The Stellar Consensus Protocol

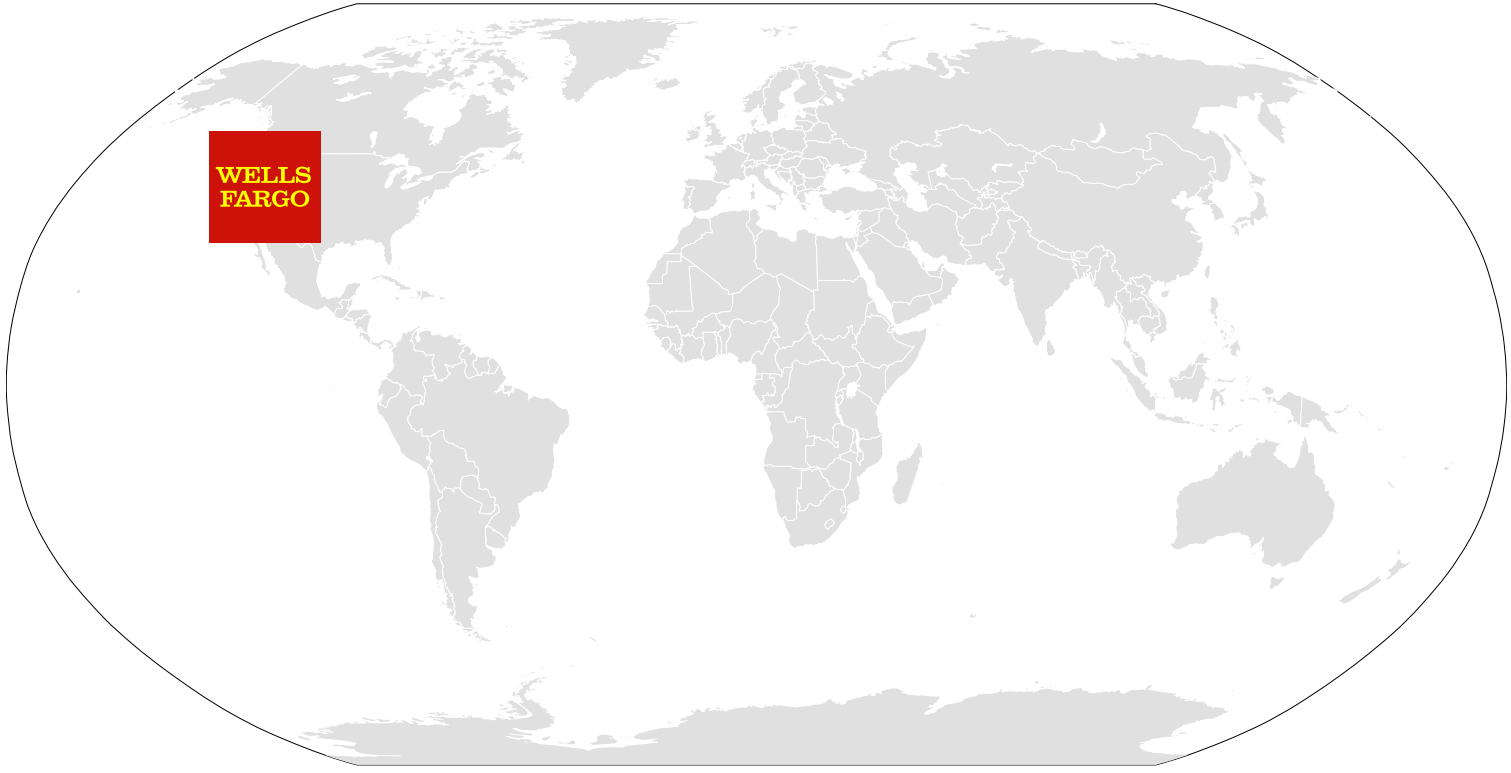
A federated model for Internet-level consensus

David Mazières

Stellar Development Foundation

Saturday, September 24, 2016

Banking system robustness



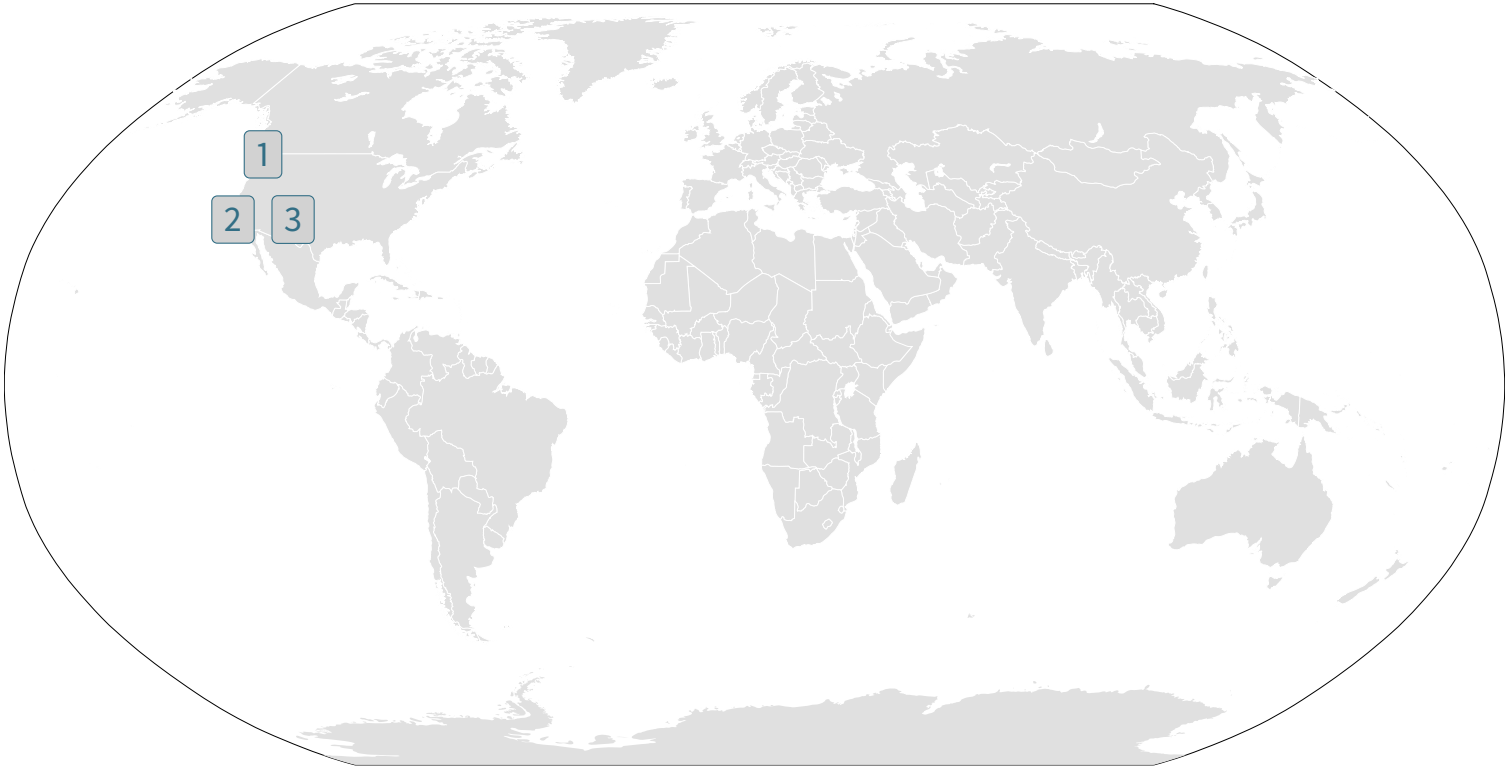
A bank's core job is tracking customer assets and liabilities

- Ensure robustness of ledger through *replication*

What about the robustness of inter-bank transfers?

- Need trusted correspondent bank; adds delay and cost
- In some cases makes transfers flat-out impossible

Banking system robustness



A bank's core job is tracking customer assets and liabilities

- Ensure robustness of ledger through *replication*

What about the robustness of inter-bank transfers?

- Need trusted correspondent bank; adds delay and cost
- In some cases makes transfers flat-out impossible

Banking system robustness

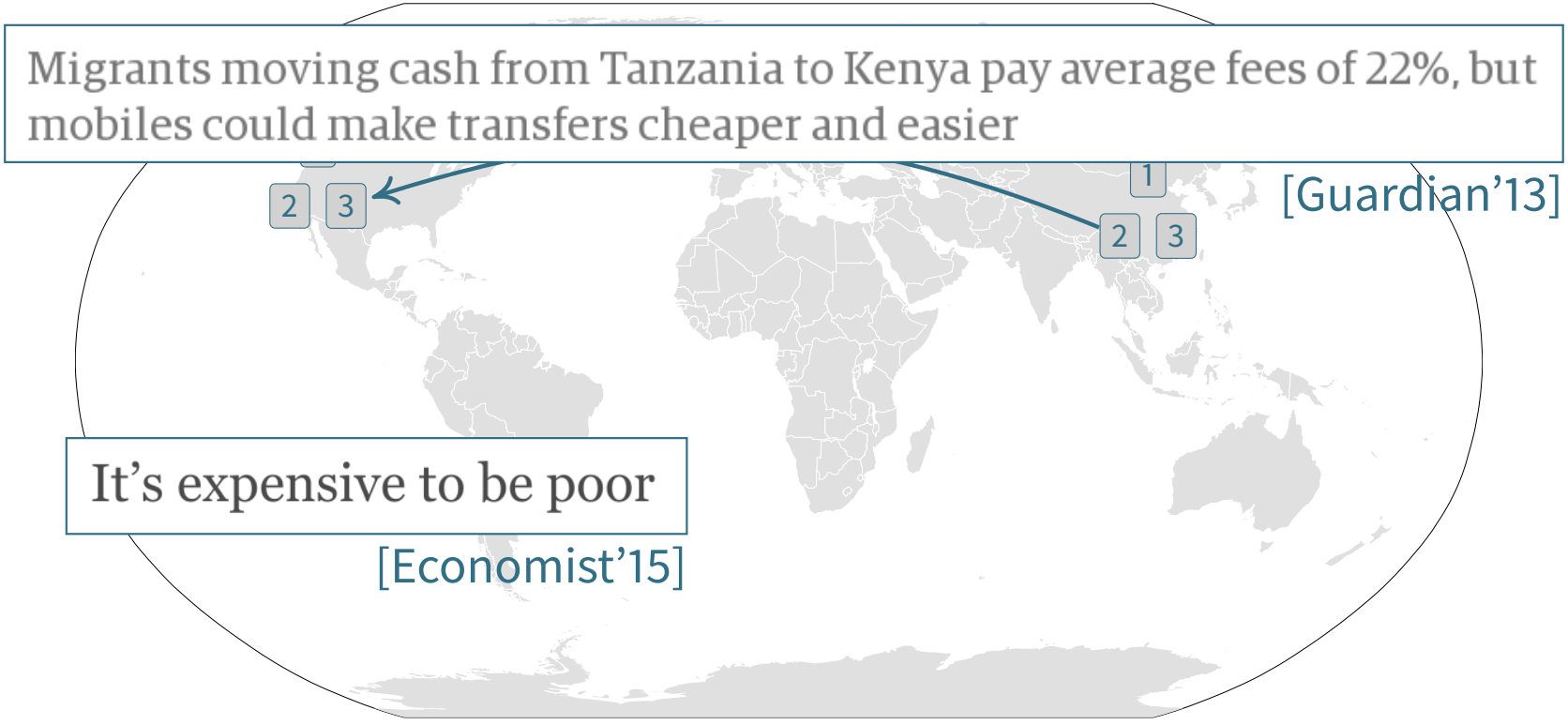
A bank's core job is tracking customer assets and liabilities

- Ensure robustness of ledger through *replication*

What about the robustness of inter-bank transfers?

- Need trusted correspondent bank; adds delay and cost
- In some cases makes transfers flat-out impossible

Banking system robustness



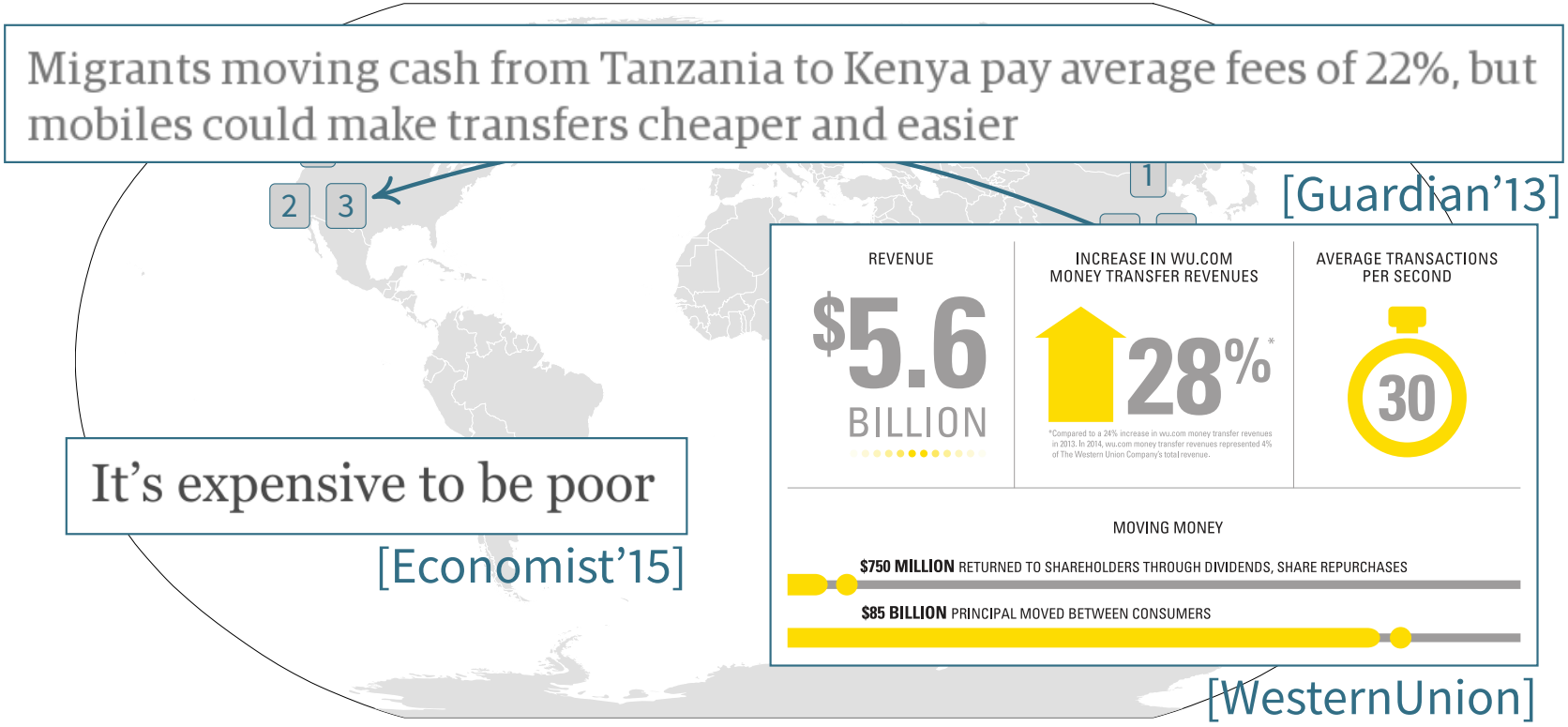
A bank's core job is tracking customer assets and liabilities

- Ensure robustness of ledger through *replication*

What about the robustness of inter-bank transfers?

- Need trusted correspondent bank; adds delay and cost
- In some cases makes transfers flat-out impossible

Banking system robustness



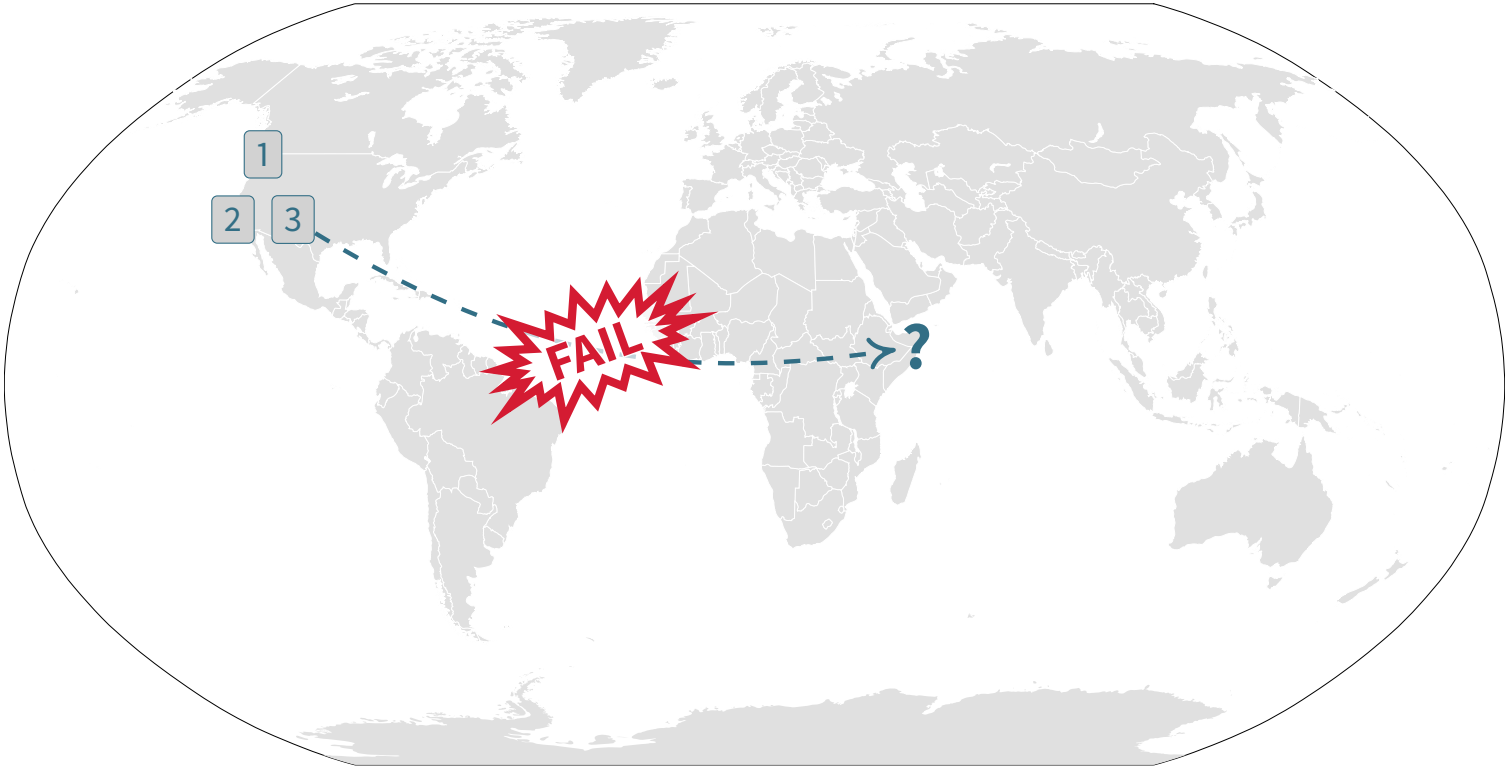
A bank's core job is tracking customer assets and liabilities

- Ensure robustness of ledger through *replication*

What about the robustness of inter-bank transfers?

- Need trusted correspondent bank; adds delay and cost
- In some cases makes transfers flat-out impossible

Banking system robustness



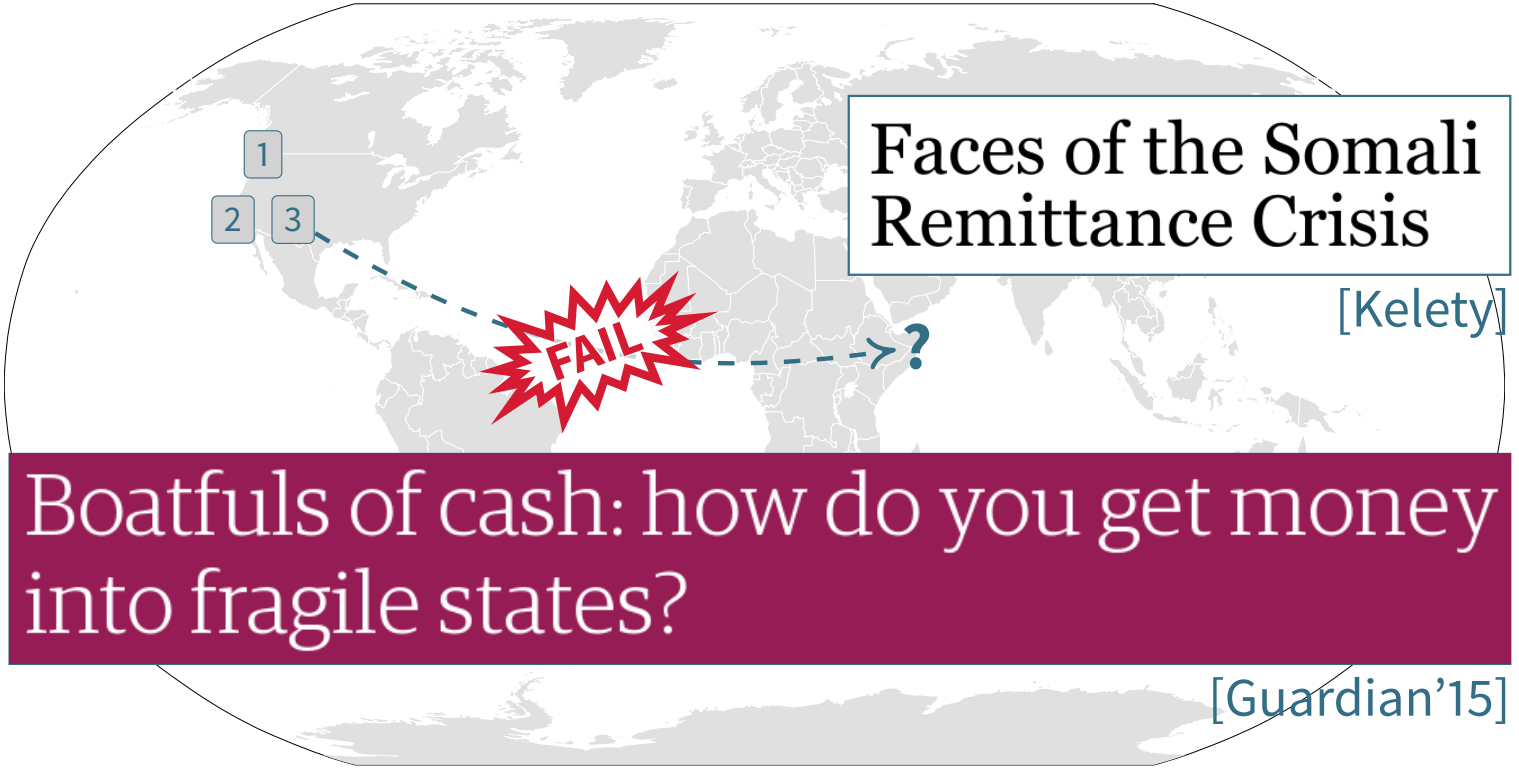
A bank's core job is tracking customer assets and liabilities

- Ensure robustness of ledger through *replication*

What about the robustness of inter-bank transfers?

- Need trusted correspondent bank; adds delay and cost
- In some cases makes transfers flat-out impossible

Banking system robustness



A bank's core job is tracking customer assets and liabilities

- Ensure robustness of ledger through *replication*

What about the robustness of inter-bank transfers?

- Need trusted correspondent bank; adds delay and cost
- In some cases makes transfers flat-out impossible

We need an inter-financial network

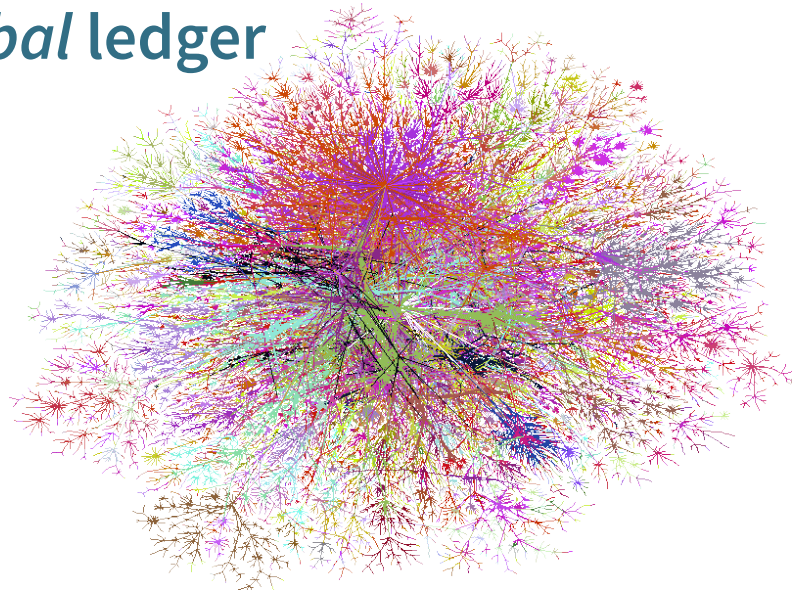
Today's financial networks like pre-Internet email

- Route a message through multiple UUCP nodes, BITNET, etc.
- Addresses not globally meaningful: `host1!host2!host3!user`

Can we make sending money as easy as Internet mail?

This would be possible if we had a *global* ledger

- Who would manage such a ledger?
- No globally trusted party or parties

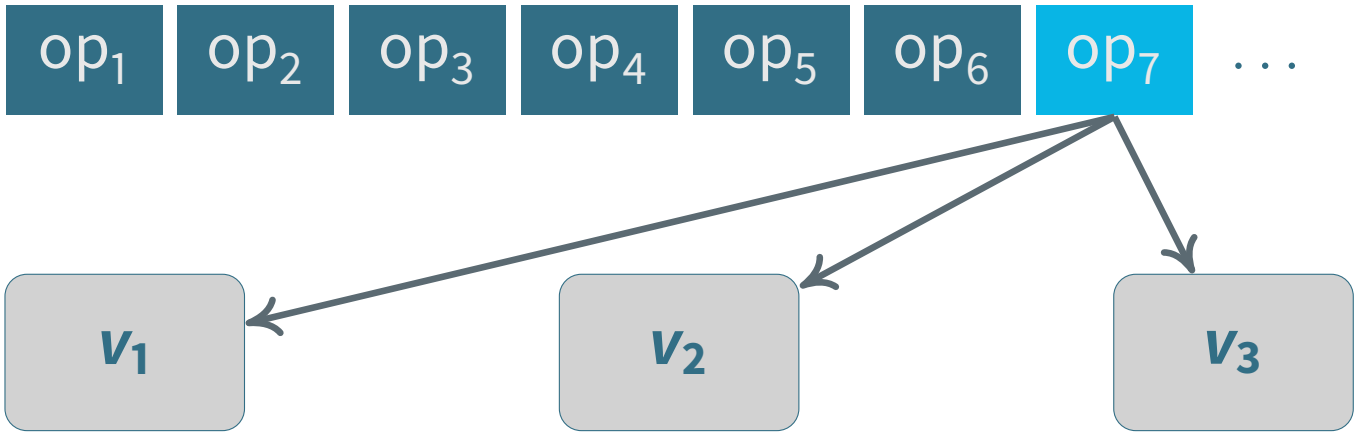


We do have an existence proof of a global decentralized system: inter-domain routing

Idea: use pairwise trust to achieve secure global consensus

- Use Internet-level consensus to realize a global ledger

Consensus: The key to replication



Main challenge of replicating data is keeping copies in sync

Common technique: *replicated state machine*

1. All replicas agree on initial state of system (easy), and
2. All replicas agree on sequence of deterministic operations

#2 requires *consensus* among replicas on each operation

- E.g., agree $\text{op}_7 = \text{credit dm \$150}$ before considering payment settled

Outline

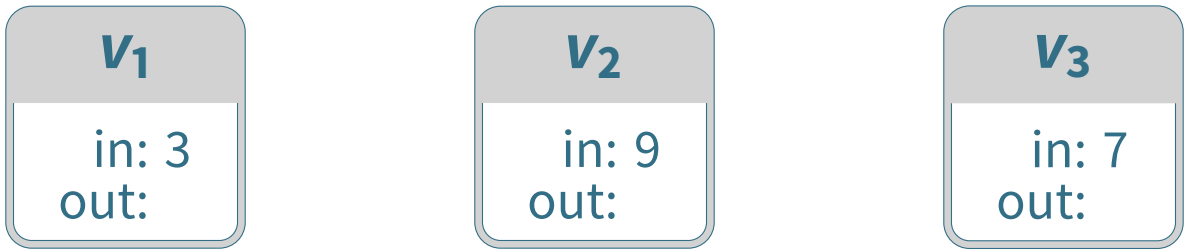
Byzantine agreement (background), recast in terms of voting on irrefutable & neutralizable statements

Federated Byzantine Agreement (FBA)

Optimal failure resilience

The Stellar consensus protocol (SCP)

The consensus problem



Goal: For multiple agents to agree on an output value

Each agent starts with an input value

- Typically a candidate for the *n*th op. on a replicated state machine

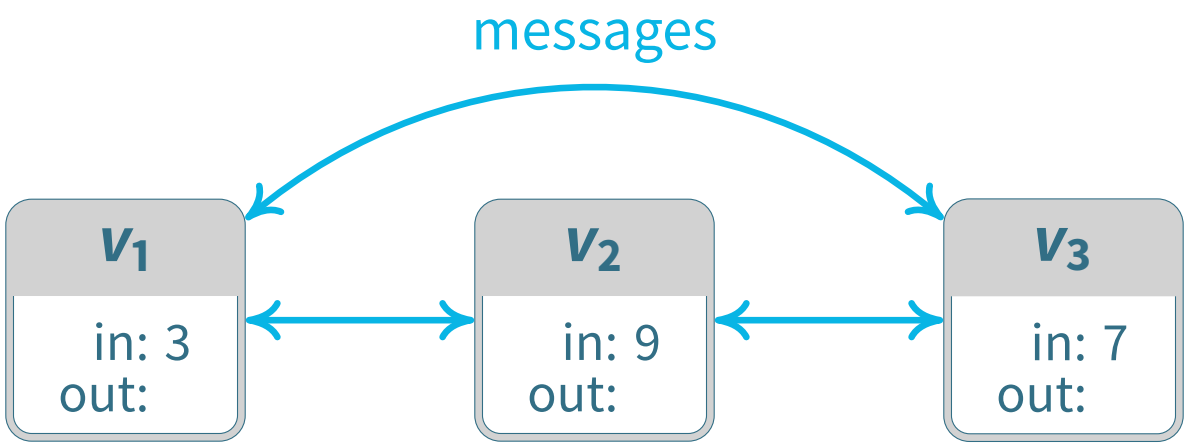
Agents communicate following some *consensus protocol*

- Use protocol to agree on one of the agent's input values

Once decided, agents output the chosen value

- Output is write-once (an agent cannot change its value)

The consensus problem



Goal: For multiple agents to agree on an output value

Each agent starts with an input value

- Typically a candidate for the *n*th op. on a replicated state machine

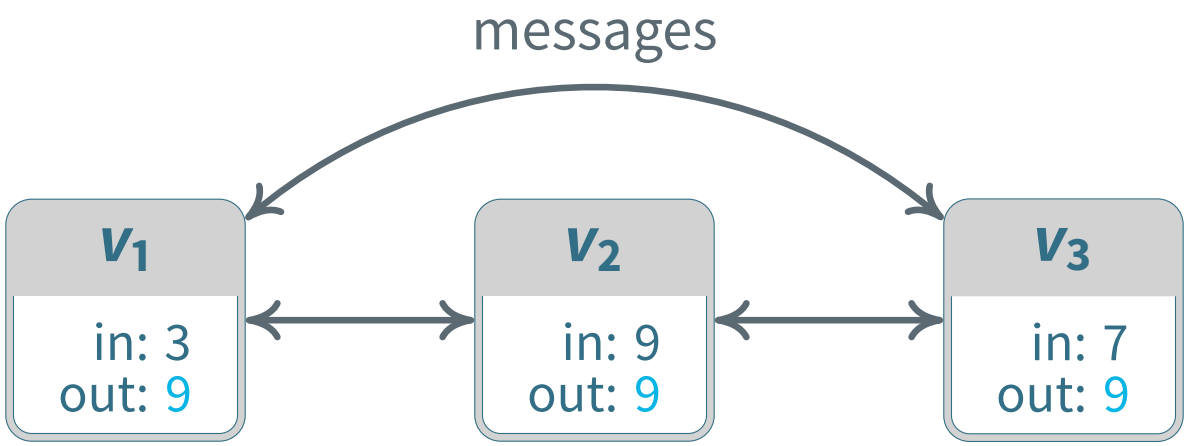
Agents communicate following some *consensus protocol*

- Use protocol to agree on one of the agent's input values

Once decided, agents output the chosen value

- Output is write-once (an agent cannot change its value)

The consensus problem



Goal: For multiple agents to agree on an output value

Each agent starts with an input value

- Typically a candidate for the *n*th op. on a replicated state machine

Agents communicate following some *consensus protocol*

- Use protocol to agree on one of the agent's input values

Once decided, agents output the chosen value

- Output is write-once (an agent cannot change its value)

Properties of a consensus protocol

A consensus protocol provides safety iff…

- All outputs produced have the same value (*agreement*), and
- The output value equals one of the agents' inputs (*validity*)

A consensus protocol provides liveness iff…

- Eventually non-faulty agents output a value (*termination*)

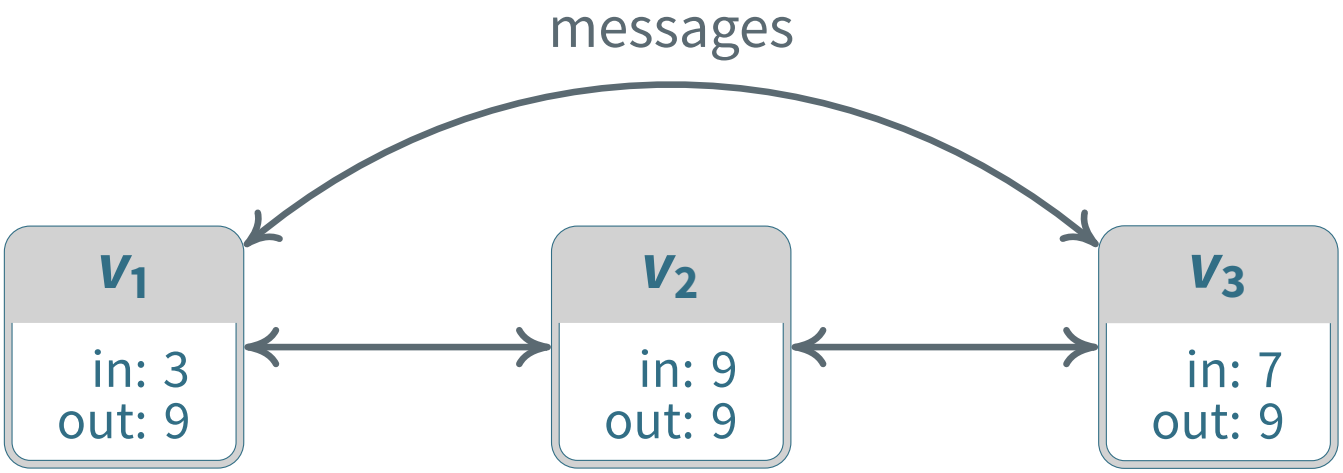
A consensus protocol provides fault tolerance iff…

- It can recover from the failure of an agent at any point
- *Fail-stop* protocols handle agent crashes
- *Byzantine-fault-tolerant* protocols handle arbitrary agent behavior

Theorem (FLP impossibility result)

No deterministic consensus protocol can guarantee all three of safety, liveness, and fault tolerance in an asynchronous system.

Bivalent states



Recall agents chose value 9 in last example

But a network outage might be indistinguishable from v_2 failing

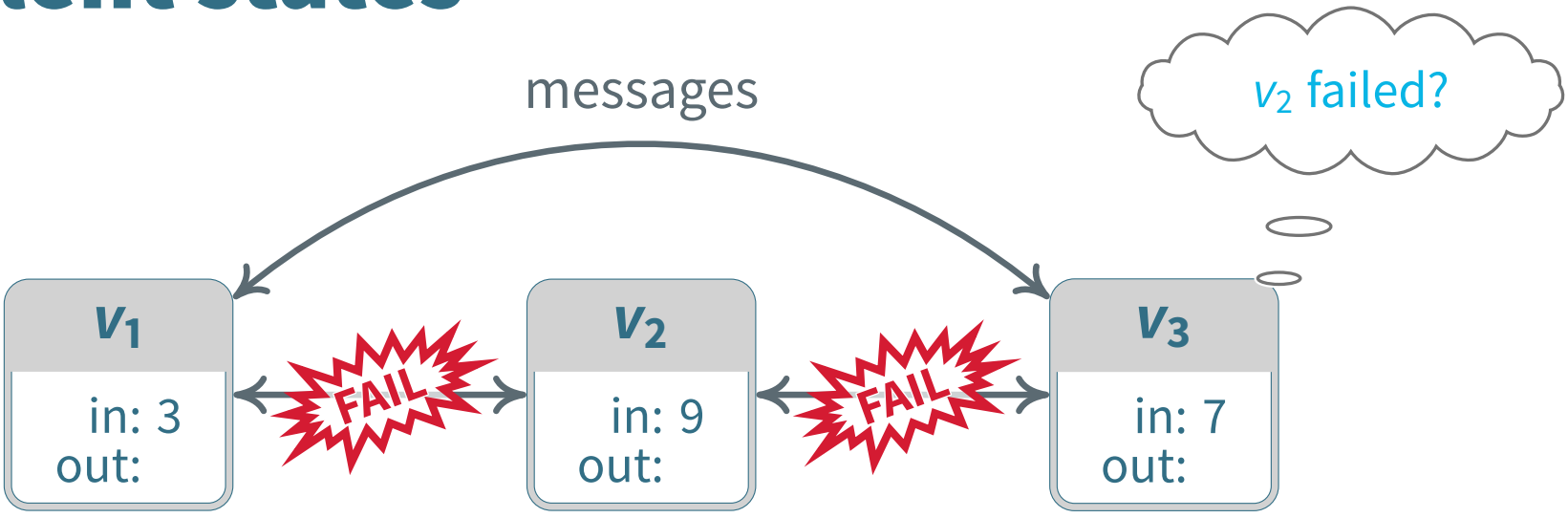
If protocol fault tolerant, v_1 and v_3 might decide to output 7

Once network back, v_2 must also output 7

Definition (Bivalent)

An execution of a consensus protocol is in a bivalent state when the network can affect which value agents choose.

Bivalent states



Recall agents chose value 9 in last example

But a network outage might be indistinguishable from v_2 failing

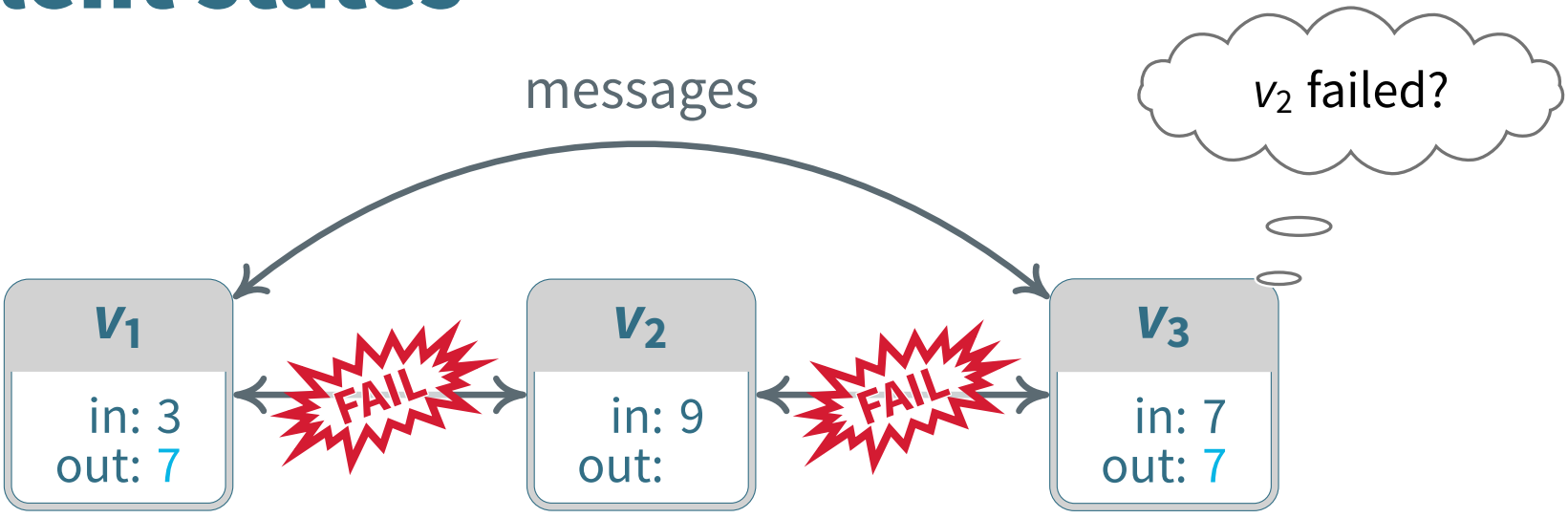
If protocol fault tolerant, v_1 and v_3 might decide to output 7

Once network back, v_2 must also output 7

Definition (Bivalent)

An execution of a consensus protocol is in a bivalent state when the network can affect which value agents choose.

Bivalent states



Recall agents chose value 9 in last example

But a network outage might be indistinguishable from v_2 failing

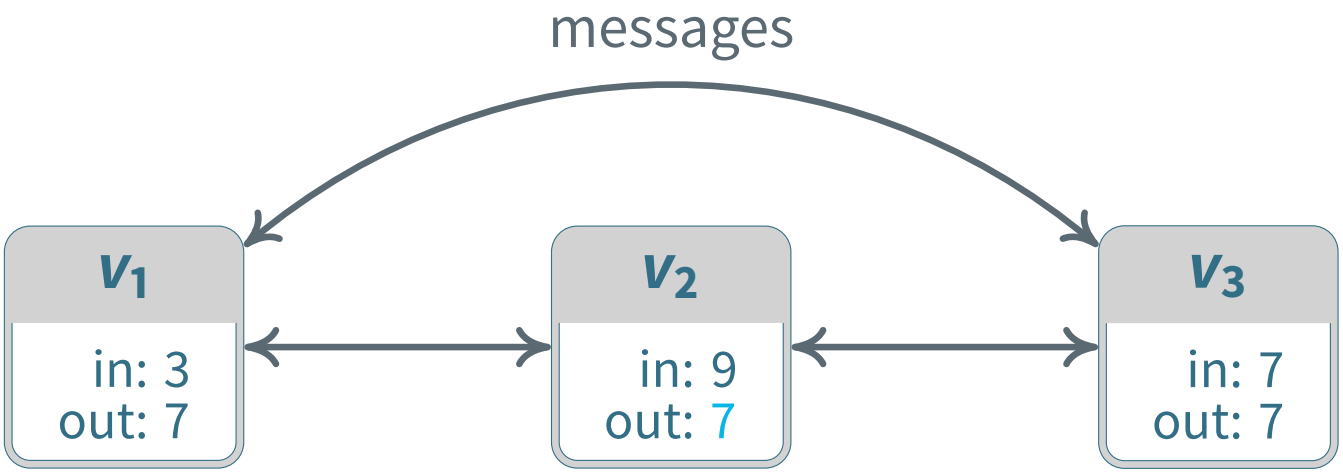
If protocol fault tolerant, v_1 and v_3 might decide to output 7

Once network back, v_2 must also output 7

Definition (Bivalent)

An execution of a consensus protocol is in a bivalent state when the network can affect which value agents choose.

Bivalent states



Recall agents chose value 9 in last example

But a network outage might be indistinguishable from v_2 failing

If protocol fault tolerant, v_1 and v_3 might decide to output 7

Once network back, v_2 must also output 7

Definition (Bivalent)

An execution of a consensus protocol is in a bivalent state when the network can affect which value agents choose.

Univalent and stuck states

Definition (Univalent, Valent)

An execution of a consensus protocol is in a univalent state when only one output value is possible. If that value is *i*, call the state *i*-valent.

Definition (Stuck)

An execution of a [broken] consensus protocol is in a stuck state when one or more non-faulty nodes can never output a value.

Recall output is write once and all outputs must agree

- Hence, no output is possible in bivalent state

If an execution starts in a bivalent state and terminates, it must at some point reach a univalent state

FLP intuition

Consider a terminating execution of a bivalent system

Let *m* be the last message received in a bivalent state

- Since *m* caused transition to univalent state, call it *deciding message*

Suppose the network had delayed *m*

- Other messages could cause transitions to other bivalent states
- Then, receiving *m* might no longer lead to a univalent state
- In this case, we say *m* has been neutralized

Overview of FLP proof.

1. **There are bivalent starting configurations.**
2. **Fault tolerance $\Longrightarrow$ network can neutralize any deciding msg**
3. **Hence, the system can remain bivalent in perpetuity.** □

How can we cope? Design protocols that terminate *in practice*

- Key property: protocol must avoid stuck states

Straw man: Vote on consensus value

Suppose you have N nodes with fail-stop behavior

Pick a quorum size $T > N/2$

If any T nodes (a quorum) all vote for a value, output that value

- E.g., Quorum A unanimously votes for 9, okay to output 9
- Any two quorums intersect, votes cannot change $\Longrightarrow$ agreement

Problem: stuck states

- Node failure could mean not everyone learns of unanimous quorum
- Split vote could make unanimous quorum impossible

Straw man: Vote on consensus value

Suppose you have N nodes with fail-stop behavior

Pick a quorum size $T > N/2$

If any T nodes (a quorum) all vote for a value, output that value

- E.g., Quorum A unanimously votes for 9, okay to output 9
- Any two quorums intersect, votes cannot change $\Longrightarrow$ agreement

Problem: stuck states

- Node failure could mean not everyone learns of unanimous quorum
- Split vote could make unanimous quorum impossible

Straw man: Vote on consensus value

Suppose you have N nodes with fail-stop behavior

Pick a quorum size $T > N/2$

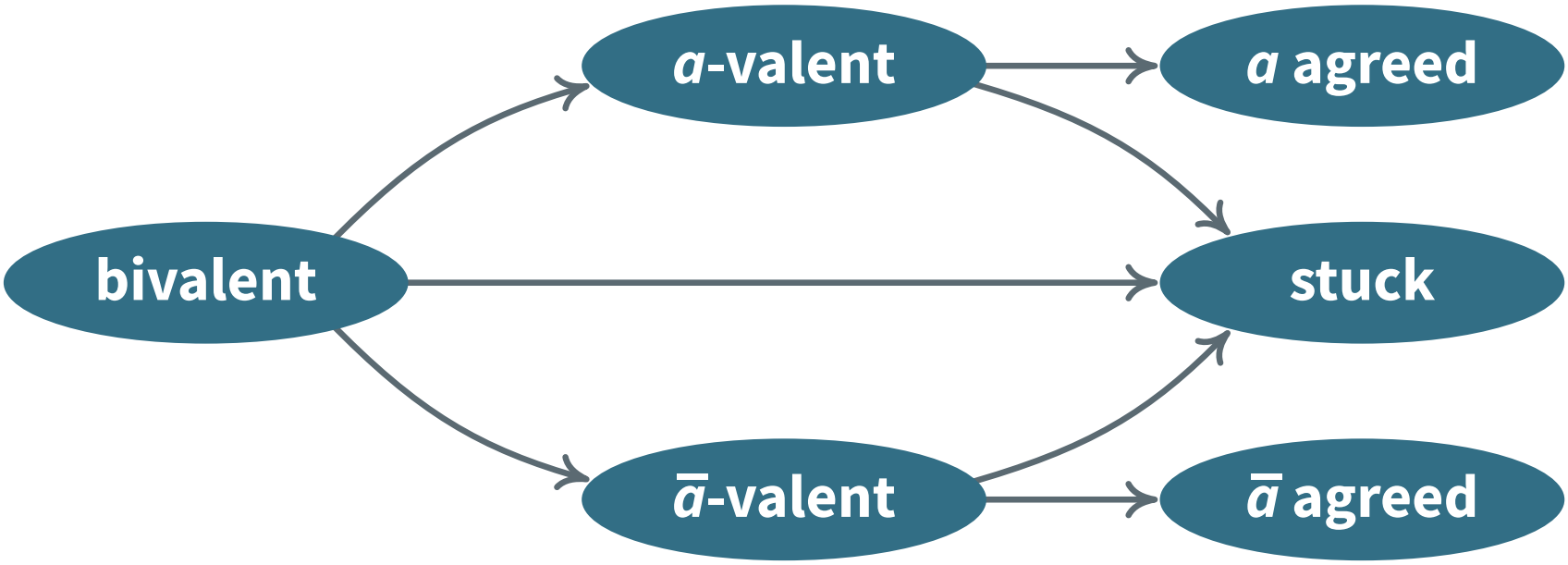
If any T nodes (a quorum) all vote for a value, output that value

- E.g., Quorum A unanimously votes for 9, okay to output 9
- Any two quorums intersect, votes cannot change $\Longrightarrow$ agreement

Problem: stuck states

- Node failure could mean not everyone learns of unanimous quorum
- Split vote could make unanimous quorum impossible

What voting gives us



You might get system-wide agreement or you might get stuck

Can't vote directly on consensus question ($\text{op}_7 \stackrel{?}{=}$ credit dm \$150)

What can we vote on without jeopardizing liveness?

1. Statements that never get stuck, and
2. Statements whose hold on consensus question can be broken if stuck

1. Statements that never get stuck

- Observation: stuck states arise because nodes can't change votes
- If nobody votes against a statement, can't get stuck

Definition (irrefutable)

An irrefutable statement is one that correct nodes never vote against.

2. Statements whose hold on consensus question can be broken

- Recall fault tolerance requires neutralizing deciding messages

Definition (neutralizable)

A neutralizable statement is one that can be rendered irrelevant to the consensus question.

How to formulate useful yet neutralizable statements?

Ballot-based neutralization [Paxos]

Vote to *commit* or *abort* ballots (the two are contradictory)

- Each ballot is $\langle n, x \rangle$ where n is a counter and x a candidate output value
- If a quorum votes to commit $\langle n, x \rangle$ (for any n), safe to output x

Invariant: all committed and stuck ballots must have same x

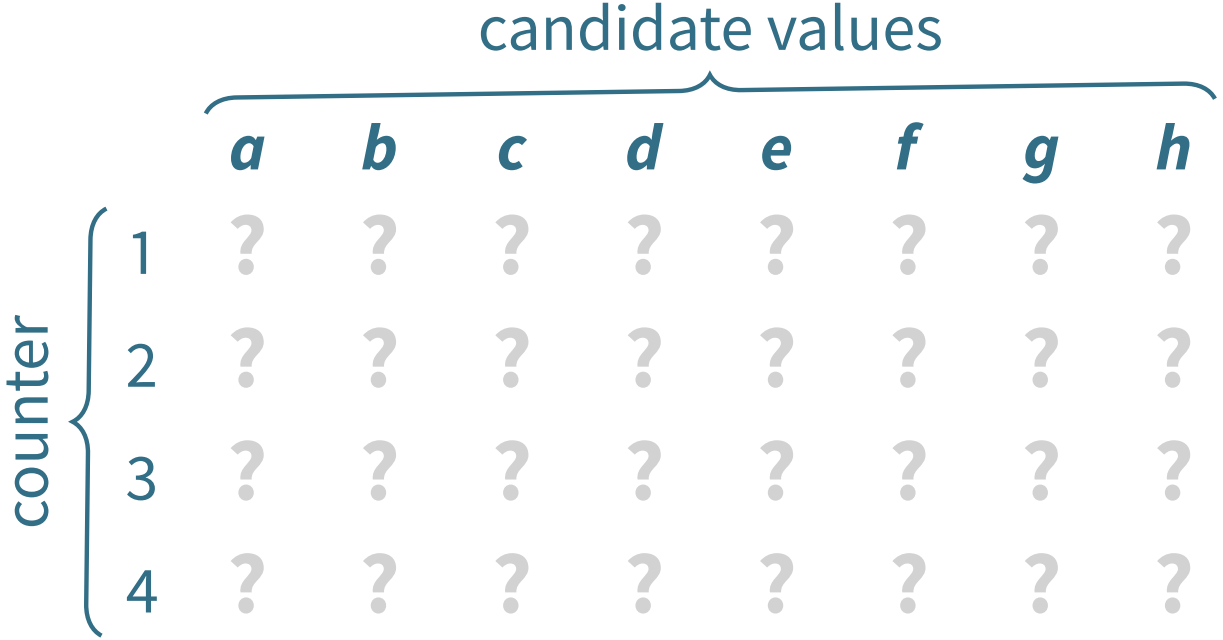
To preserve: cannot vote to commit a ballot before *preparing* it

- Prepare $\langle n, x \rangle$ by aborting all $\langle n', x' \rangle$ with $n' \leq n$ and $x' \neq x$.
- Concisely encode whole set of abort votes with PREPARE message

If ballot $\langle n, x \rangle$ stuck, neutralize by restarting with $\langle n + 1, x \rangle$

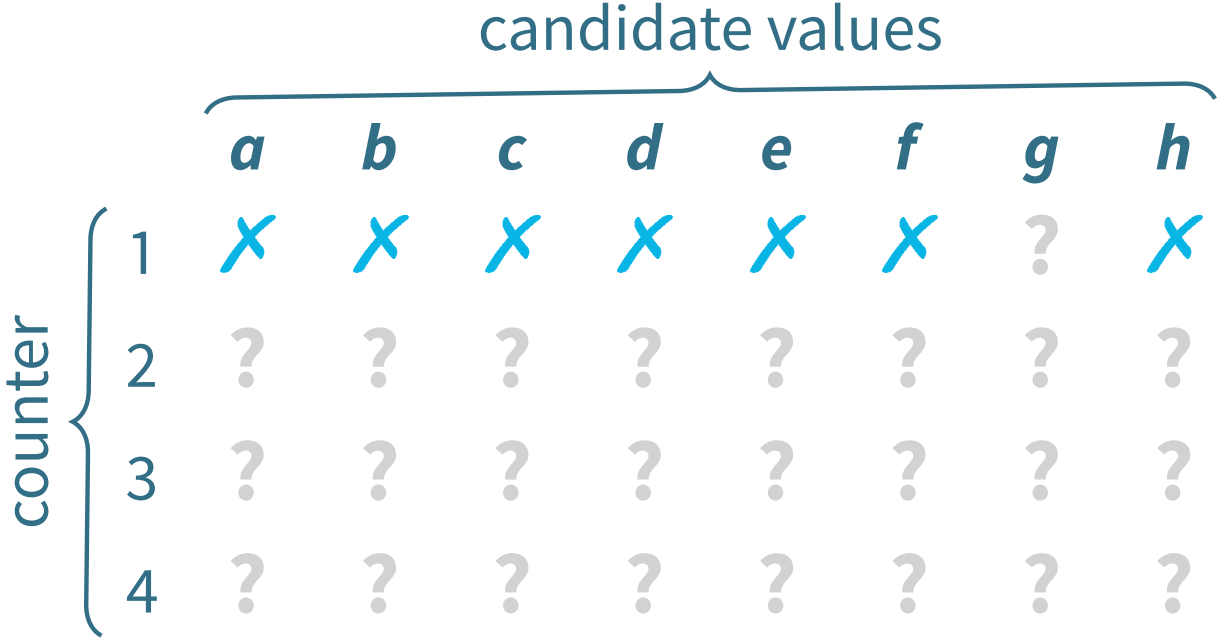
- Can prepare $\langle n + 1, x \rangle$ even if $\langle n, x \rangle$ is stuck

Paxos example



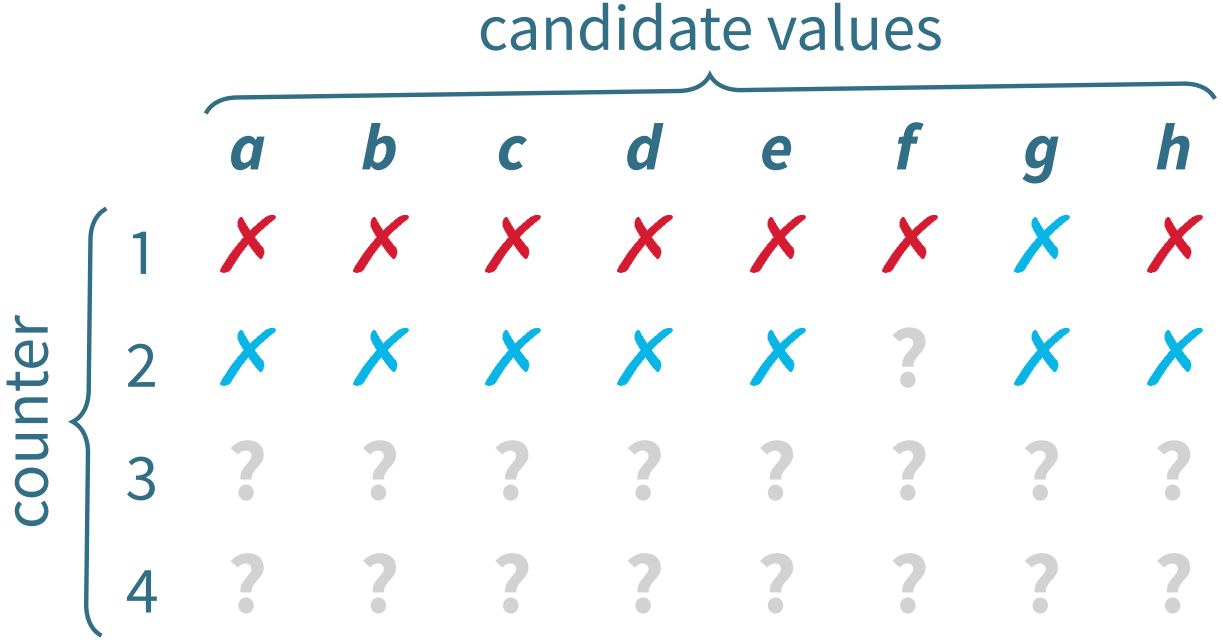
0. **Initially, all ballots are bivalent**
1. **Agree that $\langle 1, g \rangle$ is prepared and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ is stuck, so agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **See T votes to commit $\langle 3, f \rangle$ (commit-valent) and externalize f**
   - At this point nobody cares about $\langle 2, f \rangle$—neutralized
5. **Node failure makes $\langle 3, f \rangle$ stuck, prepare and commit $\langle 4, f \rangle$**

Paxos example



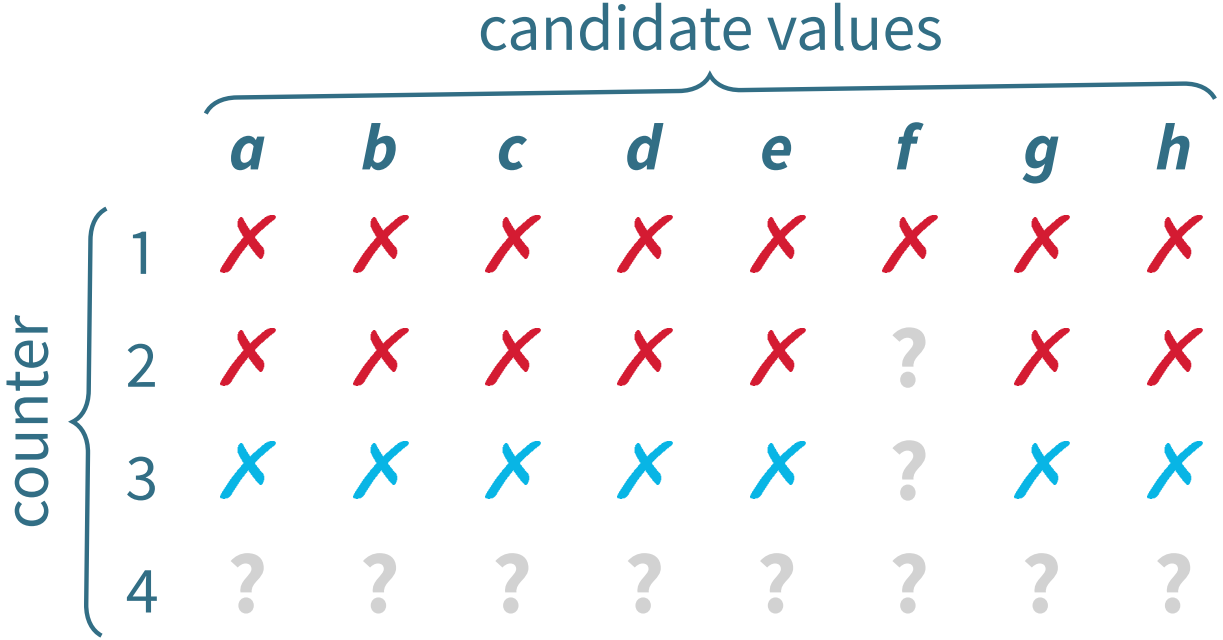
0. **Initially, all ballots are bivalent**
1. **Agree that $\langle 1, g \rangle$ is prepared and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ is stuck, so agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **See T votes to commit $\langle 3, f \rangle$ (commit-valent) and externalize f**
   - At this point nobody cares about $\langle 2, f \rangle$—neutralized
5. **Node failure makes $\langle 3, f \rangle$ stuck, prepare and commit $\langle 4, f \rangle$**

Paxos example



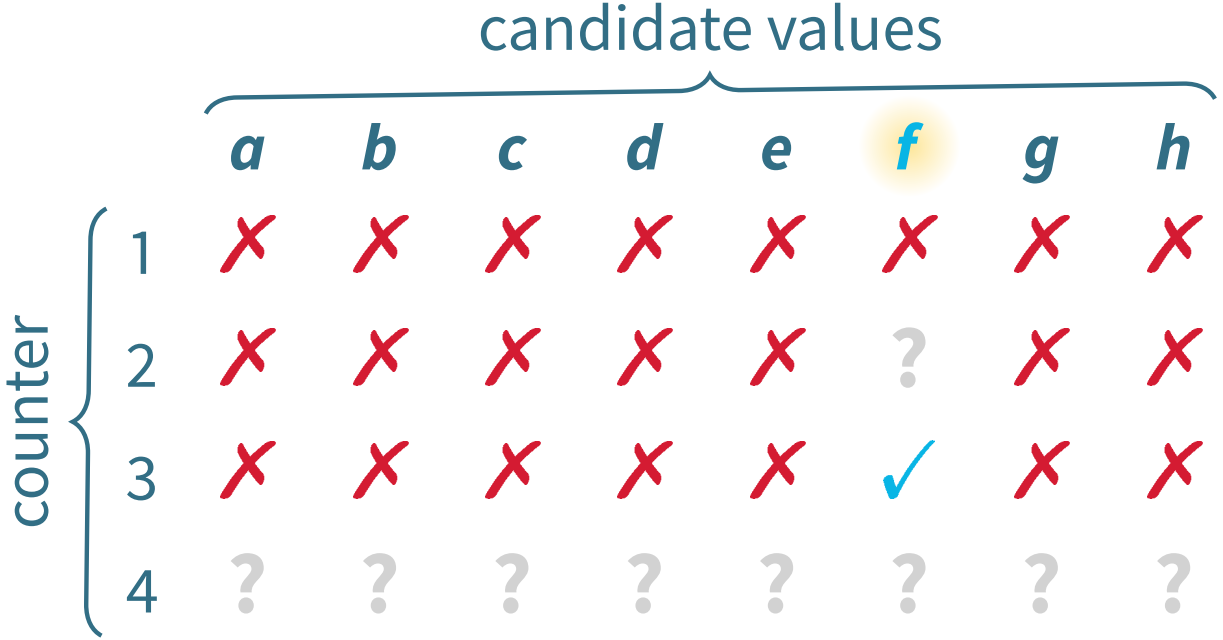
0. **Initially, all ballots are bivalent**
1. **Agree that $\langle 1, g \rangle$ is prepared and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ is stuck, so agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **See T votes to commit $\langle 3, f \rangle$ (commit-valent) and externalize f**
   - At this point nobody cares about $\langle 2, f \rangle$—neutralized
5. **Node failure makes $\langle 3, f \rangle$ stuck, prepare and commit $\langle 4, f \rangle$**

Paxos example



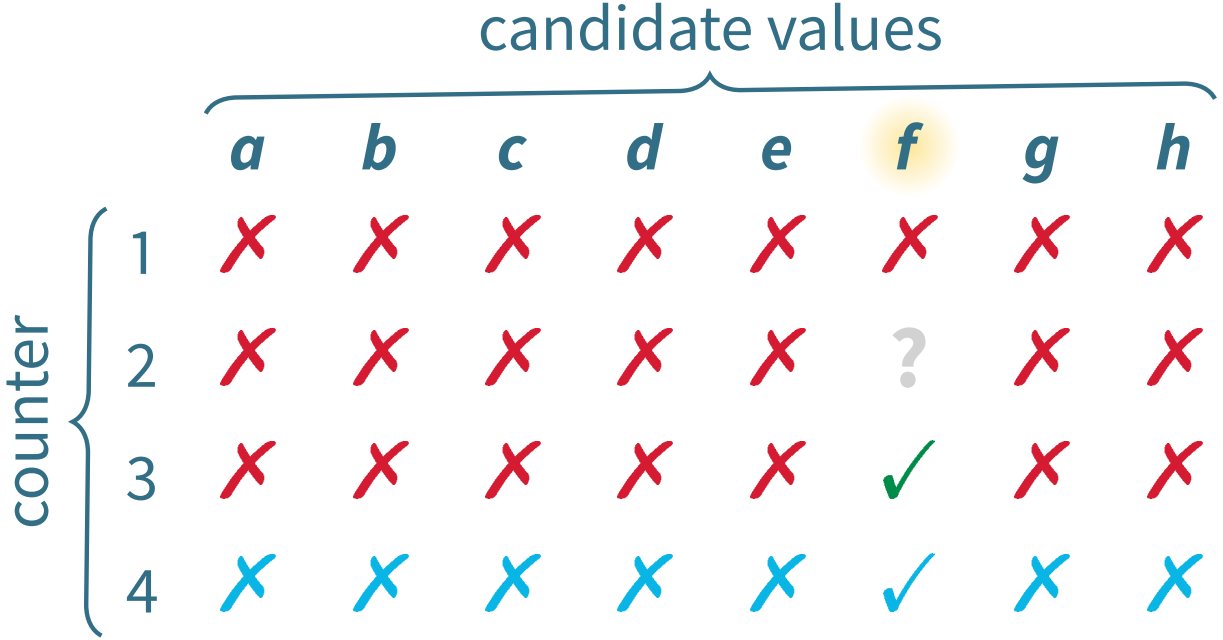
0. **Initially, all ballots are bivalent**
1. **Agree that $\langle 1, g \rangle$ is prepared and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ is stuck, so agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **See T votes to commit $\langle 3, f \rangle$ (commit-valent) and externalize f**
   - At this point nobody cares about $\langle 2, f \rangle$—neutralized
5. **Node failure makes $\langle 3, f \rangle$ stuck, prepare and commit $\langle 4, f \rangle$**

Paxos example



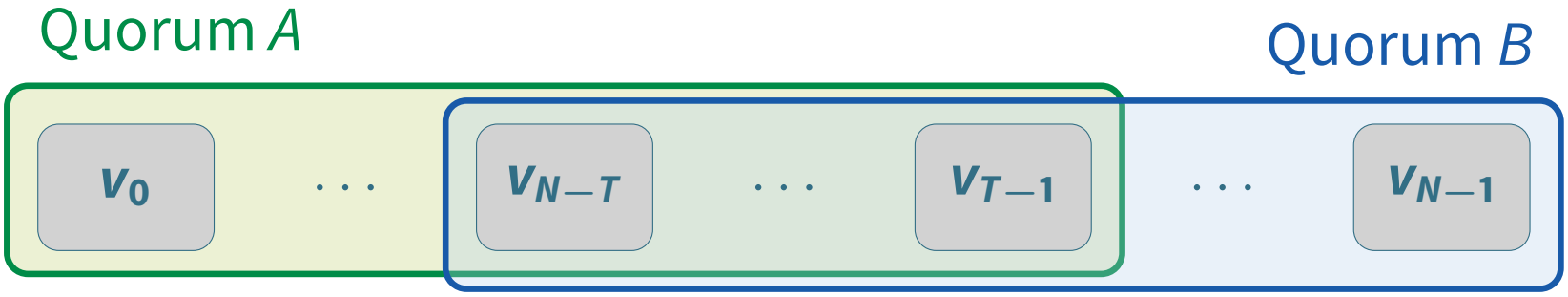
0. **Initially, all ballots are bivalent**
1. **Agree that $\langle 1, g \rangle$ is prepared and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ is stuck, so agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **See T votes to commit $\langle 3, f \rangle$ (commit-valent) and externalize f**
   - At this point nobody cares about $\langle 2, f \rangle$—neutralized
5. **Node failure makes $\langle 3, f \rangle$ stuck, prepare and commit $\langle 4, f \rangle$**

Paxos example



0. **Initially, all ballots are bivalent**
1. **Agree that $\langle 1, g \rangle$ is prepared and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ is stuck, so agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **See T votes to commit $\langle 3, f \rangle$ (commit-valent) and externalize f**
   - At this point nobody cares about $\langle 2, f \rangle$—neutralized
5. **Node failure makes $\langle 3, f \rangle$ stuck, prepare and commit $\langle 4, f \rangle$**

Byzantine agreement



What if nodes may experience Byzantine failure?

- Byzantine nodes can illegally change their votes
- In fail-stop case, safety required any two quorums to share a node
- Now, any two quorums to share a *non-faulty* node

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

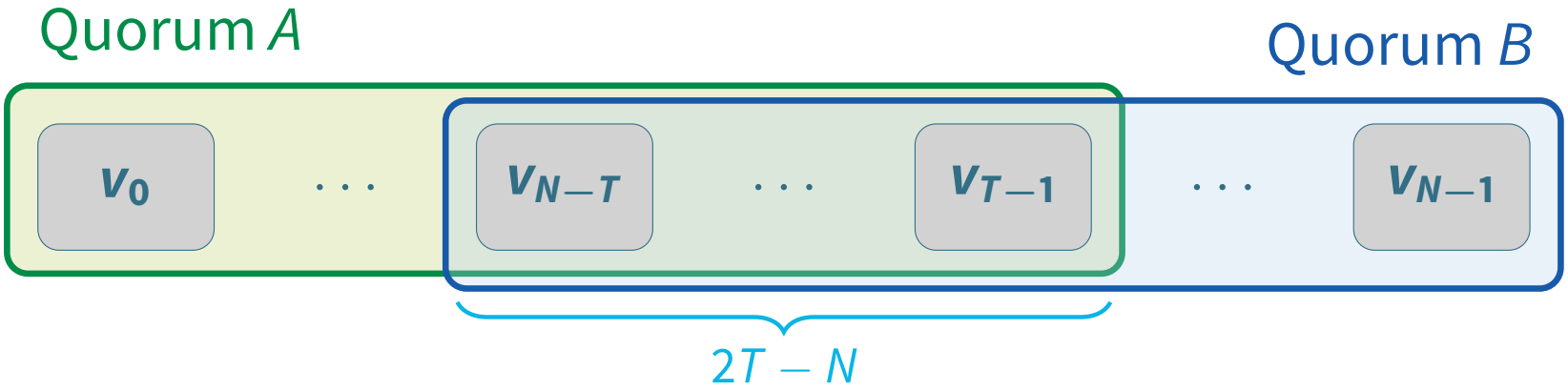
***Liveness* requires: # failures** $\leq f_L = N - T$

- At least one entirely non-faulty quorum exists

Longstanding practical protocols exist [Castro'99]

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

Byzantine agreement



What if nodes may experience Byzantine failure?

- Byzantine nodes can illegally change their votes
- In fail-stop case, safety required any two quorums to share a node
- Now, any two quorums to share a *non-faulty* node

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

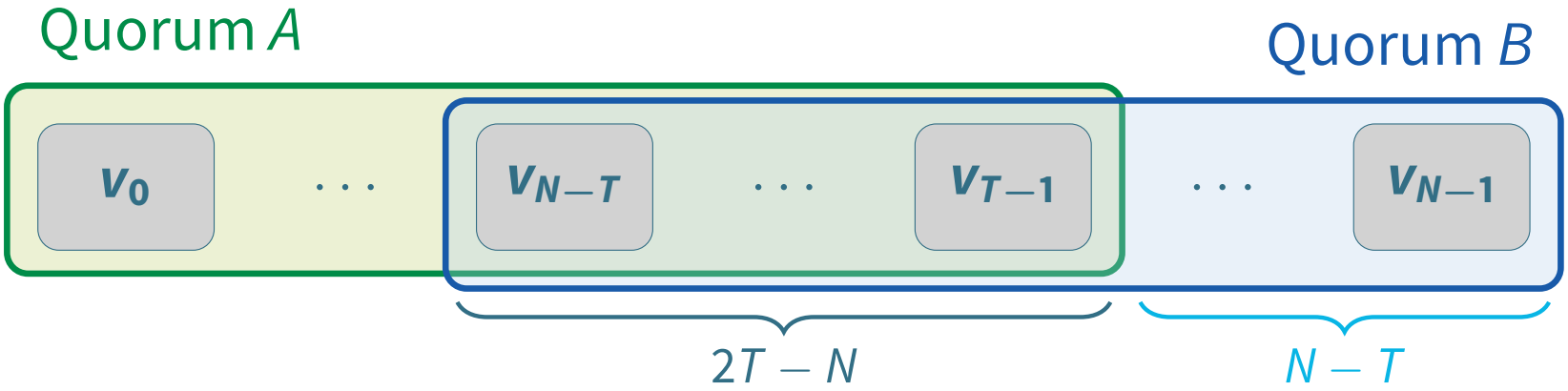
***Liveness* requires: # failures** $\leq f_L = N - T$

- At least one entirely non-faulty quorum exists

Longstanding practical protocols exist [Castro'99]

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

Byzantine agreement



What if nodes may experience Byzantine failure?

- Byzantine nodes can illegally change their votes
- In fail-stop case, safety required any two quorums to share a node
- Now, any two quorums to share a *non-faulty* node

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

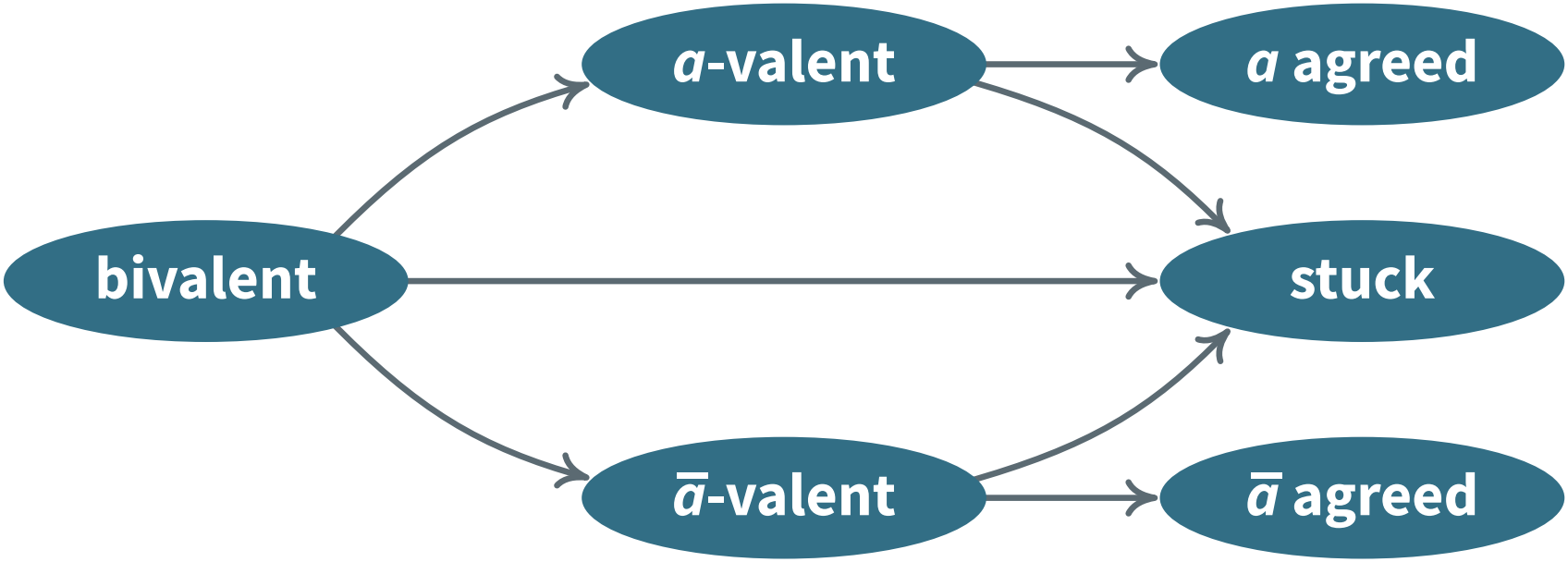
***Liveness* requires: # failures** $\leq f_L = N - T$

- At least one entirely non-faulty quorum exists

Longstanding practical protocols exist [Castro'99]

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

When has a vote succeeded?



If $f_S + 1 = 2T - N$ nodes malicious, system loses safety

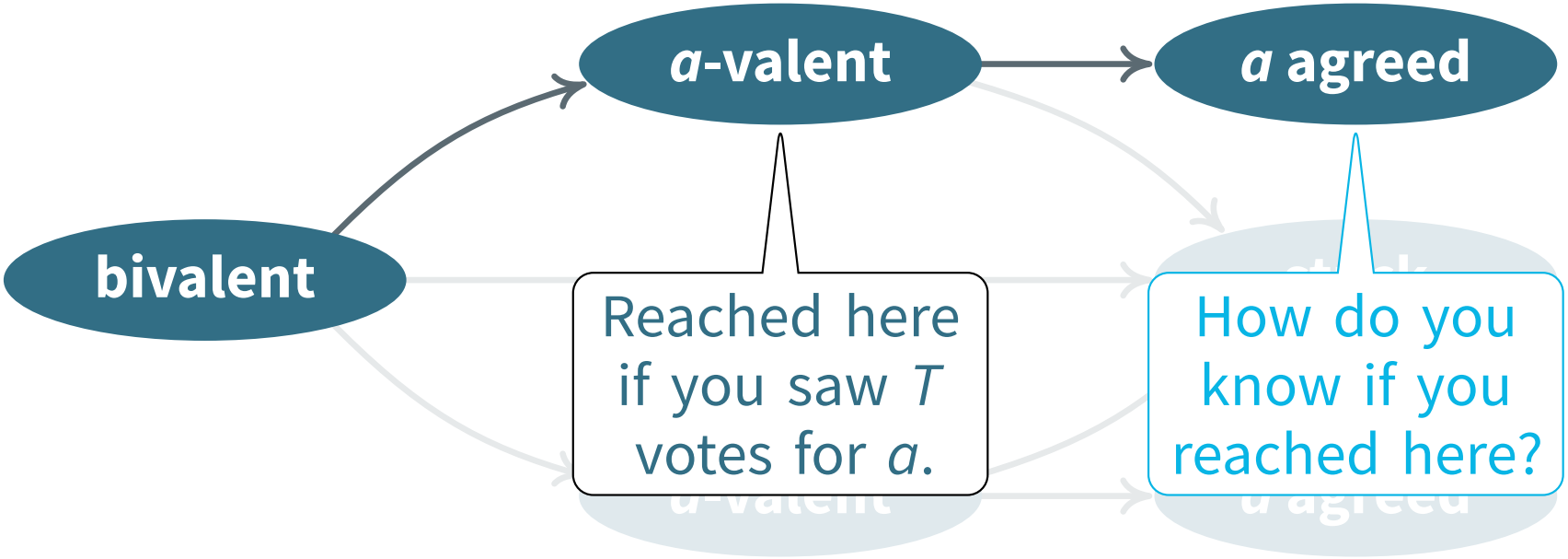
Suppose $f_S + 1$ nodes all claim to have seen T votes for a

- Can assume system is a-valent with no loss of safety

Now say $f_L + f_S + 1 = T$ nodes all make same assertion

- If $> f_L$ fail, system loses liveness (0 correct nodes in whole system)
- If $\leq f_L$ fail, $\geq f_S + 1$ remain able to convince rest that system a-valent
- All correct nodes believe system a-valent $\implies$ none stuck $\implies$ a agreed

When has a vote succeeded?



If $f_S + 1 = 2T - N$ nodes malicious, system loses safety

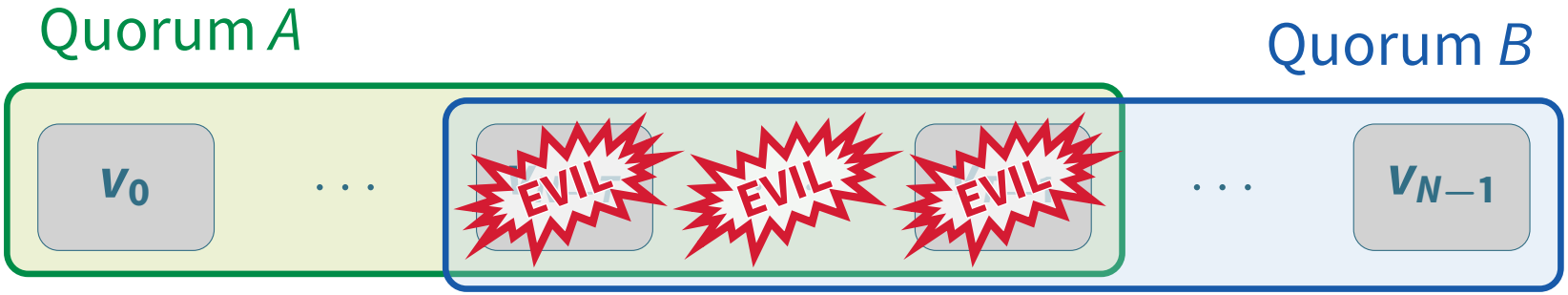
Suppose $f_S + 1$ nodes all claim to have seen T votes for a

- Can assume system is a-valent with no loss of safety

Now say $f_L + f_S + 1 = T$ nodes all make same assertion

- If $> f_L$ fail, system loses liveness (0 correct nodes in whole system)
- If $\leq f_L$ fail, $\geq f_S + 1$ remain able to convince rest that system a-valent
- All correct nodes believe system a-valent $\Longrightarrow$ none stuck $\Longrightarrow$ a agreed

When has a vote succeeded?



If $f_S + 1 = 2T - N$ nodes malicious, system loses safety

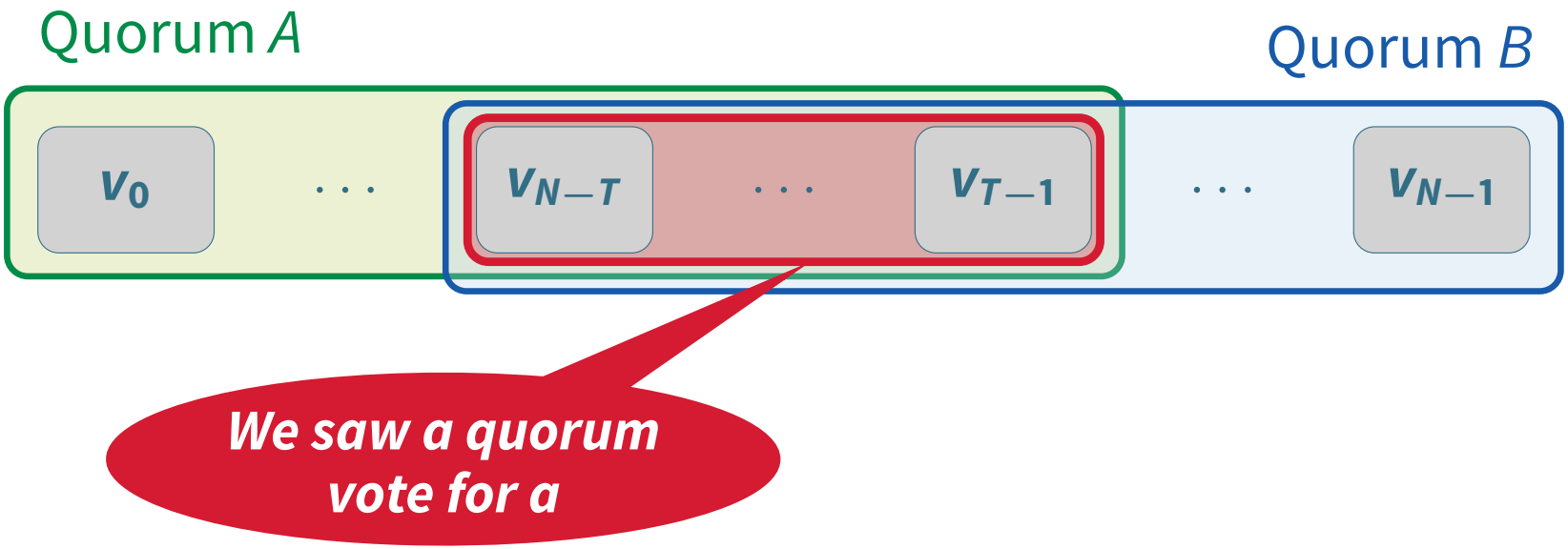
Suppose $f_S + 1$ nodes all claim to have seen T votes for a

- Can assume system is a-valent with no loss of safety

Now say $f_L + f_S + 1 = T$ nodes all make same assertion

- If $> f_L$ fail, system loses liveness (0 correct nodes in whole system)
- If $\leq f_L$ fail, $\geq f_S + 1$ remain able to convince rest that system a-valent
- All correct nodes believe system a-valent $\implies$ none stuck $\implies$ a agreed

When has a vote succeeded?



If $f_S + 1 = 2T - N$ nodes malicious, system loses safety

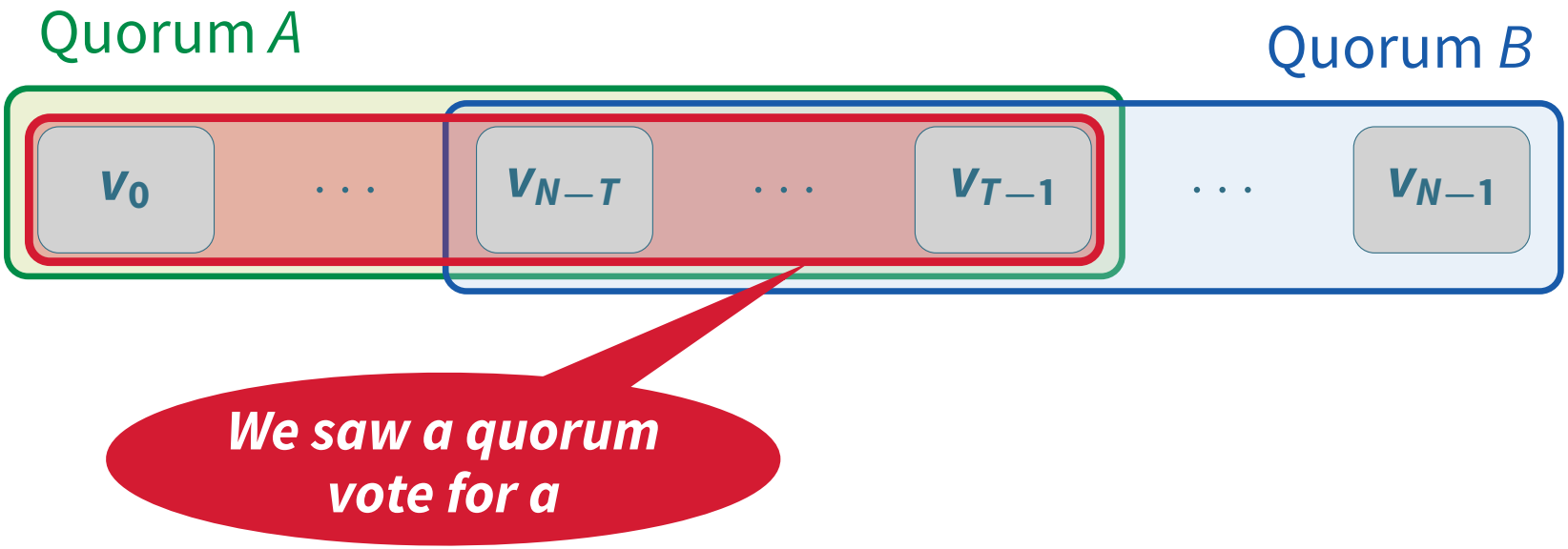
Suppose $f_S + 1$ nodes all claim to have seen T votes for a

- Can assume system is a-valent with no loss of safety

Now say $f_L + f_S + 1 = T$ nodes all make same assertion

- If $> f_L$ fail, system loses liveness (0 correct nodes in whole system)
- If $\leq f_L$ fail, $\geq f_S + 1$ remain able to convince rest that system a-valent
- All correct nodes believe system a-valent $\implies$ none stuck $\implies$ a agreed

When has a vote succeeded?



If $f_S + 1 = 2T - N$ nodes malicious, system loses safety

Suppose $f_S + 1$ nodes all claim to have seen T votes for a

- Can assume system is a-valent with no loss of safety

Now say $f_L + f_S + 1 = T$ nodes all make same assertion

- If $> f_L$ fail, system loses liveness (0 correct nodes in whole system)
- If $\leq f_L$ fail, $\geq f_S + 1$ remain able to convince rest that system a-valent
- All correct nodes believe system a-valent $\implies$ none stuck $\implies$ a agreed

Outline

Byzantine agreement (background), recast in terms of voting on irrefutable & neutralizable statements

Federated Byzantine Agreement (FBA)

Optimal failure resilience

The Stellar consensus protocol (SCP)

Federated Byzantine Agreement (FBA)

FBA is a generalization of the Byzantine agreement problem

- Byzantine agreement without magically blessing N nodes

Idea: determine quorums in decentralized way

- Each node v picks one or more *quorum slices*, where v in all its slices
- v requires any quorum to contain at least one of its slices
- Slice inclusion analogous to Internet peering/transit between ISPs

Definition (Federated Byzantine Agreement System)

An FBAS is of a a set of nodes **V** and a quorum function **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

FBA is a generalization of the Byzantine agreement problem

- Byzantine agreement without magically blessing N nodes

Idea: determine quorums in decentralized way

- Each node v picks one or more *quorum slices*, where v in all its slices
- v requires any quorum to contain at least one of its slices
- Slice inclusion analogous to Internet peering/transit between ISPs

Definition (Federated Byzantine Agreement System)

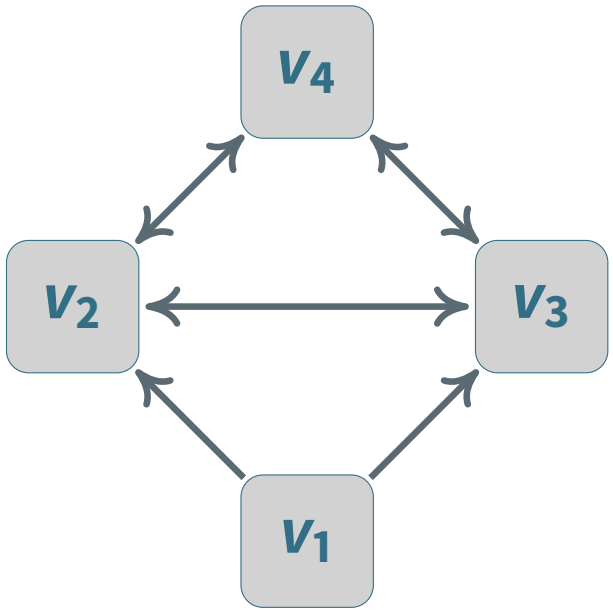
An FBAS is of a a set of nodes **V** and a quorum function **Q**, where $\mathbf{Q}(v)$ is the set slices chosen by node v.

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**

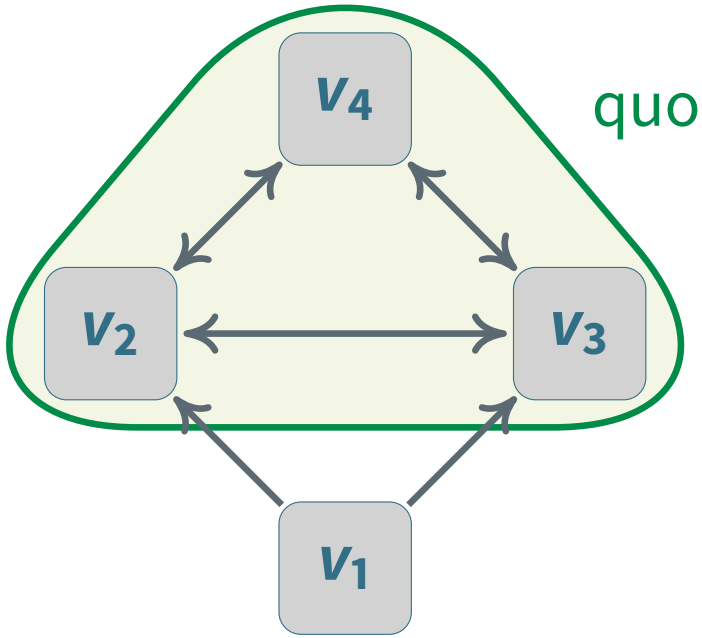
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- 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**

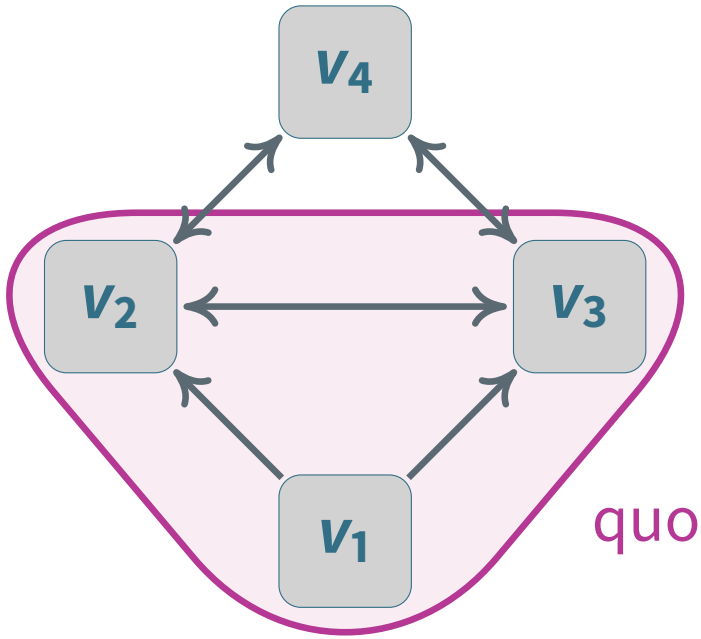
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- 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**

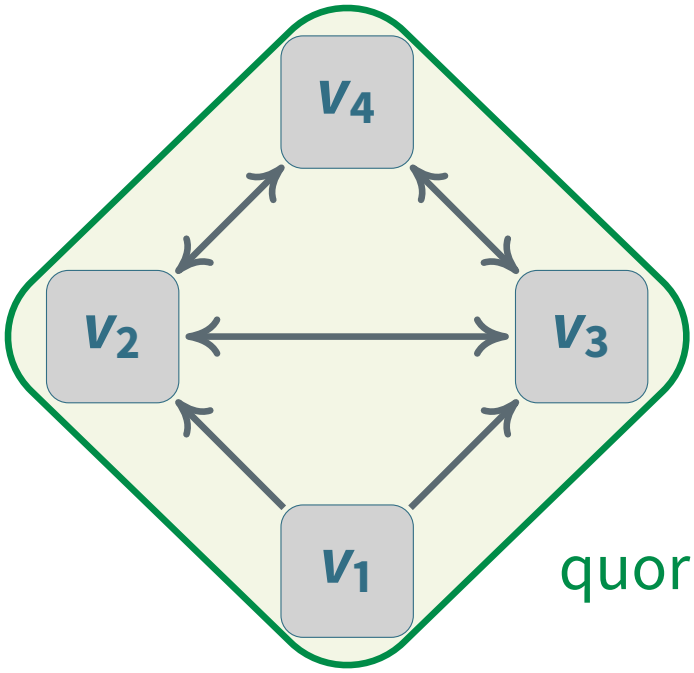
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- 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

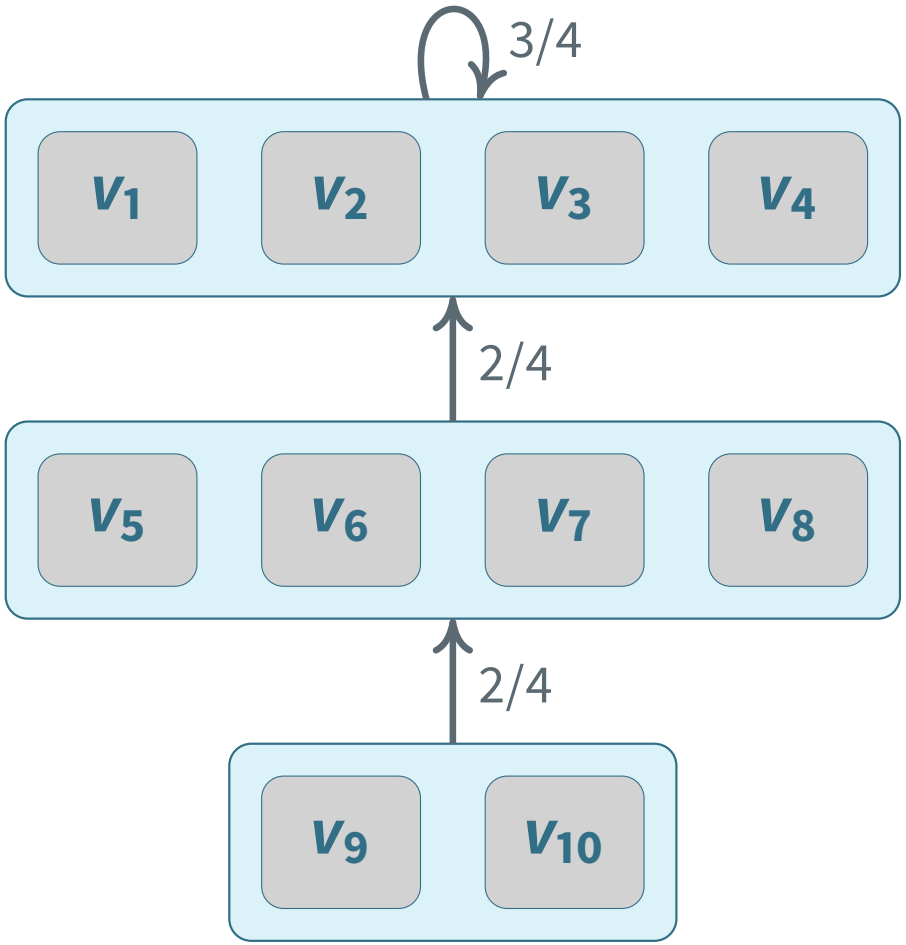
v_2, v_3, v_4 **is a quorum—contains a slice of each member**

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)

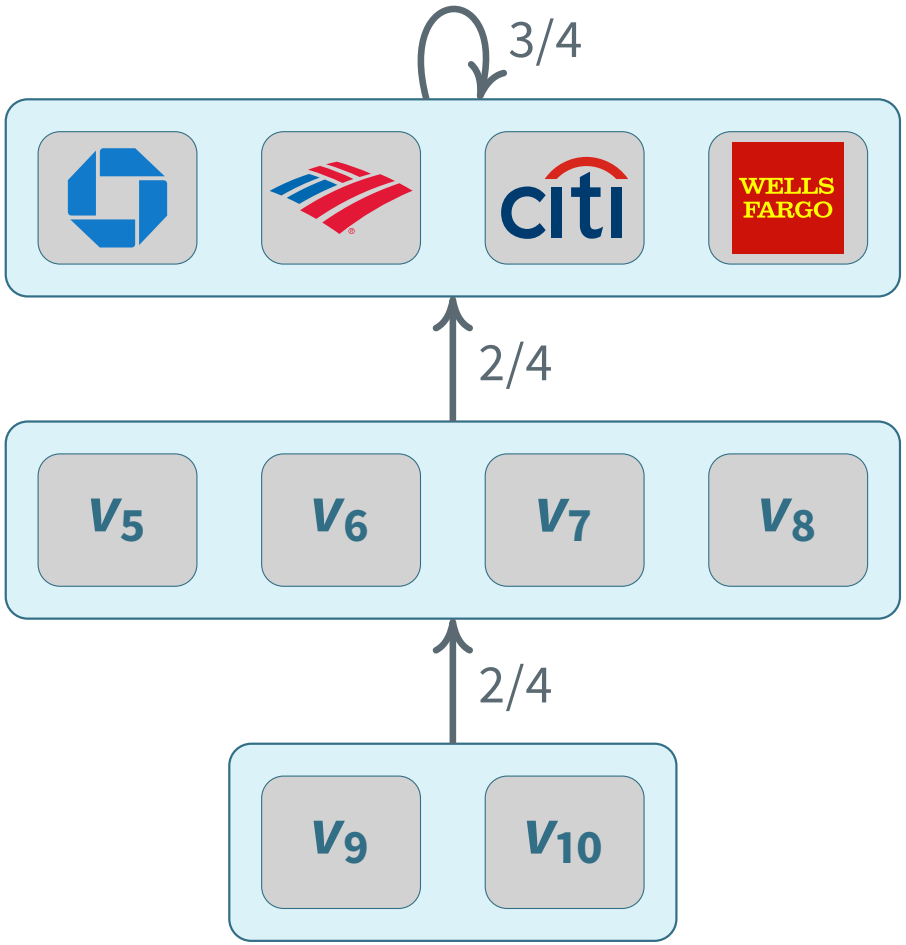
Middle tier: slice is self + any two top tier nodes

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

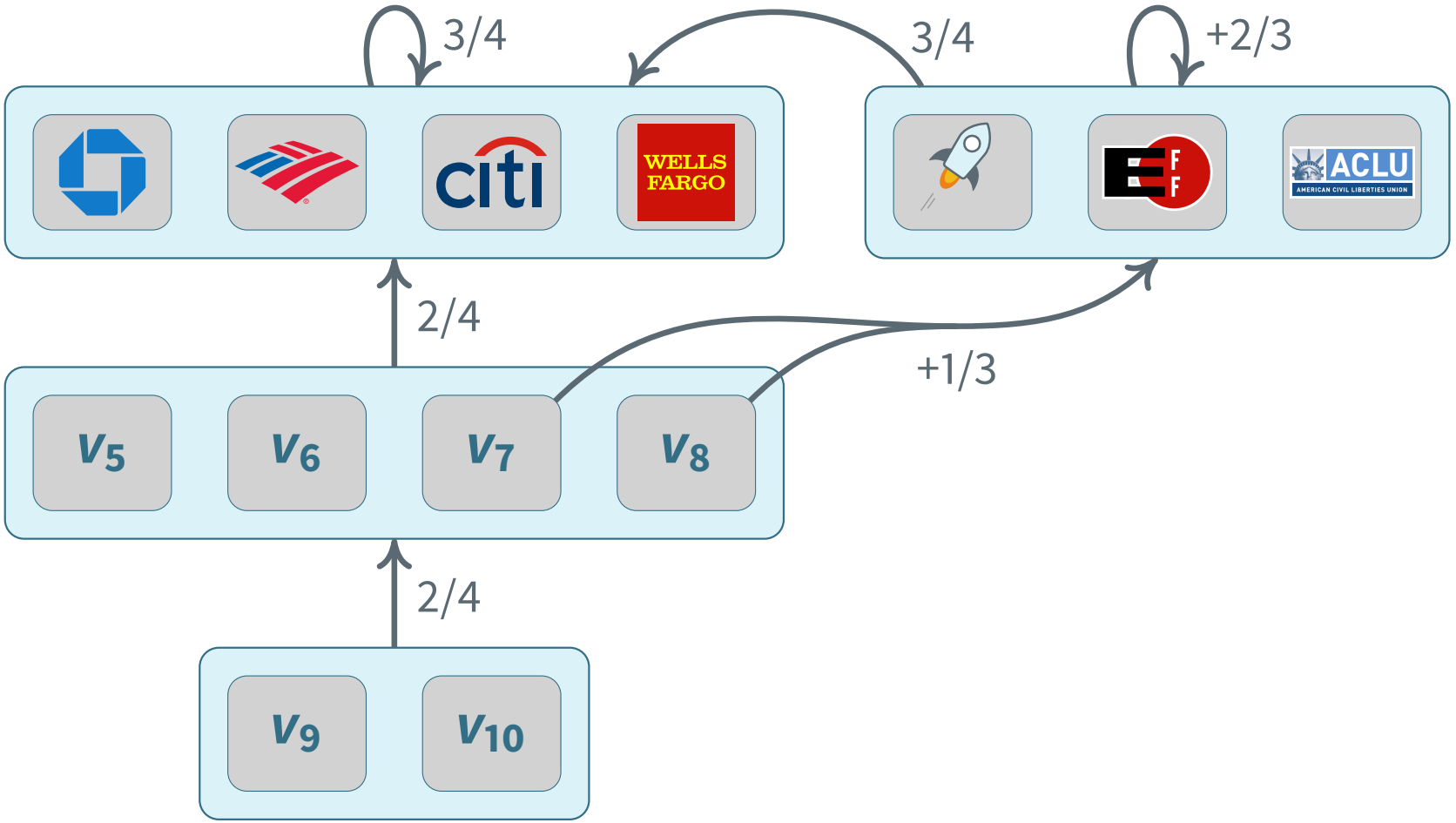
Tiered quorum slice example



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

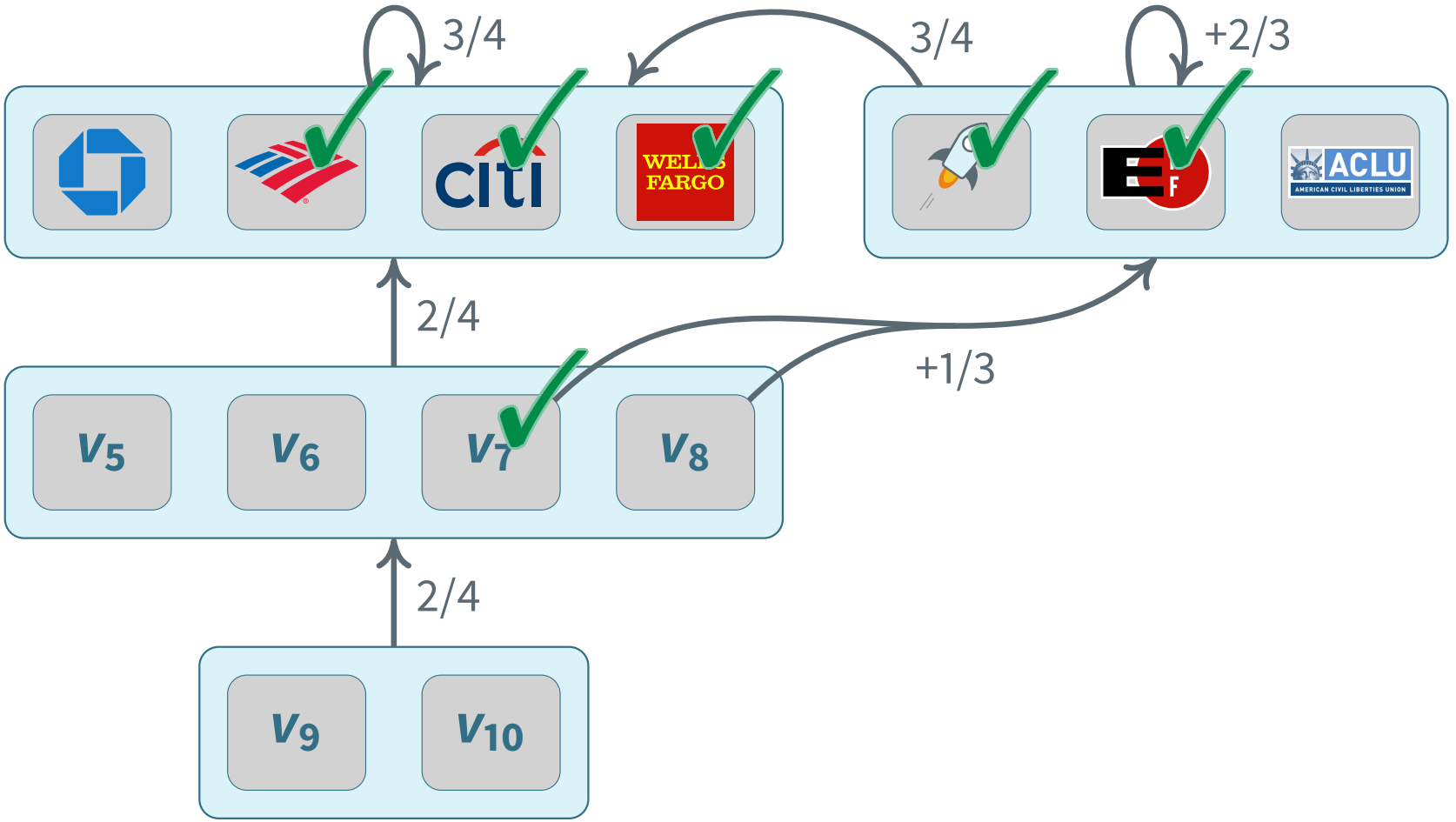
Tiered quorum slice example



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

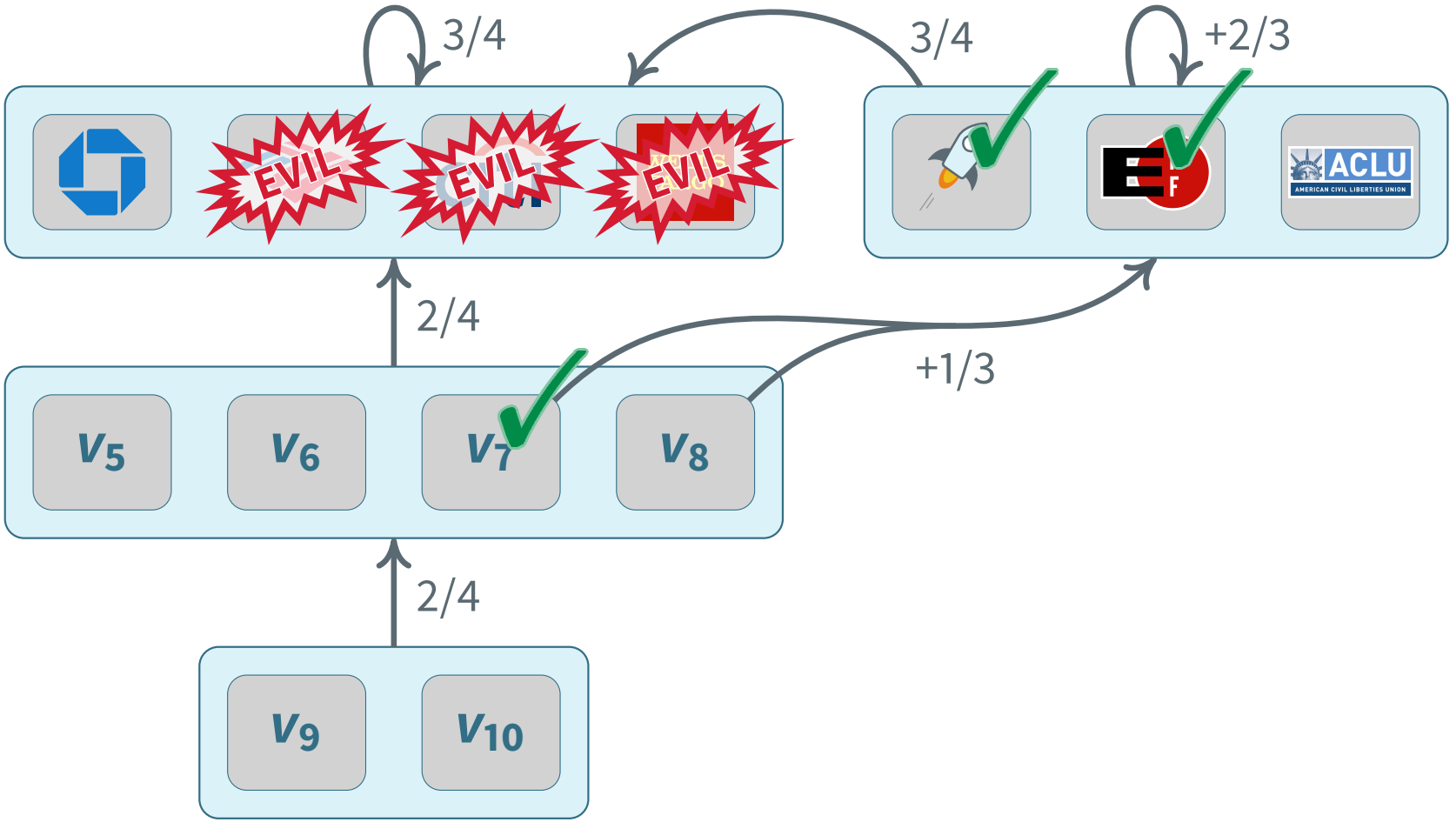
Tiered quorum slice example



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

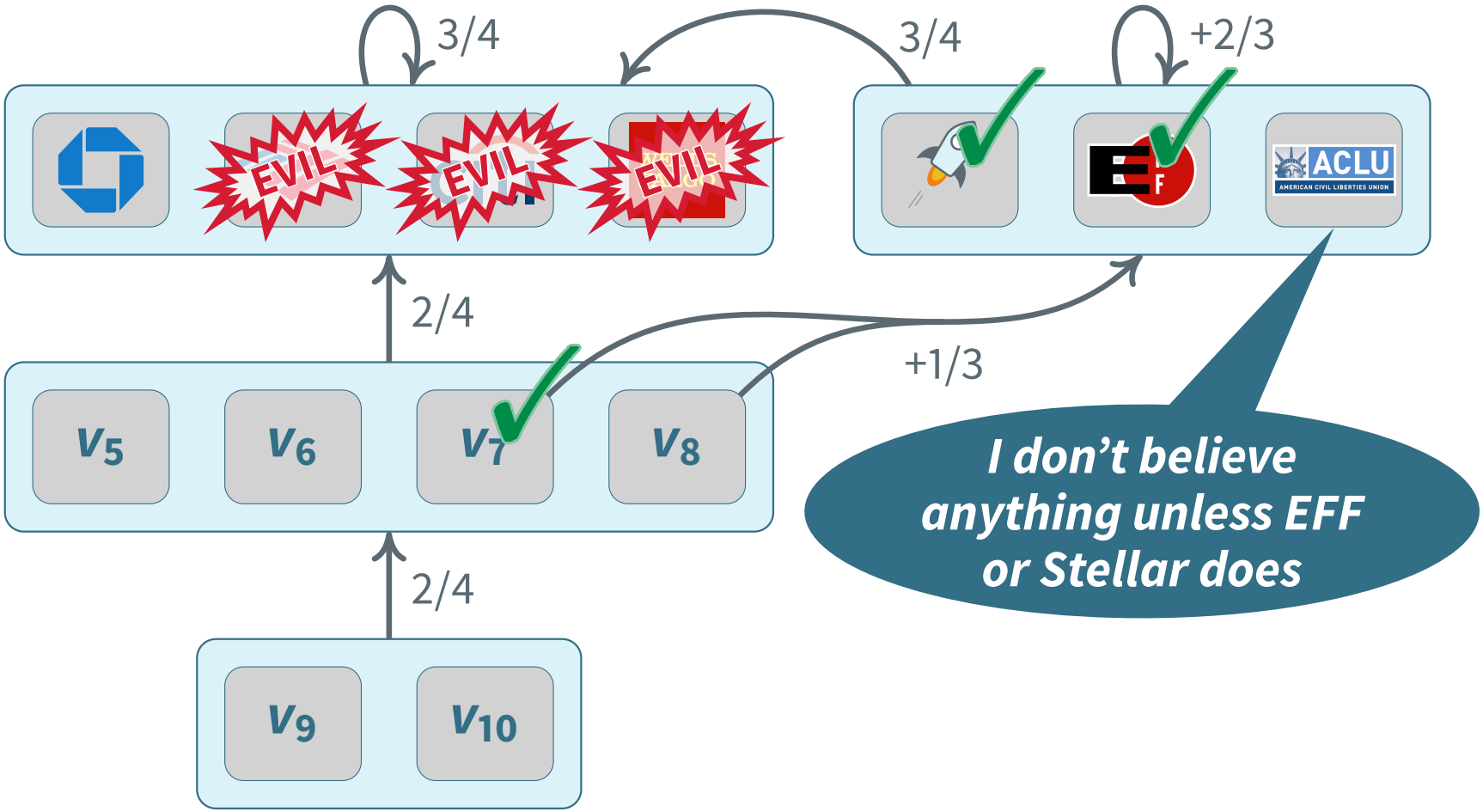
Tiered quorum slice example



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

Outline

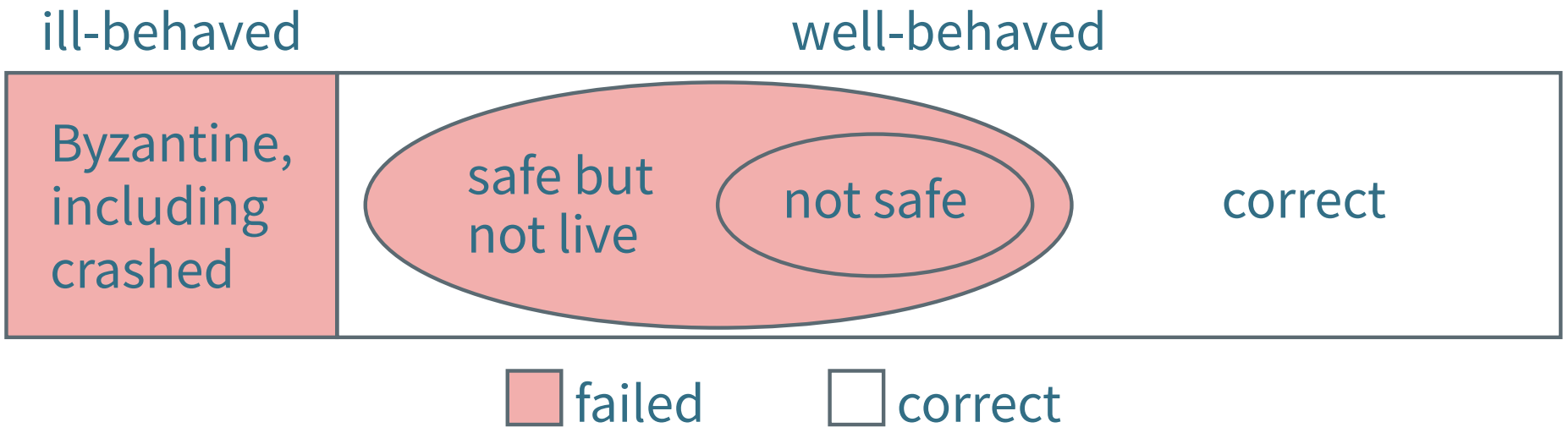
Byzantine agreement (background), recast in terms of voting on irrefutable & neutralizable statements

Federated Byzantine Agreement (FBA)

Optimal failure resilience

The Stellar consensus protocol (SCP)

FBAS failure is per-node



Each node is either *well-behaved* or *ill-behaved*

All ill-behaved nodes have *failed*

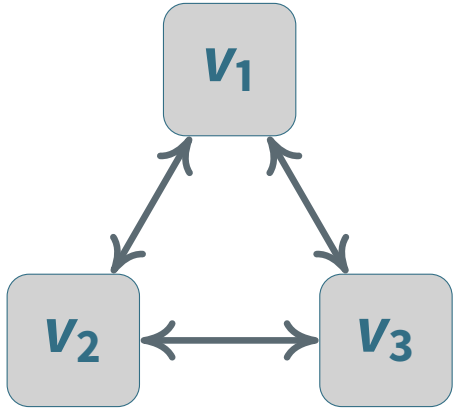
Enough ill-behaved nodes can cause well-behaved nodes to fail

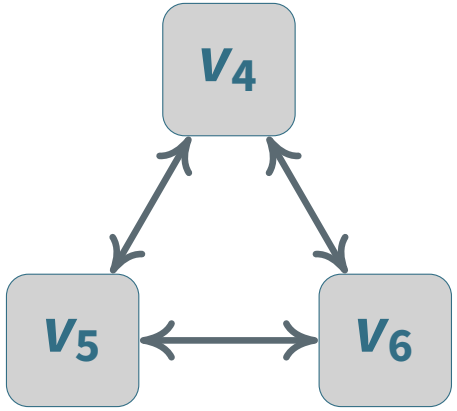
- Bad: well-behaved nodes blocked from any progress (safe but not live)
- Worse: well-behaved nodes in divergent states (not safe)

Well-behaved nodes are *correct* if they have not failed

What is necessary to guarantee safety?

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





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

Suppose there are two entirely disjoint quorums

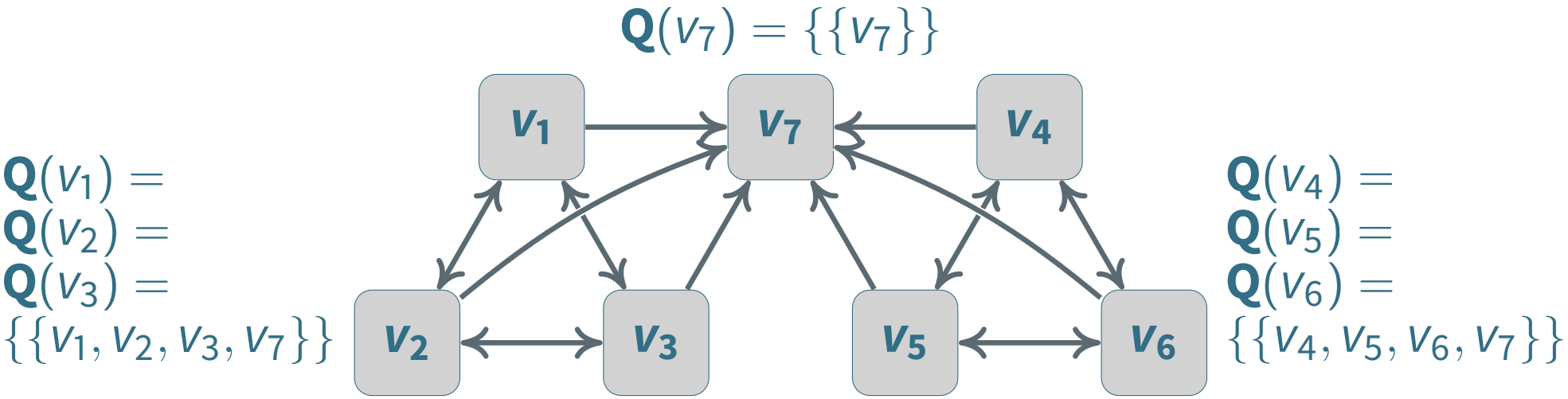
- Each can make progress with no communication from the other
- No way to guarantee the two externalize consistent statements

Like traditional consensus, safety requires quorum intersection

Definition (Quorum intersection)

An FBAS enjoys quorum intersection when every two quorums share at least one node.

What about Byzantine failures?



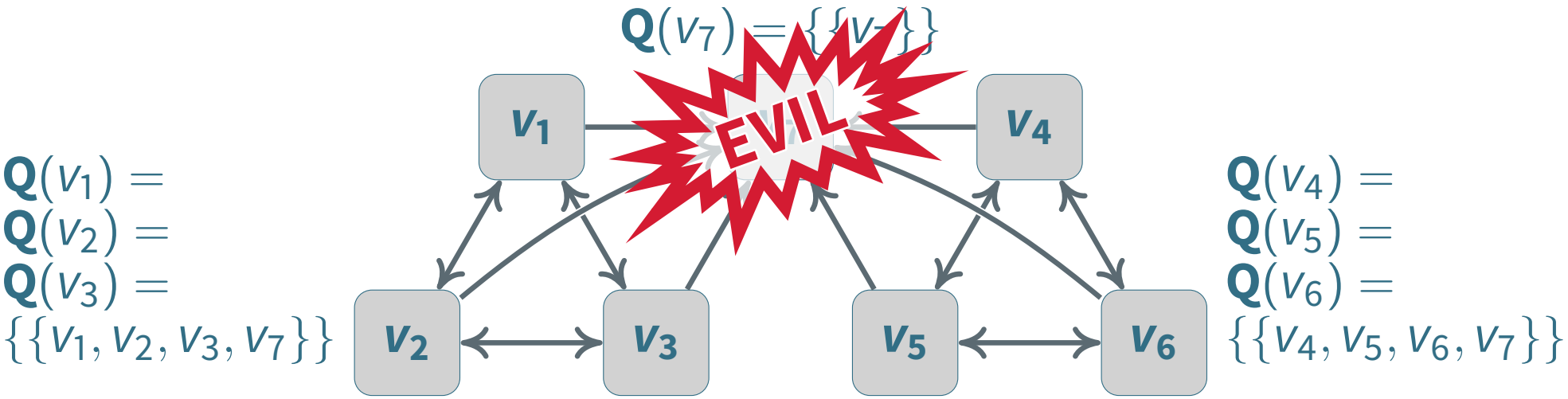
Suppose two quorums intersect only at Byzantine nodes

- Byzantine nodes behave arbitrarily
- Can feed inconsistent data to different honest nodes
- No way to guarantee safety

Necessary property for safety with Byzantine failures:
Quorum intersection *despite ill-behaved nodes*

- Means deleting ill-behaved nodes doesn't undermine intersection
- In this example, reduces to diagram on previous slide

What about Byzantine failures?



Suppose two quorums intersect only at Byzantine nodes

- Byzantine nodes behave arbitrarily
- Can feed inconsistent data to different honest nodes
- No way to guarantee safety

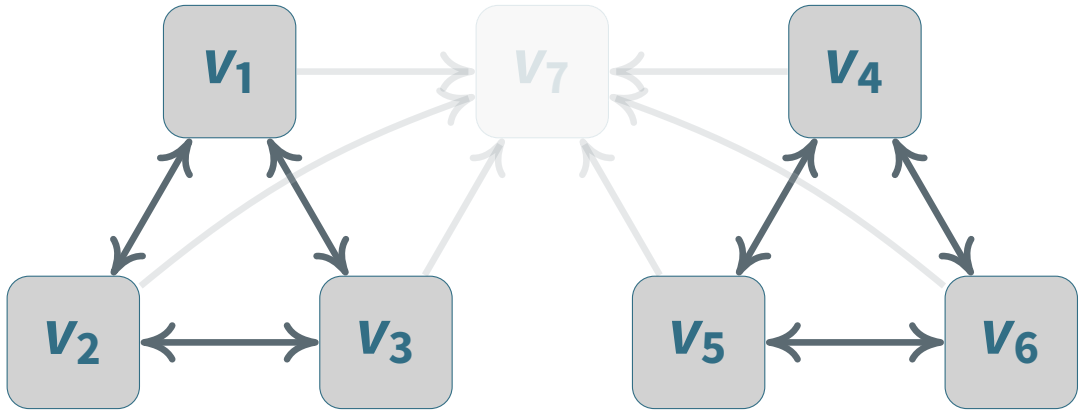
Necessary property for safety with Byzantine failures:
Quorum intersection *despite ill-behaved nodes*

- Means deleting ill-behaved nodes doesn't undermine intersection
- In this example, reduces to diagram on previous slide

What about Byzantine failures?

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

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



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

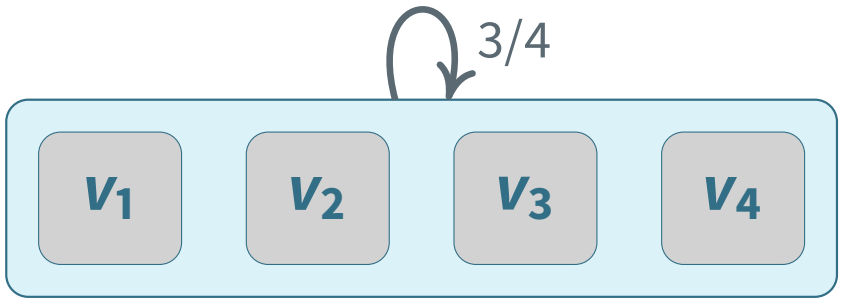
Suppose two quorums intersect only at Byzantine nodes

- Byzantine nodes behave arbitrarily
- Can feed inconsistent data to different honest nodes
- No way to guarantee safety

Necessary property for safety with Byzantine failures:
Quorum intersection *despite ill-behaved nodes*

- Means deleting ill-behaved nodes doesn't undermine intersection
- In this example, reduces to diagram on previous slide

What is necessary to guarantee liveness?



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

Suppose each of v_1's slices contains a Byzantine node

- Byzantine includes crashed—might not agree to anything
- Impossible to guarantee liveness for v_1

Definition (v-blocking)

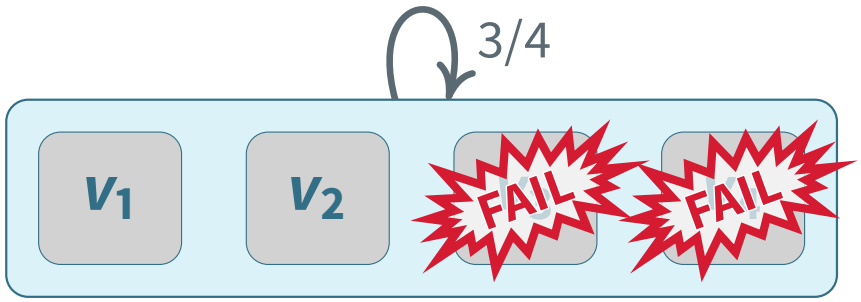
A v-blocking set contains at least one node from each of v's slices.

Failed nodes cannot be v-blocking for any correct node v

- Equivalent to saying correct nodes form a quorum

Necessary property for liveness: Correct nodes form a quorum

What is necessary to guarantee liveness?



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

Suppose each of v_1's slices contains a Byzantine node

- Byzantine includes crashed—might not agree to anything
- Impossible to guarantee liveness for v_1

Definition (v-blocking)

A v-blocking set contains at least one node from each of v's slices.

Failed nodes cannot be v-blocking for any correct node v

- Equivalent to saying correct nodes form a quorum

Necessary property for liveness: Correct nodes form a quorum

Summary of failure resilience

Suppose U is a set of well-behaved nodes in an FBAS

- Let $\overline{U}$ be the nodes not in U—might be ill-behaved

An FBAS can guarantee safety for U only if:

1. U enjoys quorum intersection despite $\overline{U}$.

Can guarantee correctness (safety+liveness) for U only if:

1. U enjoys quorum intersection despite $\overline{U}$, and
2. U is a quorum (equivalently, $\overline{U}$ is not v-blocking for any $v \in U$).

Definition (intact)

A node is intact if it is in a set U satisfying 1 and 2 above.

Theorem: quorum intersection implies one maximal intact set U

- Without quorum intersection, must subjectively assign blame and (conceptually) delete bad nodes to make notion of intact useful

Outline

Byzantine agreement (background), recast in terms of voting on irrefutable & neutralizable statements

Federated Byzantine Agreement (FBA)

Optimal failure resilience

The Stellar consensus protocol (SCP)

The Stellar Consensus Protocol [SCP]

First general FBA protocol

Guarantees safety for well-behaved nodes that enjoy quorum intersection despite ill-behaved nodes

- Means too many ill-behaved nodes can still cause divergence, but only if no other FBA protocol could guarantee safety, either
- I.e., you might regret your choice of quorum slices, but you won't regret choosing SCP over other Byzantine agreement protocols

Guarantees intact nodes will not get stuck

Core idea: *federated voting*

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

Federated voting

As before, each node *v* can *vote* for a statement *a*

- *a* must be consistent with past votes & accepted statements

Definition (ratify)

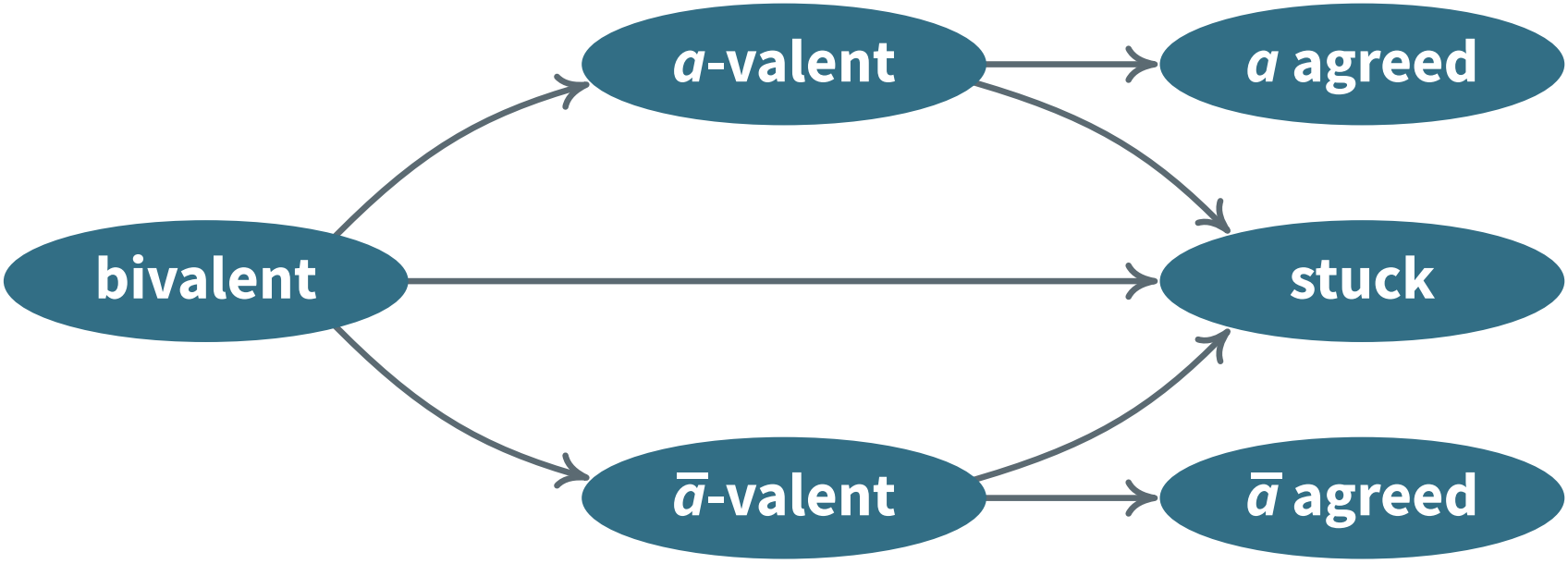
A quorum *U* ratifies a statement *a* iff every member of *U* votes for *a*.
A node *v* ratifies *a* iff *v* is a member of a quorum *U* that ratifies *a*.

Theorem: if well-behaved nodes enjoy quorum intersection despite ill-behaved nodes, won't ratify contradictory statements

Problem: intact node *v* may be unable to ratify *a* after others do

- *v* or nodes in *v*'s slices might have voted against *a*, or
- Some nodes that voted for *a* may subsequently have failed

Federated voting outcomes



Federated voting has same possible outcomes as regular voting

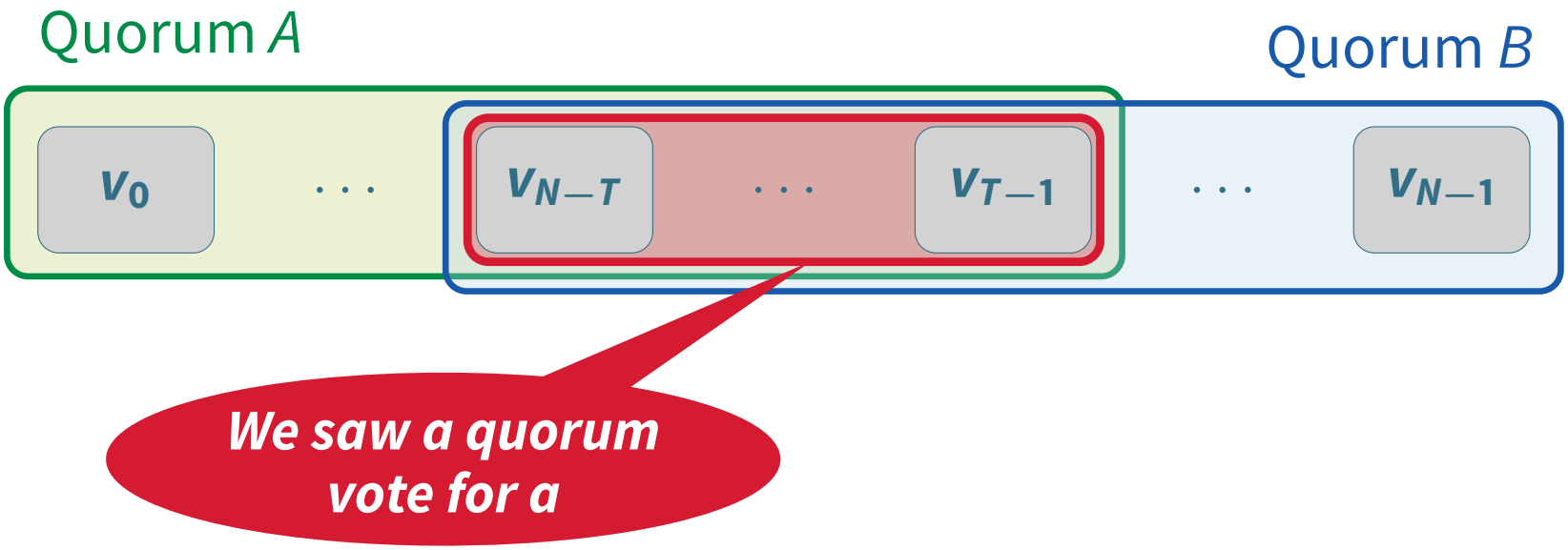
Apply the same reasoning as in centralized voting?

- Premise was whole system couldn't fail; now failure is per node
- Cannot 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 system has agreed on a?

Federated voting outcomes



Federated voting has same possible outcomes as regular voting

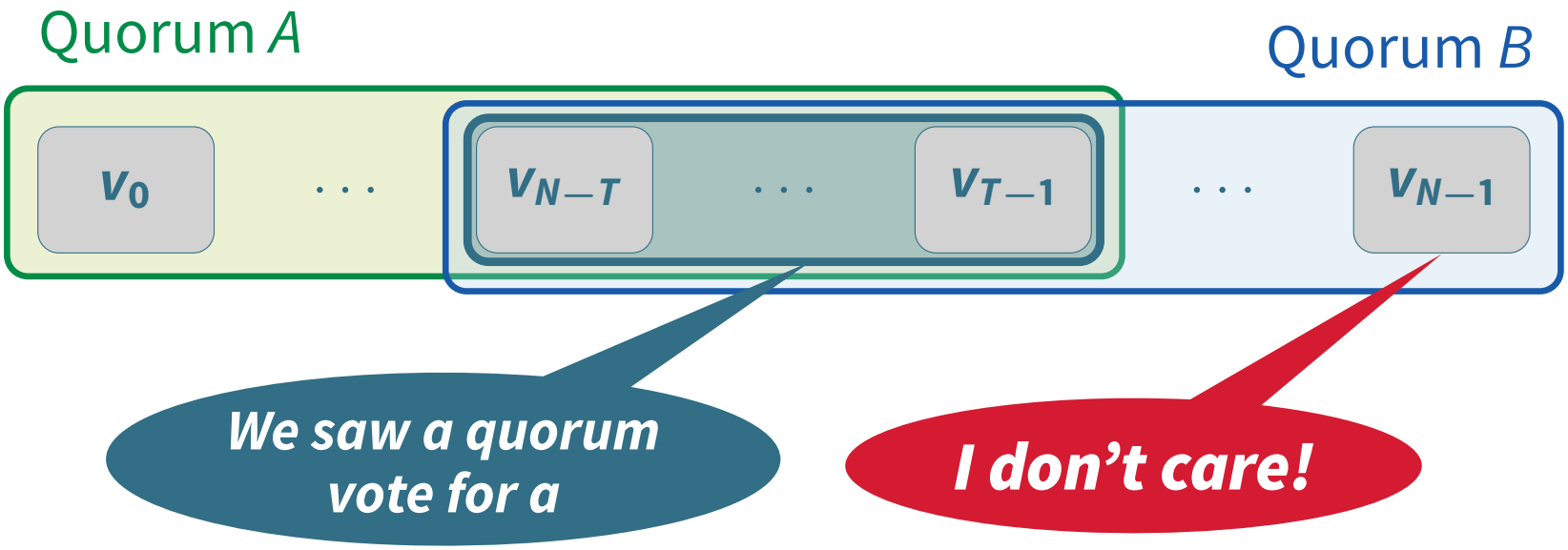
Apply the same reasoning as in centralized voting?

- Premise was whole system couldn't fail; now failure is per node
- Cannot 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 system has agreed on a?

Federated voting outcomes



Federated voting has same possible outcomes as regular voting

✗ **Apply the same reasoning as in centralized voting? No!**

- Premise was whole system couldn't fail; now failure is per node
- Cannot 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 system has agreed on a?

Accepting

Idea: say a v-blocking set claims system is a-valent

- Either whole set lying (so v is not intact), or system in fact a-valent

Definition (accept)

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

1. **A quorum containing v each either voted for or accepted a, or**
2. **Each member of a v-blocking set claims to accept a.**

#2 allows a node to accept a statement after voting against it

Theorem: intact nodes cannot accept contradictory statements

But two problems remain:

1. Still no guarantee all intact nodes can accept a statement
2. Suboptimal safety for non-intact nodes enjoying quorum intersection (v-blocking analogous to $f_L + 1$ in centralized system, not $f_S + 1$)

Confirmation

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

Definition (confirm)

A quorum confirms a statement *a* by ratifying the statement "We accepted *a*." A node confirms *a* iff it is in such a quorum.

Solves problem 2 (suboptimal safety)

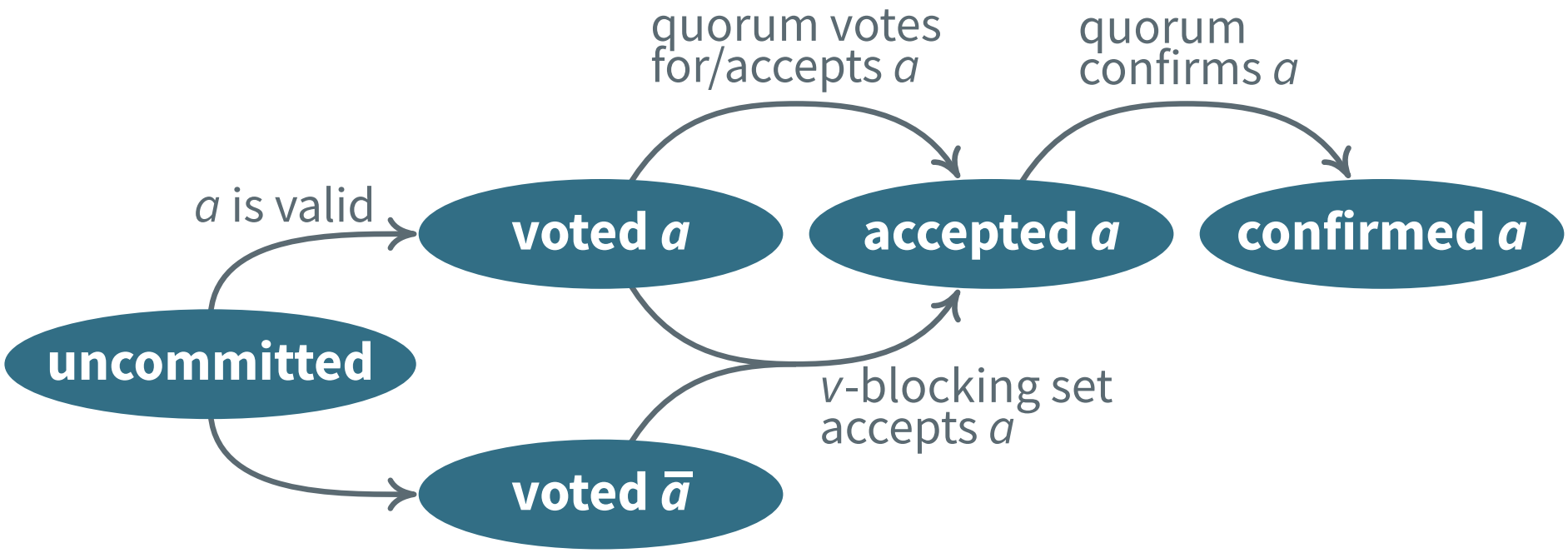
- Straight-up ratification provides optimal safety

Solves problem 1 (intact nodes unable to accept)

- Intact nodes may vote against accepted statements
- Won't vote against the *fact* that those statements were accepted
- Hence, the fact of acceptance is irrefutable

Theorem: Once a single intact node confirms a statement, all intact nodes will eventually confirm it

Summary of federated voting process



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

- Nodes enjoying quorum intersection with *v* despite ill-behaved nodes will not contradict *a*
- If *v* is intact, all other intact nodes will eventually confirm *a*

From voting to consensus

Like Paxos, vote to *commit* or *abort* ballots

- Each ballot is $\langle n, x \rangle$ where x is candidate consensus value
- Committing $\langle n, x \rangle$ choses x as the value

Same invariant: all committed & stuck ballots must have same x

Must *prepare* ballot $\langle n, x \rangle$ before voting to commit it:

- For convenience, totally order ballots with n most significant
- Prepare means agree to abort all $\langle n', x' \rangle < \langle n, x \rangle$ such that $x' \neq x$

If ballot $\langle n, x \rangle$ stuck, neutralize it by trying again with $\langle n + 1, x \rangle$

Also in paper: decentralized nomination protocol

- Tells you how to select x for system to terminate in practice
- Key idea: no way to vote against nominating value, so irrefutable

Summary of properties

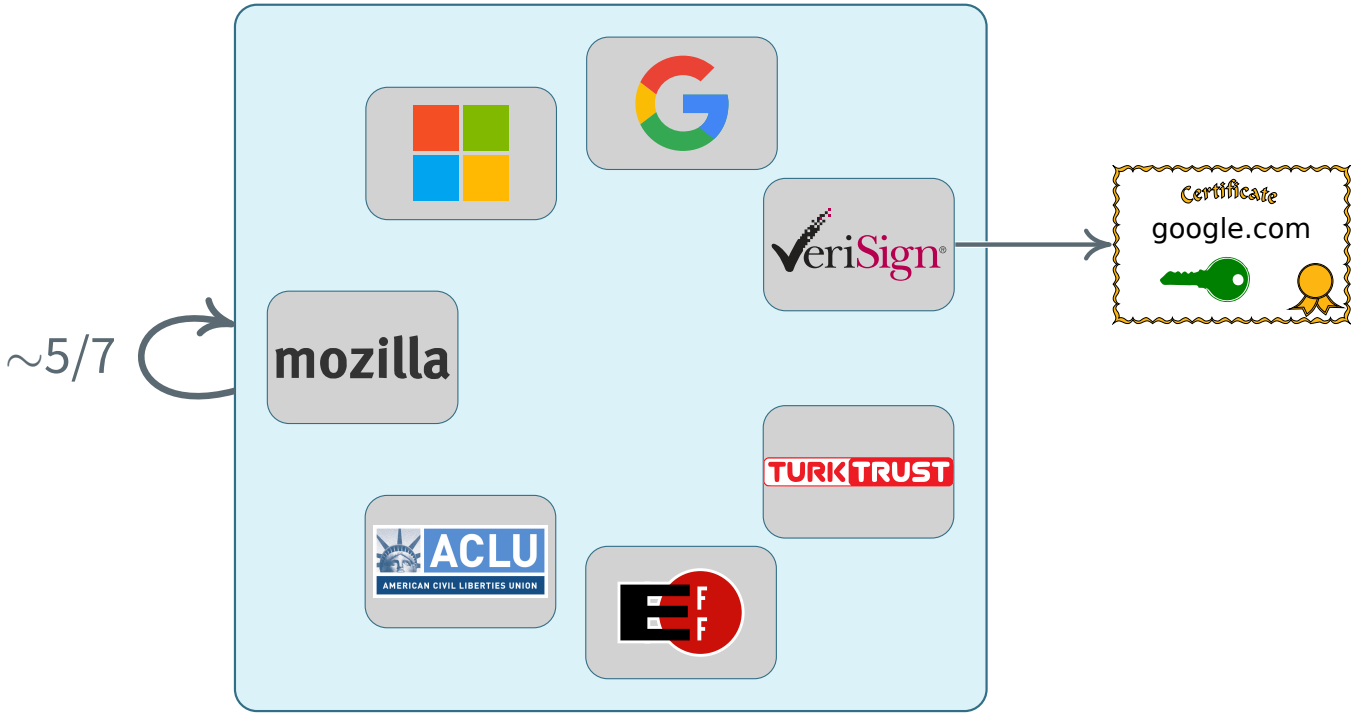
mechanism	decen-tralized	low latency	flexible trust	asympt. security
SCP	✓	✓	✓	✓
Byzantine agr.		✓	✓	✓
proof-of-work	✓			
proof-of-stake	✓	maybe		maybe

Note consensus $\neq$ cryptocurrency! SCP does not:

- ✗ Offer a rate-limited way to distribute ("mint") new digital coins
- ✗ Provide intrinsic incentives for good behavior
- ✗ Tell you whom to trust (some good configurations, some bad)

SCP has applications beyond financial networks

CA accountability



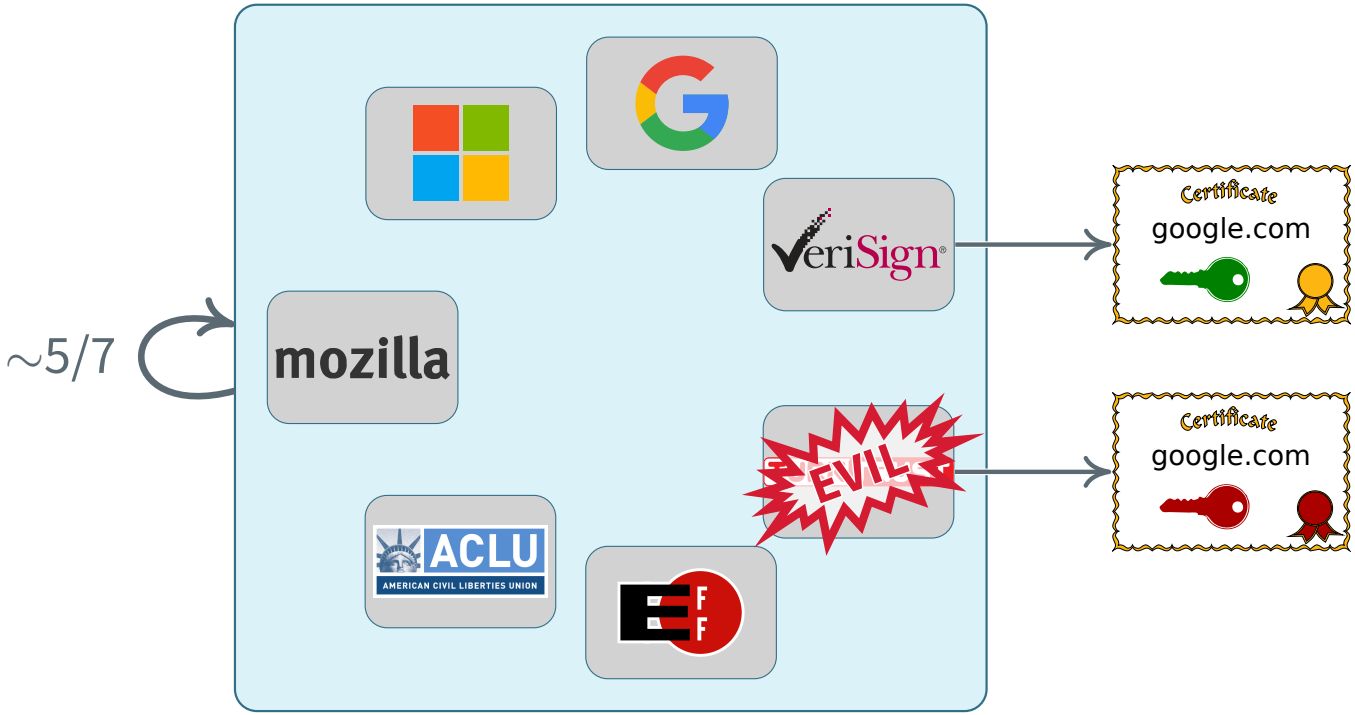
Problem: One bad certificate authority (CA) undermines TLS

- E.g., Turktrust issued fake Google certificates [ArsTechnica'13]
- Even more important for end-to-end email encryption

Can be addressed by auditing [certificate transparency], [CONIKS]

SCP can increase confidence in the auditing process

CA accountability



Problem: One bad certificate authority (CA) undermines TLS

- E.g., Turktrust issued fake Google certificates [ArsTechnica'13]
- Even more important for end-to-end email encryption

Can be addressed by auditing [certificate transparency], [CONIKS]

SCP can increase confidence in the auditing process

Build on Stellar

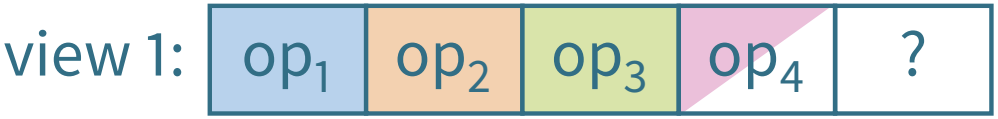
Growing list of partners building on the Stellar network

Is your organization next. . . ?

Find out more

https://www.stellar.org/

View-based neutralization [Oki]



Instead of voting on $op_1, \ldots$ directly, vote on $\langle$view 1, $op_1\rangle, \ldots$

- Each $\langle$view, op$\rangle$ selected by a single *leader* for view, so irrefutable
- E.g., chose leader by round-robin using *view#* mod *N*

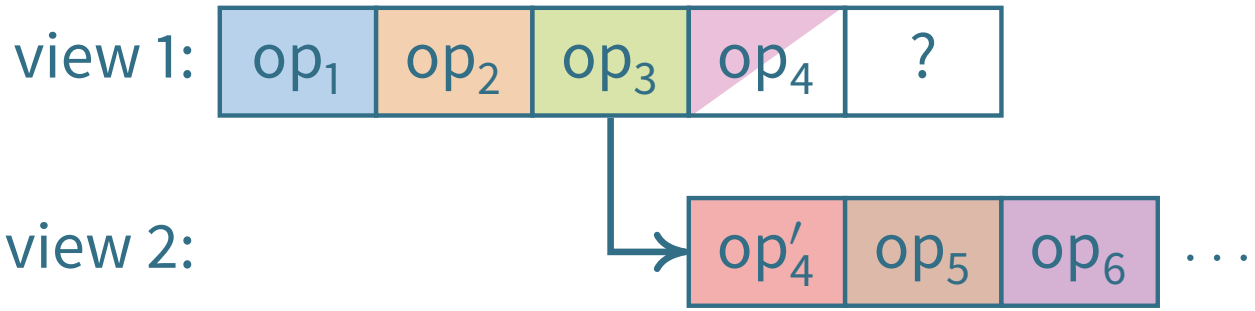
What if votes on op_4 and op_5 are stuck (e.g., leader fails)?

- Neutralize by agreeing view 1 had only 3 meaningful operations
- Vote to form view 2 that immediately follows $\langle$view 1, $op_3\rangle$

Failed to form view 2 (e.g., because a node wants $\langle$view 1, $op_4\rangle$)?

- Just go on to form view 3 after $\langle$view 1, $op_4\rangle$

View-based neutralization [Oki]



Instead of voting on $op_1, \ldots$ directly, vote on $\langle$view 1, $op_1\rangle, \ldots$

- Each $\langle$view, op$\rangle$ selected by a single *leader* for view, so irrefutable
- E.g., chose leader by round-robin using *view#* mod *N*

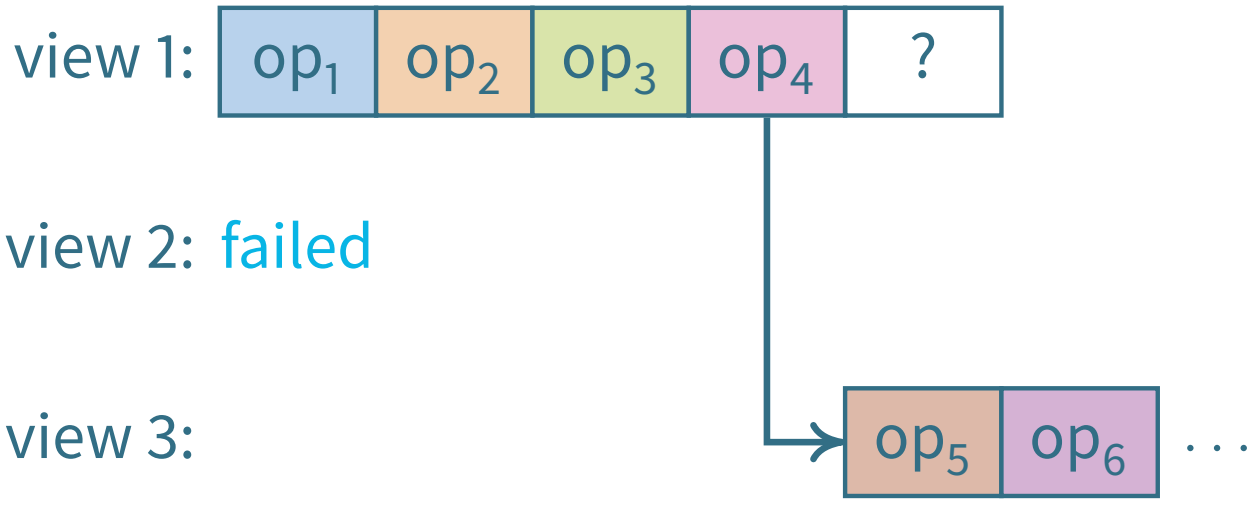
What if votes on op_4 and op_5 are stuck (e.g., leader fails)?

- Neutralize by agreeing view 1 had only 3 meaningful operations
- Vote to form view 2 that immediately follows $\langle$view 1, $op_3\rangle$

Failed to form view 2 (e.g., because a node wants $\langle$view 1, $op_4\rangle$)?

- Just go on to form view 3 after $\langle$view 1, $op_4\rangle$

View-based neutralization [Oki]



Instead of voting on $op_1, \ldots$ directly, vote on $\langle$view 1, $op_1\rangle, \ldots$

- Each $\langle$view, op$\rangle$ selected by a single *leader* for view, so irrefutable
- E.g., chose leader by round-robin using *view#* mod *N*

What if votes on op_4 and op_5 are stuck (e.g., leader fails)?

- Neutralize by agreeing view 1 had only 3 meaningful operations
- Vote to form view 2 that immediately follows $\langle$view 1, $op_3\rangle$

Failed to form view 2 (e.g., because a node wants $\langle$view 1, $op_4\rangle$)?

- Just go on to form view 3 after $\langle$view 1, $op_4\rangle$

Properties of Byzantine agreement

+ **Low latency – by human standards**
  - E.g., might require 5 communication rounds in common case
+ **Flexible trust – The *N* nodes can include anyone appropriate**
  - E.g., small nonprofit can help keep big banks honest
+ **Asymptotic security – based on standard cryptography**
  - Can tune to resist attackers with arbitrary computational power
– **Centralized control – who chooses the *N* nodes?**
  - Imagine if one party dictated all tier-one ISPs worldwide
  - Makes Byzantine agreement totally unsuitable for global ledger
  - Yet Bitcoin provides overwhelming evidence of the utility of decentralized consensus

Proof-of-work-based consensus

Digitally sign transaction history with *DMMS*

- Dynamic Membership Multi-party Signature
- Cost requires worldwide collaboration
- Each signature compounds security

Motivate attackers to participate in DMMS

- Reward participation with coin distribution or transaction fees

+ Completely decentralized and open membership

- Bitcoin introduced it, achieved astounding impact
- Clearly the missing ingredient from past consensus approaches

– Lacks benefits of Byzantine agreement

- High latency, trust determined by computing power, modest computational security assumes rational attackers

Living with FLP

General plan: devise protocols that terminate *in practice*

- …unless the network exhibits truly pathalogical behavior

FLP is for deterministic algorithms—so randomize [Ben-Or]

- Bivalent? Keep flipping a coin in a series of rounds
- Terminates with probability 1, but expected time $\Omega\left(2^{\#\text{nodes}}\right)$
- Fancy crypto can help [Rabin], but often assumes trusted dealer

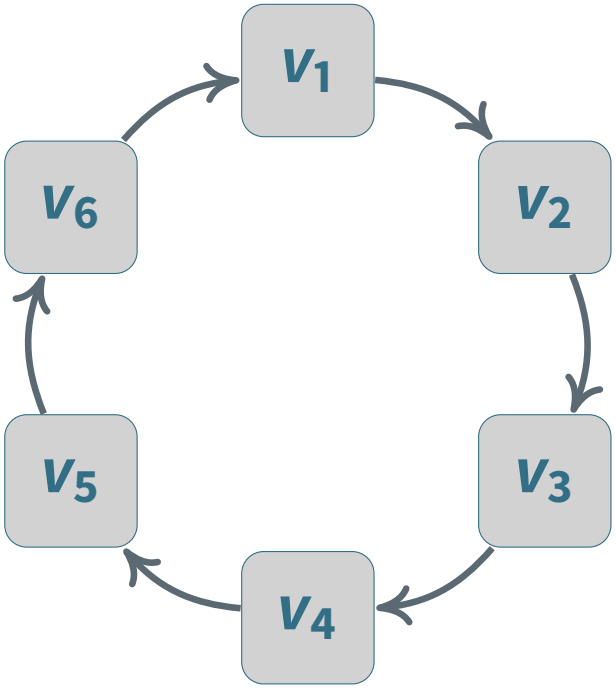
FLP is for asynchronous systems—so assume synchrony [DLS]

- E.g., long enough timeout means node has failed
- Ideally only liveness, not safety rests on this assumption

Key property: protocol must avoid stuck states

- Hence, can always hope for better luck or lengthen timeouts

Cyclic quorum slice example



$$\mathbf{Q}(v_i) = \left\{\left\{v_i, v_{(i \bmod 6)+1}\right\}\right\}$$

Traditional Byzantine agreement requires $\forall(i,j), \mathbf{Q}(v_i) = \mathbf{Q}(v_j)$

- Means no distinction between quorums and quorum slices

***Federated* Byzantine agreement accommodates different slices**

- May even have disjoint slices if you have cycles
- Shouldn't necessarily invalidate safety guarantees