Distributed Systems in ONOS with Atomix 3

Architecture and Implementation

Jordan Halterman

Member of Technical Staff @ ONF

Distributed Systems in ONOS with Atomix 3

Pre-Owl Architecture Review

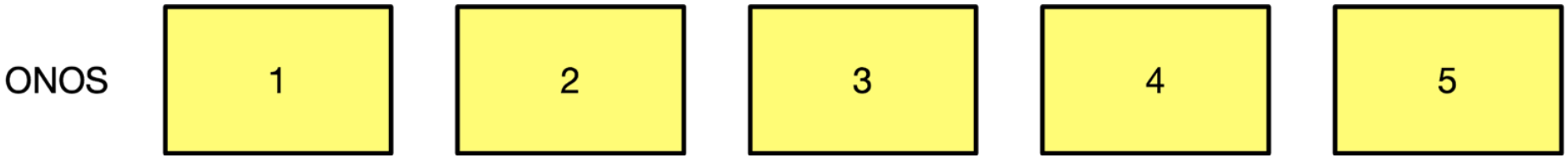
Problems and Solutions

Atomix 3 Features and Implementation

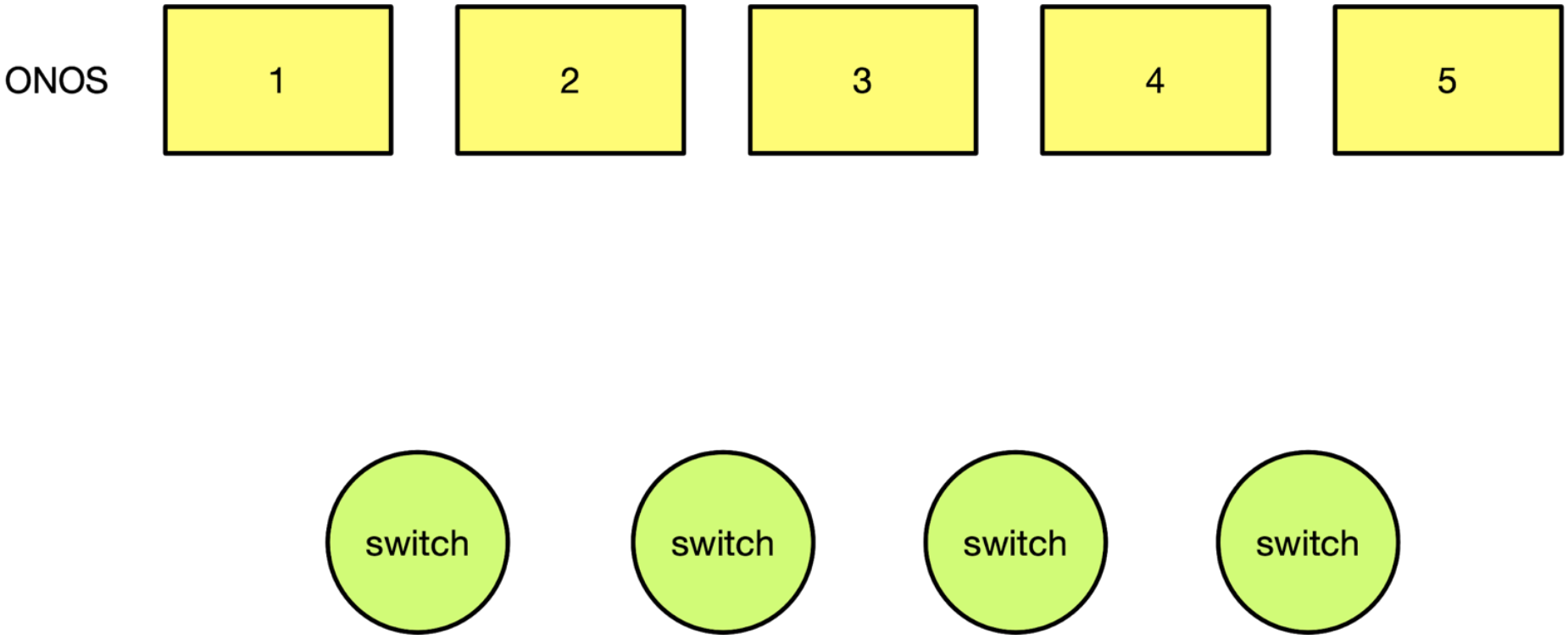
Owl Architecture

Pre-Owl Cluster Architecture

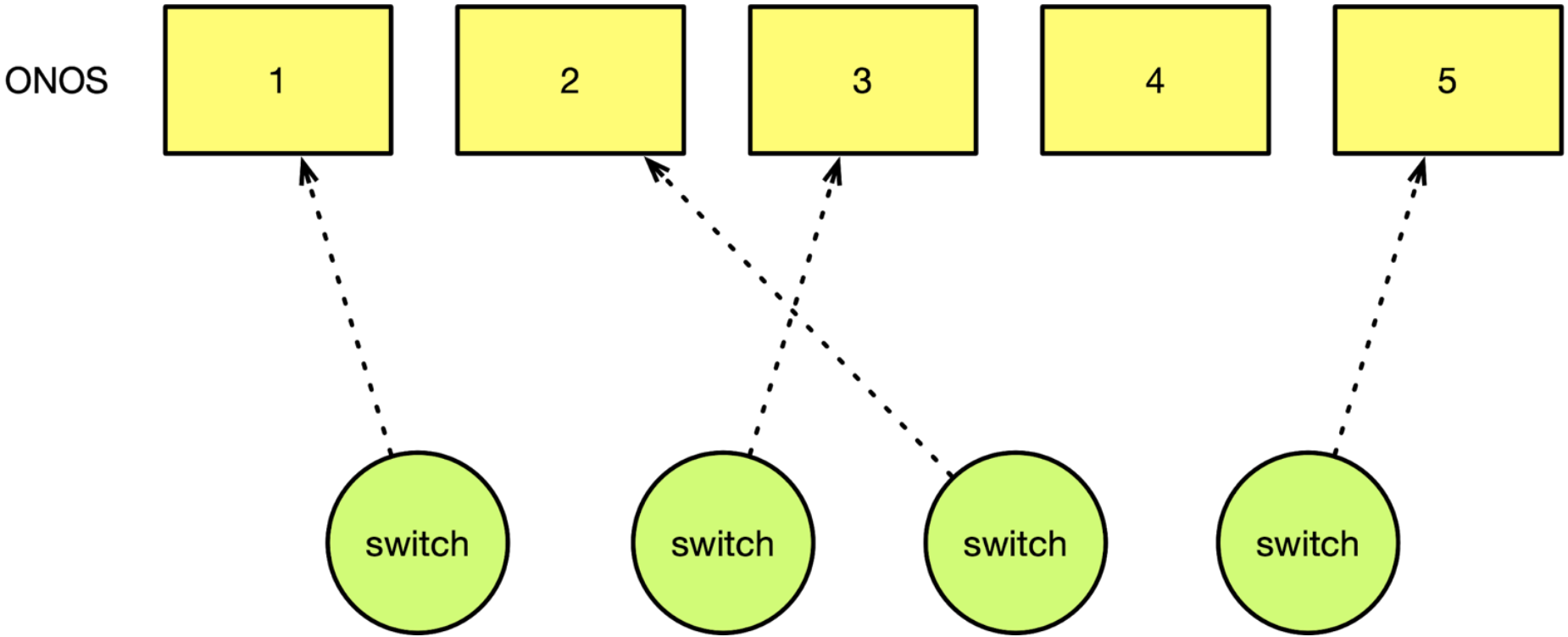
Pre-Owl Cluster



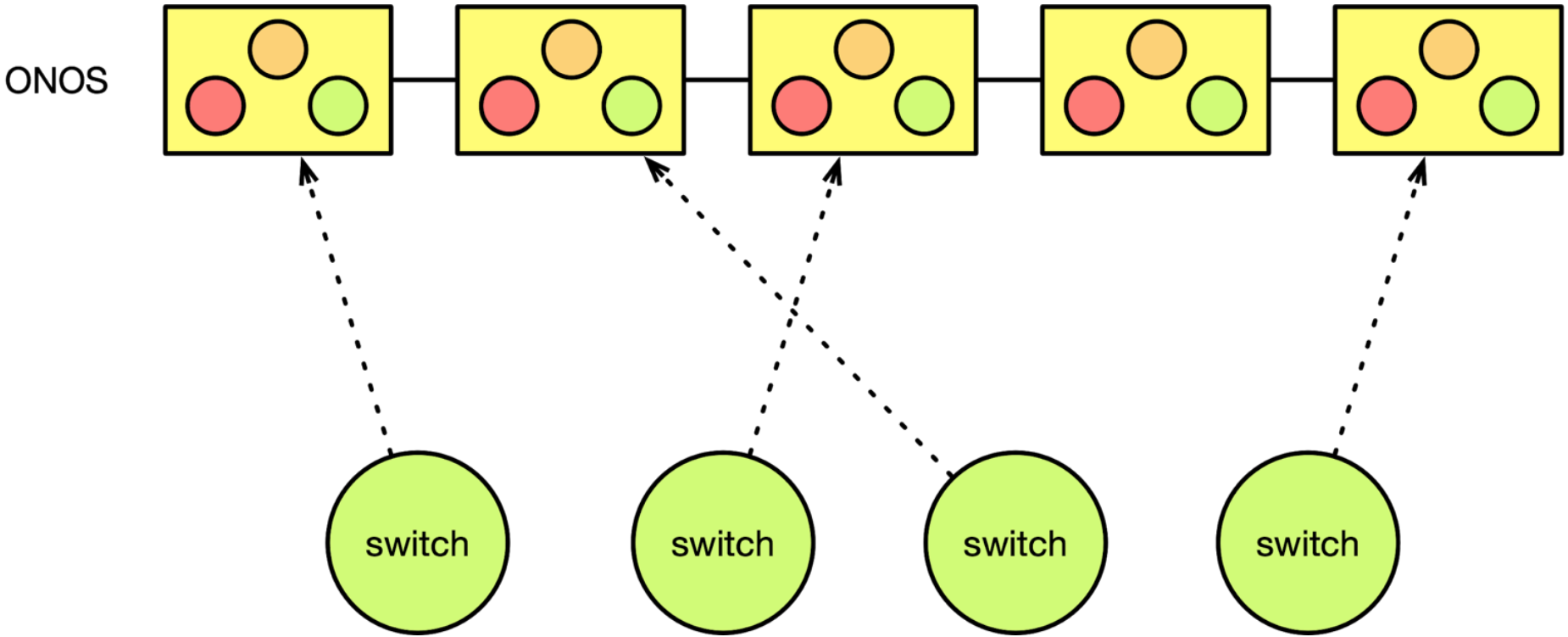
Pre-Owl Cluster



Pre-Owl Cluster



Pre-Owl Cluster



Pre-Owl Cluster

- Typically consist of an odd number of nodes
- Netty used for east-west communication
- Custom eventually consistent protocols for low latency stores
- Raft for consistent stores
- Distributed primitives provided distributed collections and concurrency primitives
- Stores built on distributed primitives or custom protocols

Pre-Owl Cluster

```
private ConsistentMultimap<IpPrefix, Route> routes;

@Activate
public void activate() {
   routes = storageService.<IpPrefix, Route>consistentMultimapBuilder()
            .withName("routes")
            .withSerializer(Serializer.using(KryoNamespaces.API))
            .build();
}

public void addRoute(IpPrefix prefix, Route route) {
   routes.put(prefix, route);
}
```

Problems?

Embedded Raft

- Imposes strict cluster configuration requirements with little benefit
- Cluster configuration must be explicitly defined
- A quorum of nodes must always be maintained
- Load on Raft partitions affects southbound protocols
- Difficult and expensive to reconfigure/scale the cluster
- 1/n primitive operations still require 2 RTT

Distributed Primitives

- Distributed primitives have proven valuable
- But mostly limited to a single protocol/consistency model
- Need for low-latency in-memory primitives has been expressed

Distributed Systems Code

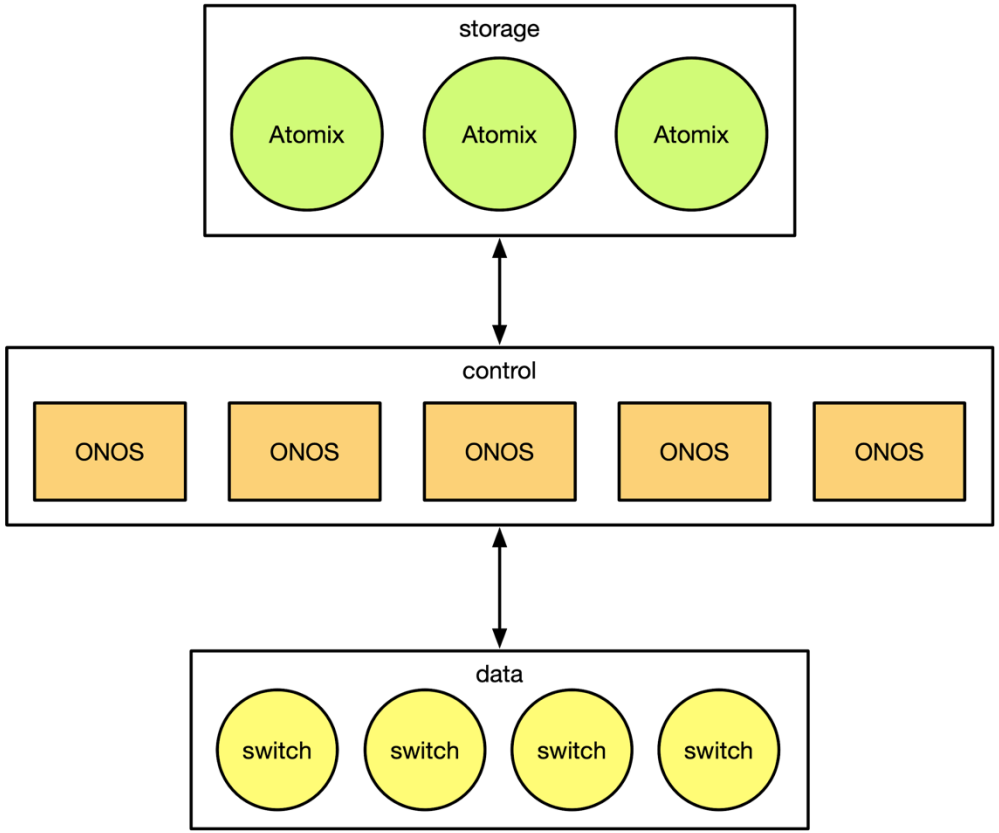
- Lots of useful distributed systems code in ONOS
  - Intra-cluster communication
  - Cluster management
  - Distributed primitives
  - Partitioning/sharding/scaling
  - Eventually consistent protocols
- But it's being maintained by ONOS only for ONOS
- ONOS could benefit from distributing it more widely

Solution?

Solution

- Re-architect ONOS cluster to separate storage from control
- Scale and upgrade controller independently of storage layer
- Build an external framework for cluster management
- Generalize ONOS distributed primitives in external framework
- More users and contributors to ONOS' distributed core
- Support multiple protocols for distributed primitives
- Support future architecture changes

Solution


storage
Atomix
Atomix
Atomix
control
ONOS
ONOS
ONOS
ONOS
ONOS
data
switch
switch
switch
switch

Solution

- Work began in ONOS 1.12
- Four stages:
  - Migrate cluster management/communication
  - Generalize distributed systems protocols
  - Migrate and adapt distributed primitives
  - Re-architect the ONOS controller

Atomix 3

What is Atomix?

- Reactive Java framework for building scalable, fault-tolerant distributed systems
- Provides primitives for every layer of distributed applications
  - Intra-cluster communication
  - Cluster membership
  - Replication
  - Synchronization
  - Fault-tolerance

What is Atomix?

- Atomix is unopinionated
- Does not prefer a specific cluster architecture
- Instead provides options for constructing different architectures
  - Can be used as a library, as a service, or both
  - Synchronous or asynchronous programming models
  - Node roles defined in configuration
  - Reliable and unreliable communication abstractions
  - Direct and pub-sub messaging
  - Protocol agnostic data structures and concurrency control

Programmatic API

```
Atomix atomix = Atomix.builder()
    .withMemberId("member-1")
    .withHost("192.168.10.1")
    .build();

atomix.start().join();
```

Features

- Cluster membership
- Cluster communication
- Distributed primitives

Cluster Membership

ClusterMembershipService

```
ClusterMembershipService membershipService = atomix.getMembershipService();

membershipService.getMembers().forEach(member -> {
   atomix.getCommunicationService().send("hello", "Hello!", member.id());
});
```

ClusterMembershipService

```
ClusterMembershipService membershipService = atomix.getMembershipService();

membershipService.addListener(event -> {
    switch (event.type()) {
        case MEMBER_ADDED:
            memberAdded(event.subject());
            break;
        case MEMBER_REMOVED:
            memberRemoved(event.subject());
            break;
    }
});
```

ClusterMembershipService

- Join cluster
- Locate cluster members
- Detect membership failures

ClusterMembershipService

```
ClusterMembershipService membershipService = atomix.getMembershipService();

membershipService.addListener(event -> {
    if (event.type() == ClusterMembershipEvent.Type.MEMBER_ADDED) {
        Member member = event.subject();
        if (member.properties().getProperty("type", "atomix").equals("onos")) {
            // This is an ONOS node!
        }
    }
});
```

Cluster Communication

Cluster Communication

- Netty 4.x
- Asynchronous
- Request-reply
- Publish-subscribe
- Reliable & unreliable
- Multicast

ClusterCommunicationService

- For location aware protocols
- Topic-based
- Request-reply
- Unicast
- Multicast
- Broadcast
- Over TCP or UDP

ClusterEventService

- For service-oriented architectures
- Topic-based
- Request-reply
- Unicast
- Multicast
- Broadcast
- Load balanced over TCP

ClusterCommunicationService

```
ClusterCommunicationService communicationService = atomix.getCommunicationService();

communicationService.subscribe("hello", this::sayHello);
```

ClusterCommunicationService

```
ClusterCommunicationService communicationService = atomix.getCommunicationService();

MemberId memberId = MemberId.from("atomix-2");
communicationService.send("hello", "Hello!", memberId)
    .thenAccept(response -> {
        LOGGER.info("{} said {}", memberId, response);
    });
```

Distributed Primitives

Distributed Primitives

- Cluster-wide replicated data structures and synchronization primitives
- Synchronous and asynchronous implementations
- Map
- Set
- Tree
- Lock
- Semaphore
- Leader election
- etc

Distributed Primitives

```
DistributedMap<String, String> map = atomix.<String, String>mapBuilder("my-map")
    ...
    .withCacheEnabled()
    .build();

map.put("onos", "awesome");
```

Distributed Primitives

```
DistributedSet<String> set = atomix.<String>setBuilder("my-set")
    ...
    .build();

if (set.remove("foo")) {
  set.add("bar");
}
```

Distributed Primitives

```
AtomicCounter counter = atomix.atomicCounterBuilder("my-counter")
    ...
    .build();

long value = counter.incrementAndGet();
counter.compareAndSet(value, value + 1);
```

Distributed Primitives

```
// Create a distributed lock primitive.
DistributedLock lock = atomix.lockBuilder("my-lock")
    ...
    .build();

// Acquire the lock then do some work and release it.
lock.lock();
try {
  doWork();
} finally {
  lock.unlock();
}
```

Distributed Primitives

```
// Create a leadership election.
LeaderElection<MemberId> election = atomix.<MemberId>leaderElectionBuilder("my-election")
    ...
    .build();

// Get the local member identifier.
MemberId localMemberId = atomix.getMembershipService().getLocalMember().id();

// Run the local member ID for leadership.
Leadership<MemberId> leadership = election.run(localMemberId);

// Send a message to the current leader to do some work.
atomix.getCommunicationService().send("do-work", new Work(), leadership.leader().id())
    .whenCompleteAsync((response, error) -> {
      if (error == null) {
        LOGGER.info("Work complete!");
      }
    });
```

Distributed Primitives

```
LeaderElection<MemberId> election = atomix.<MemberId>leaderElectionBuilder("my-election")
    ...
    .build();

election.addListener(event -> {
  MemberId newLeaderId = event.newLeadership().leader().id();
  LOGGER.info("A leadership change event occurred. New leader: {}", newLeaderId);
});
```

Distributed Primitives

```
DistributedMap<String, String> map = atomix.<String, String>mapBuilder("my-
map")
    ...
    .withCacheEnabled()
    .build();

map.put("onos", "awesome");

AsyncDistributedMap<String, String> asyncMap = map.async();

asyncMap.put("onos", "awesome").thenRun(() -> LOGGER.info("Write complete"));
```

So what?

So what?

Configuration

Deployment

Configuration

Configuration API

```
cluster {
  node {
    id: atomix-1
    host: 192.168.20.1
  }

  multicast.enabled: true
  discovery {
    type: multicast
    broadcastInterval: 1s
  }
}

partitionGroups.raft {
  type: raft
  partitions: 3
  partitionSize: 3
  storage.level: mapped
  members: [atomix-1, atomix-2, atomix-3]
}
```

Cluster Management

Cluster Management

```
ClusterMembershipService membershipService = atomix.getMembershipService();

membershipService.getMembers().forEach(member -> {
   atomix.getCommunicationService().send("hello", "Hello!", member.id());
});
```

Cluster Management

```
ClusterMembershipService membershipService = atomix.getMembershipService();

membershipService.addListener(event -> {
    switch (event.type()) {
        case MEMBER_ADDED:
            memberAdded(event.subject());
            break;
        case MEMBER_REMOVED:
            memberRemoved(event.subject());
            break;
    }
});
```

Cluster Management

Node discovery

Cluster membership

Node Discovery

- Form new cluster
- Locate existing cluster
- Pluggable abstraction
- Multiple implementations

Node Discovery

```
cluster {
 node {
   id: member-1
   host: 192.168.10.1
 }

 discovery {
   type: bootstrap
   nodes.1 {
     id: member-1
     host: 192.168.10.1
   }
   nodes.2 {
     id: member-2
     host: 192.168.10.2
   }
   nodes.3 {
     id: member-3
     host: 192.168.10.3
   }
 }
}
```

Node Discovery

```
cluster {
 node {
   id: member-1
   host: 192.168.10.1
 }

 multicast.enabled: true

 discovery {
   type: multicast
   broadcastInterval: 1s
 }
}
```

Node Discovery

```
cluster {
 node {
   id: member-1
   host: 192.168.10.1
 }

 discovery {
   type: dns
   service: onos
 }
}
```

Cluster Membership

- Join cluster
- Find new members
- Detect failures
- Pluggable abstraction
- Multiple implementations

Cluster Membership

```
cluster.protocol {
   type: heartbeat
   heartbeatInterval: 250ms
   failureThreshold: 12
}
```

Cluster Membership

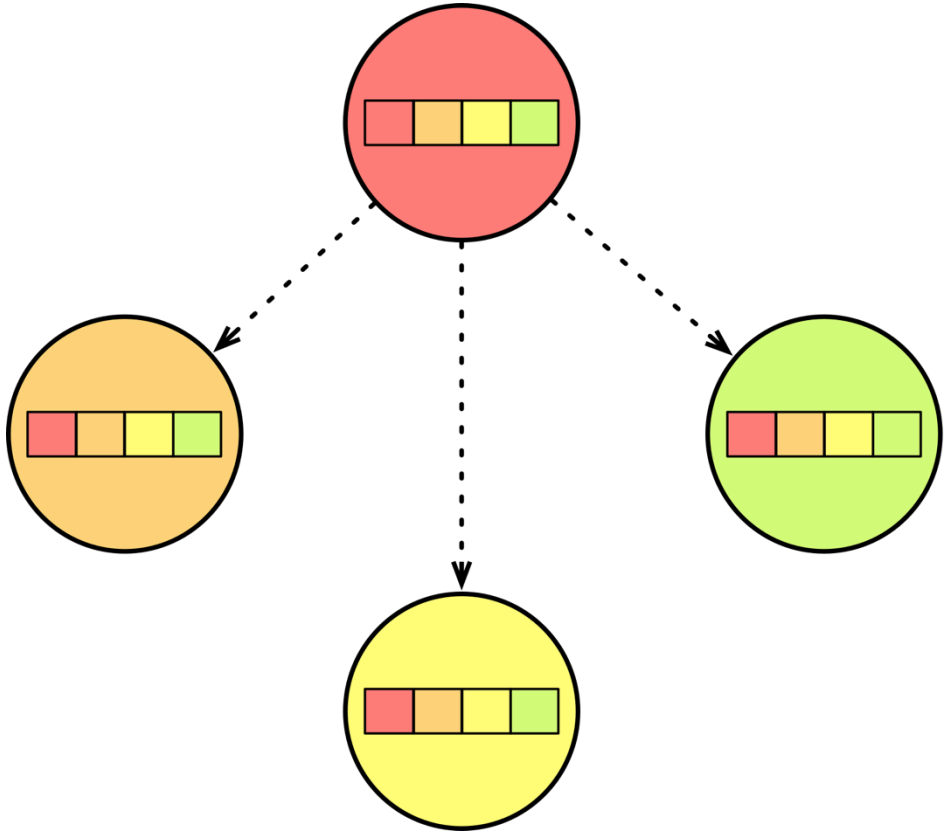
```
Atomix atomix = Atomix.builder()
    .withMemberId("member-1")
    .withHost("192.168.10.1")
    .withMulticastEnabled()
    .withMembershipProvider(MulticastDiscoveryProvider.builder()
        .withBroadcastInterval(Duration.ofSeconds(1))
        .build())
    .withMembershipProtocol(HeartbeatMembershipProtocol.builder()
        .withHeartbeatInterval(Duration.ofMillis(250))
        .build())
    .build();

atomix.start().join();
```

Heartbeat Protocol

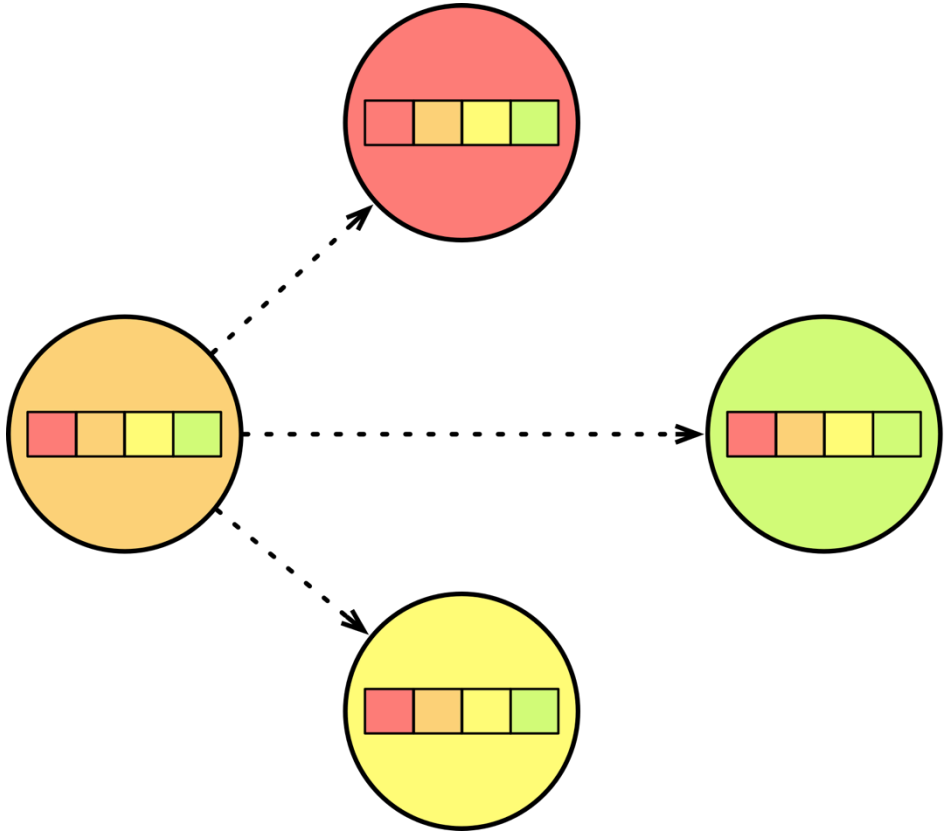
Heartbeat Protocol

- Send constant heartbeats to all peers
- If a heartbeat is not received, mark peer dead

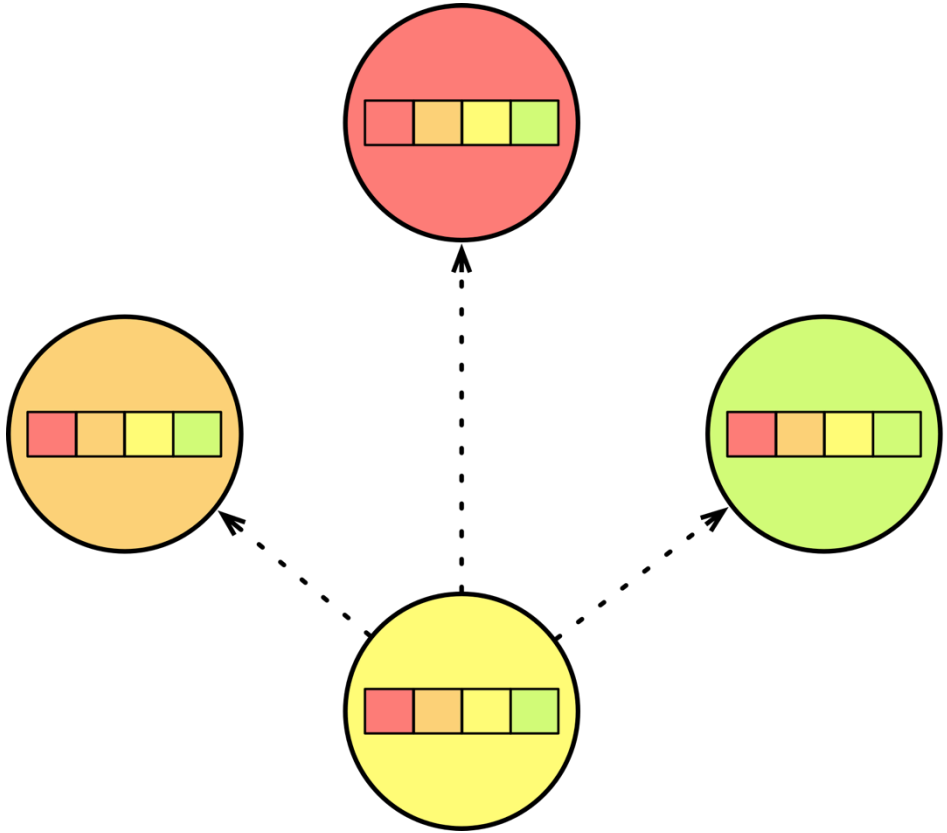
Heartbeat Protocol



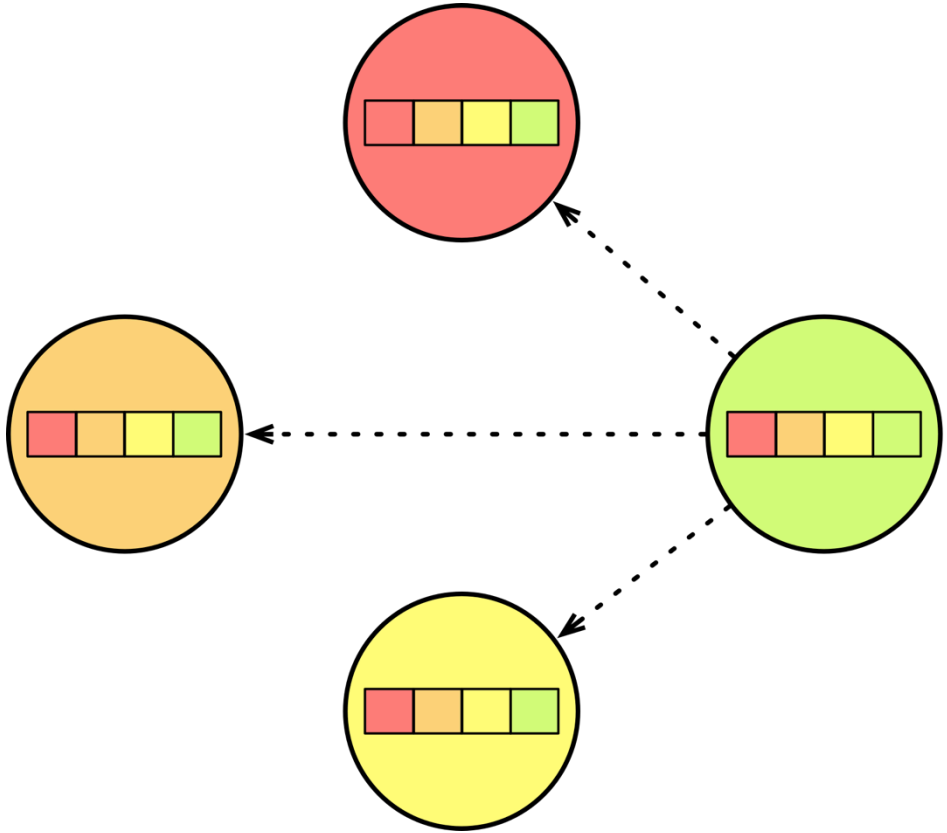
Heartbeat Protocol



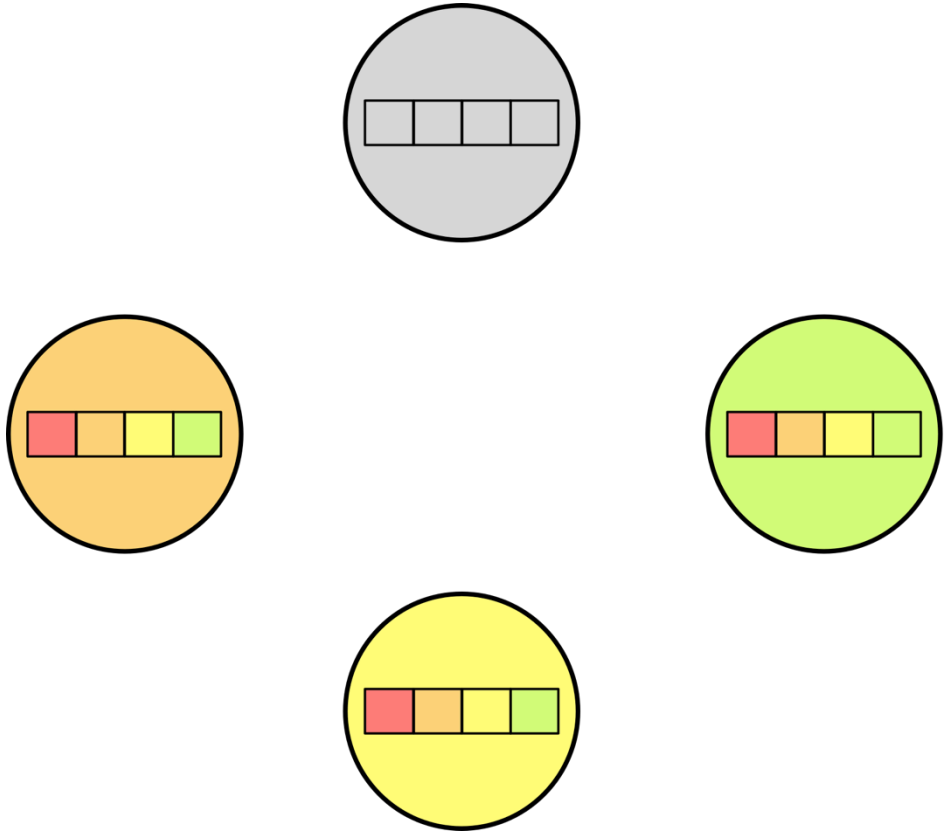
Heartbeat Protocol



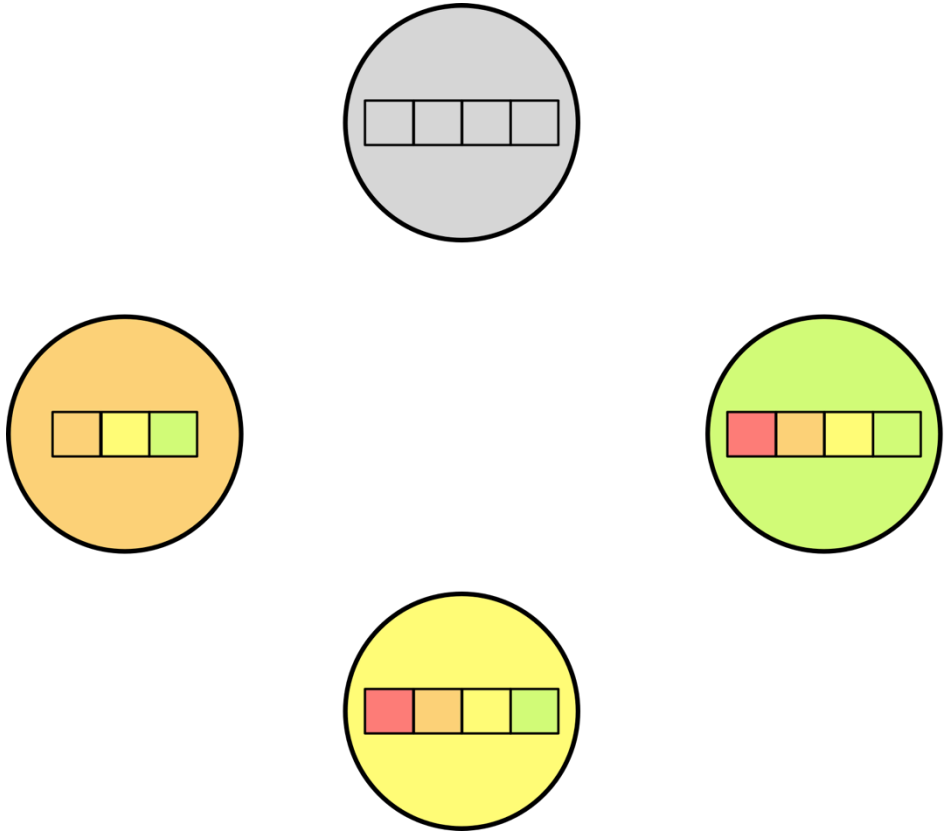
Heartbeat Protocol



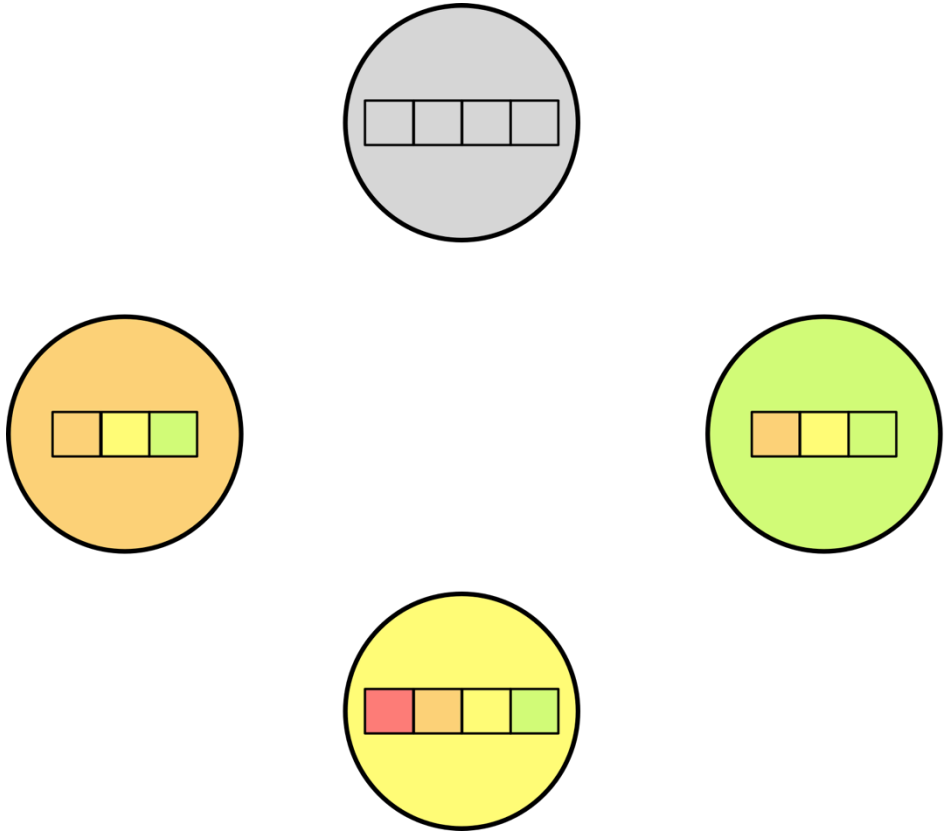
Heartbeat Protocol



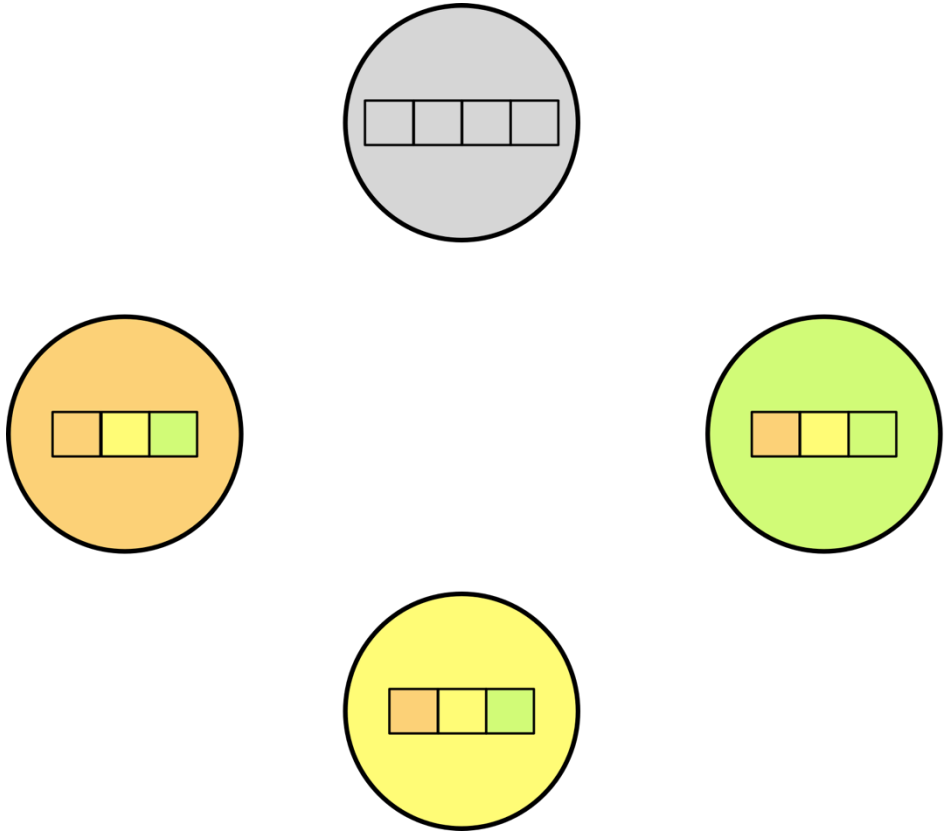
Heartbeat Protocol



Heartbeat Protocol



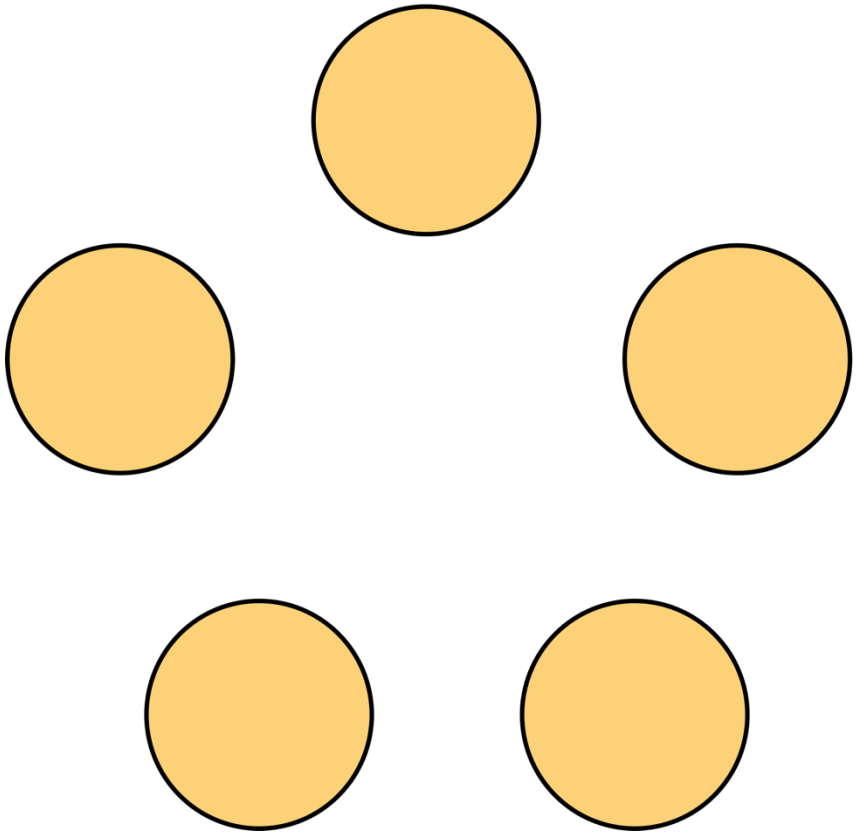
Heartbeat Protocol



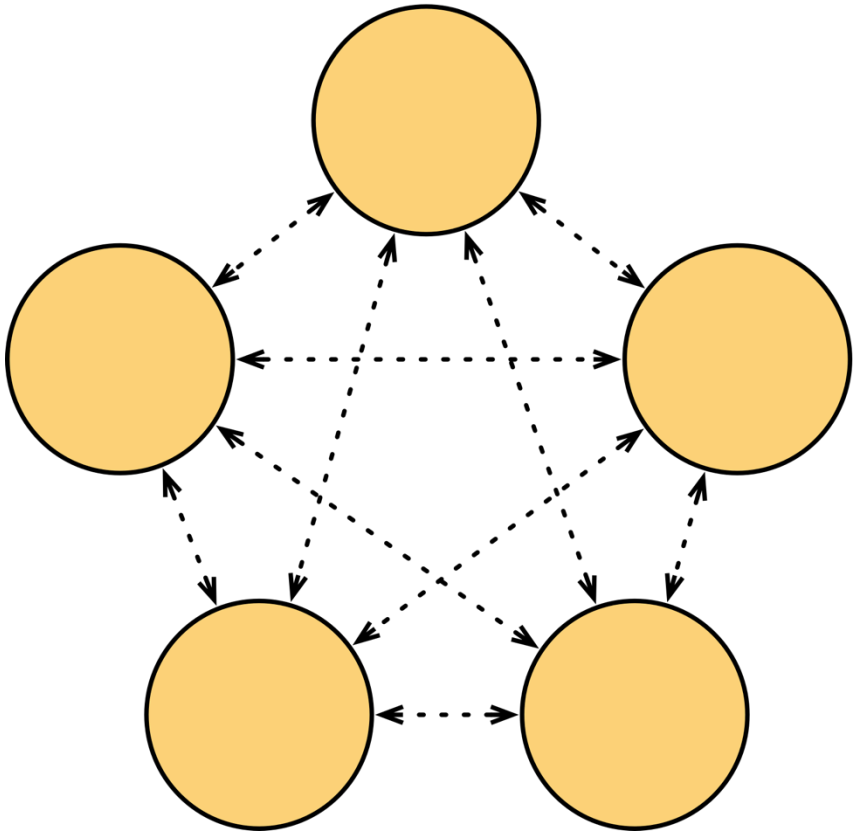
Heartbeat Protocol

```
cluster.protocol {
   type: heartbeat
   heartbeatInterval: 250ms
   failureThreshold: 12
}
```

Heartbeat Protocol



Heartbeat Protocol



Heartbeat Protocol

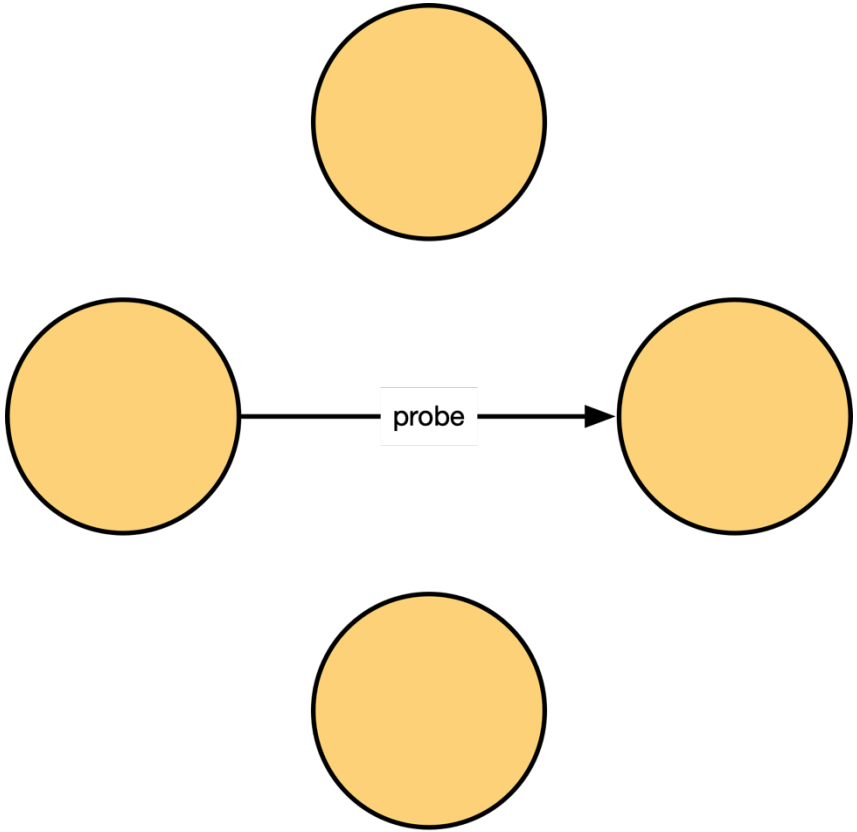
- Does not scale well
- Exponential growth in network traffic
- Does not handle simple network partitions
- More frequent false positives
- But can detect failures more quickly

SWIM Protocol

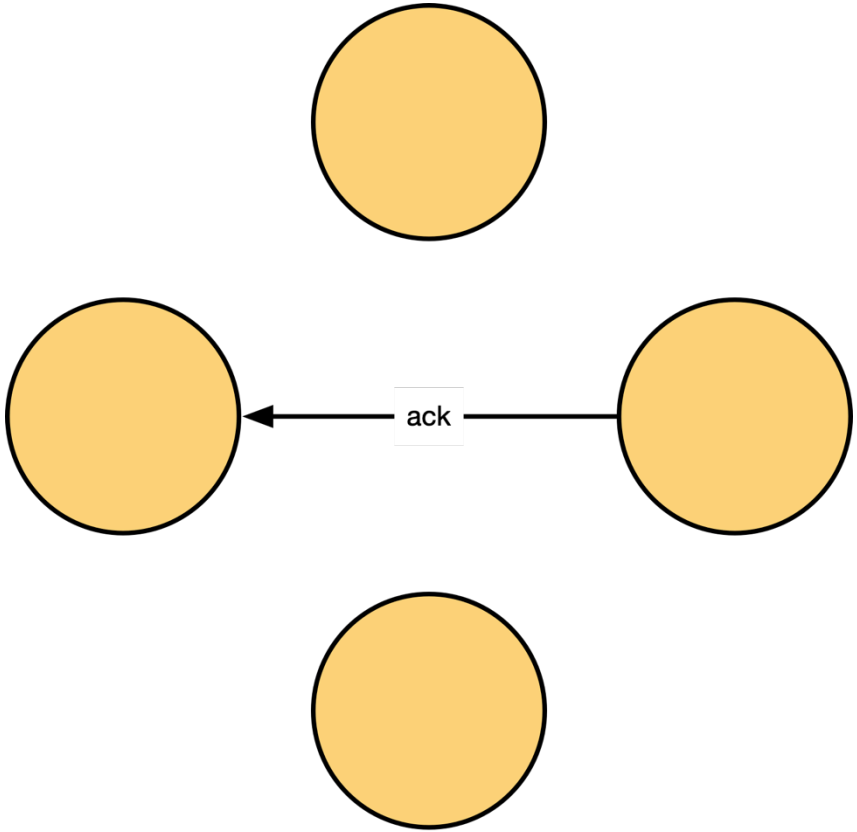
SWIM Protocol

- Reduce network load
- Improve scalability
- Reduce false positives

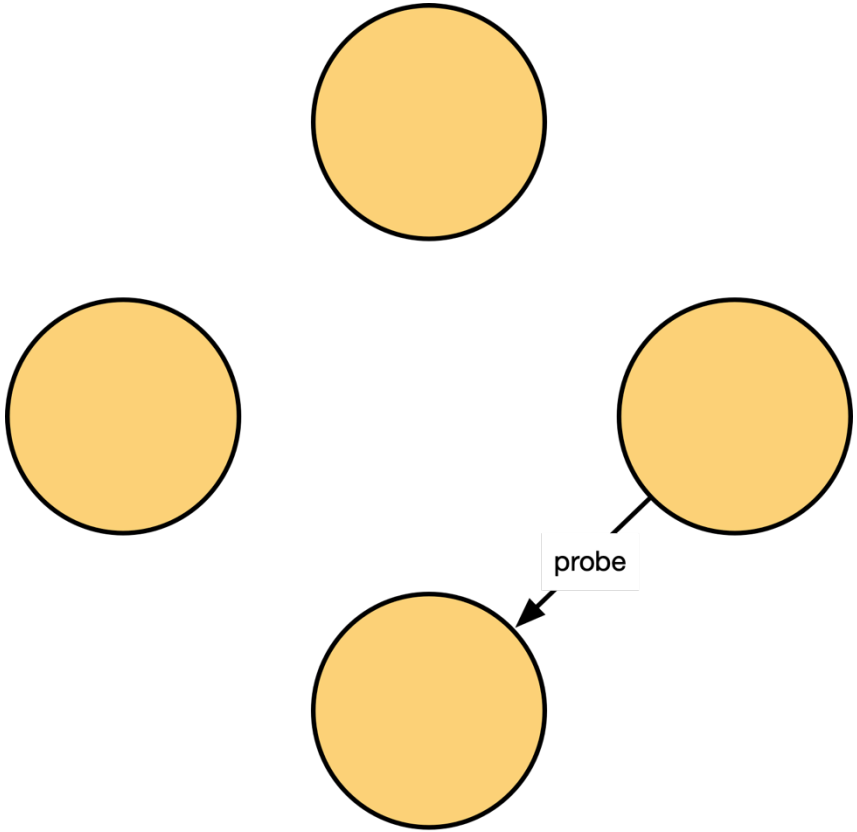
SWIM Protocol



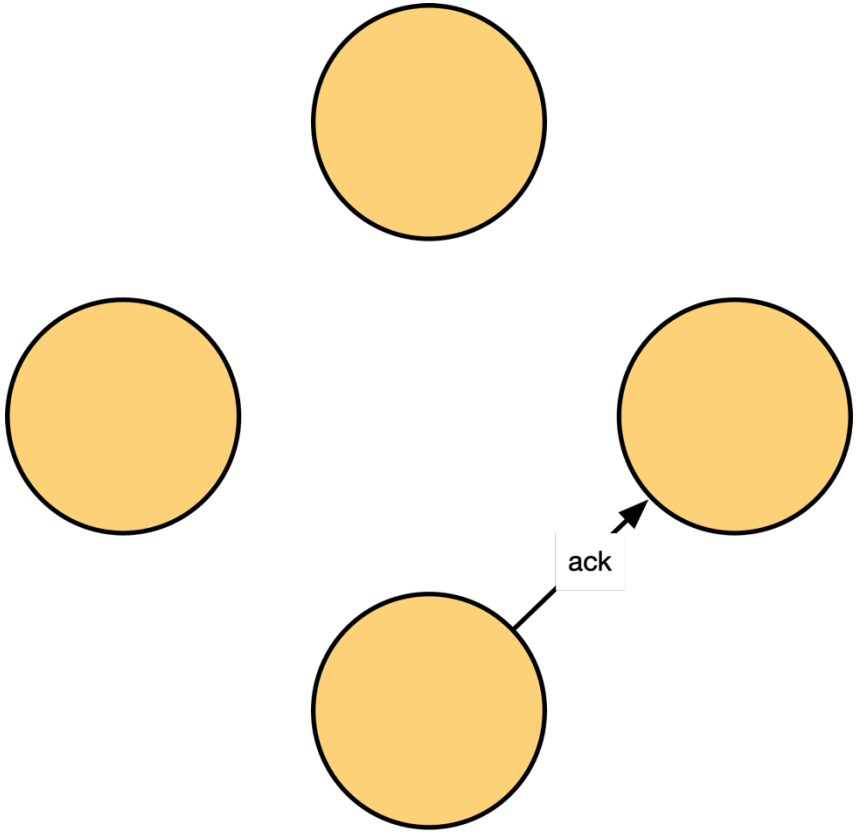
SWIM Protocol



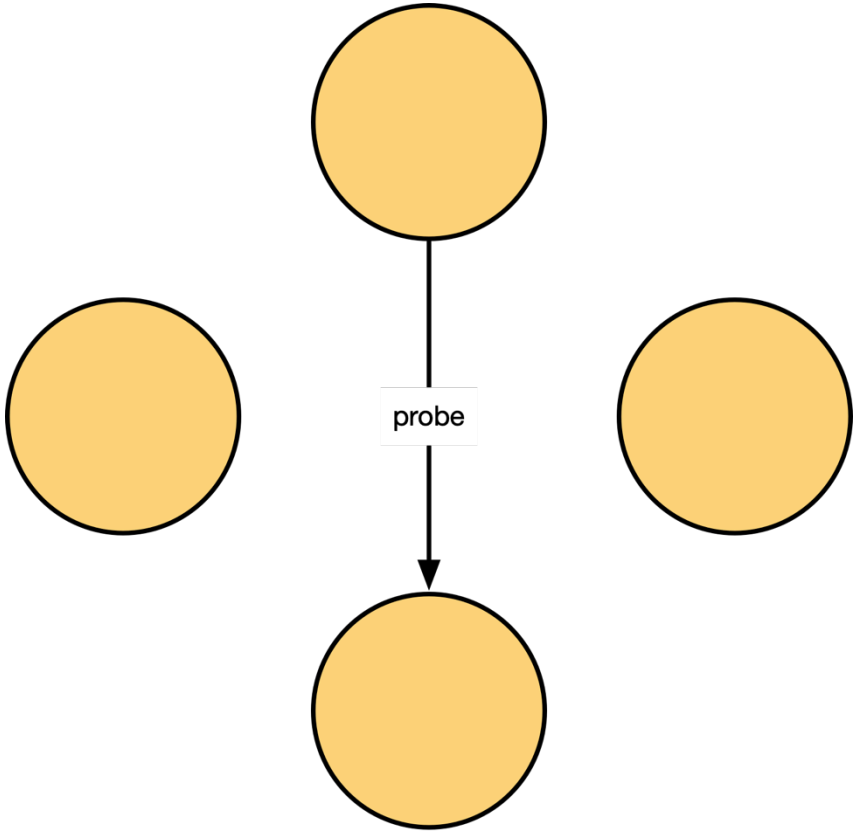
SWIM Protocol



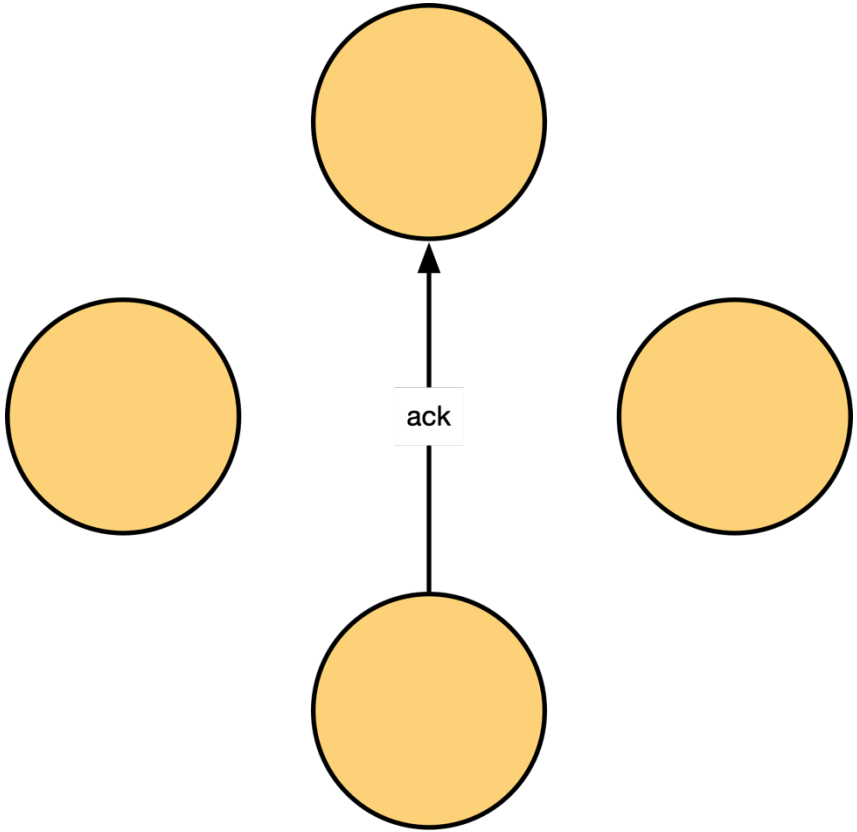
SWIM Protocol



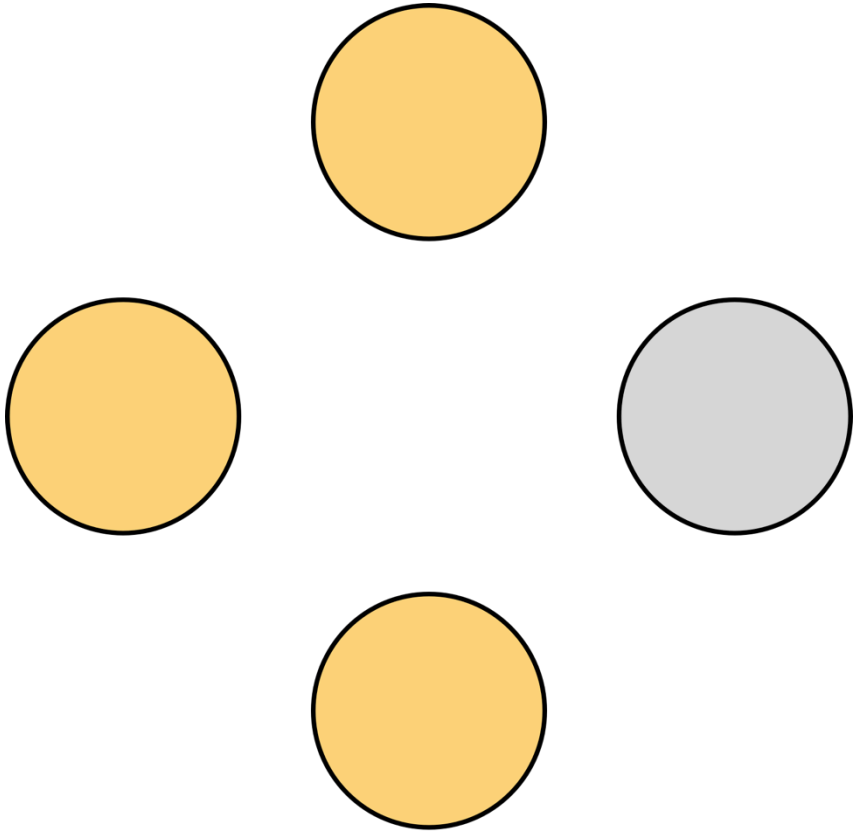
SWIM Protocol



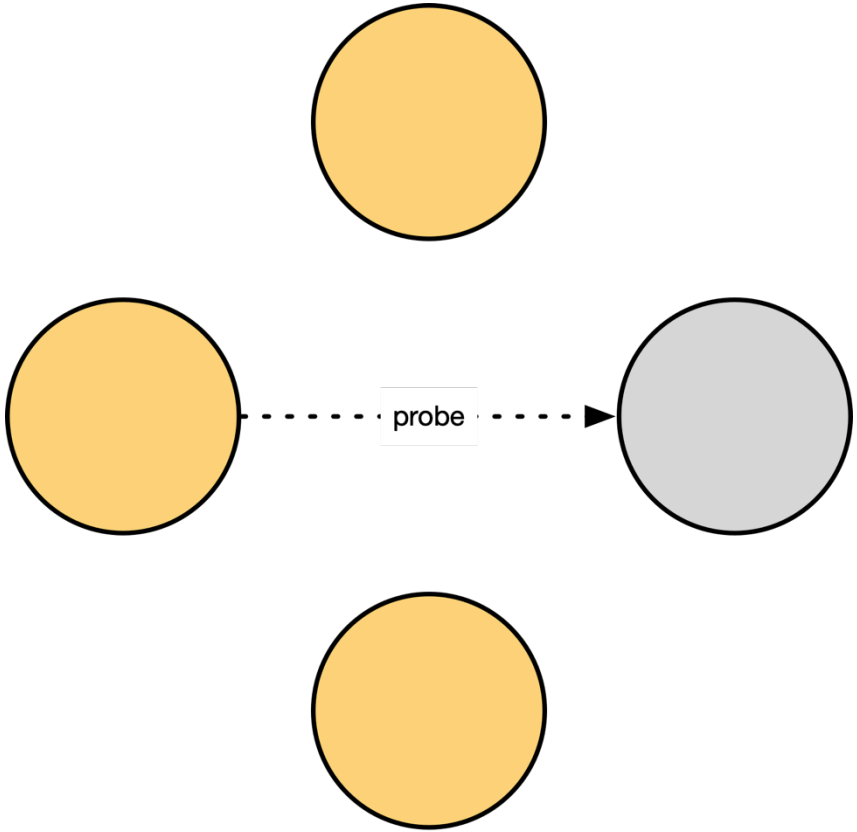
SWIM Protocol



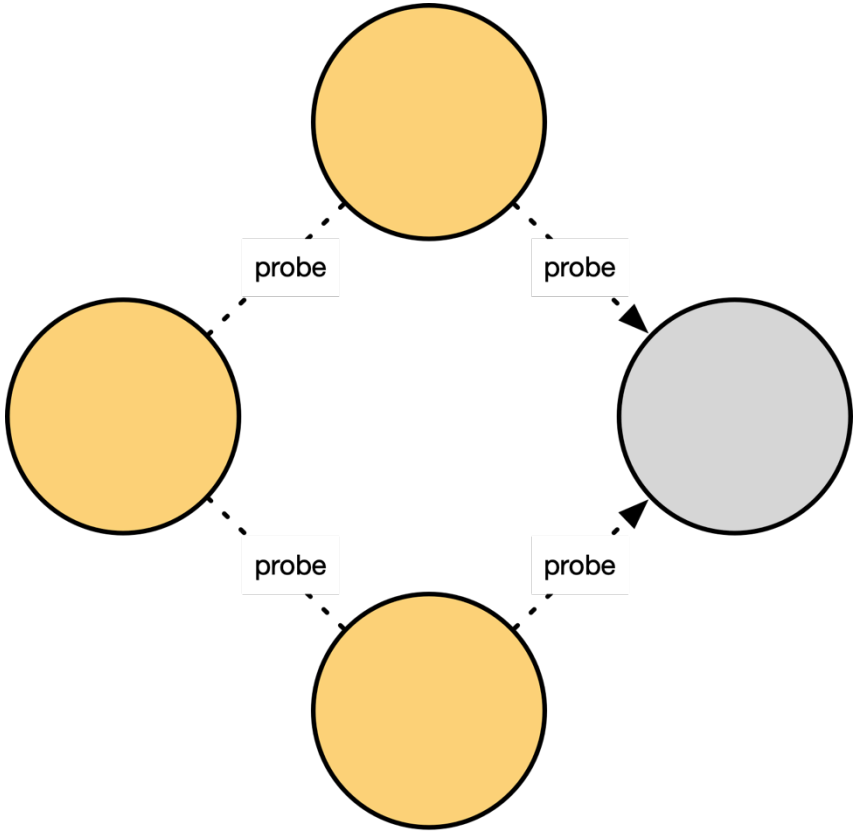
SWIM Protocol



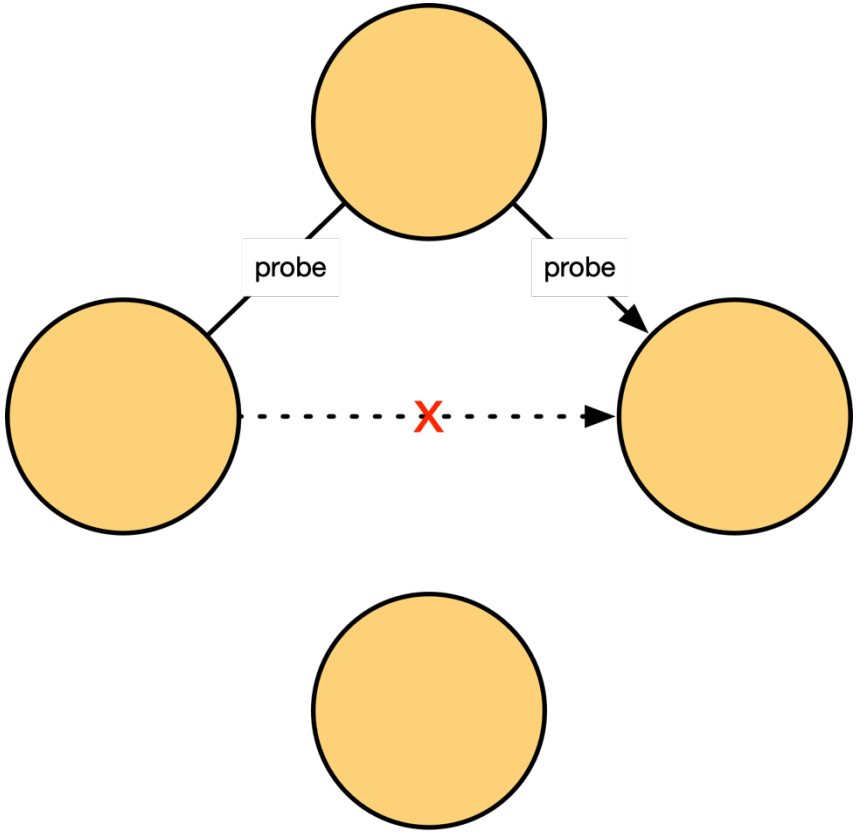
SWIM Protocol



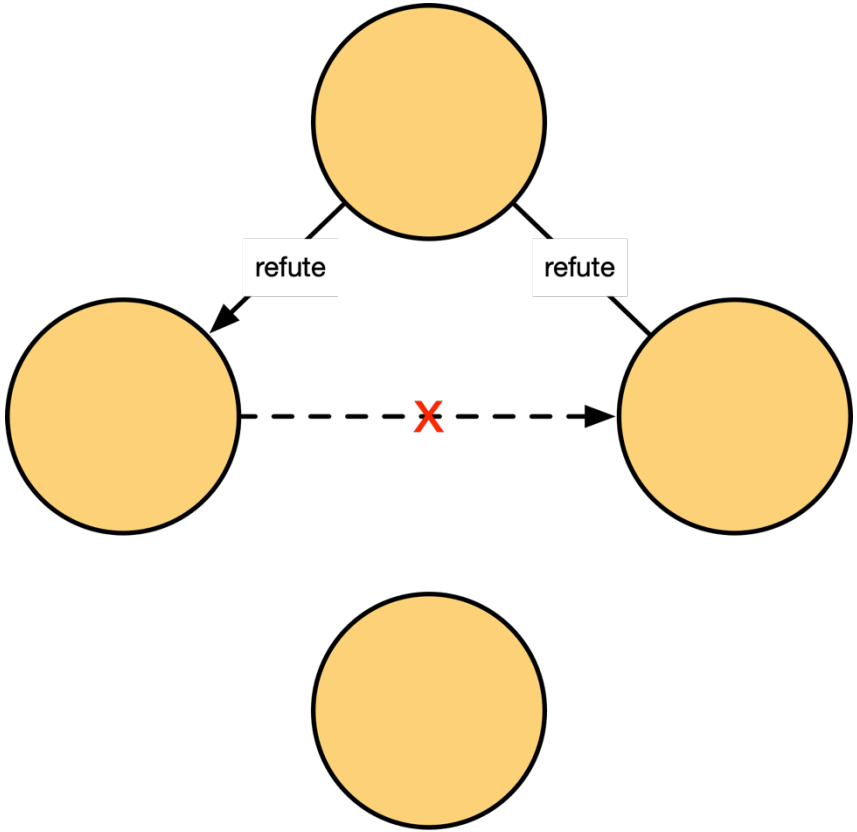
SWIM Protocol



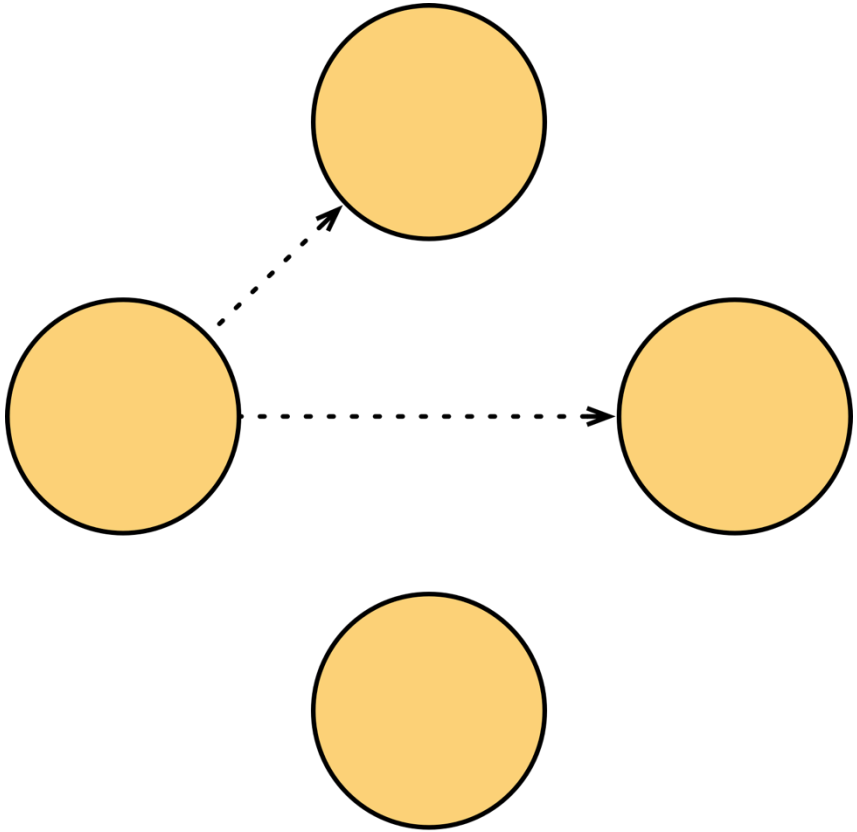
SWIM Protocol



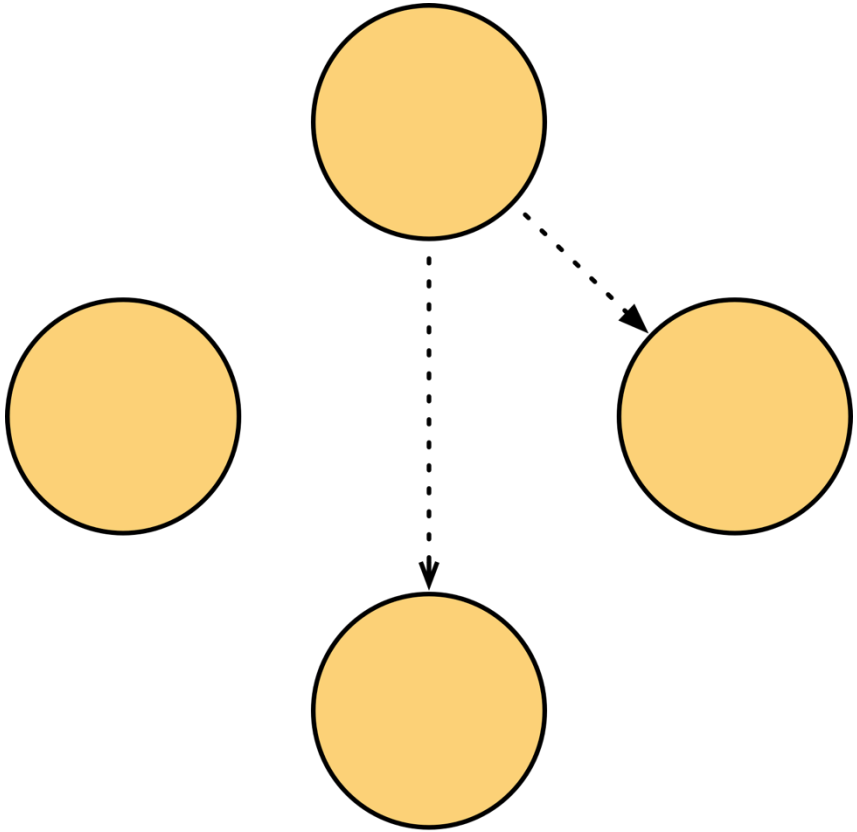
SWIM Protocol



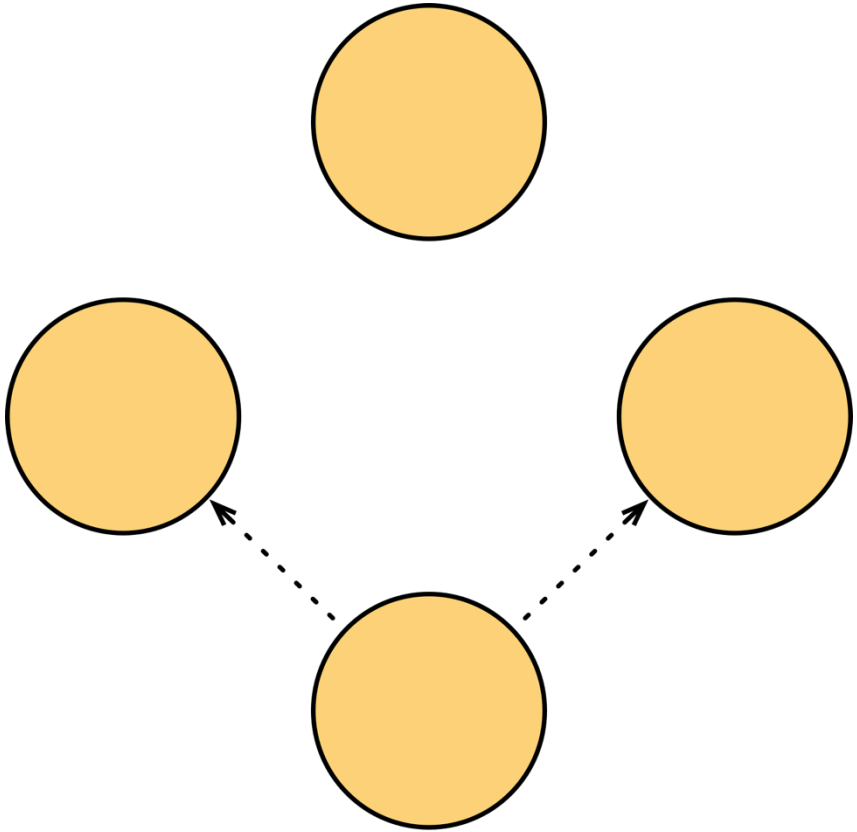
SWIM Protocol



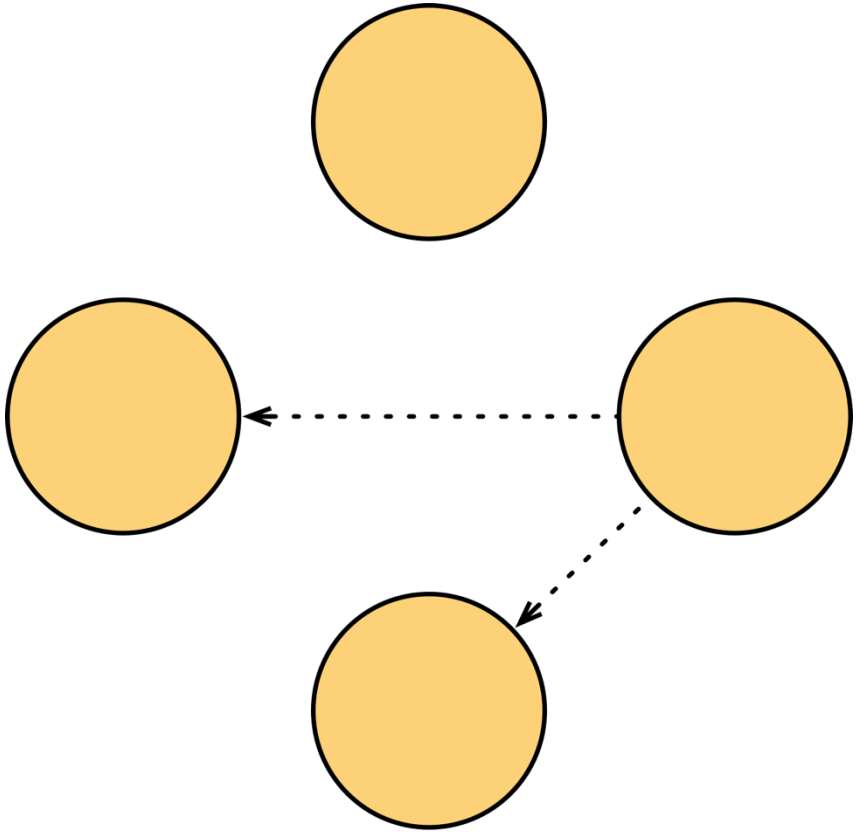
SWIM Protocol



SWIM Protocol



SWIM Protocol



SWIM Protocol

```
cluster.protocol {
   type: swim
   probeInterval: 500ms
   suspectProbes: 2
   gossipFanout: 2
   failureTimeout: 5s
}
```

SWIM Protocol

- Scales well
- Linear growth in network traffic
- Handles basic network partitions
- Avoids false positives
- But may take longer to detect failures

Replication Protocols

Replication Protocols

Raft

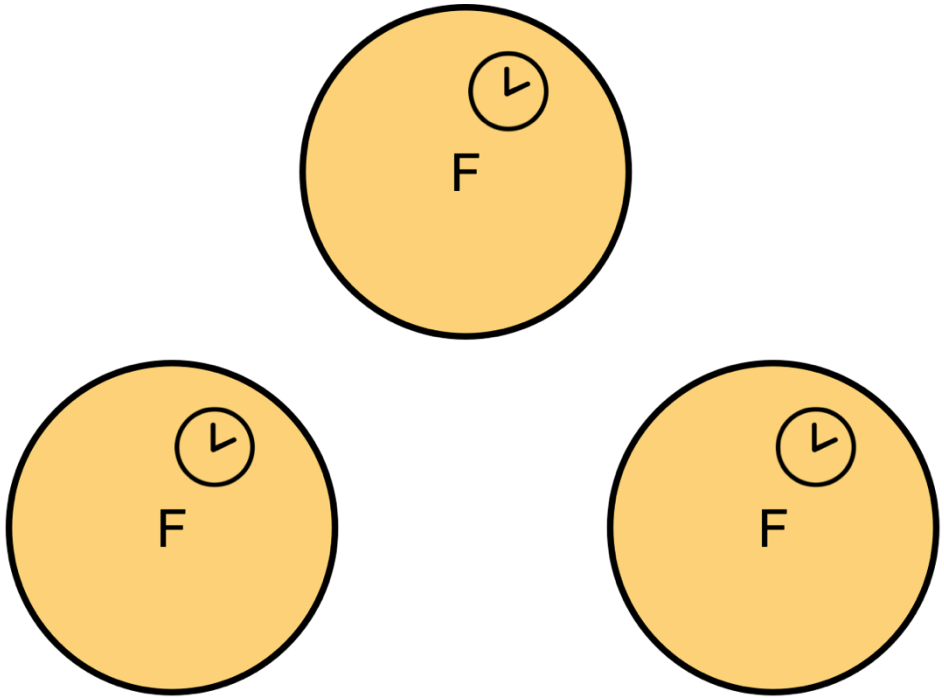
Primary-backup

Distributed log

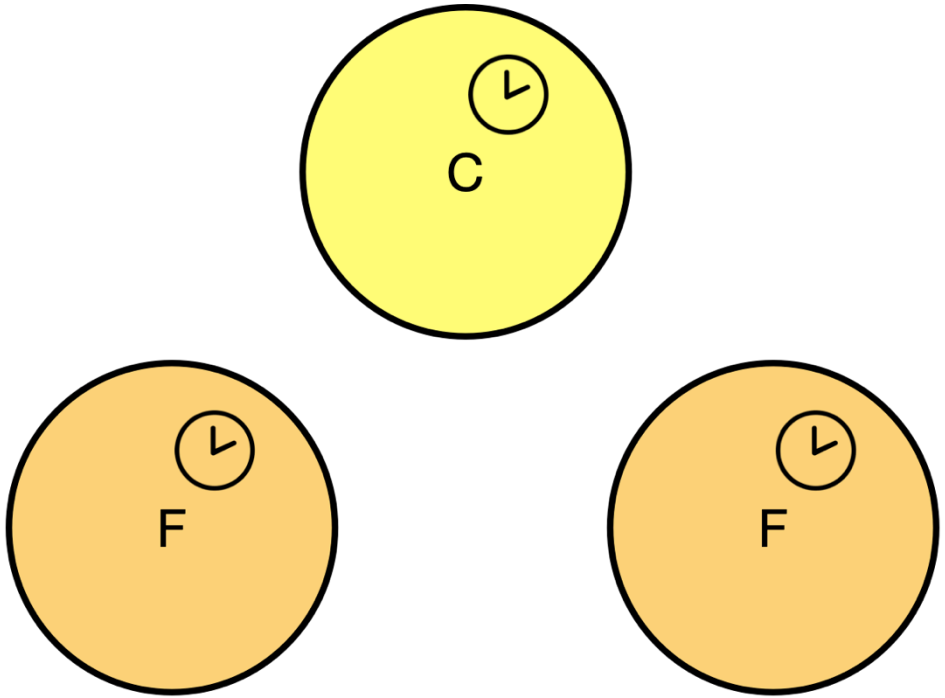
Raft Protocol

- Follower
  - Receives replicated entries from leader
  - Uses a timer to determine when leader is unavailable
- Candidate
  - Start an election
  - Request and count votes from peers
- Leader
  - Receive client requests
  - Append entries to leader and follower logs

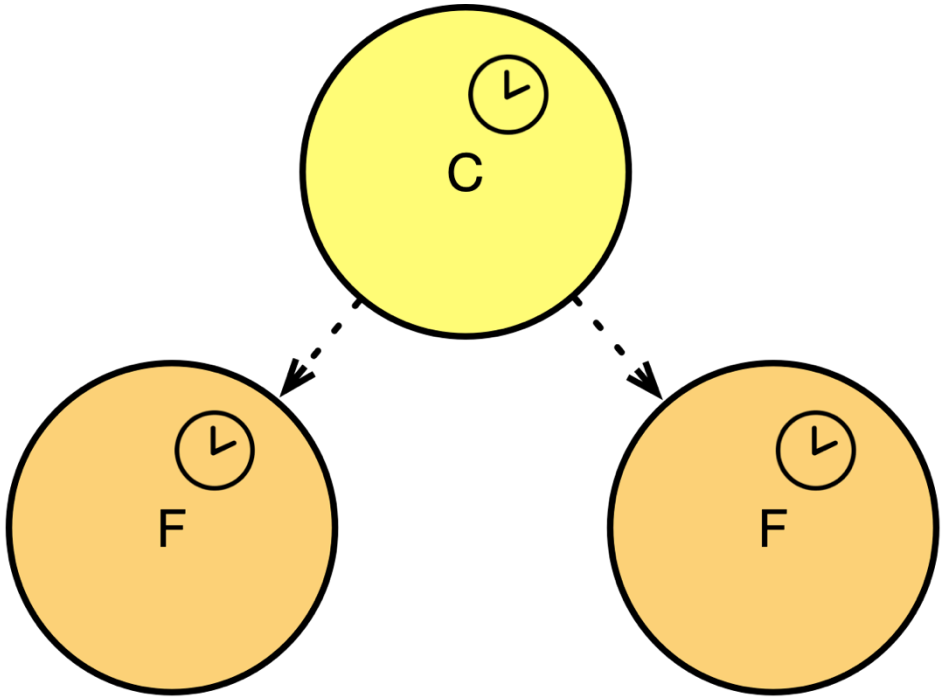
Raft Protocol



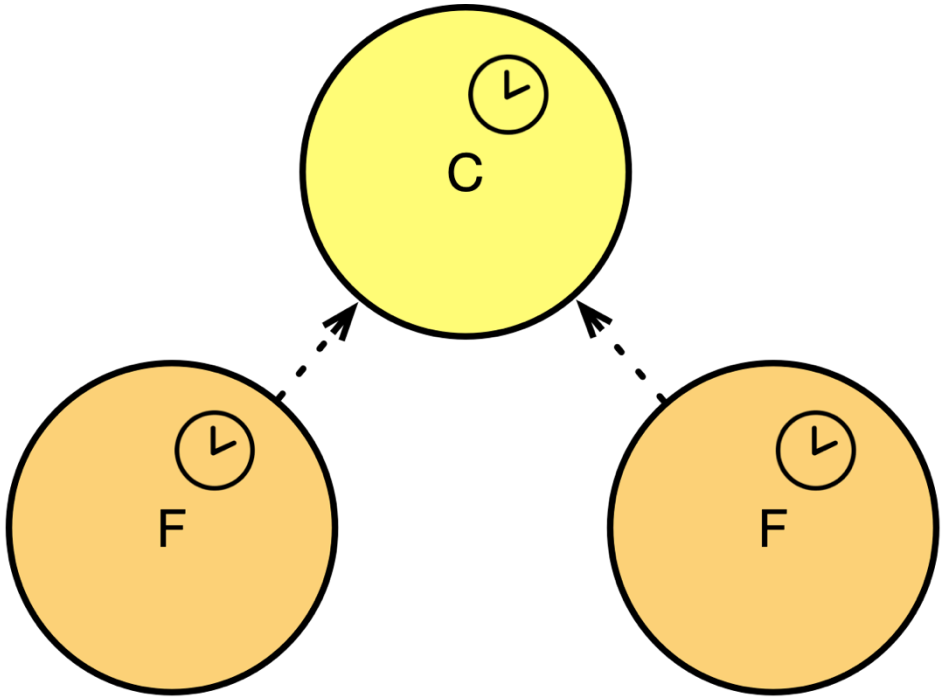
Raft Protocol



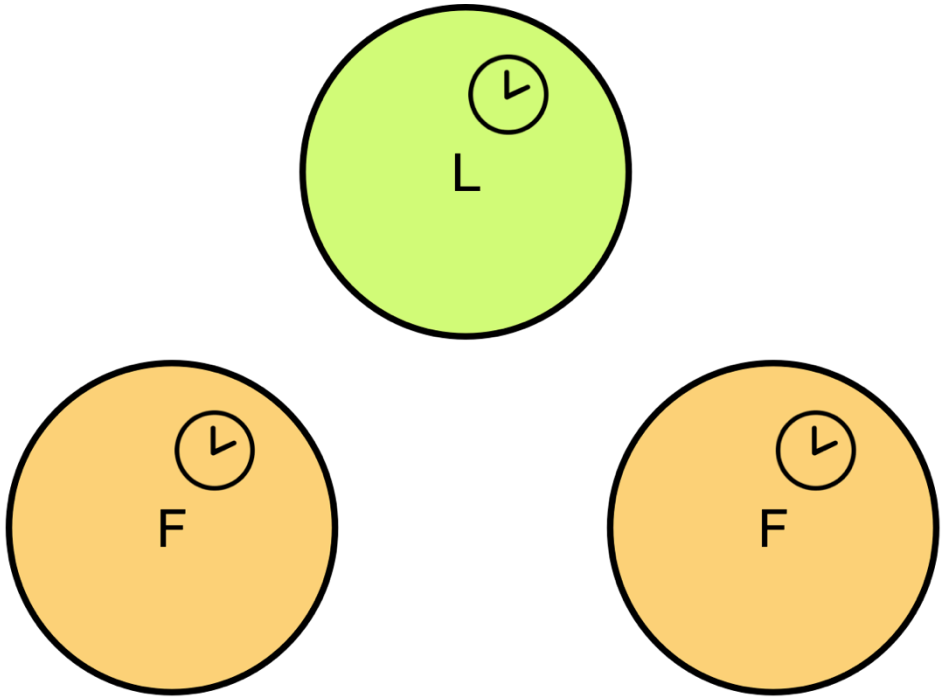
Raft Protocol



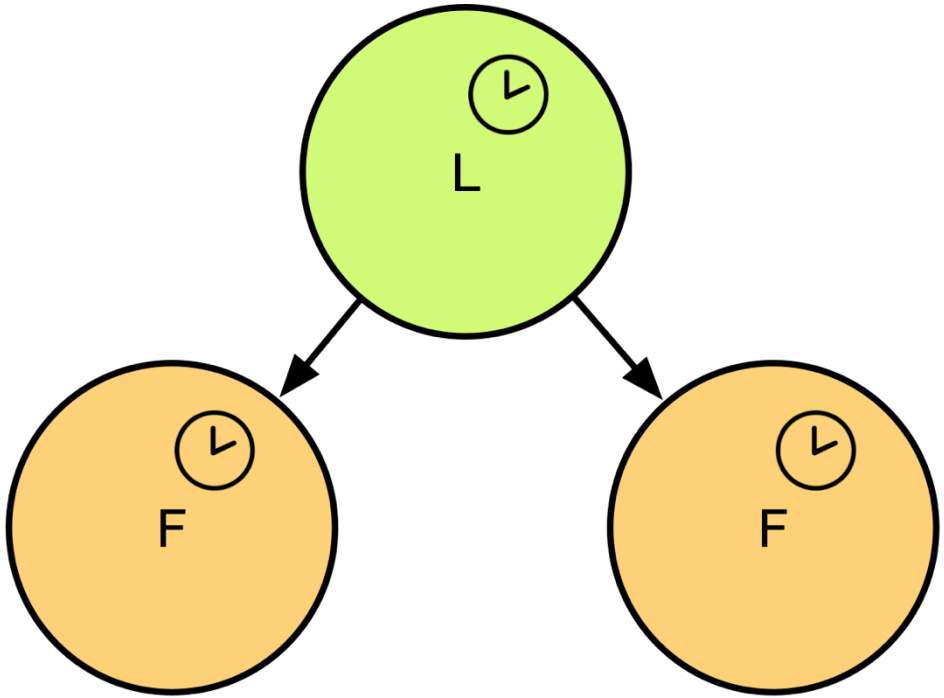
Raft Protocol



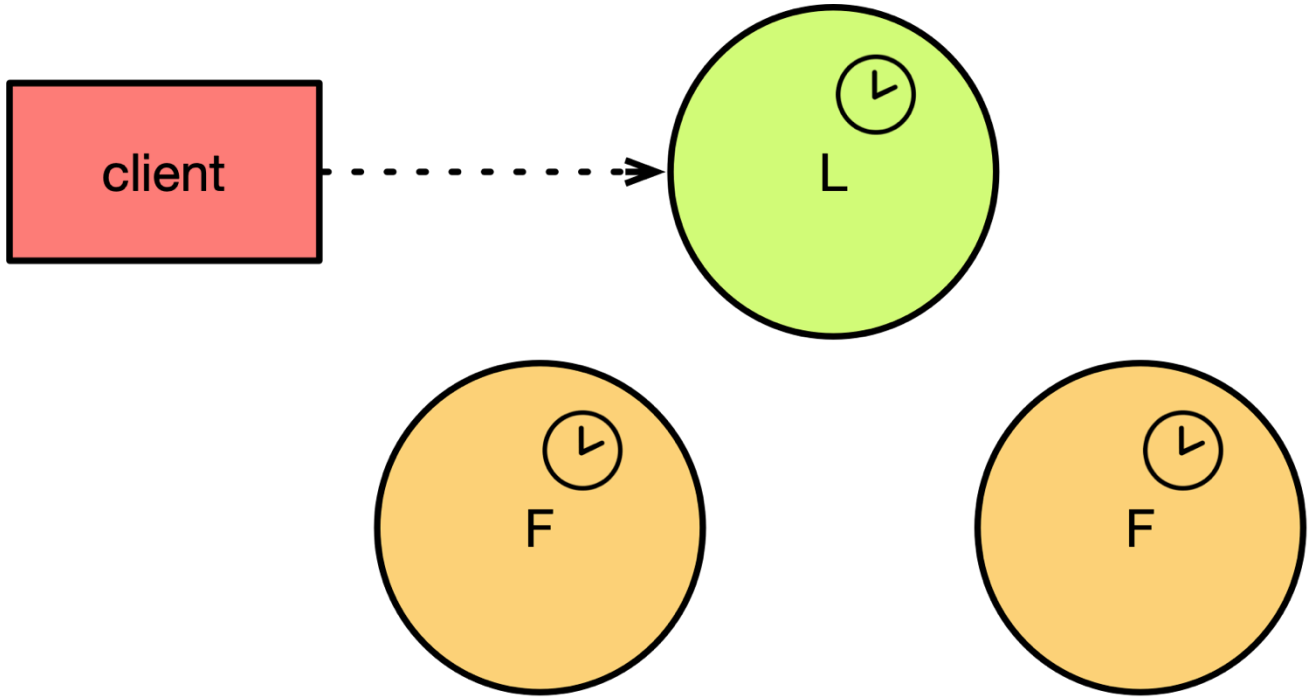
Raft Protocol



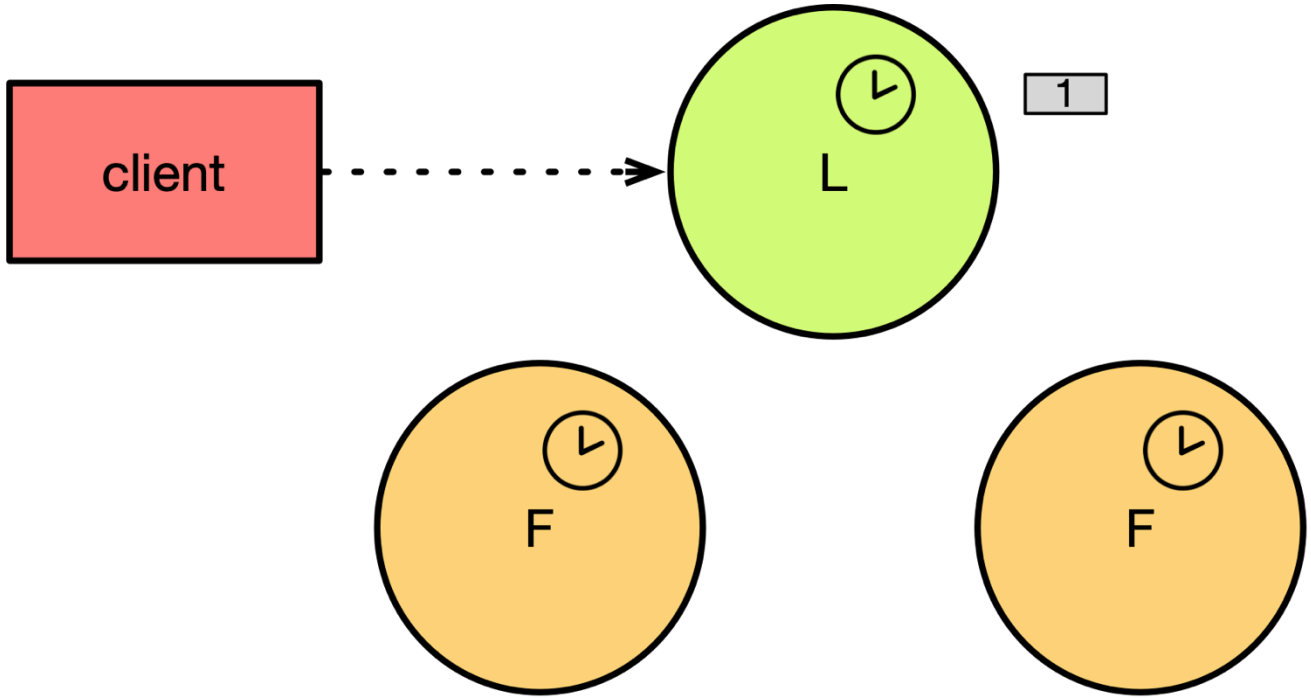
Raft Protocol



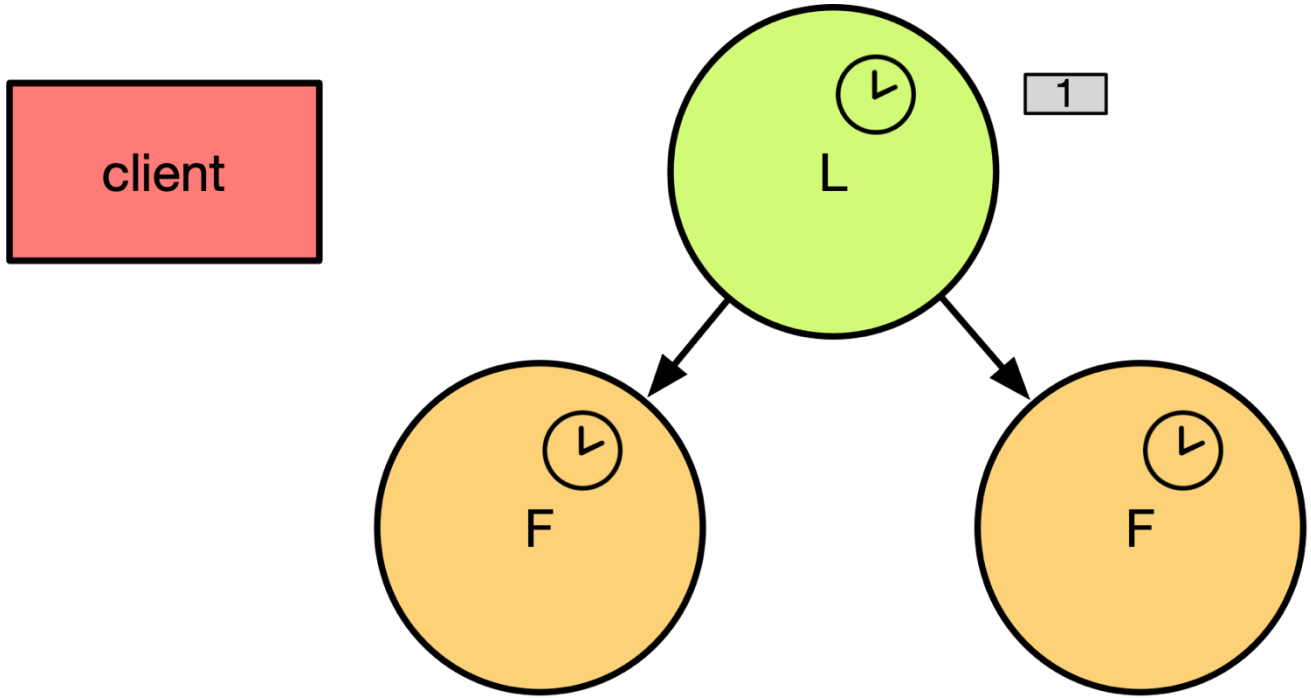
Raft Protocol



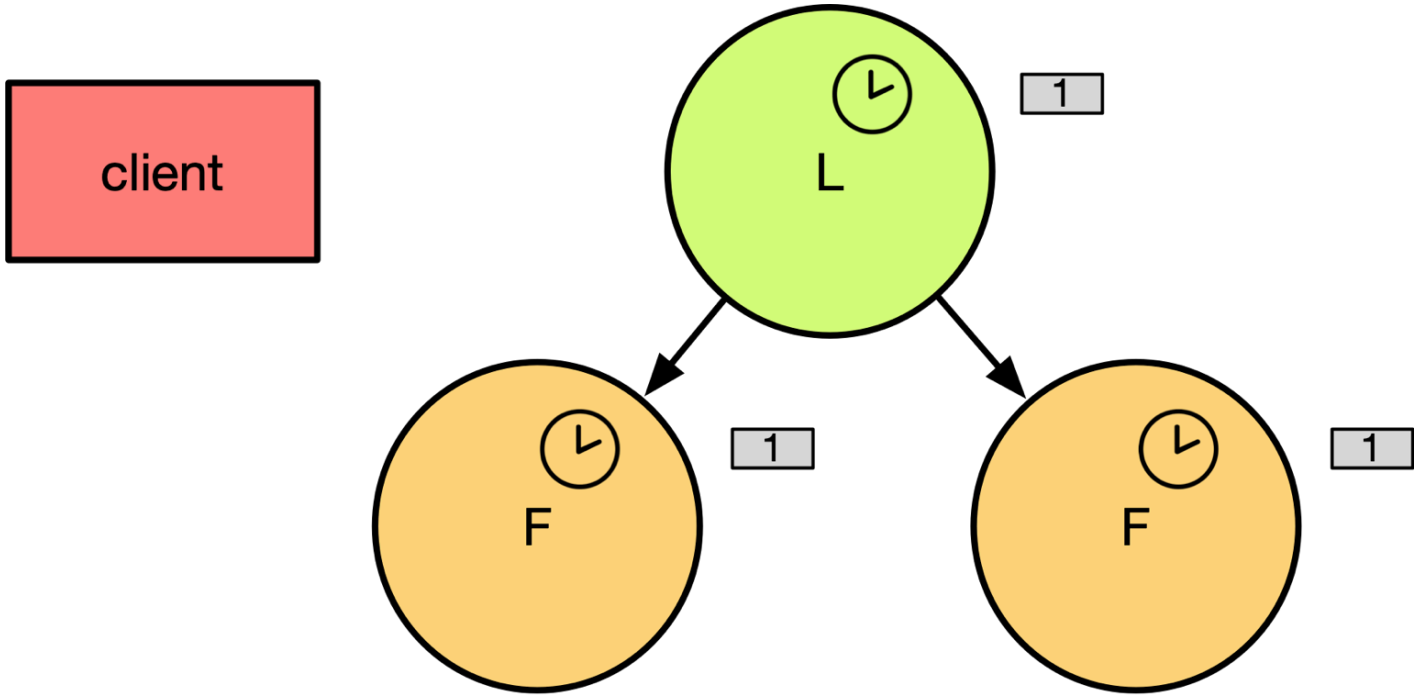
Raft Protocol



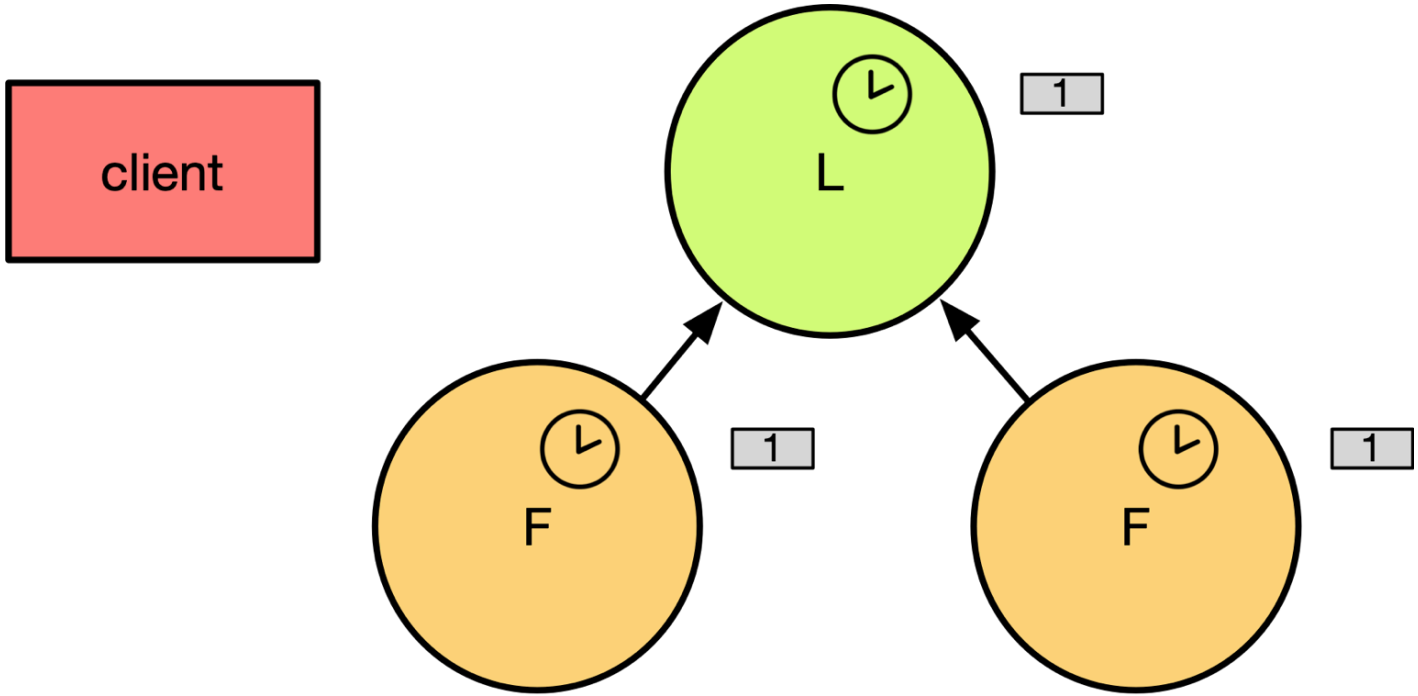
Raft Protocol



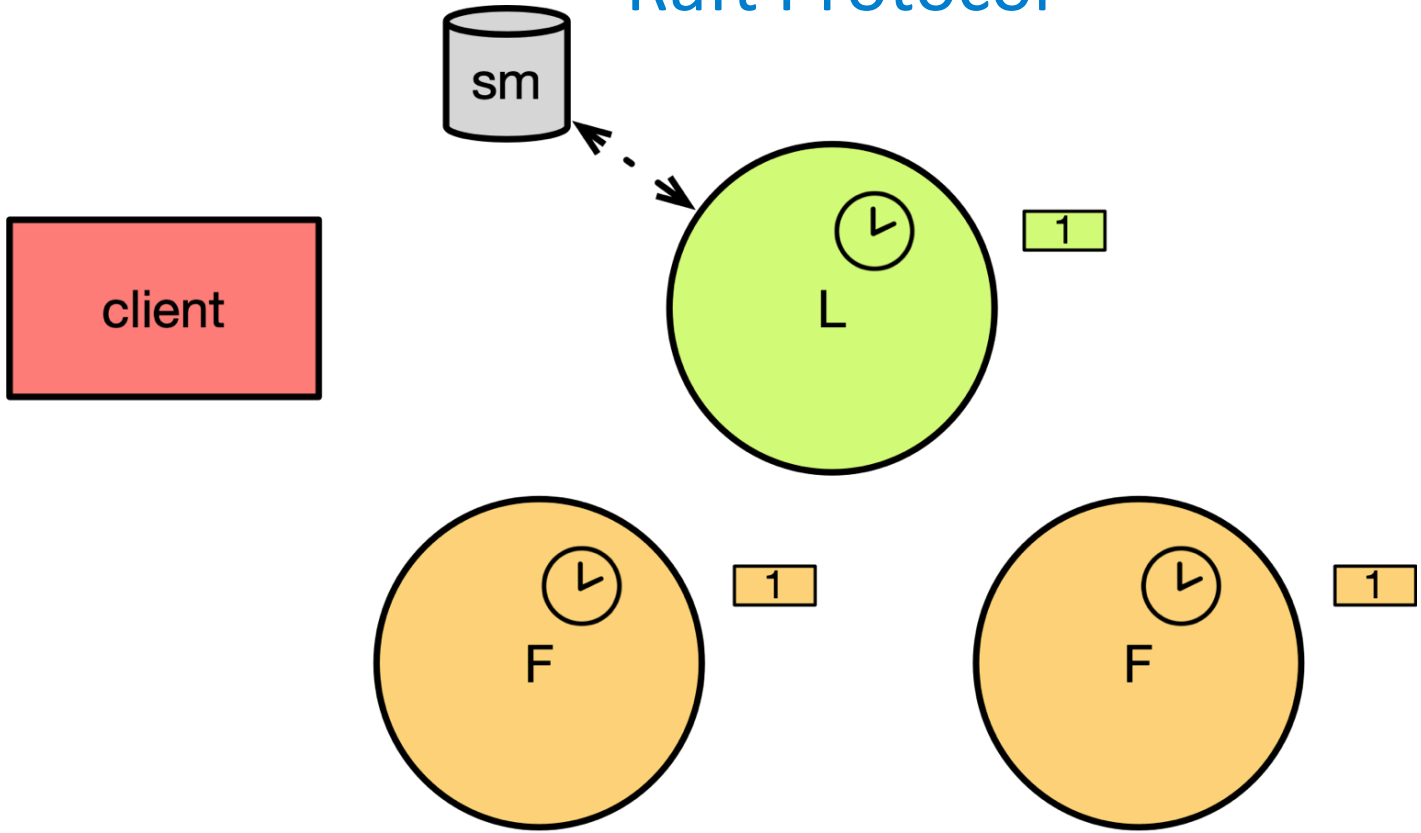
Raft Protocol



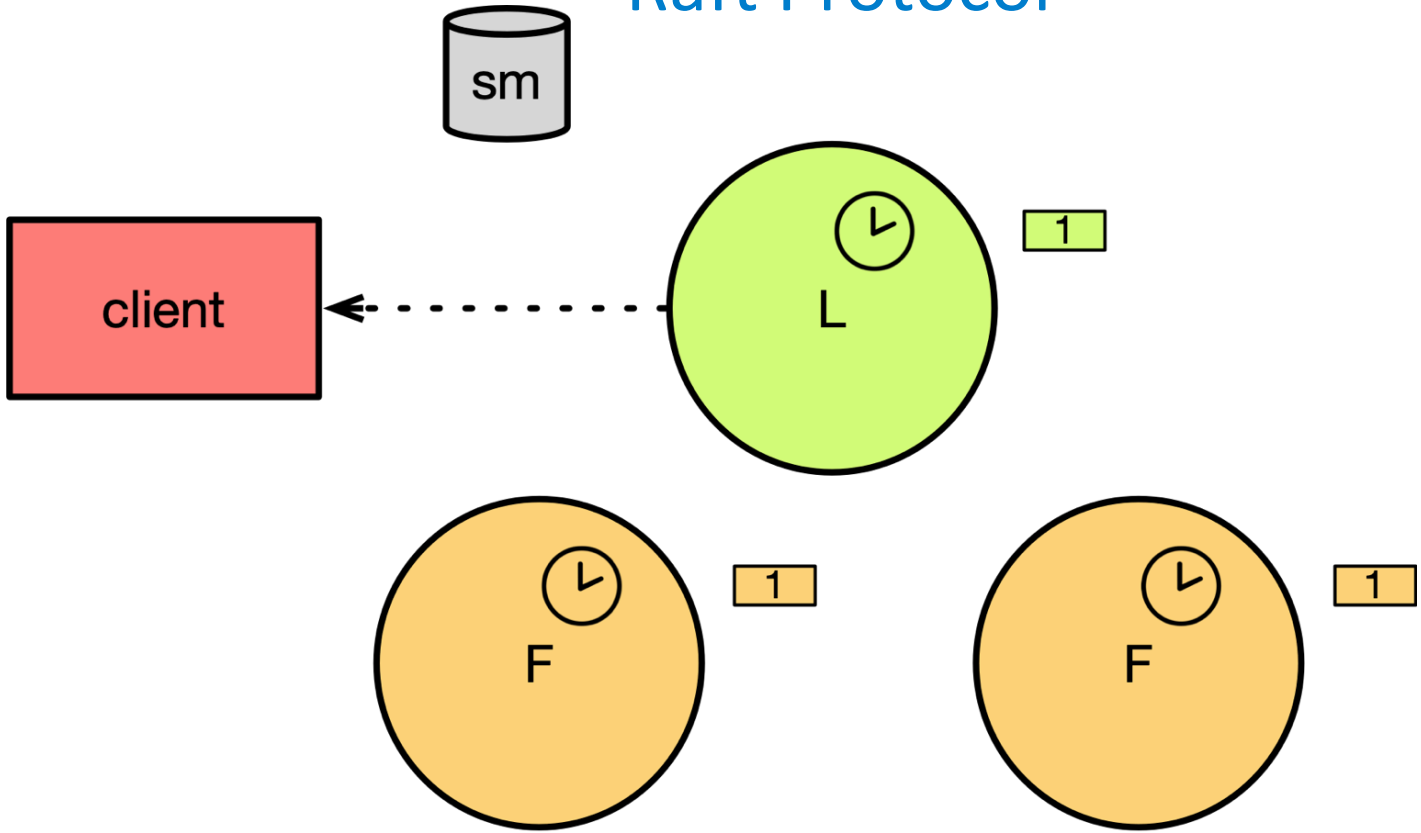
Raft Protocol



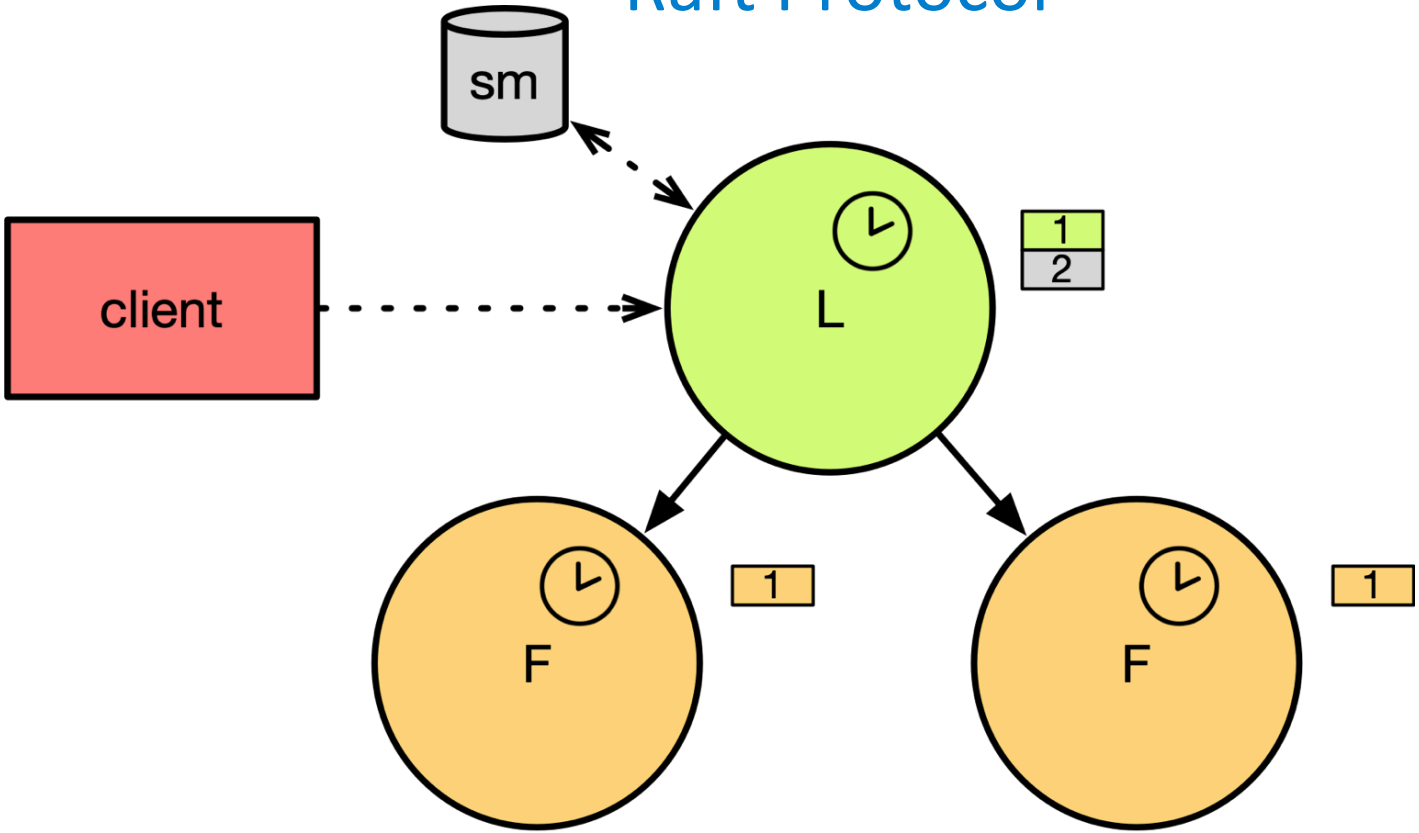
Raft Protocol



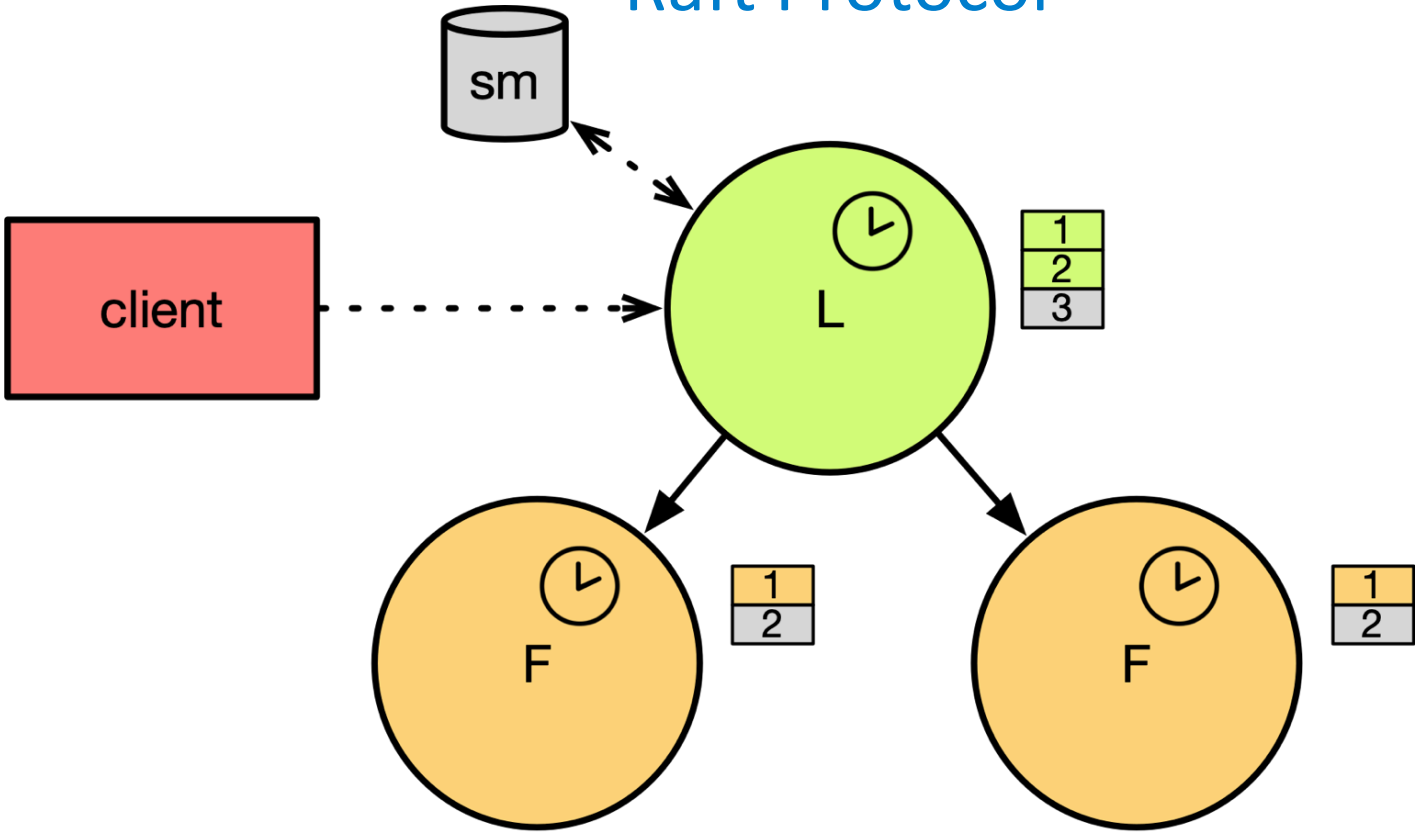
Raft Protocol



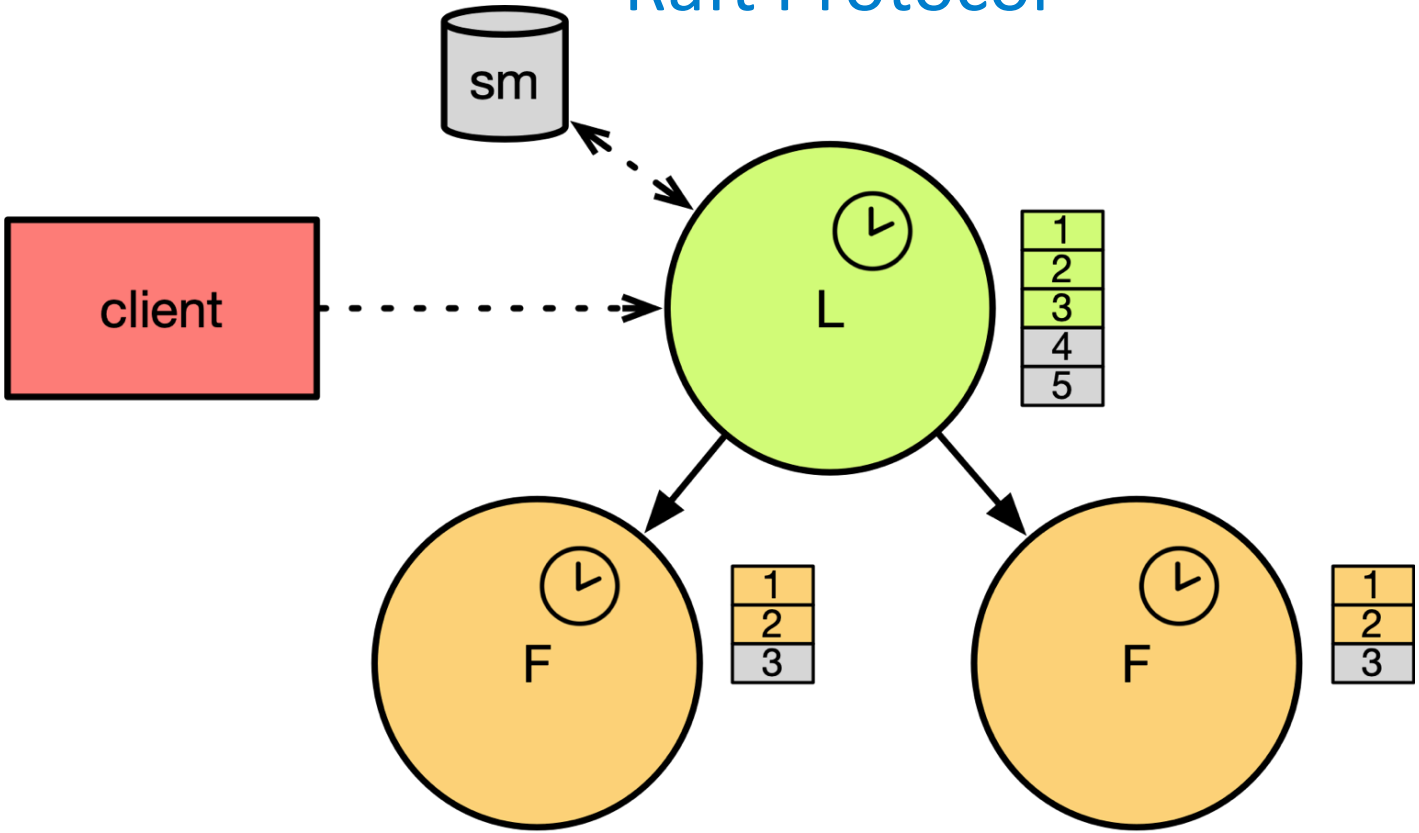
Raft Protocol



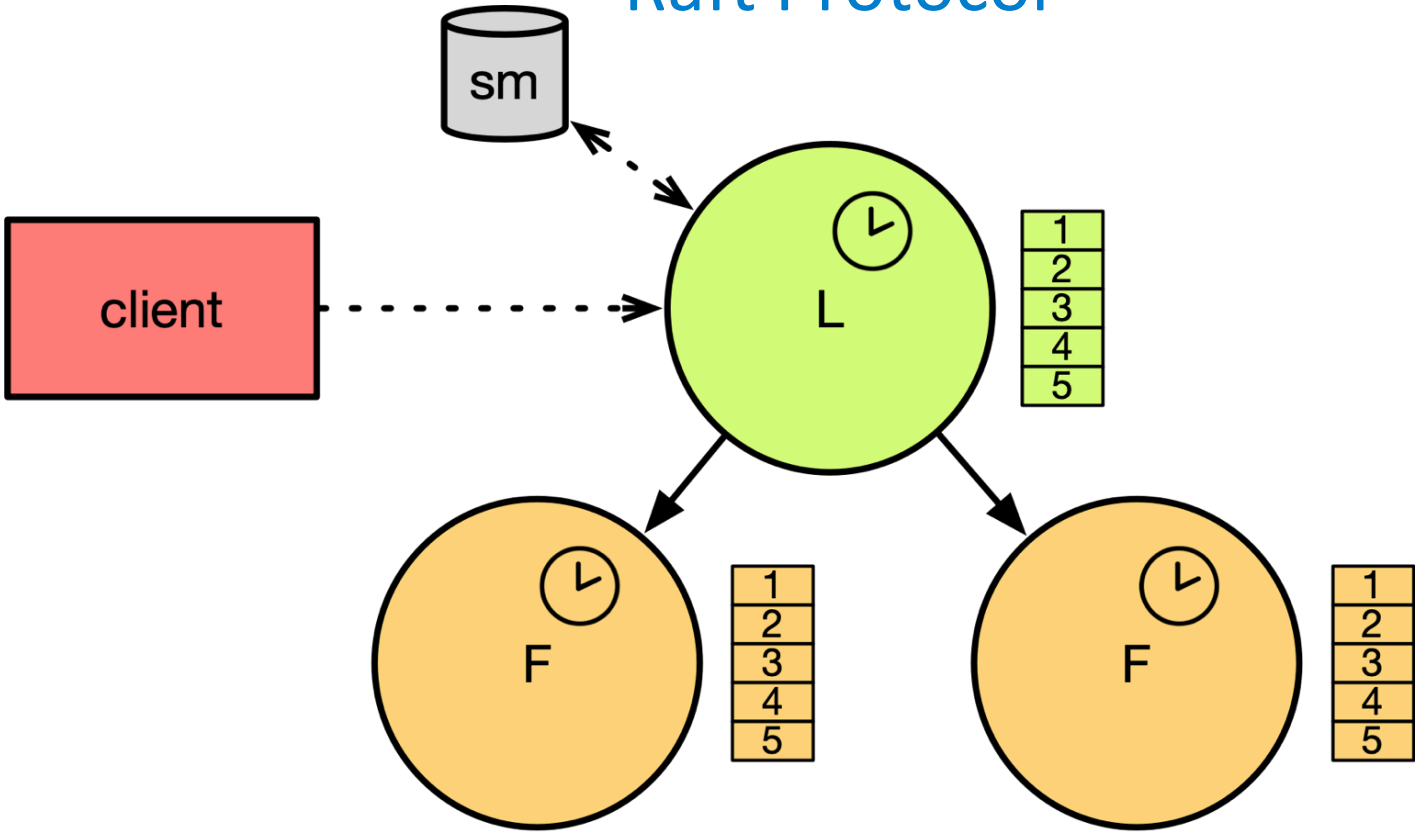
Raft Protocol



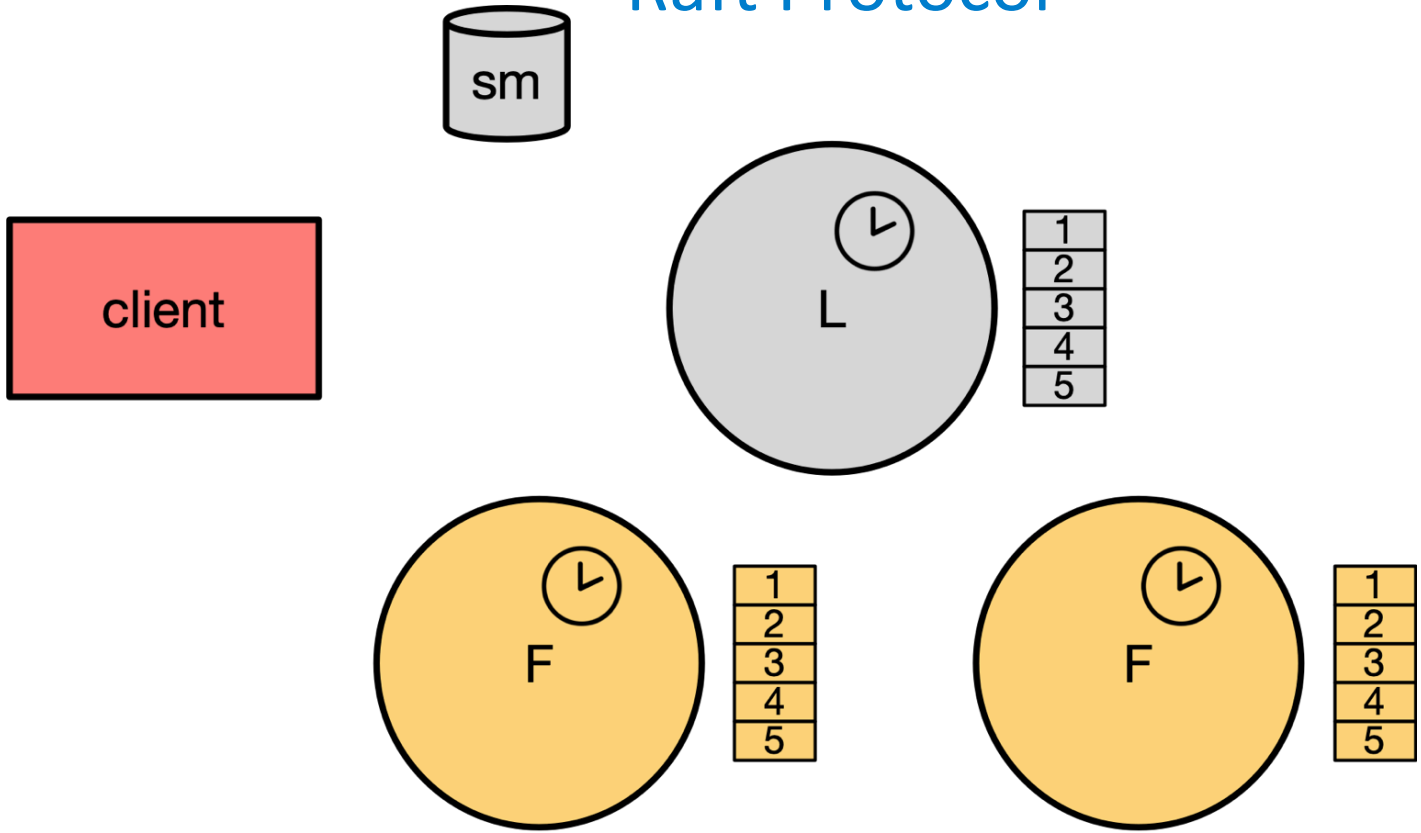
Raft Protocol



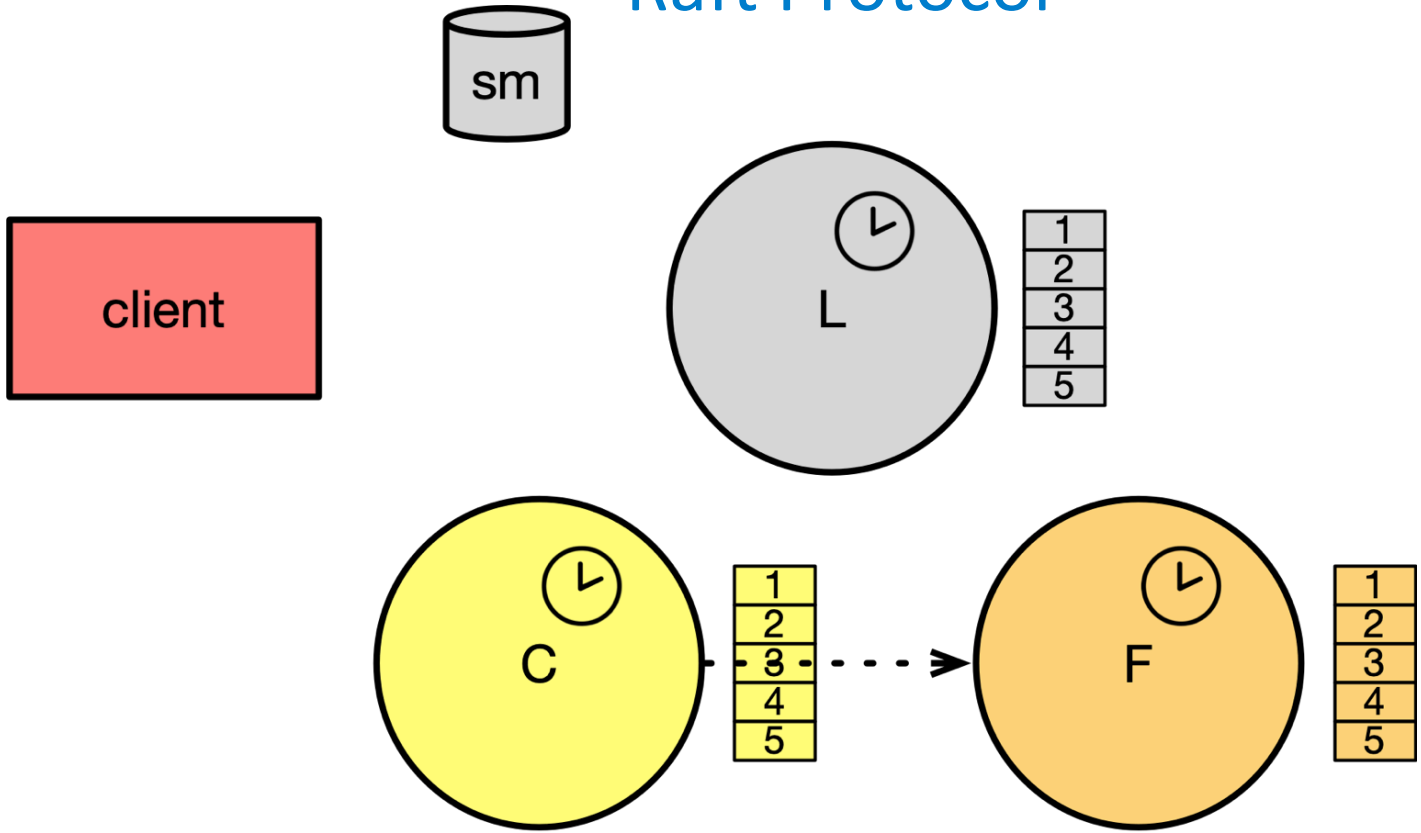
Raft Protocol



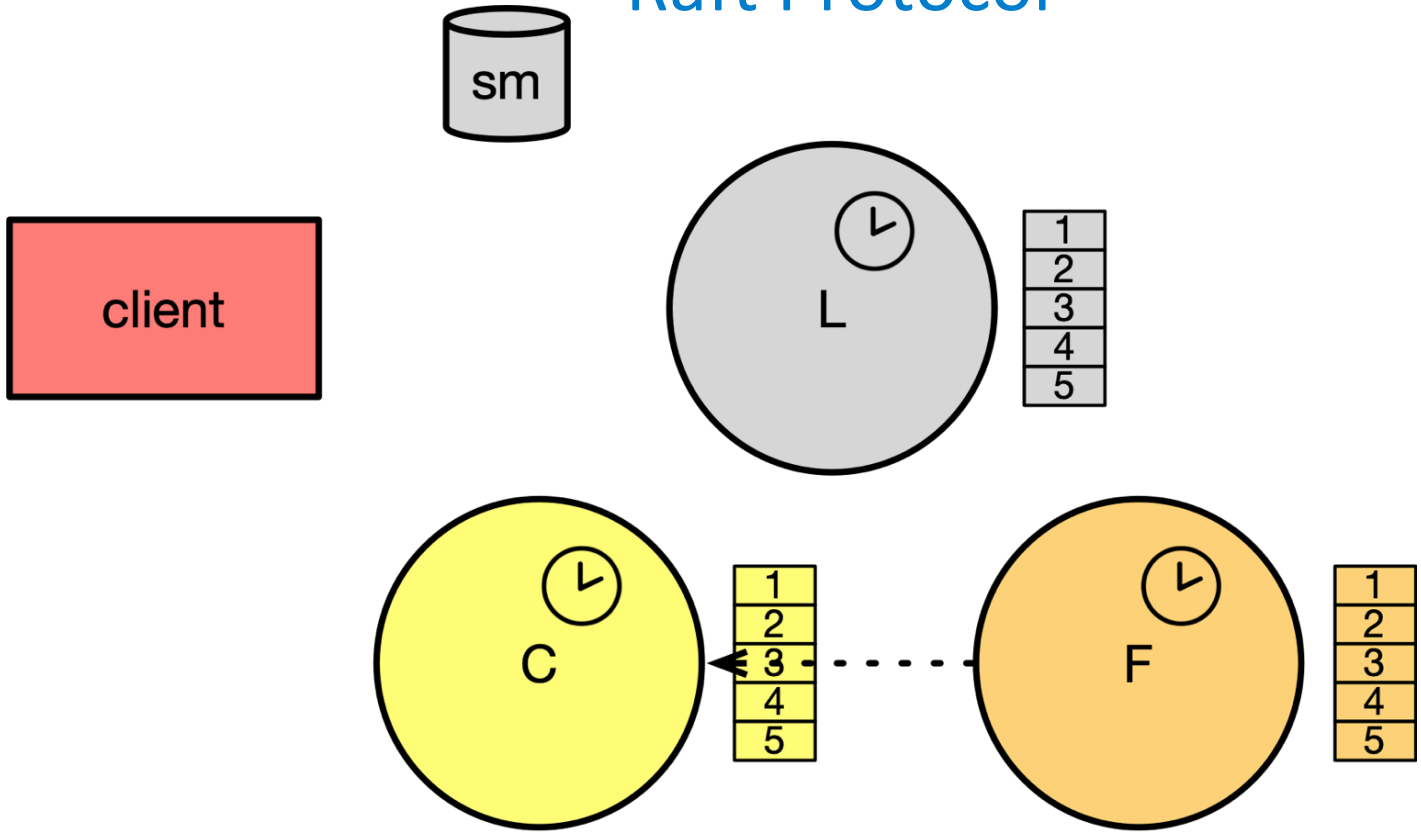
Raft Protocol



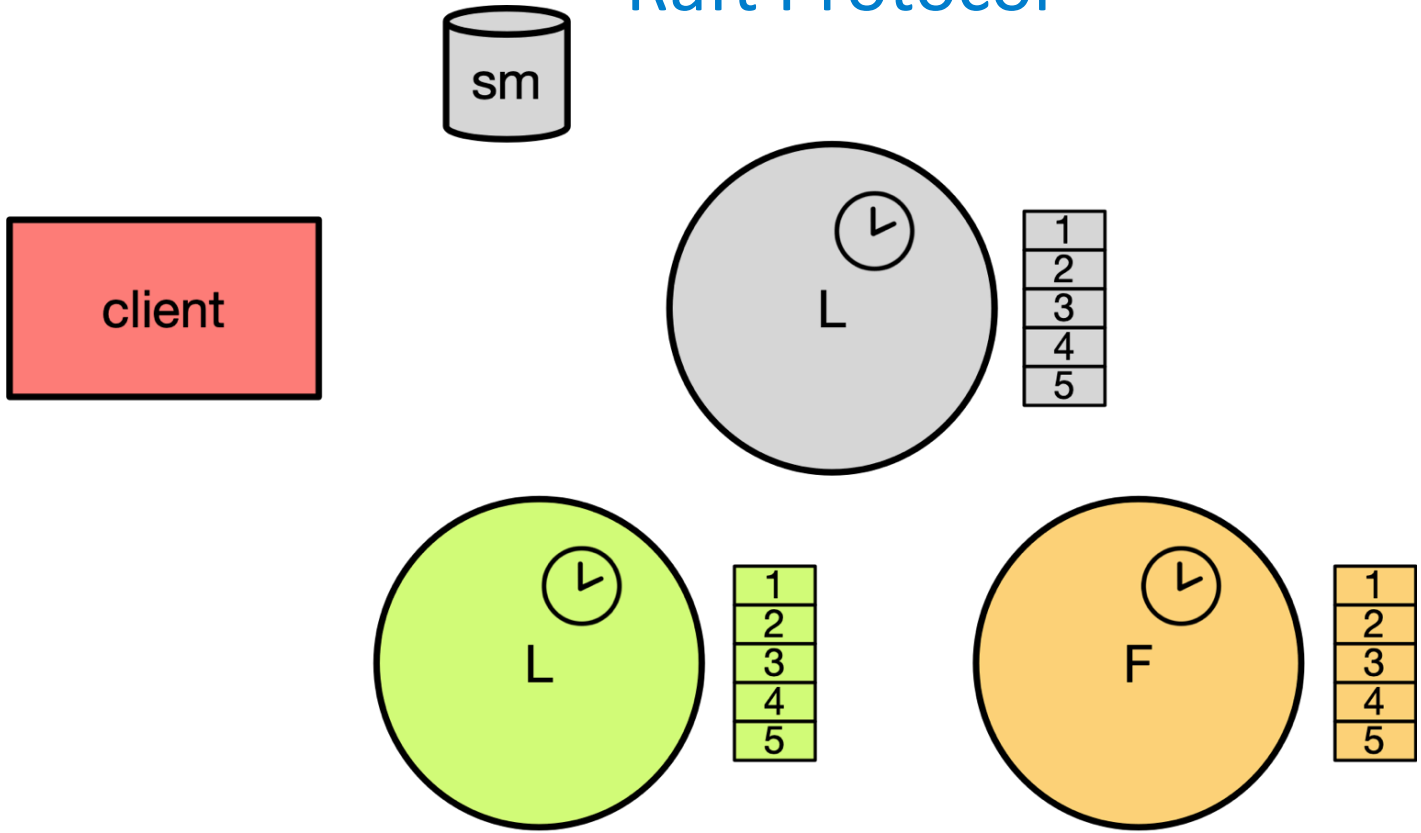
Raft Protocol



Raft Protocol



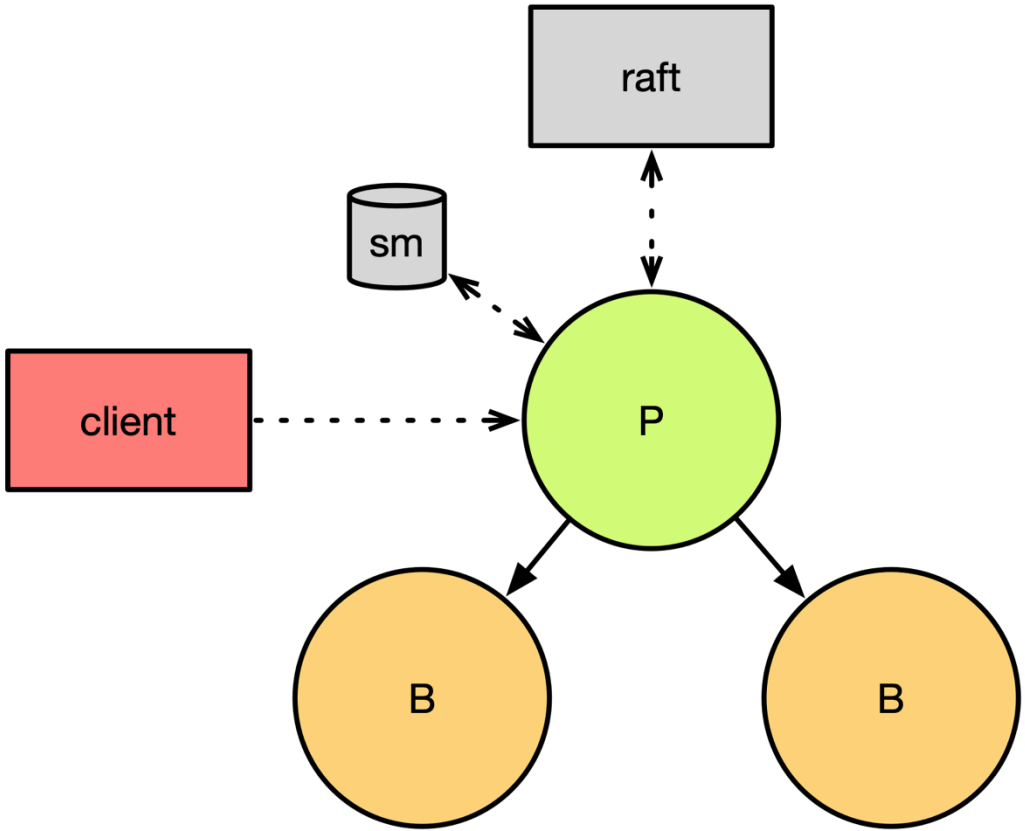
Raft Protocol



Primary Backup Protocol

- Strong leader
- Elect a primary
- Replicate state changes from primary to *n* backups
- Changes committed once replicated to *r* backups
- Leader election done through Raft

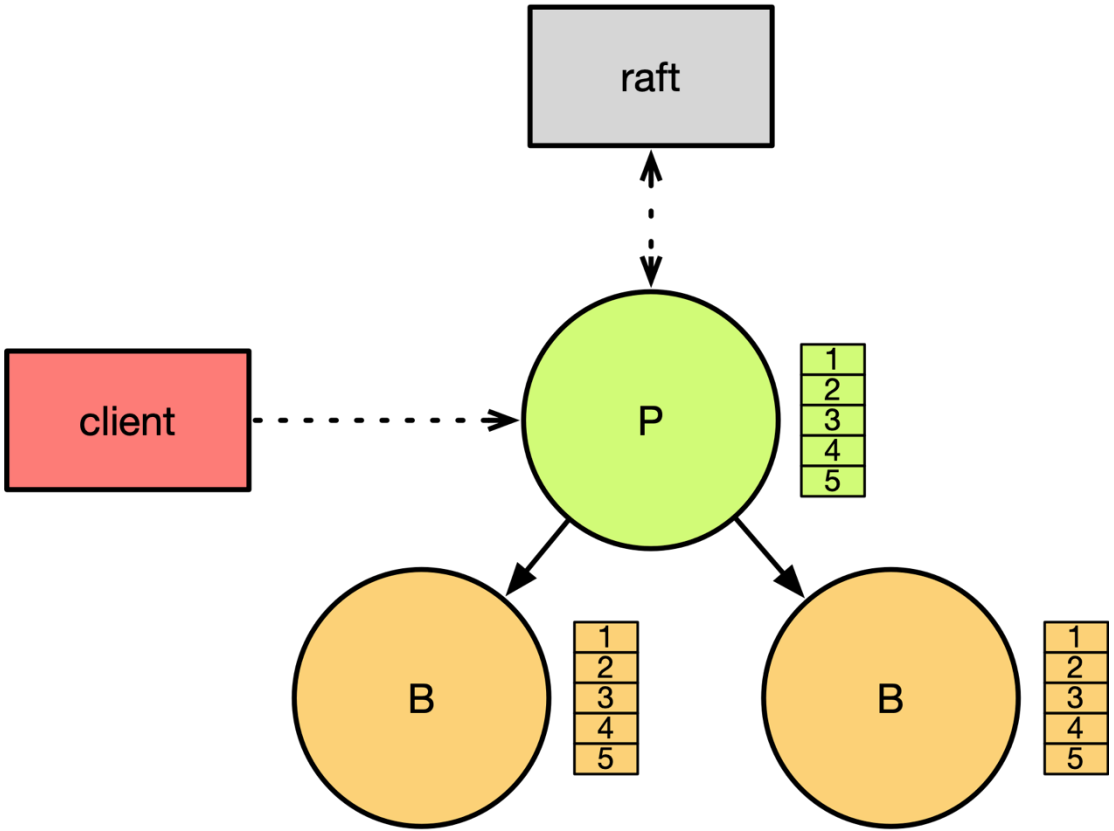
Primary Backup Protocol



Distributed Log Protocol

- Strong leader
- Elect a primary
- Replicate log entries from primary to *n* backups
- Entries committed once replicated to *r* backups
- Leader election done through Raft

Distributed Log Protocol

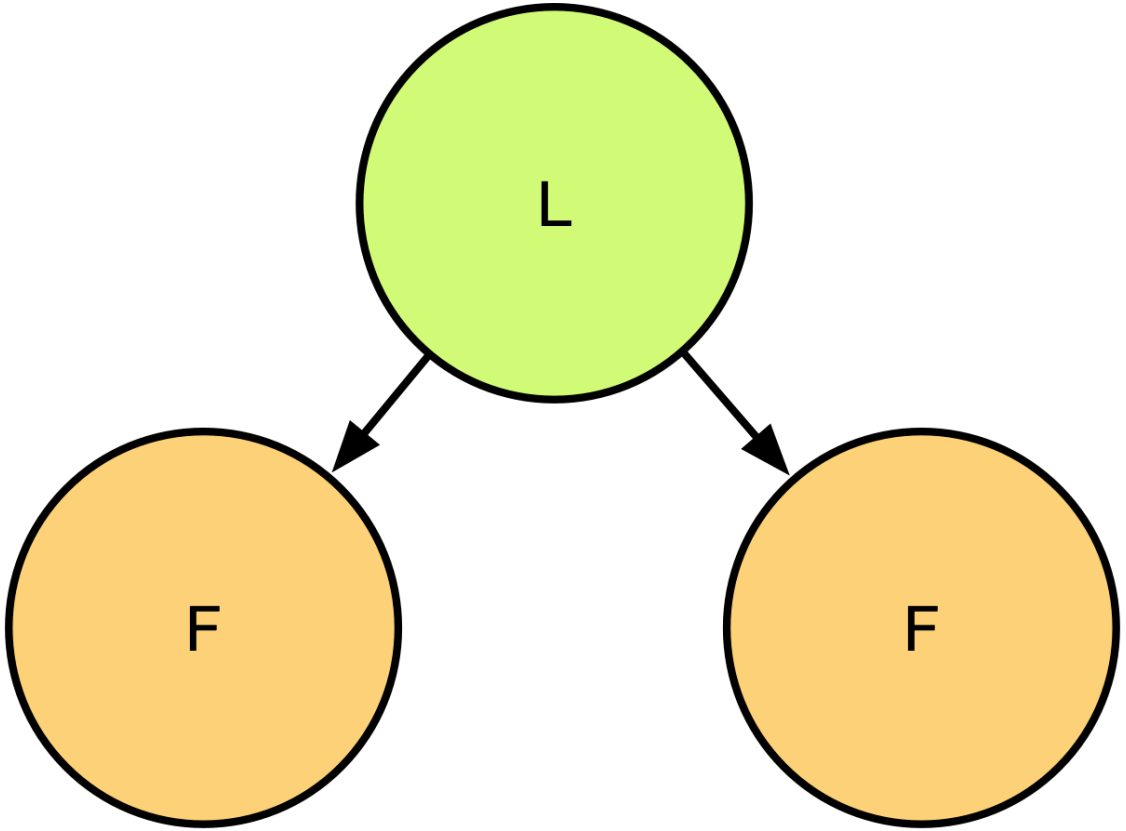


Partition Groups

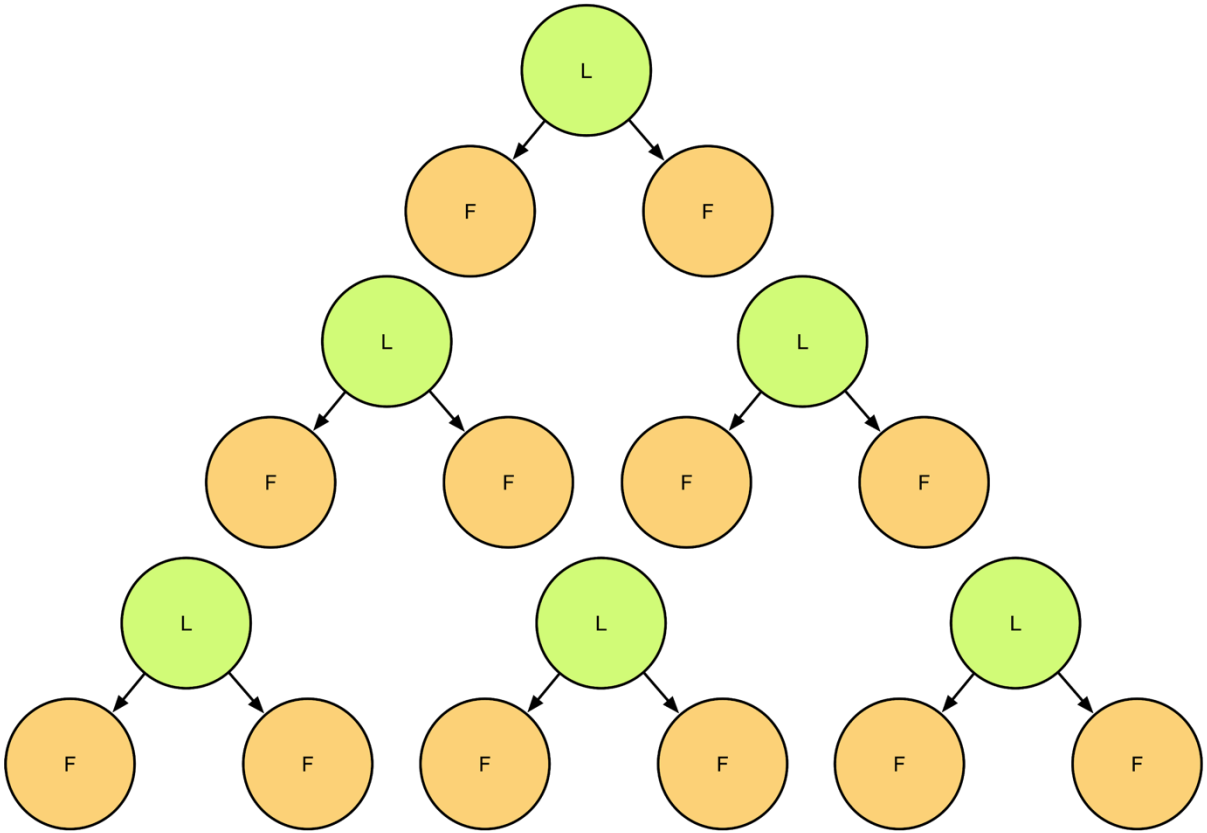
Partition Groups

- Leader-based protocols do not scale well
- Must shard protocols to scale
- Run multiple instances of each protocol in a group

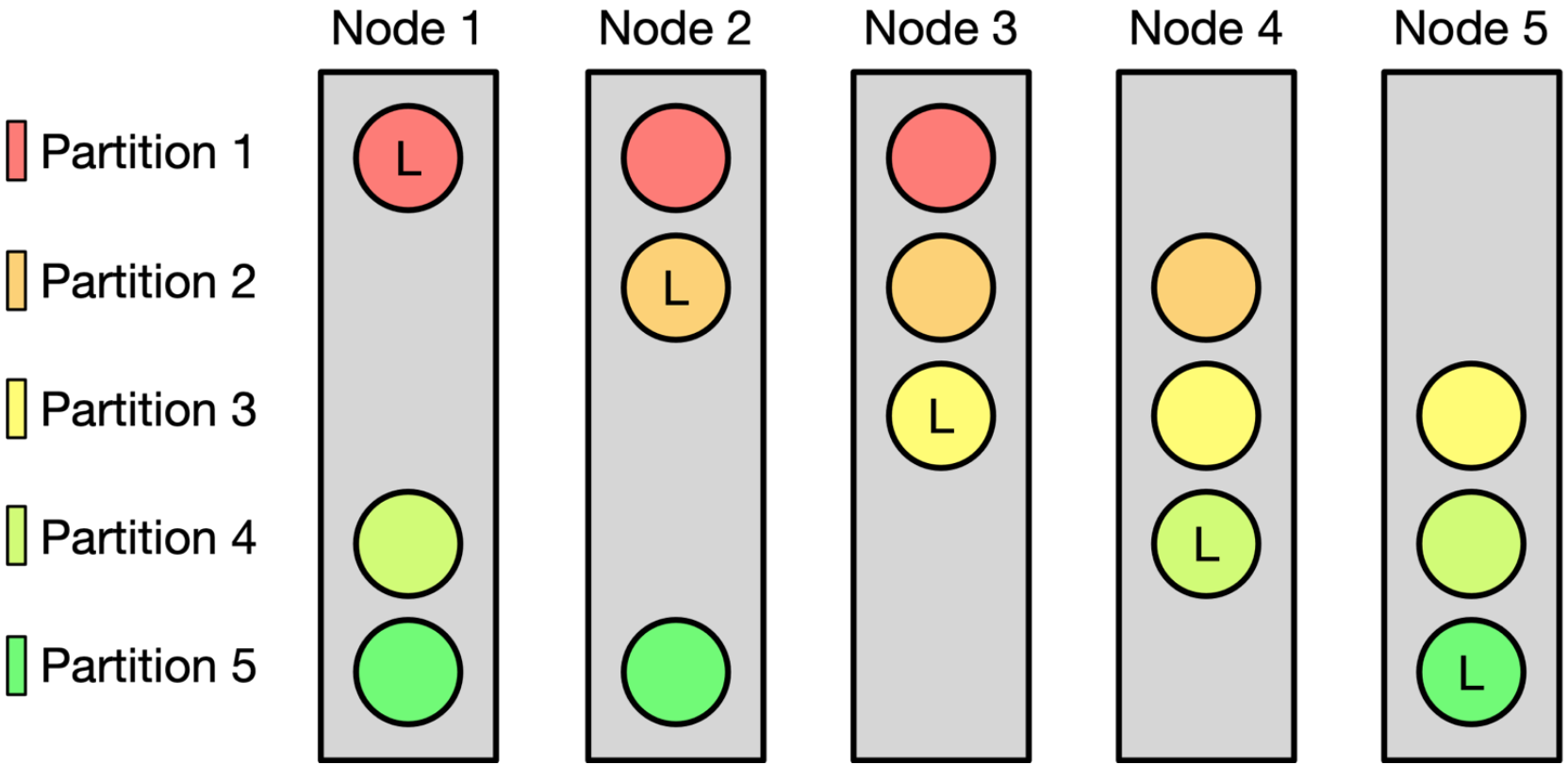
Partition Groups



Partition Groups



Partition Groups



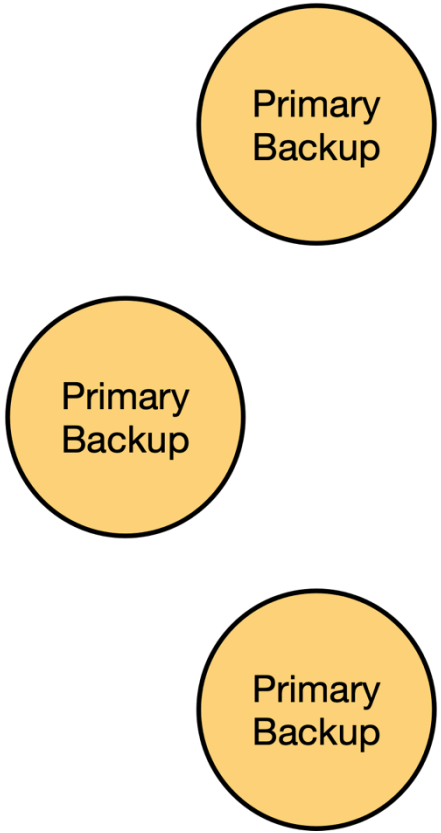
Partition Groups

- Partition groups are an abstraction for configuring sharded instances of a protocol
  - RaftPartitionGroup
  - PrimaryBackupPartitionGroup
  - LogPartitionGroup
- Partition groups may reside on any node according to its configuration

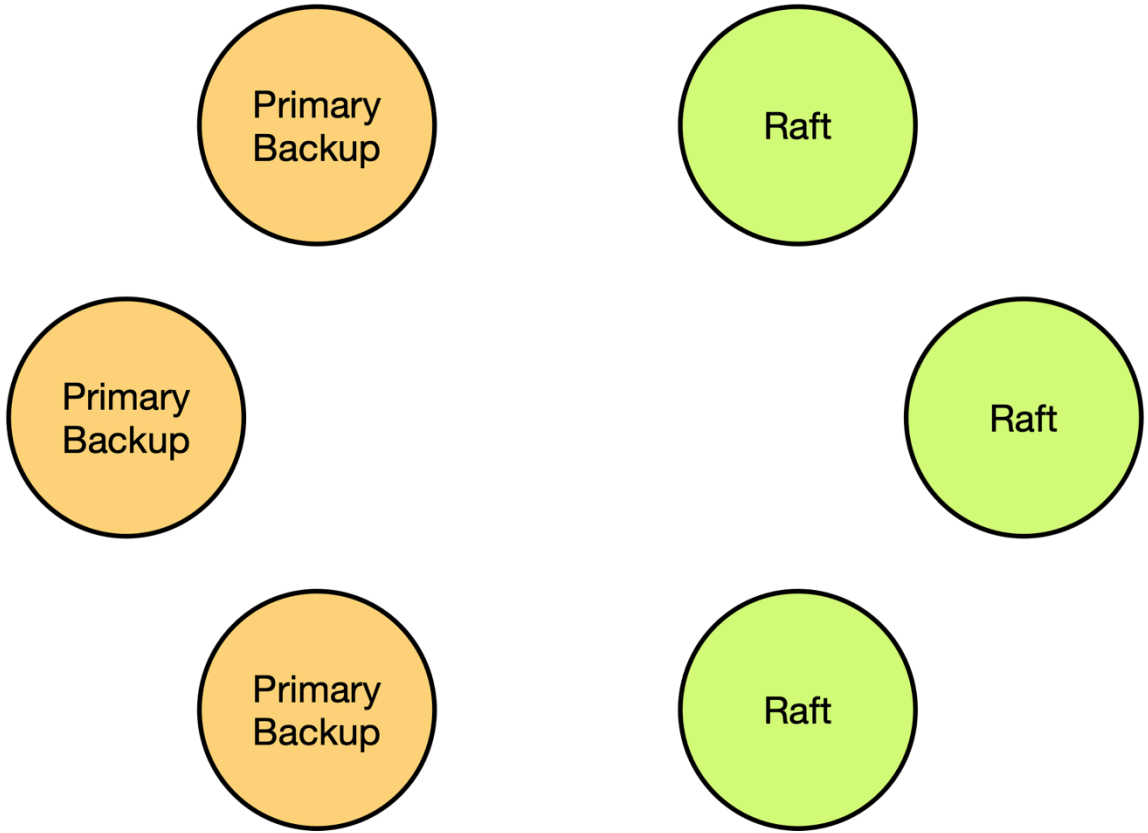
Partition Groups

```
partitionGroups.raft {
   type: raft
   partitions: 3
   storage.level: mapped
   members: [atomix-1, atomix-2, atomix-3]
}
```

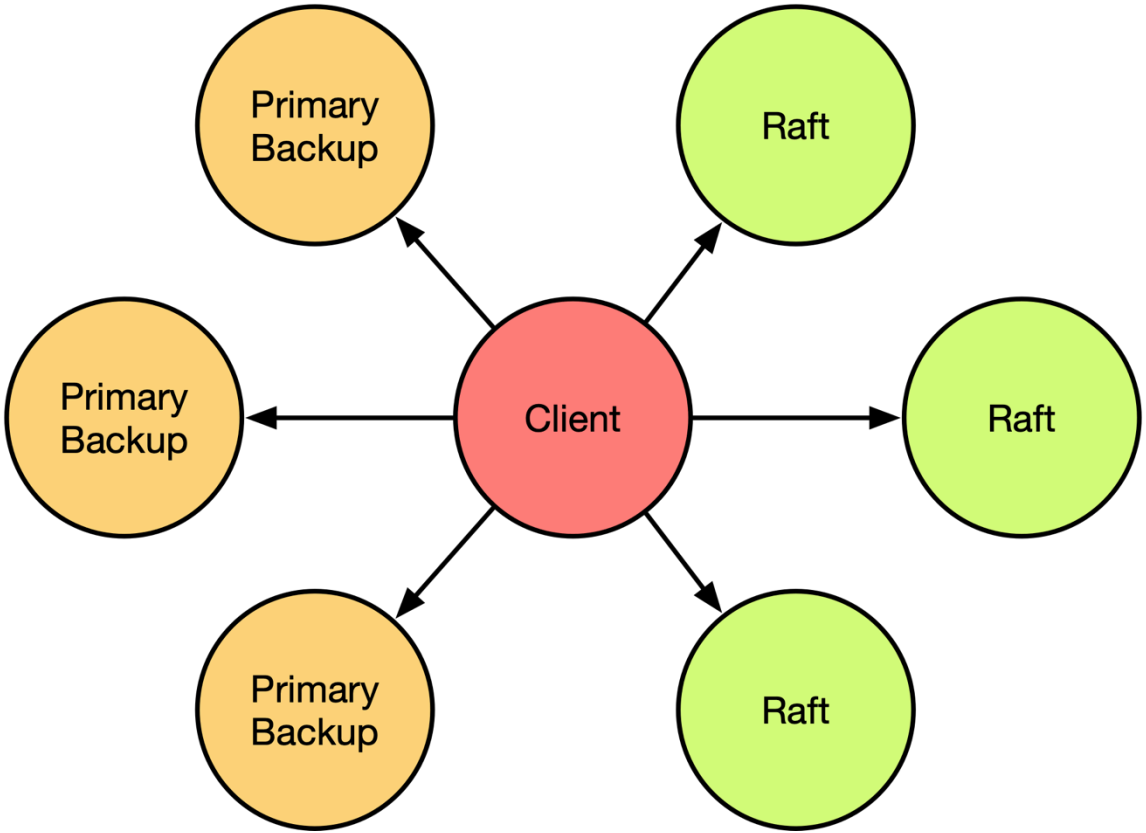
Partition Groups



Partition Groups



Partition Groups



Partition Groups

- Management group
  - A single required group
  - Primary election
  - Primitive management
  - Transaction management
- Primitive groups
  - Any number of optional groups
  - Store distributed primitives

Partition Groups

```
managementGroup {
  type: raft
  partitions: 1
  members: [atomix-1, atomix-2, atomix-3]
}

partitionGroups.raft {
  type: raft
  partitions: 3
  partitionSize: 3
  storage.level: mapped
  members: [atomix-1, atomix-2, atomix-3]
}

partitionGroups.data {
  type: primary-backup
  partitions: 32
}
```

Partition Groups

```
Atomix atomix = Atomix.builder()
    .withMemberId("member-1")
    .withManagementGroup(RaftPartitionGroup.builder("system")
        .withNumPartitions(1)
        .withMembers("member-1", "member-2", "member-3")
        .build())
    .withPartitionGroups(
        RaftPartitionGroup.builder("raft")
            .withNumPartitions(3)
            .withPartitionSize(3)
            .withStorageLevel(StorageLevel.MAPPED)
            .withMembers("member-1", "member-2", "member-3")
            .build(),
        PrimaryBackupPartitionGroup.builder("data")
            .withNumPartitions(32)
            .withMemberGroupStrategy(MemberGroupStrategy.RACK_AWARE)
            .build())
    .build();

atomix.start().join();
```

Distributed Primitives

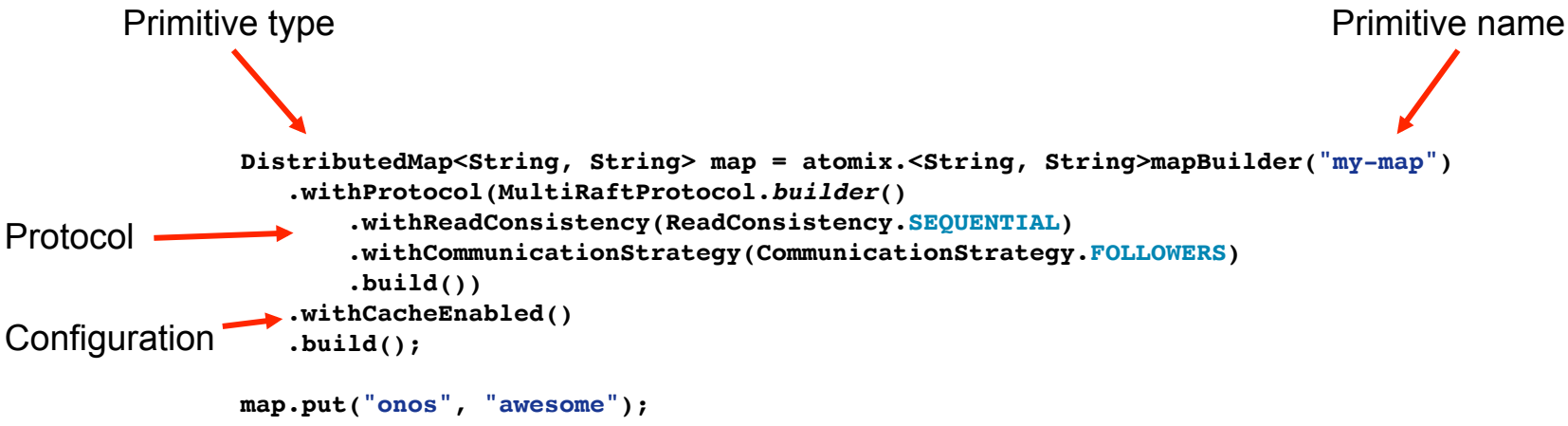
Distributed Primitives

- Interface to pre-defined replicated state machines
- Backed by a configurable protocol
- Stored on a chosen partition group

Distributed Primitives

```
DistributedMap<String, String> map = atomix.<String, String>mapBuilder("my-map")
   .withProtocol(MultiRaftProtocol.builder()
       .withReadConsistency(ReadConsistency.SEQUENTIAL)
       .withCommunicationStrategy(CommunicationStrategy.FOLLOWERS)
       .build())
   .withCacheEnabled()
   .build();

map.put("onos", "awesome");
```

Distributed Primitives



Distributed Primitives

Multi-Raft Protocol

```
DistributedMap<String, String> map = atomix.<String, String>mapBuilder("my-map")
   .withProtocol(MultiRaftProtocol.builder()
       .withReadConsistency(ReadConsistency.SEQUENTIAL)
       .withCommunicationStrategy(CommunicationStrategy.FOLLOWERS)
       .build())
   .withCacheEnabled()
   .build();

map.put("onos", "awesome");
```

Distributed Primitives

Multi-Primary Protocol

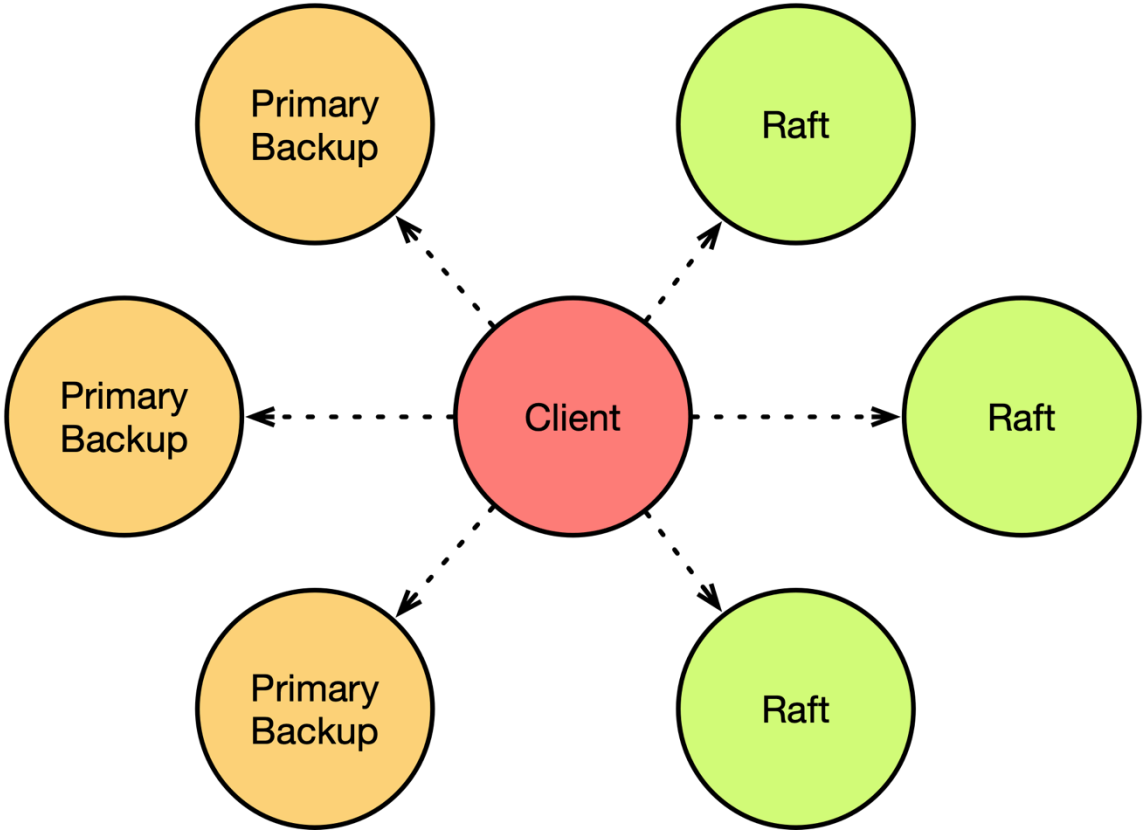
```
DistributedMap<String, String> map = atomix.<String, String>mapBuilder("my-map")
   .withProtocol(MultiPrimaryProtocol.builder()
       .withBackups(2)
       .withReplication(Replication.ASYNCHRONOUS)
       .build())
   .withCacheEnabled()
   .build();

map.put("onos", "awesome");
```

Distributed Primitives

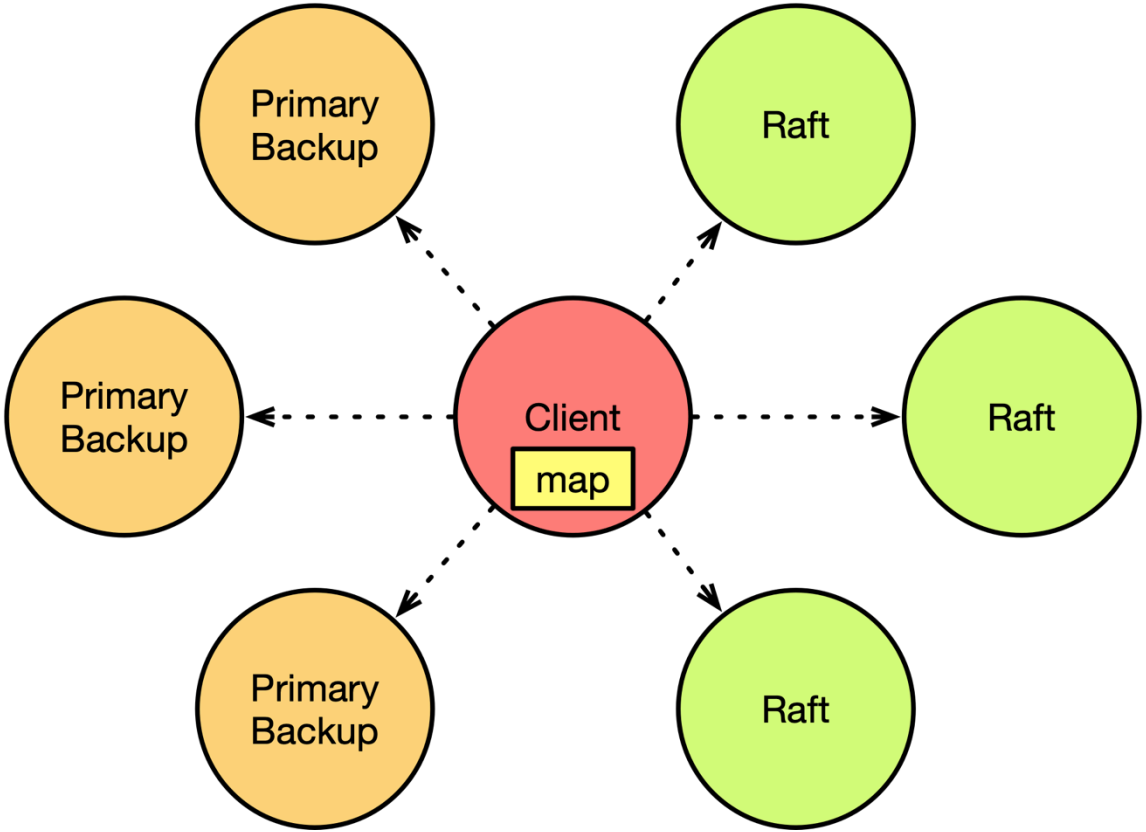
Distributed Log Protocol

```
DistributedMap<String, String> map = atomix.<String, String>mapBuilder("my-map")
   .withProtocol(DistributedLogProtocol.builder()
       .withReplication(Replication.ASYNCHRONOUS)
       .withRecovery(Recovery.RECOVER)
       .build())
   .withCacheEnabled()
   .build();

map.put("onos", "awesome");
```

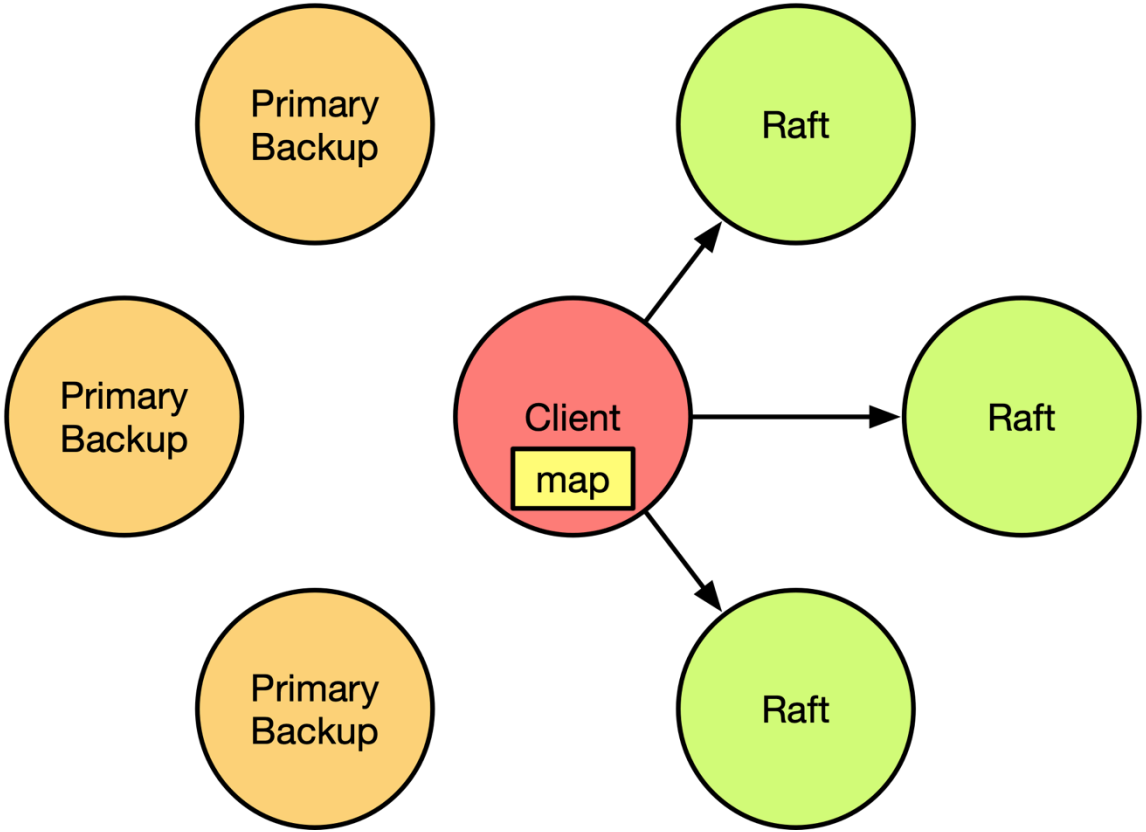
Distributed Primitives



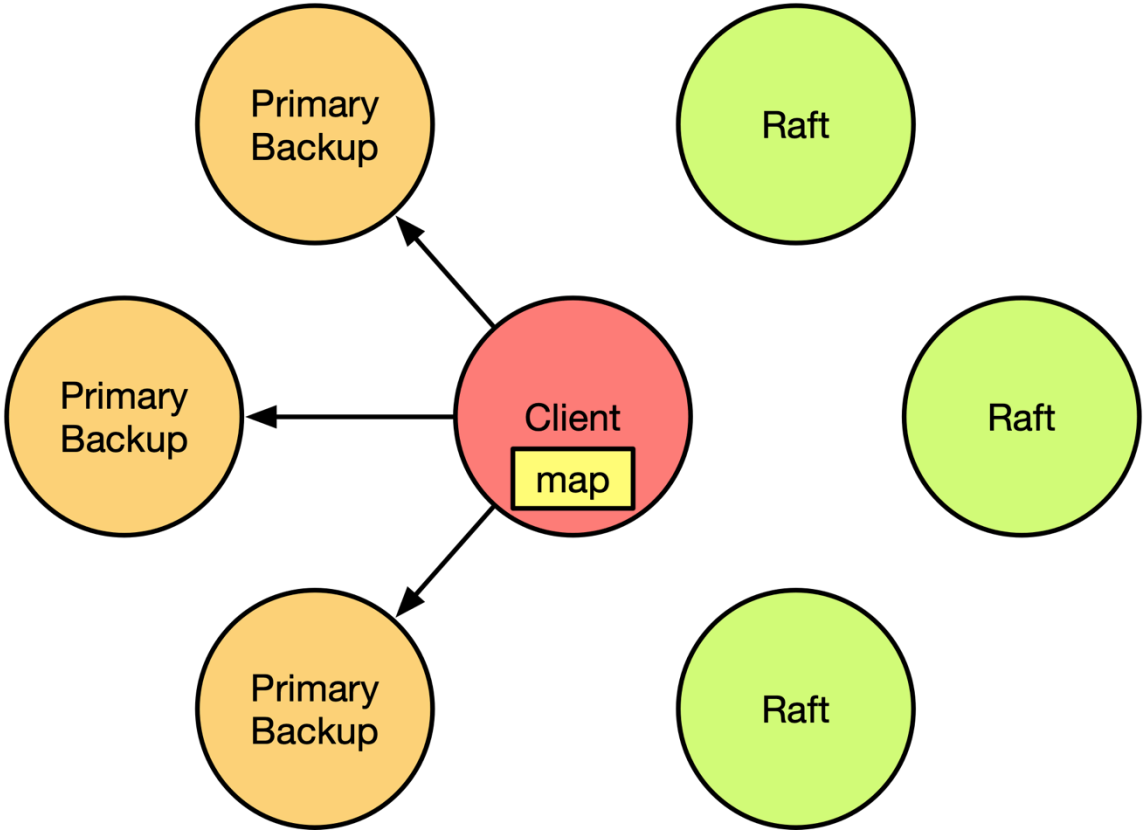
Distributed Primitives



Distributed Primitives



Distributed Primitives



Distributed Primitives

```
DistributedSet<String> set = atomix.<String>setBuilder("my-set")
    .withProtocol(MultiRaftProtocol.builder()
        .withReadConsistency(ReadConsistency.SEQUENTIAL)
        .withCommunicationStrategy(CommunicationStrategy.LEADER)
        .build())
    .build();

if (set.remove("foo")) {
  set.add("bar");
}
```

Distributed Primitives

```
AtomicCounter counter = atomix.atomicCounterBuilder("my-counter")
   .withProtocol(MultiRaftProtocol.builder()
       .withReadConsistency(ReadConsistency.LINEARIZABLE)
       .withCommunicationStrategy(CommunicationStrategy.LEADER)
       .build())
   .build();

long value = counter.incrementAndGet();
counter.compareAndSet(value, value + 1);
```

Distributed Primitives

```
// Create a distributed lock primitive.
DistributedLock lock = atomix.lockBuilder("my-lock")
    .withProtocol(MultiRaftProtocol.builder()
        .withMaxTimeout(Duration.ofSeconds(5))
        .withReadConsistency(ReadConsistency.LINEARIZABLE)
        .withCommunicationStrategy(CommunicationStrategy.LEADER)
        .build())
    .build();

// Acquire the lock then do some work and release it.
lock.lock();
try {
  doWork();
} finally {
  lock.unlock();
}
```

Distributed Primitives

```
// Create a leadership election.
LeaderElection<MemberId> election = atomix.<MemberId>leaderElectionBuilder("my-election")
    .withProtocol(MultiRaftProtocol.builder()
        .withMaxTimeout(Duration.ofSeconds(5))
        .withReadConsistency(ReadConsistency.LINEARIZABLE)
        .withCommunicationStrategy(CommunicationStrategy.LEADER)
        .build())
    .build();

// Get the local member identifier.
MemberId localMemberId = atomix.getMembershipService().getLocalMember().id();

// Run the local member ID for leadership.
Leadership<MemberId> leadership = election.run(localMemberId);

// Send a message to the current leader to do some work.
atomix.getCommunicationService().send("do-work", new Work(), leadership.leader().id())
    .whenCompleteAsync((response, error) -> {
      if (error == null) {
        LOGGER.info("Work complete!");
      }
    });
```

Distributed Primitives

```
LeaderElection<MemberId> election = atomix.<MemberId>leaderElectionBuilder("my-election")
    .withProtocol(MultiRaftProtocol.builder()
        .withMaxTimeout(Duration.ofSeconds(5))
        .withReadConsistency(ReadConsistency.LINEARIZABLE)
        .withCommunicationStrategy(CommunicationStrategy.LEADER)
        .build())
    .build();

election.addListener(event -> {
  MemberId newLeaderId = event.newLeadership().leader().id();
  LOGGER.info("A leadership change event occurred. New leader: {}", newLeaderId);
});
```

Distributed Primitives

```
DistributedLog<Entry> log = atomix.<Entry>logBuilder()
   .withProtocol(DistributedLogProtocol.builder()
       .withConsistency(Consistency.SEQUENTIAL)
       .withRecovery(Recovery.RECOVER)
       .build())
   .build();

log.produce(new Entry("Hello world!"));

log.consume(record -> {
 Entry entry = record.value();
 LOGGER.info("Entry {} was appended to the log", entry);
});
```

Distributed Primitives

```
DistributedMap<String, String> map = atomix.<String, String>mapBuilder("my-map")
   .withProtocol(MultiRaftProtocol.builder()
       .withReadConsistency(ReadConsistency.SEQUENTIAL)
       .withCommunicationStrategy(CommunicationStrategy.FOLLOWERS)
       .build())
   .withCacheEnabled()
   .build();

map.put("onos", "awesome");

AsyncDistributedMap<String, String> asyncMap = map.async();

asyncMap.put("onos", "awesome").thenRun(() -> LOGGER.info("Write complete"));
```

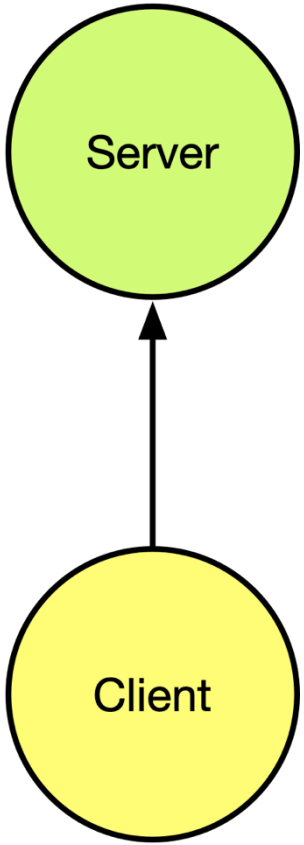
Deployment

Agent

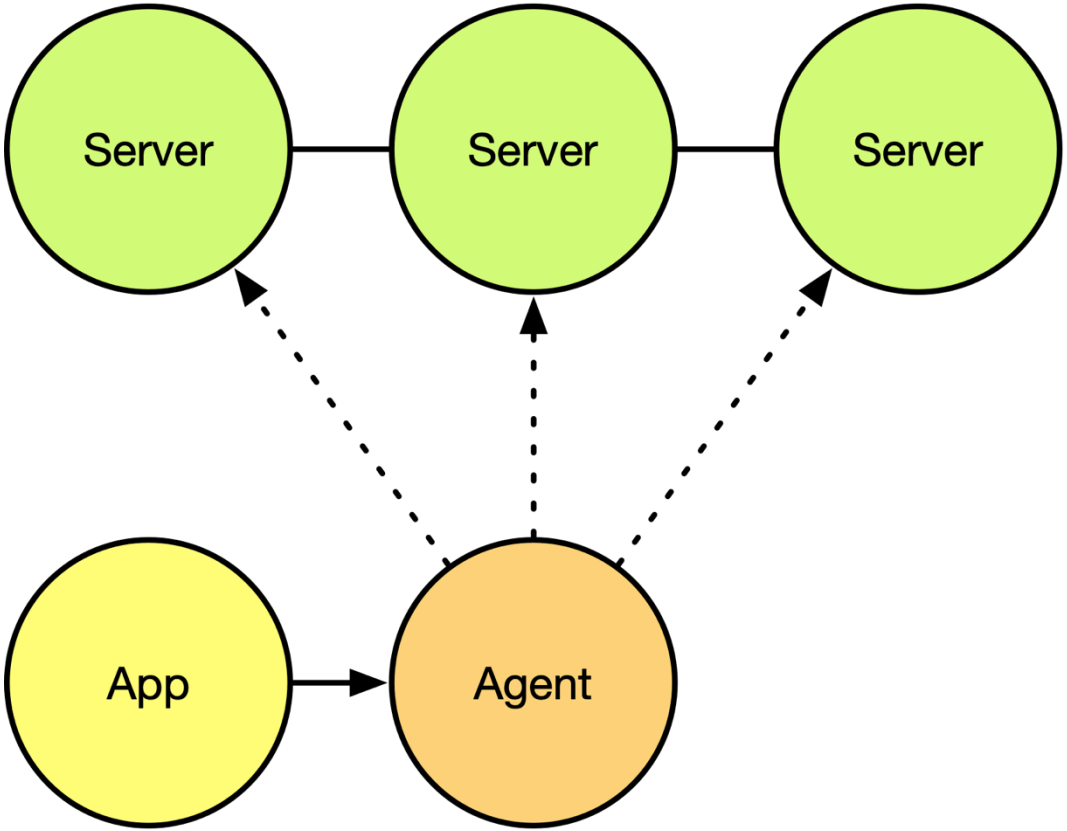
```
./bin/atomix-agent -c atomix.conf -m member-1 -a 192.168.20.1:5679
```

Cluster Architecture

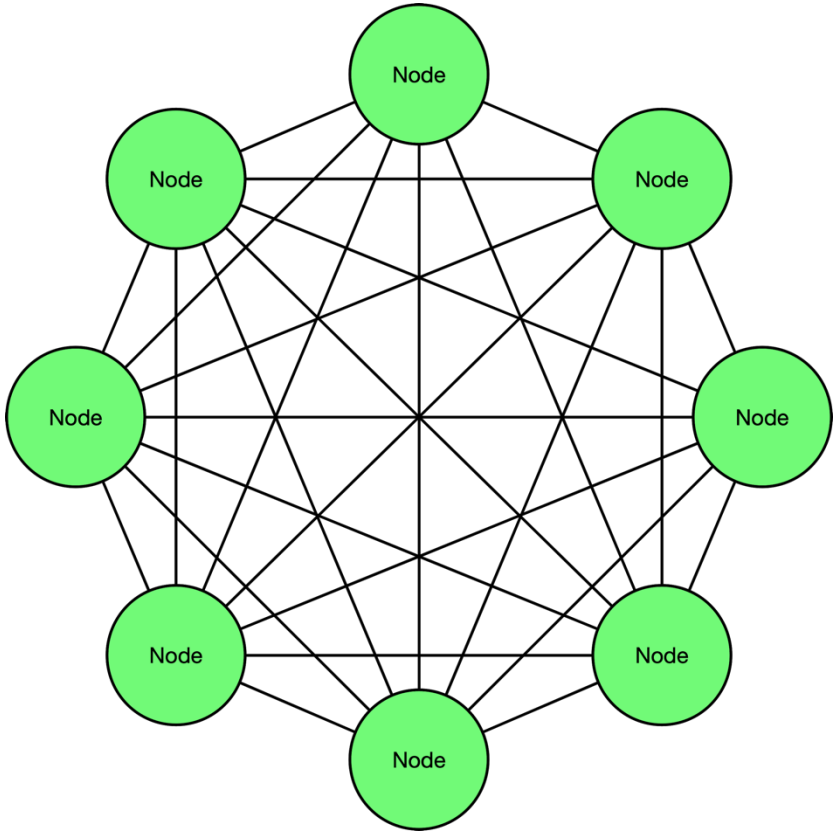
Cluster Architecture



Cluster Architecture

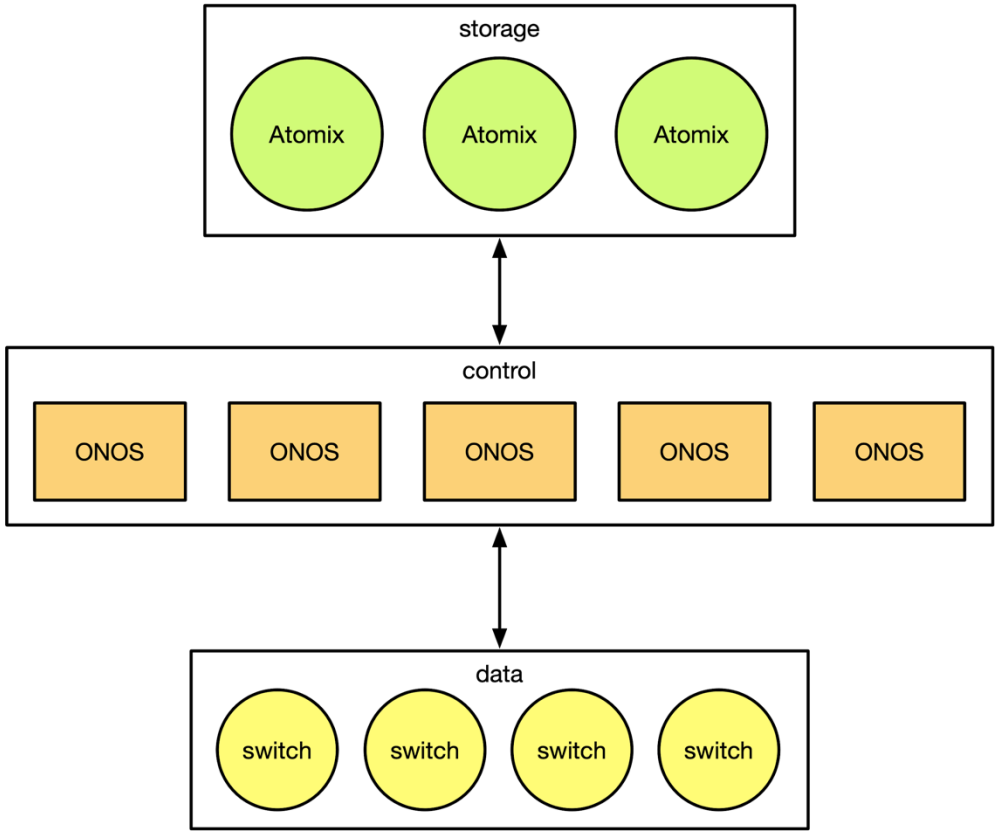


Cluster Architecture

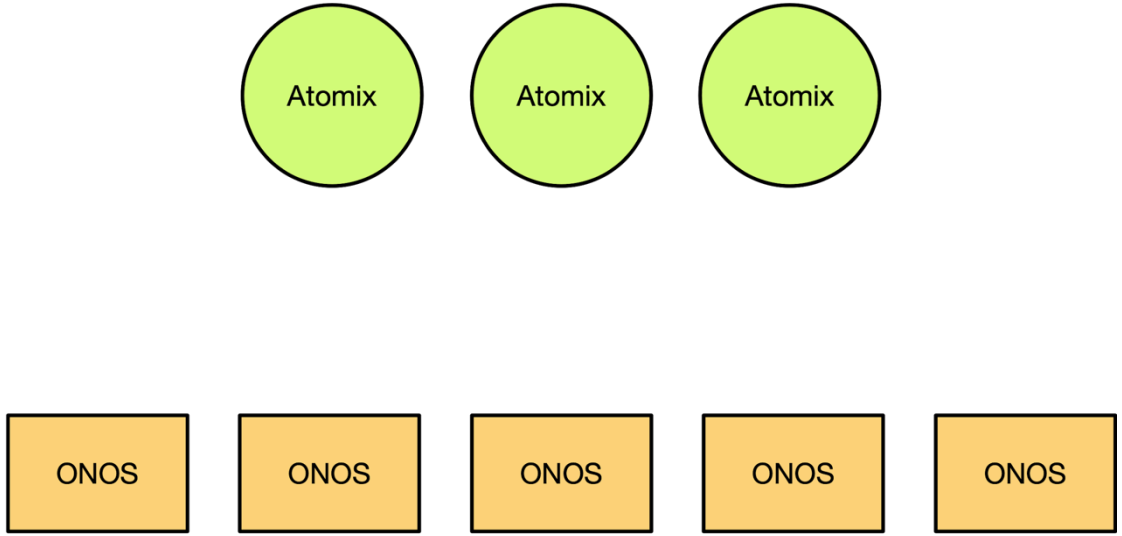


Owl Cluster Architecture

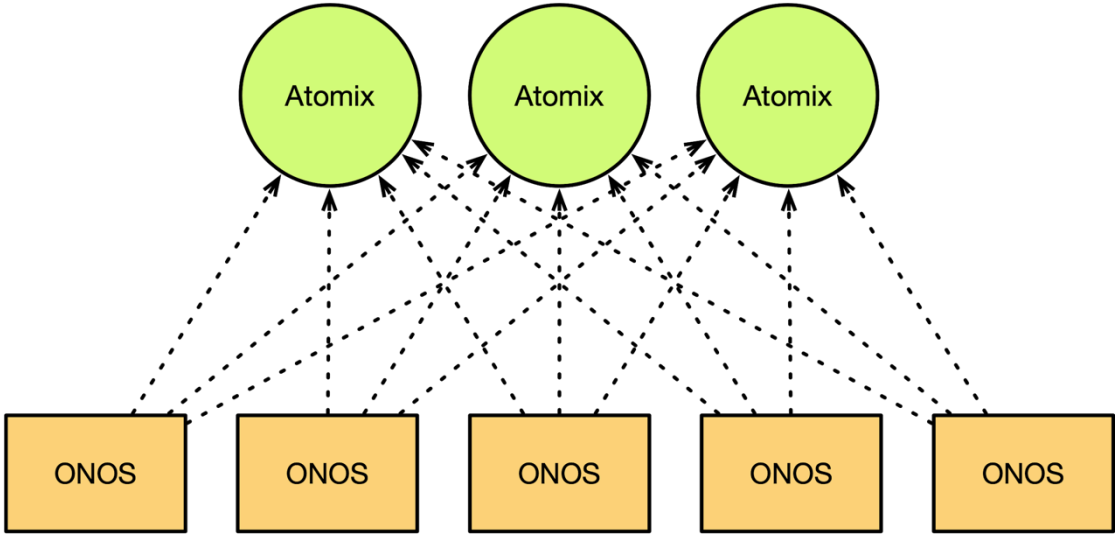
Owl Cluster Architecture



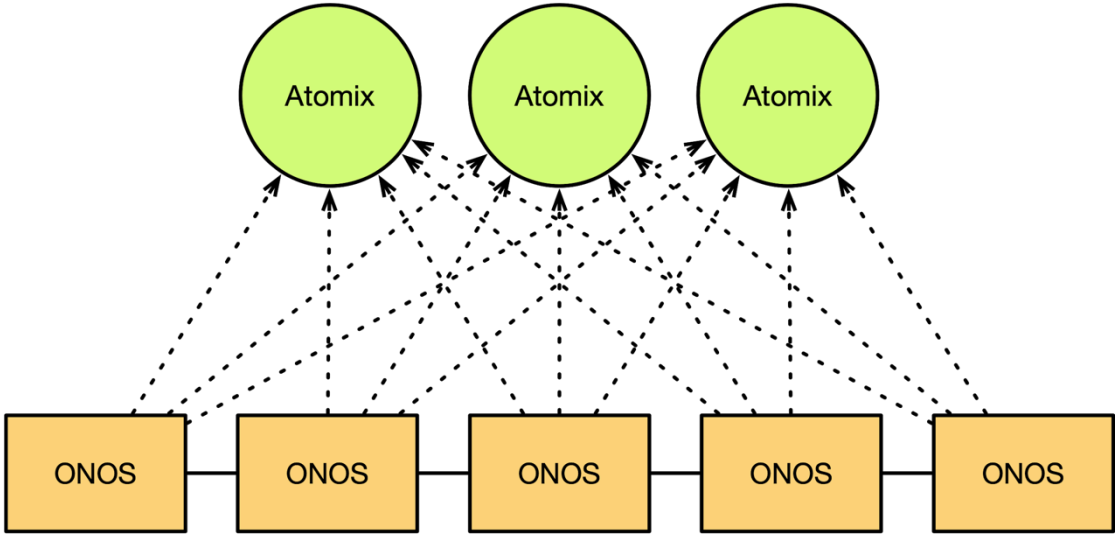
Owl Cluster Architecture



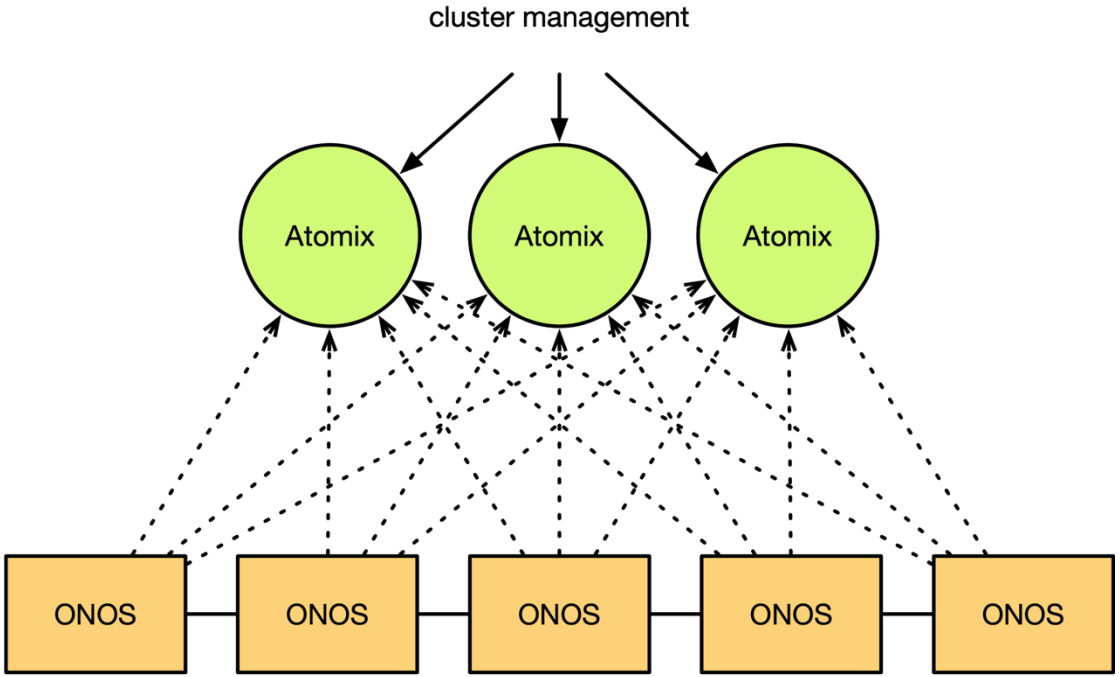
Owl Cluster Architecture



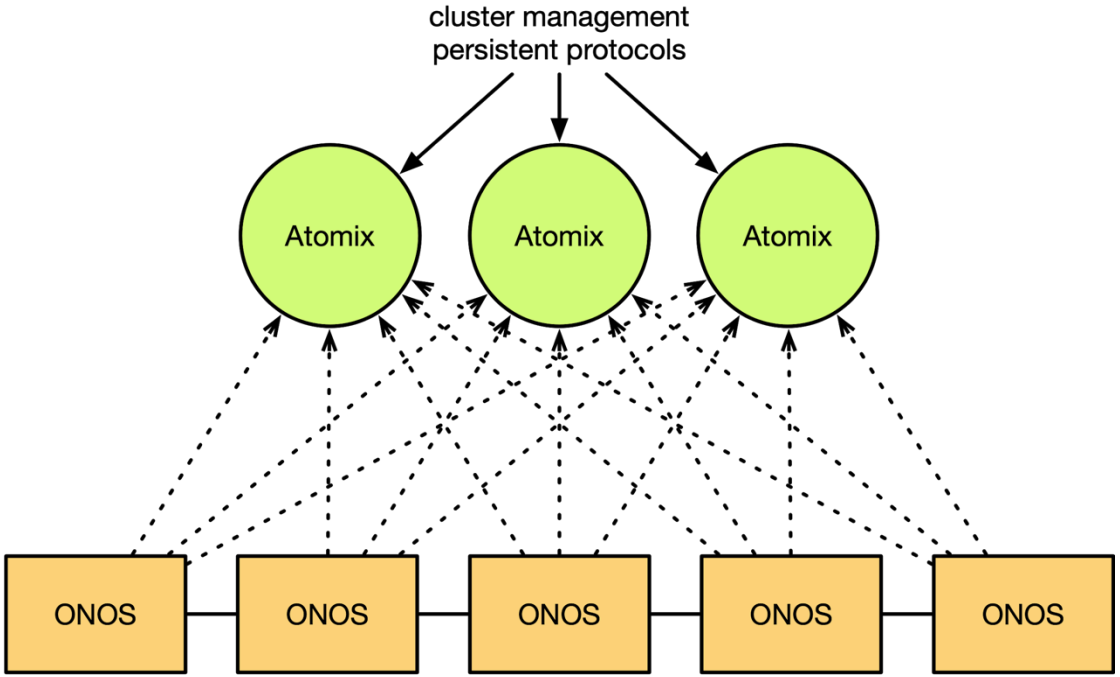
Owl Cluster Architecture



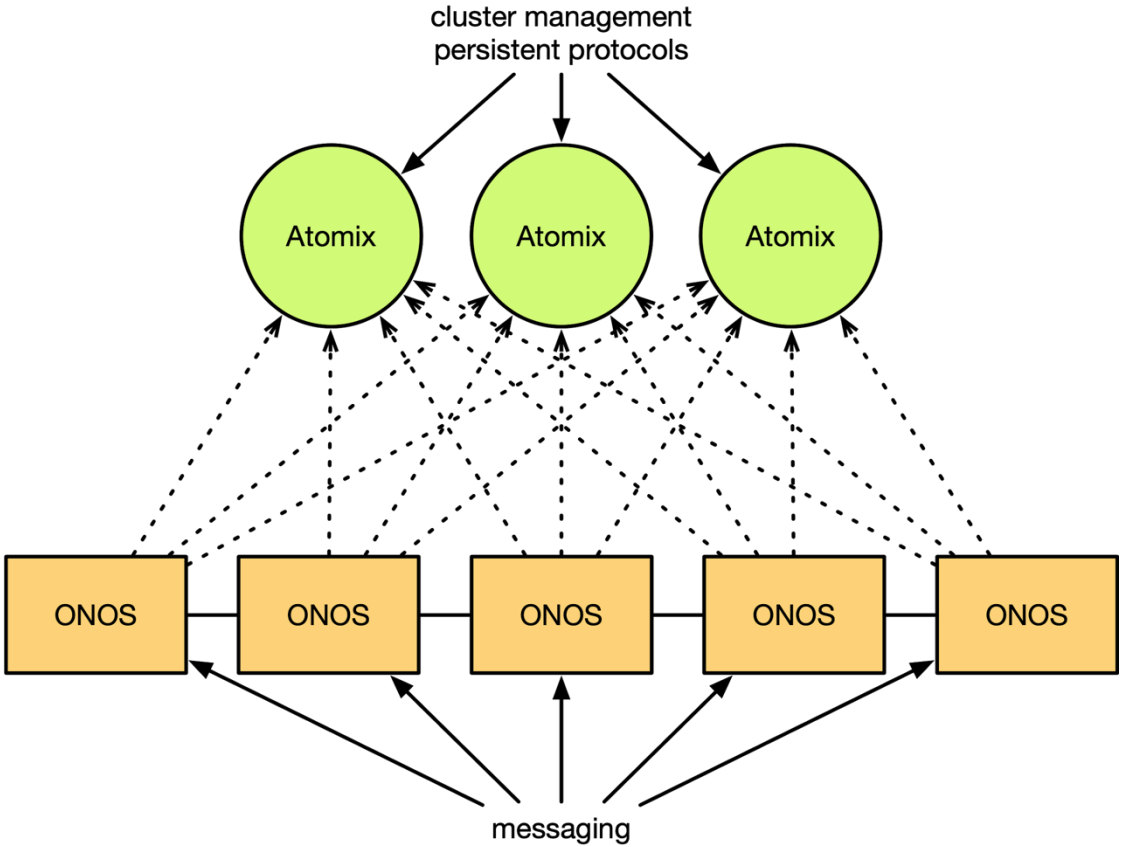
Owl Cluster Architecture



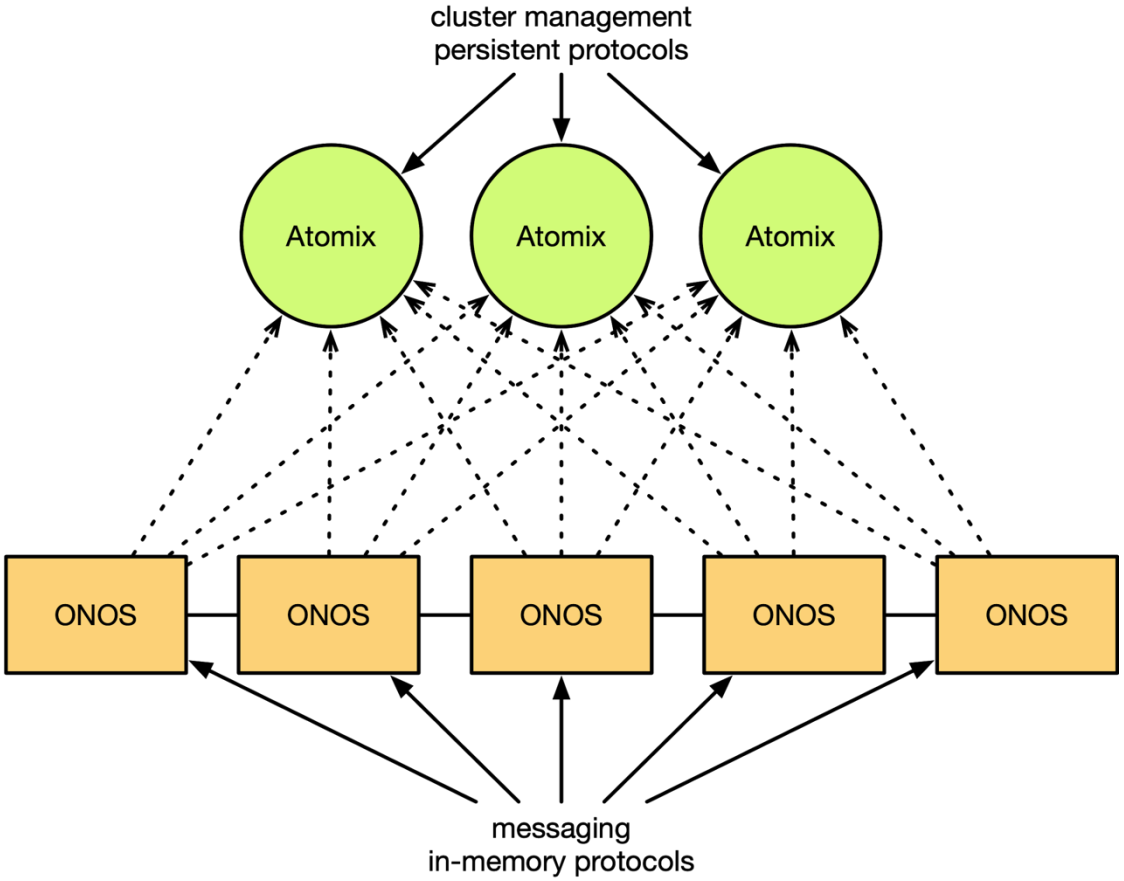
Owl Cluster Architecture



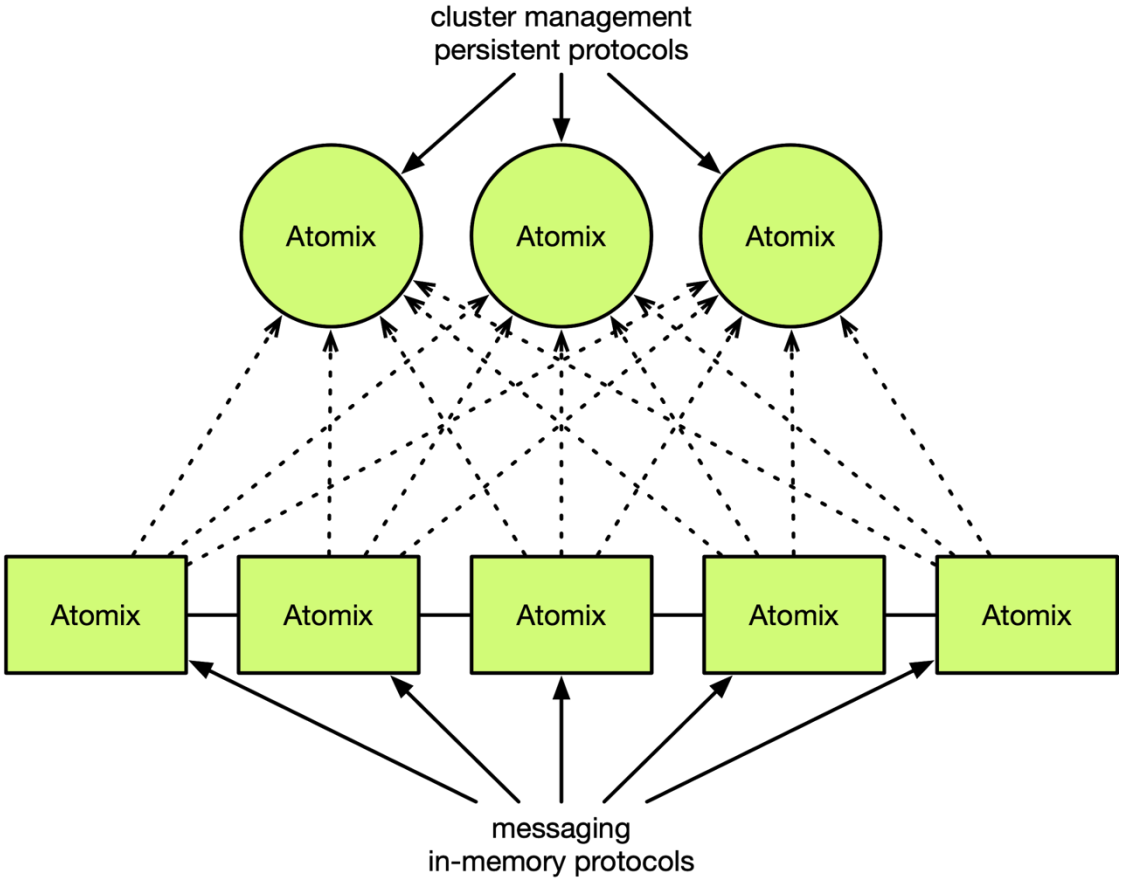
Owl Cluster Architecture



Owl Cluster Architecture



Owl Cluster Architecture



Owl Cluster Architecture

```
{
 "nodes": [
   {
     "id": "onos-1",
     "ip": "192.168.20.1",
     "port": 9876
   },
   {
     "id": "onos-2",
     "ip": "192.168.20.2",
     "port": 9876
   },
   {
     "id": "onos-3",
     "ip": "192.168.20.3",
     "port": 9876
   }
 ]
}
```

Owl Cluster Architecture


```
{
 "partitions": [
   {
     "id": 1,
     "members": [
       "onos-1",
       "onos-2",
       "onos-3"
     ]
   },
   {
     "id": 2,
     "members": [
       "onos-1",
       "onos-2",
       "onos-3"
     ]
   },
   {
     "id": 3,
     "members": [
       "onos-1",
       "onos-2",
       "onos-3"
     ]
   }
 ]
}
```

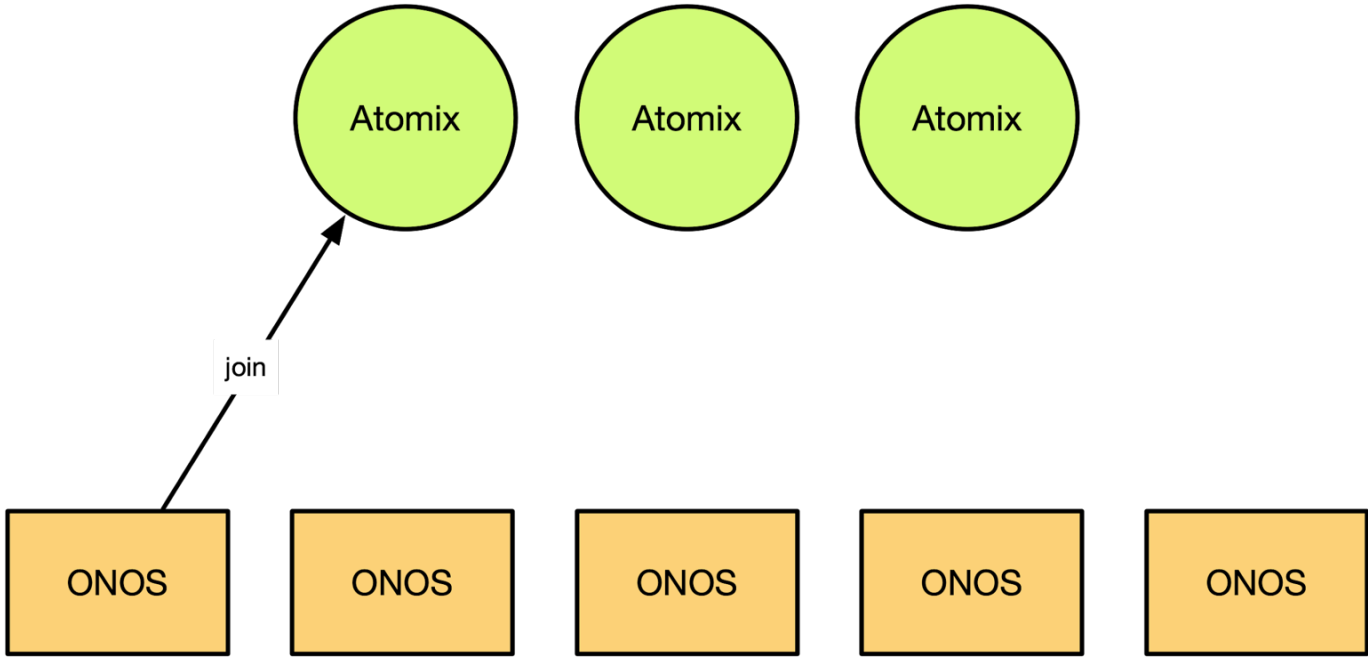
Owl Cluster Architecture

```
cluster {
 node {
   id: ${atomix.node.id}
   host: ${atomix.node.host}
 }

 discovery {
   type: bootstrap
   # ...
 }

 protocol {
   type: swim
   # ...
 }
}

managementGroup {
 type: raft
 partitions: 1
 members: [atomix-1, atomix-2, atomix-3]
}

partitionGroups.raft {
 type: raft
 partitions: 3
 storage.level: mapped
 members: [atomix-1, atomix-2, atomix-3]
}
```

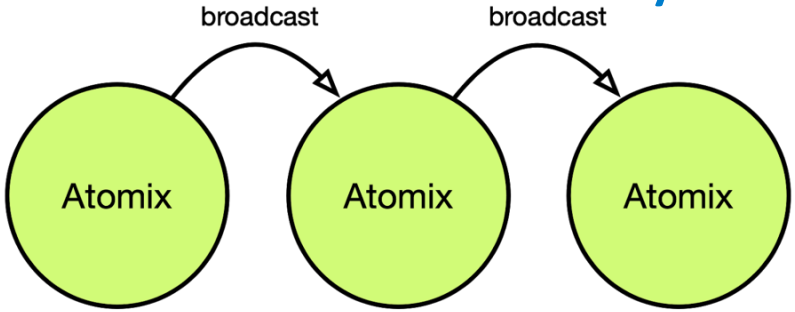
Owl Cluster Architecture

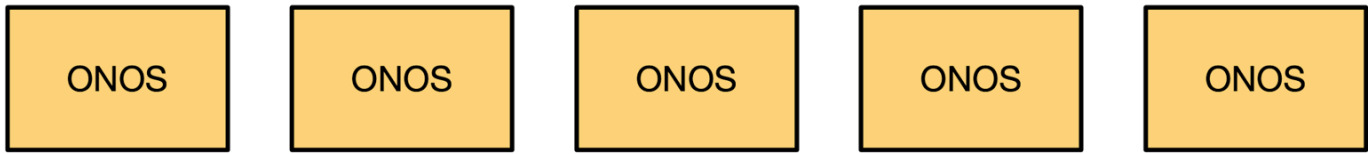
```
{
 "name": "onos",
 "node": {
   "id": "onos-1",
   "host": "192.168.20.1",
   "port": "9876"
 },
 "storage": [
   {
     "id": "atomix-1",
     "ip": "192.168.10.1",
     "port": 5679
   },
   {
     "id": "atomix-2",
     "ip": "192.168.10.2",
     "port": 5679
   },
   {
     "id": "atomix-3",
     "ip": "192.168.10.3",
     "port": 5679
   }
 ]
}
```

Service Discovery

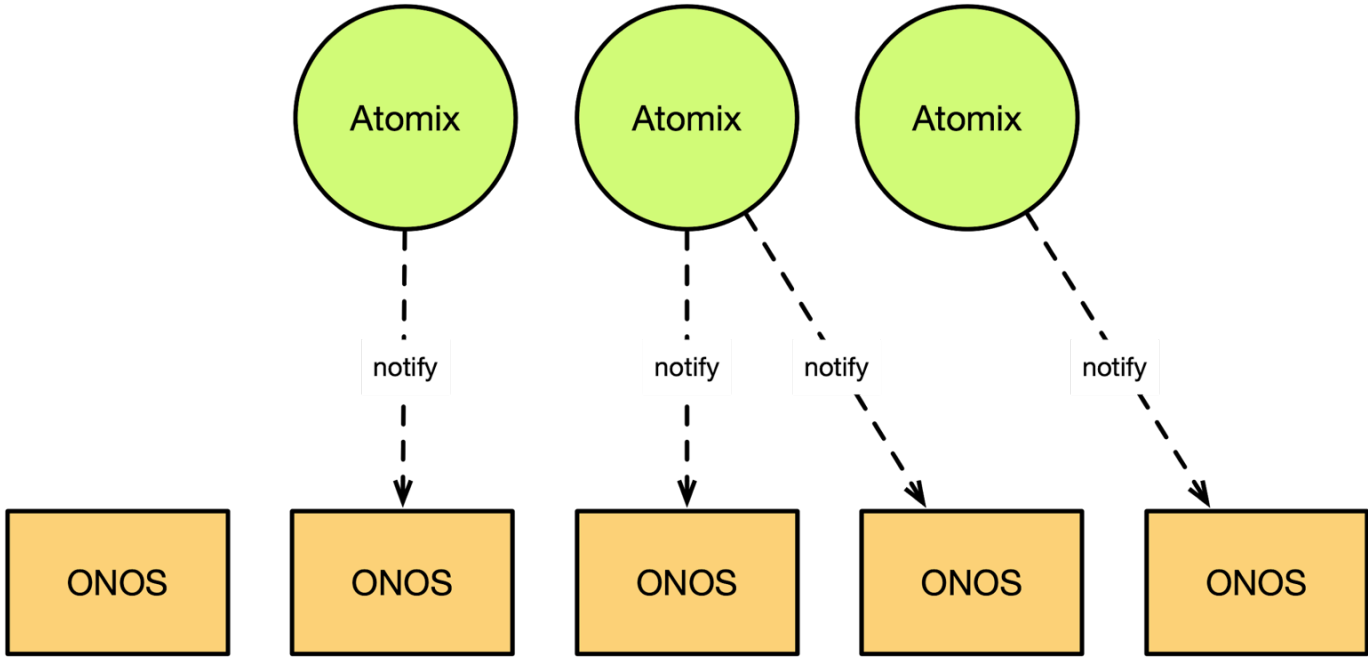


Service Discovery

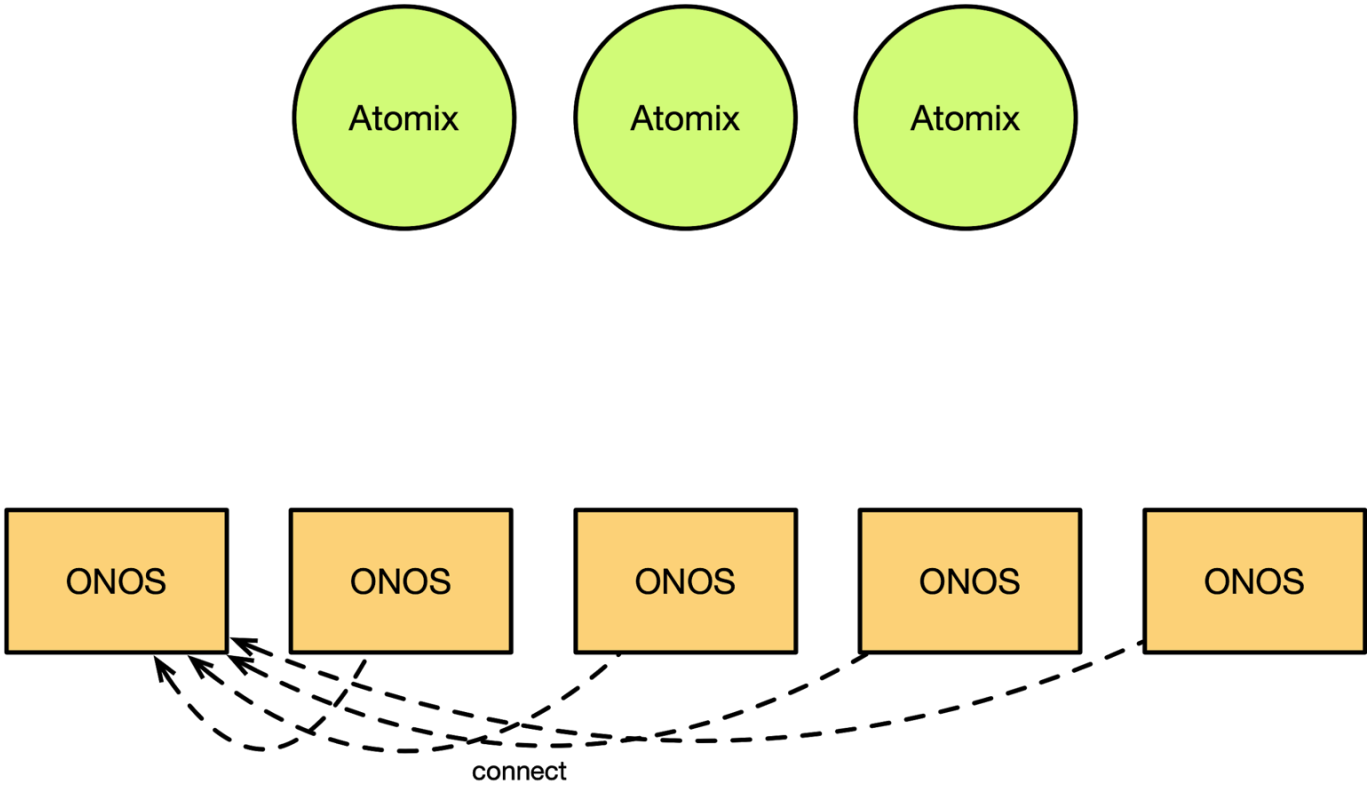




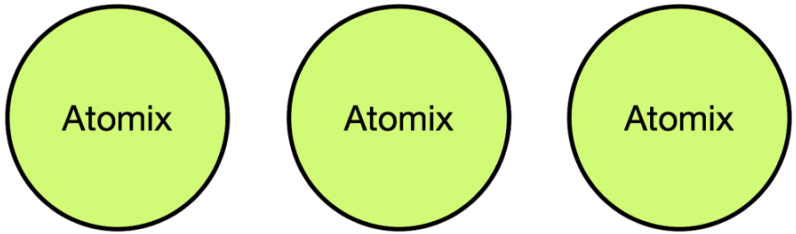
Service Discovery

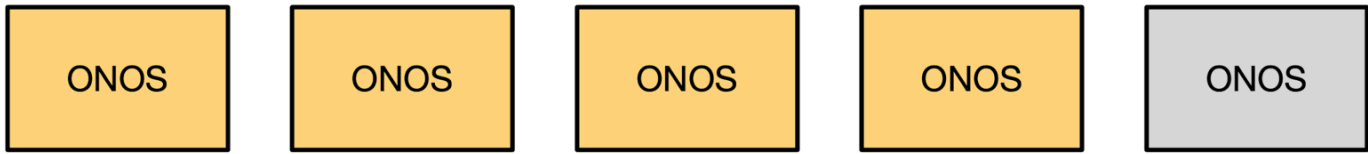


Service Discovery

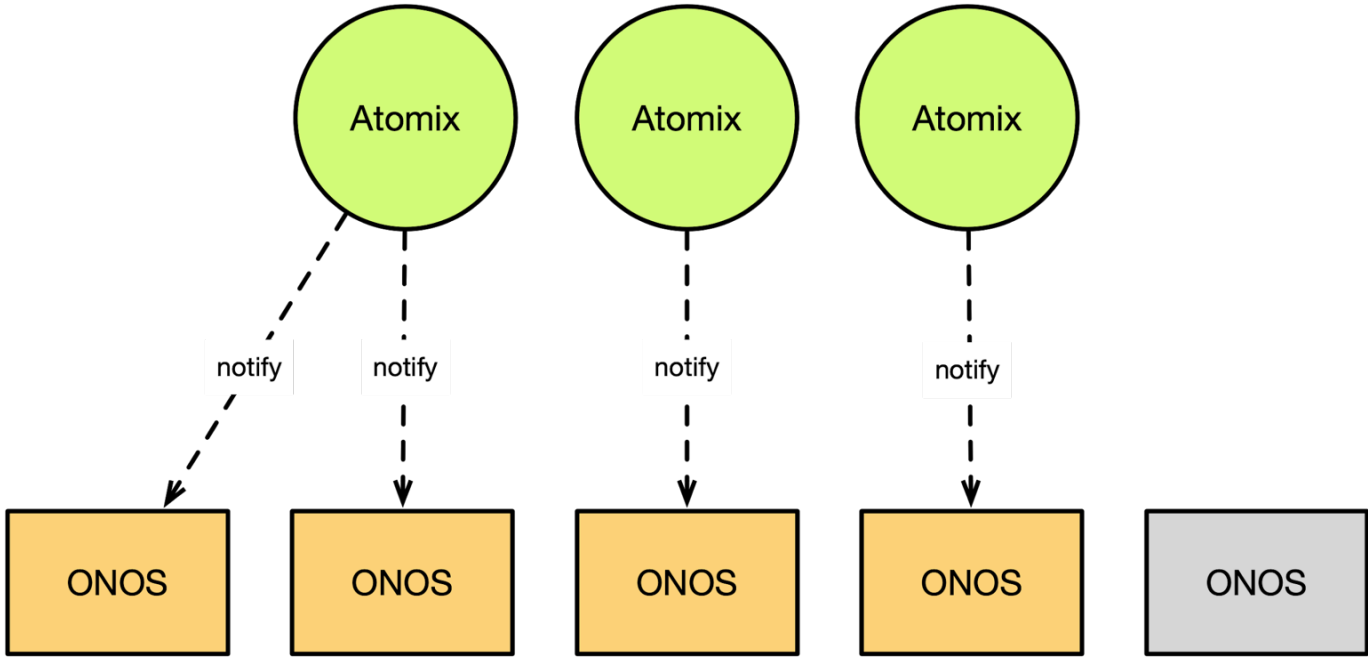


Service Discovery

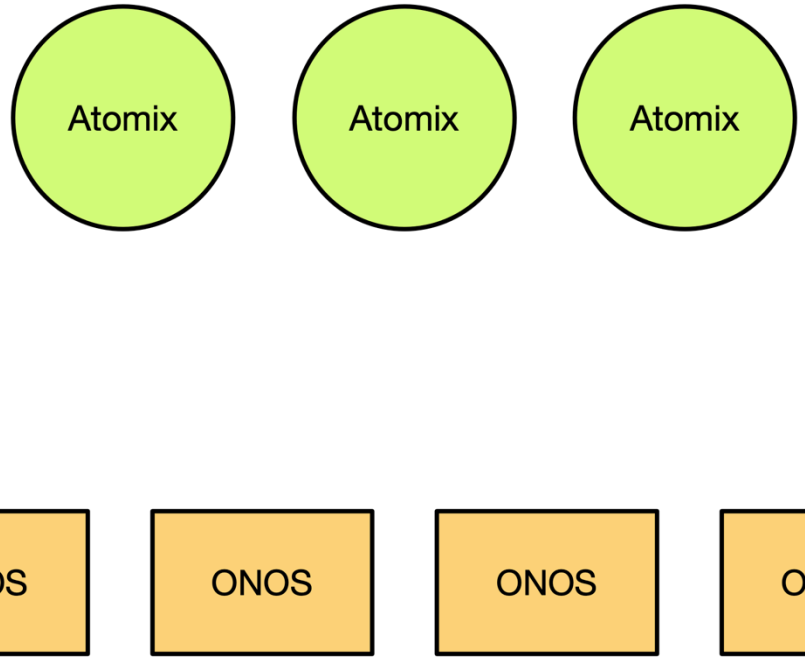




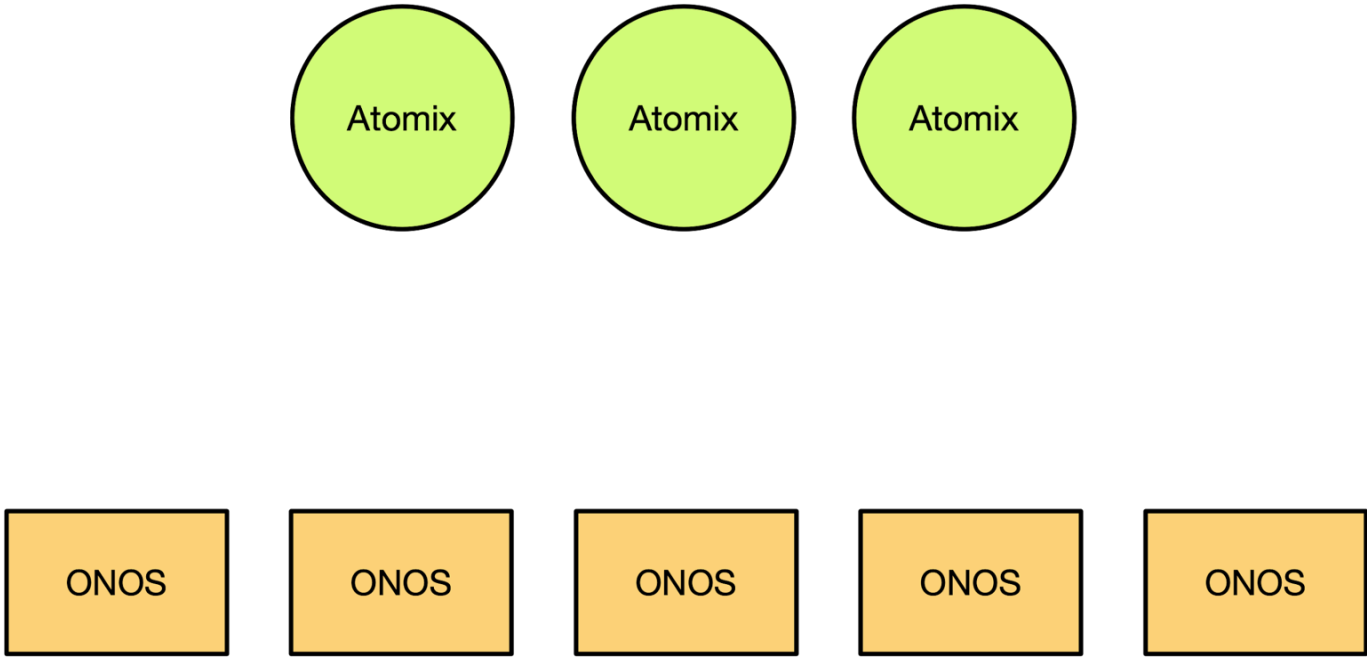
Service Discovery



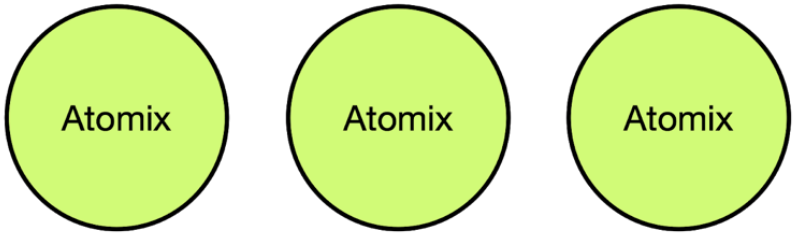
Service Discovery

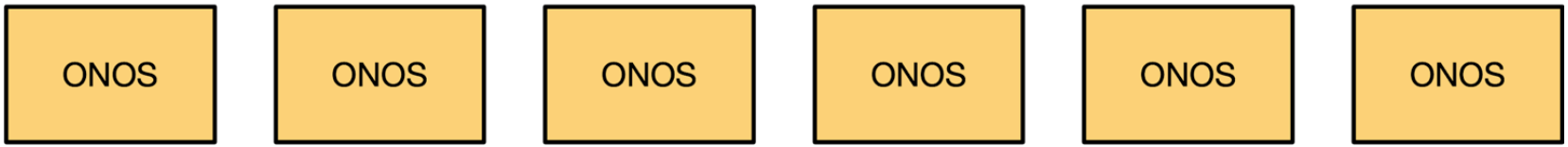


Scaling

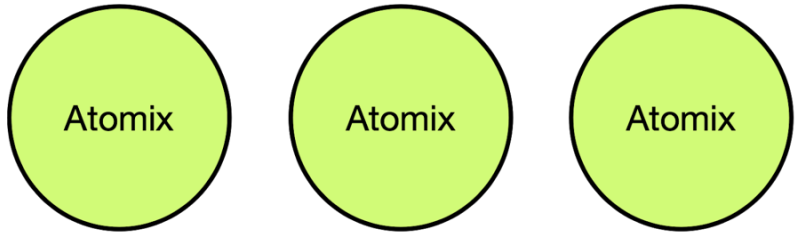


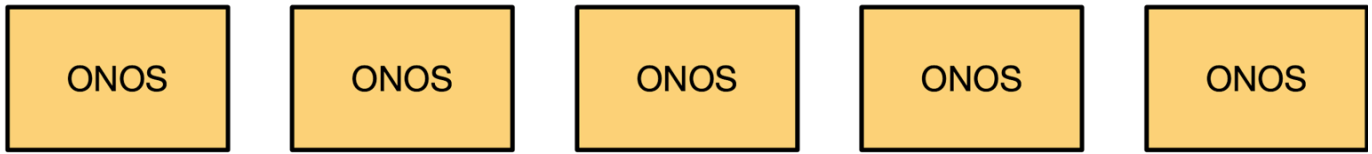
Scaling



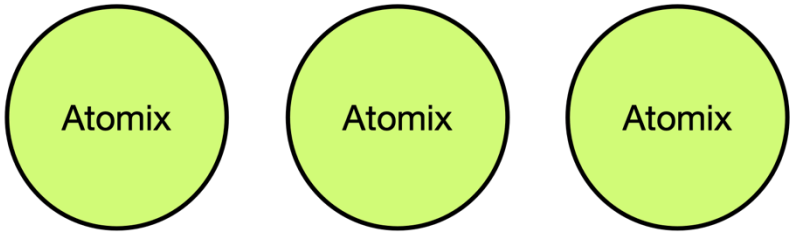


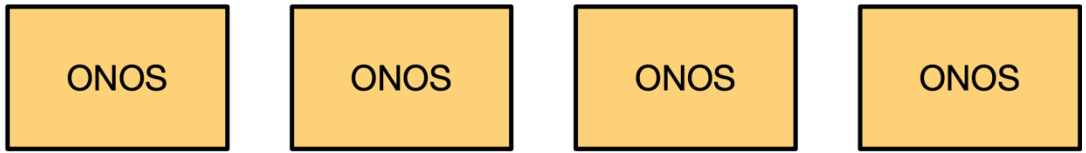
Scaling



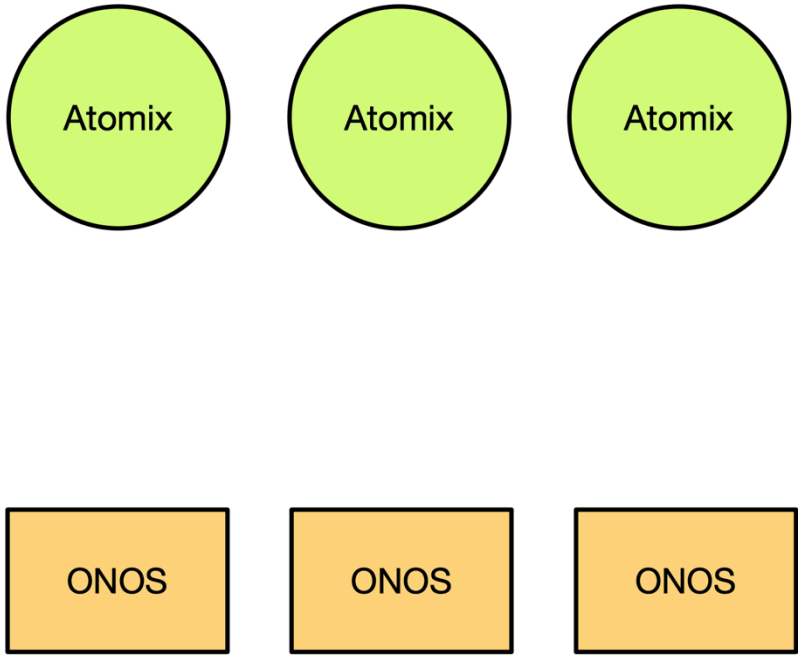


Scaling

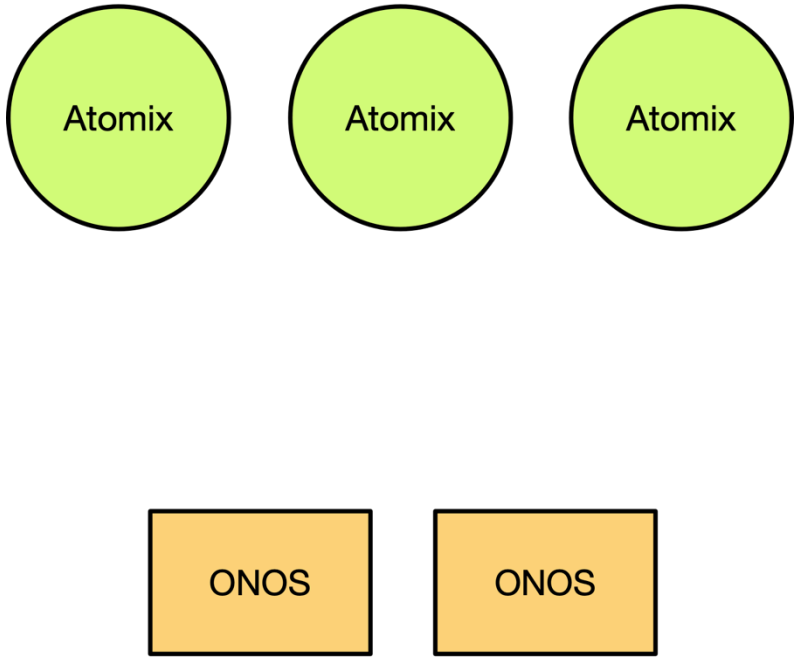




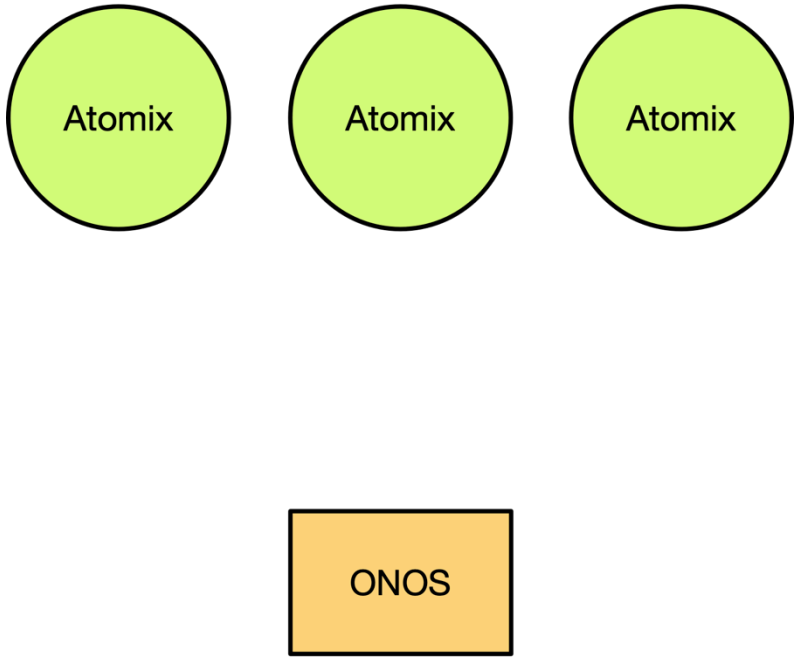
Scaling



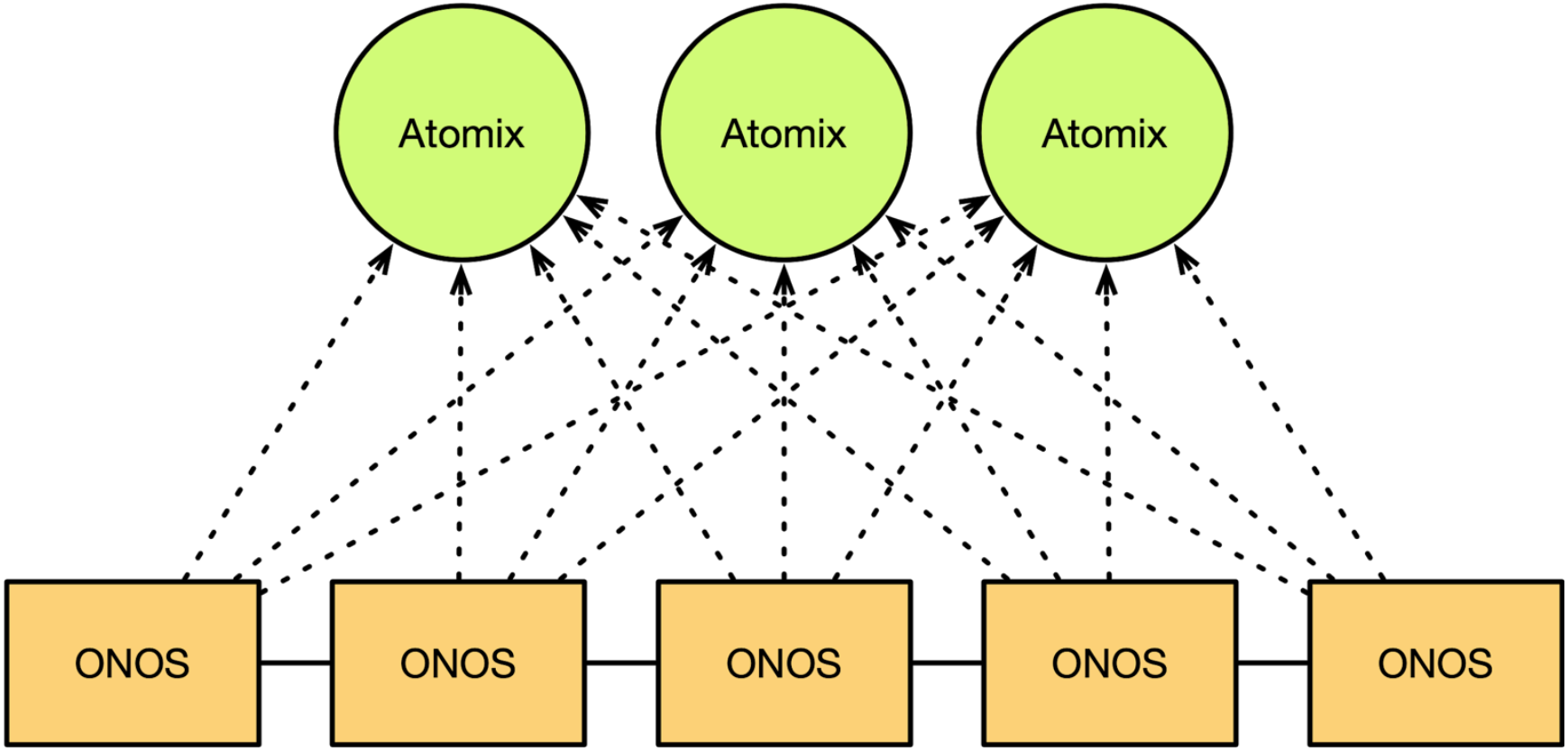
Scaling



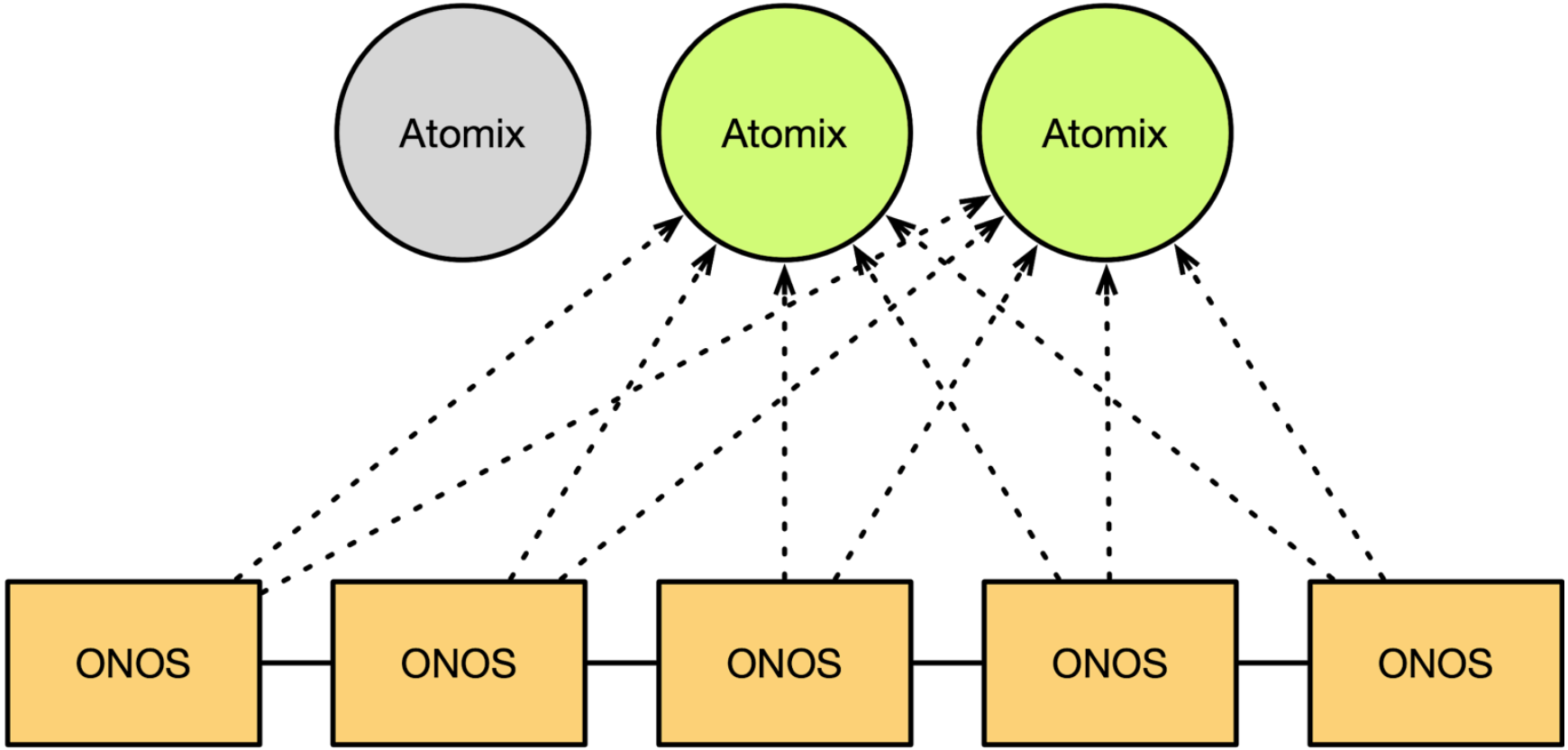
Scaling



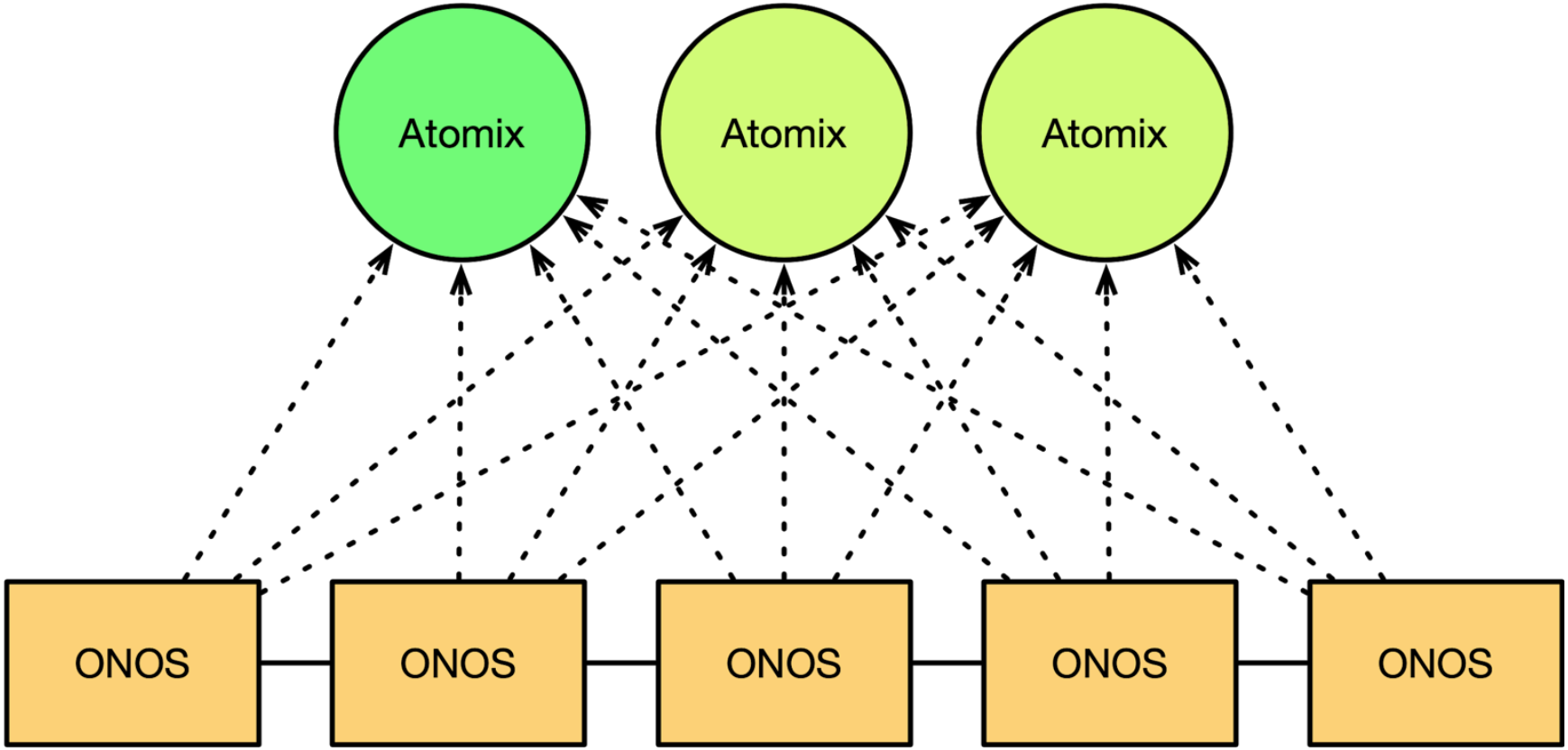
Rolling Upgrades



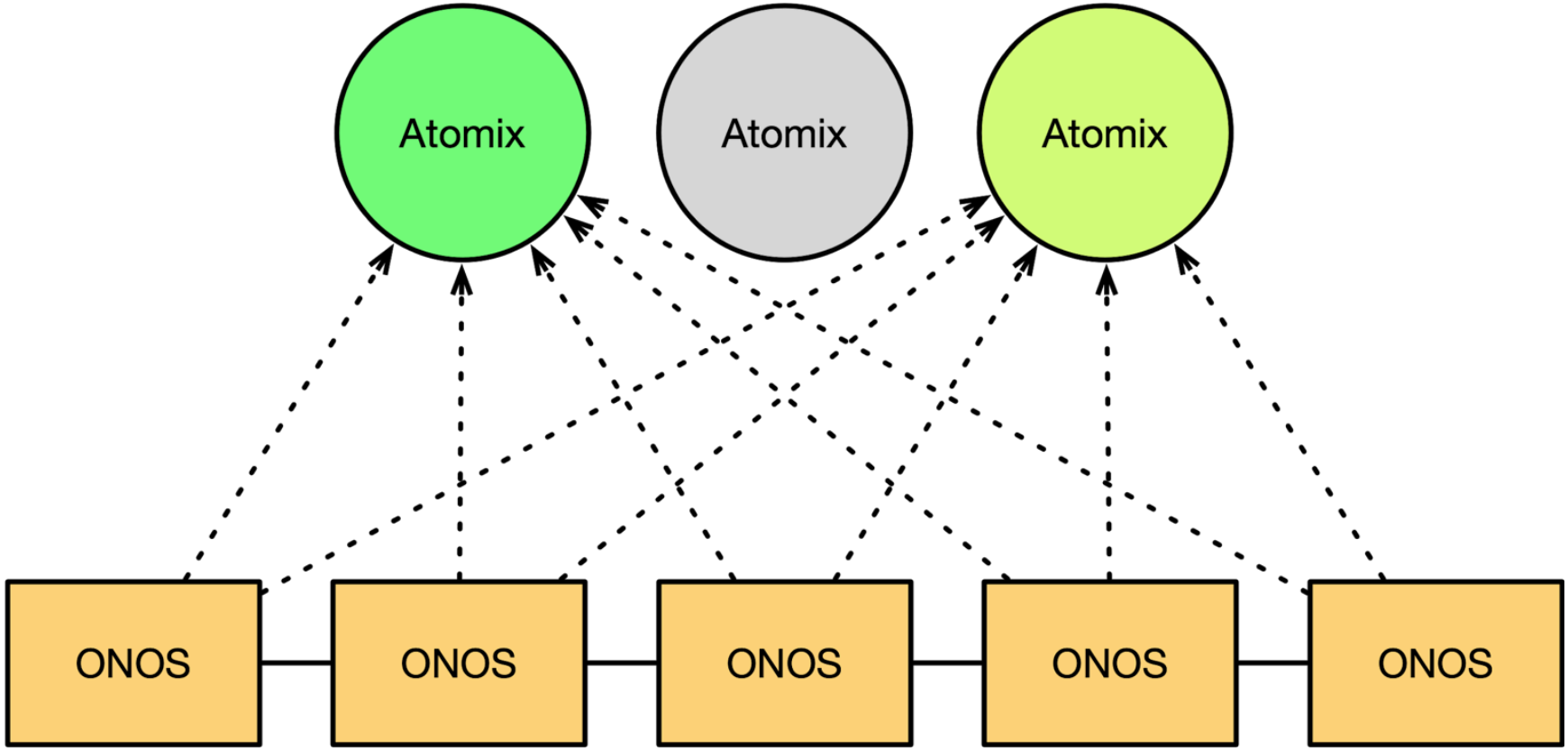
Rolling Upgrades



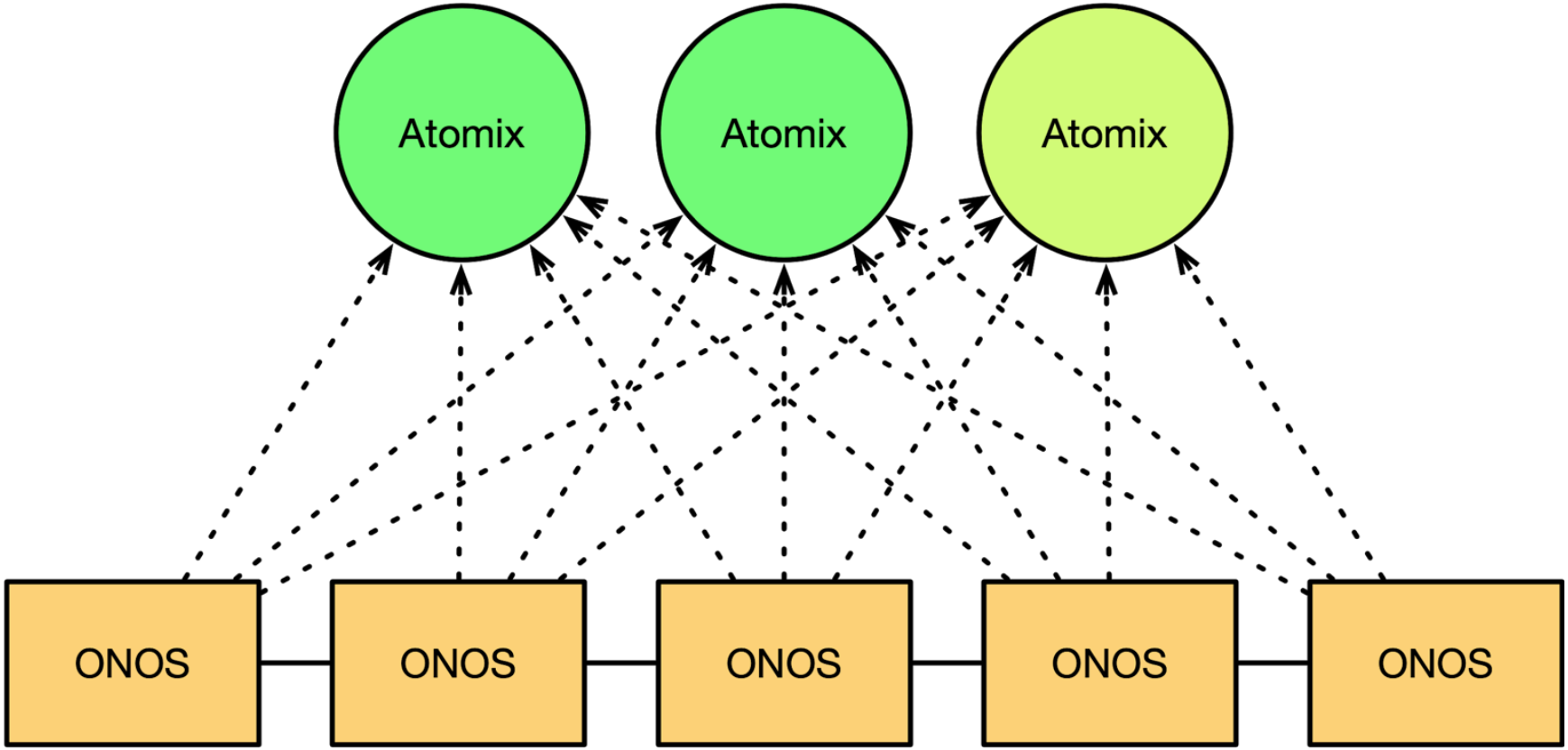
Rolling Upgrades



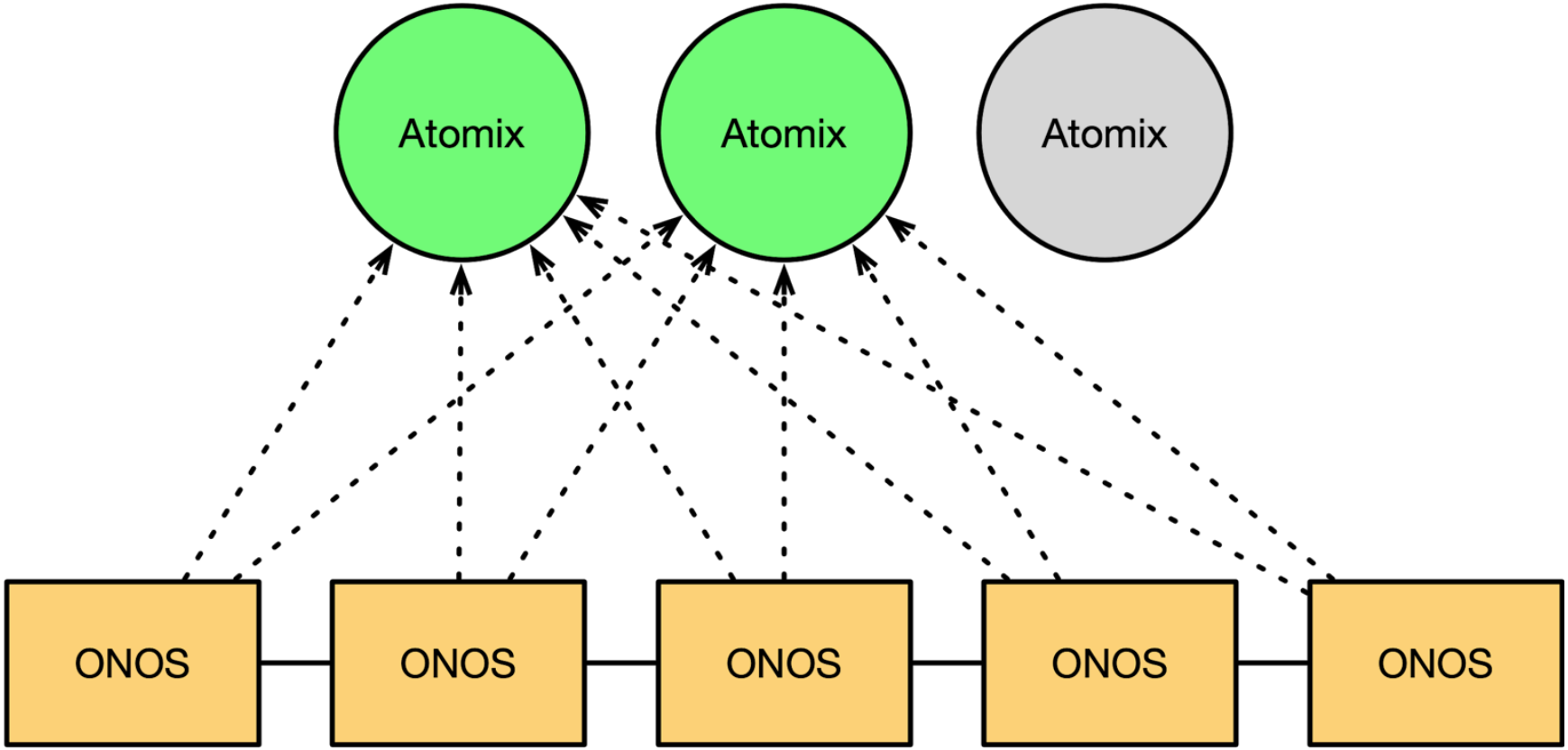
Rolling Upgrades



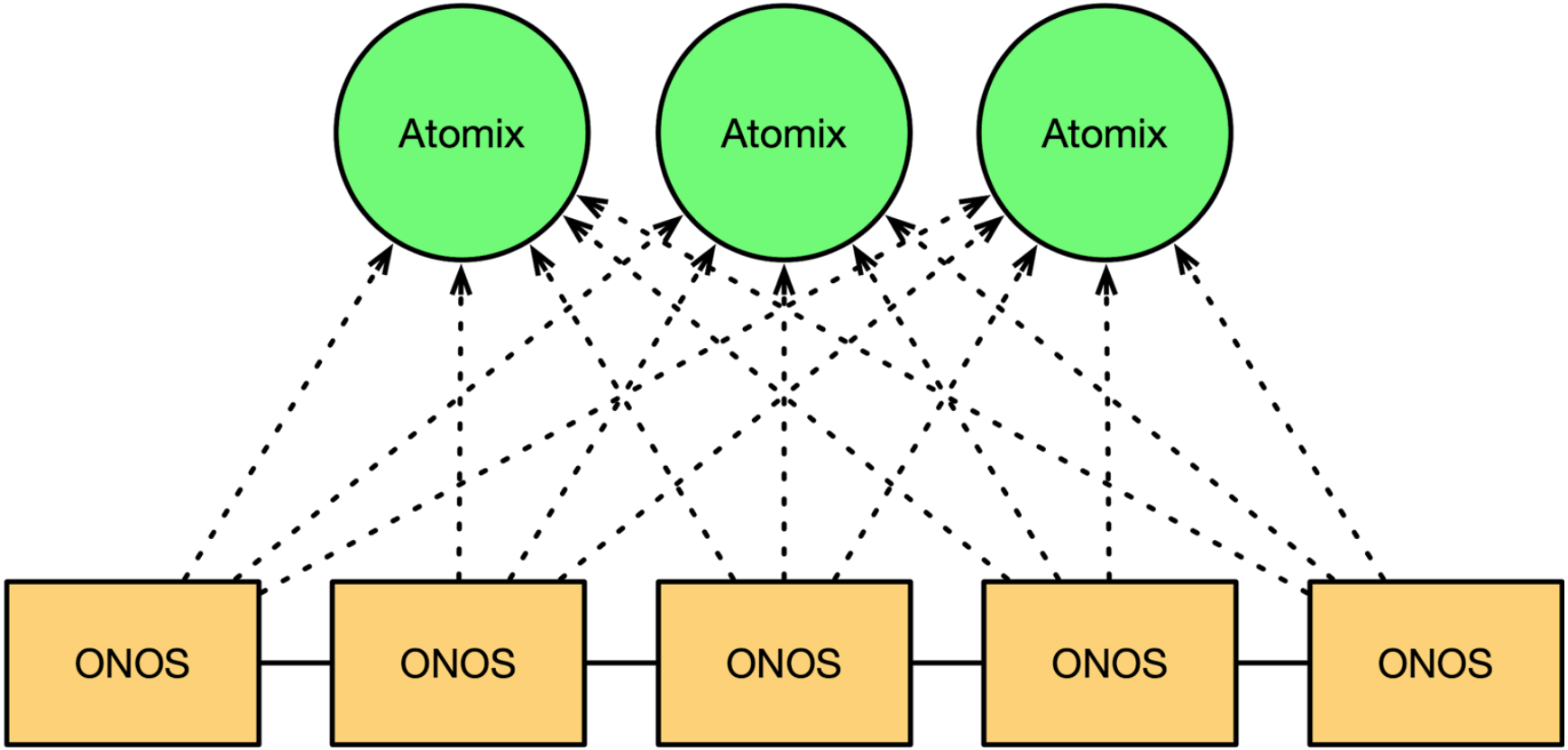
Rolling Upgrades



Rolling Upgrades



Rolling Upgrades



Compatibility

- ONOS 1.14 and 1.15 compatible with any Atomix 3.0.x release
- ONOS 2.0 compatible with any Atomix 3.1 release
- Experimental REST API only accessible via Atomix agent
- Primitive protocols not currently exposed in ONOS API

Resources

- ONOS Website: https://onosproject.org
- ONOS GitHub: https://github.com/opennetworkinglab/onos
- Atomix Website: https://atomix.io
- Atomix GitHub: https://github.com/atomix/atomix