TOP100

全球软件案例研究峰会

100

TOP 100 CASE STUDIES OF THE YEAR

全球软件案例研究峰会


Google has been developing and using containers to manage our applications for **over 10 years.**
Images by Connie Zhou

Everything at Google runs in containers:

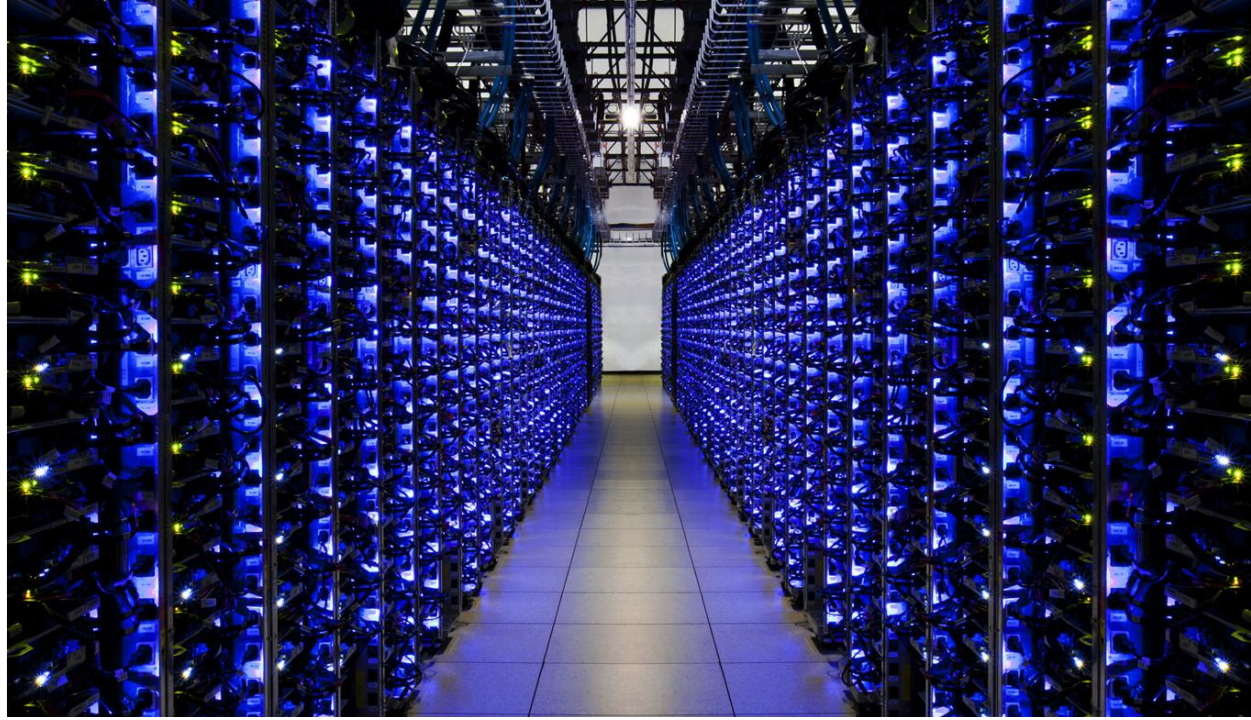
- Gmail, Web Search, Maps, ...
- MapReduce, batch, ...
- GFS, Colossus, ...
- Even GCE itself: VMs in containers

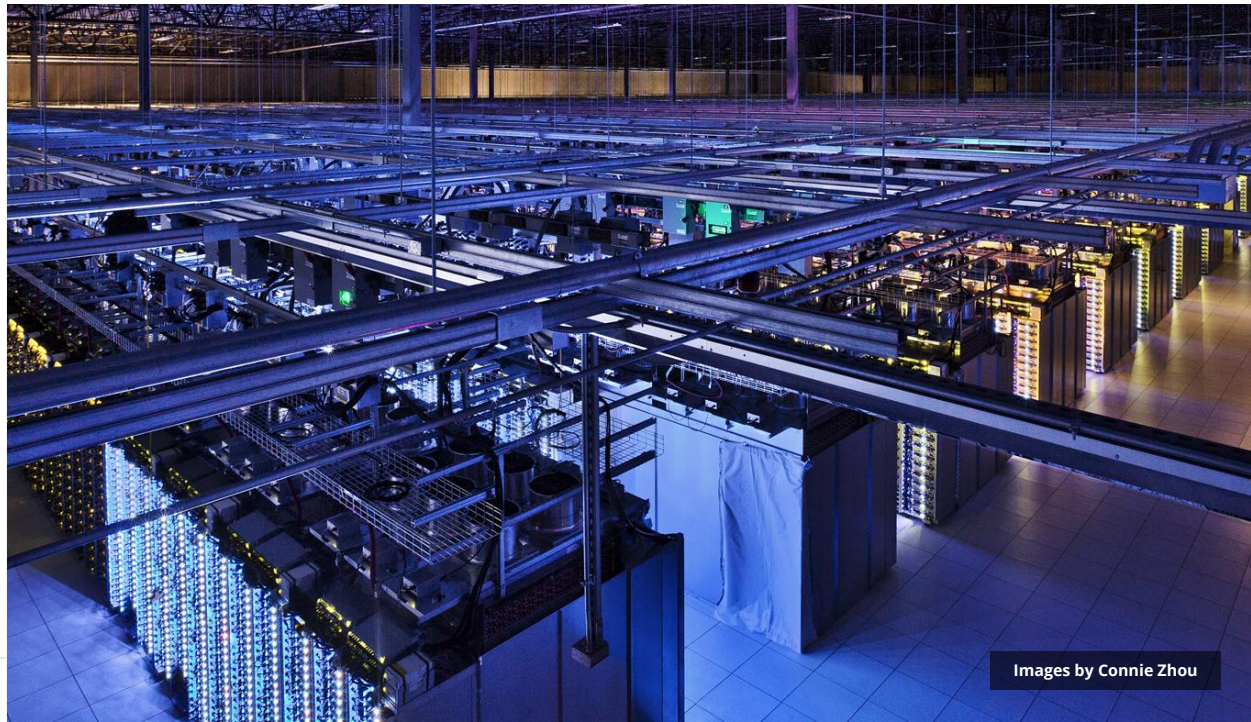
We launch over **2 billion** containers **per week**.

Shipping Containers At Clyde, by Steve Gibson

Why containers?

- Performance
- Repeatability
- Isolation
- Quality of service
- Accounting
- Visibility





Images by Connie Zhou

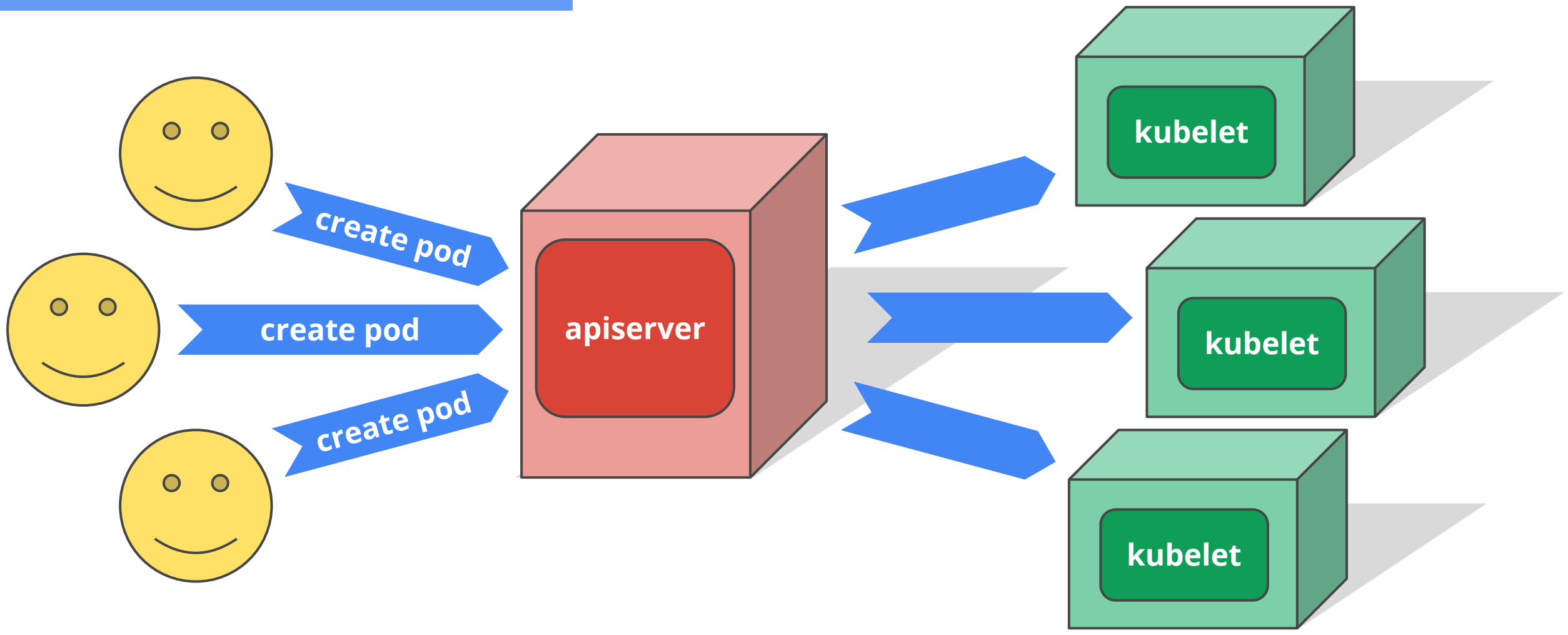
Enter Kubernetes

Greek for *"Helmsman"*; also the root of the word *"Governor"*

- Container orchestrator
- Runs Docker containers
- Supports multiple cloud and bare-metal environments
- Inspired and informed by Google's experiences
- Open source

Manage **applications**, not machines

High Level Design



Primary Concepts

Container: A sealed application package (Docker)

Pod: A small group of tightly coupled Containers

example: content syncer & web server

Controller: A loop that drives the current state towards the desired state

example: replication controller

Service: A set of running pods that work together

example: load-balanced backends

Labels: Identifying metadata attached to other objects

example: phase=canary vs. phase=prod

Selector: A query against labels, producing a set result

example: all pods where label phase == prod

Design Principles

Declarative > imperative: State your desired results

Control loops: Observe, rectify, repeat

Modularity: Components, interfaces, & plugins

Legacy compatible: Requiring apps to change is a non-starter

Network-centric: IP addresses are cheap

No grouping: Labels are the only groups

Control Loops

Drive **current state** -> **desired state**

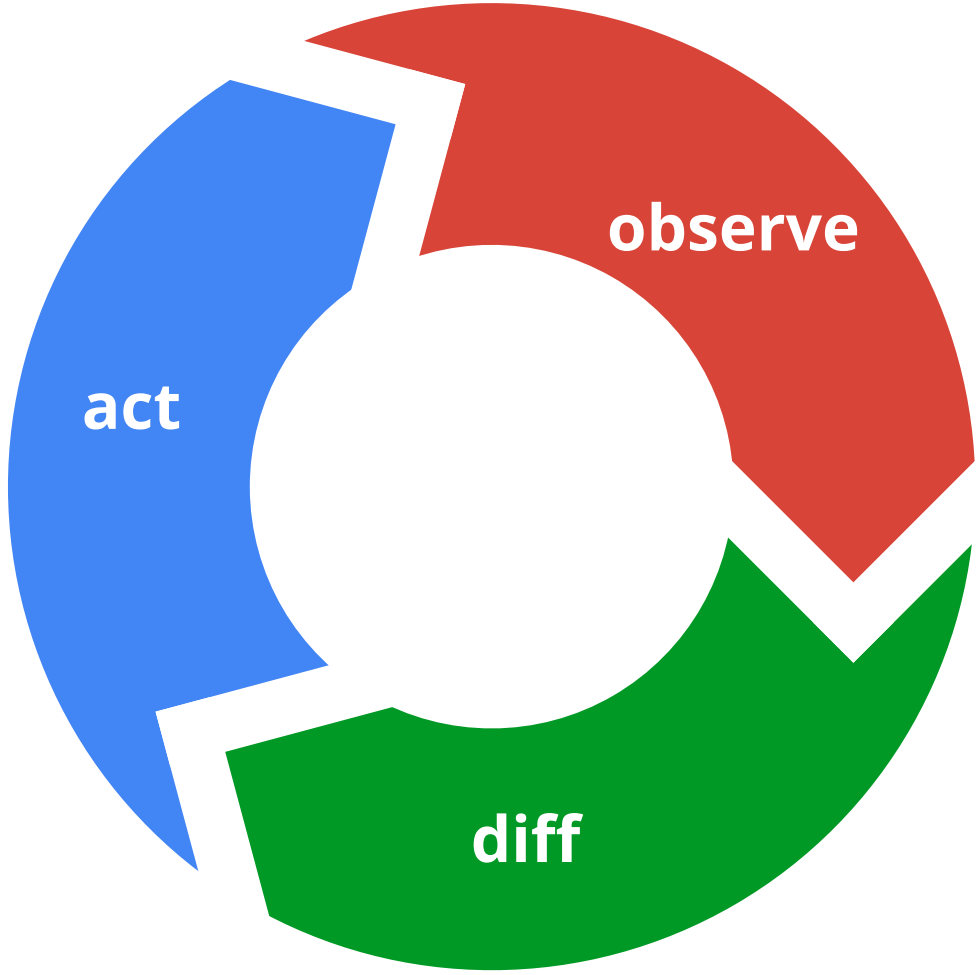
Act independently

APIs - **no shortcuts** or back doors

Observed state is truth

Recurring pattern in the system

Example: ReplicationController



Modularity

Loose coupling is a goal **everywhere**

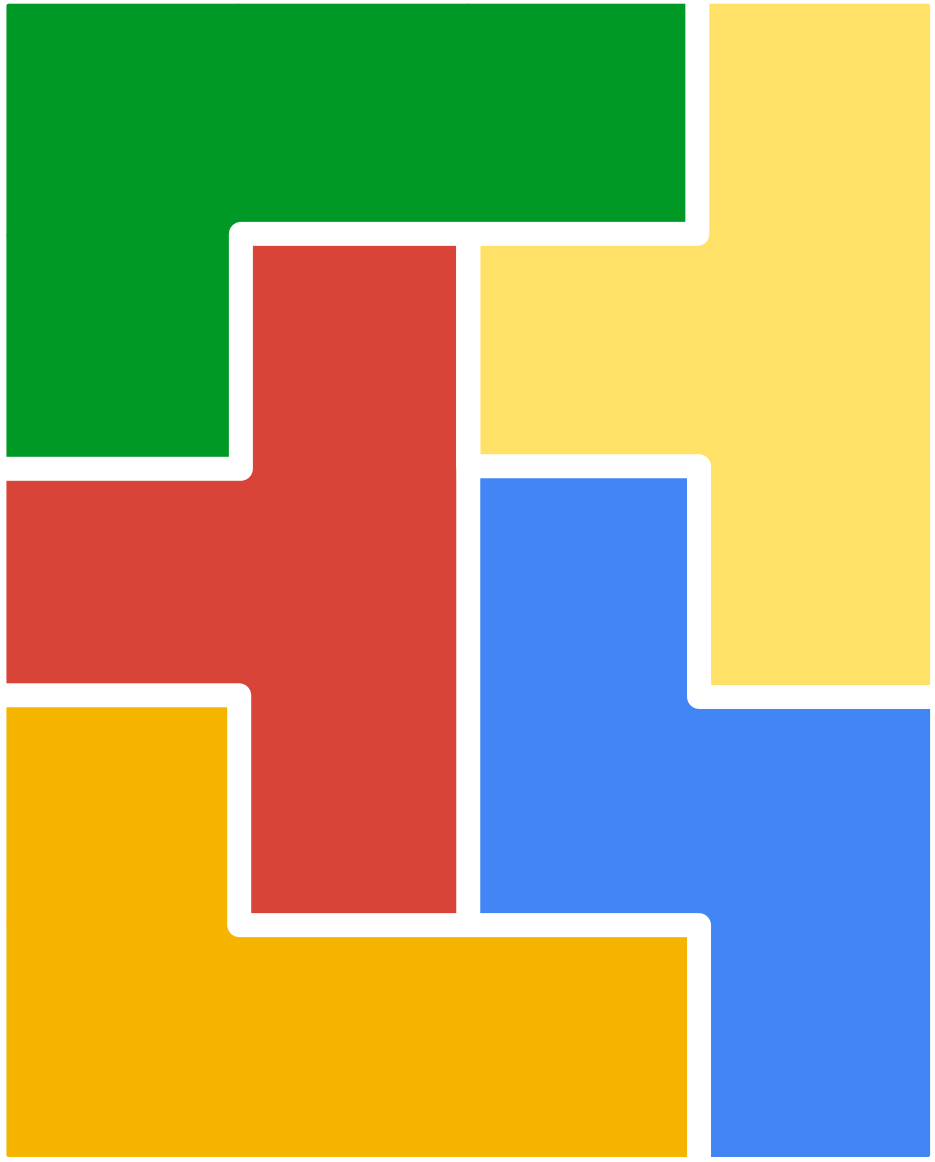
- simpler
- composable
- extensible

Code-level plugins where possible

Multi-process where possible

Isolate risk by interchangeable parts

Example: ReplicationController
Example: Scheduler



Atomic Storage

Hidden behind an abstract interface

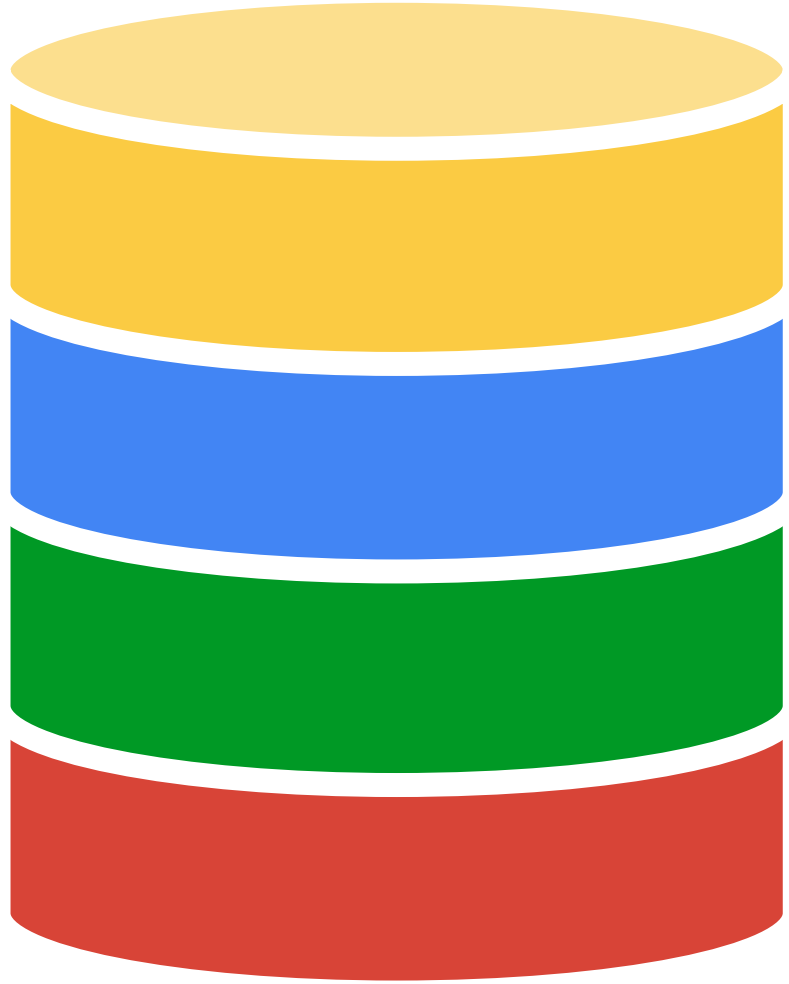
Backing store for master state

Stateless means scalable

Watchable

- this is a fundamental primitive

Using **CoreOS etcd**



Labels

Arbitrary metadata

Attached to any API object

Generally represent **identity**

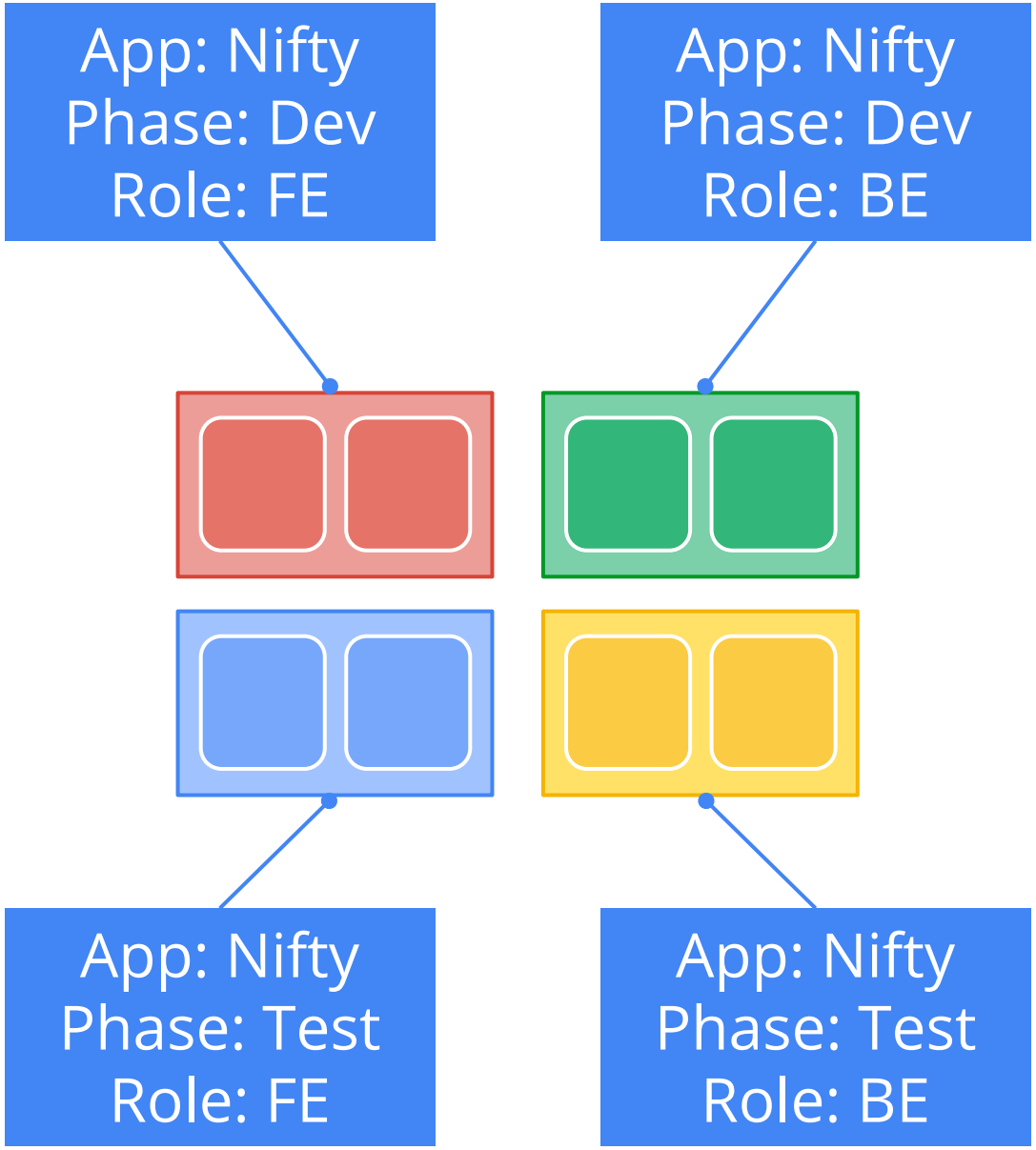
Queryable by **selectors**

- think SQL '*select … where …*'

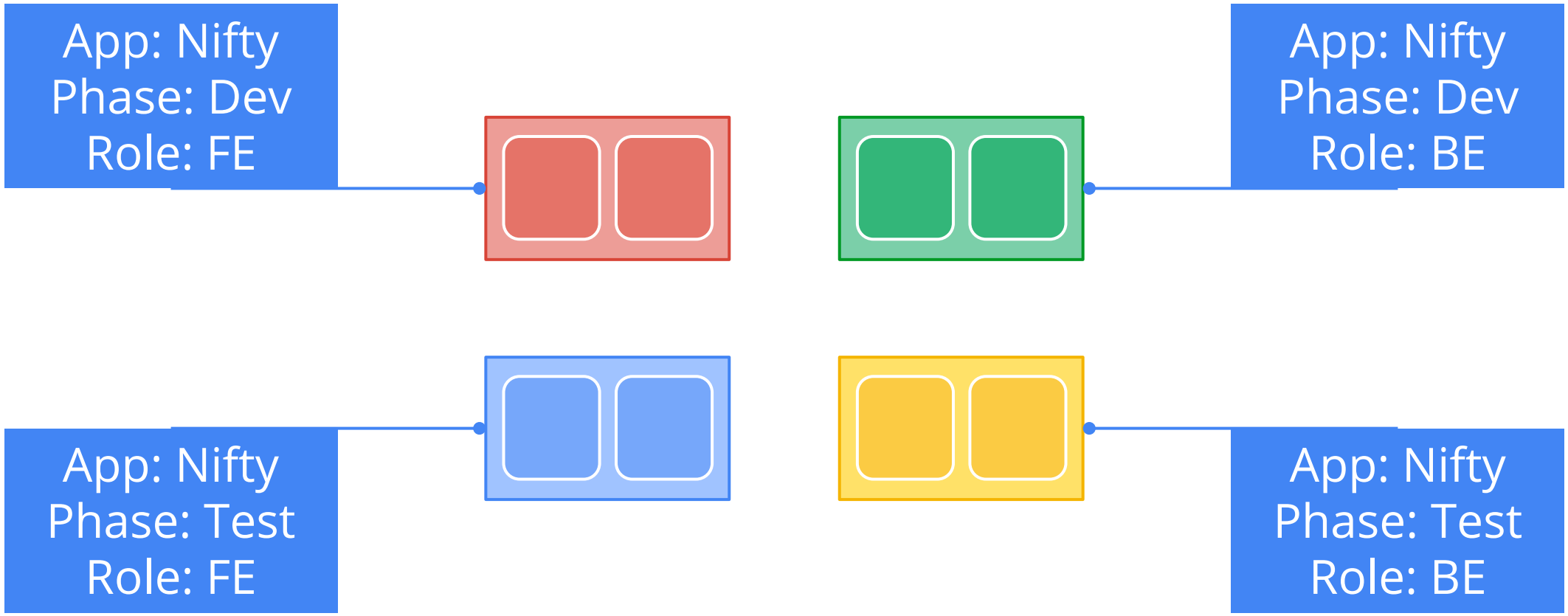
The **only** grouping mechanism

- pods under a ReplicationController
- pods in a Service

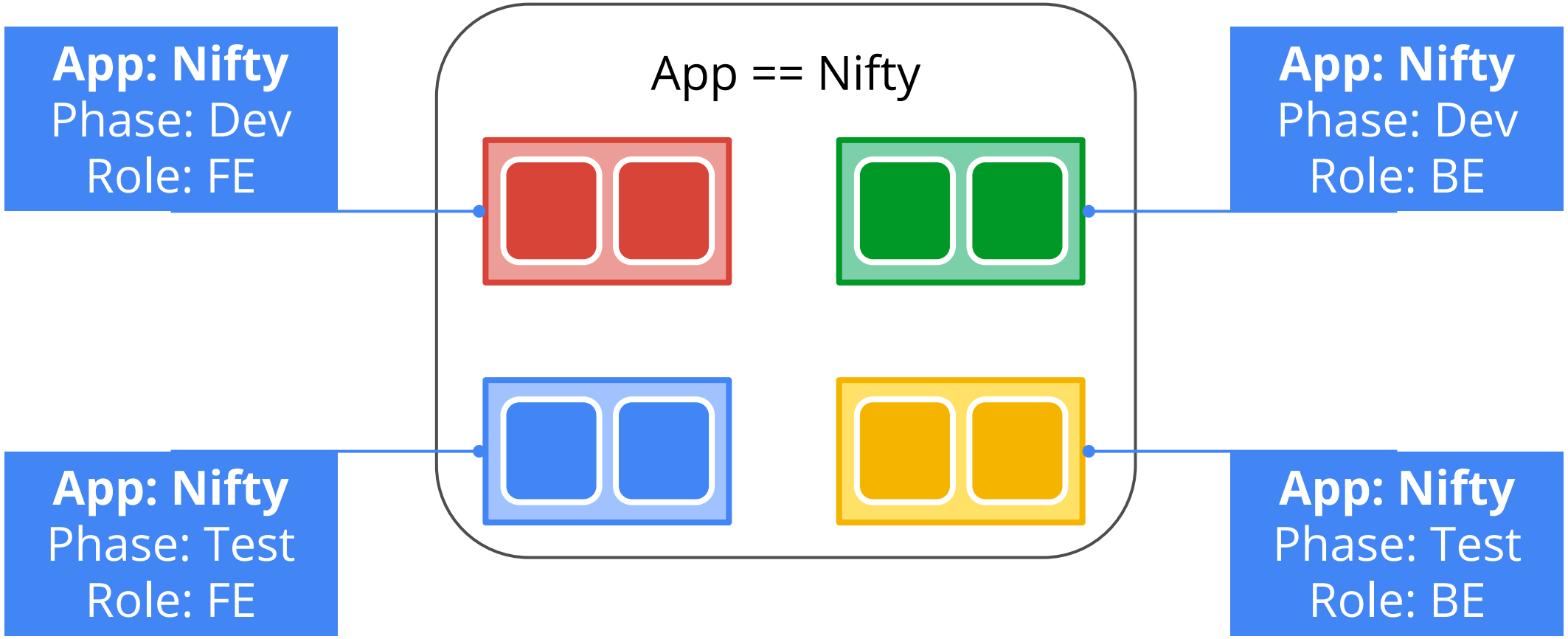
Example: "phase: canary"



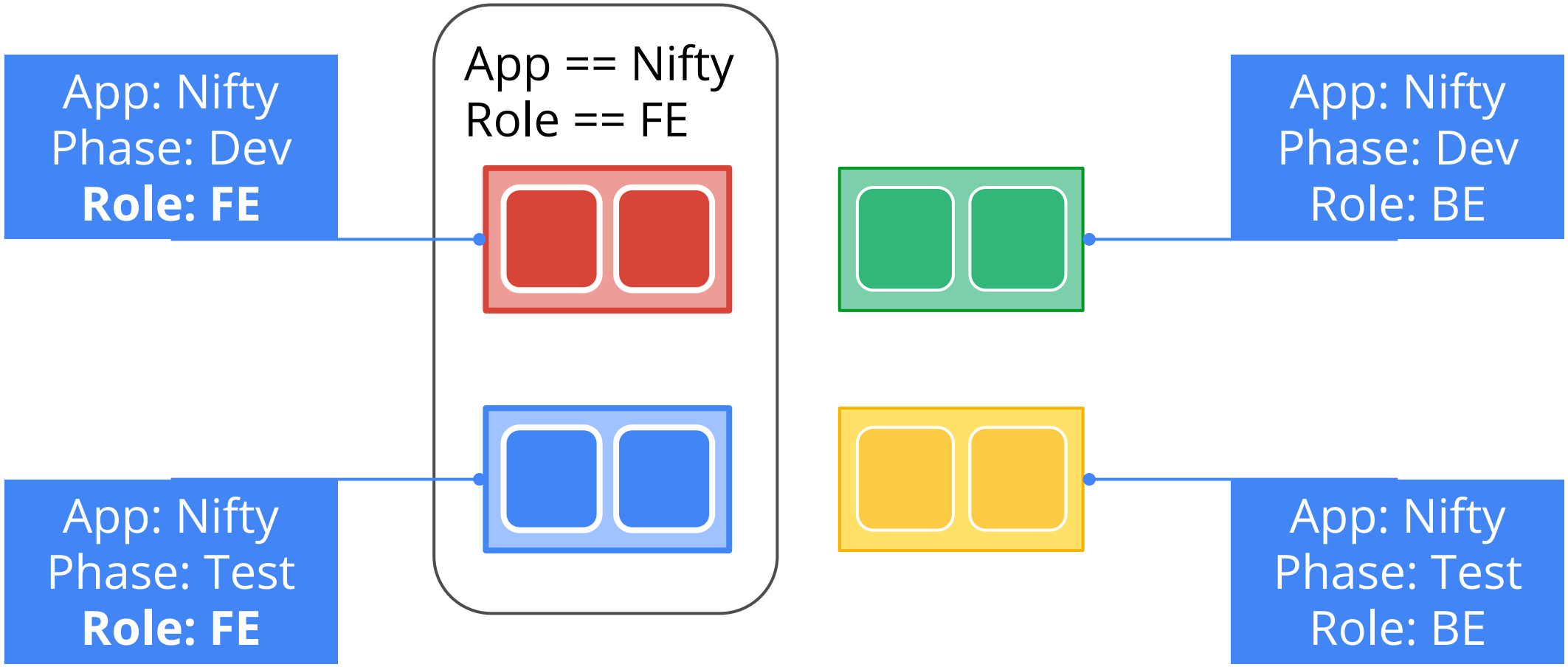
Selectors



Selectors



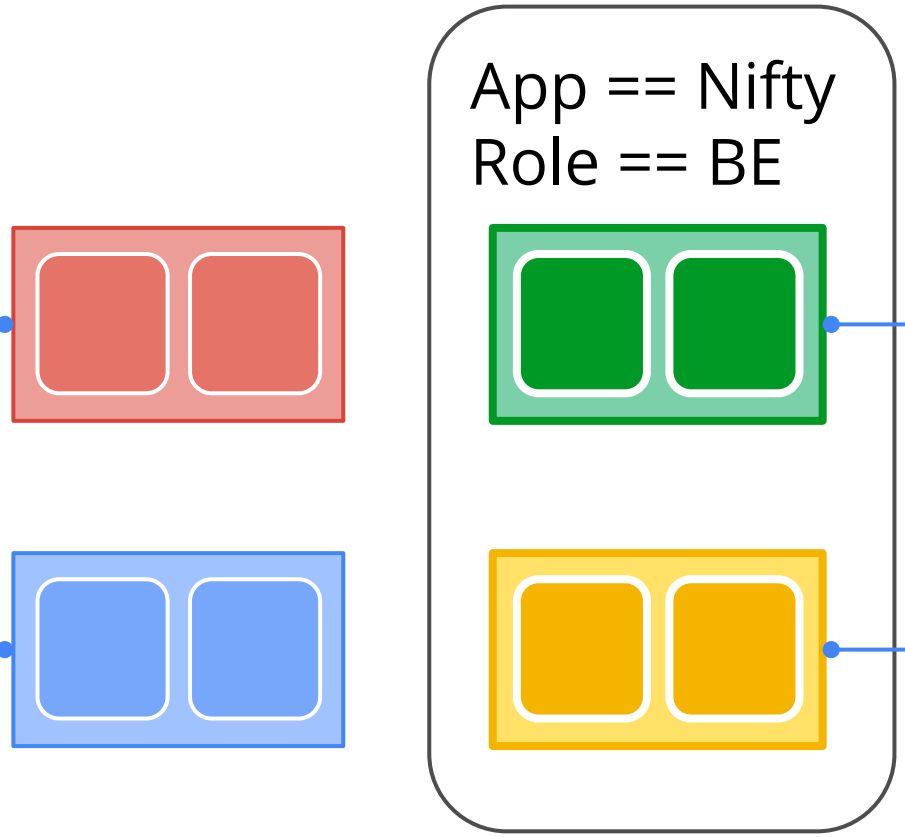
Selectors



Selectors

App: Nifty
Phase: Dev
Role: FE

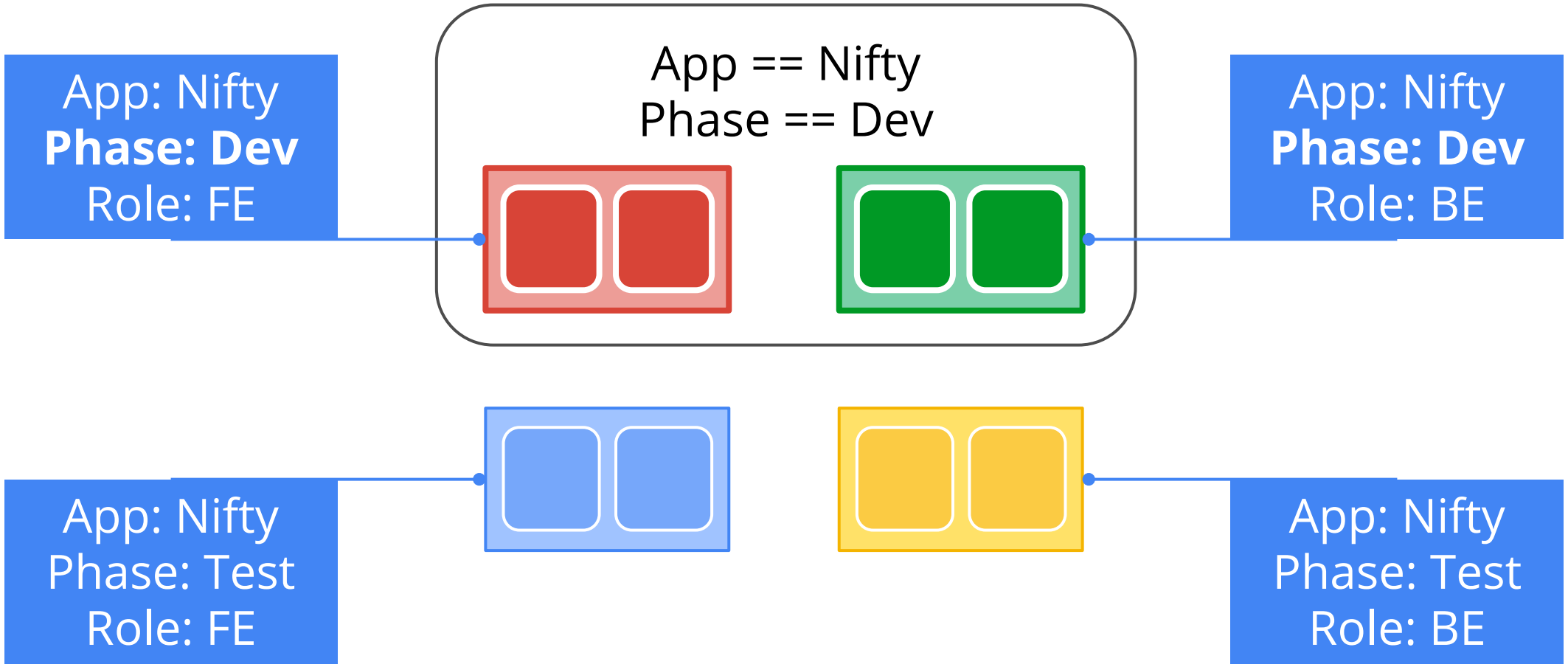
App: Nifty
Phase: Test
Role: FE



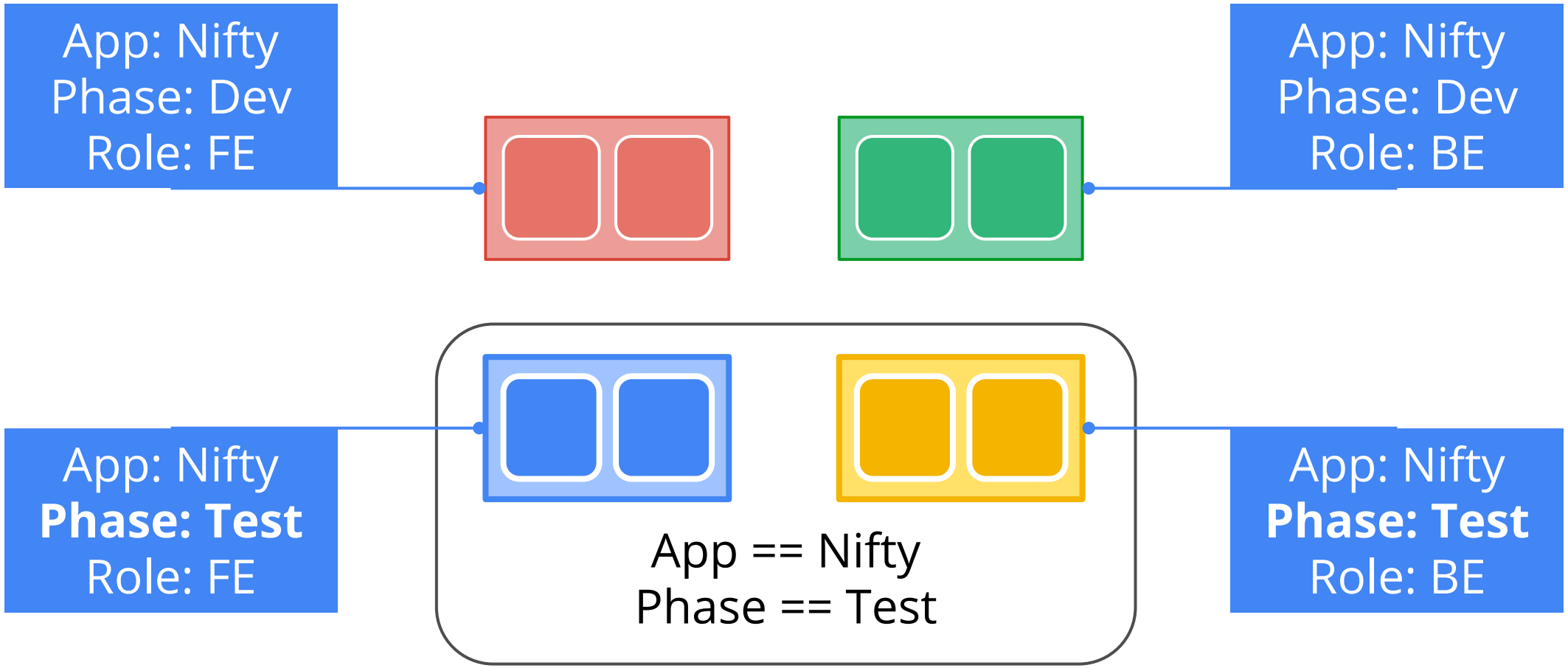
App: Nifty
Phase: Dev
Role: BE

App: Nifty
Phase: Test
Role: BE

Selectors



Selectors



Pods

Small group of containers & volumes

Tightly coupled

Scheduling atom

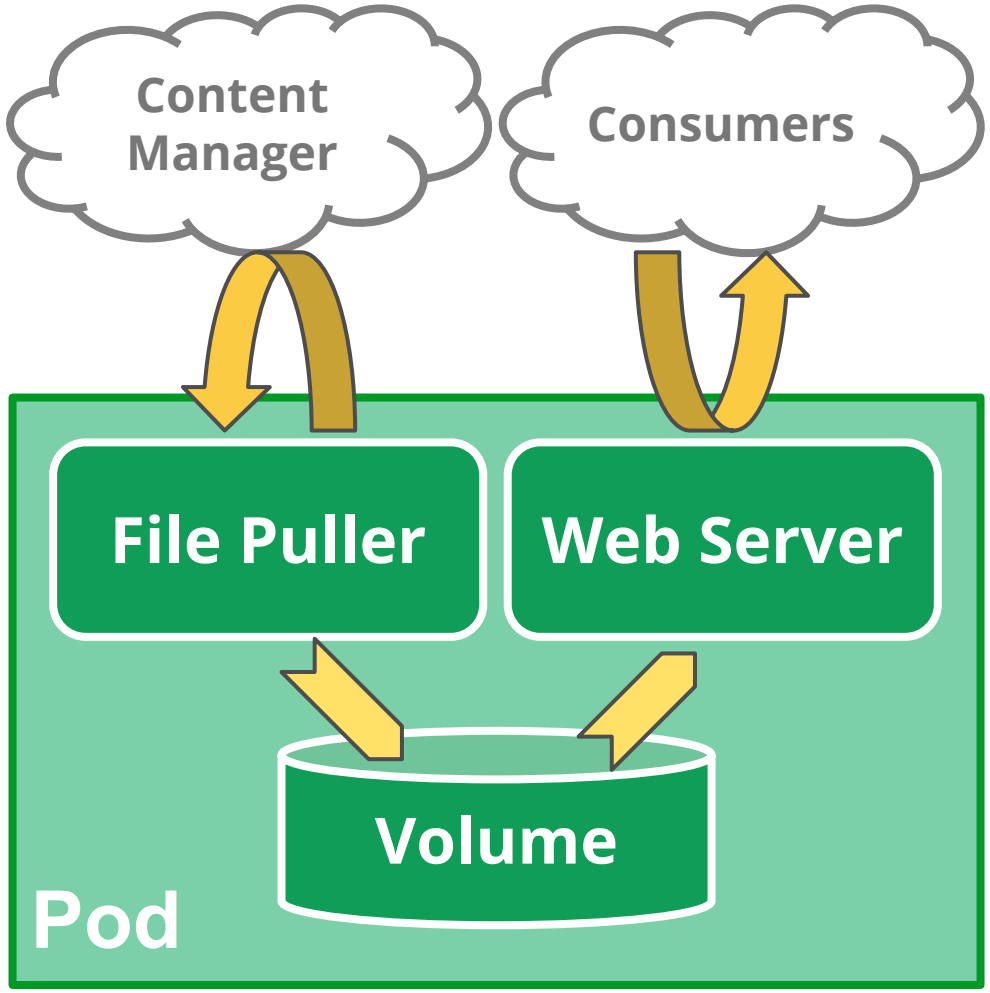
Shared namespace

- **share IP** address & localhost

Ephemeral

- can die and be replaced

Example: data puller & web server



Pod Networking

Pod IPs are **routable**

- Docker default is private IP

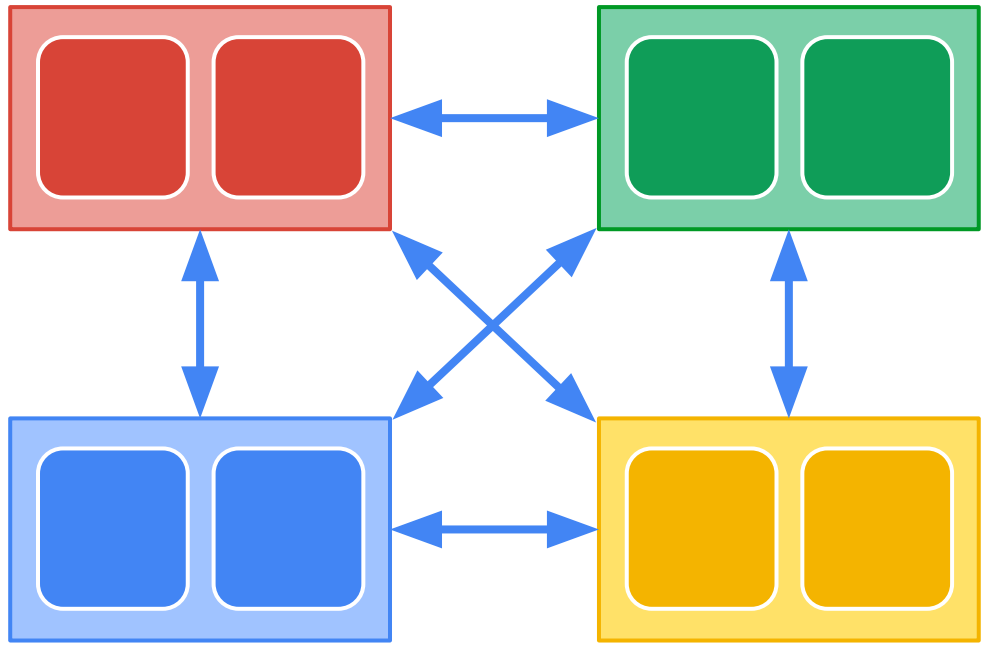
Pods can ping each other

- even across nodes

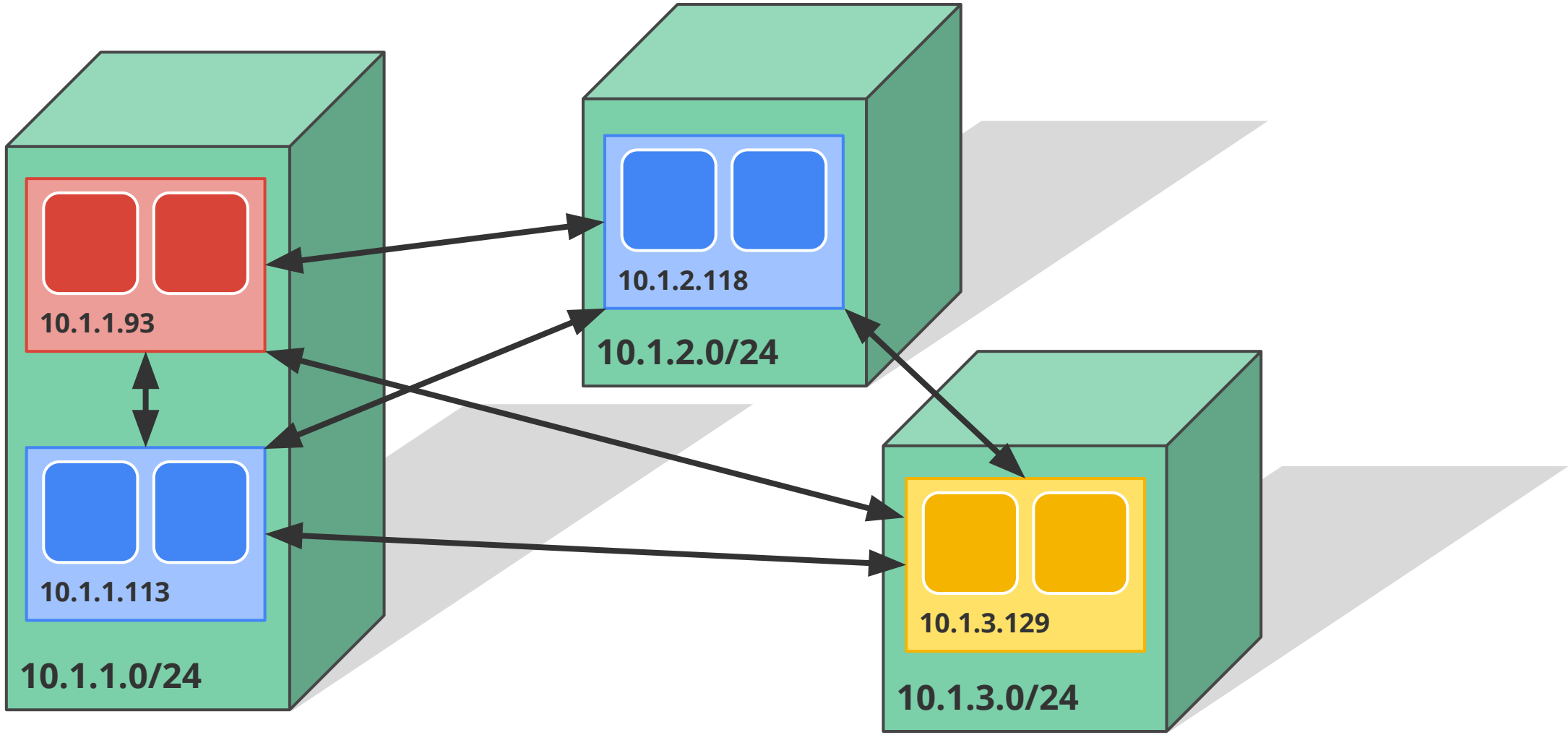
Pods can egress

- if allowed by cloud environment

Fundamental requirement



Pod Networking



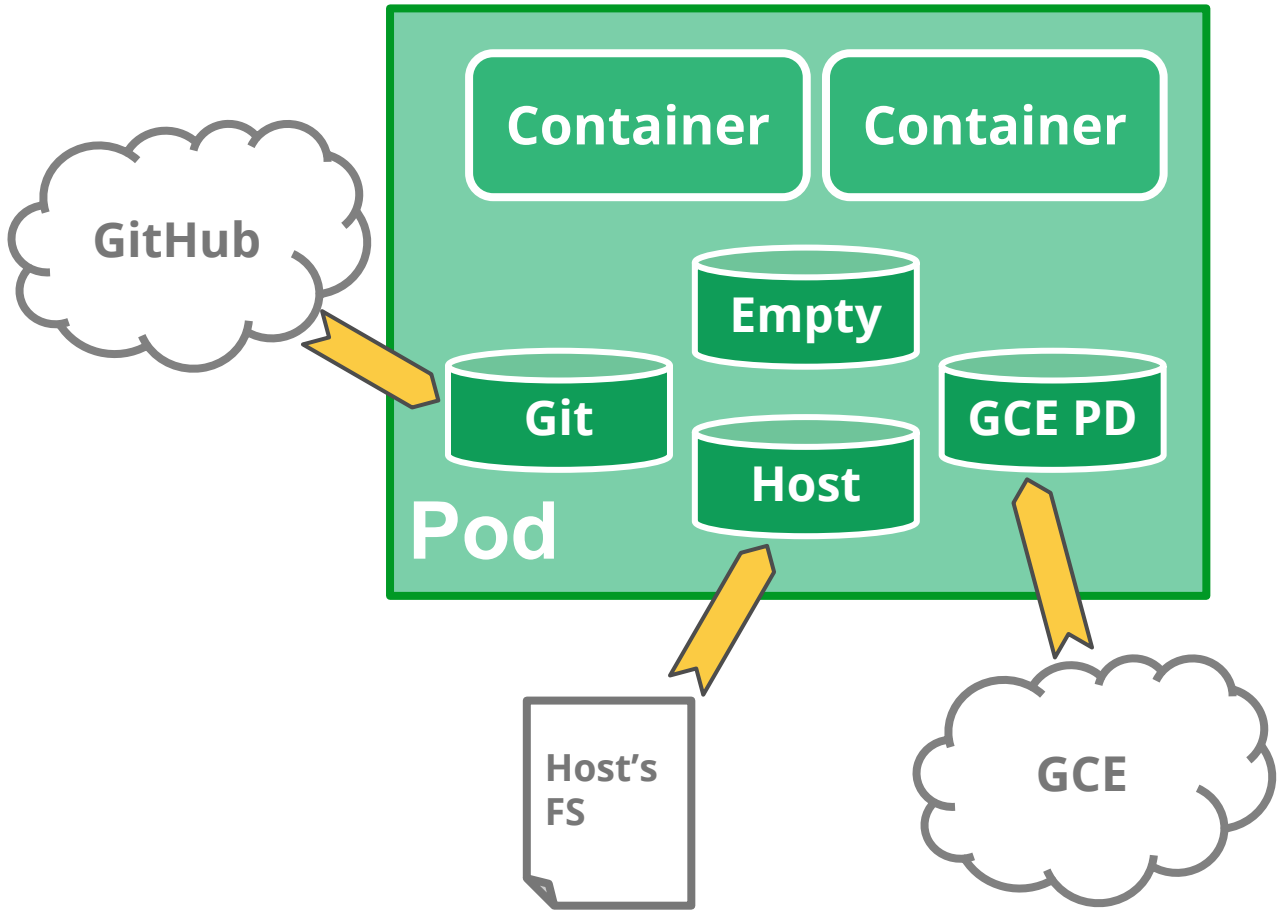
Volumes

Pod scoped

Share pod's lifetime & fate

Support various types of volumes

- Empty directory (default)
- Host file/directory
- Git repository
- GCE Persistent Disk
- ...more to come, suggestions welcome



Replication Controllers

Canonical example of control loops

Runs out-of-process wrt API server

Have 1 job: ensure N copies of a pod

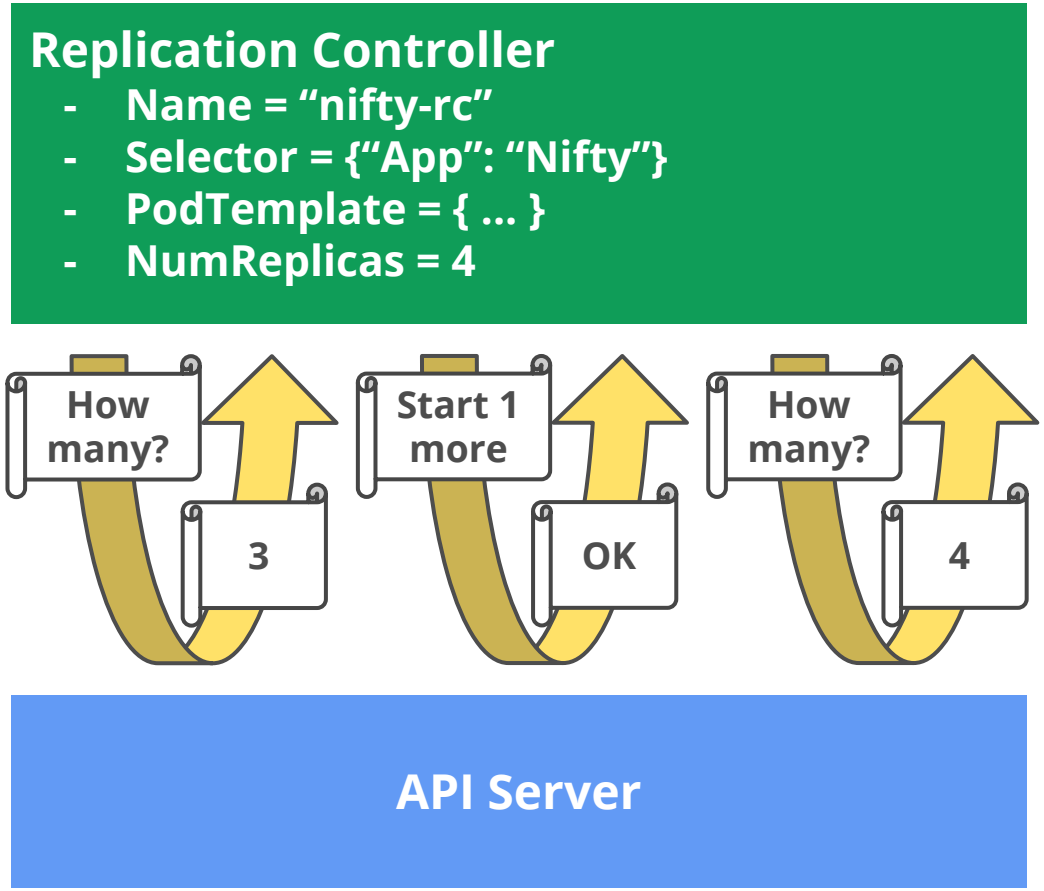
- if too few, start new ones
- if too many, kill some
- group == selector

Cleanly layered on top of the core

- all access is by public APIs

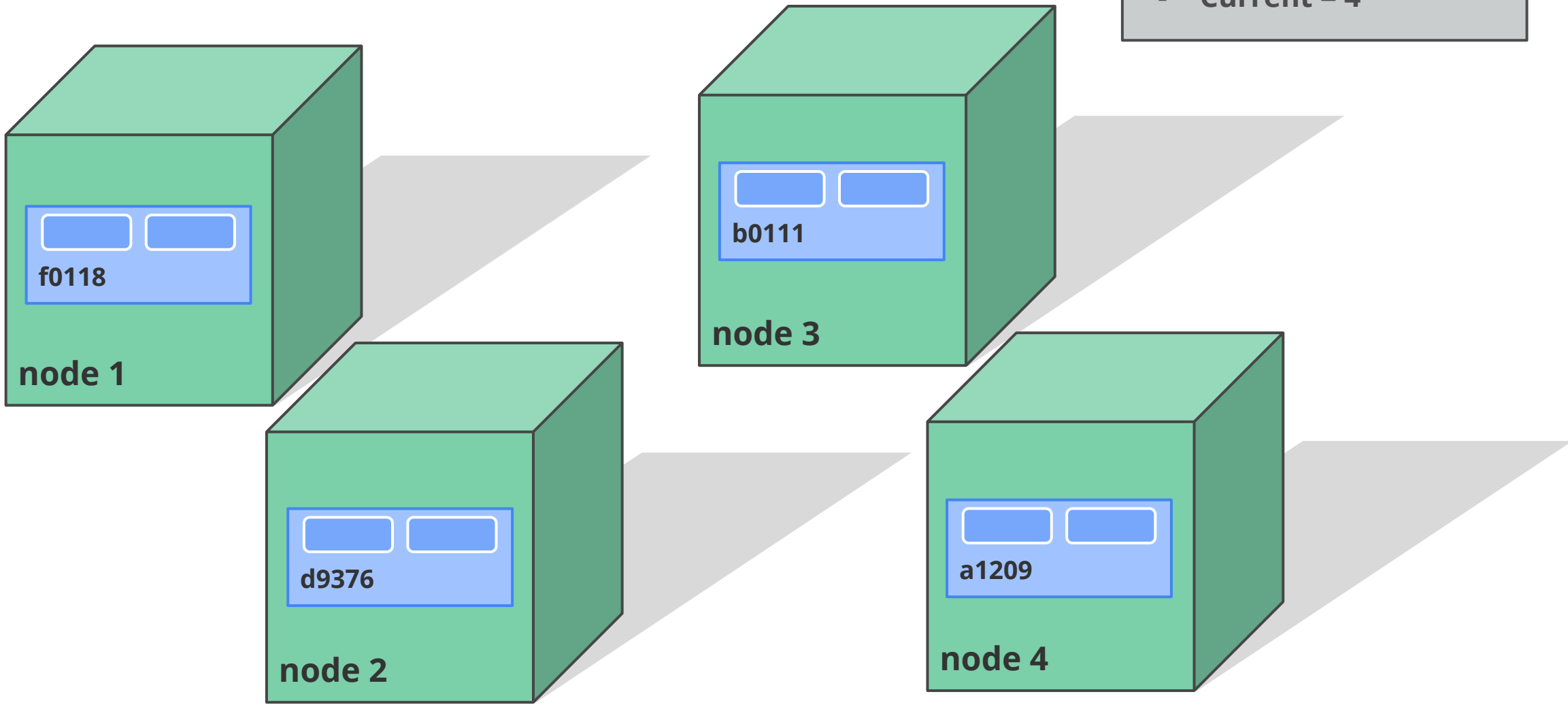
No ordinality or nominality

- replicated pods are fungible



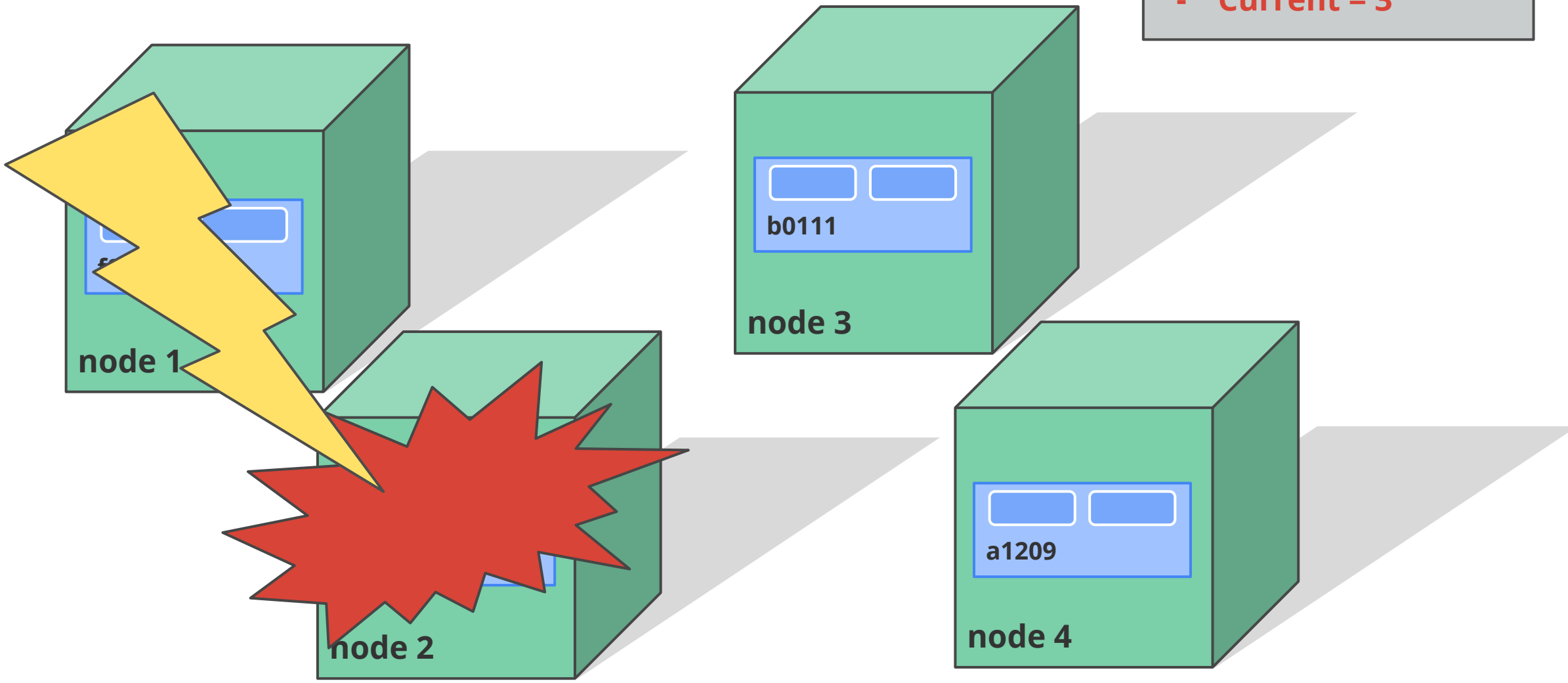
Replication Controllers

Replication Controller
- **Desired = 4**
- **Current = 4**



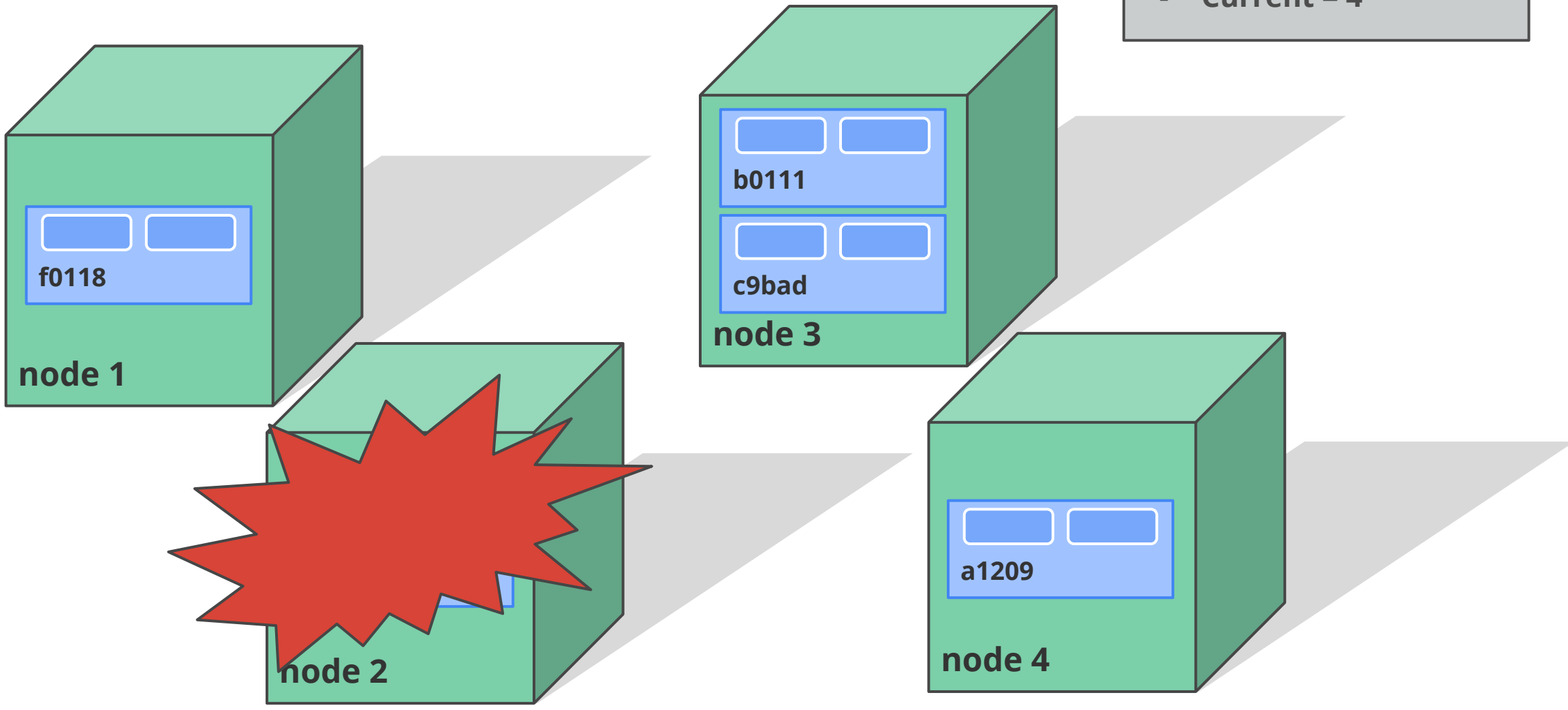
Replication Controllers

Replication Controller
- Desired = 4
- Current = 3



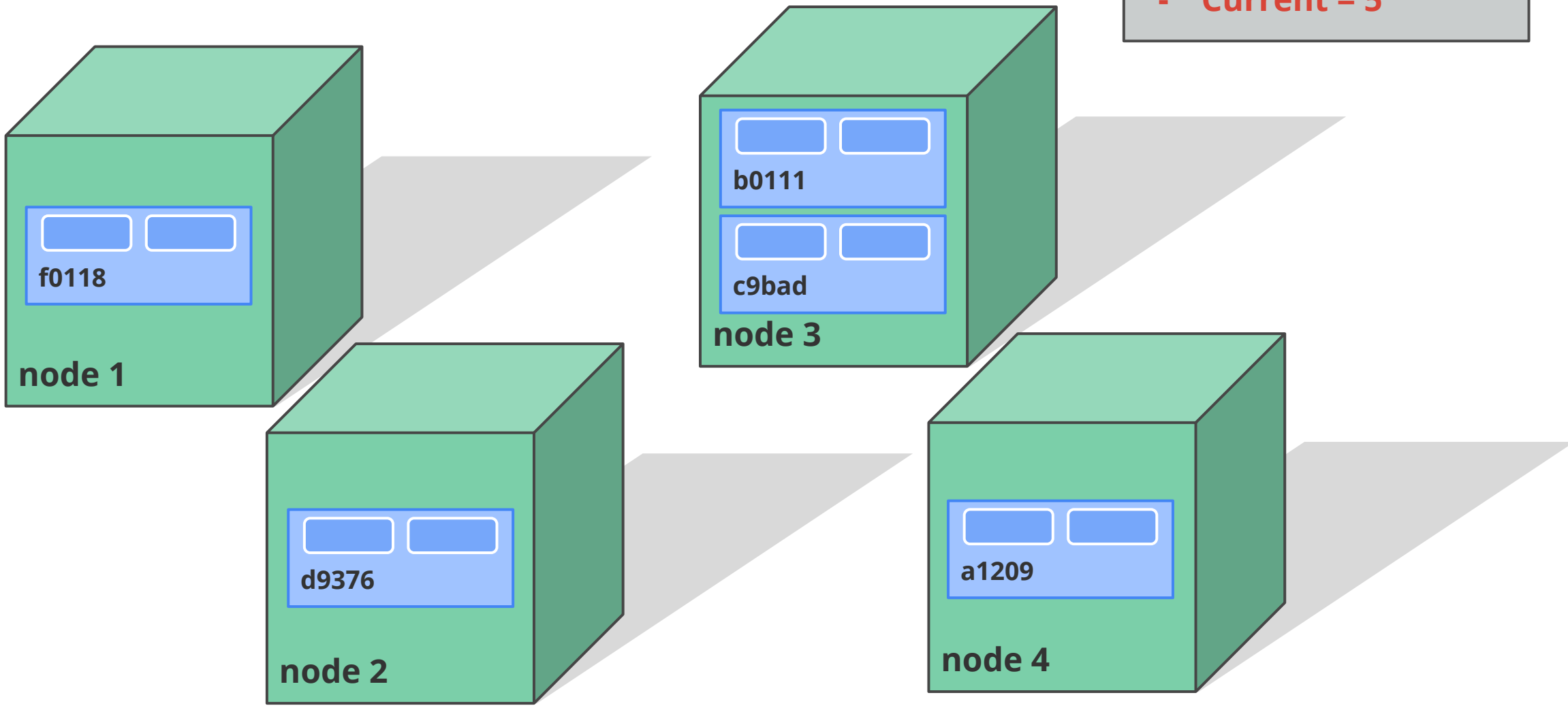
Replication Controllers

Replication Controller
- Desired = 4
- Current = 4



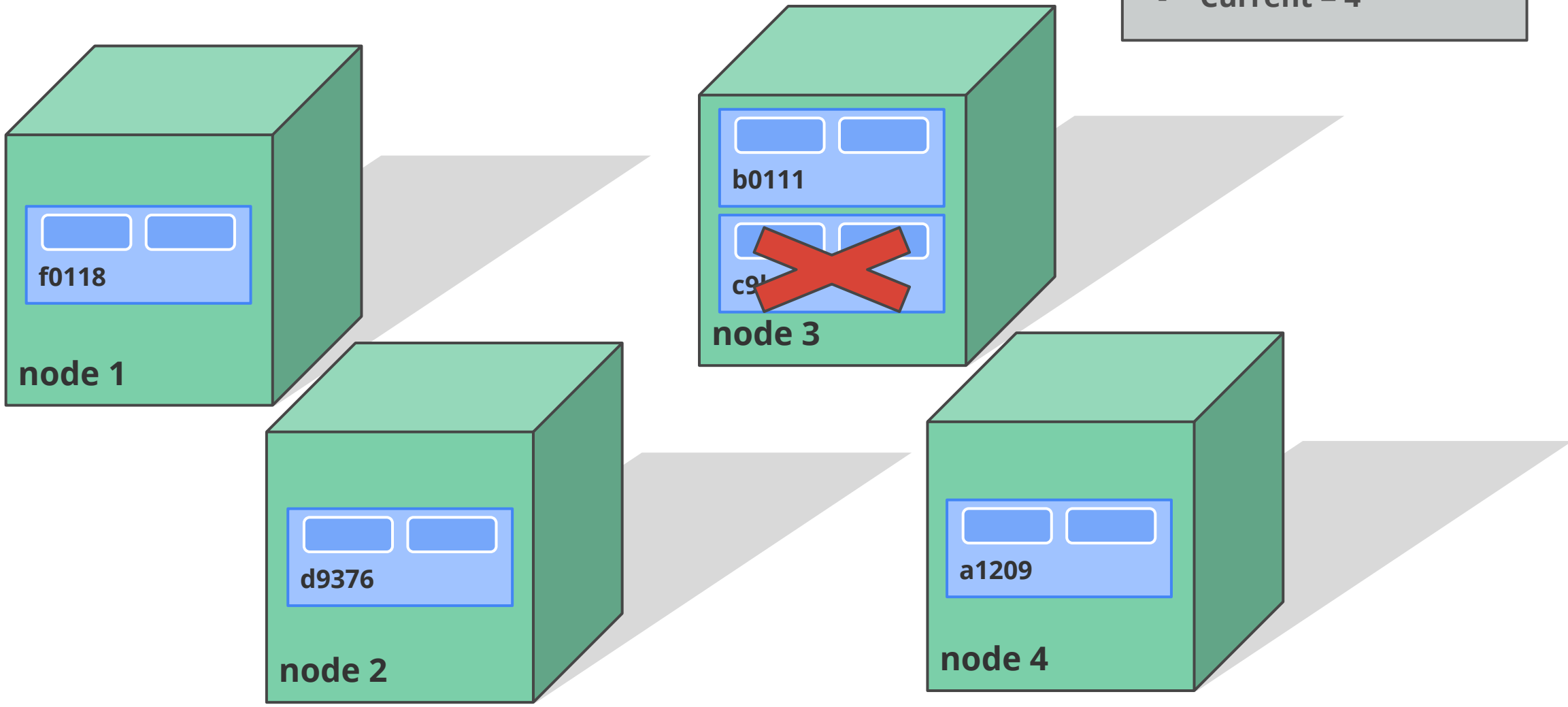
Replication Controllers

Replication Controller
- Desired = 4
- Current = 5



Replication Controllers

Replication Controller
- Desired = 4
- Current = 4



Pod Lifecycle

Pods are ephemeral

Can be *pending*, *running*, *succeeded*, or *failed*

- pod restart policy means *failed* is **really** the end
- no complex state machine logic

Failed pods are **not rescheduled** by the scheduler or apiserver

- even if a node dies
- ReplicationController is responsible for this
- keeps the scheduler **simple**

Pod-to-pod communication should consider these rules

- Services hide this all
- Makes pod-to-pod communication more formal

Services

A group of pods that **act as one**

- group == selector

Defines access policy

- only “load balanced” for now

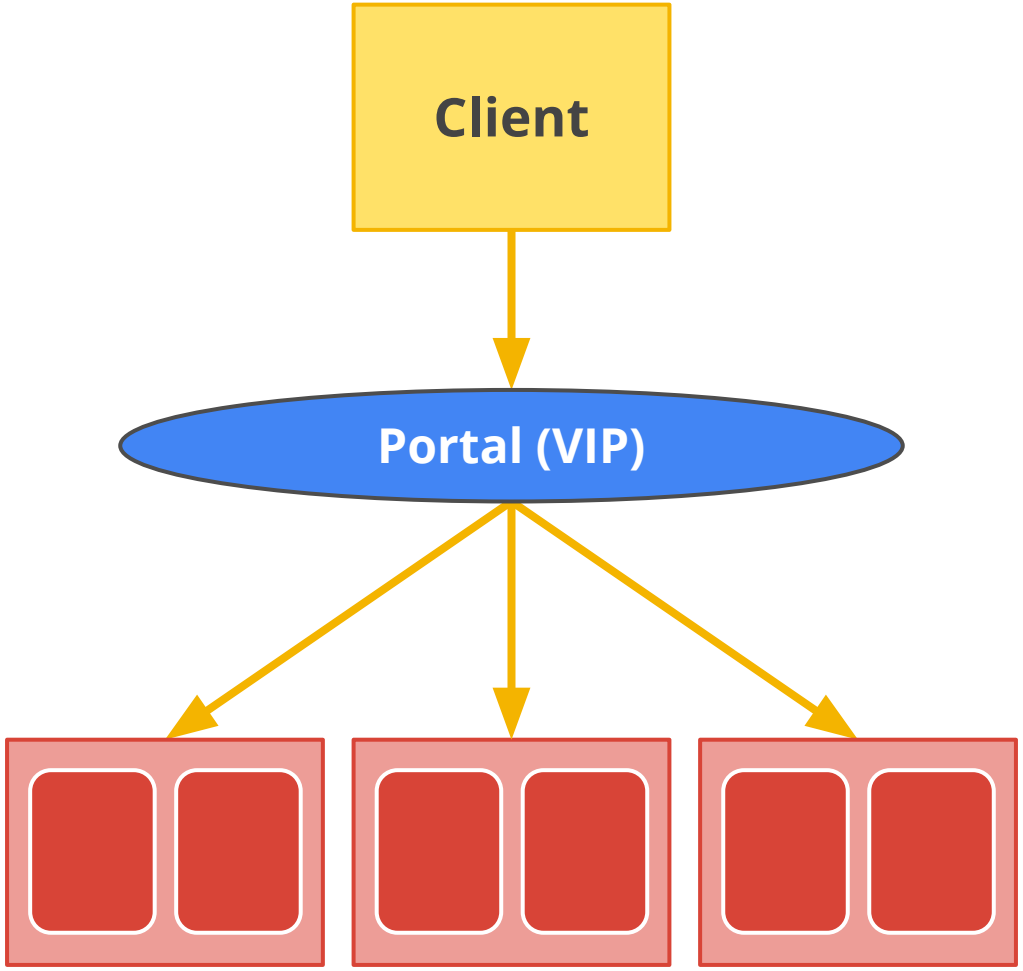
Gets a **stable** virtual IP and port

- called the service *portal*
- soon to have DNS

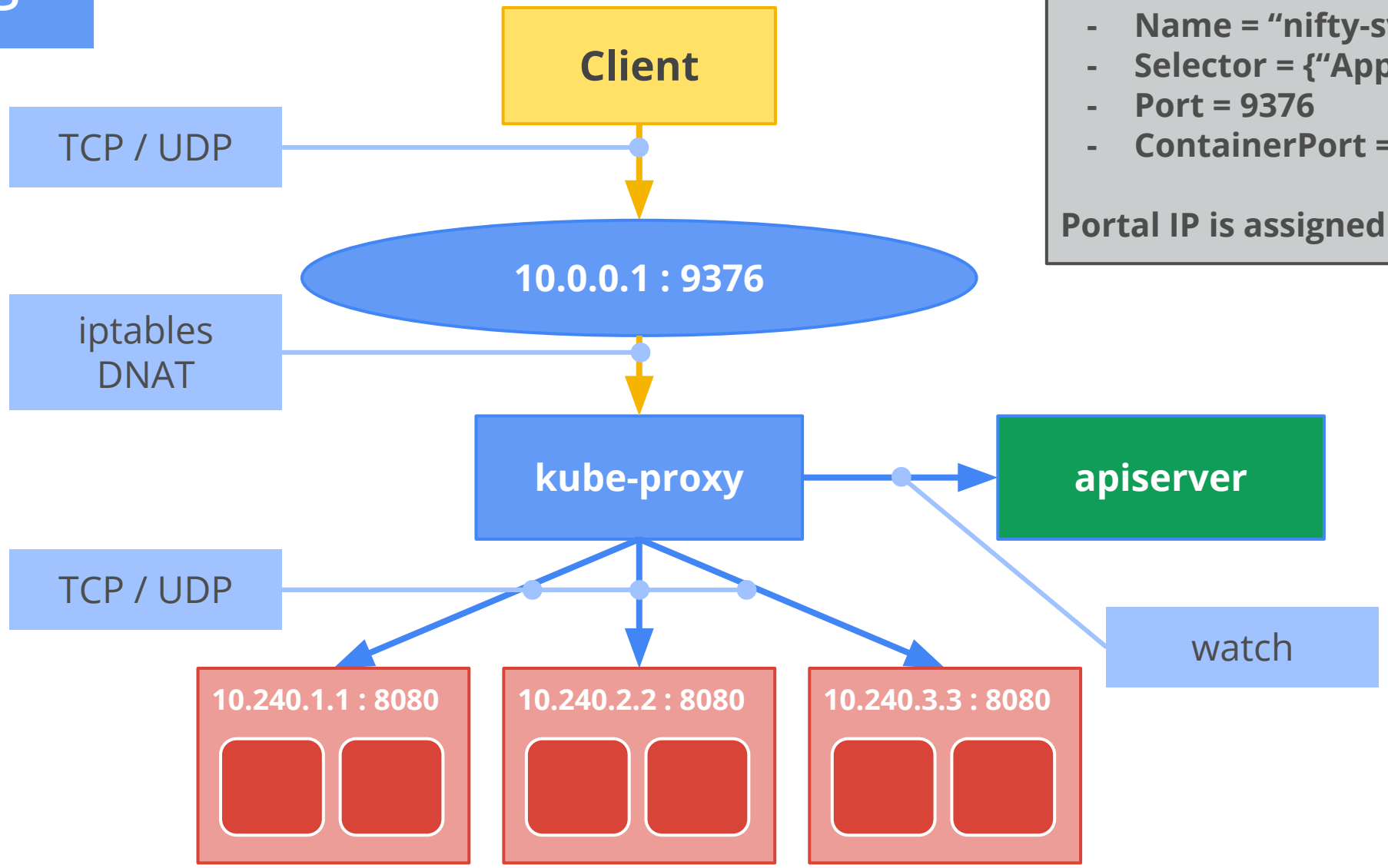
VIP is captured by *kube-proxy*

- watches the service **constituency**
- updates when backends change

Hide complexity - ideal for non-native apps



Services



Service
- **Name = “nifty-svc”**
- **Selector = {“App”: “Nifty”}**
- **Port = 9376**
- **ContainerPort = 8080**

Portal IP is assigned

News & Plans

Google just launched **Google Container Engine** (GKE)

- hosted Kubernetes
- https://cloud.google.com/container-engine/

Roadmap:

- https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/roadmap.md

Driving towards a 1.0 release in O(months)

Questions?

http://kubernetes.io

Images by Connie Zhou