CoreOS

Self driving infrastructure

Xiang Li

xiang.li@coreos.com | Head of distributed system

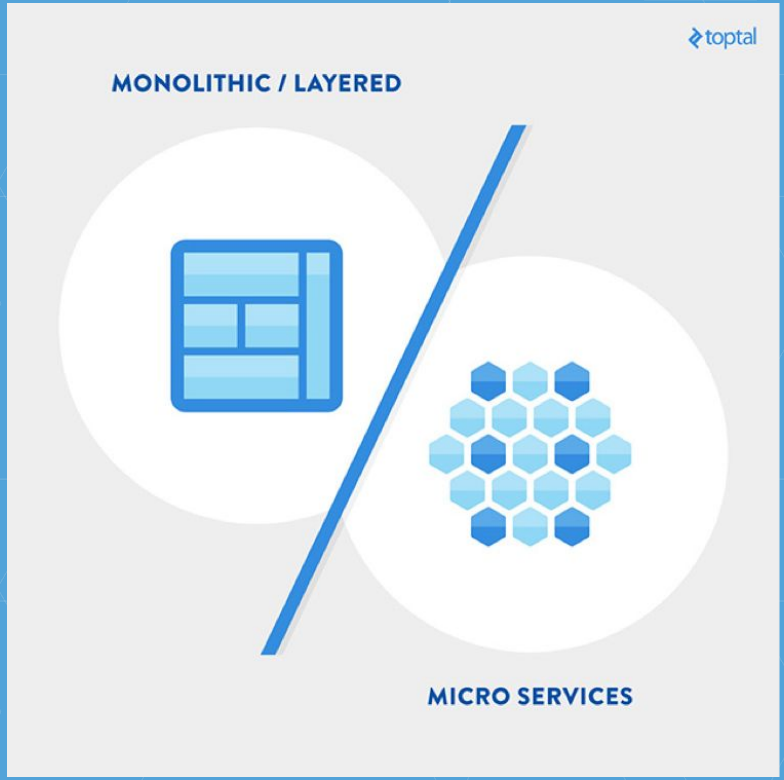
CoreOS

Topics

- Cluster management systems
- Today’s problems with operating cluster management systems
- A self-driving approach

Motivation: microservices

- Increased operational cost
  - a lot of components
  - dynamic dependencies
  - fast deployment iteration
- Solution: automation

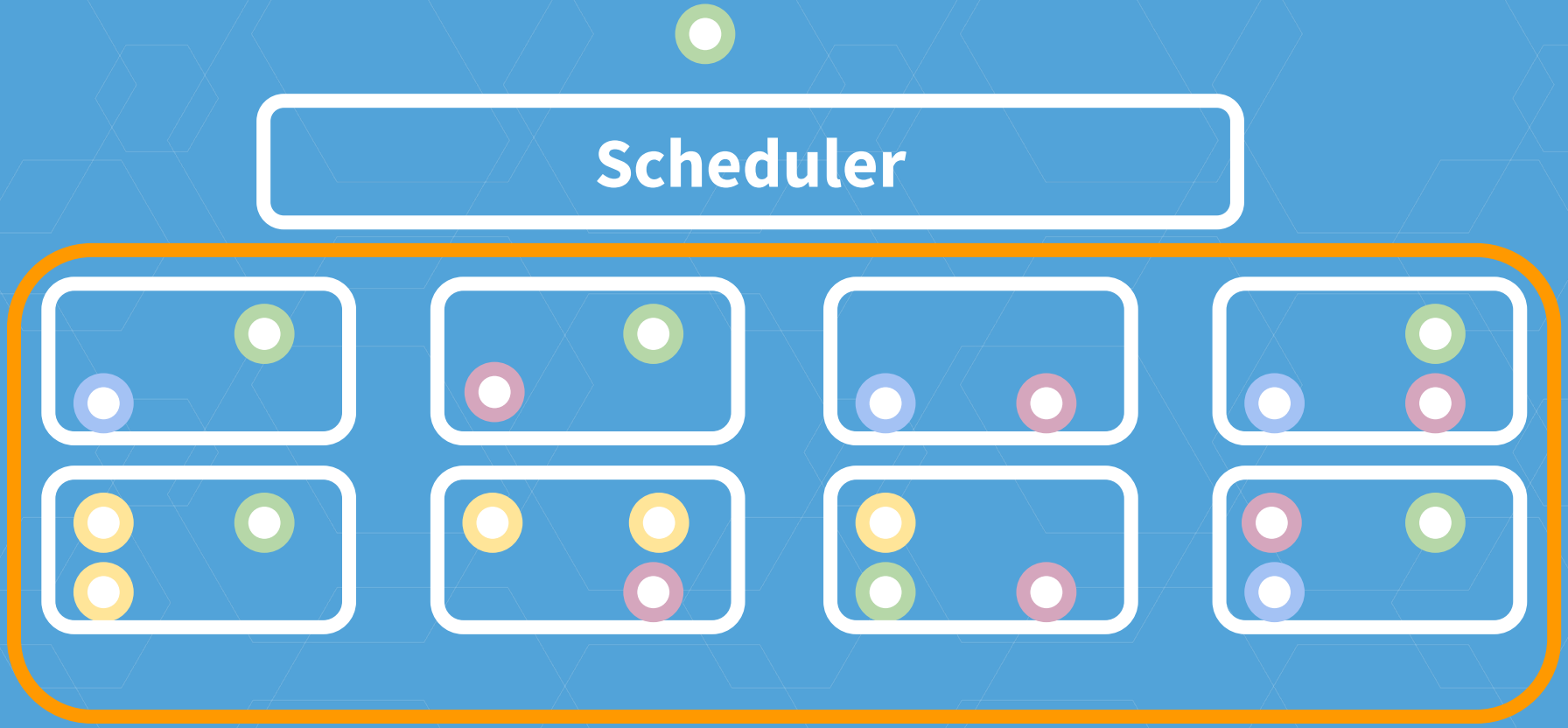


CoreOS

Cluster management system

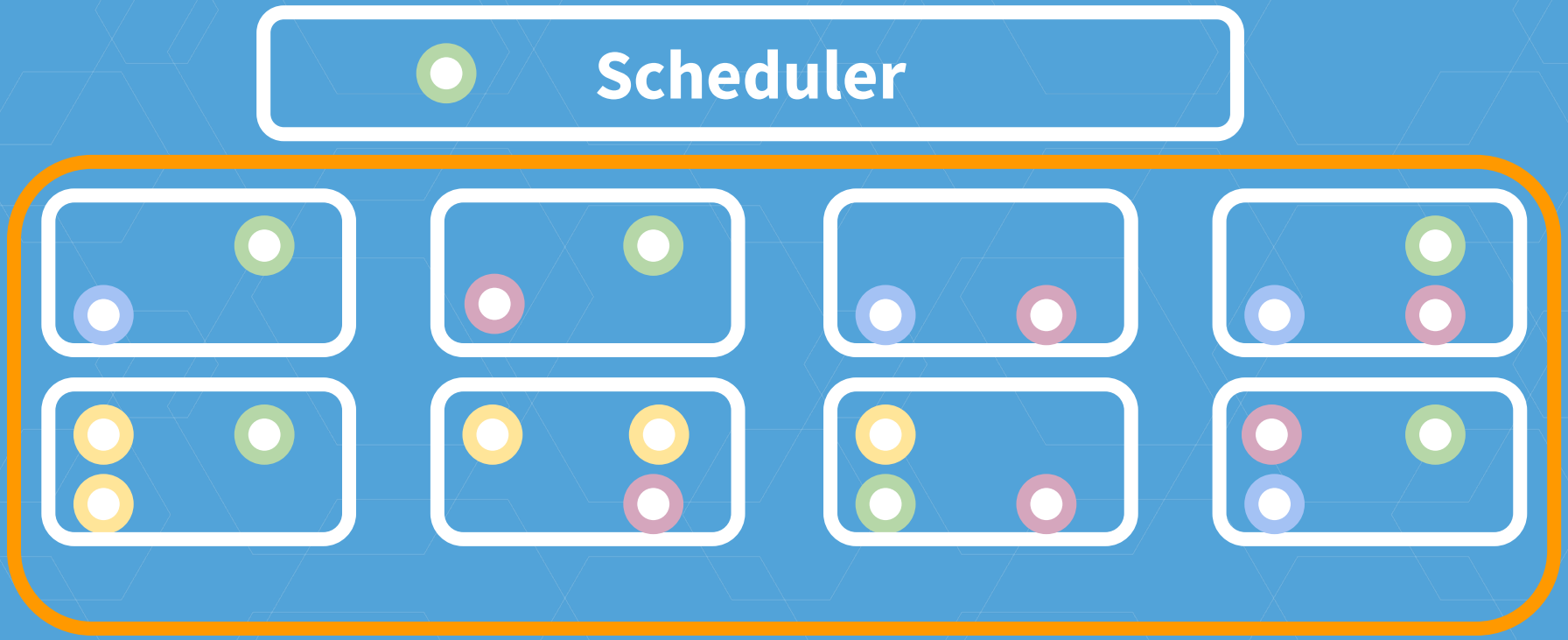
- Automation
  - Scheduling
  - Deployment
  - Healing
  - Discovery/load balancing
  - Scaling

Scheduling



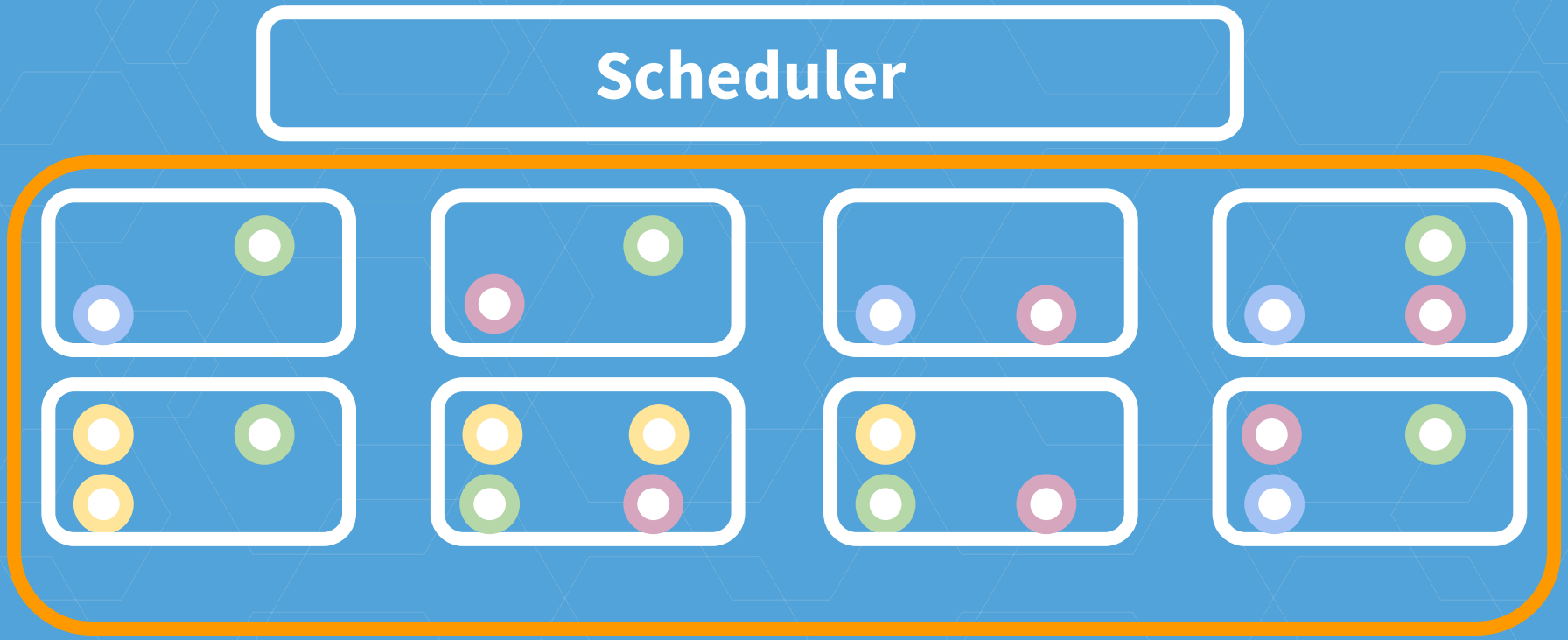
CoreOS

Scheduling



CoreOS

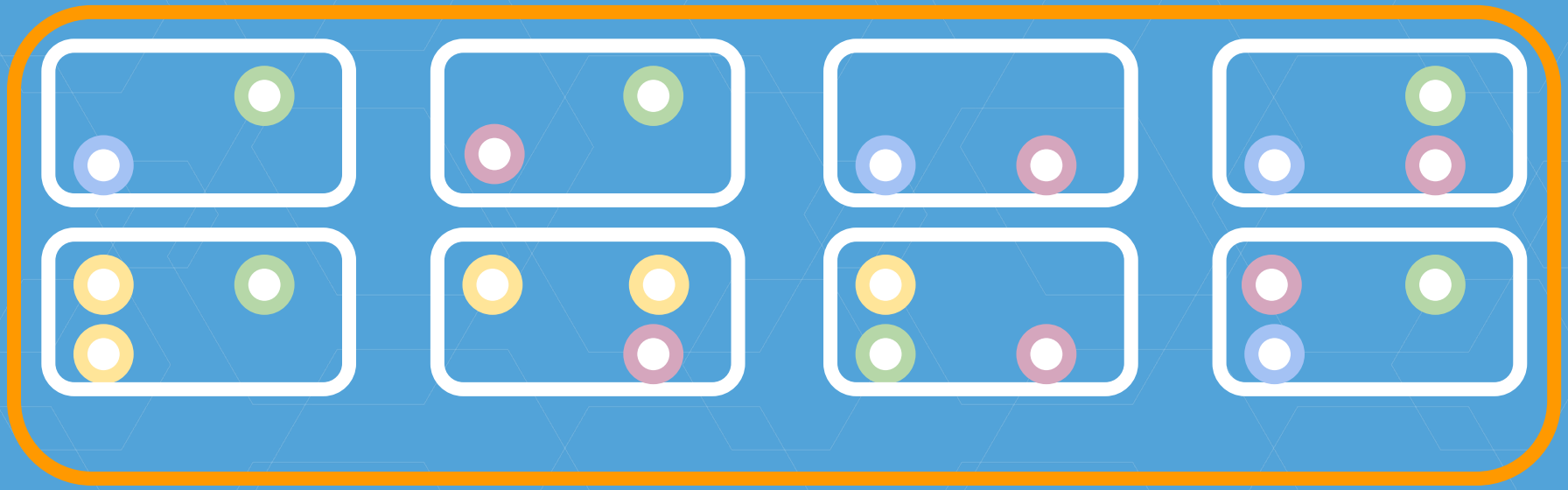
Scheduling



CoreOS

Discovery

color=yellow



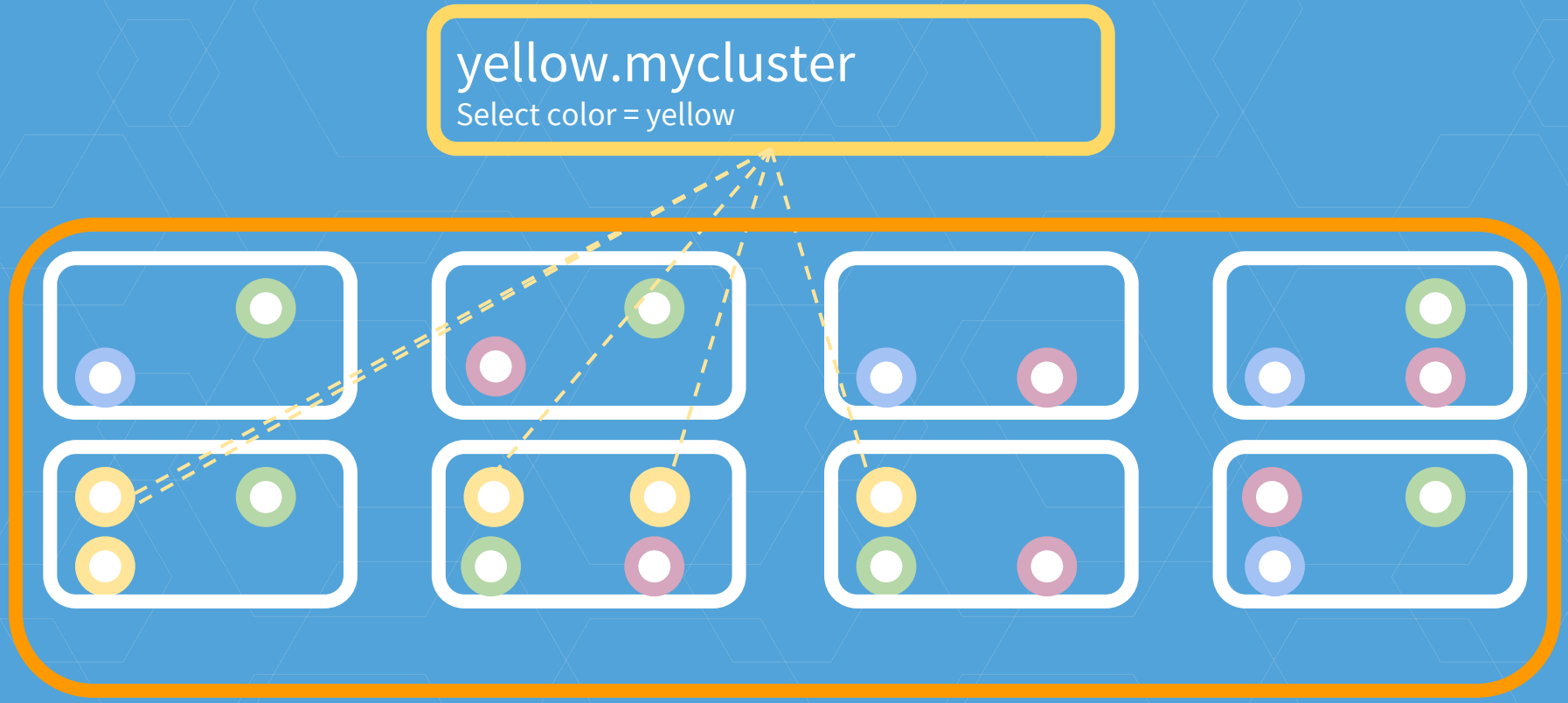
Discovery

color=yellow

Select color = yellow

CoreOS

Load balancing

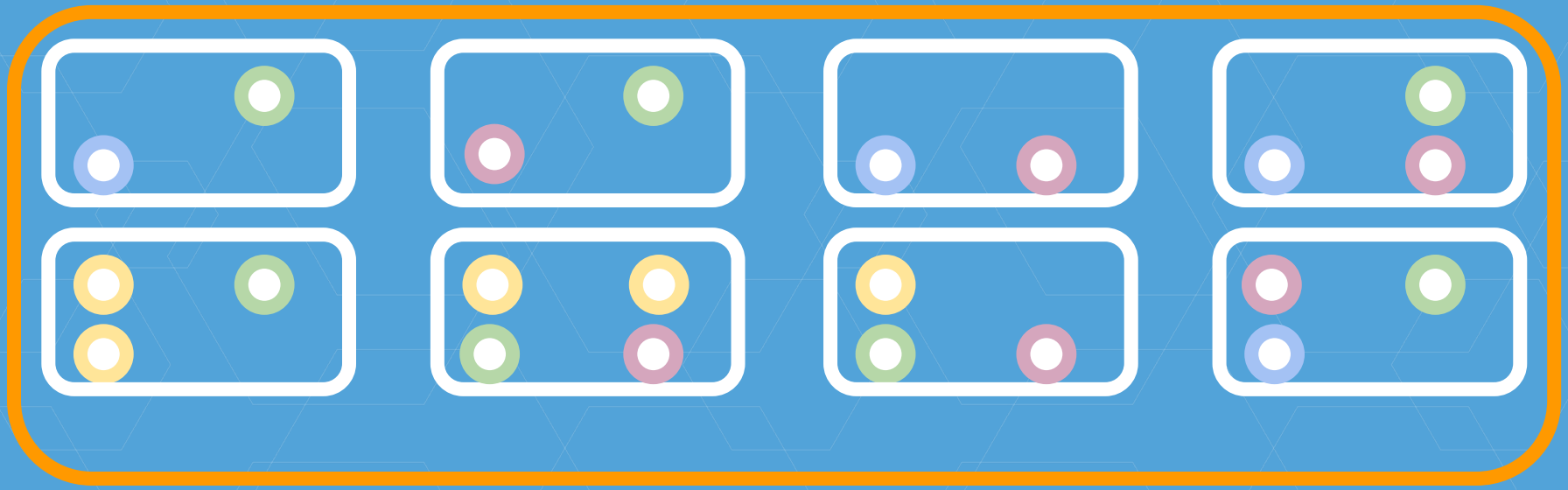


CoreOS

Healing

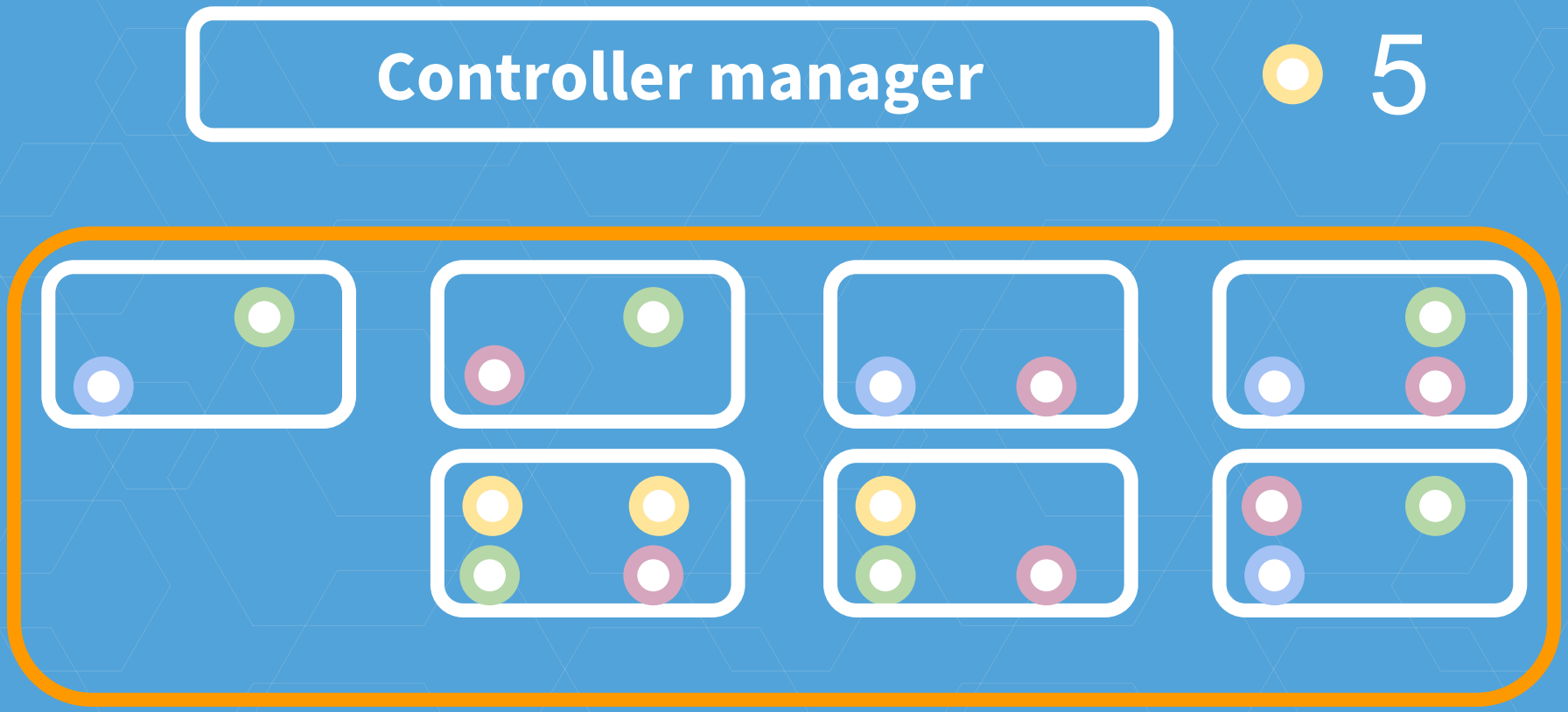
Controller manager

5



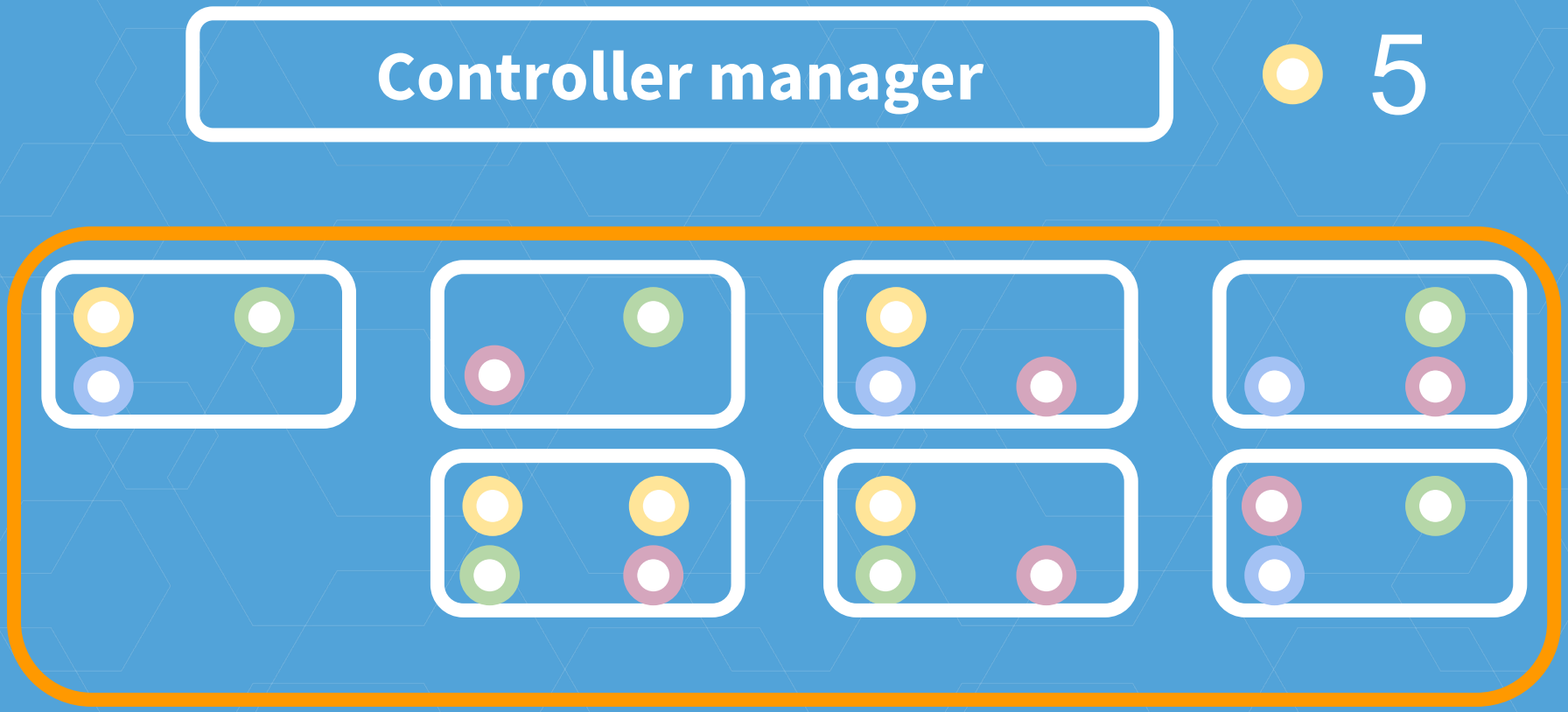
CoreOS

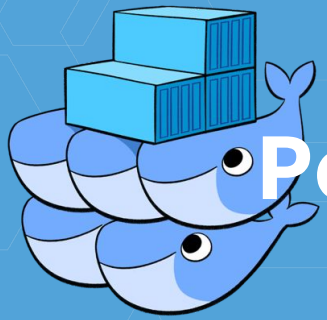
Healing



CoreOS

Healing





People love automation!

kubernetes

CoreOS

kubernetes

CoreOS

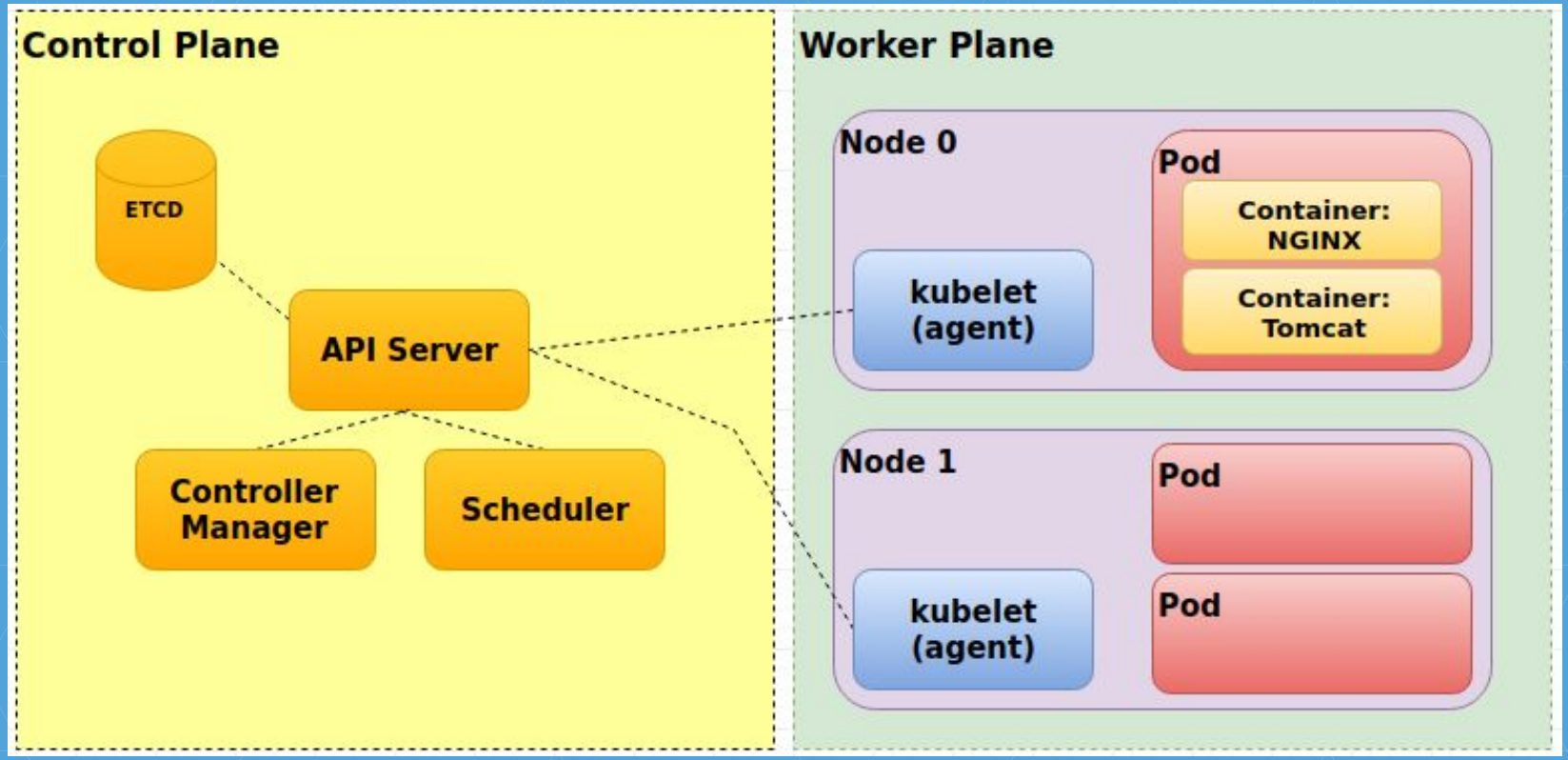
I hate Kubernetes!

CoreOS

I hate to OPERATE Kubernetes!

CoreOS

Kubernetes Architecture



CoreOS

Operating Kubernetes

- Installation
- Upgrade
- Healing
- Scaling
- Security
- Monitoring
- ...

CoreOS

Installation

- SSH
- Install kubelet
  - $pkgmanager install kubelet
- Install container runtime
  - $pkgmanager install [docker|rkt]
- Start kubelet
  - Systemctl start kubelet

CoreOS

Installation - master

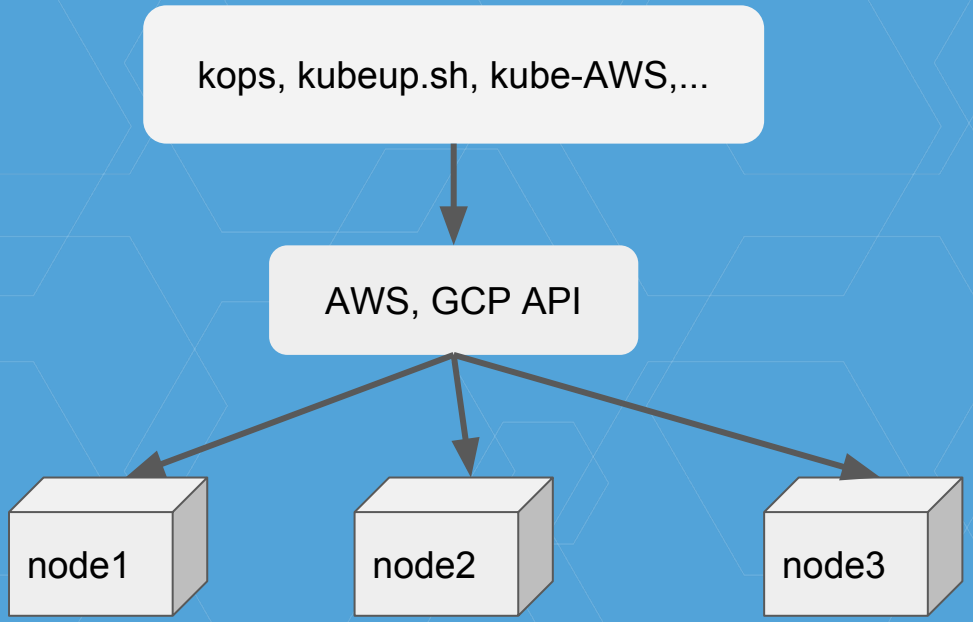
- SSH
- Install scheduler
- Install controller manager
- Install API server
- Config them correctly
- Start them

CoreOS

Installation - etcd

- SSH
- Install etcd
- Config them correctly
- Start them

Installation



CoreOS

Upgrade

- SSH
- Upgrade container runtime
- Upgrade Kubelet

CoreOS

Upgrade - master

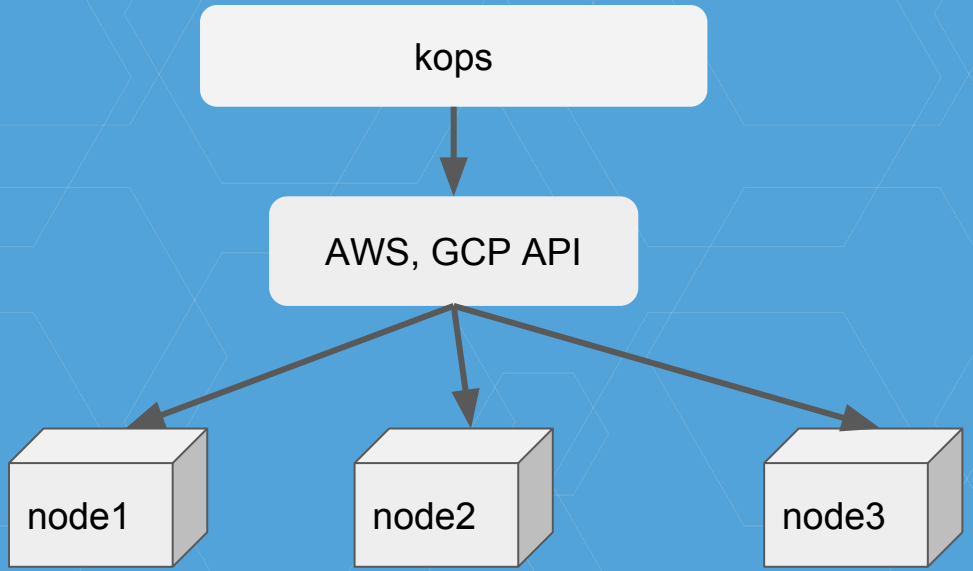
- SSH
- Upgrade master components

CoreOS

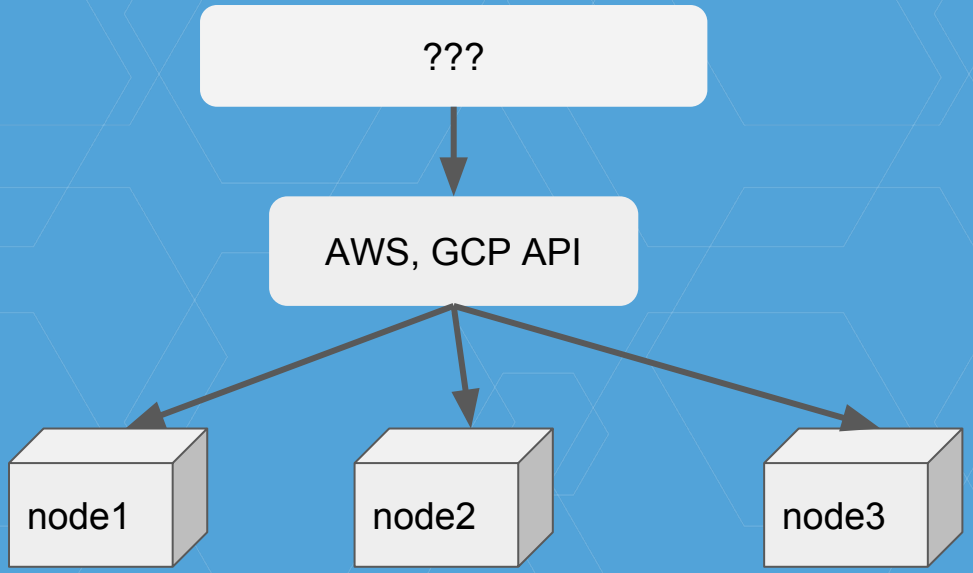
Upgrade - etcd

- SSH
- Upgrade etcd

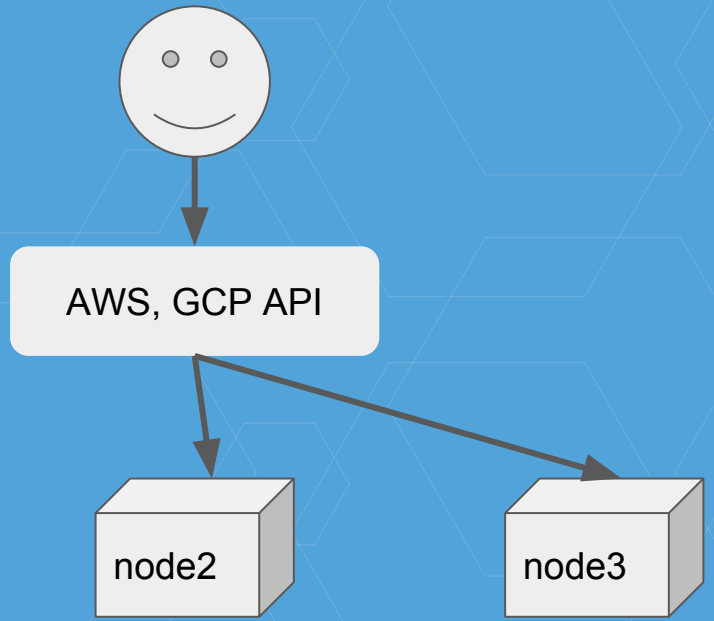
Upgrade



Rollback

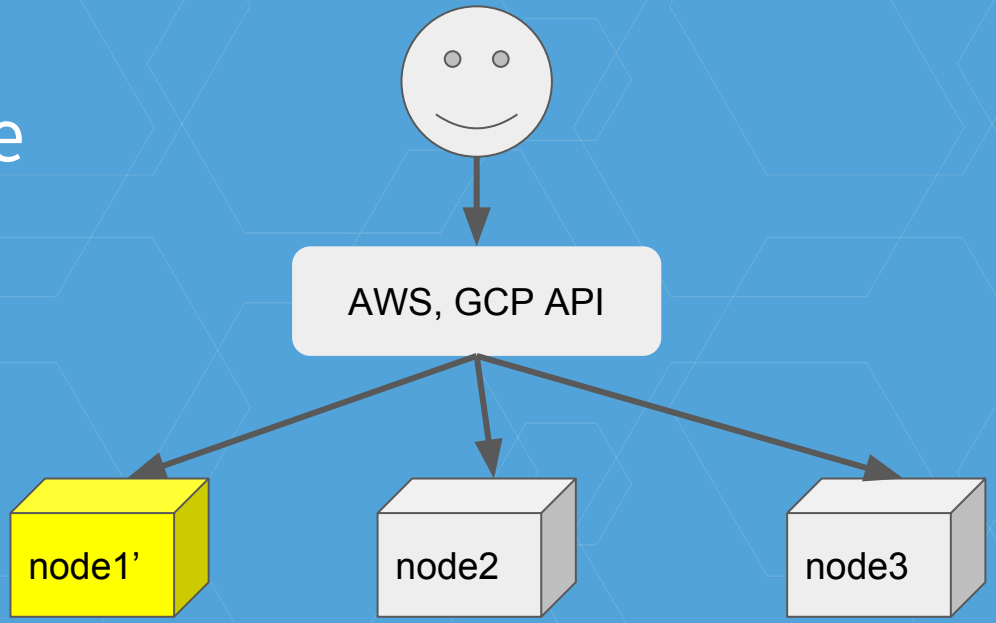


Healing



Healing

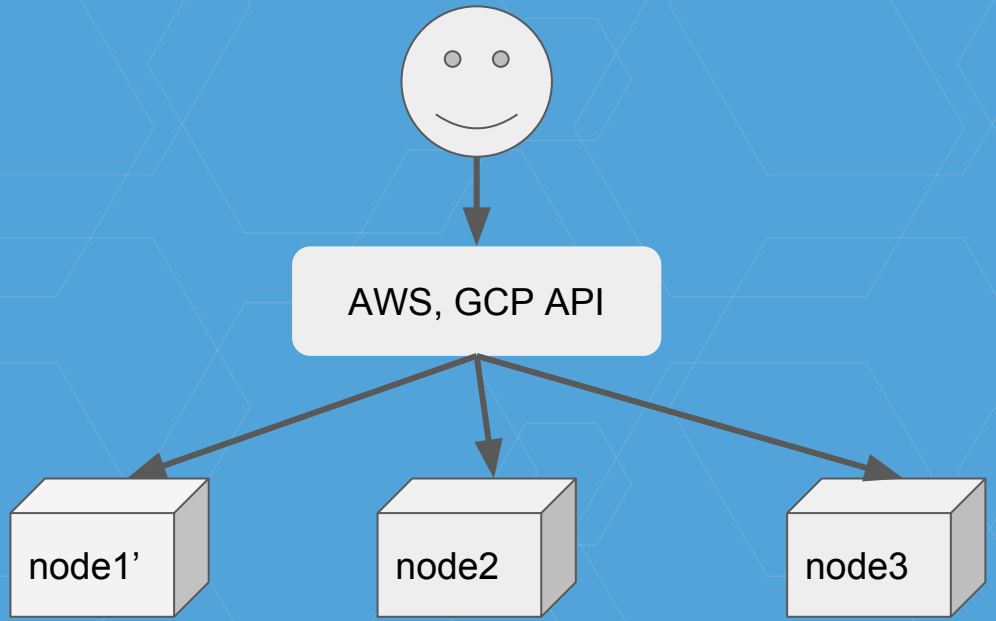
Create node



Healing

Install

Config



Problems

A lot of manual/semi-manual work
No standard way to approach all the problems

do it wrong, lose the cluster!

Self hosting

```
// gcc source code
#include <stdio.h>
int main()
{
    compile_c(argv[1]);
}
```

gcc  gcc

Self hosting

```
// golang source code
package main
import "os"

func main() {
  compile_go(os.Args[1:])
}
```

go  go

CoreOS

Self hosting

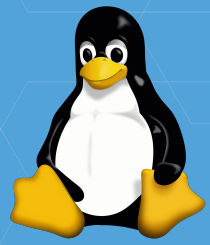
CoreOS

Self hosting

```
$ uname -s
minix
$ gcc linux.c
```

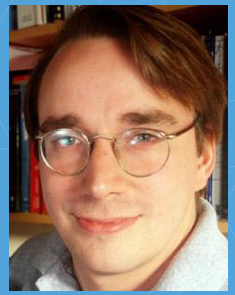
Self hosting

```
$ uname -s
minix
$ gcc linux.c
```



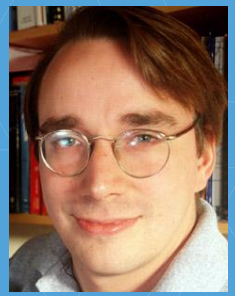
CoreOS

Self hosting



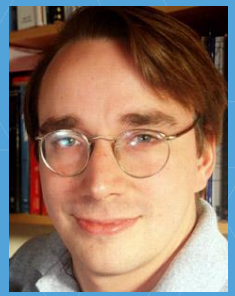
CoreOS

Self hosting

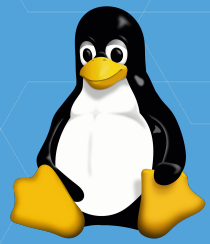


```
$ uname -s
linux
$ gcc linux.c
```

Self hosting



```
$ uname -s
linux
$ gcc linux.c
```



CoreOS

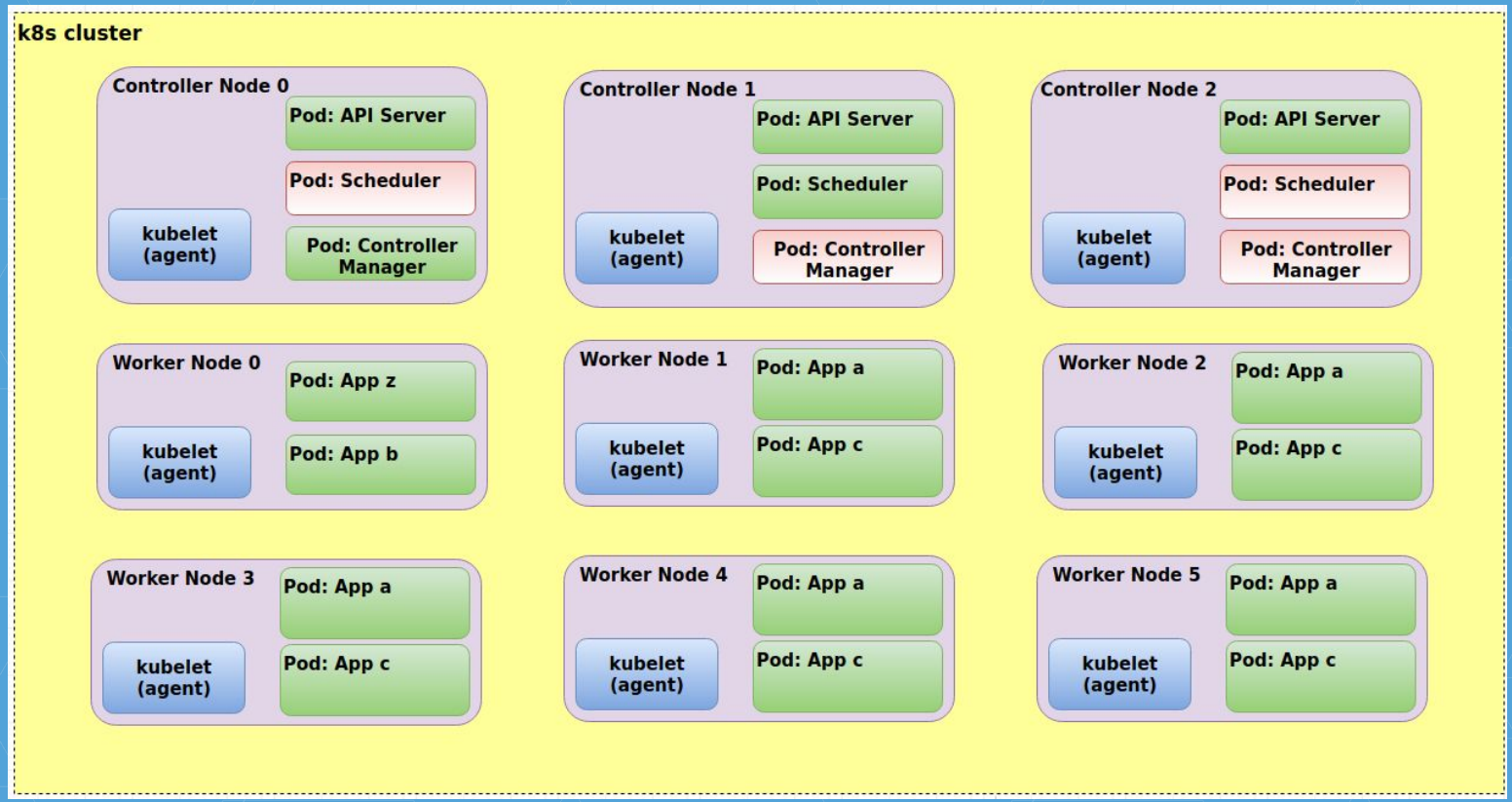
Self-hosted Kubernetes?

CoreOS

What is self-hosted Kubernetes?

- Kubernetes manages own core components
- Core components deployed as native API objects

Self-hosted k8s Architecture



CoreOS

Why Self-host Kubernetes?

- Operational expertise around app management in k8s extends to k8s itself
  - E.g. scaling
- Bootstrapping simplified
- Simply cluster life cycle management
  - E.g. updates
- Upstream improvements in Kubernetes directly translate to improvements in managing Kubernetes

CoreOS

Simplify Node Bootstrap

On-host requirements become:

- Kubelet
- Container Runtime (docker, rkt, …)

CoreOS

Any Distro Node Bootstrap

- Install kubelet
  - $pkgmanager install kubelet
- Install container runtime
  - $pkgmanager install [docker|rkt]
- Write kubeconfig
  - scp kubeconfig user@host:/etc/kubernetes/kubeconfig
- Start kubelet
  - Systemctl start kubelet

Simplify k8s lifecycle management

Manage your cluster with only kubectl

Upgrading a self-hosted Kubernetes cluster:

```
$ kubectl apply -f kube-apiserver.yaml
$ kubectl apply -f kube-scheduler.yaml
$ kubectl apply -f kube-controller-manager.yaml
$ kubectl apply -f kube-proxy.yaml
```

CoreOS

Launching a self-hosted cluster

Need an initial control plane to bootstrap a self-hosted cluster

Bootkube:

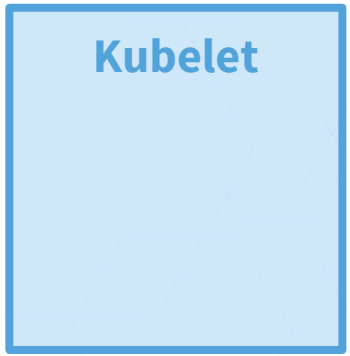
- Acts as a temporary control plane long enough to be replaced by a self-hosted control plane.
- Run only on very first node, then not needed again.

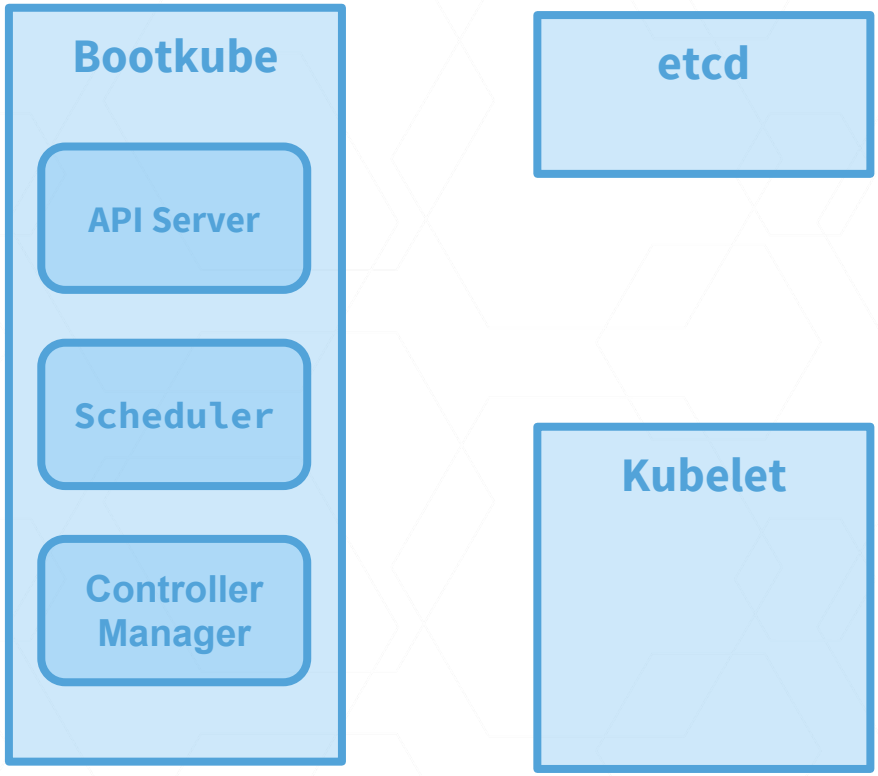
github.com/kubernetes-incubator/bootkube

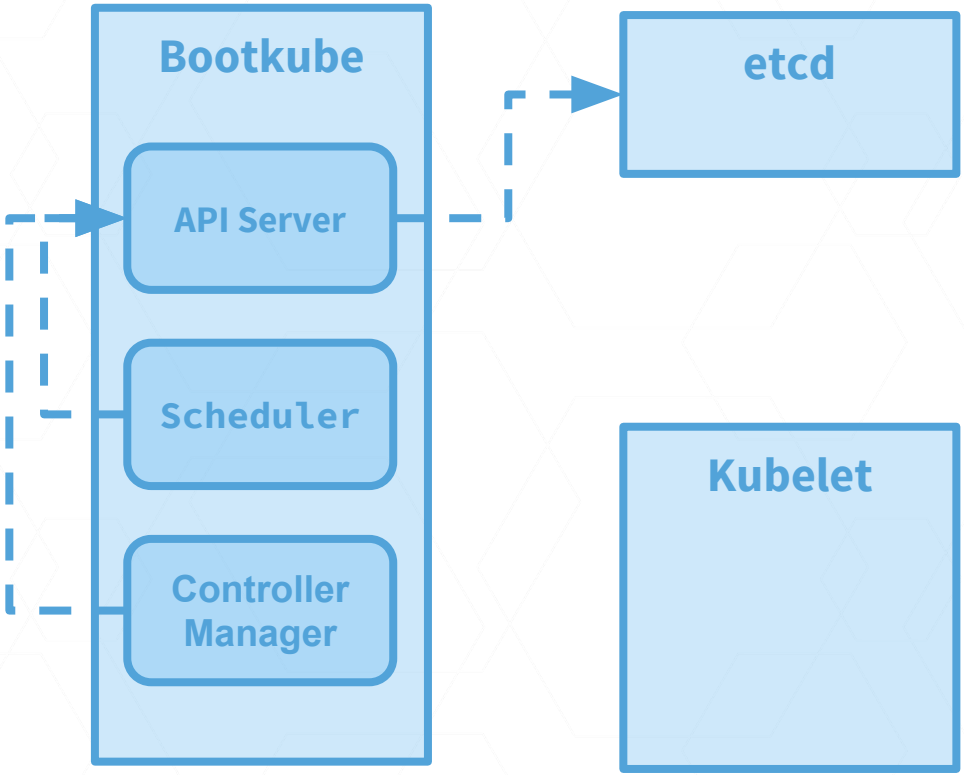
CoreOS

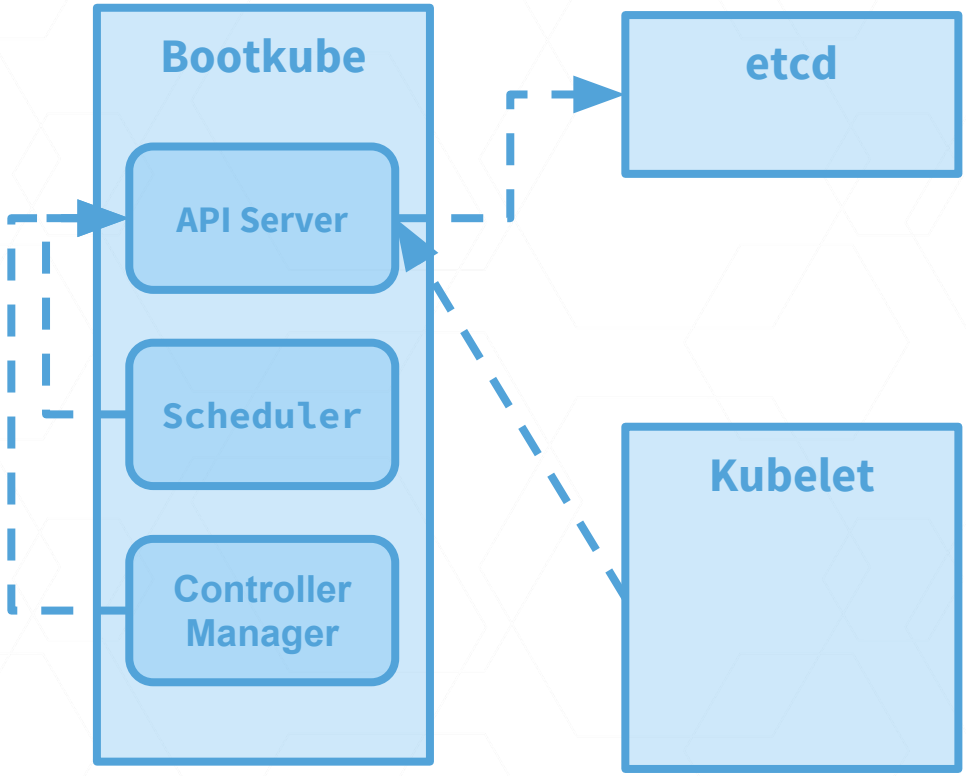
How Bootkube Works

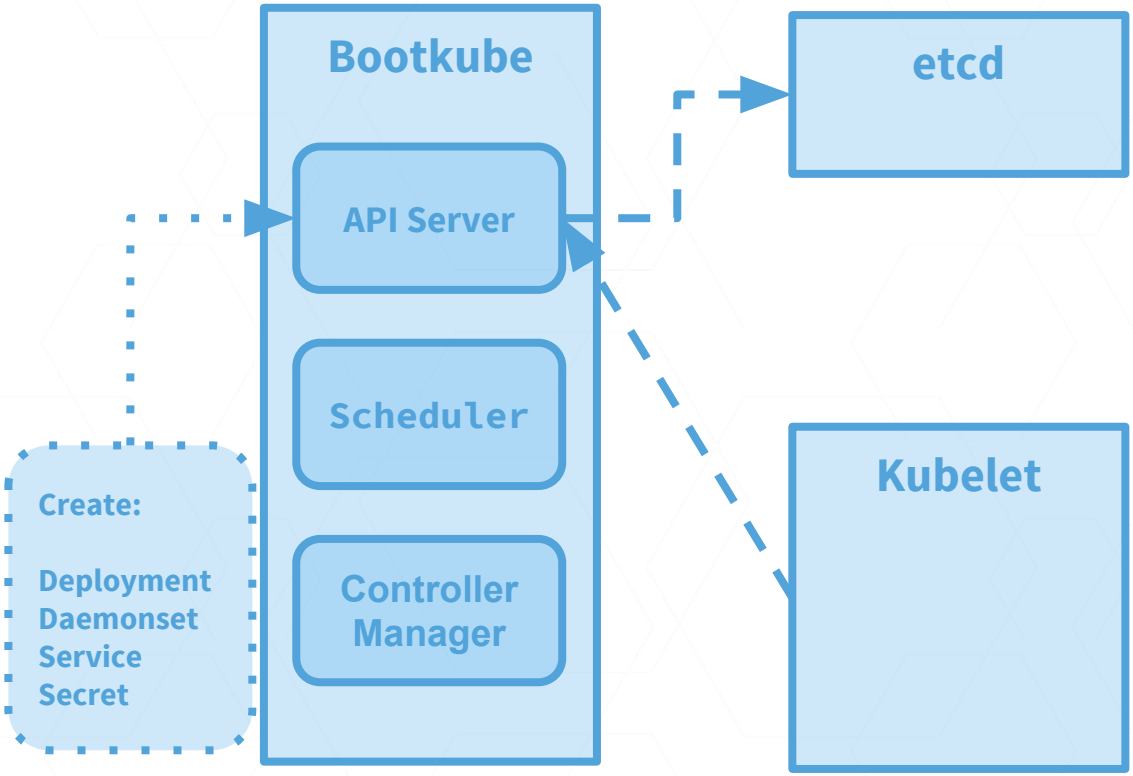
CoreOS

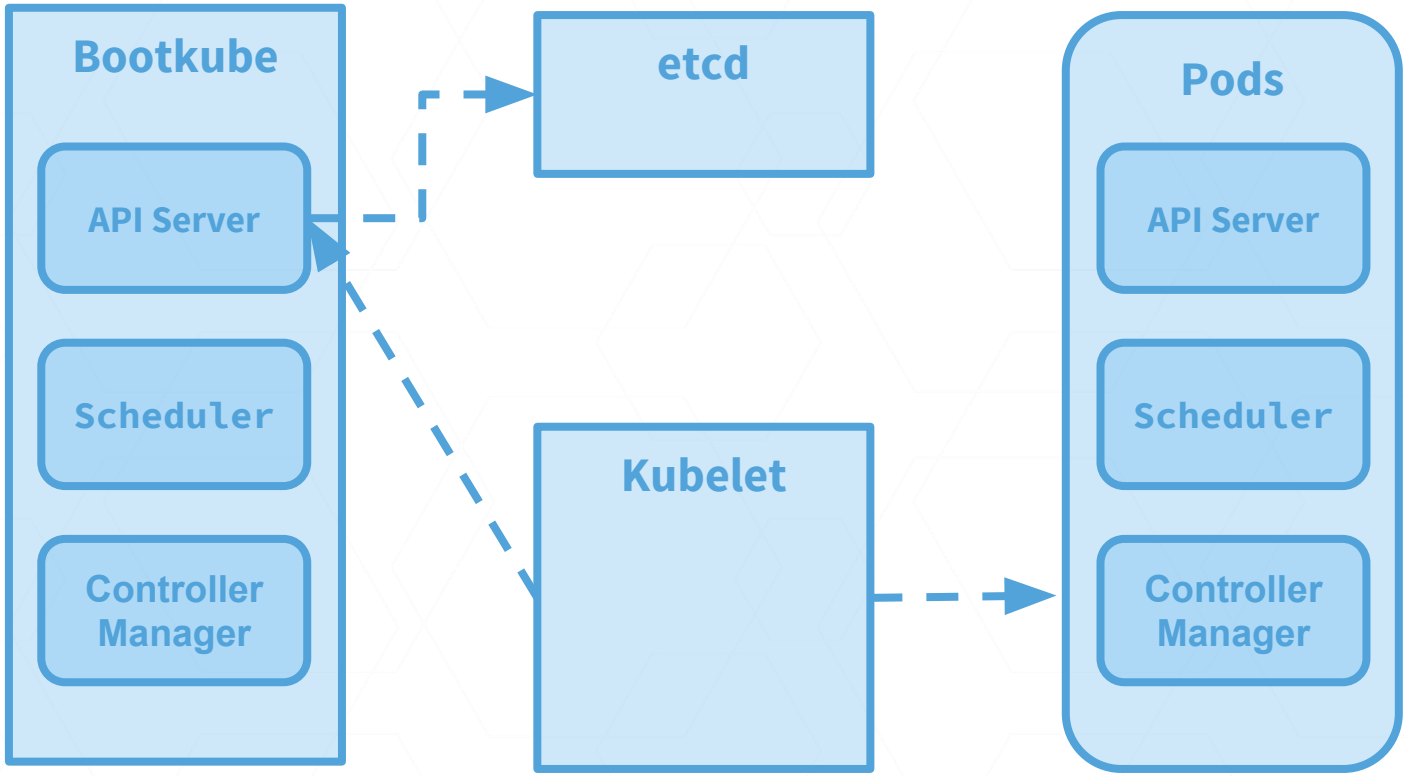


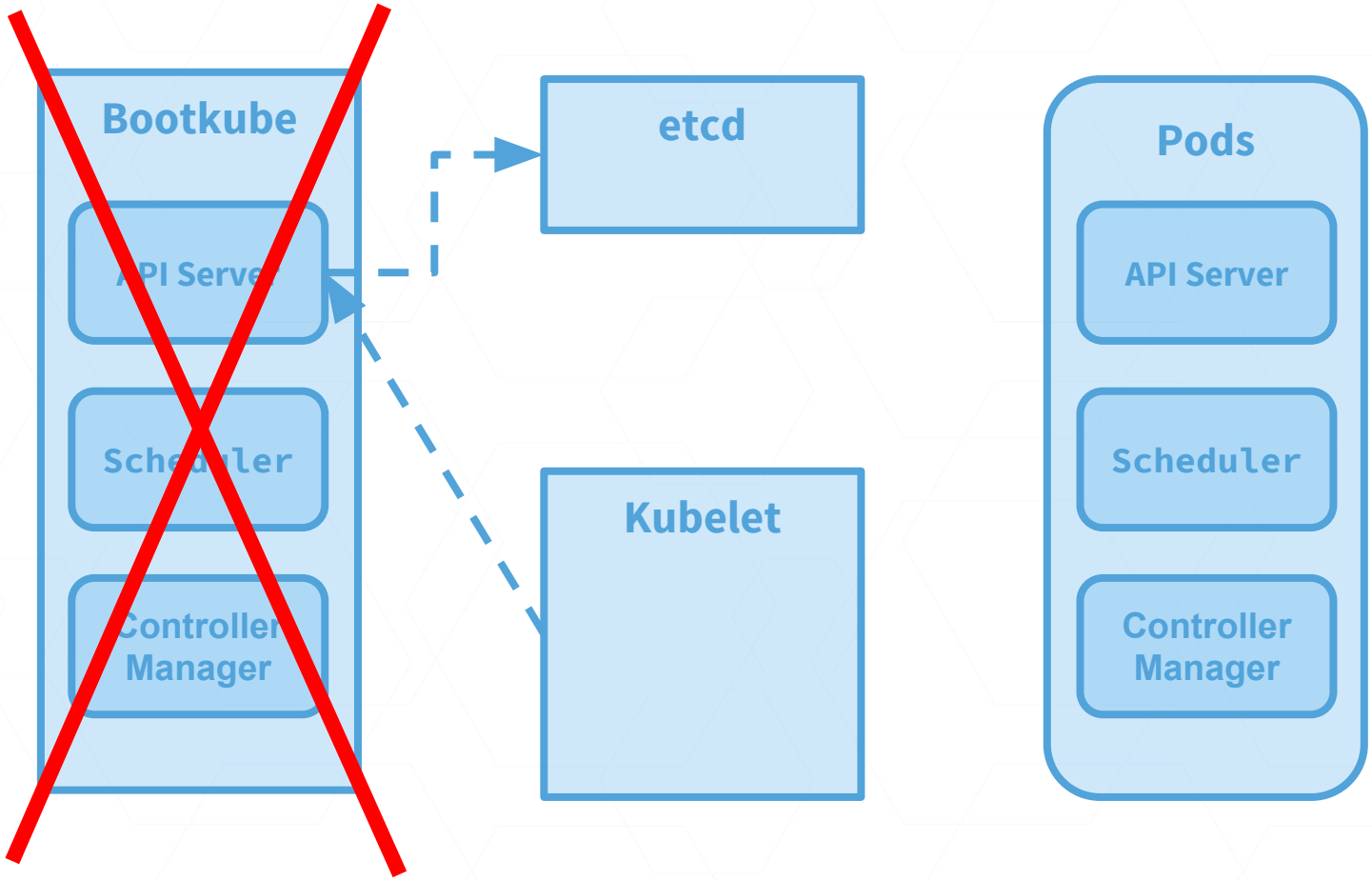

Bootkube
API Server
Scheduler
Controller Manager
etcd
Kubelet


Bootkube
API Server
Scheduler
Controller Manager
etcd
Kubelet

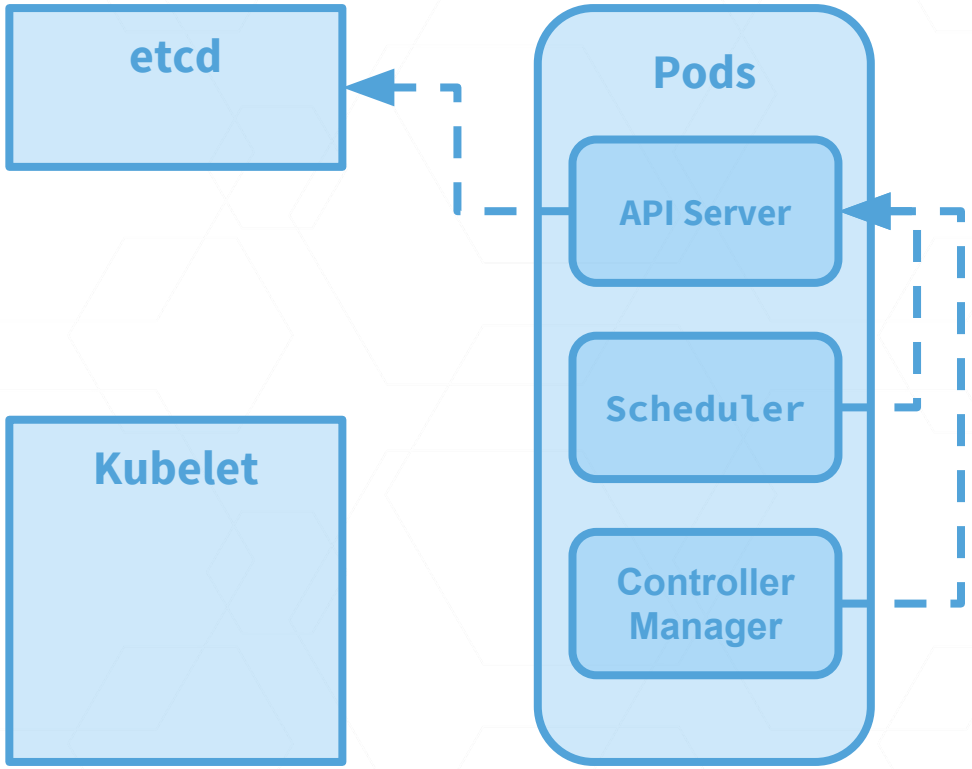

Bootkube
API Server
Scheduler
Controller Manager
etcd
Kubelet


Bootkube
API Server
Scheduler
Controller Manager
etcd
Kubelet
Create:
Deployment
Daemonset
Service
Secret

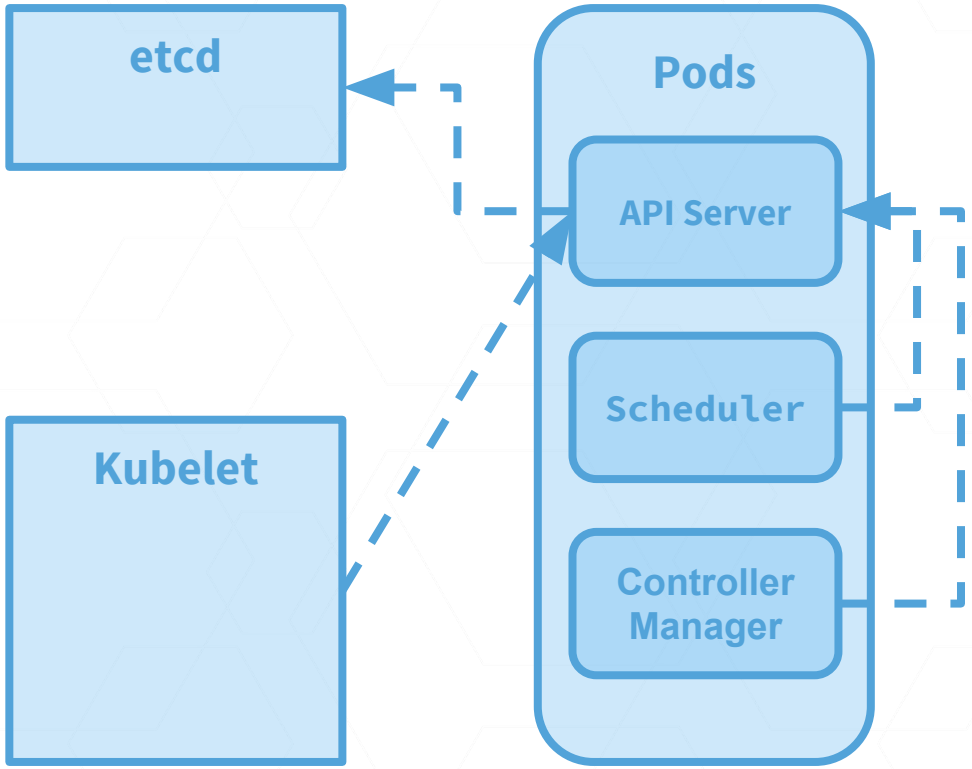

Bootkube
API Server
Scheduler
Controller Manager
etcd
Kubelet
Pods
API Server
Scheduler
Controller Manager


Bootkube
API Server
Scheduler
Controller Manager
etcd
Kubelet
Pods
API Server
Scheduler
Controller Manager

CoreOS


etcd
Pods
API Server
Scheduler
Controller Manager
Kubelet

CoreOS


etcd
Pods
API Server
Scheduler
Controller Manager
Kubelet

CoreOS

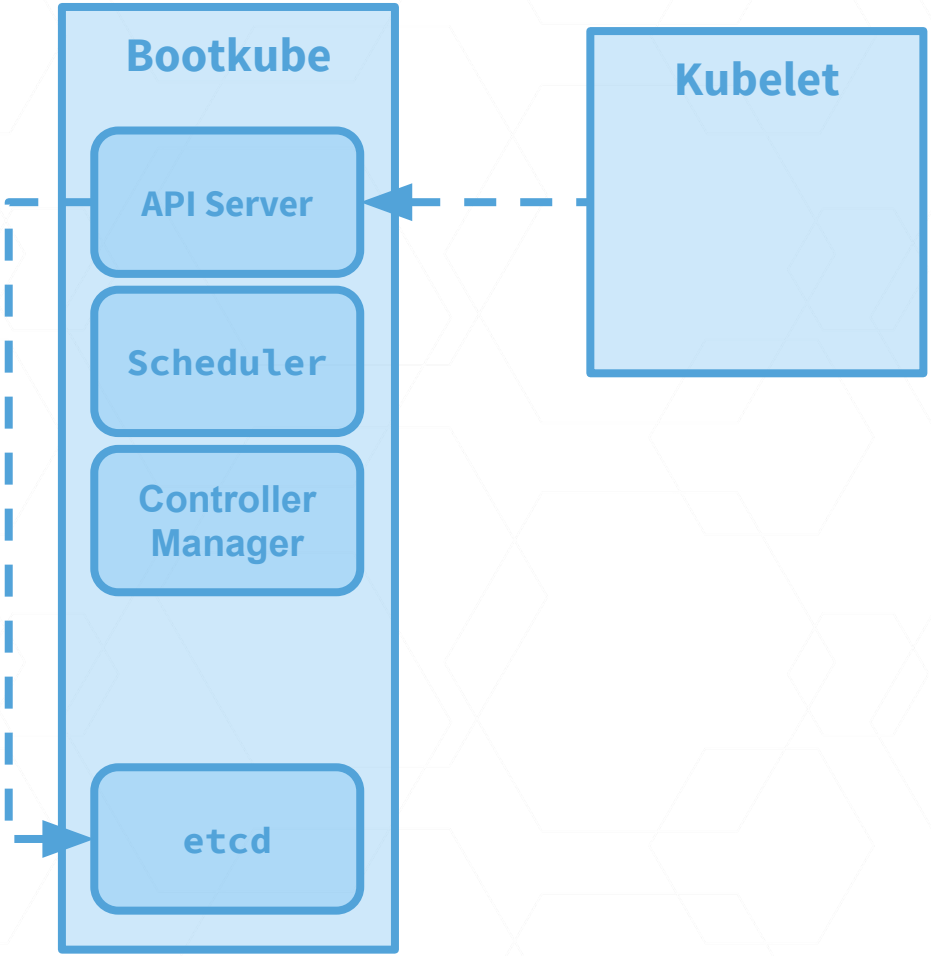
But wait! There's more!

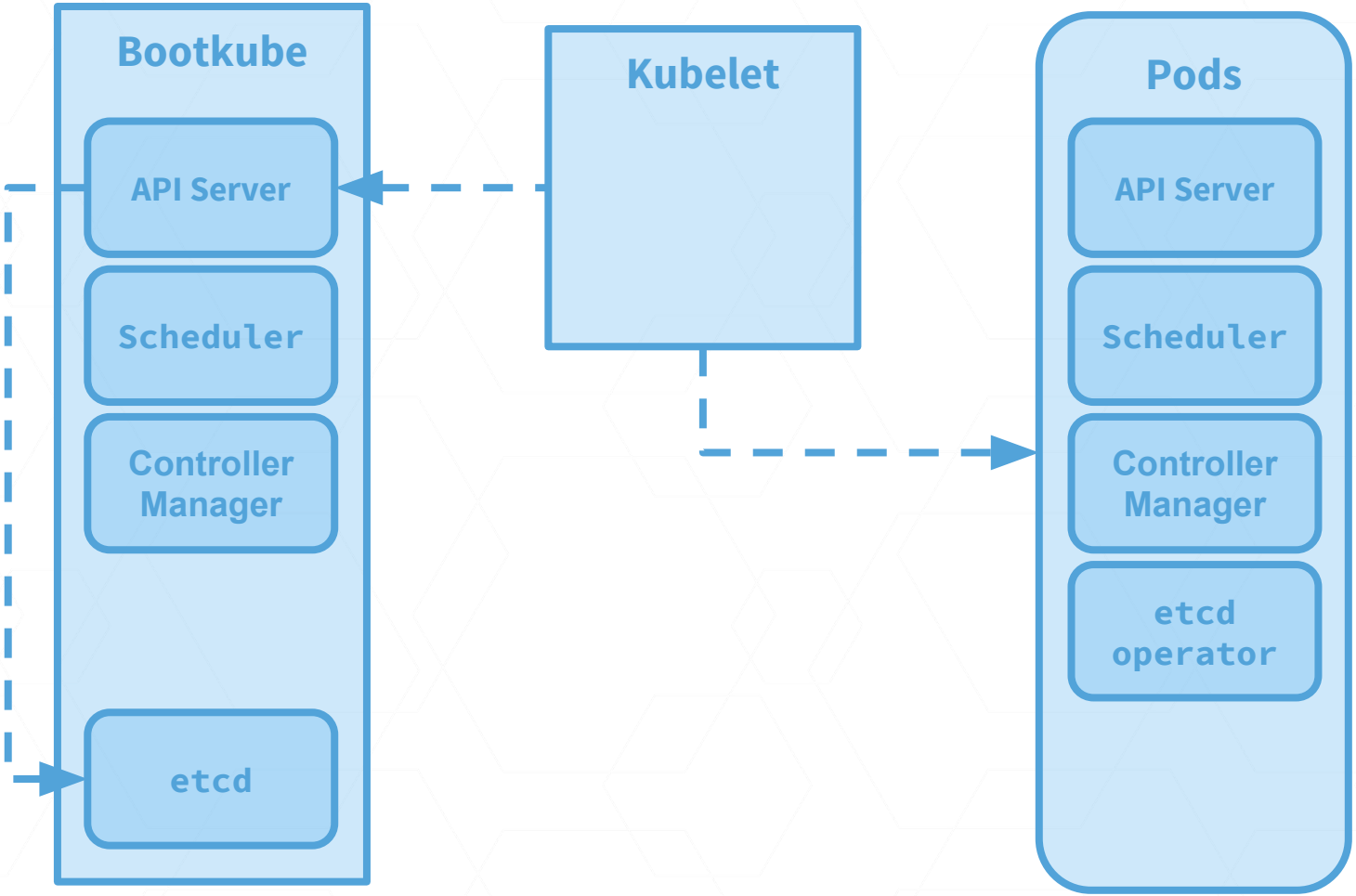
You can even self-host etcd!

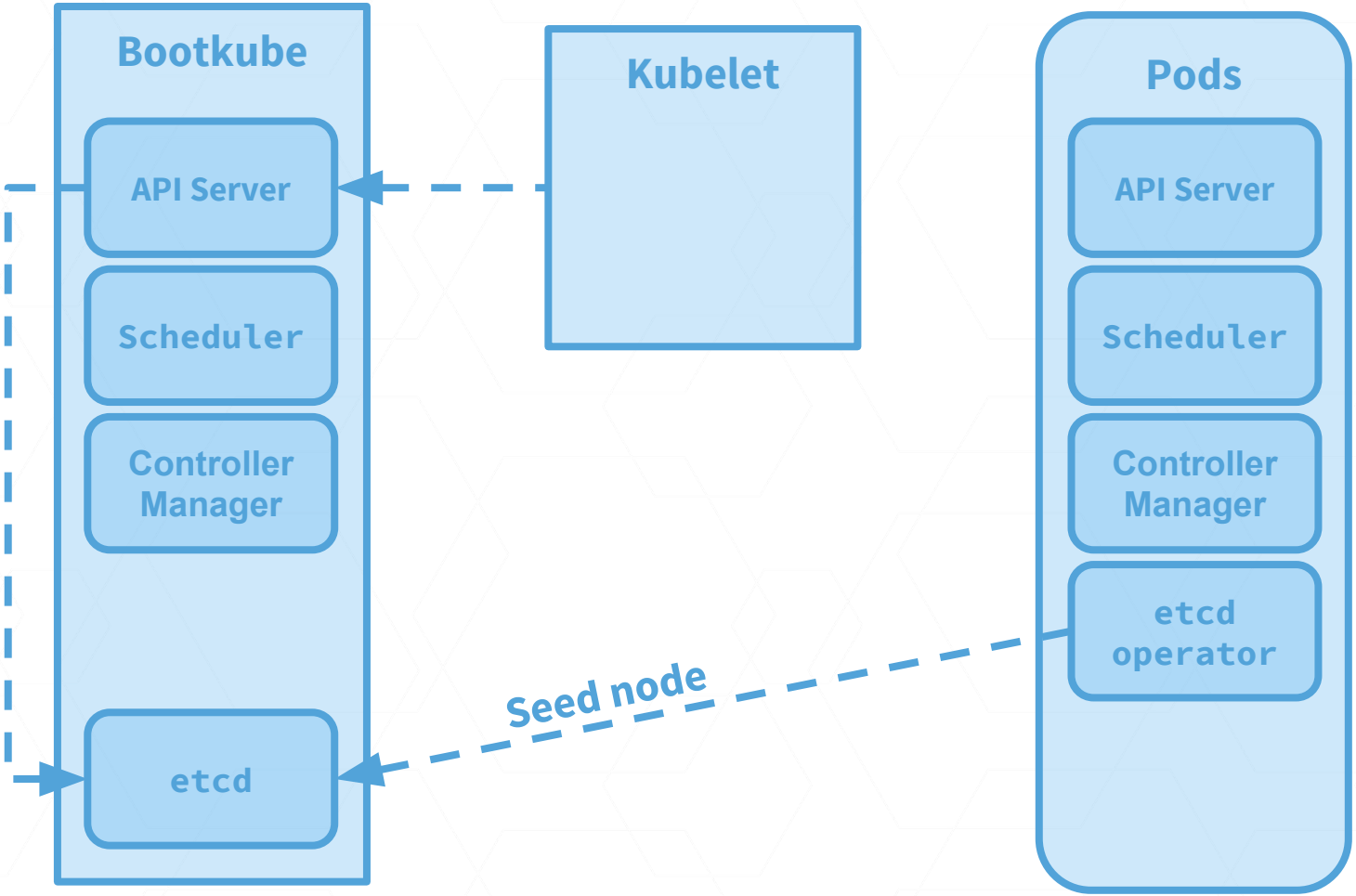
https://coreos.com/blog/introducing-the-etcd-operator.html
https://github.com/coreos/etcd-operator

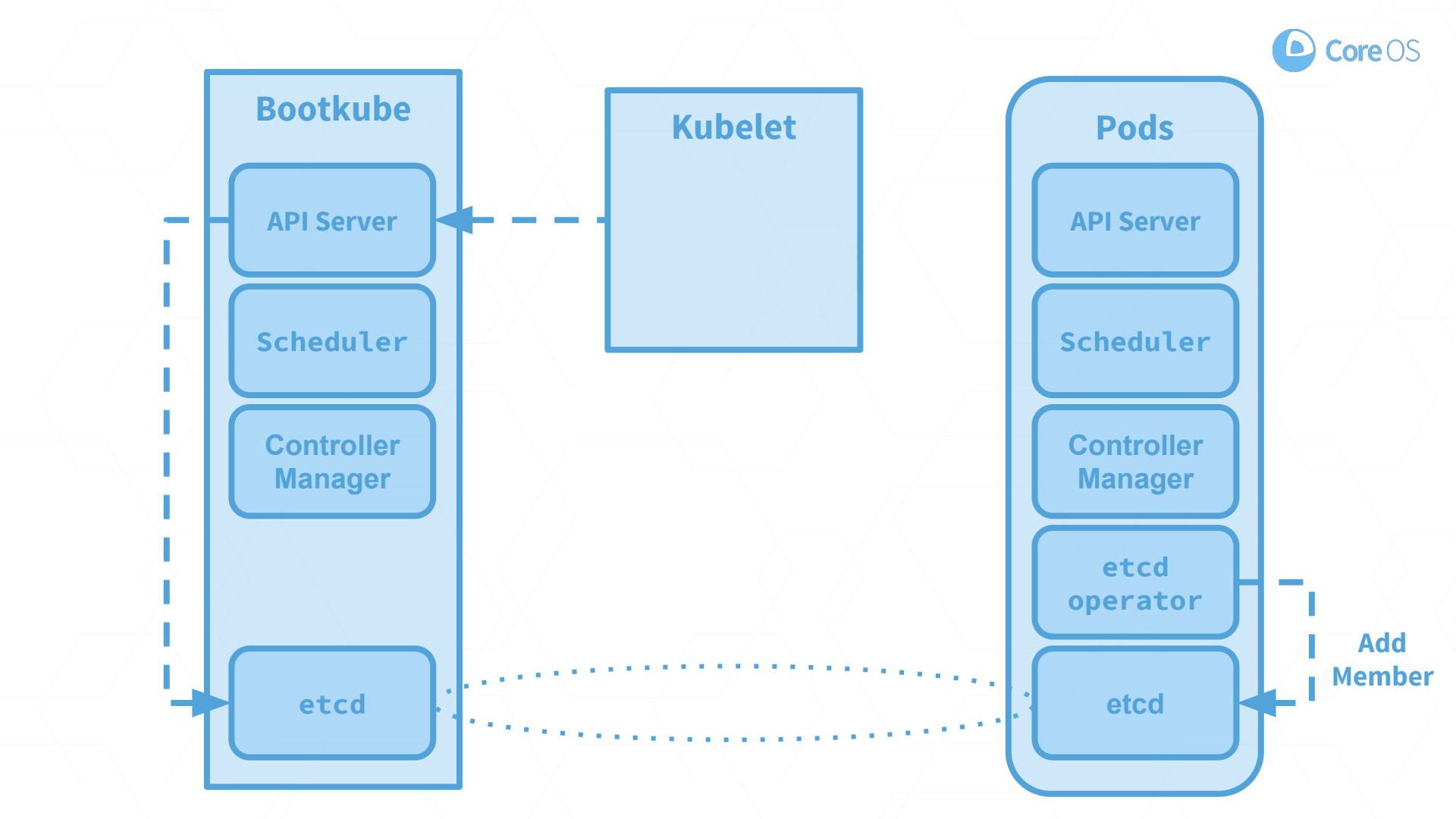
CoreOS

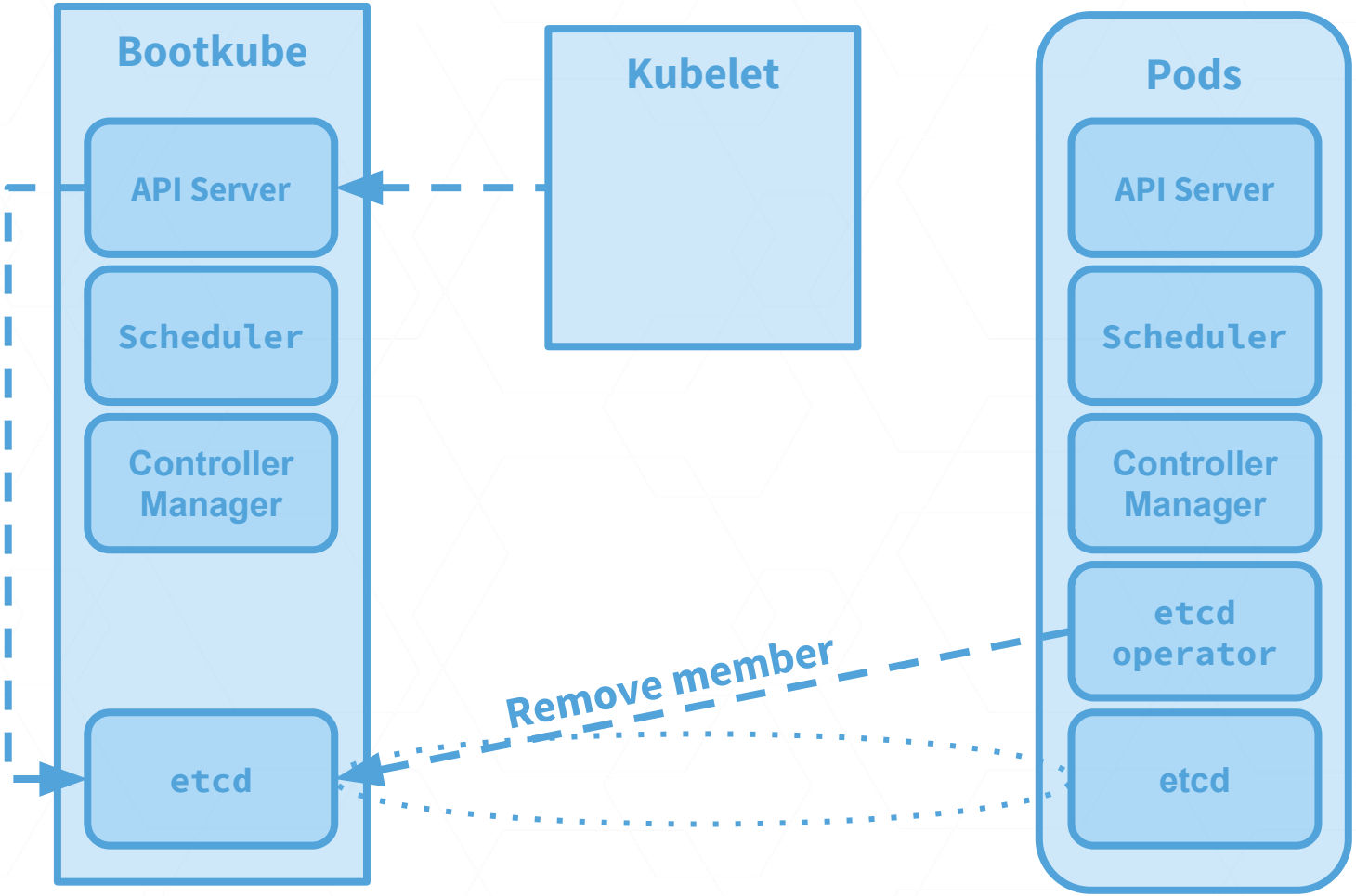
How to bootstrap self-hosted etcd


Bootkube
API Server
Scheduler
Controller Manager
etcd
Kubelet

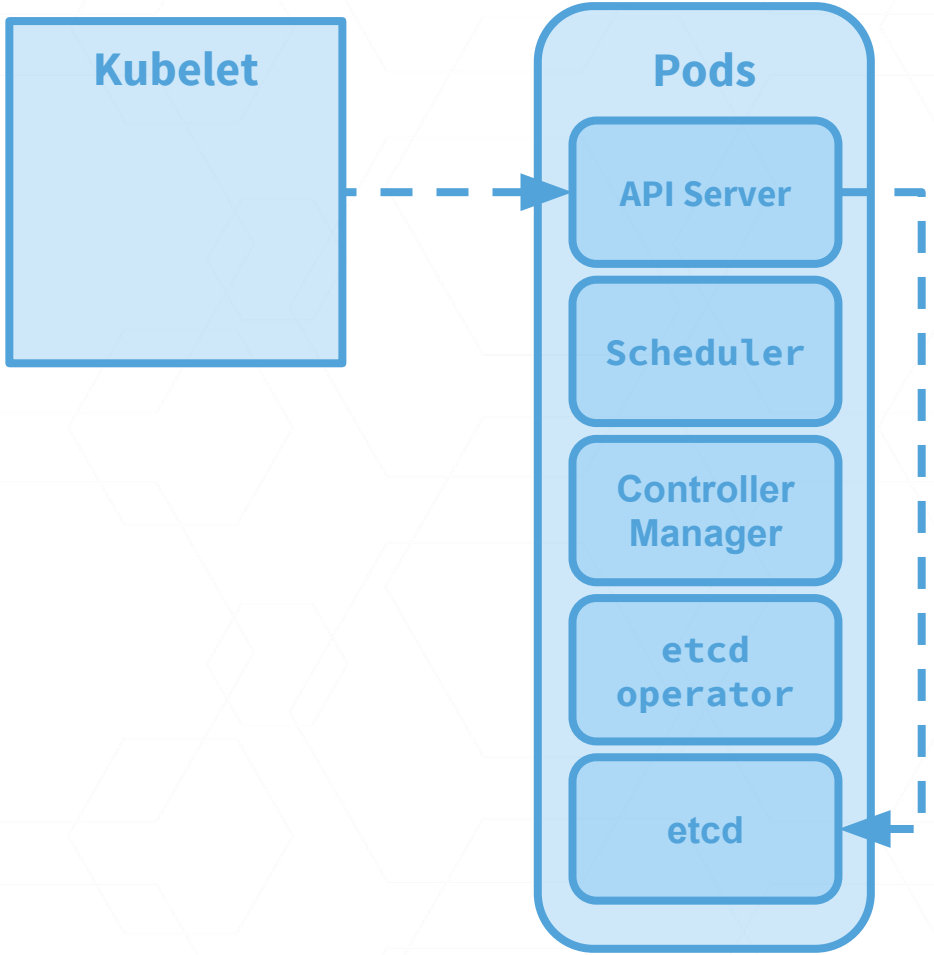

Bootkube
API Server
Scheduler
Controller Manager
etcd
Kubelet
Pods
API Server
Scheduler
Controller Manager
etcd operator


Bootkube
API Server
Scheduler
Controller Manager
etcd
Kubelet
Pods
API Server
Scheduler
Controller Manager
etcd operator
Seed node


CoreOS
Bootkube
API Server
Scheduler
Controller Manager
etcd
Kubelet
Pods
API Server
Scheduler
Controller Manager
etcd operator
etcd
Add Member


Bootkube
API Server
Scheduler
Controller Manager
etcd
Kubelet
Pods
API Server
Scheduler
Controller Manager
etcd operator
etcd
Remove member

CoreOS


Kubelet
Pods
API Server
Scheduler
Controller Manager
etcd operator
etcd

CoreOS

Disaster Recovery

Node failure in HA deployments (Kubernetes)

Partial loss of control plane components (Kubernetes)

Power cycling the entire control plane (Kubernetes)

Permanent loss of control plane (External tool)

CoreOS

Disaster Recovery

Permanent loss of control plane

- Similar situation to initial node bootstrap, but utilizing existing etcd state or etcd backup.
- Need to start a temporary replacement api-server
  - Could be binary, static pod, new tool, bootkube, etc.
- Recovery once etcd+api is available can be done via kubectl (as seen previously)

CoreOS

Self-Driving Kubernetes

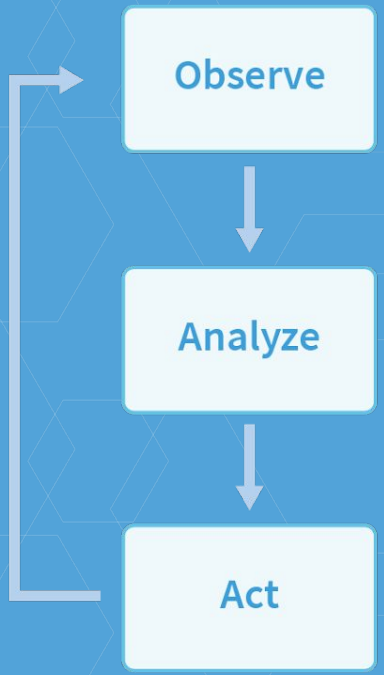
CoreOS

Self driving

- A self-hosted cluster launched via Bootkube
- Upgraded via Kubernetes APIs and an Operator
- Automated by single-button or fully automatic

Kubernetes Version Operator

CoreOS



Cluster is running v1.4.3 and configured to run v1.4.5

- API Server is v1.4.3
- Scheduler is v1.4.3

Differences from desired config

- API Server should be v1.4.5
- Scheduler should be v1.4.5

How to get there

- Upgrade all API servers Daemons to v1.4.5 safely one-by-one
- Upgrade all Scheduler Deployments to v1.4.5
- Update status to v1.4.5

CoreOS

The infrastructure

Workload driven

Automation driven

Easy to manage: self driving approach (Today's topic)

Security focused

Xiang Li

xiang.li@coreos.com

Thank you!