CS5112: Algorithms and Data Structures for Applications

Guest lecture by Gengmo Qi

31 March 2021

Slides adopted from a variety of sources(see references)

Cornell University

This lecture

- 1. Classical Consensus Algorithms

- 2. Hash pointers and data structures

- 3. Nakamoto Consensus: Proof-of-work

Recall Paxos

- Consensus on <u>one</u> value
  - Repeatedly: multi-Paxos
- Permissioned
  - Membership management
- Propose-Vote paradigm
- Key argument:
  - Majority of accepts means consensus has been reached
- Failure mode
  - Handles fail-stops well
  - What if ID=∞?   -> Byzantine fault
- Tradeoff
  - Never produces inconsistent result, but can (rarely) get stuck

Failure modes

Simple fault | **Byzantine fault**

Crash / Fail-stop | Arbitrary failures

Putting Paxos into context

Simple fault

Byzantine fault

Crash / Fail-stop

Arbitrary failures

Paxos

Classical Consensus

- Foundational theory: State Machine Replication
- Permissioned
- Solutions to the Byzantine Generals Problem:
  - 80s: Early solutions by Leslie Lamport
  - 90s-00s: PBFT provide high-performance solutions

Switching gears

Hash functions

Hash functions:

- Takes any string as input
- Map to fixed-size output(e.g. 256 bits)
- Deterministic

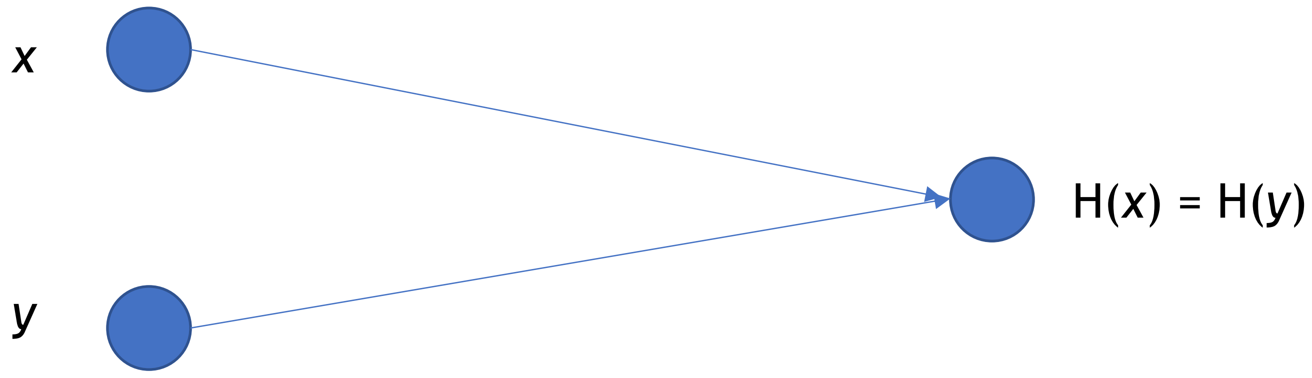
Cryptographic Hash functions:

- Collision-resistant
- Hiding
- Puzzle-friendly

Security Property 1: Collision-resistant

- It is hard to find x and y such that

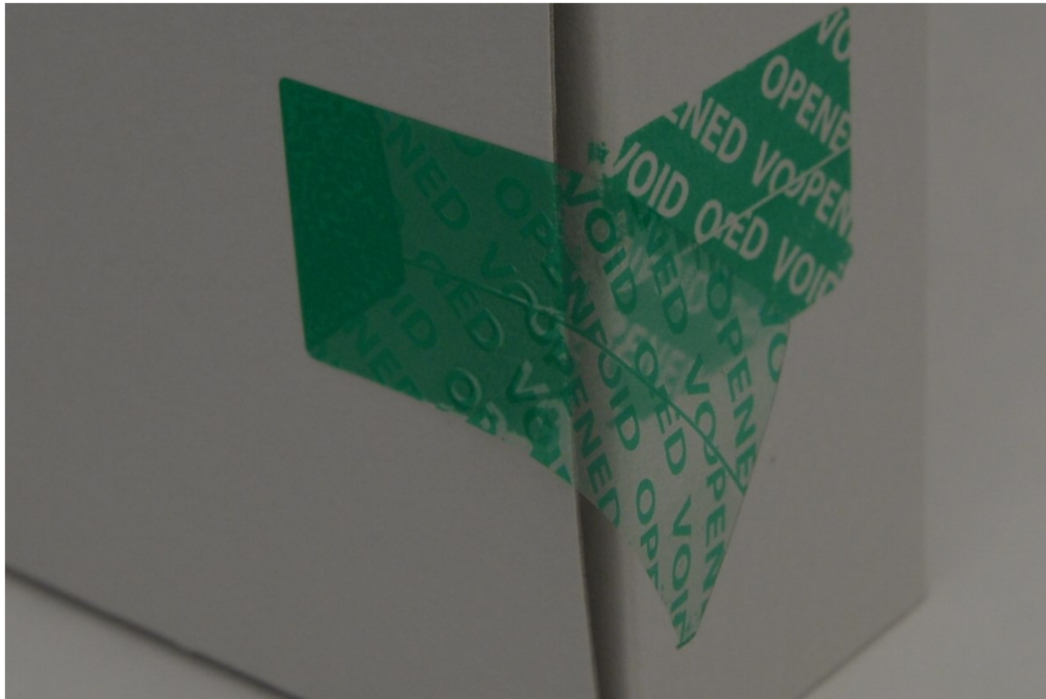
  x != y and H(x) = H(y)



Application: Hash as message digest

- If we know H(x) = H(y)
  - Then it's safe to assume that x=y

  - Application: file integrity / comparison
  - E.g. checksum



What does “hard to find” mean?

- Major topic, center of computational complexity
- Loosely speaking, we can’t absolutely prove this
- But we can show that if we could solve one problem, we could solve another problem that is widely believed to be hard
  - Because lots of people have tried to solve it and failed!
- This proves that one problem is at least as hard as another
  - “reduction”

Security Property 2: Hiding

- Given H(x), it is infeasible to find x
  - i.e. one-way

H("heads")

H("tails")

Security Property 2: Hiding

- Given H(x), it is infeasible to find x
  - i.e. one-way

H("heads")

H("tails")

easy to find *x*!

Why?

Security Property 2: Hiding

If *r* is chosen from a probability distribution that has *high min-entropy*, then given H(*r* | *x*), it is infeasible to find *x*.

H(r | "heads")

H(r | "tails")

Security Property 3: Puzzle-friendly

- Intuition: If you want to target a Hash function *H* to have a particular output value *y*, and if part of the input (i.e., *r*) is chosen in a suitably randomized fashion, then its very difficult to find the other part of the input *x* to exactly hit the target output value (*y*)

- Difficult: no strategy is better than just trying random values of x (brute-force)

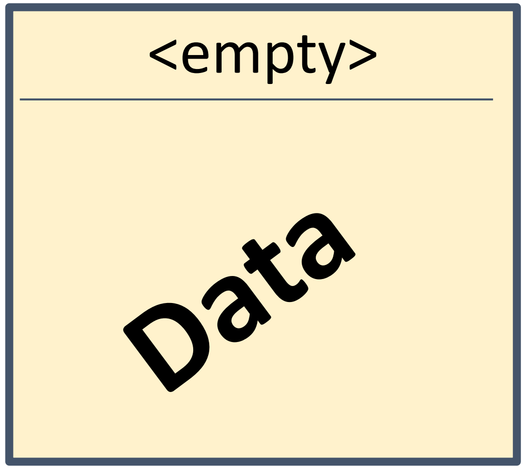
Hash Pointers and Data Structures

- Hash pointer:
  - A pointer to where the data is stored, and
  - Cryptographic hash of the data

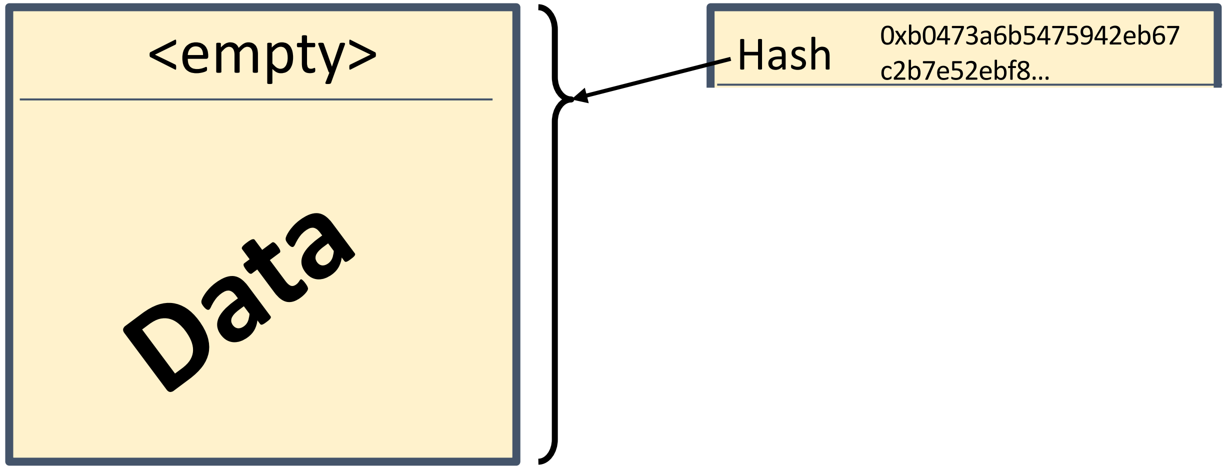
With a hash pointer, we can

- ask to retrieve the data, and
- *verify that the data hasn't been tampered with

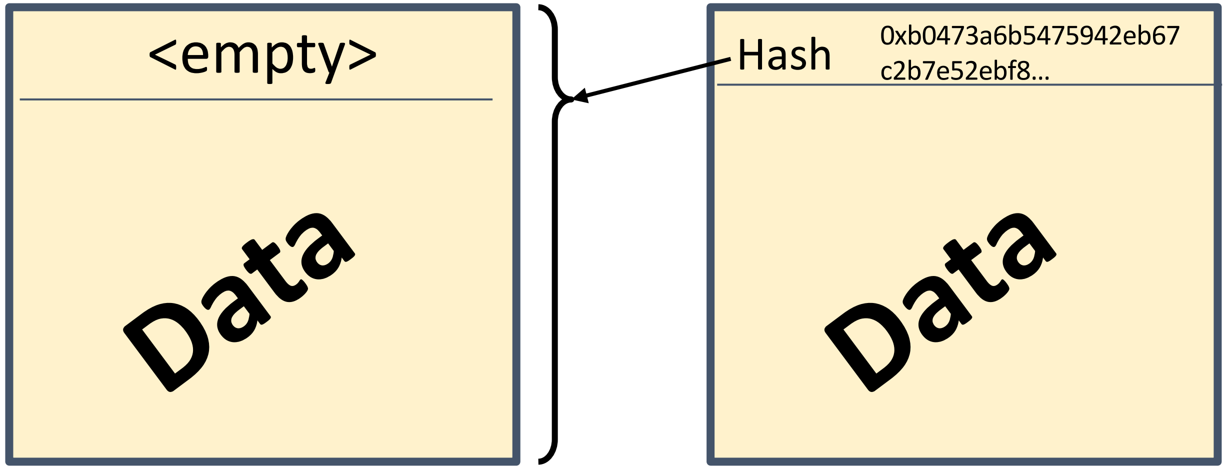
Application: Hash chaining



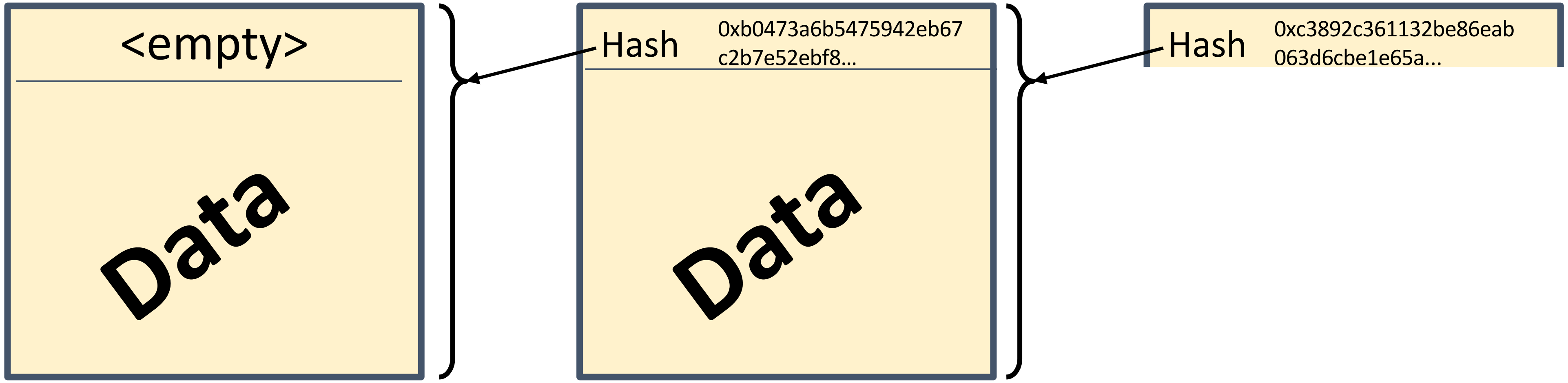
Application: Hash chaining



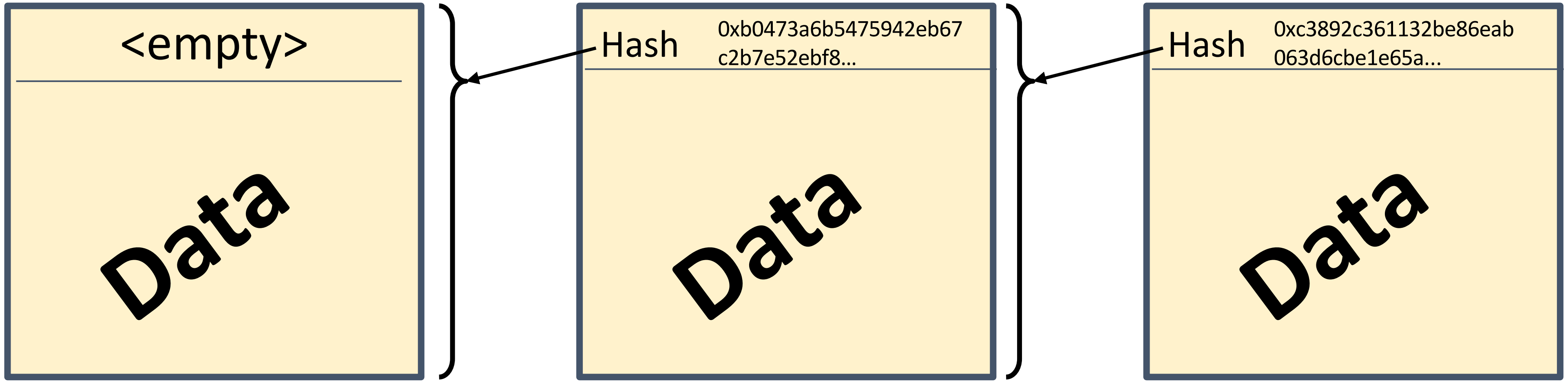
Application: Hash chaining



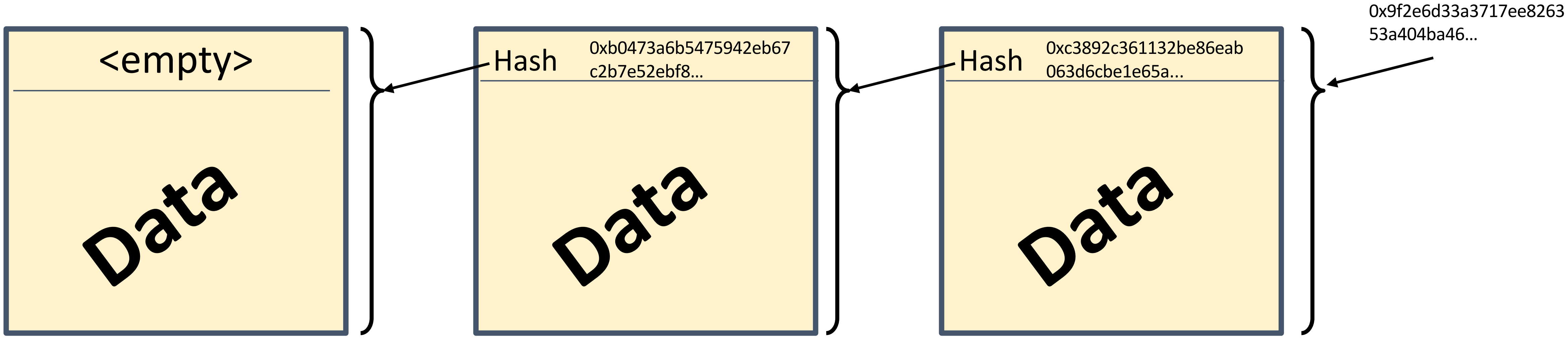
Application: Hash chaining



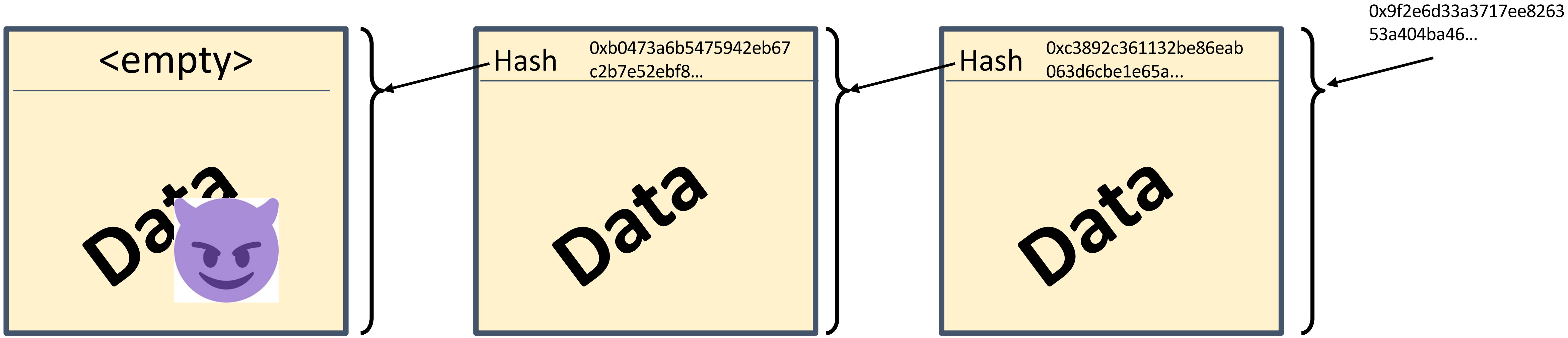
Application: Hash chaining



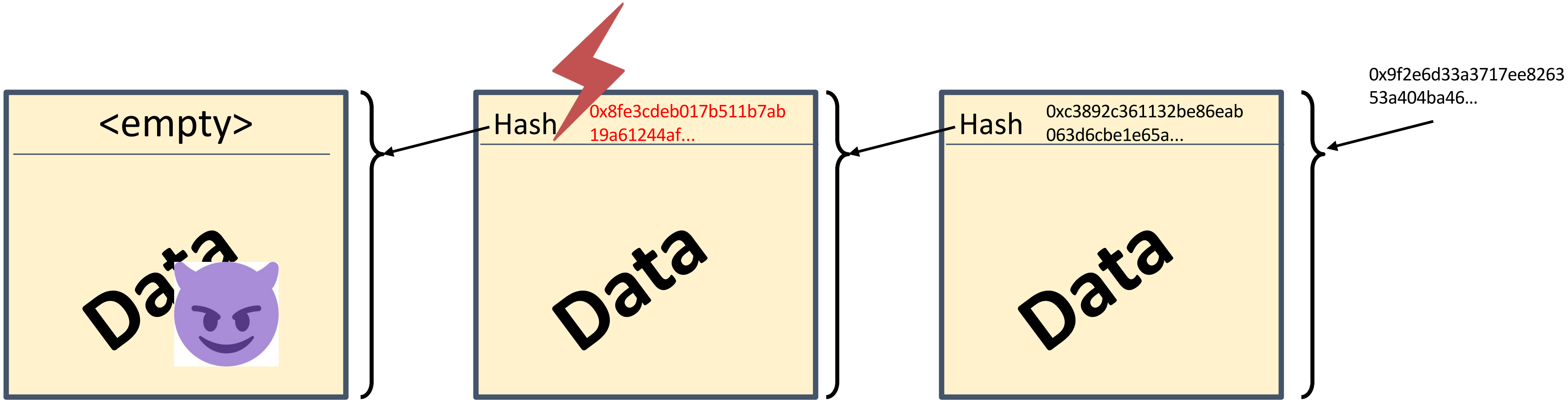
Application: Hash chaining



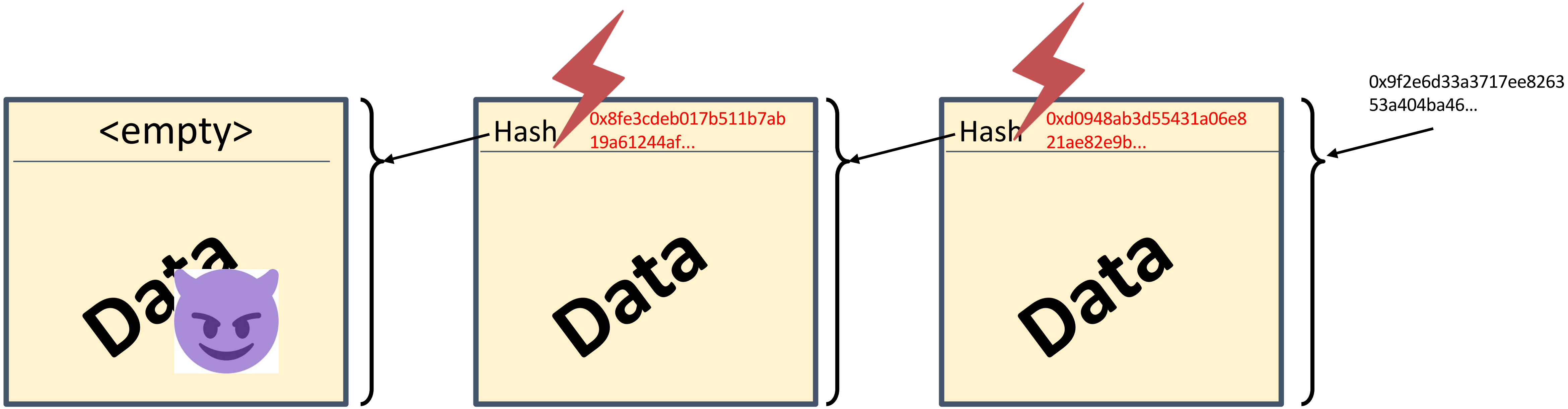
Hash chaining: Detecting tampering



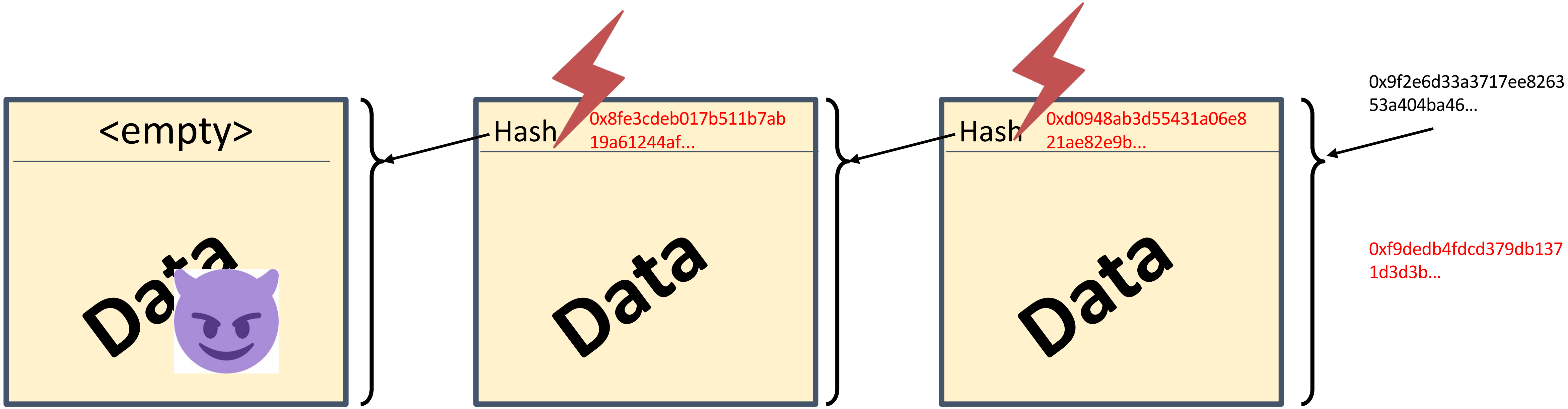
Hash chaining: Detecting tampering



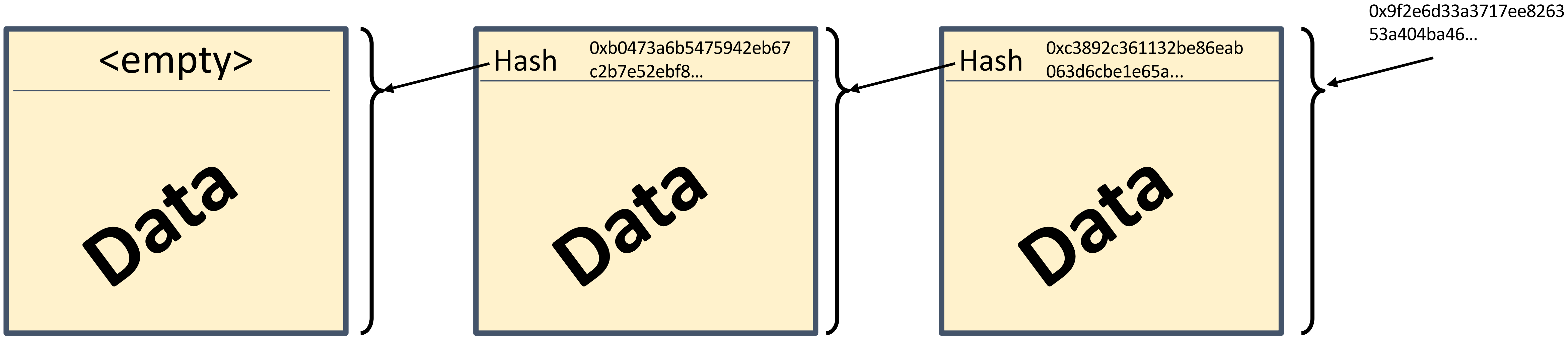
Hash chaining: Detecting tampering



Hash chaining: Detecting tampering



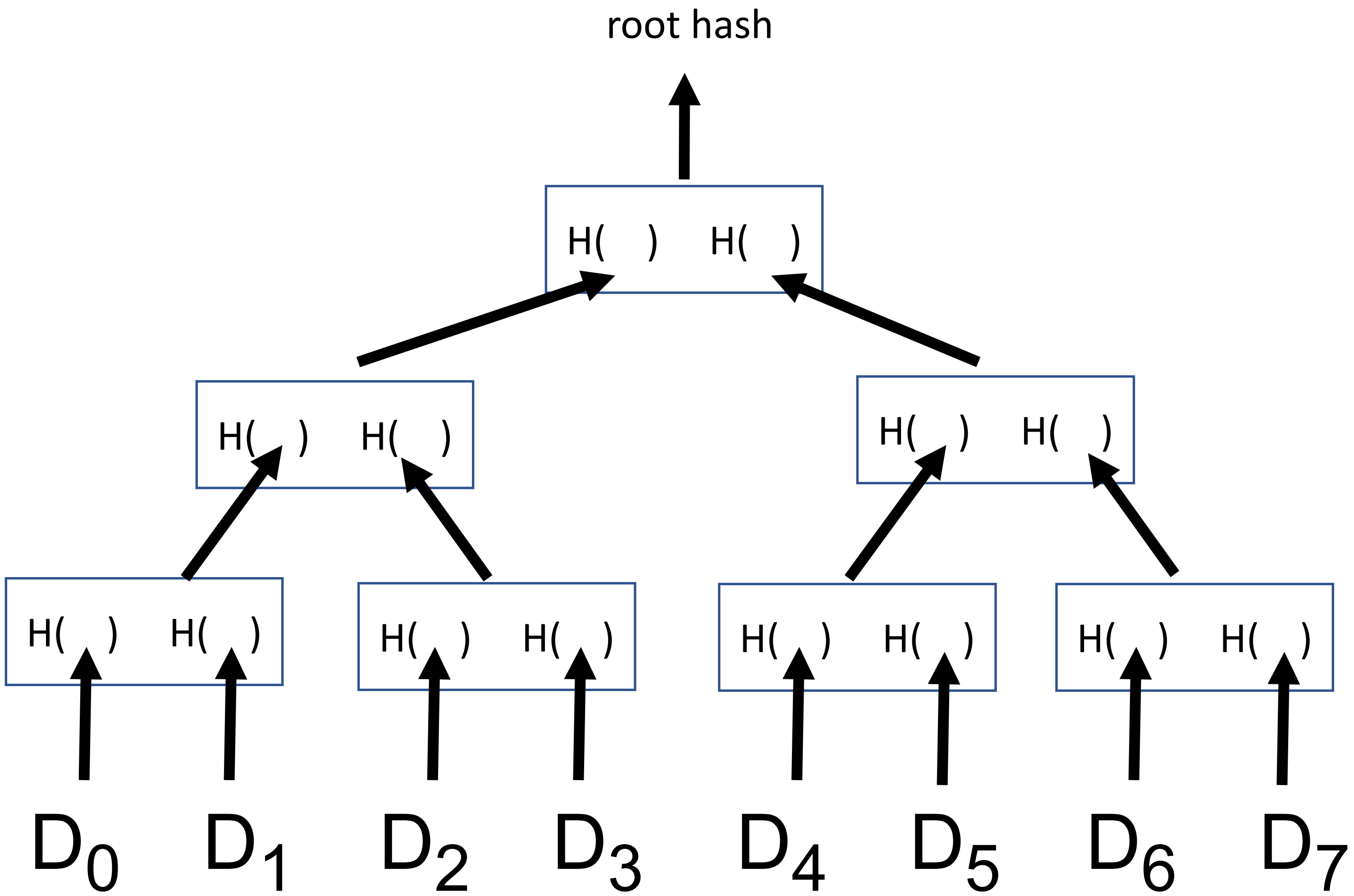
Hash chaining: Detecting tampering



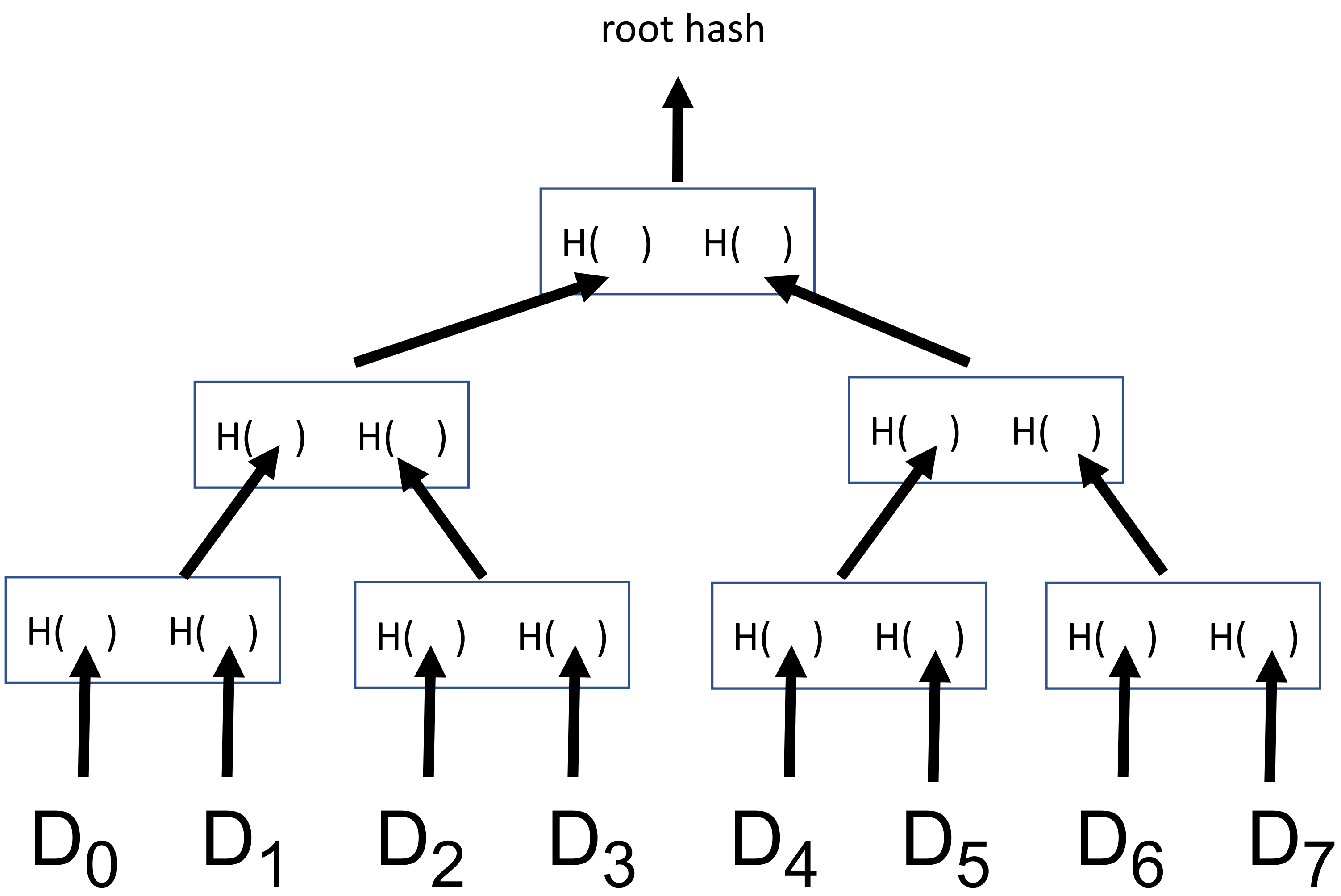
Linked List with Hash Pointers
Use Case: Tamper-evident log

Cornell University

Merkle Trees

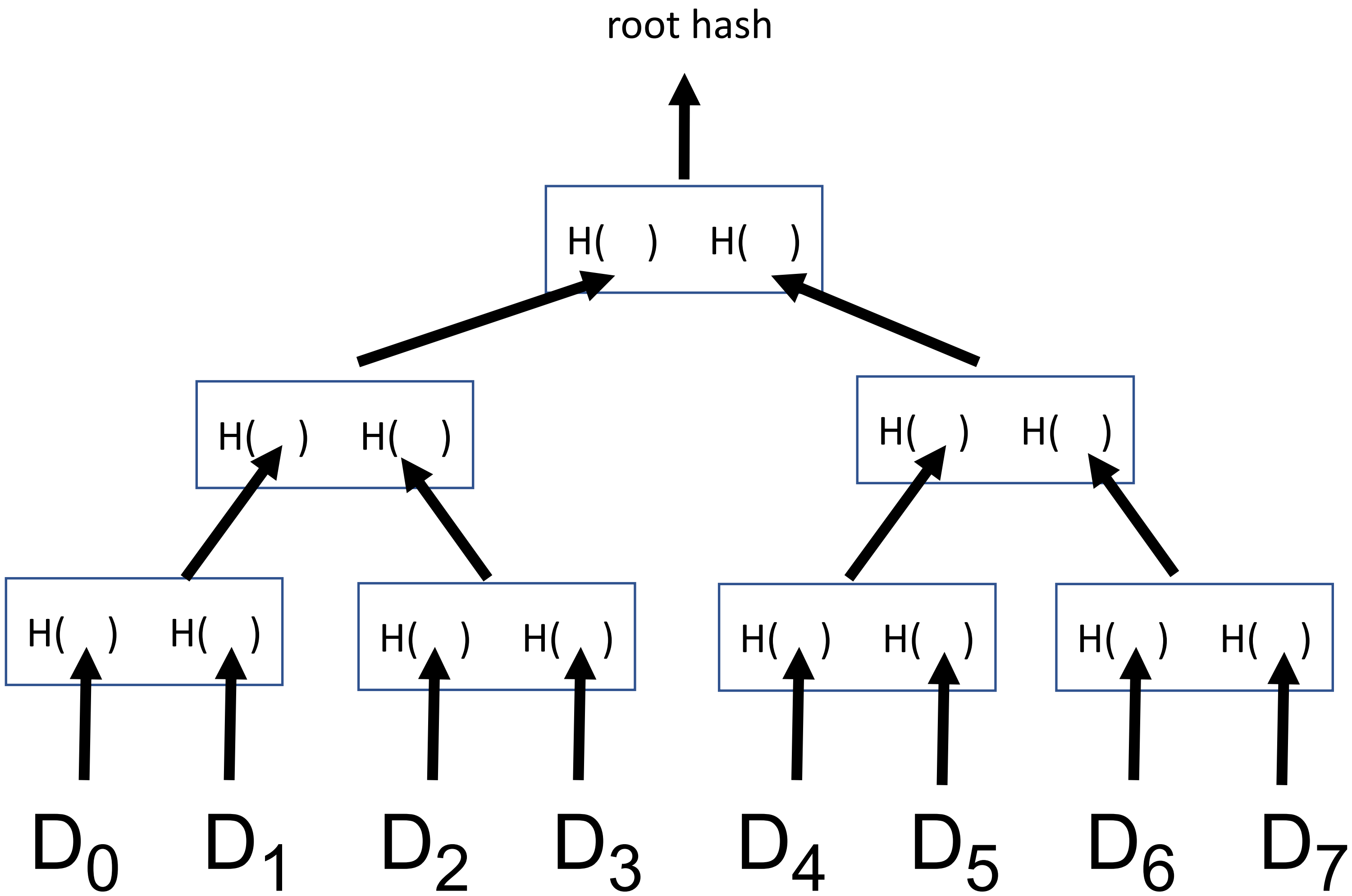


Proving Membership in a Merkle Tree



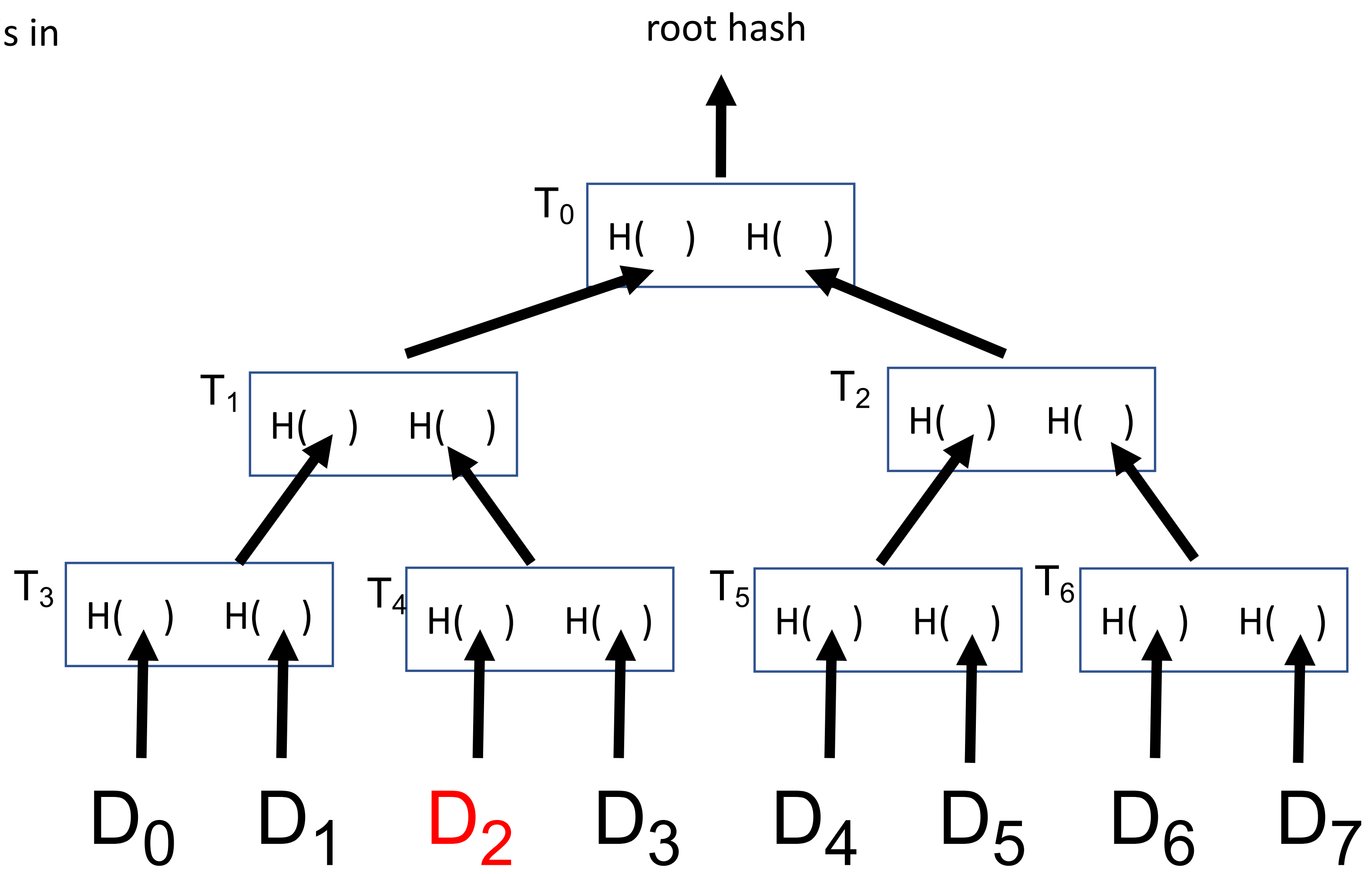
Proving Membership in a Merkle Tree

How do you prove D_2 is in the Merkle tree?



Proving Membership in a Merkle Tree

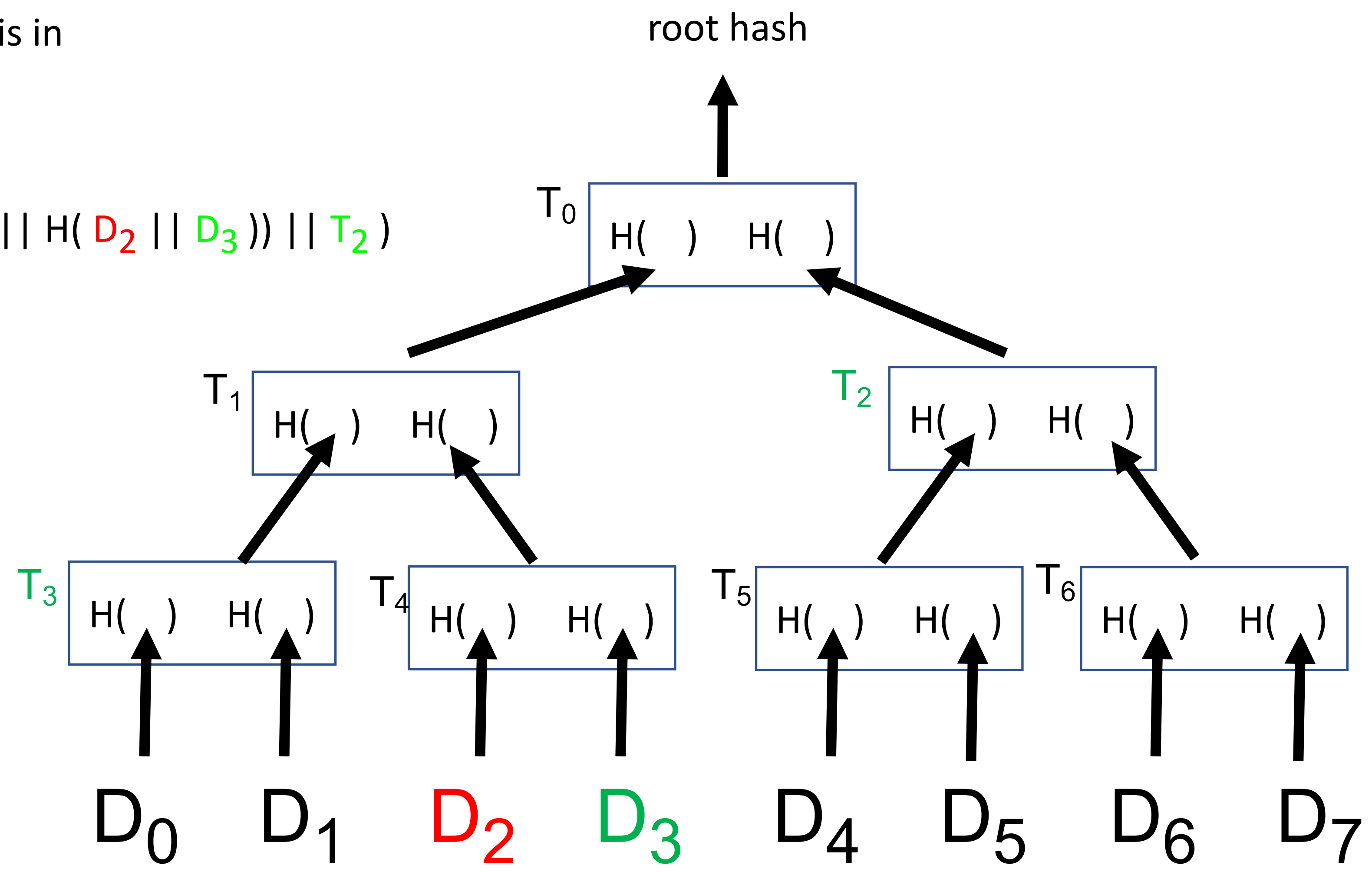
How do you prove D_2 is in the Merkle tree?



Proving Membership in a Merkle Tree

How do you prove D_2 is in the Merkle tree?

Verify if $T_0 = H(\,H(\,T_3 \,||\, H(\,D_2 \,||\, D_3\,)) \,||\, T_2\,)$ is true



Merkle Trees

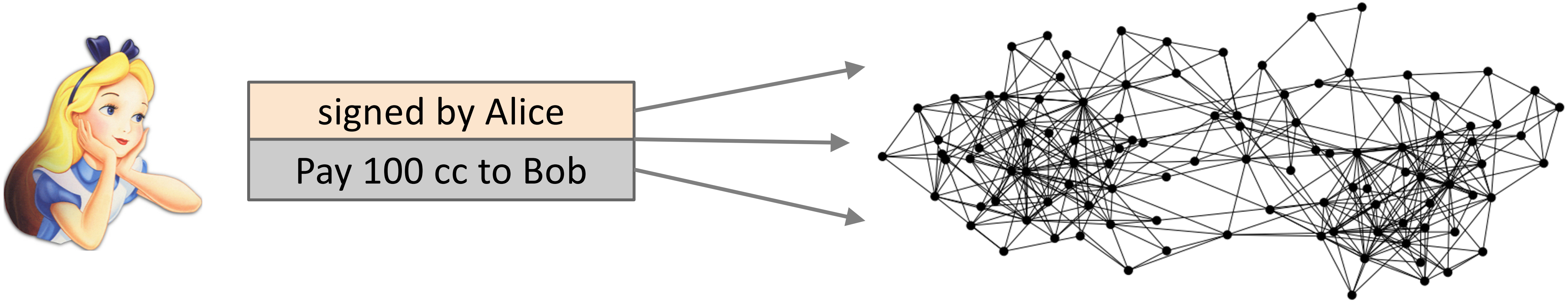
- Tree can hold many items
  - But only need to remember the root hash
  - Can verify membership in O(log n) time/space

Let’s build a global transactional system!

- Building blocks we now have:
    - Classical consensus algorithms: e.g. Paxos
    - Hash pointers and data structures
- Goal:
    - **public, decentralized, permissionless**

We want a peer-to-peer system

When Alice wants to pay Bob:
she broadcasts the transaction to all nodes



What nodes need to reach a consensus on?

- Which transactions were broadcast on the network
- Order in which these transactions occurred

→ Result of the consensus protocol:
Single, global transaction ledger for the system

How consensus <u>could</u> work in this system?

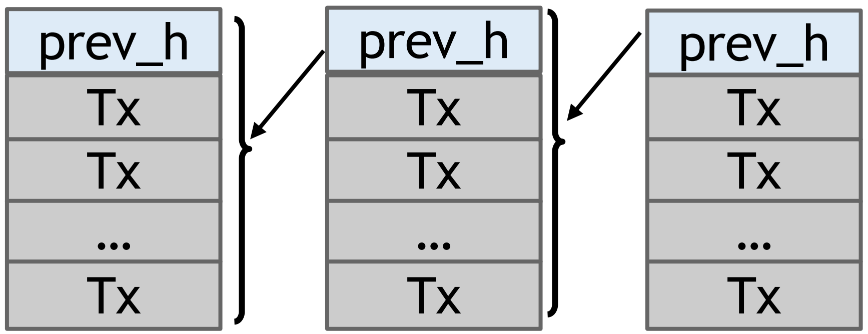
At any given time (in the peer-to-peer network):

- All nodes have a sequence of <u>blocks of transactions</u> they've reached consensus on
- Each node has a set of outstanding transactions it's heard about

How consensus could work in this system?

At any given time (in the peer-to-peer network):

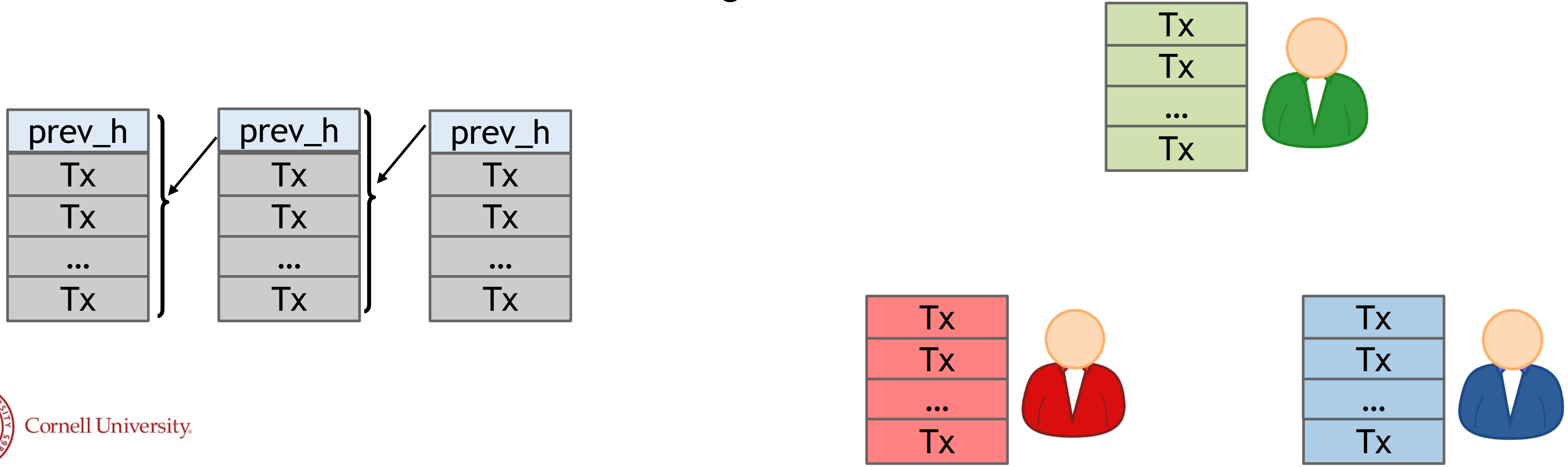
- All nodes have a sequence of blocks of transactions they've reached consensus on



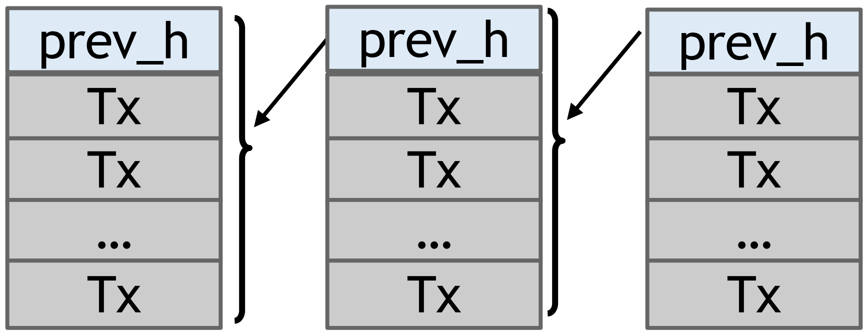
How consensus could work in this system?

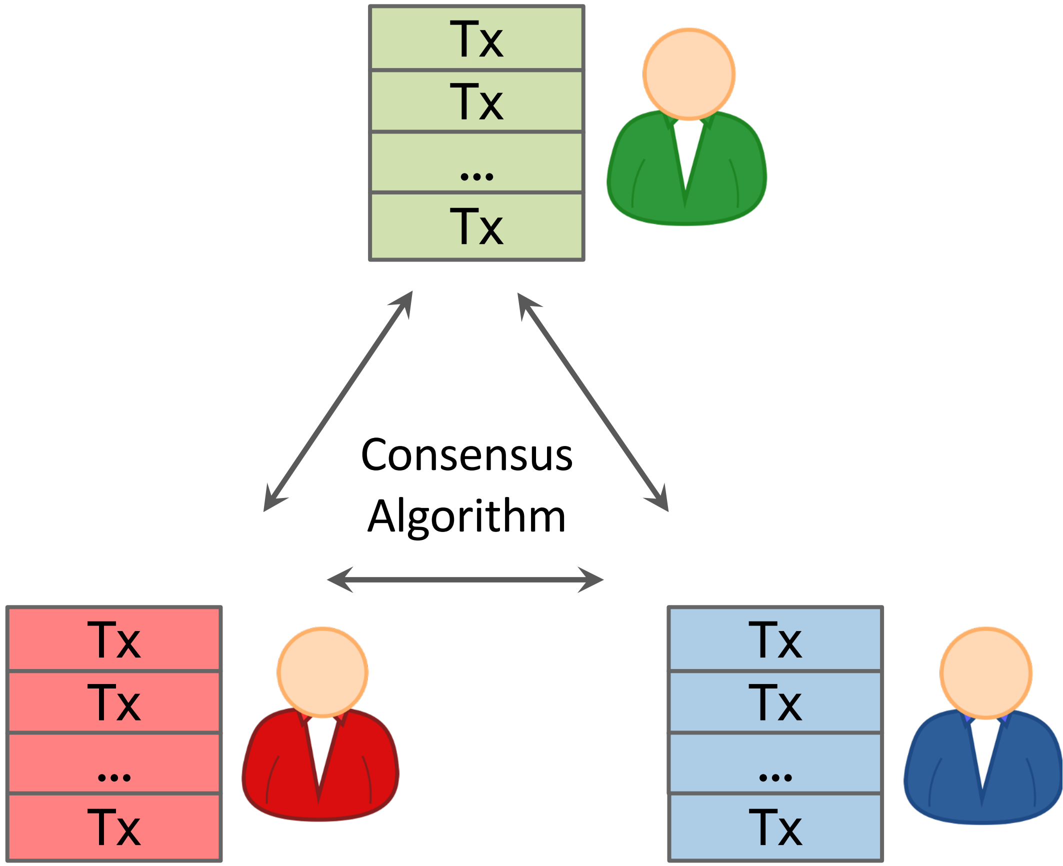
At any given time (in the peer-to-peer network):

- All nodes have a sequence of blocks of transactions they've reached consensus on
- Each node has a set of outstanding transactions it's heard about



How consensus could work in this system?

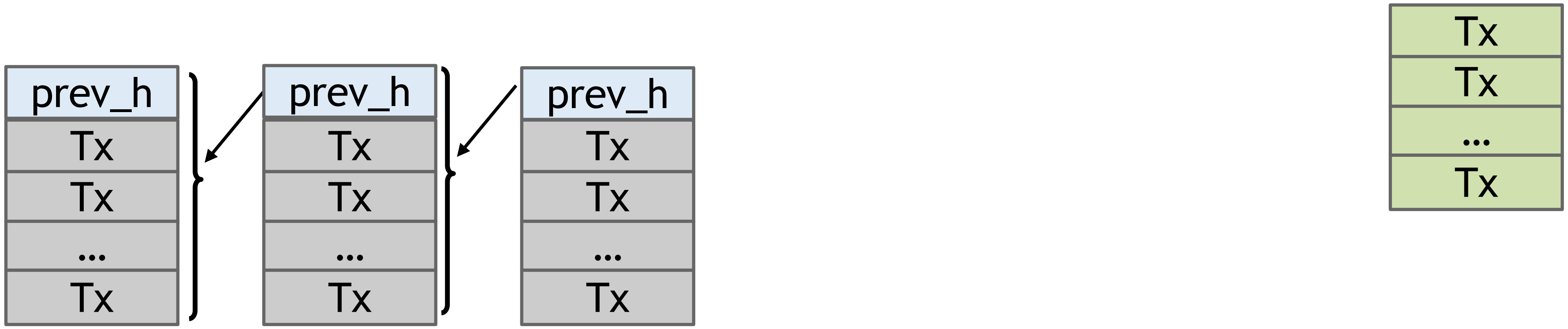




Cornell University

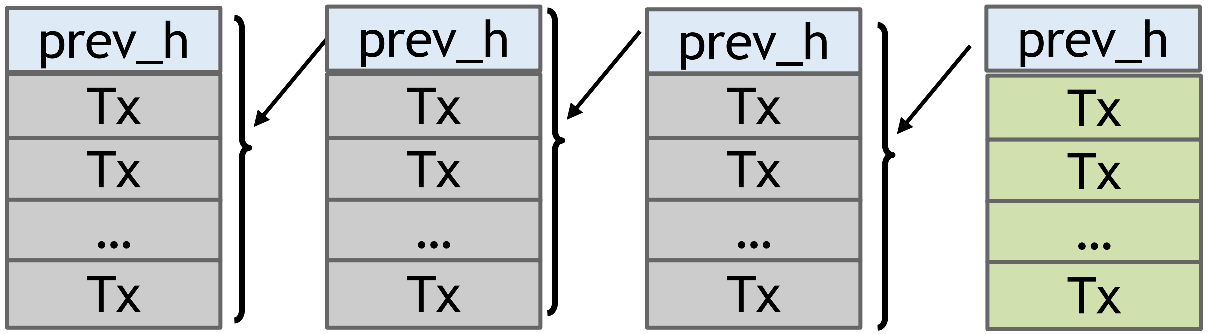
OK to select any valid block, even if proposed by only one node

How consensus could work in this system?



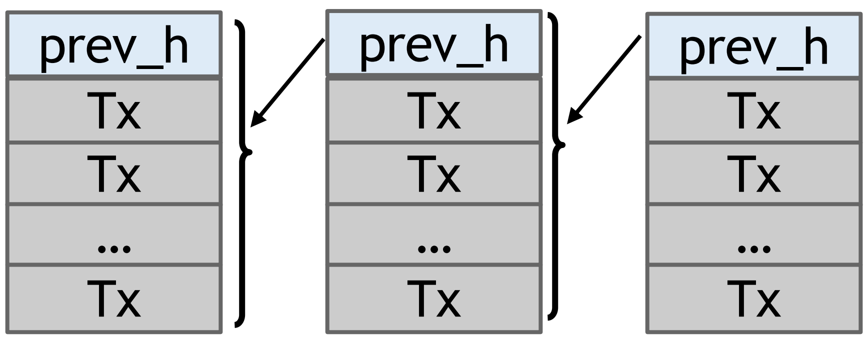
OK to select any valid block, even if proposed by only one node

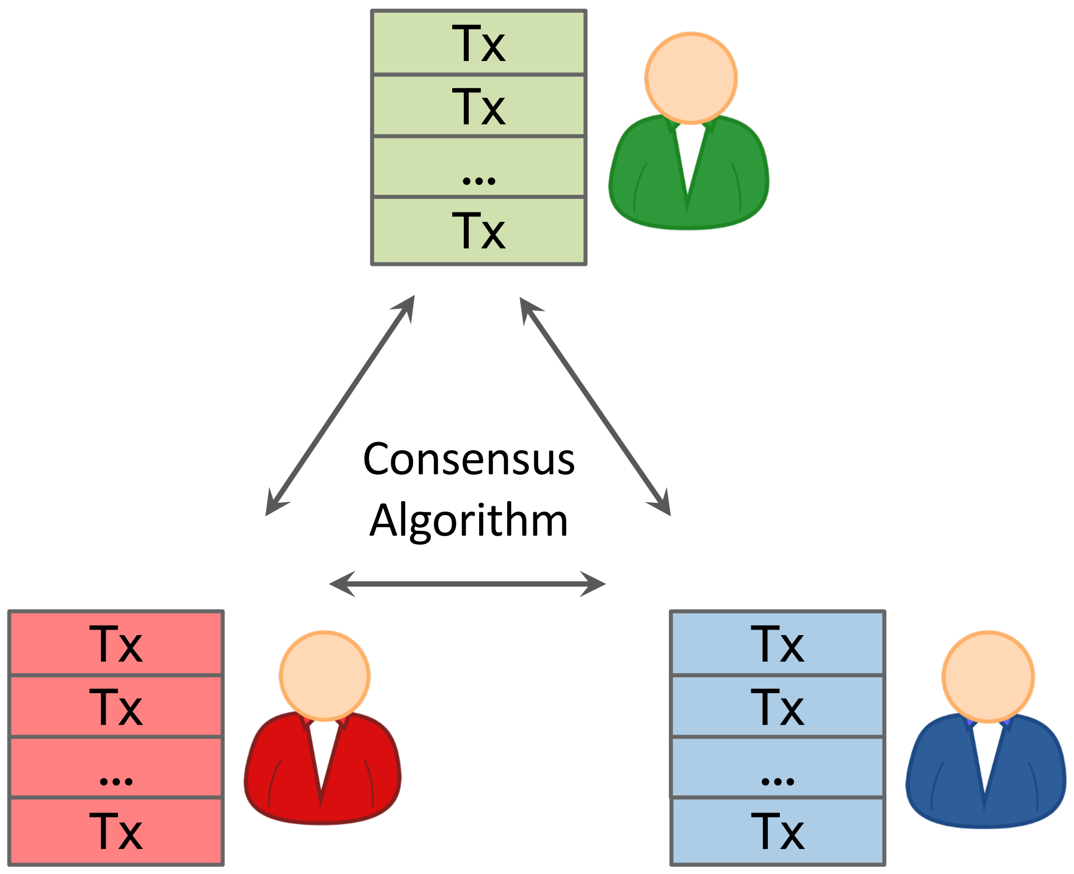
How consensus could work in this system?



OK to select any valid block, even if proposed by only one node

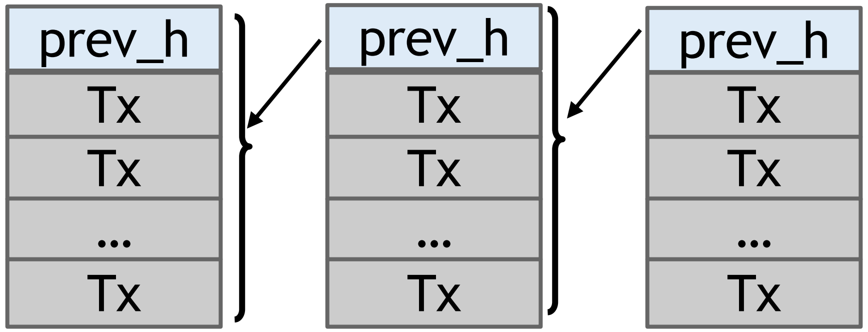
What Consensus algorithm to use?

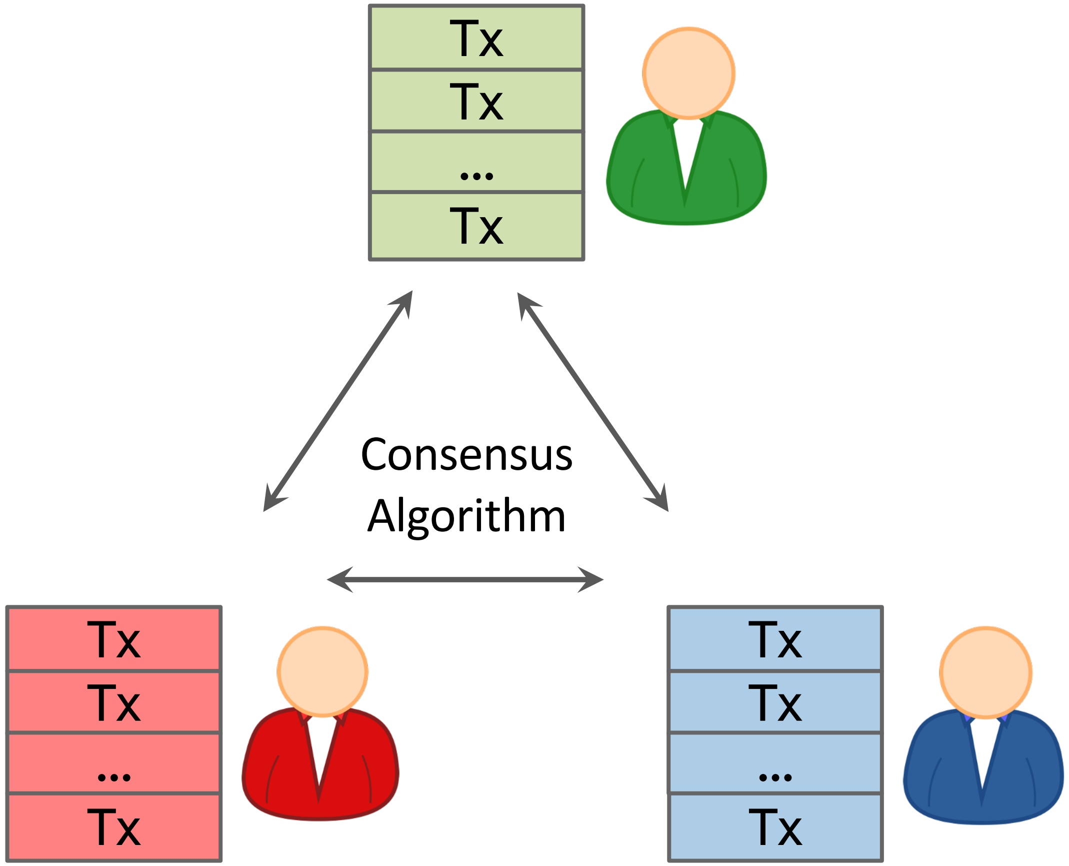




What Consensus algorithm to use?

- Why not just use Paxos?



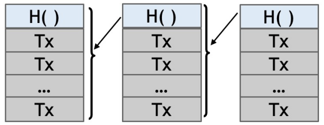


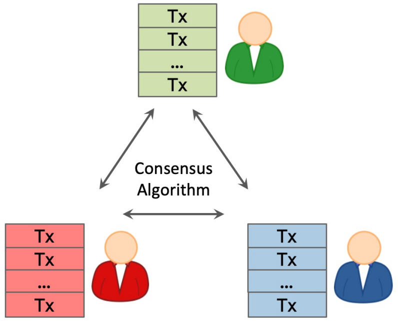
What Consensus algorithm to use?

- Why not just use Paxos?
- We want to build a **public, permissionless** system
  - Membership is permissionless: Any machine can join and leave at any time
  - 😈 Sybil attack: An attacker can spin up unlimited instances
- We are now designing in a different paradigm
  - Need a new consensus algorithm!

Key idea: implicit consensus

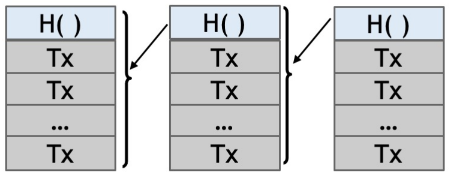
1. In each round, a random node is picked
2. This node proposes the next block in the chain
    - No voting done!
3. Other nodes implicitly accept/reject this block
    - by either extending it
    - or ignoring it and extending chain from earlier block
4. Every block contains hash of the block it extends

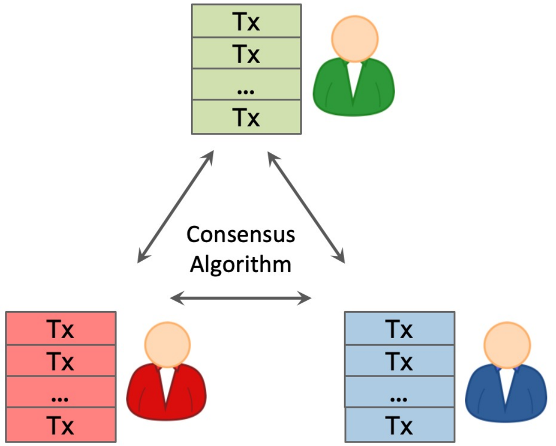




Consensus algorithm (simplified)

1. New transactions are broadcast to all nodes
2. Each node collects new transactions into a block
3. In each round a <u>random</u> node gets to broadcast its block
4. Other nodes accept the block only if all transactions in it are valid
5. Nodes express their acceptance of the block by including its hash in the next block they create



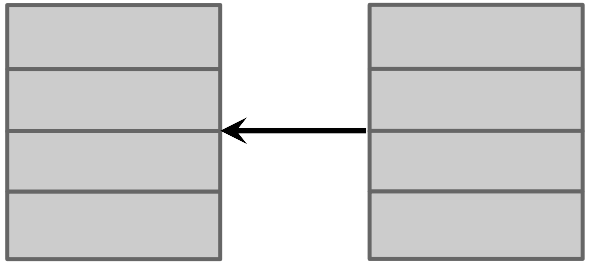


Now let's analyze if this works!

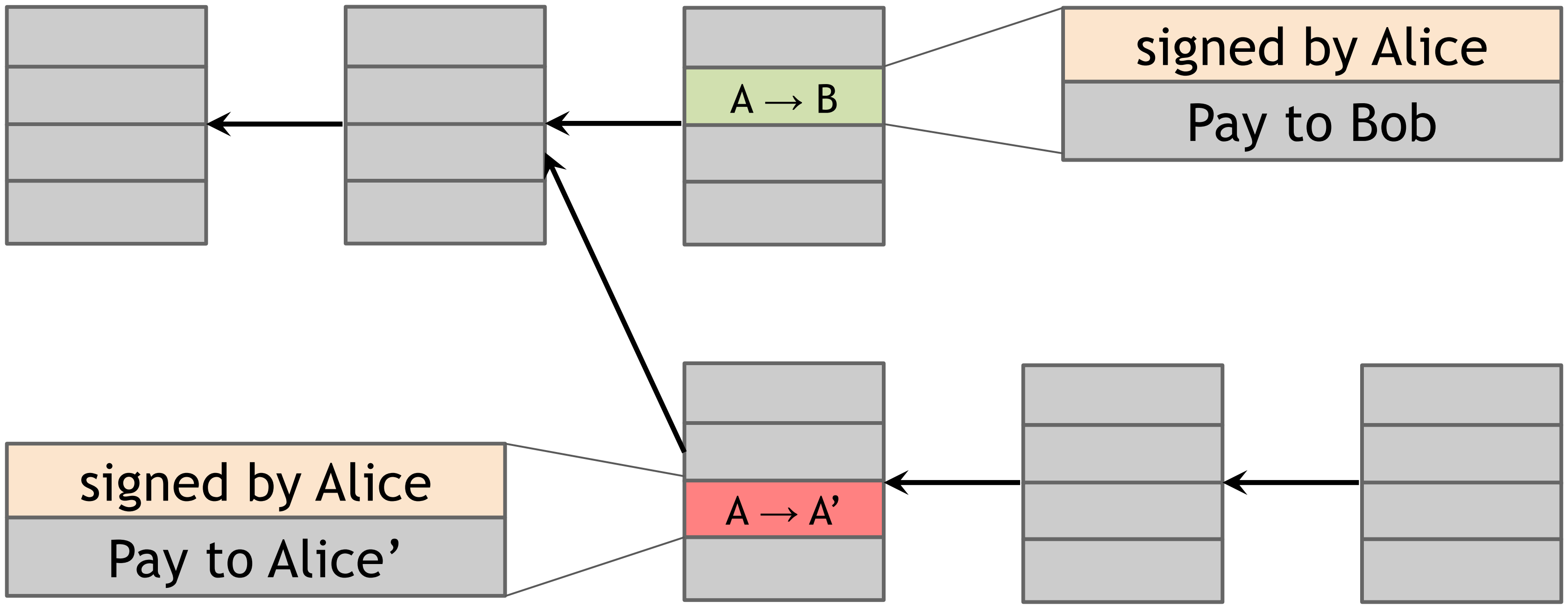
Assume a malicious adversary.

Can this adversary subvert the implicit consensus process by:

1. **Stealing funds?**
2. **Denial of service?**
3. **Double spend?**



What can a malicious node do?

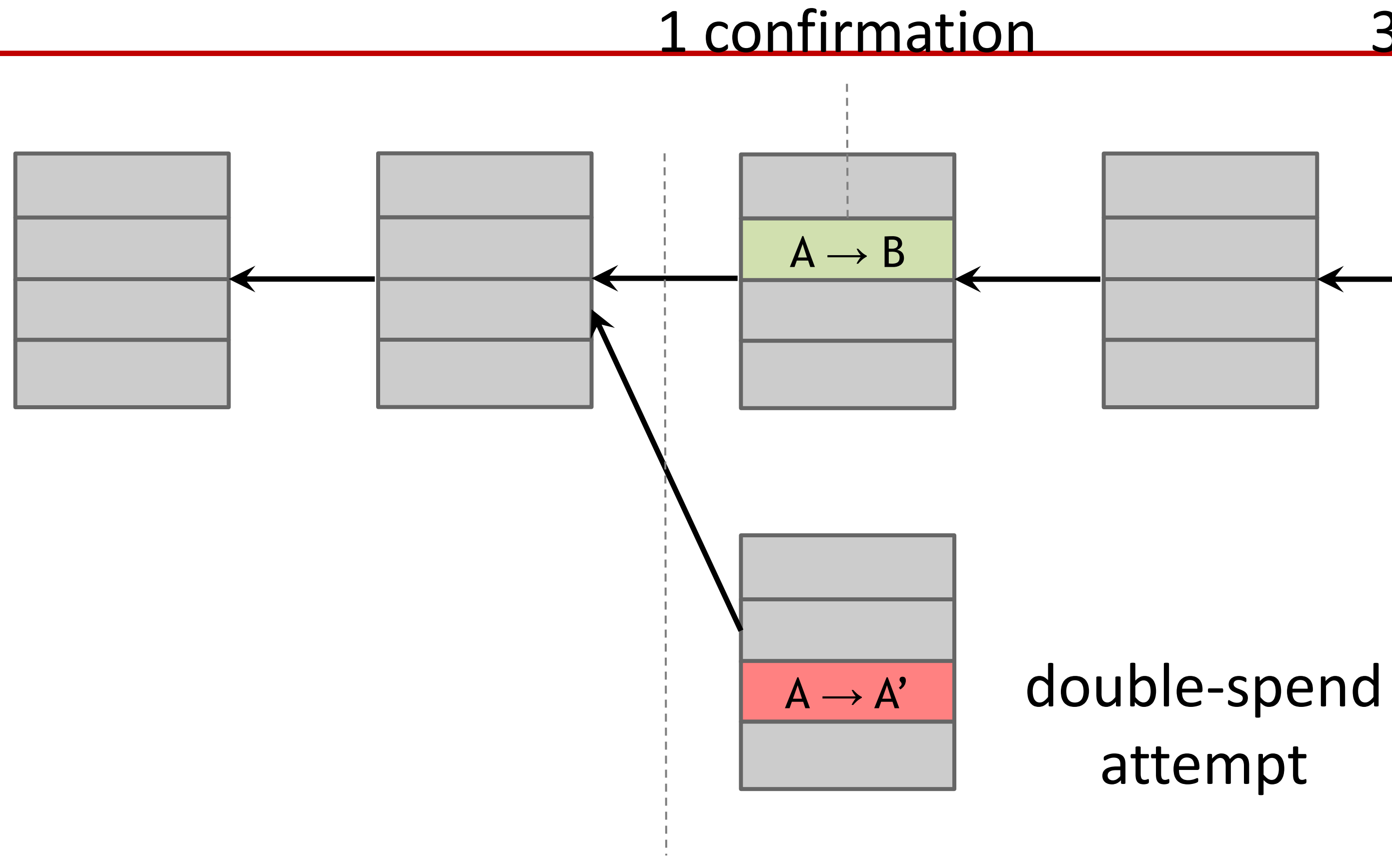


Double-spending attack

Assumption 1: Honest nodes will extend the <u>longest valid branch</u>

Assumption 2: The majority of nodes picked randomly are honest

From Bob the merchant's point of view



Hear about Alice → Bob transaction
0 confirmations

Double-spend probability <u>decreases exponentially</u> with # of confirmations

Recap

- Protection against invalid transactions is cryptographic, but enforced by consensus

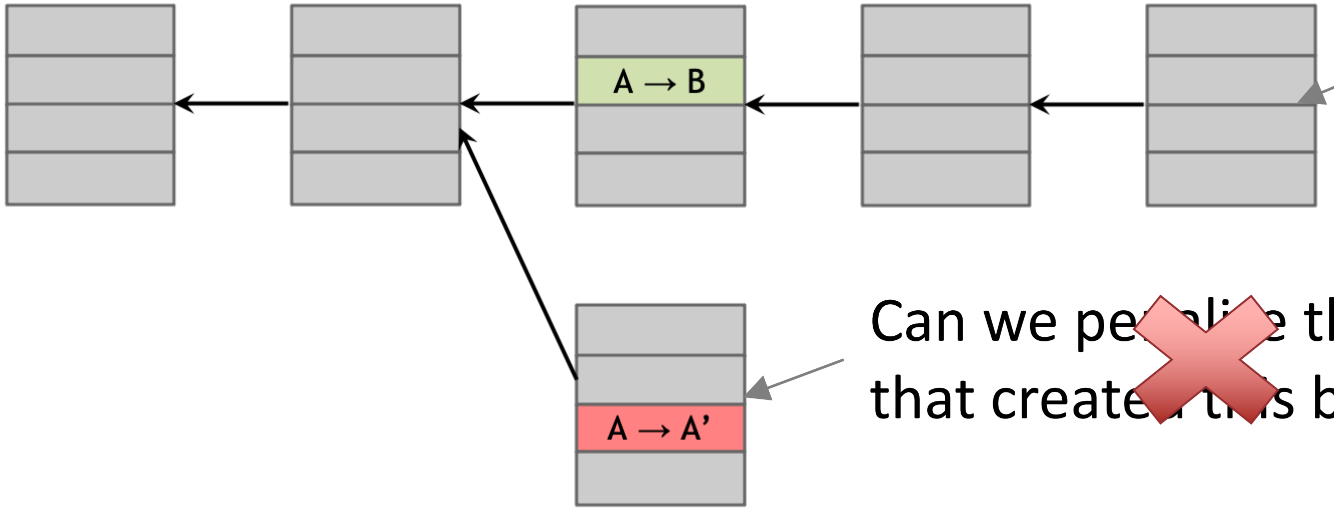
- Protection against double-spending is purely by consensus

- You’re never 100% sure a transaction is in consensus branch. Guarantee is probabilistic

- Assumptions:
  - Honest nodes will extend the <u>longest valid branch</u>
  - The majority of nodes picked randomly are honest

Assumption of honesty is problematic

Can we give nodes <u>incentives</u> for behaving honestly?

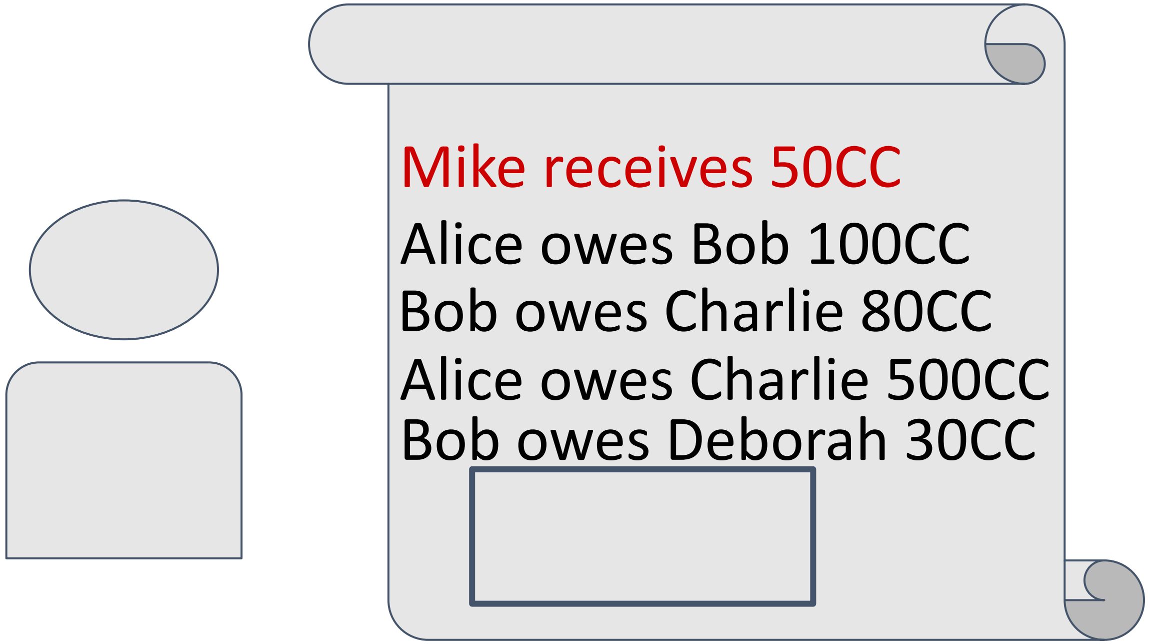


Can we reward nodes
that created these blocks?

~~Can we penalize the node
that created this block?~~

Incentives

- What’s in it for the honest block creators?



Block creator gets to “collect” the reward only if the block ends up on long-term consensus branch

Remaining problems

1. How to pick a random node?
2. How to avoid a free-for-all due to rewards?
3. How to prevent Sybil attacks?

Proof of Work

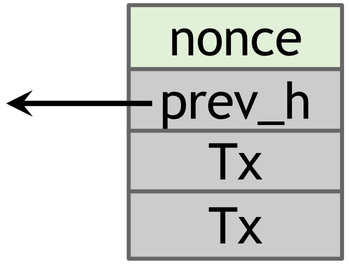
To approximate selecting a random node:

select nodes in proportion to a resource that no one can monopolize (we hope)

- In proportion to computing power: **proof-of-work**

Proof of Work

To create block, find nonce s.t.
H(nonce ‖ prev_hash ‖ tx ‖ … ‖ tx) is very small



If hash function is secure (***puzzle-friendly***):
only way to succeed is to try enough nonces until you get lucky

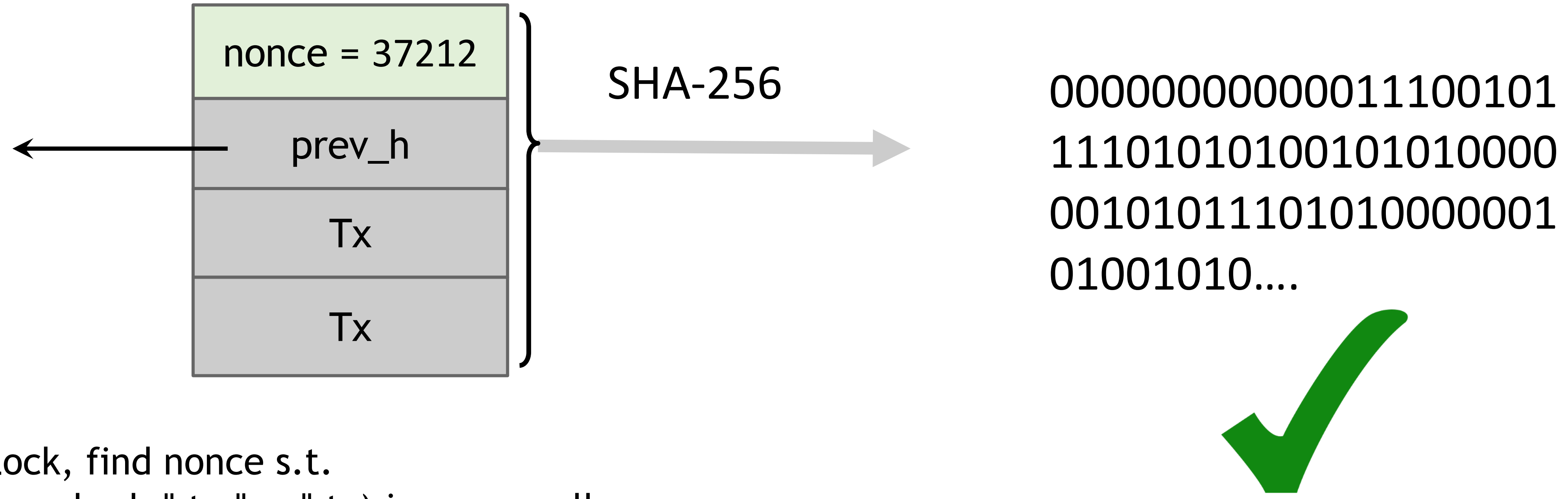
Proof of Work

To create block, find nonce s.t.
H(nonce ‖ prev_hash ‖ tx ‖ … ‖ tx) is very small

Proof of Work

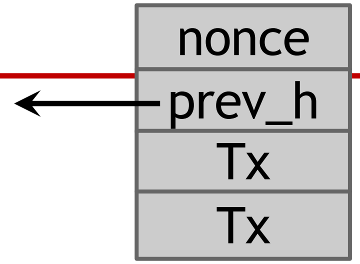
To create block, find nonce s.t.
H(nonce ‖ prev_hash ‖ tx ‖ … ‖ tx) is very small

Proof of Work

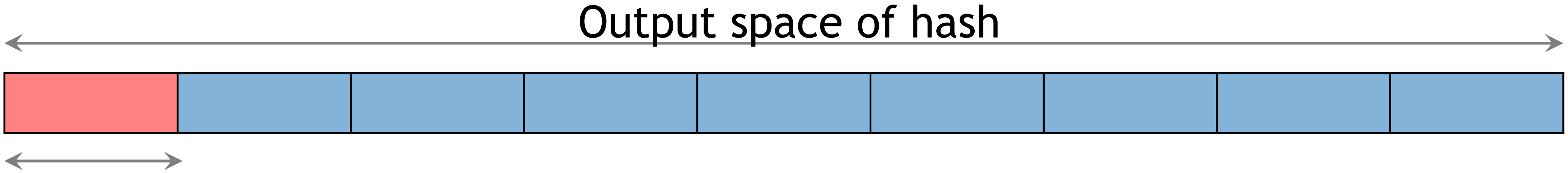


To create block, find nonce s.t.
H(nonce ‖ prev_hash ‖ tx ‖ … ‖ tx) is very small

Proof of Work



To create block, find nonce s.t.
H(nonce ‖ prev_hash ‖ tx ‖ … ‖ tx) < target



Target space

If hash function is secure:
only way to succeed is to try enough nonces until you get lucky

Equivalent views of Proof of Work

1. Select nodes in proportion to computing power
2. Let nodes compete for right to create block
3. Make it moderately hard to create new identities

Key assumption: Honest majority

Attacks infeasible if **majority of miners** weighted by hash power follow the protocol (or are honest)

This will ensure a more than 50% chance that the next block is proposed by an honest node

What’s different about Nakamoto consensus?

- Introduces economics and incentives
- Embraces randomness

If you are interested in the topic

Related Courses

- CS 5433 Blockchains, Cryptocurrencies, and Smart Contracts
  Prof. Ari Juels

- CS 5854 Networks and Markets
  Prof. Rafael Pass

- CS 5435 Computer Security
  Prof. Vitaly Shmatikov / Prof. Thomas Ristenpart

References

Slides adopted from:

- Narayanan, Bonneau, Felten, Miller, & Goldfeder. (2016). *Bitcoin and Cryptocurrency Technologies*.

http://bitcoinbook.cs.princeton.edu/

- Shi. (2020). *Foundations of Distributed Consensus and Blockchains*.

https://www.distributedconsensus.net/

- Colohan. (2016). *Distributed Systems*.

http://www.distributedsystemscourse.com/