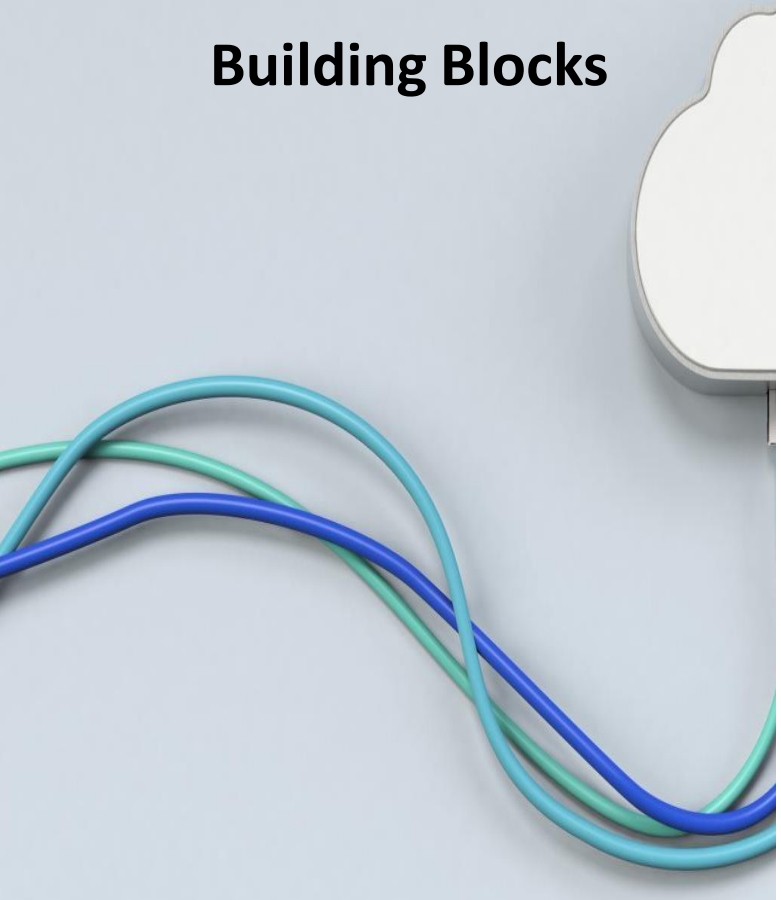


“Feature by Feature” Microservices Development for Cloud

- **ECE 4150**
- Spring 2024
- **Vijay Madisetti**

Objectives of this Lecture

Models for Deployment

Feature-by-Feature Development Models for Microservices-based Cloud Applications

Microservices-based architecture gives rise to best feature branch design metrics

Evolution of Architectural Paradigms

- Package-based architecture
- Layered N-Tier Architecture
- Message Bus Architecture
- Object-based Architecture
- Model-View-Controller (MVC) Architecture – ***Lab 2***
- Microservices-based Architecture

Reference Architectures –
e-Commerce, Business-to-Business, Banking and Financial apps

Load Balancing Tier

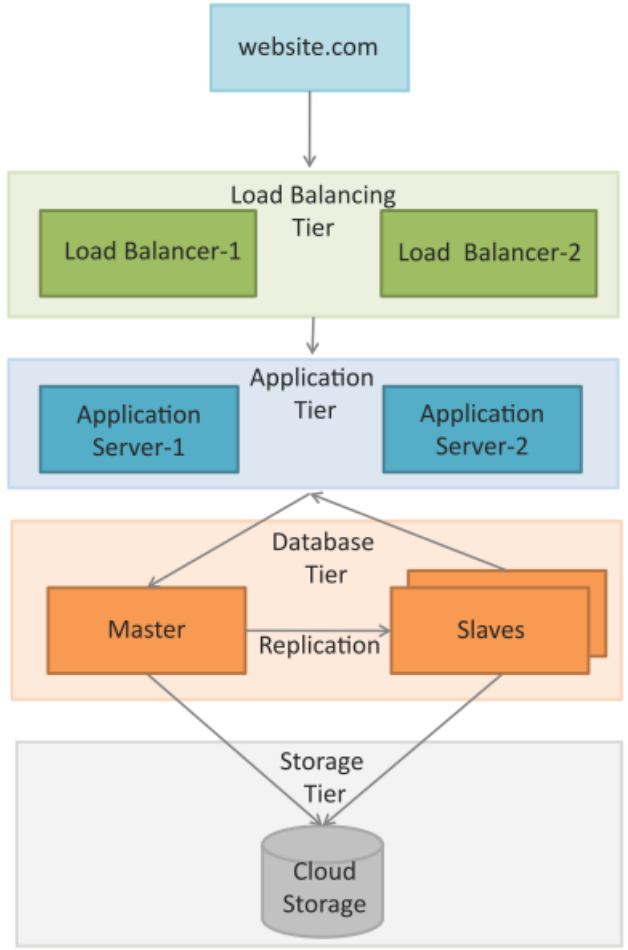
- Load balancing tier consists of one or more load balancers.

Application Tier

- For this tier, it is recommended to configure auto scaling.
- Auto scaling can be triggered when the recorded values for any of the specified metrics such as CPU usage, memory usage, etc. goes above defined thresholds.

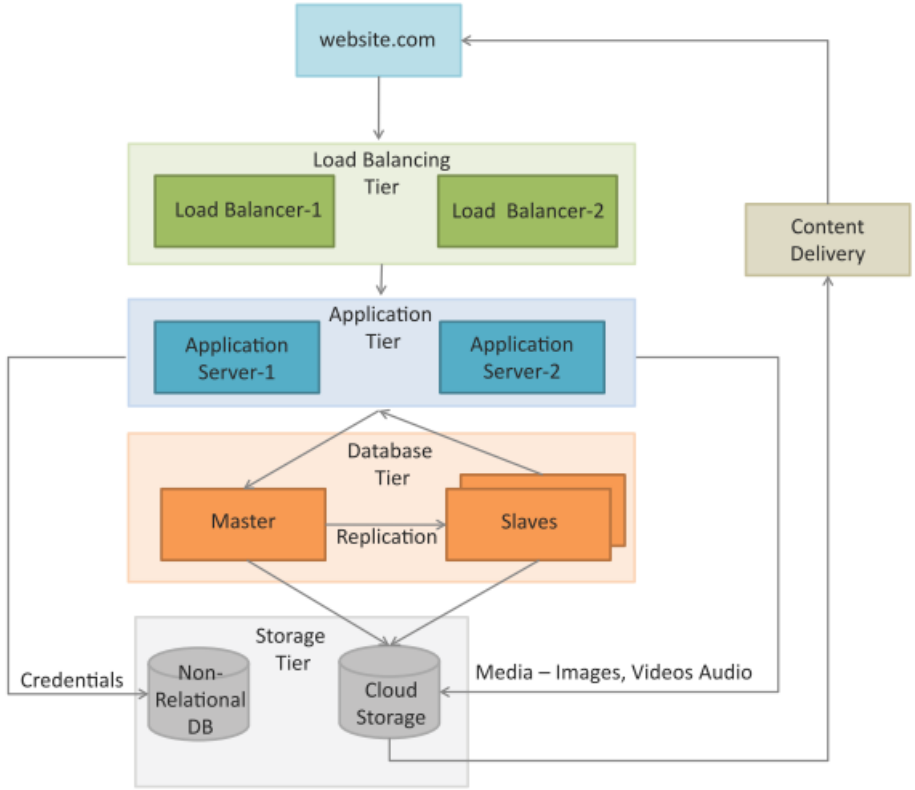
Database Tier

- The database tier includes a master database instance and multiple slave instances.
- The master node serves all the write requests and the read requests are served from the slave nodes.
- This improves the throughput for the database tier since most applications have a higher number of read requests than write requests.



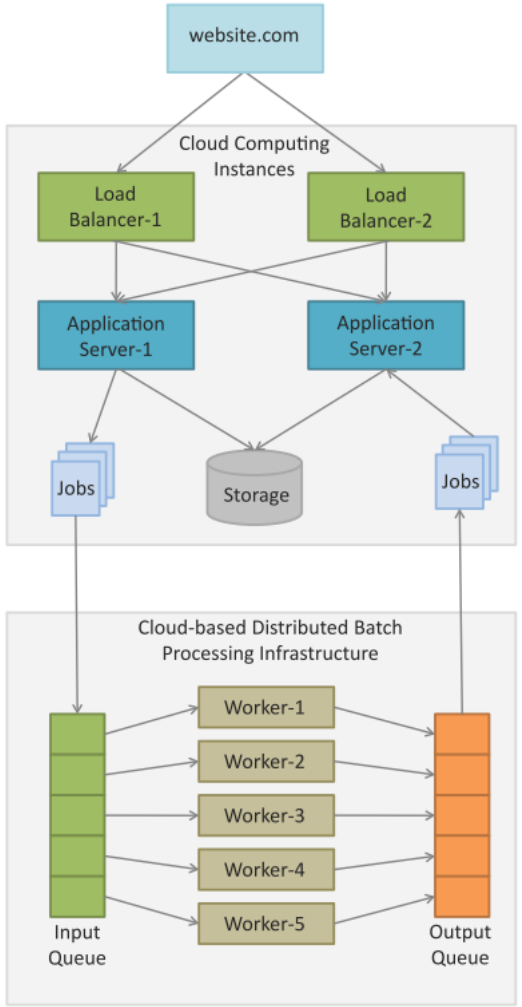
Reference Architectures –
Content delivery apps

- Figure shows a typical deployment architecture for content delivery applications such as online photo albums, video webcasting, etc.
- Both relational and non-relational data stores are shown in this deployment.
- A content delivery network (CDN) which consists of a global network of edge locations is used for media delivery.
- CDN is used to speed up the delivery of static content such as images and videos.



Reference Architectures –
Analytics apps

- Figure shows a typical deployment architecture for compute intensive applications such as Data Analytics, Media Transcoding, etc.
- Comprises of web, application, storage, computing/analytics and database tiers.
- The analytics tier consists of cloud-based distributed batch processing frameworks such as Hadoop which are suitable for analyzing big data.
- Data analysis jobs (such as MapReduce) jobs are submitted to the analytics tier from the application servers.
- The jobs are queued for execution and upon completion the analyzed data is presented from the application servers.



Service Oriented Architecture

Service Oriented Architecture (SOA) is a well-established architectural approach for designing and developing applications in the form services that can be shared and reused.

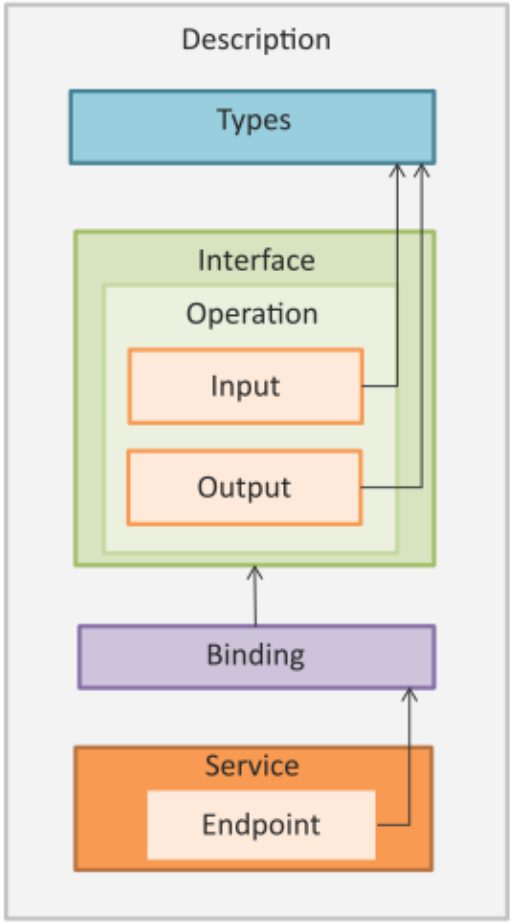
SOA is a collection of discrete software modules or services that form a part of an application and collectively provide the functionality of an application.

SOA services are developed as loosely coupled modules with no hard-wired calls embedded in the services.

The services communicate with each other by passing messages.

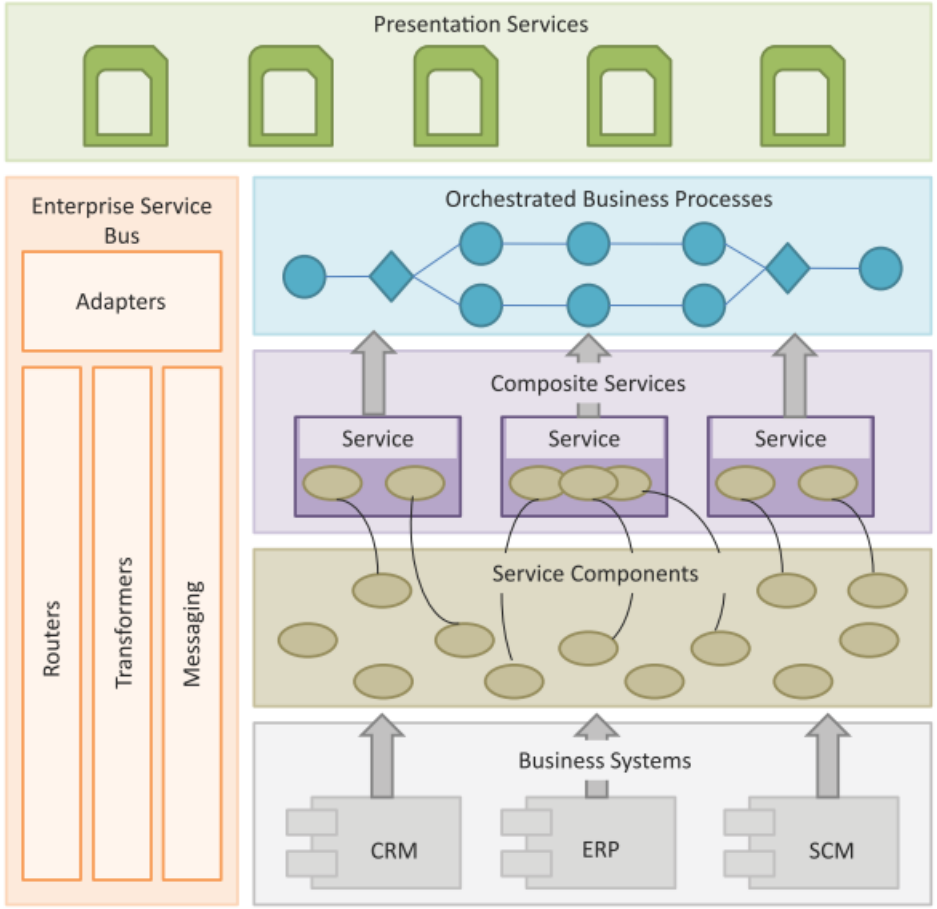
Services are described using the Web Services Description Language (WSDL).

WSDL is an XML-based web services description language that is used to create service descriptions containing information on the functions performed by a service and the inputs and outputs of the service.

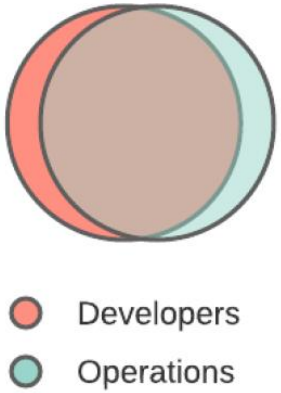


SOA Layers

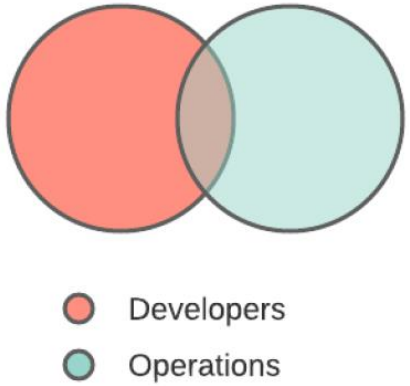
- Business Systems
  - This layer consists of custom-built applications and legacy systems such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Supply Chain Management (SCM), etc.
- Service Components
  - The service components allow the layers above to interact with the business systems. The service components are responsible for realizing the functionality of the services exposed.
- Composite Services
  - These are coarse-grained services which are composed of two or more service components. Composite services can be used to create enterprise scale components or business-unit specific components.
- Orchestrated Business Processes
  - Composite services can be orchestrated to create higher level business processes. In this layers the compositions and orchestrations of the composite services are defined to create business processes.
- Presentation Services
  - This is the topmost layer that includes user interfaces that exposes the services and the orchestrated business processes to the users.
- Enterprise Service Bus
  - This layer integrates the services through adapters, routing, transformation and messaging mechanisms.

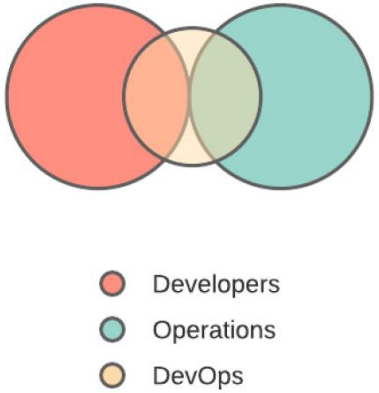


DevOps Models for Cloud Development Teams

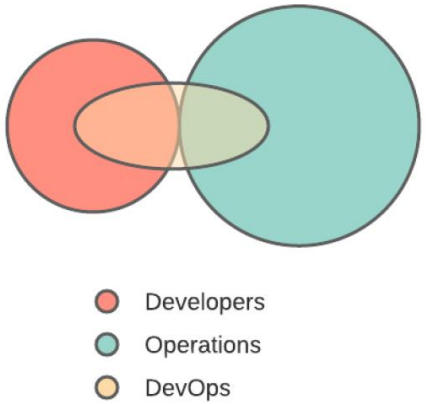


Highly technical single product teams (Facebook, Netflix,…) …

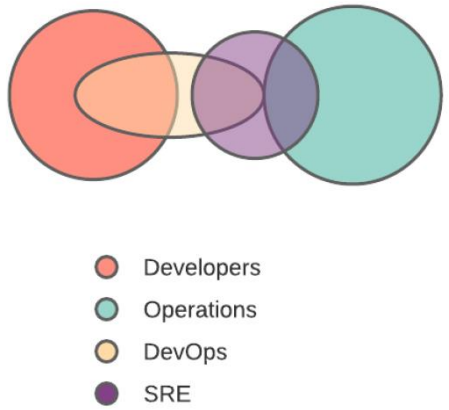


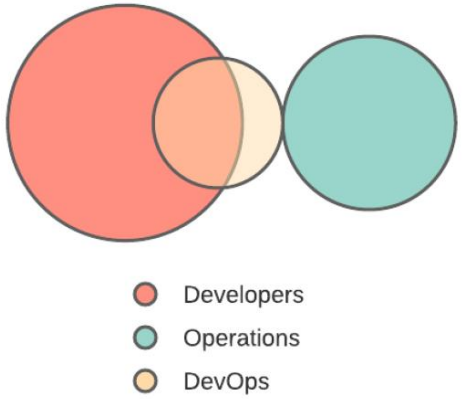


Third Party Provides DevOps Service



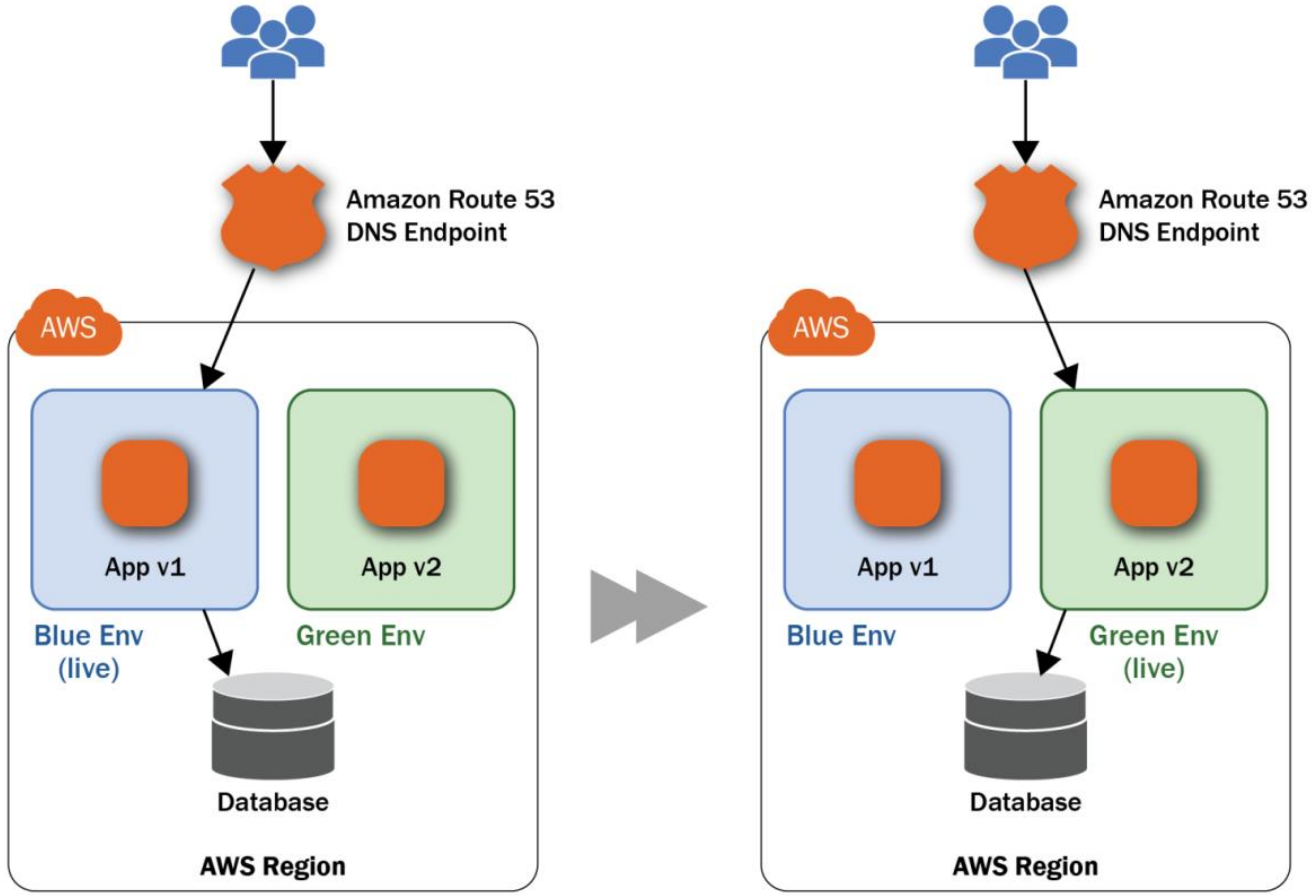
Google Model





Container model

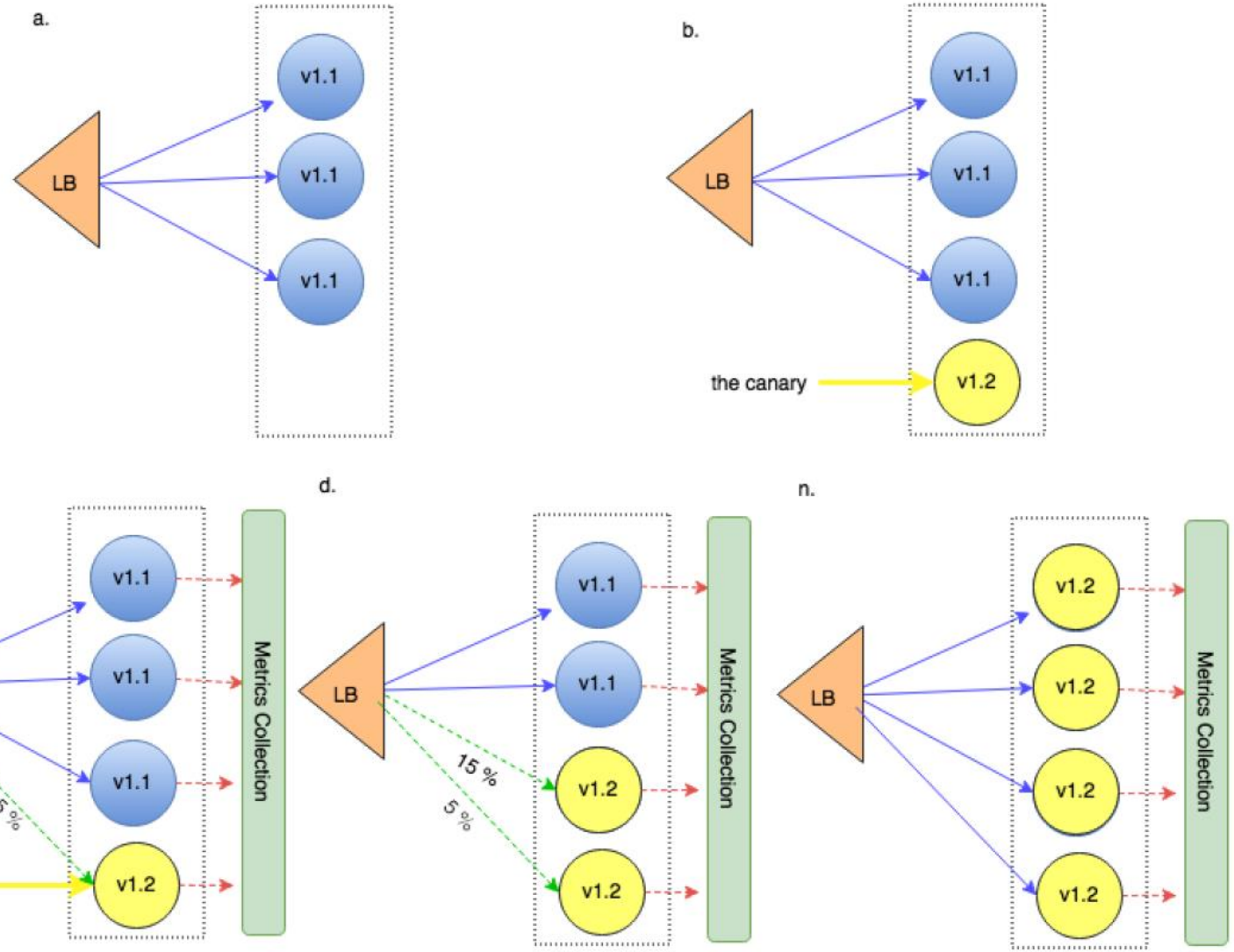
Models of Deployment of Releases in a Phased Manner



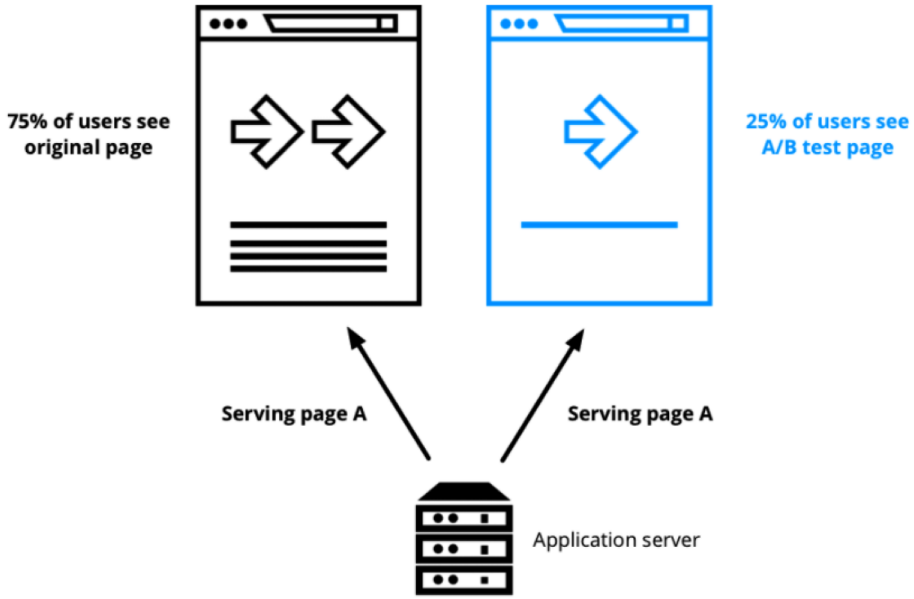
- **Blue/Green Development**

Models of Deployment of Releases in a Phased Manner

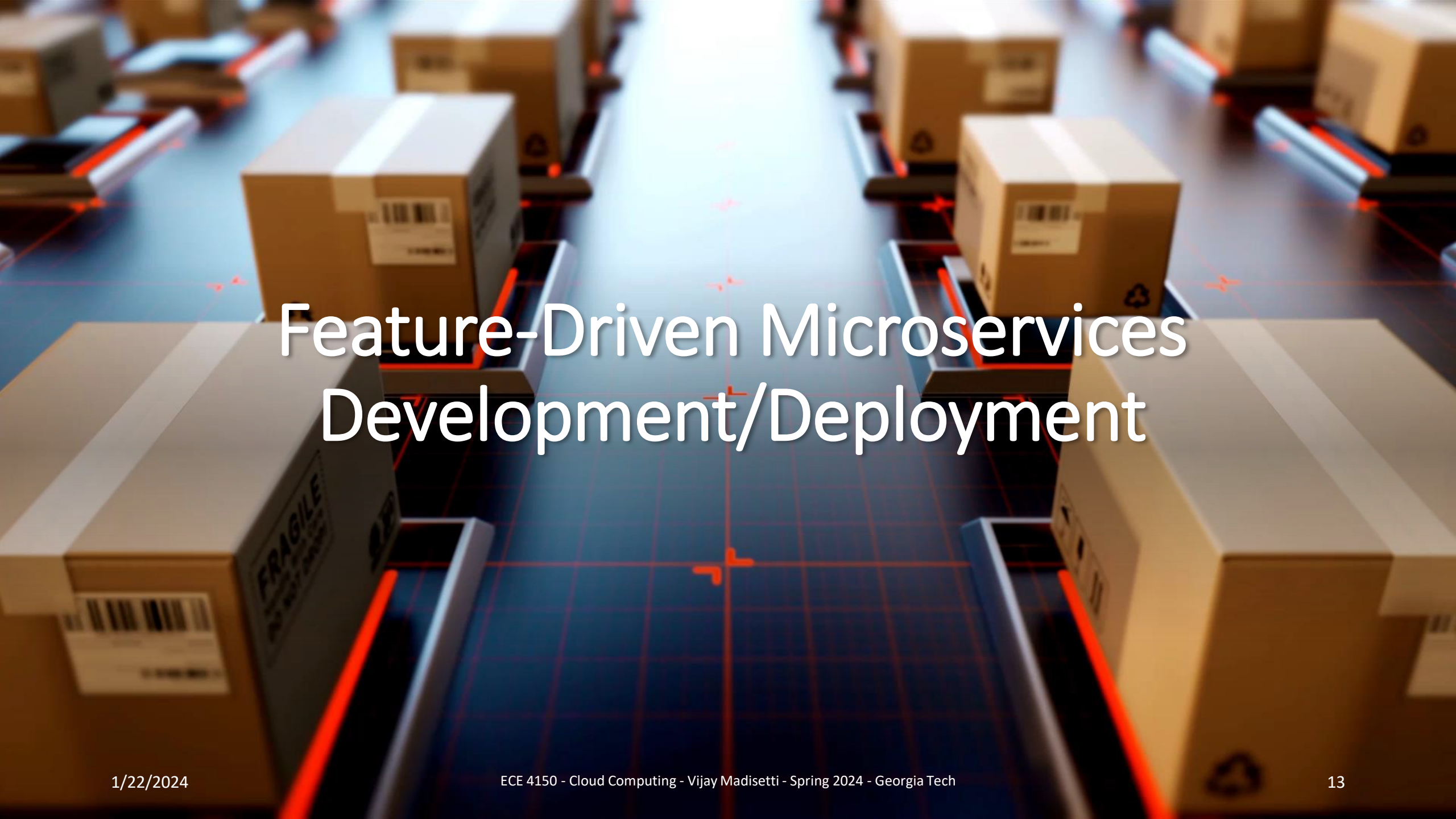
Canary Development



Models of Deployment of Releases in a Phased Manner



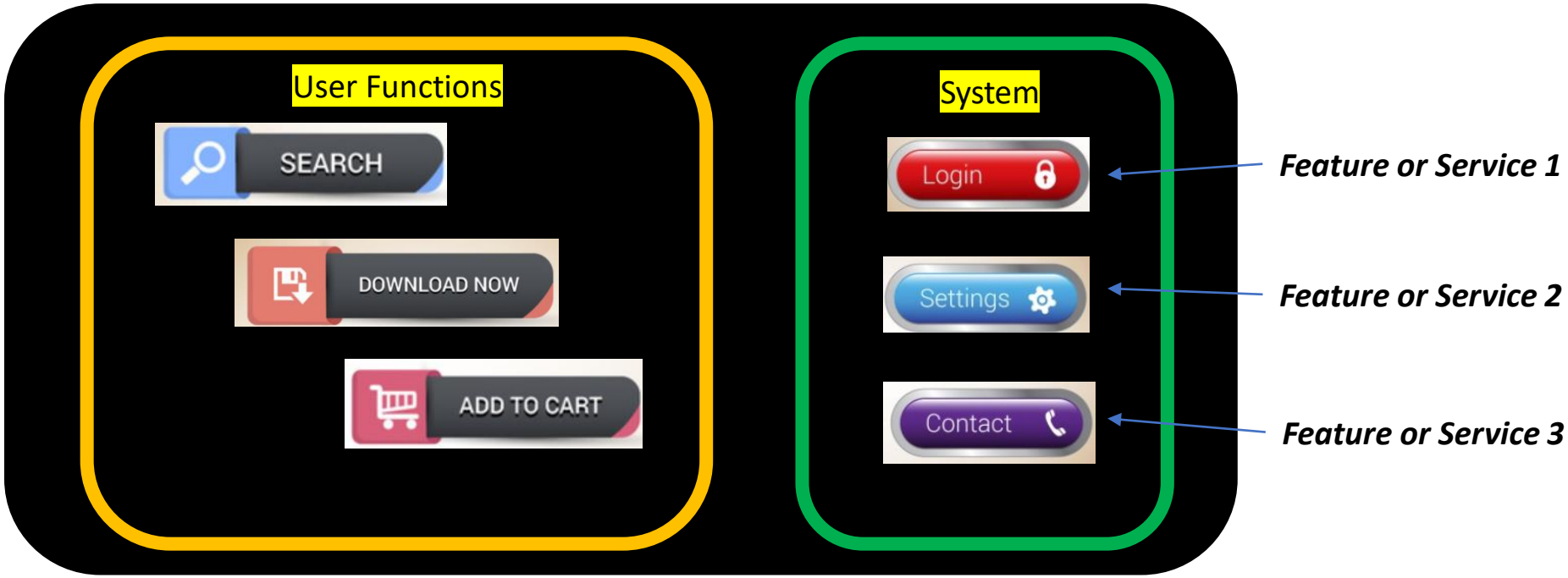
- **A/B Testing Deployment**



Feature-Driven Microservices Development/Deployment

Customer Demo: Start with *User Interface* of Cloud Application

UI



Hypothetical Photo Gallery Application

A web-based Photo Book (e.g., Lab 1)

Features include:

Add Photo

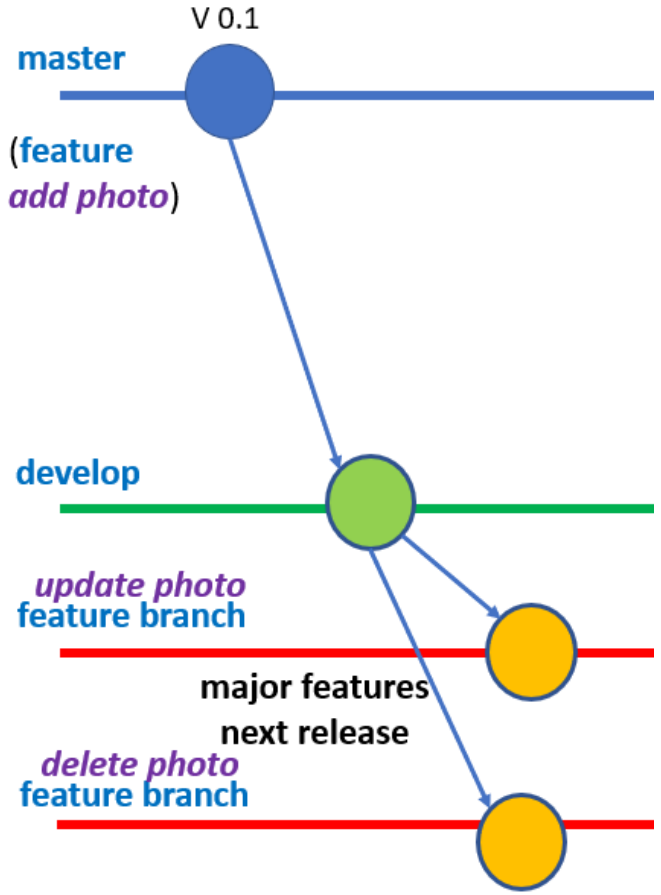
Modify Photo

Delete Photo

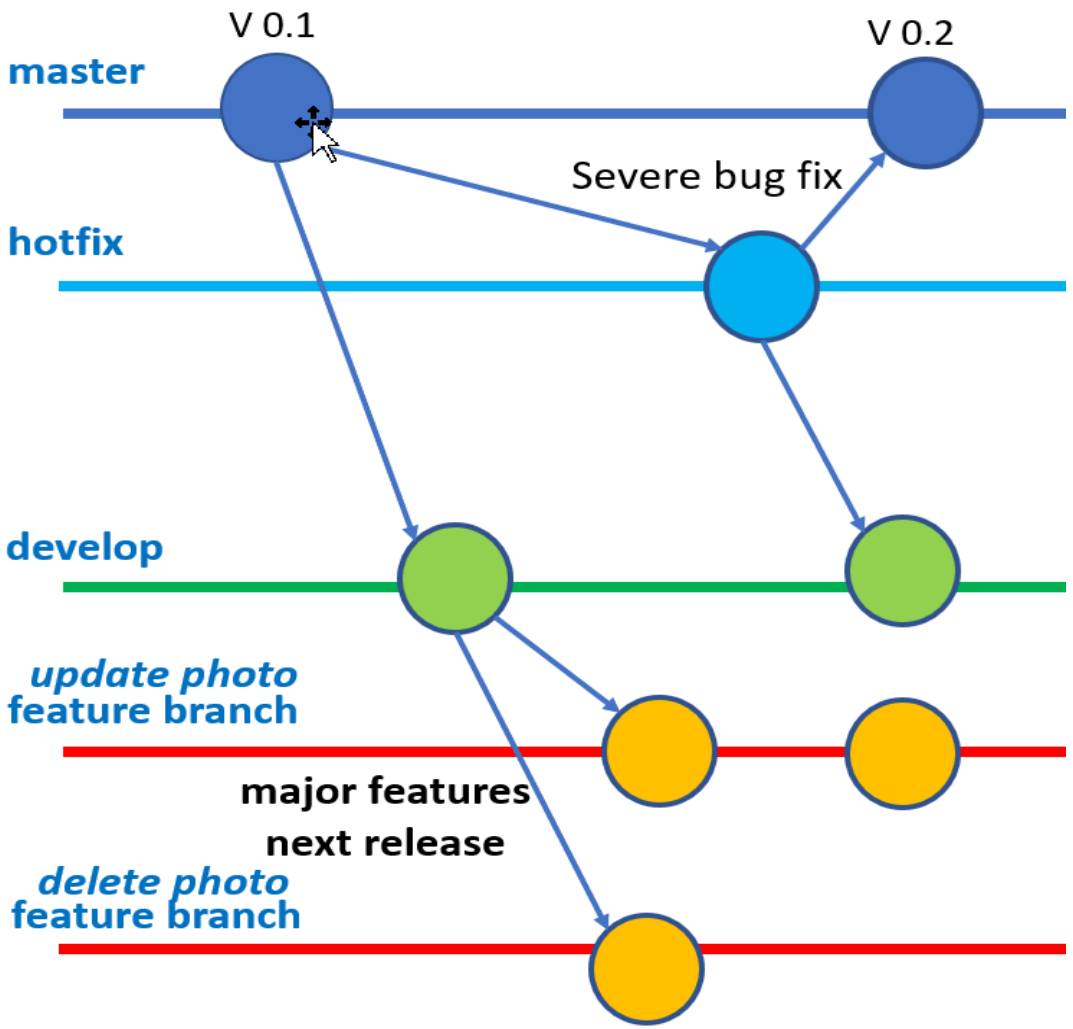
Three developers - one for each of the respective features.

Two managers for reviewing, committing, and building code.

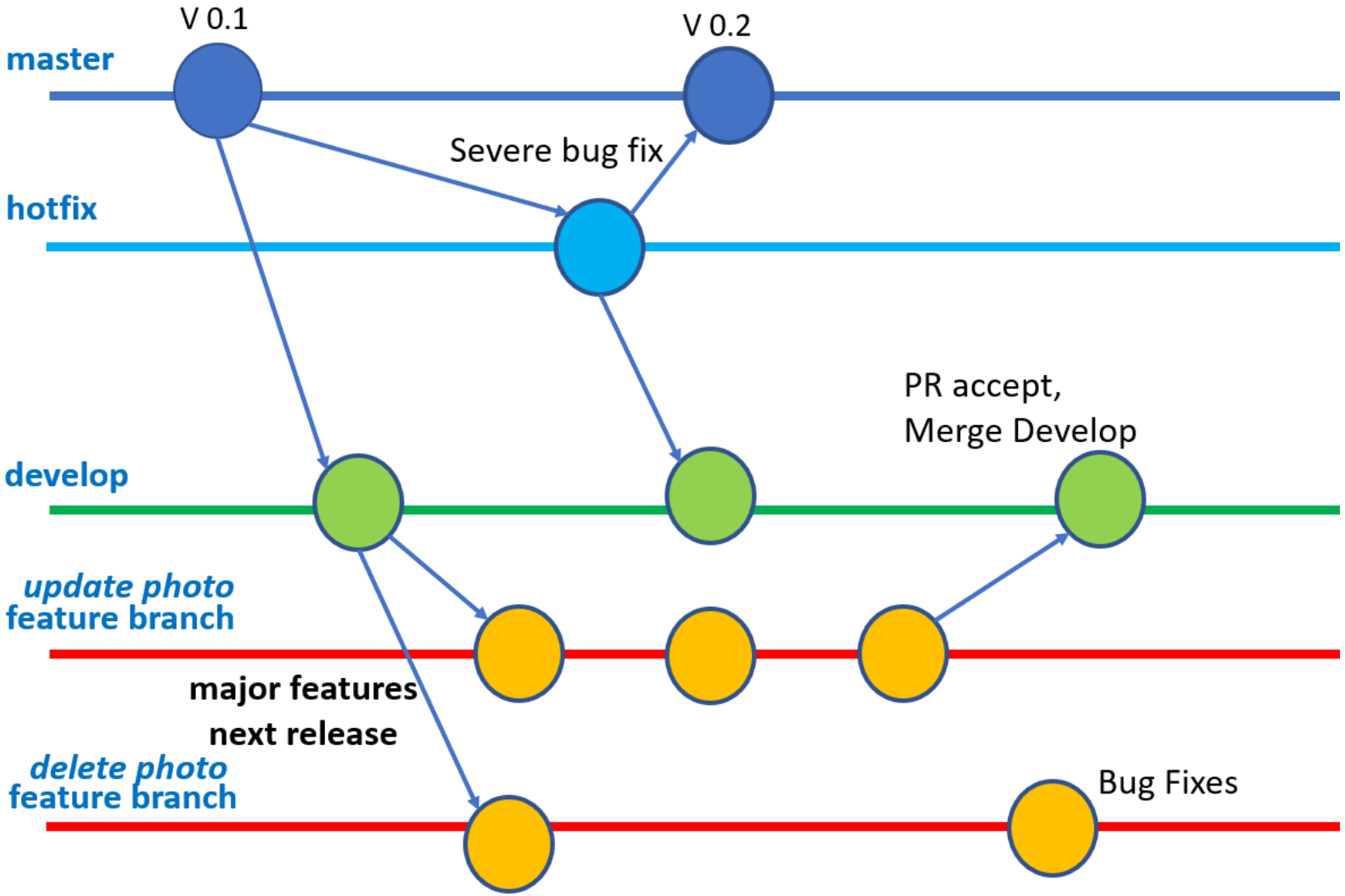
Master, Develop & Feature Branches



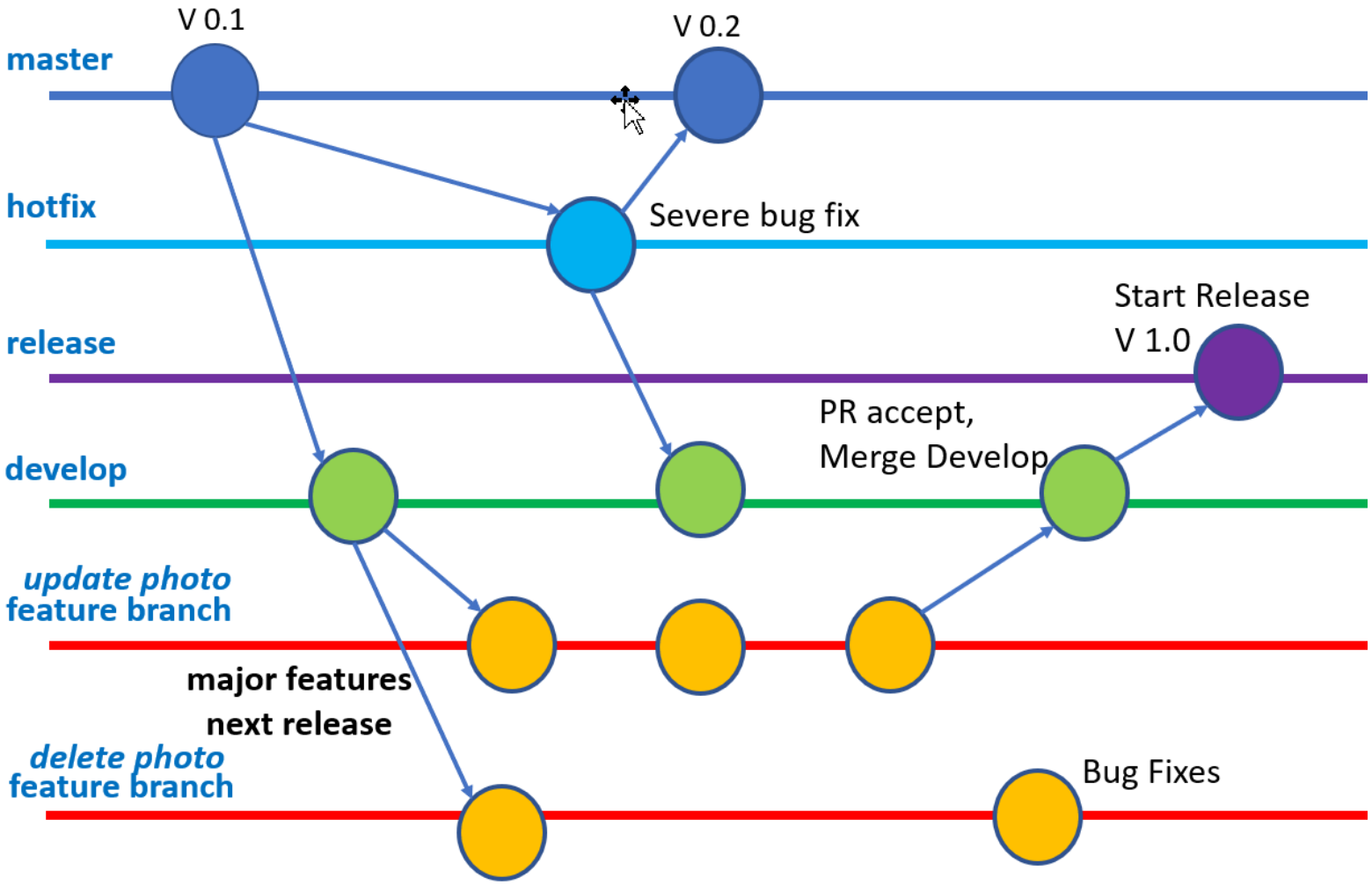
Hotfix & Develop & Feature Branches



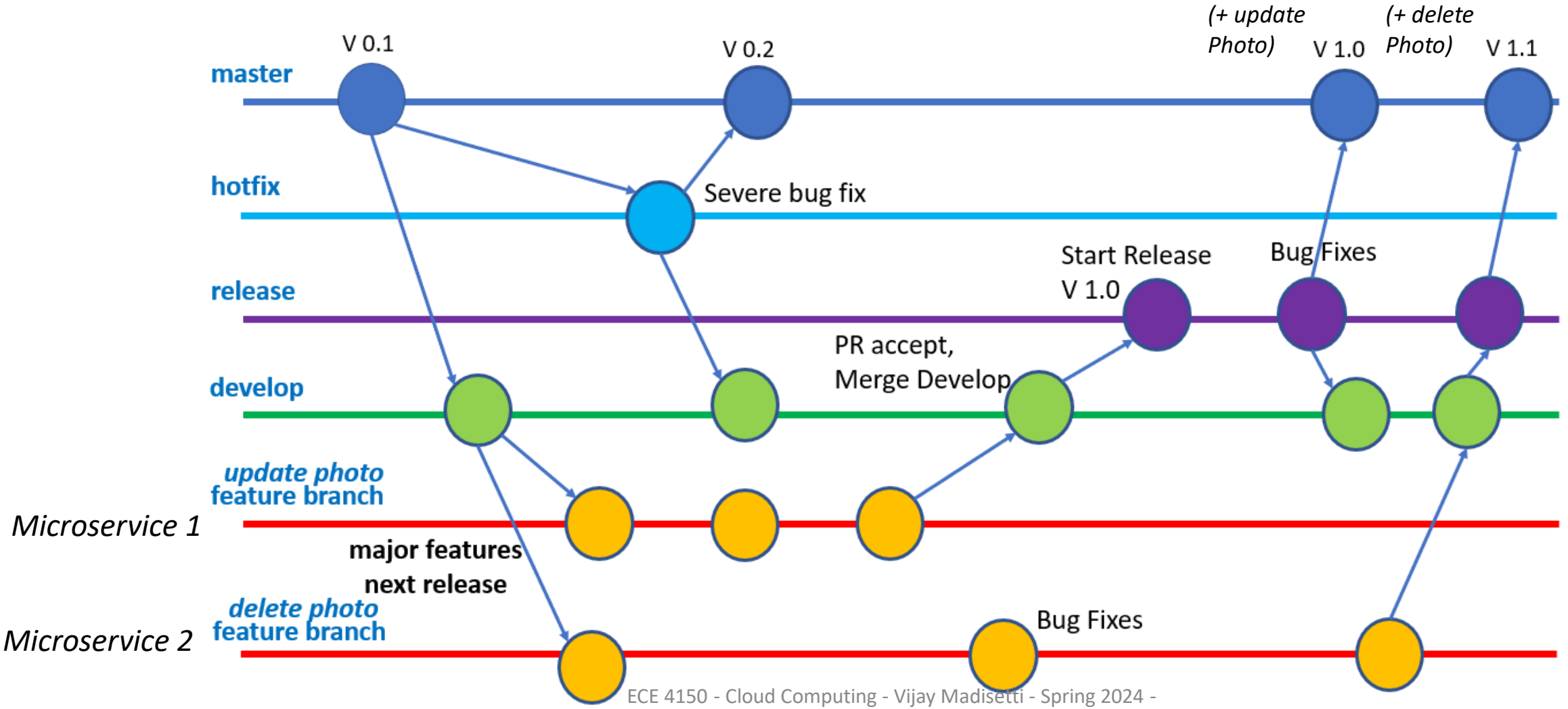
“Update Photo” Feature Merged to Develop



"Update Photo" Feature Merged to Release

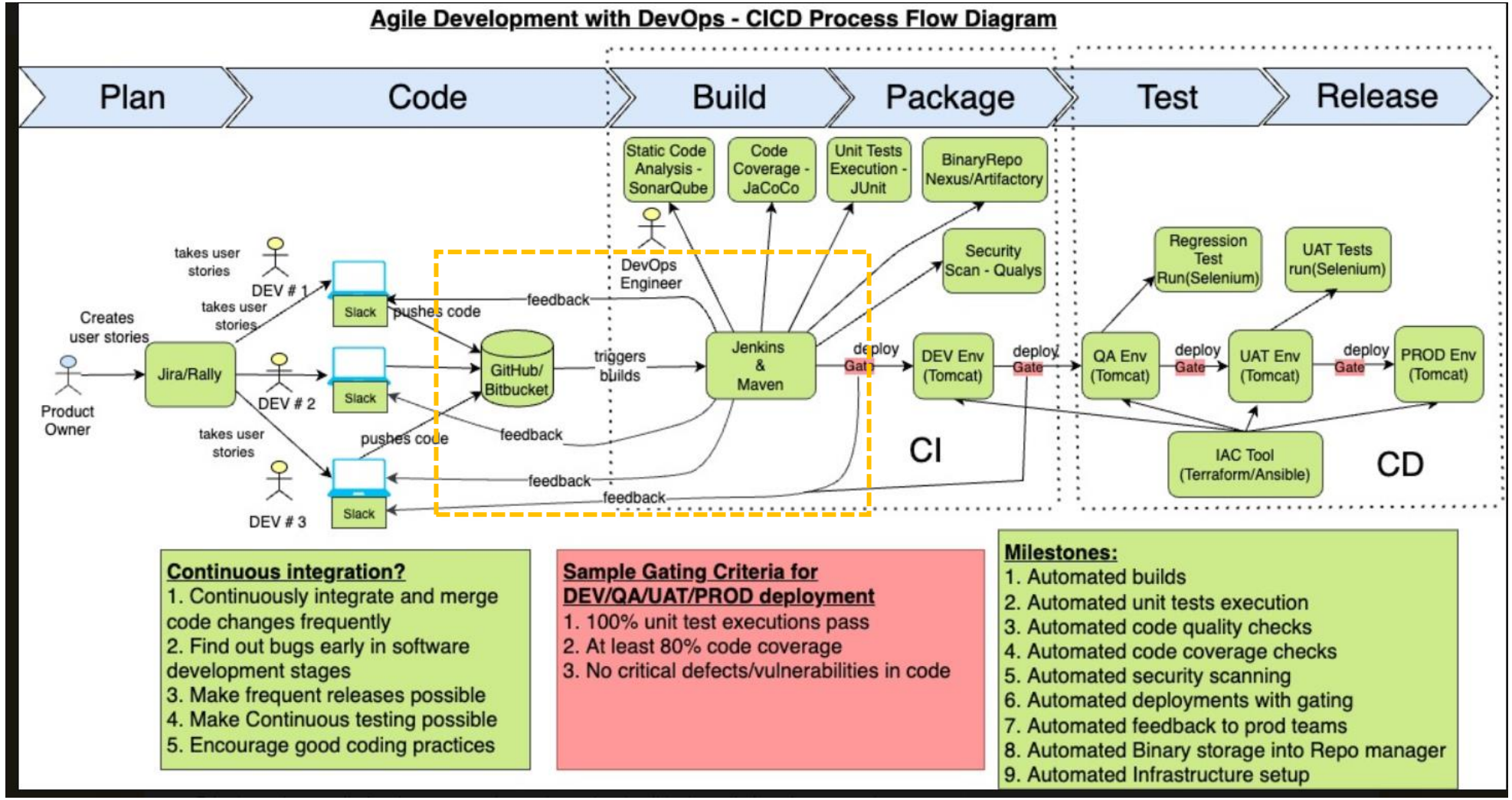


GIT - Feature "Update Photo" merged to Master (v 1.0) and Feature "Delete Photo" merged to Master (v 1.1)



Problems arrive when multiple developers are working on the same feature branch and do not merge back to develop and release branches frequently

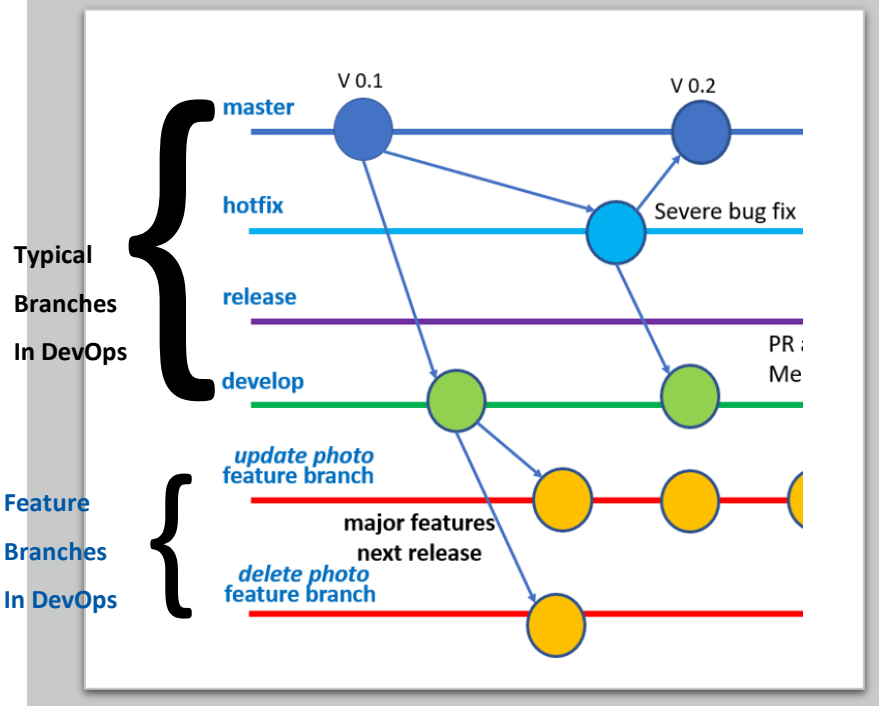
GIT + JENKINS TOOL FLOW



Guidelines for Feature Branch Developments or Microservices

Guidelines for Allowing Feature Branches

1. Permission-less innovation
2. Enable Failure
3. Disrupt Trust
4. You build it, you own it
5. End Centralized Schema and Metadata
6. Concentrate the pain
7. Test differently



Feature Branches Design – 1

- **Permissionless Innovation**
  - Each feature should be able to add independently of other features
  - No gatekeepers should be present who can regulate or control which features
  - Teams developing each branch should (ideally) not discuss their feature with other teams developing their features

Feature Branches Design – 2

Enable Failures

-- Failures should occur within the branch boundaries

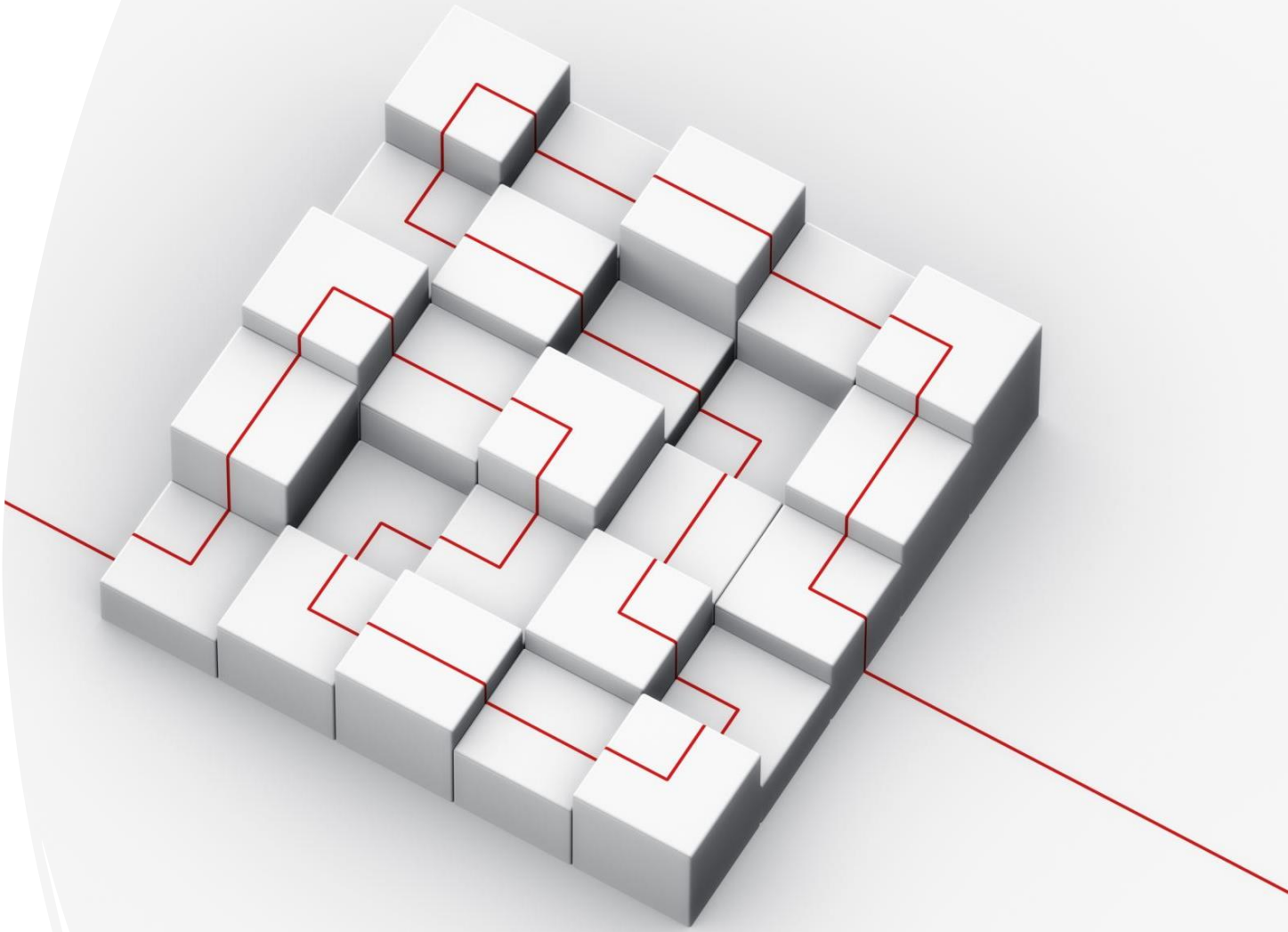
-- Failures of one branch affecting other branches will show some improper coupling of features (even in our example, *update photo* and *delete photo* are not really *decoupled*!)

-- Failures involving multiple branches add high complexity and delay to the DevSecOps processes

-- Failures are analyzed through evaluation of state persistence, dependency management, shared fate, and graceful degradation.

Feature Branches Design – 3

- **Disrupt Trust**
  - Boundaries between each Feature is the API and all features give rise to a set of APIs.
  - What is behind the APIs is hidden but is governed by Service Level Agreements
  - Trust is replaced by autonomy and accountability



Feature Branches – 4

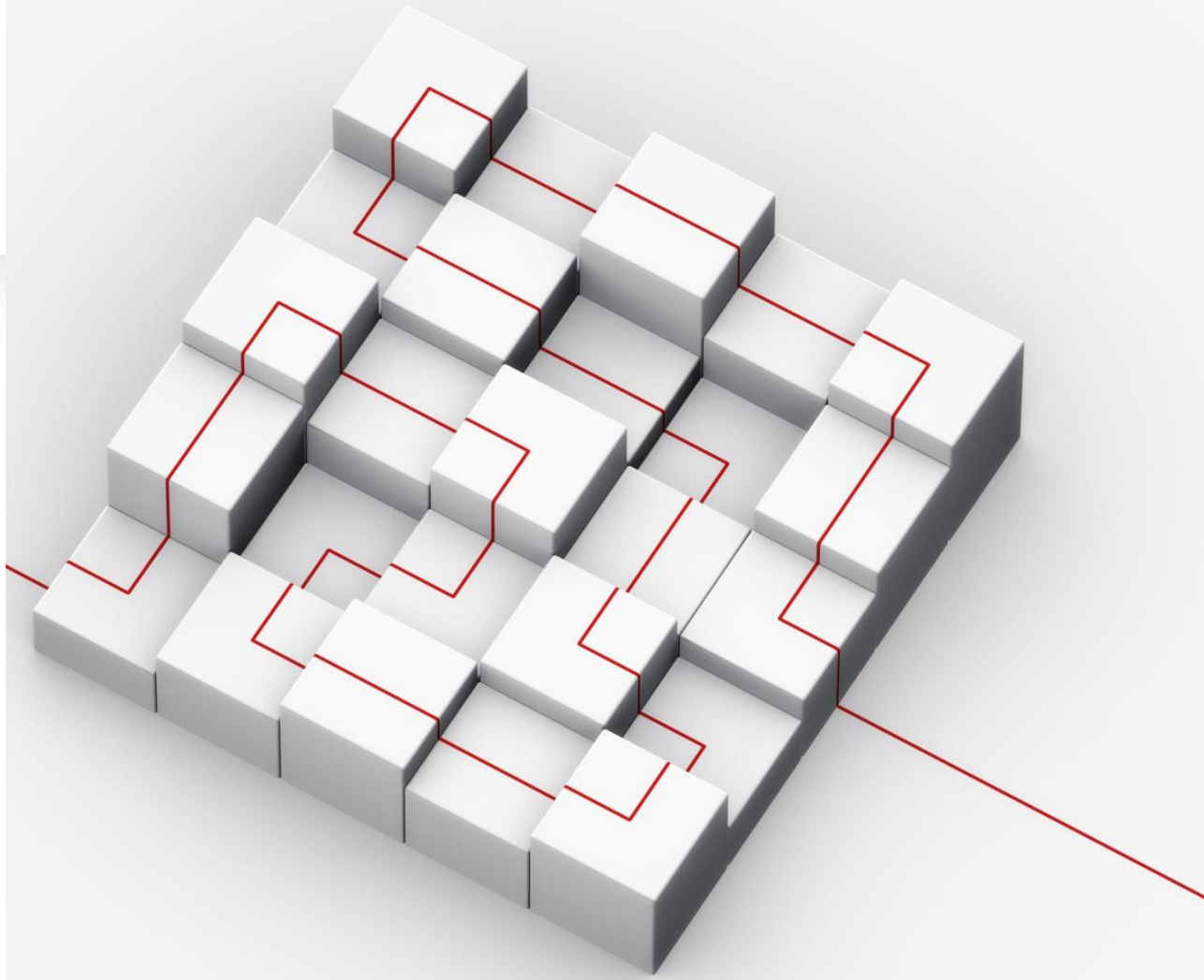
- **You build it, you own it.**
  - Each service has a team, and the team is responsible for it, scoping its functionality, architecting it, building it, and operating it. Amazon ex-CTP Werner Vogels, 2006.
  - DevSecOps has adopted this model

Feature Branches Design - 5

- **<u>Accelerate Deprecation</u>**
  - Good services, that are used frequently, stay, and unused services go away.
  - Services allow competition for resources and efficiency
  - This sounds brutal – it is. Unused services fade away

Design Feature Branches - 6

- **End Centralized Metadata and Schema**
  - No more fixed schema from relational databases controlled by centralized gatekeepers
  - Each branch develops their own schema and data formats
  - APIs had these details



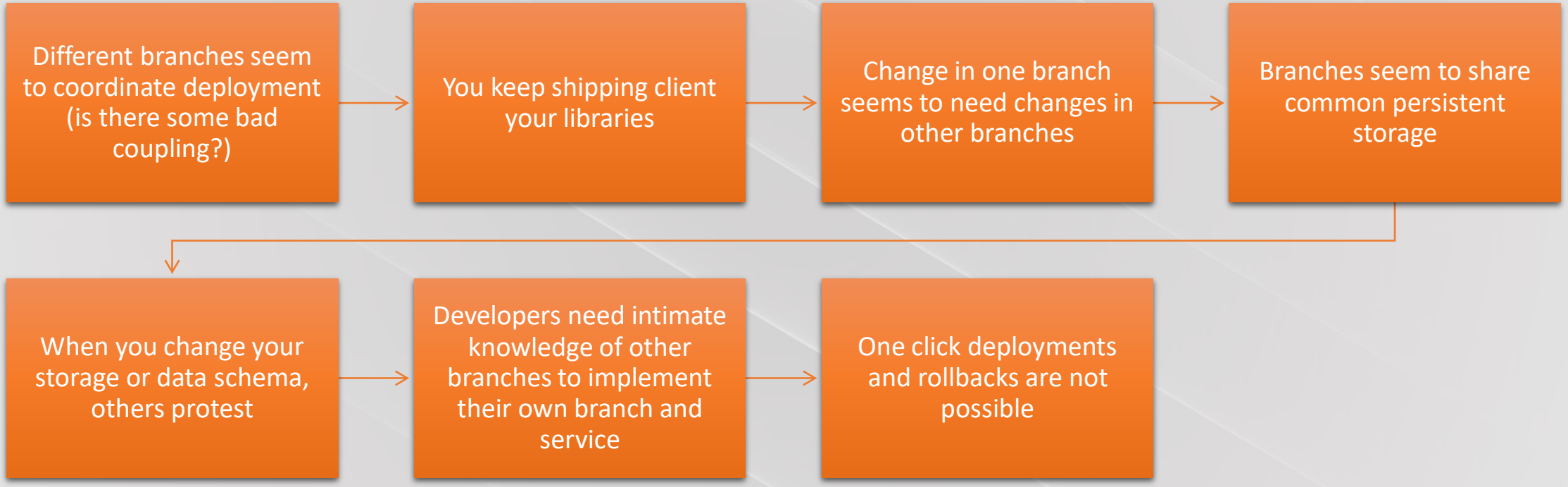
Feature Branches Design – 7

- **Concentrate the Pain**
- Threat modeling identifies the critical data and processes
- Company's security procedures and compliance requirements drive identification of critical data
- Concentrate the critical data within a small set of services, allowing remaining services to innovate freely and unburdened by these constraints.

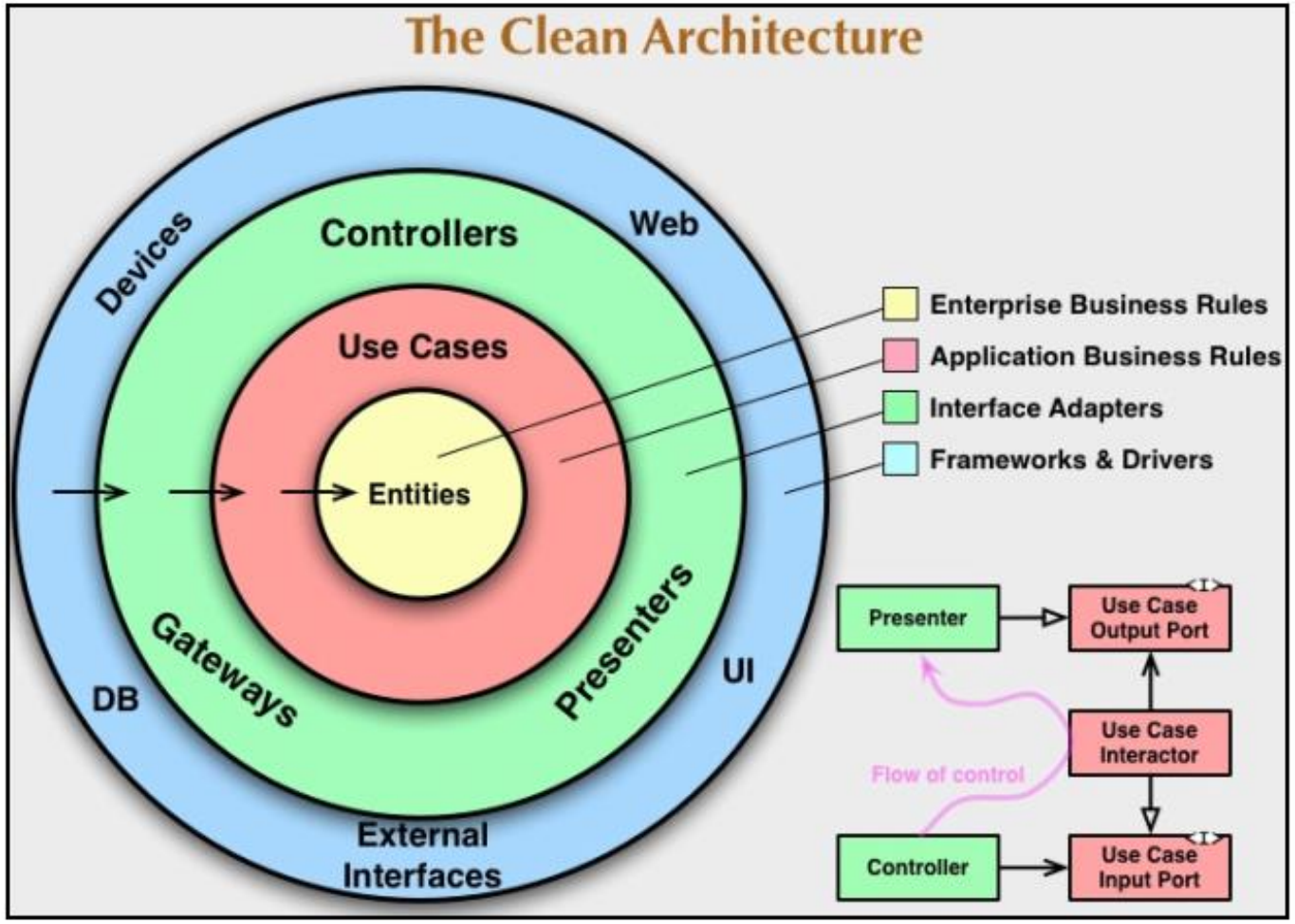
Feature Branch Design – 8

- **Test Differently**
- Test APIs earlier, even before building the service
- Ownership of service gives rise to greater coverage of testing
- Your interface needs to be tested, since that is what your customer sees.

Warning signs of bad *Feature Branch or Microservices* designs

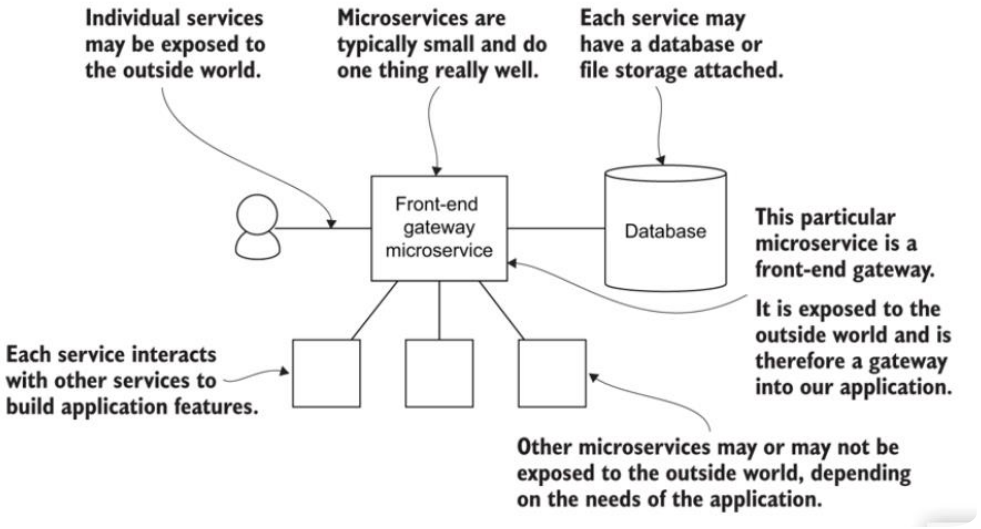


Software Product Architecture



An inner layer does not know or depend on an outer layer (relative to it)

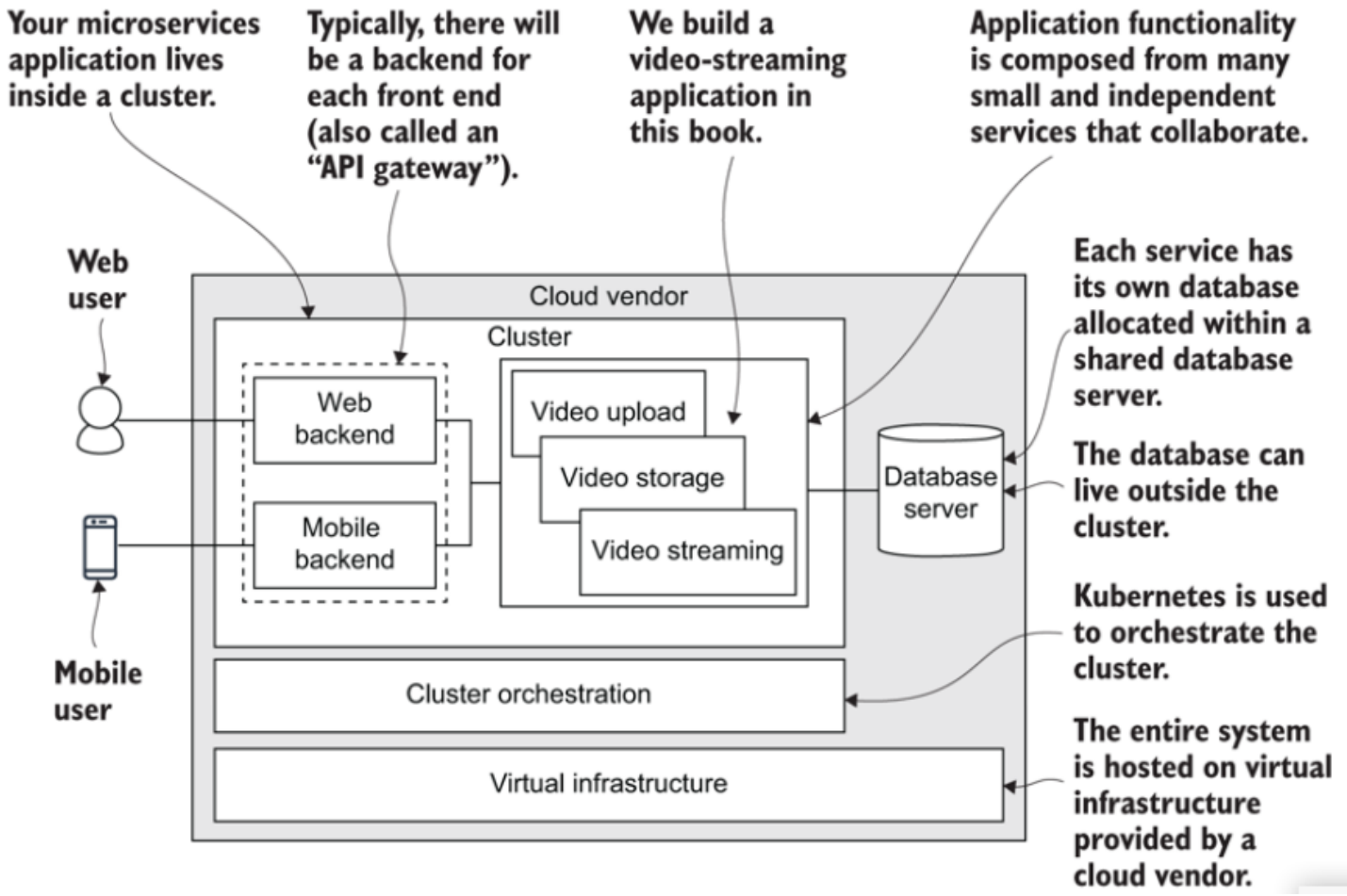
What is a Microservice



- *Microservice is a small and independent software process that runs on its own deployment schedule and can be updated independently.*
- *A microservice may have an external or internal facing API.*

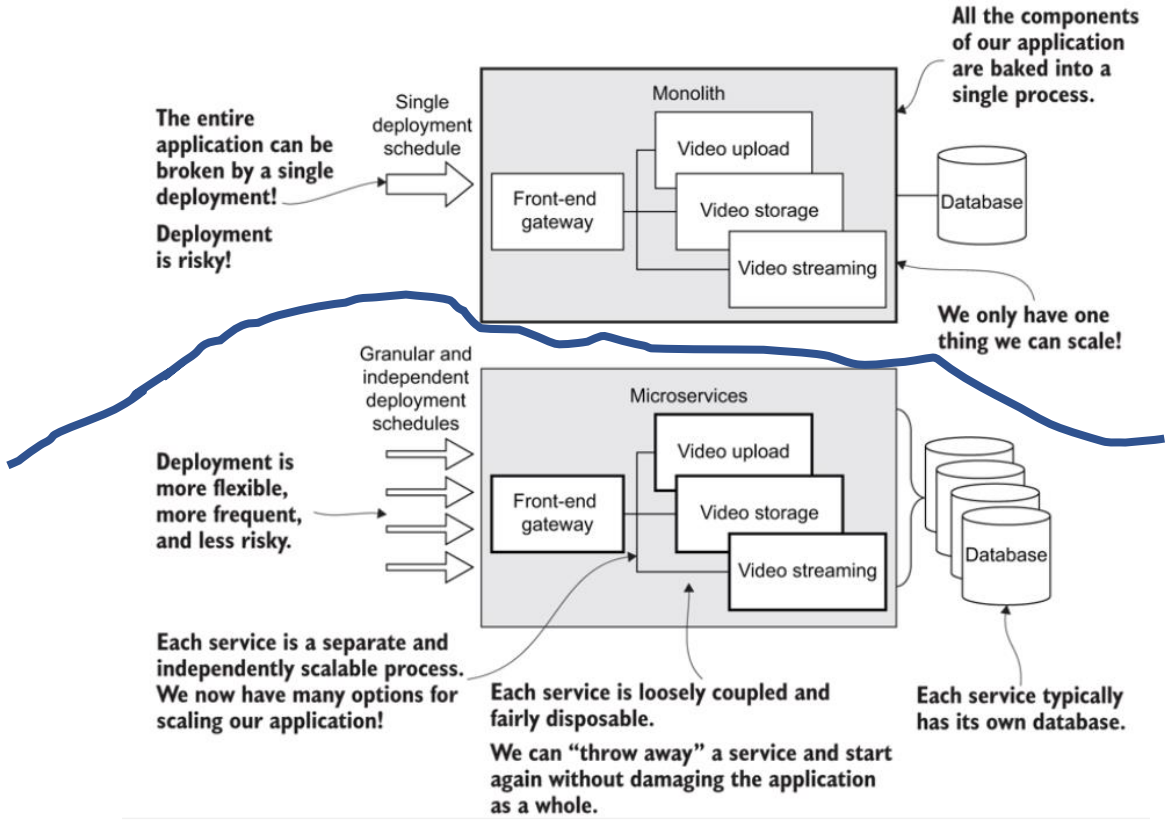
A *Microservices* Application

- A microservices application is a distributed program composed of many microservices to achieve the features and functionality of the overall application



Monolithic vs Microservices

- At a high-level they look similar.
- But the differences appear when you look closer.



Monolithic Architecture Netflix (prior 2010)

Load Balancer

Monolithic App

Account Component

Catalog Component

Recommendation Component

Customer Service Component

Database

Microservices Architecture (2010+) on AWS

Load Balancer

API Gateway

Account Service

Catalog Service

Catalog Data Service

Catalog DB

Recommendation Service

Customer Service Service

Customer DB

Summary

- You learned about version control and feature design, and how it ties to efficient microservice-based cloud applications
- You learned about models for deployment
- You learned about efficient branch design for DevOps
- We will continue to look at microservices-based design before moving to another model of PaaS in preparation for Lab 2