Stellar
Fast and secure global payments

David Mazières

Monday, May 13, 2019
Say you want to send $1 from U.S. to a customer of bank$_4$ in India. Bank$_4$ may have a *nastro* account at a European bank$_3$.
- Will disburse 60.0 INR in exchange for 0.93 EUR on deposit at bank$_3$.

Some bank$_2$ may have *nastro* accounts at bank$_3$ and your bank$_1$.
- Offers 0.93 EUR at bank$_3$ in exchange for 1.00 USD at bank$_1$.

Goal: implement quick, secure, atomic, irreversible payment.
Internet payments

<table>
<thead>
<tr>
<th>Offeror</th>
<th>Bid</th>
<th>Ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>bank₄</td>
<td>60.0 INR@bank₄</td>
<td>0.93 EUR@bank₃</td>
</tr>
</tbody>
</table>

Say you want to send $1 from U.S. to a customer of bank₄ in India. Bank₄ may have a *nostro* account at a European bank₃.

- Will disburse 60.0 INR in exchange for 0.93 EUR on deposit at bank₃.

Some bank₂ may have *nostro* accounts at bank₃ and your bank₁.

- Offers 0.93 EUR at bank₃ in exchange for 1.00 USD at bank₁.

Goal: implement quick, secure, atomic, irreversible payment.
Internet payments

<table>
<thead>
<tr>
<th>Offeror</th>
<th>Bid</th>
<th>Ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>bank₄</td>
<td>60.0 INR@bank₄</td>
<td>0.93 EUR@bank₃</td>
</tr>
<tr>
<td>bank₂</td>
<td>0.93 EUR@bank₃</td>
<td>1.00 USD@bank₁</td>
</tr>
</tbody>
</table>

Say you want to send $1 from U.S. to a customer of bank₄ in India. Bank₄ may have a nostrro account at a European bank₃.
- Will disburse 60.0 INR in exchange for 0.93 EUR on deposit at bank₃.

Some bank₂ may have nostrro accounts at bank₃ and your bank₁.
- Offers 0.93 EUR at bank₃ in exchange for 1.00 USD at bank₁.

Goal: implement quick, secure, atomic, irreversible payment.
Internet payments

<table>
<thead>
<tr>
<th>Offeror</th>
<th>Bid</th>
<th>Ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>bank_4</td>
<td>60.0 INR@bank_4</td>
<td>0.93 EUR@bank_3</td>
</tr>
<tr>
<td>bank_2</td>
<td>0.93 EUR@bank_3</td>
<td>1.00 USD@bank_1</td>
</tr>
</tbody>
</table>

Say you want to send $1 from U.S. to a customer of bank_4 in India.
Bank_4 may have a *nostro* account at a European bank_3.
- Will disburse 60.0 INR in exchange for 0.93 EUR on deposit at bank_3.

Some bank_2 may have *nostro* accounts at bank_3 and your bank_1.
- Offers 0.93 EUR at bank_3 in exchange for 1.00 USD at bank_1.

Goal: implement quick, secure, atomic, irreversible payment.
Is this a job for blockchain?
What blockchain really gives us

1. Coin distribution
   - Distribute new tokens or “cryptocoins” while limiting supply

2. Irreversible transactions*
   - Can securely exchange or transfer purely digital tokens

What if bank$_1$, bank$_4$ issue digital tokens representing USD, INR?
   - Would blockchain give us irreversible fiat money transactions?

*under certain assumptions
What blockchain really gives us

1. Coin distribution
   - Distribute new tokens or “cryptocoins” while limiting supply

2. Irreversible transactions*
   - Can securely exchange or transfer purely digital tokens

What if bank₁, bank₄ issue digital tokens representing USD, INR?
   - Would blockchain give us irreversible fiat money transactions?

*under certain assumptions
What blockchain really gives us

1. Coin distribution
   - Distribute new tokens or “cryptocoins” while limiting supply

2. Irreversible transactions*
   - Can securely exchange or transfer purely digital tokens

What if bank$_1$, bank$_4$ issue digital *tokens* representing USD, INR?
   - Would blockchain give us irreversible fiat money transactions?

*under certain assumptions
Bitcoin’s key insight: Mining

**Definition (Mining)**
Obtaining cryptocurrency tokens as a reward for making digital transactions harder to reverse.

Mutually-reinforcing solution to coin distribution+irreversibility

Proof-of-work: solve hard-to-compute, easy-to-verify puzzle
- Also: proof-of-stake, proof-of-storage, proof-of-elapsed-time, ...

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 re-split (ABC/SV)

What does this mean for token issuers?
- Could bank₁ be liable for twice as many digital dollars as it issued?
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/Btcikcoin cash)
- In November 2018, Bitcoin cash re-split (ABC/SV)

What does this mean for token issuers?
- Could bank_1 be liable for twice as many digital dollars as it issued?
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 re-split (ABC/SV)

What does this mean for token issuers? - Could bank₁ be liable for twice as many digital dollars as it issued?
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

Goal: let issuers create $1T+ in assets with liquid markets between assets and without fear of double-redemption
Today’s talk: Internet-level consensus

Don’t need mining for digital money tokens backed by banks
Instead, wait for token counterparties to commit transactions
- They commit only when their counterparties do, and so forth
Guarantees you agree with everyone you depend on
- E.g., never diverge from bank₁, bank₄ where you redeem USD and INR
- Through transitivity anyone you’d ever care about will agree
Today’s talk: Internet-level consensus

Don’t need mining for digital money tokens backed by banks.
Instead, wait for token counterparties to commit transactions:
  - They commit only when their counterparties do, and so forth.

Guarantees you agree with everyone you depend on:
  - E.g., never diverge from bank₁, bank₄ where you redeem USD and INR.
  - Through transitivity anyone you’d ever care about will agree.
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

**Hypothesis: financial network is similarly interconnected**
- Leverage transitive connectivity to agree on *the* global consensus state
- So no one can afford to disagree with the rest of the world
Outline

Consensus background

Quorums from the Internet hypothesis

Stellar Consensus Protocol
The consensus problem

Goal: For multiple agents to agree on an output value

Each agent starts with an input value
- In payments, value is an ordered batch of transactions to execute

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
  - In payments, value is an ordered batch of transactions to execute

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
- In payments, value is an ordered batch of transactions to execute

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 should aspire to two safety properties:

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

A consensus protocol should aspire to a liveness property:

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

A consensus protocol would ideally be fault-tolerant:

- 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.
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 9, Quorum $B$ either votes for 9 or isn’t unanimous

Voting is a key tool for ensuring agreement in consensus
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: \( \# \text{ failures} \leq f_S = 2T - N - 1 \)
- 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: \( \# \text{ failures} \leq f_L = N - T \) (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 $\alpha$
- We say the system is $\alpha$-valent because no contradictory statement ($\bar{\alpha}$) can receive a quorum, so $\alpha$ is the only possible output

**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

**So voting is a useful tool, but a bad consensus algorithm**
- Guarantees agreement but not liveness
Stuck votes

Say Quorum A unanimously votes for statement $\alpha$

- We say the system is $\alpha$-valent because no contradictory statement ($\bar{\alpha}$) can receive a quorum, so $\alpha$ is the only possible output

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

So voting is a useful tool, but a bad consensus algorithm

- Guarantees agreement but not liveness
Stuck votes

Say Quorum A unanimously votes for statement $a$
- We say the system is $a$-valent because no contradictory statement ($\bar{a}$) can receive a quorum, so $a$ is the only possible output

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

So voting is a useful tool, but a bad consensus algorithm
- Guarantees agreement but not liveness
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 $\alpha$-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 $\alpha$-valent
- All correct nodes believe system $\alpha$-valent $\implies$ none stuck $\implies \alpha$ agreed
When has a vote succeeded?

- Reached here if you saw $T$ votes for $a$. How do you know if you reached here?

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 $f_L > f_L$ fail, system loses liveness (0 correct nodes in whole system)
   - If $f_L \leq f_L$, $f_S + 1 \geq f_S + 1$ 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
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
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

view 1: \( \text{op}_1 \text{ op}_2 \text{ op}_3 \text{ op}_4 \ ? \)

view 2: failed

view 3: \( \text{op}_5 \text{ op}_6 \ ? \)

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

<table>
<thead>
<tr>
<th>Candidate values</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>2</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>?</td>
<td>✗</td>
</tr>
<tr>
<td>3</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

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

Consensus background

Quorums from the Internet hypothesis

Stellar Consensus Protocol
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 FBA System is of a a set of nodes $V$ and a quorum function $Q$, where $Q(v)$ is the set slices chosen by node $v$.

### Definition (Quorum)

A quorum $U \subseteq V$ is a set of nodes containing $\geq 1$ slice of each non-faulty member: $\forall v \in U, v \in$ non-faulty $\implies \exists q \in Q(v), q \subseteq U$
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 *FBA System* is of a a set of nodes \( V \) and a quorum function \( Q \),
where \( Q(v) \) is the set slices chosen by node \( v \).

**Definition (Quorum)**

A quorum \( U \subseteq V \) is a set of nodes containing \( \geq 1 \) slice of each
non-faulty member: \( \forall v \in U, v \in \text{non-faulty} \implies \exists q \in Q(v), q \subseteq U \)
**Definition (Quorum)**

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

![Diagram showing quorum slice dependencies with arrows]

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

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

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

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

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

$v_1, v_2, v_3$ 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 \( U \subseteq V \) is a set of nodes that encompasses at least one slice of each of its members: \( \forall v \in U, \exists q \in Q(v) \text{ such that } q \subseteq U \)

\[
Q(v_1) = \{\{v_1, v_2, v_3\}\}
\]
\[
Q(v_2) = Q(v_3) = Q(v_4) = \{\{v_2, v_3, v_4\}\}
\]

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 \( v_1 \)
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 \textit{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
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
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
I don’t believe anything unless EFF or Stellar does

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
FBA failure is per-node

Each node is either faulty or non-faulty

All faulty nodes have failed

Enough faulty nodes can cause non-faulty nodes to fail
- Bad: non-faulty nodes blocked from progress (safe but not live)
- Worse: non-faulty nodes in divergent states (not safe)

Non-faulty nodes are correct if they have not failed
What is necessary to guarantee safety?

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 FBA system enjoys **quorum intersection** when every two quorums share at least one non-faulty node.
What about Byzantine failures?

Who is right if faulty nodes cause disagreement?
- No protocol can guarantee agreement between nodes not intertwined
- Disagreement may be fine if, e.g., Quorum B is a Sybil attack
- Quorum slices express where nodes need agreement

**Definition (intertwined)**

Nodes $v_1$ and $v_2$ are intertwined iff every quorum of $v_1$ intersects every quorum of $v_2$ at a non-faulty node.
What about Byzantine failures?

Who is right if faulty nodes cause disagreement?
- No protocol can guarantee agreement between nodes not intertwined
- Disagreement may be fine if, e.g., Quorum B is a Sybil attack
- Quorum slices express where nodes need agreement

**Definition (intertwined)**
Nodes $v_1$ and $v_2$ are **intertwined** iff every quorum of $v_1$ intersects every quorum of $v_2$ at a non-faulty 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

**Quorum A**

- $v_0$
- $\ldots$
- $v_{N-T}$
- $\ldots$
- $v_{T-1}$
- $\ldots$
- $v_{N-1}$

**Quorum B**

**We know a quorum voted for a**

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

We know a quorum voted for $a$

I don’t care!

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 $\overline{a}$, can’t be in a quorum voting $a$
Federated voting

vote \( a \), slices = \{q_1, \ldots, q_n\}

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 exchange 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 $a$.

Non-faulty nodes cannot vote for contradictory statements
Theorem: Intertwined nodes won’t ratify contradictory statements
Problem: even in a non-faulty 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

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 non-faulty 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

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 non-faulty 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 \(f_L + 1\))
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 non-faulty quorum confirms $a$, all will
Summary of federated voting process

A vote might still get stuck

But if any node $\nu$ confirms $a$, vote not stuck, system agrees on $a$

- If intertwined, non-faulty nodes can’t contradict $a$
- If $\nu$ in intact quorum, whole quorum will eventually confirm $a$
Outline

Consensus background

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 is fully intertwined even if all non-intact nodes are faulty
- 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 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. fail-stop nodes or after manually removing malicious nodes from slices
- Should remove bad nodes from slices anyway
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 can perturb bounded number of times—each “consumes” an
    as-yet-unconfirmed value nominated by intact node

– Never know when nomination has converged—have to guess
  - Inevitable given FLP impossibility, 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 $H(\text{PubKey} \parallel \text{round})$?
- Works for Algorand because coins quantify clout
- Here risks censorship from organizations/countries with more nodes

Select leaders based on local slice weight & hashes:

\[
\begin{align*}
\text{weight}(v) &= \text{fraction of local quorum slices containing v} \\
\text{neighbors}(\text{round}) &= \{ v \mid H_1(\text{round} \parallel v) < h_{\text{max}} \cdot \text{weight}(v) \} \\
\text{priority}(\text{round}, v) &= H_2(\text{round} \parallel v)
\end{align*}
\]
- 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
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 \( \{ \text{abort} \langle n, x' \rangle \mid \langle n, x' \rangle < \langle 1, x \rangle \land x' \neq x \} \)
- 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 $\{ \text{abort } \langle n, x' \rangle \mid \langle n, x' \rangle < \langle 1, x \rangle \land x' \neq x \}$
- Vote and confirm “commit $\langle 1, x \rangle$”; output value $x$
Balloting example

<table>
<thead>
<tr>
<th>candidate values</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>candidate values</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>candidate values</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

### Key invariant: all committed & stuck ballots have same value

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

---

### Candidate values

- `?` = bivalent
- `x` = aborted
- `=` = stuck
- `✓` = committed

### Example:

0. Initially, all ballots are bivalent

1. Prepare ⟨1, g⟩ and vote to commit it

2. Lose vote on ⟨1, g⟩; agree ⟨2, f⟩ prepared and vote to commit it

3. ⟨2, f⟩ seems stuck; agree ⟨3, f⟩ prepared and vote to commit it

4. Confirm commit ⟨3, f⟩ and externalize f

   - At this point nobody cares that ⟨2, f⟩ is stuck

---

**0. Initially, all ballots are bivalent**

**1. Prepare ⟨1, g⟩ and vote to commit it**

**2. Lose vote on ⟨1, g⟩; agree ⟨2, f⟩ prepared and vote to commit it**

**3. ⟨2, f⟩ seems stuck; agree ⟨3, f⟩ prepared and vote to commit it**

**4. Confirm commit ⟨3, f⟩ and externalize f**

   - At this point nobody cares that ⟨2, f⟩ is stuck

Key invariant: all committed & stuck ballots have same value
Balloting example

candidate values

\[
\begin{array}{cccccccc}
  & a & b & c & d & e & f & g & h \\
1 & \times & \times & \times & \times & \times & \times & \times & \times \\
2 & \times & \times & \times & \times & \times & \times & \times & \times \\
3 & \times & \times & \times & \times & \times & ? & ? & ? \\
\end{array}
\]

\[\bar{?} = \text{bivalent}\]
\[\times = \text{aborted}\]
\[\bar{O} = \text{stuck}\]
\[\checkmark = \text{commited}\]

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

\[
\begin{array}{cccccccc}
  a & b & c & d & e & f & g & h \\
 1 & x & x & x & x & x & x & x \\
 2 & x & x & x & x & x & x & x \\
 3 & x & x & x & x & x & x & x \\
\end{array}
\]

? = bivalent
x = aborted
 Bulldozer symbol = stuck
✓ = committed

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

<table>
<thead>
<tr>
<th>candidate values</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

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”
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

Choose value for next ballot as follows:
- Use value from highest confirmed prepared ballot, if any
- Otherwise, use latest nomination output
SCP in the real world

Used by the Stellar DEX/payment network
- ~147 nodes today, achieving consensus every ~5 seconds
- Primary adopters have been MTOs
- But many non-currency anchors: securitized real estate, commercial bonds, carbon credits, …

Other blockchains using SCP: MobileCoin, NCNT, Pi
Questions?

www.stellar.org
### Strawman: Two-phase commit

<table>
<thead>
<tr>
<th>Buy</th>
<th>Sell</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.93 EUR@bank$_2$</td>
<td>1.00 USD@bank$_1$</td>
</tr>
<tr>
<td>60.0 INR@bank$_4$</td>
<td>0.93 EUR@bank$_3$</td>
</tr>
</tbody>
</table>

#### Decompose payment into a series of trades
- Effectively exchanging deposits at one bank for ones at another

#### Combine them into atomic transaction

#### Use well-known two-phase commit protocol across 4 banks
- Send transaction to every institution concerned
- Commit only if all participants unanimously vote to do so
- E.g., bank$_2$ can abort transaction if its offer no longer valid
## Strawman: Two-phase commit

### atomic transaction

<table>
<thead>
<tr>
<th>Buy</th>
<th>Sell</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.93 EUR@bank$_2$</td>
<td>1.00 USD@bank$_1$</td>
</tr>
<tr>
<td>60.0 INR@bank$_4$</td>
<td>0.93 EUR@bank$_3$</td>
</tr>
</tbody>
</table>

Decompose payment into a series of trades
- Effectively exchanging deposits at one bank for ones at another

Combine them into atomic transaction

Use well-known two-phase commit protocol across 4 banks
- Send transaction to every institution concerned
- Commit only if all participants unanimously vote to do so
- E.g., bank$_2$ can abort transaction if its offer no longer valid
Decompose payment into a series of trades
  - Effectively exchanging deposits at one bank for ones at another

Combine them into atomic transaction

Use well-known two-phase commit protocol across 4 banks
  - Send transaction to every institution concerned
  - Commit only if all participants unanimously vote to do so
  - E.g., bank$_2$ can abort transaction if its offer no longer valid
Strawman: Two-phase commit

- Decompose payment into a series of trades
  - Effectively exchanging deposits at one bank for ones at another

- Combine them into atomic transaction

- Use well-known two-phase commit protocol across 4 banks
  - Send transaction to every institution concerned
  - Commit only if all participants unanimously vote to do so
  - E.g., bank\textsubscript{2} can abort transaction if its offer no longer valid
Strawman: Two-phase commit

Decompose payment into a series of trades
- Effectively exchanging deposits at one bank for ones at another

Combine them into atomic transaction

Use well-known two-phase commit protocol across 4 banks
- Send transaction to every institution concerned
- Commit only if all participants unanimously vote to do so
- E.g., bank₂ can abort transaction if its offer no longer valid
The problem: Failure

**What if bank\(_2\) disappears mid transaction?**
- Don’t know whether or when it will come back online…
- Now your original dollar is tied up pending transaction resolution

**What if bank\(_2\) lies and changes vote?**
- Bank\(_2\) might convince bank\(_1\) of commit and bank\(_3\) of abort…
- Now bank\(_2\) receives USD at bank\(_1\) without paying EUR at bank\(_3\)!
- Note bank\(_4\) can collude with bank\(_2\), or create chain of “Sybil” banks
- Any majority-based scheme is useless with unknown market makers

We need secure transactions across unknown, untrusted parties
The problem: Failure

What if bank\textsubscript{2} disappears mid transaction?
- Don’t know whether or when it will come back online…
- Now your original dollar is tied up pending transaction resolution

What if bank\textsubscript{2} lies and changes vote?
- Bank\textsubscript{2} might convince bank\textsubscript{1} of commit and bank\textsubscript{3} of abort…
- Now bank\textsubscript{2} receives USD at bank\textsubscript{1} without paying EUR at bank\textsubscript{3}!
- Note bank\textsubscript{4} can collude with bank\textsubscript{2}, or create chain of “Sybil” banks
- Any majority-based scheme is useless with unknown market makers

We need secure transactions across unknown, untrusted parties
The problem: Failure

What if bank$_2$ disappears mid transaction?
- Don’t know whether or when it will come back online…
- Now your original dollar is tied up pending transaction resolution

What if bank$_2$ lies and changes vote?
- Bank$_2$ might convince bank$_1$ of commit and bank$_3$ of abort…
- Now bank$_2$ receives USD at bank$_1$ without paying EUR at bank$_3$!
- Note bank$_4$ can collude with bank$_2$, or create chain of “Sybil” banks
- Any majority-based scheme is useless with unknown market makers

We need secure transactions across unknown, untrusted parties
The problem: Failure

What if bank$_2$ disappears mid transaction?
- Don’t know whether or when it will come back online…
- Now your original dollar is tied up pending transaction resolution

What if bank$_2$ lies and changes vote?
- Bank$_2$ might convince bank$_1$ of commit and bank$_3$ of abort…
- Now bank$_2$ receives USD at bank$_1$ without paying EUR at bank$_3$!
- Note bank$_4$ can collude with bank$_2$, or create chain of “Sybil” banks
- Any majority-based scheme is useless with unknown market makers

We need secure transactions across unknown, untrusted parties
Cyclic quorum slice example

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

$$Q(v_i) = \{ \{ v_i, v_{(i \text{ mod } 6)+1} \} \}$$