Microservices & Cloud Computing – Part 2

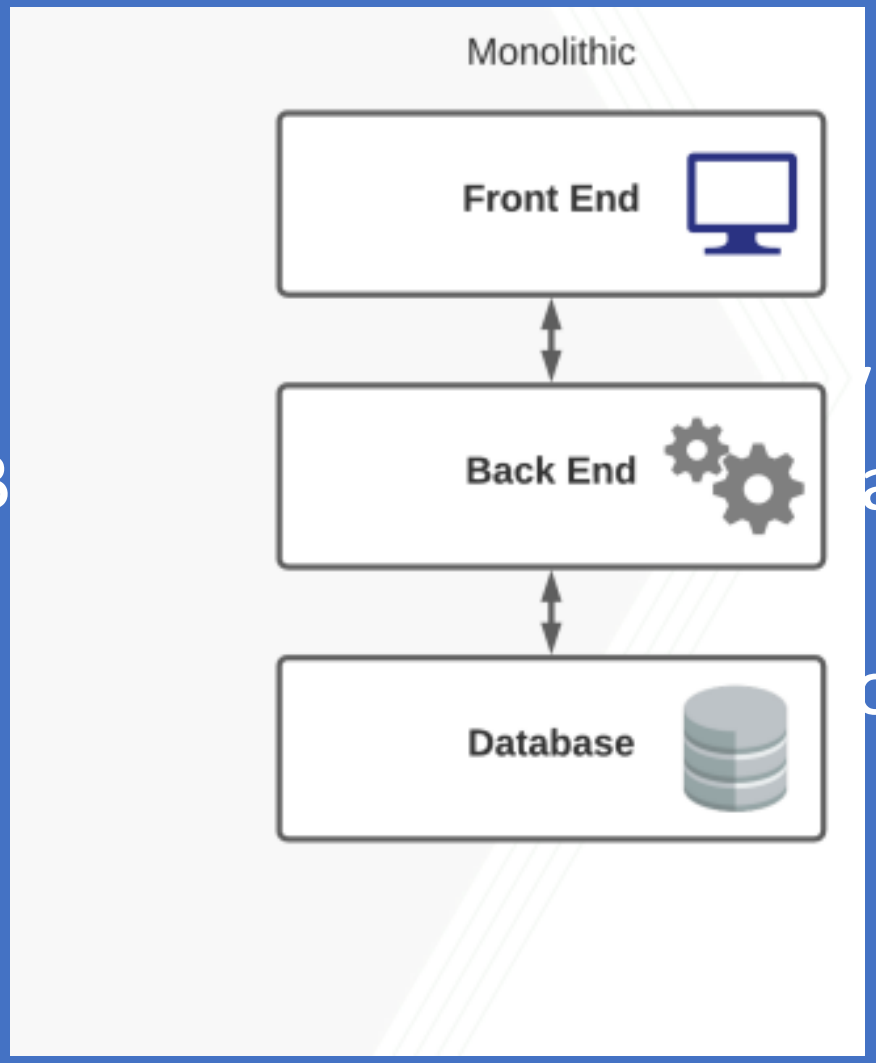
ECE 4150

Vijay Madisetti

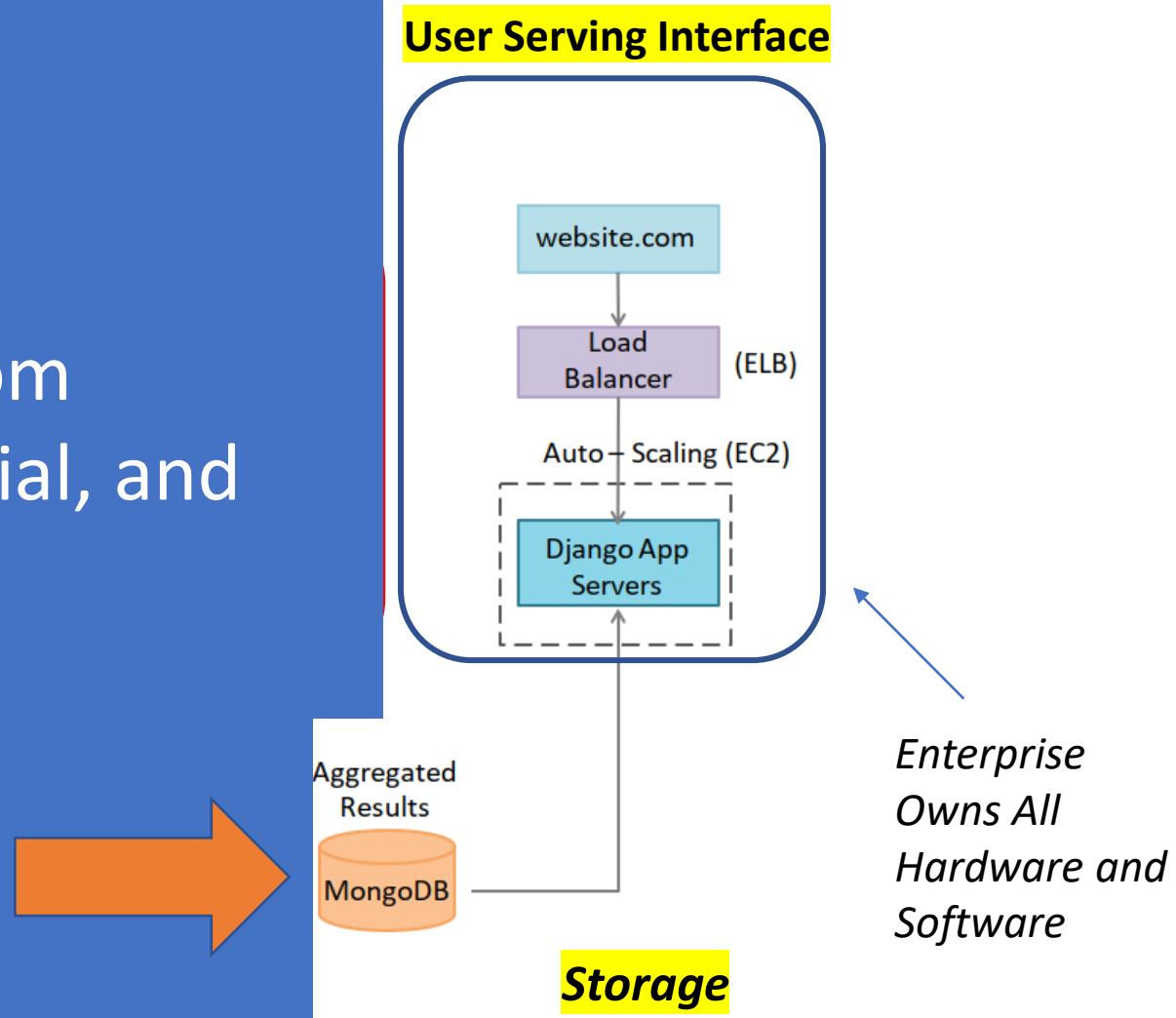
Spring 2024

Georgia Tech

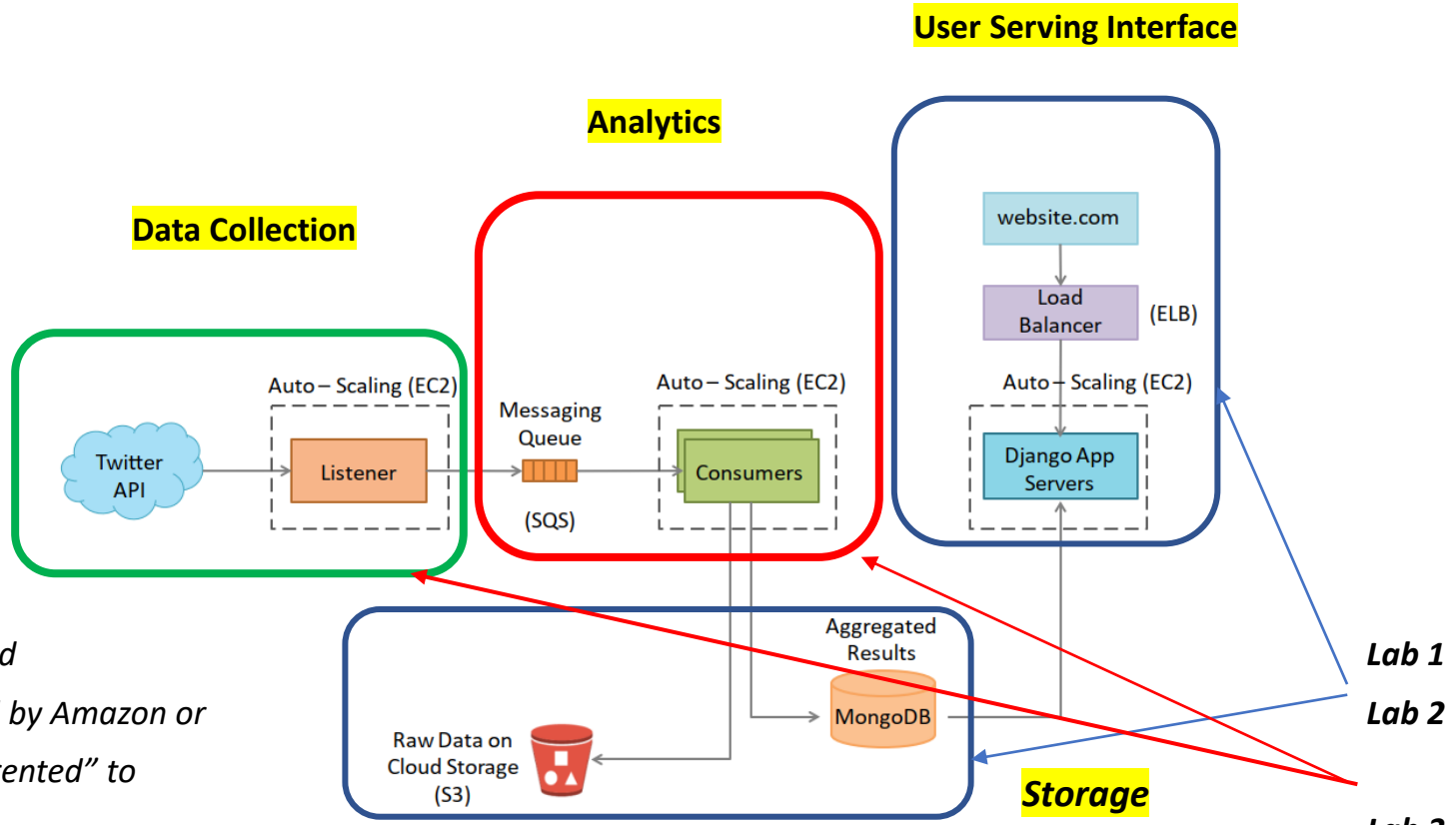
How Enterprise Applications Were *Monolithic* Prior to Cloud Computing



B from
ancial, and
ols

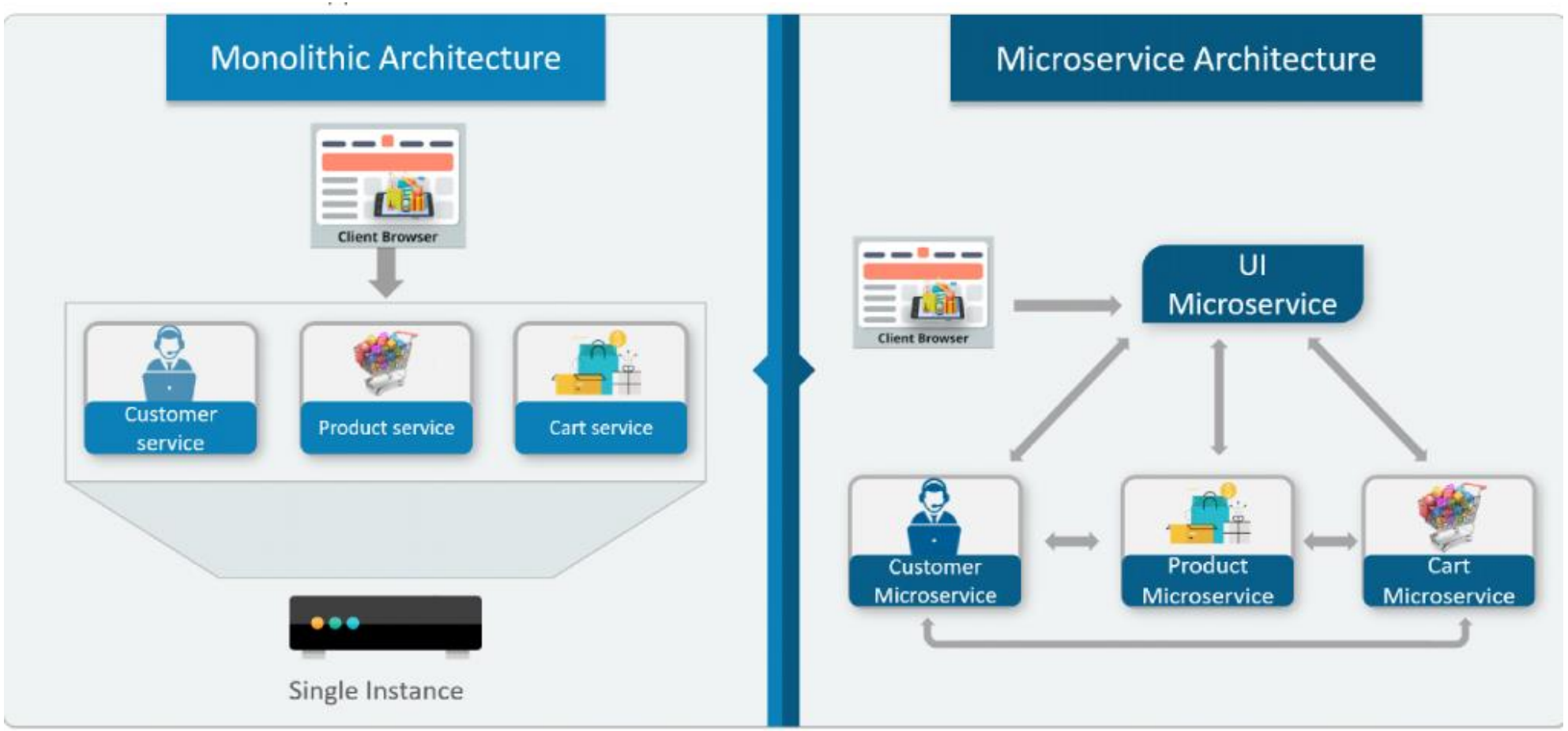


Evolution to Cloud Architectures

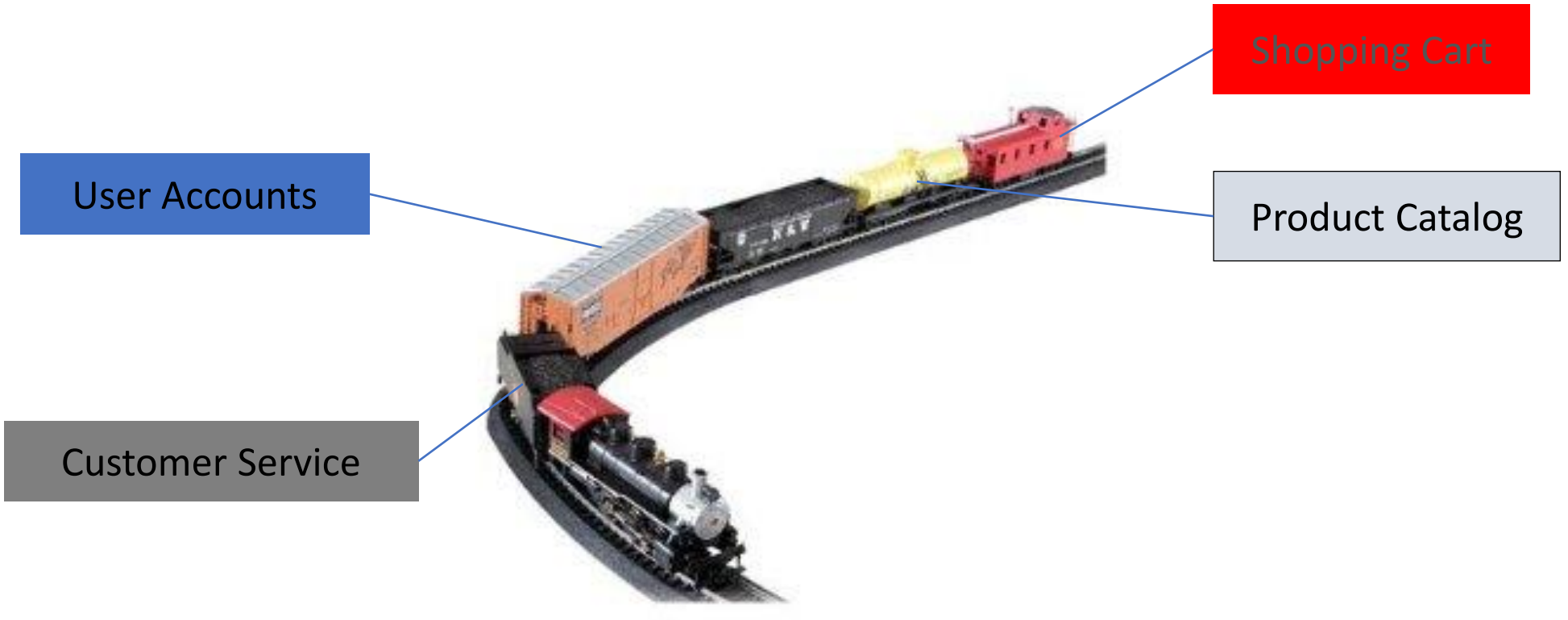


All Hardware and Software owned by Amazon or Microsoft and "rented" to Enterprise

Monolithic versus Microservices Approach



Evolution of a Monolithic App



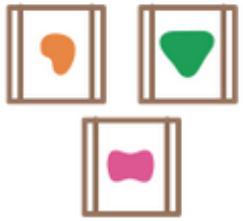
MicroServices – separate single purpose services

The Scaling Cube in Action via Microservices

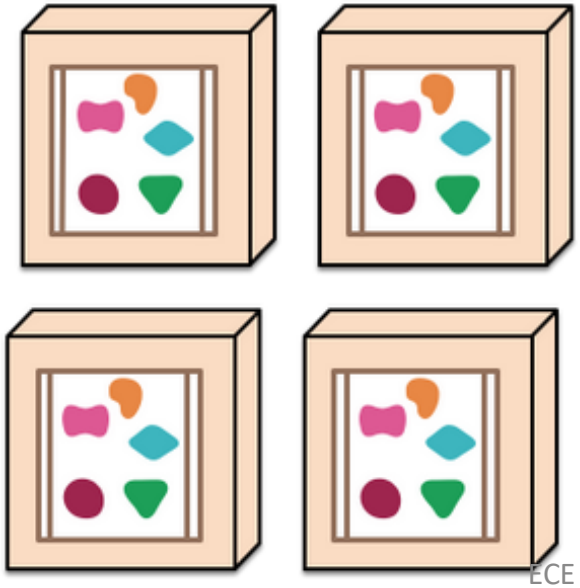
Which is why we need HDFS, Storm, Zookeeper, Leader Election, Queues, Brokers, Load Balancers, etc., etc., - Distributed Computing & Data

A monolithic application puts all its functionality into a single process...

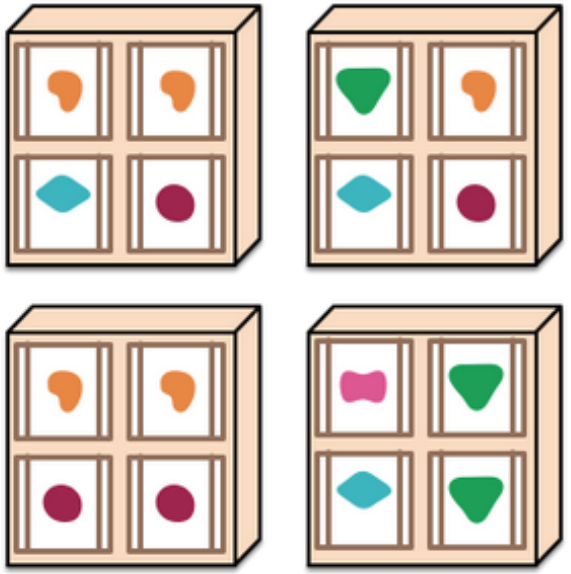
A microservices architecture puts each element of functionality into a separate service...



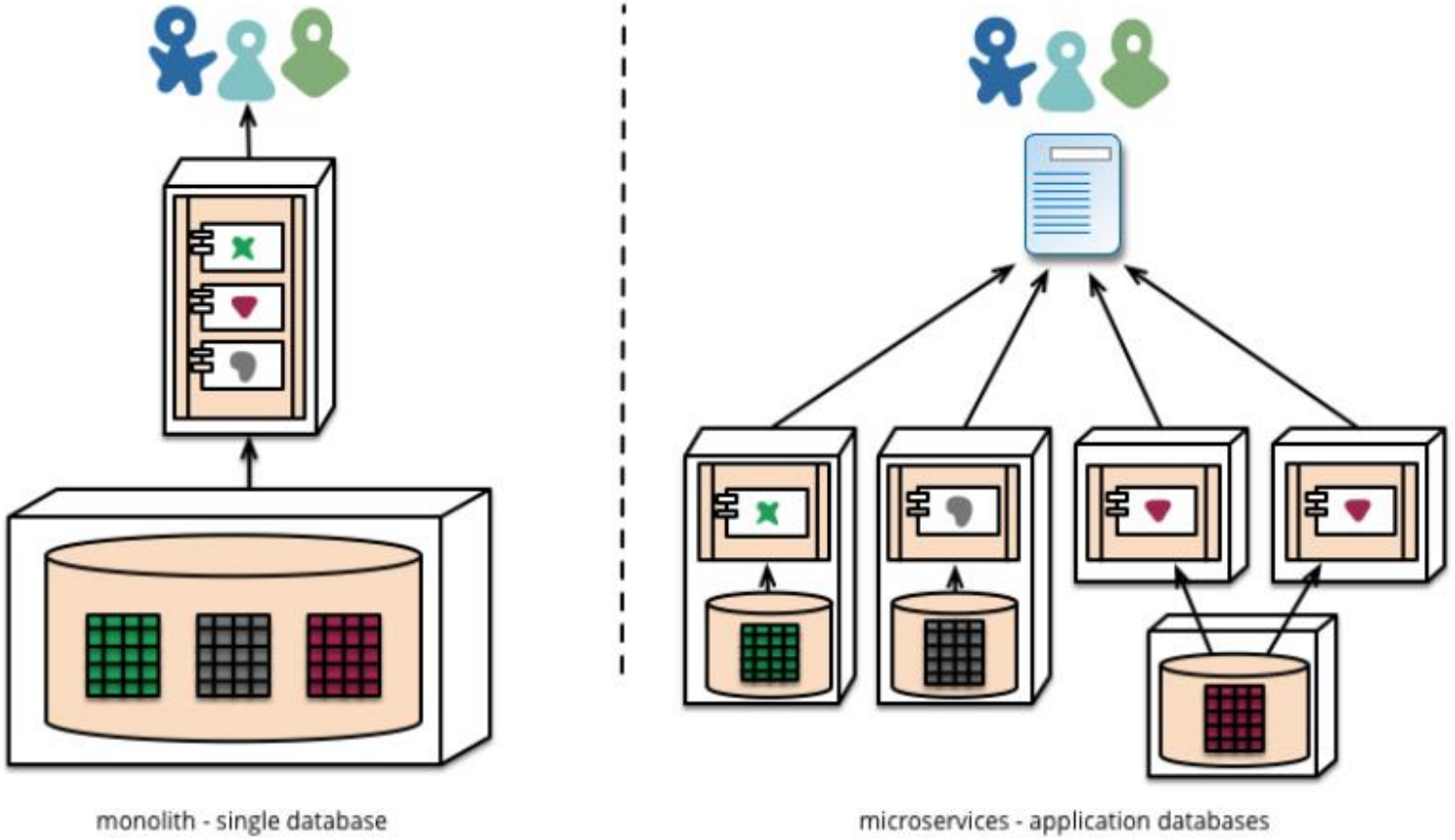
... and scales by replicating the monolith on multiple servers



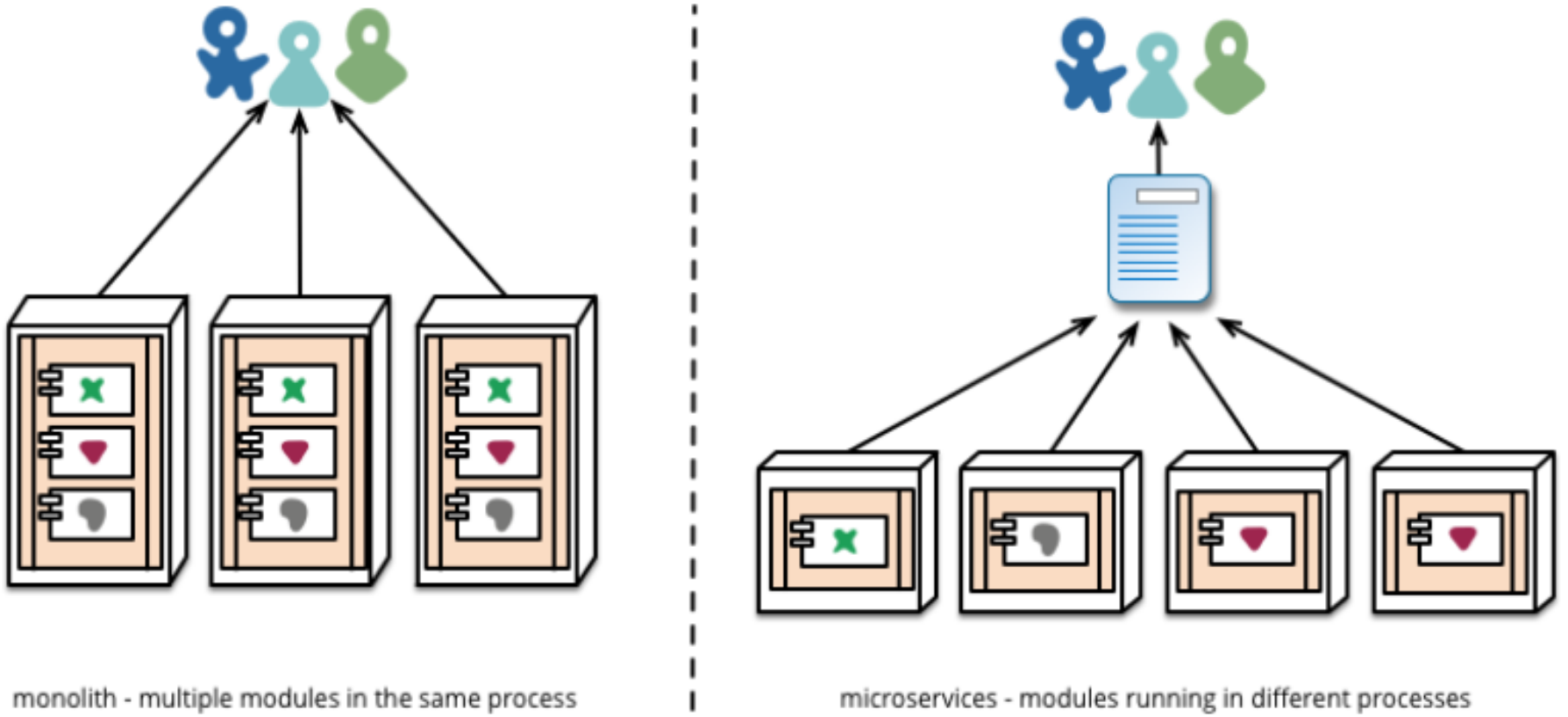
... and scales by distributing these services across servers, replicating as needed.



Replication and Partitioning



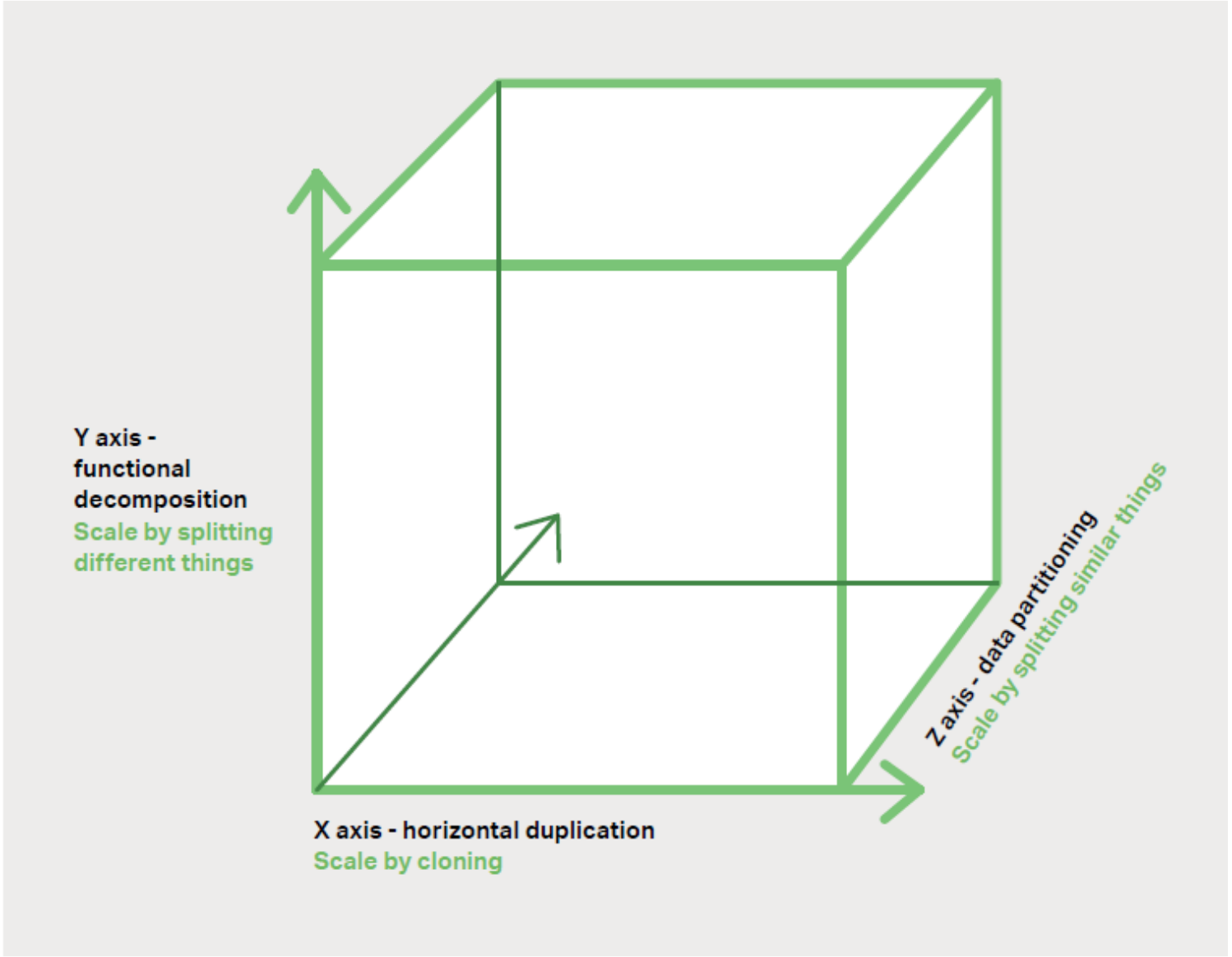
Replication and Partitioning



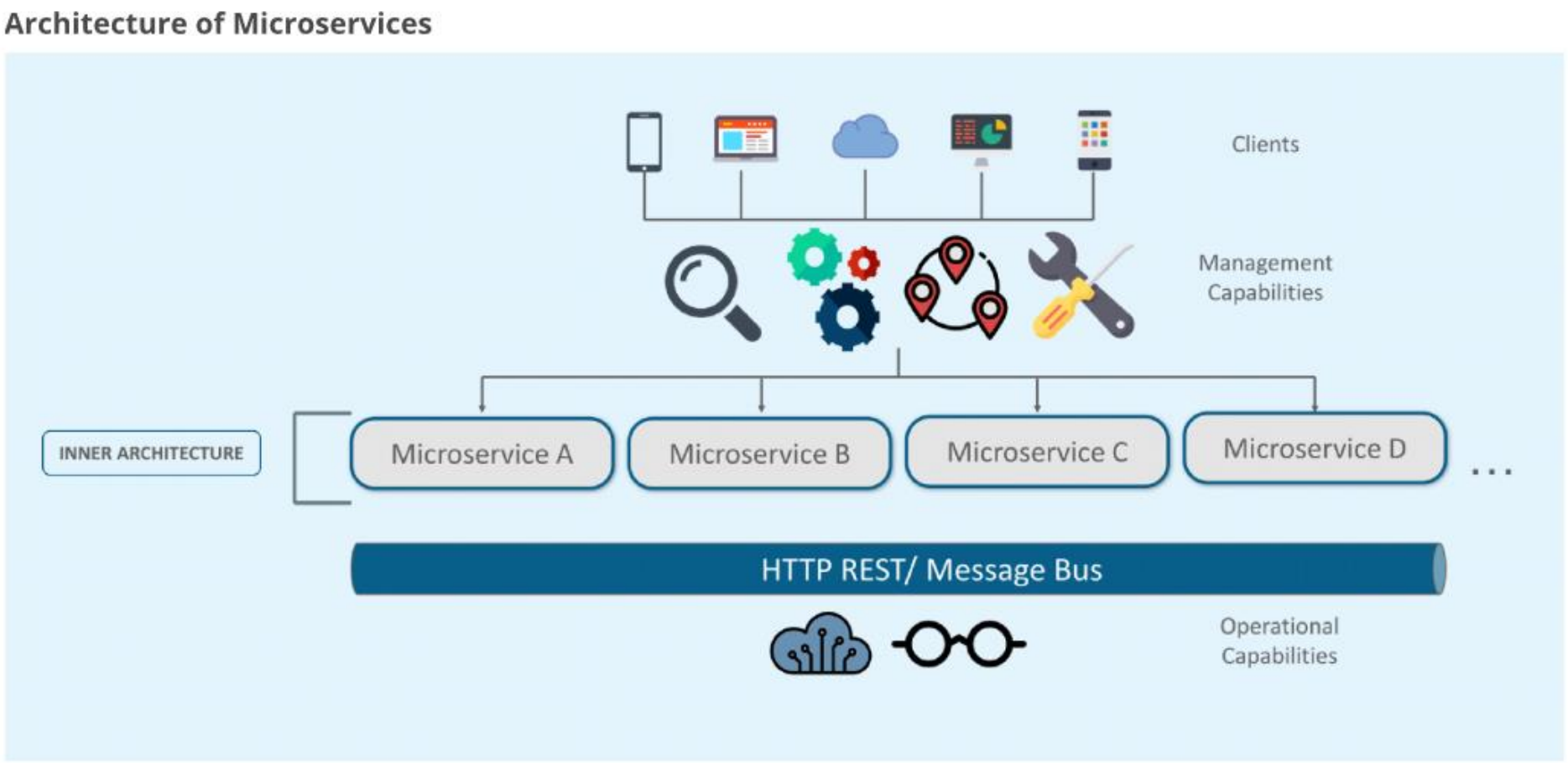
monolith - multiple modules in the same process

microservices - modules running in different processes

The Scaling Cube



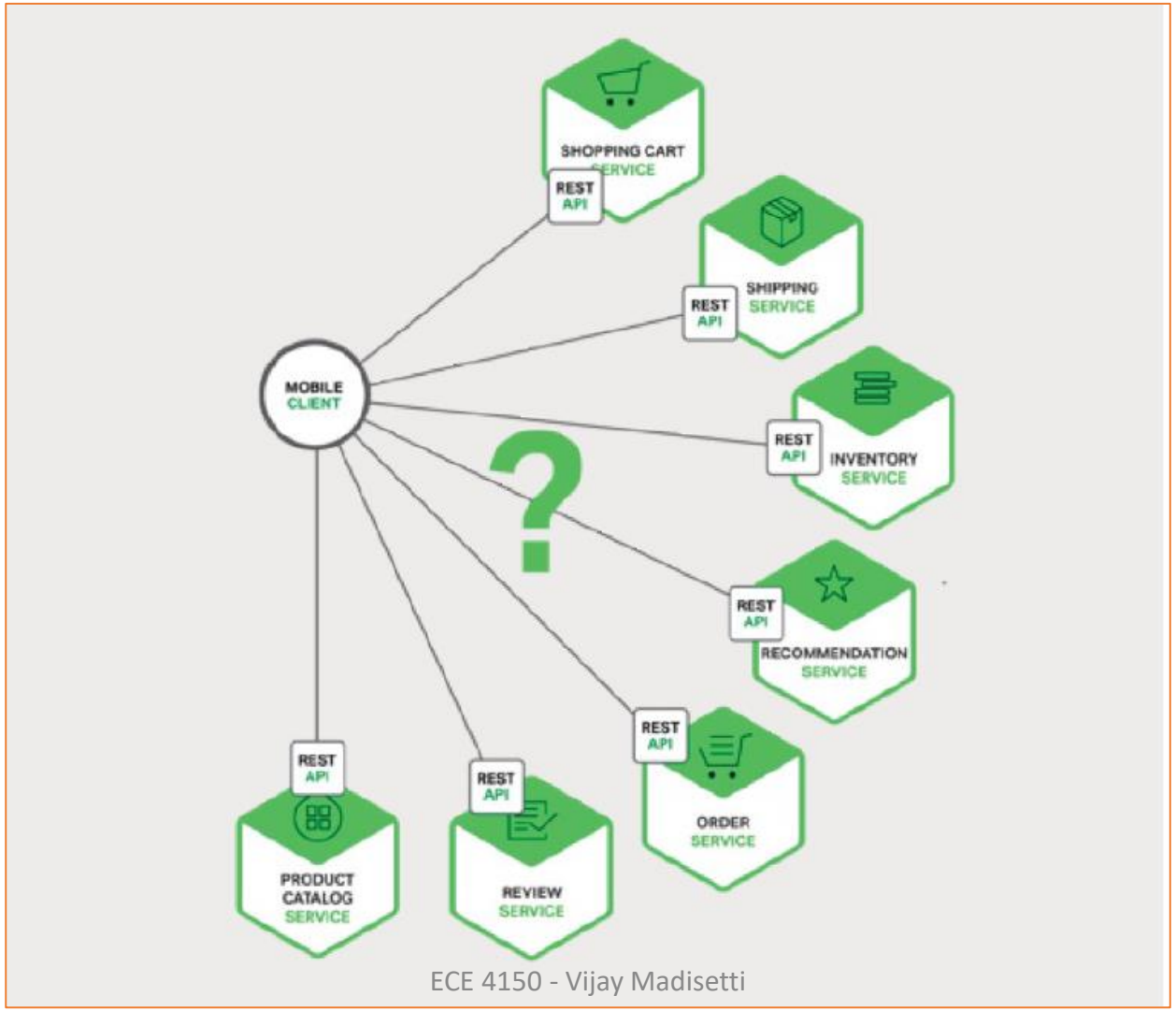
Microservices-based decomposition



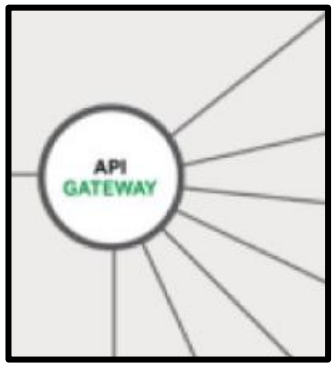
API Gateways

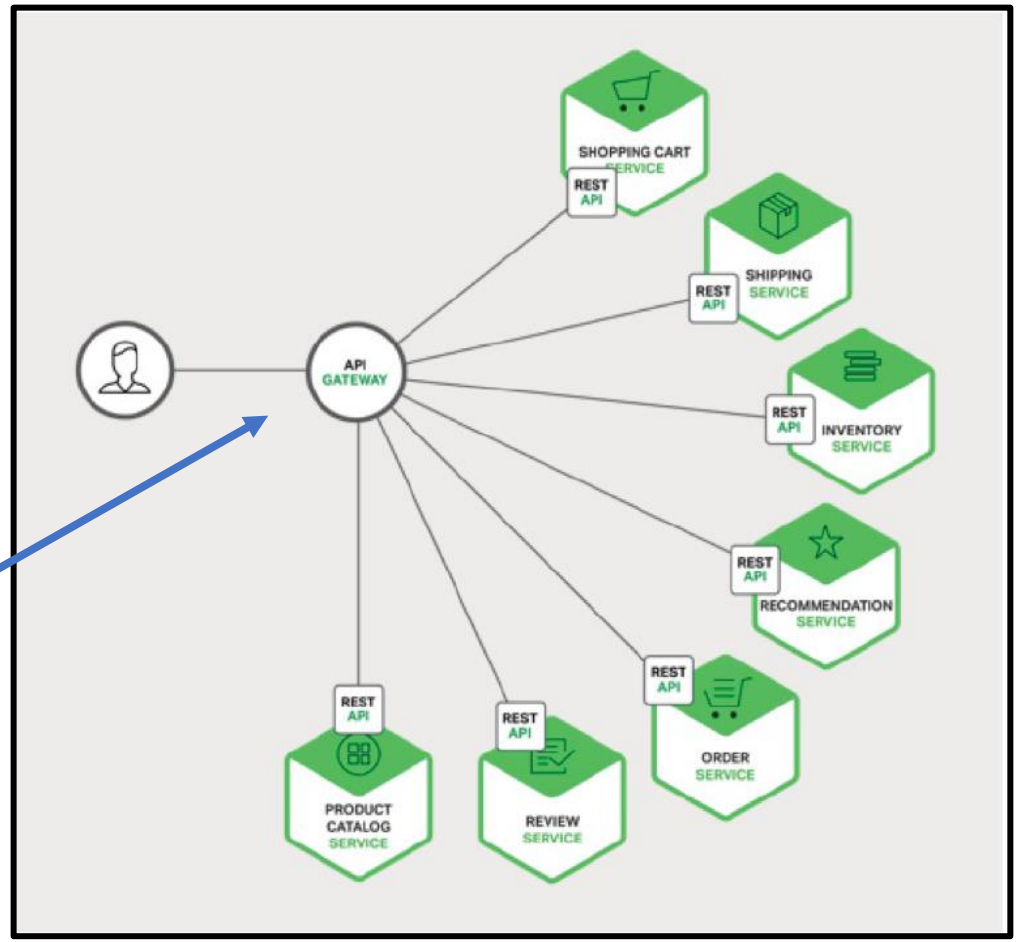
Important to Cloud Deployment

How to have single mobile UI interface with multiple “microservices” ? Option 1

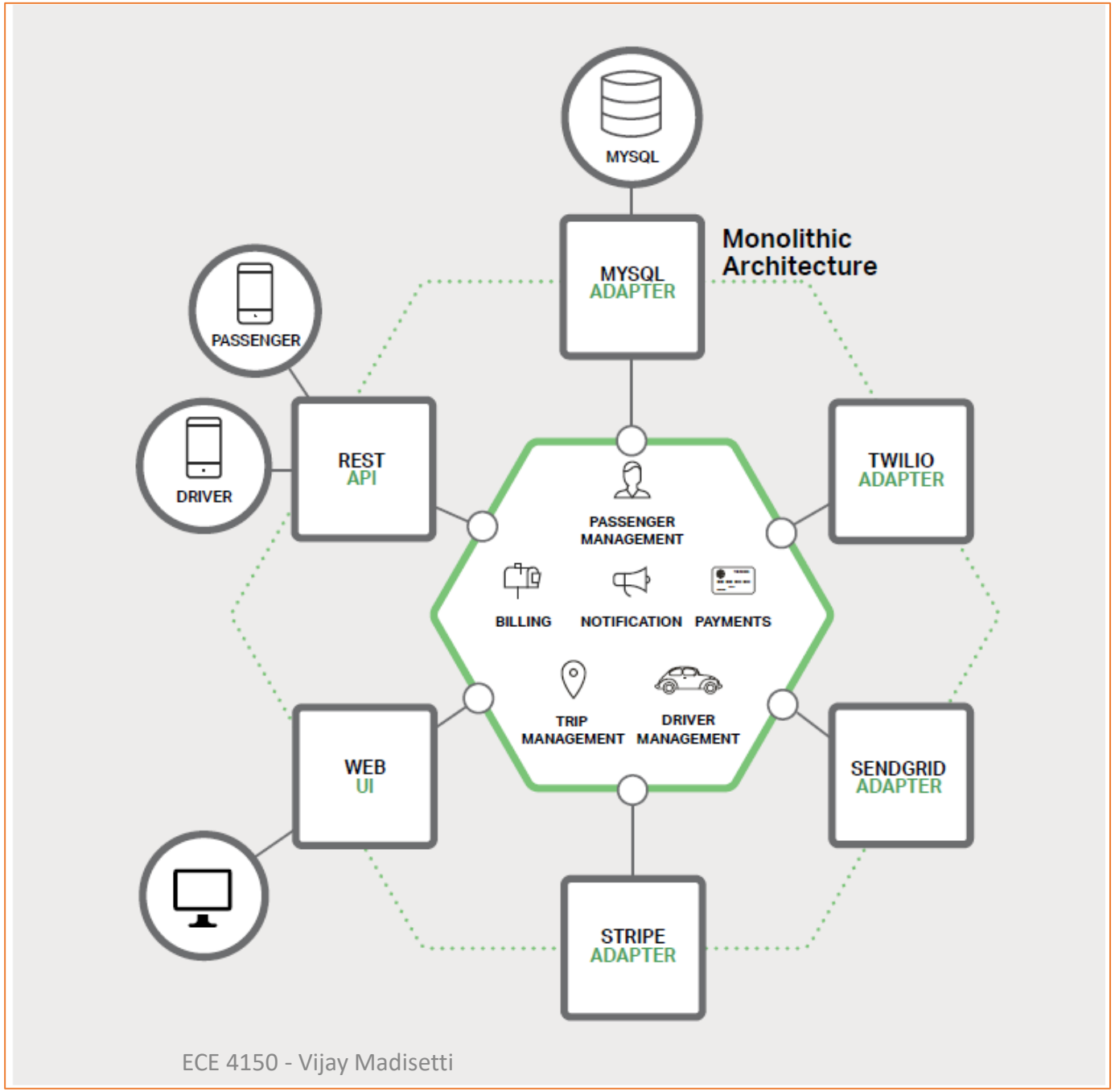


How to have the mobile UI interface with multiple microservices ? Option 2

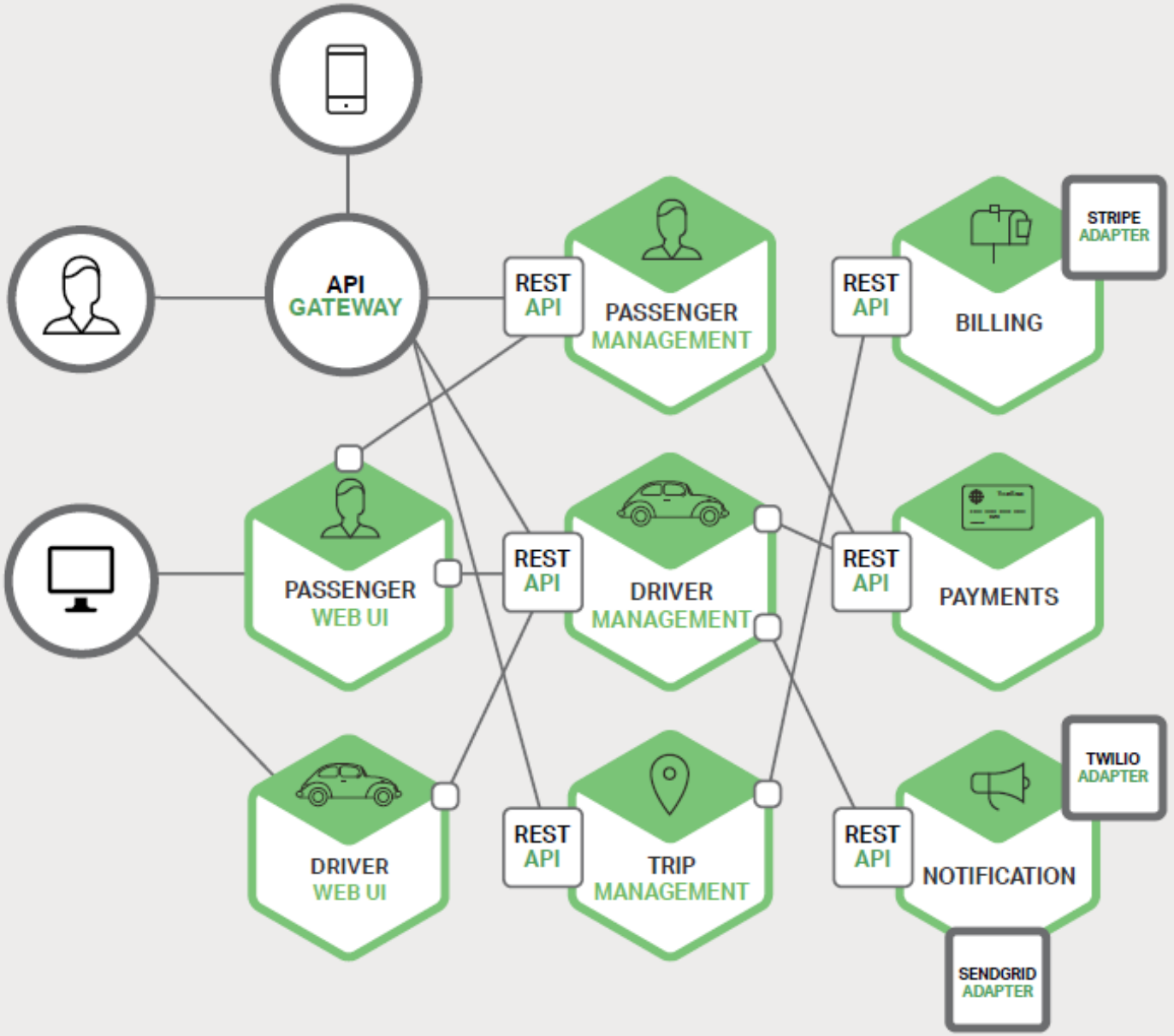


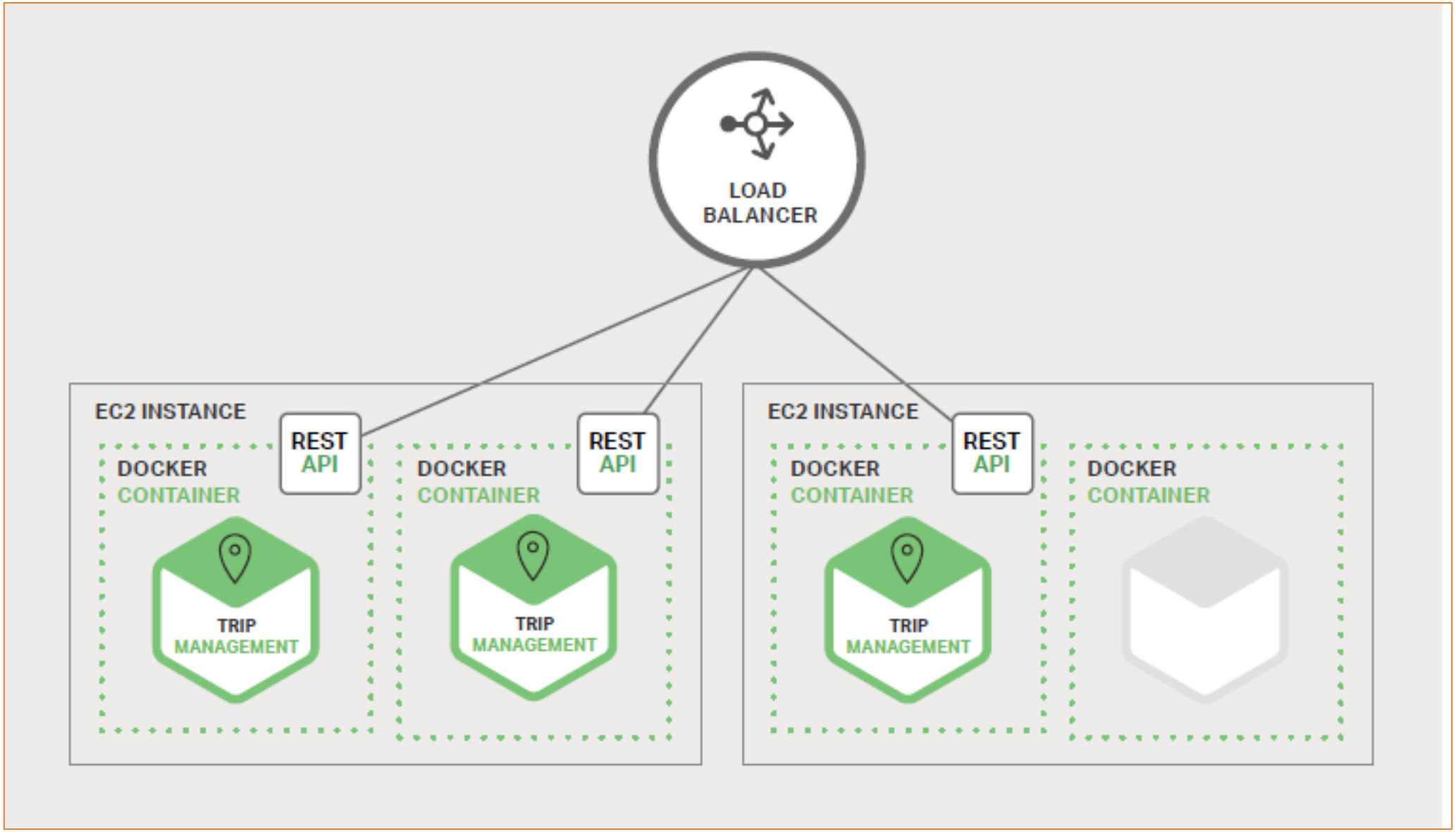


Uber-like Application - Analysis

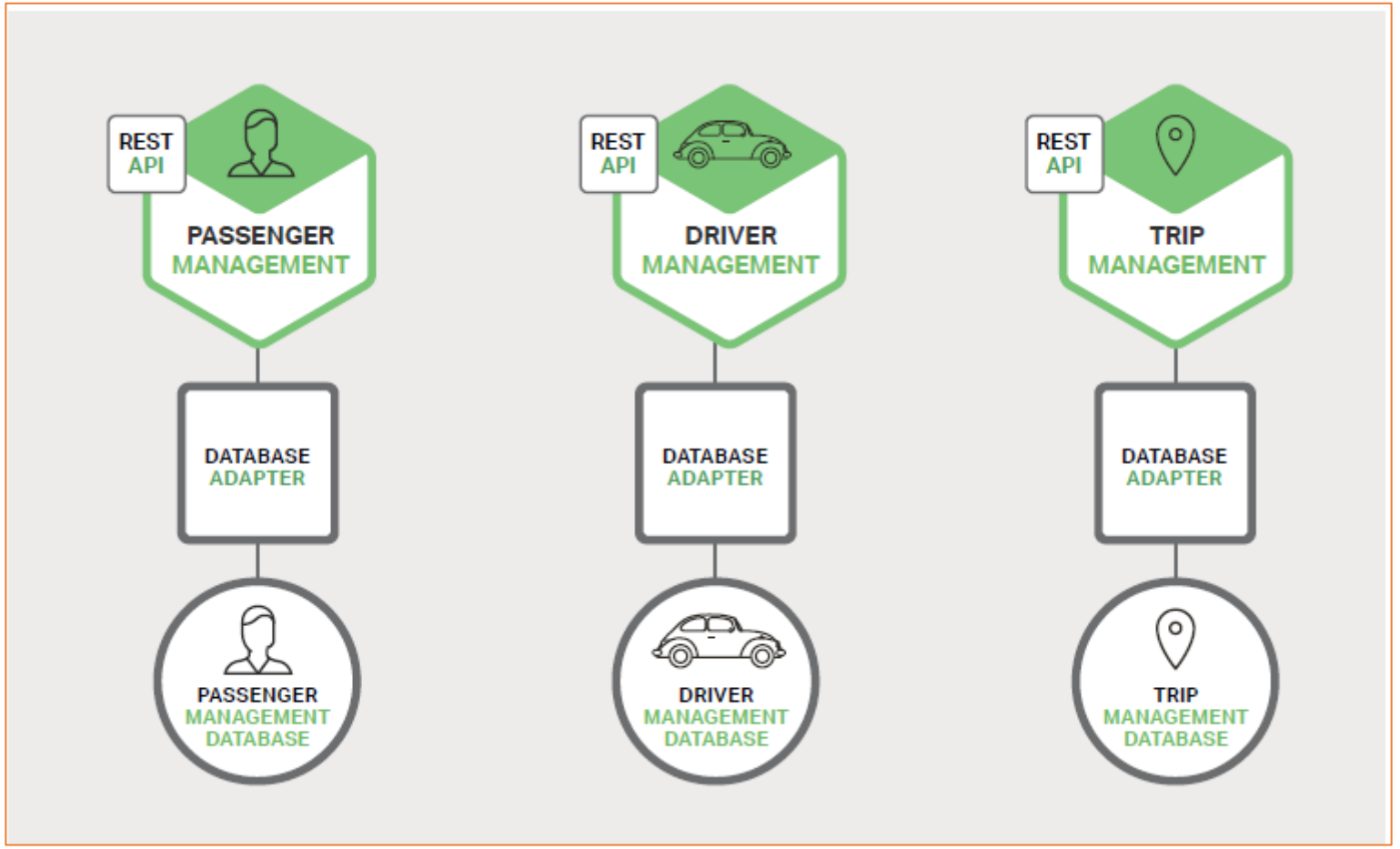


Microservices Architecture of Uber-like System




LOAD
BALANCER
EC2 INSTANCE
REST
API
DOCKER
CONTAINER
TRIP
MANAGEMENT
REST
API
DOCKER
CONTAINER
TRIP
MANAGEMENT
EC2 INSTANCE
REST
API
DOCKER
CONTAINER
TRIP
MANAGEMENT
DOCKER
CONTAINER

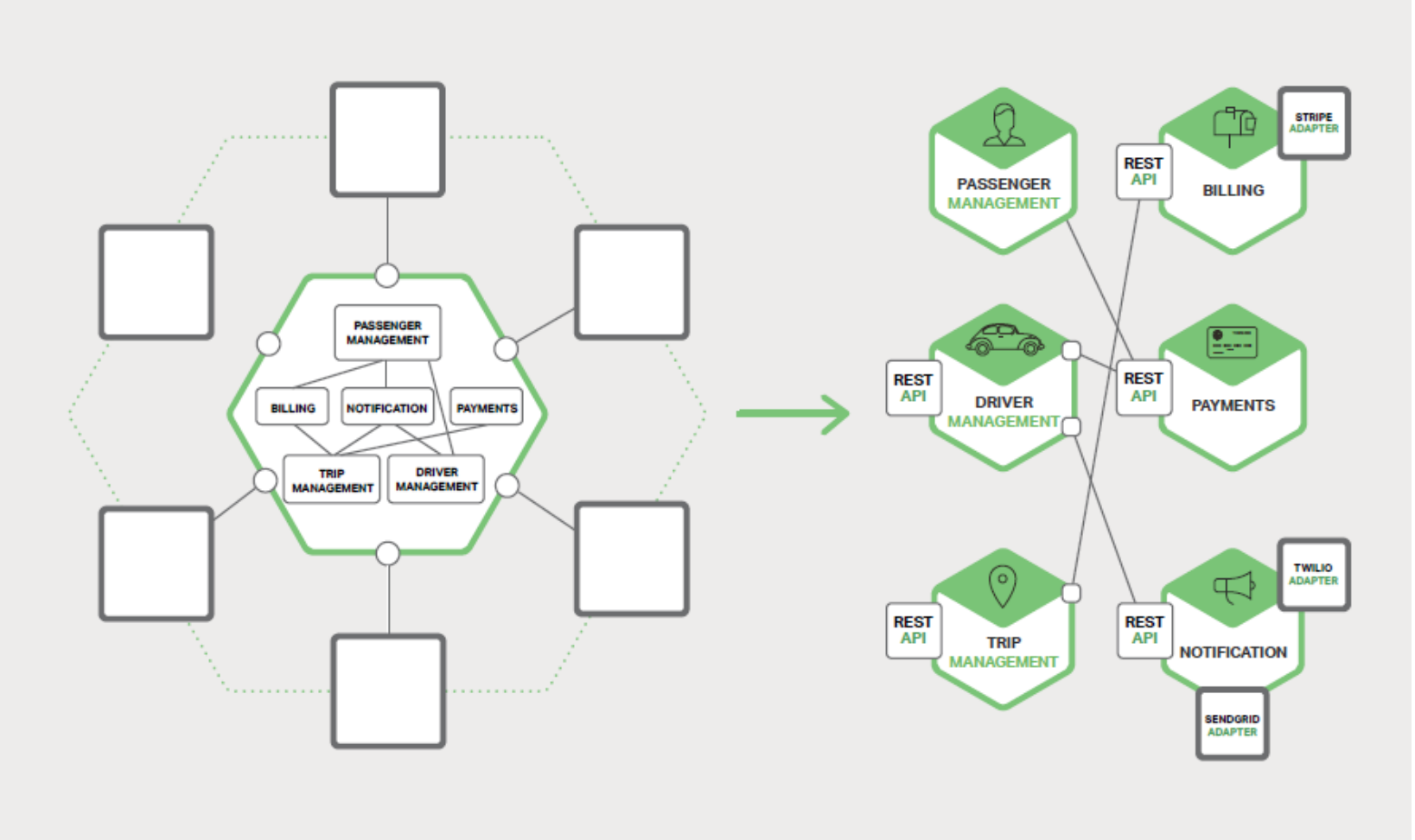
Database Architecture



Amazon.com implements hundreds of microservices

1. ORDER HISTORY
2. REVIEWS
3. BASIC PRODUCT INFO
4. RECOMMENDATION
5. INVENTORY
6. SHIPPING

How do Microservices Interact with Each Other ?

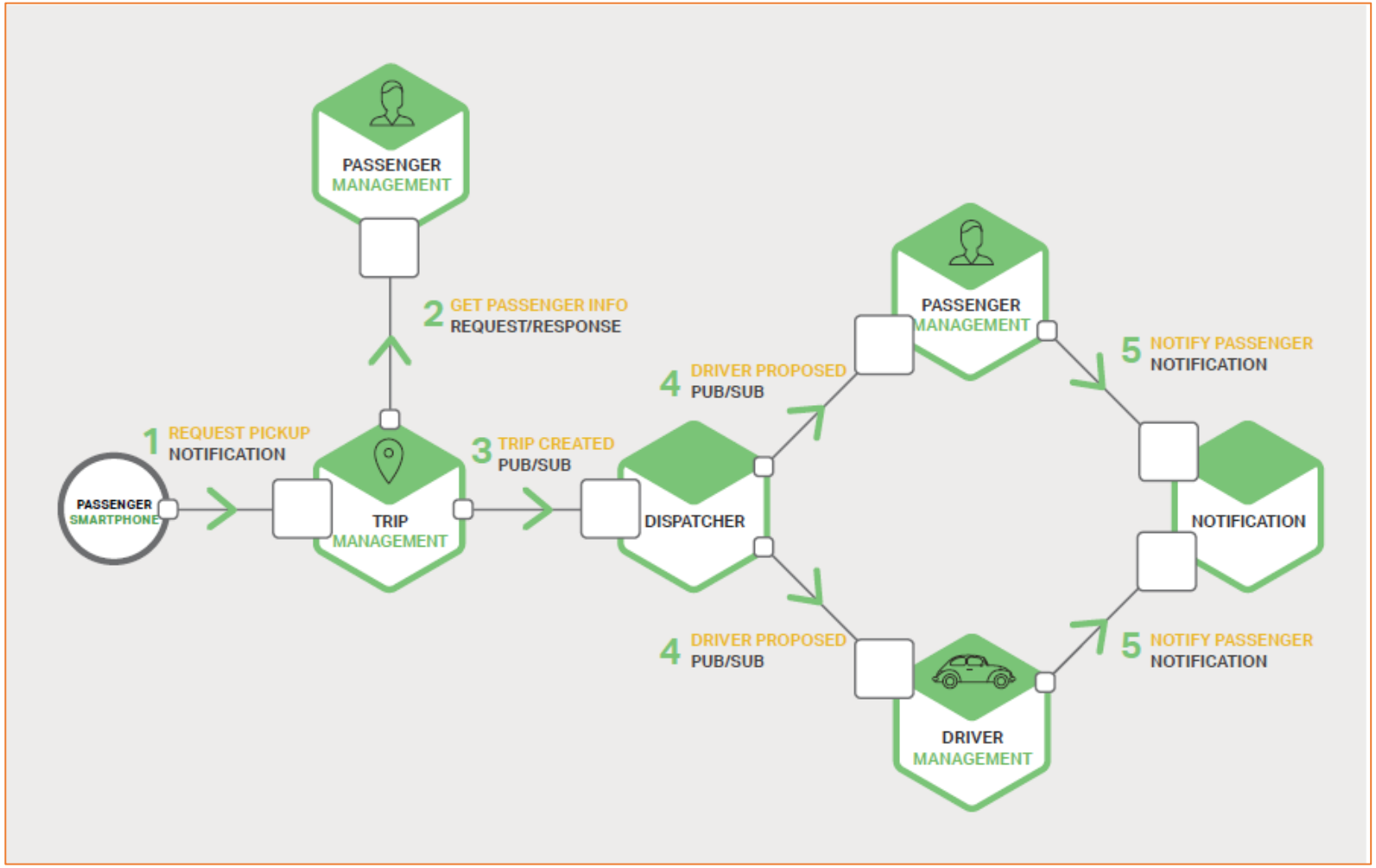


Microservice Interaction Styles

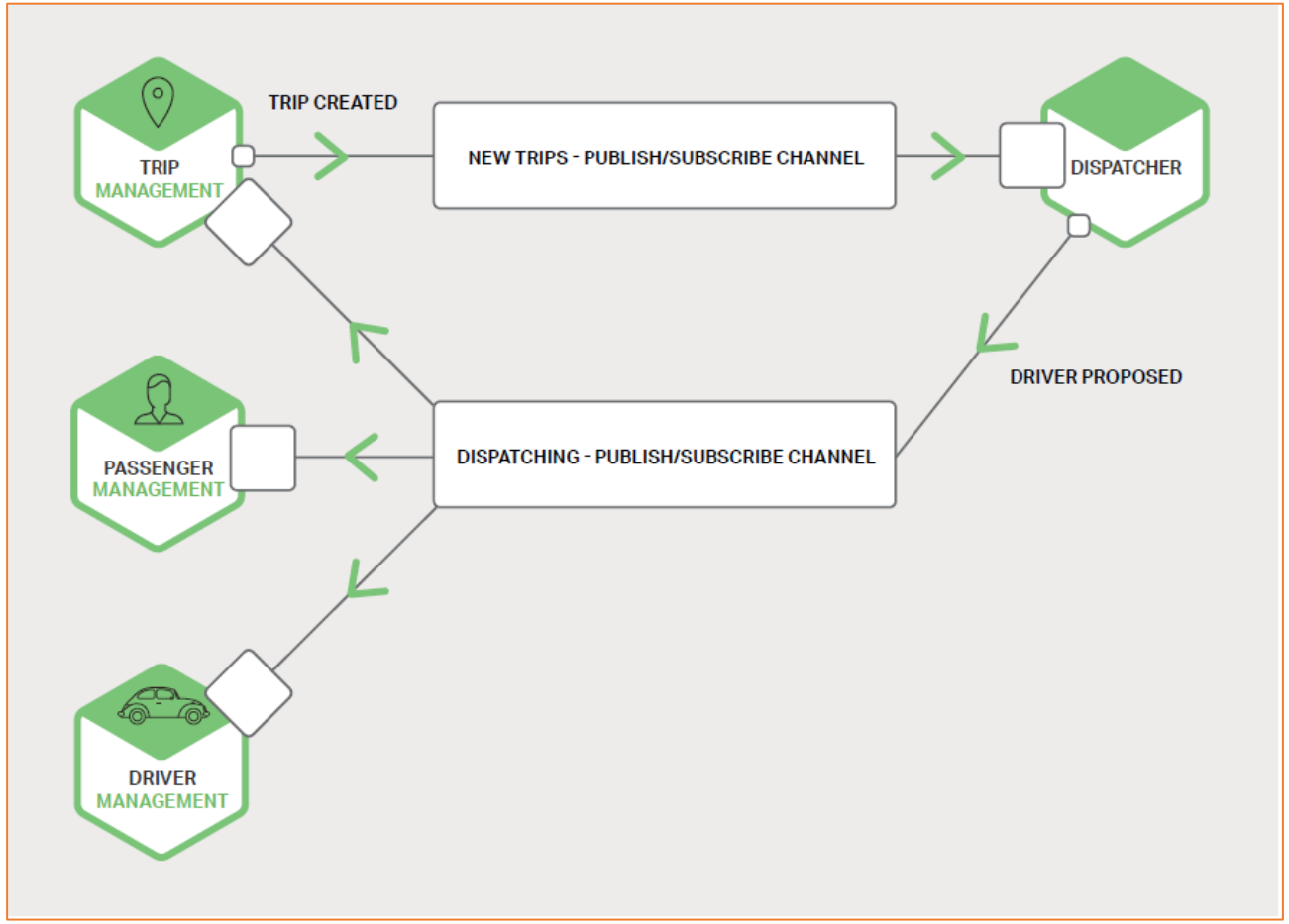
	ONE-TO-ONE	ONE-TO-MANY
SYNCHRONOUS	Request/response	—
ASYNCHRONOUS	Notification	Publish/subscribe
	Request/async response	Publish/async responses

Note: REST can be asynchronous. There is no reason a client need wait for a response In REST, it can do other things as well. It is a programming choice. A HTTP web server is usually Request/Response

Applying Interaction Styles to Microservices Architecture of Uber-like App

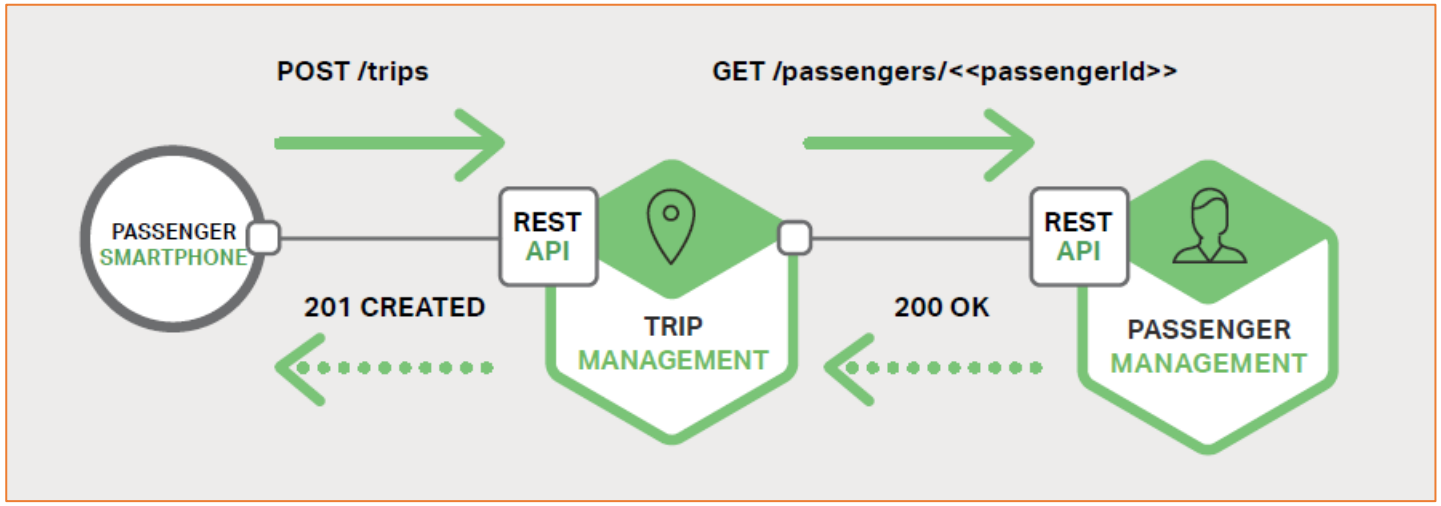


Messaging between Microservices



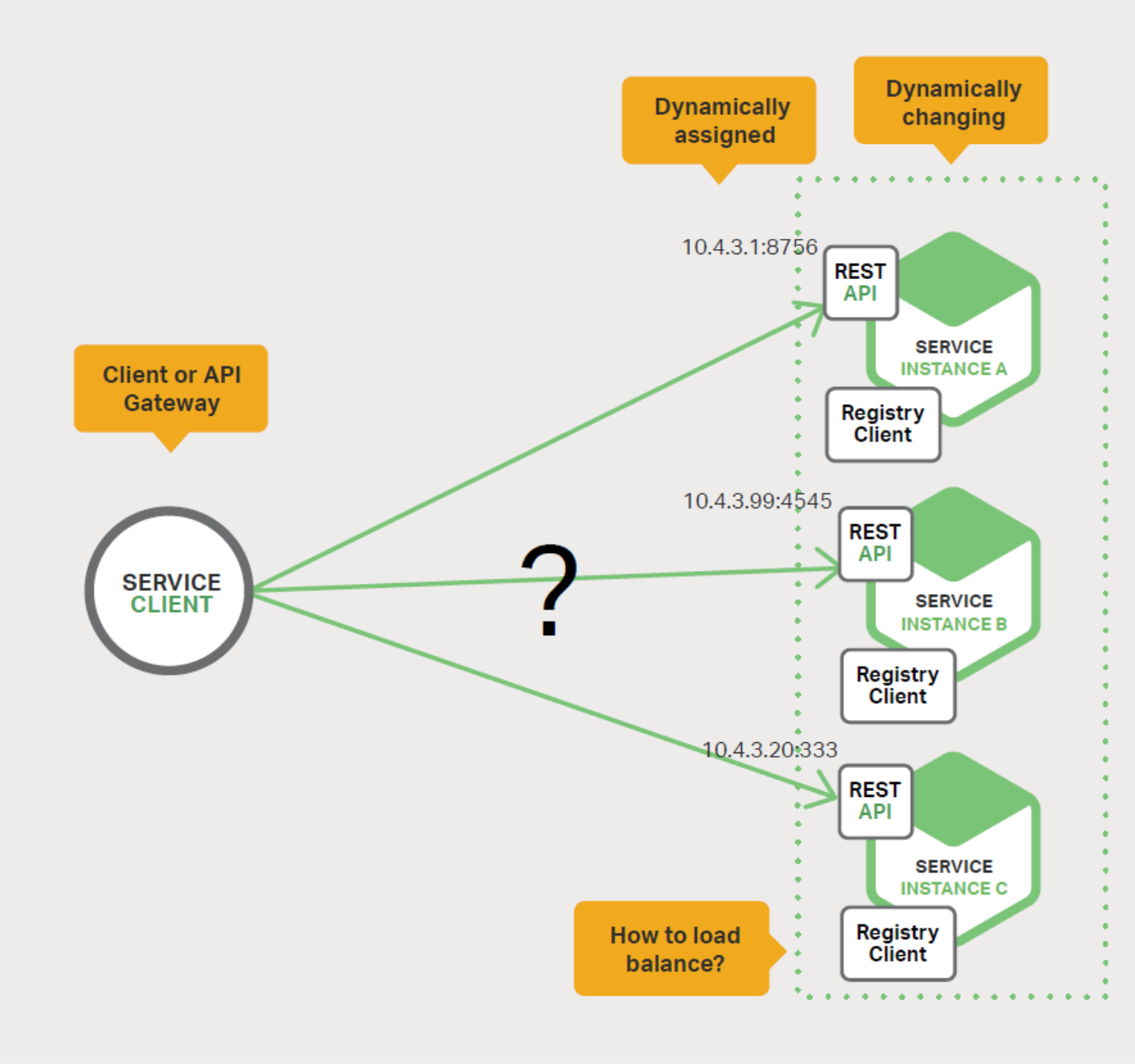
We already covered REST APIs in the class & labs

Note: Alternative to REST is the Apache Thrift API

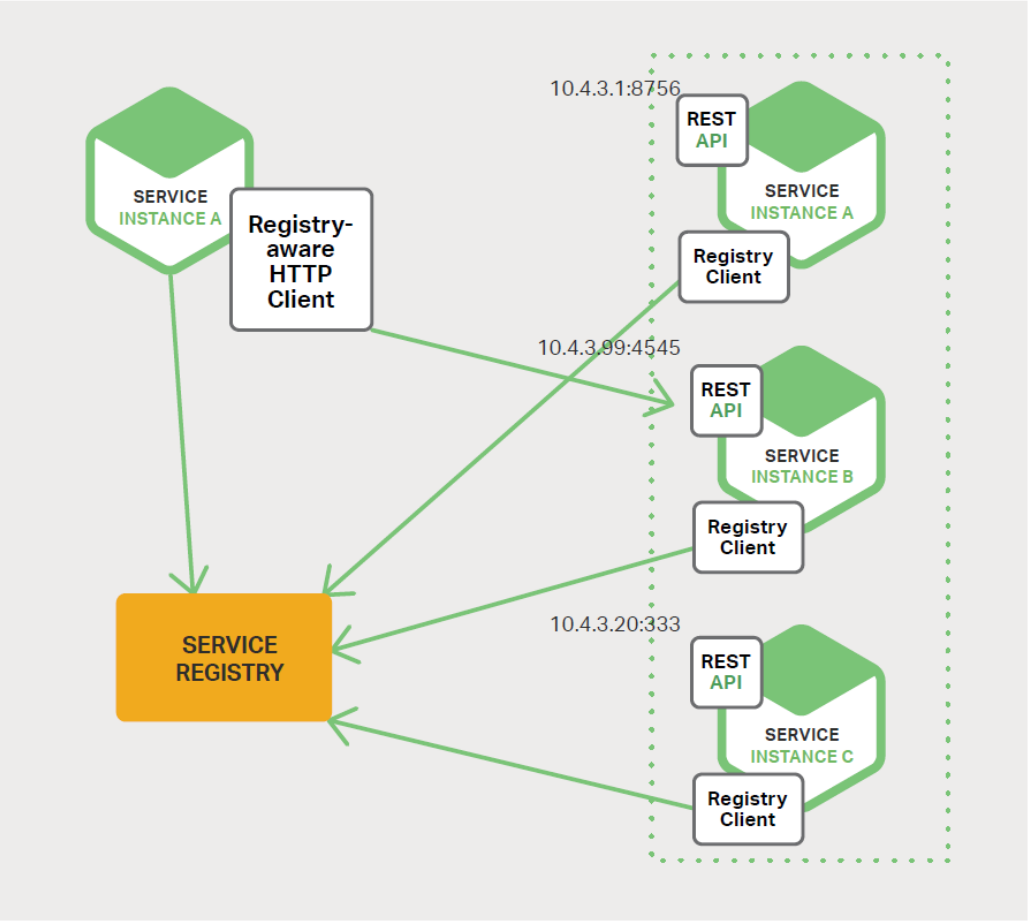


"REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems."
—Roy Fielding, *Architectural Styles and the Design of Network-based Software Architectures*

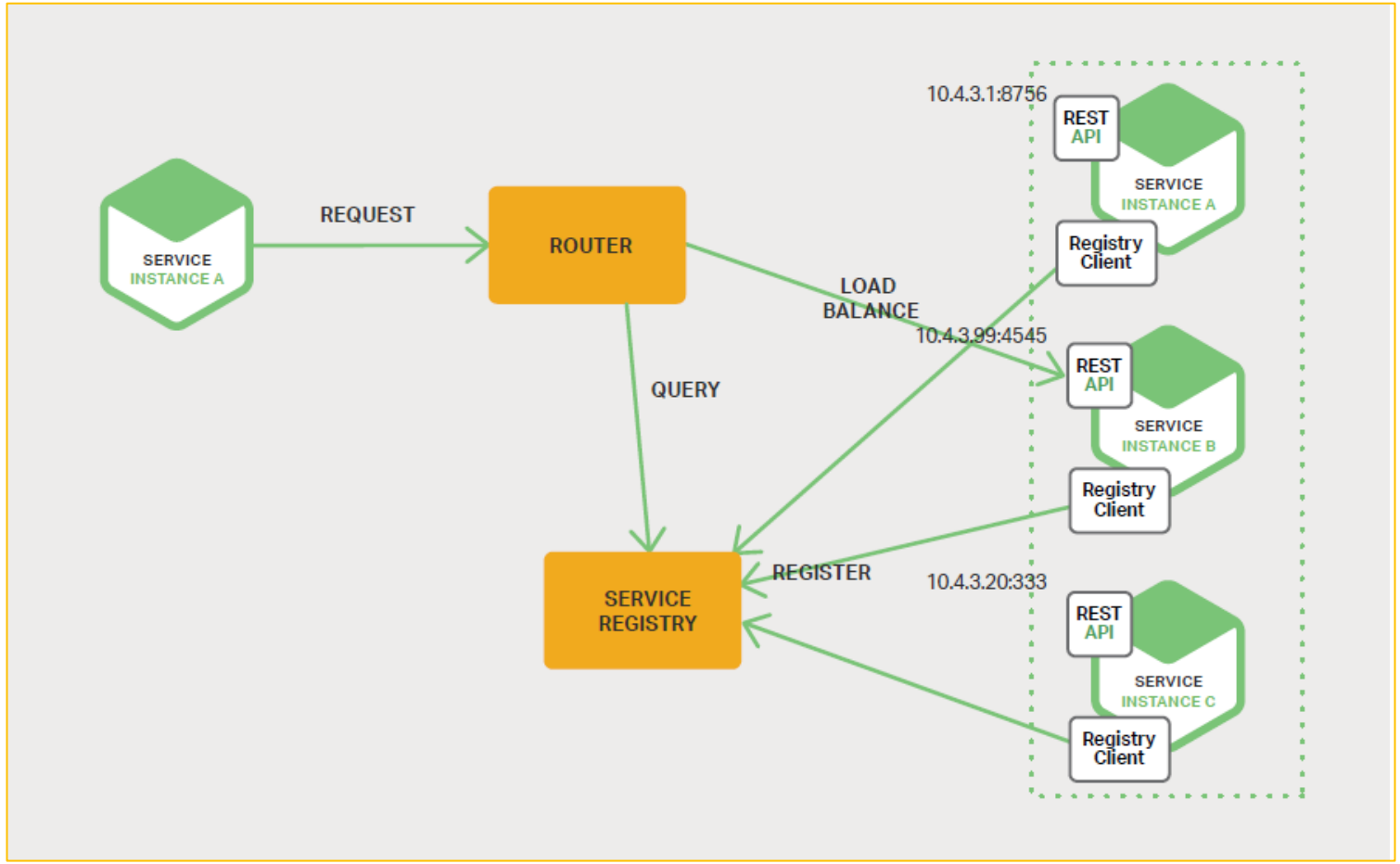
Service Discovery is Needed for Dynamic Environments



Option 1 – Client has to use a Service Registry



Option 2 – Server has to use a Service Registry



Netflix Microservices Architecture

Netflix Scale

- ~ 1/5 to 1/3 of the peak Internet traffic a day
- ~209M subscribers
- ~2+ Billion Edge API Requests/Day – 6 billions hours of streaming content each month.
- >500 MicroServices
- ~30 Engineering Teams (owning many microservices)

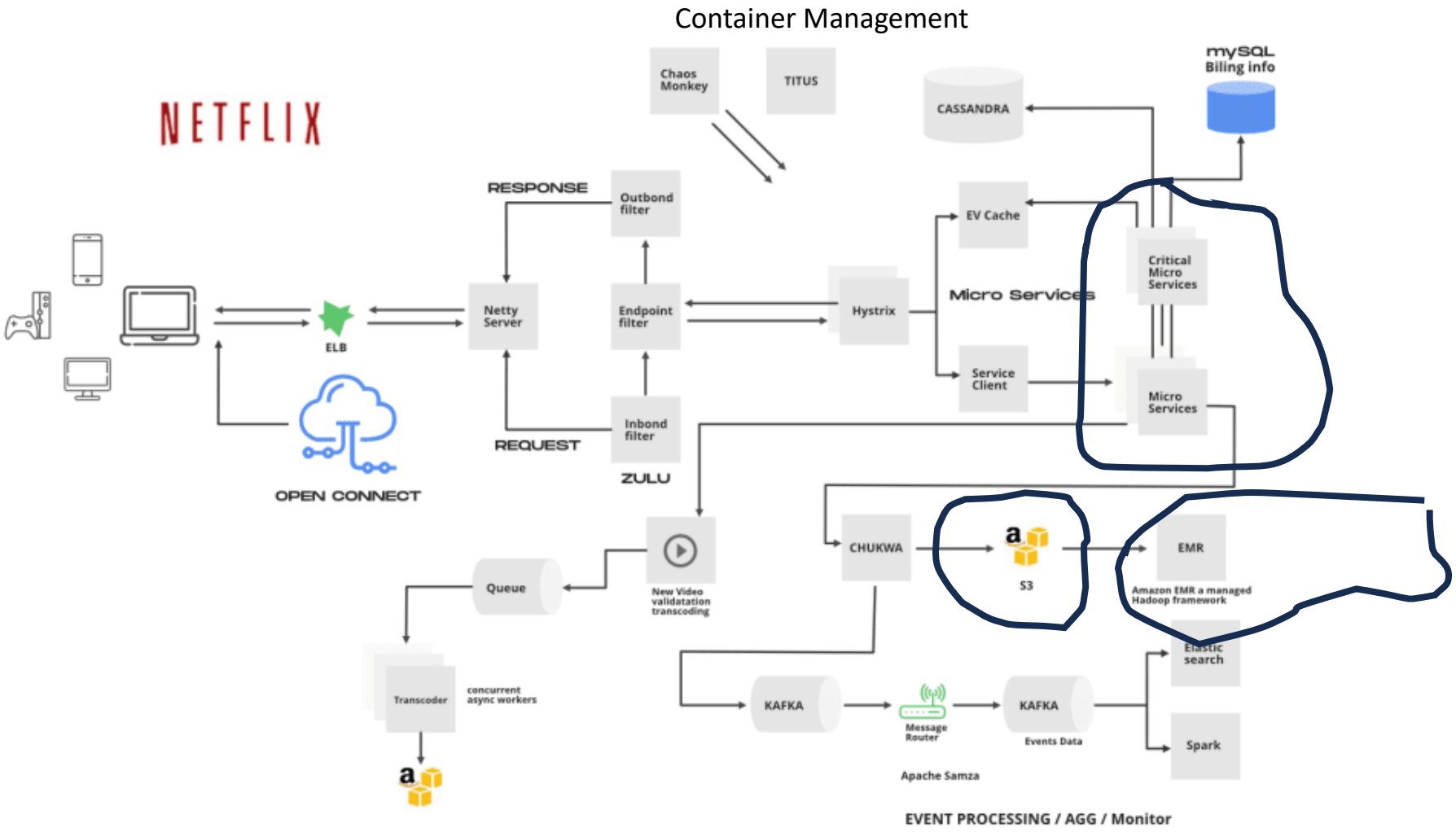
NETFLIX

Netflix - Evolution

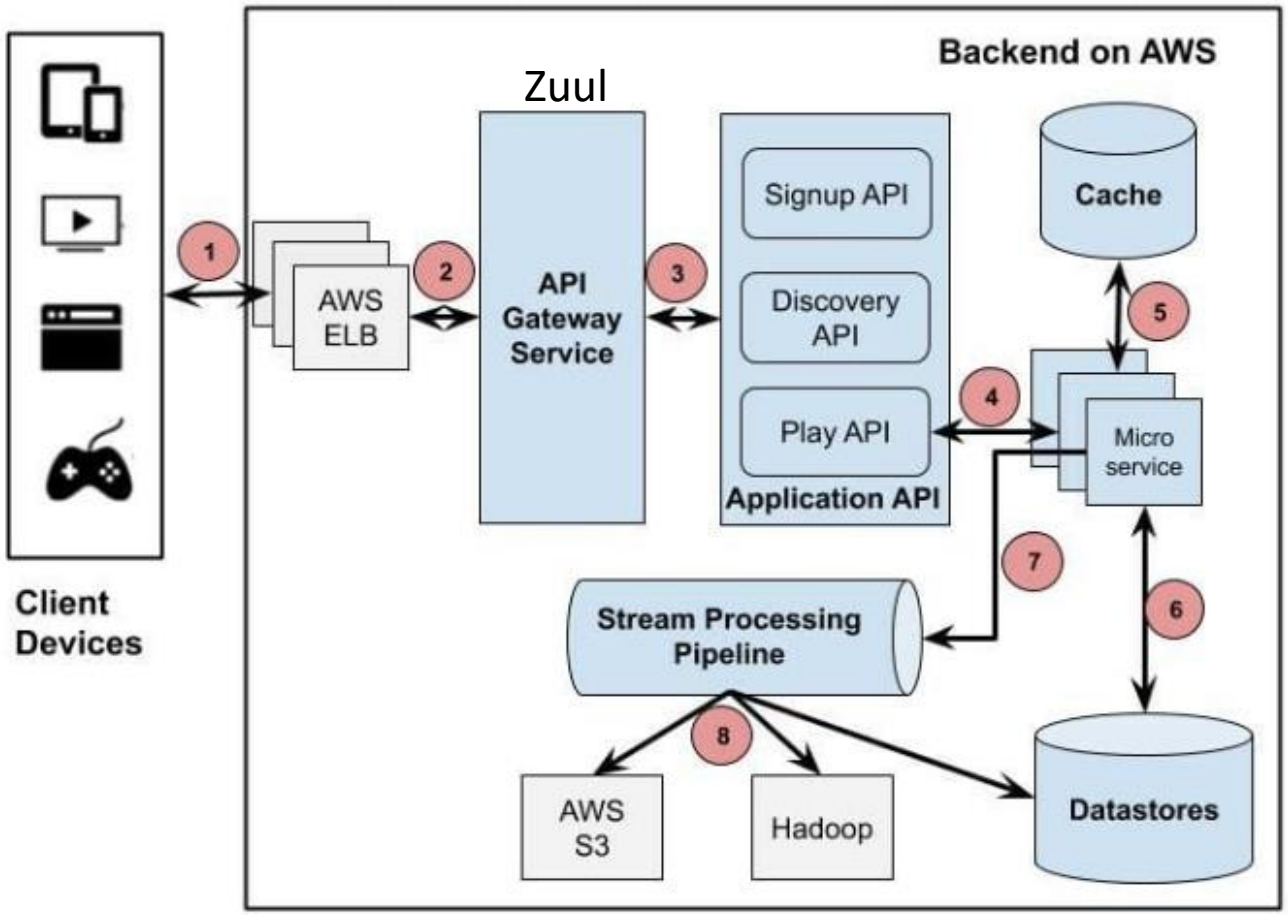
- Old DataCenter (2008)
- Everything in one WebApp (.war)

- AWS Cloud (~2010)
- 100s of Fine Grained Services

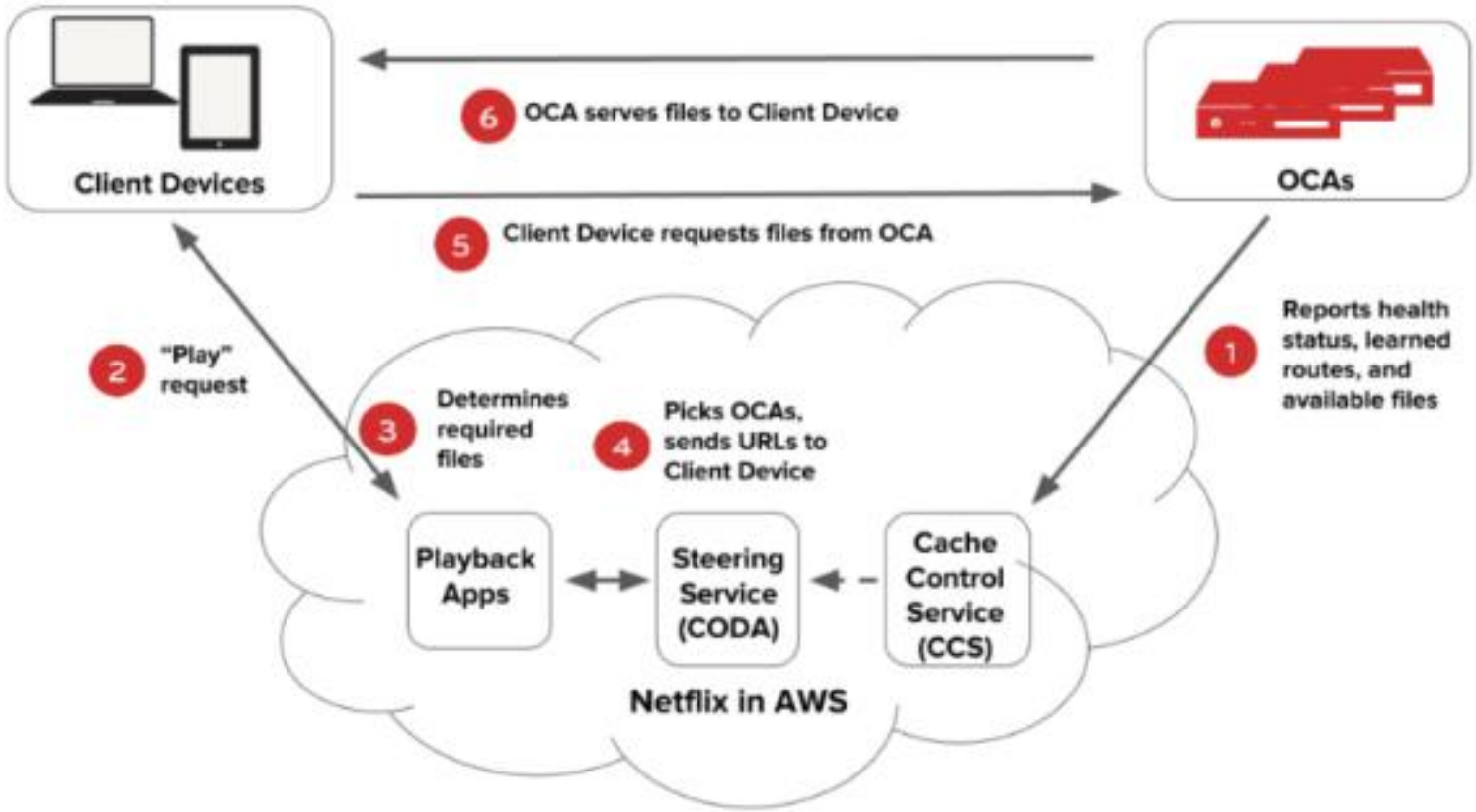
Netflix Cloud Architecture

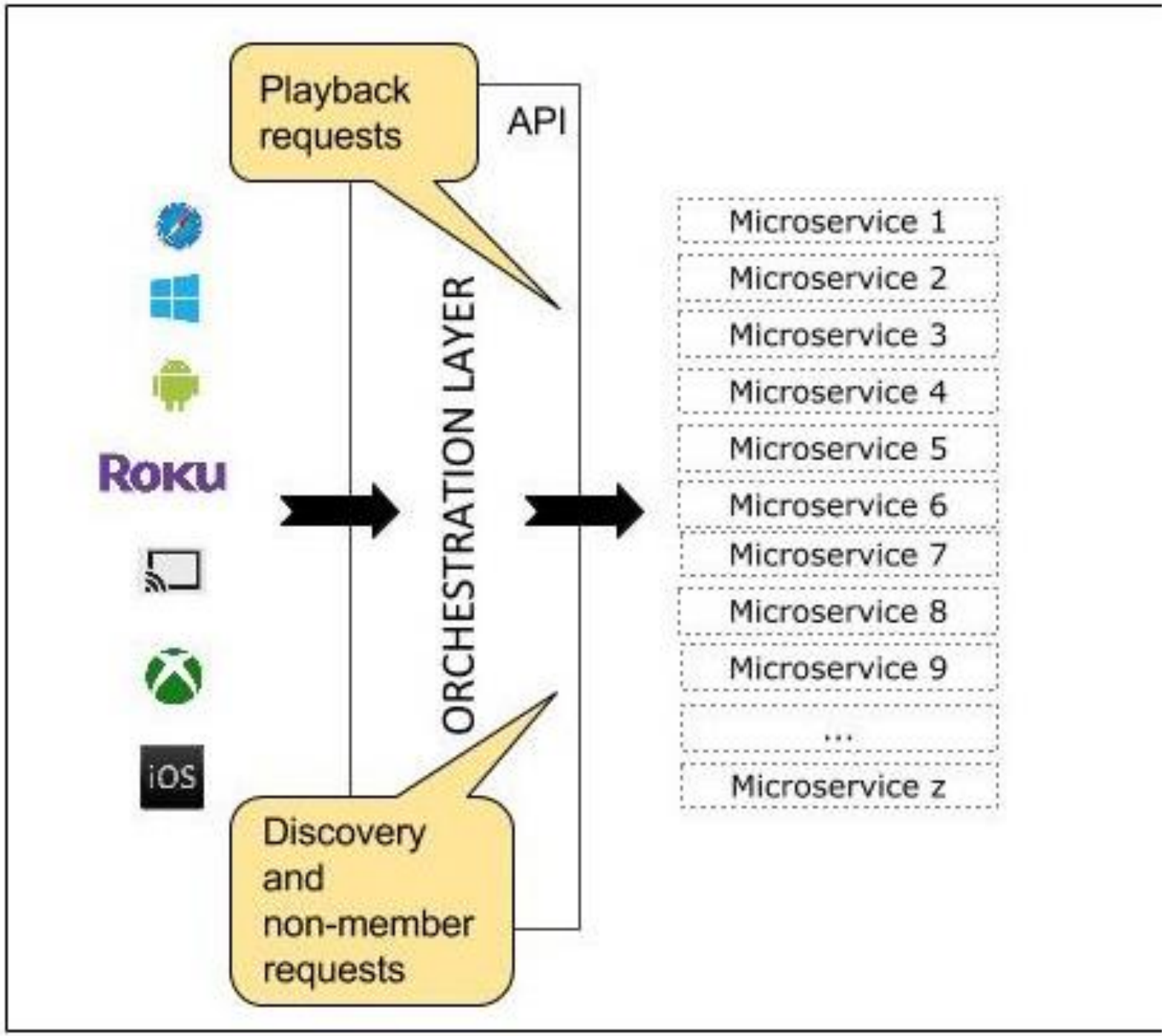


Slightly Simplified Netflix Architecture on AWS Cloud



Netflix OCA Architecture on AWS



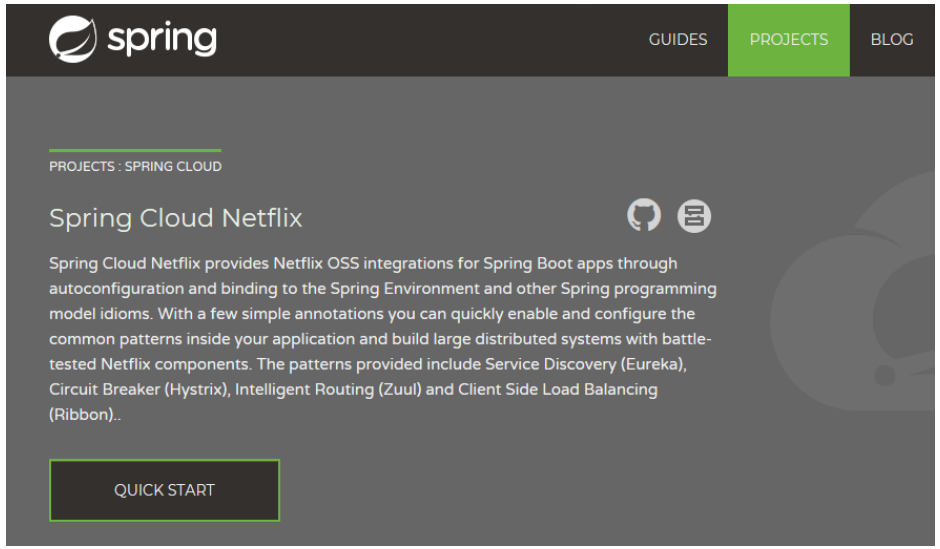


Netflix Microservices Architecture

Netflix Offers API Gateway with Service Registry - Eureka

https://cloud.spring.io/spring-cloud-netflix/

https://spring.io/guides

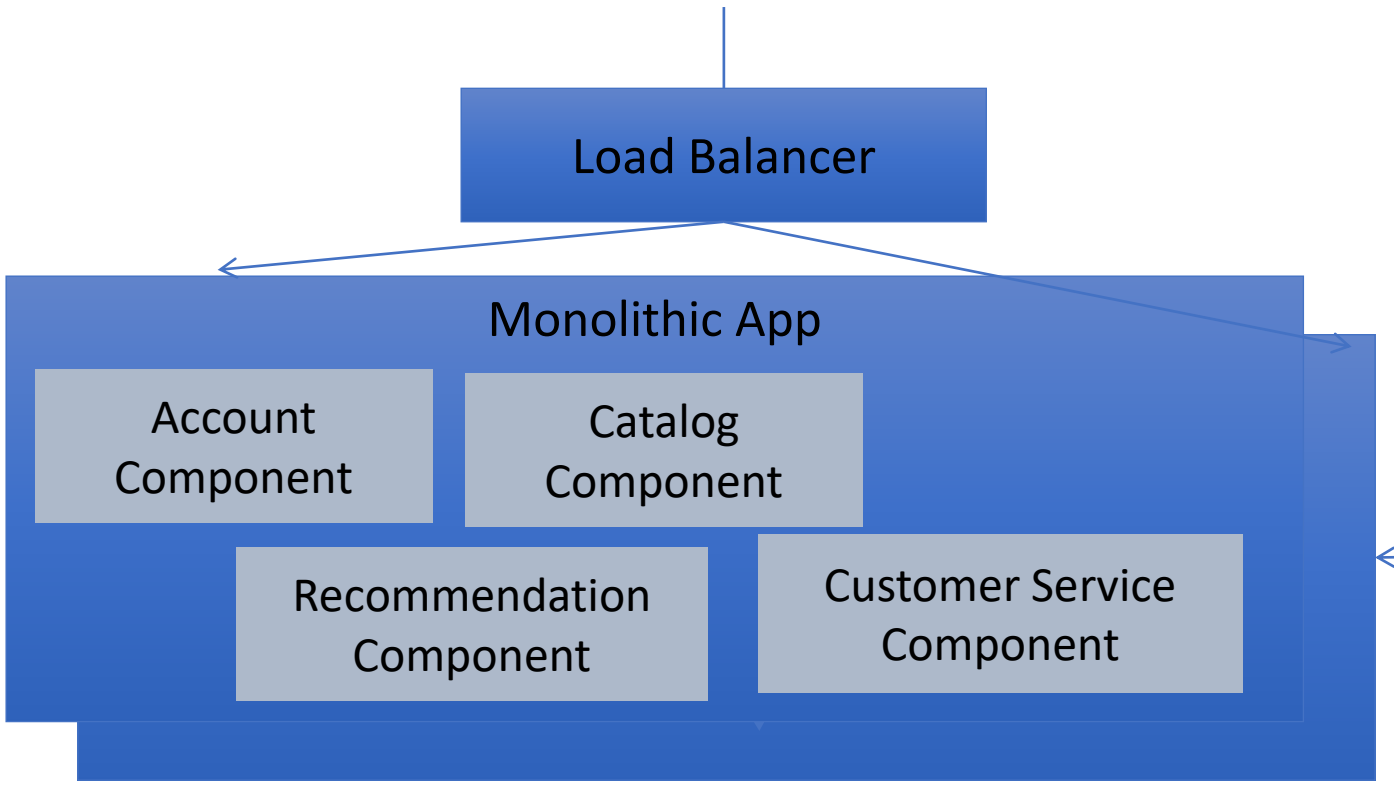


Features

Spring Cloud Netflix features:

- Service Discovery: Eureka instances can be registered and clients can discover the instances using Spring-managed beans
- Service Discovery: an embedded Eureka server can be created with declarative Java configuration
- Circuit Breaker: Hystrix clients can be built with a simple annotation-driven method decorator
- Circuit Breaker: embedded Hystrix dashboard with declarative Java configuration
- Declarative REST Client: Feign creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC annotations
- Client Side Load Balancer: Ribbon
- External Configuration: a bridge from the Spring Environment to Archaius (enables native configuration of Netflix components using Spring Boot conventions)
- Router and Filter: automatic regsitration of Zuul filters, and a simple convention over configuration approach to reverse proxy creation

Monolithic Architecture Netflix (prior 2010)



Microservices Architecture (2010+) on AWS

Load Balancer

API Gateway

Account Service

Catalog Service

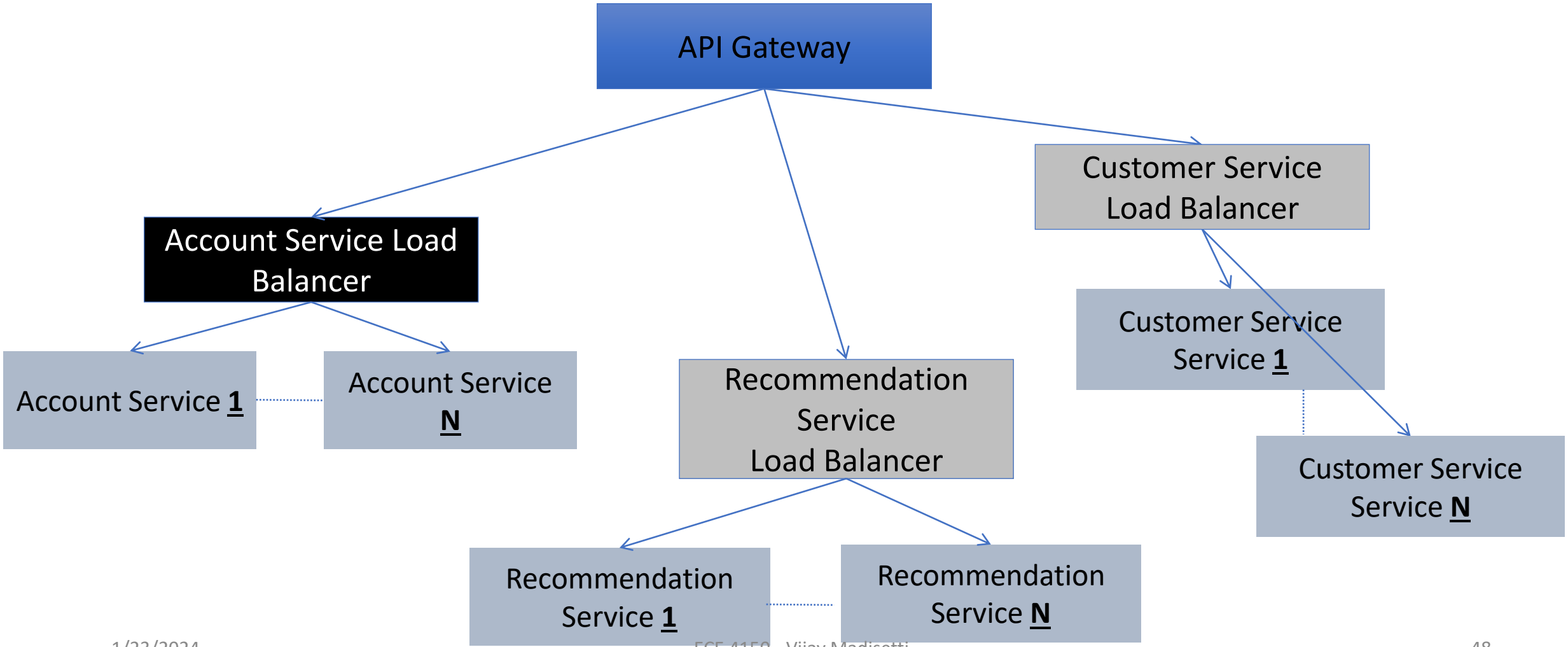
Catalog DB

Recommendation Service

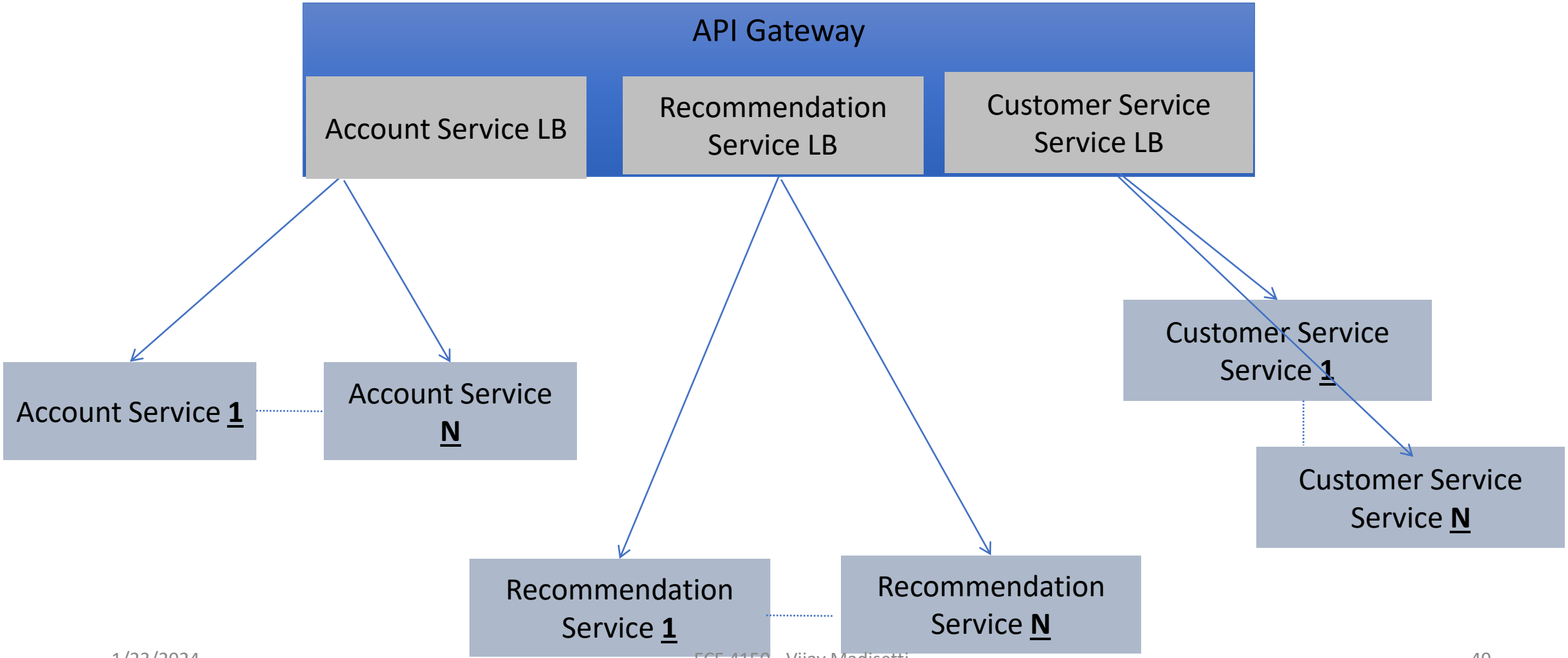
Customer Service Service

Customer DB

Central (Proxy) Loadbalancer

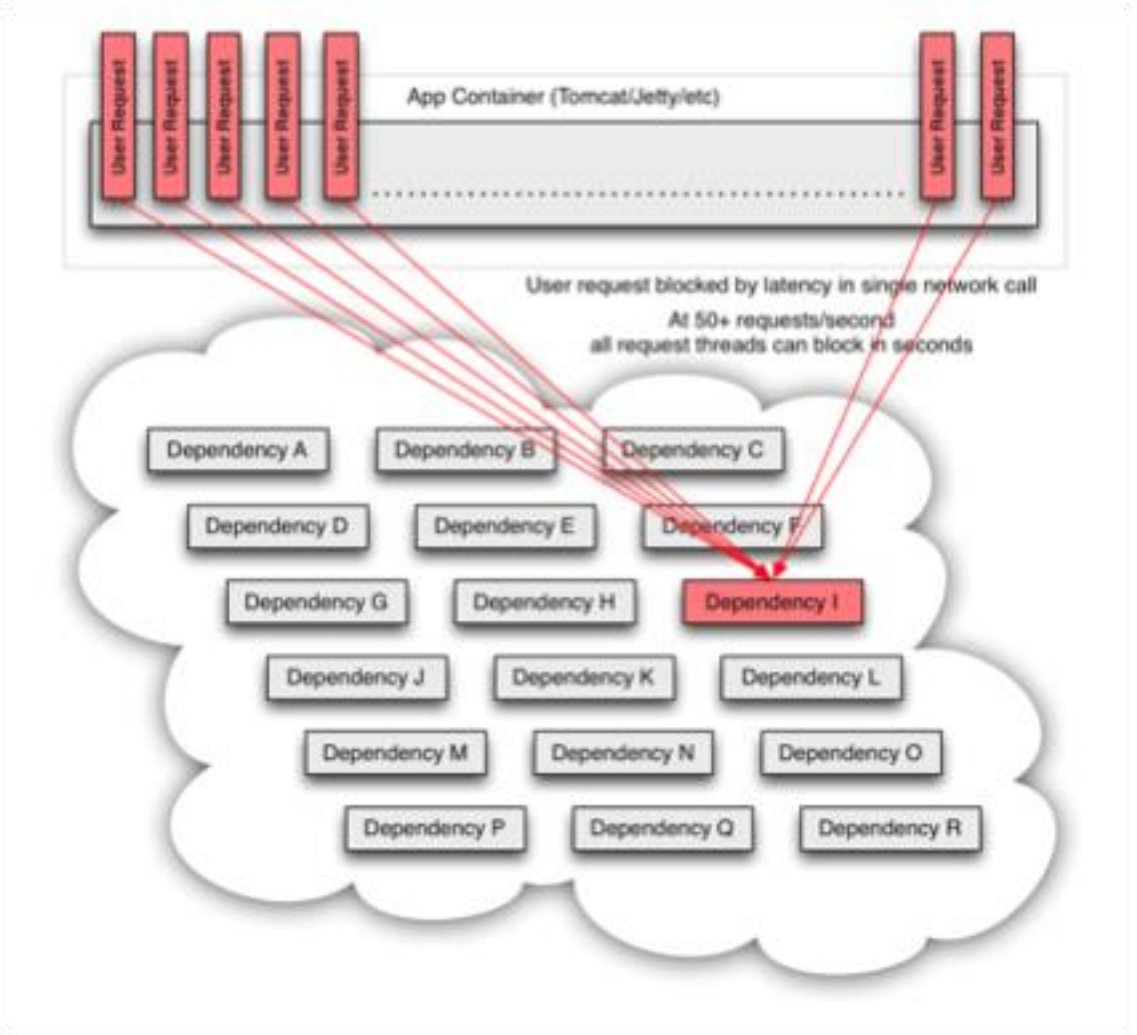


Client Loadbalancer

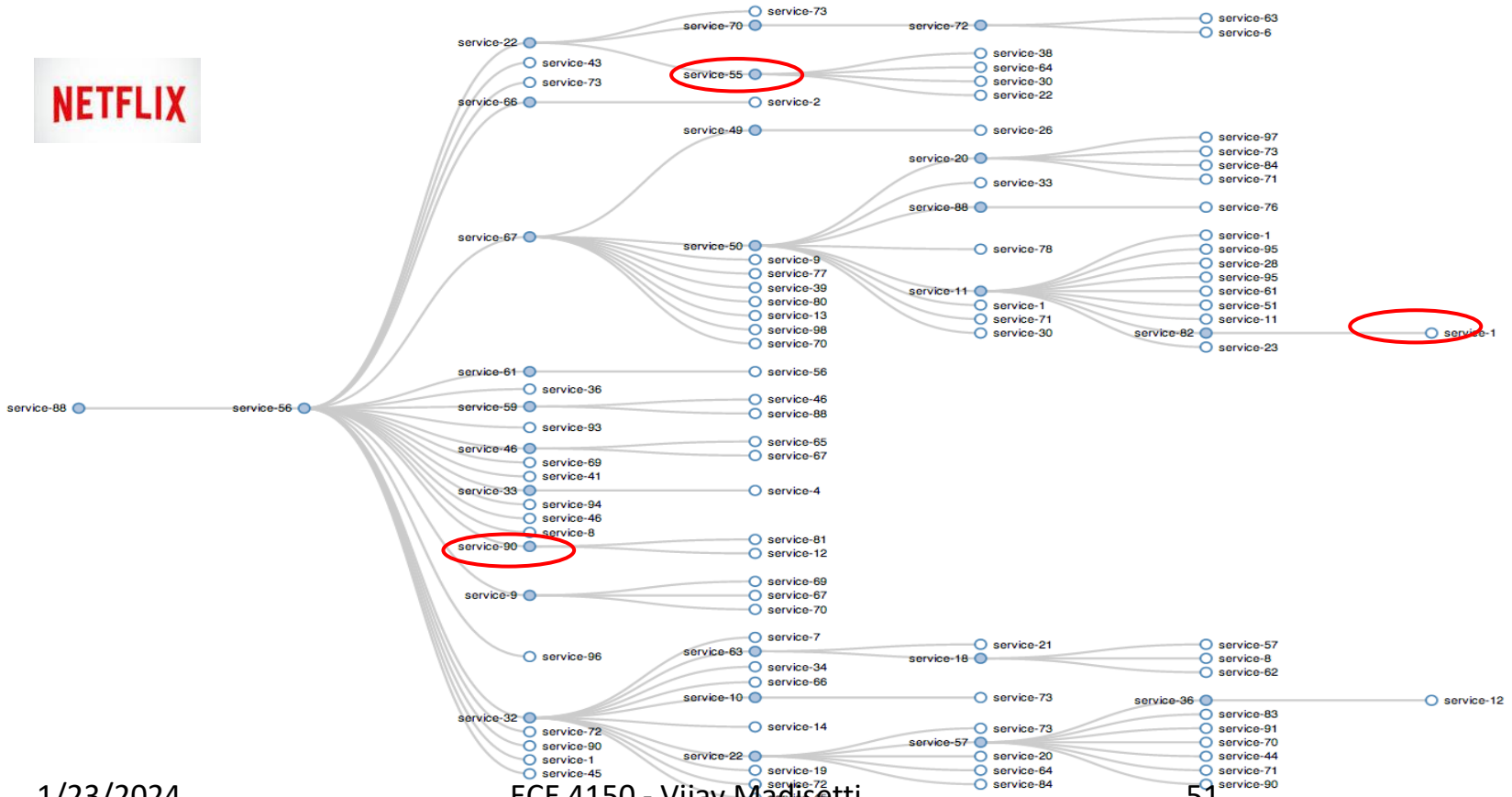


NETFLIX

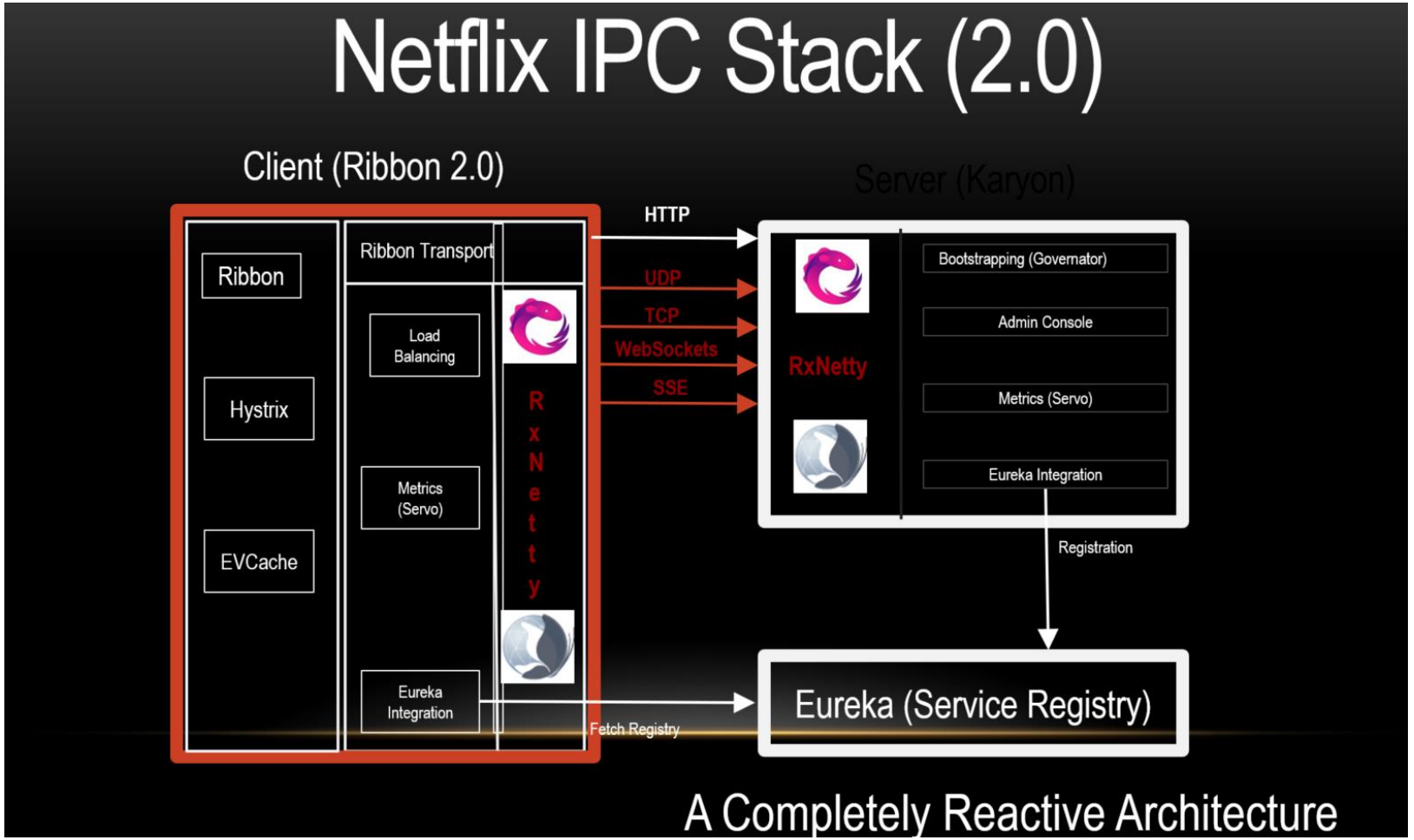
Single Microservice Can Be A Bottleneck

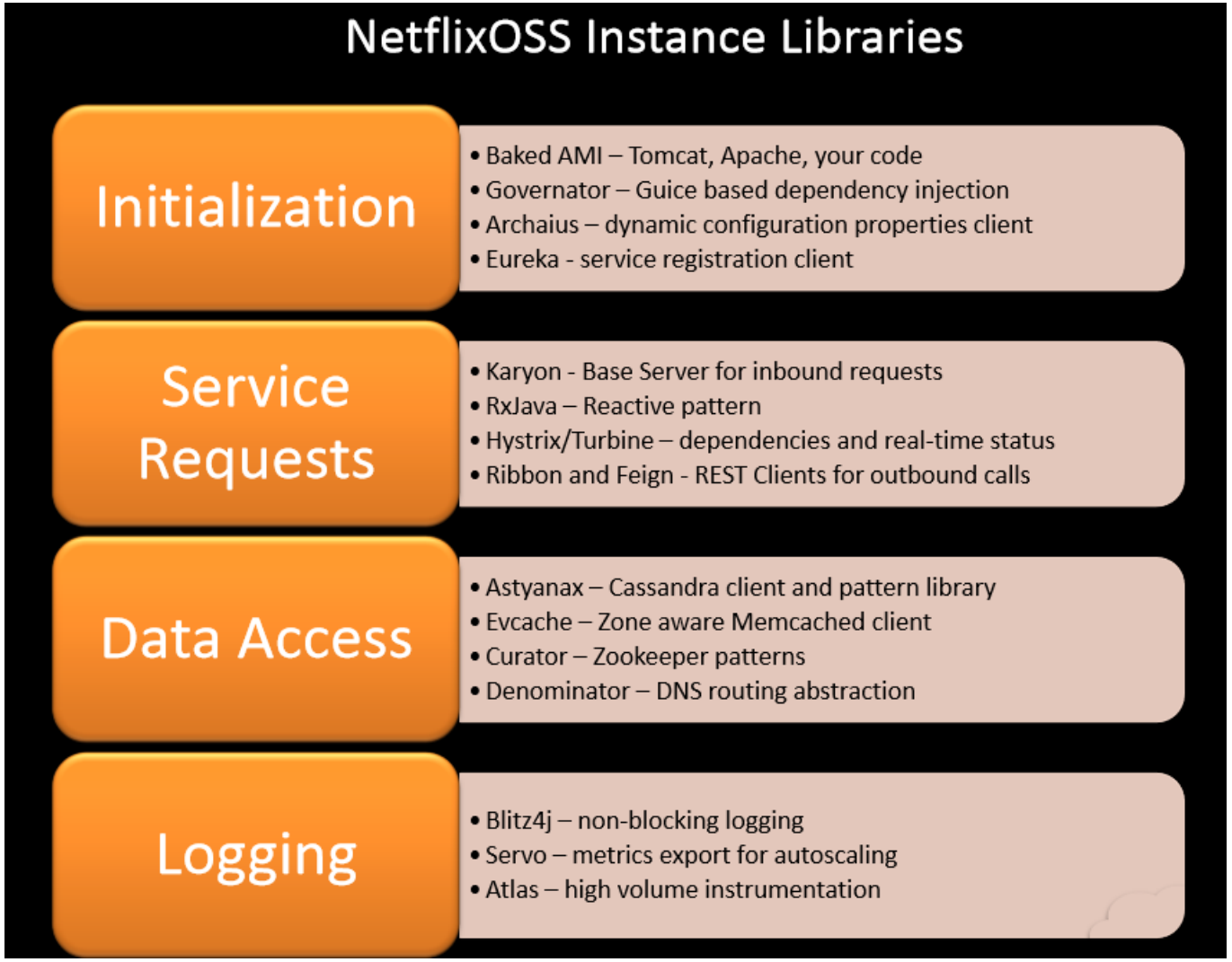


Test Resiliency – to dependencies



A Completely Reactive Architecture




NetflixOSS Instance Libraries
Initialization
• Baked AMI – Tomcat, Apache, your code
• Governator – Guice based dependency injection
• Archaius – dynamic configuration properties client
• Eureka - service registration client
Service Requests
• Karyon - Base Server for inbound requests
• RxJava – Reactive pattern
• Hystrix/Turbine – dependencies and real-time status
• Ribbon and Feign - REST Clients for outbound calls
Data Access
• Astyanax – Cassandra client and pattern library
• Evcache – Zone aware Memcached client
• Curator – Zookeeper patterns
• Denominator – DNS routing abstraction
Logging
• Blitz4j – non-blocking logging
• Servo – metrics export for autoscaling
• Atlas – high volume instrumentation

NETFLIX

- Eureka – for Service Registry/Discovery
- Karyon – for Server (Reactive or threaded/servlet container based)
- Ribbon – for IPC Client
  And Fault Tolerant Smart LoadBalancer
- Hystrix – for Fault Tolerance and Resiliency
- Archaius – for distributed/dynamic Properties
- Servo – unified Feature rich Metrics/Insight
- EVCache – for distributed cache
- Curator/Exhibitor – for zookeeper based operations

Summary

- Microservices are a newer model of cloud application development that have at least two flavors – serverless and container-based.
- Netflix uses microservices (container-based) and McDonald's uses microservices (Lambda services or serverless)
- Lab 1 shows microservices using serverless model, while Lab 5 will focus on microservices using container model. Lab 2 will focus on an intermediate step towards the container model using Flask-based Microservices development.