CS251 Fall 2023

(cs251.stanford.edu)

Fundamentals of Consensus

Ertem Nusret Tas

Byzantine Generals Problem

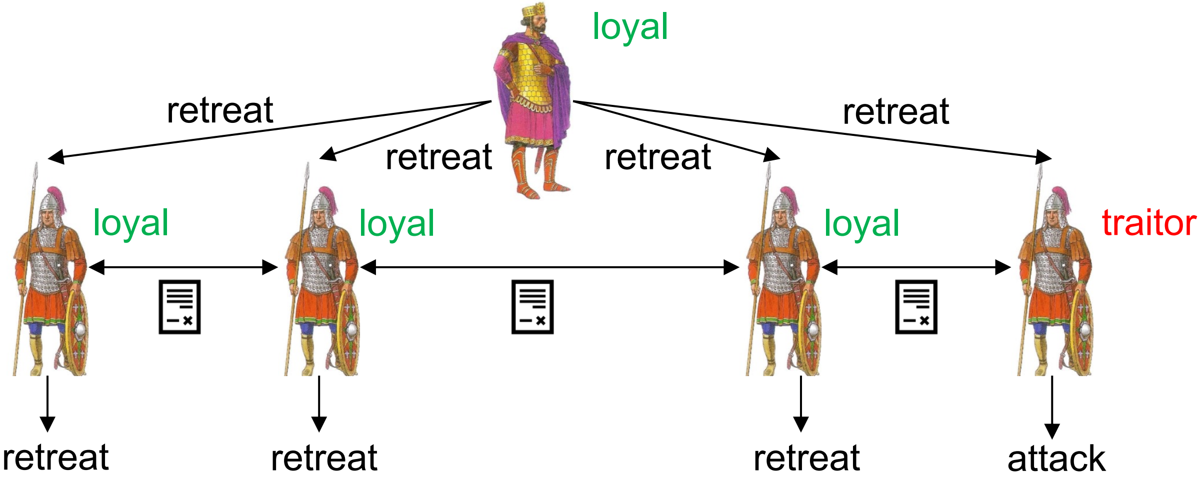
- Encapsulates the problem of reaching consensus.
- Introduced by Lamport et al. in 1982.
- Problem statement:
  - There are n generals (where n is fixed), one of which is the *commander*.
  - Some generals are *loyal*, and some of them can be *traitors* (including the commander).
  - The commander sends out an order that is either *attack* or *retreat* to each general.
  - If the commander is loyal, it sends the *same* order to all generals.
  - All generals take an action after some time.

The Byzantine Generals Problem (1982)

Byzantine Generals Problem

Goal:

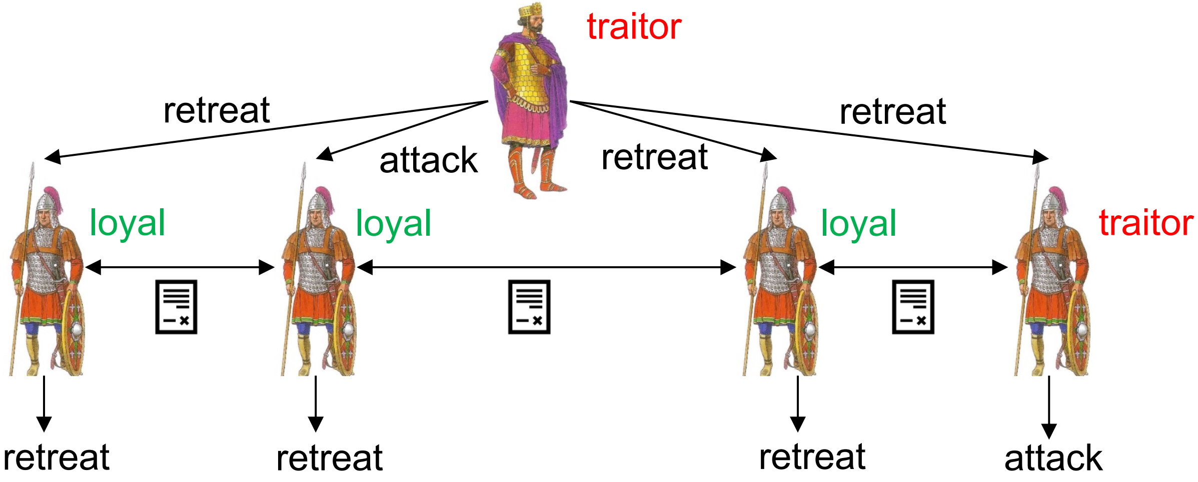
- **Agreement: No** two loyal generals take **different** actions.
- **Validity:** If the commander is loyal, then all loyal generals must take the action **suggested by the commander**.
- **Termination:** All loyal generals must eventually take some action.



Byzantine Generals Problem

Goal:

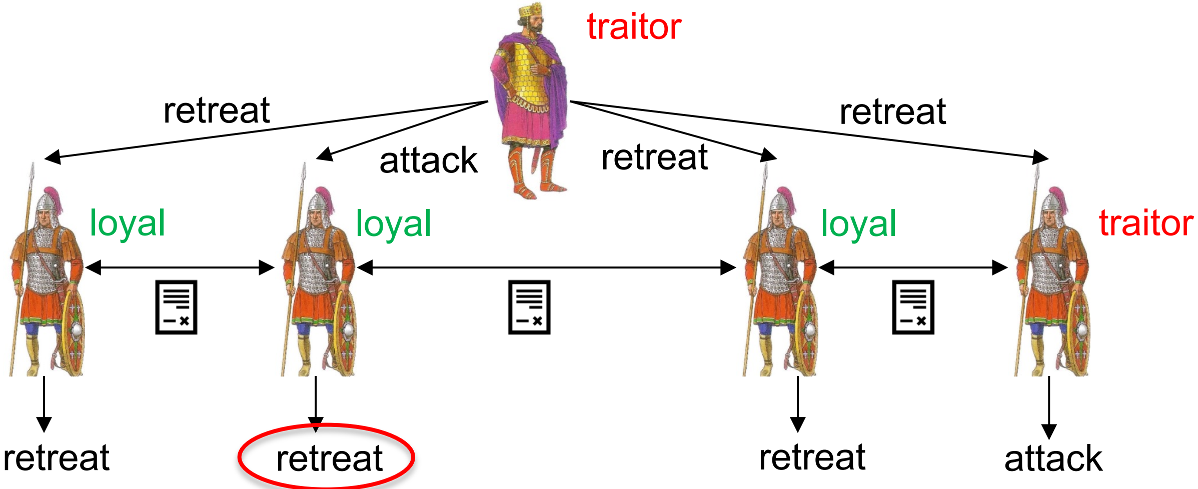
- **Agreement: No** two loyal generals take **different** actions.
- **Validity:** If the commander is loyal, then all loyal generals must take the action **suggested by the commander**.
- **Termination:** All loyal generals must eventually take some action.



Byzantine Generals Problem

Goal:

- **Agreement: No** two loyal generals take **different** actions.
- **Validity:** If the commander is loyal, then all loyal generals must take the action **suggested by the commander**.
- **Termination:** All loyal generals must eventually take some action.

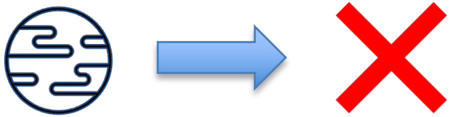


From Generals to Nodes

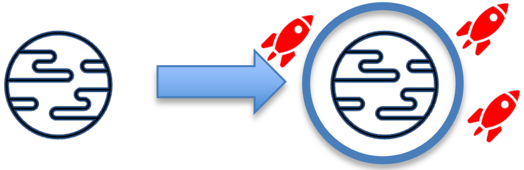
- Solution to the Byzantine Generals Problem is a *consensus protocol*.
- When modelling consensus protocols:
  - Generals → Nodes
  - Commander → Leader
  - Loyal → Honest, Traitor → Adversary
    - What can the adversarial nodes do?

Adversary

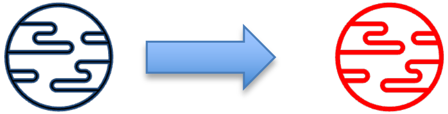
- *The* adversary can *corrupt* nodes, after which they are called **adversarial**.
  - **Crash faults if** the adversarial nodes do not send or receive any messages.



  - **Omission faults if** the adversarial nodes can selectively choose to drop or let through each messages sent or received.



  - **Byzantine faults (Byzantine adversary) if** the adversarial nodes can deviate from the protocol arbitrarily.



Adversary

We typically bound the adversary's power by assuming an upper bound (f) on the number of nodes (n) that can ever be adversarial.

- e.g., $f < n$, $f < \frac{n}{2}$, $f < \frac{n}{3}$, …

Communication

- Nodes can send messages to each other, authenticated by *signatures*.
- There is a public key infrastructure (PKI) setup.
    - Adversary cannot simulate honest nodes!
    - There are other ways to prevent such simulation (e.g., proof-of-work).

Consensus protocols typically assume that the adversary cannot forge signatures. Why?

Communication

We assume that the adversary *controls* the delivery of the messages subject to certain limits (the adversary runs the network):

- In a **synchronous network,** adversary must deliver any message sent by an honest node to its recipient(s) within Δ rounds. Here, Δ is a *known* bound.

- In an **asynchronous network**, adversary can delay any message for an arbitrary, yet finite amount of time. However, it must eventually deliver every message sent by the honest nodes.

Byzantine Generals Problem

- There are n generals (where n is fixed), one of which is the commander.
- For a public f, a subset of f generals is adversarial, and all other generals are loyal.
- The commander sends out an order that is either attack or retreat to each general.
- Network is synchronous.

Byzantine Generals Problem:

- **Agreement: No** two loyal generals take **different** actions.
- **Validity:** If the commander is loyal, then all loyal generals must take the action **suggested by the commander**.
- **Termination:** All loyal generals must eventually take some action.

Byzantine Broadcast (BB)

- There are n nodes (where n is fixed), one of which is the leader.
- For a public f, a subset of f nodes is adversarial, and all other nodes are honest
- The leader has an input value 0 or 1.
- Network is synchronous.

Byzantine Broadcast Problem:

- **Agreement: No** two honest nodes output **different** values.
- **Validity:** Leader is honest $\Rightarrow$ All honest nodes output the value **input to the leader**.
- **Termination:** All honest nodes eventually output some value.

Byzantine Broadcast (BB)

- There are n nodes (where n is fixed), one of which is the leader.
- For a public f, a subset of f nodes is adversarial, and all other nodes are honest
- The leader has an input value 0 or 1.
- Network is synchronous.

Byzantine Broadcast Problem:

- **Agreement: No** two honest nodes output **different** values.

even when the leader is adversarial!!

- **Validity:** Leader is honest $\Rightarrow$ All honest nodes output the value **input to the leader**.
- **Termination:** All honest nodes eventually output some value.

Byzantine Broadcast (BB)

- There are n nodes (where n is fixed), one of which is the leader.
- For a public f, a subset of f nodes is adversarial, and all other nodes are honest
- The leader has an input value 0 or 1.
- Network is synchronous.

Byzantine Broadcast Problem:

No double spend

- **Agreement: No** two honest nodes output **different** values.

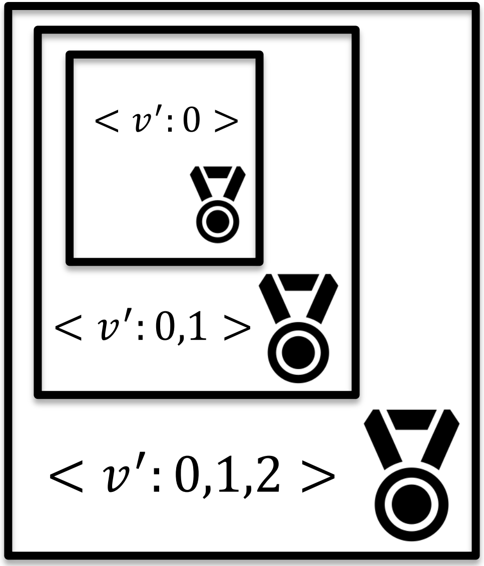
even when the leader is adversarial!!

- **Validity:** Leader is honest $\Rightarrow$ All honest nodes output the value **input to the leader**.
- **Termination:** All honest nodes eventually output some value.

No censorship

Protocol for BB: Setup

- Denote the nodes by the indices $i = 0, 1, 2, \dots, n$.
- Node 0 is the leader. Let v denote its value.
- Let V_i denote the set of values received by node i.
- Time moves in *lock-step*.



- Let $< v': i >$ denote the value v' signed by node i.
- Let $< v': i, j, \dots, l, k >$ denote *a signature chain* signed by $i, j, \dots, k$:
  - Recursive definition: $< v': i, j, \dots, l, k > = << v': i, j, \dots, l > : \; k >$

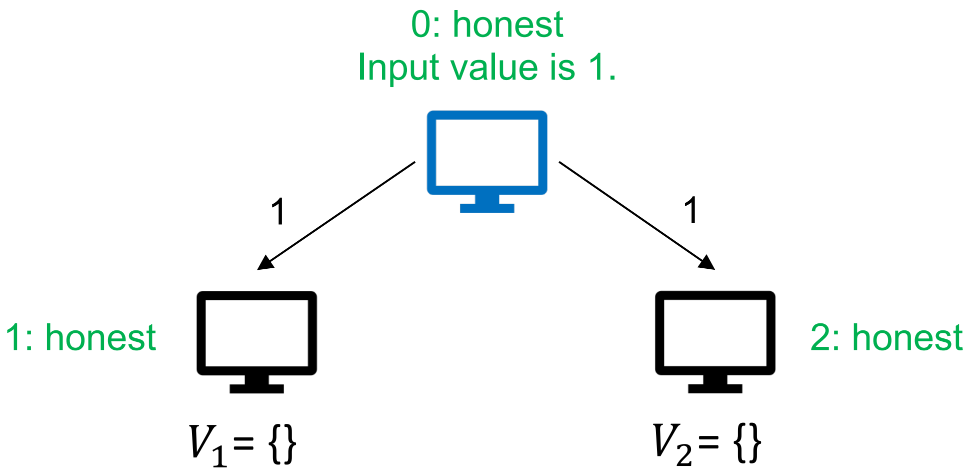
Strawman Protocol I

- Time 0: Leader broadcasts $< v: 0 >$. // v is either 0 or 1. (the broadcast value)
- Time 1:
  - Node i:
    - Upon receiving any $< v': 0 >$, add v' to V_i.
    - Decide value $\text{choice}(V_i)$.

$\text{choice}(V_i)$:

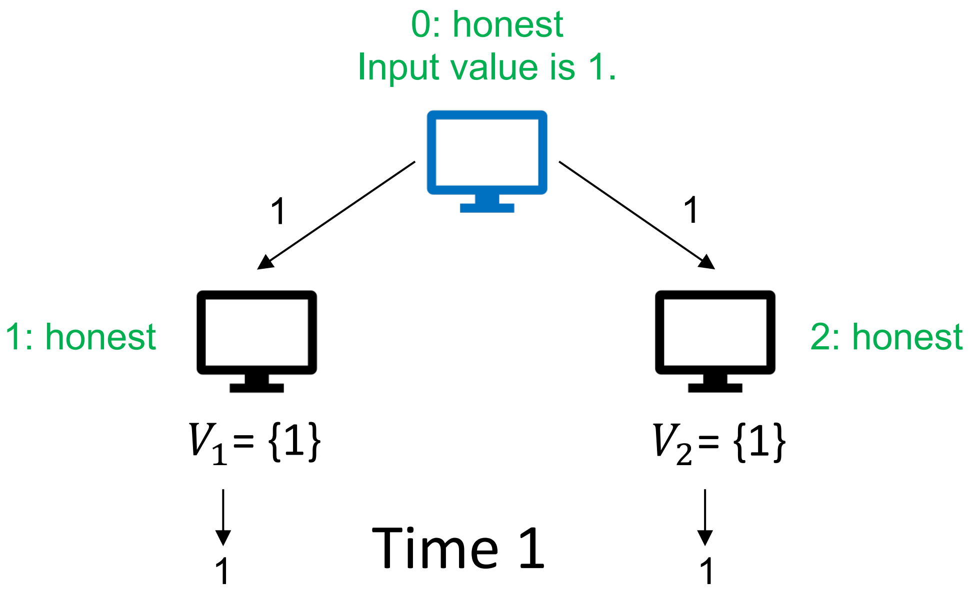
- If $V_i = \{v\}$, return v.
- Else, return 0.

Strawman Protocol I



Time 0

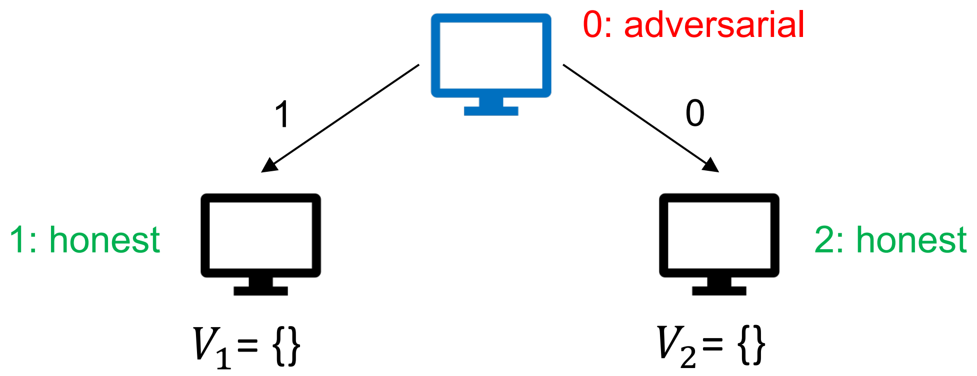
Strawman Protocol I



Validity is satisfied!

Strawman Protocol I

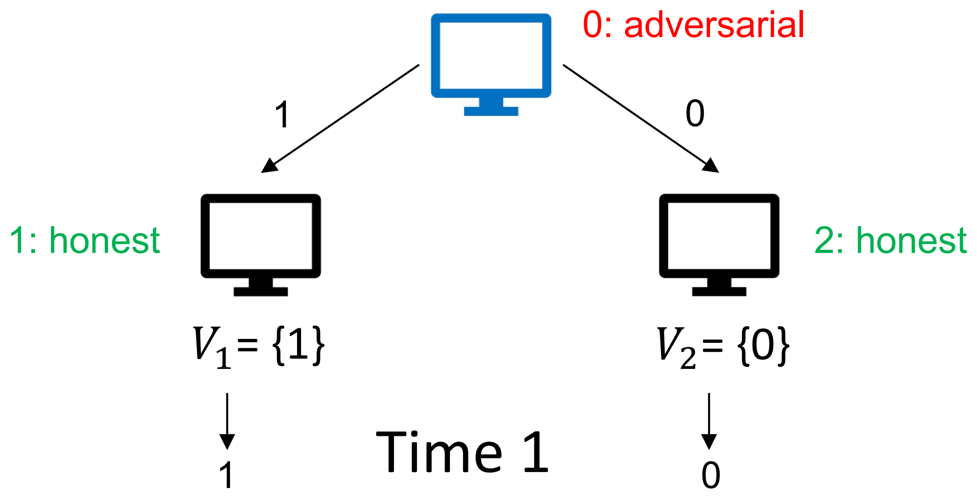
Problem: what if the leader is adversarial?



Time 0

Strawman Protocol I

Problem: what if the leader is adversarial?

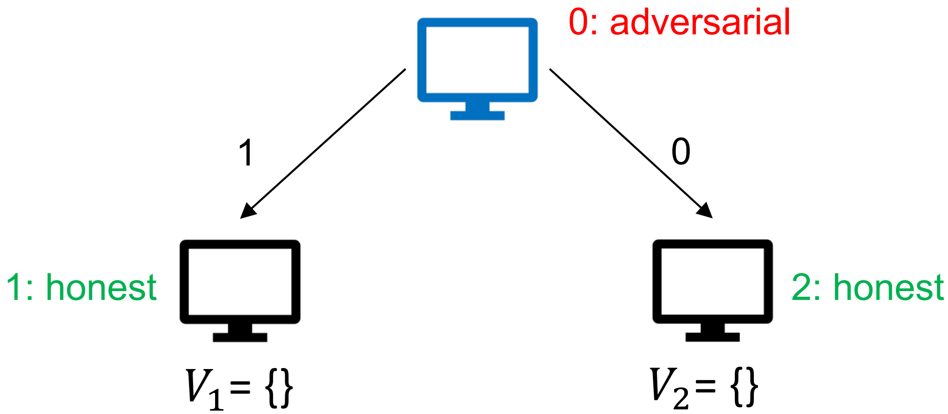


Agreement is violated!

Strawman Protocol II

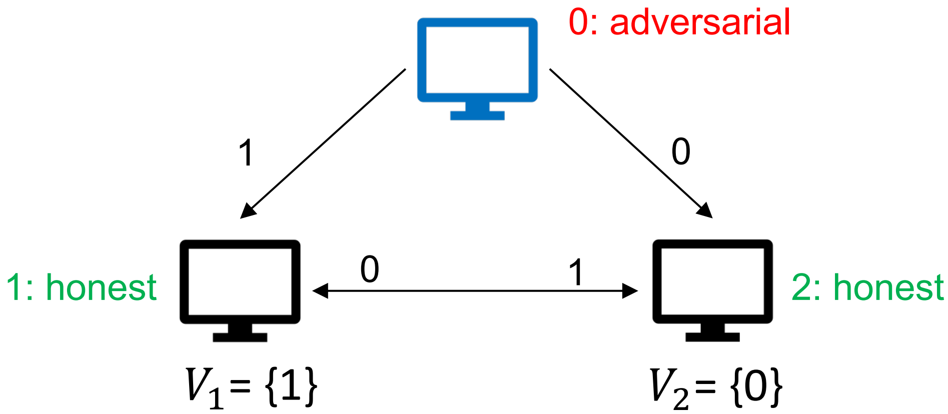
- Time 0: Leader broadcasts $< v: 0 >$. // v is either 0 or 1. (the broadcast value)
- Time 1:
  - Node i:
    - Upon receiving any $< v': 0 >$,
      add v' to V_i,
      and broadcast $< v': 0, i >$.
- Time 2:
  - Node i:
    - Upon receiving any $< v': 0, j >$, where $j \neq 0$, add v' to V_i.
    - Decide value $\mathrm{choice}(V_i)$.

Strawman Protocol II



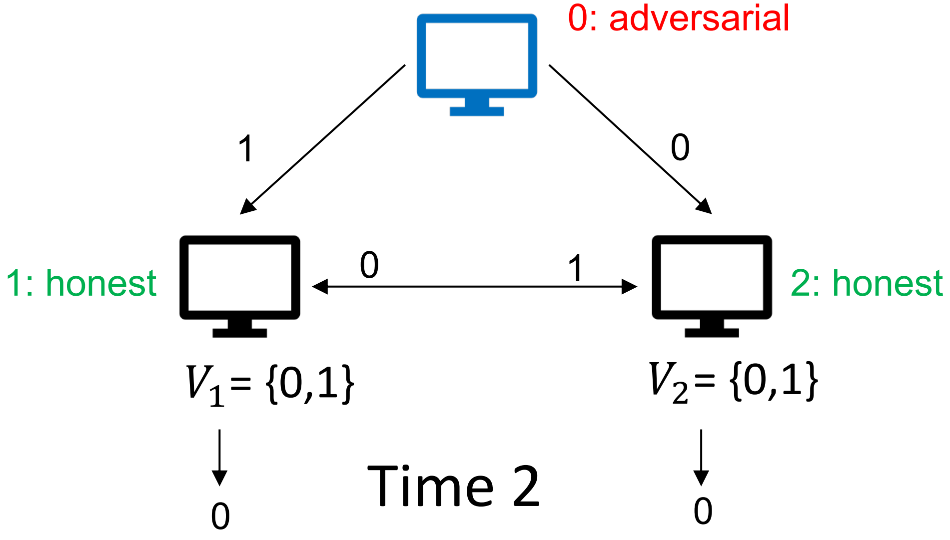
Time 0

Strawman Protocol II



Time 1

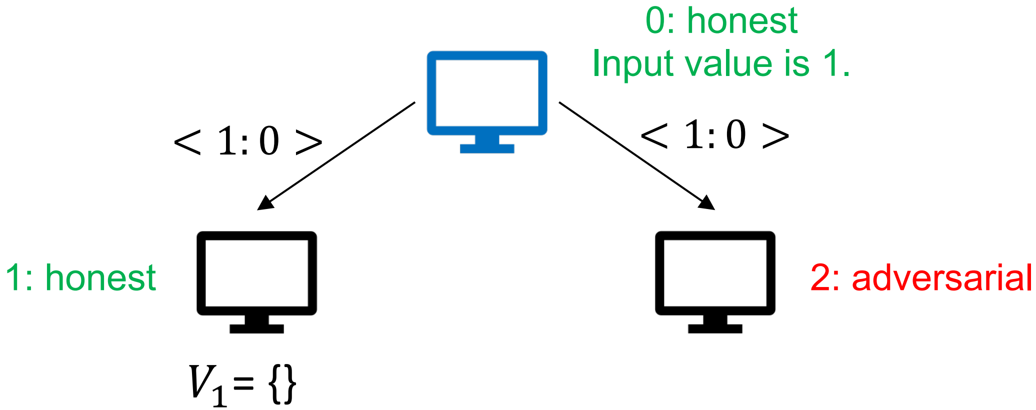
Strawman Protocol II



Agreement is satisfied!

Strawman Protocol II

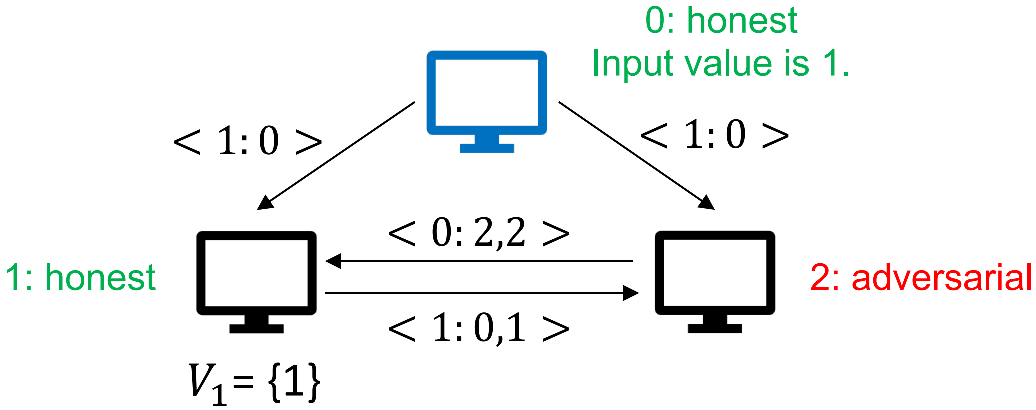
Problem: what if one of the nodes is adversarial?



Time 0

Strawman Protocol II

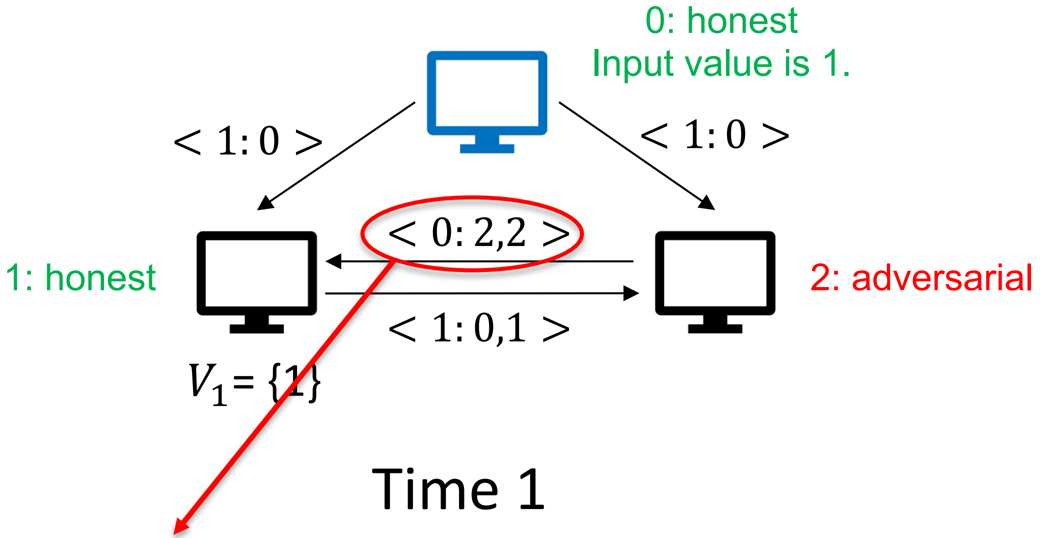
Problem: what if one of the nodes is adversarial?



Time 1

Strawman Protocol II

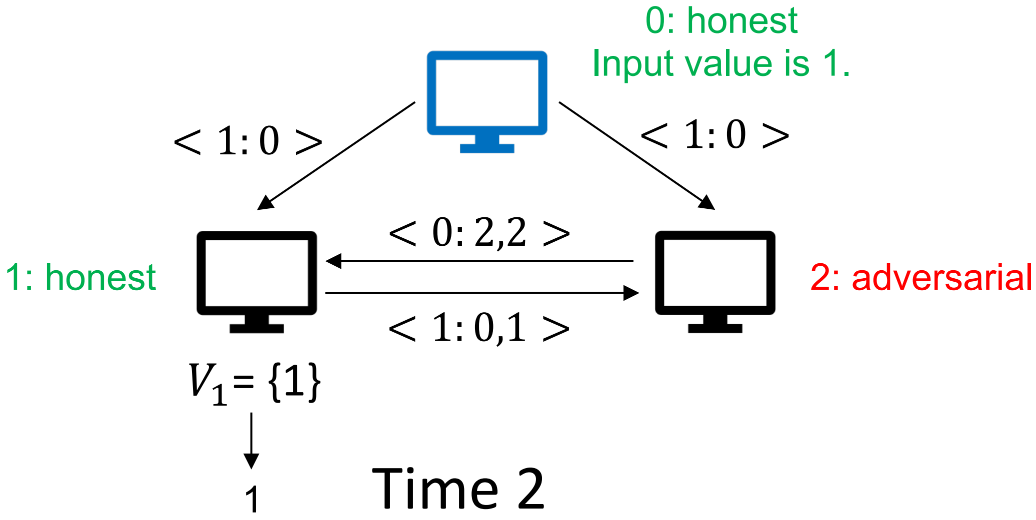
Problem: what if one of the nodes is adversarial?



Invalid since the first signature is not by the leader, i.e., node 0. Thus, 0 is not added to V_1.

Strawman Protocol II

Problem: what if one of the nodes is adversarial?



Validity is satisfied as well!

So are agreement and termination!

Dolev-Strong (1983)

- Time 0: Leader broadcasts $< v: 0 >$. // v is either 0 or 1. (the broadcast value)
- Time $t = 1, \dots, f$:
  - Node i:
    - Upon receiving any $< v': 0, i_1 \dots, i_{t-1} >$, where $i \neq i_1 \neq \cdots \neq i_{t-1}$ and $v' \notin V_i$, add v' to V_i and broadcast $< v': 0, i_1 \dots, i_{t-1}, i >$.
- Time $f + 1$:
  - Node i:
    - Upon receiving any $< v': 0, i_1 \dots, i_f >$, where $i \neq i_1 \neq \cdots \neq i_f$ and $v' \notin V_i$, add v' to V_i.
    - Decide value choice(V_i).

Authenticated Algorithms for Byzantine Agreement (1983)

Security of Dolev-Strong (1983)

Theorem (Dolev-Strong, 1983): For any $f < n$, Dolev-Strong (1983) with n nodes and $f + 1$ rounds satisfies agreement, validity and termination in a synchronous network.

(try to prove yourself … the proof is in the slides at the end of the deck)

Converse Theorem: Any (deterministic) protocol that satisfies agreement, validity and termination for n nodes in a synchronous network with resilience up to f crash (as well as Byzantine) faults must have an execution with at least $f + 1$ rounds.

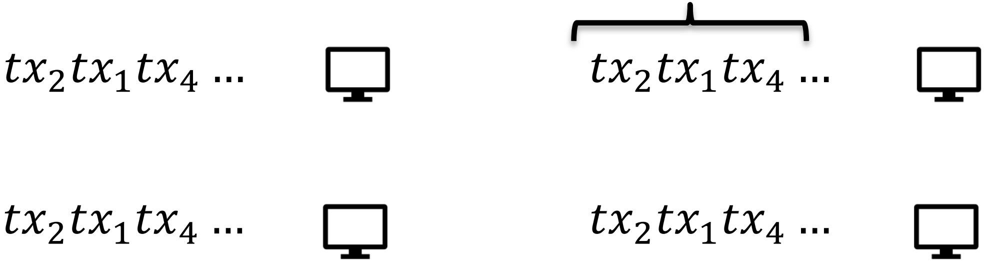
Authenticated Algorithms for Byzantine Agreement (1983)
Distributed Algorithms (1996)
A Simple Bivalency Proof that t-Resilient Consensus Requires t + 1 Rounds (1998)

State Machine Replication (SMR)

A Centralized Bank

Blockchain (State Machine Replication)

Log (Ledger): an ever-growing, linearly-ordered *sequence* of transactions.

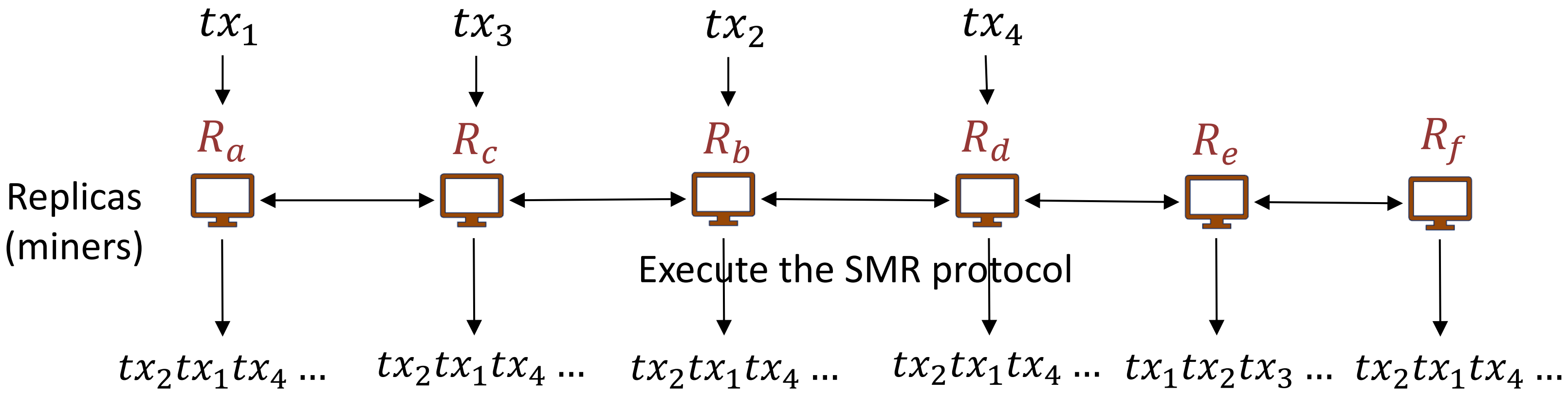


State Machine Replication (SMR)

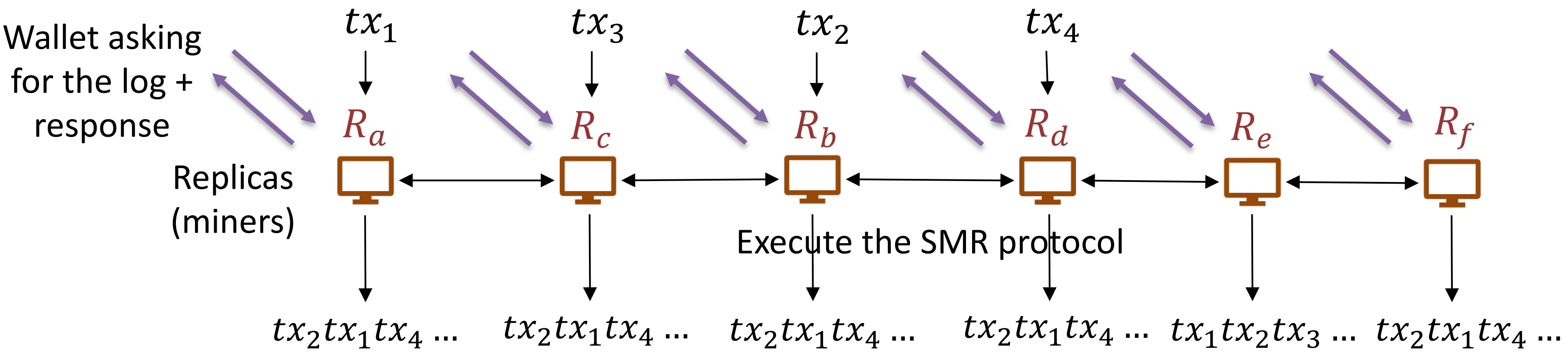
Two parties of SMR:

- *Replicas* receive transactions, execute the SMR protocol and determine the log.
- *Clients* are the learners: They communicate with the replicas to learn the log.

Goal of SMR is to ensure that the clients learn the *same* log.



State Machine Replication (SMR)



C_1

$LOG_t^1 = tx_2 tx_1 tx_4 \ldots$

Wallets are an example of a client.

C_2

$LOG_t^2 = tx_2 tx_1 tx_4 \ldots$

Clients (Wallets)

Wallets ask the replicas what the correct log is.

Clients (Wallets)

C_3

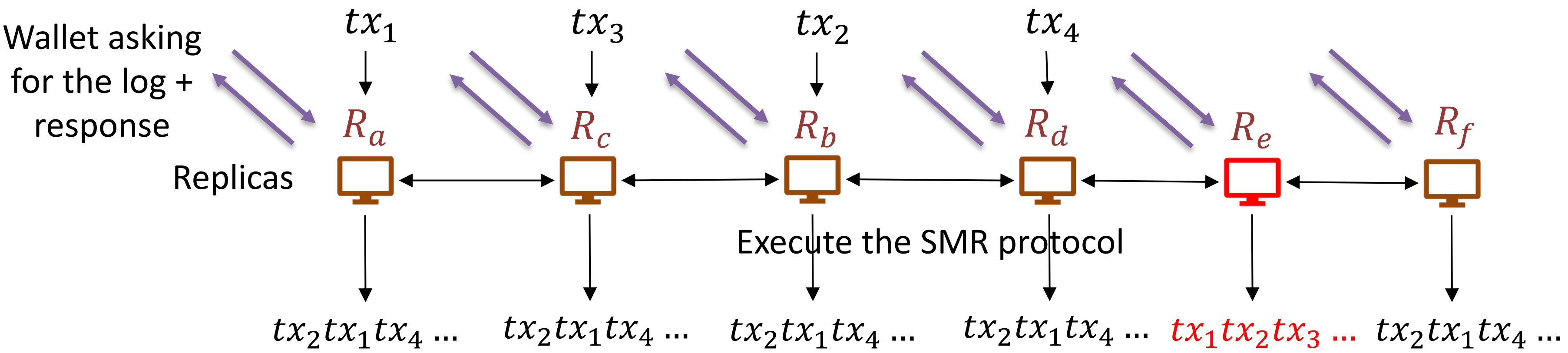
$LOG_t^3 = tx_2 tx_1 tx_4 \ldots$

Wallets do not execute the SMR protocol and do not talk to each other.

C_4

$LOG_t^4 = tx_2 tx_1 tx_4 \ldots$

State Machine Replication (SMR)



C_1

$LOG_t^1 = tx_2 tx_1 tx_4 \ldots$

C_3

$LOG_t^3 = tx_2 tx_1 tx_4 \ldots$

Clients (Wallets)

How does a wallet learn the correct log from the replicas?

- It asks the replicas what the correct log is.
- Wallet then accepts the answer given by majority of the replicas as its log.

Wallet learns the correct log if over half of the replicas are honest!

C_2

$LOG_t^2 = tx_2 tx_1 tx_4 \ldots$

Clients (Wallets)

C_4

$LOG_t^4 = tx_2 tx_1 tx_4 \ldots$

Security for SMR: Definitions

Concatenation ($\boldsymbol{A||B}$):

- Suppose we have sequences $A = tx_1tx_2$ and $\mathrm{B} = tx_3tx_4$. What is $A||B$?

$$A||B = tx_1tx_2tx_3tx_4$$

Prefix relation ($\boldsymbol{A \preccurlyeq B}$): Sequence A is said to be a prefix of sequence B, if there exists a sequence C (that is potentially empty) such that $B = A||C$.

Suppose we have $A = tx_1tx_2tx_3tx_4$, $B = tx_1tx_2tx_3$ and $D = tx_1tx_2tx_4$.

- Is B a prefix of A?
  - Yes
- Is D a prefix of A?
  - No

Security for SMR: Definitions

Two sequences A and B are consistent if either $A \preccurlyeq B$ is true or $B \preccurlyeq A$ is true or both statements are true.

Are these two logs consistent: $LOG^{Alice} = tx_1tx_2tx_3tx_4$, $LOG^{Bob} = tx_1tx_2tx_3$?

- Yes!

What about $LOG^{Alice} = tx_1tx_2tx_3$, $LOG^{Bob} = tx_1tx_2tx_3tx_4$?

- Yes!

What about $LOG^{Alice} = tx_1tx_2$, $LOG^{Bob} = tx_1tx_3$?

- No!

Security for SMR

Let LOG_t^i denote the log outputted by a client i at time t.

Then, a **secure** SMR protocol satisfies the following guarantees:

Safety (Consistency): Similar to agreement!

- For any two clients i and j, and times t and s: either $LOG_t^i \preccurlyeq LOG_s^j$ is true or $LOG_s^j \preccurlyeq LOG_t^i$ is true or both (Logs are consistent).

Liveness: Similar to validity and termination!

- If a transaction tx is input to an honest replica at some time t, then for all clients i, and times $s \geq t + T_{conf}$: $tx \in LOG_s^i$.

Security for SMR

Let LOG_t^i denote the log outputted by a client i at time t.

Then, a **secure** SMR protocol satisfies the following guarantees:

Safety (Consistency): Similar to agreement!

- For any two clients i and j, and times t and s: either $LOG_t^i \preccurlyeq LOG_s^j$ is true or $LOG_s^j \preccurlyeq LOG_t^i$ is true or both (Logs are consistent).

No double spend

Liveness: Similar to validity and termination!

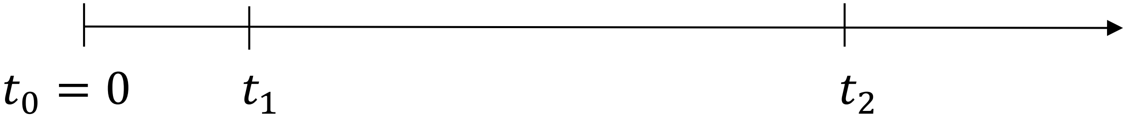
- If a transaction tx is input to an honest replica at some time t, then for all clients i, and times $s \geq t + T_{conf}$: $tx \in LOG_s^i$.

No censorship

Why is safety important?

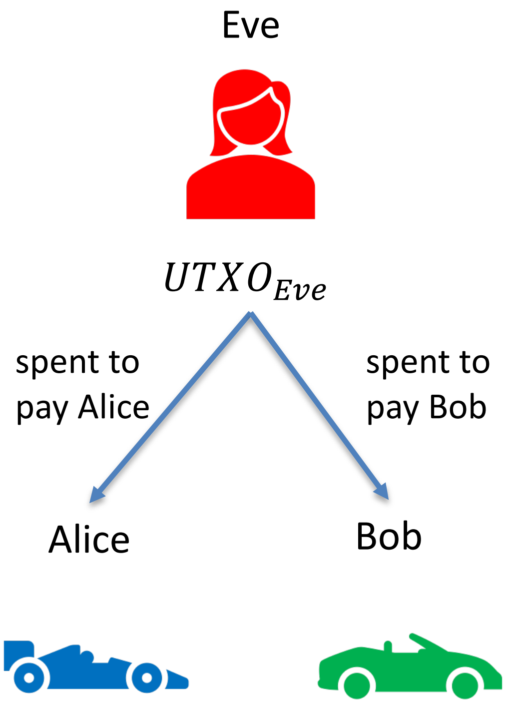
Suppose Eve has a UTXO.

- tx_1: transaction spending Eve's UTXO to pay to car vendor Alice.
- tx_2: transaction spending Eve's UTXO to pay to car vendor Bob.



- Alice's ledger at time t_1 contains tx_1: $LOG_{t_1}^{Alice} =< tx_1 >$
- Alice thinks it received Eve's payment and sends over the car.

- Bob's ledger at time t_2 contains tx_2: $LOG_{t_2}^{Bob} =< tx_2 >$
- Bob thinks it received Eve's payment and sends over the car.

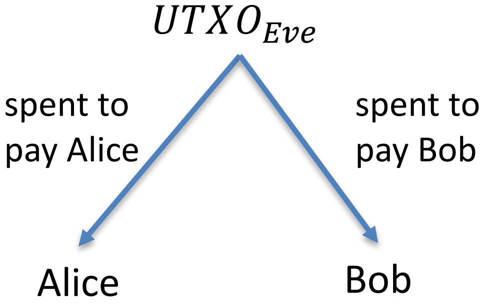


Why is safety important?

Suppose Eve has a UTXO.

- tx_1: transaction spending Eve's UTXO to pay to car vendor Alice.
- tx_2: transaction spending Eve's UTXO to pay to car vendor Bob.

Eve



$t_0 = 0$ t_1 safety violation t_2

- Alice's ledger at time t_1 contains tx_1: $LOG_{t_1}^{Alice} =< tx_1 >$
- Alice thinks it received Eve's payment and sends over the car.

- Bob's ledger at time t_2 contains tx_2: $LOG_{t_2}^{Bob} =< tx_2 >$
- Bob thinks it received Eve's payment and sends over the car.

When safety is violated, Eve can double-spend!

SMR vs. Byzantine Broadcast

- **Single shot vs. Multi-shot**
  - **Broadcast** is single shot consensus. Each node outputs a single value.
  - **State Machine Replication** is multi-shot. Each client *continuously* outputs a log, which is a sequence of transactions (values).
- **Who are the learners?**
  - In **Broadcast**, the nodes executing the protocol are the same as the nodes that output decision values.
  - In **State Machine Replication**, protocol is executed by the replicas, whereas the goal is for the clients to learn the log.
    - Replicas must ensure that the clients learn the same log.

Building an SMR protocol

Next lecture …

END OF LECTURE

Next lecture: Consensus in the Internet Setting

Security Proof for Dolev-Strong (1983)

Proof: We prove that Dolev-Strong satisfies termination, validity and agreement.

Termination: Protocol terminates in $n + 1$ time.

Validity: An honest leader signs only one value, namely its value v.

It is received by all honest nodes at time 1 and the only signature chain that can exist are those with the value v.

Security Proof for Dolev-Strong (1983)

Agreement: Suppose an honest node i added some value v' to V_i at some time $t \leq n$. Then, node i must have received a length t signature chain on v', i.e., $< v': 0, i_1 \dots, i_{t-1} >$, at time t. Now,

- If $t \leq n - 1$, node i will broadcast v' with a length $t + 1$ signature chain.
- If $t = n$, there must be a signature by an honest node among the $n - 1$ nodes $i_1 \dots, i_{n-1}$, (e.g., i_j) that broadcast v' with length $j \leq n - 1$ signature chain.

In either case, all honest nodes add v' to V_i latest at time n, i.e., before termination.

Finally, any value added by an honest node by termination is added by all other honest nodes by termination, i.e., $V_i = V_j$ for all honest nodes i, j.