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**.

Kubernetes

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

- Container orchestration
- Runs Docker containers
- Supports multiple cloud and bare-metal environments
- Inspired and informed by Google's experiences and internal systems
- **Open source**, written in **Go**

Manage applications, not machines

Design principles

Declarative > imperative: State your desired results, let the system actuate

Control loops: Observe, rectify, repeat

Simple > Complex: Try to do as little as possible

Modularity: Components, interfaces, & plugins

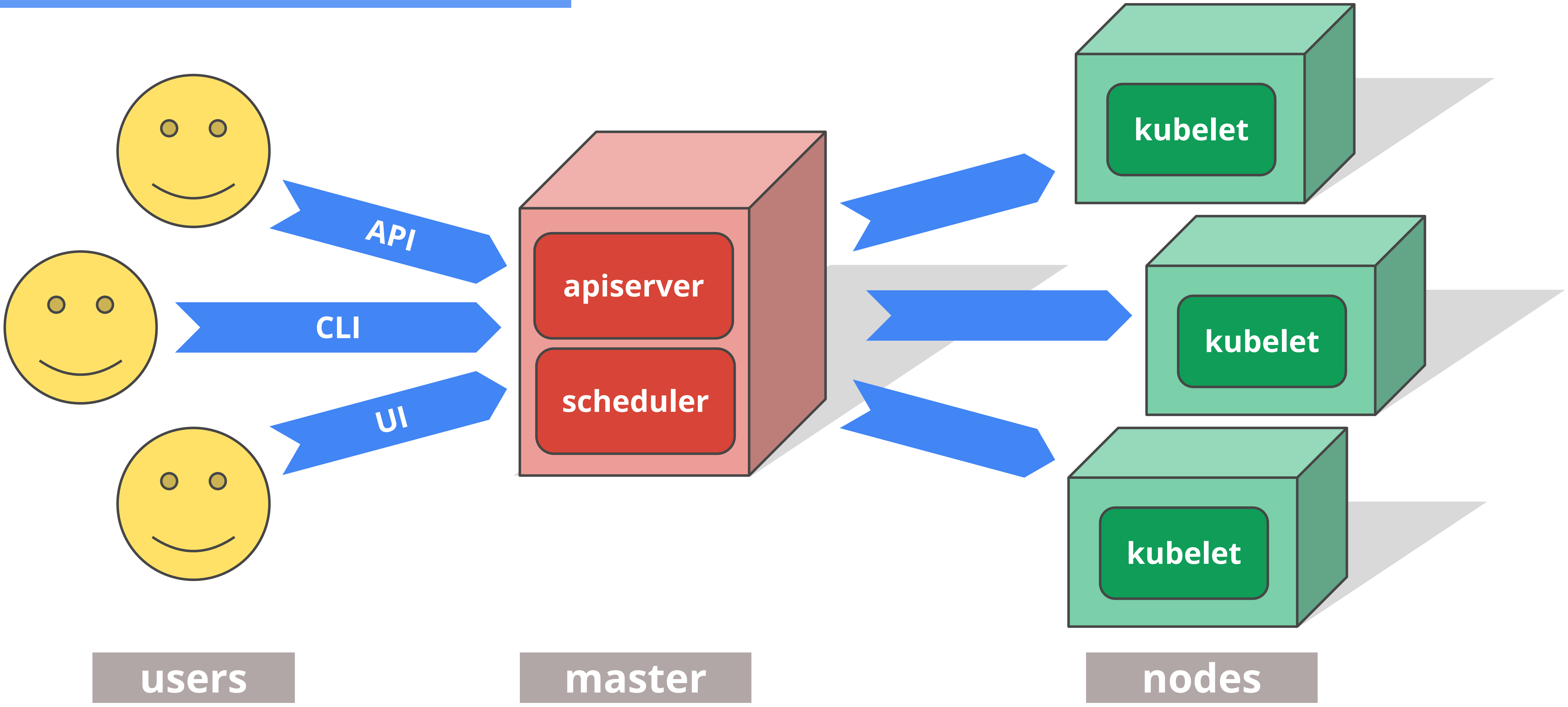
Legacy compatible: Requiring apps to change is a <u>non-starter</u>

No grouping: Labels are the <u>only</u> groups

Cattle > Pets: Manage your workload in bulk

Open > Closed: Open Source, standards, REST, JSON, etc.

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 current state towards desired state

example: replication controller

Service: A set of running pods that work together

example: load-balanced backends

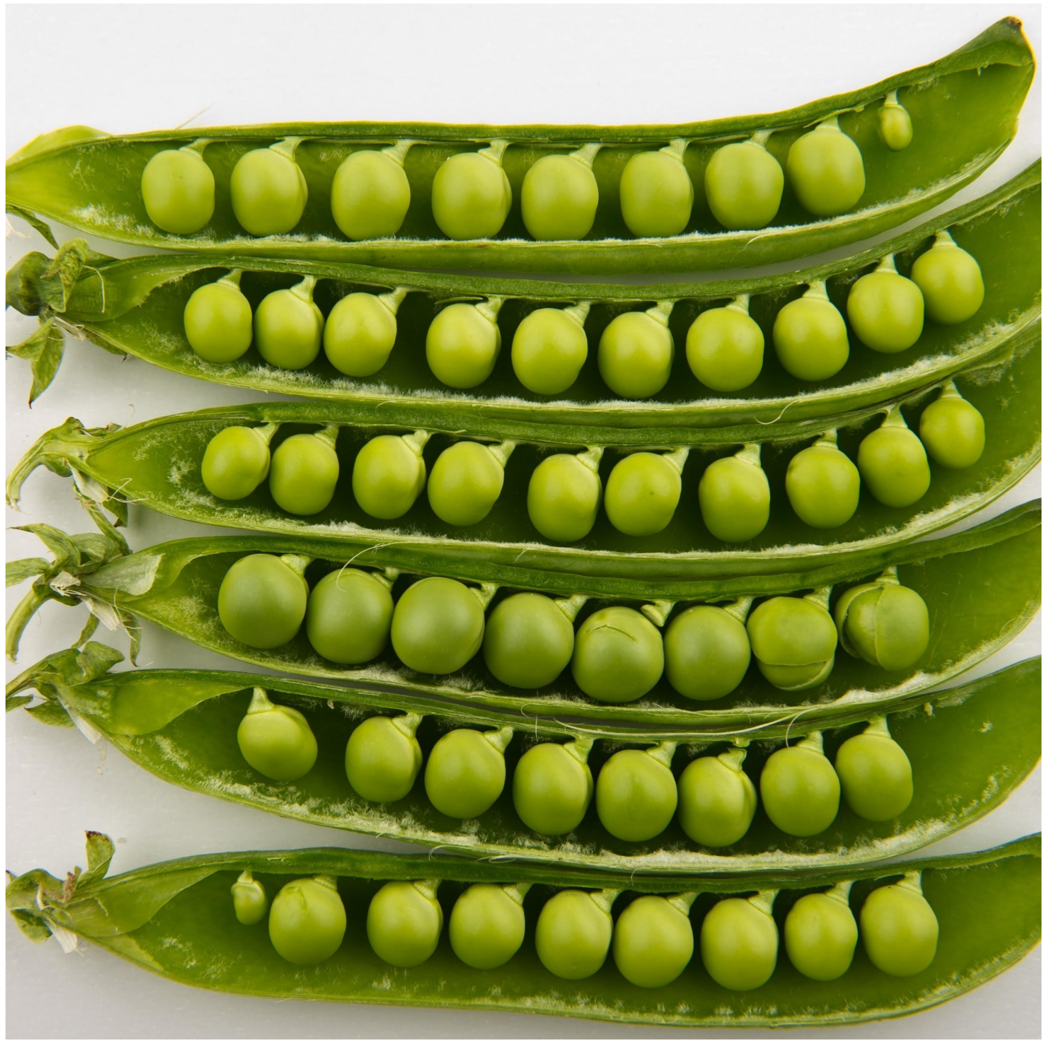
Labels: Identifying metadata attached to other objects

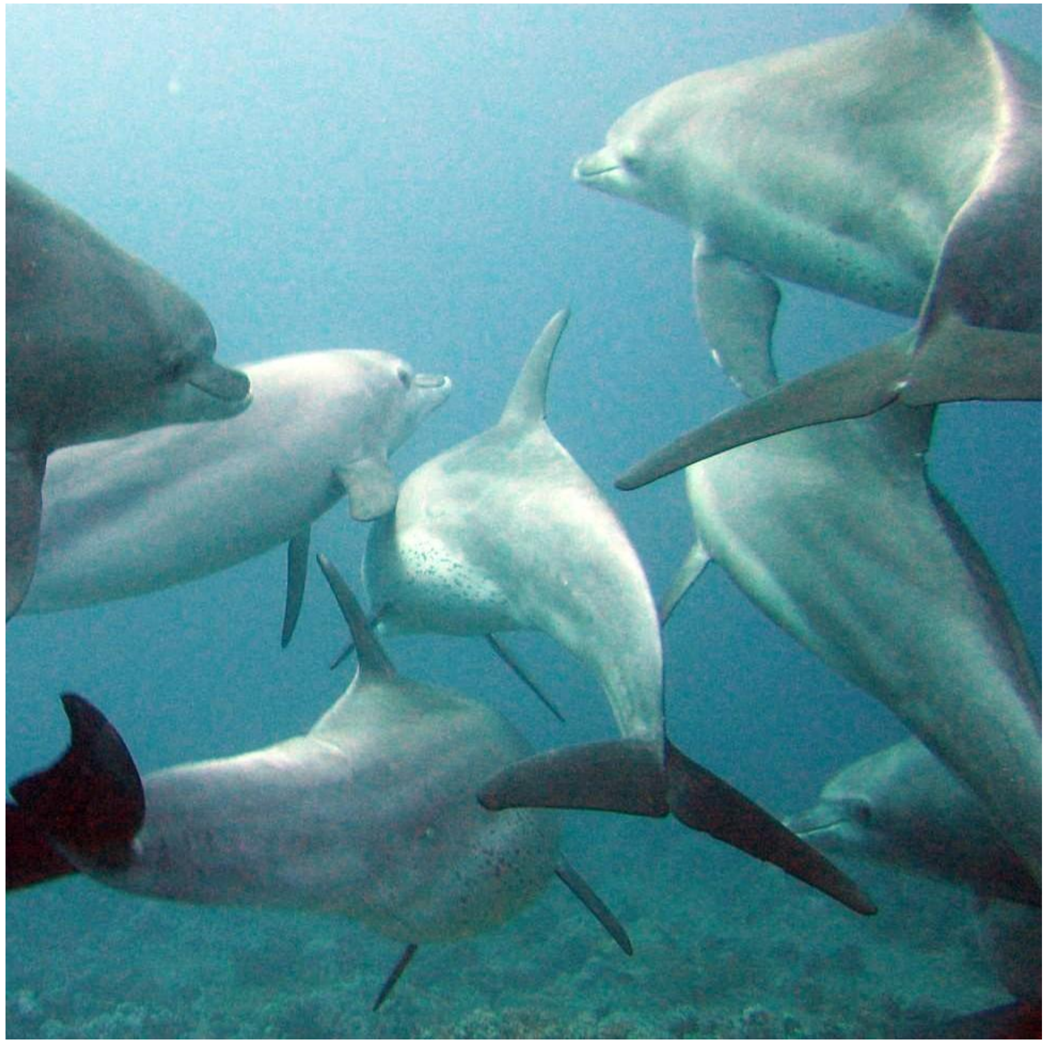
example: phase=canary vs. phase=prod

Selector: A query against labels, producing a set result

example: all pods where label phase == prod

Pods





Pods

Small group of containers & volumes

Tightly coupled

The atom of cluster scheduling & placement

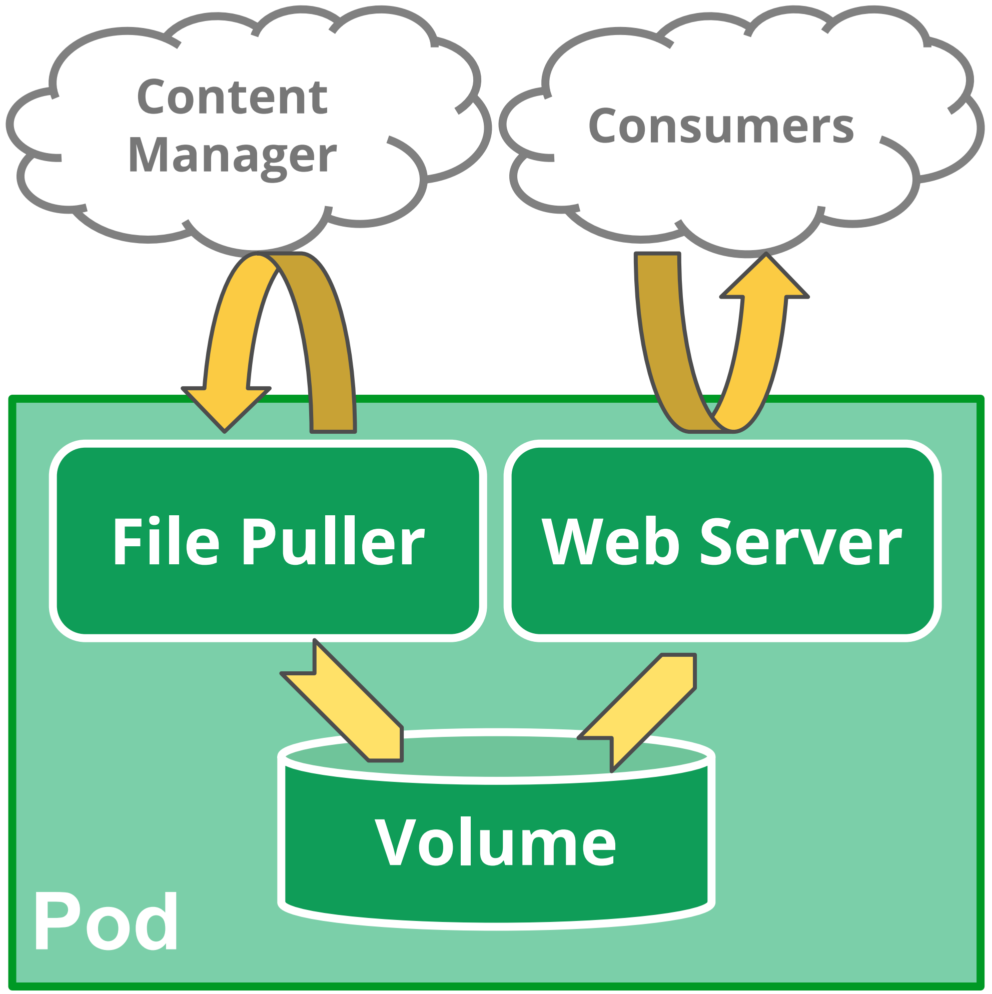
Shared namespace

- **share IP** address & localhost

Ephemeral

- can die and be replaced

Example: data puller & web server



Pod lifecycle

Once scheduled to a node, pods do not move

- restart policy means restart **in-place**

Pods can be observed *pending*, *running*, *succeeded*, or *failed*

- *failed* is **really** the end - no more restarts
- no complex state machine logic

Pods are **not rescheduled** by the scheduler or apiserver

- even if a node dies
- controllers are responsible for this
- keeps the scheduler **simple**

Apps should consider these rules

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

Labels

Arbitrary metadata

Attached to any API object

Generally represent **identity**

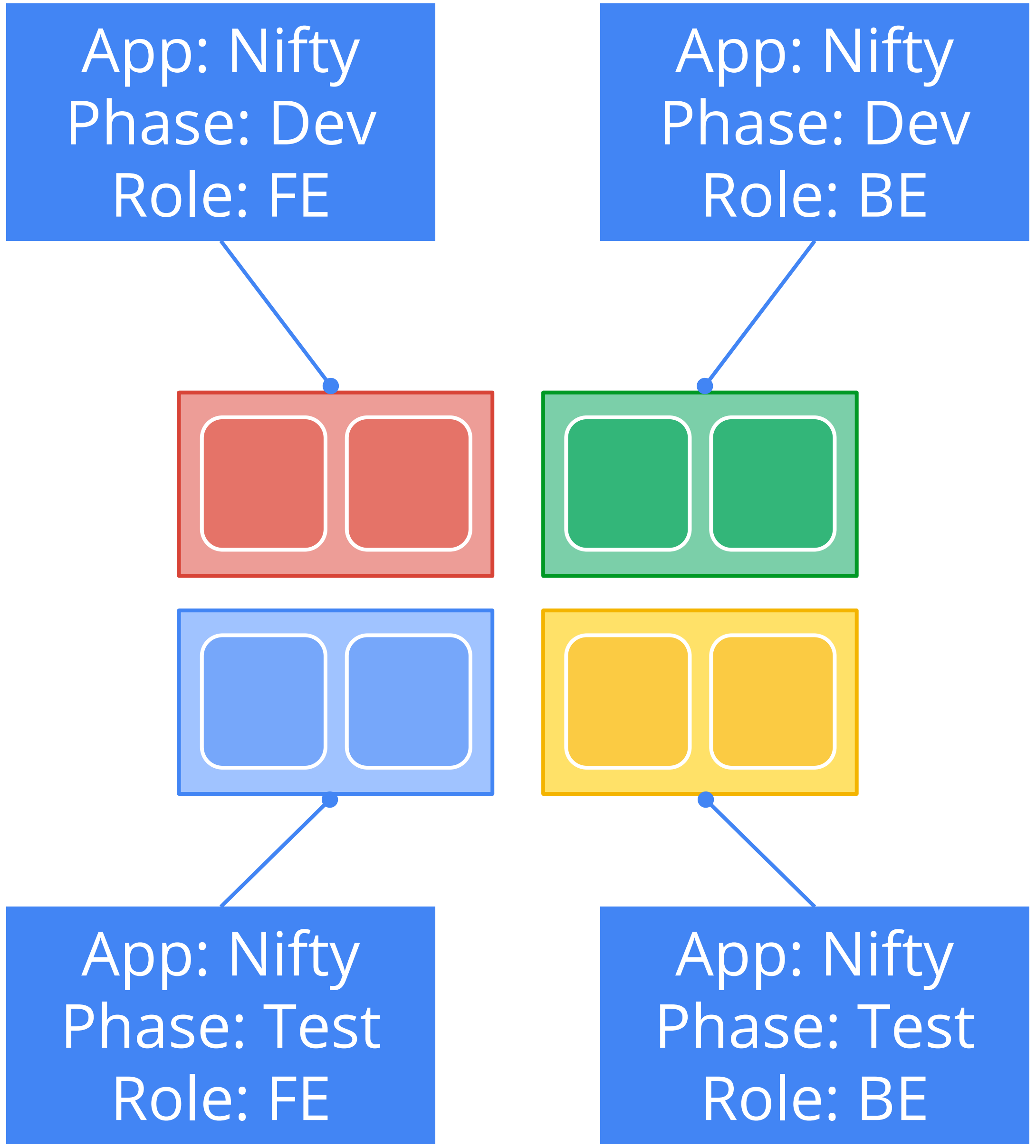
Queryable by **selectors**

- think SQL '*select ... where ...*'

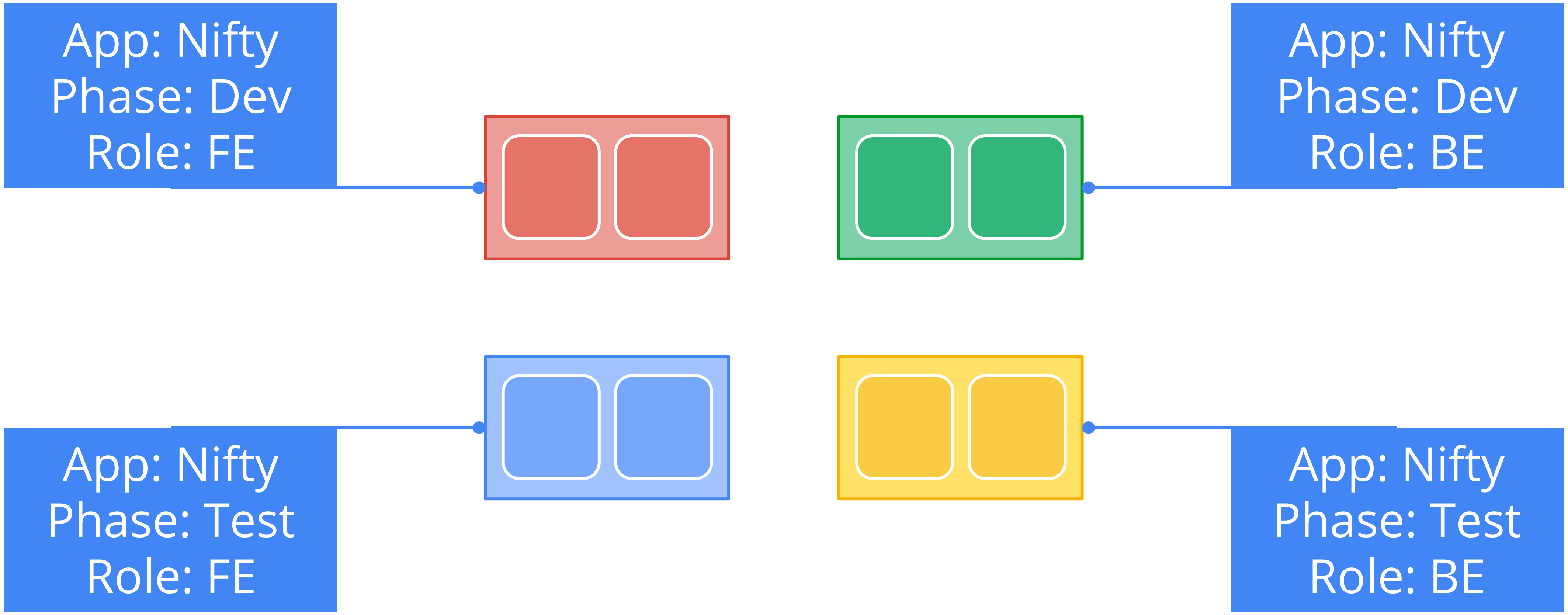
The **only** grouping mechanism

- pods under a ReplicationController
- pods in a Service
- capabilities of a node (constraints)

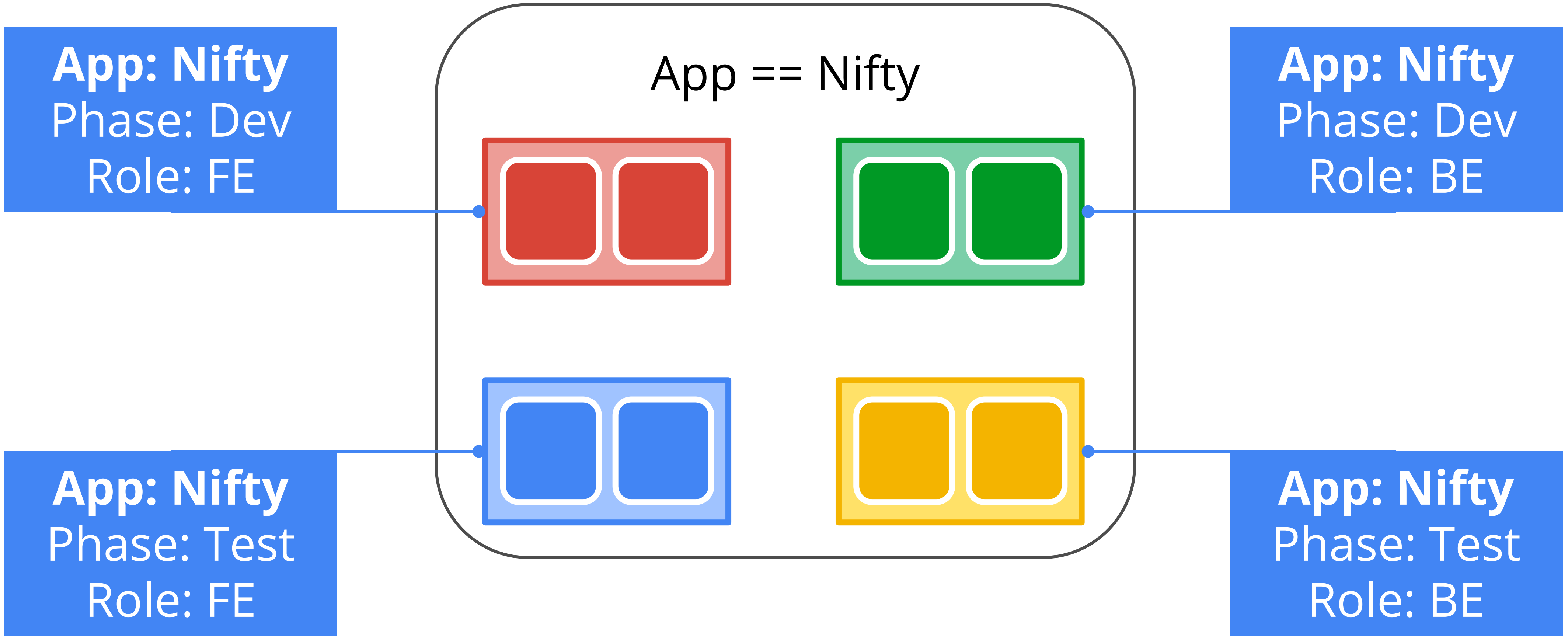
Example: "phase: canary"



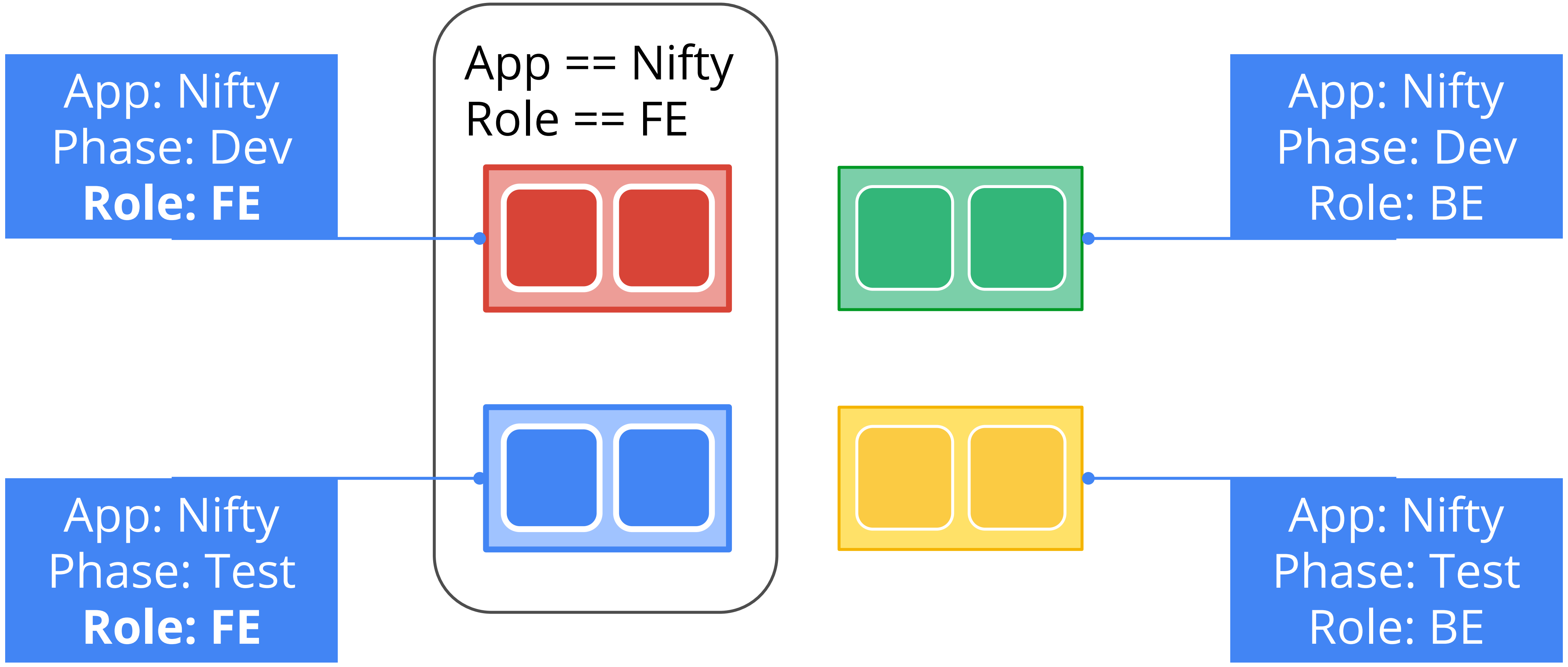
Selectors


App: Nifty
Phase: Dev
Role: FE
App: Nifty
Phase: Dev
Role: BE
App: Nifty
Phase: Test
Role: FE
App: Nifty
Phase: Test
Role: BE

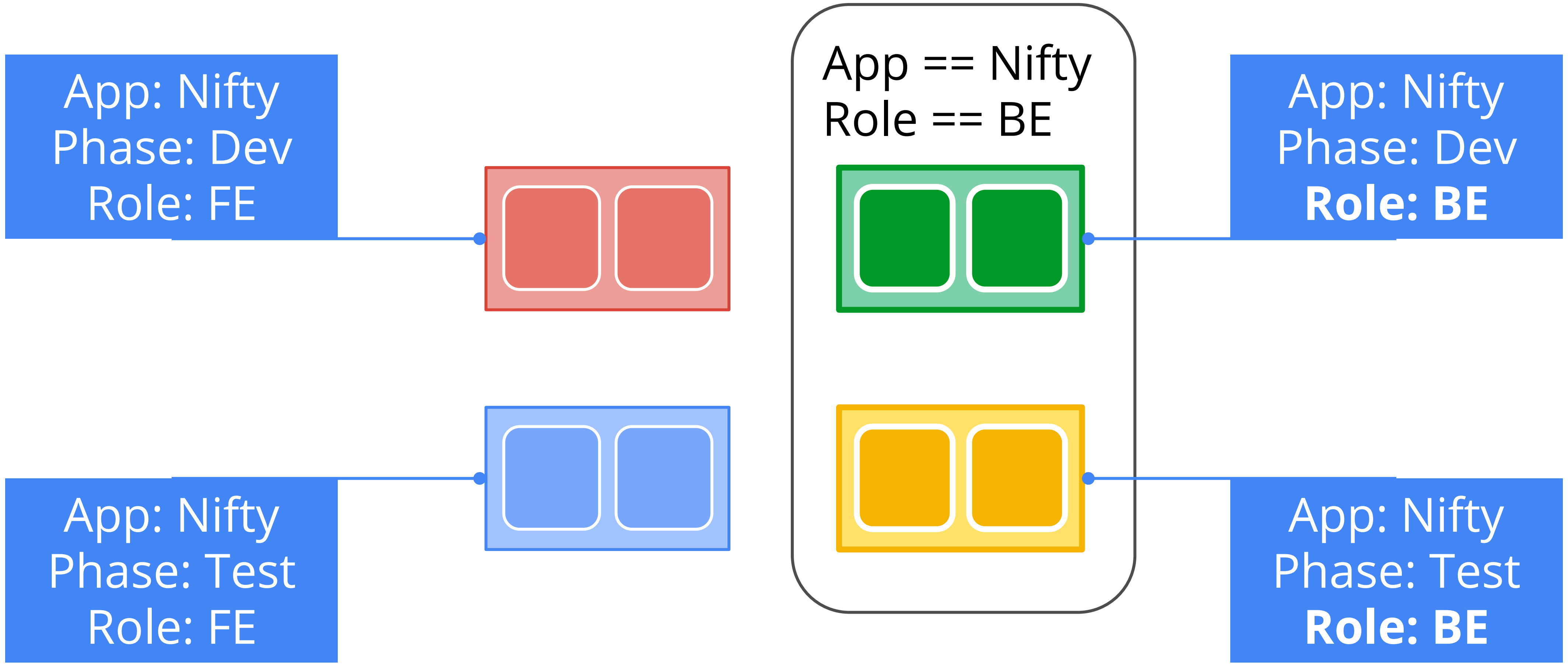
Selectors


App: Nifty
Phase: Dev
Role: FE
App == Nifty
App: Nifty
Phase: Dev
Role: BE
App: Nifty
Phase: Test
Role: FE
App: Nifty
Phase: Test
Role: BE

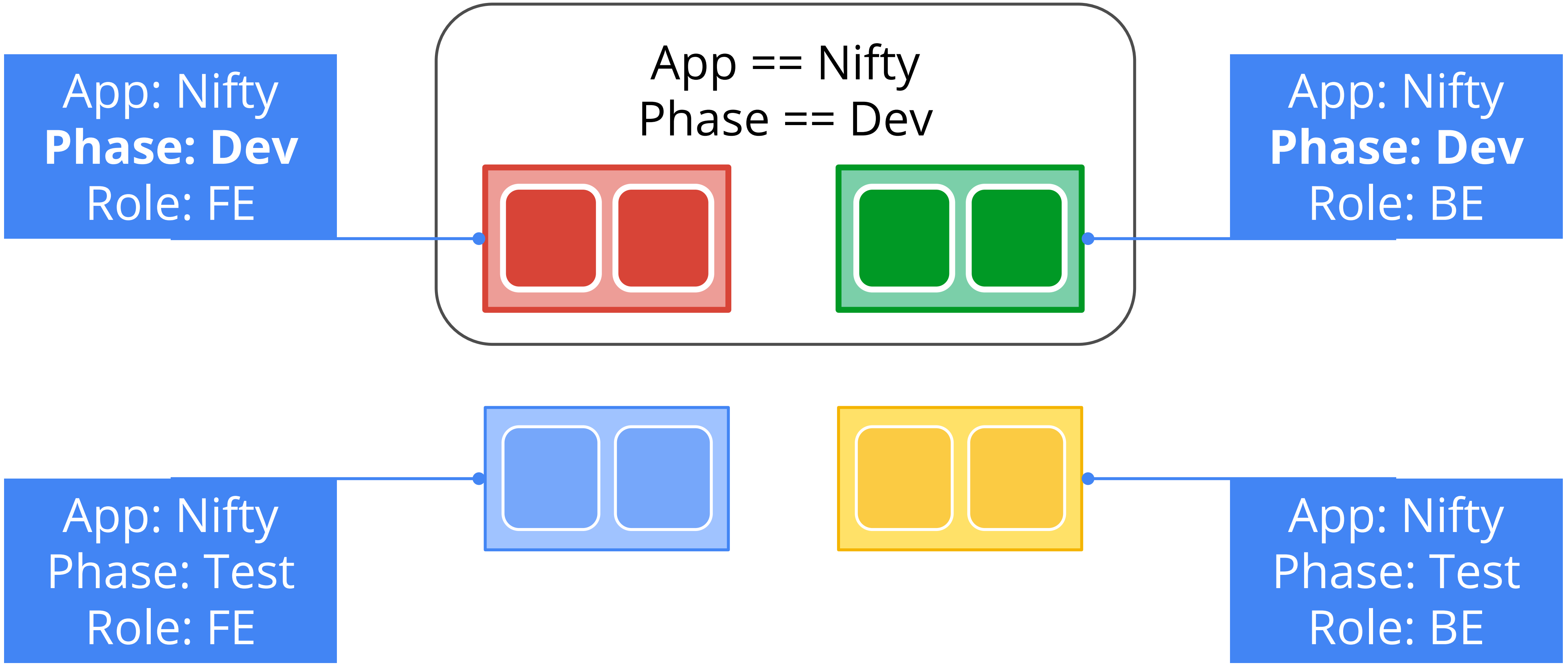
Selectors


App: Nifty
Phase: Dev
Role: FE
App == Nifty
Role == FE
App: Nifty
Phase: Dev
Role: BE
App: Nifty
Phase: Test
Role: FE
App: Nifty
Phase: Test
Role: BE

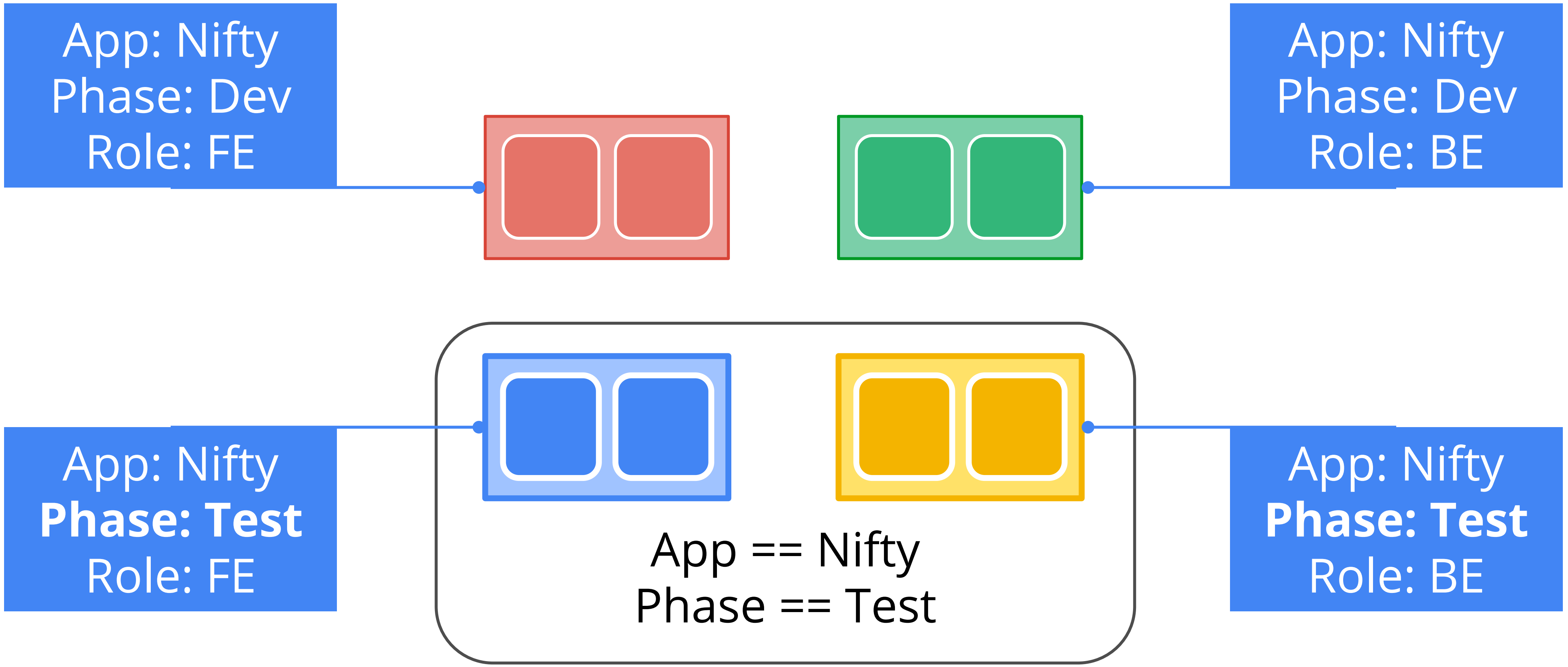
Selectors


App: Nifty
Phase: Dev
Role: FE
App == Nifty
Role == BE
App: Nifty
Phase: Dev
Role: BE
App: Nifty
Phase: Test
Role: FE
App: Nifty
Phase: Test
Role: BE

Selectors


App == Nifty
Phase == Dev
App: Nifty
Phase: Dev
Role: FE
App: Nifty
Phase: Dev
Role: BE
App: Nifty
Phase: Test
Role: FE
App: Nifty
Phase: Test
Role: BE

Selectors


App: Nifty
Phase: Dev
Role: FE
App: Nifty
Phase: Dev
Role: BE
App: Nifty
Phase: Test
Role: FE
App: Nifty
Phase: Test
Role: BE
App == Nifty
Phase == Test

Replication Controllers

Canonical example of control loops

Runs out-of-process wrt API server

Have 1 job: ensure N copies of a pod

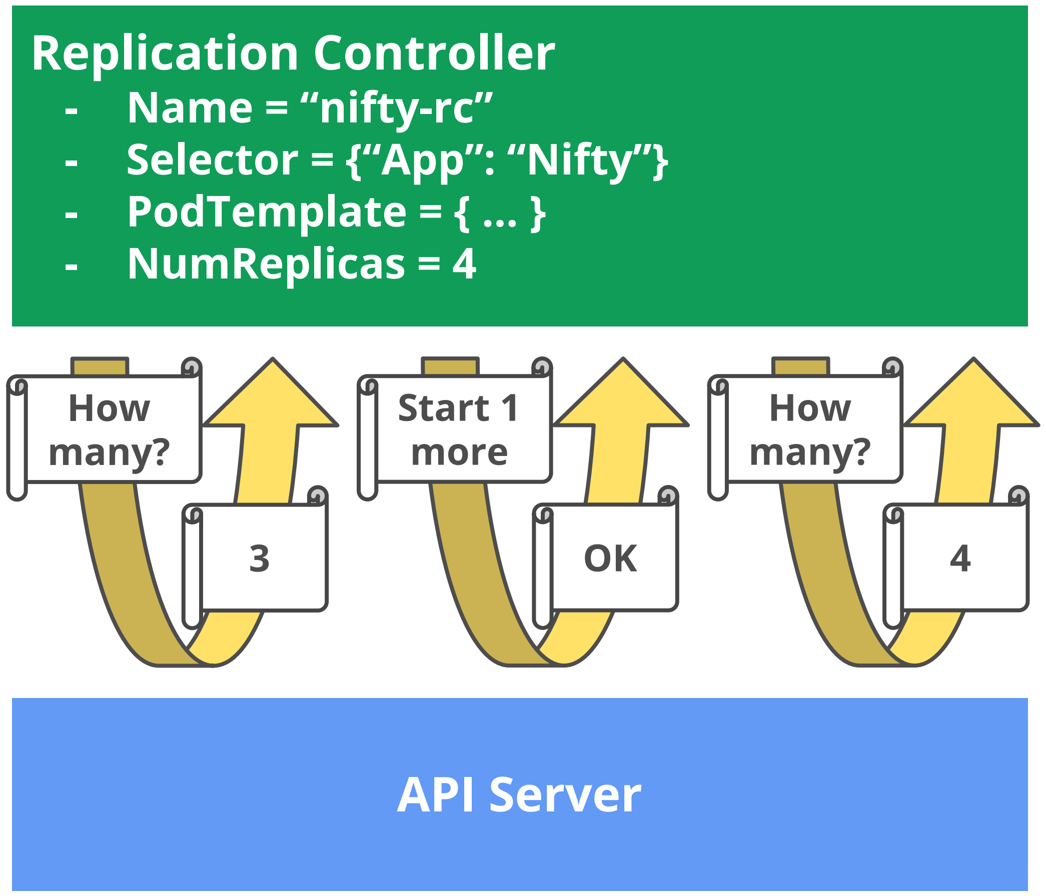
- if too few, start new ones
- if too many, kill some
- group == selector

Cleanly layered on top of the core

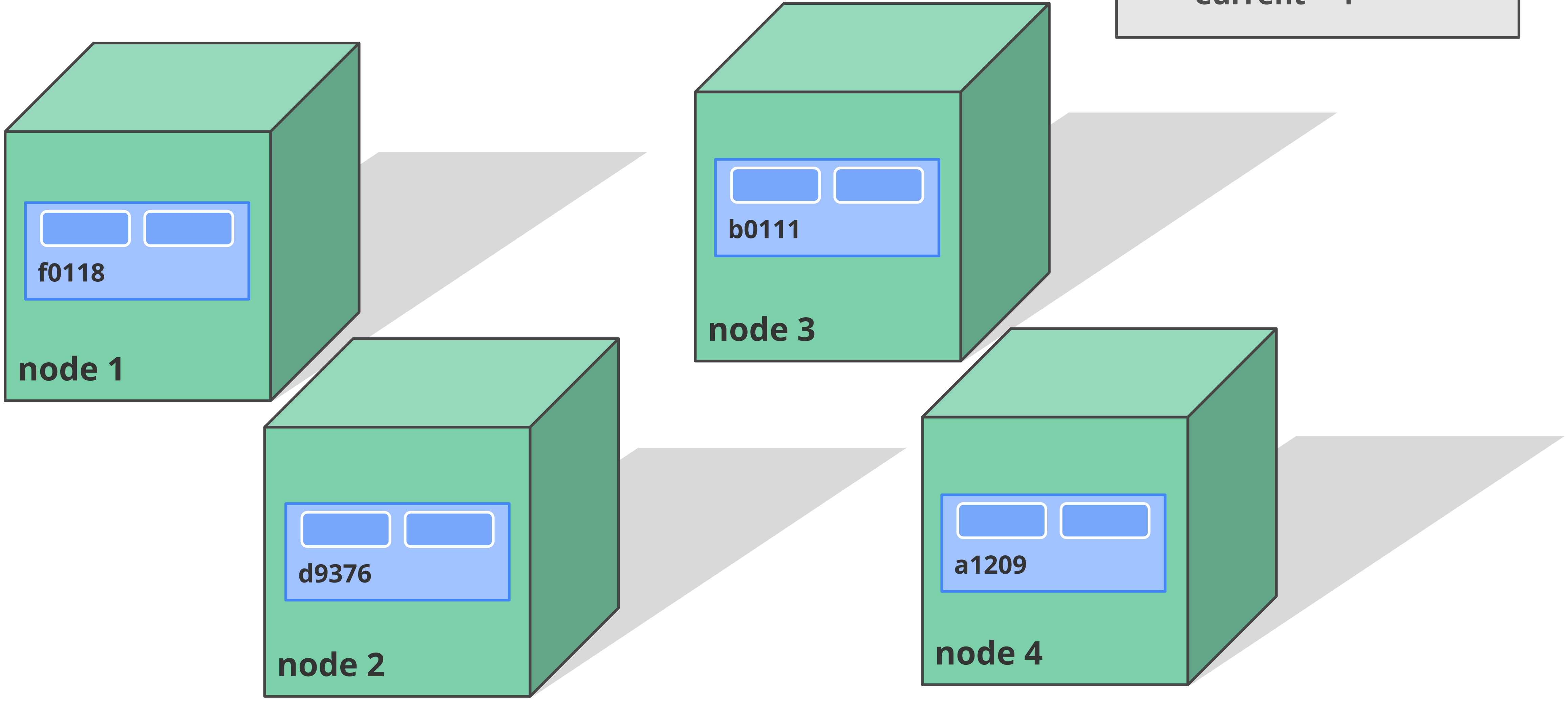
- all access is by public APIs

Replicated pods are fungible

- No implied ordinality or identity

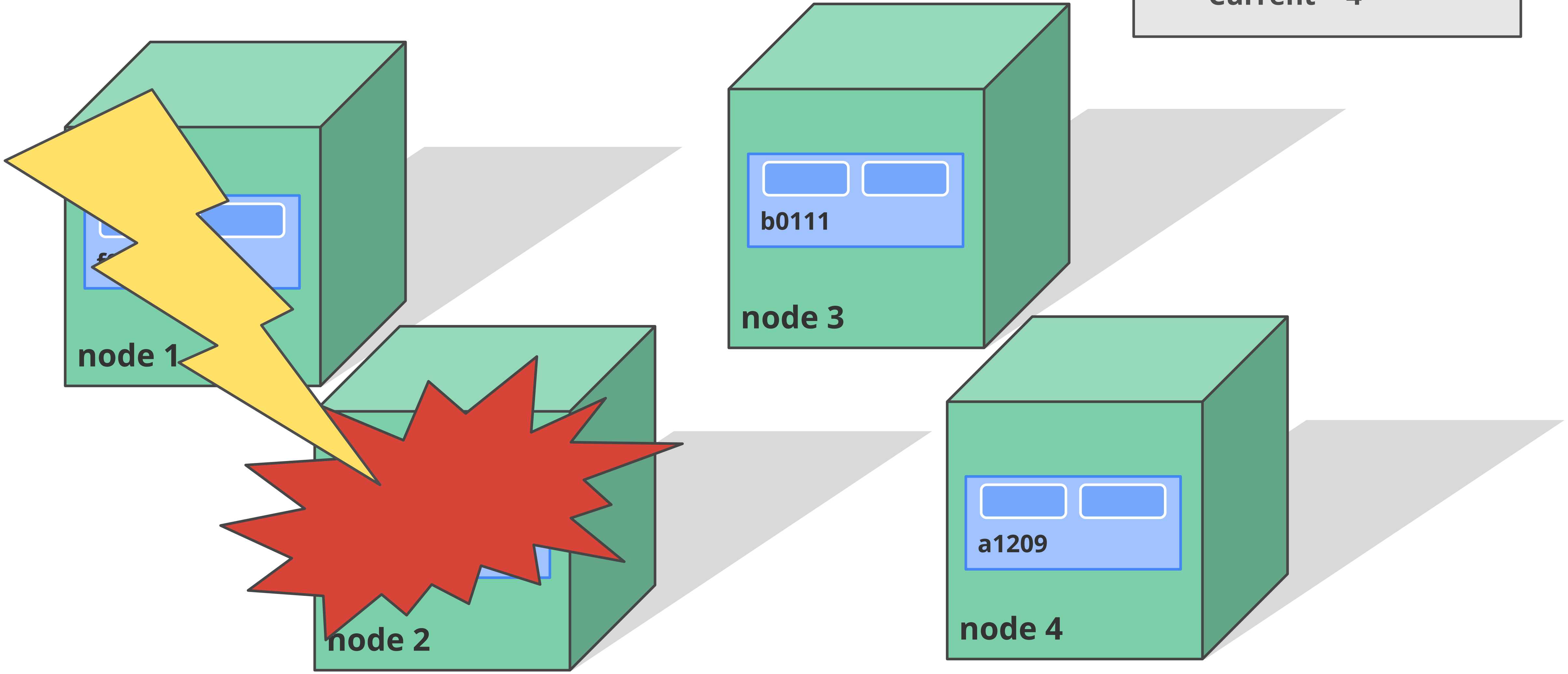


Replication Controllers



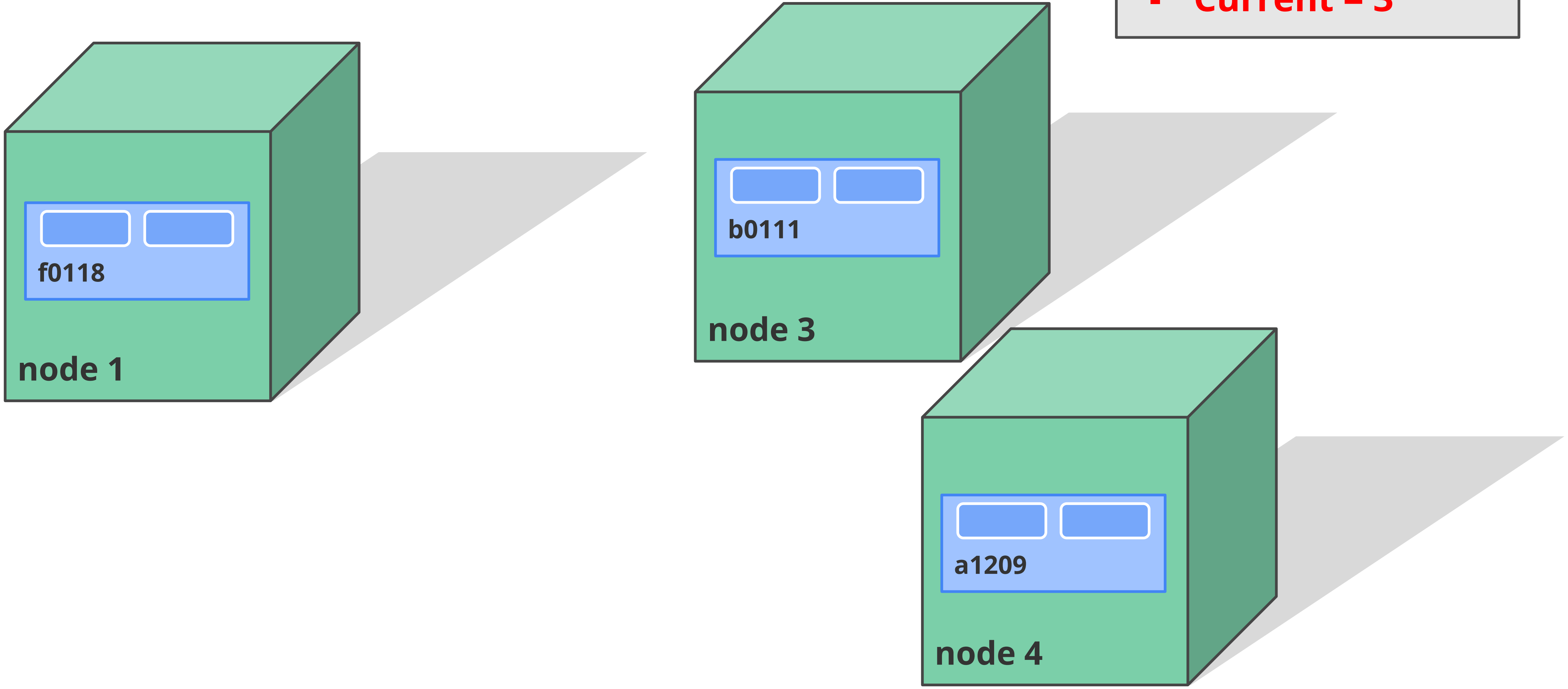
Replication Controller
- Desired = 4
- Current = 4

Replication Controllers



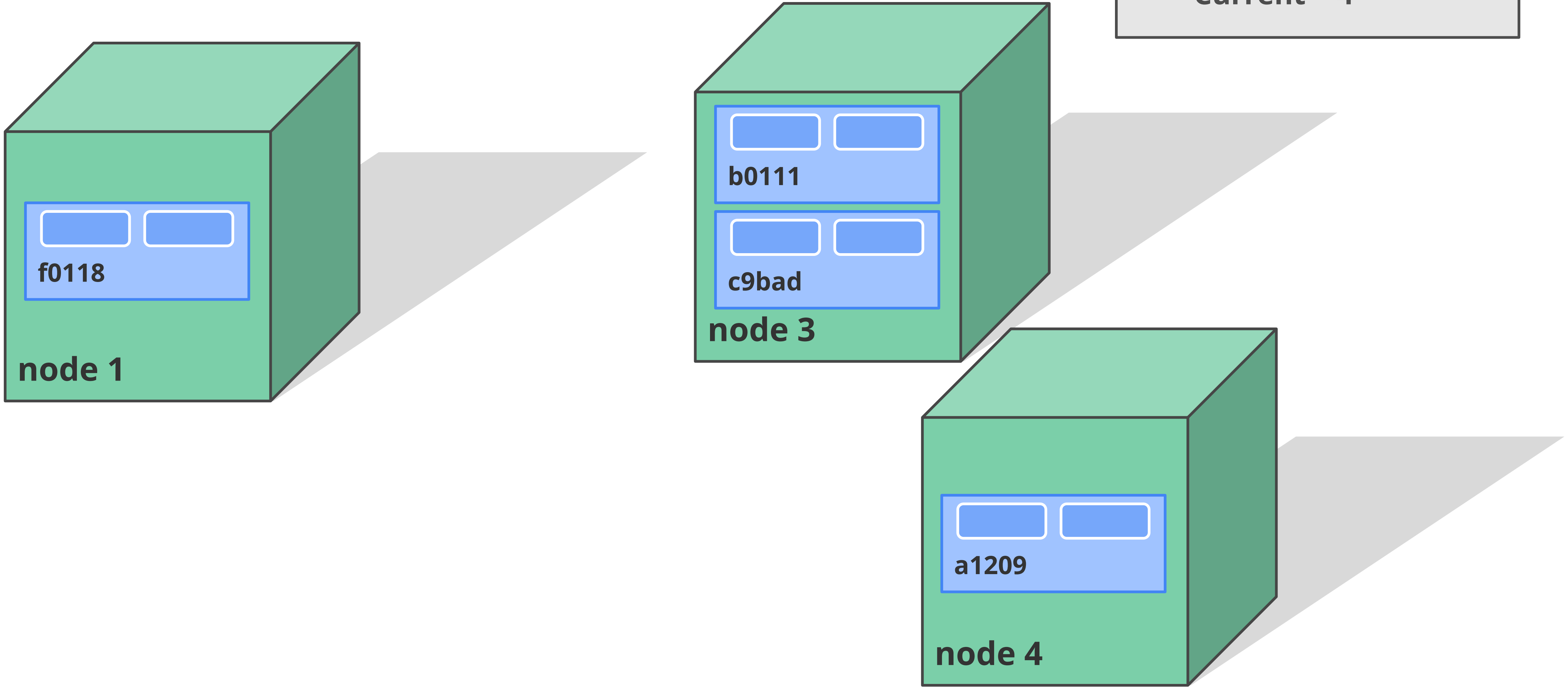
Replication Controller
- Desired = 4
- Current = 4

Replication Controllers



Replication Controller
- Desired = 4
- Current = 3

Replication Controllers

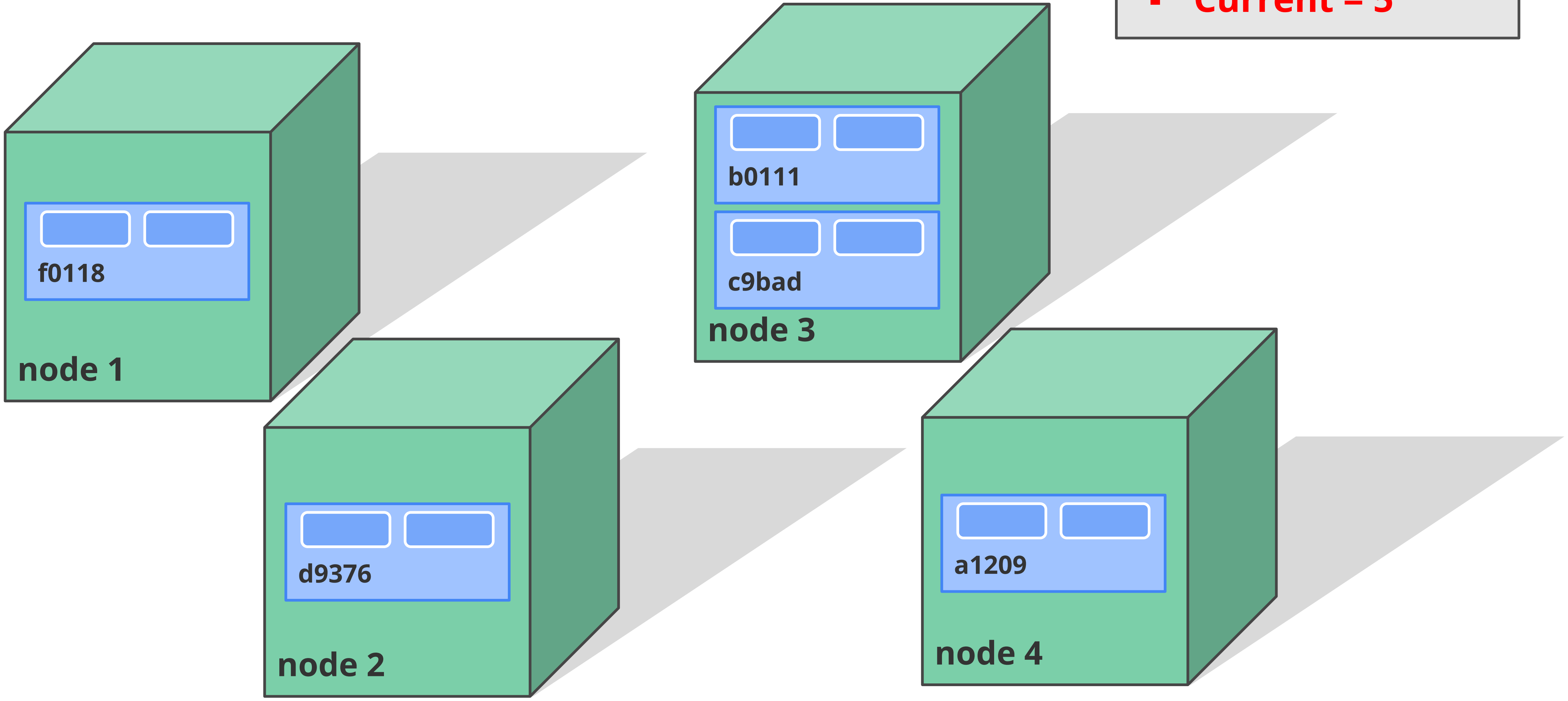


Replication Controller

- **Desired = 4**
- **Current = 4**

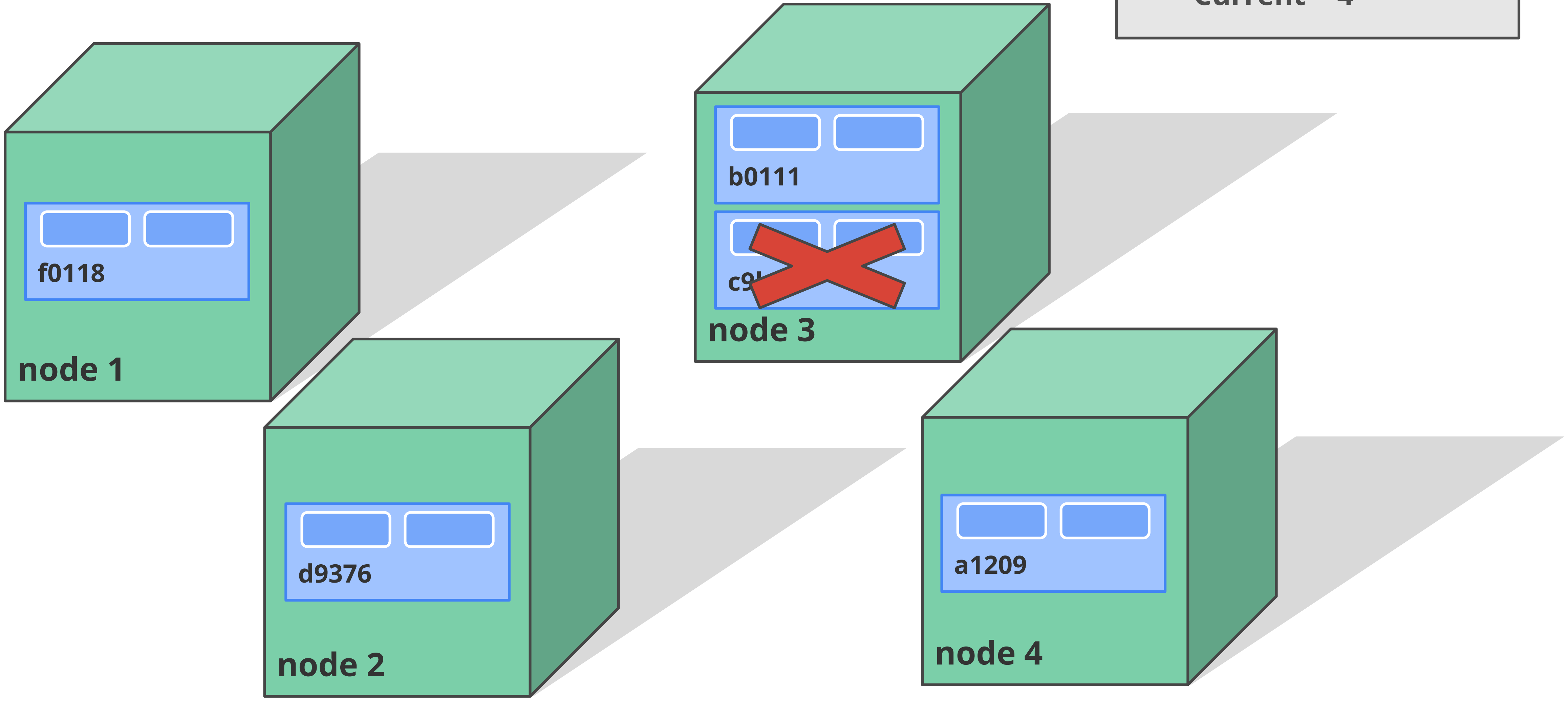
Replication Controllers

Replication Controller
- Desired = 4
- Current = 5



Replication Controllers

Replication Controller
- Desired = 4
- Current = 4



Pod networking

Pod IPs are **routable**

- Docker default is private IP

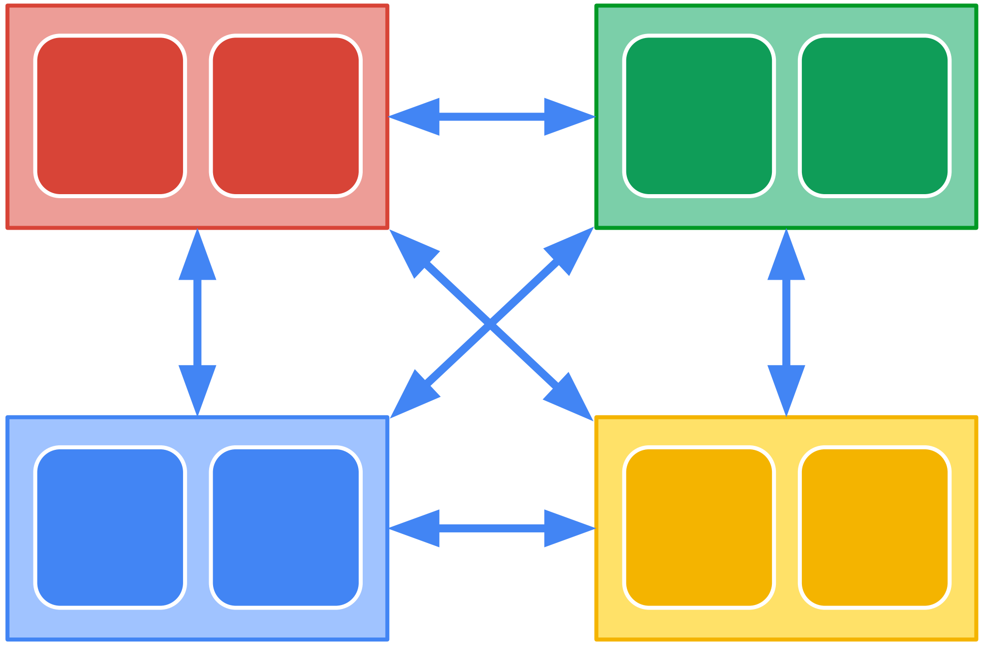
Pods can reach each other without NAT

- even across nodes

No brokering of port numbers

This is a fundamental requirement

- several SDN solutions



Services

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

- group == selector

Defines access policy

- only “load balanced” for now

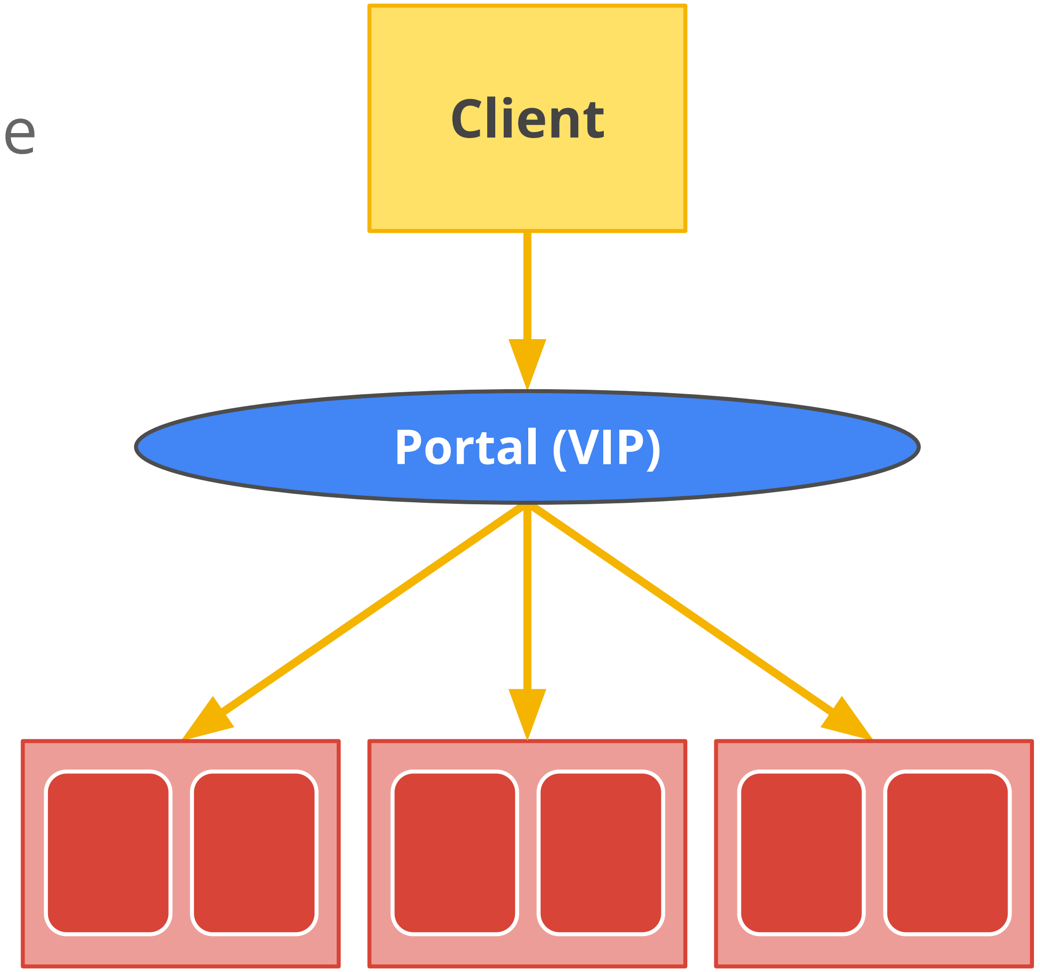
Gets a **stable** virtual IP and port

- called the service *portal*
- also a DNS name

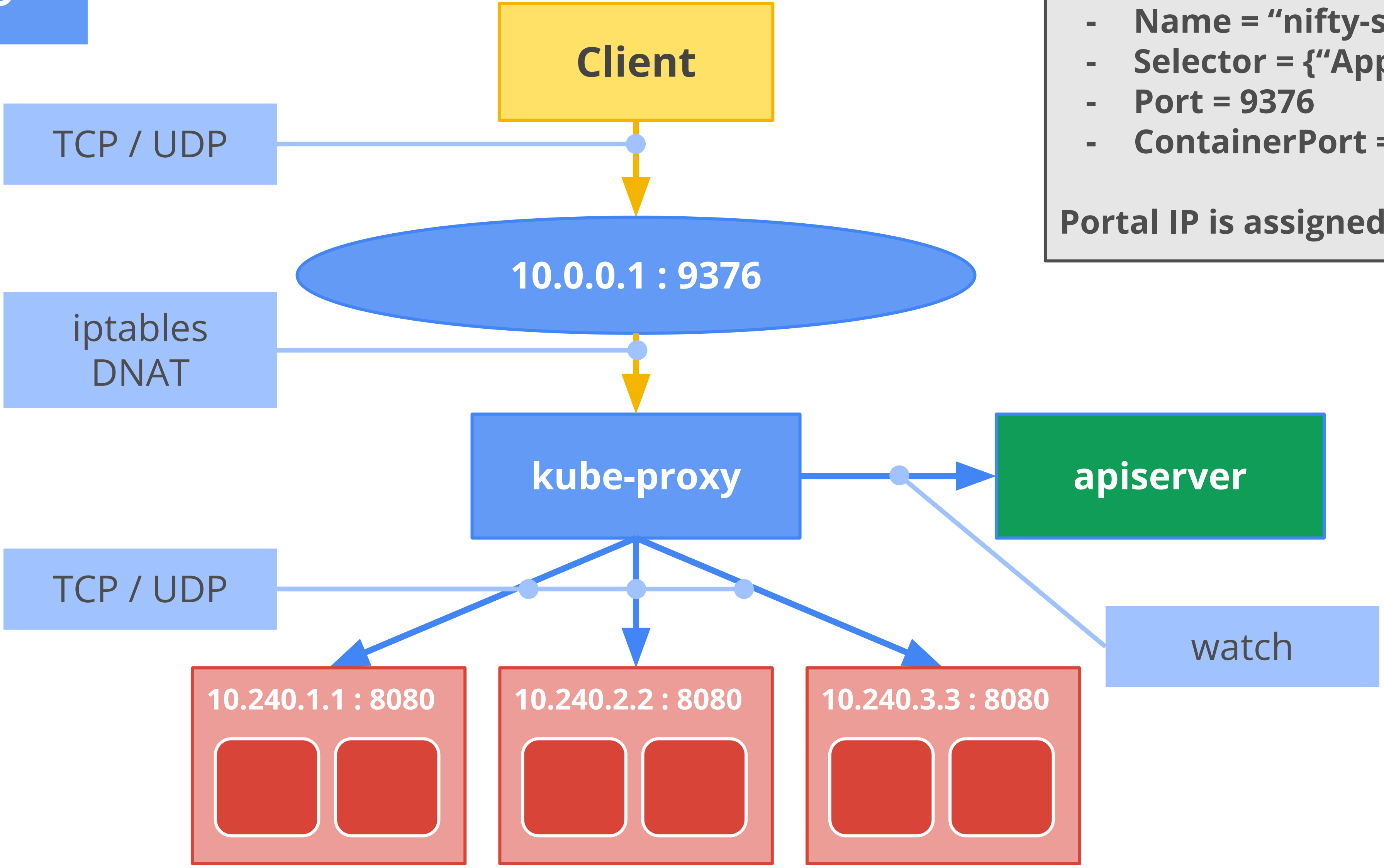
VIP is captured by *kube-proxy*

- 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

Events

A central place for information about your cluster

- filed by any component: kubelet, scheduler, etc

Real-time information on the current state of your pod

- **`kubectl describe pod foo`**

Real-time information on the current state of your cluster

- **`kubectl get --watch-only events`**
- You can also ask only for events that mention some object you care about.

Monitoring

Optional add-on to Kubernetes clusters

Run cAdvisor as a pod on each node

- gather stats from all containers
- export via REST

Run Heapster as a pod in the cluster

- just another pod, no special access
- aggregate stats

Run Influx and Grafana in the cluster

- more pods
- alternately: store in Google Cloud Monitoring

Logging

Optional add-on to Kubernetes clusters

Run fluentd as a pod on each node

- gather logs from all containers
- export to elasticsearch

Run Elasticsearch as a pod in the cluster

- just another pod, no special access
- aggregate logs

Run Kibana in the cluster

- yet another pod
- alternately: store in Google Cloud Logging