Blockchain: Smart Contracts

Lecture 5

Fundamentals of Consensus

Byzantine Generals Problem

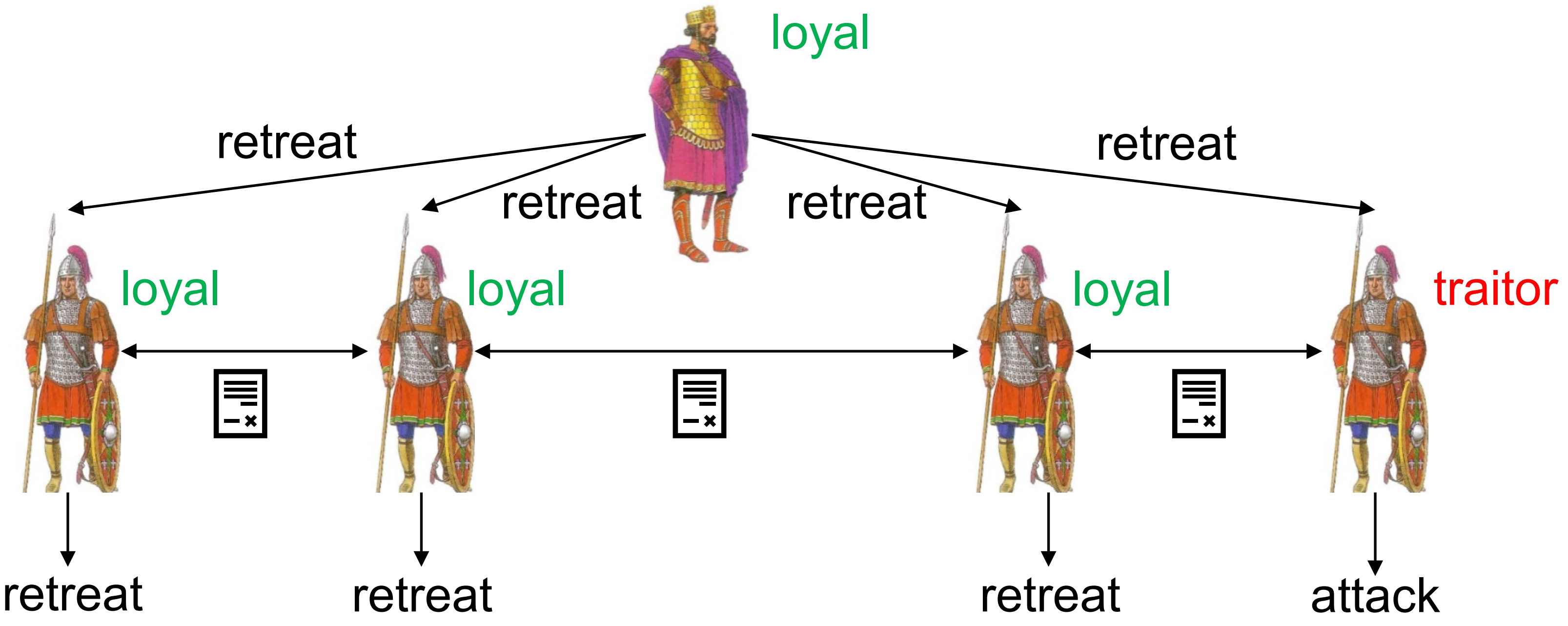
- Encapsulates the problem of reaching consensus.
- Introduced by Lamport et al. in 1982.
- Problem statement:
  - There are n generals (where n is fixed), one of which is the *commander*.
  - Some generals are *loyal*, and some of them can be *traitors* (including the commander).
  - The commander sends 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:

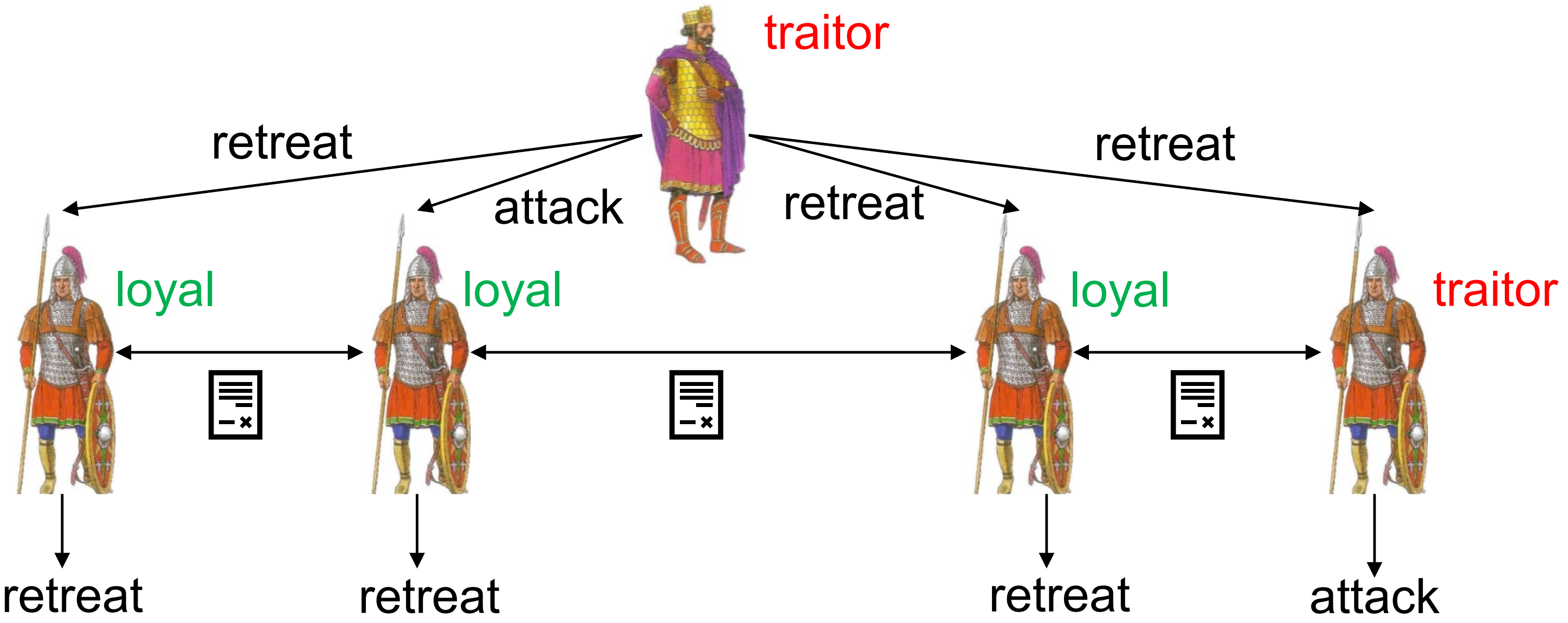
- **Agreement: No** two loyal generals take **different** actions.
- **Validity:** If the commander is loyal, then all loyal generals must take the action **suggested by the commander**.
- **Termination:** All loyal generals must eventually take some action.



Byzantine Generals Problem

Goal:

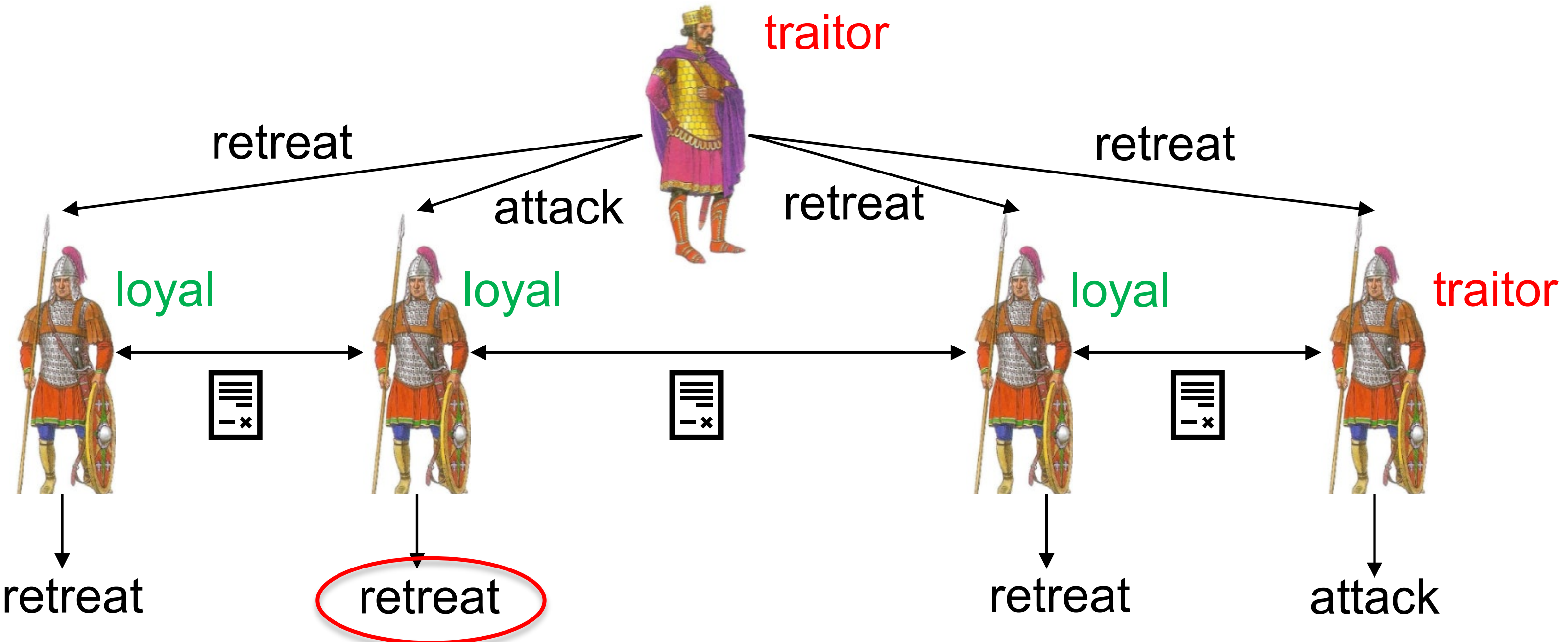
- **Agreement: No** two loyal generals take **different** actions.
- **Validity:** If the commander is loyal, then all loyal generals must take the action **suggested by the commander**.
- **Termination:** All loyal generals must eventually take some action.



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.

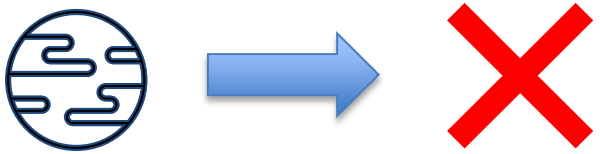


From Generals to Nodes

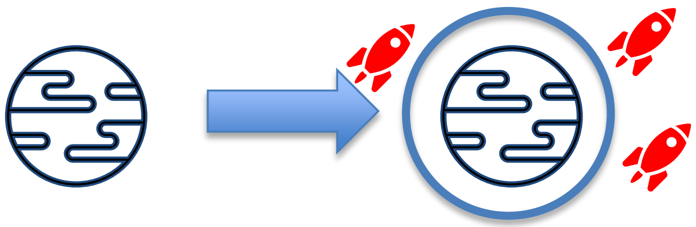
- Solution to the Byzantine Generals Problem is a *consensus protocol*.
- When modelling consensus protocols:
  - Generals → Nodes
  - Commander → Leader
  - Loyal → Honest, Traitor → Adversary
    - What can the adversarial nodes do?

Adversary

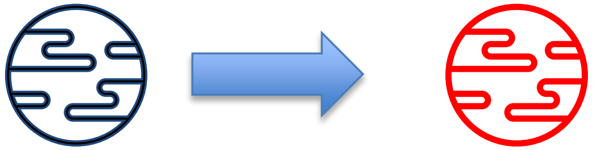
- *The* adversary can *corrupt* nodes, after which they are called **adversarial**.
  - **Crash faults if** the adversarial nodes do not send or receive any messages.



  - **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.

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.

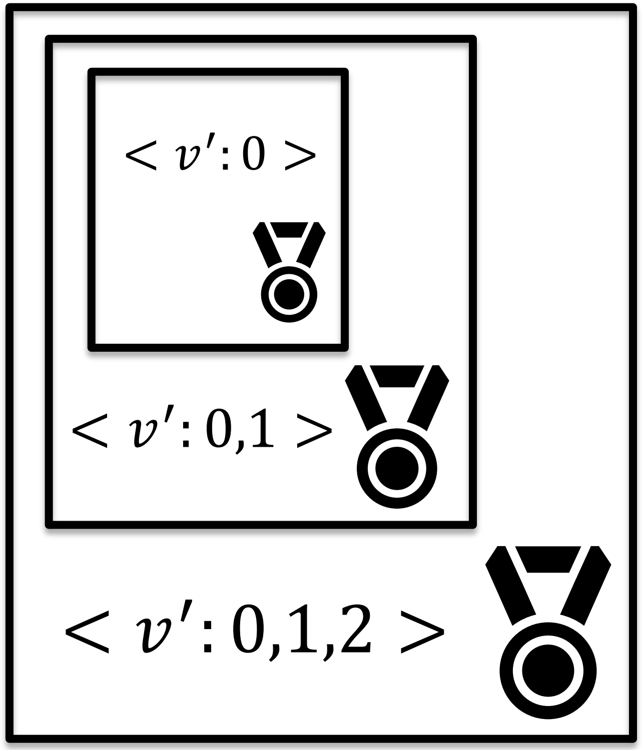
even when the leader is adversarial!!

- **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 $i = 0, 1, 2, \ldots, n$.
- 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 $< v' : i, j, \ldots, l, k >$ denote *a signature chain* signed by $i, j, \ldots, k$:
  - Recursive definition: $< v' : i, j, \ldots, l, k > = << v' : i, j, \ldots, l > : \ k >$

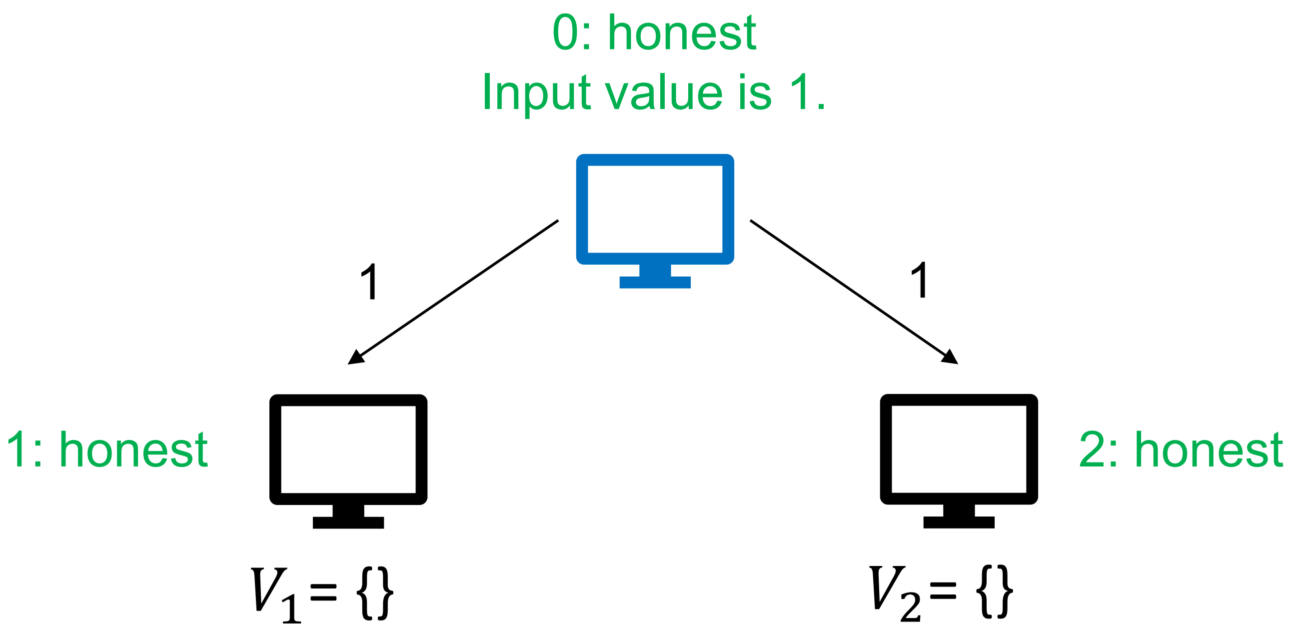
Strawman Protocol I

- 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 $\text{choice}(V_i)$.

$\text{choice}(V_i)$:

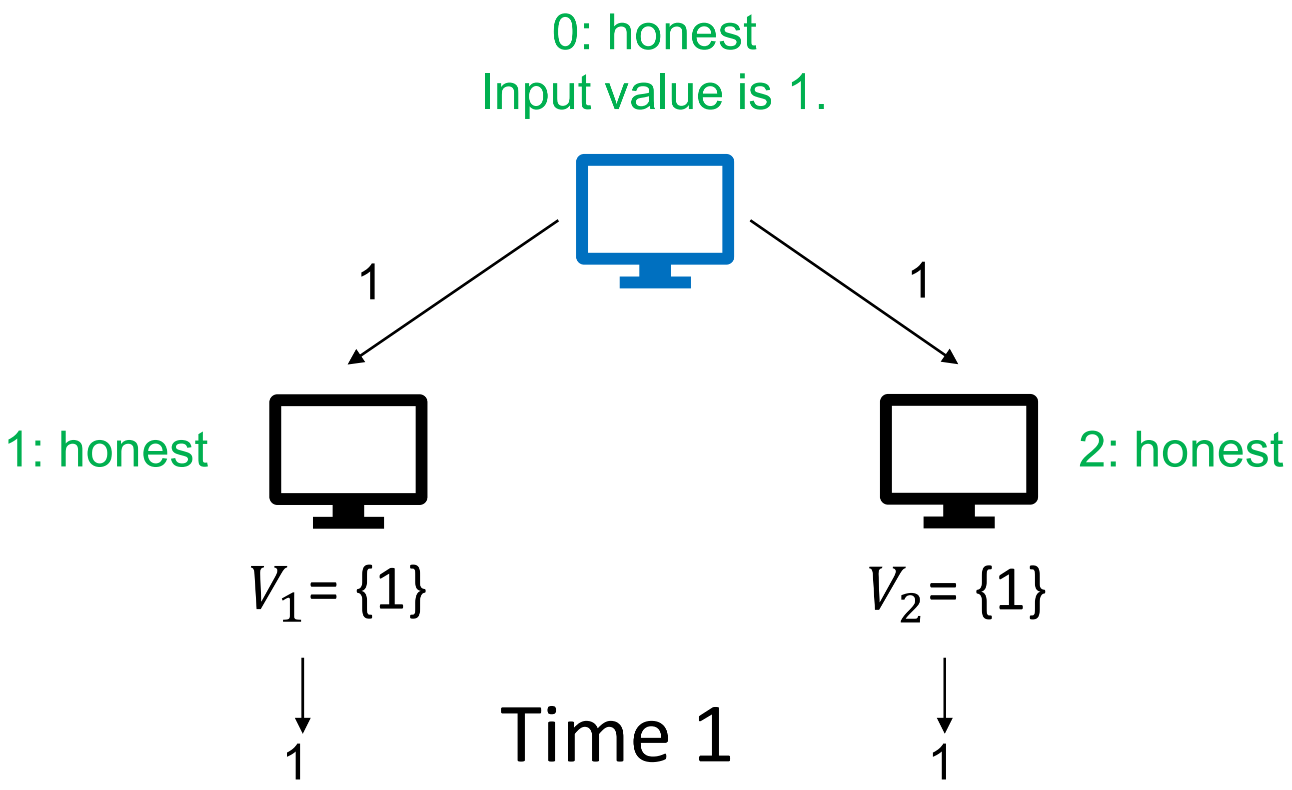
- If $V_i = \{v\}$, return v.
- Else, return 0.

Strawman Protocol I



Time 0

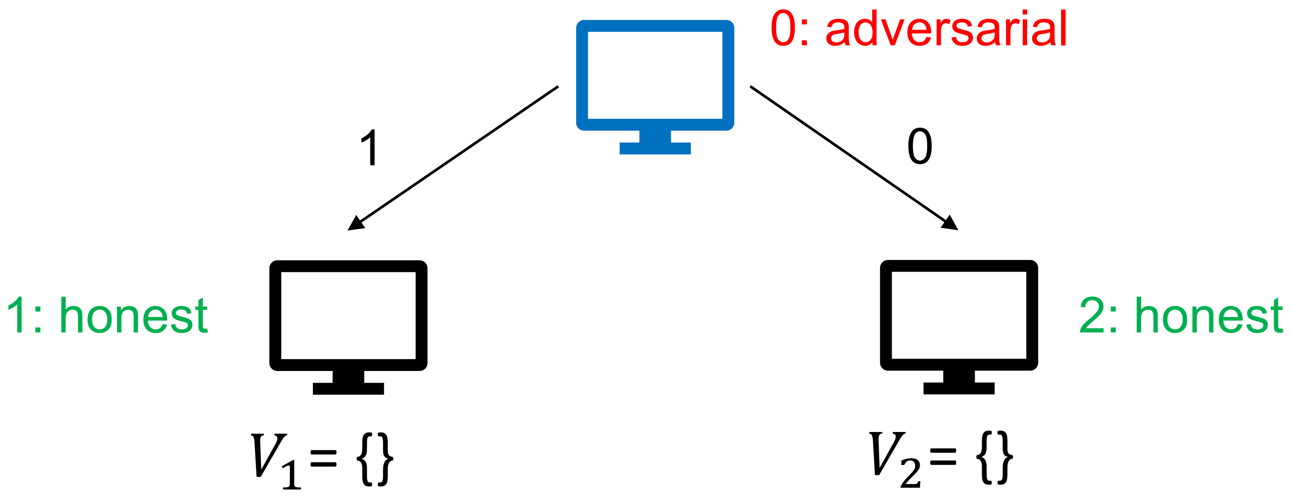
Strawman Protocol I



Validity is satisfied!

Strawman Protocol I

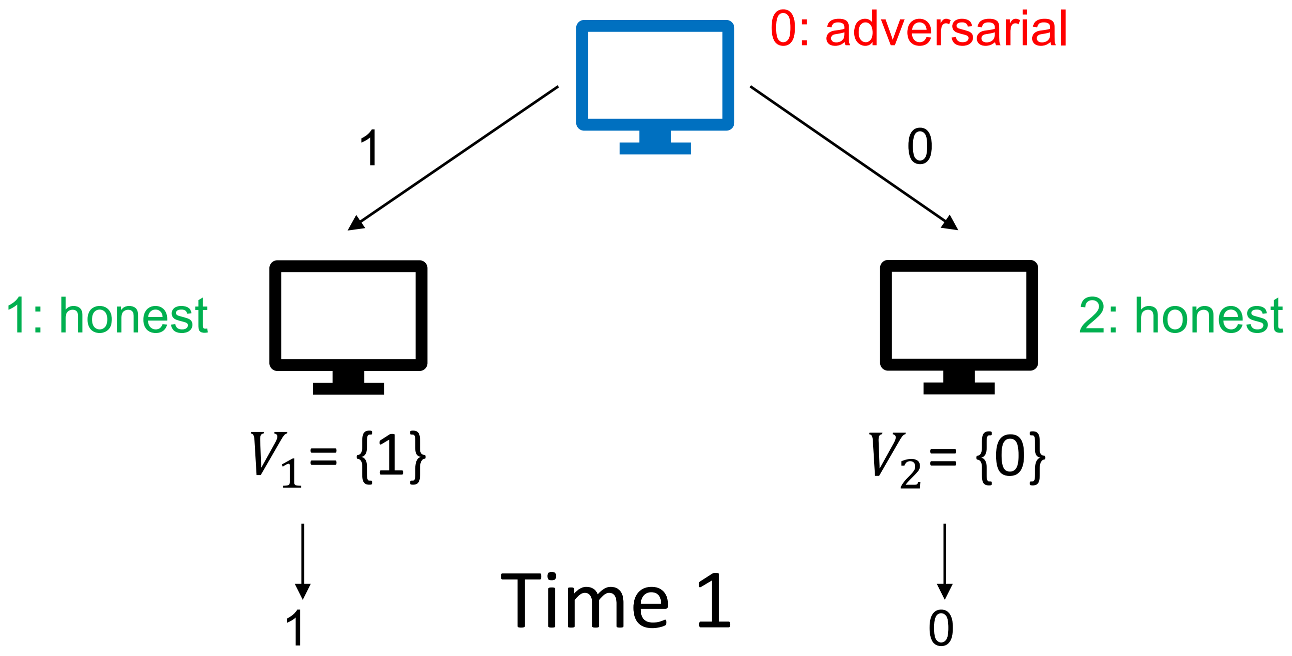
Problem: what if the leader is adversarial?



Time 0

Strawman Protocol I

Problem: what if the leader is adversarial?



Agreement is violated!

Strawman Protocol II

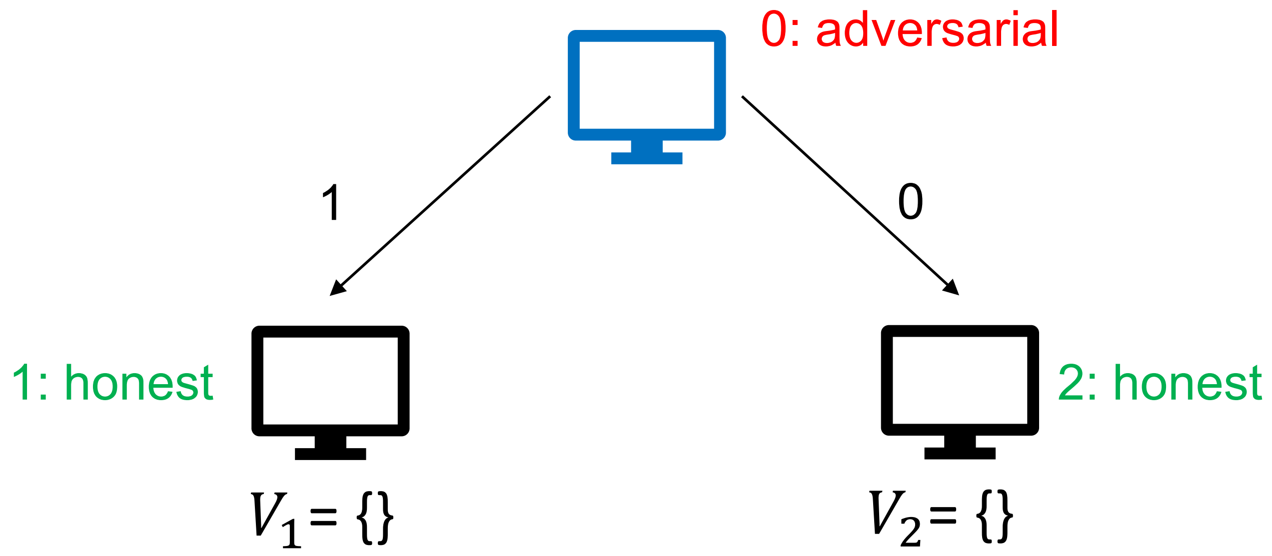
- Time 0: Leader broadcasts $< v : 0 >$.
- Time 1:
  - Node i:
    - Upon receiving any $< v' : 0 >$,

      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 $\mathrm{choice}(V_i)$.

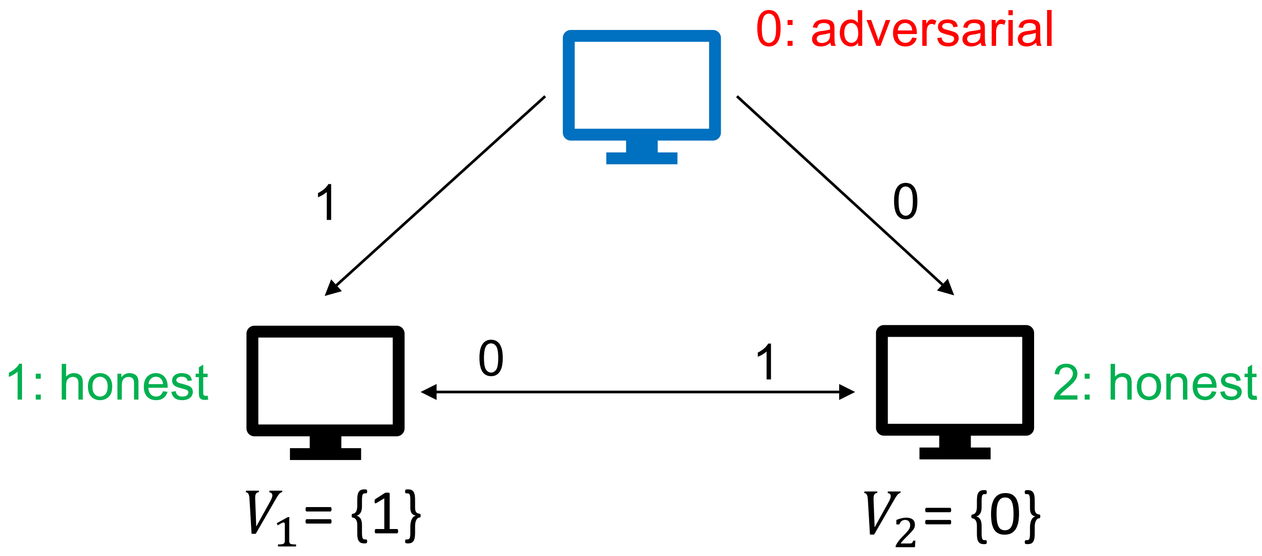
// v is either 0 or 1.
(the broadcast value)

Strawman Protocol II



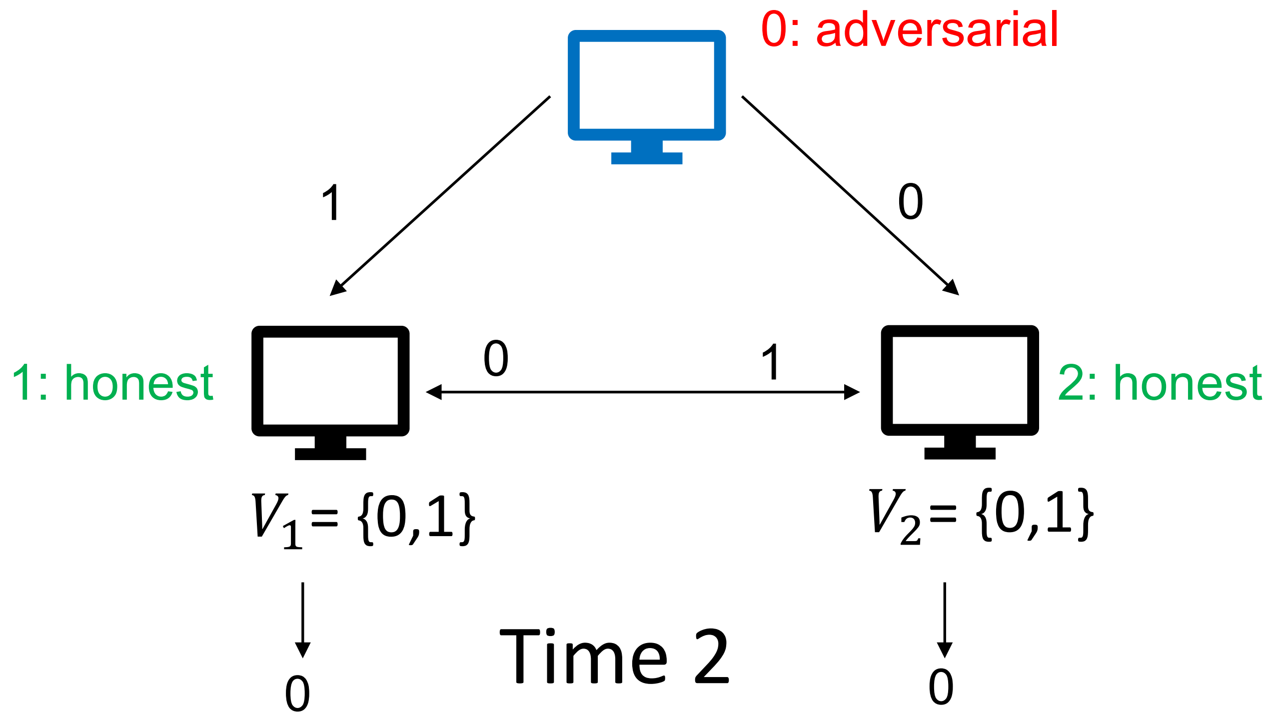
Time 0

Strawman Protocol II



Time 1

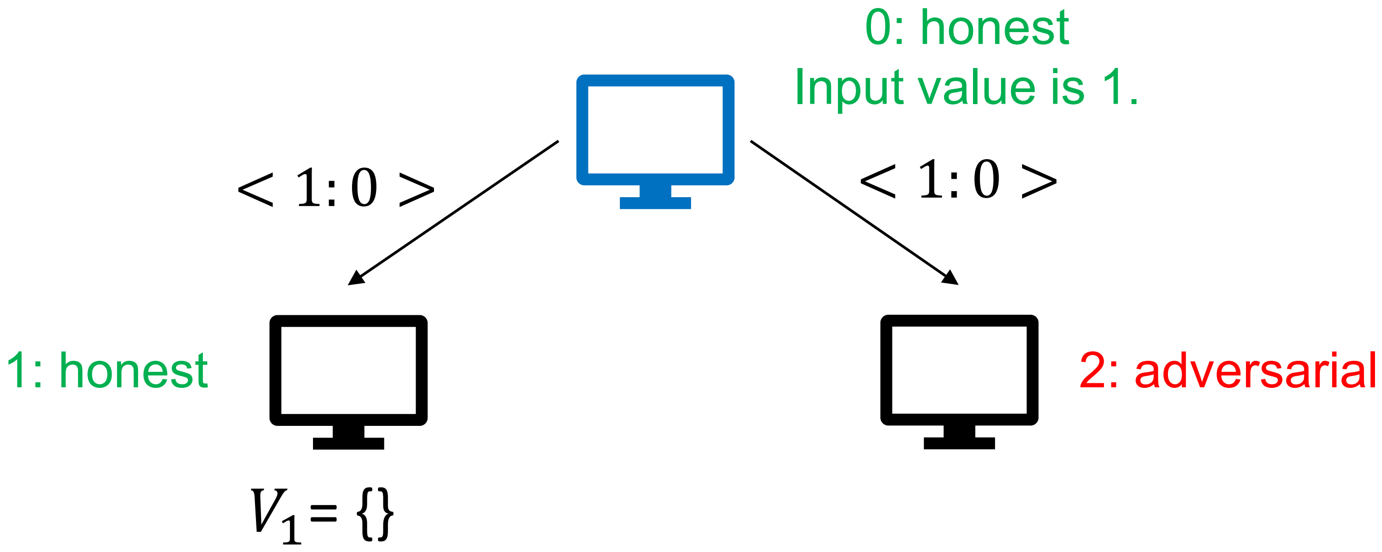
Strawman Protocol II



Agreement is satisfied!

Strawman Protocol II

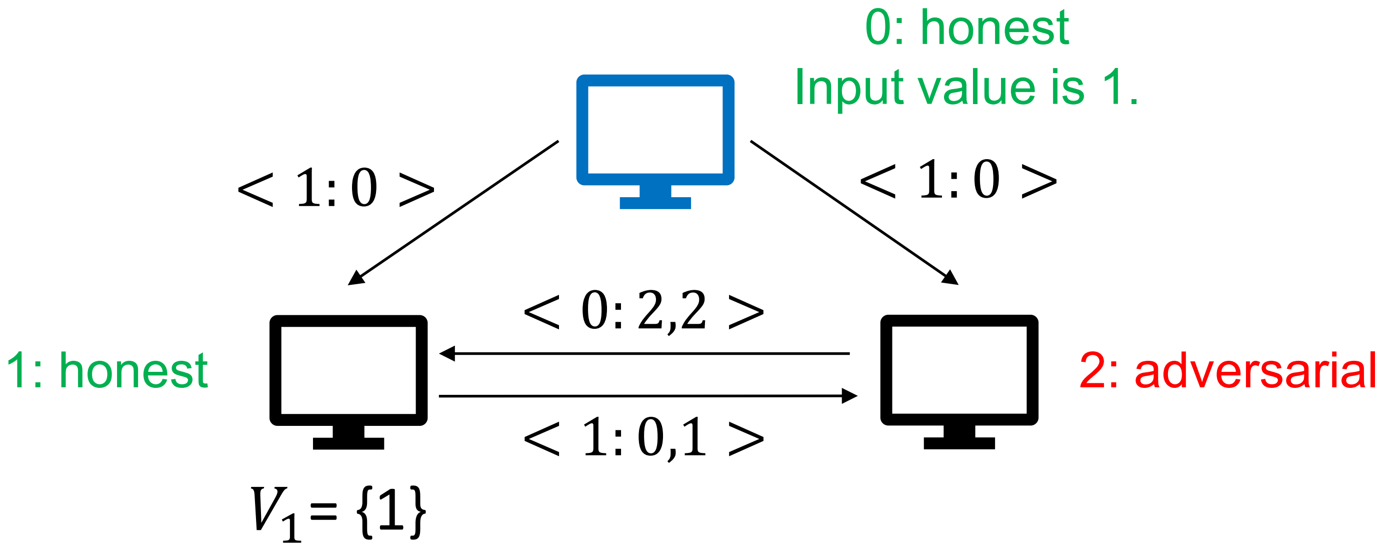
Problem: what if one of the nodes is adversarial?



Time 0

Strawman Protocol II

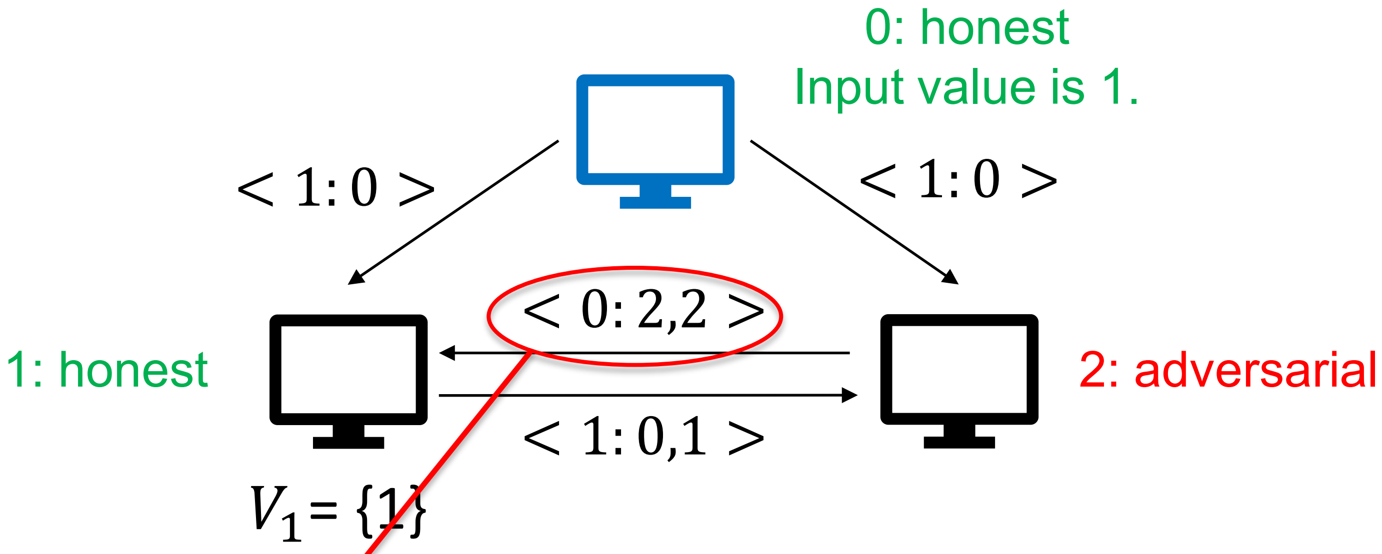
Problem: what if one of the nodes is adversarial?



Time 1

Strawman Protocol II

Problem: what if one of the nodes is adversarial?

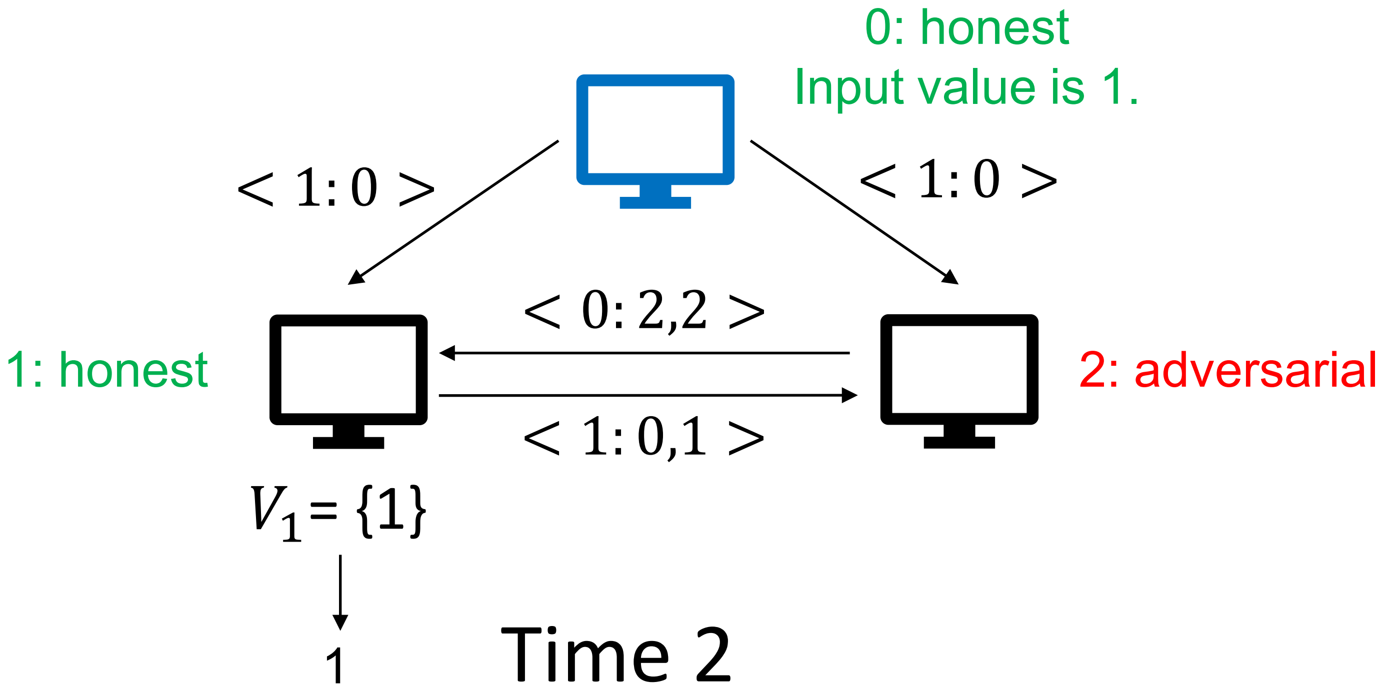


Time 1

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 $\text{choice}(V_i)$.

Authenticated Algorithms for Byzantine Agreement (1983)

Security of Dolev-Strong (1983)

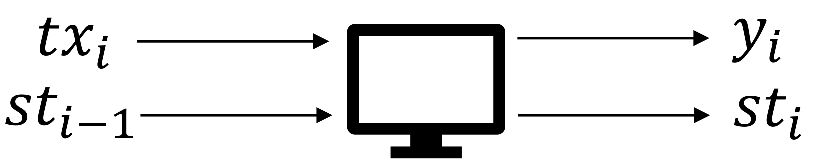
Theorem (Dolev-Strong, 1983): For any $f < n$, Dolev-Strong (1983) with n nodes and $f + 1$ rounds satisfies agreement, validity and termination in a synchronous network.

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)

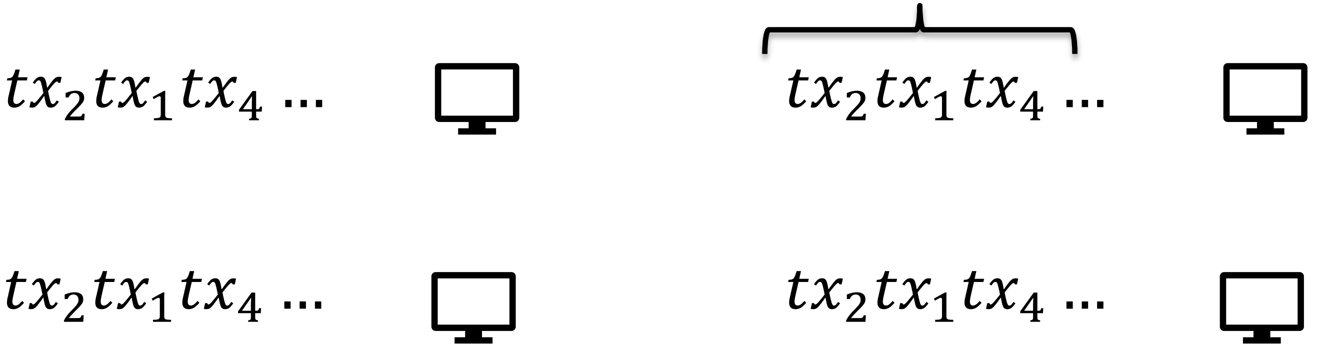
State Machine Replication (SMR)

A Centralized Bank



Blockchain (State Machine Replication)

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

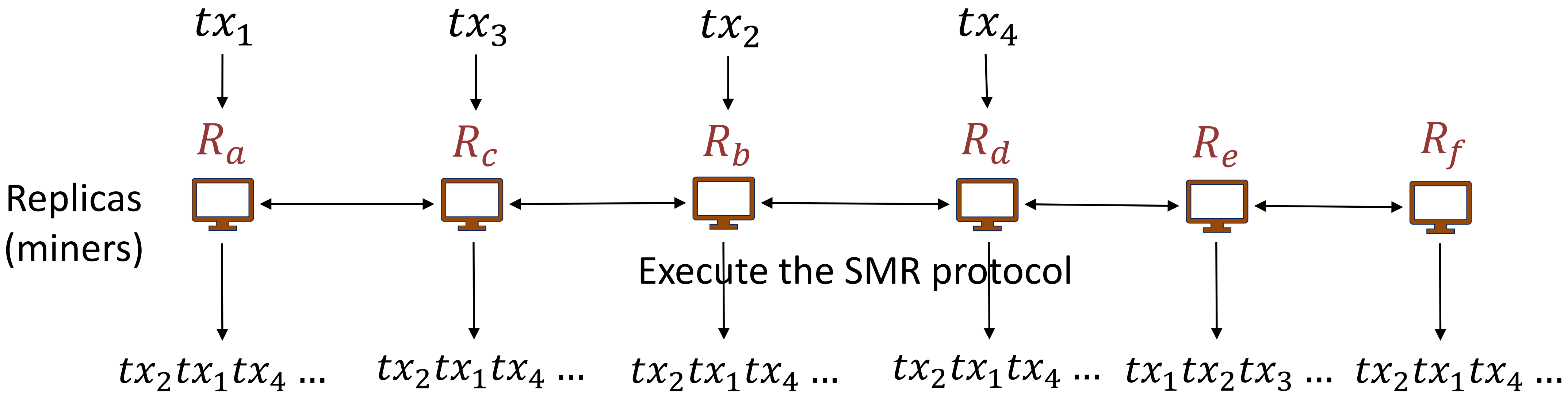


State Machine Replication (SMR)

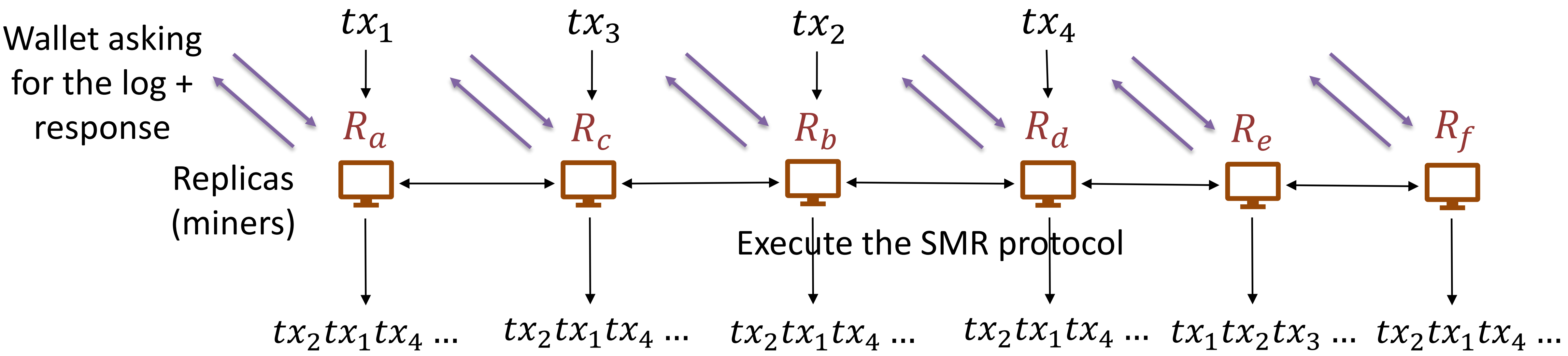
Two parties of SMR:

- *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

$LOG_t^1 = tx_2 tx_1 tx_4 \ldots$

C_2

$LOG_t^2 = tx_2 tx_1 tx_4 \ldots$

Clients (Wallets)

Clients (Wallets)

C_3

$LOG_t^3 = tx_2 tx_1 tx_4 \ldots$

C_4

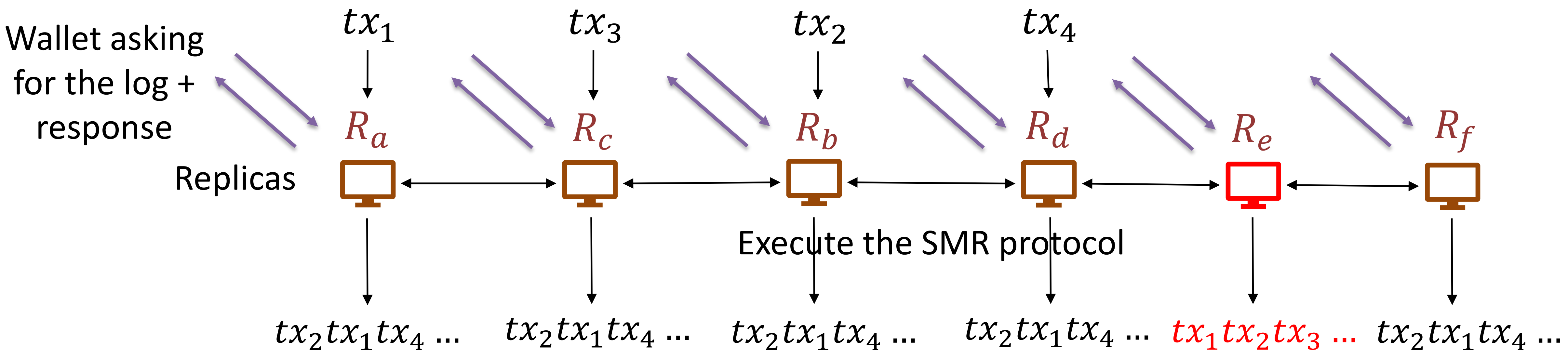
$LOG_t^4 = tx_2 tx_1 tx_4 \ldots$

Wallets are an example of a client.

Wallets ask the replicas what the correct log is.

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

State Machine Replication (SMR)



C_1

$LOG_t^1 = tx_2 tx_1 tx_4 \ldots$

C_2

$LOG_t^2 = tx_2 tx_1 tx_4 \ldots$

Clients (Wallets)

Clients (Wallets)

C_3

$LOG_t^3 = tx_2 tx_1 tx_4 \ldots$

C_4

$LOG_t^4 = tx_2 tx_1 tx_4 \ldots$

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!

Security for SMR: Definitions

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

- Suppose we have sequences $A = tx_1 tx_2$ and $\mathrm{B} = tx_3 tx_4$. What is $A||B$?

$$A||B = tx_1 tx_2 tx_3 tx_4$$

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 $A = tx_1 tx_2 tx_3 tx_4$, $B = tx_1 tx_2 tx_3$ and $D = tx_1 tx_2 tx_4$.

- 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

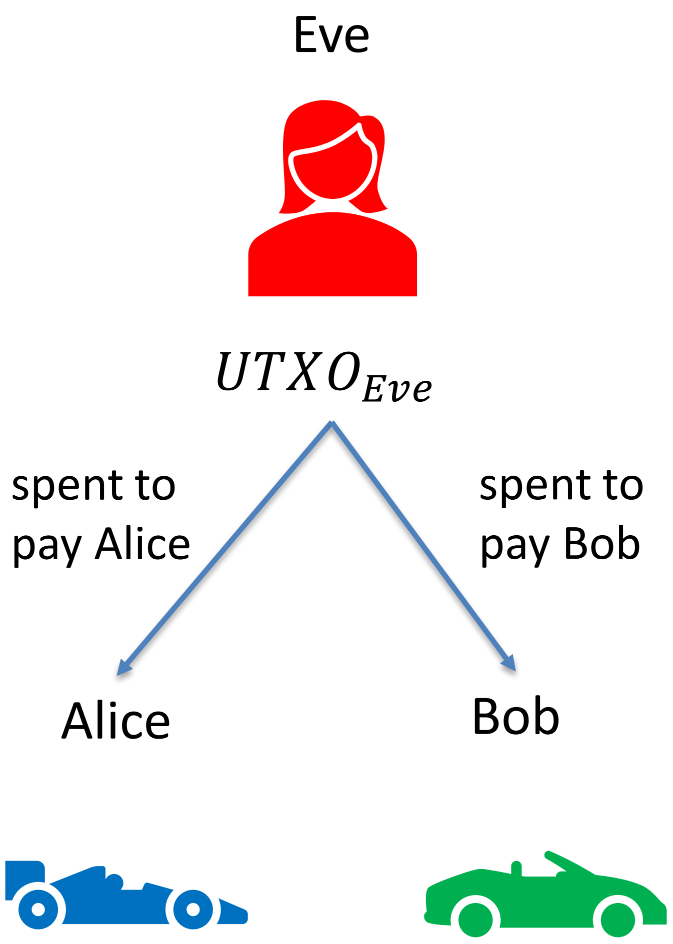
Why is safety important?

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.

$UTXO_{Eve}$

spent to pay Alice → Alice

spent to pay Bob → Bob

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.

Recap so far

- Byzantine Generals Problem
- Definition of Byzantine adversary
  - **Byzantine:** Adversarial nodes can deviate from the protocol arbitrarily!
- Synchronous and asynchronous networks
  - **Synchronous network:** known upper bound Δ on network delay
- Byzantine Broadcast
- Dolev-Strong (1983)
- State Machine Replication (SMR)
- Security properties for SMR protocols: Safety and Liveness

Sybil Attack

How to select the nodes that participate in consensus?

Two variants:

- *Permissioned:* There is a *fixed* set of nodes (as previous).
- *Permissionless*: Anyone is free to join the protocol at any time.

Can we accept any node that has a signing key to participate in consensus?

Sybil Attack!

Sybil Attack

How to select the nodes that participate in consensus?

Two variants:
- *Permissioned:* There is a *fixed* set of nodes (previous lecture).
- *Permissionless*: Anyone is free to join the protocol at any time.

Can we accept any node that has a signing key to participate in consensus?

In a **sybil attack**, a single adversary impersonates many different nodes, outnumbering the honest nodes and potentially disrupting consensus.

Sybil Resistance

Consensus protocols with Sybil resistance are typically based on a bounded (scarce) resource:

	Resource dedicated to the protocol	Some Example Blockchains
Proof-of-Work	Total computational power	Bitcoin, PoW Ethereum…
Proof-of-Stake	Total number of coins	Algorand, Cardano, Cosmos, PoS Ethereum…
Proof-of-Space/Time	Total storage across time	Chia, Filecoin…

How does Proof-of-Work prevent Sybil attacks?

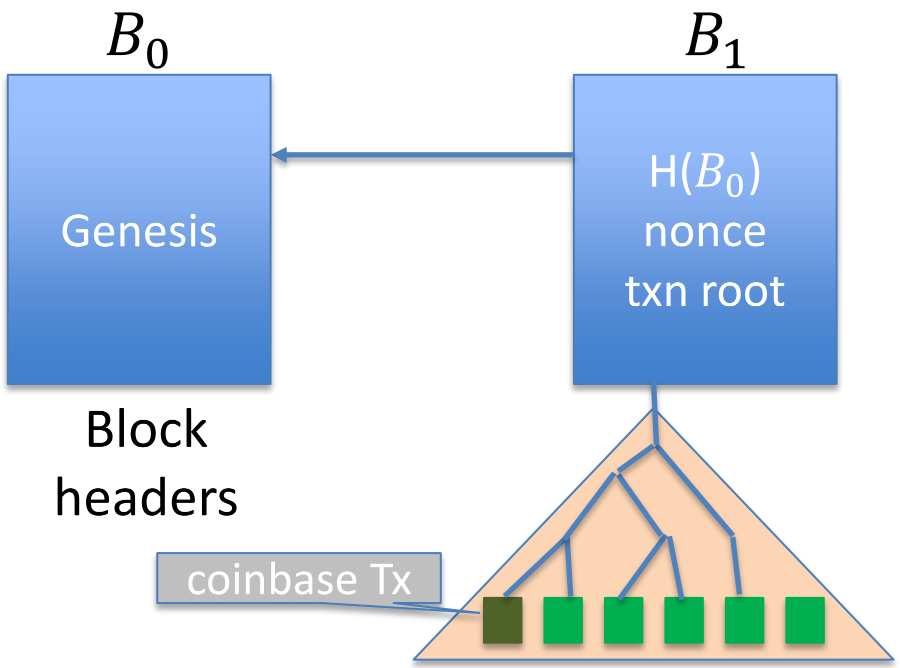
We assume that the adversary controls a small fraction of the scarce resource!
Resource gives the power to influence the protocol.
Adversary has less influence than honest nodes.

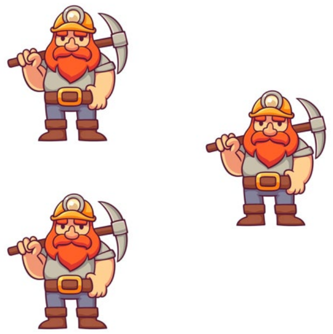
Bitcoin: Mining

To mine a new block, a miner must find *nonce* such that

$$H(h_{prev}, \text{txn root}, \text{nonce}) < \text{Target} = \frac{2^{256}}{D}$$

Each miner tries different nonces until one of them finds a nonce that satisfies the above equation.





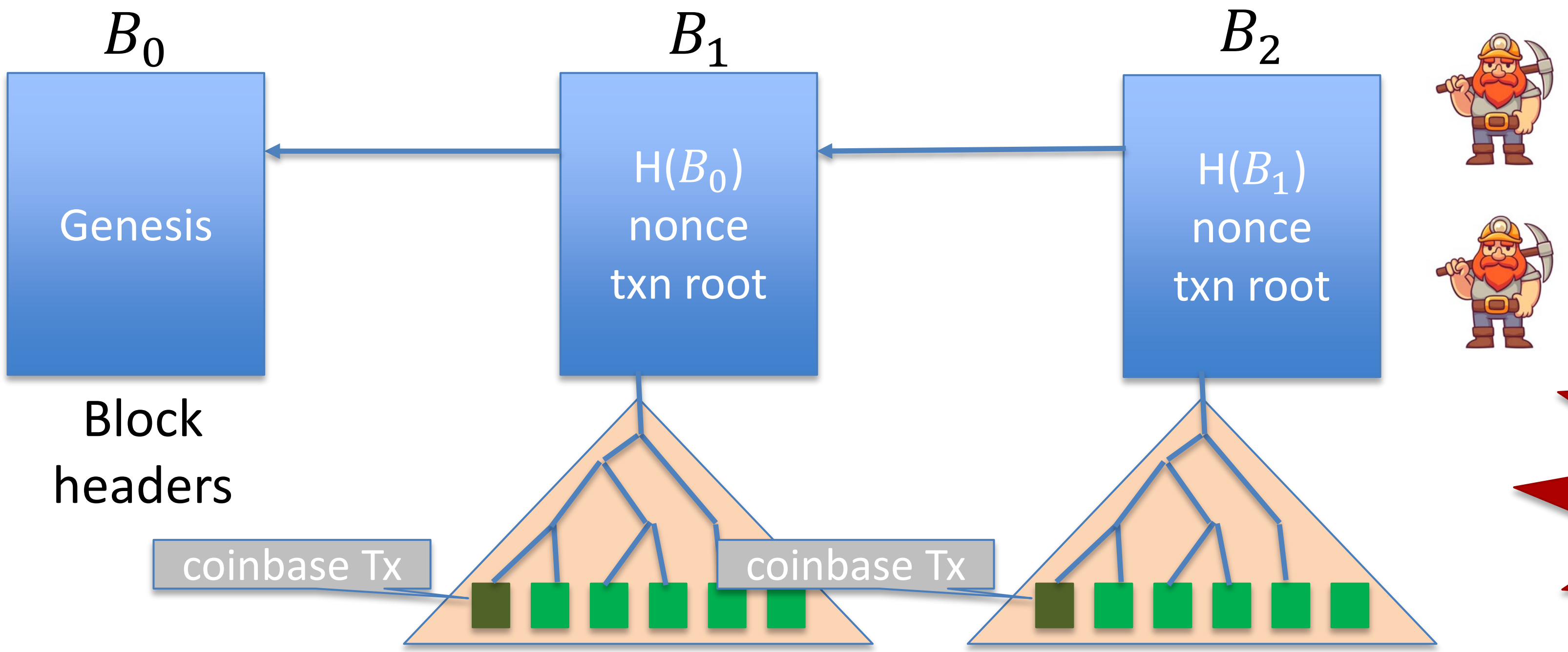
Bitcoin: Mining

To mine a new block, a miner must find *nonce* such that

$$H(h_{prev}, \text{txn root}, \text{nonce}) < \text{Target} = \frac{2^{256}}{D}$$

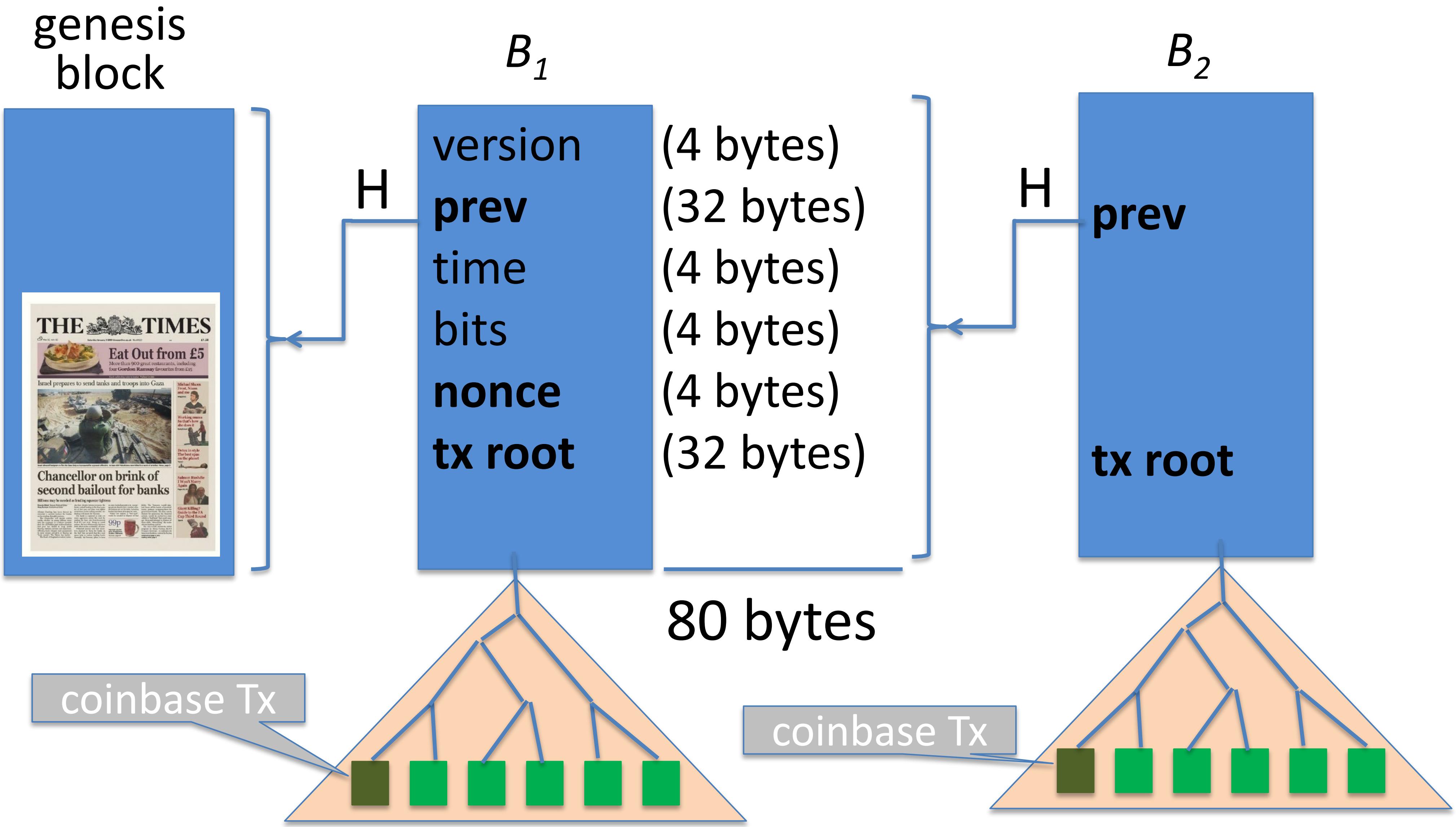
Difficulty: How many nonces on average miners try until finding a block?

Each miner tries different nonces until one of them finds a nonce that satisfies the above equation.

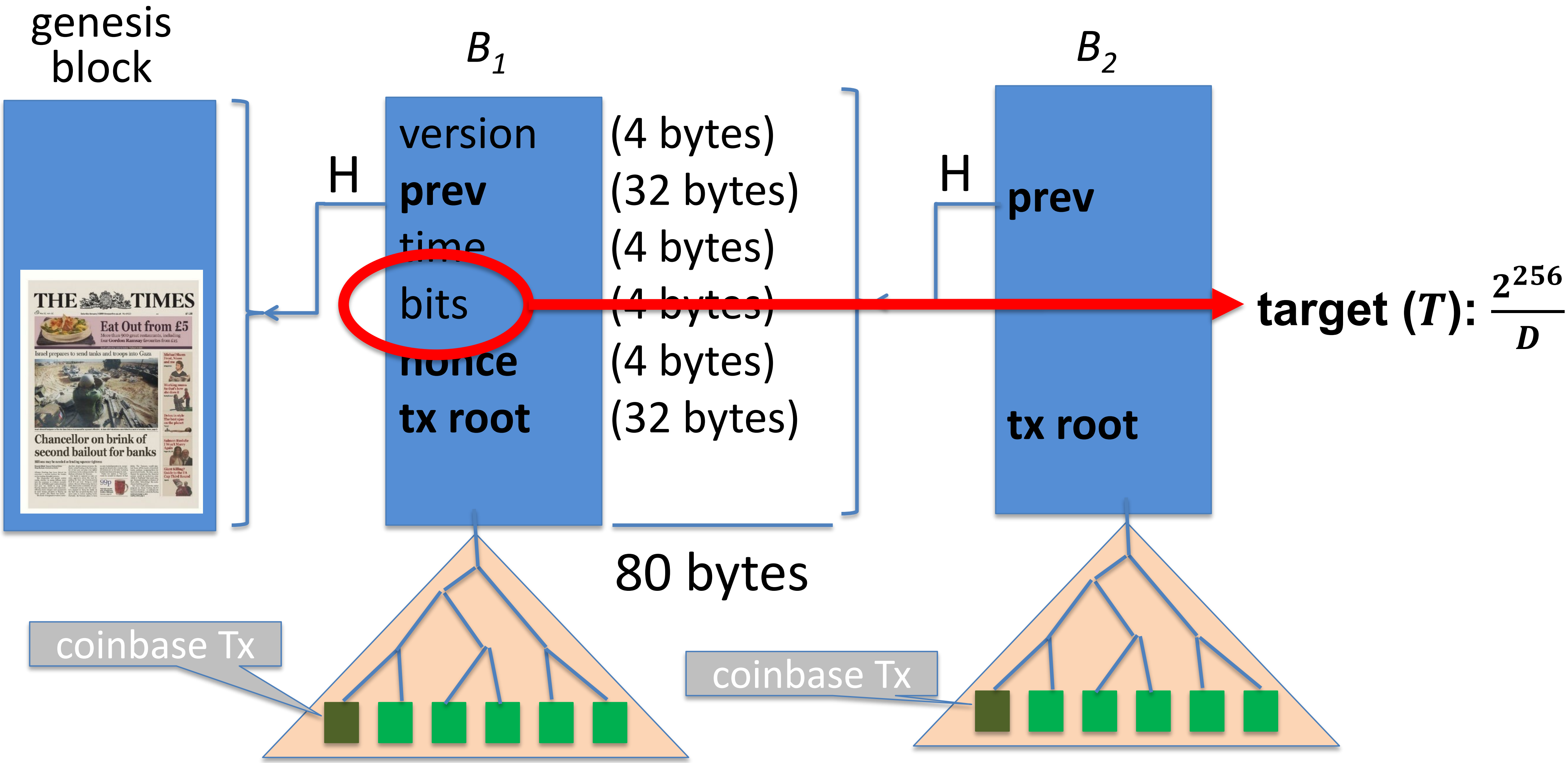


New block: random process but app. once in every 10 minutes

Bitcoin: Mining



Bitcoin: Mining

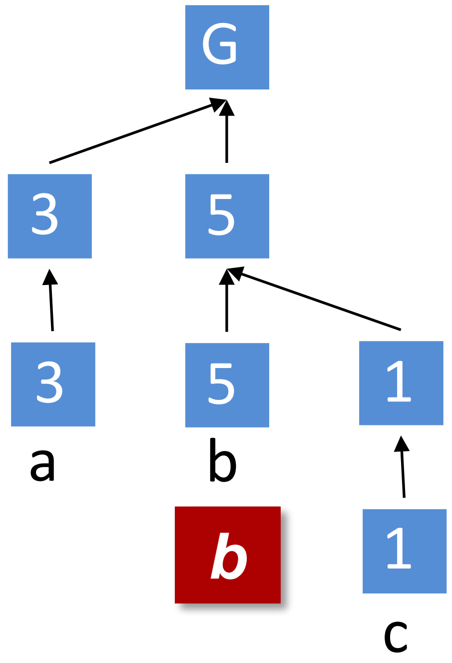


Nakamoto Consensus

Bitcoin uses **Nakamoto consensus**:

- **Fork-choice / proposal rule:** At any given time, each honest miner attempts to extend (i.e., mines on the tip of) the <u>heaviest</u> chain *held* in its view (Ties broken adversarially).

Difficulty values on the blocks



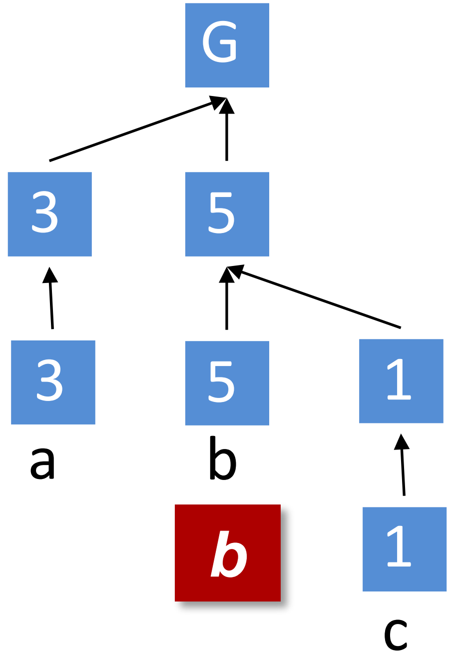
Nakamoto Consensus

Chain with the highest difficulty, i.e, largest sum
of the difficulty D within blocks!

Bitcoin uses **Nakamoto consensus**:

- **Fork-choice / proposal rule:** At any given time, each honest miner attempts to extend (i.e., mines on the tip of) the heaviest chain *held* in its view (Ties broken adversarially).

Difficulty values on the blocks

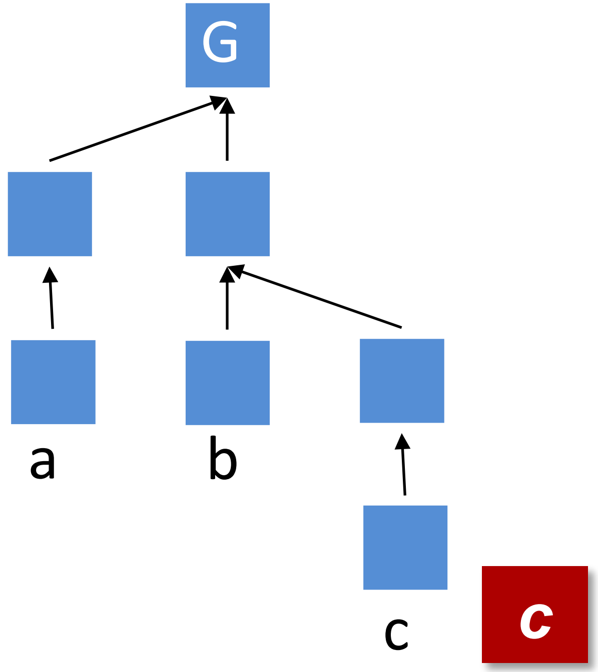


Nakamoto Consensus

Chain with the highest difficulty, i.e, largest sum of the difficulty D within blocks!

Bitcoin uses **Nakamoto consensus**:

- **Fork-choice / proposal rule:** At any given time, each honest miner attempts to extend (i.e., mines on the tip of) the heaviest (longest for us) chain *held* in its view (Ties broken adversarially).



Nakamoto Consensus

Bitcoin uses **Nakamoto consensus**:

- **Fork-choice / proposal rule:** At any given time, each honest miner attempts to extend (i.e., mines on the tip of) the <u>heaviest</u> (longest for us) chain *held* in its view (Ties broken adversarially).
- **Confirmation rule:** Each miner confirms the block (along with its prefix) that is *k*-deep within the longest chain in its view.
  - In practice, $k = 6$.
  - Miners and clients accept the transactions in the latest confirmed block and its prefix <u>as their log</u>.
  - Note that *confirmation* is different from *finalization*.
- **Leader selection rule:** Proof-of-Work.

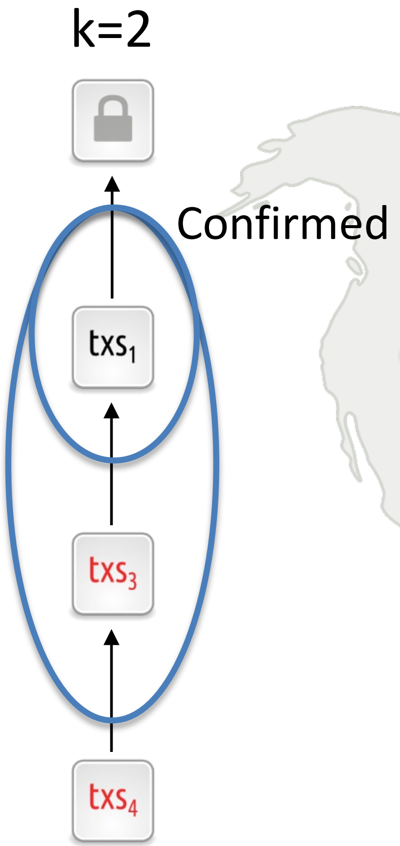
Nakamoto Consensus

Chain with the highest difficulty, i.e, largest sum of the difficulty D within blocks!

Bitcoin uses **Nakamoto consensus**:

- **Fork-choice / proposal rule:** At any given time, each honest miner attempts to extend (i.e., mines on the tip of) the <u>heaviest</u> (longest for us) chain *held* in its view (Ties broken adversarially).
- **Confirmation rule:** Each miner confirms the block (along with its prefix) that is *k*-deep within the longest chain in its view.
  - In practice, $k = 6$.
  - Miners and clients accept the transactions in the latest confirmed block and its prefix <u>as their log</u>.
  - Note that *confirmation* is different from *finalization*.
- **Leader selection rule:** Proof-of-Work.

Nakamoto Consensus



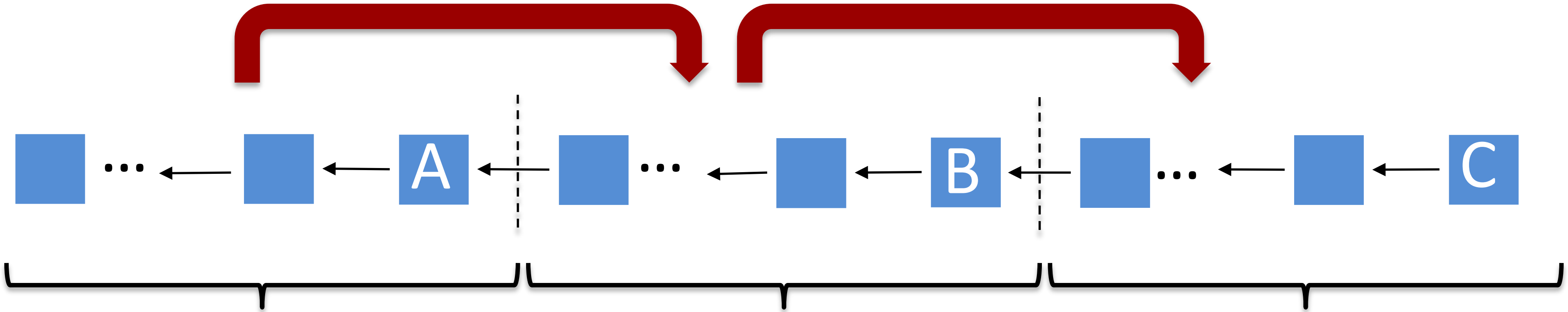
Available **under dynamic participation**

Alice's log: $txs_1 \; txs_3$

Bitcoin: Difficulty Adjustment

New target: $T_2 = T_1 \frac{t_1}{2016 \times 10\ mins}$

New target: $T_3 = T_2 \frac{t_2}{2016 \times 10\ mins}$



2016 blocks
Time it took to mine: t_1(min)
Target: T_1

2016 blocks
Time it took to mine: t_2(min)
Target: T_2

2016 blocks
Time it took to mine: t_3(min)
Target: T_3

t_2: difference between the timestamps in B and A

t_3: difference between the timestamps in C and B

New target is not allowed to be more than 4x old target.
New target is not allowed to be less than ¼ x old target.

The Bitcoin Backbone Protocol with Chains of Variable Difficulty (2016)

Consensus in the Internet Setting

Characterized by *open participation*.

Challenges:

- Adversary can create many Sybil nodes to take over the protocol.
- Honest nodes can come and go at will.

Requirements:

Achieved by Bitcoin!

- Limit adversary's participation.
    - **Sybil resistance (e.g., Proof-of-Work)!**
- Maintain availability (liveness) of the protocol when the honest nodes come and go at will, resulting in changes in the number of nodes.
    - **Dynamic availability!**

Security?

Can we show that Bitcoin is a secure state machine replication (SMR) protocol (satisfies safety and liveness) under synchrony against a Byzantine adversary?

$\beta(t)$ $\in [0,1]$ for all t

Fraction of the mining power controlled by the adversary at time t.

What is the highest $\beta(t)$ for which Bitcoin is secure??

Model for Bitcoin

- Many different miners, each with *infinitesimal* power.

  Total mining rate (growth rate of the chain): λ (1/minutes). In Bitcoin, $\lambda = 1/10$.

- Suppose Adversary is Byzantine and controls $\beta < \frac{1}{2}$ fraction of the mining power.
  - **Adversarial mining rate**: $\lambda_a = \beta\lambda$
  - **Honest mining rate**: $\lambda_h = (1 - \beta)\lambda$

- Network is **synchronous** with a known upper bound Δ on delay.

Reminder: Why is safety important?

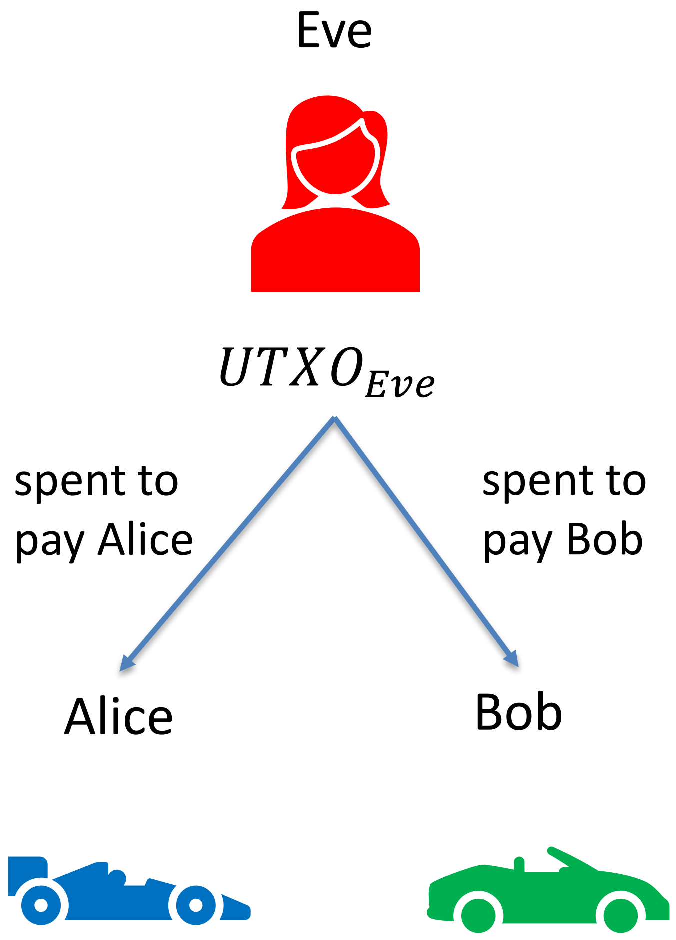
Suppose Eve has a UTXO.

- 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.

$t_0 = 0$ t_1 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.

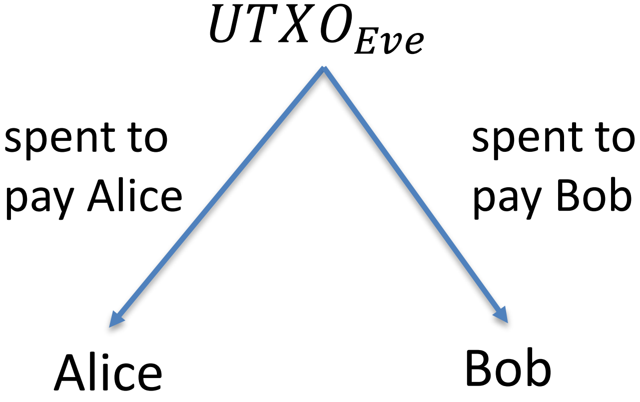


Reminder: Why is safety important?

Eve

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.



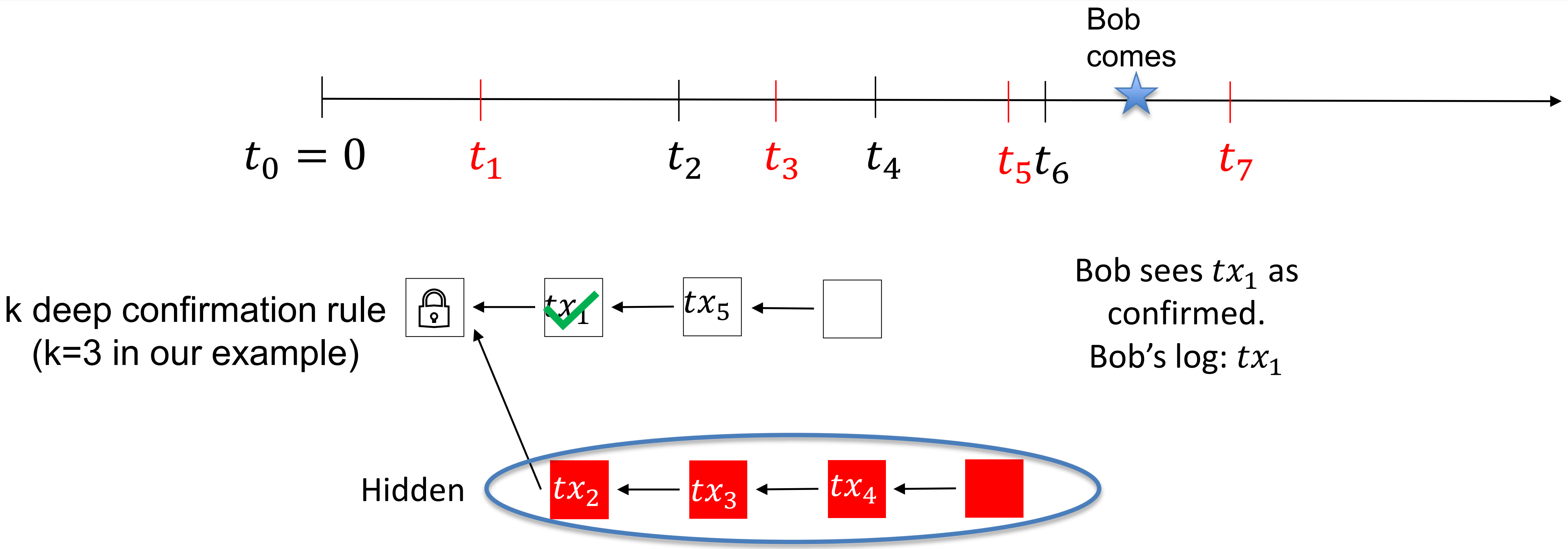
$t_0 = 0$ t_1 safety violation t_2

- 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!

Nakamoto's Private Attack: $\beta \geq 1/2$

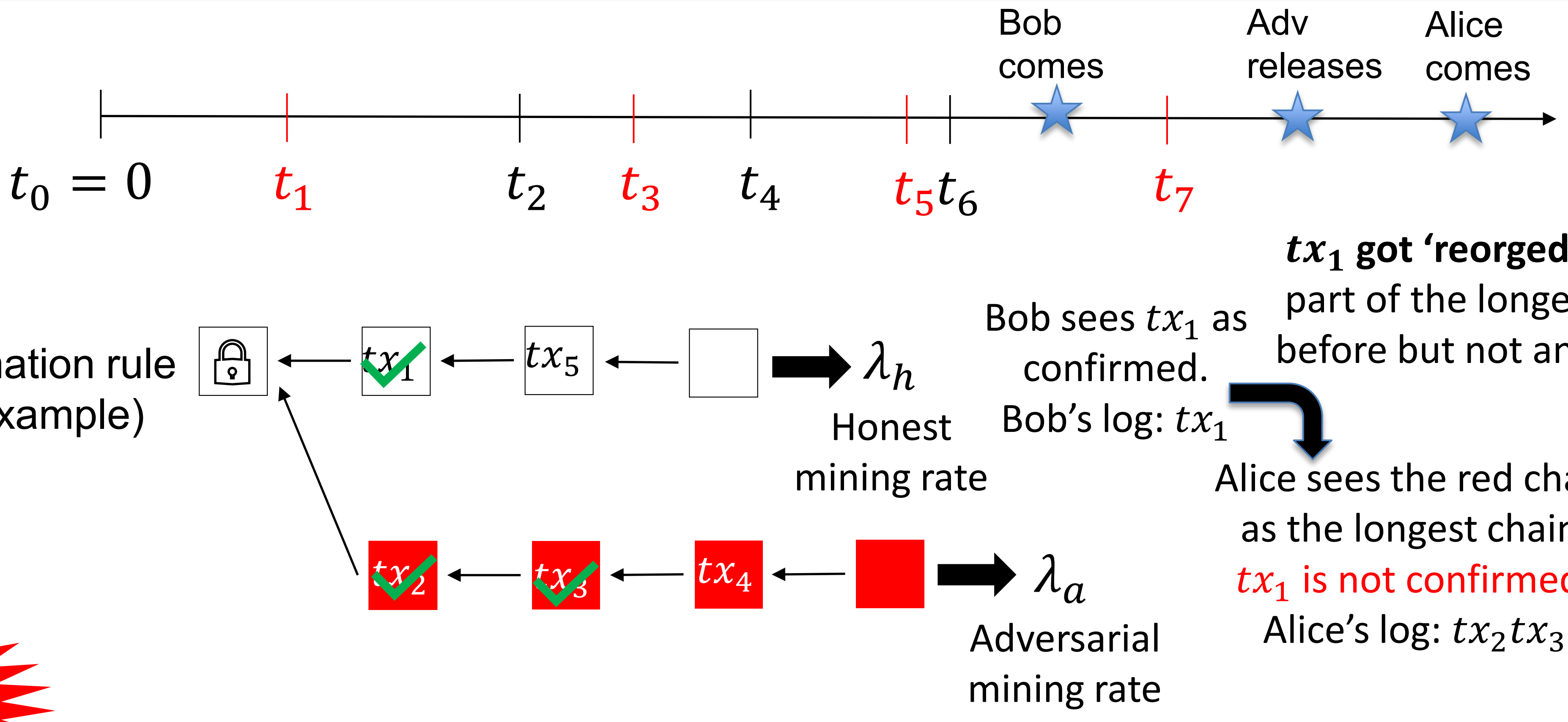


Let's show that Bitcoin is insecure if $\beta(t) \geq 1/2$

A Peer-to-Peer Electronic Cash System (2008)

Nakamoto's Private Attack: $\beta \geq 1/2$

safety violation! double spend!



k deep confirmation rule (k=3 in our example)

Bob sees tx_1 as confirmed.
Bob's log: tx_1

tx_1 got 'reorged': It was part of the longest chain before but not anymore!!

Alice sees the red chain as the longest chain.
tx_1 is not confirmed!
Alice's log: $tx_2 tx_3$

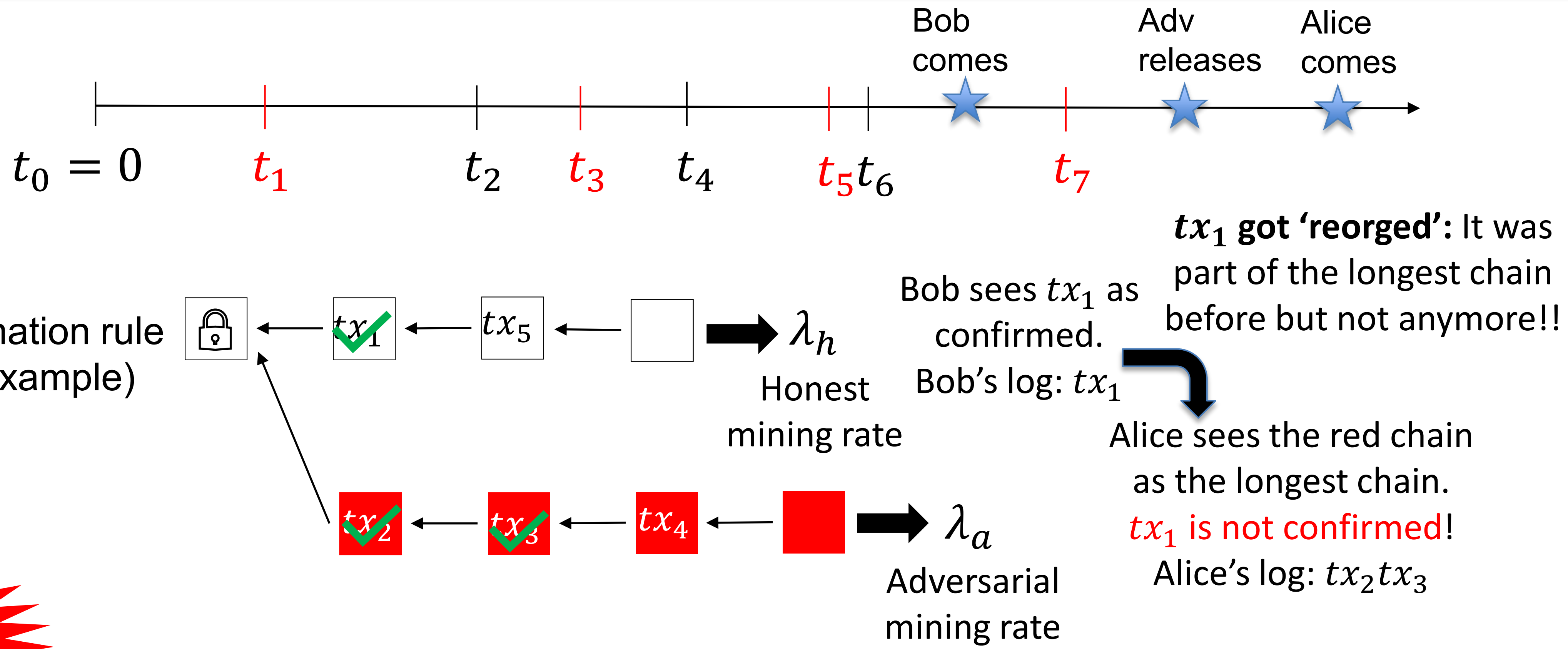
Private attack succeeds!

A Peer-to-Peer Electronic Cash System (2008)

Nakamoto's Private Attack: $\beta \geq 1/2$

safety violation! double spend!

k deep confirmation rule
(k=3 in our example)



Private attack succeeds!

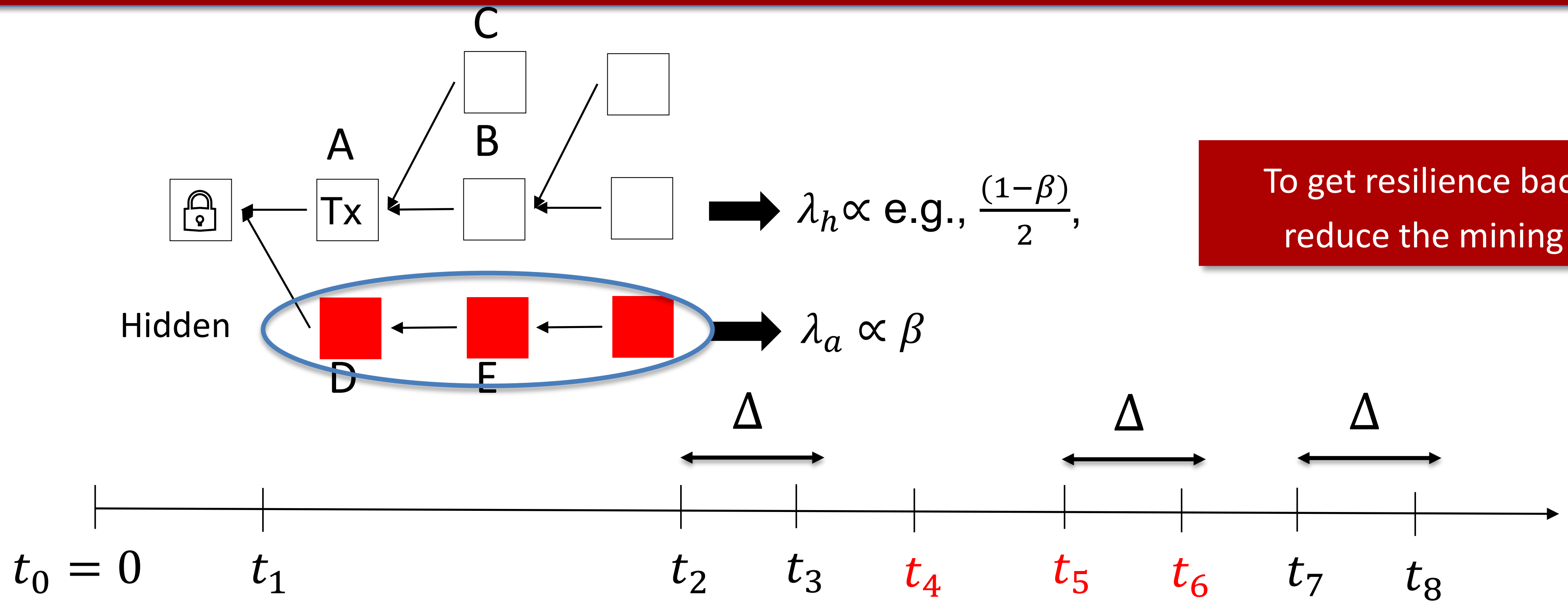
Private attack (mostly) succeeds if $\lambda_a \geq \lambda_h$, i.e., if $\beta \geq 1 - \beta$, i.e., if $\beta \geq \frac{1}{2}$.

Private attack (mostly) fails if $\lambda_a < \lambda_h$, i.e., if $\beta < 1 - \beta$, i.e., if $\beta < \frac{1}{2}$.

Can another attack succeed?

A Peer-to-Peer Electronic Cash System (2008)

Forking



To get resilience back to $\frac{1}{2}$, reduce the mining rate!

Multiple honest blocks at the same height due to network delay.
Adversary's chain grows at rate proportional to (shown by $\propto$) β!
Honest miners' chain grows at rate less than $1 - \beta$ because of forking!
Now, adversary succeeds if $\beta \geq \frac{(1-\beta)}{2}$, which implies $\beta \geq \frac{1}{3}$!!

Reminder for SMR Security

Let's recall the security definition for state machine replication (SMR) protocols. Let ch_t^i denote the confirmed (i.e., k-deep) of a client i at time t.

Safety (Consistency):

- For any two clients i and j, and times t and s: $\boldsymbol{ch_t^i} \preccurlyeq \boldsymbol{ch_s^j}$ (prefix of) or vice versa, i.e., chains are consistent.

No double spend

Liveness:

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

No censorship

Security Theorem

Theorem: If $\beta < 1/2$, there exists a small enough mining rate $\lambda(\Delta, \beta) = \lambda_a + \lambda_h$ such that Bitcoin satisfies safety and liveness except with error probability $\epsilon = e^{-\Omega(k)}$ under synchronous network (recall that k is used in the k deep confirmation rule).

- $e^{-\Omega(k)}$ is the error probability for confirmation.
- Latest result for bounding the error probability as a function of k:

$$\epsilon \leq \left(2 + 2\sqrt{\frac{1-\beta}{\beta}}\right)\big(4\beta(1-\beta)\big)^k$$

- We say 'confirmation' instead of finalization because when you *confirm* a block or transaction, you *confirm* it with an error probability…
- …unlike *finalizing* a block where there is no error probability*.

The Bitcoin Backbone Protocol: Analysis and Applications (2015)
Analysis of the Blockchain Protocol in Asynchronous Networks (2016)
Analysis of Nakamoto Consensus (2019)
Everything is a Race and Nakamoto Always Wins (2022)
Bitcoin's Latency–Security Analysis Made Simple (2022)

Security Theorem

Theorem: If $\beta < 1/2$, there exists a small enough mining rate $\lambda(\Delta, \beta) = \lambda_a + \lambda_h$ such that Bitcoin satisfies safety and liveness except with error probability $\epsilon = e^{-\Omega(k)}$ under synchronous network (recall that k is used in the k deep confirmation rule).

- $e^{-\Omega(k)}$ is the error probability for confirmation.
- Latest result for bounding the error probability as a function of k:

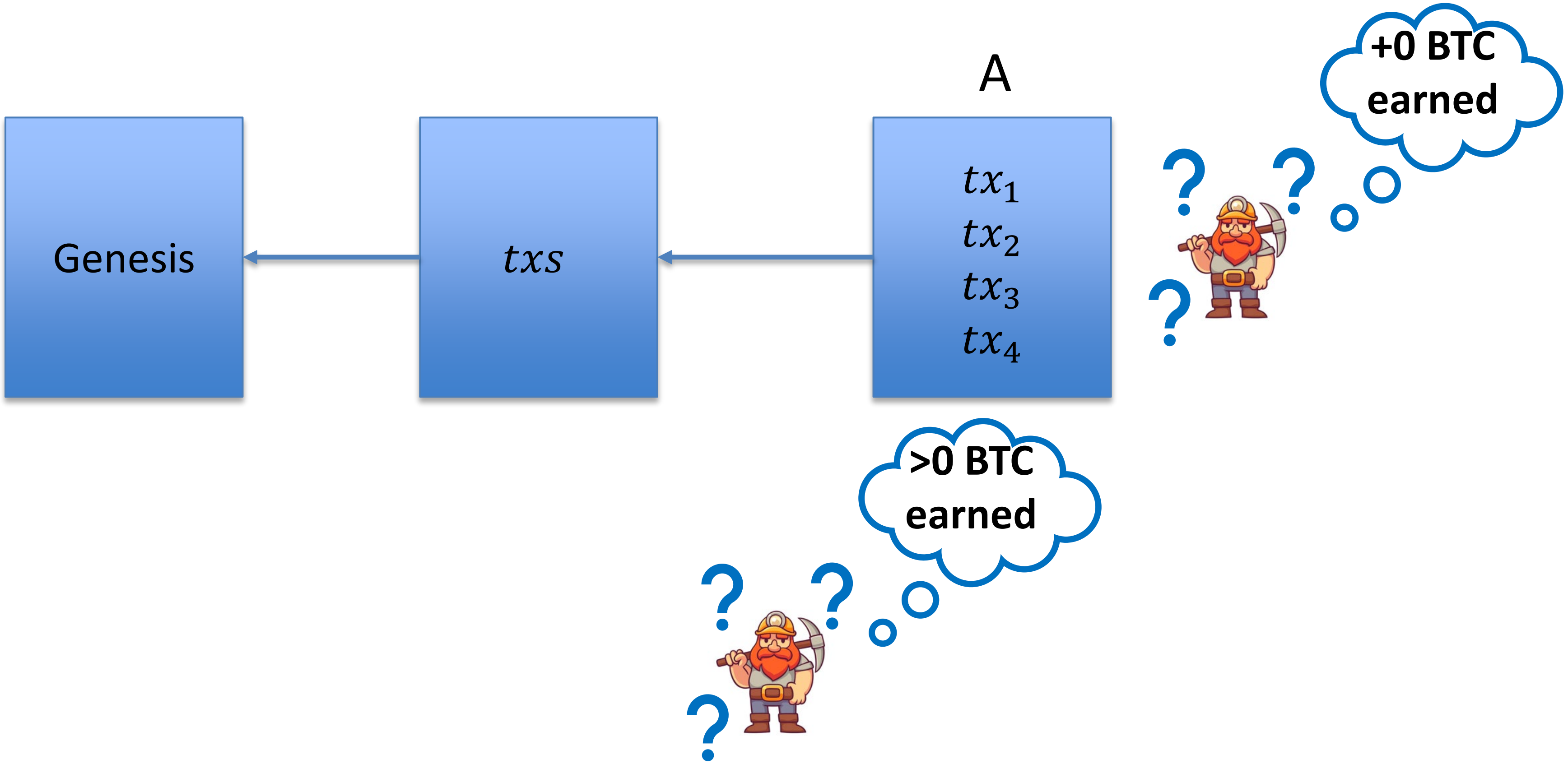
$$\epsilon \leq \left(2 + 2\sqrt{\frac{1-\beta}{\beta}}\right)\left(4\beta(1-\beta)\right)^k$$

- We say 'confirmation' instead of finalization because when you *confirm* a block or transaction, you *confirm* it with an error probability…
- …unlike *finalizing* a block where there is no error probability*.

The Bitcoin Backbone Protocol: Analysis and Applications (2015)
Analysis of the Blockchain Protocol in Asynchronous Networks (2016)
Analysis of Nakamoto Consensus (2019)
Everything is a Race and Nakamoto Always Wins (2022)
Bitcoin's Latency–Security Analysis Made Simple (2022)

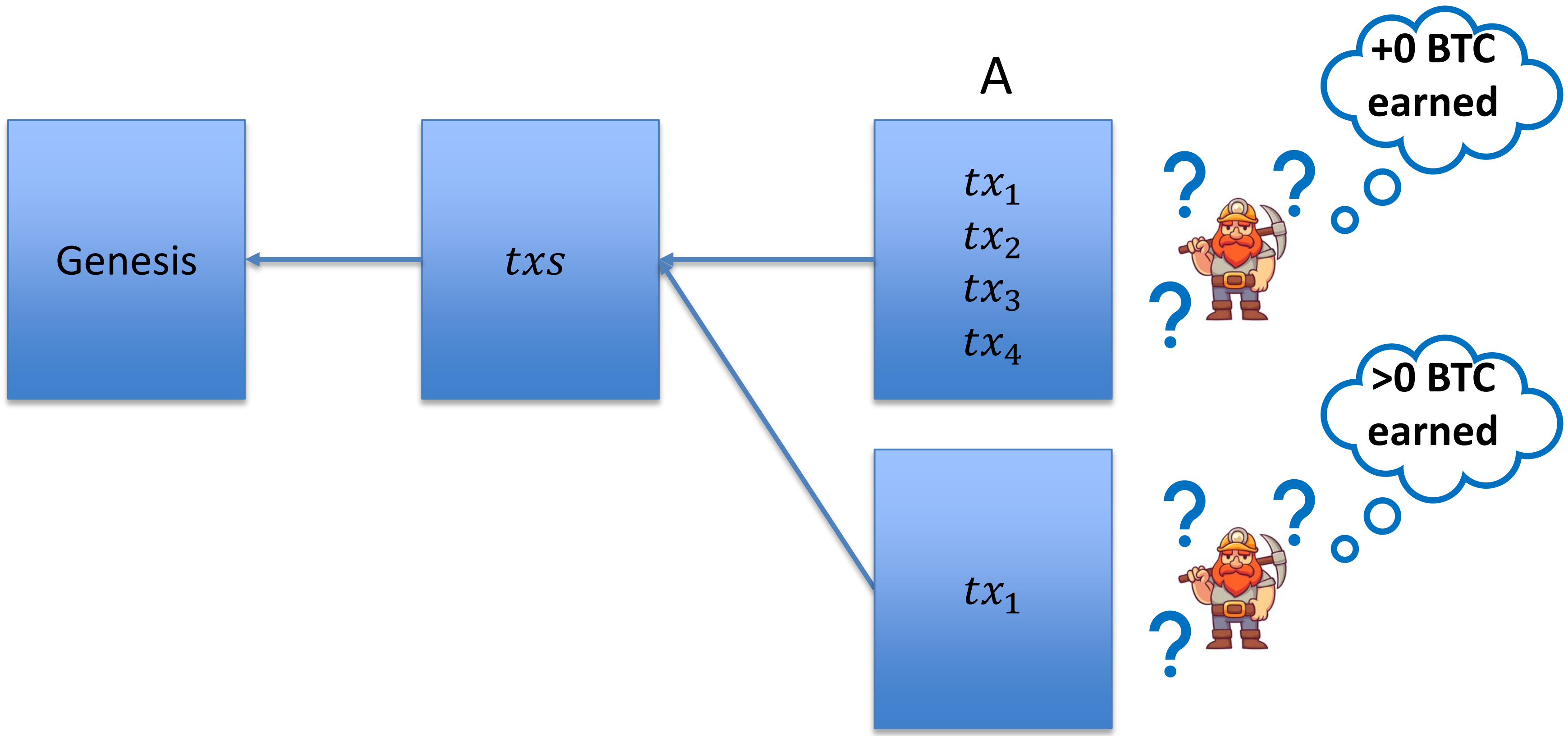
Now, we see why Bitcoin has 1 block every 10 minutes, instead of 1 block every second…

Would $\beta < 1/2$ hold in practice?



Transaction fees
(paid to the miner):
tx_1: 4 BTC
tx_2: 3 BTC
tx_3: 2 BTC
tx_4: 1 BTC

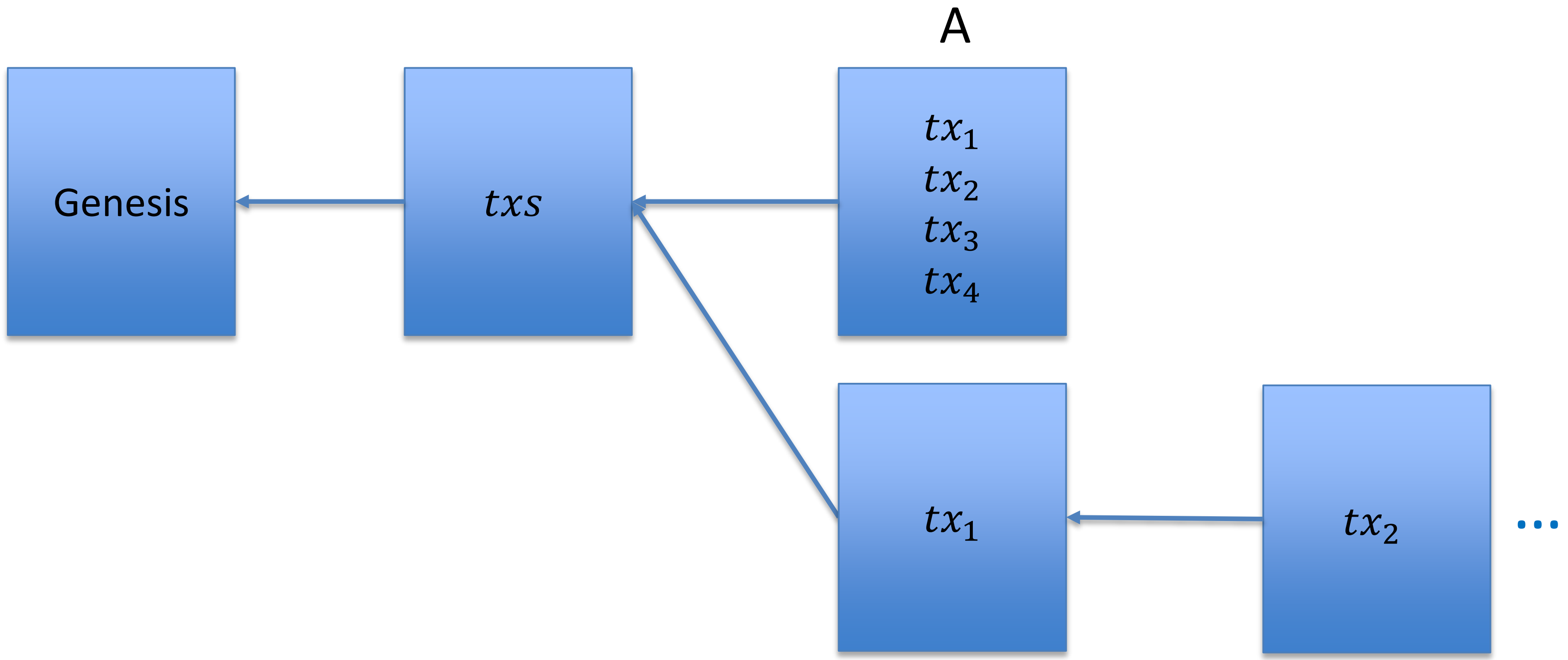
Would $\beta < 1/2$ hold in practice?



Transaction fees
(paid to the miner):

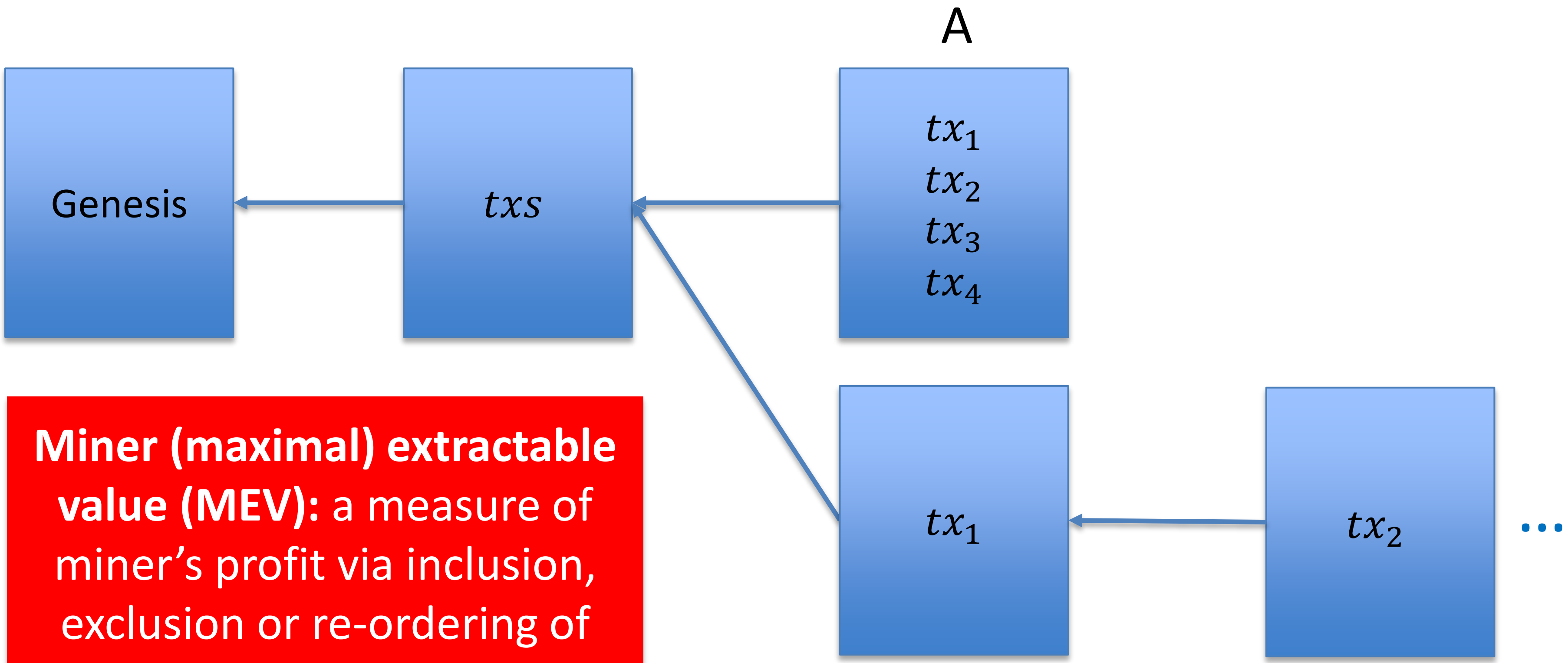
tx_1: 4 BTC
tx_2: 3 BTC
tx_3: 2 BTC
tx_4: 1 BTC

Would $\beta < 1/2$ hold in practice?



Transaction fees
(paid to the miner):
tx_1: 4 BTC
tx_2: 3 BTC
tx_3: 2 BTC
tx_4: 1 BTC

Would $\beta < 1/2$ hold in practice?



Transaction fees
(paid to the miner):
tx_1: 4 BTC
tx_2: 3 BTC
tx_3: 2 BTC
tx_4: 1 BTC

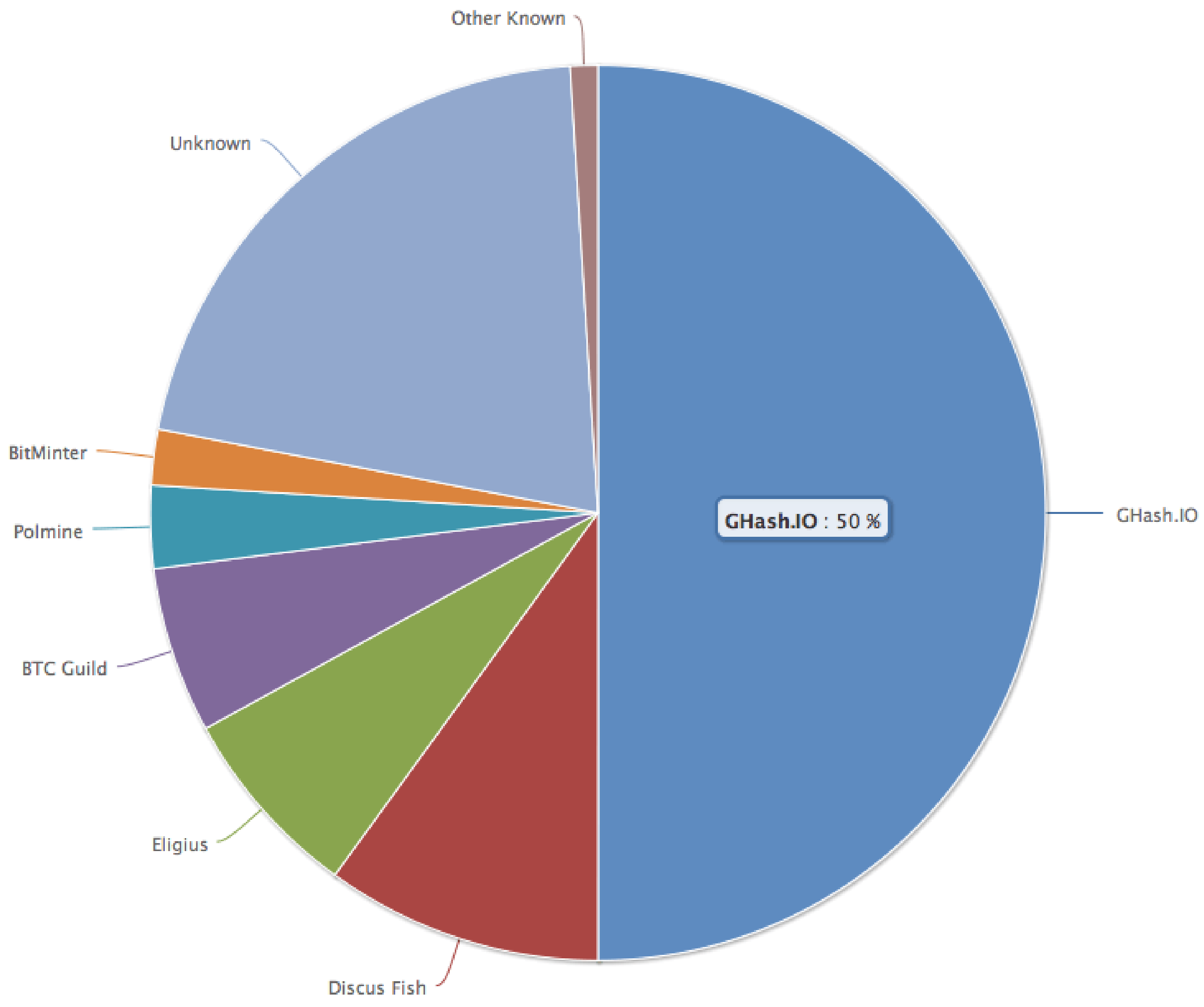
Miner (maximal) extractable value (MEV): a measure of miner's profit via inclusion, exclusion or re-ordering of transactions within its block

- Miners have incentive to violate the protocol!
- Can lead to double spends!

Need to think about incentives!!

MEV gives even more incentive to violate the protocol!!

No Attacks on Bitcoin?



Ghash.IO had >50% in 2014

- Gave up mining power

Why are visible attacks not more frequent?

Miners care about the Bitcoin price?

- Not a valid argument.
- They can 'short' the chain for profit!

Might not always be rational to attack.

No guarantees for the future!

Is Bitcoin the Endgame?

Bitcoin provides Sybil resistance and dynamic availability.

Is it the Endgame for consensus?

No!

Bitcoin is secure only under <u>synchrony</u> and loses security during periods of <u>asynchrony</u>.

It *confirms* blocks with an error probability depending on k, i.e., blocks are not <u>finalized</u>.

Energy consumption?

Next lecture: low-energy consensus using proof-of-stake