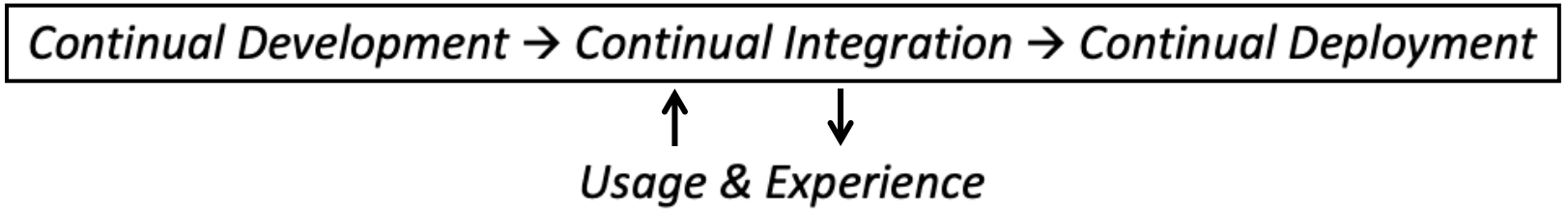
# Lifecycle Management

Prof. Ítalo Cunha

# Lifecycle Management

- Evolving a system over time
  - Classical software
    - Development, integration, and testing separate from deployment

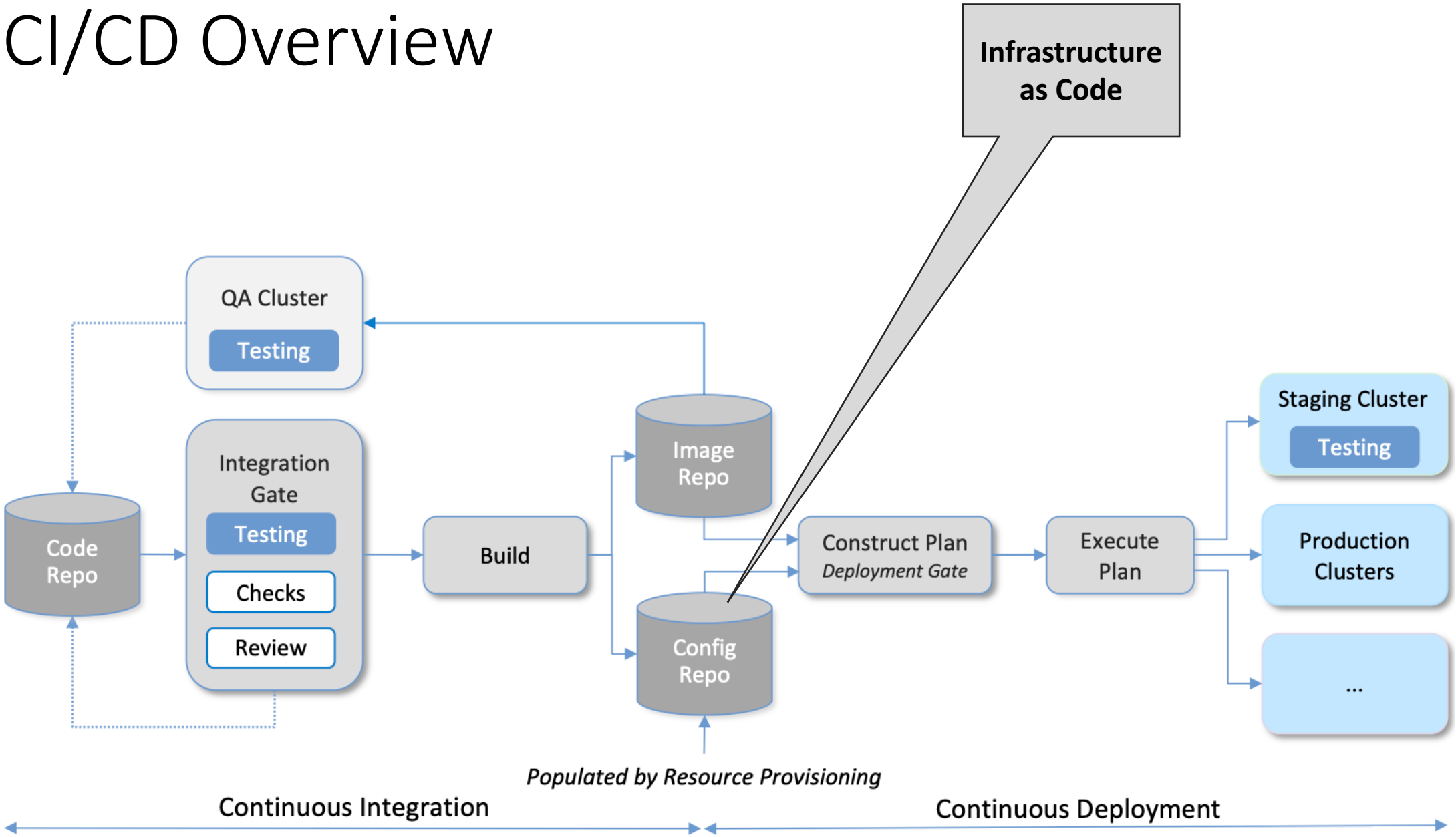



  - Cloud-native approach
    - Development, integration and testing integrated with deployment
    - Greater *feature velocity*

# From development to deployment

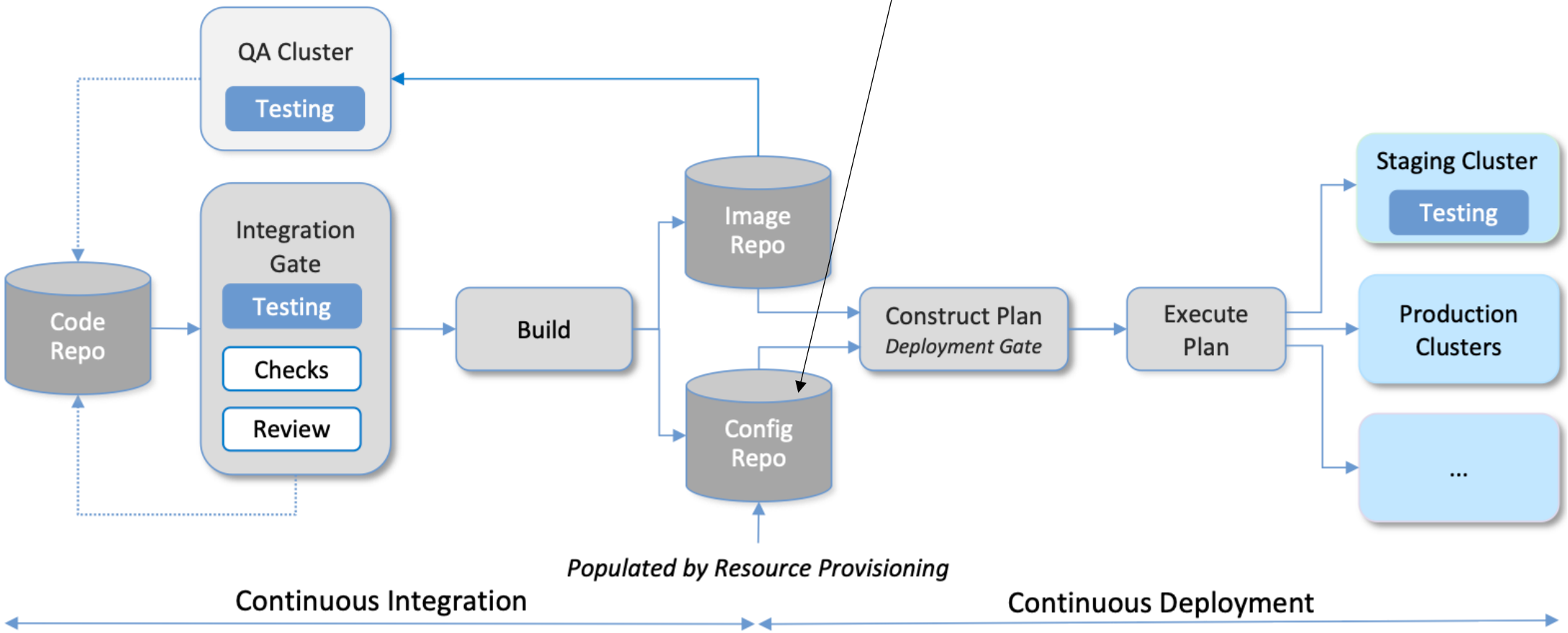
- Owning the whole pipeline requires significant effort
  - Configuration
  - Development
  - Testing & Integration
  - Deployment
  - Monitoring & Telemetry
- Not every company has the same resources as a cloud provider
  - But can still leverage open source

# CI/CD Overview



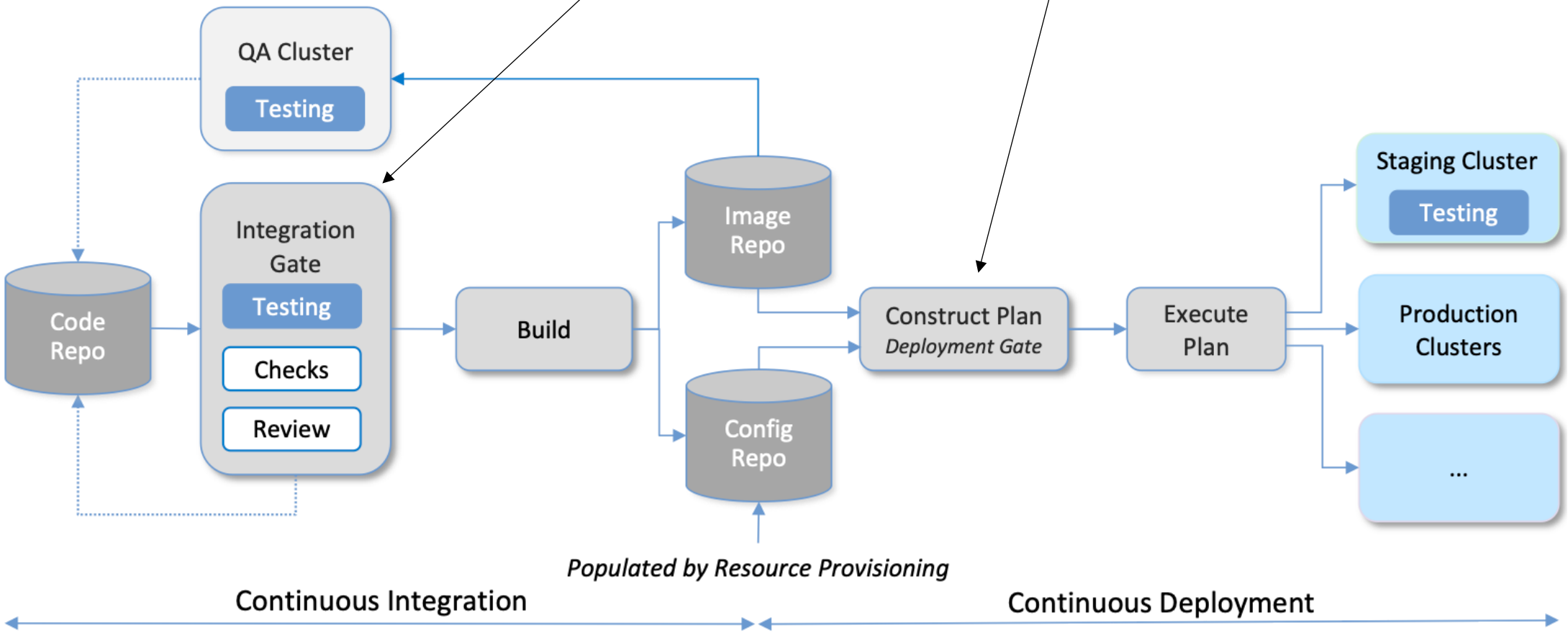
# CI/CD Overview

- Well defined artifacts between resource provisioning, CI, and CD, which operate independently
- All artifacts contained in the pipeline



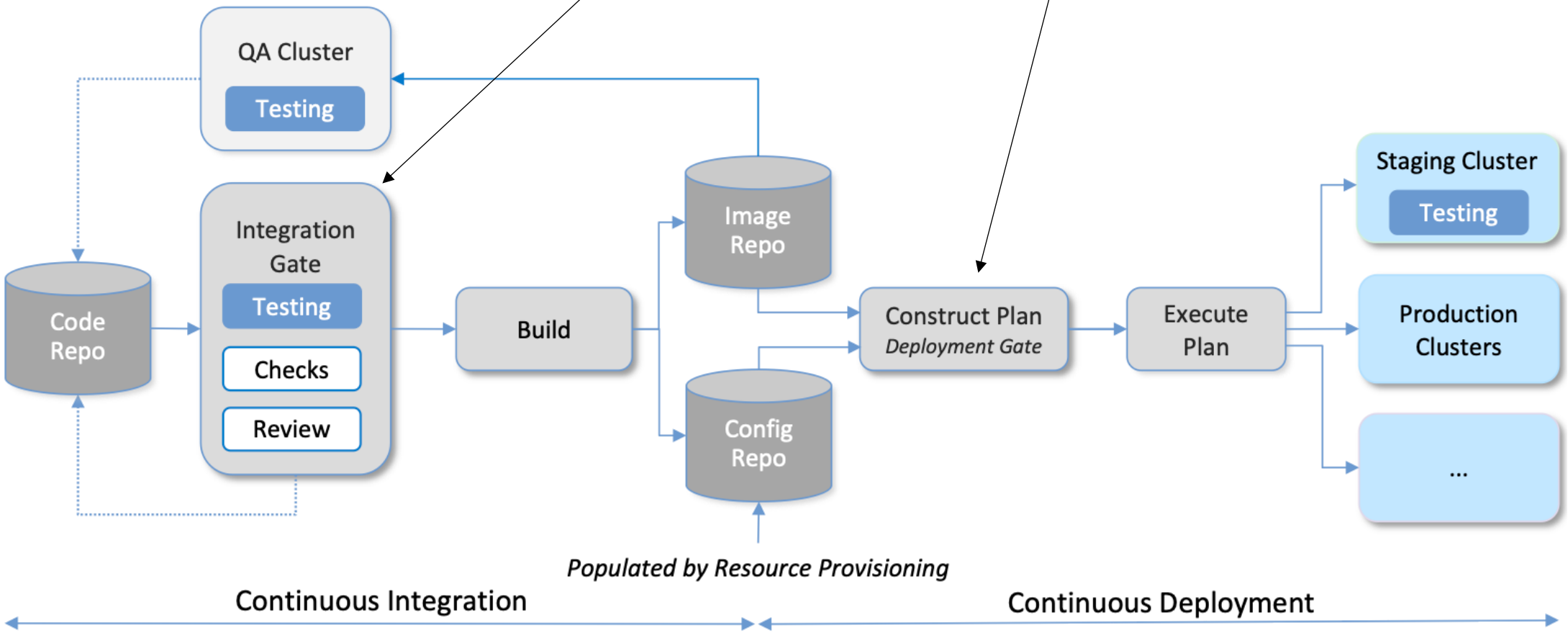
# CI/CD Overview

- Discretion at integration and deployment gates
- Continuous ≠ immediate
- Automation is faster, but can support scheduling updates



# CI/CD Overview

- Discretion at integration and deployment gates
- Continuous ≠ immediate
- Automation is faster, but can support scheduling updates



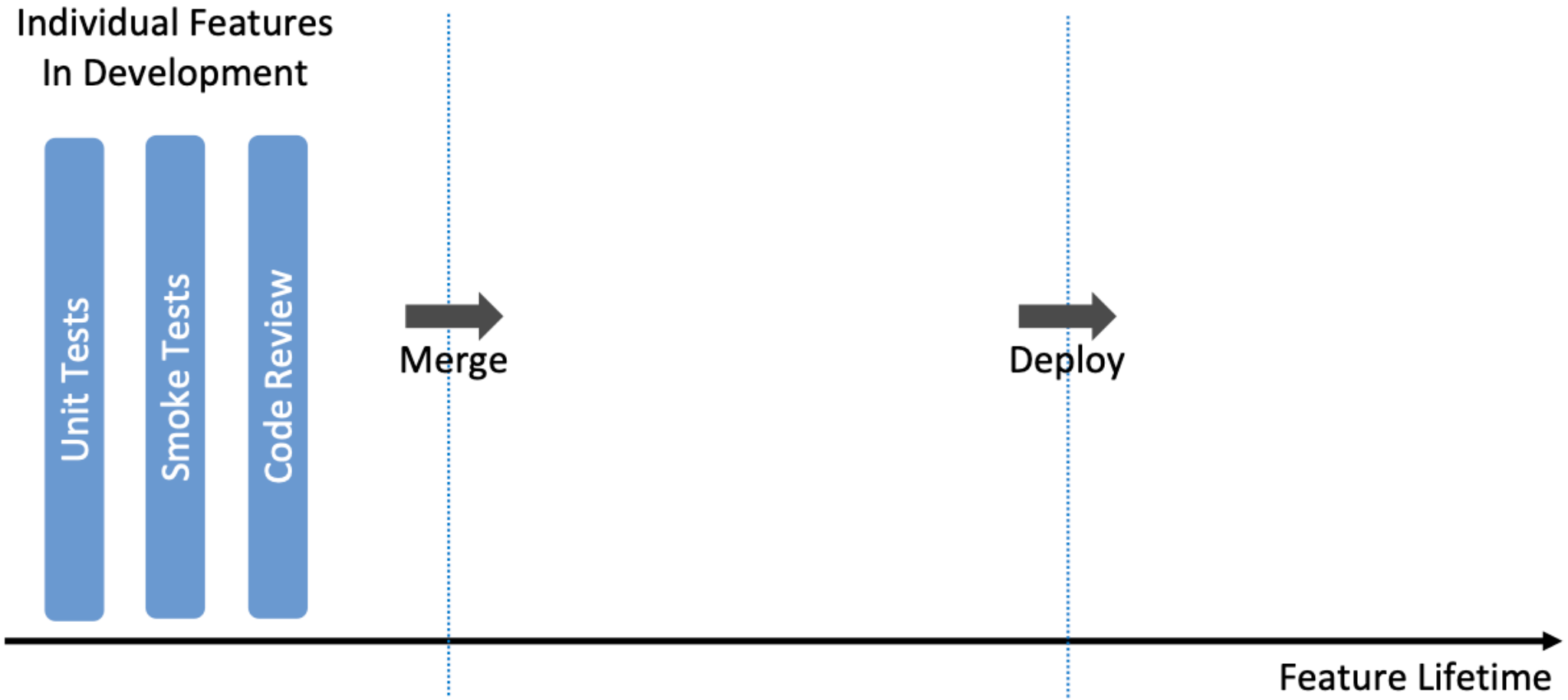
# Testing Strategies

Prof. Ítalo Cunha

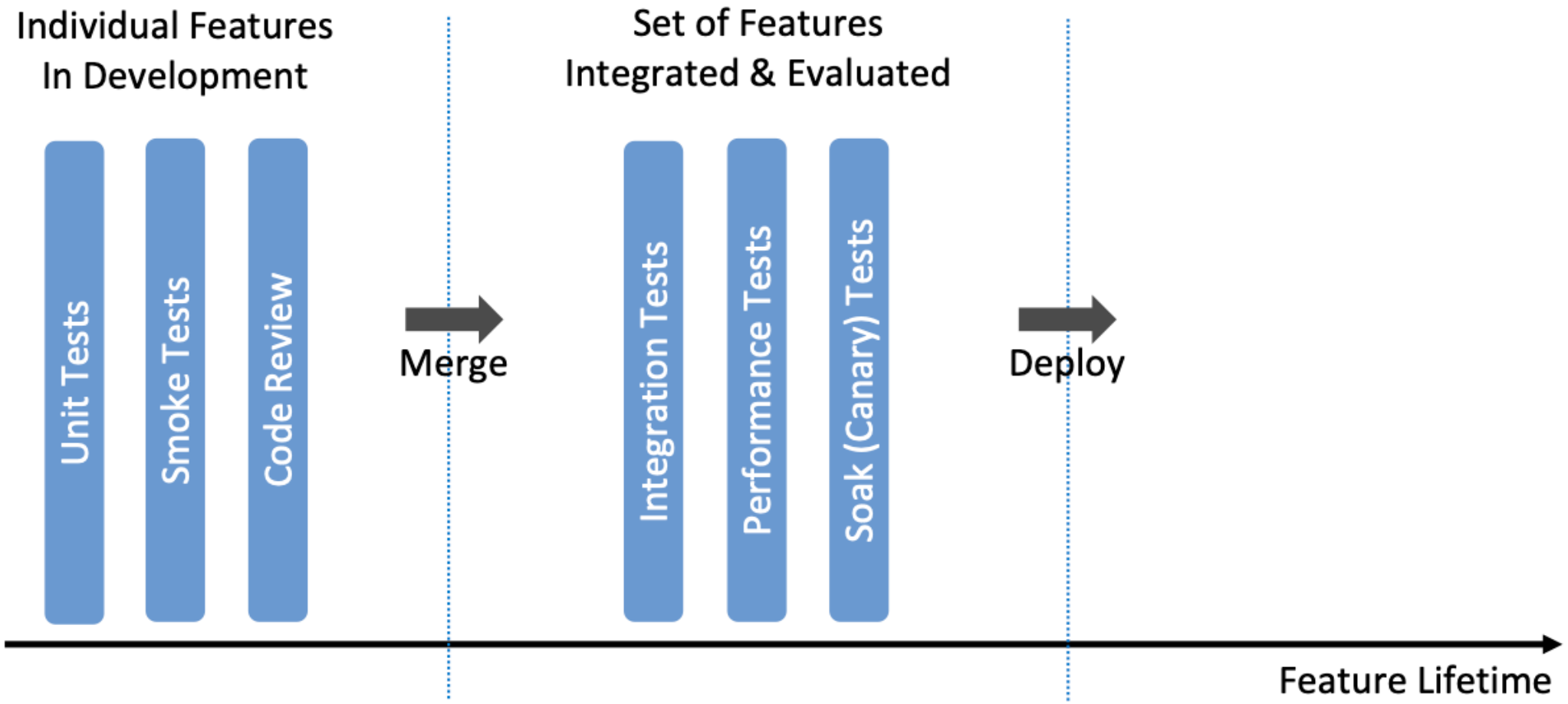
# Testing Strategies

- Balance between feature velocity and quality
  - Reliable, scalable, performant code
- Need lots of test → need automation
- Bring tests as early as possible in the pipeline

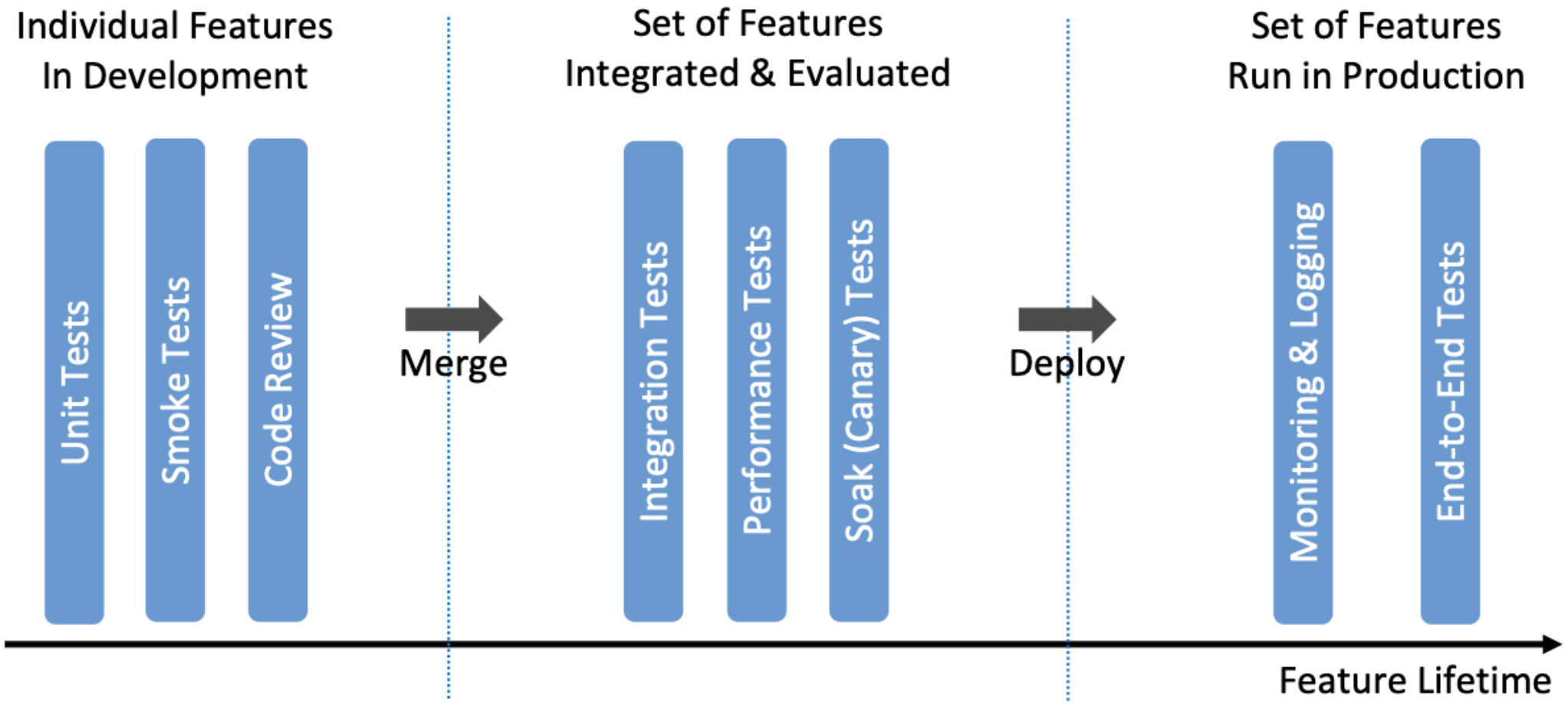
# Testing Strategies



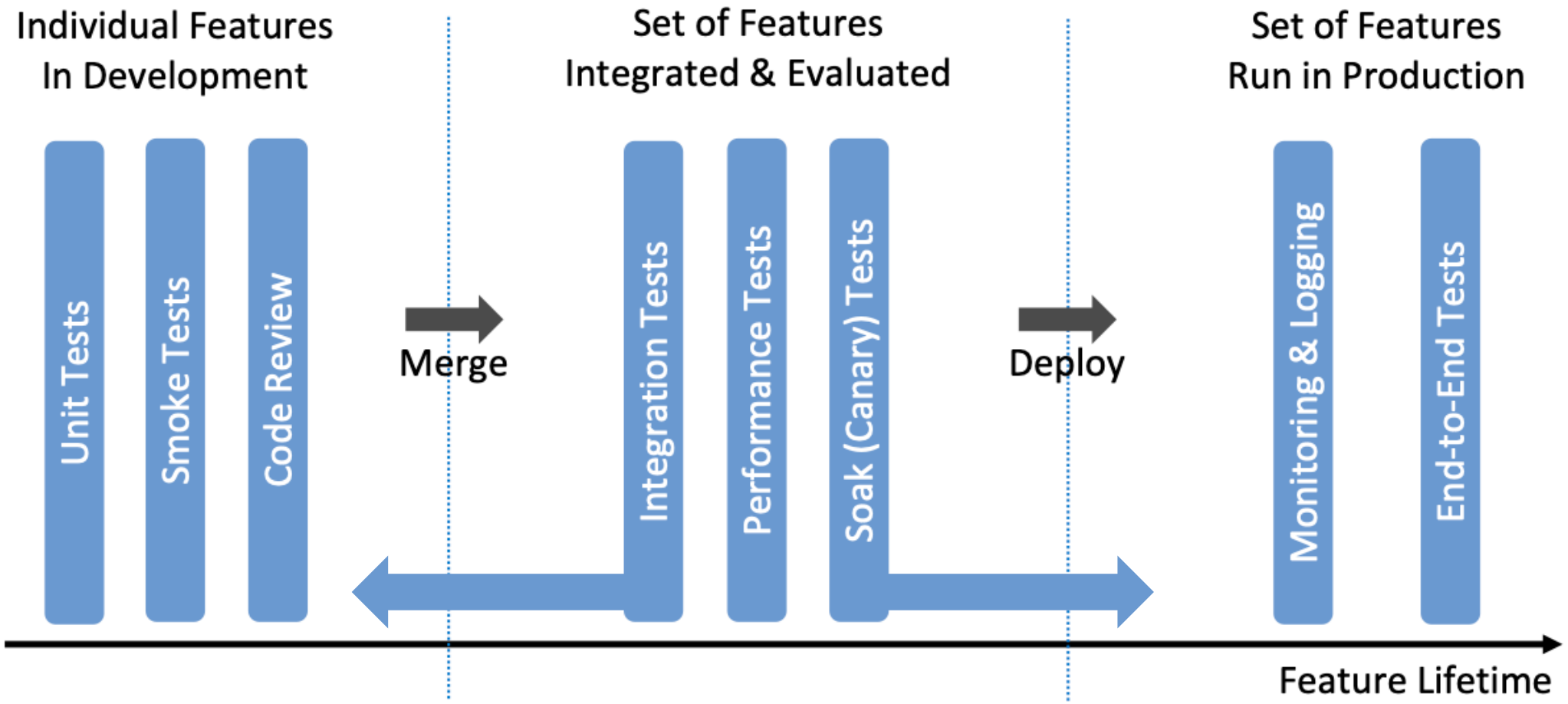
# Testing Strategies



# Testing Strategies

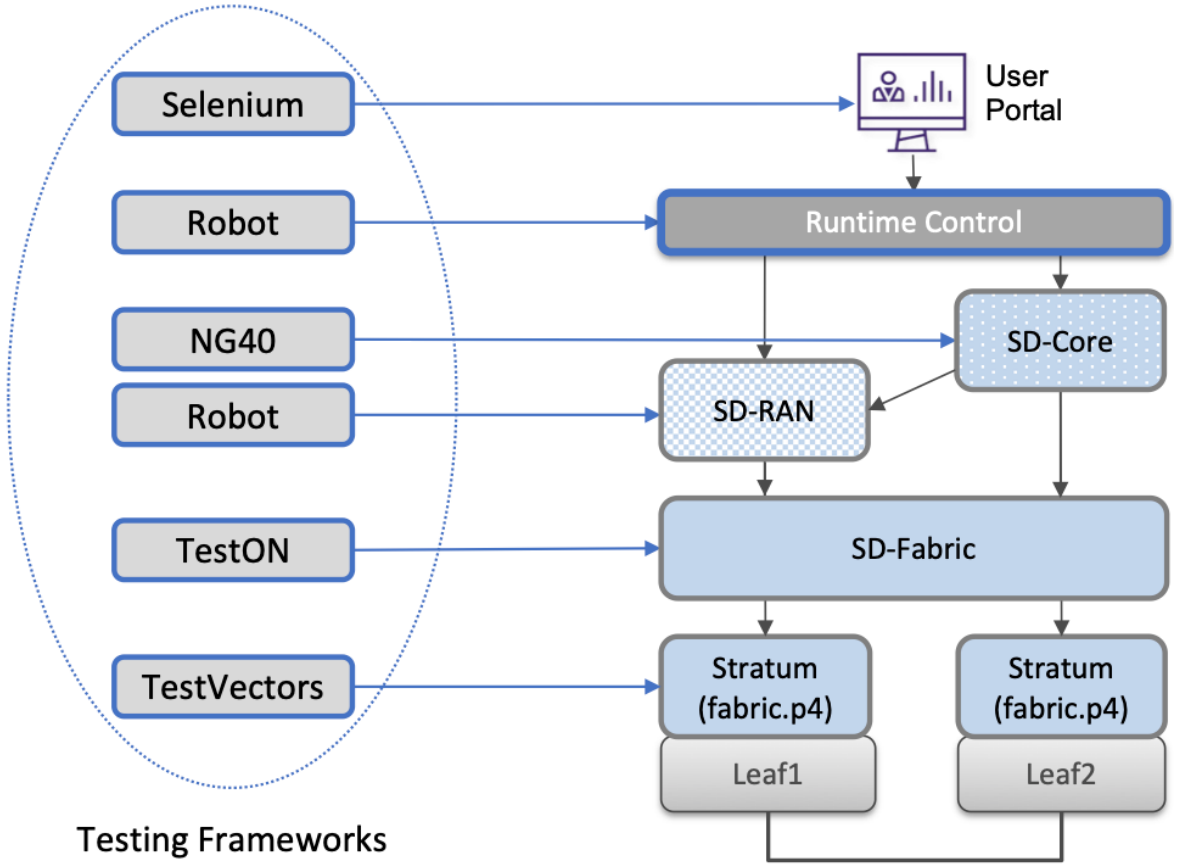


# Testing Strategies

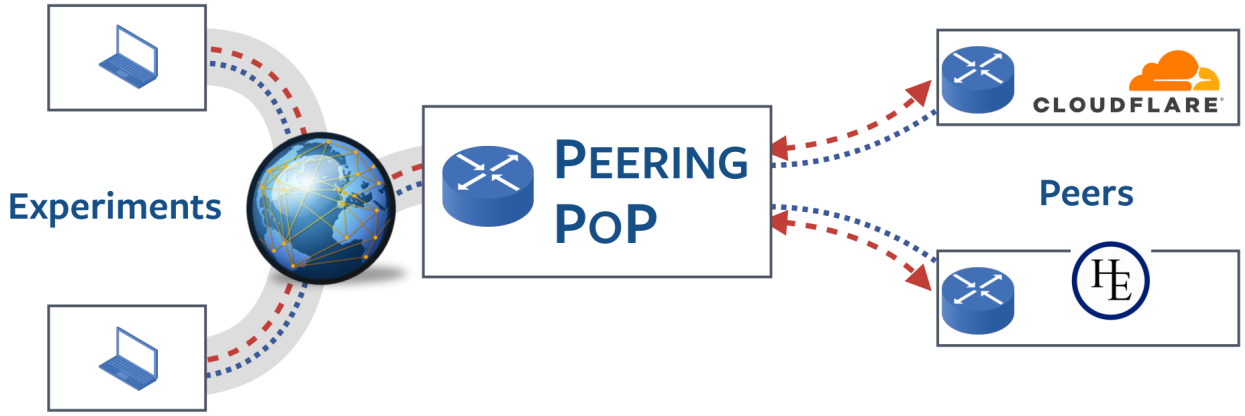


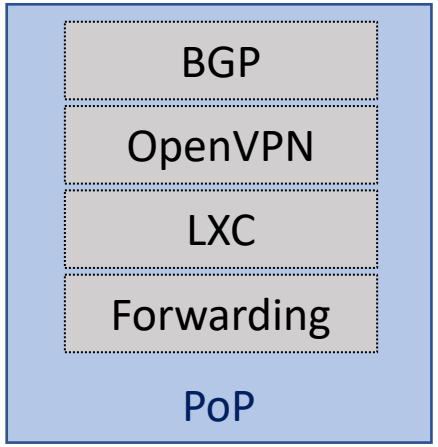
# Testing Framework

- Unit tests are easy to implement using *mocked* objects and data
- Smoke and integration tests often require a lot more tooling

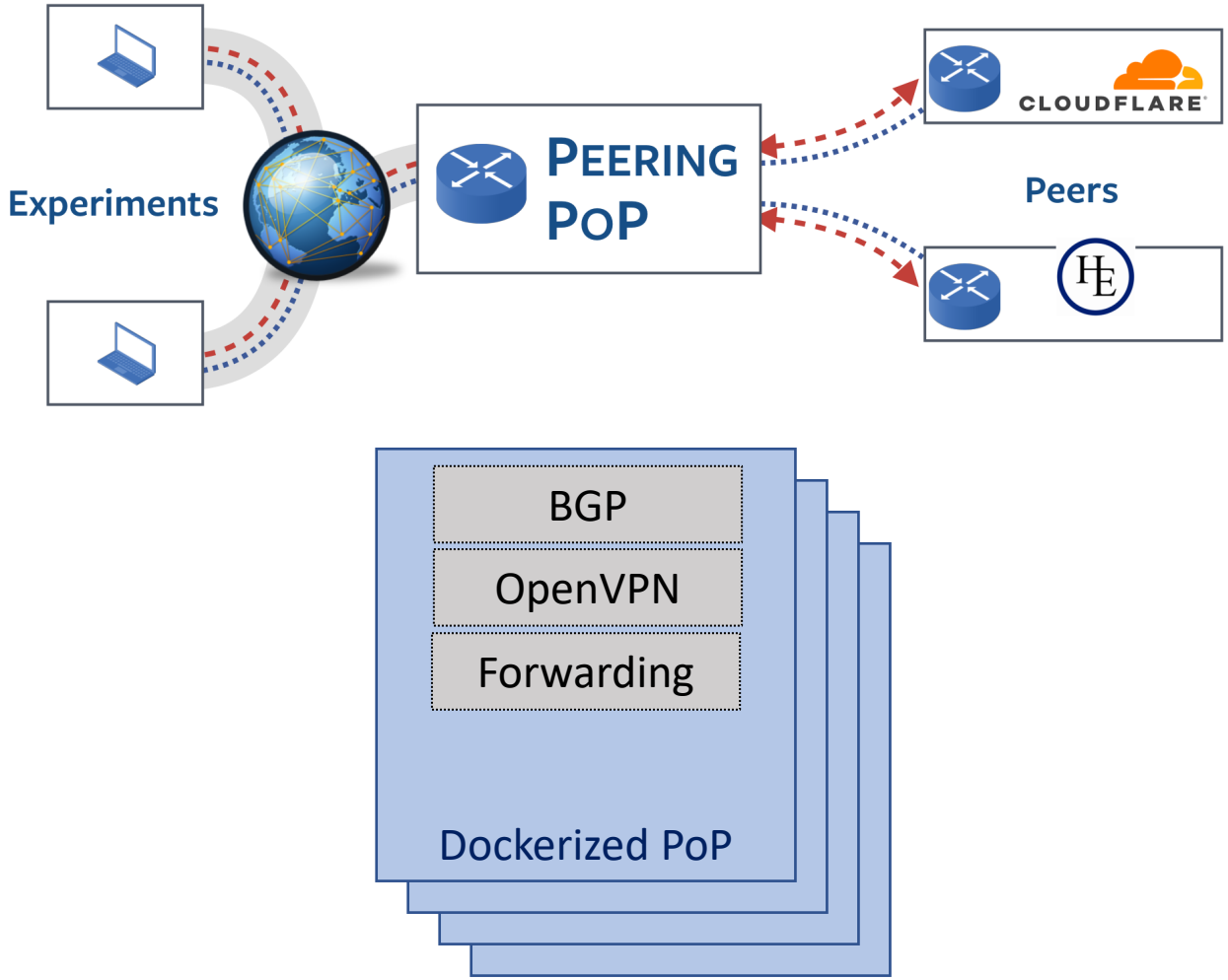


# Testing Framework

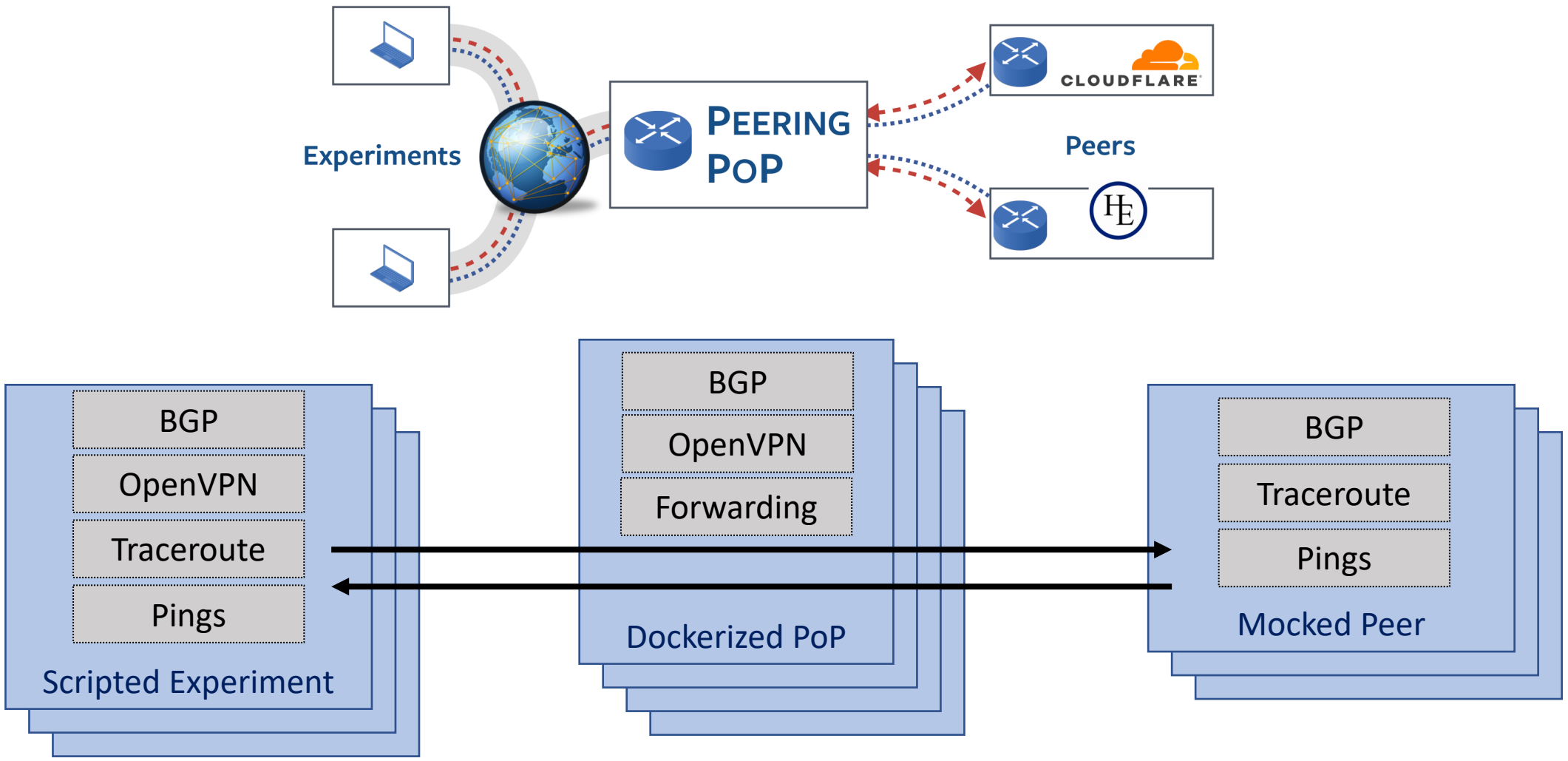


# Testing Framework



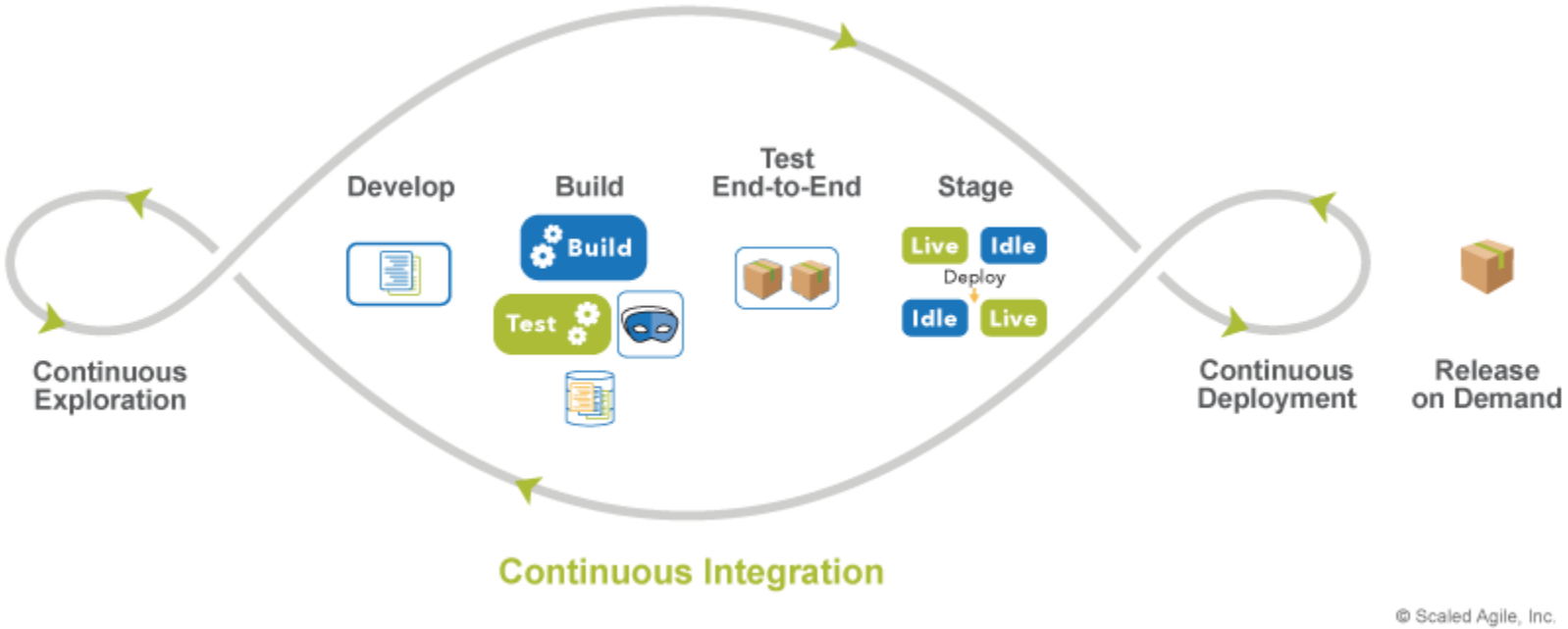
# Testing Framework



# Continuous Integration

Prof. Ítalo Cunha

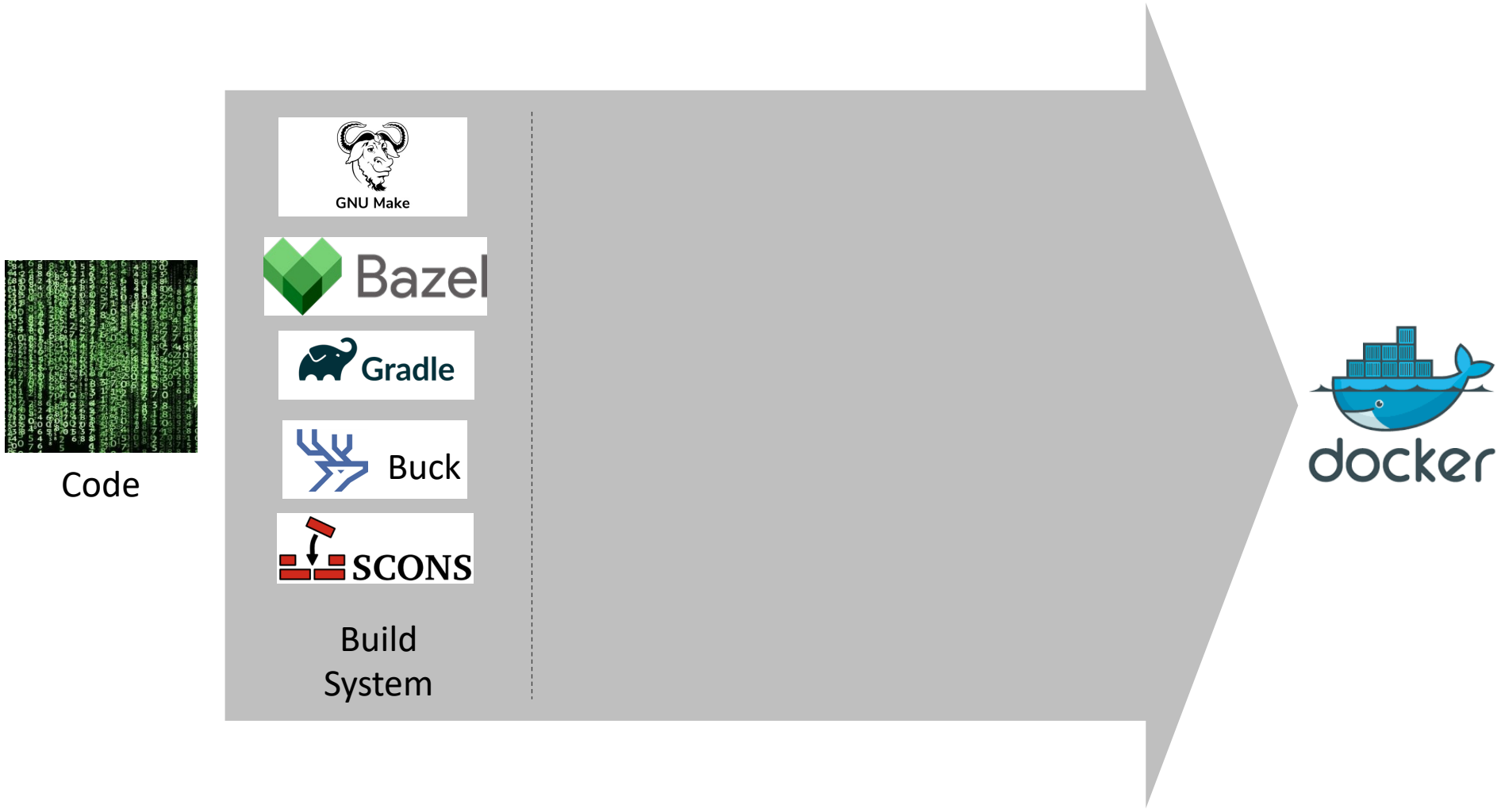
# Continuous Integration



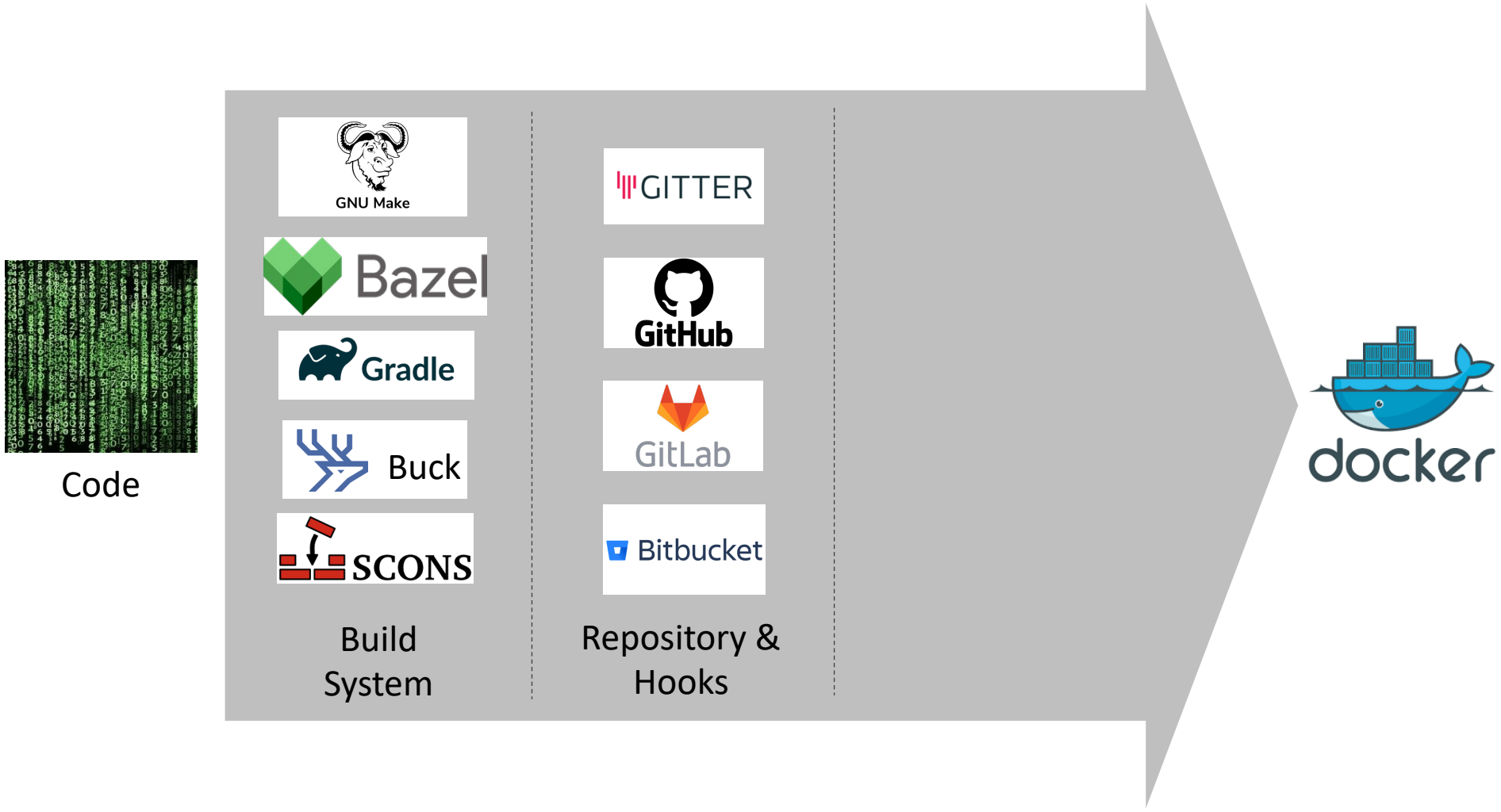
# Continuous Integration

Code

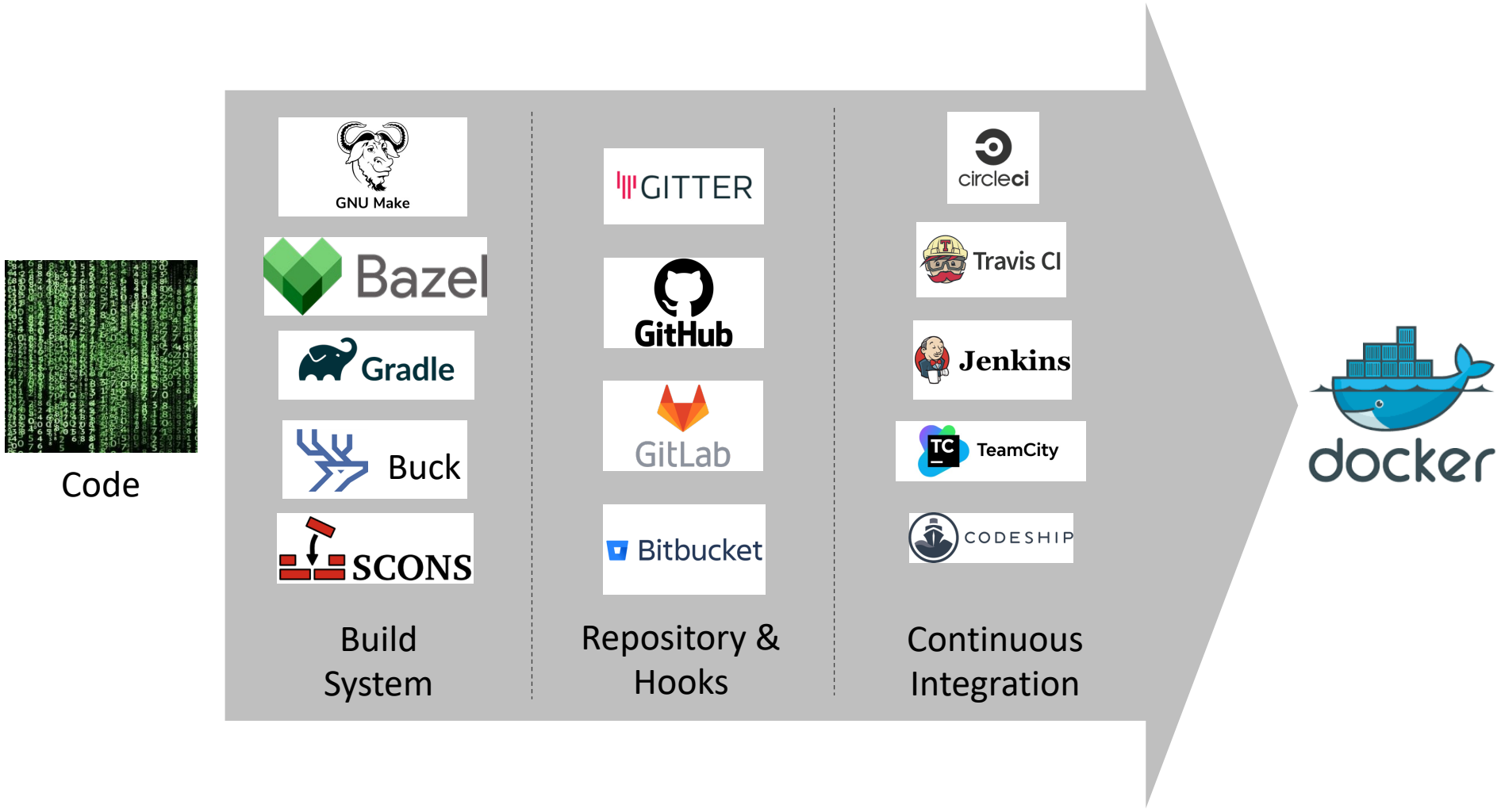
# Continuous Integration



# Continuous Integration



# Continuous Integration



# Continuous Integration

Code

GNU Make

Bazel

Gradle

Buck

SCONS

Build System

GITTER

GitHub

GitLab

Bitbucket

Repository & Hooks

circleci

Travis CI

Jenkins

TeamCity

CODESHIP

Continuous Integration

docker

Configuration

# CI Concepts

- Triggers
  - Dispatch some CI job
  - Git commit, PR submission, periodic triggers
- Job (or pipeline)
  - Execute a set of tests organized in *phases*
- Phase
  - Set of commands to achieve a goal in the testing process
    - Install dependencies
    - Configure frameworks
    - Create Docker images
    - Create requests and check responses
    - Push Docker images

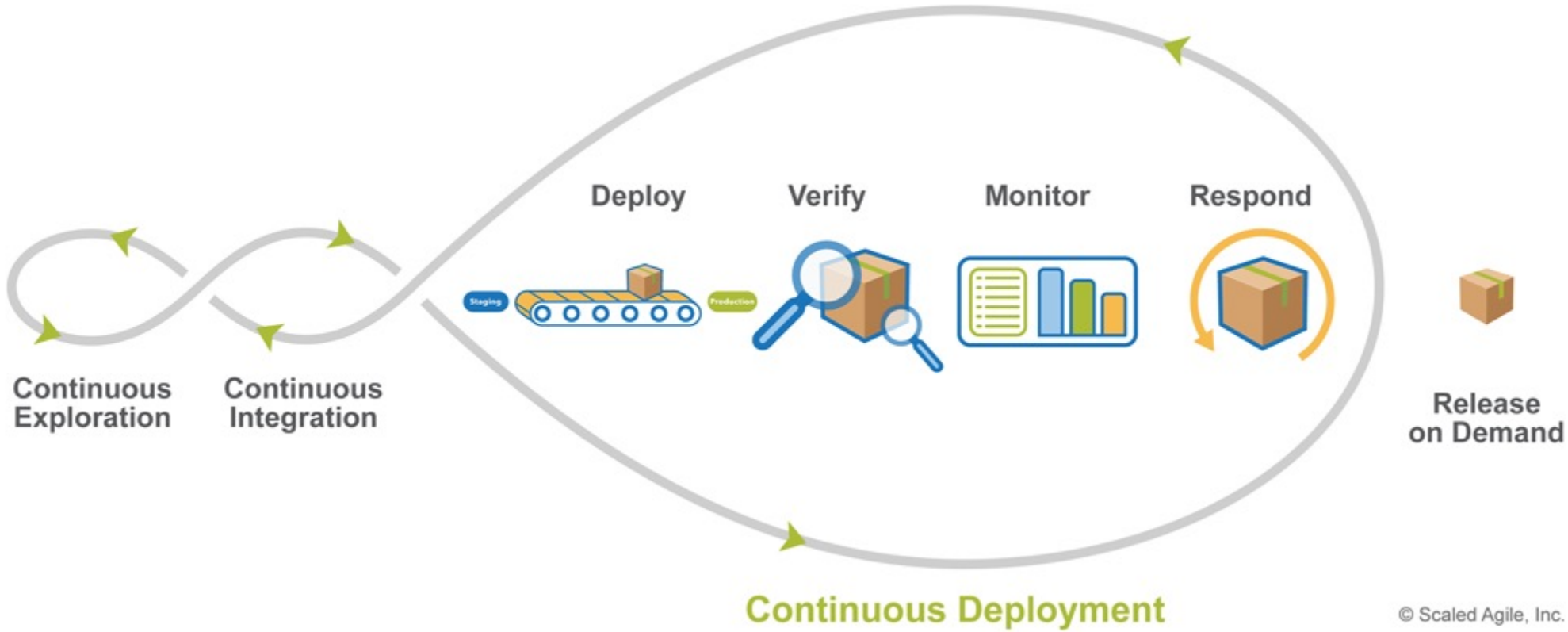
# CI Concepts

- Triggers
  - Dispatch some CI job
  - Git commit, PR submission, periodic triggers
- Job (or pipeline)
  - Execute a set of tests organized in *phases*
- Phase
  - Set of commands to achieve a goal in the testing process
    - Install dependencies
    - Configure frameworks
    - Create Docker images
    - Create requests and check responses
    - Push Docker images

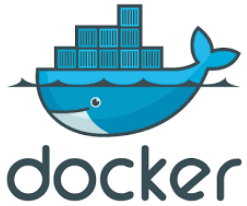
Declarative configuration for everything. Definitions committed to the code or config repositories.

# Continuous Delivery/Deployment

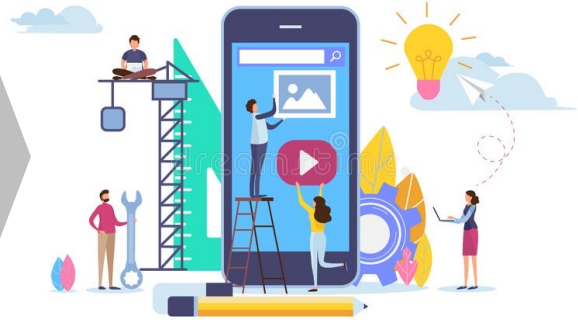
Prof. Ítalo Cunha



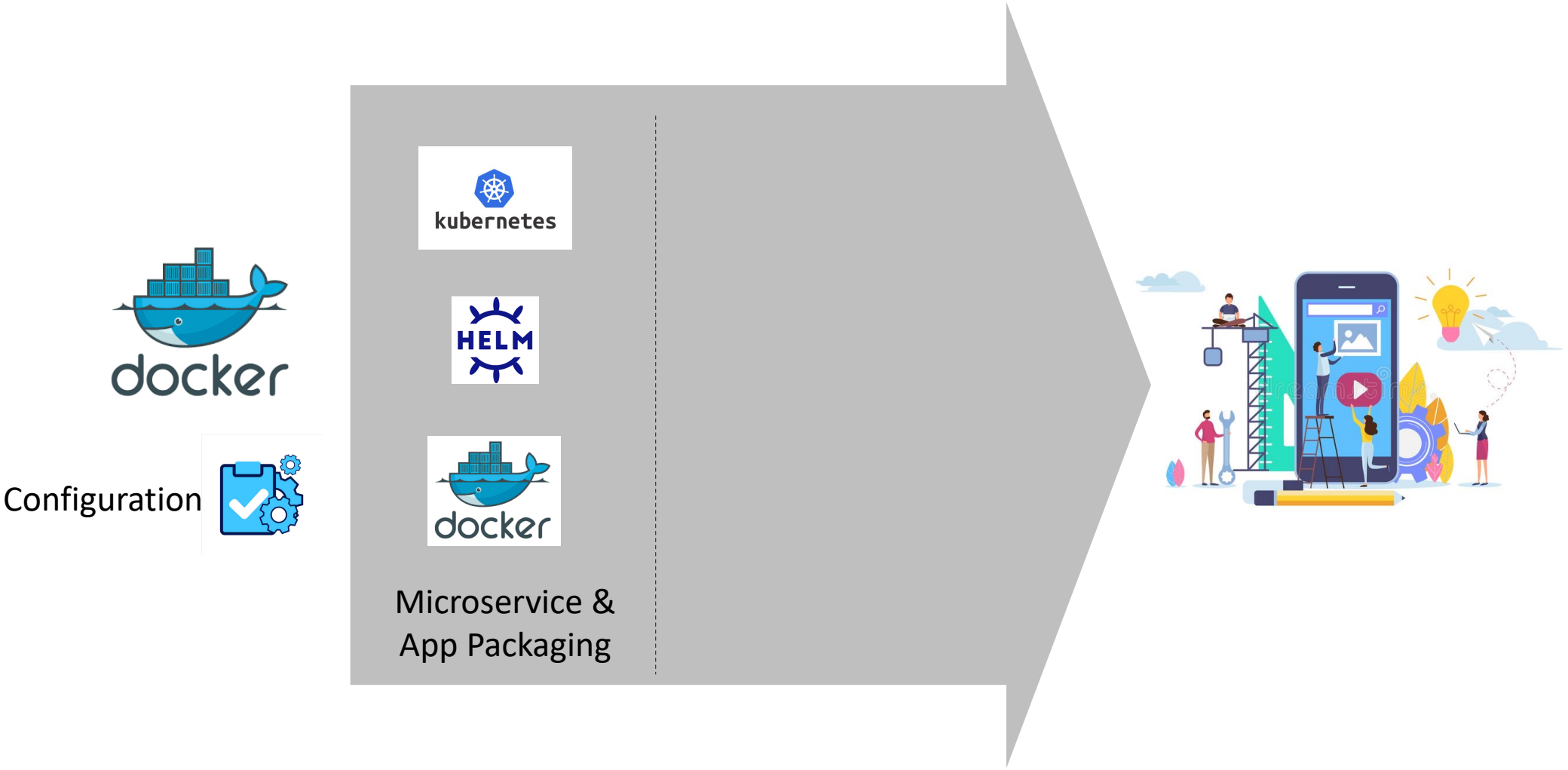
# Continuous Deployment



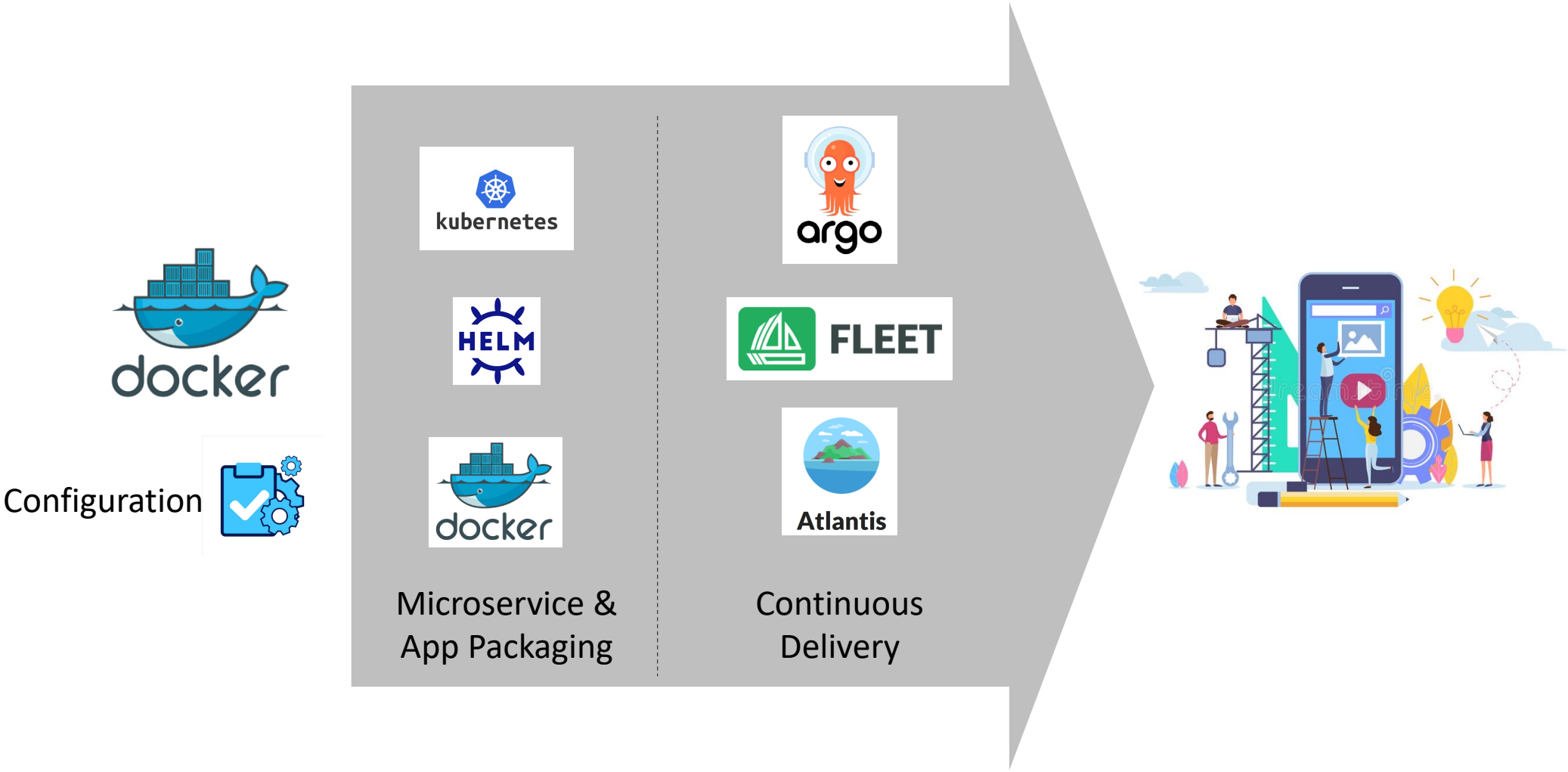

Configuration 



# Continuous Deployment



# Continuous Deployment



# CD Concepts

- Microservice
  - Kubernetes Pod + Deployment + Service
  - Helm Chart
- Resources
  - CPU, memory, volumes, load balancers, IPs
- Application, Template, Bundle
  - Set of related microservices
  - Fleet Bundle, Kubernetes ApplicationSet
- Kubernetes clusters
  - Cluster with physical resources where deployments occur

# CD Concepts

- Microservice
  - Kubernetes Pod + Deployment + Service
  - Helm Chart
- Resources
  - CPU, memory, volumes, load balancers, IPs
- Application, Template, Bundle
  - Set of related microservices
  - Fleet Bundle, Kubernetes ApplicationSet
- Kubernetes clusters
  - Cluster with physical resources where deployments occur

Declarative configuration for everything. Definitions committed to the code or config repositories.

# Versioning

Prof. Ítalo Cunha

# Tracking what is deployed

- CD needs do know what version ***is*** deployed and what version ***should*** be deployed
  - Consistent versioning is necessary
- Versioning strategies
  - Semantic versioning (`major.minor.patch`)
  - Date and time

# Versioning across lifecycle

**Development**
**Time**

- Developers commit code
- Track versions across commits (-dev)
- Tag releases

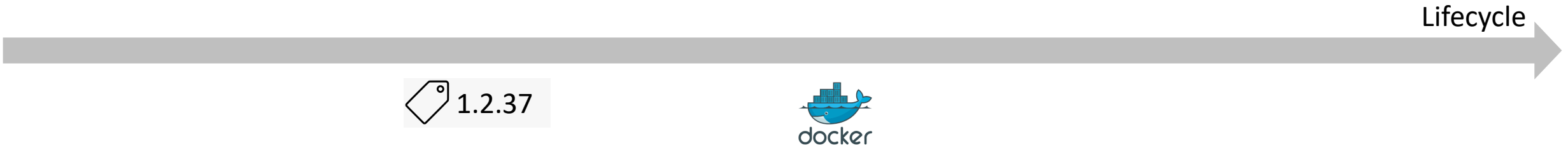
# Versioning across lifecycle

**Development Time**

- Developers commit code
- Track versions across commits (-dev)
- Tag releases

**Integration Time**

- Build images
- Upload images



# Versioning across lifecycle

**Development Time**

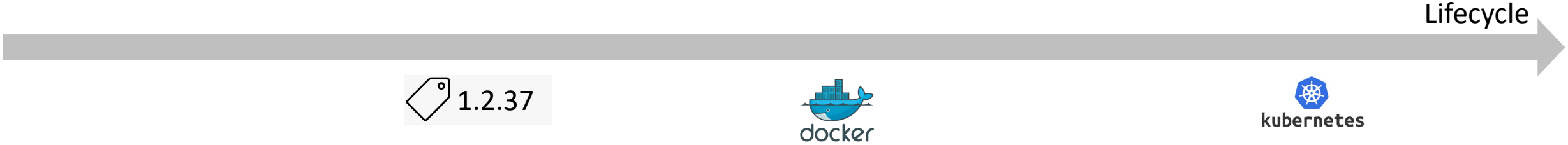
- Developers commit code
- Track versions across commits (-dev)
- Tag releases

**Integration Time**

- Build images
- Upload images

**Deployment Time**

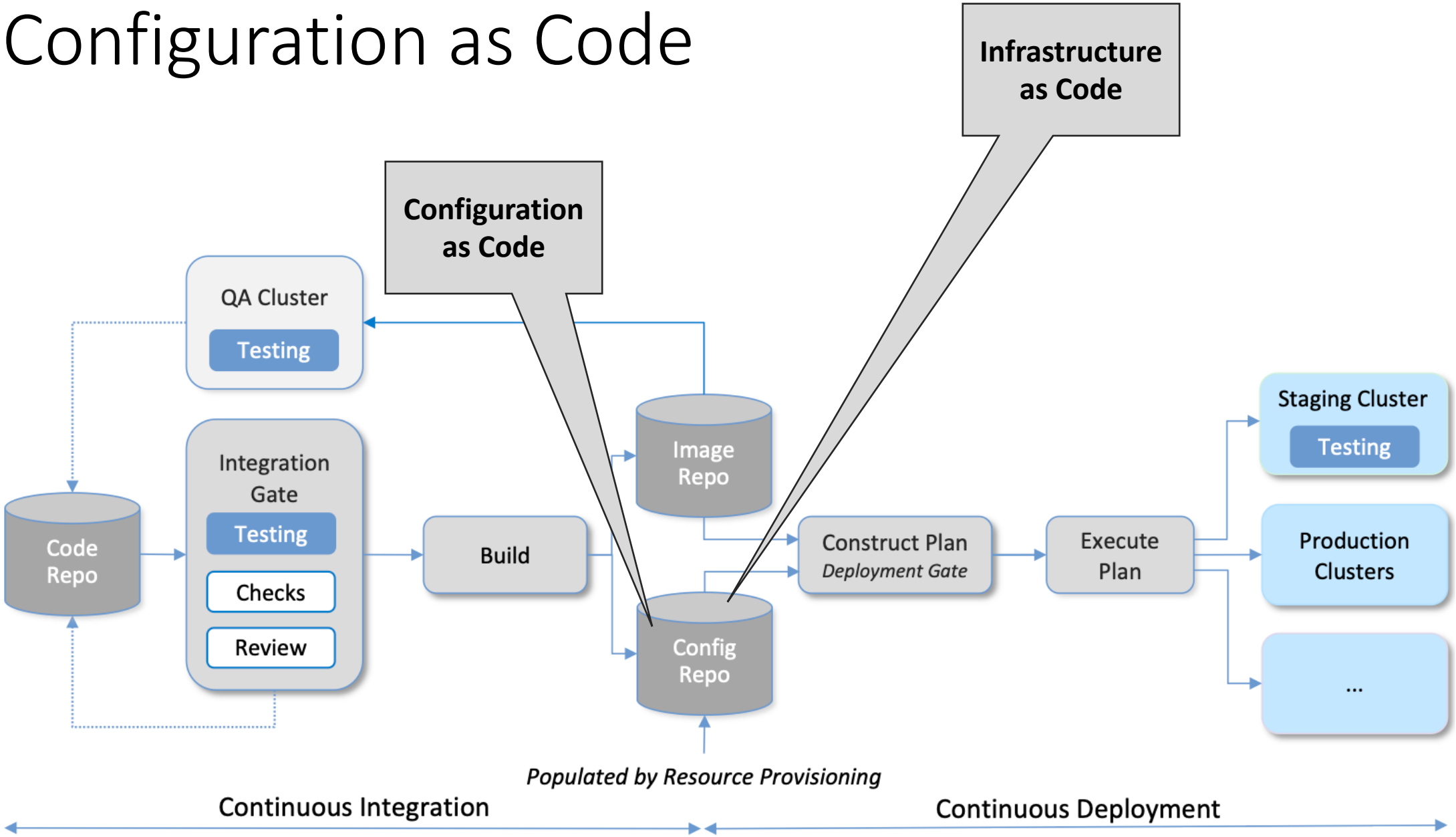
- Track deployment config
- Rollout new images



# GitOps

Prof. Ítalo Cunha

# Configuration as Code



# GitOps / Configuration as Code

- Automating operations from Git
  - What we have been talking about
- Checking in configuration in repository
  - Declarative configuration
  - Configuration is picked up by tooling
    - Integration
    - Deployment
  - Single place with code and configuration
    - From infrastructure to deployment
- Works well for mostly static configuration set by operators

# Not all configuration is created equal

- Some configuration items are ill-suited to be stored as code
- Operators of a platform and users of the platform are different
  - Users may want to change their own configuration
  - Users may be external
  - Users may want control over aspects of the platform operators have not parameterized
- Some configurations may change frequently or dynamically
  - Runtime configuration
  - Cannot be conveniently checked in to Git

# Managing secrets

- GitOps requires secrets to be available to automation tools
  - Passwords, certificates, private keys, …
- Checking in secrets to a Git repo is a no-no
  - Many users have read access, security vulnerability
- Need solution for sharing secrets between CI, CD, and microservices
  - git-crypt
  - Kubernetes SealedSecrets: Only make secret available within cluster