The Stellar Consensus Protocol

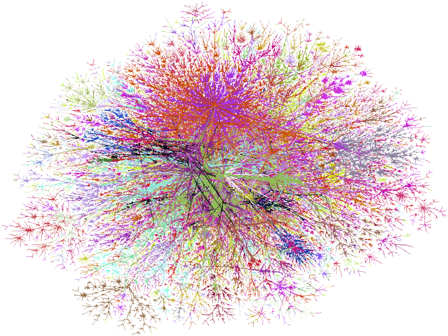
Byzantine agreement from the Internet hypothesis

David Mazières, Giuliano Losa, Eli Gafni

Stellar Development Foundation
UCLA

Tuesday February 19, 2019

The Internet hypothesis



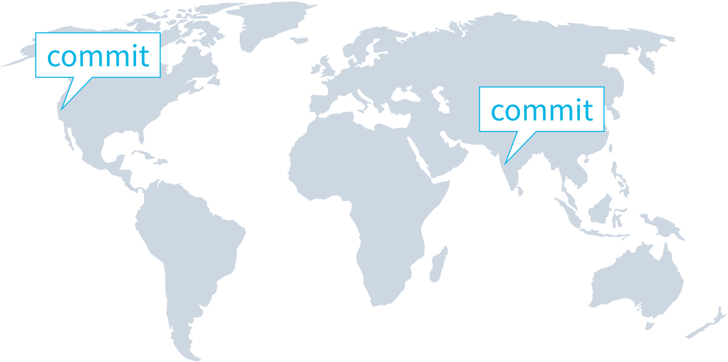
How did we end up with *The* Internet?

- Structure results from individual peering & transit relationships
- Transitively, everyone wants to talk to everyone
- An ISP can't sell access to alternate IP network

Could systems similarly agree on *The* global consensus state?

- Hypothesis: other relationships are like Internet peering
- I.e., no one can afford to disagree with the rest of the world

Transitive global agreement



Imagine you agree on transaction only when your friends do

- They agree only when their friends do, and so forth

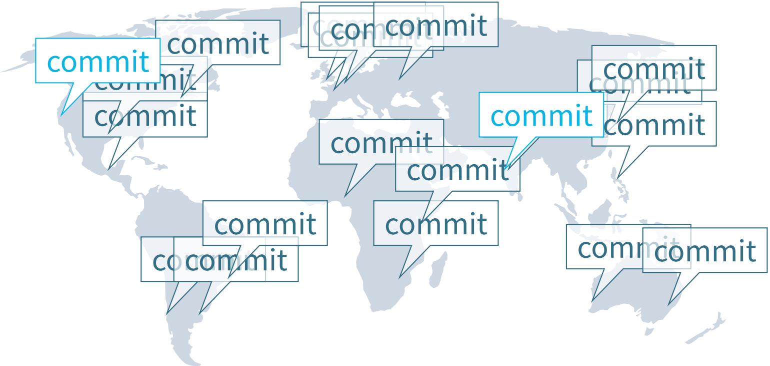
Guarantees you agree with everyone you depend on

- Through transitivity anyone you'd ever care about will agree

Approach currently used by multi-asset Stellar blockchain

- Atomically trade through intermediary assets without trusting issuer

Transitive global agreement



Imagine you agree on transaction only when your friends do

- They agree only when their friends do, and so forth

Guarantees you agree with everyone you depend on

- Through transitivity anyone you'd ever care about will agree

Approach currently used by multi-asset Stellar blockchain

- Atomically trade through intermediary assets without trusting issuer

What about mining?

Definition (Mining)

Obtaining cryptocurrency tokens as a reward for making digital transactions harder to reverse.

Most blockchains get consensus through mining

- Makes token creation and consensus a package deal
- Weaker guarantees than asynchronous Byzantine agreement

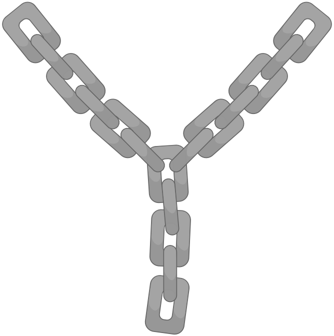
What if you want *only* want consensus?

- Trade digitized real-world assets backed by known counterparties
- Don't want or need to create new cryptocurrency
- Don't want to pay for mining (directly or indirectly)

An approach: piggyback on an existing mined blockchain

- E.g., Colored coins, ERC-20 tokens, ...

Blockchain forks



In July 2016, Ethereum executed an irregular state change

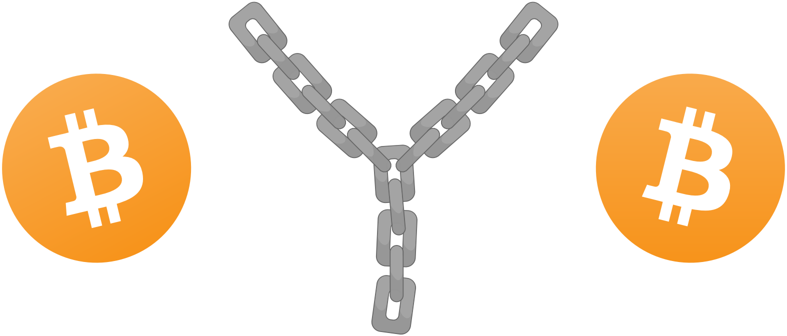
- 85% of miners opted to bail out DAO contract (lost $50M to bug)
- Remaining miners kept original rules, became *Ethereum Classic*

In August 2017, Bitcoin split in two (Bitcoin/Bitcoin cash)

In November 2018, Bitcoin cash split again (ABC/SV)

What would this mean for token issuers?

Blockchain forks



In July 2016, Ethereum executed an irregular state change

- 85% of miners opted to bail out DAO contract (lost $50M to bug)
- Remaining miners kept original rules, became *Ethereum Classic*

In August 2017, Bitcoin split in two (Bitcoin/Bitcoin cash)

In November 2018, Bitcoin cash split again (ABC/SV)

What would this mean for token issuers?

Blockchain forks

USD

USD

In July 2016, Ethereum executed an irregular state change

- 85% of miners opted to bail out DAO contract (lost $50M to bug)
- Remaining miners kept original rules, became *Ethereum Classic*

In August 2017, Bitcoin split in two (Bitcoin/Bitcoin cash)

In November 2018, Bitcoin cash split again (ABC/SV)

What would this mean for token issuers?

Mining is scary for digital asset issuers

Mining is anonymous

- Anyone with sufficient resources can extend or fork history
- Can't even *name* branch if no policy difference (just dueling miners)

Yet mining rewards insufficient to secure fiat-currency tokens

- And crypto futures let bad miners hedge positions before attack

Non-financial (geo-political) incentives to disrupt blockchain

Global Byzantine agreement would allow Fed, ECB to issue trillions in assets and enjoy liquid markets between assets

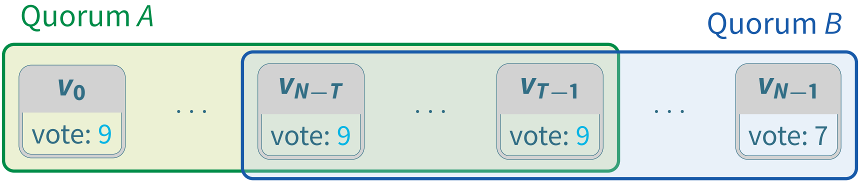
Outline

Background: centrally-mandated quorums

Quorums from the Internet hypothesis

Stellar Consensus Protocol

Fail-stop systems



Suppose you have N nodes, some of which might crash

- Each node can *vote* for at most one value

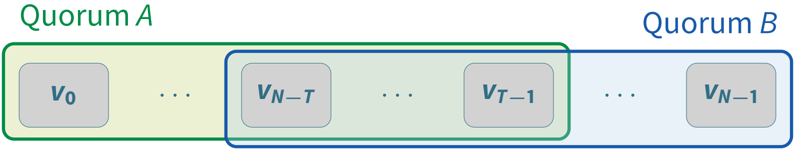
Pick a quorum size $T > N/2$

Only one value can receive unanimous quorum vote

- $T > N/2 \implies$ Any two quorums intersect
- If Quorum A votes for a, Quorum B either votes for a or isn't unanimous

Voting is a key tool for ensuring agreement in consensus

Byzantine failure



What if faulty nodes can act arbitrarily ("Byzantine failure")

- Now faulty nodes can issue conflicting votes

For safety, want at most one (valid) value to get a quorum

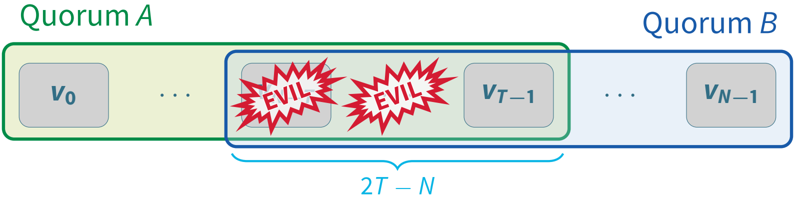
- Requires: # failures $\leq f_S = 2T - N - 1$
- Hence, any two quorums share a *non-faulty* node, can't lose history

For liveness, want at least some hope of a unanimous quorum

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

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

Byzantine failure



What if faulty nodes can act arbitrarily ("Byzantine failure")

- Now faulty nodes can issue conflicting votes

For safety, want at most one (valid) value to get a quorum

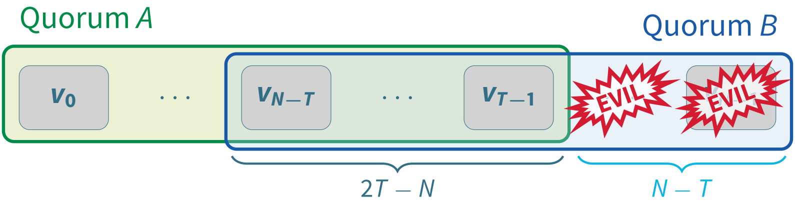
- Requires: # failures $\leq f_S = 2T - N - 1$
- Hence, any two quorums share a *non-faulty* node, can't lose history

For liveness, want at least some hope of a unanimous quorum

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

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

Byzantine failure



What if faulty nodes can act arbitrarily ("Byzantine failure")

- Now faulty nodes can issue conflicting votes

For safety, want at most one (valid) value to get a quorum

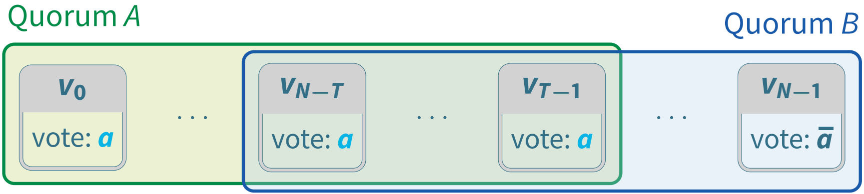
- Requires: # failures $\leq f_S = 2T - N - 1$
- Hence, any two quorums share a *non-faulty* node, can't lose history

For liveness, want at least some hope of a unanimous quorum

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

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

Stuck votes



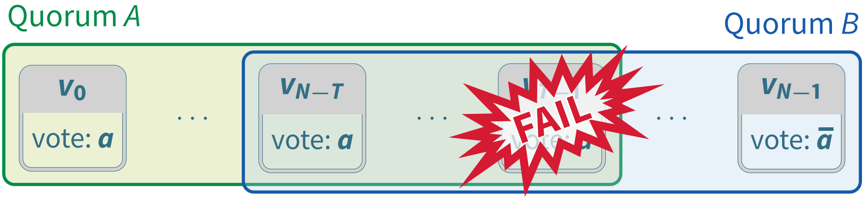
Say Quorum *A* unanimously votes for statement *a*

- Any contradictory statement ($\bar{a}$) cannot receive a quorum
- So we say the system is *a-valent*

Two reasons voting alone doesn't solve consensus

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

Stuck votes



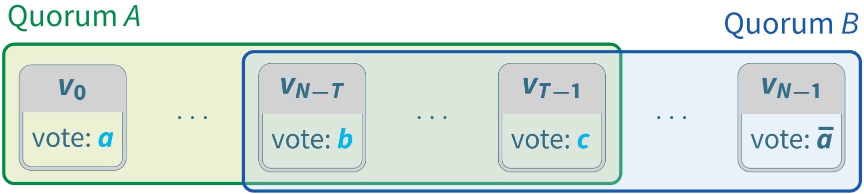
Say Quorum *A* unanimously votes for statement *a*

- Any contradictory statement ($\bar{a}$) cannot receive a quorum
- So we say the system is *a-valent*

Two reasons voting alone doesn't solve consensus

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

Stuck votes



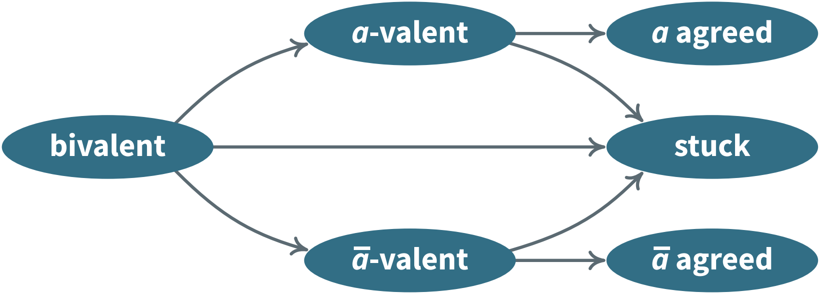
Say Quorum A unanimously votes for statement a

- Any contradictory statement ($\bar{a}$) cannot receive a quorum
- So we say the system is *a-valent*

Two reasons voting alone doesn't solve consensus

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

When has a vote succeeded?



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

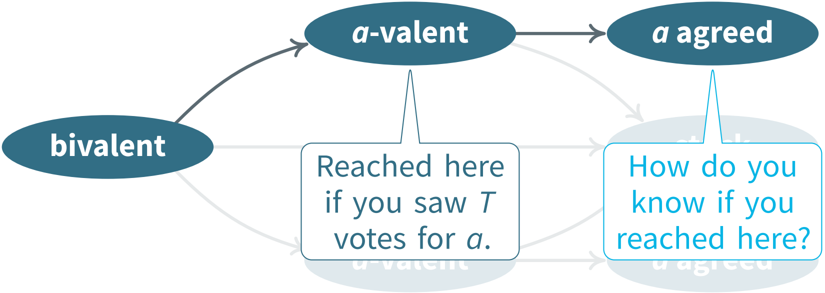
Suppose $f_S + 1$ nodes all claim to have seen T votes for a

- 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

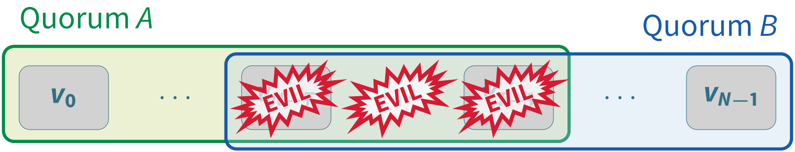
Suppose $f_S + 1$ nodes all claim to have seen T votes for a

- 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 $\Longrightarrow$ none stuck $\Longrightarrow$ a agreed

When has a vote succeeded?



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

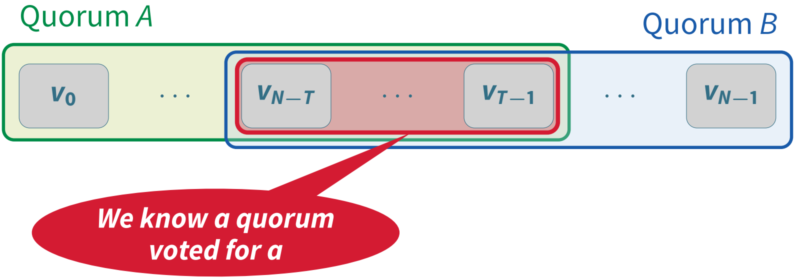
Suppose $f_S + 1$ nodes all claim to have seen T votes for a

- 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 $\Longrightarrow$ none stuck $\Longrightarrow$ a agreed

When has a vote succeeded?



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

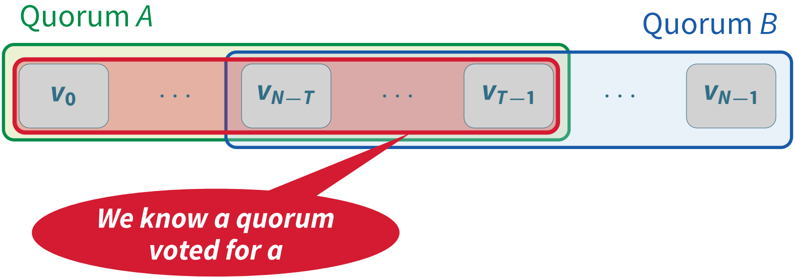
Suppose $f_S + 1$ nodes all claim to have seen T votes for a

- 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 $\Longrightarrow$ 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 $\Longrightarrow$ none stuck $\Longrightarrow$ a agreed

5 tricks of Byzantine agreement protocols

view 1: | op_1 | op_2 | op_3 | op_4 | ? |

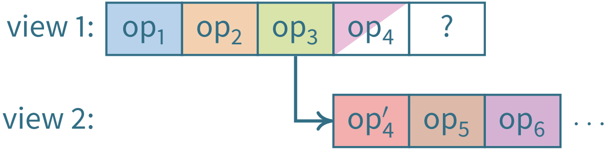
Ensure some probability of non-faulty nodes voting identically

- Assume partial synchrony (Raft, PBFT, SCP, . . .)
  assumption often used for some sort of leader election
- Everyone flips a coin [Ben Or] (finite chance of flipping same coin)
- Common coin [Rabin] (Mostéfaoui, HoneyBadger, Algorand, . . .)

Survive stuck votes

- Vote on what goes in log entries, then vote on which log entries matter (Viewstamped Replication, PBFT, Raft, . . .)
- Vote that value is “safe” before really voting for it (Paxos, SCP, . . .)

5 tricks of Byzantine agreement protocols



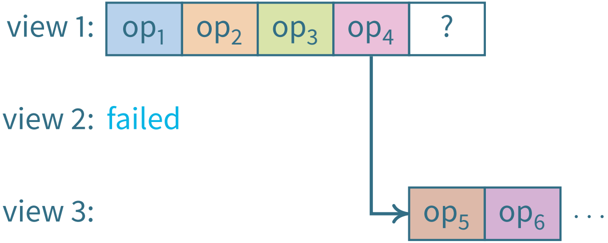
Ensure some probability of non-faulty nodes voting identically

- Assume partial synchrony (Raft, PBFT, SCP, ...)
  assumption often used for some sort of leader election
- Everyone flips a coin [Ben Or] (finite chance of flipping same coin)
- Common coin [Rabin] (Mostéfaoui, HoneyBadger, Algorand, ...)

Survive stuck votes

- Vote on what goes in log entries, then vote on which log entries matter (Viewstamped Replication, PBFT, Raft, ...)
- Vote that value is "safe" before really voting for it (Paxos, SCP, ...)

5 tricks of Byzantine agreement protocols



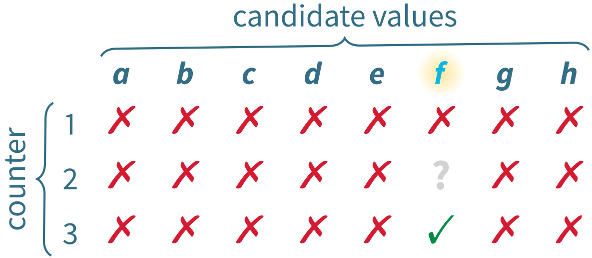
Ensure some probability of non-faulty nodes voting identically

- Assume partial synchrony (Raft, PBFT, SCP, ...)
  assumption often used for some sort of leader election
- Everyone flips a coin [Ben Or] (finite chance of flipping same coin)
- Common coin [Rabin] (Mostéfaoui, HoneyBadger, Algorand, ...)

Survive stuck votes

- Vote on what goes in log entries, then vote on which log entries matter (Viewstamped Replication, PBFT, Raft, ...)
- Vote that value is "safe" before really voting for it (Paxos, SCP, ...)

5 tricks of Byzantine agreement protocols



Ensure some probability of non-faulty nodes voting identically

- Assume partial synchrony (Raft, PBFT, SCP, . . .)
  assumption often used for some sort of leader election
- Everyone flips a coin [Ben Or] (finite chance of flipping same coin)
- Common coin [Rabin] (Mostéfaoui, HoneyBadger, Algorand, . . .)

Survive stuck votes

- Vote on what goes in log entries, then vote on which log entries matter (Viewstamped Replication, PBFT, Raft, . . .)
- Vote that value is “safe” before really voting for it (Paxos, SCP, . . .)

Outline

Federated Byzantine Agreement (FBA)

Based on Byzantine agreement—consensus among closed group

- But majority-based Byzantine agreement vulnerable to *Sybil attacks*
- Idea: defeat Sybil attacks with decentralized quorum selection

Each node v picks one or more sets of nodes called *quorum slices*

- v considers each slice important enough to speak for whole network
- Choice based on real-world identities
  E.g., put issuers of all tokens you care about in all of your slices

Definition (Federated Byzantine Agreement System)

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

Definition (Quorum)

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

Federated Byzantine Agreement

Based on Byzantine agreement—consensus among closed group

- But majority-based Byzantine agreement vulnerable to *Sybil attacks*
- Idea: defeat Sybil attacks with decentralized quorum selection

Each node v picks one or more sets of nodes called *quorum slices*

- v considers each slice important enough to speak for whole network
- Choice based on real-world identities
  E.g., put issuers of all tokens you care about in all of your slices

Definition (Federated Byzantine Agreement System)

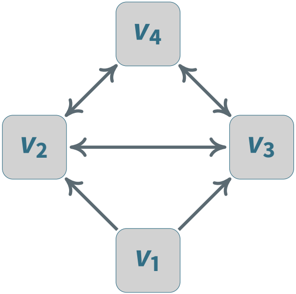
An FBAS is of a a set of nodes $\mathbf{V}$ and a quorum function $\mathbf{Q}$, where $\mathbf{Q}(v)$ is the set slices chosen by node v.

Definition (Quorum)

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

Definition (Quorum)

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



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

Visualize quorum slice dependencies with arrows

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

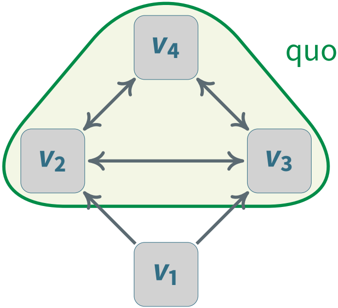
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

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

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

Definition (Quorum)

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



quorum for v_2, v_3, v_4

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

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

Visualize quorum slice dependencies with arrows

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

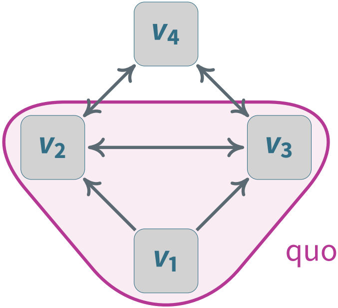
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

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

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

Definition (Quorum)

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



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

quorum slice for v_1, but not a quorum

Visualize quorum slice dependencies with arrows

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

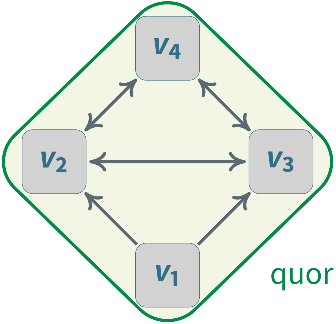
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

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

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

Definition (Quorum)

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



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

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

quorum for $v_1, \ldots, v_4$

Visualize quorum slice dependencies with arrows

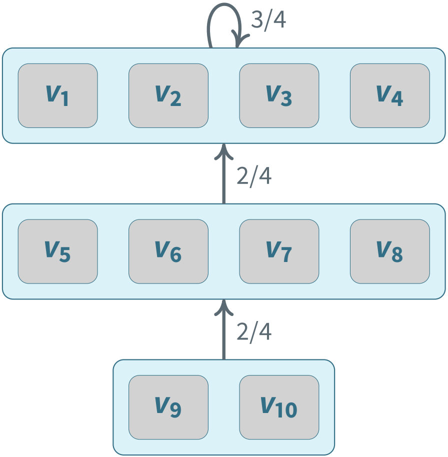
v_2, v_3, v_4 **is a quorum—contains a slice of each member**

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

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

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

Tiered quorum slice example



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

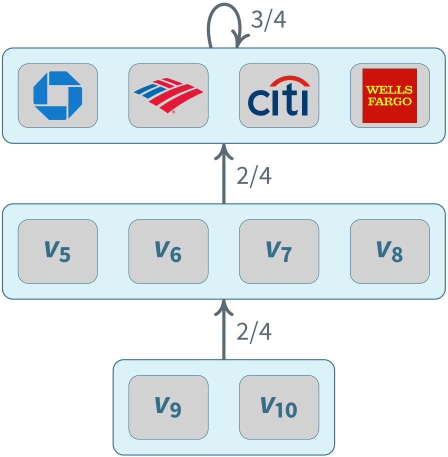
Middle tier: slice is self + any two top tier nodes

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

Like the Internet, no central authority appoints top tier

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

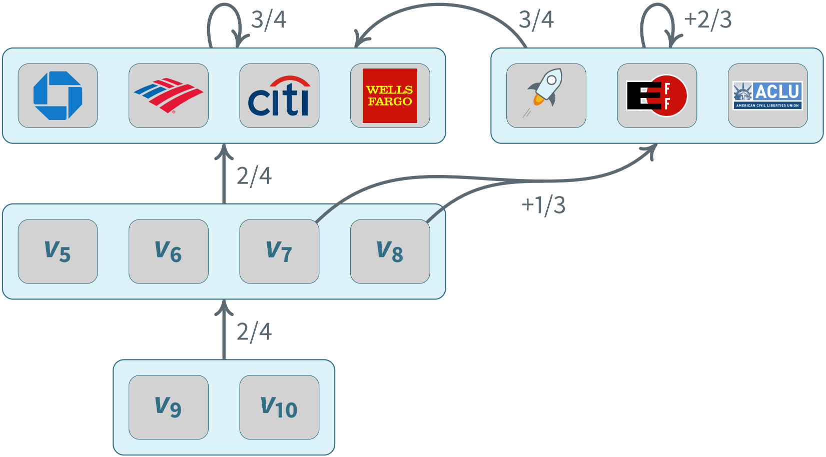
Tiered quorum slice example



Like the Internet, no central authority appoints top tier

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

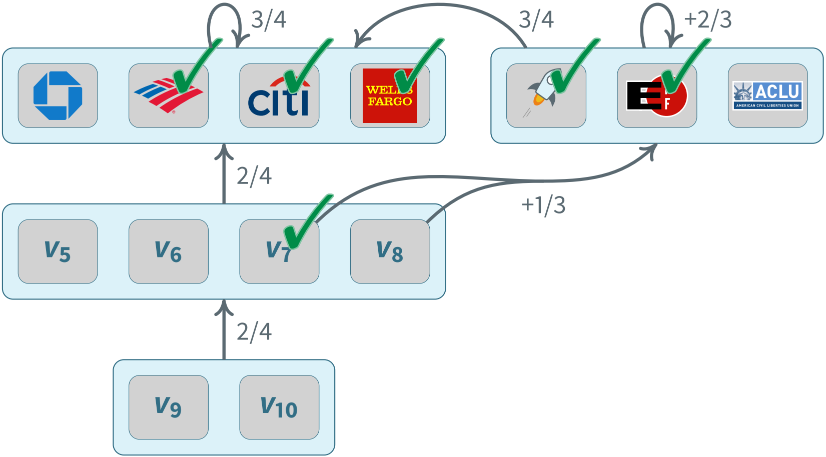
Tiered quorum slice example



Like the Internet, no central authority appoints top tier

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

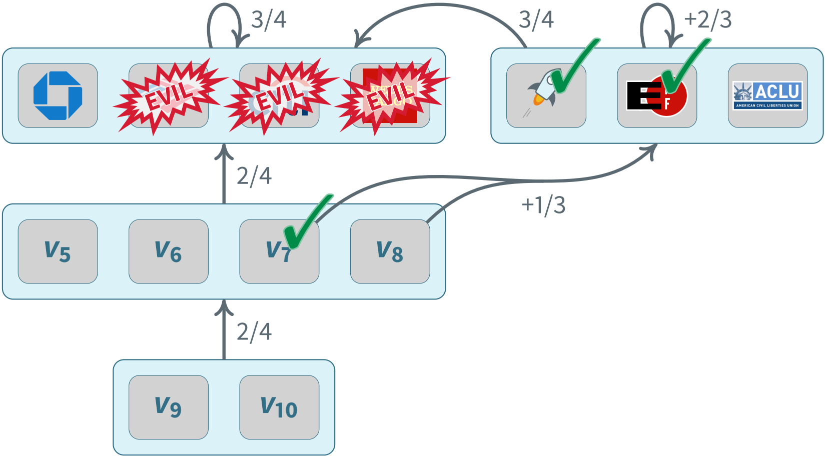
Tiered quorum slice example



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

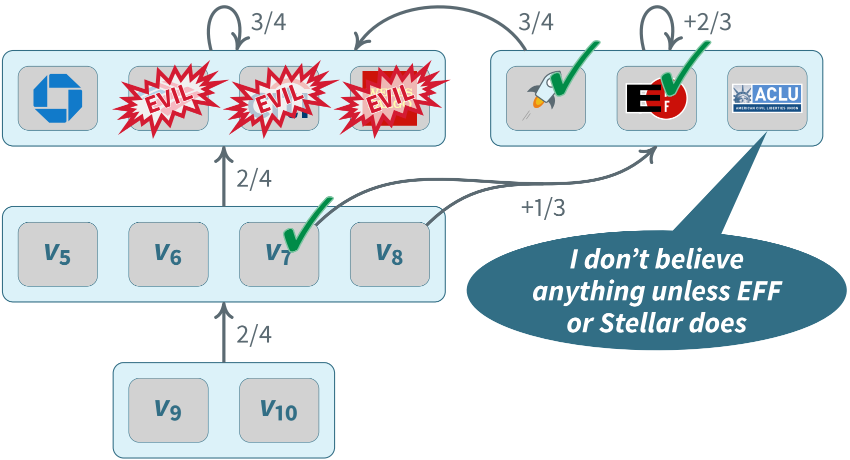
Tiered quorum slice example



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

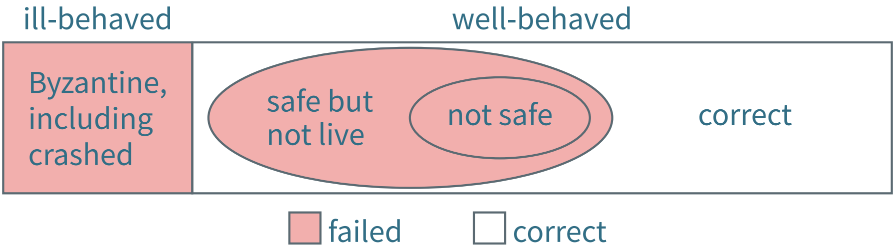
Tiered quorum slice example



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

FBAS failure is per-node



Each node is either *well-behaved* or *ill-behaved* (faulty)

All ill-behaved nodes have *failed*

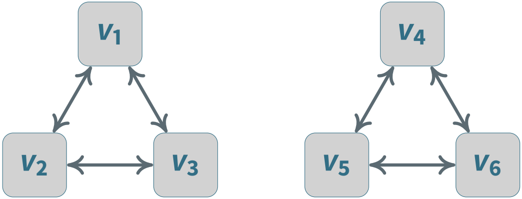
Enough ill-behaved nodes can cause well-behaved nodes to fail

- 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

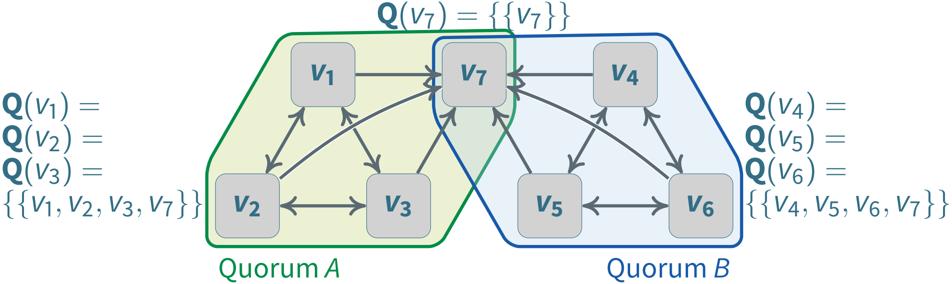
- Each can make progress with no communication from the other
- No way to guarantee the two externalize consistent statements

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?



What if two quorums intersect only at faulty nodes?

- No way to guarantee safety when nodes not intertwined

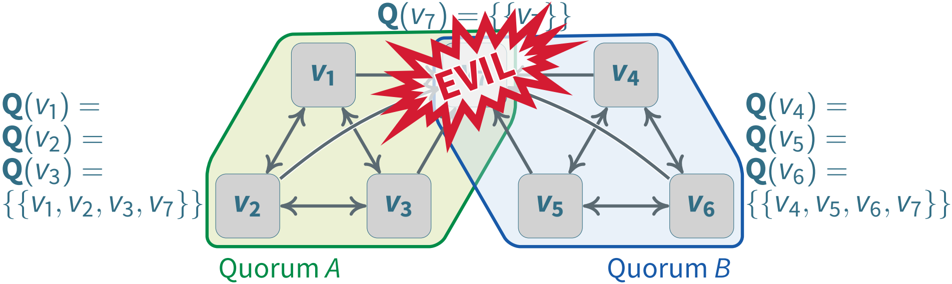
Definition (Quorum-revised)

A quorum $U \subseteq \mathbf{V}$ is a set of nodes that encompasses at least one slice of each of its well-behaved members (ignoring faulty slices).

Definition (intertwined)

Nodes v_1 and v_2 are intertwined iff every quorum of v_1 intersects every quorum of v_2 at a well-behaved node.

What about Byzantine failures?



What if two quorums intersect only at faulty nodes?

- No way to guarantee safety when nodes not intertwined

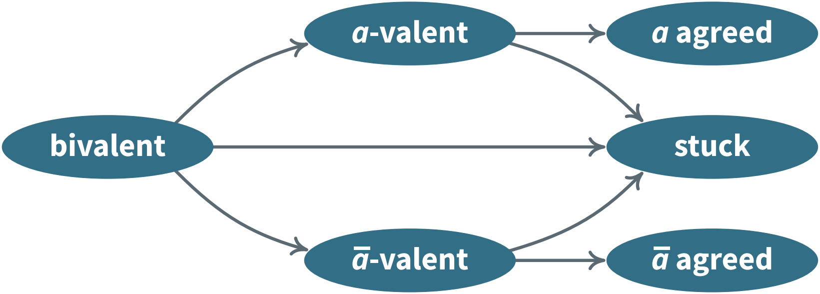
Definition (Quorum-revised)

A quorum $U \subseteq \mathbf{V}$ is a set of nodes that encompasses at least one slice of each of its well-behaved members (ignoring faulty slices).

Definition (intertwined)

Nodes v_1 and v_2 are intertwined iff every quorum of v_1 intersects every quorum of v_2 at a well-behaved node.

Adapting Byzantine agreement to slices



Slice-based quorums yield same outcomes as T-of-N

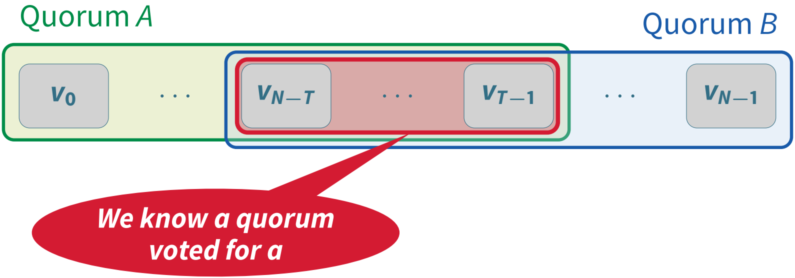
Apply the same reasoning as in centralized voting?

- Premise was whole system couldn't fail; now failure is per node
- Cannot assume correctness of quorums you don't belong to

Any place a classical Byzantine agreement protocol waits for $f_S + 1$ nodes, there is no equivalent with quorum slices

- First-hand quorums are the only way to know system a-valent
- Once you vote for $\bar{a}$, can't be in a quorum voting a

Adapting Byzantine agreement to slices



Slice-based quorums yield same outcomes as T-of-N

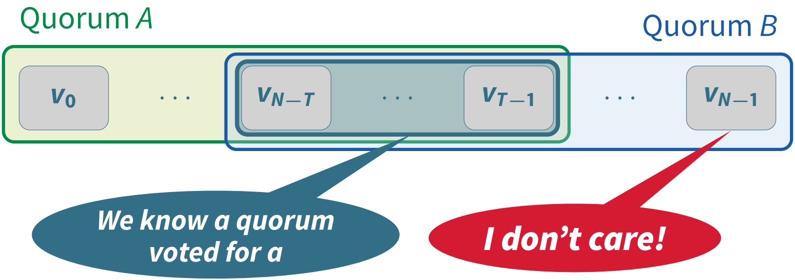
Apply the same reasoning as in centralized voting?

- Premise was whole system couldn't fail; now failure is per node
- Cannot assume correctness of quorums you don't belong to

Any place a classical Byzantine agreement protocol waits for $f_S + 1$ nodes, there is no equivalent with quorum slices

- First-hand quorums are the only way to know system a-valent
- Once you vote for $\overline{a}$, can't be in a quorum voting a

Adapting Byzantine agreement to slices



Slice-based quorums yield same outcomes as T-of-N

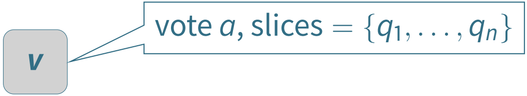
✗ **Apply the same reasoning as in centralized voting? No!**

- Premise was whole system couldn't fail; now failure is per node
- Cannot assume correctness of quorums you don't belong to

Any place a classical Byzantine agreement protocol waits for $f_S + 1$ nodes, there is no equivalent with quorum slices

- First-hand quorums are the only way to know system a-valent
- Once you vote for $\bar{a}$, can't be in a quorum voting a

Federated voting



Nodes unilaterally join the system w/o permission

- Nodes are named by their public signature keys

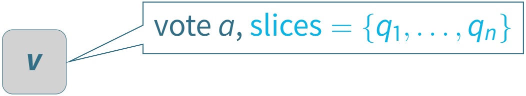
Each node chooses one or more quorum slices

- In theory, could be arbitrary sets
- To represent compactly, use recursive k-of-n threshold specification

Nodes exchanges signed vote messages to agree on statements

- Every vote specifies quorum slices
- Allows dynamic quorum discovery while assembling votes

Federated voting



Nodes unilaterally join the system w/o permission

- Nodes are named by their public signature keys

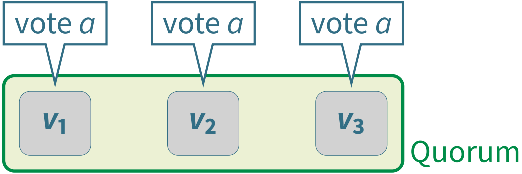
Each node chooses one or more quorum slices

- In theory, could be arbitrary sets
- To represent compactly, use recursive *k*-of-*n* threshold specification

Nodes exchanges signed vote messages to agree on statements

- Every vote specifies quorum slices
- Allows dynamic quorum discovery while assembling votes

Ratifying 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.

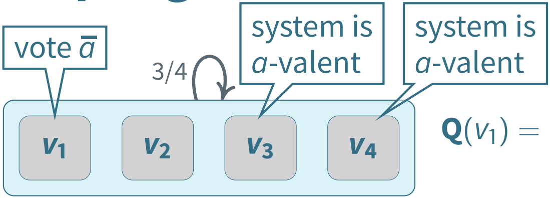
Well-behaved nodes cannot vote for contradictory statements

Theorem: Intertwined nodes won't ratify contradictory statements

Problem: even in a well-behaved quorum, some node v may be unable to ratify some statement a after other nodes do

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

Accepting statements



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

What if one node in each of v_1's slices says system is a-valent?

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

Definition (accept)

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

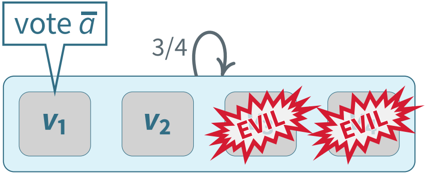
1. **A quorum containing v each either voted for or accepted a, or**
2. **Each of v's quorum slices has a node claiming to accept a.**

#2 lets a node accept a statement after voting against it, but...

1. Still no guarantee all supposedly live nodes can accept a statement
2. Intertwined nodes can accept diverging statements

(Intuition: we wanted $f_S + 1$ notes, but have to settle for $f_L + 1$)

Accepting statements



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

What if one node in each of v_1's slices says system is a-valent?

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

Definition (accept)

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

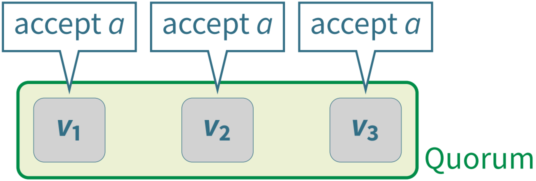
1. **A quorum containing v each either voted for or accepted a, or**
2. **Each of v's quorum slices has a node claiming to accept a.**

#2 lets a node accept a statement after voting against it, but…

1. Still no guarantee all supposedly live nodes can accept a statement
2. Intertwined nodes can accept diverging statements

(Intuition: we wanted $f_S + 1$ notes, but have to settle for $f_L + 1$)

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.

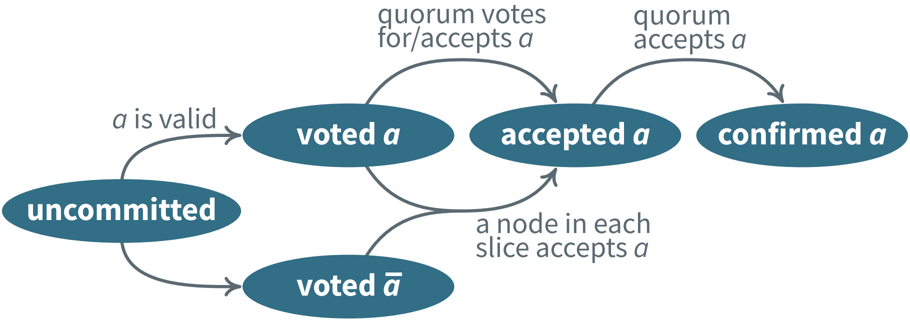
Solves problem 2 (suboptimal safety) w. straight-up ratification

Solves problem 1 (live nodes unable to accept)

- Nodes with liveness guarantee may vote against accepted statements
- Won't vote against the *fact* that those statements were accepted
- Hence, can't get split confirmation vote

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

Summary of federated voting process



A vote might still get stuck

But if any node *v* confirms *a*, vote not stuck, system agrees on *a*

- If intertwined, well-behaved nodes can't contradict *a*
- If *v* in intact quorum, whole quorum will eventually confirm *a*

Outline

Background: centrally-mandated quorums

Quorums from the Internet hypothesis

Stellar Consensus Protocol

Stellar Consensus Protocol [SCP,spec]

Guarantees safety when nodes are *intertwined*

- This is optimal—if not intertwined, no protocol can guarantee safety
- I.e., you may regret your choice of quorum slices, but you won't regret choosing SCP over other slice-based Byzantine agreement protocols

Guarantees liveness for an *intact* quorum

- Intact = non-faulty quorum that enjoys quorum intersection after removing non-intact nodes
- Weaker notions of intact are possible (e.g., intertwined quorum), but seem to have limitations:
  - Must prove each statement you make (requires large history, messages),
  - Can't change quorums slices mid-protocol, and/or
  - Must prove some some set of nodes necessary for a node's quorum

SCP's two standard tricks

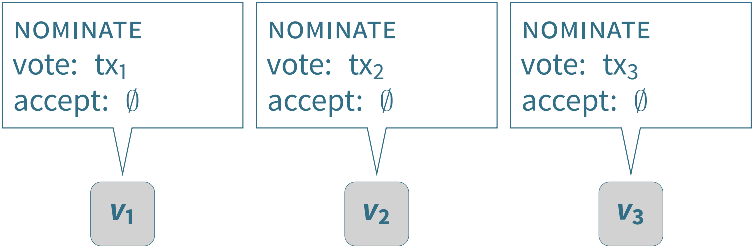
Employ a synchronous FBA protocol as a *nomination* subroutine

- Challenge: can't have leader election w/o agreement on who exist
- Nomination eventually converges and all nodes get same value
- But okay if agreement fails—happens when synchrony violated

Use *balloting* to deem a value "safe" before voting for it (~Paxos)

- Ensures intact nodes never gets stuck
- Guarantees termination under partial synchrony w. Byzantine failure if you repeat nomination on timeout
- Otherwise, termination guaranteed w. crash-stop nodes or after manually removing malicious nodes from slices
- Should remove bad nodes from anyway, so single-nomination faster

Strawman nomination



Idea: every node reliably broadcasts proposed value (~Bracha)

Every node votes for its own proposed value

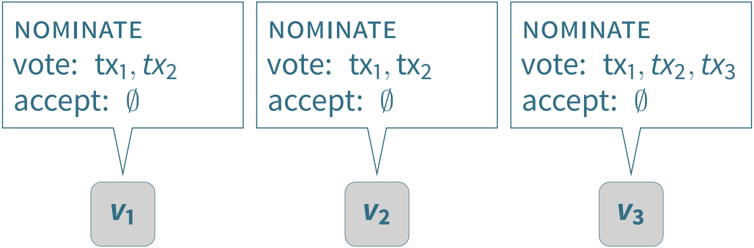
Every node also votes for values it learns from others

Eventually, nodes accept and confirm nominated values

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

Deterministically combine all confirmed nominated values

Strawman nomination



Idea: every node reliably broadcasts proposed value (~Bracha)

Every node votes for its own proposed value

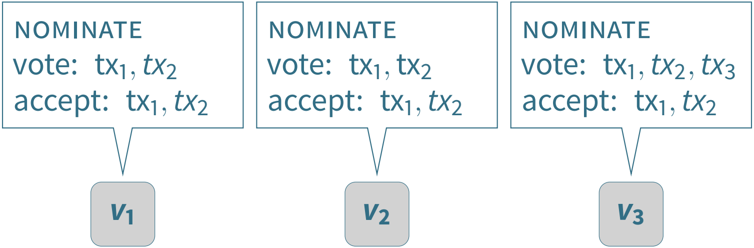
Every node also votes for values it learns from others

Eventually, nodes accept and confirm nominated values

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

Deterministically combine all confirmed nominated values

Strawman nomination



Idea: every node reliably broadcasts proposed value (~Bracha)

Every node votes for its own proposed value

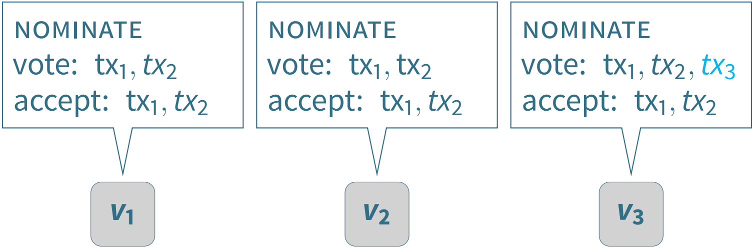
Every node also votes for values it learns from others

Eventually, nodes accept and confirm nominated values

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

Deterministically combine all confirmed nominated values

Strawman nomination



Idea: every node reliably broadcasts proposed value (~Bracha)

Every node votes for its own proposed value

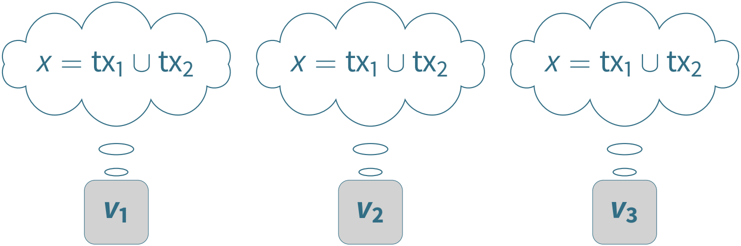
Every node also votes for values it learns from others

Eventually, nodes accept and confirm nominated values

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

Deterministically combine all confirmed nominated values

Strawman nomination



Idea: every node reliably broadcasts proposed value (~Bracha)

Every node votes for its own proposed value

Every node also votes for values it learns from others

Eventually, nodes accept and confirm nominated values

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

Deterministically combine all confirmed nominated values

Properties of nomination strawman

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

+ Once an intact node confirms a value nominated, all will

- Direct consequence of federated voting

+ A bounded number of values can get nominated

- Need votes from intact nodes to nominate values
- Can only cast bounded number of votes before confirming first value

+ Nomination guaranteed to converge eventually

- Attacker perturb bounded number of times—each time "consuming" an as yet unconfirmed value nominated by intact nodes

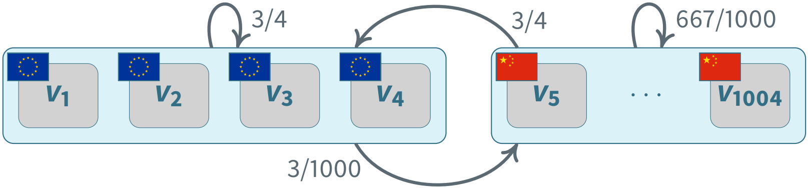
– Never know when nomination has converged—have to guess

- Inevitable given partial synchrony, but still unfortunate

– Lots of values floating around wastes bandwidth, computation

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

Reducing # nominated values



Choose leader pseudorandomly by highest H(PubKey‖round)**?**

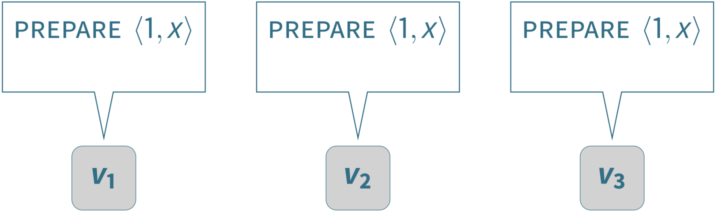
- Works for Algorand because coins quantify clout
- Here risks censorship from organizations/countries with more nodes

Select leaders based on local slice weight & hashes:

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

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

Balloting



Use Paxos-like Balloting for asynchronous agreement

Define ballot as a pair $b = \langle n, x \rangle$

- n is a ballot counter (allows arbitrarily many ballots)
- x is a candidate value
- Conceptually vote to commit and abort individual ballots

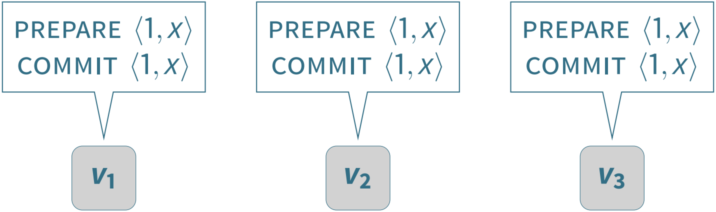
Must *prepare* a ballot before voting to commit

- Requires aborting lesser conflicting ballots before voting to commit

Balloting mechanics:

- Prepare $\langle 1, x \rangle$ by confirming $\{\text{abort } \langle n, x' \rangle \mid \langle n, x' \rangle < \langle 1, x \rangle \wedge x' \neq x\}$
- Vote and confirm "commit $\langle 1, x \rangle$"; output value x

Balloting



Use Paxos-like Balloting for asynchronous agreement

Define ballot as a pair $b = \langle n, x \rangle$

- n is a ballot counter (allows arbitrarily many ballots)
- x is a candidate value
- Conceptually vote to commit and abort individual ballots

Must *prepare* a ballot before voting to commit

- Requires aborting lesser conflicting ballots before voting to commit

Balloting mechanics:

- Prepare $\langle 1, x \rangle$ by confirming $\{\text{abort } \langle n, x' \rangle \mid \langle n, x' \rangle < \langle 1, x \rangle \wedge x' \neq x\}$
- Vote and confirm "commit $\langle 1, x \rangle$"; output value x

Balloting example

candidate values

counter	a	b	c	d	e	f	g	h
1	?	?	?	?	?	?	?	?
2	?	?	?	?	?	?	?	?
3	?	?	?	?	?	?	?	?

- ? = bivalent
- ✗ = aborted
- ⃠ = stuck
- ✓ = commited

0. **Initially, all ballots are bivalent**
1. **Prepare** $\langle 1, g\rangle$ **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. $\langle 2, f\rangle$ **seems stuck; agree** $\langle 3, f\rangle$ **prepared and vote to commit it**
4. **Confirm commit** $\langle 3, f\rangle$ **and externalize** f
   - At this point nobody cares that $\langle 2, f\rangle$ is stuck

Key invariant: all committed & stuck ballots have same value

Before vote to commit, only one “non-aborted rectangle”

Balloting example

candidate values

counter	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*
1	✗	✗	✗	✗	✗	✗	?	?
2	?	?	?	?	?	?	?	?
3	?	?	?	?	?	?	?	?

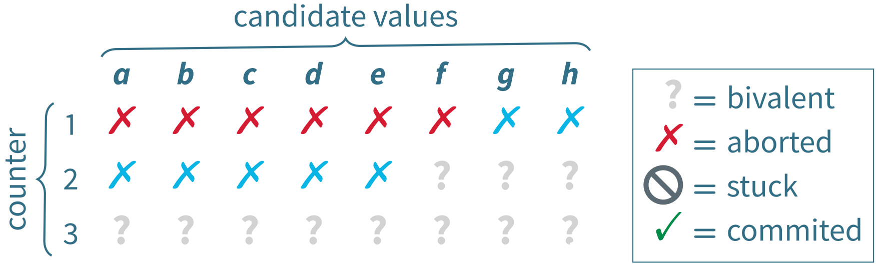
- ? = bivalent
- ✗ = aborted
- ⃠ = stuck
- ✓ = commited

0. **Initially, all ballots are bivalent**
1. **Prepare $\langle 1, g \rangle$ 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. **$\langle 2, f \rangle$ seems stuck; agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **Confirm commit $\langle 3, f \rangle$ and externalize f**
    - At this point nobody cares that $\langle 2, f \rangle$ is stuck

Key invariant: all committed & stuck ballots have same value

Before vote to commit, only one “non-aborted rectangle”

Balloting example

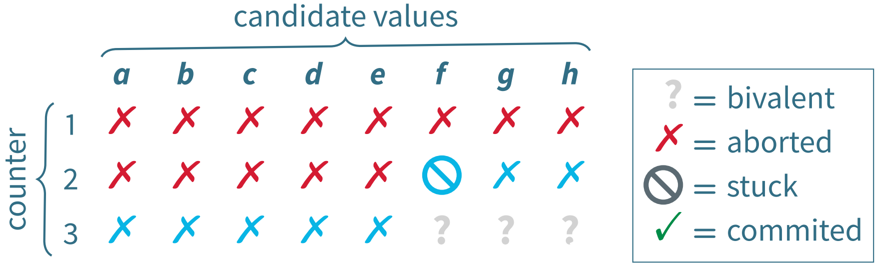


0. **Initially, all ballots are bivalent**
1. **Prepare $\langle 1, g \rangle$ and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ seems stuck; agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **Confirm commit $\langle 3, f \rangle$ and externalize f**
   - At this point nobody cares that $\langle 2, f \rangle$ is stuck

Key invariant: all committed & stuck ballots have same value

Before vote to commit, only one "non-aborted rectangle"

Balloting example

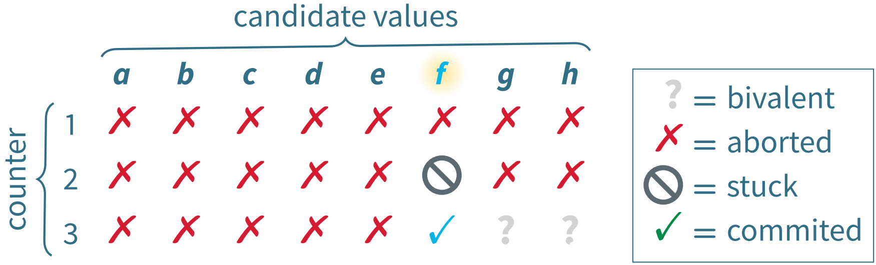


0. **Initially, all ballots are bivalent**
1. **Prepare** $\langle 1, g \rangle$ **and vote to commit it**
2. **Lose vote on** $\langle 1, g \rangle$**; agree** $\langle 2, f \rangle$ **prepared and vote to commit it**
3. $\langle 2, f \rangle$ **seems stuck; agree** $\langle 3, f \rangle$ **prepared and vote to commit it**
4. **Confirm commit** $\langle 3, f \rangle$ **and externalize** f
   - At this point nobody cares that $\langle 2, f \rangle$ is stuck

Key invariant: all committed & stuck ballots have same value

Before vote to commit, only one “non-aborted rectangle”

Balloting example

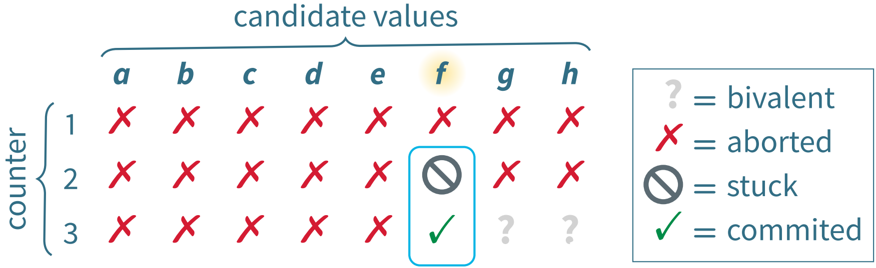


0. **Initially, all ballots are bivalent**
1. **Prepare $\langle 1, g \rangle$ and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ seems stuck; agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **Confirm commit $\langle 3, f \rangle$ and externalize f**
    - At this point nobody cares that $\langle 2, f \rangle$ is stuck

Key invariant: all committed & stuck ballots have same value

Before vote to commit, only one "non-aborted rectangle"

Balloting example



0. **Initially, all ballots are bivalent**
1. **Prepare** $\langle 1, g \rangle$ **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. $\langle 2, f \rangle$ **seems stuck; agree** $\langle 3, f \rangle$ **prepared and vote to commit it**
4. **Confirm commit** $\langle 3, f \rangle$ **and externalize** f
   - At this point nobody cares that $\langle 2, f \rangle$ is stuck

Key invariant: all committed & stuck ballots have same value

Before vote to commit, only one "non-aborted rectangle"

SCP state and messages

```
struct SCPPrepare {
  // Current ballot you are trying to ratify prepared
  SCPBallot ballot;

  // Most recent accepted prepared ballot if any
  // (highest ballot of "non-aborted rectangle")
  SCPBallot *prepared;

  // Ballot counter below which everything aborted
  // (lowest counter of "non-aborted rectangle")
  uint32 aCounter;

  // Counter of most recent confirmed prepared ballot or 0
  // (only if value same as ballot field)
  uint32 hCounter;

  // Oldest ballot counter node is voting to commit or 0
  // (ballot value must be same as ballot field)
  uint32 cCounter;
};
```

Complete prepare state only 5 counters, 2 values

Self-clocking ballot counters

Bump ballot counter on increasing timeout

- Standard trick for terminating with partial synchrony

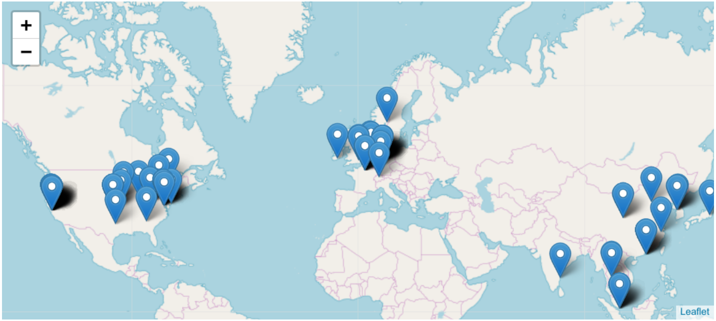
Need intact nodes to spend increasing time on same ballot

- Arm timer only when you say quorum at same or higher counter
- Immediately increase counter if blocking set higher
- C.f. DLS partial synchrony round model

Two options for updating value when updating ballot counter

1. Use value from highest confirmed prepared ballot, else latest nomination output (terminates w. fail-stop nodes & partial synchrony)
2. Re-run nomination protocol at each counter (terminates w. Byzantine failure & partial synchrony)

SCP in the real world



Used by the Stellar DEX/payment network

- ~120 nodes today, achieving consensus every ~5 seconds
- In use today for trading a wide range of tokens
- Better for real-world assets than mining—issuers can run validators

Other blockchains using SCP: MobileCoin, NCNT

Applications beyond blockchain: certificate transparency, firmware transparency, delegated namespaces

Questions?

www.stellar.org