etcd

Xiang Li

CoreOS

CoreOS

Motivation

CoreOS Updates Coordination

Update OS from version 1 to version 2

Motivation

CoreOS Updates Coordination

Update OS from version 1 to version 2

Unavailability

Cluster Wide Reboot Lock

Need to reboot?

- Decrement the semaphore key atomically
- Reboot and wait...
- After reboot increment the semaphore key.

Motivation

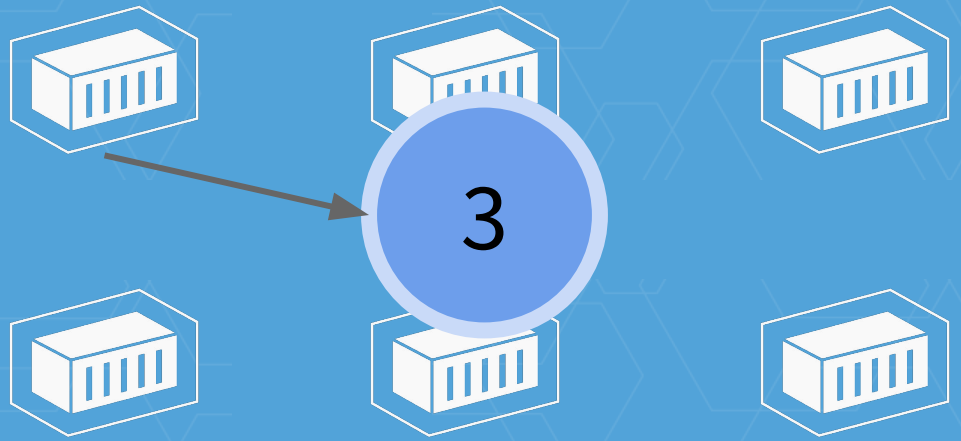
CoreOS Updates Coordination

Update OS from version 1 to version 2

Motivation

CoreOS Updates Coordination

Update OS from version 1 to version 2



Motivation

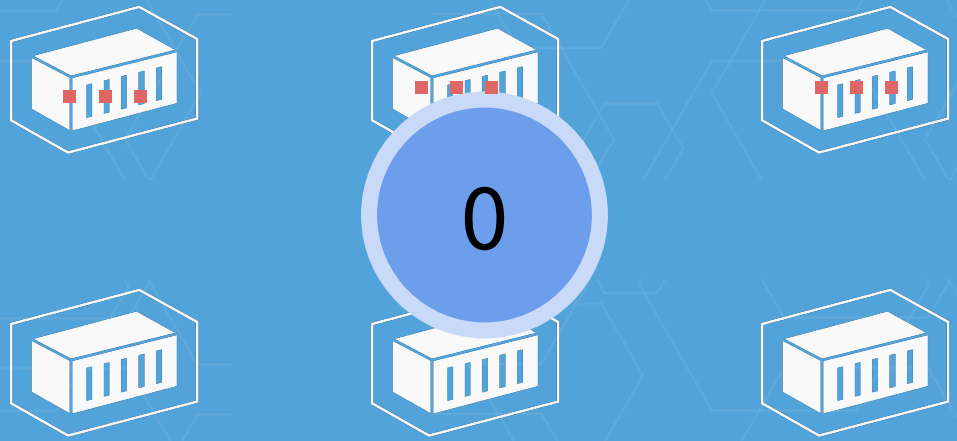
CoreOS Updates Coordination

Update OS from version 1 to version 2

Motivation

CoreOS Updates Coordination

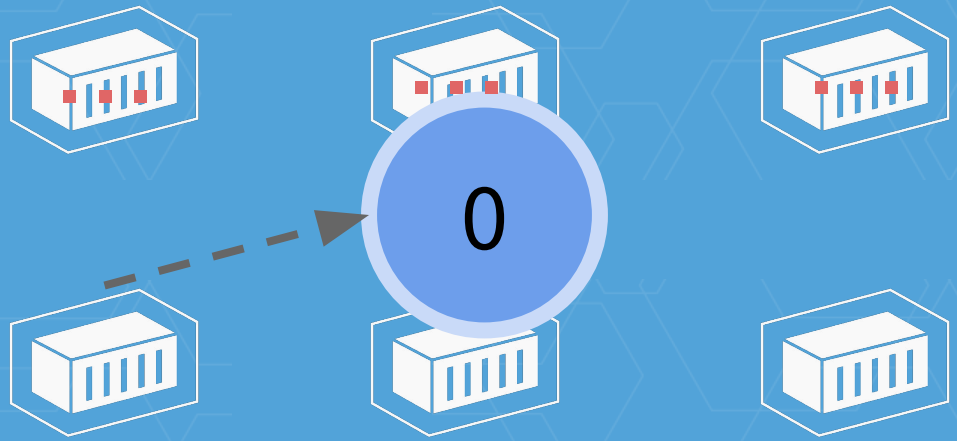
Update OS from version 1 to version 2



Motivation

CoreOS Updates Coordination

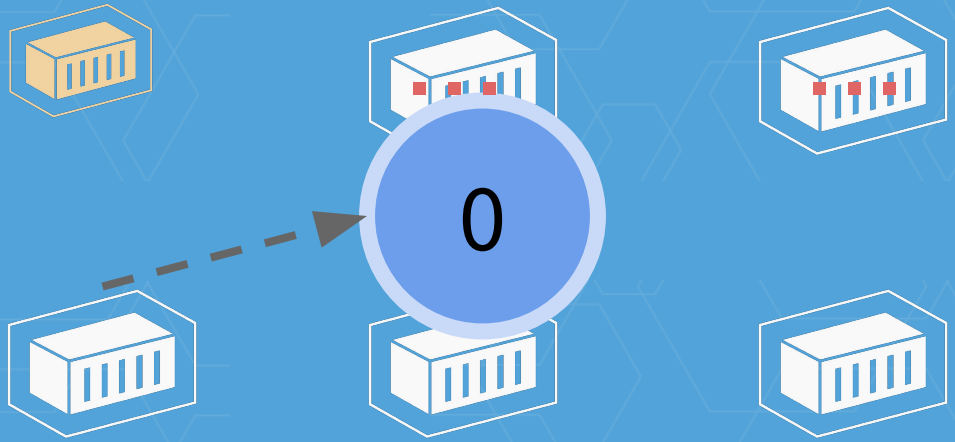
Update OS from version 1 to version 2



Motivation

CoreOS Updates Coordination

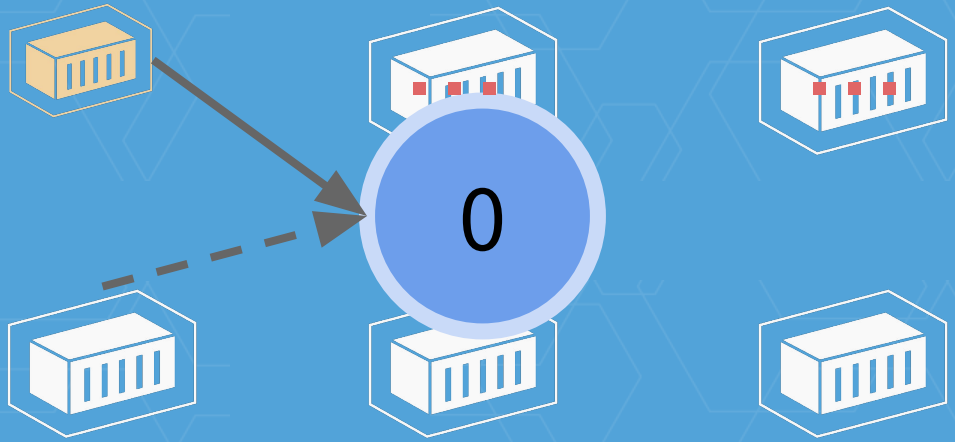
Update OS from version 1 to version 2



Motivation

CoreOS Updates Coordination

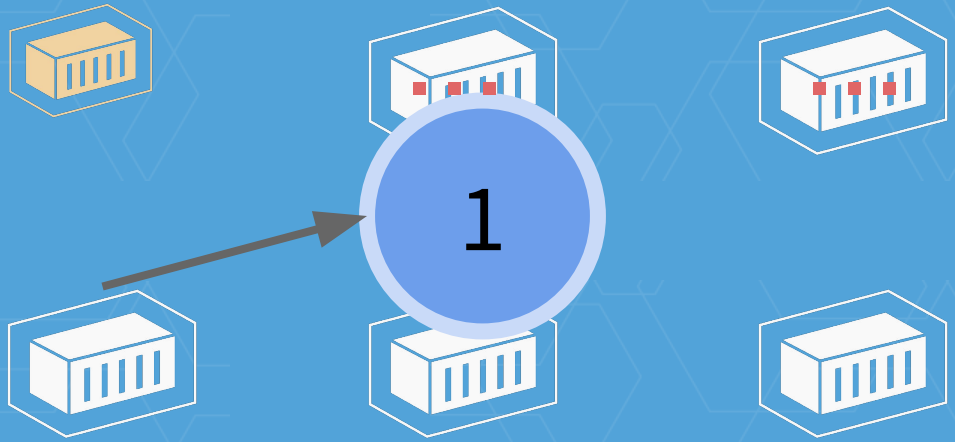
Update OS from version 1 to version 2



Motivation

CoreOS Updates Coordination

Update OS from version 1 to version 2



Motivation

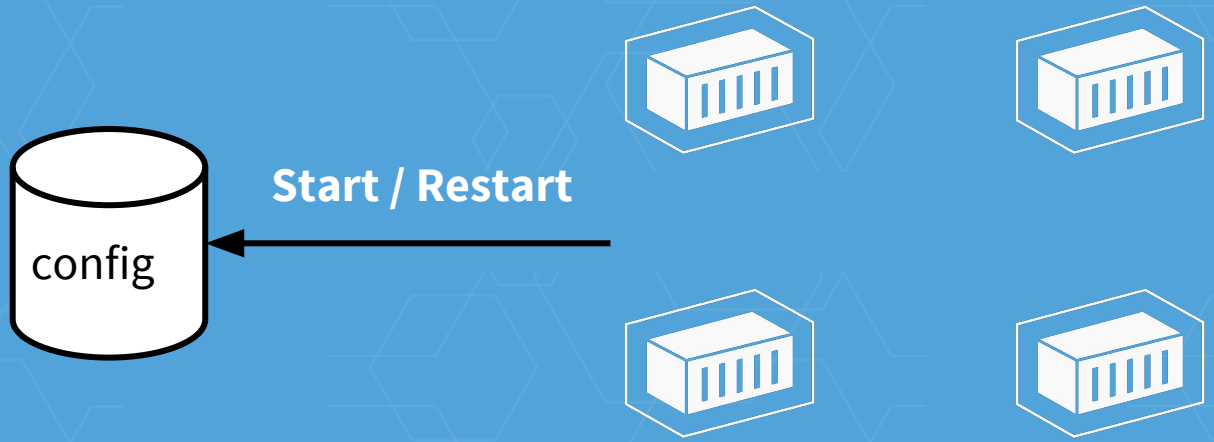
CoreOS Updates Coordination

Update OS from version 1 to version 2

Motivation

12 factor Apps

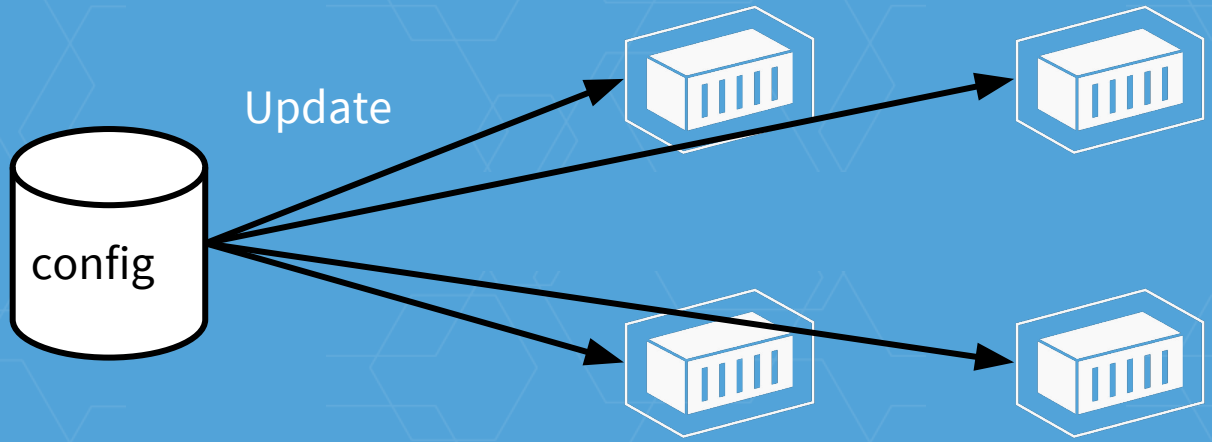
- Store configuration in environment



Motivation

12 factor Apps

- Store configuration in environment



Motivation

12 factor Apps

- Store configuration in environment

Requirements

- Strong Consistency
  - mutual exclusive at any time for locking purpose
- Highly Available
  - resilient to single point of failure
  - resilient to network partition
- Watchable
  - push configuration updates to application

Hard

- High Availability == Distributed System
- Distributed system is hard
  - Network fails
  - Machine fails

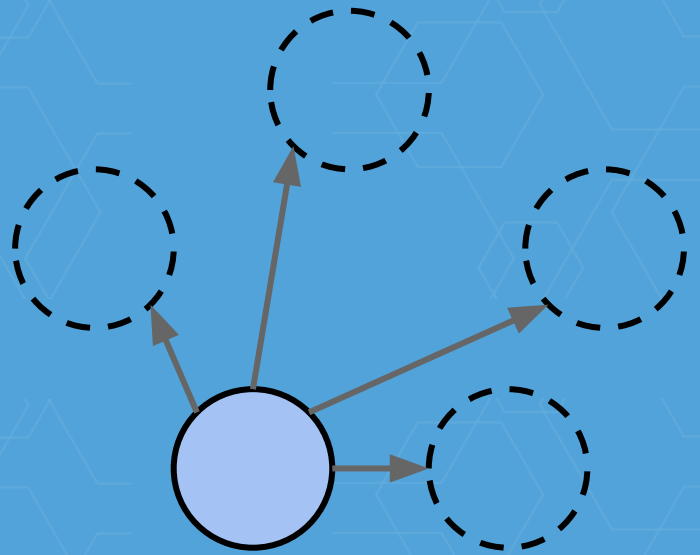
Distributed System

- Network issues
  - message lost
  - message reorder
  - concurrent messages

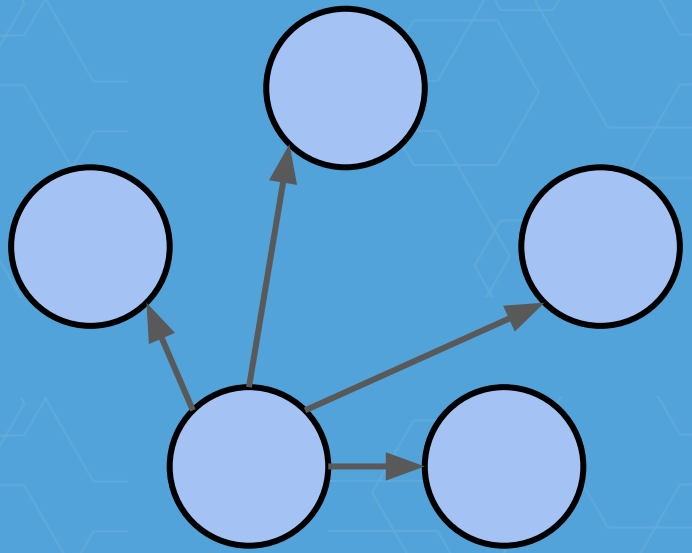
Distributed System

- Machine issues
  - power fais
  - OS fails

Make Consistent Decision

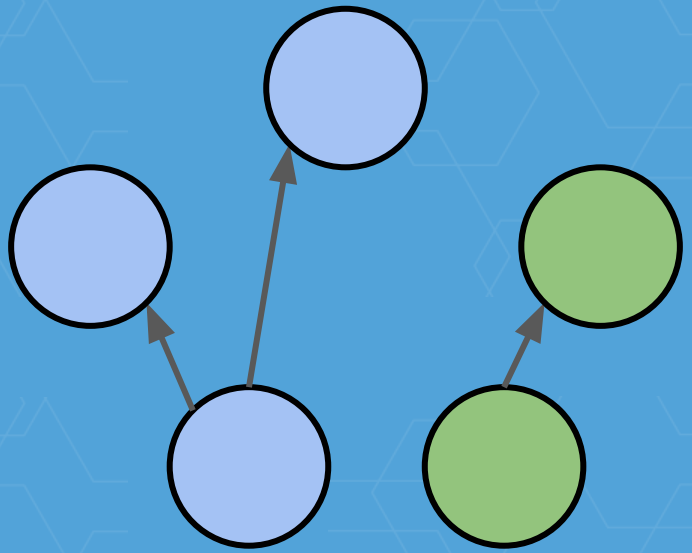


Make Consistent Decision



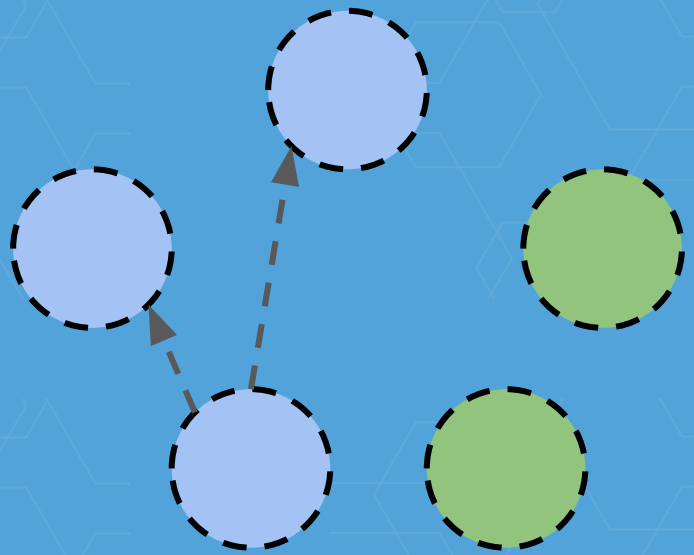
Making Consistent Decision

- Concurrent messages



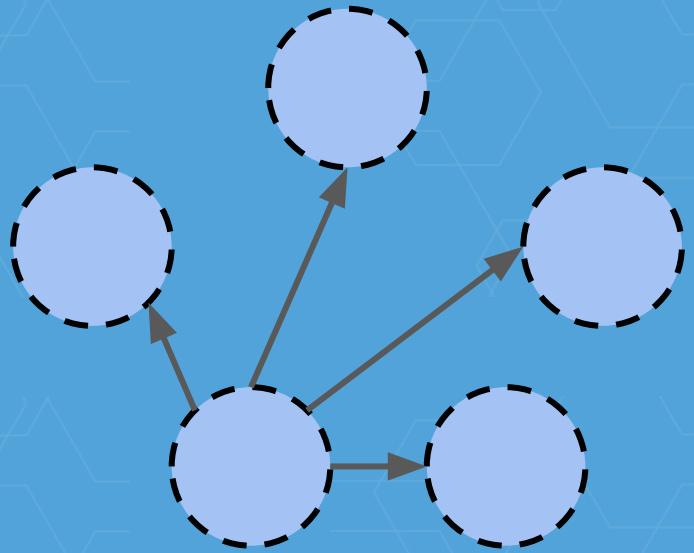
Making Consistent Decision

- Two phase commit



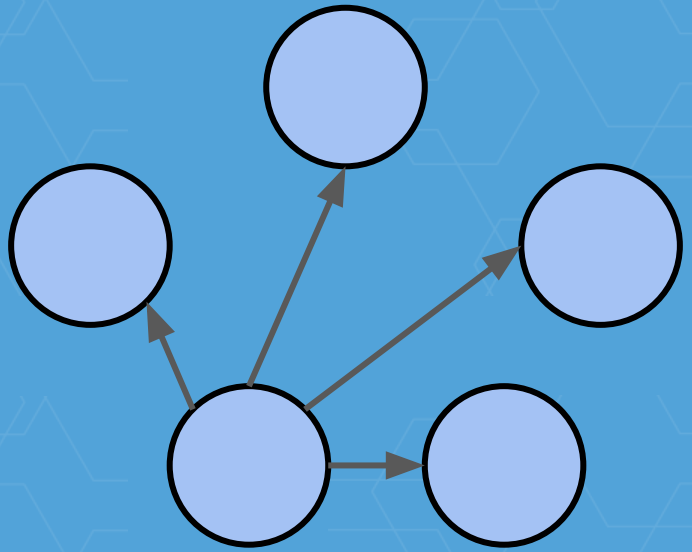
Making Consistent Decision

- Two phase commit



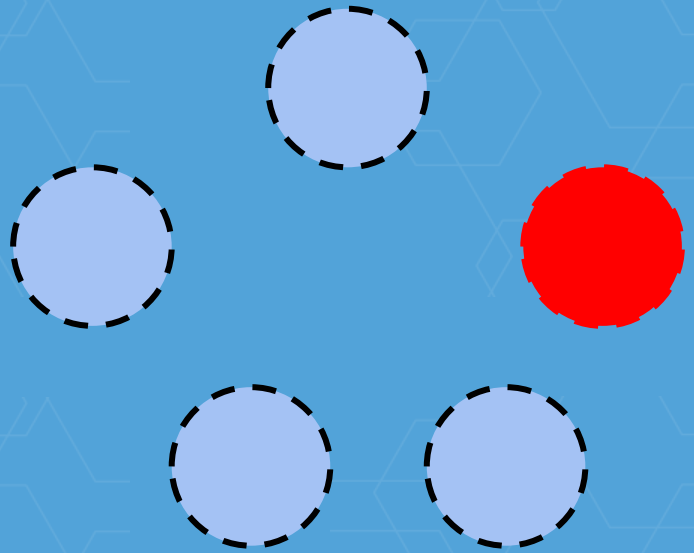
Making Consistent Decision

- Two phase commit



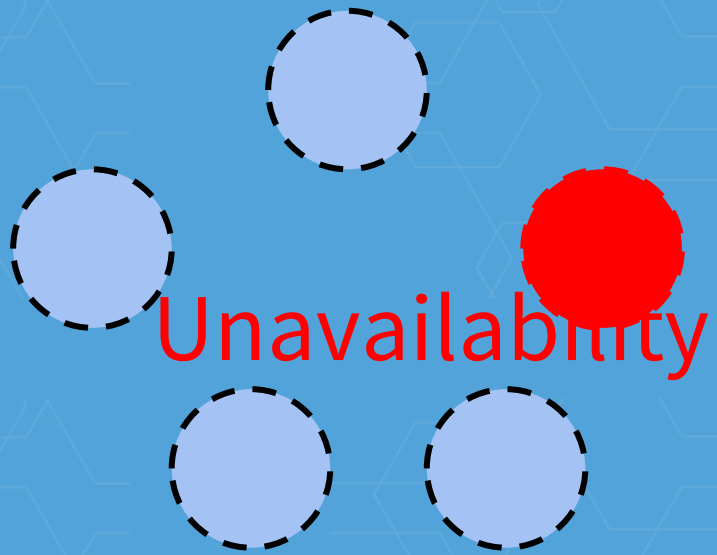
Making Consistent Decision

- Machine failures



Making Consistent Decision

- Machine failures



CAP Theorem

- Choose Two of them
  - Consistency
  - Partition tolerance
  - Availability
    - tricky!

CAP Theorem

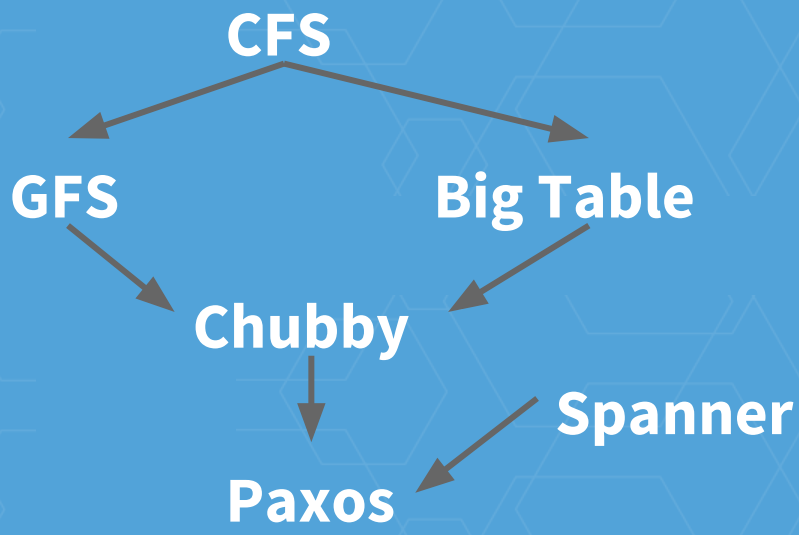
- Choose Two of them
  - **Consistency**
  - **Partition tolerance**
  - Availability

CP System

- Distributed Consensus Algorithm
  - Considered as one of the hardest problem in computer science
  - Paxos
    - Making a single consistent decision
  - Raft/ZAB/multi-paxos are all multi-round paxos
    - Making a sequence of consistent decisions

Common Problem

- Google
  - “All” infrastructure relies on Paxos



Common Problem

- Amazon
  - Replicated log
    - all of the amazon aws services relies Paxos
- Microsoft
  - Boxwood
    - foundation for Storage Infrastructure
- Hadoop
  - ZooKeeper

Common Problem

- Container??

Why solve it again?

- ZooKeeper
  - no runtime reconfiguration
  - high operational cost
  - unacceptable major issue rate
    - 30% issues are still open
    - 70% of them are above major
  - not a fan of JVM

etcd

Stats

- ~ 10,000 stars
- ~ 300 contributors

etcd

- the heart of container based cloud
  - Kubernetes
  - Docker Swarm
  - Cloud Foundry Diego

Introduction

- A key value store
  - Replicated
  - Highly available
  - Consistent

Introduction

- play.etcd.io

Key-Value API

Put(key, value)

Get(key), Range(from, to)

Delete(key), DeleteRange(from, to)

Watch API

```
w = Watch(key or range)
for {
    r = w.Recv()
    print(r.Event) // PUT
    print(r.KV) // foo,bar
}
```

Lease API

```
l = CreateLease(15 * second)
Put(foo, bar, l)

l.KeepAlive()

l.Revoke()
```

Mini transaction

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

Multi-Version

Put(foo, bar)

Put(foo, bar1)

Put(foo, bar 2)

Get(foo)

Multi-Version

Put(foo, bar)

Put(foo, bar1)

Put(foo, bar2)

Get(foo) -> bar2

Get(foo, 1)

Multi-Version

Put(foo, bar)

Put(foo, bar1)

Put(foo, bar2)

Get(foo) -> bar2

Get(foo, 1) -> bar

Demo

Recipes

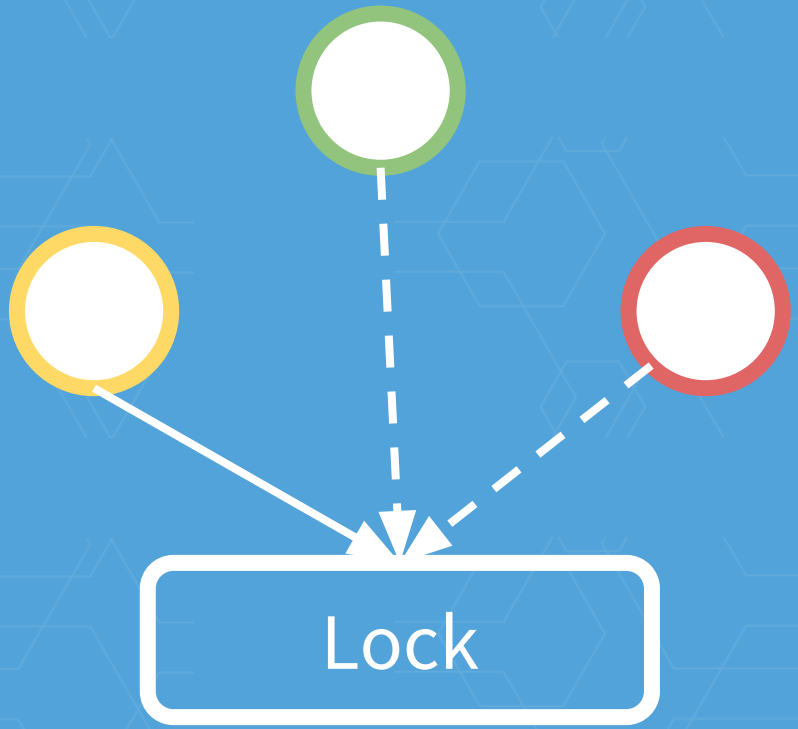
Lock

Leader Election

Barrier

Double Barrier

Spin Lock



Spin Lock

Lock

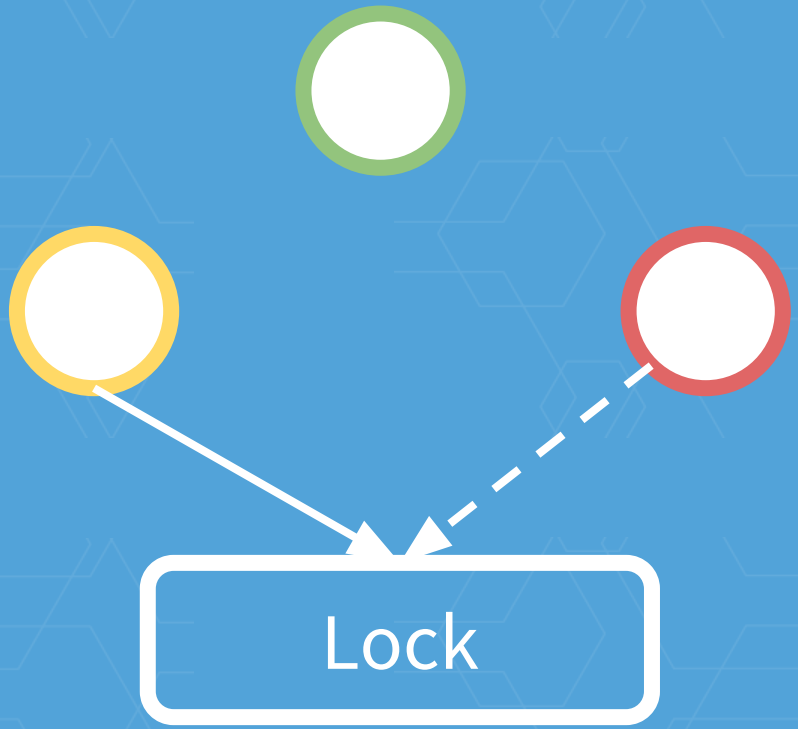
```
for {
    r = Txn.If(value(lock)==""").Then(lock="id").Commit()
    if r.Success {
        break
    }
    sleep(some_interval)
}
```

Spin Lock

Unlock

```
put(lock,"")
```

Mutex



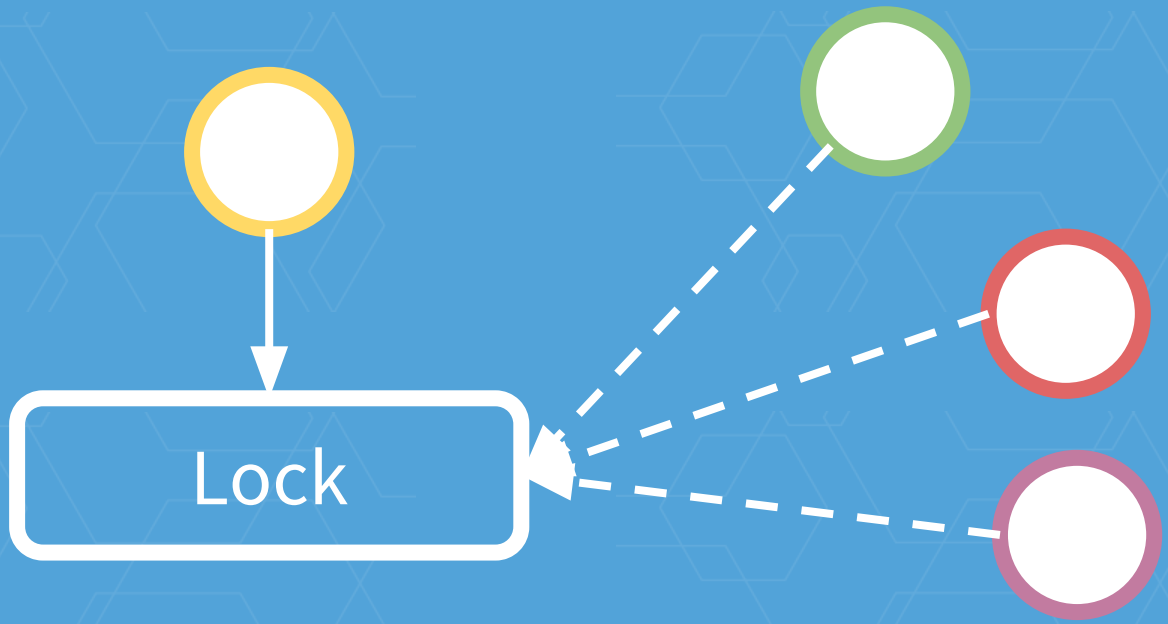
Mutex

Lock

```
for {
    r = Txn.If(value(lock)=="").Then(lock="id").Commit()
    if r.Success {
        break
    }
    wait(lock.delete)
}
```

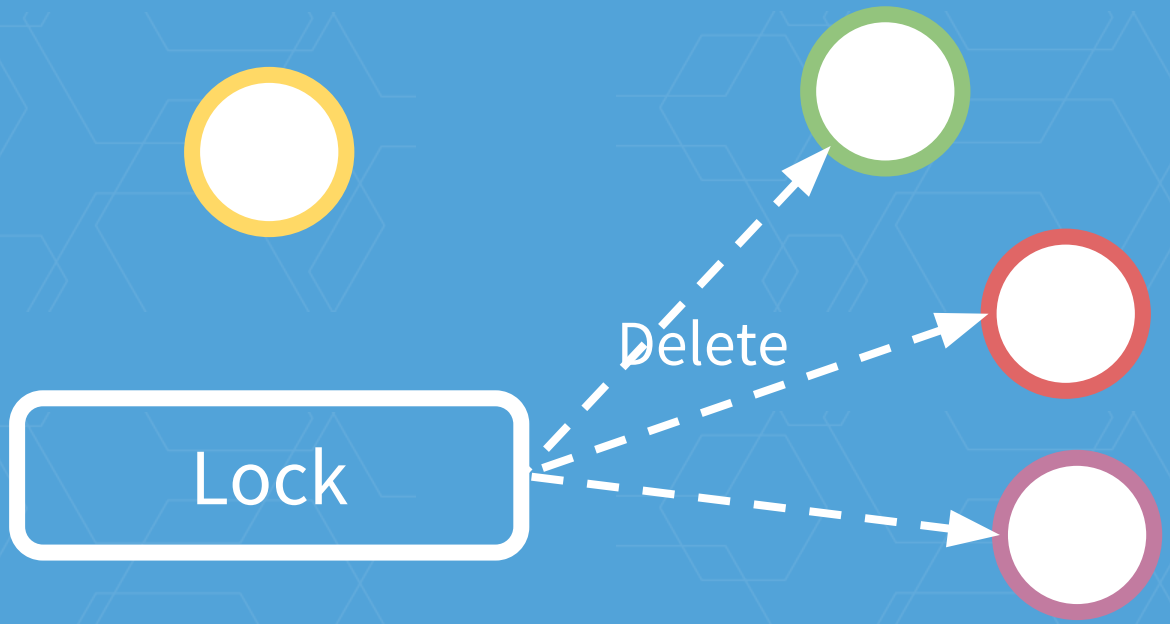
Distributed locking

- Herd effect



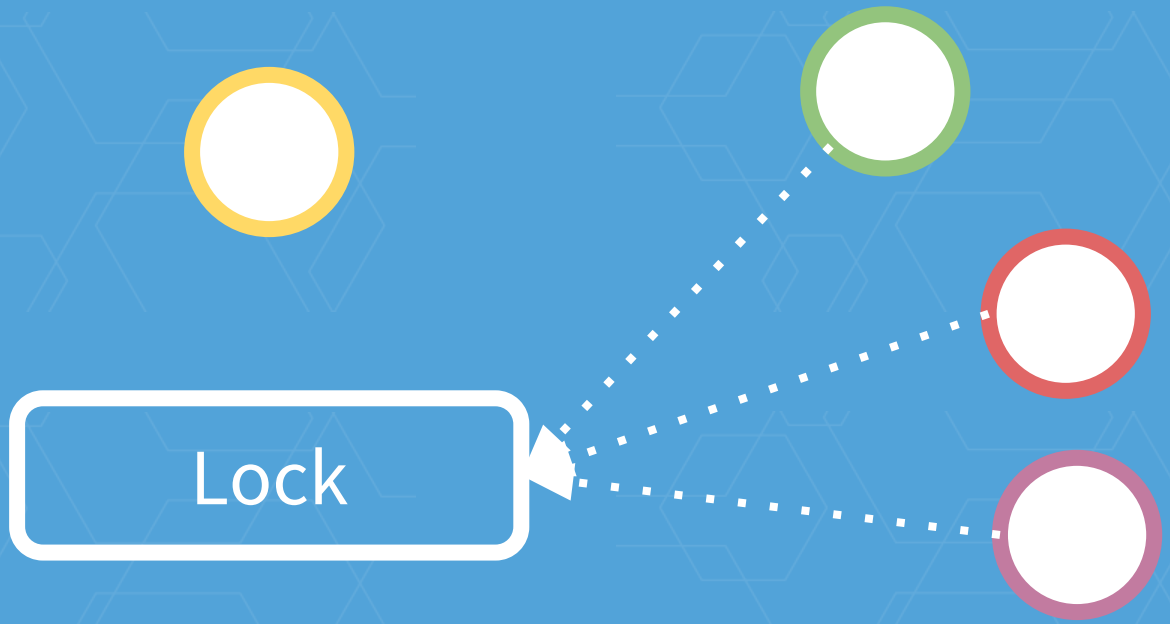
Distributed locking

- Herd effect



Distributed locking

- Herd effect



Distributed locking

- Herd effect

Distributed locking

- Herd effect

A better mutex

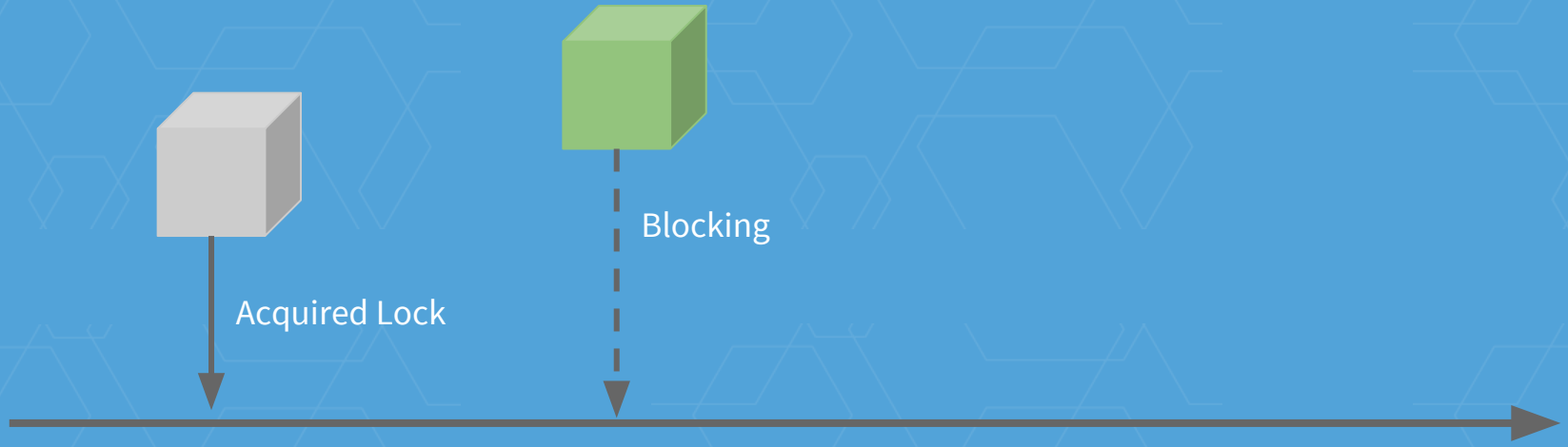
```
mylock = lock + ID
for {
    resp = Txn(ctx).If(!exist(mylock)).
        Then(put(mylock, "").
        Else(get(mylock)).
        Commit()
    myTurn = mylock.Rev
    waitUntil(lock.delete, myTurn-1)
}
```

Distributed locking

- Deadlock prevention
  - machine fails to unlock before goes down

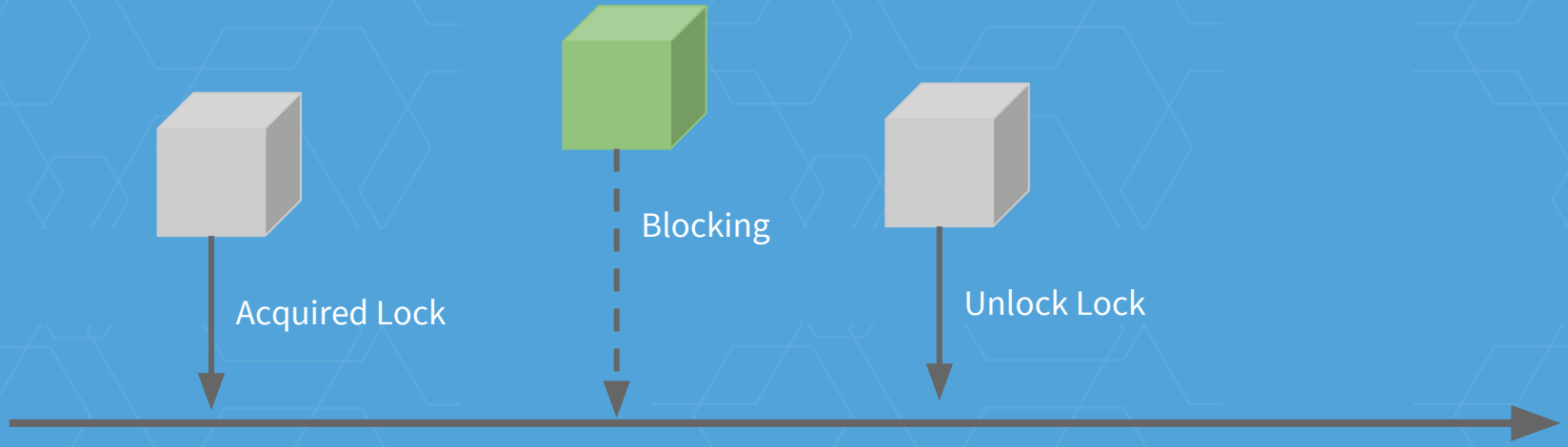
Distributed locking

- Deadlock prevention
  - machine fails to unlock before goes down



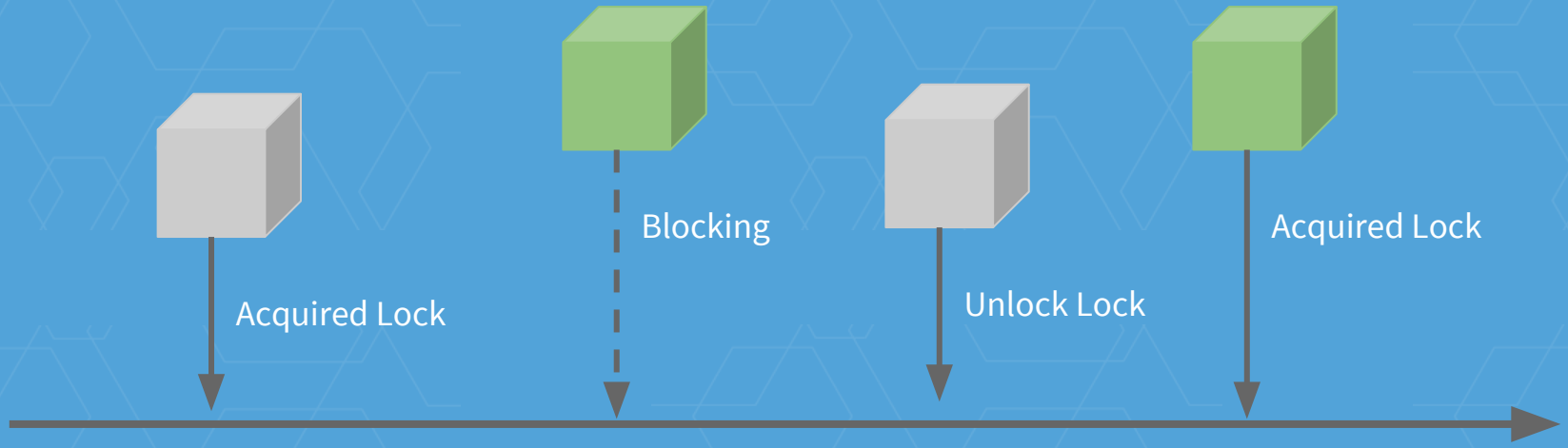
Distributed locking

- Deadlock prevention
  - machine fails to unlock before goes down



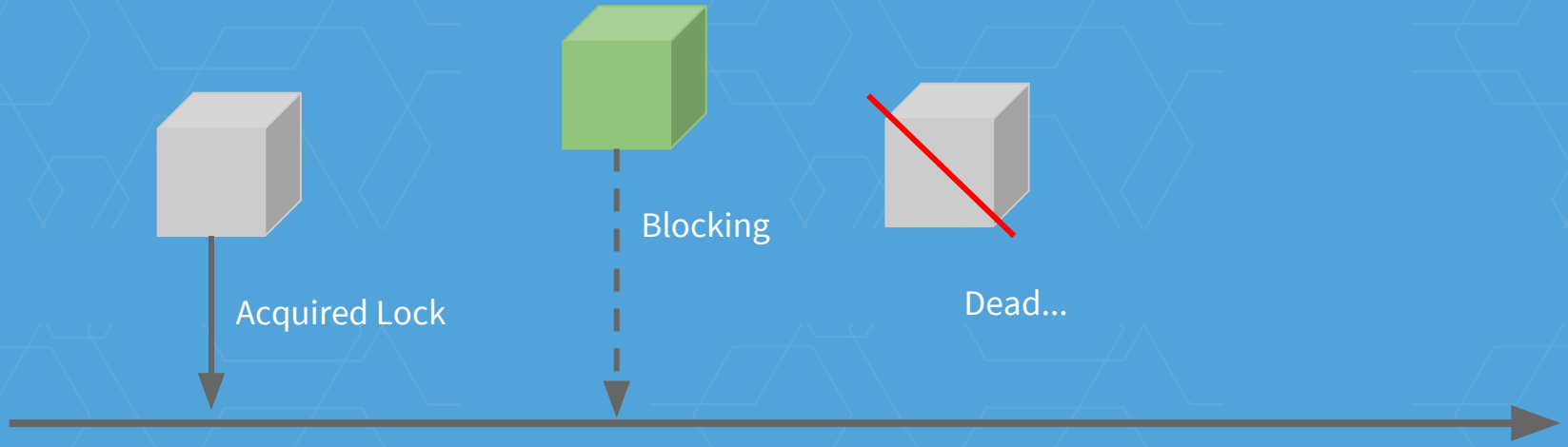
Distributed locking

- Deadlock prevention
  - machine fails to unlock before goes down



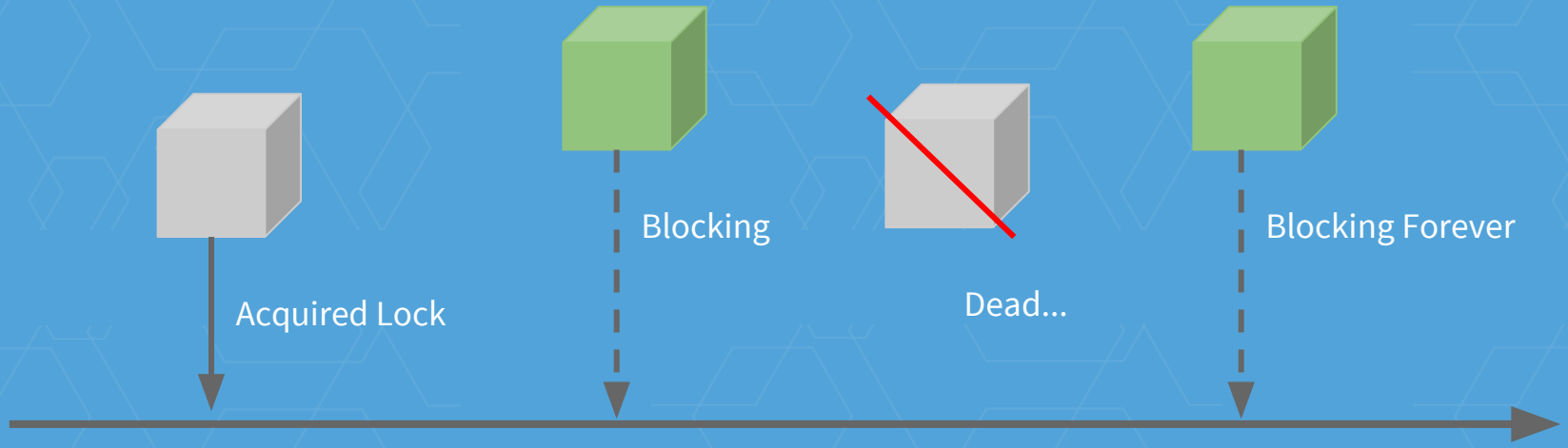
Distributed locking

- Deadlock prevention
  - machine fails to unlock before goes down



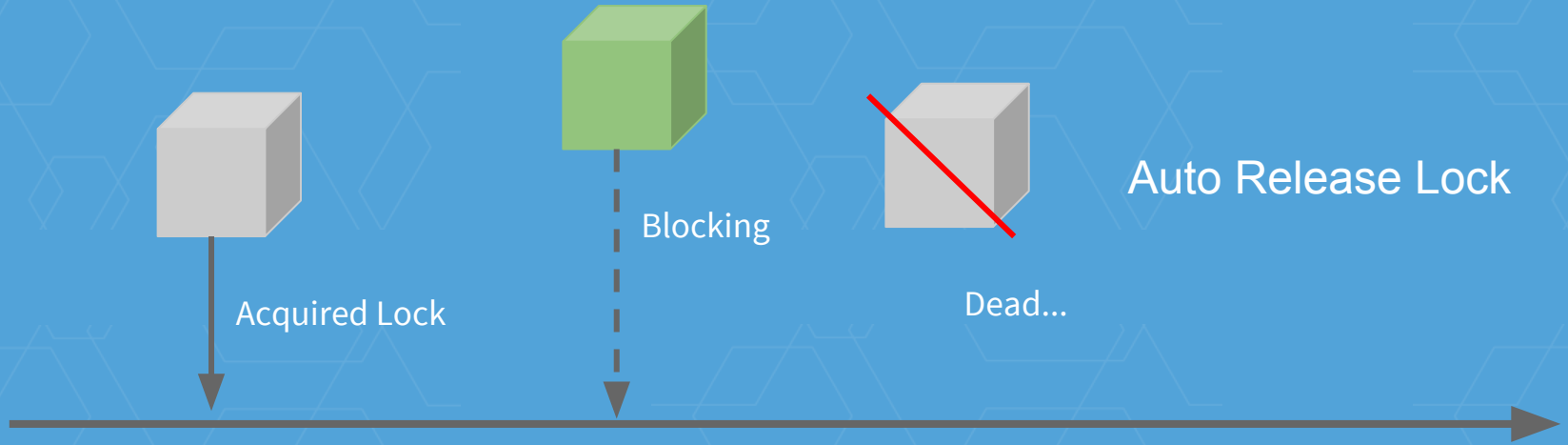
Distributed locking

- Deadlock prevention
  - machine fails to unlock before goes down



Distributed locking

- Deadlock prevention
  - machine fails to unlock before goes down

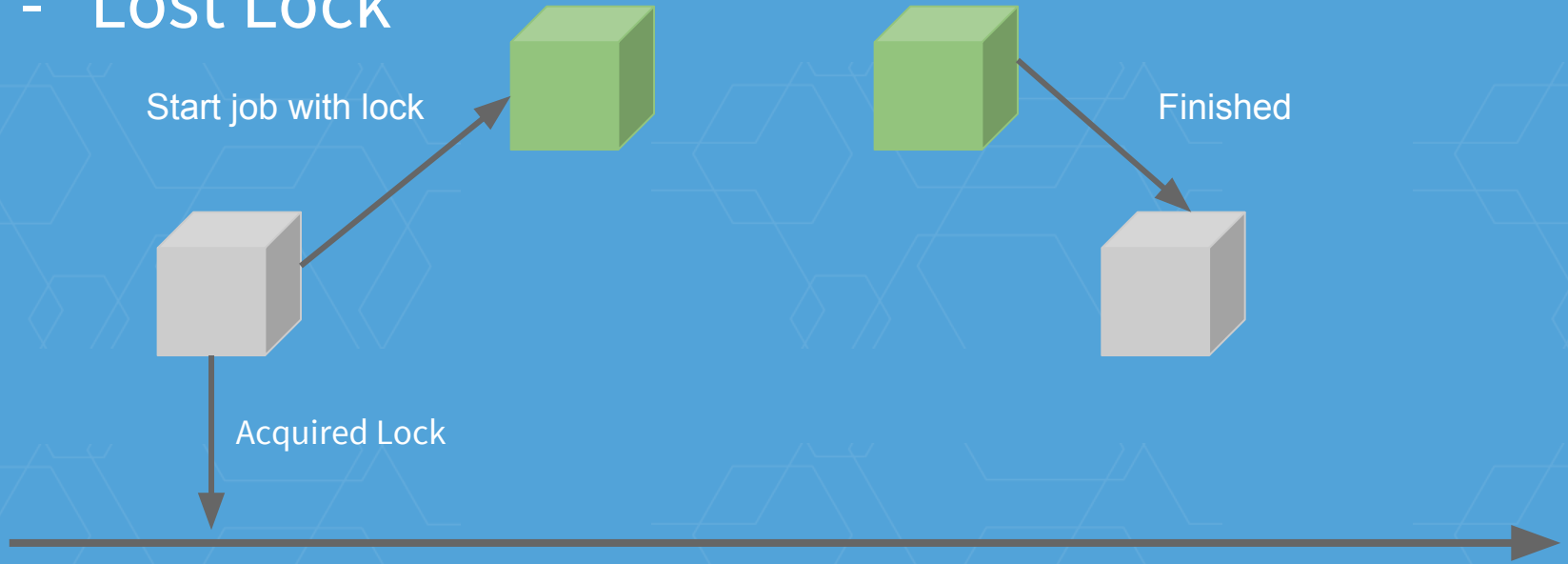


Distributed locking

- Lost Lock

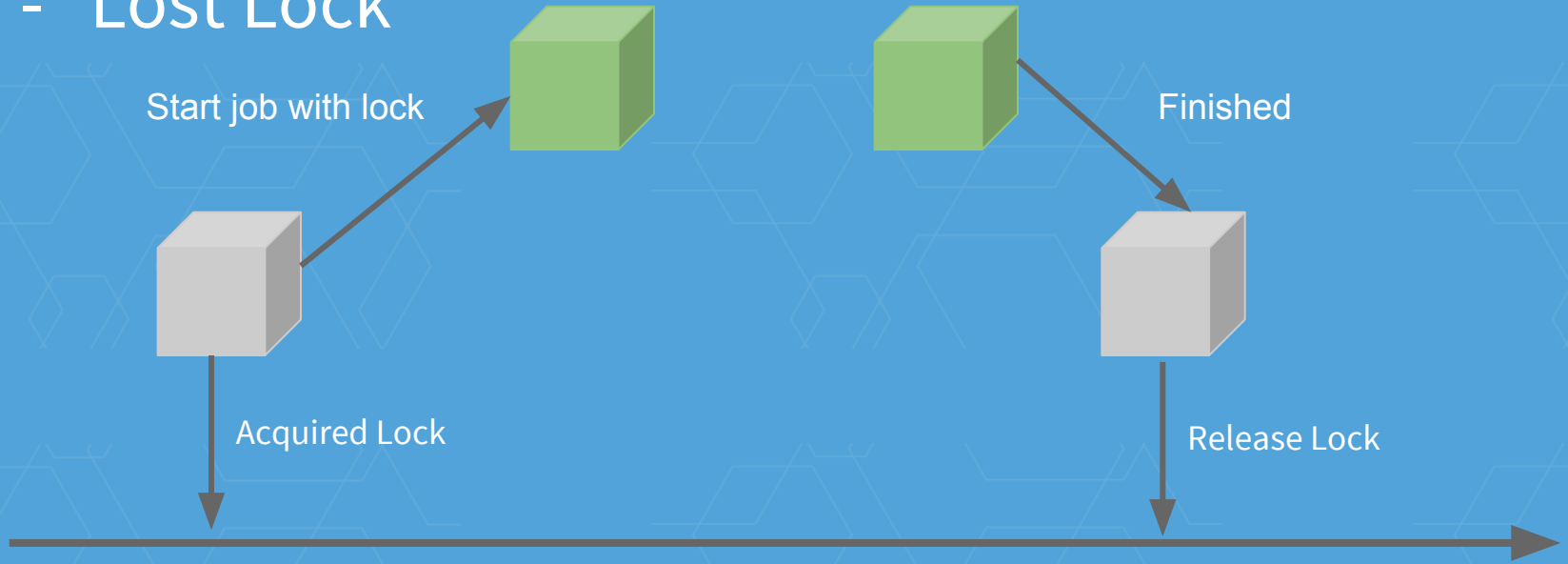
Distributed locking

- Lost Lock



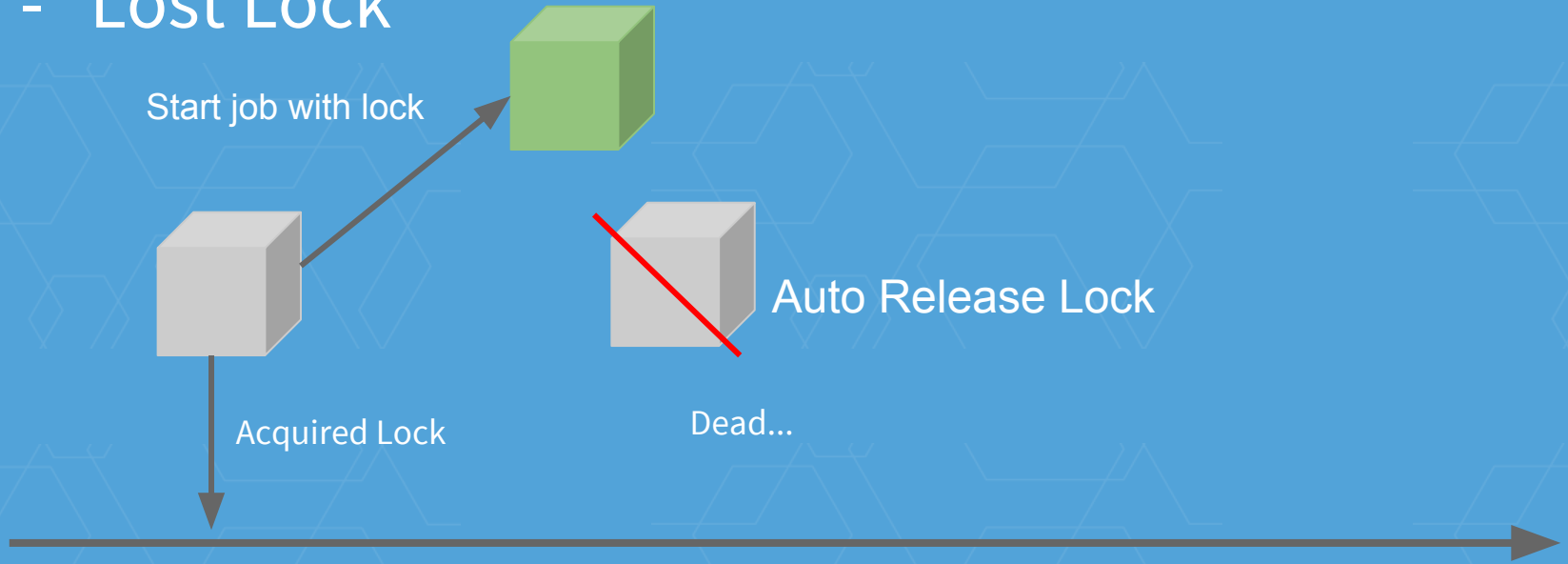
Distributed locking

- Lost Lock



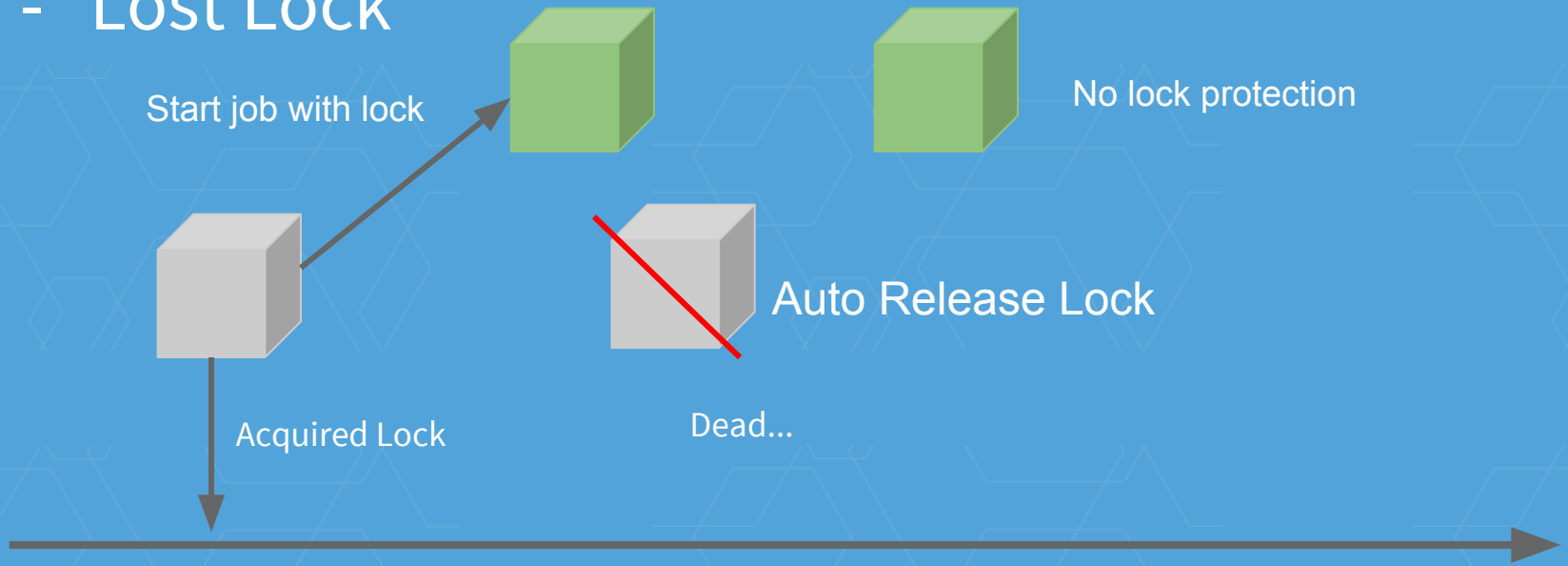
Distributed locking

- Lost Lock



Distributed locking

- Lost Lock

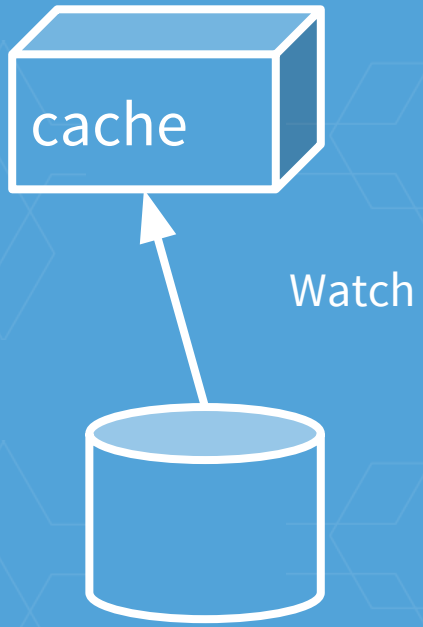


Distributed locking

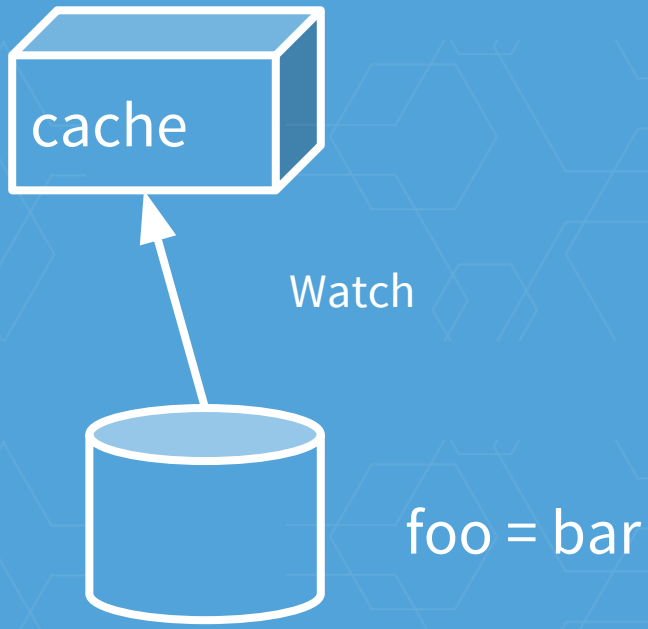
No prefect distributed lock

- Distributed lock provides different guarantees than thread.Lock
- Know what your system really need

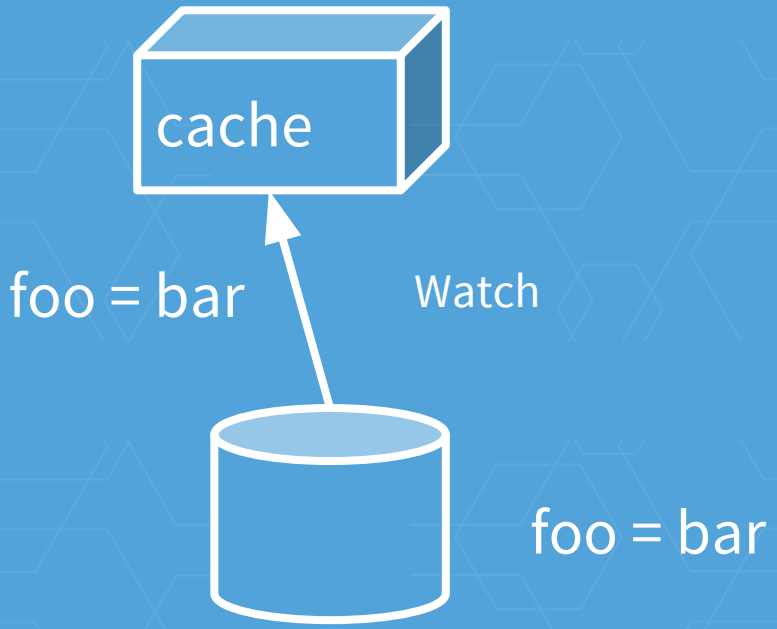
Efficient caching



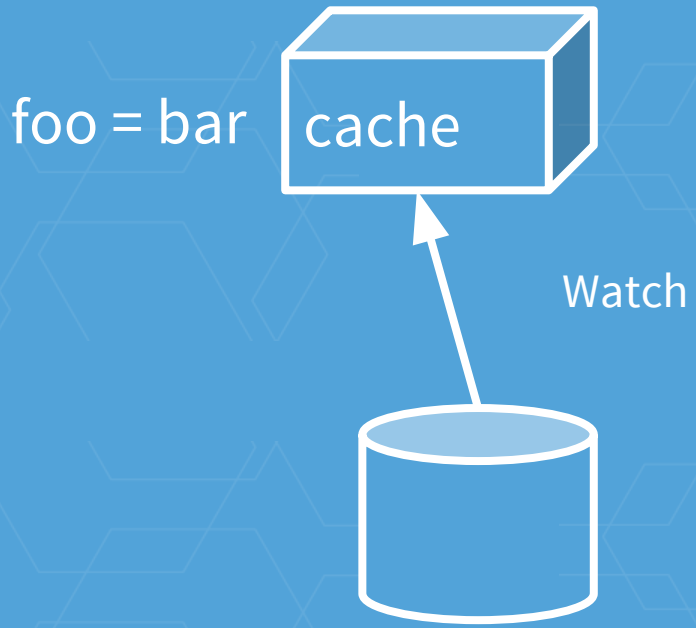
Efficient caching



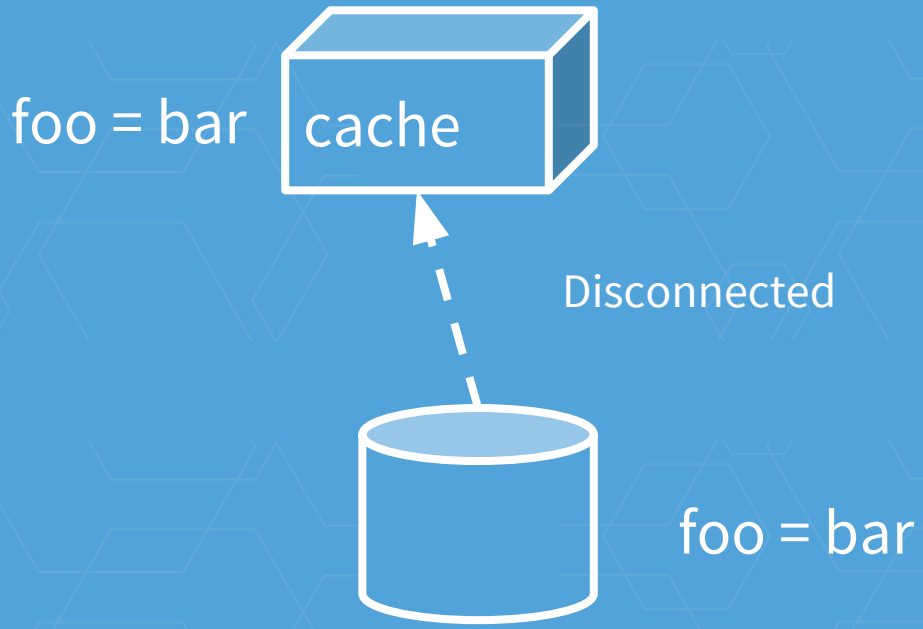
Efficient caching



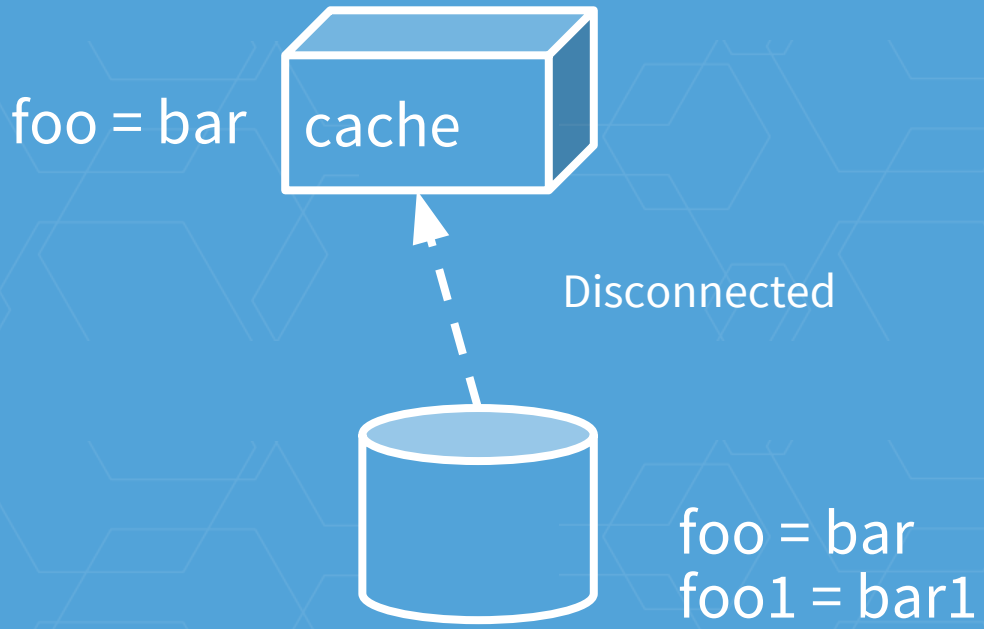
Efficient caching



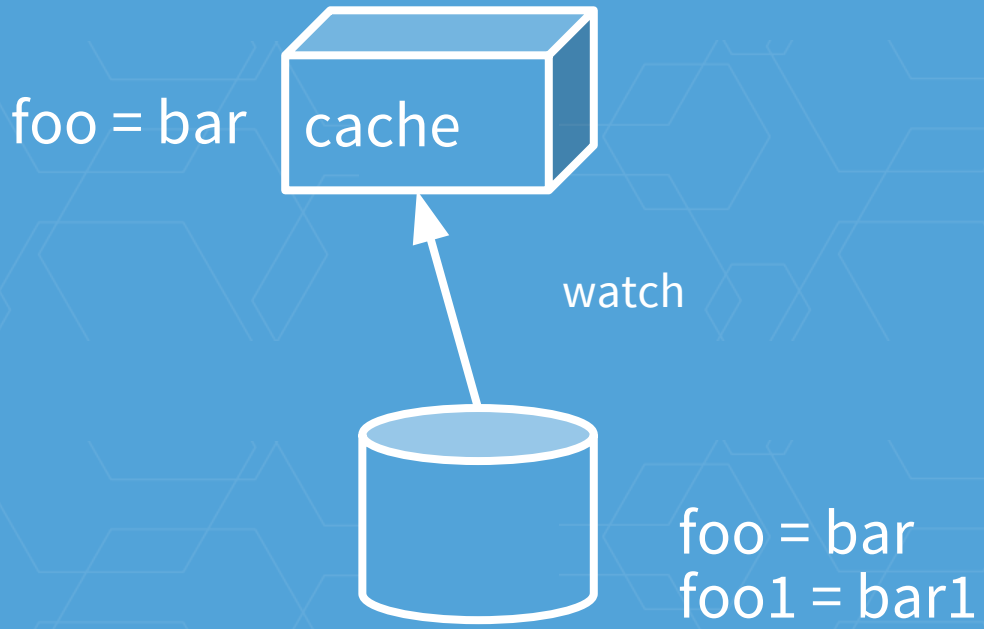
Efficient caching



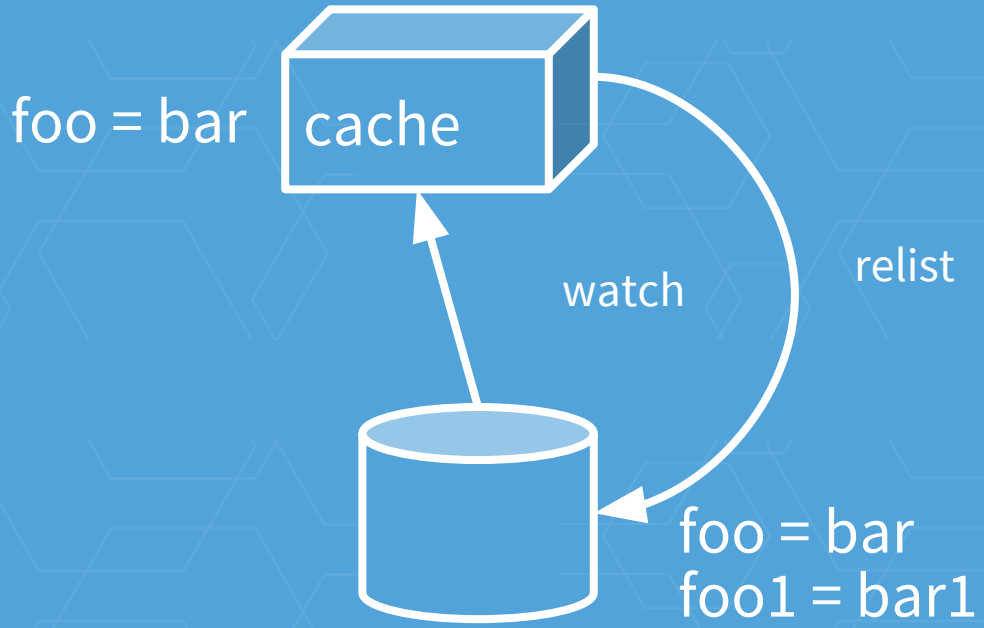
Efficient caching



Efficient caching



Efficient caching



Efficient caching

Relist can be expensive

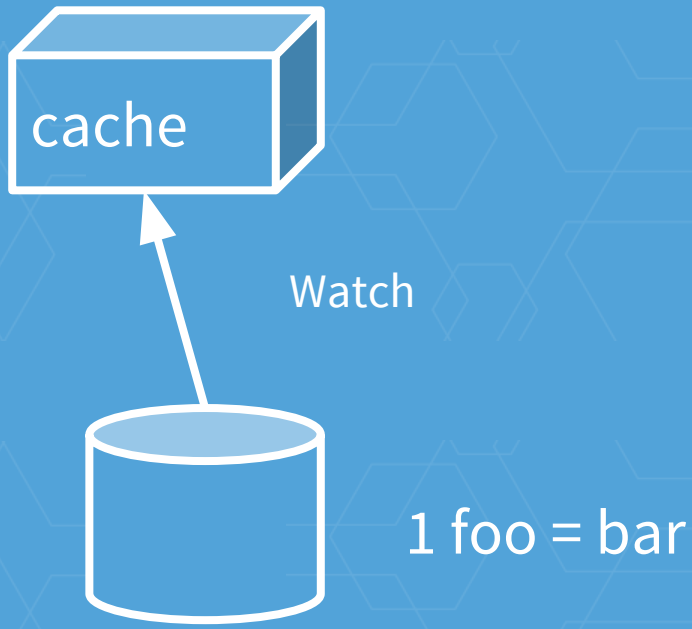
- Millions of items

Efficient caching

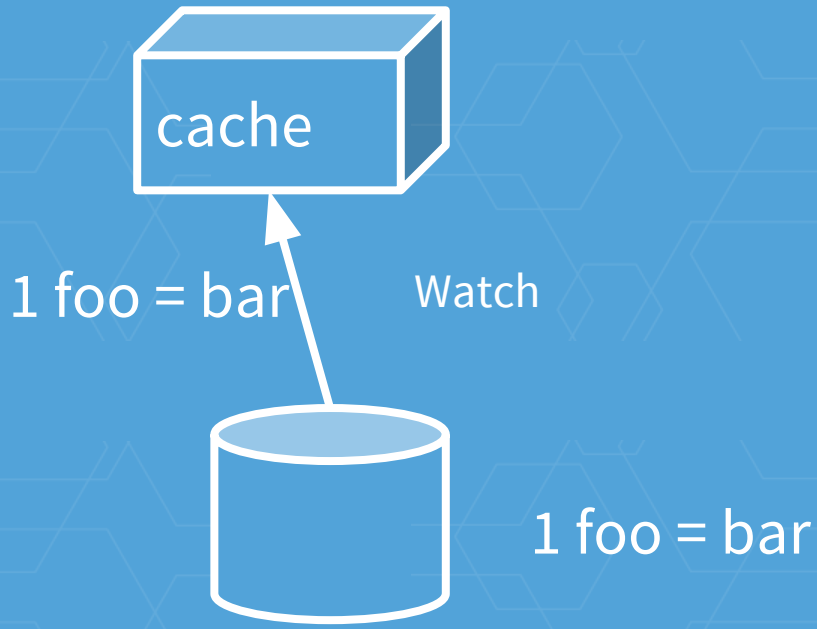
etcd supports watch from previous revision

- Critical feature for failure recovery

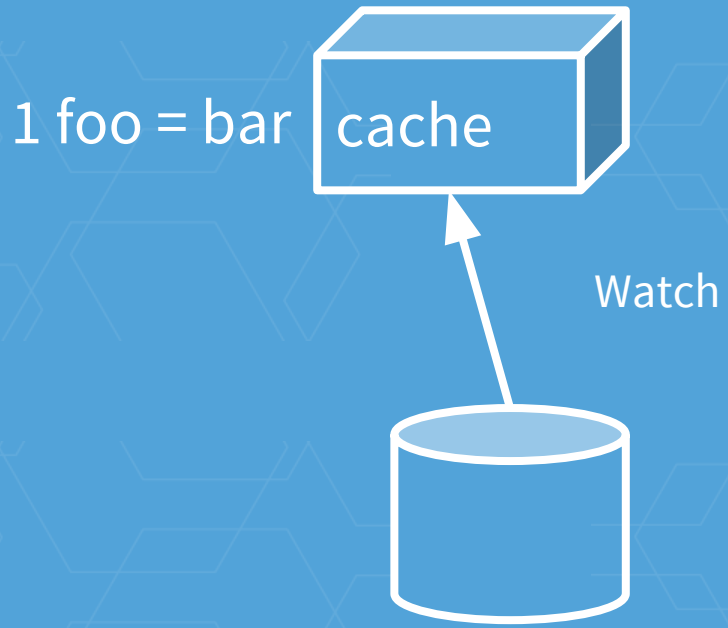
Efficient caching



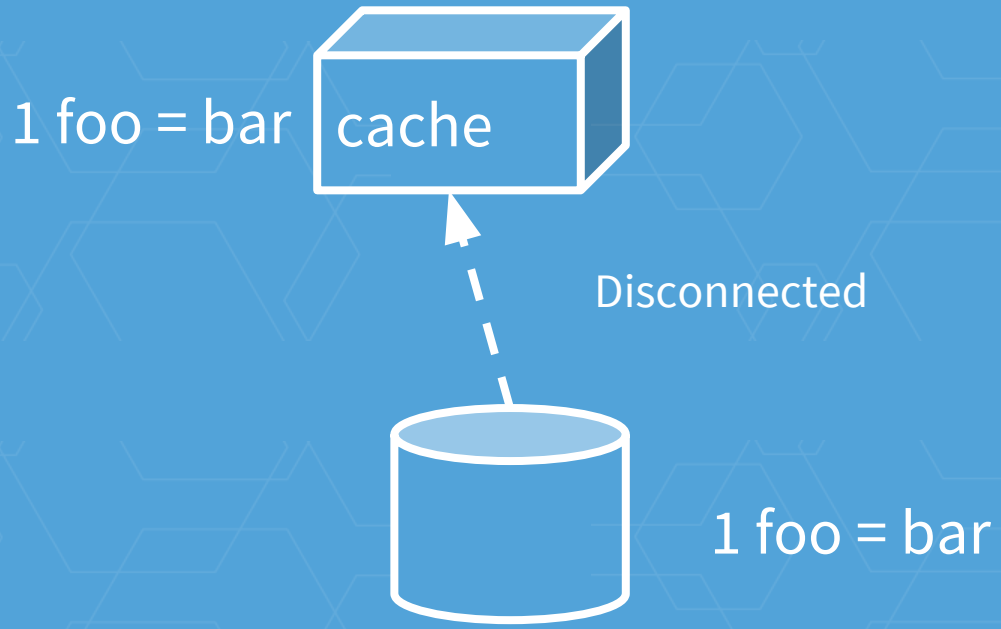
Efficient caching



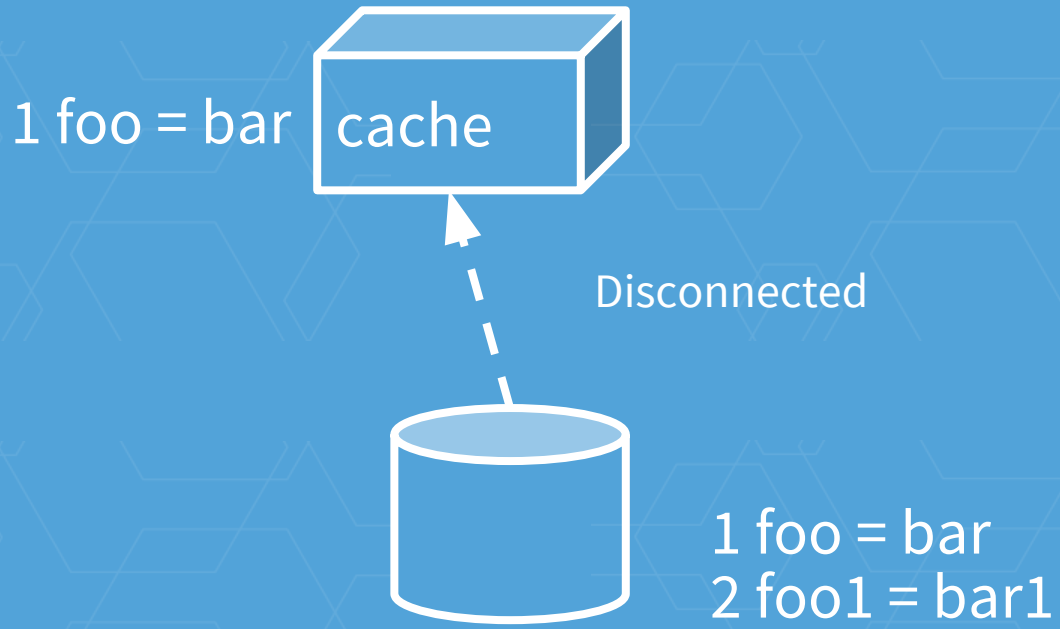
Efficient caching



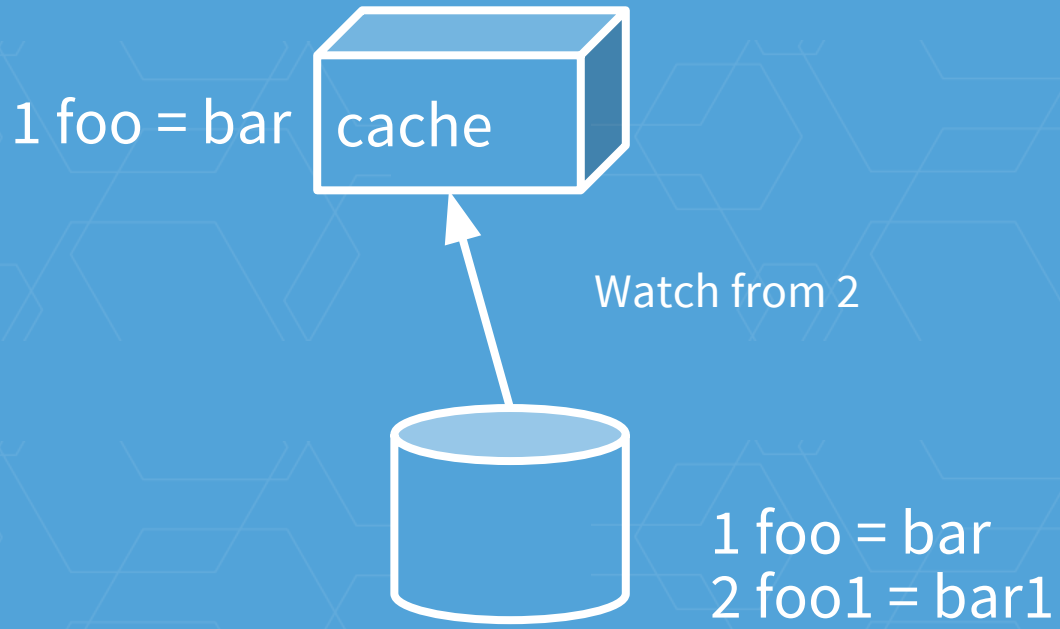
Efficient caching



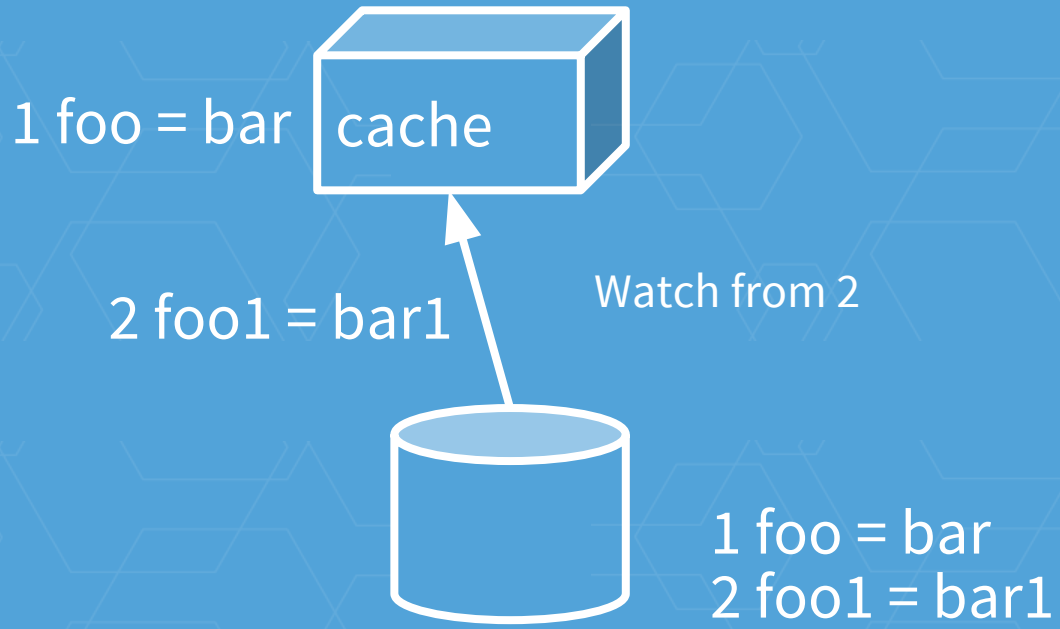
Efficient caching



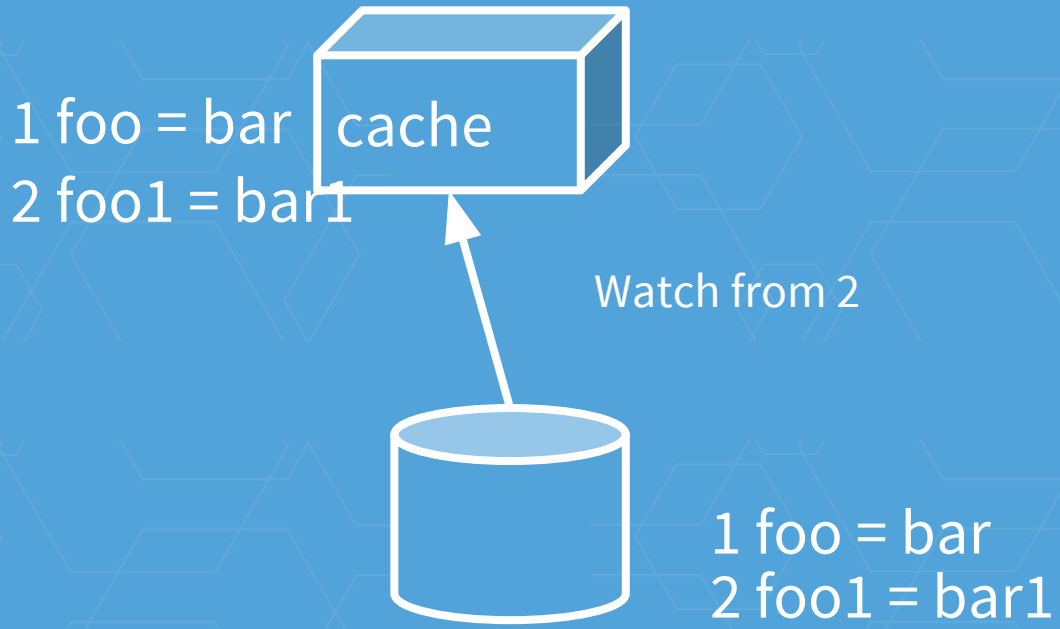
Efficient caching



Efficient caching



Efficient caching



Consistency

Raft as Replication Engine

- standard implementation described in paper
- active community

Consistency

Raft

- elect a stable leader via voting
  - detect leader loss via heartbeating
- leader replicates commands

Consistency

etcd/raft

- designed for correctness and performance
- used by a lot of serious projects
- cockroachdb
  - Open source Spanner
- tikv
  - Open source distributed transactional KV

Consistency

- Sequential Consistency
  - It is NOT possible to put a key and get it from any node in the cluster back immediately
  - Tricks are there to “cheat” you

Consistency

- Sequential Consistency
  - distributed adder

Node 1	Node 2	Node 3
+1 (1)	+1 (1)	+1 (1)

Consistency

- Sequential Consistency
  - distributed adder

Node 1	Node 2	Node 3
+1 (1)	+1 (1)	+1 (1)
+5 (6)		

Consistency

- Sequential Consistency
  - distributed adder

Node 1	Node 2	Node 3
+1 (1)	+1 (1)	+1 (1)
+5 (6)	+5 (6)	
	+ 7 (13)	

Consistency

- Sequential Consistency
  - distributed adder

Node 1	Node 2	Node 3
+1 (1)	+1 (1)	+1 (1)
+5 (6)	+5 (6)	+5 (6)
+ 7 (13)	+ 7 (13)	+ 7 (13)

Consistency

- Eventual Consistency
  - distributed adder

Node 1	Node 2	Node 3
+8 (8)	+100 (100)	+3 (3)

Consistency

- Eventual Consistency
  - distributed adder

Node 1	Node 2	Node 3
+8 (8)	+100 (100)	+3 (3)
+8 (16)	+8 (108)	+8 (11)
Reset! (0)	Reset! (0)	Reset! (0)

Reliability

99% at small scale is easy

- failure is infrequent and human manageable

99% at large scale is not enough

- human unmanageable

99.999% at large scale

- trustable system at bottom layer

Reliability

WAL

- write operations before executing it
- never truncate
- rolling CRC protected

Reliability

Snapshot

- append-only B+Tree
- easy to prevent corruption

Torturing Databases for Fun and Profit

- complicated database crashes

Reliability

Resource Control

- back pressure
- offhead storage
- incremental snapshot

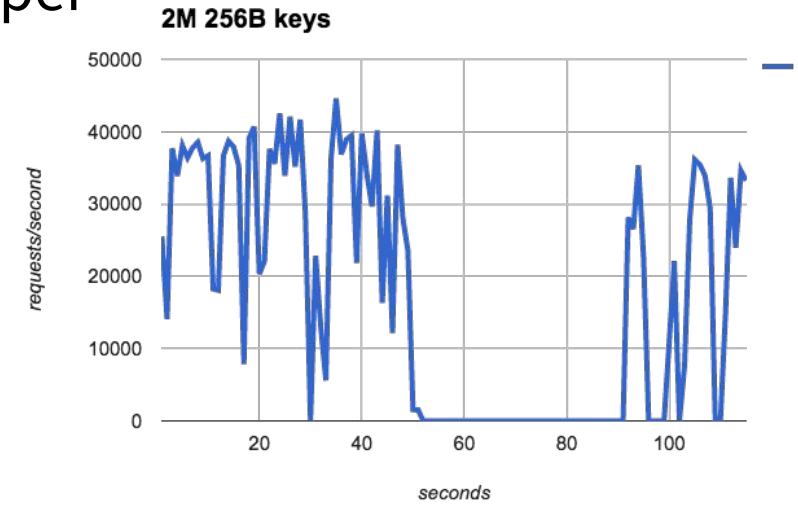
Reliability

Extensive testing

- dash.etcd.io

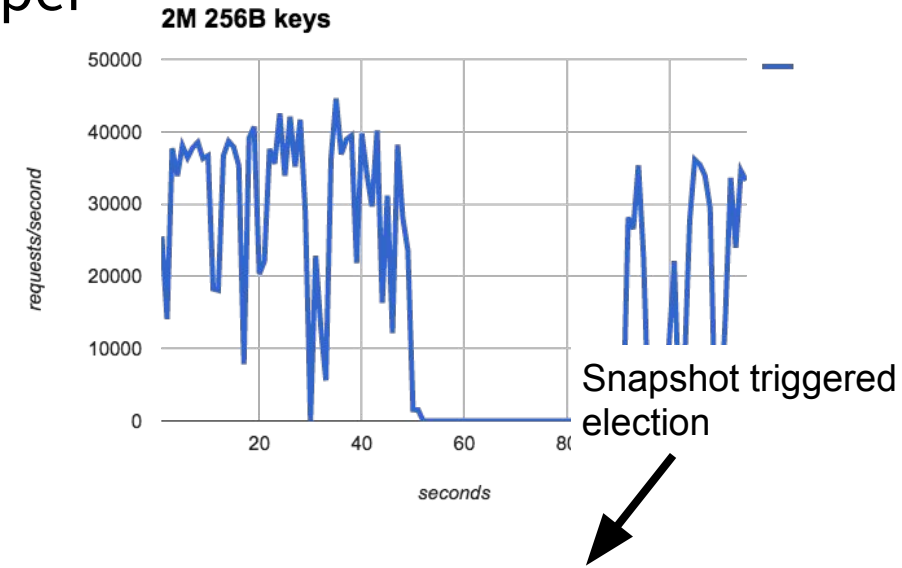
Performance

ZooKeeper



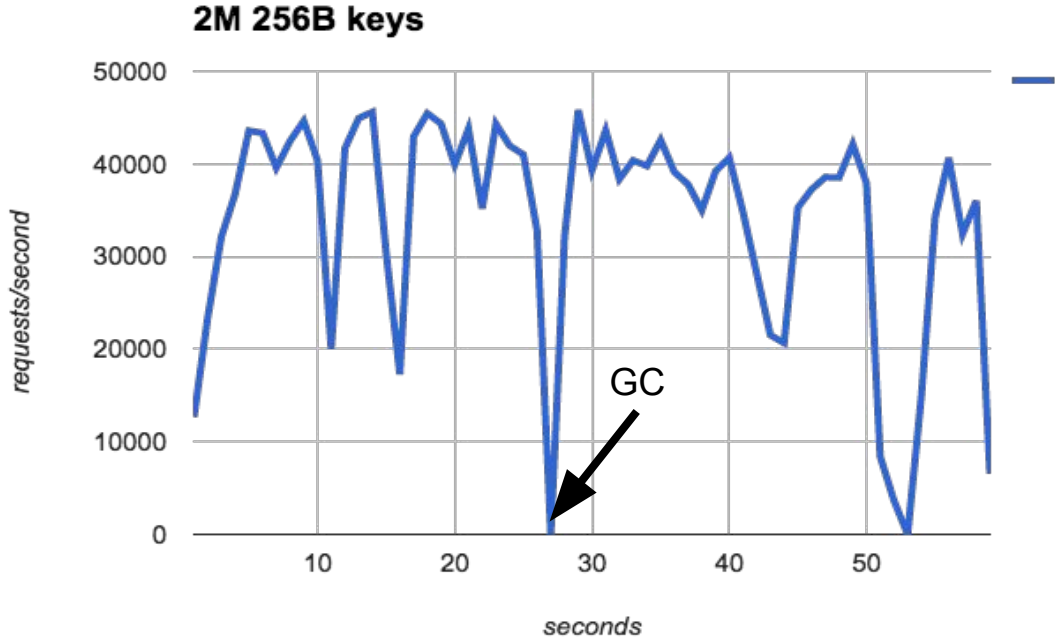
Performance

ZooKeeper



Performance

ZooKeeper - snapshot disabled



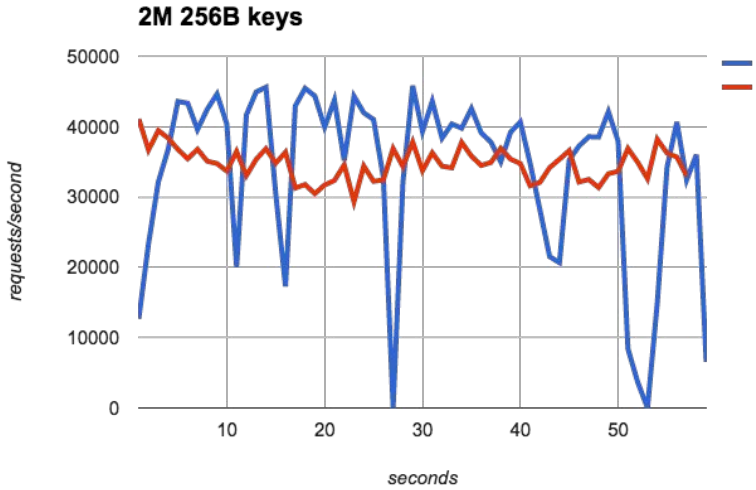
Reliable Performance

- Similar to ZooKeeper with snapshot disabled
  - Incremental snapshot

- No Garbage Collection Pauses
  - Off-heap storage

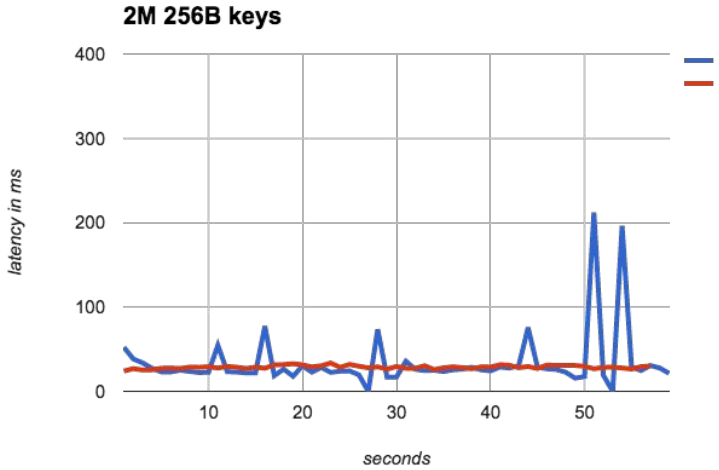
Performance

ZooKeeper vs etcd



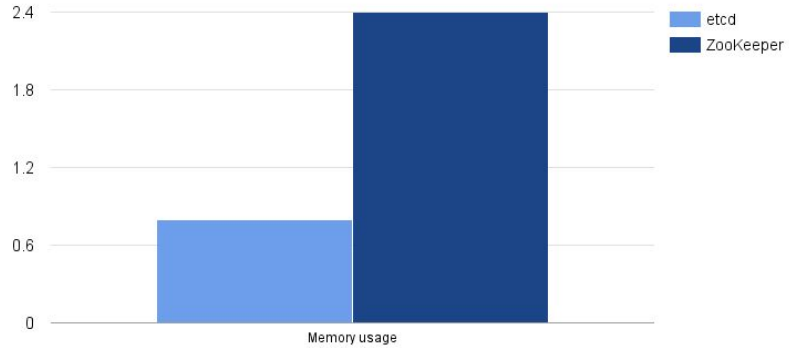
Performance

ZooKeeper vs etcd



Performance

ZooKeeper vs etcd



Operation

- Runtime Reconfiguration
- Point-in-time Backup
- Extensive Metrics
- Database Size Quota

Proxy

- A Horizontally Scalable Layer of etcd
- Key-Value Pair Caching
- Watch Coalescing
- …

etcd vs Other Projects

- Do one thing
- Only do one thing
- Do it REALLY well

etcd vs Other Projects

- Do one thing
- Only do one thing
- Do it REALLY well

Reliability

Thanks