Kubernetes the Very Hard Way

Laurent Bernaille

Staff Engineer, Infrastructure

@lbernail

DATADOG

Datadog

Over 350 integrations
Over 1,200 employees
Over 8,000 customers
Runs on millions of hosts
Trillions of data points per day

10000s hosts in our infra
10s of k8s clusters with 50-2500 nodes
Multi-cloud
Very fast growth

lbernail

Why Kubernetes?

Dogfooding

Improve k8s integrations

Immutable

Move from Chef

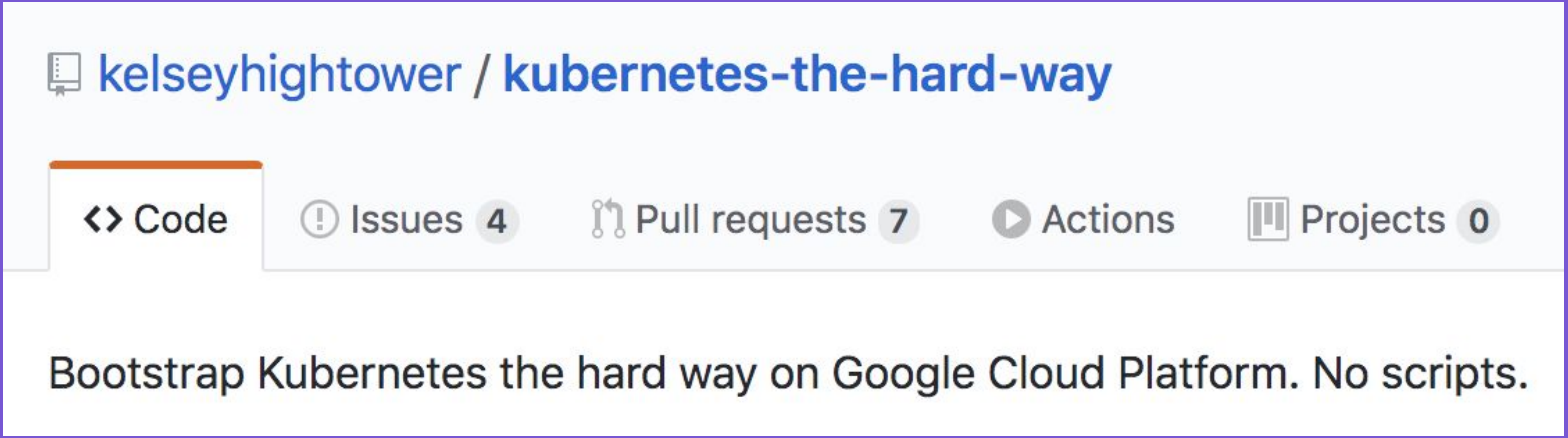
Multi Cloud

Common API

Community

Large and Dynamic

The very hard way?



It was *much* harder

This talk is about the fine prints

"Of course, you will need a HA master setup"

"Oh, and yes, you will have to manage your certificates"

"By the way, networking is slightly more complicated, look into CNI / ingress controllers"

What happens after “Kube 101”

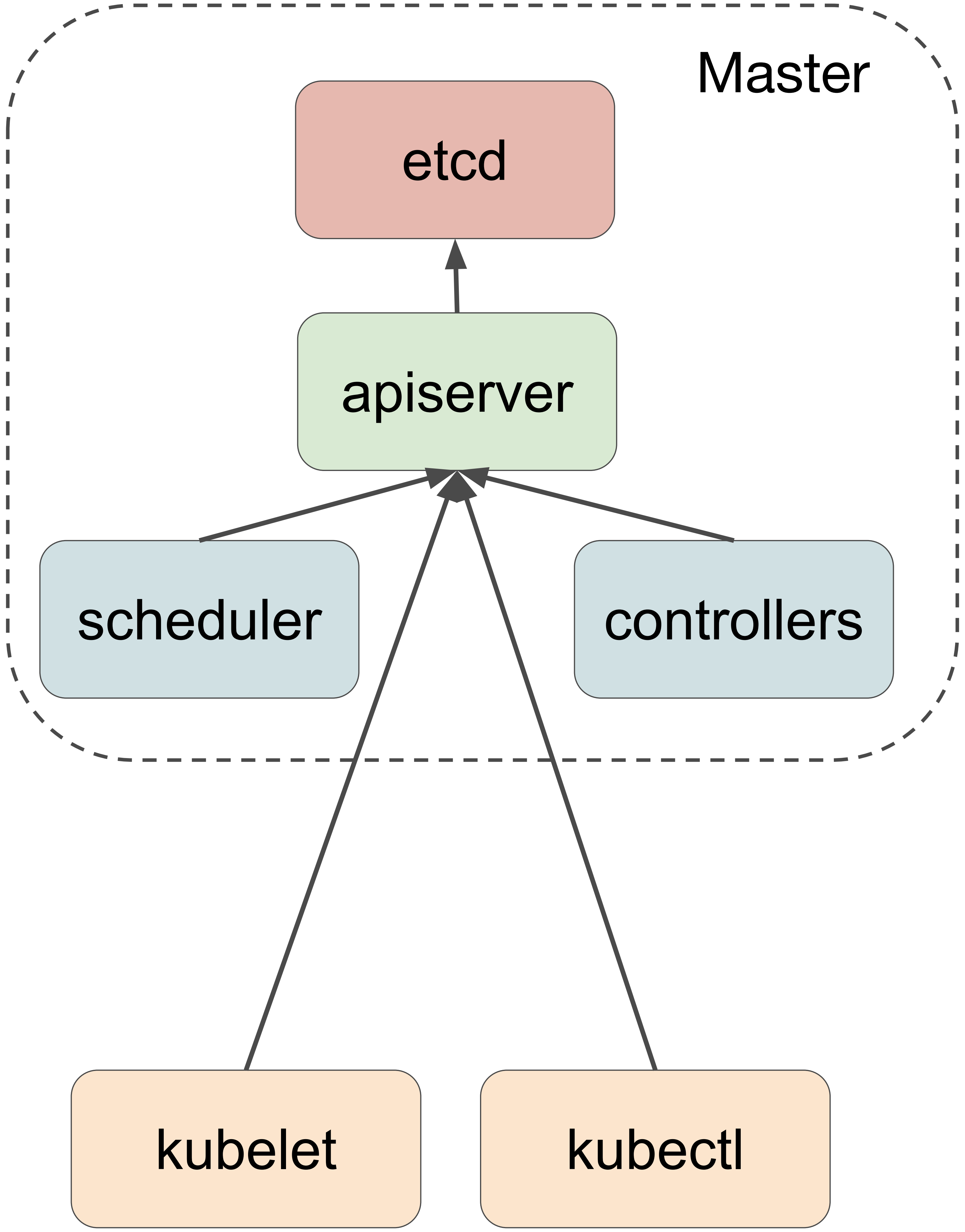
1. Resilient and Scalable Control Plane
2. Securing the Control Plane
    a. Kubernetes and Certificates
    b. Exceptions?
    c. Impact of Certificate Rotation
3. Efficient networking
    a. Giving pod IPs and routing them
    b. Accessing services: Client-side load-balancing:
    c. Ingresses: Getting data in the cluster

What happens after “Kube 101”

1. Resilient and Scalable Control Plane
2. Securing the Control Plane
    a. Kubernetes and Certificates
    b. Exceptions?
    c. Impact of Certificate Rotation
3. Efficient networking
    a. Giving pod IPs and routing them
    b. Accessing services: Client-side load-balancing:
    c. Ingresses: Getting data in the cluster

Resilient and Scalable Control Plane

Kube 101 Control Plane



lbernail

Making it resilient

lbernail

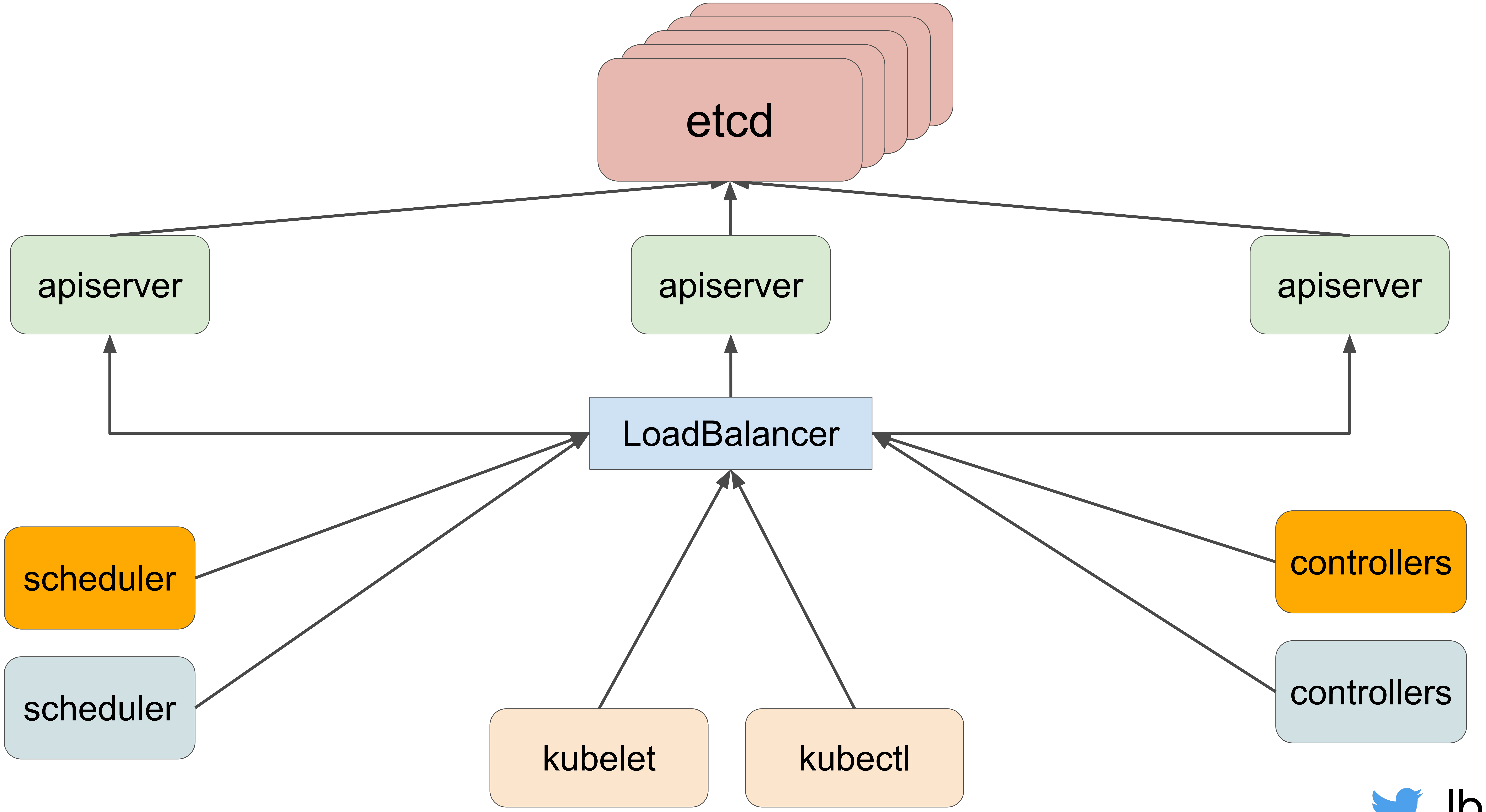
Separate etcd nodes

lbernail

Single active Controller/scheduler

lbernail

Split scheduler/controllers



lbernail

What happens after "Kube 101"

1. Resilient and Scalable Control Plane
2. Securing the Control Plane
    a. Kubernetes and Certificates
    b. Exceptions?
    c. Impact of Certificate Rotation
3. Efficient networking
    a. Giving pod IPs and routing them
    b. Accessing services: Client-side load-balancing:
    c. Ingresses: Getting data in the cluster

lbernail

Kubernetes and Certificates

From “the hard way”

```
cat > ca-config.json <<EOF
{
  "signing": {
    "default": {
      "expiry": "8760h"
    },
    "profiles": {
      "kubernetes": {
        "usages": ["signing", "key encipherment", "server auth", "client auth"],
        "expiry": "8760h"
      }
    }
  }
}
```

“Our cluster broke after ~1y”

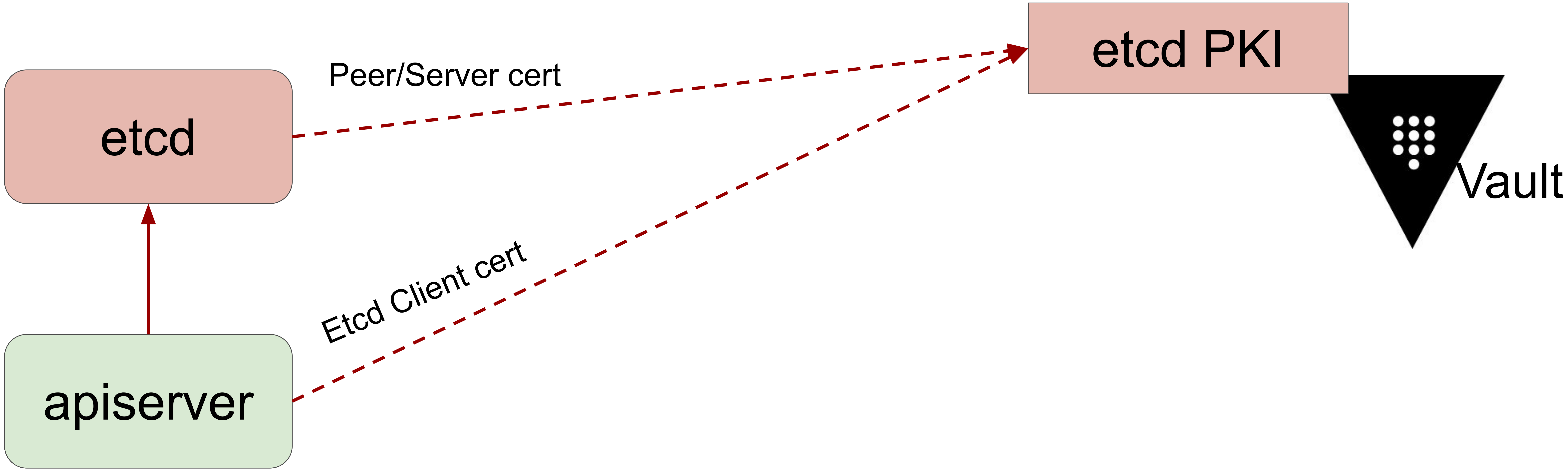
```
cat > ca-config.json <<EOF
{
  "signing": {
    "default": {
      "expiry": "8760h"
    },
    "profiles": {
      "kubernetes": {
        "usages": ["signing", "key encipherment", "server auth", "client auth"],
        "expiry": "8760h"
      }
    }
  }
}
```

lbernail

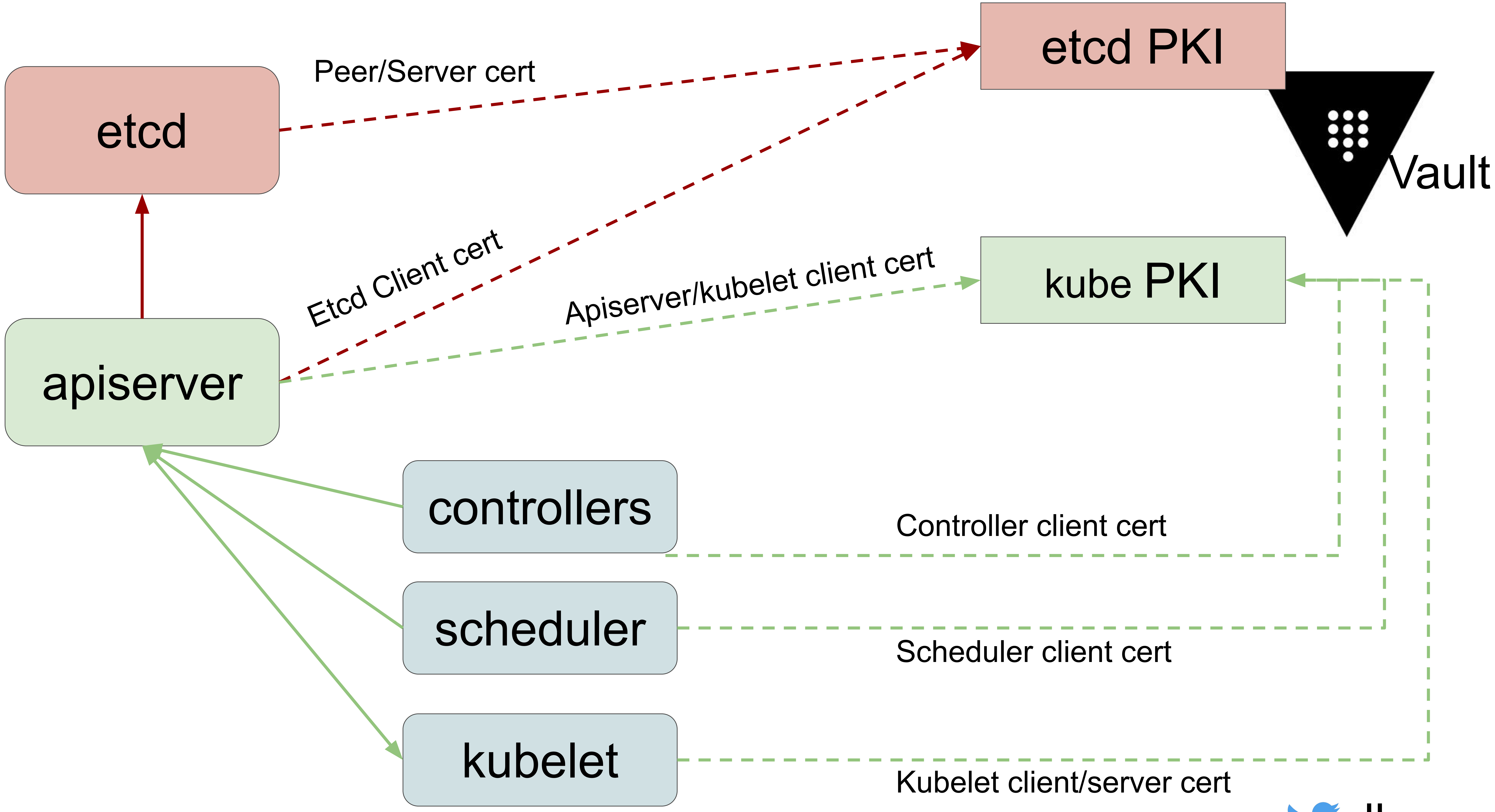
Certificates in Kubernetes

- Kubernetes uses certificates everywhere
- Very common source of incidents
- Our Strategy: Rotate all certificates daily

Certificate management

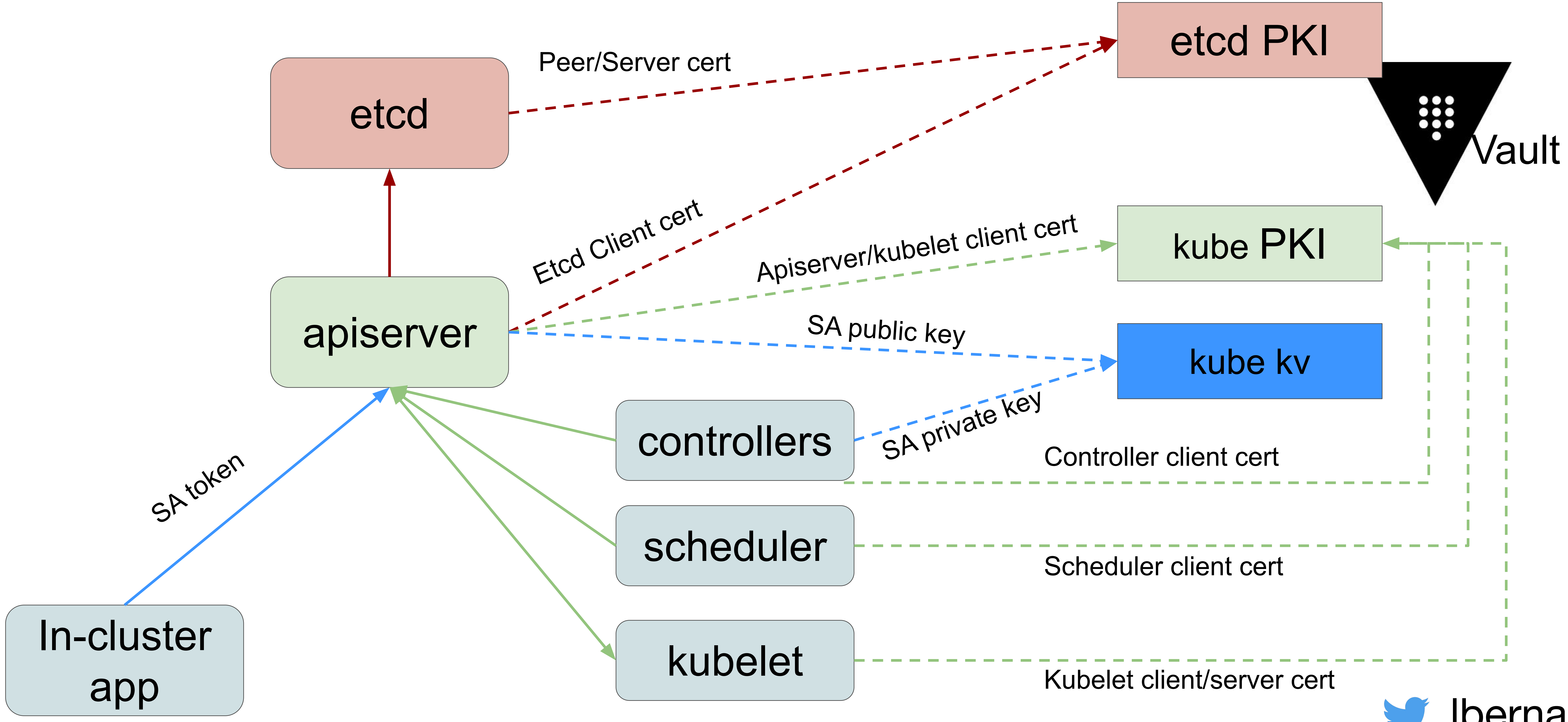


Certificate management



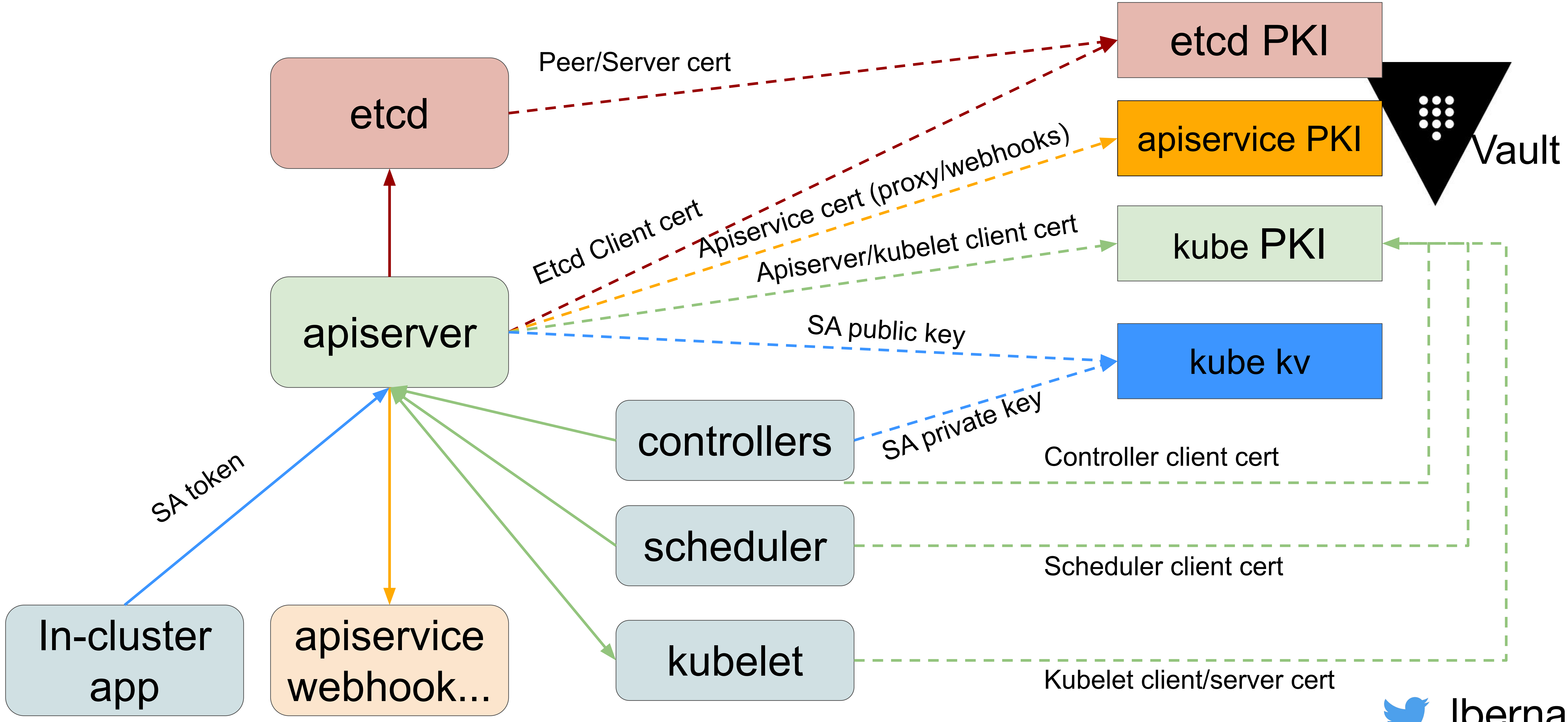
lbernail

Certificate management



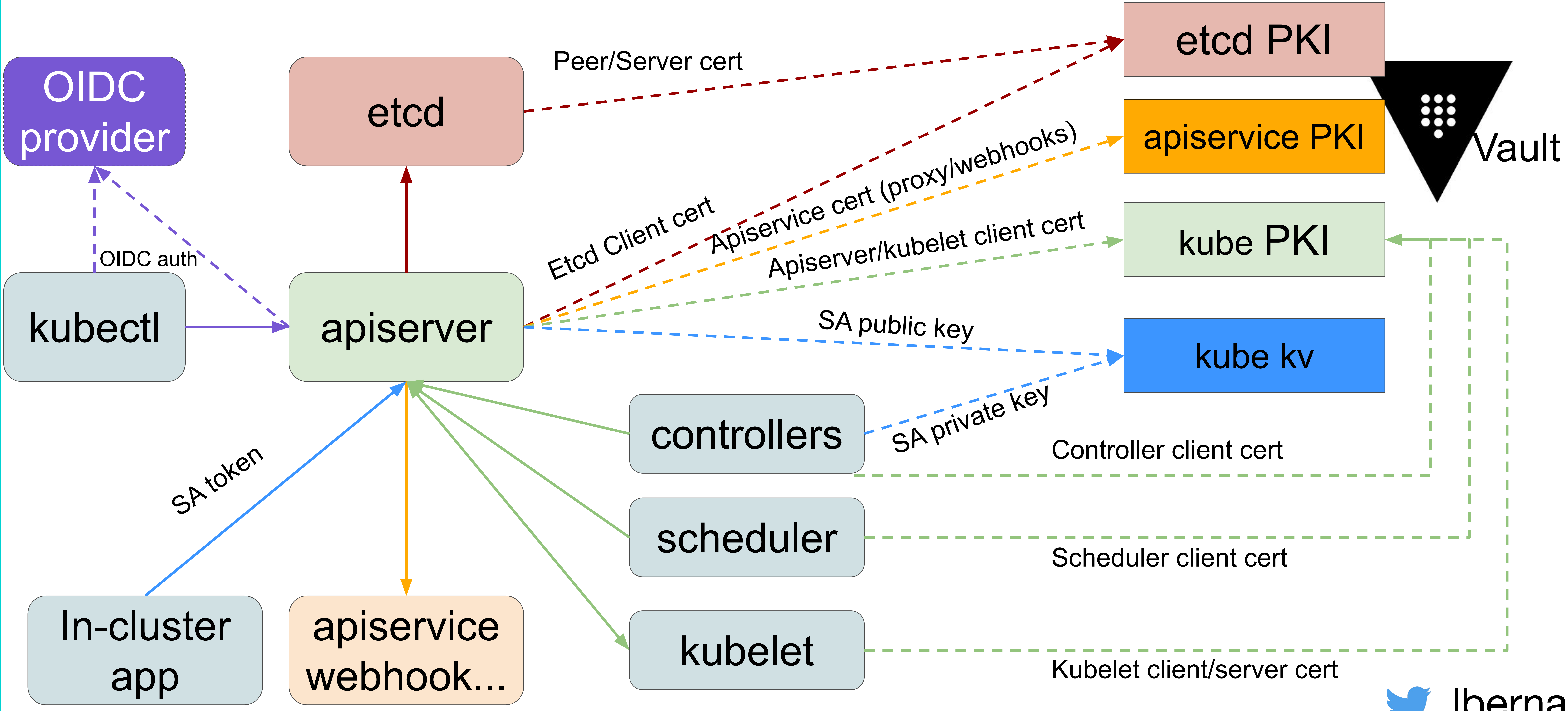
lbernail

Certificate management



lbernail

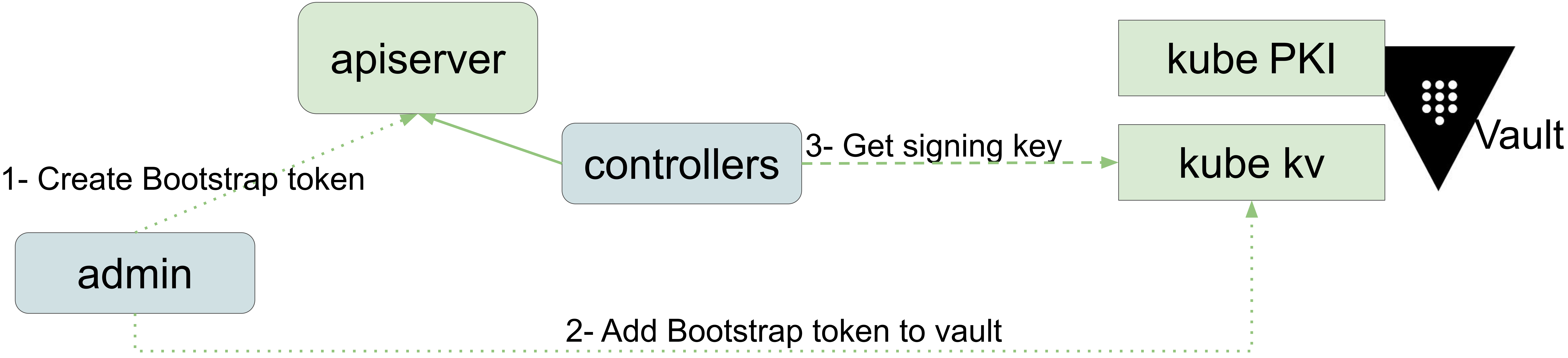
Certificate management



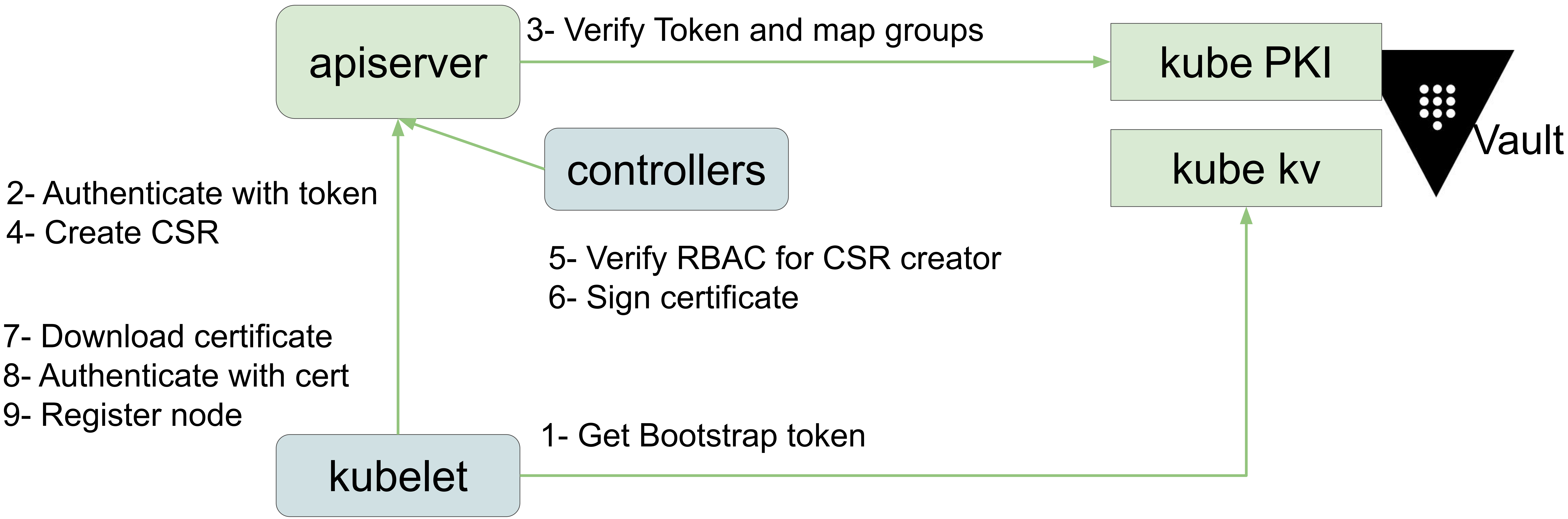
lbernail

Exception ?
Incident…

Kubelet: TLS Bootstrap



Kubelet: TLS Bootstrap

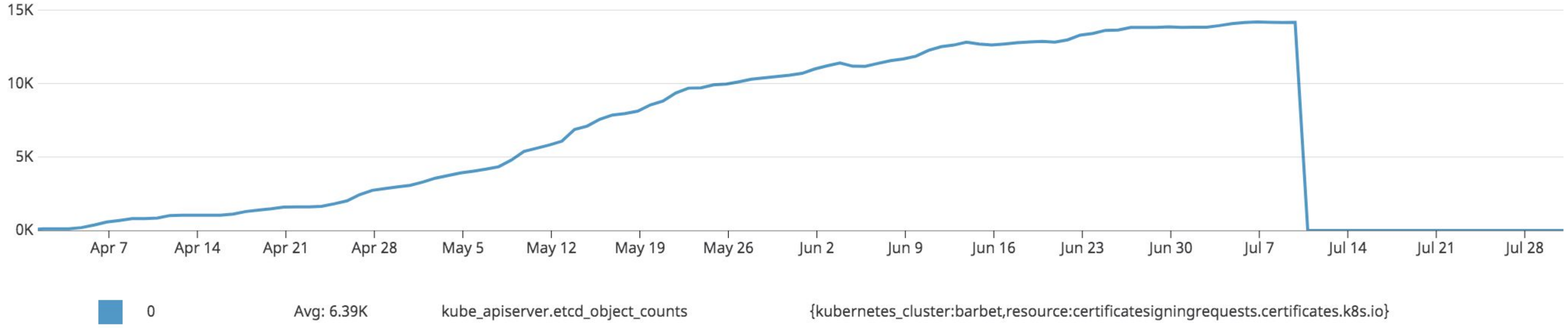


lbernail

Kubelet certificate issue

1. One day, some Kubelets were failing to start or took 10s of minutes
2. Nothing in logs
3. Everything looked good but they could not get a cert
4. Turns out we had a lot of CSRs in flight
5. Signing controller was having a hard time evaluating them all

CSR resources in the cluster
Lower is better!



lbernail

Why?

Initial creation

1. Authenticate with bootstrap token, mapped to group "system:bootstrappers"
2. Create CSR
3. "system:bootstrappers" has role "system:certificates.k8s.io:certificatesigningrequests:nodeclient"

Renewal

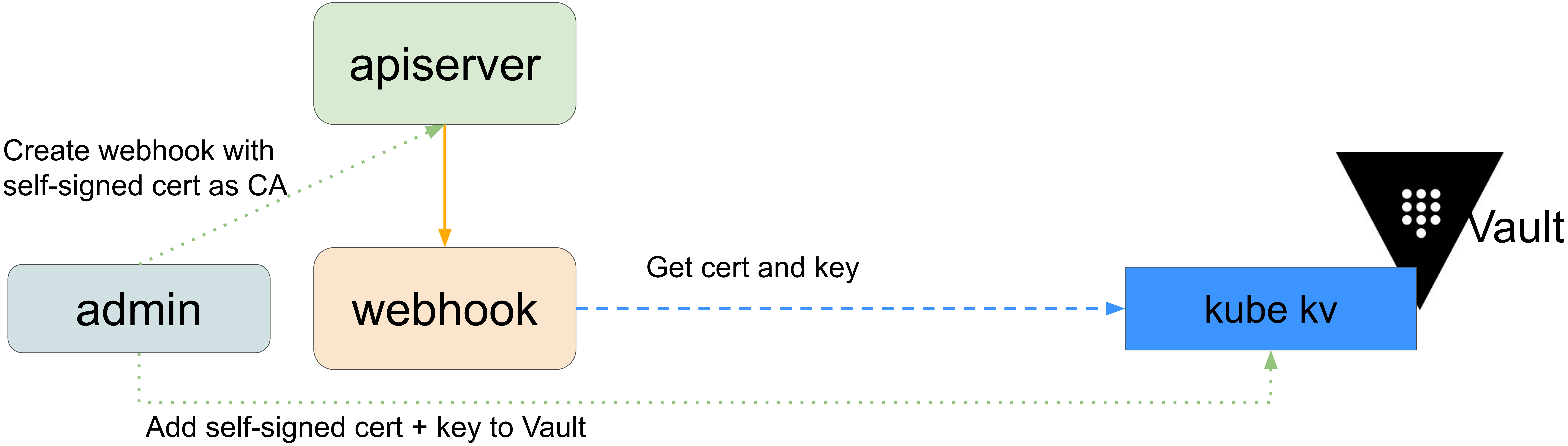
1. Authenticate with current node certificate, mapped to group "system:nodes"
2. Create CSR
3. Not allowed for auto-sign

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: bootstrap:auto-approve-selfnodeclient-csrs
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:certificates.k8s.io:certificatesigningrequests:selfnodeclient
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:bootstrappers
```

Also needed for "system:nodes"

Exception 2?
Incident 2…

Temporary solution



One day, after ~1 year

- Creation of resources started failing (luckily only a Custom Resource)
- Cert had expired...

Take-away

- Rotate server/client certificates
- Not easy

But, “If it’s hard, do it often”

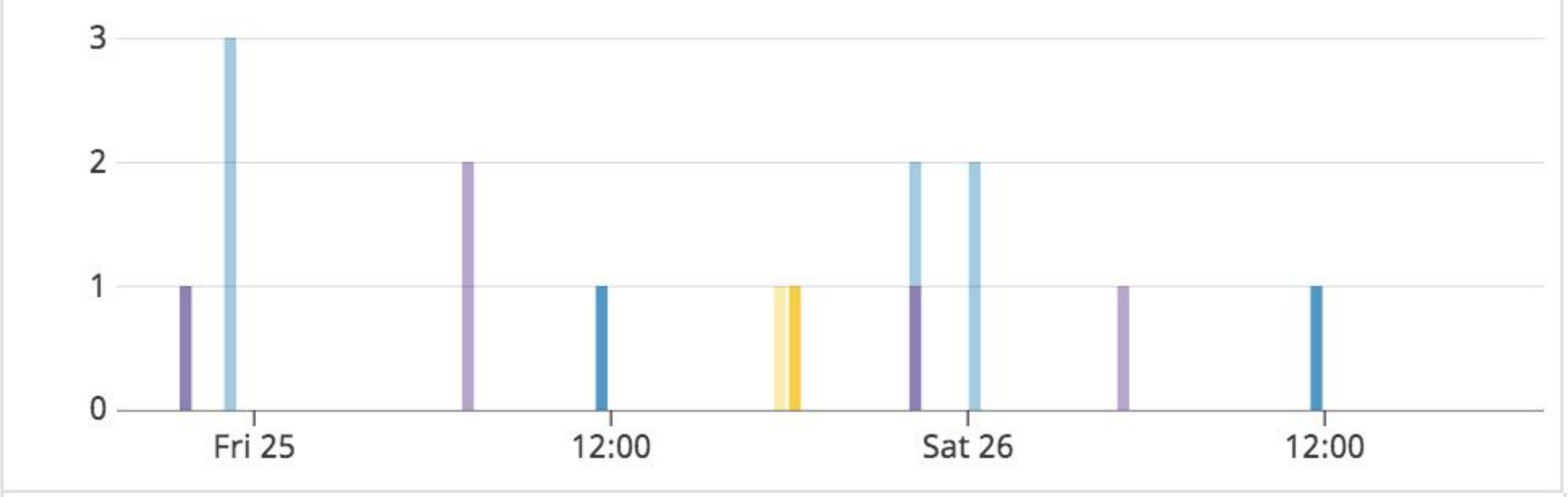
> *no expiration issues anymore*

lbernail

Impact of Certificate rotation

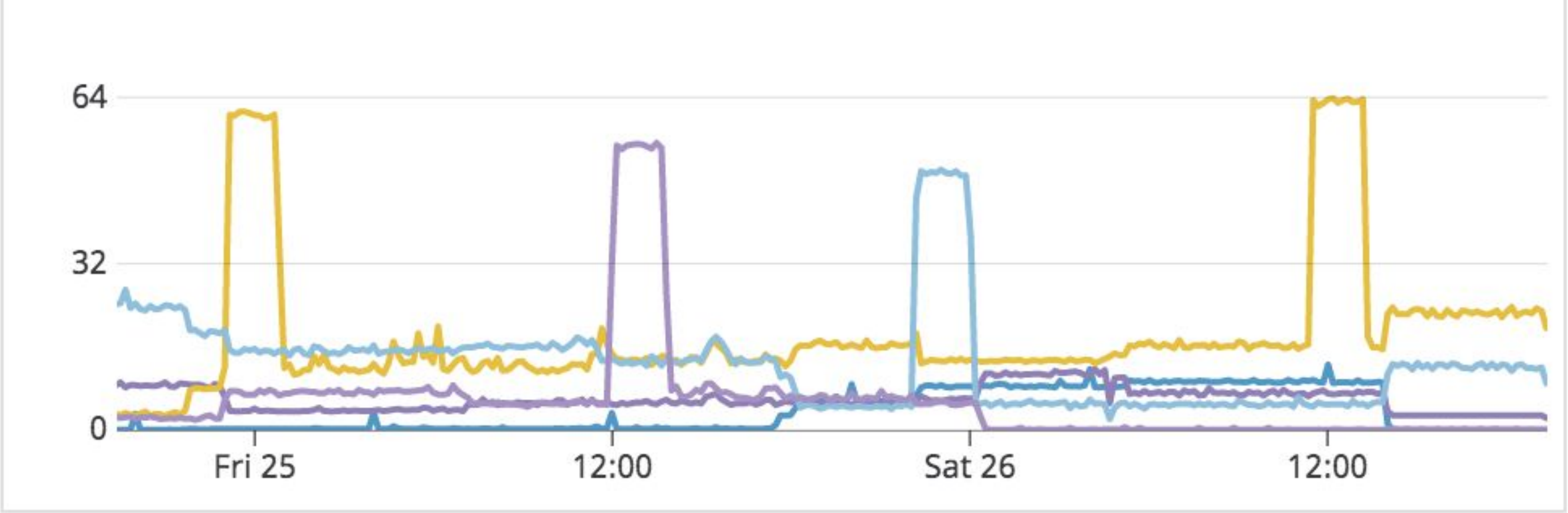
Apiserver restarts

apiserver restarts



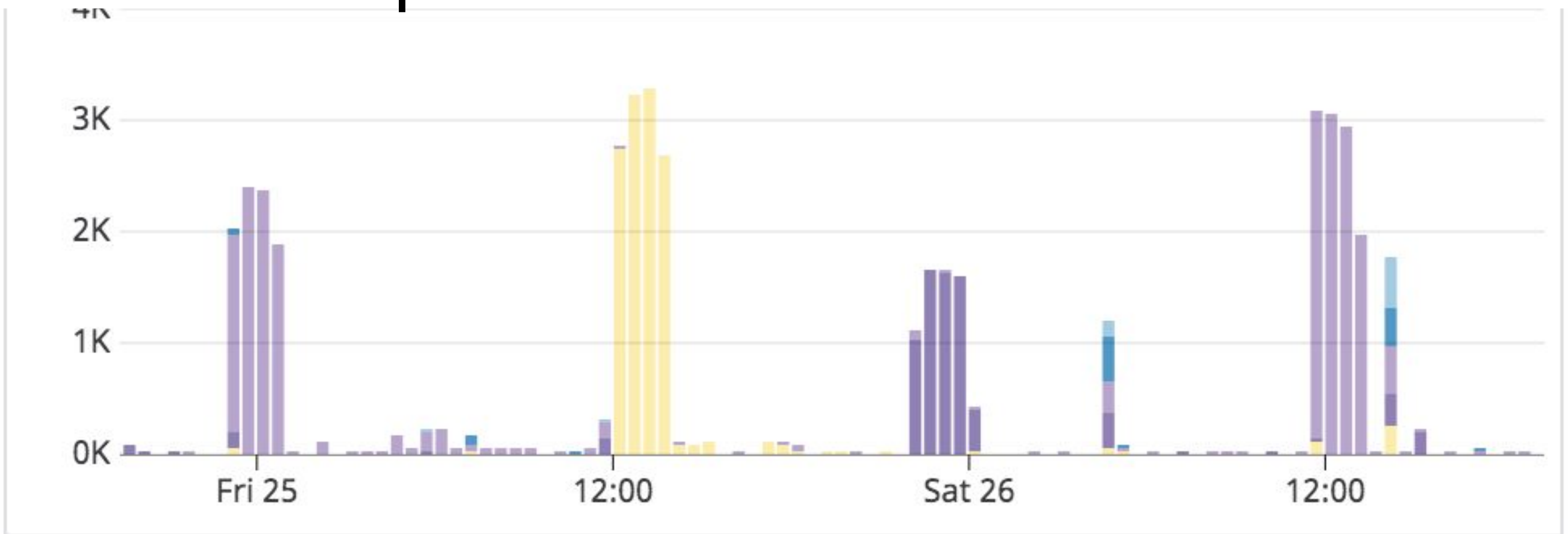
We have multiple apiservers
We restart each daily

etcd traffic



Significant etcd network impact
(caches are repopulated)

etcd slow queries

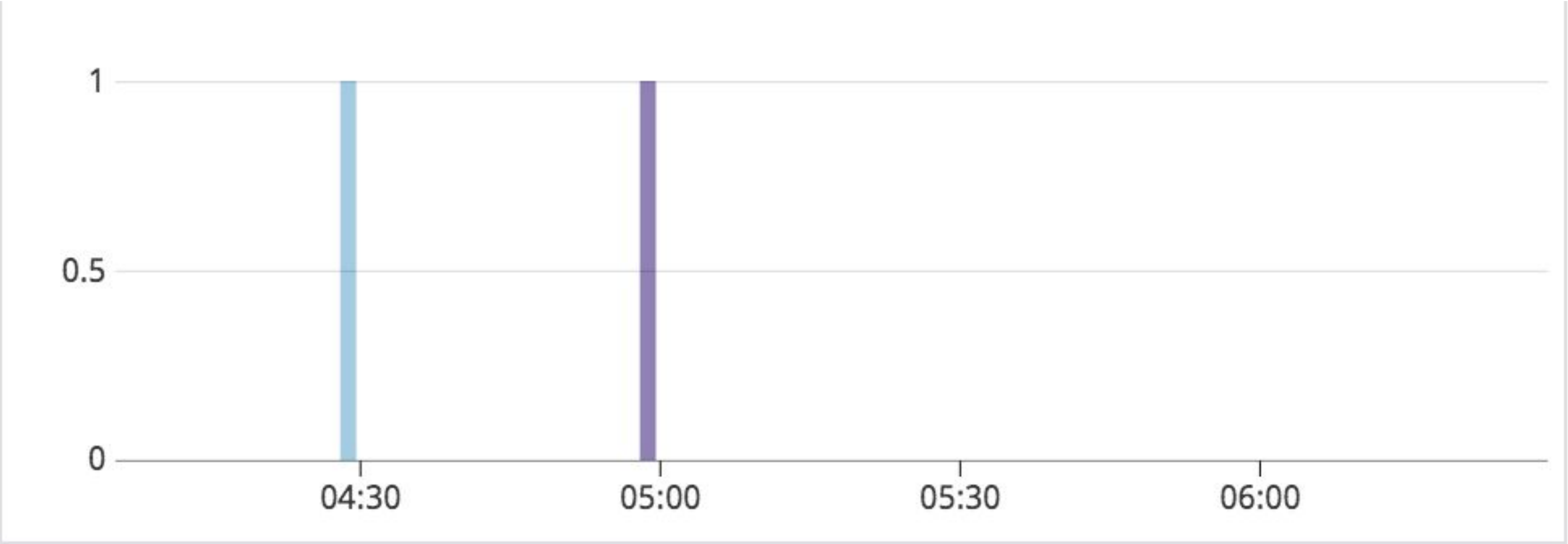


Significant impact on etcd performances

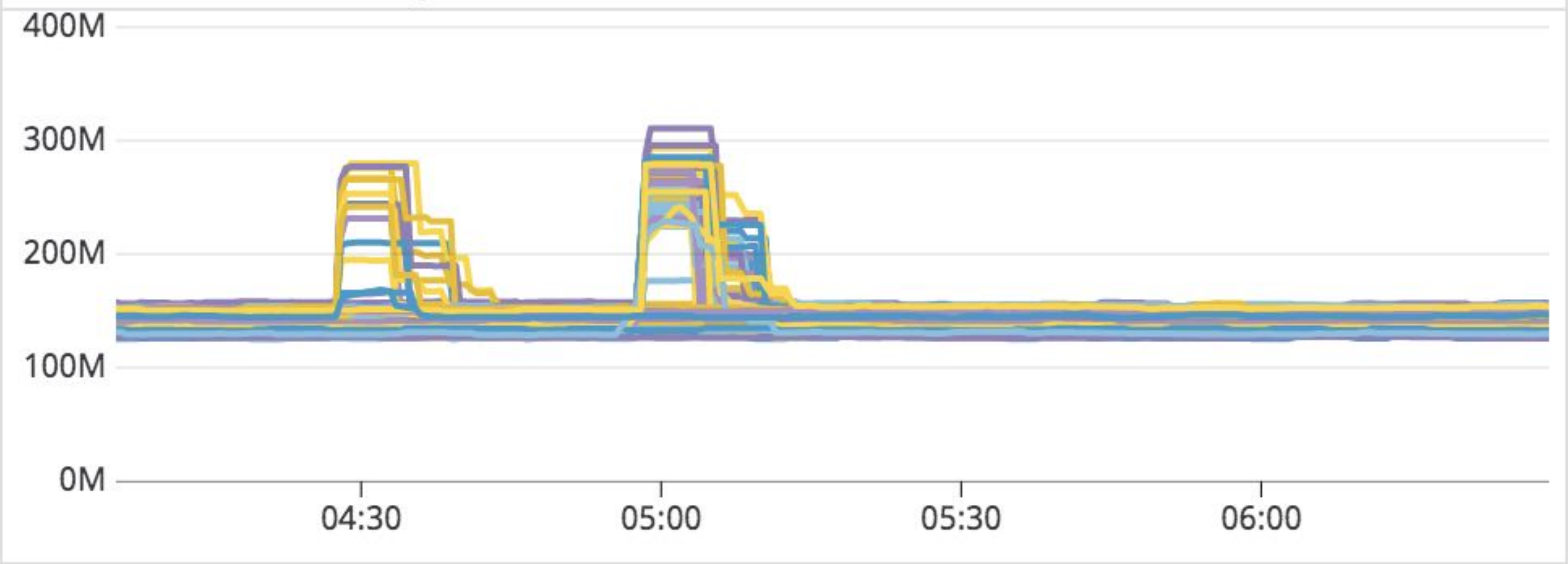
lbernail

Apiserver restarts, continued

apiserver restarts



coredns memory usage



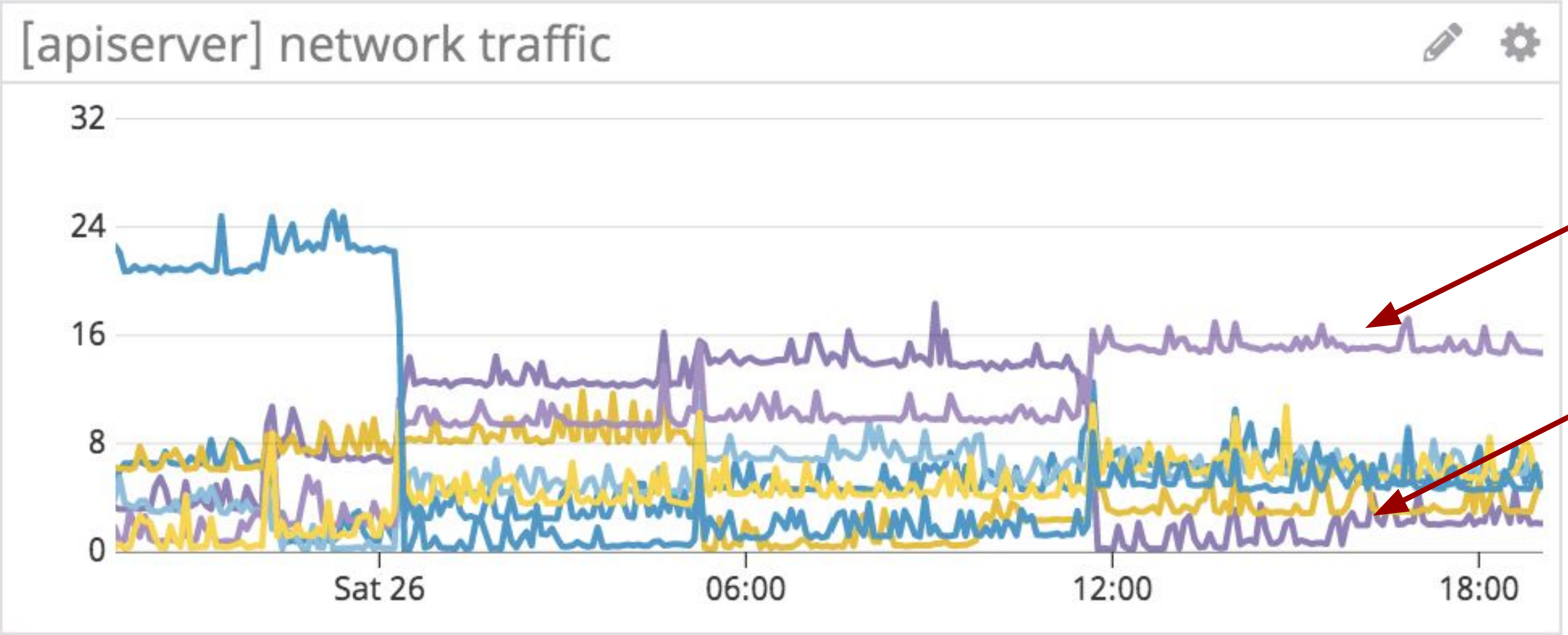
- Apiserver restarts
- clients reconnect and refresh their cache

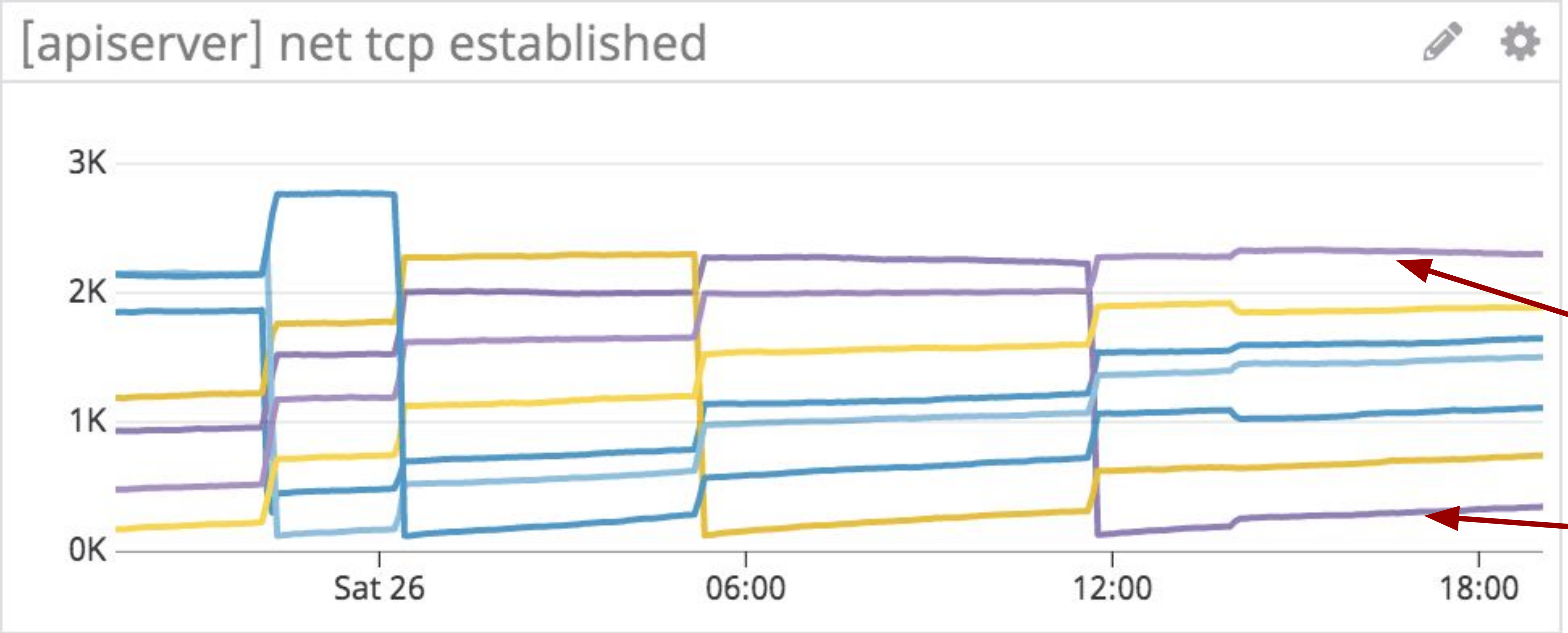
> Memory spike for impacted apps

No real mitigation today

lbernail

Unbalanced apiserver traffic





15MB/s

2.5MB/s

Number of connections / traffic very unbalanced
Because connections are very long-lived

More clients => Bigger impact clusterwide

2300 connections

300 connections

lbernail

Take-away

Restarting components is not transparent

- Not limited to apiservers, some issues with the Kubelet too

It would be great if

- Components could transparently reload certs (server & client)
- Clients could wait 0-Xs to reconnect to avoid thundering herd
- Reconnections did not trigger memory spikes
- Connections were rebalanced (kill them after a while?)

What happens after "Kube 101"

1. Resilient and Scalable Control Plane
2. Securing the Control Plane
    a. Kubernetes and Certificates
    b. Exceptions?
    c. Impact of Certificate Rotation
3. Efficient networking
    a. Giving pod IPs and routing them
    b. Accessing services: Client-side load-balancing:
    c. Ingresses: Getting data in the cluster

lbernail

Efficient networking

Network challenges

Throughput

Trillions of data points daily

Scale

1000-2000 nodes clusters

Latency

End-to-end pipeline

Topology

Multiple clusters

Access from standard VMs

Giving pods IPs & Routing them

From “the Hard Way”

Routes

Create network routes for each worker instance:

```
for i in 0 1 2; do
  gcloud compute routes create kubernetes-route-10-200-${i}-0-24 \
    --network kubernetes-the-hard-way \
    --next-hop-address 10.240.0.2${i} \
    --destination-range 10.200.${i}.0/24
done
```

node IP

Pod CIDR for this node

lbernail

Small cluster? Static routes

Node 1

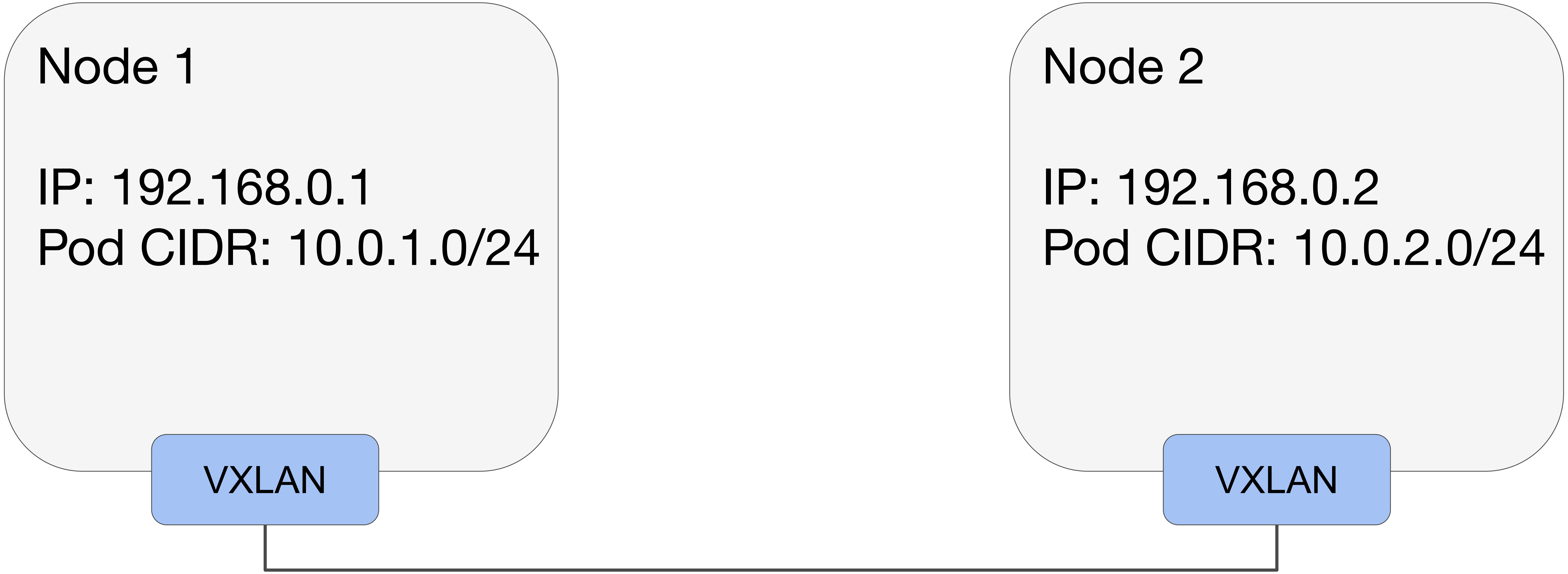
IP: 192.168.0.1
Pod CIDR: 10.0.1.0/24

Node 2

IP: 192.168.0.2
Pod CIDR: 10.0.2.0/24

Routes (local or cloud provider)
10.0.1.0/24 => 192.168.0.1
10.0.2.0/24 => 192.168.0.2

Mid-size cluster? Overlay



Tunnel traffic between hosts
Examples: Calico, Flannel

Large cluster with a lot of traffic?
Native pod routing

Performance

Datapath: no Overlay
Control plane: simpler

Addressing

Pod IPs are accessible from

- Other clusters
- VMs

In practice

On premise

BGP
- Calico
- Kube-router

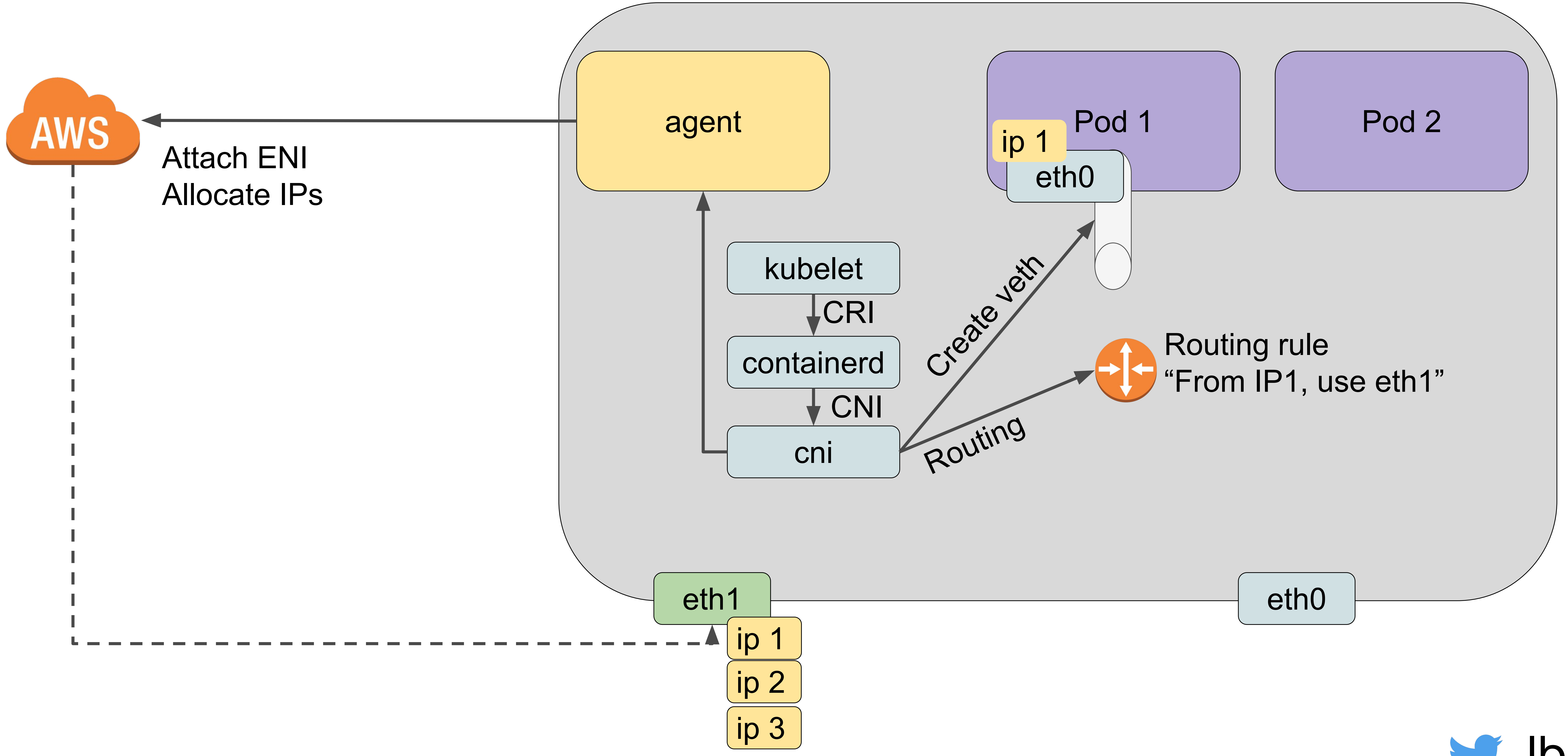
GCP

IP aliases

AWS

Additional IPs on ENIs
- AWS EKS CNI plugin
- Lyft CNI plugin
- Cilium ENI IPAM

A bit more complex on AWS



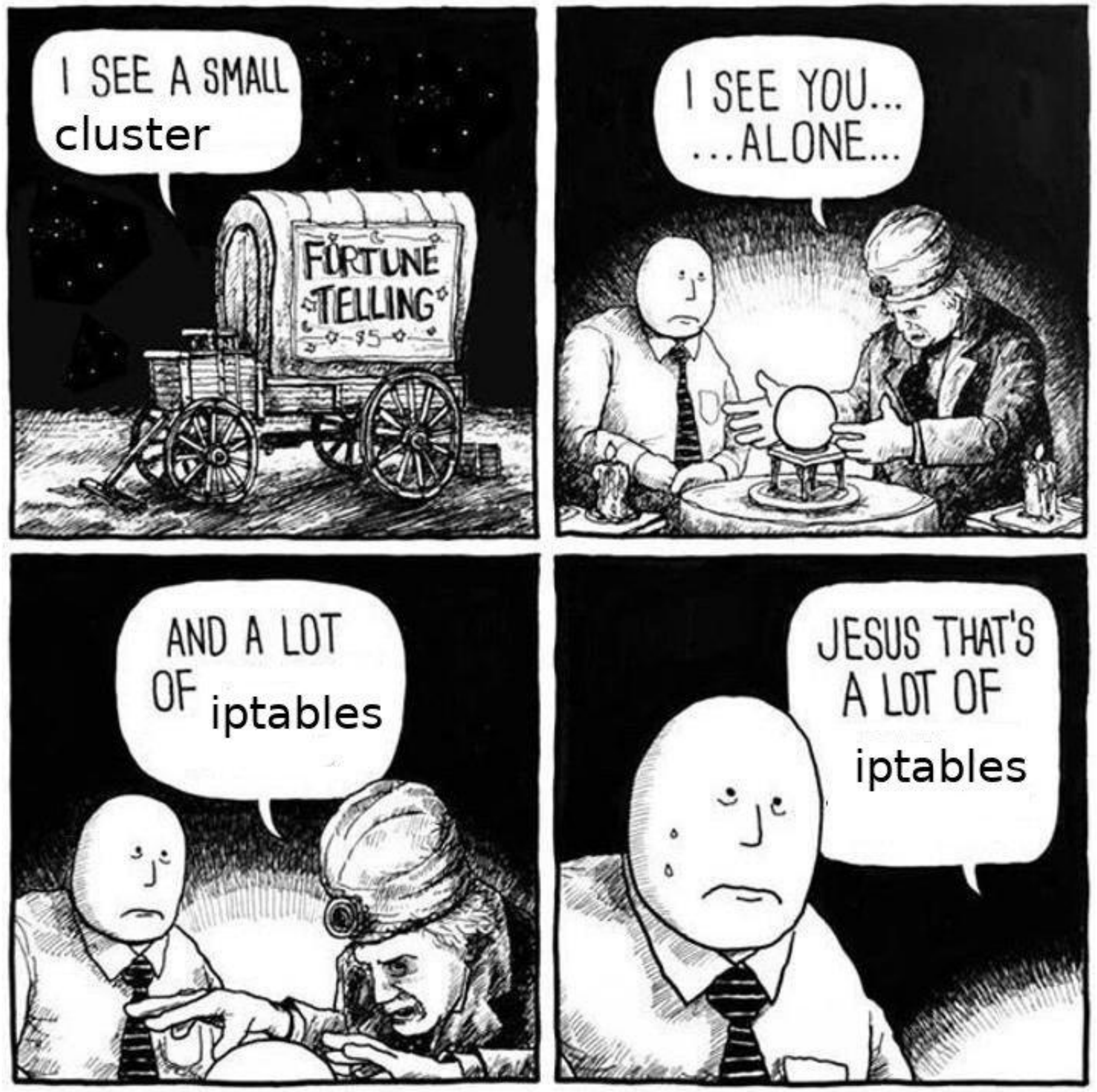
lbernail

Take-away

- Native pod routing has worked very well at scale
- A bit more complex to debug
- Much more efficient datapath
- Topic is still dynamic (Cilium introduced ENI recently)
- Great relationship with Lyft / Cilium

Accessing Services

Kube-proxy default: iptables

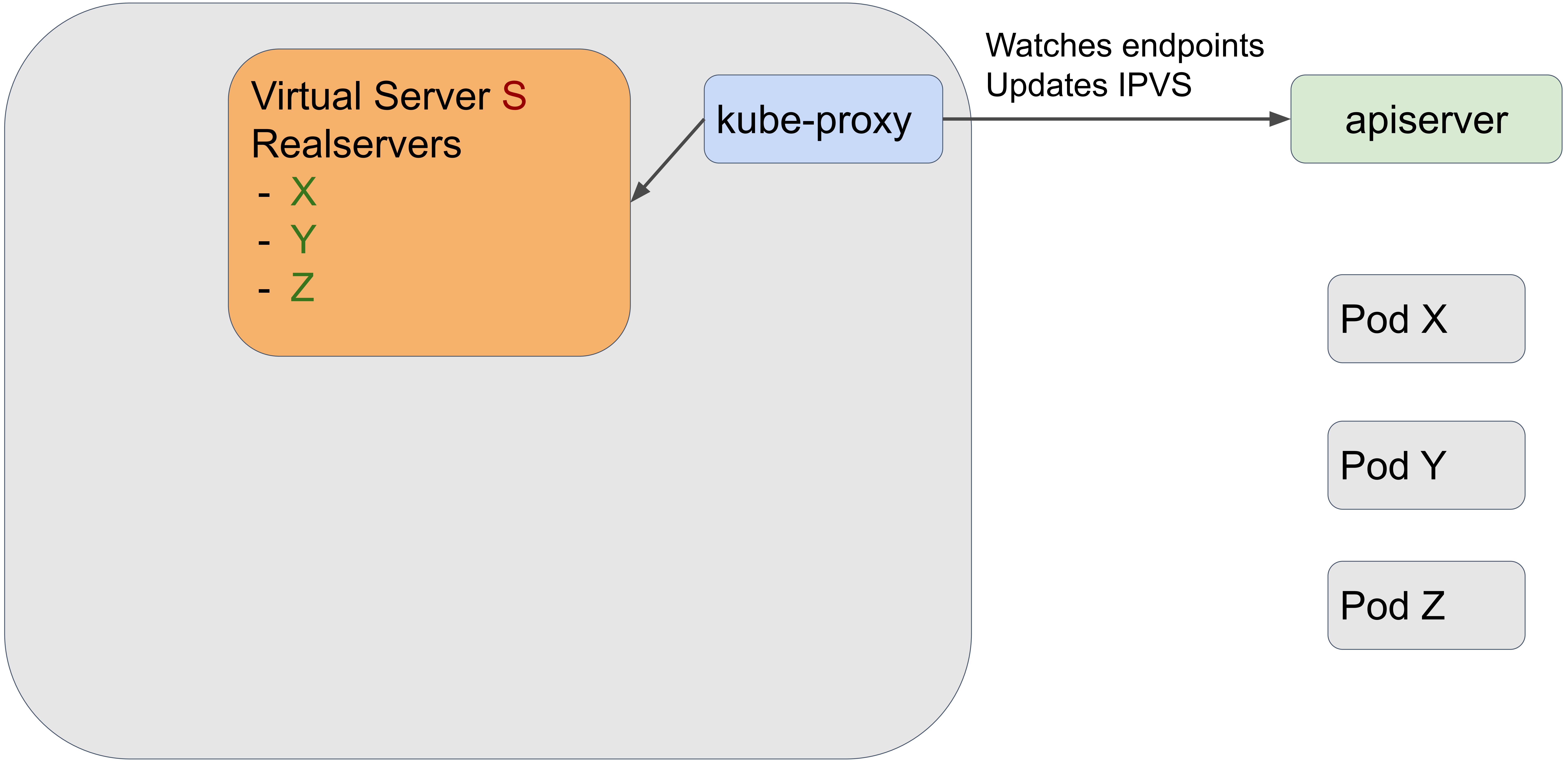


Mid size cluster
iptables -S -t nat | wc -l
48688

lbernail

Alternative: IPVS

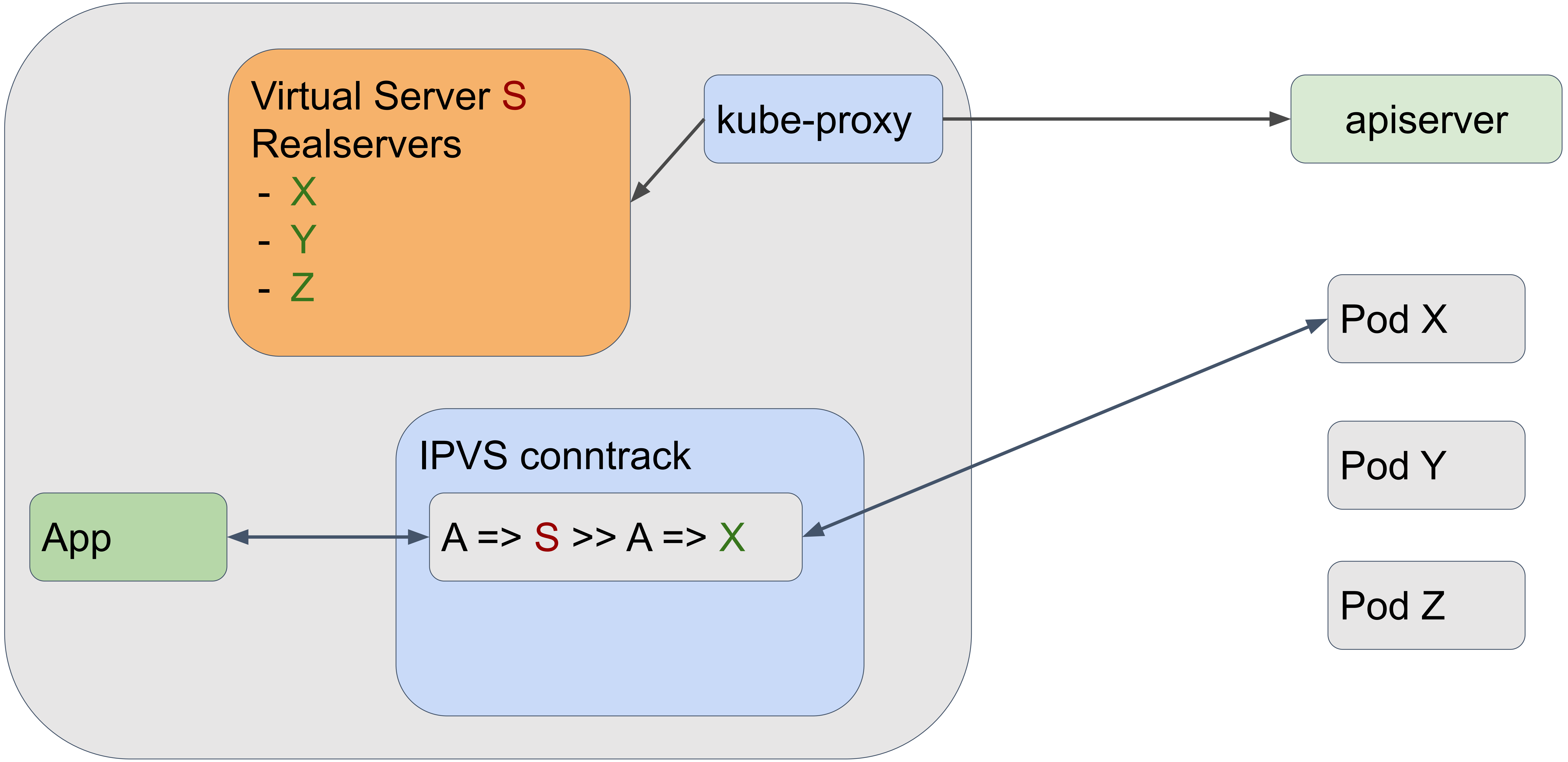
Service ClusterIP:Port S
Backed by pod:port X, Y, Z



lbernail

New connection

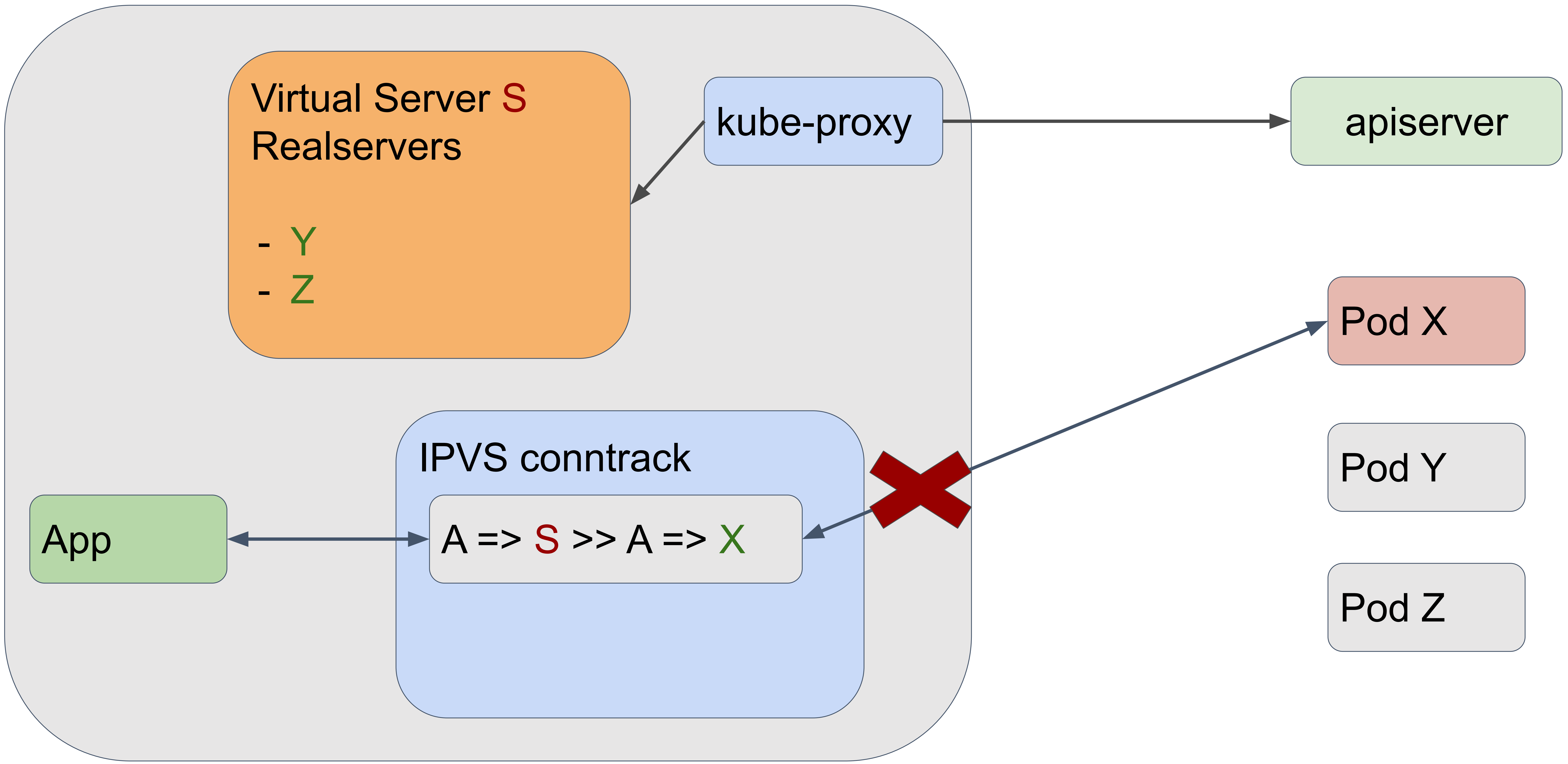
App establishes connection to S
IPVS associates Realserver X



lbernail

Pod X deleted

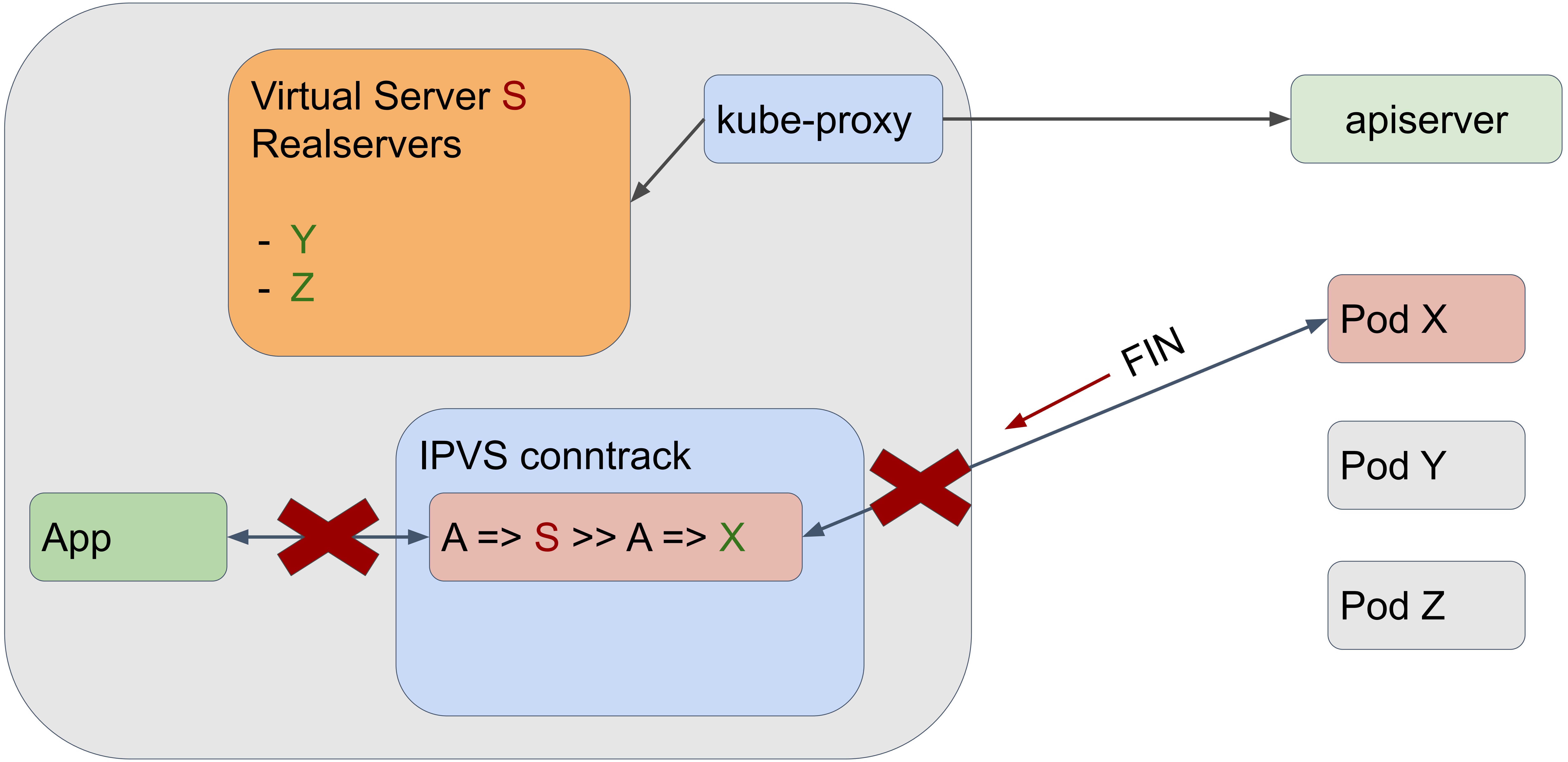
Apiserver removes X from S endpoints
Kube-proxy removes X from realservers
Kernel drops traffic (no realserver)



lbernail

Pod X deleted

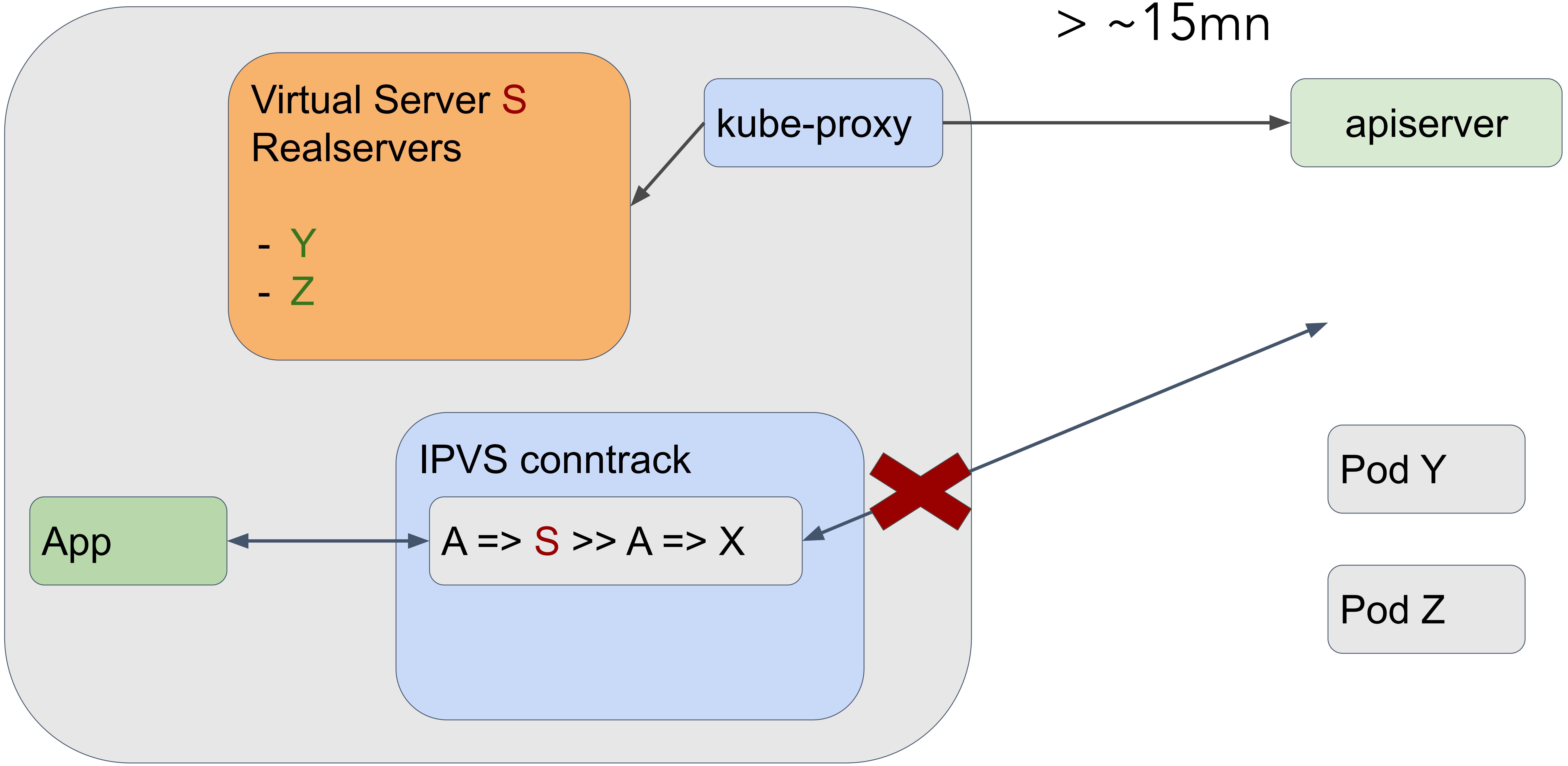
Pod X sends FIN on exit
Conntrack entry deleted
Connection from A terminates



lbernail

What if X doesn’t send FIN?

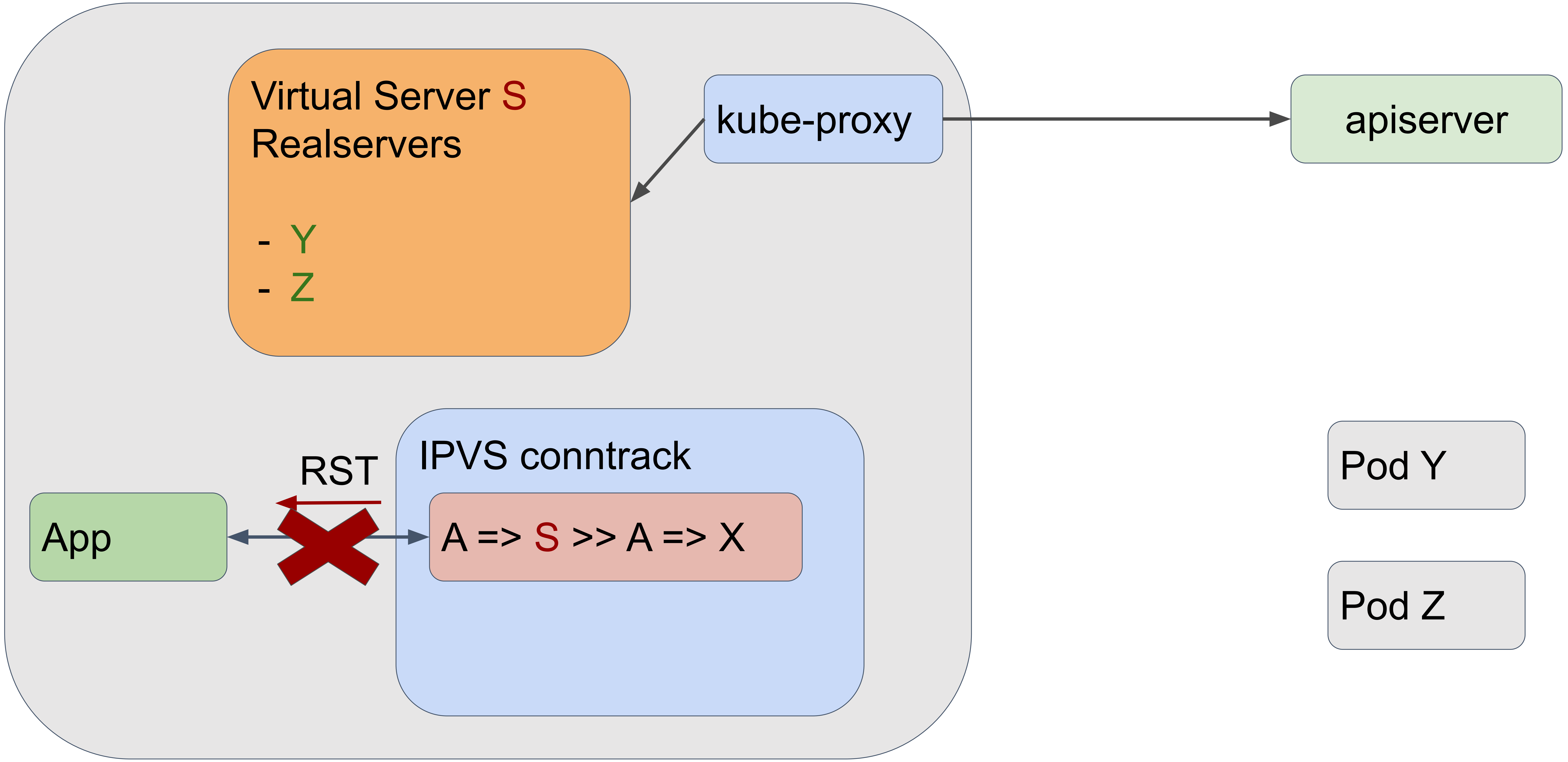
Traffic blackholed until App detects issue
> tcp_retries2 (default 15)
> ~15mn



lbernail

Mitigation

net/ipv4/vs/expire_nodest_conn
Delete conntrack entry on next packet
Forcefully terminate (RST)



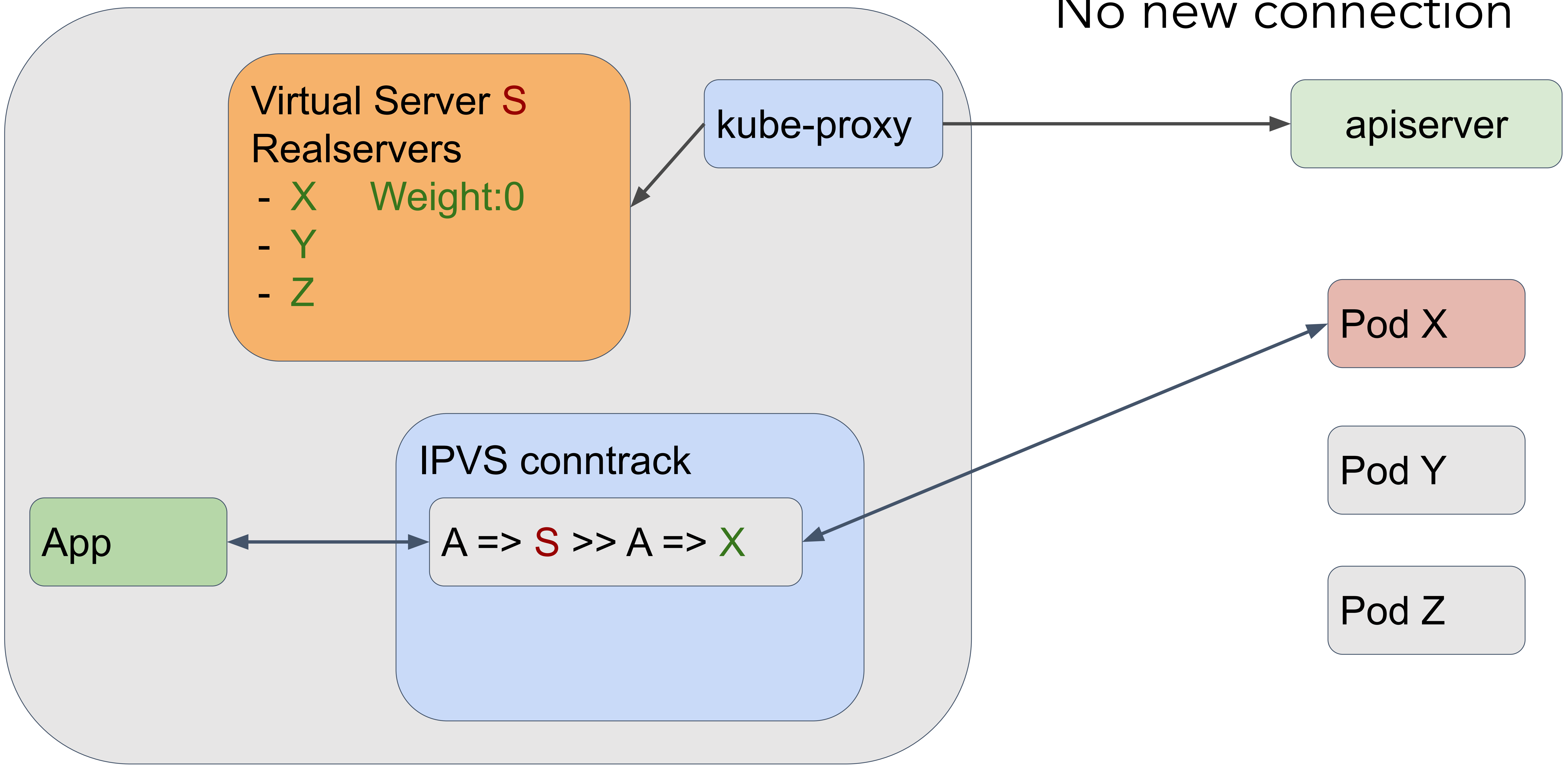
lbernail

Limit?

- No graceful termination
- As soon as a pod is Terminating connections are destroyed
- Addressing this took time

Graceful termination

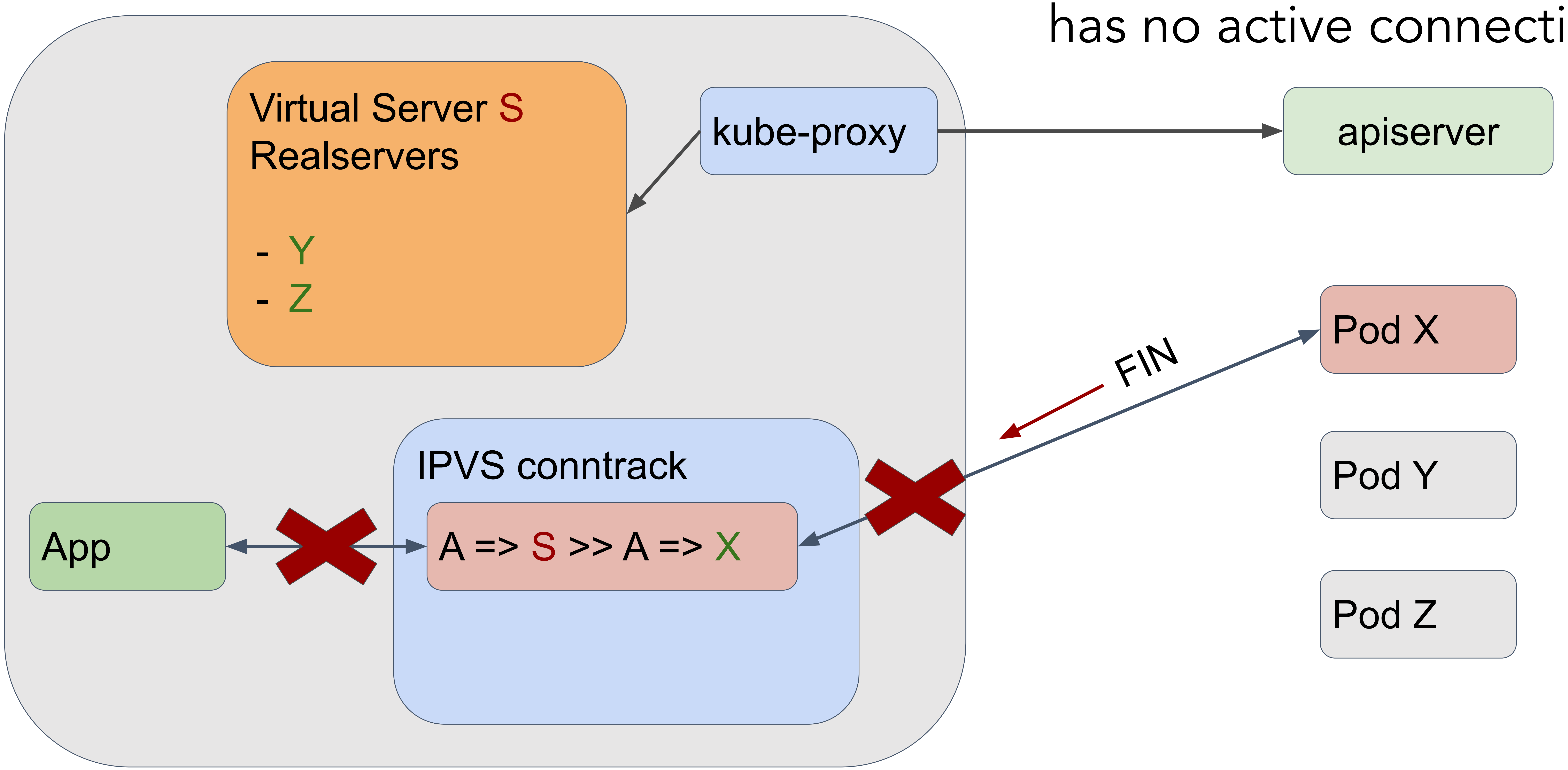
Apiserver removes X from S endpoints
Kube-proxy sets Weight to 0
No new connection



lbernail

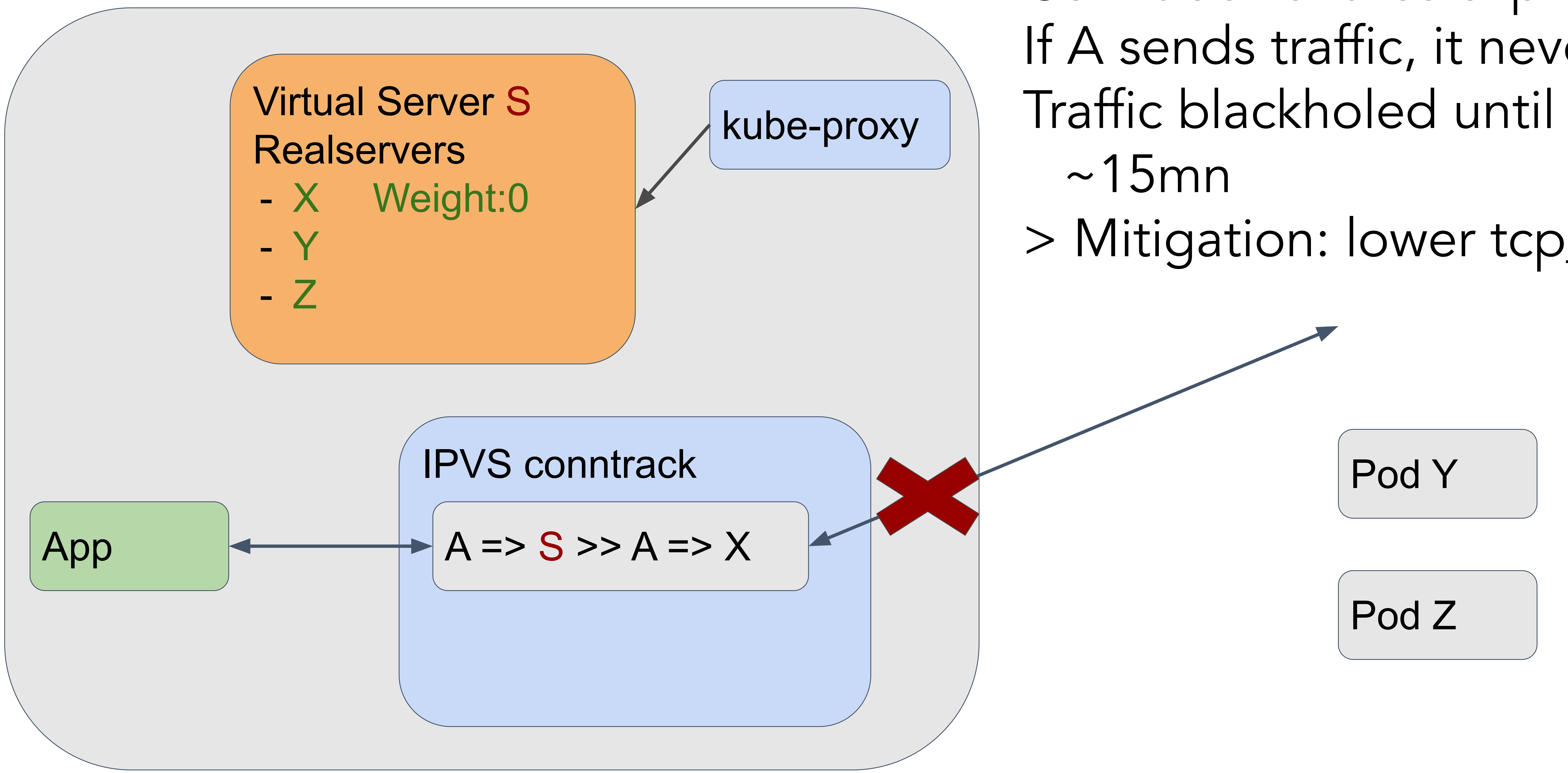
Garbage collection

Pod exits and sends FIN
Kube-proxy removes realserver when it has no active connection



lbernail

What if X doesn't send FIN?



Conntrack entries expires after 900s
If A sends traffic, it never expires
Traffic blackholed until App detects issue
~15mn
> Mitigation: lower tcp_retries2

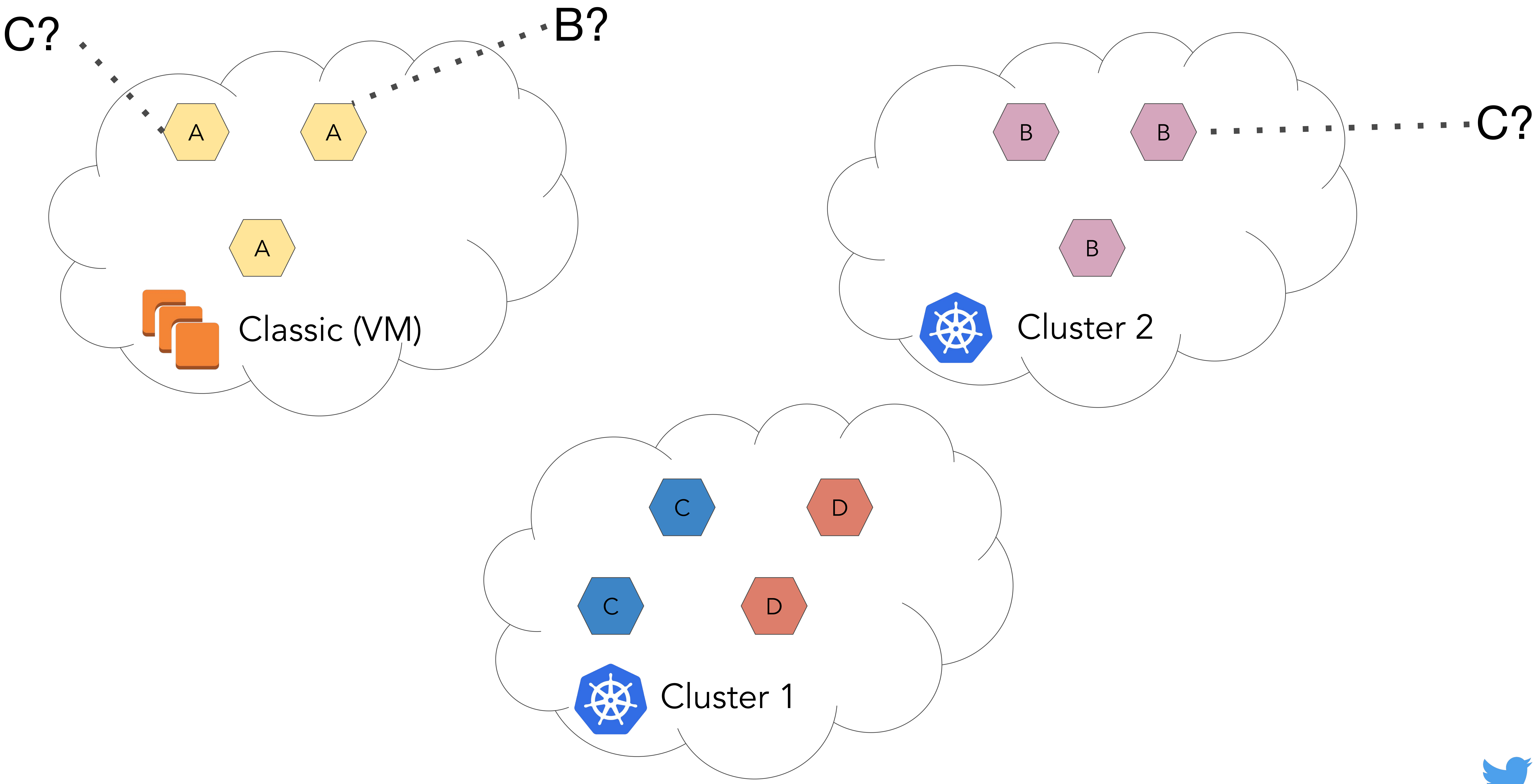
lbernail

Take-away

- IPVS has been great for us
- IPVS is in a good state now
- Several improvements in the works
- But harder than we expected
- I ended up reviewer/approver for kube-proxy/IPVS

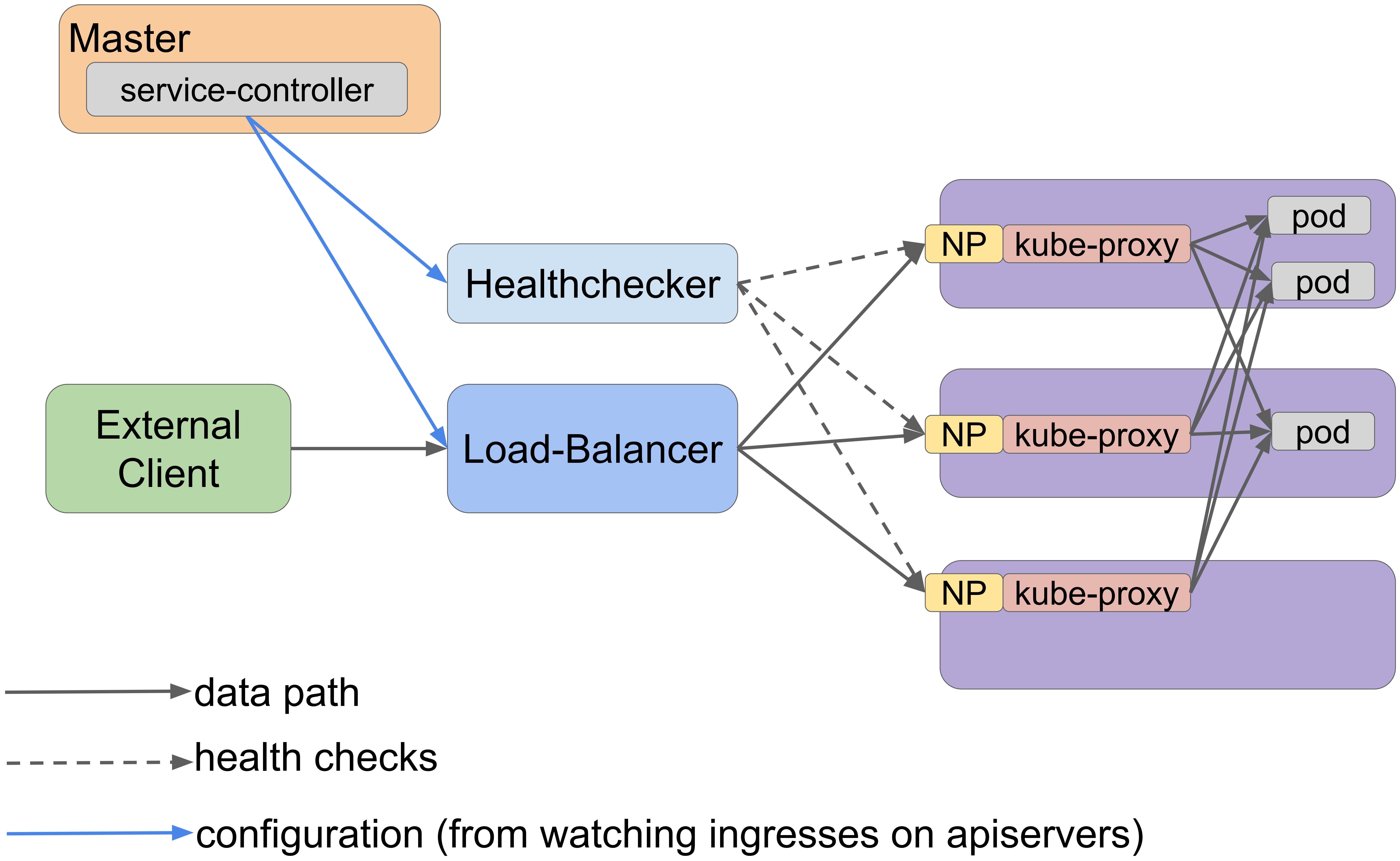
Ingresses

Ingress: cross-clusters, VM to clusters



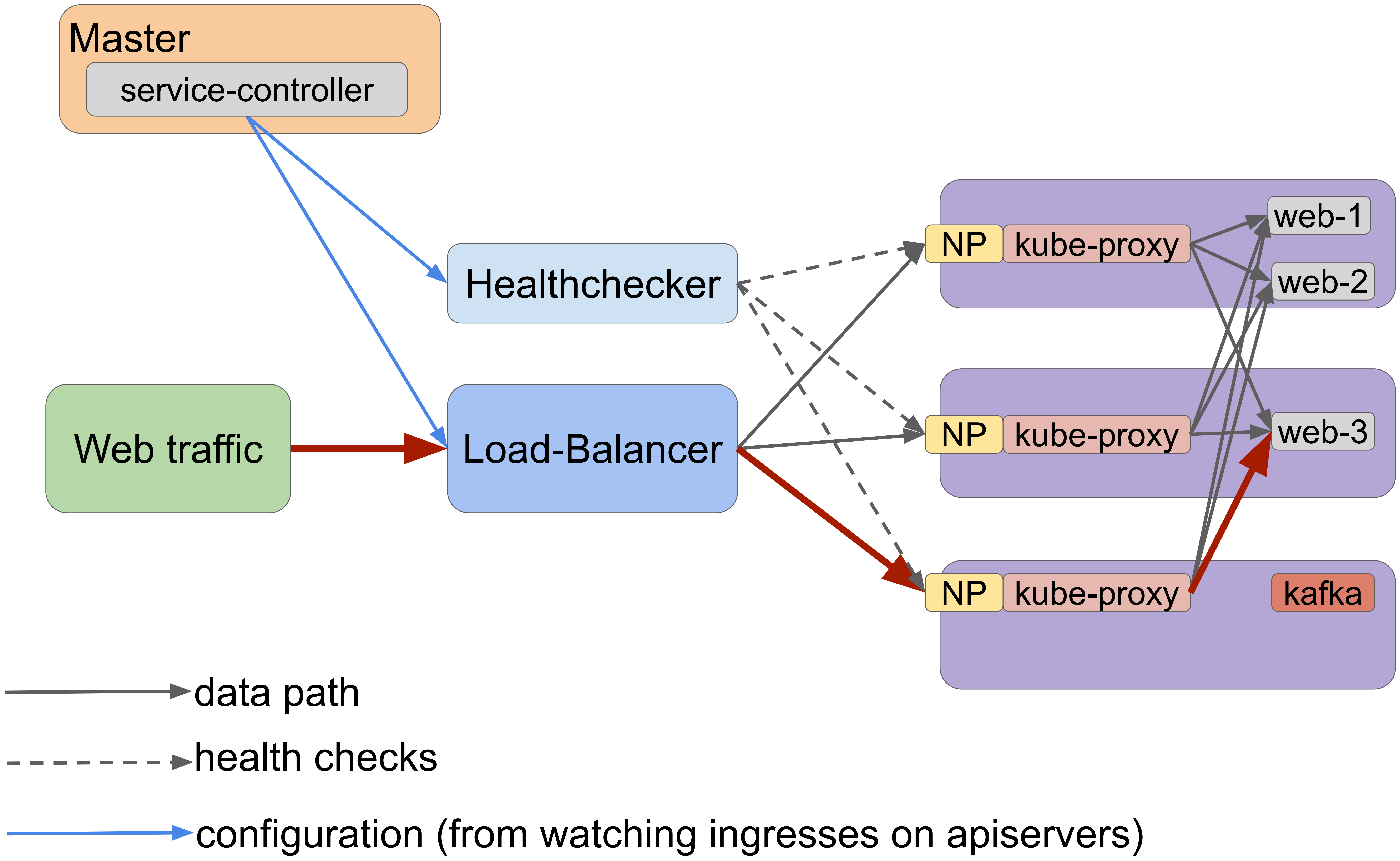
lbernail

Kubernetes default: LB service



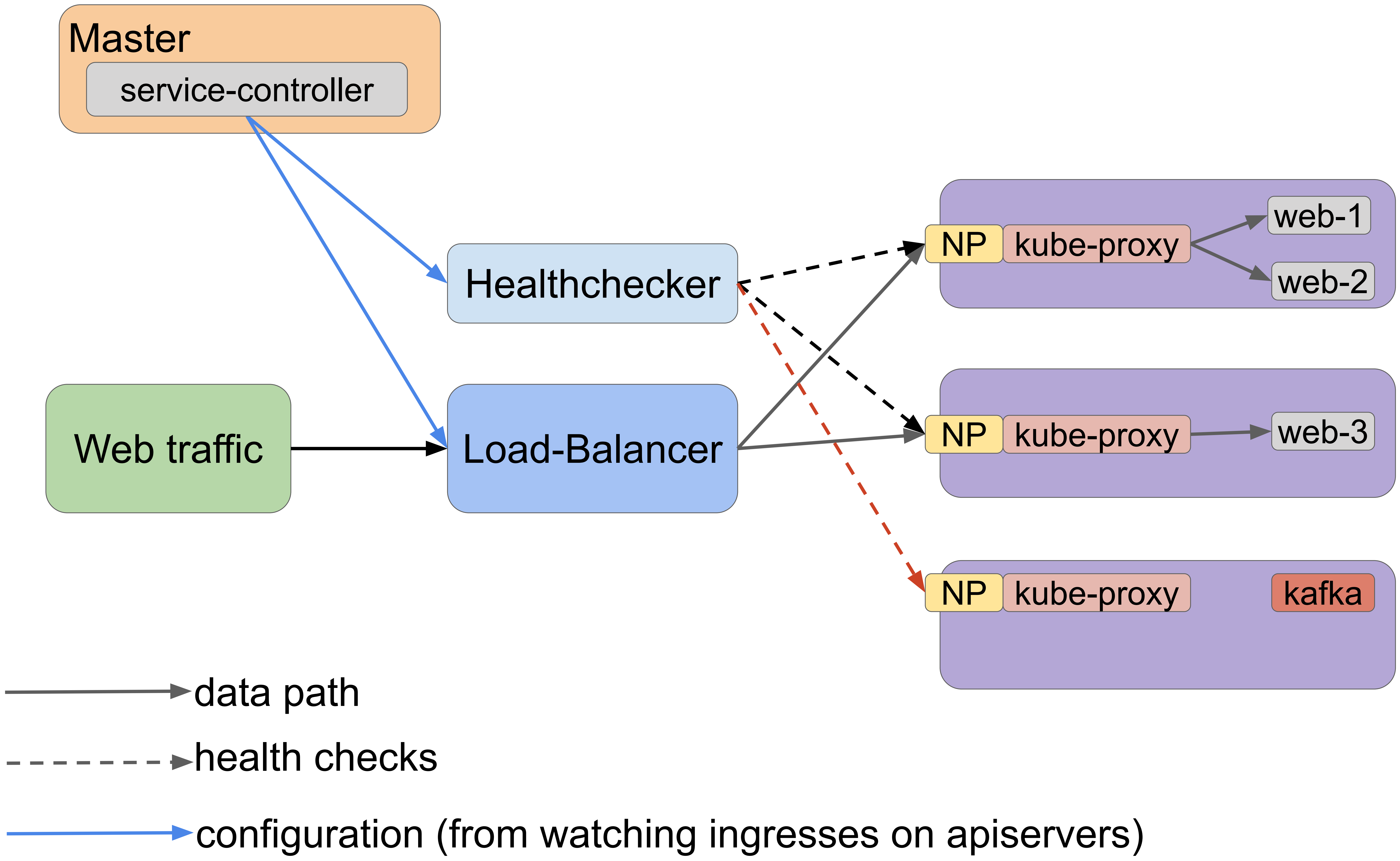
lbernail

Inefficient Datapath & cross-application impacts

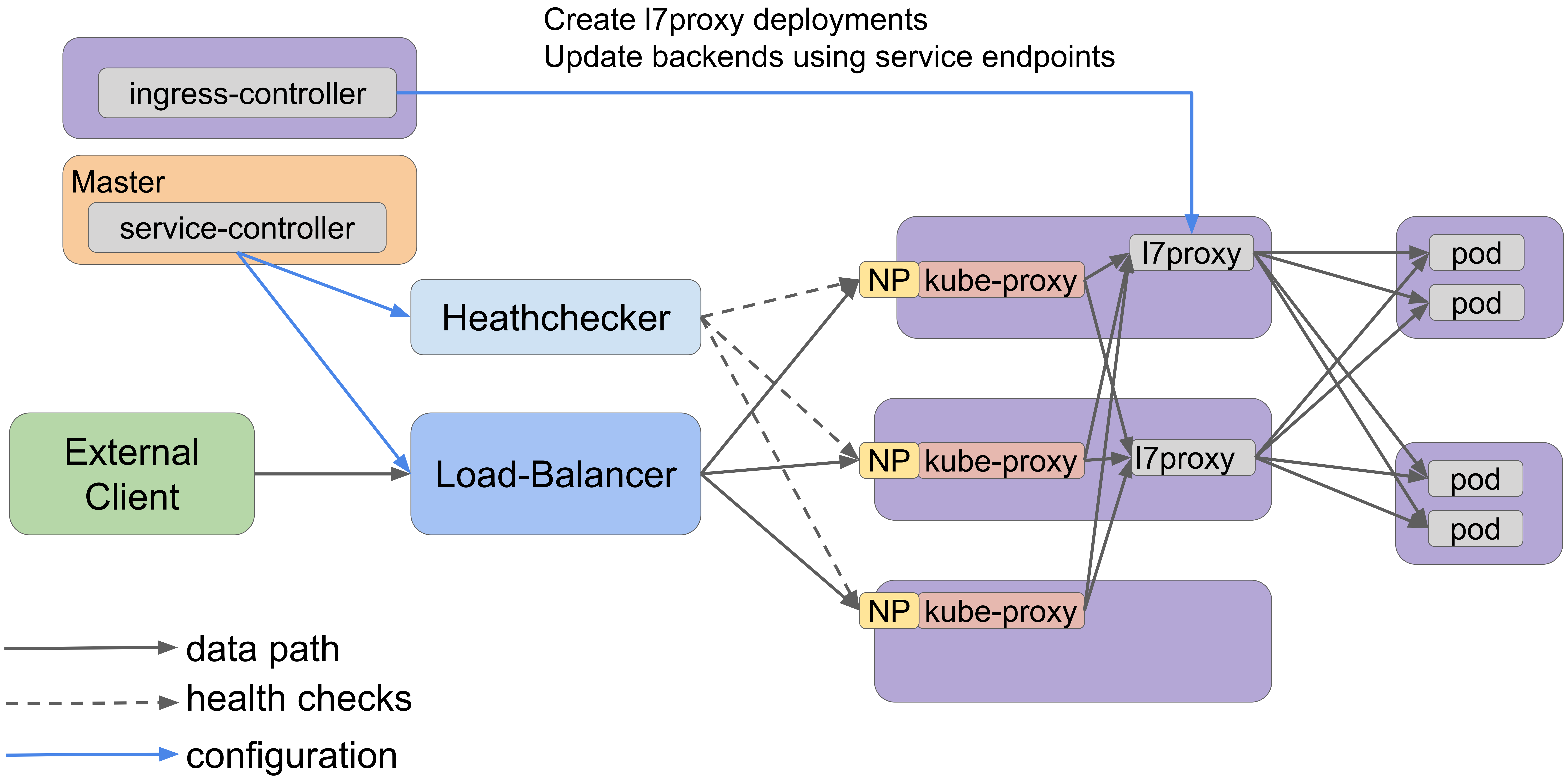


lbernail

ExternalTrafficPolicy: Local?



L7-proxy ingress controller



from watching ingresses/endpoints on apiservers (ingress-controller)
from watching LoadBalancer services (service-controller)

lbernail

Challenges

Limits

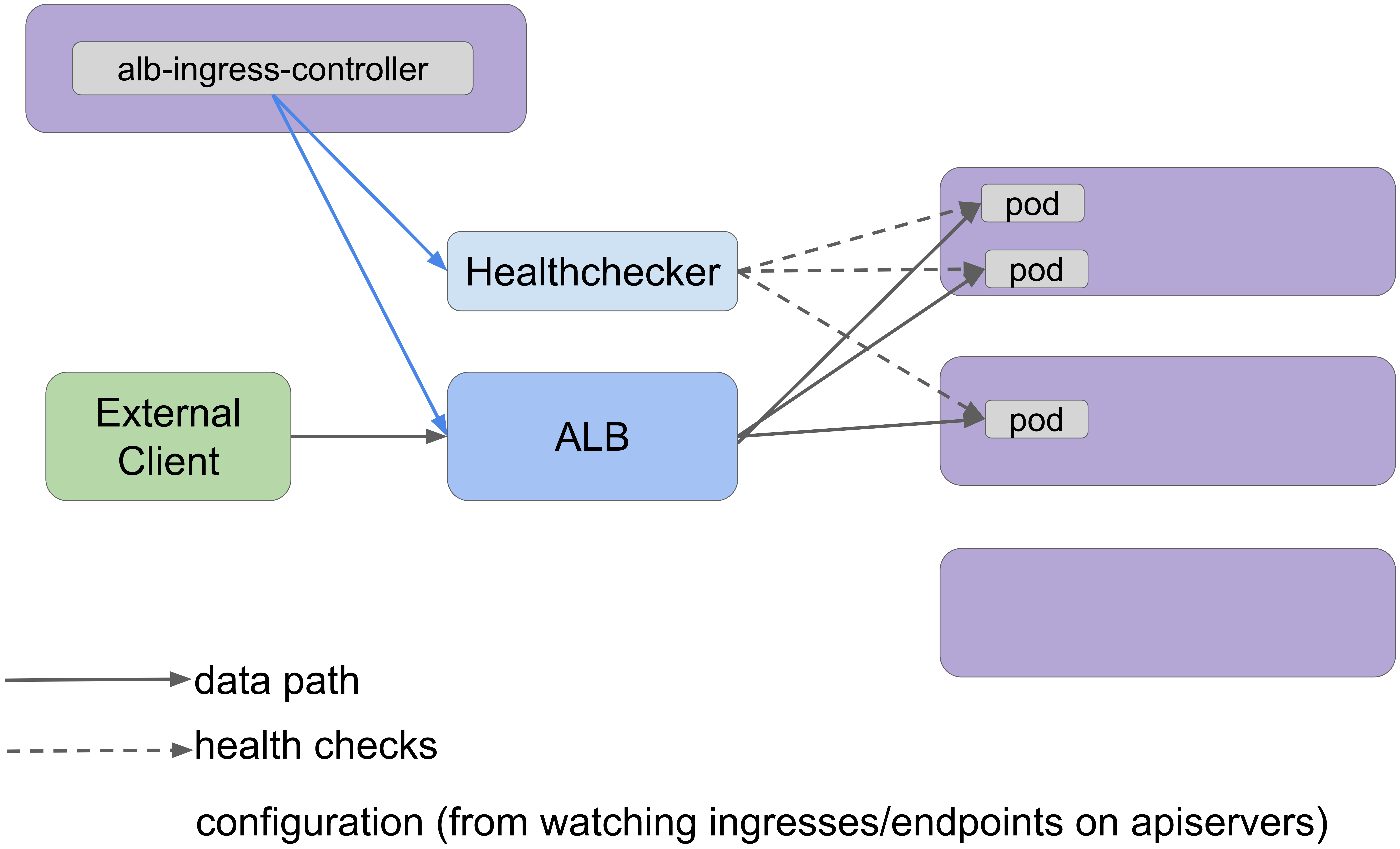
All nodes as backends (1000+)
Inefficient datapath
Cross-application impacts

Alternatives?

ExternalTrafficPolicy: Local?
> Number of nodes remains the same
> Issues with some CNI plugins
K8s ingress
> Still load-balancer based
> Need to scale ingress pods
> Still inefficient datapath

lbernail

Our target: native routing



lbernail

Remaining challenges

Limited to HTTP ingresses

No support for TCP/UDP

Ingress v2 should address this

Registration delay

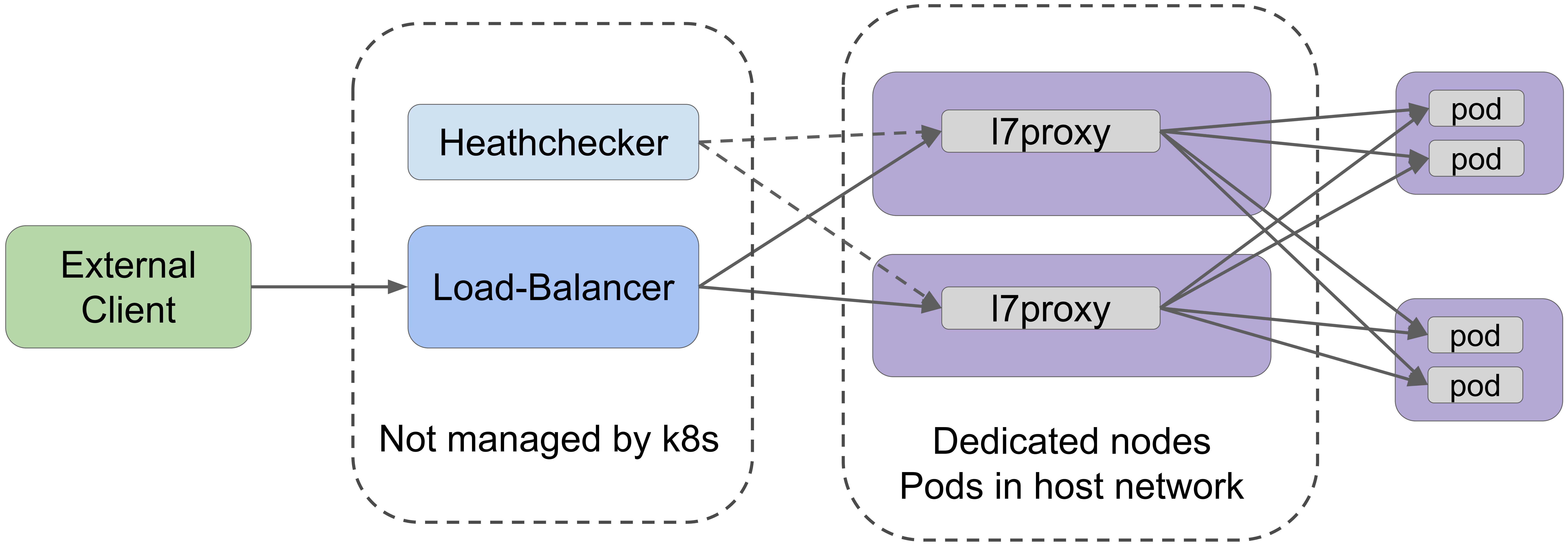
Slow registration with LB
Pod rolling-updates much faster

Mitigations
- MinReadySeconds
- Pod ReadinessGates

Workaround

TCP / Registration delay not manageable

> Dedicated gateways



lbernail

Take-away

- Ingress solutions are not great at scale yet
- May require workarounds
- Definitely a very important topic for us
- The community is working on v2 Ingresses

Conclusion

A lot of other topics

- DNS (it's always DNS!)
- Challenges with Stateful applications
- How to DDOS <insert ~anything> with Daemonsets
- Node Lifecycle
- Cluster Lifecycle
- Deploying applications
- …

You want more horror stories?

"Kubernetes the very hard way at Datadog"
https://www.youtube.com/watch?v=2dsCwp_j0yQ

"10 ways to shoot yourself in the foot with Kubernetes"
https://www.youtube.com/watch?v=QKI-JRs2RIE

"Kubernetes Failure Stories"
https://k8s.af

Key lessons

Self-managed Kubernetes is hard

> If you can, use a managed service

Networking is not easy (especially at scale)

The main challenge is not technical

> Build a team

> Transforming practices and training users is very important

lbernail

Thank you

We're hiring!
https://www.datadoghq.com/careers/

laurent@datadoghq.com
@lbernail

DATADOG