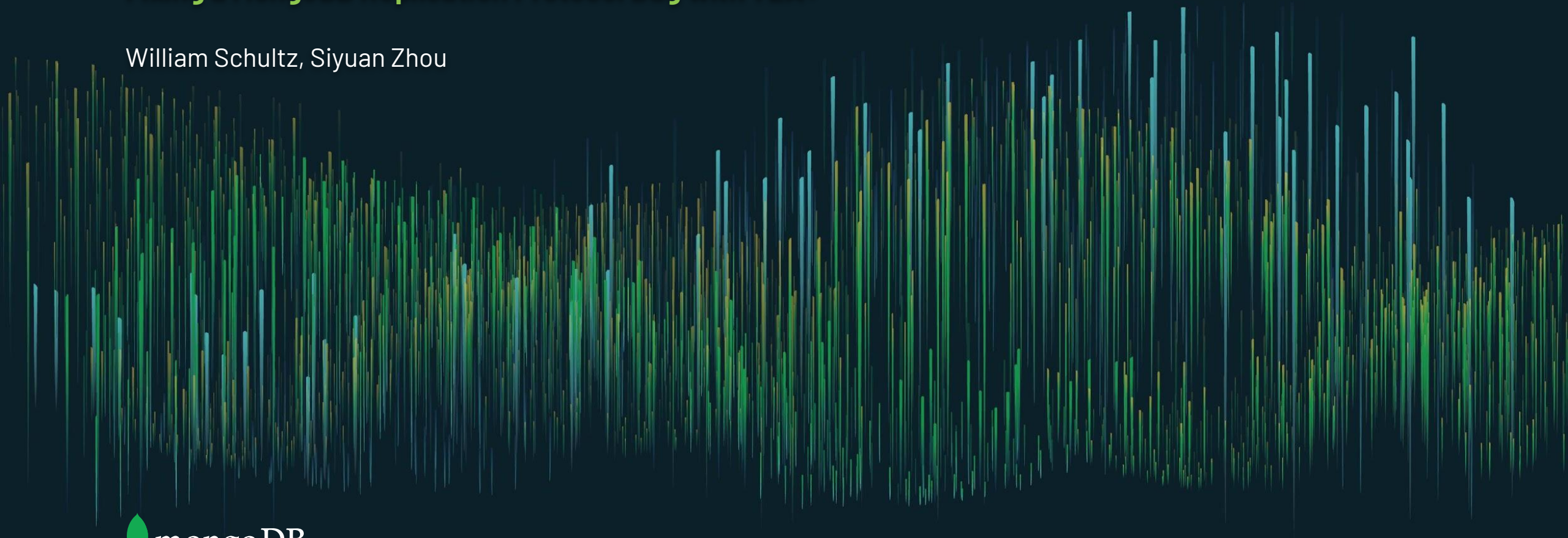
A Bug's Life

Fixing a MongoDB Replication Protocol Bug with TLA+

William Schultz, Siyuan Zhou



mongoDB.

Talk Outline

- MongoDB Background
- Life of a Replication Bug
- Specifying the Replication Protocol in TLA+
- Model Checking the Specification
- Takeaways
  - *Without a formal model, it's nearly impossible to get a complex distributed protocol right. TLA+ and TLC are tools that make this possible for practicing software engineers.*

Background

Background

MongoDB

- MongoDB is a document oriented database
- A MongoDB database consists of a set of collections, where a collection is a set of unique documents e.g.

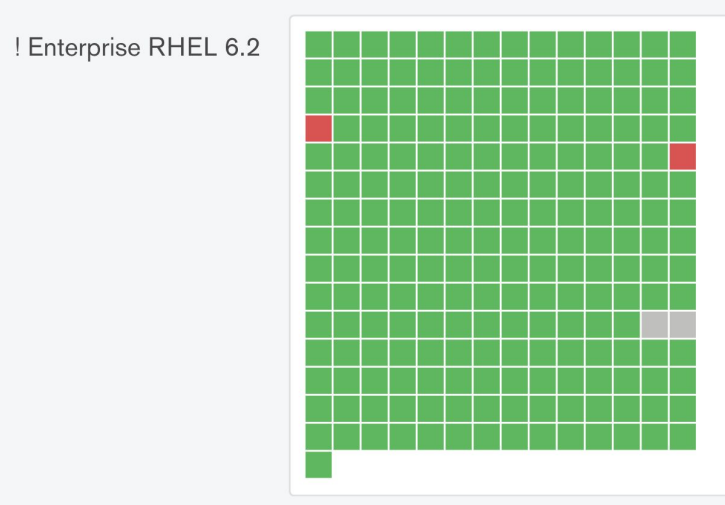
```
{
    _id: 1,
    name: “Will”,
    company: “MongoDB”,
    age: 25
}
```

Background

MongoDB Test Infrastructure

- We have an extensive and mature testing infrastructure
- 1000s of hours of testing are run on new commits every day
  - Includes unit/integration tests, randomized fuzzing, concurrency tests, Jepsen, etc.

Makespan 1h 42m 19s
Time Spent 826h 19m 22s



Background

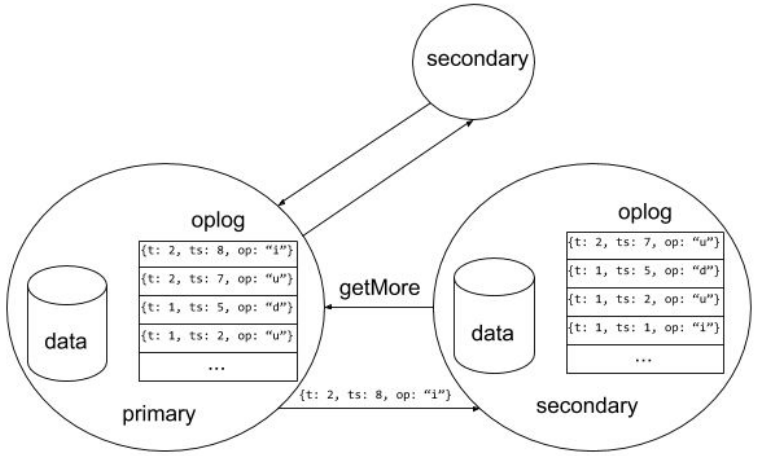
Replication in MongoDB

- MongoDB provides the ability to run a database as a ***replica set***
- This is a set of MongoDB nodes that coordinate to provide high availability using a consensus protocol

Background

Replication in MongoDB

- Replica sets use a consensus protocol similar to Raft
- There is a primary node that inserts writes into the *oplog*
- Secondaries fetch log entries from other nodes and apply them



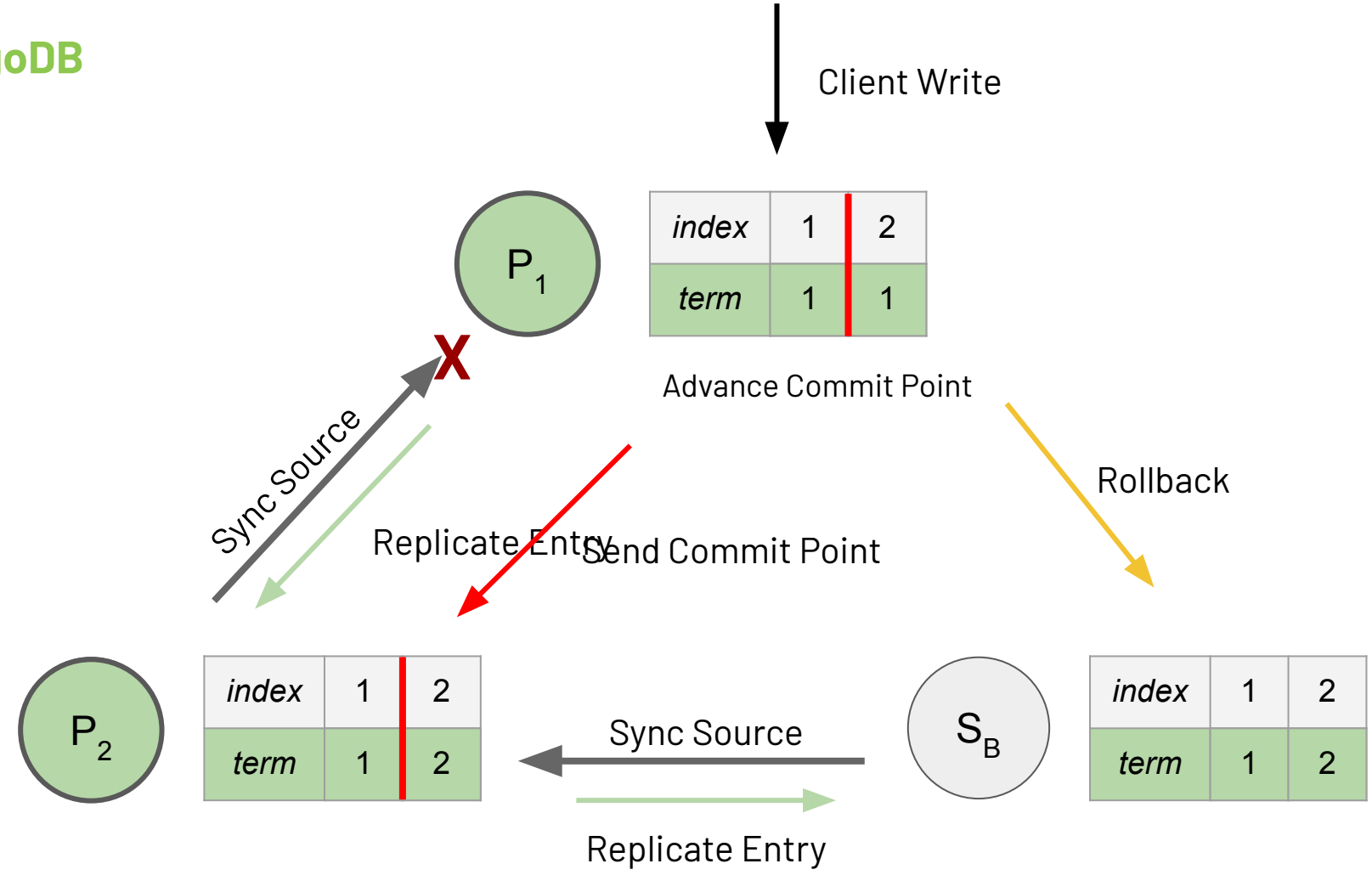
Background

Replication in MongoDB

- Protocol is leader based
- Replica set leaders are totally ordered by *term*
  - The *term* is a monotonic counter maintained on each node

Background

Replication in MongoDB



Background

Replication in MongoDB

- Replica set concepts to keep in mind:
  - sync source
  - commit point
  - rollback
  - term
  - “branch of log history”

Life of a Replication Bug

Bug Timeline

- Series of safety and liveness bugs in replication protocol
- Stemmed from one bug found in 2016

Bug Timeline

1. 2016 - Heartbeat Commit Point Propagation (*Safety*)
2. 2018 - No Available Sync Source (*Liveness*)
3. 2019 - Sync Source Cycles (*Liveness*)
4. 2019 - Commit Point Held Back on Stale Nodes (*Liveness*)
5. 2019 - Initial Solution Fails in 5 Node Replica Set (*Safety*)

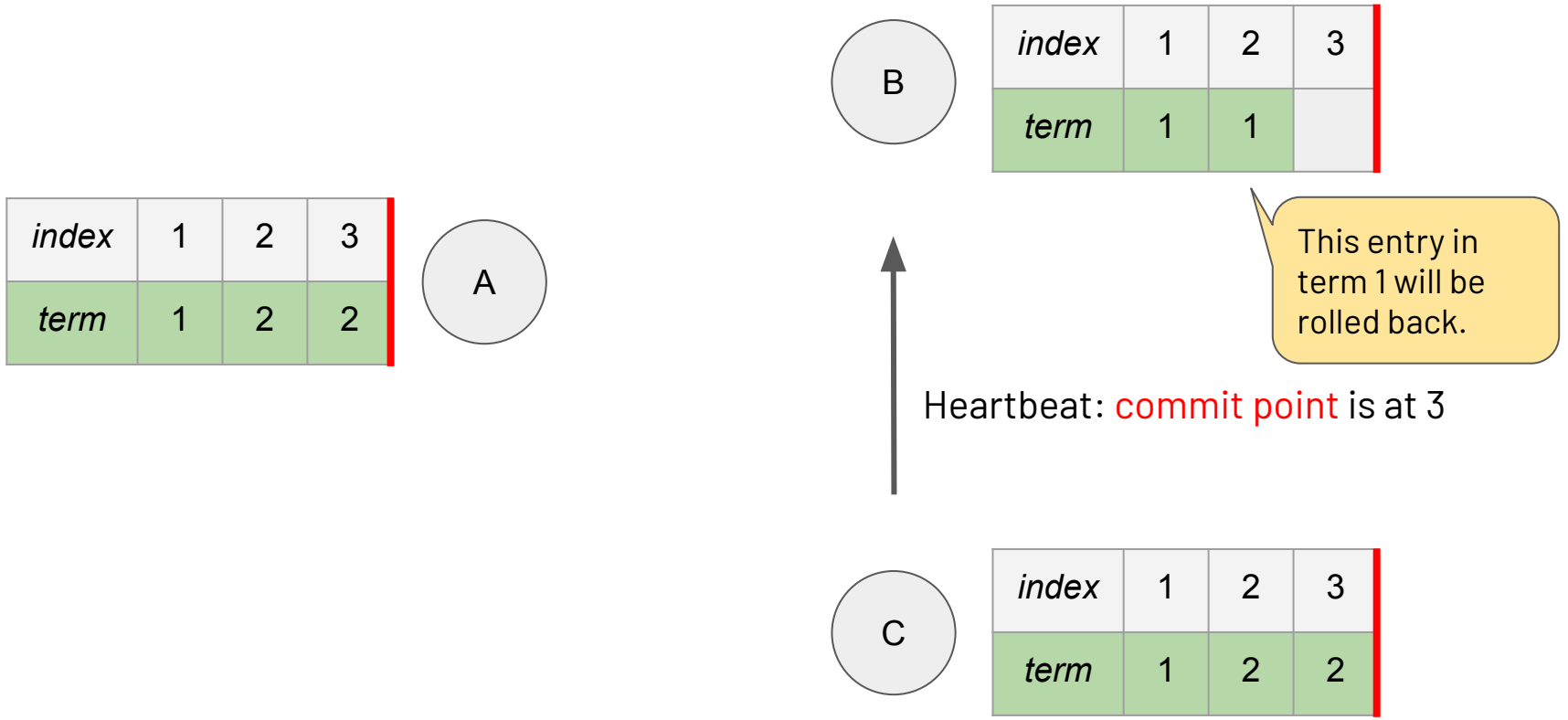
Bug 1: Heartbeat Commit Point Propagation

2016

- Correctness bug found in November 2016 by a replication team engineer
- Allowed for nodes to erroneously mark log entries as “committed”
  - Consequence: client could read data it thinks is durable, even if it isn’t

Bug 1: Heartbeat Commit Point Propagation

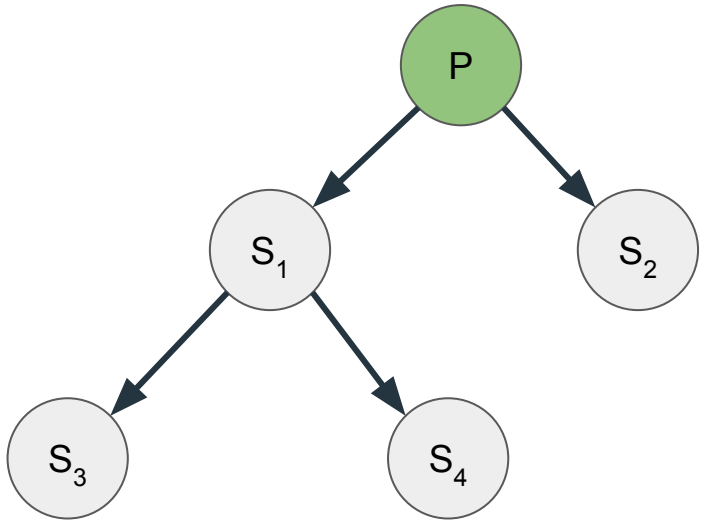
2016



Bug 1: Heartbeat Commit Point Propagation

2016

- *Solution*: Only update commit point via sync source spanning tree
  - Guarantees commit points are sent between nodes on same branch of log history



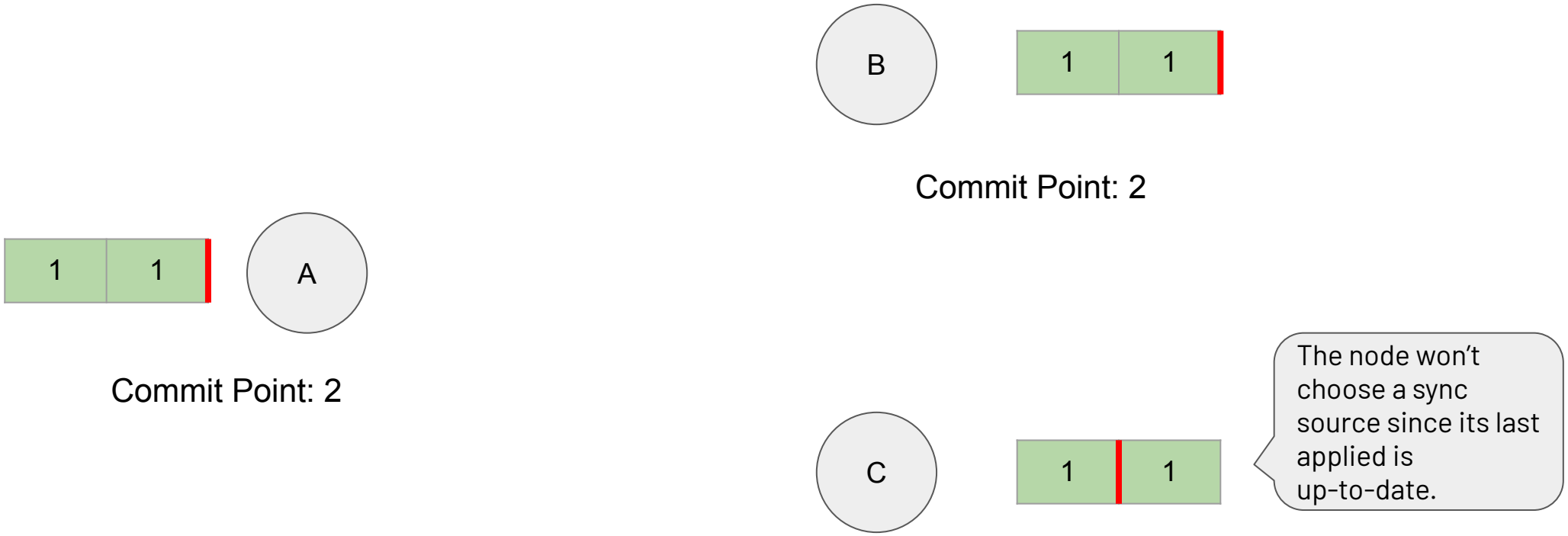
Bug 2: No Available Sync Source

2018

- New liveness bug found in February 2018
  - Discovered in our test infrastructure

Bug 2: No Available Sync Source

2018



Bug 2: No Available Sync Source

2018

- *Solution*: Relax rule sync source selection rules
  - Allow nodes to sync from someone with a higher commit point if logs are the same

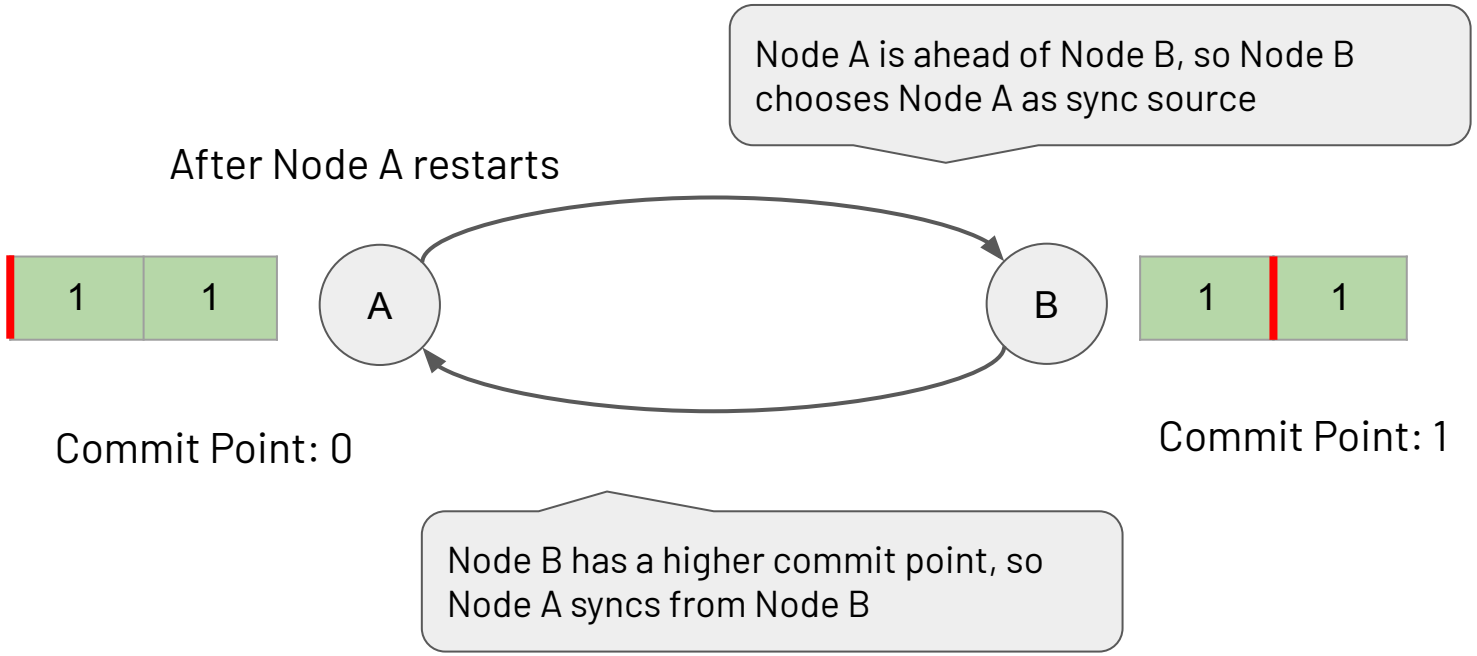
Bug 3: Sync Source Cycle

2019

- Liveness bug of a new variety
- Nodes may get into sync source cycles
  - Prevents them from ever syncing new log entries
- Consequence of the previous alteration to sync source selection rules

Bug 3: Sync Source Cycle

2019



Bug 3: Sync Source Cycle

2019

- *Solution*: Rethink commit point propagation
- Key idea: learn commit point if it is on your branch of history
  - Via heartbeats or sync source

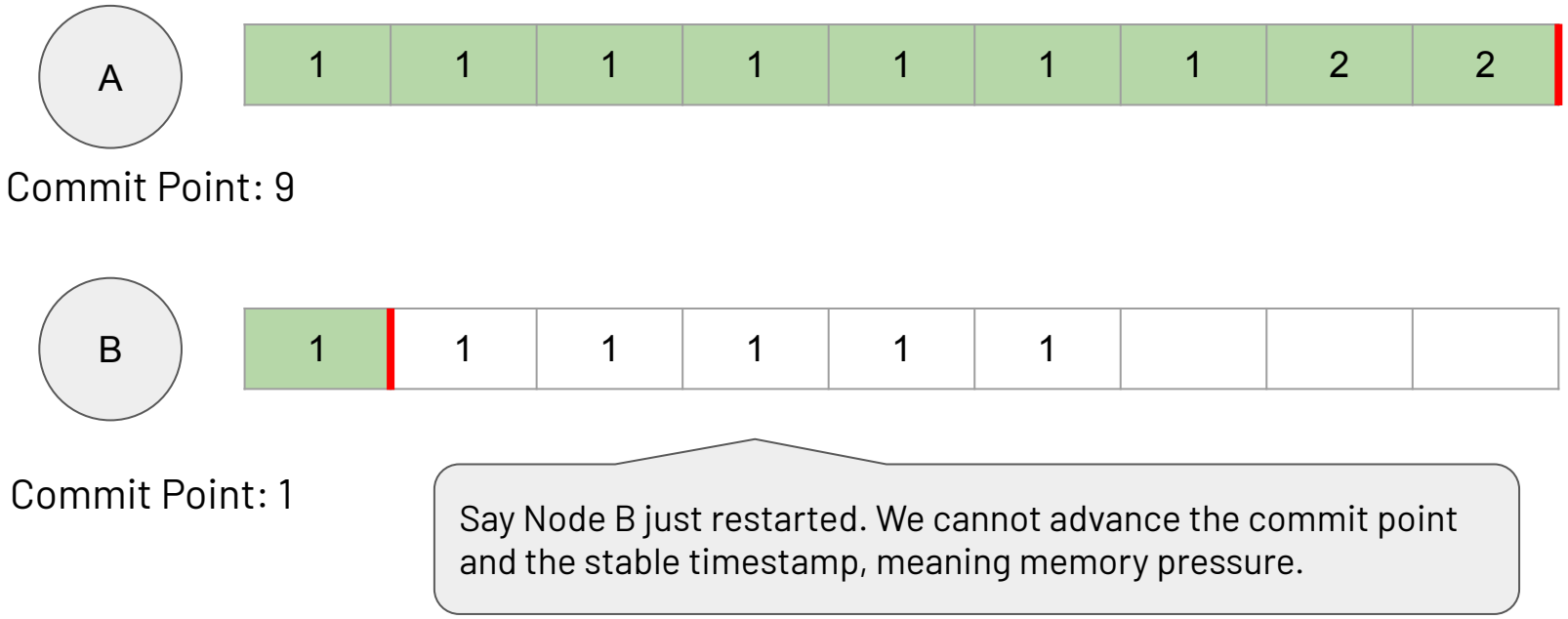
Bug 4: Commit Point Held Back on Stale Nodes

2019

- New liveness bug noticed in February 2019
- Stale nodes may not be able to advance their commit point

Bug 4: Commit Point Held Back on Stale Nodes

2019



Bug 4: Commit Point Held Back on Stale Nodes

2019

- *Solution*
  - Relax condition for commit point updates from your sync source
  - Sync source guaranteed to be on the same history branch

Bug 5: Initial Solution Fails in 5 Node Replica Set

2019

- The original solution to Bug 1 was believed safe, even though it had liveness issues
  - Discovered that the solution was not safe in replica sets with > 4 nodes
  - Could lead to nodes erroneously committing log entries in certain cases

Specifying the Protocol in TLA+

MongoDB Replication TLA+ Spec

Variables

```
\* The server's term number.
VARIABLE globalCurrentTerm
\* The server's state (Follower, Candidate, or Leader).
VARIABLE state
\* The commit point learned by each server.
VARIABLE commitPoint
\* A sequence of log entries.
VARIABLE log
\* The current sync source of each server, if it has one.
VARIABLE syncSource
```

MongoDB Replication TLA+ Spec

Initial State Predicate

```
\* Define initial values for all variables.
Init == /\ globalCurrentTerm = 0
        /\ state             = [i \in Server |-> Follower]
        /\ commitPoint       = [i \in Server |-> [term |-> 0, index |-> 0]]
        /\ syncSource        = [i \in Server |-> Nil]
        /\ log               = [i \in Server |-> << >>]
```

MongoDB Replication TLA+ Spec

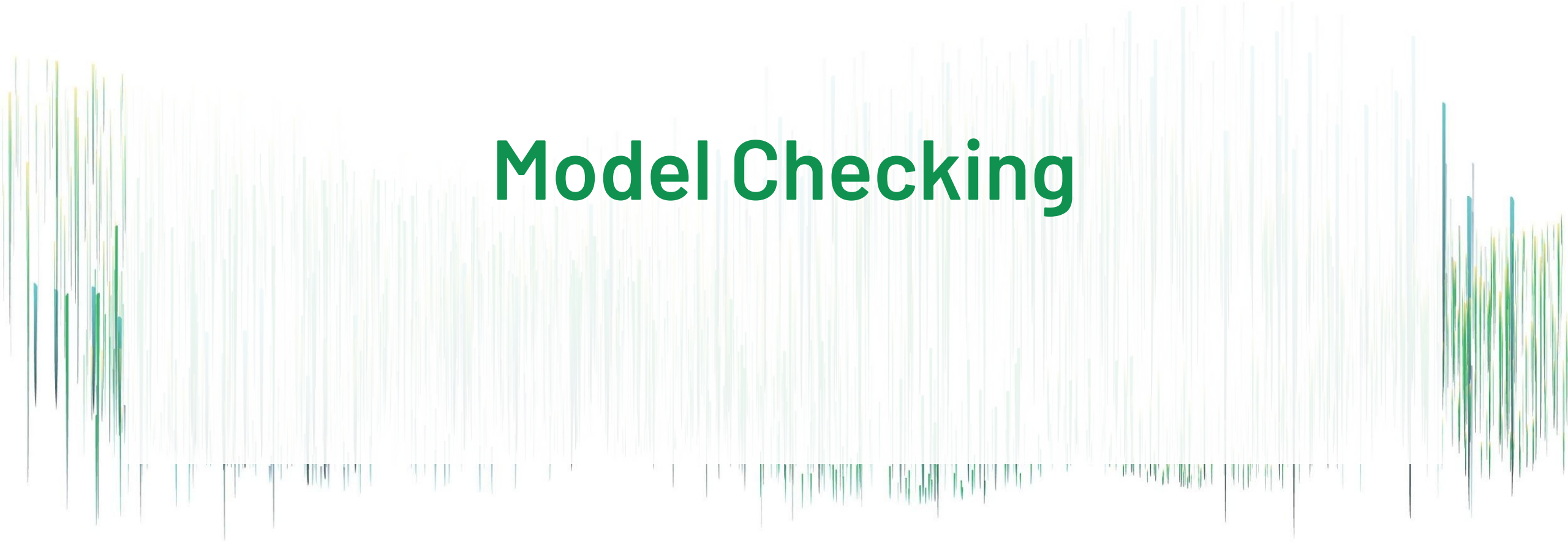
Next State Relation

```
\* Defines how the variables may transition.
Next ==
    \* -- Replication protocol
    \/ AppendOplogAction
    \/ RollbackOplogAction
    \/ BecomePrimaryByMagicAction
    \/ ClientWriteAction
    \/ ChooseNewSyncSourceAction
    \* -- Commit point learning protocol
    \/ AdvanceCommitPoint
    \/ LearnCommitPointAction
```

MongoDB Replication TLA+ Spec

Statistics

- Original spec
  - **295 lines** of TLA+ including comments & model checking helpers
- Extended spec that models sync source selection
  - **378 lines** of TLA+ including comments & model checking helpers



Model Checking

Model Checking

Bug Timeline

1. 2016 - Heartbeat Commit Point Propagation (*Safety*)
2. 2018 - No Available Sync Source (*Liveness*)
3. 2019 - Sync Source Cycles (*Liveness*)
4. 2019 - Commit Point Held Back on Stale Nodes (*Liveness*)
5. 2019 - Initial Solution Fails in 5 Node Replica Set (*Safety*)

Model Checking

Bug Timeline

1. **2016 - Heartbeat Commit Point Propagation (*Safety*)**
2. 2018 - No Available Sync Source (*Liveness*)
3. 2019 - Sync Source Cycles (*Liveness*)
4. 2019 - Commit Point Held Back on Stale Nodes (*Liveness*)
5. 2019 - Initial Solution Fails in 5 Node Replica Set (*Safety*)

Model Checking

Bug 1: Heartbeat Commit Point Propagation, 3 nodes

- The action used for propagating commit points via heartbeat

```
\* Node i learns the commit point from j via heartbeat.
LearnCommitPoint(i, j) ==
    /\ CommitPointLessThan(i, j)
    /\ commitPoint' = [commitPoint EXCEPT ![i] = commitPoint[j]]
    /\ UNCHANGED <<electionVars, logVars, syncSource>>
```

Model Checking

Bug 1: Heartbeat Commit Point Propagation, 3 nodes

- The invariant we want to check:

```
RollbackCommitted(i) ==
   \E j \in Server:
       /\ CanRollbackOplog(i, j)
       /\ IsCommitted(i, Len(log[i]))

\* The invariant to check.
NeverRollbackCommitted == \A i \in Server: ~RollbackCommitted(i)
```

Model Checking

Bug 1: Heartbeat Commit Point Propagation, 3 nodes, *LearnCommitPoint* action

- Model checking stats:
  - 3 nodes, a symmetry set
  - Propagate commit point via heartbeats (`LearnCommitPoint` action)
  - State constraints: `globalCurrentTerm ≤ 3, ∀ i ∈ Server : Len(log[i]) ≤ 3`
  - Invariant: `NeverRollbackBeforeCommitPoint`
  - 2015 Macbook Pro, 3 CPU Cores (3.1 GHz Intel Core i7)
  - **Invariant violation** found in ~2 seconds after generating ~500 distinct states
  - **9177 distinct states** in full state space
  - TLC config

Model Checking

Bug 1: Heartbeat Commit Point Propagation, 3 nodes

- We can try fix the protocol with a new action definition:

```
LearnCommitPointFromSyncSource(i, j) ==
    /\ ENABLED AppendOplog(i, j) \* only learn commit point from sync source.
    /\ LearnCommitPoint(i, j)
```

Model Checking

Bug 1: Heartbeat Commit Point Propagation, 3 nodes, *LearnCommitPointFromSyncSource* action

- Model checking stats:
  - 3 nodes, a symmetry set
  - Propagate commit point via sync source (`LearnCommitPointFromSyncSource` action)
  - State constraints: `globalCurrentTerm ≤ 3, ∀ i ∈ Server : Len(log[i]) ≤ 3`
  - Invariant: `NeverRollbackBeforeCommitPoint`
  - 2015 Macbook Pro, 3 CPU Cores (3.1 GHz Intel Core i7)
  - **No errors found,** TLC finished in ~3 seconds
  - **7402 distinct states** in full state space
    - 1775 fewer states than the previous model
  - TLC config

Model Checking

Bug 1: Heartbeat Commit Point Propagation, 3 nodes

- The protocol appears to be safe when using the sync source propagation rule
- Is it safe with more than 3 nodes, though?

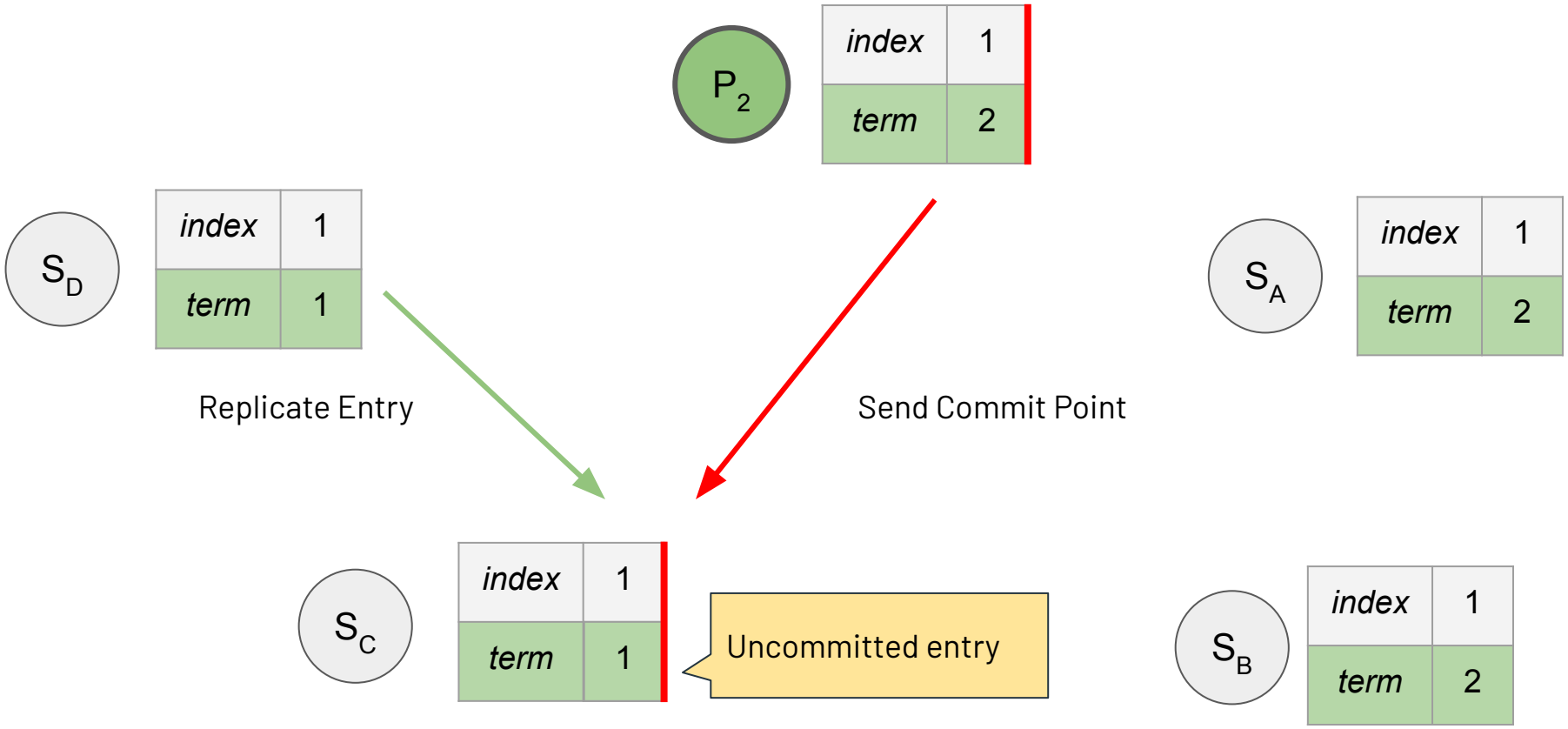
Model Checking

Bug 1: Heartbeat Commit Point Propagation, 5 nodes, *LearnCommitPointFromSyncSource* action

- Model checking stats:
  - 5 nodes, a symmetry set
  - Propagate commit point via sync source (`LearnCommitPointFromSyncSource` action)
  - State constraints: `globalCurrentTerm ≤ 3, ∀ i ∈ Server : Len(log[i]) ≤ 3`
  - Invariant: `NeverRollbackBeforeCommitPoint`
  - Ubuntu 16.10 workstation, 10 CPU cores (3.40GHz Intel Core i7)
  - **Invariant violation** found in ~2 seconds after generating ~3000 distinct states
  - **230,091 distinct states** in full state space, TLC finished in ~1 min.
  - This bug was never found in production or testing
  - TLC config

Model Checking

Bug 1: Heartbeat Commit Point Propagation, 5 nodes



Model Checking

Bug Timeline

1. 2016 - Heartbeat Commit Point Propagation (*Safety*)
2. **2018 - No Available Sync Source (*Liveness*)**
3. 2019 - Sync Source Cycles (*Liveness*)
4. 2019 - Commit Point Held Back on Stale Nodes (*Liveness*)
5. 2019 - Initial Solution Fails in 5 Node Replica Set (*Safety*)

Model Checking

Bug 2: No Available Sync Source

- We define a liveness property of commit points that we want to hold true

```
\* At any time, if two nodes' commit points are not the same, they
\* will be the same eventually.
CommitPointEventuallyPropagates ==
   /\ \A i, j \in Server:
       (commitPoint[i] /= commitPoint[j] ~> commitPoint[i] = commitPoint[j])
```

Model Checking

Bug 2: No Available Sync Source

- Slight modification needed to account for perpetual rollbacks

```
\* At any time, if two nodes' commit points are not the same, they
\* will be the same eventually.
\* This is checked after all possible rollback is done.
CommitPointEventuallyPropagates ==
   /\ \A i, j \in Server:
       (commitPoint[i] /= commitPoint[j] ~>
              (~ENABLED RollbackOplogAction => commitPoint[i] = commitPoint[j]))
```

Model Checking

Bug 2: No Available Sync Source

- Demonstrated the original liveness bug with TLC
  - 3 nodes
  - Property: `CommitPointEventuallyPropagates`
  - 2015 Macbook Pro, 3 CPU Cores (3.1 GHz Intel Core i7)
  - **Temporal Property Violation** found in 1 min. 06s
  - **19,694 distinct states** generated
  - TLC config

Model Checking

Bug Timeline

1. 2016 - Heartbeat Commit Point Propagation (*Safety*)
2. 2018 - No Available Sync Source (*Liveness*)
3. **2019 - Sync Source Cycles (*Liveness*)**
4. 2019 - Commit Point Held Back on Stale Nodes (*Liveness*)
5. 2019 - Initial Solution Fails in 5 Node Replica Set (*Safety*)

Model Checking

Bug 3: Sync Source Cycles

- We add an action to model sync source selection
- And then specify our correctness property
  - We will specify this particular liveness property as an invariant

Model Checking

Bug 3: Sync Source Cycles

```
\* Server i chooses server j as its new sync source.
\* i can choose j as a sync source if log[i] is a prefix of log[j] and log[j] is longer than log[i].
ChooseNewSyncSource(i, j) ==
    /\ \/ IsLogPrefix(i, j)
       \* If logs are equal, allow choosing sync source if it has a newer commit point.
       \/ /\ log[i] = log[j]
          /\ commitPoint[j].index > commitPoint[i].index
    /\ state[i] = Follower \* leaders don't need to sync oplog entries.
    /\ syncSource' = [syncSource EXCEPT ![i] = j]
    /\ UNCHANGED <<electionVars, logVars, commitPoint>>
```

Model Checking

Bug 3: Sync Source Cycles

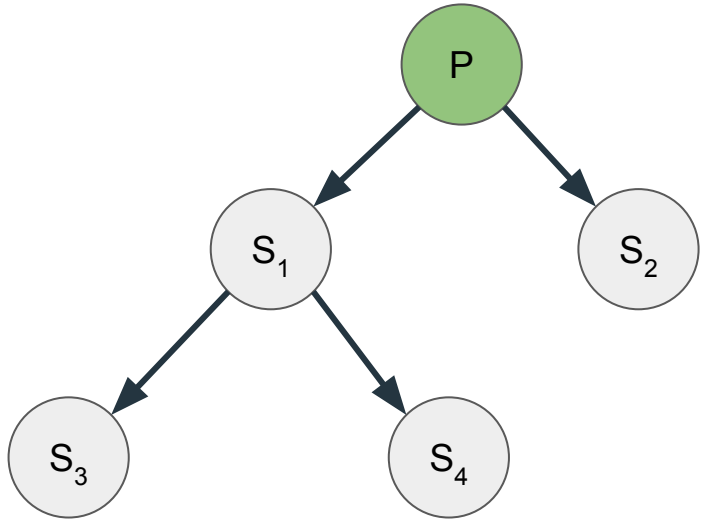
- Specifying the case of a 2 node cycle is much easier:

```
\* Does a 2 node sync source cycle exist?
SyncSourceCycleTwoNode ==
   \E s, t \in Server :
       /\ s /= t
       /\ syncSource[s] = t
       /\ syncSource[t] = s
```

Model Checking

Bug 3: Sync Source Cycles

- We can also specify the general case i.e. a multi-node cycle
- Core idea: model the sync source spanning tree/graph in TLA+



Nodes

Edges

Model Checking

Bug 3: Sync Source Cycles

```
\* The set of all paths (with bounded length) in the node graph that consists
\* of edges <<s,t>> where s has t as a sync source.
SyncSourcePaths ==
   {p \in BoundedSeq(Server, Cardinality(Server)) :
       \A i \in 1..(Len(p)-1) : syncSource[p[i]] = p[i+1]}
```

Model Checking

Bug 3: Sync Source Cycles

```
\* Is there a non-trivial path in the sync source graph from node i to node j?
\* This rules out trivial paths i.e. those of length 1.
SyncSourcePath(i, j) ==
   \E p \in SyncSourcePaths :
       /\ Len(p) > 1
       /\ p[1] = i \* the source node.
       /\ p[Len(p)] = j \* the target node.

\* Does a general (multi-node) sync source cycle exist?
SyncSourceCycle == \E s \in Server : SyncSourcePath(s, s)
```

Model Checking

Bug 3: Sync Source Cycles

- Finally, we can ask specifically for cycles of size > 2

```
\* The sync source cycle predicate.
NonTrivialSyncCycle == SyncSourceCycle /\ ~SyncSourceCycleTwoNode

\* The invariant.
NoNonTrivialSyncCycle == ~NonTrivialSyncCycle
```

Model Checking

Bug 3: Sync Source Cycles

- Model checking stats:
  - 4 nodes, a symmetry set
  - State constraints: `globalCurrentTerm ≤ 3, ∀ i ∈ Server : Len(log[i]) ≤ 3`
  - Invariant: `NoNonTrivialSyncCycle`
  - 2015 Macbook Pro, 3 CPU Cores (3.1 GHz Intel Core i7)
  - **Invariant violation** found in ~6 seconds
  - **226,262 distinct states** in full state space
  - Multi-node sync source cycle was never seen in production or testing
  - TLC config

Takeaways

Takeaways

- Hard to know if a protocol is really correct without a formal model
  - It's very difficult for humans to reason about edge cases of concurrent/distributed algorithms
- Even very *simple* and *abstract* models can help catch non-trivial bugs
  - No models allowed more than 3 log entries per node
  - Asynchronous message passing was not modeled explicitly in our spec

Takeaways

- We expect that formally modeling our system upfront could have saved 100s of hours of engineering time
  - Multi-year effort to root out all these bugs
  - Only took a few weeks to model and check the protocol using TLA+
- Future goal is to integrate TLA+ into design process at MongoDB

Takeaways

- The specs and models used can be found here: https://github.com/will62794/mongo-repl-tla-models


mongoDB®