Analysis of the Blockchain Protocol in Asynchronous Networks

Rafael Pass and Lior Seeman and abhi shelat

Asynchronous network:

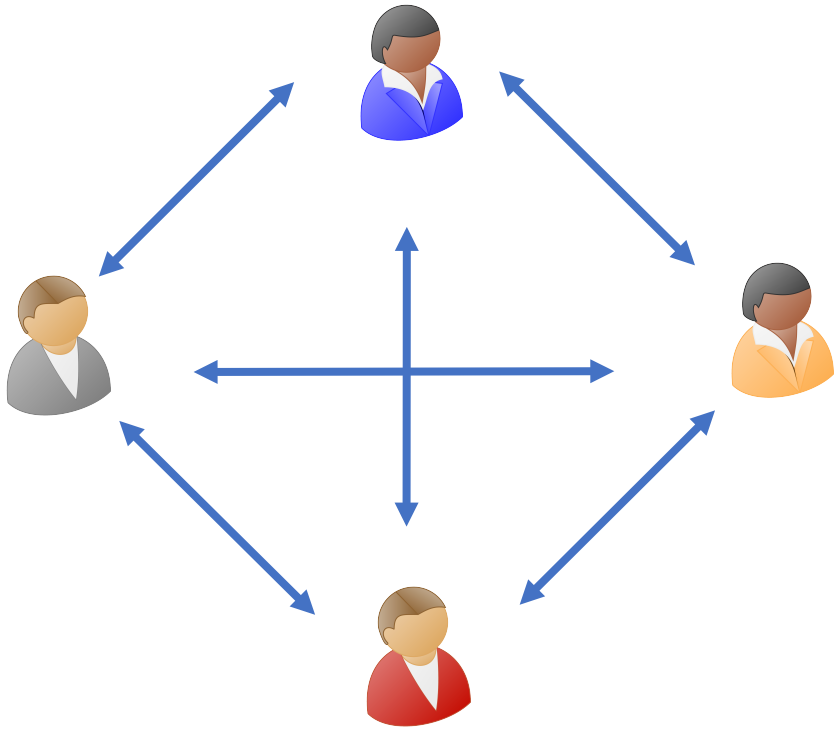
adversary controls scheduling/delivery of messages between honest parties.

Challenges?

Recall, that in the case of proof of work, delaying information sent between parties gives the adversary more computational time.

Thus, if the adversary is able to delay messages from honest parties by an arbitrary amount, they have sufficient time to create a longer dishonest chain. Importantly, if the adversary can delay messages arbitrarily, it does not require a majority of computing power to mount this attack.

Network

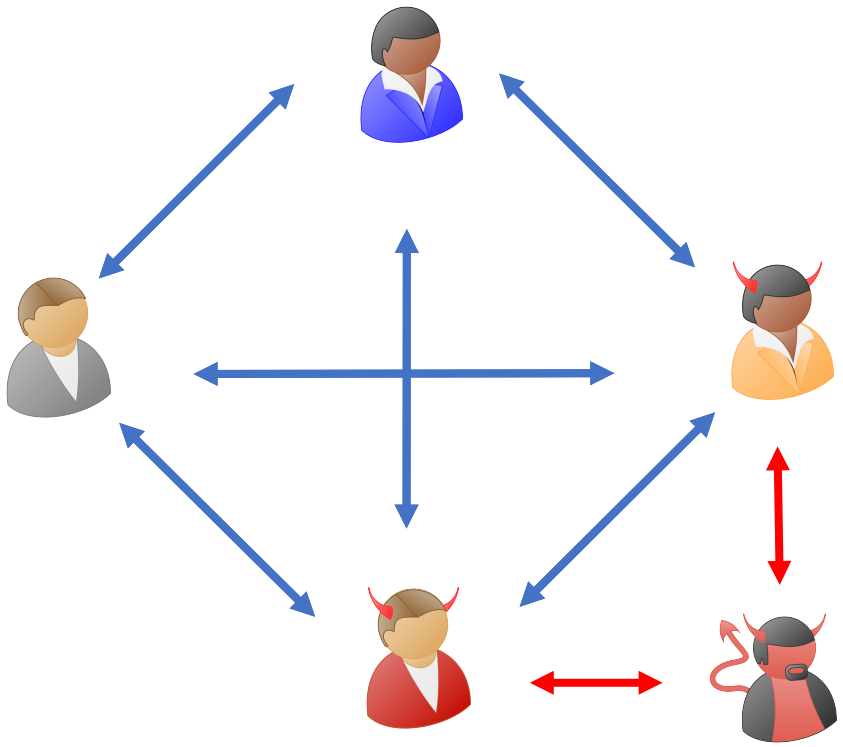


We model the network via broadcast channels that each party has access to.

Adversary

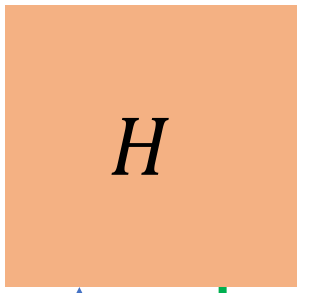
One of the nicest aspects of the blockchain is the fact that it is permissionless. i.e. parties can join and leave the network as they please.

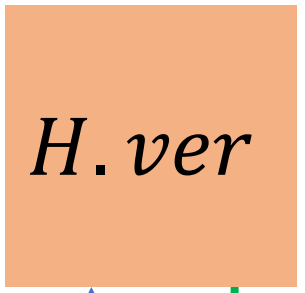
When a party joins, we assume that the adversary decides to either corrupt them, or let them be honest. As long as the fraction remains less than ρ. In the general scenario, a party can decide to switch between being honest and corrupt. This is considered in the paper, but for this talk we assume the static case where a party's role is decided when it joins the network.



Can corrupt up to ρ fraction of the total participants.

Modelling the hash function as a Random Oracle





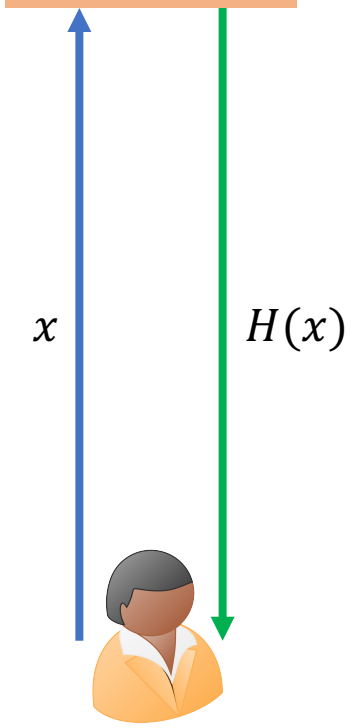
We model the computation of the hash function to be external network. It should be noted that this is just **a model**, and does not exist in reality.

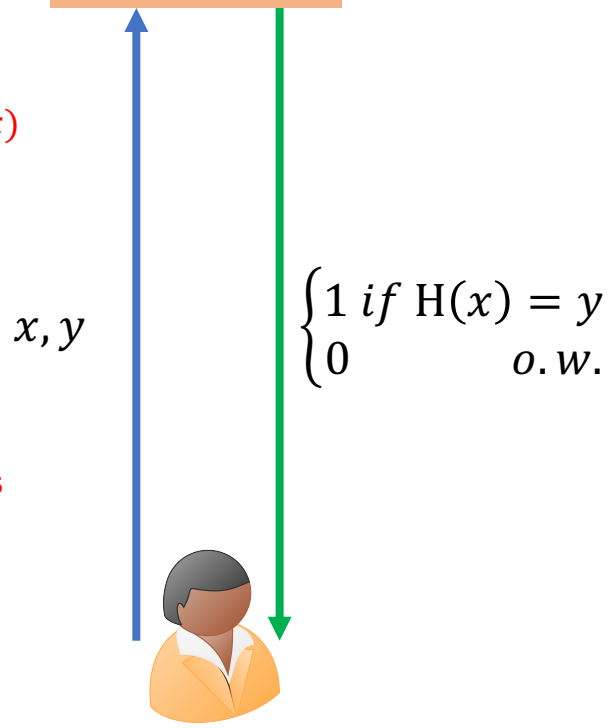
In this model, there exists two global oracles that every party has access to. The adversary cannot control the access of the honest parties to these oracles.

The first oracle, the party queries x and receives $H(x)$ in return. The second oracle is queried with x, y and returns 1 if and only if $H(x) = y$.

While the second oracle can be implemented using the first, we separate them out to give parties asymmetric access to these oracles. At every time step (round), a party is limited to a single query to H, while they can make unlimited queries to $H.ver$. This is modeled to capture the fact that miners often use heuristics to verify the value of the hashes.

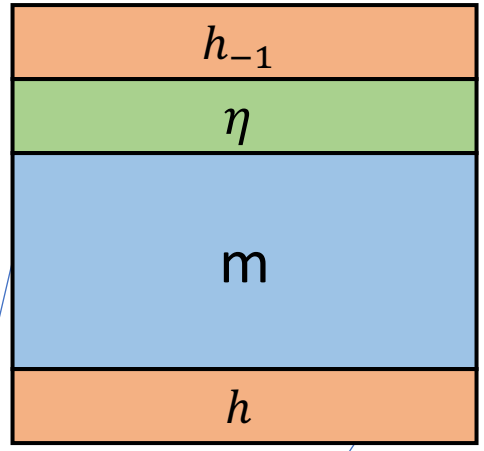
Note that if the adversary corrupts k parties, it can make k queries to the oracle in a single time step.





1 query per player per round

∞ queries per player per round



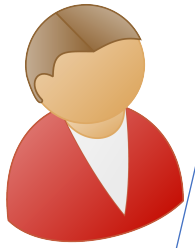
pointer to the previous record

nonce

record (transactions)

pointer to the current record

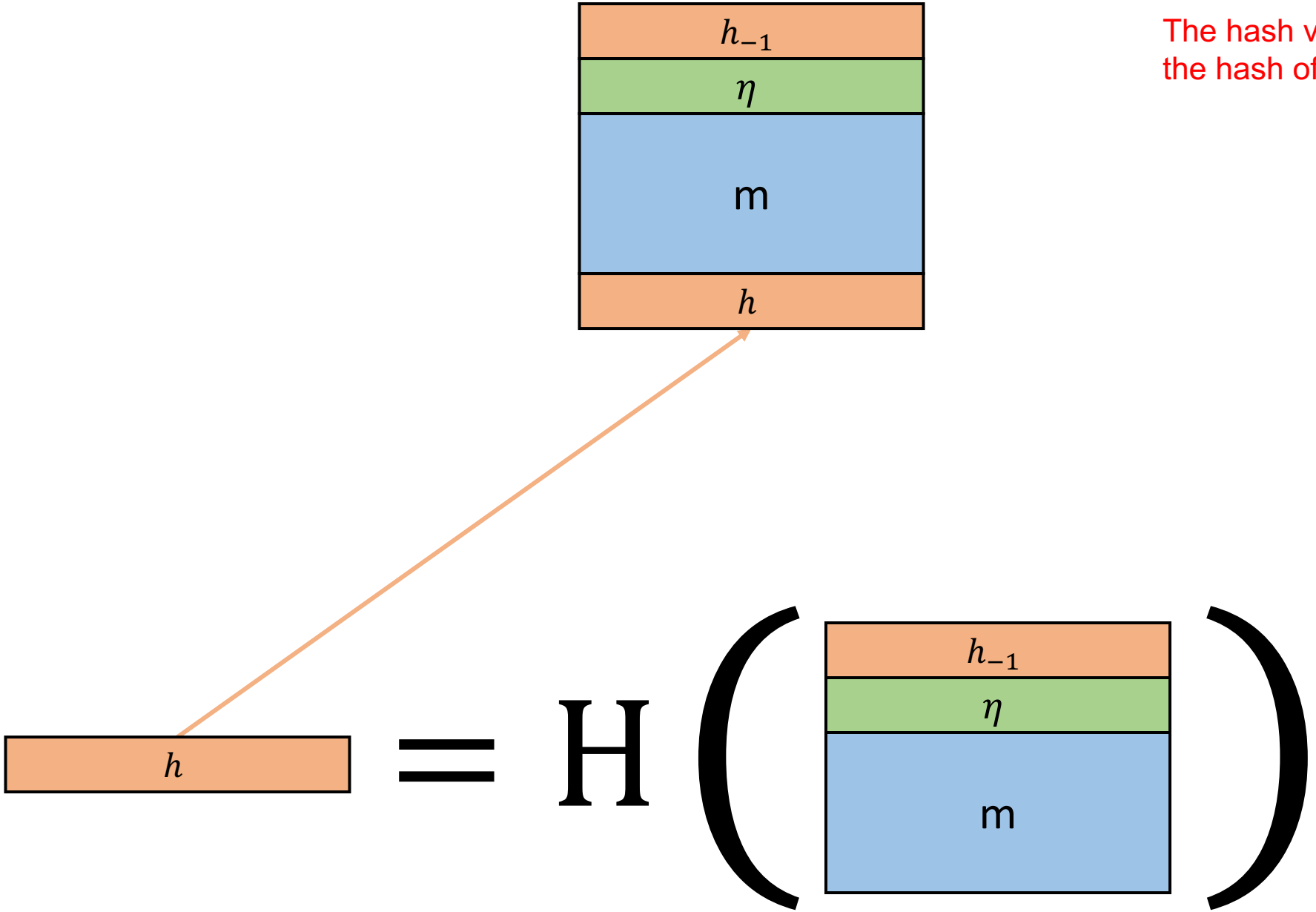
$\vec{m} = \texttt{getRecords}(\text{state})$



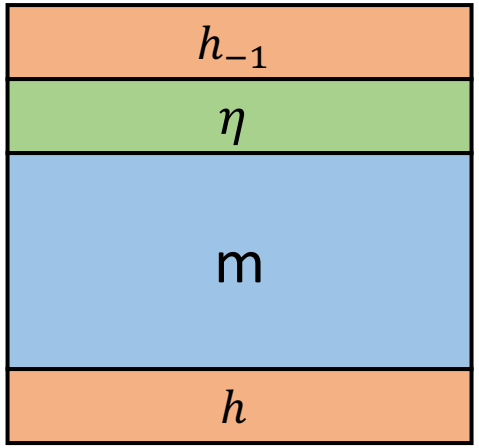
The state that each party maintains is a chains of blocks (blockchain!)

The contents of each block is described in the figure above. The pointer to the preceding block is a (logical) pointer implemented as a hash value.

The actual transactions are stored in the body of the block.



The hash value is computed as the hash of the rest of the block.



We define what it means for a single block to be valid with respect to its predecessor.

This is defined by the blockchain parameter D_p.

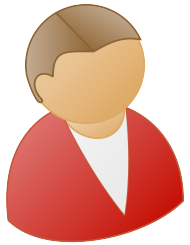
For our model, we assume that the difficulty parameter is fixed throughout execution. But in bitcoin, this is updated every 2 weeks.

For a block to be valid with respect to its predecessor:

$$h < D_p$$

For a randomly chosen η the probability of this event is p

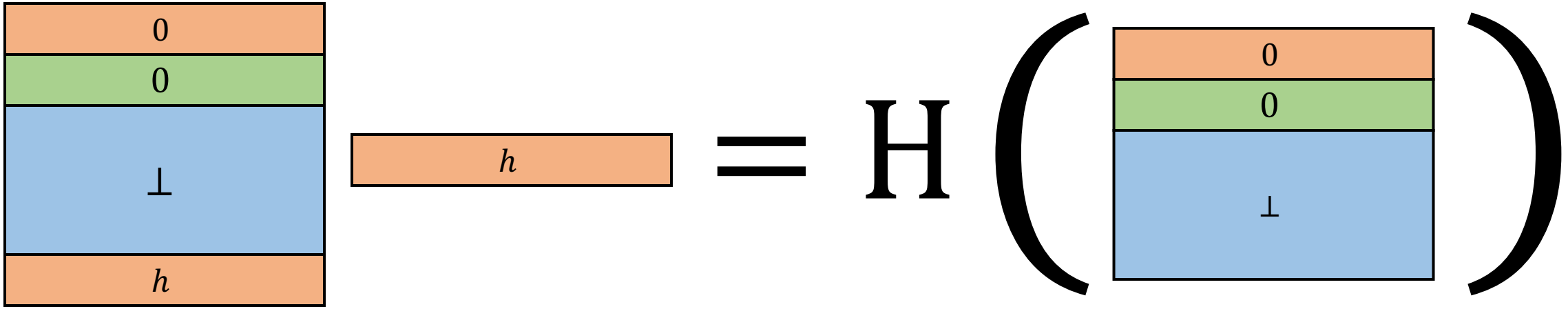
Given the validity of a single block, we now define the validity of the entire state.



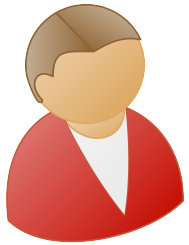
state

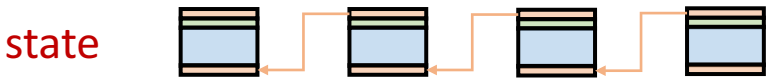
Validity checks for state:

1. Contains the genesis block
2. Each block valid
3. Transactions are valid



Genesis Block





Validity checks for state:

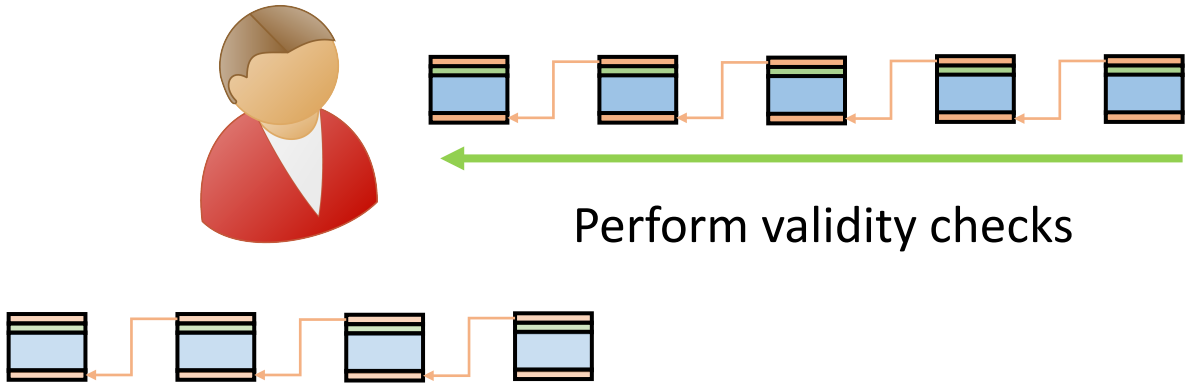
1. Contains the genesis block
2. Each block valid
3. Transactions are valid

Protocol Execution

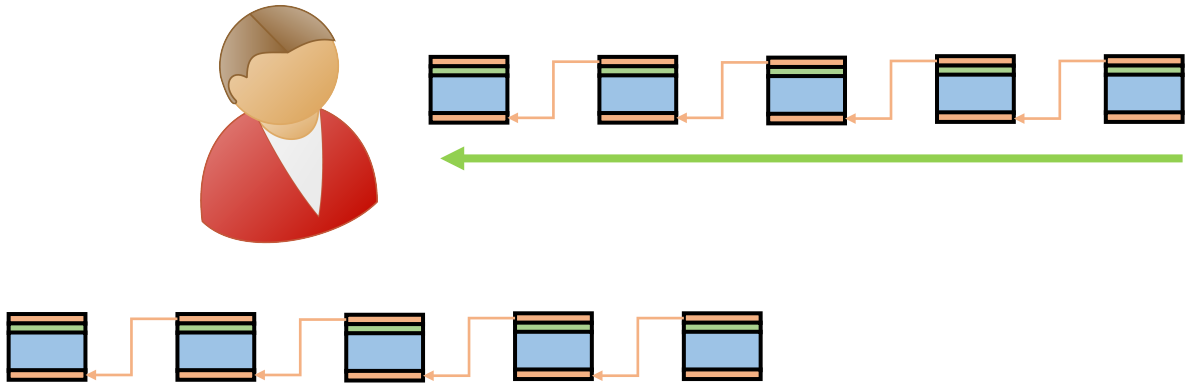
We now move on to the actual execution of the protocol. This is modeled as actions in each time step, which we shall refer to as a round.

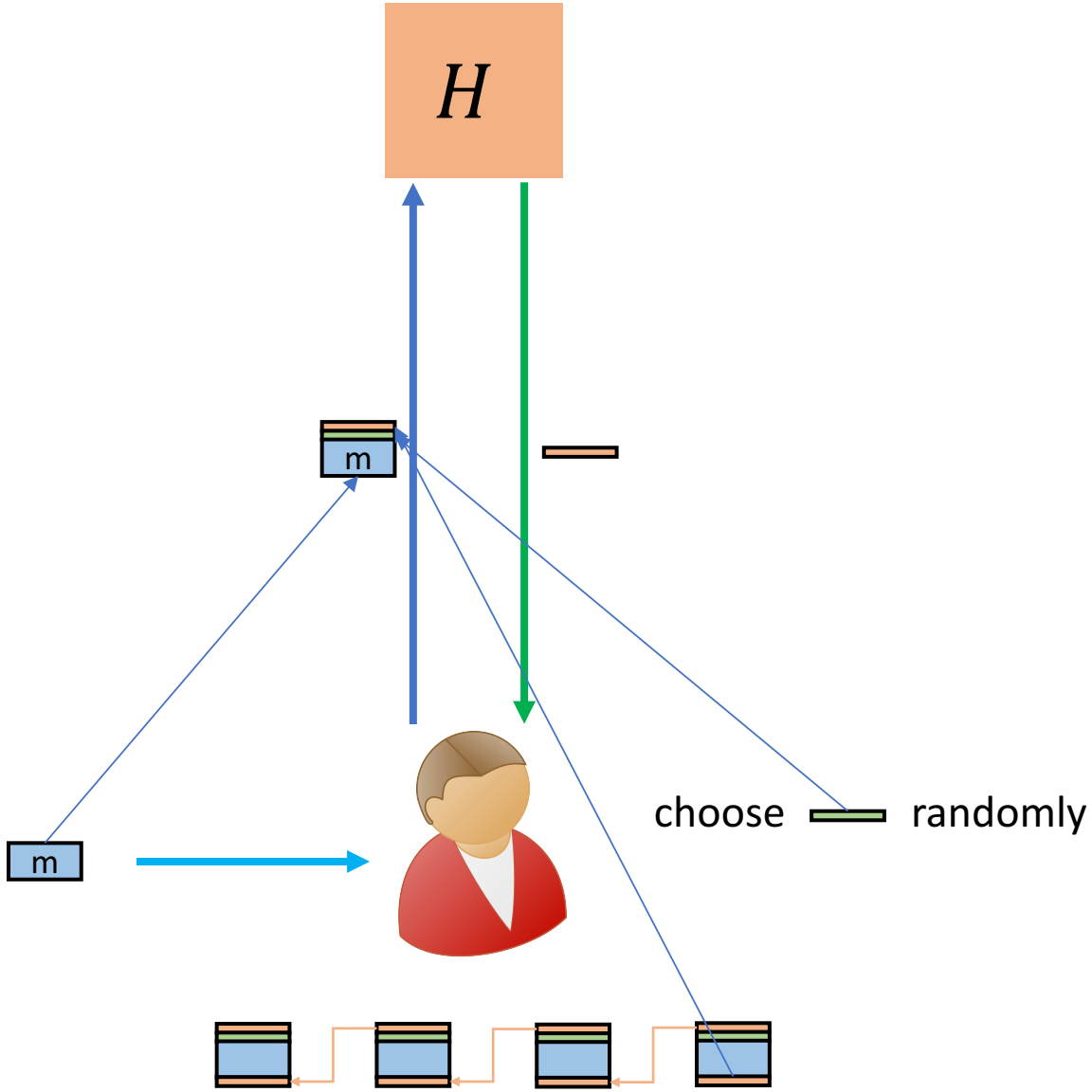
At the beginning of the round, the party may receive a state from the network. It checks the validity of the state.

While this is useful for our model. It is very inefficient to send the entire chain in each round, and the bitcoin blockchain optimizes the amount of information sent.



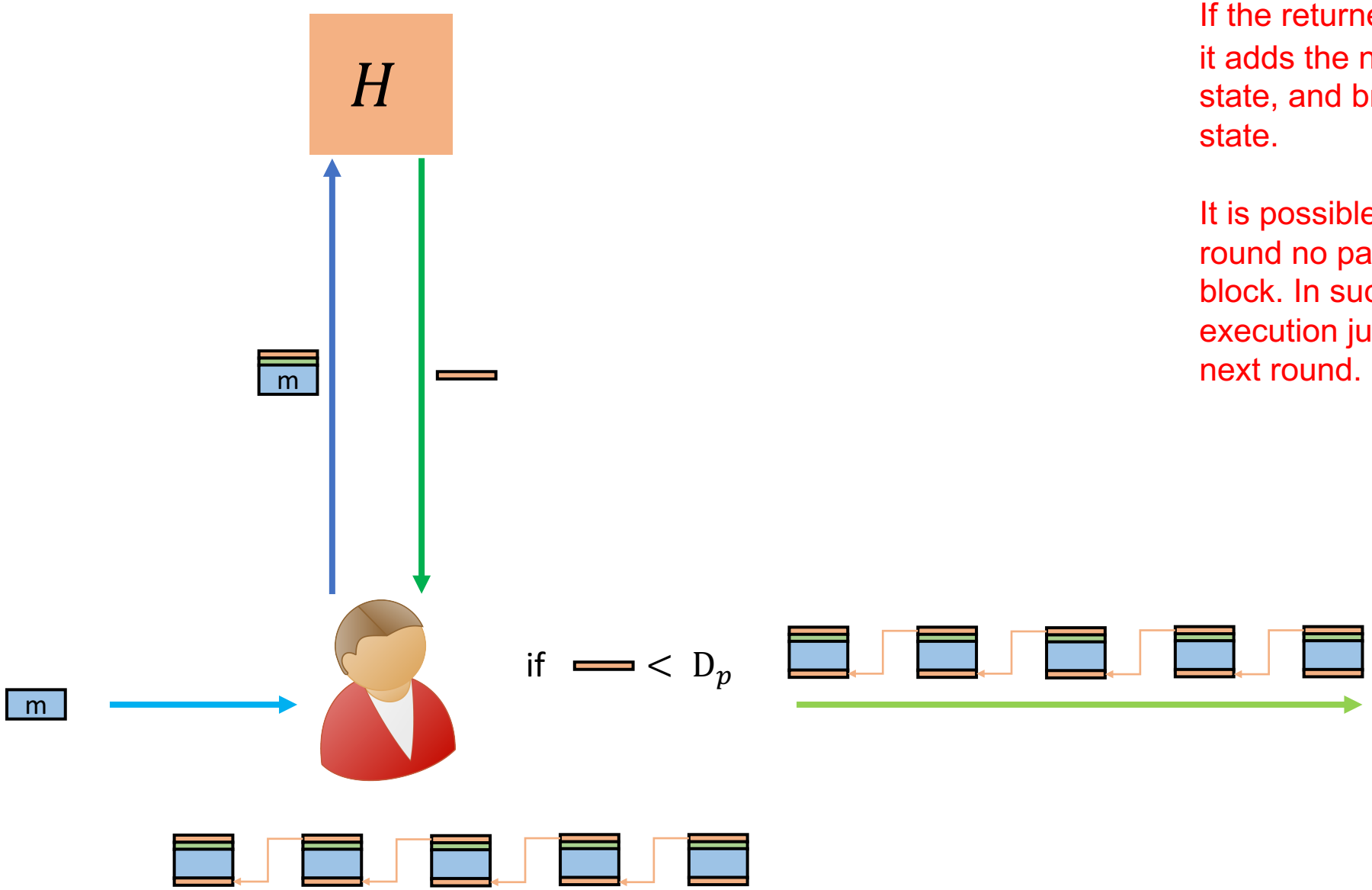
If the state is valid and the chain longer than its own, it updates its state to be the one it has received.





Next it receives a message m. In bitcoin, it hears it on the network.

It attempts to add this to the chain by querying the oracle H with a randomly chosen nonce.

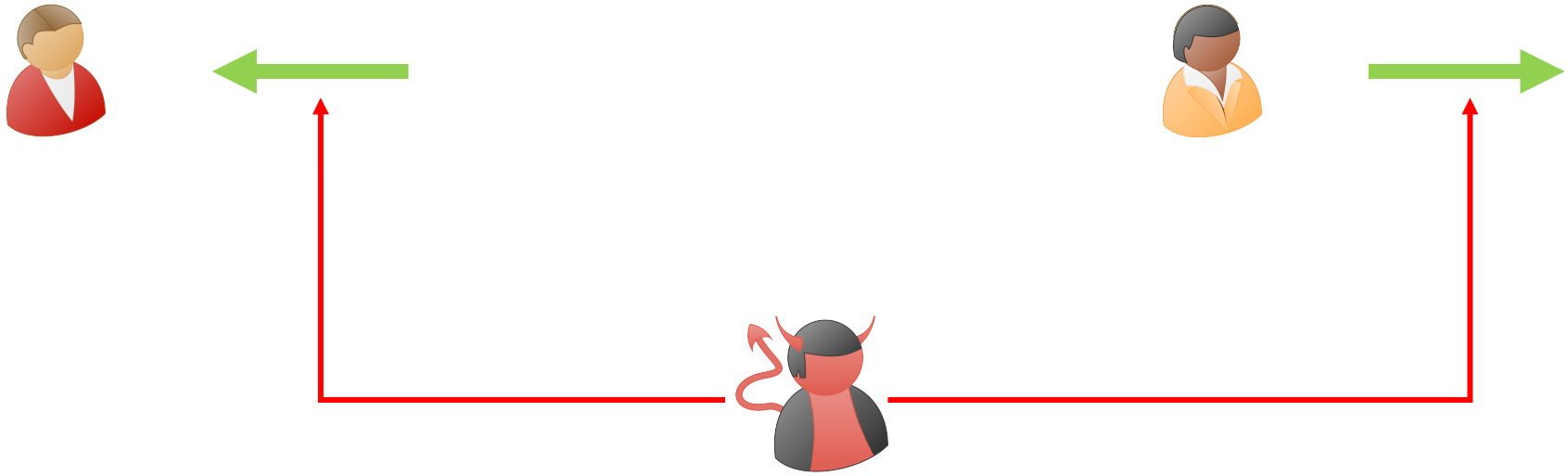


If the returned hash value is $< D_p$, it adds the new block to its own state, and broadcasts its updated state.

It is possible that in any given round no party is able to mine a block. In such a case, the execution just progresses onto the next round.

Δ-Asynchrony

The adversary can delay the messages on the broadcast from being delivered by at most Δ rounds, where Δ is some parameter. This is what allows us to model asynchronous networks.



delay messages by at most Δ rounds

Blockchain Properties

We describe the properties that captures the security of the blockchain.

Parameters

Recall that p is the probability that a randomly chosen nonce will lead to an accepting block.

ρ is the fraction of the adversarial players.

n is the total number of players

Δ is the upper bound on the number of rounds a message can be delayed by the adversary.

$$\alpha = 1 - (1-p)^{(1-\rho)n}$$

α is the probability that some honest party succeeds in mining a block in a given round

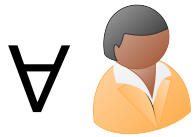
$$\beta = \rho n p$$

β is the expected number of blocks that the adversary can mine in a given round

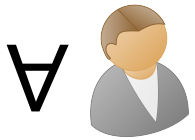
$$\gamma = \frac{\alpha}{1 + \Delta\alpha}$$

γ can be roughly thought of as the discounted value of α due to the delay

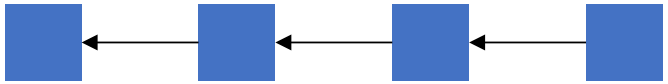
Chain Growth



round r



round $r' \geq r + \Delta$



consistent length

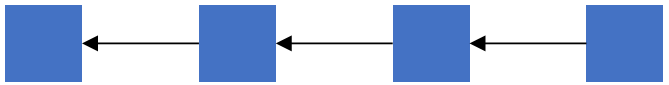
For the consistent length:
For every honest party, if the length of its chain in some round r is k. Then by round $r + \Delta$ every honest party must have a chain of length k.

For chain growth:
We want that every party's chain grows by some number of blocks T.

Note that for this property we're only bothered with the length of the chain, and not the actual content of the chains.

round r

round $r' = r + \mathrm{t}$



maximum chain length

minimum chain length

difference=chain growth $\geq T$

Chain growth theorem

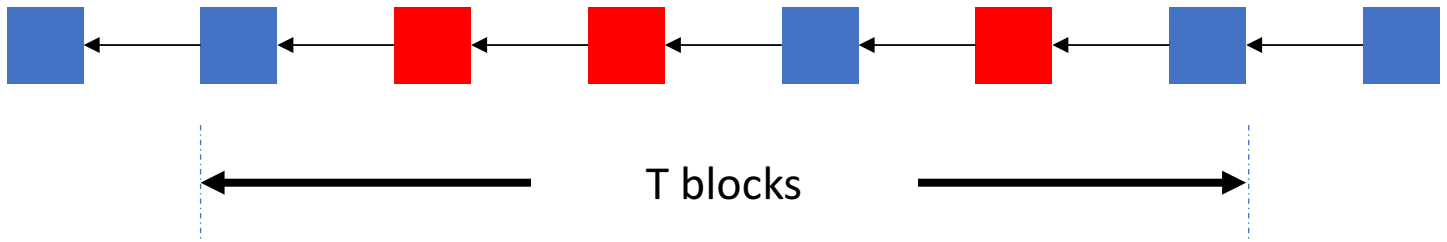
At any point, for any T, the chain grows by at least T in the previous $\frac{T}{(1-\delta)\gamma}$ rounds.

The larger the value of δ, the more likely the theorem is to hold.

Think of δ as a security tuning parameter.

Chain Quality

We consider any T consecutive blocks in the chain.



honest block

adversarial block

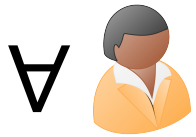
Measure fraction of honest blocks in T consecutive blocks.

Chain quality theorem

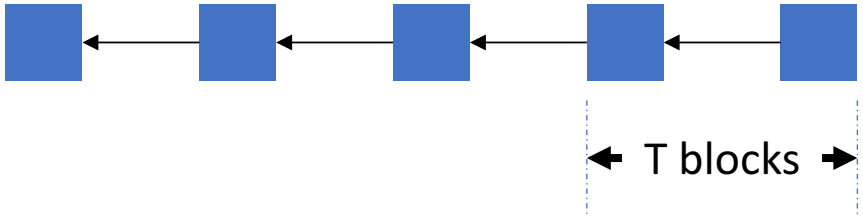
At any point, for any T consecutive blocks, the fraction of honest blocks is at least $1 - (1 + \delta)\frac{\beta}{\gamma}$.

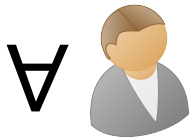
Think of δ as a security tuning parameter.

T-consistency



round r





round $r' \geq r$

||

For any honest party in round r, other than the last T blocks the rest of the chain must be a prefix to every other honest party's chain.

In essence, this ensures that once a block is T blocks deep in a chain, it appears on every party's blockchain.

Chain consistency theorem

If $\alpha(1-2(\Delta+1)\alpha) \geq (1+\delta)\beta$, then the chain satisfies consistency.

Think of δ as a security tuning parameter.

Even though we've talked about a permissionless setting, our parameters seem to depend on the total number of parties n. For this we use only an upper bound on the number of parties. The further the bound is from the actual number of parties, the worse is the performance of the blockchain.

We use a loose upper bound on the number of parties.

The delay Δ cannot be arbitrary, and is restricted by a relation between p, n and Δ.

Interesting question:

Vary p based on the exact number of parties during some period rather than an upper bound?

The Bitcoin Backbone Protocol with Chains of Variable Difficulty (CRYPTO 2017)

Garay, Kiayias, Leonardos

We describe an alternate modelling of our system which will be easier to work with as opposed to the model of the random oracle.

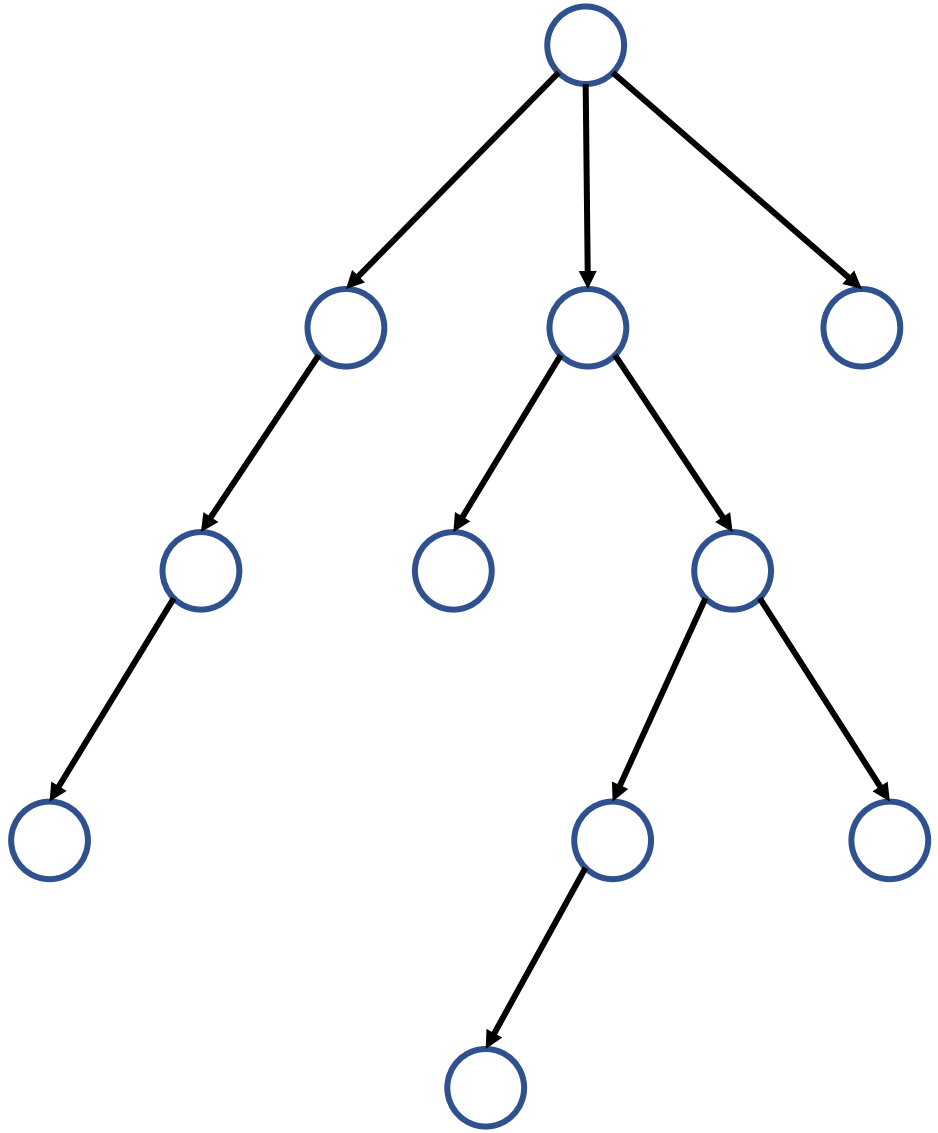
$\mathcal{F}_{tree}$ model

Replace the hash function oracle

$\mathcal{F}_{tree}$ model

Keeps track of all valid chains of records through a tree.

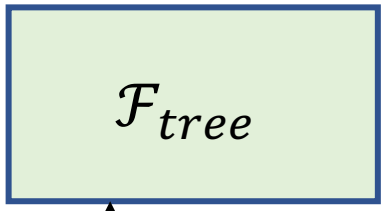
There is global functionality that keeps track of all valid chains via a tree. It initializes the tree with an empty root and grows with queries from the parties.



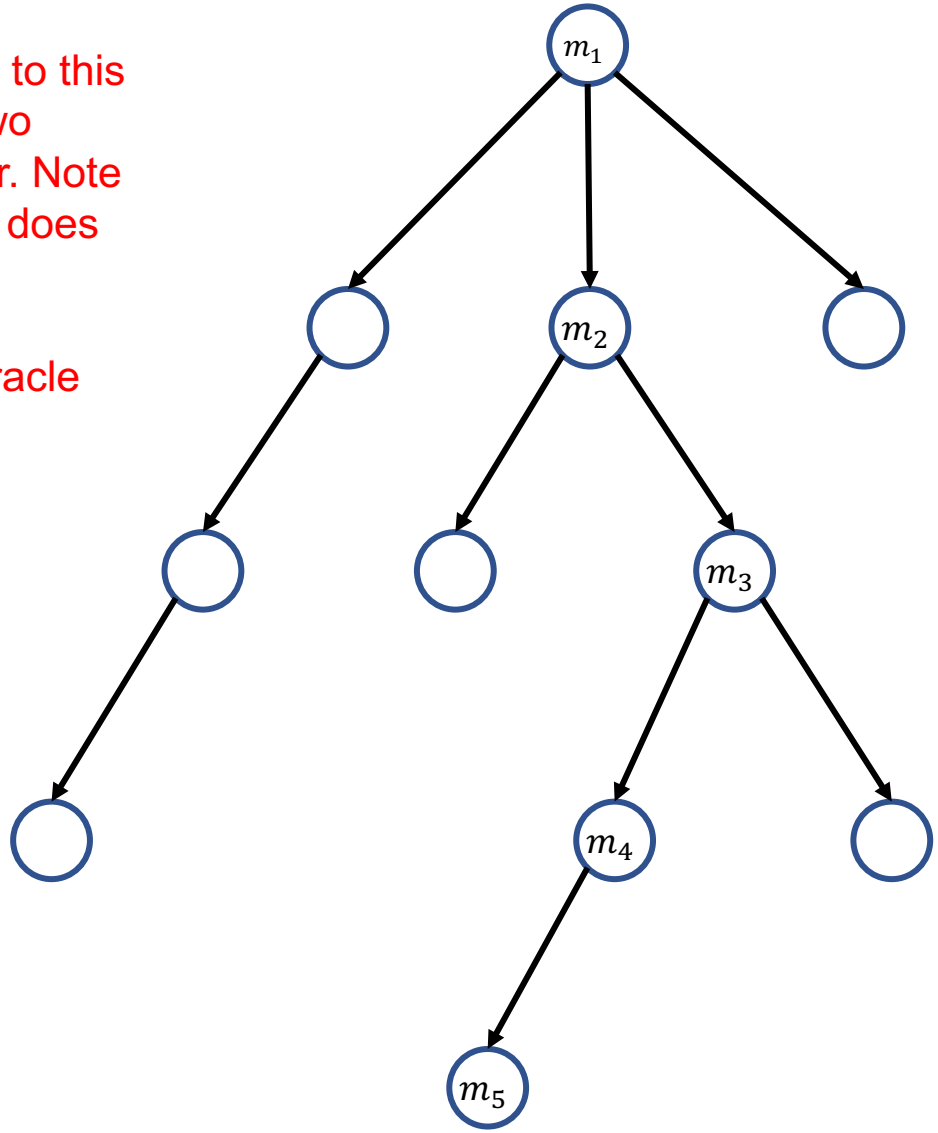
$\mathcal{F}_{tree}$ model

Every party now has access to this single $\mathcal{F}_{tree}$ instead of the two oracles we'd modeled earlier. Note that this is still a model (and does not exist in reality).

The parties can query the oracle with the extend query.



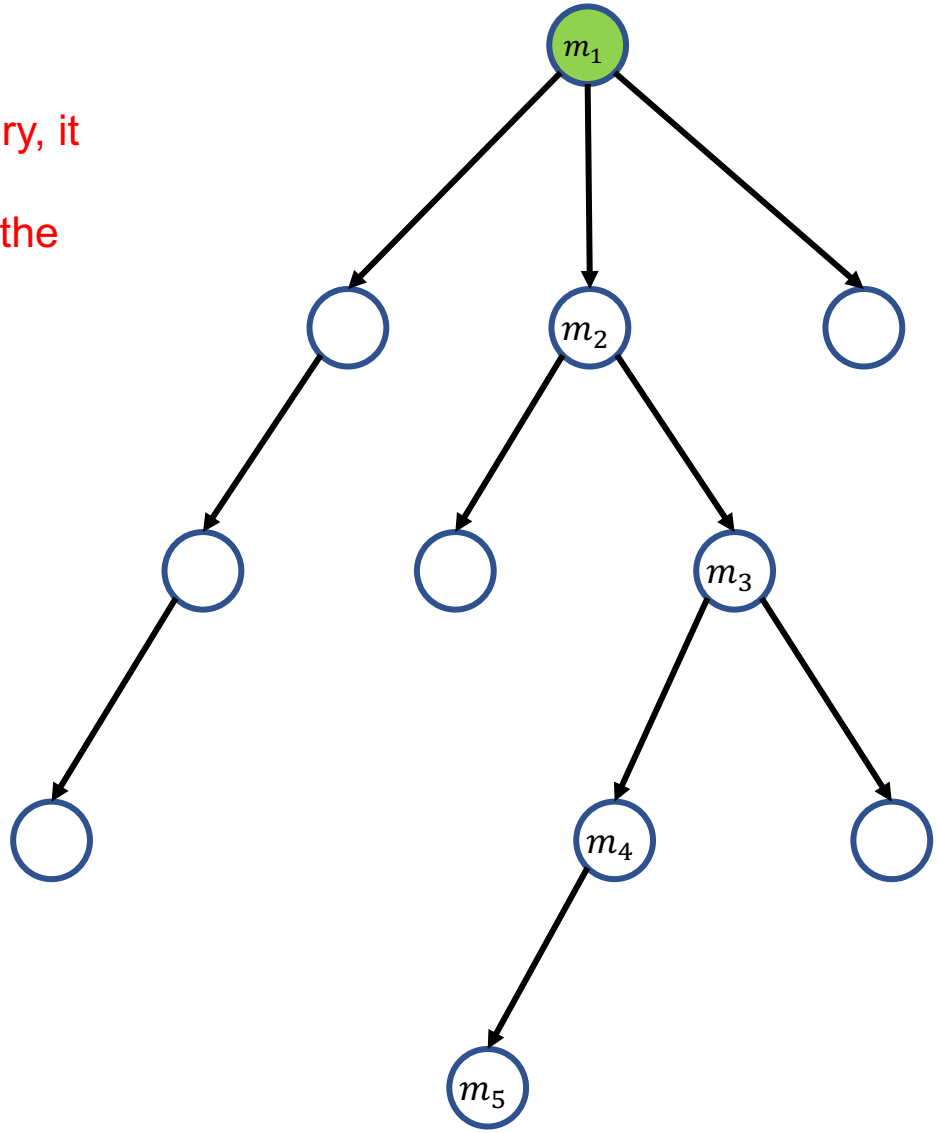
extend $((m_1, \cdots, m_5), m)$



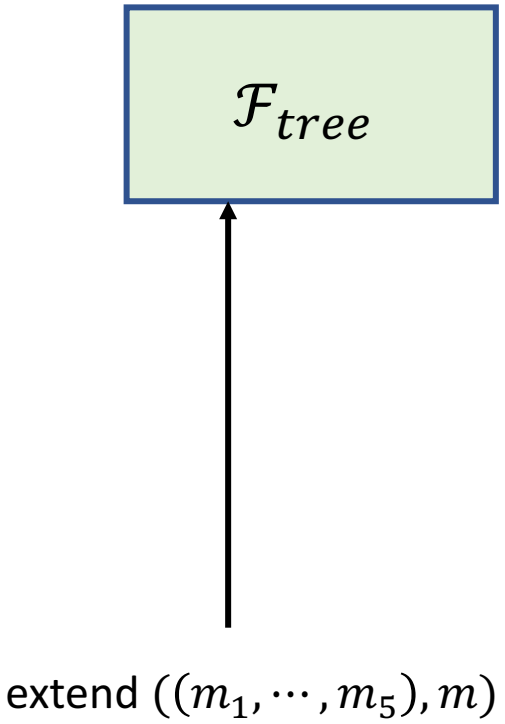
$\mathcal{F}_{tree}$ model

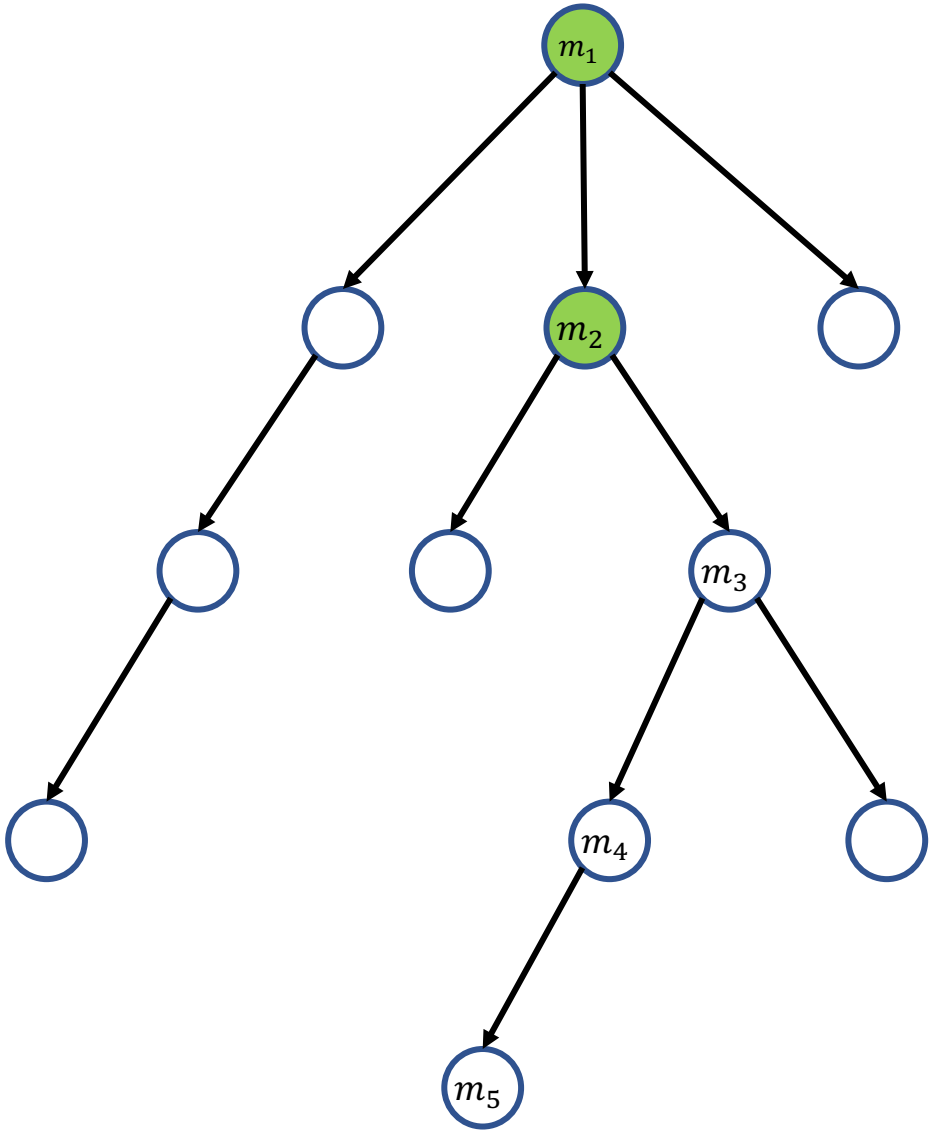
On receiving the extend query, it first checks if the path m_1, m_2, m_3, m_4, m_5 exists on the tree.

extend $((m_1, \cdots, m_5), m)$

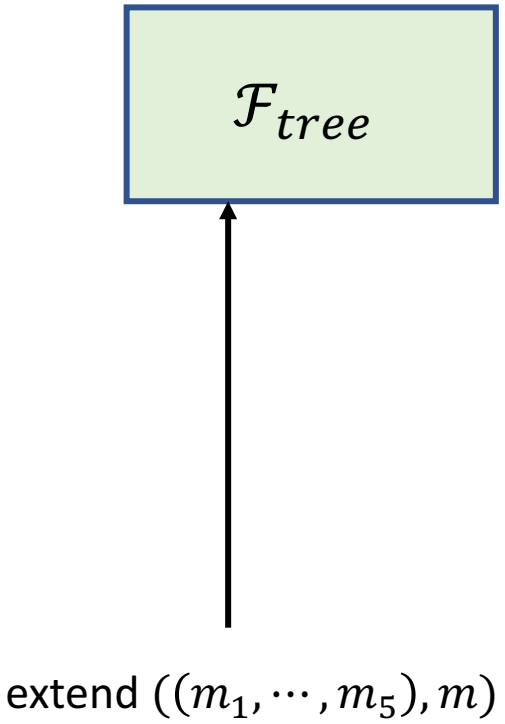


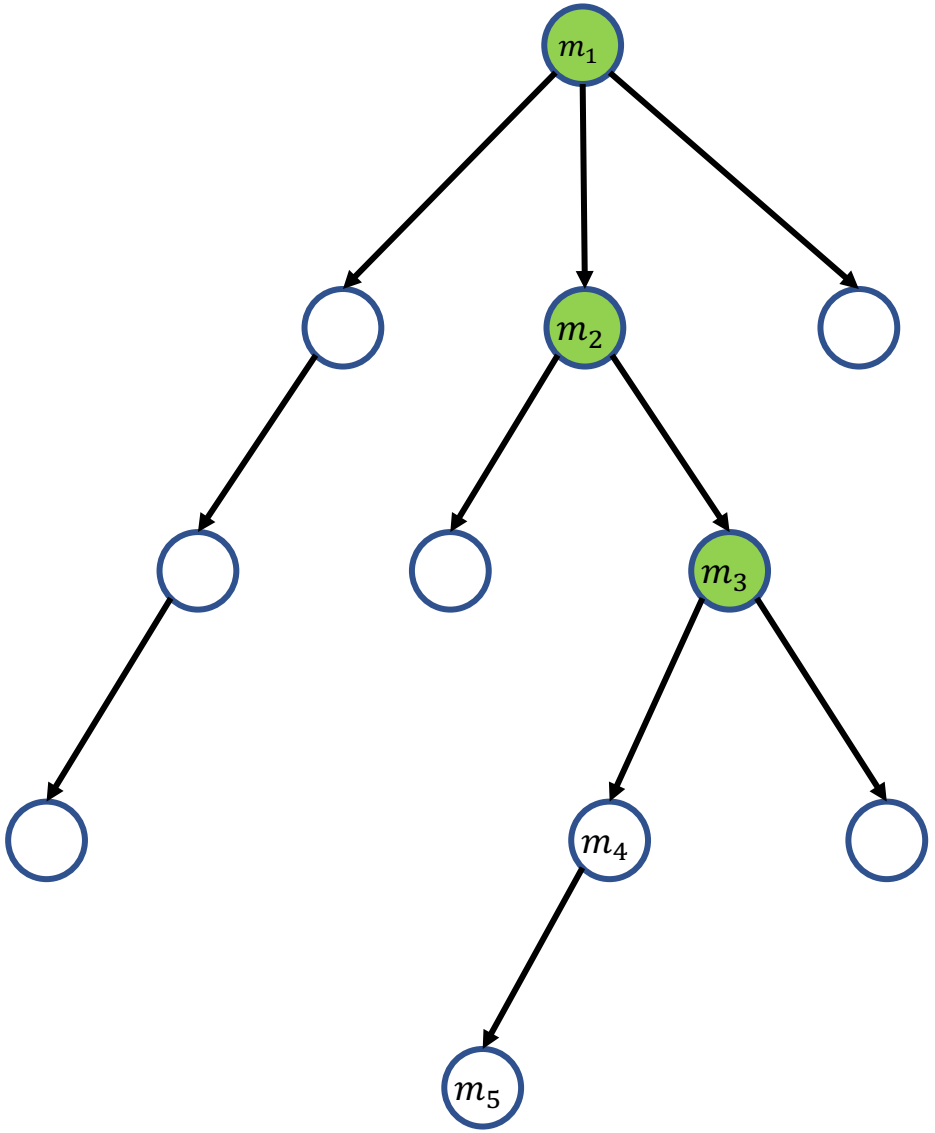
$\mathcal{F}_{tree}$ model



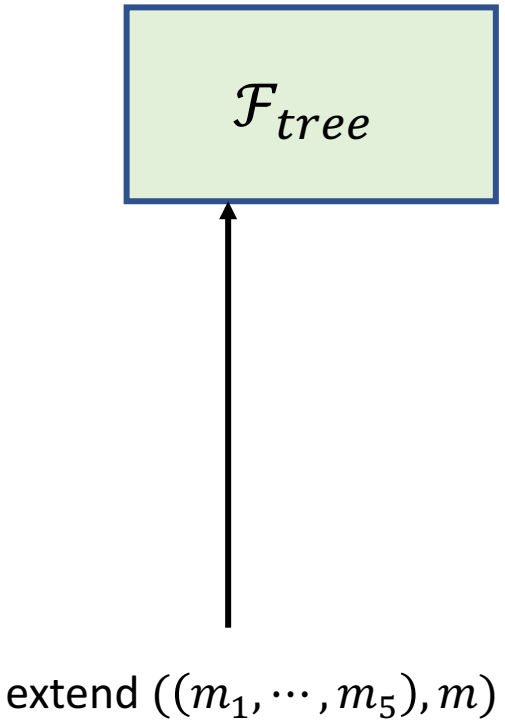


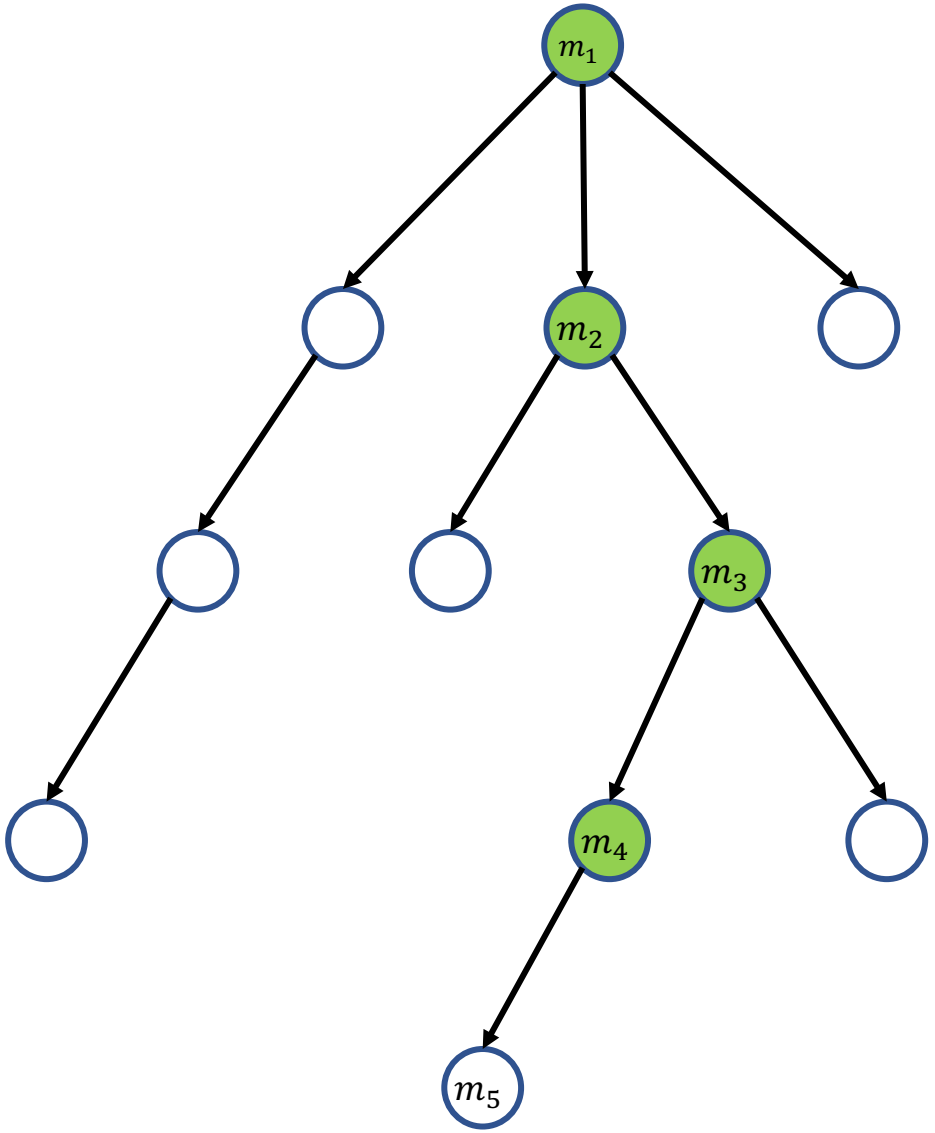
$\mathcal{F}_{tree}$ model



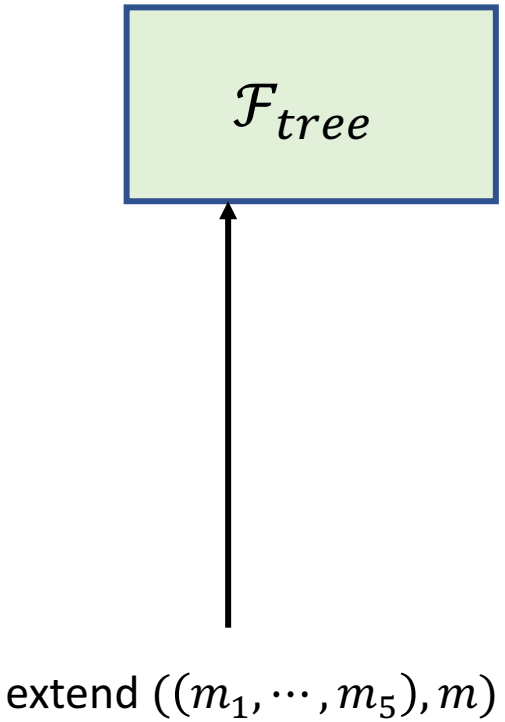


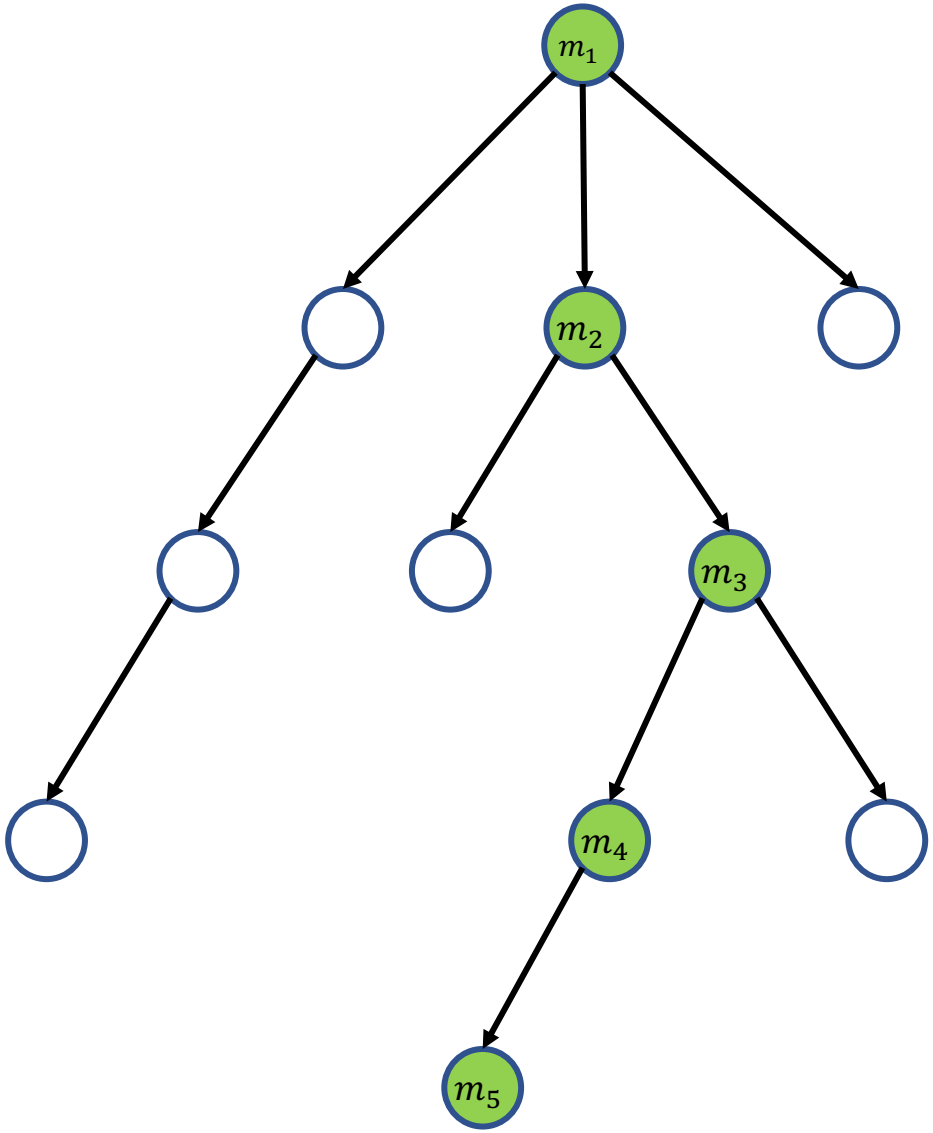
$\mathcal{F}_{tree}$ model





$\mathcal{F}_{tree}$ model

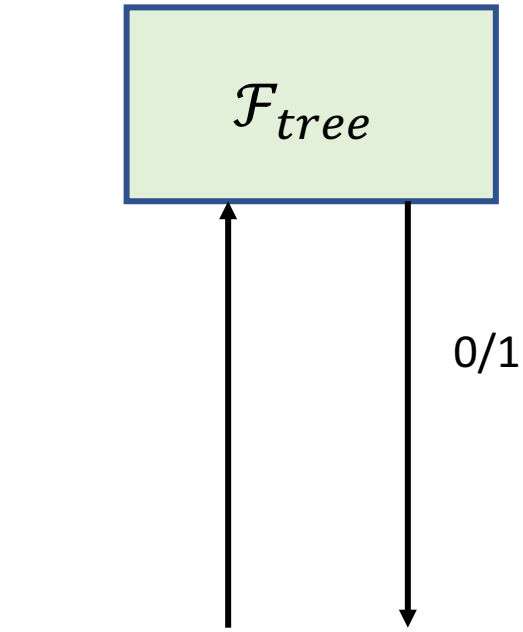


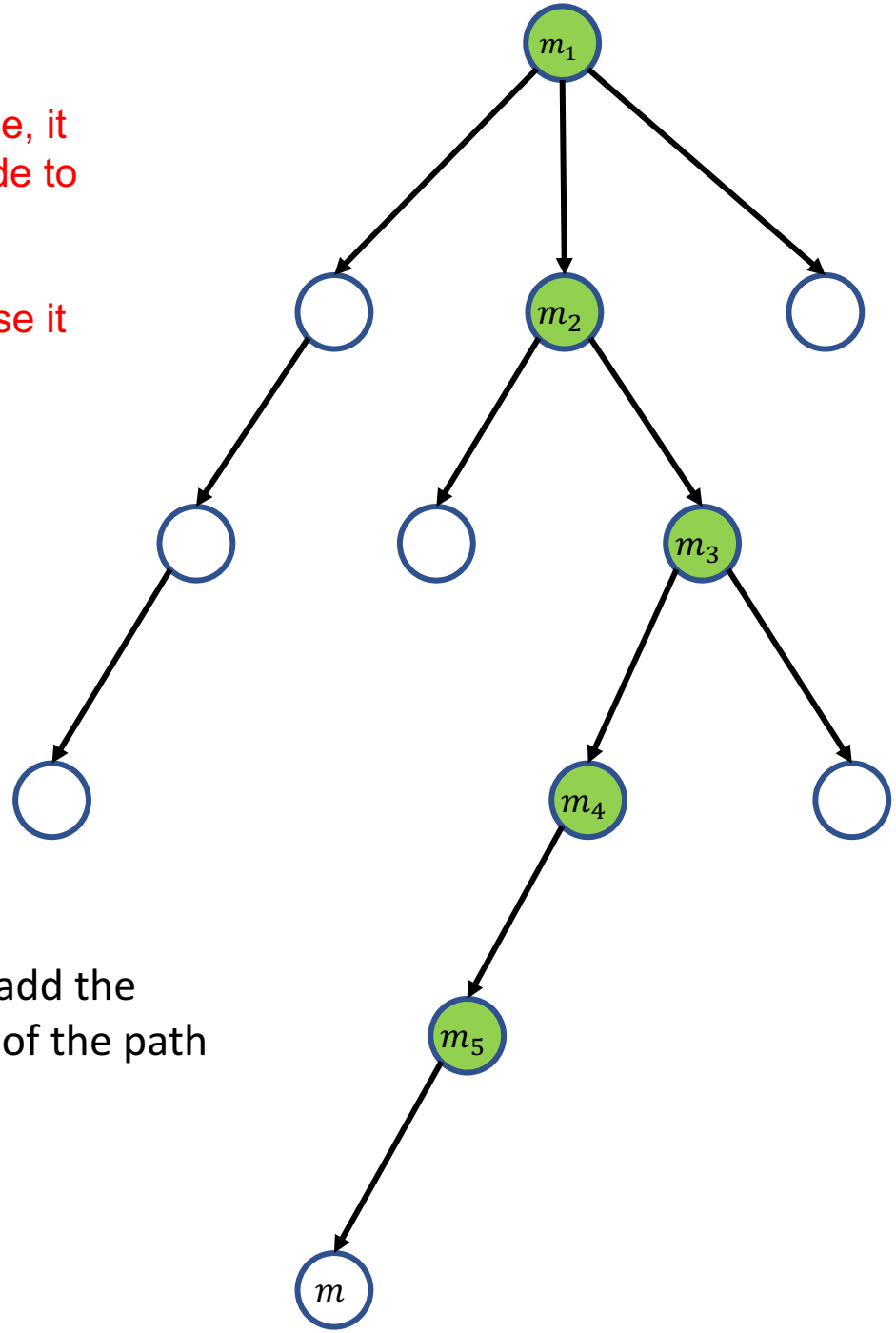


$\mathcal{F}_{tree}$ model

If the path verifies on the tree, it attempts to add the new node to the tree.

If it succeeds it returns 1, else it returns 0.

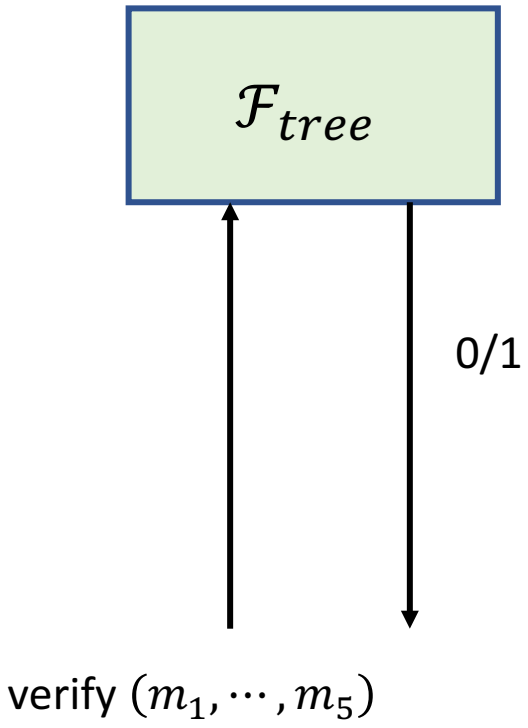


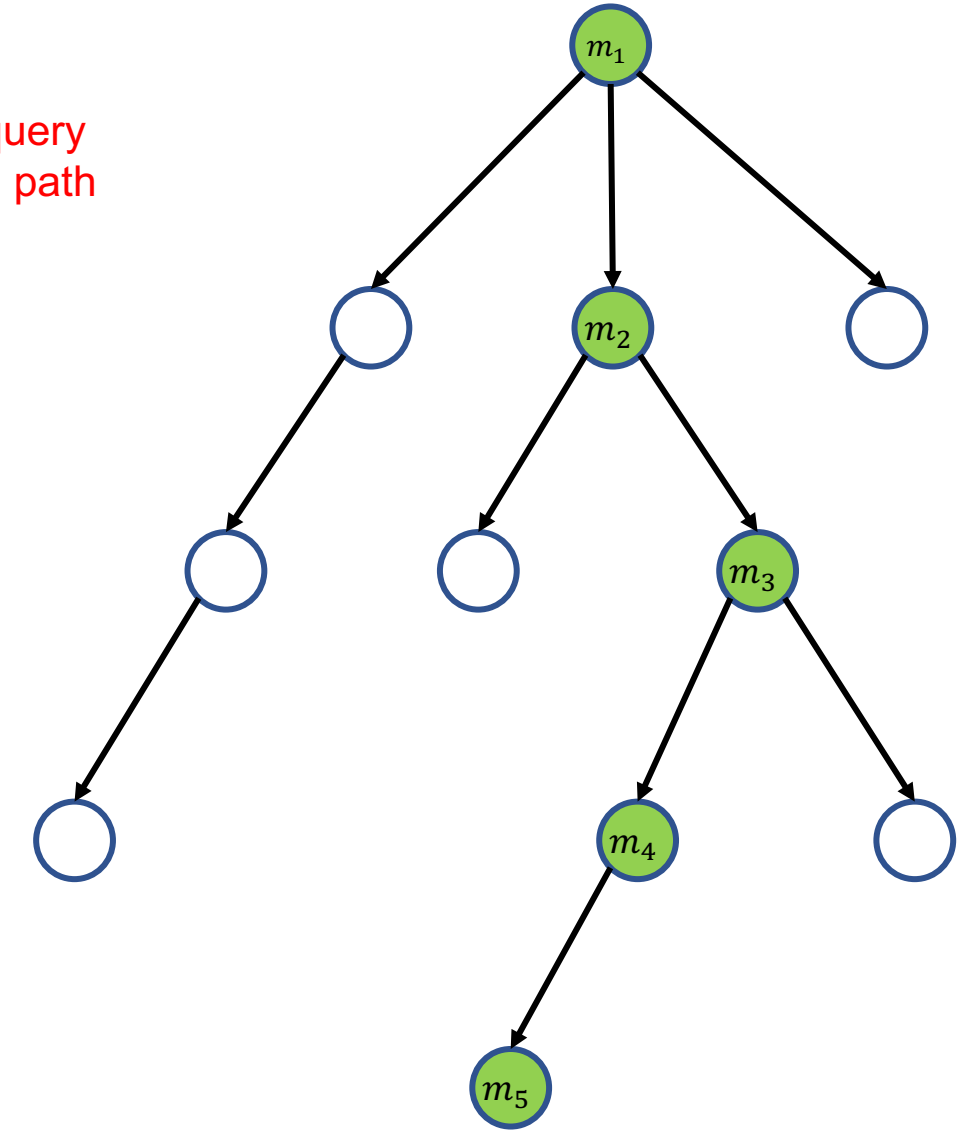


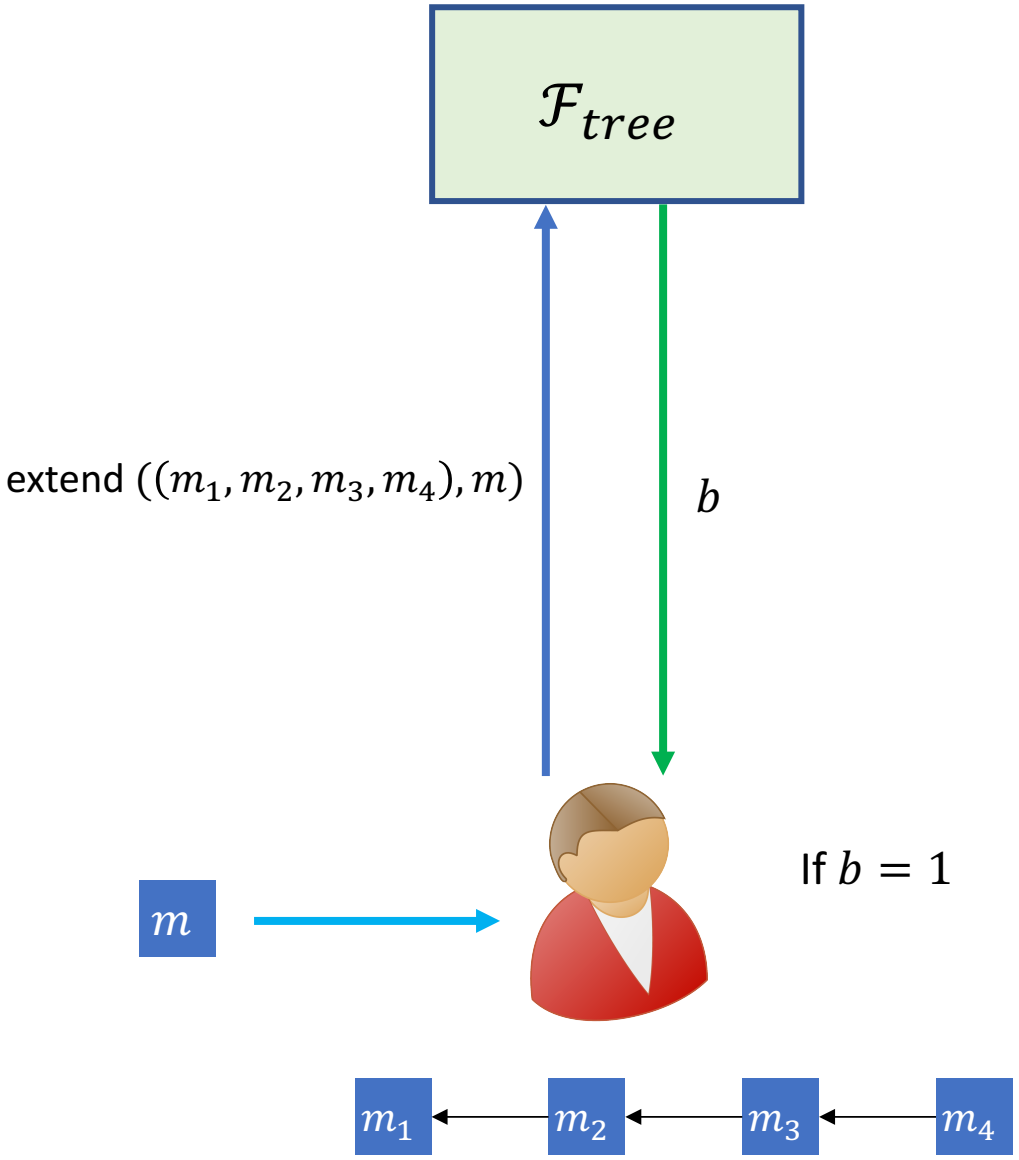
With probability p add the node m at the end of the path

$\mathcal{F}_{tree}$ model

Likewise, there is a verify query that returns 1 if the queried path exists on the tree.







We now replace the oracle computing H with the oracle $\mathcal{F}_{tree}$.

The chain now just consists of records/transactions.

When a player wants to add a new record to the chain, it queries the oracle with its existing chain and the new record.

If the new record is successfully added to the path on the tree, the party broadcasts its chain.

Why $\mathcal{F}_{tree}$ model?

The probability of a miner being successful is independent of the current chain queried.

This is unfortunately not true in the random oracle mode. And we will want to use this fact when we prove the blockchain properties.

We do not know how to prove these properties in the random oracle model.

What if there's some fancy attack in the hash function model which isn't covered in the $\mathcal{F}_{tree}$ model?

Now we can replace the hash function model with our tree model moving forward with the proofs.

Prove that any attack in the hash function model can be translated to an attack in the $\mathcal{F}_{tree}$ model.

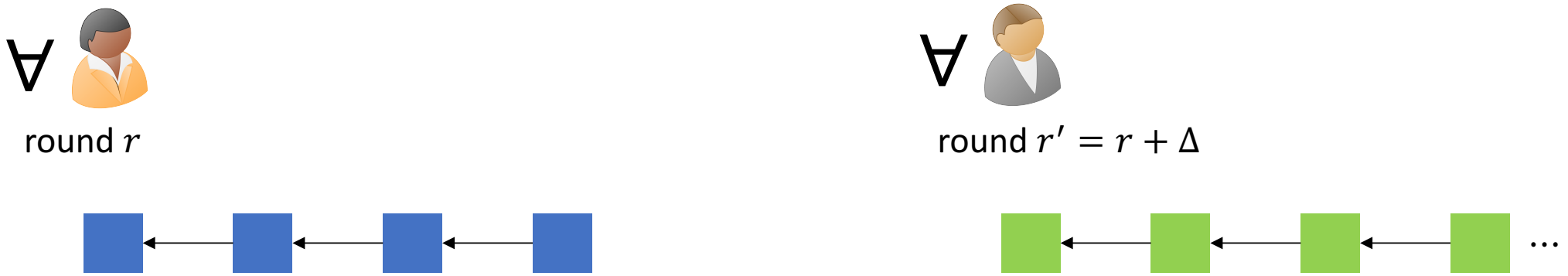
Proof of blockchain properties

Chain growth

Chain quality

Consistency

Proof of chain growth



consistent length

Why?

When an honest party in round r receives information about a chain of length k, then by round $r + \Delta$ every other party must have received the same information.

Of course the parties may discard the chain if they already have a bigger chain, but that means that by round $r + \Delta$ every honest party has a chain of length at least k

Hybrid Freezing Model

Hyb_r

After round r:

1. Remove all new messages of the adversary.
   Undelivered messages delayed by Δ.
2. Delay all messages sent by the honest parties by Δ
3. Whenever messages being delayed, all honest parties freeze.

We want to simplify our model further to analyze the current property. To do so we define the model Hyb_r where we remove all adversarial messages and delay the messages sent by the honest parties to their maximum of Δ rounds. In addition, the honest parties do not mine (freeze) when their messages are being delayed.

We prove a relation between the length of the chain of honest parties in both models.

$\forall t, \forall$  $Len_{REAL}^{r+t} \geq$  $Len_{Hyb_r}^{r+t}$

Informal: Once I switch to the hybrid, the chain in the real execution grows at least as fast as in the hybrid execution.

Proof by induction:

if $t = 0$: trivial

Assume true for all $t' < t$: Chain at $r + t - 1$ in $REAL$ at least as long as in Hyb_r

Hyb_r

- Succeeds in mining in round $r + t$
- Receives chain from adversary
- Honest player j succeeds in mining at an earlier round.

$REAL$

- Succeeds in mining in round $r + t$
- Would have received the same chain earlier
- Player j's chain would have been received earlier.

Lemma:

$\forall t$

$$Longest^{r+t}_{Hyb_r} > \quad Longest^{r}_{Hyb_r} + (1-\delta)\gamma t$$

other than with negligible probability in t

Proof:

For every round that honest parties are not frozen:

they have received all sent messages ⇒ all honest parties have chains of the same length

The longest chain is extended with probability α.

We now look at the longest chain held by an honest party in this hybrid freezing model.

This is similar to a Markov process where the chain extends in each non-frozen round with probability α, independent of how it grew in the past.

The proof relies on the use of the Chernoff bound.

Lemma:

$\forall t$

$$Longest^{r+t}_{REAL} > \quad Longest^{r}_{REAL} + (1-\delta)\gamma t$$

other than with negligible probability in t

Proof:

$\forall t$

$$Longest^{r+t}_{Hyb_r} > \quad Longest^{r}_{Hyb_r} + (1-\delta)\gamma t$$

We use the two previous lemmas to prove a lower bound on the growth of the longest chain in the real experiment.

Lemma:

$\forall t$

$$Longest_{REAL}^{r+t} > \quad Longest_{REAL}^{r} + (1-\delta)\gamma t$$

other than with negligible probability in t

Proof:

$\forall t$

$$Longest_{Hyb_r}^{r+t} > \quad Longest_{REAL}^{r} + (1-\delta)\gamma t$$

Lemma:

$\forall t$

$$Longest_{REAL}^{r+t} > \quad Longest_{REAL}^{r} + (1-\delta)\gamma t$$

other than with negligible probability in t

Proof:

$\forall t$

$$Longest_{REAL}^{r+t} > \quad Longest_{REAL}^{r} + (1-\delta)\gamma t$$

But we've only proved a lower bound for the longest chain.

Easy extension to show that for any chain, in t rounds it grows by at least $(1-\delta)\gamma t$.

Or alternatively, grows by at least T in $\frac{T}{(1-\delta)\gamma}$ rounds.

other than with probability negligible in T.

Proof of blockchain properties

Chain growth

Chain quality

Consistency

Lemma:

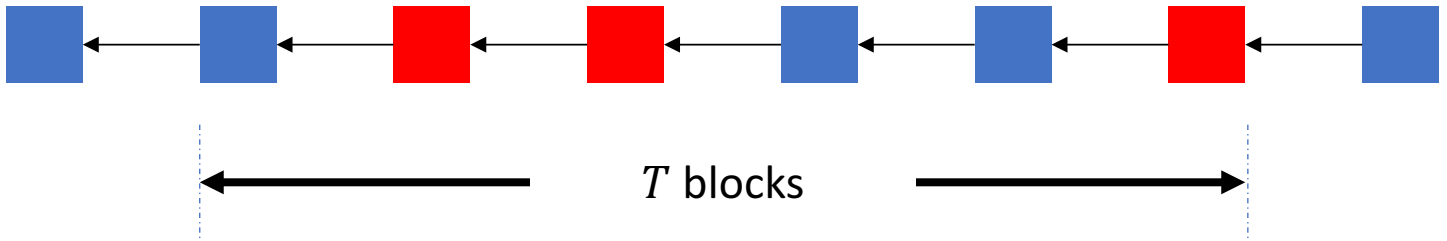
# of adversarial blocks in any window of t rounds $< (1 + \delta)\beta t$

other than with probability negligible in βt

This follows from an easy application of the Chernoff bound.

It states that within a window of t rounds, the adversary can mine at most a fixed number of blocks.

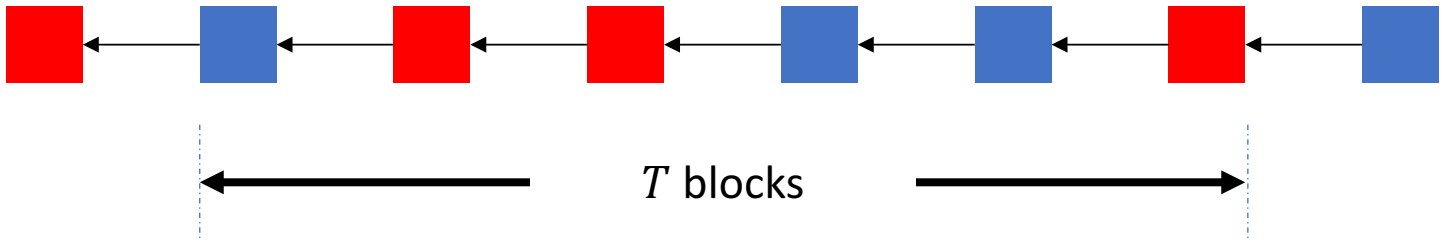
Consider the chain of any 



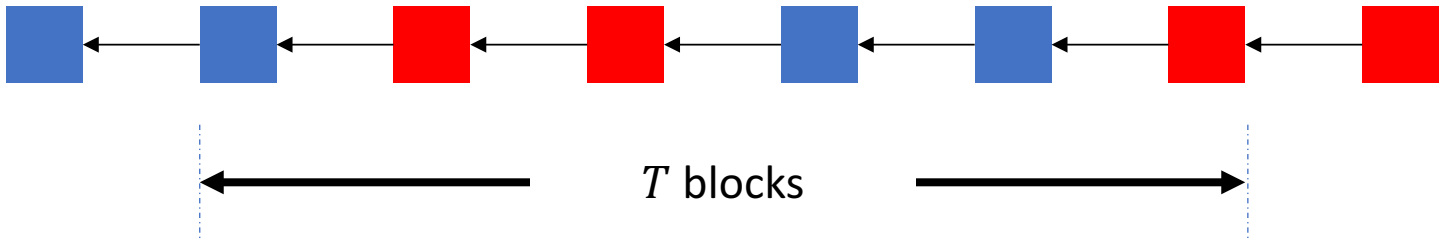
Can the two neighbors to the window be adversarial?

Consider the chain of any honest party. We now look at any sequence of T blocks on this chain. Can the block just before or after the window be adversarial?

Consider the chain of any 

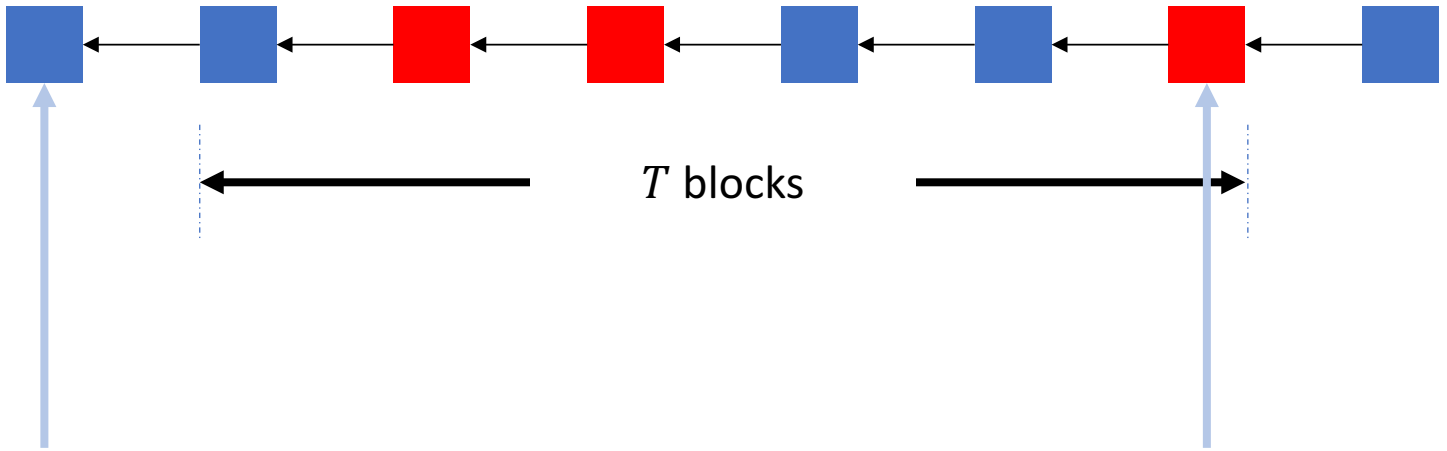


Consider the chain of any 



Since we're trying to establish a lower bound on the fraction of adversarial blocks in this sequence of T blocks, if either of the neighboring blocks was adversarial, we could have included them by extending T. So let us focus on the case that the sequence is surrounded by honest blocks on either side.

Consider the chain of any 



We know that if a block is accepted by the honest players, it must have mined prior to the round $r + t$

From the construction of $\mathcal{F}_{tree}$, all T blocks must have been mined between rounds r and $r + t$.

Essentially the proof relies on the fact that there is an upper bound on the time required for the chain to grow by T blocks. In the same time window, we have shown an upper bound on the number of adversarial blocks. Combining the two we arrive at the final result.

From chain growth theorem,

$$t \leq \frac{T}{(1-\delta')\gamma}$$

Upper bound on # of adversarial blocks in any window of size t

$$(1+\delta'')\beta t \leq \frac{(1+\delta'')}{(1-\delta')} T \frac{\beta}{\gamma}$$

Fraction of adversarial blocks for suitable choice of δ', δ''

$$(1+\delta)\frac{\beta}{\gamma}$$

Care must be taken when dealing with negligible probabilities.

Proof of blockchain properties

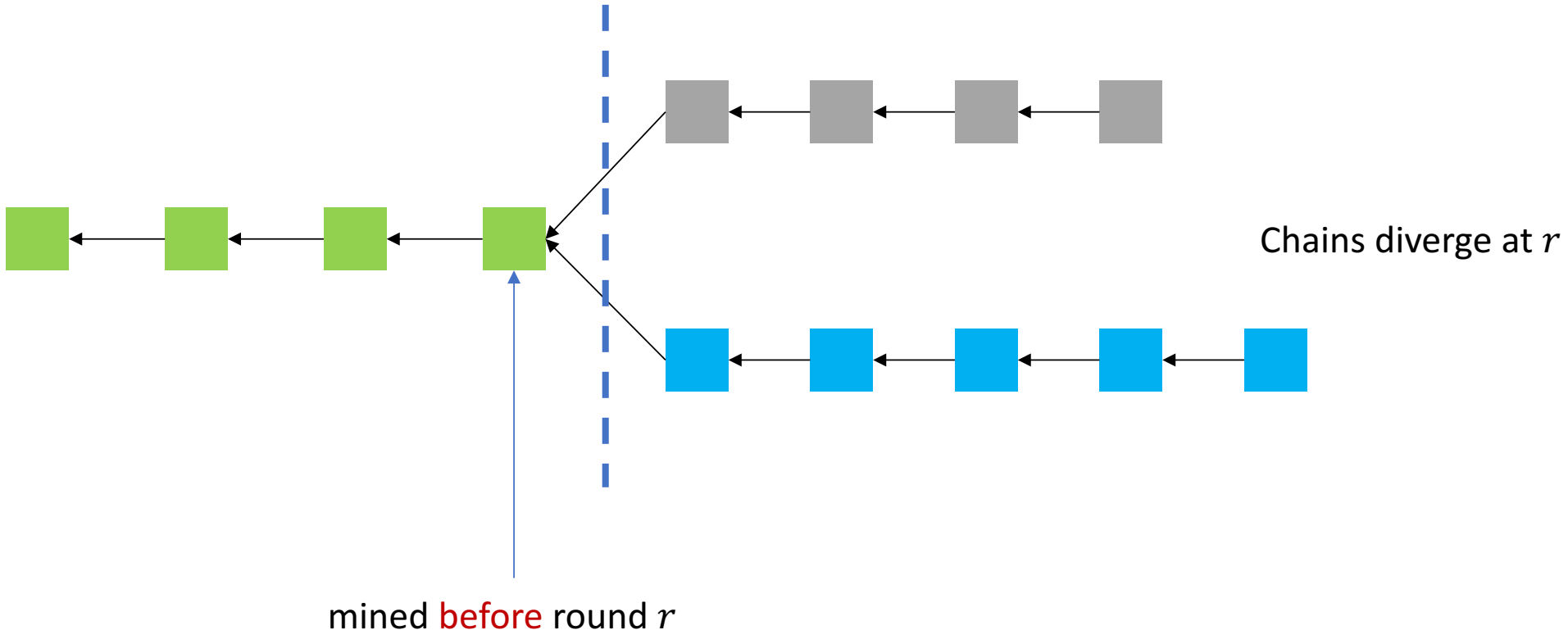
Chain growth

Chain quality

Consistency

Divergence

We say that two chains diverge at round r if the last common block of the two chains was mined prior to round r



Cannot find

round r

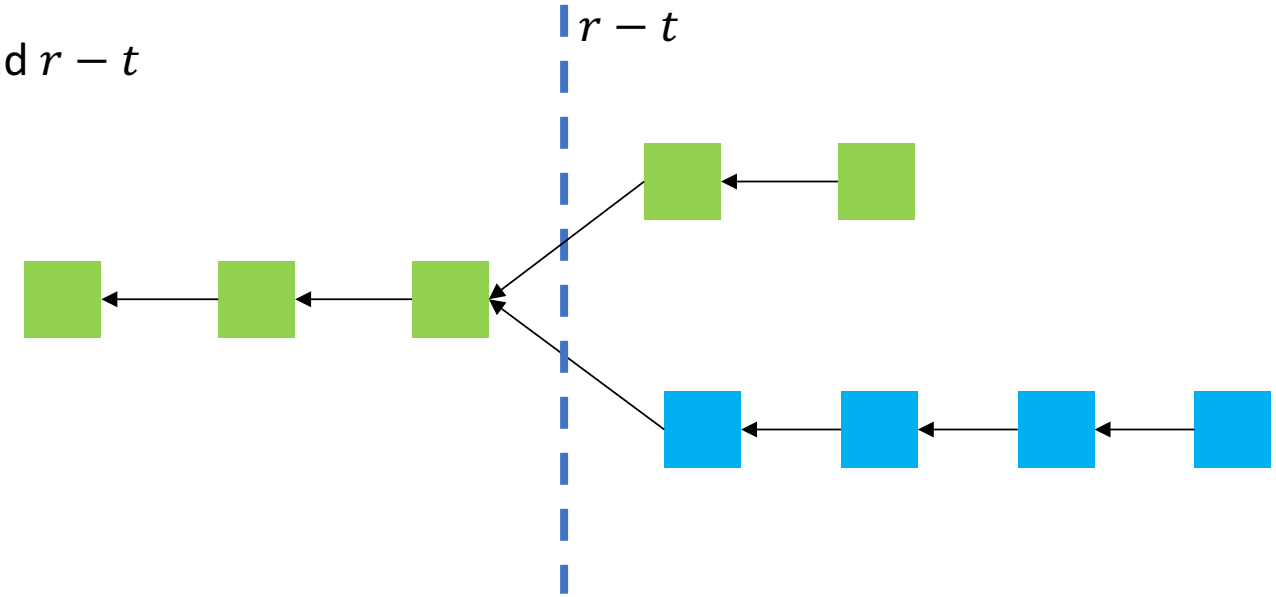
round $r' \geq r$

Consider the chain at round $r - t$

Diverge at around $r - t$

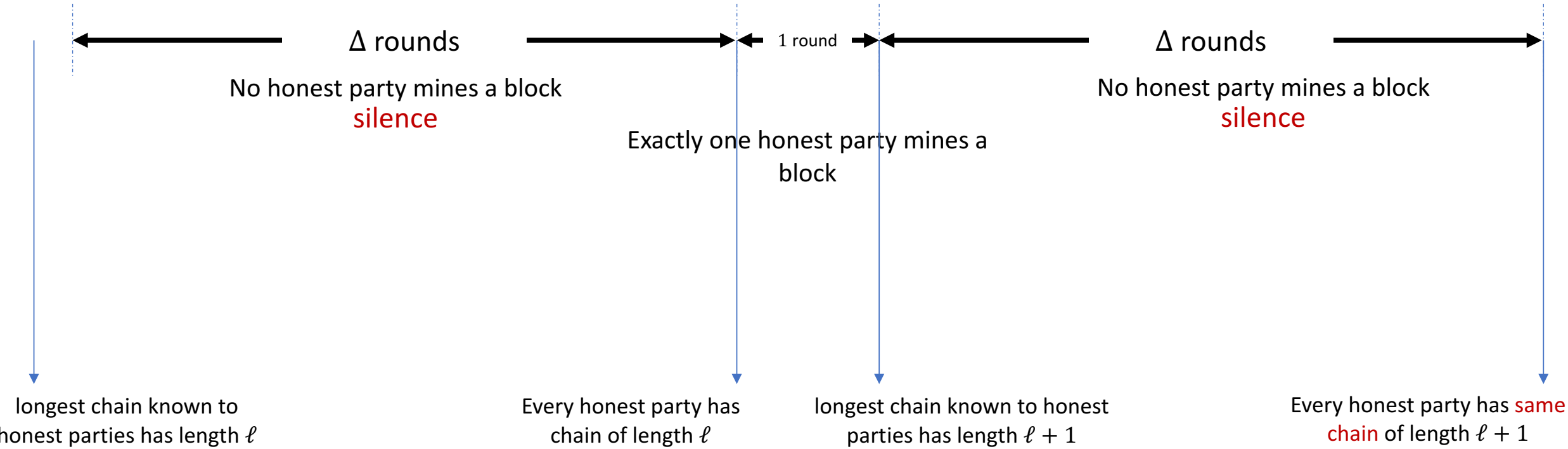
The crux of the proof relies on the lemma which says that the further back in time (rounds) that you go, the hard it will be to find two chains that diverge.

If you increase t, it becomes harder to find chains that have diverged at round $r - t$



other than with probability negligible over t

Convergence Opportunity



# of convergence opportunities in a t round interval is at least:

$$(1-\delta)\big(1-2\alpha(\Delta+1)\big)\alpha t$$

For the proof, we define a window of $2\Delta + 1$ rounds called a converge opportunity.
There is a single round where exactly 1 honest party mines which is sandwiched between two windows of Δ rounds where no honest party mines a block.
If there were no adversarial parties mining a block, this window allows for every honest party to converge on the same chain.

If in a t round interval, there is even a single convergence opportunity, the honest parties will have the same chain at this point and we'd be done with the proof.

The remained of the proof shows that no adversary can succeed in disrupting each such convergence opportunity.

The only way that the adversary attacks is by mining $l + 1$ blocks at each such opportunity.

It doesn't have to extend the longest chain, it could have mined even before this period started. (selfish mining)

can't withhold blocks for "too long"

Even with this additional period, the adversary mines fewer than $(1 - \delta)\big(1 - 2\alpha(\Delta + 1)\big)\alpha t$ blocks.

There is at least one convergence opportunity in t rounds

Recall (From last class)

Chain Growth

Chains grow at a steady rate

Chain Quality

Any k consecutive blocks contains “sufficiently many” honest blocks.

Consistency

Honest nodes agree on all but last k blocks.

Application: Public Ledger

Requirements?

Public

Immutable

Ordered

Consistent view

Property 1: Liveness

If an honest player wants to add a message, it should appear on the (local) ledger of every honest player after some “wait time”.

state

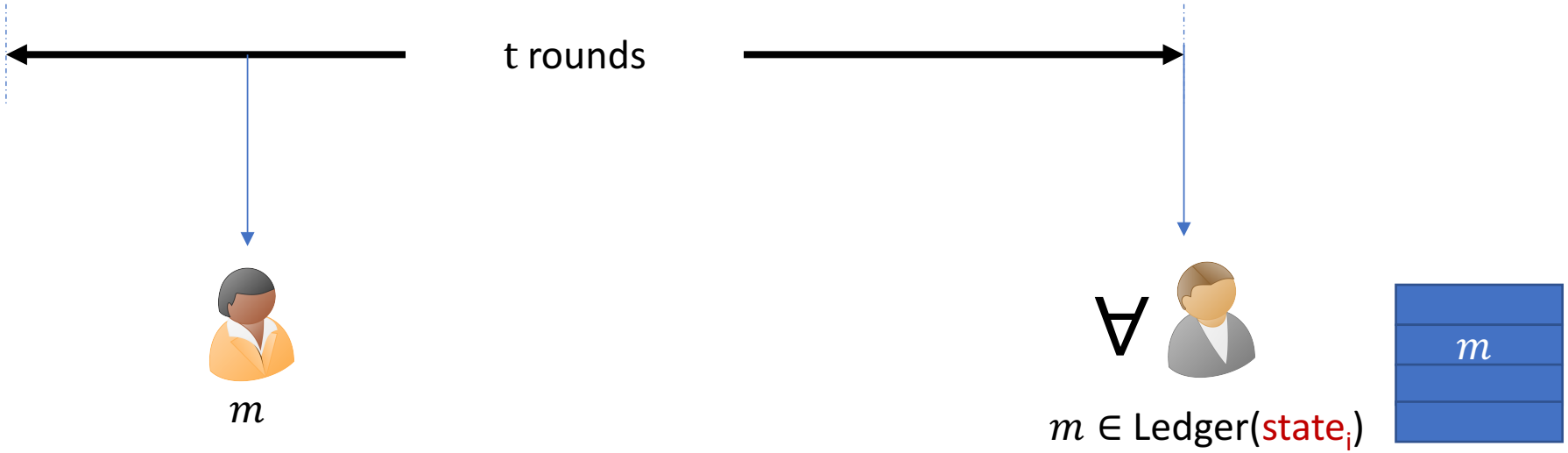
Property 1: Liveness

If an honest player wants to add a message, it should appear on the (local) ledger of every honest player after some “wait time”.

Ledger(state)

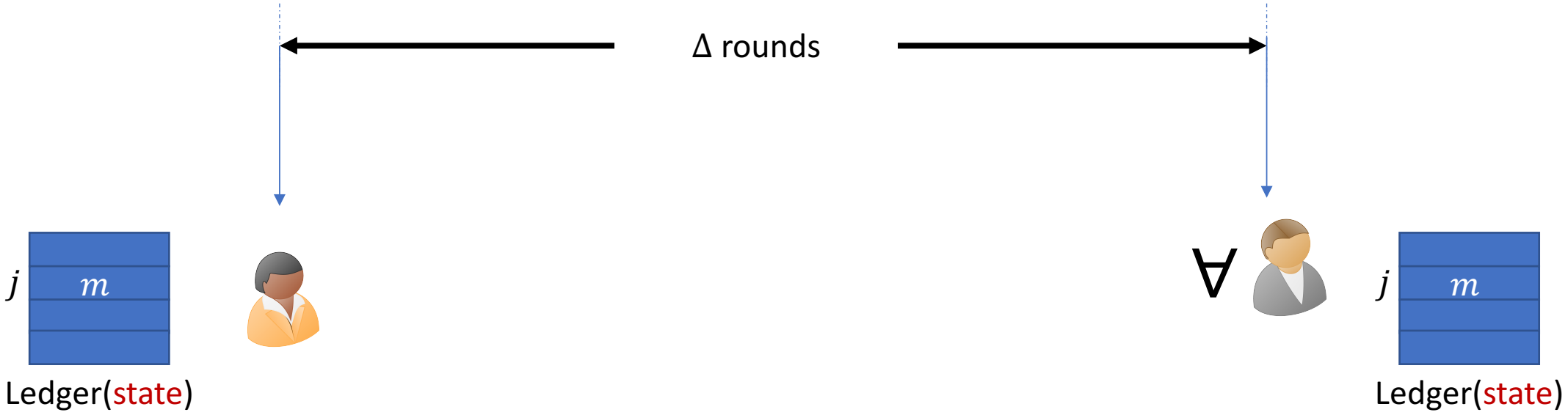
Property 1: Liveness

If an honest player wants to add a message, it should appear on the (local) ledger of every honest player after some “wait time”.



Property 2: Persistence

If a message *m* gets added to the (local) ledger of some honest player at position *j*, then it will appear at position *j* in the (local) ledger of every other honest party, at all future times (after some small delay).



Public ledger from T-consistent Blockchain

At any point, a party's local ledger is determined by truncating its current view of the blockchain by T blocks

- Liveness: Follows from Chain-growth and Chain-quality
- Persistence: Follows from Consistency and Chain-growth (consistent lengths) properties