Fast and secure global payments with Stellar

Marta Lokhava, Giuliano Losa (Galois), David Mazières, Graydon Hoare,
Nicolas Barry, Eli Gafni (UCLA), Jonathan Jove, Rafał Malinowsky, and Jed McCaleb

June 1, 2020

Administrivia

No lecture Wednesday—I will have office hours instead

Project presentations 3pm-4:20pm Mon., Tues., Wed. of next week (May 8, 9, 10)

Prepare a 7-minute presentation (+1 minute of questions)

- Primarily present slides
- Also show a brief demo if possible
- Can switch off presenting, or just have one group member present

Schedule will be posted shortly

- Please let us know if you can't make time, or have to leave at 4:20pm

You will likely want to share your desktop for a demo

- Please avoid any embarrassing/distracting window content

Obligatory disclaimer

“Prof. Mazières’s contribution to this work was as a paid consultant, and was not part of his Stanford University duties or responsibilities.”

Things we take for granted

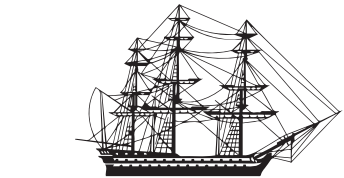
A bank account in a stable currency such as USD

Access to well-regulated investments

Cheap international money transfers

Globally accepted, fee-free credit cards

Things we take for granted



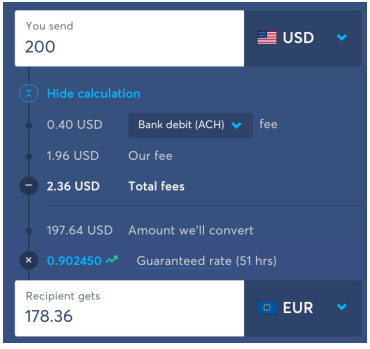
A bank account in a stable currency such as USD

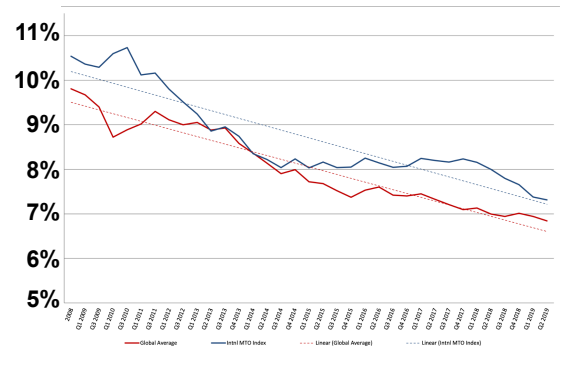
Access to well-regulated investments

Cheap international money transfers

Globally accepted, fee-free credit cards

Things we take for granted





A bank account in a stable currency such as USD

Access to well-regulated investments

Cheap international money transfers

Globally accepted, fee-free credit cards

Things we take for granted

All No Foreign Transaction Fee Cards

m-pesa

A bank account in a stable currency such as USD

Access to well-regulated investments

Cheap international money transfers

Globally accepted, fee-free credit cards

Stellar: equitable access to the financial system

1. **Open membership**
   - Anyone can issue, trade, and hold assets
   - All developers access the same API, from students to Franklin Templeton or IBM
2. **Issuer-enforced finality**
   - Security of issued tokens depends only on issuer (what we expect today)
   - Still need secure servers, but issuer owns or designates them
3. **Cross-issuer atomicity**
   - Trade any asset for any other (ensures you can bootstrap markets)
   - Get the best price on any trade without trusting your trading partner
   - Atomically trade through multiple assets w/o exchange-rate risk
     (E.g., trade NGN $\rightarrow$ Sketchy-Asset $\rightarrow$ PHP with no risk from Sketchy-Asset)

Non-solutions

Nacha

Extend national payment network (ACH, SEPA, UPI) globally

- Requires compliance with national regulations, closed to new assets

Everyone just issues and manages their own assets

- Can't pay or trade across systems, closed to new assets

Move Paypal onto Ethereum as an ERC-20 token

- Double redemption risk not under issuer's control

Non-solutions

Extend national payment network (ACH, SEPA, UPI) globally

- Requires compliance with national regulations, closed to new assets

Everyone just issues and manages their own assets

- Can’t pay or trade across systems, closed to new assets

Move Paypal onto Ethereum as an ERC-20 token

- Double redemption risk not under issuer’s control

Non-solutions

Name	Symbol	Market Cap	Algorithm	Hash Rate	1h Attack Cost	NiceHash-able
Bitcoin	BTC	$176.48 B	SHA-256	91,832 PH/s	*$326,796*	0%
Ethereum	ETH	$26.58 B	Ethash	167 TH/s	*$146,621*	4%
BitcoinCashABC	BCH	$4.49 B	SHA-256	2,524 PH/s	*$8,982*	13%
BitcoinSV	BSV	$3.58 B	SHA-256	2,310 PH/s	*$8,219*	14%
Litecoin	LTC	$3.06 B	Scrypt	253 TH/s	*$16,436*	5%
EthereumClassic	ETC	$815.84 M	Ethash	6 TH/s	*$5,464*	98%

Extend national payment network (ACH, SEPA, UPI) globally

- Requires compliance with national regulations, closed to new assets

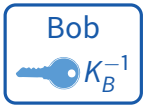
Everyone just issues and manages their own assets

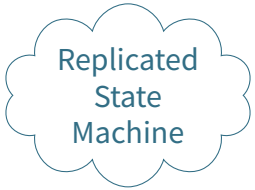
- Can’t pay or trade across systems, closed to new assets

Move Paypal onto Ethereum as an ERC-20 token

- Double redemption risk not under issuer’s control

Stellar transaction model





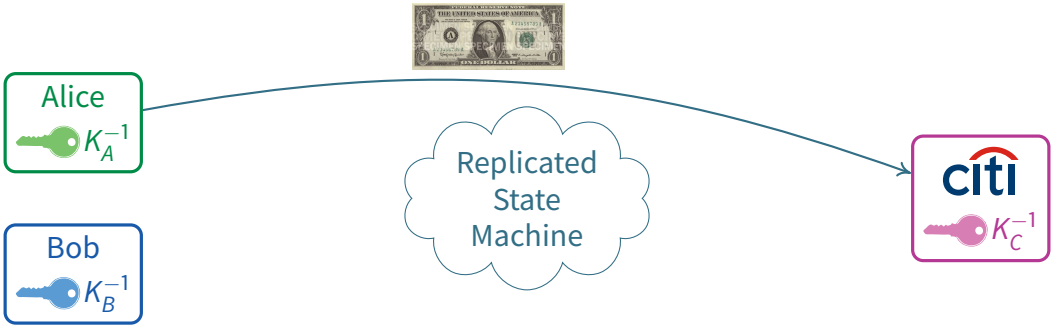
Global replicated state machine (RSM) executes transactions to keep ledger state

- Accounts named by public key authorizing operations on the account
- Accounts can issue assets; issuing account part of asset name

Transactions guarantee atomicity

- Multiple operations from multiple accounts with either all succeed or all fail
- *Path payments* atomically trade through multiple assets (e.g., $1\,K_D\$ \rightarrow 1\,K_C\$ \rightarrow 1\,K_B$ babysit)

Stellar transaction model



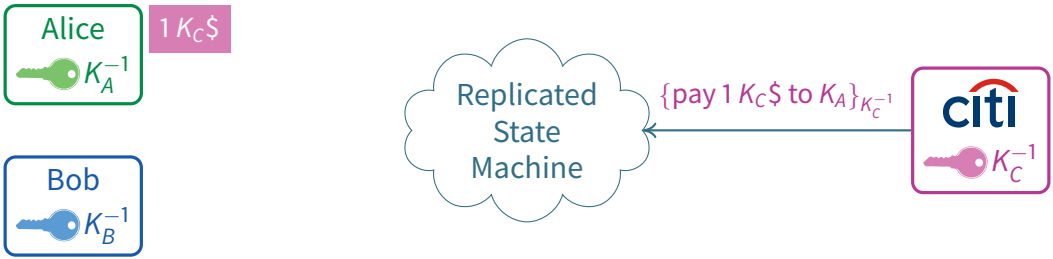
Global replicated state machine (RSM) executes transactions to keep ledger state

- Accounts named by public key authorizing operations on the account
- Accounts can issue assets; issuing account part of asset name

Transactions guarantee atomicity

- Multiple operations from multiple accounts with either all succeed or all fail
- *Path payments* atomically trade through multiple assets (e.g., $1\,K_D\$ \rightarrow 1\,K_C\$ \rightarrow 1\,K_B$ babysit)

Stellar transaction model



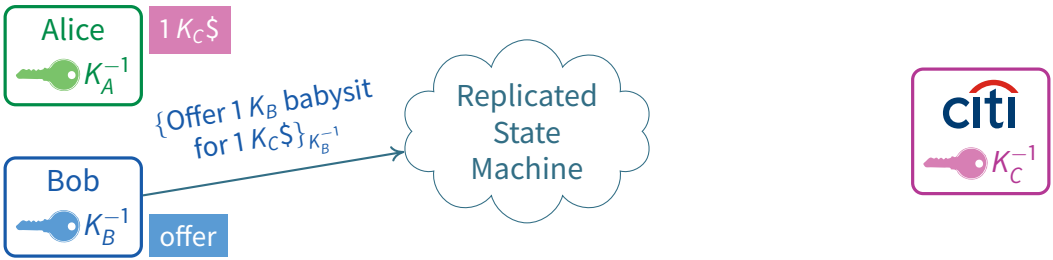
Global replicated state machine (RSM) executes transactions to keep ledger state

- Accounts named by public key authorizing operations on the account
- Accounts can issue assets; issuing account part of asset name

Transactions guarantee atomicity

- Multiple operations from multiple accounts with either all succeed or all fail
- *Path payments* atomically trade through multiple assets (e.g., $1\,K_D\$ \rightarrow 1\,K_C\$ \rightarrow 1\,K_B$ babysit)

Stellar transaction model



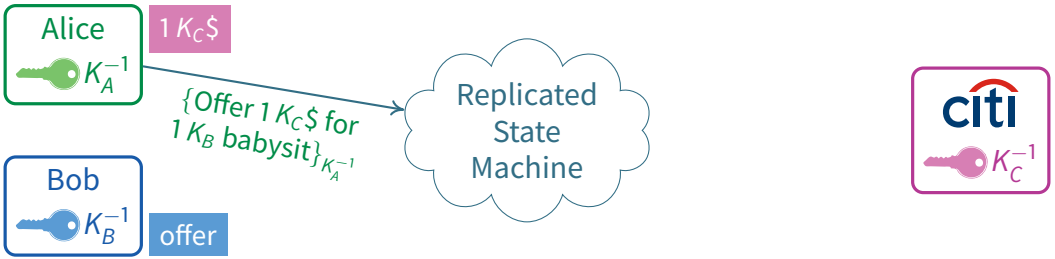
Global replicated state machine (RSM) executes transactions to keep ledger state

- Accounts named by public key authorizing operations on the account
- Accounts can issue assets; issuing account part of asset name

Transactions guarantee atomicity

- Multiple operations from multiple accounts with either all succeed or all fail
- *Path payments* atomically trade through multiple assets (e.g., $1\,K_D\$ \rightarrow 1\,K_C\$ \rightarrow 1\,K_B$ babysit)

Stellar transaction model



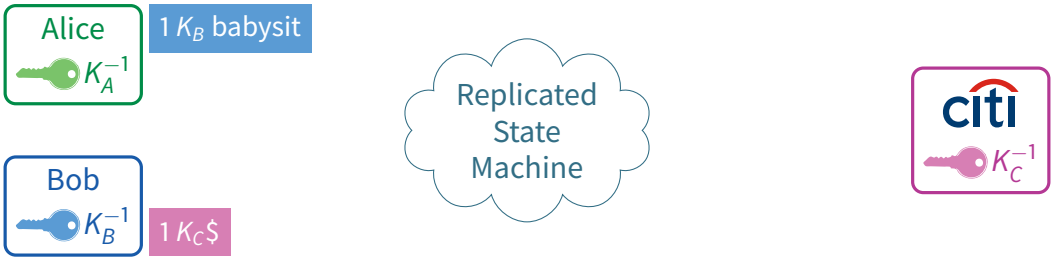
Global replicated state machine (RSM) executes transactions to keep ledger state

- Accounts named by public key authorizing operations on the account
- Accounts can issue assets; issuing account part of asset name

Transactions guarantee atomicity

- Multiple operations from multiple accounts with either all succeed or all fail
- *Path payments* atomically trade through multiple assets (e.g., 1 $K_D$$ $\rightarrow$ 1 $K_C$$ $\rightarrow$ 1 K_B babysit)

Stellar transaction model



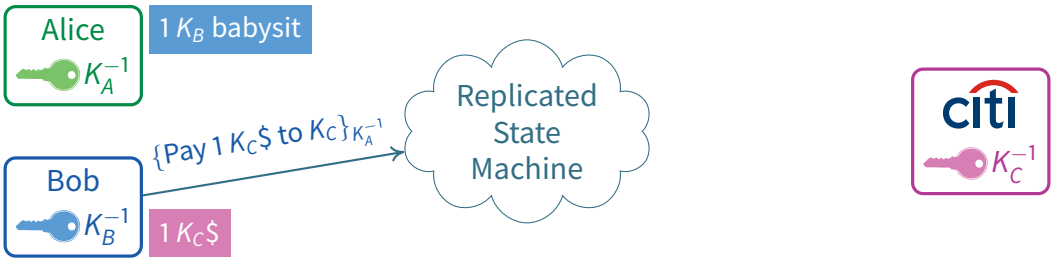
Global replicated state machine (RSM) executes transactions to keep ledger state

- Accounts named by public key authorizing operations on the account
- Accounts can issue assets; issuing account part of asset name

Transactions guarantee atomicity

- Multiple operations from multiple accounts with either all succeed or all fail
- *Path payments* atomically trade through multiple assets (e.g., $1\,K_D\$ \rightarrow 1\,K_C\$ \rightarrow 1\,K_B$ babysit)

Stellar transaction model



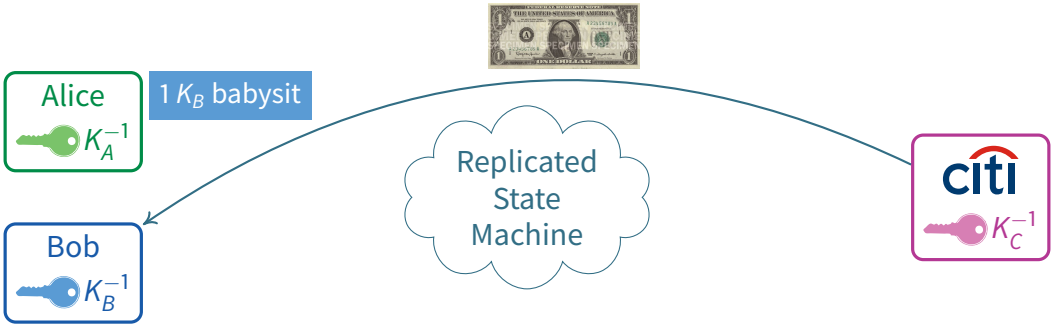
Global replicated state machine (RSM) executes transactions to keep ledger state

- Accounts named by public key authorizing operations on the account
- Accounts can issue assets; issuing account part of asset name

Transactions guarantee atomicity

- Multiple operations from multiple accounts with either all succeed or all fail
- *Path payments* atomically trade through multiple assets (e.g., $1\,K_D\$ \to 1\,K_C\$ \to 1\,K_B$ babysit)

Stellar transaction model



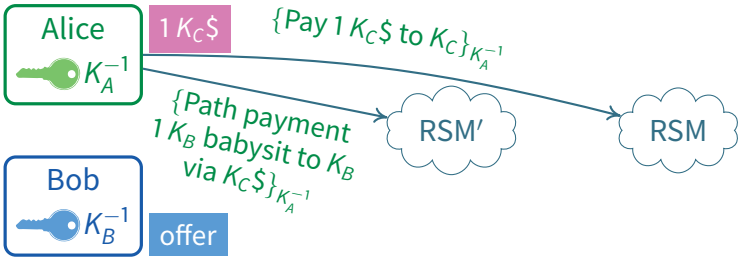
Global replicated state machine (RSM) executes transactions to keep ledger state

- Accounts named by public key authorizing operations on the account
- Accounts can issue assets; issuing account part of asset name

Transactions guarantee atomicity

- Multiple operations from multiple accounts with either all succeed or all fail
- *Path payments* atomically trade through multiple assets (e.g., $1\,K_D\$ \rightarrow 1\,K_C\$ \rightarrow 1\,K_B$ babysit)

How to guarantee ledger integrity?



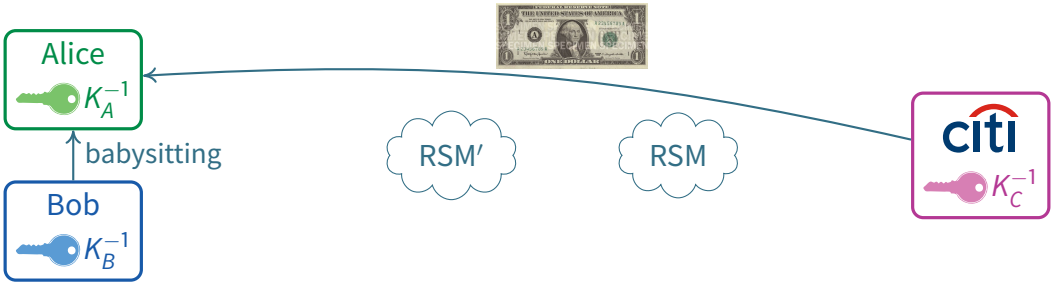
Model only works if everyone agrees on ledger state

- If ledger forks, system vulnerable to *double-spend attack*
- E.g., Alice gets both babysitting and $1, Bob can't redeem $K_C$$

Solution: Bob had better *follow* the server Citi uses to redeem $K_C$$

- Unless/until that server agrees, Bob shouldn't recognize Alice's babysitting credit

How to guarantee ledger integrity?



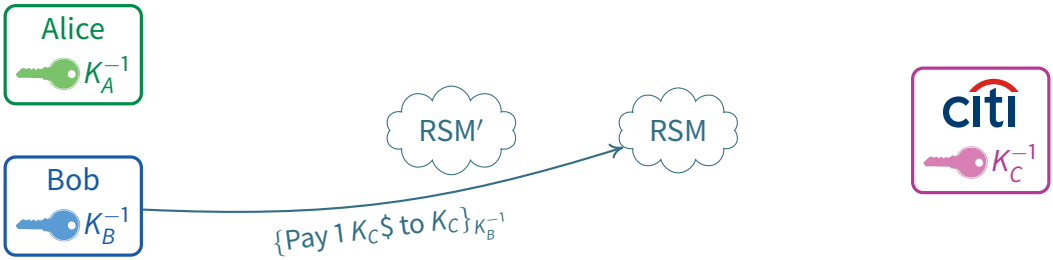
Model only works if everyone agrees on ledger state

- If ledger forks, system vulnerable to *double-spend attack*
- E.g., Alice gets both babysitting and $1, Bob can't redeem K_C\$

Solution: Bob had better *follow* the server Citi uses to redeem K_C\$

- Unless/until that server agrees, Bob shouldn't recognize Alice's babysitting credit

How to guarantee ledger integrity?



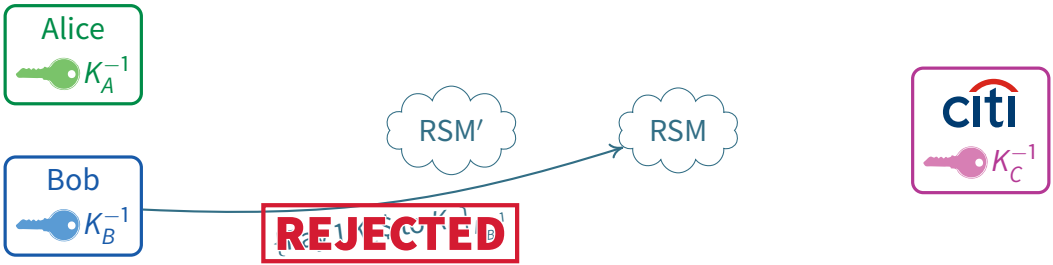
Model only works if everyone agrees on ledger state

- If ledger forks, system vulnerable to *double-spend attack*
- E.g., Alice gets both babysitting and $1, Bob can't redeem $K_C$$

Solution: Bob had better *follow* the server Citi uses to redeem $K_C$$

- Unless/until that server agrees, Bob shouldn't recognize Alice's babysitting credit

How to guarantee ledger integrity?



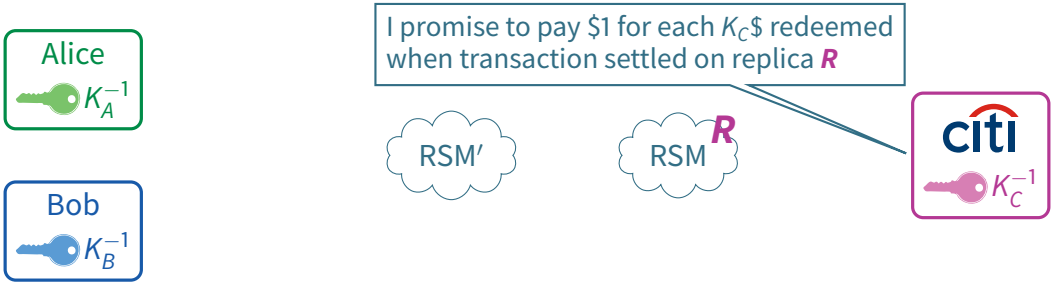
Model only works if everyone agrees on ledger state

- If ledger forks, system vulnerable to *double-spend attack*
- E.g., Alice gets both babysitting and $1, Bob can't redeem K_C\$

Solution: Bob had better *follow* the server Citi uses to redeem K_C\$

- Unless/until that server agrees, Bob shouldn't recognize Alice's babysitting credit

How to guarantee ledger integrity?



Model only works if everyone agrees on ledger state

- If ledger forks, system vulnerable to *double-spend attack*
- E.g., Alice gets both babysitting and $1, Bob can't redeem $K_C$$

Solution: Bob had better *follow* the server Citi uses to redeem $K_C$$

- Unless/until that server agrees, Bob shouldn't recognize Alice's babysitting credit

The Internet hypothesis

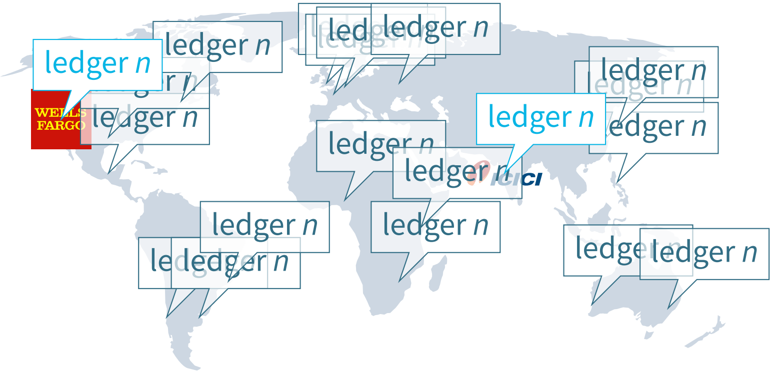
Will two organizations that don't follow each other agree on ledger state?

- Yes if the follow graph transitively converges

Hypothesis: any two nodes you'd care about transitively follow a common node

- Empirically true of Internet (e.g., China⟷Stanford⟷Google) and legacy payments
- If they don't, maybe it doesn't matter (risk limited to in-flight transactions)

The Internet hypothesis



Will two organizations that don’t follow each other agree on ledger state?

- Yes if the follow graph transitively converges

Hypothesis: any two nodes you’d care about transitively follow a common node

- Empirically true of Internet (e.g., China⟷Stanford⟷Google) and legacy payments
- If they don’t, maybe it doesn’t matter (risk limited to in-flight transactions)

Consensus from the Internet hypothesis

Stellar consensus protocol (SCP) secures Stellar ledger

- Safety and liveness formally verified for arbitrary configurations meeting requirements

Key idea: broadcast protocol steps that you will take if nodes you follow will, too

- Take steps if and only if all nodes mutually satisfied

For availability, follow multiple acceptable sets of peers, called *quorum slices*

- Take step if any quorum slice unanimously willing
- E.g., slices(v_1) = all sets comprising a majority from each of 3 organizations

Definition (Quorum)

A *quorum* is a non-empty set of nodes containing a slice of each non-faulty member.

Consensus from the Internet hypothesis

Stellar consensus protocol (SCP) secures Stellar ledger

- Safety and liveness formally verified for arbitrary configurations meeting requirements

Key idea: broadcast protocol steps that you will take if nodes you follow will, too

- Take steps if and only if all nodes mutually satisfied

For availability, follow multiple acceptable sets of peers, called *quorum slices*

- Take step if any quorum slice unanimously willing
- E.g., slices(v_1) = all sets comprising a majority from each of 3 organizations

Definition (Quorum)

A *quorum* is a non-empty set of nodes containing a slice of each non-faulty member.

Consensus from the Internet hypothesis

Stellar consensus protocol (SCP) secures Stellar ledger

- Safety and liveness formally verified for arbitrary configurations meeting requirements

Key idea: broadcast protocol steps that you will take if nodes you follow will, too

- Take steps if and only if all nodes mutually satisfied

For availability, follow multiple acceptable sets of peers, called *quorum slices*

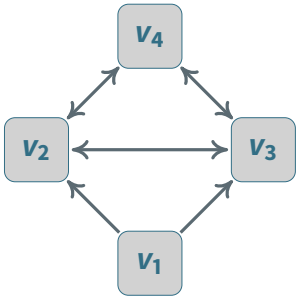
- Take step if any quorum slice unanimously willing
- E.g., slices(v_1) = all sets comprising a majority from each of 3 organizations

Definition (Quorum)

A *quorum* is a non-empty set of nodes containing a slice of each non-faulty member.

Definition (Quorum)

A *quorum* is a non-empty set of nodes containing a slice of each non-faulty member.



$$\text{slices}(v_1) = \{\{v_1, v_2, v_3\}\}$$
$$\text{slices}(v_2) = \text{slices}(v_3) = \text{slices}(v_4) = \{\{v_2, v_3, v_4\}\}$$

Visualize quorum slice dependencies with arrows

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

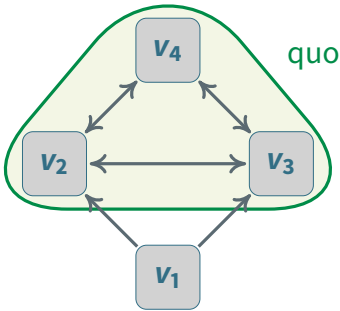
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- 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* is a non-empty set of nodes containing a slice of each non-faulty member.



quorum for v_2, v_3, v_4

$\text{slices}(v_1) = \{\{v_1, v_2, v_3\}\}$

$\text{slices}(v_2) = \text{slices}(v_3) = \text{slices}(v_4) = \{\{v_2, v_3, v_4\}\}$

Visualize quorum slice dependencies with arrows

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

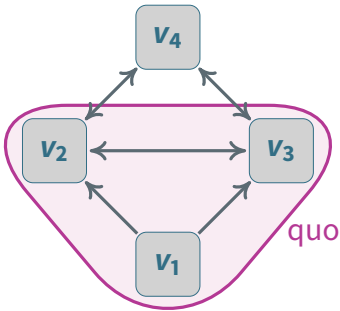
v_1, v_2, v_3 is a slice for v_1, but not a quorum

- 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* is a non-empty set of nodes containing a slice of each non-faulty member.



$\text{slices}(v_1) = \{\{v_1, v_2, v_3\}\}$

$\text{slices}(v_2) = \text{slices}(v_3) = \text{slices}(v_4) = \{\{v_2, v_3, v_4\}\}$

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**

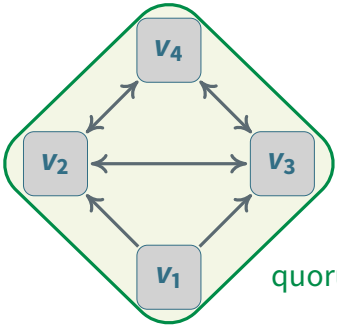
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- 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* is a non-empty set of nodes containing a slice of each non-faulty member.



$\text{slices}(v_1) = \{\{v_1, v_2, v_3\}\}$

$\text{slices}(v_2) = \text{slices}(v_3) = \text{slices}(v_4) = \{\{v_2, v_3, v_4\}\}$

quorum for $v_1, \ldots, v_4$

Visualize quorum slice dependencies with arrows

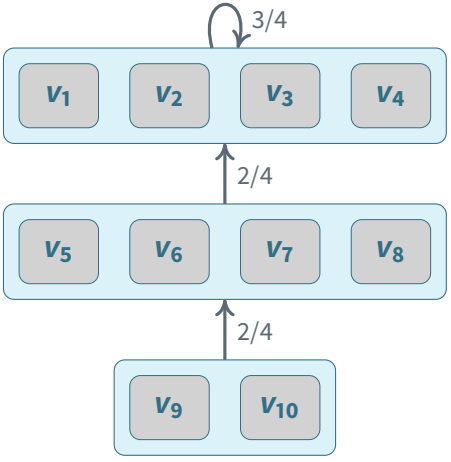
v_2, v_3, v_4 **is a quorum—contains a slice of each member**

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 slices in action



Top tier: slice is three out of $\{v_1, v_2, v_3, v_4\}$ (including self)

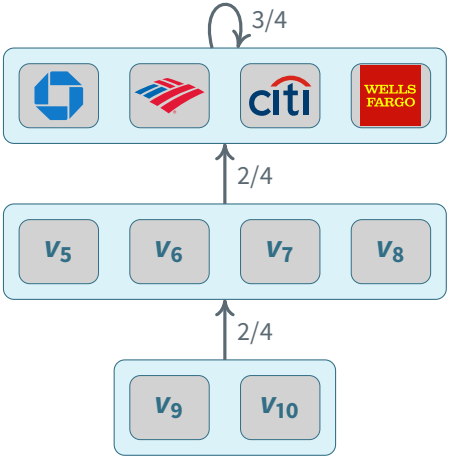
Middle tier: slice is self + any two top tier nodes

Leaf tier: slice is self + any two middle tier nodes

Like the Internet, no central authority appoints top tier

- But market can decide on *de facto* tier one organizations
- Don't even require exact agreement on who is a top tier node

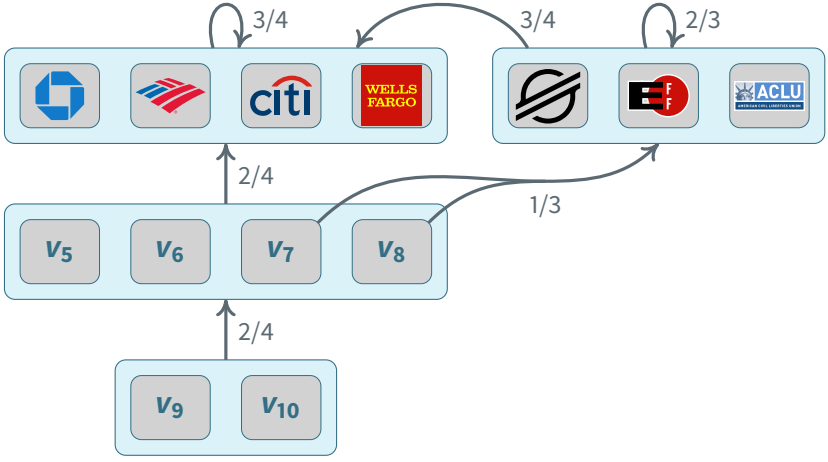
Quorum slices in action



Like the Internet, no central authority appoints top tier

- But market can decide on *de facto* tier one organizations
- Don’t even require exact agreement on who is a top tier node

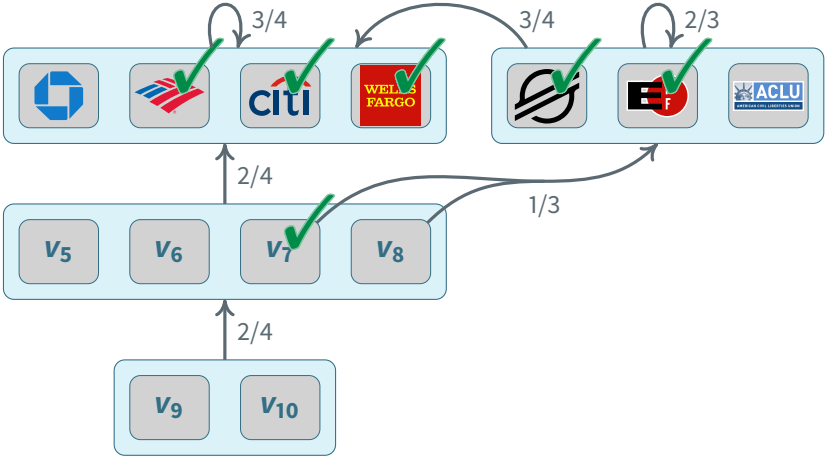
Quorum slices in action



Like the Internet, no central authority appoints top tier

- But market can decide on *de facto* tier one organizations
- Don’t even require exact agreement on who is a top tier node

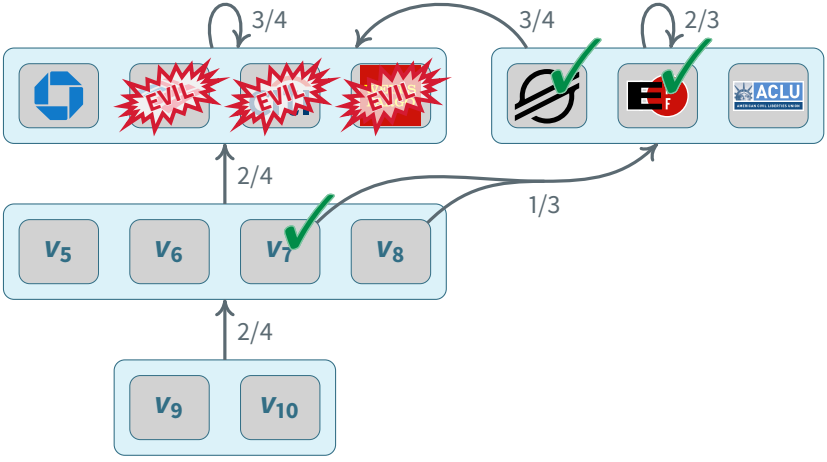
Quorum slices in action



Example: Citibank pays $1,000,000,000 to v_7

- Colludes to reverse transaction and double-spend same money to v_8
- Stellar & EFF won't revert, so ACLU cannot accept and v_8 won't either

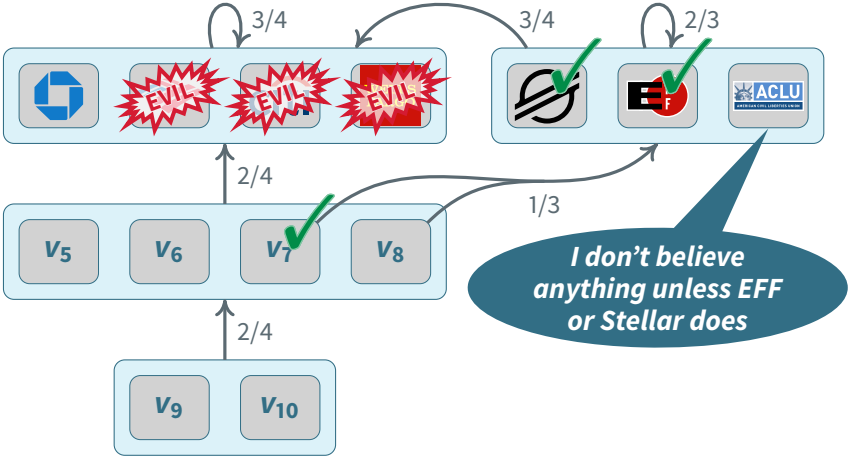
Quorum slices in action



Example: Citibank pays $1,000,000,000 to v_7

- Colludes to reverse transaction and double-spend same money to v_8
- Stellar & EFF won't revert, so ACLU cannot accept and v_8 won't either

Quorum slices in action

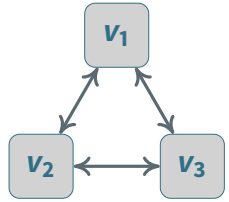


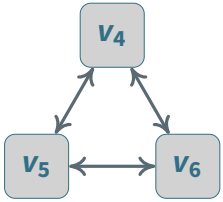
Example: Citibank pays $1,000,000,000 to v_7

- Colludes to reverse transaction and double-spend same money to v_8
- Stellar & EFF won't revert, so ACLU cannot accept and v_8 won't either

What is necessary to guarantee safety?

$\text{slices}(v_1) =$
$\text{slices}(v_2) =$
$\text{slices}(v_3) =$
$\{\{v_1, v_2, v_3\}\}$





$\text{slices}(v_4) =$
$\text{slices}(v_5) =$
$\text{slices}(v_6) =$
$\{\{v_4, v_5, v_6\}\}$

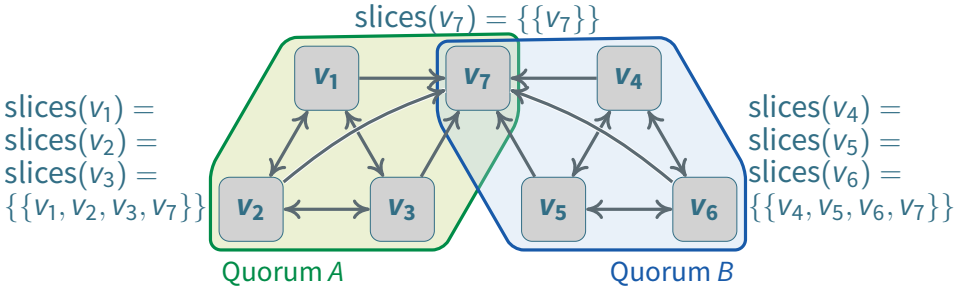
Suppose there are two entirely disjoint quorums

- Each can make progress with no communication from the other
- No way to guarantee the two sides will remain in agreement—system is unsafe

Like traditional consensus, safety requires *quorum intersection*

- The property that any two quorums share at least one non-faulty node.

What about malicious failures?



No protocol can guarantee agreement if quorum intersection is malicious

Is it a problem if faulty nodes cause disagreement?

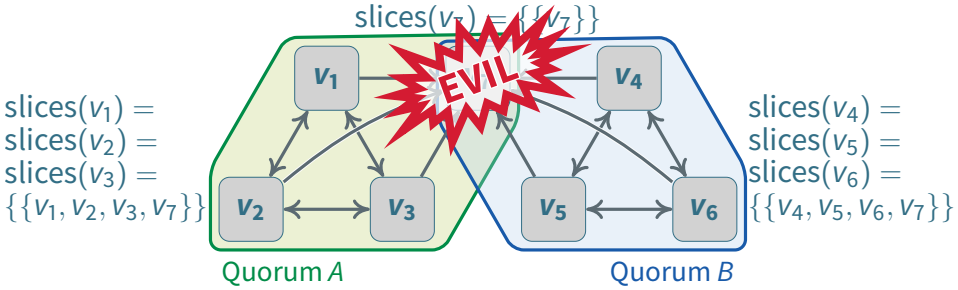
- Might be fine—e.g., Quorum B might be a Sybil attack

With open membership, safety is a property of *pairs of nodes*, not whole system

Two nodes *intertwined* if any two of their quorums share a non-faulty node

- Necessary and sufficient condition to guarantee agreement

What about malicious failures?



No protocol can guarantee agreement if quorum intersection is malicious

Is it a problem if faulty nodes cause disagreement?

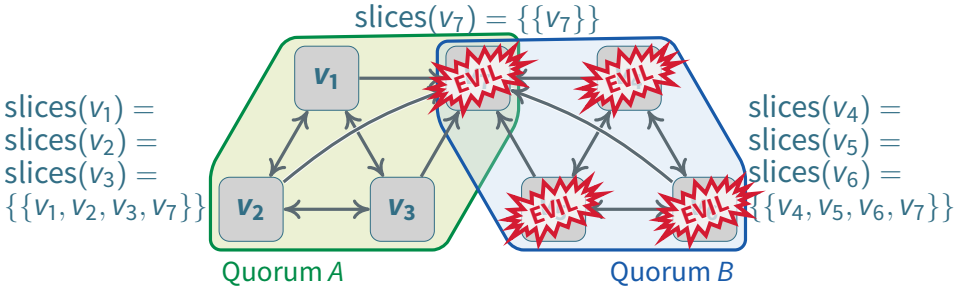
- Might be fine—e.g., Quorum B might be a Sybil attack

With open membership, safety is a property of *pairs of nodes*, not whole system

Two nodes *intertwined* if any two of their quorums share a non-faulty node

- Necessary and sufficient condition to guarantee agreement

What about malicious failures?



No protocol can guarantee agreement if quorum intersection is malicious

Is it a problem if faulty nodes cause disagreement?

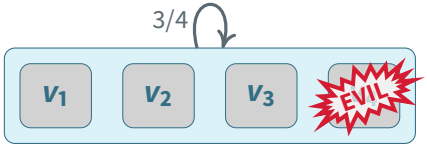
- Might be fine—e.g., Quorum B might be a Sybil attack

With open membership, safety is a property of *pairs of nodes*, not whole system

Two nodes *intertwined* if any two of their quorums share a non-faulty node

- Necessary and sufficient condition to guarantee agreement

Liveness



$$\text{slices}(v_i) = \{\{v_1, v_2, v_3\}, \{v_1, v_2, v_4\}, \{v_1, v_3, v_4\}\}$$

Necessary requirements to ensure protocol can finish (liveness):

- Need at least one uniformly non-faulty and intertwined quorum
- Possibly more—minimum practical requirement is open research problem

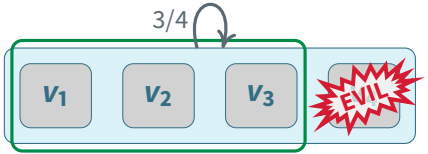
Sufficient property for liveness: be member of an *intact* set:

- Uniformly non-faulty quorum that remains intertwined even if all non-members are faulty

A set S is *v-blocking* if it intersects every slice of node v

- If a v-blocking set is uniformly faulty, v cannot be a member of an intact set
- Converse untrue—just because v doesn't have a faulty blocking set doesn't mean v intact
- But concept useful because v can locally check whether a set is v-blocking

Liveness



$$\text{slices}(v_i) = \{\{v_1, v_2, v_3\}, \{v_1, v_2, v_4\}, \{v_1, v_3, v_4\}\}$$

Necessary requirements to ensure protocol can finish (liveness):

- Need at least one uniformly non-faulty and intertwined quorum
- Possibly more—minimum practical requirement is open research problem

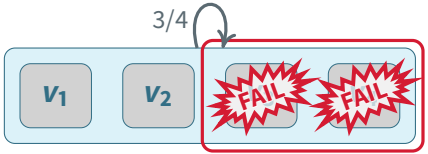
Sufficient property for liveness: be member of an *intact* set:

- Uniformly non-faulty quorum that remains intertwined even if all non-members are faulty

A set S is *v-blocking* if it intersects every slice of node v

- If a v-blocking set is uniformly faulty, v cannot be a member of an intact set
- Converse untrue—just because v doesn't have a faulty blocking set doesn't mean v intact
- But concept useful because v can locally check whether a set is v-blocking

Liveness



$$\text{slices}(v_i) = \{\{v_1, v_2, v_3\}, \{v_1, v_2, v_4\}, \{v_1, v_3, v_4\}\}$$

Necessary requirements to ensure protocol can finish (liveness):

- Need at least one uniformly non-faulty and intertwined quorum
- Possibly more—minimum practical requirement is open research problem

Sufficient property for liveness: be member of an *intact* set:

- Uniformly non-faulty quorum that remains intertwined even if all non-members are faulty

A set S is v-*blocking* if it intersects every slice of node v

- If a v-blocking set is uniformly faulty, v cannot be a member of an intact set
- Converse untrue—just because v doesn't have a faulty blocking set doesn't mean v intact
- But concept useful because v can locally check whether a set is v-blocking

Key concepts in SCP

The cascade theorem

- If a quorum includes an intact node, transitively blocks intact set
- If blocking sets trigger protocol steps, state changes cascade to all intact nodes

Federated voting

- “Bad” consensus protocol that can get stuck (can be indistinguishable from slow)
- When it succeeds, all intact nodes guaranteed to terminate and agree

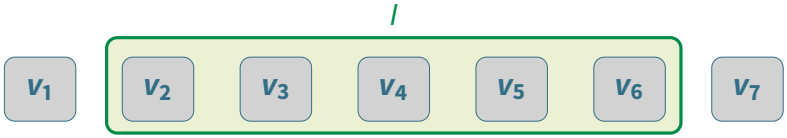
Balloting

- Uses federated voting to make a consensus protocol that won’t get stuck

Nomination

- How to pick the initial input for balloting

Cascade theorem



Let $I = \{v_2, \ldots v_6\}$ be an intact set

Let $Q = \{v_1, \ldots v_4\}$ be a quorum intersecting I, let $S = Q$

- Intuitively, S will be a set of nodes reaching some protocol state

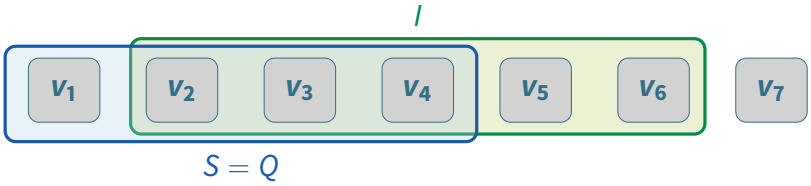
If S is not v-blocking for any $v \in I \setminus S = \{v_5, v_6\}$, then I not intact (contradiction)

- Say all nodes outside I faulty (ignore their slices), means I not intertwined

Suppose S is v_5-blocking, then update $S \leftarrow S \cup \{v_5\}$

Repeat the process until eventually $S \supseteq I$ (S has cascaded throughout I)

Cascade theorem



Let $I = \{v_2, \ldots v_6\}$ be an intact set

Let $Q = \{v_1, \ldots v_4\}$ be a quorum intersecting I, let $S = Q$

- Intuitively, S will be a set of nodes reaching some protocol state

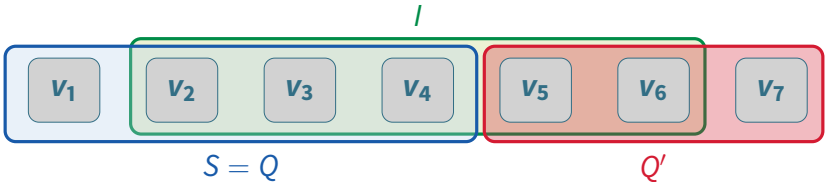
If S is not v-blocking for any $v \in I \setminus S = \{v_5, v_6\}$, then I not intact (contradiction)

- Say all nodes outside I faulty (ignore their slices), means I not intertwined

Suppose S is v_5-blocking, then update $S \leftarrow S \cup \{v_5\}$

Repeat the process until eventually $S \supseteq I$ (S has cascaded throughout I)

Cascade theorem



Let $I = \{v_2, \ldots v_6\}$ be an intact set

Let $Q = \{v_1, \ldots v_4\}$ be a quorum intersecting I, let $S = Q$

- Intuitively, S will be a set of nodes reaching some protocol state

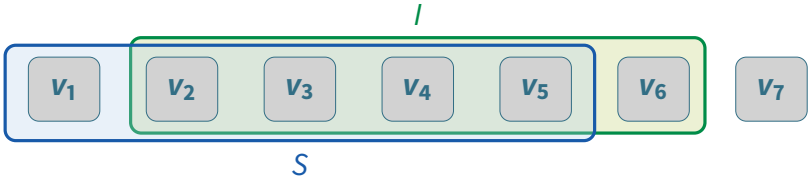
If S is not v-blocking for any $v \in I \setminus S = \{v_5, v_6\}$, then I not intact (contradiction)

- Say all nodes outside I faulty (ignore their slices), means I not intertwined

Suppose S is v_5-blocking, then update $S \leftarrow S \cup \{v_5\}$

Repeat the process until eventually $S \supseteq I$ (S has cascaded throughout I)

Cascade theorem



Let $I = \{v_2, \ldots v_6\}$ be an intact set

Let $Q = \{v_1, \ldots v_4\}$ be a quorum intersecting I, let $S = Q$

- Intuitively, S will be a set of nodes reaching some protocol state

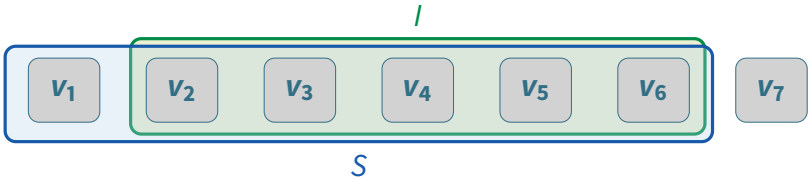
If S is not v-blocking for any $v \in I \setminus S = \{v_5, v_6\}$, then I not intact (contradiction)

- Say all nodes outside I faulty (ignore their slices), means I not intertwined

Suppose S is v_5-blocking, then update $S \leftarrow S \cup \{v_5\}$

Repeat the process until eventually $S \supseteq I$ (S has cascaded throughout I)

Cascade theorem



Let $I = \{v_2, \ldots v_6\}$ be an intact set

Let $Q = \{v_1, \ldots v_4\}$ be a quorum intersecting I, let $S = Q$

- Intuitively, S will be a set of nodes reaching some protocol state

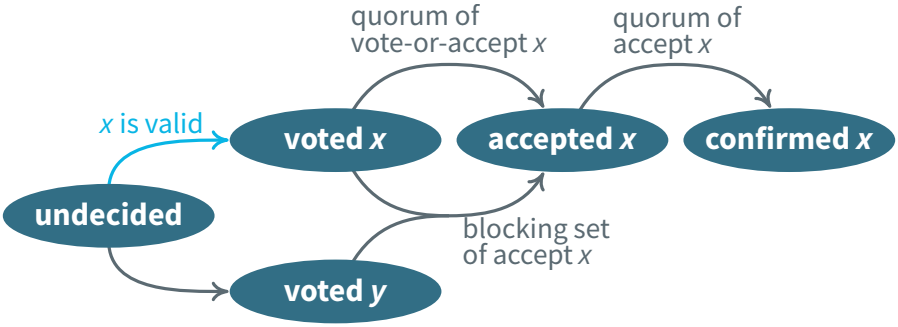
If S is not v-blocking for any $v \in I \setminus S = \{v_5, v_6\}$, then I not intact (contradiction)

- Say all nodes outside I faulty (ignore their slices), means I not intertwined

Suppose S is v_5-blocking, then update $S \leftarrow S \cup \{v_5\}$

Repeat the process until eventually $S \supseteq I$ (S has cascaded throughout I)

Federated voting



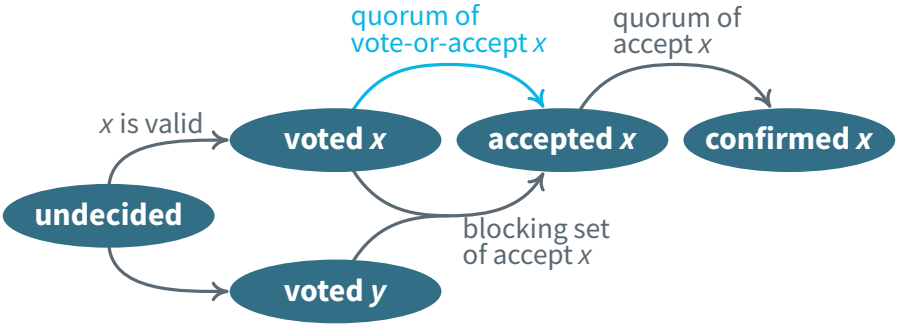
***vote* for any valid statement *x* that doesn't contradict past votes/accepts**

***accept* *x* when *x* does not contradict past accepts and:**

- You are in a quorum where each member votes for or accepts *x*, or
- Every node in a blocking set accepts *x* (in which case can forget past votes contradicting *x*)

***confirm* *x* when in quorum that accepts it (can externalize after this)**

Federated voting



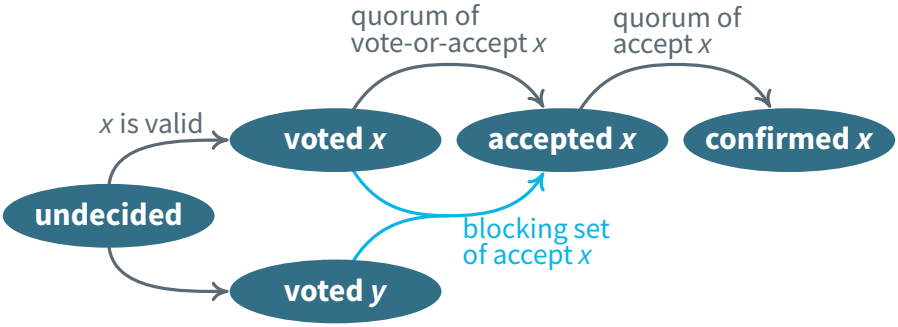
***vote* for any valid statement *x* that doesn't contradict past votes/accepts**

***accept* *x* when *x* does not contradict past accepts and:**

- You are in a quorum where each member votes for or accepts *x*, or
- Every node in a blocking set accepts *x* (in which case can forget past votes contradicting *x*)

***confirm* *x* when in quorum that accepts it (can externalize after this)**

Federated voting



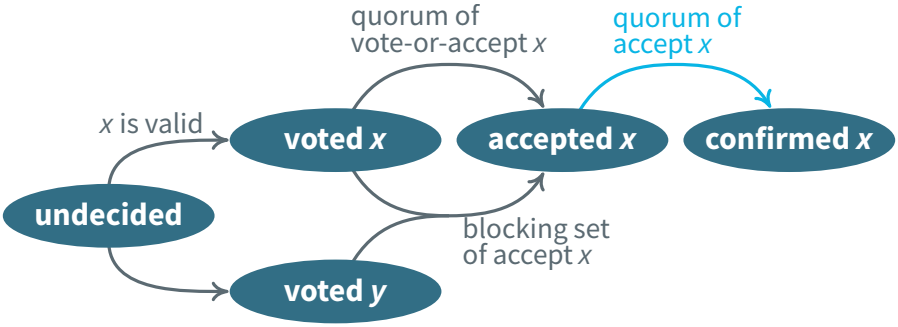
***vote* for any valid statement *x* that doesn't contradict past votes/accepts**

***accept* *x* when *x* does not contradict past accepts and:**

- You are in a quorum where each member votes for or accepts *x*, or
- Every node in a blocking set accepts *x* (in which case can forget past votes contradicting *x*)

***confirm* *x* when in quorum that accepts it (can externalize after this)**

Federated voting



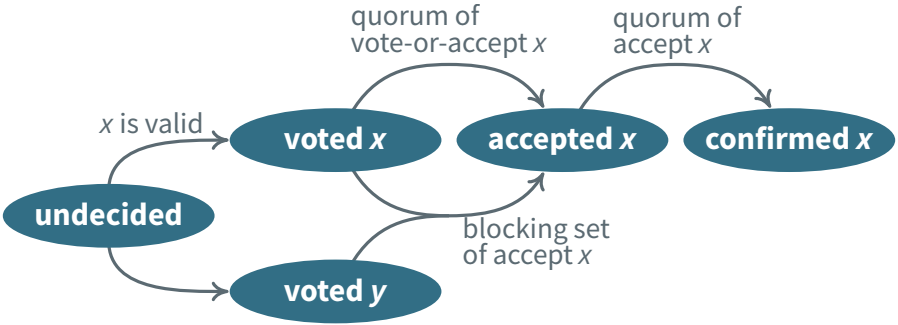
***vote* for any valid statement *x* that doesn't contradict past votes/accepts**

***accept* *x* when *x* does not contradict past accepts and:**

- You are in a quorum where each member votes for or accepts *x*, or
- Every node in a blocking set accepts *x* (in which case can forget past votes contradicting *x*)

***confirm* *x* when in quorum that accepts it (can externalize after this)**

Federated voting properties



Intertwined nodes cannot confirm contradictory statements

- Means confirmed statement enjoy “optimal safety”

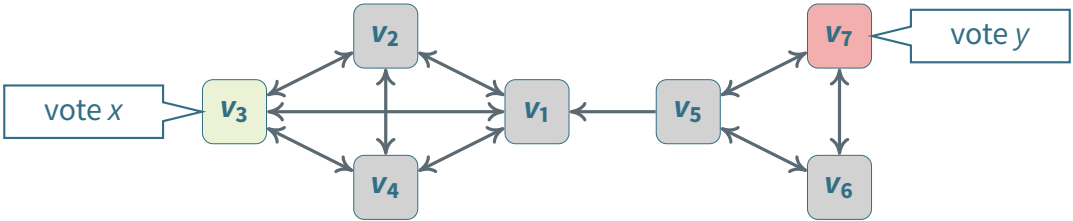
Confirmed statements cascade from one intact node to maximum intact set

- Means you can assume confirmed statements without risking liveness

If a set of intact nodes all vote for same statement, all will eventually confirm

- But if they don’t, the vote can get permanently stuck

Federated voting example



Vote for a valid statement

- E.g., $x =$ "Choose transaction set T for ledger n in ballot b"

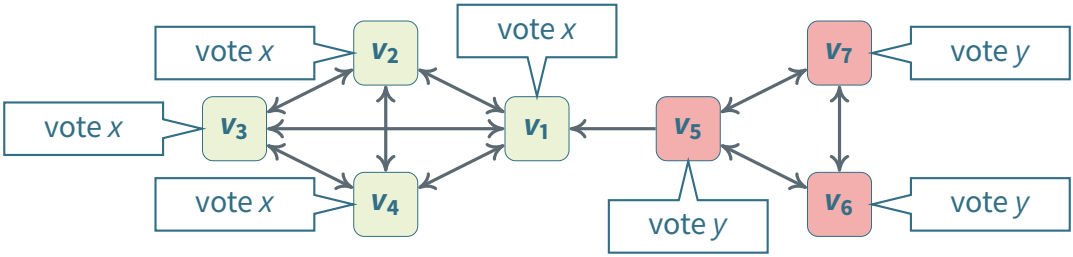
Accept if you are in a quorum that unanimously votes for or accepts x

Also accept if a blocking set unanimously accepts

- Note v_5 won't act on x before confirming it (so malicious v_1 can't make v_5 disagree with v_7)

Confirm statement if you are in a quorum that unanimously accepts

Federated voting example



Vote for a valid statement

- E.g., $x =$ "Choose transaction set T for ledger n in ballot b"

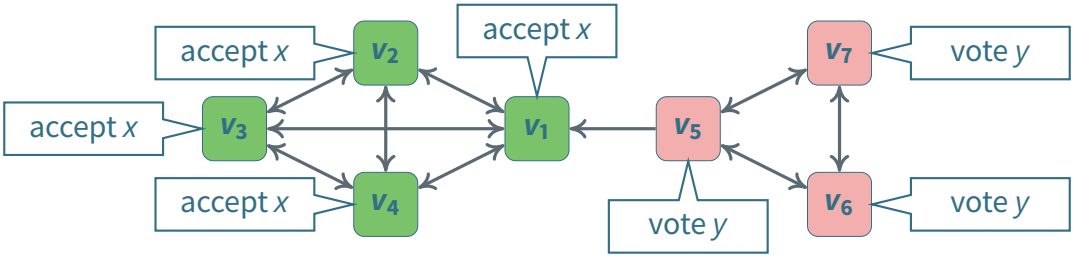
Accept if you are in a quorum that unanimously votes for or accepts x

Also accept if a blocking set unanimously accepts

- Note v_5 won't act on x before confirming it (so malicious v_1 can't make v_5 disagree with v_7)

Confirm statement if you are in a quorum that unanimously accepts

Federated voting example



Vote for a valid statement

- E.g., $x =$ "Choose transaction set T for ledger n in ballot b"

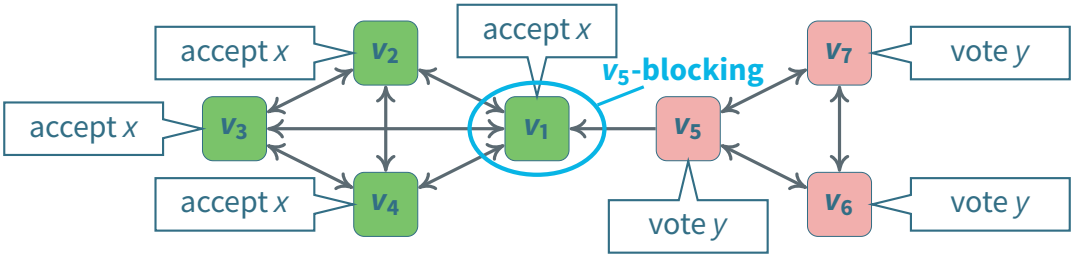
Accept if you are in a quorum that unanimously votes for or accepts x

Also accept if a blocking set unanimously accepts

- Note v_5 won't act on x before confirming it (so malicious v_1 can't make v_5 disagree with v_7)

Confirm statement if you are in a quorum that unanimously accepts

Federated voting example



Vote for a valid statement

- E.g., $x =$ "Choose transaction set T for ledger n in ballot b"

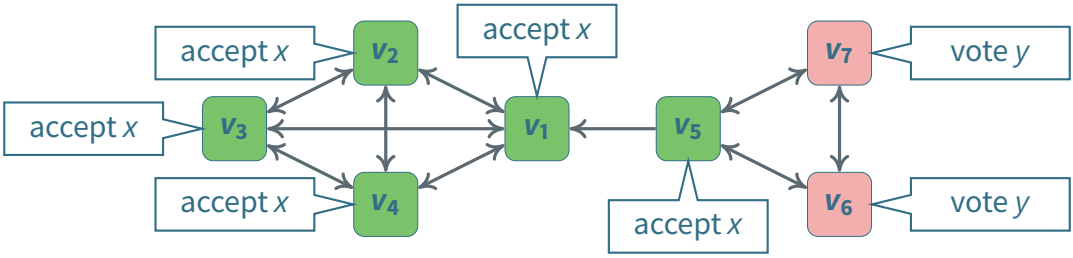
Accept if you are in a quorum that unanimously votes for or accepts x

Also accept if a blocking set unanimously accepts

- Note v_5 won't act on x before confirming it (so malicious v_1 can't make v_5 disagree with v_7)

Confirm statement if you are in a quorum that unanimously accepts

Federated voting example



Vote for a valid statement

- E.g., $x =$ "Choose transaction set T for ledger n in ballot b"

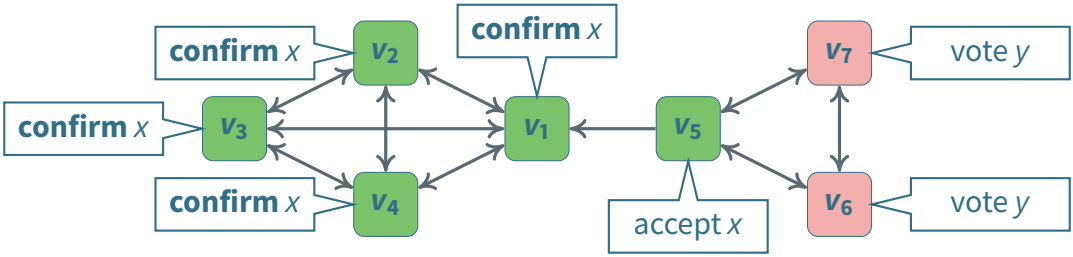
Accept if you are in a quorum that unanimously votes for or accepts x

Also accept if a blocking set unanimously accepts

- Note v_5 won't act on x before confirming it (so malicious v_1 can't make v_5 disagree with v_7)

Confirm statement if you are in a quorum that unanimously accepts

Federated voting example



Vote for a valid statement

- E.g., $x =$ "Choose transaction set T for ledger n in ballot b"

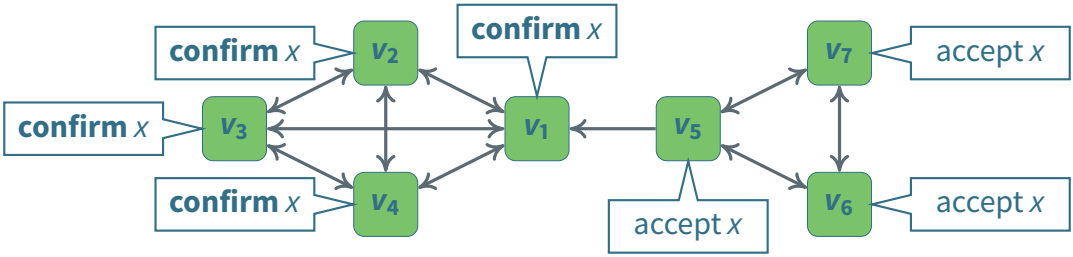
Accept if you are in a quorum that unanimously votes for or accepts x

Also accept if a blocking set unanimously accepts

- Note v_5 won't act on x before confirming it (so malicious v_1 can't make v_5 disagree with v_7)

Confirm statement if you are in a quorum that unanimously accepts

Federated voting example



Vote for a valid statement

- E.g., $x =$ "Choose transaction set T for ledger n in ballot b"

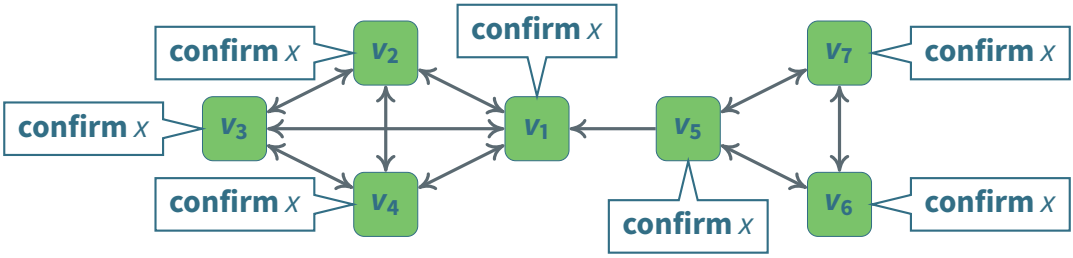
Accept if you are in a quorum that unanimously votes for or accepts x

Also accept if a blocking set unanimously accepts

- Note v_5 won't act on x before confirming it (so malicious v_1 can't make v_5 disagree with v_7)

Confirm statement if you are in a quorum that unanimously accepts

Federated voting example



Vote for a valid statement

- E.g., $x =$ "Choose transaction set T for ledger n in ballot b"

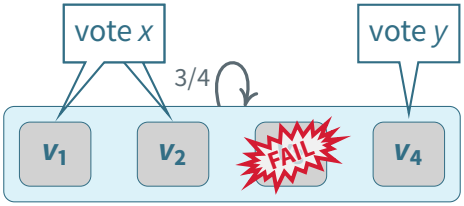
Accept if you are in a quorum that unanimously votes for or accepts x

Also accept if a blocking set unanimously accepts

- Note v_5 won't act on x before confirming it (so malicious v_1 can't make v_5 disagree with v_7)

Confirm statement if you are in a quorum that unanimously accepts

Balloting overview



Problem: federated voting can get stuck

- Can't necessarily detect—Is node v_3 slow, or did it fail?

Idea: proceed through a series of numbered *ballots* (c.f. Paxos)

- Try to decide in ballot n, but after timeout try again in ballot $n + 1$
- New ballot lets nodes vote for new values—e.g., v_4 might want to vote for x

Challenge: must not decide different values in different ballots!

- E.g., what if v_3 externalized x before failing?

Key invariant: All stuck and decided ballots must chose same value

Statements in balloting votes

PREPARE $\langle n, x \rangle$	states that no value other than x was or will ever be decided in any ballot $\leq n$
COMMIT $\langle n, x \rangle$	states x is decided in ballot n

Begin ballot n by attempting federated vote on PREPARE $\langle n, x \rangle$

- Did a previous ballot confirm PREPARE? Set x to value from highest such ballot
- Otherwise, pick x from nomination protocol (in a few slides)

A node must not vote for COMMIT $\langle n, x \rangle$ unless it has confirmed PREPARE $\langle n, x \rangle$

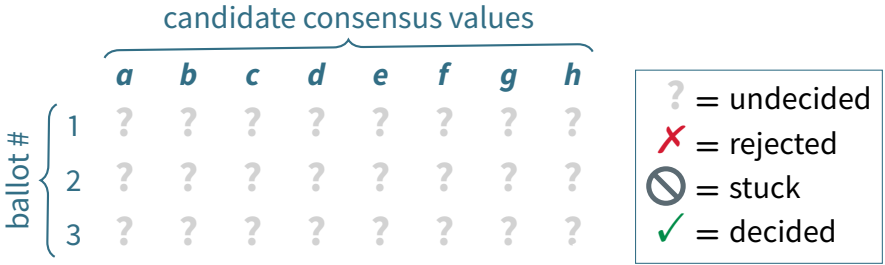
- Guarantees different ballots cannot decide different values

Externalize a value x after confirming COMMIT $\langle n, x \rangle$

Synchronize ballots across nodes using the cascade theorem

- Arm timer for ballot n only when in a quorum on ballot $\geq n$
- Immediately jump to higher ballot if a blocking set has higher ballot number

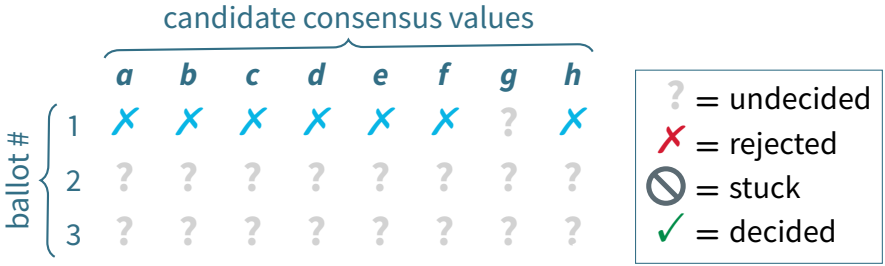
Balloting example



0. **Initially, all ballots are undecided (since no votes)**
1. **Confirm PREPARE $\langle 1, g \rangle$ and vote for COMMIT $\langle 1, g \rangle$**
2. **Lose vote on COMMIT $\langle 1, g \rangle$; confirm PREPARE $\langle 2, f \rangle$**
3. **Ballot 2 times out; confirm PREPARE $\langle 3, f \rangle$ and vote for COMMIT $\langle 3, f \rangle$**
4. **Confirm COMMIT $\langle 3, f \rangle$ and externalize f**
   - At this point nobody cares that COMMIT $\langle 2, f \rangle$ is stuck

Preserves invariant: all decided & stuck ballots have same value

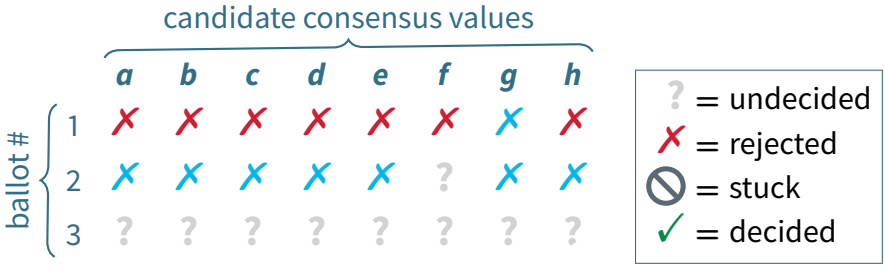
Balloting example



0. **Initially, all ballots are undecided (since no votes)**
1. **Confirm PREPARE $\langle 1, g \rangle$ and vote for COMMIT $\langle 1, g \rangle$**
2. **Lose vote on COMMIT $\langle 1, g \rangle$; confirm PREPARE $\langle 2, f \rangle$**
3. **Ballot 2 times out; confirm PREPARE $\langle 3, f \rangle$ and vote for COMMIT $\langle 3, f \rangle$**
4. **Confirm COMMIT $\langle 3, f \rangle$ and externalize f**
   - At this point nobody cares that COMMIT $\langle 2, f \rangle$ is stuck

Preserves invariant: all decided & stuck ballots have same value

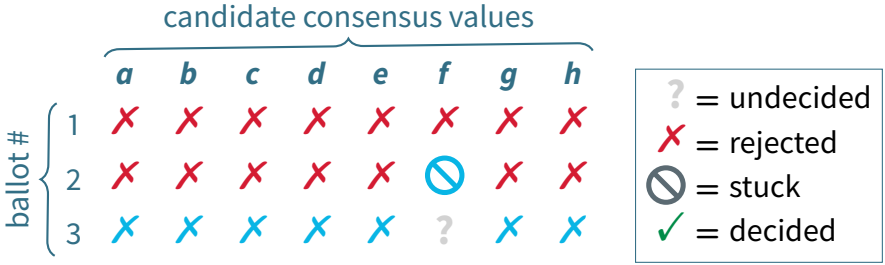
Balloting example



0. **Initially, all ballots are undecided (since no votes)**
1. **Confirm PREPARE $\langle 1, g \rangle$ and vote for COMMIT $\langle 1, g \rangle$**
2. **Lose vote on COMMIT $\langle 1, g \rangle$; confirm PREPARE $\langle 2, f \rangle$**
3. **Ballot 2 times out; confirm PREPARE $\langle 3, f \rangle$ and vote for COMMIT $\langle 3, f \rangle$**
4. **Confirm COMMIT $\langle 3, f \rangle$ and externalize f**
   - At this point nobody cares that COMMIT $\langle 2, f \rangle$ is stuck

Preserves invariant: all decided & stuck ballots have same value

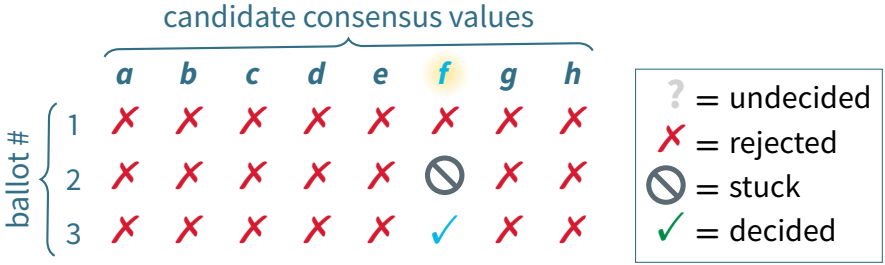
Balloting example



0. **Initially, all ballots are undecided (since no votes)**
1. **Confirm PREPARE $\langle 1, g \rangle$ and vote for COMMIT $\langle 1, g \rangle$**
2. **Lose vote on COMMIT $\langle 1, g \rangle$; confirm PREPARE $\langle 2, f \rangle$**
3. **Ballot 2 times out; confirm PREPARE $\langle 3, f \rangle$ and vote for COMMIT $\langle 3, f \rangle$**
4. **Confirm COMMIT $\langle 3, f \rangle$ and externalize f**
   - At this point nobody cares that COMMIT $\langle 2, f \rangle$ is stuck

Preserves invariant: all decided & stuck ballots have same value

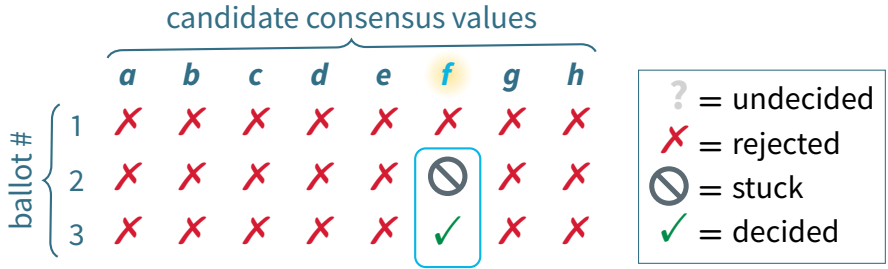
Balloting example



0. **Initially, all ballots are undecided (since no votes)**
1. **Confirm PREPARE $\langle 1, g \rangle$ and vote for COMMIT $\langle 1, g \rangle$**
2. **Lose vote on COMMIT $\langle 1, g \rangle$; confirm PREPARE $\langle 2, f \rangle$**
3. **Ballot 2 times out; confirm PREPARE $\langle 3, f \rangle$ and vote for COMMIT $\langle 3, f \rangle$**
4. **Confirm COMMIT $\langle 3, f \rangle$ and externalize f**
   - At this point nobody cares that COMMIT $\langle 2, f \rangle$ is stuck

Preserves invariant: all decided & stuck ballots have same value

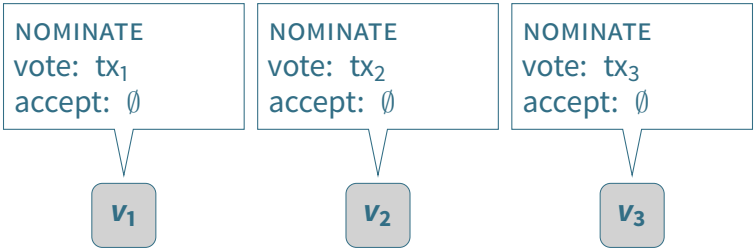
Balloting example



0. **Initially, all ballots are undecided (since no votes)**
1. **Confirm PREPARE $\langle 1, g \rangle$ and vote for COMMIT $\langle 1, g \rangle$**
2. **Lose vote on COMMIT $\langle 1, g \rangle$; confirm PREPARE $\langle 2, f \rangle$**
3. **Ballot 2 times out; confirm PREPARE $\langle 3, f \rangle$ and vote for COMMIT $\langle 3, f \rangle$**
4. **Confirm COMMIT $\langle 3, f \rangle$ and externalize f**
   - At this point nobody cares that COMMIT $\langle 2, f \rangle$ is stuck

Preserves invariant: all decided & stuck ballots have same value

Strawman nomination



Goal: Choose a value to prepare for next if no confirmed PREPARE

Every node votes for its own proposed value

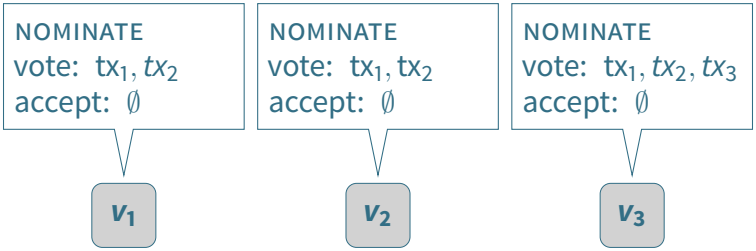
Every node also votes for values it learns from others

Eventually, nodes accept and confirm nominated values

- Stop voting for new values once any value confirmed
  e.g., v_1 and v_2 will never vote for v_3

Deterministically combine all confirmed nominated values

Strawman nomination



Goal: Choose a value to prepare for next if no confirmed PREPARE

Every node votes for its own proposed value

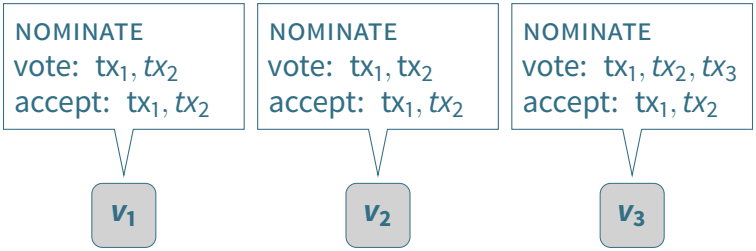
Every node also votes for values it learns from others

Eventually, nodes accept and confirm nominated values

- Stop voting for new values once any value confirmed
  e.g., v_1 and v_2 will never vote for v_3

Deterministically combine all confirmed nominated values

Strawman nomination



Goal: Choose a value to prepare for next if no confirmed PREPARE

Every node votes for its own proposed value

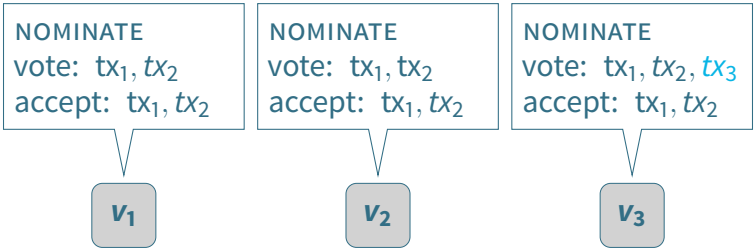
Every node also votes for values it learns from others

Eventually, nodes accept and confirm nominated values

- Stop voting for new values once any value confirmed
  e.g., v_1 and v_2 will never vote for v_3

Deterministically combine all confirmed nominated values

Strawman nomination



Goal: Choose a value to prepare for next if no confirmed PREPARE

Every node votes for its own proposed value

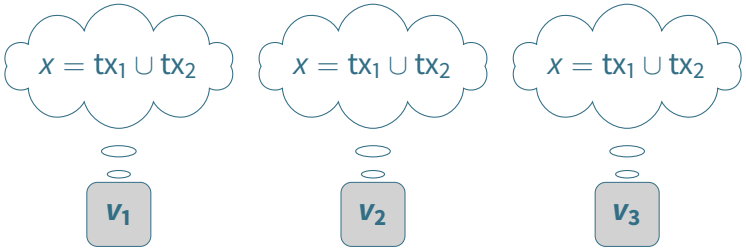
Every node also votes for values it learns from others

Eventually, nodes accept and confirm nominated values

- Stop voting for new values once any value confirmed
  e.g., v_1 and v_2 will never vote for v_3

Deterministically combine all confirmed nominated values

Strawman nomination



Goal: Choose a value to prepare for next if no confirmed PREPARE

Every node votes for its own proposed value

Every node also votes for values it learns from others

Eventually, nodes accept and confirm nominated values

- Stop voting for new values once any value confirmed
  e.g., v_1 and v_2 will never vote for v_3

Deterministically combine all confirmed nominated values

Properties of nomination strawman

+ At least one value will get nominated (assuming intact quorum)

+ Once a single intact node confirms a value nominated, whole intact set will

- Direct consequence of how confirmed statements cascade in federated voting

+ A bounded number of values can get nominated

- Need votes from intact nodes to accept (then confirm) nominated values
- Well-behaved nodes can cast only a bounded number of votes before confirming first value

+ Nomination is guaranteed to converge eventually

- Attacker can perturb bounded number of times—each “consumes” an as-yet-unconfirmed value nominated by an intact node

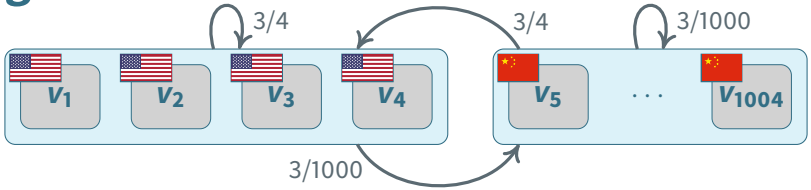
– Never know when nomination has converged—have to guess

- Inevitable given fundamental impossibility result [FLP]

– Lots of values floating around wastes bandwidth, computation

- Can we use some sort of leader selection to reduce costs?

Reducing # nominated values



Choose leader pseudorandomly by highest $H(\text{PubKey}\|\text{round})$**?**

- Works for Algorand because coins quantify clout
- Here risks censorship from organizations/countries with more nodes

Select leaders based on local slice weight & hashes:

$$\begin{aligned} \text{weight}(v) &= \text{fraction of local quorum slices containing } v \\ \text{neighbors}(\text{round}) &= \{\, v \mid H_1(\text{round}\|v) < h_{\max} \cdot \text{weight}(v) \,\} \\ \text{priority}(\text{round}, v) &= H_2(\text{round}\|v) \end{aligned}$$

- Round leader is neighbor with highest priority
- After n rounds, echo nomination votes of leaders of round $\leq n$
- Tends to converge, always does if identical quorum slices

SCP vs. traditional BFT consensus

With open membership, SCP cannot round-robin among leaders

- Slice-based leader selection not guaranteed to produce unique leader

In SCP, a quorum is only meaningful to its members

- Attackers can create unanimously bad quorum, good nodes won't recognize it
- Collection of signatures from a quorum doesn't prove anything to non-members

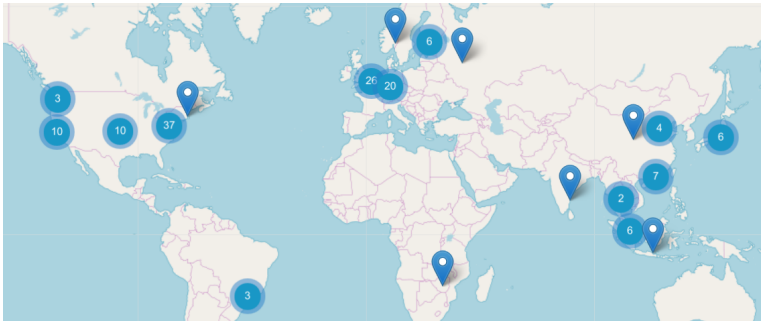
In SCP, can't reason backwards about safety

- Common BFT technique: "$f + 1$ nodes said x is true, assume with no loss of safety"
- In Stellar, safety is pairwise; can lose more safety by assuming incorrect facts

SCP *can* reason backwards about liveness using locally-checkable blocking sets

- Idea: first solve consensus for intact nodes, then add confirmation vote
- Confirmation vote could get stuck for non-intact nodes—but already not live
- Adds one extra communication round compared to analogous BFT

Status



Production network has been running since September 2015

- Ledger closes every 5 seconds, currently allows 1,000 operations/ledger
- Presently 110 nodes, 61 validators, 17 “tier-one” nodes run by 5 organizations

Shows open-membership Byzantine agreement is viable

- Would have been hard to attract, e.g., IBM to closed system

30+ assets tracked on 3rd-party stellar.expert

Questions?

www.stellar.org