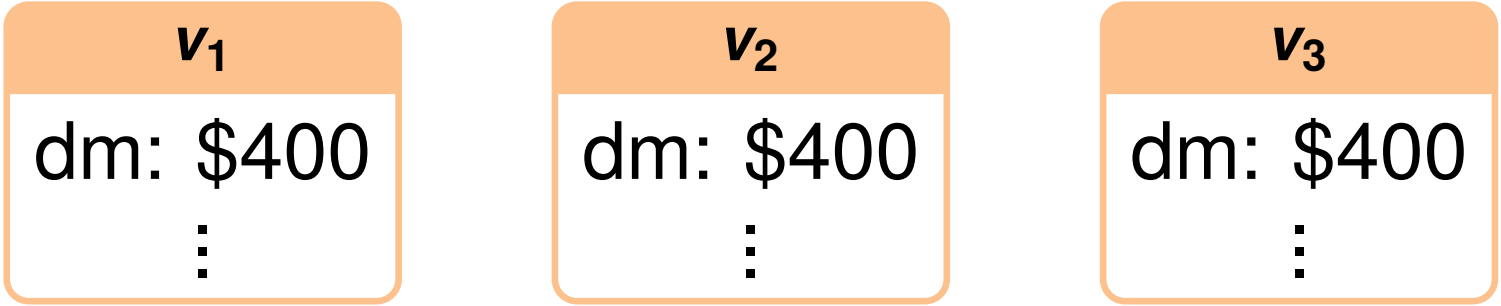
The Stellar Consensus Protocol

David Mazières

Stellar Development Foundation
Stanford CS140/CS251

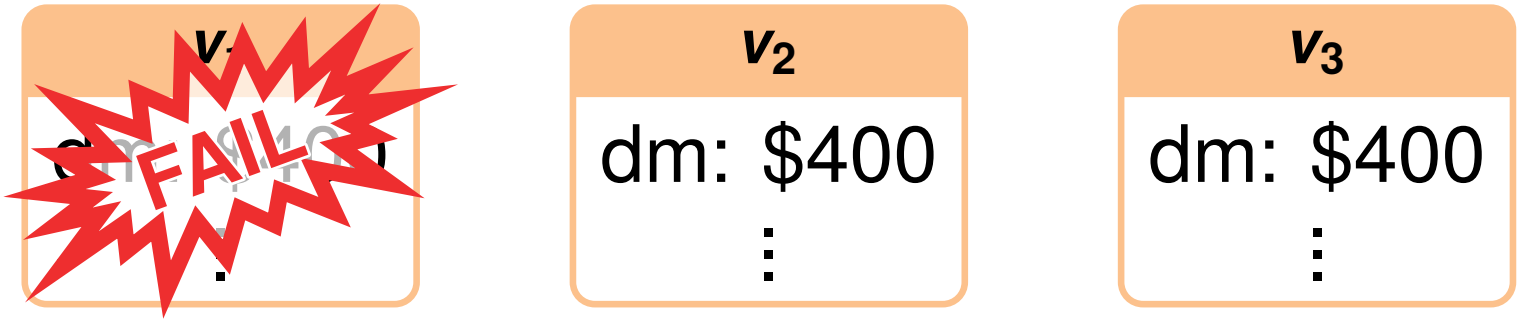
Wednesday, November 11, 2015

Replication



- Replication is the key to robust, high-availability systems
- Example: replicate bank ledger on three machines
  - Won't lose data if a machine dies
- Challenge: must keep all replicas identical
  - Availability requires updating state even after a failure
  - Yet a network partition is indistinguishable from a replica failure
  - Creates a danger of lost writes and divergence

Replication

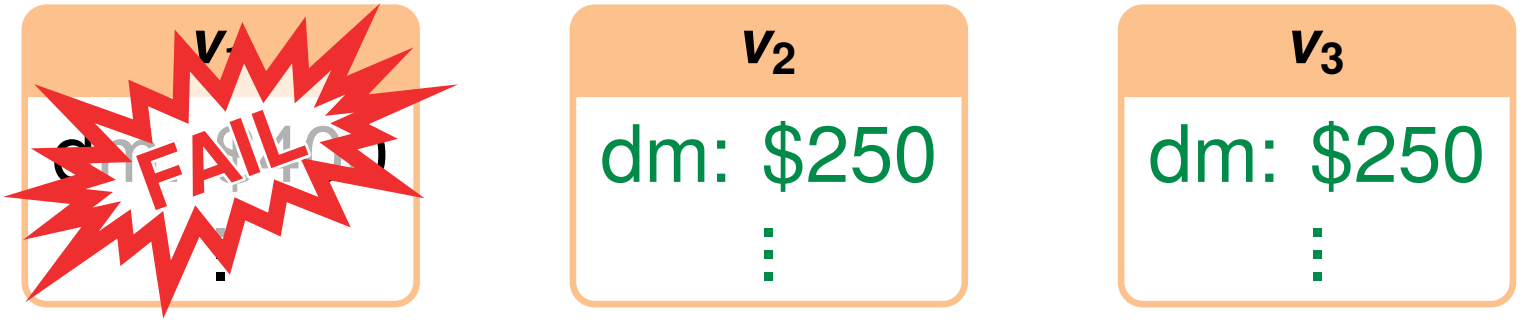


- Replication is the key to robust, high-availability systems
- Example: replicate bank ledger on three machines

→ Won't lose data if a machine dies

- Challenge: must keep all replicas identical
  - Availability requires updating state even after a failure
  - Yet a network partition is indistinguishable from a replica failure
  - Creates a danger of lost writes and divergence

Replication

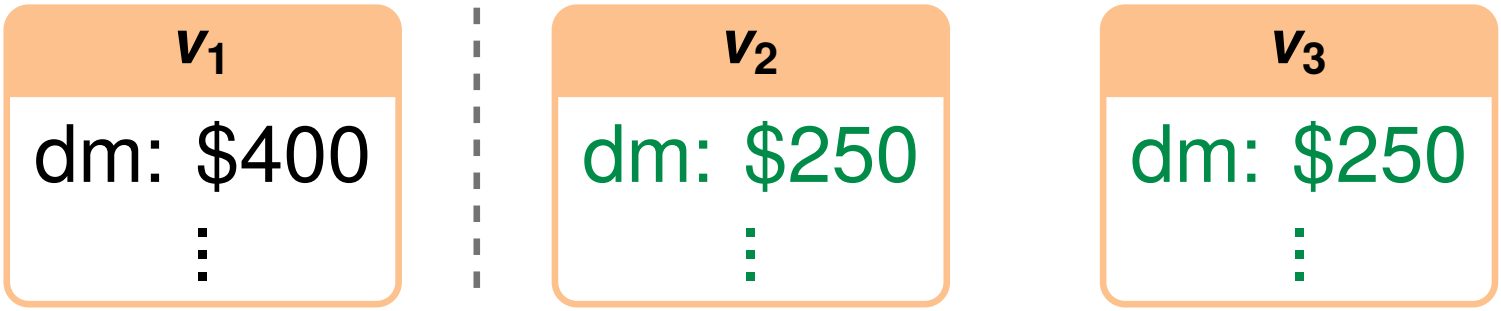


- Replication is the key to robust, high-availability systems
- Example: replicate bank ledger on three machines
  - Won't lose data if a machine dies
- Challenge: must keep all replicas identical

→ Availability requires updating state even after a failure

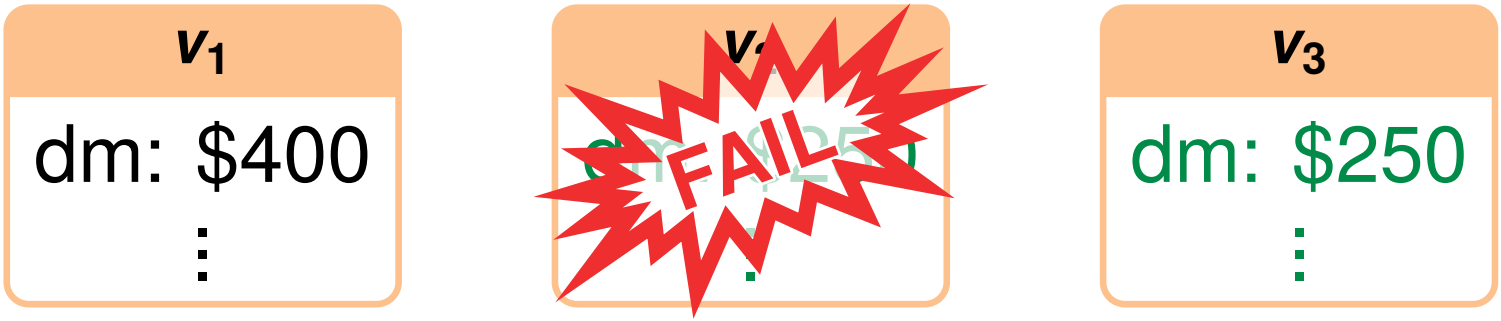
  - Yet a network partition is indistinguishable from a replica failure
  - Creates a danger of lost writes and divergence

Replication



- Replication is the key to robust, high-availability systems
- Example: replicate bank ledger on three machines
  - Won't lose data if a machine dies
- Challenge: must keep all replicas identical
  - Availability requires updating state even after a failure
  - → Yet a network partition is indistinguishable from a replica failure
  - Creates a danger of lost writes and divergence

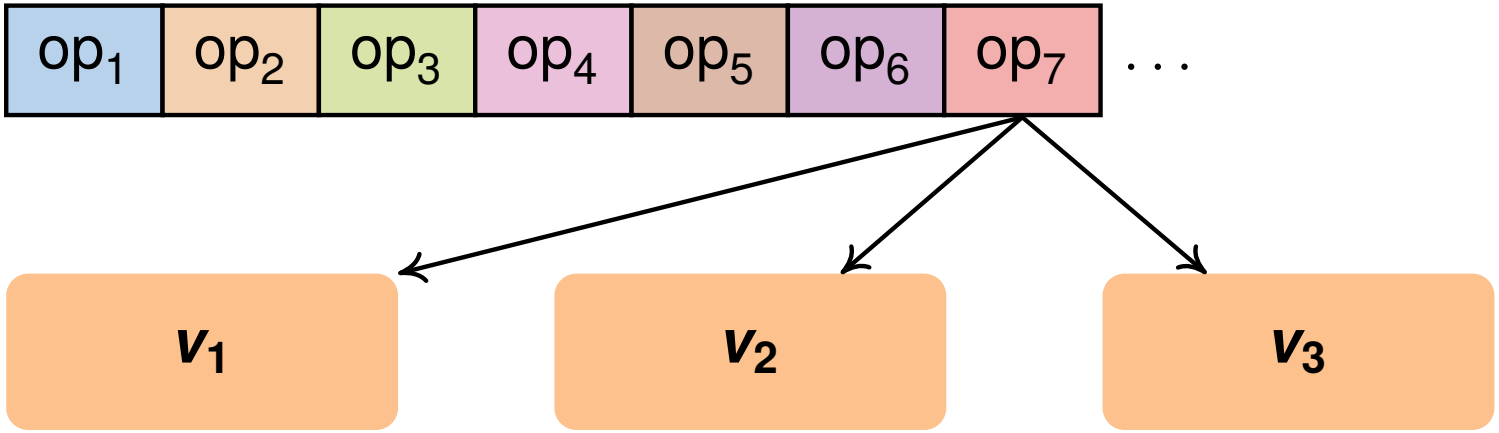
Replication



- Replication is the key to robust, high-availability systems
- Example: replicate bank ledger on three machines
  - Won't lose data if a machine dies
- Challenge: must keep all replicas identical
  - Availability requires updating state even after a failure
  - Yet a network partition is indistinguishable from a replica failure

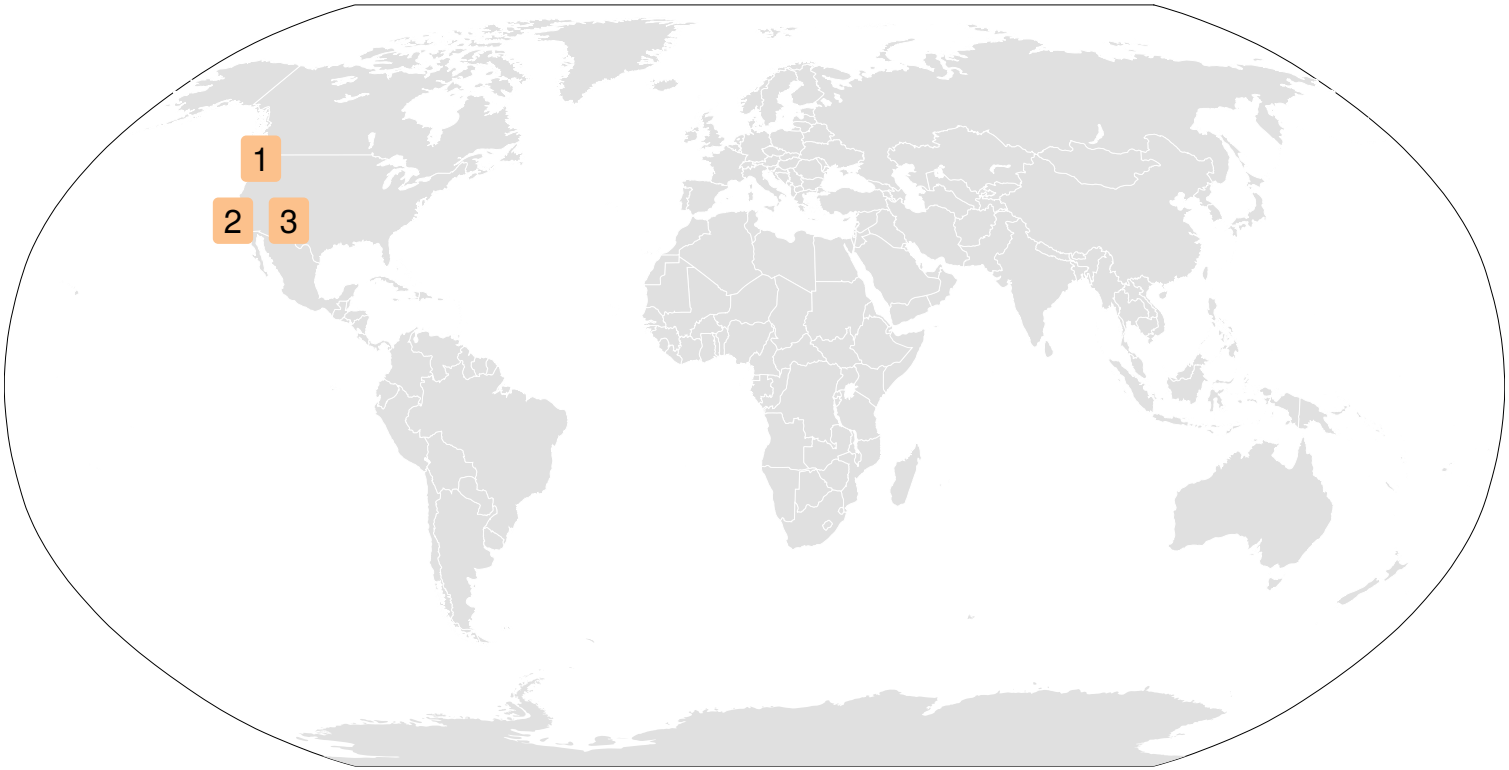
→ Creates a danger of lost writes and divergence

Replicated state machines



- To keep replicas in sync, can use *replicated state machines*
  1. All replicas agree on initial state of system, and
  2. All replicas agree on sequence of deterministic operations
- #1 is trivial to ensure (software initializes database)
- #2 requires consensus among replicas on each operation
  - E.g., agree op_7 = debit dm $150 before considering payment settled

Banking system robustness



- A bank's core job is tracking customer assets and liabilities
  - Can store *ledger* as a replicated state machine
  - Apply identical sequence of transactions at each replica
- What about the robustness of inter-bank transfers?

Banking system robustness

- A bank's core job is tracking customer assets and liabilities
  - Can store *ledger* as a replicated state machine
  - Apply identical sequence of transactions at each replica

→ What about the robustness of inter-bank transfers?

Inter-bank transactions

- Within countries/regions, use payment networks (ACH, Fedwire)
  - Trust government to regulate network so banks can't steal money
- Internationally, use *ad hoc* correspondent banking relationships
  - Route route payments through whole series of banks
  - Need pairwise trust at each step of the way
- The result?
  - High transaction costs

    Migrants moving cash from Tanzania to Kenya pay average fees of 22%, but mobiles could make transfers cheaper and easier

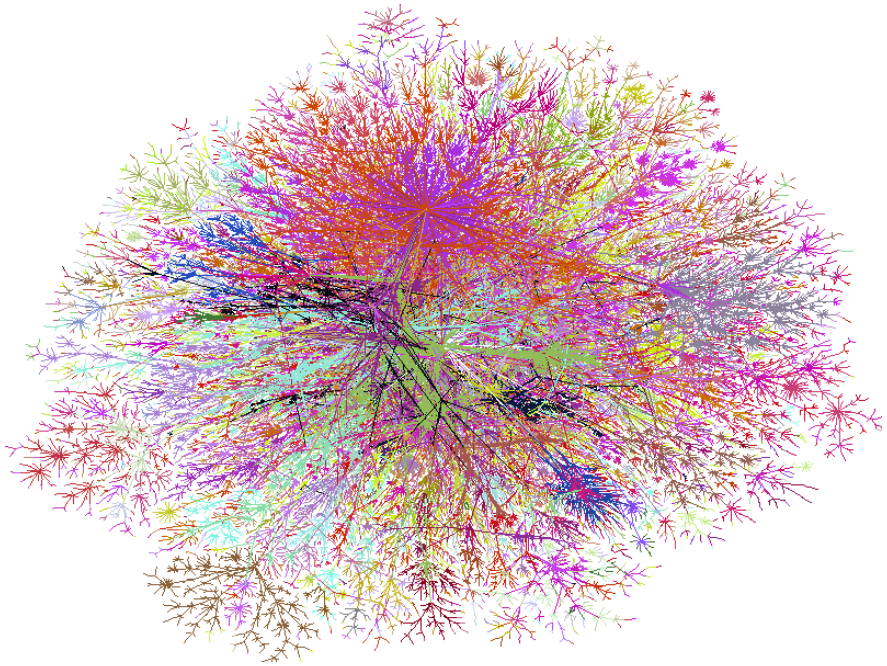
    [guardian'13]
  - Limited reach, high latency

    Boatfuls of cash: how do you get money into fragile states?

    [guardian'15]

An inter-financial network

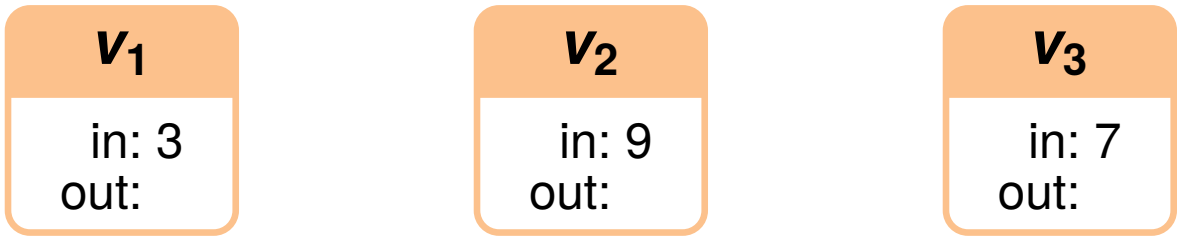
- Today's financial networks like computer networks before Internet
- Can we create an *inter-financial network* akin to the Internet?
  - Sending money should be as easy and cheap as sending email
- This would be possible if we had a *global* ledger
  - Transfer money across continents by atomically updating ledger
- Problem: there is no globally trusted party to manage a global ledger
  - Only local trust by each bank
- But the Internet is also built out of pairwise peering relationships!
- Idea: leverage local trust to secure global consensus



Outline

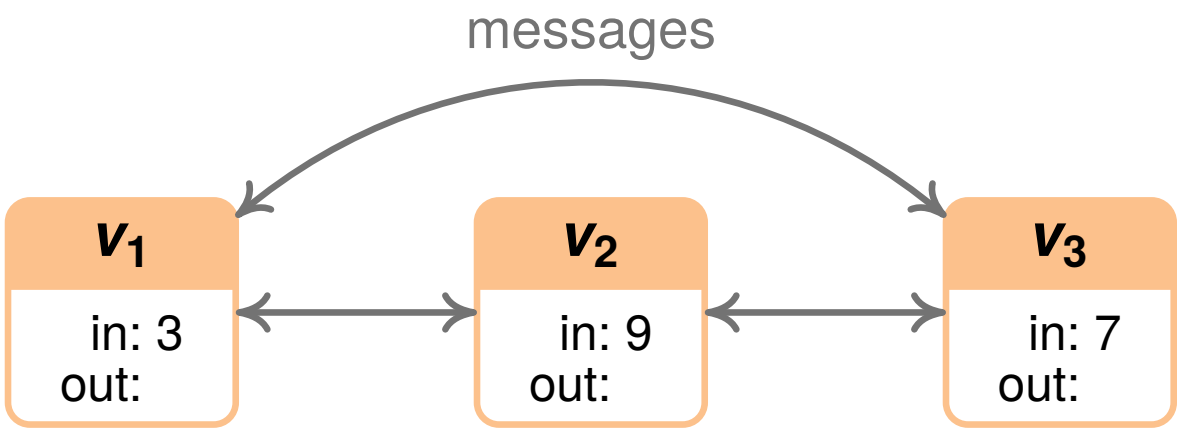
1. Textbook consensus
2. Proof-of-work consensus
3. Federated Byzantine agreement (FBA)

The consensus problem



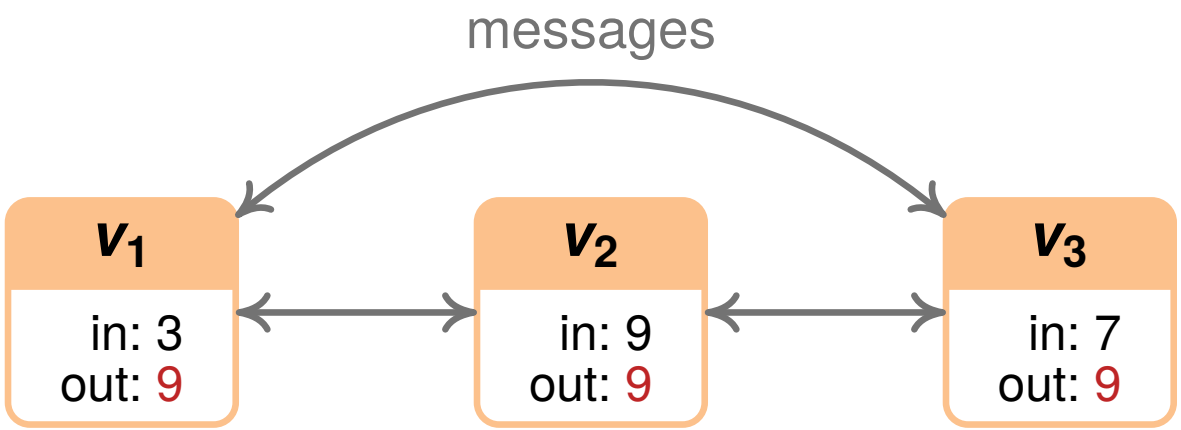
- Goal: For multiple agents to agree on an output value
- → Each agent starts with an input value
  - Typically a candidate for the *n*th op. on a replicated state machine
  - Agents' inputs may differ; any agent's input is okay to output
- Agents communicate following some *consensus protocol*
  - Use 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' inputs may differ; any agent's input is okay to output
- → 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' inputs may differ; any agent's input is okay to output
- 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)

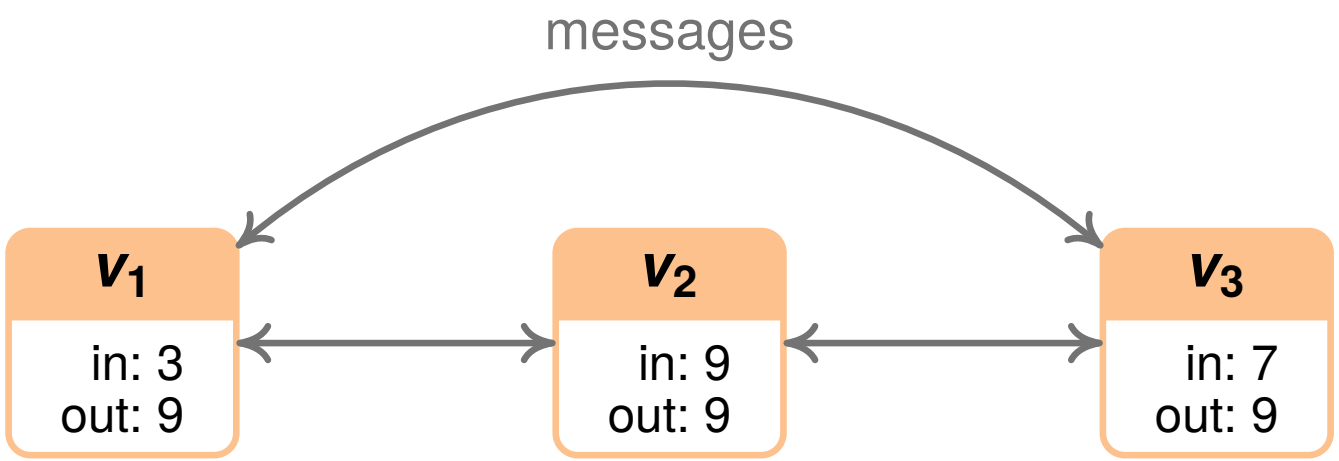
Properties of a consensus protocol

- A consensus protocol provides *safety* if...
  - Agreement – All outputs produced have the same value, and
  - Validity – The output value equals one of the agents' inputs
- A consensus protocol provides *liveness* if...
  - Termination – Eventually non-faulty agents output a value
- A consensus protocol provides *fault tolerance* if...
  - It can recover the failure of an agent at any point
  - Fail-stop protocols handle agent crashes
  - Byzantine-fault-tolerant protocols handle compromised agents

Theorem (FLP impossibility result)

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

Bivalent states



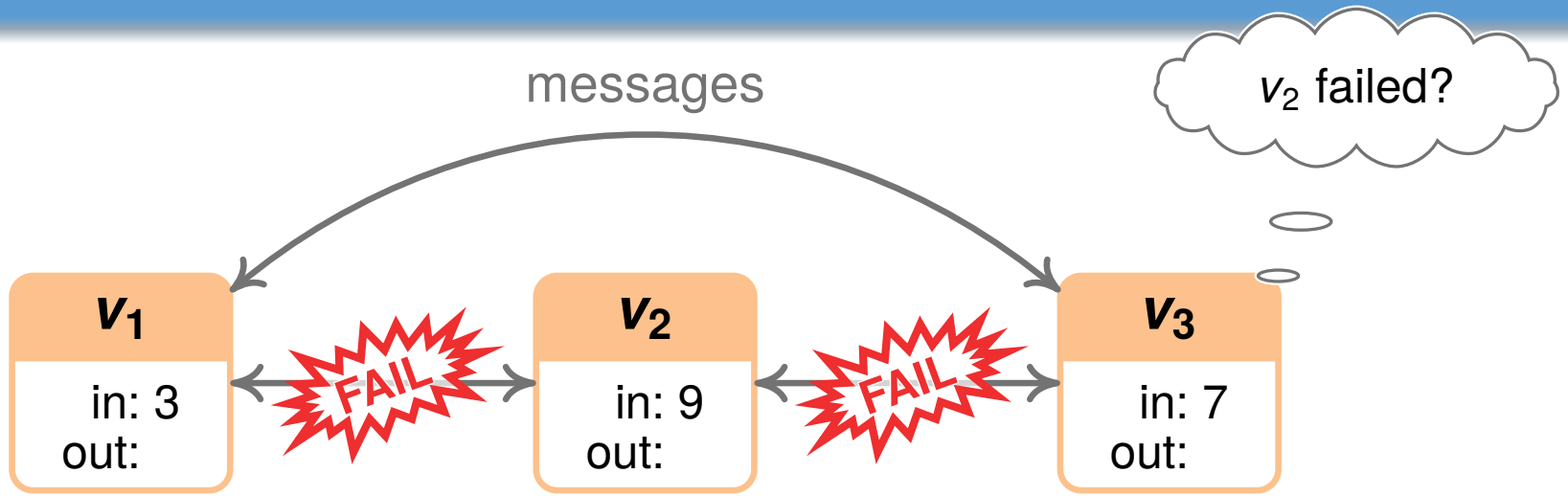
→ Recall agents chose value 9 in last example

- But a network outage might be indistinguishable from v_2 failing
- Nodes v_1 and v_3 might decide to output 7
- Once network back, Agent 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

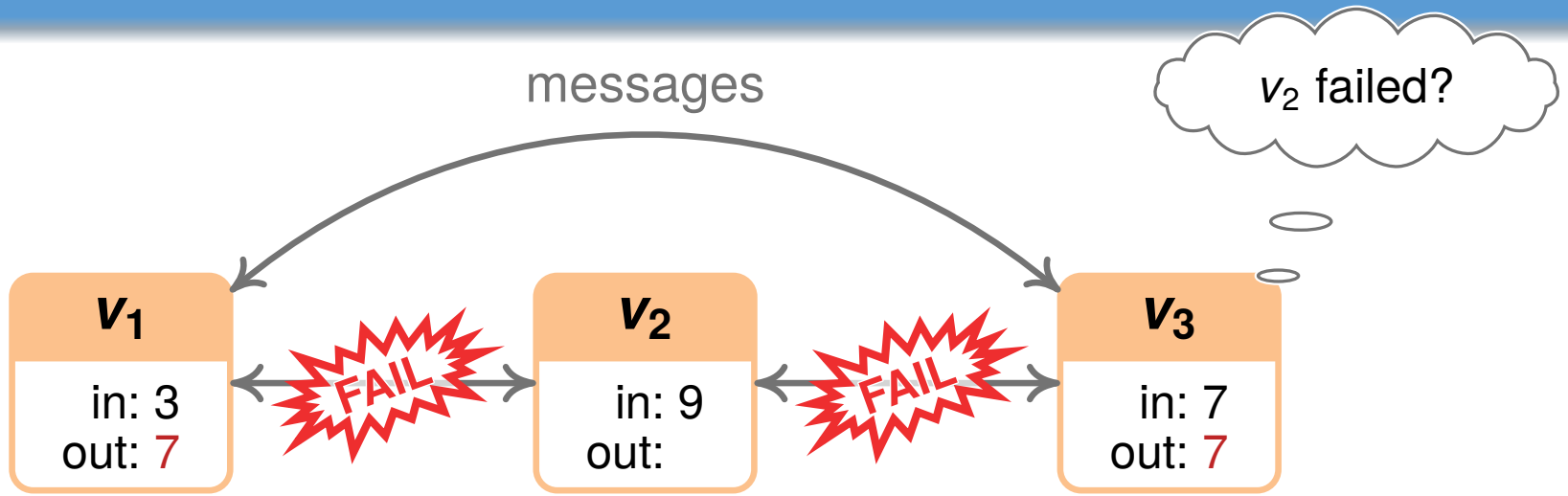


- Recall agents chose value 9 in last example
- → But a network outage might be indistinguishable from v_2 failing
- Nodes v_1 and v_3 might decide to output 7
- Once network back, Agent 2 must also output 7

Definition (Bivalent)

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

Bivalent states

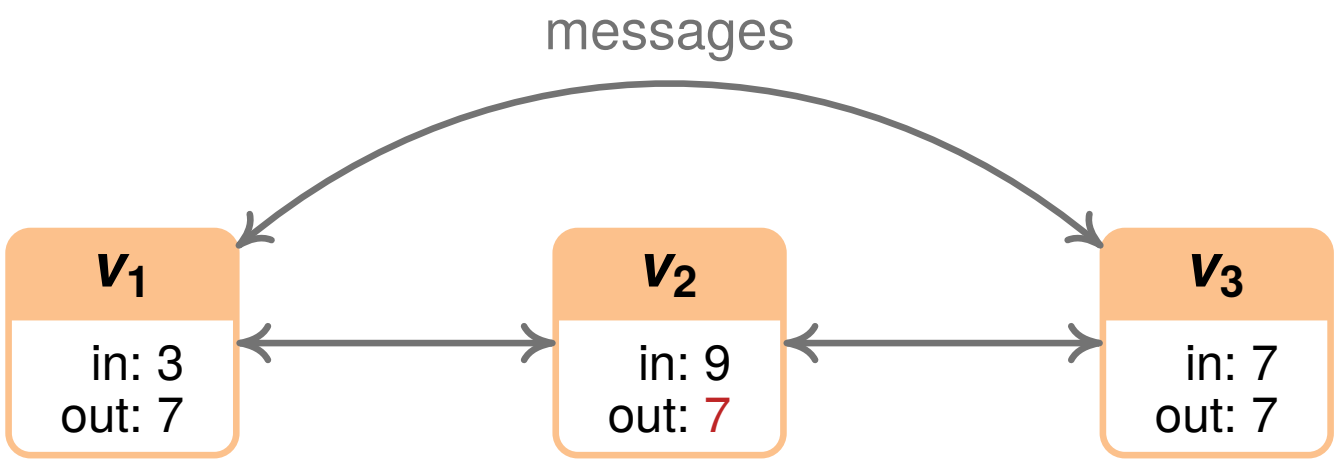


- Recall agents chose value 9 in last example
- But a network outage might be indistinguishable from v_2 failing
- → Nodes v_1 and v_3 might decide to output 7
- Once network back, Agent 2 must also output 7

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
- Nodes v_1 and v_3 might decide to output 7

→ Once network back, Agent 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)

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

Definition (stuck)

An execution of a 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 last message received in a bivalent state
  - Call m the execution's deciding message
  - Any terminating execution requires a 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

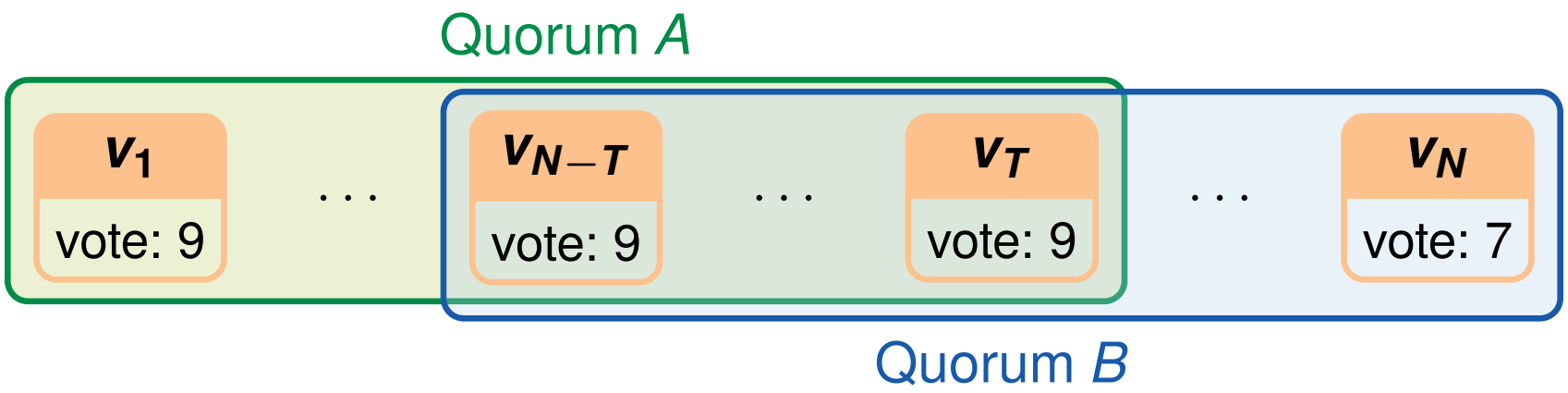
Overview of FLP proof.

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

Living with FLP

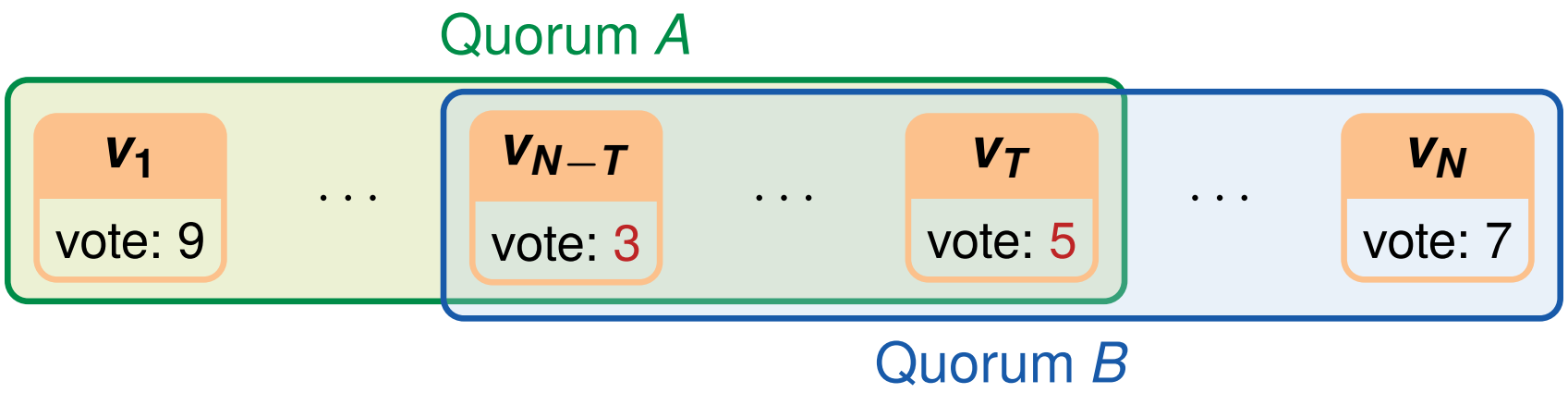
- Real systems need consensus despite FLP, so what to do?
- FLP is for deterministic algorithms—so randomize [Ben-Or]
  - Bivalent? Keep flipping a coin in a series of rounds
  - Terminate if enough nodes randomly pick same value in same round
  - Terminates with probability 1, but exponential expected time
- No guaranteed termination doesn't mean it won't terminate anyway
  - Can devise protocol that terminates in practice unless the network exhibits truly pathalogical behavior
- Make assumptions about synchrony [DLS]
  - E.g., long enough timeout means node has failed
  - Ideally only liveness, not safety rests on this assumption

Voting



- Given N nodes, pick a quorum size $T > N/2$
- If any T nodes (a quorum) vote for a value, output that value
  - E.g., if quorum A unanimously votes for 9, choose 9
  - Any two quorums intersect, nodes cannot change votes $\Longrightarrow$ safe
- Does simple voting solve the consensus problem?

Voting

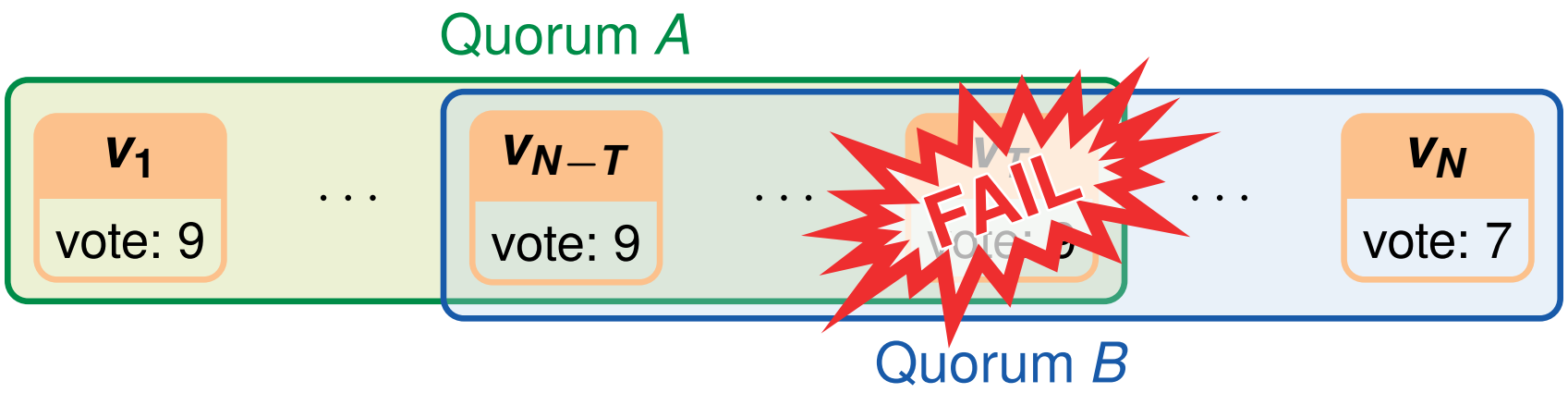


- Given N nodes, pick a quorum size $T > N/2$
- If any T nodes (a quorum) vote for a value, output that value
  - E.g., if quorum A unanimously votes for 9, choose 9
  - Any two quorums intersect, nodes cannot change votes $\Longrightarrow$ safe
- Problem: does not guarantee liveness

$\longrightarrow$ Split vote could make unanimous quorum impossible

  - Note failure could make unanimous quorum impossible

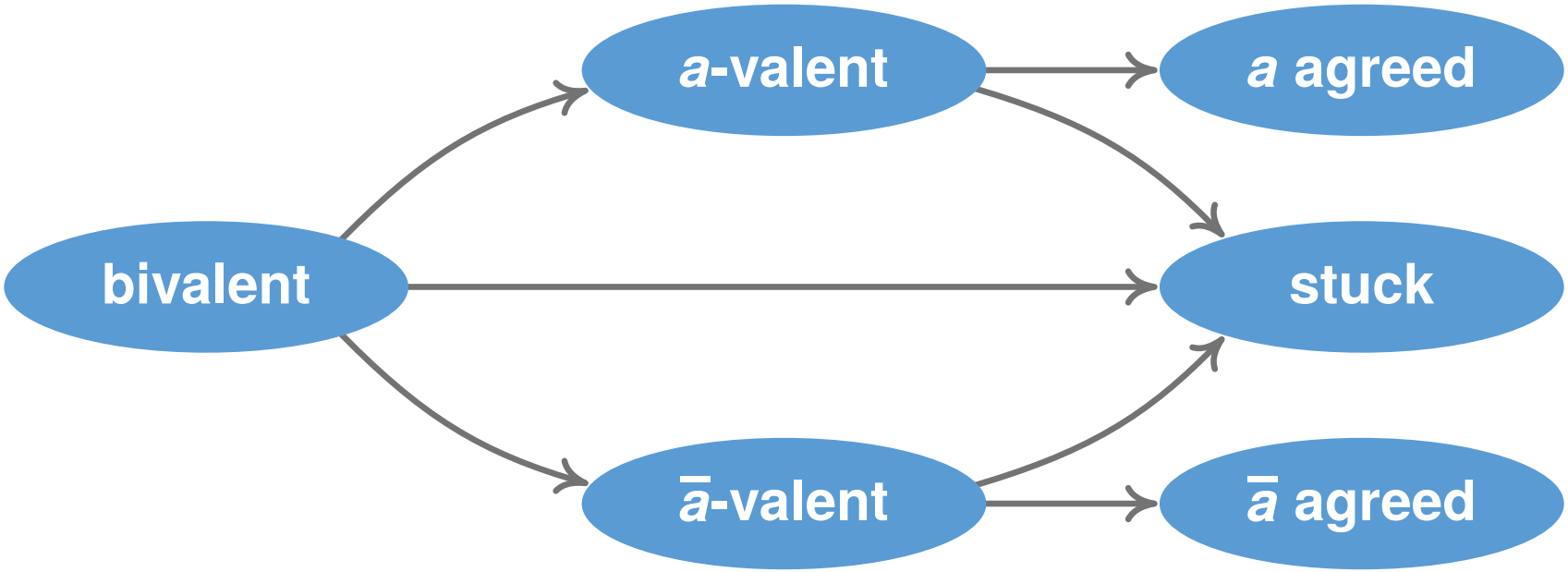
Voting



- Given N nodes, pick a quorum size $T > N/2$
- If any T nodes (a quorum) vote for a value, output that value
  - E.g., if quorum A unanimously votes for 9, choose 9
  - Any two quorums intersect, nodes cannot change votes $\Longrightarrow$ safe
- Problem: does not guarantee liveness
  - Split vote could make unanimous quorum impossible

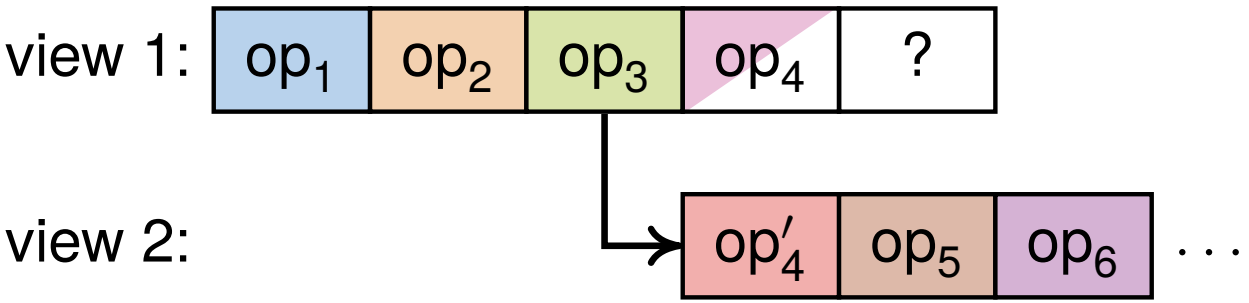
$\longrightarrow$ Note failure could make unanimous quorum impossible

What voting gives us



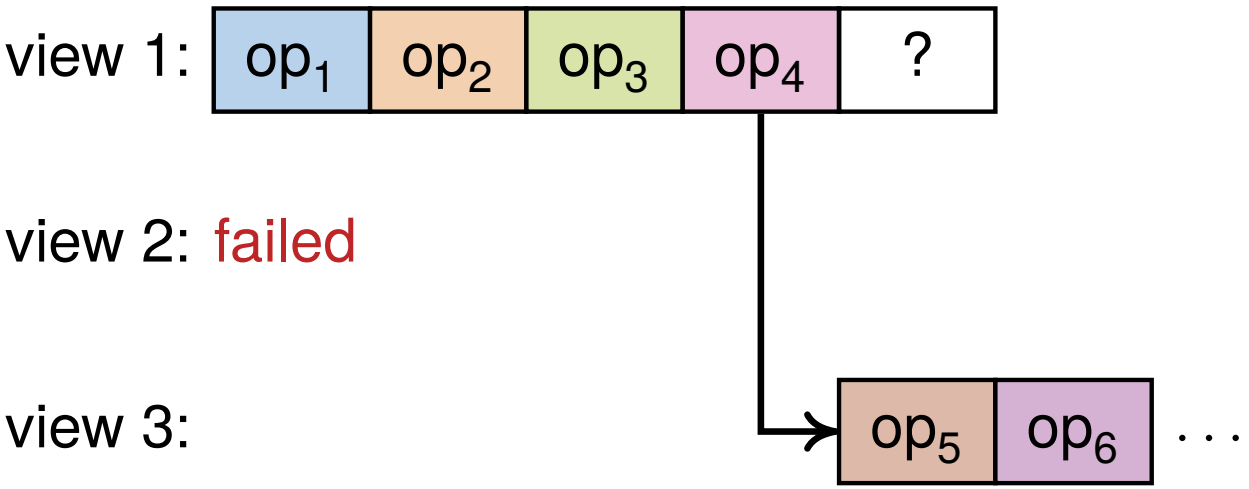
- You might get agreement or you might get stuck
  - Can't vote directly on consensus question ($op_7 \stackrel{?}{=}$ debit dm \$150)
- What kind of statement is it safe to vote on?
  - An irrefutable statement – all correct nodes will vote identically
  - A neutralizable one – can break its hold on consensus question
- How to formulate useful yet neutralizable statements?

View-based neutralization [Oki]



- Instead of voting on $op_1, \ldots$ directly, vote on $\langle \text{view } 1, op_1 \rangle, \ldots$
- Vote stuck/inconclusive on op_4, op_5?
  - Each view controlled by a single leader, who might have failed
  - No problem; just agree view 1 had only 3 meaningful operations
- → Vote to form view 2 with previous operation $\langle \text{view } 1, op_3 \rangle$
- Failed to form view 2 (e.g., because someone wants op_4)?
  - Just go on to form view 3 after $\langle \text{view } 1, op_4 \rangle$

View-based neutralization [Oki]



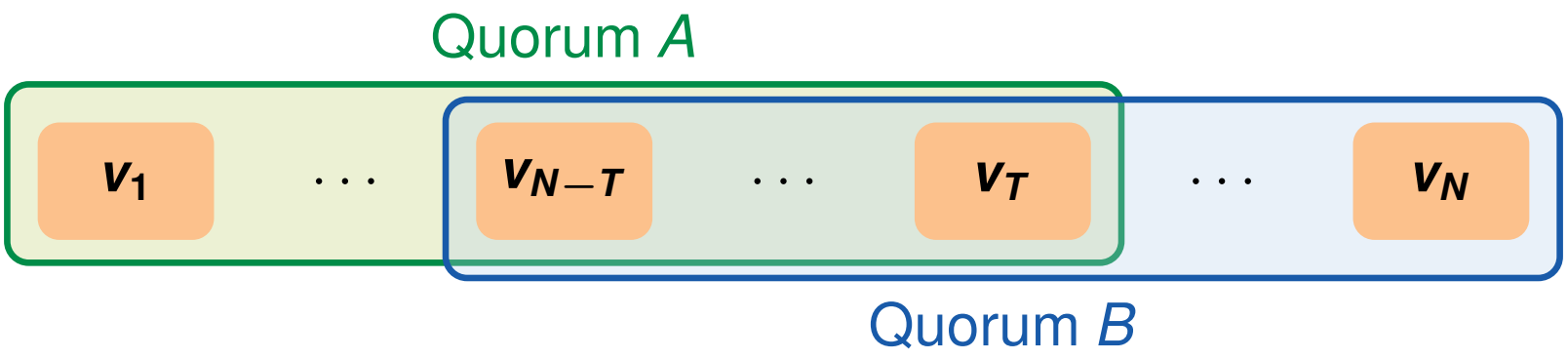
- Instead of voting on $op_1, \ldots$ directly, vote on $\langle \text{view } 1, op_1 \rangle, \ldots$
- Vote stuck/inconclusive on op_4, op_5?
  - Each view controlled by a single leader, who might have failed
  - No problem; just agree view 1 had only 3 meaningful operations
- Vote to form view 2 with previous operation $\langle \text{view } 1, op_3 \rangle$

→ Failed to form view 2 (e.g., because someone wants op_4)?
  - Just go on to form view 3 after $\langle \text{view } 1, op_4 \rangle$

Ballot-based neutralization [Paxos]

- Vote to *commit* or *abort* sets of ballots
  - Each ballot is $\langle n, x \rangle$ where x is candidate operation
  - Totally order ballots with n most significant
  - Committing $\langle n, x \rangle$ outputs x as consensus value
- To avoid getting stuck, can't just vote to commit some $\langle n, x \rangle$:
  - Must first agree all $\{ \langle n', x' \rangle \mid \langle n', x' \rangle < \langle n, x \rangle \text{ and } x' \neq x \}$ aborted
  - We say $\langle n, x \rangle$ is *prepared* when these are aborted
- Invariant: all committed and stuck ballots have same value
- If ballot $\langle n, x \rangle$ stuck, neutralize by restarting with $\langle n + 1, x \rangle$

Byzantine agreement



- What if nodes may experience Byzantine failure?
  - Byzantine nodes can change illegally their votes
- *Safety* requires: # failures $\leq f_S = 2T - N - 1$
  - In fail-stop case, any two quorums must share a node
  - In Byzantine case, any two quorums must share a *non-faulty* node
- *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

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 resist attackers with arbitrary computational power
– Centralized control – who chooses then N nodes?
  - Imagine if one party dictated all tier-one ISPs worldwide
  - Makes Byzantine agreement totally unsuitable for global ledger

Outline

1. Textbook consensus
2. Proof-of-work consensus
3. Federated Byzantine agreement (FBA)

Background: Proof of work

- 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

Outline

1. Textbook consensus
2. Proof-of-work consensus
3. Federated Byzantine agreement (FBA)

Federated Byzantine agreement (FBA)

- FBA is a new model for Byzantine agreement protocols
  - Goal: Combine Byzantine agreement with decentralized control
- Idea: determine quorums in a decentralized way
  - Each node v picks one or more *quorum slices*
  - v acts on a statement only if one of its slices unanimously agrees

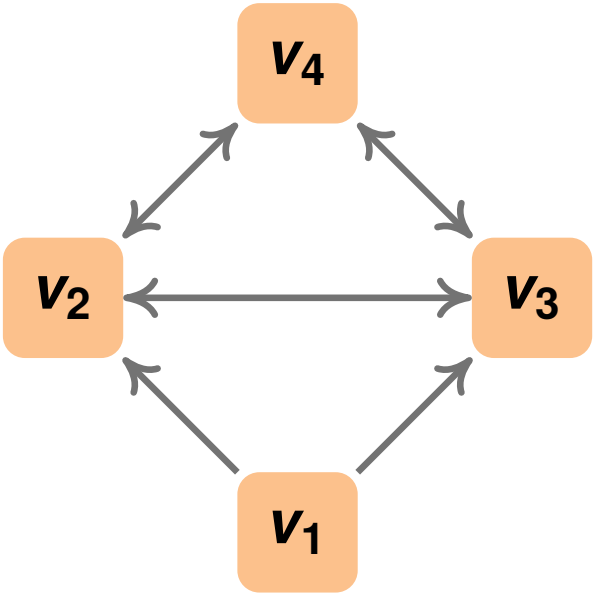
Definition (Federated Byzantine Agreement System)

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 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 slices



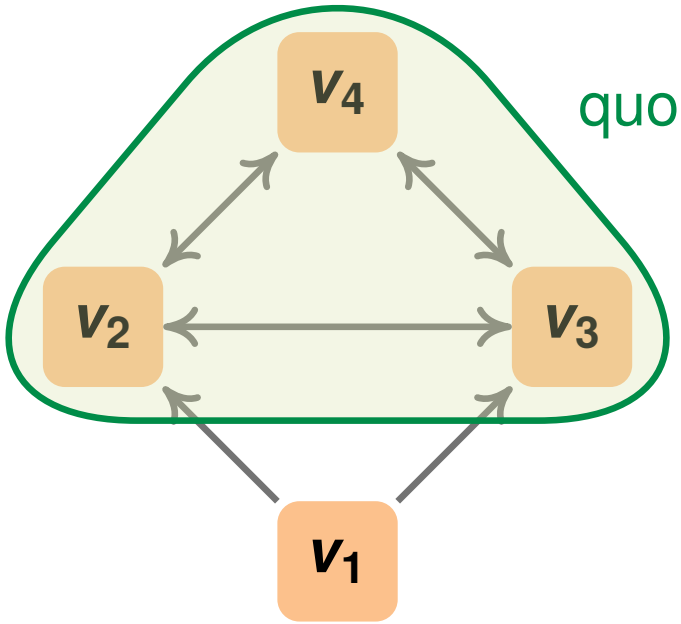
$$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3\}\}$$

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

→ Visualize quorum slice dependencies with arrows

- v_2, v_3, v_4 is a quorum—contains a slice of each member
- v_1, v_2, v_3 is a slice for v_1, but not a quorum
  - Doesn't contain a slice for v_2, v_3, who demand v_4's agreement
- $v_1, \ldots, v_4$ is the smallest quorum containing v_1

Quorum slices



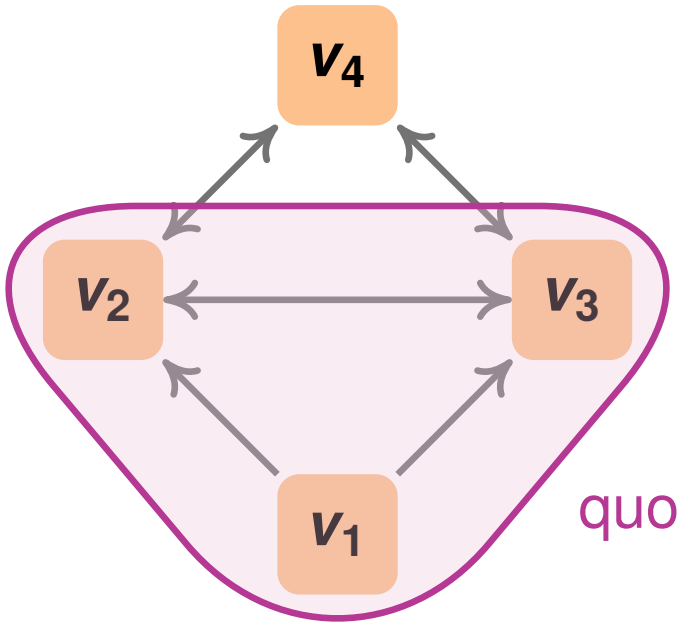
quorum for v_2, v_3, v_4

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

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

- 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

Quorum slices



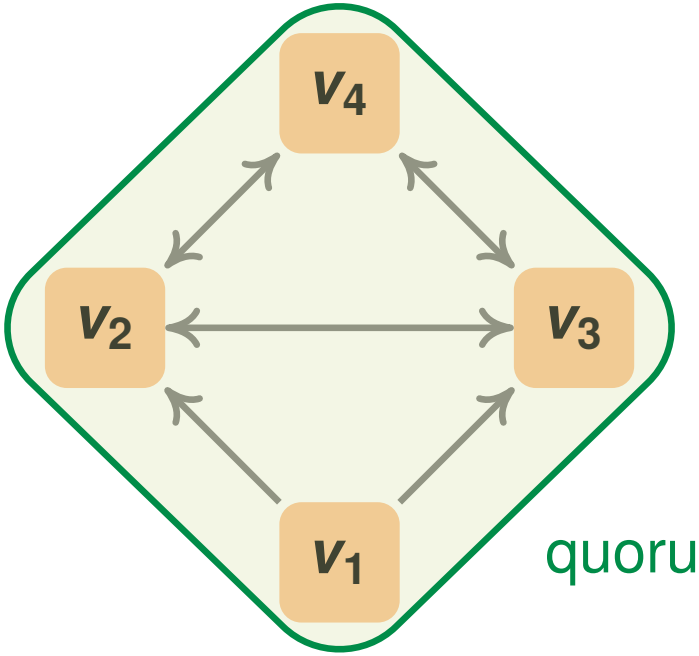
$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3\}\}$

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

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

Quorum slices



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

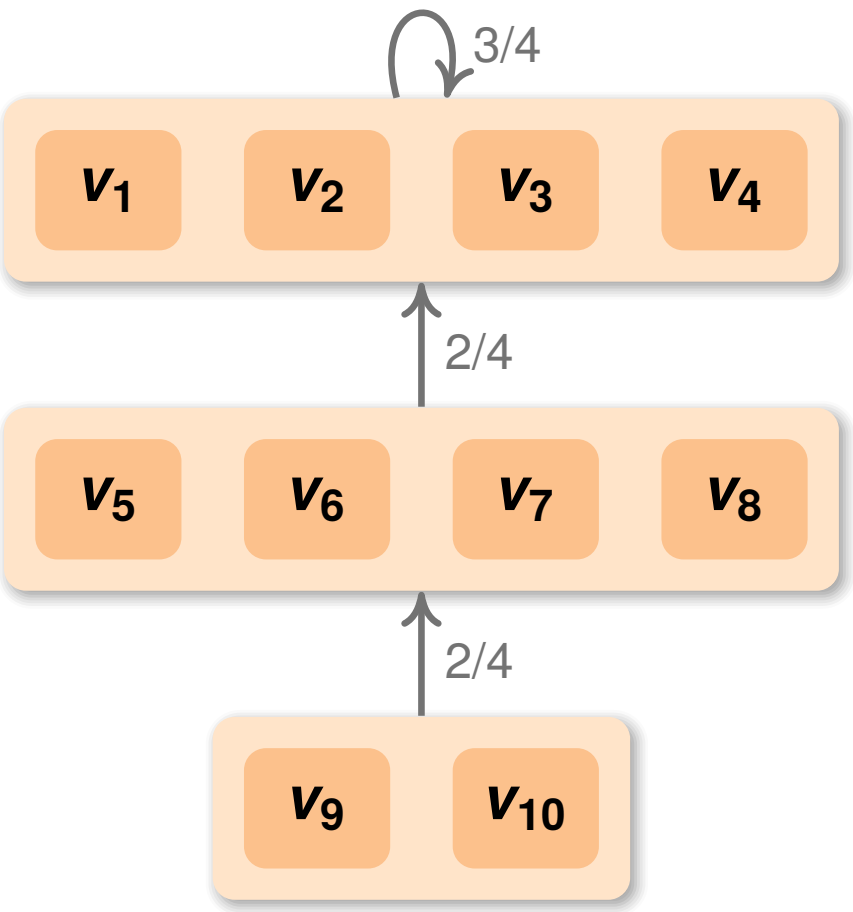
$\mathbf{Q}(v_2) = \mathbf{Q}(v_3) = \mathbf{Q}(v_4) = \{\{v_2, v_3, v_4\}\}$

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



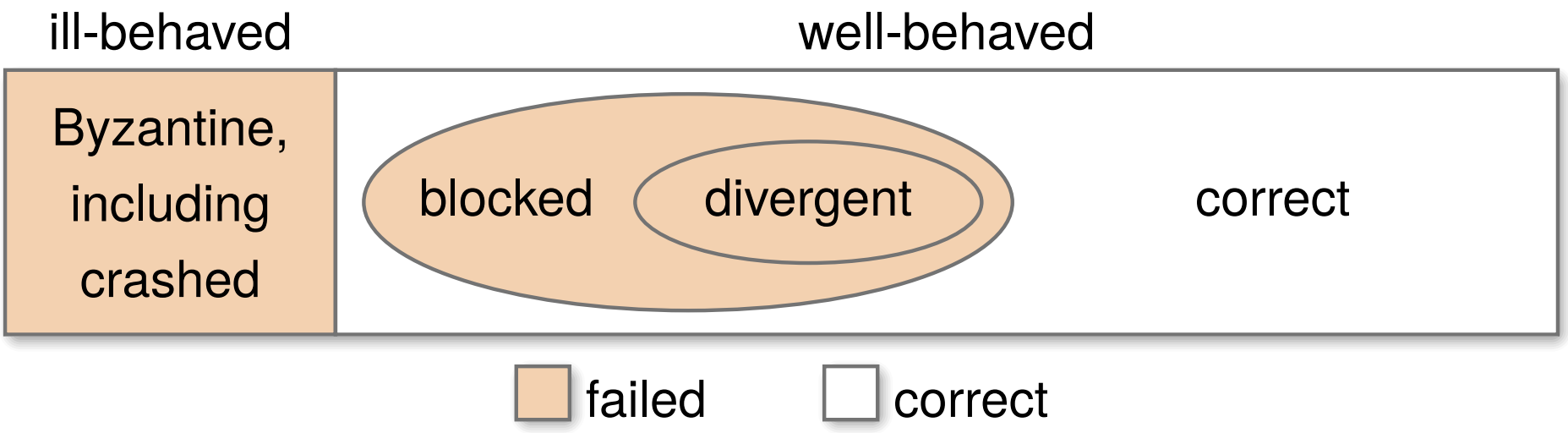
Top tier: slice is three out of $\{v_1, v_2, v_3, v_4\}$ (including self)

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
  - Don't even require exact agreement on who is a top tier node

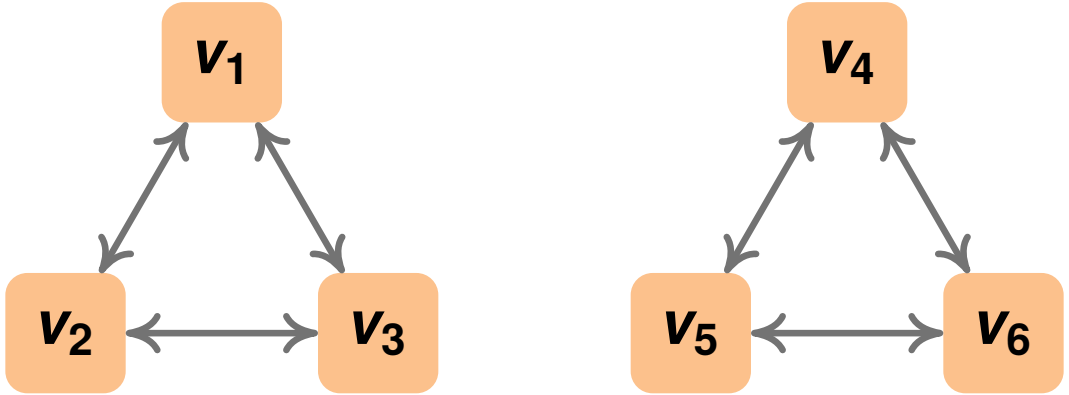
FBAS failure is per-node



- Each node is either *well-behaved* or *ill-behaved*
- All ill-behaved nodes have *failed*
- Enough ill-behaved nodes can induce failure in well-behaved ones
  - *Blocked* nodes are well-behaved but lack liveness
  - *Divergent* nodes are well-behaved but lack safety & liveness
- Well-behaved nodes are *correct* if they have not failed

When is safety impossible?

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



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

- 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

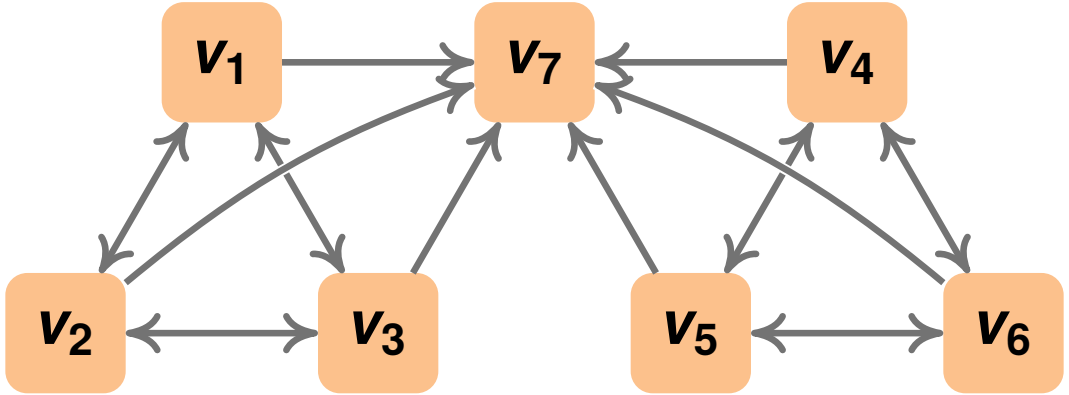
Definition (Quorum intersection)

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

What about Byzantine failures?

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

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



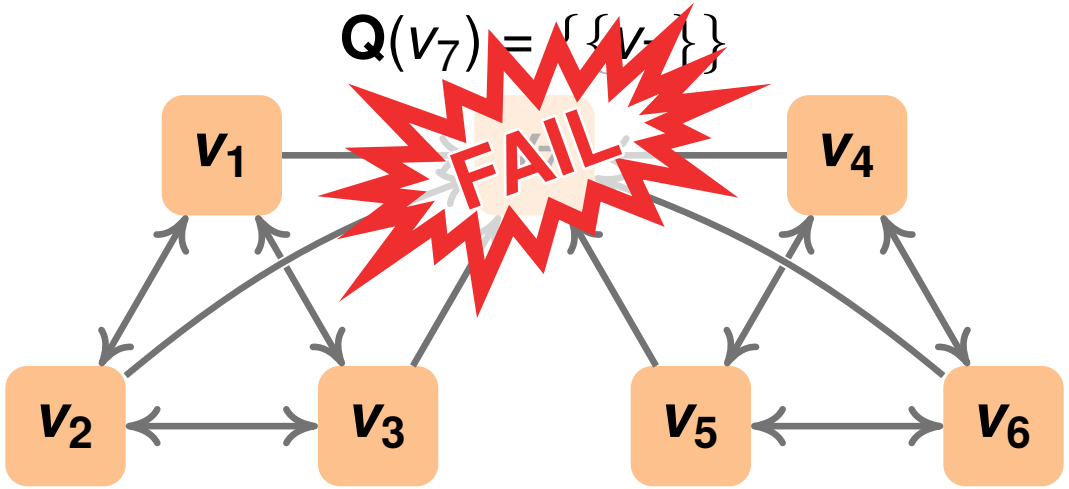
$\mathbf{Q}(v_4) =$
$\mathbf{Q}(v_5) =$
$\mathbf{Q}(v_6) =$
$\{\{v_4, v_5, v_6, v_7\}\}$

- 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
- Revised necessary property for safety:
  Need quorum intersection *after deleting malicious nodes*
  - E.g., reduces above diagram to one 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

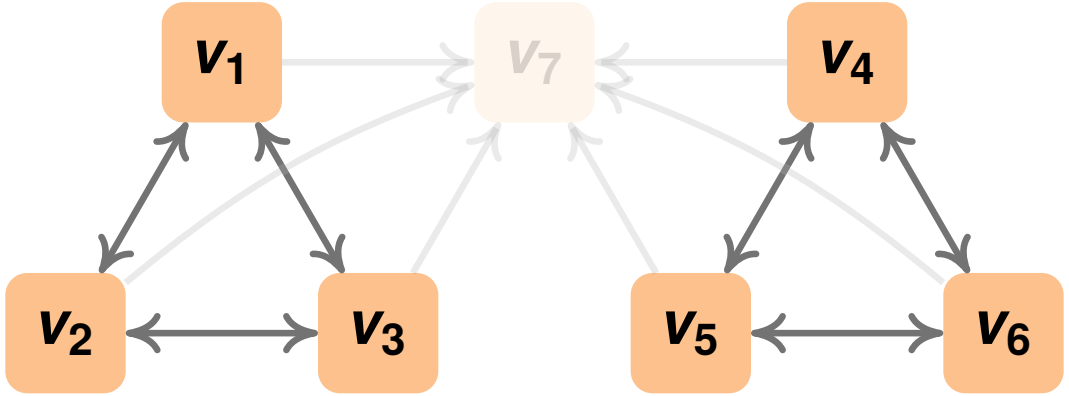
- Byzantine nodes behave arbitrarily
- Can feed inconsistent data to different honest nodes
- No way to guarantee safety

- Revised necessary property for safety:
  Need quorum intersection *after deleting malicious nodes*
  - E.g., reduces above diagram to one on previous slide

What about Byzantine failures?

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

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

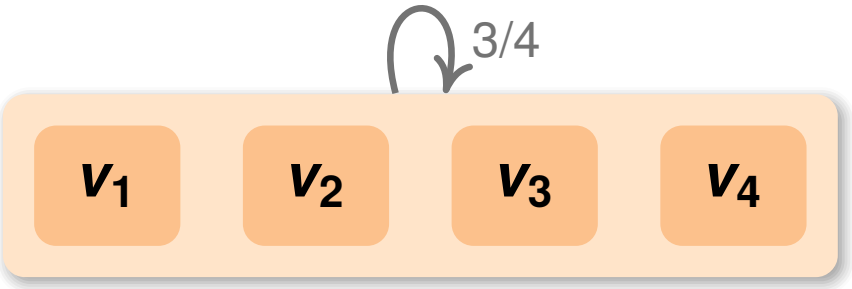


$\mathbf{Q}(v_4) =$
$\mathbf{Q}(v_5) =$
$\mathbf{Q}(v_6) =$
$\{\{v_4, v_5, v_6, v_7\}\}$

- 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
- Revised necessary property for safety:
  Need quorum intersection *after deleting malicious nodes*

$\longrightarrow$ E.g., reduces above diagram to one on previous slide

When is liveness impossible?



$\mathbf{Q}(v_1) = v_1$ 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 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.

- Necessary property for liveness:
  Failed nodes cannot be v-blocking for any correct node v
  - Equivalently, correct nodes must form a quorum
  - Otherwise, can't guarantee all honest nodes will remain live

When is liveness impossible?

$\mathbf{Q}(v_1) = v_1$ 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 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.

- Necessary property for liveness:
  Failed nodes cannot be v-blocking for any correct node v
  - Equivalently, correct nodes must form a quorum
  - Otherwise, can't guarantee all honest nodes will remain live

FBAS security

- Optimally safe FBAS guarantees safety for any set U of nodes iff:
  1. U is unanimously well behaved, and
  2. U enjoys quorum intersection after deleting all nodes not in U.
- Optimally live FBAS guarantees liveness for nodes in U iff:
  1. U fulfills the above requirements for safety, and
  2. U is a quorum—nodes outside U are not v-blocking for any $v \in U$.

Definition (intact)

Intact nodes are those that can be guaranteed safety and liveness.

- Result: quorum intersection gives one maximal set of intact nodes
  - Without quorum intersection, must conceptually delete nodes with bad quorum slices to set security goals

The Stellar consensus protocol (SCP)

- First FBA protocol, guarantees safety & liveness for intact nodes
- Also guarantees safety for nodes enjoying quorum intersection
  - Conservative quorum slices endanger liveness, not safety
- SCP does not optimize all axes, however:
  - Communication complexity (# messages/rounds)
  - Recovery from liveness failures (after reconfiguration)
  - Efficiency in the face of Byzantine network behavior
- The core of the protocol is *federated voting*
  - Nodes exchanges votes to agree on statements
  - Every vote includes the voter's quorum slices
  - Dynamically discover quorums while assembling votes

Federated voting

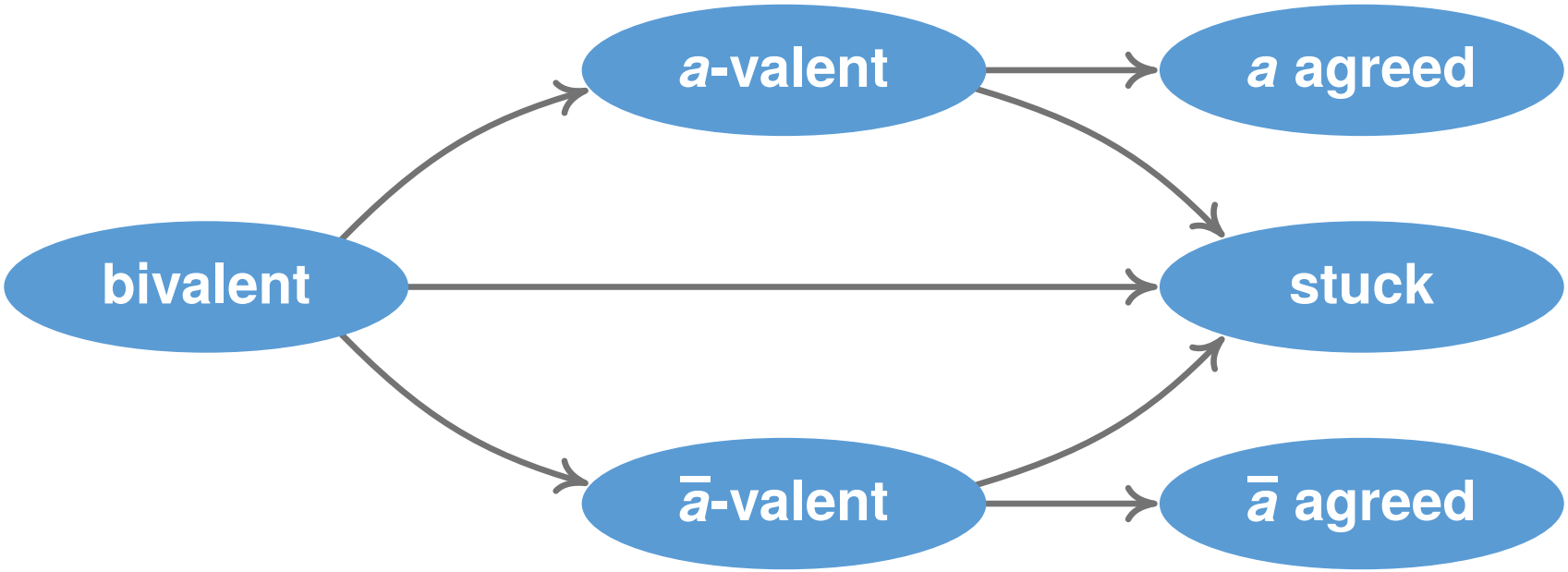
- Each node v can *vote* for a statement a provided
  1. a is valid and consistent with past statements v accepted, and
  2. v has never voted against a and promises it never will.

Definition (ratify)

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

- Result: well-behaved nodes enjoying quorum intersection cannot ratify contradictory statements
- Problem: intact node v may be unable to ratify a after others do
  - v 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
- Need a way to agree on statement *a* even after voting against it
- Need a way to know system has agreed on *a*
  - Don't assume *a* unless all honest nodes can eventually accept it
- Standard centralized voting techniques don't apply
  - No federated equivalent of seeing $f_S + f_L + 1$ identical votes

Accepting

Definition (accept)

Node v accepts a consistent statement a iff either:

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

- Intuition: say a v-blocking set claims to accept statement a
  - Either all are lying (so v is not intact), or a quorum accepted a
- Result: two intact nodes cannot accept contradictory statements
- Now a node can accept a statement it voted against!
- But still have two problems:
  - Still no guarantee all intact nodes can accept a statement
  - We've weakened safety for non-intact nodes with quorum intersection

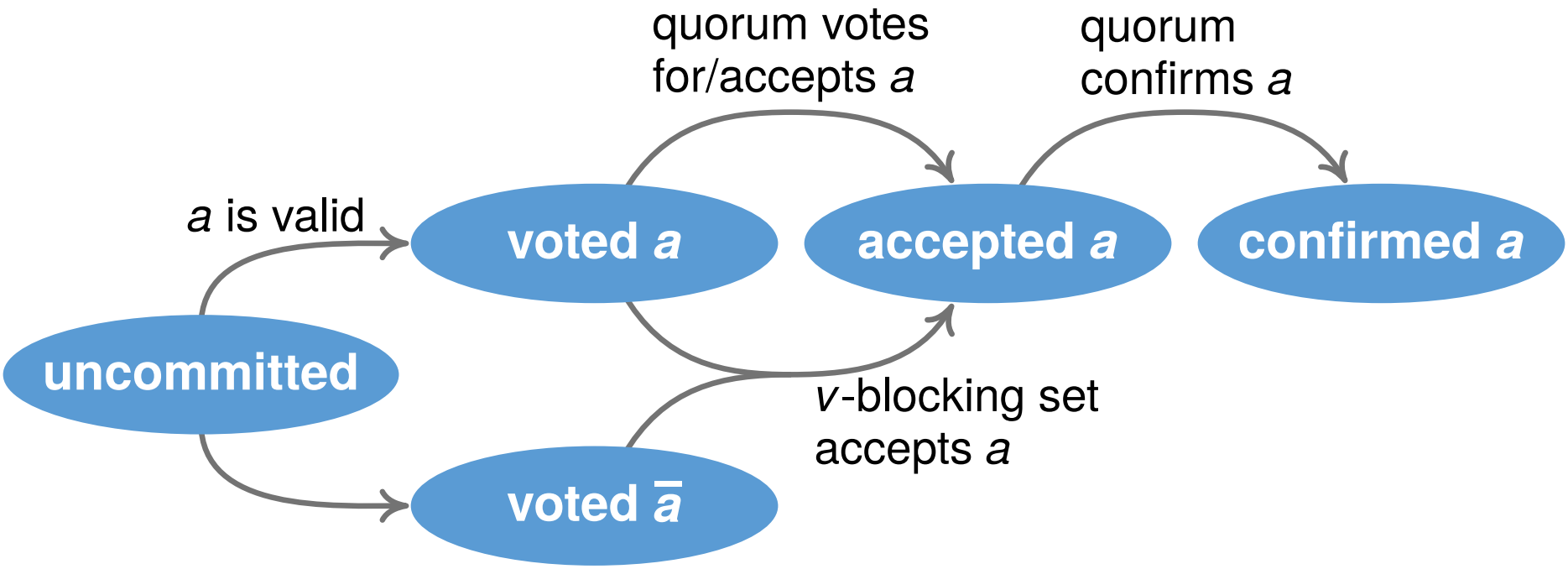
Confirmation

Definition (confirm)

A quorum U_a in an FBAS confirms a statement a by ratifying the statement "I accepted a." A node confirms a iff it is in such a quorum.

- Idea: Hold a second vote on the fact that the first vote succeeded
  - Intact nodes may vote against accepted statements
  - Won't vote against the *fact* they were accepted
- The fact that an intact node accepted a statement a is irrefutable
- Result 1: Confirmation provides ratification safety
- Result 2: Once a single intact node confirms a statement, all intact nodes can eventually confirm it

Summary of voting process



- Once a node *v* confirms *a*, it can safely act on *a* because
  - Nodes with quorum intersection will not contradict *a*
  - If *v* is intact, all nodes will eventually confirm *a*
- A node that locally confirms *a* knows system has agreen on *a*

From voting to consensus

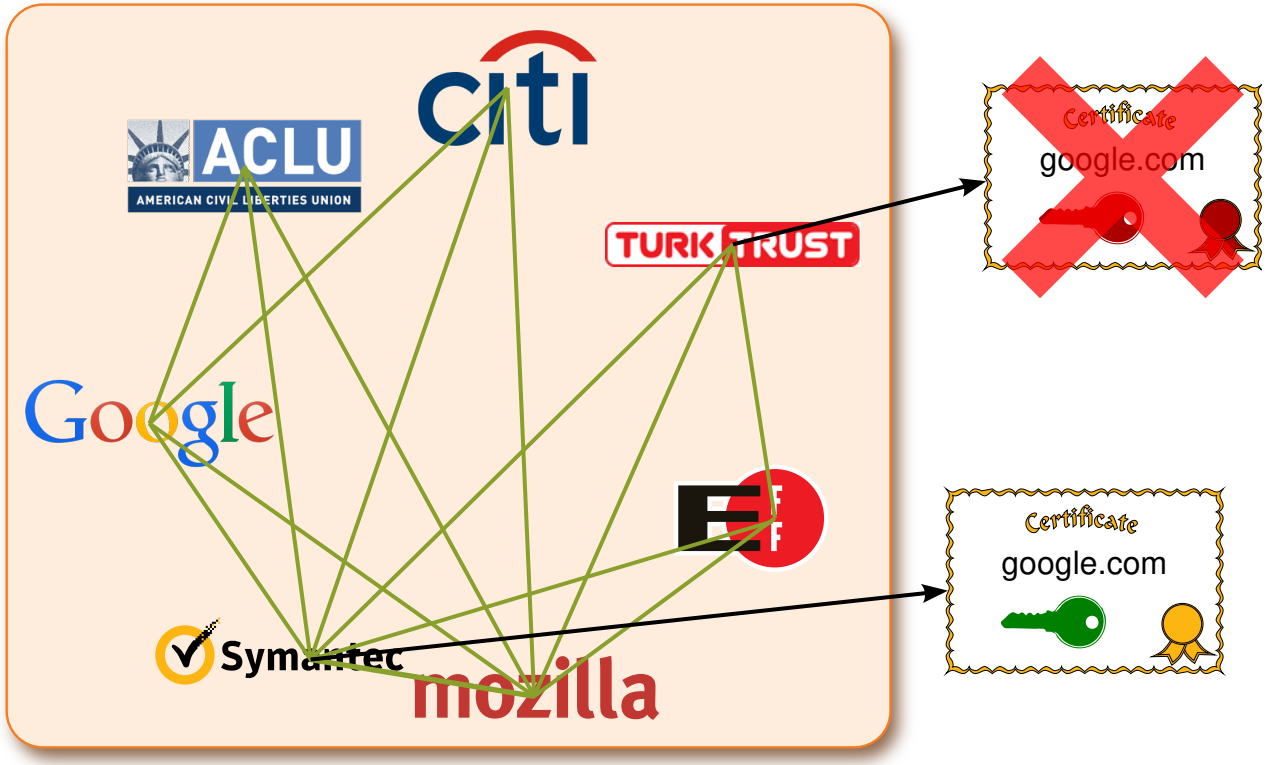
- Like Paxos, vote to *commit* or *abort* ballots
  - Each ballot is $\langle n, x \rangle$ where x is candidate consensus value
  - Committing $\langle n, x \rangle$ choses x as the value
- To avoid getting stuck, before voting to commit $\langle n, x \rangle$:
  - Agree to abort all $\langle n', x' \rangle < \langle n, x \rangle$ such that $x' \neq x$ (*prepare* $\langle n, x \rangle$)
  - If ballot $\langle n, x \rangle$ stuck, try again with $\langle n + 1, x \rangle$
- Disclaimer: not usually how Paxos is explained, but equivalent
  - Paxos embeds leader id in ballot, leader picks only one value
  - SCP embeds x directly in ballot to avoid need for leader
- Common case of protocol is two rounds of federated voting
  1. Agree to prepare a ballot (accept+confirm set of *aborts*)
  2. Agree to commit a ballot (accept+confirm a *commit*)

Summary of properties

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

- 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

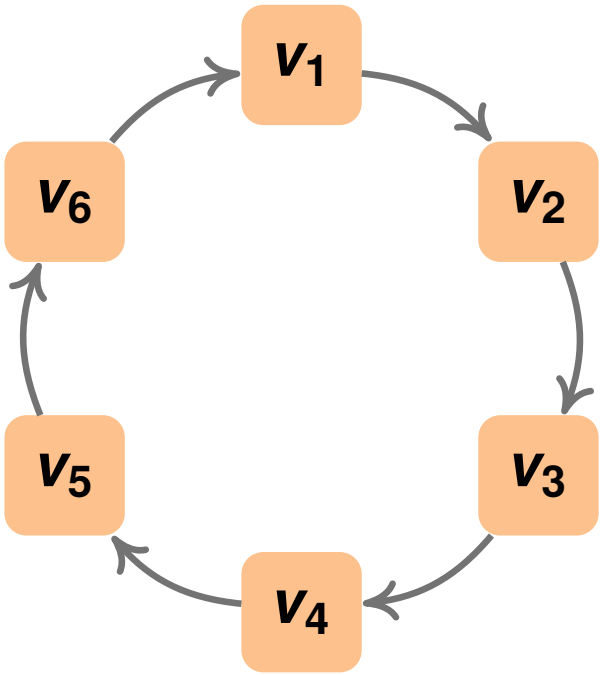


- Problem: One bad certificate authority (CA) undermines TLS
  - E.g., Turktrust issued fake Google certificates [ArsTechnica'13]
  - Even more important for end-to-end email encryption
- Can be addressed by auditing [certificate transparency], [CONIKS]
- SCP can increase confidence in the auditing process

Thank you

https://www.stellar.org/, [SCP]

Cyclic quorum slice example



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

- 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