etcd

the cornerstore of distributed systems
(built using Go)

Jonathan Boulle

CoreOS

@baronboulle

What is etcd?

/etc
distributed

hence, the name…

a clustered
key-value store

GET and SET operations

a **building block** for higher order systems

primitives for building reliable distributed systems

What is etcd?

- distributed `/etc`
  - cluster-level configuration
- clustered key-value store
- primitives for building reliable distributed systems
  - distributed locking system
  - distributed scheduling system
- supports a lot of large distributed applications
  - SkyDNS, Kubernetes, CloudFoundry, ...

History of etcd

History of etcd

- 2013.8 Alpha release
  - v0.1-v0.4
- 2015.2 Stable release
  - v2.0+
  - stable replication engine (new Raft implementation)
  - stable v2 API
- 2016.?
  - v3.0
  - efficient, powerful API
  - highly scalable backend

History of etcd

- Production-ready!
  - long running failure injection tests
  - no known data loss issues
  - no known inconsistency issues
  - used in critical CoreOS systems like locksmith and fleet
  - trusted by Google, Pivotal, compose and many more!

Why build etcd?

Why build etcd?

- CoreOS: "Secure the internet"
- Updating servers = rebooting servers
- Move towards new application container paradigm
- Need a:
  - shared configuration store (for service discovery etc)
  - distributed lock manager (to co-ordinate reboots)
- Existing solutions were inflexible (undocumented binary API), difficult to configure

Why use etcd?

Why use etcd?

- Highly available
- Highly reliable
- Strong consistency guarantees
- Simple, fast HTTP API
- Open source

"For the most critical data of a distributed system"

How does etcd work?

How does etcd work?

- Replicated log to model a state machine
- Raft: *"In Search of an Understandable Consensus Algorithm"* (Ongaro, 2014)
- Three key concepts
  - Leaders
  - Elections
  - Terms
- etcd clusters elect a leader; all state changes performed by that leader

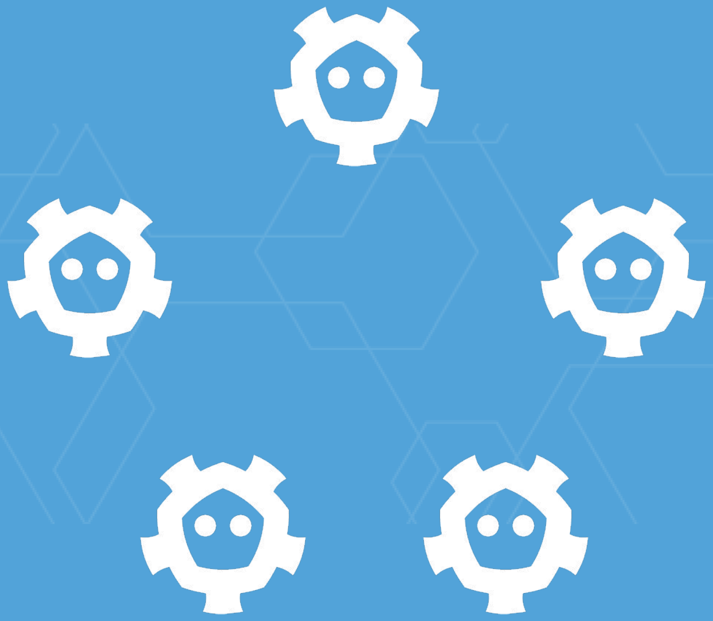
How does etcd work?

- Go
- `/bin/etcd`
  - daemon
  - 2379 (client requests/HTTP + JSON API)
  - 2380 (peer-to-peer/HTTP + protobuf)
- `/bin/etcdctl`
  - command line client
  - `net/http, encoding/json, ...`

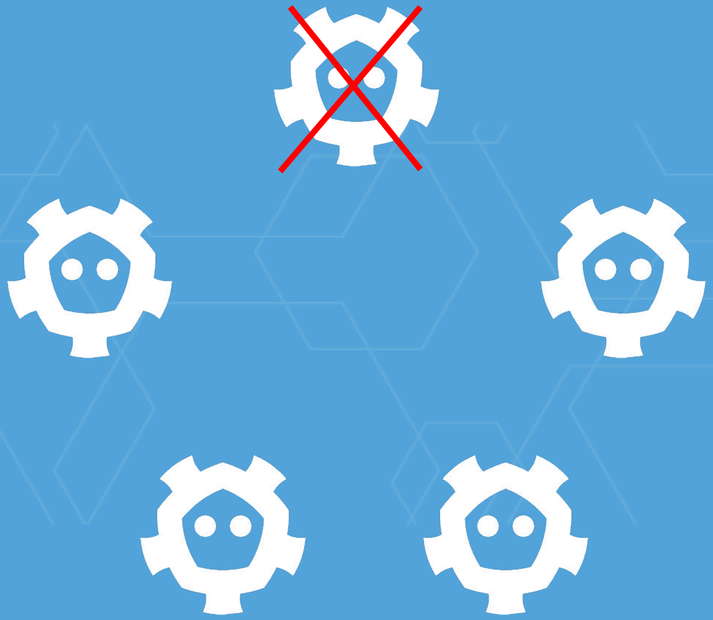
etcd basics

clusters

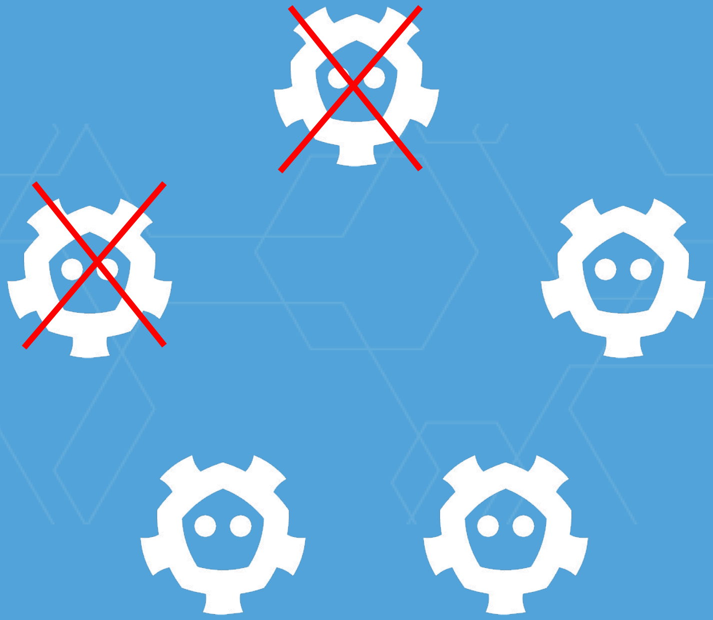
Available



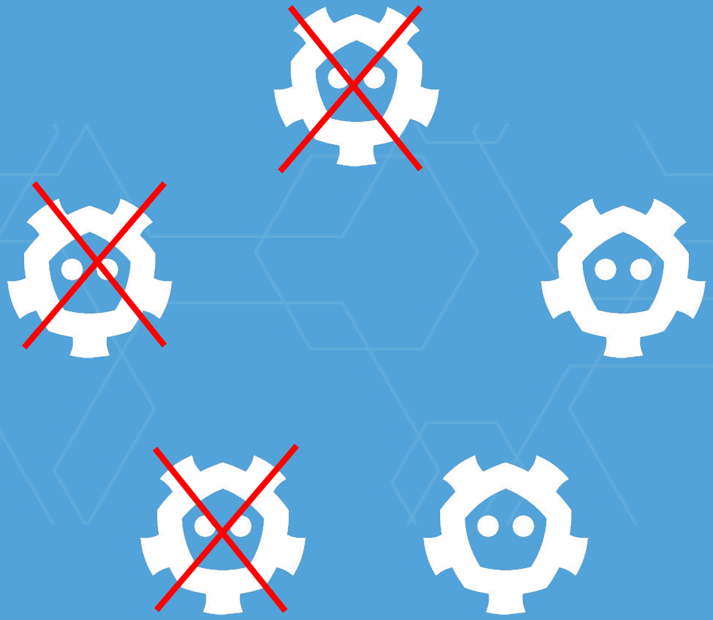
Available



Available



Unavailable



etcd basics

API

Simple HTTP API (v2)

- `GET /v2/keys/foo`
  - Get the value of a key
- `GET /v2/keys/foo?wait=true`
  - Wait for changes on key foo
- `PUT /v2/keys/foo -d value=bar`
  - Set the value of a key
- `DELETE /v2/keys/foo`
  - Delete a key

Compare-and-Swap

```
PUT /v2/keys/foo?prevValue=bar -d
value=ok
```

```
CAS(/foo, bar, ok)

if /foo == bar
   set(/foo, ok)
else
   do nothing
```

Compare-and-Delete

```
DELETE /v2/keys/foo?prevValue=bar
```

```
CAD(/foo, bar)

if /foo == bar
   delete(/foo)
else
   do nothing
```

Simple HTTP API (v2)

Native Go bindings

```
import "github.com/coreos/etcd/client"


cl := client.New(client.Config{})
kapi := client.NewKeysAPI(cl)
kapi.Set("foo", "bar", ...)
```

etcd apps

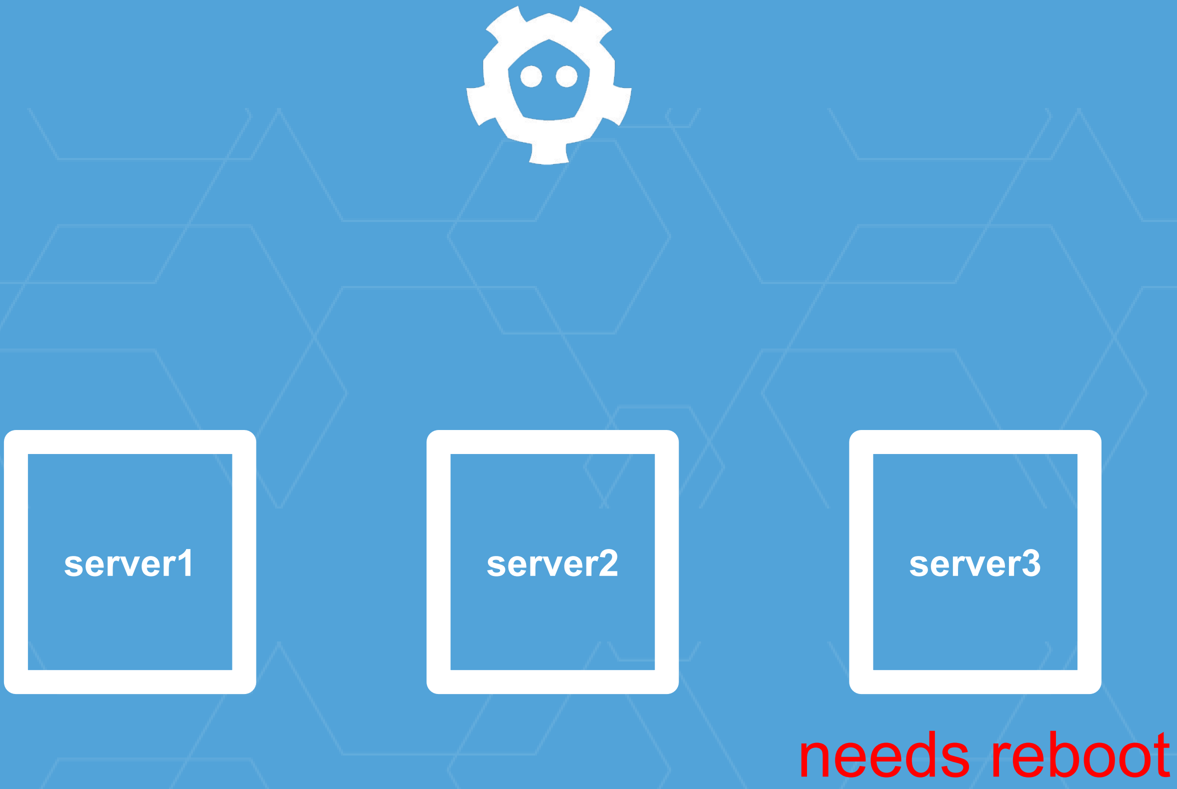
etcd apps

locksmith

locksmith

- cluster wide reboot lock
    - "semaphore for reboots"
- CoreOS updates happen *automatically*
    - stop all the machines restarting at once...

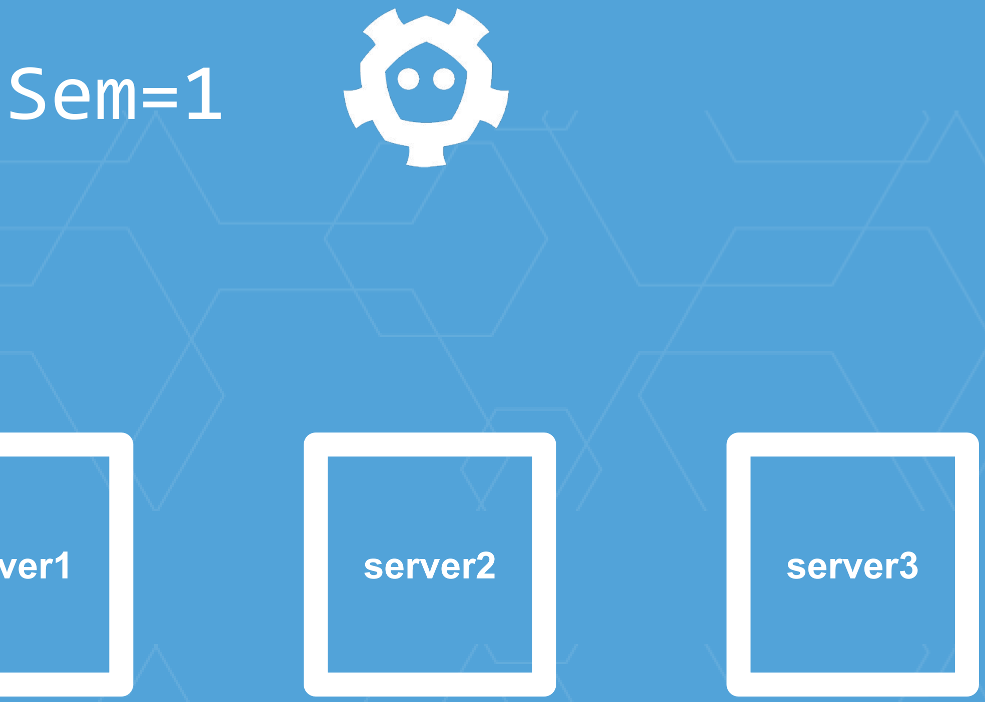
Cluster Wide Reboot Lock



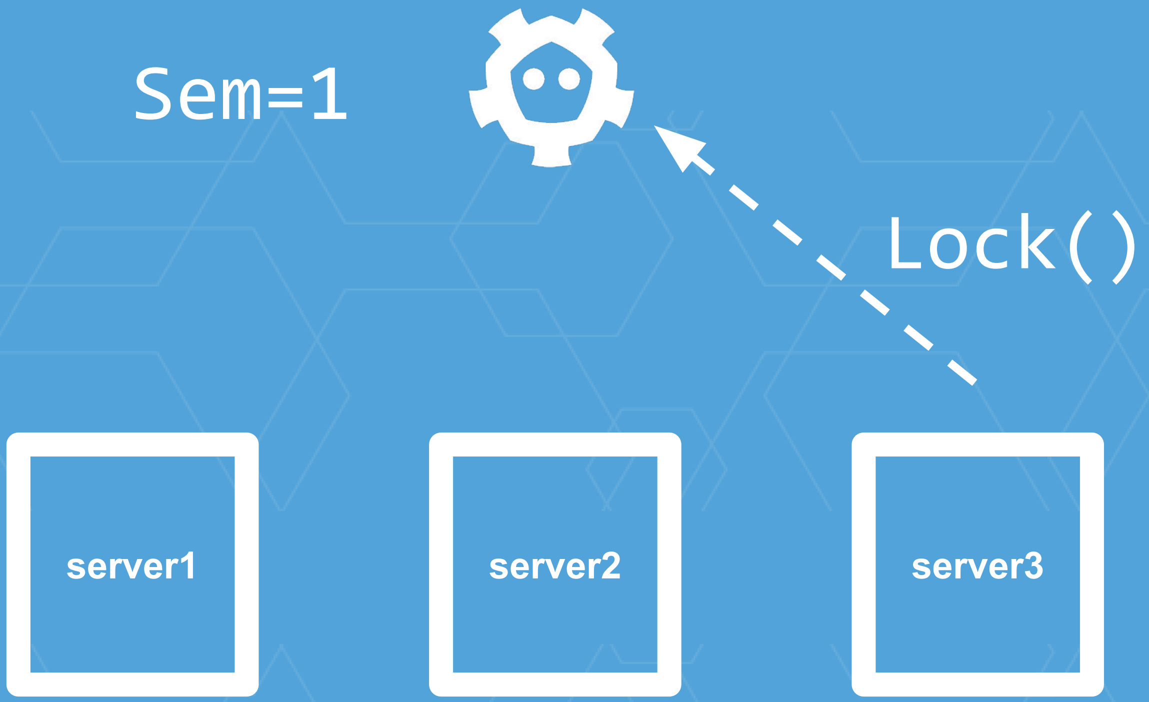
Cluster Wide Reboot Lock

- Need to reboot? Decrement the semaphore key (*atomically*) with etcd.
- `manager.Reboot()` and wait...
- After reboot, increment the semaphore key in etcd (*atomically*).

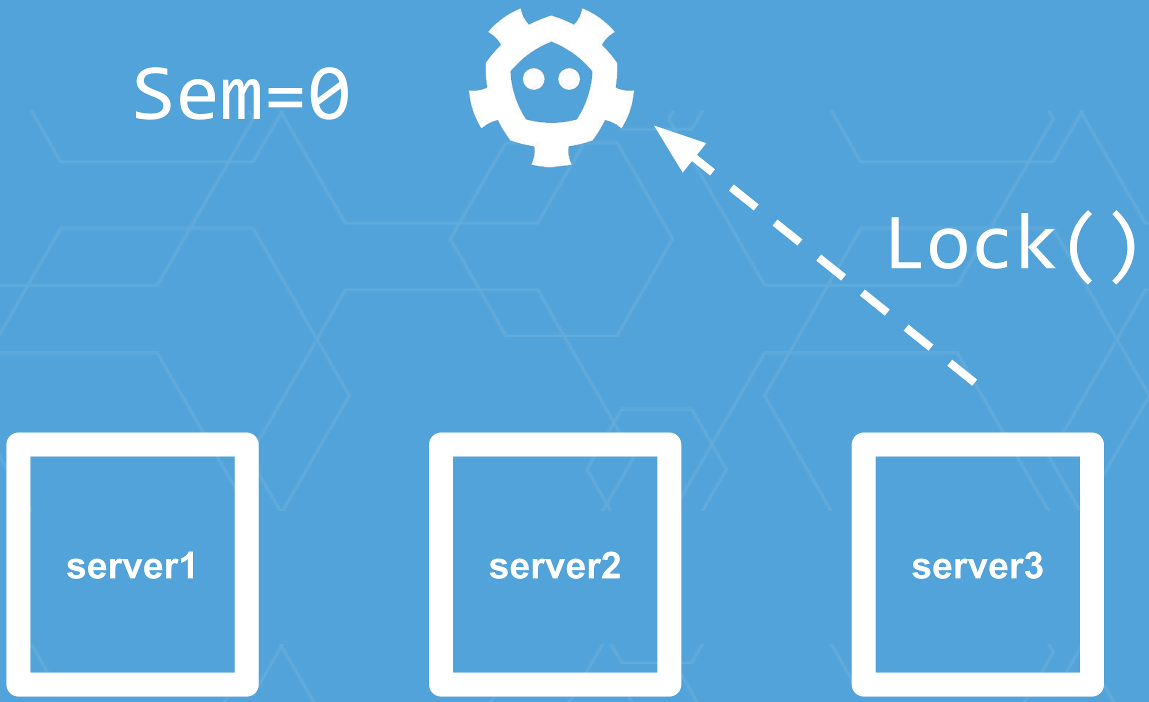
Cluster Wide Reboot Lock



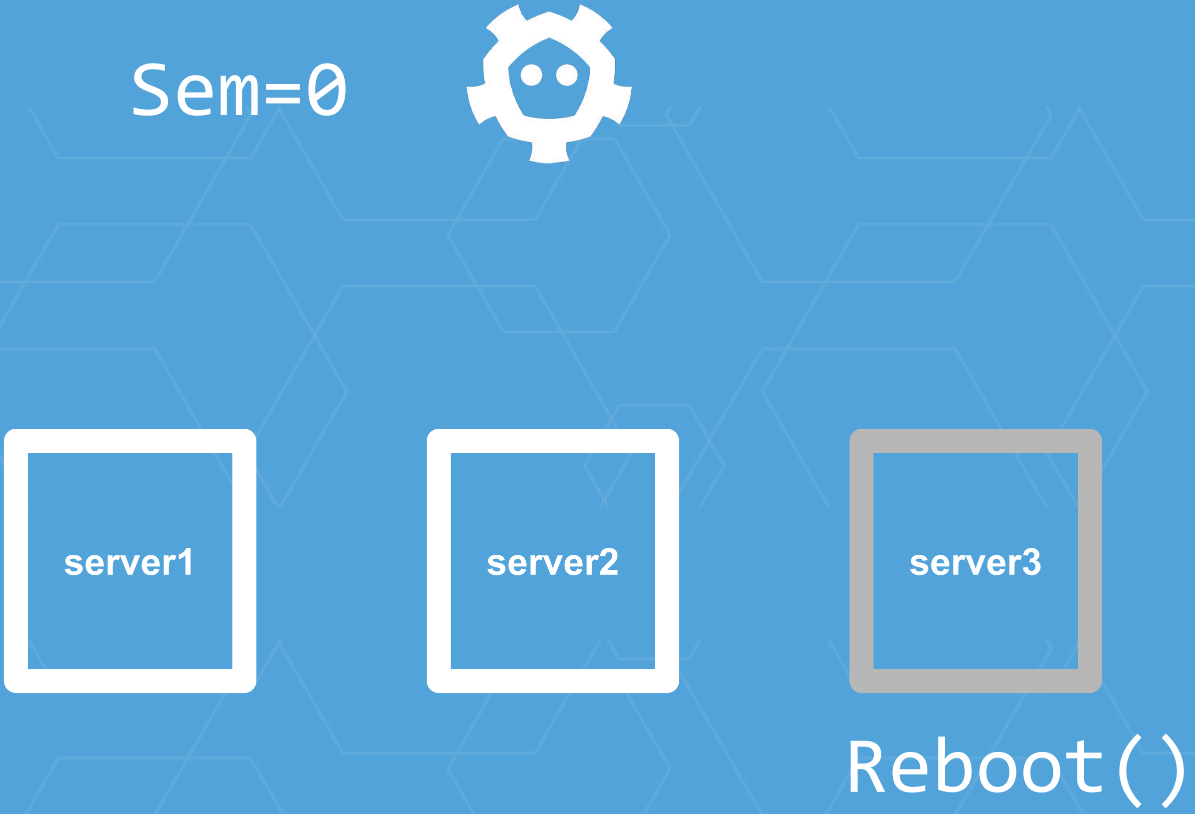
Cluster Wide Reboot Lock



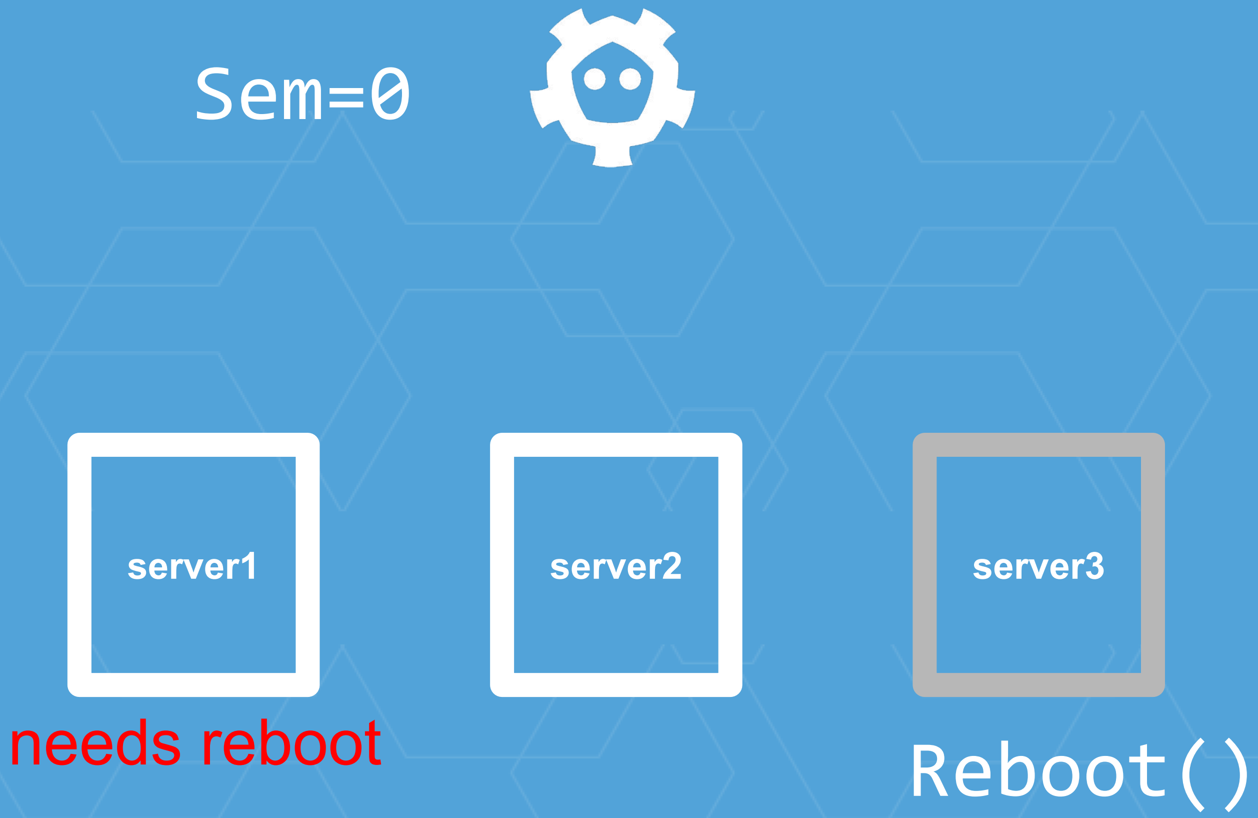
Cluster Wide Reboot Lock



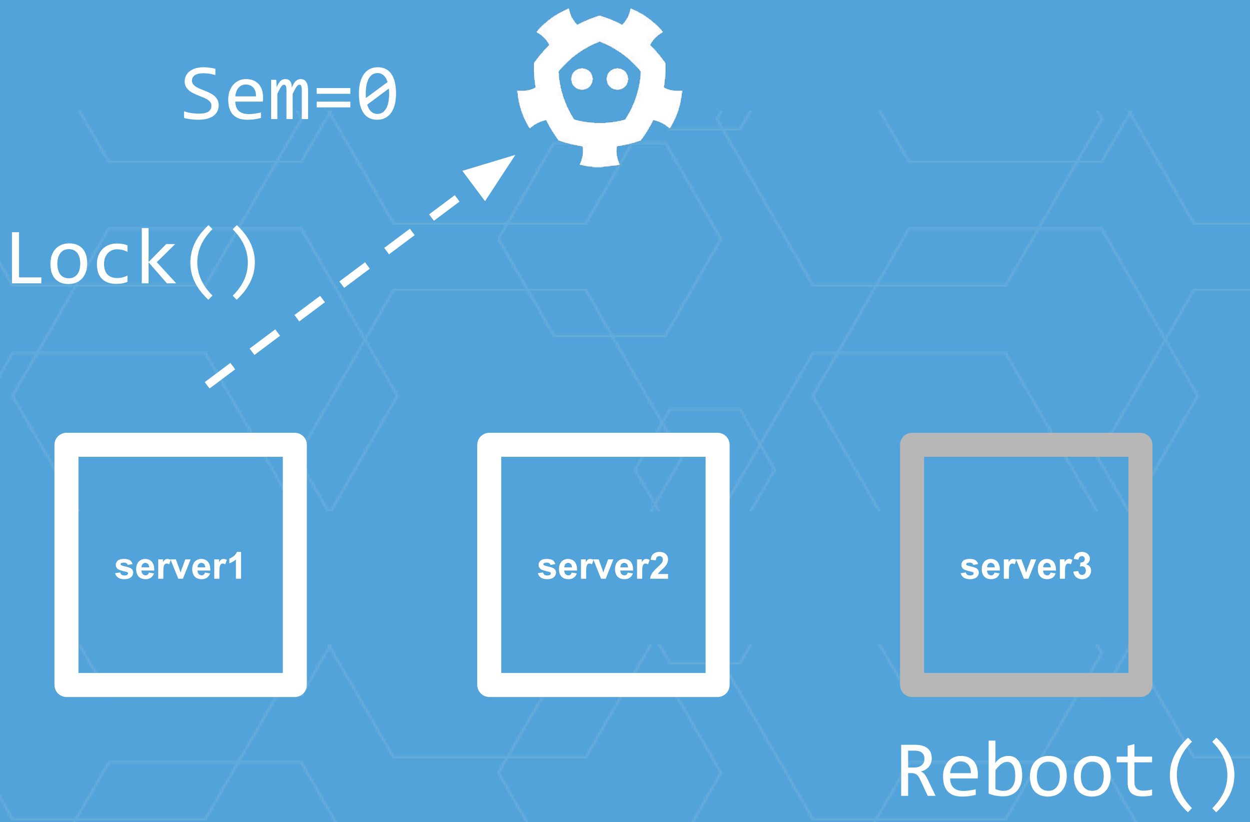
Cluster Wide Reboot Lock



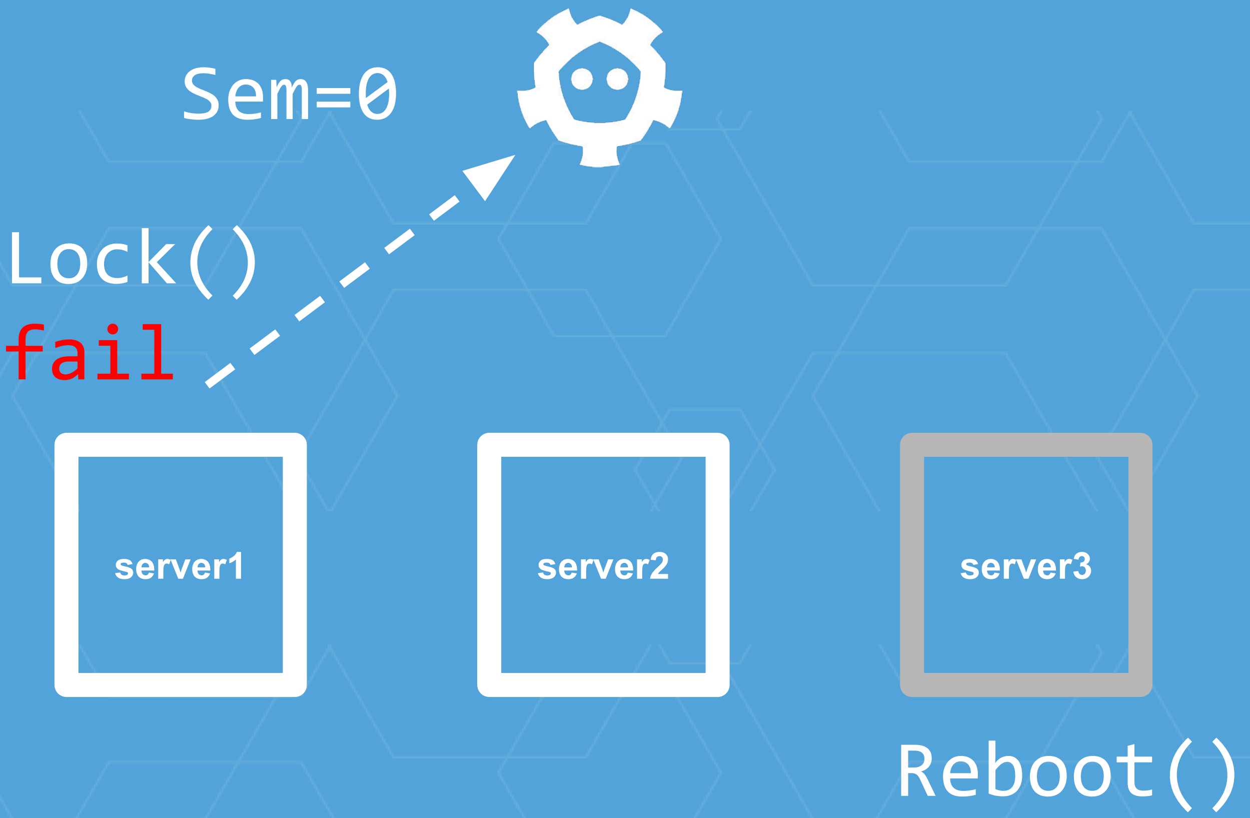
Cluster Wide Reboot Lock



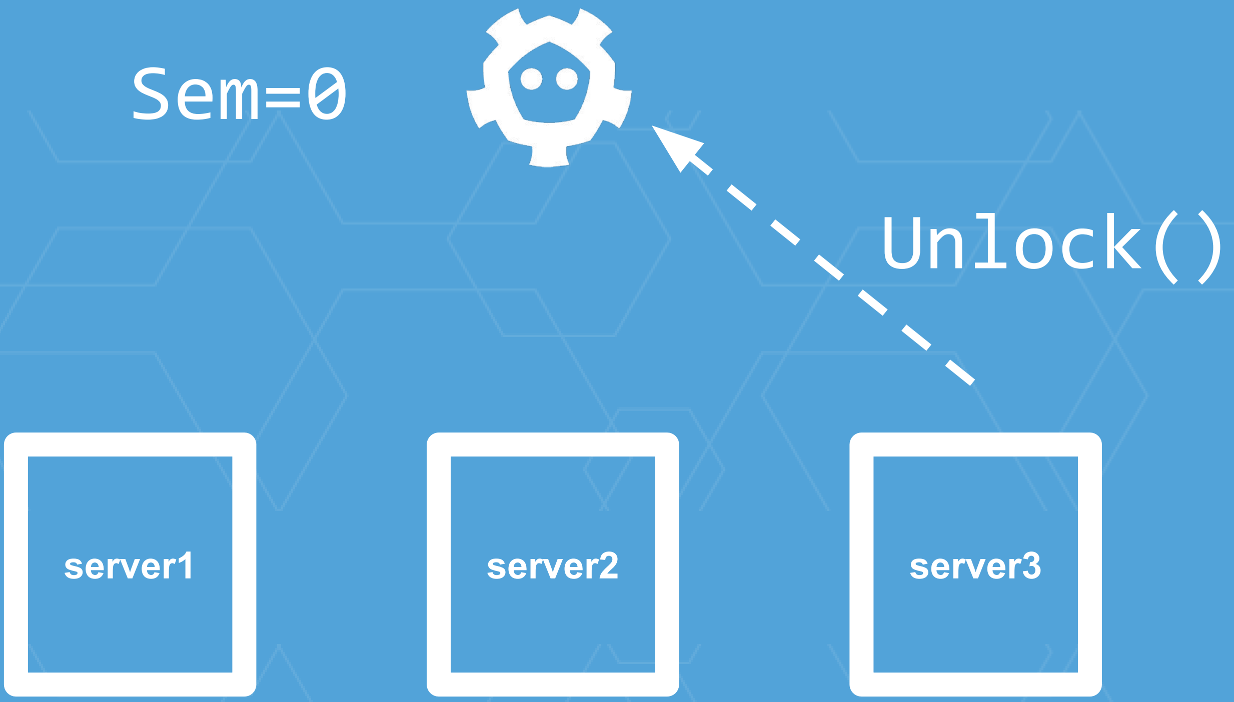
Cluster Wide Reboot Lock



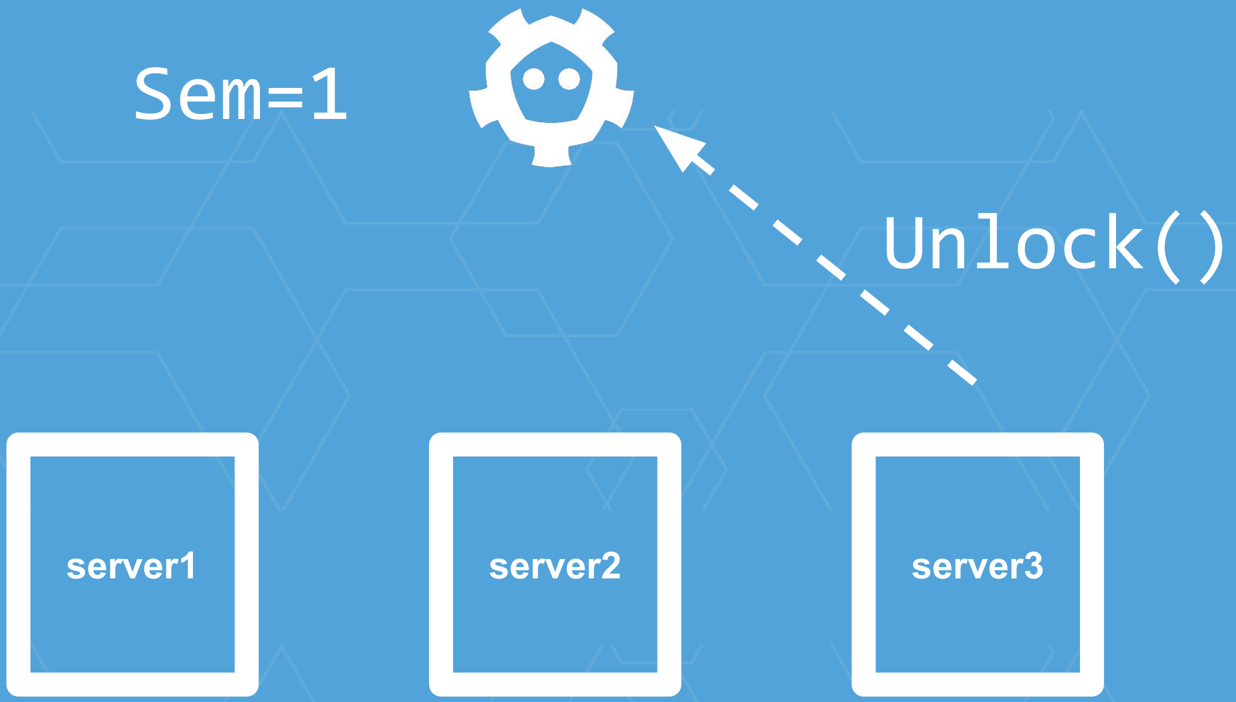
Cluster Wide Reboot Lock



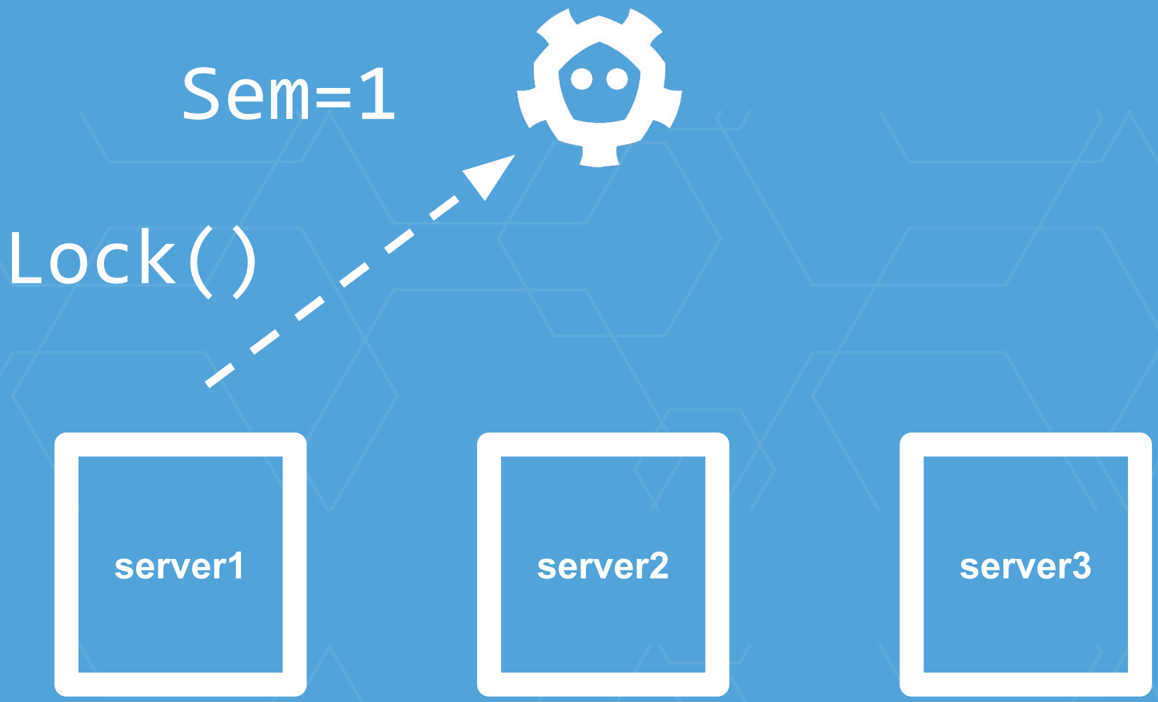
Cluster Wide Reboot Lock



Cluster Wide Reboot Lock



Cluster Wide Reboot Lock



etcd apps

skydns

skydns

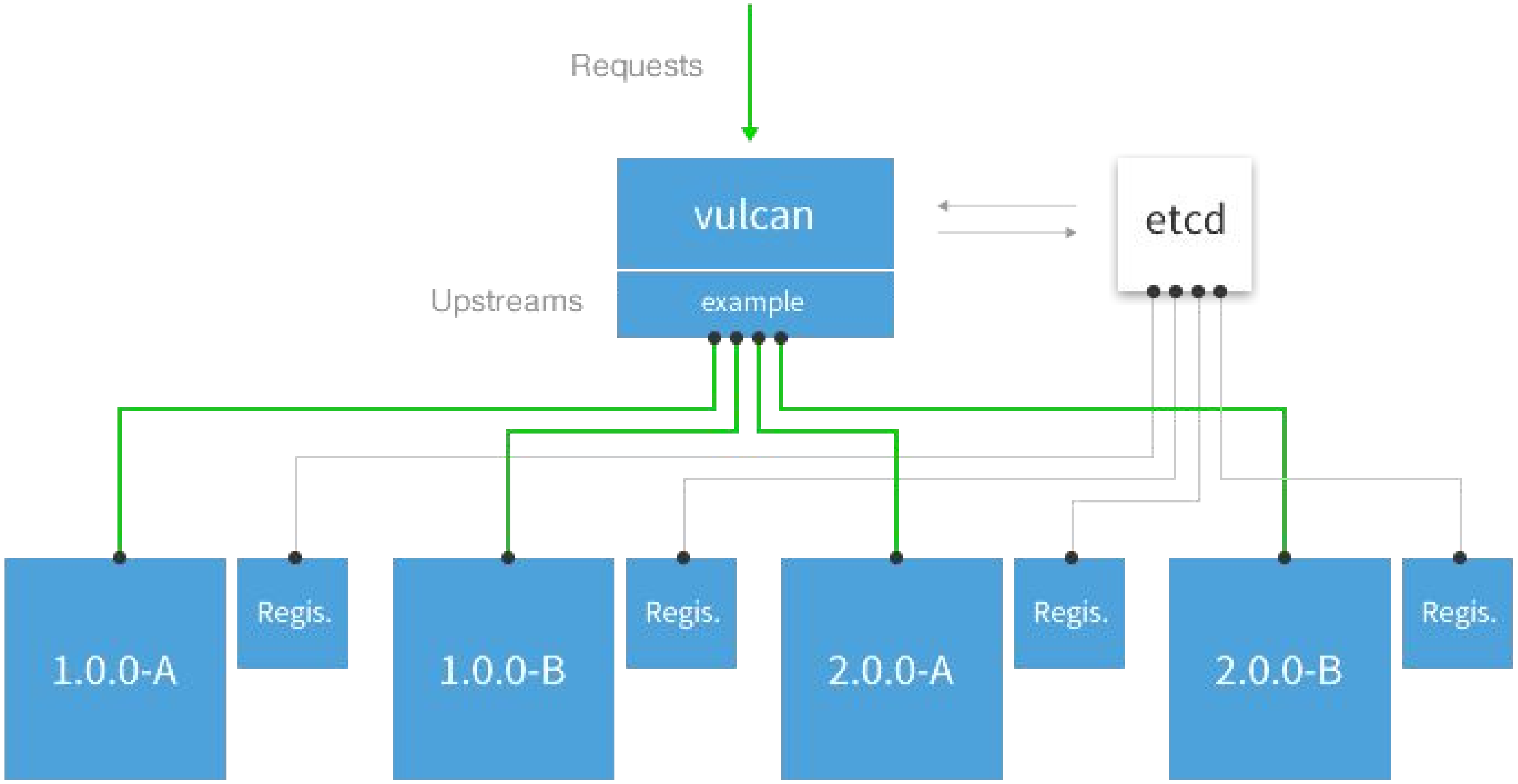
- Service discovery and DNS server
- backed by etcd for all configuration/records

etcd apps

vulcand

vulcand

- "programmatic, extendable proxy for microservices"
- HTTP load balancer
- etcd for all configuration


Requests
vulcan
Upstreams
example
etcd
1.0.0-A
Regis.
1.0.0-B
Regis.
2.0.0-A
Regis.
2.0.0-B
Regis.

etcd apps

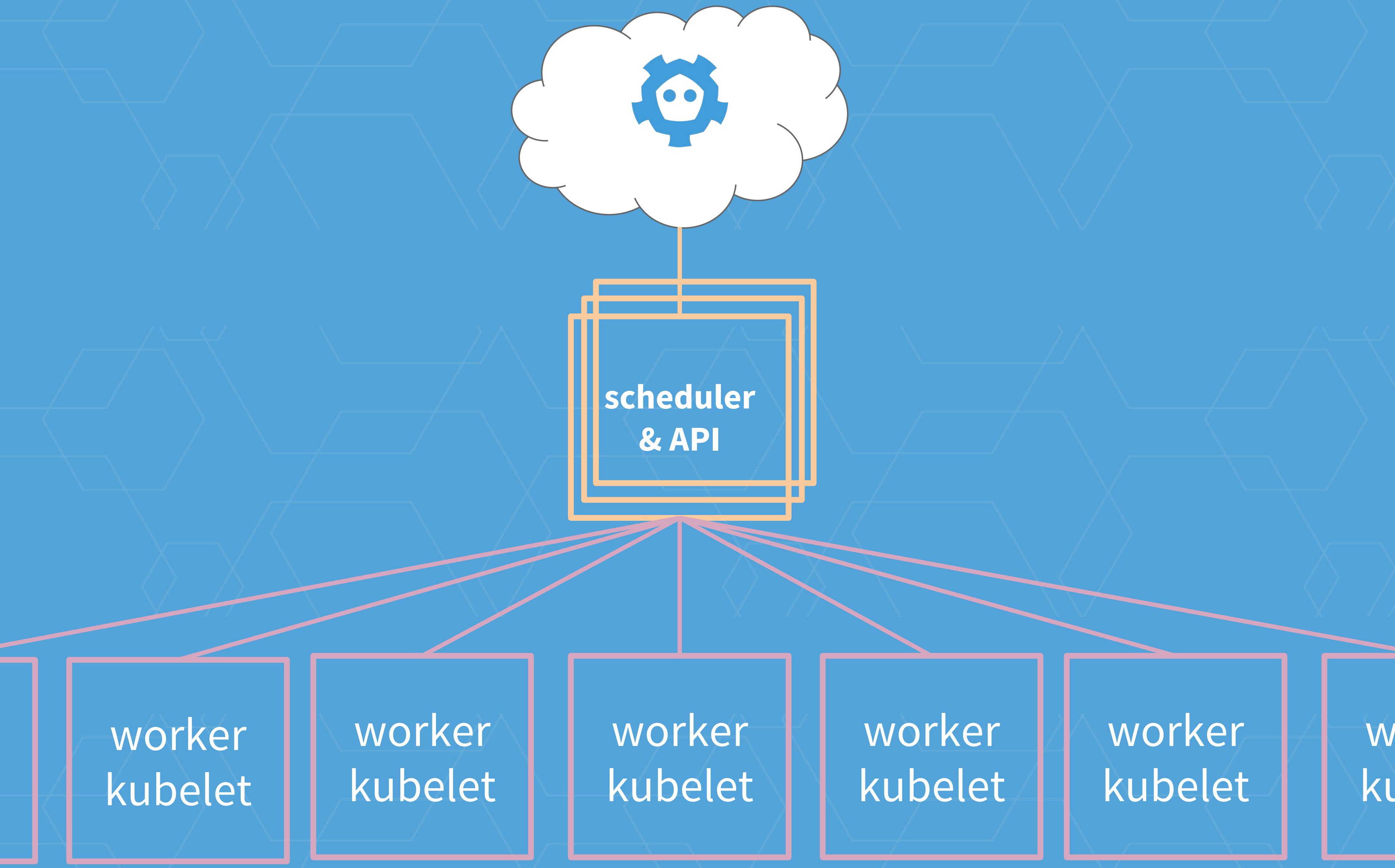
confd

confd

- simple configuration templating
- for "dumb" applications
- watch etcd for changes, render templates with new values, reload applications

etcd apps

Kubernetes


scheduler
& API
worker
kubelet
worker
kubelet
worker
kubelet
worker
kubelet
worker
kubelet

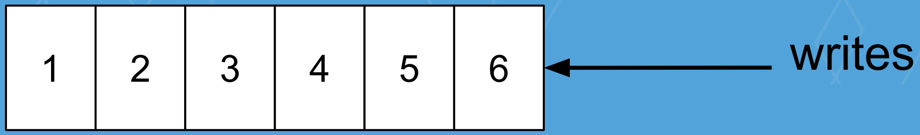
Scaling etcd

Scaling etcd to the next level

- Recent improvements
  - Asychronous snapshots
  - Request pipelining
- Future improvements
  - v3 and beyond

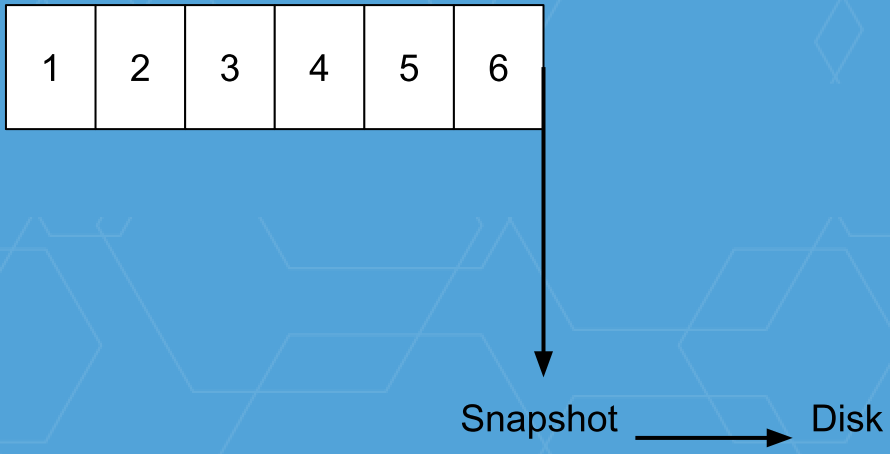
Recent improvements (v2)

- Asynchronous snapshotting
  - log-based system
  - snapshot before purging log



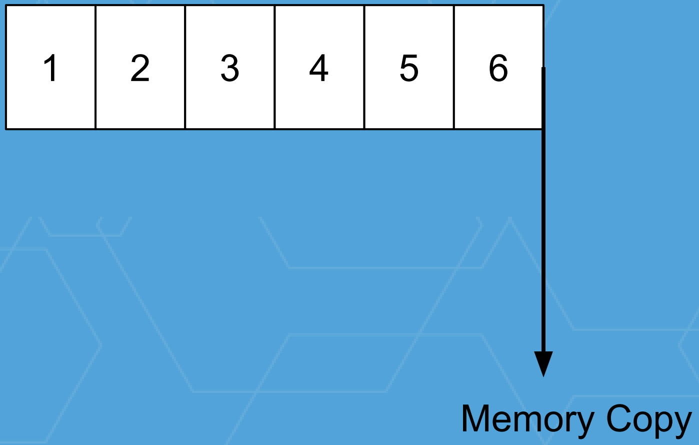
Recent improvements (v2)

- Asynchronous snapshotting
  - log-based system
  - snapshot before purging log



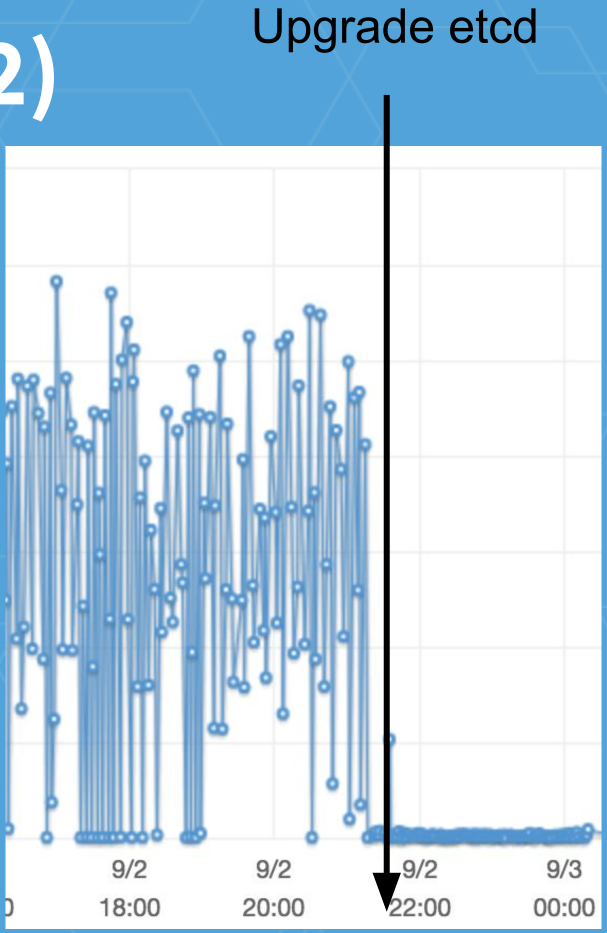
Recent improvements (v2)

- Asynchronous snapshotting
  - log-based system
  - snapshot before purging log



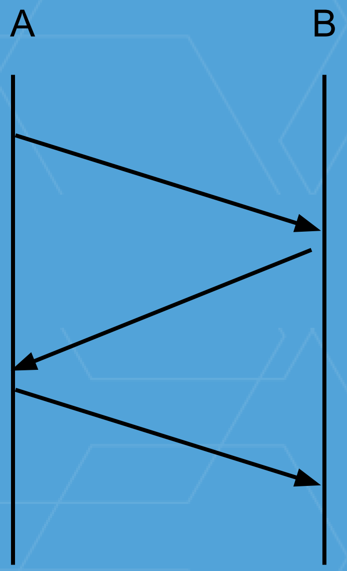
Recent improvements (v2)

- Asynchronous snapshotting
  - result from discovery service



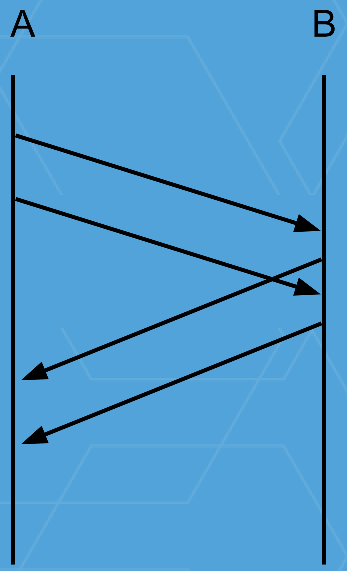
Recent improvements (v2)

- raft pipelining
  - etcd previously used *synchronous RPCs*
    - send next message only after getting previous reply



Recent improvements (v2)

- raft pipelining
    - etcd now uses *RPC pipelining*
        - send series of messages without waiting for replies



Future improvements (v3)

"Scaling etcd to thousands of nodes"

etcd v3.0

- Efficient and powerful API
- Disk-backed storage
- Incremental snapshots

The future (v3)

- Efficient and powerful API
  - flat binary key space
  - multi-object transaction
  - native leasing API
  - native locking API
  - gRPC (HTTP2 + protobuf)

Key space

- Flat binary key-value space
  - `coreos=awesome`
  - `coreos/etcd=kv`
  - `coreos/rkt=container`
- Keep it as simple as possible
  - want hierarchy?
    - build your own layer on top of kv

v3 API

- Put
  - `foo=bar`
- Get
- Range (consistent multi-get)
  - single key: foo
  - prefix: foo->fop (exclude)
  - range: foo->foo1
- Delete Range
  - same as range

KV API

```
KV.Put("foo", "bar")
KV.Get("foo")
KV.Range("foo", "foo10")
KV.Delete("foo")
KV.DeleteRange("foo", "foo10")
```

v3 API

- Mini transaction
  - two phases
    - *compare*
    - *execution* (either *success* or *failure*)
  - compare on value, index, etc.
  - execute a list of basic operations

v3 API

- Mini transaction
  - compare and swap
    - *compare*: foo=bar
    - *success*: foo=bar2
  - multiple object transaction
    - *compare*: cond1=true && cond2=true
    - *success*: pass=true
    - *failure*: pass=false

Mini Transaction

```
Tx.If(
  Compare(Value("foo"), ">", "bar"),
  Compare(Version("foo"), "=", 2),
  ...
).Then(
  OpPut("ok","true")...
).Else(
  OpPut("ok","false")...
).Commit()
```

v3 API

- Watch
  - support multiple keys and prefixes per stream
    - `watchKey(foo)`
    - `watchPrefix(coreos)`
  - support watch from historical point
    - `watchKey(foo, index_of_an_hour_ago)`
    - user-driven history compaction

gRPC

- Efficient
  - multiple streams share one TCP connection
  - compacted encoding format (protobuf)
- Rich generated libraries in tens of languages
  - Go, Java, Python, C++...

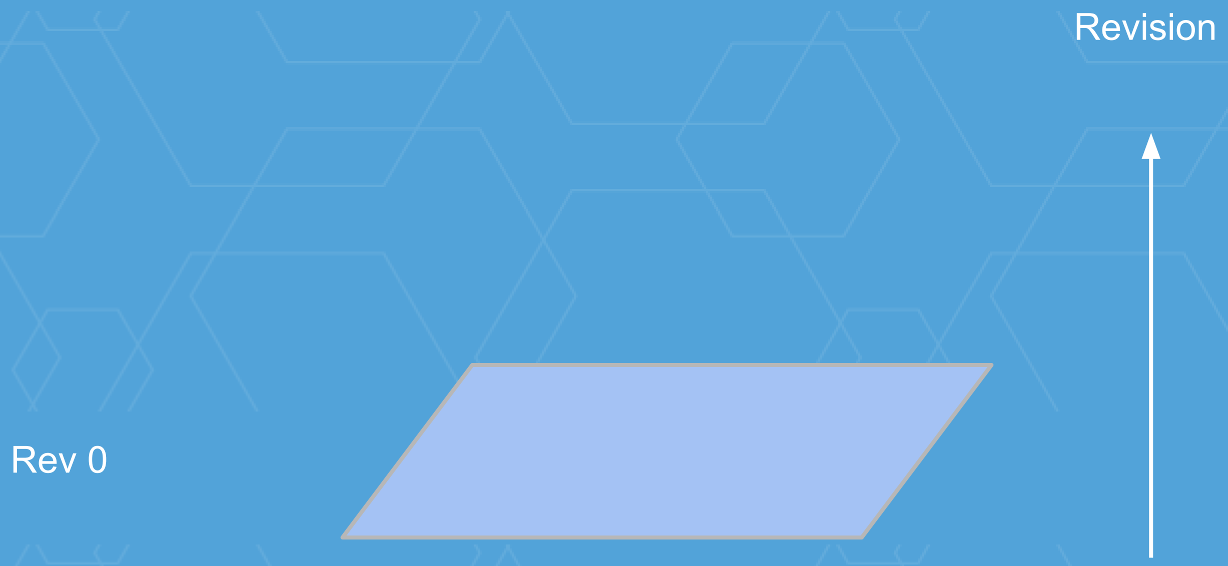
The future (v3)

- Incremental snapshot
  - only save the delta instead of full data set
  - less I/O and CPU cost per snapshot
  - no bursty resource usage, more stable performance

The future (v3)

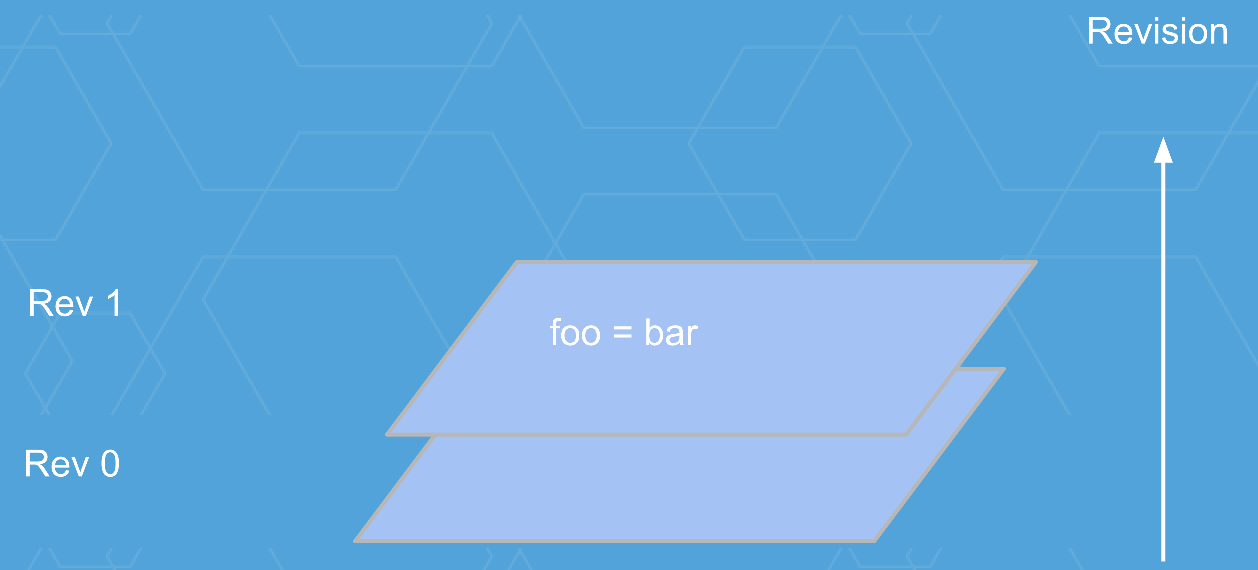
- Disk backend
    - keep the cold historical data on disk
    - keep the hot data in memory
    - support "entire history" watches
    - user-facing compaction API

MVCC



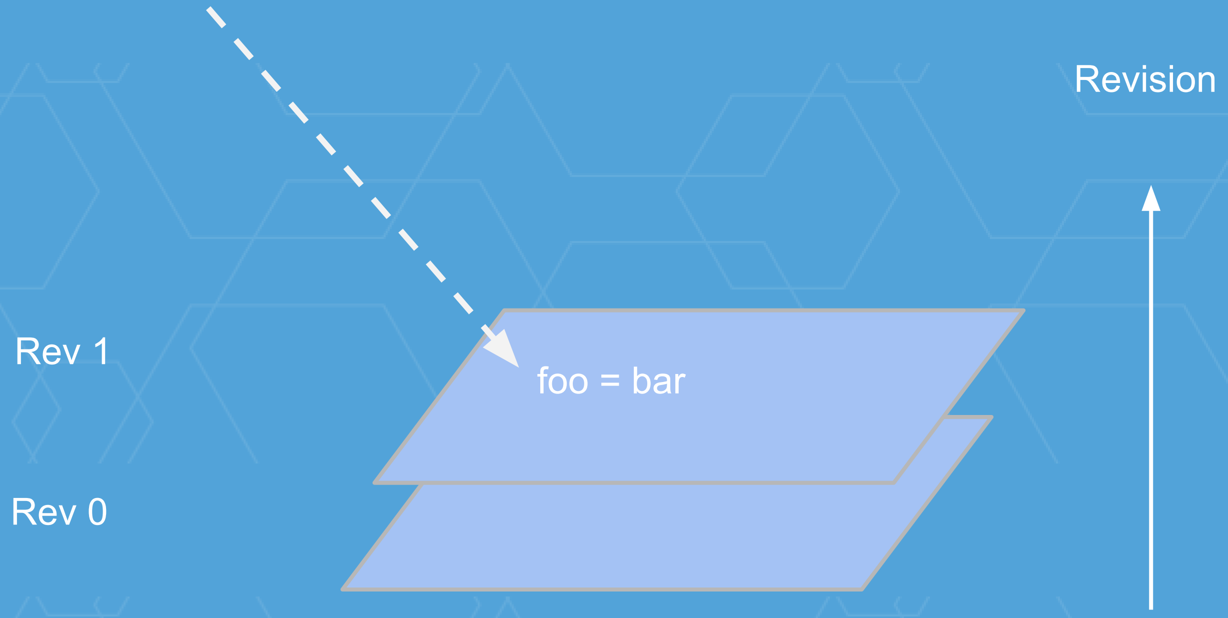
MVCC

```
KV.Put("foo", "bar") -> increase Rev
```



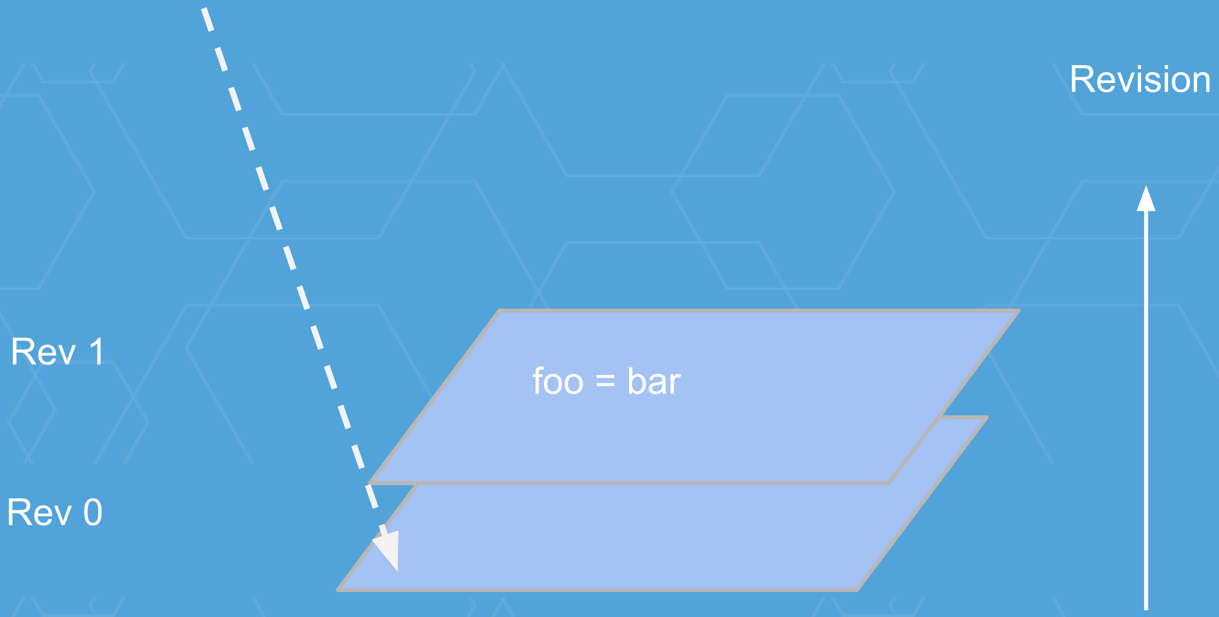
MVCC

```
KV.Get("foo") = "bar"
```



MVCC

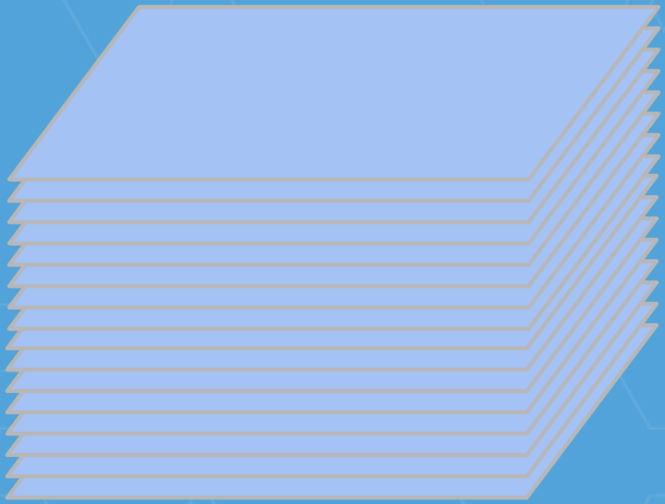
```
KV.Get("foo", WithRev(0)) = nil
```



Compaction

Why?

- Too many revisions



Compaction

Analyse the old revisions to be compacted

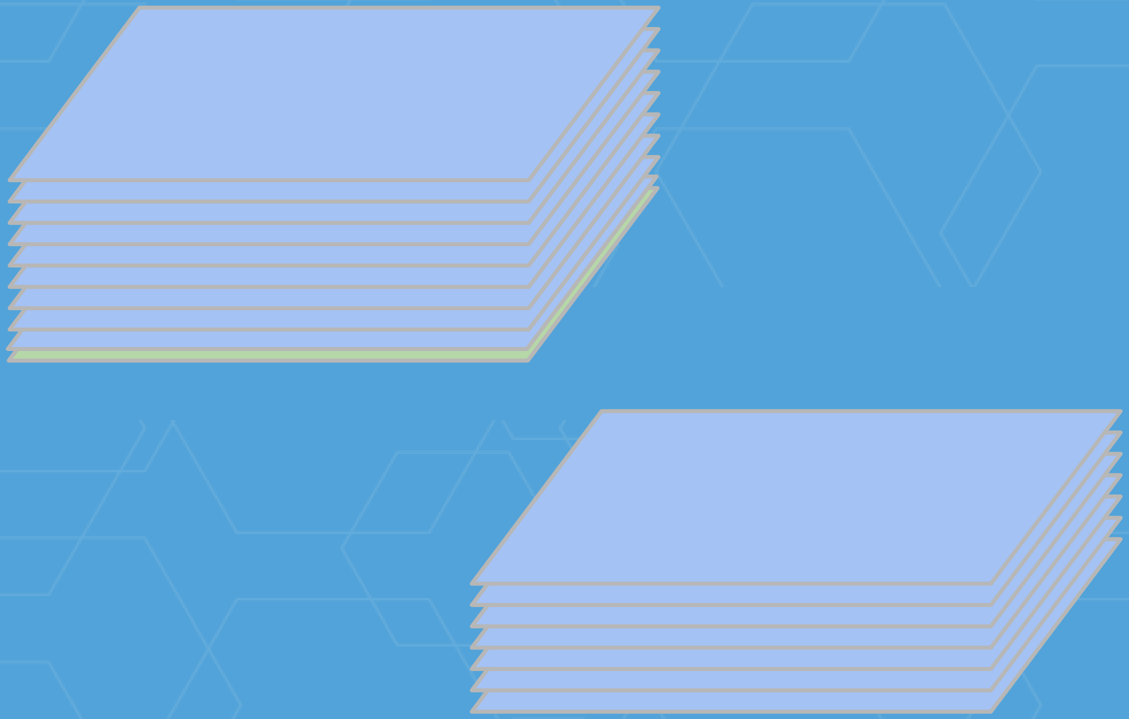
Compaction

Rule 1: the key with tombstone can be removed

Rule 2: keep the latest version of a non-tombstone key

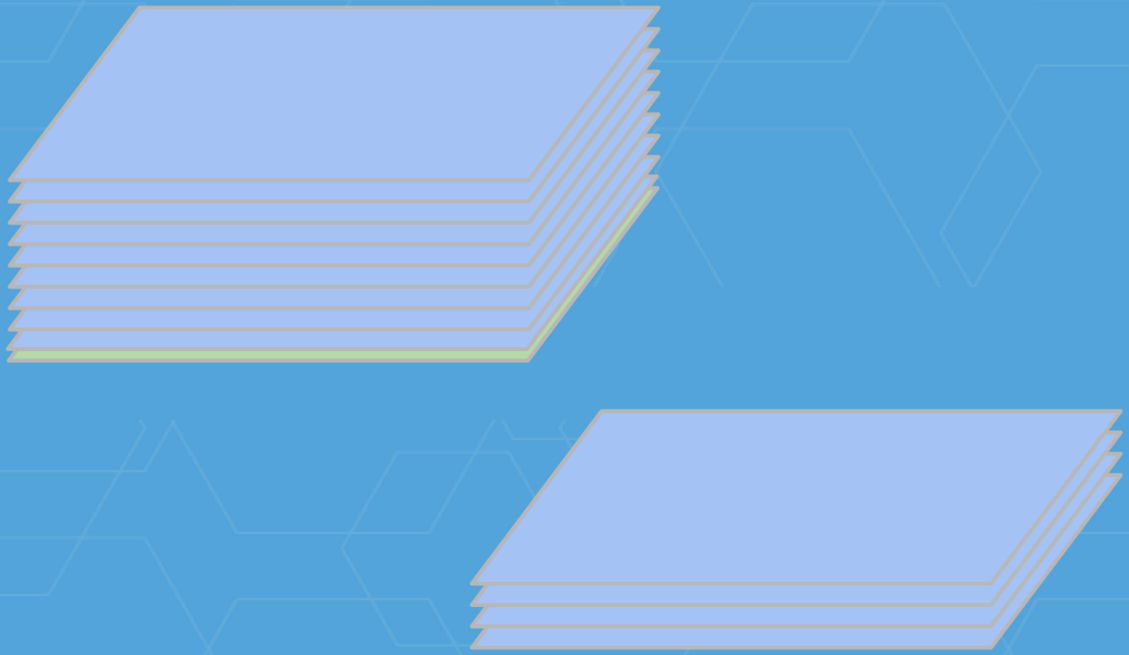
Compaction

Clean up old revisions in background



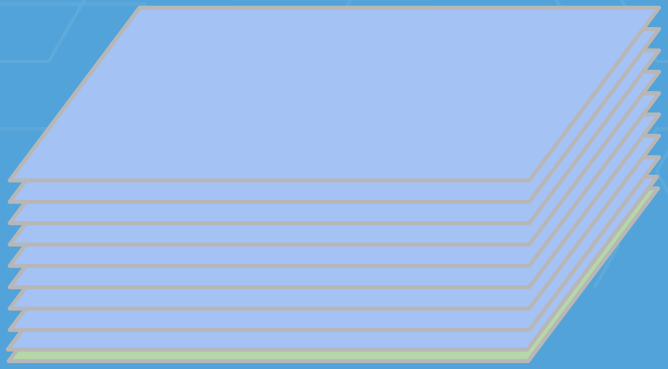
Compaction

Clean up old revisions in background



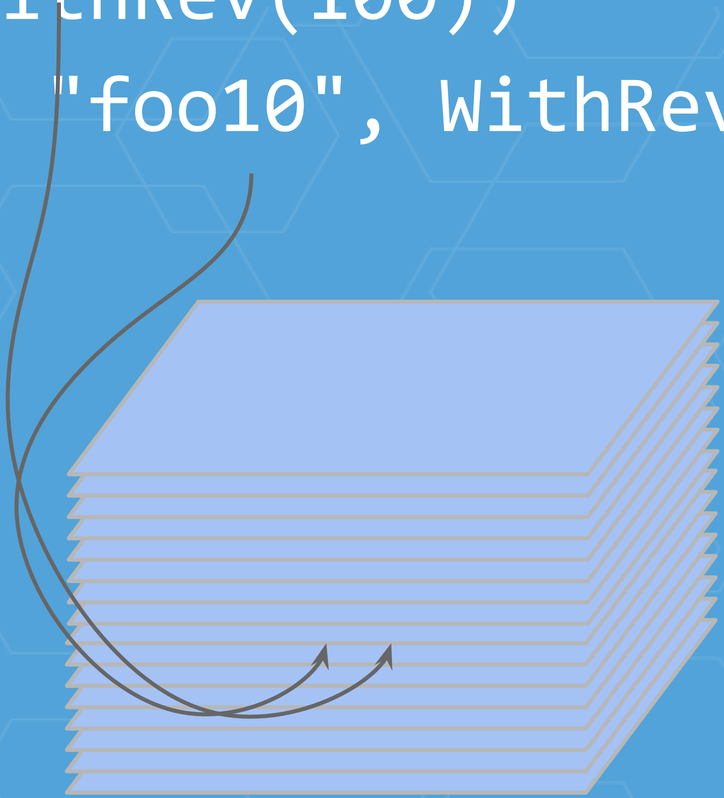
Compaction

Done!



KV Snapshot

```
// working on snapshot of KV at Rev 100
KV.Get("foo", WithRev(100))
KV.Range("foo", "foo10", WithRev(100))
```



Lease

```
l := lease.Create(10*second)

kv.Put("foo", "bar", l.ID)

// key will be removed without keeping
// alive the lease
go KeepAlive(l.id)
```

Watch

```
events, err := watcher.Watch("foo")
if err != nil {
  // handle error
}
for r := <- responses {
  // consume received events
}
```

Recipes

Leader election

- `election.Elect("eFoo"), election.Resign("eFoo")`

Locking

- `locking.Acquire("lFoo"), locking.Release("lFoo")`

Barrier

- `barrier.Enter("bFoo"), barrier.Leave("bFoo")`

etcd and go

etcd and go: the good

- Extremely fast development speed
- Generally robust standard libraries
- Healthy, active ecosystem
- Simple but powerful concurrency

... i.e., all the usual reasons people like Go

etcd and go: the less good

- HTTP pipelining + CloseNotify = no fun
  - http://www.projectclearwater.org/adventures-in-debugging-etcd-http-pipelining-and-file-descriptor-leaks/
  - https://github.com/golang/go/issues/13165
- CloseNotify = no fun
  - https://github.com/golang/go/issues/9524
- Unpredictable GC = latency spikes
  - https://github.com/coreos/etcd/issues/4111
- Scheduler starvation = dead Raft goroutine
  - citation needed

Thanks!

Join us!
github.com/coreos/etcd

We are hiring!
coreos.com/careers