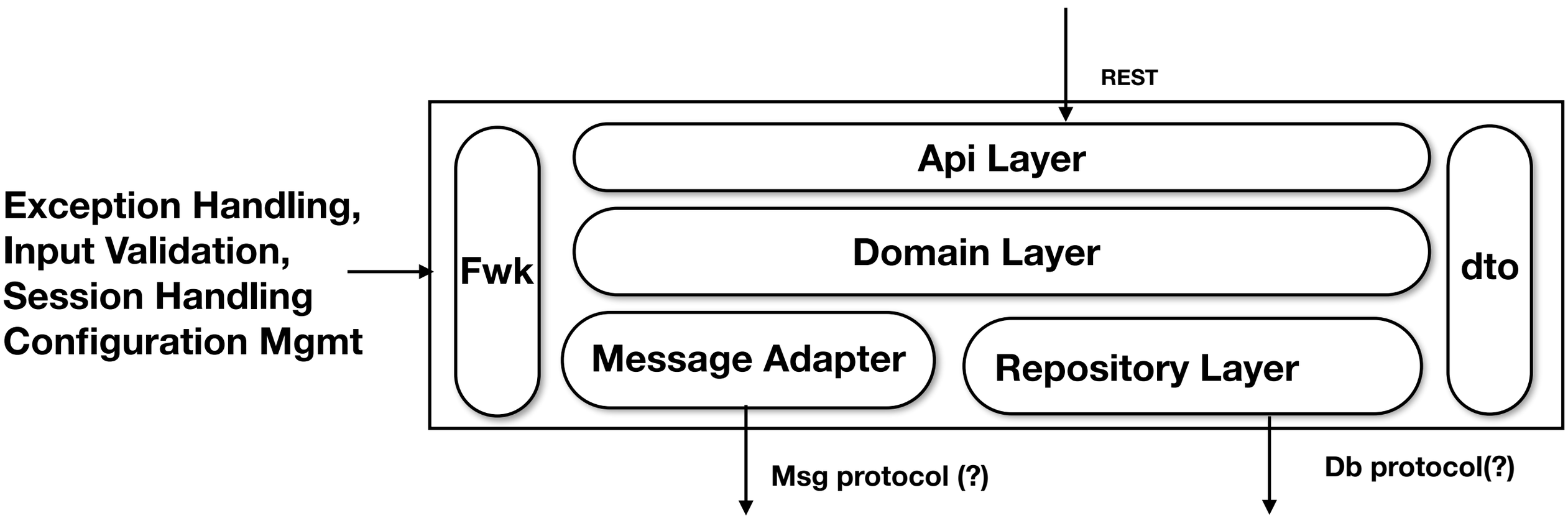
where - Component/Service/Object/Method

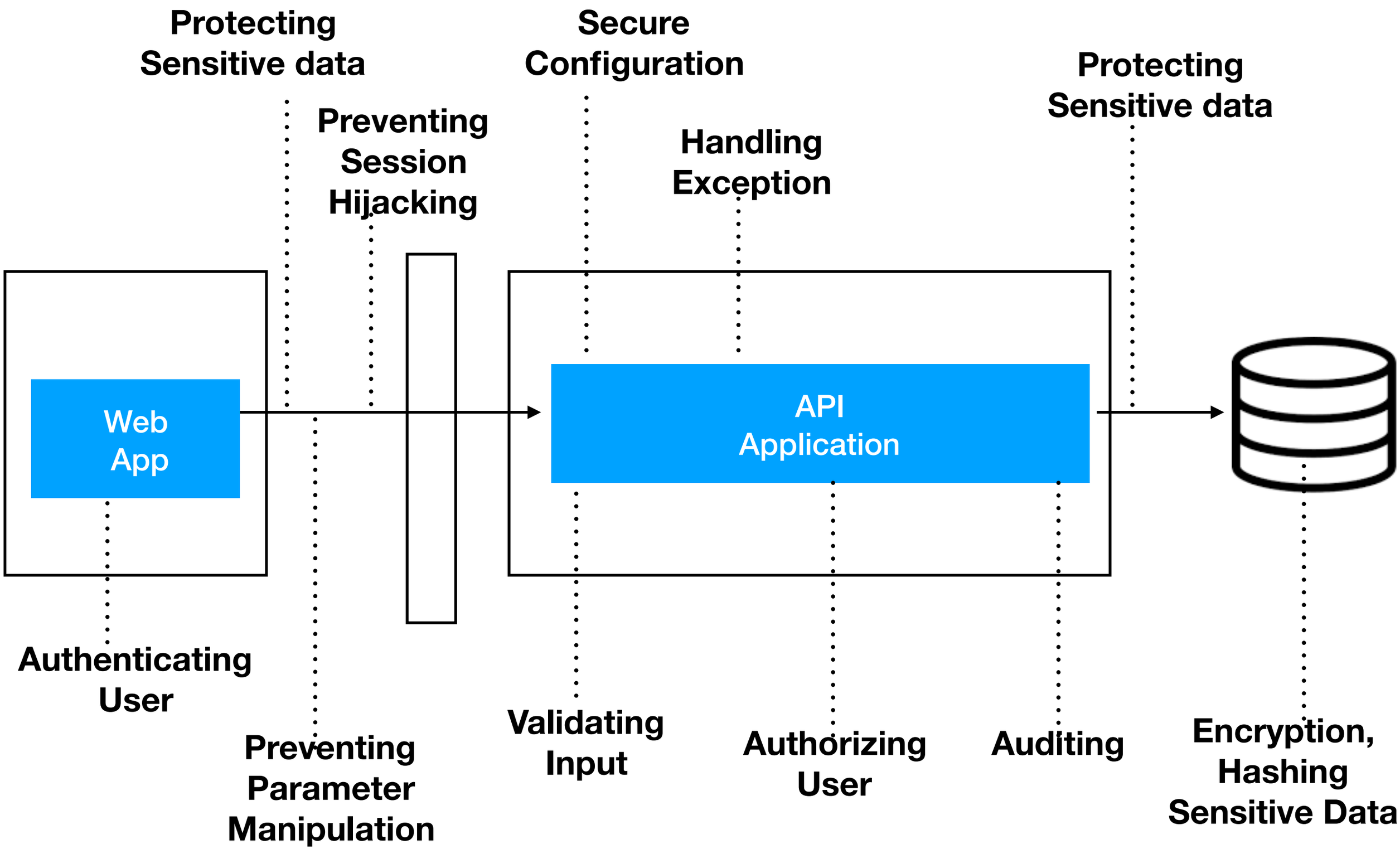
when - Timestamp:

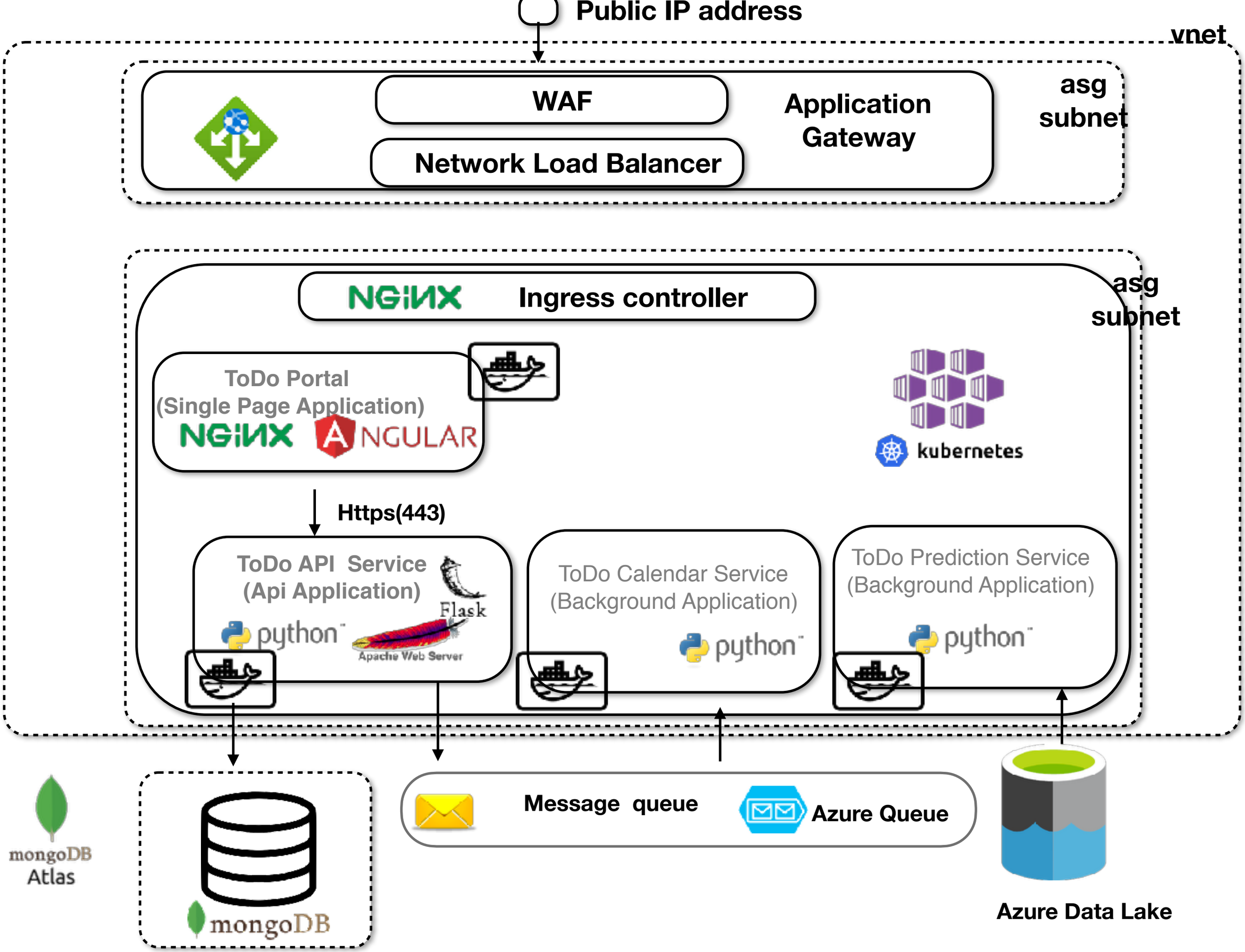
Step 4. Privacy and Confidentiality



Step 5. Cross cutting concerns



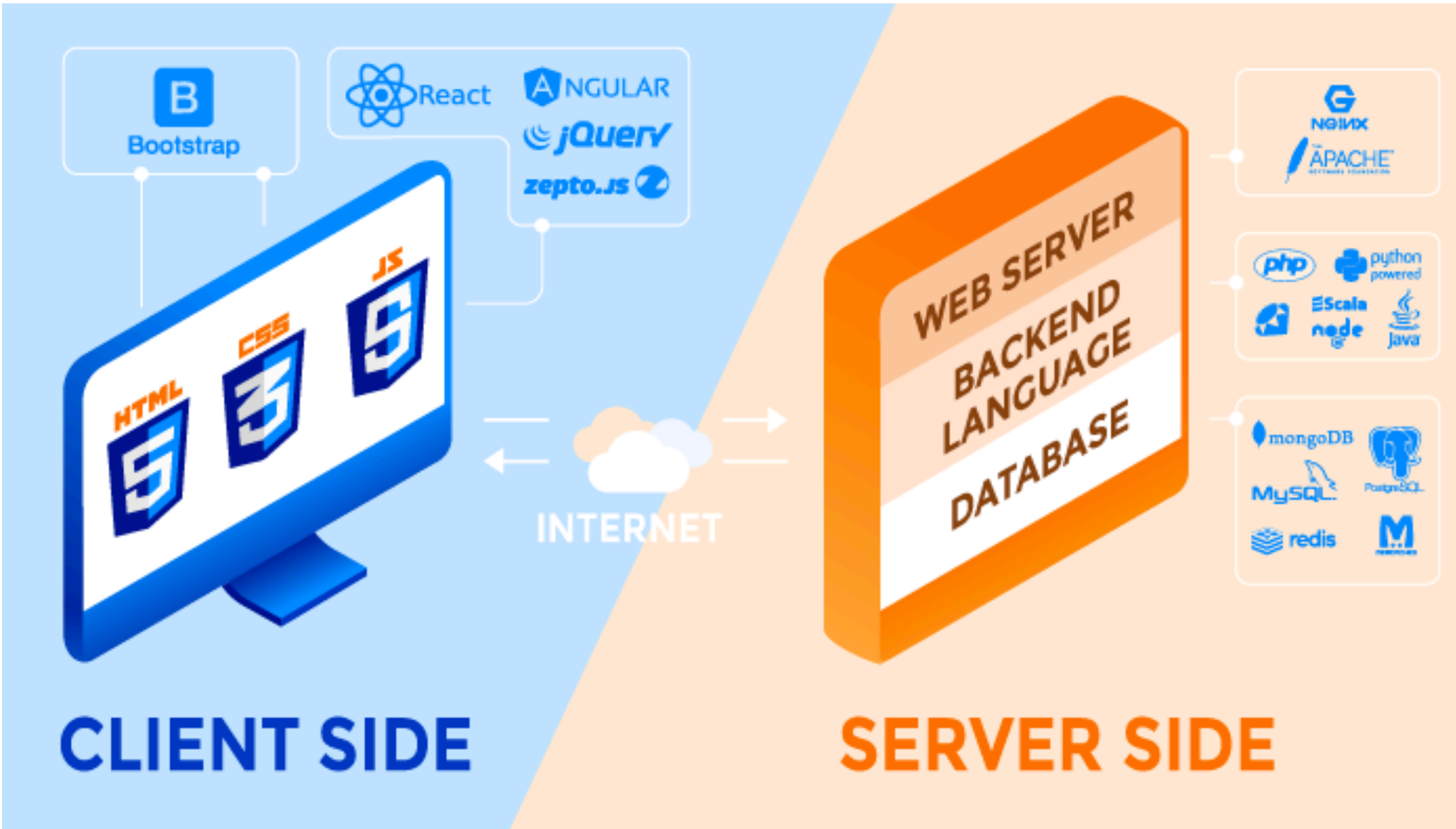

Protecting Sensitive data
Secure Configuration
Protecting Sensitive data
Preventing Session Hijacking
Handling Exception
Web App
API Application
Authenticating User
Preventing Parameter Manipulation
Validating Input
Authorizing User
Auditing
Encryption, Hashing Sensitive Data

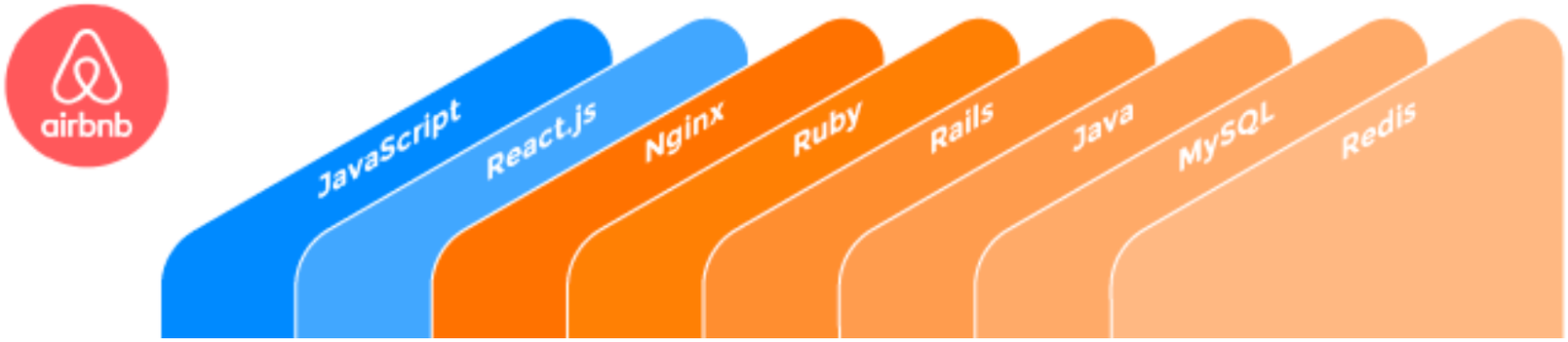

Public IP address
vnet
WAF
Application Gateway
Network Load Balancer
asg subnet
NGINX
Ingress controller
asg subnet
ToDo Portal
(Single Page Application)
NGINX
ANGULAR
kubernetes
Https(443)
ToDo API Service
(Api Application)
Flask
python
Apache Web Server
ToDo Calendar Service
(Background Application)
python
ToDo Prediction Service
(Background Application)
python
mongoDB
Atlas
mongoDB
Message queue
Azure Queue
Azure Data Lake

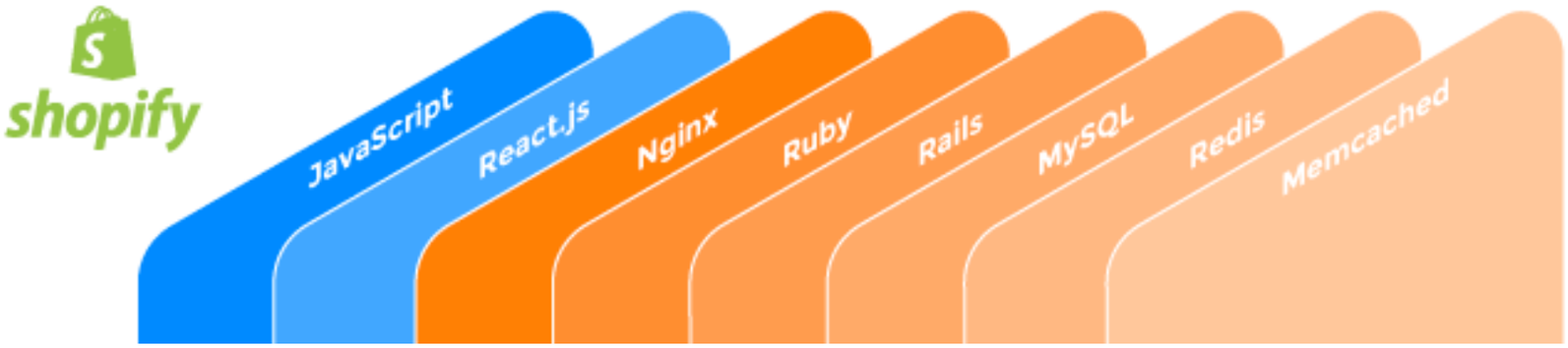
6. Cross cutting concerns

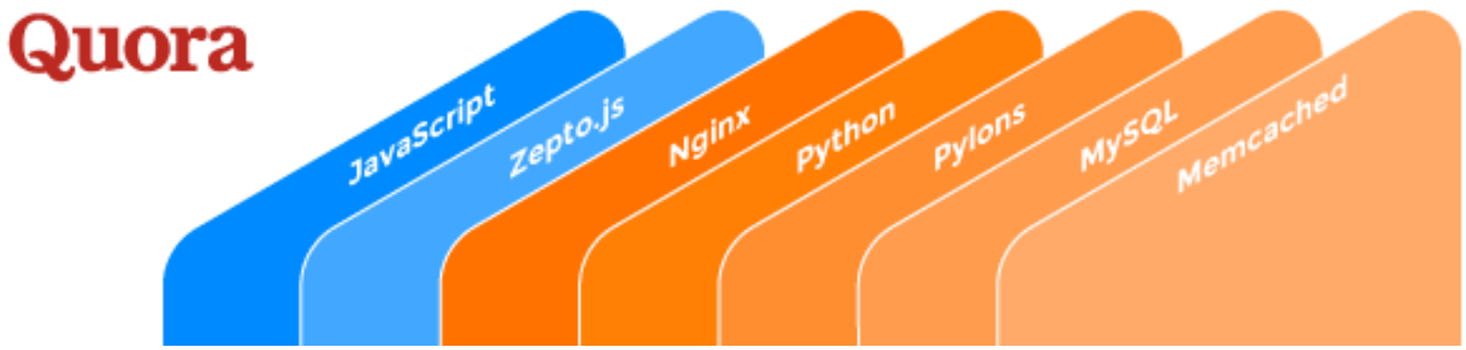
The **crosscutting concern** is a concern which is needed in almost every module of an application.

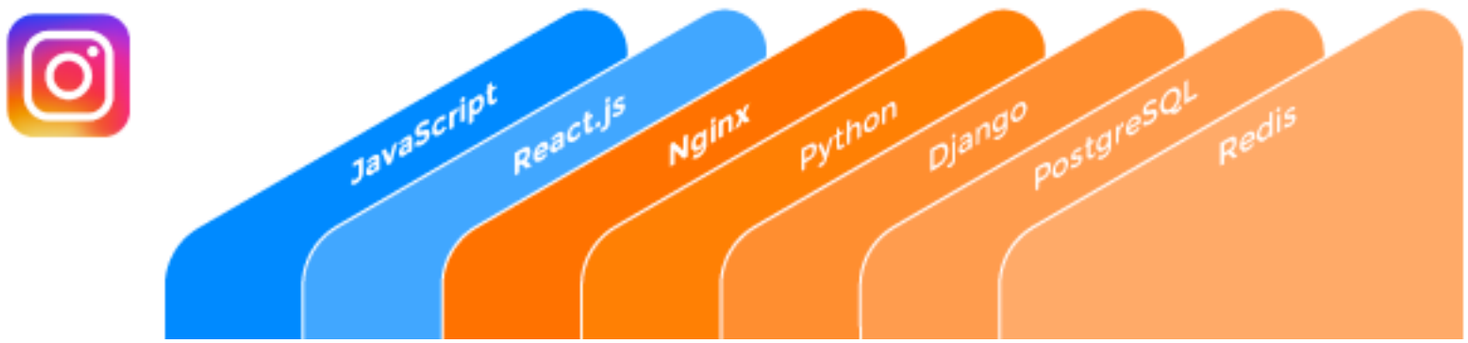
7. Technology View



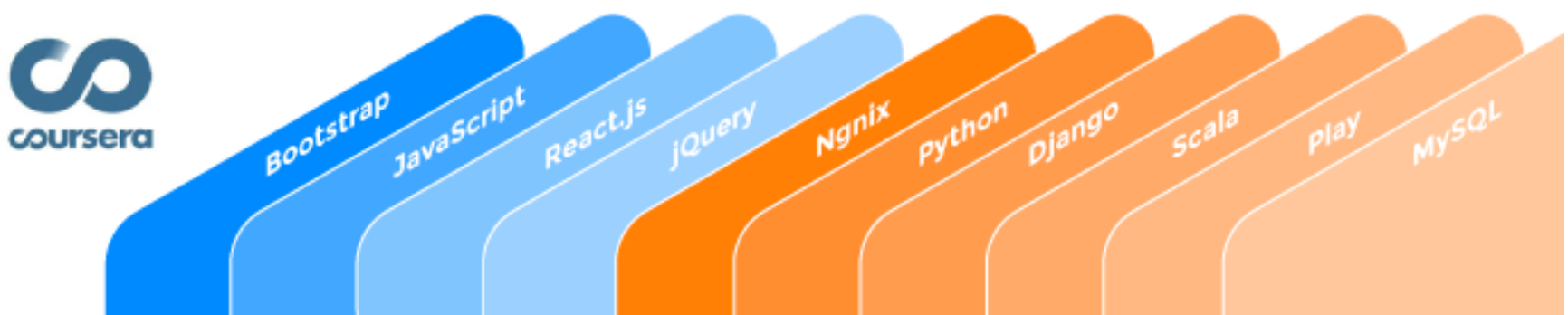

airbnb
JavaScript
React.js
Nginx
Ruby
Rails
Java
MySQL
Redis

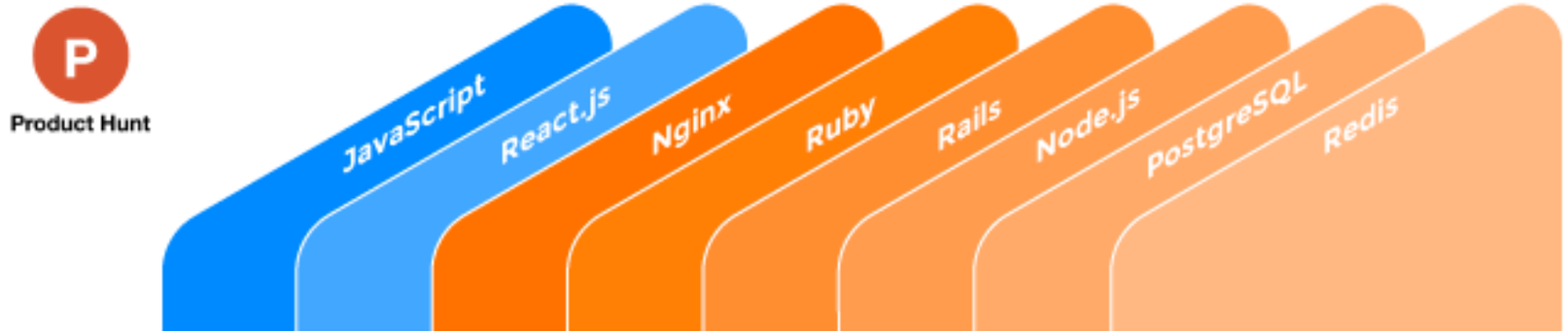

shopify
JavaScript
React.js
Nginx
Ruby
Rails
MySQL
Redis
Memcached

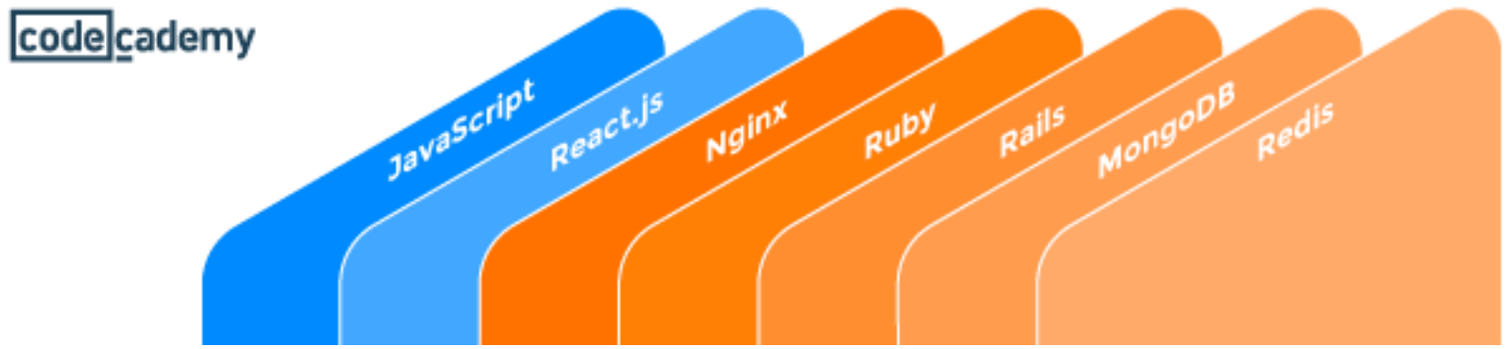

Quora
JavaScript
Zepto.js
Nginx
Python
Pylons
MySQL
Memcached

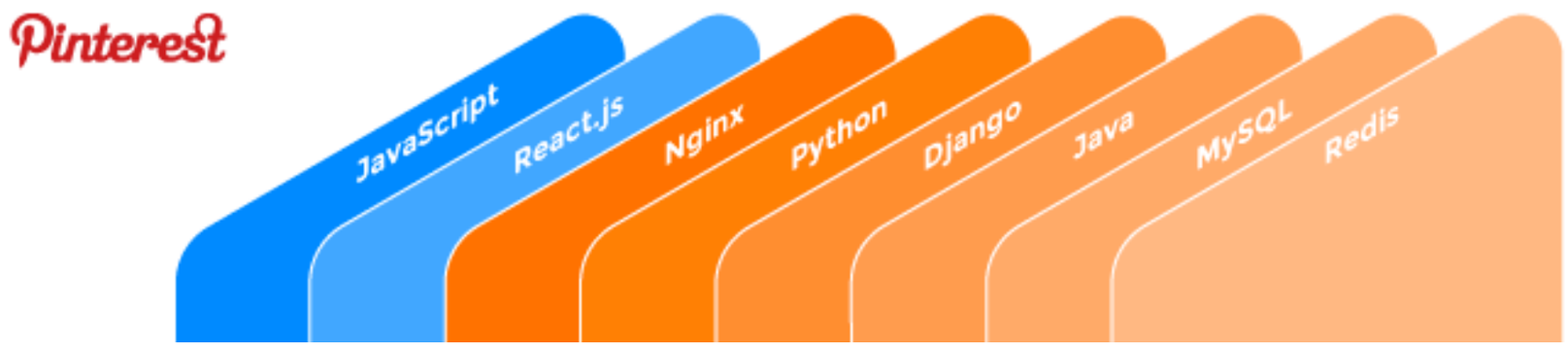

JavaScript
React.js
Nginx
Python
Django
PostgreSQL
Redis

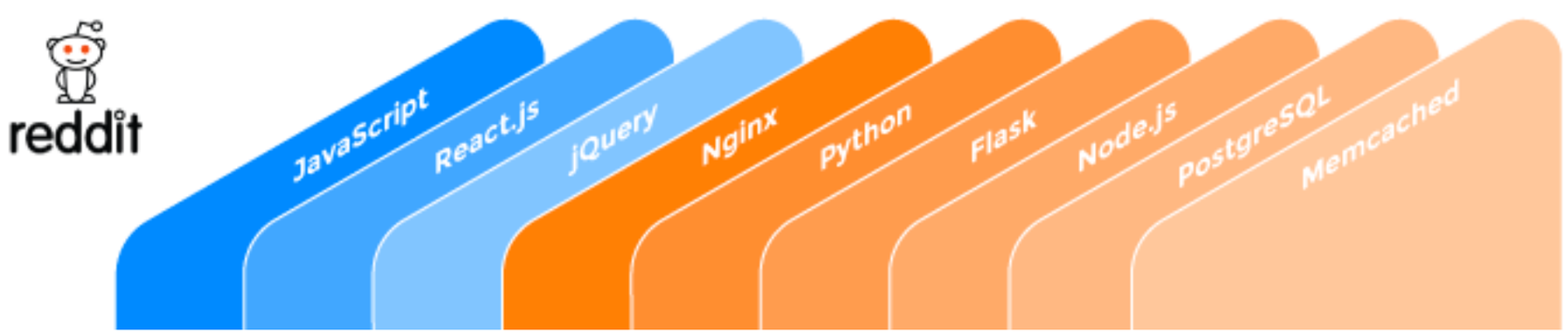
Instagram


coursera
Bootstrap
JavaScript
React.js
jQuery
Ngnix
Python
Django
Scala
Play
MySQL


Product Hunt
JavaScript
React.js
Nginx
Ruby
Rails
Node.js
PostgreSQL
Redis


codecademy
JavaScript
React.js
Nginx
Ruby
Rails
MongoDB
Redis


Pinterest
JavaScript
React.js
Nginx
Python
Django
Java
MySQL
Redis


reddit
JavaScript
React.js
jQuery
Nginx
Python
Flask
Node.js
PostgreSQL
Memcached

JavaScript
React.js
Apache
PHP
MySQL
Memcached

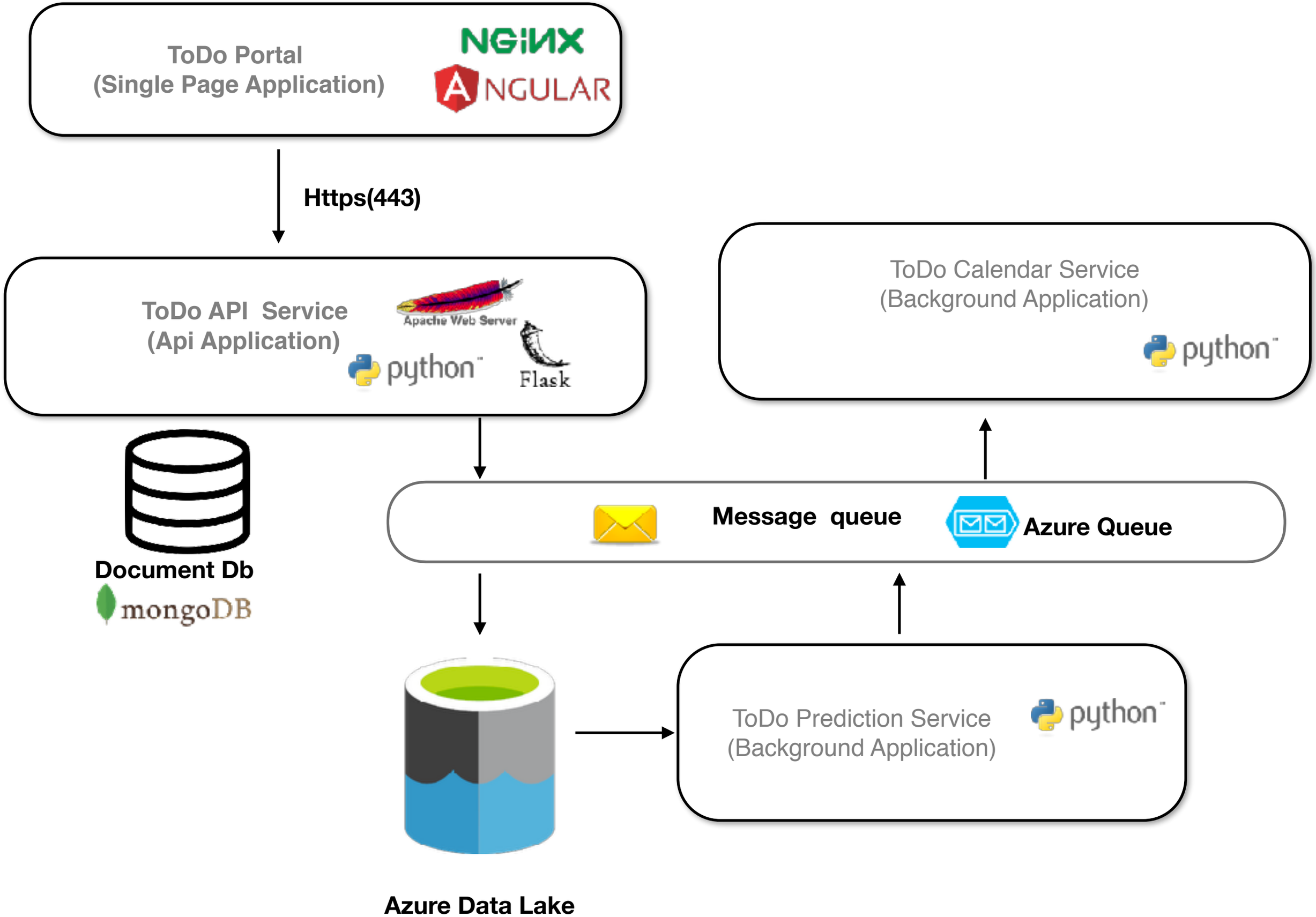
NUMBER OF CONTRIBUTORS ON GITHUB

JUNE 2017

AVERAGE DEVELOPER SALARIES IN THE US

BY TECHNOLOGY

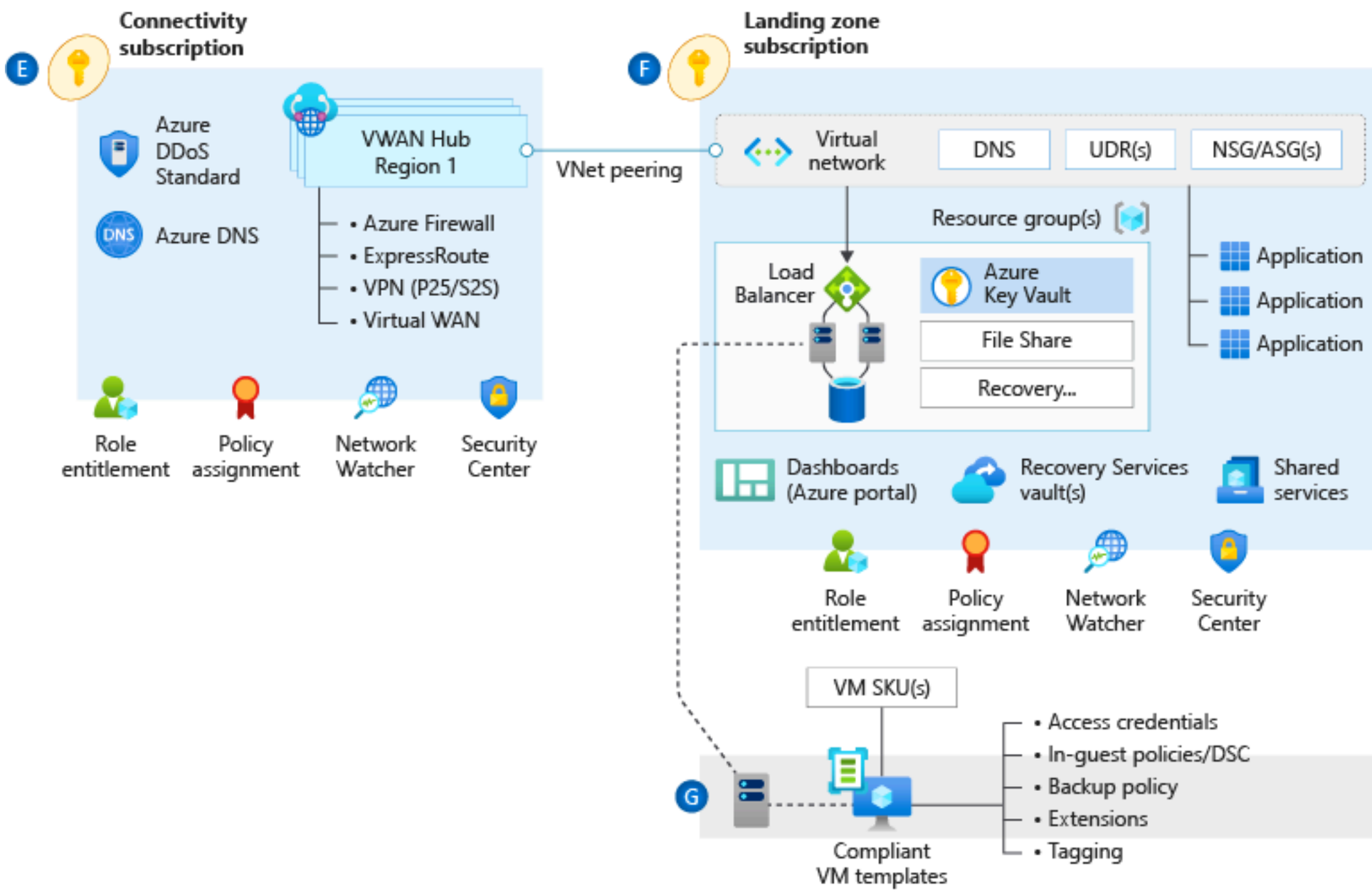
Source: Indeed.com

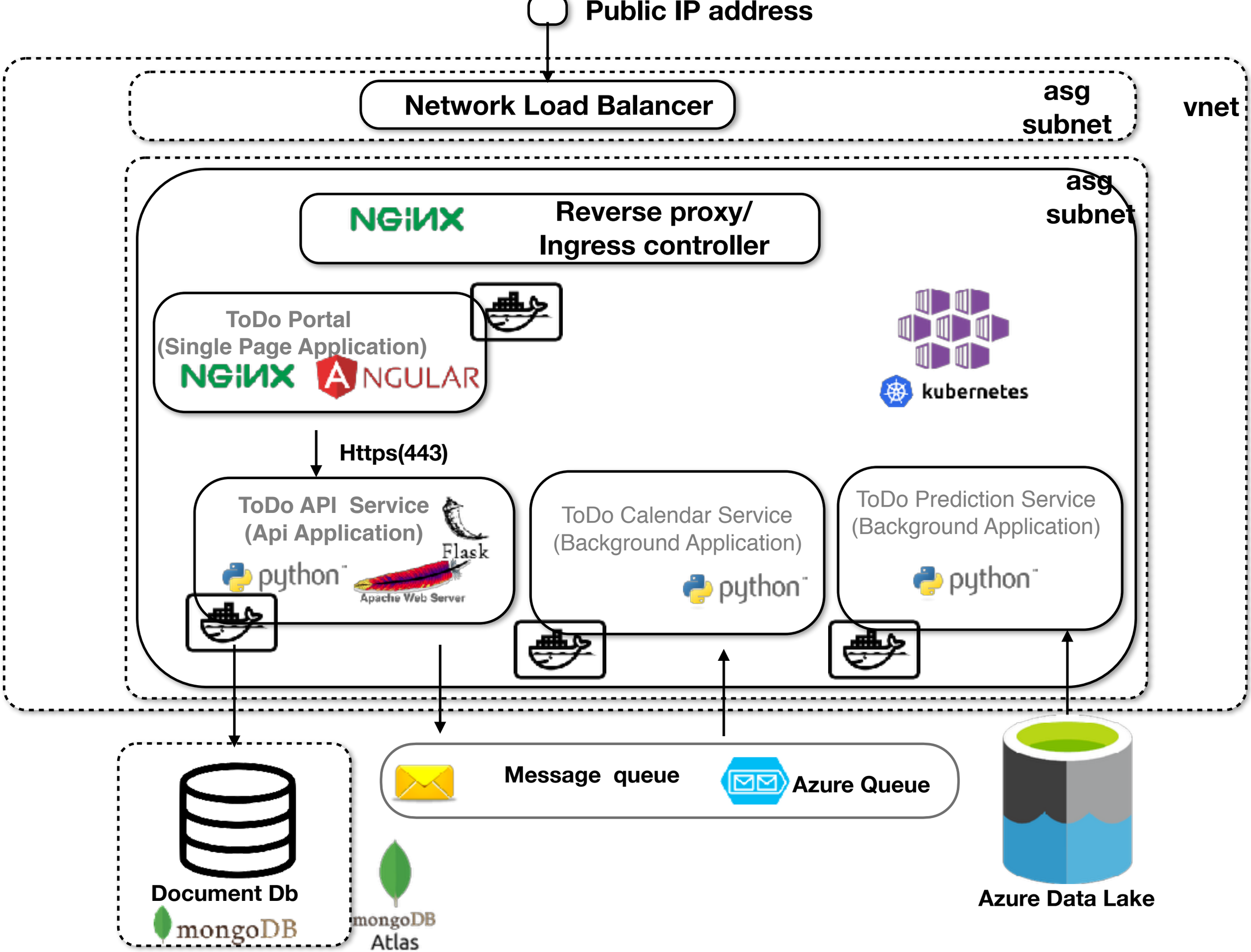

ToDo Portal
(Single Page Application)
NGINX
ANGULAR
Https(443)
ToDo API Service
(Api Application)
Apache Web Server
python
Flask
ToDo Calendar Service
(Background Application)
python
Document Db
mongoDB
Message queue
Azure Queue
ToDo Prediction Service
(Background Application)
python
Azure Data Lake

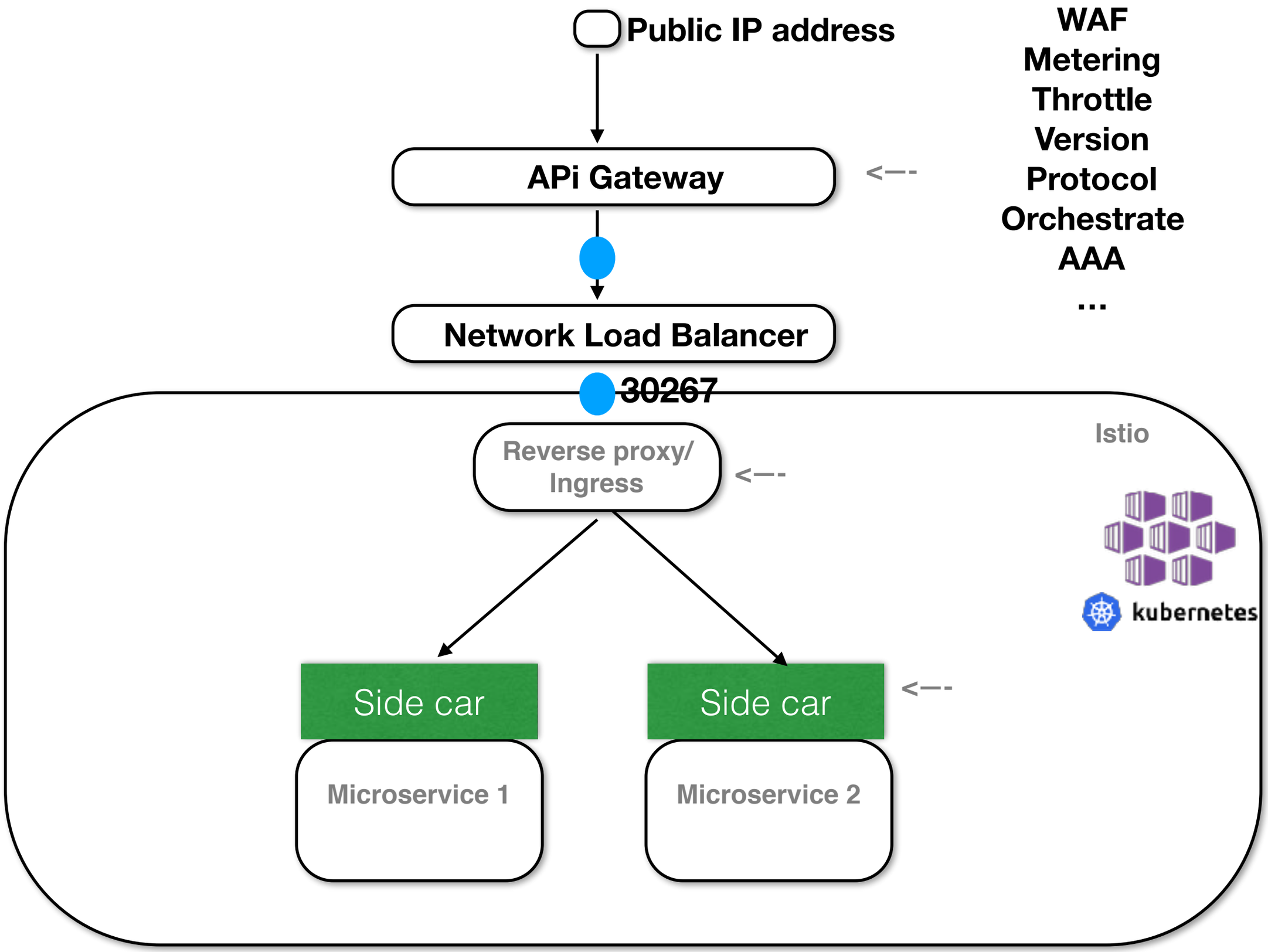
Budget	
Libraries	
Learning curve	
Support	
License	
Expertise	
Infra	
Community	
Security	
Development time	
Vendor	

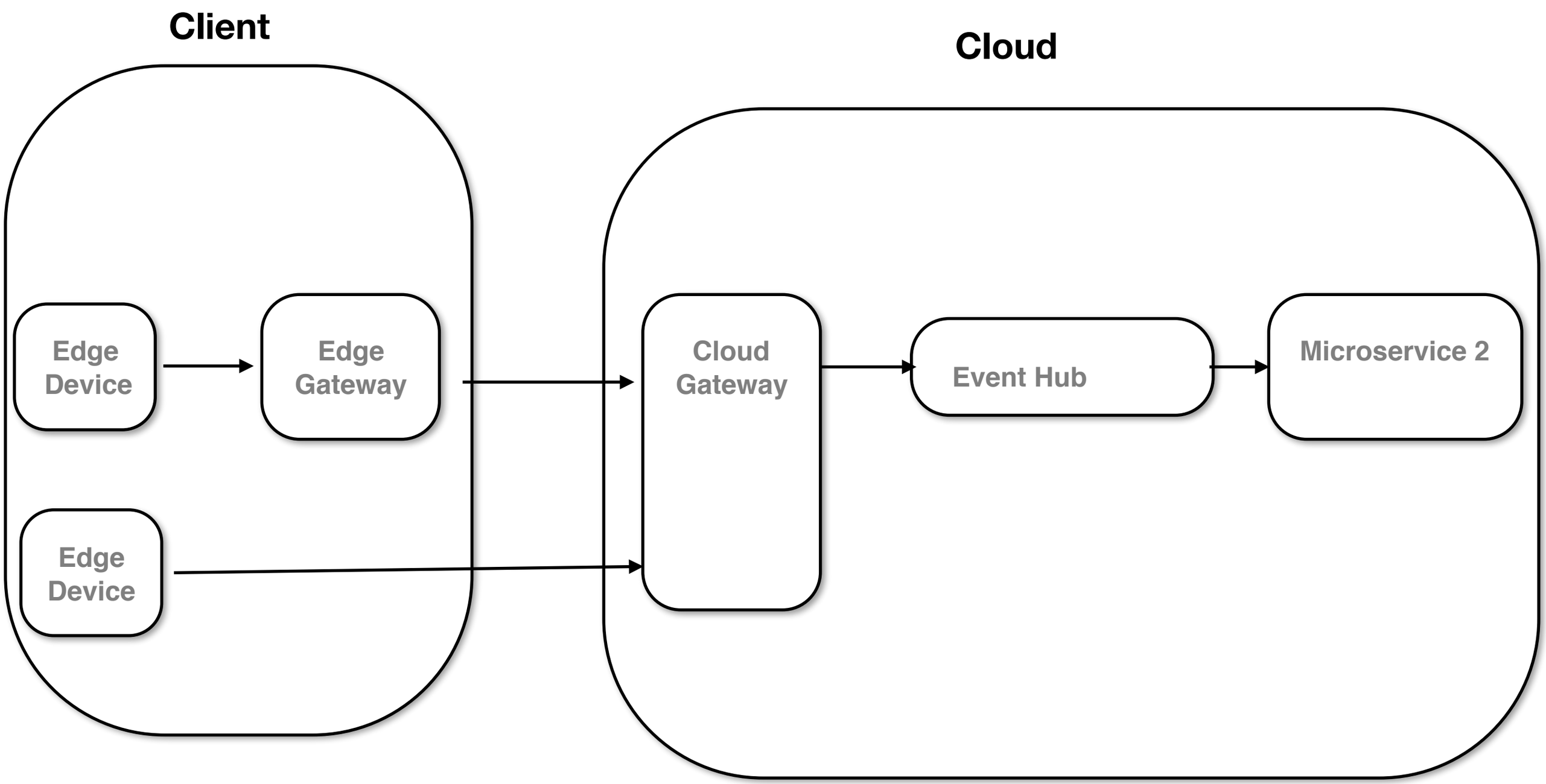
8. Infrastructure View

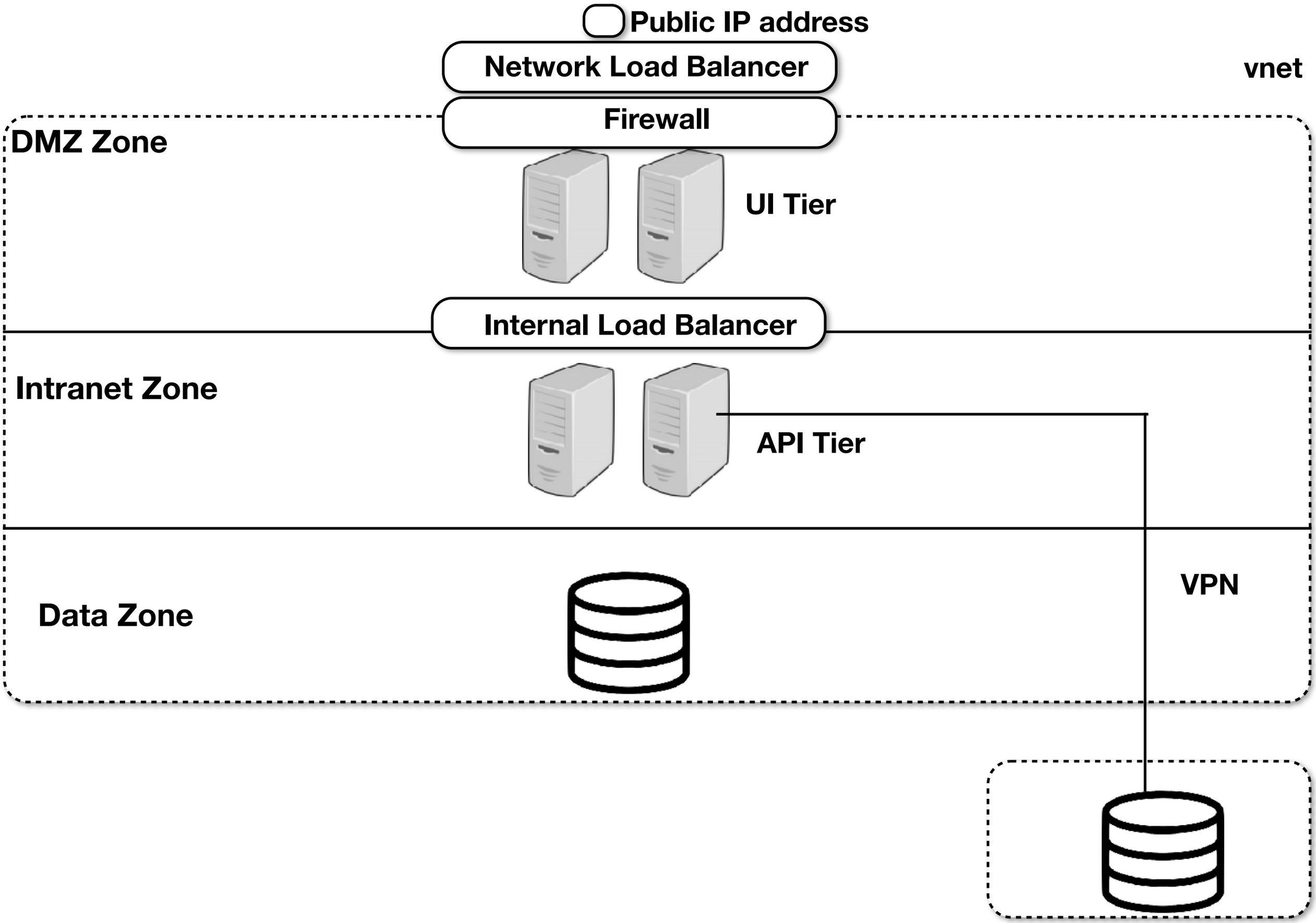
Physical View



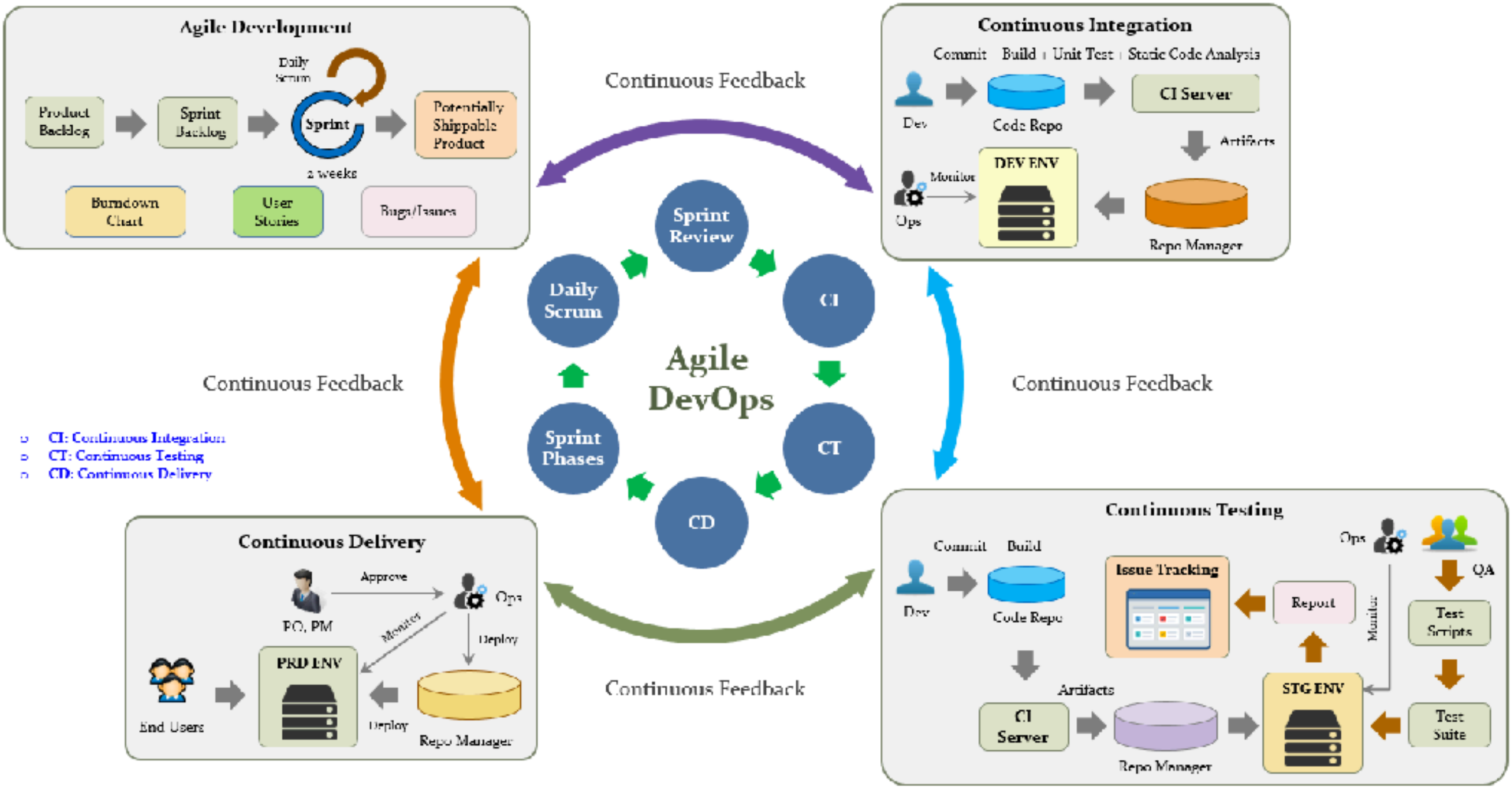

Public IP address
Network Load Balancer
asg
subnet
vnet
asg
subnet
NGINX
Reverse proxy/
Ingress controller
ToDo Portal
(Single Page Application)
NGINX
ANGULAR
kubernetes
Https(443)
ToDo API Service
(Api Application)
Flask
python
Apache Web Server
ToDo Calendar Service
(Background Application)
python
ToDo Prediction Service
(Background Application)
python
Document Db
mongoDB
mongoDB
Atlas
Message queue
Azure Queue
Azure Data Lake


Public IP address
APi Gateway
<—-
WAF
Metering
Throttle
Version
Protocol
Orchestrate
AAA
...
Network Load Balancer
30267
Istio
Reverse proxy/
Ingress
<—-
kubernetes
Side car
Side car
<—-
Microservice 1
Microservice 2

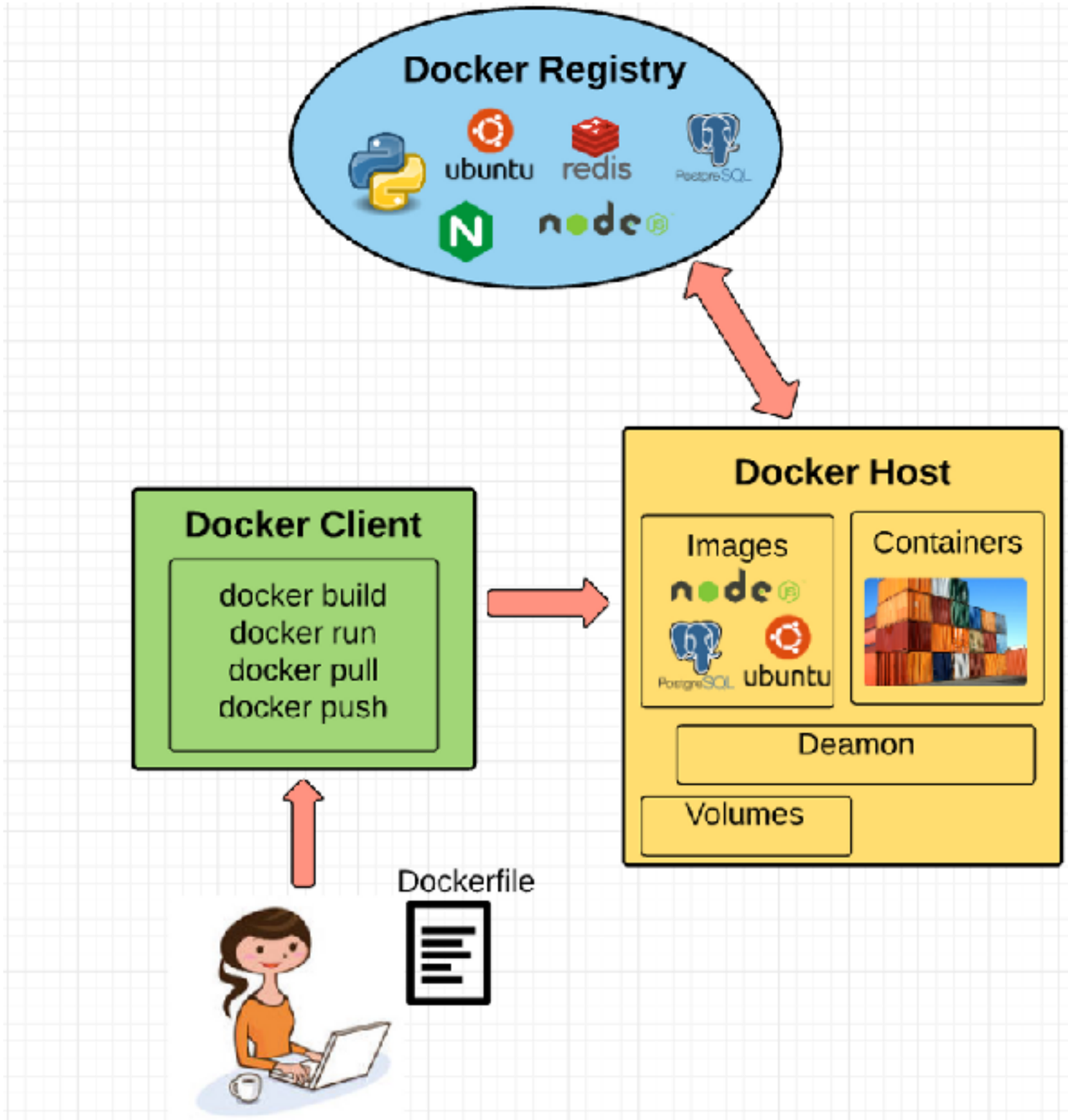

Client
Cloud
Edge Device
Edge Gateway
Edge Device
Cloud Gateway
Event Hub
Microservice 2


Public IP address
Network Load Balancer
vnet
Firewall
DMZ Zone
UI Tier
Internal Load Balancer
Intranet Zone
API Tier
VPN
Data Zone

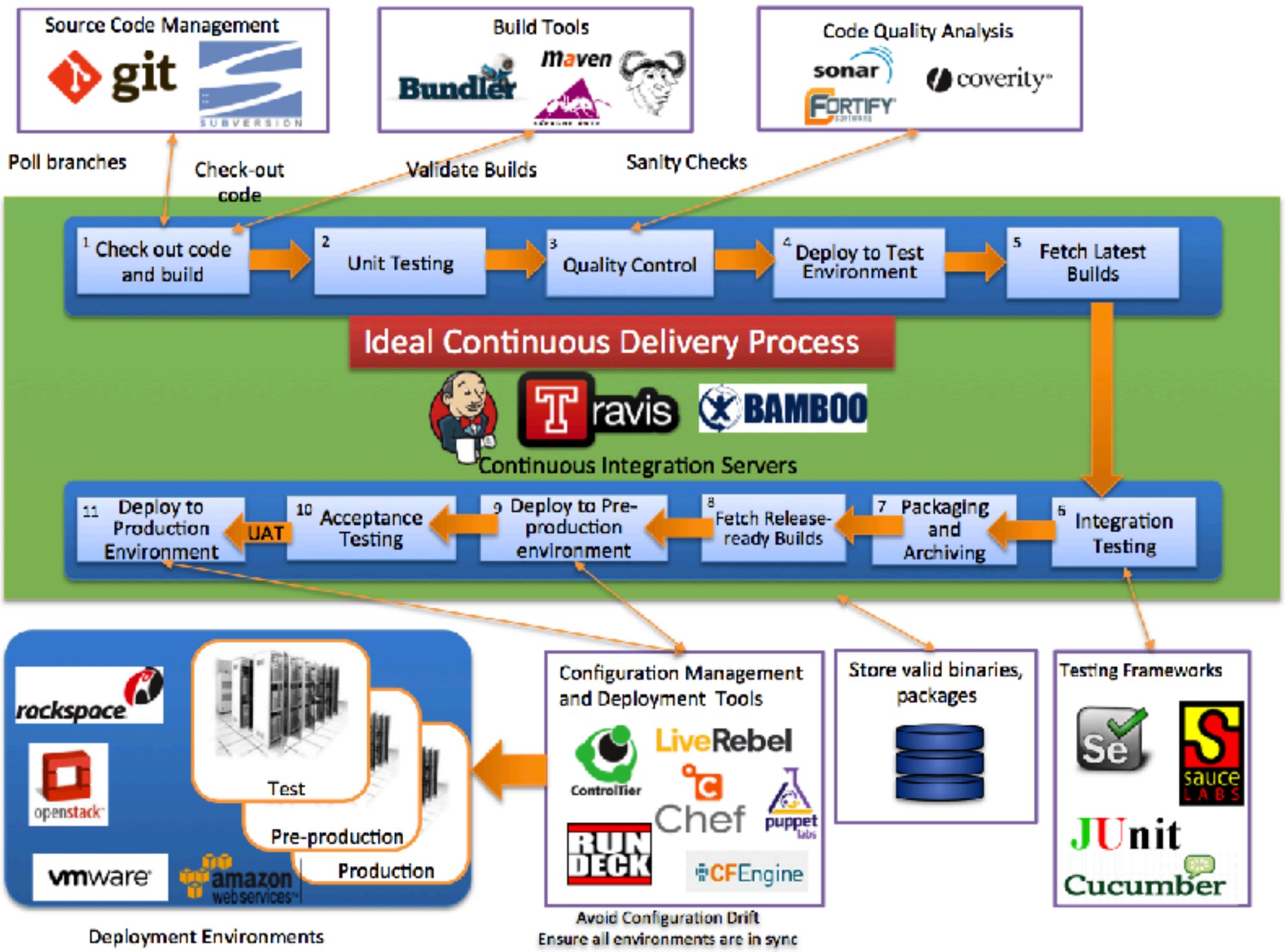
9. DevOps View



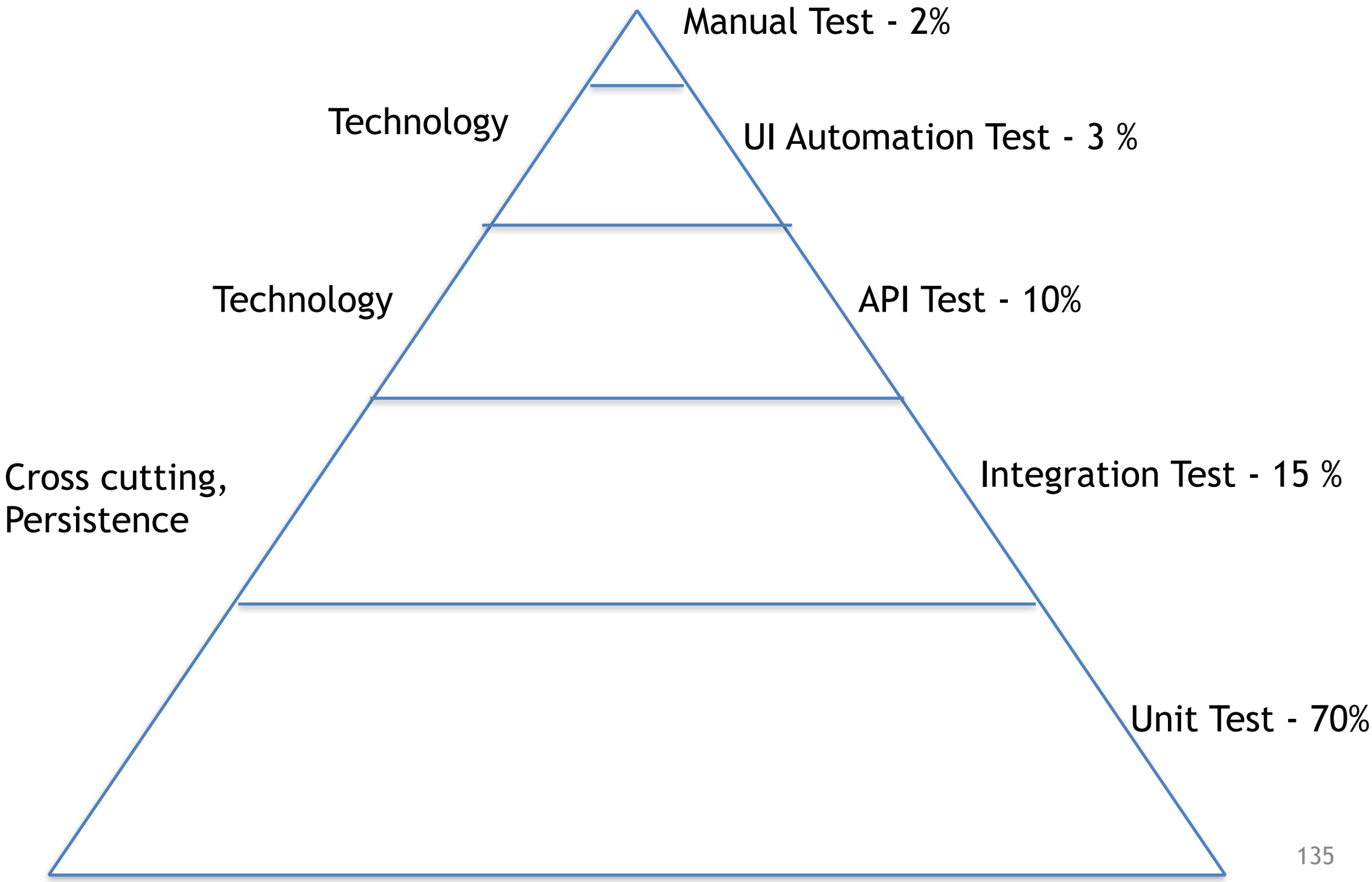
Step 1. Containerizing



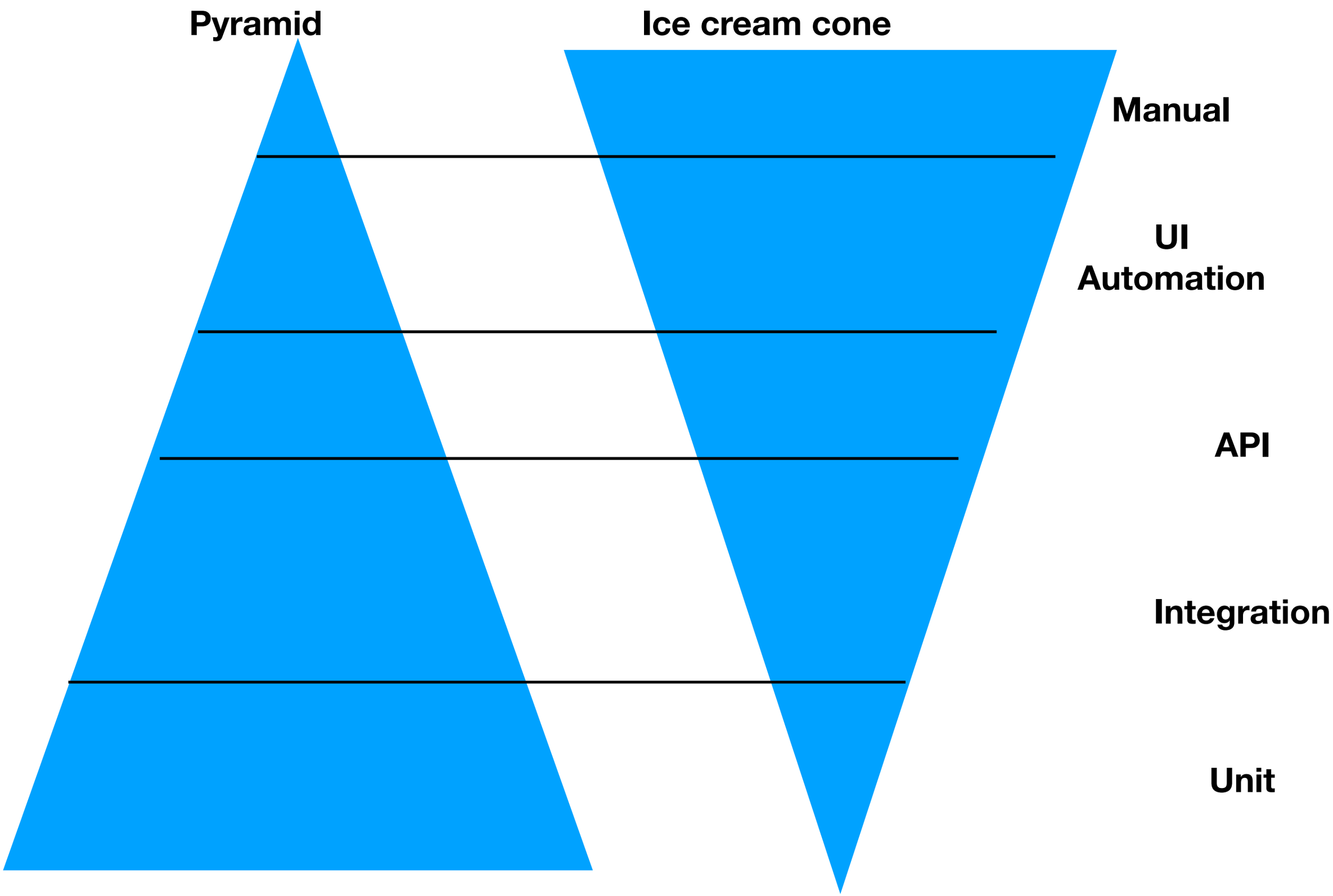
Step 2. Integrating infrastructure automation with CI/CD

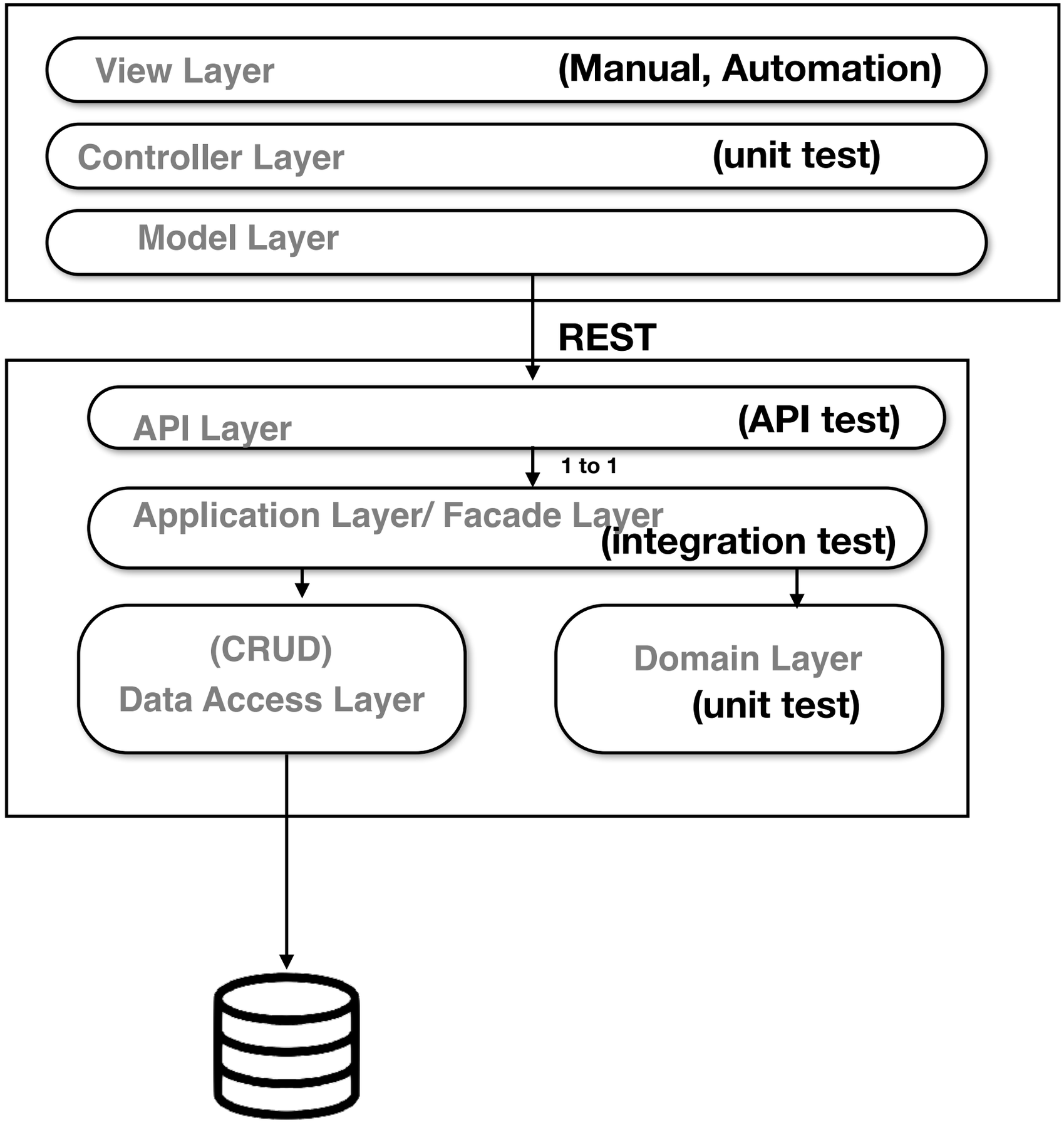


Step 3. test automation and aligning QA with development

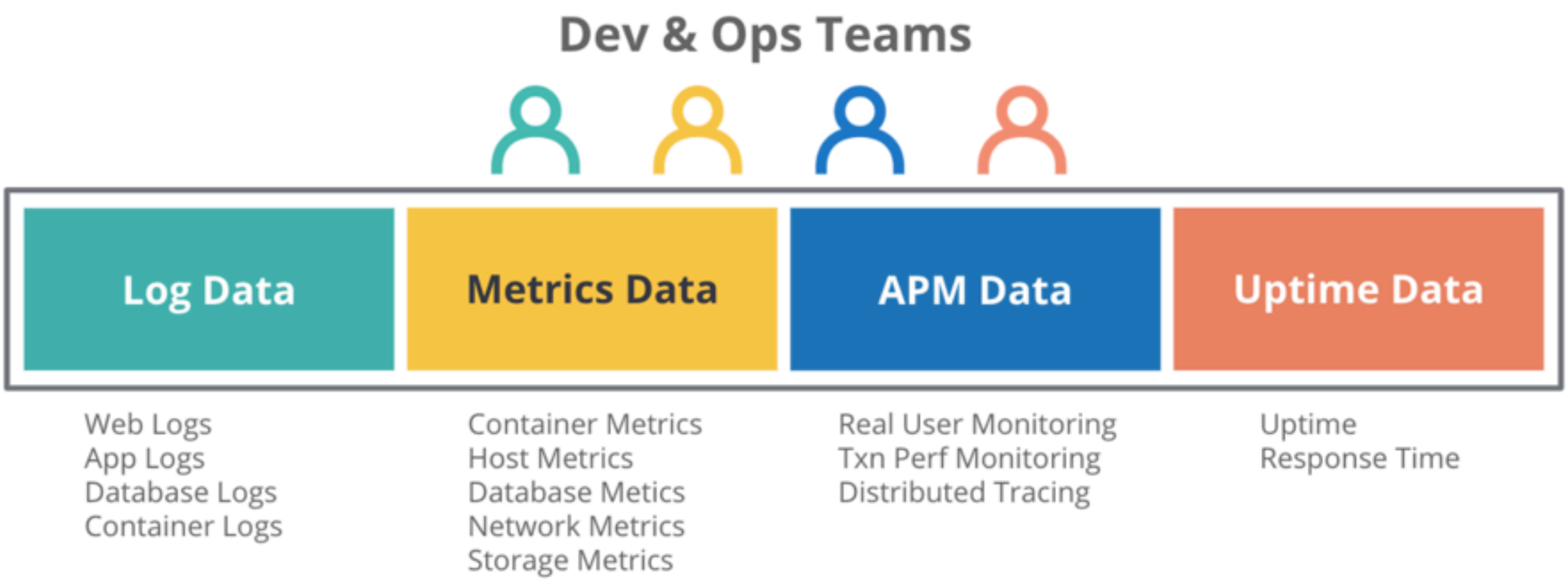


Pyramid Test

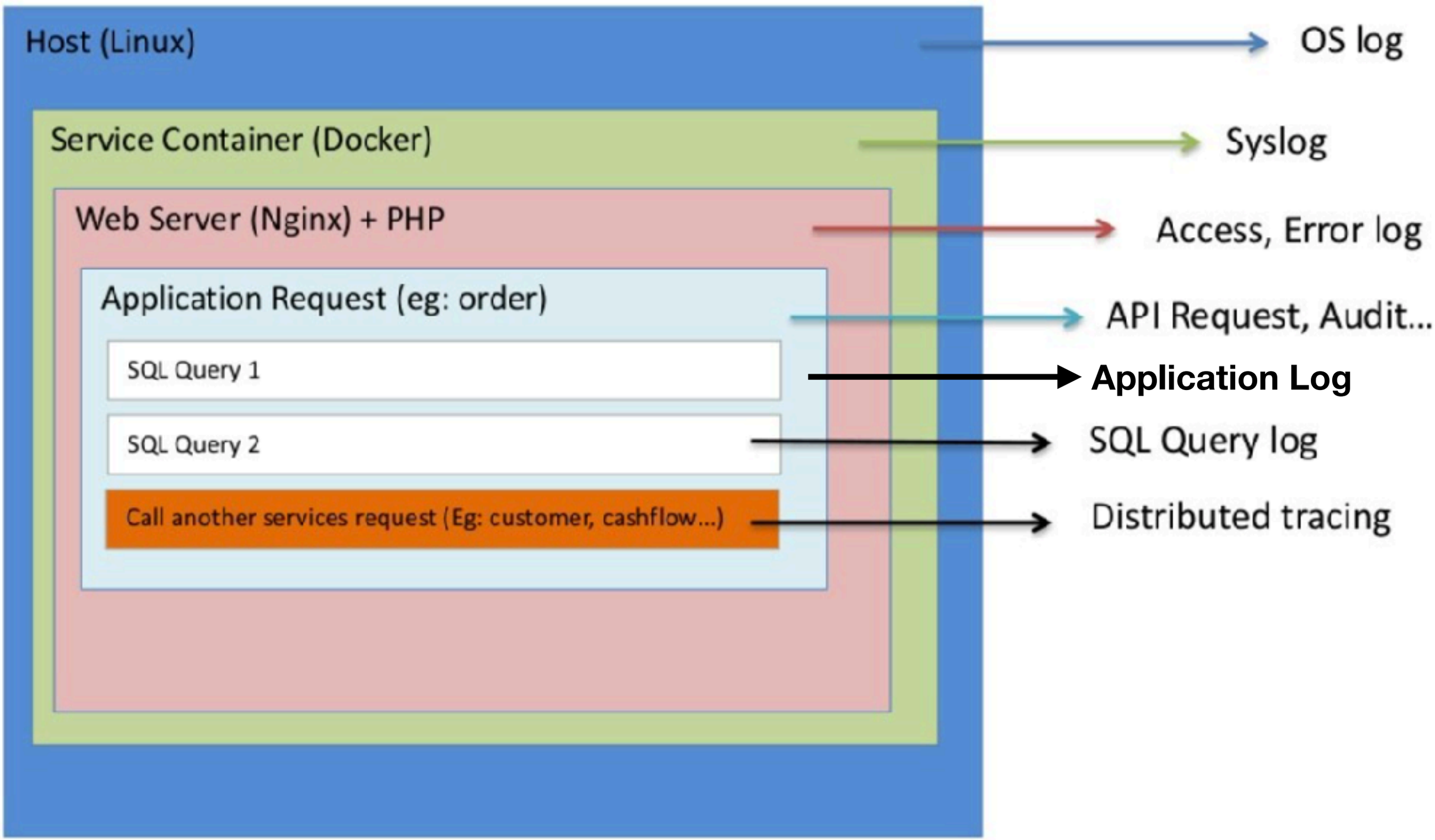


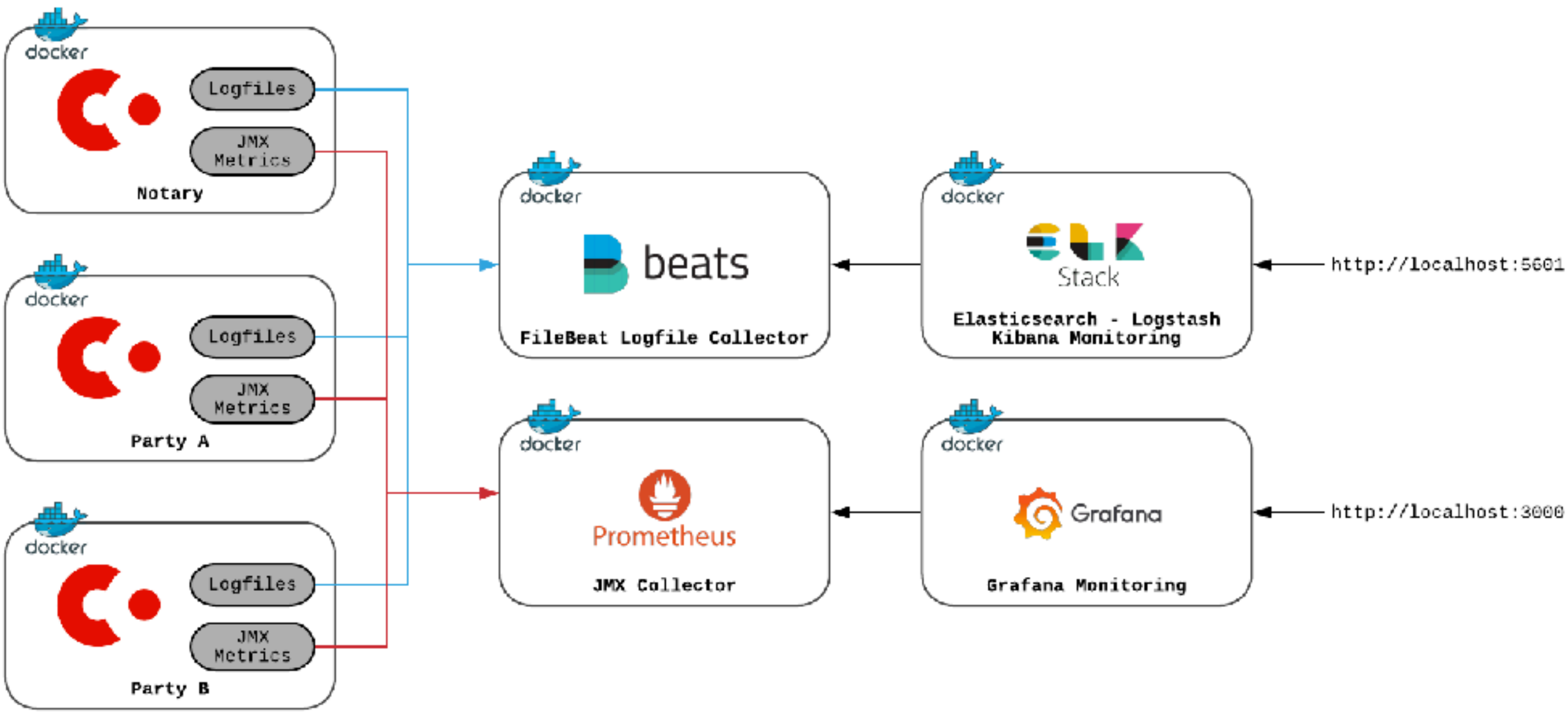

View Layer
(Manual, Automation)
Controller Layer
(unit test)
Model Layer
REST
API Layer
(API test)
1 to 1
Application Layer/ Facade Layer
(integration test)
(CRUD)
Data Access Layer
Domain Layer
(unit test)

Step 4. Setup Monitoring

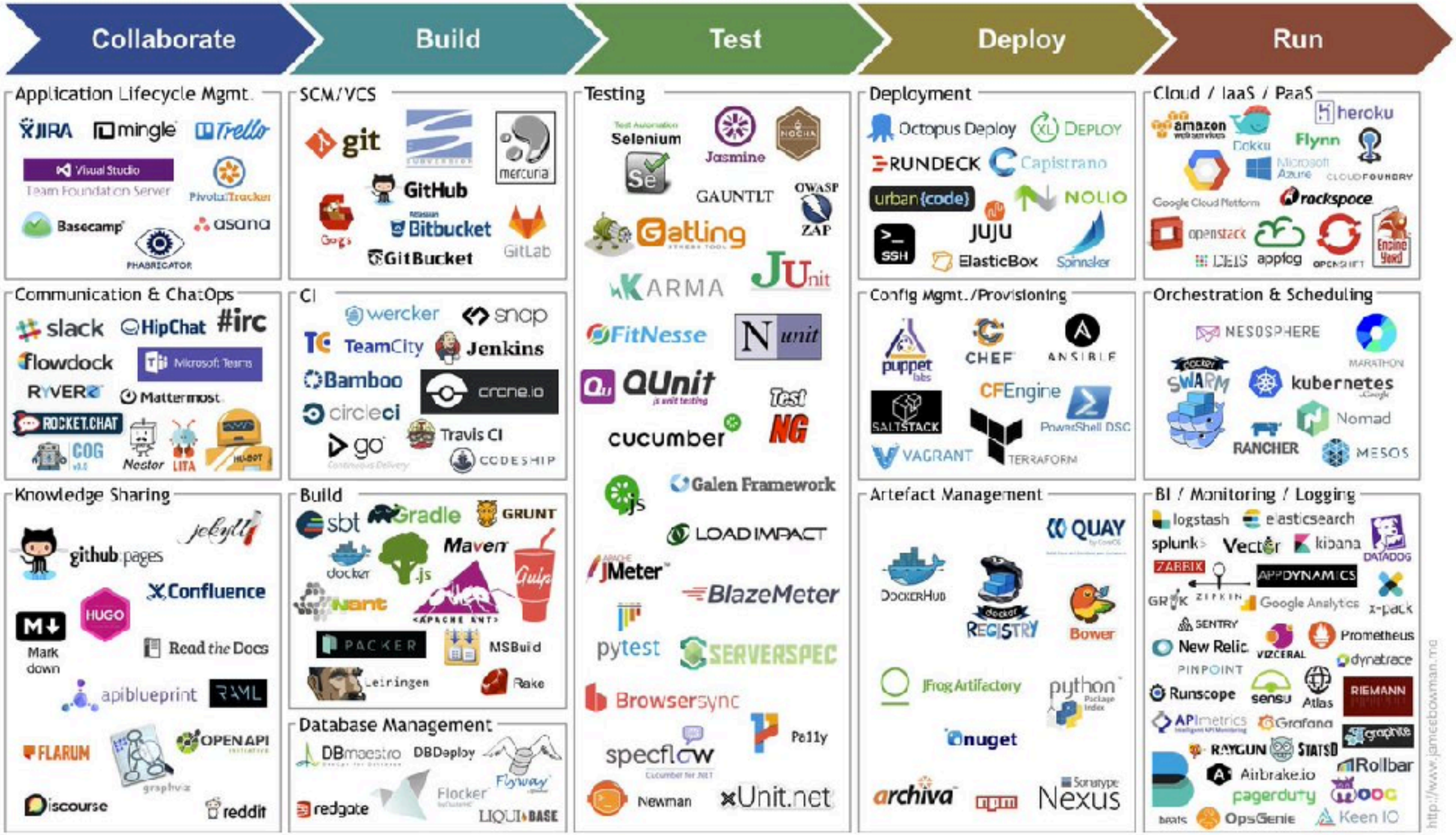


Activity **Resource**

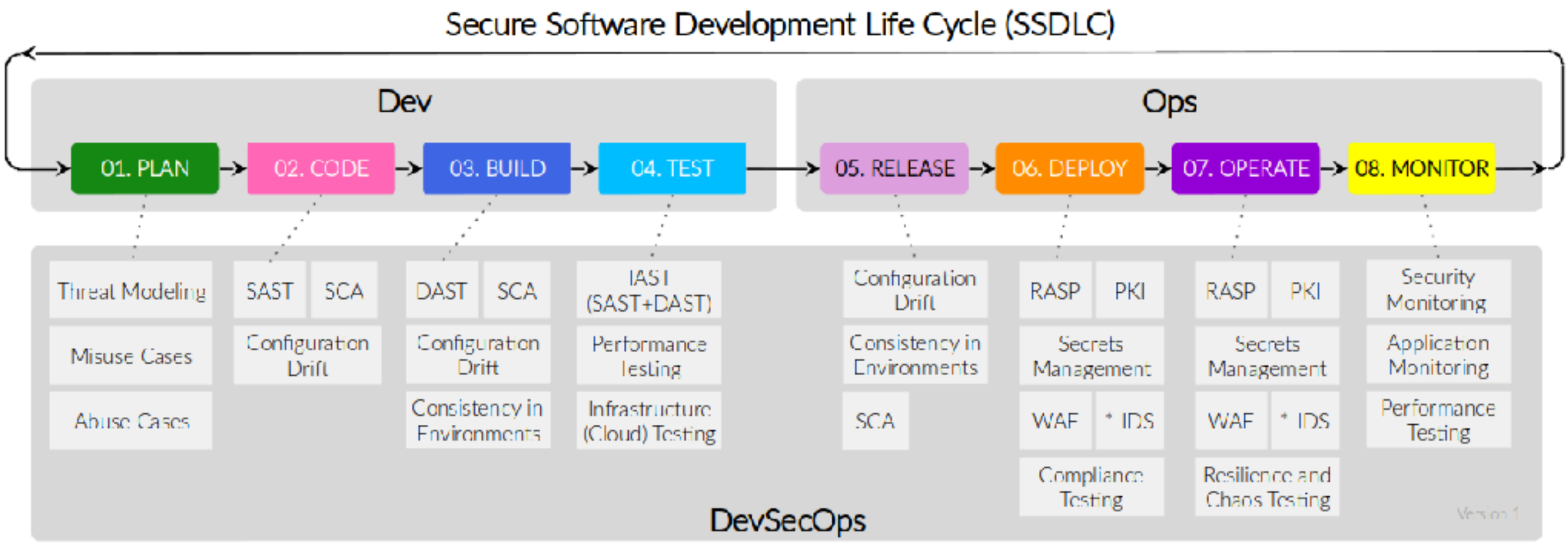

Host (Linux)
Service Container (Docker)
Web Server (Nginx) + PHP
Application Request (eg: order)
SQL Query 1
SQL Query 2
Call another services request (Eg: customer, cashflow...)
OS log
Syslog
Access, Error log
API Request, Audit...
Application Log
SQL Query log
Distributed tracing


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

Reference


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

Step 5. SecOps



DevSecOps Pipeline Techstack

Legend: DevOps DevSecOps

INOVO | VENTURE PARTNERS

CI/CD Pipeline

Commit code to repository

Build

Deploy

IDE Plugins

JFrog, snyk, Skim dev

Pre-Commit Hooks

Talisman, P, git-secrets, HOUND

Secrets Management

HashiCorp Vault, Secret Manager, AWS Secrets Manager

SAST

Checkmarx, DerScanner, ECG

SCA

GitHub, BLACKDUCK, snyk

DAST

RAPID7, acunetix, CONTRAST SECURITY

IAST

CONTRAST SECURITY, SYNOPSYS, VERACODE

Secure Infrastructure as code

clair, anchore, DOCKSCAN

Compliance as code

KitchenCI, SERVERSPEC, CHEF INSPEC

RASP

Signal Sciences, TREND MICRO, imperva

Web Application Firewall

imperva, CLOUDFLARE, wallarm

Monitoring Tools

DATADOG, Nagios, PerfOps

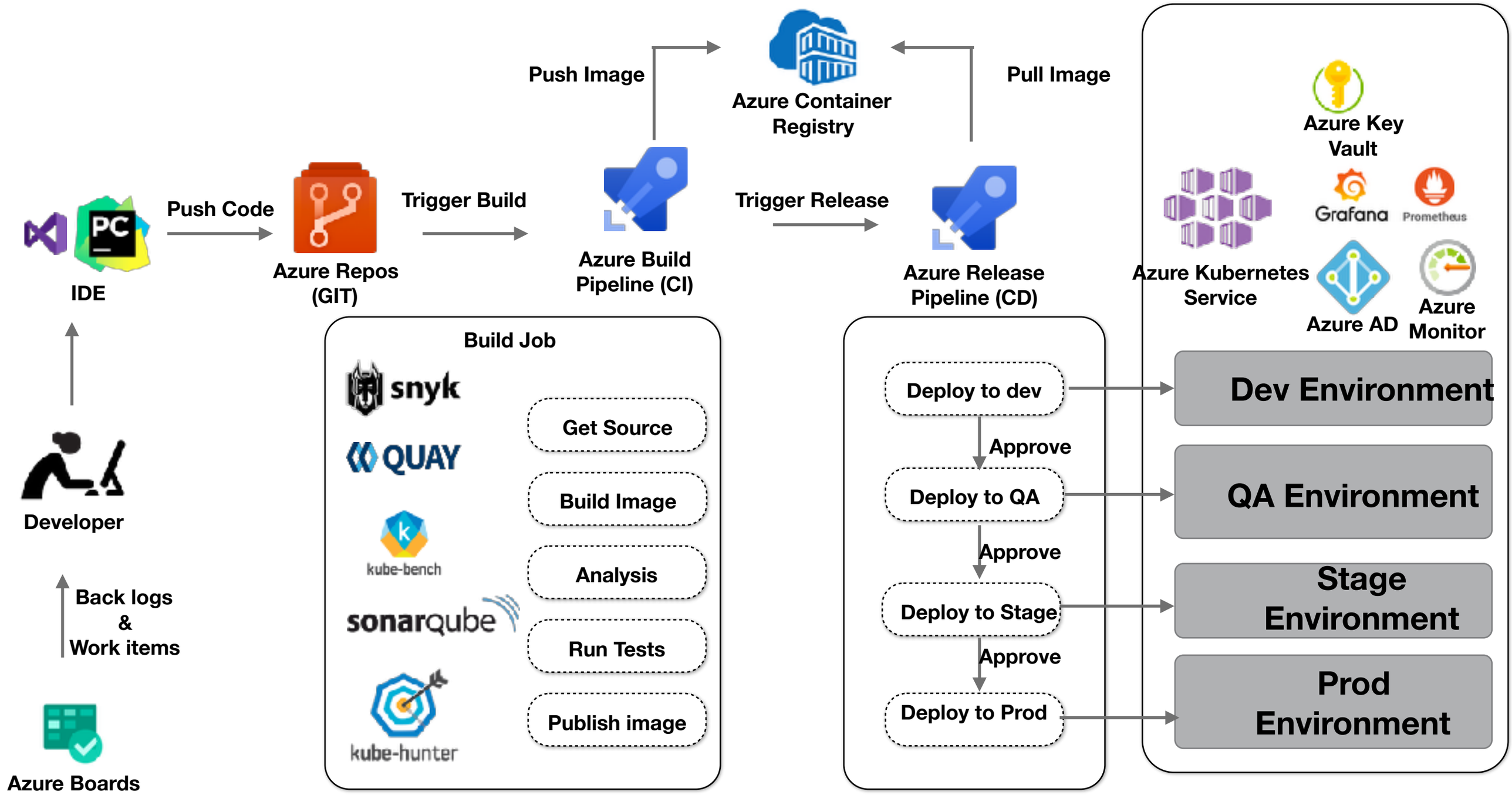
Chaos Engineering

Gremlin, ChaosToolkit

Vulnerability management

Hacker, ARCHERY a security tool, F

Reference model



Cost

consistent workloads

batch jobs

Varying Loads

Varying workloads

Non prod

Reserved VM Instances

1-year 40-45%

3-year 60-65%

Spot Instances

Serverless services

Vendor lock-in
Expensive
cold starts
Debug

Autoscaling

DevTest Labs

Automate Start Stop

Right

Storage Type

Archive

Retention

Policy

Compress

Before Storage

cost of

API access

Data Transfer

Cost

Monitor & Correct Cost Anomalies

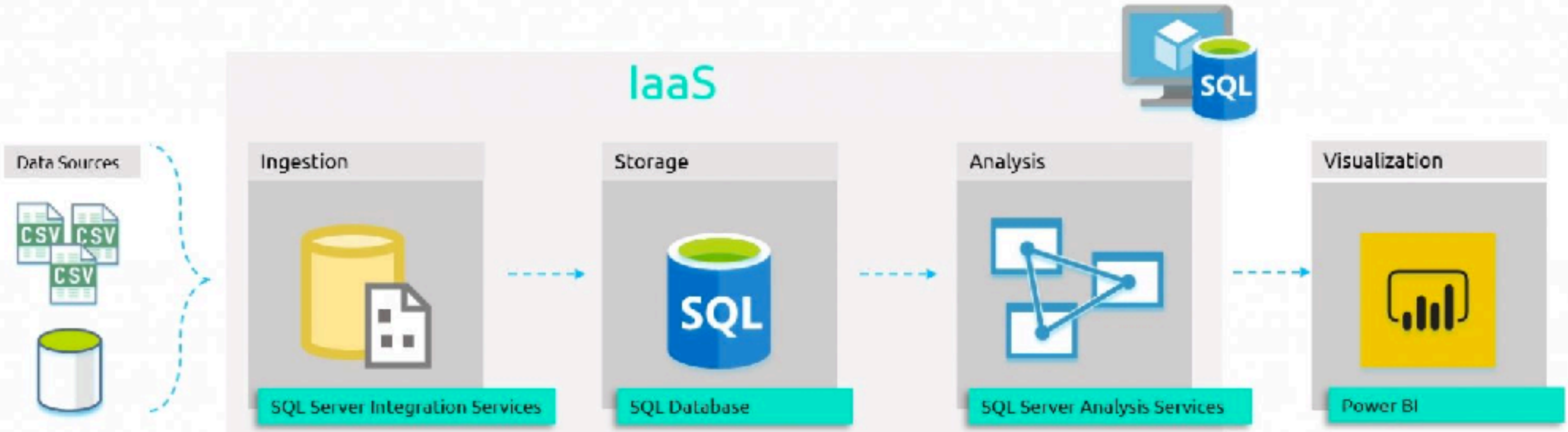
Utilize Heat Maps

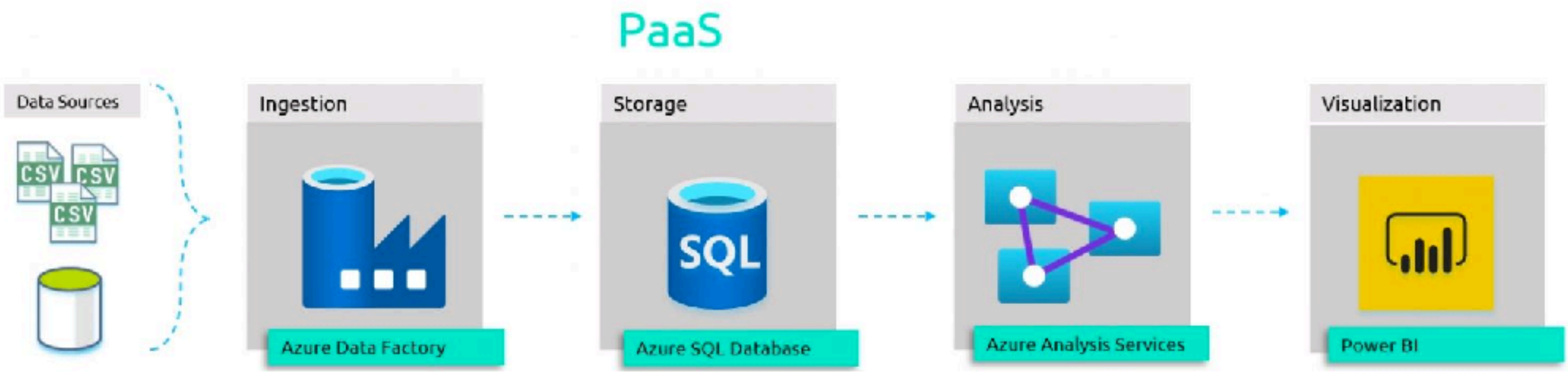
Cost Optimization Tools

Harness

Cloudhealth

Cloudability


IaaS
SQL
Data Sources
CSV
CSV
CSV
Ingestion
SQL Server Integration Services
Storage
SQL
SQL Database
Analysis
SQL Server Analysis Services
Visualization
Power BI


PaaS
Data Sources
CSV
CSV
CSV
Ingestion
Azure Data Factory
Storage
SQL
Azure SQL Database
Analysis
Azure Analysis Services
Visualization
Power BI

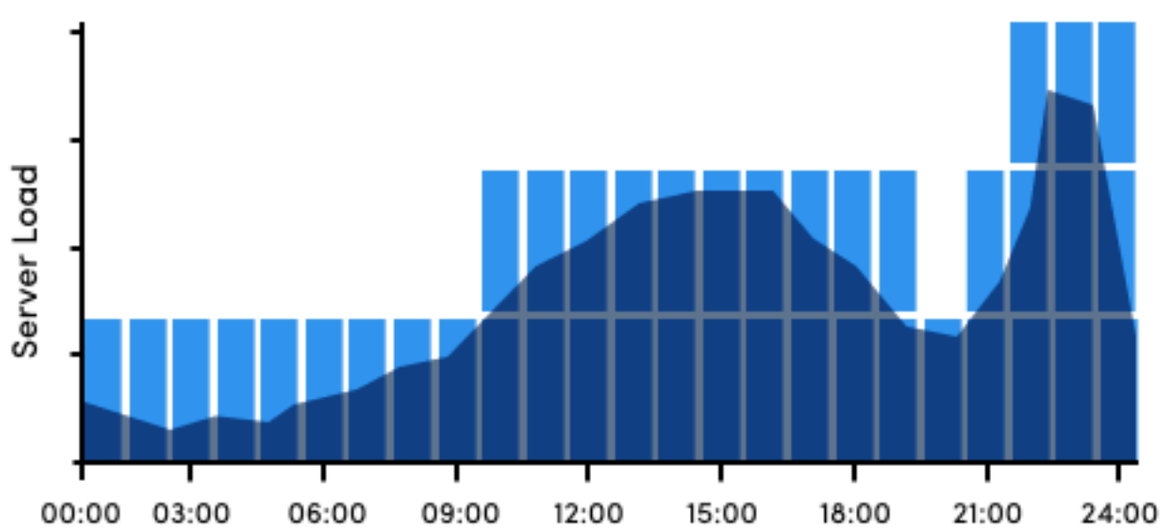
IaaS Architecture			
Service type	**Region**	**Description**	**Estimated monthly cost**
Virtual Machines	UK South	1 DS12 v2 (4 vCPUs, 28 GB RAM) x 730 Hours, Windows – SQL Server, Pay as you go, 2 managed OS disks – S4, 100 transaction units; 5 GB Bandwidth	£554.79
		Total	**£554.79**
PaaS Architecture			
Service type	**Region**	**Description**	**Estimated monthly cost**
Azure SQL Database	UK South	Single Database, DTU Purchase Model, Standard Tier, S1: 20 DTUs, 250 GB included storage per DB, 1 Database(s), x 730 Hours, 5 GB Retention	£27.43
Data Factory	UK South	Azure Data Factory V2 Type, Data Pipeline Service Type, Azure Integration Runtime: 30 Activity Run(s), 600 Data movement unit(s), 200 Pipeline activities, 0 Pipeline activities – External,	£134.90
Azure Analysis Services	UK South	Standard S0 (Hours), 1 Instance(s), 360 Hours	£217.33
		Total	**£379.66**
Disclaimer			

All prices shown are in British Pound (£). This is a summary estimate, not a quote. For up to date pricing information please visit https://azure.microsoft.com/pricing/calculator/

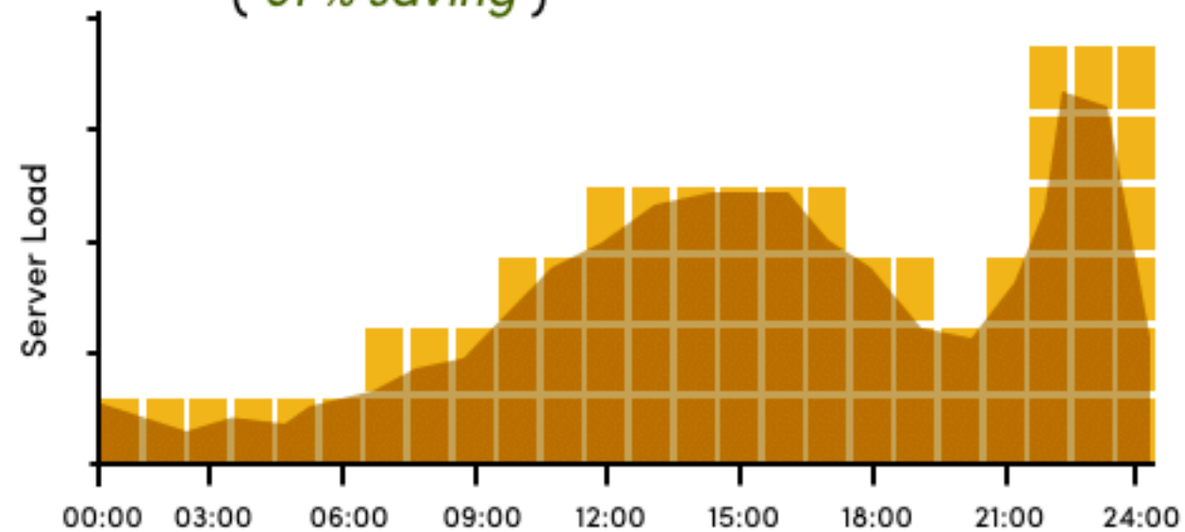
This estimate was created at 11/8/2020 5:25:55 PM UTC.

Downsize under-utilized instances

41 Instance Hours
m4.large@ $0.1/hr
= $4.1/day



70 Instance Hours
t2.small@ $0.023/hr
= $1.61/day
(*61 % saving*)



m4.large is utilized less than 50% between 0 to 9 am. Server usage increases from 9 am to 2 pm. Here it requires two m4.large instances. Again, usage starts decreasing till 6 pm and increases in the night.

Don't use serverless for constant, heavy loads. Serverless can be up to 3 times more expansive to operate than a dedicated server for such workloads.

Set an expiry limit for all CloudWatch logs. The default is to never expire.

A dual core VM on Azure with 4GB of RAM costs about $100 a month on Azure if you used it 24/7. $1,200 a year .

There are 168 hours in a week. If you used your VM for 50 hours a week for work, that is **less than $400 a year**.

an AWS On-Demand M4.large is $73/month in us-east-1 and $91/month in ap-southeast-2. Choose the cheapest region that is closest to your customers.

Reduce cost on API access

PUT calls cost $0.005/1000 calls. So to upload a 10GB file in 5MB chunks it will cost roughly $0.01. Whereas, for 10KB file chunks, it will cost approximately $5.00.

Minimizing data transfer costs

Reduce cost on API access

Delete or migrate unwanted files after a certain date

Automate Infrastructure

Use optimization tools

Pause Your Idle Clusters when Not in Use

Set up budgets and allocate costs to teams and projects

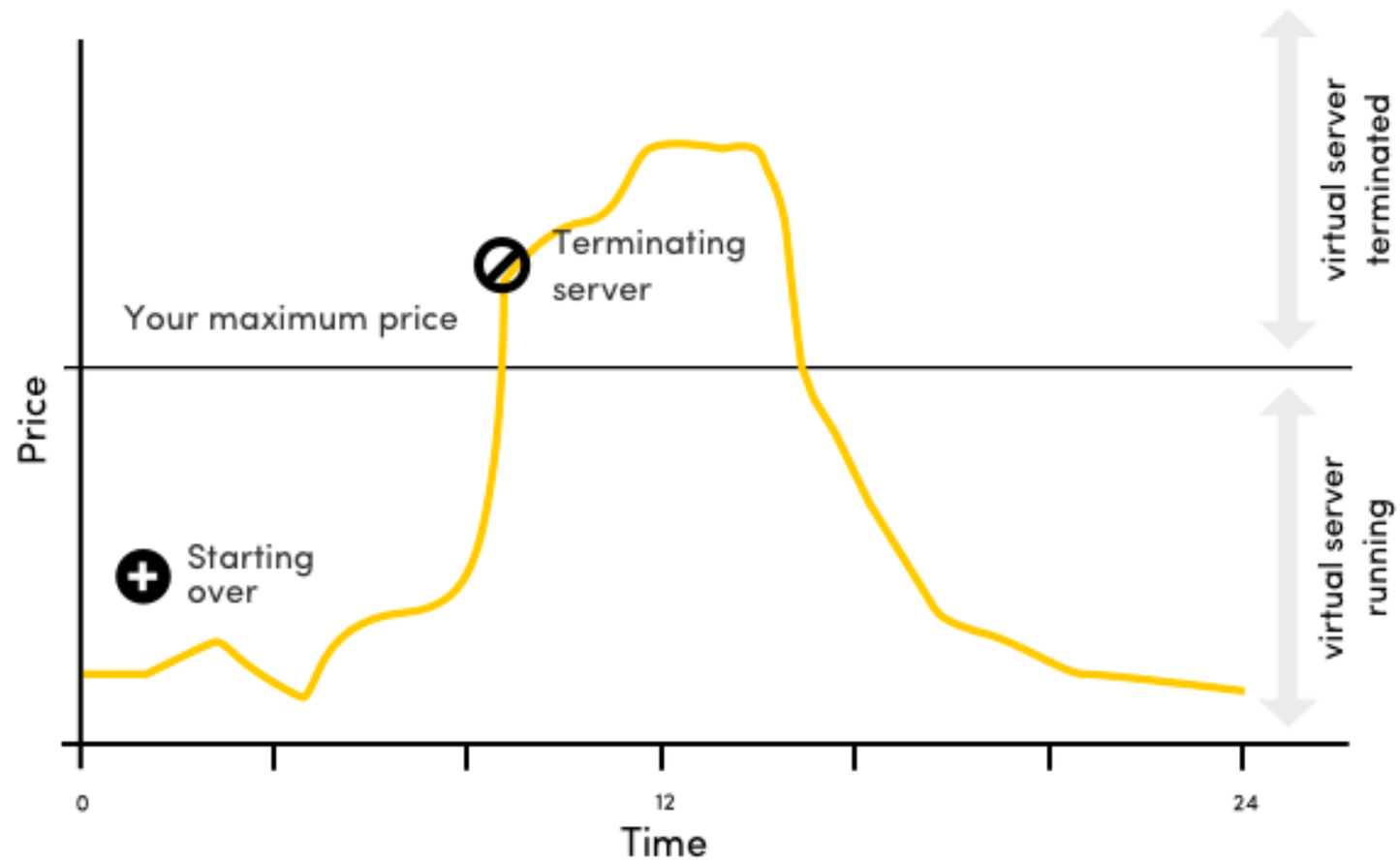
- **Hot –** Used for regular access but has the highest storage cost and lowest access cost
- Cool – Lower storage costs, higher access costs and used for less frequent access
- Archive – Lowest storage cost and has the highest access cost

- **Dev Test Labs:** Use this feature when you need to use Virtual Machines for your Developers or Test engineers, as this VM has features like auto start/shutdown features where you can save compute cost when not in use.

- **Zone 1**: US, US Gov, Canada, Europe, UK, France, and Switzerland
- **Zone 2:** Asia, Australia, India, Japan, and Korea
- **Zone 3:** Brazil, South Africa, and UAE
- **DE Zone 1**: Germany

DE Zone 1 has the lowest egress charges and Zone 1 is the cheapest public zone.

Whereas Zone 2 and 3 each are more expensive respectively than other zones. So, for e.g., if we have vNet peering between the US in Zone 1 and Asia in Zone 2, so traffic leaving from US vNet is charged according to Zone 1 rates and traffic entering Asia vNet is charged as per Zone 2 rates and vice versa


Terminating
server
Your maximum price
Starting
over
Price
virtual server
terminated
virtual server
running
0
12
24
Time

Part 3
Arch Risks

	Risk	I	L	R	C	Notes
1	Performance of the View Todo is significantly impacted by large data volumes.	8	4	32	High	Testing to be undertaken to gauge impact.
2	Limitations of JPA API result in complex workarounds in repository implementation or workarounds in other architectural layers.	6	4	24	Medium	Can fall back to Hibernate which provides richer ORM functionality.
3	The Open Source tools and components selected lack the capabilities of equivalent commercial products limiting the features that can be implemented (e.g. GIS components).	6	3	18	High	Investigate alternative products.

- Risk – The description of the architectural risk.
- Impact to Project – The impact the risk will have on the project
    - (1 = Minor impact, 10 = Showstopper)
- Likelihood – The likelihood of the risk eventuating (1 = Low, 953 = Highly likely)
- Rating – The overall rating for the risk based on Impact * Likelihood (1 = Minor, 90 = Critical)
- Confidence – The confidence in resolving the issue should it eventuate (Low, Medium, High)

- ATAM
- ARID
- SAAM
-

Evaluate Architecture (ATAM)

identify all Architectural approaches

A1 (CQRS)
A2(Cube)
A3 (Caching)
...

identify all quality requirements

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

analyse Scenario -> Approach

S1 -> A1, A2
S2 -> A6,A8, A9
S3-> ?
S4 -> A6, ?

=> Risk & trade off's

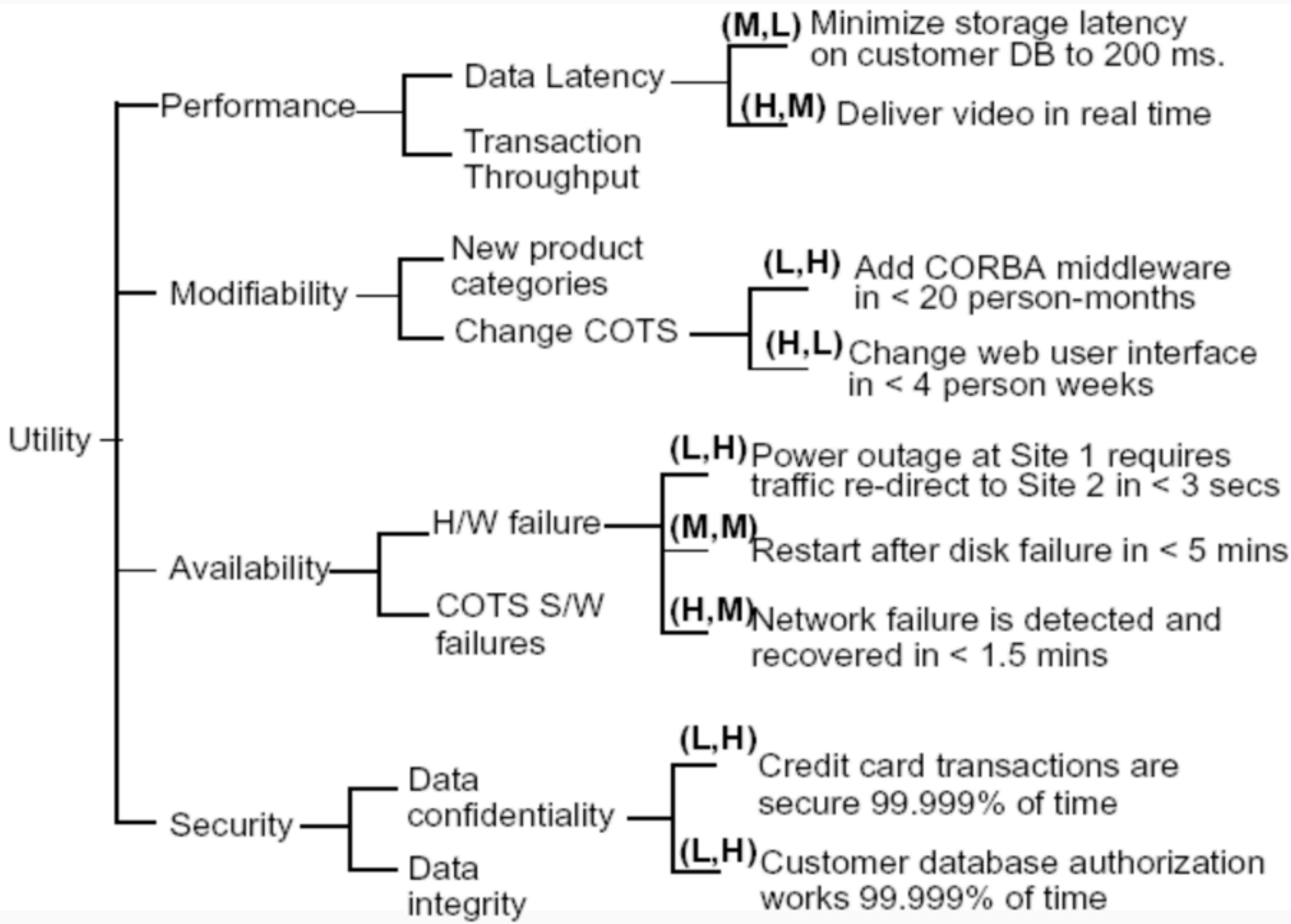
brainstorm for scenarios

s8
s9
s10
...

analyse Scenario -> Approach

S8 -> A1, A2
S9 -> A6,A8, A9
S10-> ?

=> Risk & trade off's


Utility
Performance
Data Latency
(M,L) Minimize storage latency on customer DB to 200 ms.
(H,M) Deliver video in real time
Transaction Throughput
Modifiability
New product categories
Change COTS
(L,H) Add CORBA middleware in < 20 person-months
(H,L) Change web user interface in < 4 person weeks
Availability
H/W failure
(L,H) Power outage at Site 1 requires traffic re-direct to Site 2 in < 3 secs
(M,M) Restart after disk failure in < 5 mins
(H,M) Network failure is detected and recovered in < 1.5 mins
COTS S/W failures
Security
Data confidentiality
(L,H) Credit card transactions are secure 99.999% of time
Data integrity
(L,H) Customer database authorization works 99.999% of time

Summary

Engineering for Performance

Performance Objectives
(Response Time, Throughput, Resource Utilization, Workload)

Performance Modeling
(Scenarios, Objectives, Workloads, Requirements, Budgets, Metrics)

Architecture and Design Guidelines
(Principles, Practices and Patterns)

Performance and Scalability Frame

Coupling and Cohesion	Resource Management
Communication	Caching, State Management
Concurrency	Data Structures / Algorithms

Measuring, Testing, Tuning

Measuring	Testing	Tuning
Response Time	Load Testing	Network
Throughput	Stress Testing	System
Resource Utilization	Capacity Testing	Platform
Workload		Application

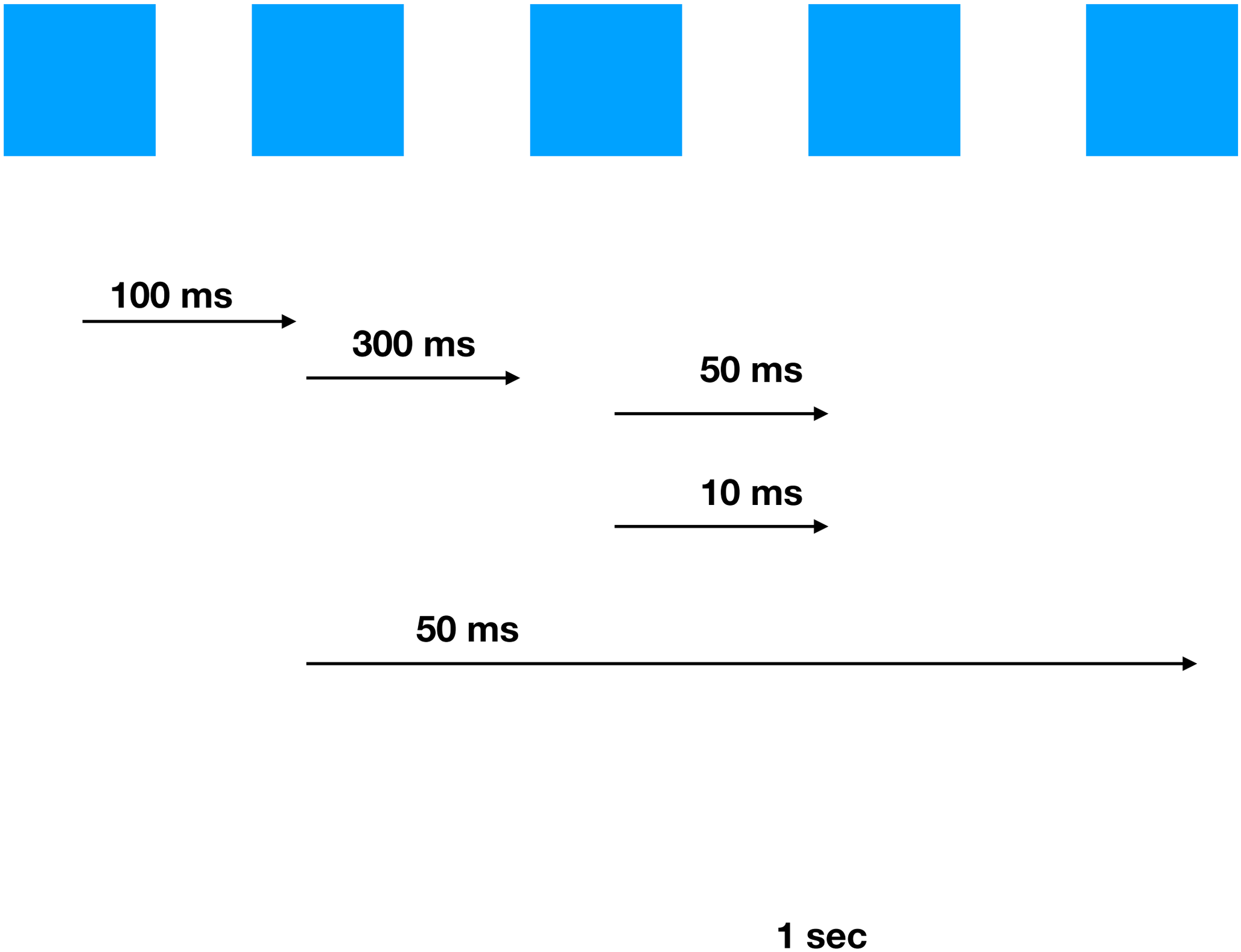
Roles
(Architects, Developers, Testers, Administrators)

Life Cycle
(Requirements, Design, Develop, Test, Deploy, Maintain)

Performance Modeling Process

1. Identify Key Scenarios
2. Identify Workloads
3. Identify Performance Objectives
4. Identify Budget
5. Identify Processing Steps
6. Allocate Budget
7. Evaluate
8. Validate

Itera


100 ms
300 ms
50 ms
10 ms
50 ms
1 sec

Performance tactics Check list

Reduce network I/O

Reduce disk I/O

Compression / minify

Chunky call (batch)

Caching (database, in memory, distributed, client)

Lazy Loading

Eager Loading

Parallel

Scalability tactics Check list

Stateless

Vertical Scaling / Scale up (h/w upgrade)

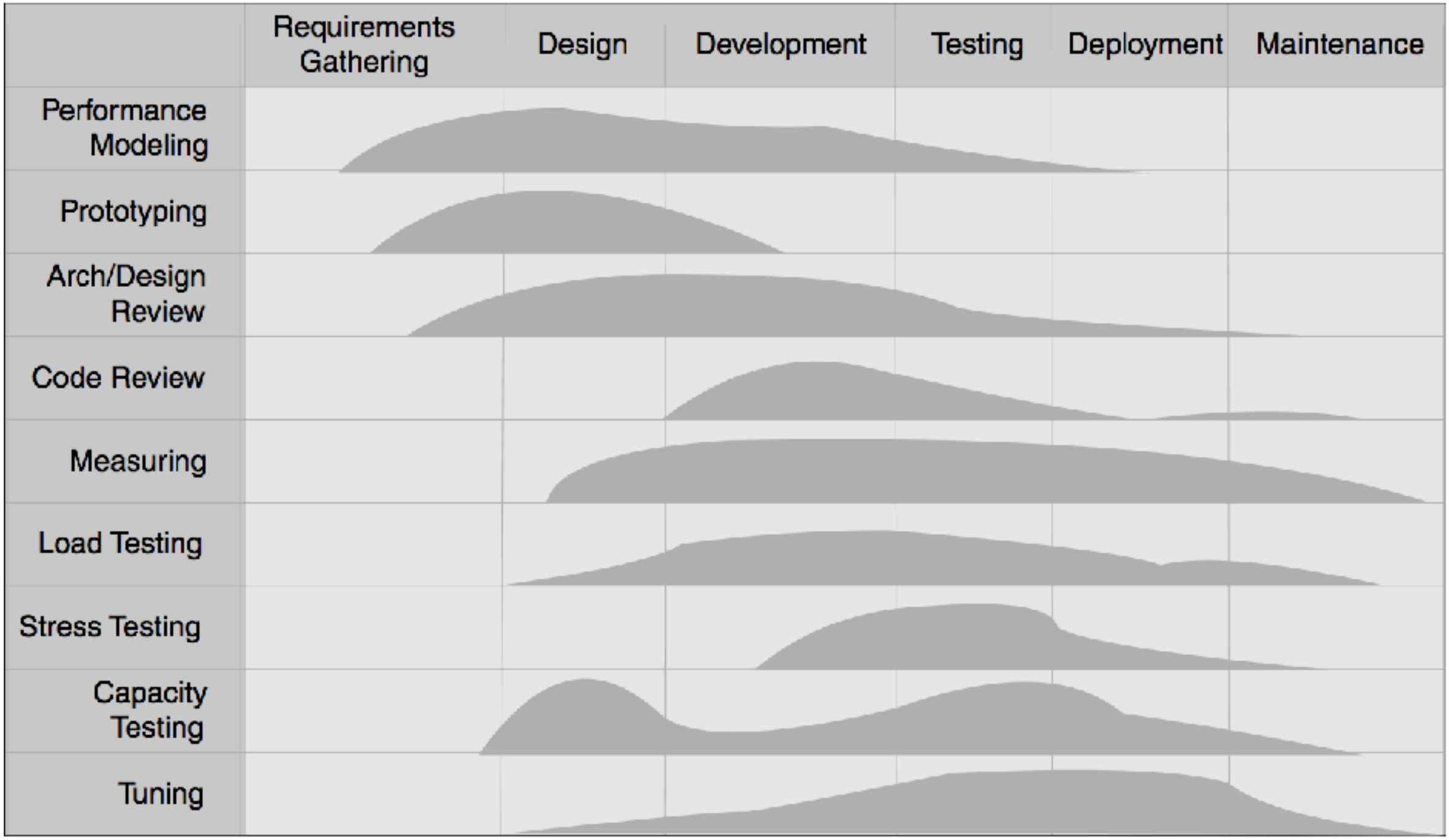
Clone / Scale Out (Honrizonal Scaling) / Auto Scaling

Splitting (Vertical Slicing)

Sharding / Parititioning (Horizontal Slicing)

Async / Messaging/ EDA

Caching

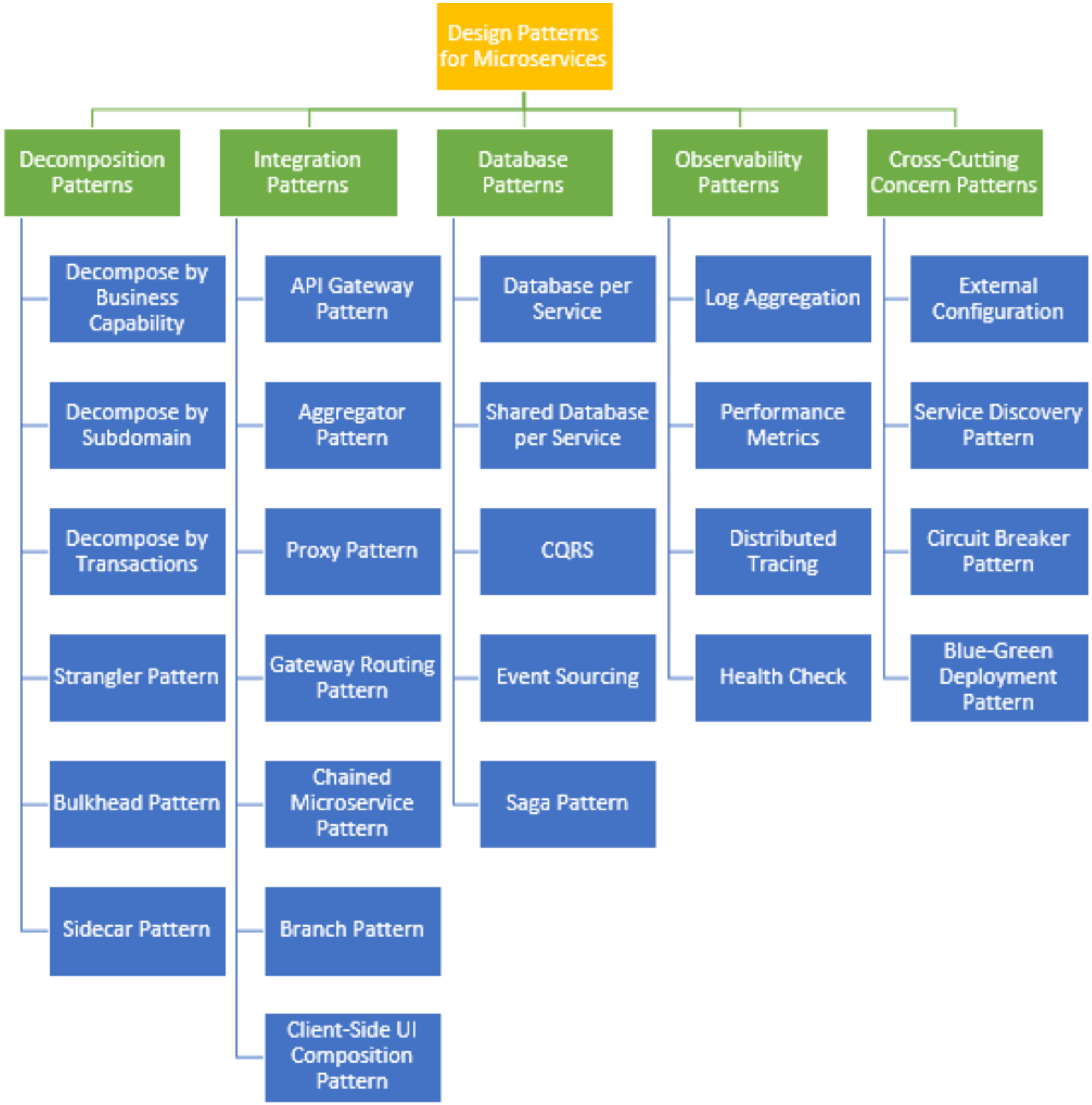

Requirements Gathering
Design
Development
Testing
Deployment
Maintenance
Performance Modeling
Prototyping
Arch/Design Review
Code Review
Measuring
Load Testing
Stress Testing
Capacity Testing
Tuning

Threat Modeling Method	Features
STRIDE	• Helps identify relevant mitigating techniques • Is the most mature • Is easy to use but is time consuming
PASTA	• Helps identify relevant mitigating techniques • Directly contributes to risk management • Encourages collaboration among stakeholders • Contains built-in prioritization of threat mitigation • Is laborious but has rich documentation
LINDDUN	• Helps identify relevant mitigation techniques • Contains built-in prioritization of threat mitigation • Can be labor intensive and time consuming
CVSS	• Contains built-in prioritization of threat mitigation • Has consistent results when repeated • Has automated components • Has score calculations that are not transparent
Attack Trees	• Helps identify relevant mitigation techniques • Has consistent results when repeated • Is easy to use if you already have a thorough understanding of the system
Persona non Grata	• Helps identify relevant mitigation techniques • Directly contributes to risk management • Has consistent results when repeated • Tends to detect only some subsets of threats
Security Cards	• Encourages collaboration among stakeholders • Targets out-of-the-ordinary threats • Leads to many false positives
hTMM	• Contains built-in prioritization of threat mitigation • Encourages collaboration among stakeholders • Has consistent results when repeated
Quantitative TMM	• Contains built-in prioritization of threat mitigation • Has automated components • Has consistent results when repeated
Trike	• Helps identify relevant mitigation techniques • Directly contributes to risk management • Contains built-in prioritization of threat mitigation • Encourages collaboration among stakeholders • Has automated components • Has vague, insufficient documentation
VAST Modeling	• Helps identify relevant mitigation techniques • Directly contributes to risk management • Contains built-in prioritization of threat mitigation • Encourages collaboration among stakeholders • Has consistent results when repeated • Has automated components • Is explicitly designed to be scalable • Has little publicly available documentation
OCTAVE	• Helps identify relevant mitigation techniques • Directly contributes to risk management • Contains built-in prioritization of threat mitigation • Encourages collaboration among stakeholders • Has consistent results when repeated • Is explicitly designed to be scalable • Is time consuming and has vague documentation

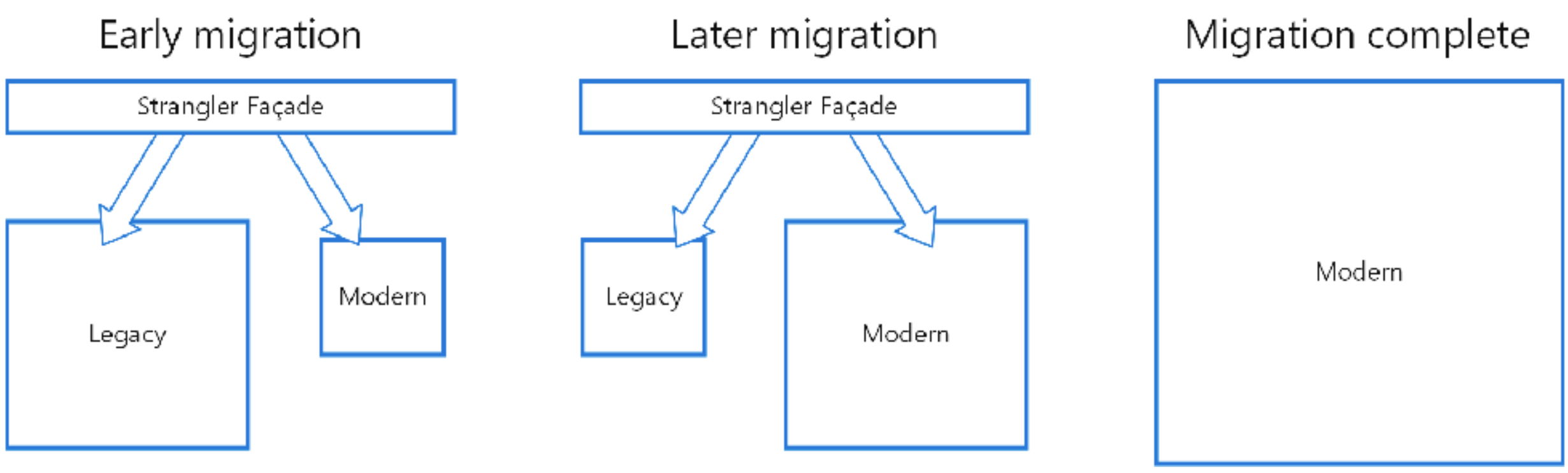
Security

- Authentication (First Defense) “who are you”
- Authorization “what can you do”
- Audit (Last Defense) “what did you do”
- Data Security
  - In Transit
  - In Rest
- Input Validations
- Exception management
- Session management
- Secret management

Microservice Patterns

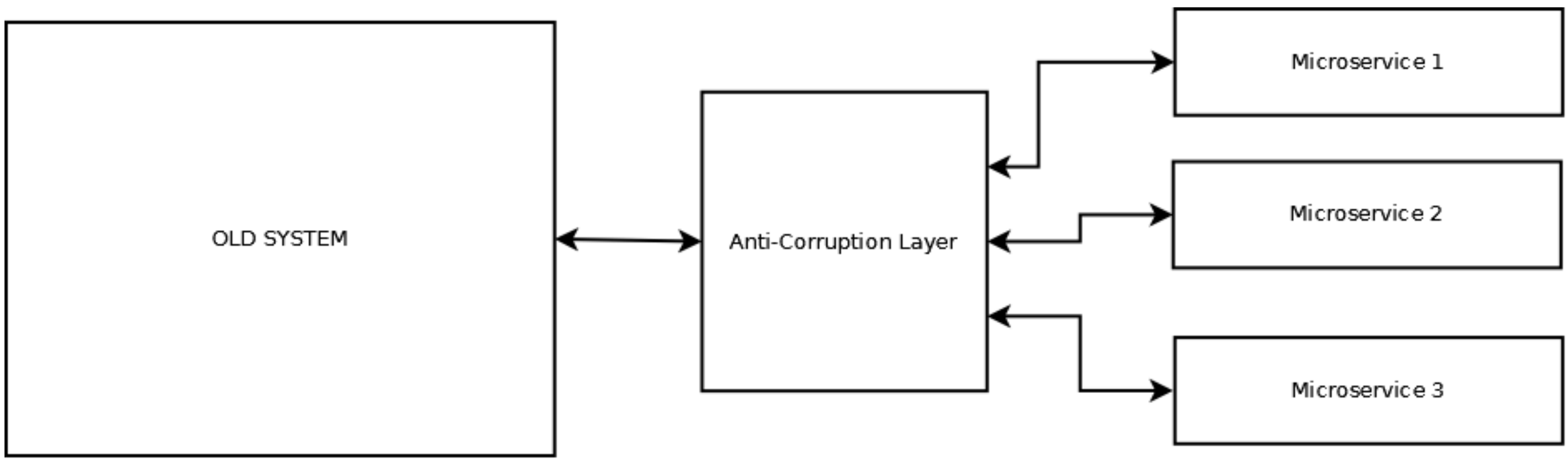


Strangler pattern



- Create a façade that intercepts requests going to the backend legacy system. Over time, as features are migrated to the new system, the legacy system is eventually "strangled" and is no longer necessary.

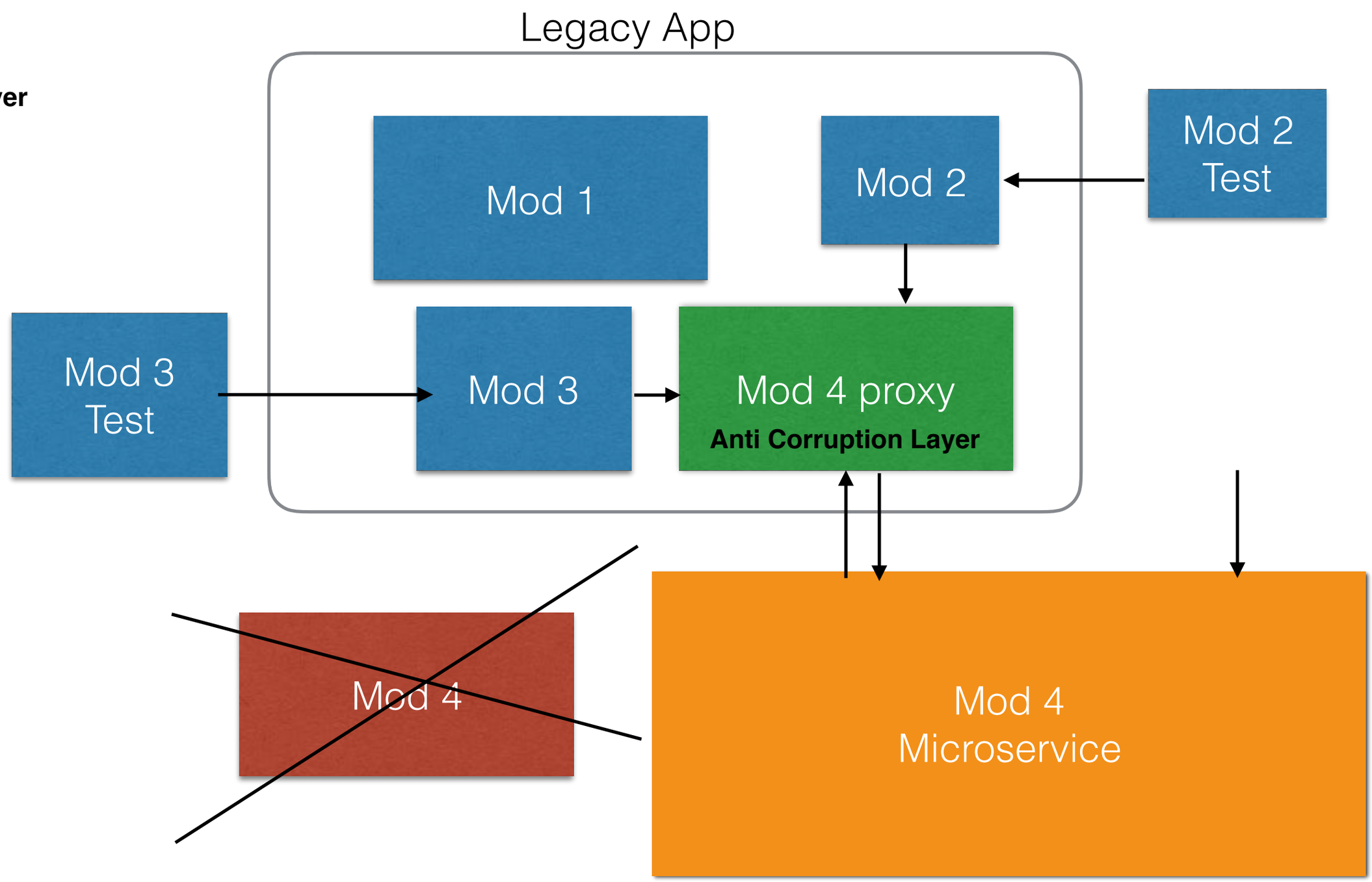
Anti-Corruption Layer

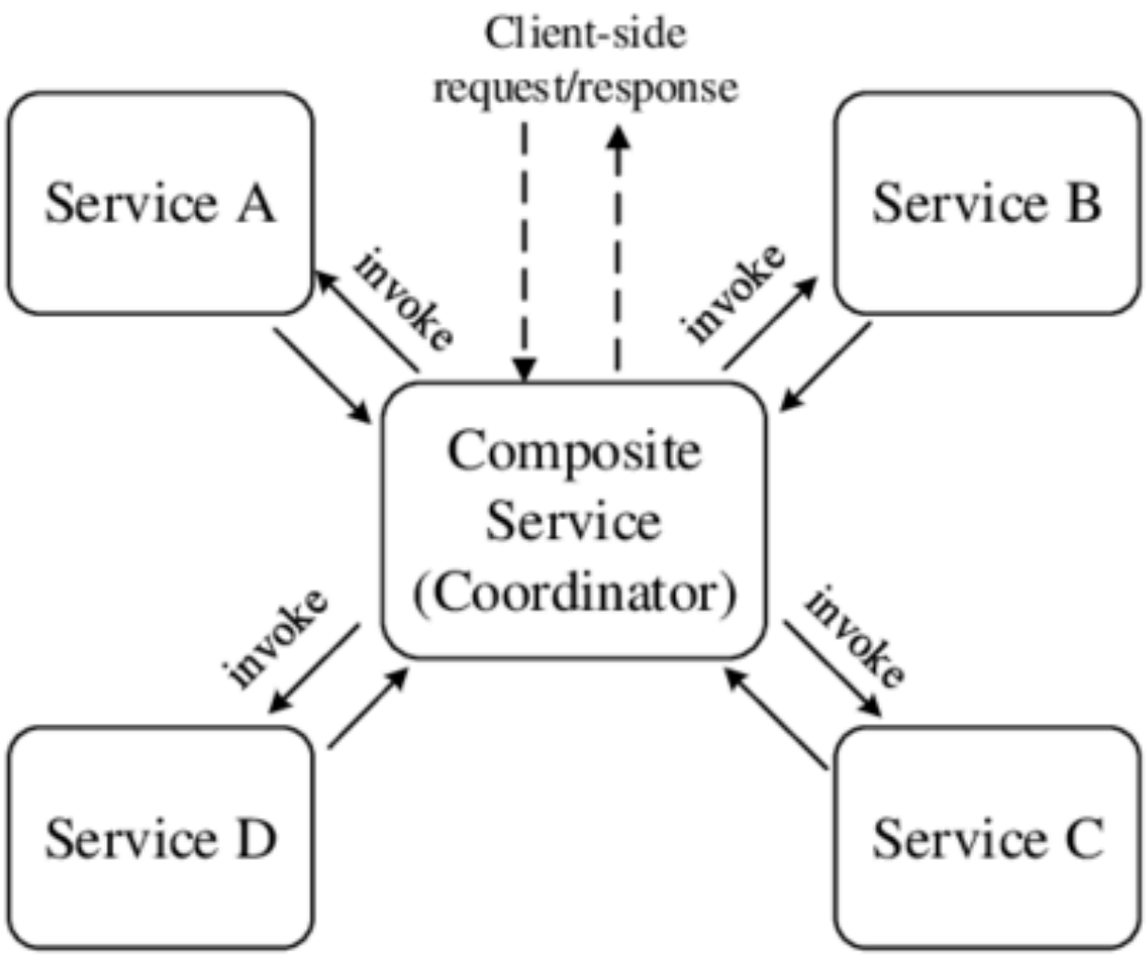


- When you create a new microservice, you notice that some business or technical assumptions or flow should be remodeled. instead of a Big Rewrite ACL maps one domain onto another so that services that use second domain do not have to be "corrupted" by concepts from the first.

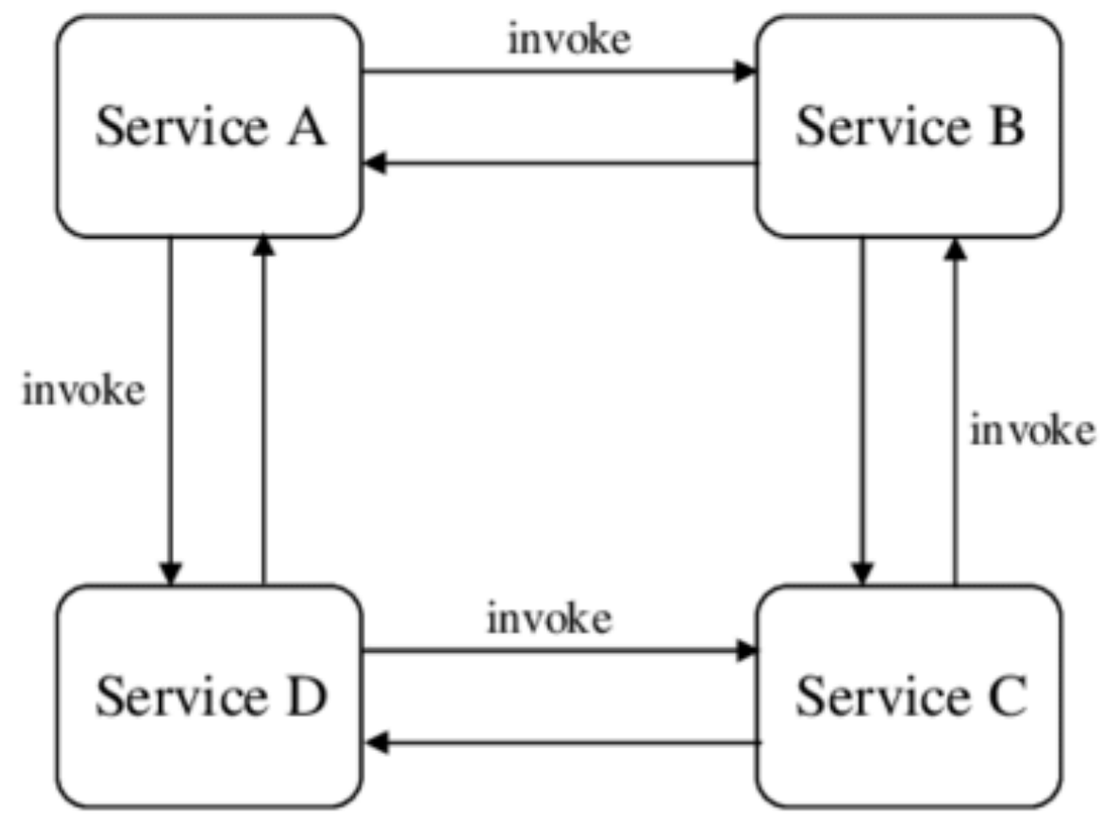
- Such modifications may result in changes to the events that a bounded context publishes.

unit test
Strangler Pattern
Anti Corruption Layer



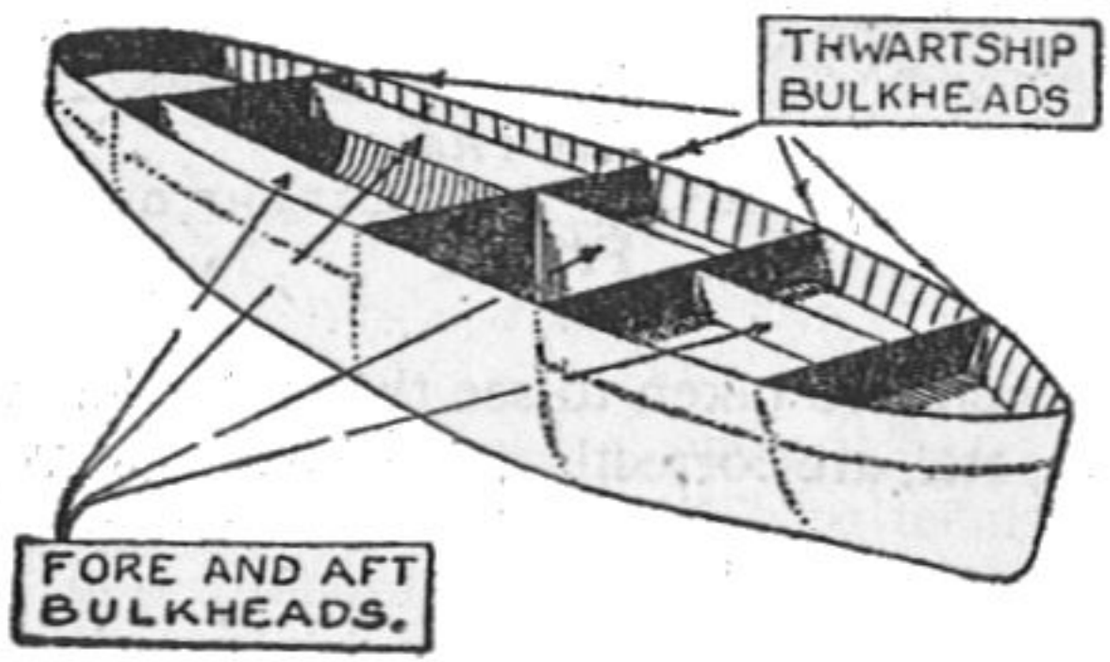


(a) Web Service Orchestration



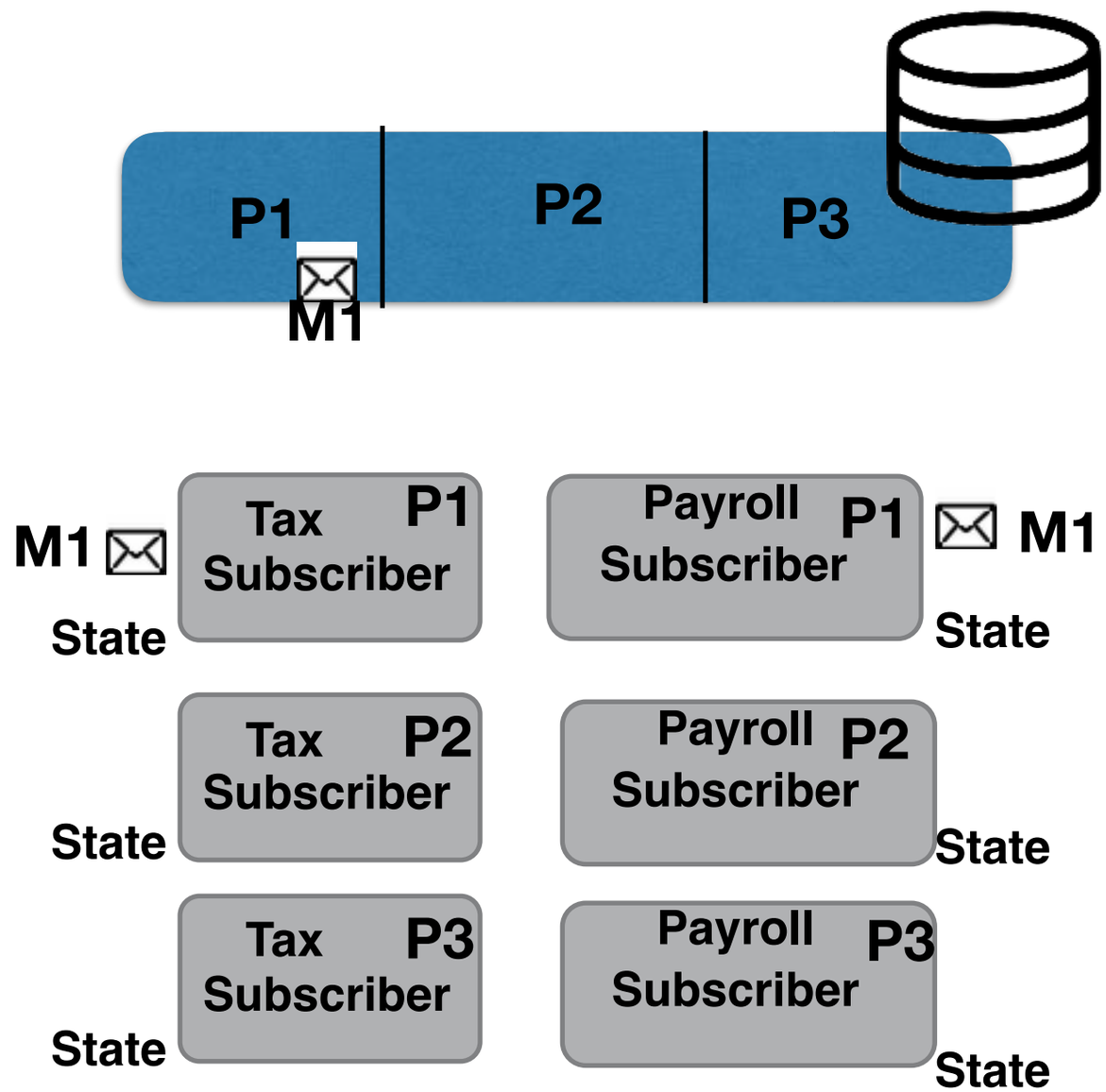
(b) Web Service Choreography

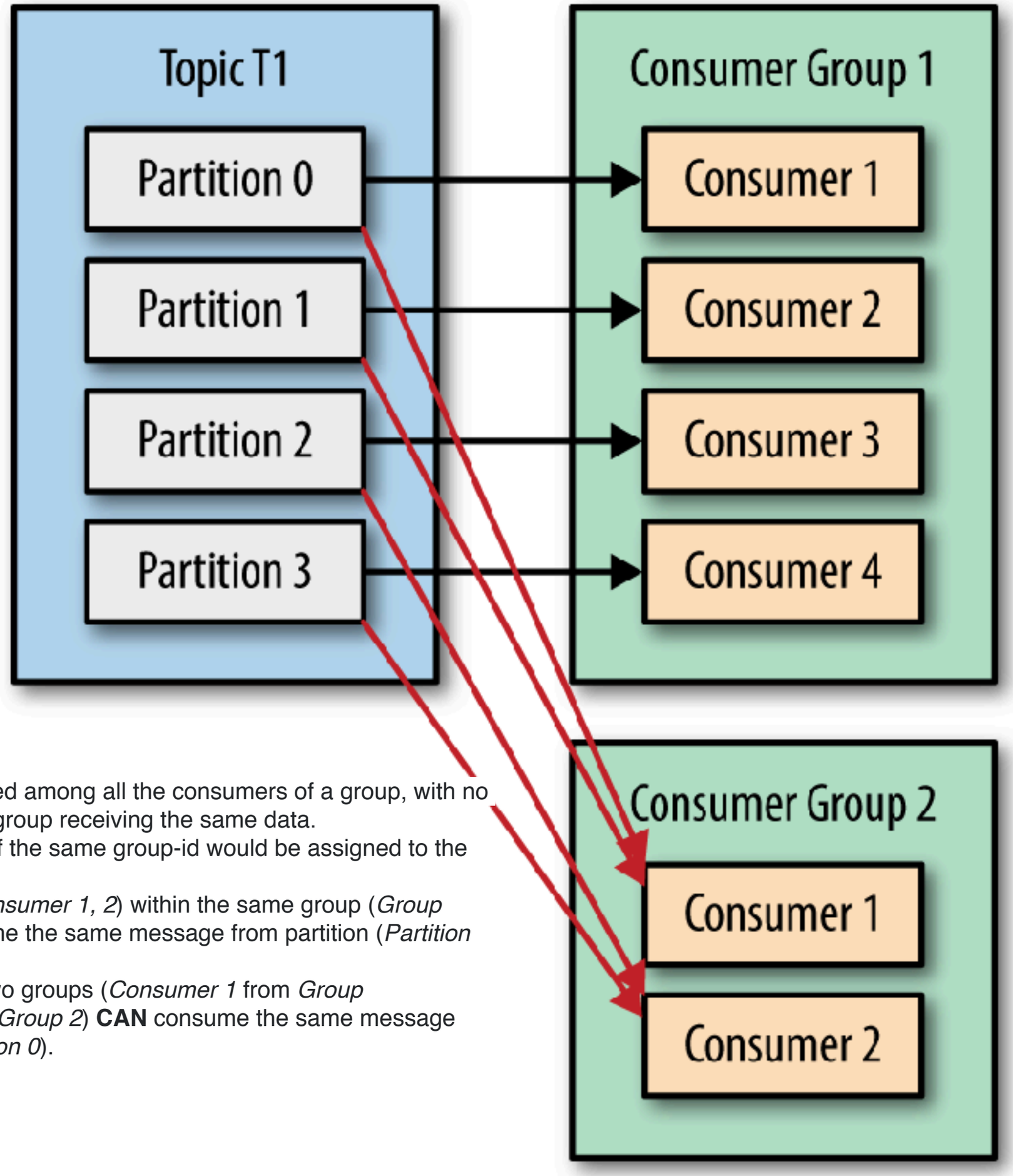
bulkhead

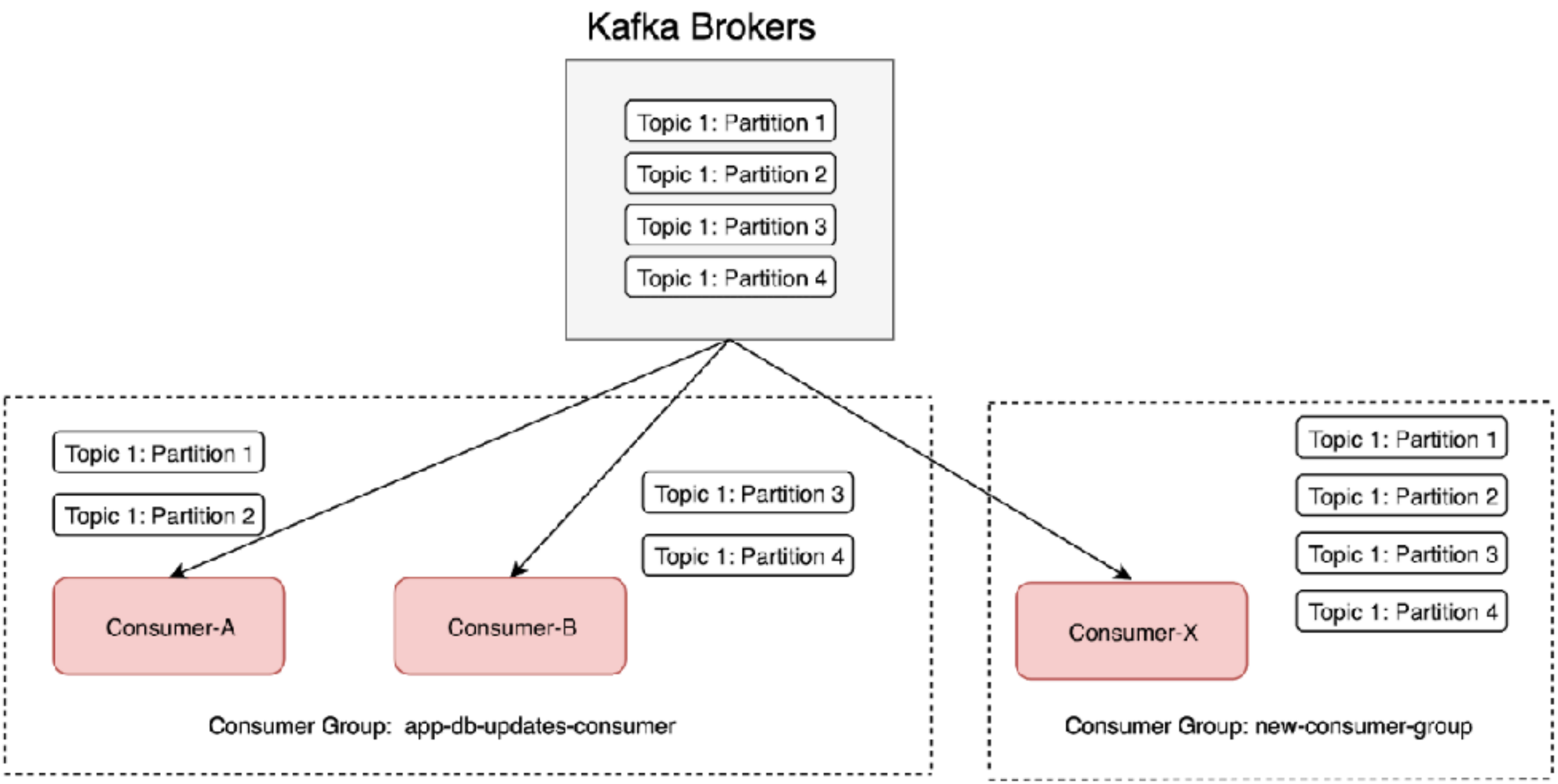


- In general, the goal of the bulkhead pattern is to avoid faults in one part of a system to take the entire system down. The term comes from ships where a ship is divided in separate watertight compartments to avoid a single hull breach to flood the entire ship; it will only flood one bulkhead.cir

- Partition service instances into different groups, based on consumer load and availability requirements.


P1
P2
P3
M1
M1
Tax Subscriber P1
State
Payroll Subscriber P1
M1
State
Tax Subscriber P2
State
Payroll Subscriber P2
State
Tax Subscriber P3
State
Payroll Subscriber P3
State



- Data is equally divided among all the consumers of a group, with no two consumers of a group receiving the same data.
- No two consumers of the same group-id would be assigned to the same partition.
- Two consumers (*Consumer 1, 2*) within the same group (*Group 1*) **CAN NOT** consume the same message from partition (*Partition 0*).
- Two consumers in two groups (*Consumer 1* from *Group 1*, *Consumer 1* from *Group 2*) **CAN** consume the same message from partition (*Partition 0*).

Multiple Kafka Consumer Groups