

Solution Architect





Somkiat Puisungnoen

Somkiat Puisungnoen

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Intro

Software Craftsmanship

Software Practitioner at สยามชัมนาภิกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Somkiat Puisungnoen 15 mins · Bangkok · ⚙️

Java and Bigdata



somkiat.cc

Page Messages Notifications 3 Insights Publishing Tools Settings Help ▾

somkiat.cc

@somkiat.cc

Home Posts Videos Photos

Liked Following Share ...

+ Add a Button

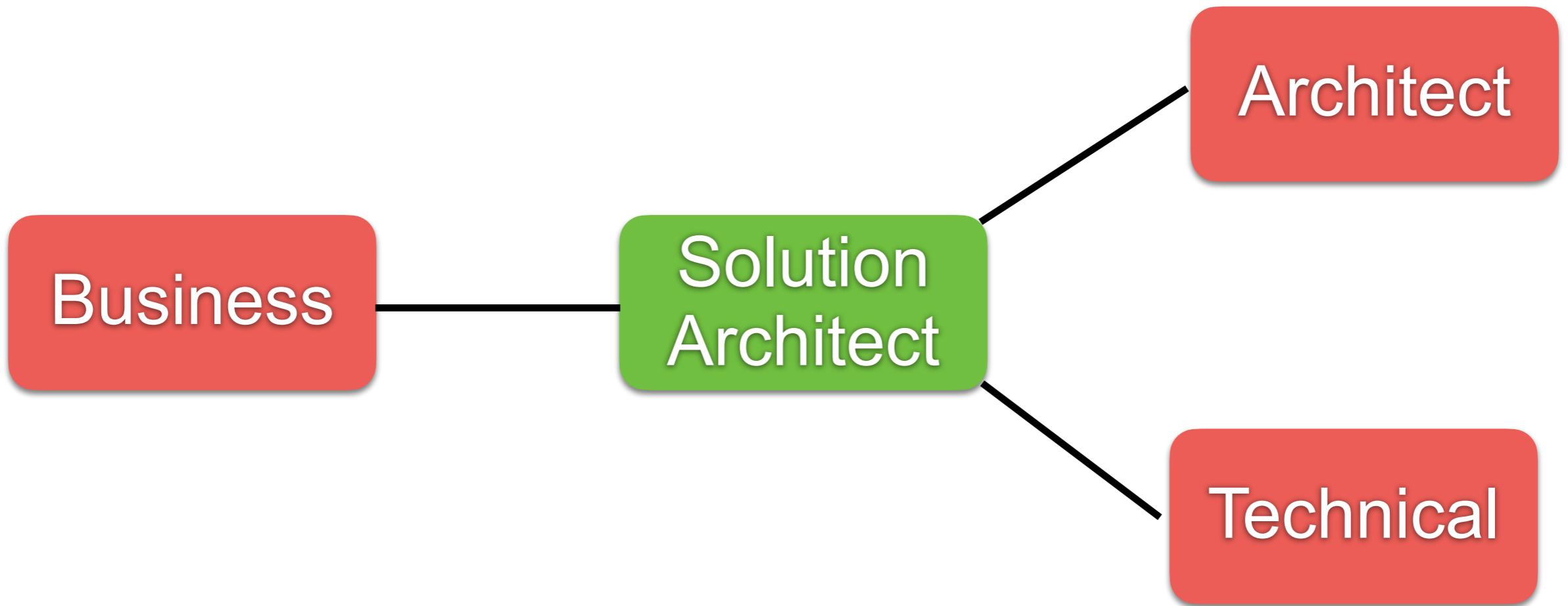


**[https://github.com/up1/
course-solution-architecture-2023](https://github.com/up1/course-solution-architecture-2023)**



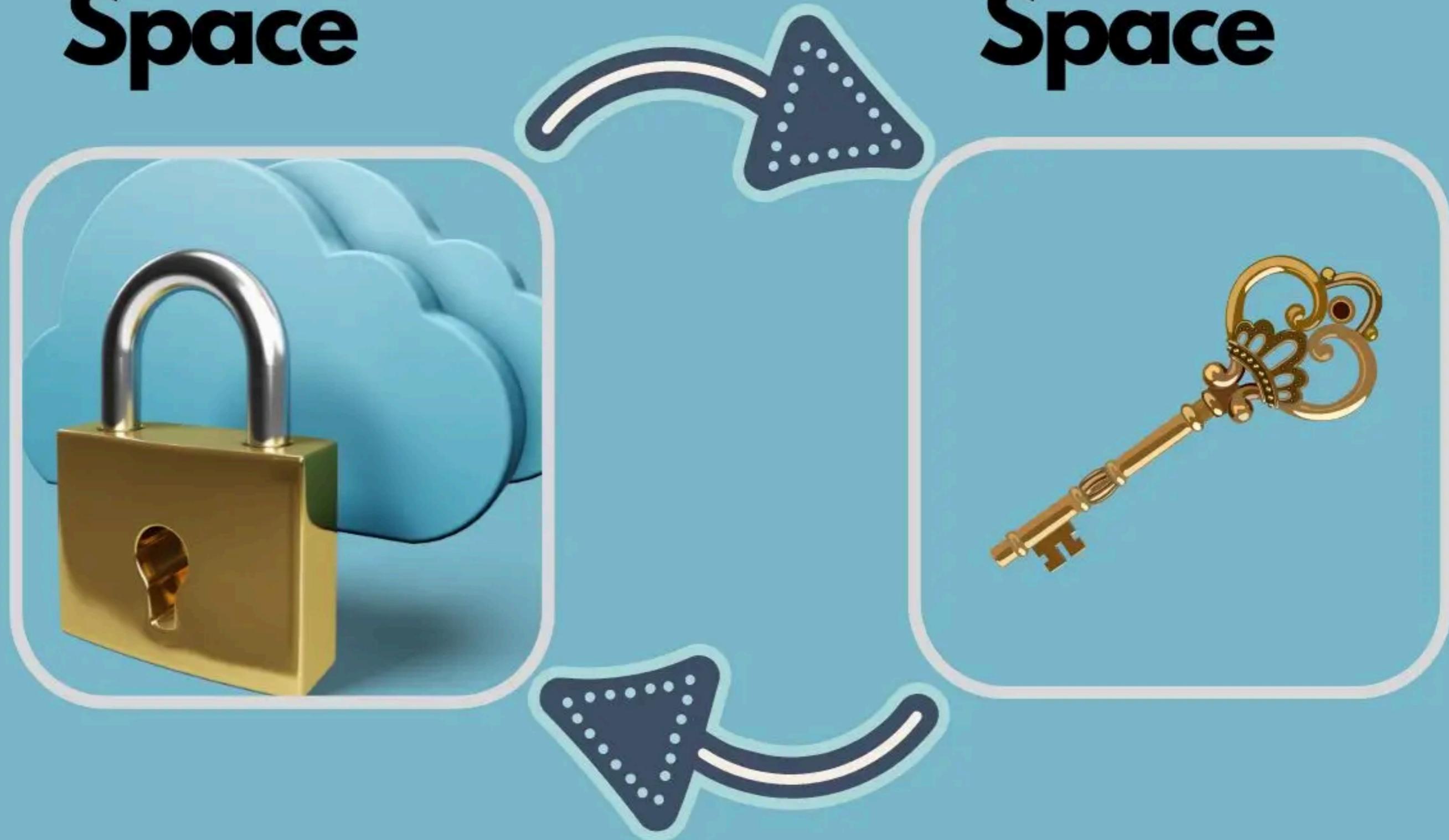
Road to Solution Architect



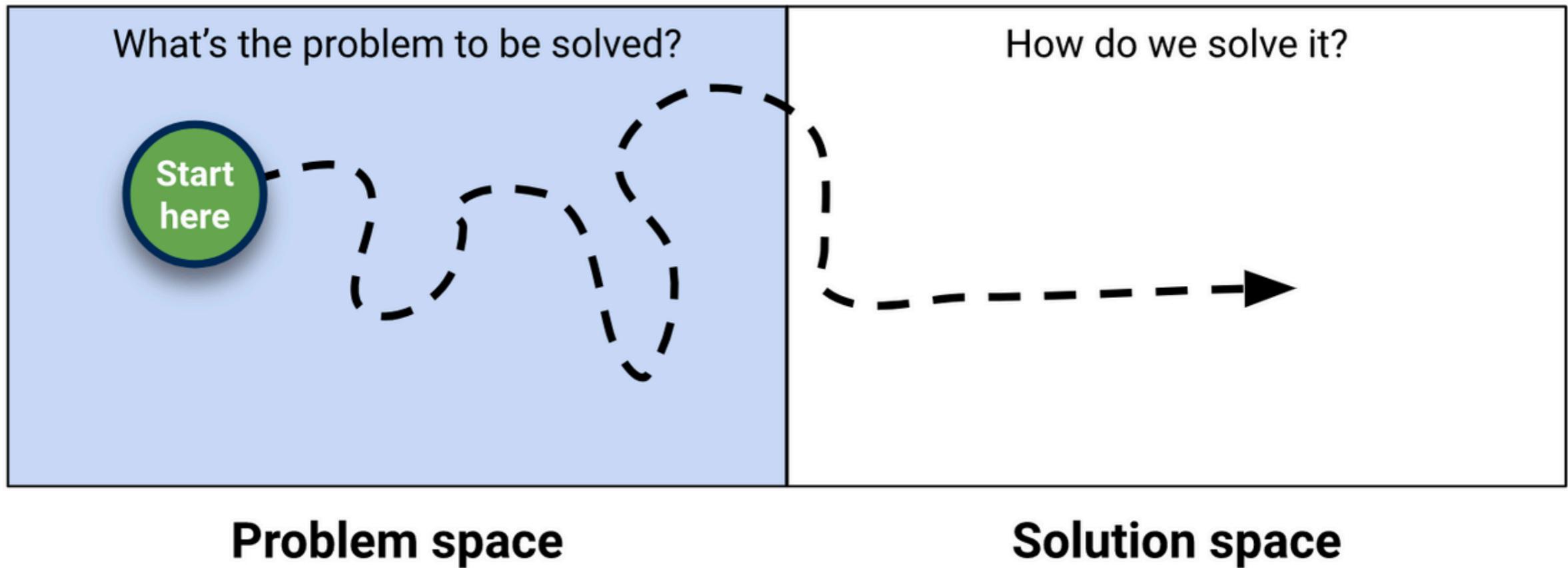


Problem Space

Solution Space



Problem and Solution Space



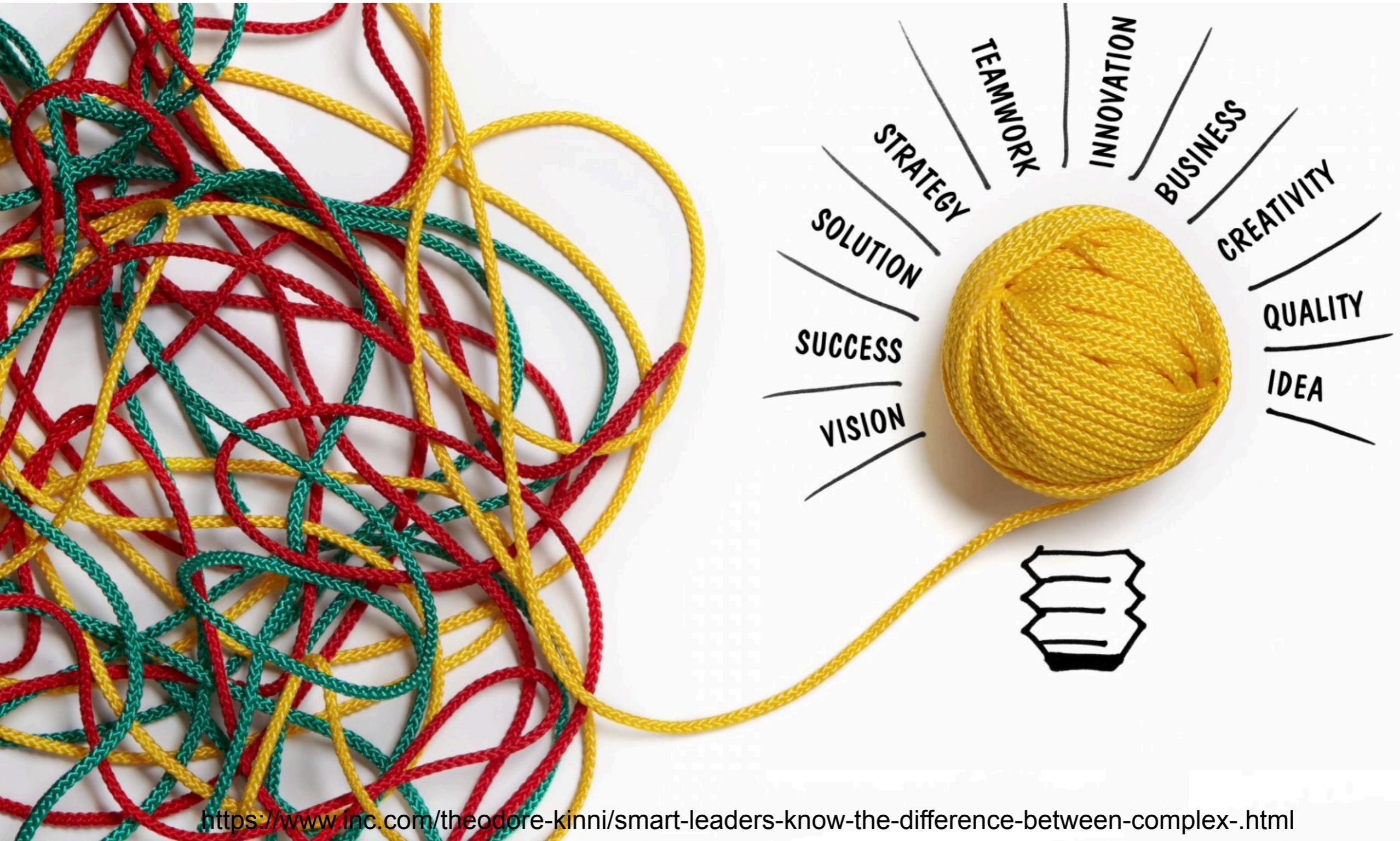
Solution Architecture

the building block for an overall enterprise software **solution** that addresses specific **problems** and **requirements**.

Long-term sustainability and solid foundation

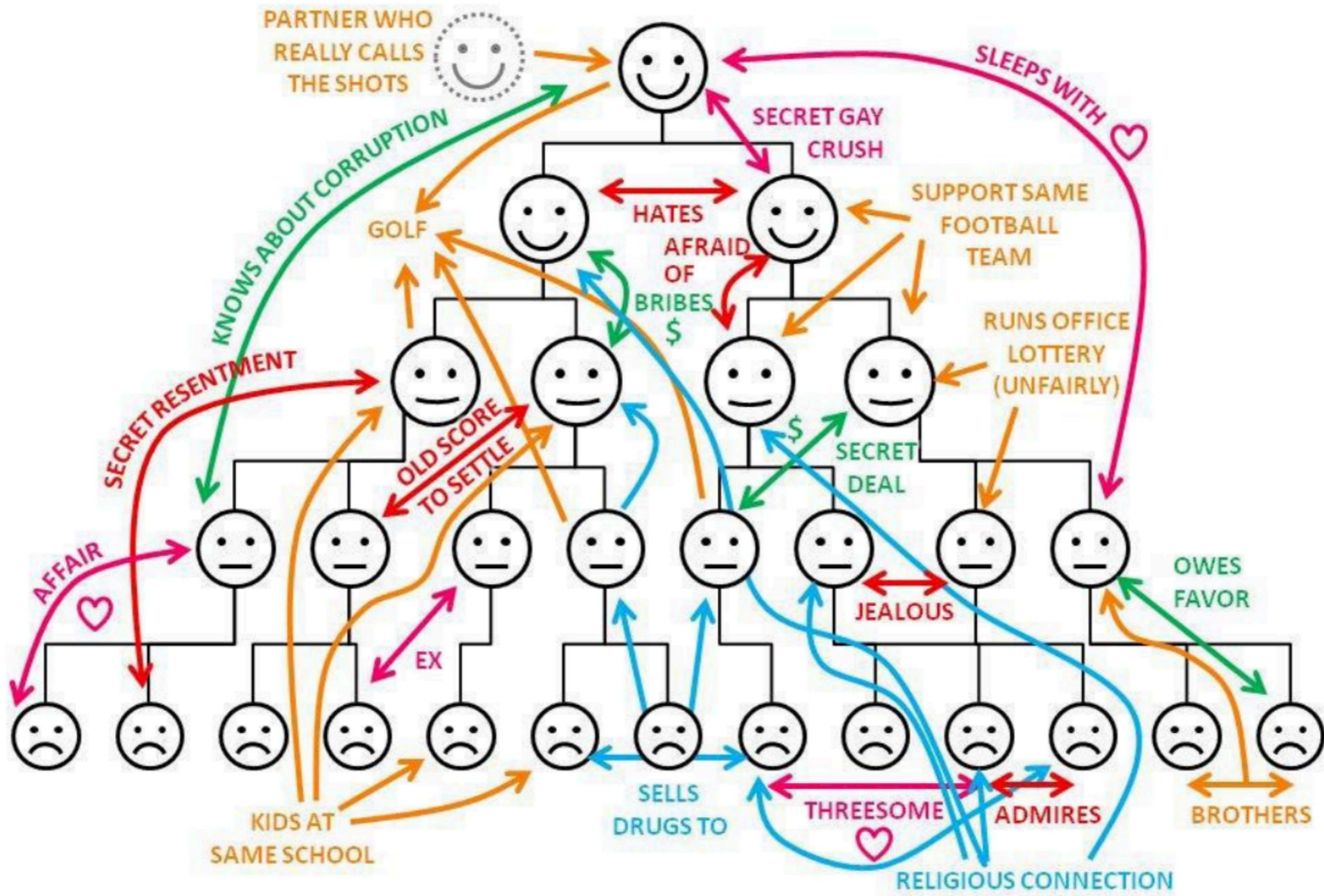


Complex vs Complicate



<https://www.inc.com/theodore-kinni/smart-leaders-know-the-difference-between-complex-.html>

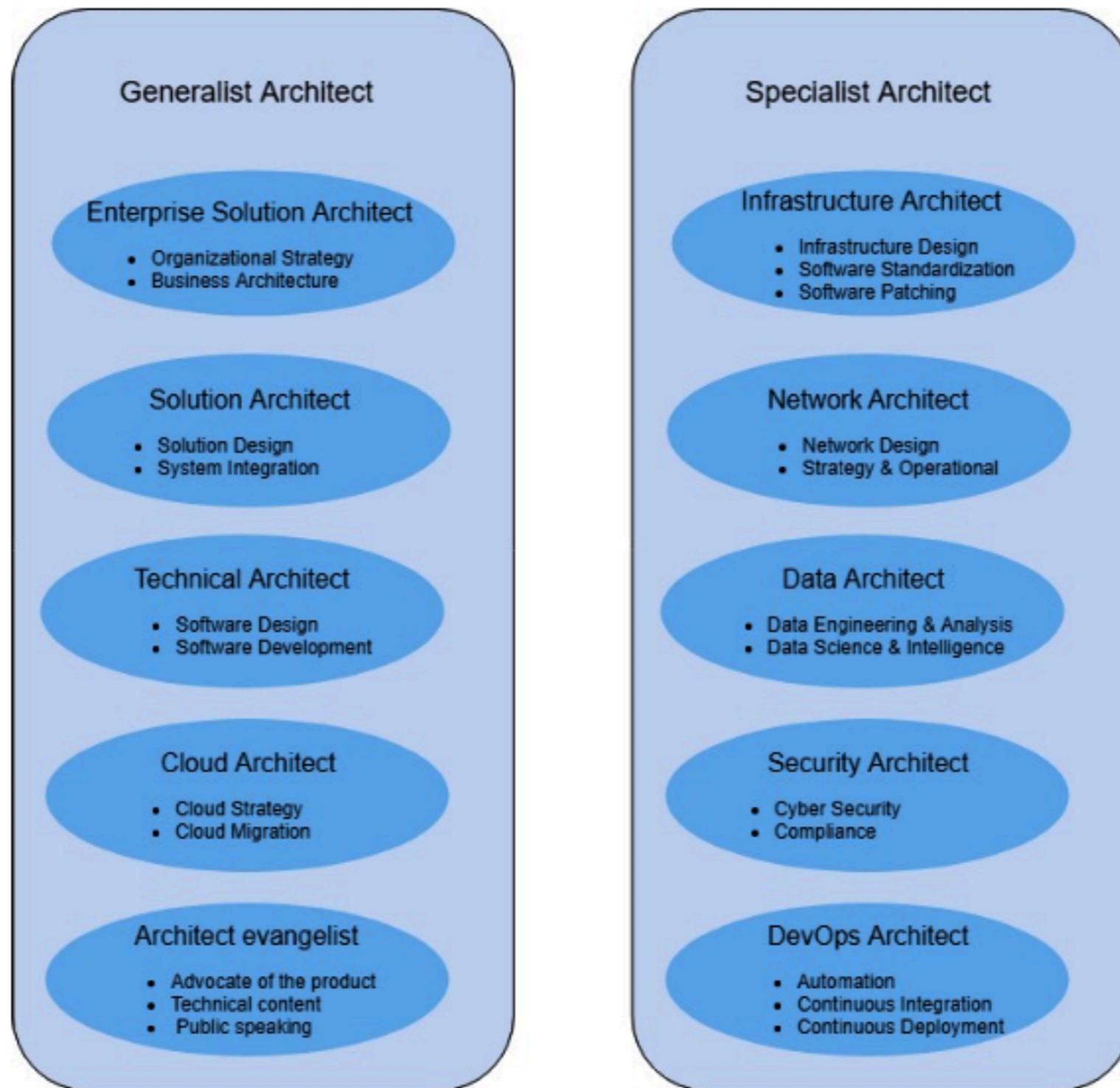




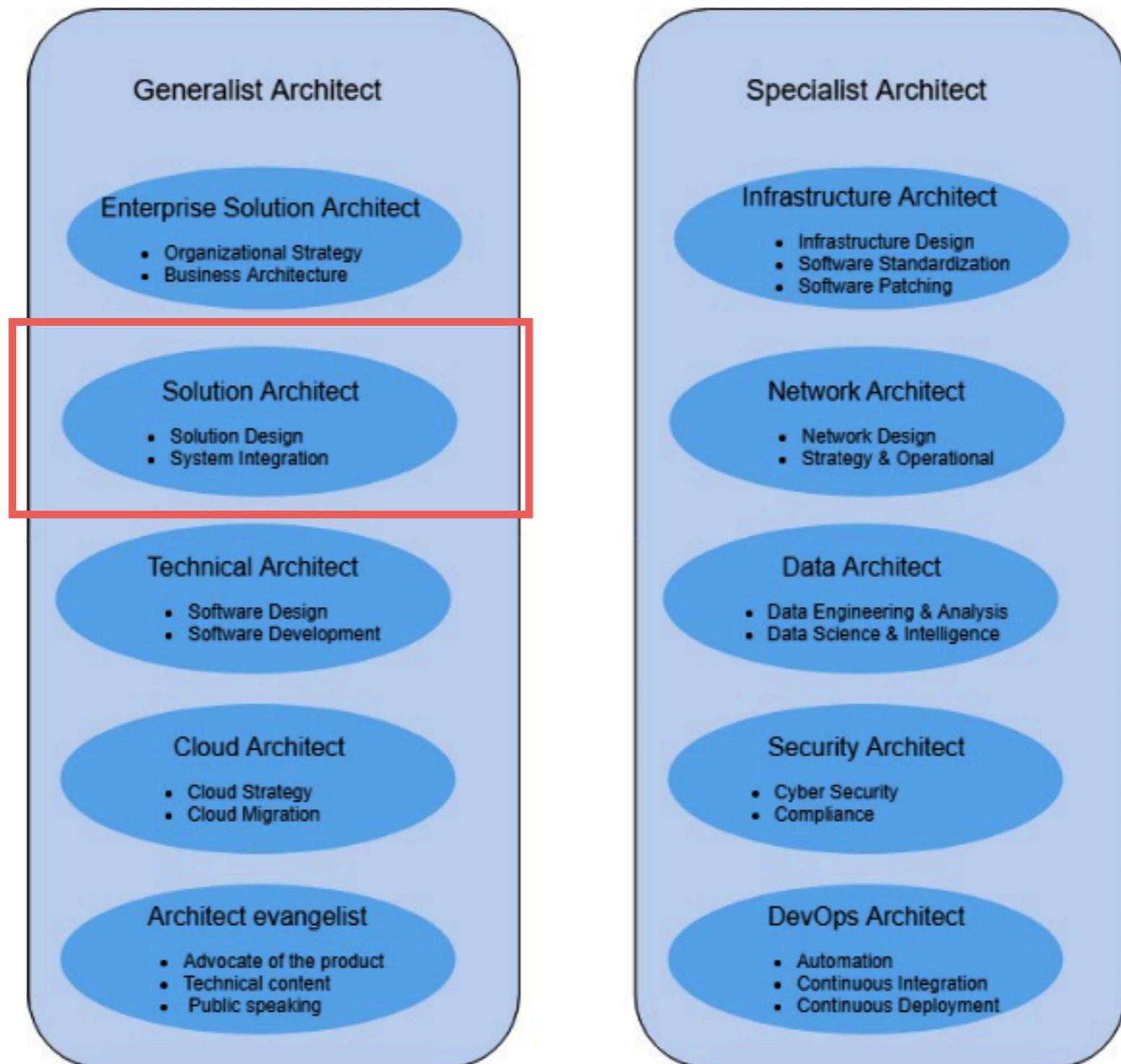
Types of Solution Architect



Types of Solution Architect



Types of Solution Architect

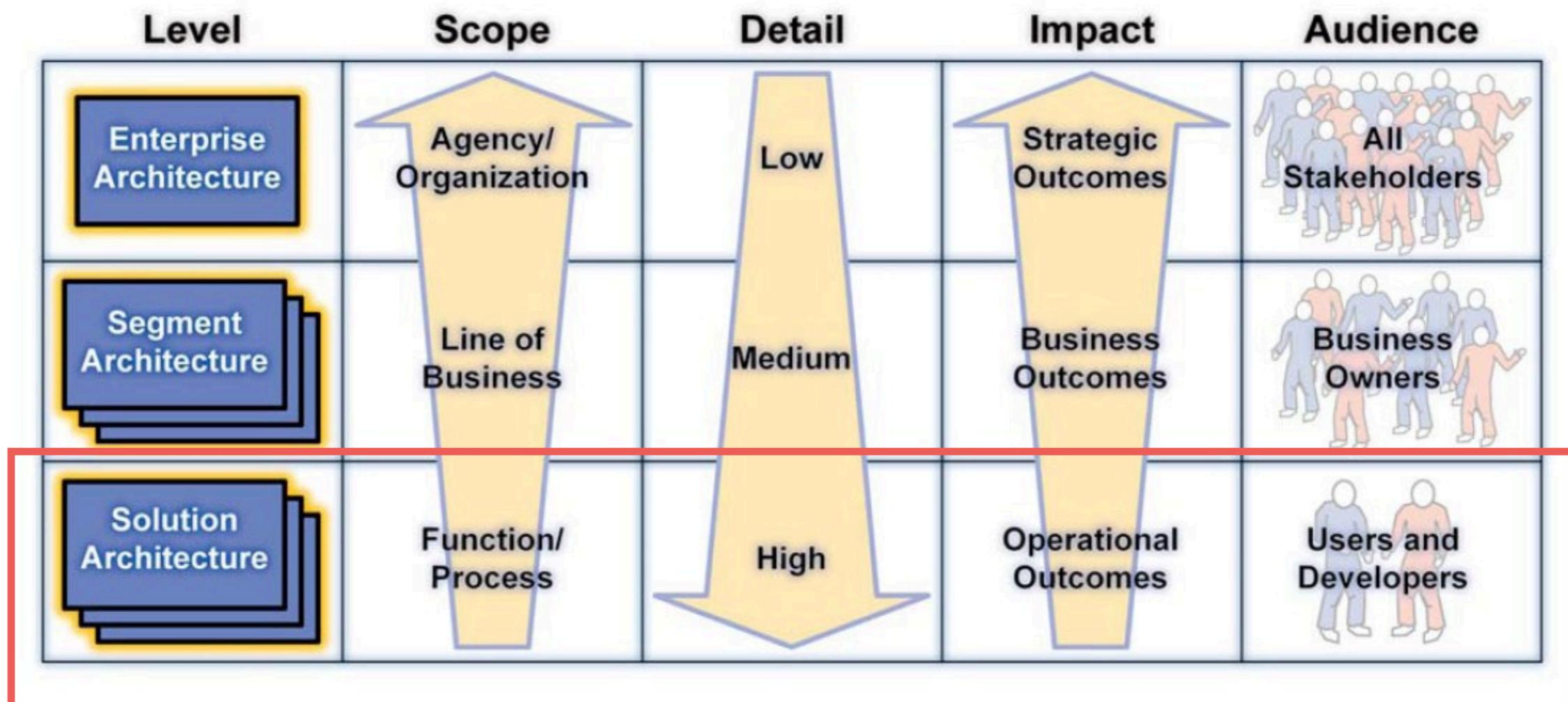


Solution Architect

Solution design
System integration



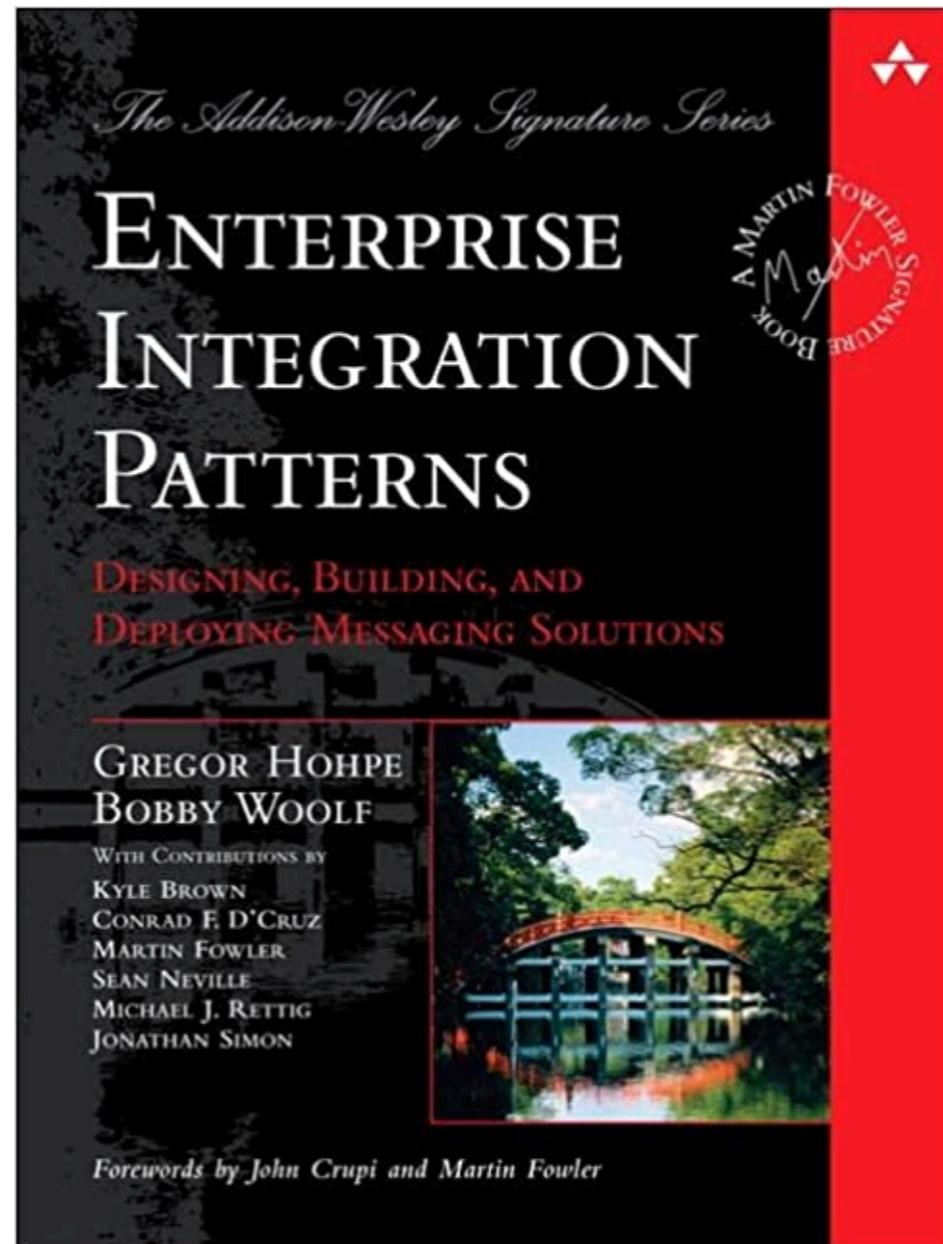
Solution Architecture



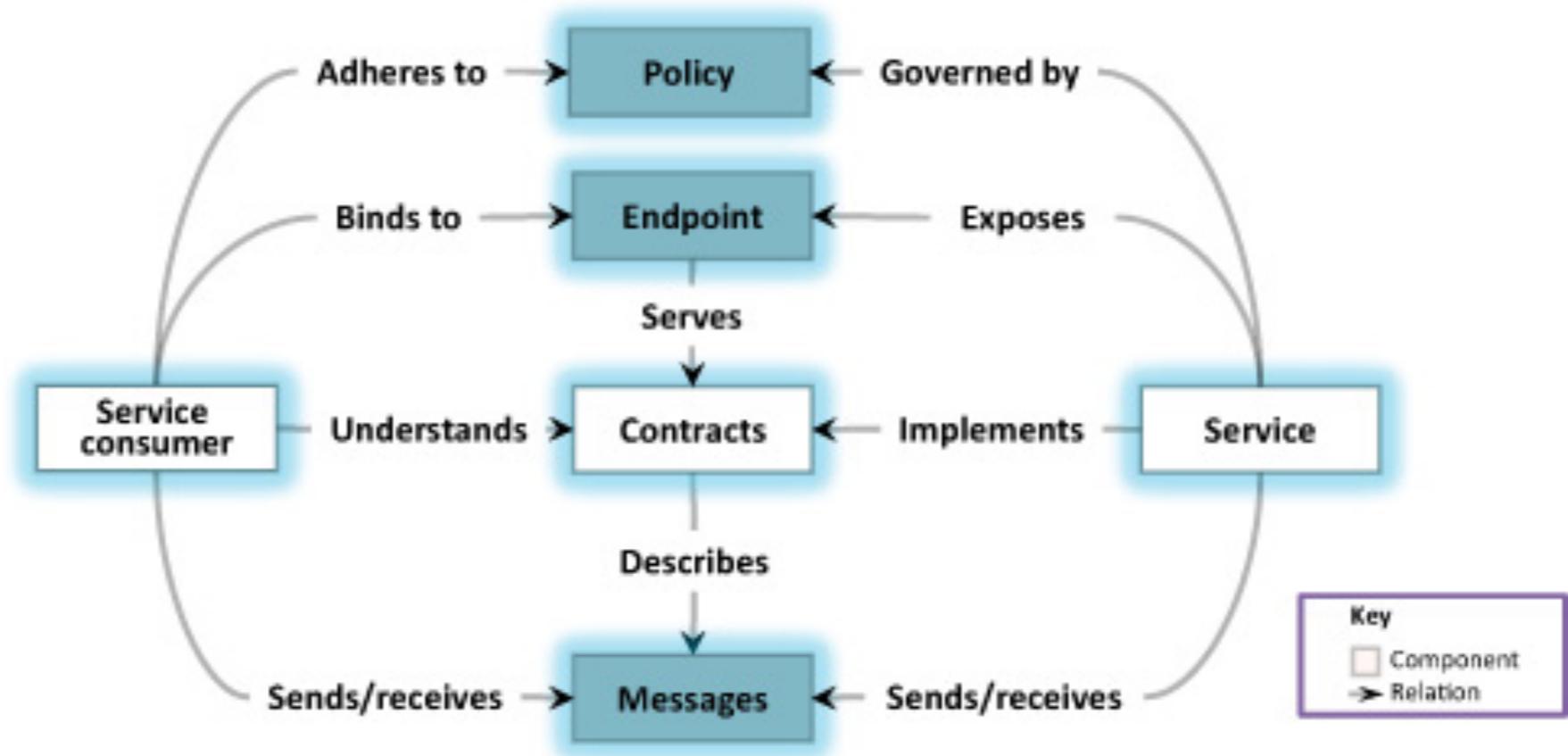
https://en.wikipedia.org/wiki/Solution_architecture



Enterprise integration patterns



System Integration



Context Mapping

Context Map Cheat Sheet

Context Map Patterns

Open / Host Service

A Bounded Context offers a defined set of services that expose functionality for other systems. Any downstream system can then implement their own integration. This is especially useful for integration requirements with many other systems. Example: public APIs.



Conformist

The downstream team conforms to the model of the upstream team. There is no translation of models. Couples the Conformist's domain model to another bounded context's model.



Anticorruption Layer

The anticonnection layer is a layer that isolates a client's model from another system's model by translation. Only couples the integration layer (or adapter) to another bounded context's model but not the domain model itself.



Shared Kernel

Two teams share a subset of the domain model including code and maybe the database. Typical examples: shared JARs, DLLs or a shared database schema. Teams with a Shared Kernel are often mutually dependent and should form a Partnership.



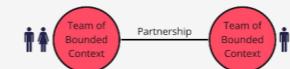
Customer / Supplier

There is a customer / supplier relationship between teams. The downstream team is considered to be the customer. Downstream requirements factor into upstream planning. Therefore, the downstream team gains some influence over the priorities and tasks of the upstream team.



Partnership

Partnership is a cooperative relationship between two teams. These teams establish a process for coordinated planning of development and joint management of integration.



Published Language

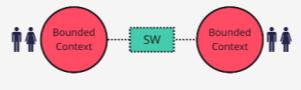
A Published Language is a well documented shared language between Bounded Contexts which can translate in and out from that language. Published Language is often combined with Open Host Service. Typical examples are iCalendar or vCard.



Bounded Context
OHS + PL

Separate Ways

Bounded Contexts and their corresponding teams have no connections because integration is sometimes too expensive or it takes very long to implement. The teams chose to go separate ways in order to focus on their specific solutions.



Team Relationships

Mutually Dependent

Two software artifacts or systems in two bounded contexts need to be delivered together to be successful and work. There is often a close, reciprocal link between data and functions between the two systems.



Free

Changes in one bounded context do not influence success or failure in other bounded contexts. There is, therefore, no organizational or technical link of any kind between the teams.



Upstream / Downstream

Actions of an upstream team will influence the downstream counterpart while the opposite might not be true. This influence can apply to code but also on less technical factors such as schedule or responsiveness to external requests.



Context Map Cheat Sheet v2: https://github.com/ddd-crew/context_mapping | License: Creative Commons Attribution-ShareAlike 4.0 | Contains quotes from the DDD Reference by Eric Evans https://domainlanguage.com/wp-content/uploads/2016/05/DDD_Reference_2015-03.pdf



Responsibilities

Define the solution architecture

Assess and mitigate risk

Lead the development team

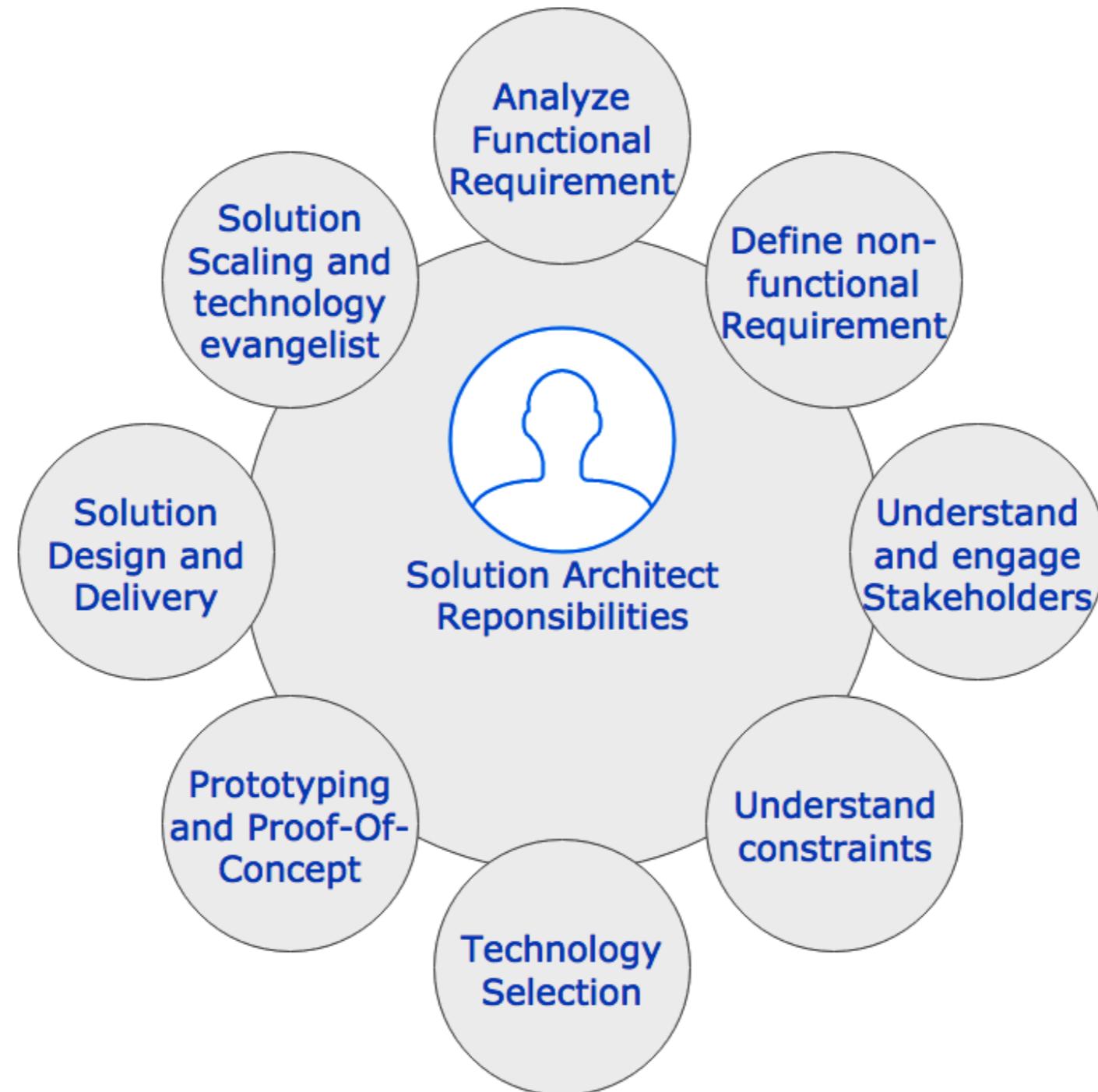
Evaluate and select technology

Ensure scalability and maintainability

**Communicate with stakeholder
(technical and non-technical)**



Responsibilities



**Ensure the org's IT solution
Effective, Efficient
and align with business goal**



Define the solution architecture

Identify business requirement

Overall solution architecture

Provide technical specification



Balance between Functional and Non-Functional



Non-Functional
Requirements

Functional
Requirements

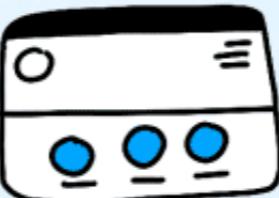


FUNCTIONAL REQUIREMENTS

examples of

NON-FUNCTIONAL REQUIREMENTS

the website will have a homepage



the homepage should load within 1.5 seconds

the website will store customer data



customer data will be encrypted to level 4 encryption standards

website customers can log into their accounts



the website has the capacity for 5.000 logged-in users at any one time

the website will always be available to customers



the website must have a 99.7% uptime

customers can access the websites on their phones

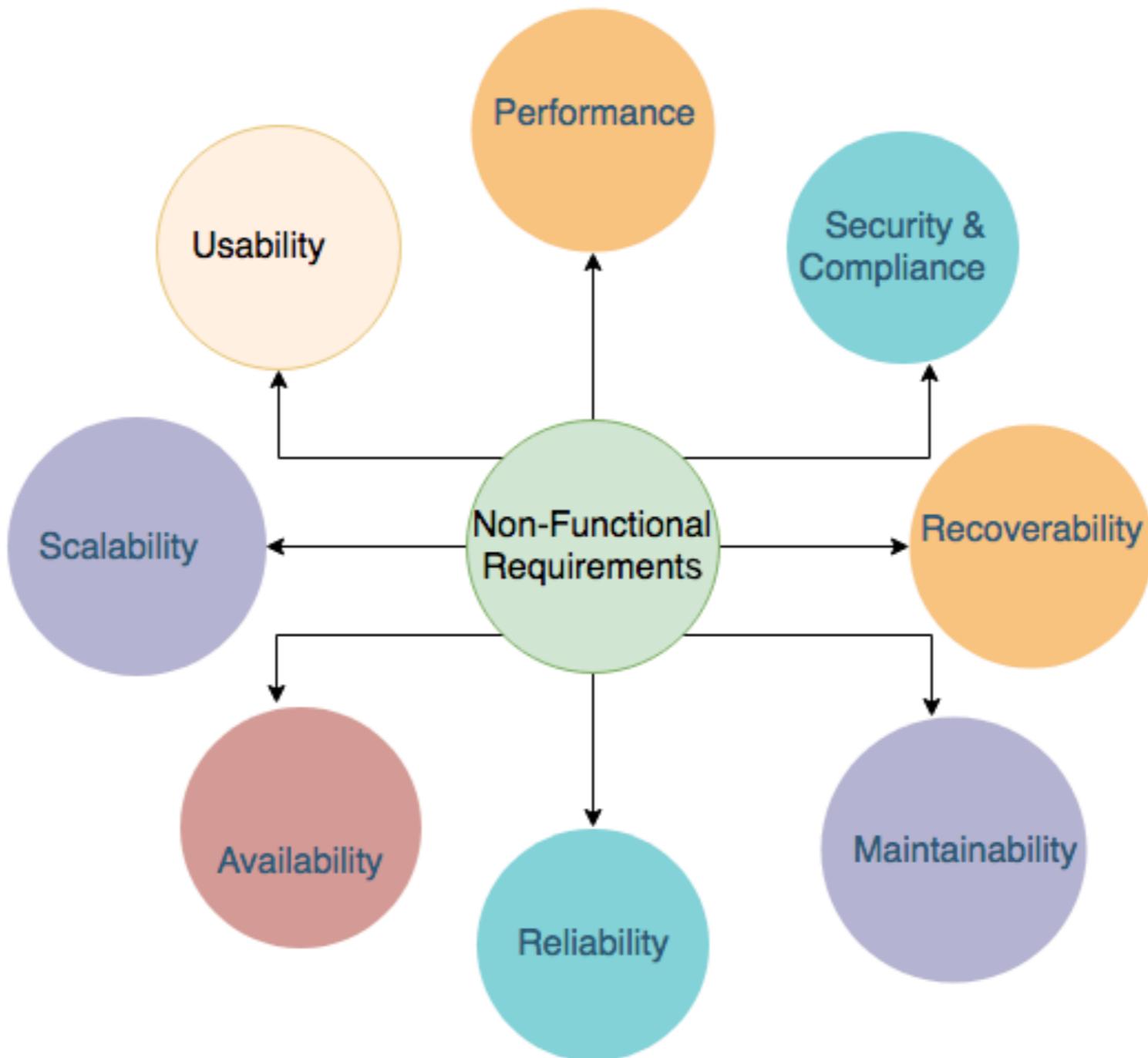


the website will be compatible with iOS 12.8 and above

*plano



NFRs ?



NFRs in most project

High availability

Scalability

Application performance

Network request/response latency

Disaster recovery

Security and compliance

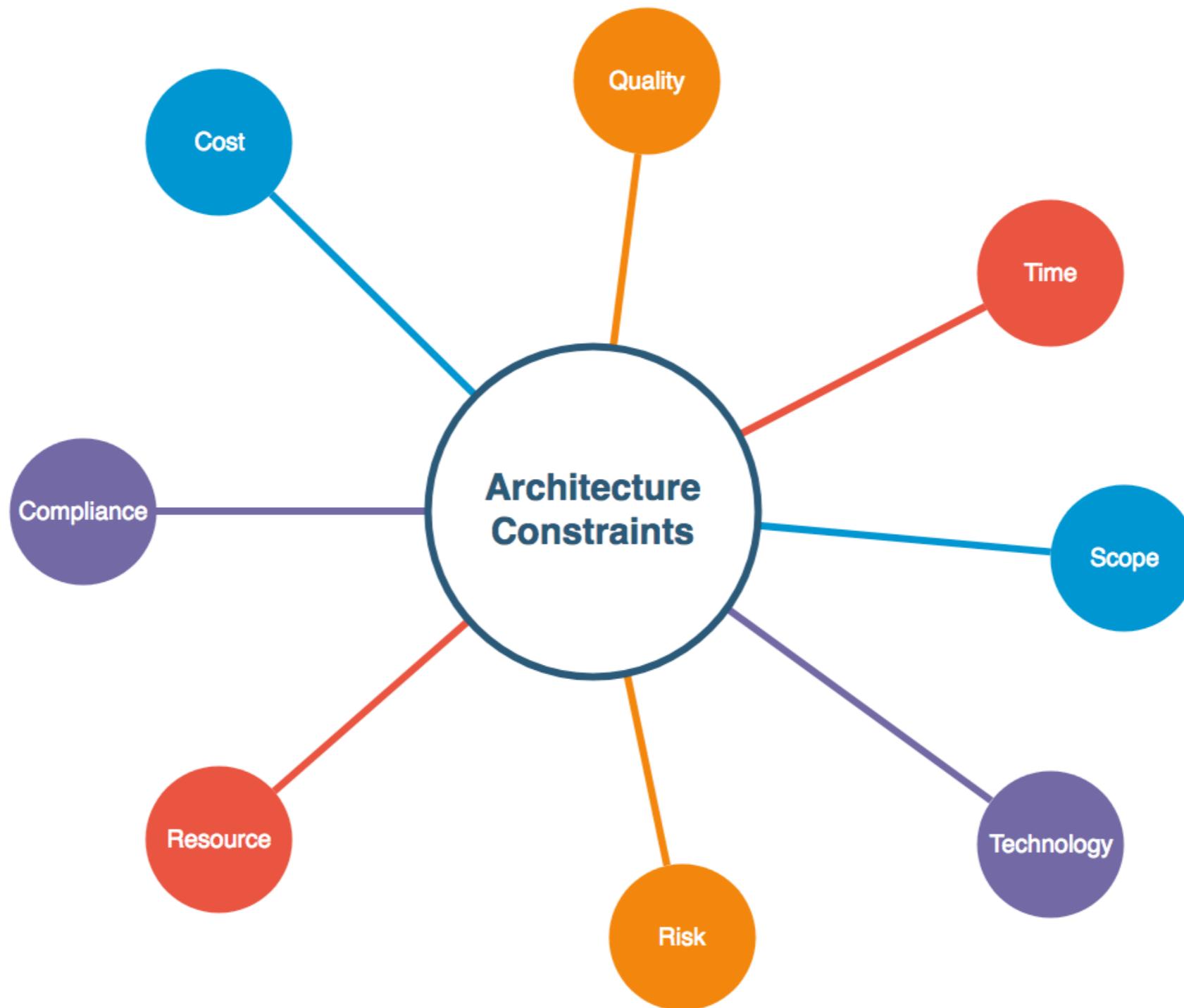


Solution with constraints

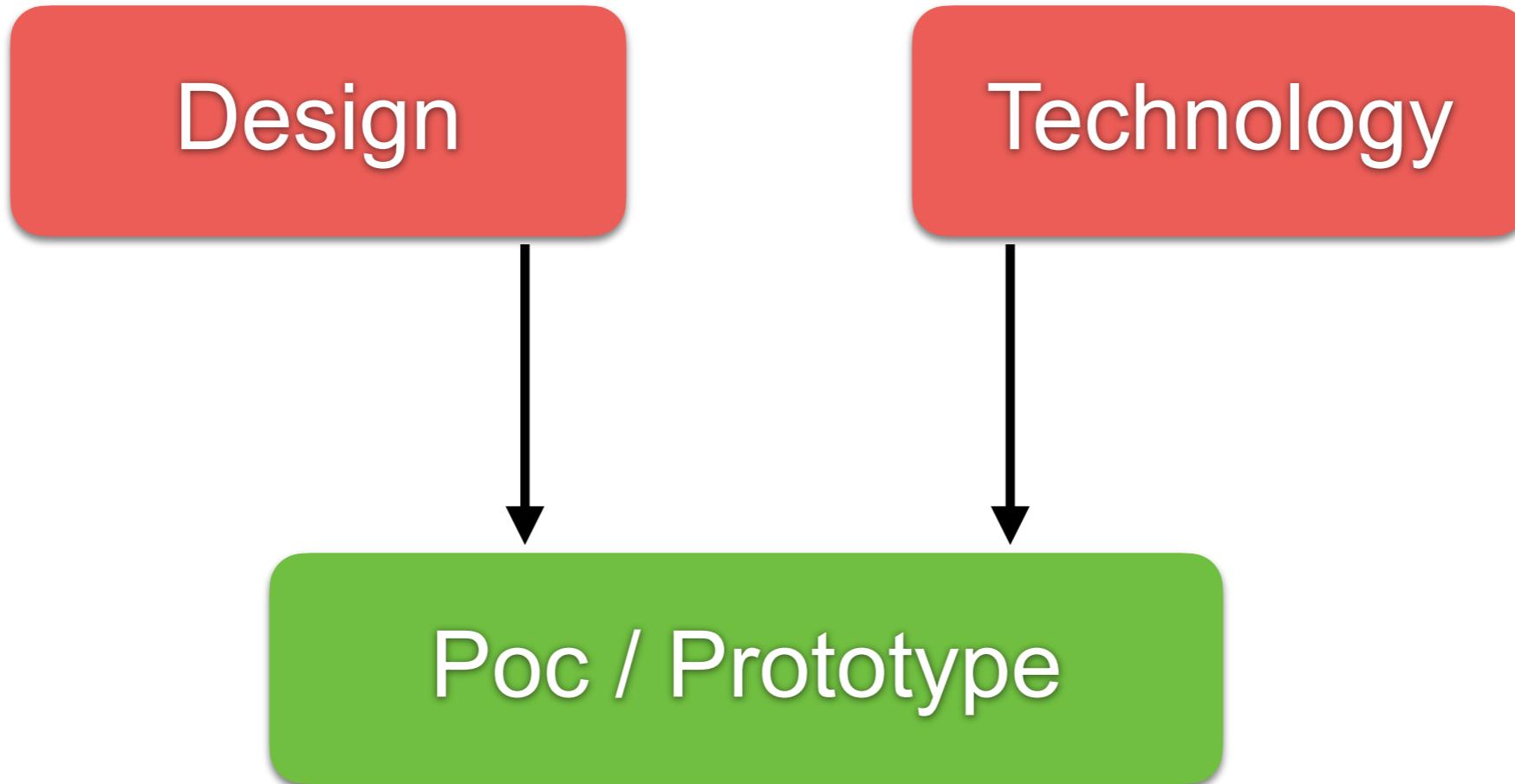
Cost
Budget
Timeline
Regulatory
etc ...



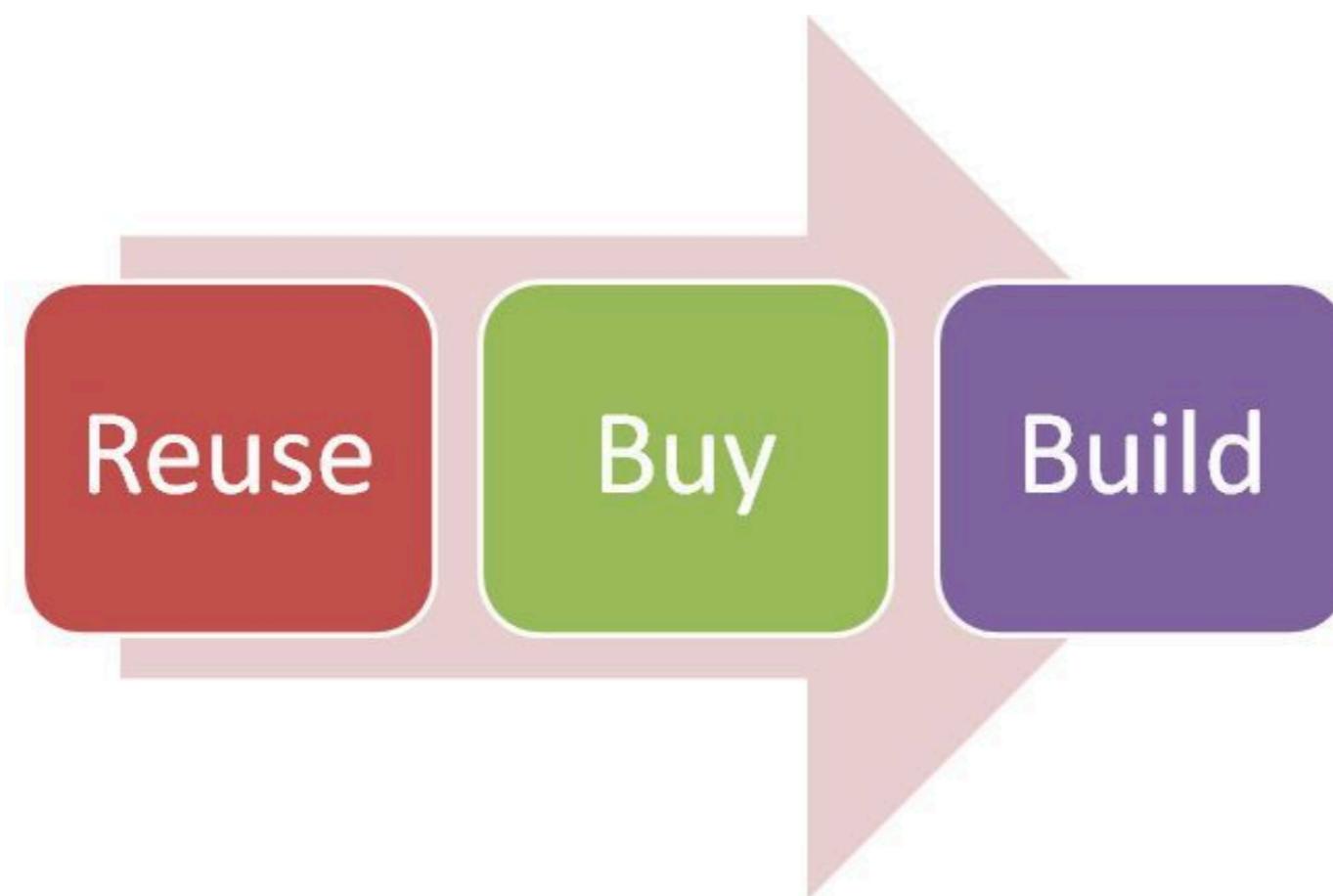
Solution with constraints



Solution with constraints



Solution with constraints



Thinking about Reuse !!

Real resume ?

Client side ?

Site effect from change ?



Just enough Solution Architecture



Minimal impact

Just enough Solution Architecture

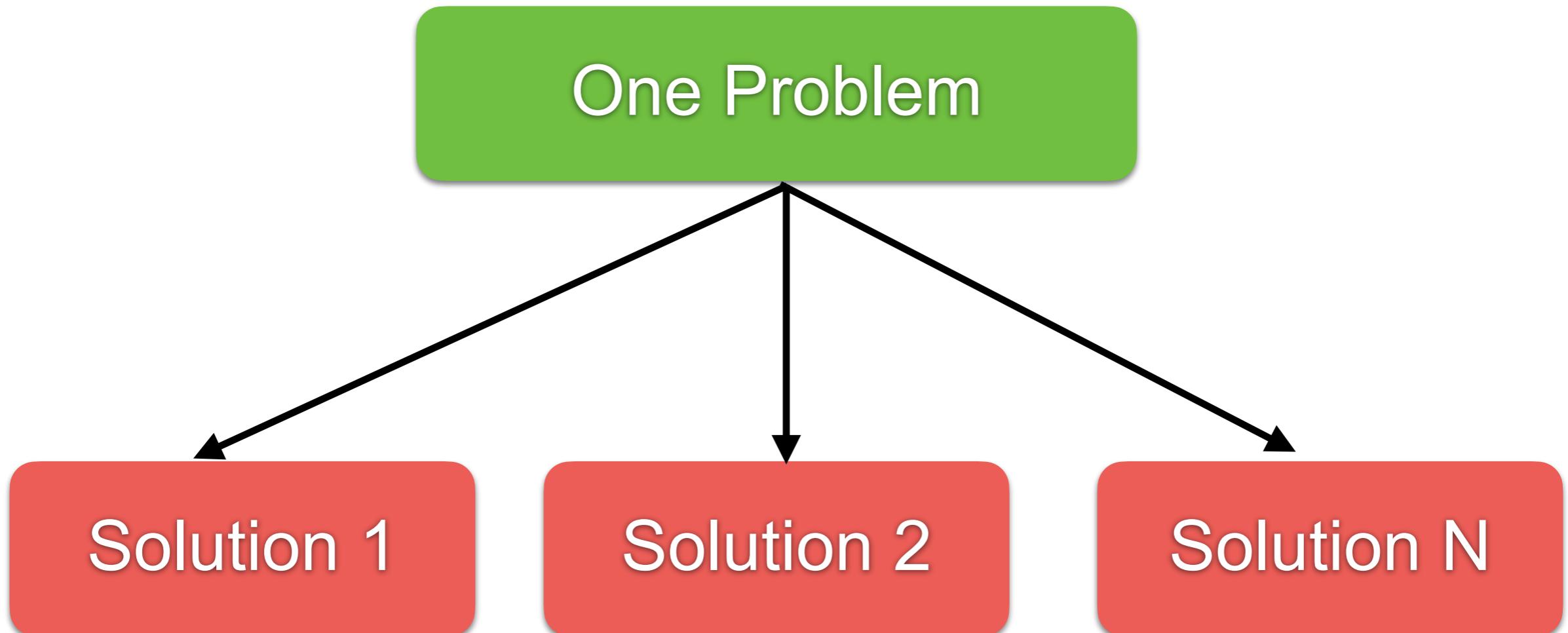
Fit with current environments



Mazimize Return on Investment (ROI)

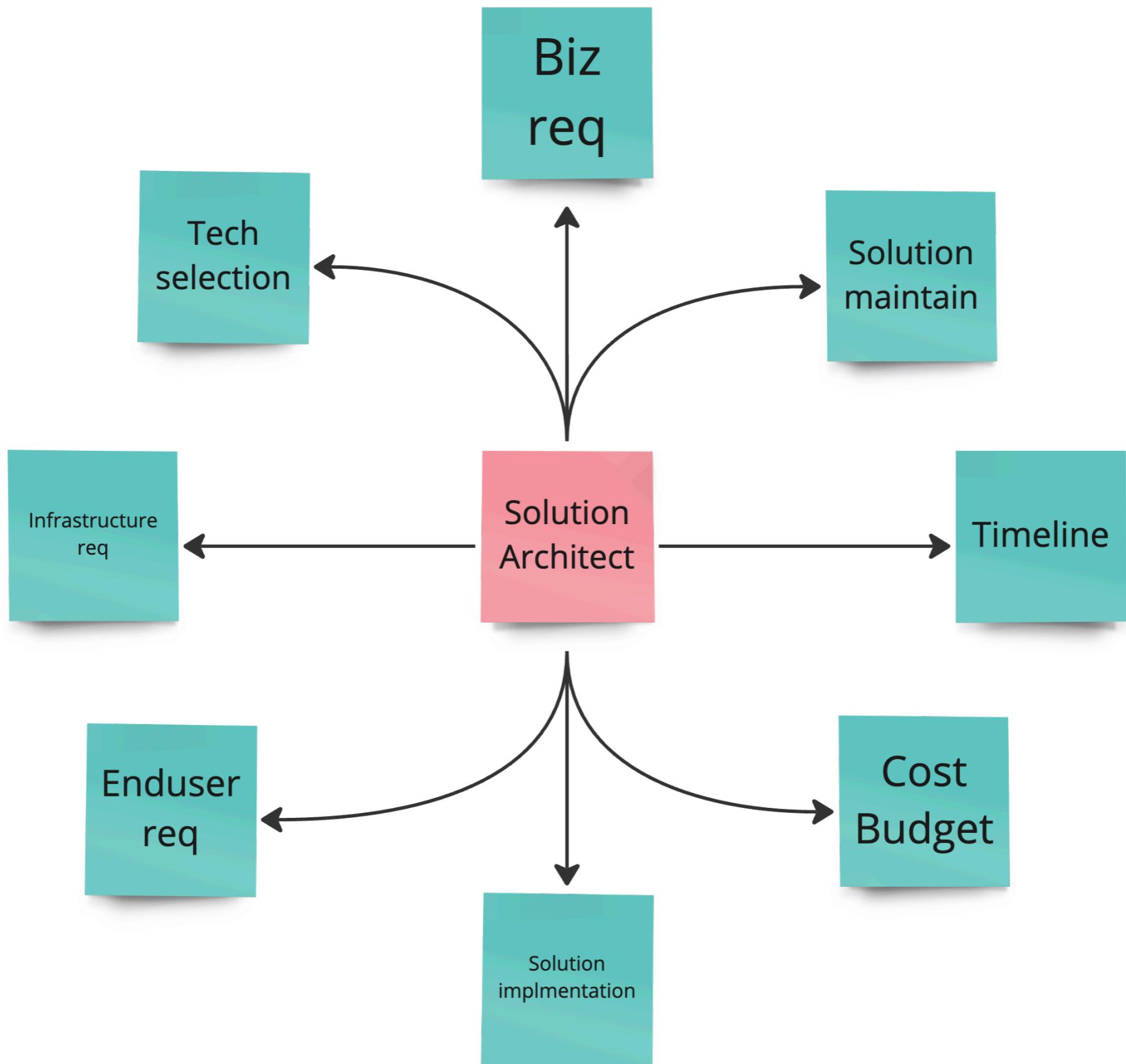


Best Solution ?



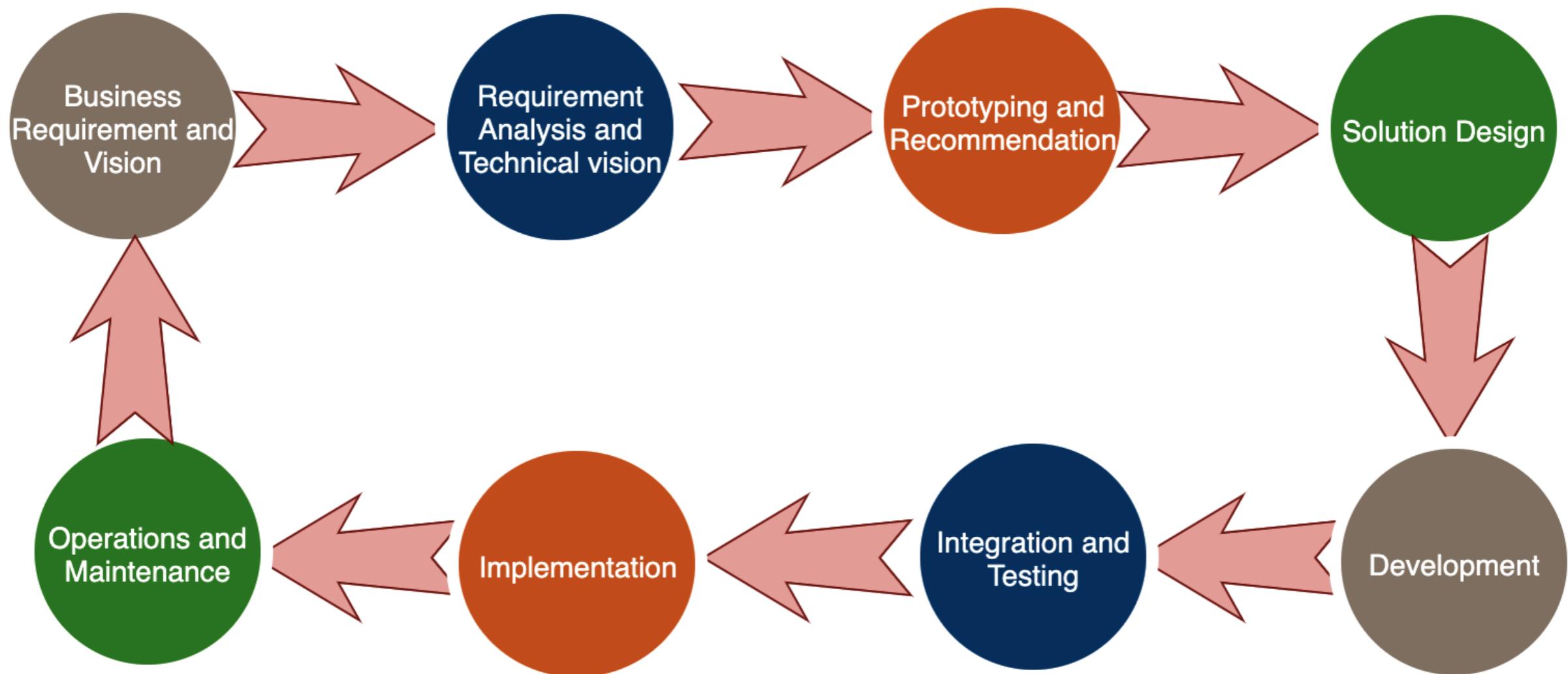
Business needs Quality of delivery





Solution Delivery Life Cycle

Iterative process



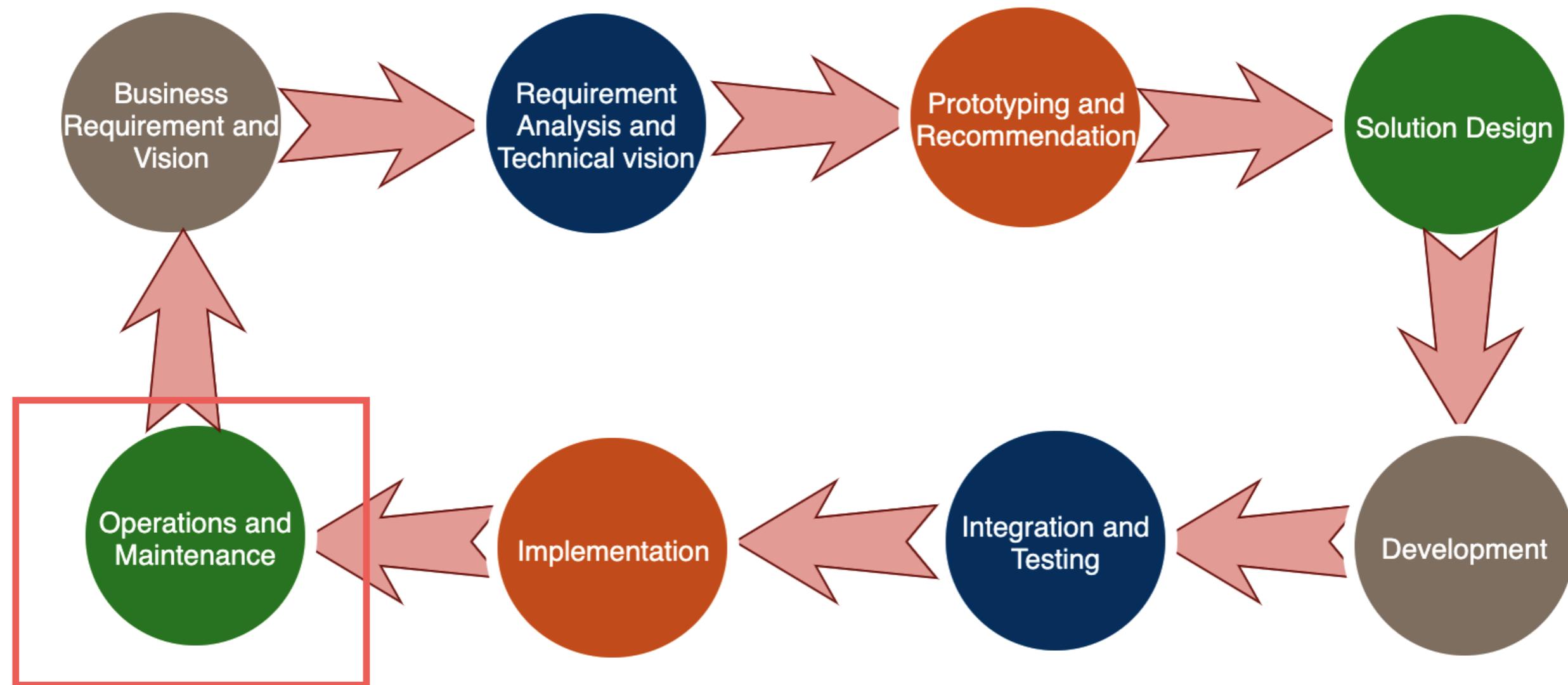
TODOs with Documents

Solution standard
High-level design
Cross-system integration
Implementation approach
Monitoring/observability approach
Pros/Cons of design choices
Audit and compliance requirements



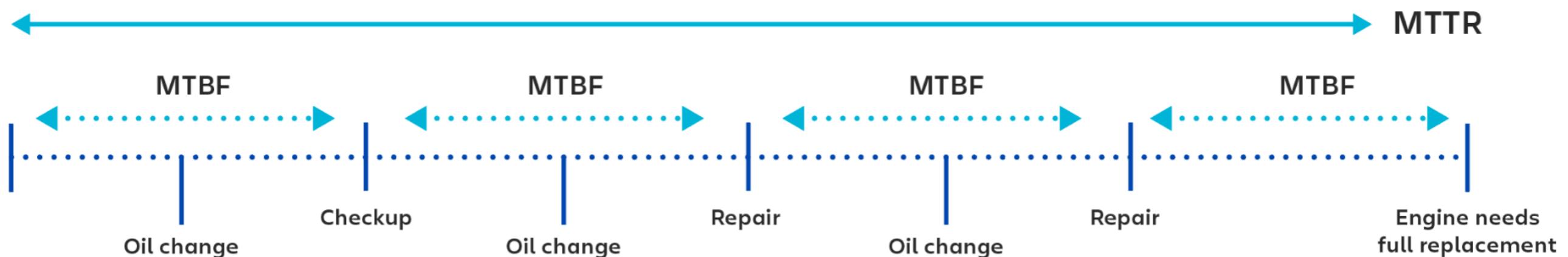
Solution Delivery Life Cycle

Iterative process



Operation and Maintenance

How to scale product to meet demand
Ensure High H/A without impact



DevOps key metrics

The Four Key Metrics

Accelerate by Nicole Forsgren, PhD, Jez Humble, and Gene Kim



1

LEAD TIME

Lead time is the time it takes to go from a customer making a request to the request being satisfied. Shorter lead times enable faster feedback.

DEPLOYMENT FREQUENCY

Deployment frequency is a proxy metric for batch size; the more frequently you deploy the smaller the size of the batch. Small batch sizes reduce cycle times, reduce risk and overhead, improve efficiency, increase motivation and urgency, and reduce costs and schedule growth.

2

3

MEAN TIME TO RESTORE

Reliability is traditionally measured as time between failures, but in a modern software organization failure is inevitable. Thus, reliability is measured by how long it takes to restore service when a failure occurs.

CHANGE FAIL PERCENTAGE

This metric looks at the percentage of changes made to production that fail; the same as percent complete and accurate in Lean product delivery.

4



Architecture

© 2022 - 2023 Siam Chamnankit Company Limited. All rights reserved.

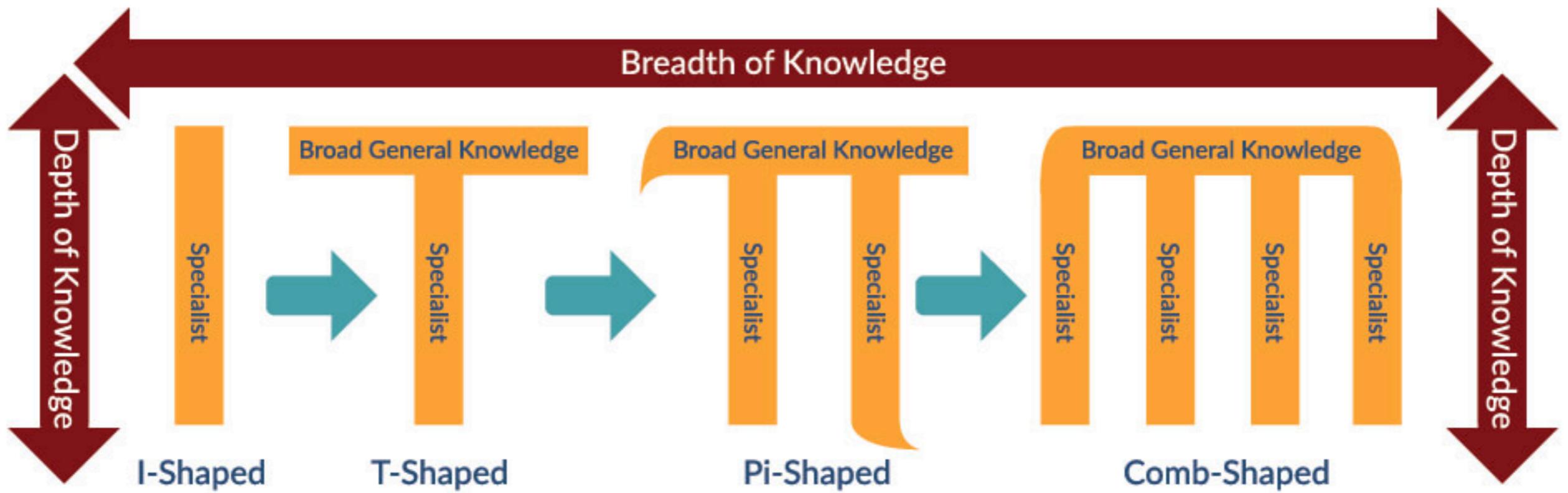
Productivity



<https://cloud.google.com/blog/products/devops-sre/the-2019-accelerate-state-of-devops-elite-performance-productivity-and-scaling>



Shape of Skills



Evolution of Architecture

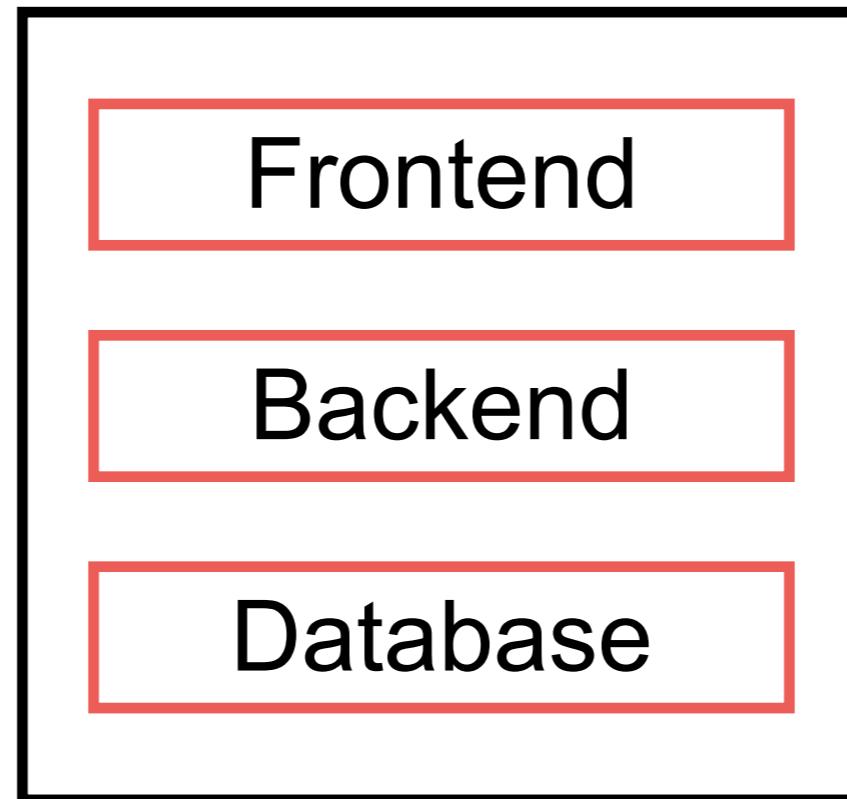


Monolith

App



Layer/Tier



Layer/Tier

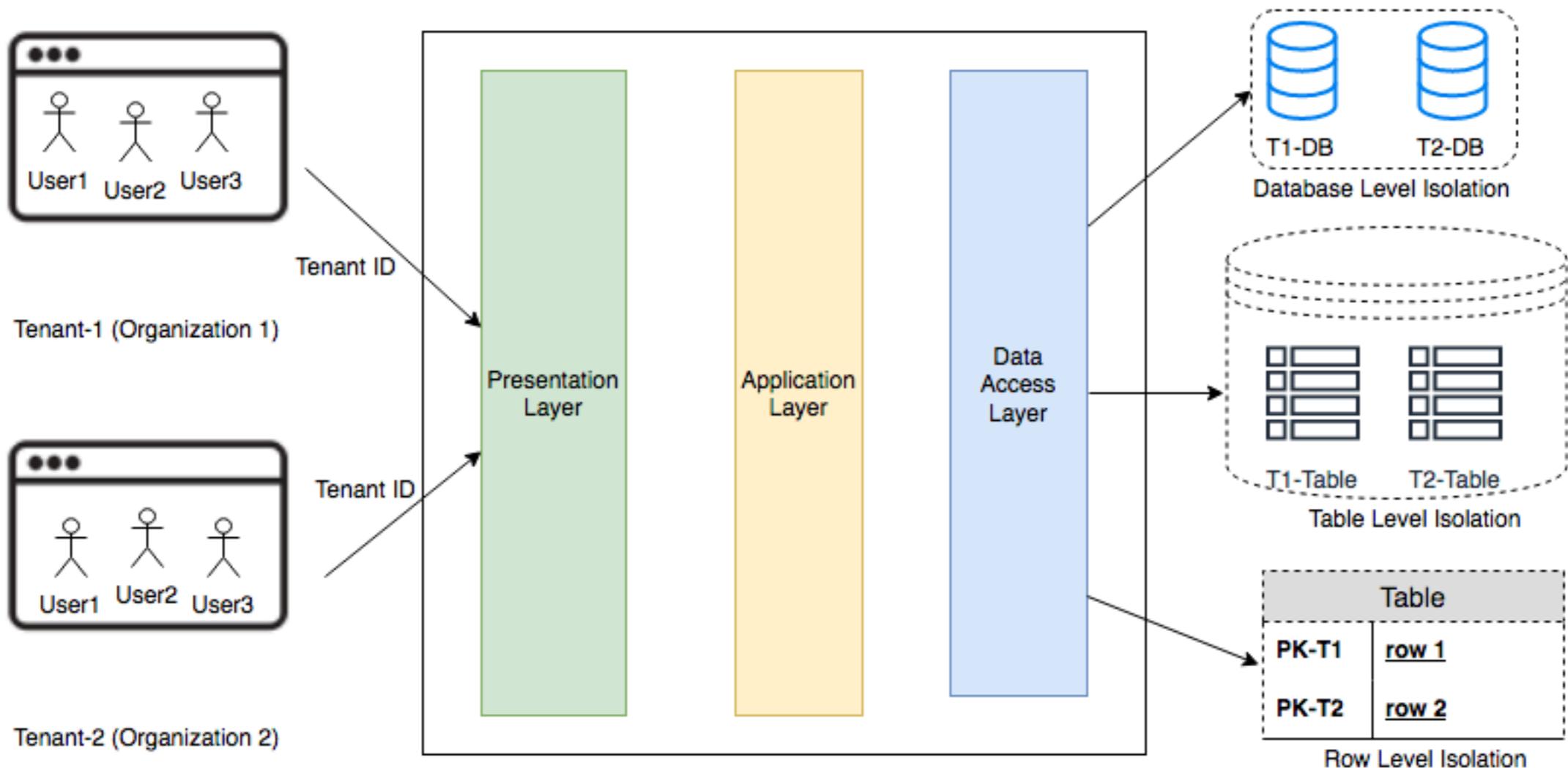
Frontend

Backend

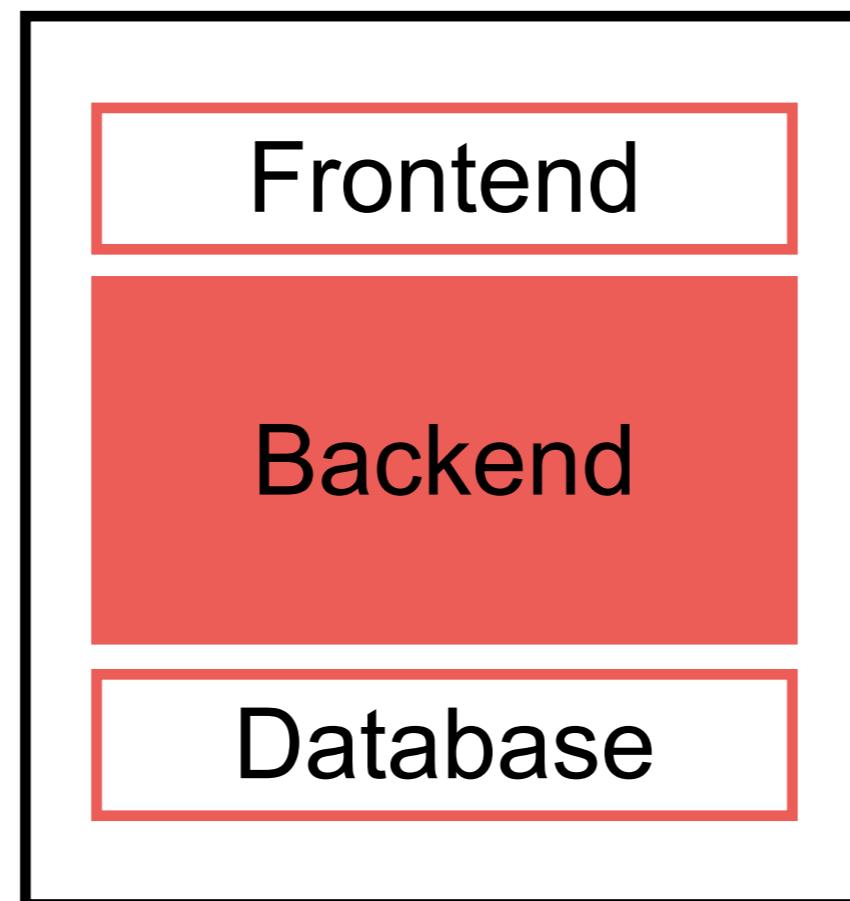
Database



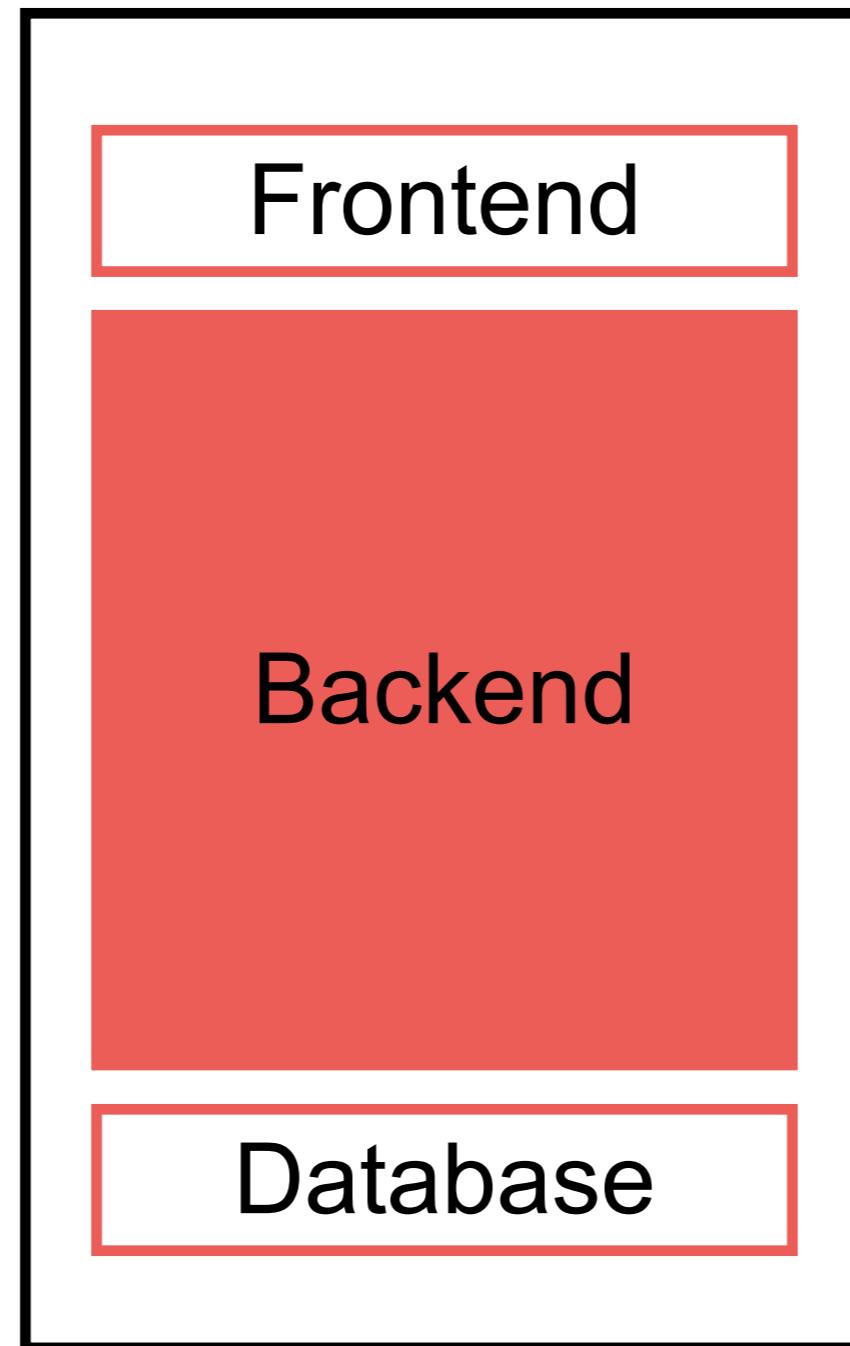
Layer/Tier



More features ...



More features ...



More features ...

Frontend

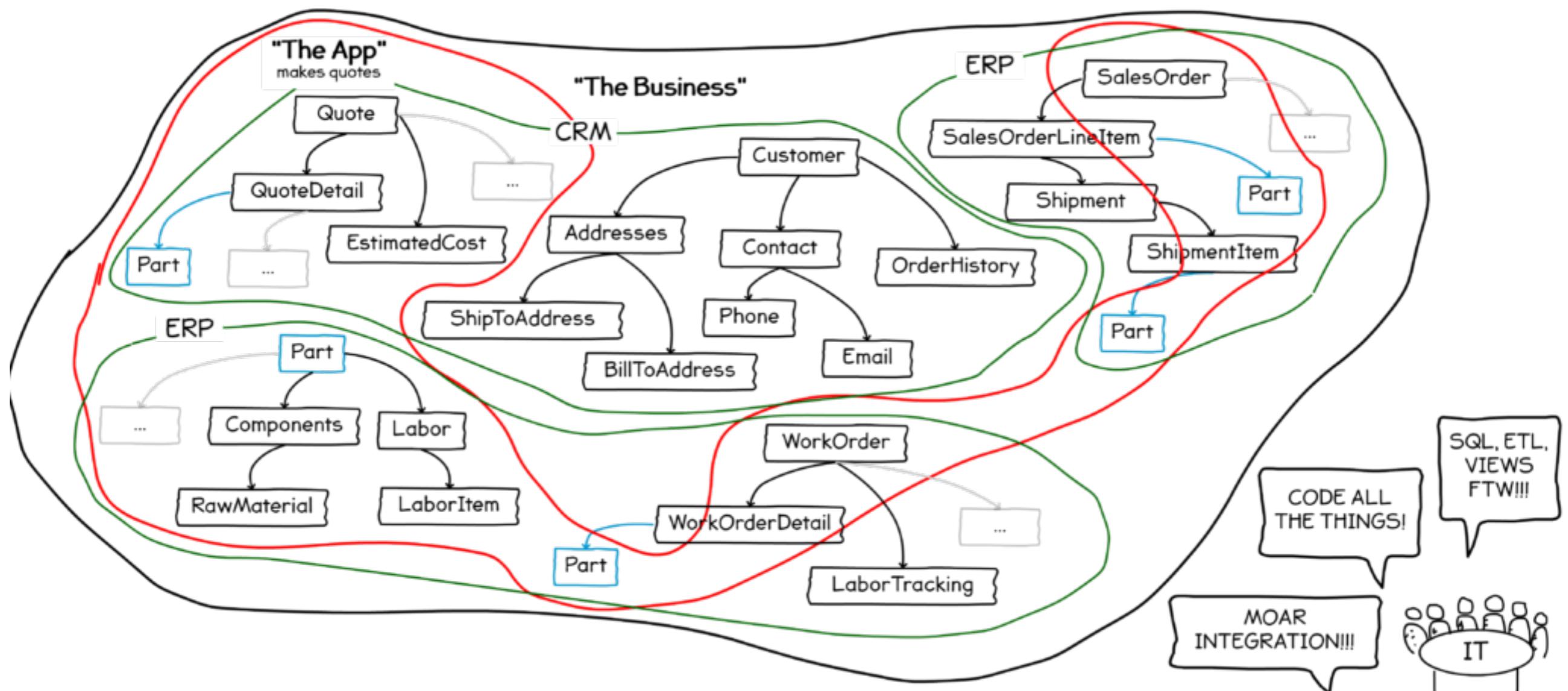


In spaghetti code, the relations between the pieces of code are so tangled that it is nearly impossible to add or change something without unpredictably breaking something somewhere else.

Database



Monolith Hell

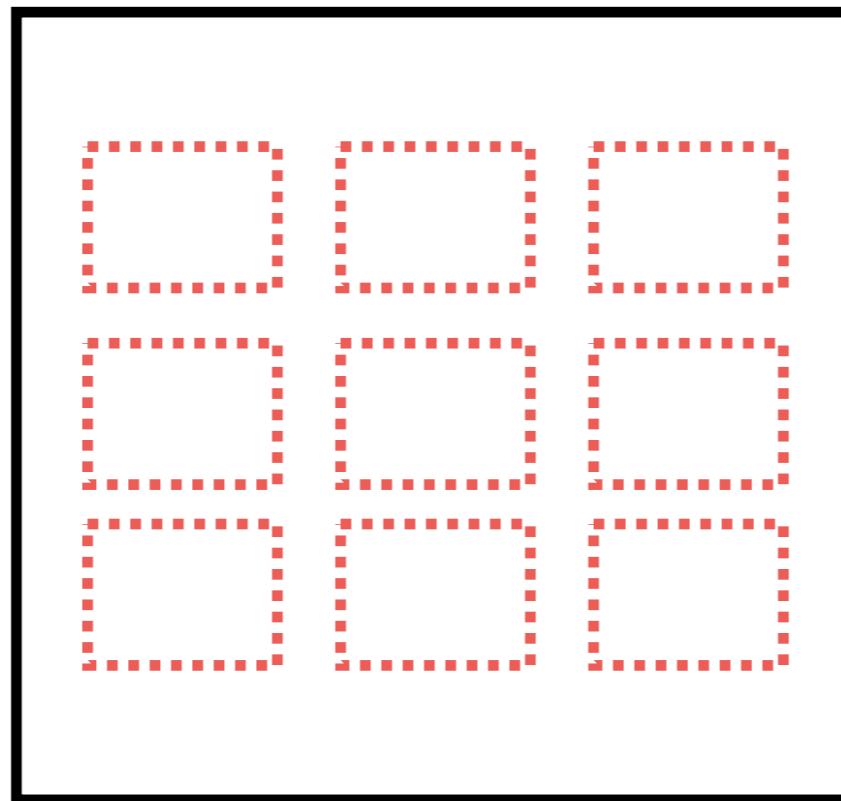


We need to improve

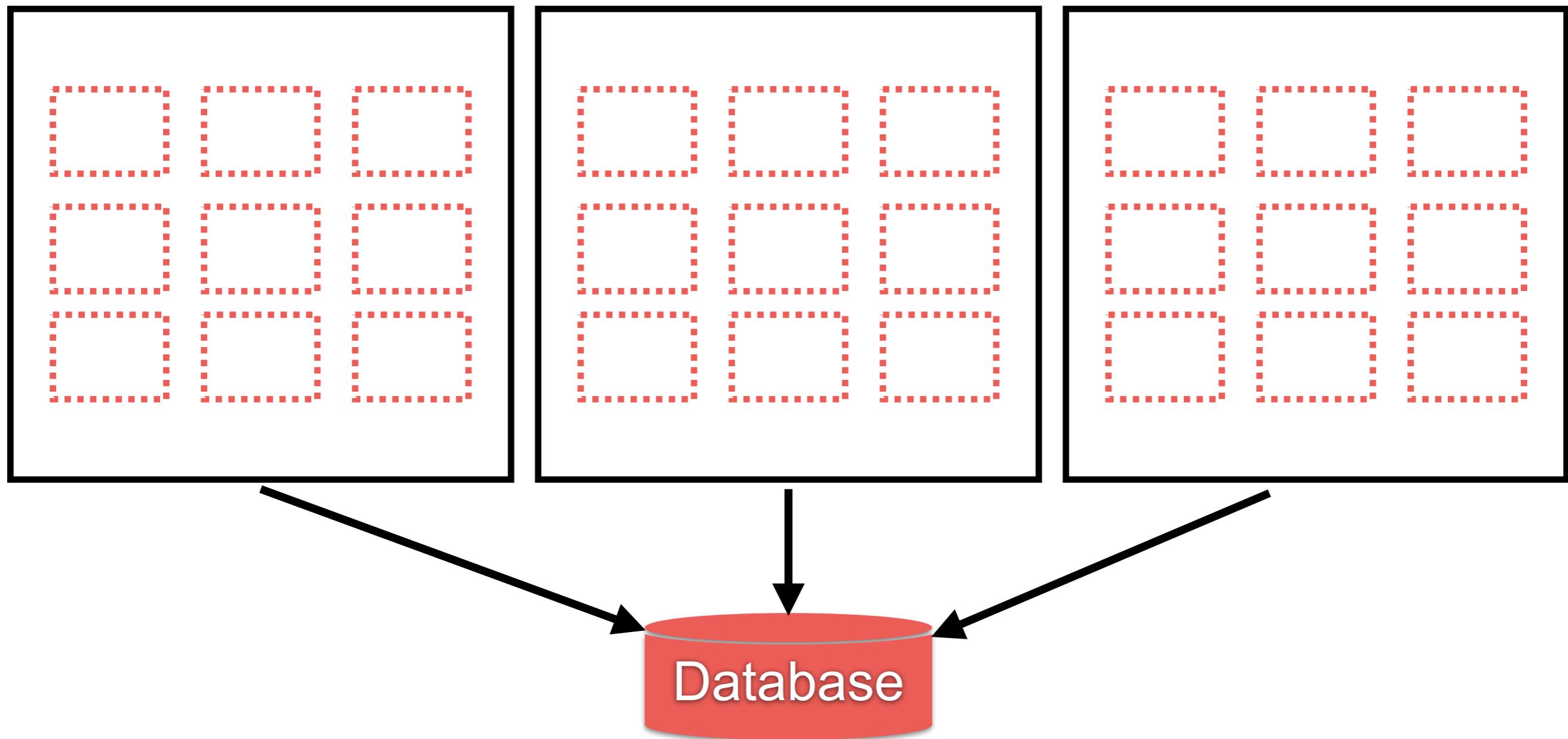
We need to get better



Modules



How to scale services ?



What happens when we need more servers ?



What if we don't use all modules equally ?



How can we update individual models/database ?



**Do all modules need to use the
same Database, Language and
Runtime ... ?**



We need to improve

We need to get better



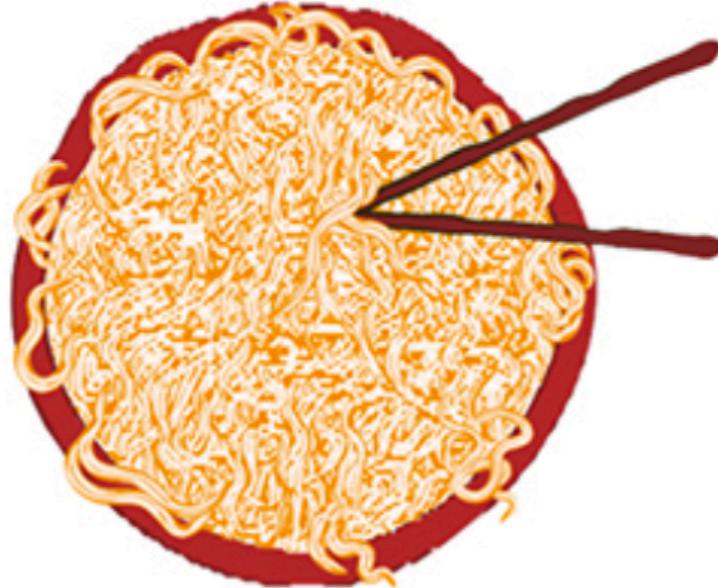
Service-Oriented Architecture



SOA

1990s and earlier

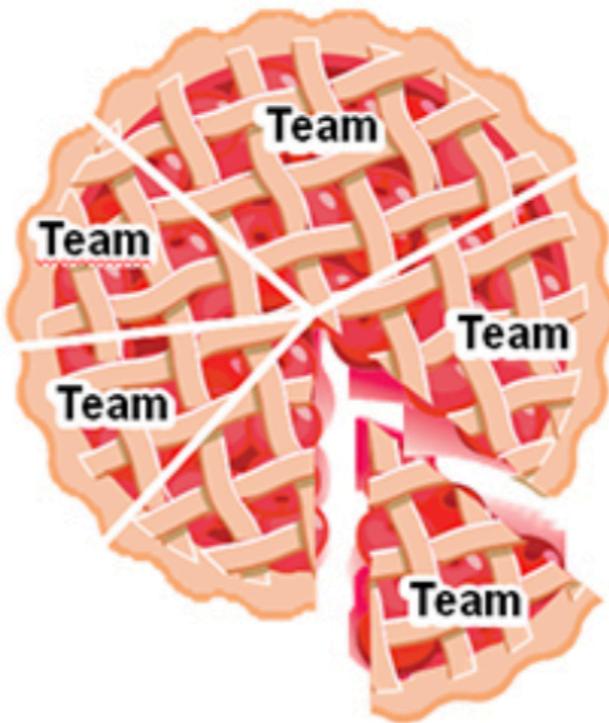
Pre-SOA (monolithic)
Tight coupling



For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

2000s

Traditional SOA
Looser coupling



Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

2010s

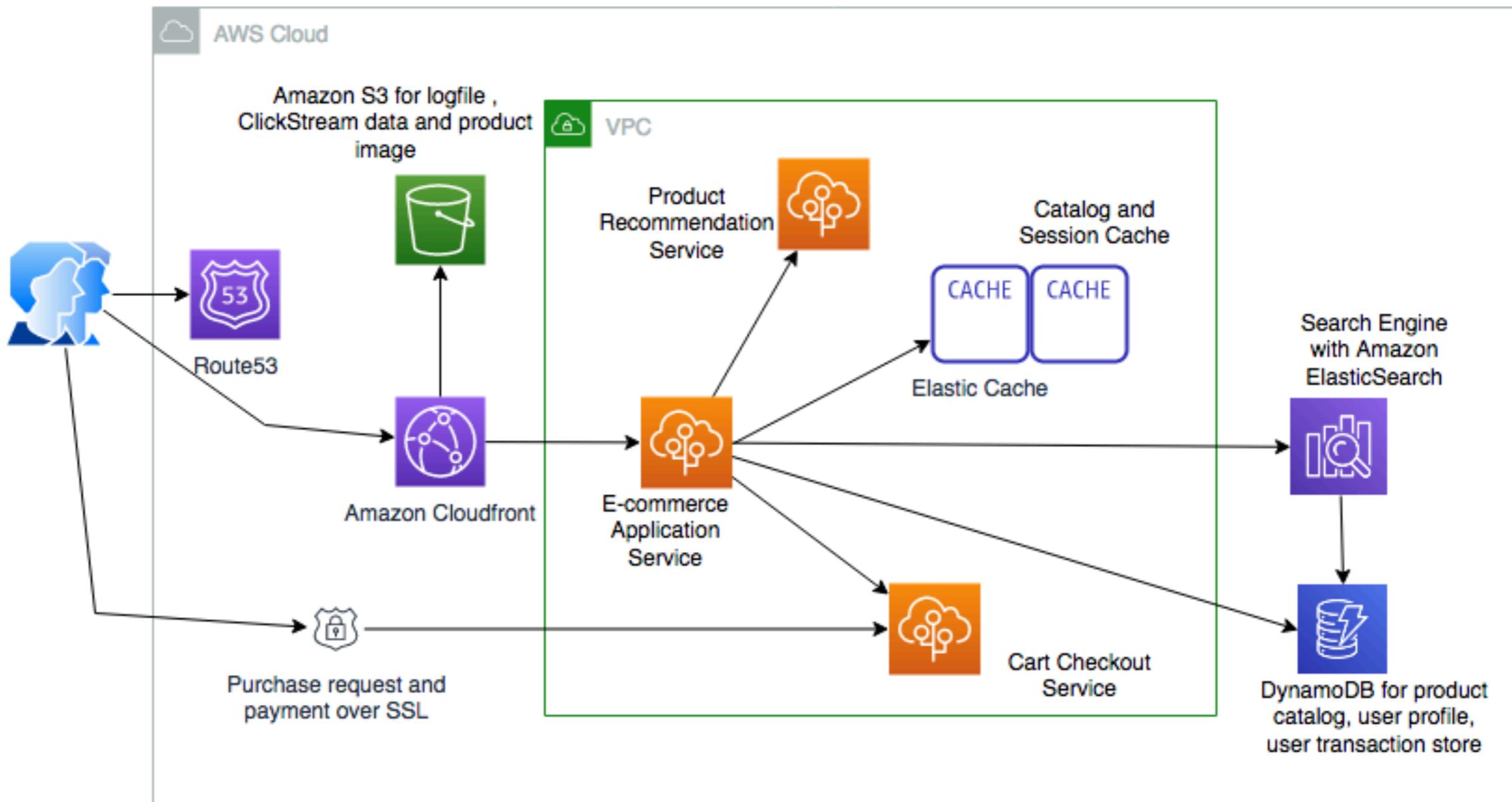
Microservices
Decoupled



Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.



SOA



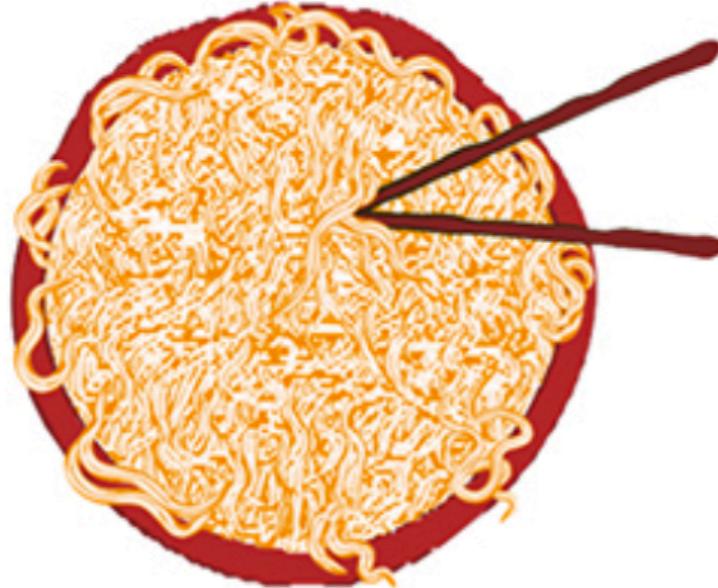
Microservices



Microservices

1990s and earlier

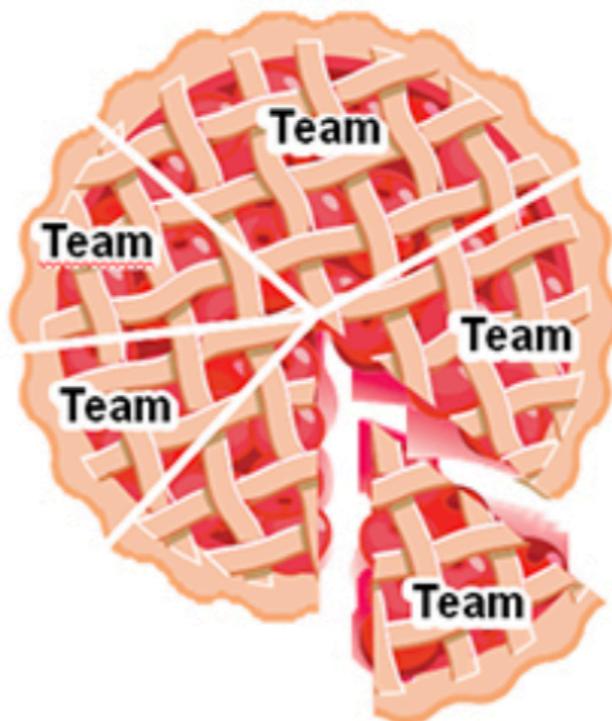
Pre-SOA (monolithic)
Tight coupling



For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

2000s

Traditional SOA
Looser coupling



Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

2010s

Microservices
Decoupled



Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.



What is service ?

Standalone and loosely couple

Independently deployable software component

Implement some useful functionality

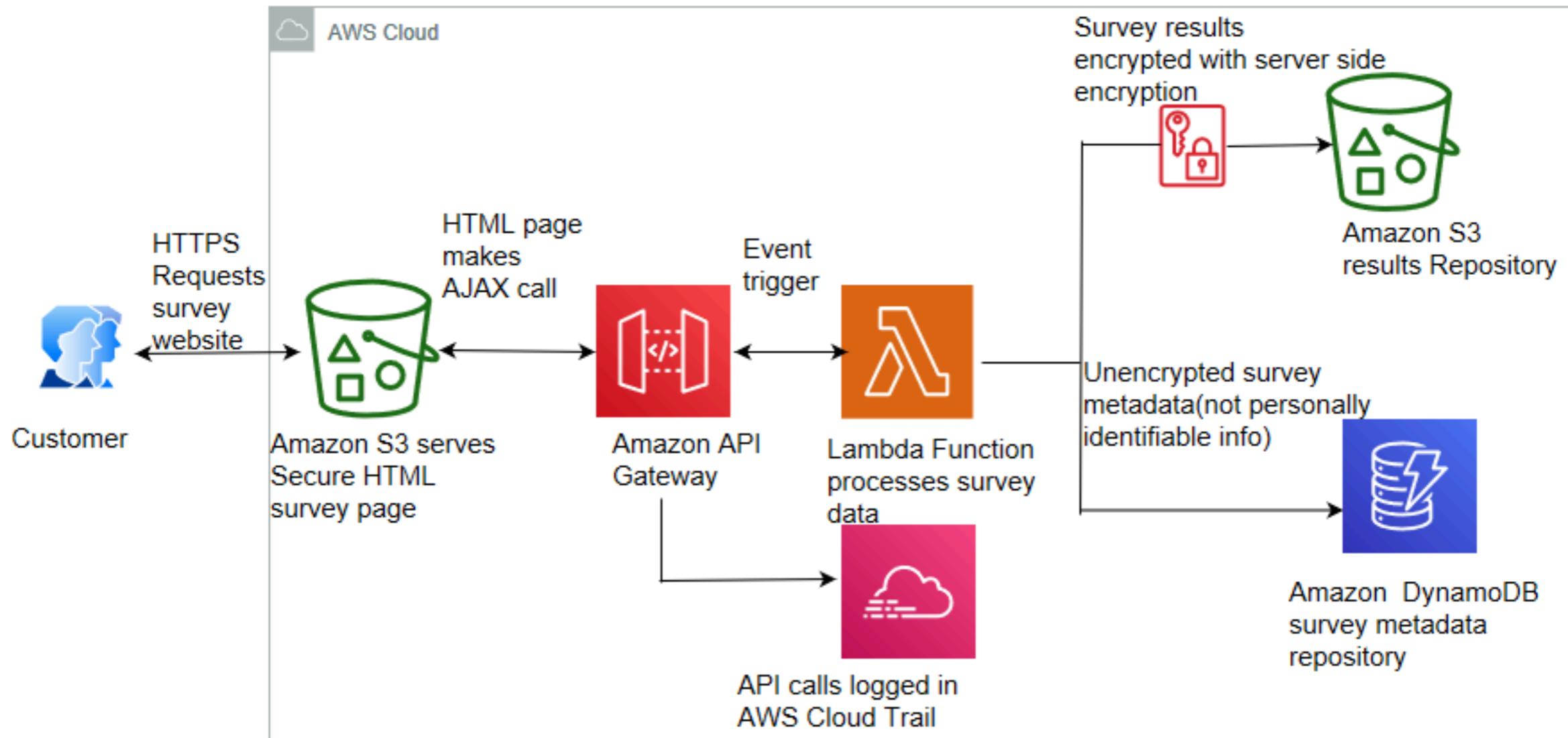


Serverless

Function as a Service



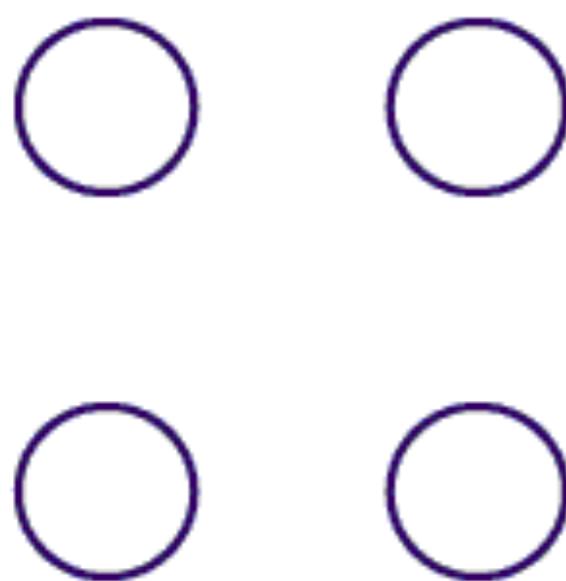
Serverless



Think about Loosely couple architecture

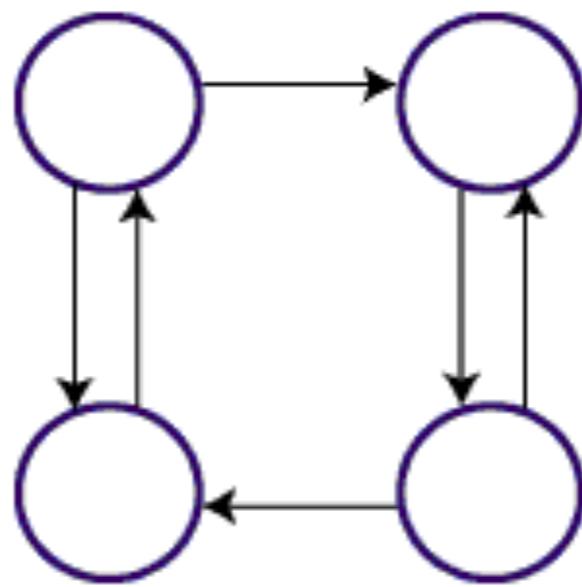


Types of coupling



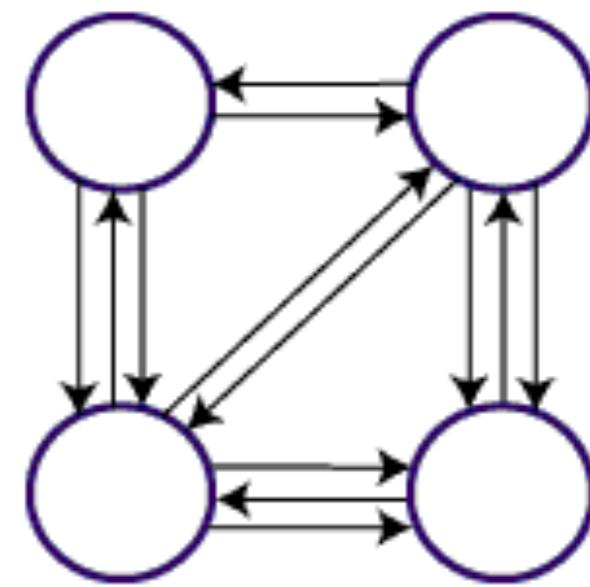
Uncoupled: no
dependencies

(a)



Loosely Coupled:
Some dependencies

(b)

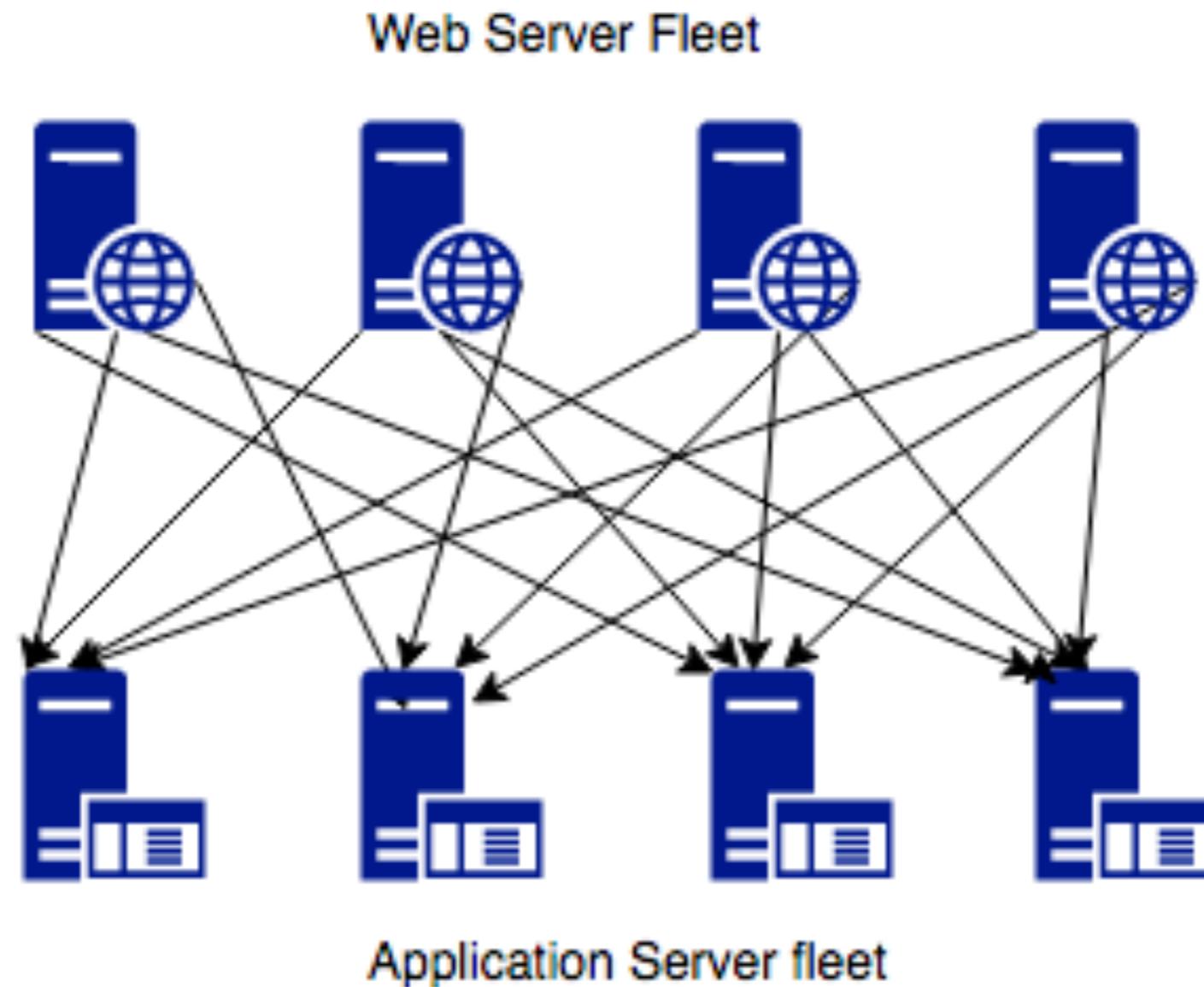


Highly Coupled:
Many dependencies

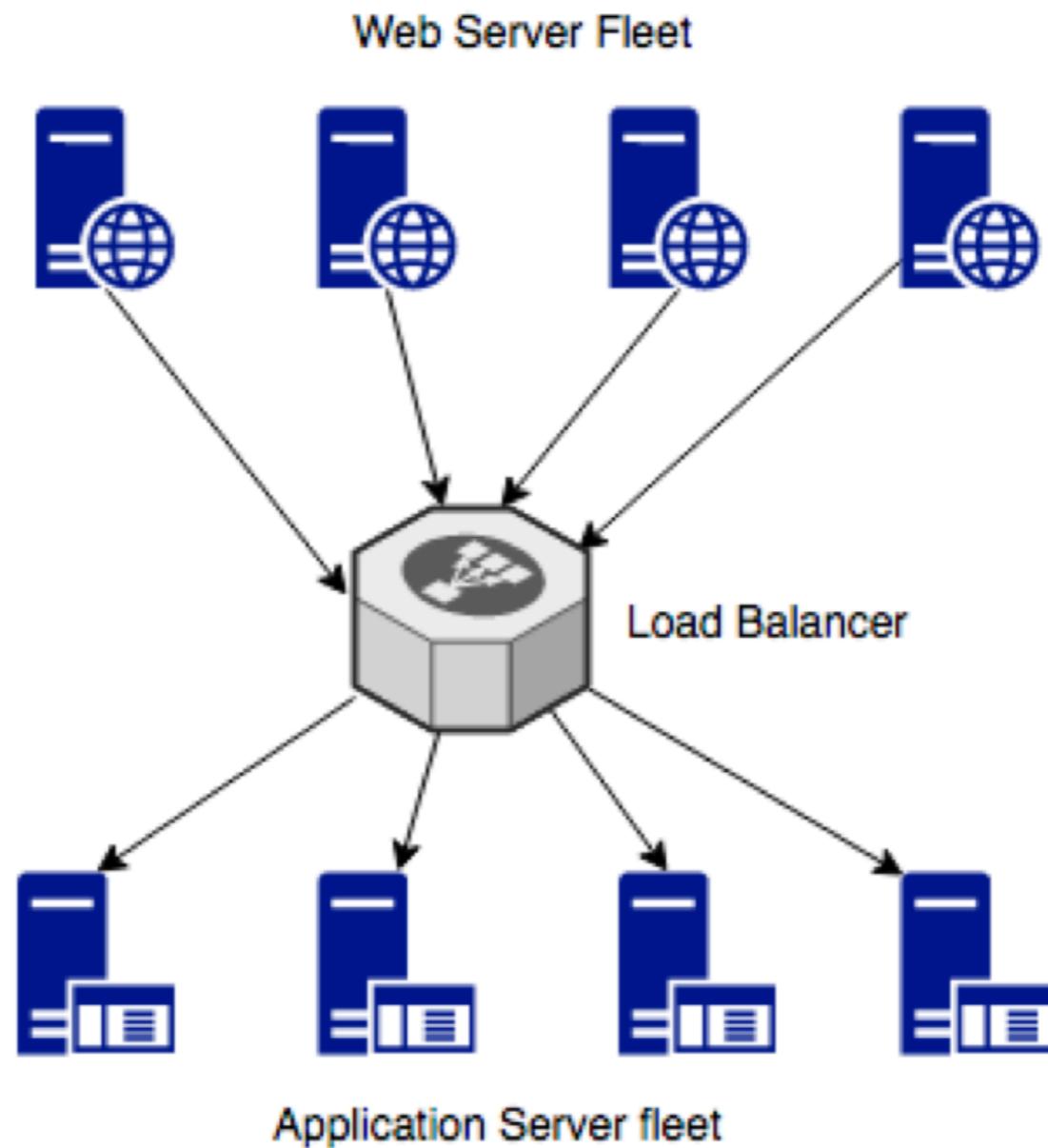
(c)



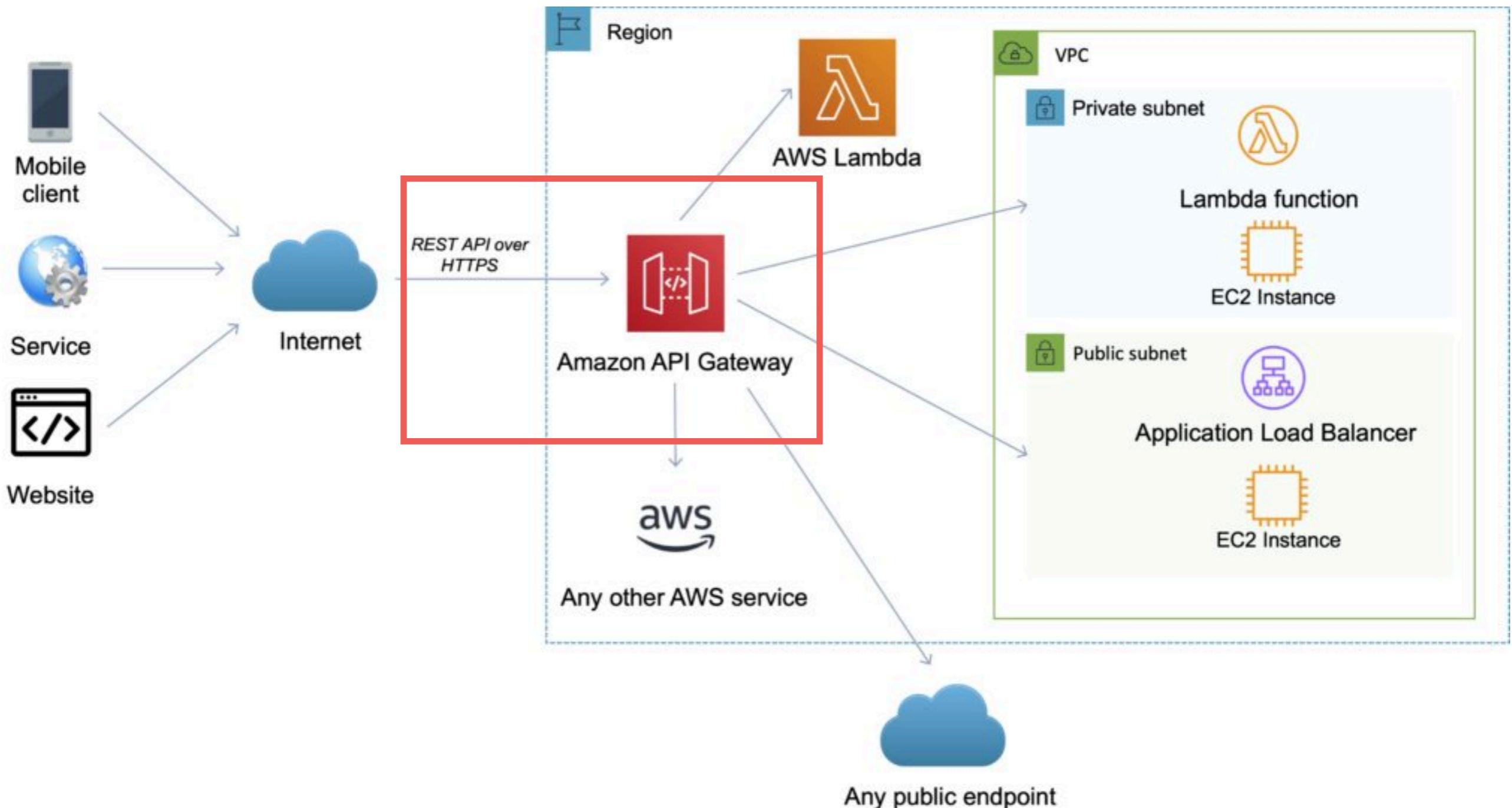
Tight coupling



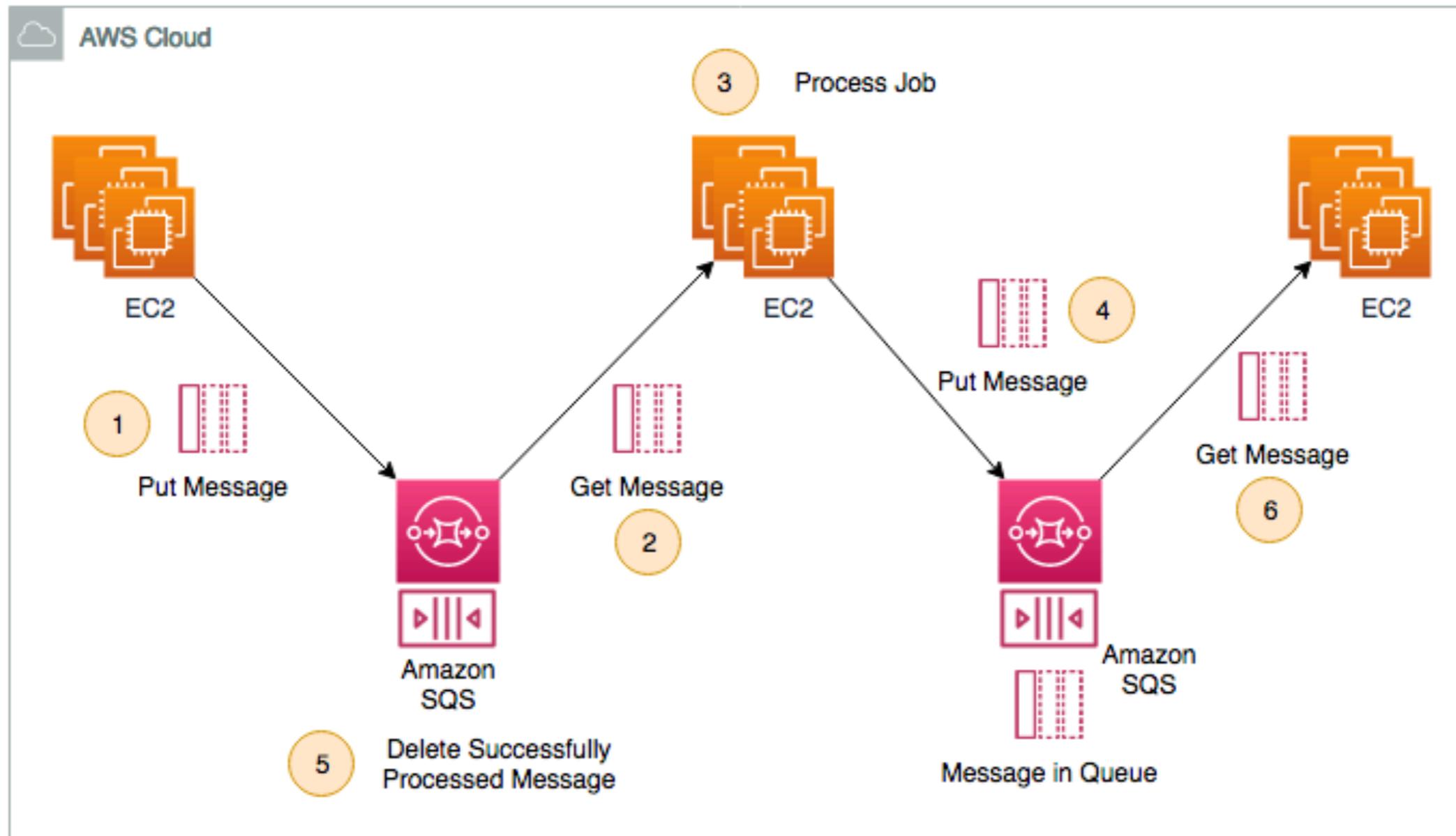
Load balance



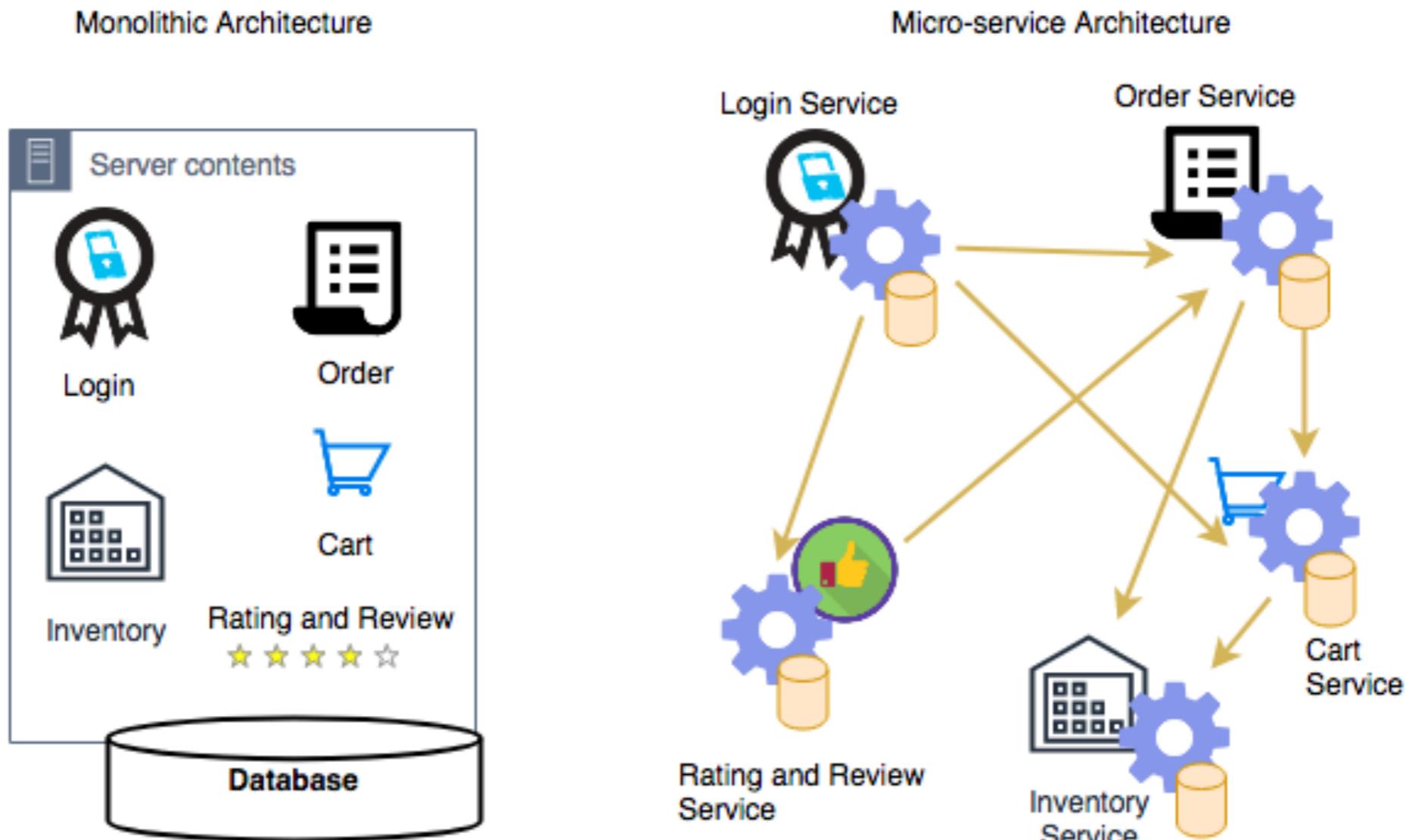
API gateway



Messaging :: Queue



Microservices



Caching ?

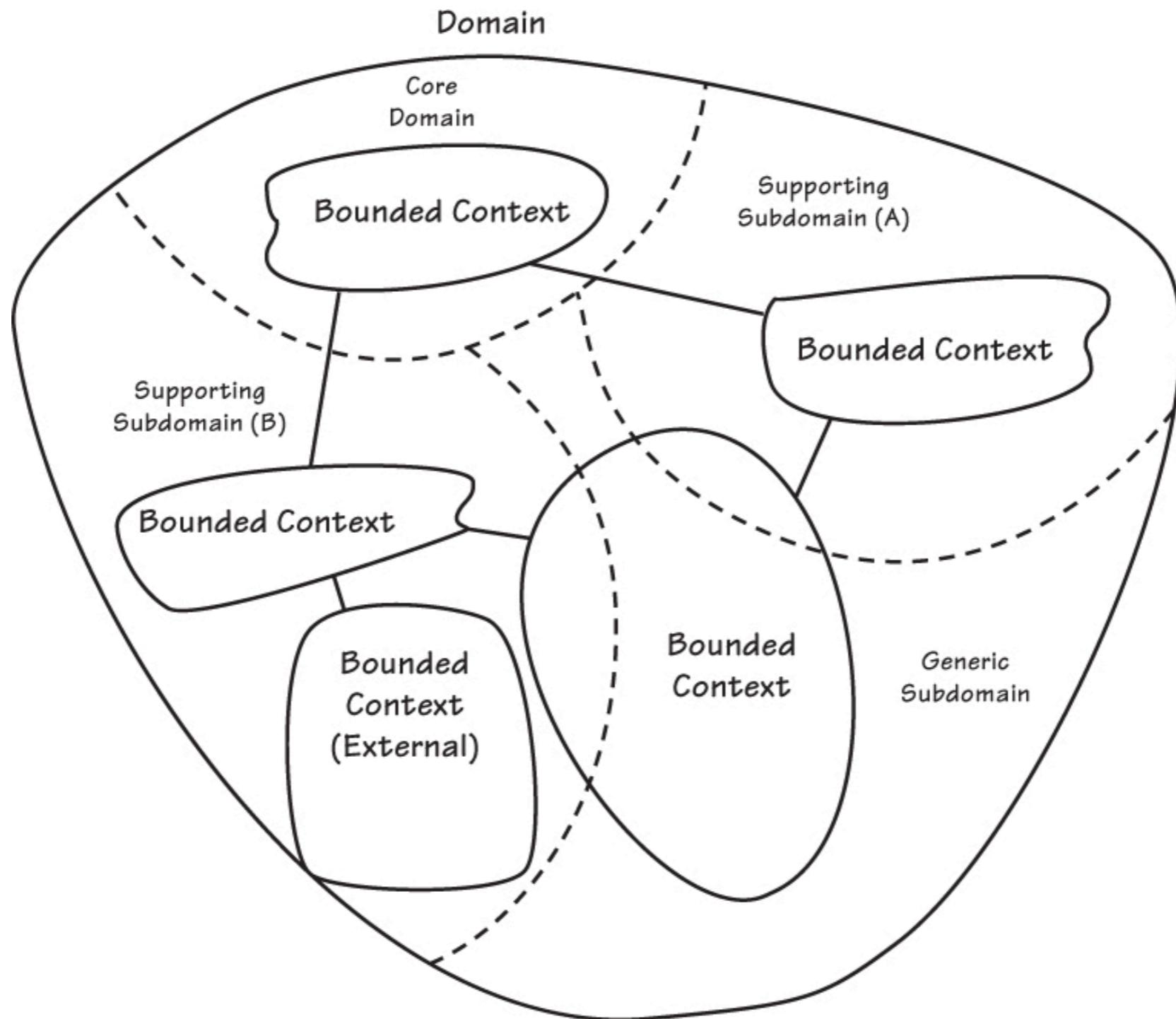
Solution Architecture Layers



Strategic Design



Subdomain in DDD



Domain in DDD

Core domain

Supporting subdomain

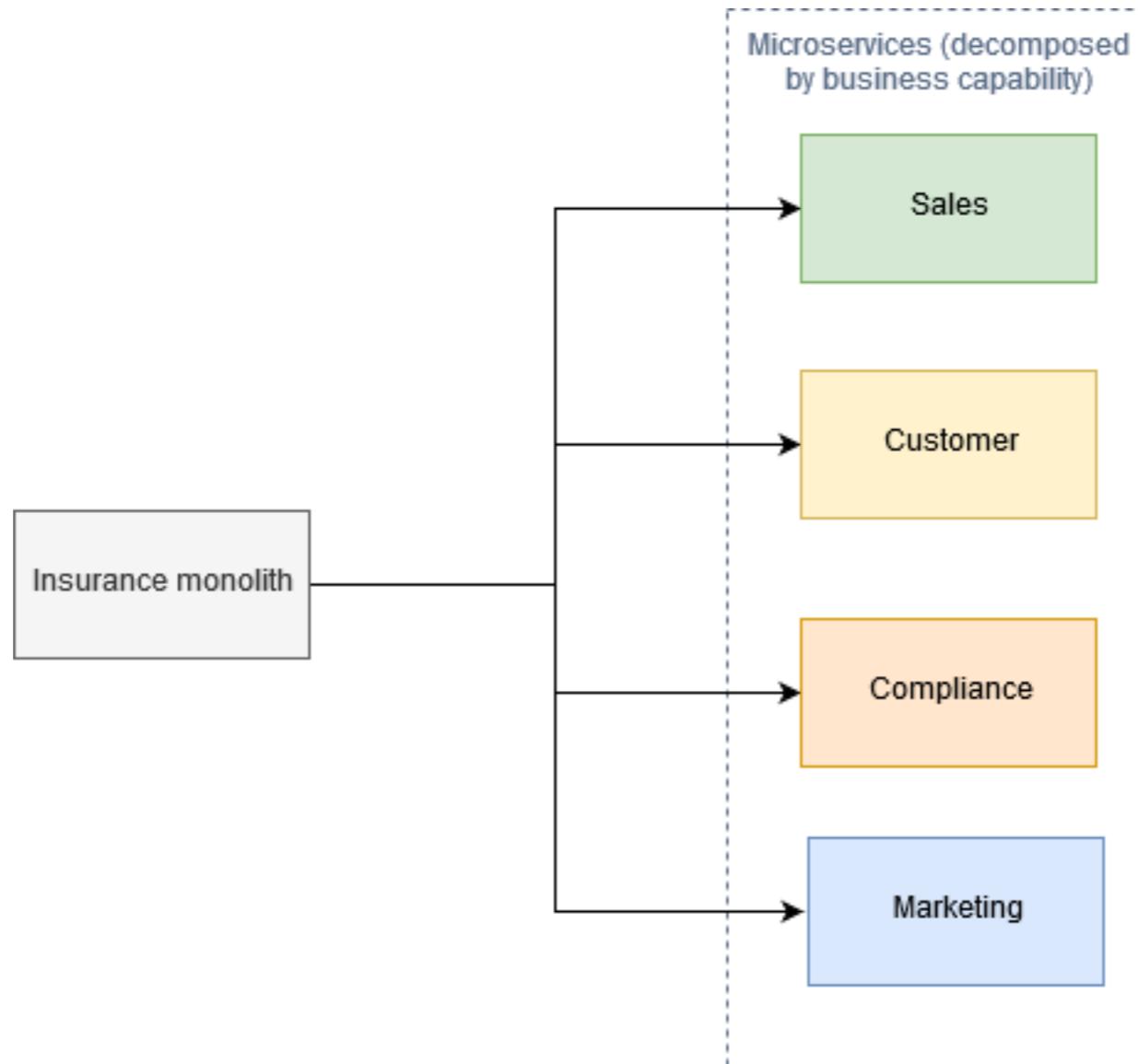
Generic subdomain



Decoupling from Monolith



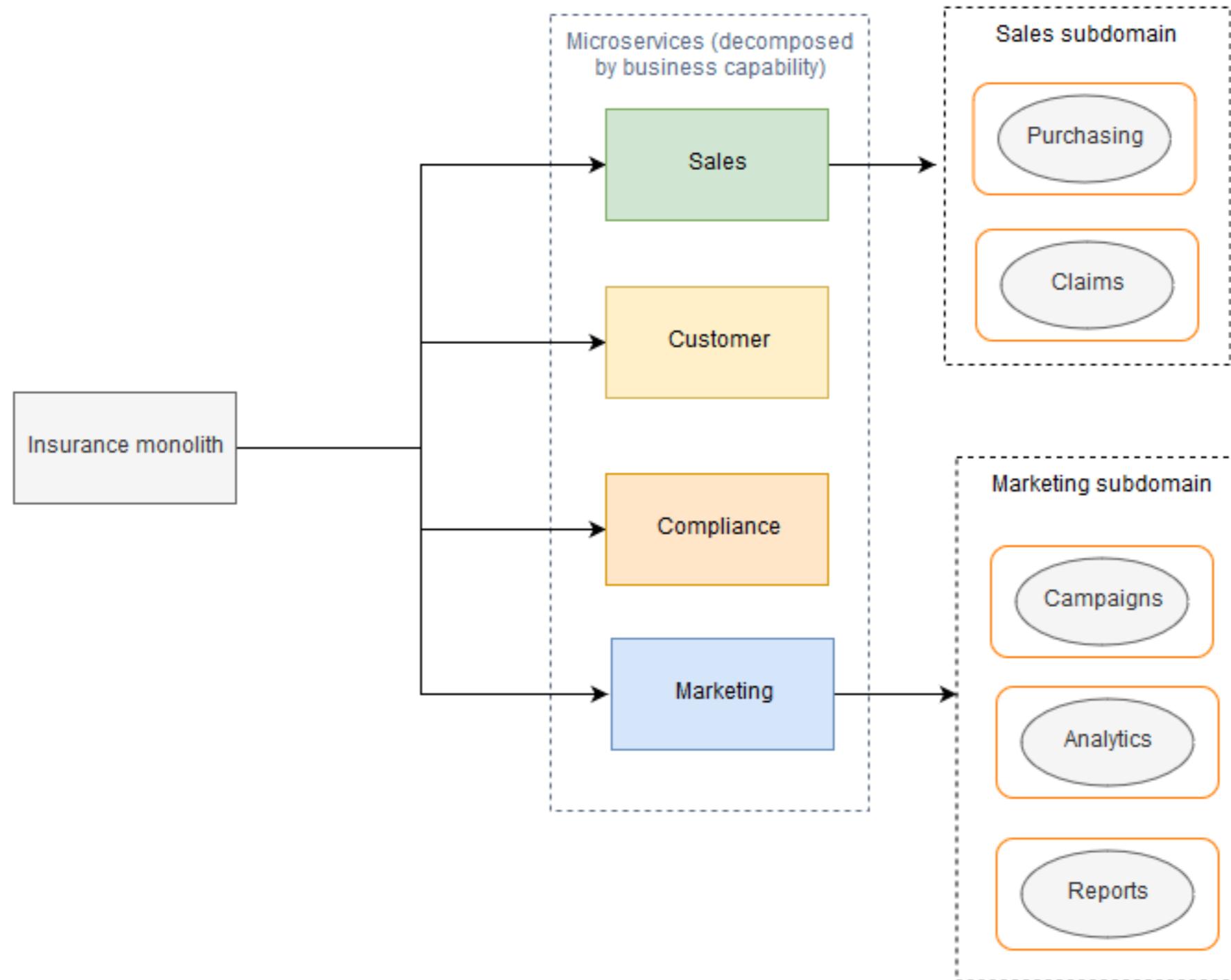
Decompose by Business



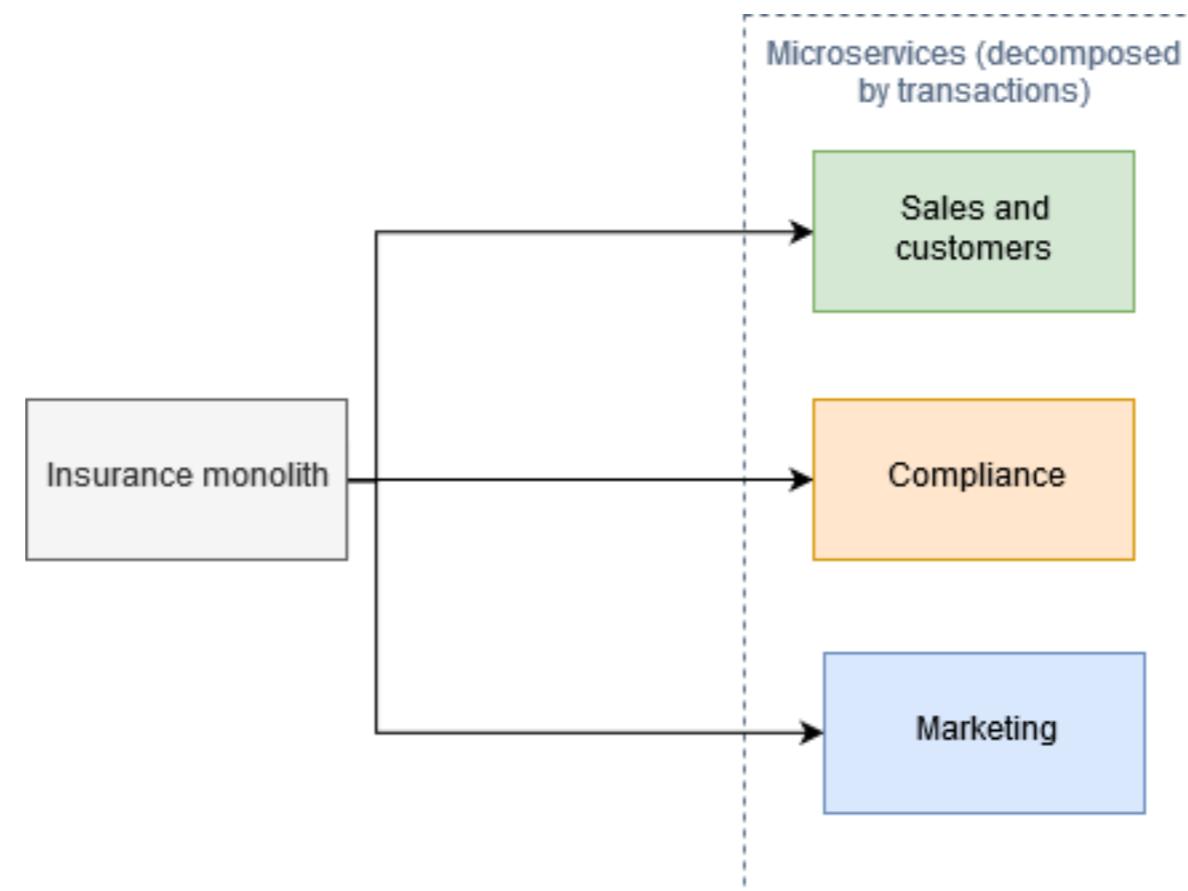
<https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-decomposing-monoliths/decomposing-patterns.html>



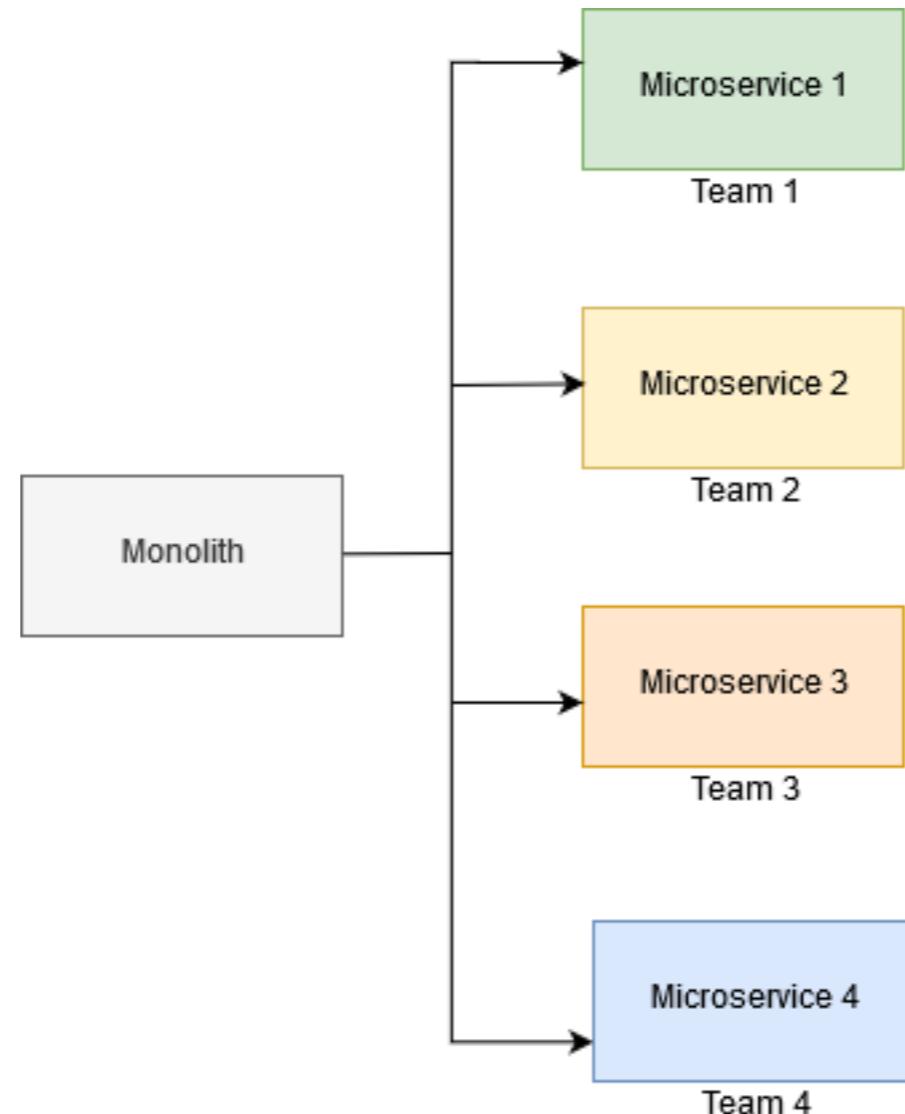
Decompose by Subdomain



Decompose by Transaction



Decompose by Team pattern



Solution Architecture in Agile World



**Respond to change faster
Need more flexible
Working with stakeholder**





VS



Architecture Scaling

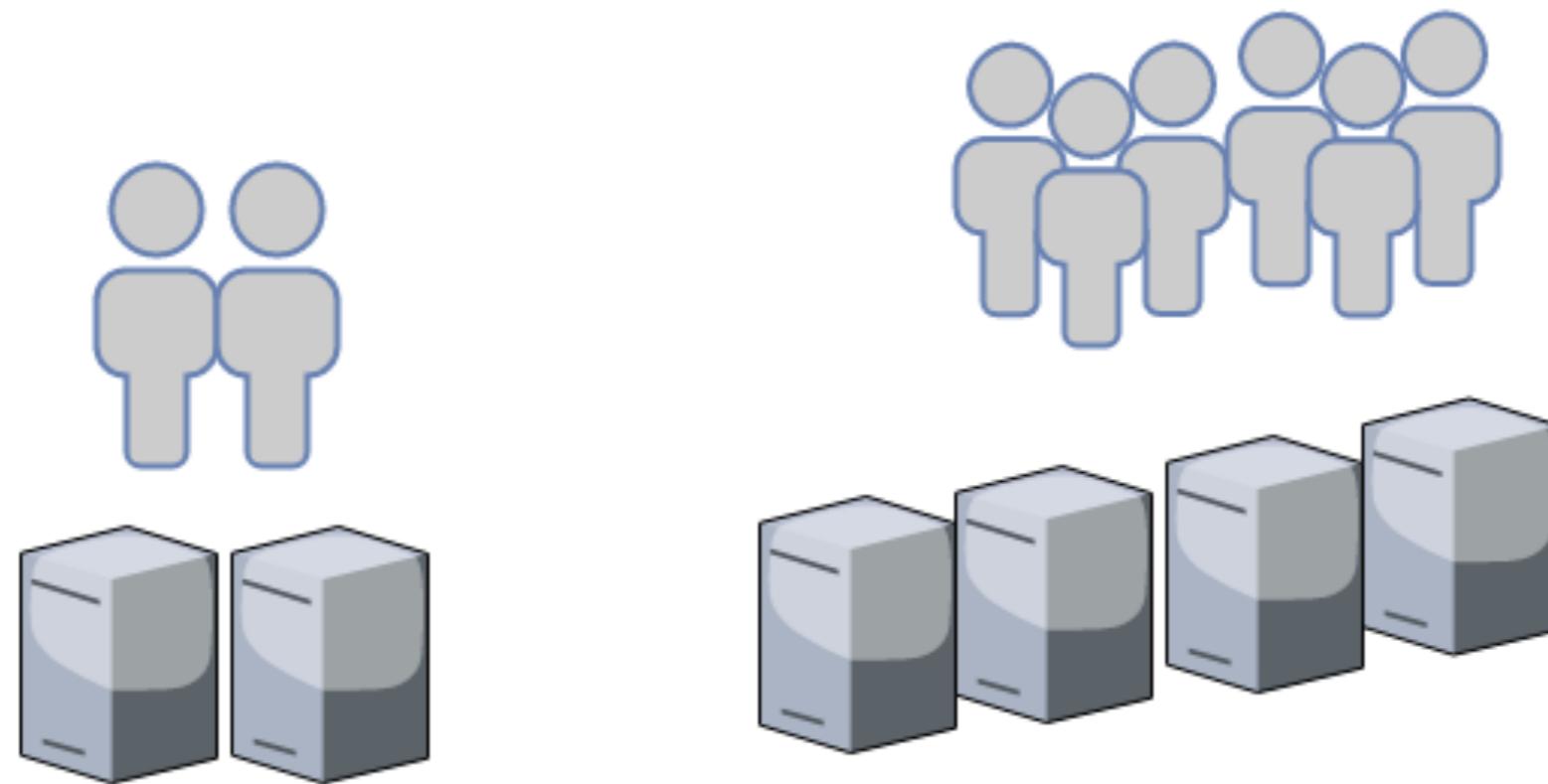


Architecture Scaling

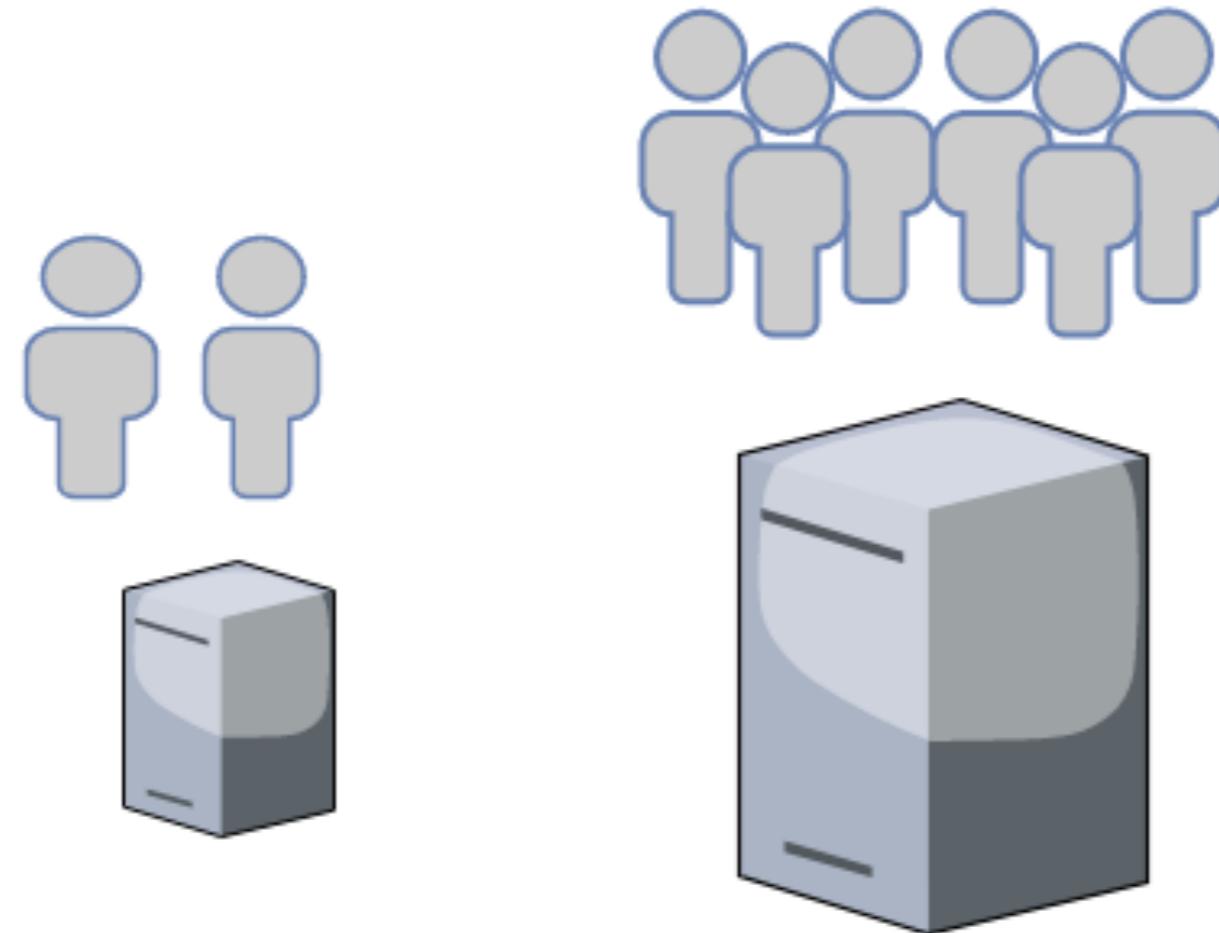
Horizontal
Vertical
Elastic



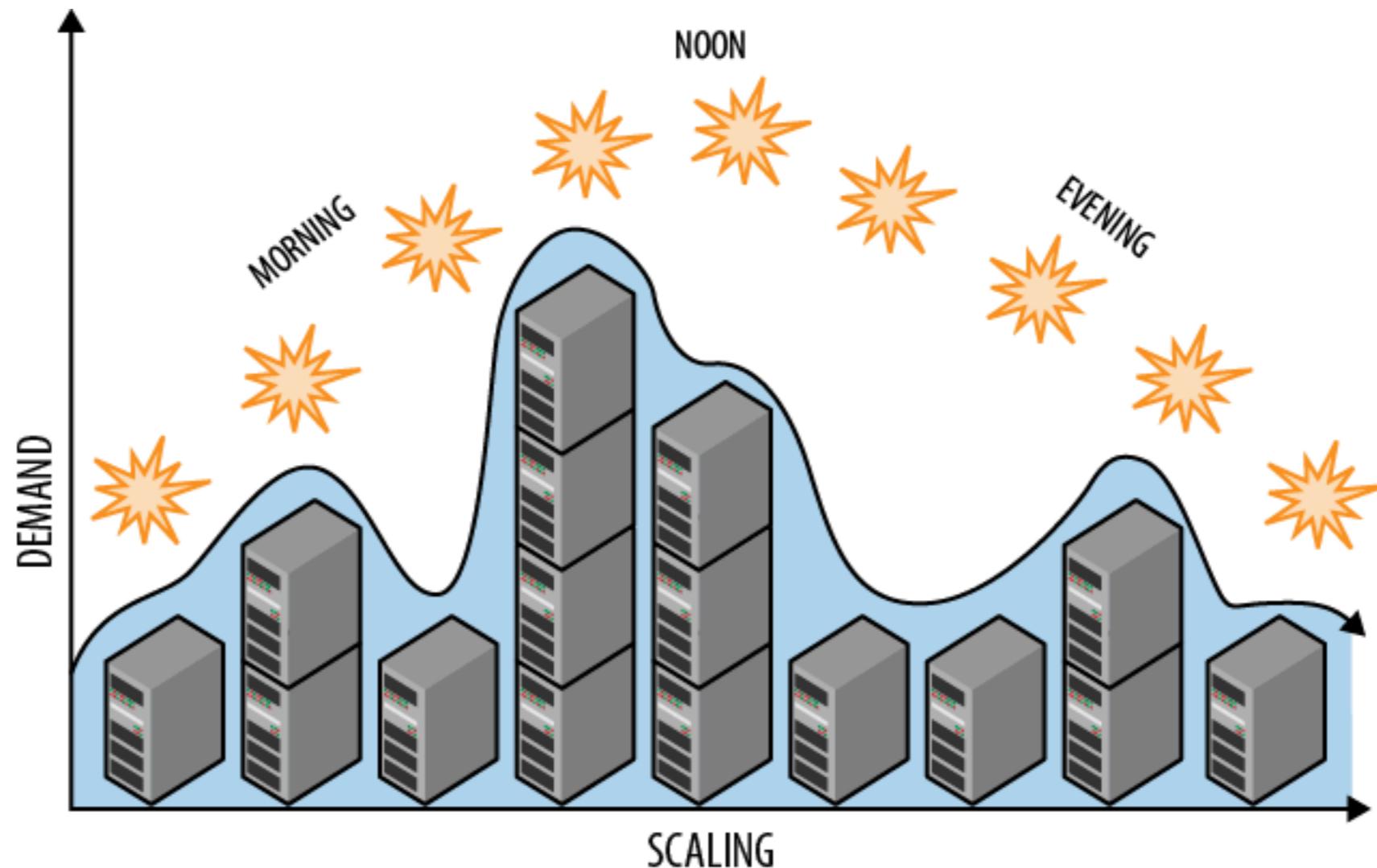
Horizontal Scaling



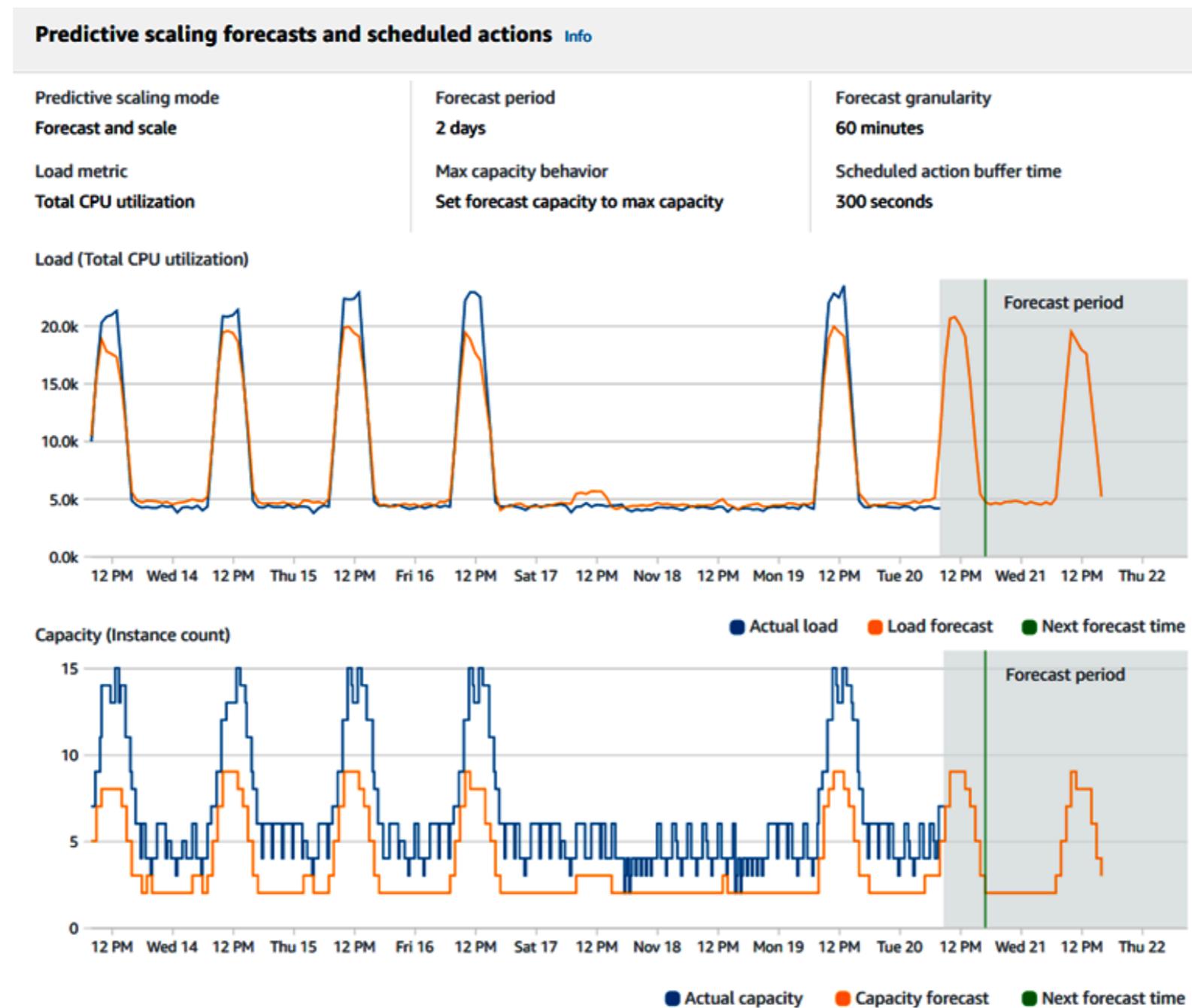
Vertical Scaling



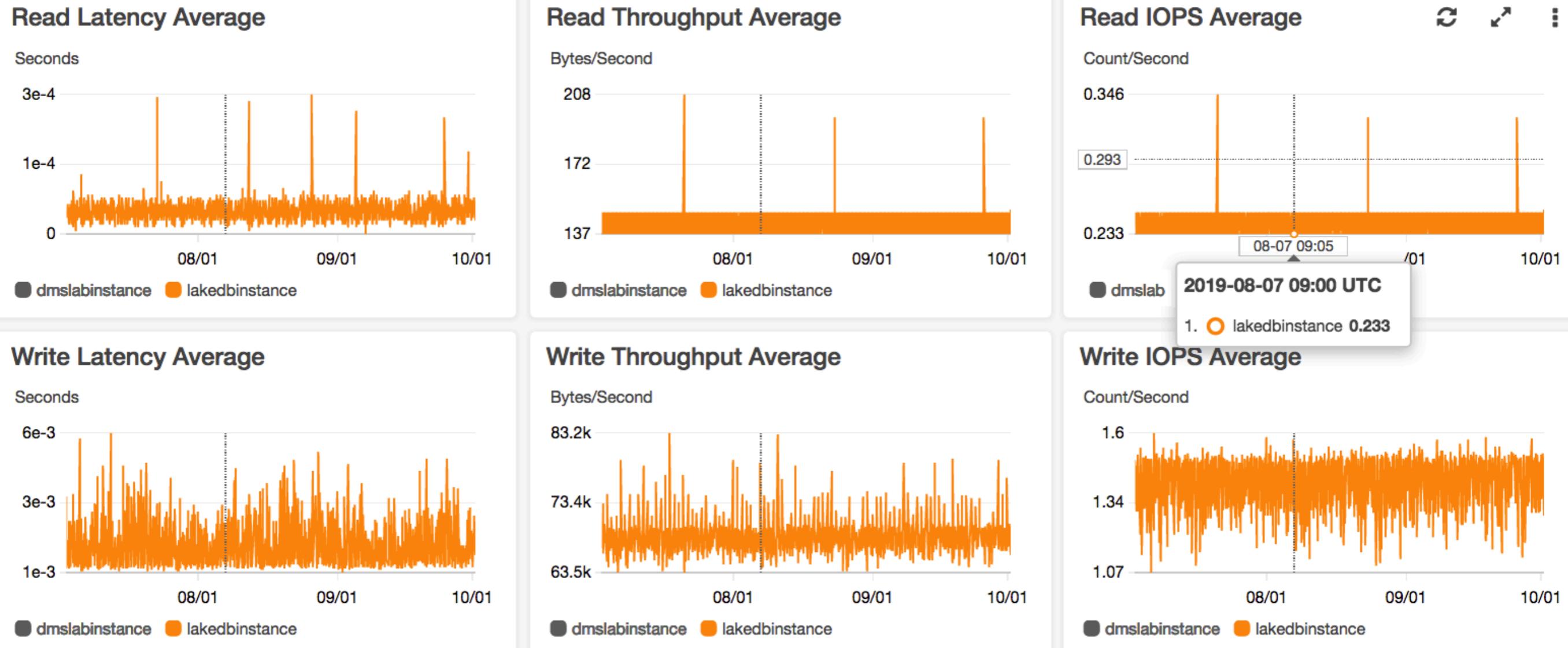
Elastic Scaling



Predictive Scaling ?



Predictive Scaling ?



Collect your data ?



Predictive Scaling from Patterns

Weekend 5X than weekday ?

Daytime 10X that night time ?

Shopping season

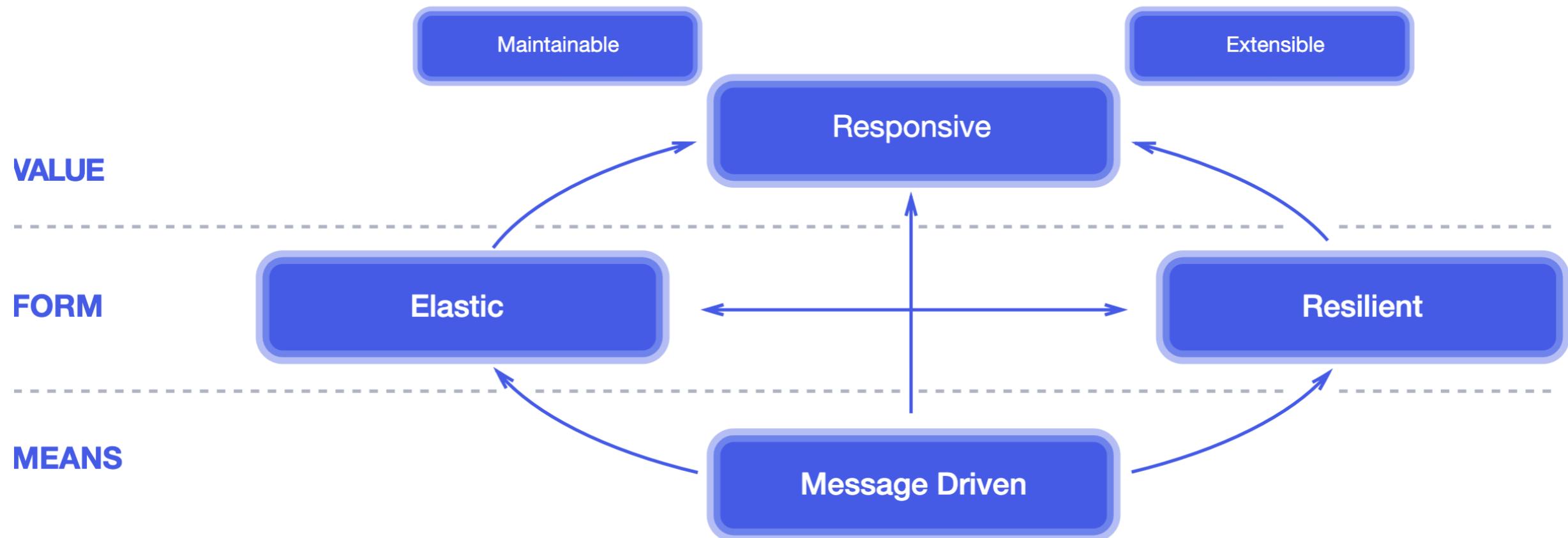
Holiday

End of month

1st and 15th in every month



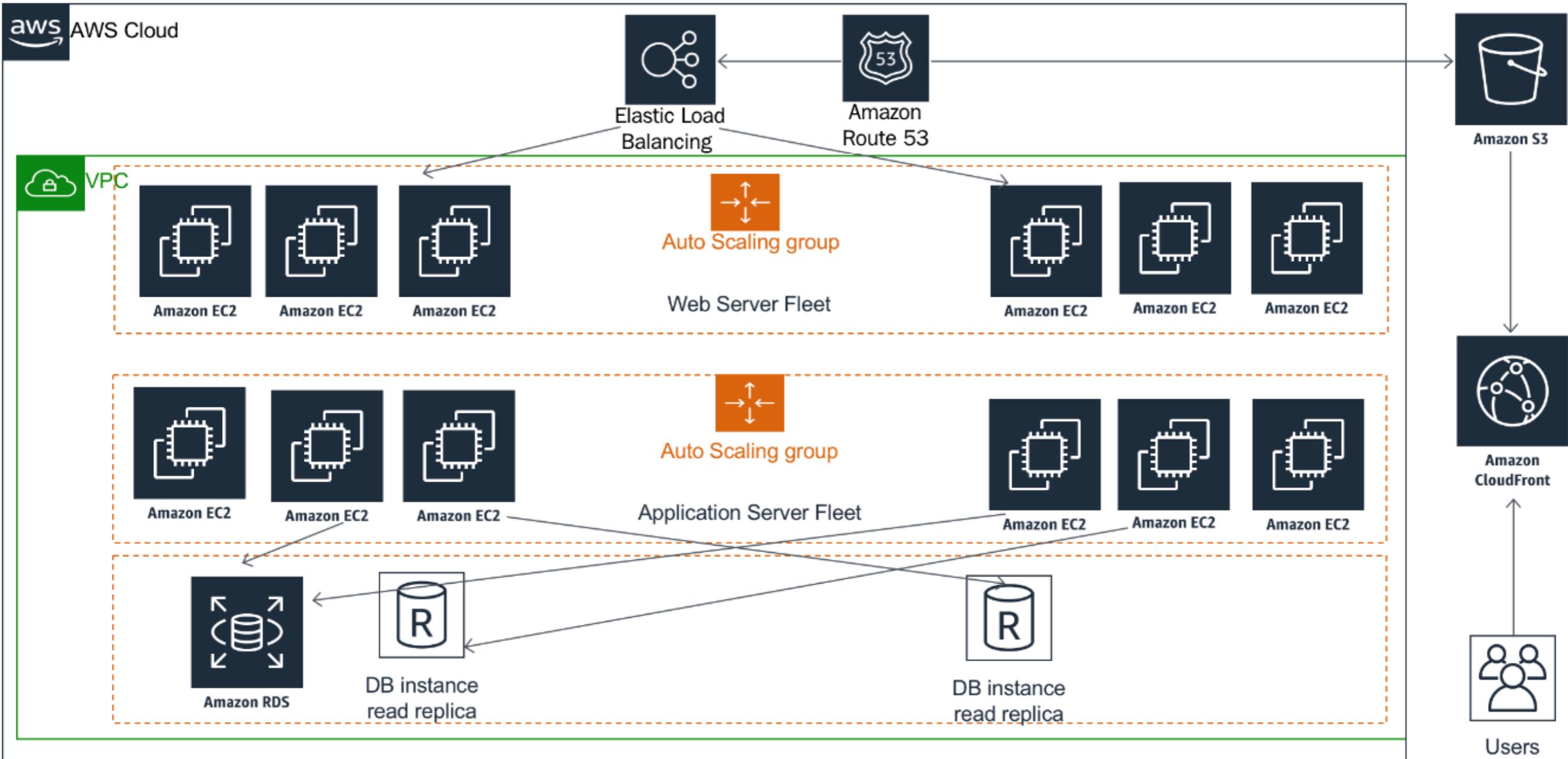
Reactive Architecture



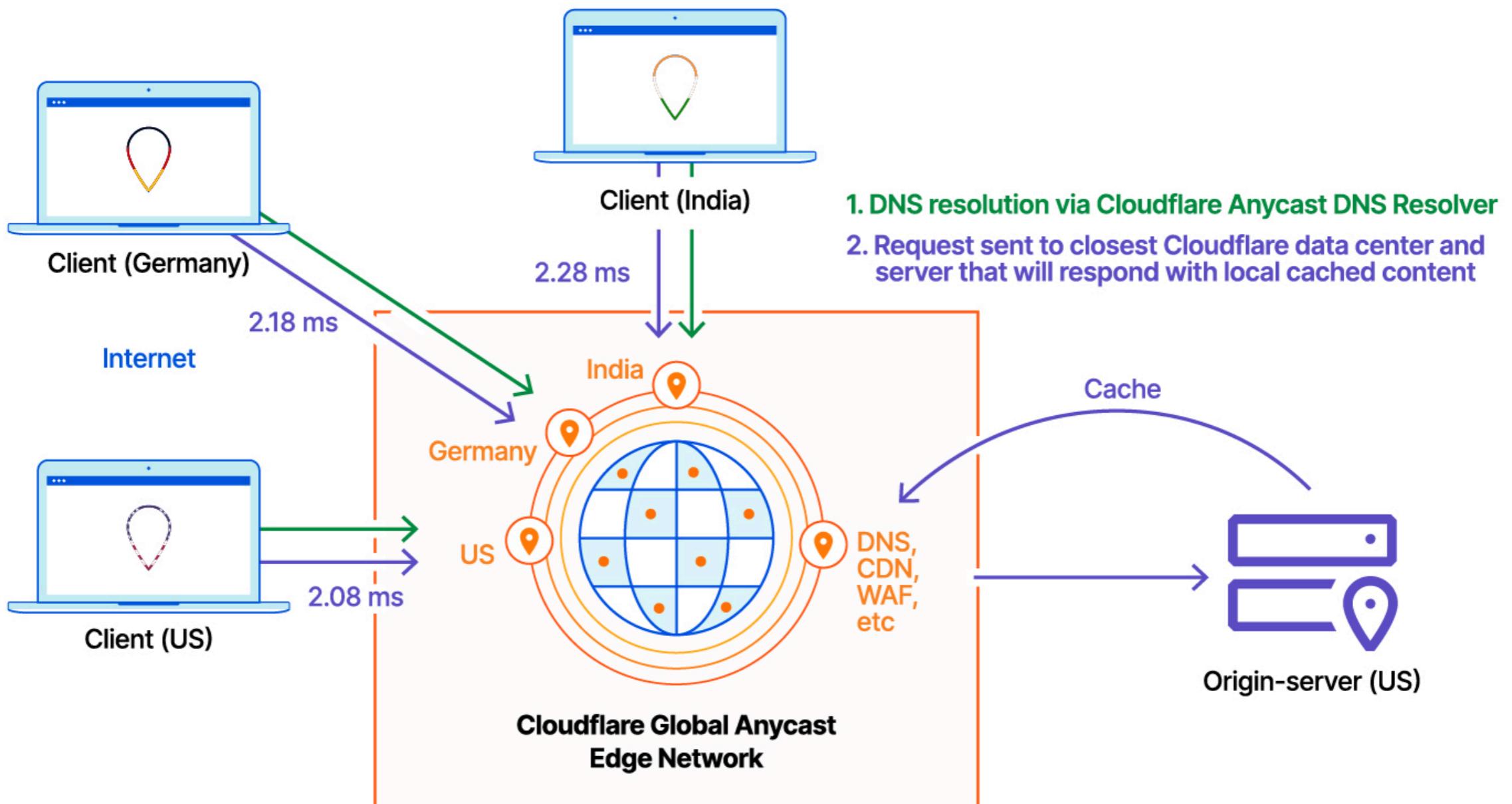
<https://www.reactivemanifesto.org/>



Example :: Solution



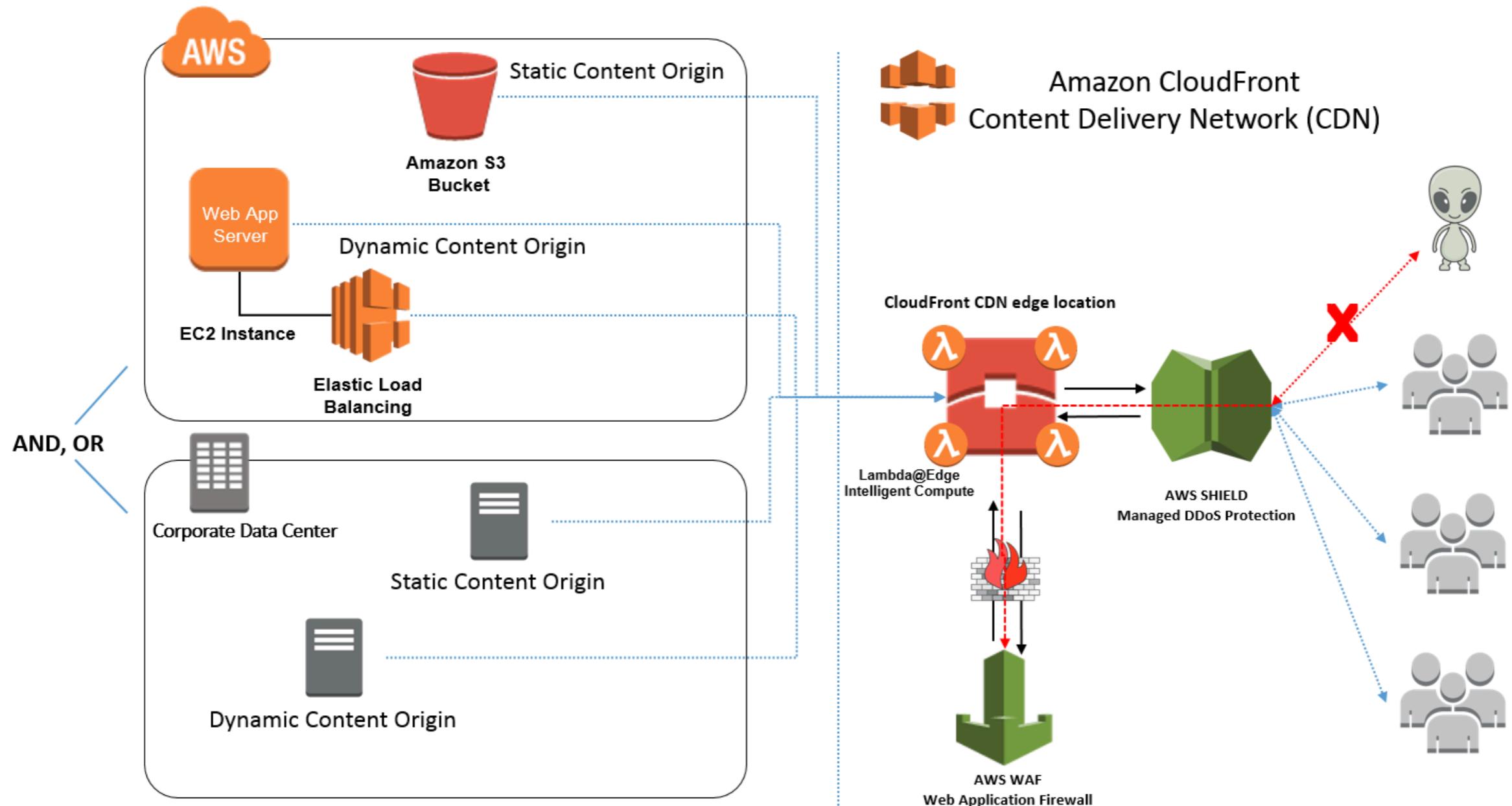
CDN with Cloudflare



<https://developers.cloudflare.com/reference-architecture/cdn-reference-architecture/>



CDN with AWS



<https://aws.amazon.com/caching/cdn/>



Database Scaling ?

Horizontal
Vertical
Sharding
Working with NoSQL



Database model ?

410 systems in ranking, March 2023

Rank			DBMS	Database Model	Score		
Mar 2023	Feb 2023	Mar 2022			Mar 2023	Feb 2023	Mar 2022
1.	1.	1.	Oracle 	Relational, Multi-model 	1261.29	+13.77	+9.97
2.	2.	2.	MySQL 	Relational, Multi-model 	1182.79	-12.66	-15.45
3.	3.	3.	Microsoft SQL Server 	Relational, Multi-model 	922.01	-7.08	-11.77
4.	4.	4.	PostgreSQL 	Relational, Multi-model 	613.83	-2.67	-3.10
5.	5.	5.	MongoDB 	Document, Multi-model 	458.78	+6.02	-26.88
6.	6.	6.	Redis 	Key-value, Multi-model 	172.45	-1.39	-4.31
7.	7.	7.	IBM Db2	Relational, Multi-model 	142.92	-0.04	-19.22
8.	8.	8.	Elasticsearch	Search engine, Multi-model 	139.07	+0.47	-20.88
9.	9.	↑ 10.	SQLite 	Relational	133.82	+1.15	+1.64
10.	10.	↓ 9.	Microsoft Access	Relational	132.06	+1.03	-3.37
11.	↑ 12.	↑ 14.	Snowflake 	Relational	114.40	-1.26	+28.17
12.	↓ 11.	↓ 11.	Cassandra 	Wide column	113.79	-2.43	-8.35
13.	13.	↓ 12.	MariaDB 	Relational, Multi-model 	96.84	+0.03	-11.47
14.	14.	↓ 13.	Splunk	Search engine	87.97	+0.89	-7.39
15.	15.	↑ 16.	Amazon DynamoDB 	Multi-model 	80.77	+1.08	-1.03
16.	16.	↓ 15.	Microsoft Azure SQL Database	Relational, Multi-model 	77.44	-1.31	-7.23

<https://db-engines.com/en/ranking>



Architecture

© 2022 - 2023 Siam Chamnankit Company Limited. All rights reserved.

Use right storage for write need !

Data durability

Data availability

Data integrity

Data throughput (read/write)

Data size

Data load

Data query



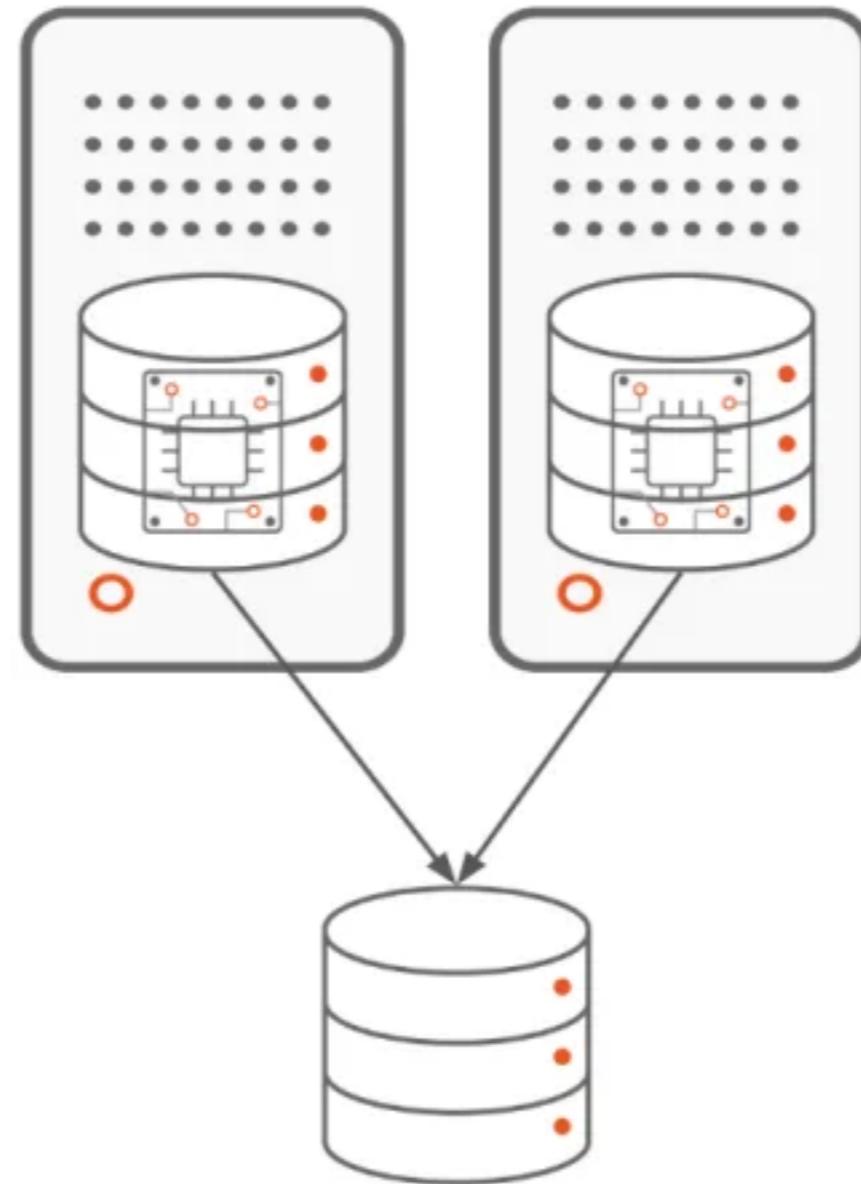
HA for Database

Redundancy/Clustering
Supporting subdomain
Generic subdomain



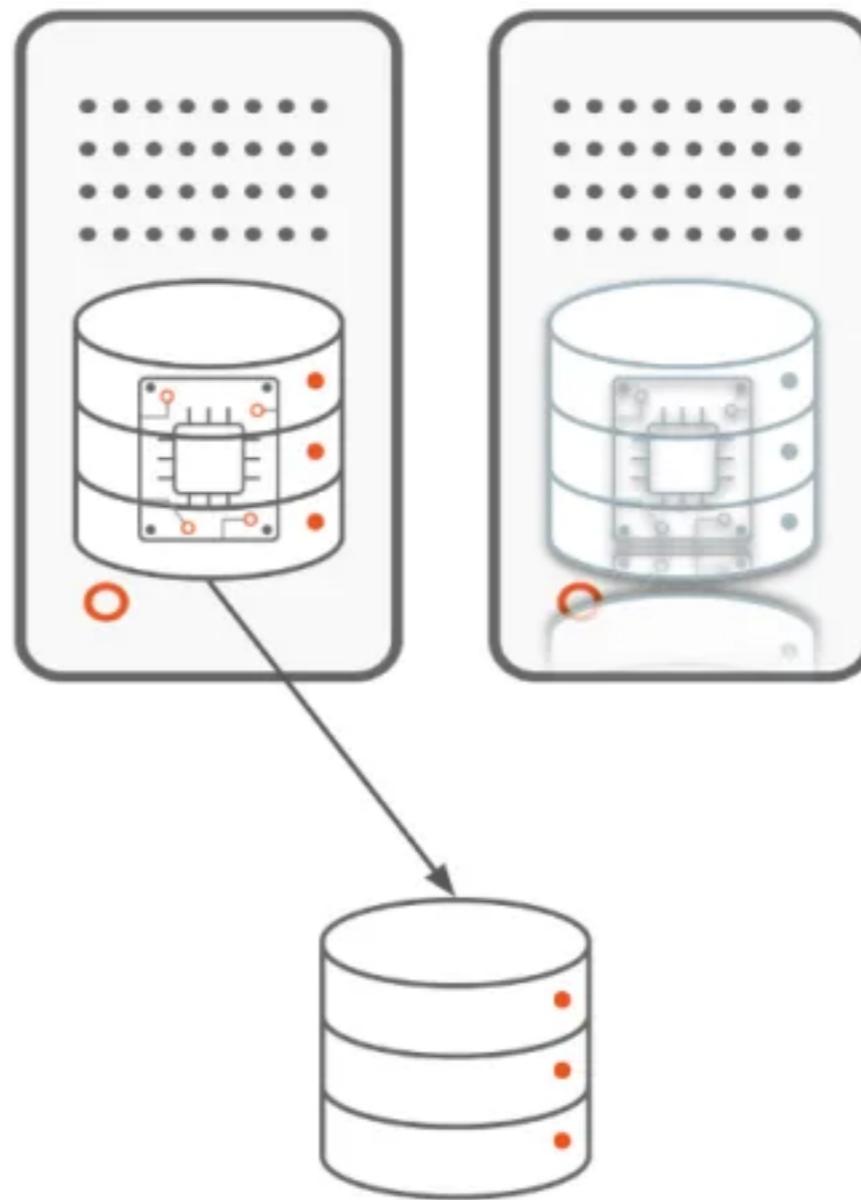
Redundancy/Clustering

Active/Active



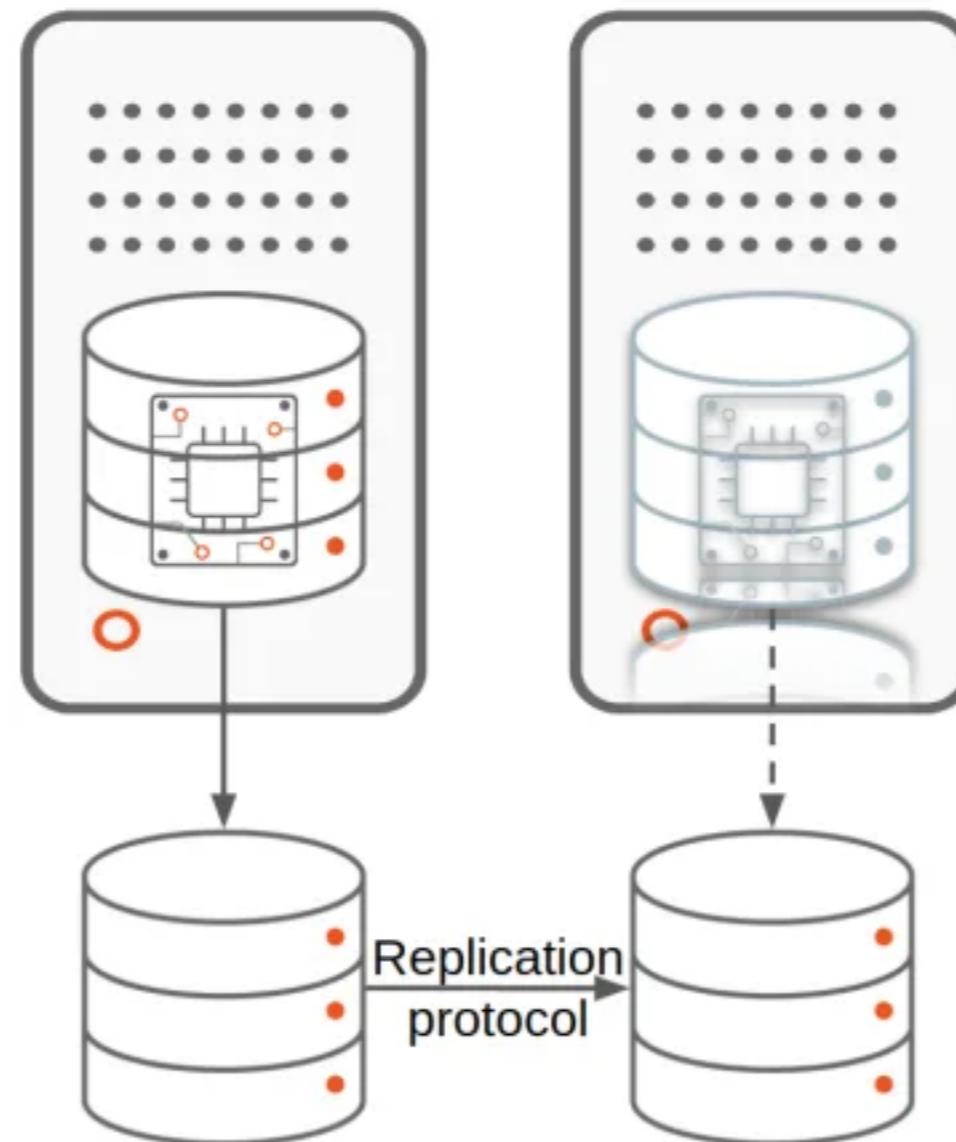
Redundancy/Clustering

Active/Passive



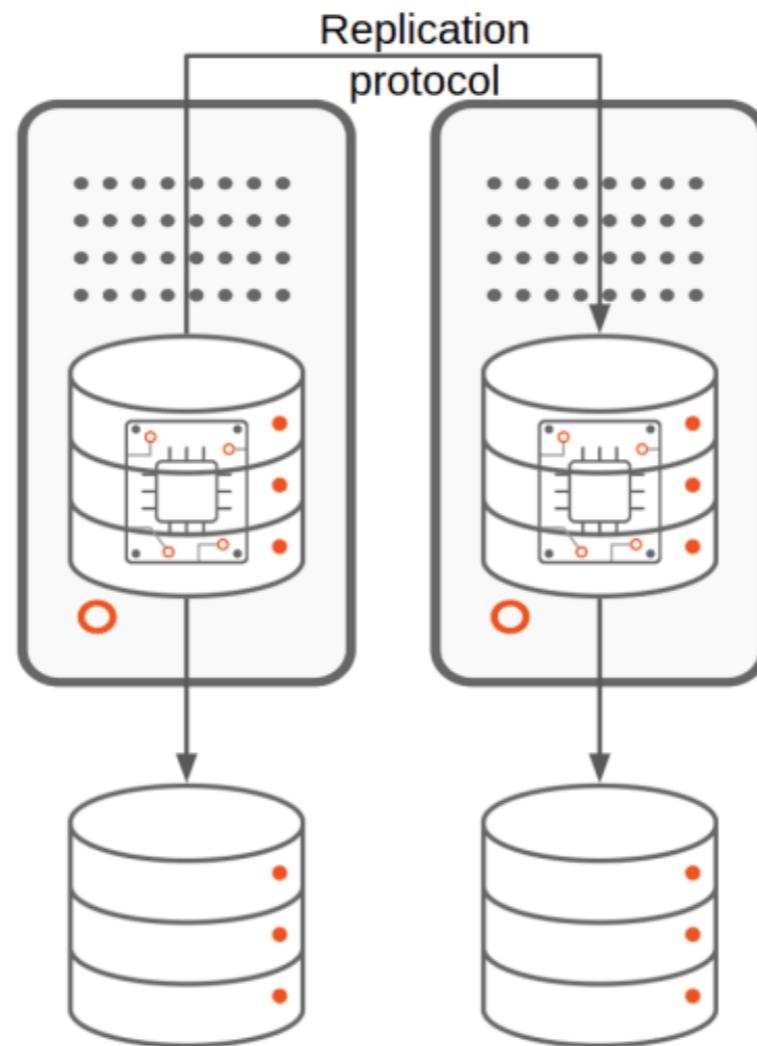
Redundancy/Clustering

Storage-based replication for database clustering

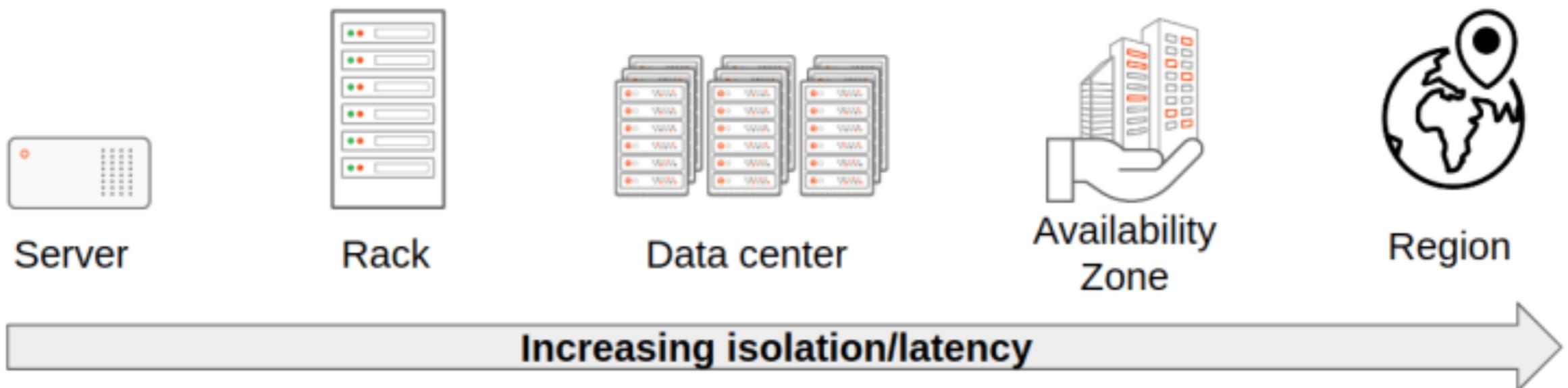


Redundancy/Clustering

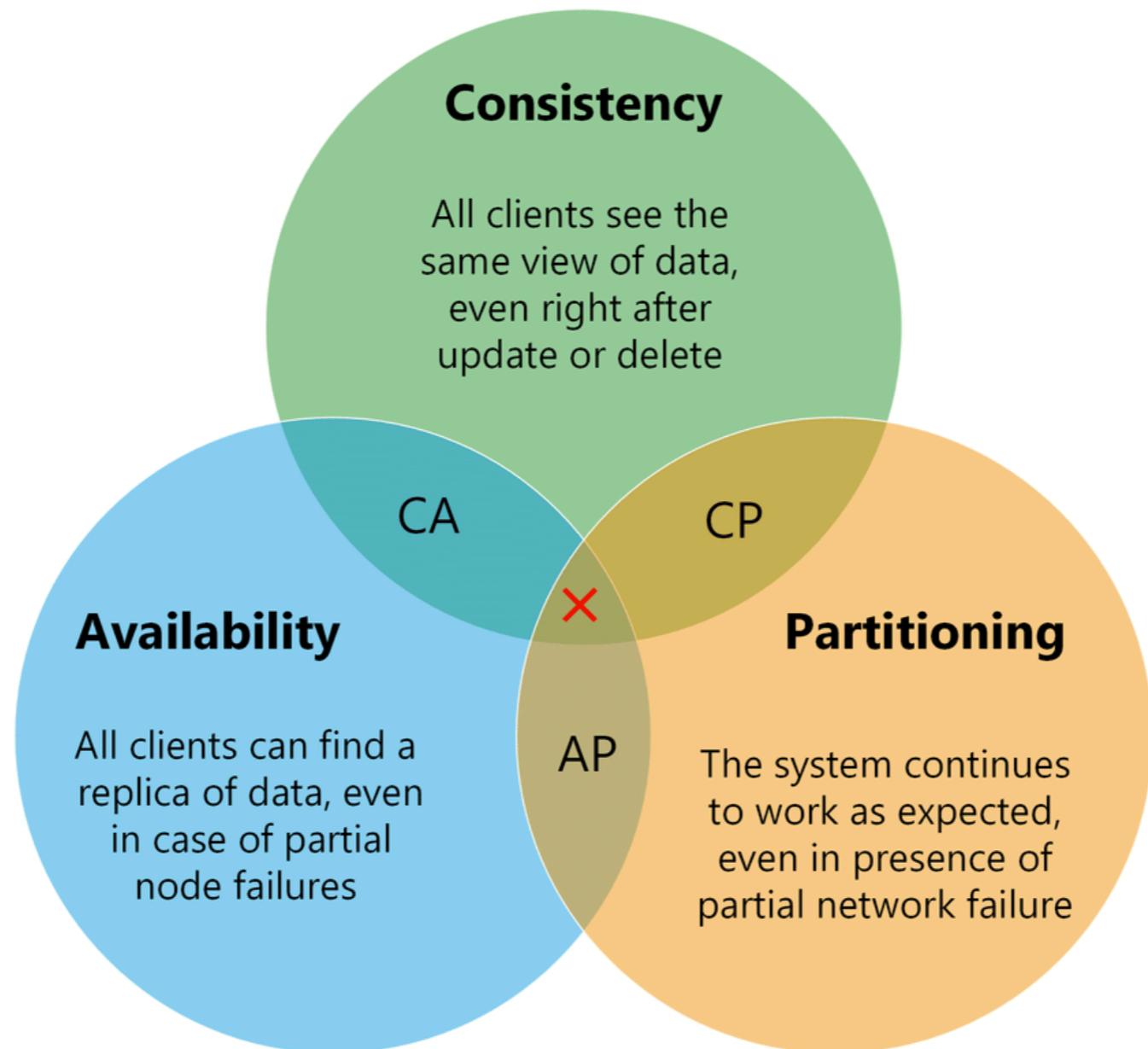
Database solution-based replication for database clustering



Isolation of Database ?



CAP theorem



Data types ?

Data Type	Database model	Example
Transactional Structured	Relational	MySQL PostgreSQL
Key-value semi-structured Unstructured	NoSQL	MongoDB Cassandra
In-memory	Cache (key-value)	Redis Memecached ElasticCached
Streaming	Temporary storage	Apache Kafka Amazon Kinesis Spark streaming
Object	File-based	Amazon S3 Google Storage SAN
Search	Searching	Apache Solo Elasticsearch

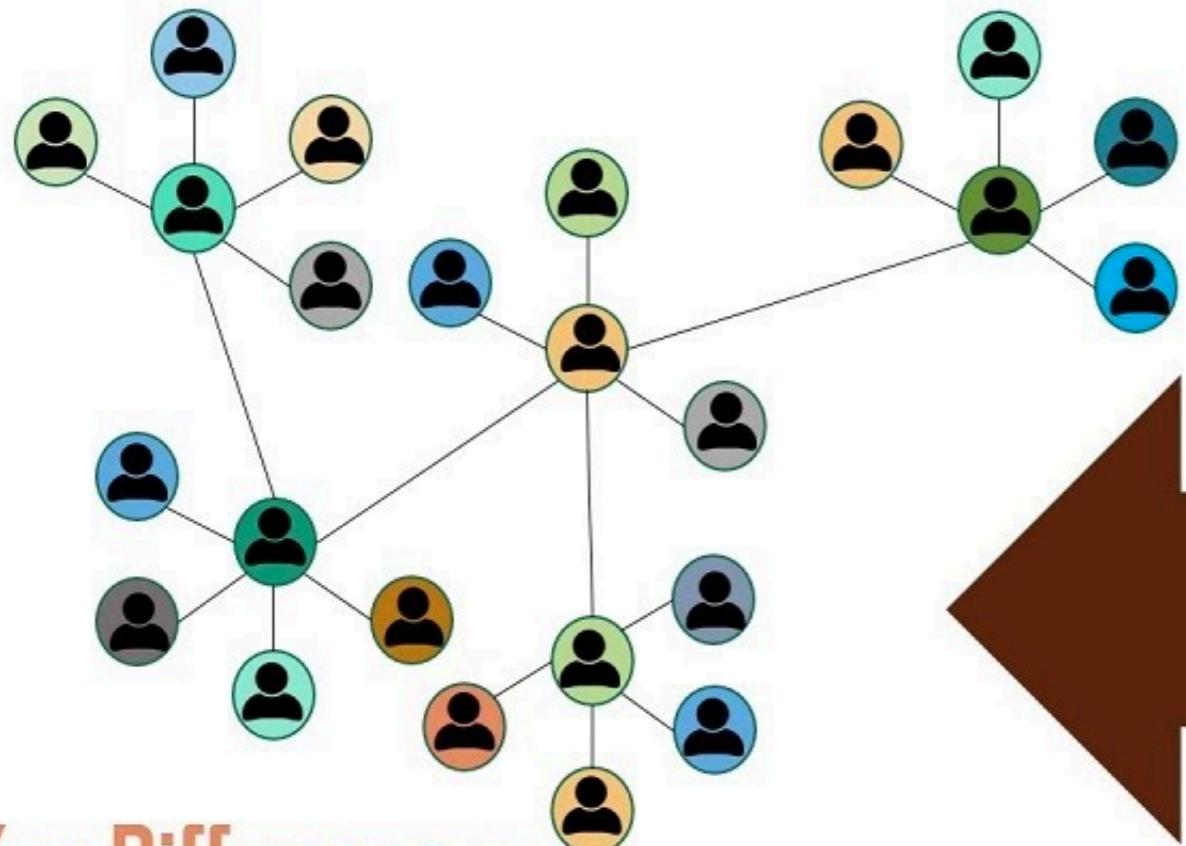


Architecture Reliability

Architecture Resilience



Centralization



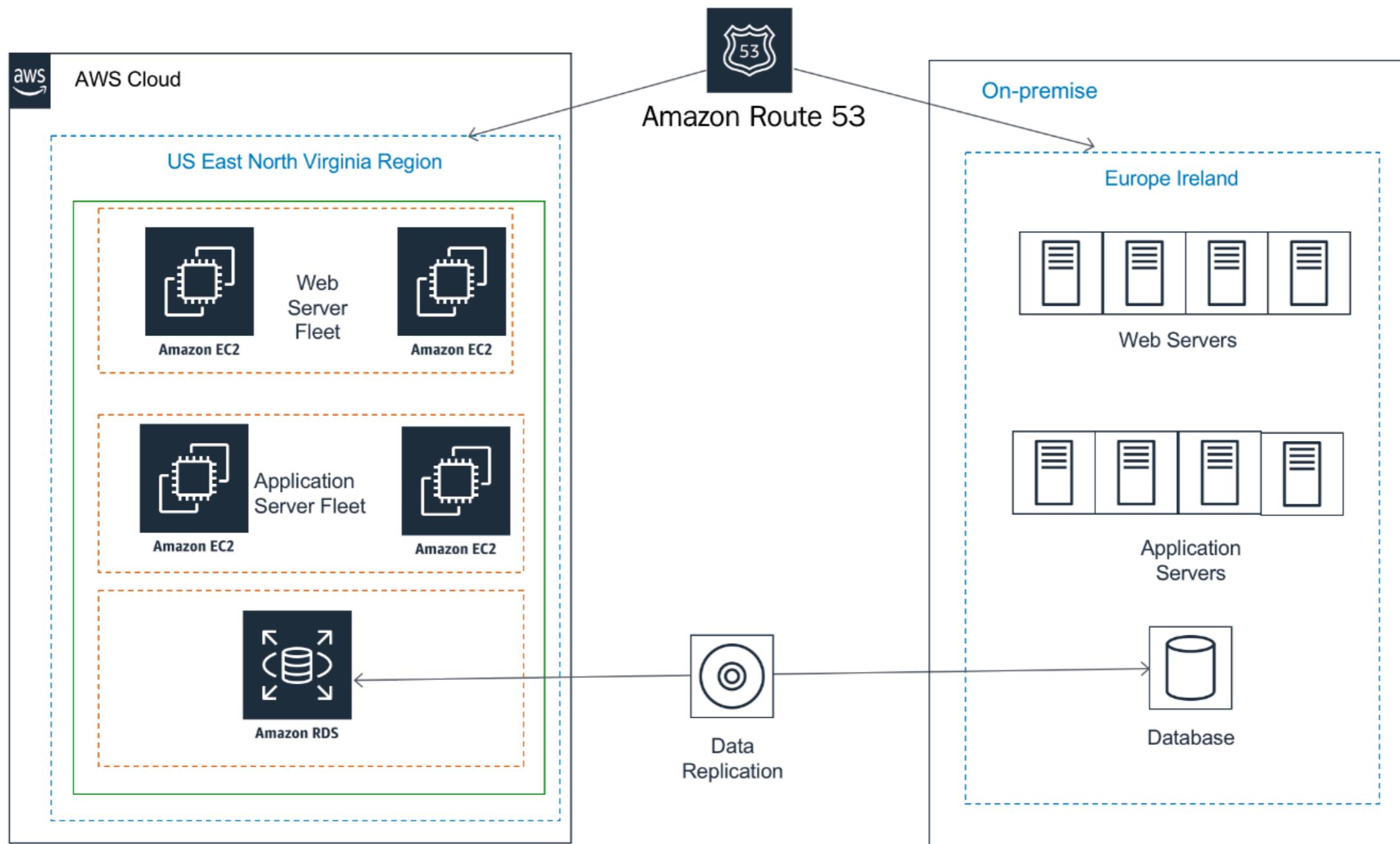
Decentralization



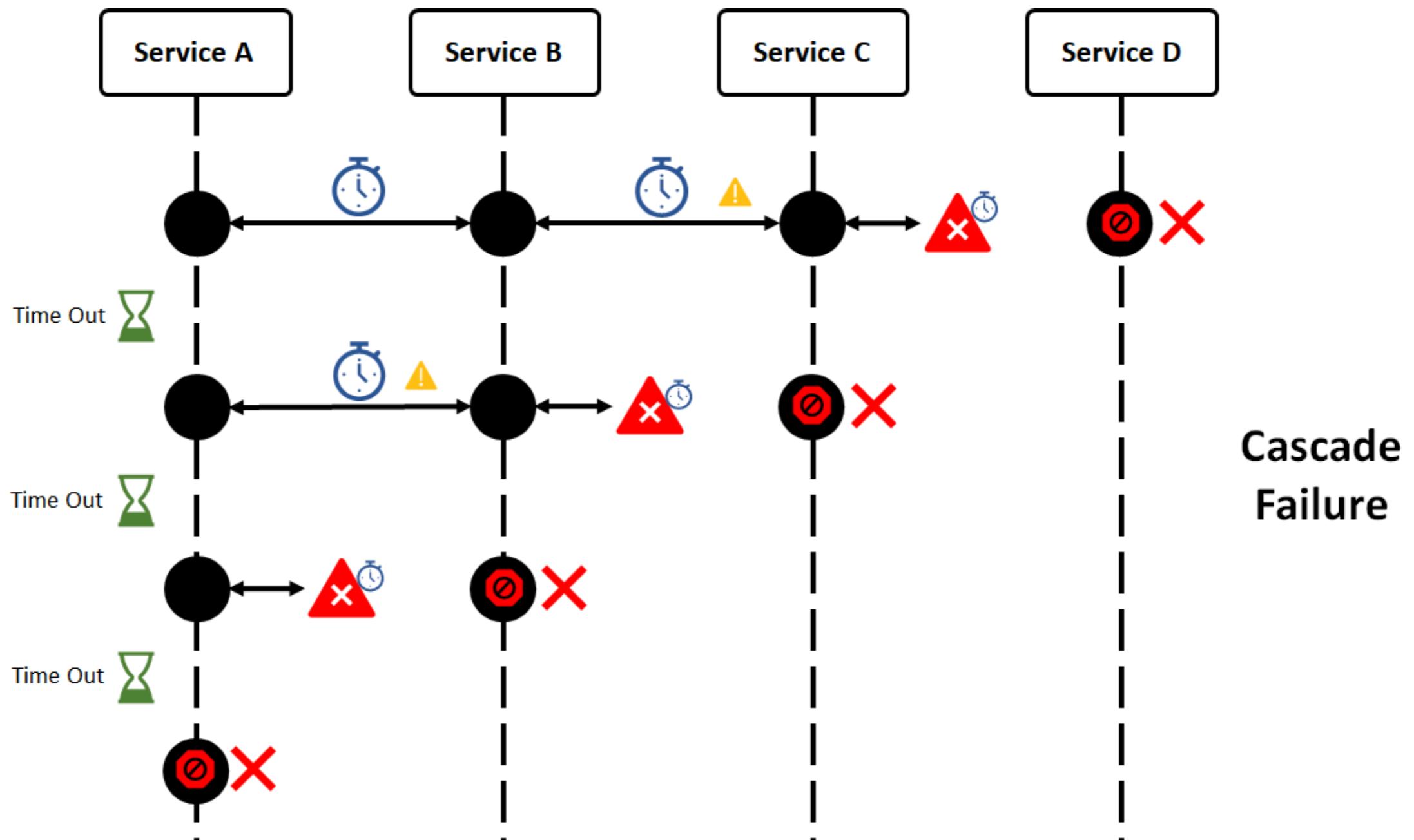
Key Differences



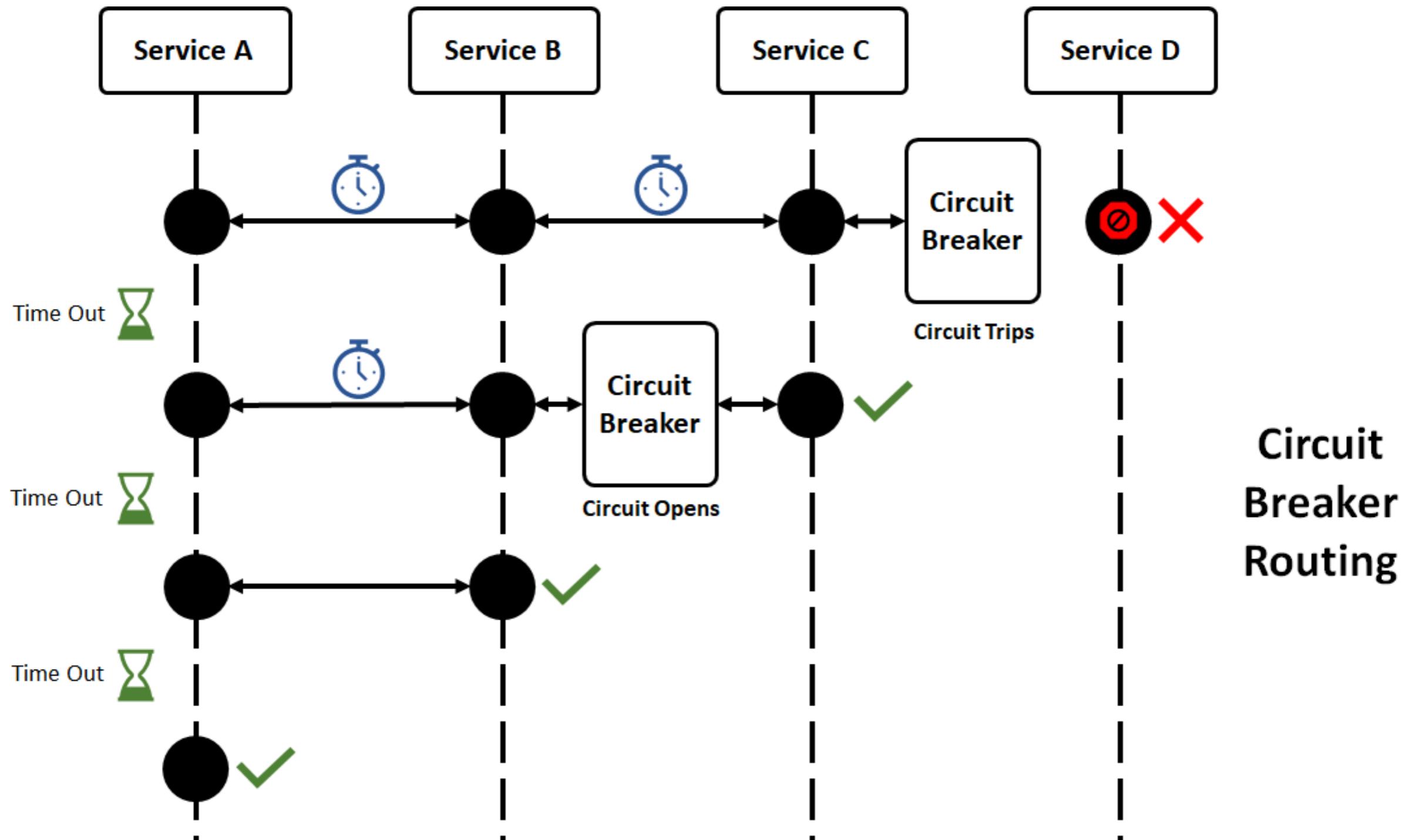
Disaster Recovery Architecture



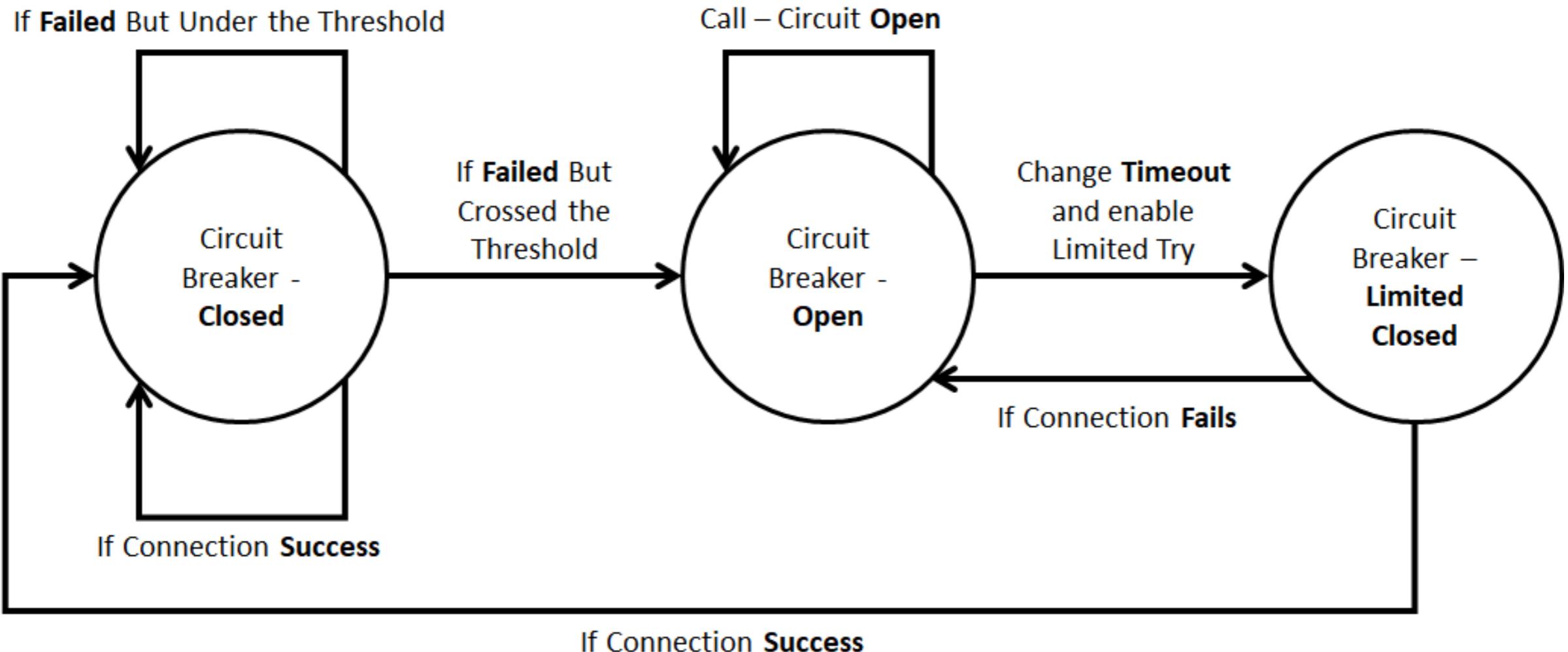
Without Circuit breaker



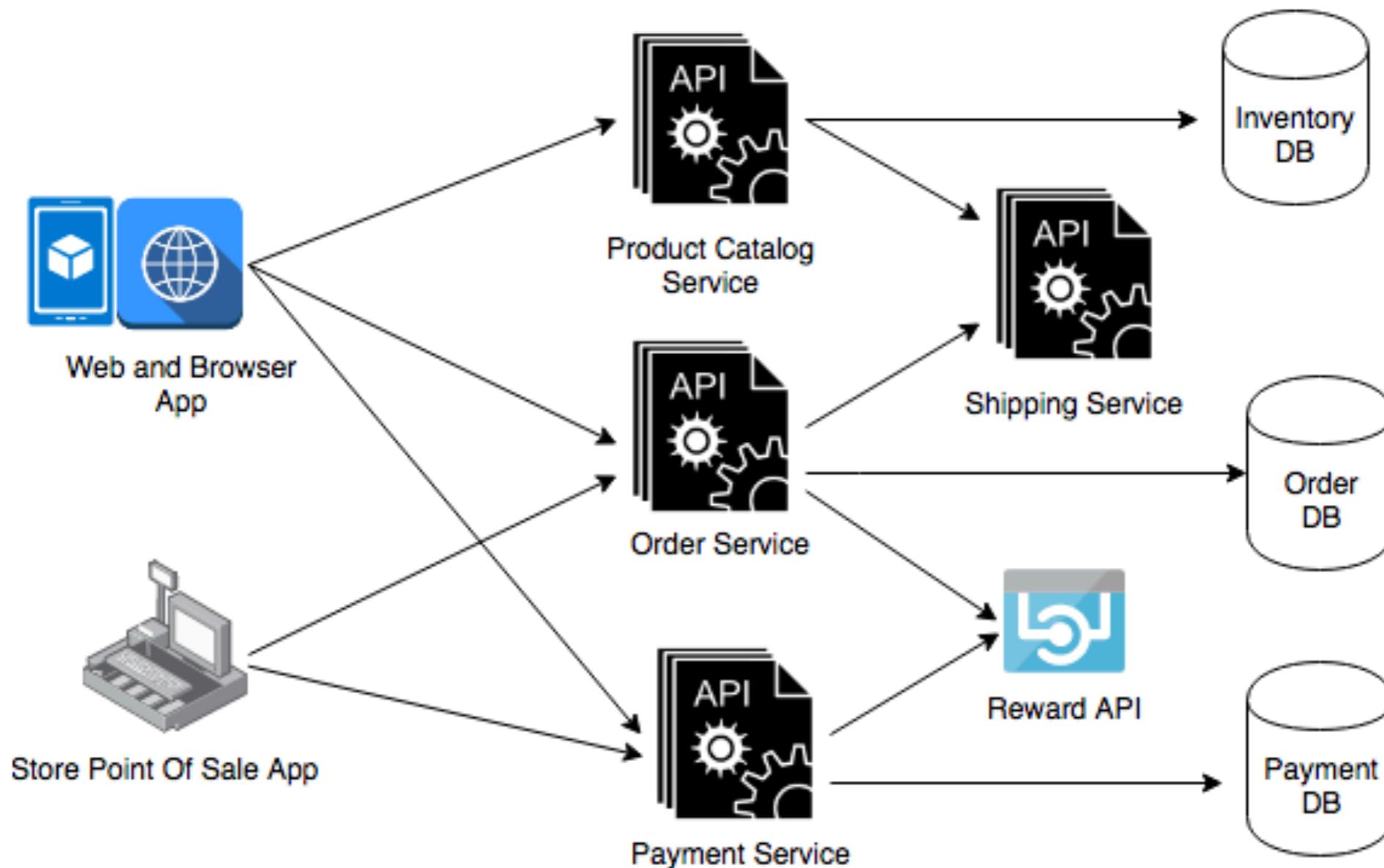
With Circuit breaker



Circuit breaker



Extendability and Reusability !!



Excellence Operation

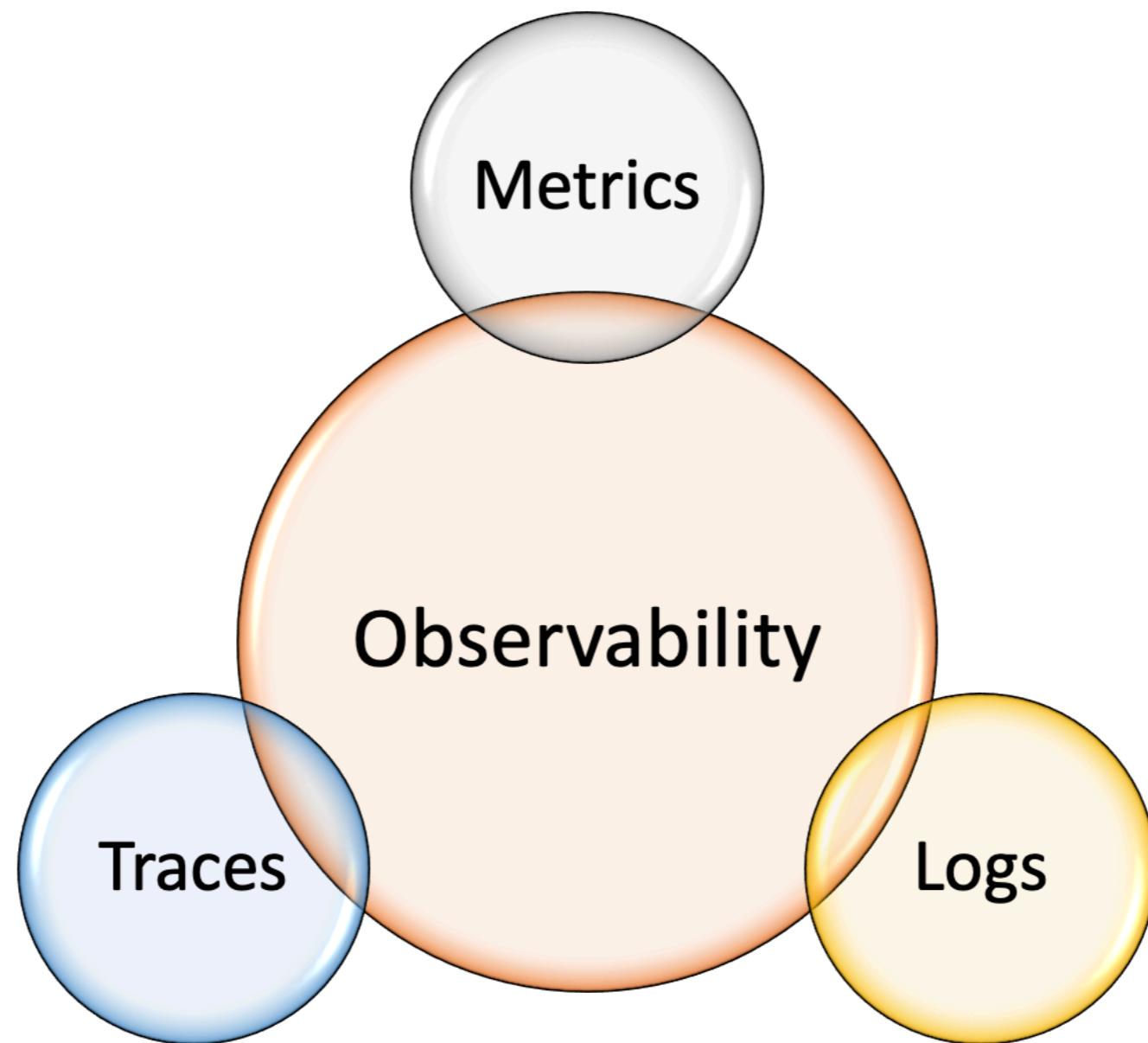


Excellence operation and maintain

**Zero downtime !!
Minimal outage
High quality
Proactive**



Better understand the system



https://linkedin.github.io/school-of-sre/level101/metrics_and_monitoring/observability/



Distributed Tracing

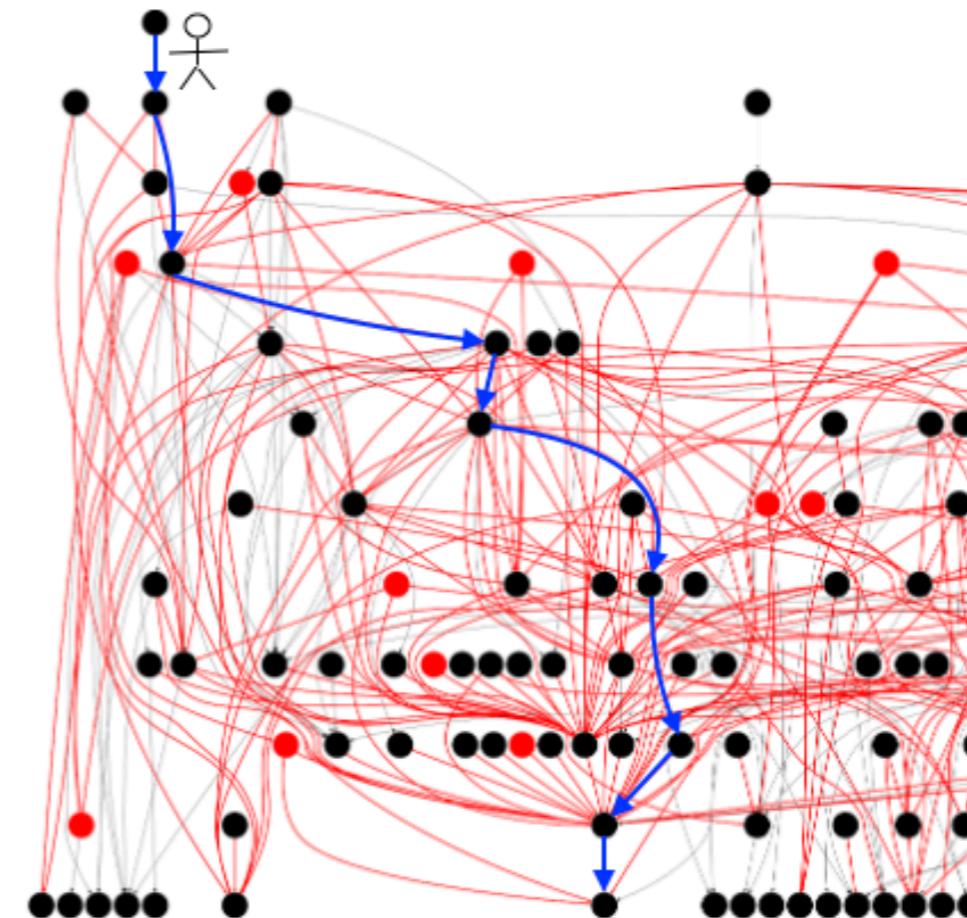
What happened to my request?

- = Service-to-Service Connection
- = Individual Request Path

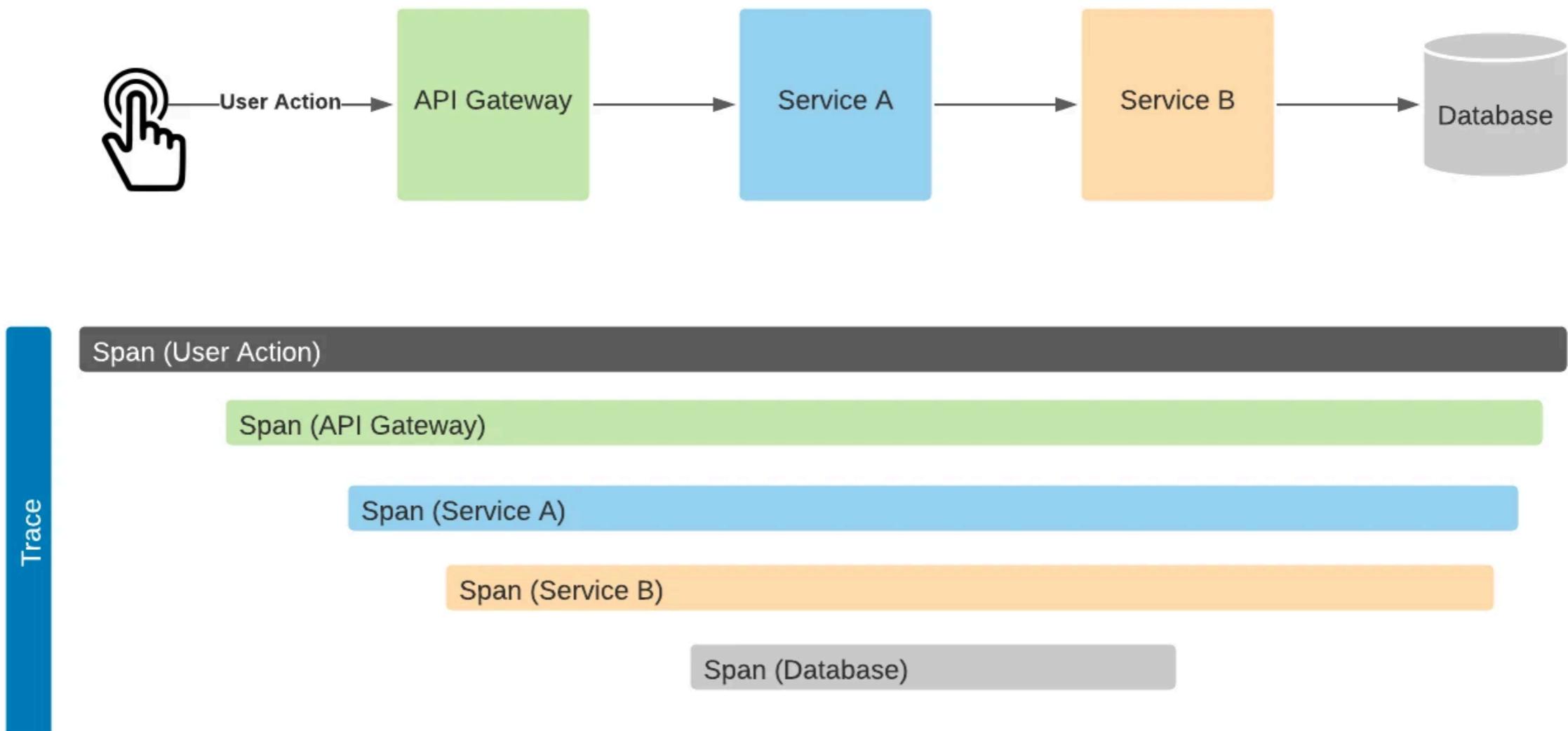
Without Distributed Tracing



With Distributed Tracing



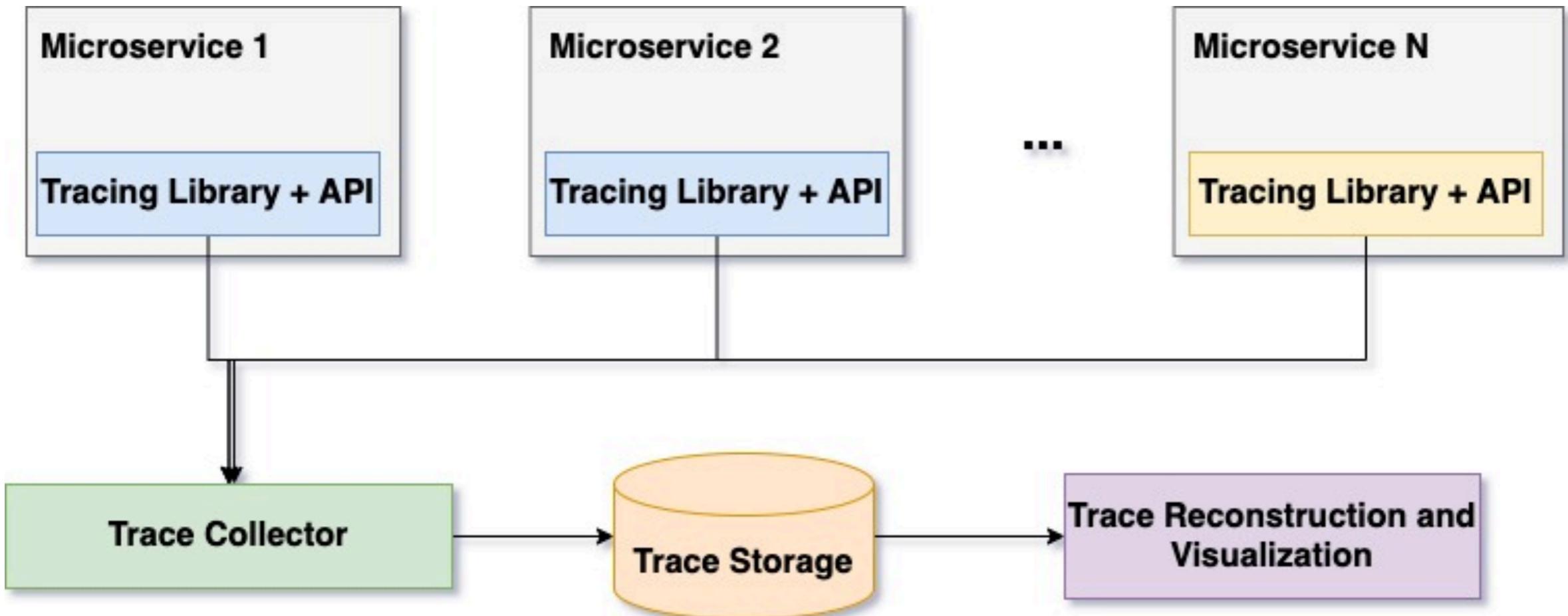
Distributed Tracing



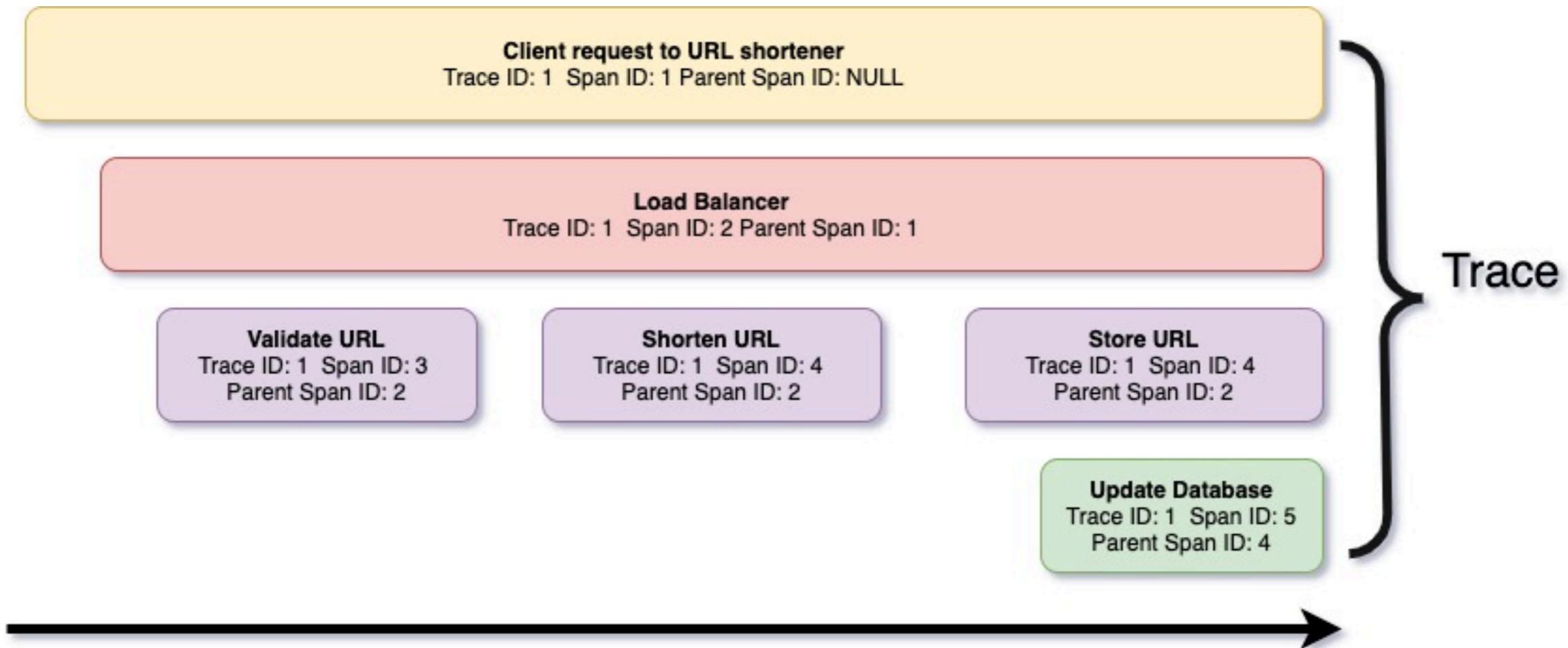
<https://www.dynatrace.com/news/blog/open-observability-distributed-tracing-and-observability/>



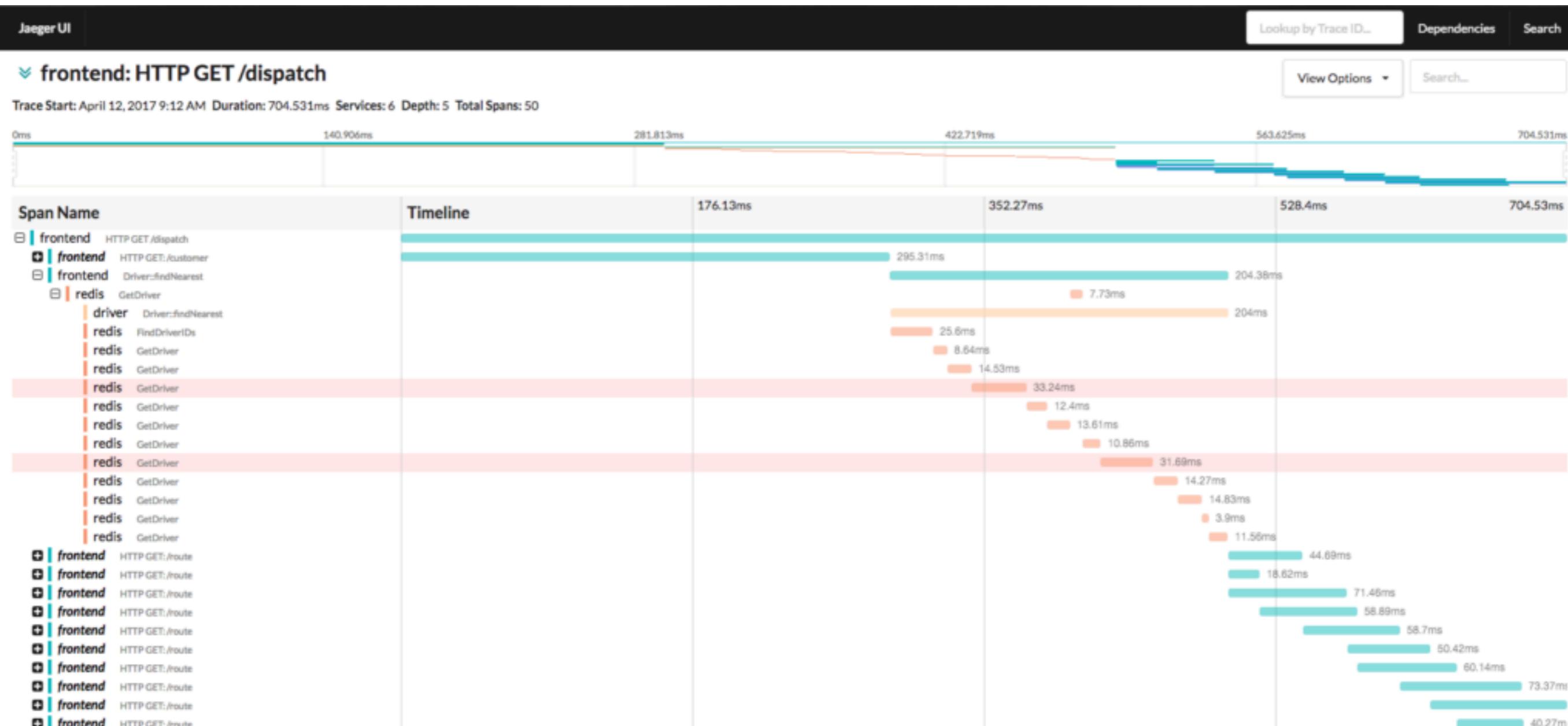
Distributed Tracing



Distributed Tracing



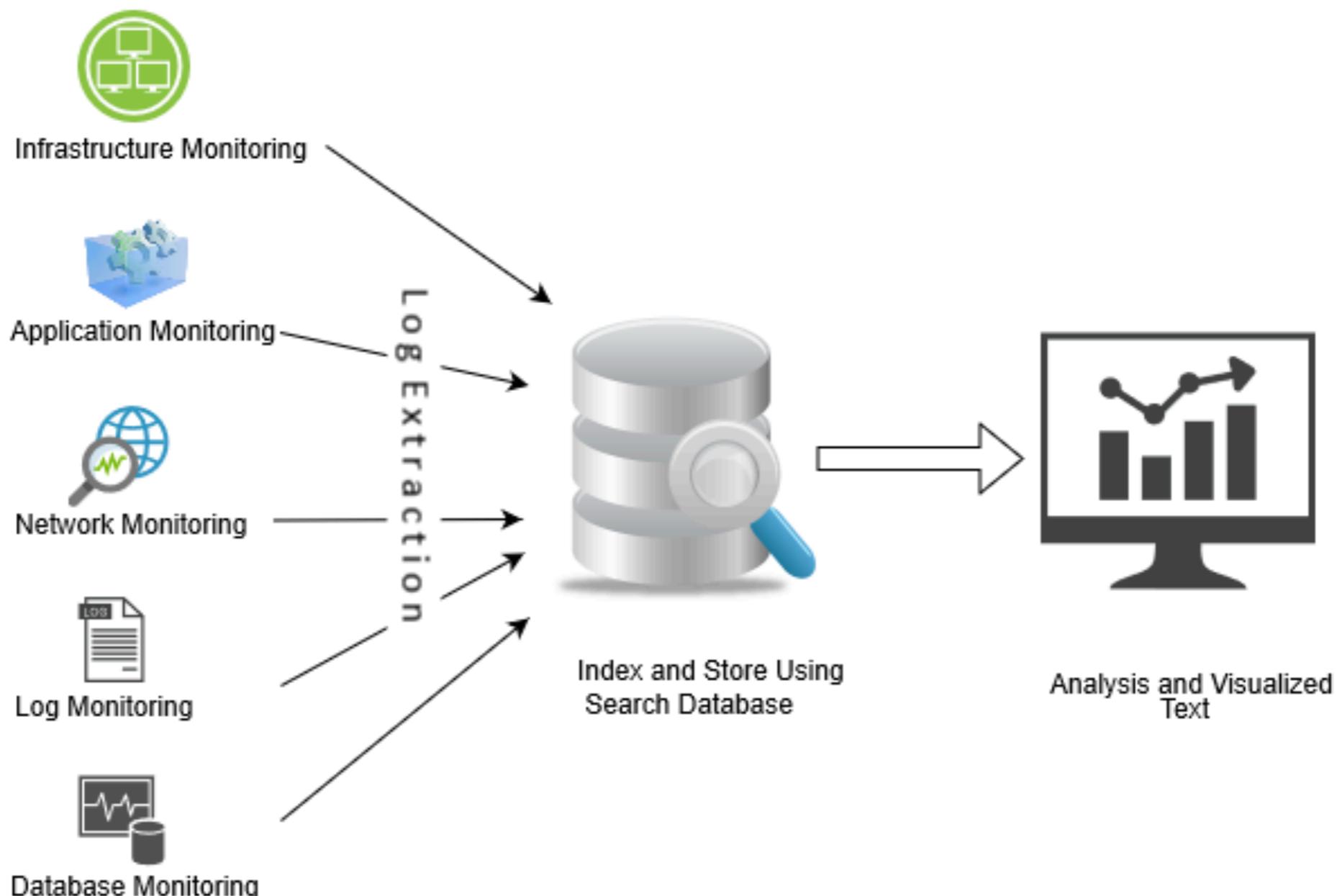
Distributed Tracing



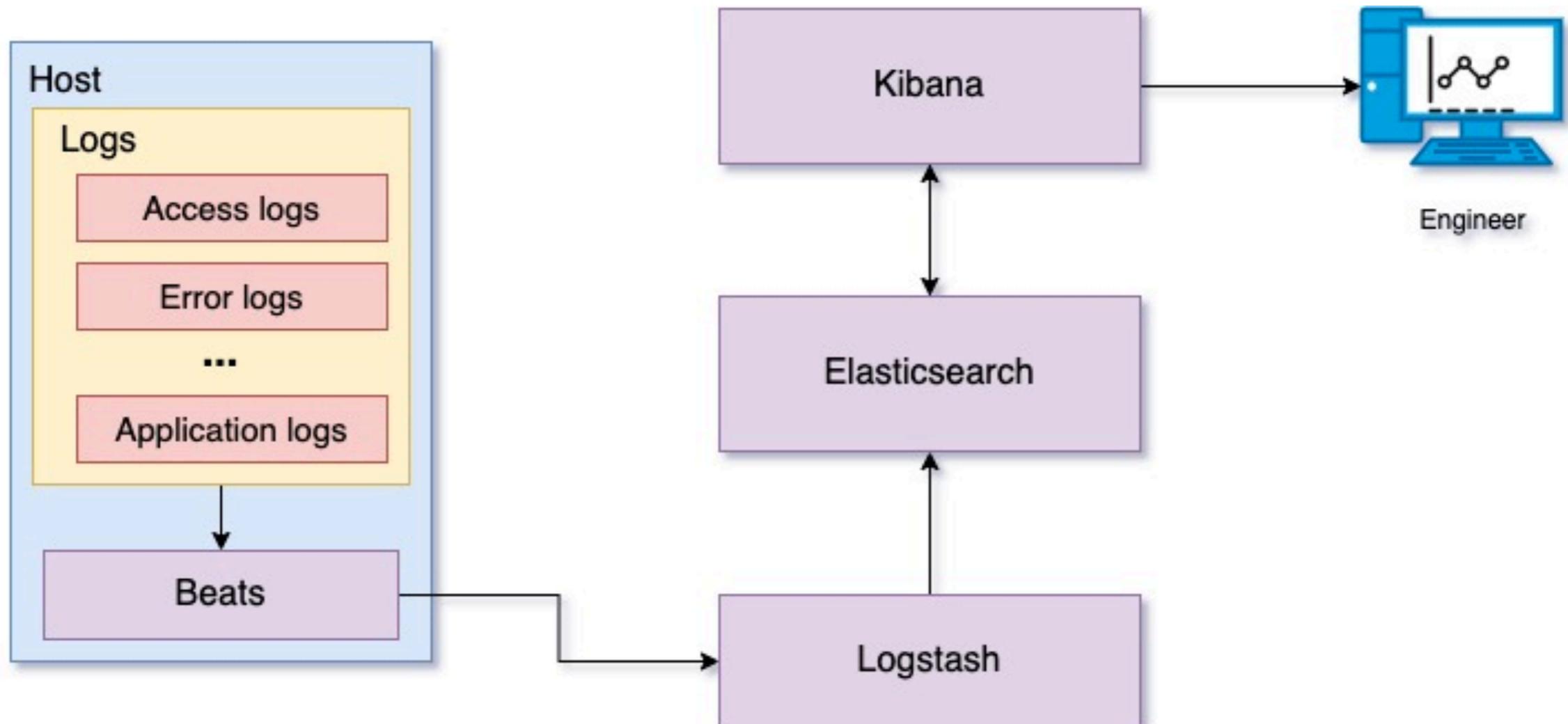
<https://zipkin.io/>



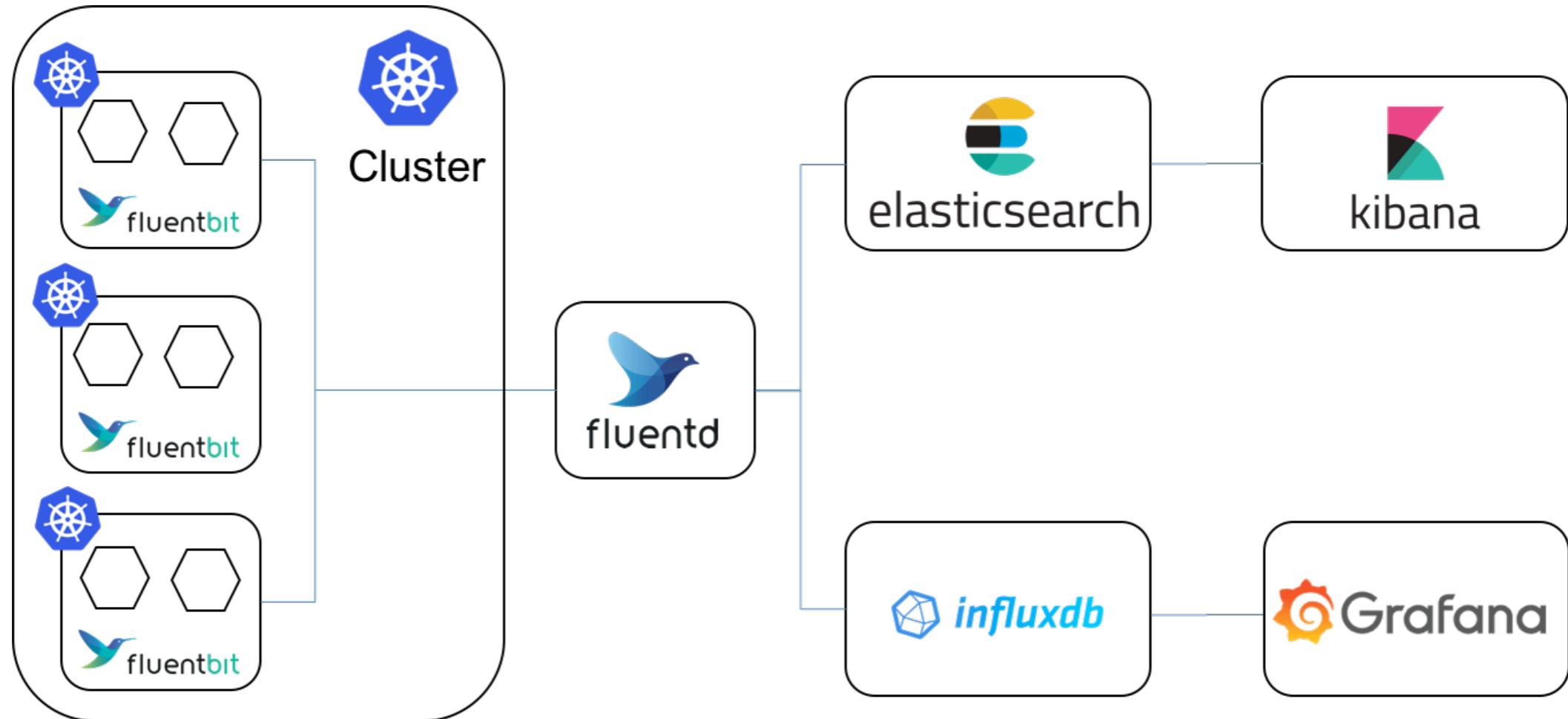
Centralized Logging



Centralized Logging



Centralized Logging



Data
Collection

Data
Aggregation
& Processing

Indexing &
storage

Analysis &
visualization



Automating Everything



Automated

Most accidents come from human error
Increase productivity
Save cost

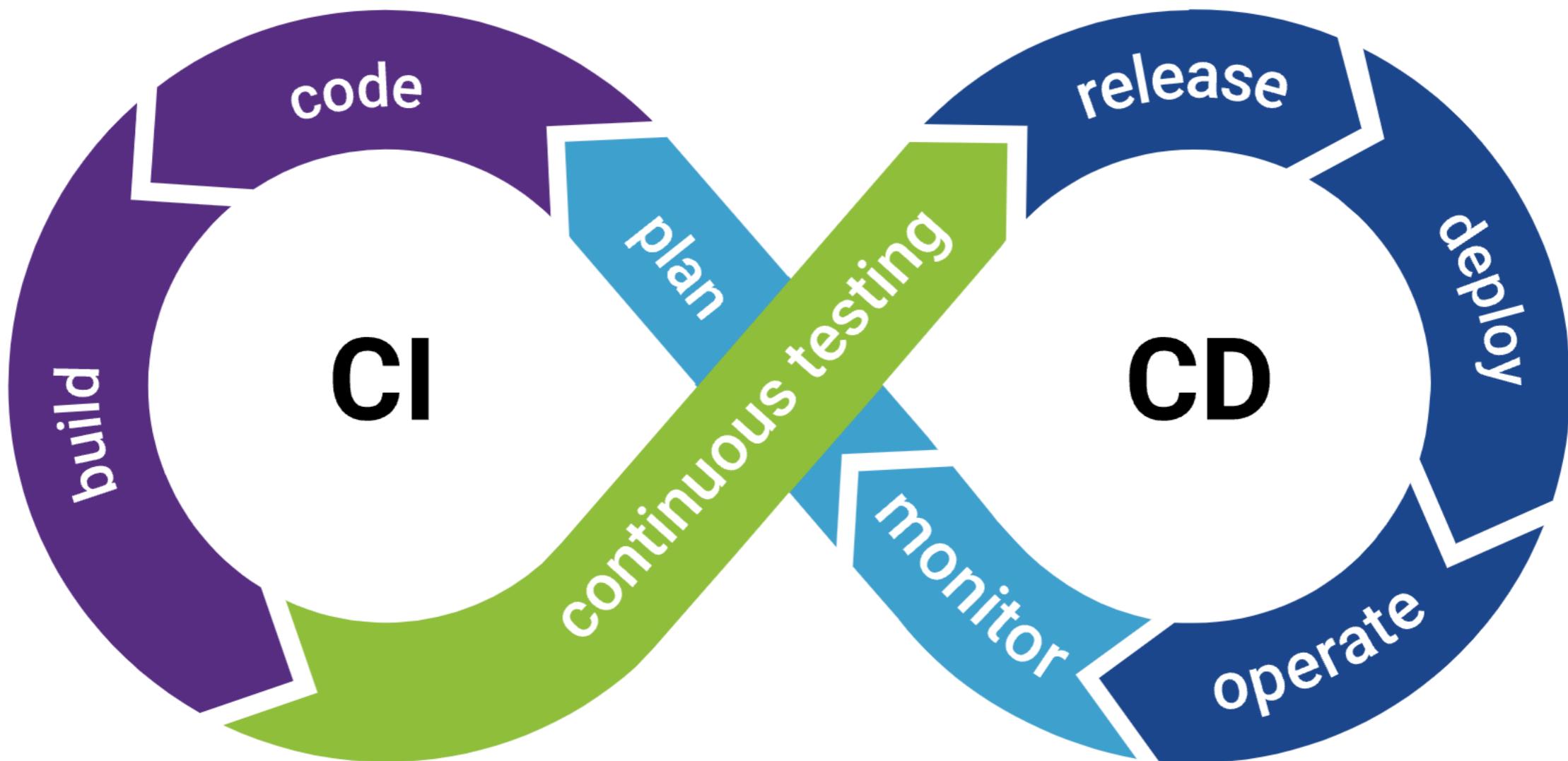


Automated ?

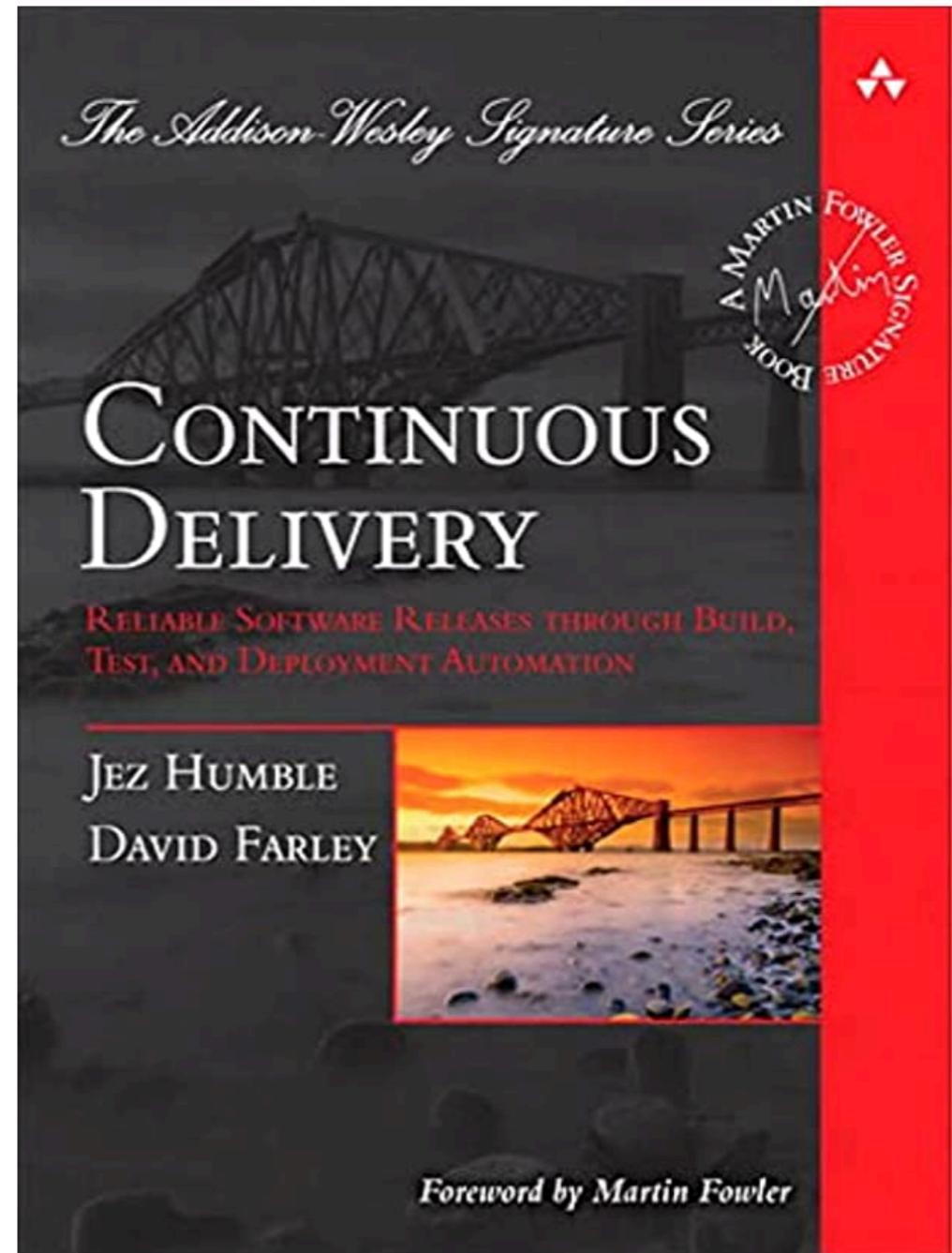
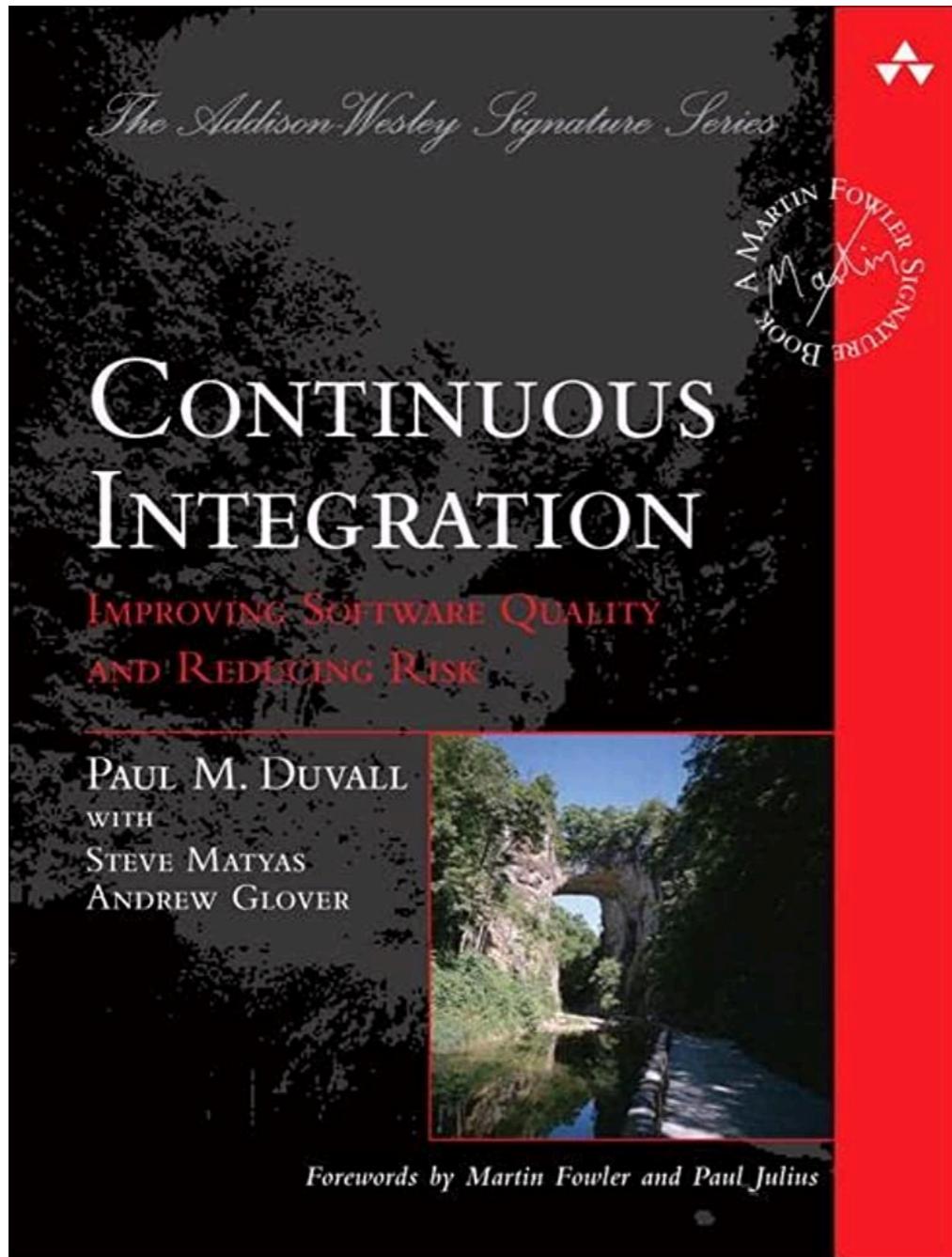
Testing
Infrastructure
Logging, monitoring, alerting
Deployment
Security



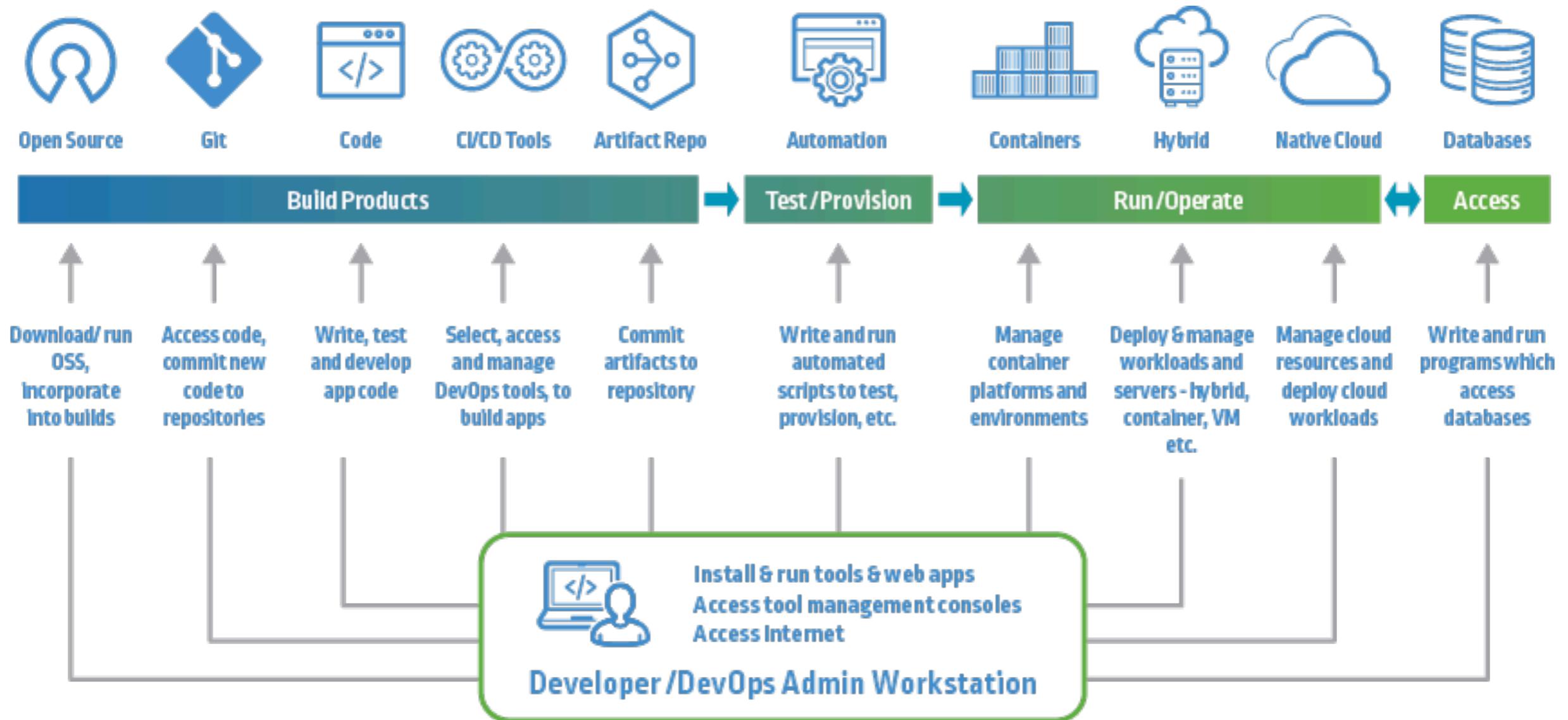
CI/CD



CI/CD



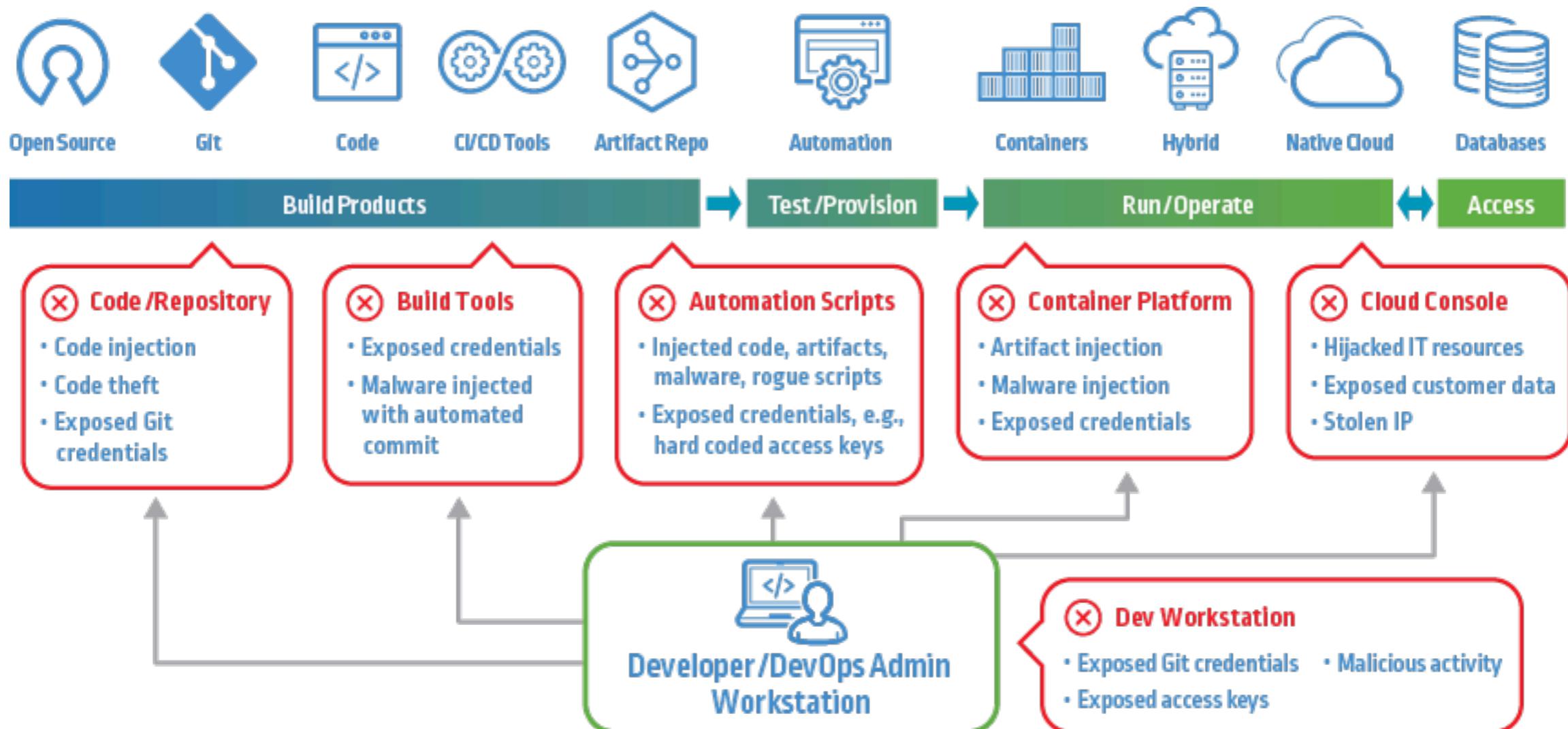
CI/CD



<https://www.cyberark.com/what-is/ci-cd-pipeline/>



CI/CD with secret management



<https://www.cyberark.com/what-is/ci-cd-pipeline/>

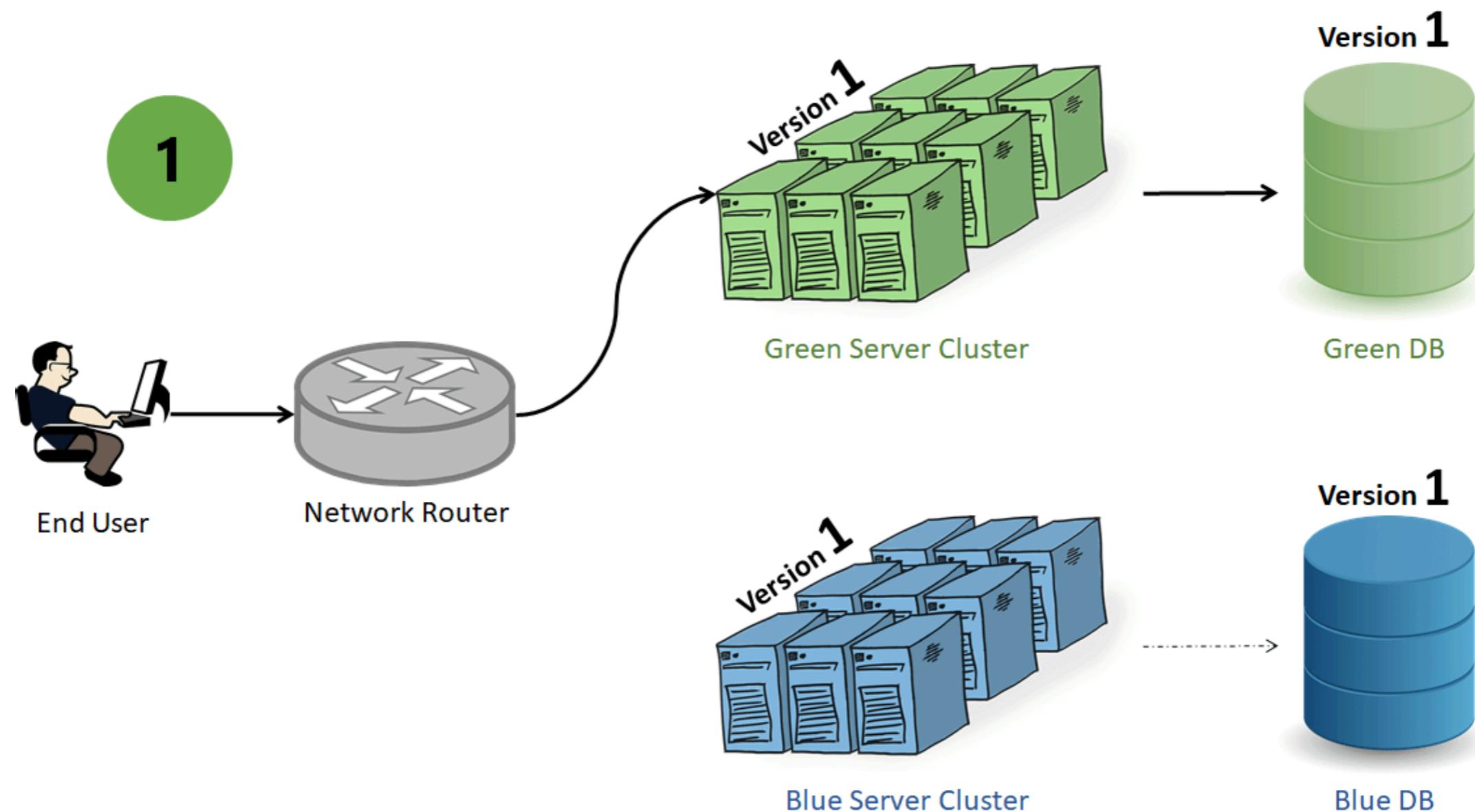


Deployment strategies

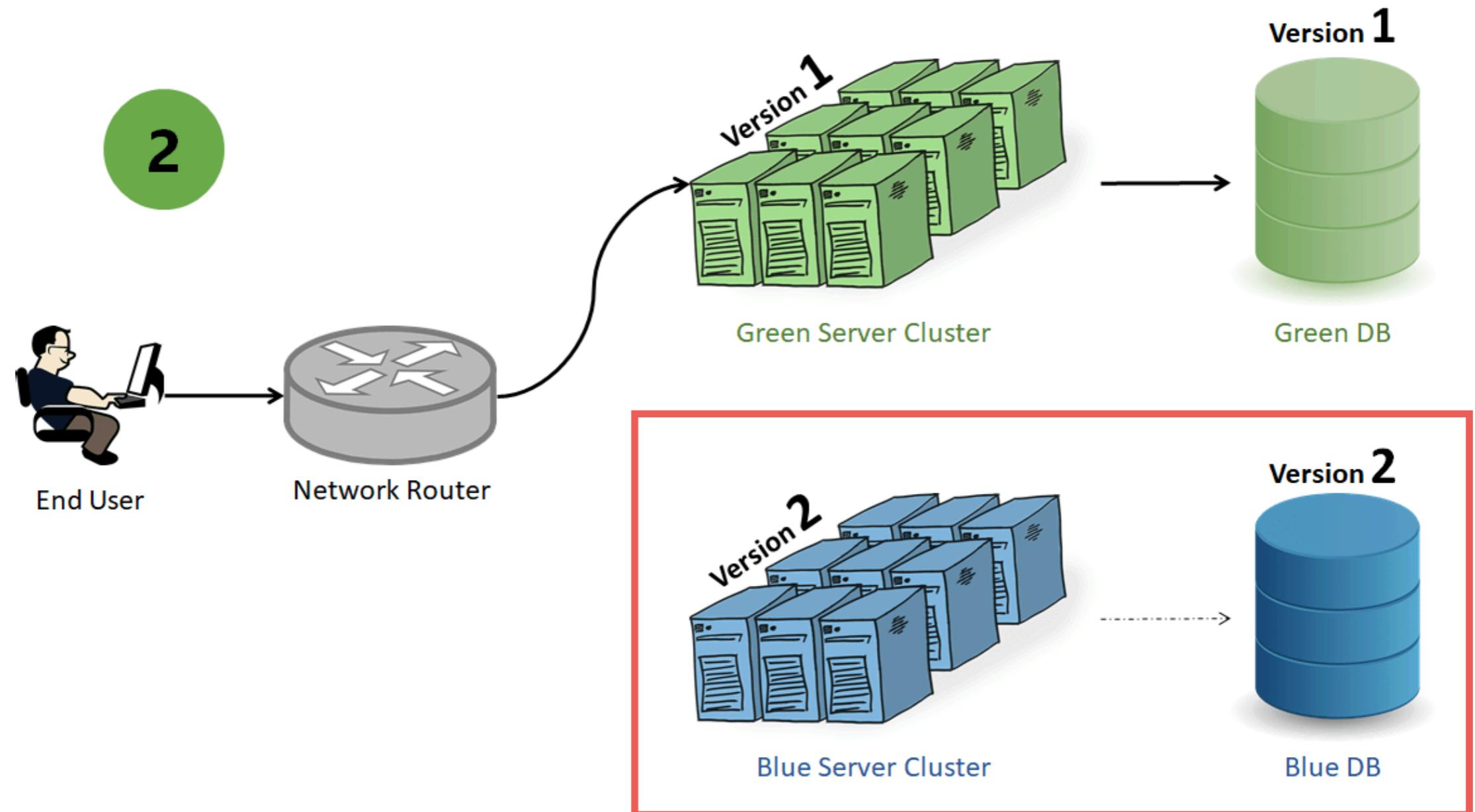
Big bang !!
Blue-green deployment
Canary release
Proactive



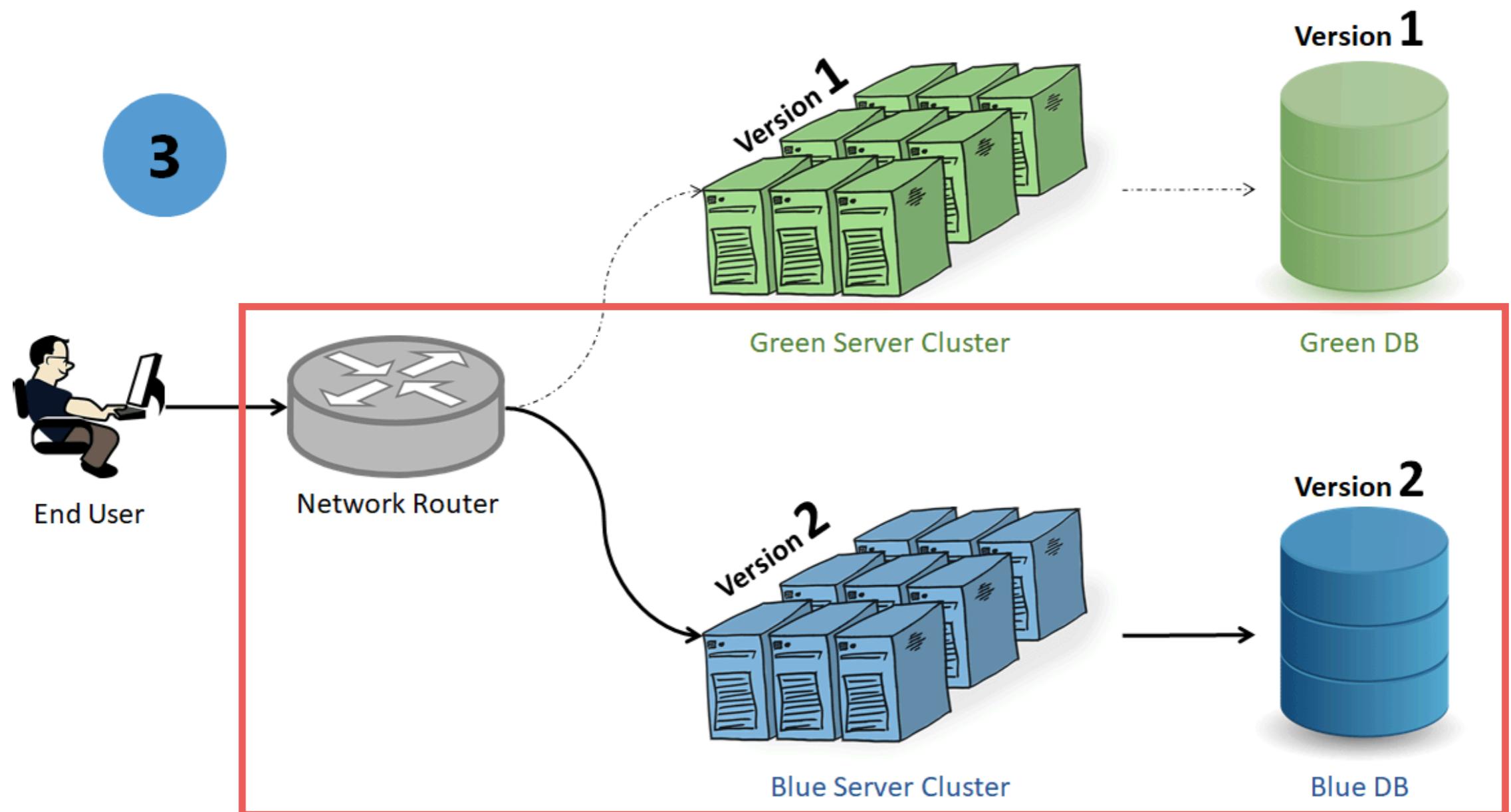
Blue-green deployment



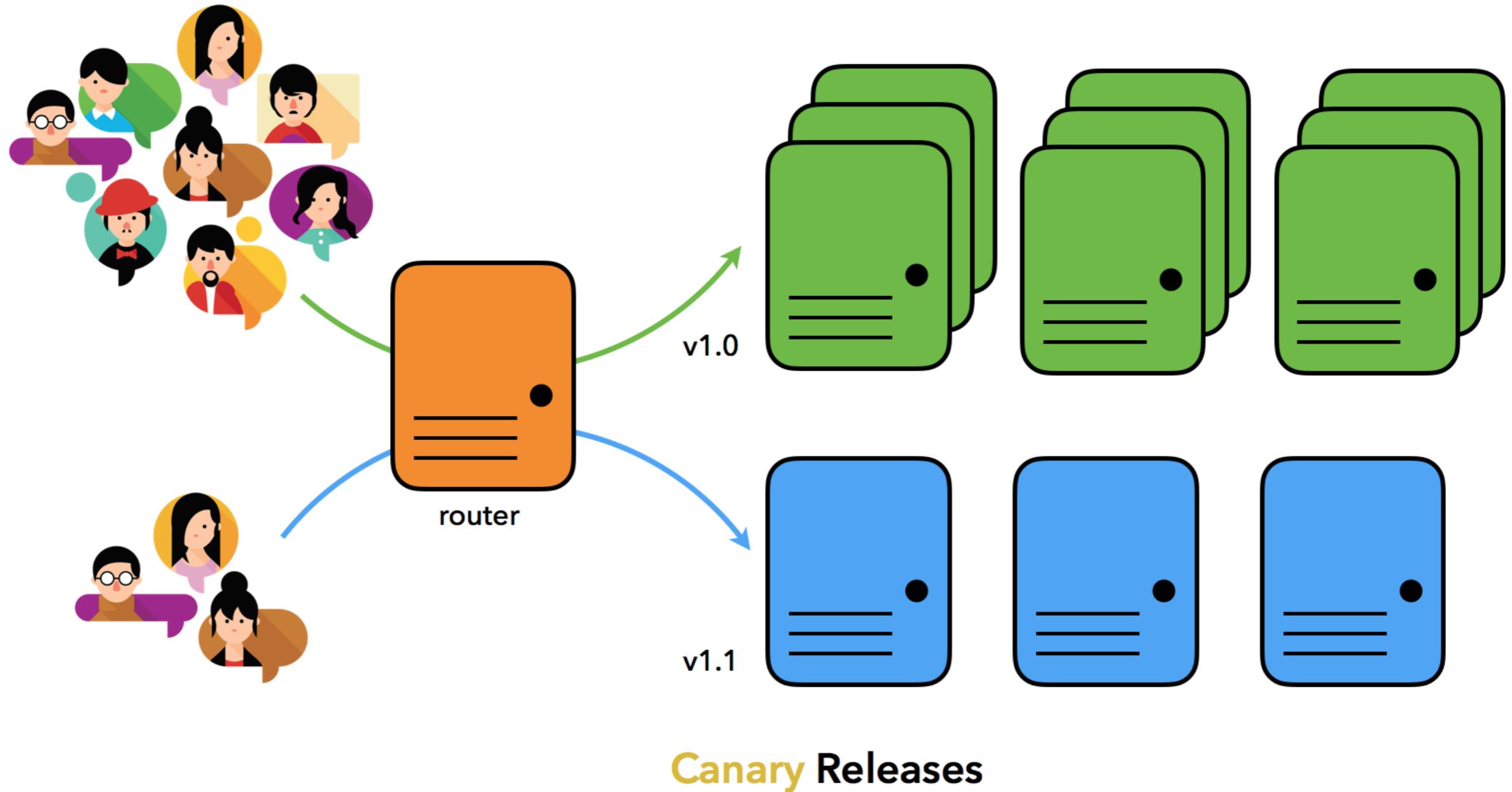
Blue-green deployment



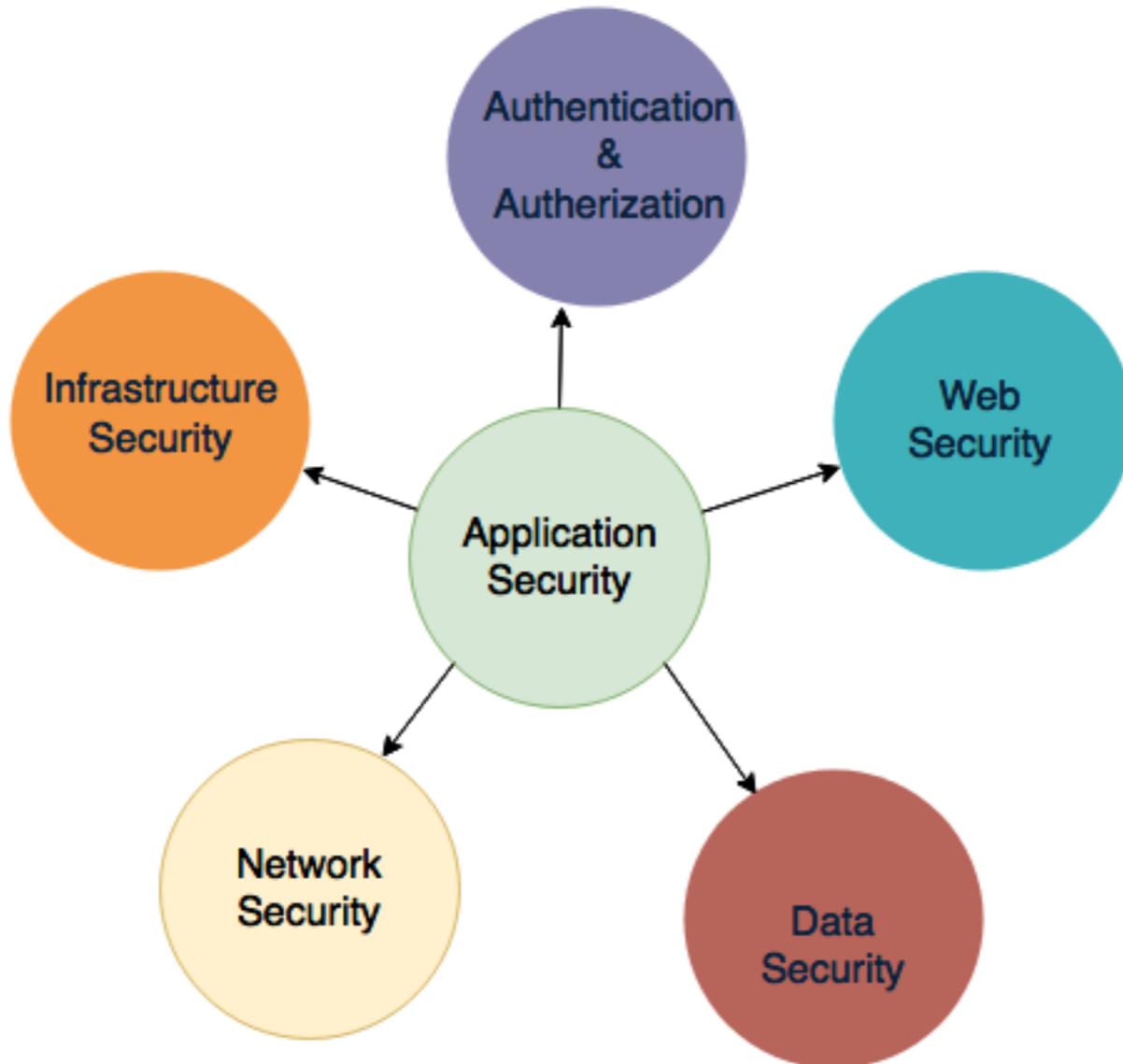
Blue-green deployment



Canary release

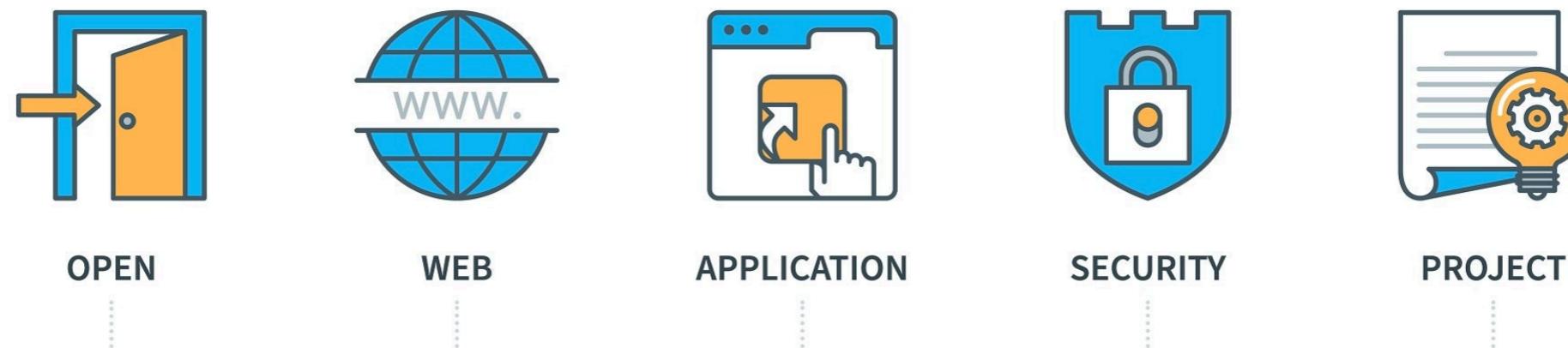


Security and Compliance



Security and Compliance

OWASP Top 10
Web/API/Mobile/Docker/Kubernetes



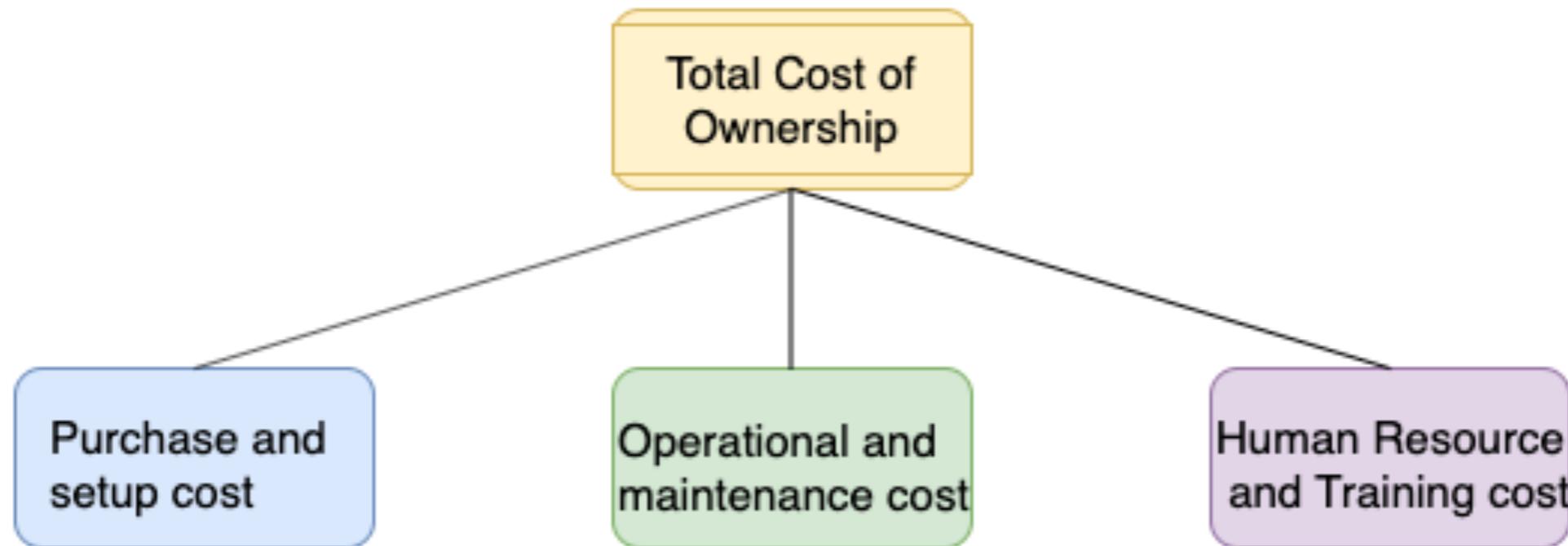
OWASP

<https://owasp.org/www-project-top-ten/>

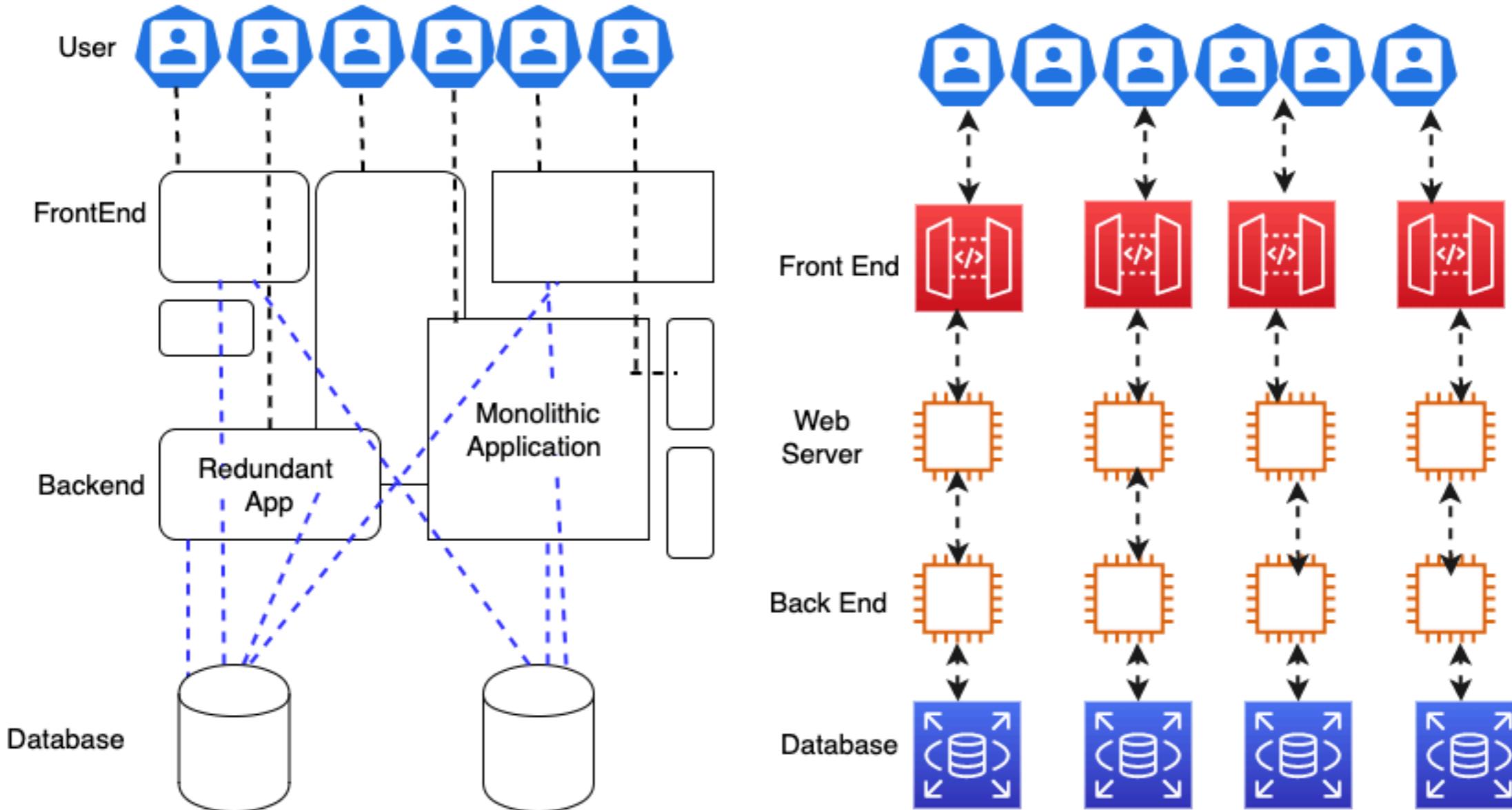


Reduce architecture complexity

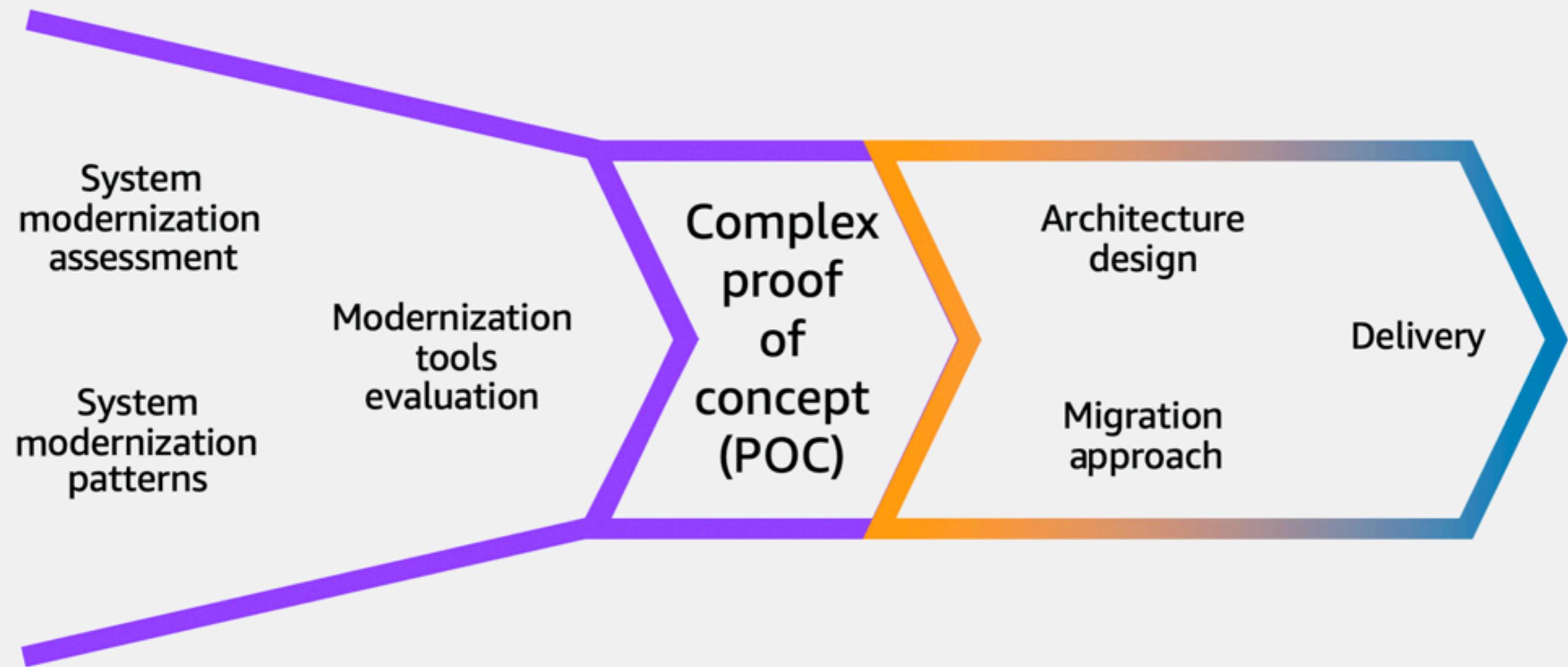
Reduce cost

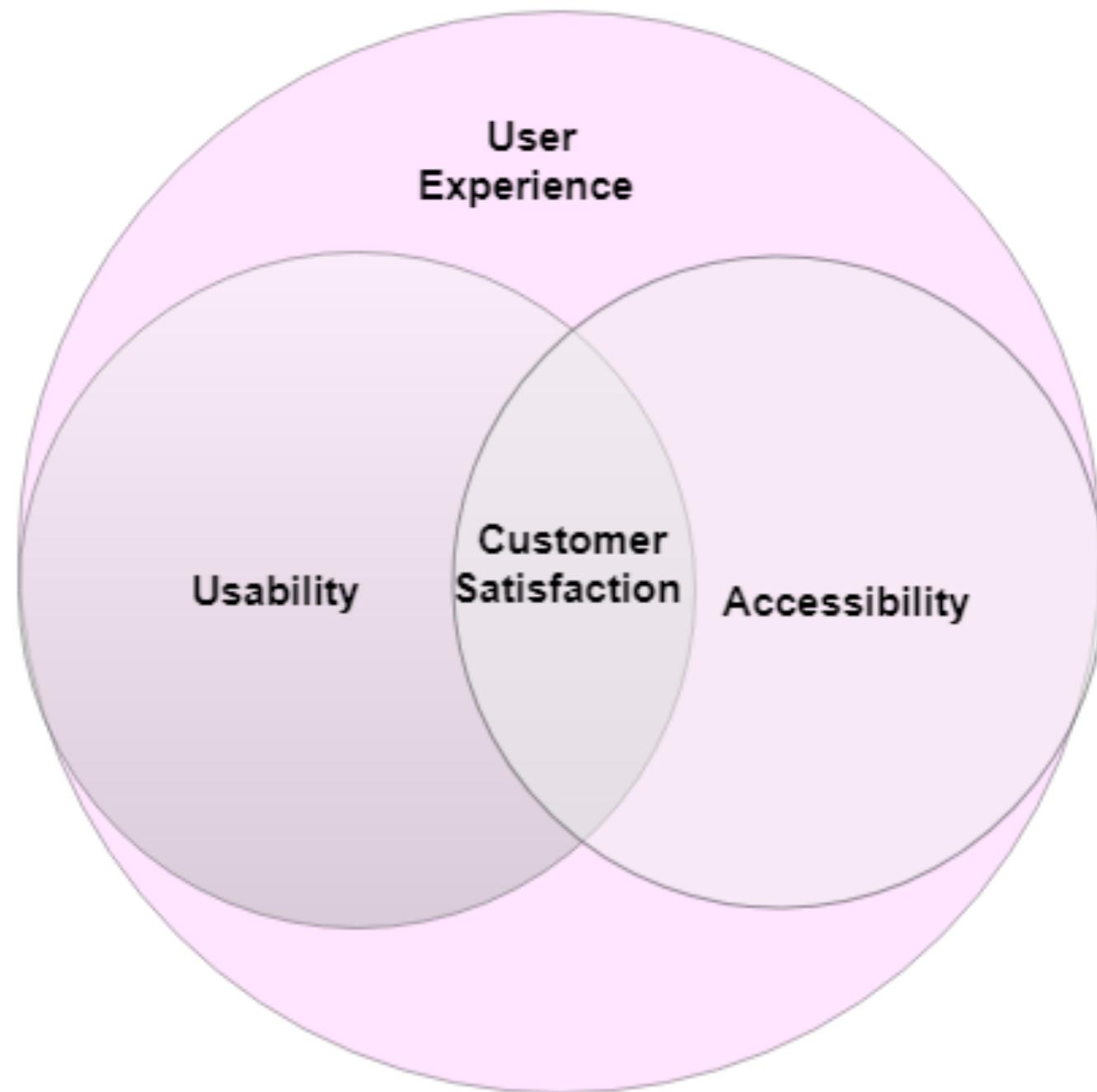


Reduce architecture complexity



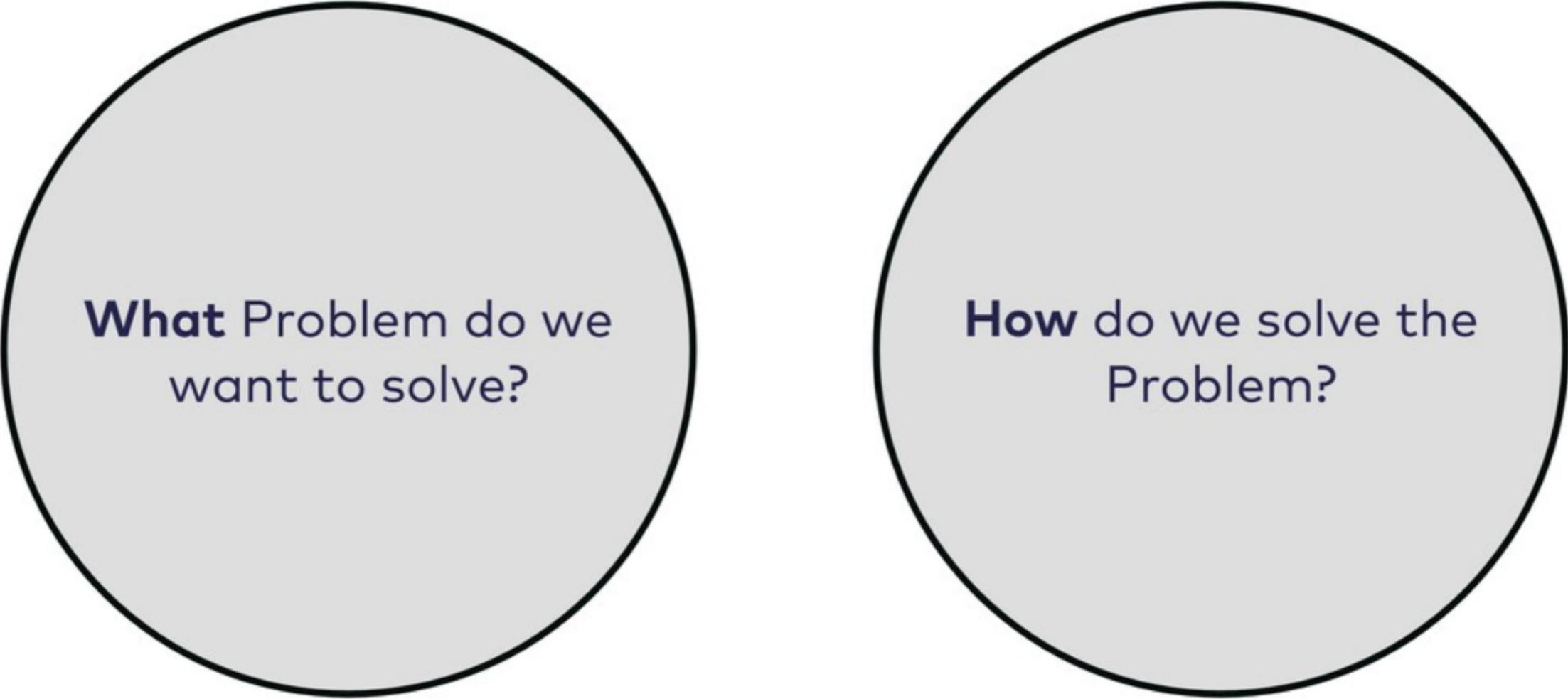
Working with Legacy





Domain Driven Design

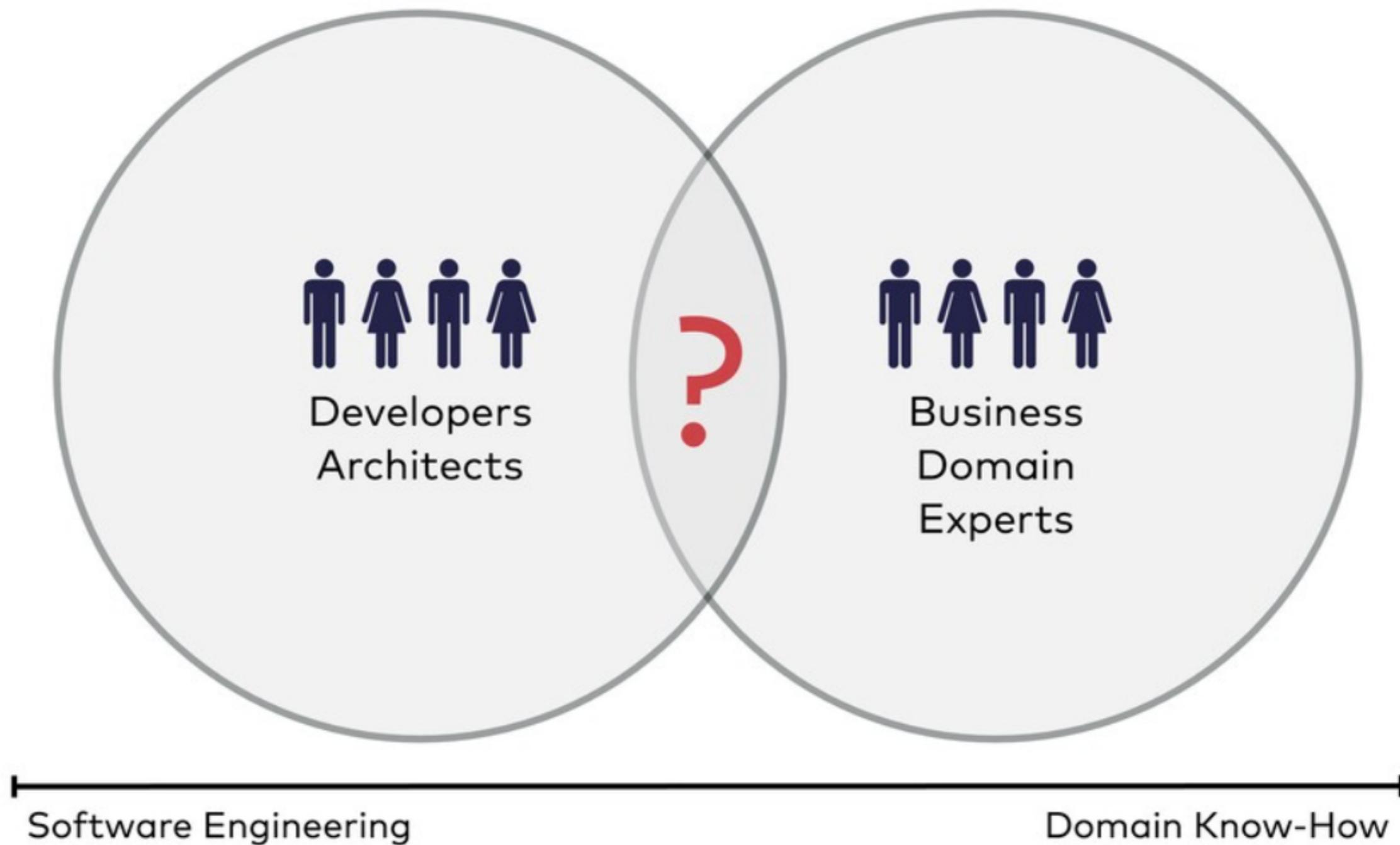




What Problem do we
want to solve?

How do we solve the
Problem?





How the business names things

Window

Painting



Trolley



TV

Chair



Desk



How the business names things

How developers name things

Window

TransparencyFactory

EntertainmentProviderSingleton

RollableStuffContainer

Trolley

TV

Painting

DecoratorImpl

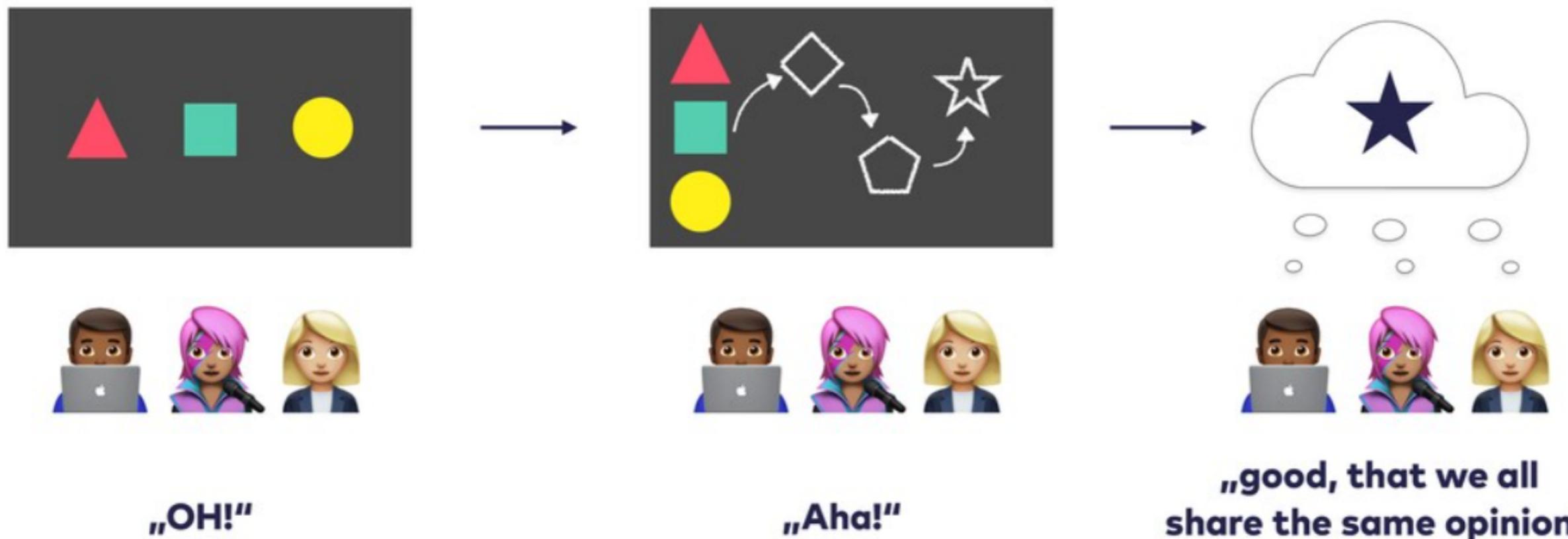
Desk

WorkEnablementDevice

RestProvider

Chair



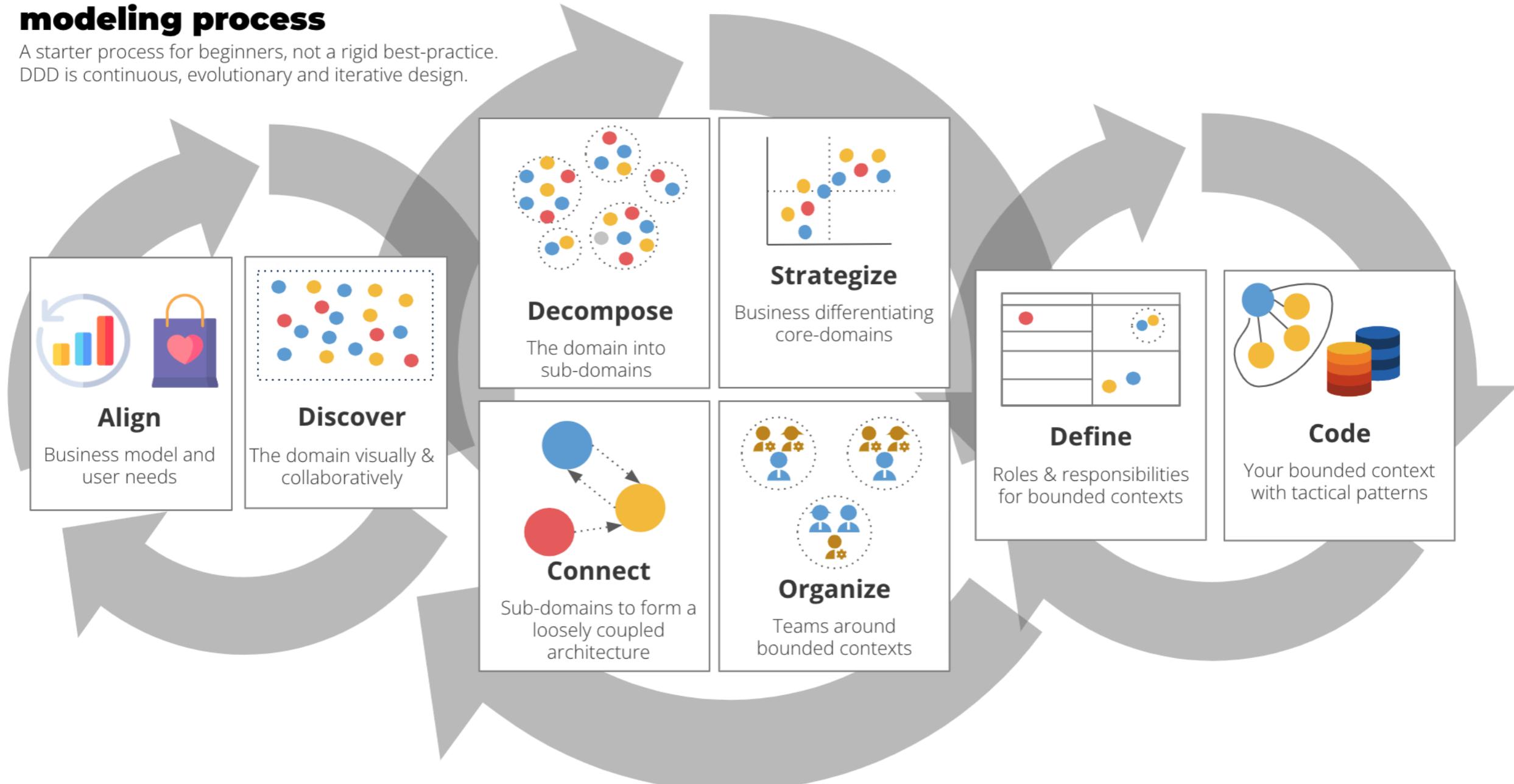


Inspired by Jeff Patton and Luke Barren



Domain-Driven Design starter modeling process

A starter process for beginners, not a rigid best-practice.
DDD is continuous, evolutionary and iterative design.



<https://github.com/ddd-crew/ddd-starter-modelling-process/tree/master>



Workshop



S.A.D

Solution Architect Document



Solution Architect

Consistent communication for successful app delivery

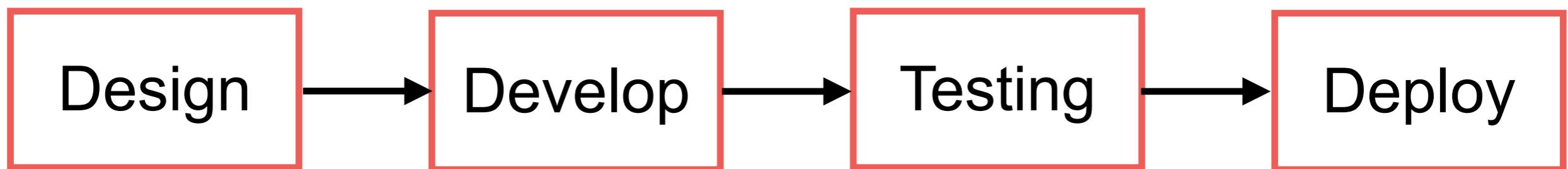
Communicate a solution design to all technical and non-technical stakeholders

Documentation !!



S.A.D

End-to-end view of application
Help everyone to be on the same view
Easy to understand for everyone



S.A.D

Define impact of solution for estimation,
planning and delivery purposes

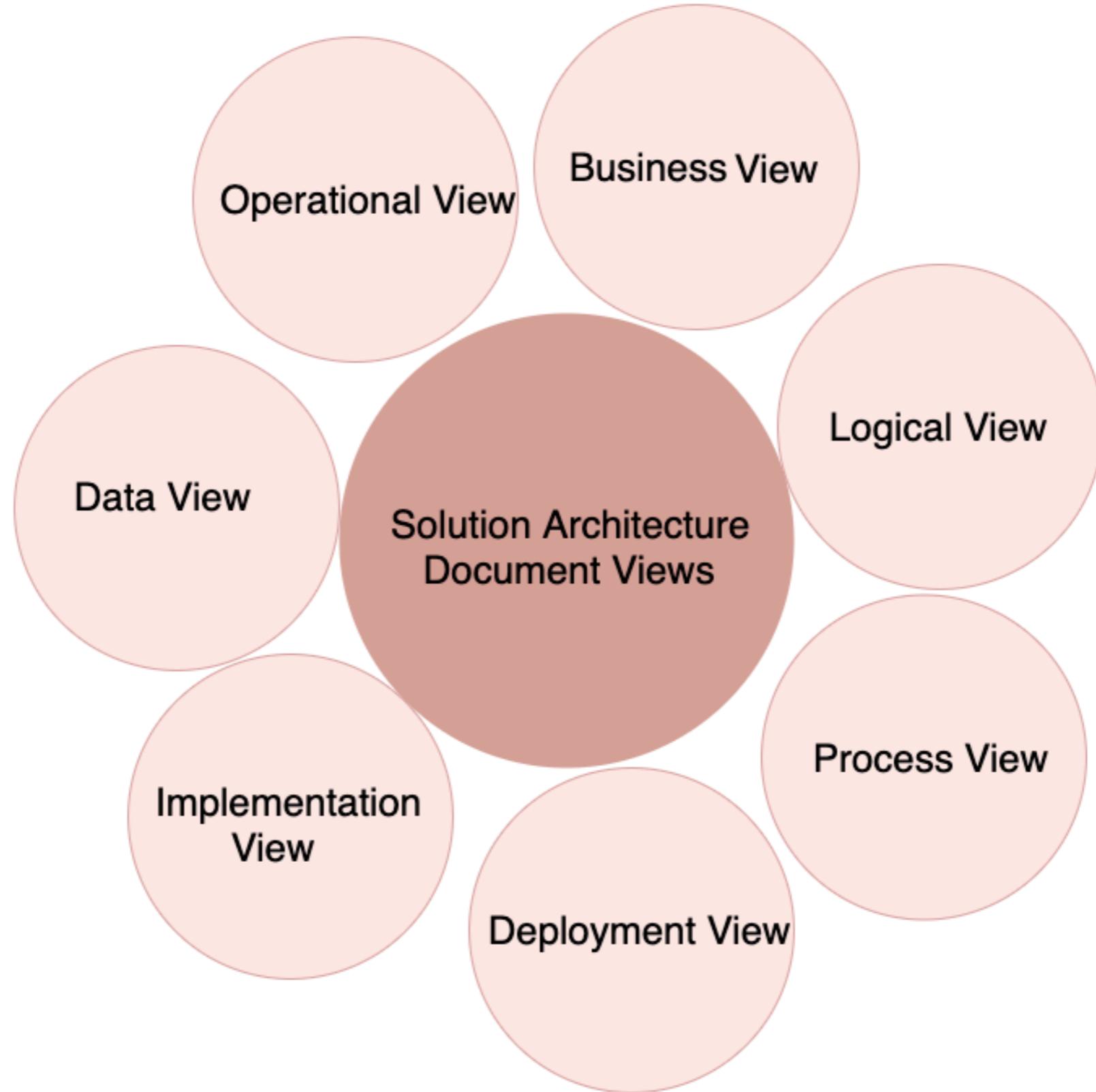
Define business process, continuation and
operation need for solution after production
launch



S.A.D

Easy to understand for everyone
Help everyone to be on the same view

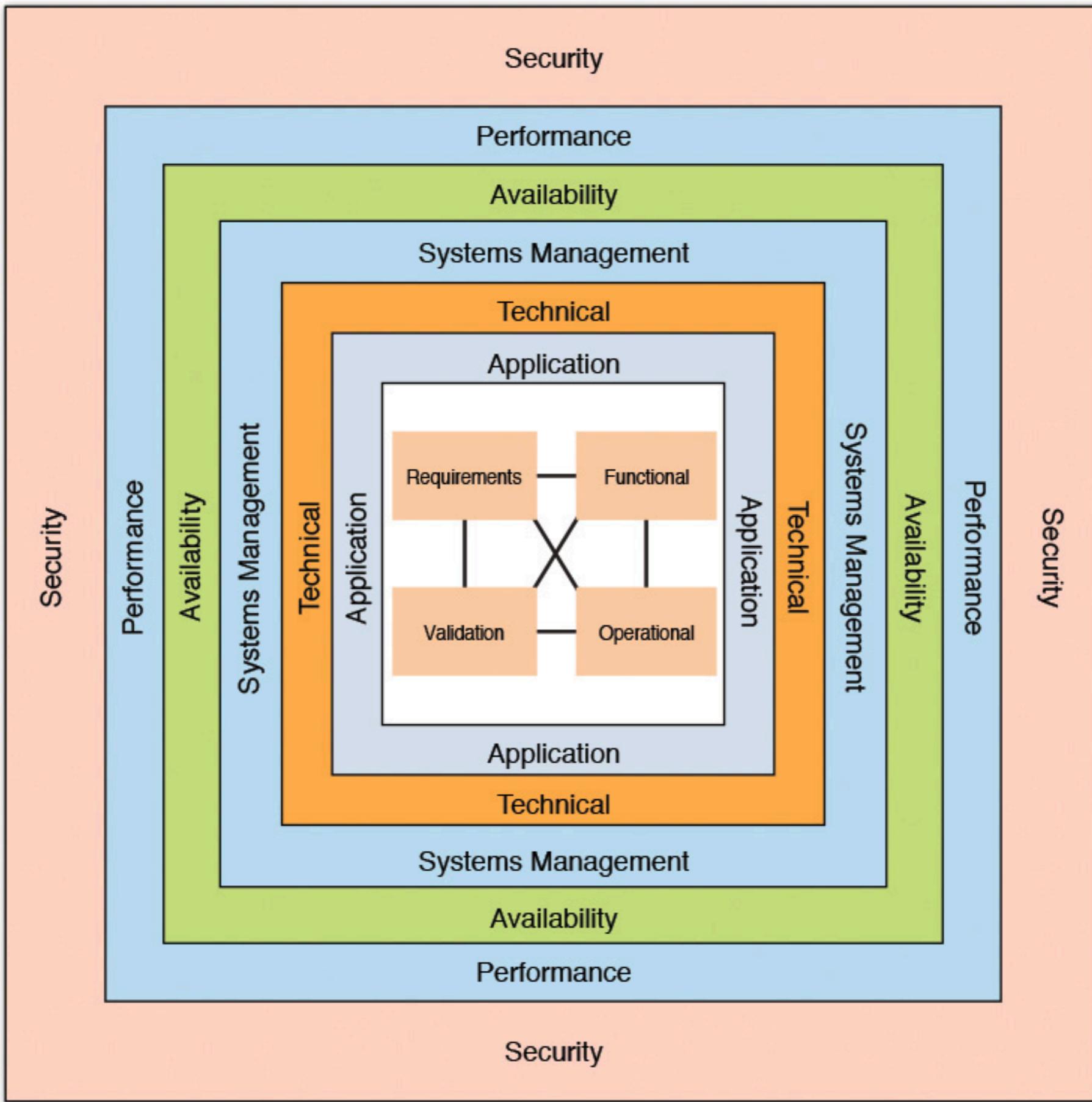




Standard Diagram

UML (Unifies Modeling Language)
Block diagram (Vision, Draw.io)
Diagram-as-a-Code

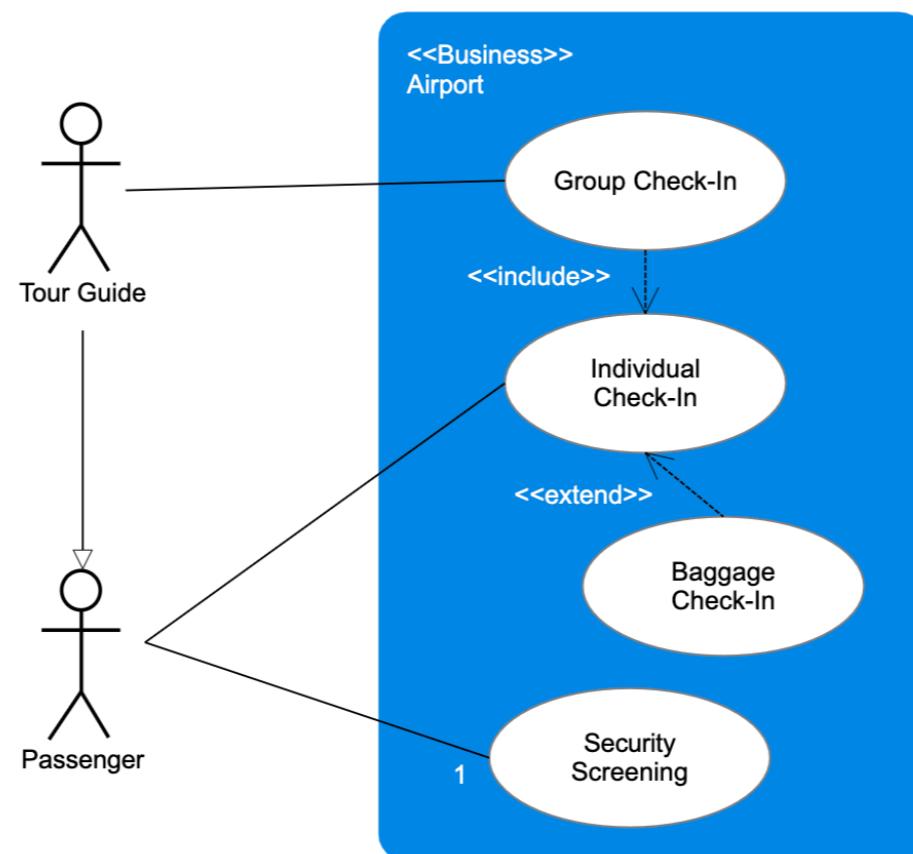




Business view

Architecture design is solving business purposes
Show value of solution and product

Use-case diagram



Logical view

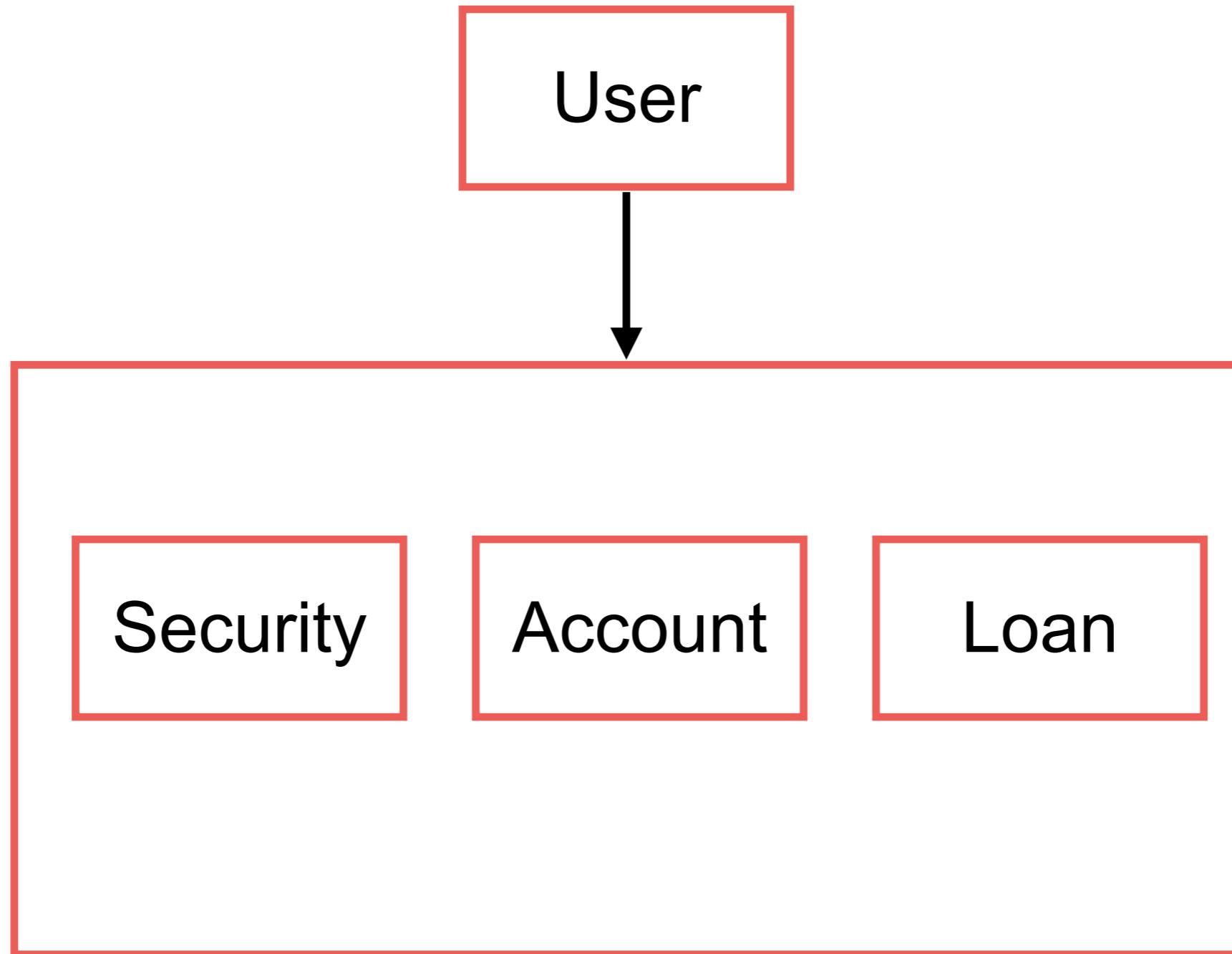
Present packages on system this business and design can understand !!

Connection between packages ?

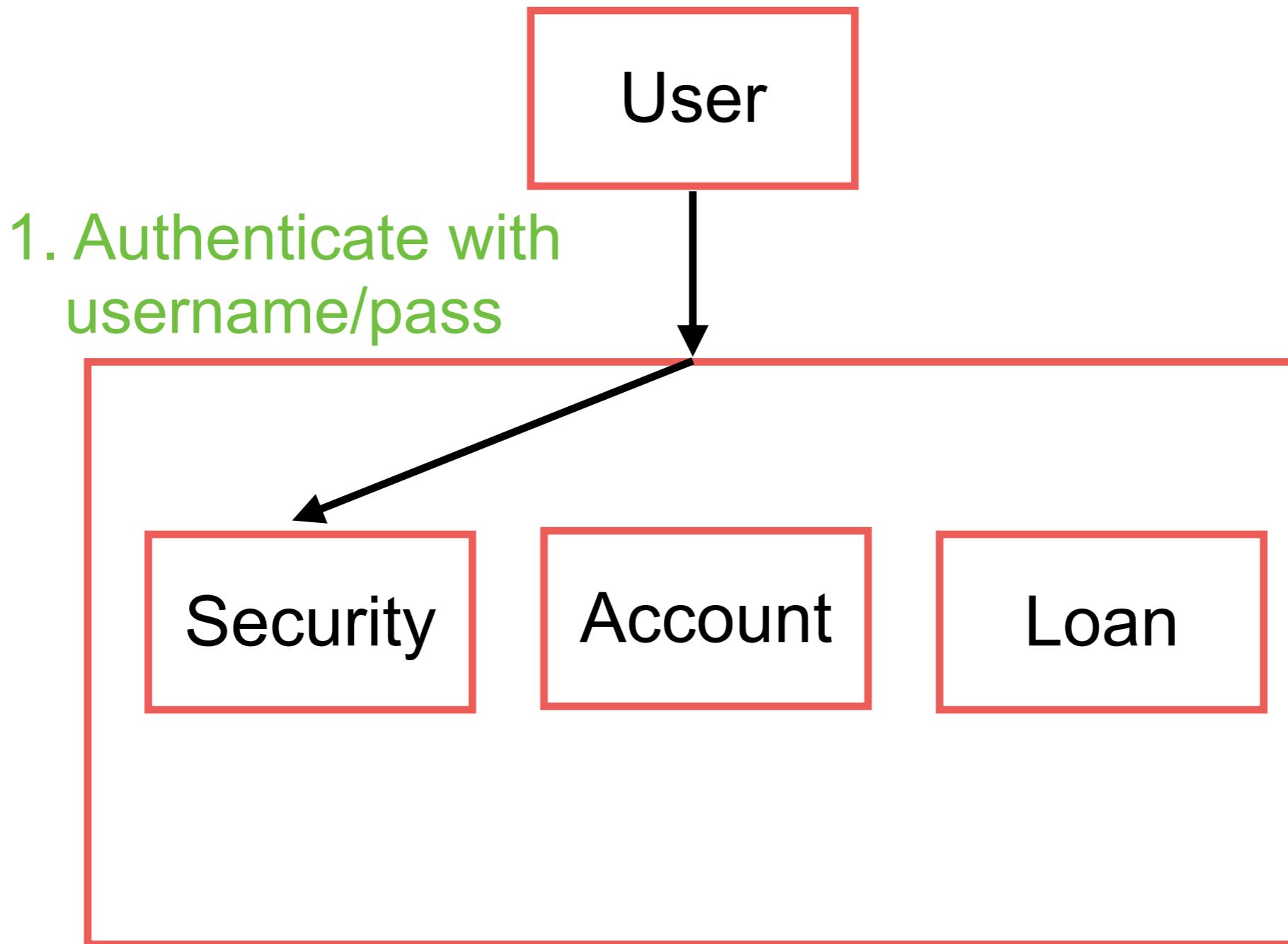
How user can interact ?



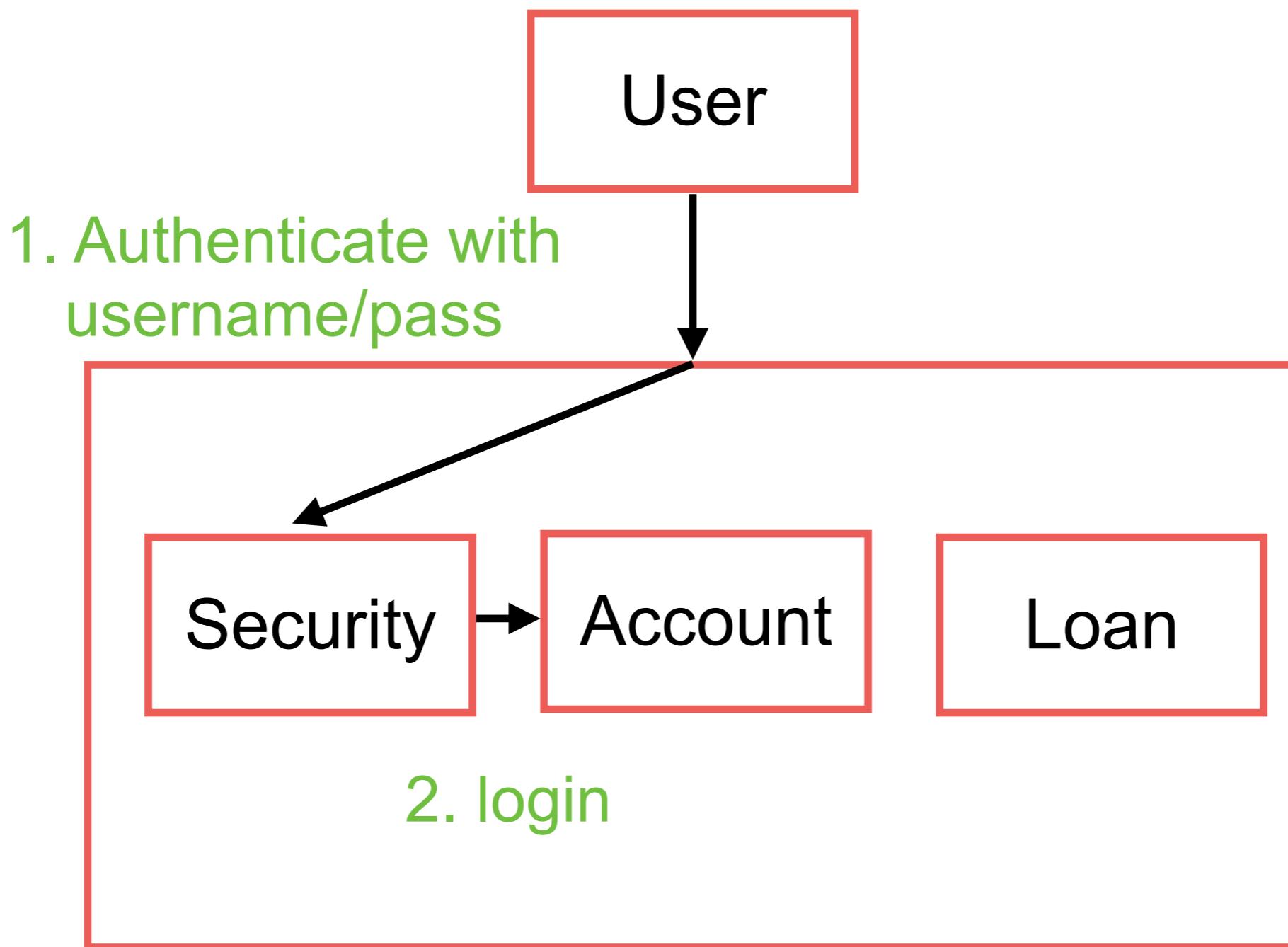
Logical view for Banking app



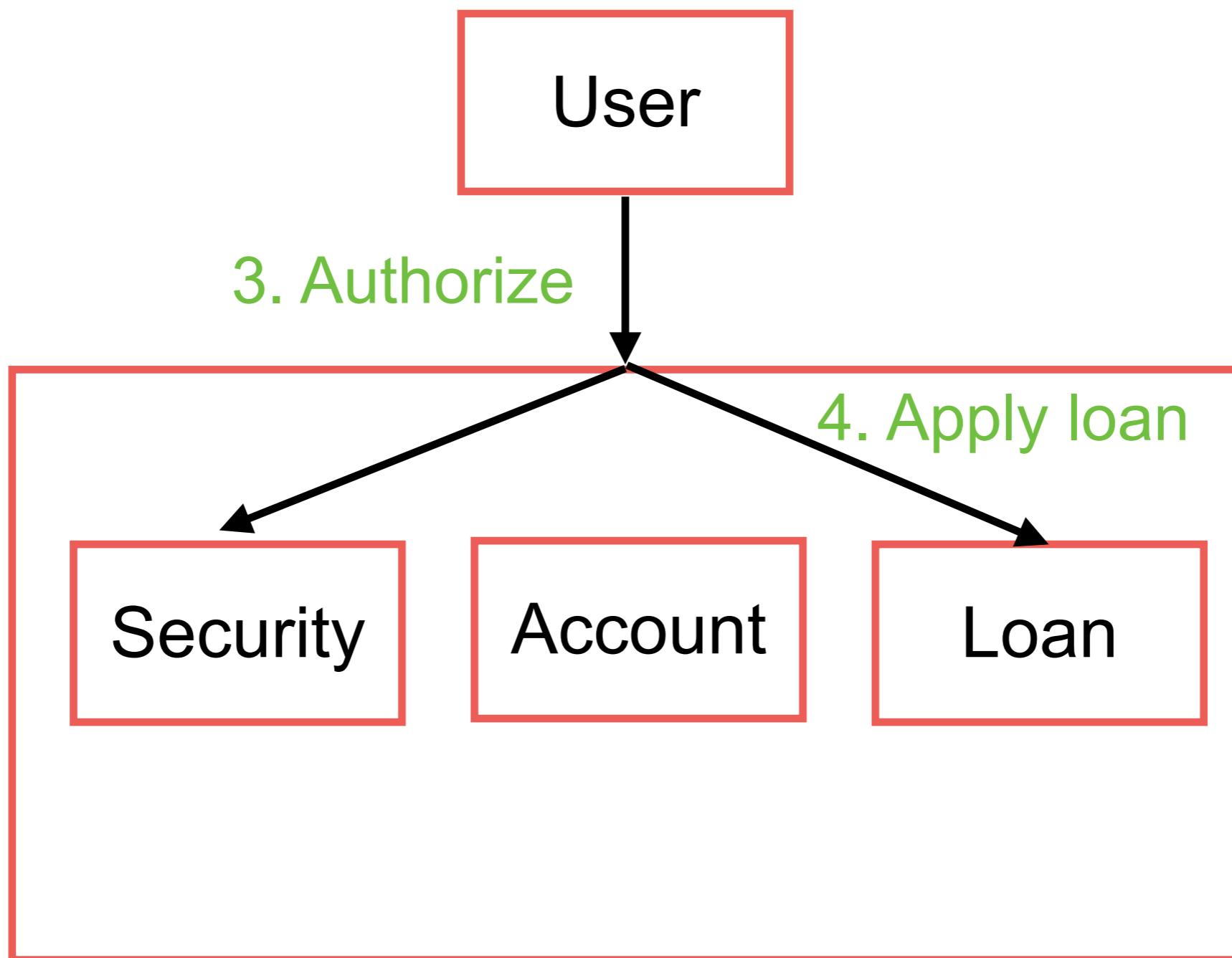
1. Need authentication process



2. Use data from Account



3. Apply for Loan

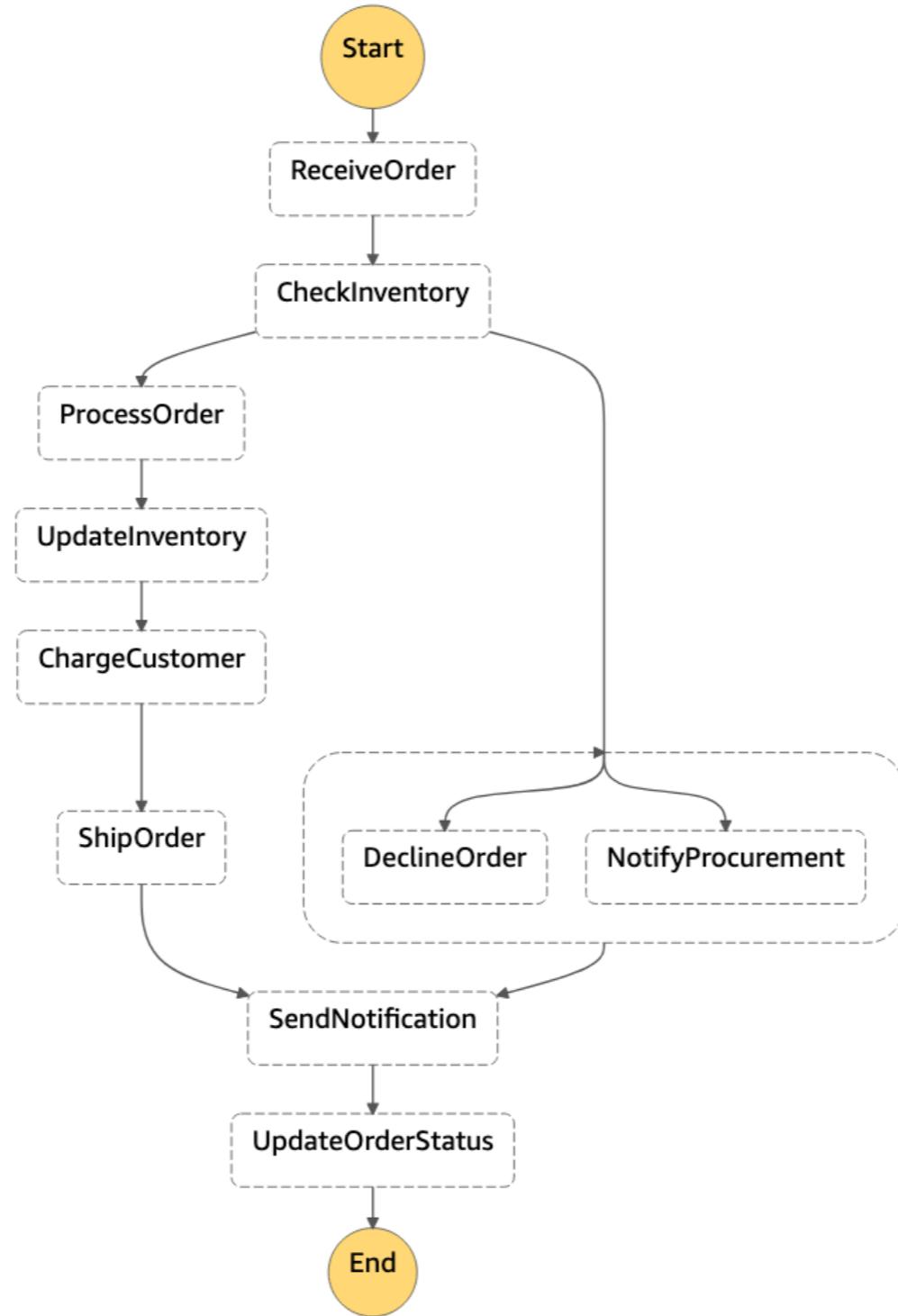


Process view

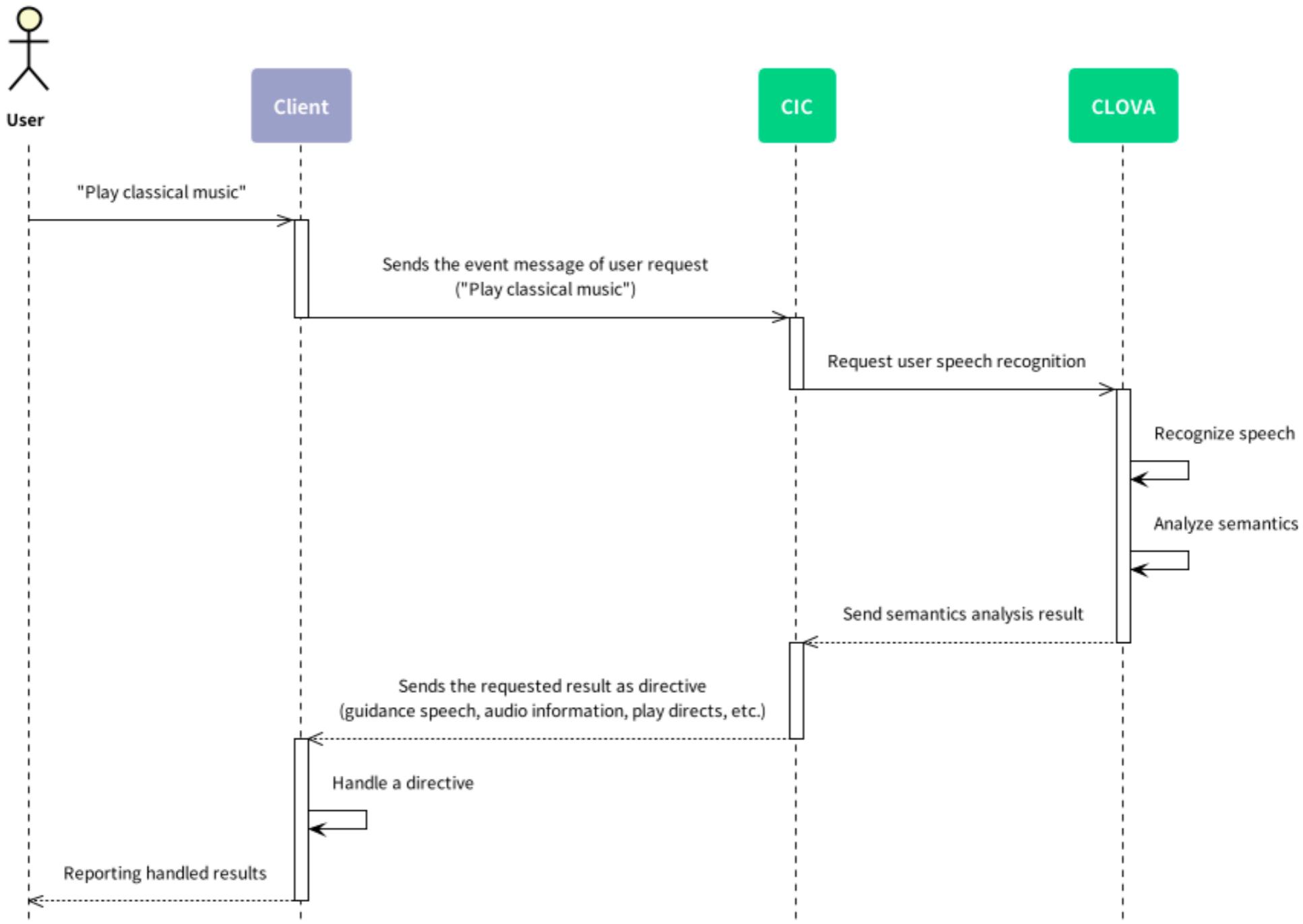
How the processes of system work together ?
Use state/sequence diagram



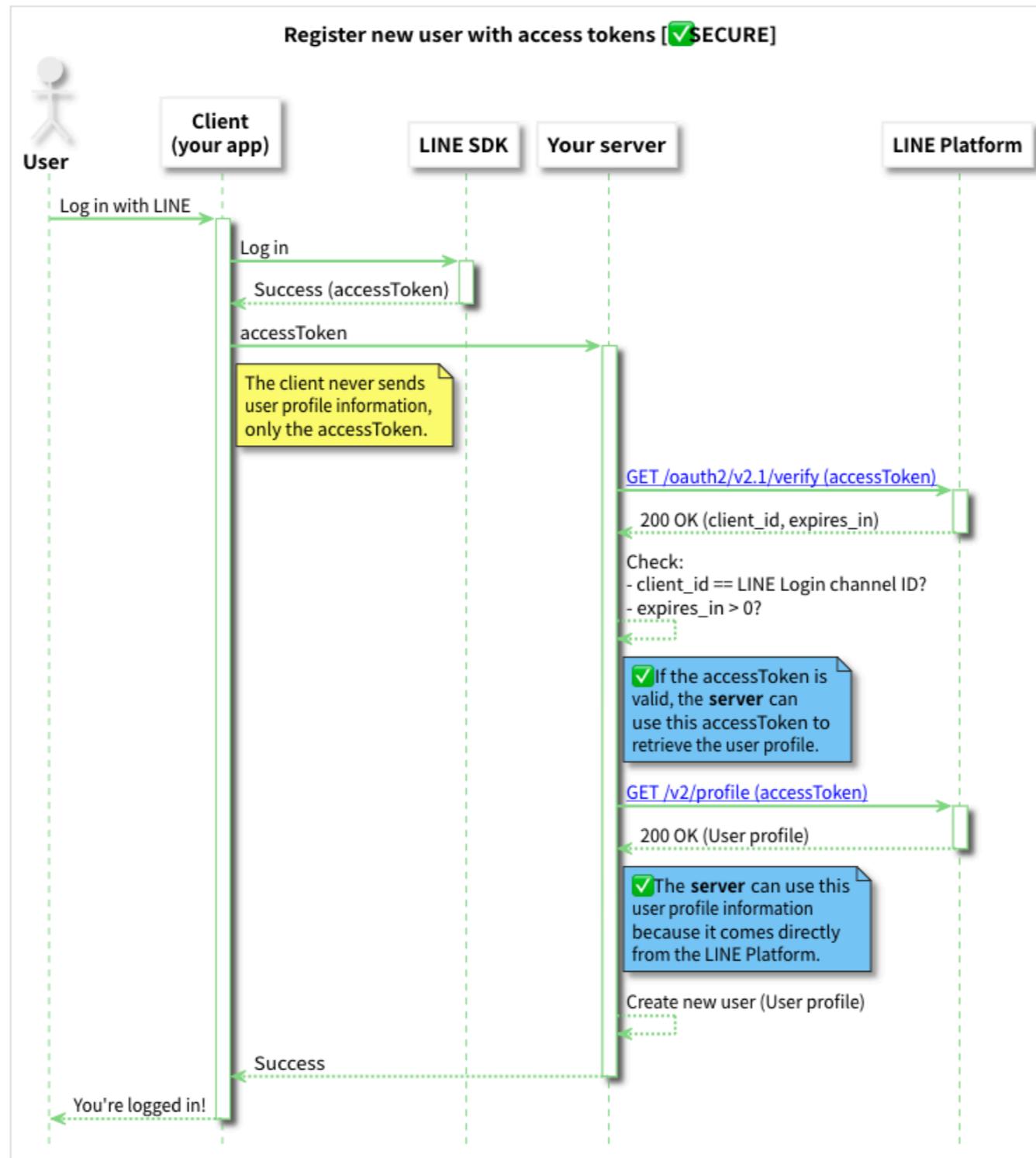
Process view (Event storming)



Sequence diagram



Sequence diagram



Deployment view

How application is going to work in Prod ?

Show all components of system are connected

Show physical layout of the system

Firewall

Load
balance

App/web
server

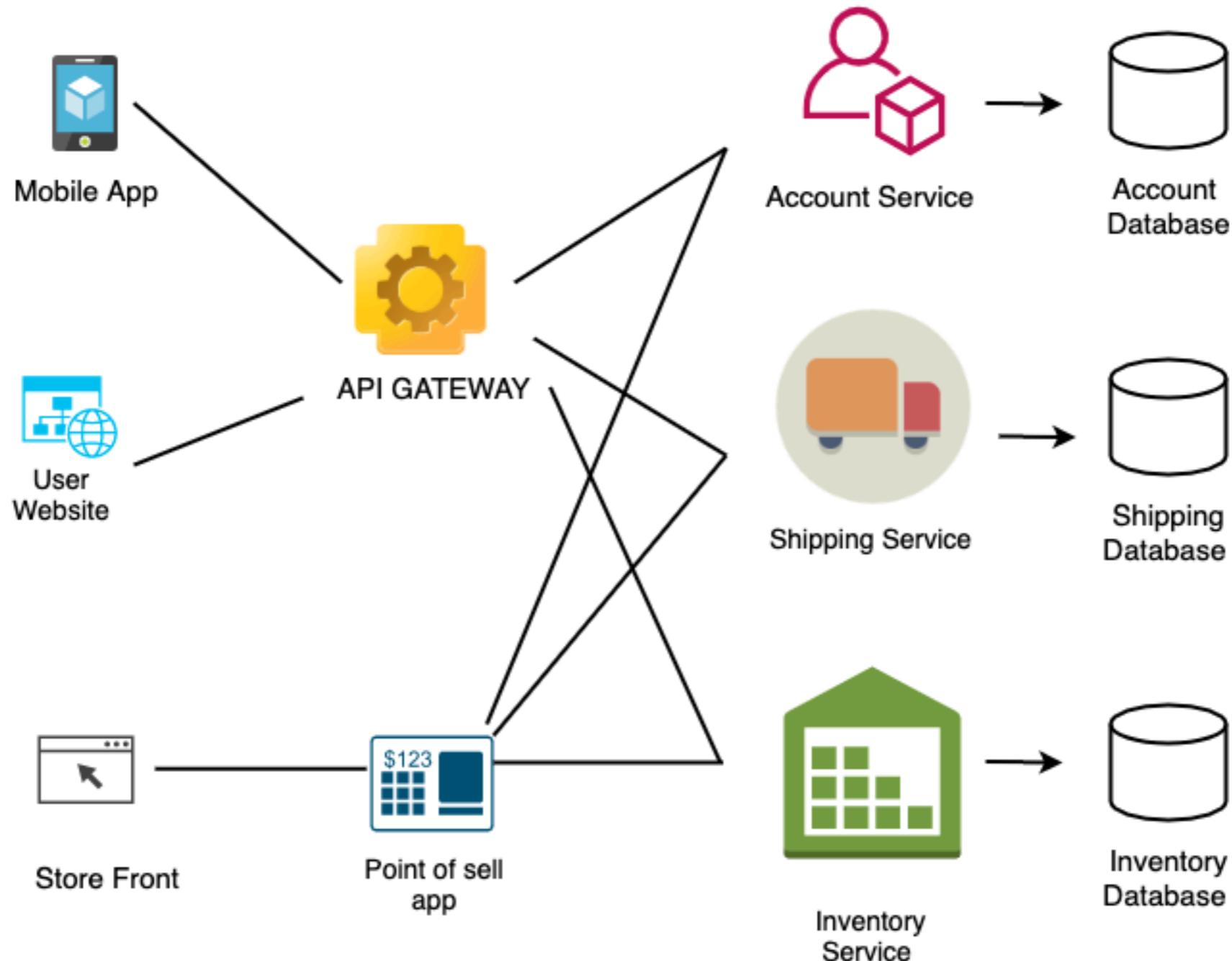
Database

API
gateway

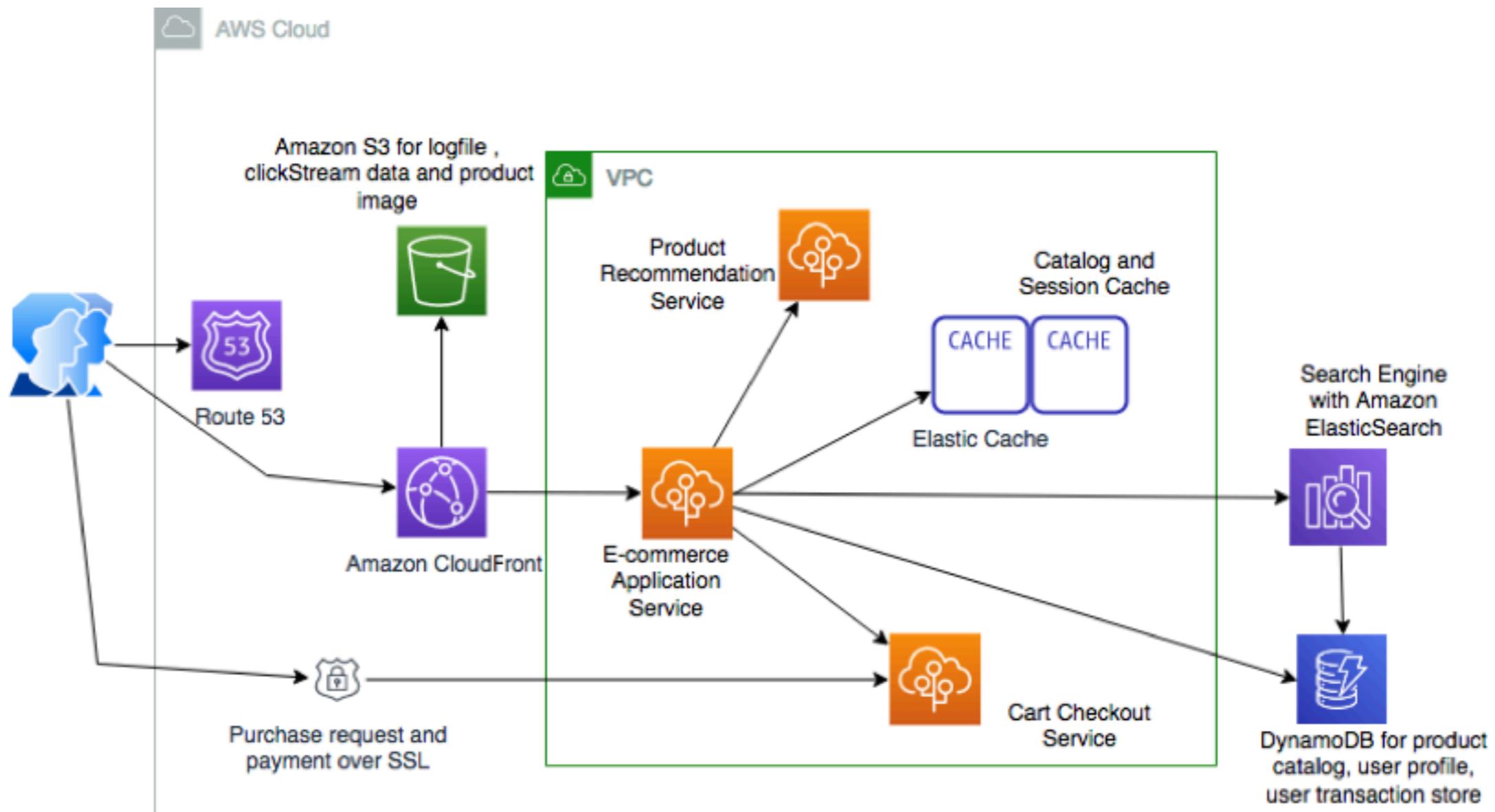
Services



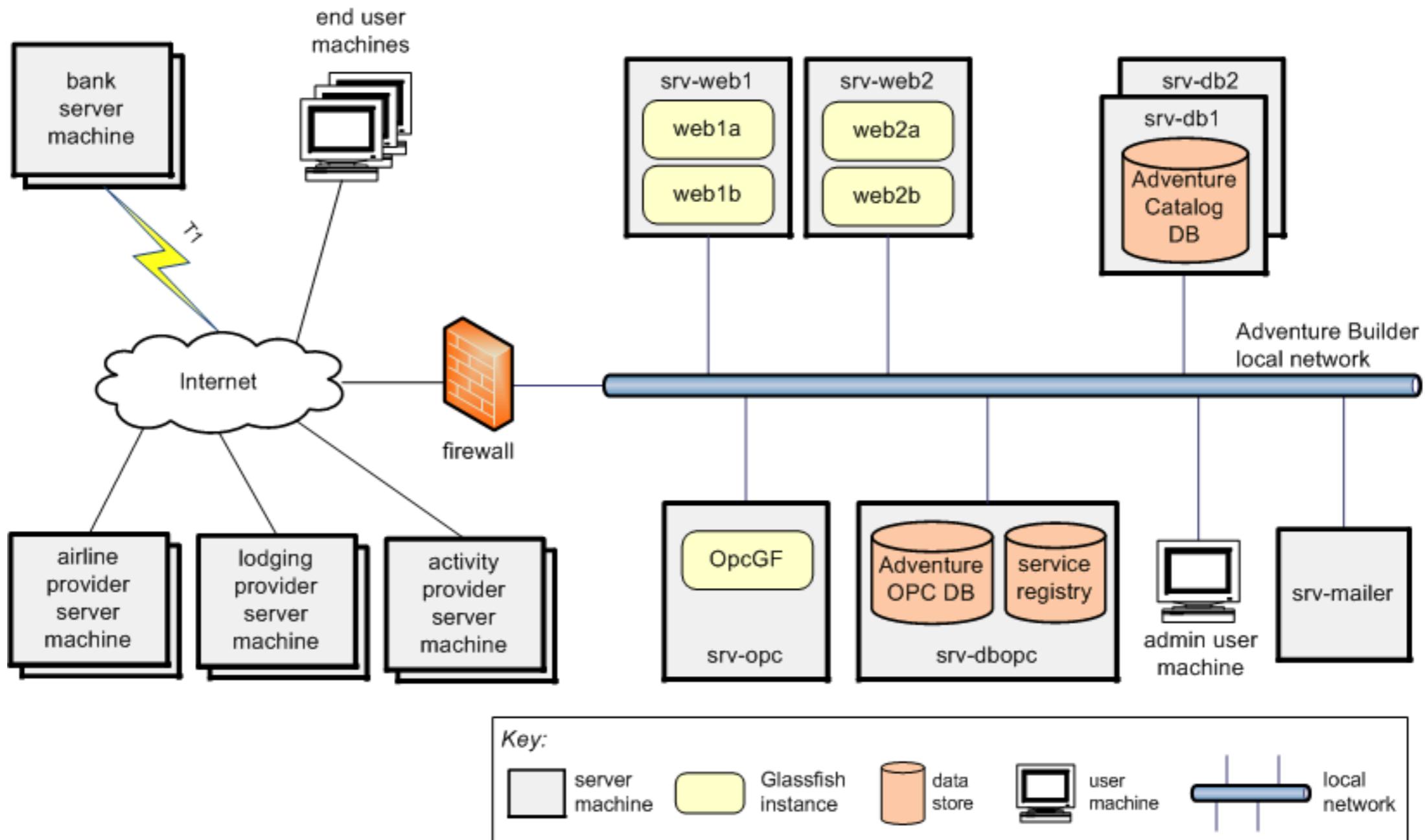
Deployment view

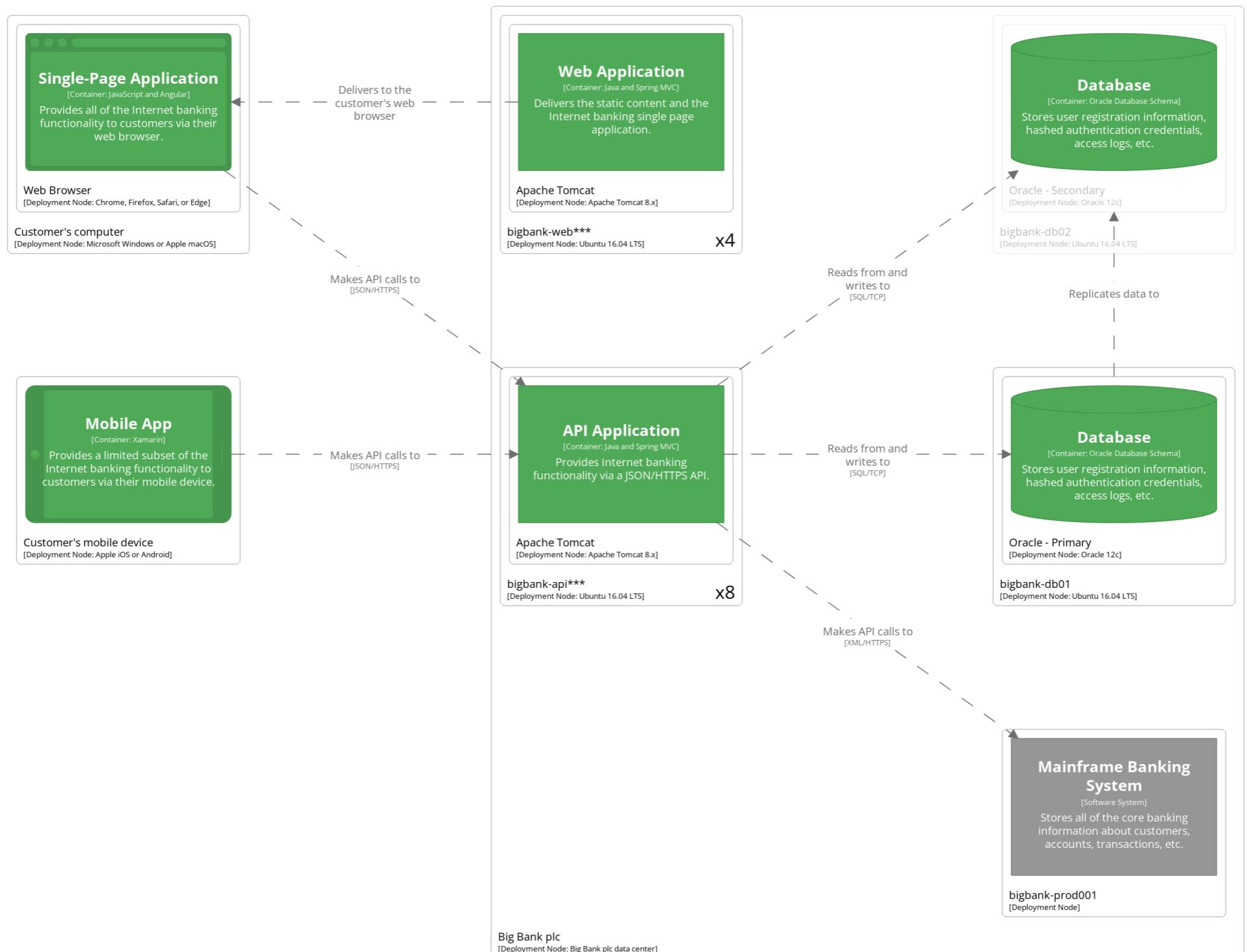


Deployment view



Deployment view



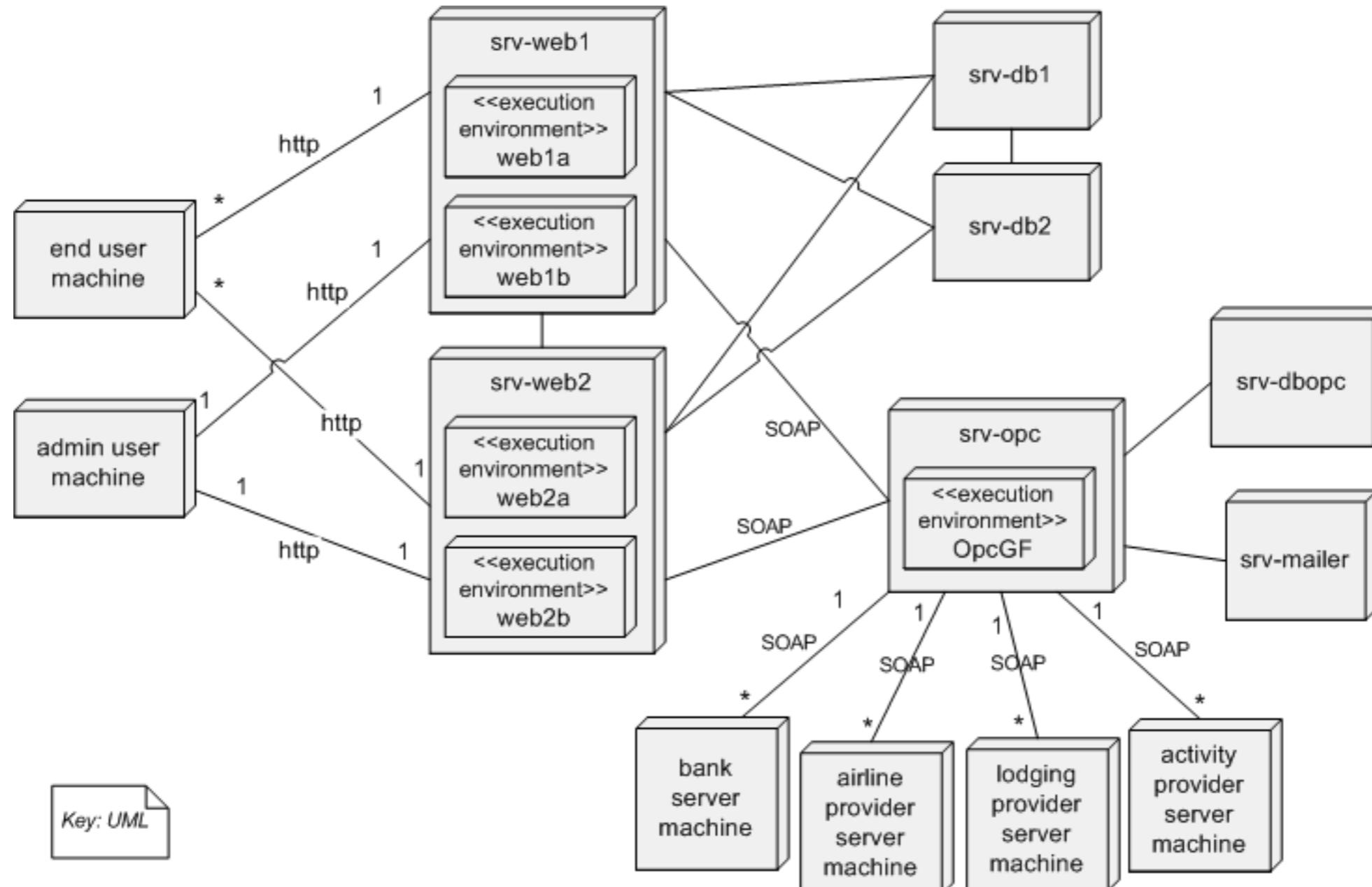


[Deployment] Internet Banking System - Live

An example live deployment scenario for the Internet Banking System - diagram created with Structurizr.
Wednesday, March 22, 2023 at 8:16 AM Coordinated Universal Time



Deployment view



Implementation view

Show architecture and technology choices
Pros and Cons

N-Tiers

Microservices

Event-driven
architecture

NodeJS

Java

Go

MySQL

MongoDB

Redis



Implementation view

Development team uses an implementation view to create a detail design



Data view

How data flow between different components ?

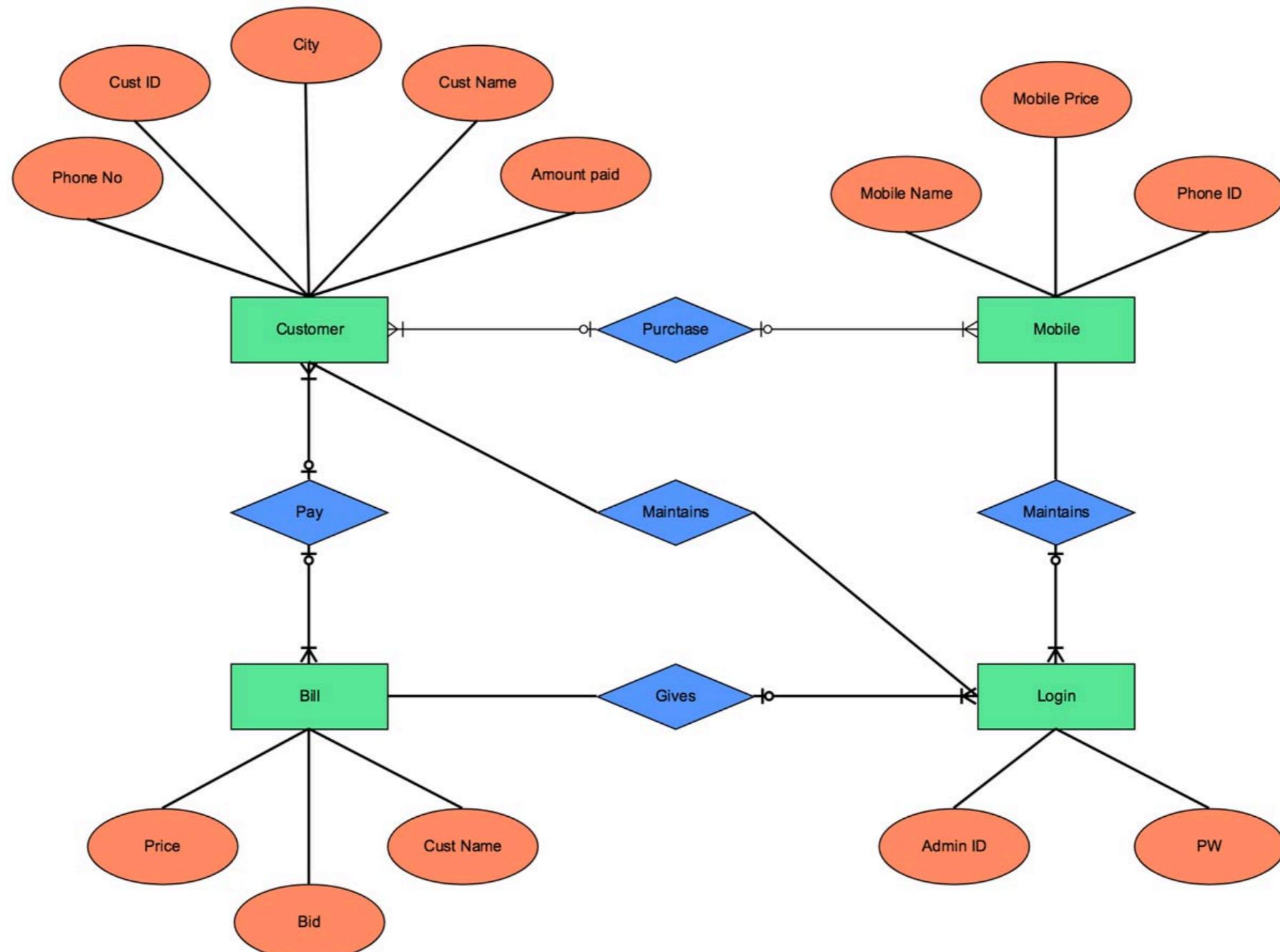
How data will stored ?

Explain data security and data integrity

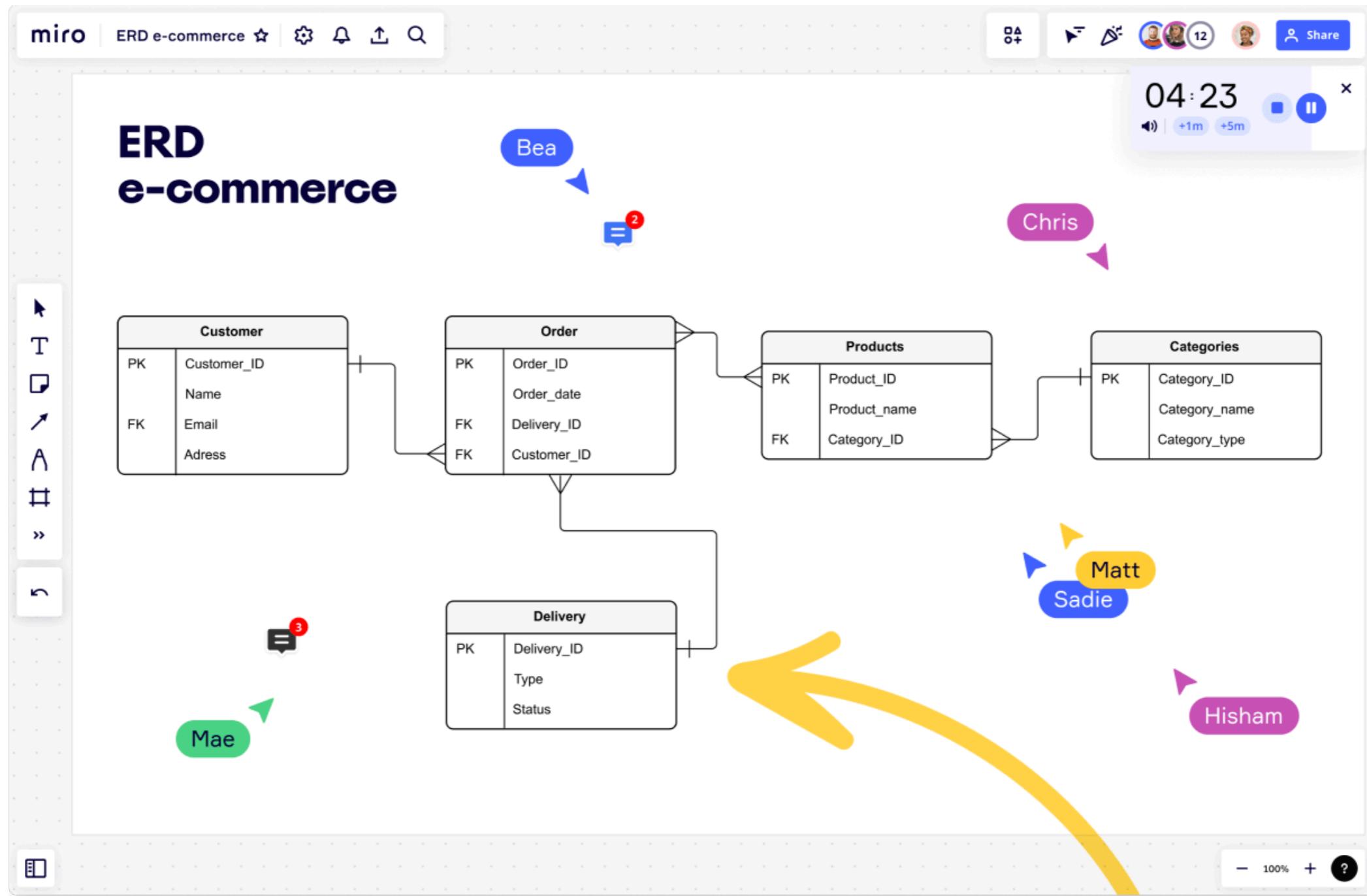
Use ER diagram (Entity Relationship)



ER diagram



ER diagram



Operation view

Post-launch operation

Define SLA (service-level agreement)

Alert and monitoring

Disaster recovery plan

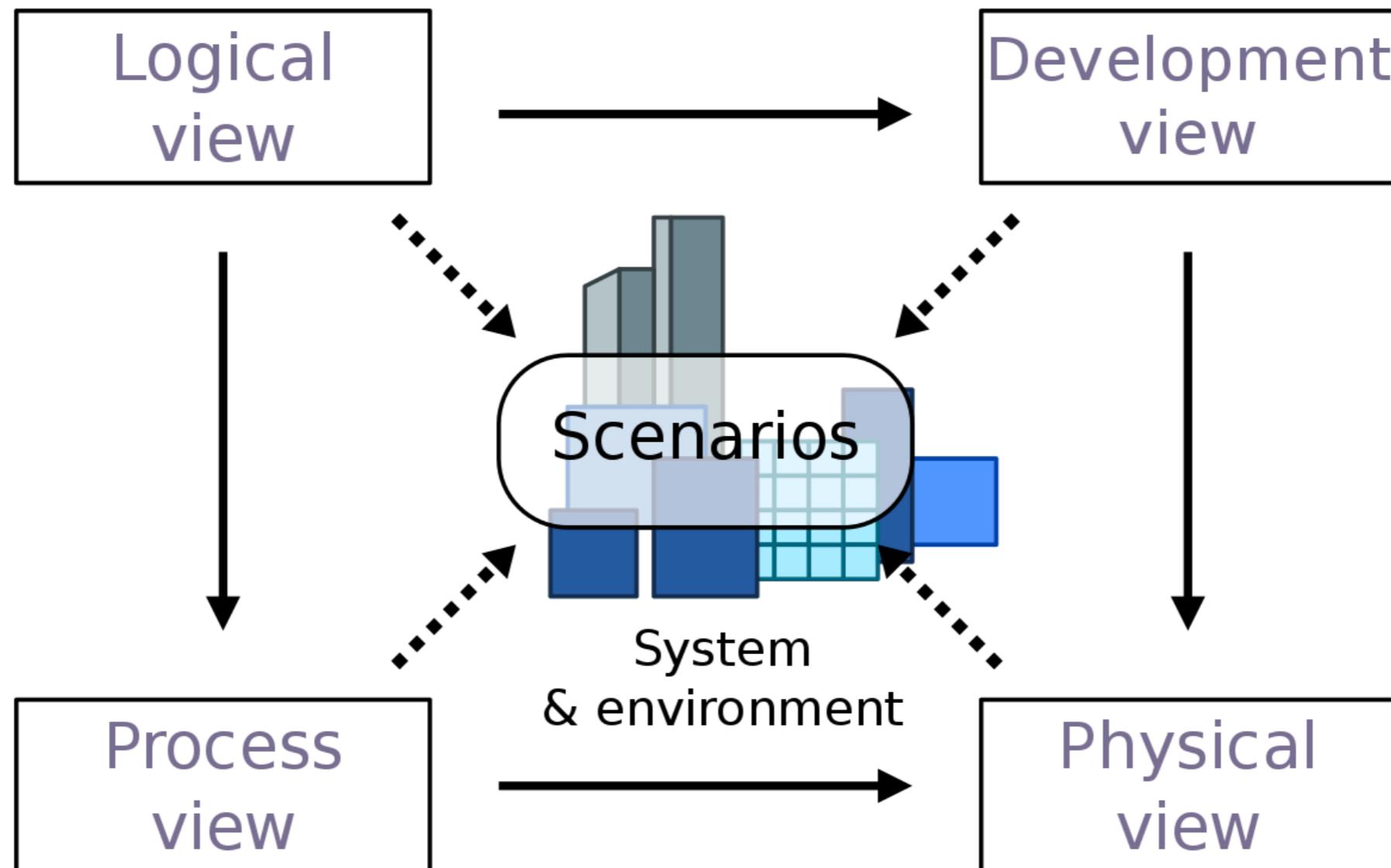


4 + 1 Architectural view model

https://en.wikipedia.org/wiki/4%2B1_architectural_view_model



4 + 1 Architectural view model



https://en.wikipedia.org/wiki/4%2B1_architectural_view_model



Working with C4 architecture modeling

<https://c4model.com/>



C4 Model

Context

Containers

Components

Code



1. System Context

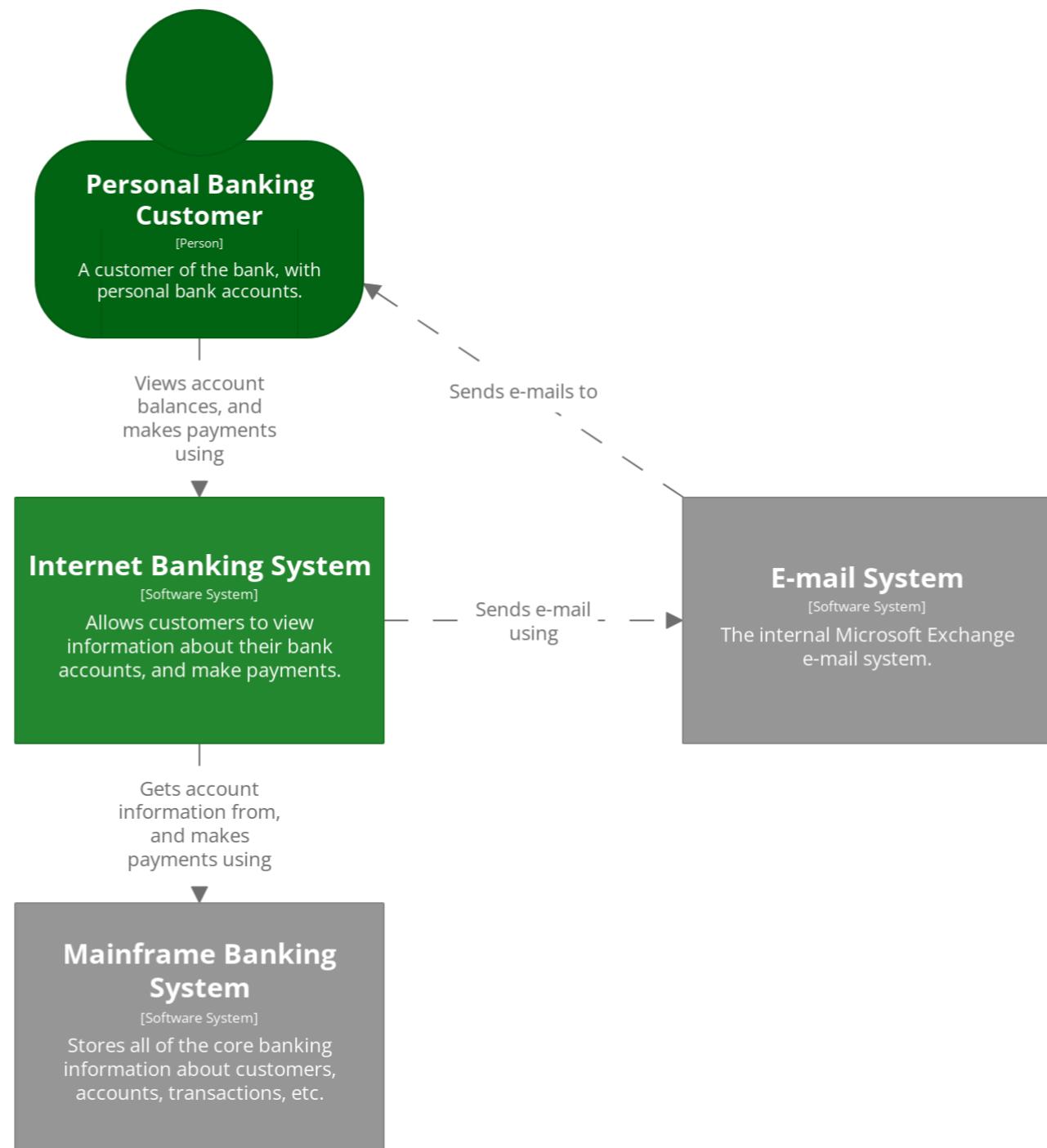
High level/big picture of system

Focus on actors/roles/personas

Communication between user and system

For non-technical stakeholders





[System Context] Internet Banking System

The system context diagram for the Internet Banking System - diagram created with Structurizr.
Wednesday, March 22, 2023 at 8:16 AM Coordinated Universal Time

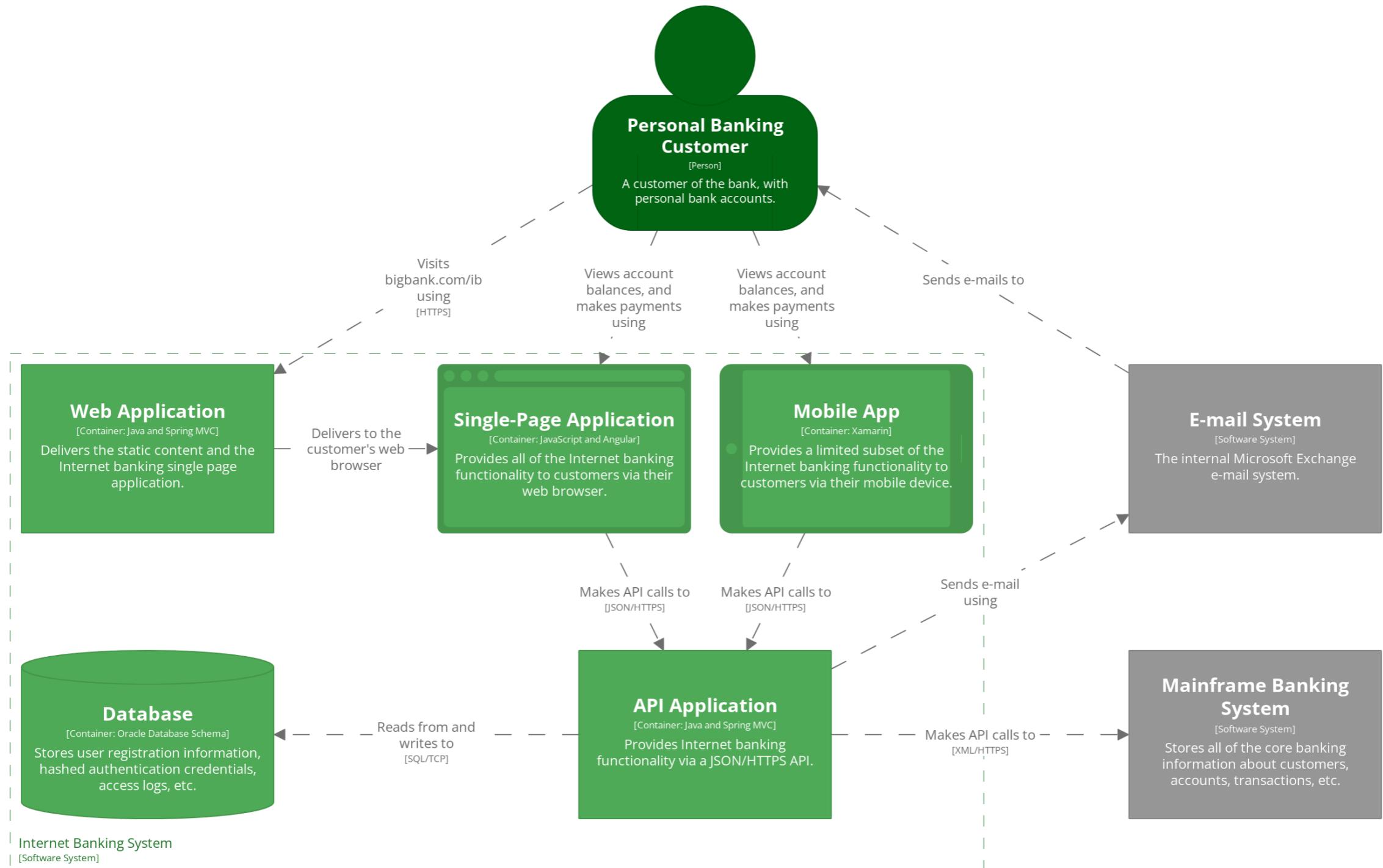


2. Containers

Separately runnable/deployable unit
Web server, SPA, Mobile app, database
Show technology choices
Communication between containers !!

Useful for software developer and operations





[Container] Internet Banking System

The container diagram for the Internet Banking System - diagram created with Structurizr.
Wednesday, March 22, 2023 at 8:16 AM Coordinated Universal Time



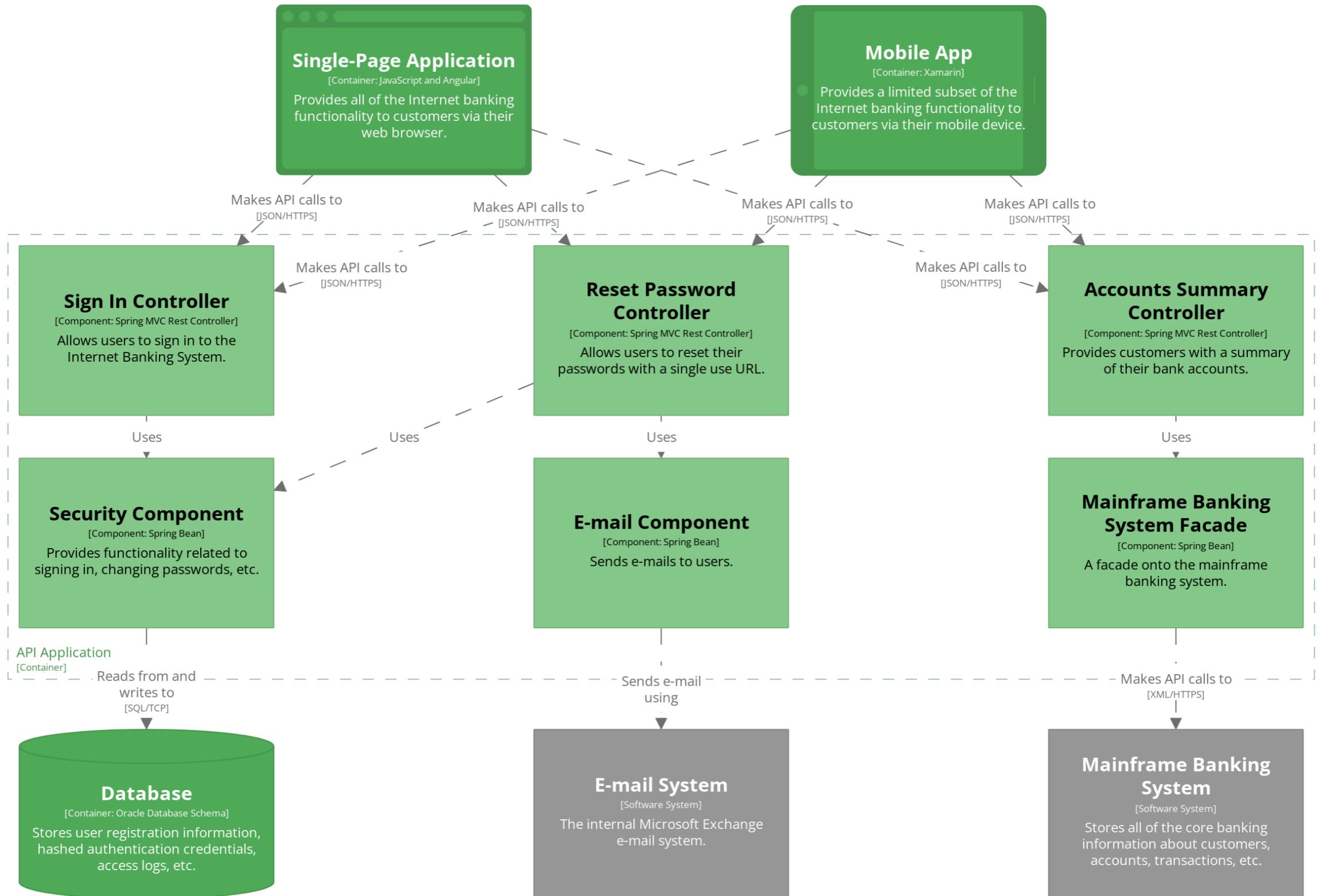
3. Components

Detail in each container

Collaboration between architect and developer

Responsibility and technology/implementation detail





[Component] Internet Banking System - API Application

The component diagram for the API Application - diagram created with Structurizr.

Wednesday, March 22, 2023 at 8:16 AM Coordinated Universal Time



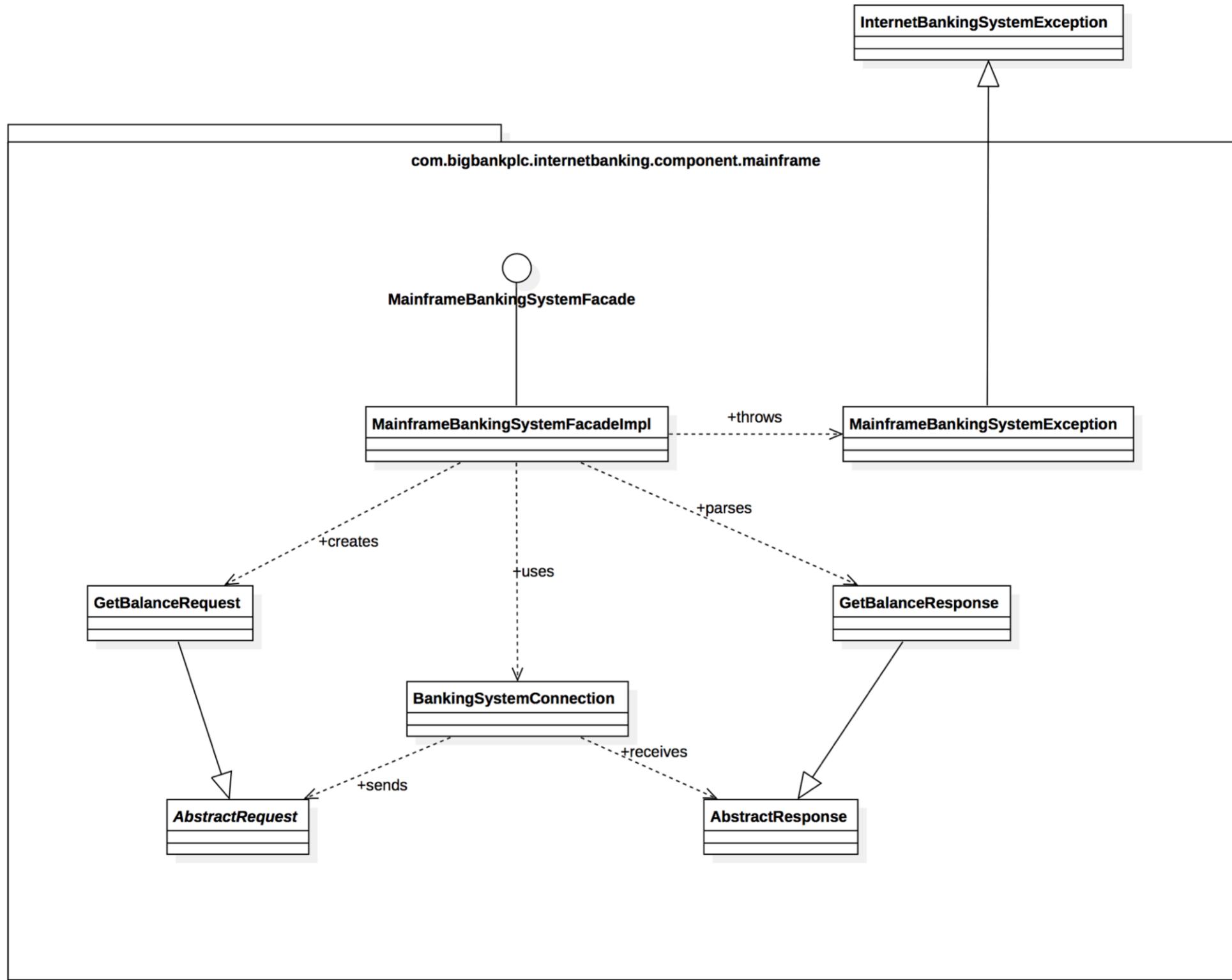
4. Code

How to implement/code in each component ?

UML => class diagram

Database => ER diagram





Design-first vs Code-first ?



C4 Model



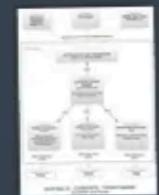
System Context

The system plus users
and system dependencies



Containers

The overall shape of the architecture
and technology choices



Components

Logical components and their
interactions within a container



Classes

Component or pattern
implementation details

**Overview
first**

**Zoom and
filter**

**Details
on demand**



Workshop with C4 Model



Food delivery



Food delivery

สร้าง food delivery app สำหรับ platform ต่าง ๆ
โดยมี backend service ต่าง ๆ เพื่อสนับสนุน



Food delivery use case !!

ลูกค้าทำการสั่งซื้้อาหารจากร้านที่เลือก



1. Context

ลูกค้าทำการสั่งซื้ออาหารจากร้านที่เลือก



2. Containers

ระบบ Food delivery app ประกอบไปด้วย

Web app

ReactJS

Android
app

Kotlin

iOS app

Swift



2. Containers

ระบบ backend ประกอบไปด้วย

Backend for
Frontend

Go

Backend

Java

Database

MongoDB

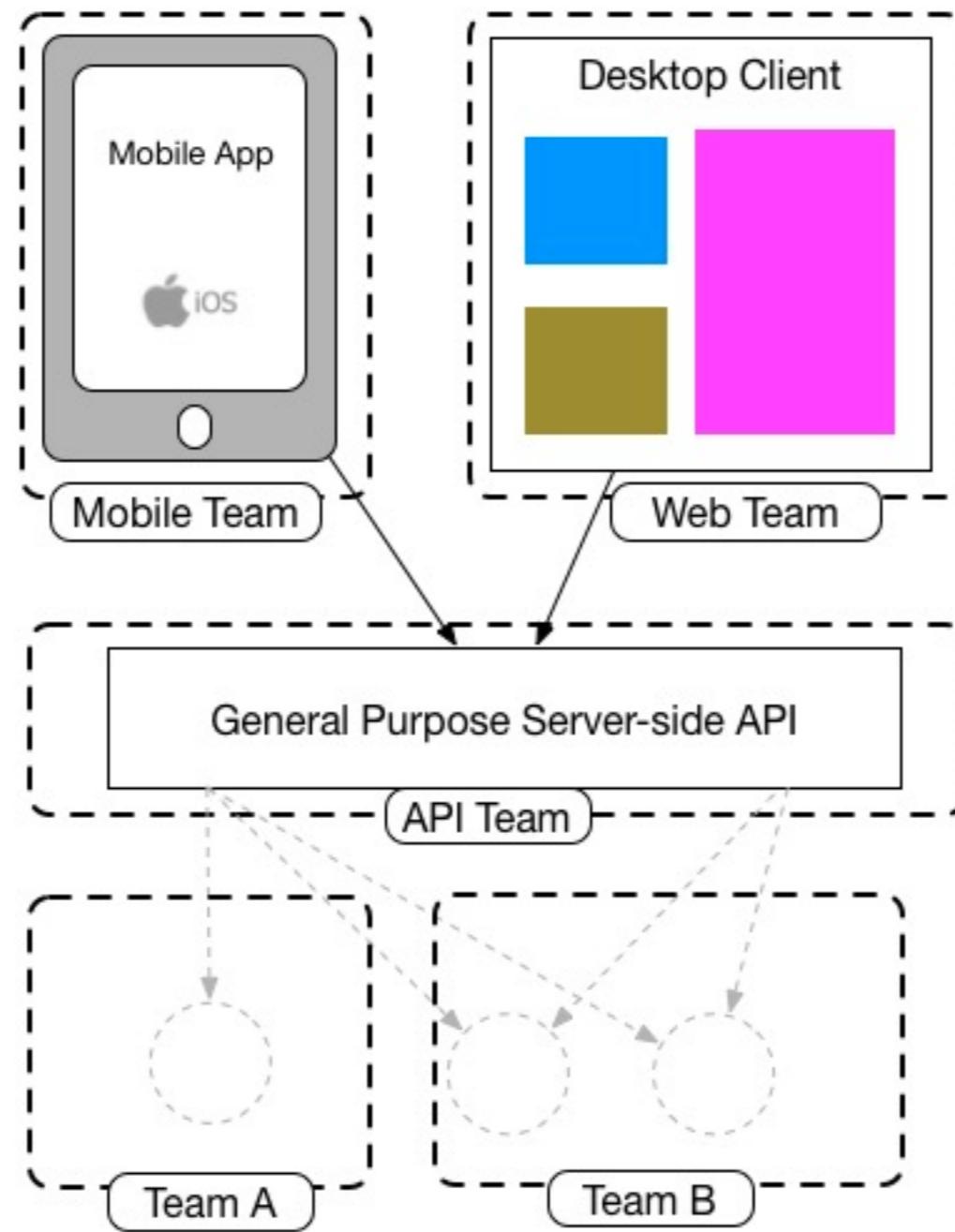


Backend for Frontend (BFF)

Aggregation data layer
Provide authorization process



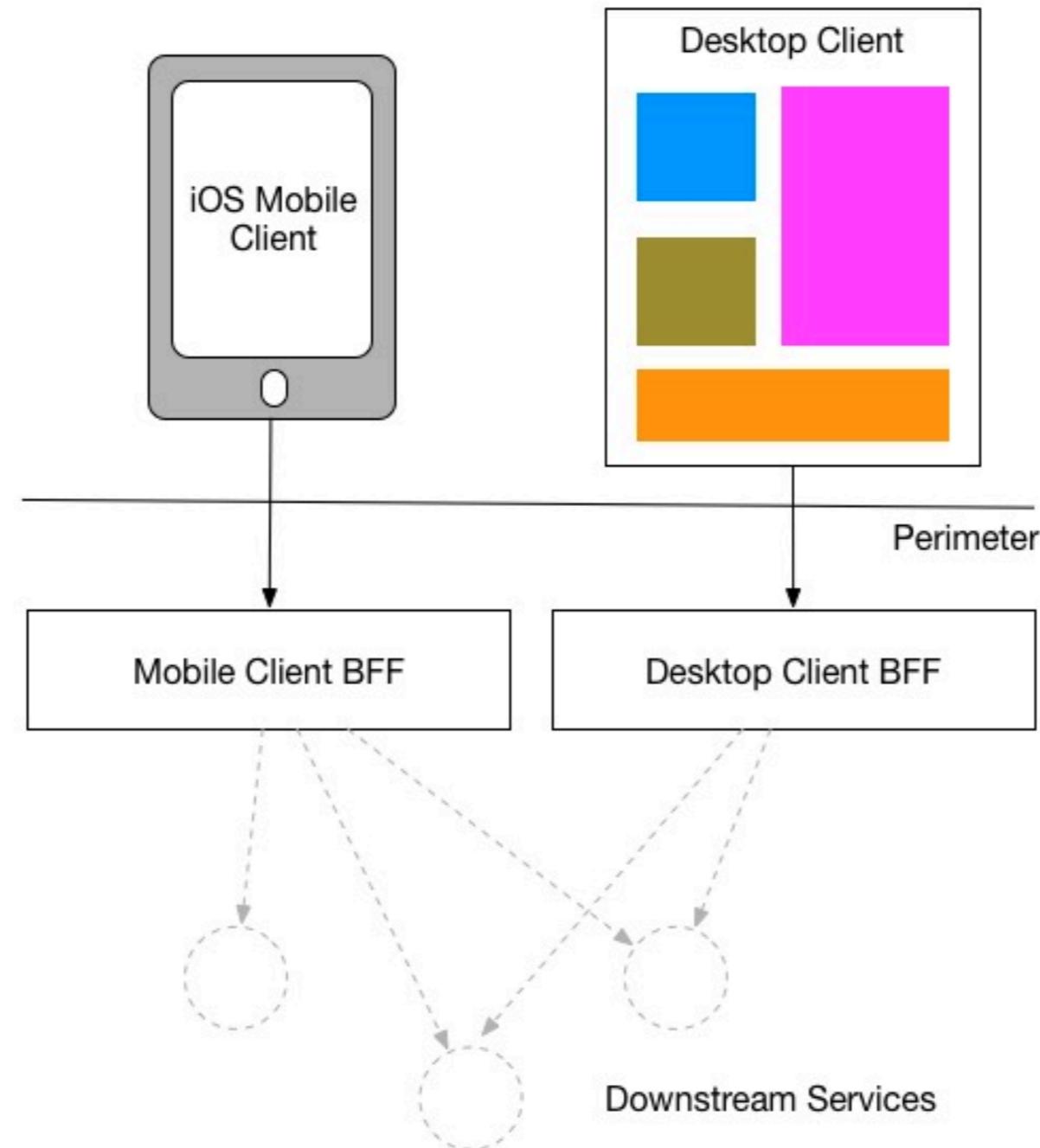
Backend for Frontend (BFF)



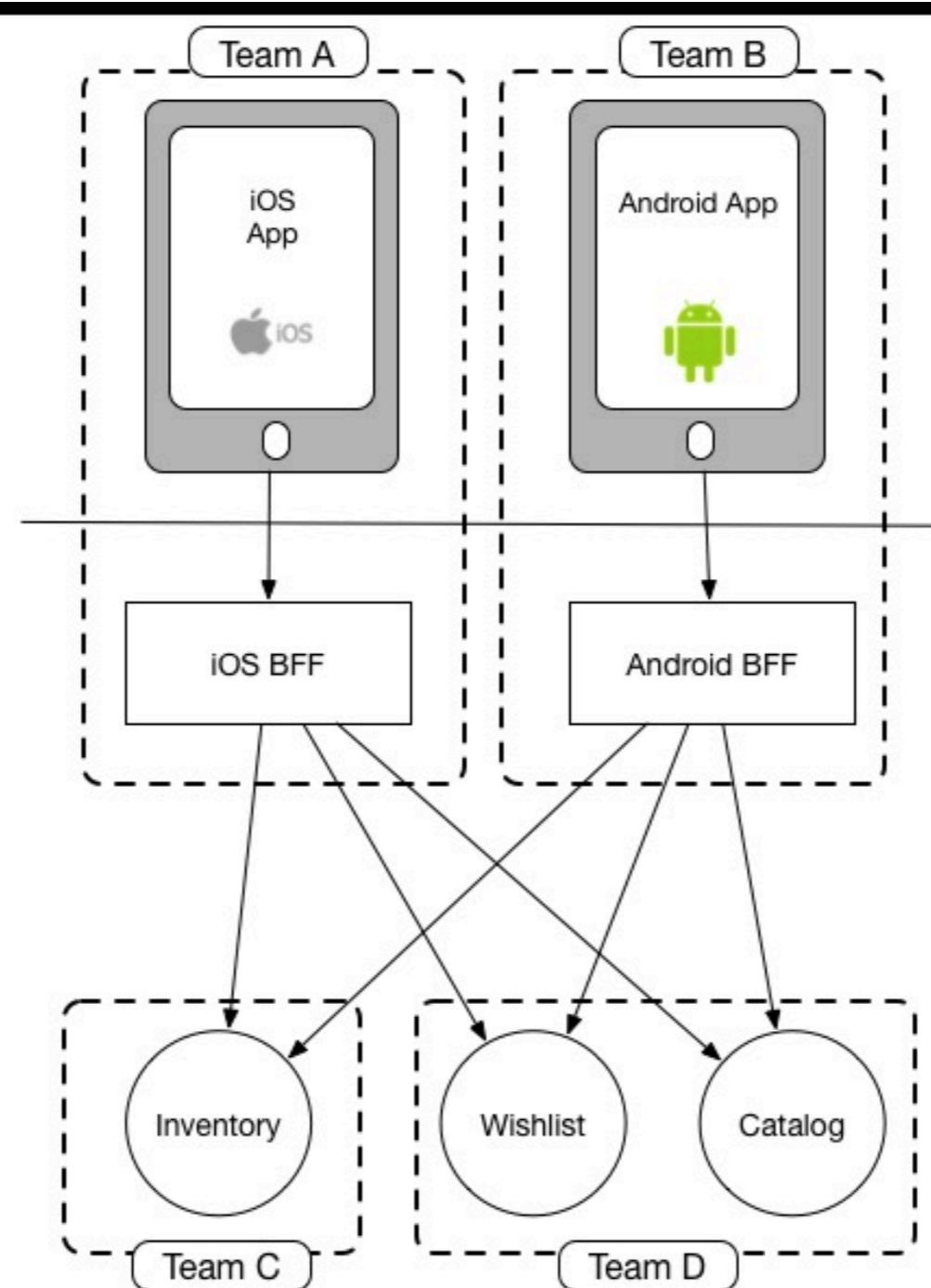
<https://samnewman.io/patterns/architectural/bff/>



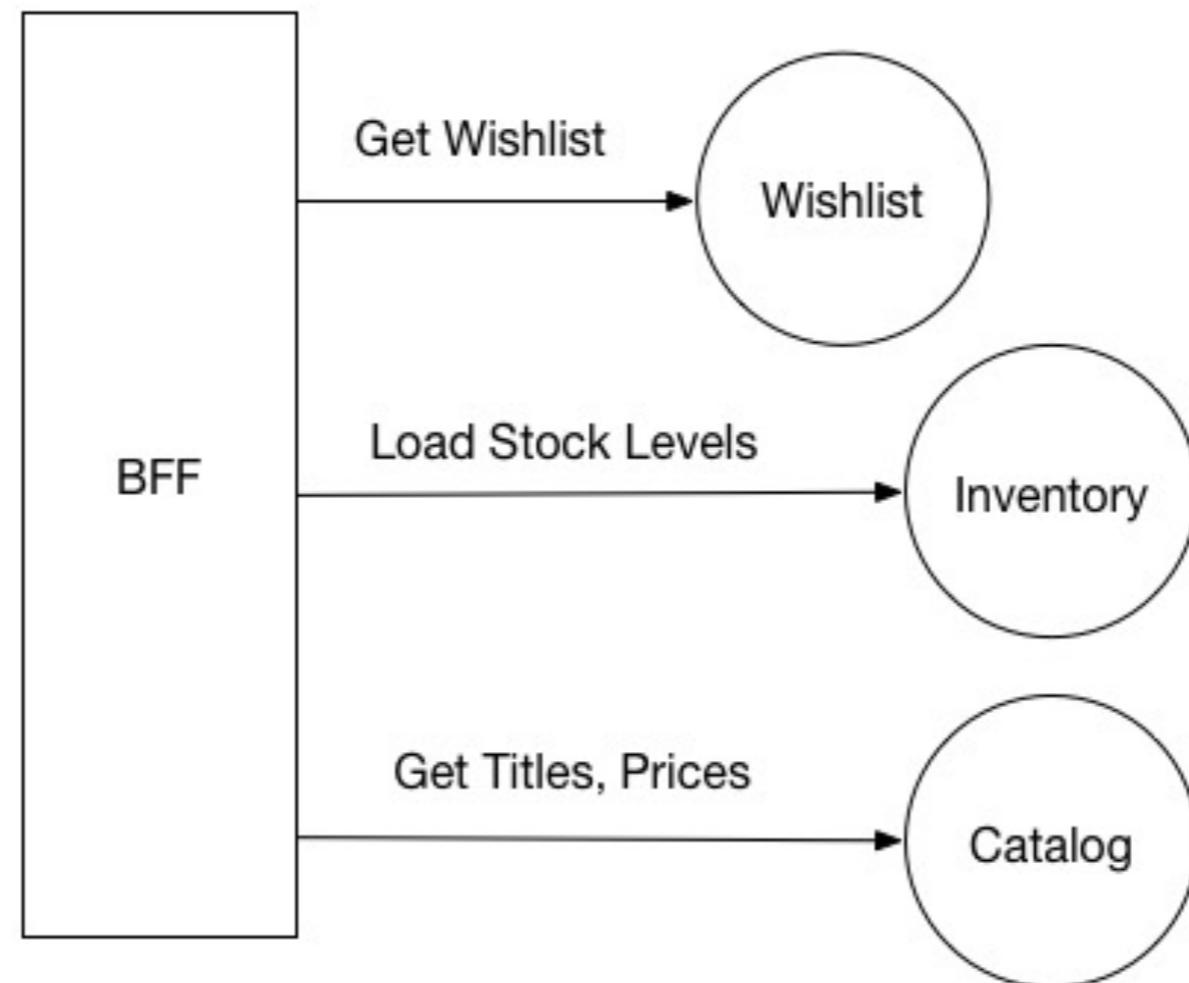
Backend for Frontend (BFF)



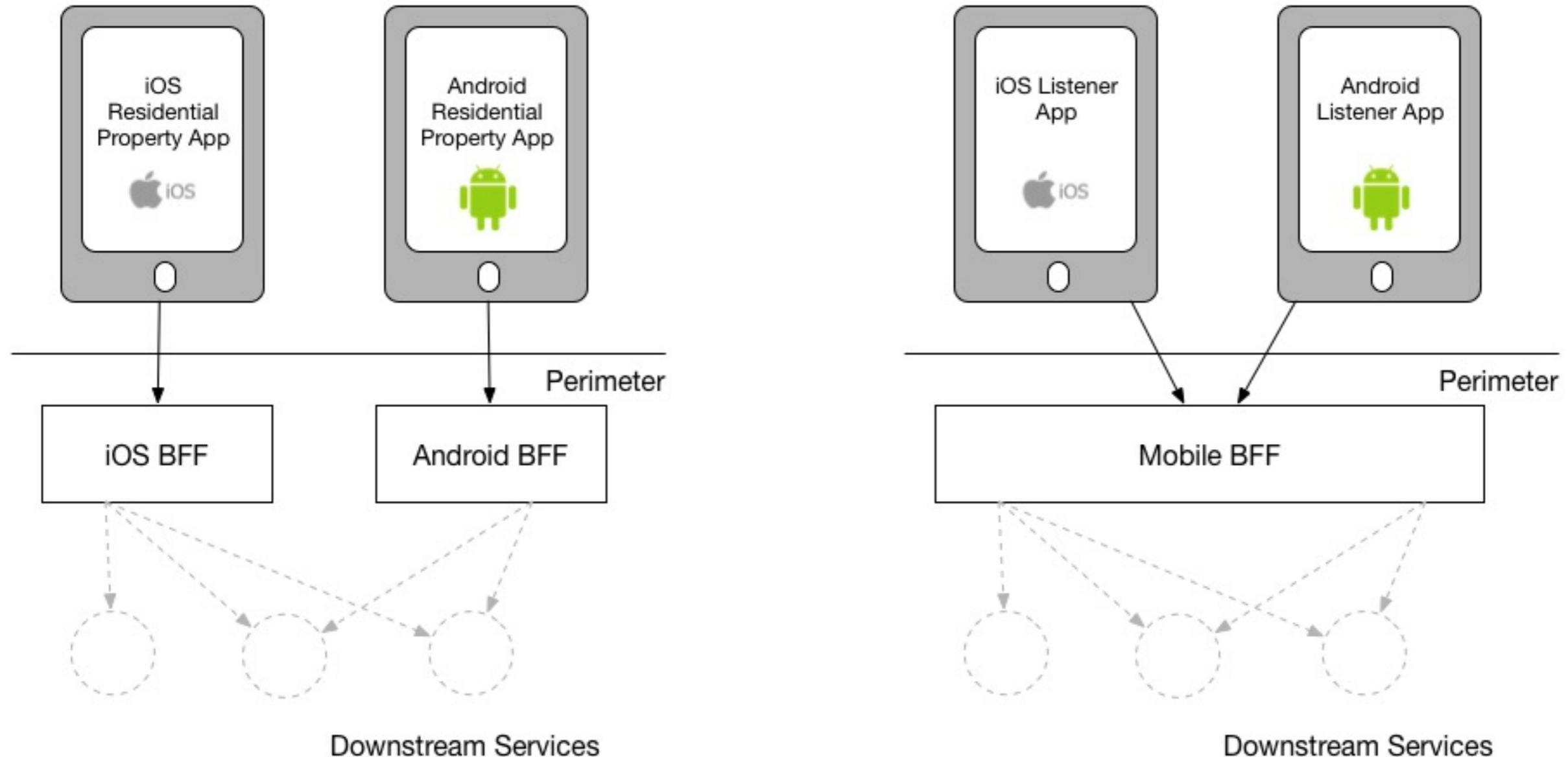
Backend for Frontend (BFF)



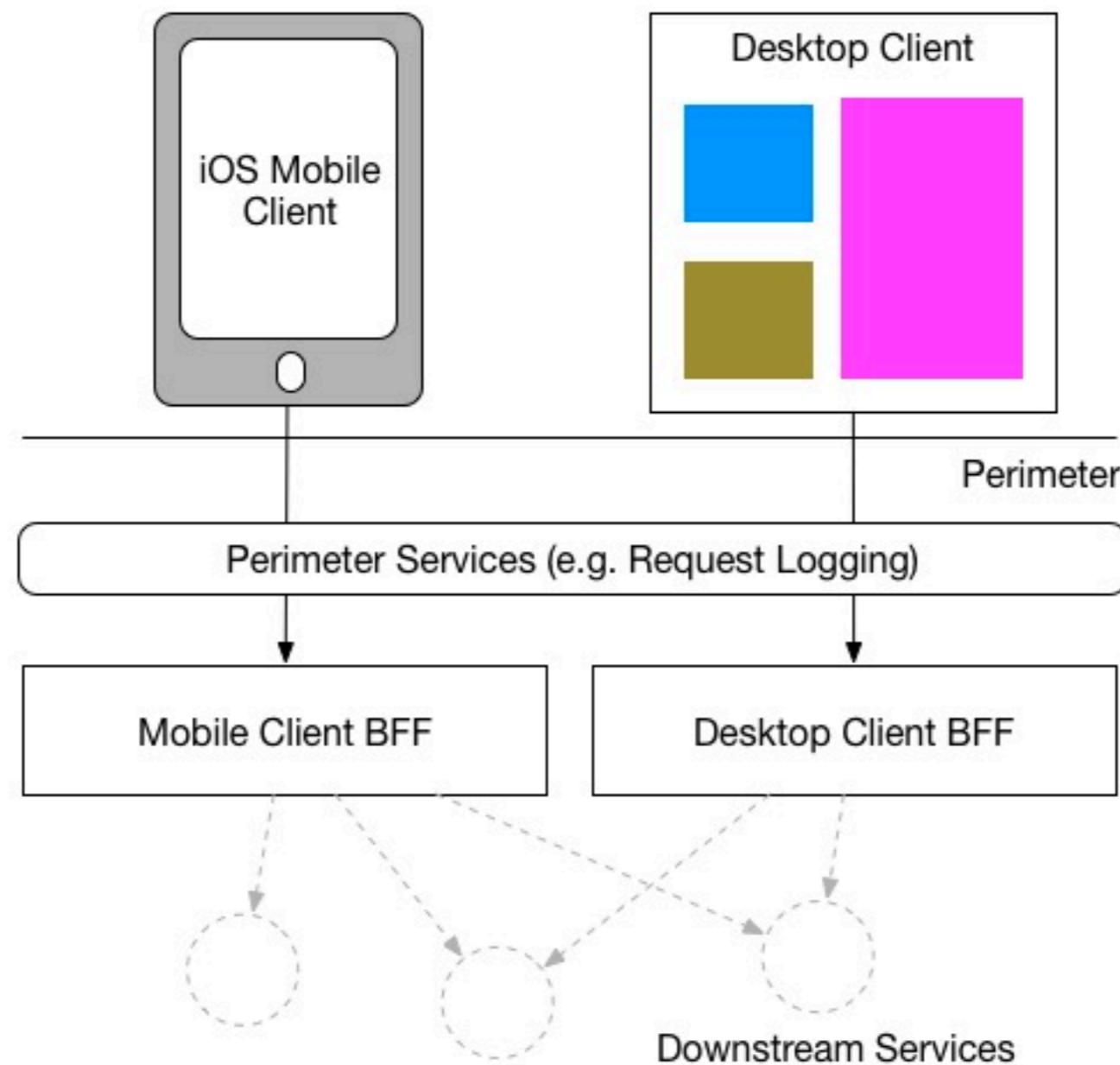
Backend for Frontend (BFF)

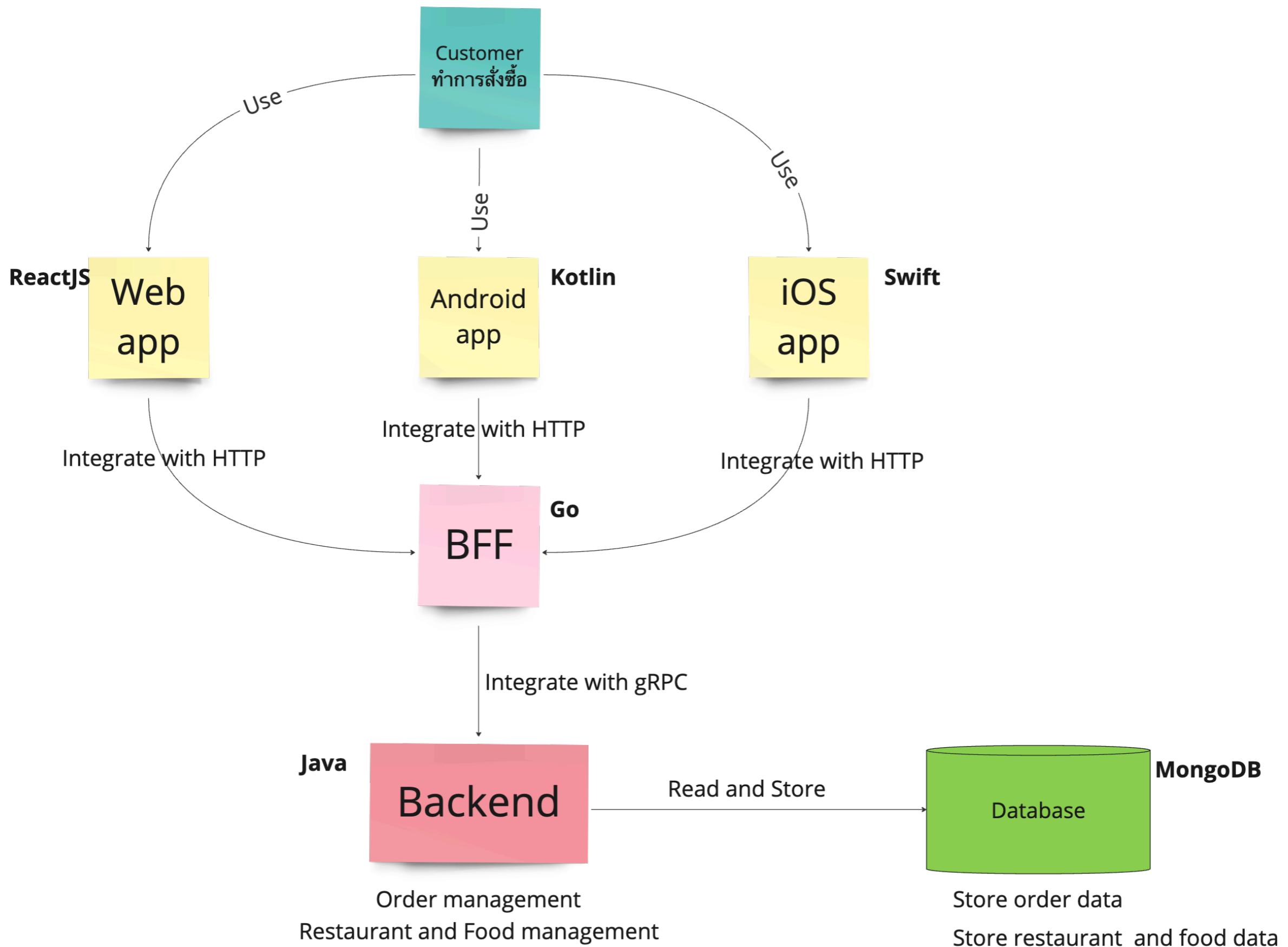


Backend for Frontend (BFF) ?

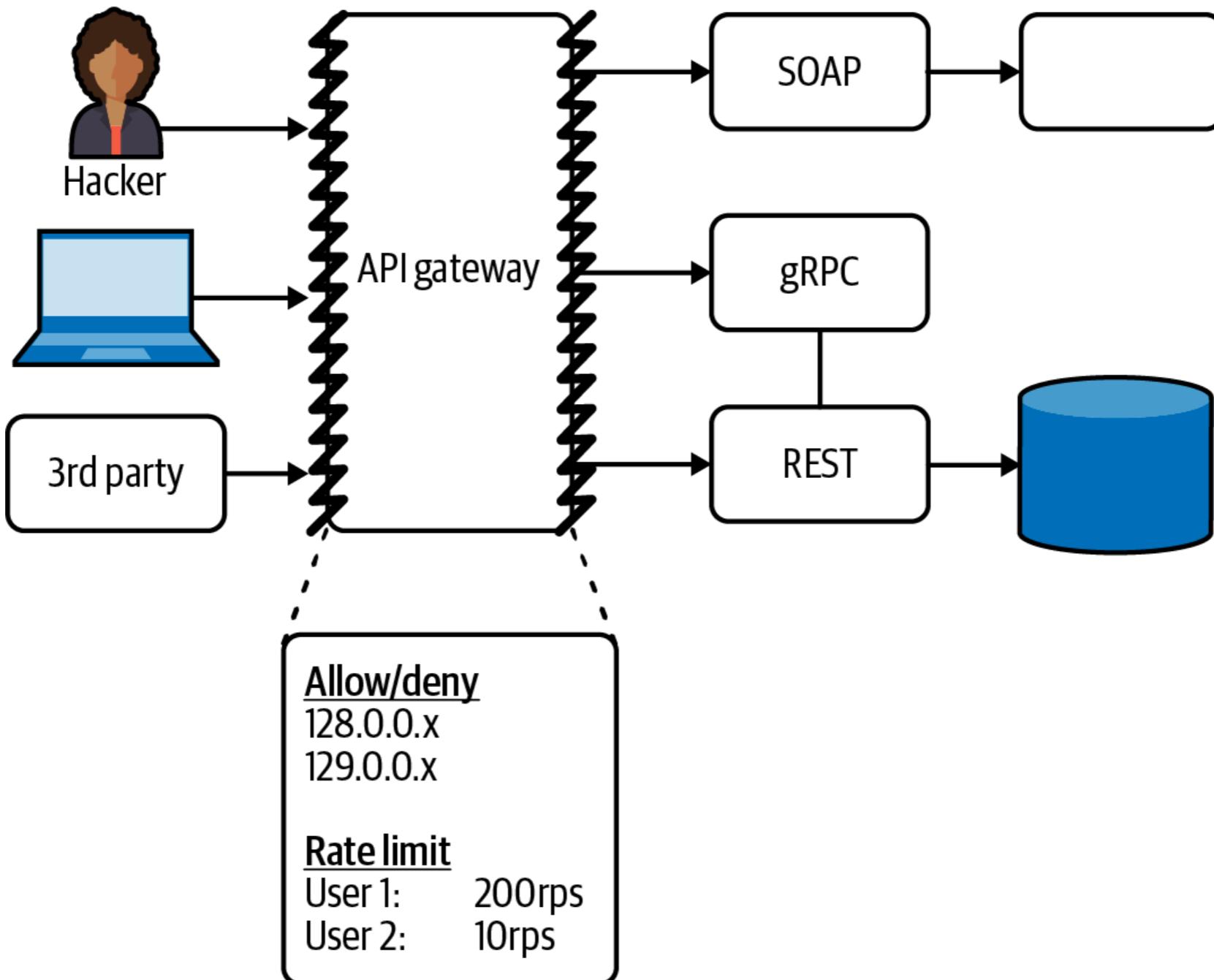


Backend for Frontend (BFF)

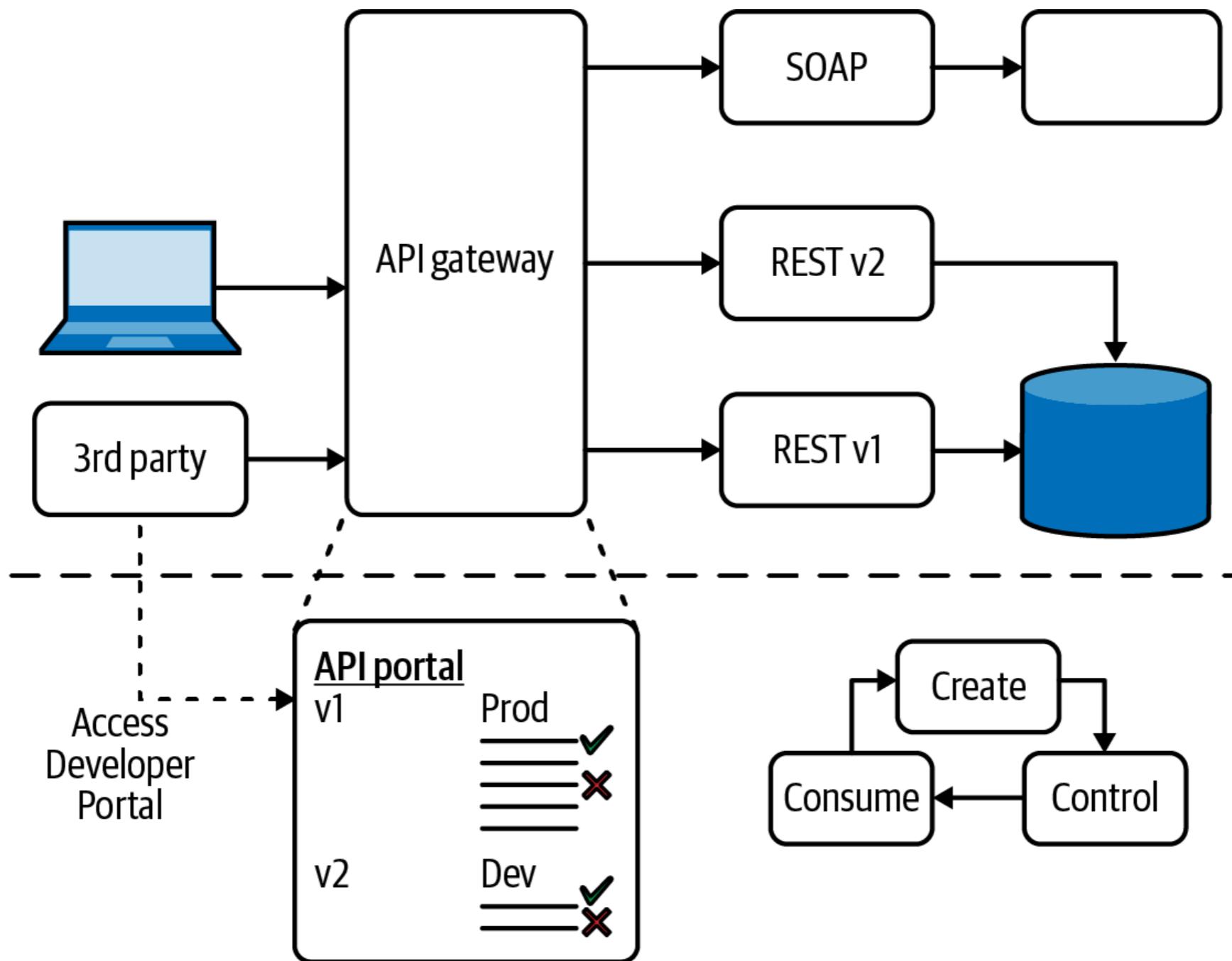




Need API Gateway ?

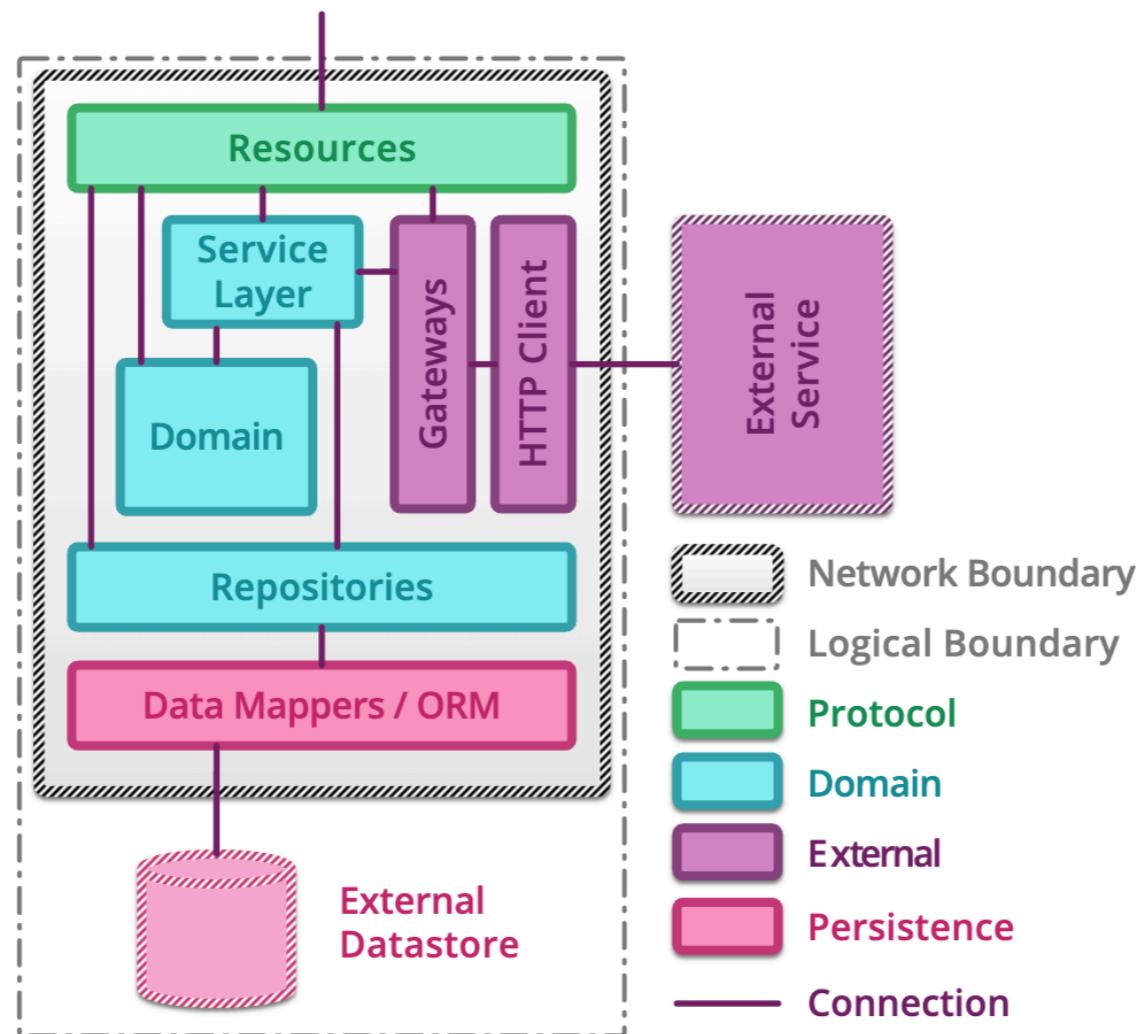


Need API Gateway ?



3. Components

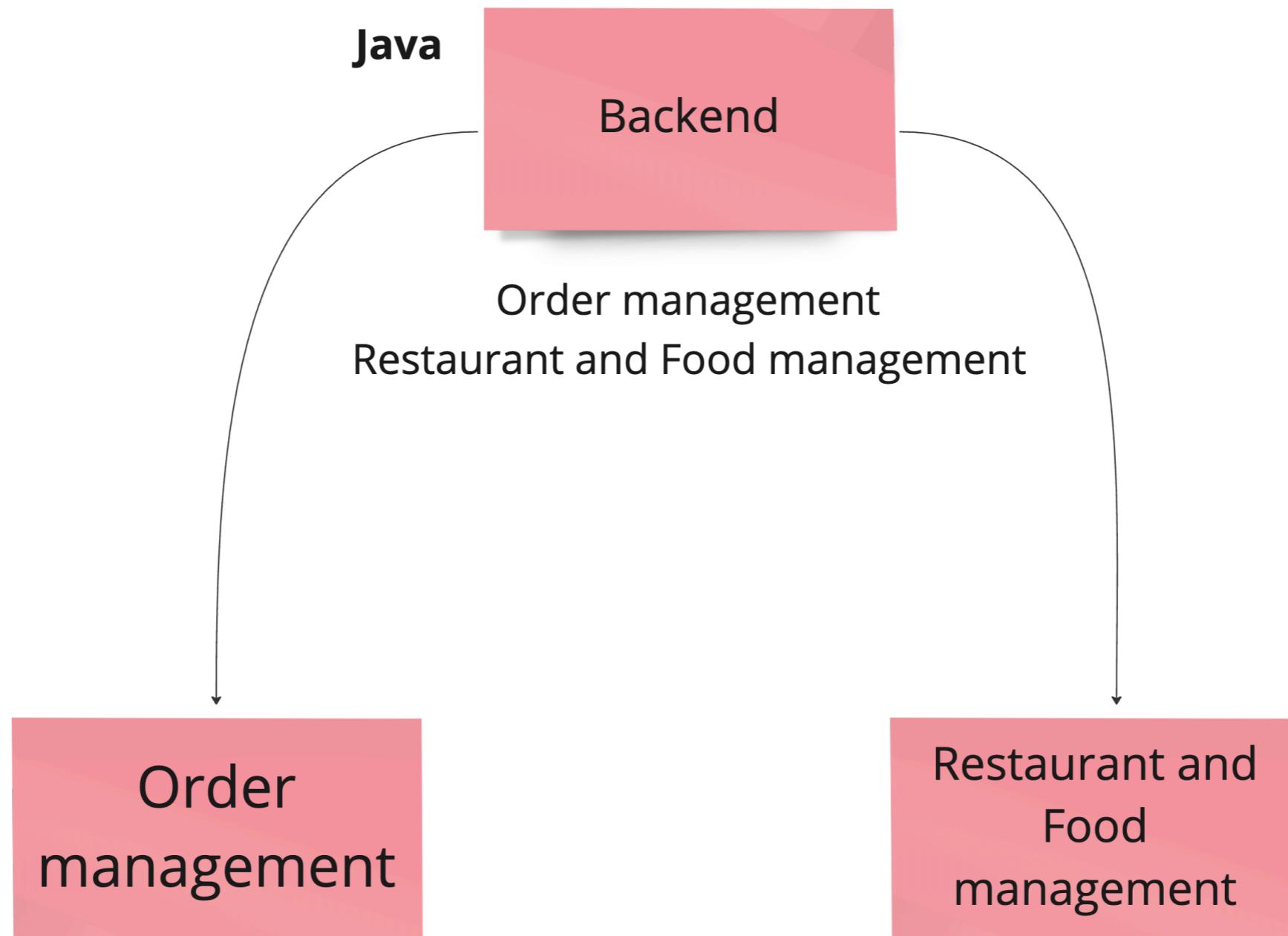
โครงสร้างการทำงานของเตล์และ container

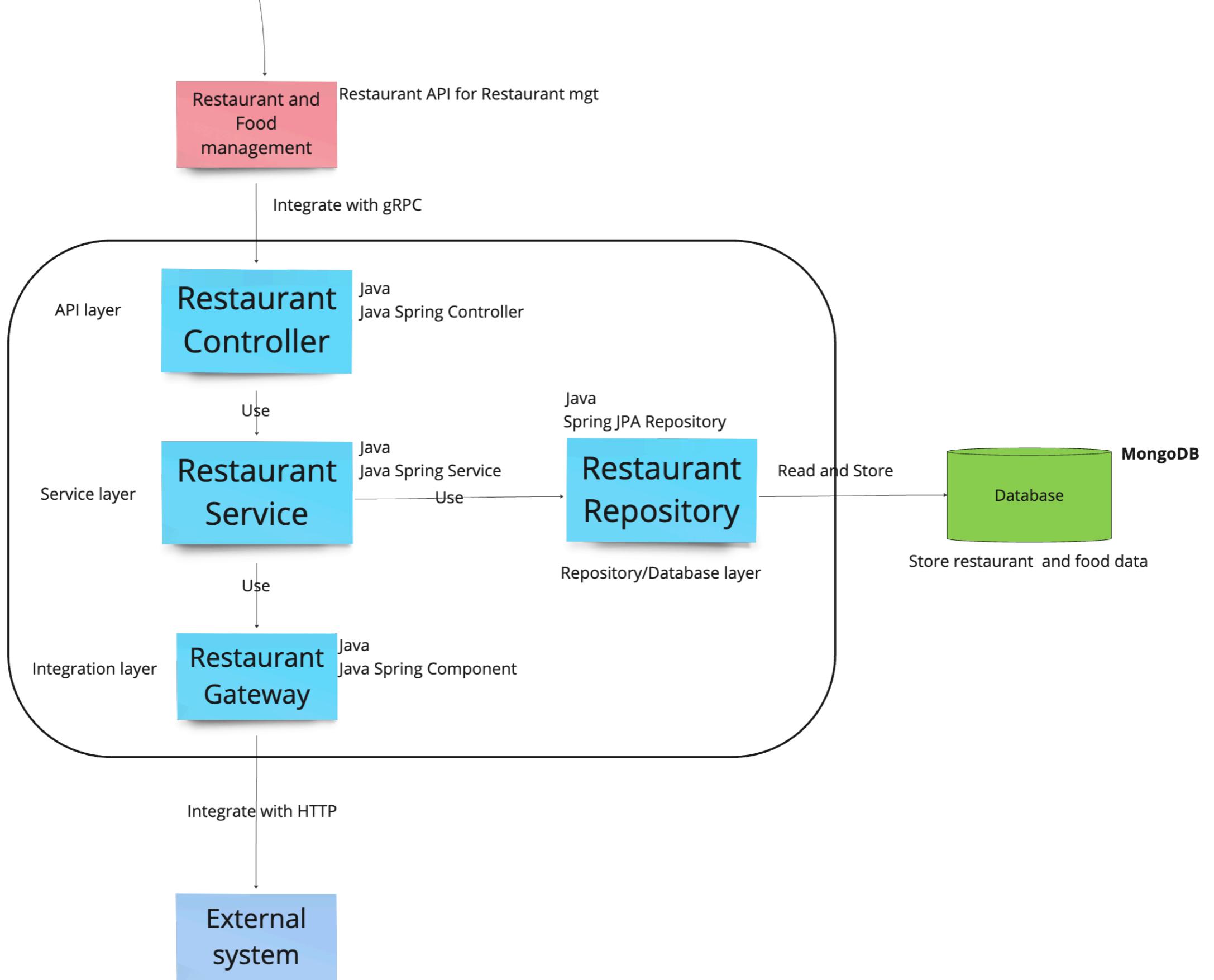


<https://martinfowler.com/articles/microservice-testing/>

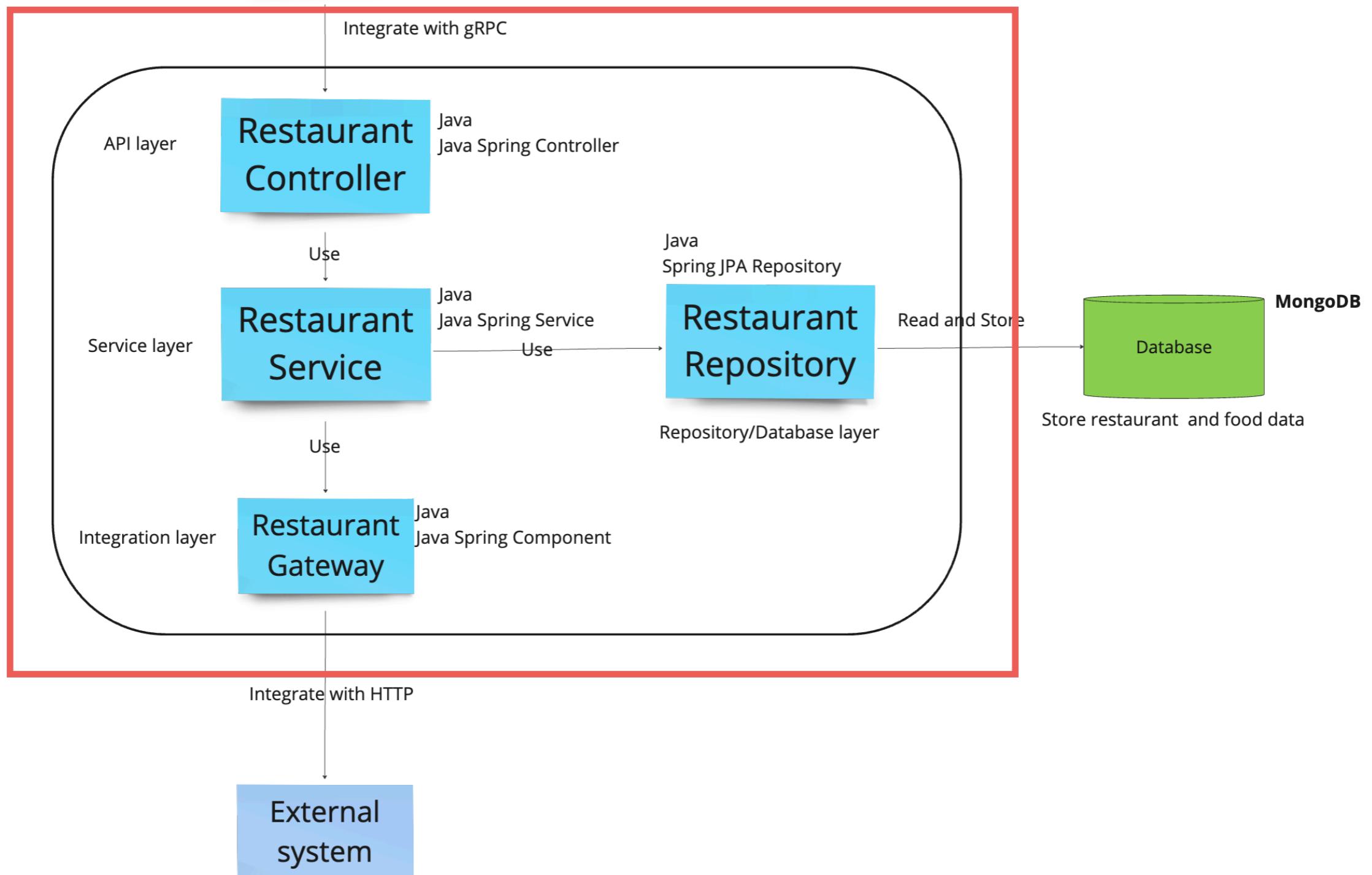


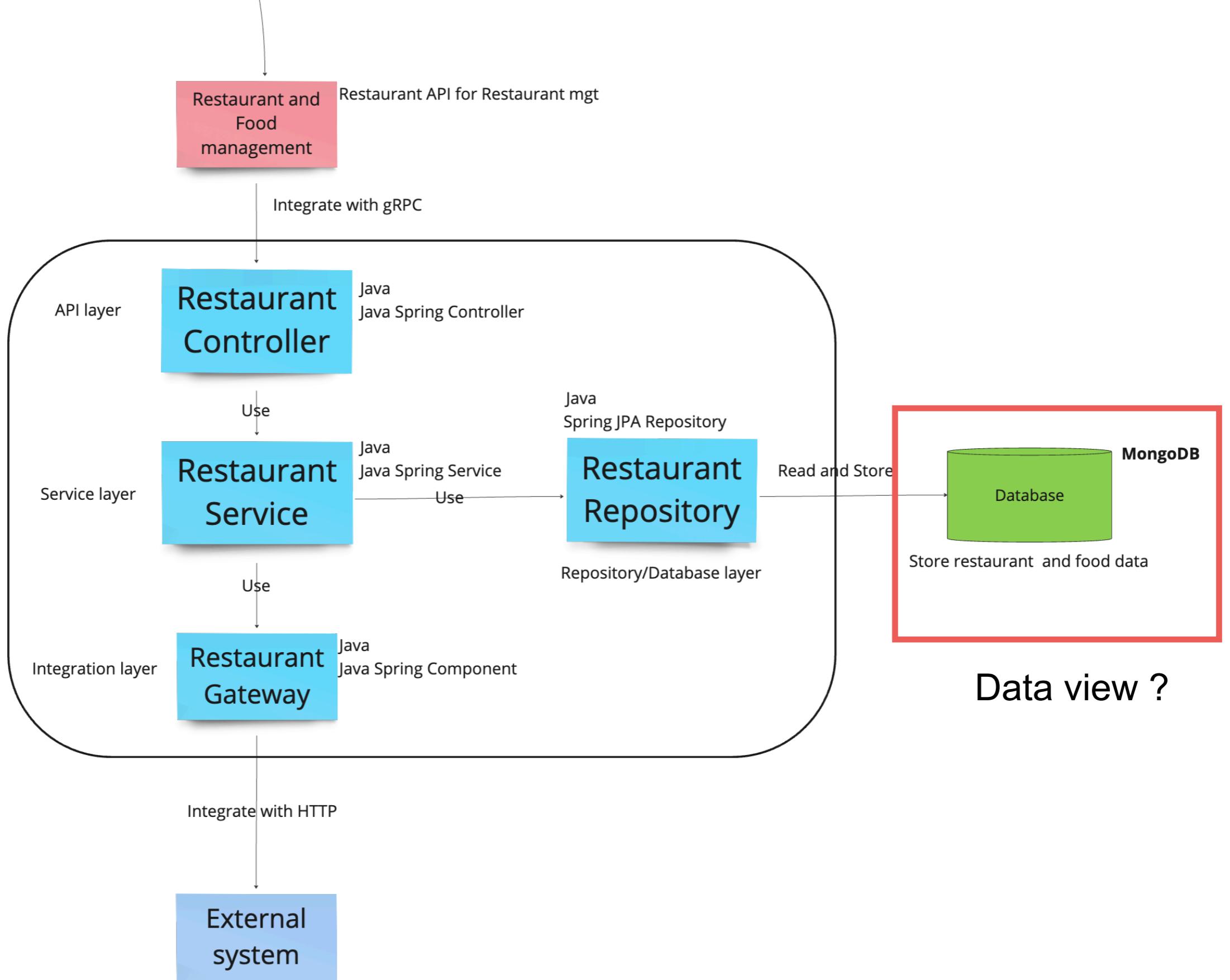
Example with Backend (Java)





Code with UML class/activity/sequence diagram





Testing ?



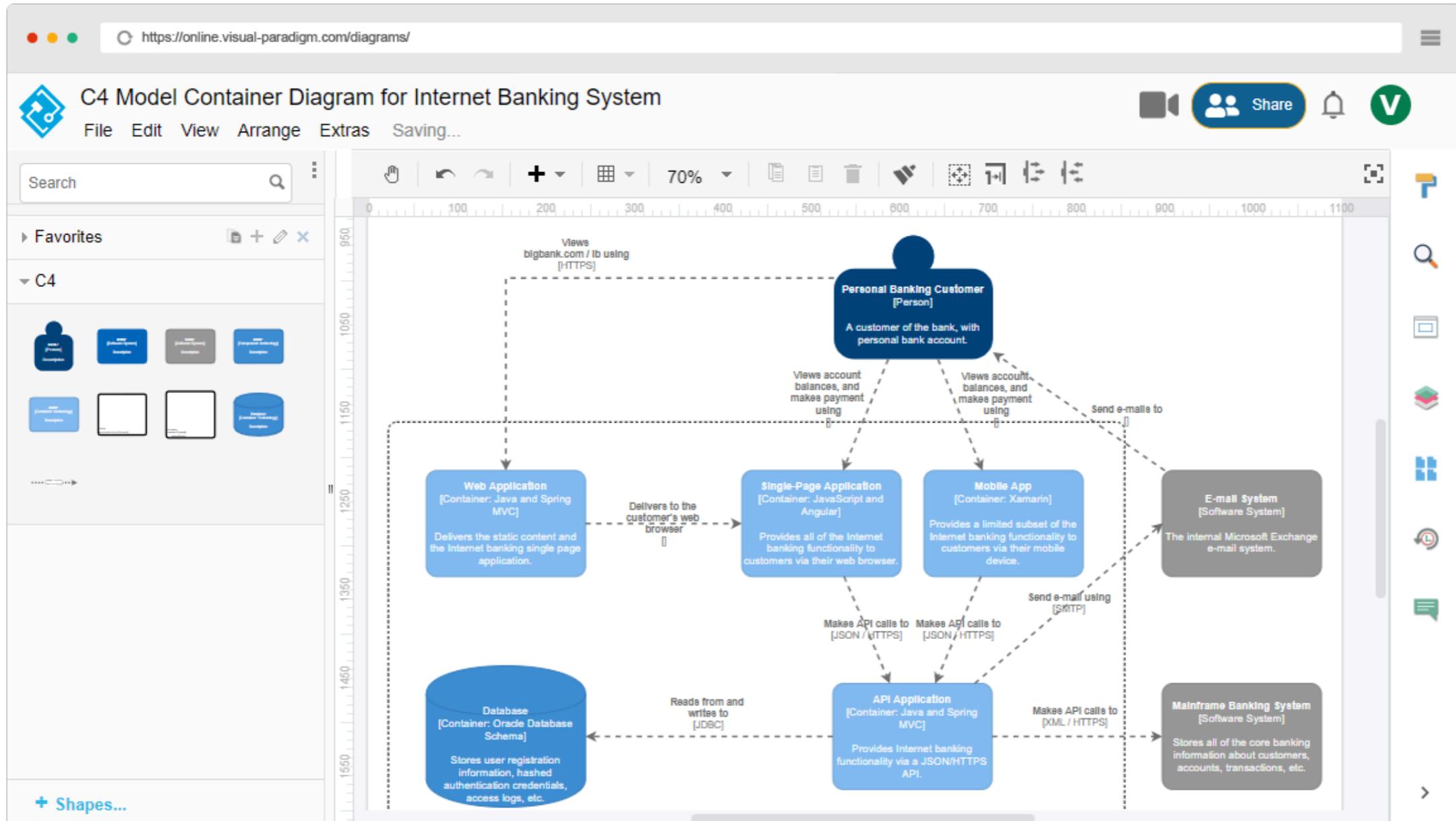
Deployment view ?



Operation view ?



Online C4 Model Software



<https://online.visual-paradigm.com/diagrams/features/c4-model-tool/>



Docs as Code



Write the Docs is a global community of people who care about documentation. We have a Slack community, conferences on 3 continents, and local meetups!

Quick search

Go

[Home](#) » [Site map](#) » [Learning Resources](#) » [Documentation Guide](#) »

Docs as Code

author: [Eric Holscher](#) & the Write the Docs community

Documentation as Code (*Docs as Code*) refers to a philosophy that you should be writing documentation with the same tools as code:

- Issue Trackers
- Version Control (Git)
- Plain Text Markup (Markdown, reStructuredText, Asciidoc)
- Code Reviews
- Automated Tests

This means following the same workflows as development teams, and being integrated in the product team. It enables a culture where writers and developers both feel ownership of documentation, and work together to make it as good as possible.

Generally a *Docs as Code* approach gives you the following benefits:

- Writers integrate better with development teams

<https://www.writethedocs.org/guide/docs-as-code/>



Architecture

© 2022 - 2023 Siam Chamnankit Company Limited. All rights reserved.

More resources

Awesome Software Architecture

<https://awesome-architecture.com/modeling/architecture-diagram/>



Example with Spring Modulith

<https://www.somkiat.cc/coding-with-spring-modulith/>



During solution design, Building a POC (Proof of concept)

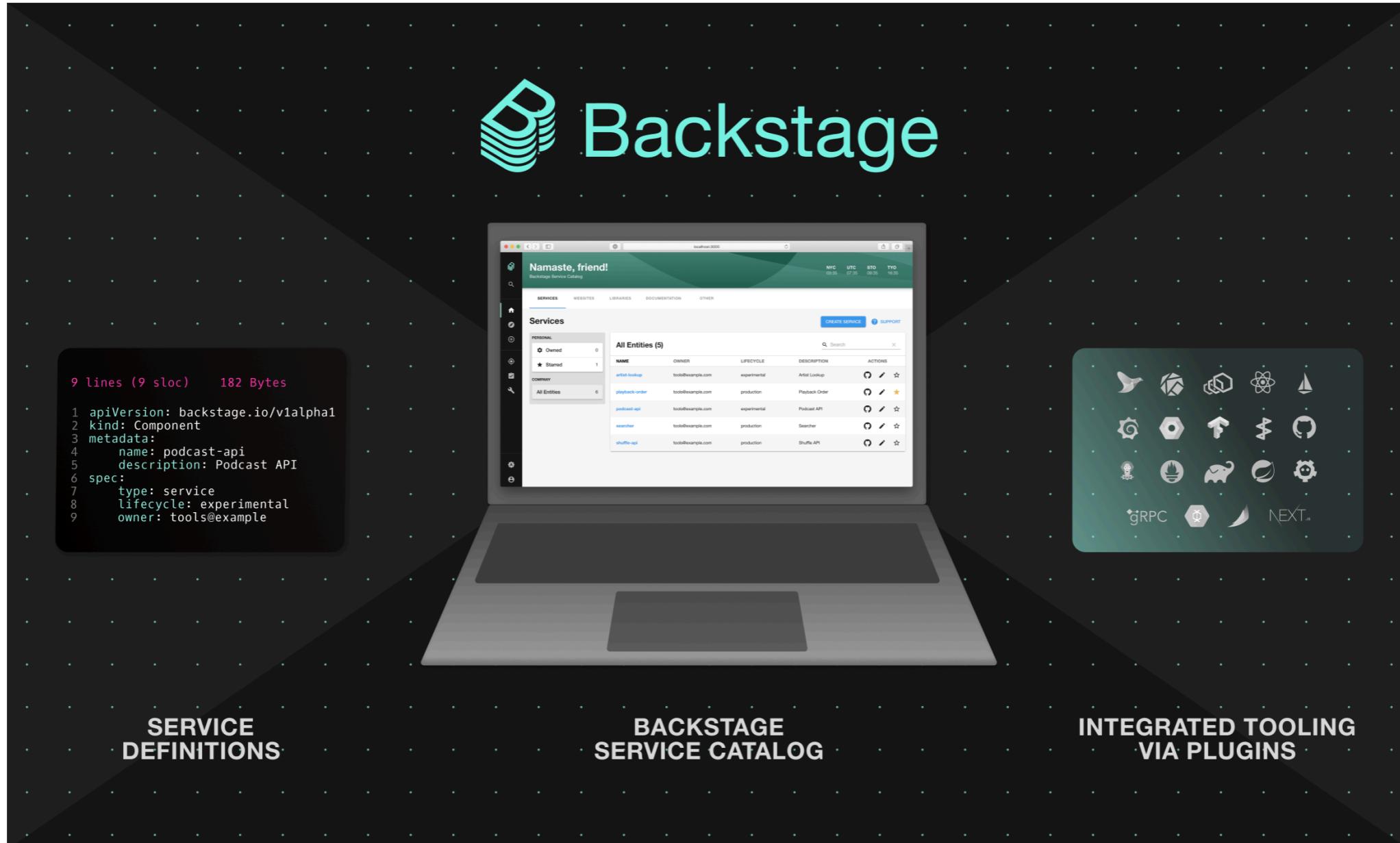


Example

Standardized architecture diagram ?



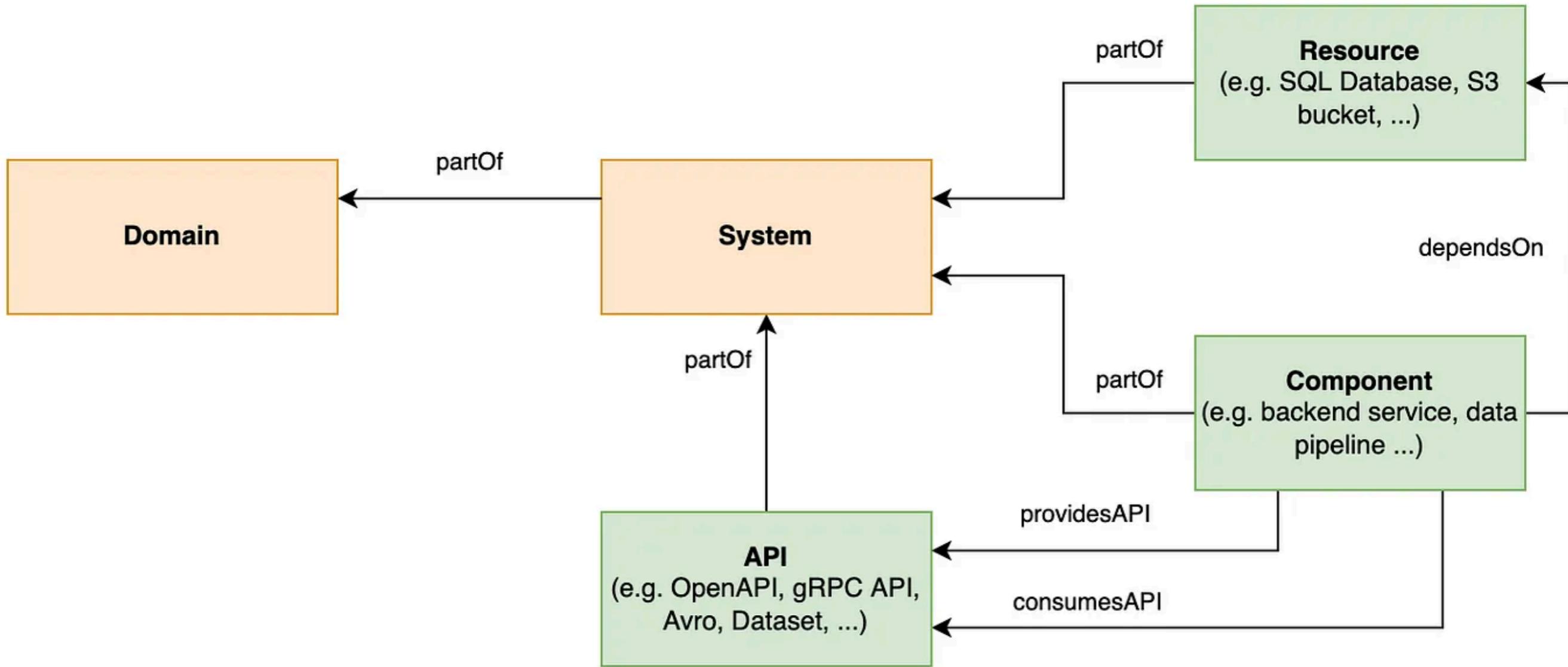
Backstage at Spotify



<https://backstage.io/docs/features/software-catalog/>



Backstage at Spotify



<https://backstage.io/docs/features/software-catalog/>



Demo

The screenshot shows the Backstage Catalog interface. The left sidebar contains navigation links: Home, APIs, Docs, Explore, Tech Radar, Cost Insights, and GraphQL. The main area is titled "Backstage Catalog" and displays a list of "Owned (3)" components. The table columns are: NAME, SYSTEM, OWNER, TYPE, LIFECYCLE, DESCRIPTION, TAGS, and ACTIONS. The components listed are:

NAME	SYSTEM	OWNER	TYPE	LIFECYCLE	DESCRIPTION	TAGS	ACTIONS
playback-order	audio-playback	user:guest	service	production	Playback Order	java playback	Edit View Star
searcher		user:guest	service	production	Searcher	go	Edit View Star
shuffle-api	audio-playback	user:guest	service	production	Shuffle API	go	Edit View Star

On the left, there are filters for Kind (Component), Type (all), and sections for PERSONAL (Owned: 3, Starred: 0) and BACKSTAGE (All: 15).

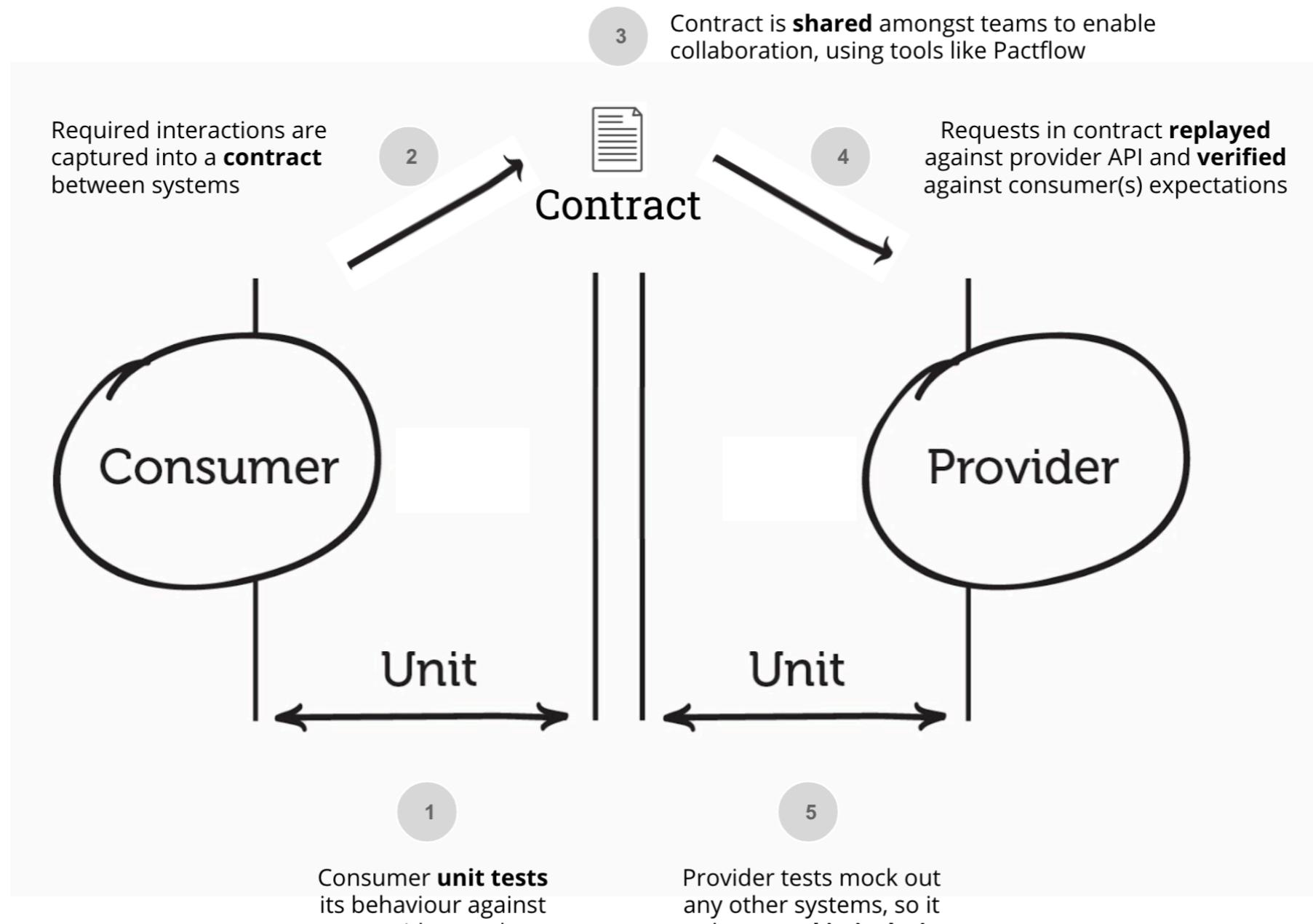
<https://demo.backstage.io/>



Component and Contract Testing



Contract Testing



<https://docs.pact.io/>



Pact Broker

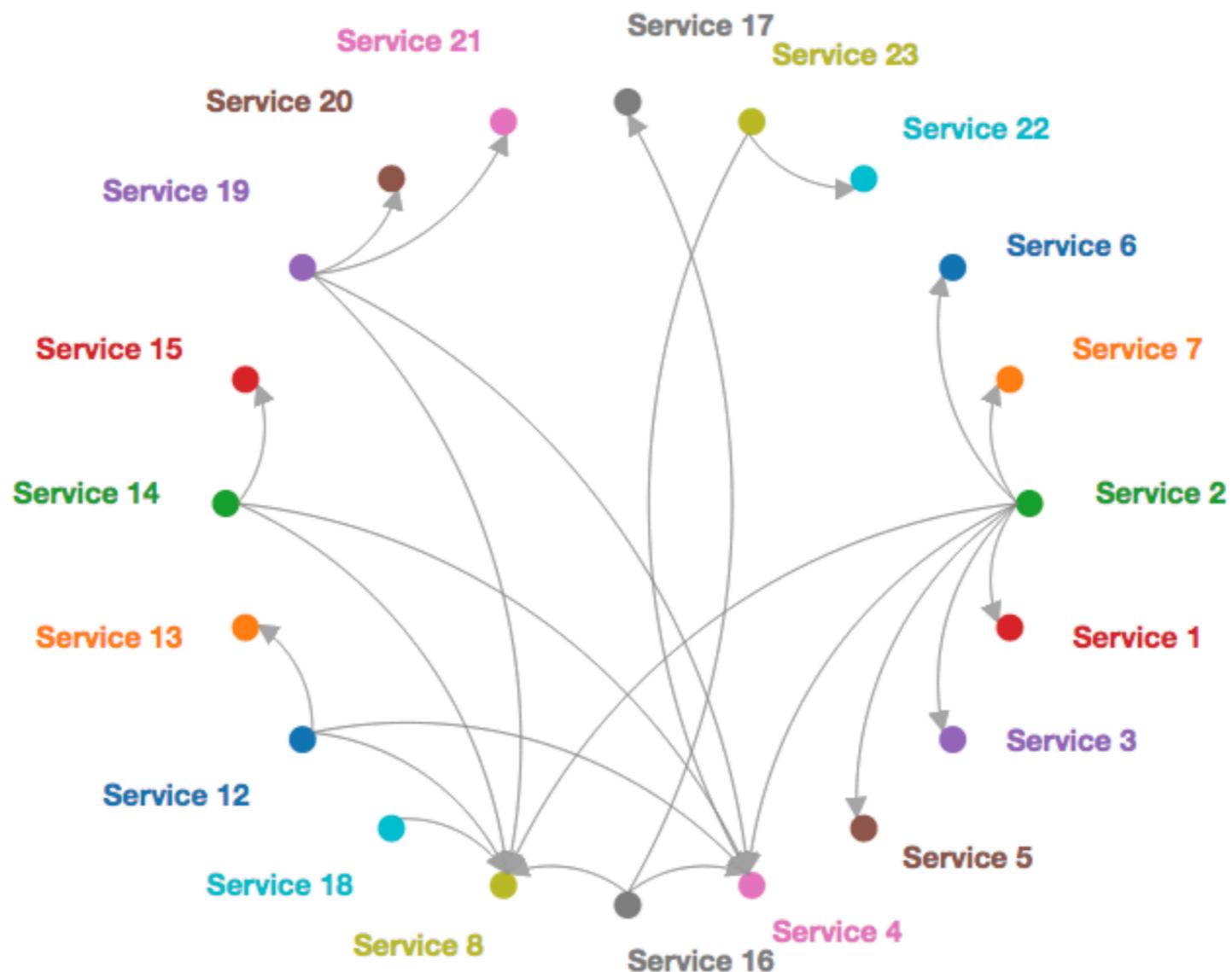
Pacts

Consumer ↓↑	Provider ↓↑	Latest pact published	Last verified
Foo	Animals	2 minutes ago	2 days ago
Foo	Bar	7 days ago	15 days ago ⚠
Foo	Hello World App	1 day ago	
Foo	Wiffles	less than a minute ago	7 days ago
Some other app	A service	26 days ago	less than a minute ago
The Android App	The back end	less than a minute ago	

<https://docs.pact.io/>



Pact Broker



<https://docs.pact.io/>



ADR

Architecture Decision Record

<https://github.com/joelparkerhenderson/architecture-decision-record>

