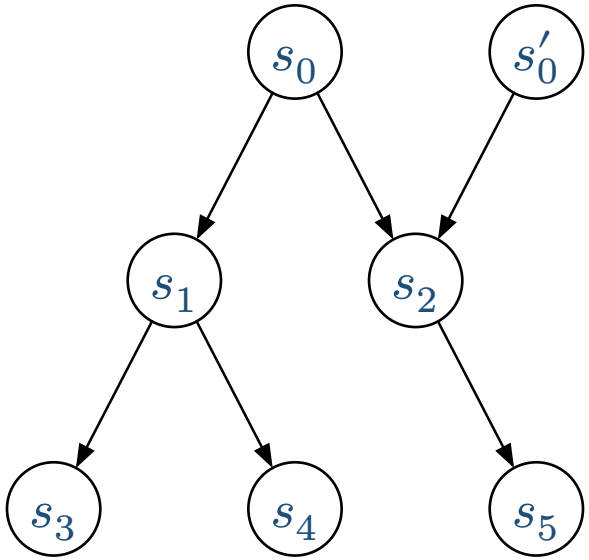
UNIVERSITY OF CAMBRIDGE

Modelling orchestration

Andrew Jeffery, supervised by Prof. Richard Mortier

24th October 2024 @SRG

What is model checking?



Model checkers

Stateright Shuttle TLA+ (TLC)

What is orchestration?

The automated management of a system

Orchestrators

Kubernetes

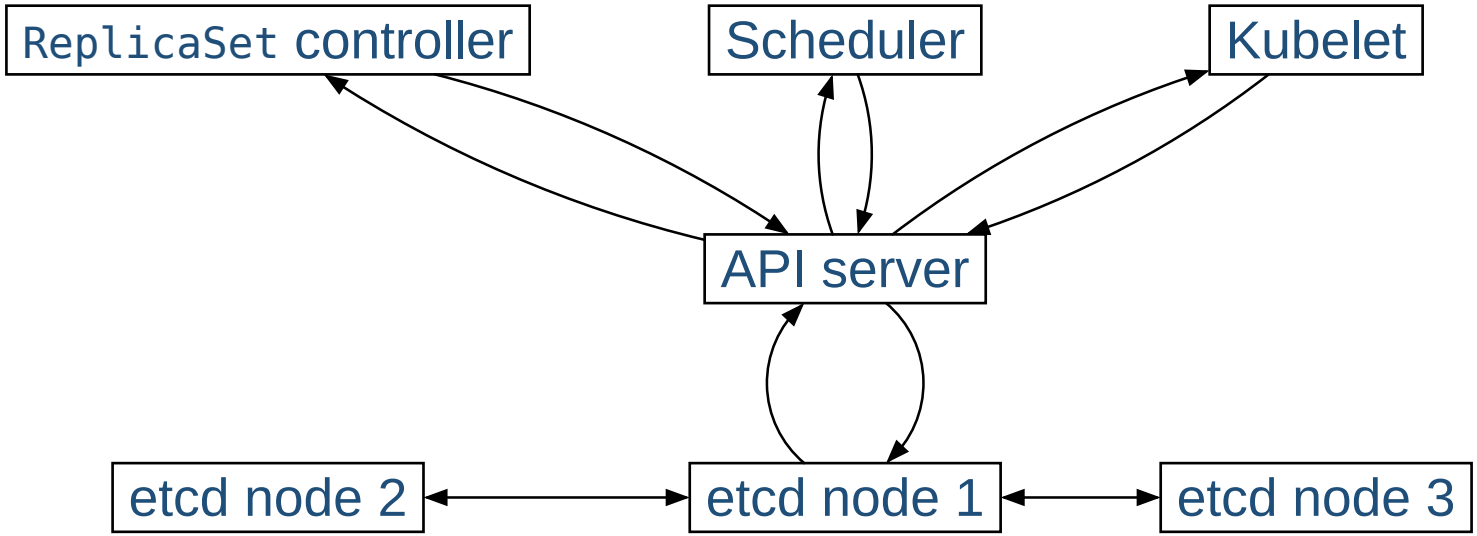
Mesos

Deployment environments

Environment	Private datacenter	Public Cloud	Near-edge (AWS Wavelength) (`r5.2xlarge`)
vCPUs	192 / 120 (/socket)	192	8
Memory	?	768 GiB	64 GiB
Disk Capacity	?	Elastic	Elastic
Disk Bandwidth	?	50 Gbps	<4.75 Gpbs
LAN Latency (RTT)	<1ms	0.3ms (Between AWS AZs)	<1ms
LAN Bandwidth	?	50 Gbps	<10 Gbps

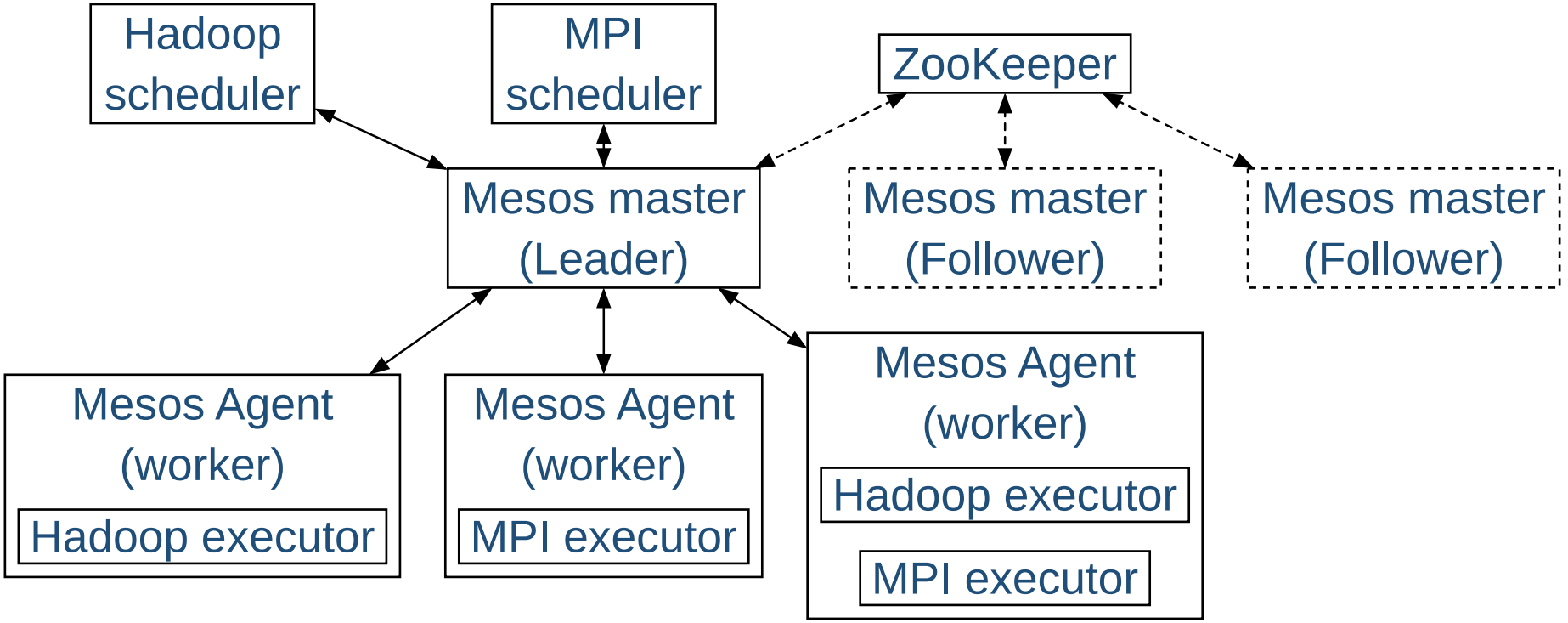
Comparison of maximum likely resources per machine in each environment.

Kubernetes architecture

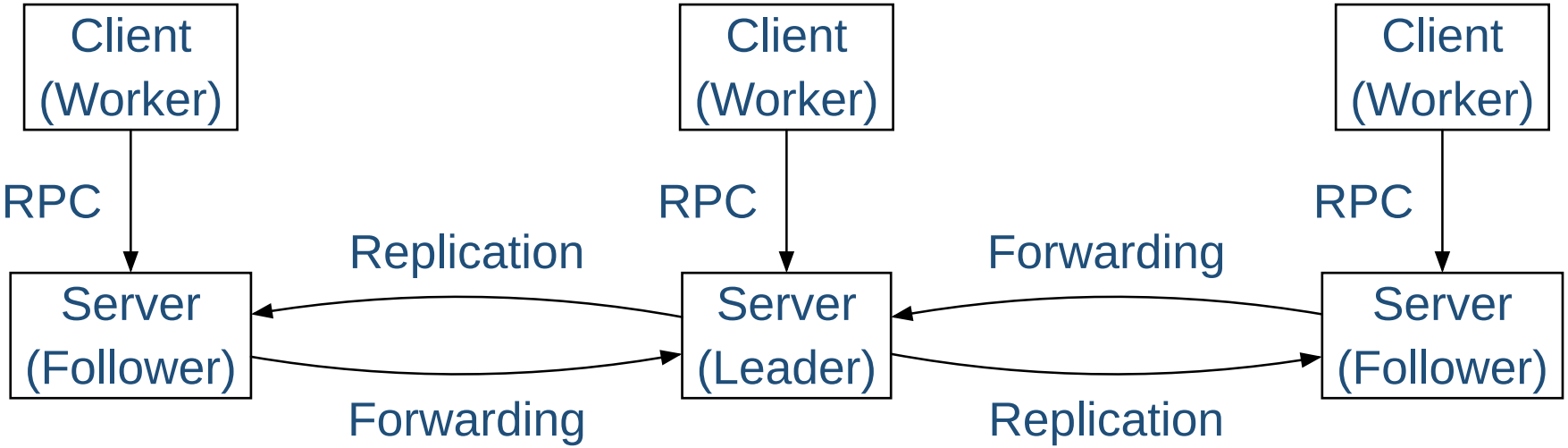


Flow of requests to schedule a application instance from creation in etcd.

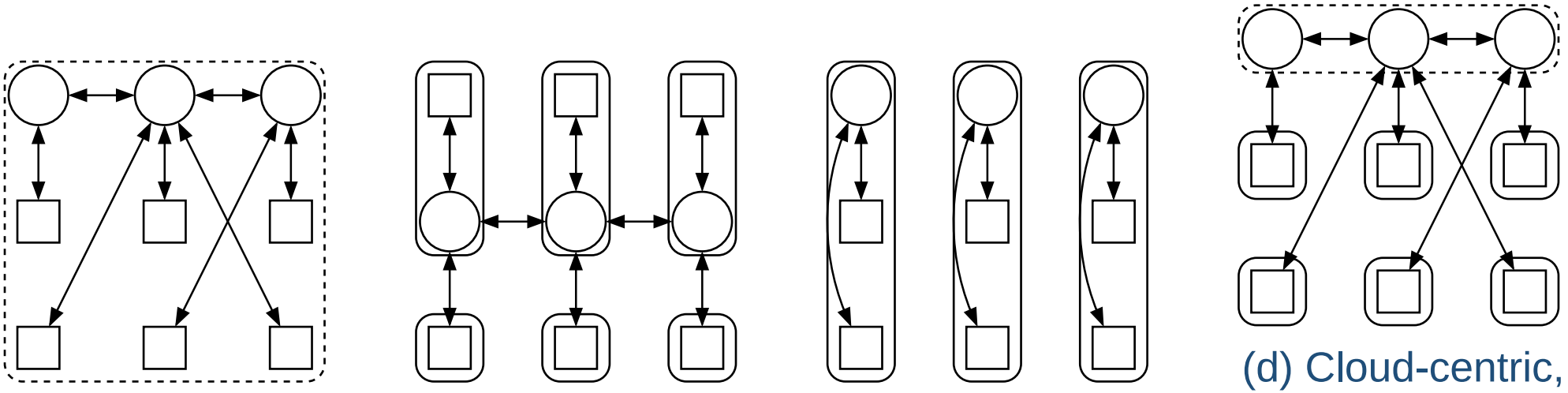
Other orchestration platforms: Mesos



Other orchestration platforms: Nomad



Existing edge platforms

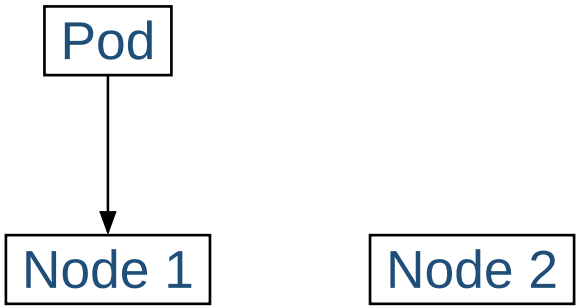


(a) All-cloud, K8s. (b) Multi-site, K8s. (c) Single-site, K3s. (d) Cloud-centric, KubeEdge.

Circles are control-plane and datastore nodes, squares are worker nodes.

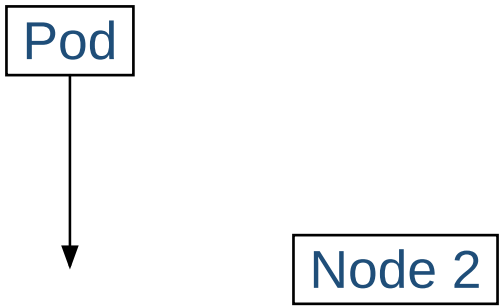
What is a scheduler

$$\text{scheduler} : N \rightarrow p \rightarrow n \quad n \in N, p \in P$$



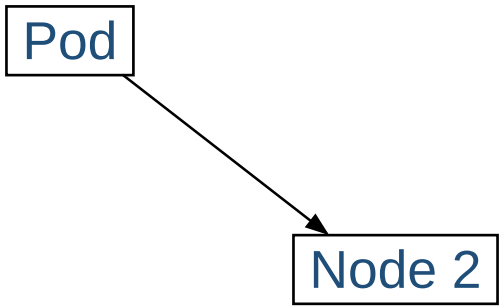
Making it fault tolerant: part 1

$$\text{scheduler} : N \rightarrow p \rightarrow n \quad n \in N, p \in P$$

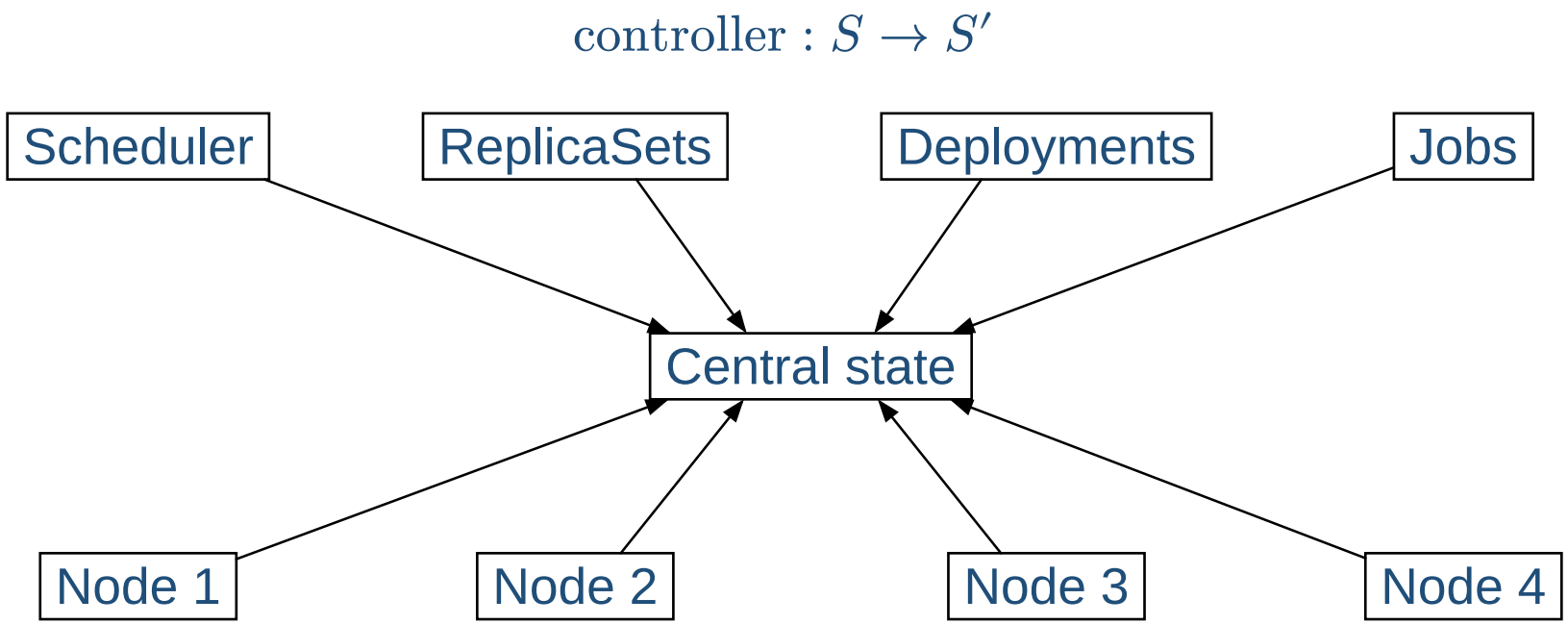


Making it fault tolerant: part 2

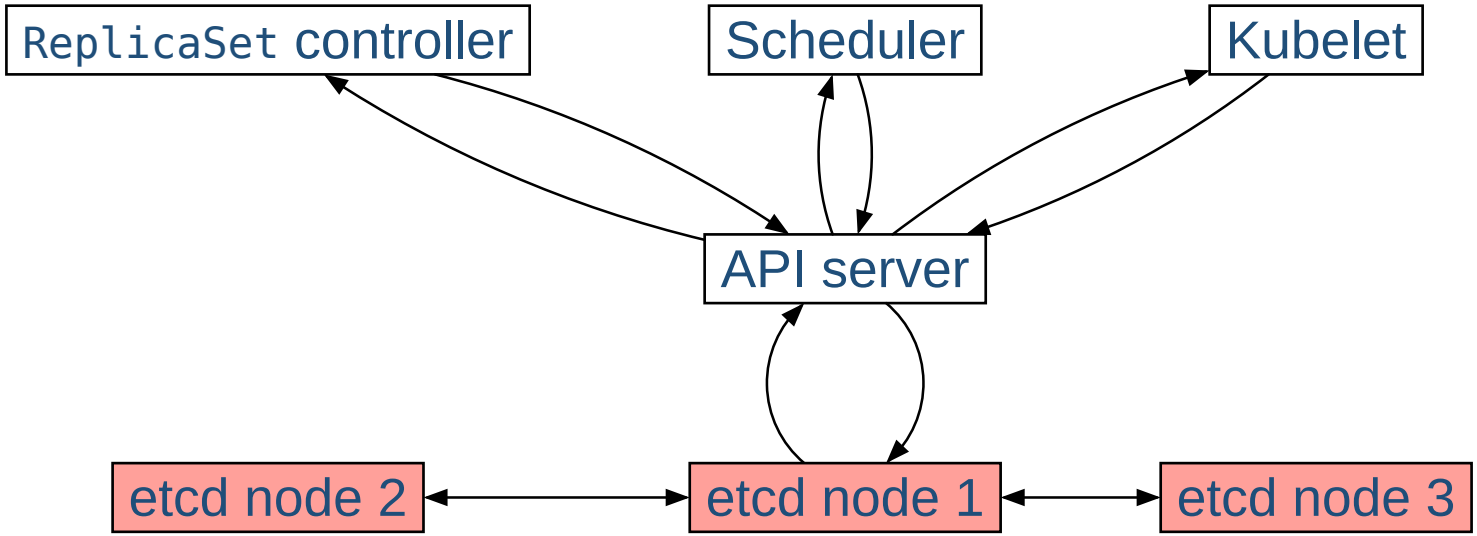
$$\text{scheduler} : N \rightarrow p \rightarrow n \quad n \in N, p \in P$$



Adding higher level control



What is the problem?



Flow of requests to schedule a application instance from creation in etcd.

Defining orchestration

An orchestration platform is a system of controllers $c \in C$ that operate on a state $s \in S$, driving the *current state* of the system to match the *desired state*.

State-based form

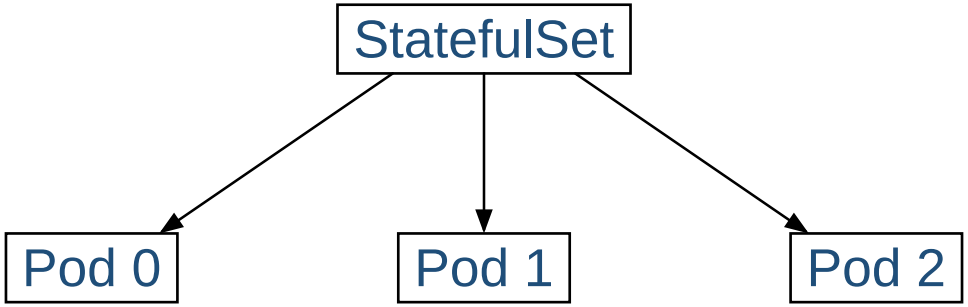
$$\text{Controller} : s \to s'$$

Op-based form

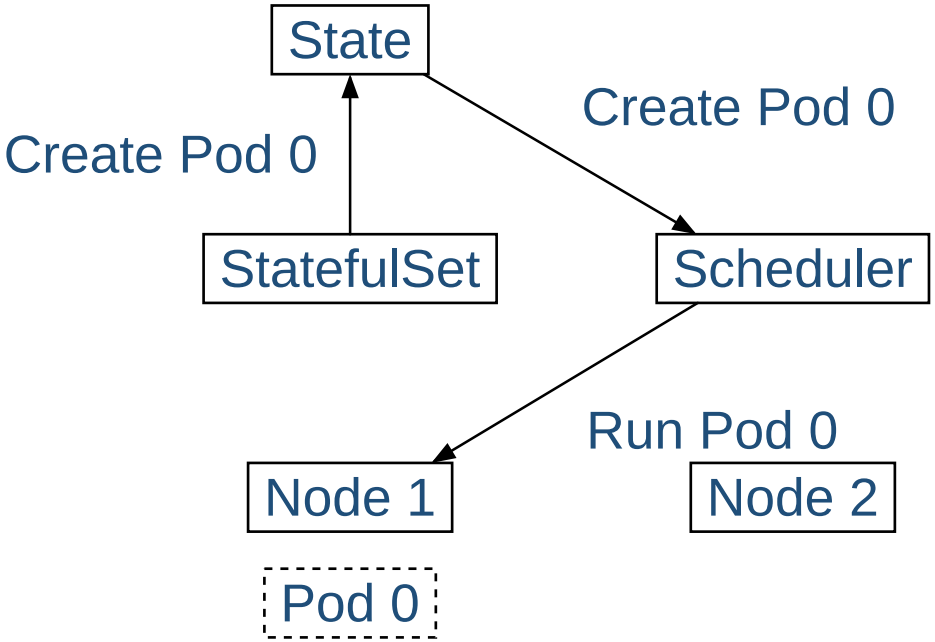
$$\text{Controller} : s \to o \quad s \in S, o \in O$$

$$\text{Apply}(s, o) = s'$$

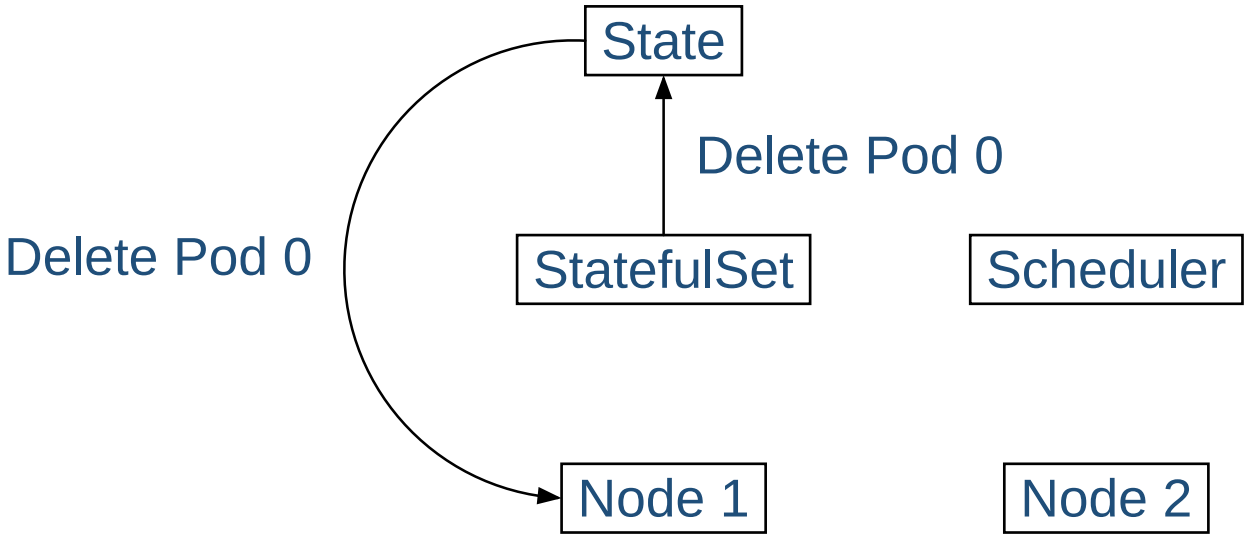
Statefulset background



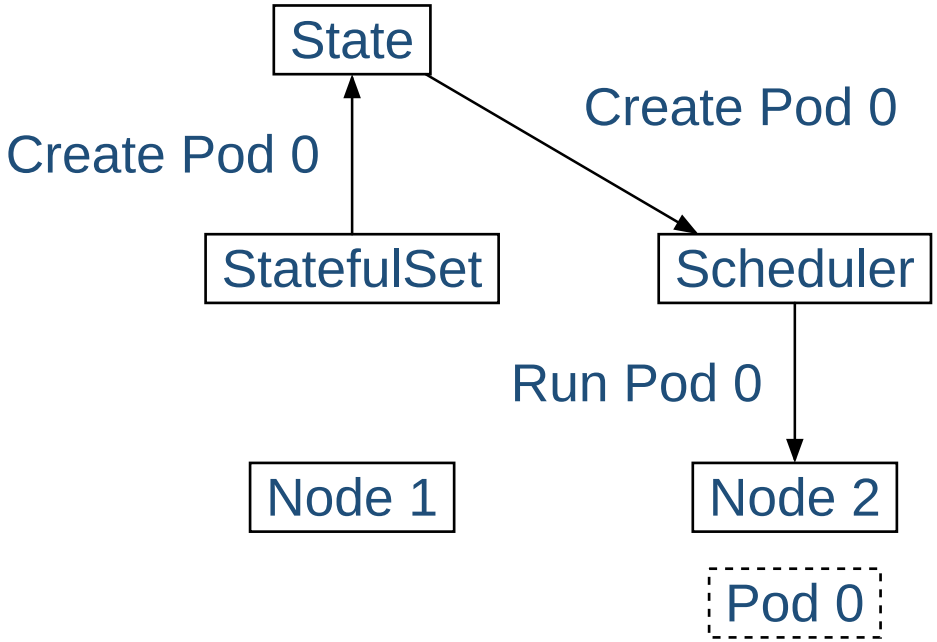
Example orchestration flow



Example orchestration flow



Example orchestration flow



Extracting the model

1. State
   - which controllers are in our system? (Statefulset, Scheduler, Nodes)
   - what is the state of resources? (Pod)
2. Operations
   - Defined by controllers in the system
   - Create Pod
   - Run Pod
   - Environmental
   - Restart node

Building a concrete model

- Themelios is the concrete model
- Rust for model implementation
- Stateright for model checking

Controller definition

```
trait Controller {
  type Operation: Into<ControllerOperation>;

  type State;

  fn step(&self, global_state: &StateView,
                 local_state: &mut Self::State)
                 -> Option<Self::Operation>;
}
```

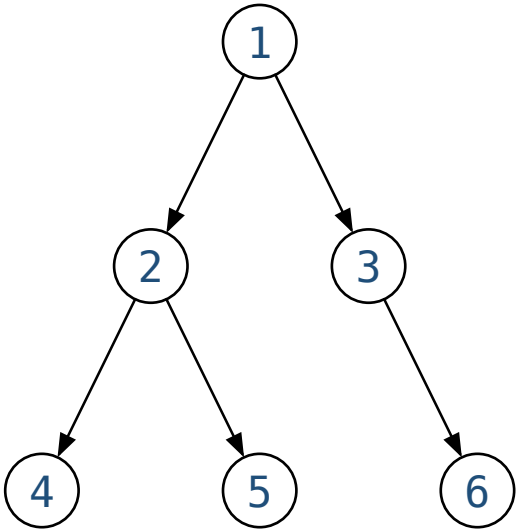
A sample property: unique names

Pods running on Nodes have unique names

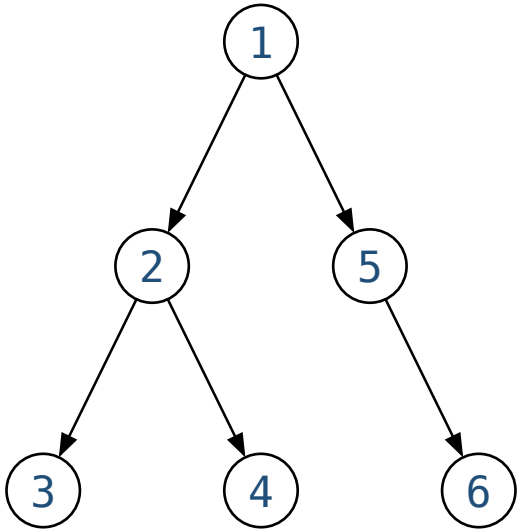
And in the model checker

```
properties.add(Expectation::Always, "node: pods on nodes are unique",
    |model, state| {
        let mut node_pods = BTreeSet::new();
        for c in 0..model.controllers.len() {
            let cstate = state.get_controller(c);
            if let ControllerStates::Node(n) = cstate {
                for node in &n.running {
                    if !node_pods.insert(node) {
                        return false; // property failed
        }   }   }   }
        true // property successful
    },
);
```

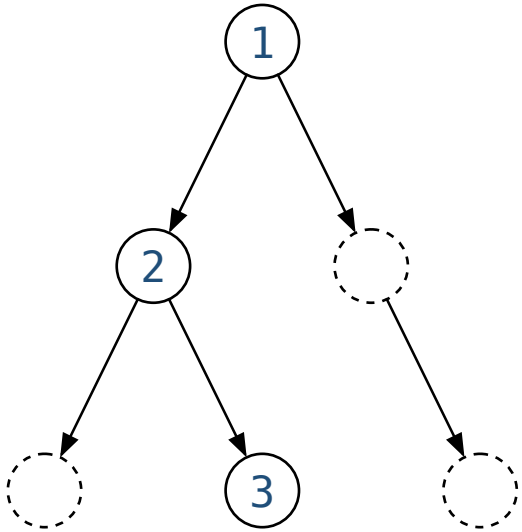
How can we execute the checker?



(a) Breadth-first search.



(b) Depth-first search.



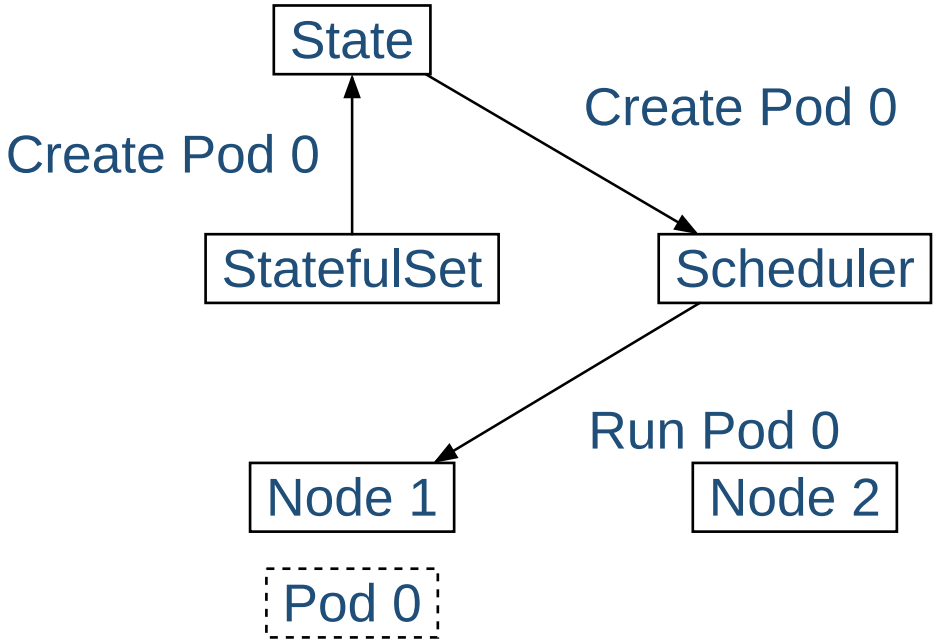
(c) Simulation search.

Traversal order of searches.

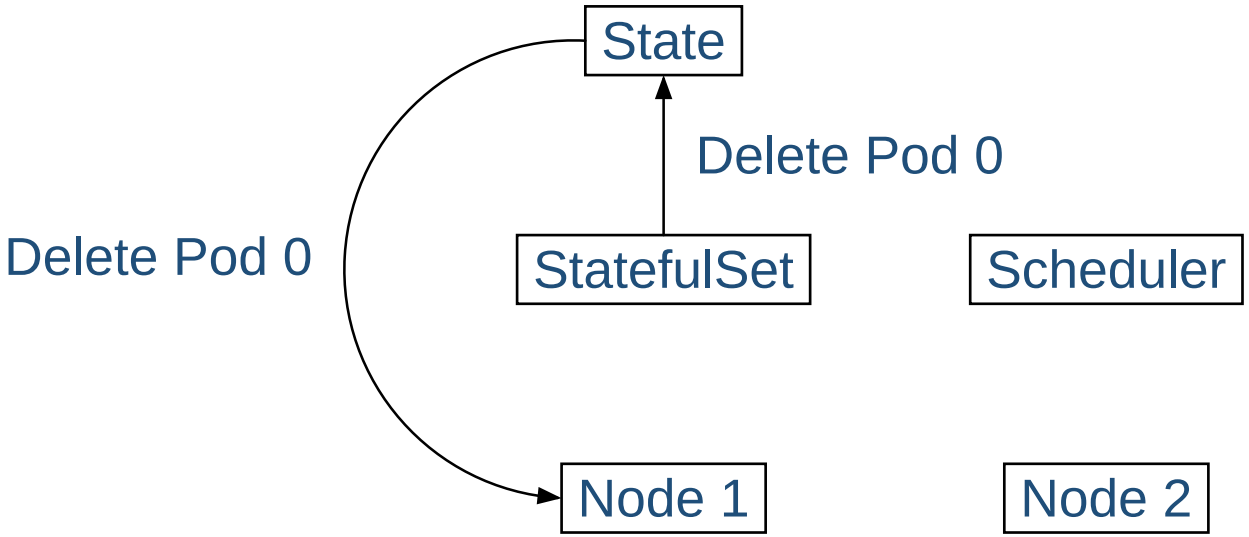
Exploring different consistency levels

- Synchronous, lock step everything
- Sessions for stale reads, writes still lock step
  - Can be durable between sessions
  - **or not…**
- Causal (see Dismerge)

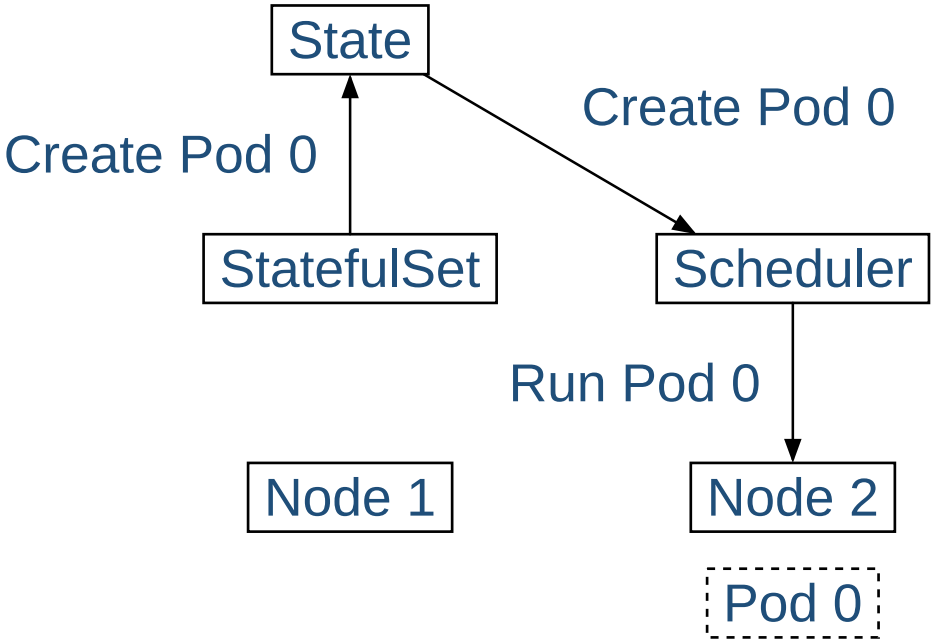
Stale reads bug



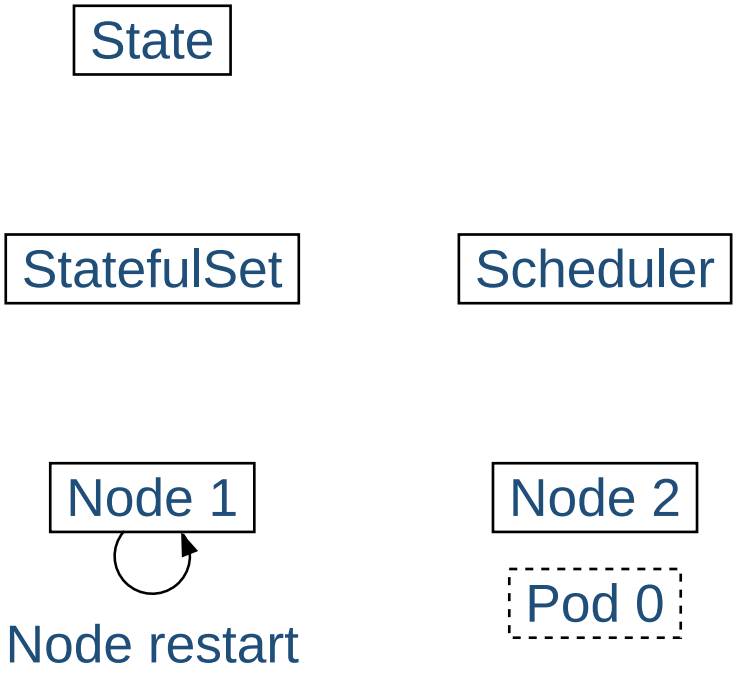
Stale reads bug



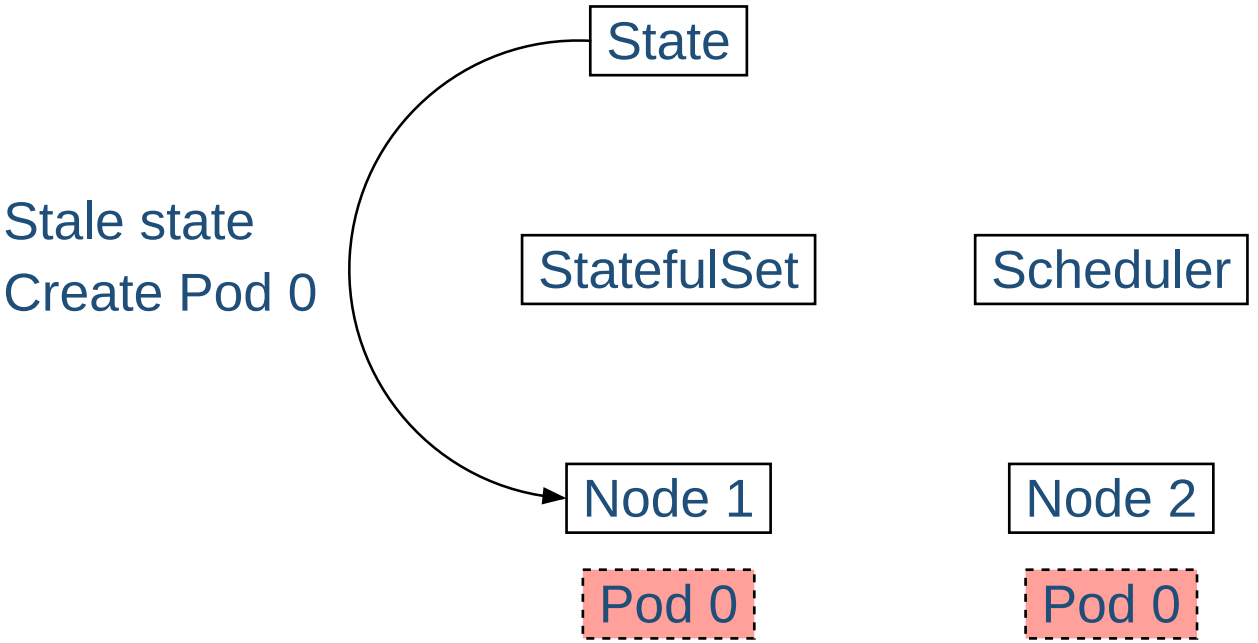
Stale reads bug



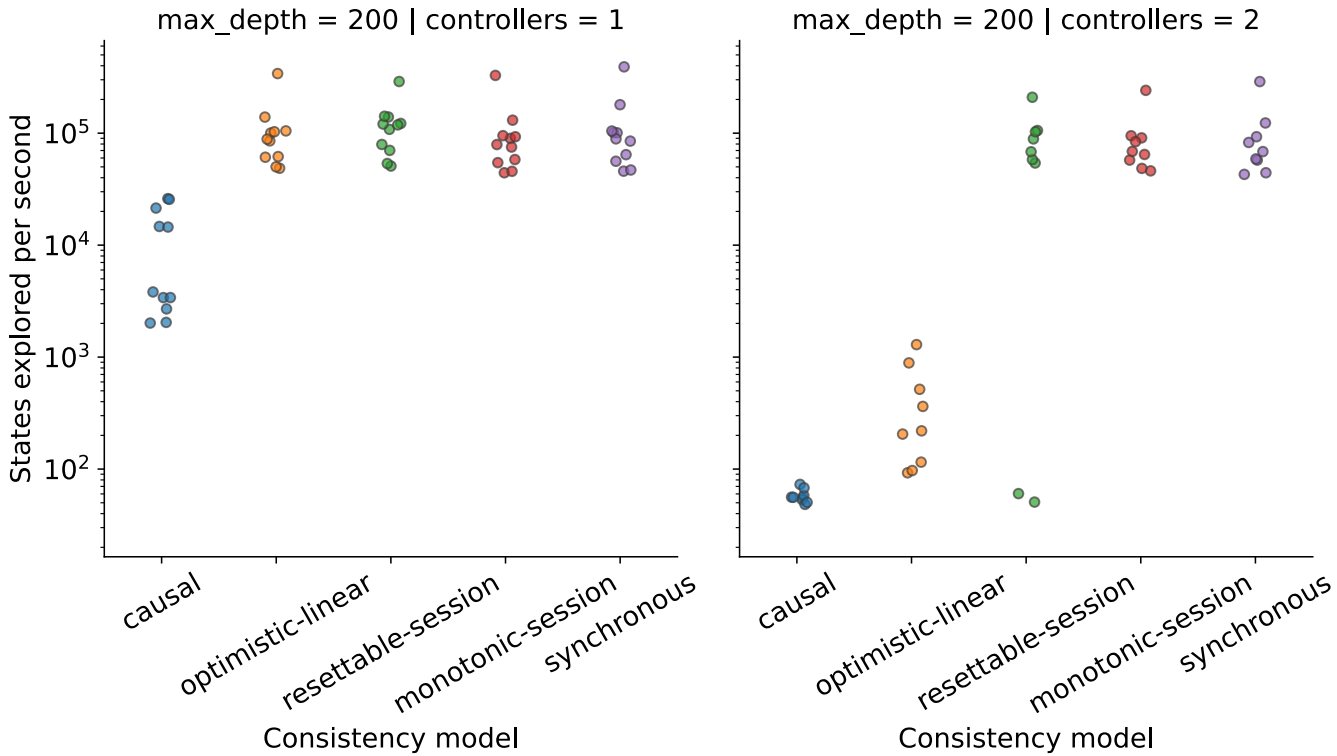
Stale reads bug



Stale reads bug

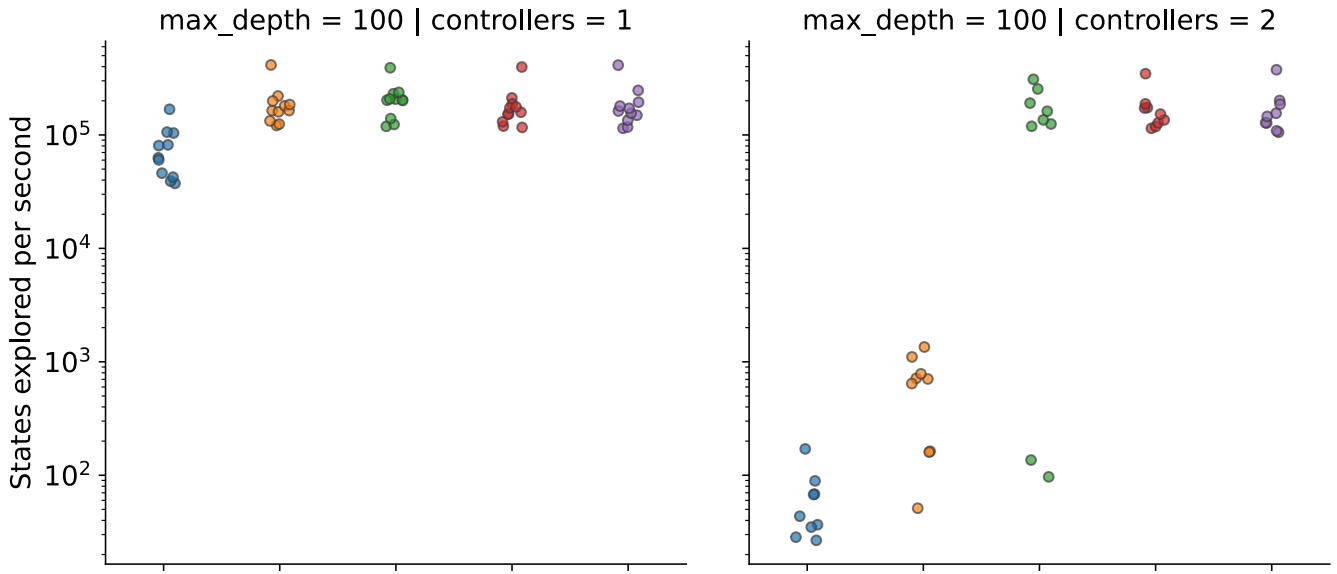


Performance: state generation



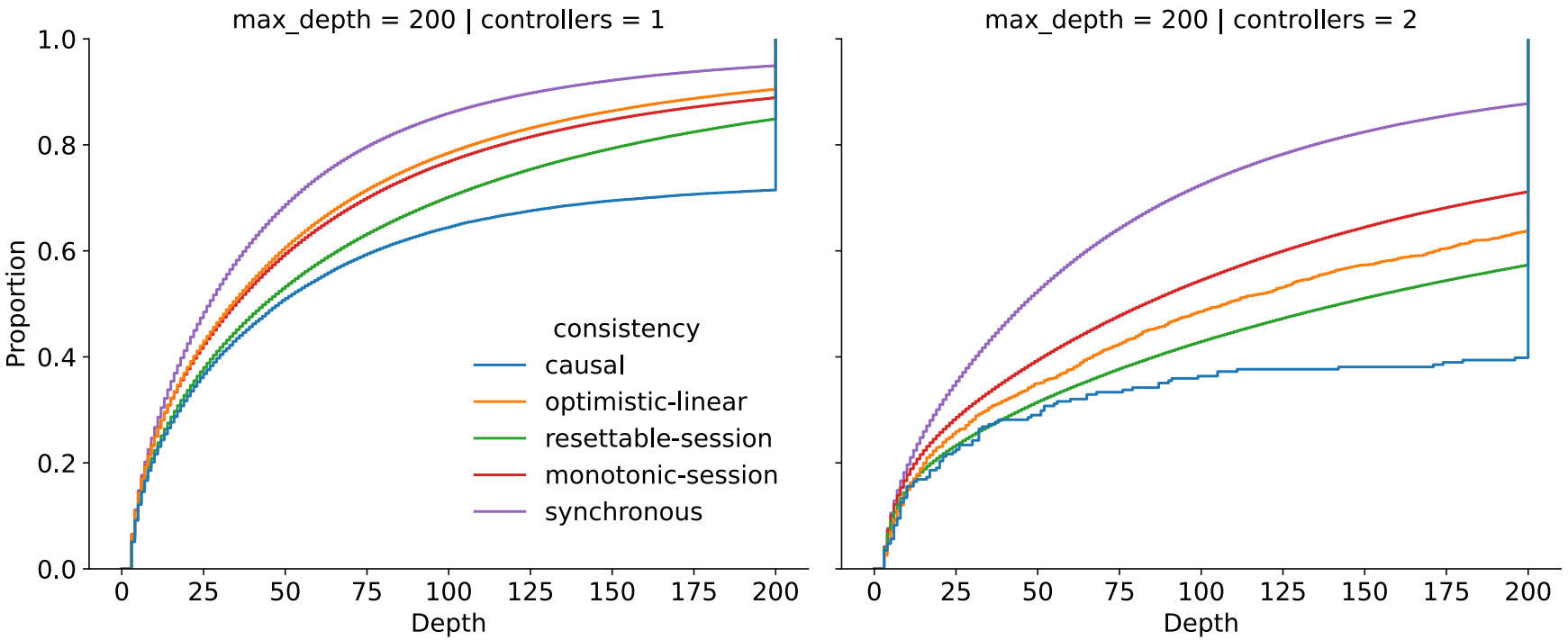
States generated per run.

Performance: state generation



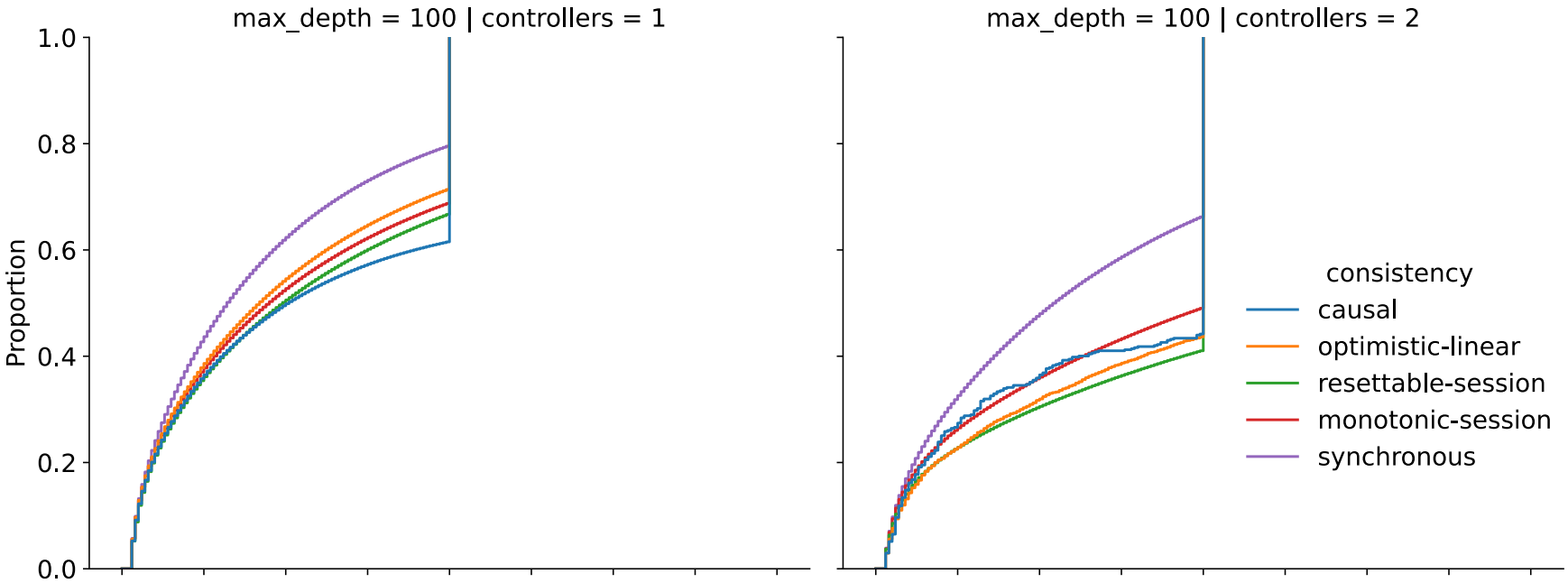
States generated per run.

Performance: depth exploration



Distribution of depths covered.

Performance: depth exploration

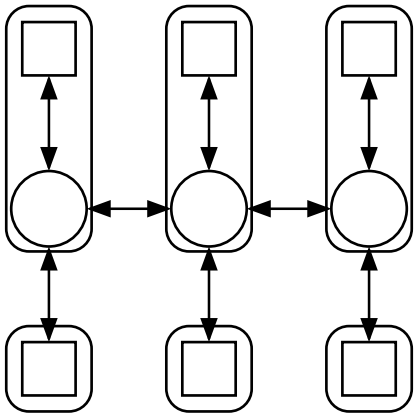


Distribution of depths covered.

Using the consistency models

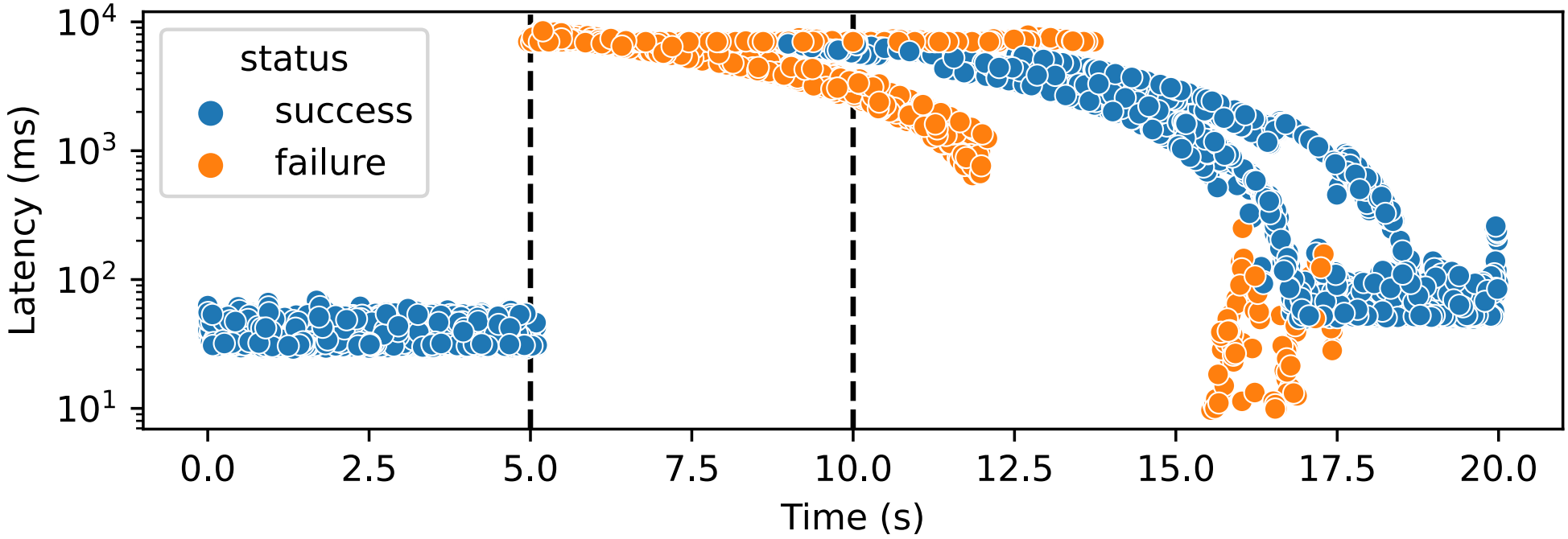
- Some controllers may need to be adapted
- Themelios provides a harness for finding broken properties
- It also provides a systematic way to check adaptations

Orchestration near the edge



Multi-site, K8s.

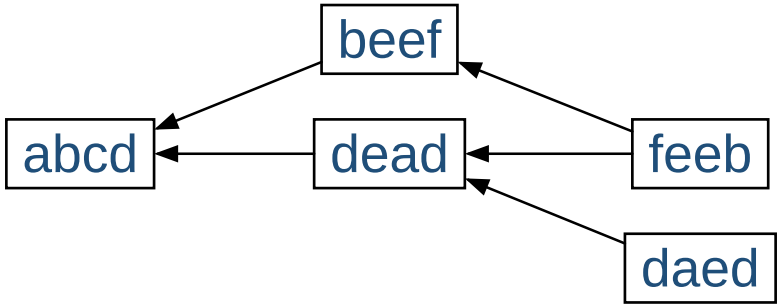
Availability of etcd



Addressing history

Etcd

Dismerge

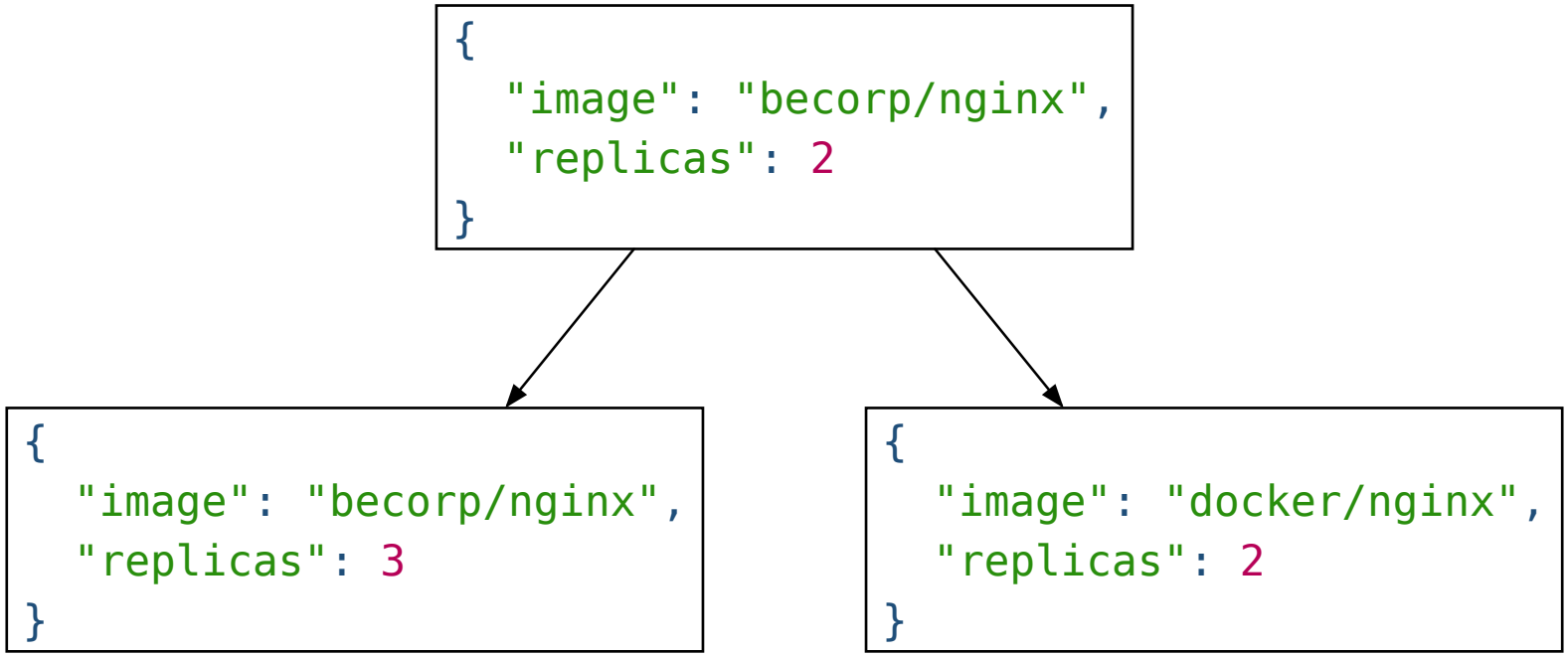


Durability

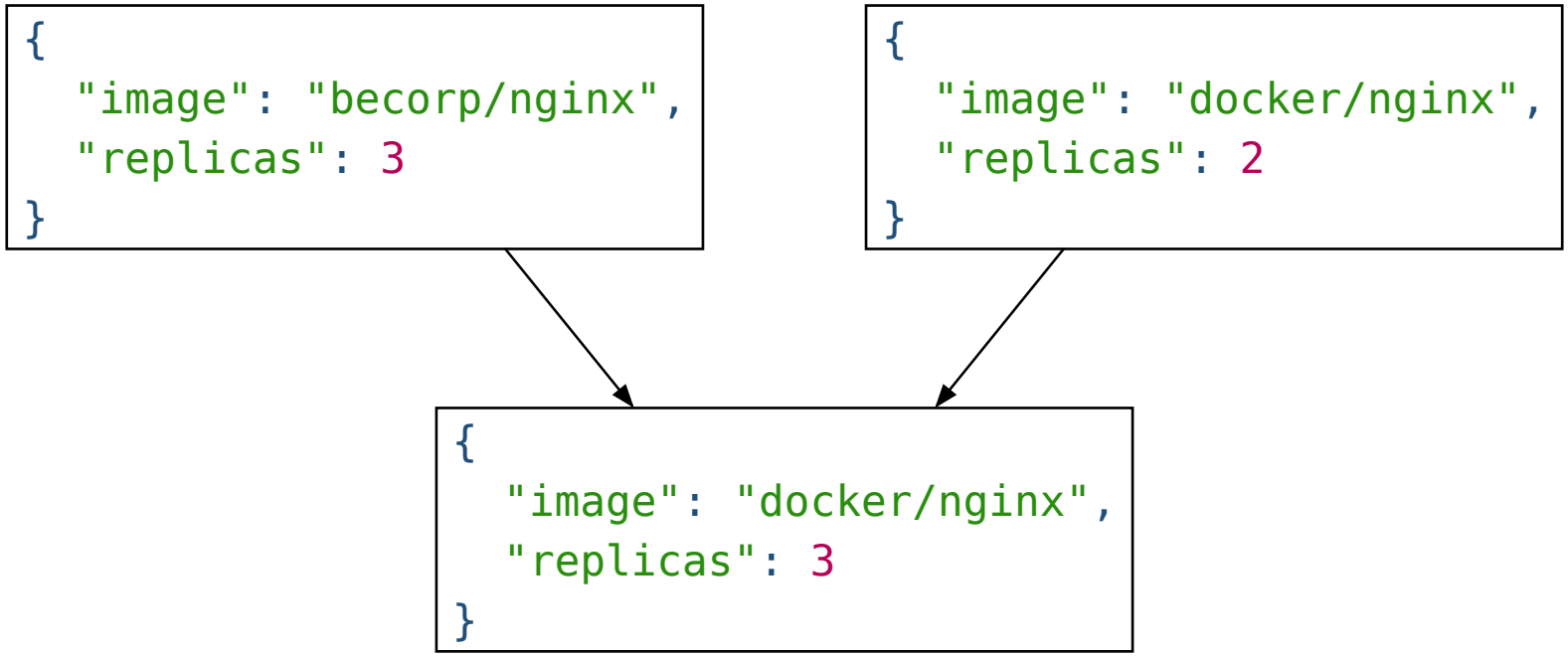
Hash	Node 1	Node 2	Node 3
abcd	✓	✓	✓
beef		✓	
dead	✓		✓

- Datastore nodes keep a note of whether peers are ahead, behind, or in sync with themselves.
- This enables clients to query replication statuses of their changes.

Merge behaviour



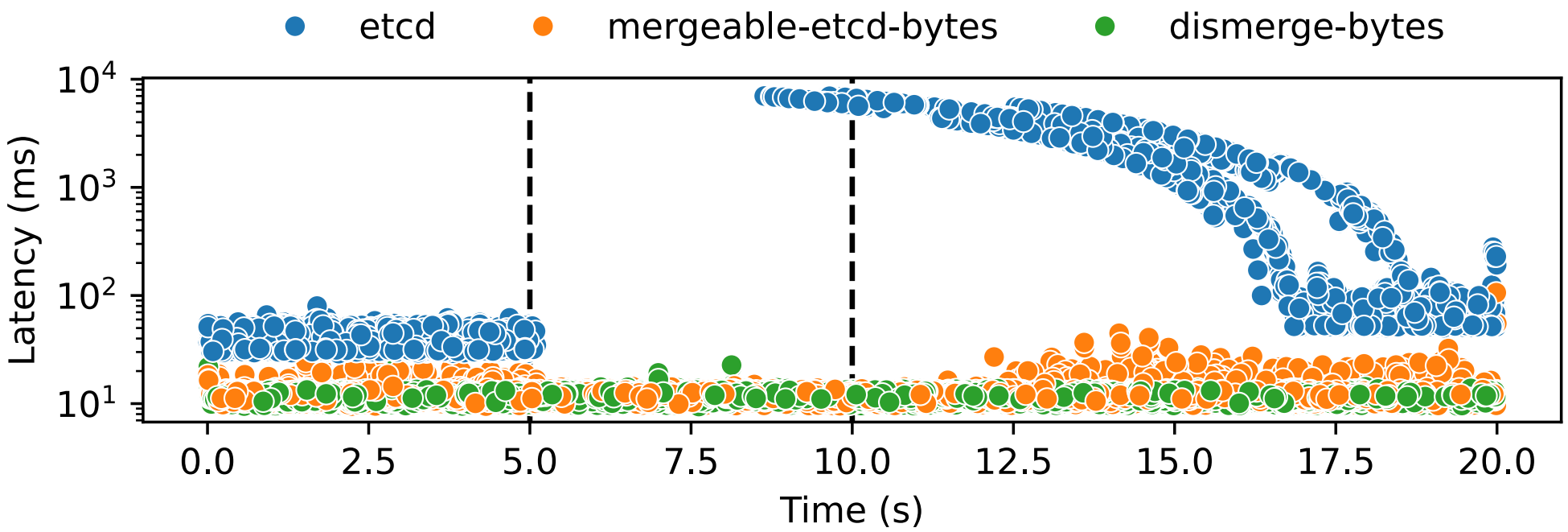
Merge behaviour



Value representation

```
#[derive(Reconcile, Hydrate, Serialize, Deserialize)]
struct Deployment {
  image:    String,
  replicas: u32,
}
```

Performance vs etcd



Future work

- Merging LSKV and Dismerge for a confidential edge datastore
- Building datastores directly into the model checking
- Exploring changes to the controllers for the different consistency models

The end

Thanks to Martin Kleppmann and Jörg Ott for being my examiners.

Using the hashes as revisions

- Primarily the revisions are opaque, and so changing them to hashes is not a large issue for clients.
- To tell if clients are up to date they can observe the latest hashes from the response header.
- There is also a new API for Dismerge that enables clients to determine which change came before another (unless concurrent).