The Stellar Consensus Protocol (SCP)

draft-mazieres-dinrg-scp-00

Nicolas Barry, David Mazières, Jed McCaleb, Stanislas Polu

IETF101

Monday, March 19, 2018

An open Byzantine agreement protocol

Majority-based voting doesn't work against Sybil attacks

Instead, determine quorums in decentralized way based on trust

- Let $\mathbf{V}$ be all nodes in the world
- Each $v \in \mathbf{V}$ would accept any of $\mathbf{Q}(v) = \{q_1, \ldots, q_n\}$ as a quorum
- But q_i is *not* a quorum—it is a quorum slice
- A quorum must (transitively) satisfy *all* of its members

Definition (Quorum)

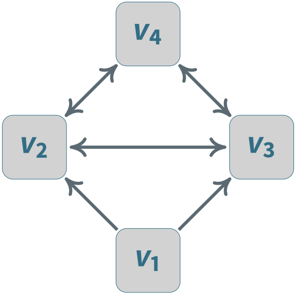
A quorum $U \subseteq \mathbf{V}$ is a set of nodes that contains at least one slice of each of its members: $\forall v \in U, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq U$

Assumes trust overlaps transitively. Analogies:

- Transitive reachability on the Internet
- Rough agreement on who constitutes a tier-1 ISP
- Overlapping notions of valid certificate authorities

Definition (Quorum)

A quorum $U \subseteq \mathbf{V}$ is a set of nodes that encompasses at least one slice of each of its members: $\forall v \in U, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq U$



$$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3\}\}$$
$$\mathbf{Q}(v_2) = \mathbf{Q}(v_3) = \mathbf{Q}(v_4) = \{\{v_2, v_3, v_4\}\}$$

Visualize quorum slice dependencies with arrows

v_2, v_3, v_4 **is a quorum—contains a slice of each member**

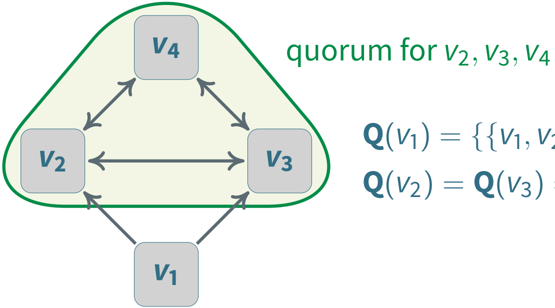
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- Doesn't contain a slice for v_2, v_3, who demand v_4's agreement

$v_1, \ldots, v_4$ **is the smallest quorum containing** v_1

Definition (Quorum)

A quorum $U \subseteq \mathbf{V}$ is a set of nodes that encompasses at least one slice of each of its members: $\forall v \in U, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq U$



$$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3\}\}$$
$$\mathbf{Q}(v_2) = \mathbf{Q}(v_3) = \mathbf{Q}(v_4) = \{\{v_2, v_3, v_4\}\}$$

Visualize quorum slice dependencies with arrows

v_2, v_3, v_4 **is a quorum—contains a slice of each member**

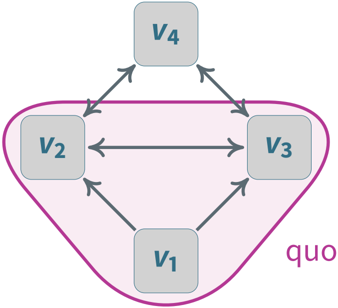
v_1, v_2, v_3 **is a slice for v_1, but not a quorum**

- Doesn't contain a slice for v_2, v_3, who demand v_4's agreement

$v_1, \ldots, v_4$ **is the smallest quorum containing v_1**

Definition (Quorum)

A quorum $U \subseteq \mathbf{V}$ is a set of nodes that encompasses at least one slice of each of its members: $\forall v \in U, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq U$



$$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3\}\}$$
$$\mathbf{Q}(v_2) = \mathbf{Q}(v_3) = \mathbf{Q}(v_4) = \{\{v_2, v_3, v_4\}\}$$

quorum slice for v_1, but not a quorum

Visualize quorum slice dependencies with arrows

v_2, v_3, v_4 **is a quorum—contains a slice of each member**

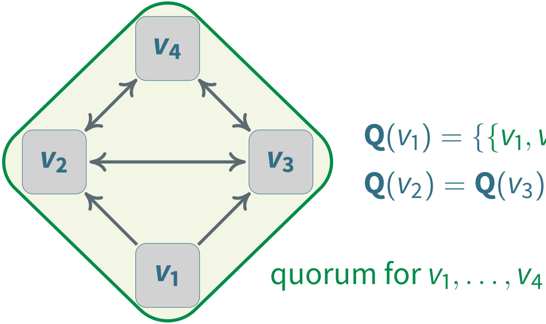
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- Doesn't contain a slice for v_2, v_3, who demand v_4's agreement

$v_1, \ldots, v_4$ **is the smallest quorum containing** v_1

Definition (Quorum)

A quorum $U \subseteq \mathbf{V}$ is a set of nodes that encompasses at least one slice of each of its members: $\forall v \in U, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq U$



$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3\}\}$

$\mathbf{Q}(v_2) = \mathbf{Q}(v_3) = \mathbf{Q}(v_4) = \{\{v_2, v_3, v_4\}\}$

Visualize quorum slice dependencies with arrows

v_2, v_3, v_4 **is a quorum—contains a slice of each member**

v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- Doesn't contain a slice for v_2, v_3, who demand v_4's agreement

$v_1, \ldots, v_4$ **is the smallest quorum containing** v_1

Quorum slice representation

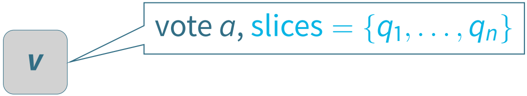
```
union PublicKey switch (PublicKeyType type) {
case PUBLIC_KEY_TYPE_ED25519:
   uint256 ed25519;
};

// supports things like: A,B,C,(D,E,F),(G,H,(I,J,K,L))
// only allows 2 levels of nesting
struct SCPQuorumSet {
   uint32 threshold;        // the k in k-of-n
   PublicKey validators<>;
   SCPQuorumSet innerSets<>;
};
```

Can't represent arbitrary quorum slices compactly

Instead, use two-levels of k-of-n configuration

Federated voting



Nodes exchanges vote messages to agree on statements

- Well-behaved nodes cannot vote for contradictory statements
- Every vote specifies quorum slices
- Allows dynamic quorum discovery while assembling votes

Two important thresholds for statement a at node v:

- quorum threshold – a quorum containing v unanimously votes for a
- blocking threshold – $\forall q \in \mathbf{Q}(v), \exists v' \in q$ such that v' voted for a (no contradictory $\overline{a} \neq a$ can reach quorum threshold w/o illegal votes)

v ratifies a iff a reaches quorum threshold at v

- Can't ratify contradictory statements if you have quorum intersection despite [i.e., after deleting] ill-behaved nodes (qidin)

Vote messages

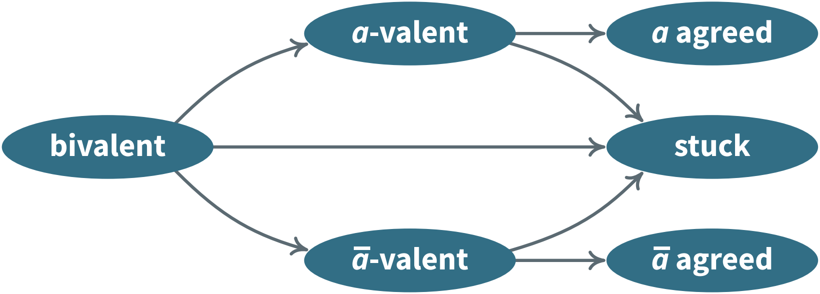
```
typedef opaque Hash[32];  // SHA-256
struct SCPStatement {
   PublicKey nodeID;      // v (node signing message)
   uint64 slotIndex;
   Hash quorumSetHash;
   SCPStatement pledges;
};

typedef opaque Signature<64>;
struct SCPEnvelope {
   SCPStatement statement;
   Signature signature;
};
```

Transmit quorum slices as SHA-256 hash of SCPQuorumSet

- Use side protocol to request preimage if not cached

Federated voting outcomes



Before any node votes, system is bivalent

- Any value may be ratified

If a node ratifies a, system is a-valent

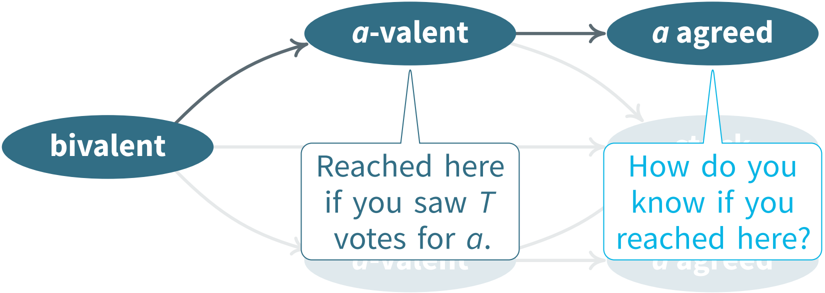
- With qidin, no contradictory $\bar{a}$ can be ratified

If every node learns system a-valent, then system agrees on a

System can also get stuck at any point along the way

- Non-faulty node can't ratify a because voting for $\bar{a}$
- Or ratified a and don't know it because of crash & message loss

When have we reached agreement?



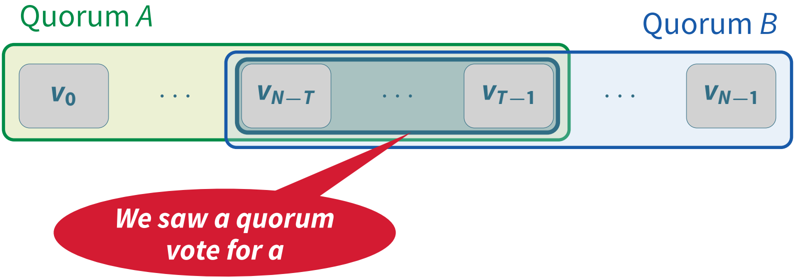
Centralized protocols (e.g., PBFT) accept statement if quorum intersection says ratified

- Centralized systems care about whole-system failure, not per-node
- Now can't assume correctness of quorums you don't belong to

First-hand ratification now the only way to know system *a*-valent

- How to agree on statement *a* even after voting against it?
- How to know everyone else will learn system agreed on *a*?

When have we reached agreement?



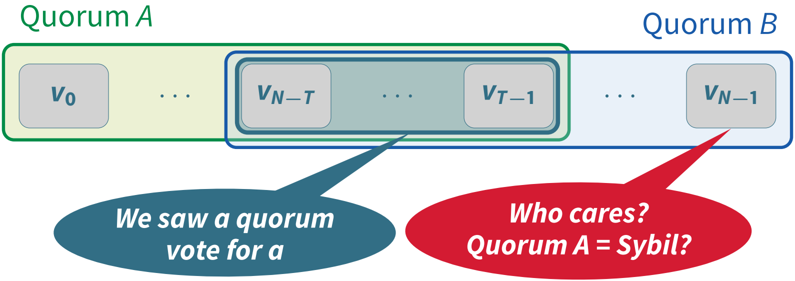
Centralized protocols (e.g., PBFT) accept statement if quorum intersection says ratified

- Centralized systems care about whole-system failure, not per-node
- Now can't assume correctness of quorums you don't belong to

First-hand ratification now the only way to know system a-valent

- How to agree on statement a even after voting against it?
- How to know everyone else will learn system agreed on a?

When have we reached agreement?



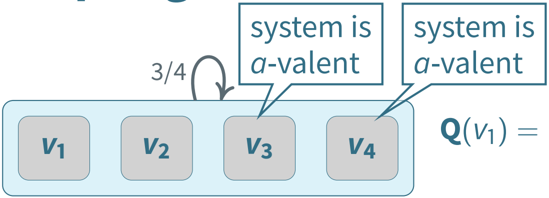
Centralized protocols (e.g., PBFT) accept statement if quorum intersection says ratified

- Centralized systems care about whole-system failure, not per-node
- Now can't assume correctness of quorums you don't belong to

First-hand ratification now the only way to know system *a*-valent

- How to agree on statement *a* even after voting against it?
- How to know everyone else will learn system agreed on *a*?

Accepting statements



$$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3\}, \{v_1, v_2, v_4\}, \{v_1, v_3, v_4\}\}$$

What if "system is a-valent" reaches blocking threshold at v_1?

- Either true or v_1 not member of any well-behaved quorum (no liveness)

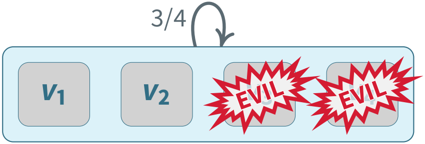
Node v accepts a statement a consistent with history iff either:

1. "I vote a or I accept a" reaches quorum threshold, or
2. "I accept a" reaches blocking threshold

#2 lets nodes accept statements they voted against, but

- Nodes can accept contradictory statements in cases with no fully honest quorum but where you still have qidin
- No guarantee all nodes in non-faulty quorum will accept a

Accepting statements



$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3\}, \{v_1, v_2, v_4\}, \{v_1, v_3, v_4\}\}$

What if "system is a-valent" reaches blocking threshold at v_1?

- Either true or v_1 not member of any well-behaved quorum (no liveness)

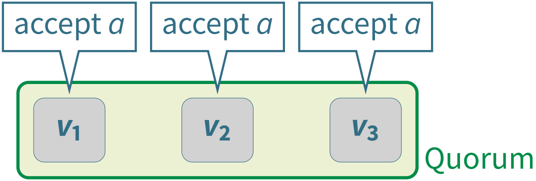
Node v accepts a statement a consistent with history iff either:

1. "I vote a or I accept a" reaches quorum threshold, or
2. "I accept a" reaches blocking threshold

#2 lets nodes accept statements they voted against, but

- Nodes can accept contradictory statements in cases with no fully honest quorum but where you still have qidin
- No guarantee all nodes in non-faulty quorum will accept a

Confirming statements



Idea: Hold a second vote on the fact that the first vote succeeded

Node v confirms a by ratifying "I accepted a."

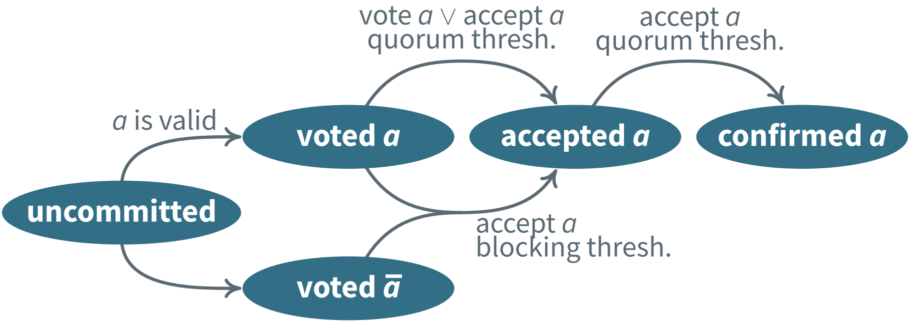
Solves safety through quorum threshold of ratification

Also solves nodes in honest quorum being unable to accept

- Nodes in well-behaved quorum may vote against accepted statements
- Won't vote against the *fact* that those statements were accepted

Theorem: If 1 node in well-behaved quorum confirms a, all will

Summary of federated voting process



A node v that locally confirms a knows system has agreed on a

- If **Q**() admits any safe protocol, well-behaved nodes can't contradict a
- If v in well-behaved quorum, whole quorum will eventually confirm a

SCP nomination message

```
typedef opaque Value<>;

struct SCPNomination {
  Value votes<>;     // vote to nominate these values
  Value accepted<>;  // assert that these are accepted
};

union SCPStatement switch (SCPStatementType type) {
  case SCP_ST_NOMINATE:
    SCPNomination nominate;
  /* ... */
};
```

Nodes broadcast nominated values in `votes`

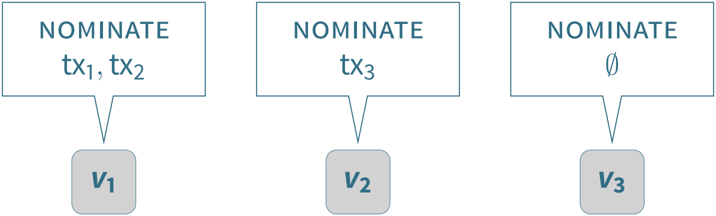
- Initially vote values in all received votes (ignoring optimization here)

Upon accepting nomination of a, move from `votes` to `accepted`

Stop voting for new values when any confirmed nominated

- But continue accepting and repeating votes already cast

Nomination flow



Nodes nominate values and re-nominate any nominations seen

Stop adding to `votes` once any value confirmed nominated

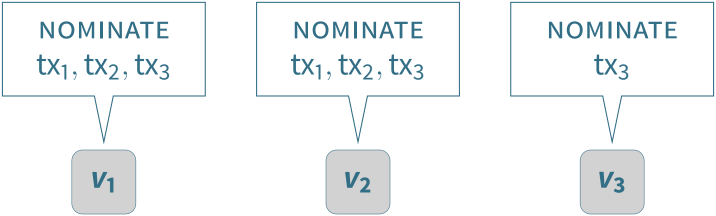
Converge on set of nominated values

Deterministically combine nominations into *composite* value x

All nodes guaranteed to converge on same value x!

- Complication: impossible to know when protocol has converged [FLP]
- c.f. asynchronous reliable broadcast

Nomination flow



Nodes nominate values and re-nominate any nominations seen

Stop adding to `votes` once any value confirmed nominated

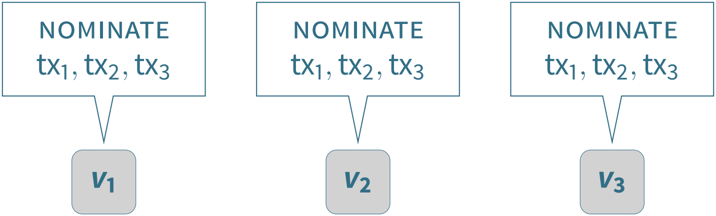
Converge on set of nominated values

Deterministically combine nominations into *composite* value x

All nodes guaranteed to converge on same value x!

- Complication: impossible to know when protocol has converged [FLP]
- c.f. asynchronous reliable broadcast

Nomination flow



Nodes nominate values and re-nominate any nominations seen

Stop adding to `votes` once any value confirmed nominated

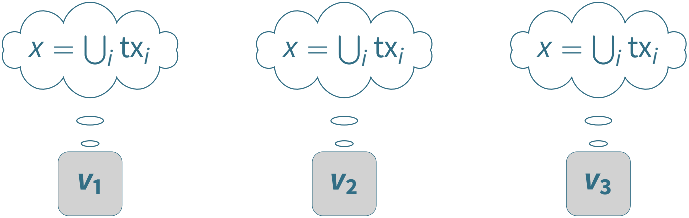
Converge on set of nominated values

Deterministically combine nominations into *composite* value x

All nodes guaranteed to converge on same value x!

- Complication: impossible to know when protocol has converged [FLP]
- c.f. asynchronous reliable broadcast

Nomination flow



Nodes nominate values and re-nominate any nominations seen

Stop adding to `votes` once any value confirmed nominated

Converge on set of nominated values

Deterministically combine nominations into *composite* value x

All nodes guaranteed to converge on same value x!

- Complication: impossible to know when protocol has converged [FLP]
- c.f. asynchronous reliable broadcast

SCP ballots

```
struct SCPBallot {
  uint32 counter;              // n
  Value value;                 // x
};
```

Composite nominated must be run through balloting

- Guarantees safety even if started before nomination converges

A ballot b is a pair $\langle b.n, b.x\rangle$ where $b.x$ is a candidate output value

- Ballots totally ordered with field n more significant than x
- Nodes may vote to commit or abort a ballot, not both
- If federated voting confirms commit b for any b, can output value $b.x$

Let prepared$(b) = \{$abort $b_{\mathbf{old}} \mid b_{\mathbf{old}} < b$ and $b_{\mathbf{old}}.x \neq b.x\}$

Invariant: cannot vote commit b unless federated voting has confirmed every statement in prepared(b)

SCP prepare message

```
struct SCPPrepare {
  SCPBallot ballot;           // b
  SCPBallot *prepared;        // p
  SCPBallot *preparedPrime; // p'
  uint32 nC;                  // c.n
  uint32 nH;                  // h.n
};

union SCPStatement switch (SCPStatementType type) {
  case SCP_ST_PREPARE:
    SCPPrepare prepare;
  /* ... */
};
```

Prepare fields

`ballot.`*x* starts at 1, increases w. timeouts, msg receipt

`ballot.`*n* *b.x* from highest *b* for which prepared(b) confirmed (if any) otherwise composite nomination value

`prepared` highest *b* for which sender accepted prepared(*b*)

`prepared`′ highest *b* with accepted prepared(*b*) and different *x* from `prepared`

`nH` *b.n* from highest *b* with confirmed prepared(*b*), else 0

`nC` if not 0 and `ballot.`$x = 1$, implies votes for commit $\langle nC, x\rangle$, commit $\langle nC + 1, x\rangle, \ldots,$ commit $\langle nH, x\rangle$

SCP confirm message

```
struct SCPConfirm {
    SCPBallot ballot;    // b
    uint32 nPrepared;    // p.n
    uint32 nCommit;      // c.n
    uint32 nH;           // h.n
};

union SCPStatement switch (SCPStatementType type) {
  case SCP_ST_CONFIRM:
    SCPConfirm confirm;
  /* ... */
};
```

Implies votes for all messages in the set $\{\textbf{accept}(\textbf{commit}\ b') \mid \texttt{nCommit} \le b'.n \le \texttt{nH}\ \textbf{and}\ b'.x = \texttt{ballot.x}\}$

Implies `SCPPrepare` **with ballot** $\langle \infty, \texttt{confirm.ballot}.x \rangle$**, prepared** $\langle \texttt{confirm.nPrepared}, \texttt{confirm.ballot}.x \rangle$**, and nH value** ∞**.**

SCP externalize message

```
struct SCPExternalize {
    SCPBallot commit;      // c
    uint32 nH;             // h.n
};

union SCPStatement switch (SCPStatementType type) {
  case SCP_ST_EXTERNALIZE:
    SCPExternalize externalize;
  /* ... */
};
```

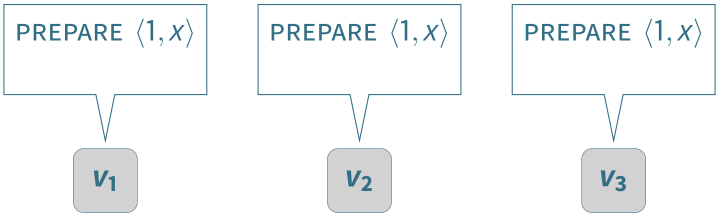
By the time you send this, already externalized `commit`.x

- Means you have confirmed committed a ballot with `commit`.x
- Goal is definitive record to help other nodes prove value/catch up

Implies `SCPConfirm` **with ballot** $\langle\infty,$ `externalize.commit`.$x\rangle$, **nPrepared** `externalize.commit`.n, **and nH** ∞

Implies `SCPConfirm` **with ballot** $\langle\infty,$ `externalize.commit`.$x\rangle$, **nPrepared** `externalize.commit`.n, **nH** `externalize.nH`, **and a special quorum slice declaration of only the sending node**

Balloting flow



In the common case, will prepare and commit nominated value

Else, arm timer when ballot counter reaches quorum threshold

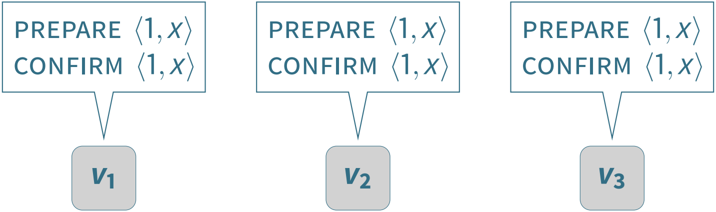
Bump counter and restart with new ballot whenever

- Timer fires
- A blocking threshold is at a higher ballot counter

Nomination may finish converging in background

Or if any value confirmed prepared, all nodes will eventually see it confirmed prepared and start using that value

Balloting flow



In the common case, will prepare and commit nominated value

Else, arm timer when ballot counter reaches quorum threshold

Bump counter and restart with new ballot whenever

- Timer fires
- A blocking threshold is at a higher ballot counter

Nomination may finish converging in background

Or if any value confirmed prepared, all nodes will eventually see it confirmed prepared and start using that value

Questions?