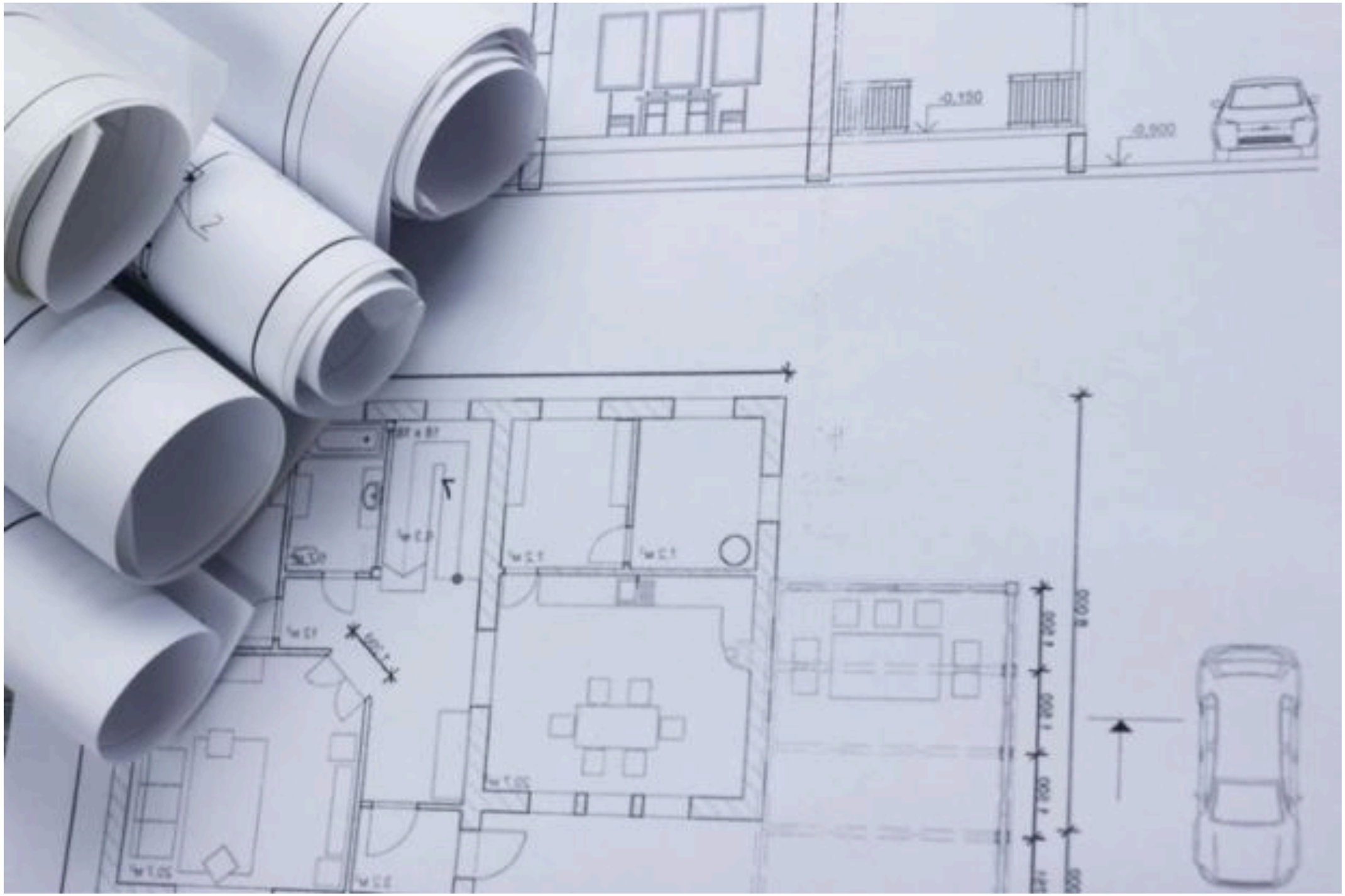


Software Architecture

» 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

Ui

APi

Domain

DAL

Database

5 Communicate

Key functional requirements

2

Understands Domain

NFR

3

Quality attributes, Measure (Architectural Requirements)

Collect

4

Choose

Approaches (Architectural Decisions)

1

Knows

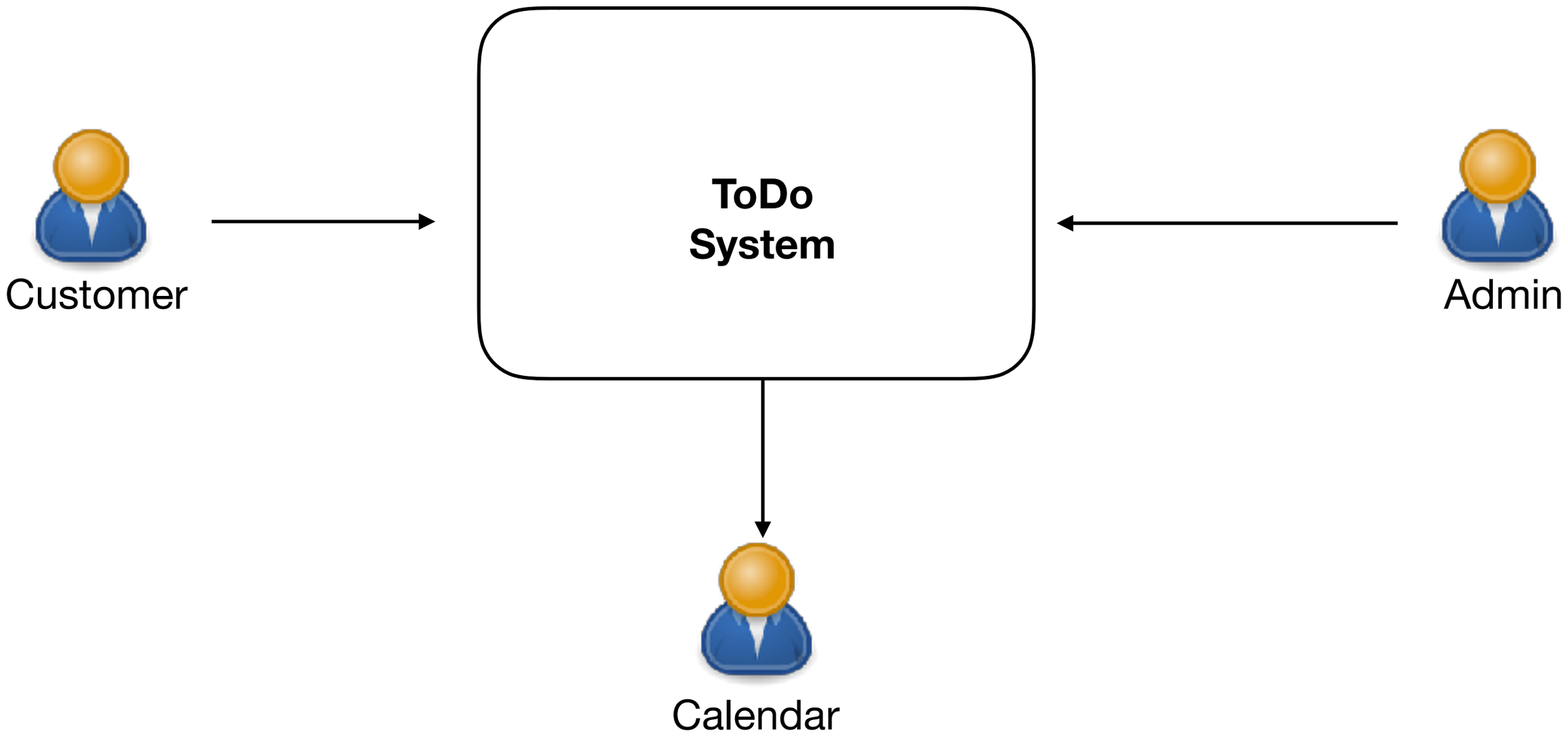
Arch Tactics, pattern, styles
Arch Principles
Arch anti patterns
Reference Arch (*)

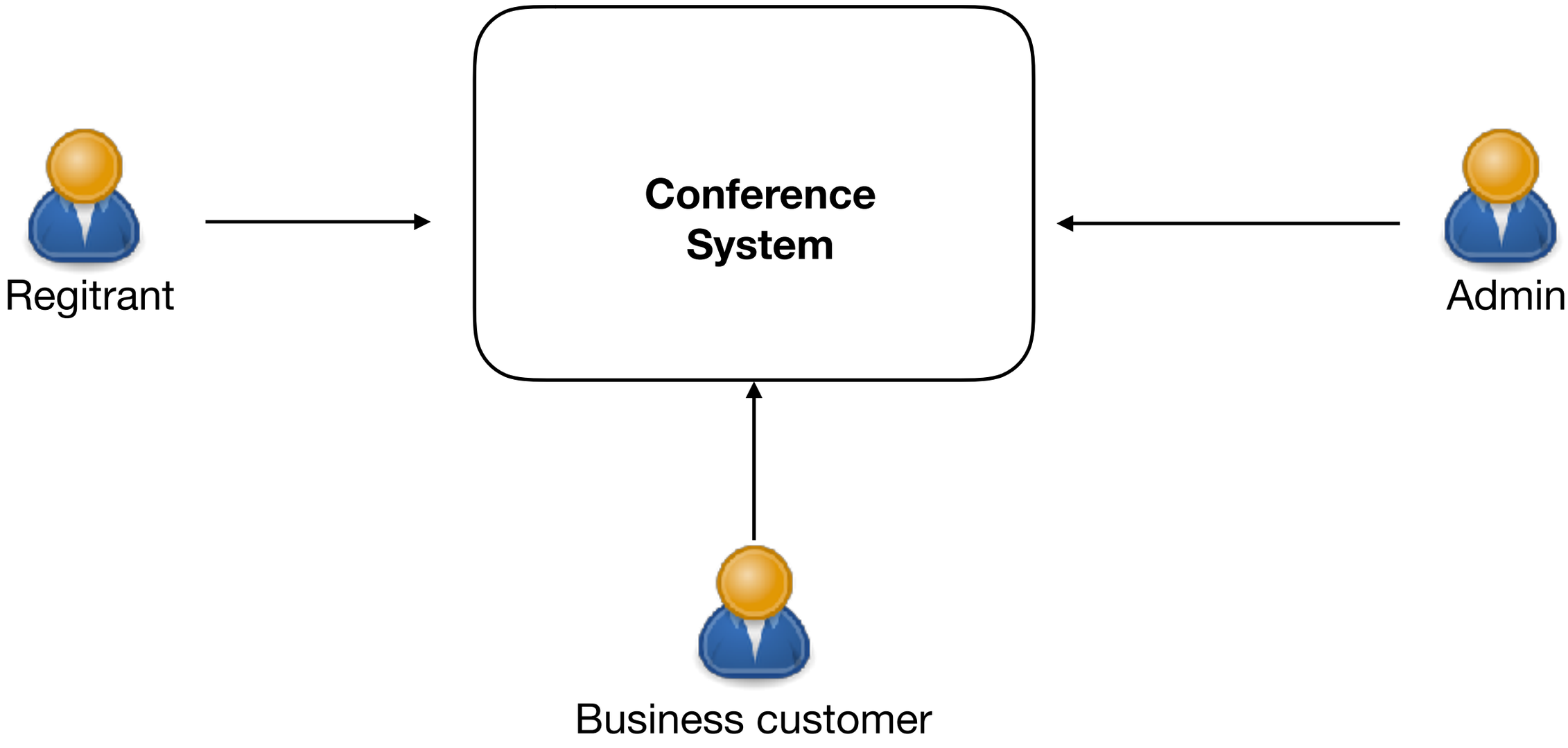
Software Architecture Views

Part 1

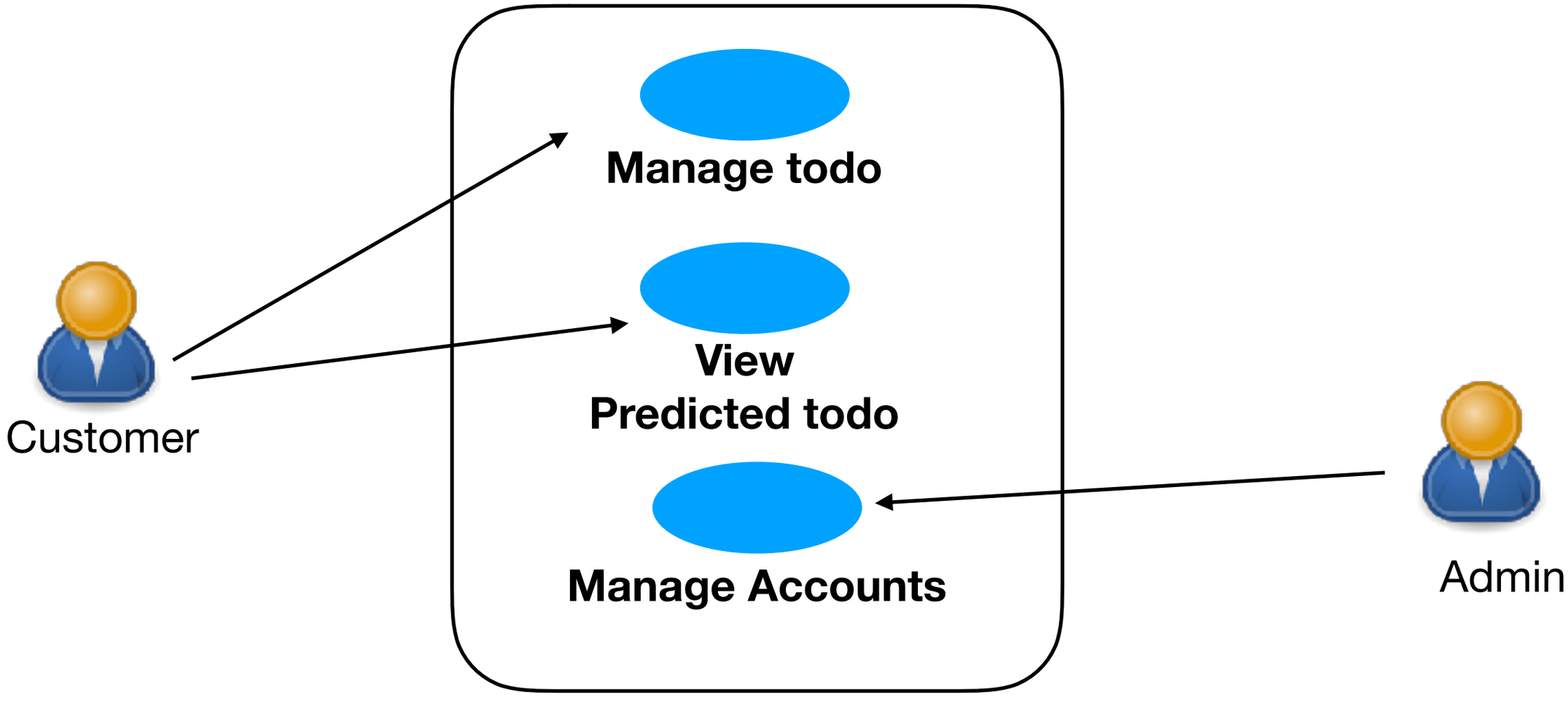
Arch Req

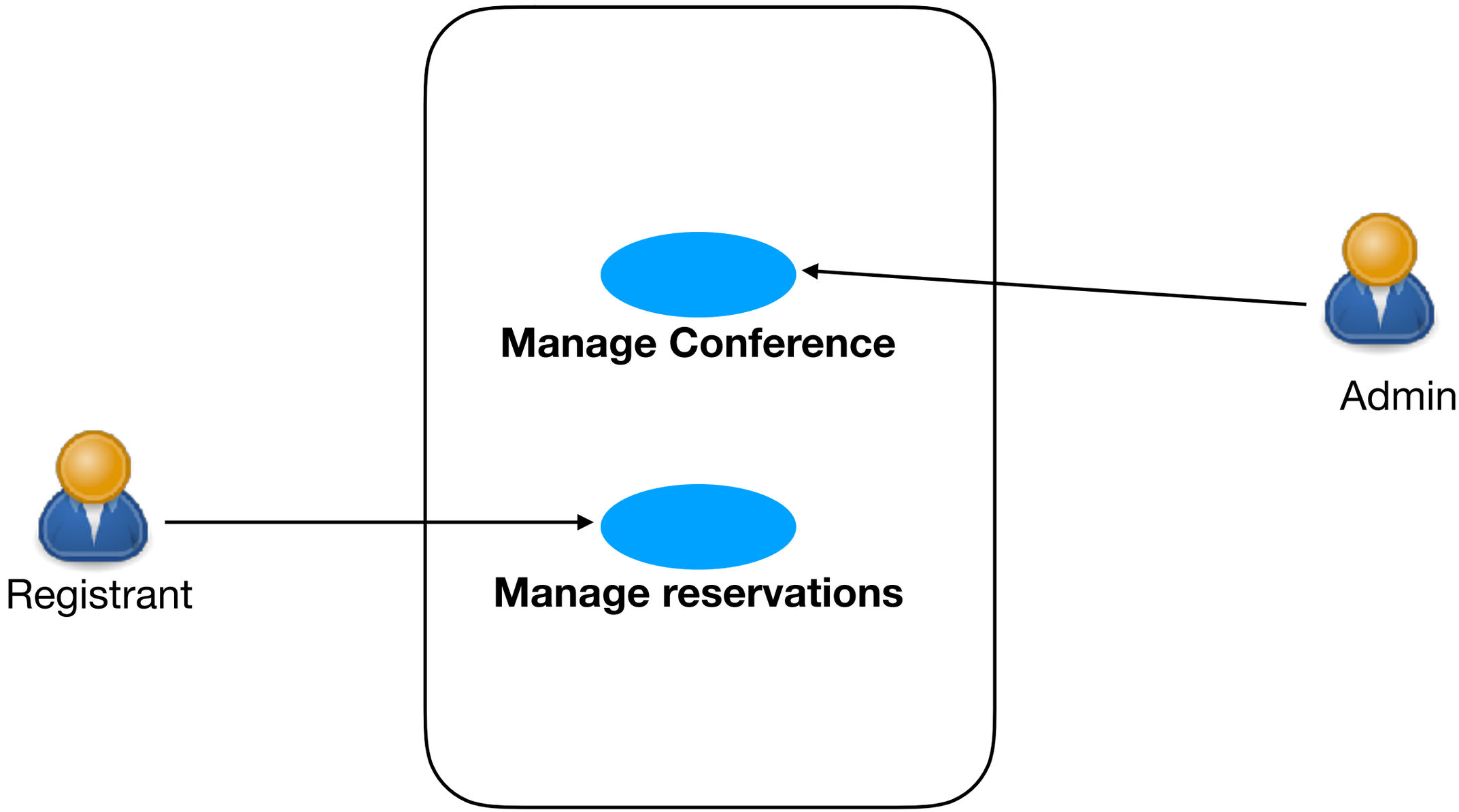
1. Context view



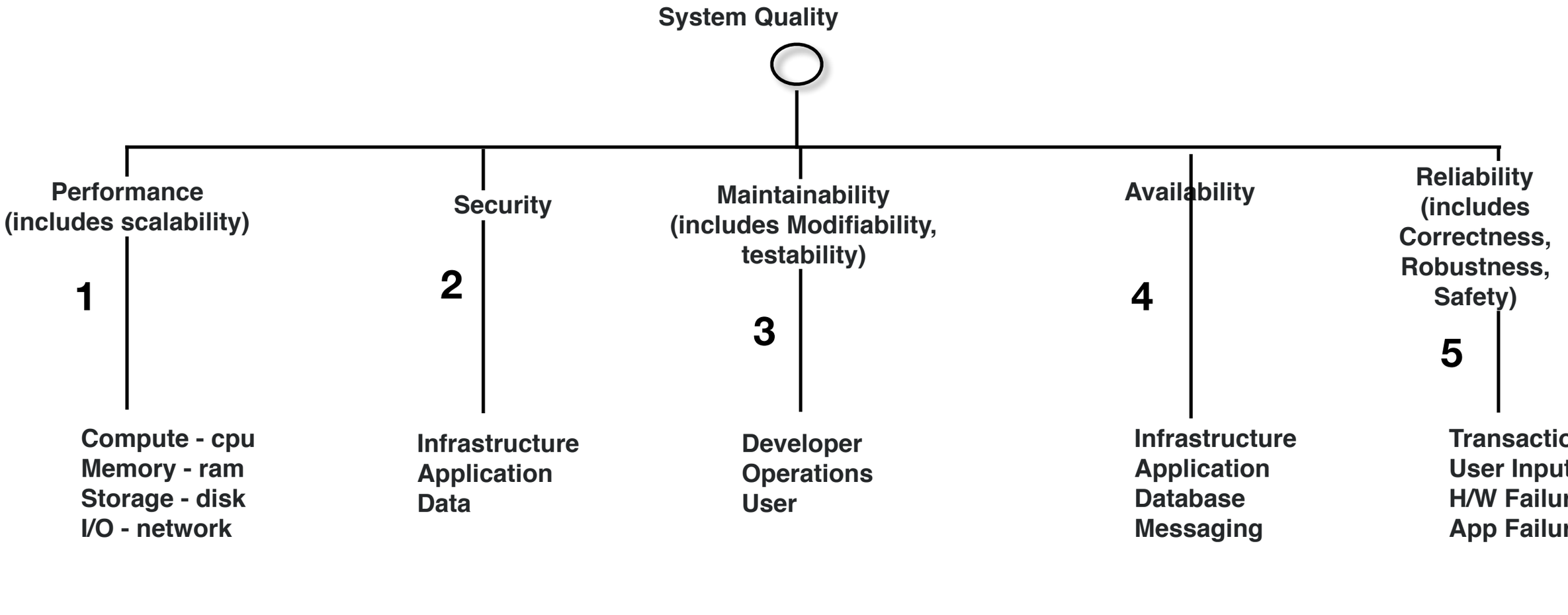

Conference System
Regitrant
Admin
Business customer

2. Functional view




Manage Conference
Admin
Registrant
Manage reservations

3. Quality View



Usability
Portability
Interoperability

Quality 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

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

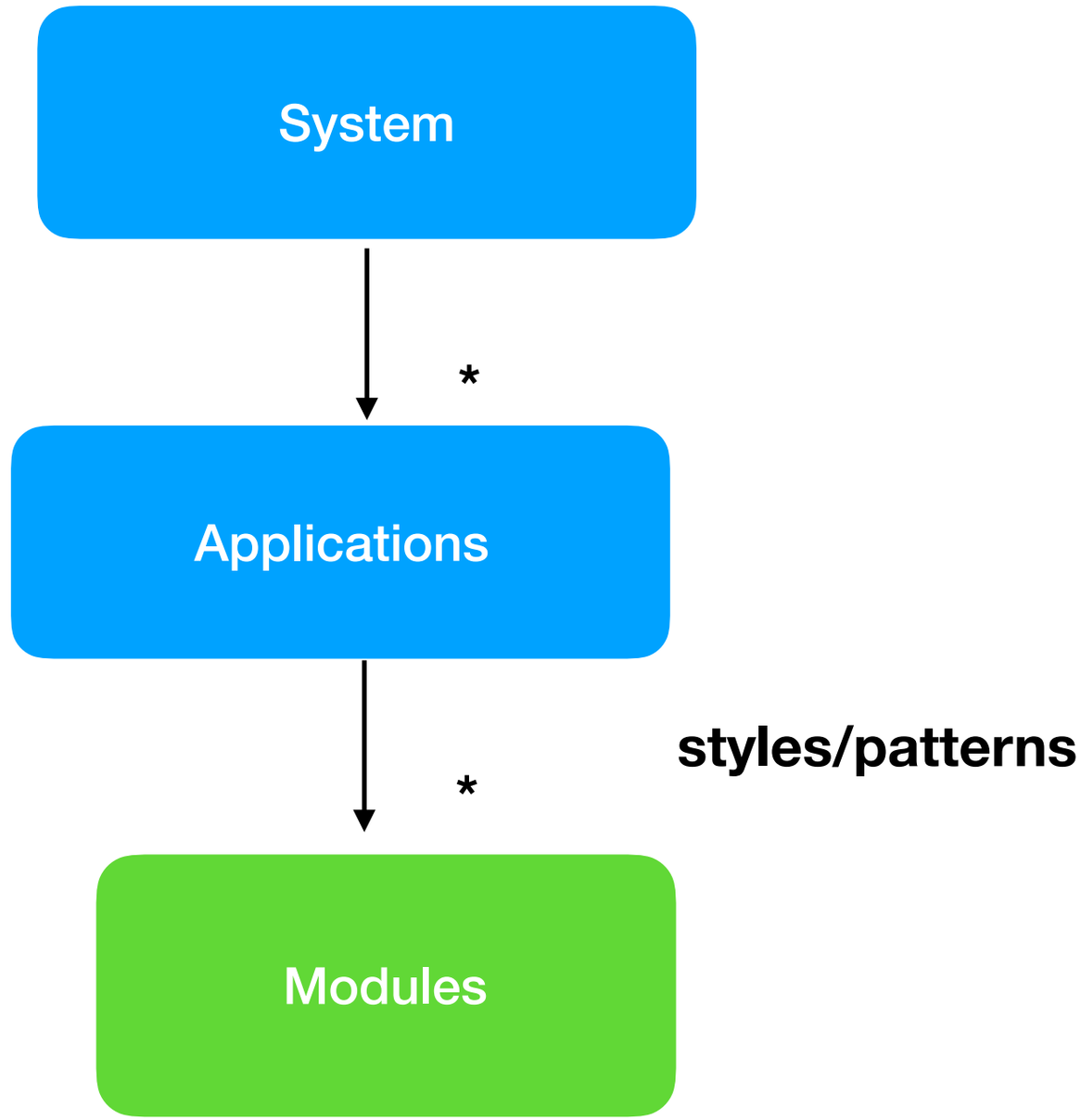
1	Will use open source Technologies only
2	
3	

Part 2

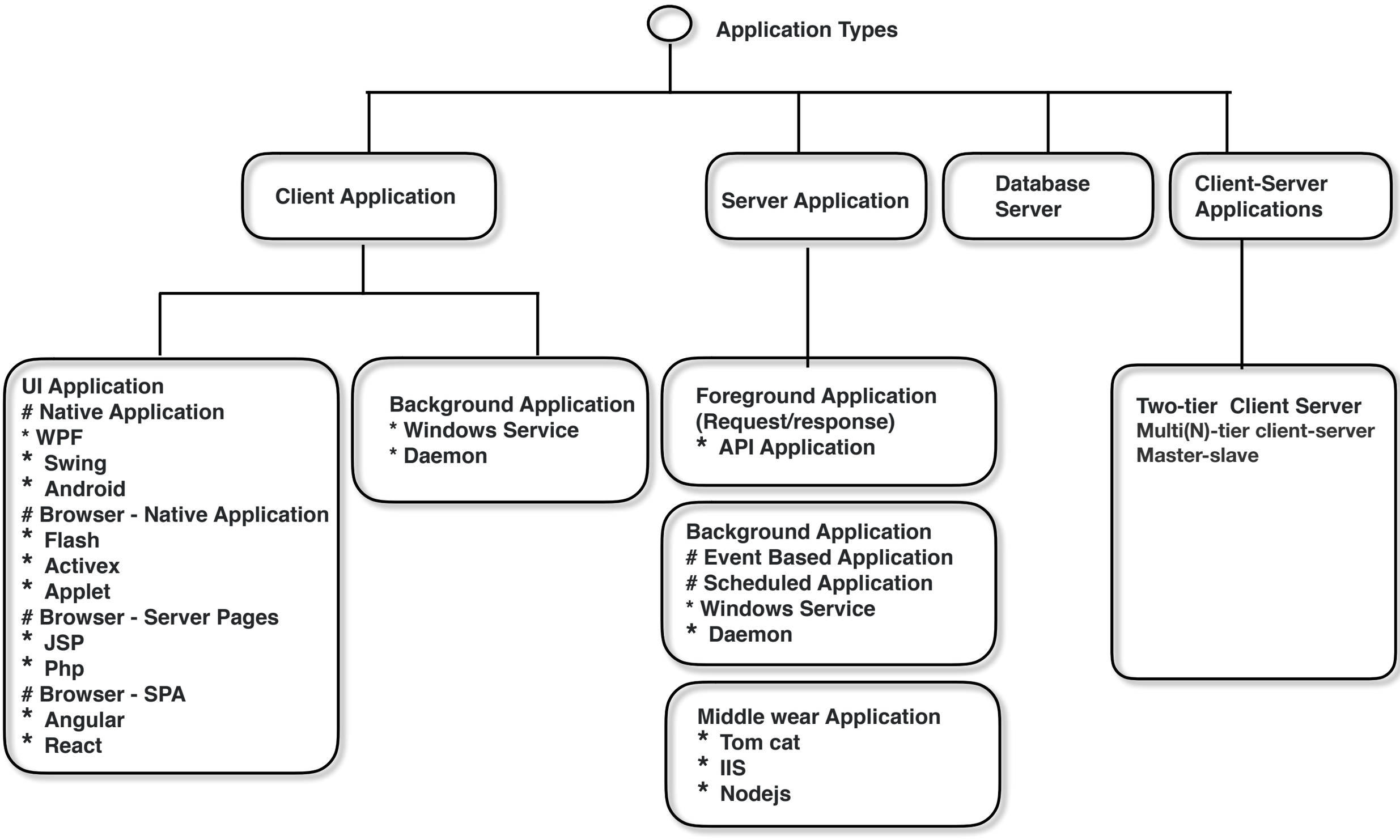
Arch Design

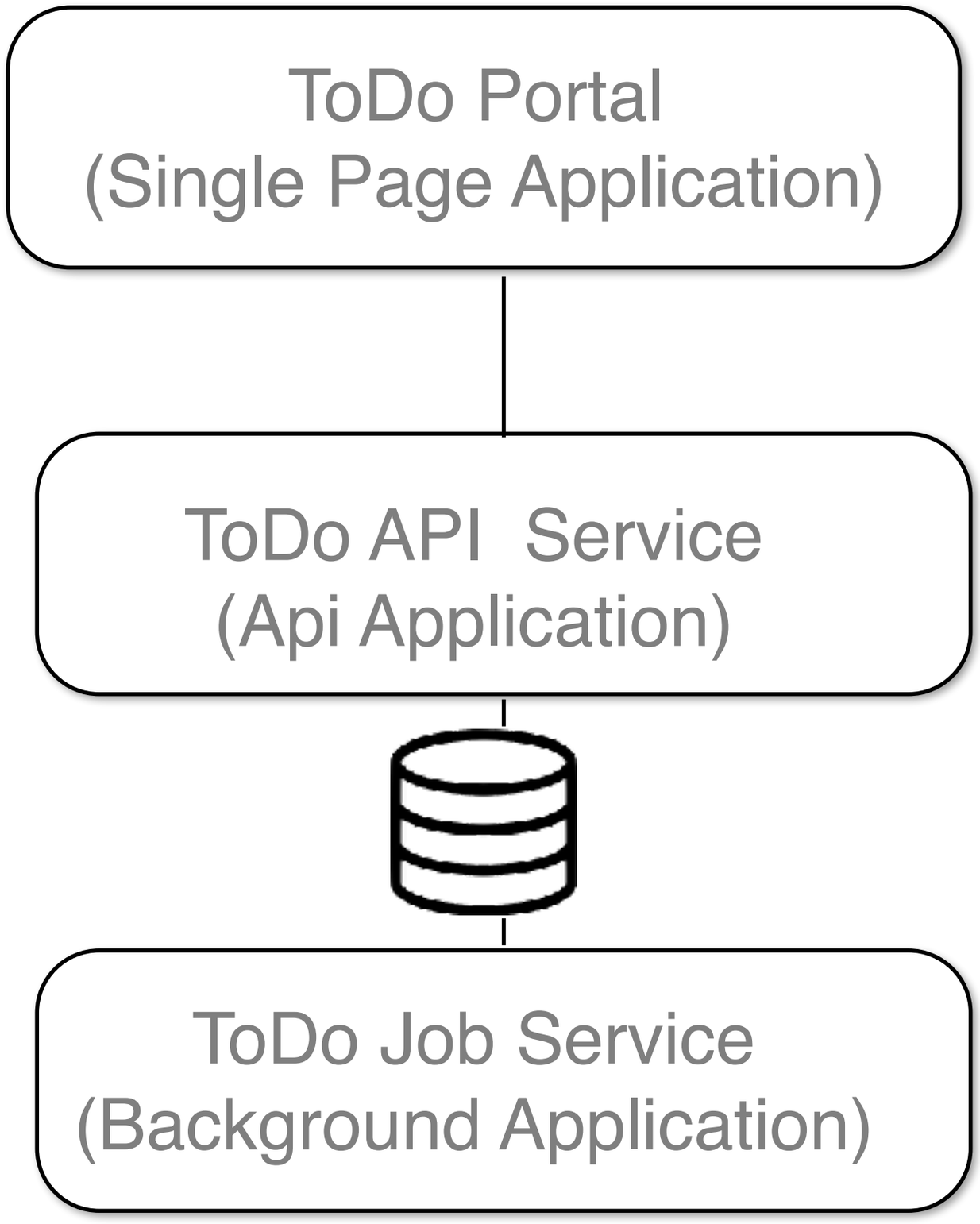
Decisions

6. Logical View

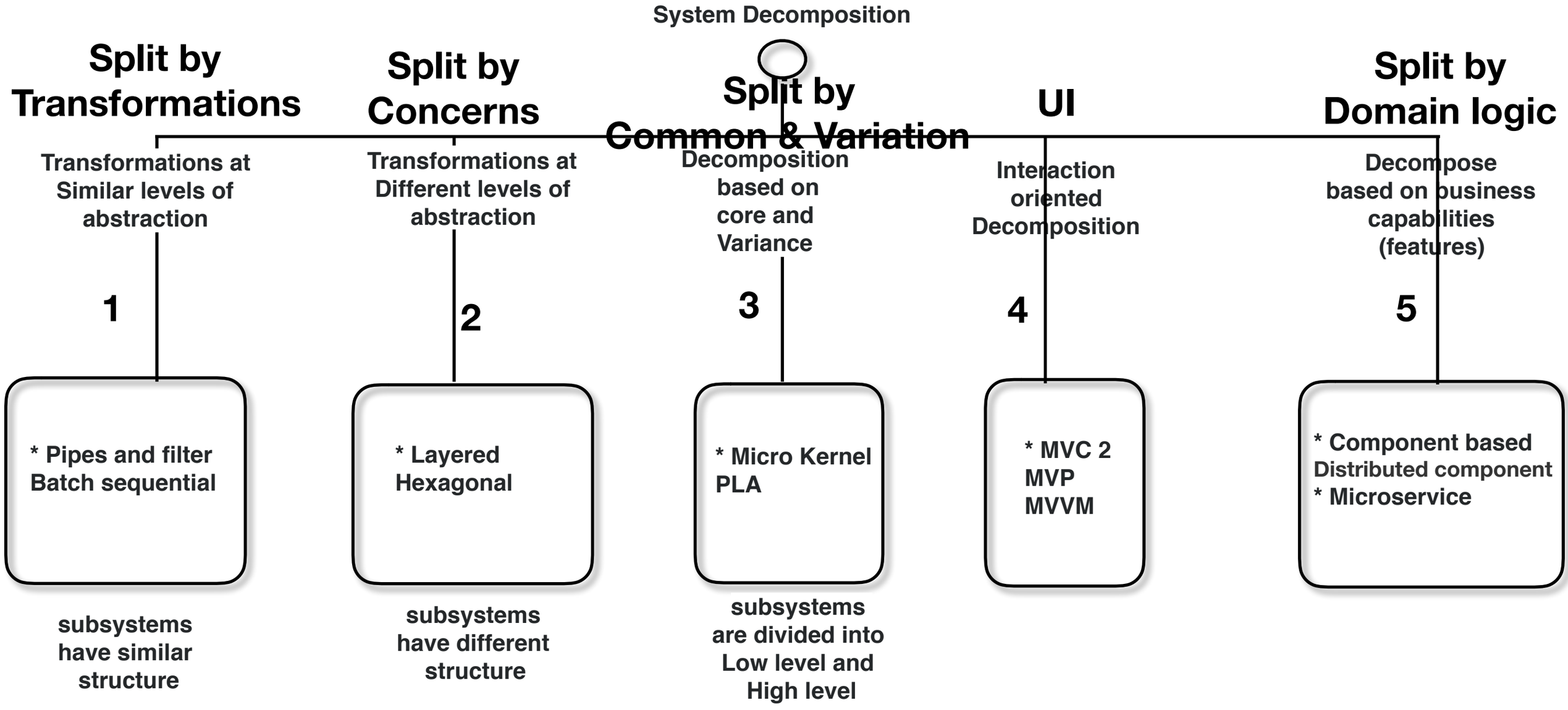


Step 1. Decompose the system into Applications/Tiers




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

Step 1. Decompose application/ Tiers



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

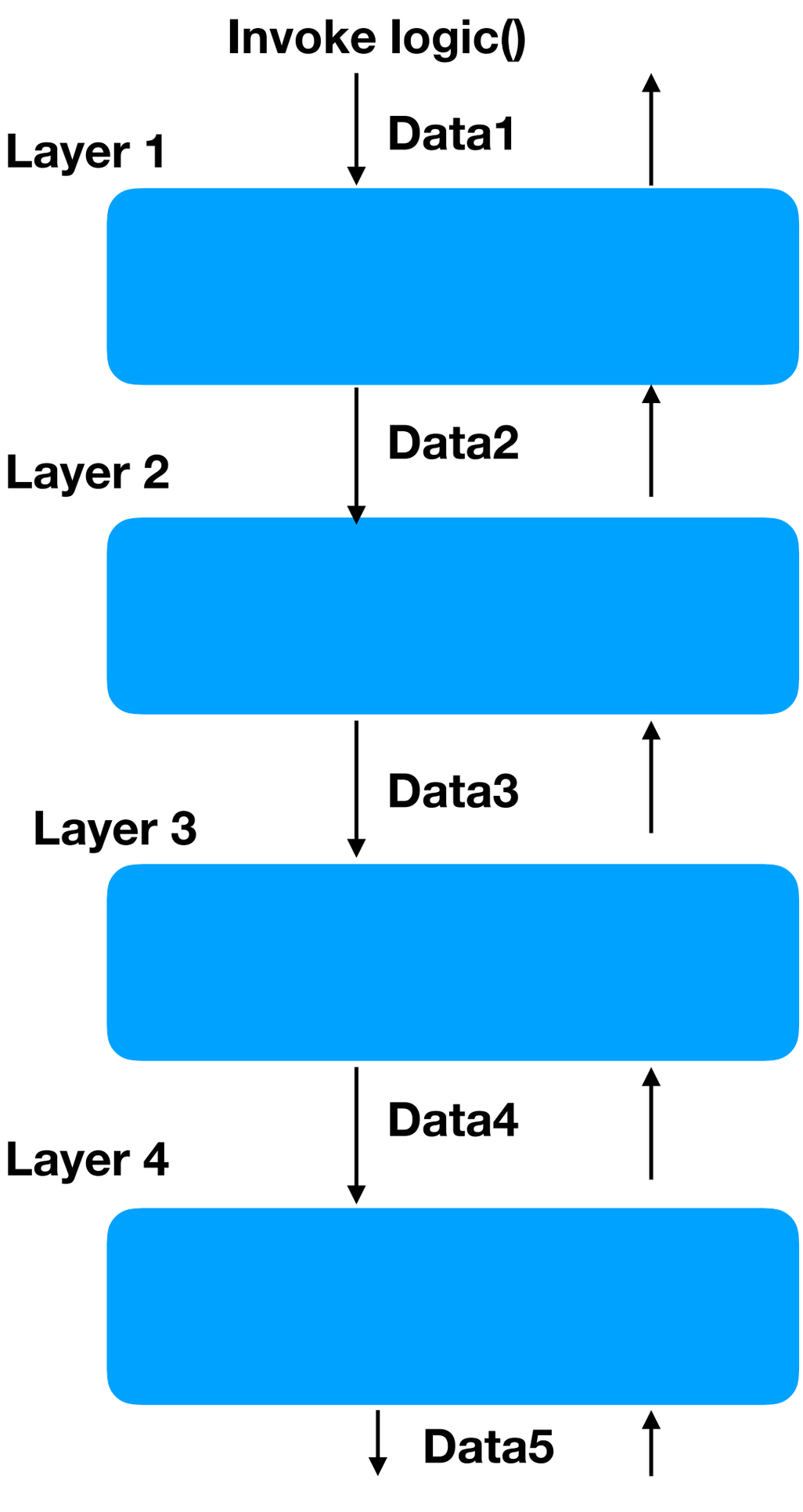
f3(int)

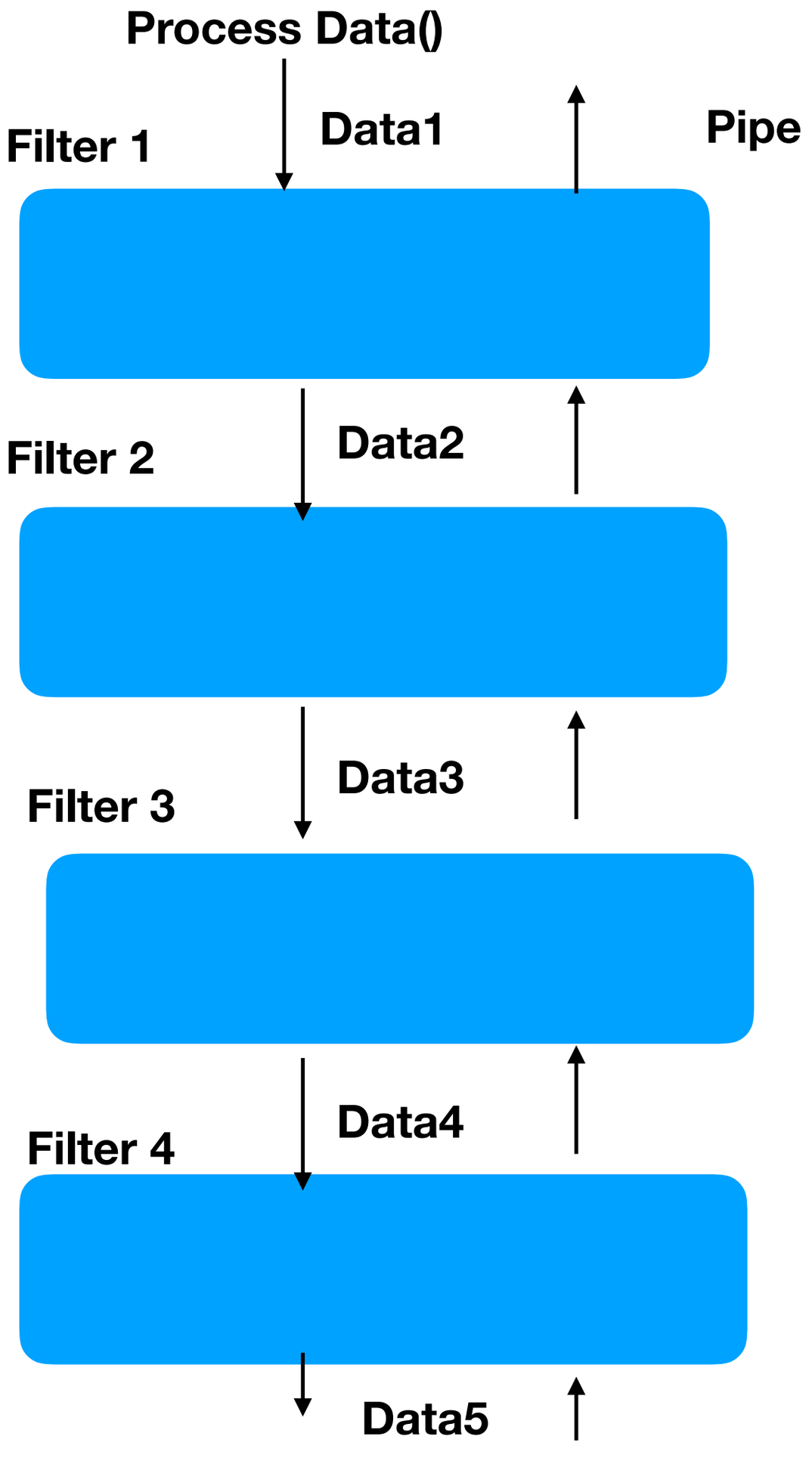
Data4

Filter 4

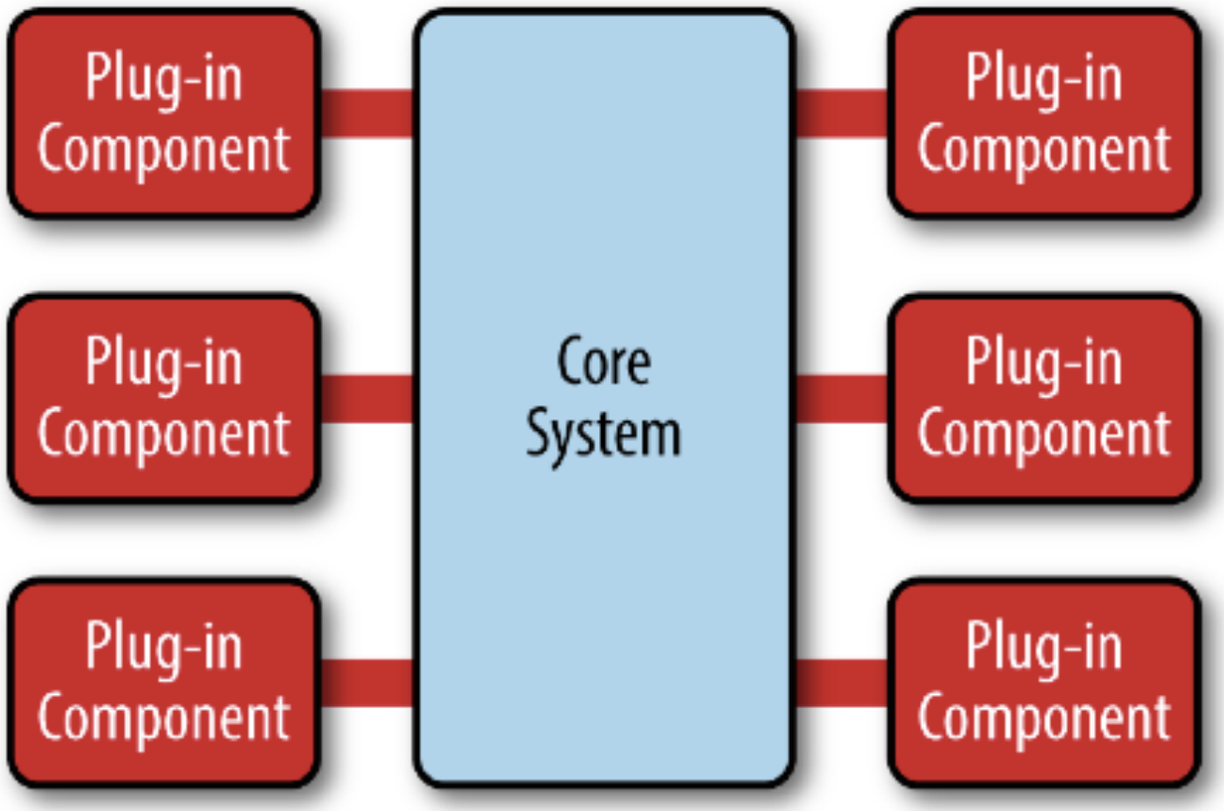
f4(int)

Data5

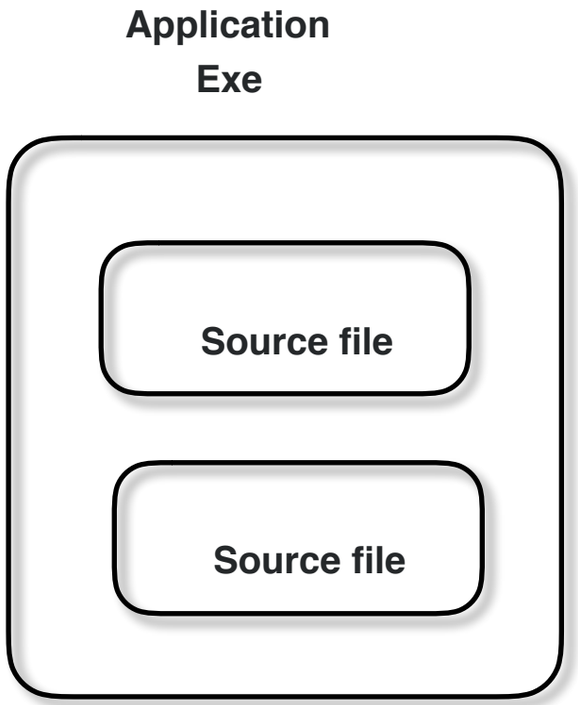

Invoke logic()
Layer 1
Data1
Data2
Layer 2
Data3
Layer 3
Data4
Layer 4
Data5


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

Microkernel Architecture

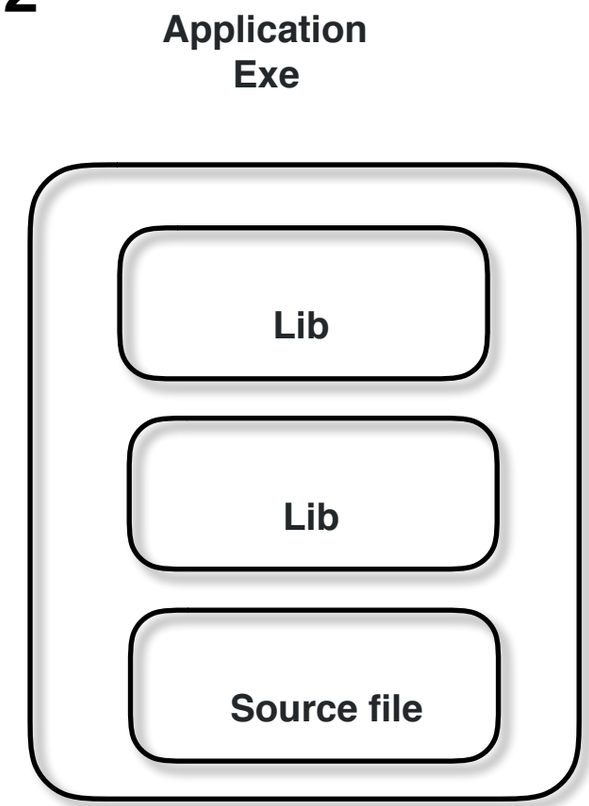


1



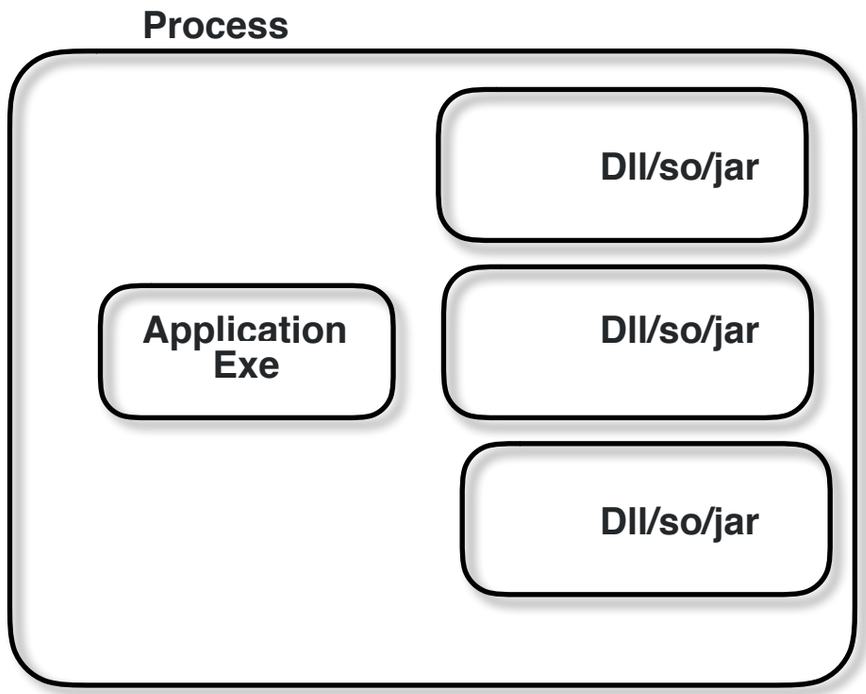
Compile time monolithic

2



Link time monolithic

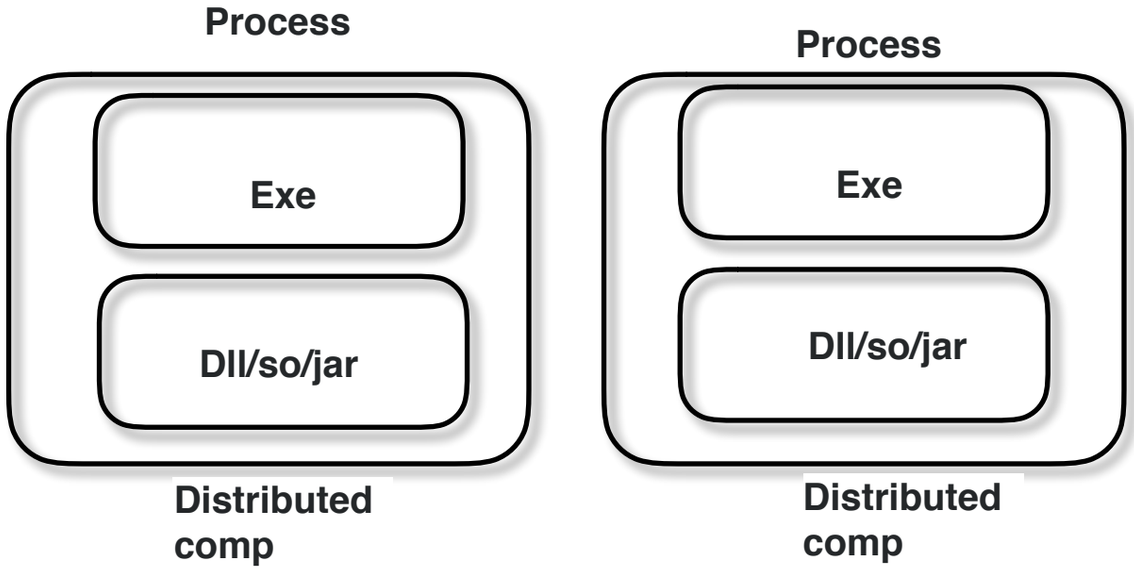
3



Runtime monolithic

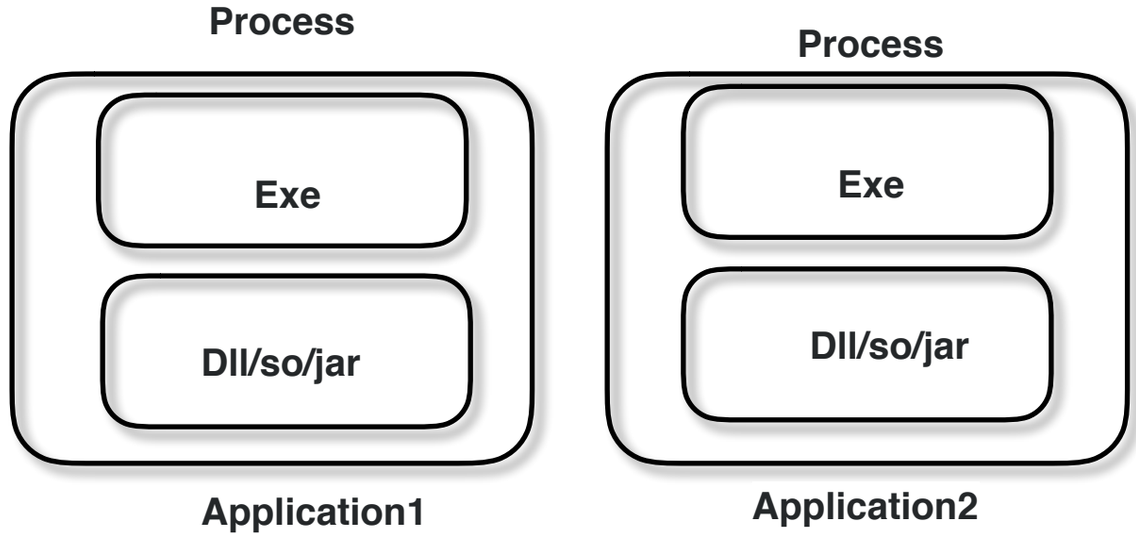
In process comp

4



Distributed Application

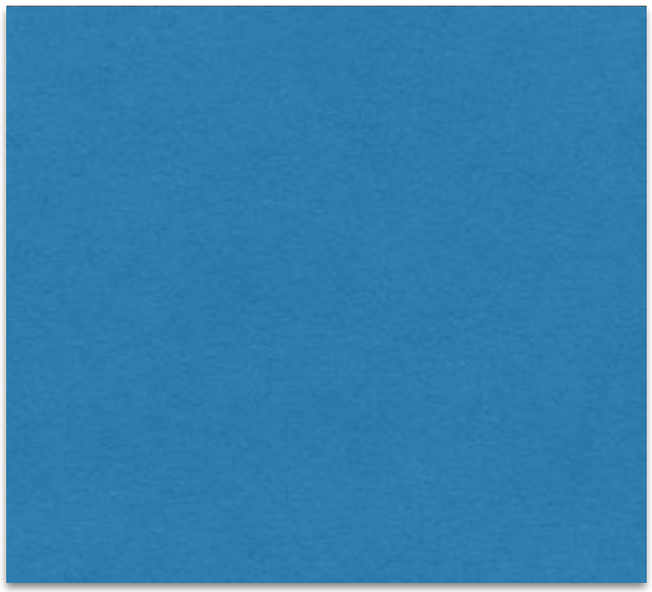
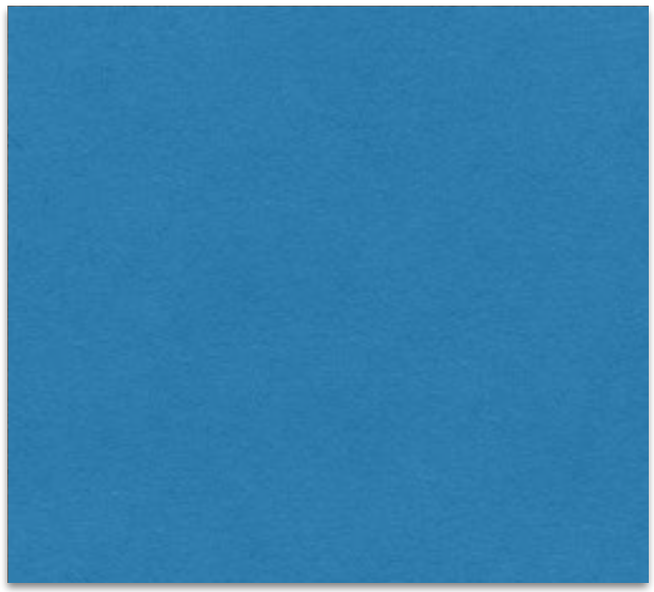
5



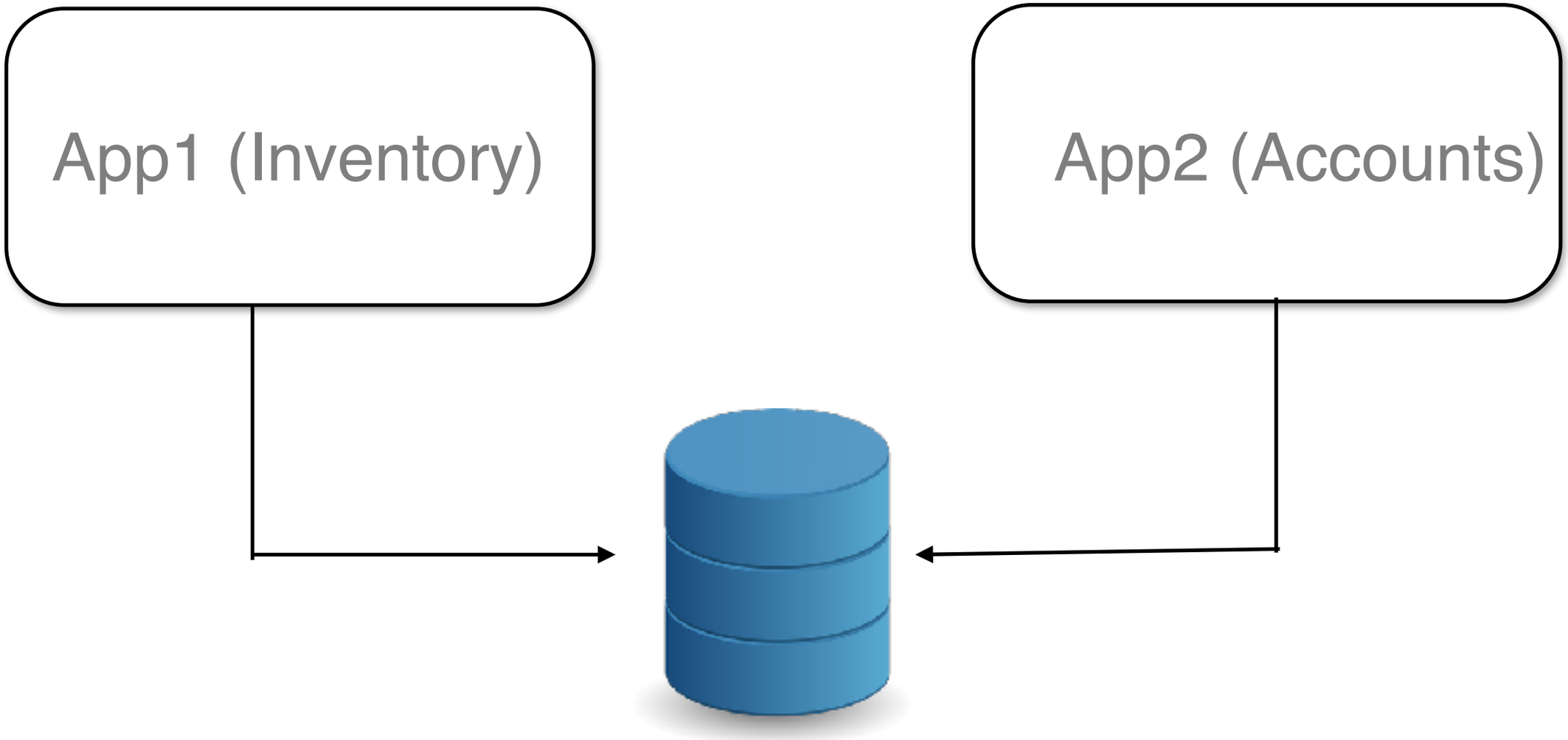
Micro Application

Break an app into smaller manageable piece

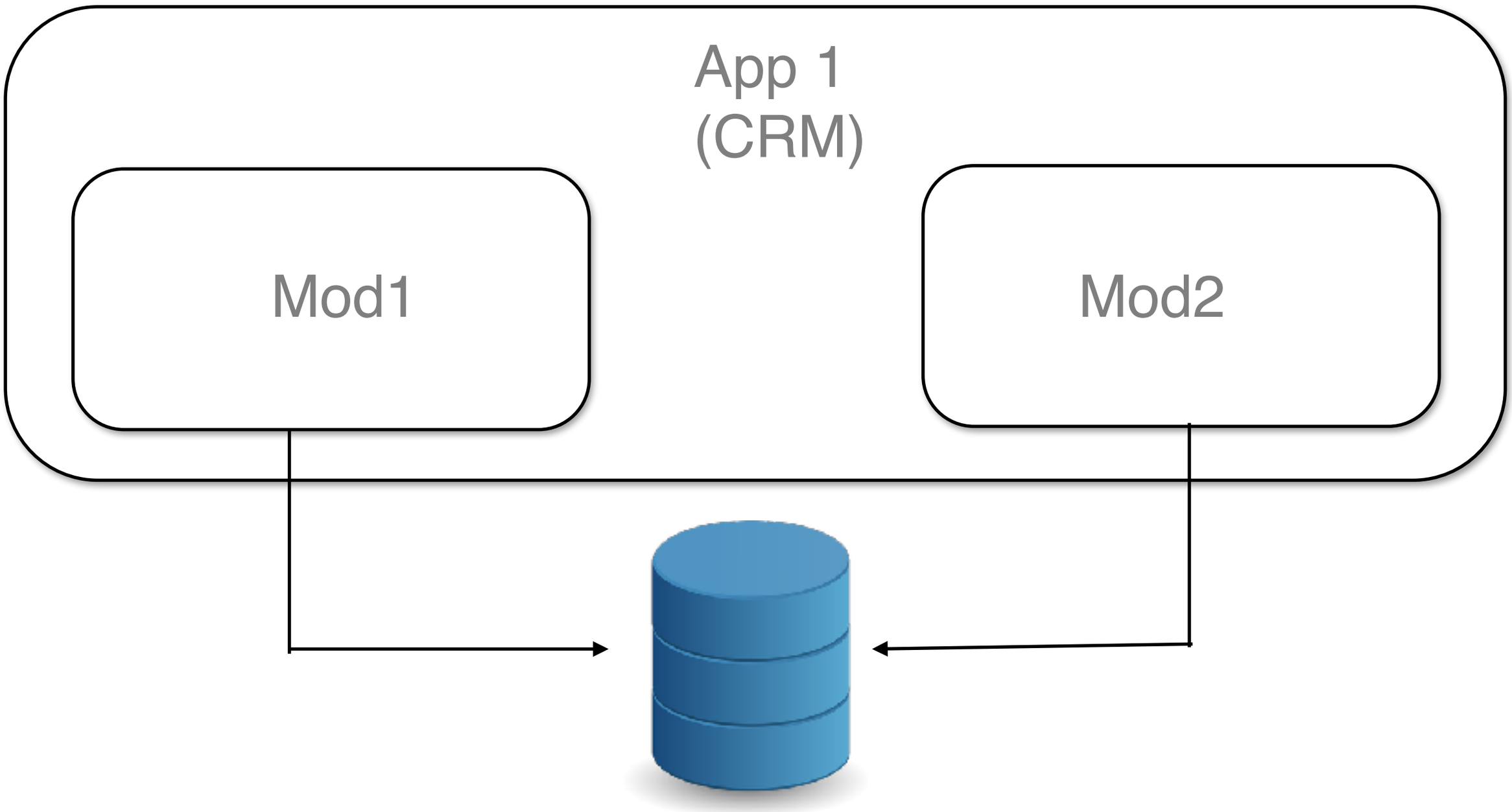
	Application Decomposed into Modules	**Application Decomposed into Small Application**	**W**	**Score**
Database / Storage	Shared	Not shared	2	
Infra (Hosting)	Shared	Not shared	3	
Sorce Control	Shared	Not Shared	2	
CI/CD (Build Server)	Shared	Not Shared	3	
Code	Share	?		
Fun Requirements	Shared	Not Shared	1	
SCRUM Team / Sprint	Shared	Not Shared	1	
Test Cases	Shared	Not Shared	1	
Architecture	Shared	Not Shared	1	
Technology Stack / Fwks	Shared	Not Shared	1	

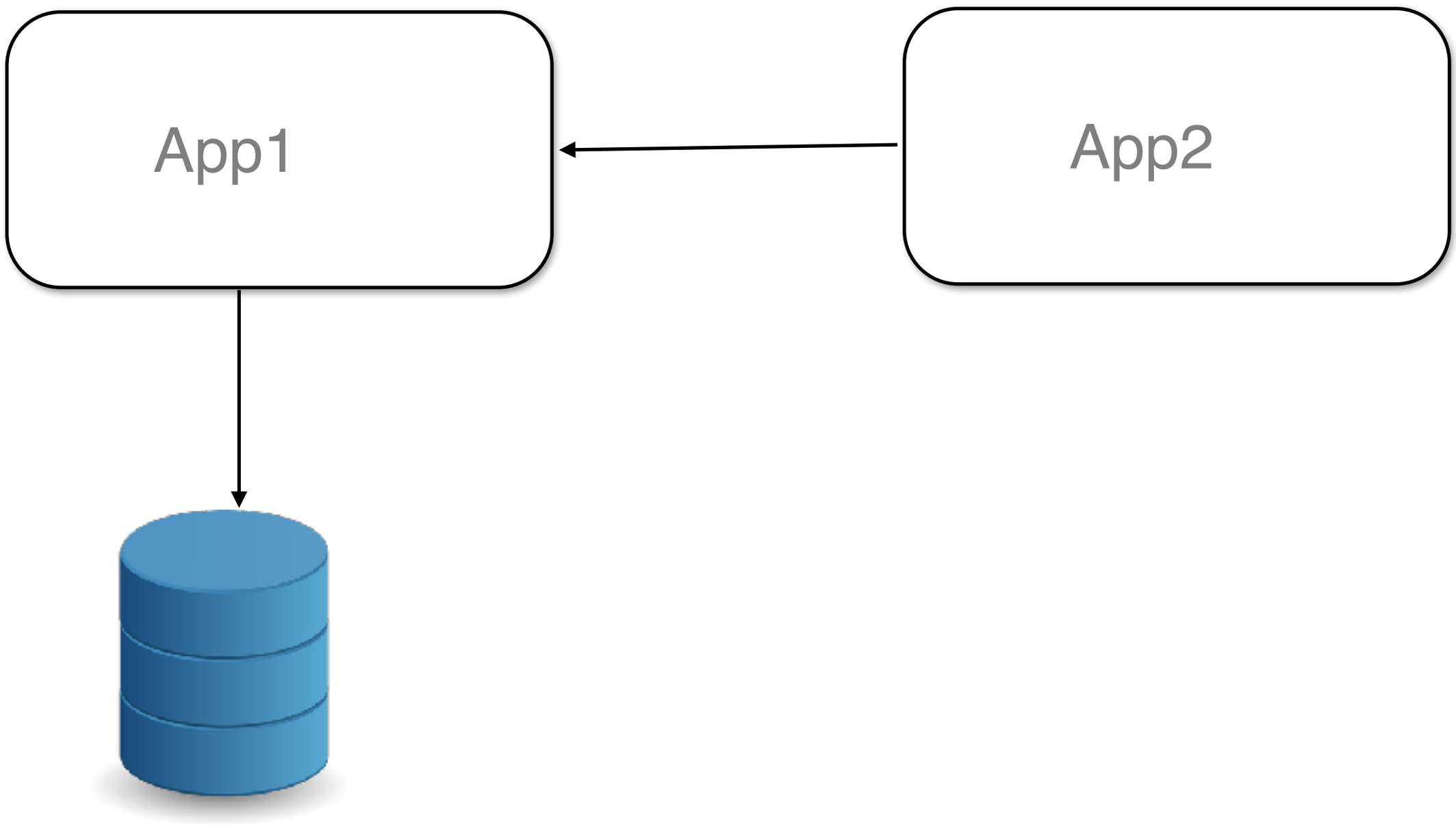
1. Database



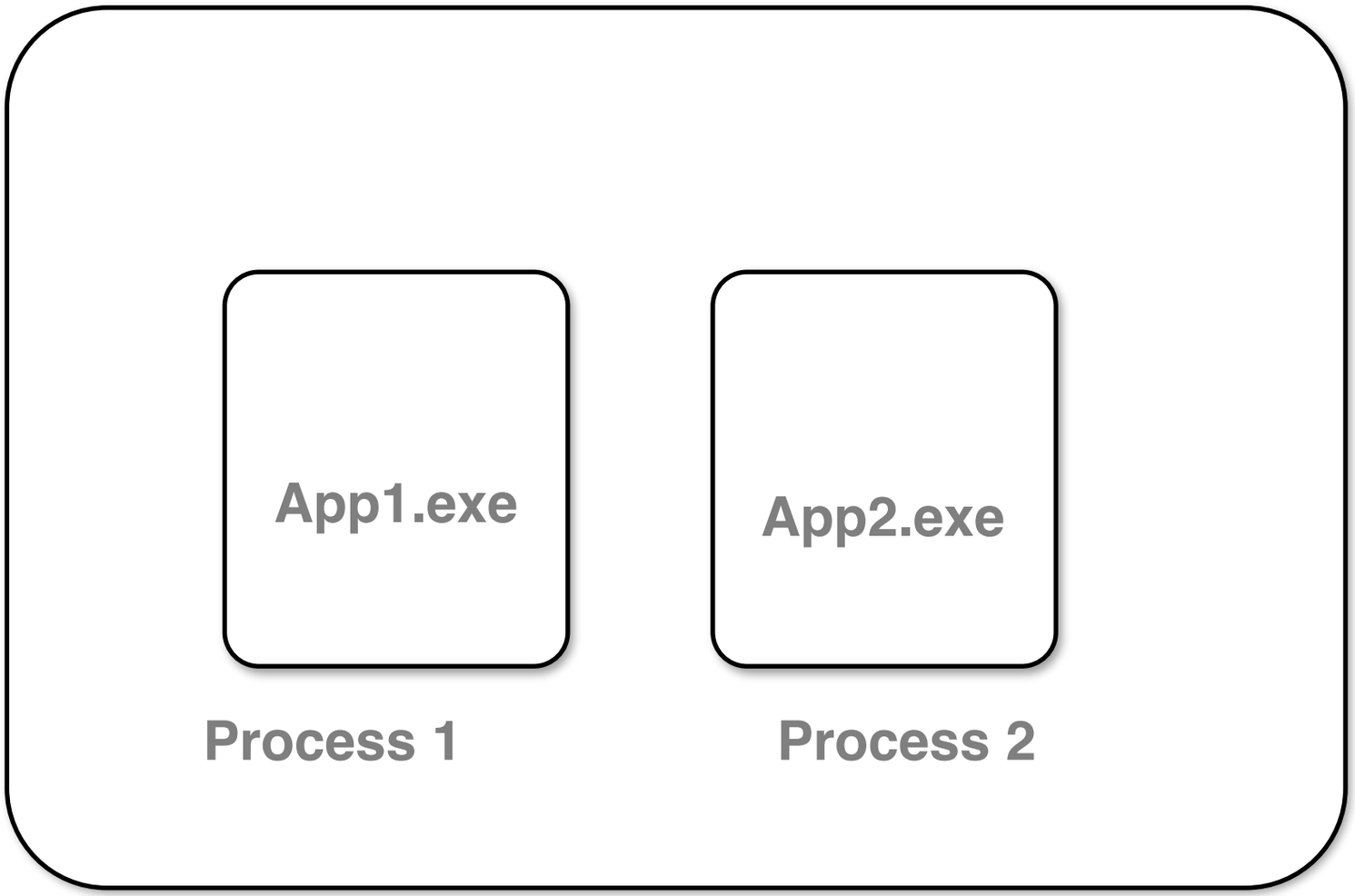
1. Database



1. Database



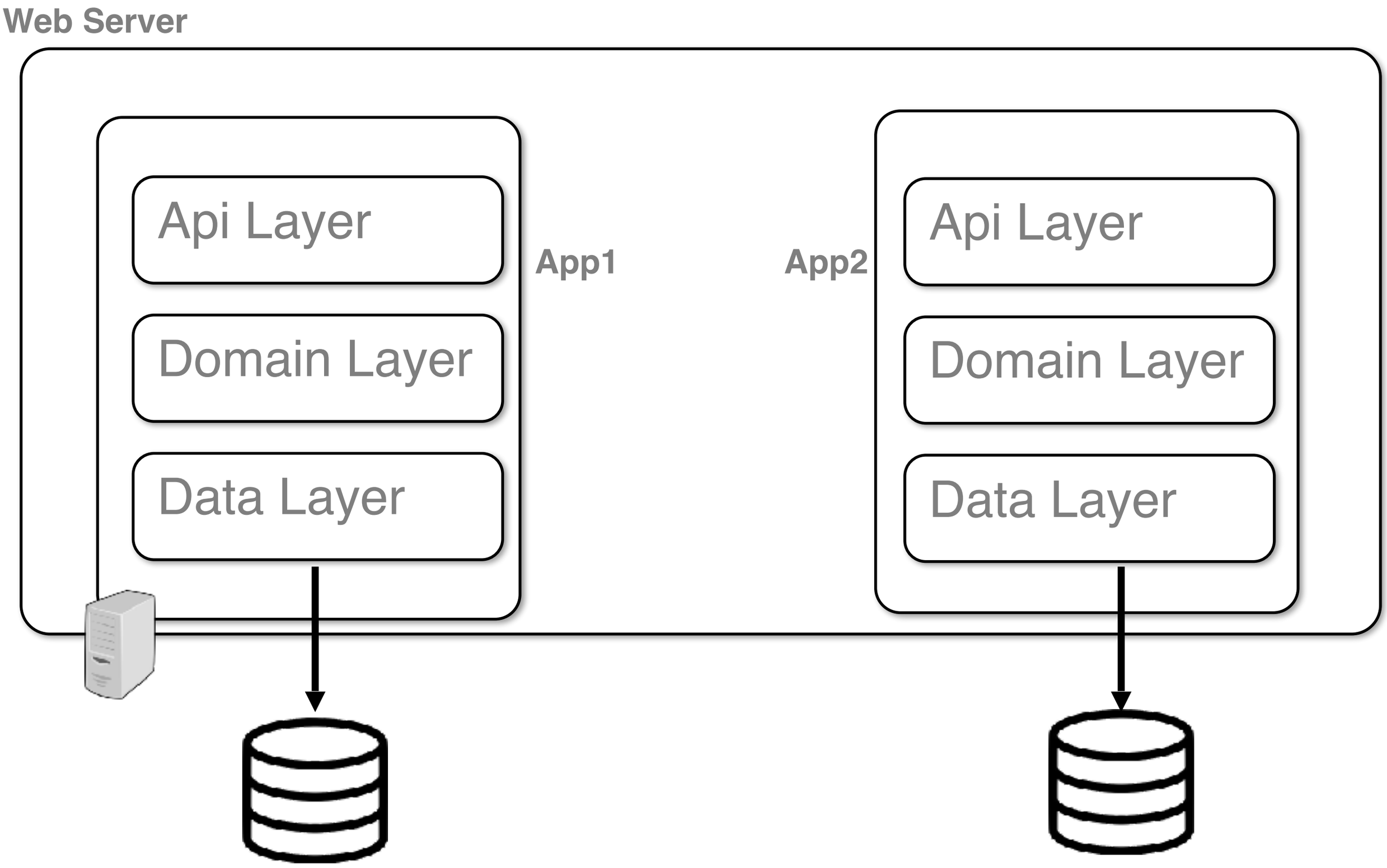
Server Machine (physical / virtual / docker container)



3 types of infrastructure

1. Physical
2. Virtual
3. Containerized

2. Infrastructure



ToDo Portal
(Single Page Application)

ToDo API Service
(Api Application)

ToDo Calendar Service
(Background Application)

ToDo Prediction Service
(Background Application)

validation,
formatting,
Routing,
State mgmt

View Layer

Controller Layer

Model

Technology

Logic

REST

Technology

Todo API
Service

Api Layer

Domain Layer

Message Adapter

Repository Layer

dto

Technology

Logic

Technology

Msg protocol (?)

Db protocol(?)

Message Queue

Calendar Sync
Service

Message Layer

Domain Layer

Calendar Adapter

dto

Data Access Layer

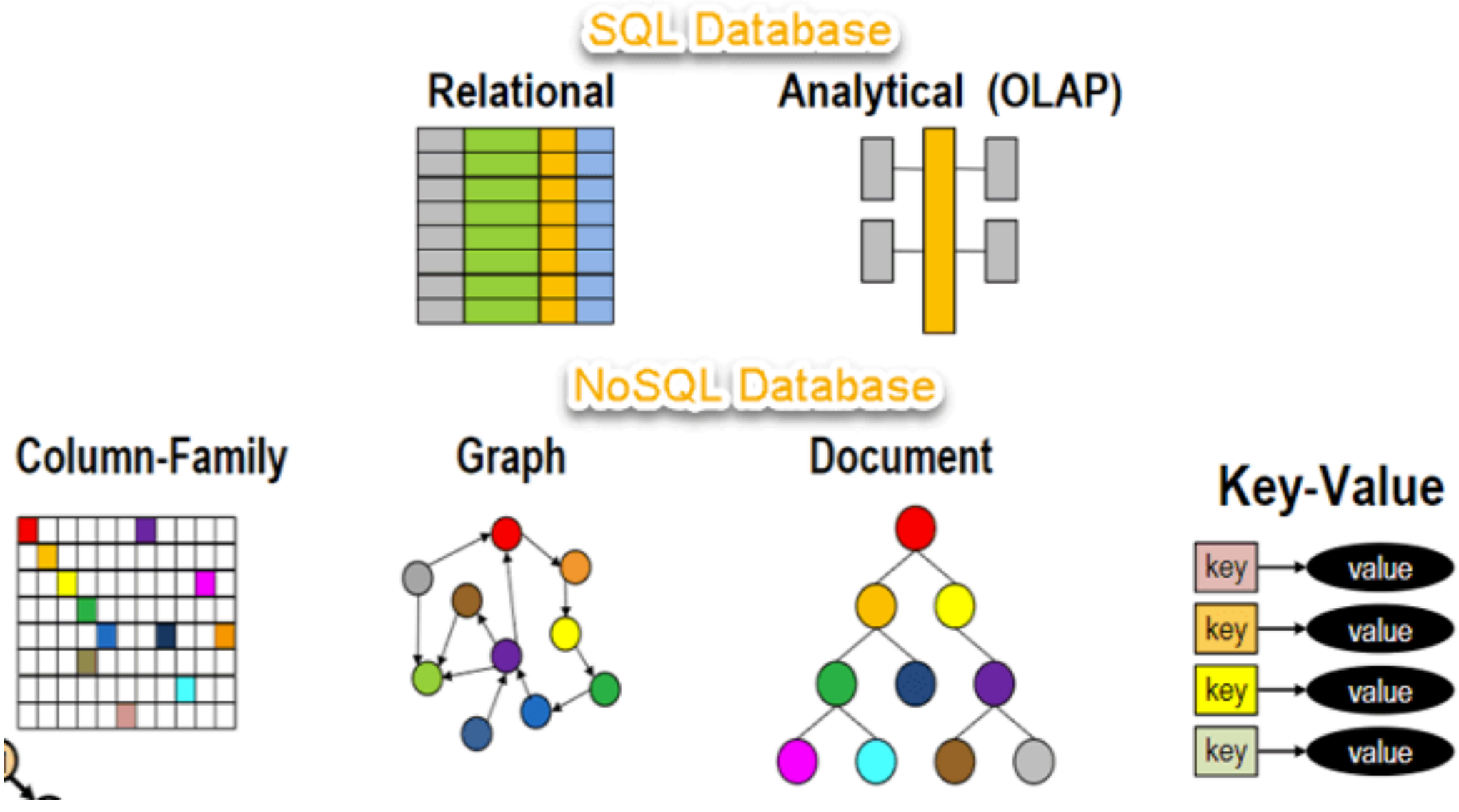
Domain Layer

Message Layer

dto

Todo Prediction
Service

8. Data View


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

Storage

Structured

ACID (OLTP)

RDBMS

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

Analytics (OLAP)

DWH

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

Semi Structured

Dictionary (key value)

Persistent

Key Value

- AWS Dynamo Db
- Azure Cosmos
- Google Big Table

Cache

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

2D Key Value

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

Wide Column

Nested Objects

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

Entity Relationship

Graph

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

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

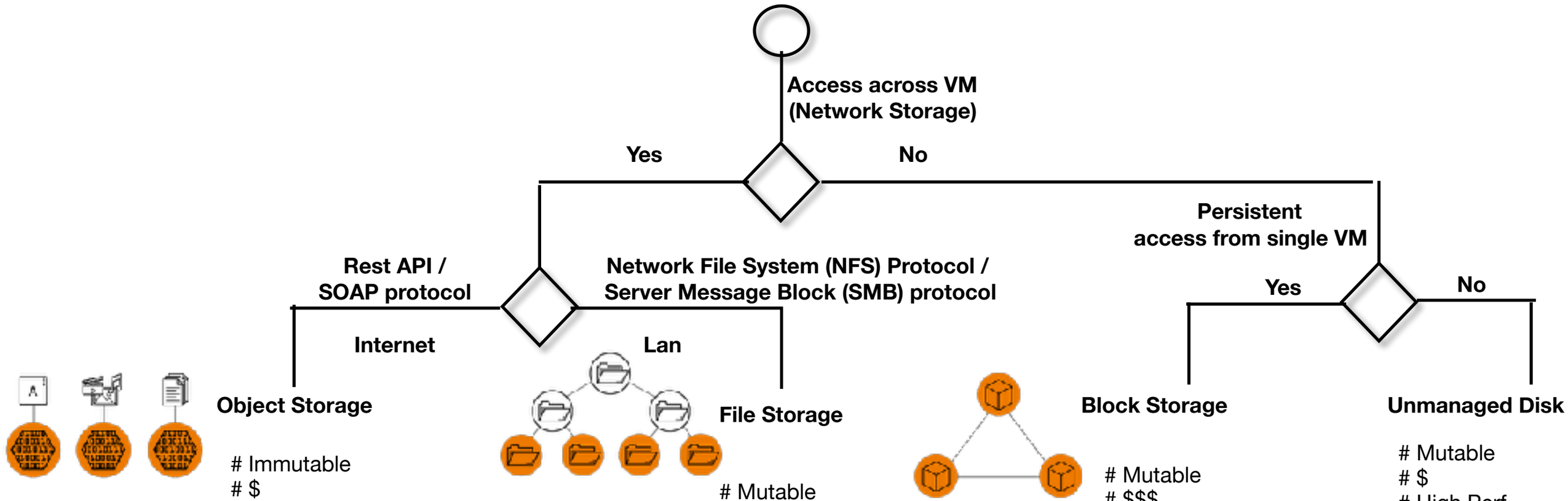
geo Spatial

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

Un Structured

- Binary storage

Binary Storage



Object Storage

Immutable
$
Low Perf
High Volume
0.01$/GB

- Azure Blobs
- AWS S3
- google Cloud Storage

File Storage

Mutable
$$$$
Medium Perf
Medium Voulme

- Azure Files
- EFS
- Google Filestore
- NFS

Block Storage

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

- Azure Disks
- EBS
- Google Persistent Disk

Unmanaged Disk

Mutable
$
High Perf

- Local Disk

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

Data Lake

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

- Azure Blobs
- S3
- google Cloud Storage

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)

RDBMS vs. Columnar: OLTP vs. OLAP

scgupta.link/datastores

OLTP

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

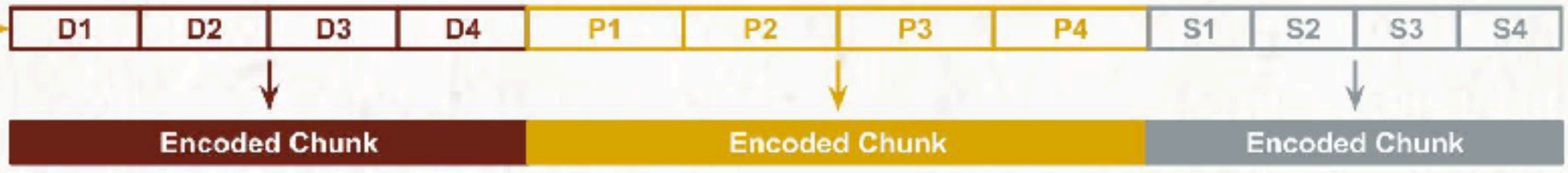
Logical Table

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

RDBMS: Row-Oriented Datastore

Columnar: Column-Oriented Datastore

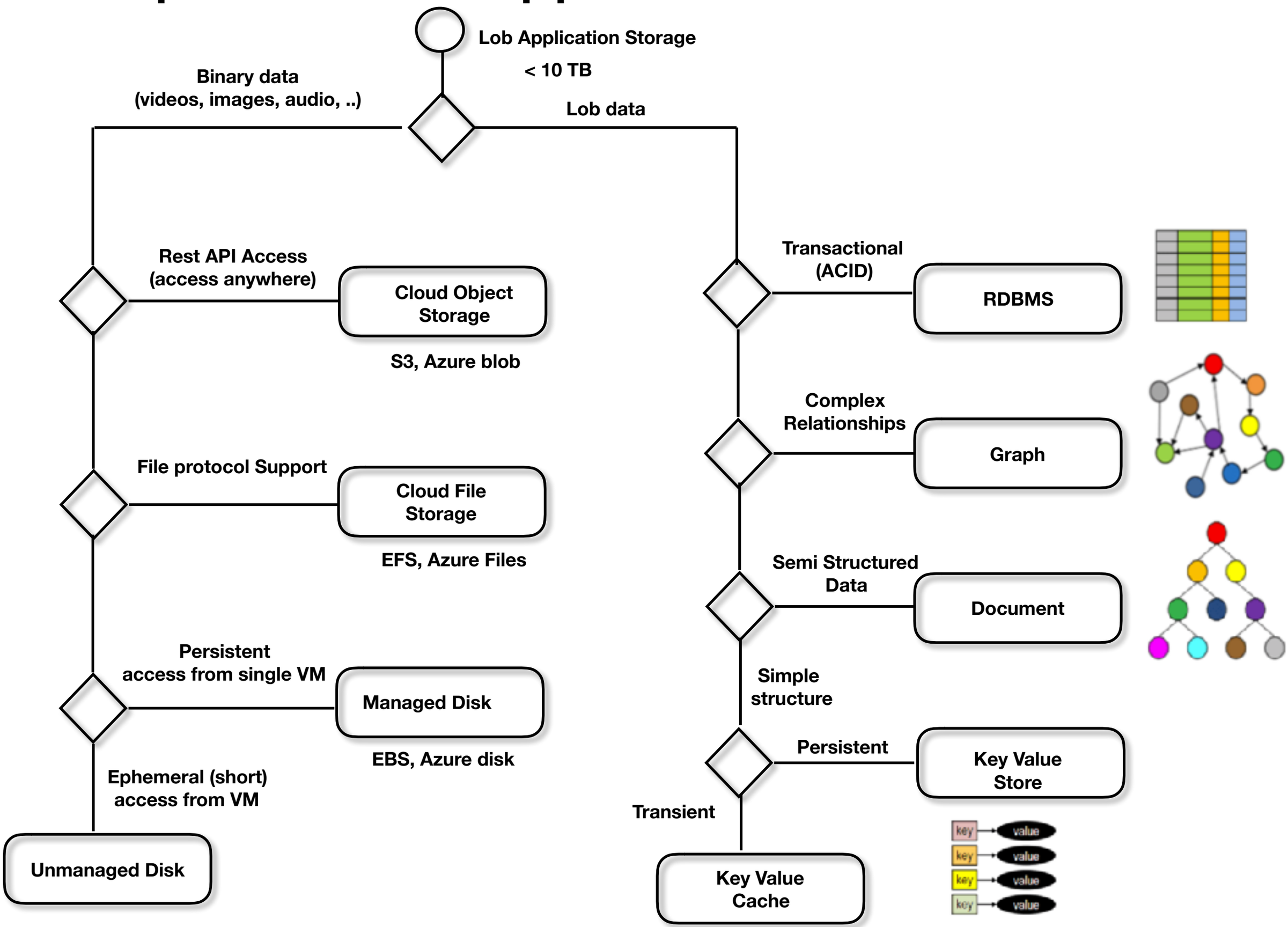
OLAP



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

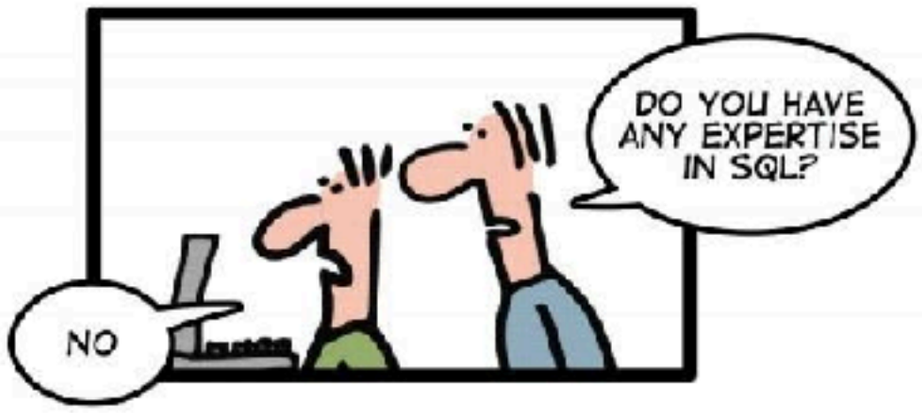
scgupta.me
twitter.com/**scgupta**
linkedin.com/in/**scgupta**

Step 1. Choose App Persistence Mechanism



Object Store	(rest) File
File Storage	(smb) File
• Rdbms	• **Postgres**, MySql
• Key-Value Stores:	• Oracle Coherence, **Redis**, Kyoto Cabinet
• Wide Column Databases:(big table)	• Apache HBase, Apache **Cassandra**
• Document Databases (tree):	• **MongoDB**, CouchDB
• Full Text Search Engines:	• **Elastic Search**, Apache Lucene, Apache Solr
• Graph Databases:	• **neo4j**, FlockDB
• Time Series Databases:	• **Prometheus,influx**
Multi Model	Cosmos
DataLake	**Azure Blob** **AWS S3**

HOW TO WRITE A CV

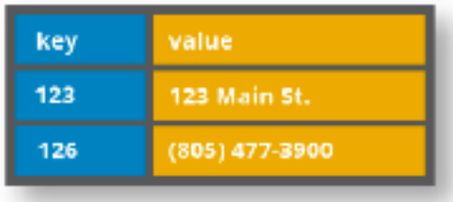


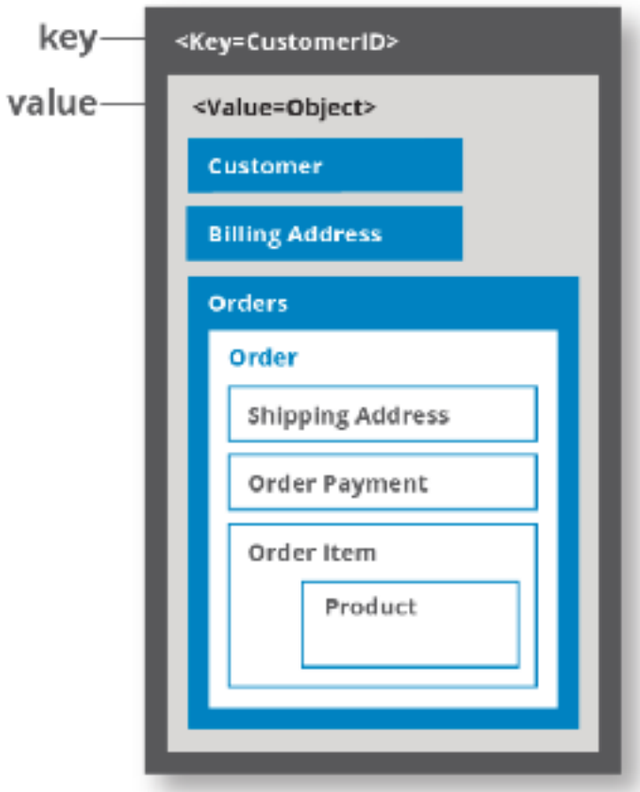
Leverage the NoSQL boom

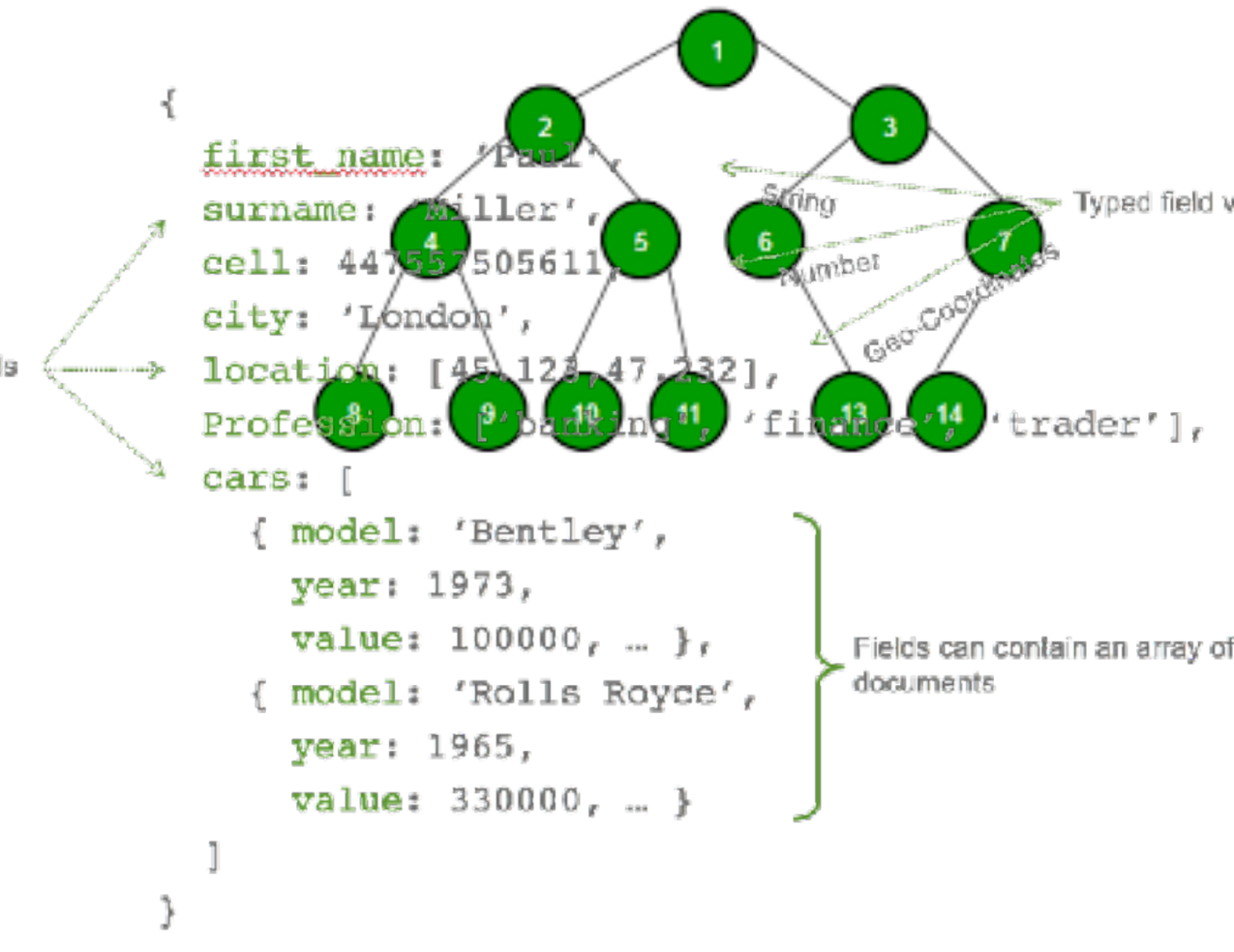
Rdbms (Referential Integrity)

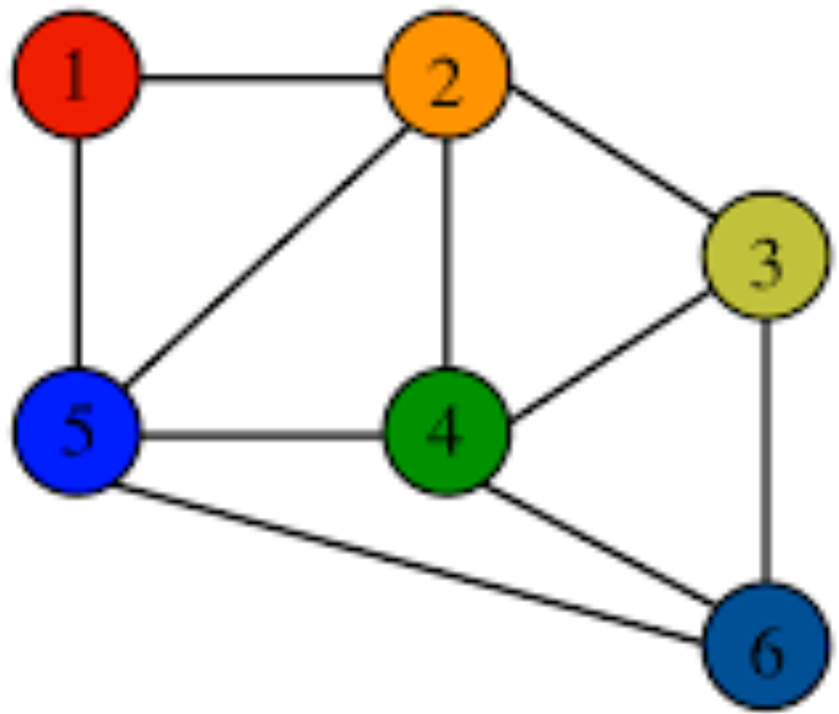
tblPerson			
ID	Name	Email	GenderID
1	Jade	j@j.com	2
2	Mary	m@m.com	3
3	Martin	ma@ma.com	1
4	Rob	r@r.com	NULL
5	May	may@may.com	2
6	Kristy	k@k.com	NULL

tblGender	
ID	Gender
1	Male
2	Female
3	Unknown









Row-oriented

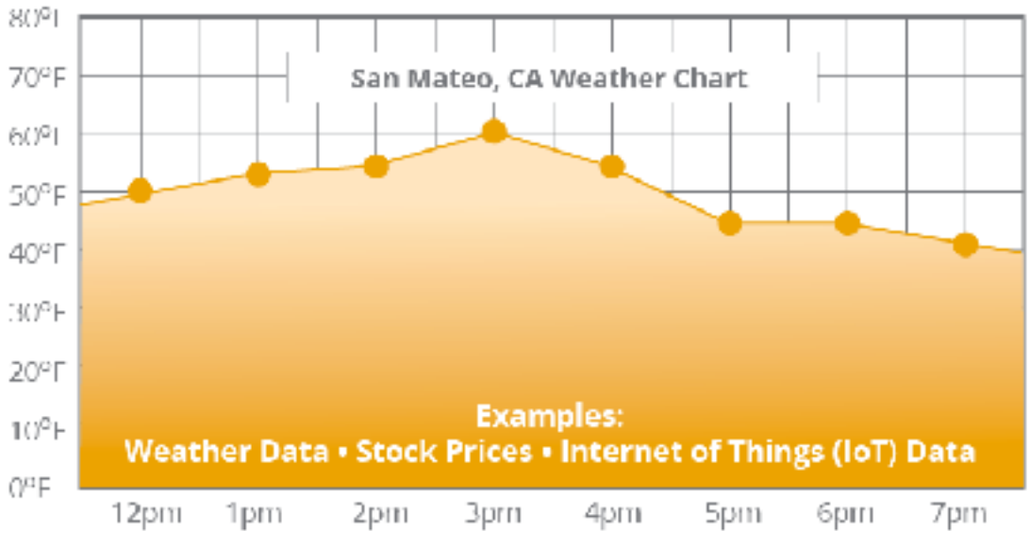
ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

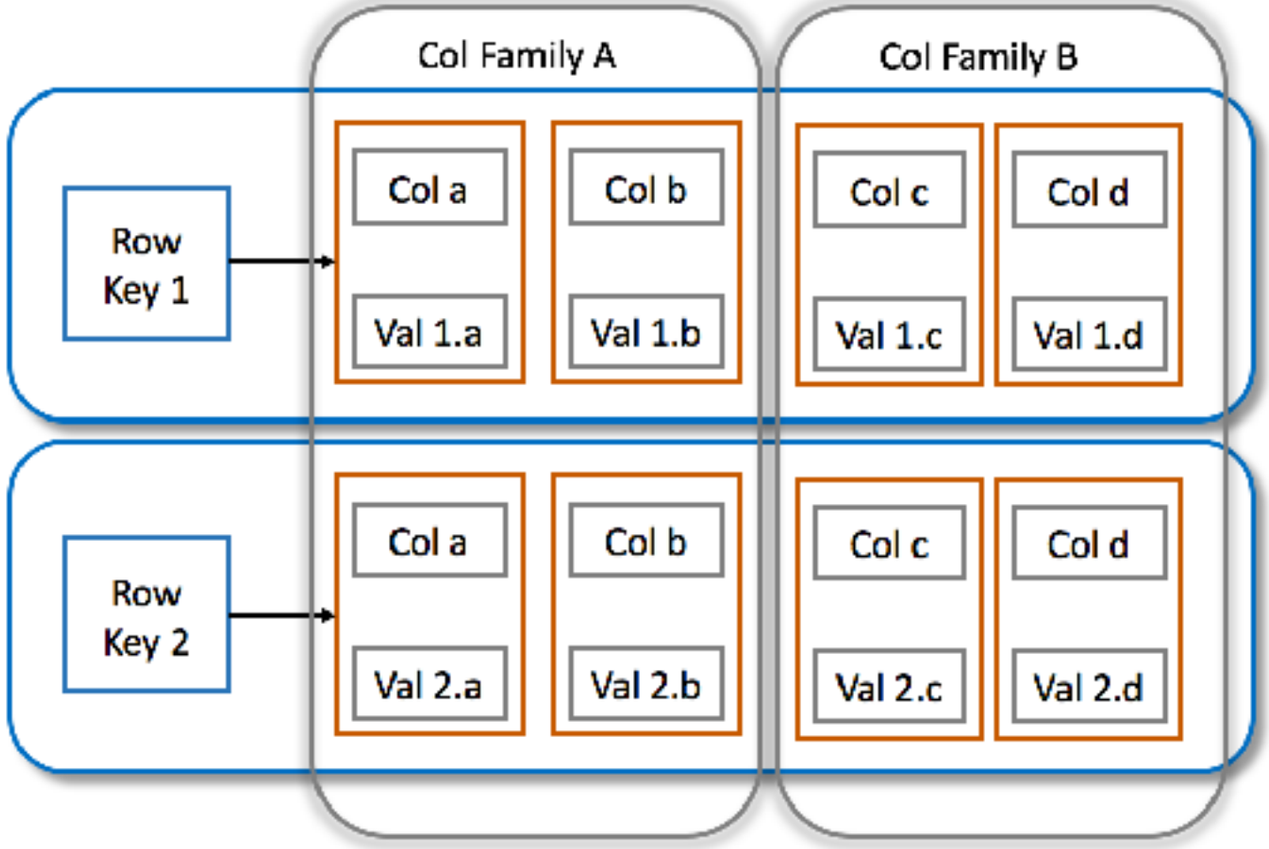
Name	ID
John	001
Karen	002
Bill	003

Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003



A Time Series Database is a database that contains data for each point in time.



Text Db

• Rdbms	• **Postgres**, MySql
• Key-Value Stores:	• Oracle Coherence, **Redis**, Kyoto Cabinet
• Wide Column Databases:	• Apache HBase, Apache **Cassandra**
• Document Databases:	• **MongoDB**, CouchDB
• Full Text Search Engines:	• **Elastic Search**, Apache Lucene, Apache Solr
• Graph Databases:	• **neo4j**, FlockDB
• Time Series Databases:	• **Prometheus,influx**
Multi Model	Cosmos

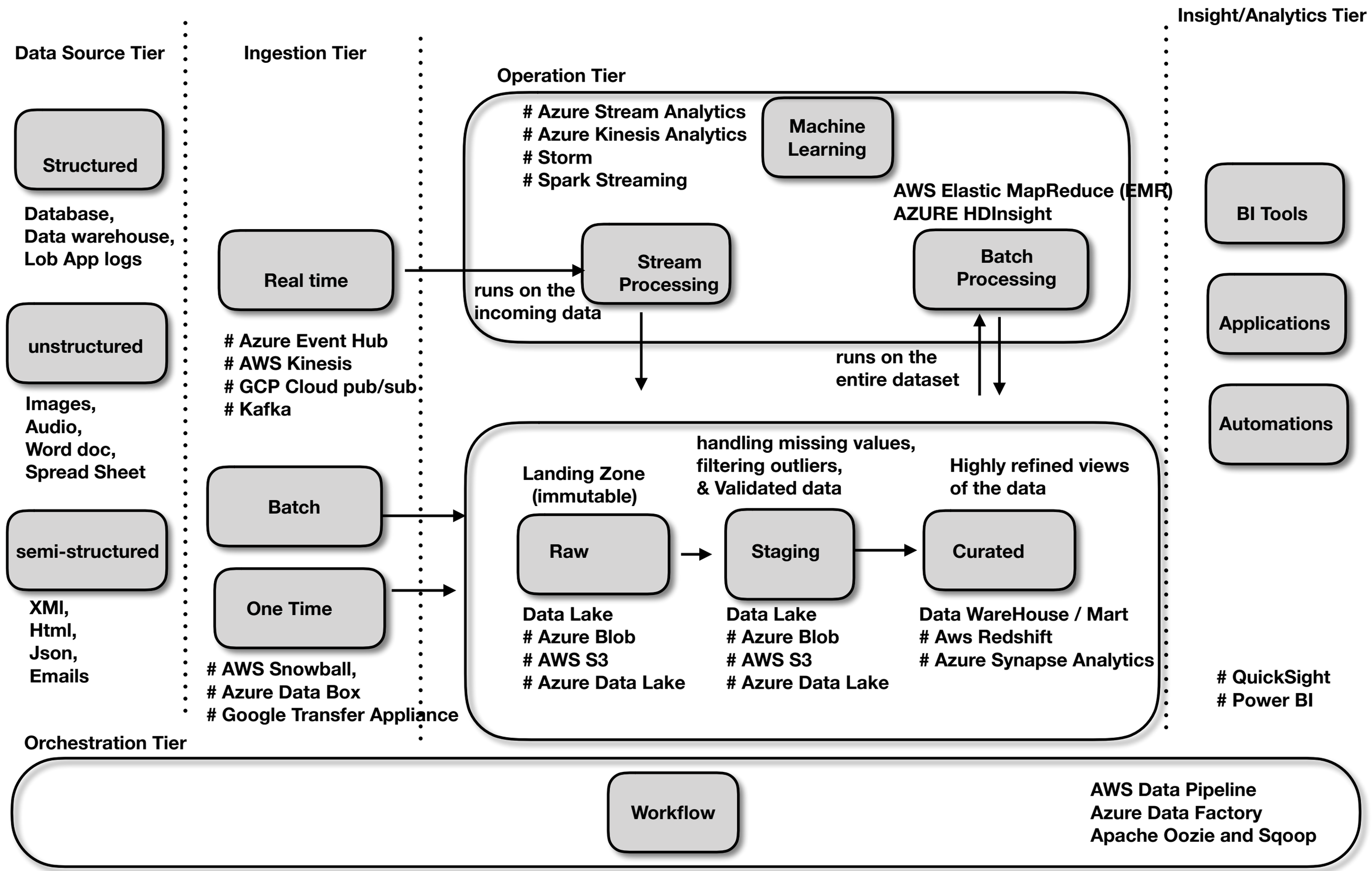
- Twitter uses Redis to deliver your Twitter timeline
- Pinterest uses Redis to store lists of users, followers, unfollowers, boards, and more
- Coinbase uses Redis to enforce rate limits and guarantee correctness of Bitcoin transactions
- Quora uses Memcached to cache results from slower, persistent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

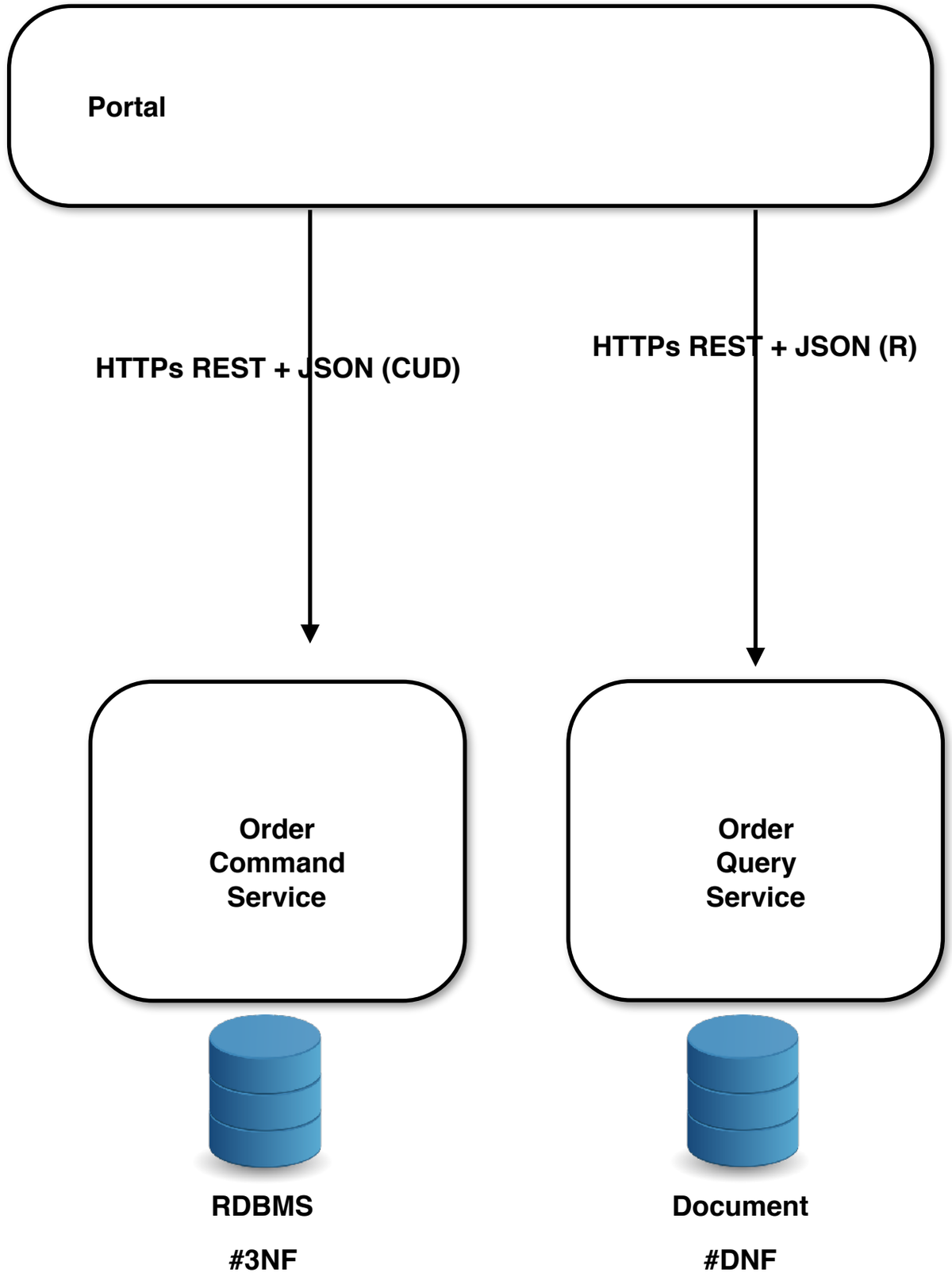
- Spotify uses Cassandra to store user profile attributes and metadata about artists, songs, etc. for better personalization
- Facebook initially built its revamped Messages on top of HBase, but is now also used for other Facebook services like the Nearby Friends feature and search indexing
- Outbrain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

ToDo Api
(Api Application)

Document Db

Step 2. Choose Analytical Persistence Mechanism




Portal
HTTPs REST + JSON (CUD)
HTTPs REST + JSON (R)
Order
Command
Service
Order
Query
Service
RDBMS
#3NF
Document
#DNF

command vs Event
event sourcing
eventual consistency
materialised view
SAGA (compensation)

Portal

HTTPs REST + JSON (R)

Front controller
Ssl termination
AuthN
AuthZ
Metering
Throttling
Aggregate
Protocol translation
Versioning
Caching
...

WAF

Reverse Proxy/ API Gateway

Msg

Grpc

Stream Analytics

Domain Event Bus (command / Events)

OrderCreated

No Stock

CreateOrder

CreateOrder
No Stock

Event Service

Order
Command
Service

Order
Query
Service

DWH Analytics

Data Lake
event store

Batch Analytics

Daily/weekly

RDBMS

#3NF

Document

#DNF

DWH
<<structured>>

Video

Audio

Db archive

Audit log

Audio

Debug log

Images

..

System logs

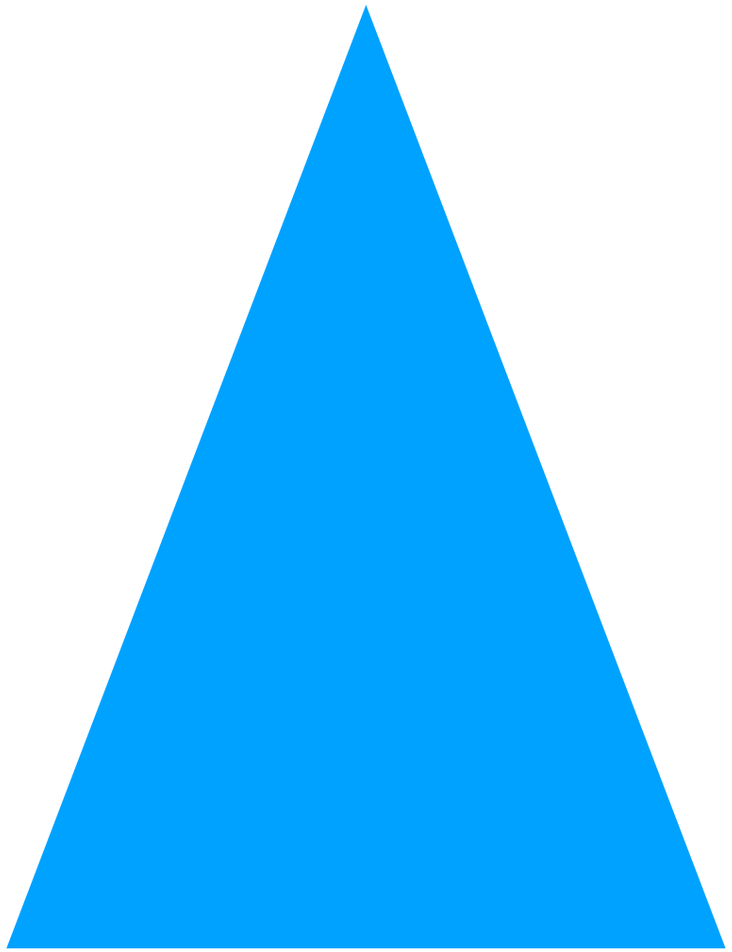
Data Lake

Column Store

TimeSeries Store

Object Store

Availability



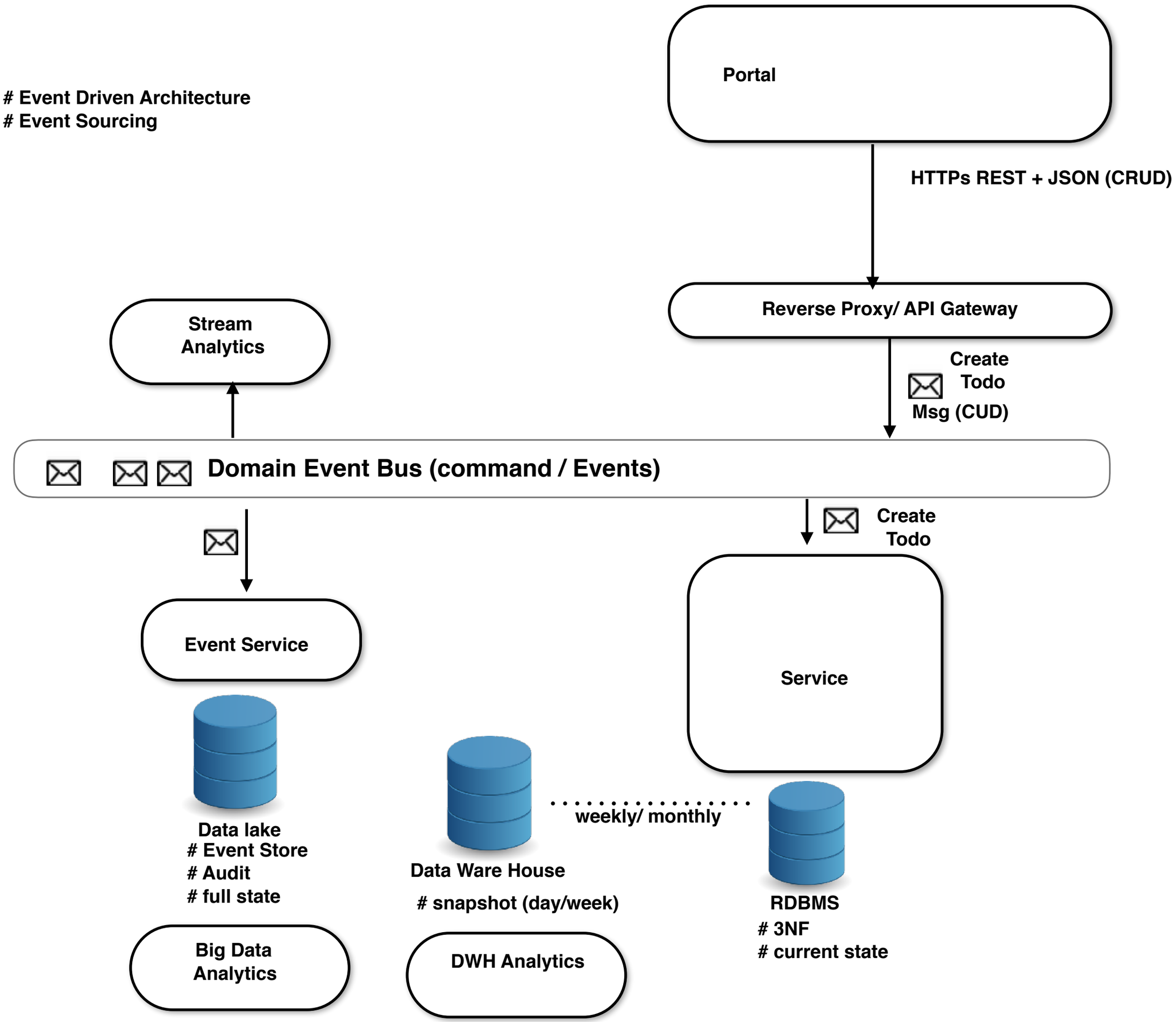
Consistency

Write

Read

Distributed

Event Driven Architecture
Event Sourcing



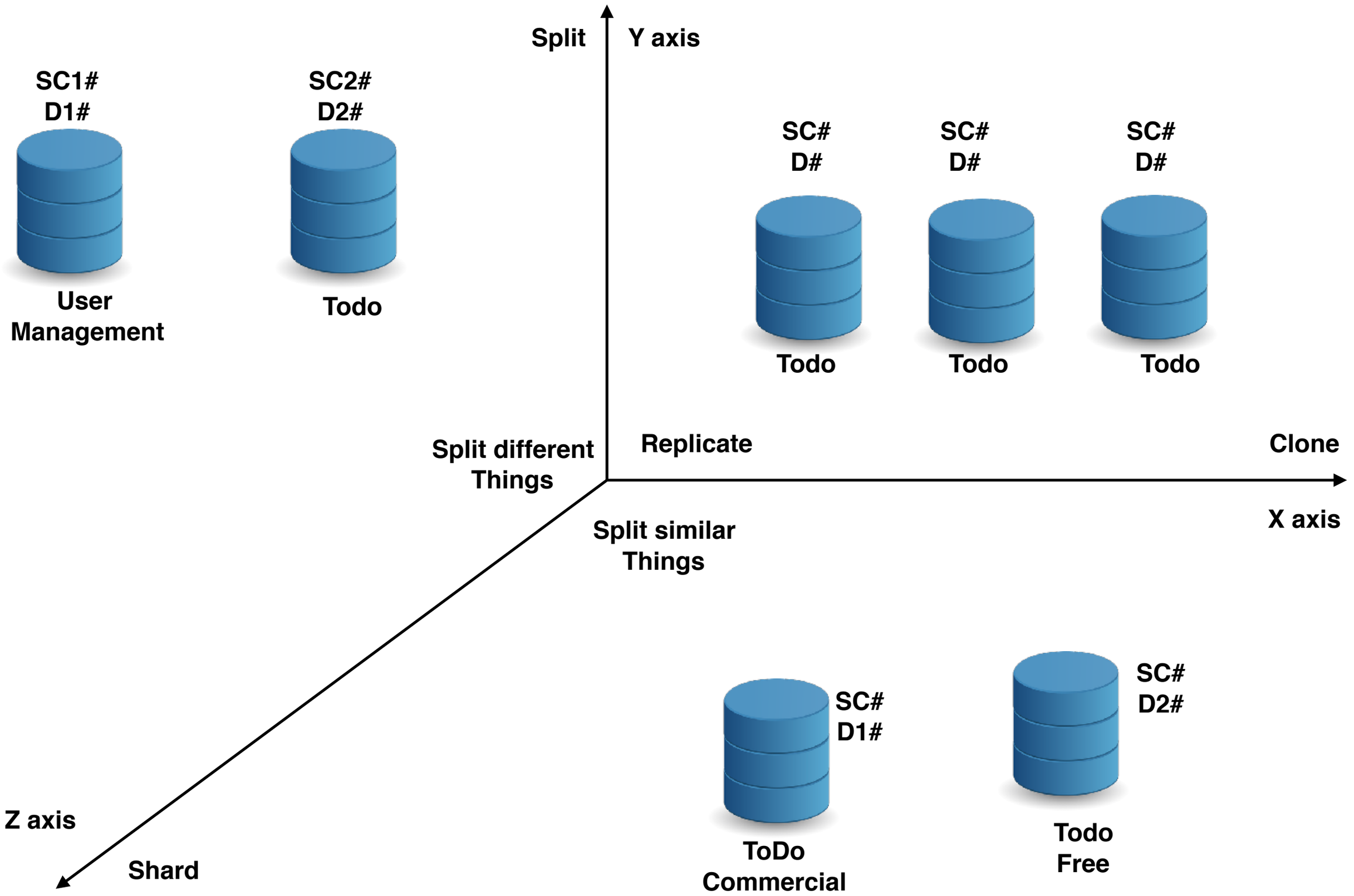
Data Lake

ToDo Prediction Service
(Batch Analytics)

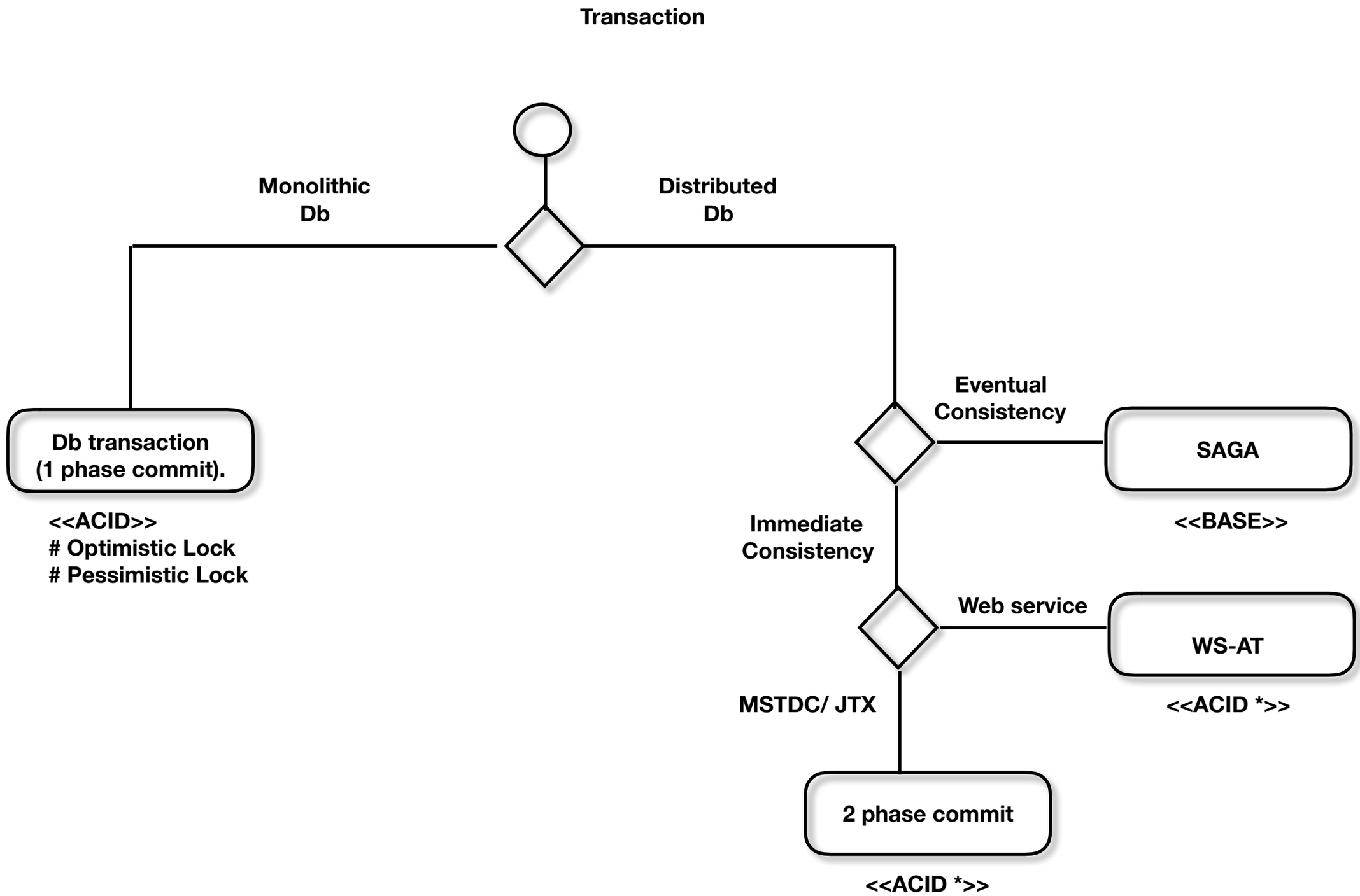
Step 3. Size the storage

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

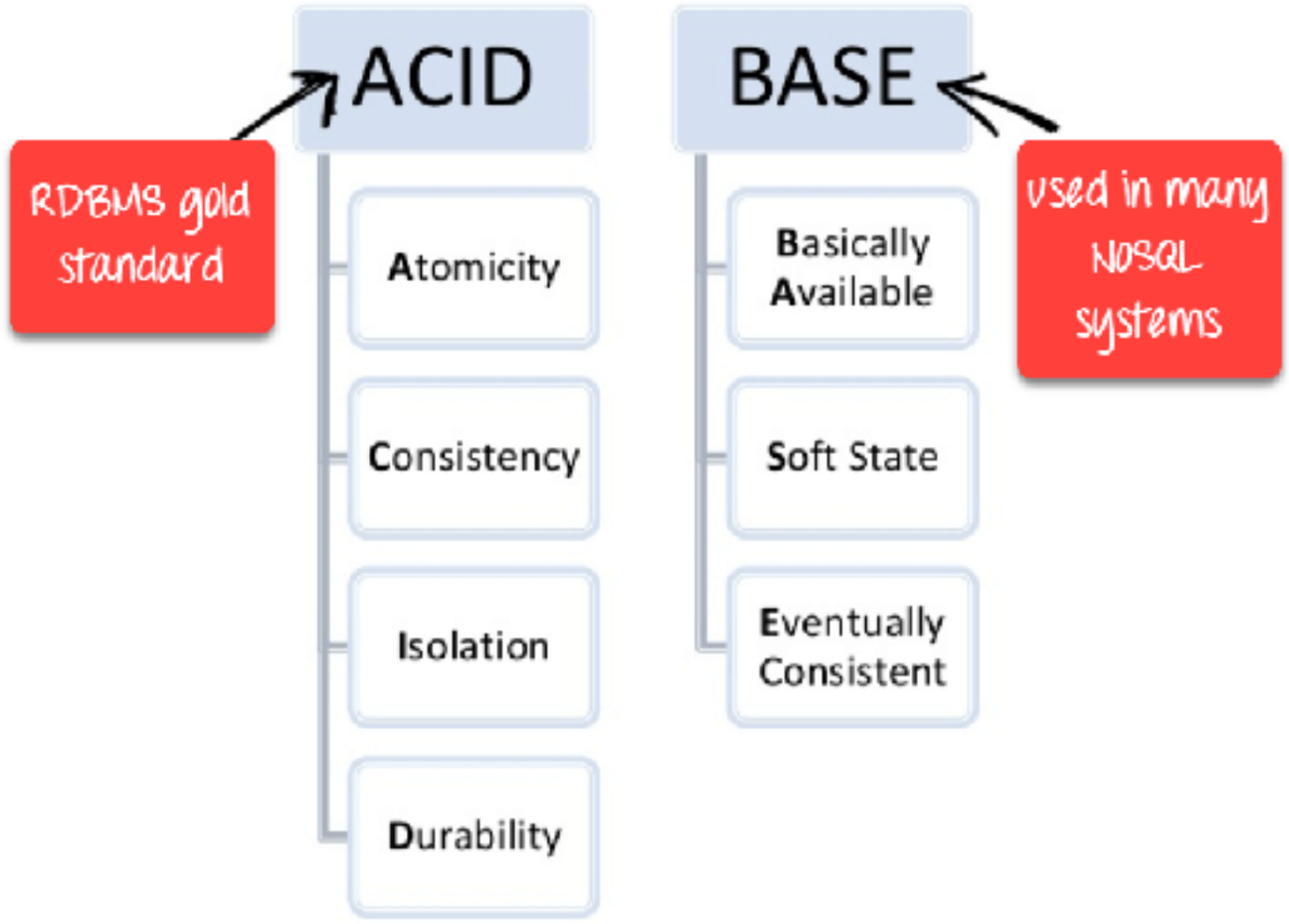
Scalability Cube - 50 rules for high Scalability



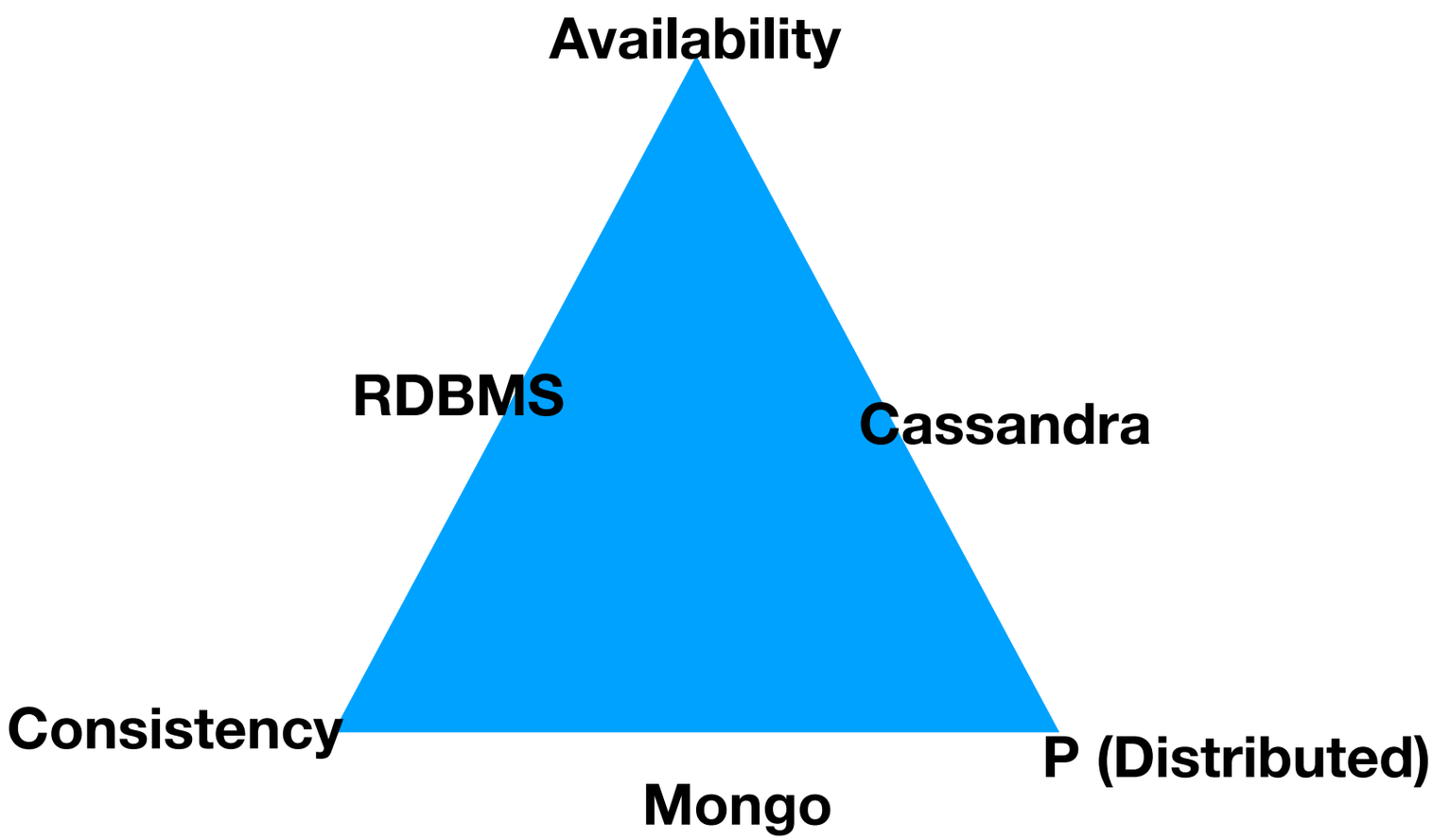
Step 4. Transaction and Concurrency

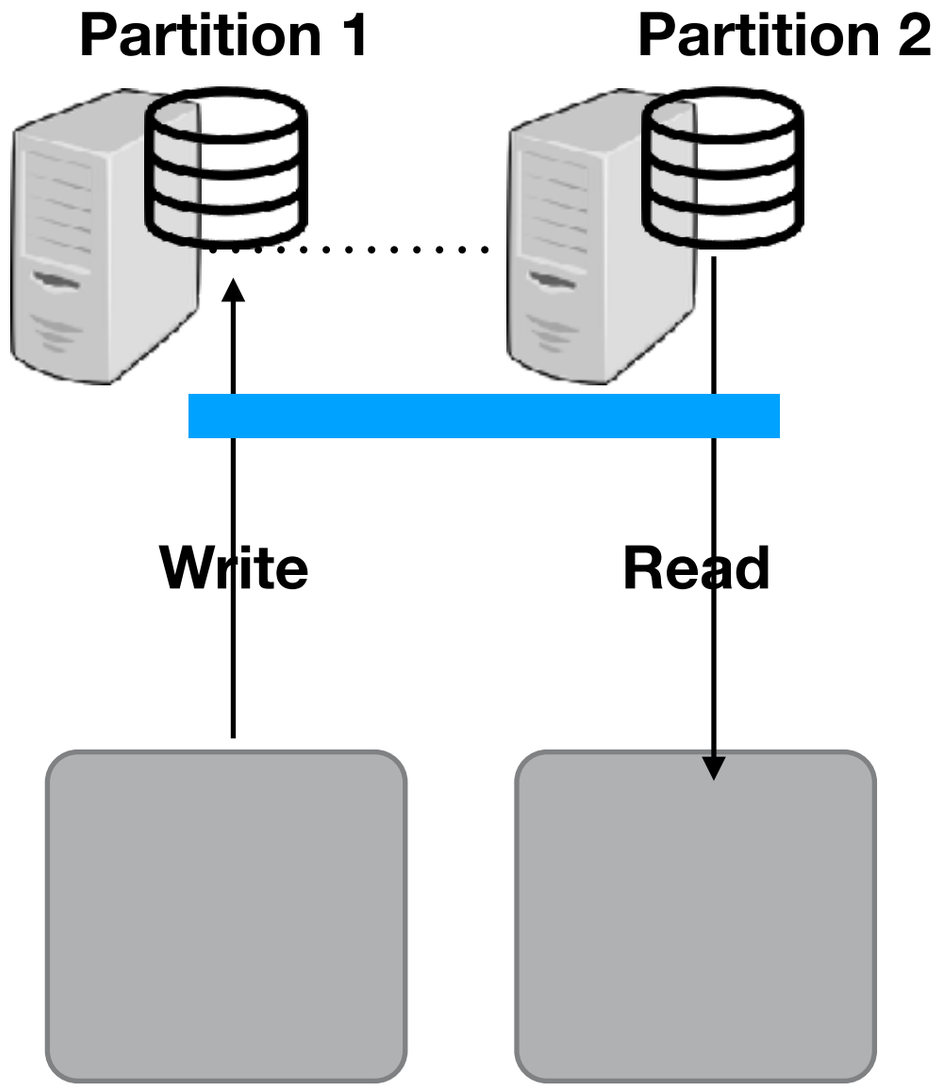


* 2PC **is not resilient to all possible failure configurations**.

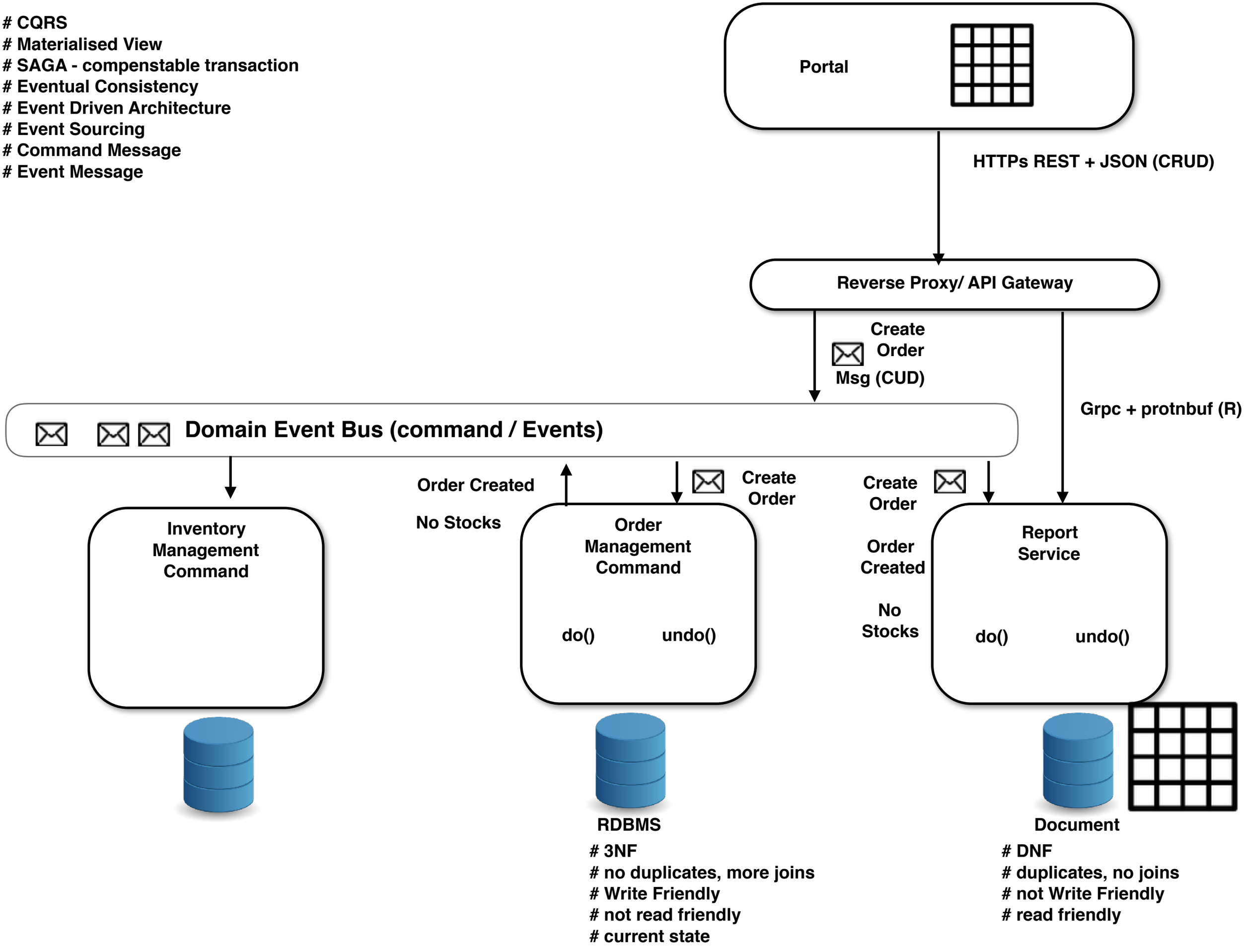

ACID
RDBMS gold standard
Atomicity
Consistency
Isolation
Durability
BASE
used in many NOSQL systems
Basically Available
Soft State
Eventually Consistent

CAP Theorem


Availability
RDBMS
Cassandra
Consistency
P (Distributed)
Mongo


Partition 1
Partition 2
Write
Read

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



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
```

Step 5. 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

ToDo
Prediction Service

Event Service

Todo
Service

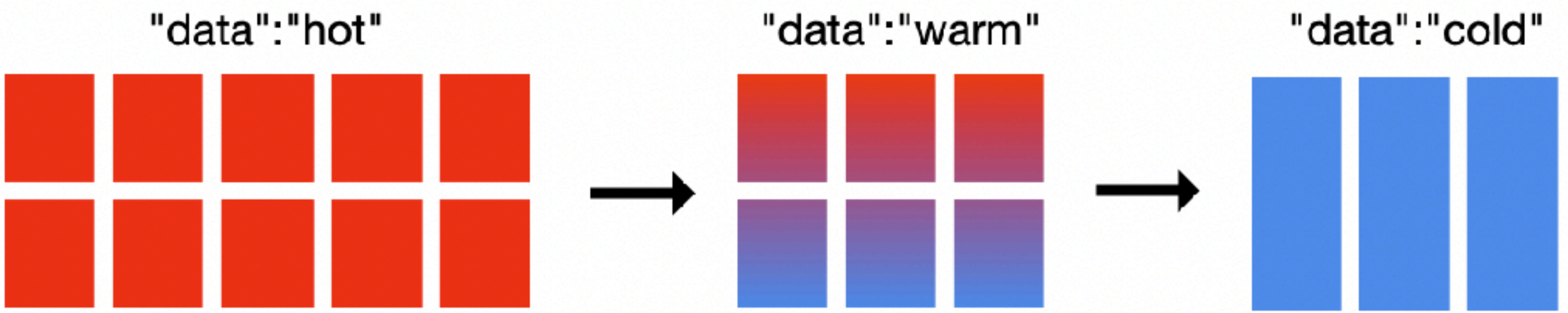
Data Lake

Event Store

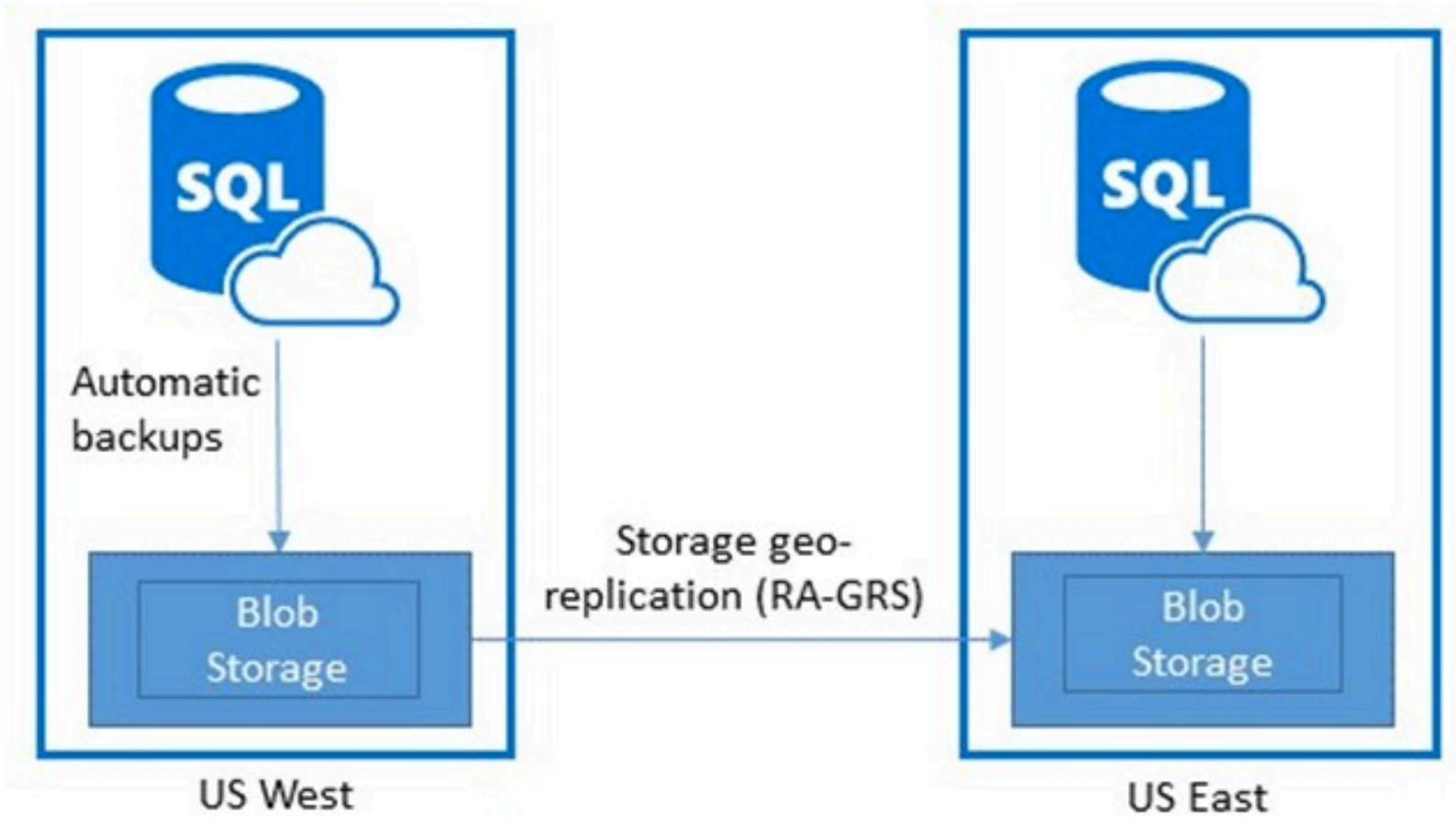
RDBMS

Lob Data

Step 6. Archive



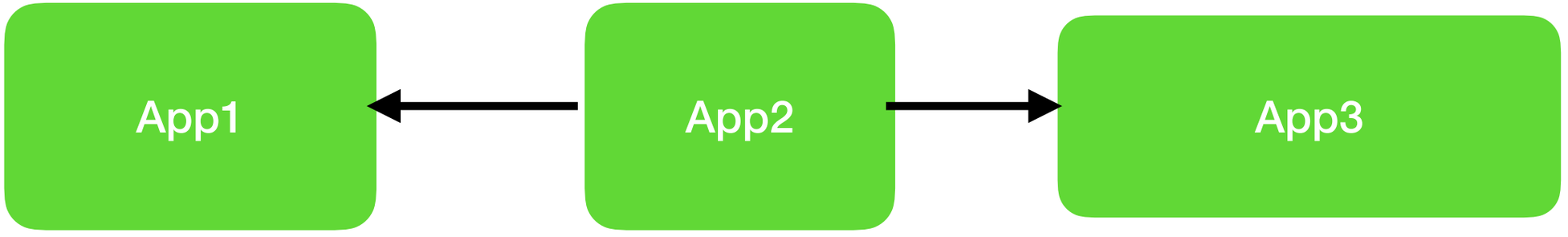
Step 7. Backup and Recovery



* Full backups
* differential backups
* transaction log backups

* Standard Geo-Replication
* Active Geo-Replication

9. Integration View

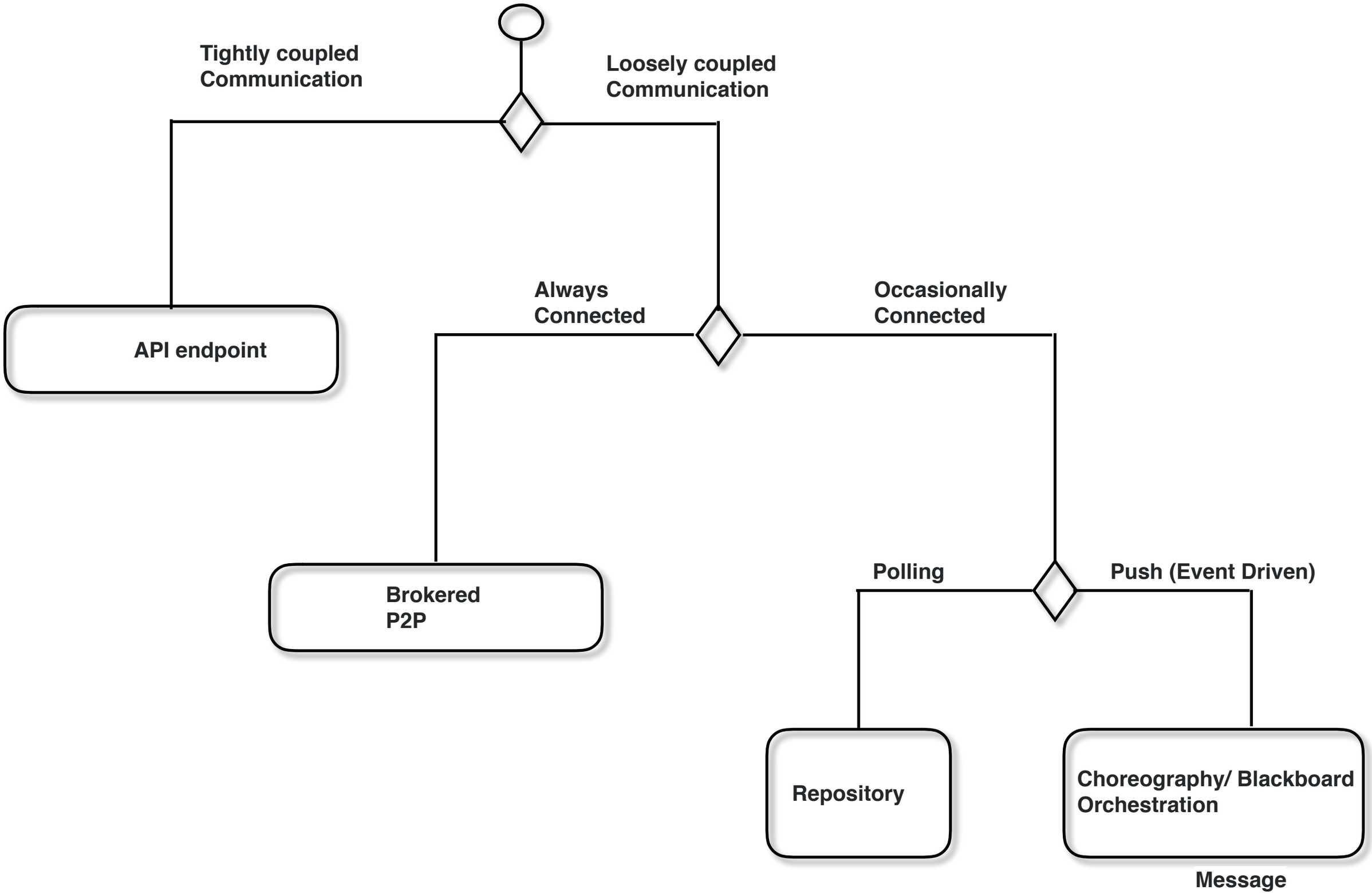


	Rabbitmq	Kafka	Redis
Scale	50K msg per second	1 million/ sec	1 million/ sec
Transient messages	Yes	No	Y
Persistent yes	Yes	Yes	Y
Protocol	AMQP		
One to one	Y	N	Y
One to many	Y	Y	Y
Advanced Message Queueing Protocol-based routing	Y	N	N
consumption pattern	Pull+Push	Pull	
Client Libraries	Py, java, php,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

Step 2. Choose Communication Mechanism



ToDo Portal
(Single Page Application)

Https(443)

ToDo API Service
(Api Application)

Document Db

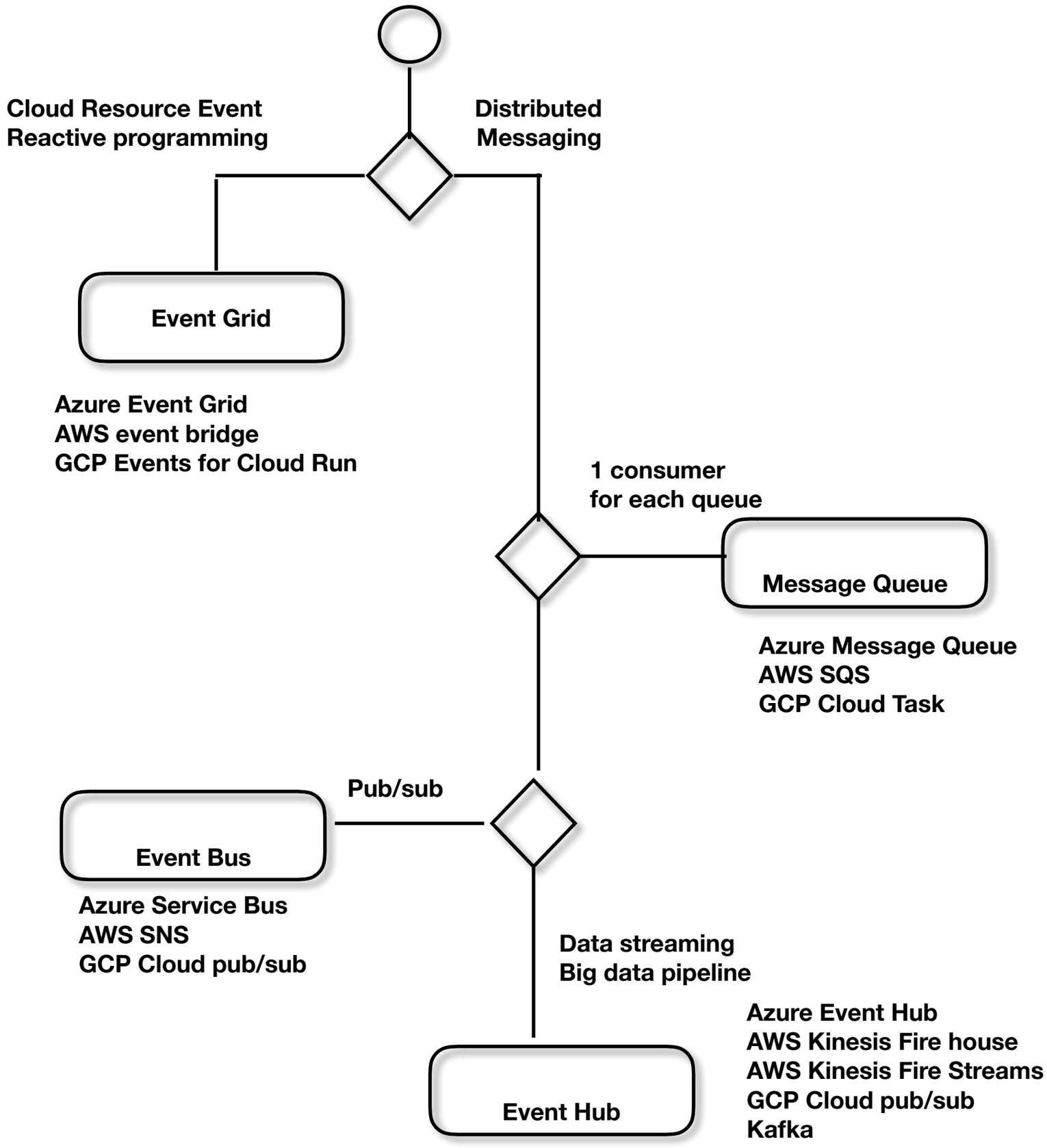
ToDo Calendar Service
(Background Application)

Message

Data Lake

ToDo Prediction Service
(Background Application)

Step 3. Choose Message Engine



ToDo Portal
(Single Page Application)

Https(443)

ToDo API Service
(Api Application)

Document Db

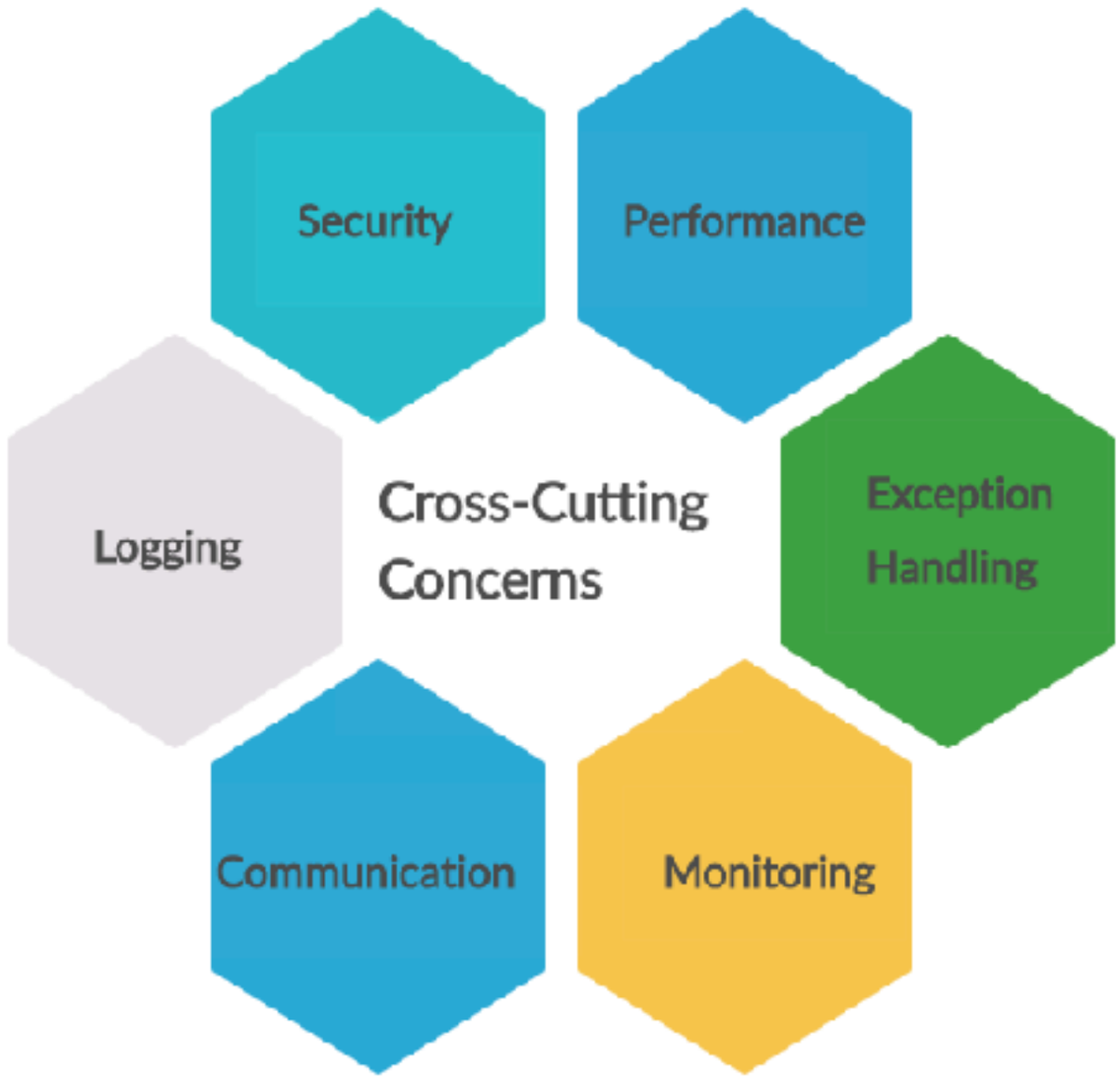
ToDo Calendar Service
(Background Application)

Message queue

Data Lake

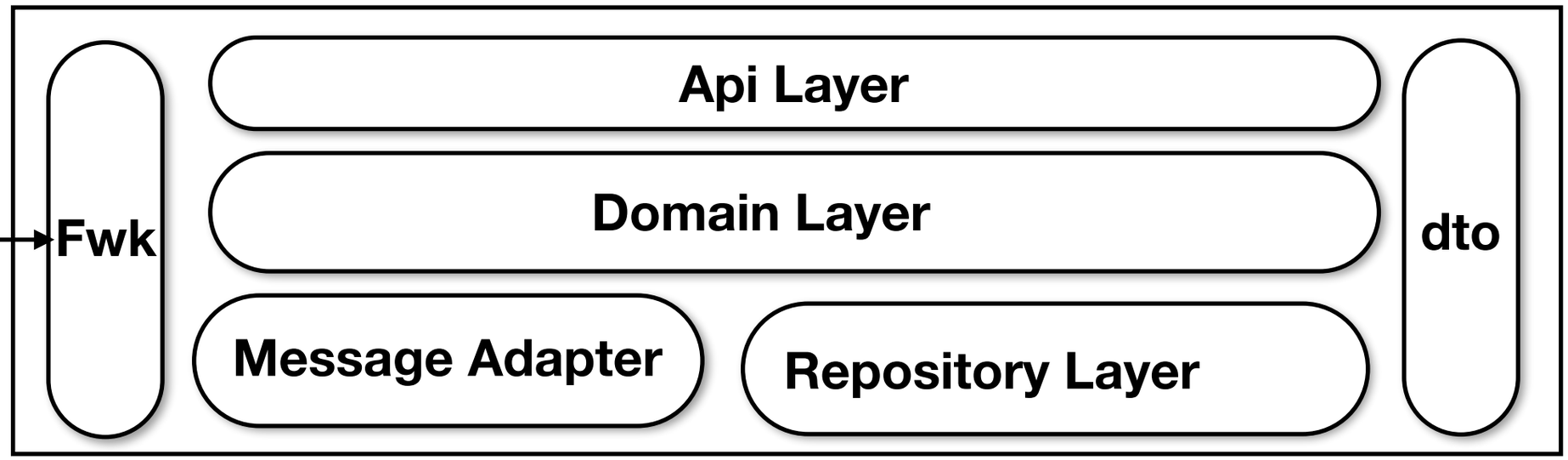
ToDo Prediction Service
(Background Application)

10. Cross cutting concerns

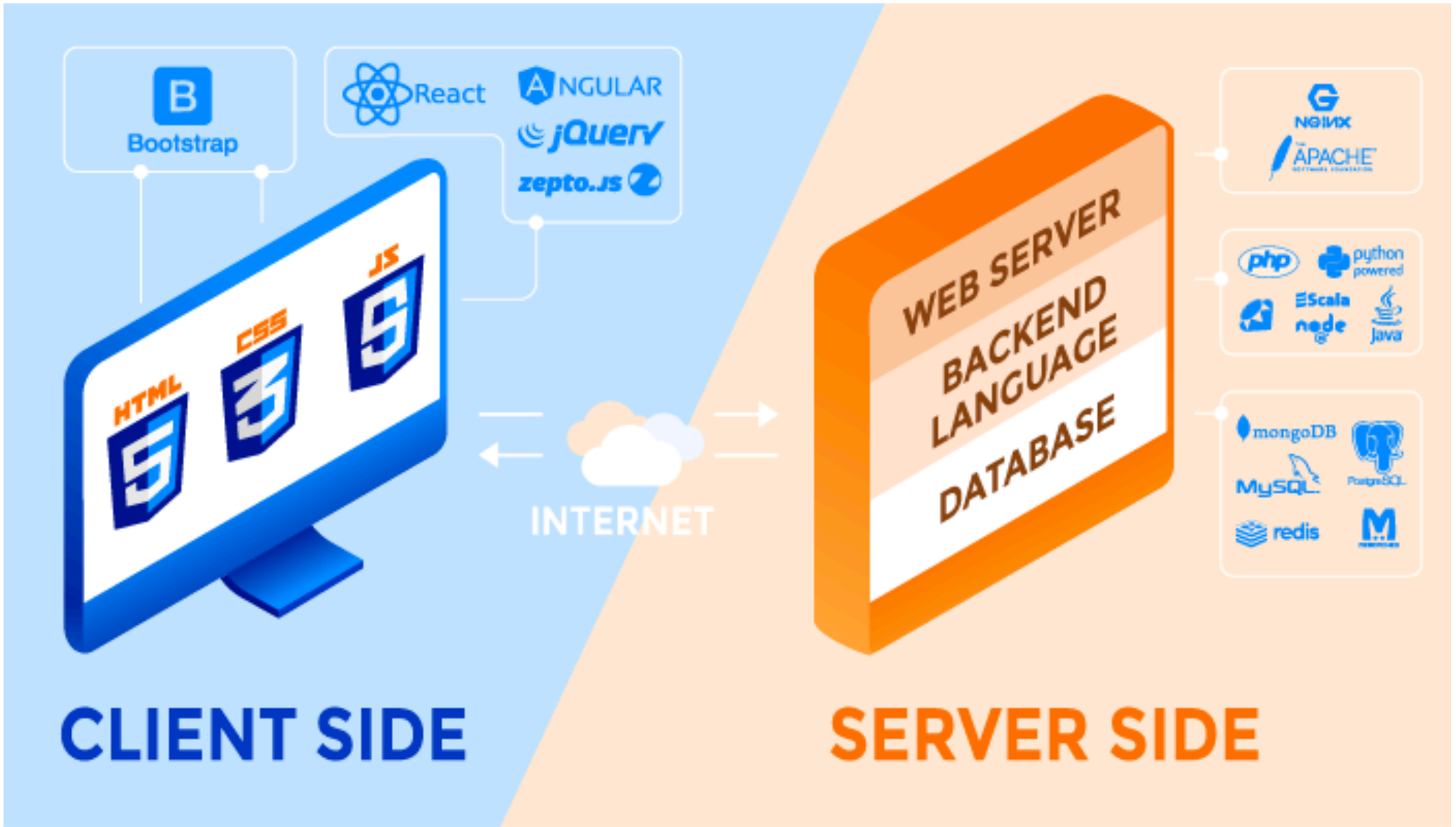


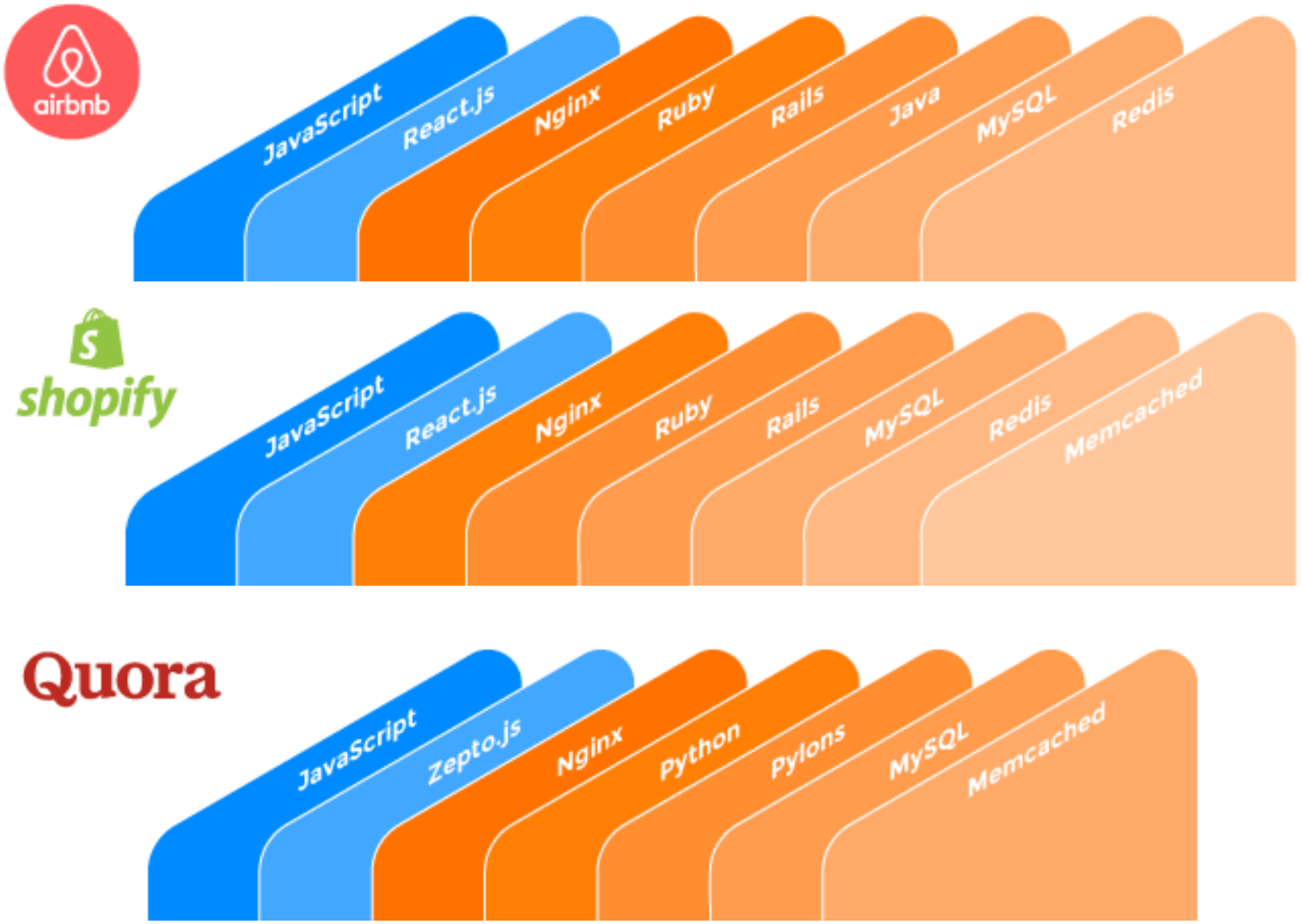
The **crosscutting concern** is a concern which is needed in almost every module of an application.

Logging,
Exception Handling,
Input Validation,
Syncronization,



11. Technology View



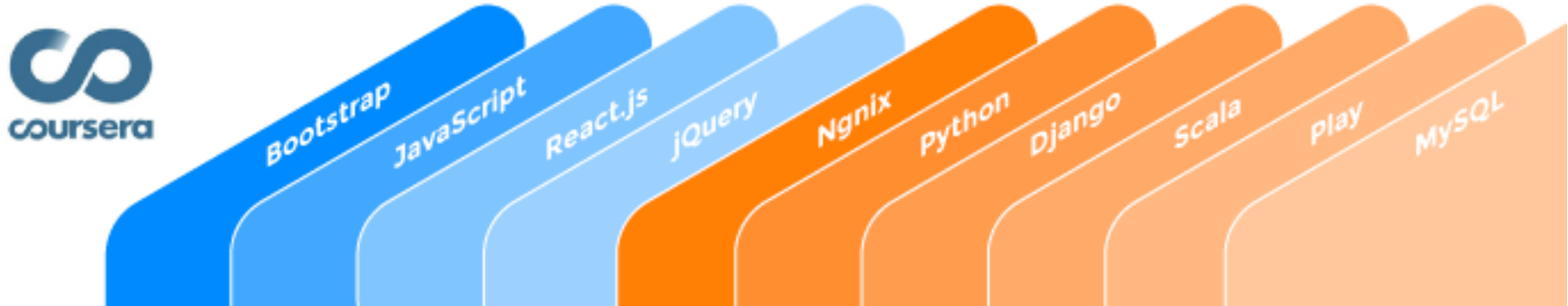

airbnb
JavaScript
React.js
Nginx
Ruby
Rails
Java
MySQL
Redis

shopify
JavaScript
React.js
Nginx
Ruby
Rails
MySQL
Redis
Memcached

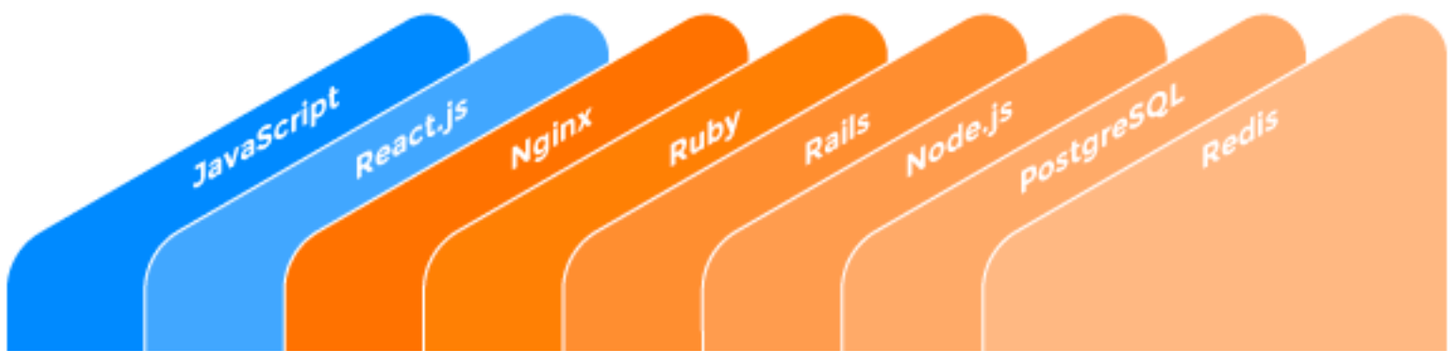
Quora
JavaScript
Zepto.js
Nginx
Python
Pylons
MySQL
Memcached

JavaScript
React.js
Nginx
Python
Django
PostgreSQL
Redis

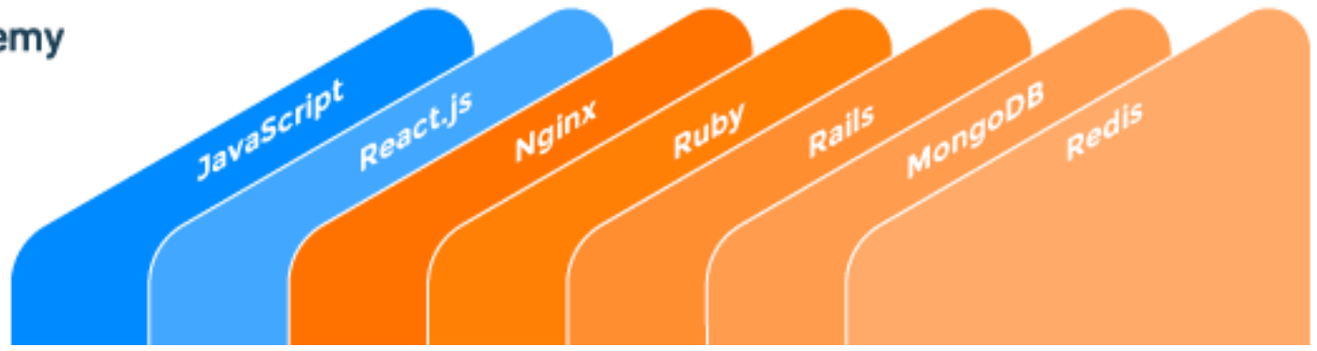
[Instagram]


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

Product Hunt

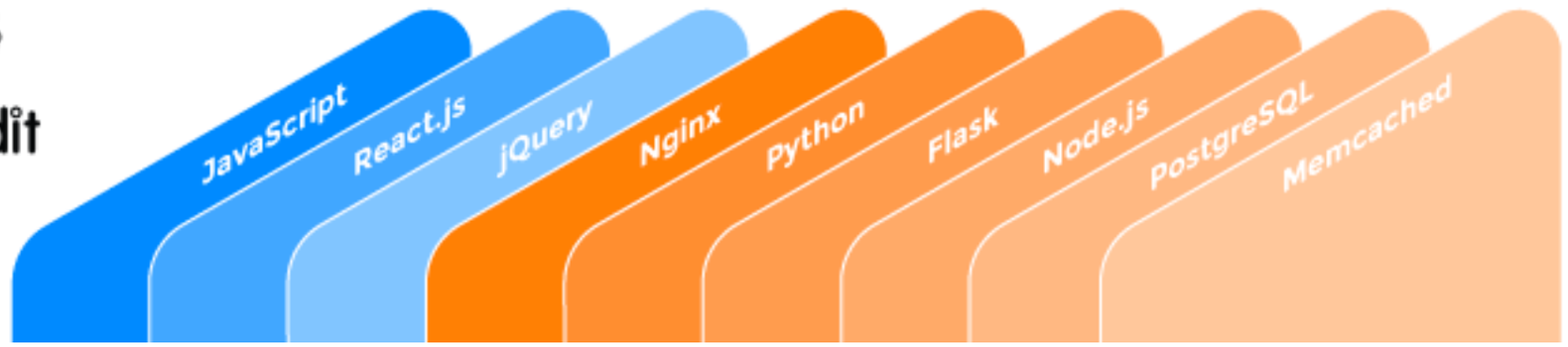


codecademy



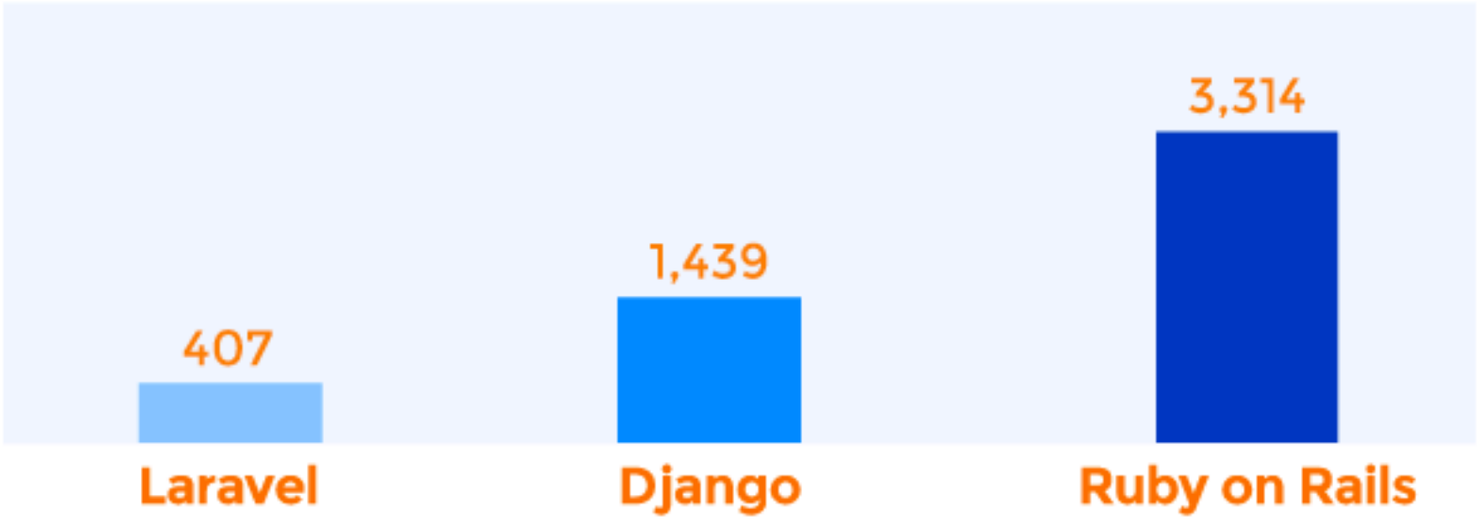
Pinterest

reddit



NUMBER OF CONTRIBUTORS ON GITHUB

JUNE 2017



AVERAGE DEVELOPER SALARIES IN THE US

BY TECHNOLOGY

Ruby	Python	PHP	JAVA
$55.78/hour	$55.19/hour	$42.68/hour	$54.13/hour

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

Azure Data Lake

ToDo Prediction Service
(Background Application)

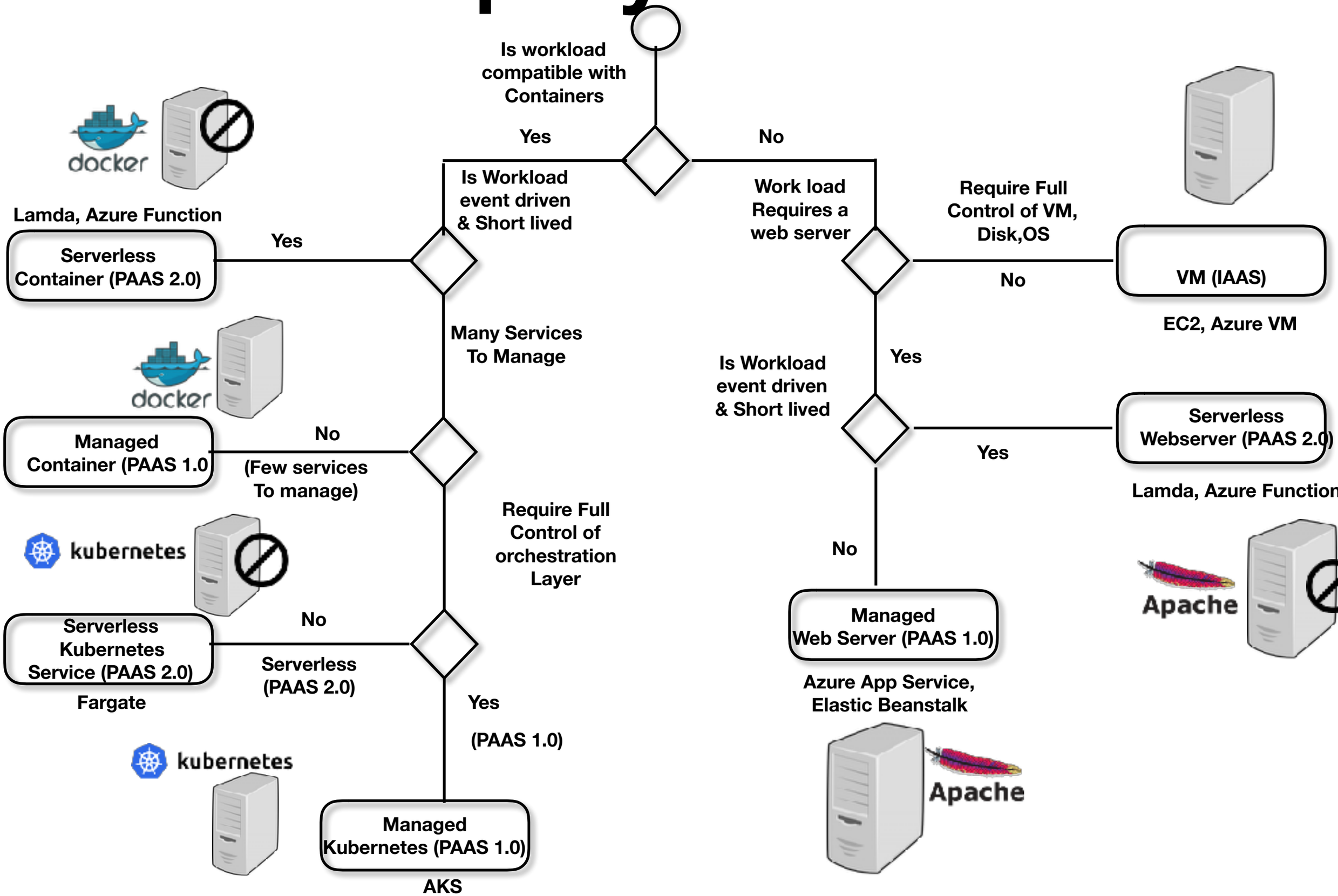
python

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

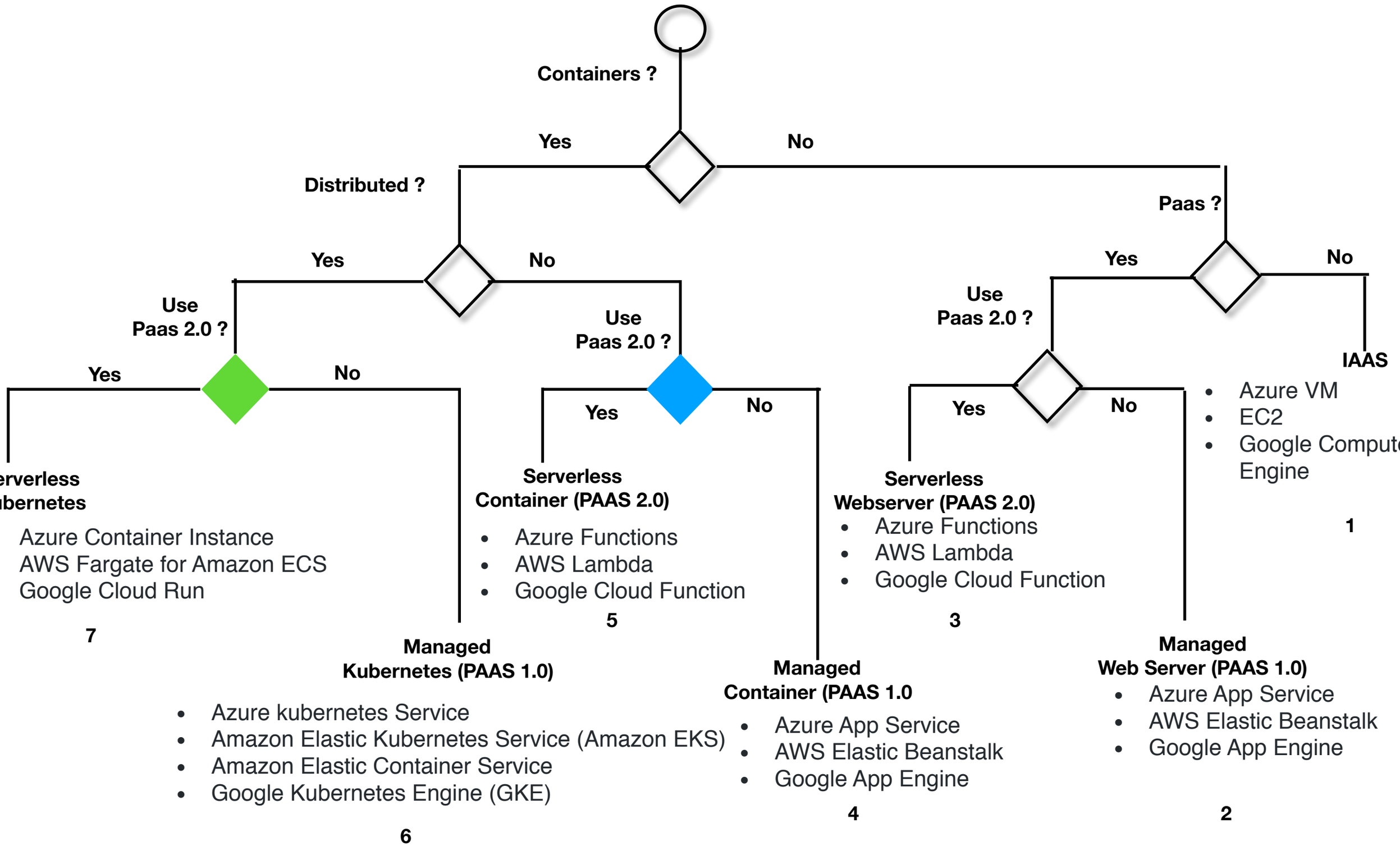
Copy

Reference architecture

12. 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

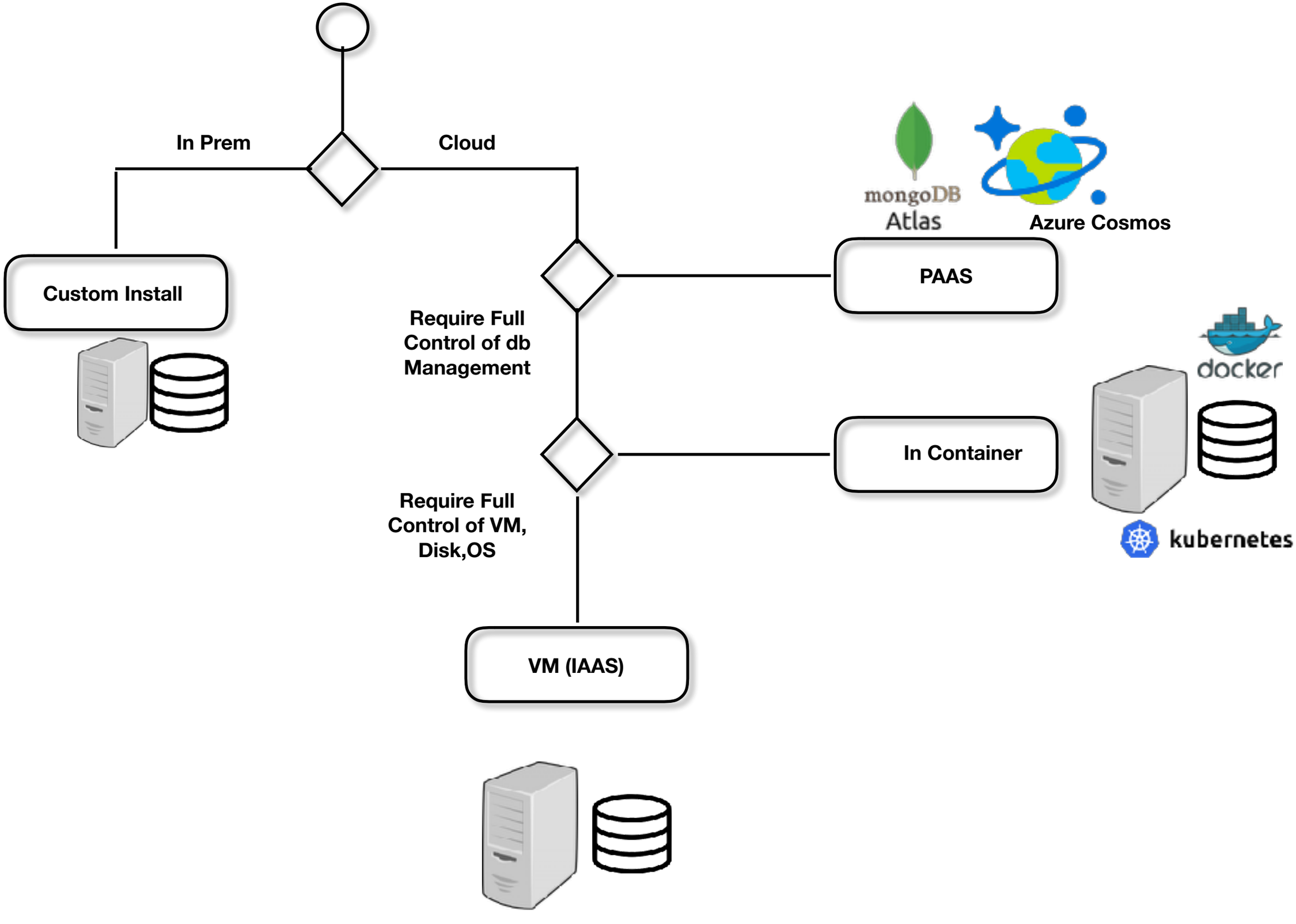
Azure Queue

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

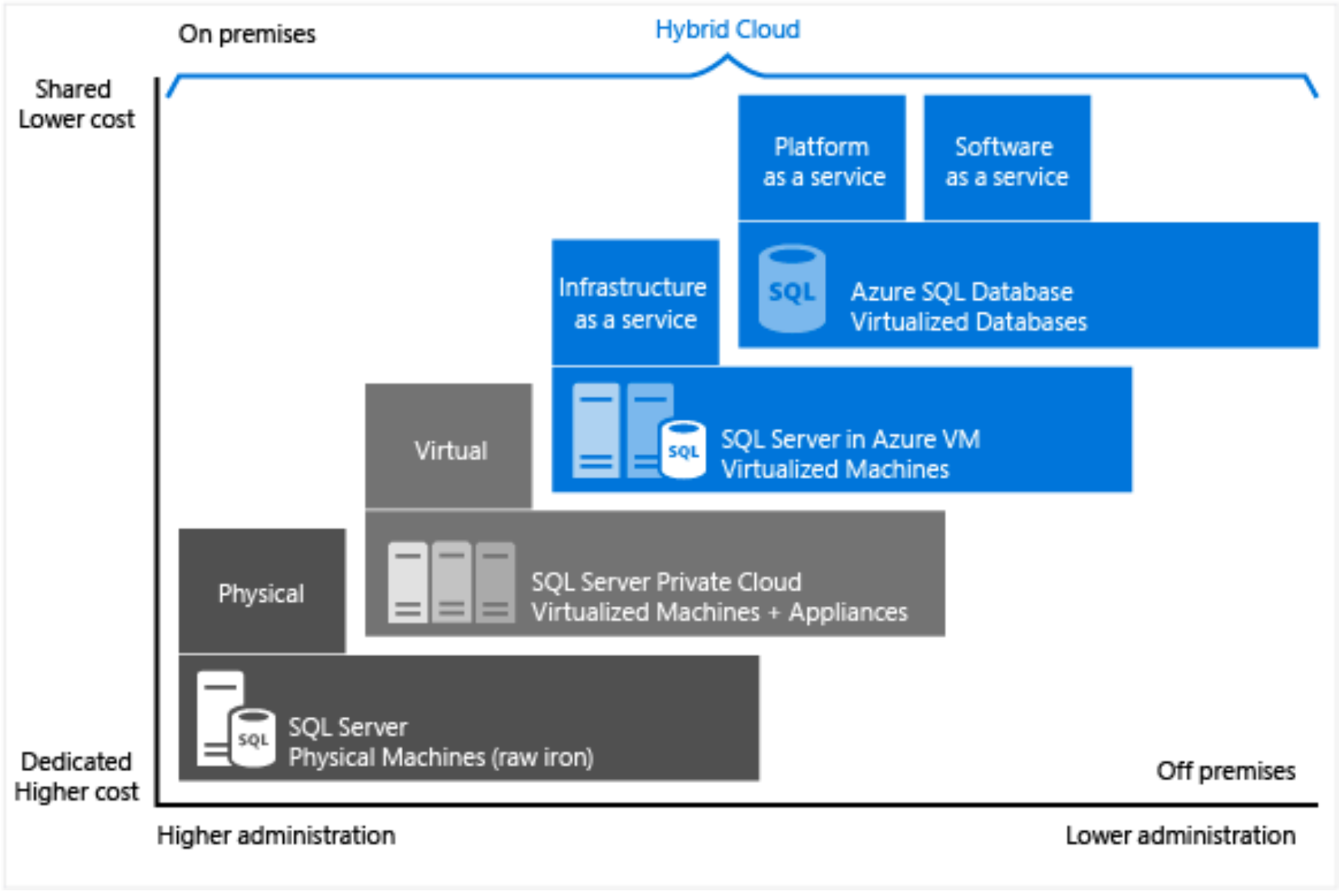
Document Db

mongoDB

mongoDB
Atlas

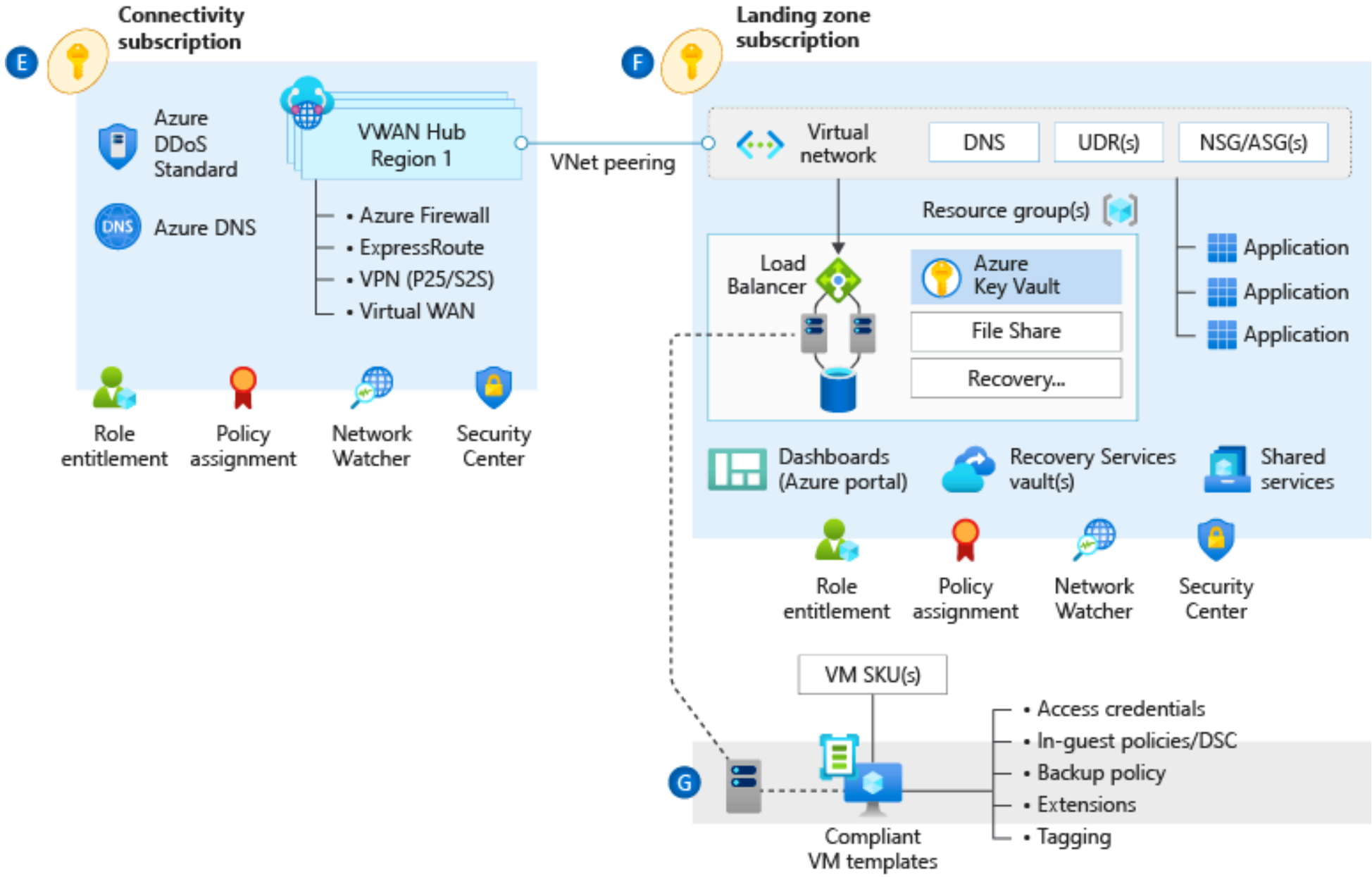
Azure Data Lake

Deploying Data Tier



13. Infrastructure View

Physical View



Public IP address

vnet

Network Load Balancer

asg
subnet

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

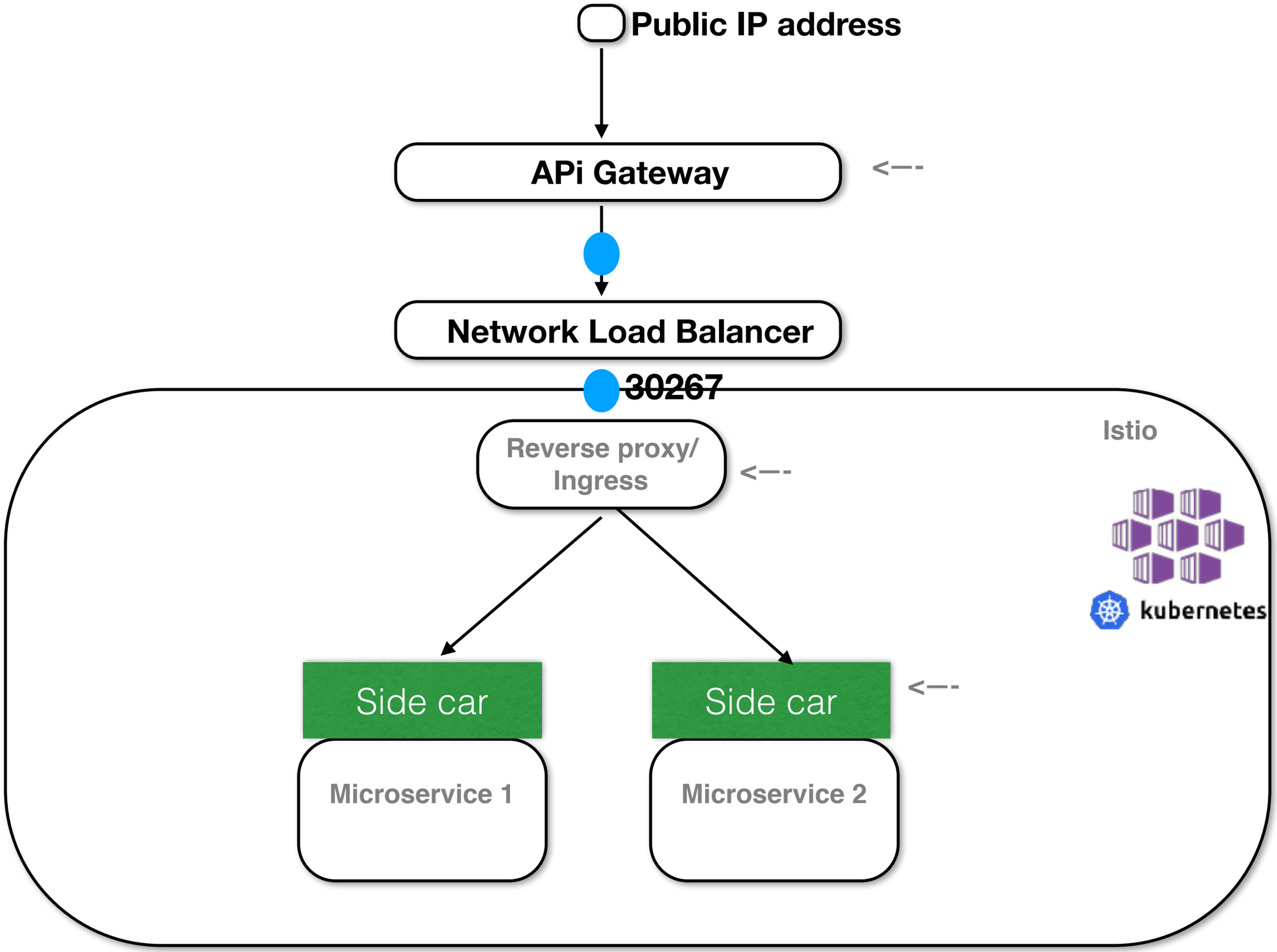
ToDo Prediction Service
(Background Application)
python

Document Db
mongoDB

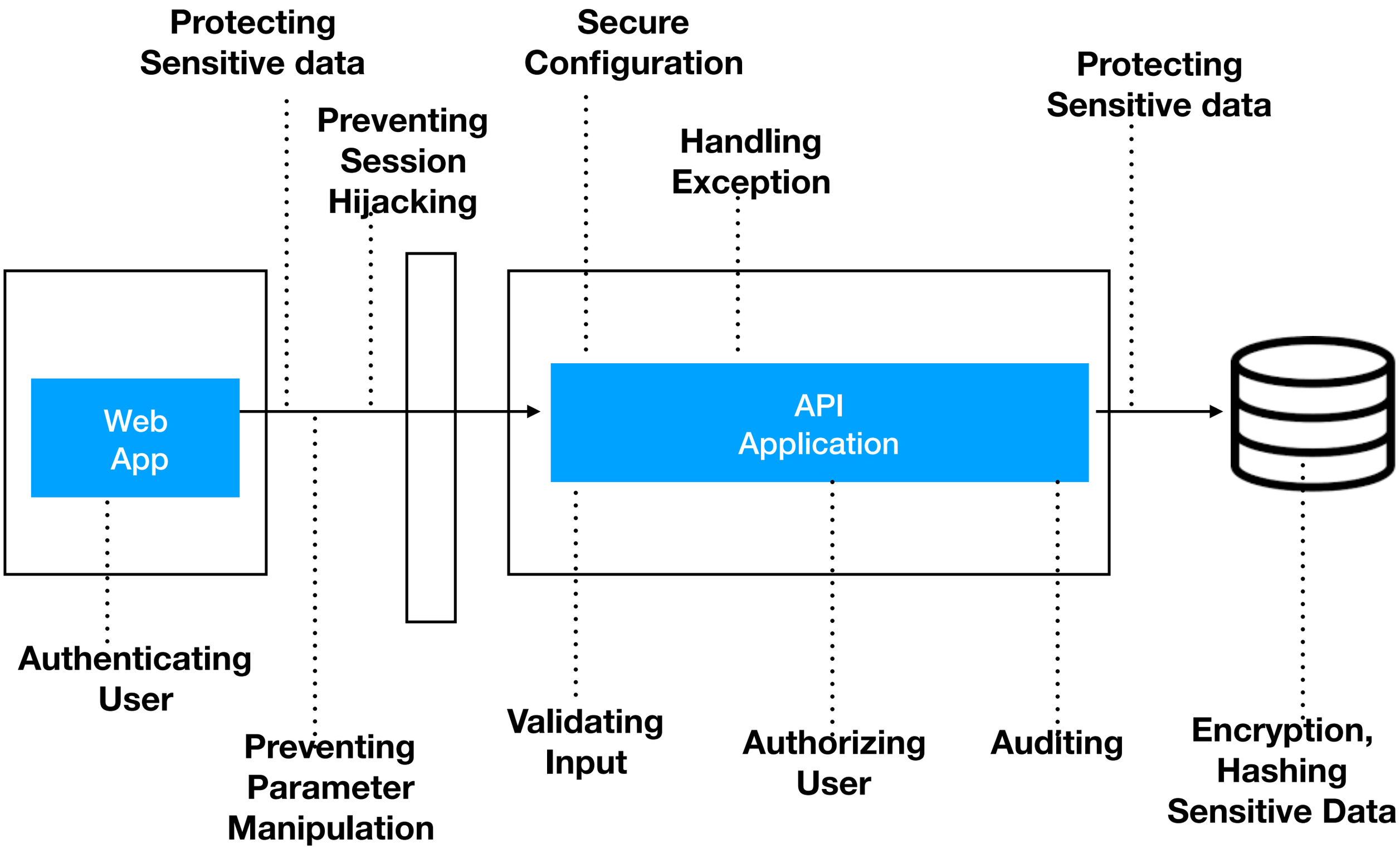
mongoDB
Atlas

Message queue
Azure Queue

Azure Data Lake


Public IP address
APi Gateway
<—-
Network Load Balancer
30267
Istio
Reverse proxy/
Ingress
<—-
kubernetes
Side car
Side car
<—-
Microservice 1
Microservice 2

14. Application Security View



Concerns

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

AAA

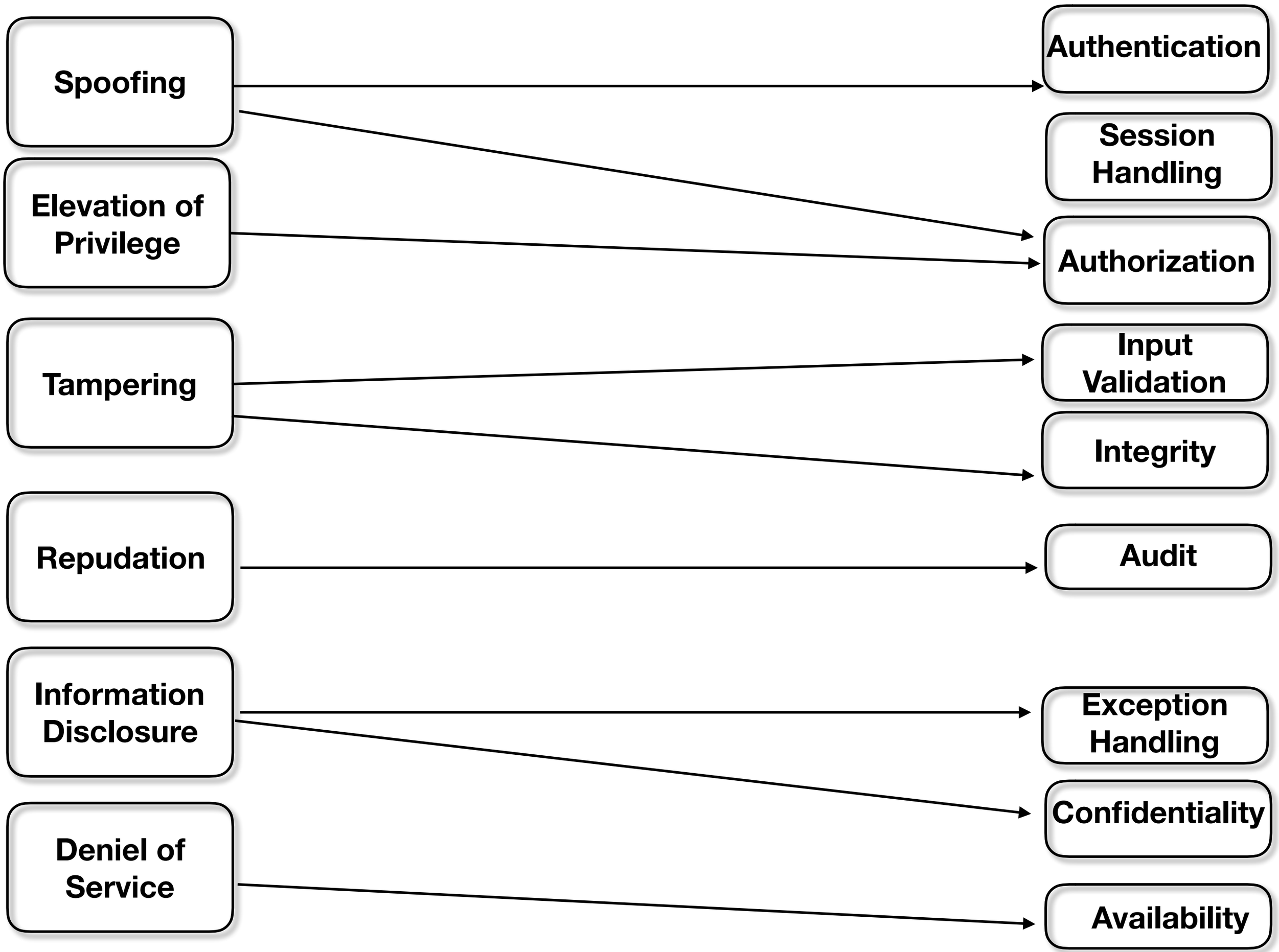
Approach

- SAML
- SSO
- Federated
- OAuth2
- Kerberos
-

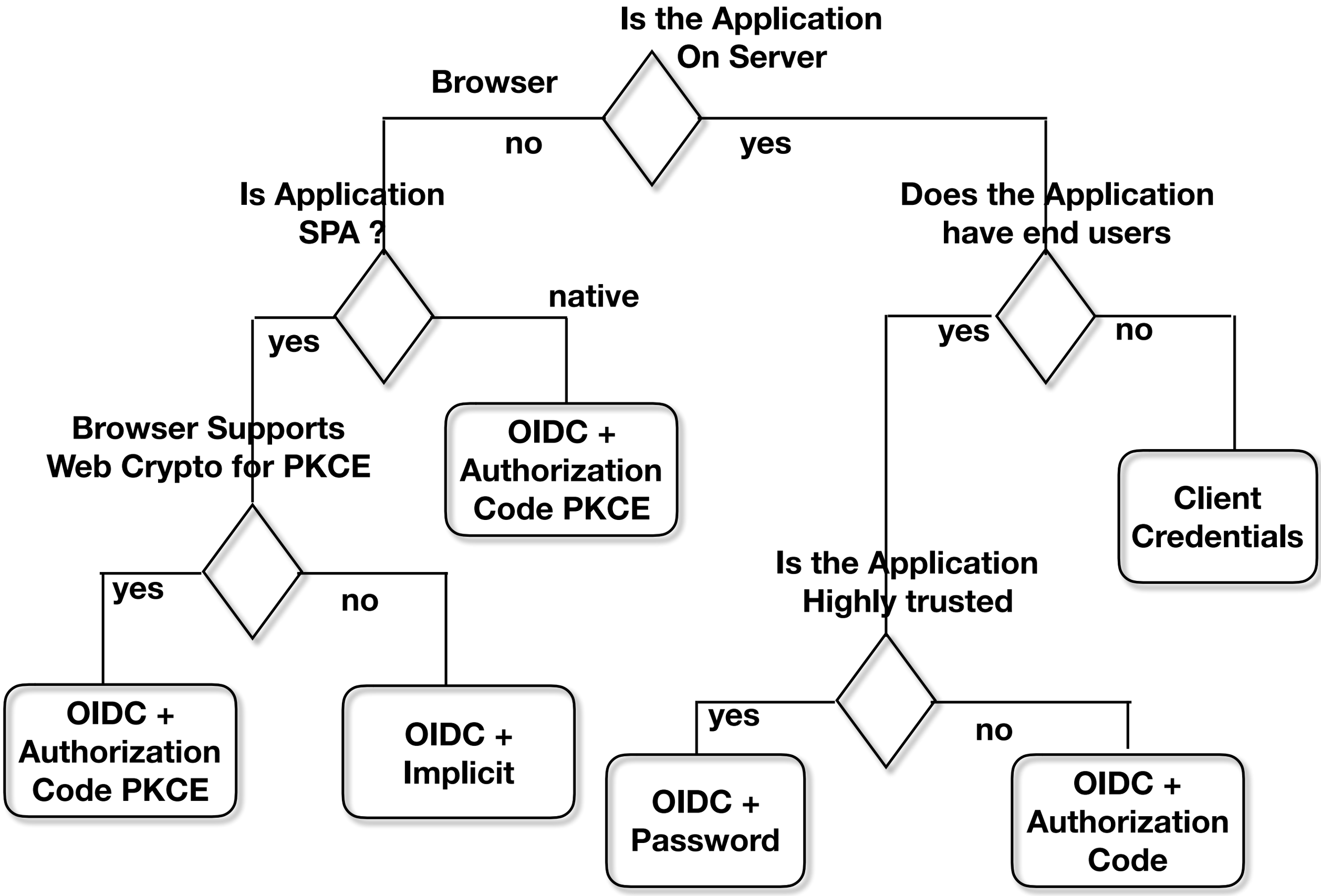
Threat Modeling Method	Features
STRIDE	• Helps identify relevant mitigating techniques • Is the most mature • Is easy to use but is time consuming
PASTA	• Helps identify relevant mitigating techniques • Directly contributes to risk management • Encourages collaboration among stakeholders • Contains built-in 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

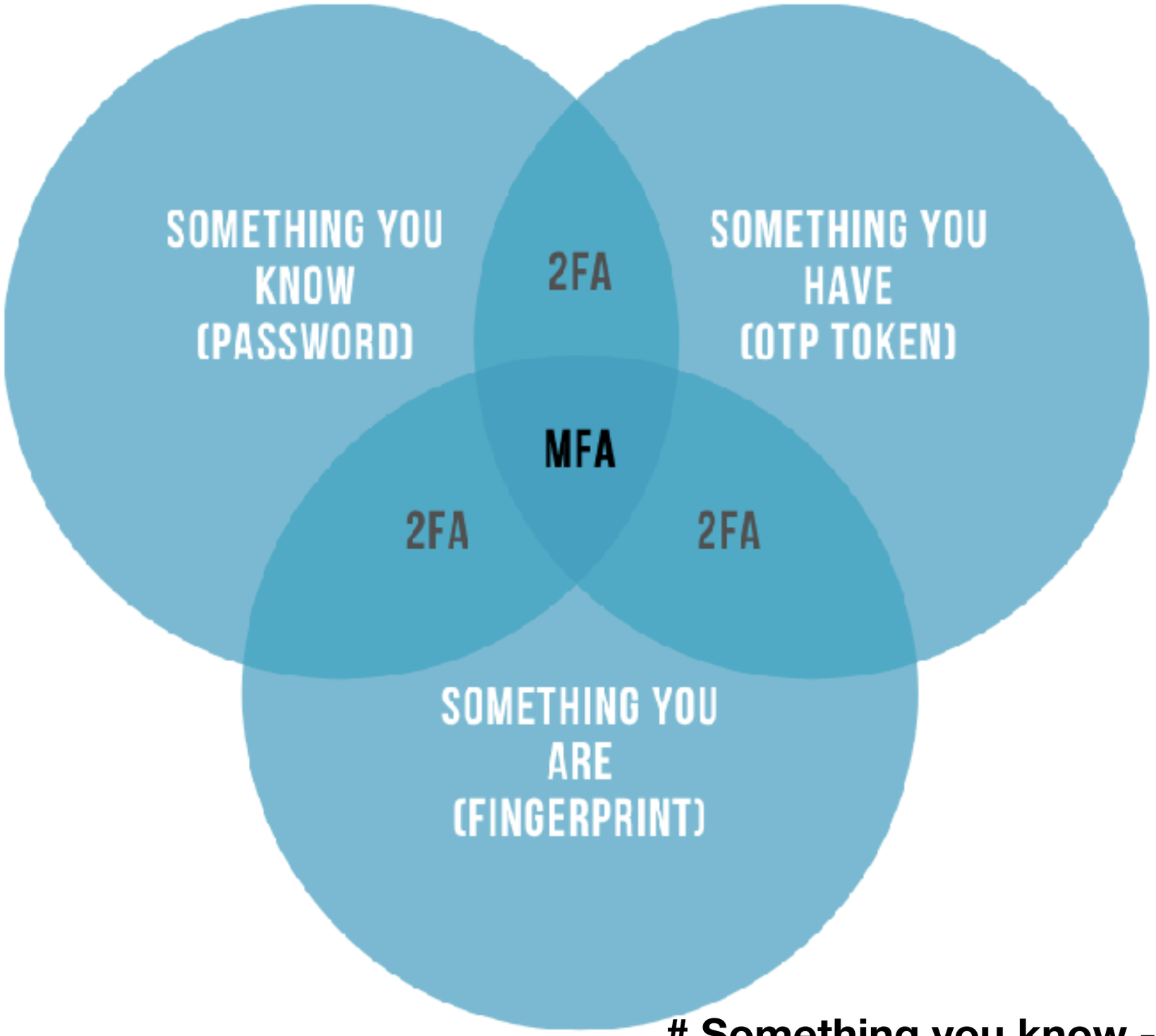
PASTA

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


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

Step 1. Authentication

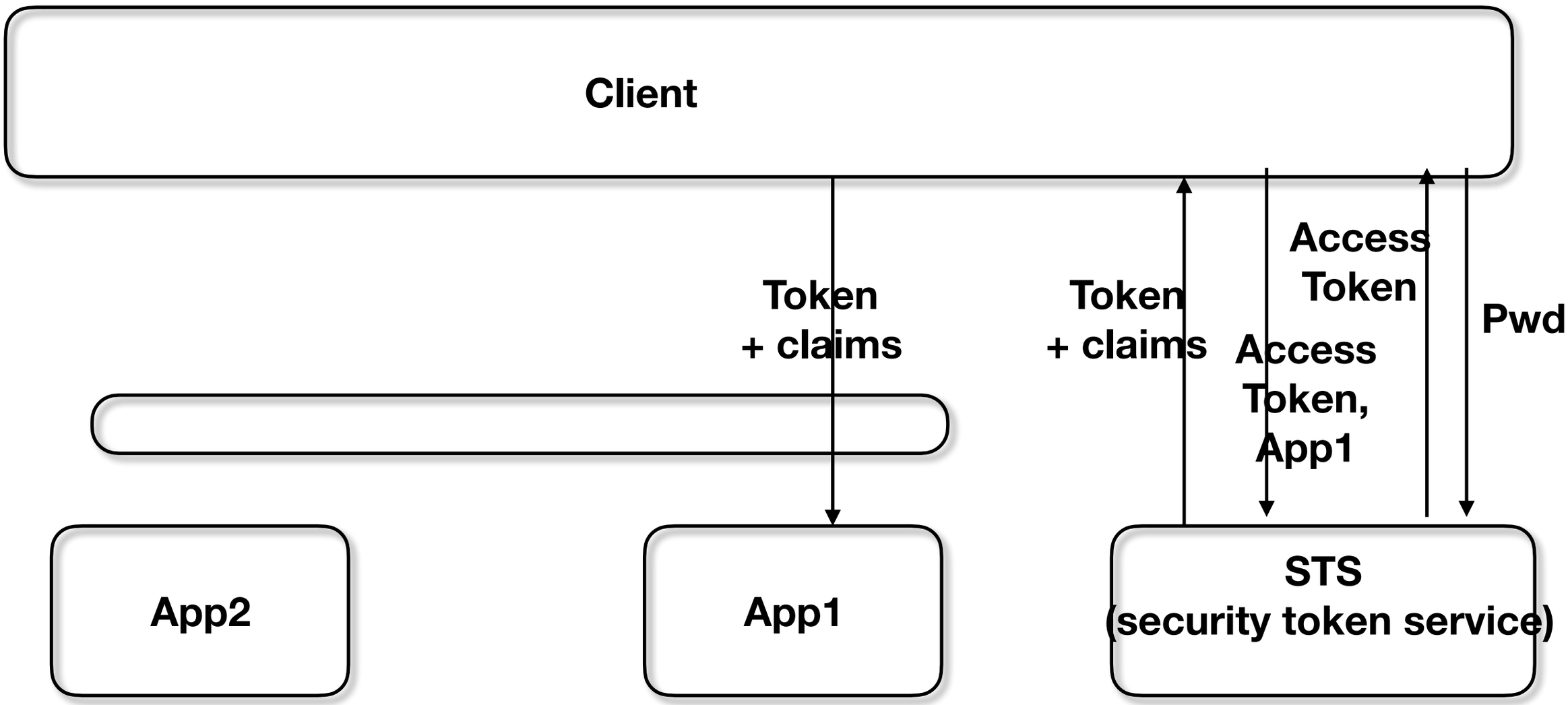




 # Something you know - pwd, security question
 # Something you Have - otp, cert, rsa
 # Something you are - voice, face, dna, ..
 # Where you are - country, office, …
 # Something you do - behaviour

- OAuth2 + Openid connect
- SAML <— legacy (only if constraint)
- Kerberos / LDAP <— intranet
- Basic <— vulnerable
- Digest <— vulnerable
- Custom <— vulnerable
- …

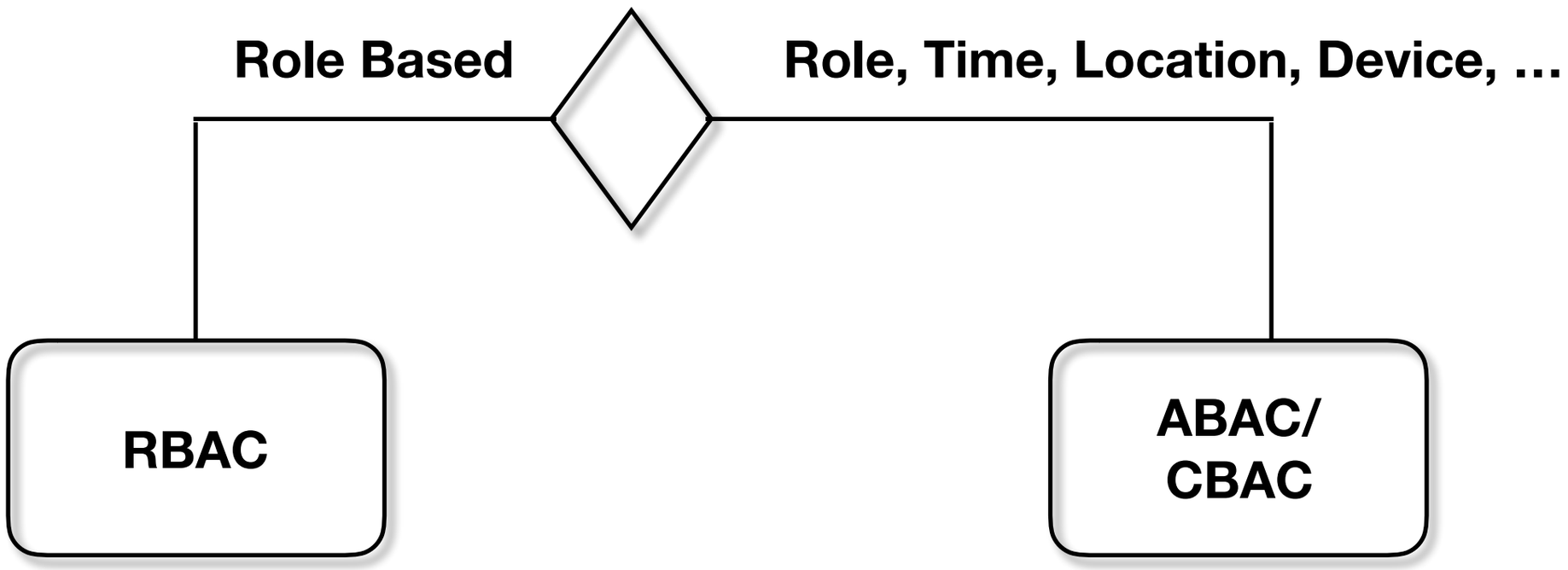
Centralize



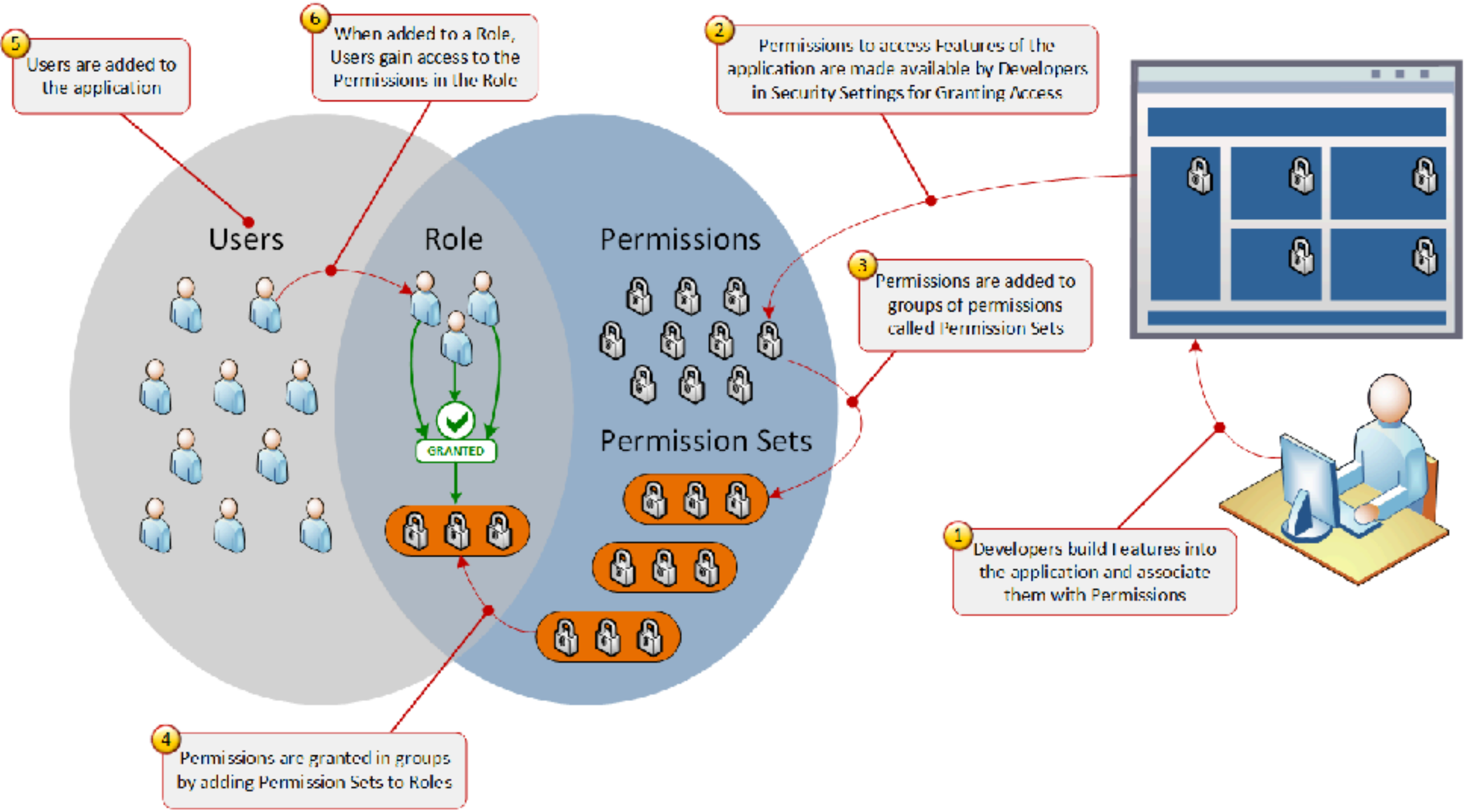
User profile
Dept -> claim1 ,claim2
Country -> claim3
FTE-> claim4
time->claim5
…->claim6

Oauth2 + openidconnect <— internet
Kerberos and LDAP <—intranet

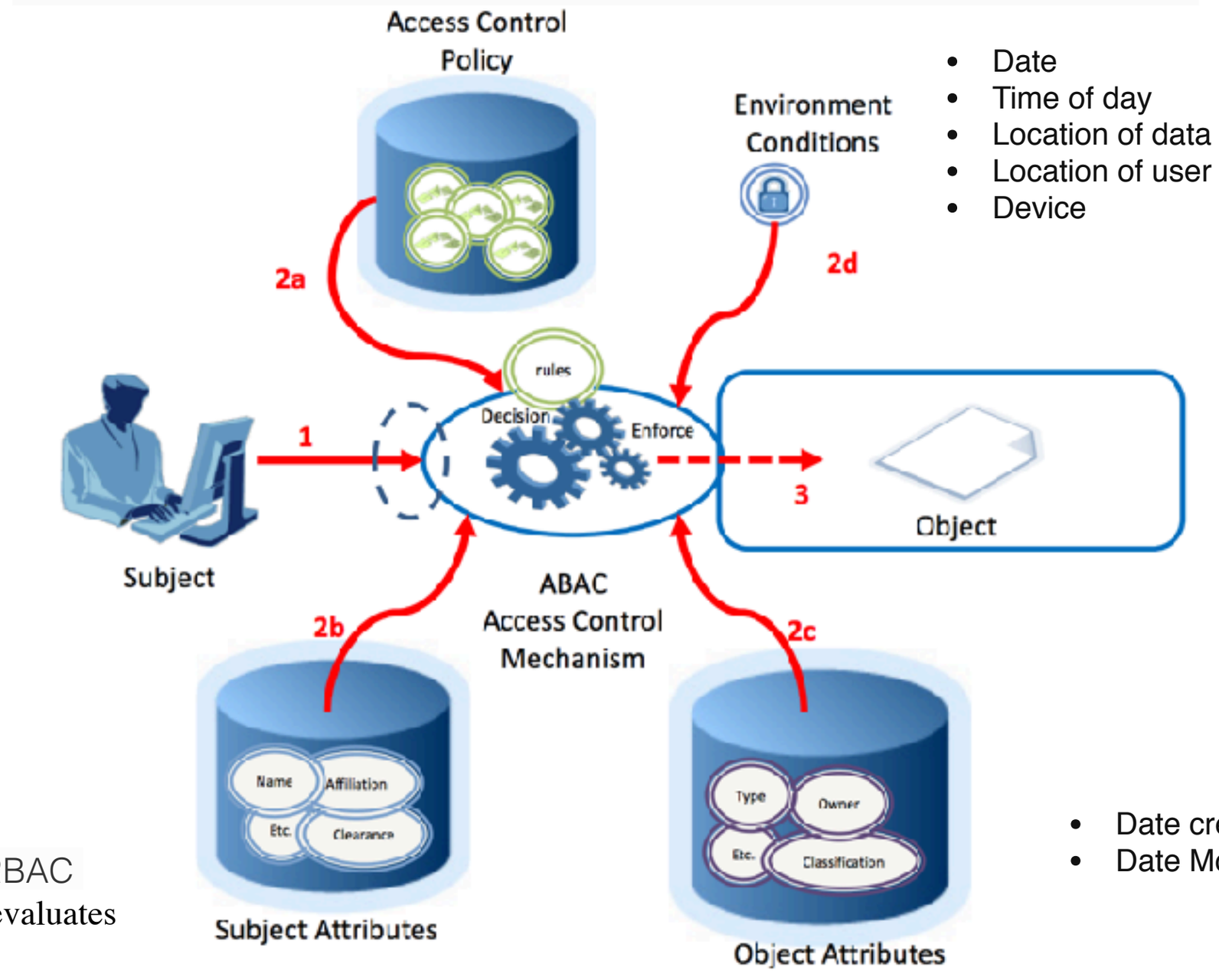
Step 2. Authorization



Role Based Access Control (RBAC)



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



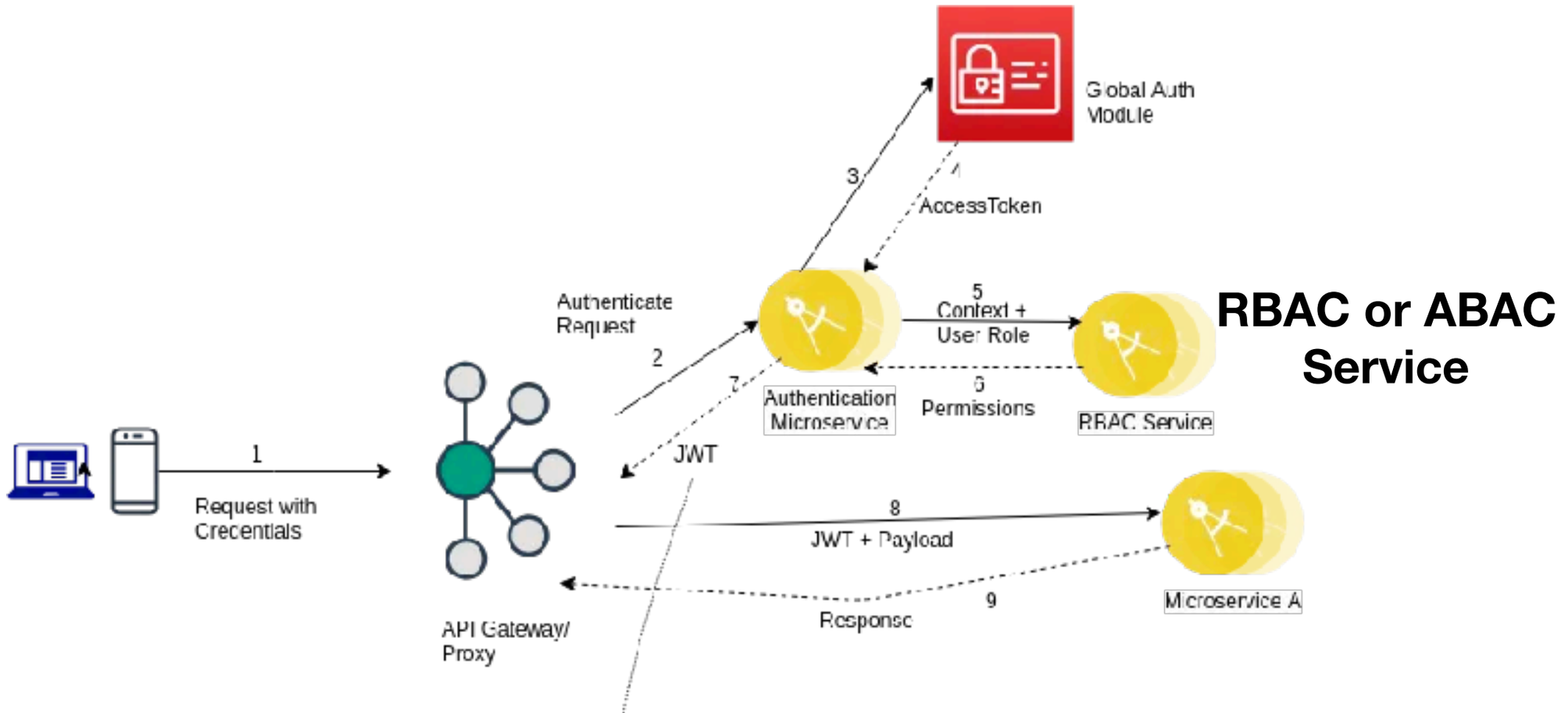
- Date
- Time of day
- Location of data
- Location of user
- Device

- 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



RBAC or ABAC Service


```
{
"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

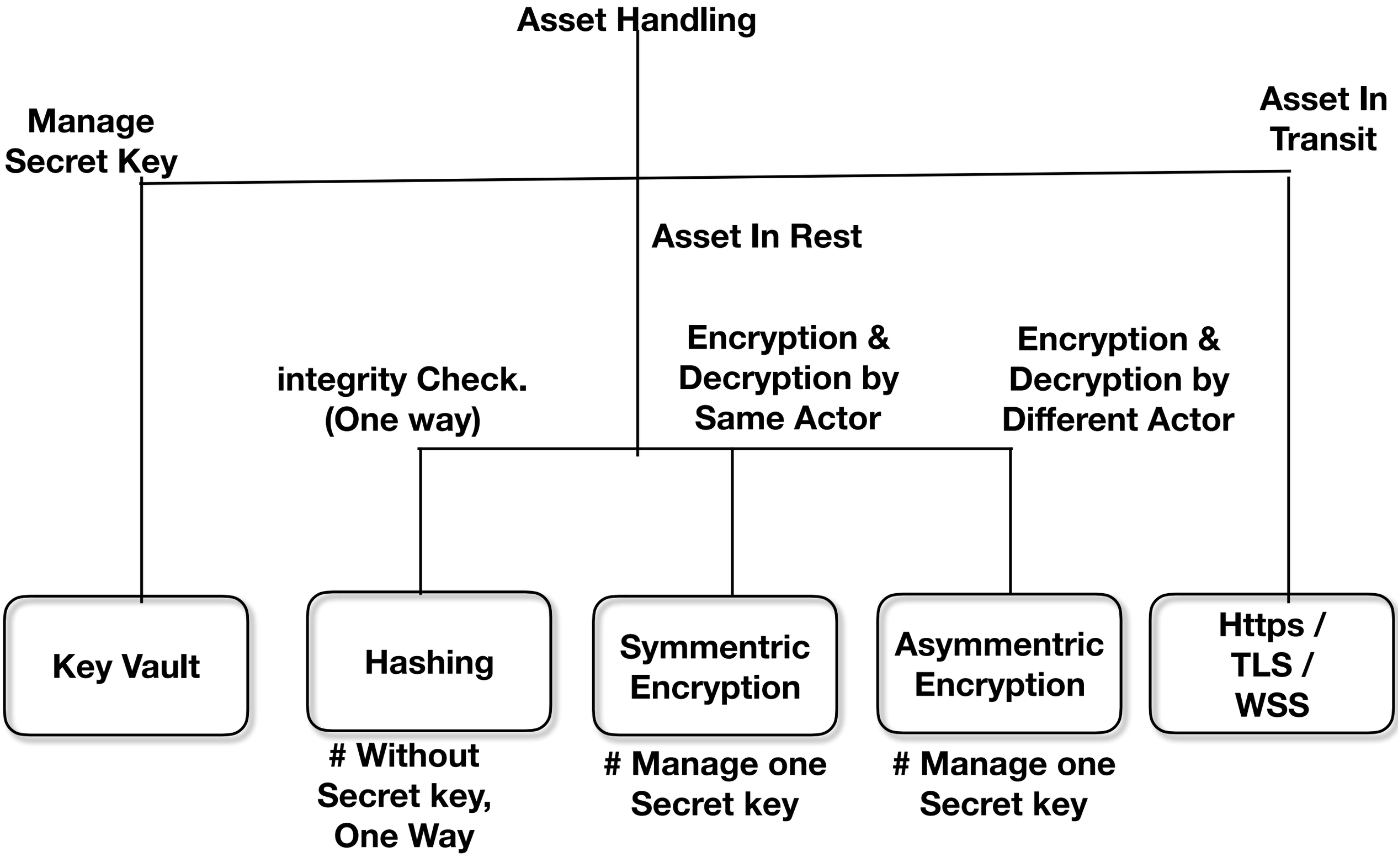
who - Identity

what - Action

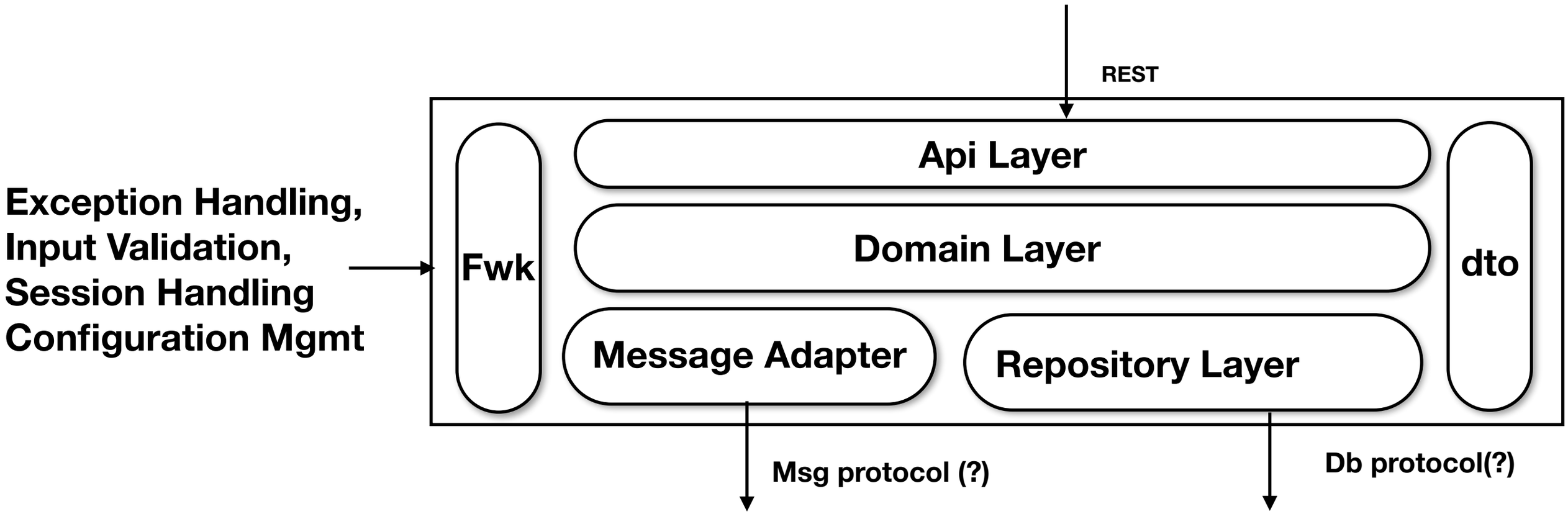
where - Component/Service/Object/Method

when - Timestamp:

Step 4. Privacy and Confidentiality



Step 5. Cross cutting concerns



Public IP address

vnet

asg subnet

WAF

Network Load Balancer

Application Gateway

asg subnet

NGINX 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

mongoDB Atlas

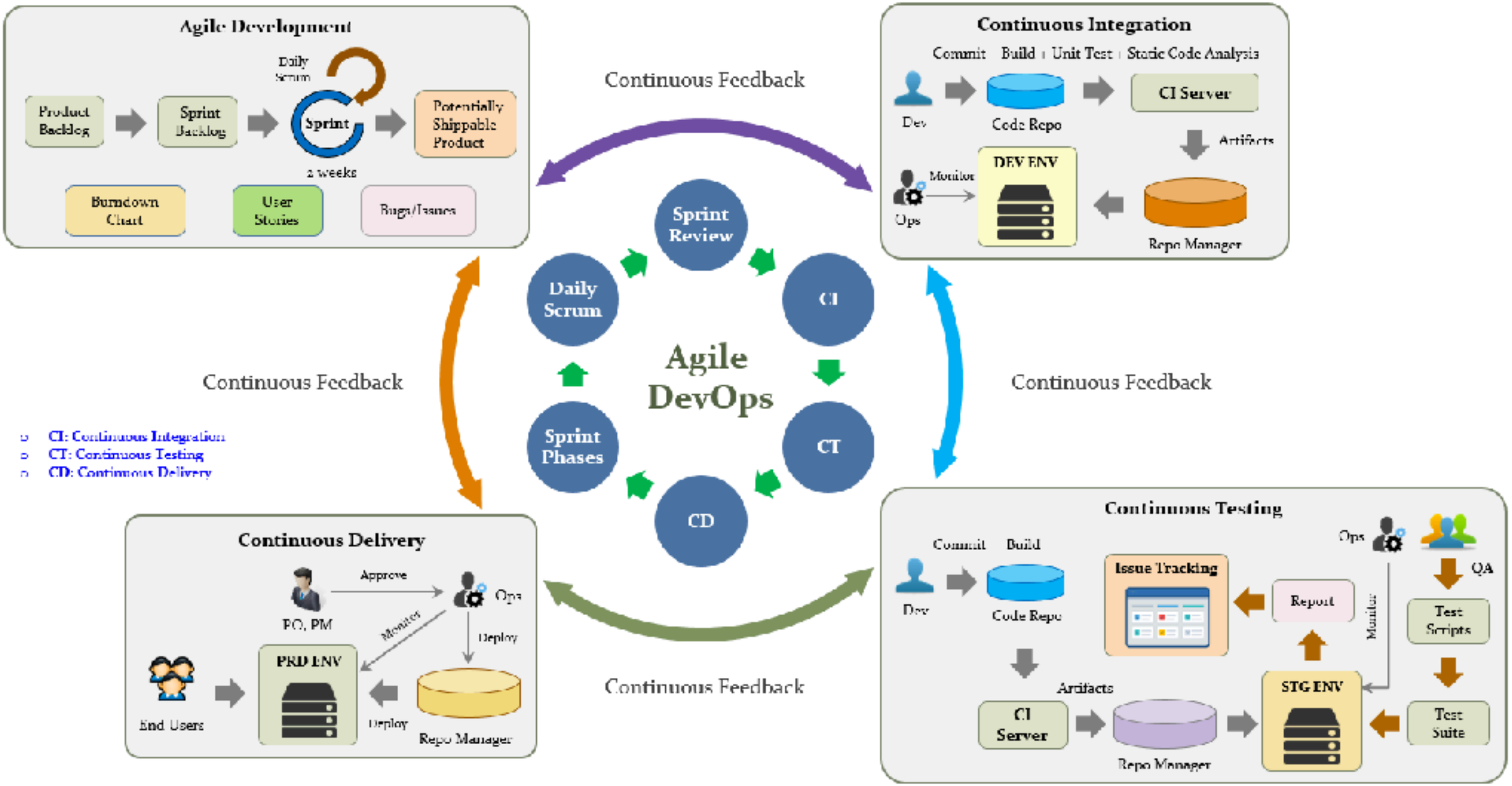
mongoDB

Message queue

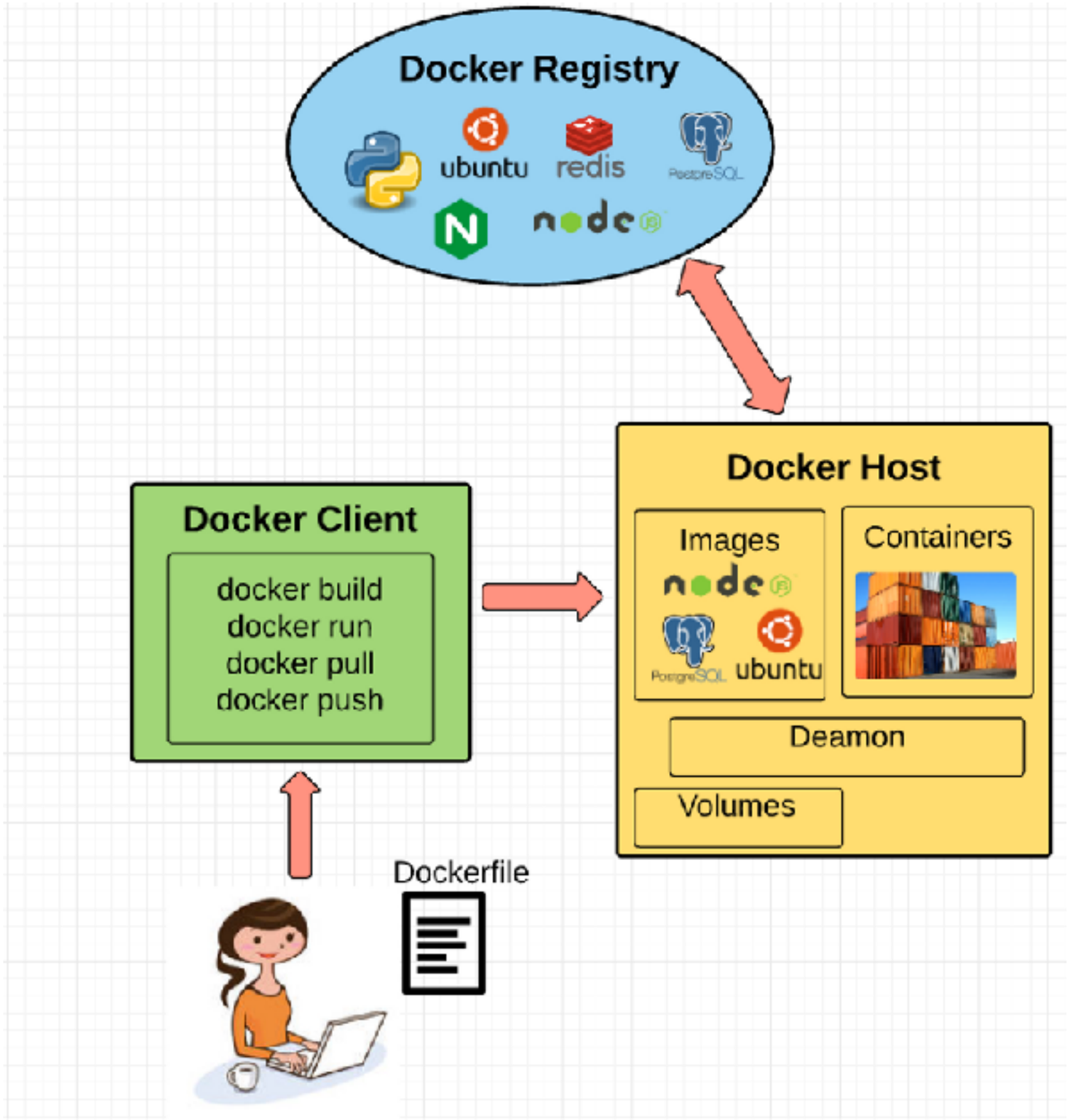
Azure Queue

Azure Data Lake

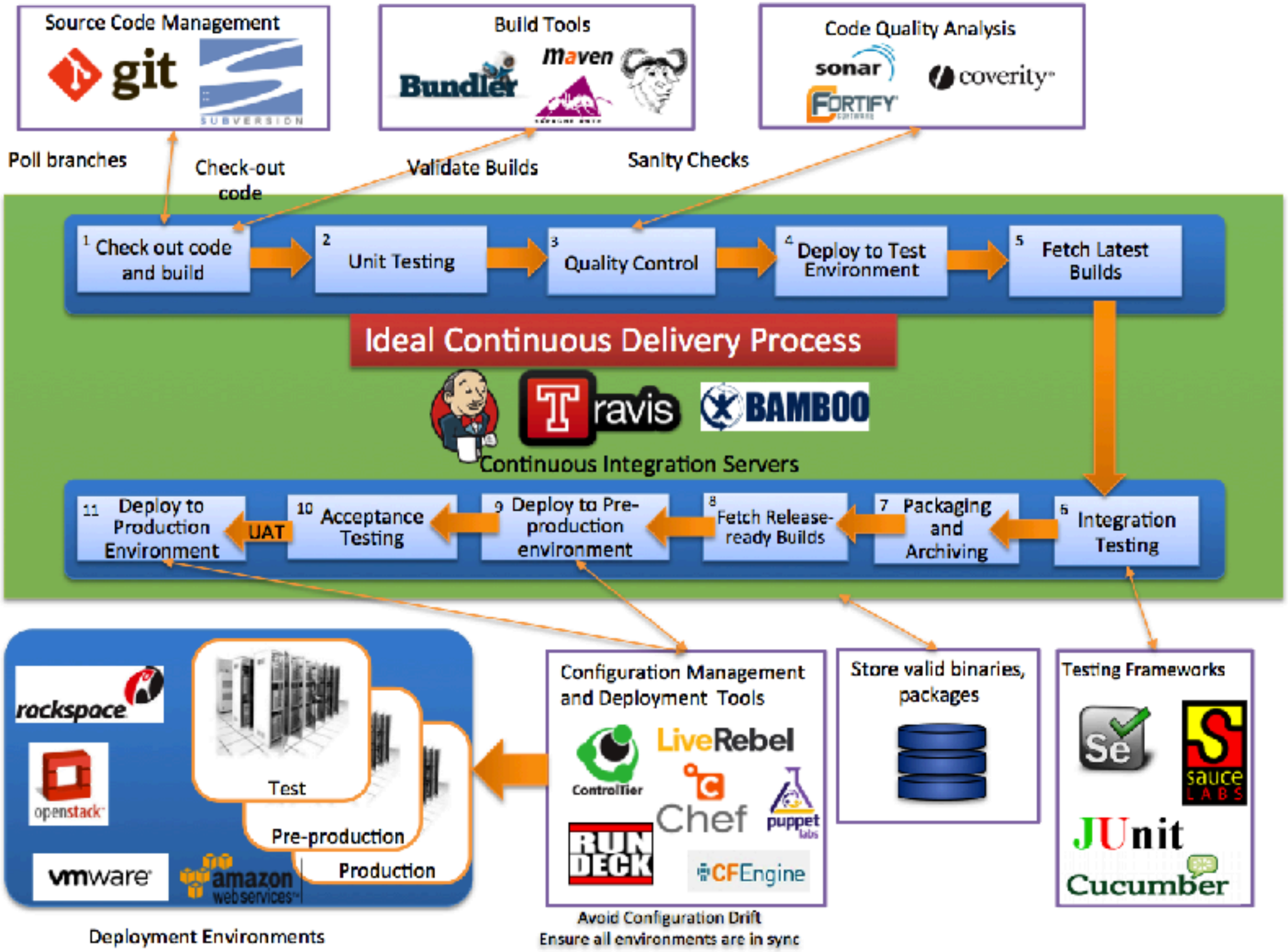
15. DevOps View



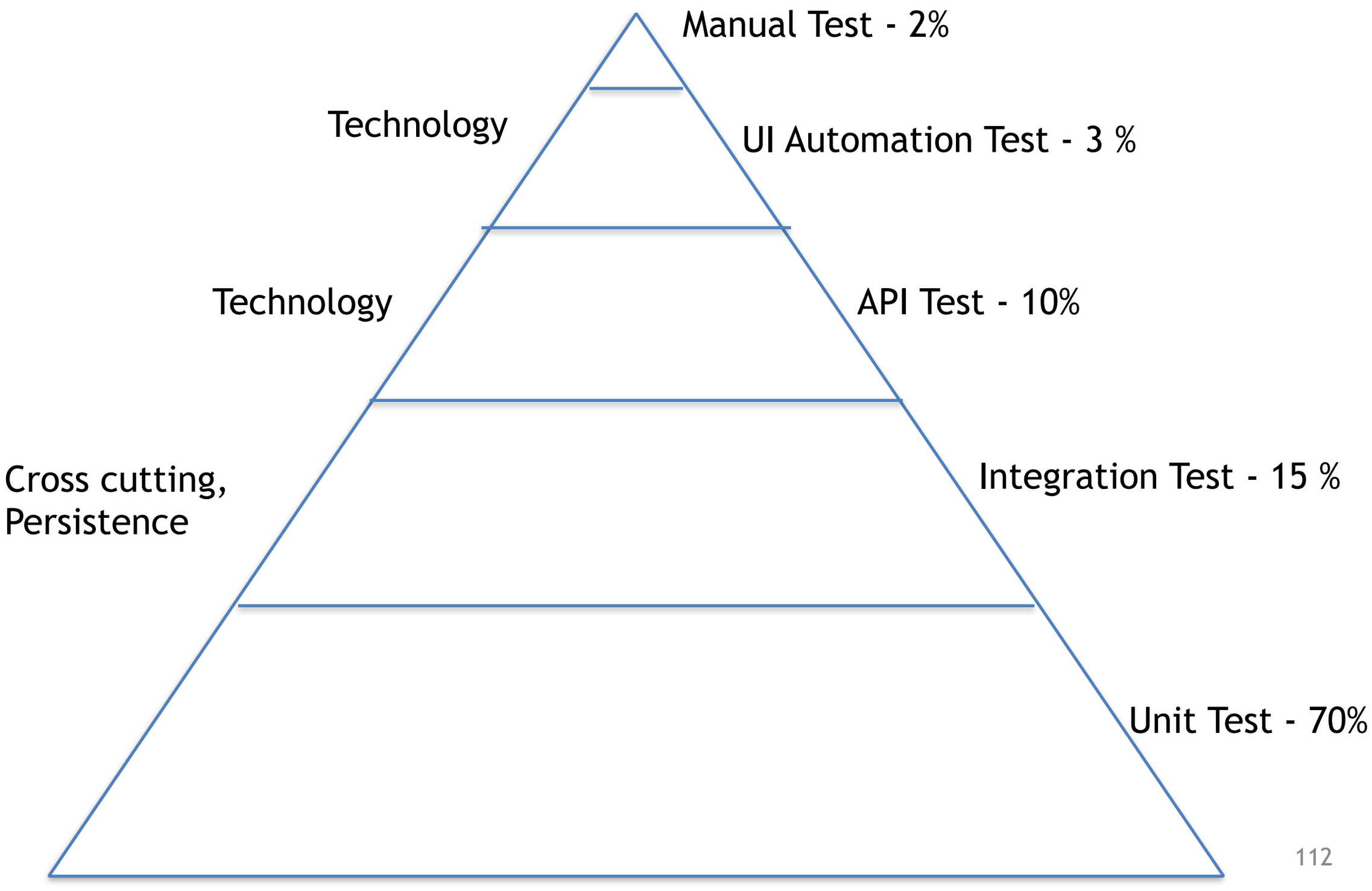
Step 1. Containerizing



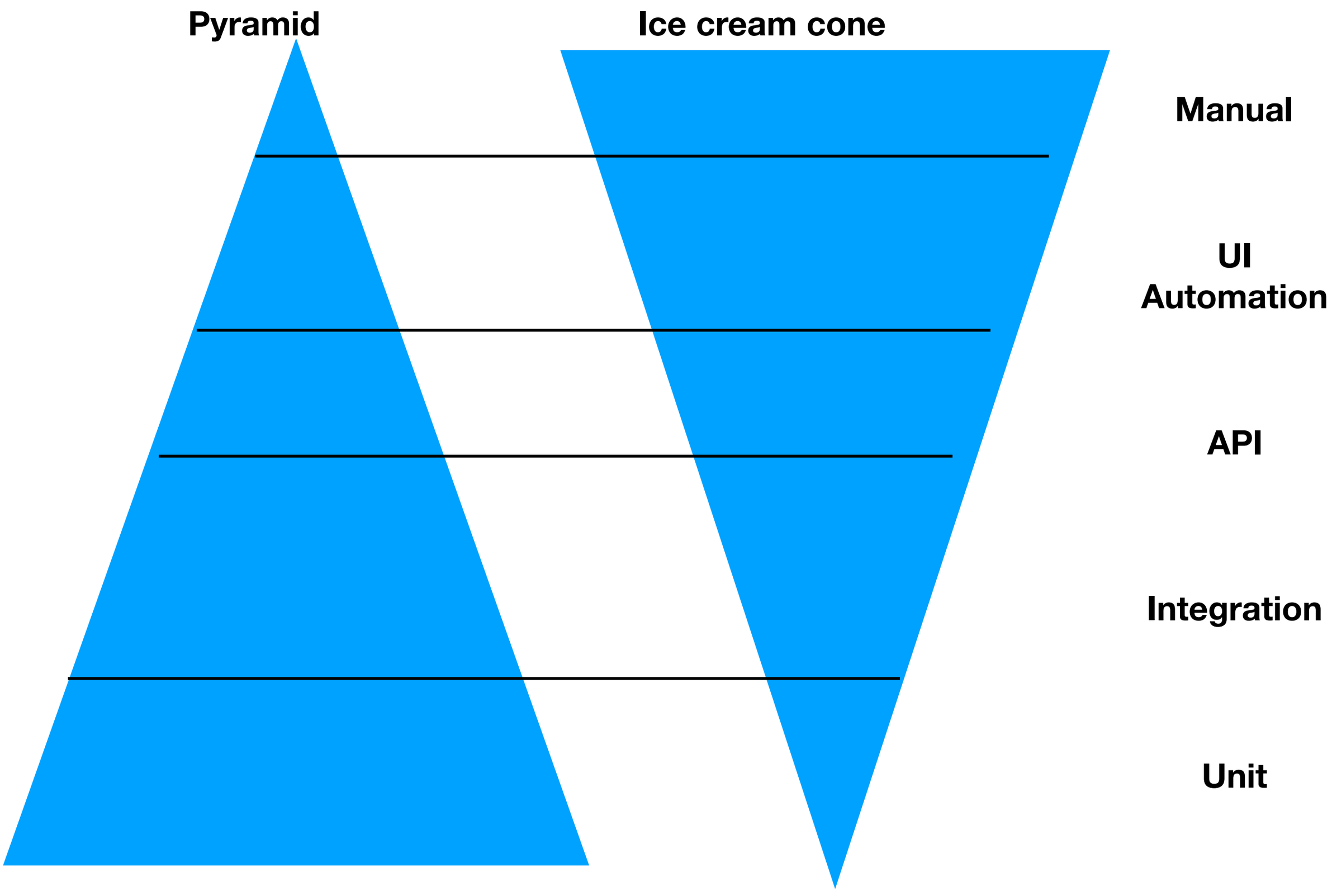
Step 2. Integrating infrastructure automation with CI/CD

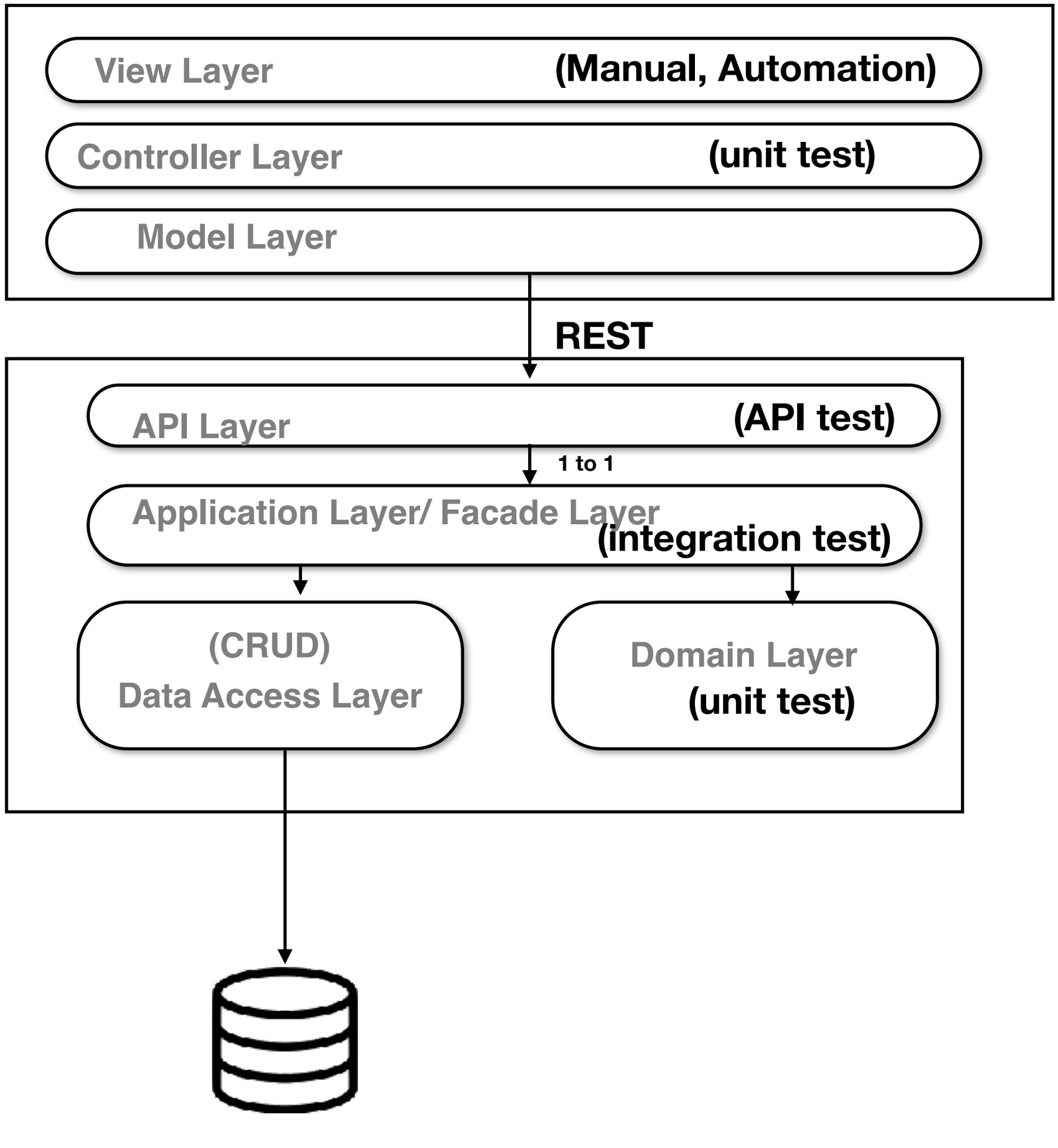


Step 3. test automation and aligning QA with development

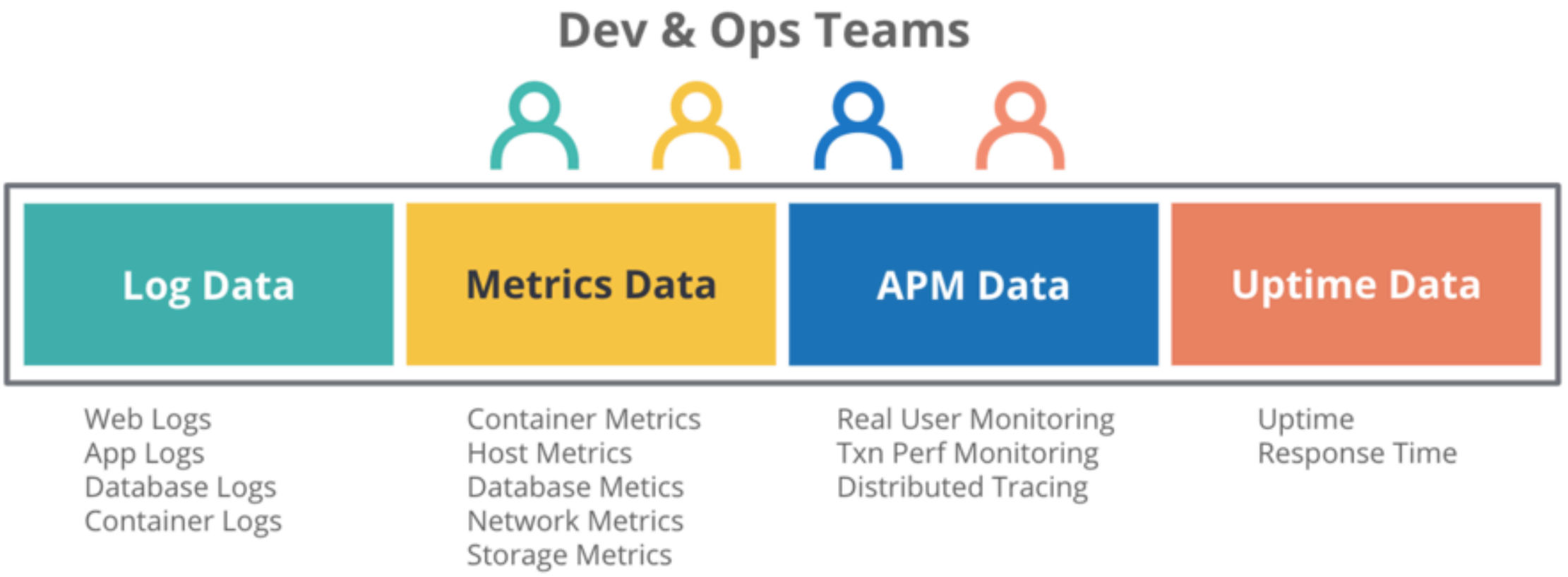


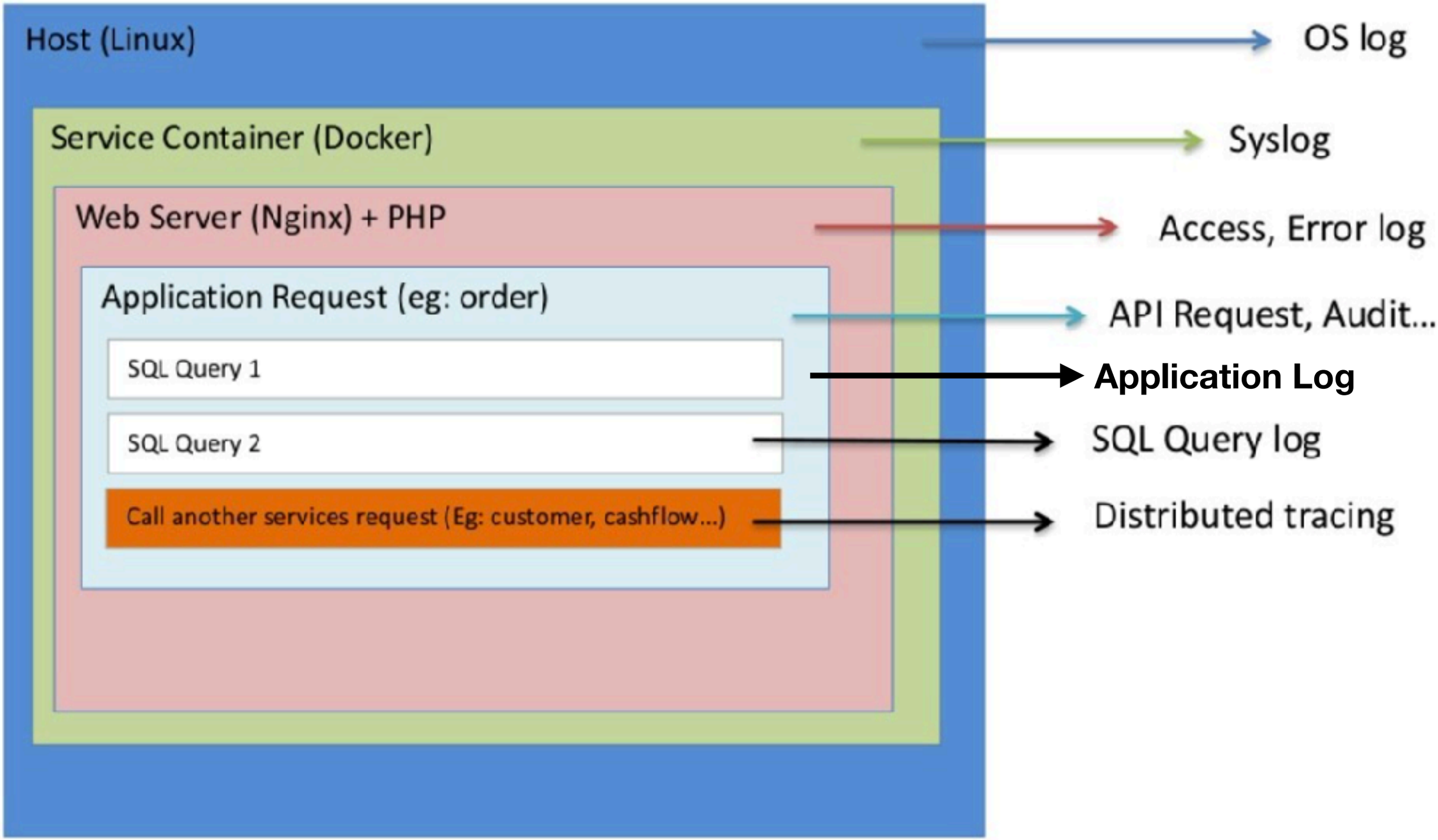
Pyramid Test

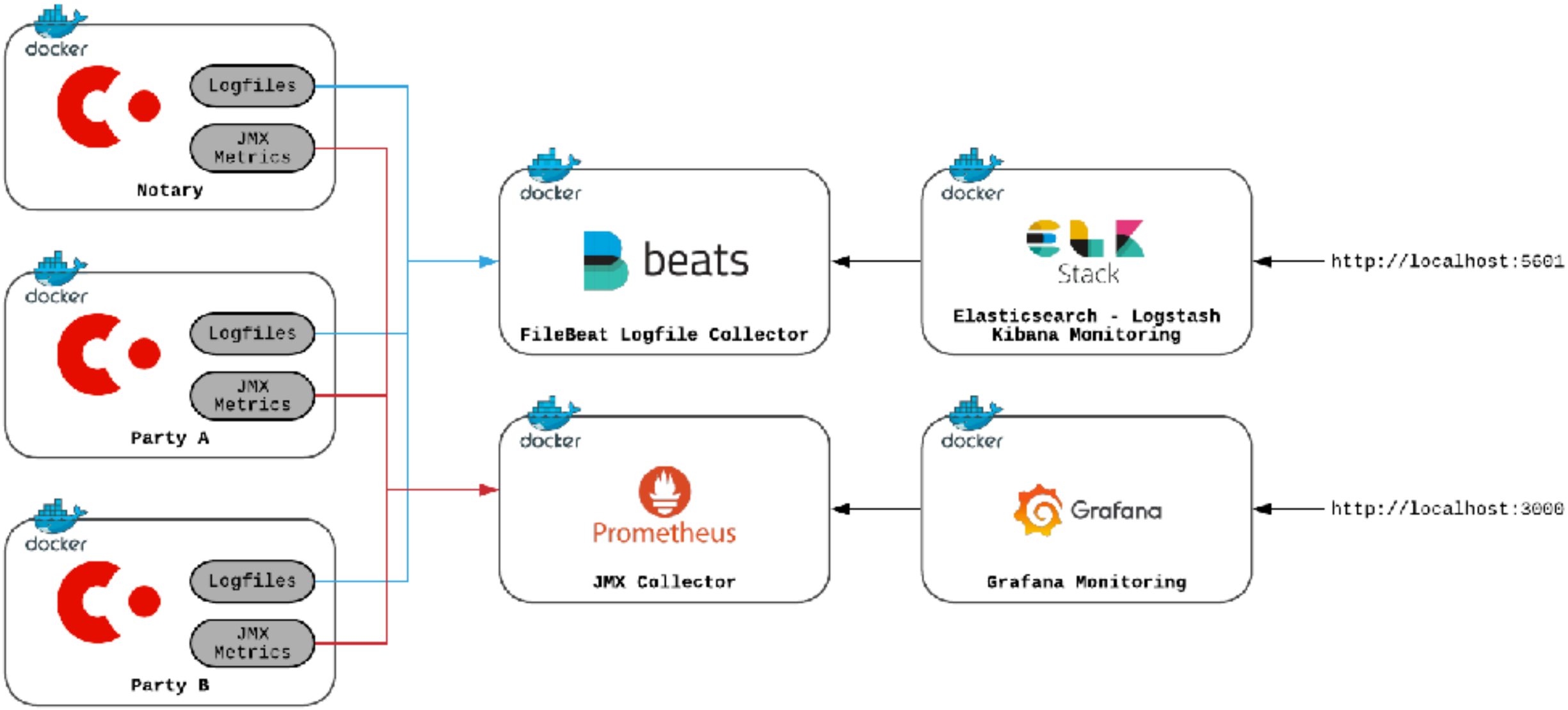



View Layer
(Manual, Automation)
Controller Layer
(unit test)
Model Layer
REST
API Layer
(API test)
1 to 1
Application Layer/ Facade Layer
(integration test)
(CRUD)
Data Access Layer
Domain Layer
(unit test)

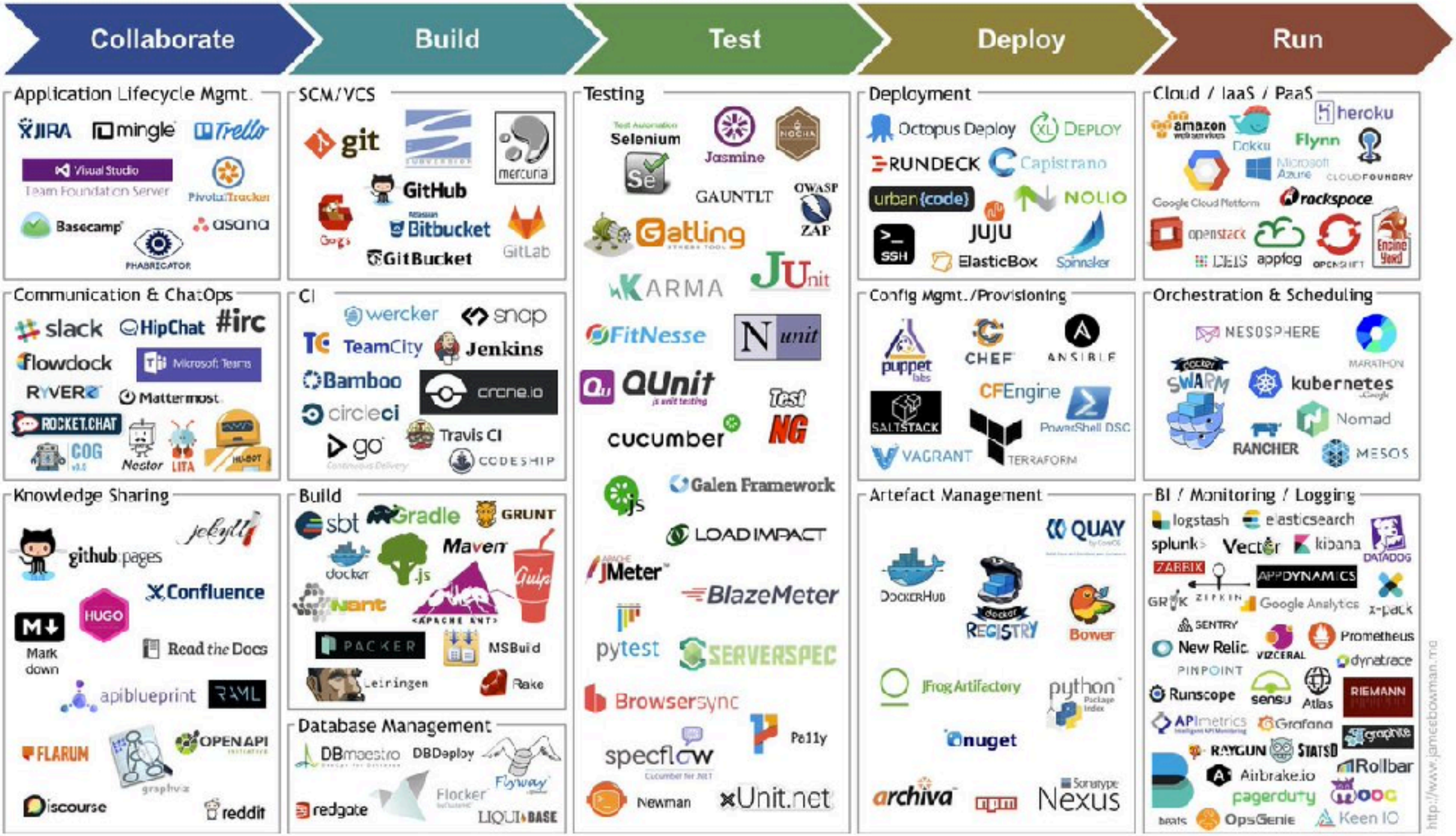
Step 4. Setup Monitoring



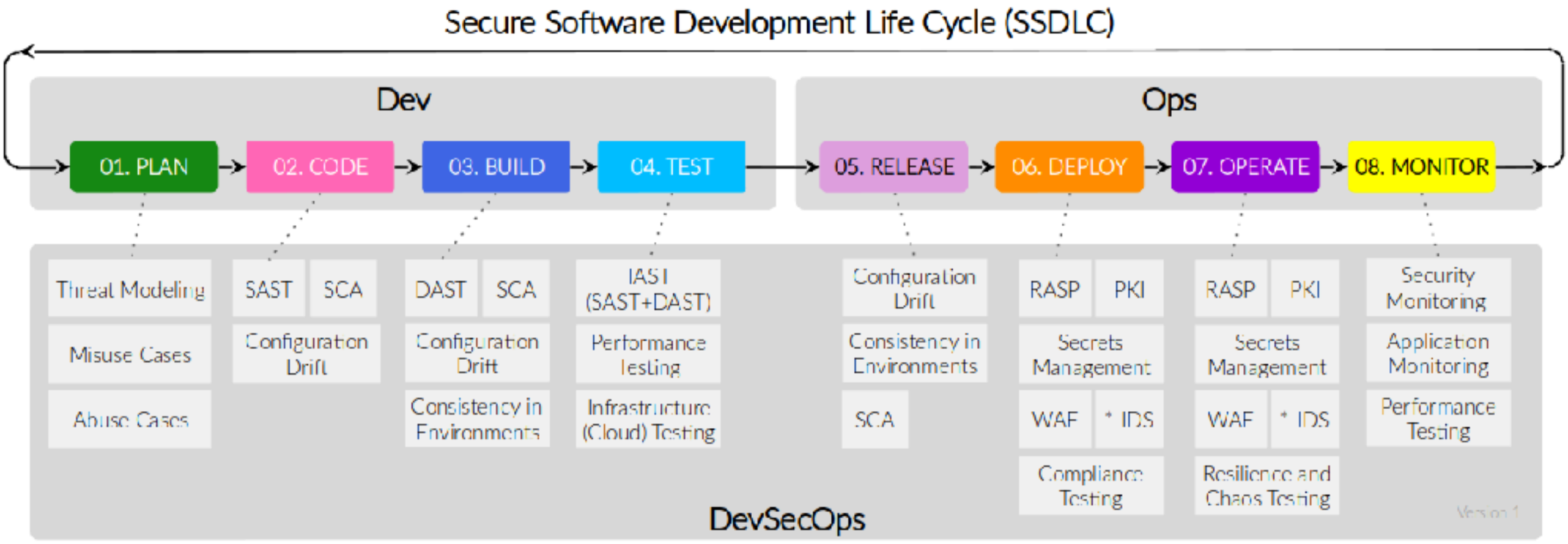

Host (Linux)
Service Container (Docker)
Web Server (Nginx) + PHP
Application Request (eg: order)
SQL Query 1
SQL Query 2
Call another services request (Eg: customer, cashflow...)
OS log
Syslog
Access, Error log
API Request, Audit...
Application Log
SQL Query log
Distributed tracing


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
Bitbucket
GitBucket
GitLab
Testing
Selenium
Jasmine
NOCHA
GAUNTLT
OWASP ZAP
Gatling
JUnit
KARMA
FitNesse
N unit
QUnit
cucumber
Test NG
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
Flynn
Docker
Microsoft Azure
CLOUDFOUNDRY
Google Cloud Platform
rackspace
openstack
DEIS
appfog
OPENSHIFT
Engine Yard
Communication & ChatOps
slack
HipChat
#irc
flowdock
Microsoft Teams
RYVER
Mattermost
ROCKET.CHAT
COG
Nectar
LITA
HU-BOT
CI
wercker
snap
TeamCity
Jenkins
Bamboo
circleci
Travis CI
go
CODESHIP
Config Mgmt./Provisioning
puppet labs
CHEF
ANSIBLE
CFEngine
SALTSTACK
PowerShell DSC
VAGRANT
TERRAFORM
Orchestration & Scheduling
MESOSPHERE
MARATHON
SWARM
kubernetes
Nomad
RANCHER
MESOS
Knowledge Sharing
github pages
jekyll
Confluence
HUGO
Mark down
Read the Docs
apiblueprint
RAML
FLARUM
OPEN API
graphviz
Discourse
reddit
Build
sbt
Gradle
GRUNT
docker
Maven
Gulp
Nant
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
dynatrace
PINPOINT
Runscope
sensu
Atlas
RIEMANN
APImetrics
Grafana
graphite
RAYGUN
STATSD
Rollbar
Airbrake.io
pagerduty
OOOC
beats
OpsGenie
Keen IO
http://www.jamescbowman.net

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

Pre-Commit Hooks

Talisman, git-secrets, HOUND

Secrets Management

HashiCorp Vault, Secret Manager, AWS Secrets Manager

SAST

Checkmarx, DerScanner, ECG

SCA

GitHub, BLACKDUCK, snyk

DAST

RAPID7, acunetix, CONTRAST SECURITY

IAST

CONTRAST SECURITY, SYNOPSYS, VERACODE

Secure Infrastructure as code

clair, anchore, DOCKSCAN

Compliance as code

KitchenCI, SERVERSPEC, CHEF INSPEC

RASP

Signal Sciences, TREND MICRO, imperva

Web Application Firewall

imperva, CLOUDFLARE, wallarm

Monitoring Tools

DATADOG, Nagios, PerfOps

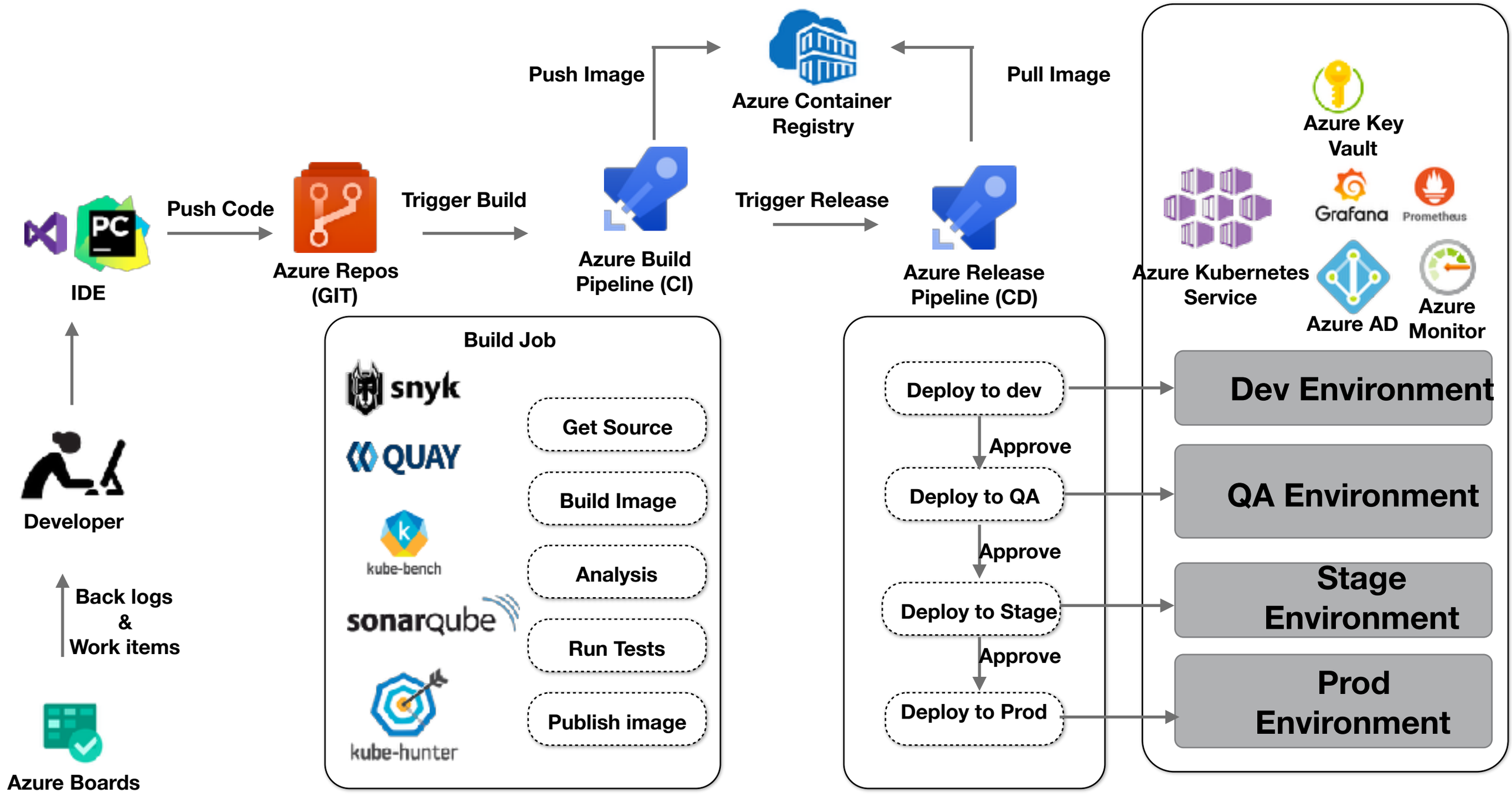
Chaos Engineering

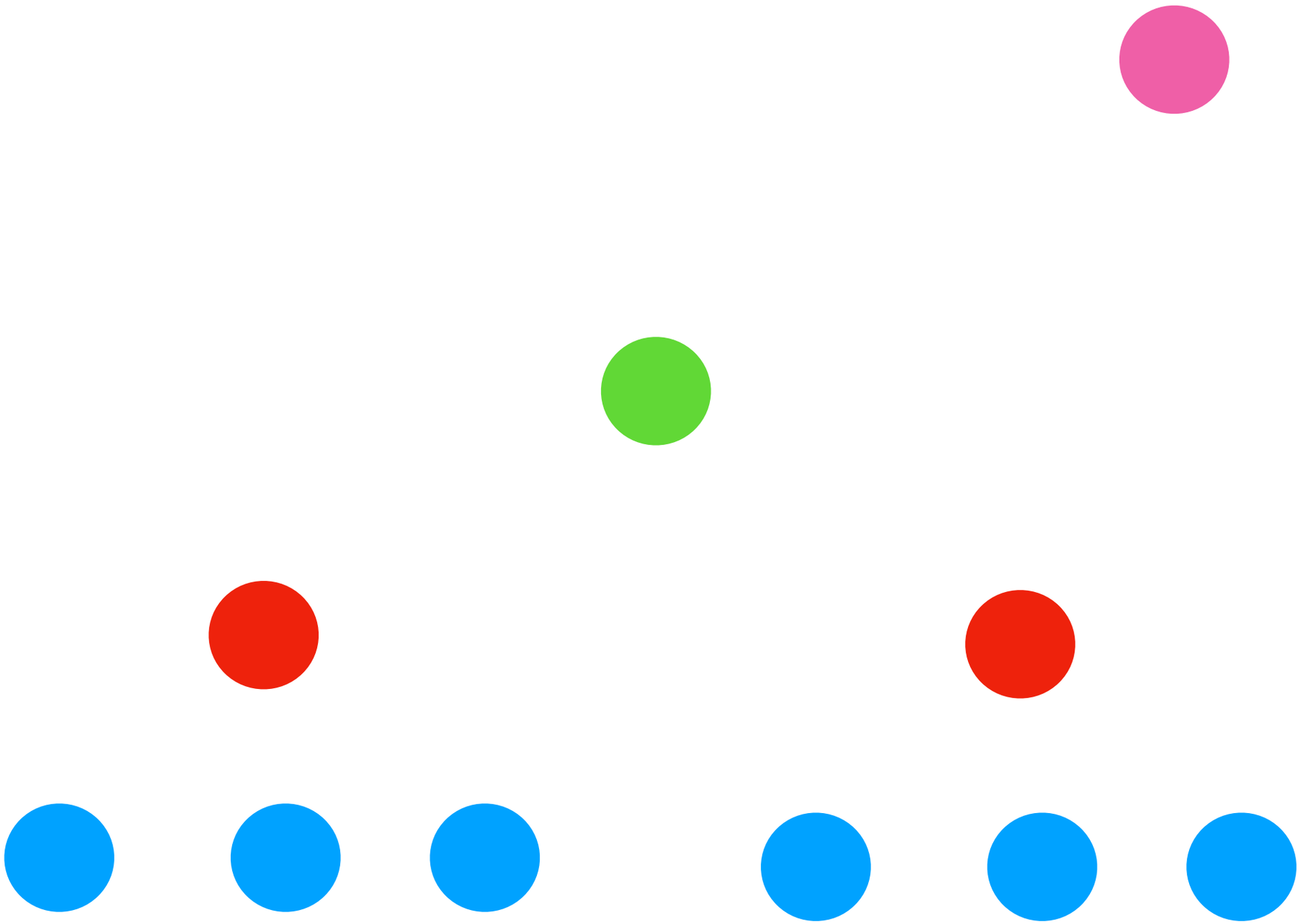
Gremlin, ChaosToolkit

Vulnerability management

Hacker, ARCHERY

Reference model





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

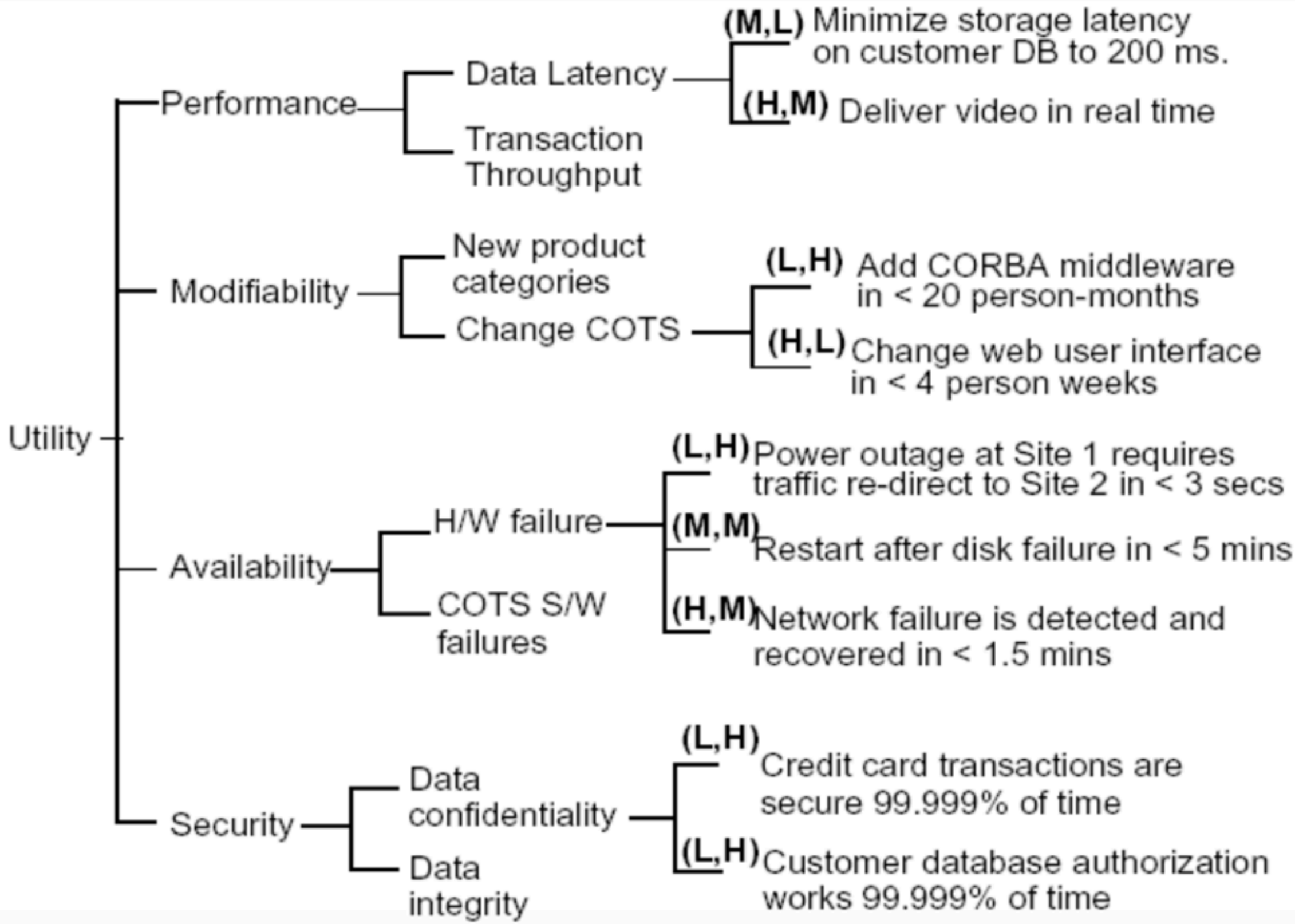
 # brainstorm for scenarios

 # 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.
Transaction Throughput
(H,M) Deliver video in real time
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
COTS S/W failures
(H,M) Network failure is detected and recovered in < 1.5 mins
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

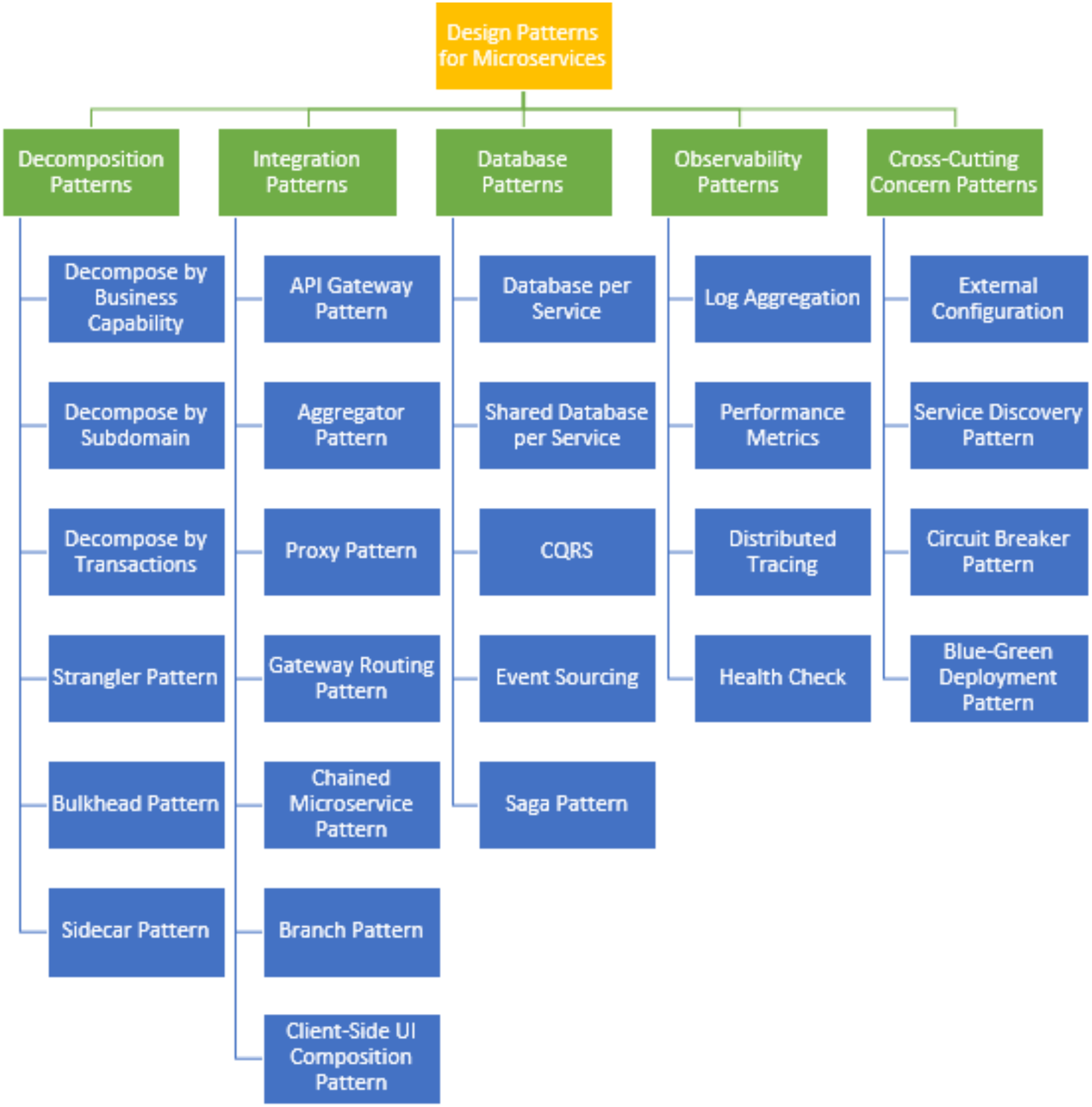
	Requirements Gathering	Design	Development	Testing	Deployment	Maintenance
Performance Modeling						
Prototyping						
Arch/Design Review						
Code Review						
Measuring						
Load Testing						
Stress Testing						
Capacity Testing						
Tuning						

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

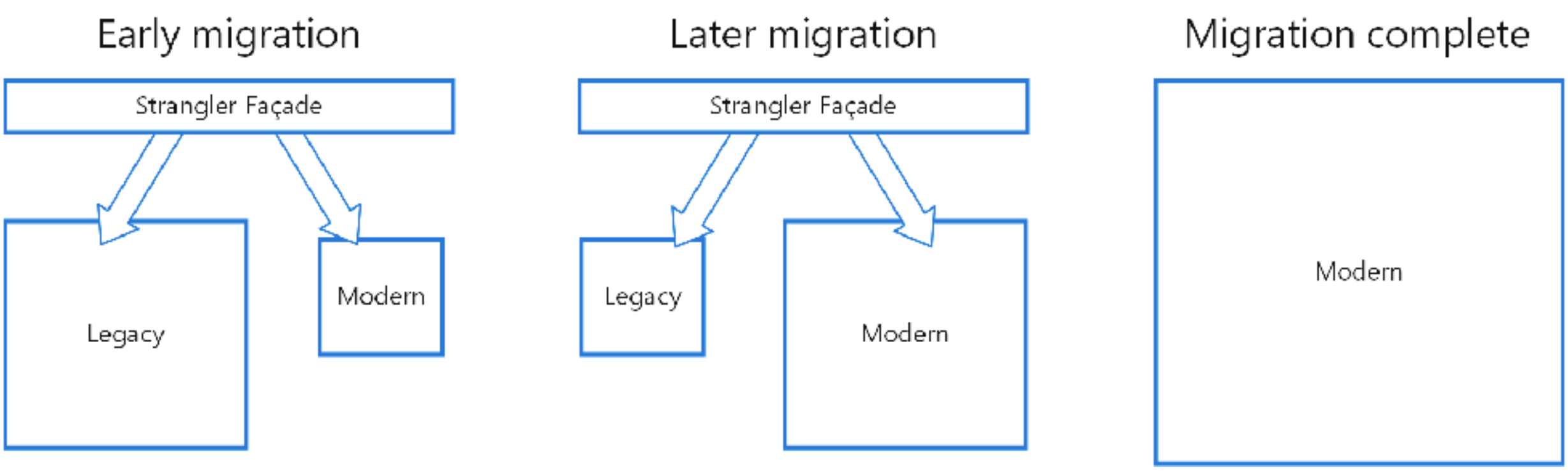
Security

- 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

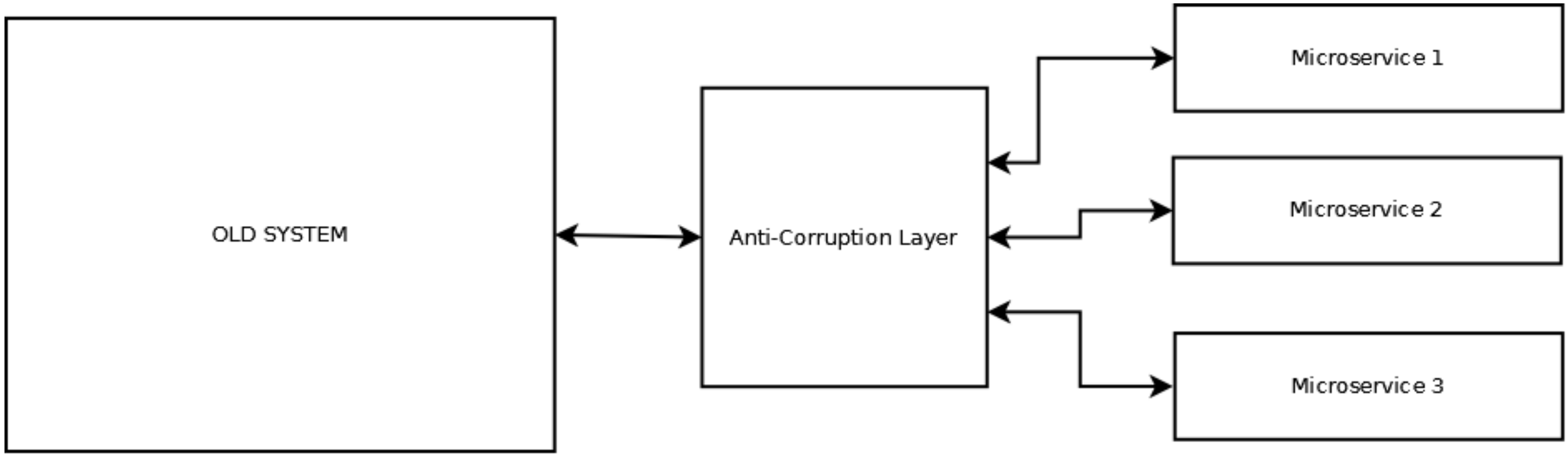


Strangler pattern



- 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.

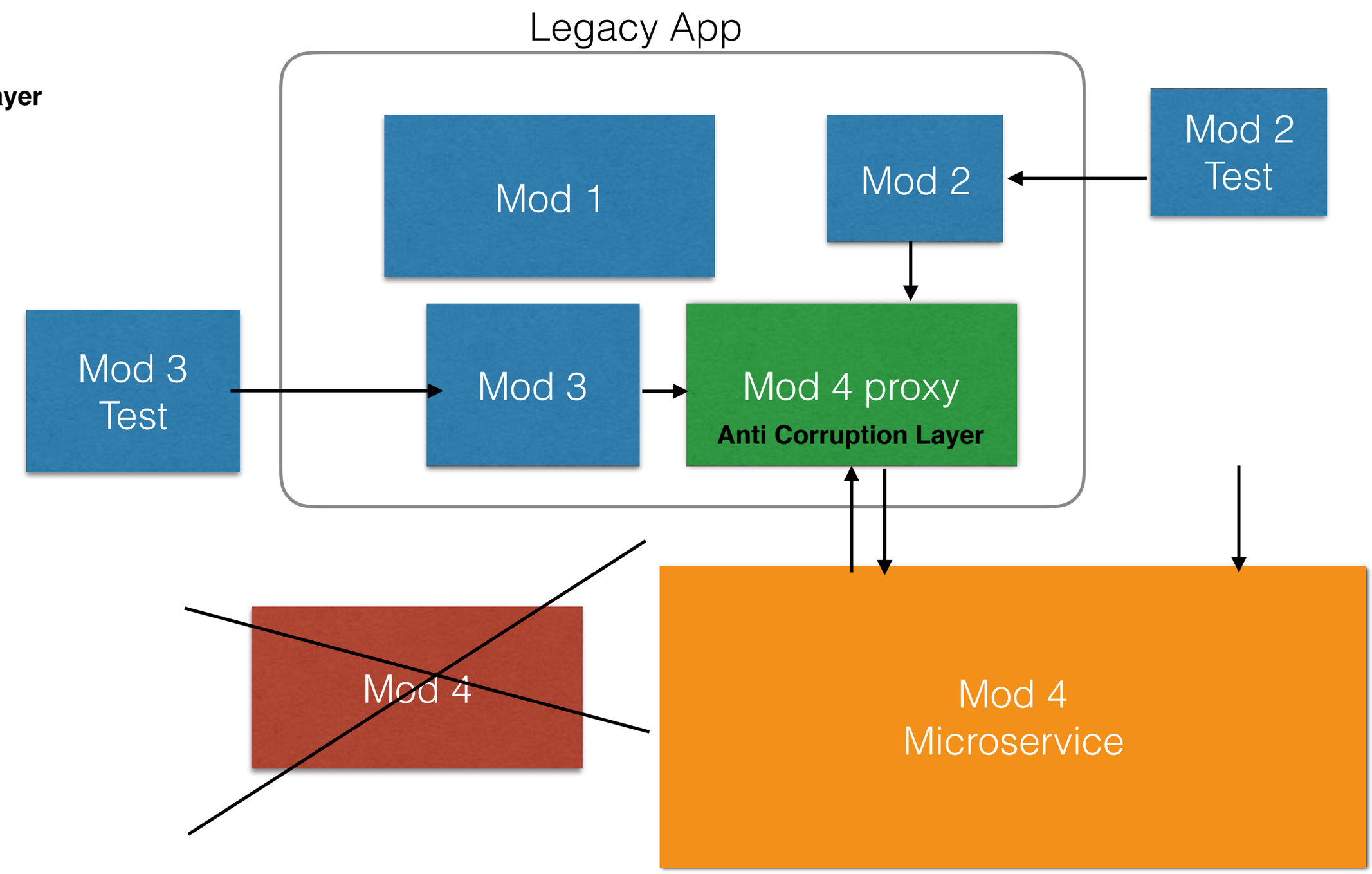
Anti-Corruption Layer

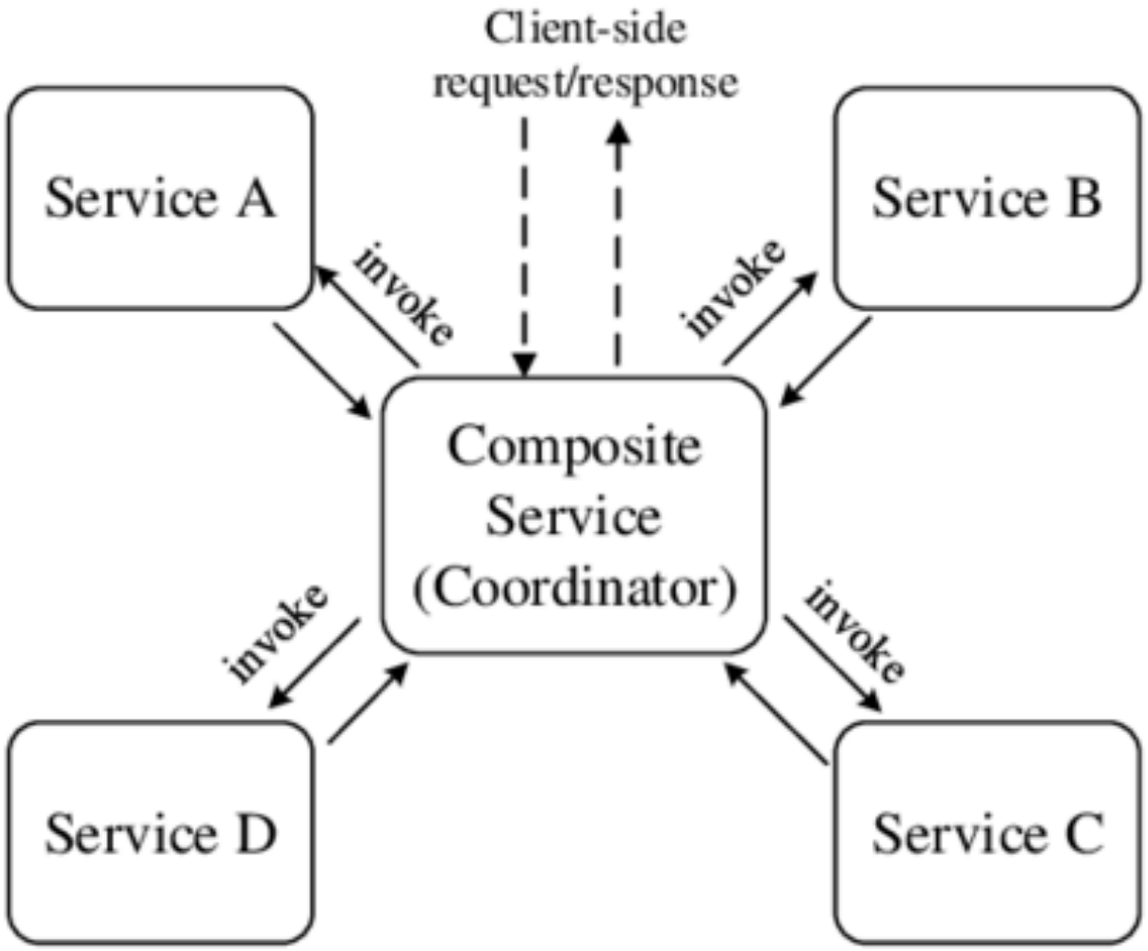


- When you create a new microservice, you notice that some business or technical assumptions or flow should be remodeled. instead of a Big Rewrite ACL maps one domain onto another so that services that use second domain do not have to be "corrupted" by concepts from the first.

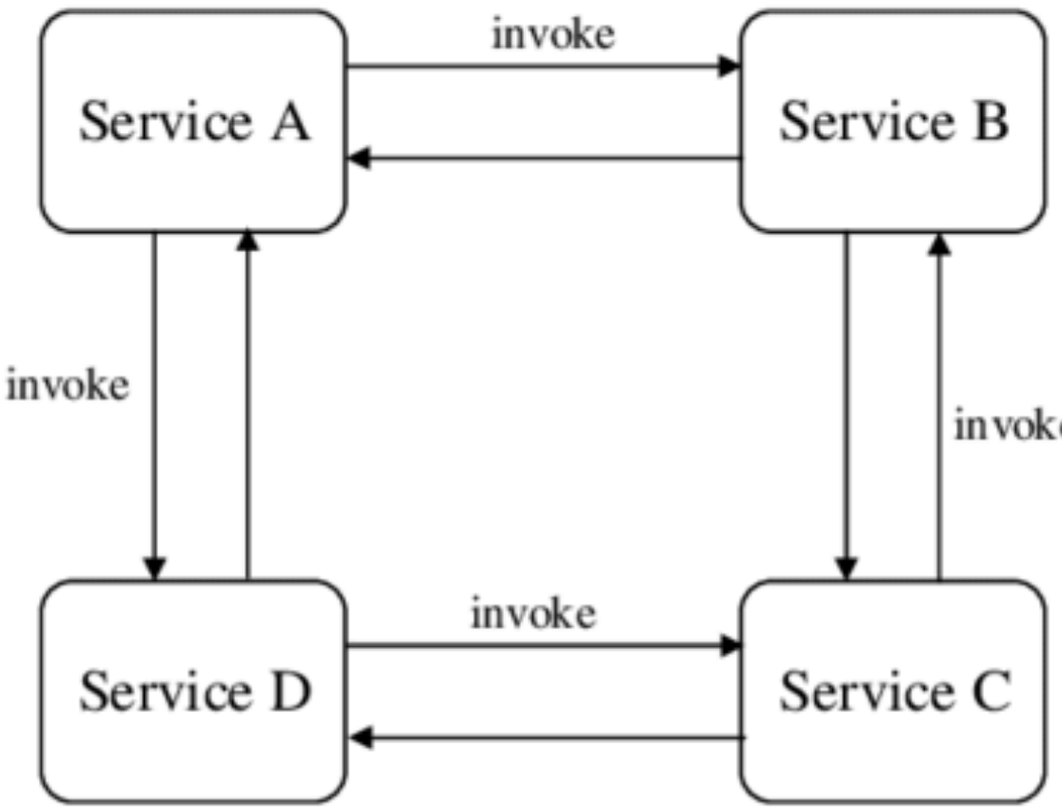
- Such modifications may result in changes to the events that a bounded context publishes.

 # unit test
 # Strangler Pattern
 # Anti Corruption Layer



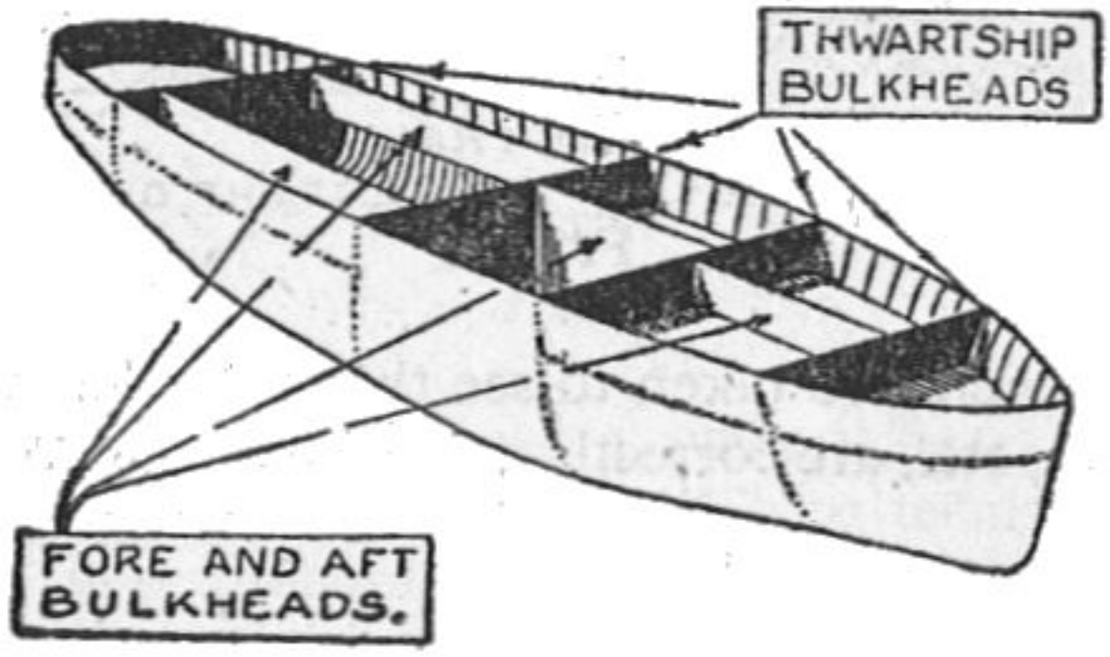


(a) Web Service Orchestration



(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.

Todo

- Performance scenarios For data entered apps