

# **CPSC 3640/5400: Decentralized Payments, Contracts, and Finance for Humans and AI**

## **Lecture 2: Bitcoin Fundamentals**

---

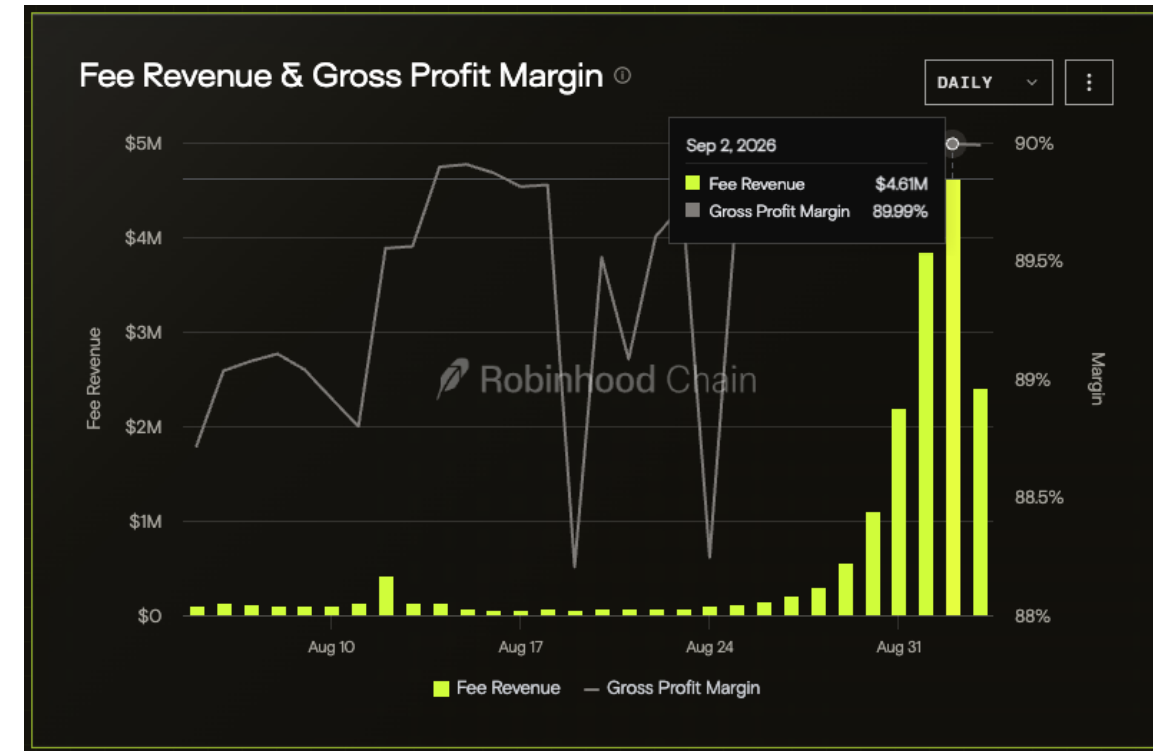
Instructor: Prof. Fan Zhang

Yale University

2026 Fall

# News of the week

- Robinhood Chain revenue surged to an ATH (\$4.61M on Sept. 2)
  - driven largely by memecoin speculation
- Interesting data:
  - <1% revenue to Ethereum (data avai.)
  - ~10% goes to Arbitrum (tech provider)
  - Robinhood keeps ~89%.
- Ethereum scales through L2s. But if L2s capture most of the direct revenue, how should ETH itself capture value?



# Today: Building blocks of Bitcoin

- Digital Signatures
  - Bitcoin users authorize transactions by digitally signing them
- Hash Functions
  - Numerous applications in Bitcoin and all blockchains
- Proof of Work (PoW)
  - The consensus layer of Bitcoin
  - Prevents double spending

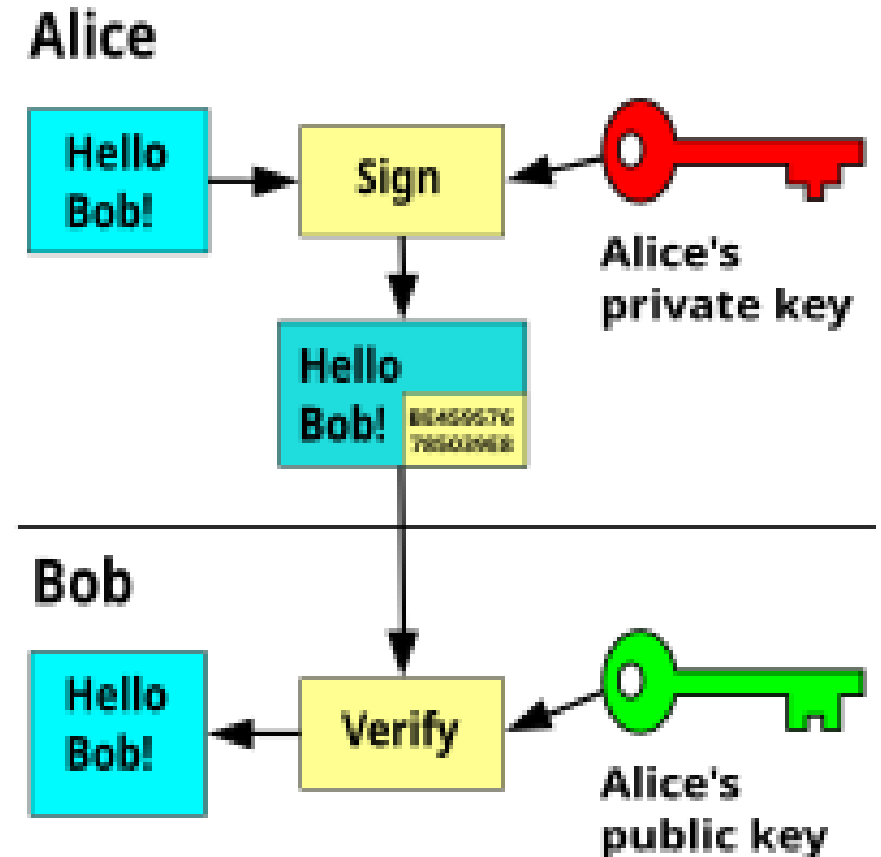
# Slides in this color are optional

- They provide additional learning materials

# Digital signatures

# Digital signatures

- Three algorithms
- $(SK, PK) = \text{KeyGen}()$ 
  - PK is published to the world
  - SK is kept secret
- $\sigma = \text{Sign}(SK, m)$
- $\{0, 1\} = \text{Verify}(m, \sigma, PK)$ 
  - Return 1 if  $\sigma$  is produced by the correct SK w.r.t. PK
  - Return 0 otherwise
- Key feature: can't produce valid signatures without SK.

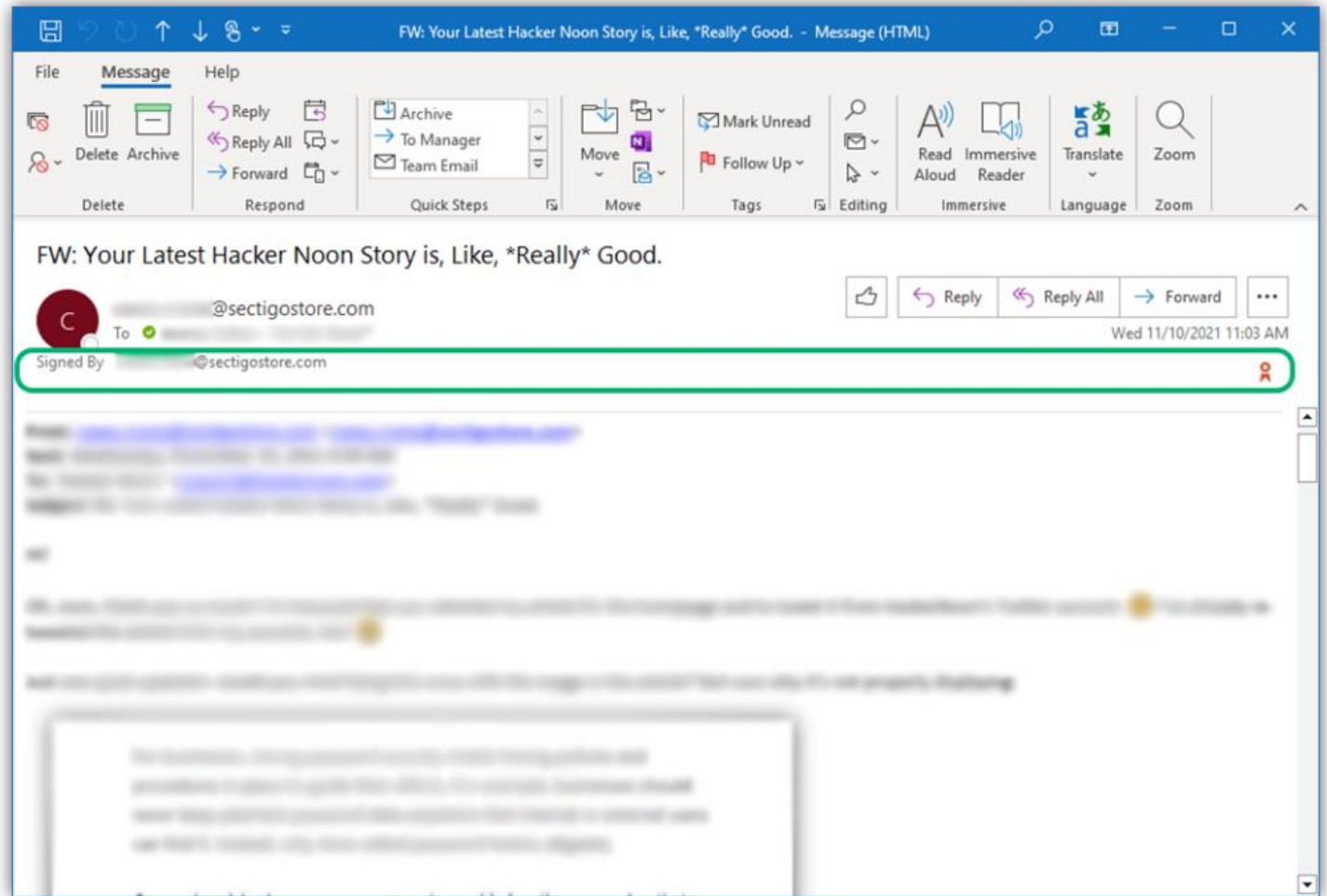
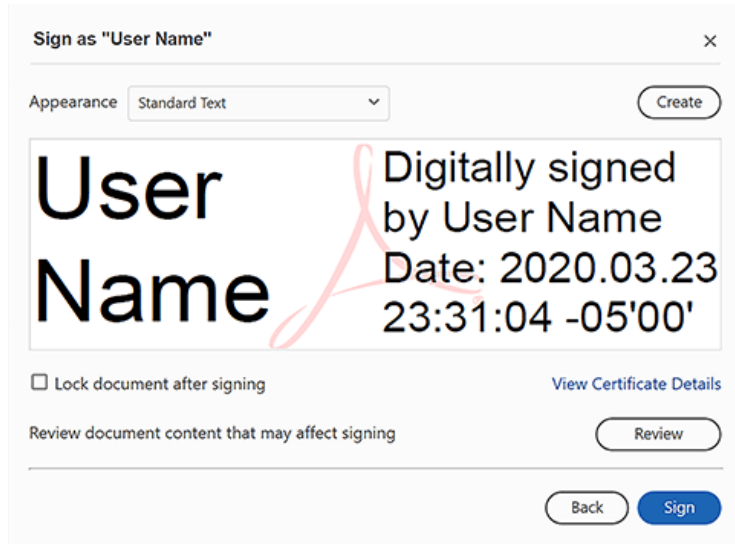


# Widely used outside blockchains



- When visiting google.com, the browser checks signatures against  $PK_{\text{google.com}}$

# Widely used outside blockchains



# Bitcoin uses ECDSA and Schnorr

- SK is a random big number,  $PK = g^{SK}$  (for some fixed  $g$ )
- A sample ECDSA key pair:
  - SK=0440e8f3638cc4b68d75201a1a9e3761a9c66b56439105a911d74091d1d0b5be (32 bytes)
  - PK=02abe7cc8e994a43d9b5c4d5d45cd723af1e728d7afe94789ec460ae7fb0d151bc (33 bytes, compressed)
- Two things to avoid:
  - Don't want your keys **stolen** (attacker gets them)
  - Don't want your keys **lost** (no body has them)

# Must use high-quality randomness

```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
             // guaranteed to be random.  
}
```

A bad random number generator.



(August 2026) Hackers reportedly drained more than \$100 million worth of bitcoin from Coldcard hard wallets.

# Don't let others generate your keys



- How can a malicious paper wallet generator steal your Bitcoins?

# Don't show private keys on camera



Bloomberg TV host Matt Miller flashed one of the private keys at the camera, unaware that he'd opened the account up to theft. It wasn't long before someone [collected the Bitcoin](#).

--- Dec 23, 2013

Risks of getting keys stolen

Store the keys in your laptop, phone, Google Drive.

Only ever store the key on a hardened hardware wallet.

Risks of losing keys

# Wallet design (we will come back to this)

- How to trade off these risks in a usable way?
- Good practices:
  - Social recovery
  - Multisig wallets
  - Hot-cold separation
  - Setting daily transaction limits
- A major adoption challenge, driving users to bank their coins in centralized services.

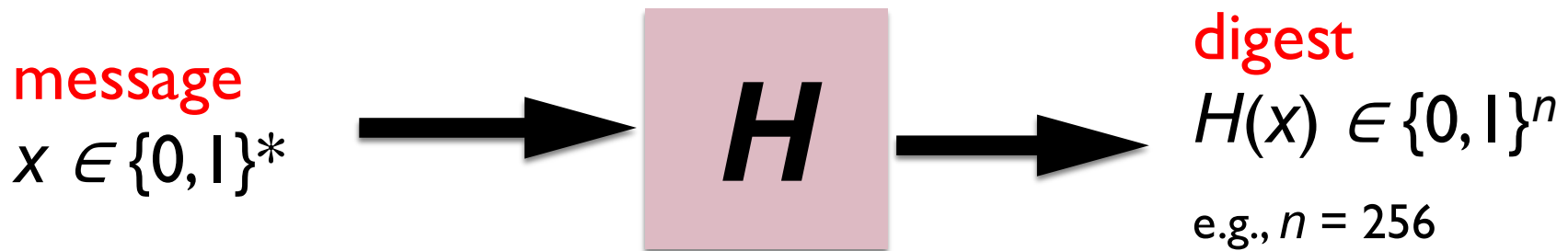
# Hash functions

The workhorses of modern cryptography



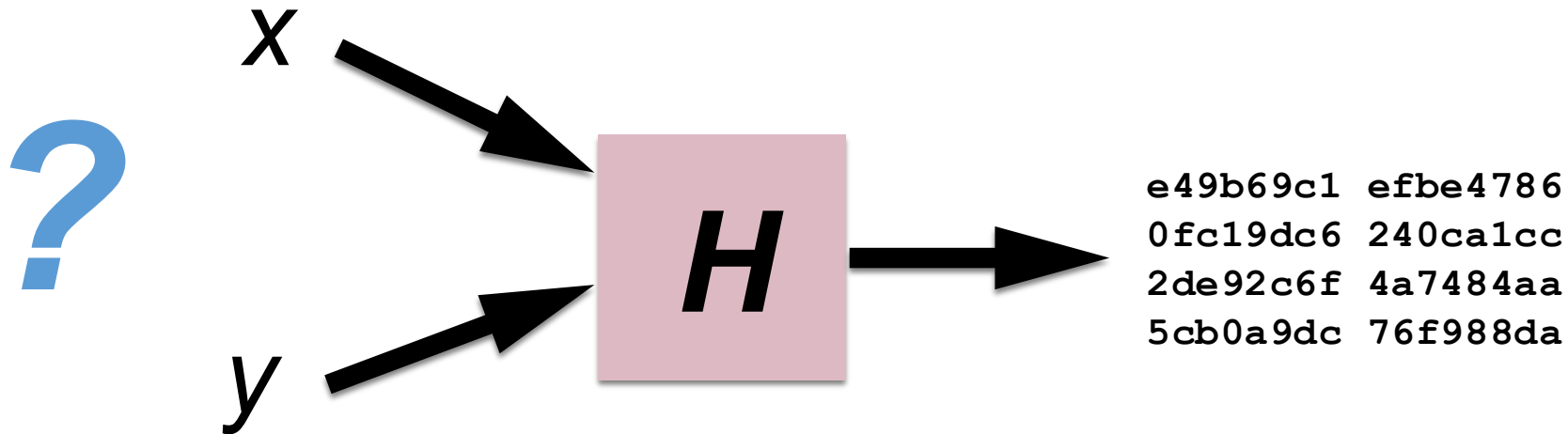
# Hash function: Definition

- Deterministic function maps **message**  $x$  (of any length) to fixed-length **digest** (e.g., 256 bits)
- Usually written as  $H(x)$ , read as the hash of  $x$



# Collision Resistant Hash Functions (CRH)

It is hard to find  $x \neq y$  such that  $H(x) = H(y)$



- Note: Collisions must exist for any CRH (why?)
- CRH requires that **it is hard to find one**
- How hard? E.g.,  $2^{128}$  steps –  $10^{13}$  year at  $10^{18}$  steps/second, longer than universe's age

# Common cryptographic hash functions

- Common examples are SHA (Secure Hash Algorithm) Family
- **SHA-1 (designed by NIST in 1995 to withstand  $2^{80}$  steps)**
  - In 2005, Wang et al announced an algorithm that can find collisions in  $2^{69}$  steps
  - First real collision demonstrated in 2017 by Google/CWI after spending 6,500 CPU-years + 110 GPU-years ( $\sim 2^{63}$  steps)
- **SHA-2** family is still believed to be secure and in wide use
  - Individual hash functions are named SHA-n ( $n \in \{224, 256, 384, 512\}$ )
- **SHA-3** released by NIST in 2015 after a 5-year design competition
- Example: <https://codebeautify.org/sha2-hash-generator>
- We can think of them as an idealized model called **Random Oracle**.

# The ideal hash functions: random oracles

```
Tape = {} # initially empty
def ROn(x):
    if x in Tape:
        Return Tape[x]
    else:
        Sample n bit random output y
        Tape[x] = y
        return Tape[x]
```

An *imaginary* global function available to everyone in the world

**x**

**y=RO<sub>n</sub>(x)**

- RO **cannot** be realized in practice. (Real-world CRH come close)
- Simplicity! We will use RO to analyze Bitcoin.

# **Blockchain applications of CRHs**

# I. Transaction IDs

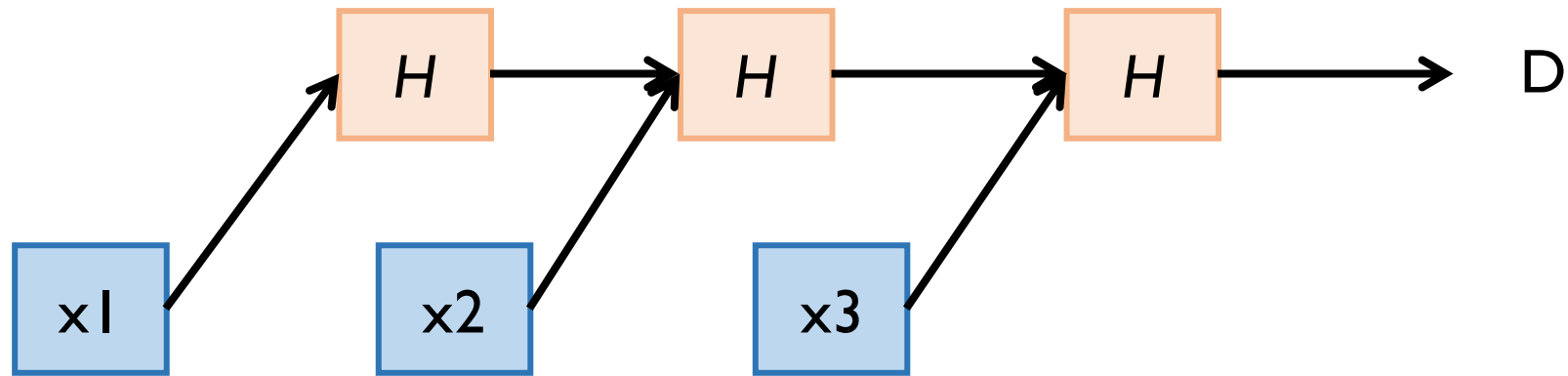
- Hash of transaction data is **transaction identifier (TXID)**
- Guaranteed to be unique (thanks to which property of CRH?)
- Examples:
  - [f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16](https://blockchain.info/tx/f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16):  
First ever Bitcoin transaction, to Hal Finney (2010)
  - [a1075db55d416d3ca199f55b6084e2115b9345e16c5cf302fc80e9d5fbf5d48d](https://blockchain.info/tx/a1075db55d416d3ca199f55b6084e2115b9345e16c5cf302fc80e9d5fbf5d48d):  
Pizza purchase for 10,000 BTC (2010)

# Digest of many items

- Recall:  $H(x)$  is the digest of  $x$
- What if we want to produce digest for many items?
- Natural ideas
  - Concatenate individual hashes
  - hash the concatenation of all items
- Better data structure: hash chain, hash tree

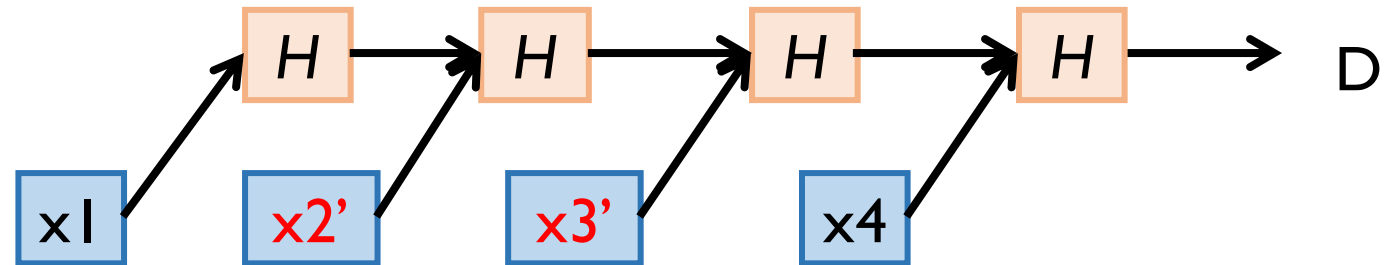
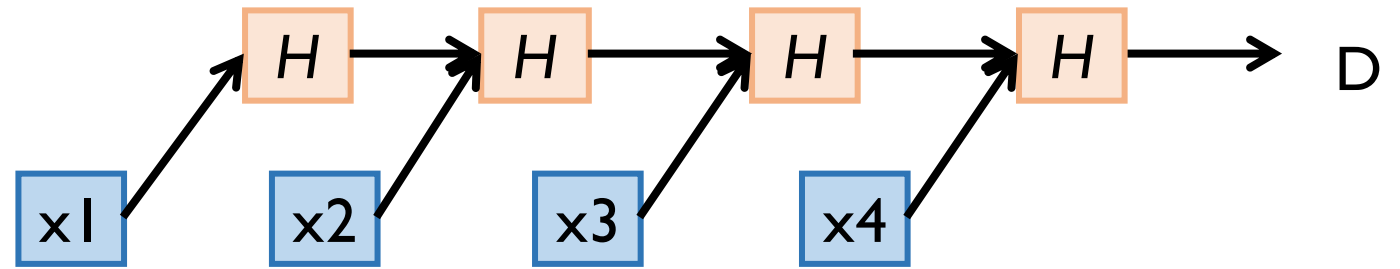
# Hash chain

- Suppose we want to compute digest of  $\{x_1, x_2, \dots, x_n\}$
- $h_1 = H(x_1)$
- $h_2 = H(h_1 \parallel x_2)$
- ...
- $h_n = H(h_{n-1} \parallel x_n) \rightarrow h_n$  is the digest



# Hash chain

- Is hash chain collision resistant?
- Suppose not...
- $S1 = \{x1, x2, x3, x4\}$
- $S2 = \{x1, x2', x3', x4\}$



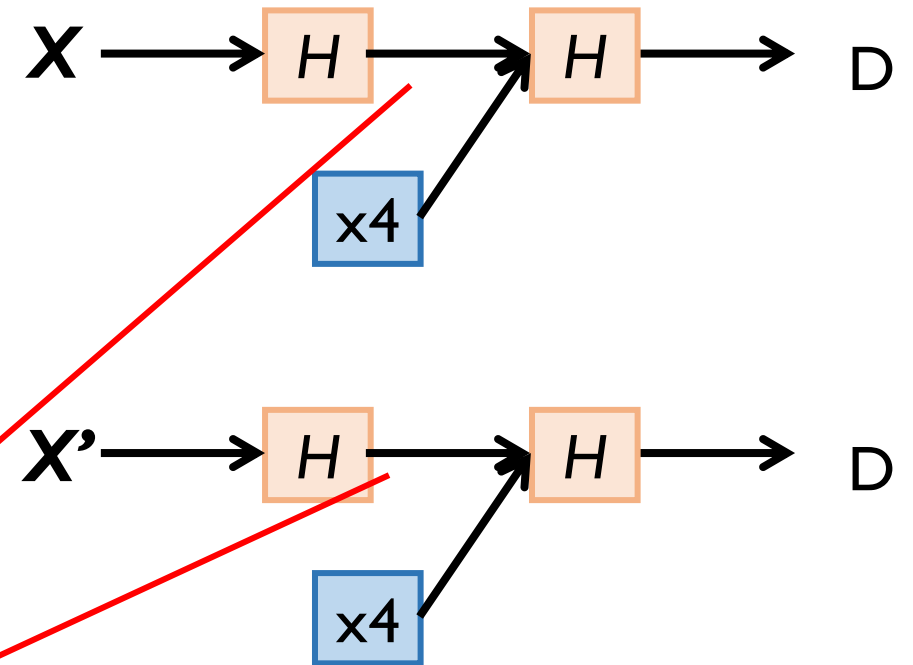
# Hash chain

- Is hash chain collision resistant?
- Suppose not...
- $S1 = \{x1, x2, x3, x4\}$
- $S2 = \{x1, x2', x3', x4\}$

Collision?

If yes, we are done.

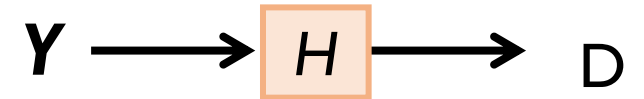
If not, let  $H(X) || X_4 = Y$ ,  
 $H(X') || X_4 = Y'$



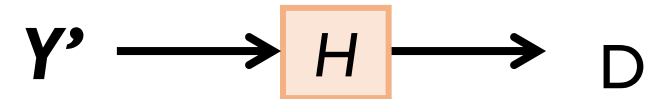
# Hash chain

- Is hash chain collision resistant?
- Suppose not...
- $S1 = \{x1, x2, x3, x4\}$
- $S2 = \{x1, x2', x3', x4\}$

Since the final digest is the same, this is a collision!

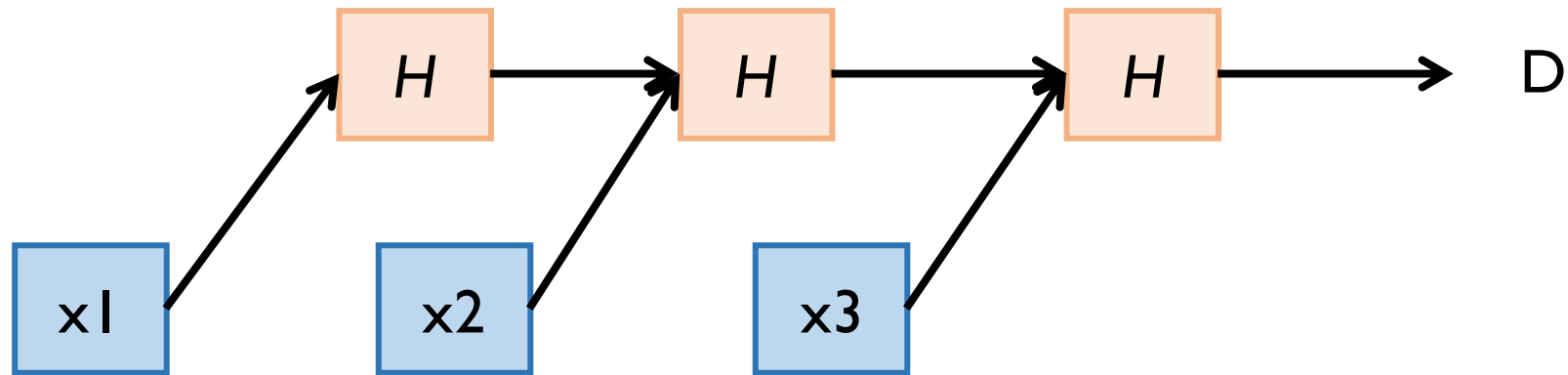


- We can find collision in H (in much fewer than  $2^n$  steps)
- QED (proof by contradiction)



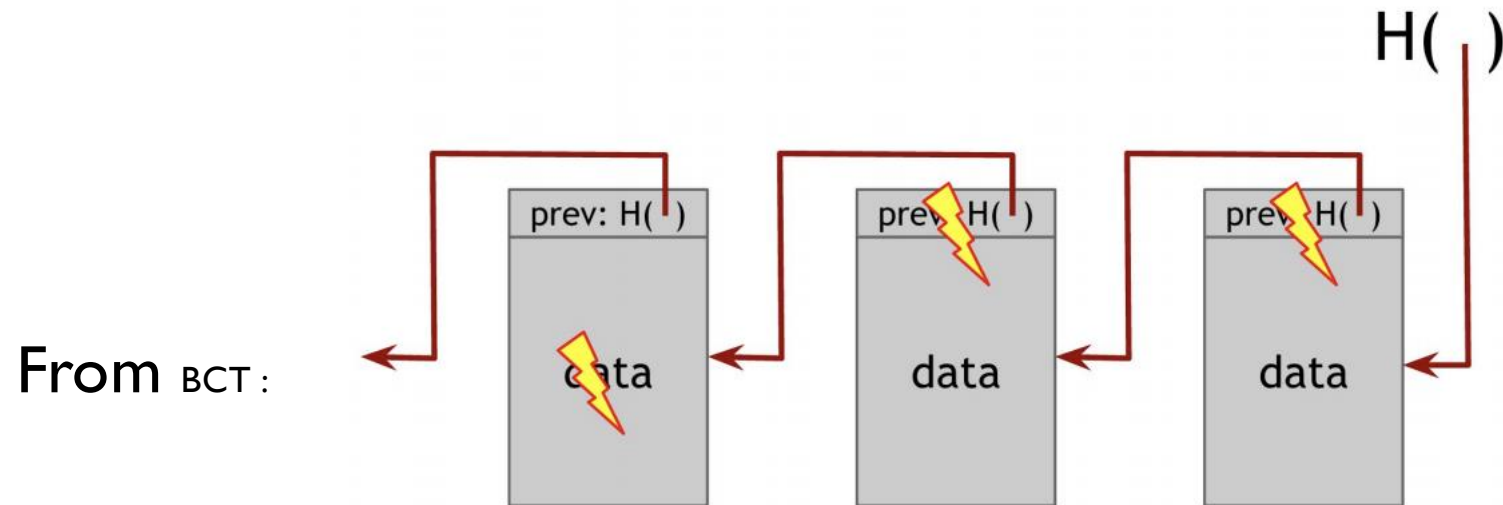
# Hash chain

- Feature: digest commits to the **ordering**
  - $(x_1, x_2, x_3)$  v.s.  $(x_1, x_3, x_2)$  have different digests
- Bitcoin headers form a hash chain



## 2. Chaining blockchain blocks

- Block header includes hash of previous block

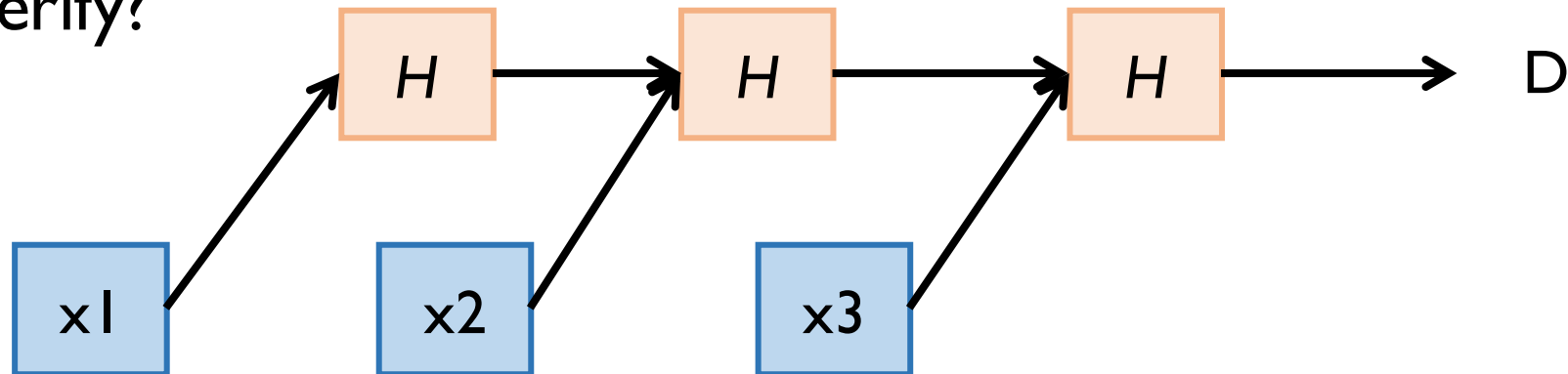


- The hash of the last block defines a unique sequence of blocks.

[BCT] “[Bitcoin and Cryptocurrency Technologies](#)” by Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder.

# Hash chain

- Downside: linear-size **membership proof**
- Verifier: with input  $D$  (digest)
- Prover: with input  $\{x_1, x_2, \dots, x_n\}$
- Membership proof  $\pi_x$  convinces Verifier that  $x \in X$
- What's the membership proof of  $x_1$  for hash chain?
- Answer:  $(x_2, \dots, x_n)$
- How to verify?



# 3) Merkle Tree (Ralph Merkle '1979)

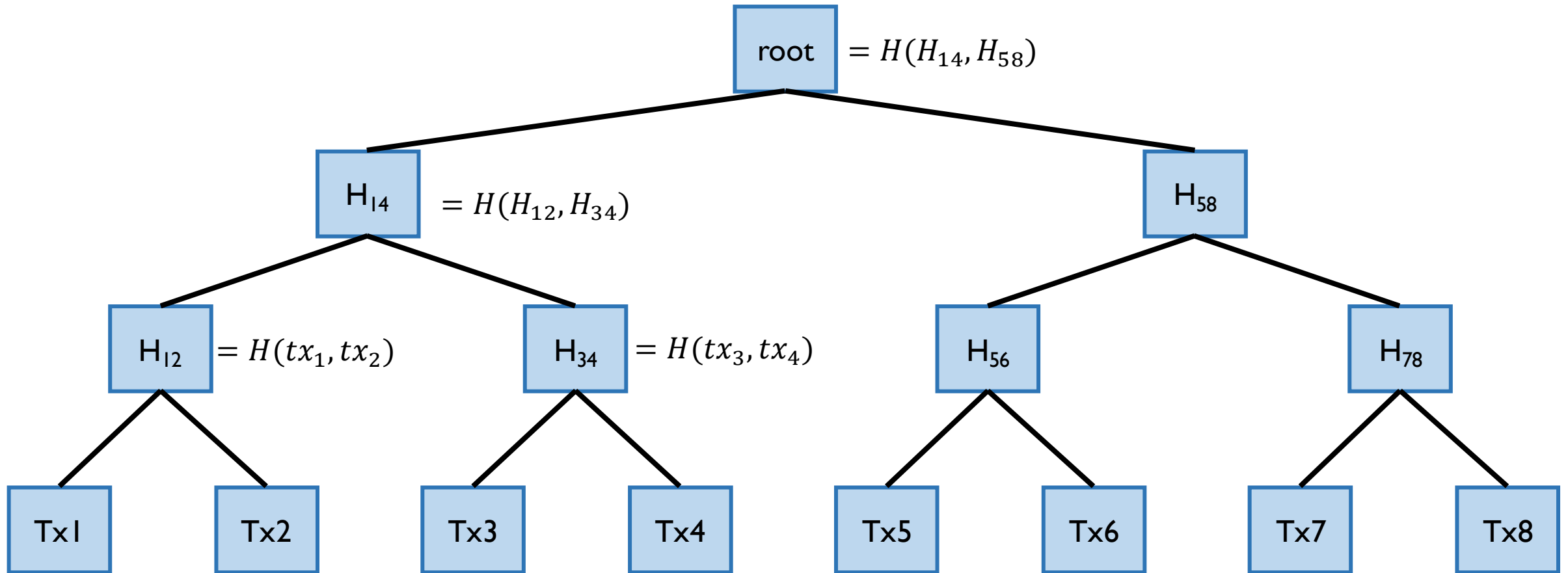


Merkle at the Singularity Summit 2007

- Key feature: *compact membership proof*
- $d$  is the digest of  $X = \{x_1, x_2, \dots, x_n\}$
- Hash chain  $|\pi_X| = O(n)$
- Hash tree  $|\pi_X| = O(\log n)$

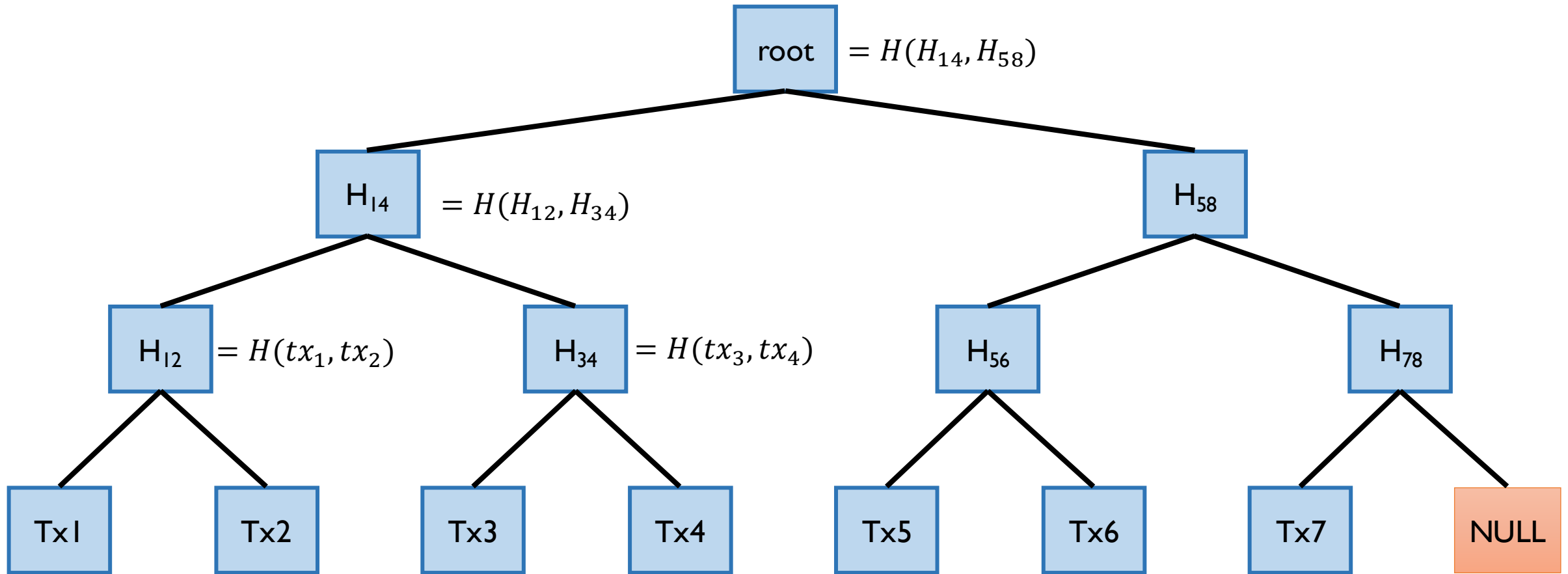
# Merkle Tree

Root is the digest to all 8 elements (e.g., transactions).



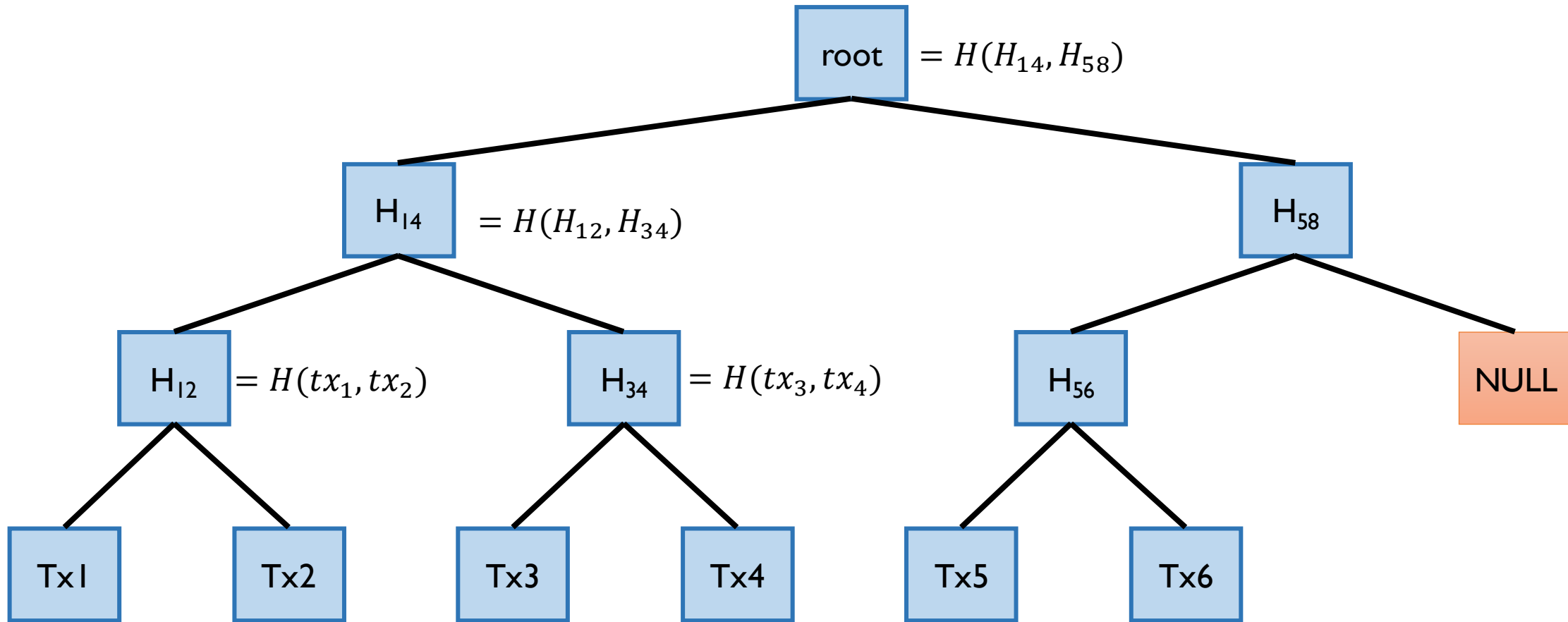
# Merkle Tree

Root is the digest to all 8 elements (e.g., transactions).



# Merkle Tree

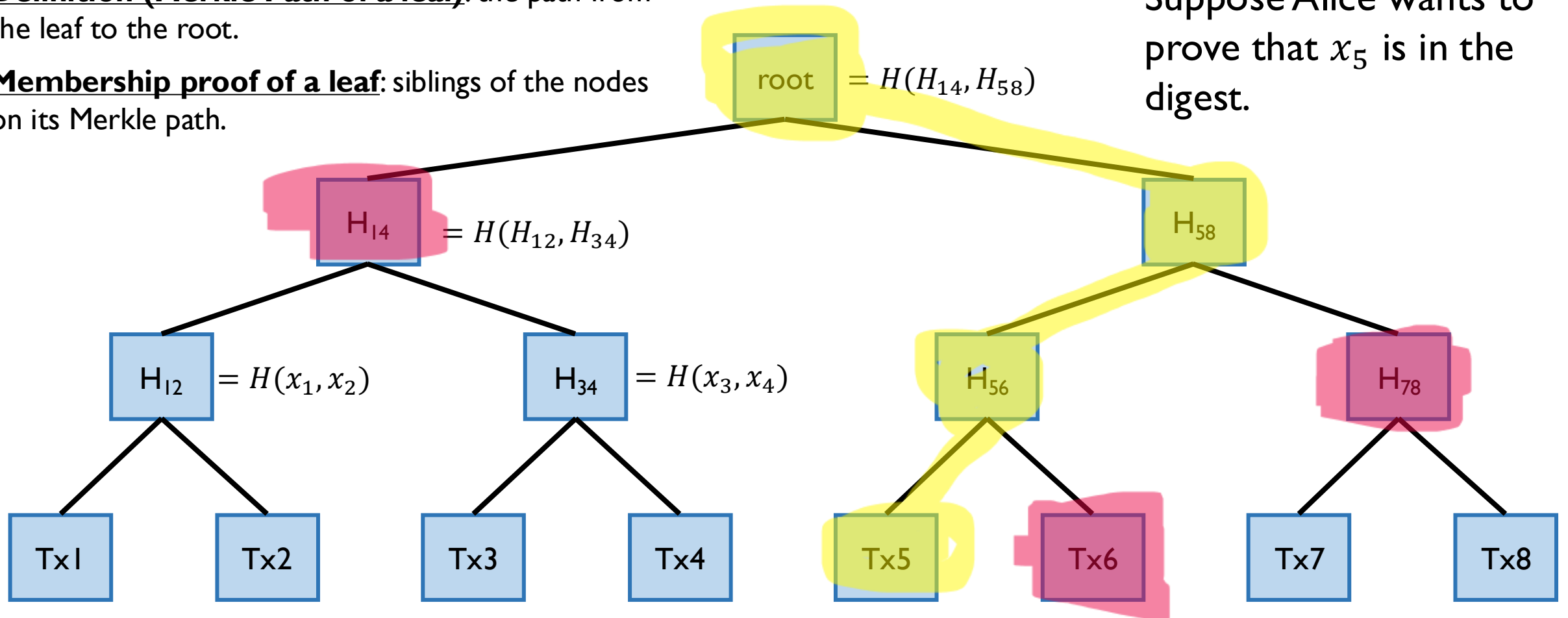
Root is the digest to all 8 elements (e.g., transactions).



# Membership proof

**Definition (Merkle Path of a leaf):** the path from the leaf to the root.

**Membership proof of a leaf:** siblings of the nodes on its Merkle path.



# Verify the membership proof for $tx_5$

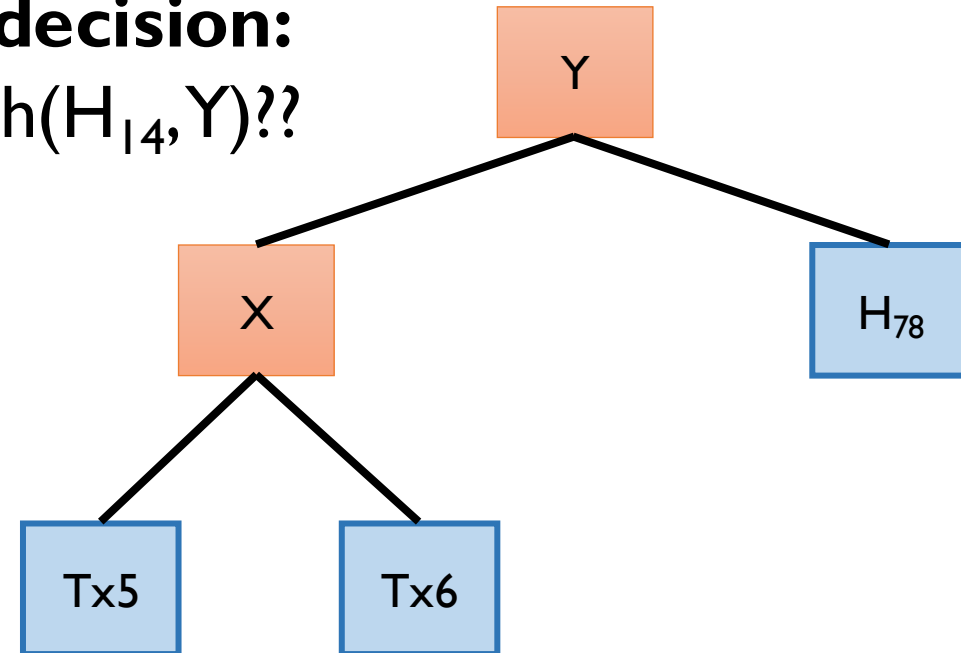
**Definition (Merkle Path of a leaf):** the path from the leaf to the root.

**Membership proof of a leaf:** siblings of the nodes on its Merkle path.

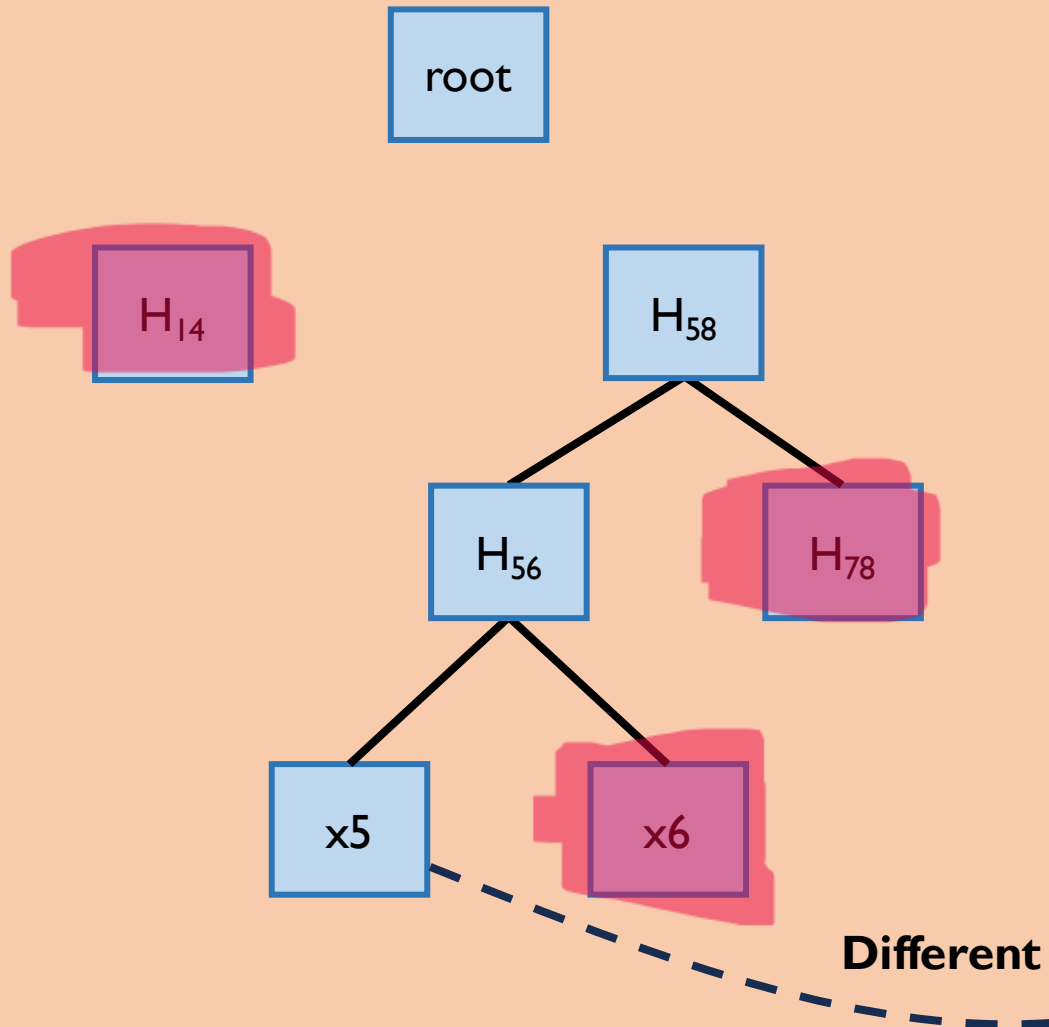
root (verifier has this)

$H_{14}$

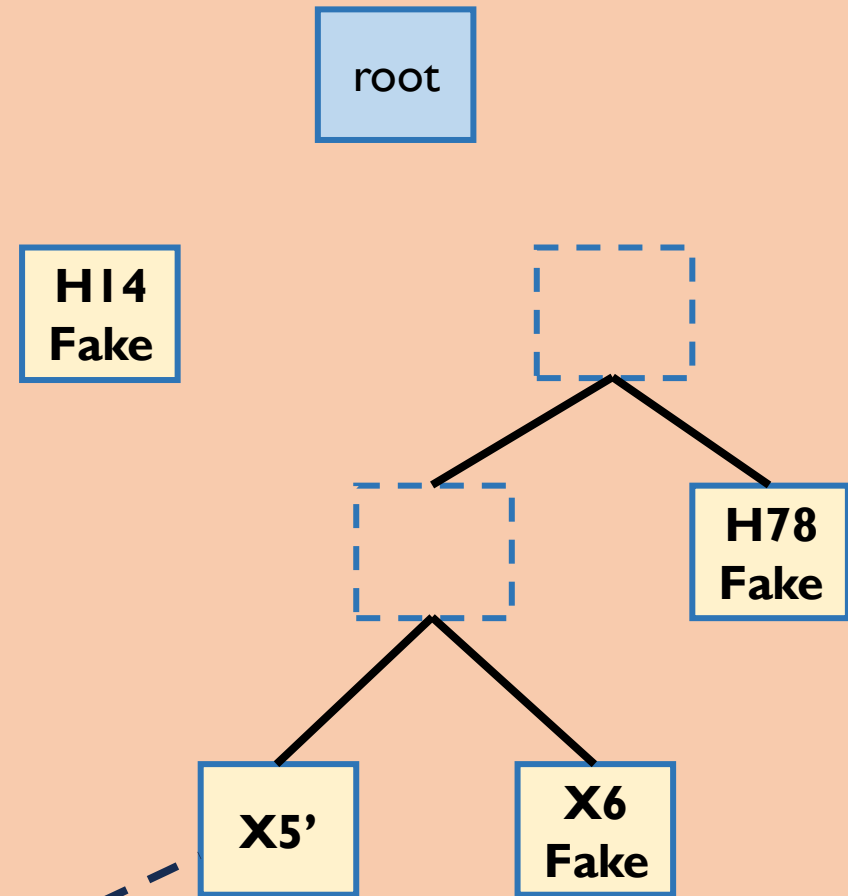
**Verifier's decision:**  
 $\text{root} = \text{Hash}(H_{14}, Y)??$



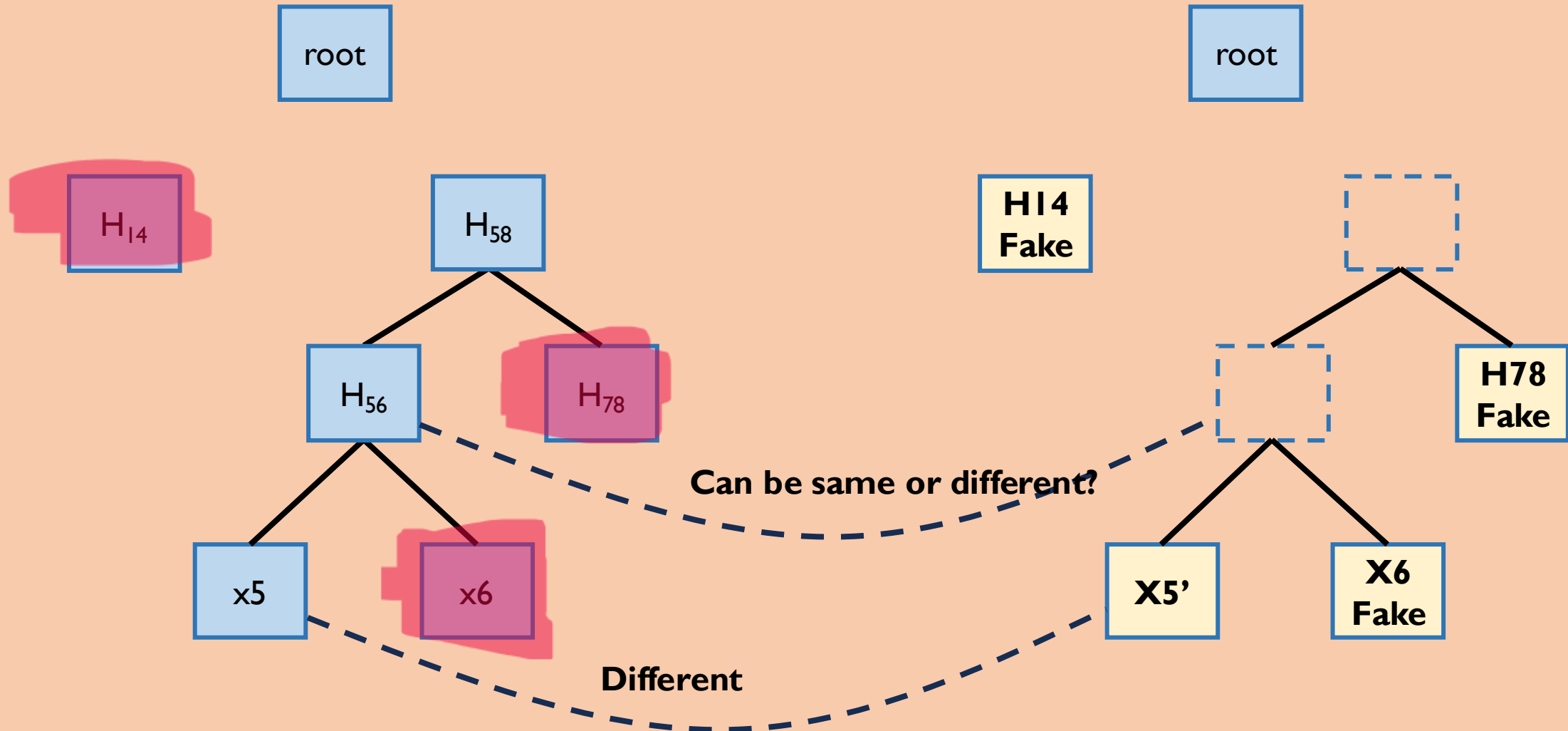
# Is it feasible to fake membership proofs?



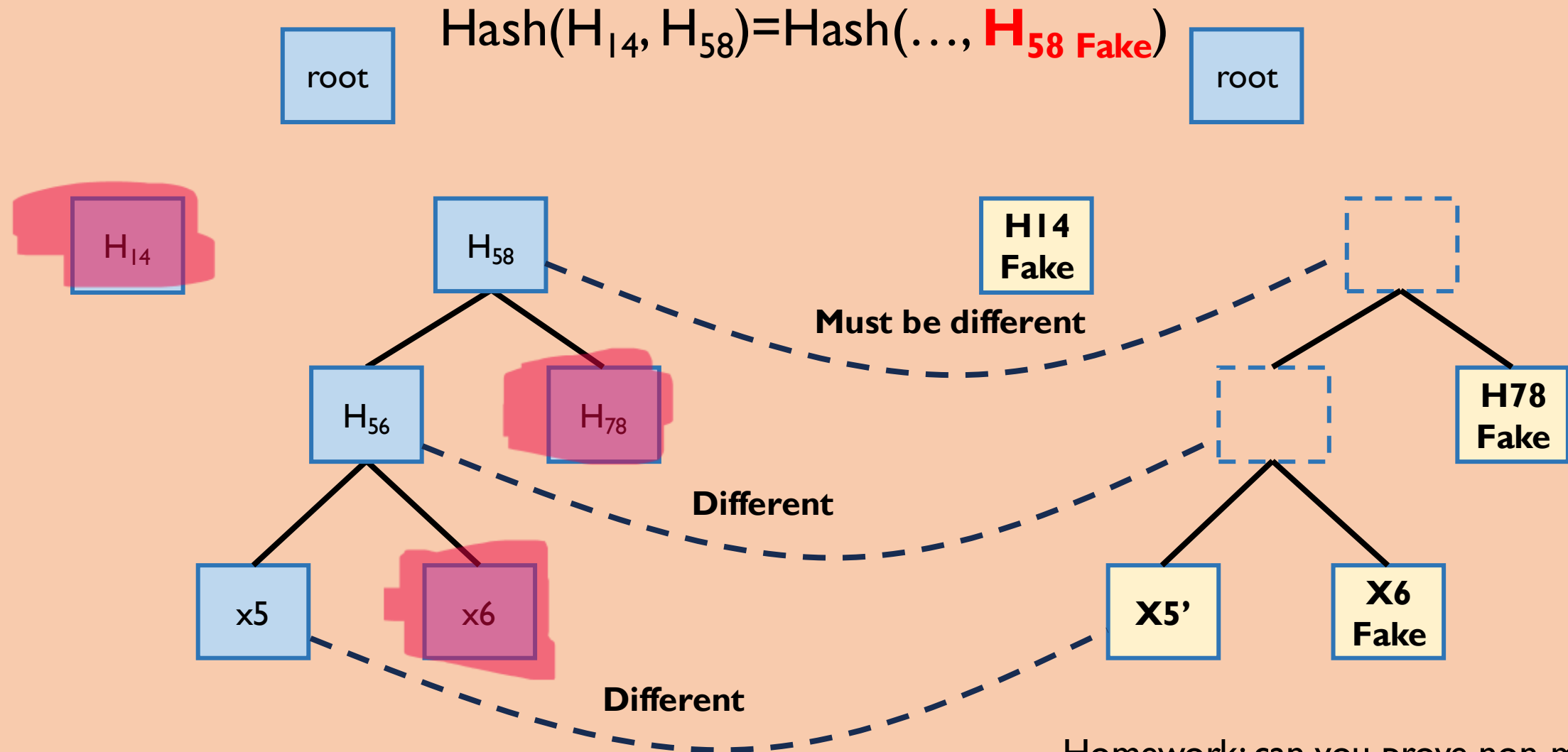
Suppose yellow nodes are a membership proof  $x_5' \neq x_5$



# Is it feasible to fake membership proofs?



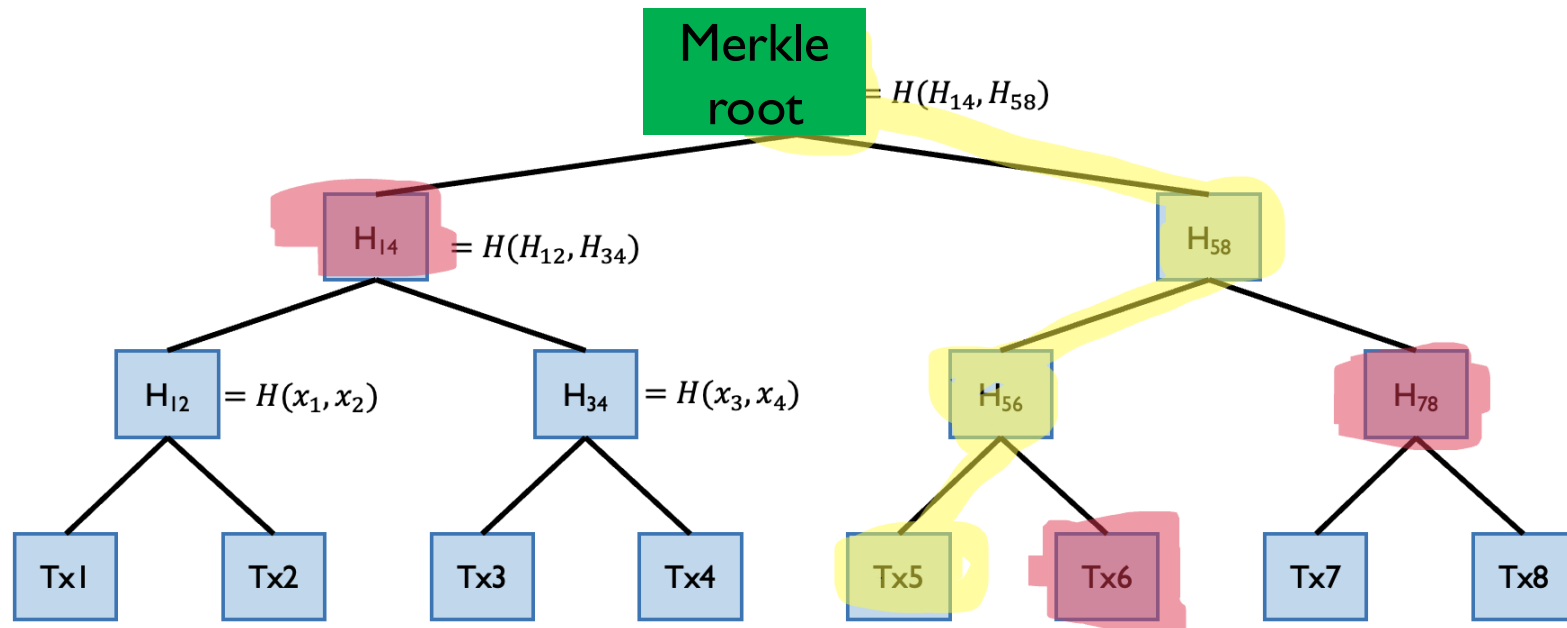
# Is it feasible to fake membership proofs?



Homework: can you prove non-membership?

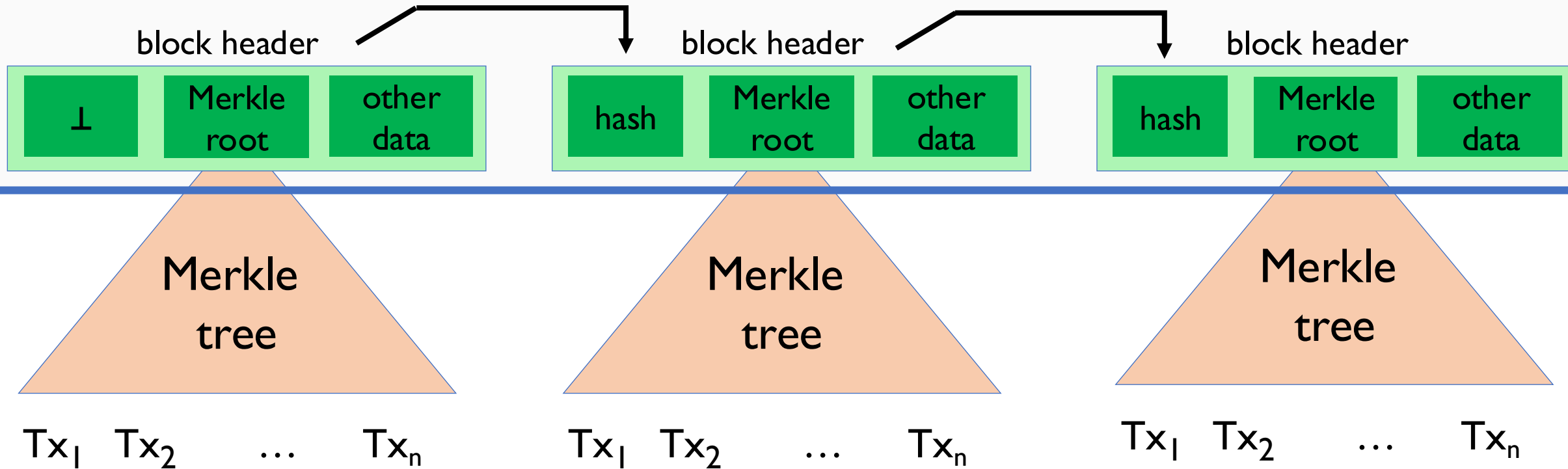
# Merkle Tree summary

- hash tree
- **root** = digest for all leaves
- log-size membership proofs
- Infeasible to fake membership proof
  - proof by contradiction
  - faking membership proof implies collision



# Recap: CRH used in blockchain data structure

blockchain

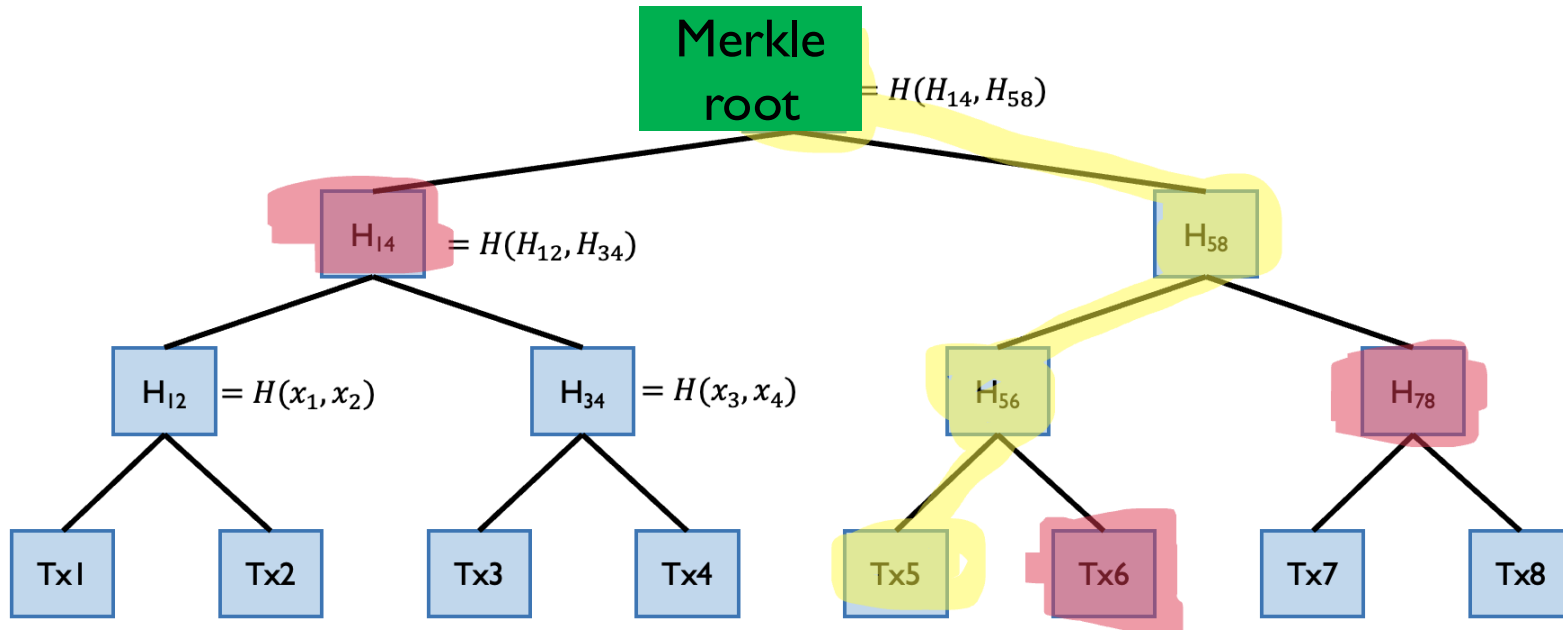


# Light client

- Merchant needs to verify if a payment transaction  $T$  is in the blockchain
- Problem: Merchant can't afford downloading the entire ledger.
- Solution: only downloads block headers
  - A header is 10x - 100x smaller than a full block.

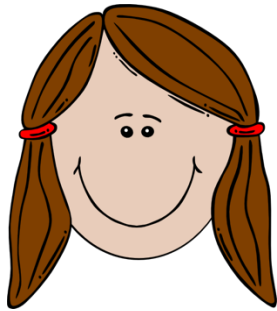


Suppose **Jitter Bus** accepts Bitcoin



\* Note that Merchant can't verify that Tx5 was indeed sent by Prover. (Users are pseudonymous on Bitcoin.) It merely verifies that Tx5 is *in* the blockchain, and therefore safe to serve the coffee.

Stores **Merkle root**



Prover

Here is a proof that my TX (Tx5) has been included in the tree:  $(Tx_5, Tx_6, H_{78}, H_{14})$

Small proof saves bandwidth!



Verifier

# Proof of Work

- What prevents double spending in Bitcoin
- What defines the authoritative ledger
- How Bitcoin miners are elected to become “block creators”

# Proof of work (a leader election method)

- To mine block, miner builds a block  $B$ 
  - $B = \text{header} || \text{transactions}$
  - $\text{header} = \dots || \text{nonce} || \dots$
- $B$  is a **valid PoW solution** if  $H(B)$  has  $k$  leading zeros
  - $k$  is **difficulty**, adjusted dynamically
  - Currently  $k \sim 80$
  - no other ways but keep trying  $N$

