Explicitly Supporting Stretch Clusters in Ceph

Greg Farnum

FOSDEM 2020

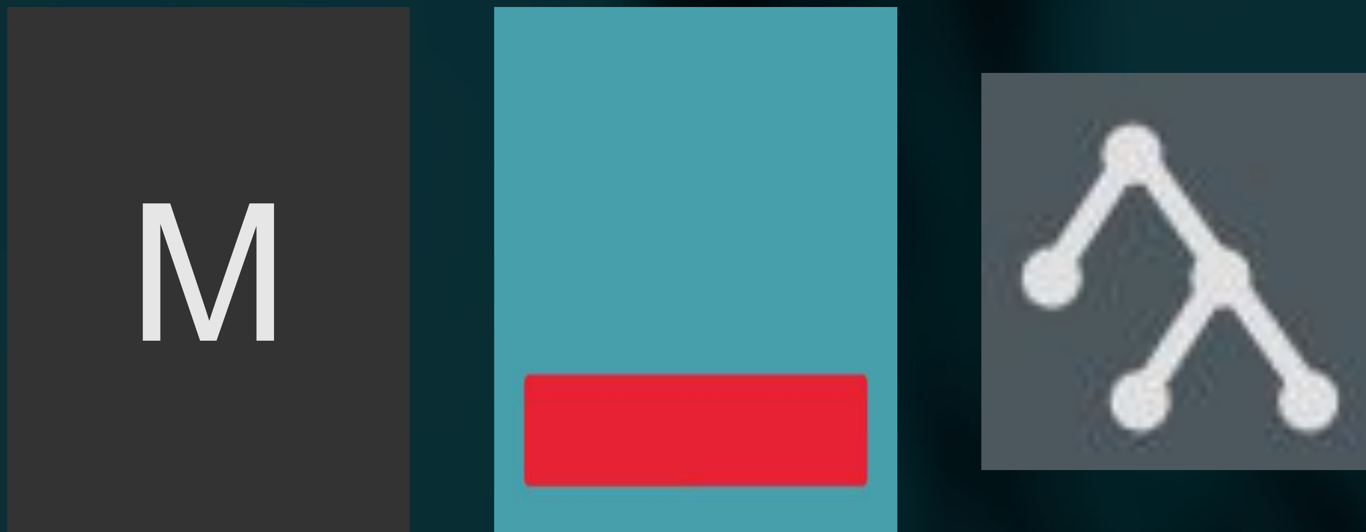
ceph

I'M GREG

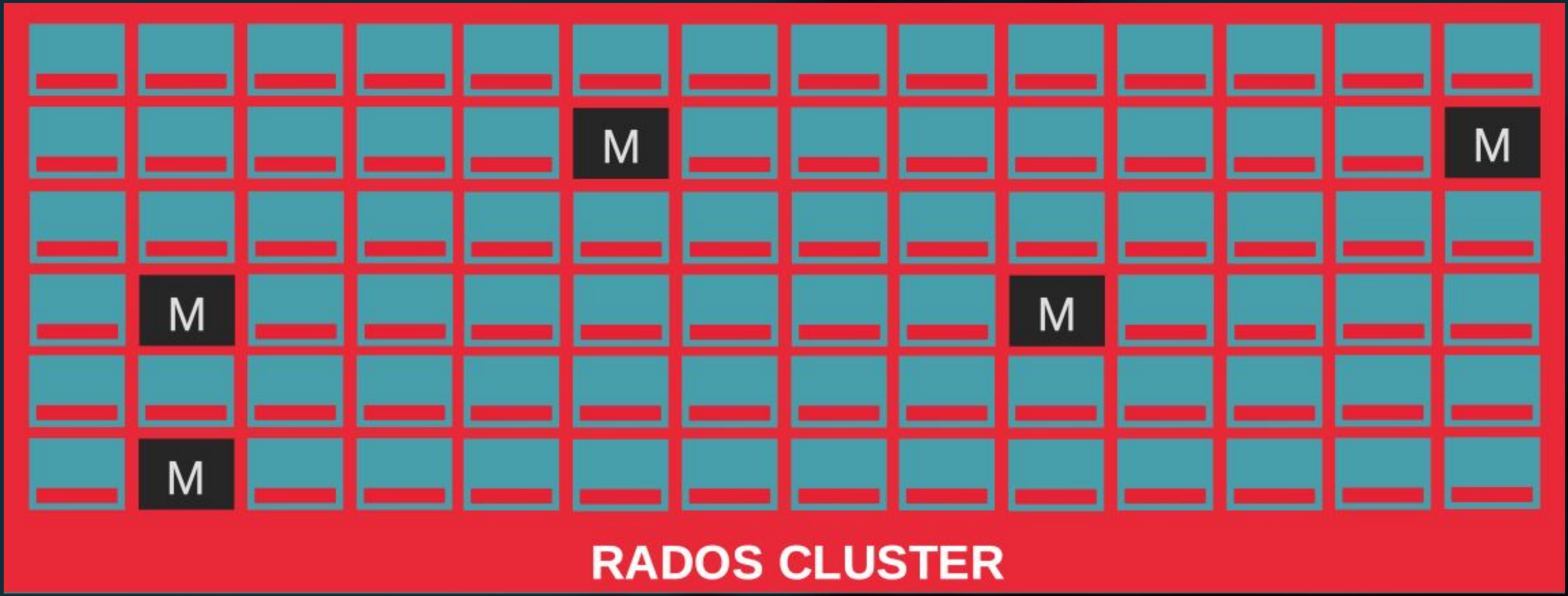
- Working on Ceph since 2009 (10 years!), all over
- gfarnum@redhat.com
- @gregsfortytwo

ceph

Ceph Daemons



Ceph Cluster Daemons

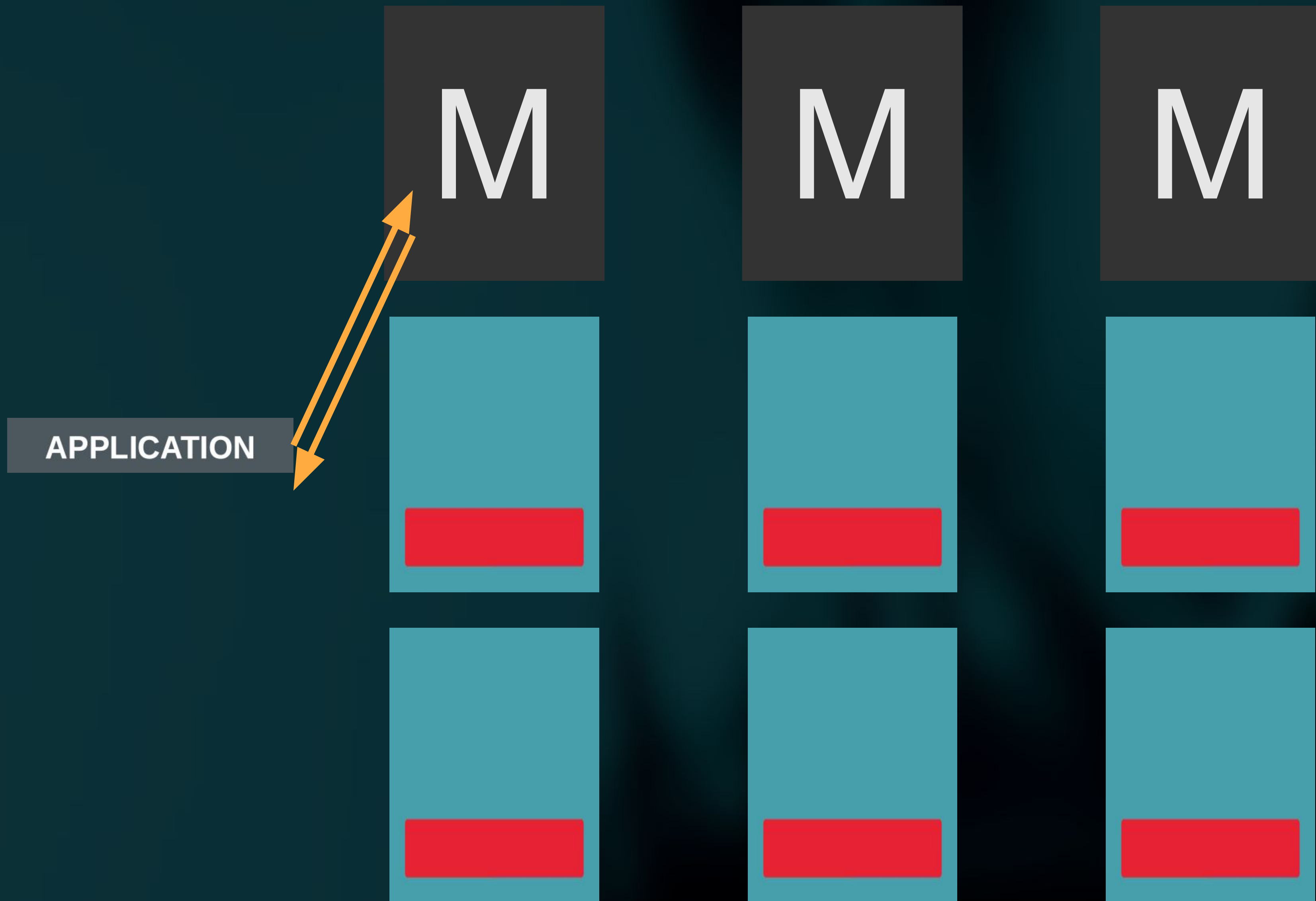


Ceph Cluster Daemons

RADOS WRITES

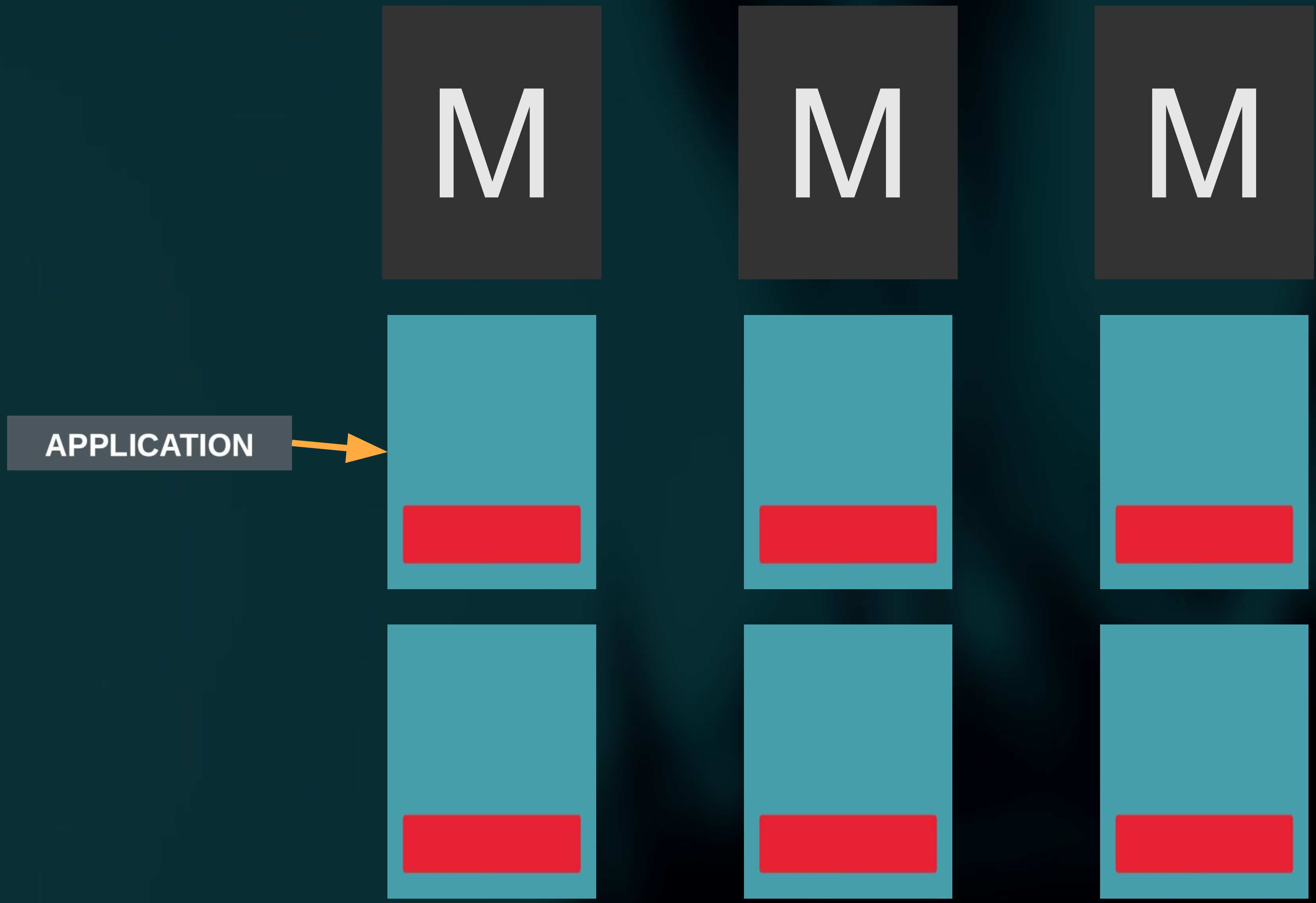
- Application connects to monitors, gets “OSDMap” describing cluster
    - Only once!
- Runs CRUSH algorithm to learn which OSD owns the object (is “primary”)
- Sends write operation to primary OSD
- Primary OSD receives request, validates it
    - Sends operation to replica OSDs for this PG
    - Each OSD commits operation to disk, then replies to primary
    - Primary OSD replies to client

ceph



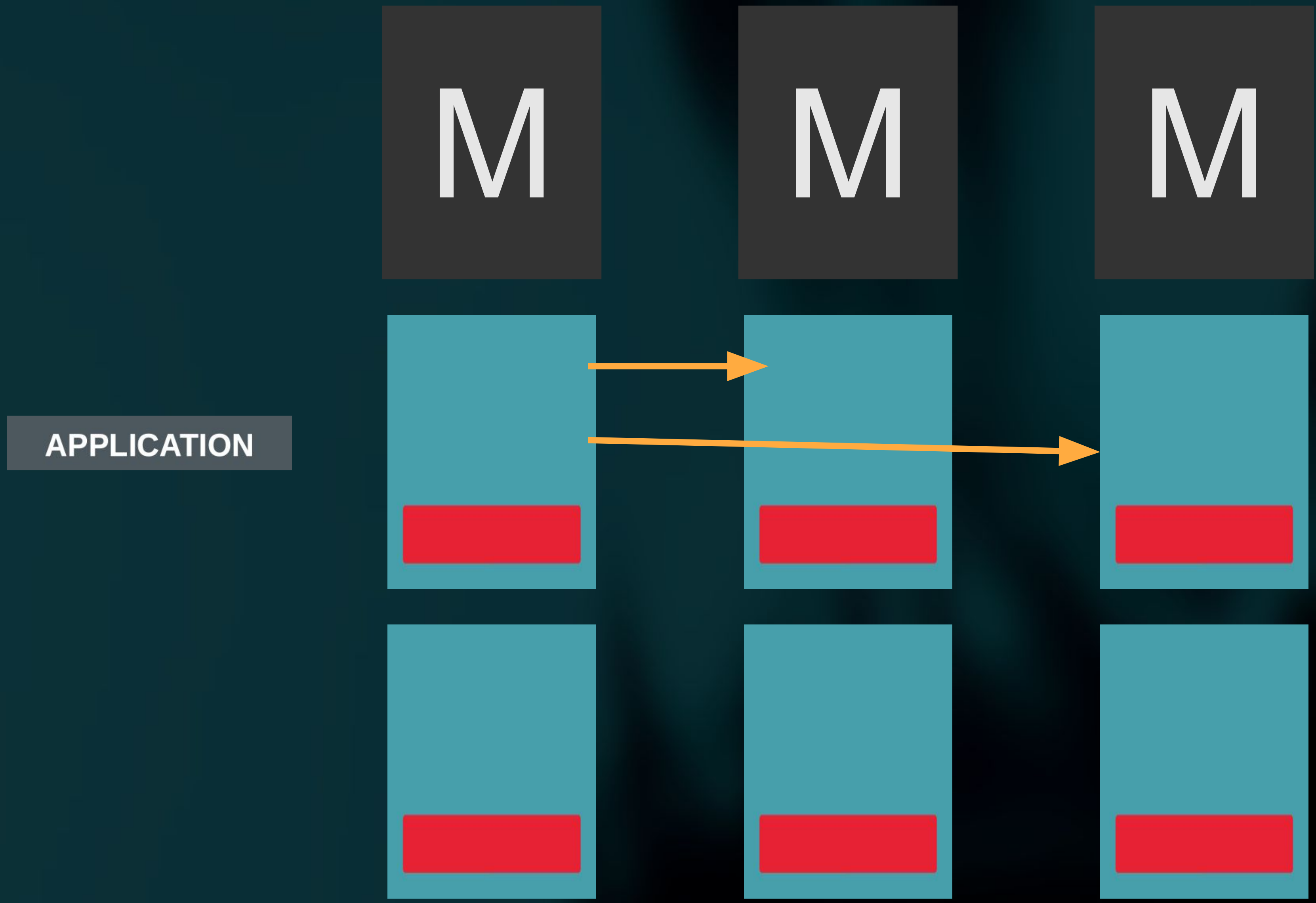
Writes in RADOS

ceph



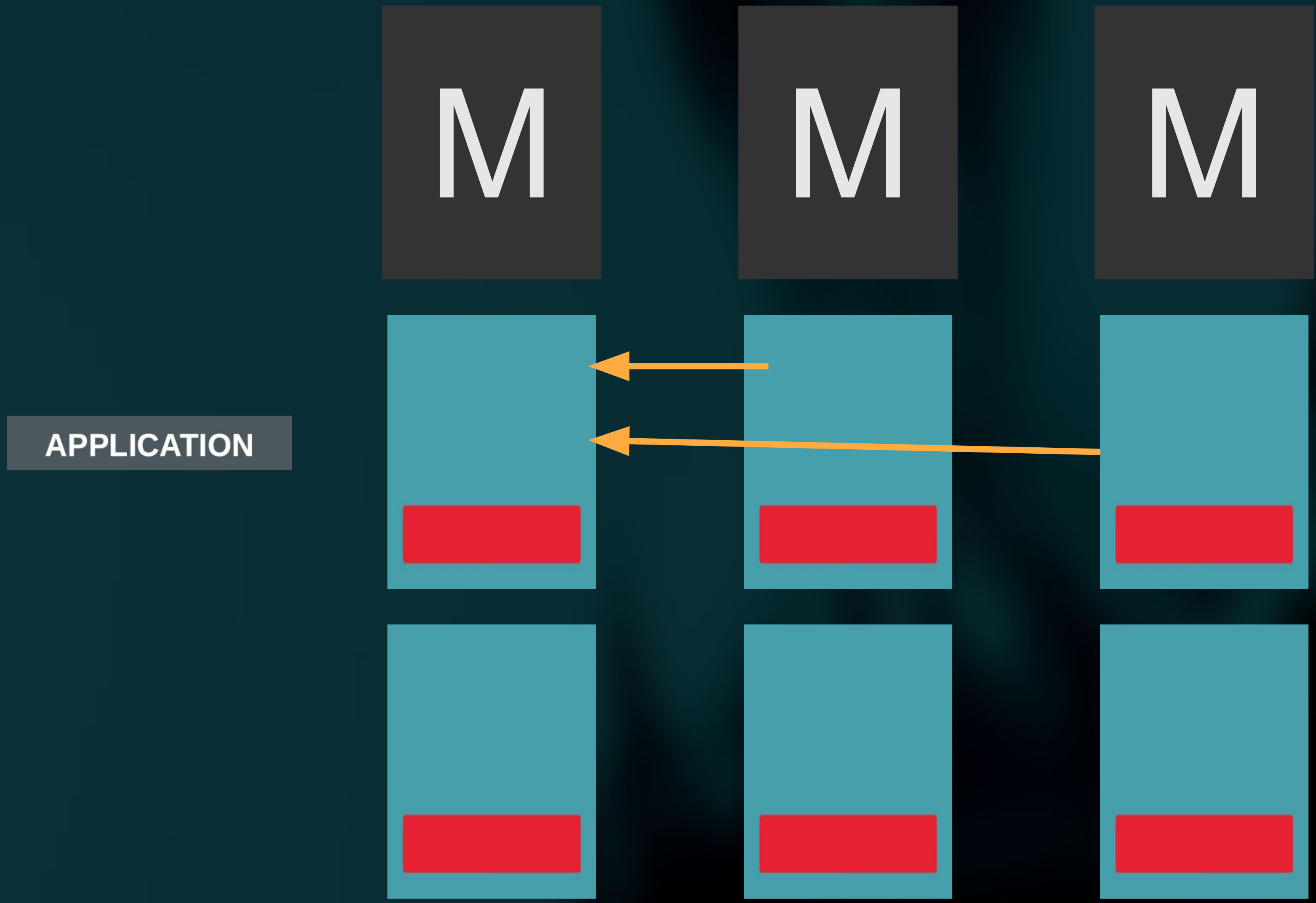
Writes in RADOS

ceph

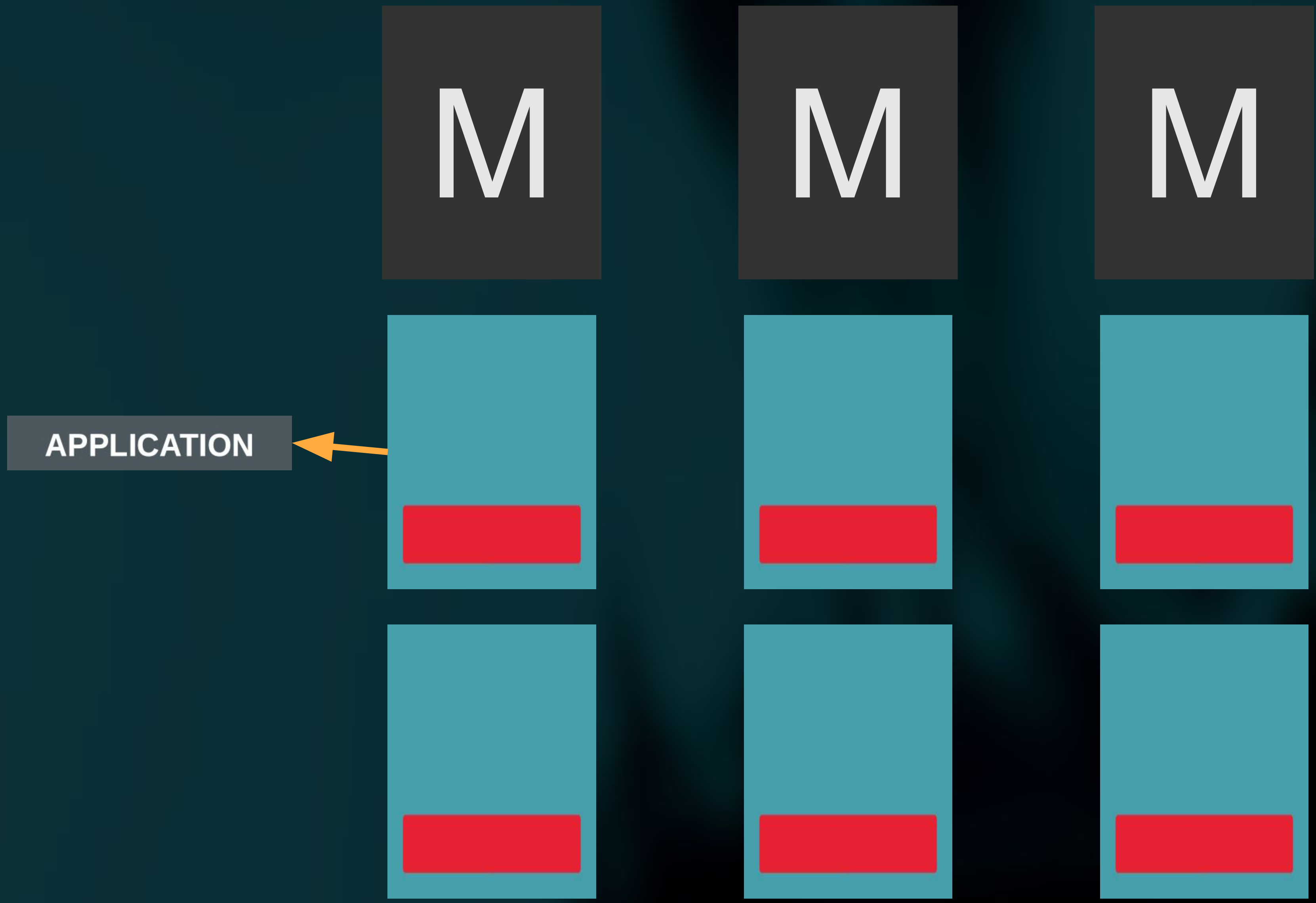


Writes in RADOS

ceph



Writes in RADOS



Writes in RADOS

STRETCH CLUSTER: DEFINITION

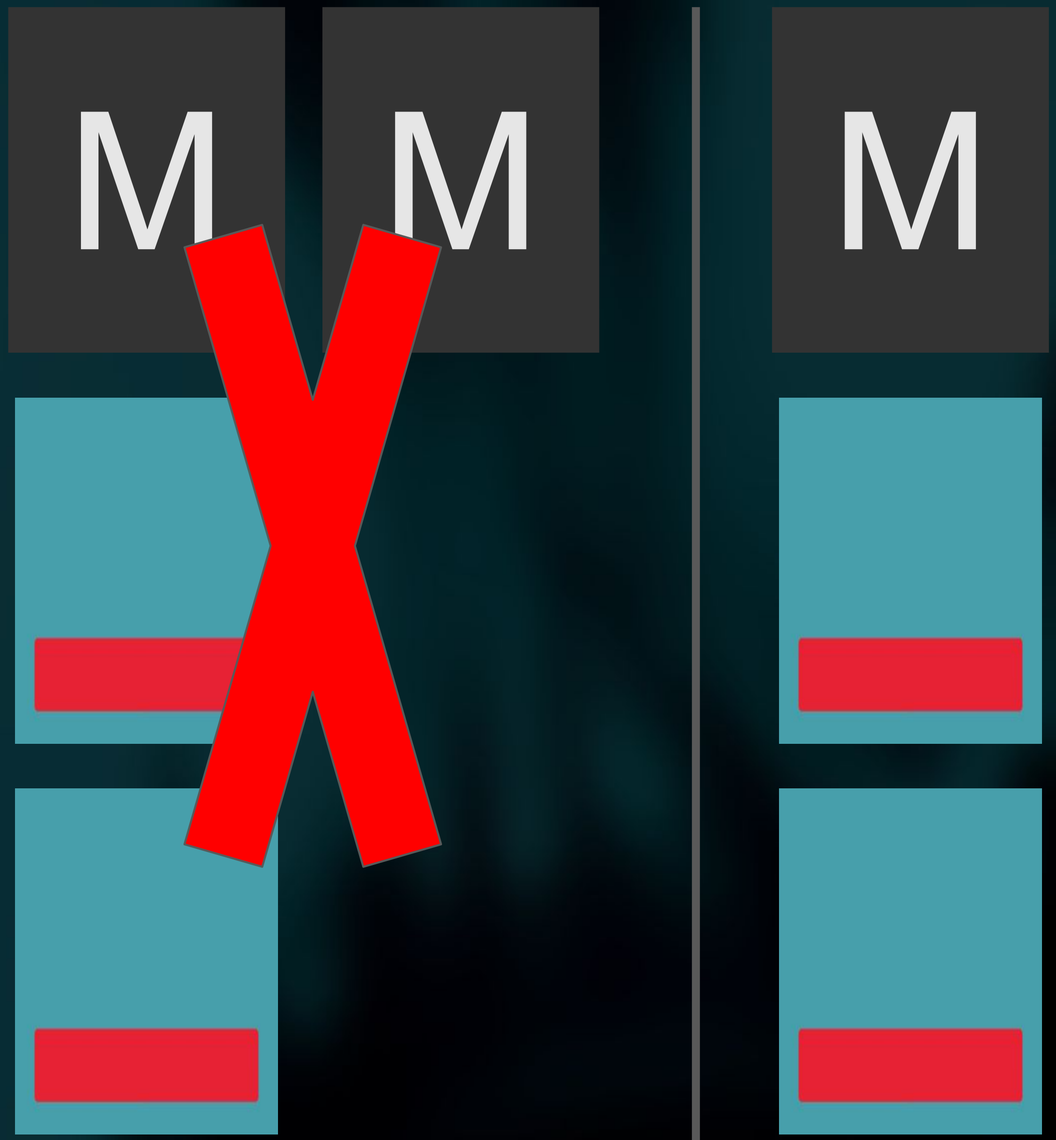
A cluster with servers in geographically separated data centers run over a WAN. We expect to still have LAN-like high-speed, low-latency connections, but limited links.

In particular, a much-higher than usual likelihood of (possibly asymmetric) network splits, and of the temporary or complete loss of an entire DC (⅓ to ½ the total cluster).

You can deploy a stretch cluster today…

Stretch Clusters Today: 3 Data Centers

Stretch Clusters Today: 2 Data Centers



Stretch Clusters Today: Don't Do This!

You can deploy a
stretch cluster today…
But there’s a problem

ceph

Monitor Leader Elections

LEADERS

- Leaders coordinate everything the monitors do
- All updates go to the leader
- Leaders distribute changes to everybody else (“peons”)

Interesting consequences:

- Peons don’t talk to each other
- Leaders talk to everybody else
- The only all-to-all communication is during elections, to choose the leader

LEADER ELECTIONS

- Start an election for some reason (turned on, timed out, etc)
- Bump the election epoch (so we can detect old messages/peers)
- Send a PROPOSE to all other monitors

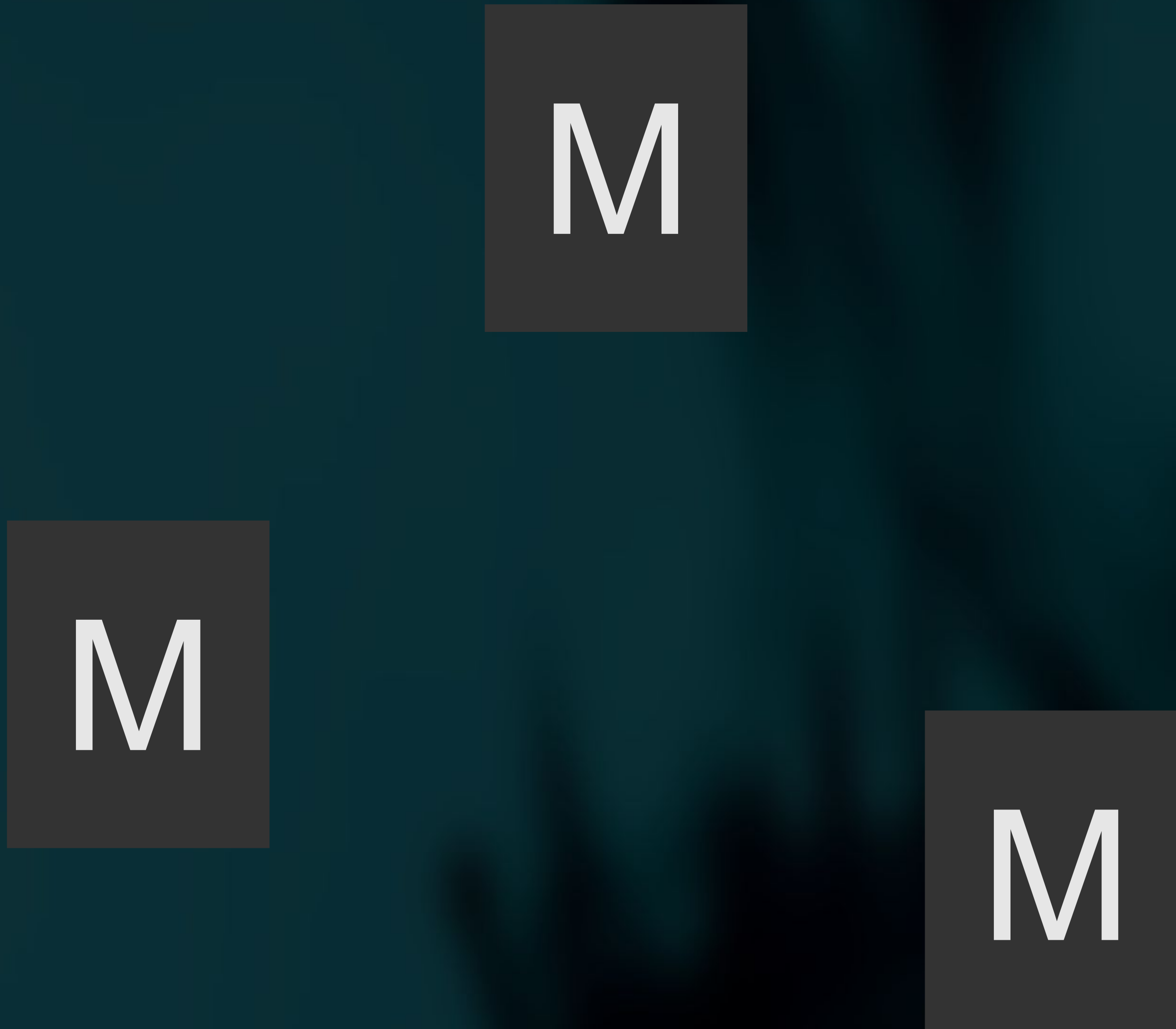
- When receiving a PROPOSE:
- If sender is not in quorum, start an election so they can join
- If sender is lower ID than us (and anybody else we've seen this election), DEFER to them
- If sender is higher ID than us, bump epoch and propose ourself

ceph

LEADER ELECTIONS

- If we get a DEFER from all our peers, become leader and send a VICTORY message
- If we time out an election and >half the monitors have DEFERed to us, become leader and send VICTORY
- If we time out and haven’t had a quorum DEFER to us and we haven’t gotten a victory, bump epoch and send out PROPOSE messages again

ceph



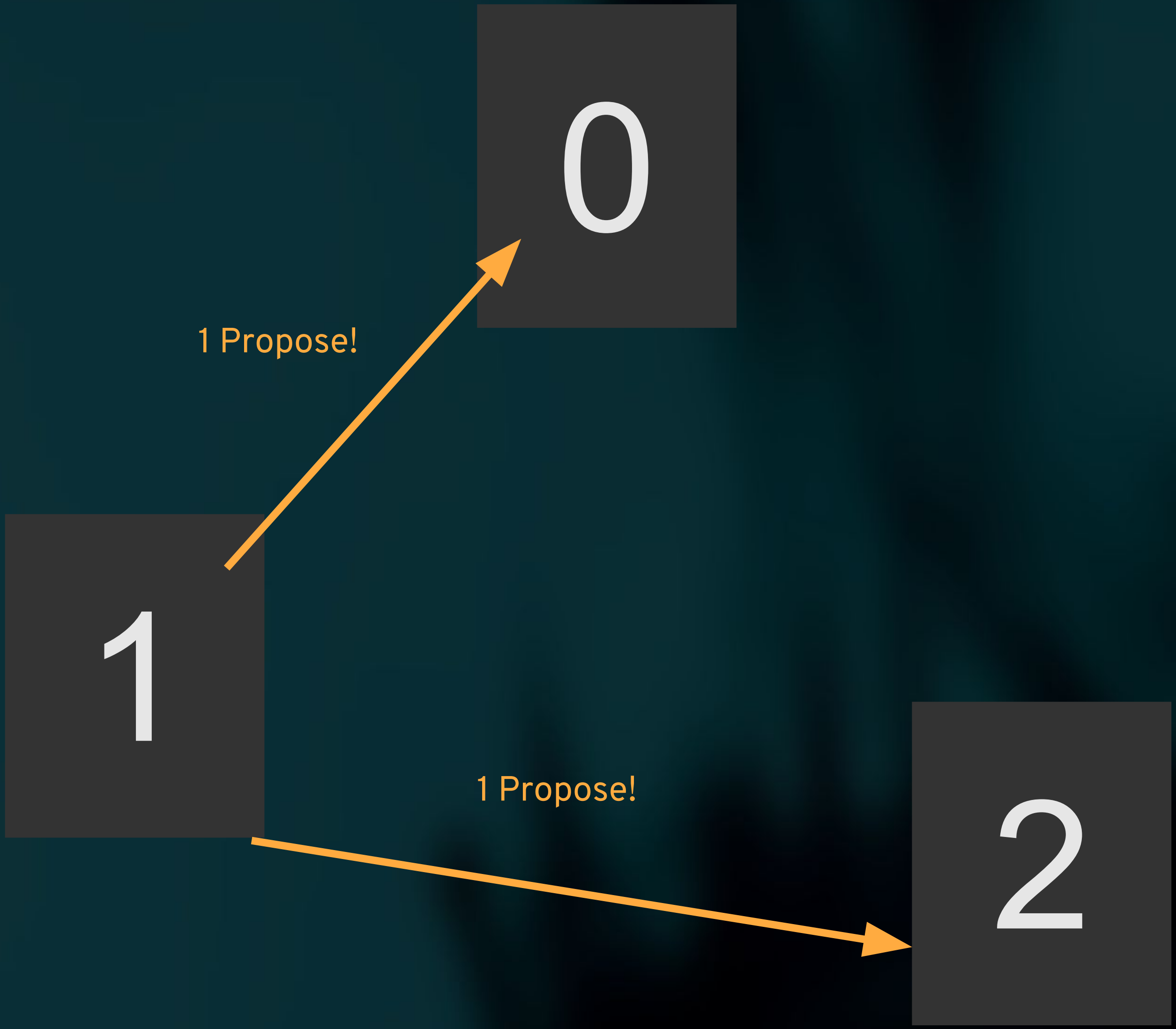
Leader Elections

0

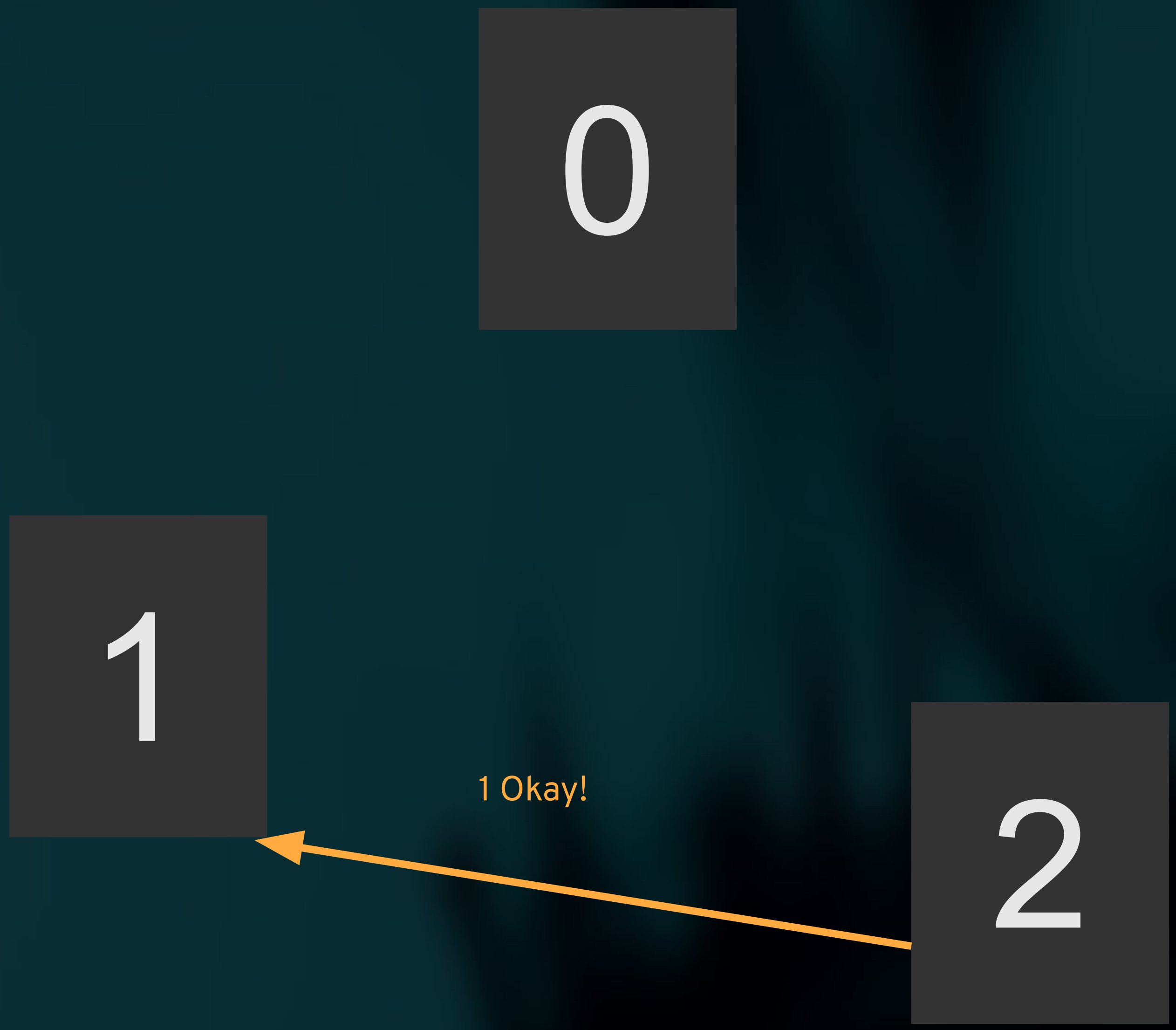
1

2

Leader Elections

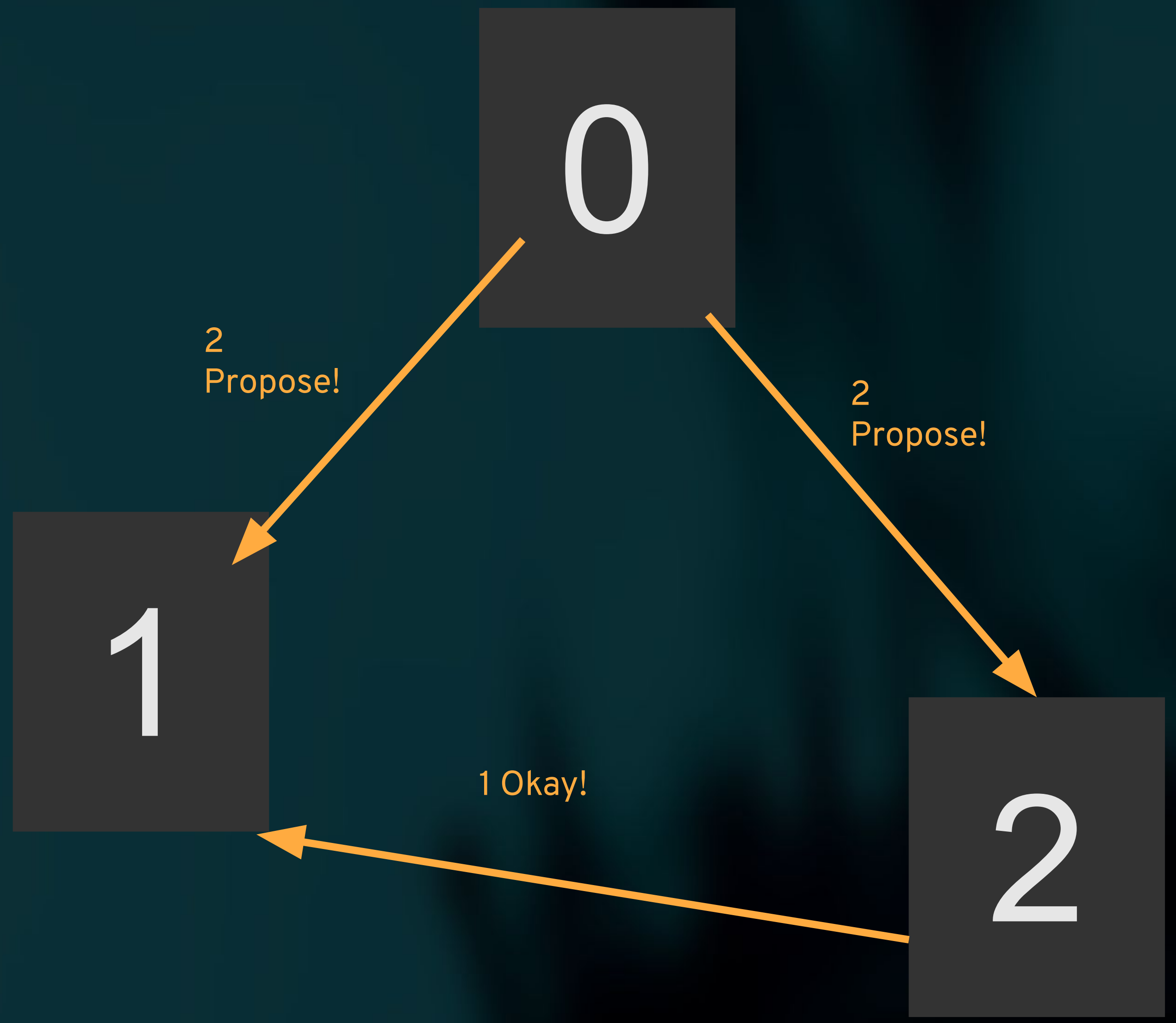


Leader Elections: Propose a new leader (yourself)

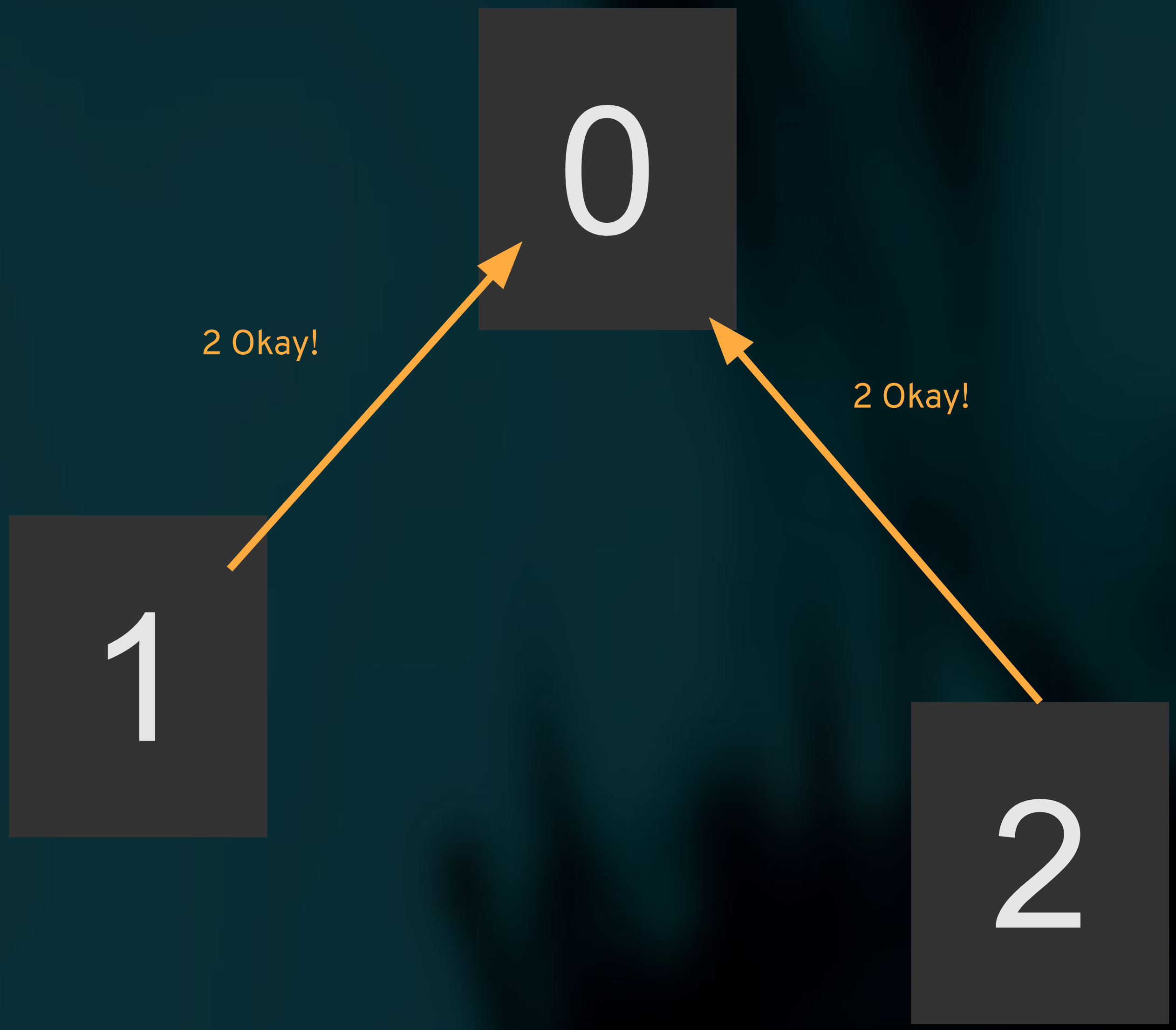


Leader Elections: Ack the propose, if it’s a better number

ceph

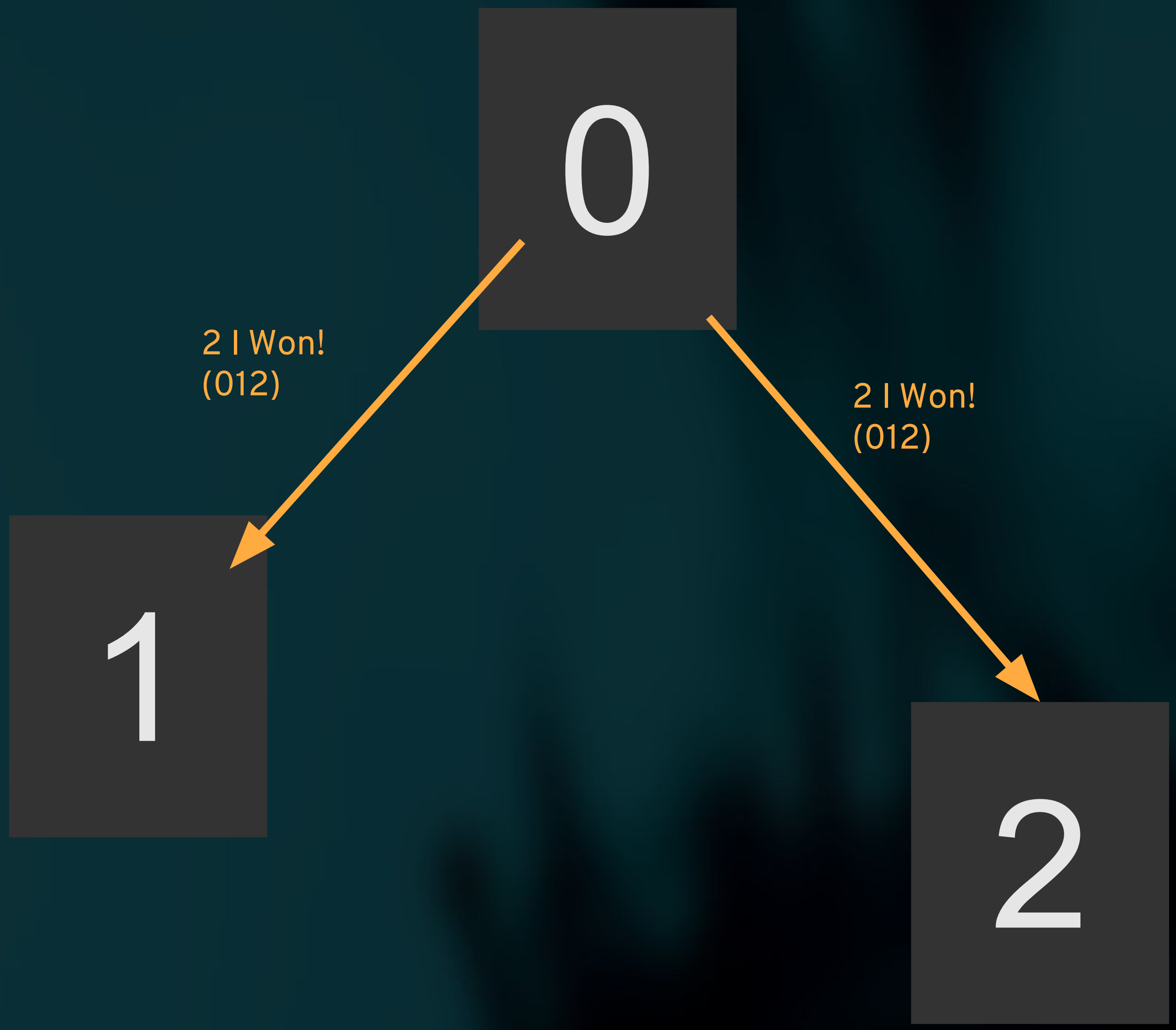


Leader Elections: Bump epoch and propose, if you're better



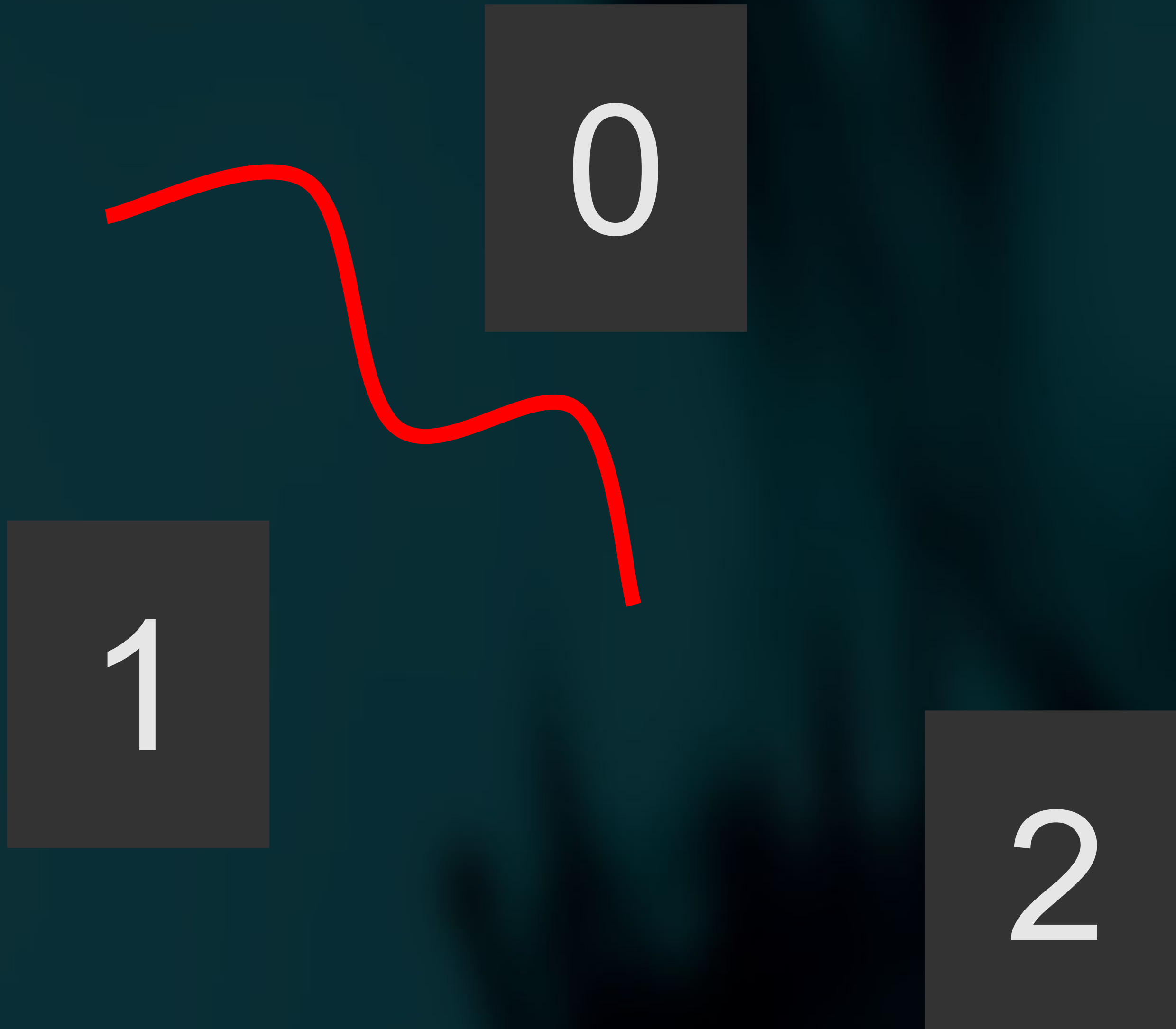
Leader Elections: Ack the propose, if it’s a better number

ceph

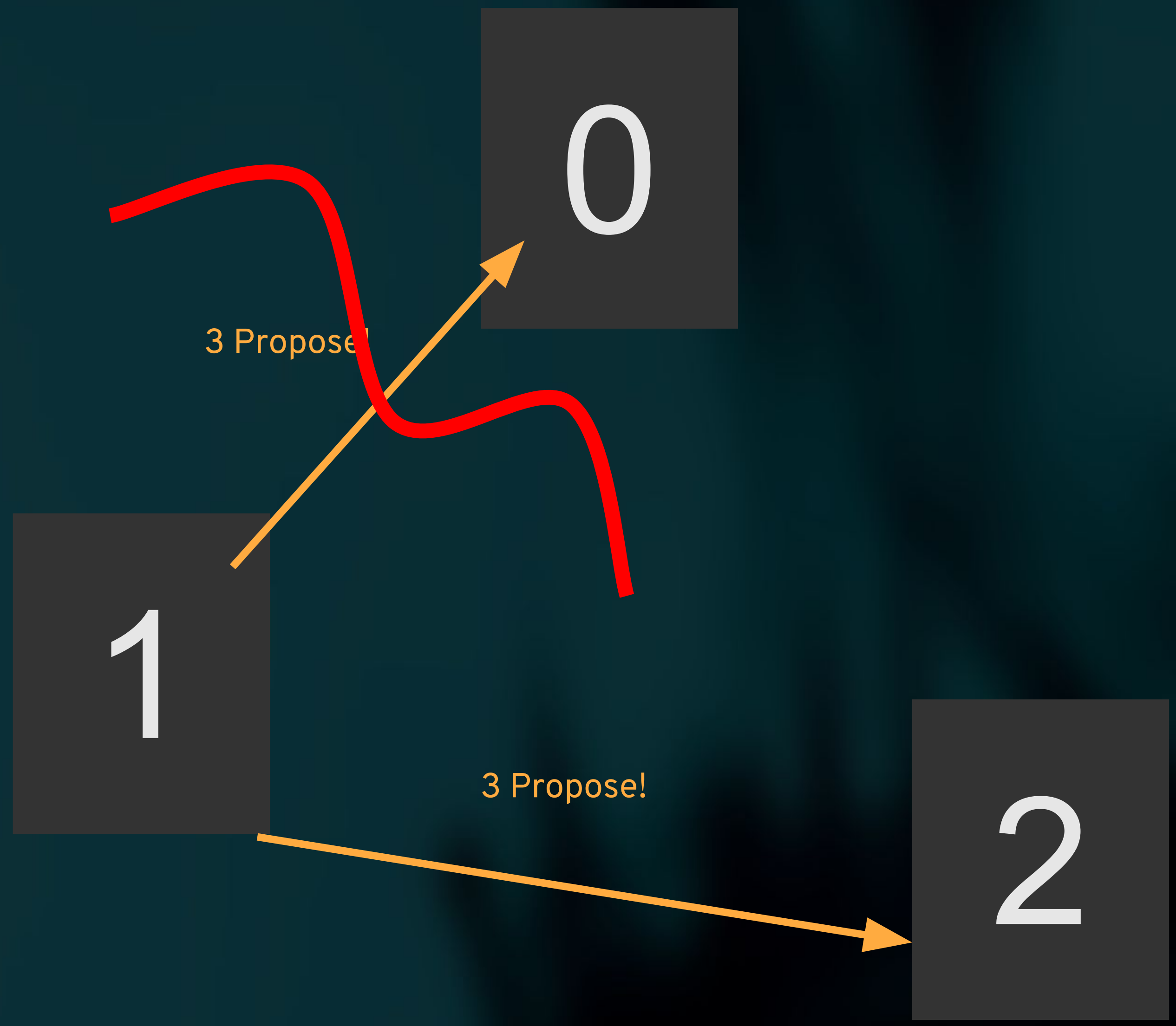


Leader Elections: Win if everybody acks you

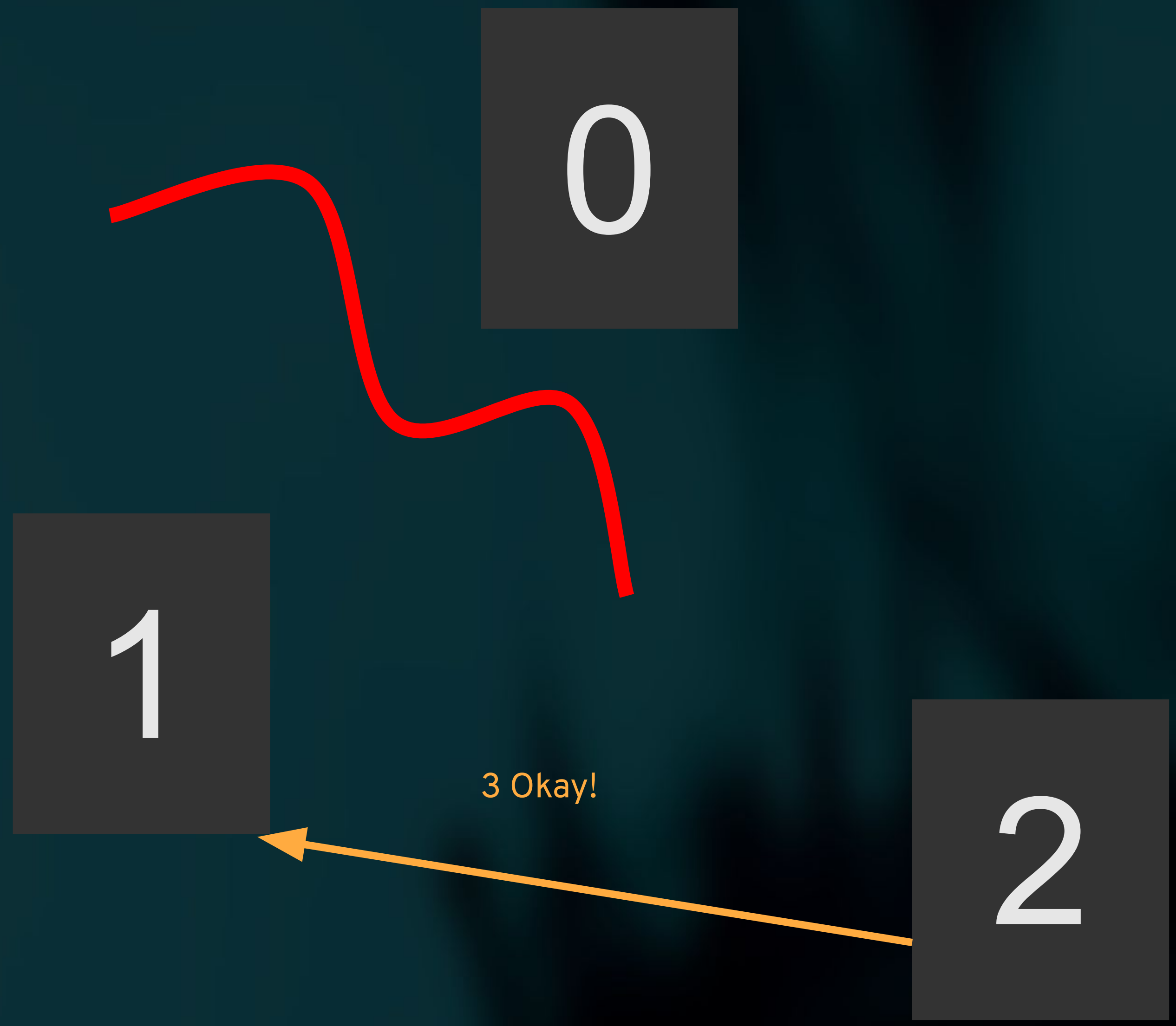
ceph



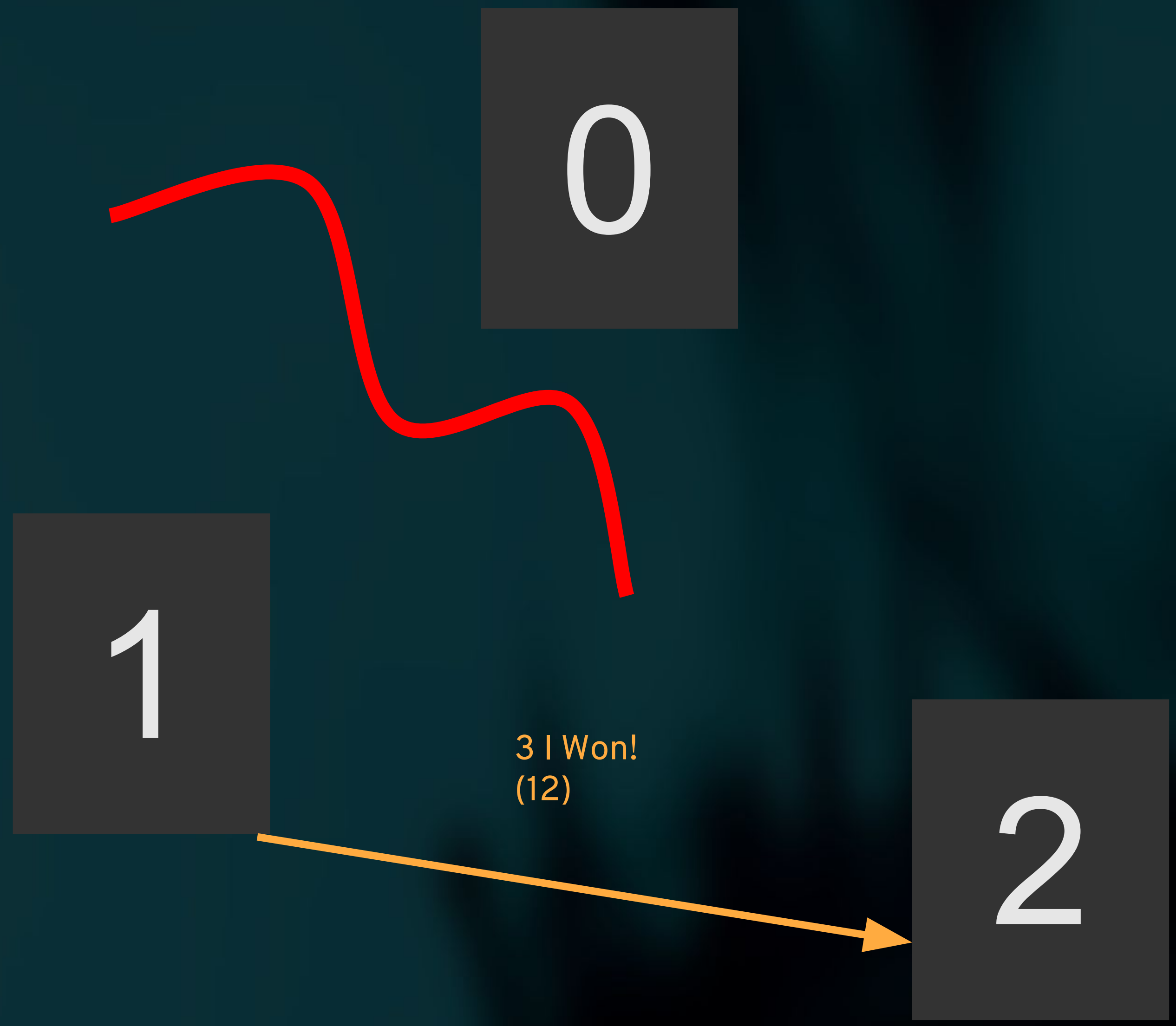
Leader Elections w/ Netsplit



Leader Elections w/ Netsplit: Propose a new leader (yourself)

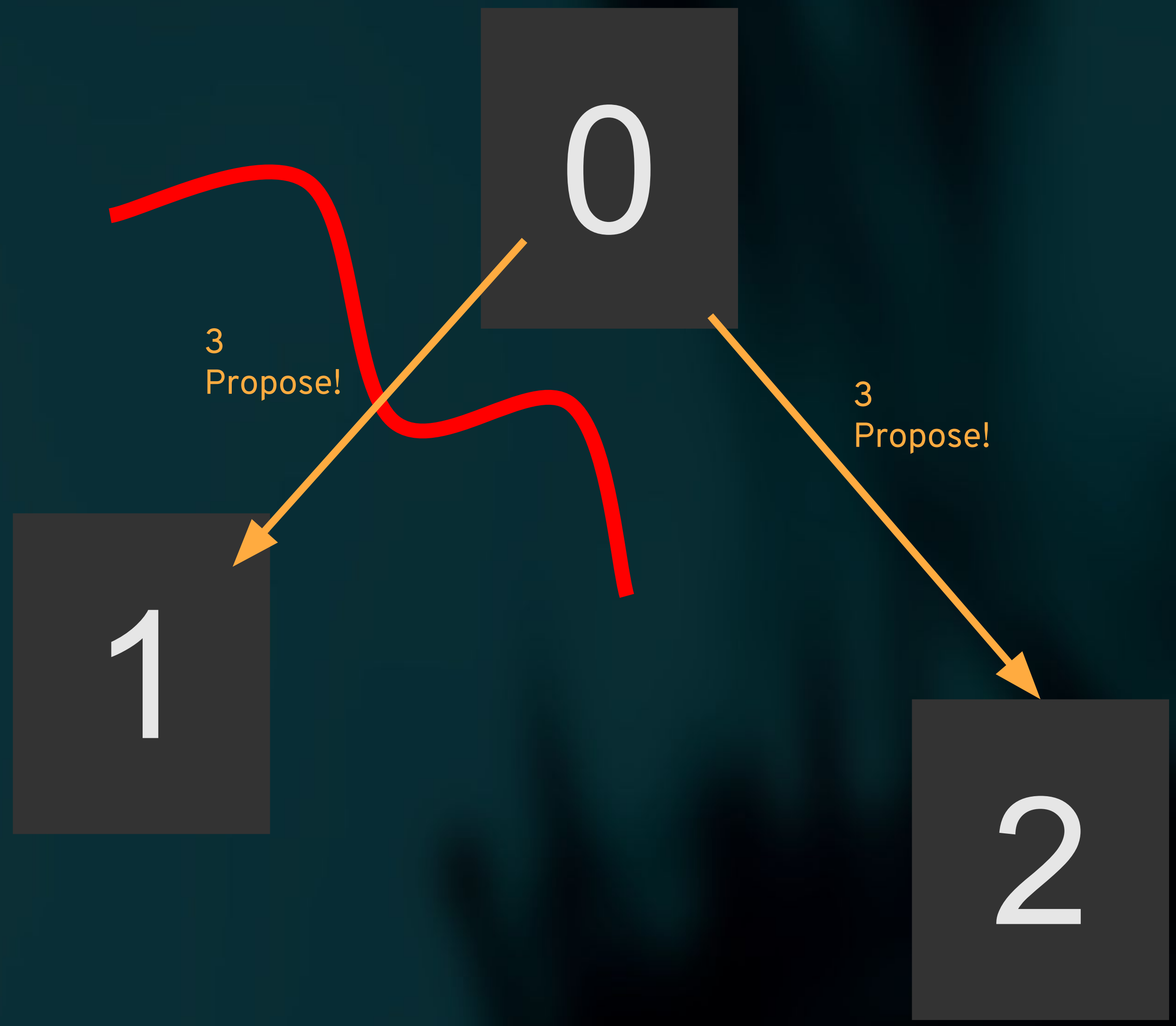


Leader Elections: Ack the propose, if it’s a better number

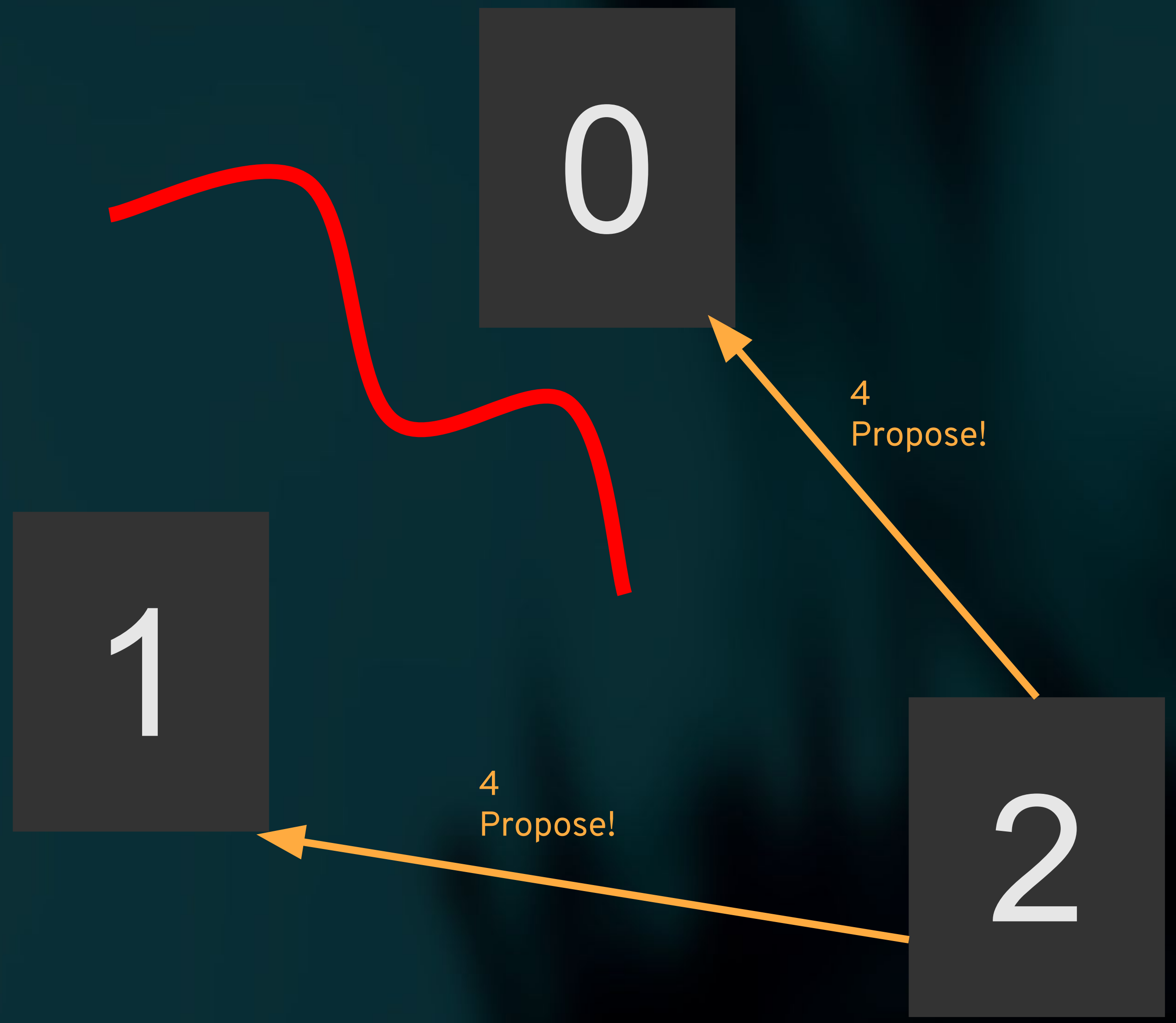


Leader Elections w/ Netsplit: Win if the election times out and you got enough acks

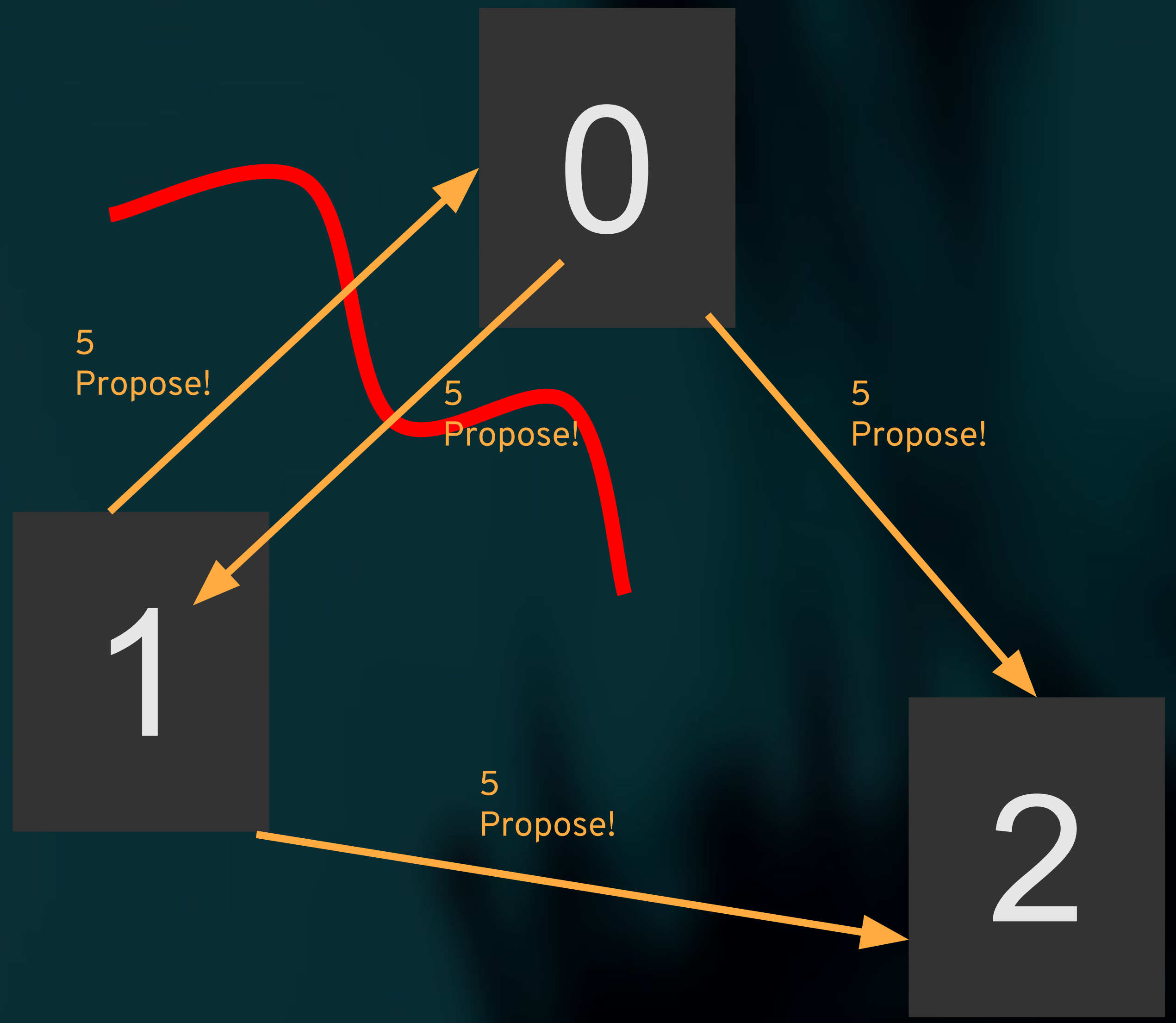
ceph



Leader Elections w/ Netsplit: Propose a new leader (yourself)



Leader Elections w/ Netsplit: Bump epoch and propose, if proposer is out of quorum



Leader Elections w/ Netsplit: ...oh no

ceph

New Leader Elections: The Plan

MAKE CODE CHANGEABLE

- Code mixed message passing and election logic in same functions
- Update it:
  - Split out new “ElectionLogic” class; deals with abstract Propose and Ack concepts
  - Message passing remains in “Elector”, which calls into ElectionLogic
  - Write ElectionLogic unit tests!!! (Simple time-step framework)
- Makes election algorithm dramatically easier to iterate and experiment with
- Detected several issues in updated algorithms without ever running a real cluster
  - Validate algorithm changes in <1 second
  - Easy to create complex scenarios (connectivity, Elector state, etc) in short functions

UNIT TESTING CODE

```
void blocked_connection_continues_election(ElectionLogic::election_strategy strategy)
{
  Election election(5, strategy);
  election.block_bidirectional_messages(0, 1);
  election.start_all();
  int steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  // This is a failure mode!
  ASSERT_FALSE(election.election_stable());
  ASSERT_FALSE(election.quorum_stable(6)); // double the timer_steps we use
  election.unblock_bidirectional_messages(0, 1);
  steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  ASSERT_TRUE(election.election_stable());
  ASSERT_TRUE(election.quorum_stable(6)); // double the timer_steps we use
  ASSERT_TRUE(election.check_leader_agreement());
  ASSERT_TRUE(election.check_epoch_agreement());
}
```

~500 line test harness; simple-to-complicated tests

ceph

UNIT TESTING CODE

```
void blocked_connection_continues_election(ElectionLogic::election_strategy strategy)
{
  Election election(5, strategy);
  election.block_bidirectional_messages(0, 1);
  election.start_all();
  int steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  // This is a failure mode!
  ASSERT_FALSE(election.election_stable());
  ASSERT_FALSE(election.quorum_stable(6)); // double the timer_steps we use
  election.unblock_bidirectional_messages(0, 1);
  steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  ASSERT_TRUE(election.election_stable());
  ASSERT_TRUE(election.quorum_stable(6)); // double the timer_steps we use
  ASSERT_TRUE(election.check_leader_agreement());
  ASSERT_TRUE(election.check_epoch_agreement());
}
```

Create an election setup

~500 line test harness; simple-to-complicated tests

ceph

UNIT TESTING CODE

```
void blocked_connection_continues_election(ElectionLogic::election_strategy strategy)
{
  Election election(5, strategy);
  election.block_bidirectional_messages(0, 1);
  election.start_all();
  int steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  // This is a failure mode!
  ASSERT_FALSE(election.election_stable());
  ASSERT_FALSE(election.quorum_stable(6)); // double the timer_steps we use
  election.unblock_bidirectional_messages(0, 1);
  steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  ASSERT_TRUE(election.election_stable());
  ASSERT_TRUE(election.quorum_stable(6)); // double the timer_steps we use
  ASSERT_TRUE(election.check_leader_agreement());
  ASSERT_TRUE(election.check_epoch_agreement());
}
```

electors 0 and 1 can't talk to each other

~500 line test harness; simple-to-complicated tests

ceph

UNIT TESTING CODE

```
void blocked_connection_continues_election(ElectionLogic::election_strategy strategy)
{
  Election election(5, strategy);
  election.block_bidirectional_messages(0, 1);
  election.start_all();
  int steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  // This is a failure mode!
  ASSERT_FALSE(election.election_stable());
  ASSERT_FALSE(election.quorum_stable(6)); // double the timer_steps we use
  election.unblock_bidirectional_messages(0, 1);
  steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  ASSERT_TRUE(election.election_stable());
  ASSERT_TRUE(election.quorum_stable(6)); // double the timer_steps we use
  ASSERT_TRUE(election.check_leader_agreement());
  ASSERT_TRUE(election.check_epoch_agreement());
}
```

Turn on all the electors

~500 line test harness; simple-to-complicated tests

ceph

UNIT TESTING CODE

```
void blocked_connection_continues_election(ElectionLogic::election_strategy strategy)
{
  Election election(5, strategy);
  election.block_bidirectional_messages(0, 1);
  election.start_all();
  int steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  // This is a failure mode!
  ASSERT_FALSE(election.election_stable());
  ASSERT_FALSE(election.quorum_stable(6)); // double the timer_steps we use
  election.unblock_bidirectional_messages(0, 1);
  steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  ASSERT_TRUE(election.election_stable());
  ASSERT_TRUE(election.quorum_stable(6)); // double the timer_steps we use
  ASSERT_TRUE(election.check_leader_agreement());
  ASSERT_TRUE(election.check_epoch_agreement());
}
```

Run forward in time, up to 100 message intervals

~500 line test harness; simple-to-complicated tests

UNIT TESTING CODE

```
void blocked_connection_continues_election(ElectionLogic::election_strategy strategy)
{
  Election election(5, strategy);
  election.block_bidirectional_messages(0, 1);
  election.start_all();
  int steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  // This is a failure mode!
  ASSERT_FALSE(election.election_stable());
  ASSERT_FALSE(election.quorum_stable(6)); // double the timer_steps we use
  election.unblock_bidirectional_messages(0, 1);
  steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  ASSERT_TRUE(election.election_stable());
  ASSERT_TRUE(election.quorum_stable(6)); // double the timer_steps we use
  ASSERT_TRUE(election.check_leader_agreement());
  ASSERT_TRUE(election.check_epoch_agreement());
}
```

When an election is stable:
Electors have no timeouts pending, and
no messages in flight

~500 line test harness; simple-to-complicated tests

ceph

UNIT TESTING CODE

```
void blocked_connection_continues_election(ElectionLogic::election_strategy strategy)
{
  Election election(5, strategy);
  election.block_bidirectional_messages(0, 1);
  election.start_all();
  int steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  // This is a failure mode!
  ASSERT_FALSE(election.election_stable());
  ASSERT_FALSE(election.quorum_stable(6)); // double the timer_steps we use
  election.unblock_bidirectional_messages(0, 1);
  steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  ASSERT_TRUE(election.election_stable());
  ASSERT_TRUE(election.quorum_stable(6)); // double the timer_steps we use
  ASSERT_TRUE(election.check_leader_agreement());
  ASSERT_TRUE(election.check_epoch_agreement());
}
```

Assert the quorum has changed recently

timer_steps is how may timesteps before an elector decides it won't get a reply to messages. That's 3, by default.

~500 line test harness; simple-to-complicated tests

UNIT TESTING CODE

```
void blocked_connection_continues_election(ElectionLogic::election_strategy strategy)
{
  Election election(5, strategy);
  election.block_bidirectional_messages(0, 1);
  election.start_all();
  int steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  // This is a failure mode!
  ASSERT_FALSE(election.election_stable());
  ASSERT_FALSE(election.quorum_stable(6)); // double the timer_steps we use
  election.unblock_bidirectional_messages(0, 1);
  steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  ASSERT_TRUE(election.election_stable());
  ASSERT_TRUE(election.quorum_stable(6)); // double the timer_steps we use
  ASSERT_TRUE(election.check_leader_agreement());
  ASSERT_TRUE(election.check_epoch_agreement());
}
```

Run forward again

~500 line test harness; simple-to-complicated tests

ceph

UNIT TESTING CODE

```
void blocked_connection_continues_election(ElectionLogic::election_strategy strategy)
{
  Election election(5, strategy);
  election.block_bidirectional_messages(0, 1);
  election.start_all();
  int steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  // This is a failure mode!
  ASSERT_FALSE(election.election_stable());
  ASSERT_FALSE(election.quorum_stable(6)); // double the timer_steps we use
  election.unblock_bidirectional_messages(0, 1);
  steps = election.run_timesteps(100);
  ldout(g_ceph_context, 1) << "ran in " << steps << " timesteps" << dendl;
  ASSERT_TRUE(election.election_stable());
  ASSERT_TRUE(election.quorum_stable(6)); // double the timer_steps we use
  ASSERT_TRUE(election.check_leader_agreement());
  ASSERT_TRUE(election.check_epoch_agreement());
}
```

All the electors agree who the leader is

All the electors agree what election epoch it is

~500 line test harness; simple-to-complicated tests

ceph

DEVELOP A NEW ALGORITHM

Key ideas:

- Maintain connection scores between each monitor
    - And share these broadly so everybody has an almost-current view of connectivity
- Handle propose messages based on score instead of ID number
- Specify monitors as “disallowed leaders”
    - A tiebreaker monitor might be far away and slow!
- …and that’s really it in broad strokes
    - It’s more complicated in detail: unlike IDs, scores change! Lots of new and newly-explicit invariants and checks, care in sharing and changing score views, etc

Pull request: https://github.com/ceph/ceph/pull/32336

UNIT TESTING CODE

```
ConnectionTracker& ct0 = election.electors[0]->peer_tracker;
ConnectionReport& cr0 = *get_connection_reports(ct0);
cr0.history[1] = 0.5;
cr0.history[2] = 0.5;
ct0.increase_version();
```

Muck around with the scores to set them directly and lie about system state

```
election.ping_interval = 0; // disable pinging to update the scores
ldout(g_ceph_context, 5) << "mangled the scores to be different" << dendl;

election.start_all();
election.run_timesteps(50);
ASSERT_TRUE(election.quorum_stable(30));
ASSERT_TRUE(election.election_stable());
ASSERT_TRUE(election.check_leader_agreement());
ASSERT_TRUE(election.check_epoch_agreement());
```

~500 line test harness; simple-to-complicated tests

ceph

UNIT TESTING CODE

Tracks all the peers

```
ConnectionTracker& ct0 = election.electors[0]->peer_tracker;
ConnectionReport& cr0 = *get_connection_reports(ct0);
cr0.history[1] = 0.5;
cr0.history[2] = 0.5;
ct0.increase_version();
```

Tracks a single peer's score

```
election.ping_interval = 0; // disable pinging to update the scores
ldout(g_ceph_context, 5) << "mangled the scores to be different" << dendl;

election.start_all();
election.run_timesteps(50);
ASSERT_TRUE(election.quorum_stable(30));
ASSERT_TRUE(election.election_stable());
ASSERT_TRUE(election.check_leader_agreement());
ASSERT_TRUE(election.check_epoch_agreement());
```

~500 line test harness; simple-to-complicated tests

ceph

UNIT TESTING CODE

```
ConnectionTracker& ct0 = election.electors[0]->peer_tracker;
ConnectionReport& cr0 = *get_connection_reports(ct0);
cr0.history[1] = 0.5;
cr0.history[2] = 0.5;
ct0.increase_version();
```

```
election.ping_interval = 0; // disable pinging to update the scores
ldout(g_ceph_context, 5) << "mangled the scores to be different" << dendl;

election.start_all();
election.run_timesteps(50);
ASSERT_TRUE(election.quorum_stable(30));
ASSERT_TRUE(election.election_stable());
ASSERT_TRUE(election.check_leader_agreement());
ASSERT_TRUE(election.check_epoch_agreement());
```

Disable pinging so it doesn't update to be correct

~500 line test harness; simple-to-complicated tests

UNIT TESTING CODE

```
ConnectionTracker& ct0 = election.electors[0]->peer_tracker;
ConnectionReport& cr0 = *get_connection_reports(ct0);
cr0.history[1] = 0.5;
cr0.history[2] = 0.5;
ct0.increase_version();
```

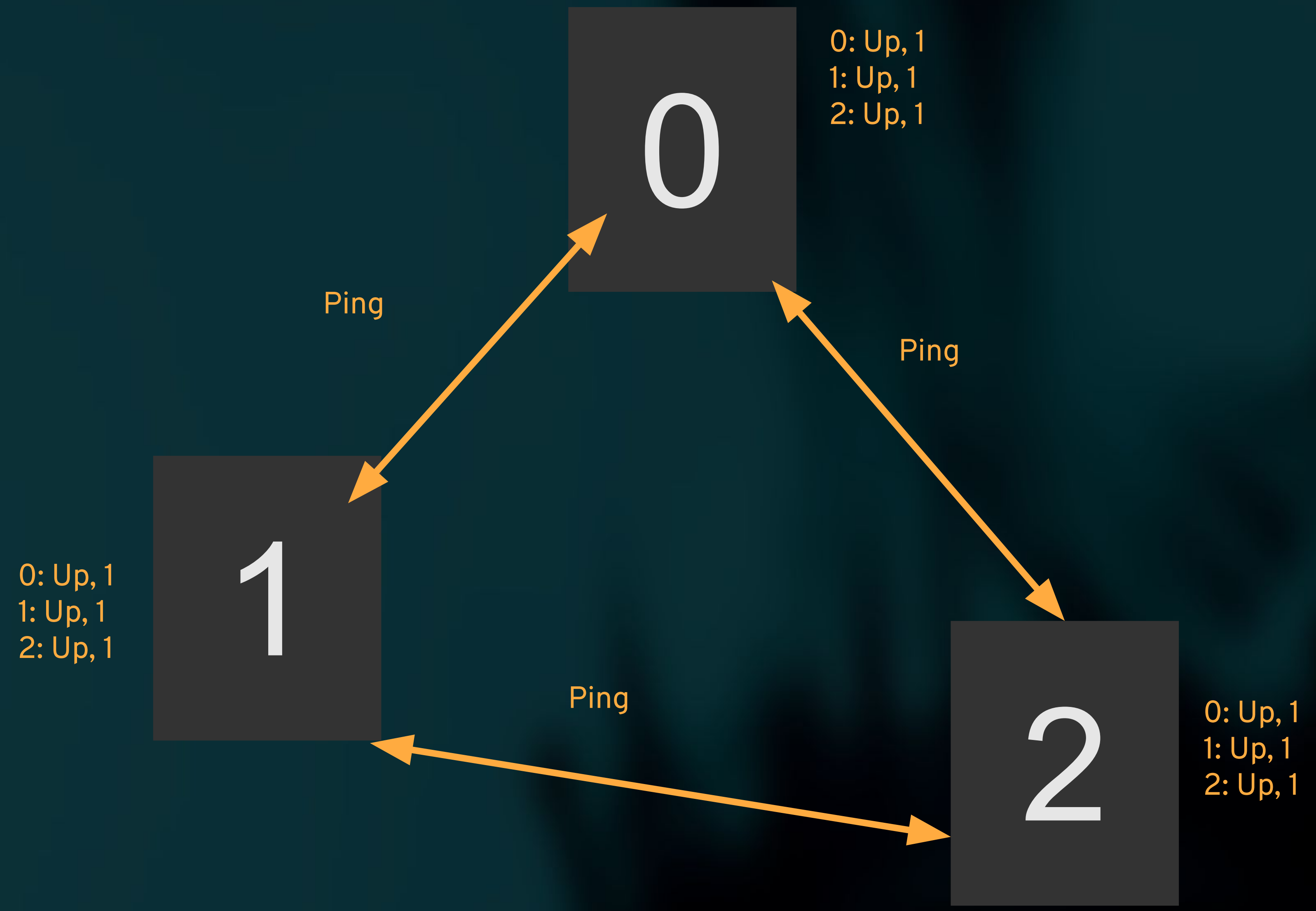
```
election.ping_interval = 0; // disable pinging to update the scores
ldout(g_ceph_context, 5) << "mangled the scores to be different" << dendl;

election.start_all();
election.run_timesteps(50);
ASSERT_TRUE(election.quorum_stable(30));
ASSERT_TRUE(election.election_stable());
ASSERT_TRUE(election.check_leader_agreement());
ASSERT_TRUE(election.check_epoch_agreement());
```

Run forward a bit and validate everybody!

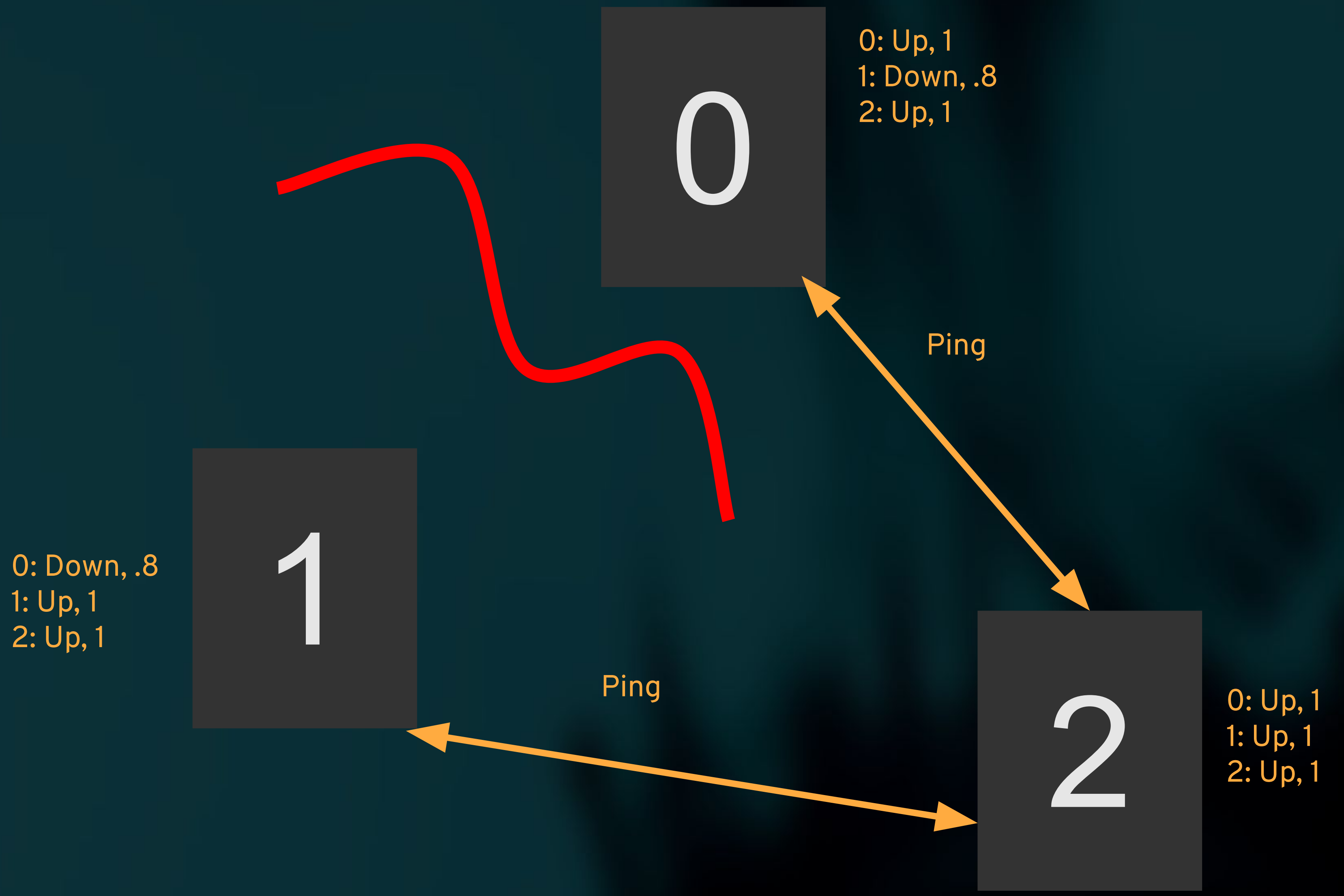
~500 line test harness; simple-to-complicated tests

New Leader Elections: Connectivity mode



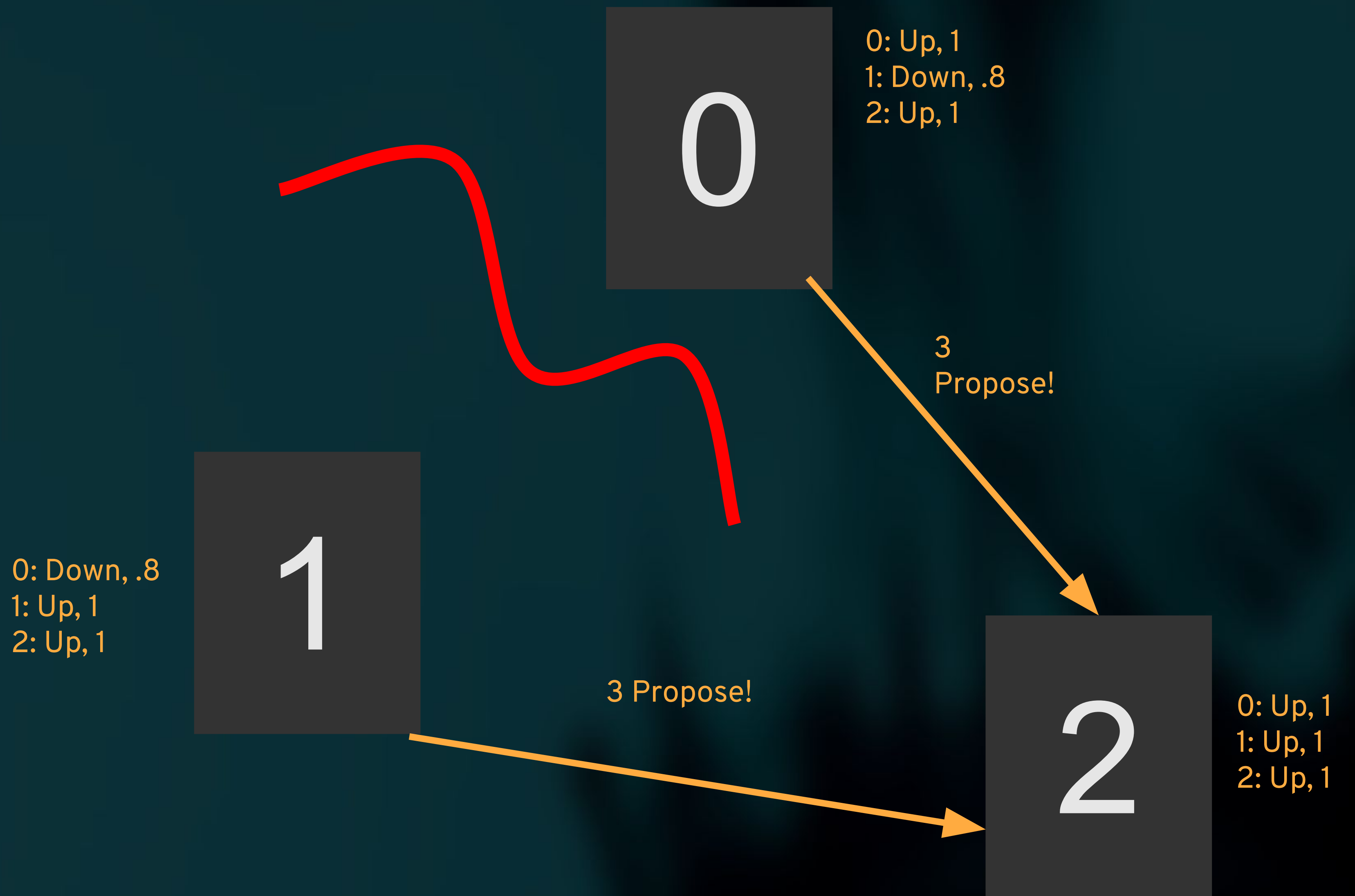
Leader Elections: Pinging

ceph



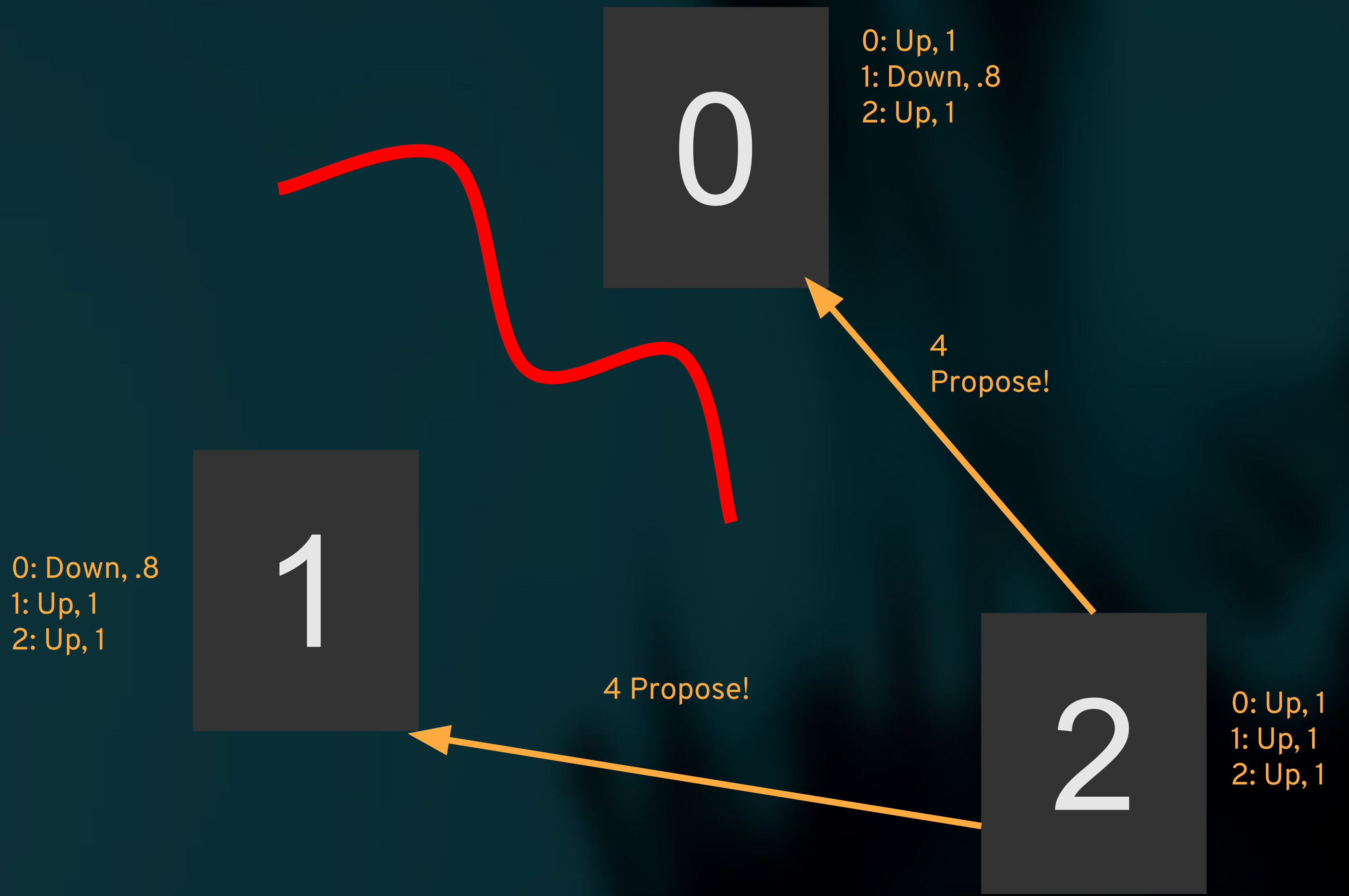
Leader Elections: Pinging

ceph

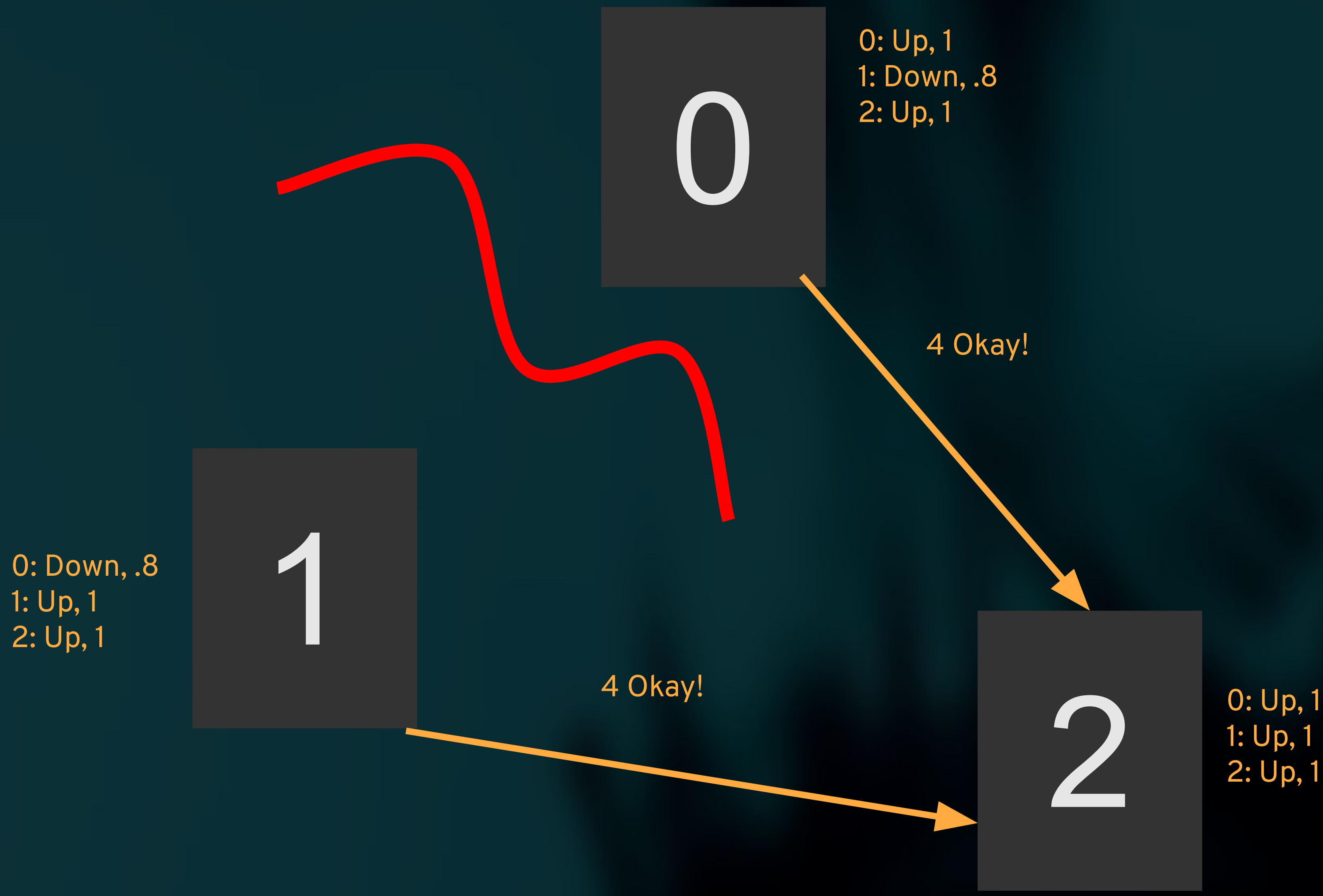


Leader Elections: Propose a new leader (yourself); send scores you know of

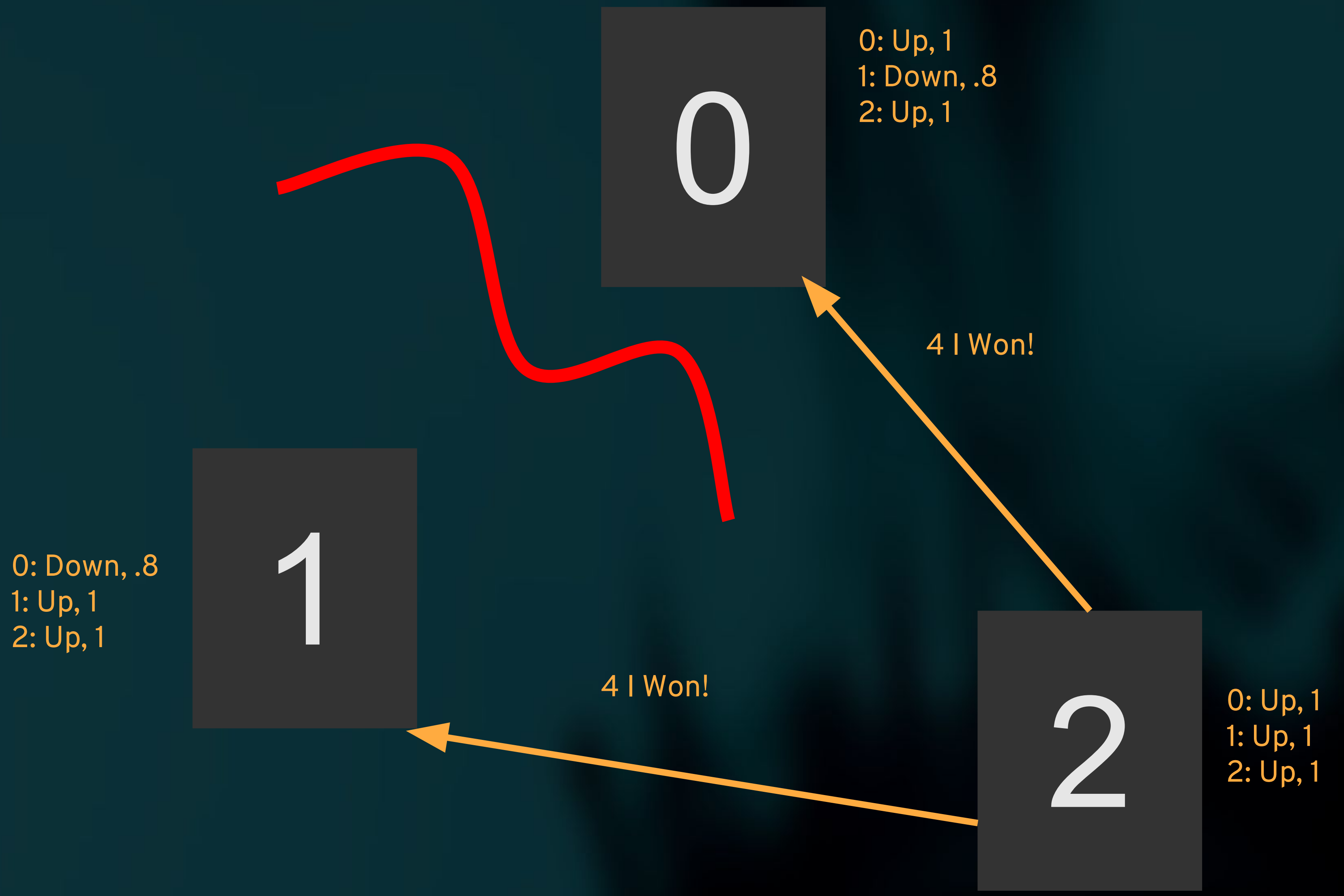
ceph



Leader Elections: Bump epoch and propose, if your score is better

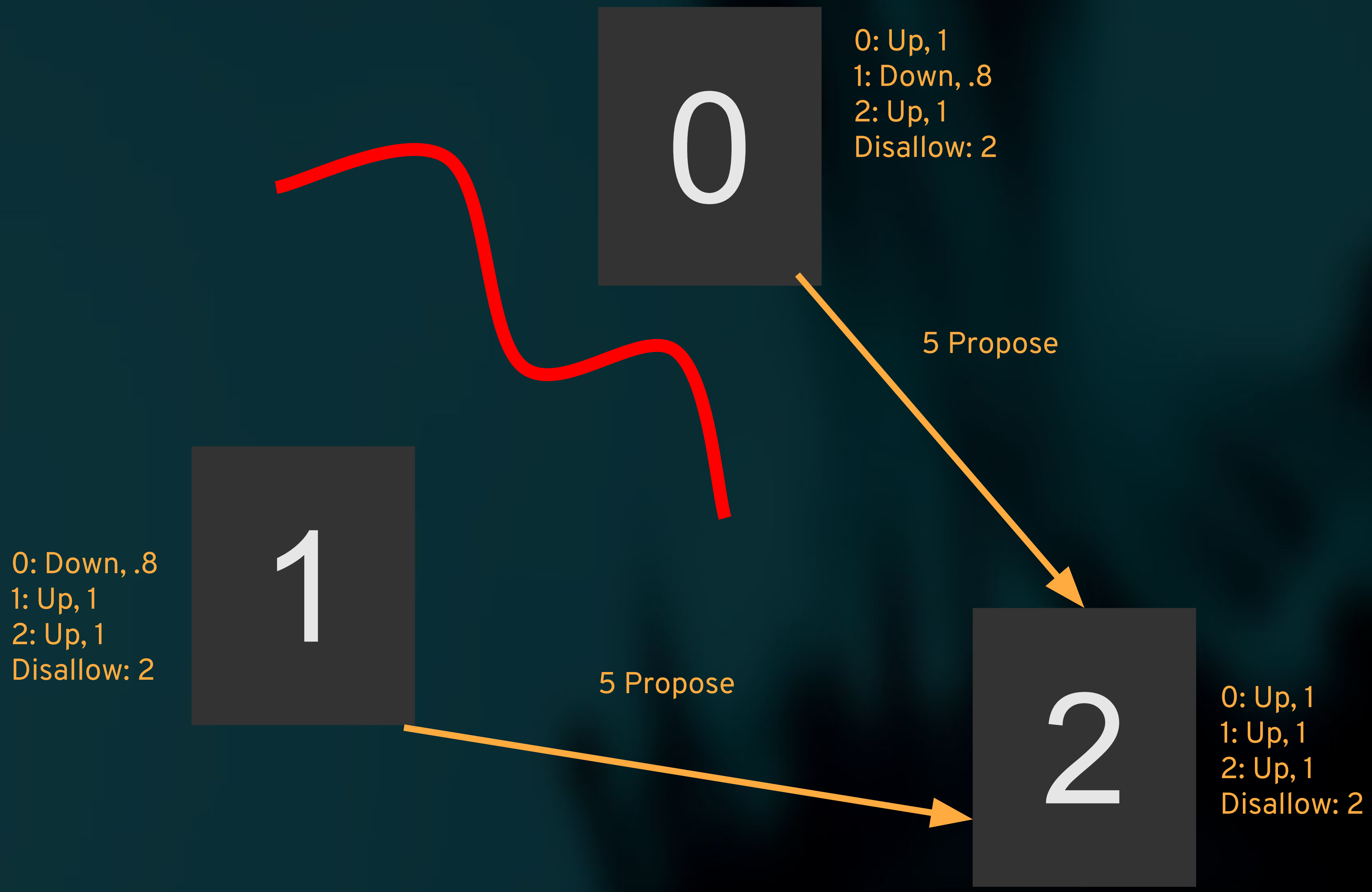


Leader Elections: Ack the propose, if it’s a better score



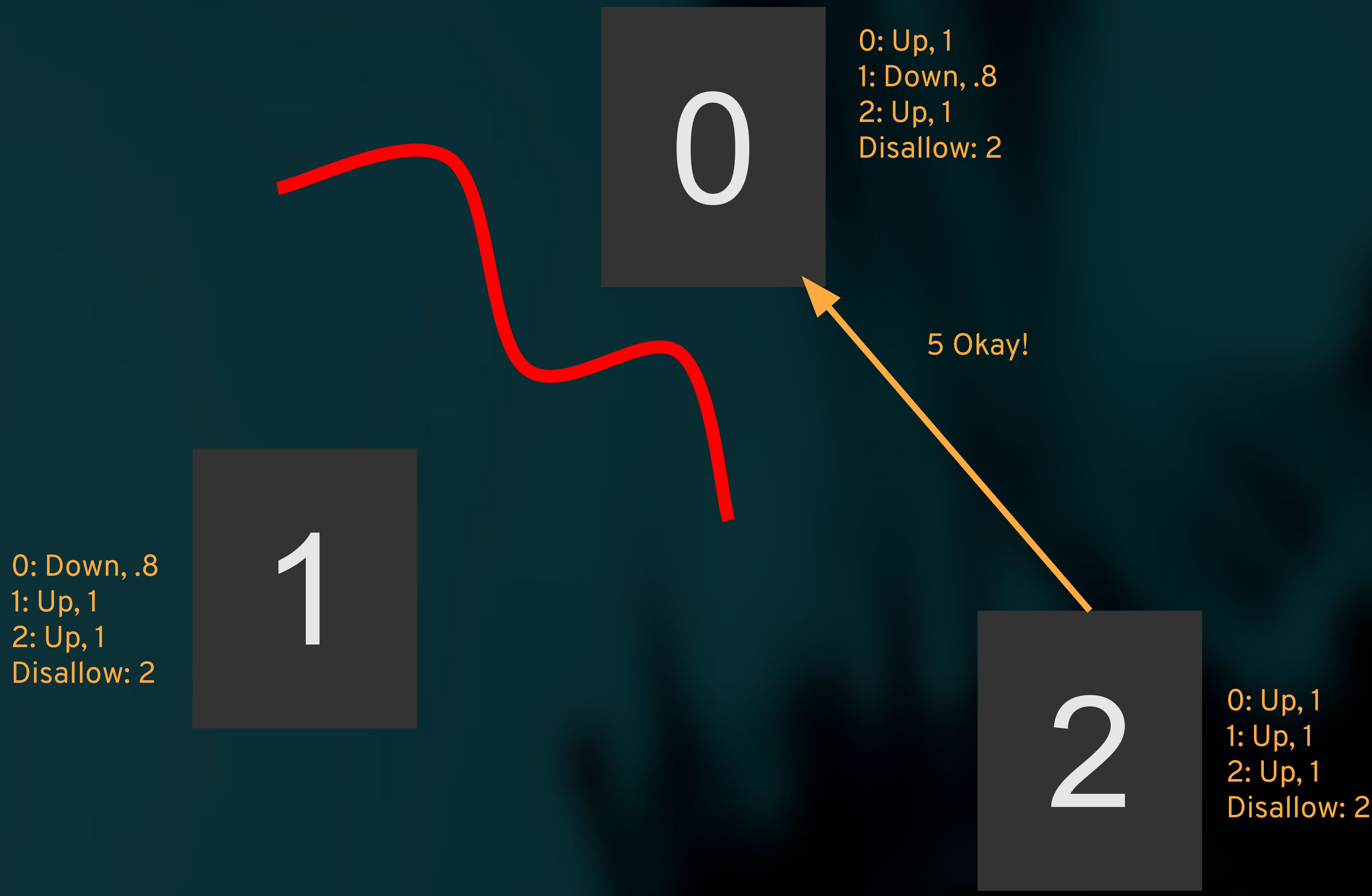
Leader Elections: Win if everybody acks you

ceph

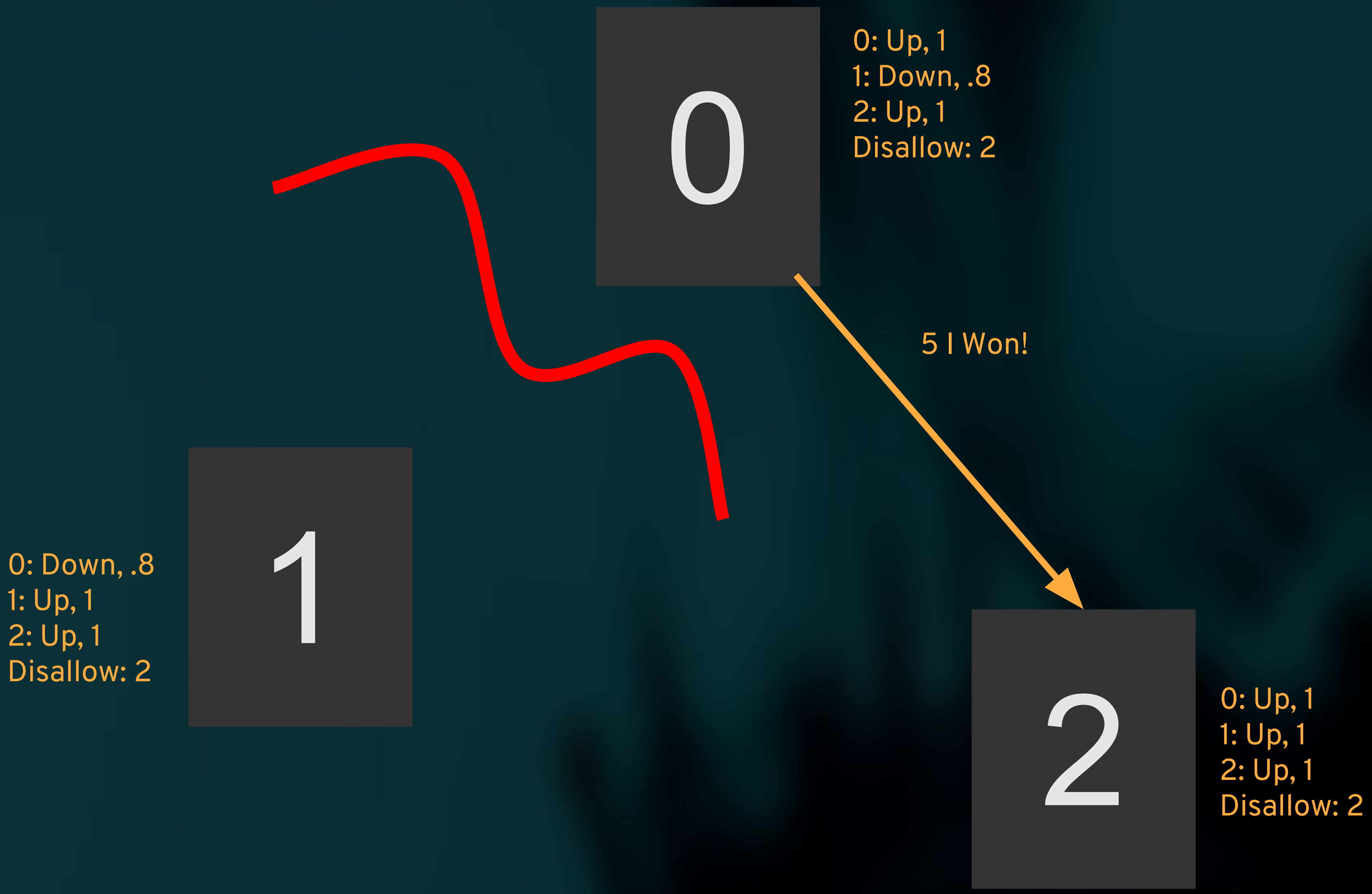


Leader Elections: You can disallow monitors as leaders

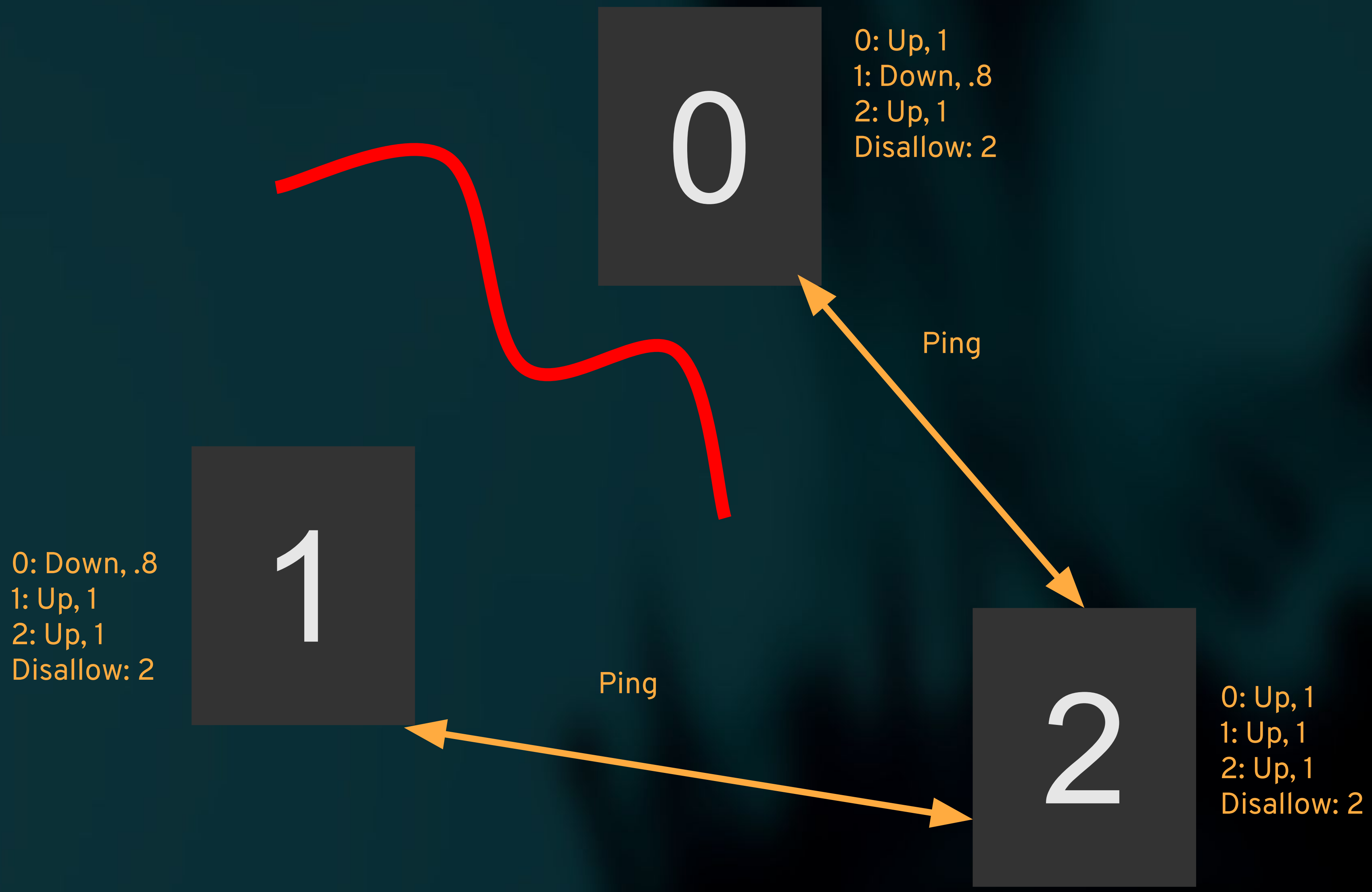
ceph



Leader Elections: You can disallow monitors as leaders

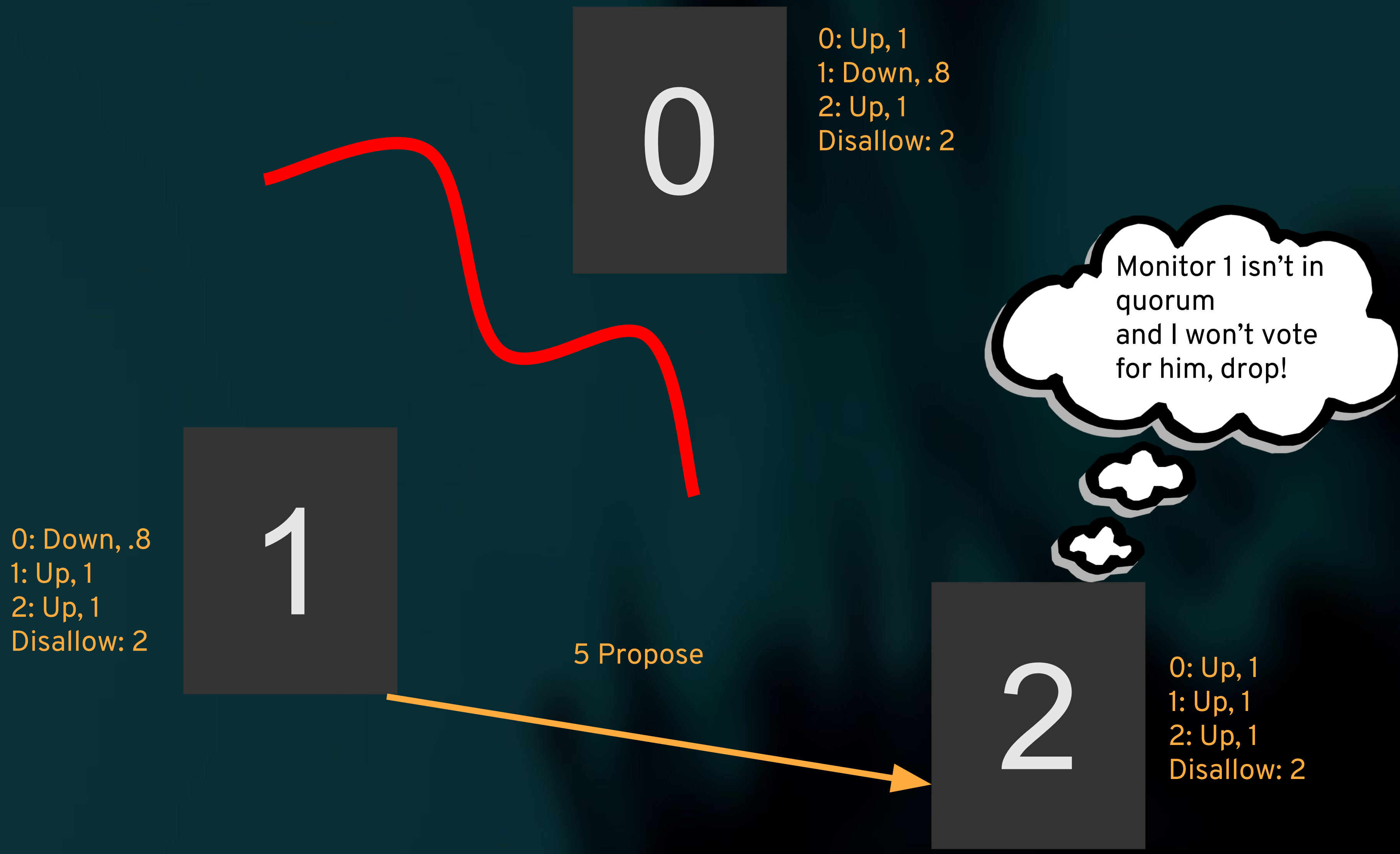


Leader Elections: You can disallow monitors as leaders



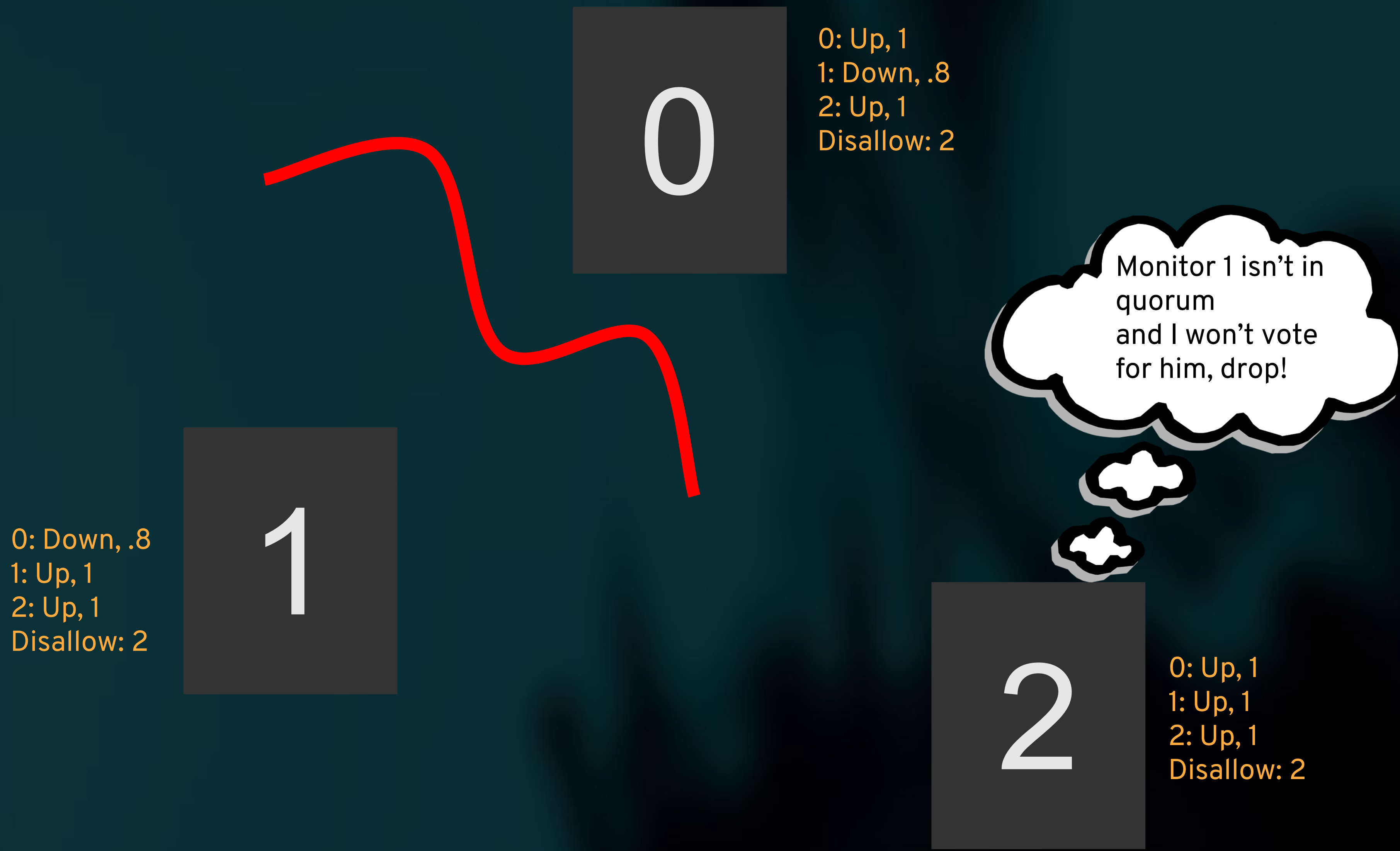
Leader Elections: Connectivity Mode Can Ignore Out-of-Quorum Peers

ceph



Leader Elections: Connectivity Mode Can Ignore Out-of-Quorum Peers

ceph



Leader Elections: Connectivity Mode Can Ignore Out-of-Quorum Peers

ceph

Connectivity mode: Monitors are happy now

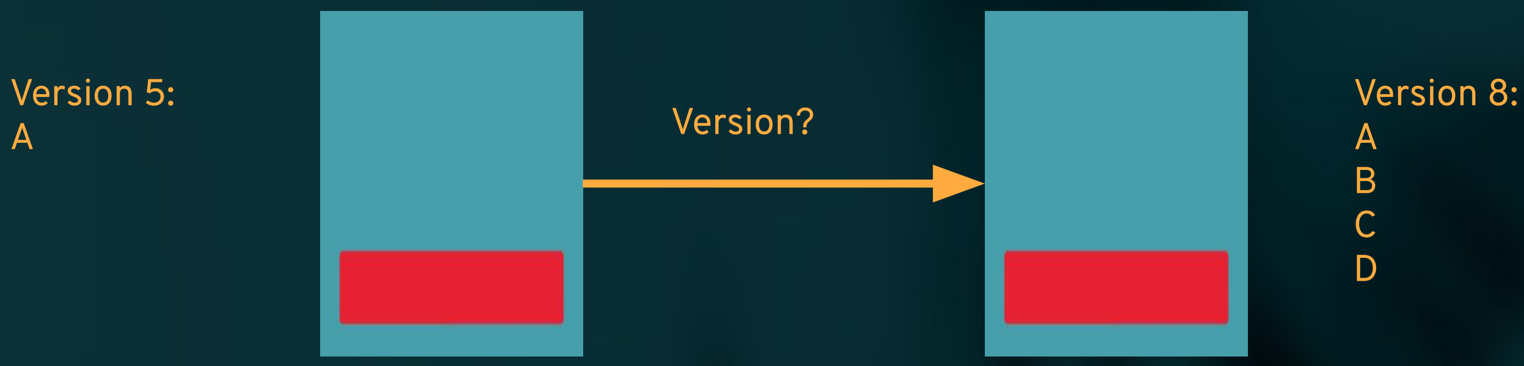
OSD Peering

ceph

OSD PEERING

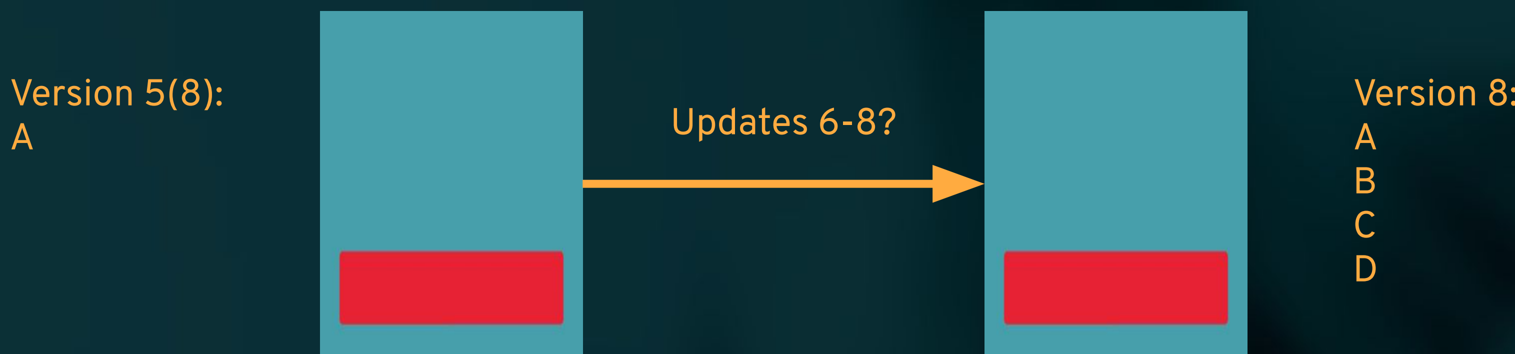
- Primary queries old peers for data version
- When versions mismatch, primary asks for update logs
- Update logs tell primary which objects it needs
- Primary asks old peers for newest copies of all changed objects

ceph

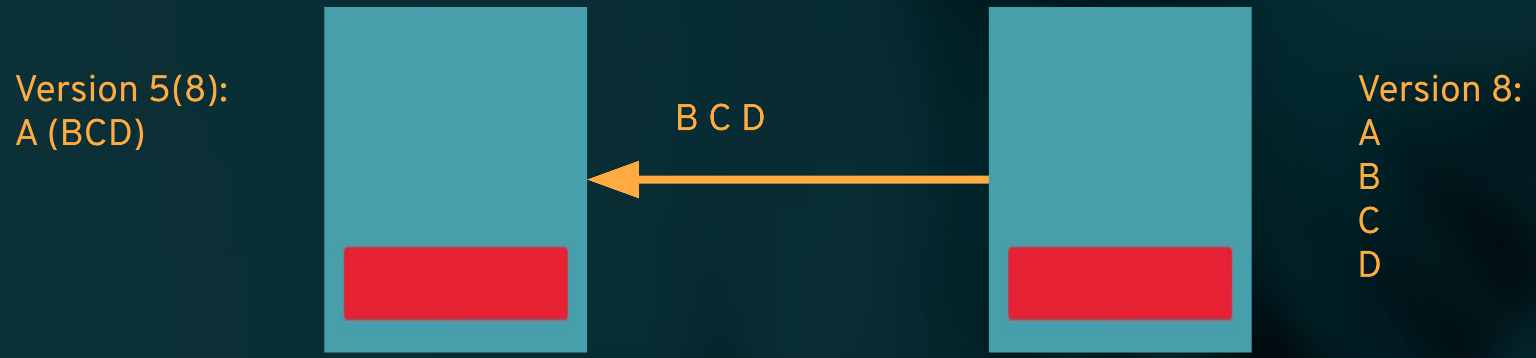


OSD Peering: Get Newest Version

OSD Peering: Get Newest Version

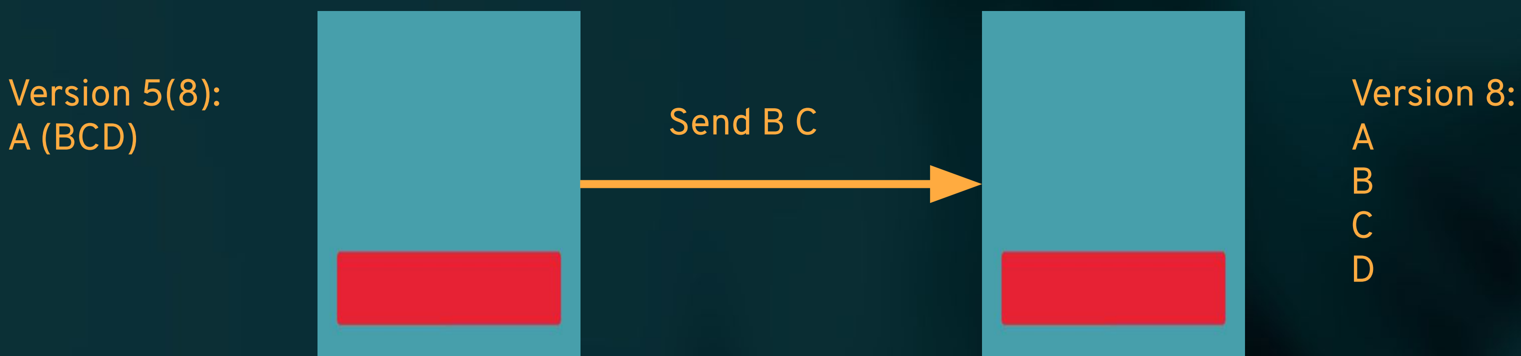


OSD Peering: Get Update Logs

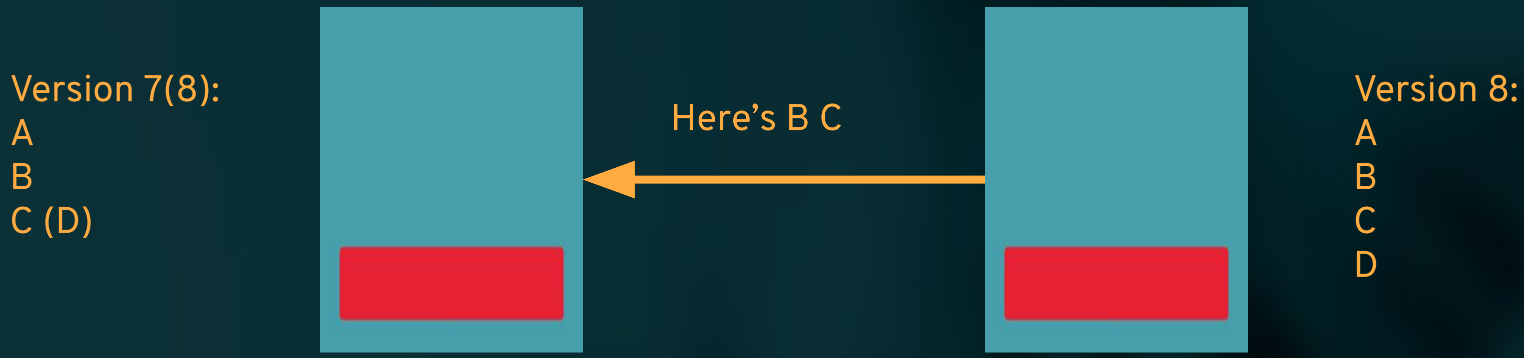


OSD Peering: Get Update Logs

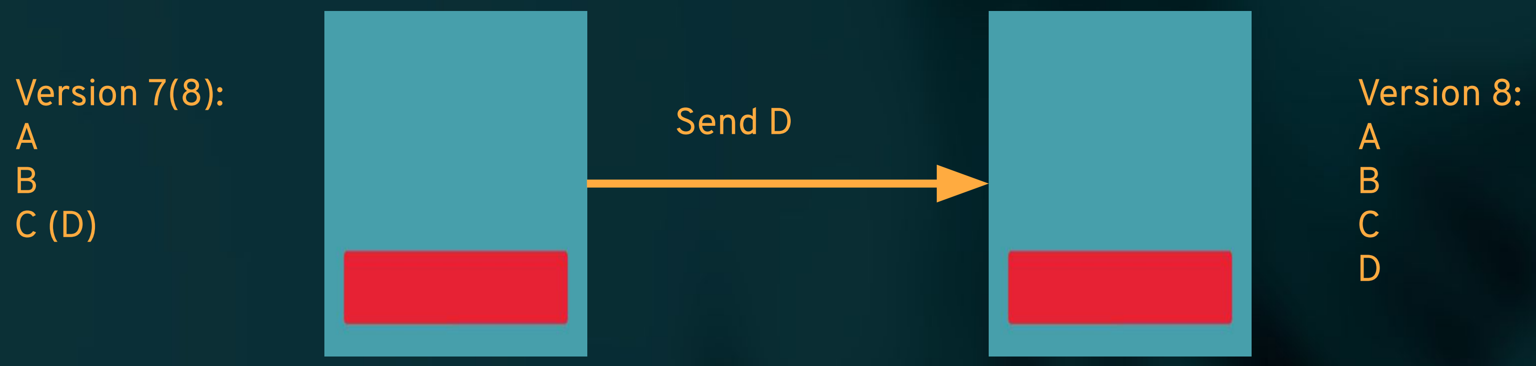
ceph



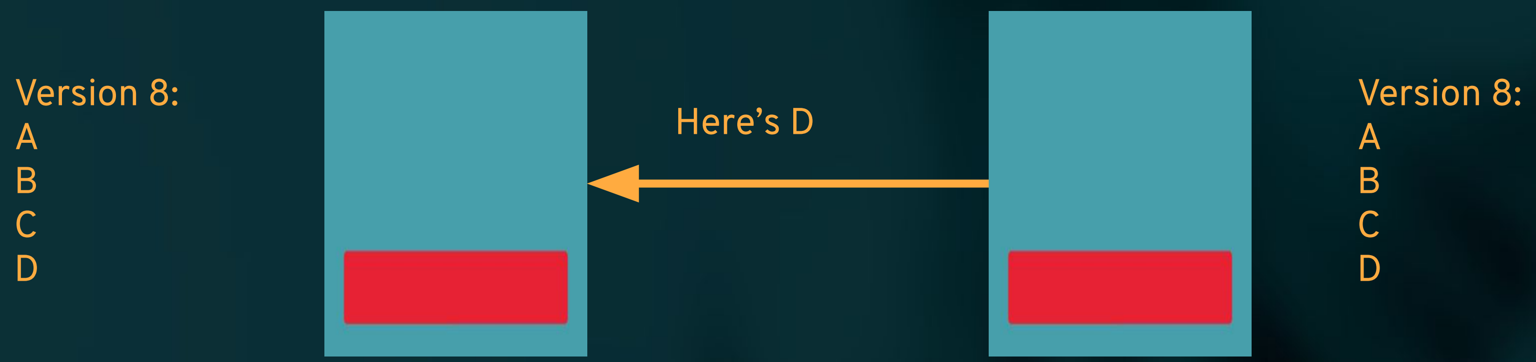
OSD Peering: Get Newest Objects



OSD Peering: Get Newest Objects



OSD Peering: Get Newest Objects

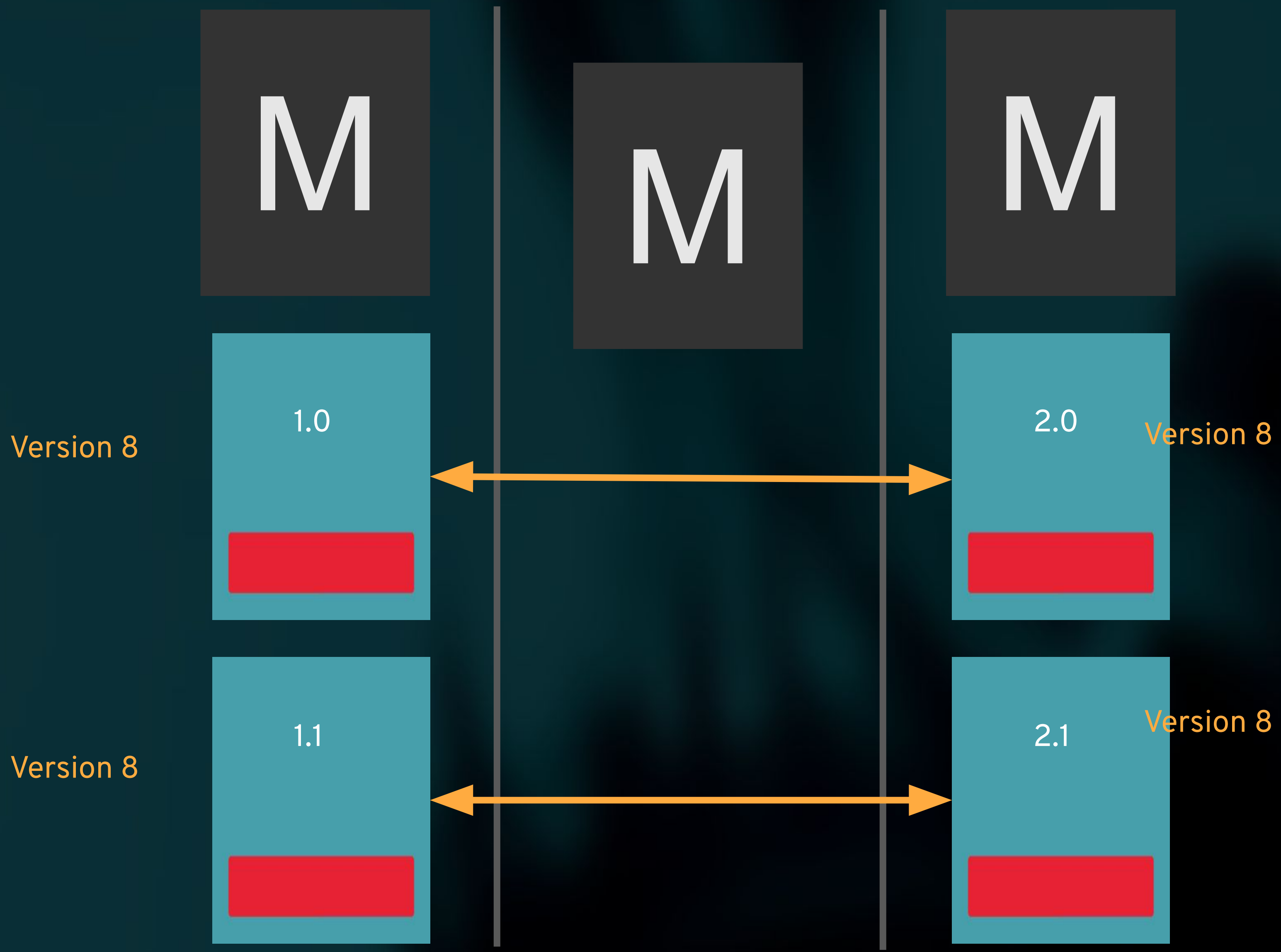


OSD Peering: Get Newest Objects

ceph

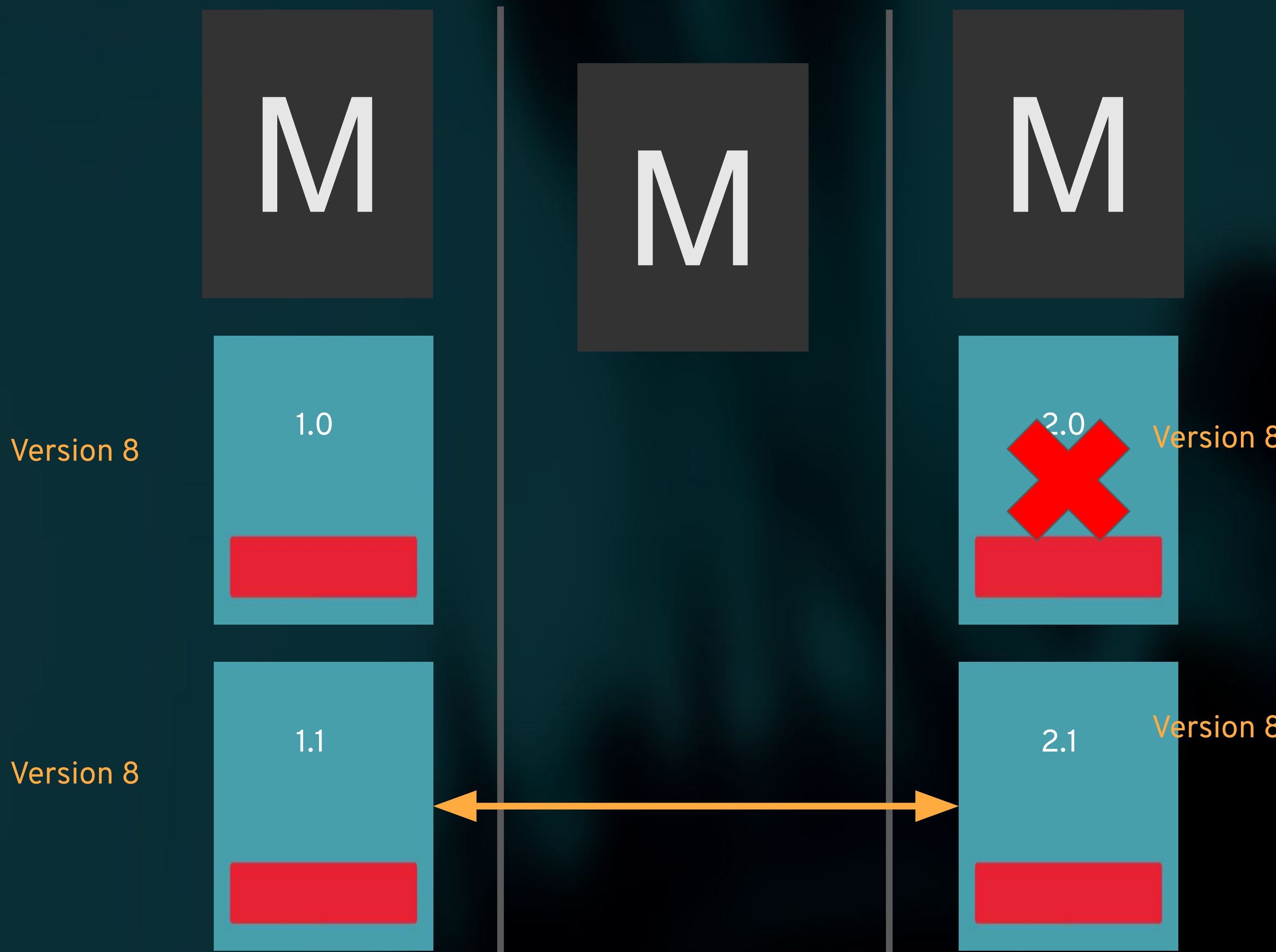
PEERING IN REAL LIFE

- Generally 3 copies of all data
- “min size” required to serve IO: usually 2
  - So we can recover even if one of these guys fails – if we went active with 1 OSD and it died we are out of luck!
- OSDs know specific versions (nobody else sees all updates)
- ...but monitors know updates were ALLOWED
  - This is how we identify the old peers to collect data from

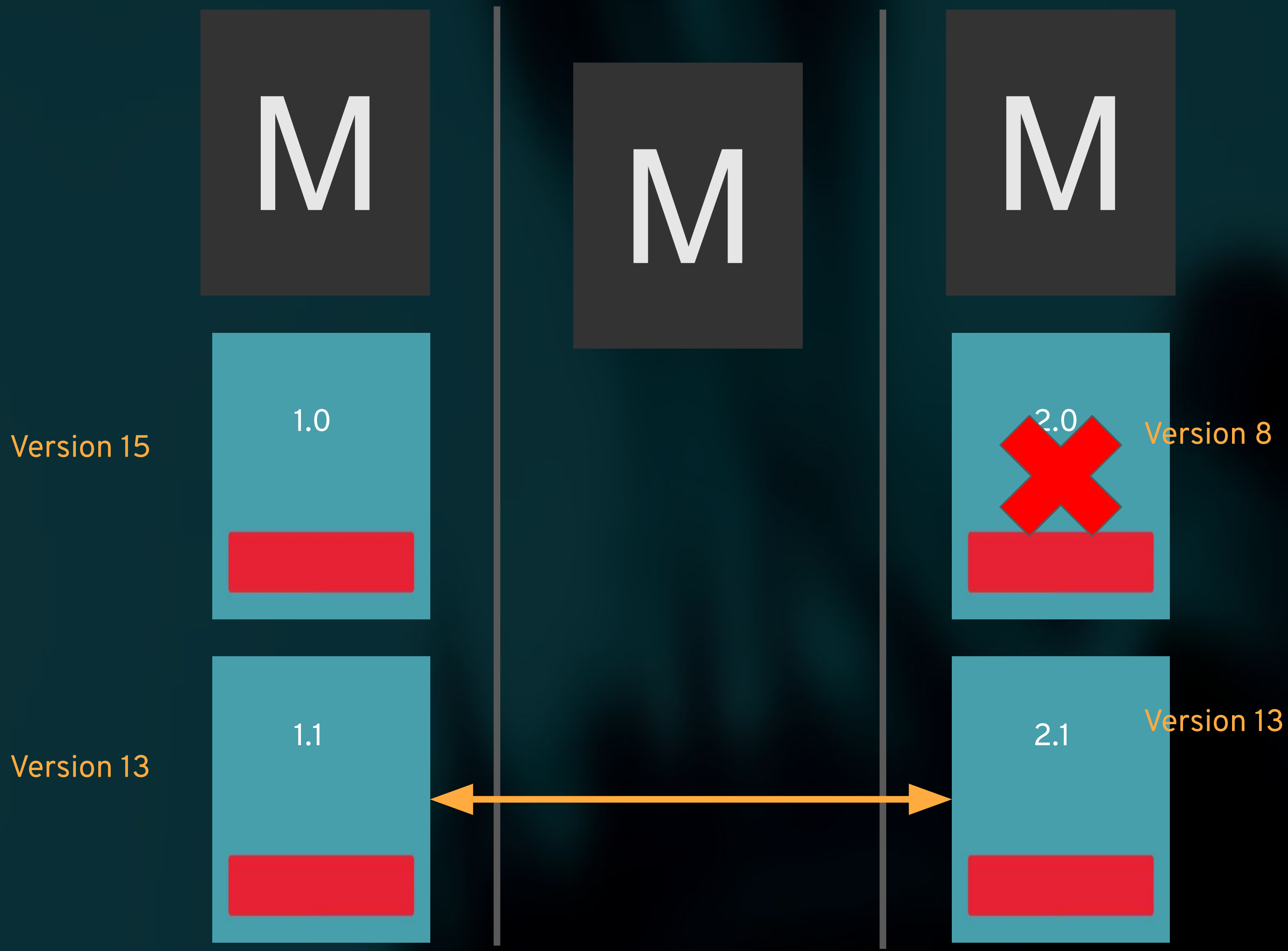


Peering: The Problem

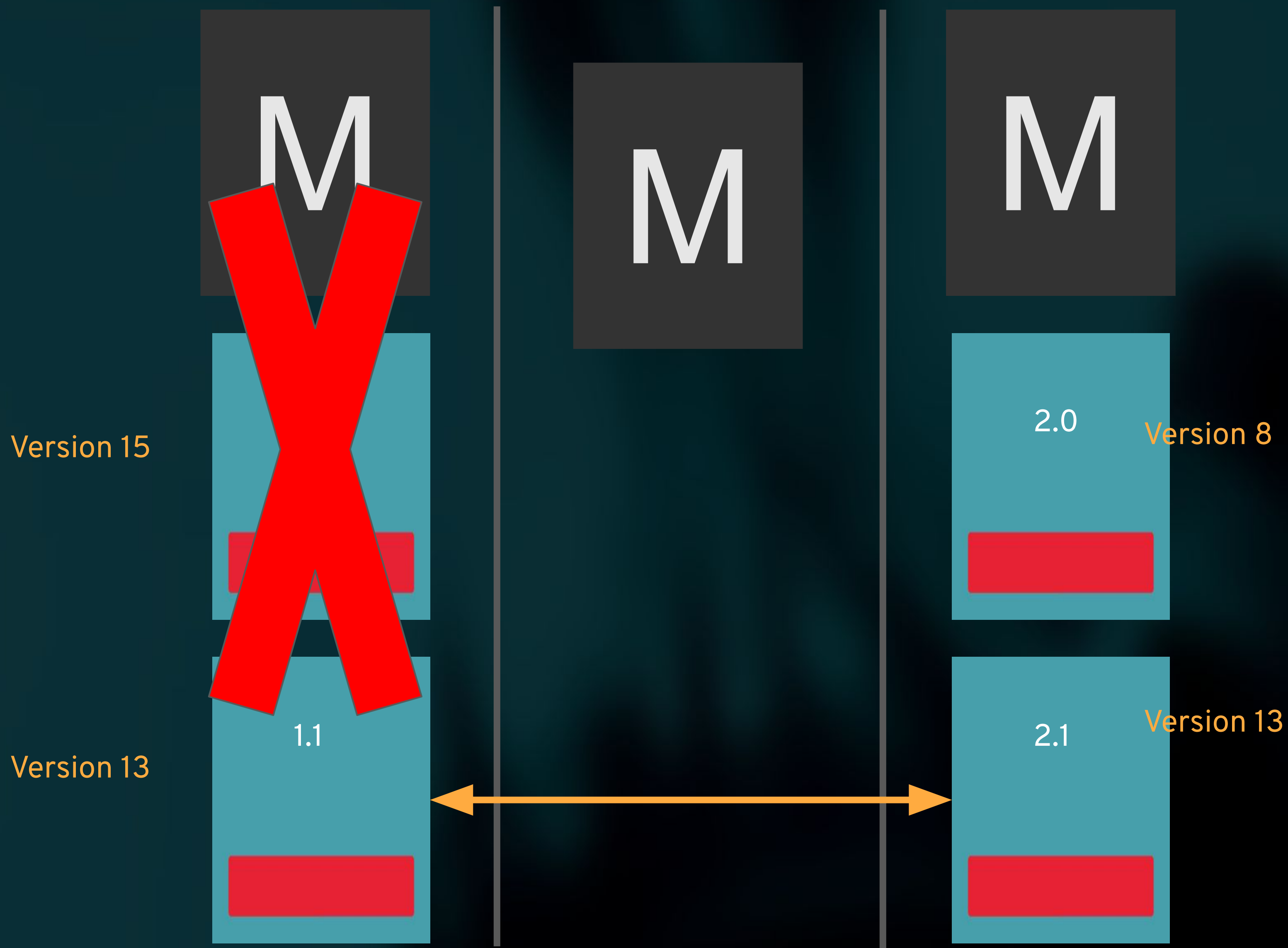
ceph



Peering: The Problem



Peering: The Problem



Peering: The Problem

DEAD DC? OUCH!

- If we lose 1 of 2 data centers, the odds are good that the survivor has old data (and if it does, it always knows it)
    - MOST of the data will be current, but we need ALL of it!
    - Being out-of-date because of 1 rebooting OSD server? BAD :(
- Ceph is VERY careful not to roll back in time by mistake

DEAD DC SOLUTION: STRETCH MODE

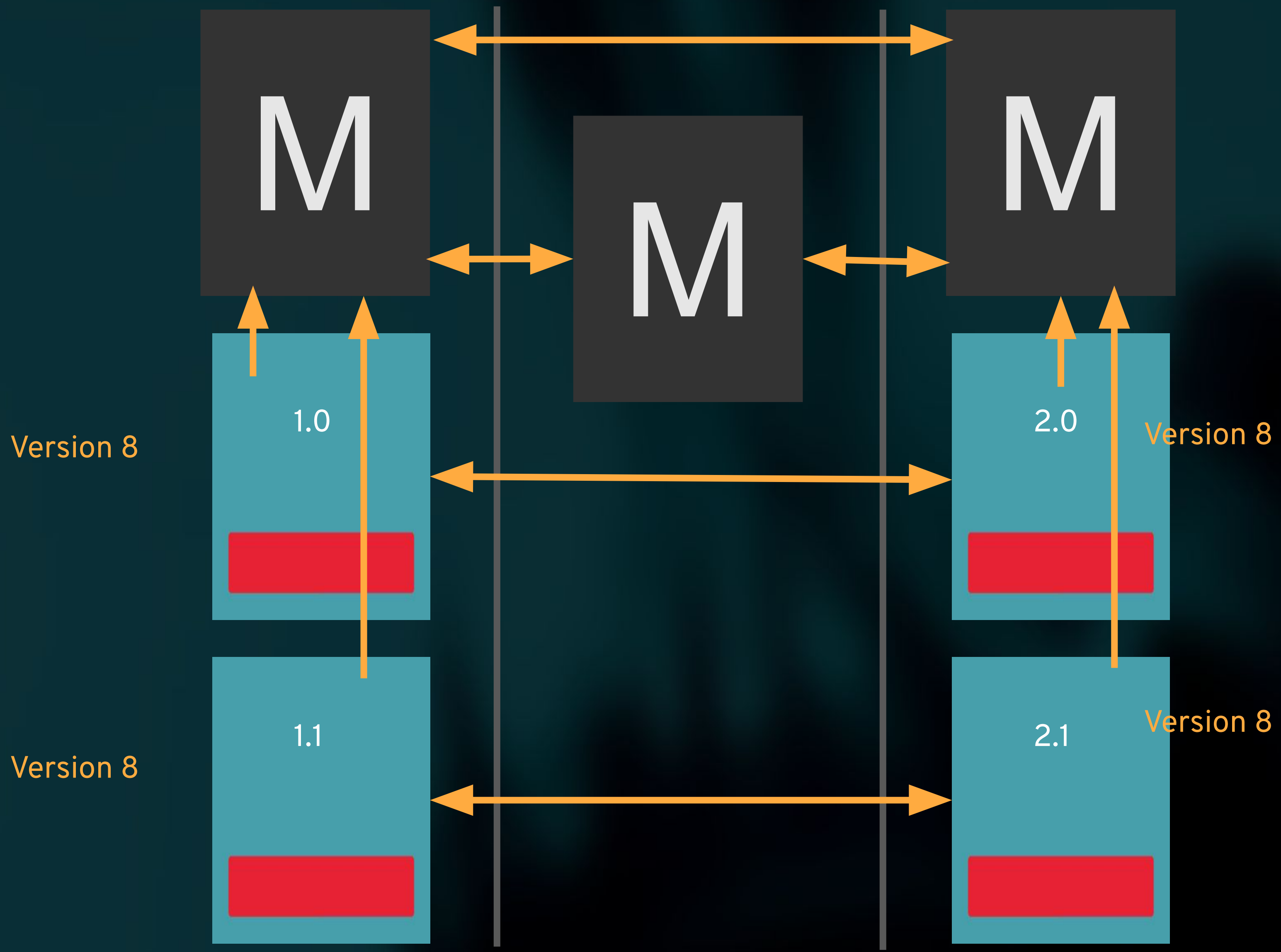
- Design target: 2 data centers, 2 copies in each
- Restrict OSD<->monitor communications to within a single DC (no “rogue” OSDs talking to the tiebreaker monitor to stay alive-but-inaccessible)
- Extend the peering algorithm: an “acting set” must contain OSDs from multiple data centers to serve IO
  - Ensures survival of an OSD loss AND a data center loss!

This is in-progress

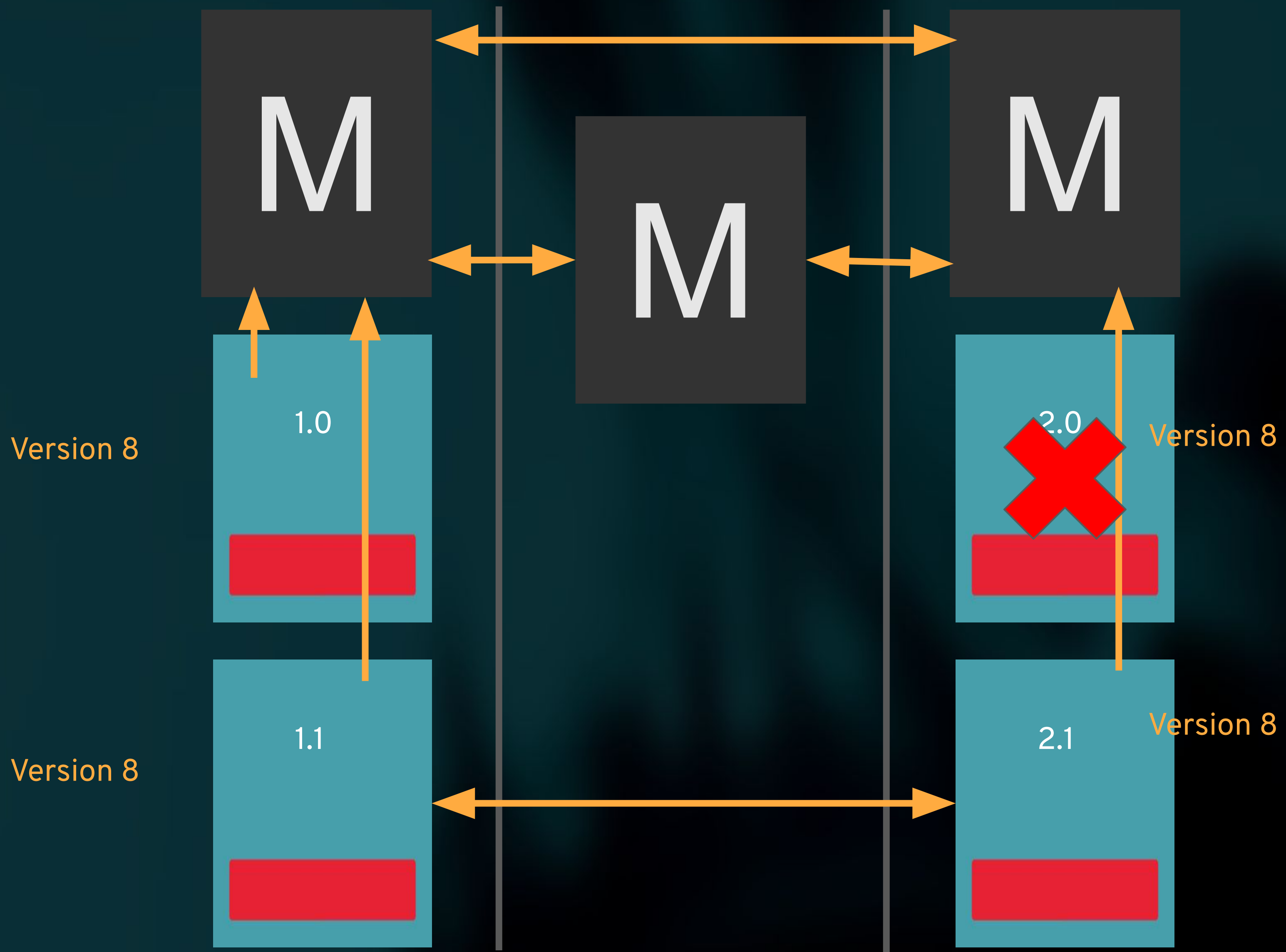
STRETCH MODE: HANDLING DC FAILURE

- OSDs only talk to their own-DC monitor(s)
- Require OSDs from multiple DCs
- Missing DCs? Missing data access :(
  - But NOT data loss! :)
- For 2-DC clusters, if a whole DC goes down:
  - Surviving DC and tiebreaker monitor declare DC dead
  - Remove multi-DC requirement from peering
  - Go active — we know we have newest data because every write had to go through our DC!

ceph

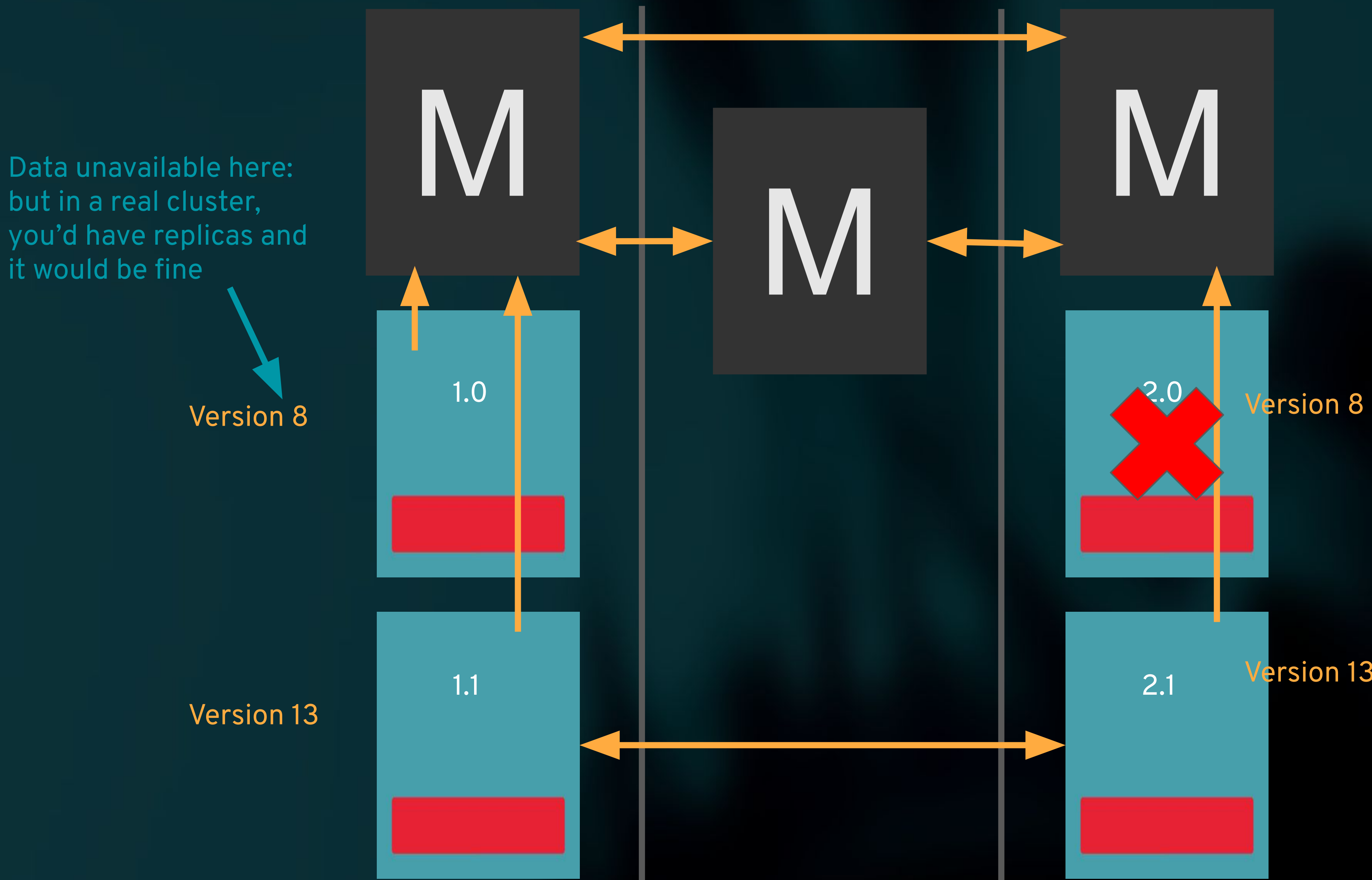


Stretch Cluster Peering: New Rules



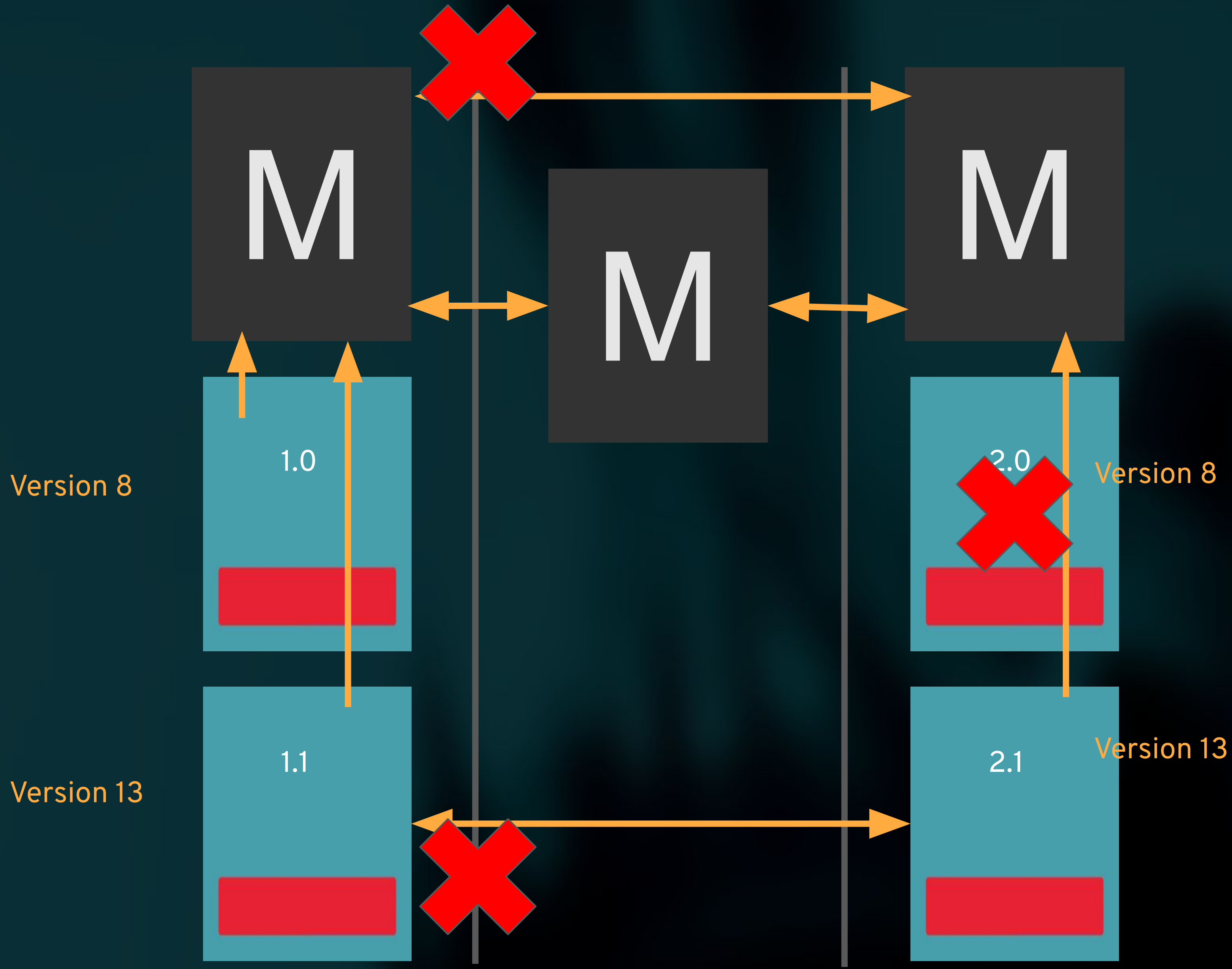
Stretch Cluster Peering: New Rules

ceph

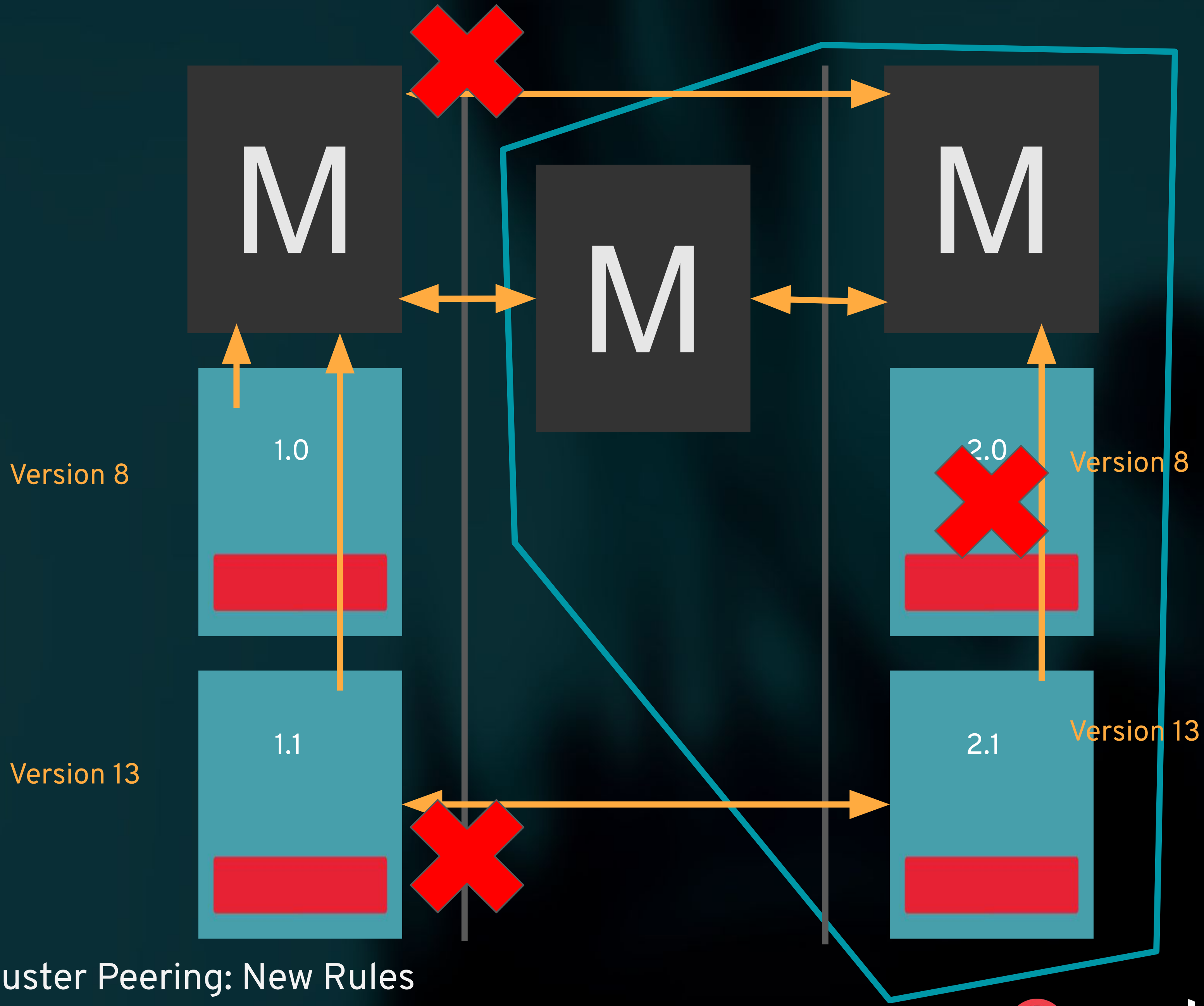


Stretch Cluster Peering: New Rules

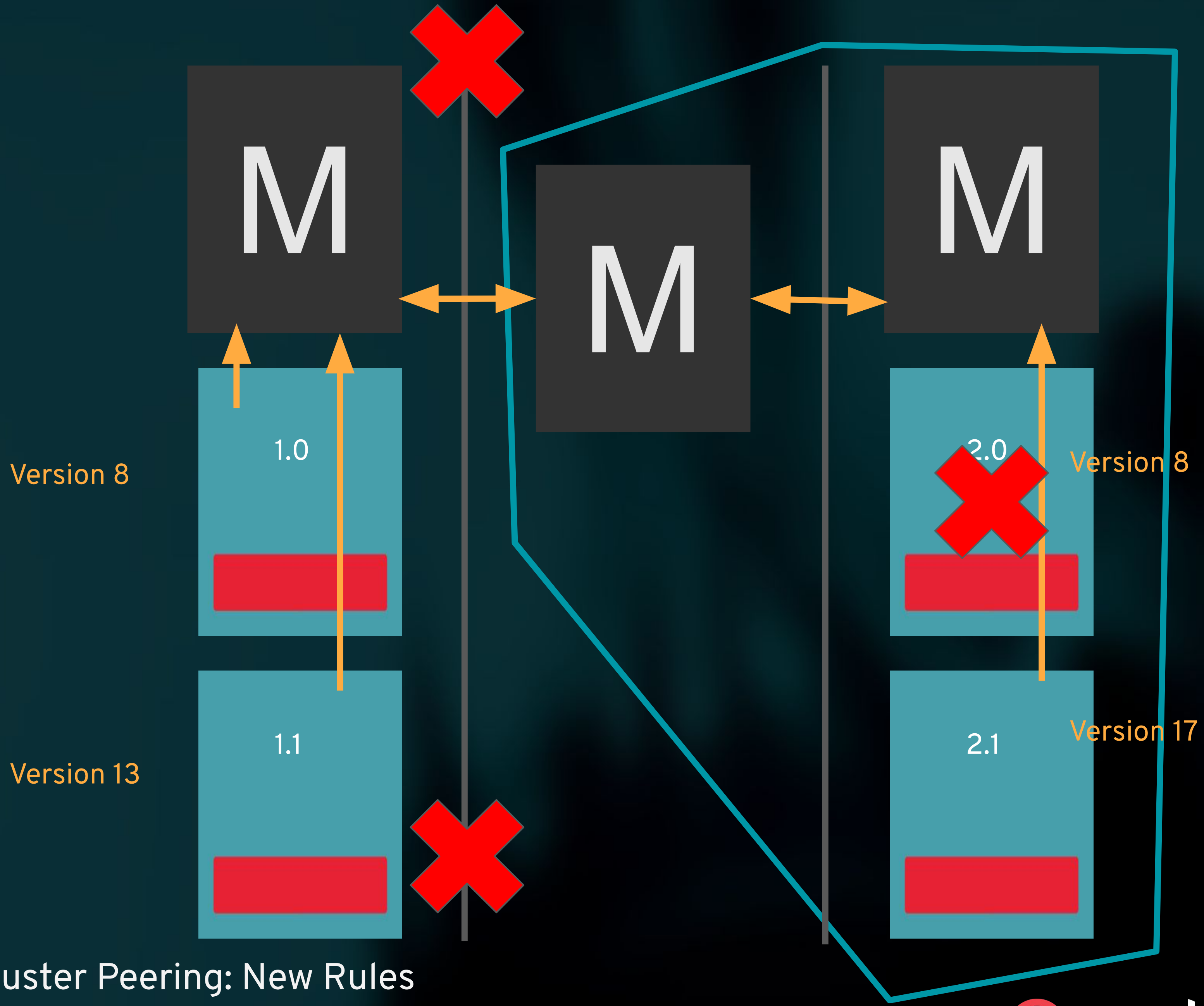
ceph



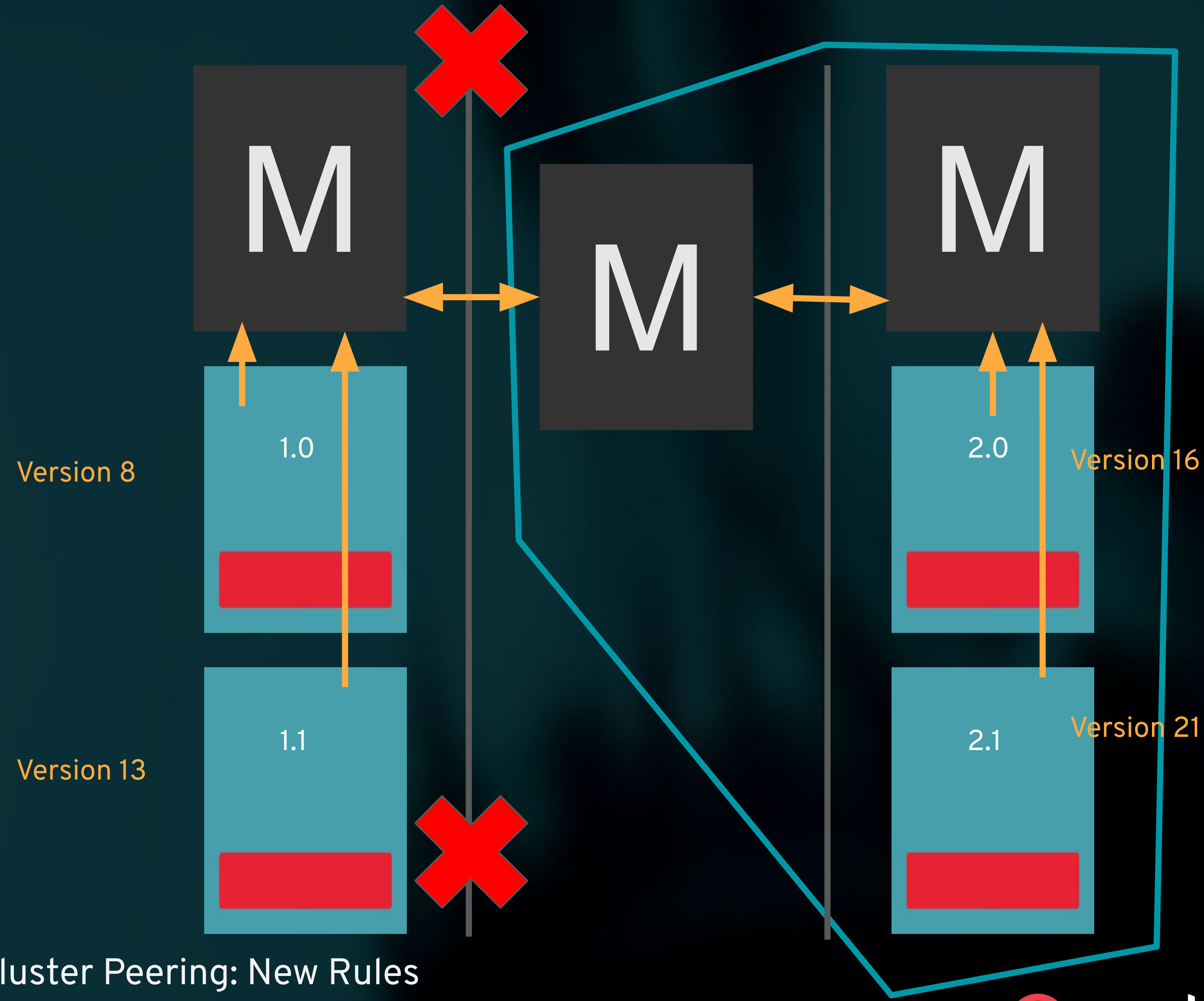
Stretch Cluster Peering: New Rules



Stretch Cluster Peering: New Rules



Stretch Cluster Peering: New Rules



Stretch Cluster Peering: New Rules

ceph

The End

Questions?