Verifying the Liveness of Eventually-Synchronous BFT Consensus Protocols

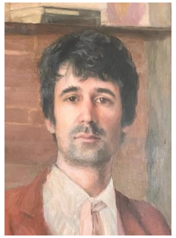
A pragmatic approach

Giuliano Losa
Stellar Development Foundation

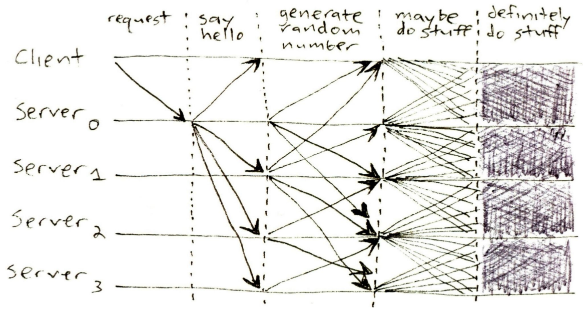
Motivation: checking that the TetraBFT consensus protocol satisfies its liveness property

TetraBFT (PODC 2024) is a new BFT consensus protocol with a subtle liveness argument, and we would like to mechanically verify it

Qianyu Yu
HKUST


Giuliano Losa
SDF

Xuechao Wang
HKUST



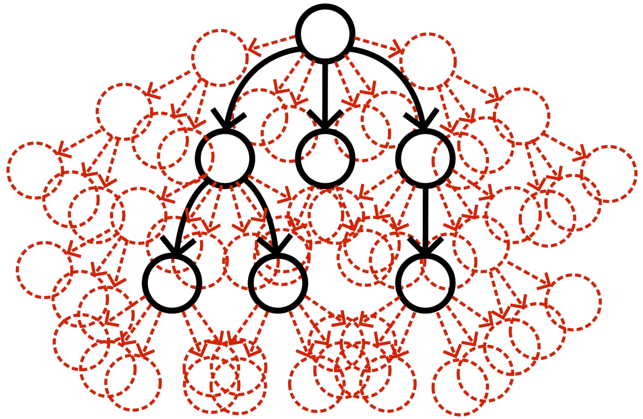
The Saddest Moment, James Mickens

Due to Byzantine behavior, state-space exploration with TLC is intractable

TLC explicitly enumerates bounded behaviors

This is a fully-automated, push-button solution

But it is too slow, because Byzantine behavior creates too many successors to each state



Given a user-provided inductive invariant, Apalache can efficiently that a state predicate always holds

An inductive invariant holds holds initially and that every step preserves it:

$$\begin{aligned} &\land \quad Init \Rightarrow Inv \\ &\land \quad Inv \land Next \Rightarrow Inv' \end{aligned}$$

For bounded domains, this can be encoded into SMT formulas, which Apalache passes to the Z3 solver for efficient checking.

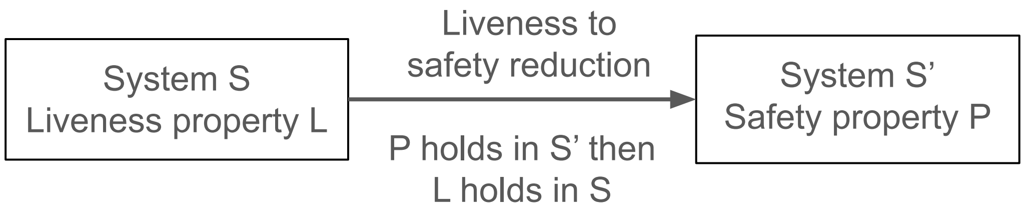
We obtain a semi-automated proof method for verifying that a state predicate P always holds:

1. The user provides a candidate inductive invariant I
2. Apalache checks that I is indeed inductive and that I ⇒ P
    a. If not, go back to 1

To use inductive invariants for liveness, we propose a simple liveness-to-safety reduction

Safety VS Liveness

- Safety: all reachable states satisfy P
- Liveness: fairness conditions + temporal property of behaviors
  - Example: if every message is eventually delivered, nodes eventually decide



Liveness to safety in practice

- Some tools produce a new safety problem automatically
  - Works best in conjunction with automated checking methods
  - Manually finding an inductive invariants for an automatically-generated safety problem is not fun! (and difficult)
- We give an "easy" reduction for the special case of eventually-synchronous BFT protocols

In this talk, we explain the liveness-proof methodology using the Paxos consensus algorithm

System model

- We have a fixed set of nodes (or "acceptors") in a message-passing network
- The network is initially asynchronous, and becomes synchronous after time GST
- Local clocks that advance at the same rate
- Less than ½ of the nodes may crash-stop

The Consensus Problem

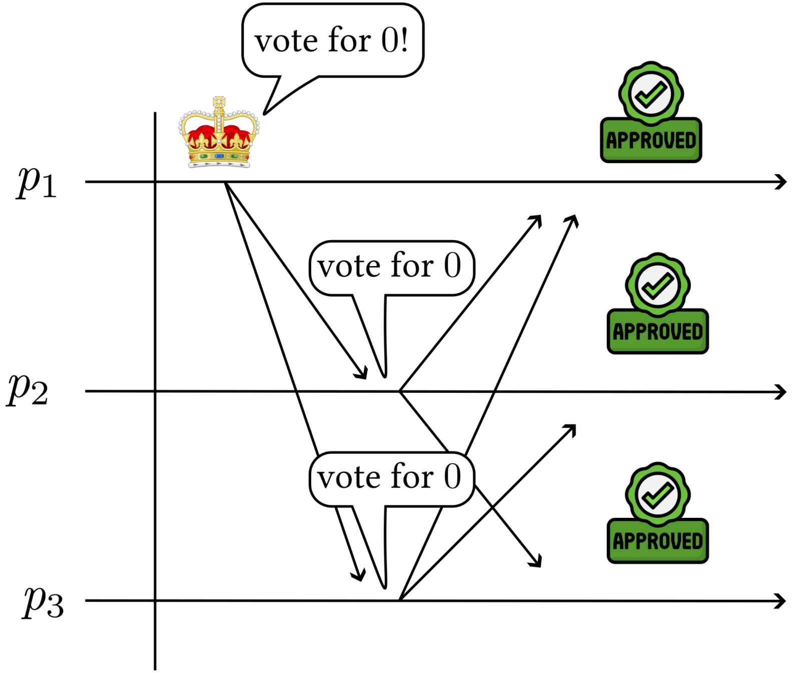
Nodes each start with a private value and must decide on a common value.

- Safety: no two well-behaved nodes decide differently (and, if all nodes are well-behaved, then they decide one of the inputs)
- Liveness: eventually, all well-behaved nodes decide

A leader instruct nodes to vote for a value

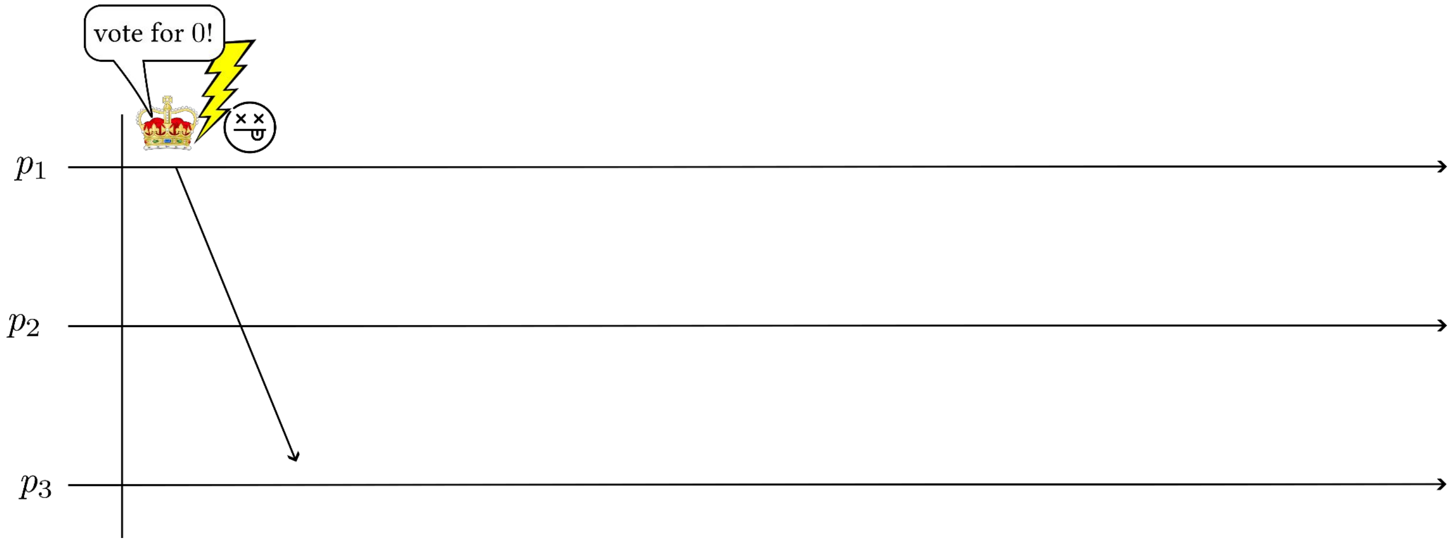
Nodes vote for the leader’s value

A node decides a value when a majority of nodes vote for it



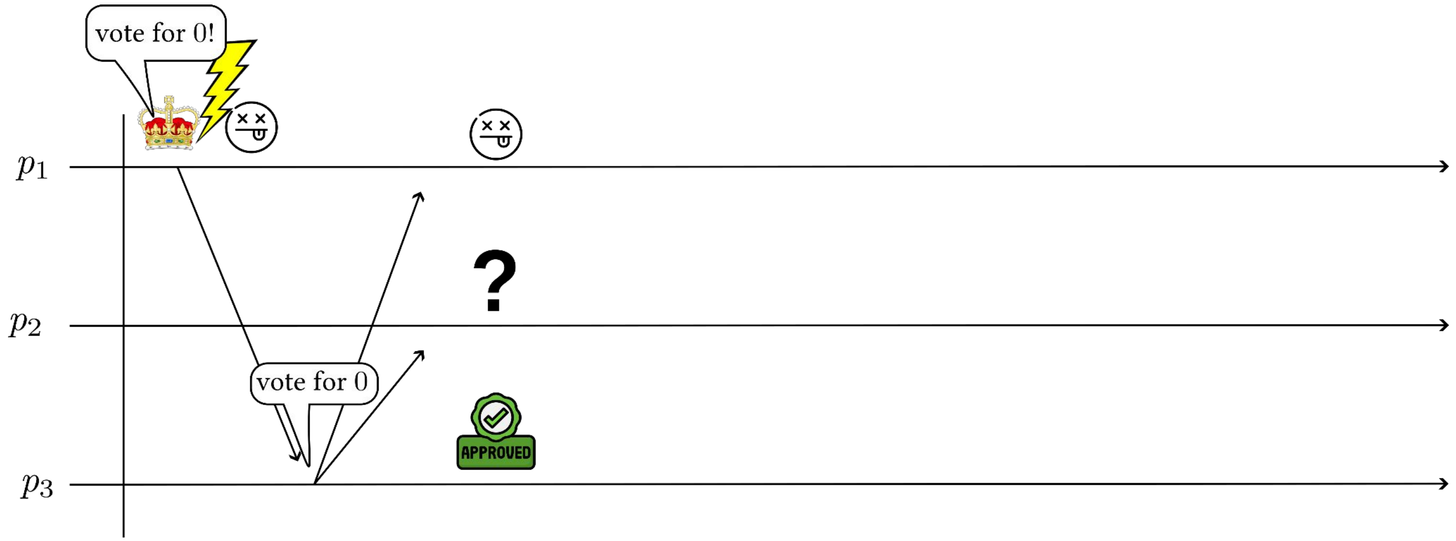
If the first leader fails to impose its decision, nodes time-out and try again with a new leader

Each such attempt is called a ballot



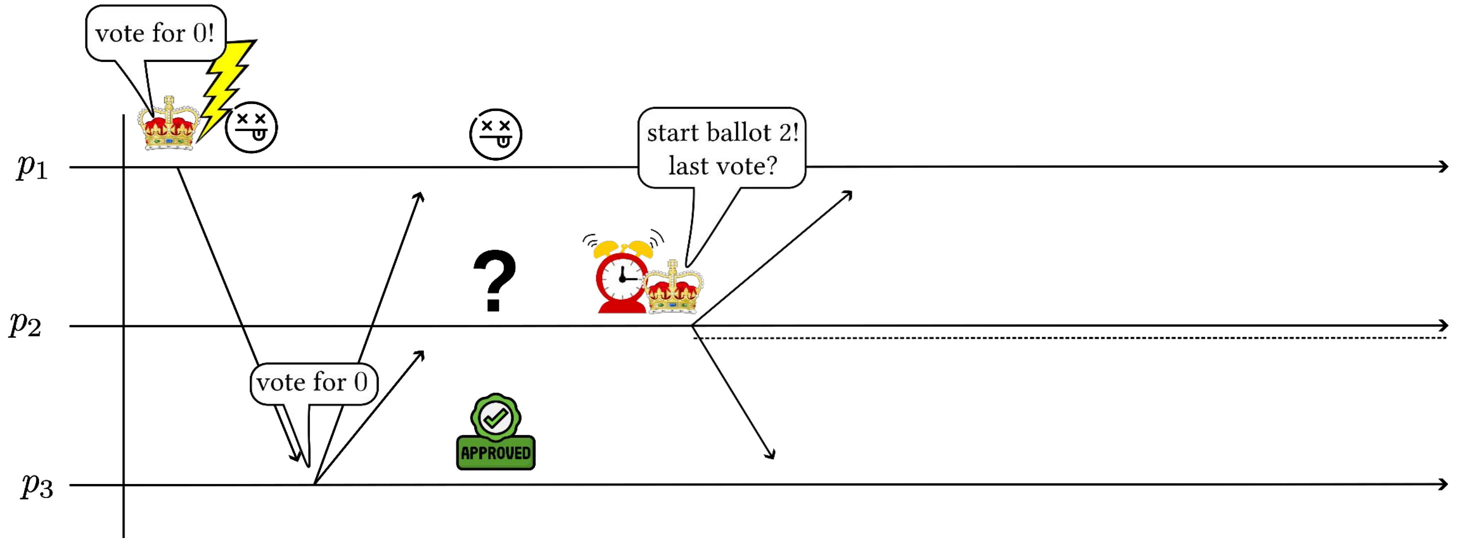
If the first leader fails to impose its decision, nodes time-out and try again with a new leader

Each such attempt is called a ballot



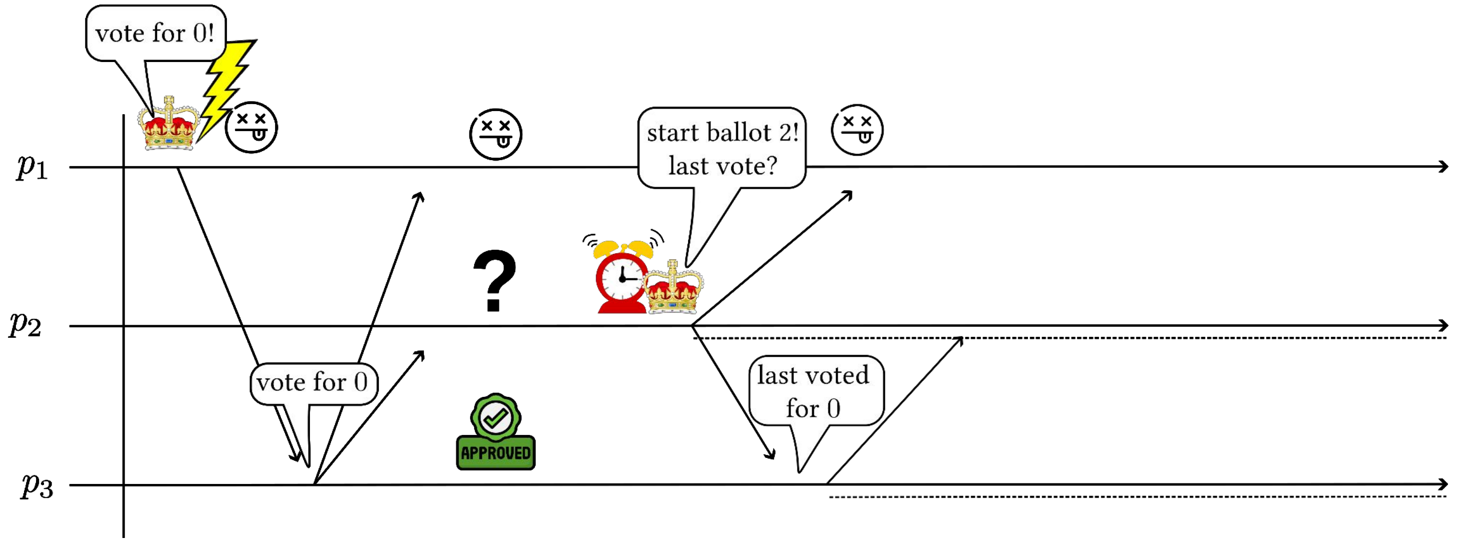
If the first leader fails to impose its decision, nodes time-out and try again with a new leader

Each such attempt is called a ballot



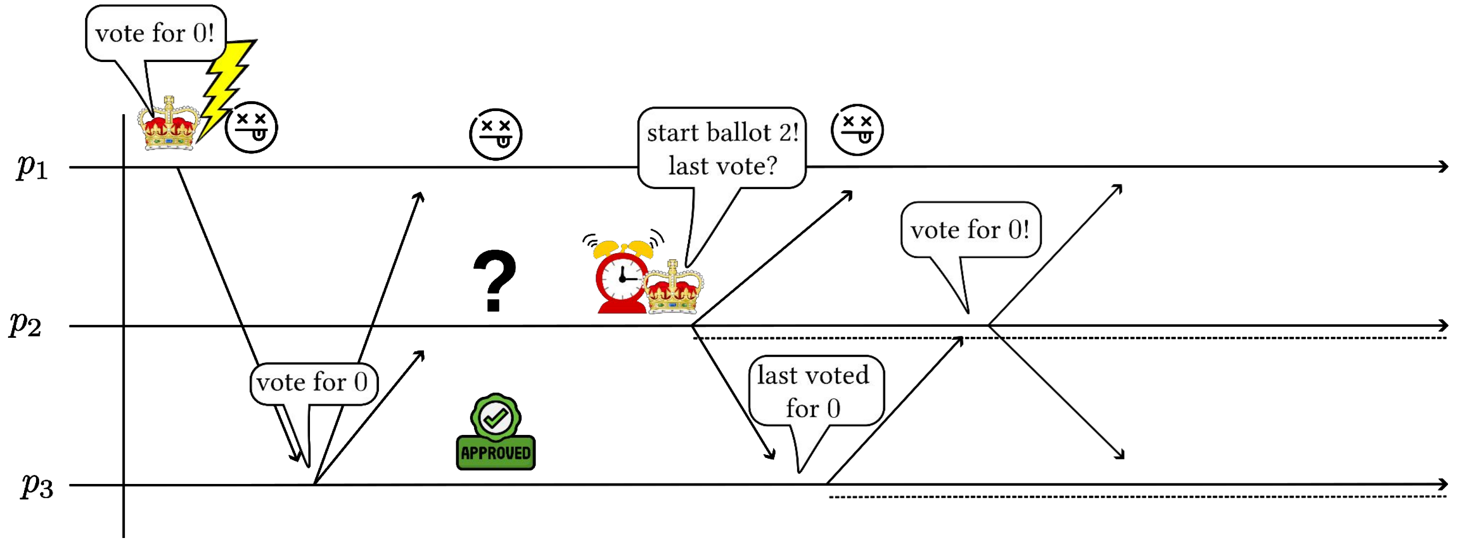
If the first leader fails to impose its decision, nodes time-out and try again with a new leader

Each such attempt is called a ballot



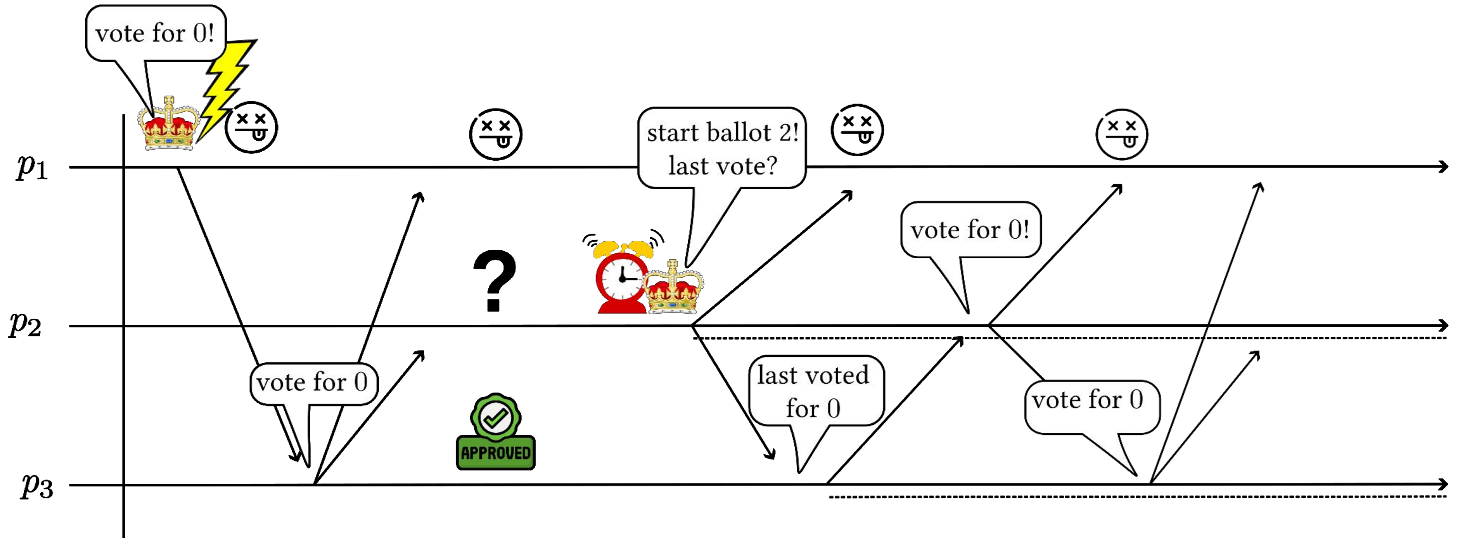
If the first leader fails to impose its decision, nodes time-out and try again with a new leader

Each such attempt is called a ballot



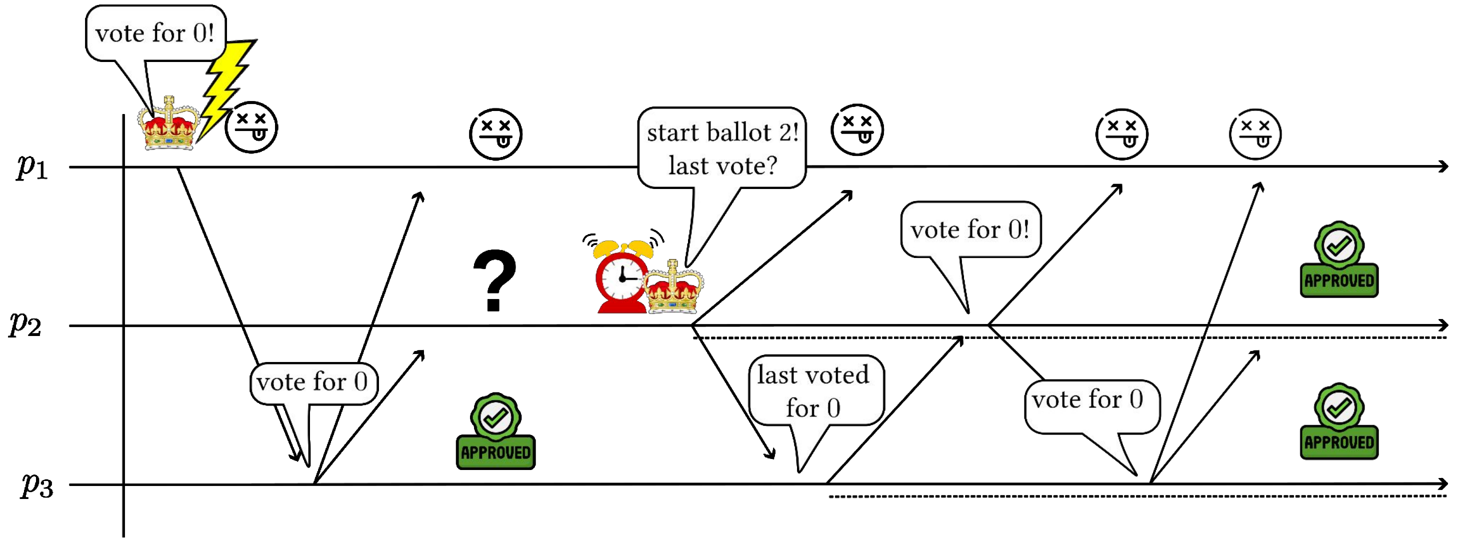
If the first leader fails to impose its decision, nodes time-out and try again with a new leader

Each such attempt is called a ballot

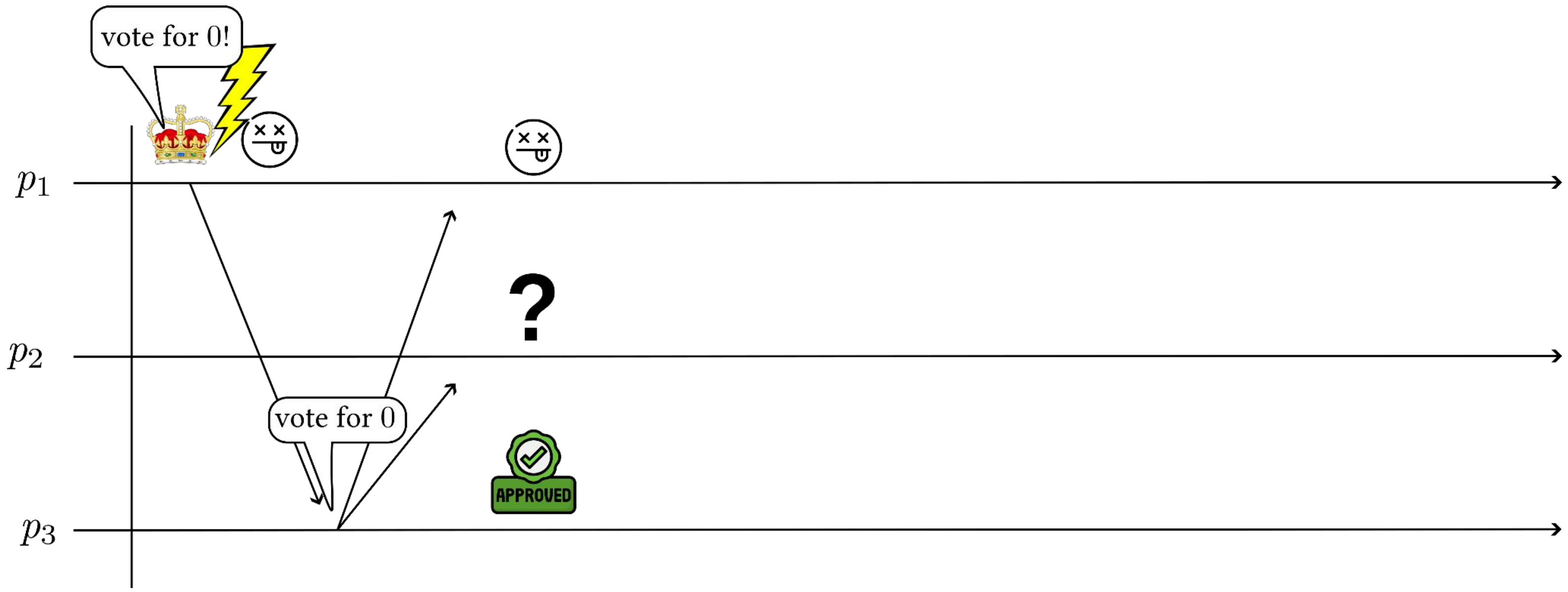


If the first leader fails to impose its decision, nodes time-out and try again with a new leader

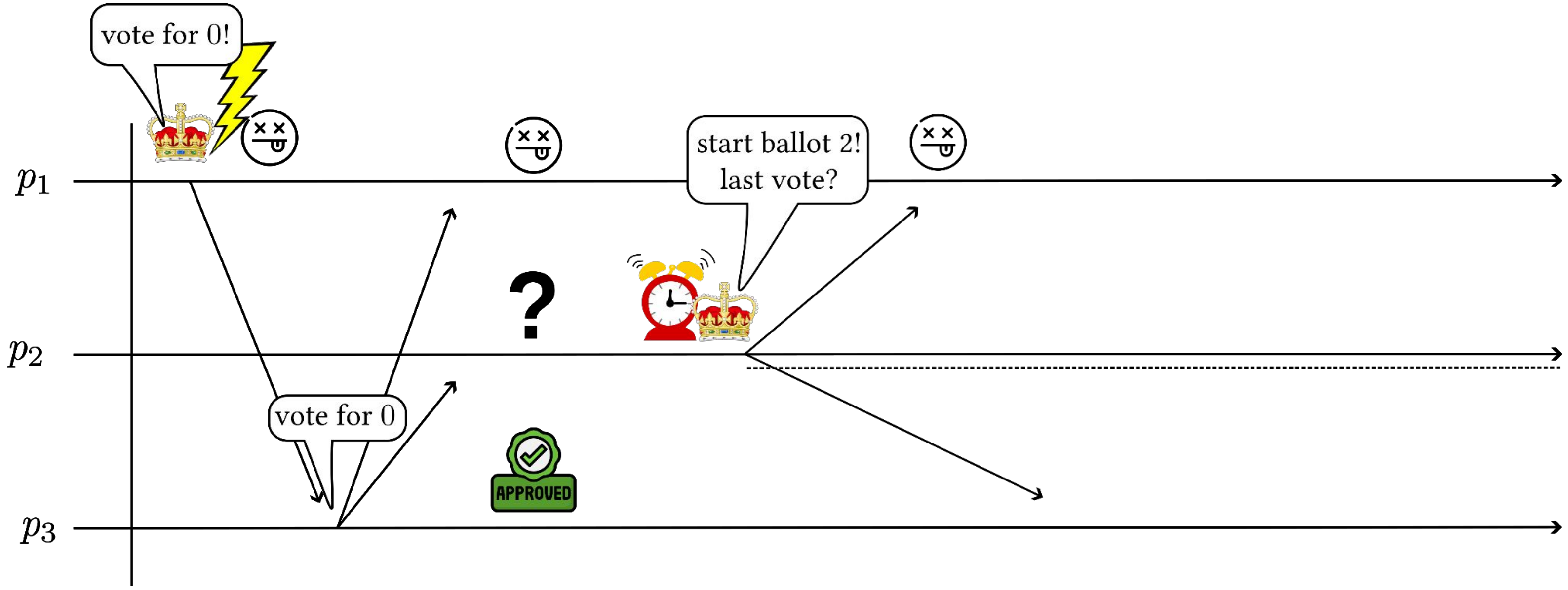
Each such attempt is called a ballot



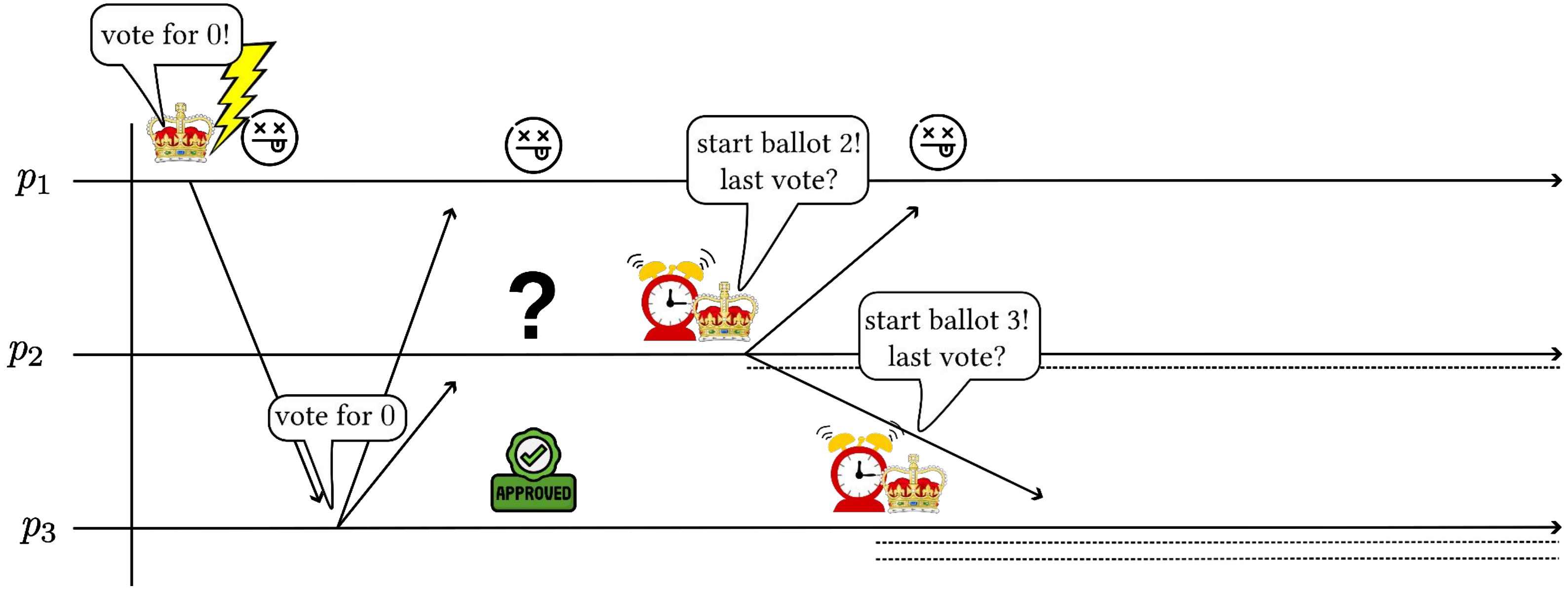
Depending on message delay, node might “step on each other’s toes” by starting new ballots too early



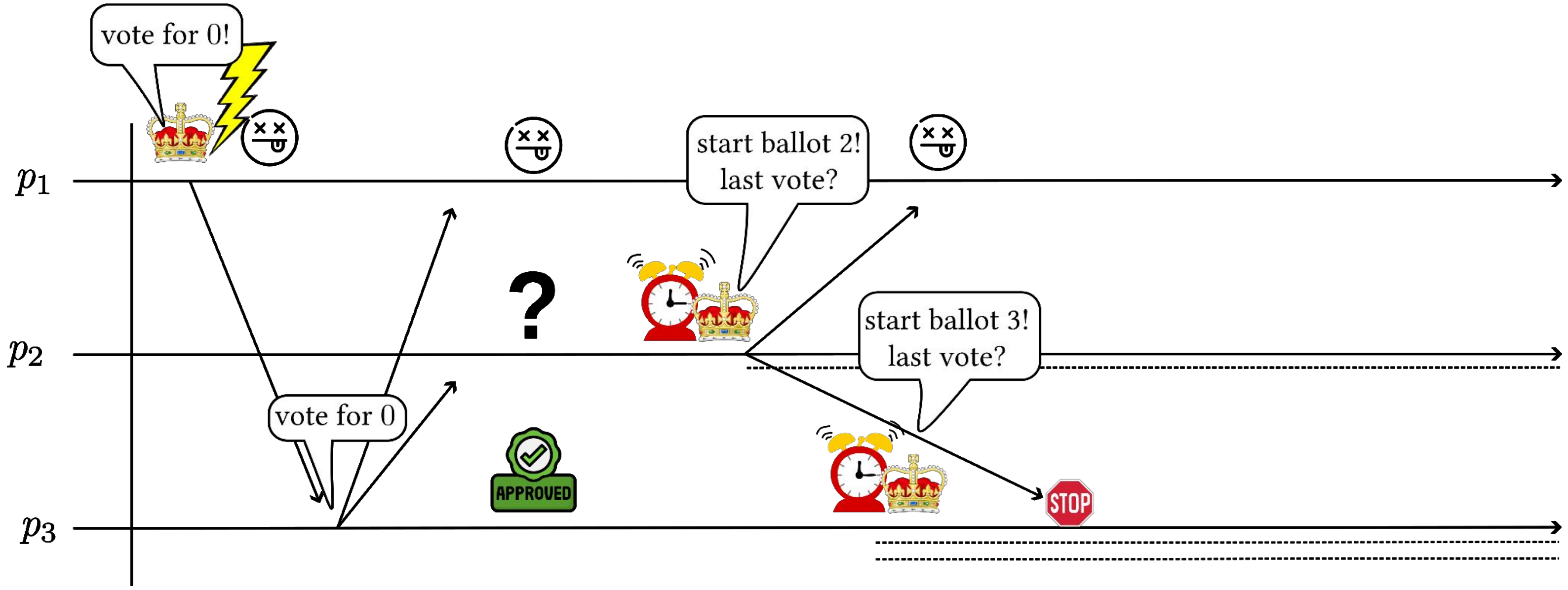
Depending on message delay, node might “step on each other’s toes” by starting new ballots too early



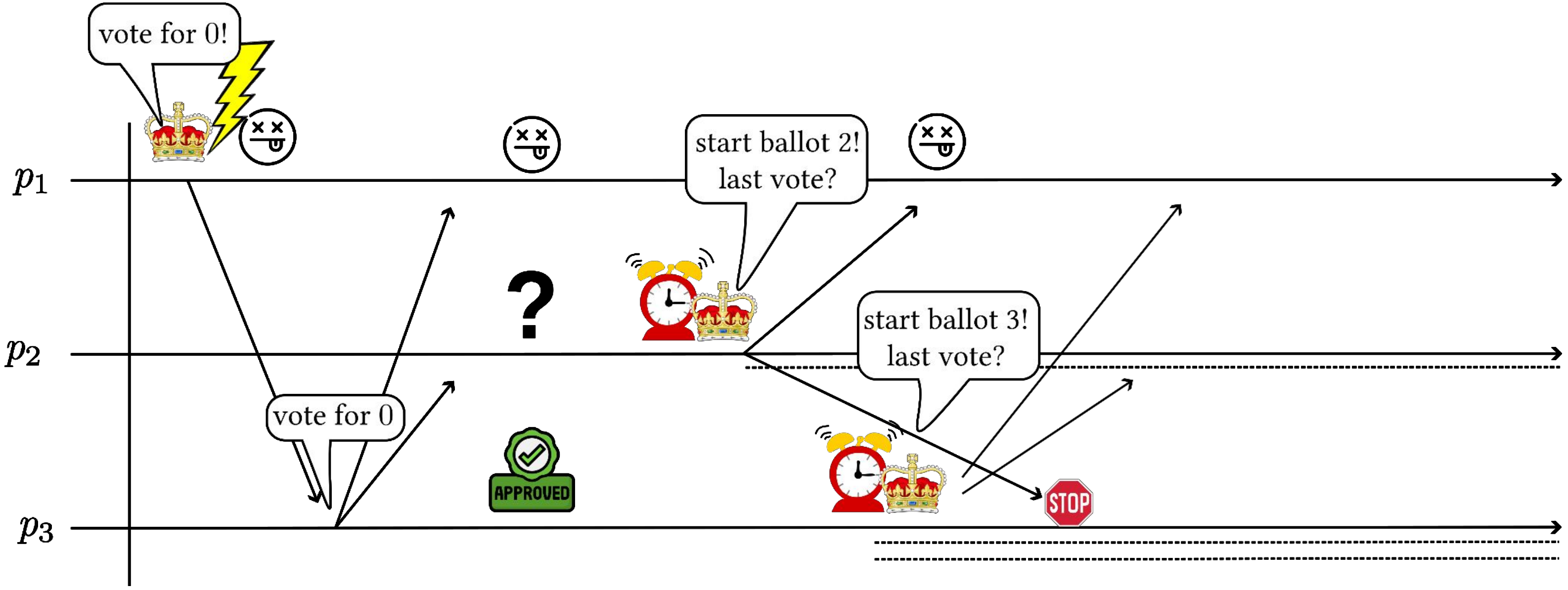
Depending on message delay, node might “step on each other’s toes” by starting new ballots too early



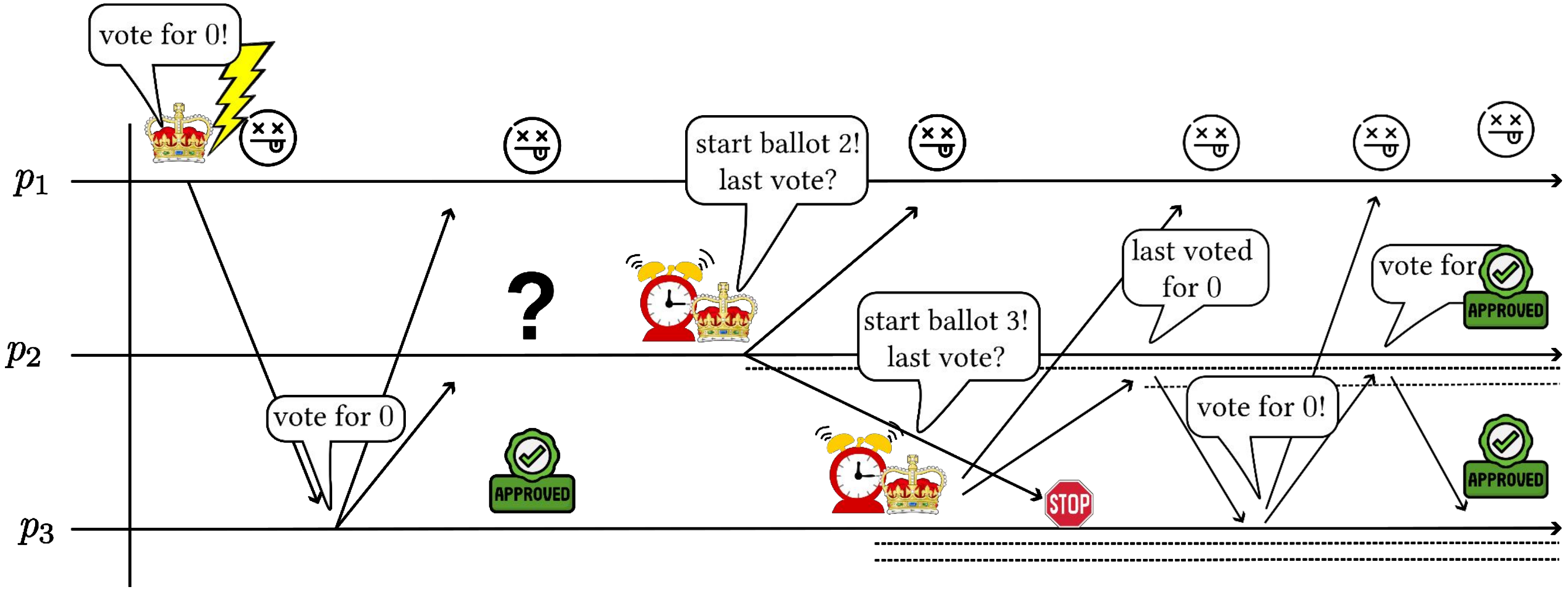
Depending on message delay, node might “step on each other’s toes” by starting new ballots too early



Depending on message delay, node might “step on each other’s toes” by starting new ballots too early



Depending on message delay, node might "step on each other's toes" by starting new ballots too early



To make things easier, we abstract timers away

Instead of modeling message delay and timers, we prove the following liveness property:

If there is a *good ballot* B such that:

- No node starts a higher ballot
- The leader of B is well-behaved

Then, eventually, all well-behaved nodes decide

In TetraBFT, the general idea is the same but the protocol is more complicated

- There are 6 types of messages per ballot
- The leader may be malicious!
  - Nodes must verify that the leader’s proposal is legitimate
  - Main difficulty: despite differing views, the leader must make sure that other nodes will be able to check that it’s proposal is legitimate

The Paxos algorithm in TLA+ (excerpts)

$\textsc{variables}\ votes,\ currBal,\ proposals,\ crashed$

$$
\begin{aligned}
&TypeOK \triangleq \\
&\quad \land\ votes \in [Acceptor \rightarrow \textsc{subset}\ (Ballot \times Value)] \\
&\quad \land\ currBal \in [Acceptor \rightarrow Ballot \cup \{-1\}] \\
&\quad \land\ proposals \in \textsc{subset}\ (Ballot \times Value) \\
&\quad \land\ crashed \in \textsc{subset}\ Acceptor
\end{aligned}
$$

The Paxos algorithm in TLA+ (excerpts)

$\textsc{variables}\ votes,\ currBal,\ proposals,\ crashed$

$$
\begin{array}{l}
TypeOK \triangleq \\
\quad \land\ \ votes \in [Acceptor \rightarrow \textsc{subset}\ (Ballot \times Value)] \\
\quad \land\ \ currBal \in [Acceptor \rightarrow Ballot \cup \{-1\}] \\
\quad \land\ \ proposals \in \textsc{subset}\ (Ballot \times Value) \\
\quad \land\ \ crashed \in \textsc{subset}\ Acceptor
\end{array}
$$

$$
\begin{array}{l}
chosen \triangleq \{v \in Value : \exists\, b \in Ballot : \exists\, Q \in Quorum : \\
\quad \forall\, a \in Q : \langle b,\, v\rangle \in votes\}
\end{array}
$$

The Paxos algorithm in TLA+ (excerpts)

$\textsc{variables}\ votes,\ currBal,\ proposals,\ crashed$

$$
\begin{array}{l}
TypeOK\ \triangleq \\
\quad \land\quad votes \in [Acceptor \to \textsc{subset}\ (Ballot \times Value)] \\
\quad \land\quad currBal \in [Acceptor \to Ballot \cup \{-1\}] \\
\quad \land\quad proposals \in \textsc{subset}\ (Ballot \times Value) \\
\quad \land\quad crashed \in \textsc{subset}\ Acceptor
\end{array}
$$

$$
\begin{array}{l}
chosen\ \triangleq\ \{v \in Value : \exists\, b \in Ballot : \exists\, Q \in Quorum : \\
\quad \forall\, a \in Q : \langle b,\ v\rangle \in votes\}
\end{array}
$$

$$
\begin{array}{l}
Init\ \triangleq \\
\quad \land\ votes = [a \in Acceptor \mapsto \{\}] \\
\quad \land\ currBal = [a \in Acceptor \mapsto\ -1] \\
\quad \land\ proposals = \{\} \\
\quad \land\ crashed = \{\}
\end{array}
$$

The Paxos algorithm in TLA+ (excerpts)

$\textsc{variables}\ votes,\ currBal,\ proposals,\ crashed$

$$
\begin{array}{l}
TypeOK \triangleq \\
\quad \land \quad votes \in [Acceptor \rightarrow \textsc{subset}\ (Ballot \times Value)] \\
\quad \land \quad currBal \in [Acceptor \rightarrow Ballot \cup \{-1\}] \\
\quad \land \quad proposals \in \textsc{subset}\ (Ballot \times Value) \\
\quad \land \quad crashed \in \textsc{subset}\ Acceptor
\end{array}
$$

$$
\begin{array}{l}
chosen \triangleq \{v \in Value : \exists\, b \in Ballot : \exists\, Q \in Quorum : \\
\quad \forall\, a \in Q : \langle b, v \rangle \in votes\}
\end{array}
$$

$$
\begin{array}{l}
Init \triangleq \\
\quad \land\ votes = [a \in Acceptor \mapsto \{\}] \\
\quad \land\ currBal = [a \in Acceptor \mapsto -1] \\
\quad \land\ proposals = \{\} \\
\quad \land\ crashed = \{\}
\end{array}
$$

$$
\begin{array}{l}
Crash(a) \triangleq \\
\quad \land\ \ crashed' = crashed \cup \{a\} \\
\quad \land\ \ \exists\, Q \in Quorum : \forall\, a2 \in Q : a2 \notin crashed' \\
\quad \land\ \ \textsc{unchanged}\ \langle votes, currBal, proposals \rangle
\end{array}
$$

The Paxos algorithm in TLA+ (excerpts)

$\text{VARIABLES } votes, currBal, proposals, crashed$

$$\begin{aligned} TypeOK \triangleq\ & \land\ votes \in [Acceptor \rightarrow \text{SUBSET } (Ballot \times Value)] \\ & \land\ currBal \in [Acceptor \rightarrow Ballot \cup \{-1\}] \\ & \land\ proposals \in \text{SUBSET } (Ballot \times Value) \\ & \land\ crashed \in \text{SUBSET } Acceptor \end{aligned}$$

$$\begin{aligned} chosen \triangleq\ & \{v \in Value : \exists\, b \in Ballot : \exists\, Q \in Quorum : \\ & \quad \forall\, a \in Q : \langle b, v \rangle \in votes\} \end{aligned}$$

$$\begin{aligned} Init \triangleq\ & \land\ votes = [a \in Acceptor \mapsto \{\}] \\ & \land\ currBal = [a \in Acceptor \mapsto -1] \\ & \land\ proposals = \{\} \\ & \land\ crashed = \{\} \end{aligned}$$

$$\begin{aligned} Crash(a) \triangleq\ & \land\ crashed' = crashed \cup \{a\} \\ & \land\ \exists\, Q \in Quorum : \forall\, a2 \in Q : a2 \notin crashed' \\ & \land\ \text{UNCHANGED } \langle votes, currBal, proposals \rangle \end{aligned}$$

$$\begin{aligned} Propose(b, v) \triangleq\ & \land\ Leader(b) \notin crashed \\ & \land\ \forall\, prop \in proposals : prop[1] \neq b \\ & \land\ \exists\, Q \in Quorum : ShowsSafeAt(Q, b, v) \\ & \land\ proposals' = proposals \cup \{\langle b, v \rangle\} \\ & \land\ \text{UNCHANGED } \langle votes, currBal, crashed \rangle \end{aligned}$$

$$\begin{aligned} IncreaseCurrBal(a, b) \triangleq\ & \land\ a \notin crashed \\ & \land\ b > currBal[a] \\ & \land\ currBal' = [currBal \text{ EXCEPT } ![a] = b] \\ & \land\ \text{UNCHANGED } \langle votes, proposals, crashed \rangle \end{aligned}$$

$$\begin{aligned} VoteFor(a, b, v) \triangleq\ & \land\ a \notin crashed \\ & \land\ currBal[a] \leq b \\ & \land\ \forall\, vt \in votes[a] : vt[1] \neq b \\ & \land\ \langle b, v \rangle \in proposals \\ & \land\ votes' = [votes \text{ EXCEPT } ![a] = @ \cup \{\langle b, v \rangle\}] \\ & \land\ currBal' = [currBal \text{ EXCEPT } ![a] = b] \\ & \land\ \text{UNCHANGED } \langle crashed, proposals \rangle \end{aligned}$$

Methodology, step 1: prophesize the “good ballot”

$\textsc{variables}\ \mathit{votes},\ \mathit{currBal},\ \mathit{proposals},\ \mathit{crashed},\ \mathit{goodBallot}$

$$\begin{aligned}
&\mathit{TypeOK}\ \triangleq\\
&\quad \land\ \ \mathit{votes} \in [\mathit{Acceptor} \to \textsc{subset}\ (\mathit{Ballot} \times \mathit{Value})]\\
&\quad \land\ \ \mathit{currBal} \in [\mathit{Acceptor} \to \mathit{Ballot} \cup \{-1\}]\\
&\quad \land\ \ \mathit{proposals} \in \textsc{subset}\ (\mathit{Ballot} \times \mathit{Value})\\
&\quad \land\ \ \mathit{crashed} \in \textsc{subset}\ \mathit{Acceptor}\\
&\quad \land\ \ \mathit{goodBallot} \in \mathit{Ballot}
\end{aligned}$$

$$\begin{aligned}
&\mathit{Init}\ \triangleq\\
&\quad \land\ \mathit{votes} = [a \in \mathit{Acceptor} \mapsto \{\}]\\
&\quad \land\ \mathit{currBal} = [a \in \mathit{Acceptor} \mapsto -1]\\
&\quad \land\ \mathit{proposals} = \{\}\\
&\quad \land\ \mathit{crashed} = \{\}\\
&\quad \land\ \mathit{goodBallot} \in \mathit{Ballot}
\end{aligned}$$

$$\begin{aligned}
&\mathit{Crash}(a)\ \triangleq\\
&\quad \land\ \ a \neq \mathit{Leader}(\mathit{goodBallot})\\
&\quad \land\ \ \mathit{crashed}' = \mathit{crashed} \cup \{a\}\\
&\quad \land\ \ \exists\, Q \in \mathit{Quorum} : \forall\, a2 \in Q : a2 \notin \mathit{crashed}'\\
&\quad \land\ \ \textsc{unchanged}\ \langle \mathit{votes},\ \mathit{currBal},\ \mathit{proposals},\ \mathit{goodBallot} \rangle
\end{aligned}$$

$$\begin{aligned}
&\mathit{Propose}(b,\ v)\ \triangleq\\
&\quad \land\ \ \mathit{Leader}(b) \notin \mathit{crashed}\\
&\quad \land\ \ \forall\, \mathit{prop} \in \mathit{proposals} : \mathit{prop}[1] \neq b\\
&\quad \land\ \ \exists\, Q \in \mathit{Quorum} : \mathit{ShowsSafeAt}(Q,\ b,\ v)\\
&\quad \land\ \ \mathit{proposals}' = \mathit{proposals} \cup \{\langle b,\ v \rangle\}\\
&\quad \land\ \ \textsc{unchanged}\ \langle \mathit{votes},\ \mathit{currBal},\ \mathit{crashed},\ \mathit{goodBallot} \rangle
\end{aligned}$$

$$\begin{aligned}
&\mathit{IncreaseCurrBal}(a,\ b)\ \triangleq\\
&\quad \land\ a \notin \mathit{crashed}\\
&\quad \land\ b \leq \mathit{goodBallot}\\
&\quad \land\ b > \mathit{currBal}[a]\\
&\quad \land\ \mathit{currBal}' = [\mathit{currBal}\ \textsc{except}\ ![a] = b]\\
&\quad \land\ \textsc{unchanged}\ \langle \mathit{votes},\ \mathit{proposals},\ \mathit{crashed},\ \mathit{goodBallot} \rangle
\end{aligned}$$

$$\begin{aligned}
&\mathit{VoteFor}(a,\ b,\ v)\ \triangleq\\
&\quad \land\ \ a \notin \mathit{crashed}\\
&\quad \land\ \ \mathit{currBal}[a] \leq b\\
&\quad \land\ \ \forall\, \mathit{vt} \in \mathit{votes}[a] : \mathit{vt}[1] \neq b\\
&\quad \land\ \ \langle b,\ v \rangle \in \mathit{proposals}\\
&\quad \land\ \ \mathit{votes}' = [\mathit{votes}\ \textsc{except}\ ![a] = @ \cup \{\langle b,\ v \rangle\}]\\
&\quad \land\ \ \mathit{currBal}' = [\mathit{currBal}\ \textsc{except}\ ![a] = b]\\
&\quad \land\ \ \textsc{unchanged}\ \langle \mathit{crashed},\ \mathit{proposals},\ \mathit{goodBallot} \rangle
\end{aligned}$$

Without Byzantine behavior, we can check the following temporal property with TLC… but, with TetraBFT, we run into scaling issues

$$
\begin{aligned}
&\mathit{FairSpec} \triangleq \\
&\qquad \land\ \mathit{Init} \\
&\qquad \land\ \Box[\mathit{Next}]_{\mathit{vars}} \\
&\qquad \land\ \forall\, a \in \mathit{Acceptor},\ b \in \mathit{Ballot},\ v \in \mathit{Value} : \\
&\qquad\qquad \land\ \mathrm{WF}_{\mathit{vars}}(\mathit{IncreaseCurrBal}(a, b)) \\
&\qquad\qquad \land\ \mathrm{WF}_{\mathit{vars}}(\mathit{VoteFor}(a, b, v)) \\
&\qquad\qquad \land\ \mathrm{WF}_{\mathit{vars}}(\mathit{Propose}(b, v))
\end{aligned}
$$

$$\mathit{Liveness} \triangleq \Diamond(\mathit{chosen} \neq \{\})$$

$$\textsc{theorem}\ \mathit{LiveSpec} \Rightarrow \mathit{Liveness}$$

Instead, we will verify that all fairly-scheduled actions are eventually disabled, and that once this is the case we must have a decision

$$
\begin{array}{l}
\mathit{Liveness} \triangleq \\
\quad \forall\, a \in \mathit{Acceptor},\ b \in \mathit{Ballot},\ v \in \mathit{Value} : \\
\qquad \land\ \neg \textsc{enabled}\ \mathit{IncreaseCurrBal}(a,\ b) \\
\qquad \land\ \neg \textsc{enabled}\ \mathit{VoteFor}(a,\ b,\ v) \\
\qquad \land\ \neg \textsc{enabled}\ \mathit{Propose}(b,\ v) \\
\quad \Rightarrow \mathit{chosen} \neq \{\}
\end{array}
$$

Methodology, step 2: verify that all fairly-scheduled actions are self-disabling

Since we work in a finite domain, the property implies that, if those actions are fairly scheduled, then eventually all will be disabled

This implies that, if Liveness holds and the actions are fairly scheduled, then eventually one value is chosen

Methodology, step 2a: add ghost variable to remember which actions took place already

$\textsc{variables}\ votes,\ currBal,\ proposals,\ crashed,\ goodBallot,$
$\quad proposeTaken,\ voteForTaken,\ increaseCurrBalTaken$

$TypeOK \triangleq$
$\quad \land\ votes \in [Acceptor \rightarrow \textsc{subset}\ (Ballot \times Value)]$
$\quad \land\ currBal \in [Acceptor \rightarrow Ballot \cup \{-1\}]$
$\quad \land\ crashed \in \textsc{subset}\ Acceptor$
$\quad \land\ goodBallot \in Ballot$
$\quad \land\ proposals \in \textsc{subset}\ (Ballot \times Value)$
$\quad \land\ proposeTaken \in [Ballot \times Value \rightarrow \textsc{boolean}]$
$\quad \land\ voteForTaken \in [Acceptor \times Ballot \times Value \rightarrow \textsc{boolean}]$
$\quad \land\ increaseCurrBalTaken \in [Acceptor \times Ballot \rightarrow \textsc{boolean}]$

$Init \triangleq$
$\quad \land\ votes = [a \in Acceptor \mapsto \{\}]$
$\quad \land\ currBal = [a \in Acceptor \mapsto -1]$
$\quad \land\ crashed = \{\}$
$\quad \land\ proposals = \{\}$
$\quad \land\ goodBallot \in Ballot$
$\quad \land\ proposeTaken = [x \in Ballot \times Value \mapsto \textsc{false}]$
$\quad \land\ voteForTaken = [x \in Acceptor \times Ballot \times Value \mapsto \textsc{false}]$
$\quad \land\ increaseCurrBalTaken = [x \in Acceptor \times Ballot \mapsto \textsc{false}]$

$Propose(b, v) \triangleq$
$\quad \land\ Leader(b) \notin crashed$
$\quad \land\ \forall\, prop \in proposals : prop[1] \neq b$
$\quad \land\ \exists\, Q \in Quorum : ShowsSafeAt(Q, b, v)$
$\quad \land\ proposals' = proposals \cup \{\langle b, v \rangle\}$
$\quad \land\ proposeTaken' = [proposeTaken\ \textsc{except}\ ![\langle b, v \rangle] = \textsc{true}]$
$\quad \land\ \textsc{unchanged}\ \langle votes,\ currBal,\ crashed,\ goodBallot,$
$\qquad\quad voteForTaken,\ increaseCurrBalTaken \rangle$

$IncreaseCurrBal(a, b) \triangleq$
$\quad \land\ a \notin crashed$
$\quad \land\ goodBallot > -1 \Rightarrow b \leq goodBallot$
$\quad \land\ b > currBal[a]$
$\quad \land\ currBal' = [currBal\ \textsc{except}\ ![a] = b]$
$\quad \land\ increaseCurrBalTaken' =$
$\qquad [increaseCurrBalTaken\ \textsc{except}\ ![\langle a, b \rangle] = \textsc{true}]$
$\quad \land\ \textsc{unchanged}\ \langle votes,\ crashed,\ goodBallot,\ proposals,$
$\qquad\quad proposeTaken,\ voteForTaken \rangle$

$VoteFor(a, b, v) \triangleq$
$\quad \ldots$

Methodology, step 2b: assert and verify with an inductive invariant that all fair actions are self-disabling

$$
\begin{array}{l}
\mathit{SelfDisabling} \triangleq \\
\quad \forall\, a \in \mathit{Acceptor},\ b \in \mathit{Ballot},\ v \in \mathit{Value} : \\
\quad\ \wedge\ \ \mathit{voteForTaken}[\langle a,\ b,\ v\rangle] \Rightarrow \\
\qquad\quad \neg\textsc{enabled}\ \mathit{VoteFor}(a,\ b,\ v) \\
\quad\ \wedge\ \ \mathit{increaseCurrBalTaken}[\langle a,\ b\rangle] \Rightarrow \\
\qquad\quad \neg\textsc{enabled}\ \mathit{IncreaseCurrBal}(a,\ b) \\
\quad\ \wedge\ \ \mathit{proposeTaken}[\langle b,\ v\rangle] \Rightarrow \\
\qquad\quad \neg\textsc{enabled}\ \mathit{Propose}(b,\ v)
\end{array}
$$

Methodology, step 3: verify with an inductive invariant that, once all actions are disabled, we have a decision

$$\begin{array}{l} \mathit{GoodBallotIsMaxBallot} \triangleq \\ \quad \forall\, a \in \mathit{Acceptor},\ b \in \mathit{Ballot},\ v \in \mathit{Value} : \\ \qquad \langle b, v\rangle \in \mathit{proposals} \lor b = \mathit{currBal}[a] \\ \qquad \Rightarrow b \leq \mathit{goodBallot} \end{array}$$

$$\mathit{QuorumNotCrashed} \triangleq \exists\, Q \in \mathit{Quorum} : Q \cap \mathit{crashed} = \{\}$$

$$\begin{array}{l} \mathit{VoteForProposal} \triangleq \forall\, a \in \mathit{Acceptor},\ b \in \mathit{Ballot},\ v \in \mathit{Value} : \\ \quad \mathit{VotedFor}(a, b, v) \Rightarrow \langle b, v\rangle \in \mathit{proposals} \end{array}$$

$$\begin{array}{l} \mathit{ProposalsUnique} \triangleq \\ \quad \forall\, a1, a2 \in \mathit{Acceptor},\ b \in \mathit{Ballot},\ v1, v2 \in \mathit{Value} : \\ \qquad \langle b, v1\rangle \in \mathit{proposals} \land \langle b, v2\rangle \in \mathit{proposals} \Rightarrow v1 = v2 \end{array}$$

$$\mathit{GoodBallotLeaderNotCrashed} \triangleq \neg \mathit{Leader}(\mathit{goodBallot}) \in \mathit{crashed}$$

$$\begin{array}{l} \mathit{LivenessInvariant} \triangleq \\ \quad \land\ \mathit{TypeOK} \\ \quad \land\ \mathit{VoteForProposal} \\ \quad \land\ \mathit{ProposalsUnique} \\ \quad \land\ \mathit{GoodBallotIsMaxBallot} \\ \quad \land\ \mathit{QuorumNotCrashed} \\ \quad \land\ \mathit{GoodBallotLeaderNotCrashed} \end{array}$$

$$\textsc{theorem}\ \mathit{Spec} \Rightarrow \Box \mathit{LivenessInvariant}$$

$$\textsc{theorem}\ \mathit{LivenessInvariant} \Rightarrow \mathit{Liveness}$$

Verifying liveness of eventually-synchronous consensus protocols, in a nutshell

1. We modify the specification to non-deterministically pick a good ballot B and enforce that no higher ballots are started and that the leader of B is well-behaved
2. We verify that each fair action is self-disabling. Under fair scheduling, this means that all (the finitely many) actions will eventually be disabled.
3. We verify that, once all actions are disabled, we have a decision

In steps 2 and 3 we verify safety properties
We can use an inductive invariant verified with Apalache

We are able to check the liveness of TetraBFT for interesting problem sizes

Problem size

- 3 rounds (with 5 sub-phases per round)
- 3 possible decision values
- 3 nodes n_1, n_2, n_3
- Either n_2 or n_3 are malicious (but not both)
- Quorums are $\{n_1,n_2\}$ and $\{n_1,n_3\}$

Resources used by Apalache

- ~ few hours
- ~ 40 GB of RAM
- 16 cores on a recent (2022) desktop machine

Specifications are available at
https://github.com/nano-o/tetrabft-tla

Future work

- Prove what we assumed about timer management, i.e. that eventually, there is a ballot led by a well-behaved node such that no node ever starts a higher ballot (Paxos)
    - Requires a real-time model of the behavior after GST
- Same thing for TetraBFT, where this is more complex: round progression must be governed by a so-called BFT synchronizer algorithm