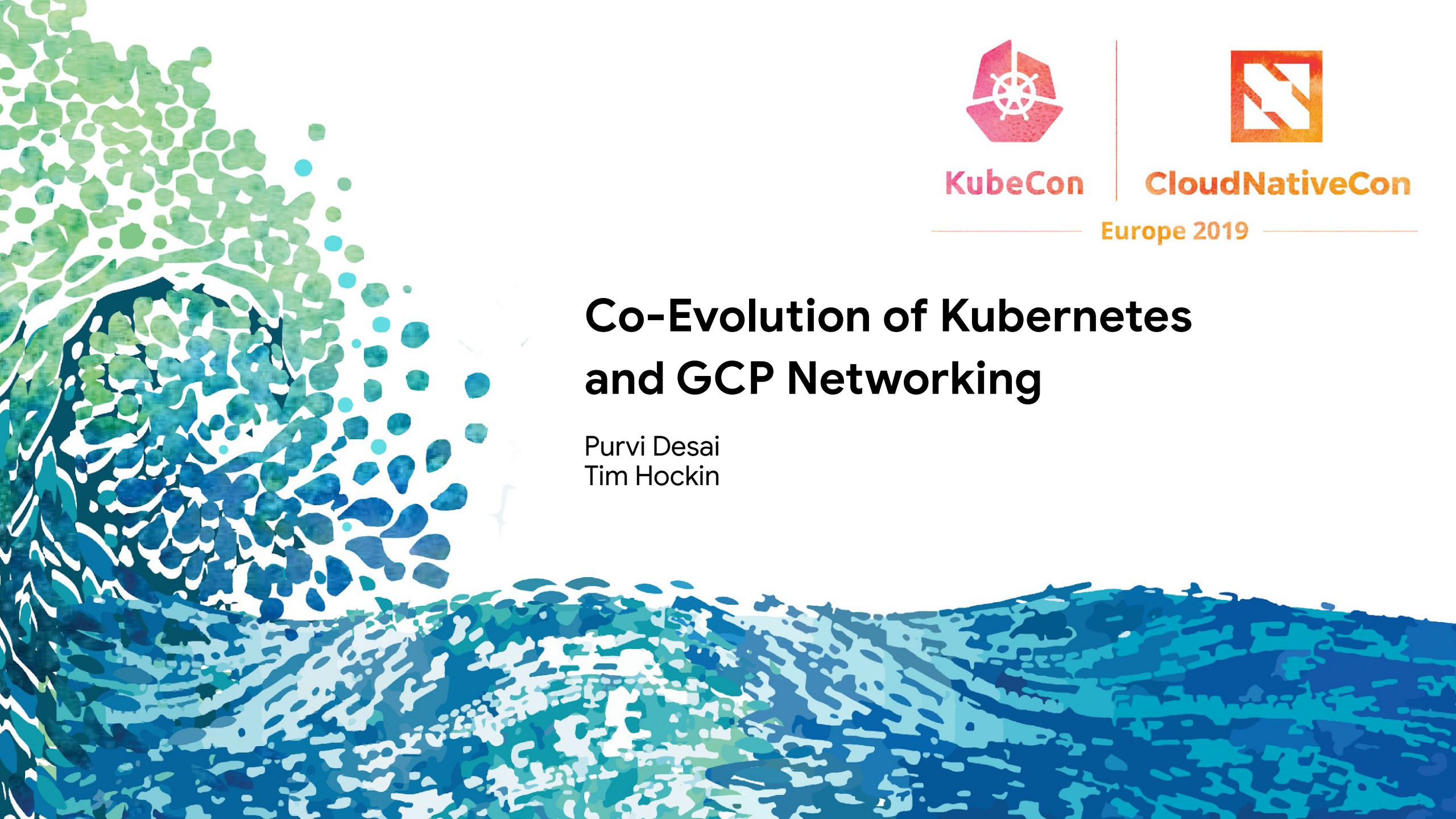


KubeCon | CloudNativeCon

Europe 2019

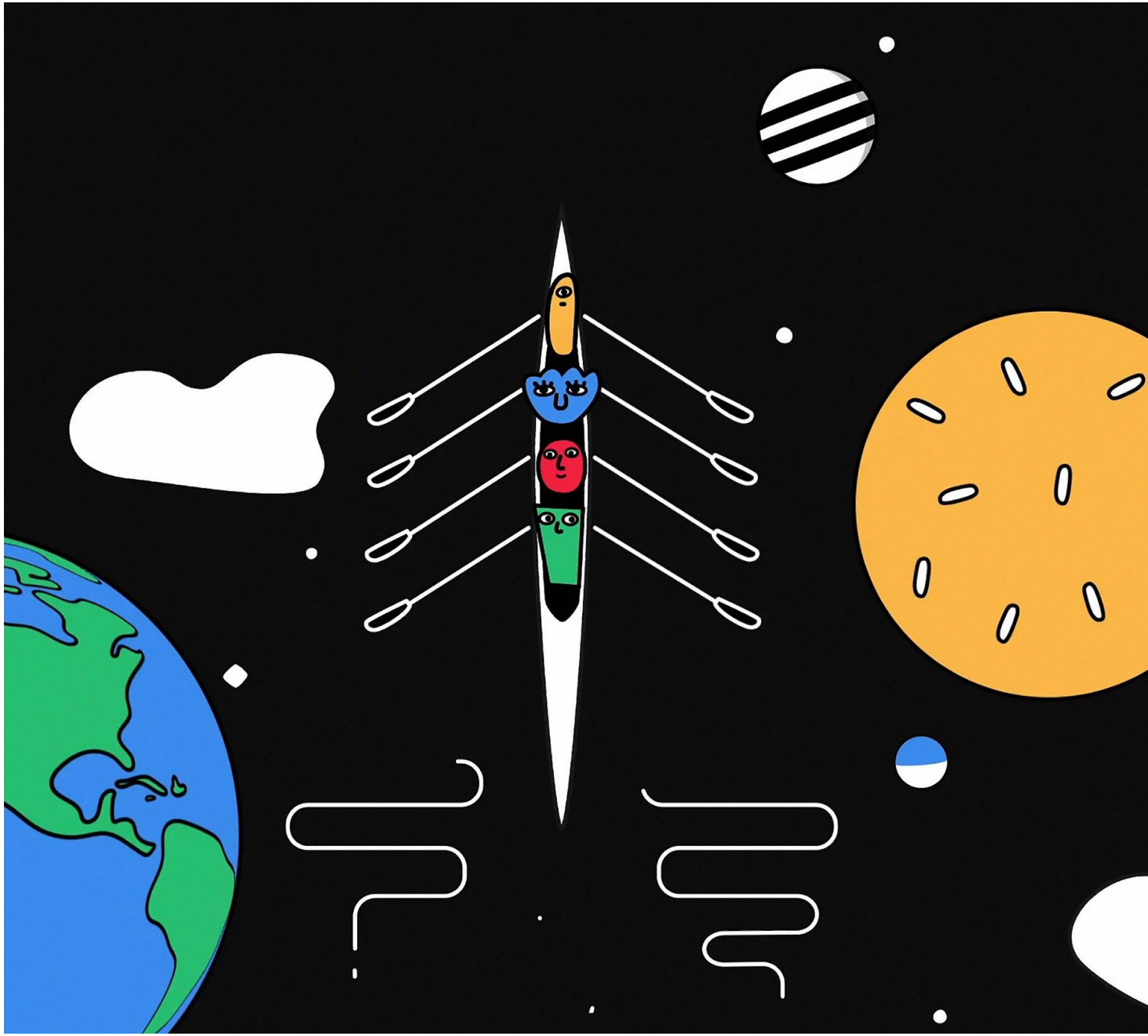
Co-Evolution of Kubernetes and GCP Networking

Purvi Desai
Tim Hockin

Why did Kubernetes take off?

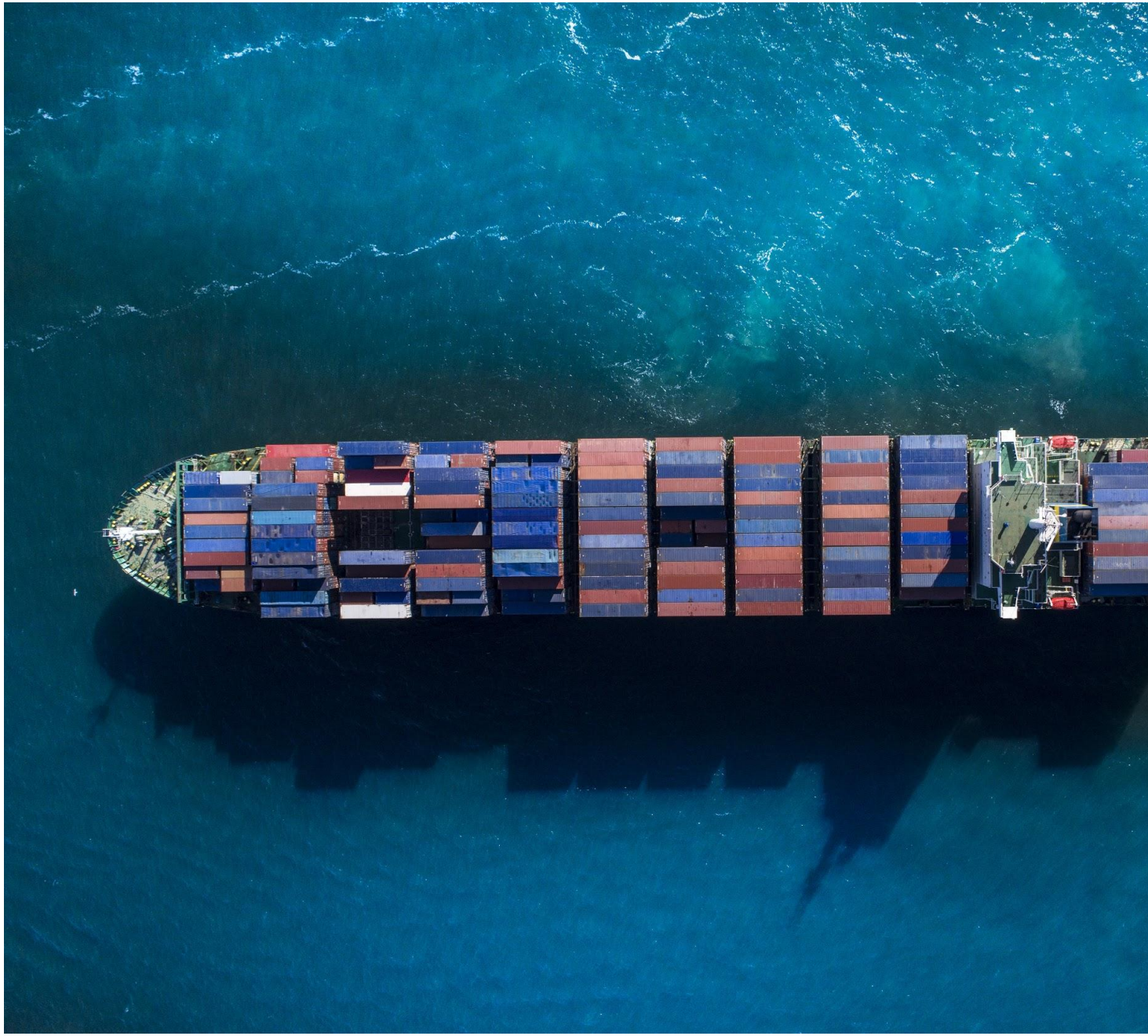
- Focused on app owners and app problems
- “Opinionated enough”
- Assumes platform implementations will vary
- Designed to work with popular OSS
- Follows understood conventions (mostly)

Google Cloud



Networking is at the heart of Kubernetes

- Almost every k8s-deployed app needs it
- Networking can be complex
- Details vary a lot between environments
- App developers shouldn’t have to be networking experts



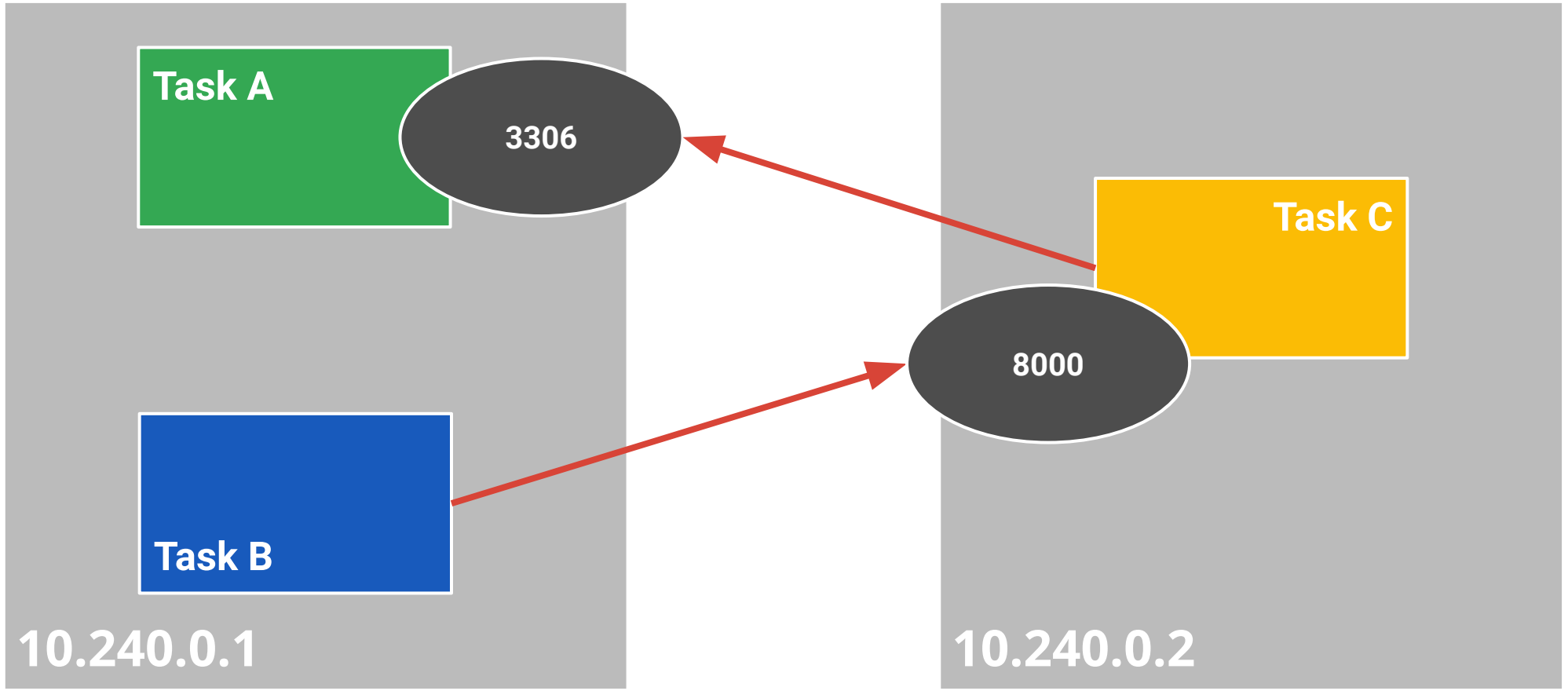
Google Cloud

In the beginning

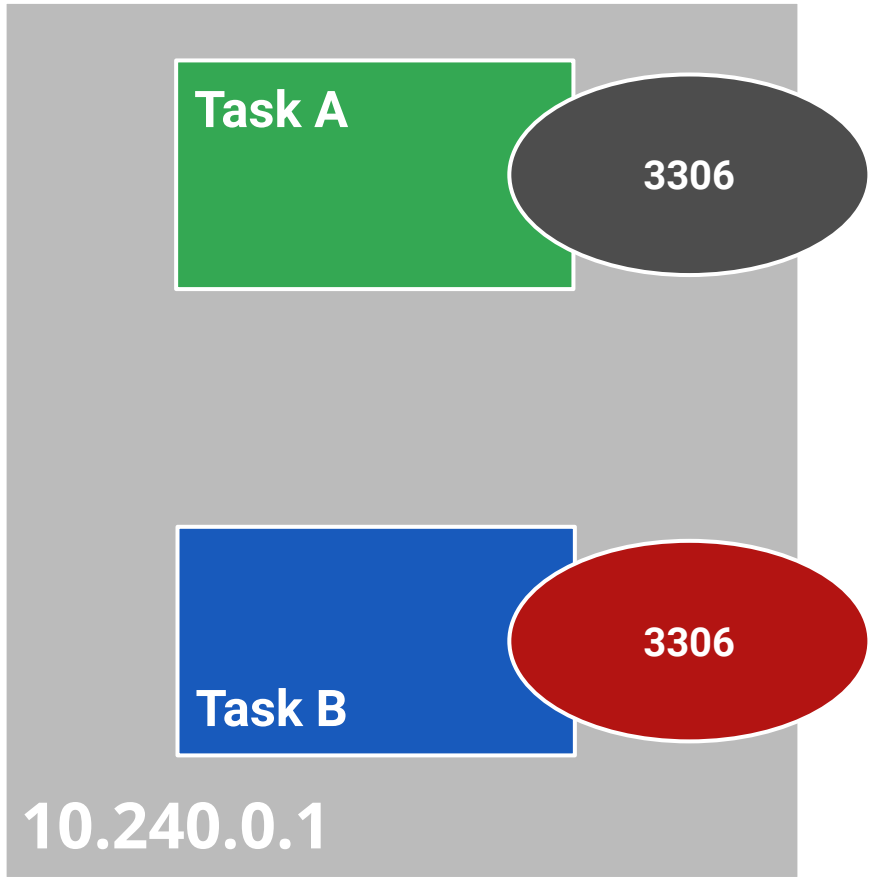
Lineage of Borg

Survey of Container Networking

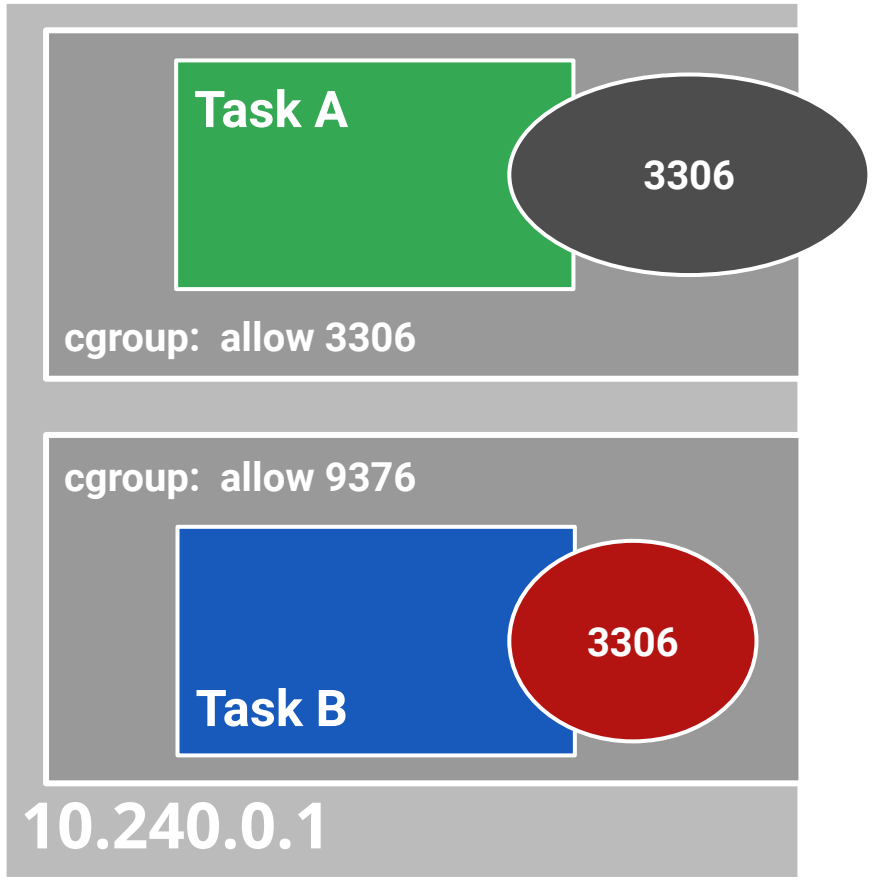
Borg model



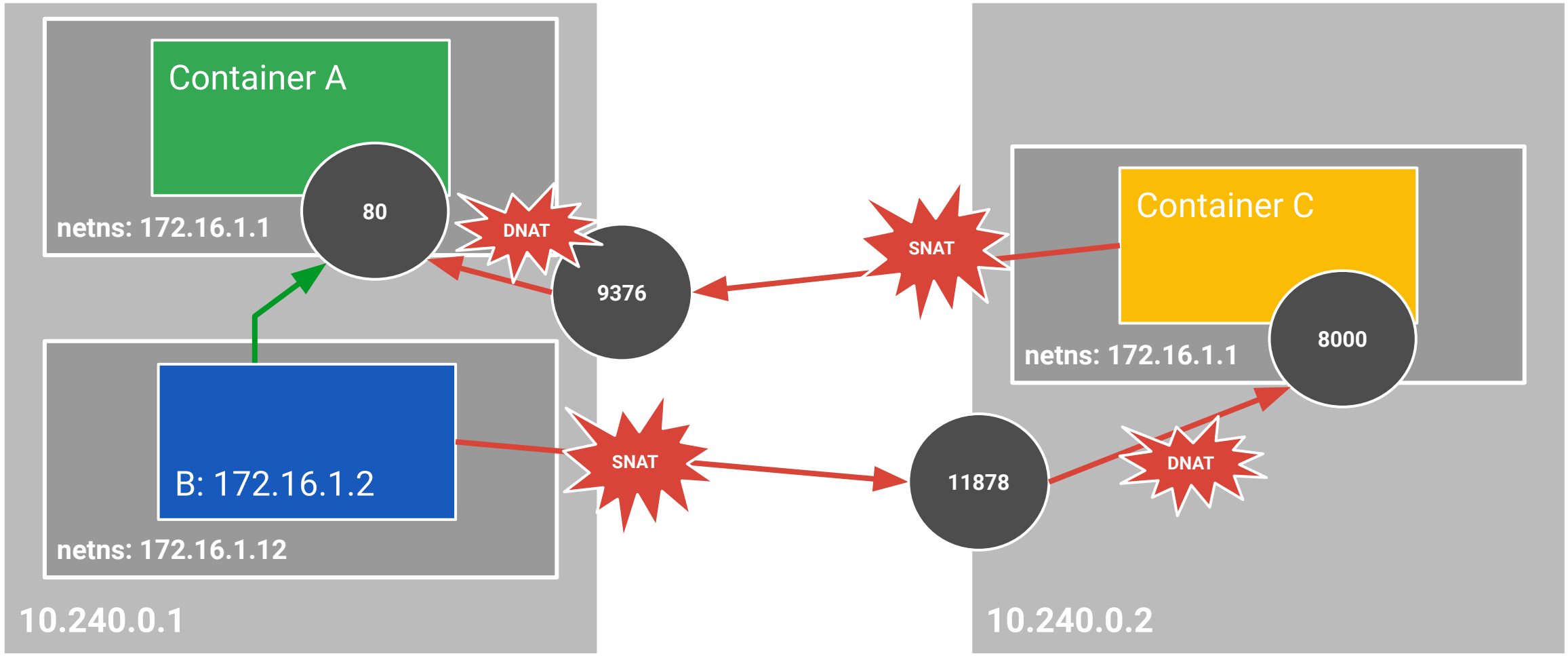
Borg model



Borg model


Task A
3306
cgroup: allow 3306
cgroup: allow 9376
3306
Task B
10.240.0.1

Original docker model



Google Cloud

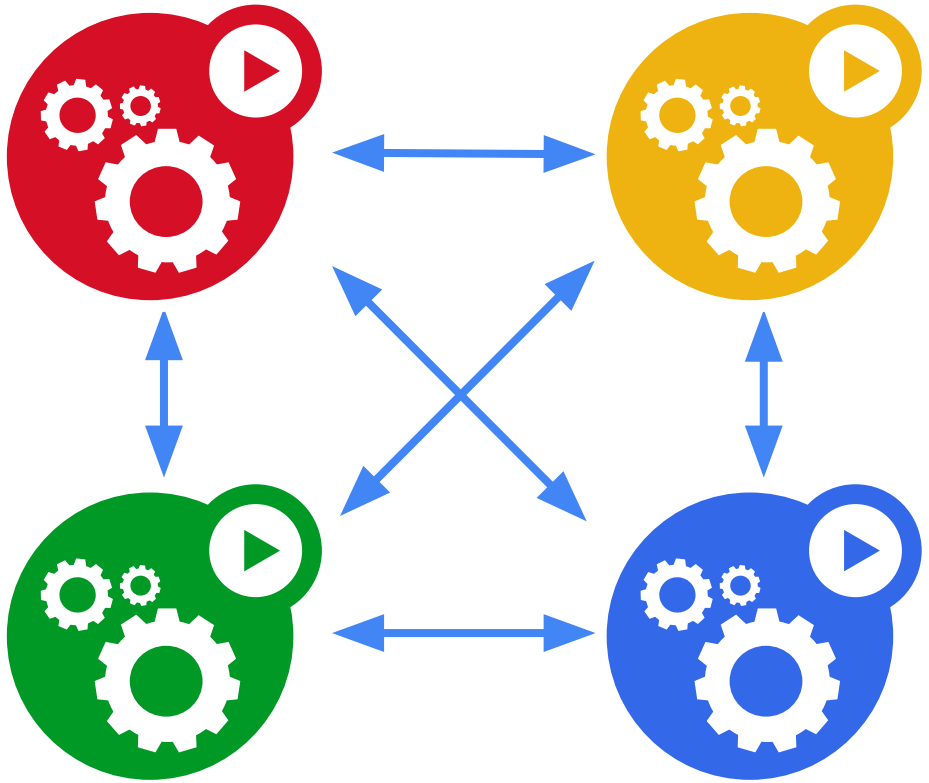
Kubernetes network model

Users should never have to worry about collisions that they themselves didn't cause

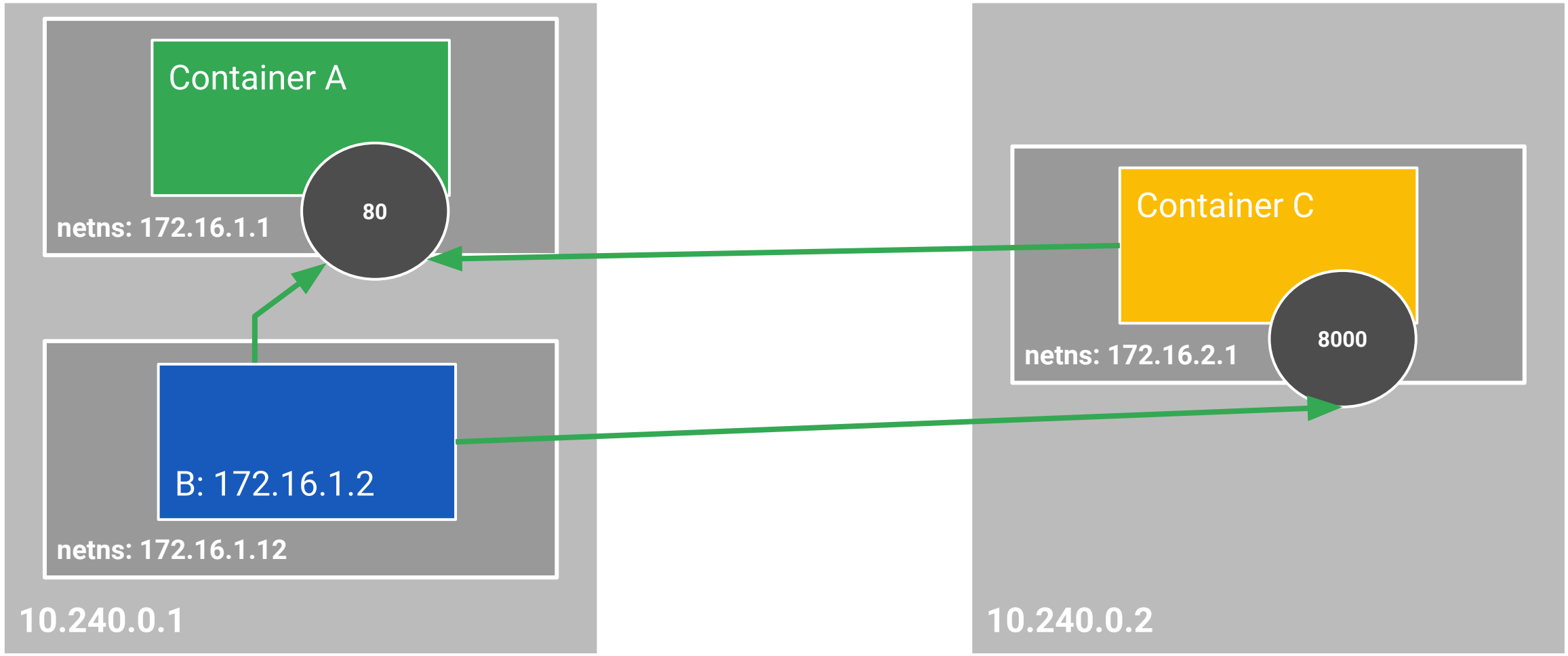
App developers shouldn't have to be networking experts

A real IP for every Pod

- Pod IPs are accessible from other pods, regardless of which VM they are on
- No brokering of port numbers



Kubernetes model



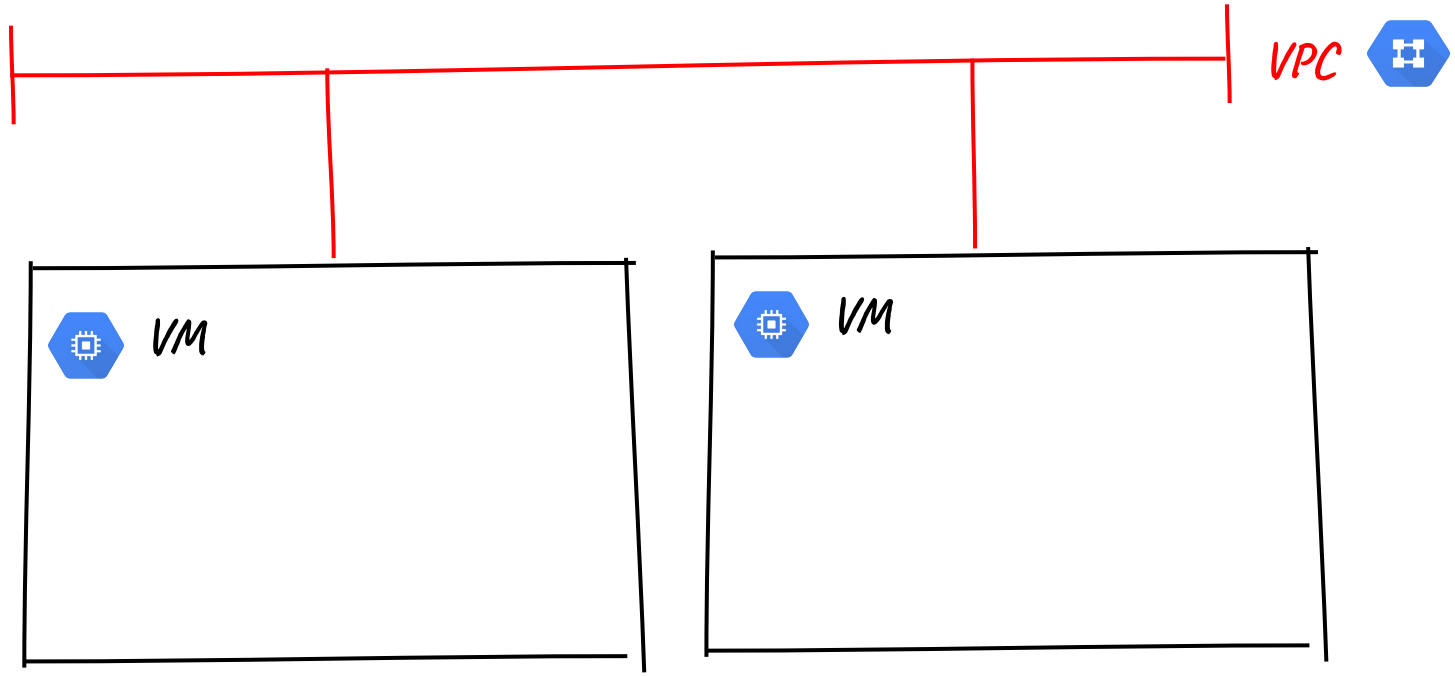
Google Cloud

Proof of concept

Early Experiments on GCP

Cloud networking

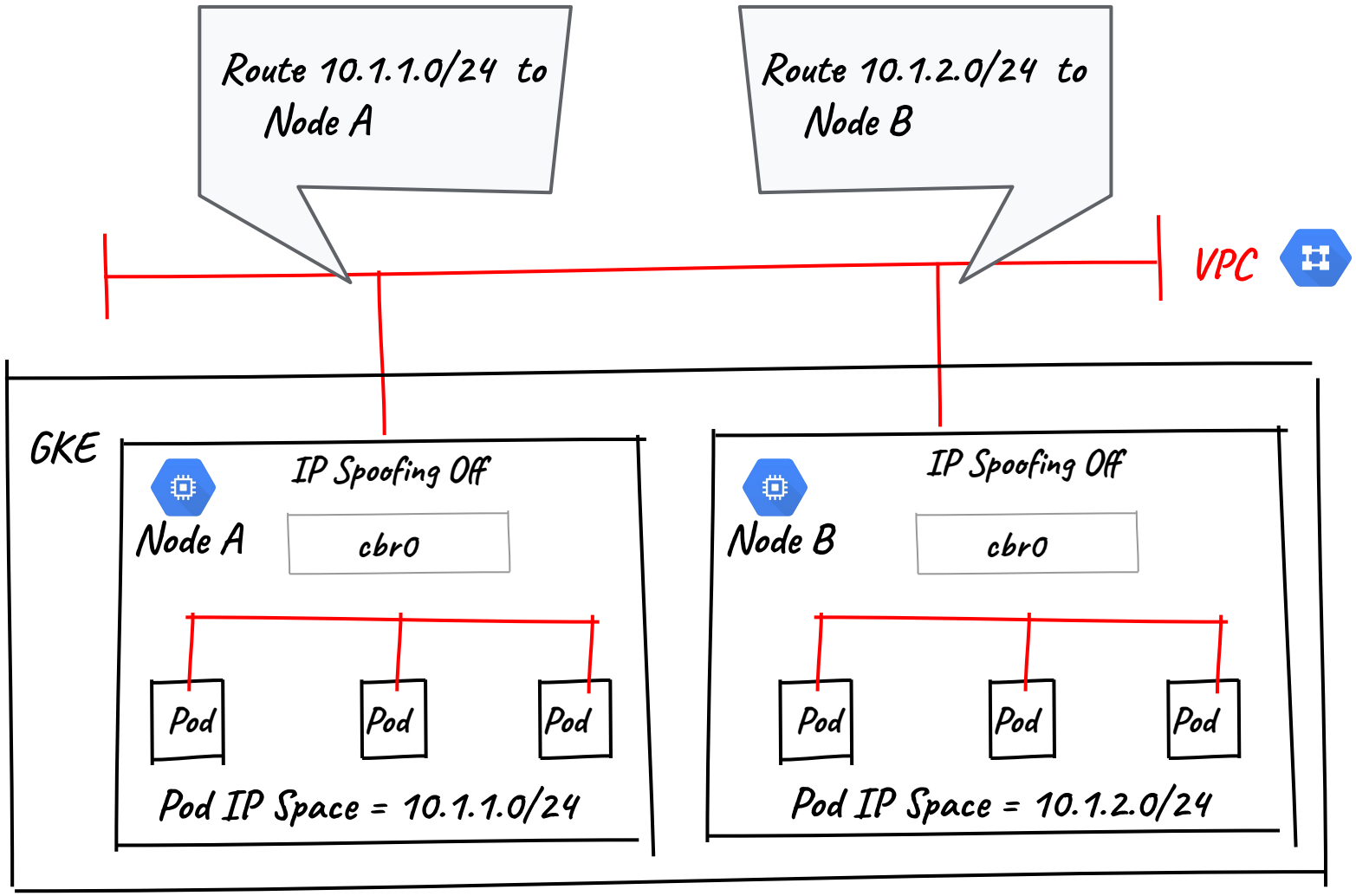
- VM Centric
- Containers are not really a part of design space
- What were the possibilities?



Google Cloud

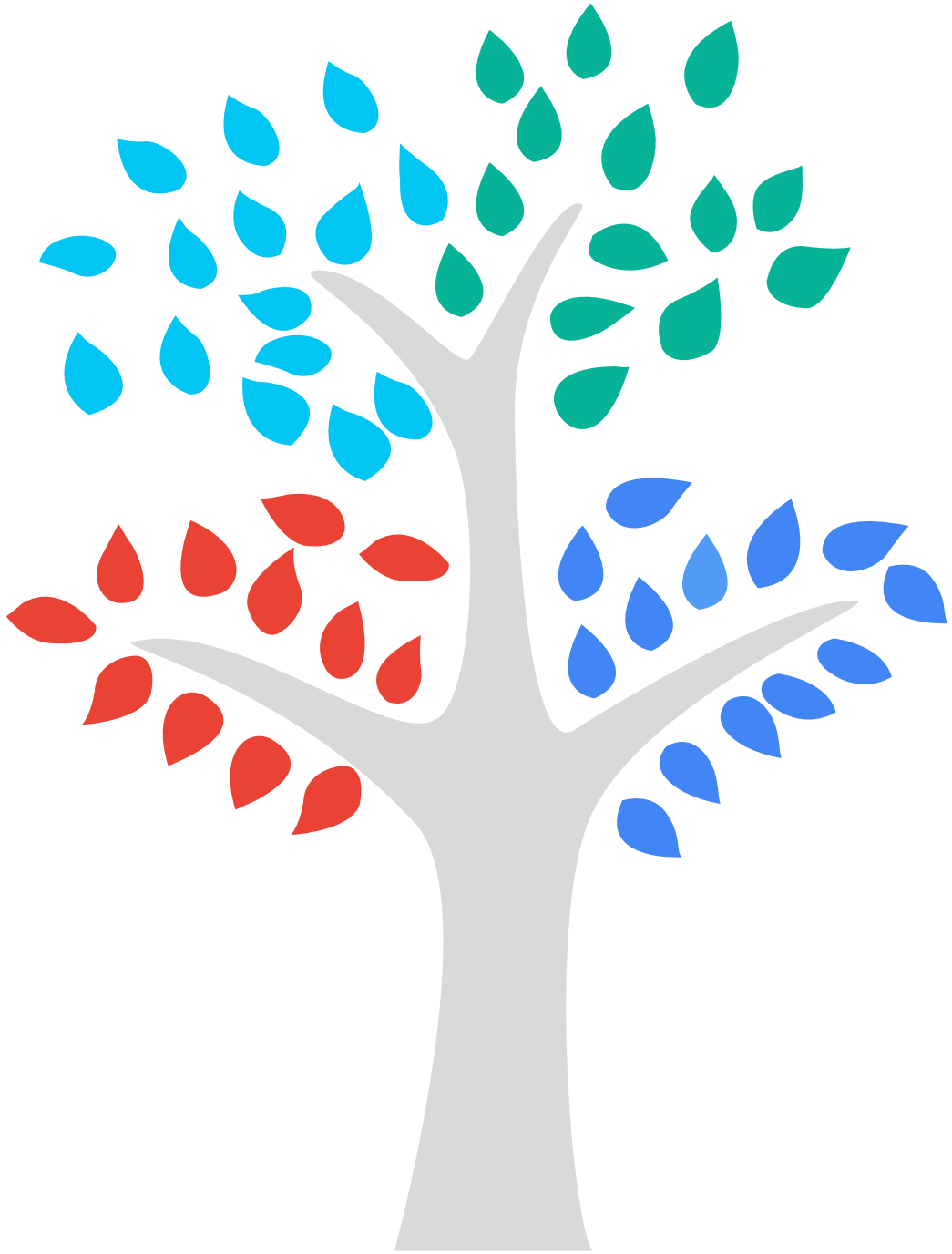
Found a toehold

- **The “Routes” API**
- Every VM claims to be a router
- Disable IP spoofing protection



The beginning of co-evolution

- Foundations were set
- UX was good - IP-per-Pod worked!
- We were able to push limits to 100 routes
- Does anyone remember how many nodes Kubernetes 1.0 supported?



Google Cloud

Co-evolution Journey

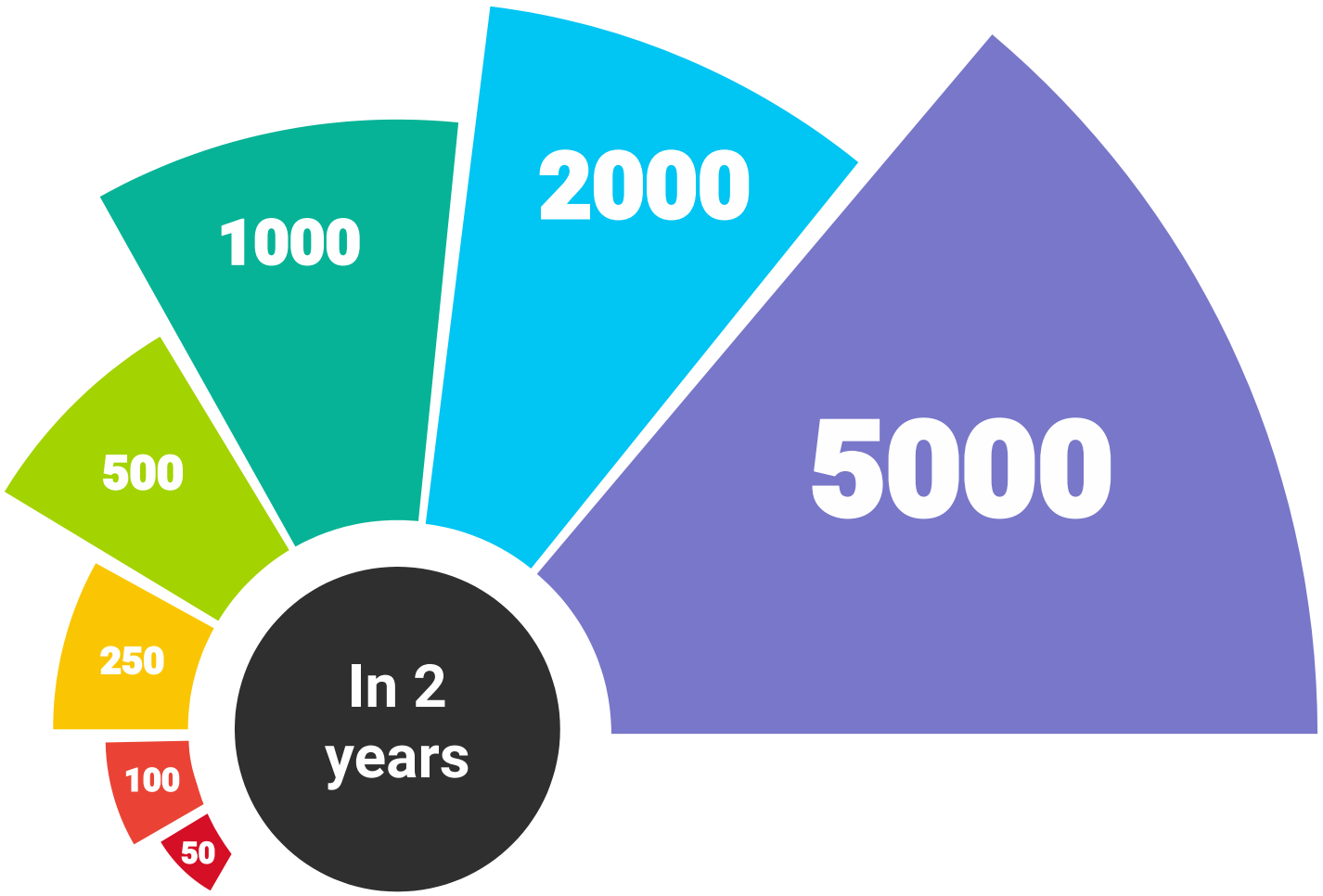
Cluster Networking

Services and L4 Load Balancers

L7 load balancer

Cluster Networking Routes model

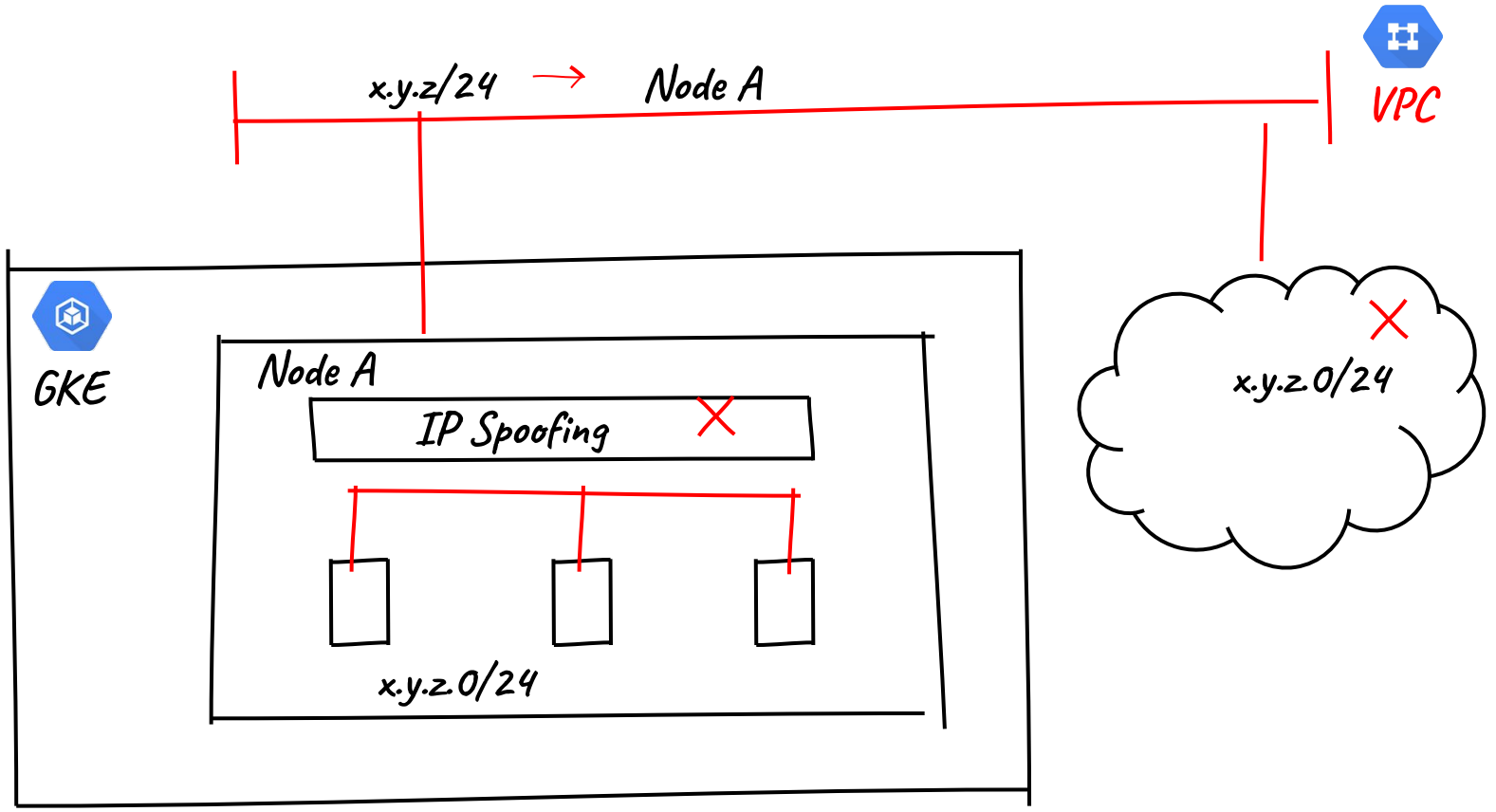
- Drove major architectural changes to scale GCP's Routes subsystem
- Rapid scaling over 2 years



Google Cloud

What's the catch?

- IP spoofing disabled
- Semi-hidden allocations - potential for collisions with future uses of IPs
- Overlapping routes caused real confusion, hard to debug



We can do better

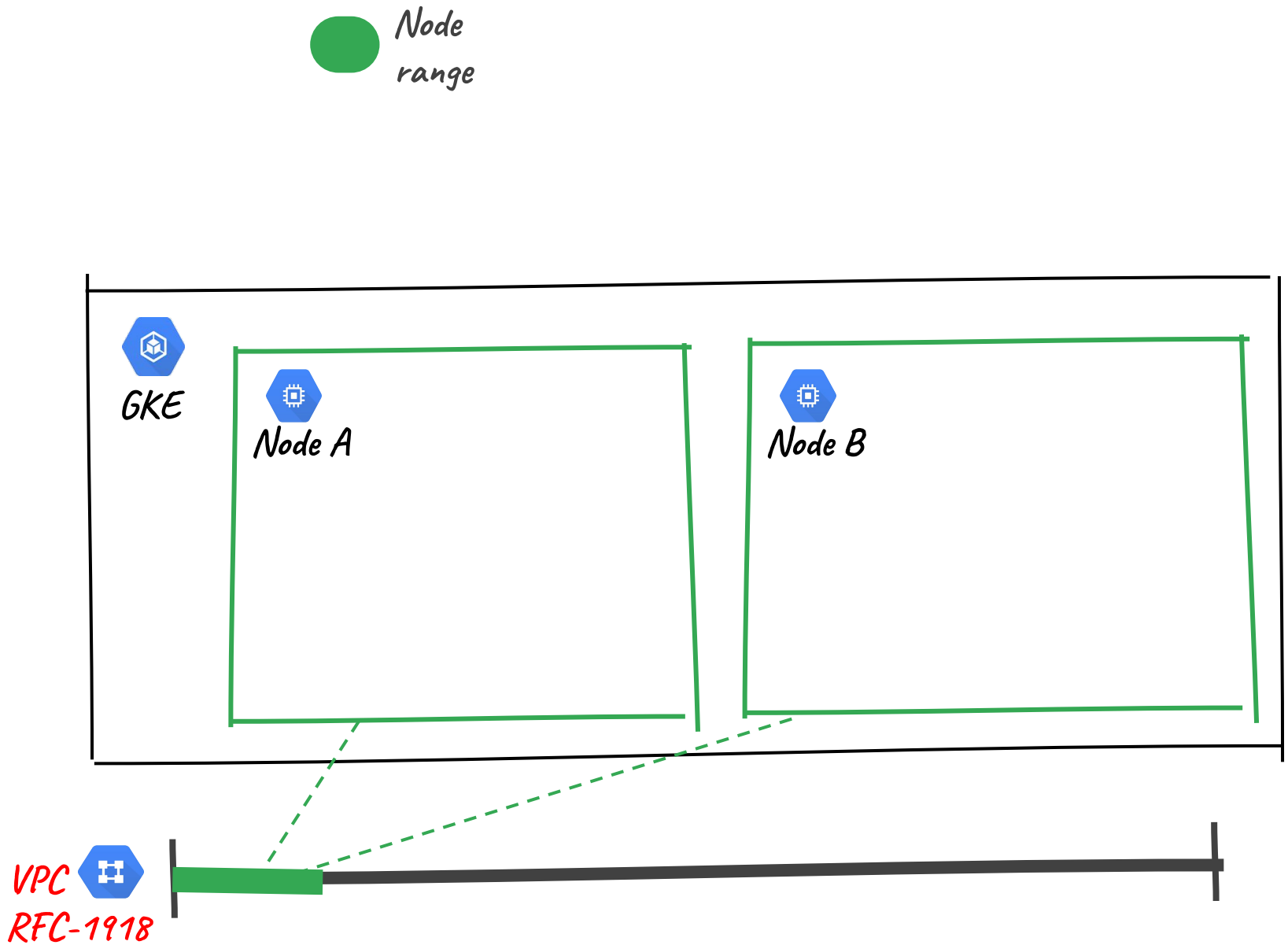
Better integration with other products

Hard to reason about & debug

Need a deeper concept: Alias IPs

Alias IPs & integrated networking

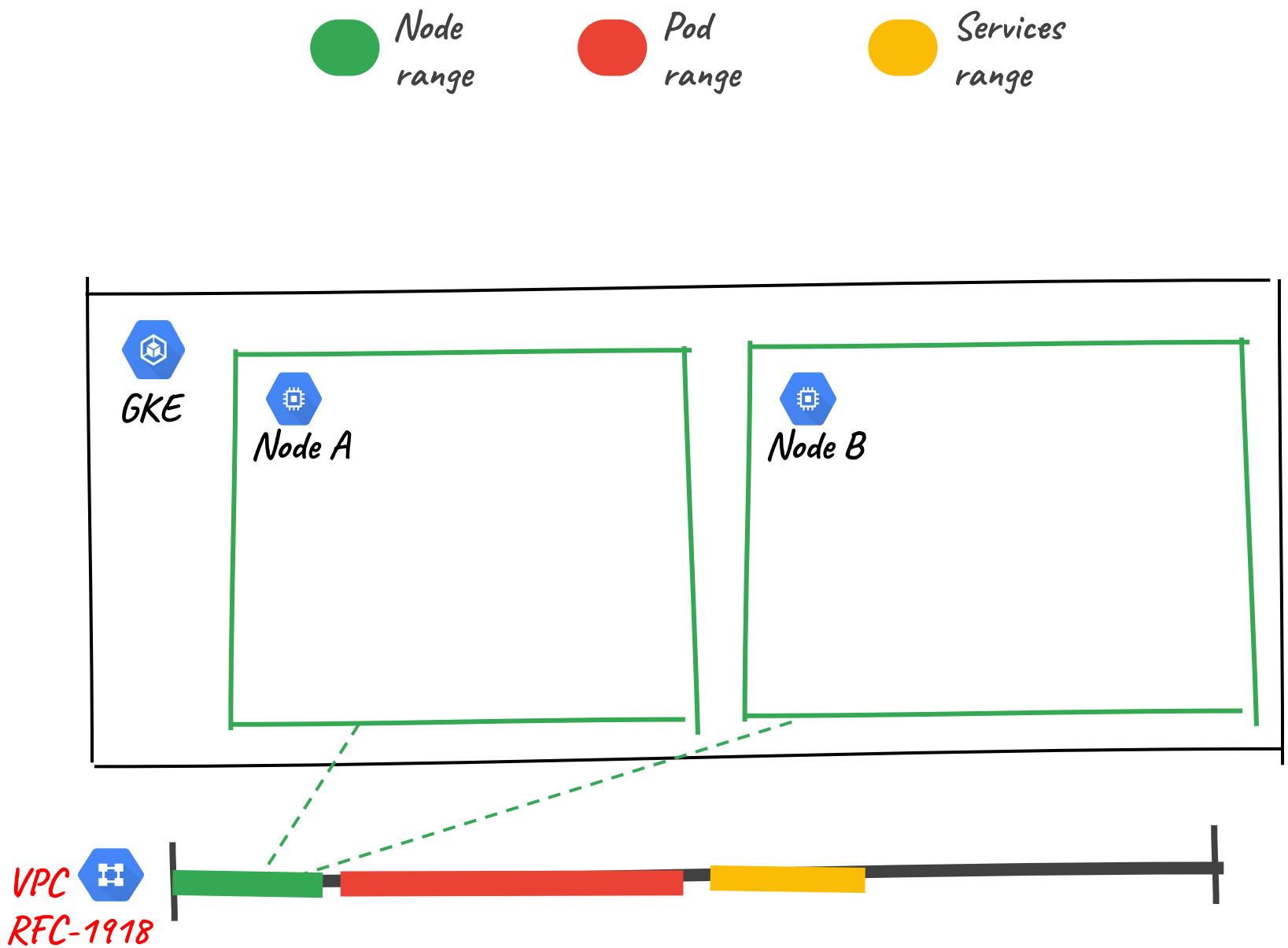
- Allocate range for nodes



Google Cloud

Alias IPs & integrated networking

- Allocate range for nodes
- Allocate ranges for pods and services



Google Cloud

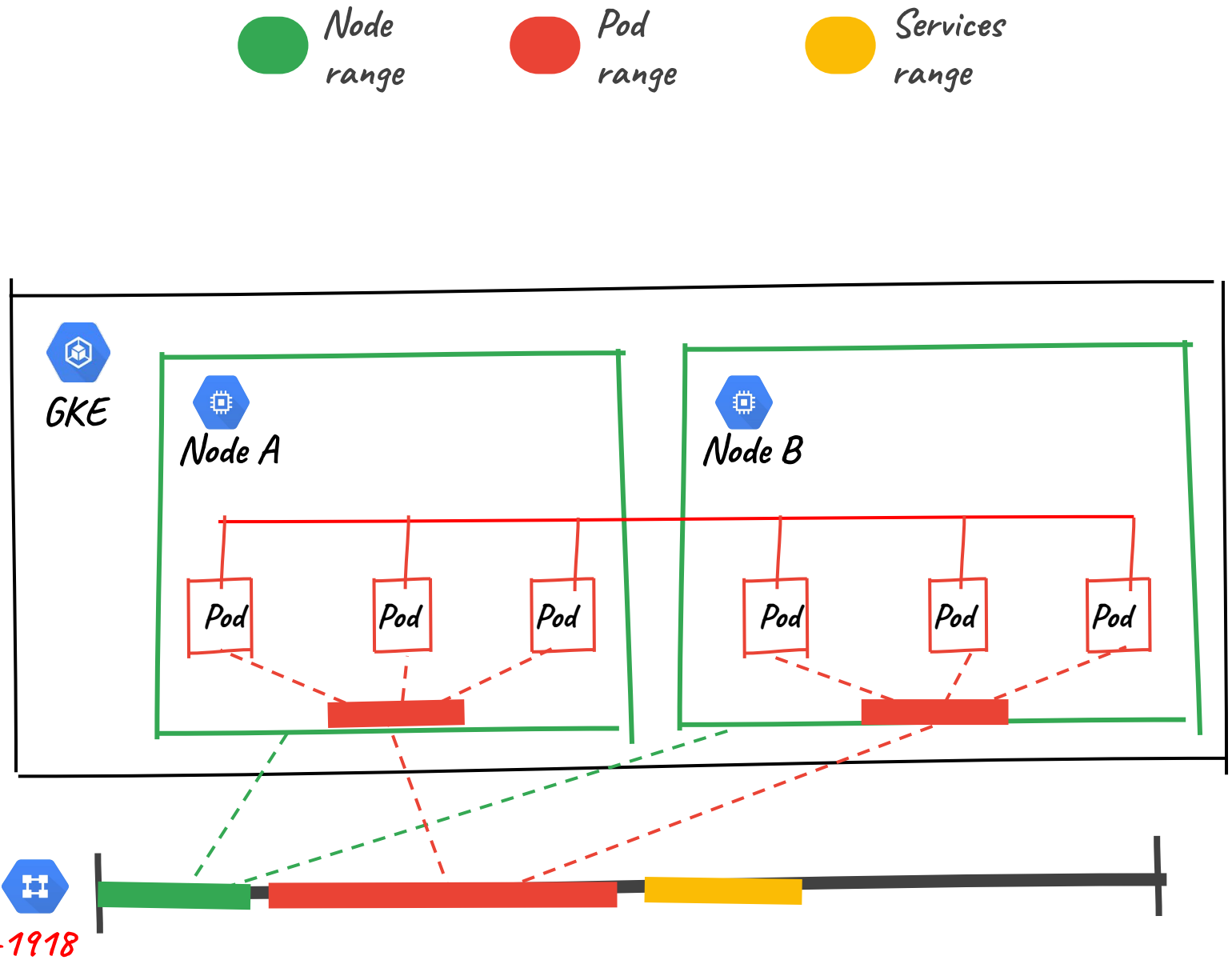
Alias IPs & integrated networking

- Allocate range for nodes
- Allocate ranges for pods and services
- Carve off per-VM pod-ranges automatically as alias IPs
- SDN understands Alias IPs
- Per-node IPAM is in cloud

Google Cloud

Alias IPs & integrated networking

- Allocate range for nodes
- Allocate ranges for pods and services
- Carve off per-VM pod-ranges automatically as alias IPs
- SDN understands Alias IPs
- Per-node IPAM is in cloud, on-node IPAM is on-node
- No VPC collisions, now or future



Google Cloud

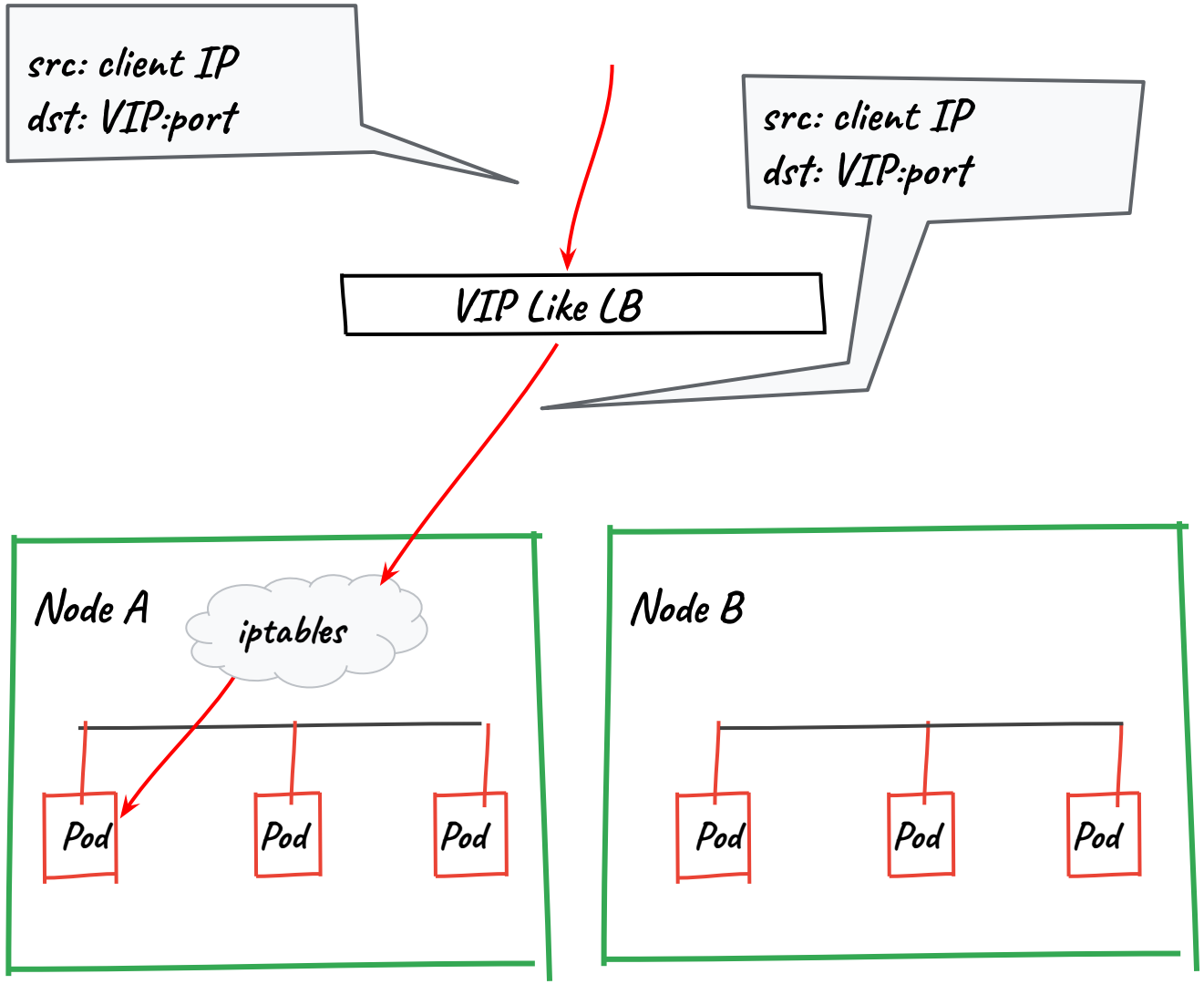
Services & load-balancers

LB support centered around clouds

Implemented by the cloud provider controller

VIP Like LBs

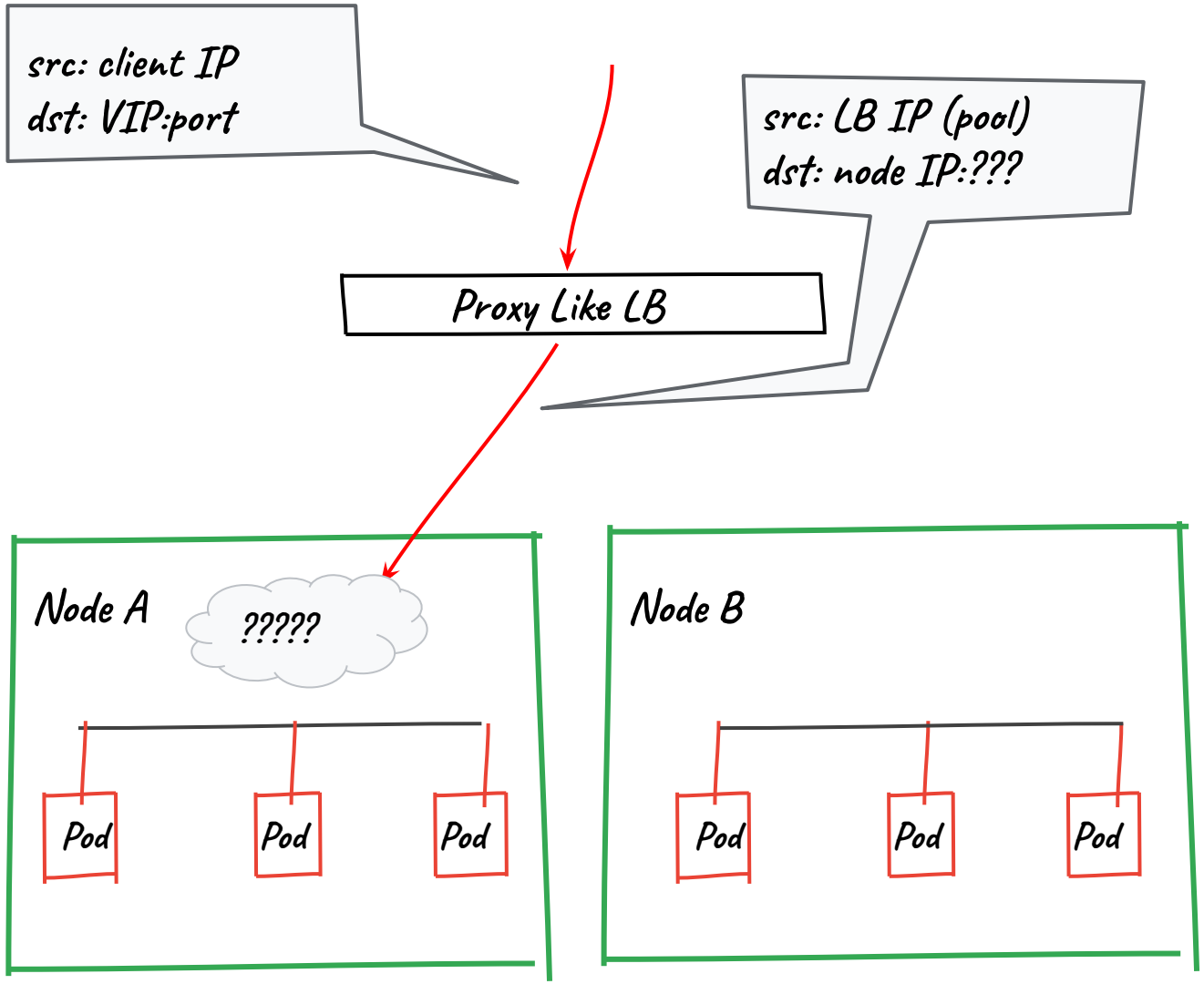
- LB Delivers Packet from original client IP to original VIP
- IPTables are programmed to capture the VIP just like a Cluster IP
- IPTables takes care of the rest
- GCP's Network LB is VIP-Like
- LB only knows Nodes, k8s translates to Services and Pods



Google Cloud

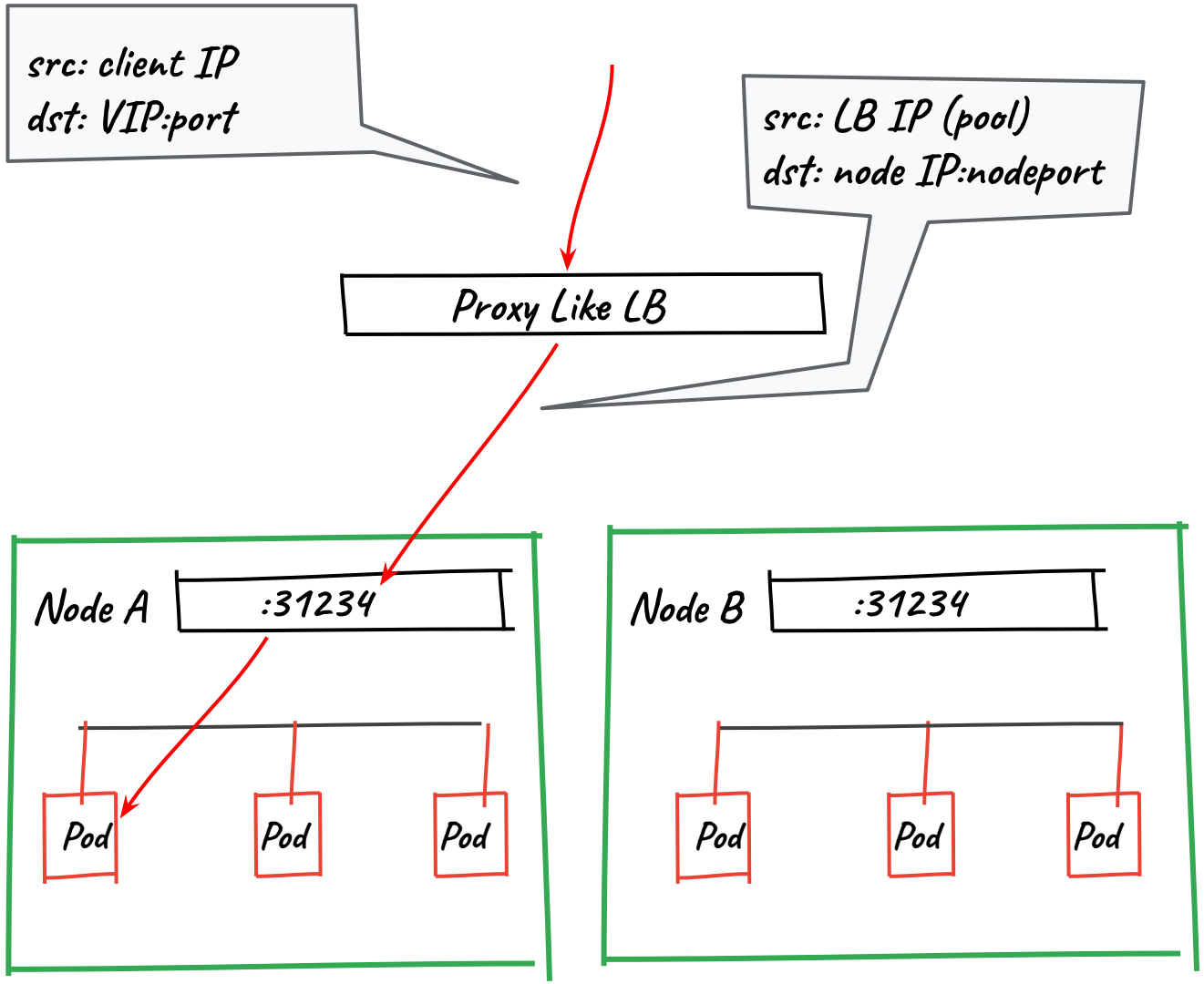
Proxy Like LBs

- LB acts as proxy and delivers packet from proxy to Node or Pod
- AWS's ELB is Proxy-Like
- Again, LBs only understand Nodes, not Pods or Services
- How to indicate which Service?



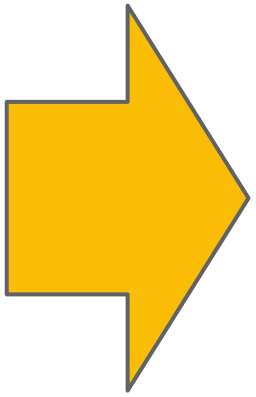
Introduction of NodePorts

- Allocate a static port across all nodes, one for each LB'ed Service
- Simple to understand model
- Portable: No external dependencies

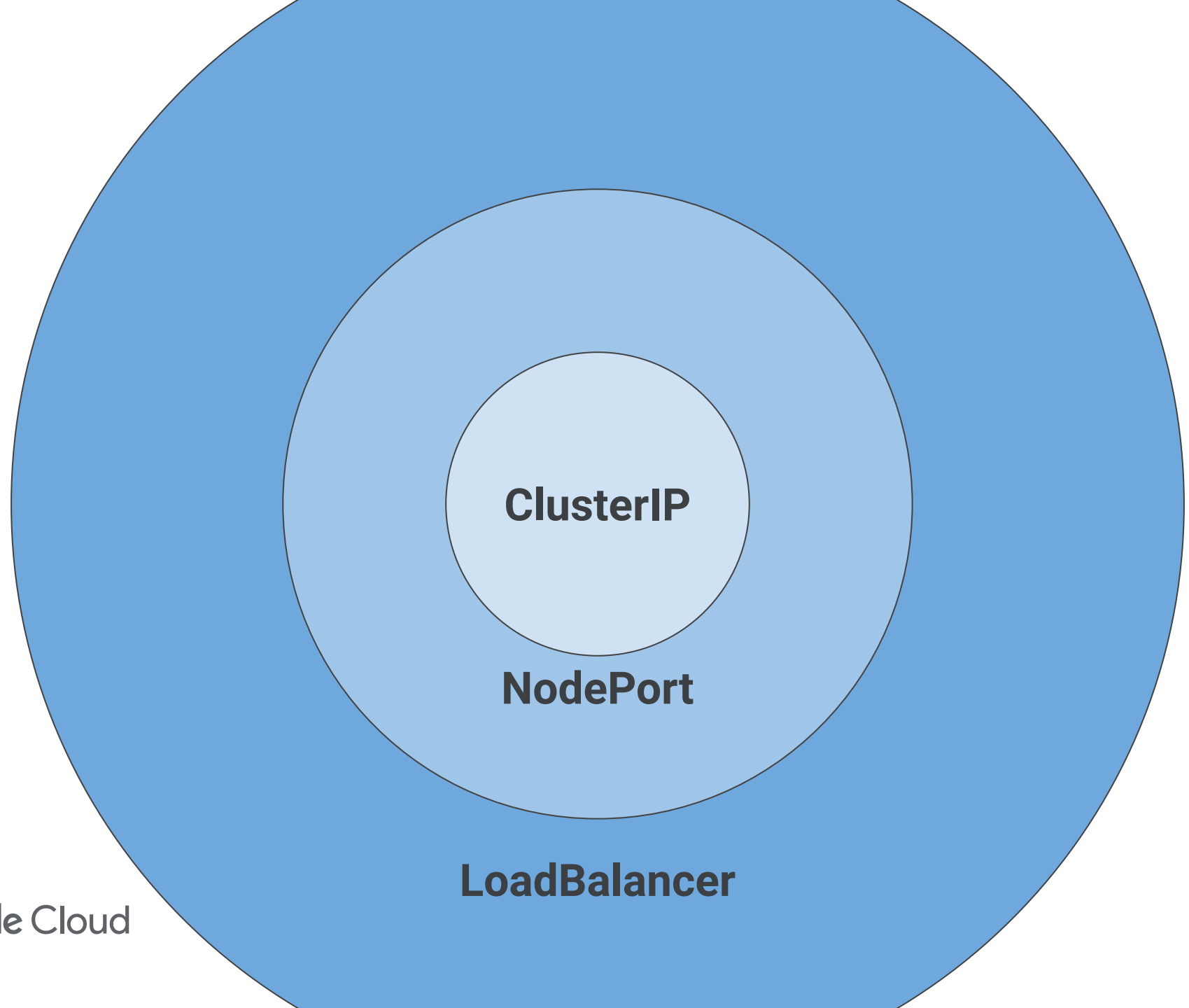


What about portability?

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend
```



```
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  type: LoadBalancer
  clusterIP: 10.15.251.118
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
    nodePort: 30669
  selector:
    app: guestbook
    tier: frontend
status:
  loadBalancer:
    ingress:
    - ip: 35.193.47.73
```



Google Cloud

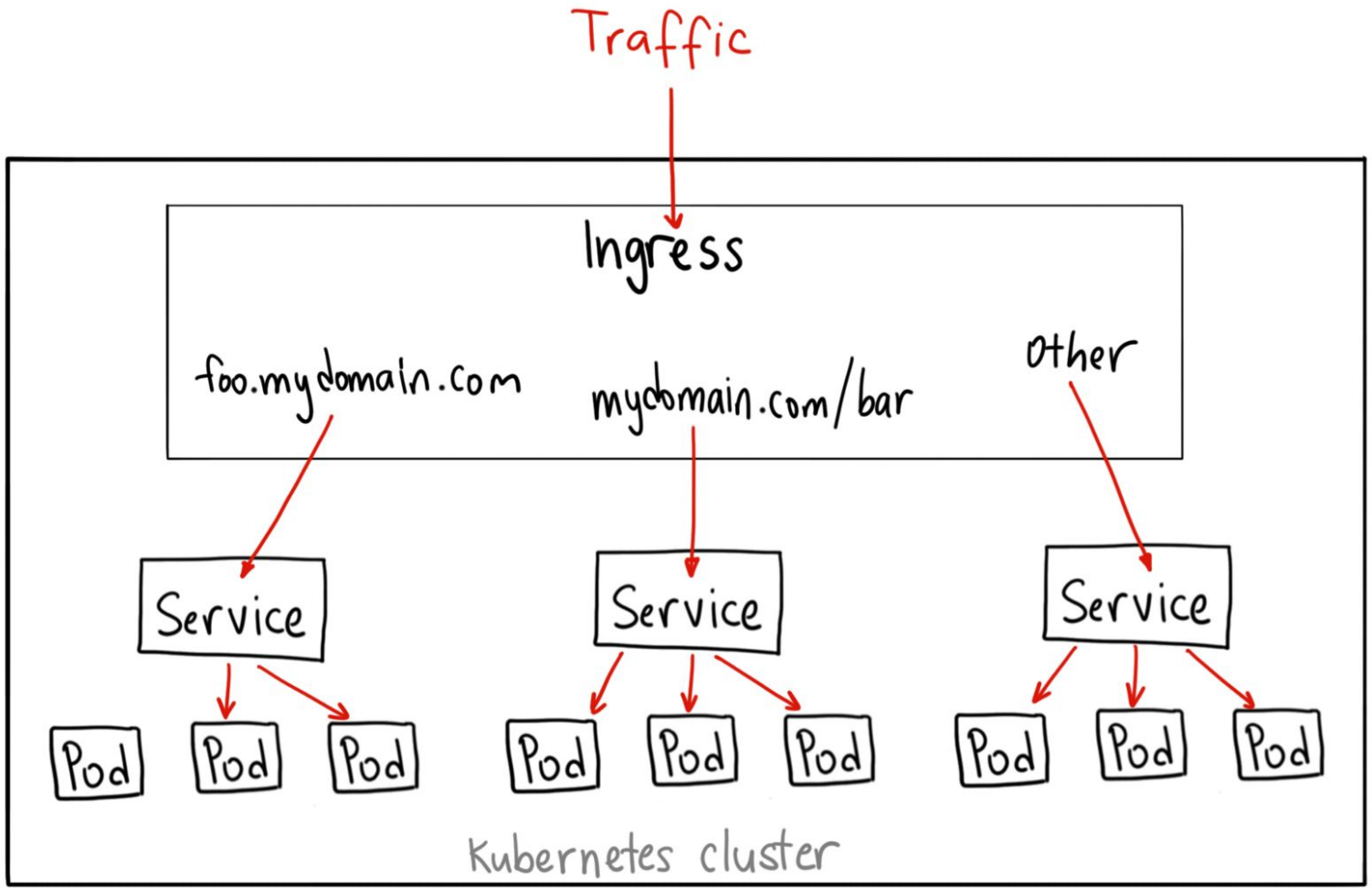
Ingress: L7 LB

All (or almost) L7 LBs are proxy like

NodePorts are a decent starting point

Ingress

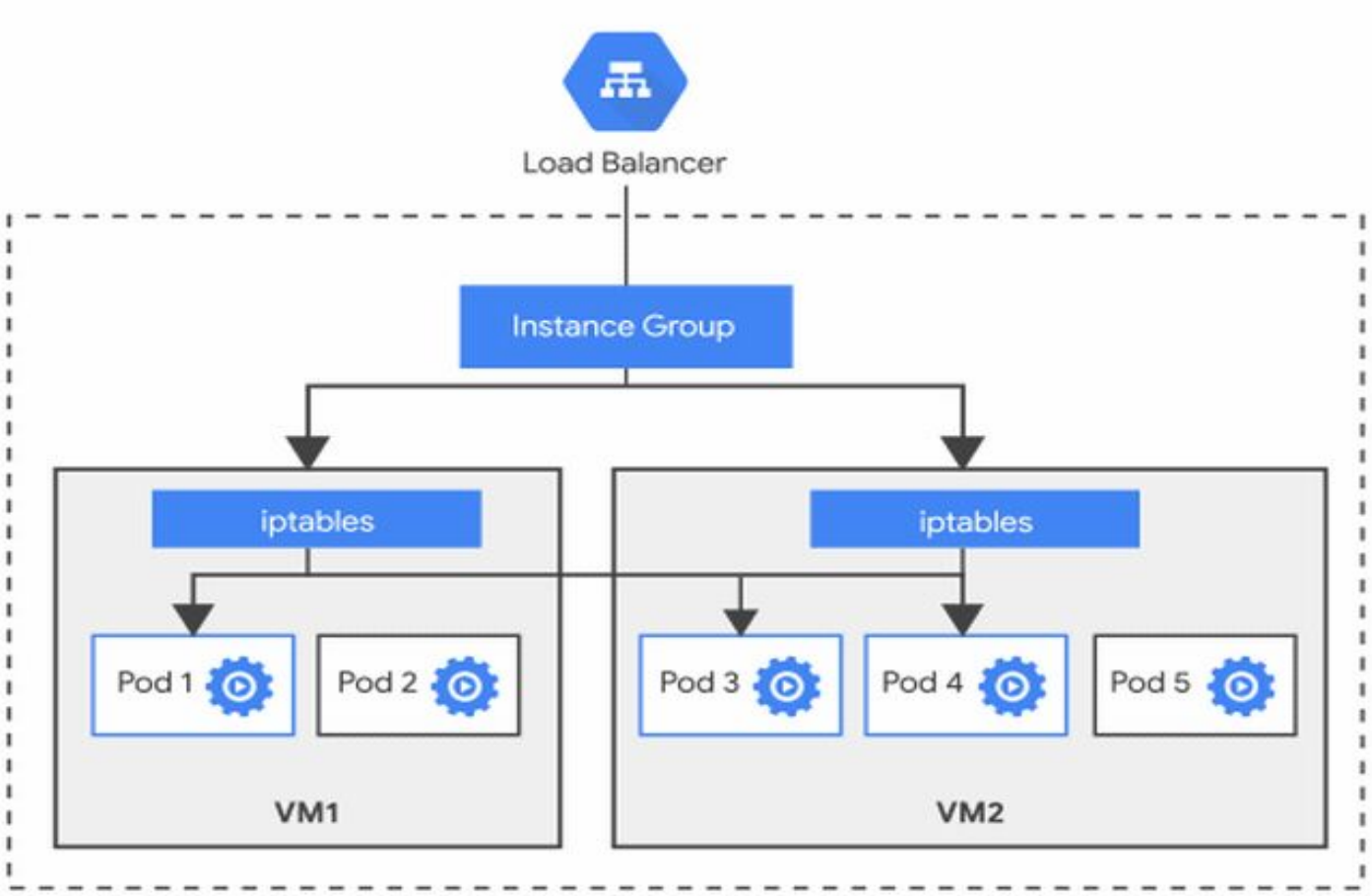
Portable L7 LB Abstraction



Google Cloud

Advancing LBs From

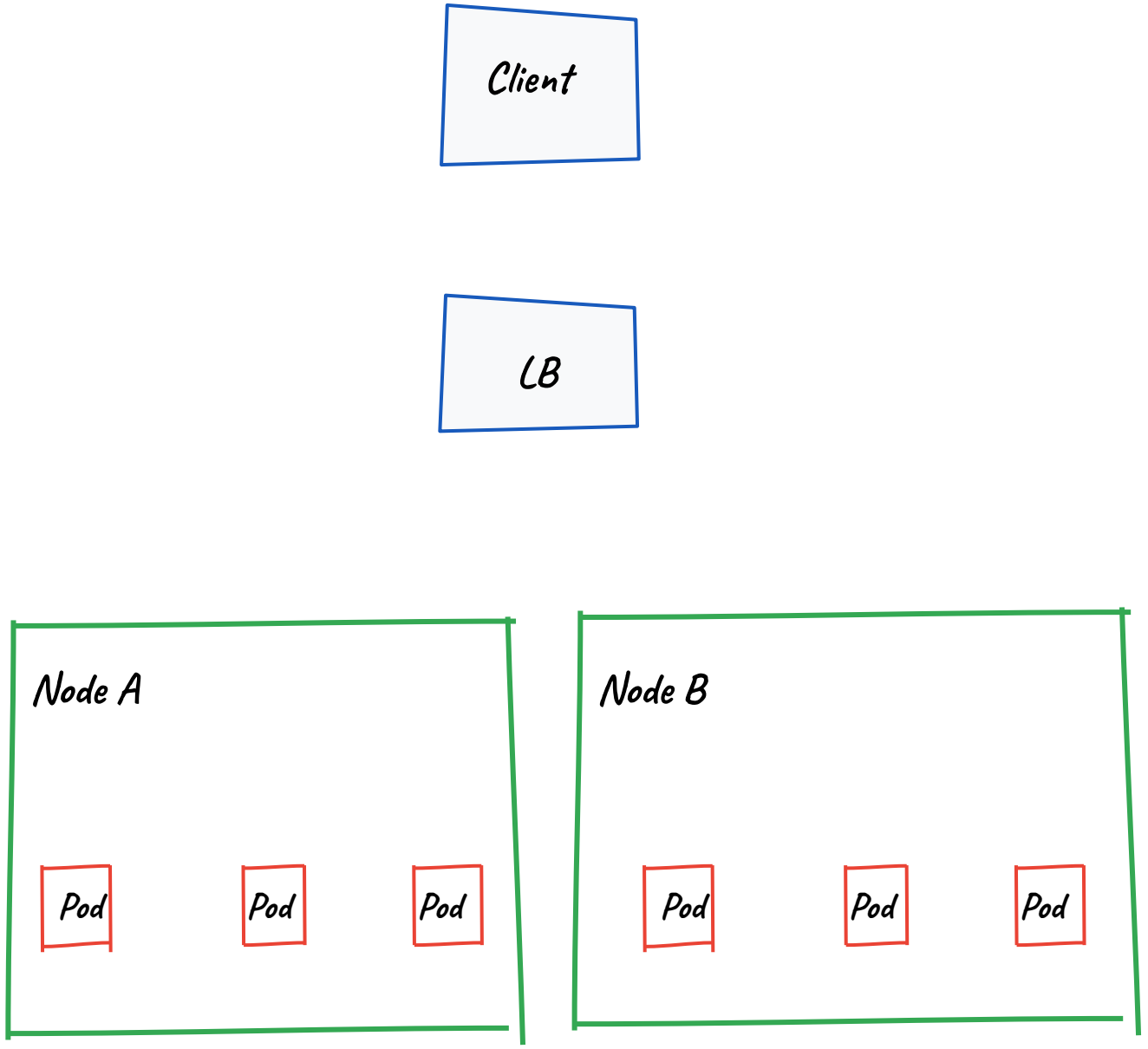
- Two levels of load balancing
- Inaccurate cloud health checks
- Inaccurate Load Balancing
- Multiple Network hops
- Loss of LB features



Google Cloud

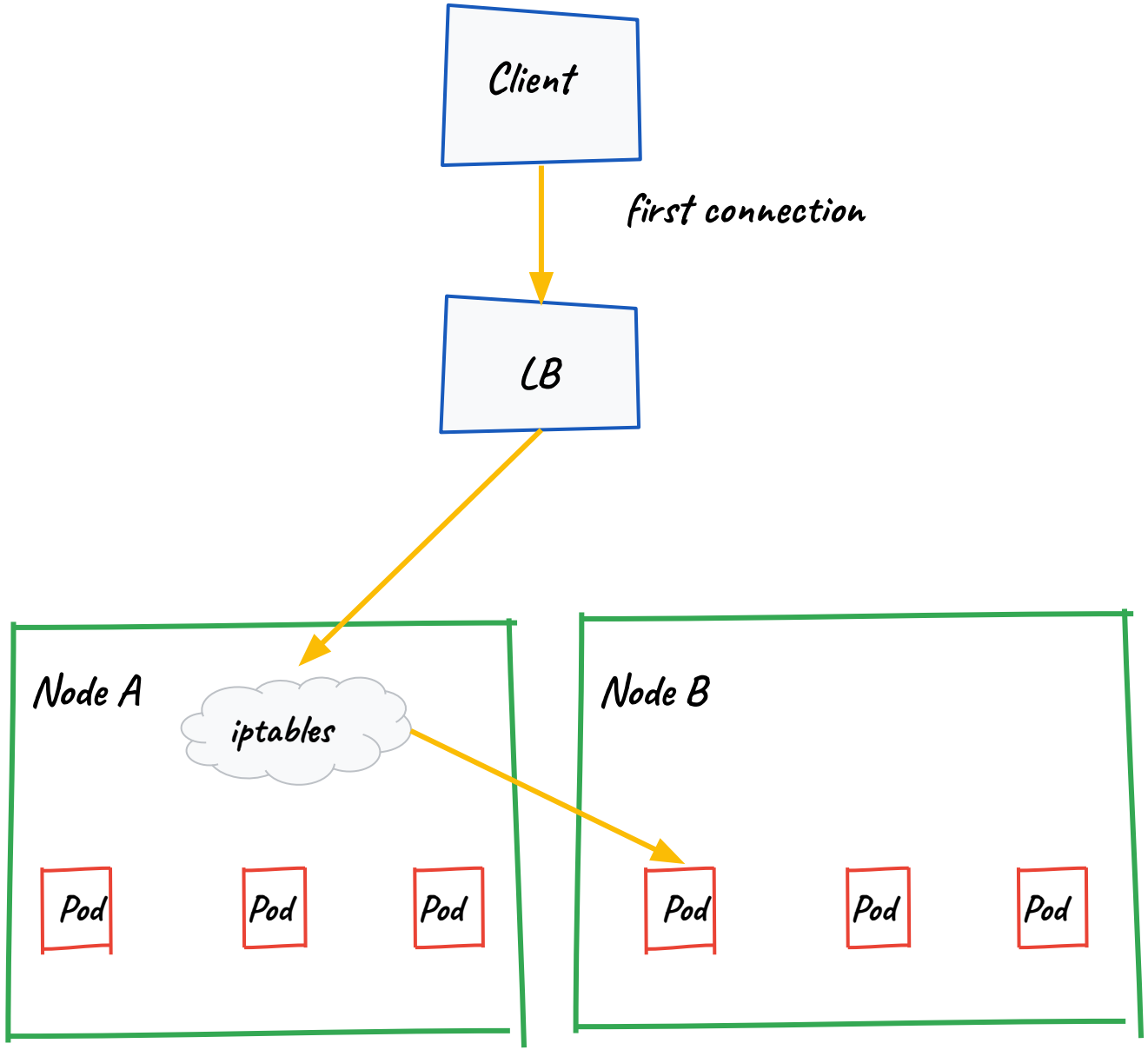
Example: Cookie Affinity

- A feature of GCP's HTTP LB
- LB returns a cookie to client
- Ensures repeated connections go to same backend



Example: Cookie Affinity

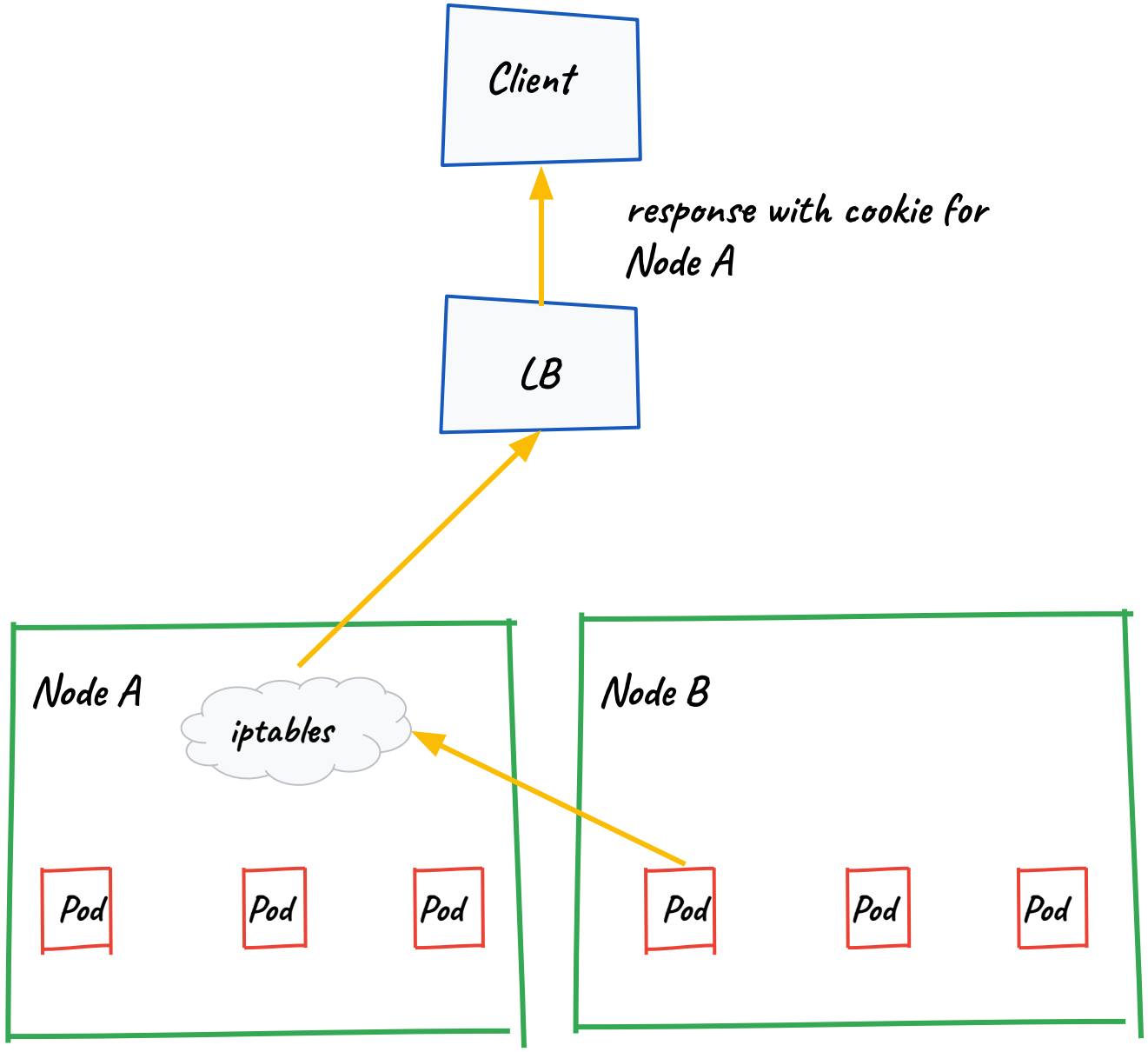
- A feature of GCP's HTTP LB
- LB returns a cookie to client
- Ensures repeated connections go to same backend



Google Cloud

Example: Cookie Affinity

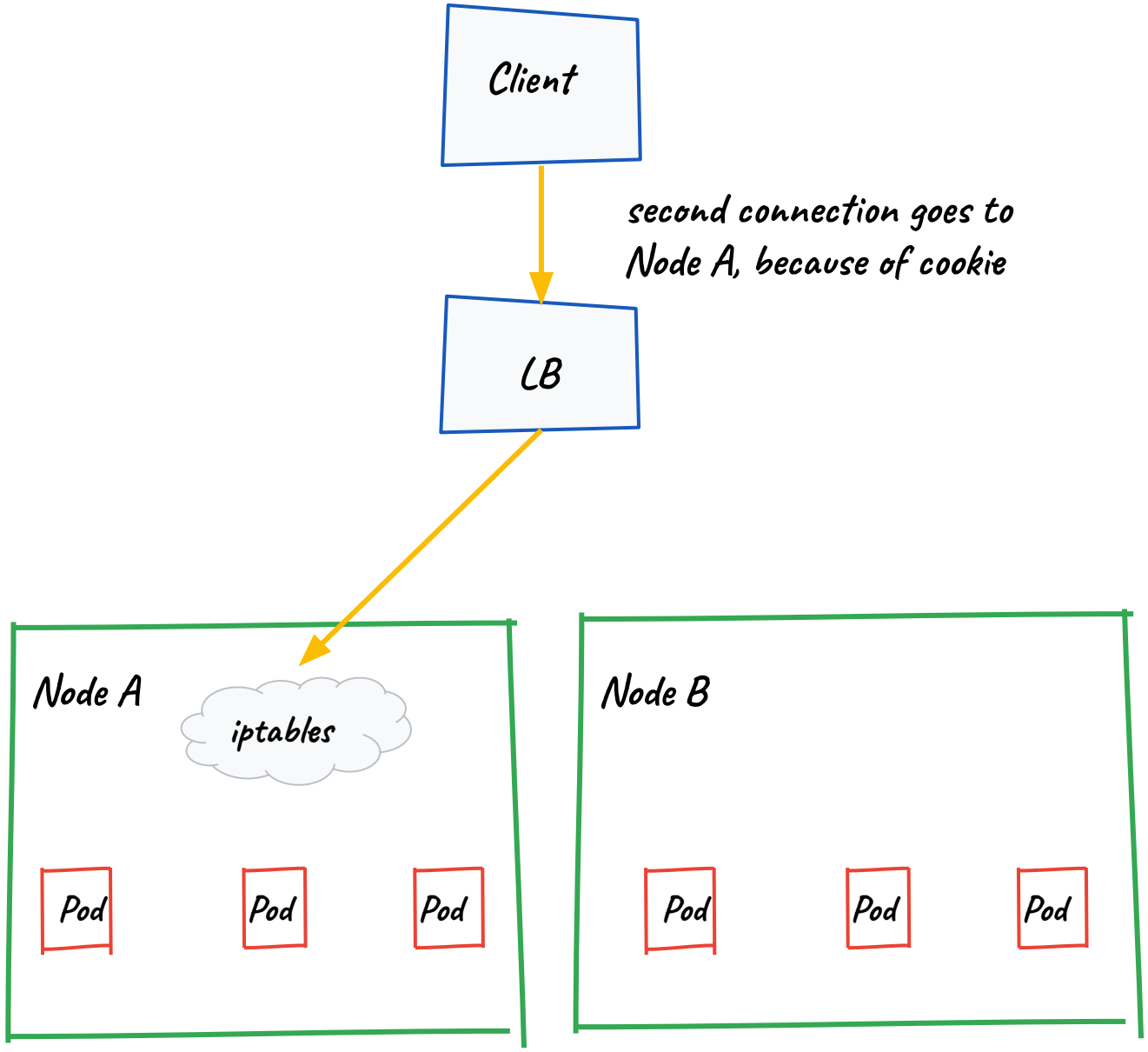
- A feature of GCP's HTTP LB
- LB returns a cookie to client
- Ensures repeated connections go to same backend



Google Cloud

Example: Cookie Affinity

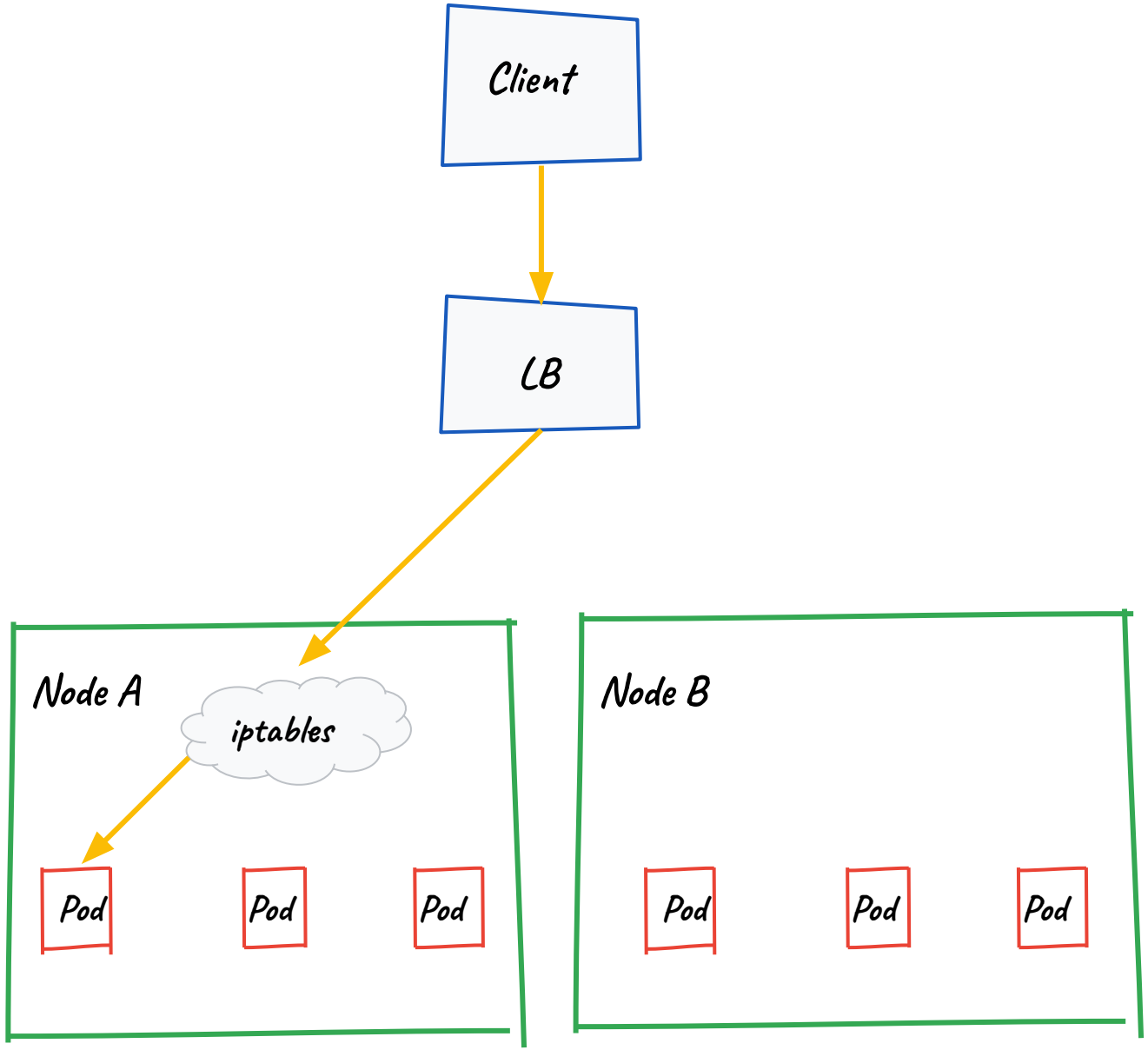
- A feature of GCP's HTTP LB
- LB returns a cookie to client
- Ensures repeated connections go to same backend



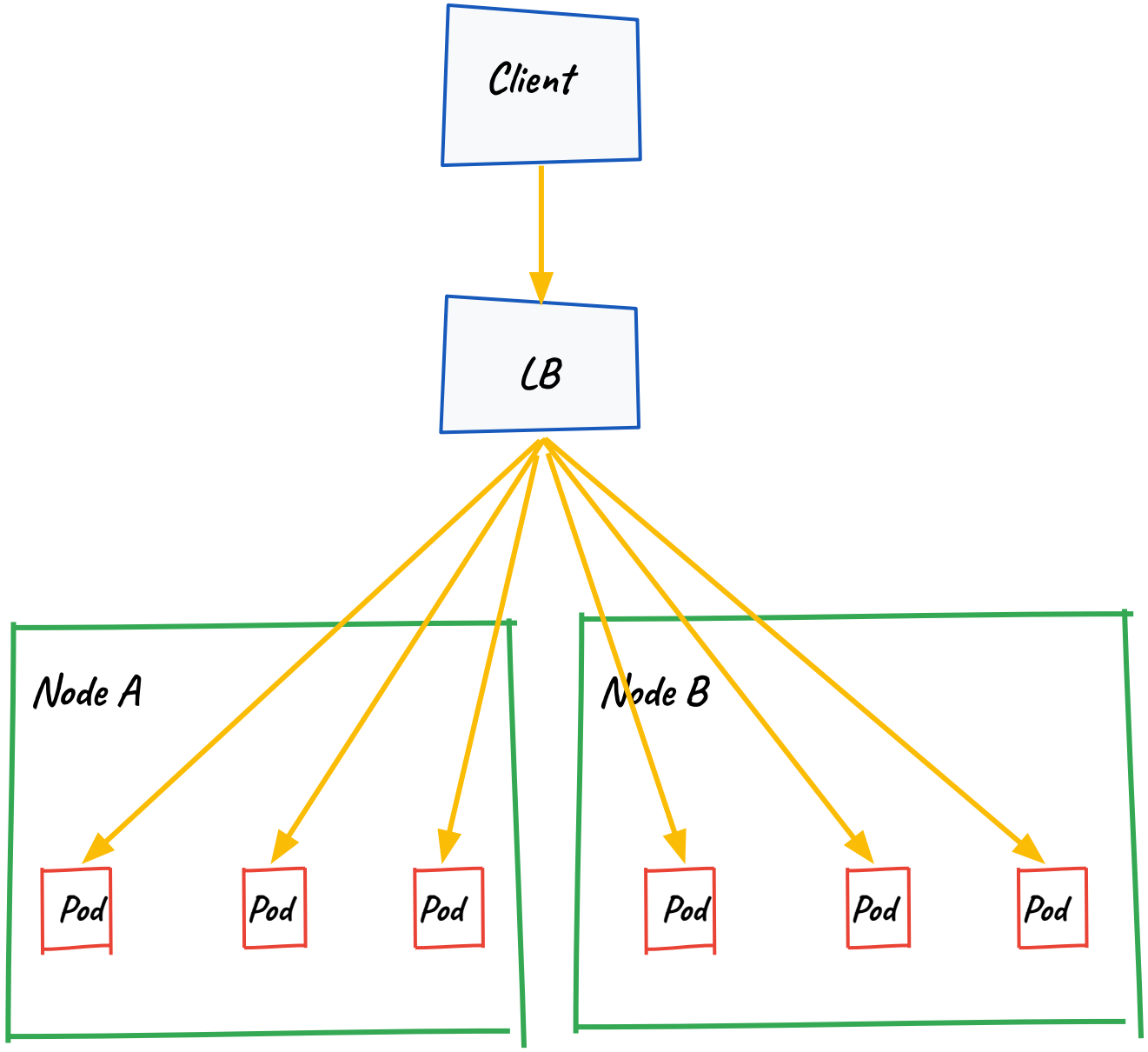
Google Cloud

Example: Cookie Affinity

- A feature of GCP's HTTP LB
- LB returns a cookie to client
- Ensures repeated connections go to same backend
- Second hop is not cookie-aware

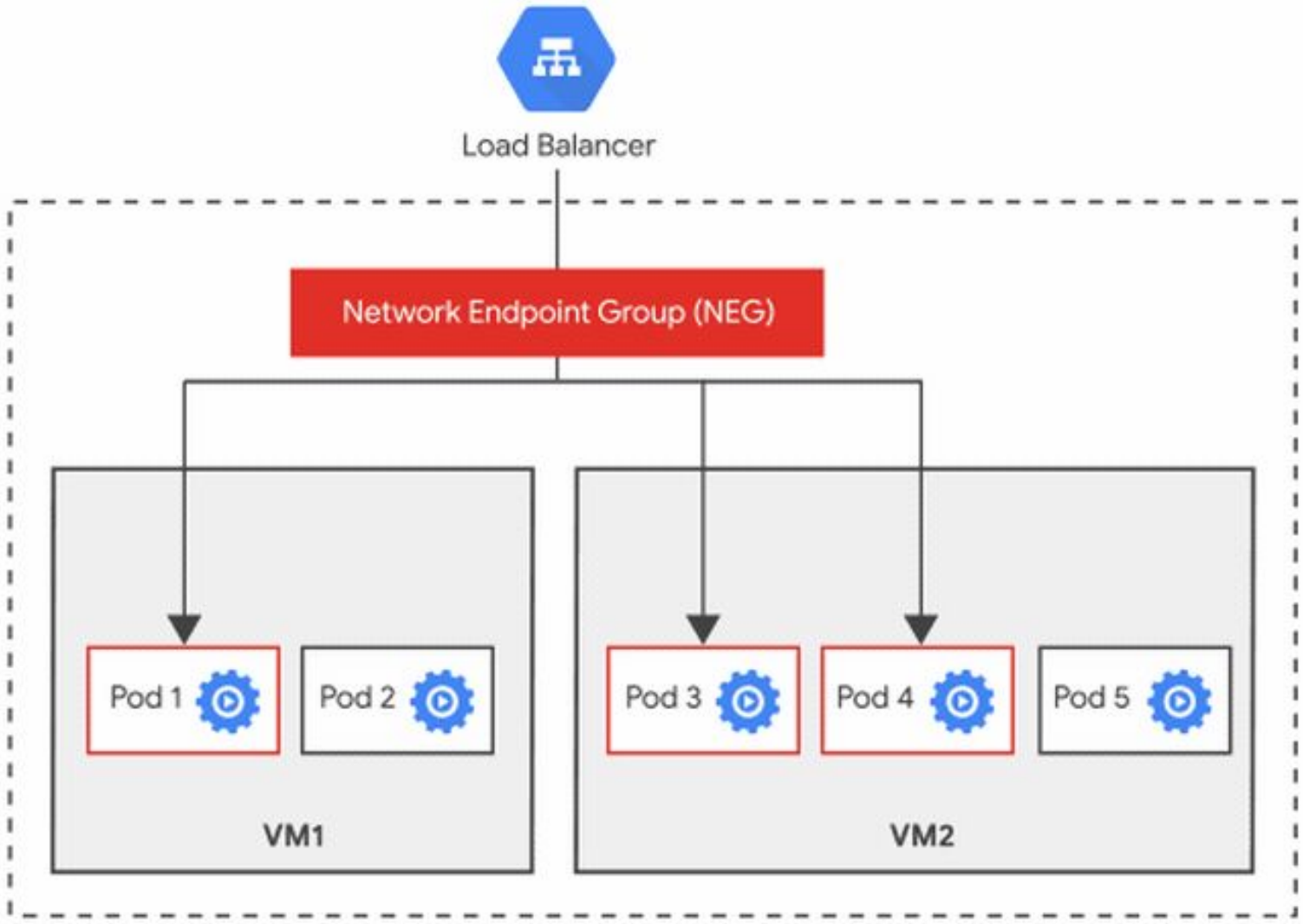


Why can't we load balance to Pod IPs?



Network Endpoint Groups in GCE LB

- Now HTTP LB can target pod IPs, not just VMs
- Features like cookie affinity “Just Work”
- Balances the load without downsides of a second hop



Google Cloud

Containers as first Class GCP SDN endpoints

Alias IPs made Pods as first class endpoint on VPC

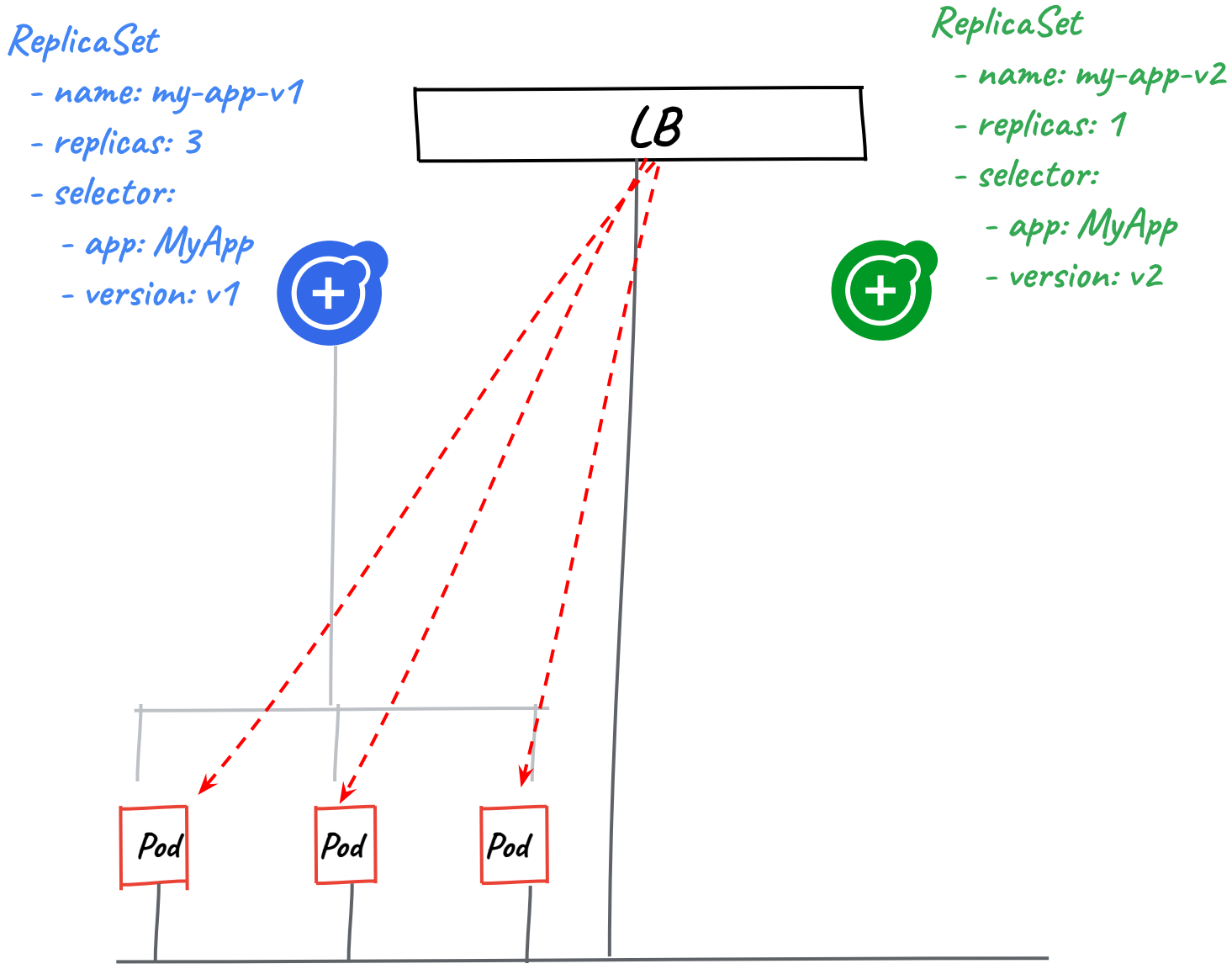
Network endpoint groups made load balancing for containers as efficient and feature rich as VMs

Problems when load-balancing to Pods

Programming external LBs is slower than iptables

Possible to cause an outage by rolling update going faster than LB

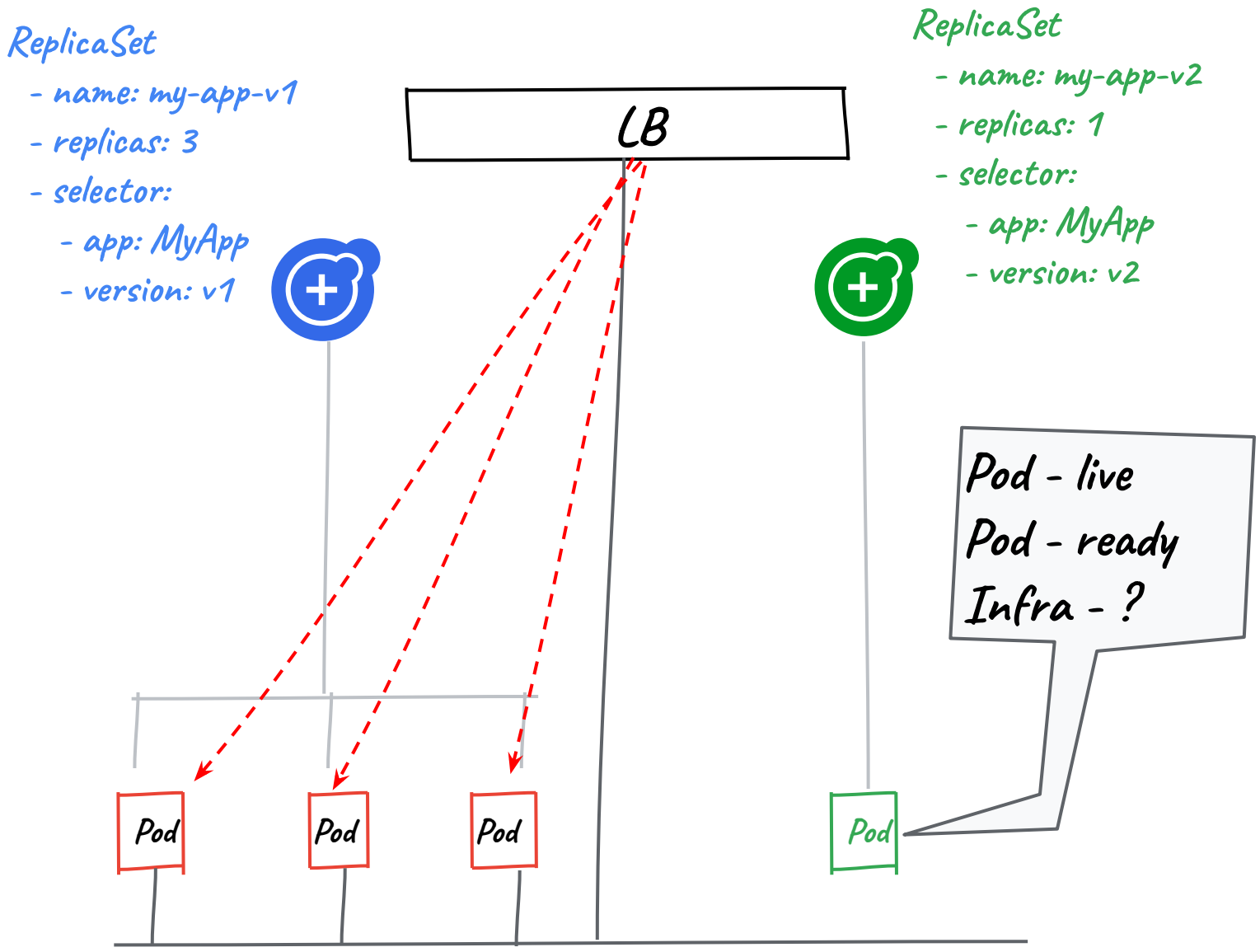
Rolling Update



Google Cloud

Rolling Update

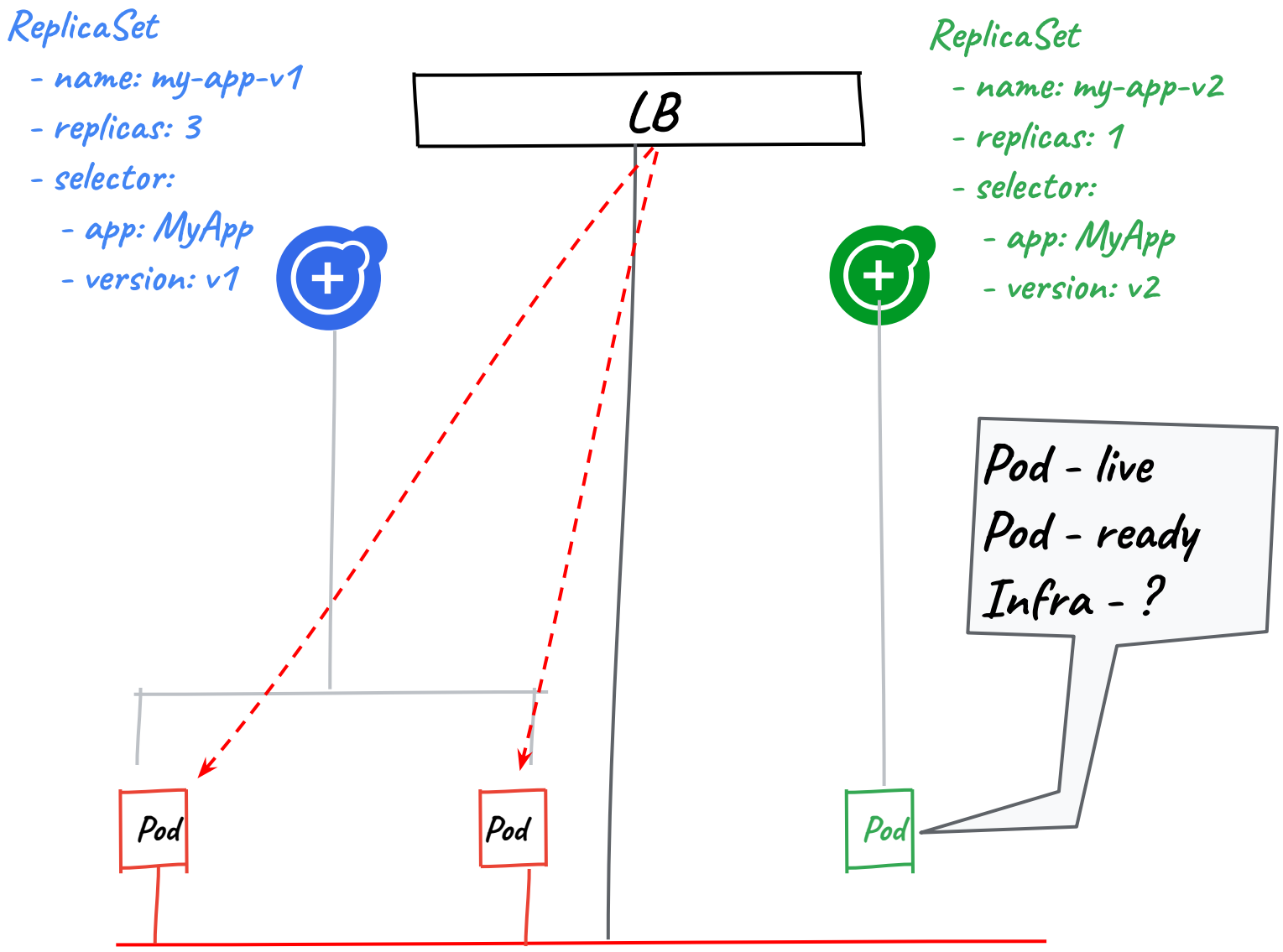
- Pod Liveness : state of application in pod -a live or not
- Pod Readiness : ready to receive traffic



Google Cloud

Wait for Infrastructure?

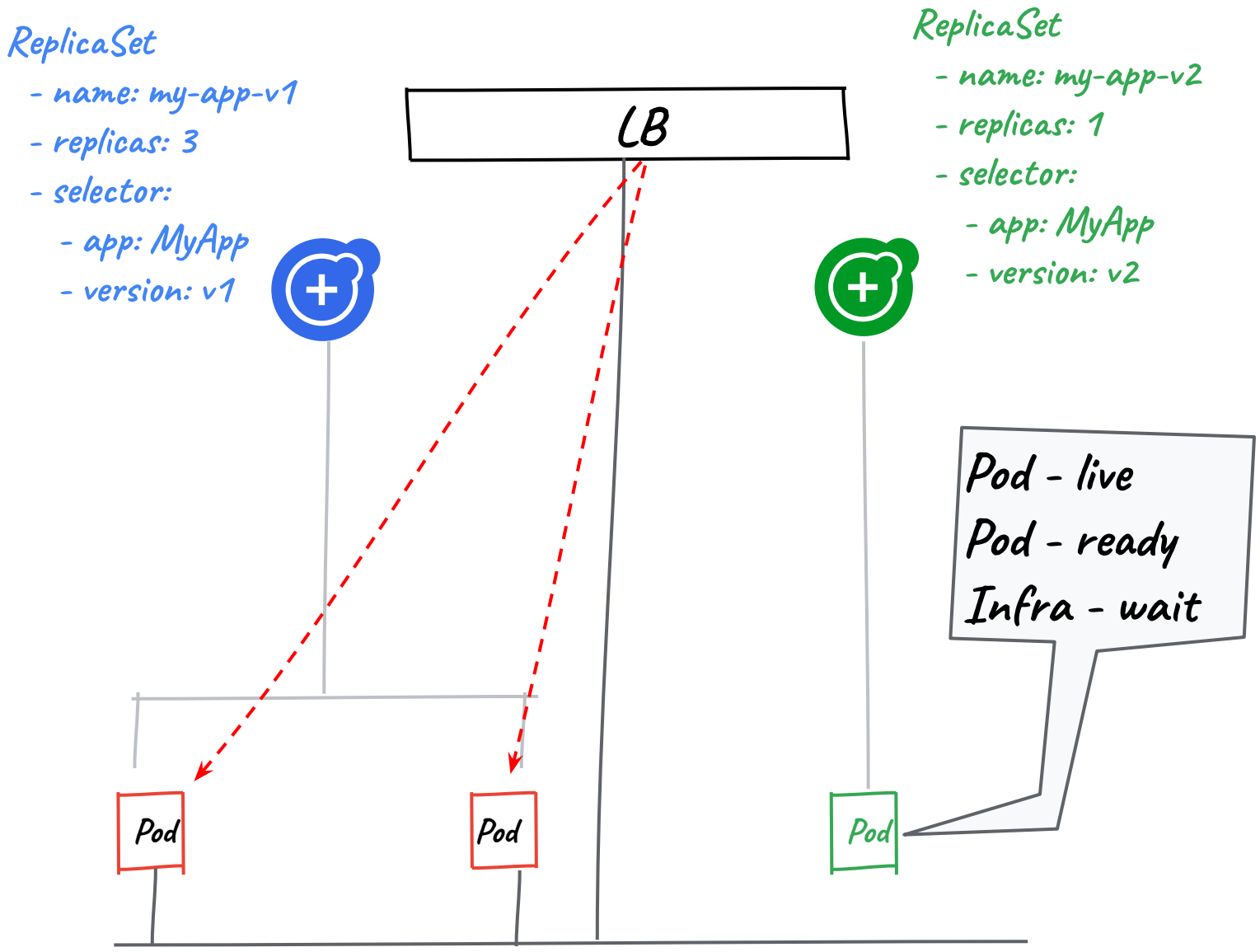
- LB not programmed but Pod reports ready
- Pod from previous replicaset removed.
- Capacity reduced !.



Google Cloud

Pod Ready ++

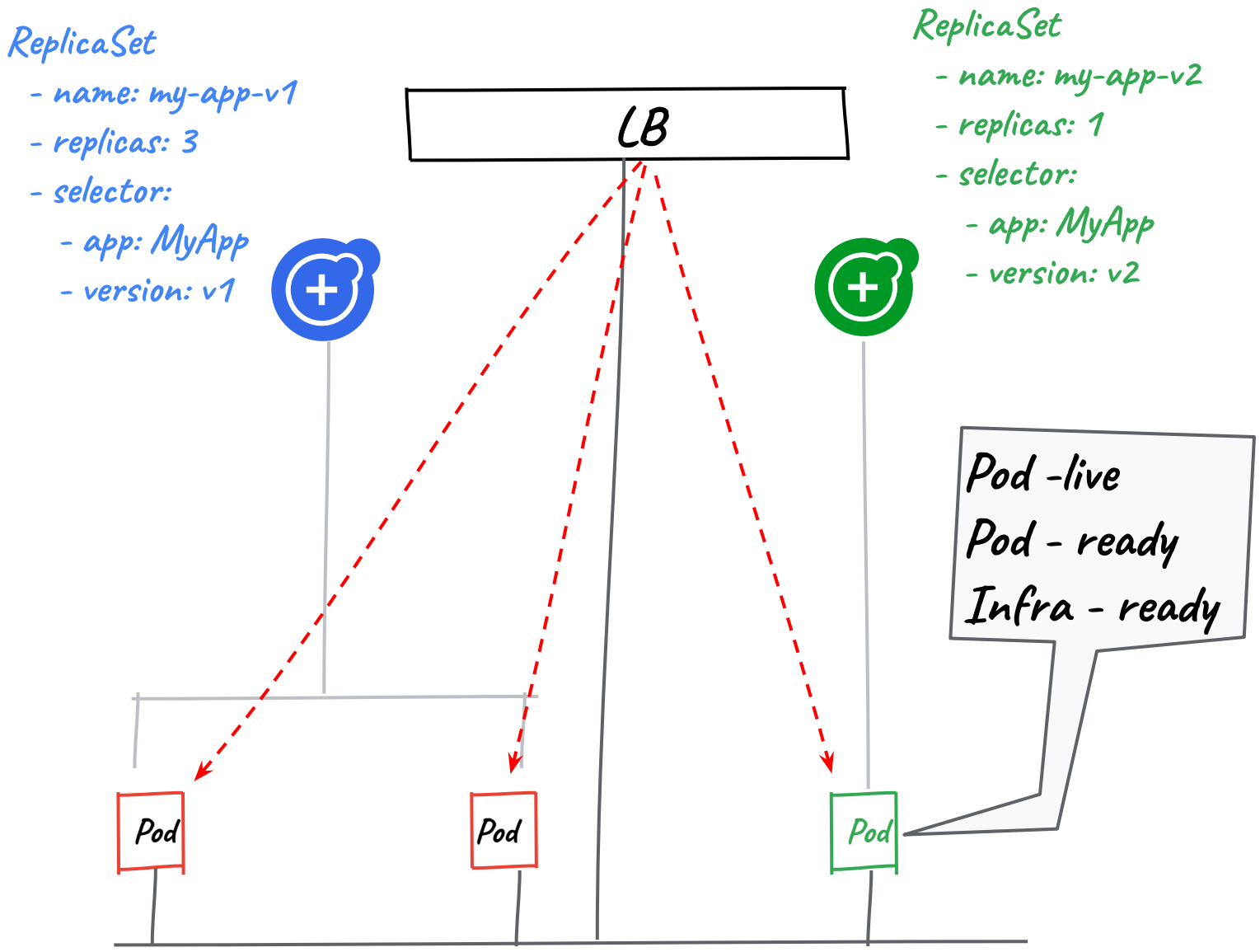
- New state in Pod life cycle to wait - Pod Ready ++



Google Cloud

Pod Ready ++

- New state in Pod life cycle to wait - Pod Ready ++



Google Cloud

What about all the features?

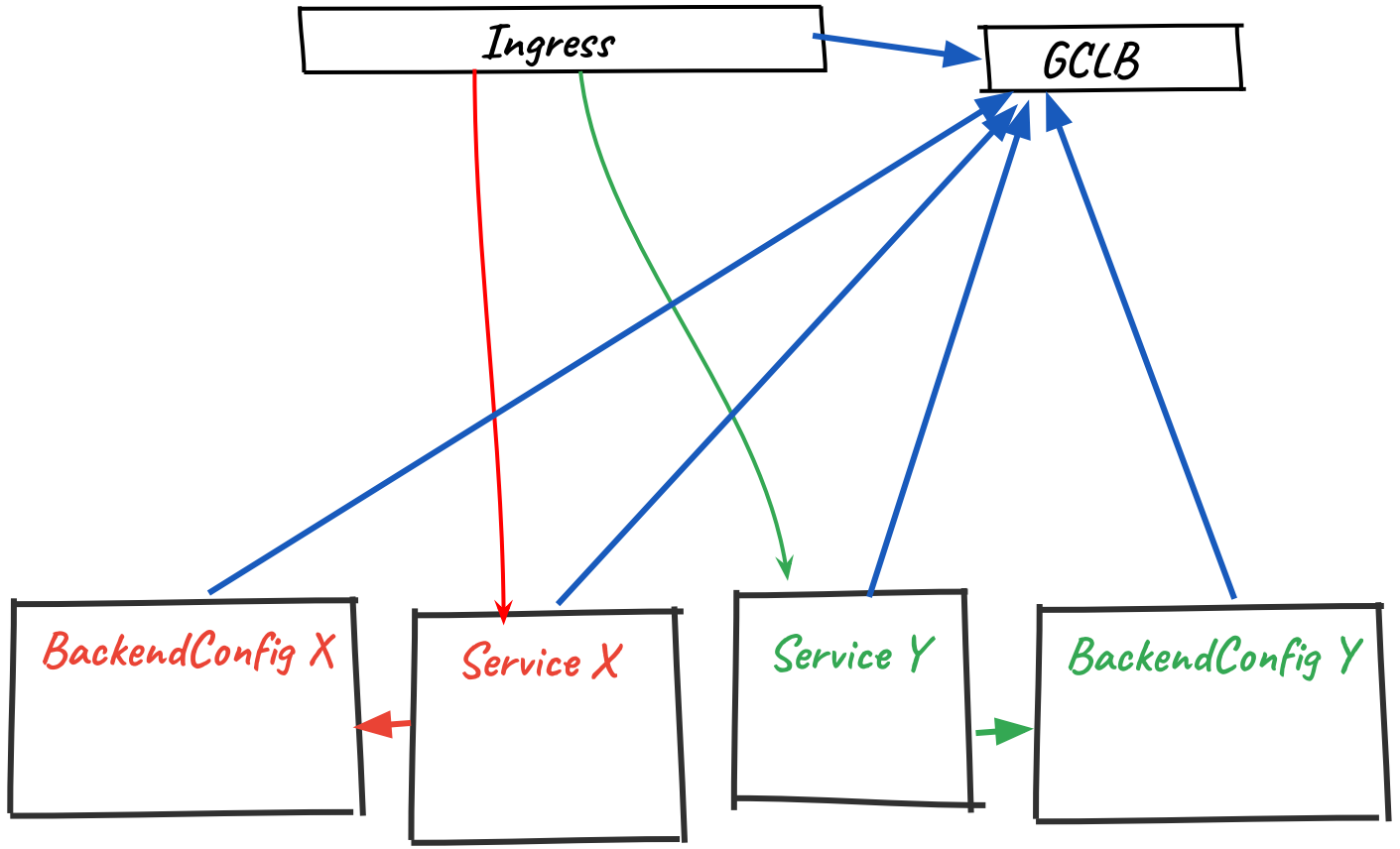
Every LB has features not expressed by Kubernetes

Principle: Most implementations must be able to support most features

Express GCP's LB features

- CRD to the rescue
  - Linked from Service
  - Implementation specific

- BackendConfig
  - Allows us to expose features to GCP users without bothering anyone else



BackendConfig

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  annotations:
    beta.cloud.google.com/backend-config:
      '{"ports": {"http":"config-http"}}'
spec:
  type: NodePort
  selector:
    app: my-app
  ports:
  - name: http
    port: 80
    targetPort: 8080
```

```
apiVersion: cloud.google.com/v1beta1
kind: BackendConfig
metadata:
  name: config-http
spec:
  cdn:
    enabled: true
    cachePolicy:
      includeHost: true
      includeProtocol: true
  iap:
    enabled: false
  timeoutSec: 5
  sessionAffinity:
    affinityType: GENERATED_COOKIE
    affinityCookieTtlSec: 180
```

Mistakes in Abstractions?

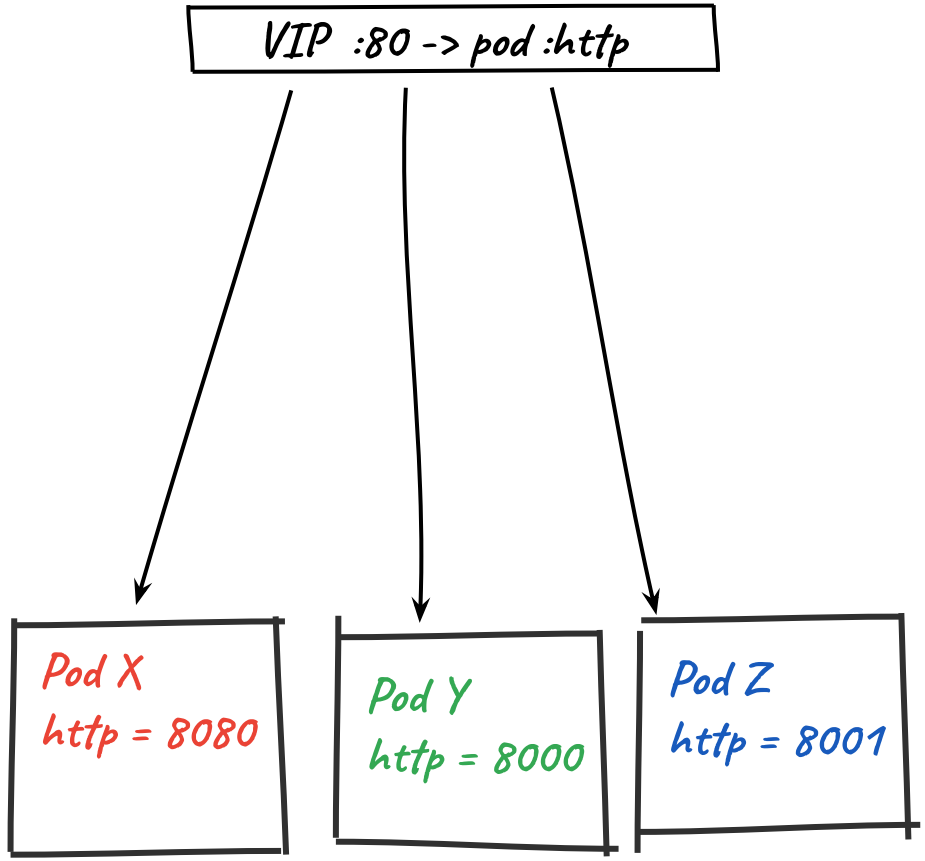
Too Flexible?

Not Flexible Enough?

Too Monolithic?

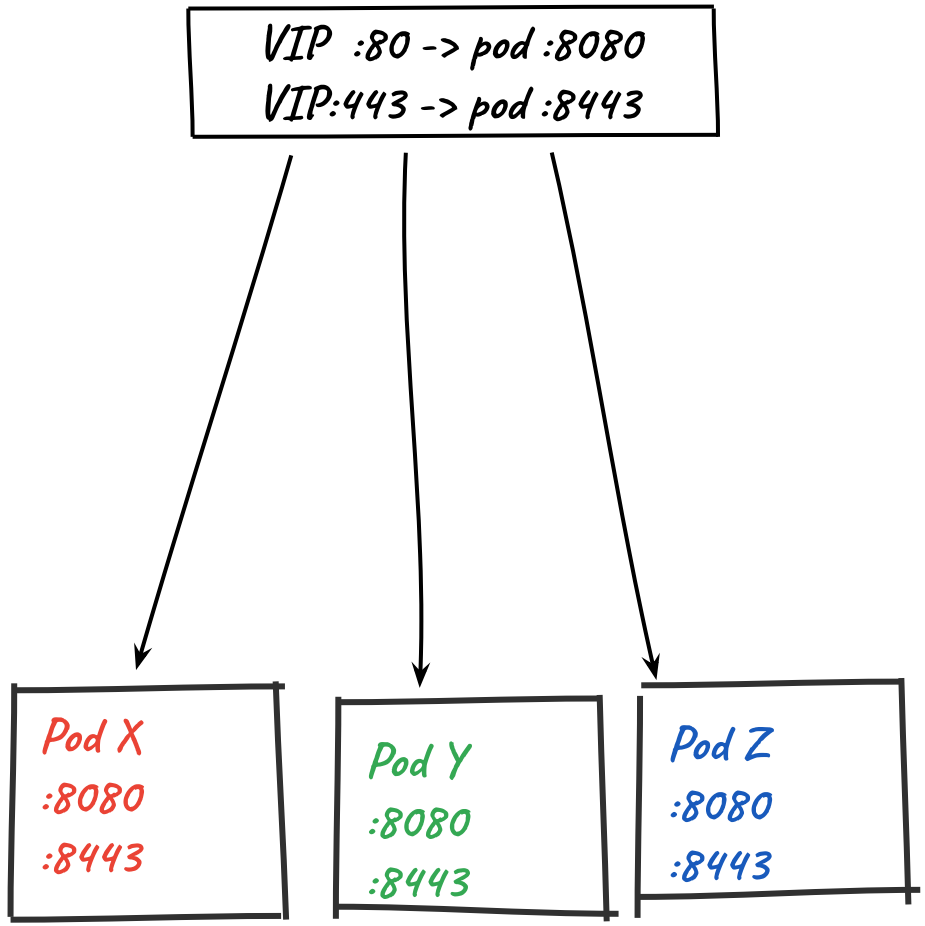
Too flexible?

- Service is a very flexible abstraction
  - Target ports
  - Named ports
- Makes it hard to implement in some fabrics
  - DSR is incompatible with port remapping
- Inspired by docker's port-mapping model
- Hindsight: should probably have made it simpler



Not flexible enough?

- Service is not flexible enough in other ways
  - Can't forward ranges
  - Can't forward a whole IP
- Makes it hard for some apps to use services
  - Dynamic ports
  - Large numbers of ports



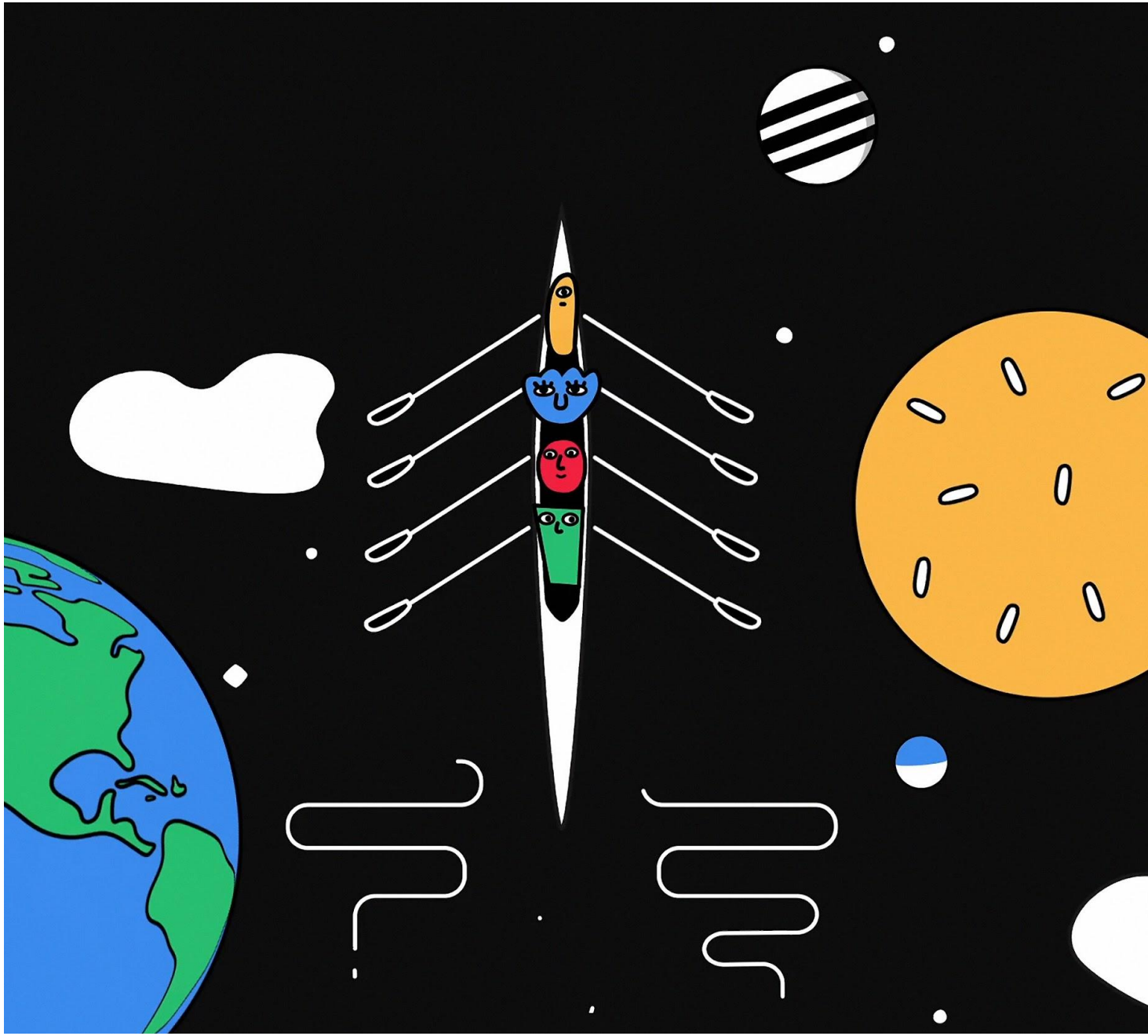
Too monolithic?

- Service API is monolithic and complex
  - `type` field does not capture all variants
  - Headless vs VIP
  - Selector vs manual

- External LB support is built-in but primitive
  - Should have had readiness gates long ago
  - No meaningful status

Google Cloud

Looking ahead



Google Cloud

Want more?

Come to the SIG-Network Intro & Deep-Dive on Thursday!

Thank You!

Purvi Desai
@purvid

Tim Hockin
@thockin

Google Cloud