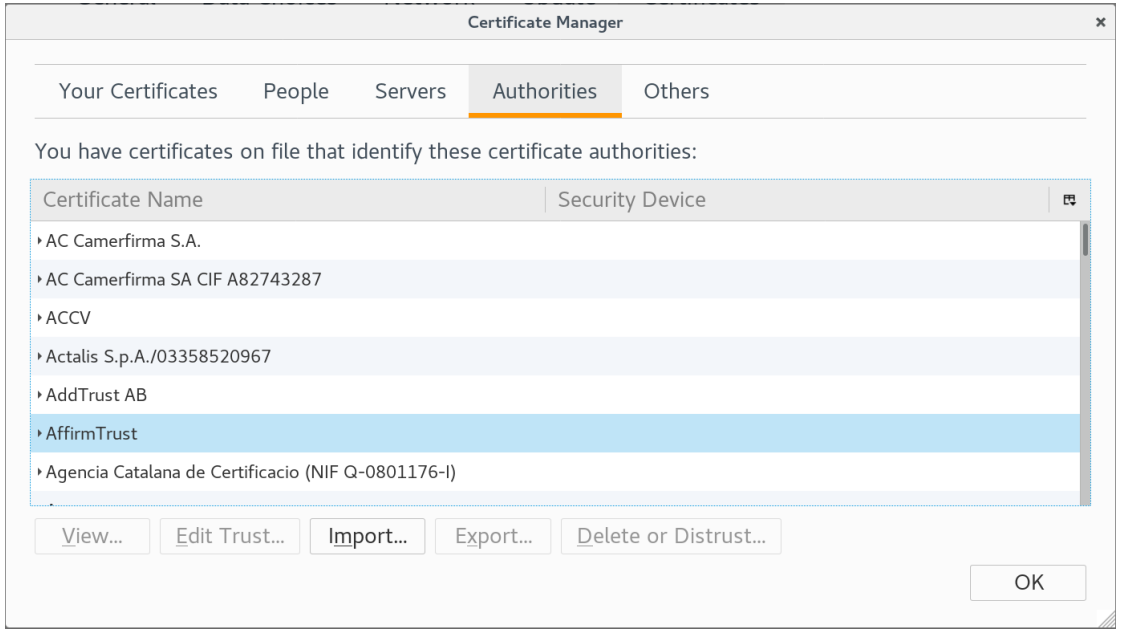
Internet-level consensus is practical

David Mazières

Stellar Development Foundation

Monday, April 24, 2017

Scaling trust



What if you want *all* CAs in the world to agree on something?

Problem: Who are "all" CAs?

- The 180+ CAs with 65+ owners in Mozilla's root CA list?
- The root CA set shipped in your OS distribution?
- Your organization's in-house CA that isn't globally recognized?
- Really want all of the above *for all people*

This is the *Internet-level consensus* (ILC) problem

Application 1: Global timestamp service

Certificate Transparency provides trusted logs alongside CAs

- Generalize CT logging to leverage logs for timestamping documents?

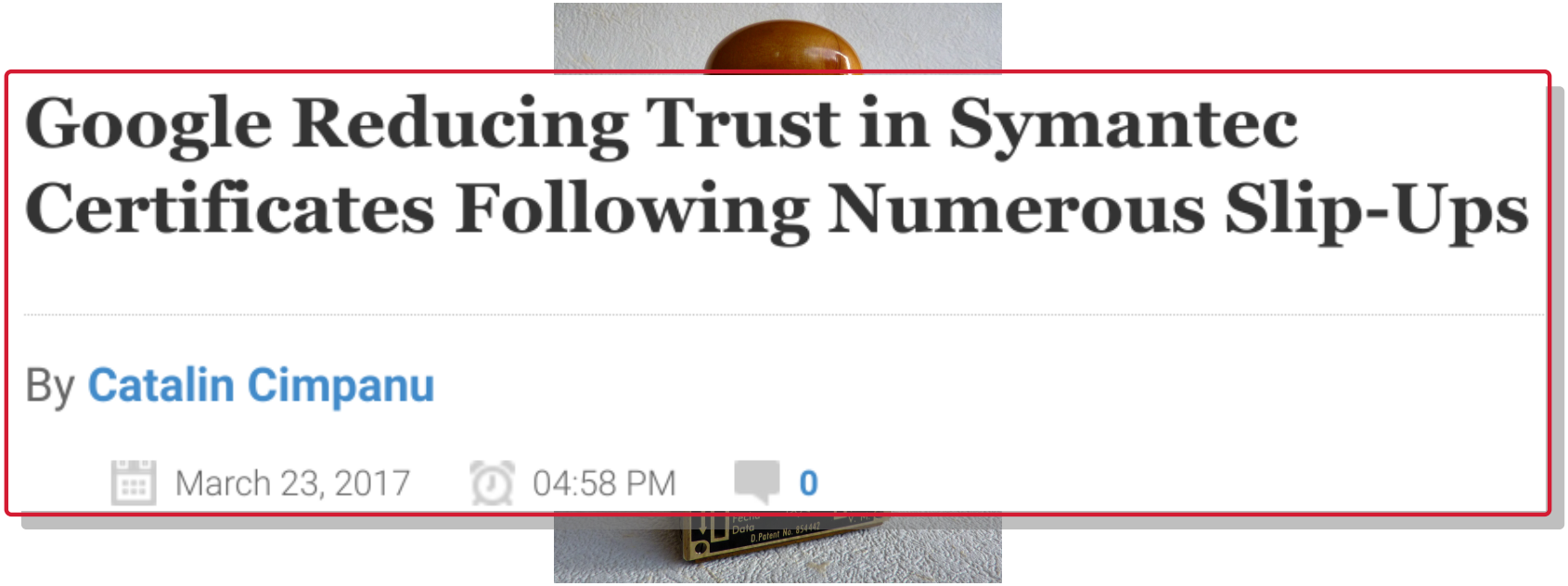
Problem: which log to use?

- Problem: different people trust different logs
- Don’t know in advance to whom you will need to prove timestamp

What if your log choice proves untrustworthy?

Using ILC for timestamps would avoid this problem

Application 1: Global timestamp service



Google Reducing Trust in Symantec Certificates Following Numerous Slip-Ups

By Catalin Cimpanu

March 23, 2017 04:58 PM 0

Certificate Transparency provides trusted logs alongside CAs

- Generalize CT logging to leverage logs for timestamping documents?

Problem: which log to use?

- Problem: different people trust different logs
- Don’t know in advance to whom you will need to prove timestamp

What if your log choice proves untrustworthy?

Using ILC for timestamps would avoid this problem

Application 2: Software transparency

In 2016, FBI ordered Apple to sign a compromised bootloader

- Apple appears to have refused, but how can we know for sure?

Make software updates visible through *software transparency*

- Devices refuse to install updates not in public log
- Log integrity secured through ILC

Binary transparency and secure package management can both benefit from ILC

Application 3: Internet payments

Suppose you want to send $1 to Nigeria over the Internet

May require multiple banks, currency trades

- Need to avoid getting stuck or allowing double spending
- ILC can secure atomic transactions across multiple organizations …even organizations with no prior relationship!

Technique currently in use by Stellar payment network

Application 3: Internet payments

Offeror	Bid	Ask
$bank_2$	0.93 EUR@$bank_3$	1.00 USD@$bank_1$
$bank_4$	300 NGN@$bank_4$	0.93 EUR@$bank_3$

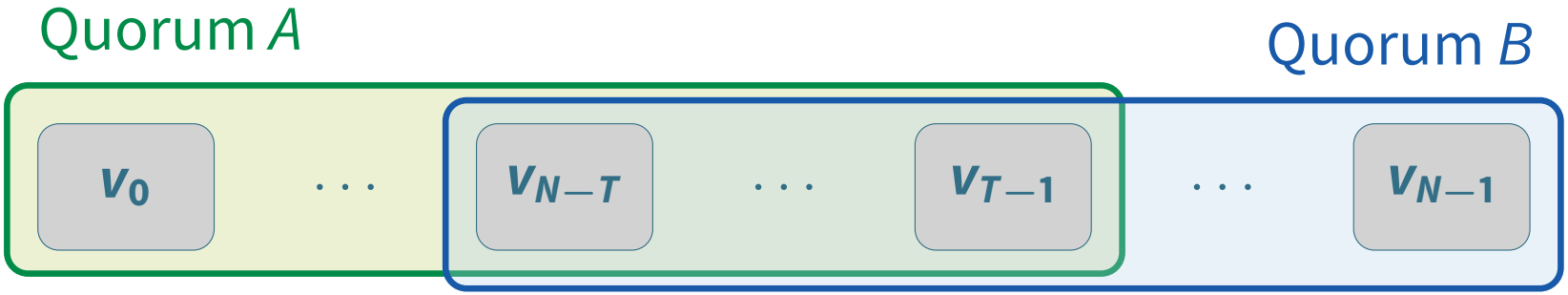
Suppose you want to send $1 to Nigeria over the Internet

May require multiple banks, currency trades

- Need to avoid getting stuck or allowing double spending
- ILC can secure atomic transactions across multiple organizations
  …even organizations with no prior relationship!

Technique currently in use by Stellar payment network

Background: Byzantine agreement



Practical solution to consensus among N nodes

- Pick T such that any T nodes constitute a *quorum*
- Tolerates faulty nodes that crash or send contradictory messages

***Safety* requires: # failures $\leq f_S = 2T - N - 1$**

- Hence, any two quorums share a *non-faulty* node, can't lose history

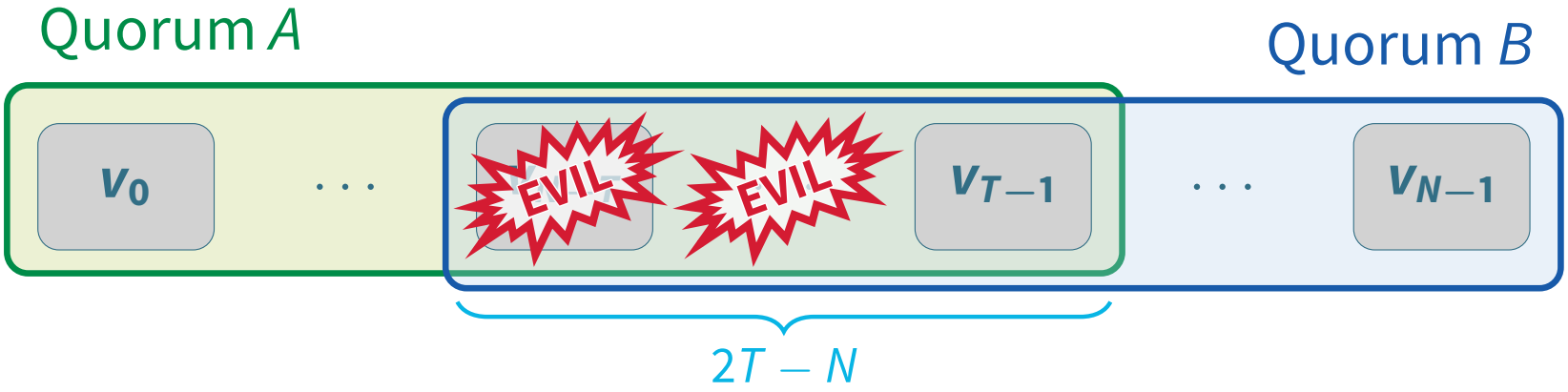
***Liveness*: # failures $\leq f_L = N - T$ (1 non-faulty quorum)**

Typically $N = 3f + 1$ and $T = 2f + 1$ to tolerate $f_S = f_L = f$ failures

Problem: at Internet level, can't enumerate N nodes

- Politically infeasible to decree set of globally trusted authorities

Background: Byzantine agreement



Practical solution to consensus among N nodes

- Pick T such that any T nodes constitute a *quorum*
- Tolerates faulty nodes that crash or send contradictory messages

***Safety* requires: # failures $\leq f_S = 2T - N - 1$**

- Hence, any two quorums share a *non-faulty* node, can't lose history

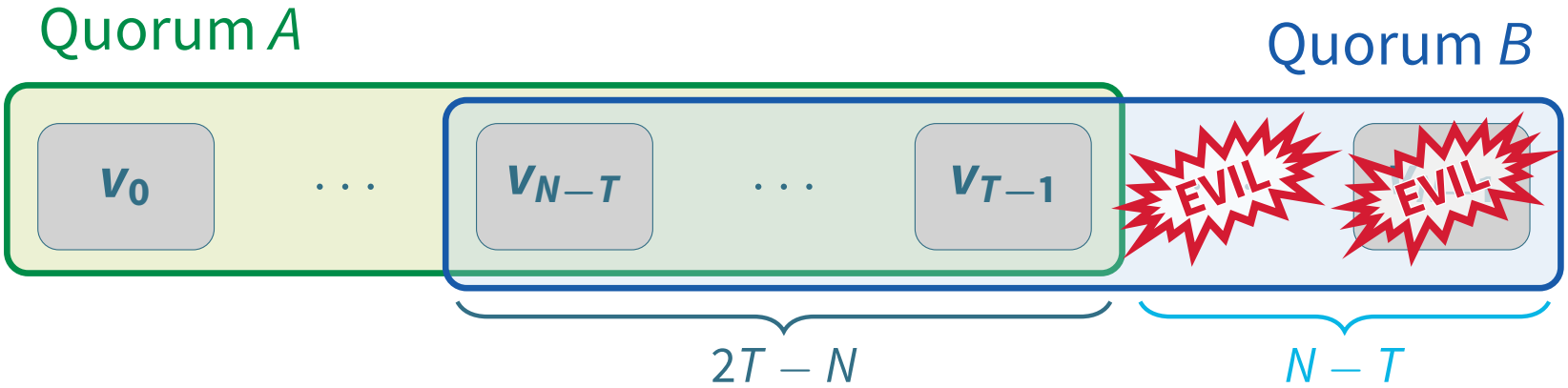
***Liveness*: # failures $\leq f_L = N - T$ (1 non-faulty quorum)**

Typically $N = 3f + 1$ and $T = 2f + 1$ to tolerate $f_S = f_L = f$ failures

Problem: at Internet level, can't enumerate N nodes

- Politically infeasible to decree set of globally trusted authorities

Background: Byzantine agreement



Practical solution to consensus among N nodes

- Pick T such that any T nodes constitute a *quorum*
- Tolerates faulty nodes that crash or send contradictory messages

***Safety* requires: # failures $\leq f_S = 2T - N - 1$**

- Hence, any two quorums share a *non-faulty* node, can't lose history

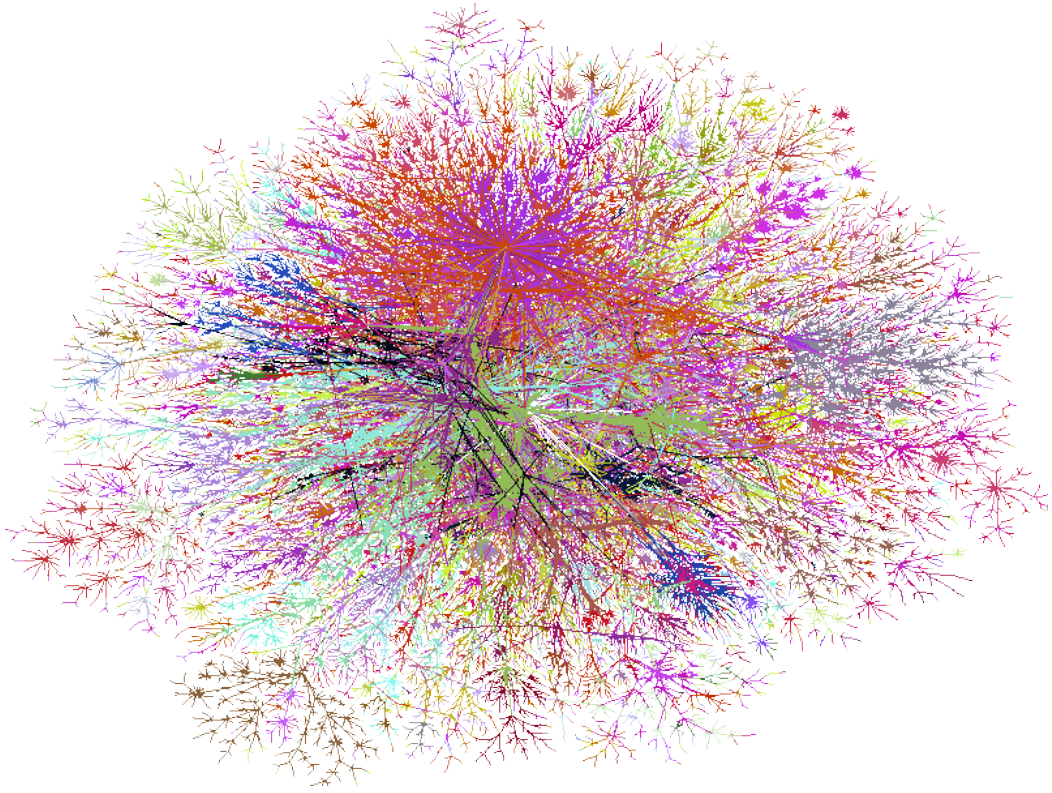
***Liveness*: # failures $\leq f_L = N - T$ (1 non-faulty quorum)**

Typically $N = 3f + 1$ and $T = 2f + 1$ to tolerate $f_S = f_L = f$ failures

Problem: at Internet level, can't enumerate N nodes

- Politically infeasible to decree set of globally trusted authorities

Decentralized global systems



The Internet itself provides an interesting precedent

- Single global network
- No globally trusted consortium dictates network structure
- Rather, network made from *pairwise* peering & transit relationships

Can we use pairwise trust to achieve secure global consensus?

Federated Byzantine Agreement (FBA)

Generalize Byzantine agreement so participants determine quorums in decentralized way

Each node v picks one or more *quorum slices*

- v only trusts quorums that are a superset of one of its slices
- If you care about an authority, put it in all your slices

Definition (Federated Byzantine Agreement System)

An FBAS is of a a set of nodes $\mathbf{V}$ and a quorum function $\mathbf{Q}$, where $\mathbf{Q}(v)$ is the set slices chosen by node v.

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 $q \subseteq U$

Federated Byzantine Agreement

Generalize Byzantine agreement so participants determine quorums in decentralized way

Each node v picks one or more *quorum slices*

- v only trusts quorums that are a superset of one of its slices
- If you care about an authority, put it in all your slices

Definition (Federated Byzantine Agreement System)

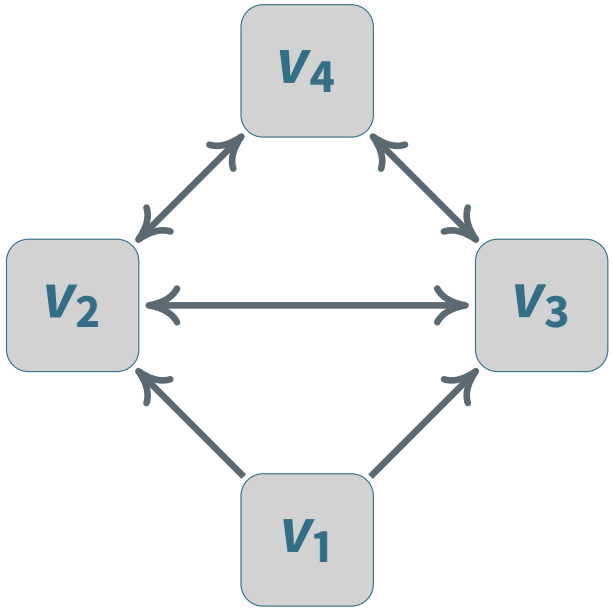
An FBAS is of a a set of nodes $\mathbf{V}$ and a quorum function $\mathbf{Q}$, where $\mathbf{Q}(v)$ is the set slices chosen by node v.

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 $q \subseteq U$

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**

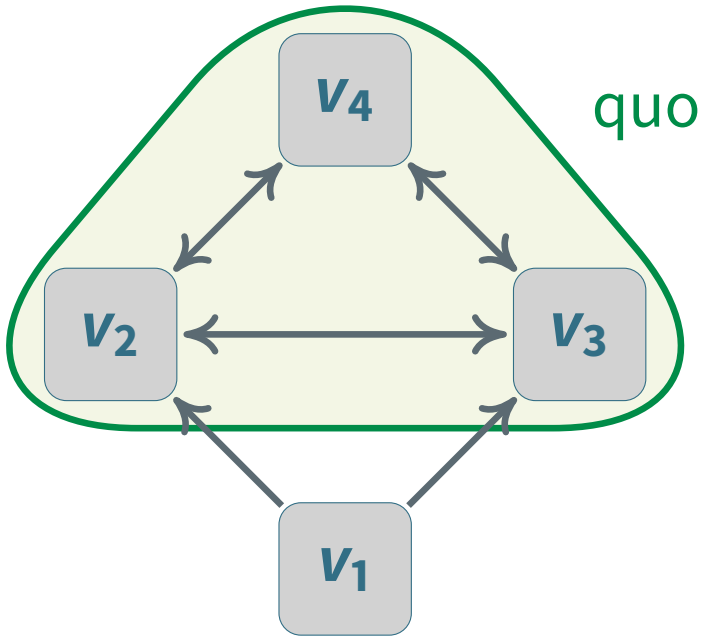
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- 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 $q \subseteq U$



quorum for v_2, v_3, v_4

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

Visualize quorum slice dependencies with arrows

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

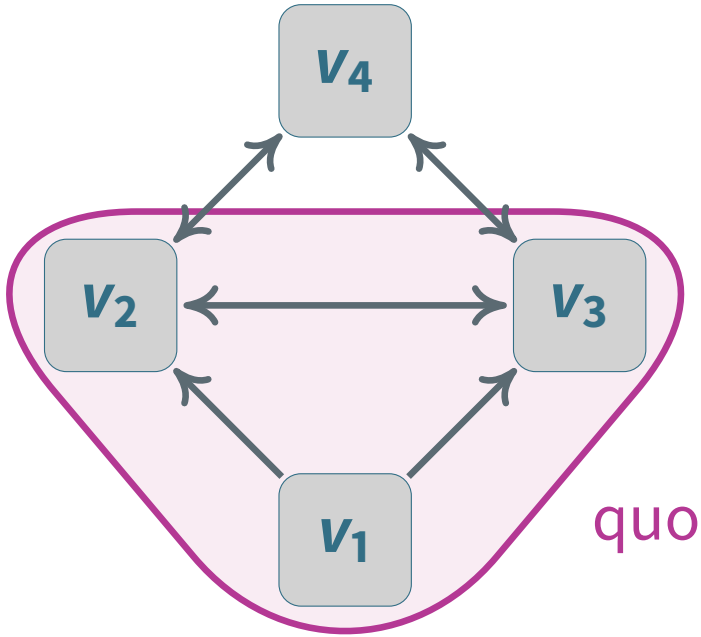
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- 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**

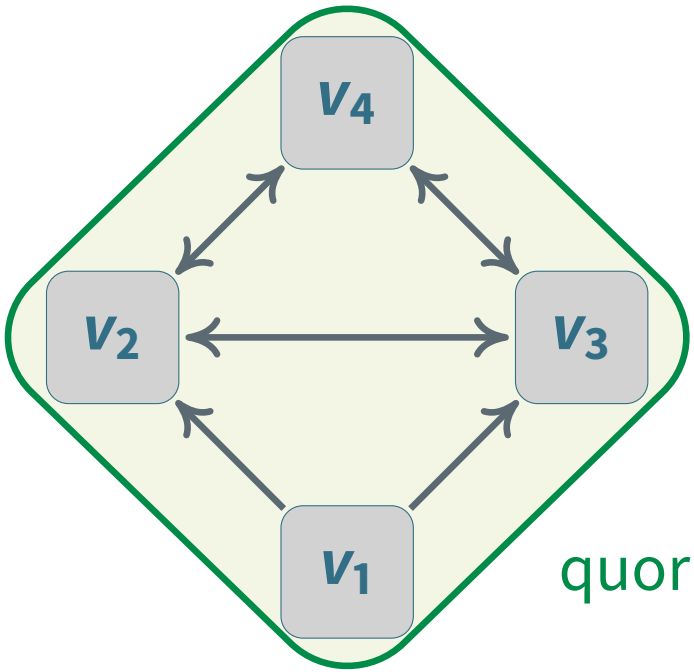
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- 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 $q \subseteq U$



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

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

quorum for $v_1, \ldots, v_4$

Visualize quorum slice dependencies with arrows

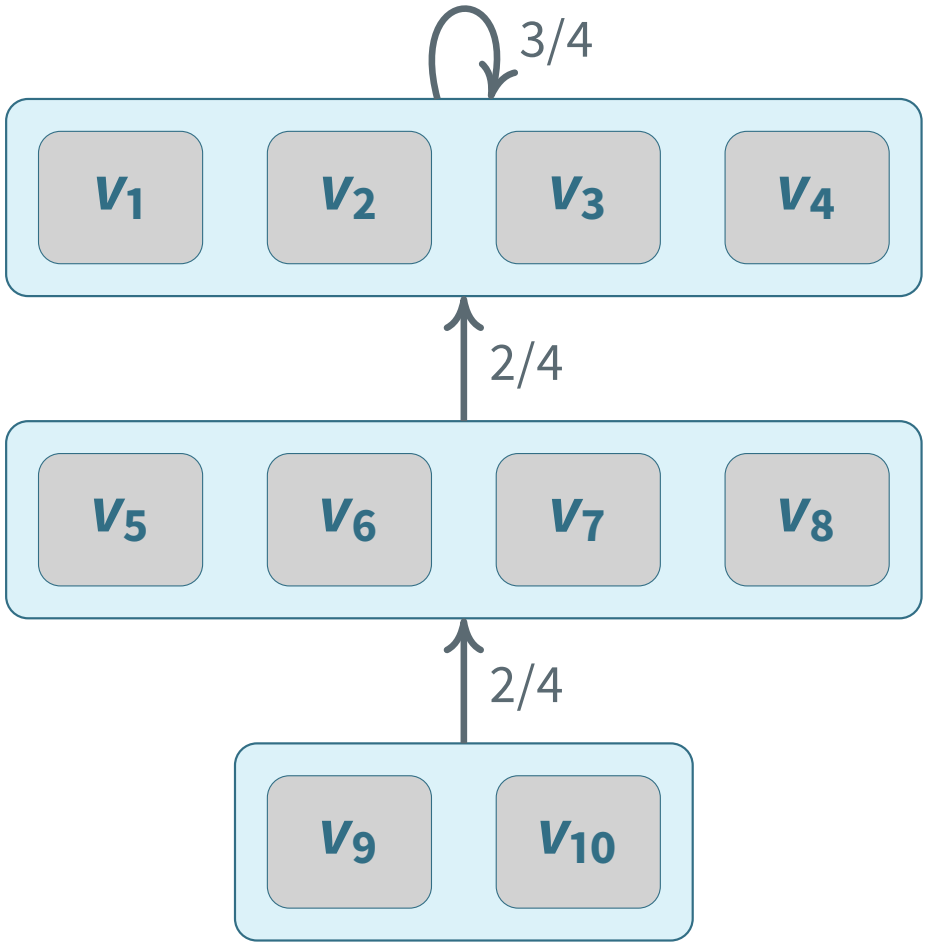
v_2, v_3, v_4 **is a quorum—contains a slice of each member**

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

Tiered quorum slice example



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

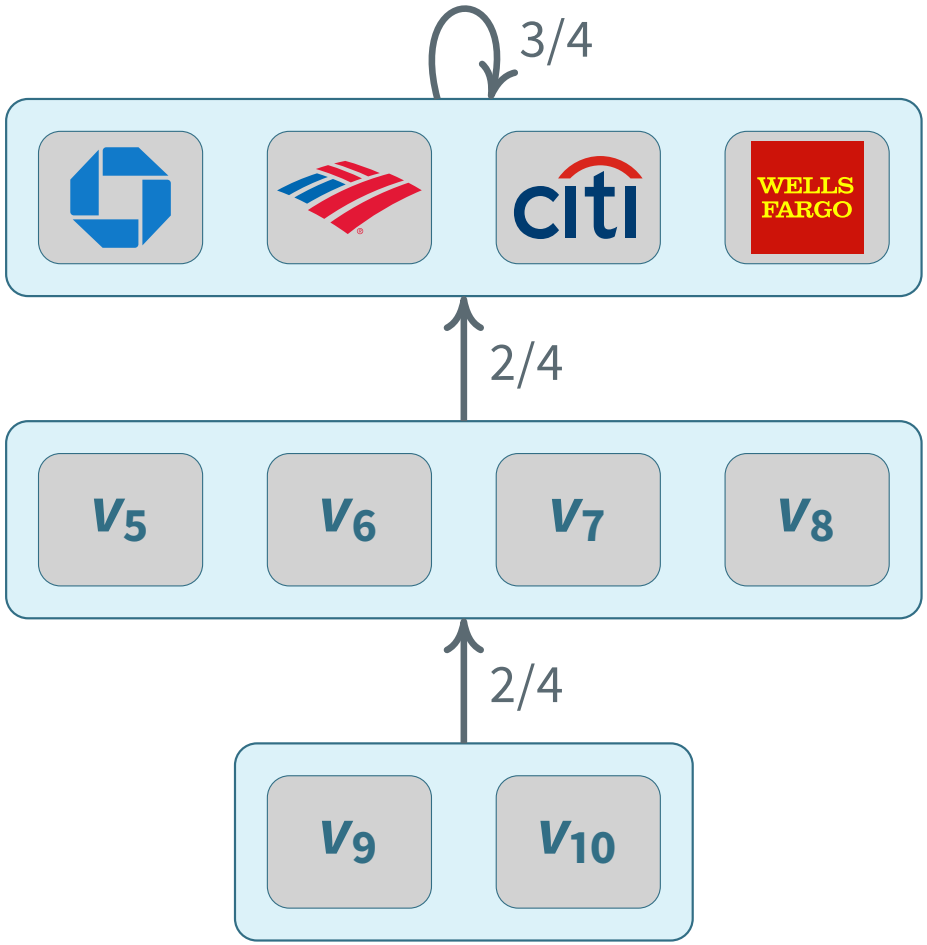
Middle tier: slice is self + any two top tier nodes

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

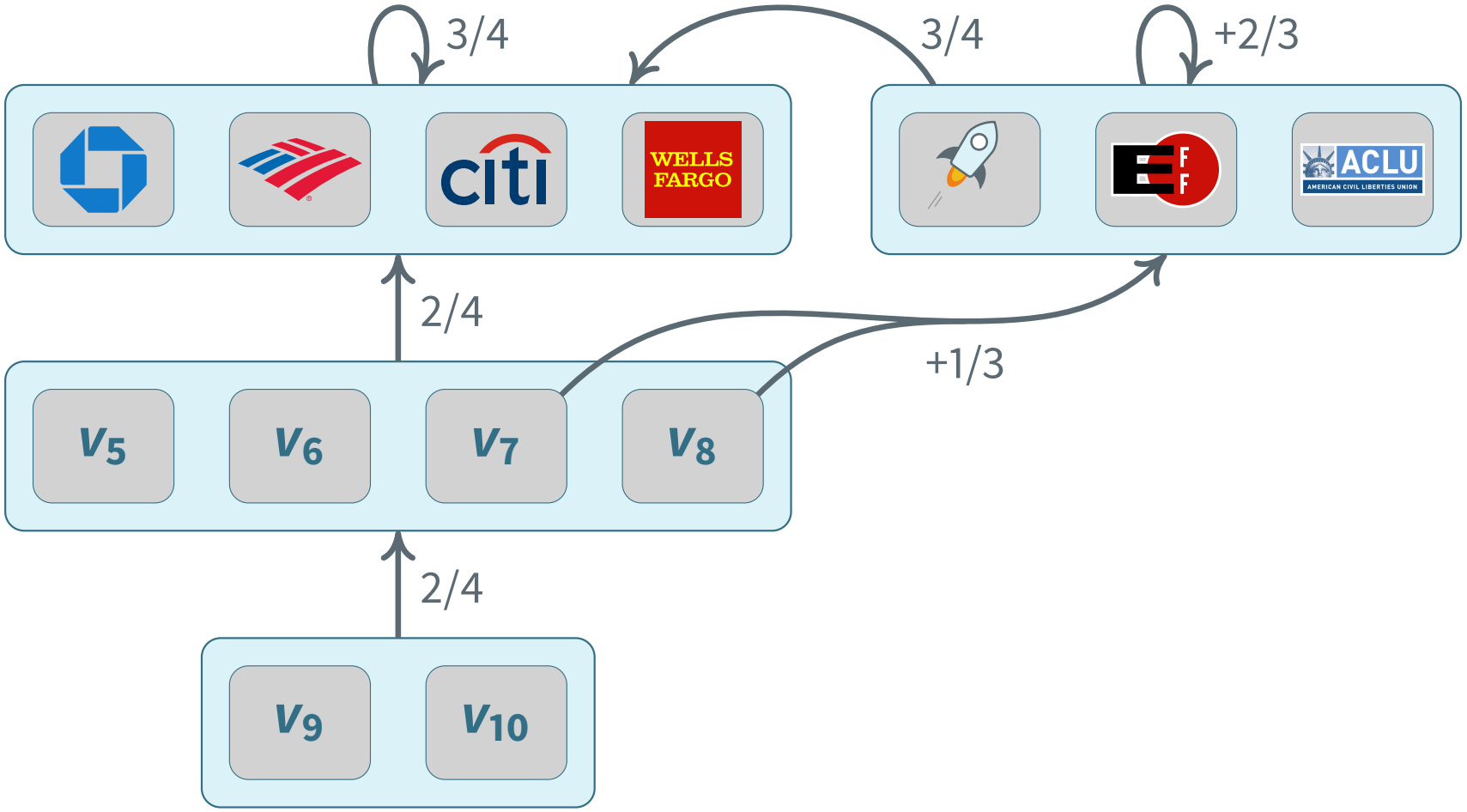
Tiered quorum slice example



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

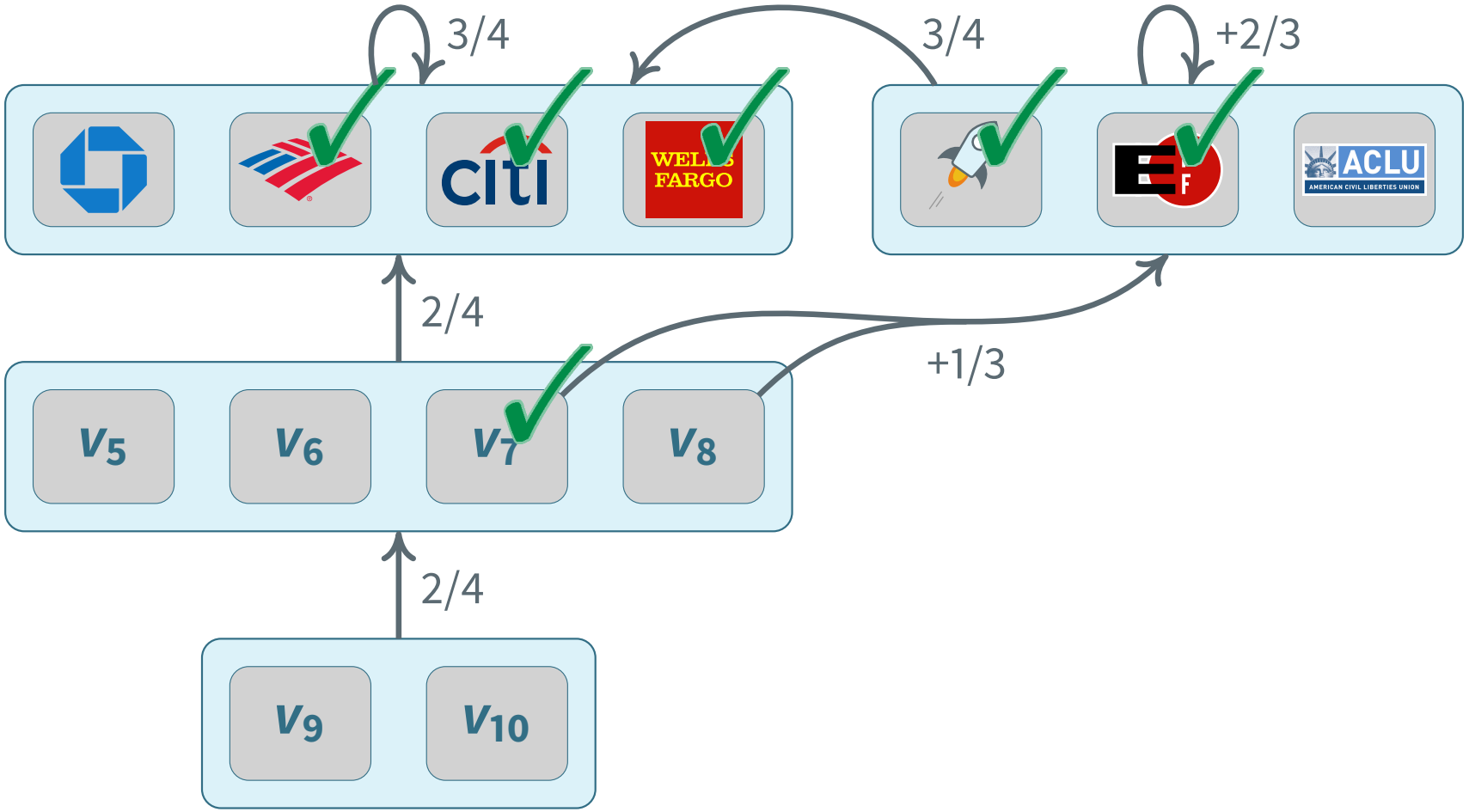
Tiered quorum slice example



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

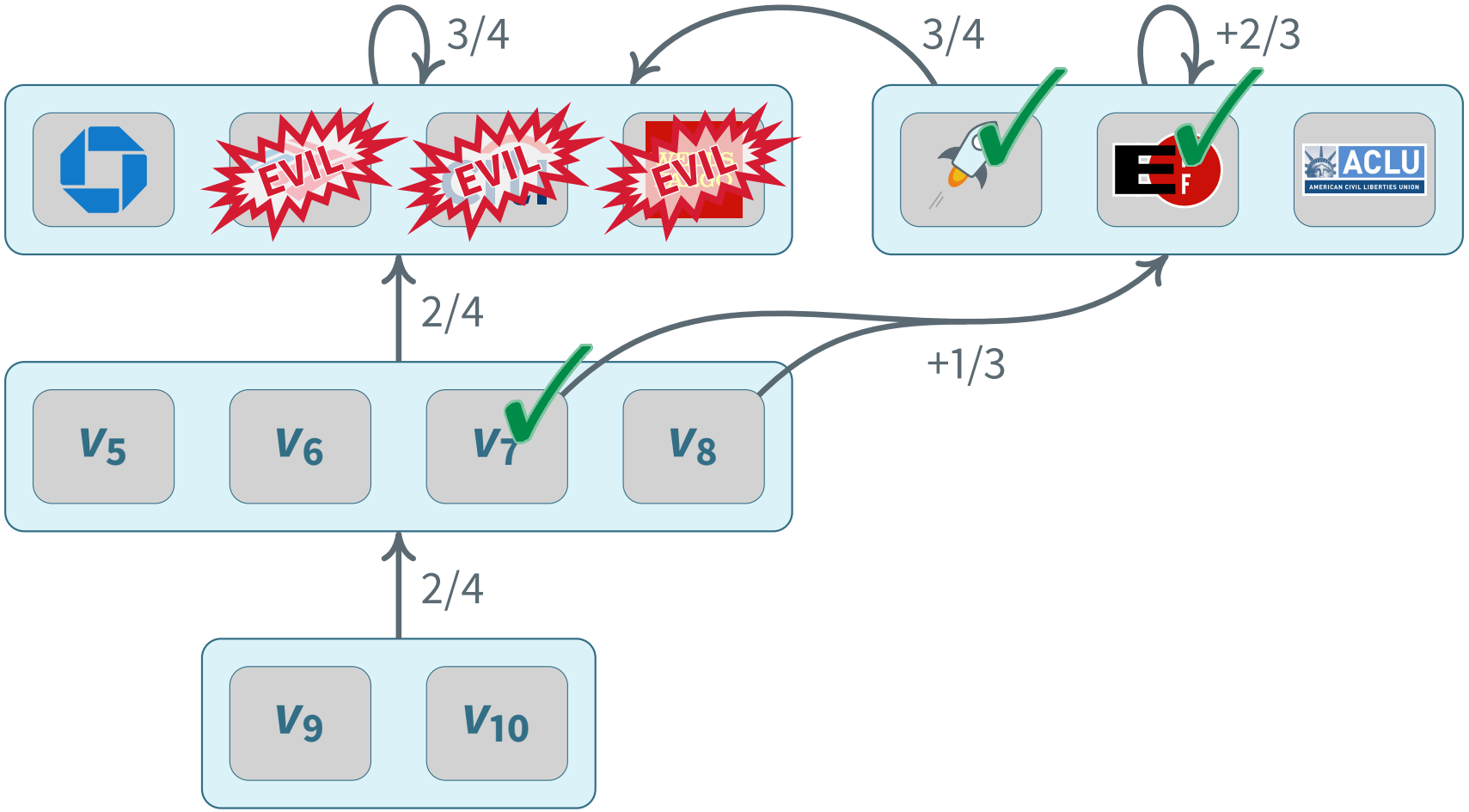
Tiered quorum slice example



Example: Citibank pays $1,000,000,000 Chase dollars 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

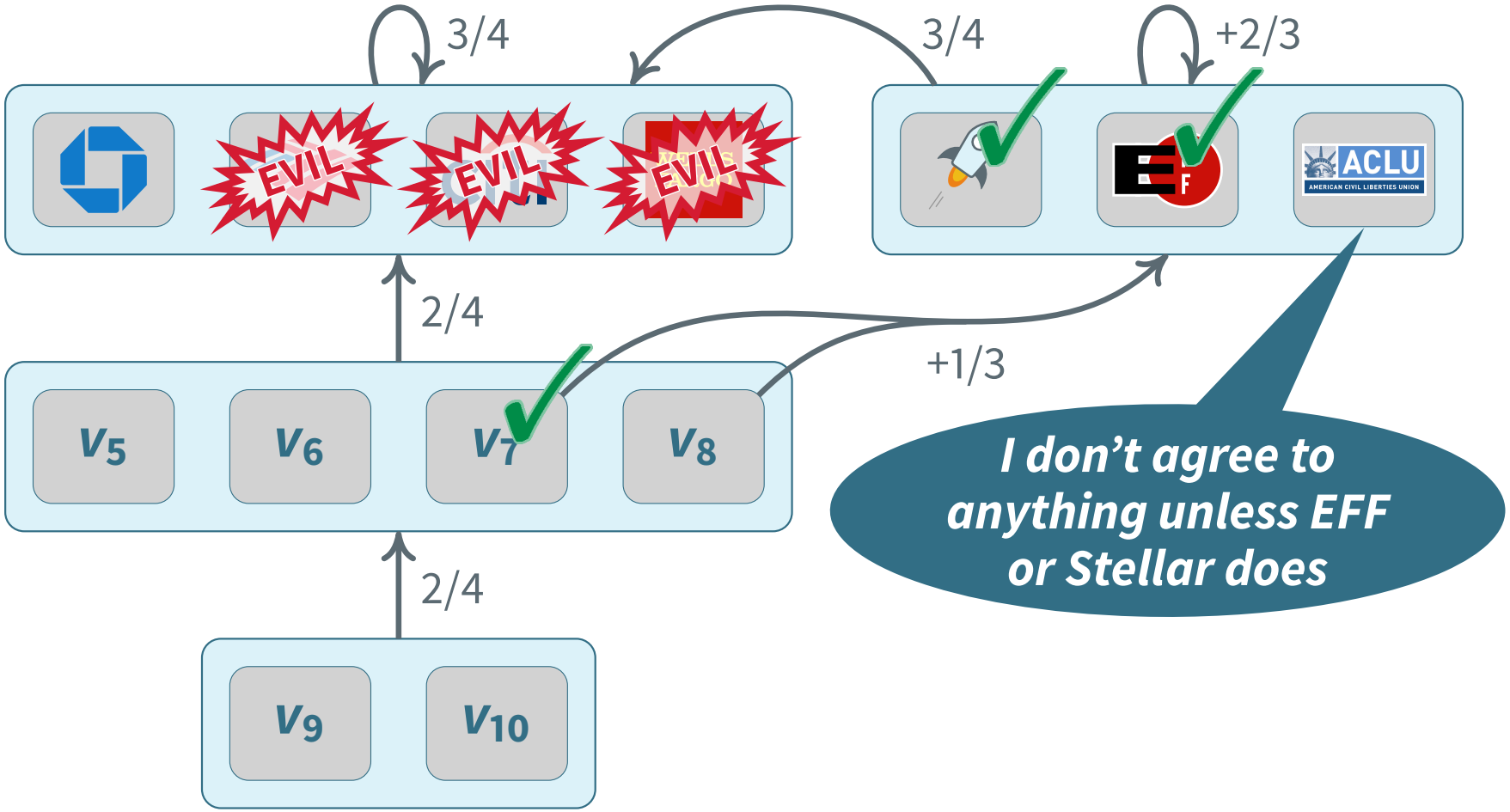
Tiered quorum slice example



Example: Citibank pays $1,000,000,000 Chase dollars 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

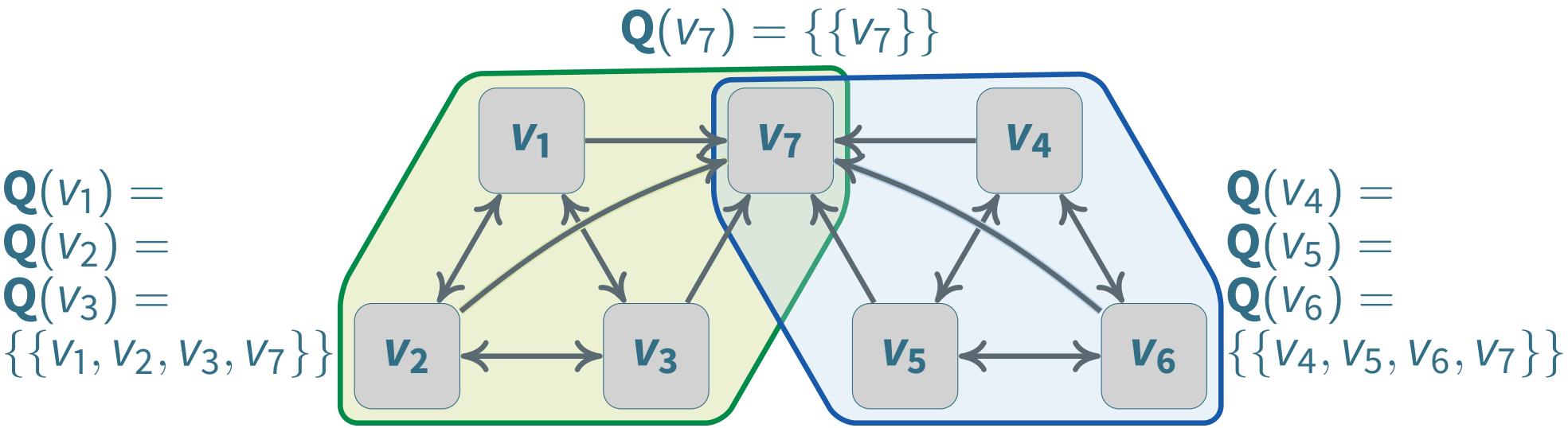
Tiered quorum slice example



Example: Citibank pays $1,000,000,000 Chase dollars 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

FBA fault tolerance



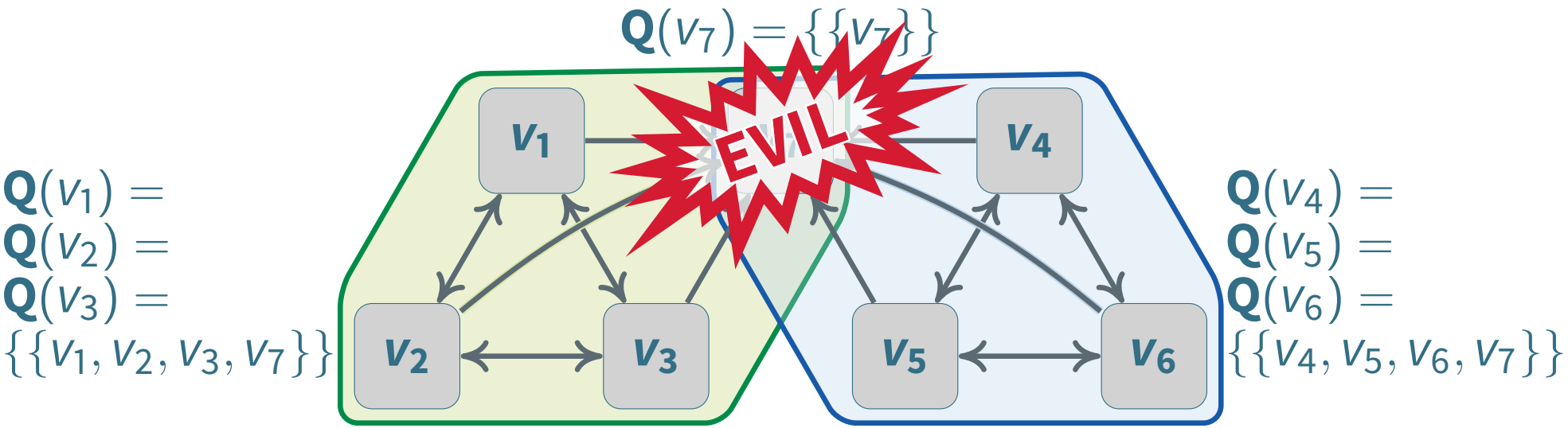
For safety, every two quorums must share a correct node

- Conceptually remove all ill-behaved nodes from all slices
- If two disjoint quorums result, can't guarantee safety

For liveness, well-behaved nodes must form a quorum

- Otherwise, depend on failed nodes to reach agreement

FBA fault tolerance



For safety, every two quorums must share a correct node

- Conceptually remove all ill-behaved nodes from all slices
- If two disjoint quorums result, can't guarantee safety

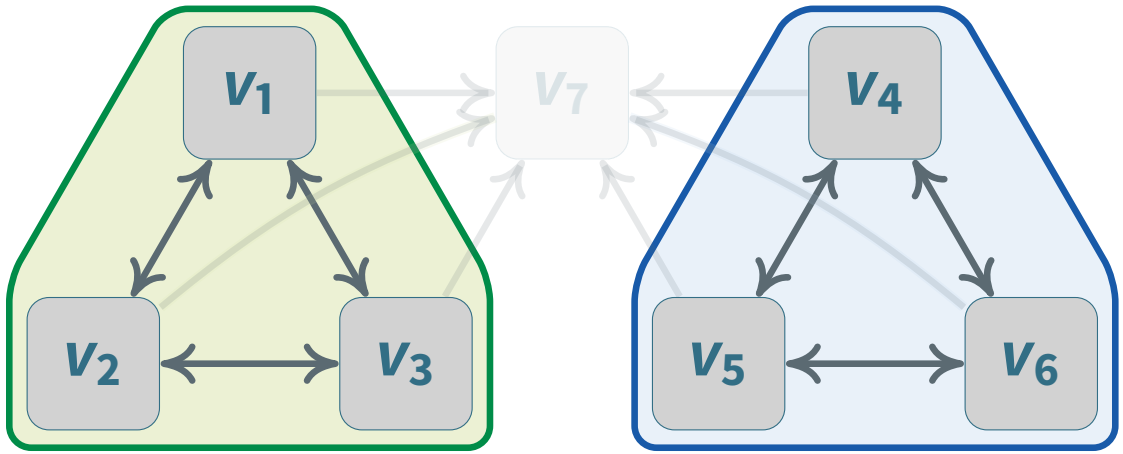
For liveness, well-behaved nodes must form a quorum

- Otherwise, depend on failed nodes to reach agreement

FBA fault tolerance

$\mathbf{Q}(v_7) = \{\{v_7\}\}$

$\mathbf{Q}(v_1) =$
$\mathbf{Q}(v_2) =$
$\mathbf{Q}(v_3) =$
$\{\{v_1, v_2, v_3, v_7\}\}$



$\mathbf{Q}(v_4) =$
$\mathbf{Q}(v_5) =$
$\mathbf{Q}(v_6) =$
$\{\{v_4, v_5, v_6, v_7\}\}$

For safety, every two quorums must share a correct node

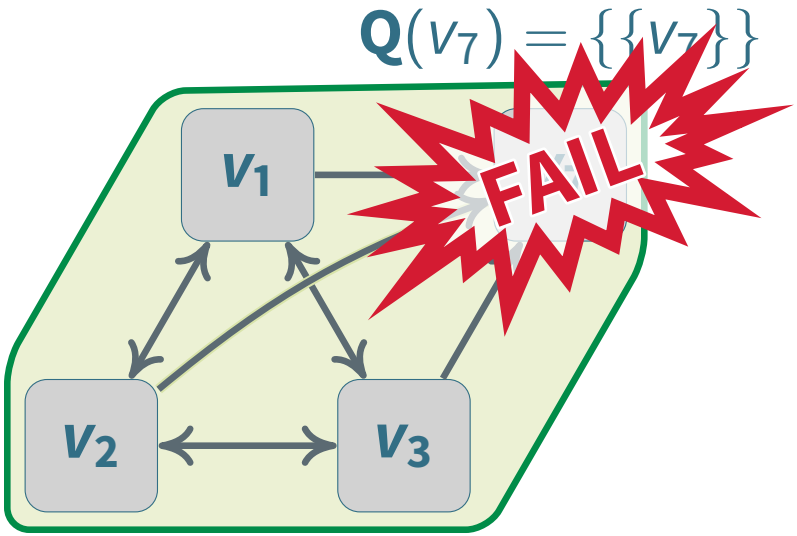
- Conceptually remove all ill-behaved nodes from all slices
- If two disjoint quorums result, can't guarantee safety

For liveness, well-behaved nodes must form a quorum

- Otherwise, depend on failed nodes to reach agreement

FBA fault tolerance

$\mathbf{Q}(v_1) =$
$\mathbf{Q}(v_2) =$
$\mathbf{Q}(v_3) =$
$\{\{v_1, v_2, v_3, v_7\}\}$



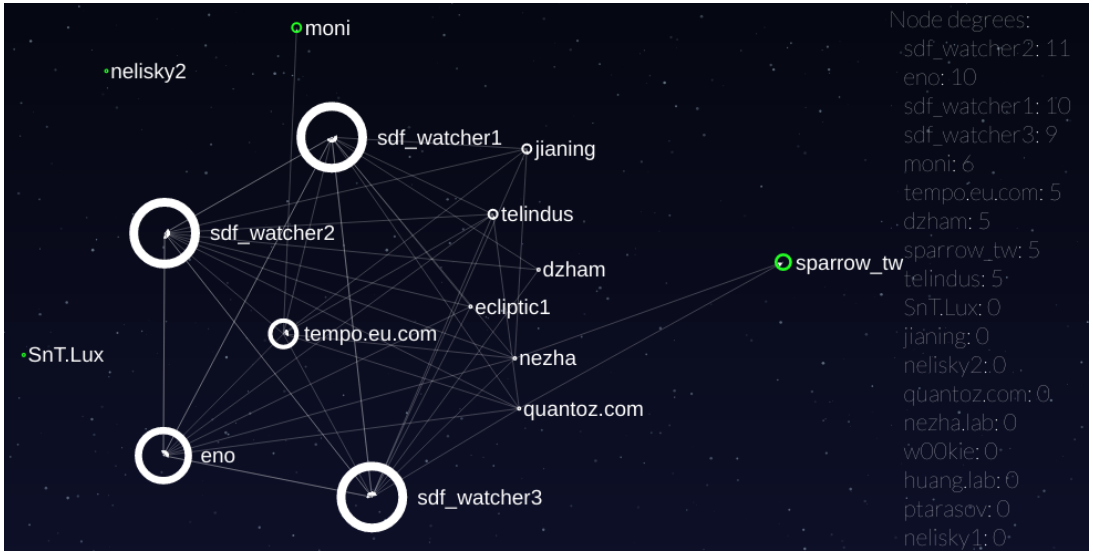
For safety, every two quorums must share a correct node

- Conceptually remove all ill-behaved nodes from all slices
- If two disjoint quorums result, can't guarantee safety

For liveness, well-behaved nodes must form a quorum

- Otherwise, depend on failed nodes to reach agreement

The Stellar Consensus Protocol [SCP]



FBA protocol in production use by Stellar payment network

Guarantees safety if every two quorums share honest node

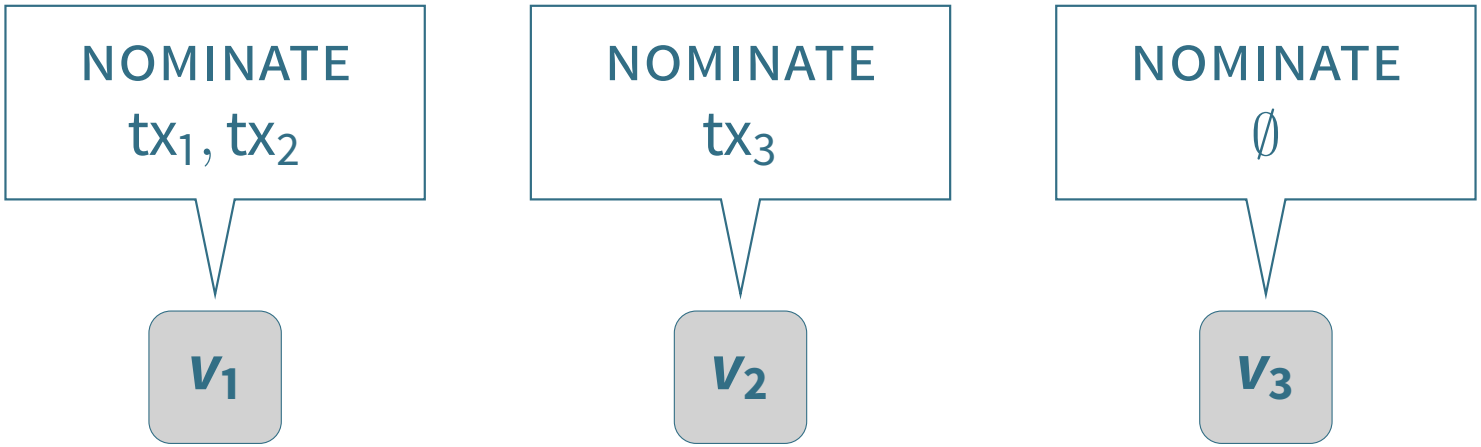
- Optimal, as otherwise can't guarantee safety

Guarantees a well-behaved quorum will not get stuck

Core idea: *federated voting*

- Nodes exchanges signed vote messages to agree on statements
- Every message also specifies the voter's quorum slices
- Allows dynamic quorum discovery while assembling votes

SCP: High-level view



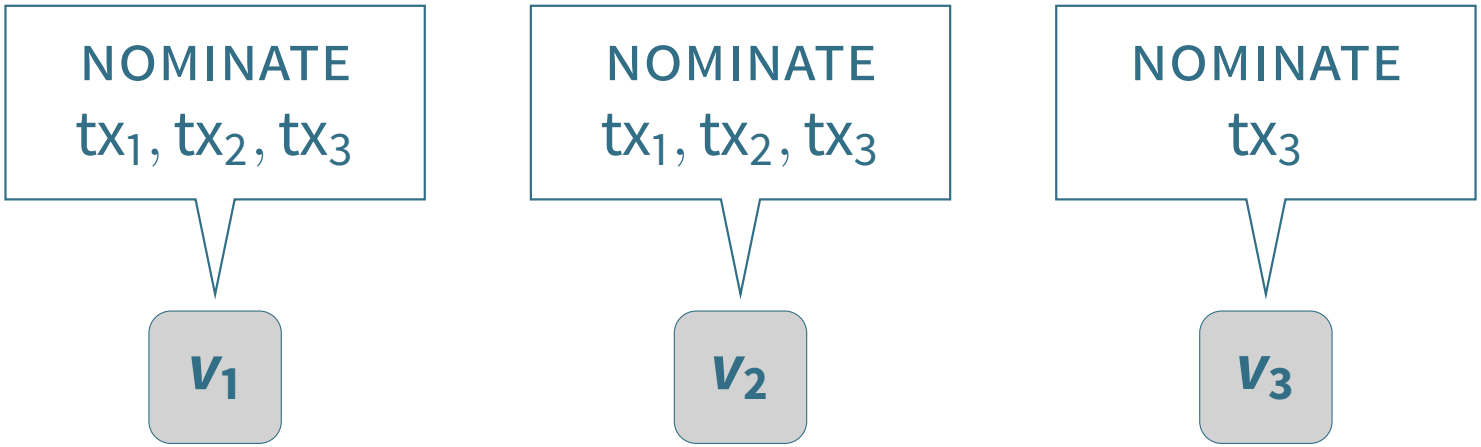
Phase 1: Nomination

- Nodes nominate values
- Propagate and converge on set of nominated values
- Deterministically combine nominated values into *composite* value x
- Problem: impossible to know when protocol has converged

Phase 2: Balloting

- Similar to Byzantine Paxos, but with federated voting
- Works (less efficiently) even when Phase 1 hasn't converged

SCP: High-level view



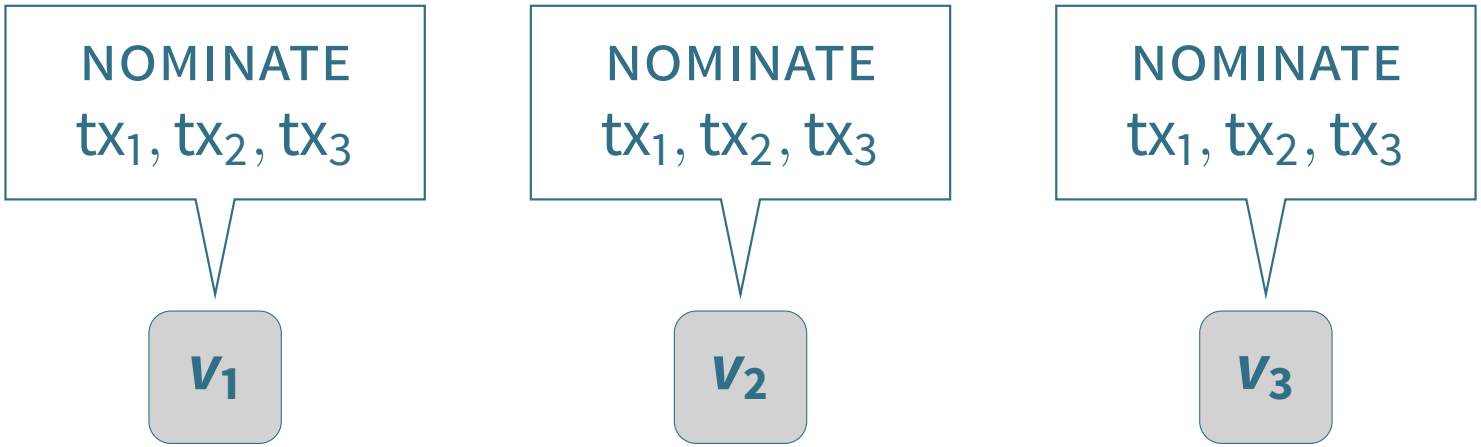
Phase 1: Nomination

- Nodes nominate values
- Propagate and converge on set of nominated values
- Deterministically combine nominated values into *composite* value x
- Problem: impossible to know when protocol has converged

Phase 2: Balloting

- Similar to Byzantine Paxos, but with federated voting
- Works (less efficiently) even when Phase 1 hasn't converged

SCP: High-level view



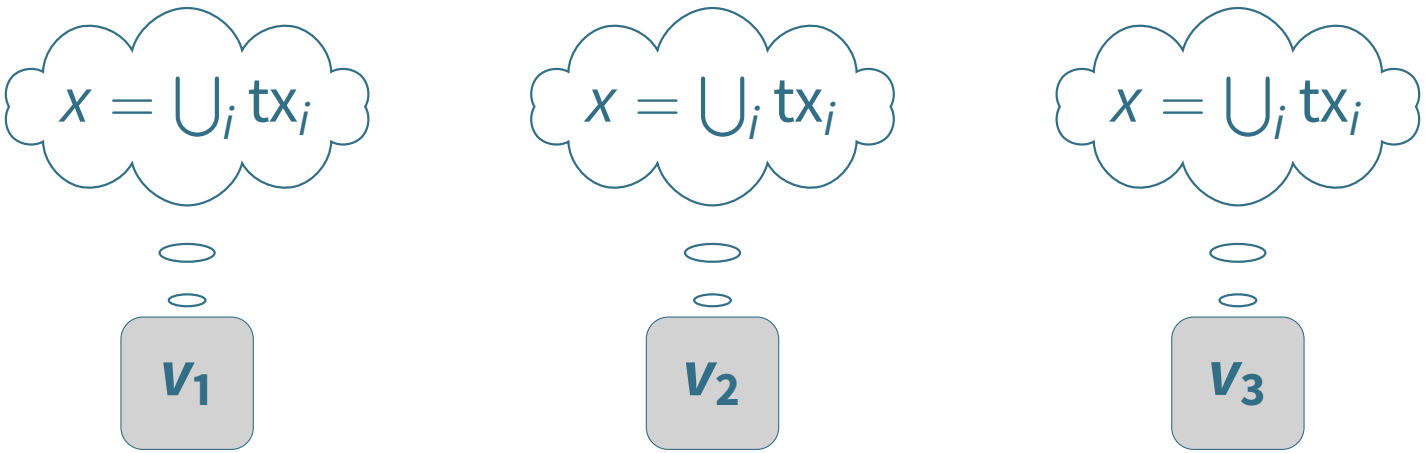
Phase 1: Nomination

- Nodes nominate values
- Propagate and converge on set of nominated values
- Deterministically combine nominated values into *composite* value x
- Problem: impossible to know when protocol has converged

Phase 2: Balloting

- Similar to Byzantine Paxos, but with federated voting
- Works (less efficiently) even when Phase 1 hasn't converged

SCP: High-level view



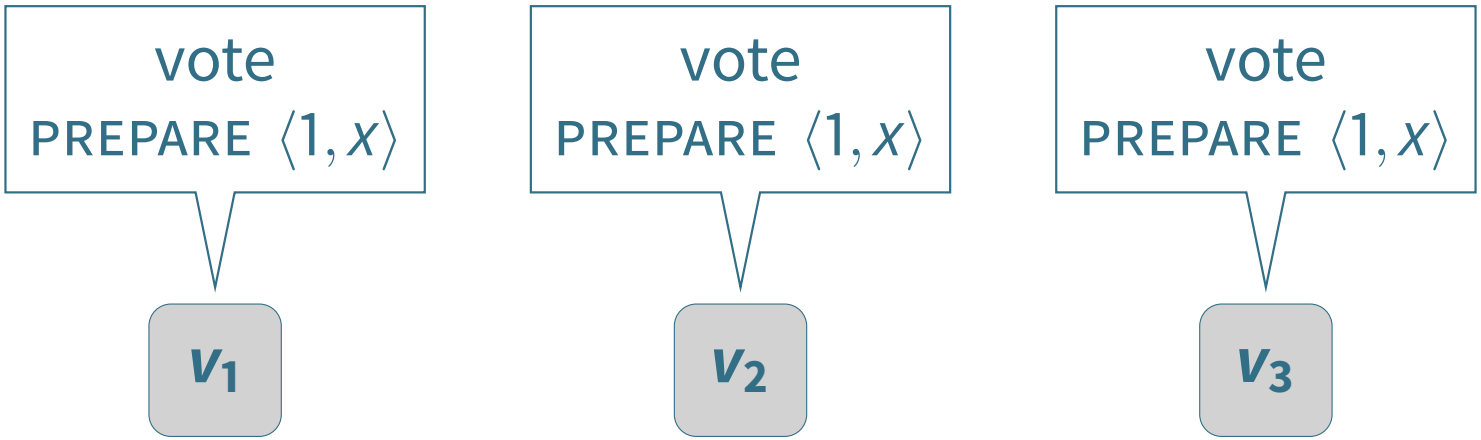
Phase 1: Nomination

- Nodes nominate values
- Propagate and converge on set of nominated values
- Deterministically combine nominated values into *composite* value x
- Problem: impossible to know when protocol has converged

Phase 2: Balloting

- Similar to Byzantine Paxos, but with federated voting
- Works (less efficiently) even when Phase 1 hasn't converged

SCP: High-level view



Phase 1: Nomination

- Nodes nominate values
- Propagate and converge on set of nominated values
- Deterministically combine nominated values into *composite* value x
- Problem: impossible to know when protocol has converged

Phase 2: Balloting

- Similar to Byzantine Paxos, but with federated voting
- Works (less efficiently) even when Phase 1 hasn't converged

Comparison to other approaches

mechanism	open network	low latency	flexible trust	asympt. security
SCP	✓	✓	✓	✓
Byzantine agr.		✓	✓	✓
proof-of-work	✓			
proof-of-stake	✓	maybe		maybe

Use traditional Byzantine agreement over closed server set?

- Paranoid users will check outside audits anyway (ersatz FBA)
- Might as well formalize the arrangement to get optimal safety

Use Bitcoin block chain (proof-of-work) for ILC?

- Consensus intricately tied up with coin distribution & incentives
- Incentives might be insufficient or ill-suited to other applications

More information

Stellar consensus protocol:
`www.stellar.org/papers/stellar-consensus-protocol.pdf`

IETF Internet-level consensus (ILC) mailing list:
`www.ietf.org/mailman/listinfo/ilc`

Stellar development foundation: `www.stellar.org`