Blockchain and Digital Currencies

Lecture 1

PHBS 2024 M3

Agenda

- Review of cryptographic hash functions

- Important data structures used in blockchain: Hash pointer and Merkle Tree

☑ What is this about? What is the usage?

Cryptographic Hash Functions

Hash Function: Mathematical Function with following 3 properties:

The input can be any string of any size.

It produces a fixed-size output. (say, 256-bit long)

Is efficiently computable. (say, $O(n)$ for n-bit string)

Such general hash function can be used to build hash tables, but they are not of much use in cryptocurrencies.
What we need are **cryptographic** hash functions.

Cryptographic Hash Functions

A Hash Function is **cryptographically secure** if it satisfies the following 3 **security properties**:

Property 1: Collision Resistance

Property 2: Hiding

Property 3: "Puzzle Friendliness"

☑ What is this about? What is the usage?

Cryptographic Hash Functions

A Hash Function is **cryptographically secure** if it satisfies the following 3 **security properties**:

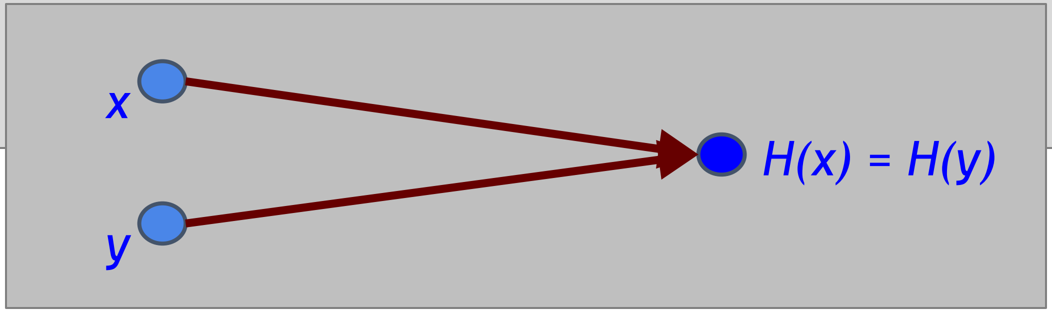
Property 1: Collision Resistance

Property 2: Hiding

Property 3: “Puzzle Friendliness”

Crypto Hash Property 1: Collision Resistance

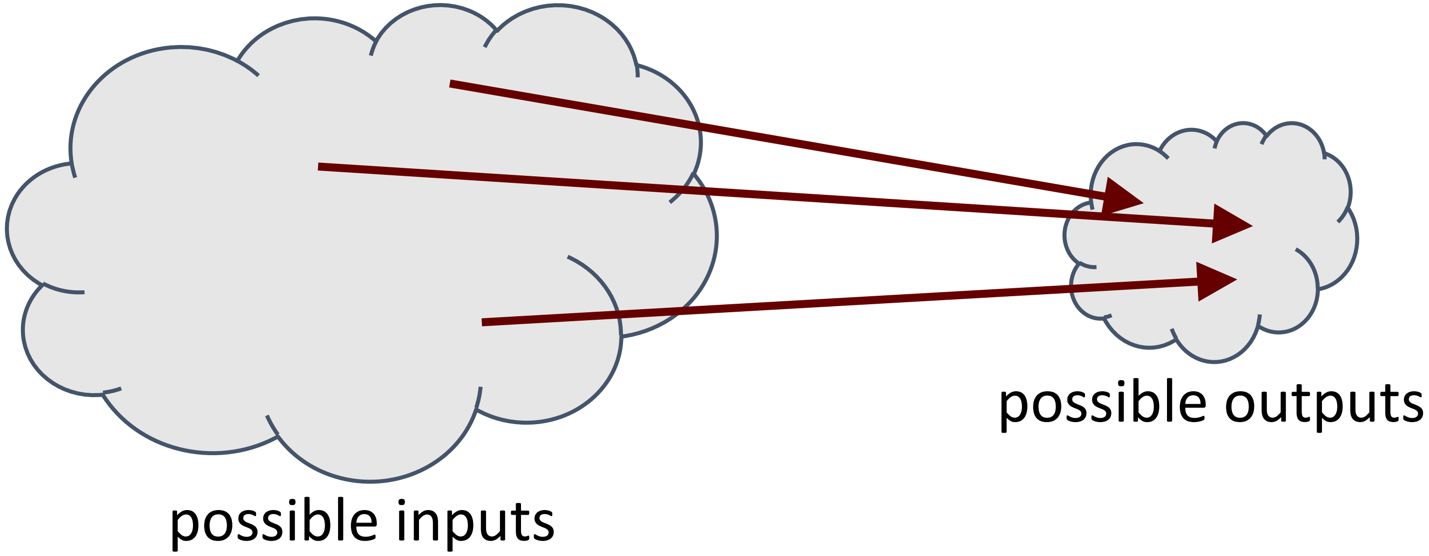
Collision Resistance: A hash function *H* is said to be **collision resistant** if it is infeasible to find two values, *x* and *y*, such that *x != y*, yet *H(x) = H(y)*.



In other words: If we have *x* and *H(x)*, we can "never" find an *y* with a matching *H(y)*, such that *H(x) = H(y)*

Collision Resistance ?! (1)

Collisions do exist ...



... but can anyone find them?

Collision Resistance ?! (2)

How to find a collision for a 256 bit string?

try 2^{130} randomly chosen inputs, there is 99.8% chance that two of them will collide

This works no matter what *H* is …, but it takes too long to matter

Given n random integers drawn from a discrete uniform distribution with range [1,d], what is the probability p(n; d) that at least two numbers are the same? (d = 365 gives the usual birthday problem.)

$$p(n;d) = \begin{cases} 1 - \prod_{k=1}^{n-1}\left(1 - \frac{k}{d}\right) & n \le d \\ 1 & n > d \end{cases}$$

$$\approx 1 - e^{-\frac{n(n-1)}{2d}}$$

$$\approx 1 - \left(\frac{d-1}{d}\right)^{\frac{n(n-1)}{2}}$$

Conversely, if n(p; d) denotes the number of random integers drawn from [1,d] to obtain a probability p that at least two numbers are the same, then

$$n(p;d) \approx \sqrt{2d \cdot \ln\left(\frac{1}{1-p}\right)}.$$

https://en.wikipedia.org/wiki/Birthday_problem

Collision Resistance ?! (3)

Q: Is there a faster way to find collisions?

A: For some possible *H*'s, yes.
For others, we don't know of one.

No *H* has been proven collision-free.

True randomness helps here!
The PRNG mechanism from the Information Security class should help you understand.
Also, improper usage of PRNG will cause easy collision.

Collision Resistance

Application: Hash as a **Message Digest**

If we know that $H(x) = H(y)$, it is safe to assume that $x = y$.

Example: To recognize a file that we saw before, just remember its hash.

This works because hash is small.

Examples: downloading software verification, comparing huge .mat files, comparing whether 2 huge Oracle tables are identical, …

Cryptographic Hash Functions

A Hash Function is **cryptographically secure** if it satisfies the following 3 **security properties**:

Property 1: Collision Resistance

Property 2: Hiding

Property 3: “Puzzle Friendliness”

Crypto Hash Property 2: Hiding

We want something like this:

Given *H(x)*, it is infeasible to find *x*.

Example:

H("heads")

H("tails")

easy to find *x*!

The value for *x* is easy to find because the distribution is not "spread out" (only two values!)

Crypto Hash Property 2: Hiding (cont)

Hiding: A hash function H is said to be **hiding** if when a secret value r is chosen from a probability distribution that has high min-entropy, then, given $H(r \,||\, x)$, it is infeasible to find x.

"$r \,||\, x$" stands for "r concatenated with x"

"High min-entropy" means that the distribution is "very spread out", so that no particular value is chosen with more than negligible probability.

☑ How to create r?

Application of Hiding Property: Commitment

Want to "seal a value in an envelope", and
"open the envelope" later.

Commit to a value, reveal it later.

Application of Hiding Property: Commitment

Commitment Scheme consists of two algorithms:

- com := commit(msg,key) takes message and secret key, and returns commitment
- verify(com,msg,key) returns true if com = commit(msg,key) and false otherwise.

We require two security properties:

- Hiding: Given com, it is infeasible to find msg.
- Binding: It is infeasible to find two pairs *(msg,key)* and *(msg',key')* s.t. *msg != msg'* and *commit(msg,key) == commit (msg',key')*.

Implementation of Commitment

- commit(msg,key) := H(key || msg)
- verify(com,msg,key) := (H(key || msg) == com)

Proof of security properties:

- Hiding: Given H(key || msg), it is infeasible to find msg.
- Binding: It is infeasible to find *msg != msg'*

  such that *H(key || msg) == H(key || msg')*

Cryptographic Hash Functions

A Hash Function is **cryptographically secure** if it satisfies the following 3 **security properties**:

Property 1: Collision Resistance

Property 2: Hiding

Property 3: "Puzzle Friendliness"

Crypto Hash Property 3: "Puzzle Friendliness"

Puzzle Friendliness: A hash function H is said to be **puzzle friendly** if for every possible n-bit output value y, if k is chosen from a distribution with high min-entropy, then it is infeasible to find x such that $H(k \mid\mid x) = y$, in time significantly less than 2^n.

If a hash function is puzzle friendly, then there is no solving strategy for this type of puzzle that is much better than trying random values of x.

Bitcoin mining is just such a computational puzzle.

Block 649095: Summary

首页 / 块 - 0000000000000000000a08a5b0b606ece83630f325981ee3642d05202f3346be

摘要

高度	649,095	版本	0x20a00000	块哈希	0000000000000000000a08a5b0b606ece83630f325981ee3642d05202f3346be
确认数	6	难度	28.05 T / 17.35 T	前一个块	0000000000000000000ad3e9daa032f43d720b0d8463ca4f3b6702d7e1520f91
大小	1,391,442 Bytes	Bits	0x17103a12	后一个块	00000000000000000004331175a02cfdf83a7f133fc0f114b3d264c58f0b6c2f
Stripped Size	867,093 Bytes	Nonce	0x83b77482	Merkle Root	c17d60c79bb984a52b82e6c380d702b38cd6b560ab718d7b12a646fad98ad165
Weight	3,992,721	播报方	BTC.com		
数量	2,074	时间	2020-09-20 00:29:41	其它区块浏览器	BLOCKCHAIR

第649095区块：

https://www.blockchain.com
https://www.blockchain.com/btc/block/0000000000000000000a08a5b0b606ece83630f325981ee3642d05202f3346be

Hash	0000000000000000000a08a5b0b606ece83630f325981ee3642d05202f3346be
Confirmations	7
Timestamp	2020-09-20 00:29
Height	649095
Miner	BTC.com
Number of Transactions	2,074
Difficulty	17,345,997,805,929.09
Merkle root	c17d60c79bb984a52b82e6c380d702b38cd6b560ab718d7b12a646fad98ad165
Version	0x20a00000
Bits	386,939,410
Weight	3,992,721 WU
Size	1,391,442 bytes
Nonce	2,209,838,210
Transaction Volume	8664.40757568 BTC
Block Reward	6.25000000 BTC
Fee Reward	0.48518918 BTC

Qualified （Valid） Blocks

- First, for a block to be called qualified, it must meet certain conditions
- For the Bitcoin blockchain, this condition is proof of workload, i.e., a certain amount of computing power must be spent over time to find a qualified block
- The quantitative metrics for qualifying are Difficulties

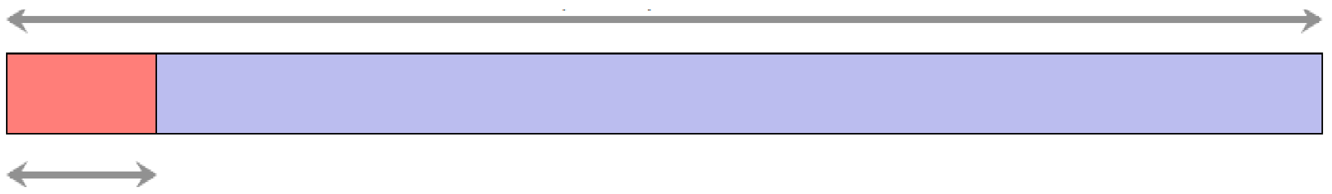
The number of bits in the block's hash that starts with a zero - the more bits, the more work

- 0000000000000000000ed997696ad0850e3daa08dfb514a2764444055a1270ae
- 0000000001000000000ed997696ad0850e3daa08dfb514a2764444055a1270ae
- 0100000000000000000000000000000000000000000000000000000000000000000
  0000000000000000000000

Difficulty 17,345,997,805,929.09

Difficulty Illustrations

0000…0000 Range of all hash values from smallest to largest FFFF…FFFF



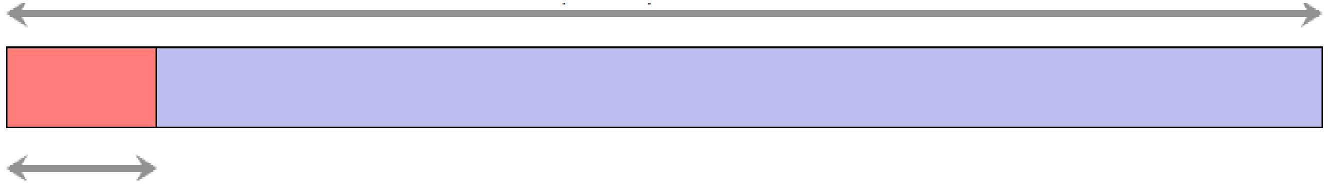
Area that meets the conditions

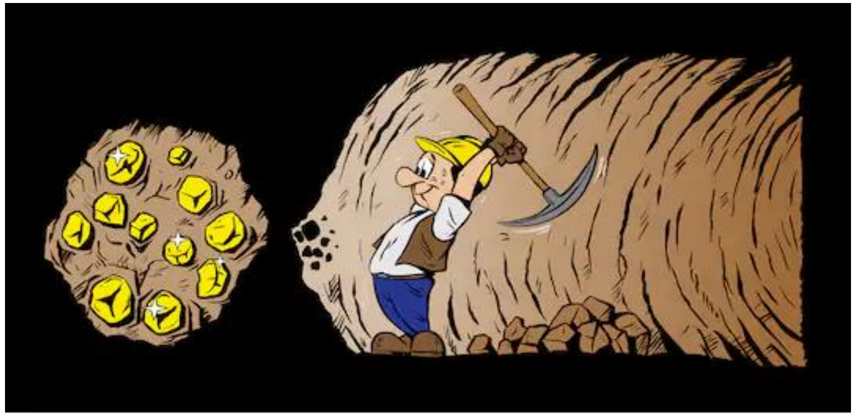
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Mining Qualified Blocks

- The more bits in the hash value of a block that starts with a zero, the higher the workload and the corresponding difficulty.
- Finding the qualified hash value, that is, finding the qualified block, is very difficult and requires a lot of time and energy, so it is called mining

Difficult to solve but easy to verify.





Hash Pointers

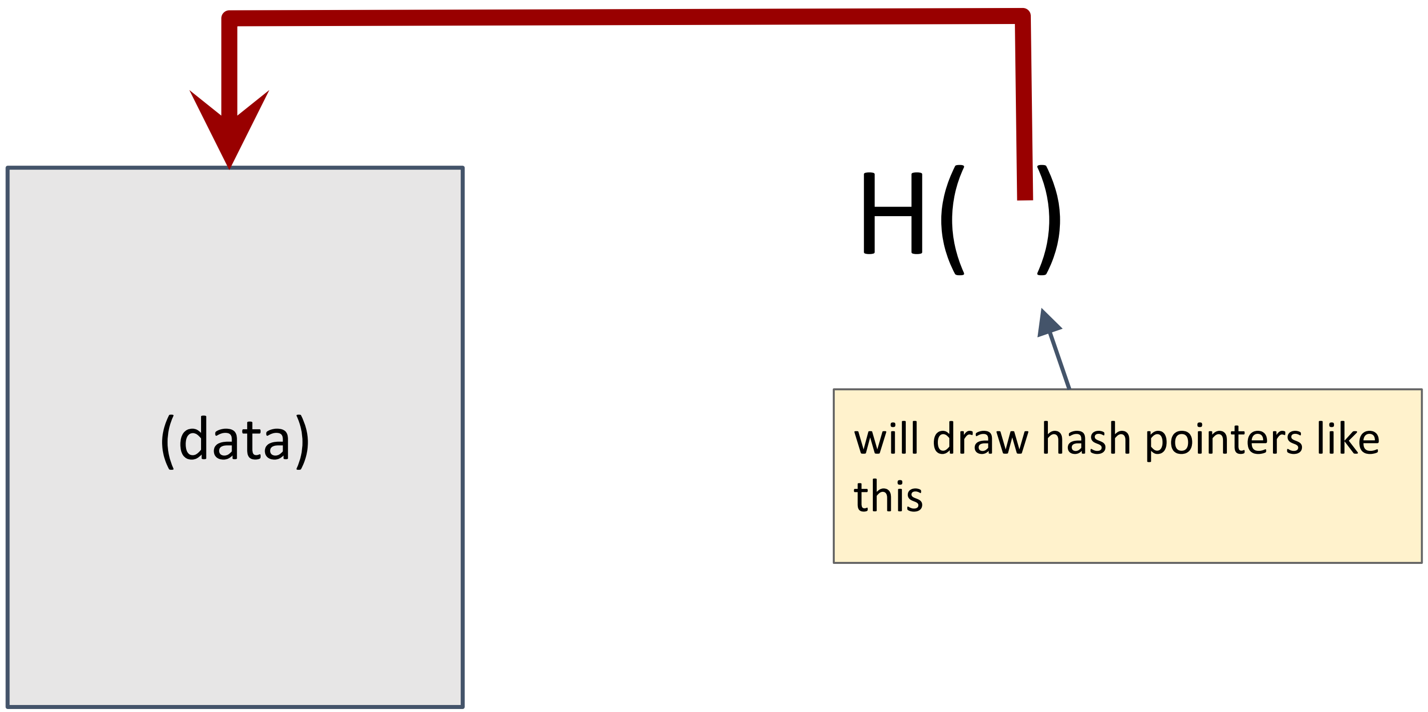
Hash Pointer is:

- pointer to where some info is stored, and
- (cryptographic) hash of the info (Typically SHA-256 is used, where SHA stands for Secure Hash Algorithm)

Given a Hash Pointer, we can

- ask to get the info back, and
- verify that it hasn't changed

Hash Pointers

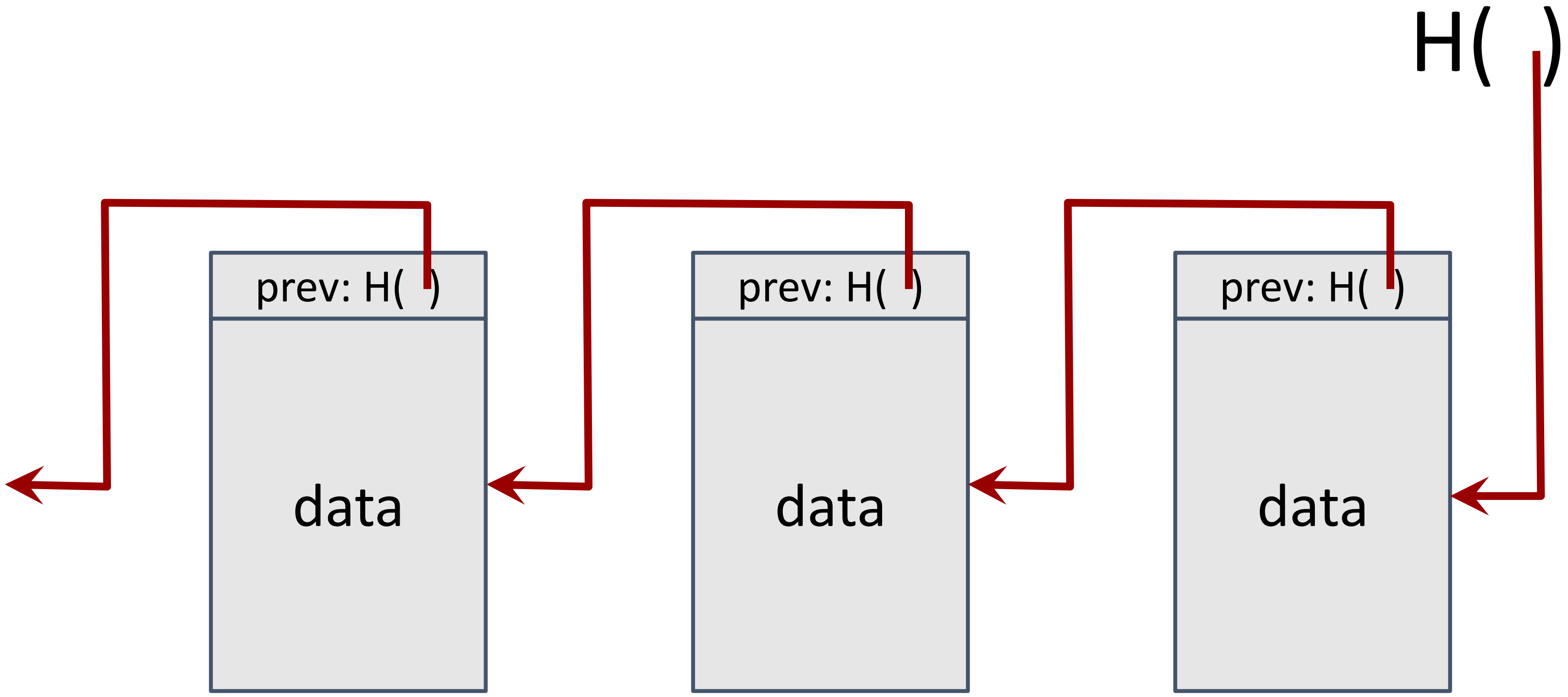


Hash Pointers

Key Idea:

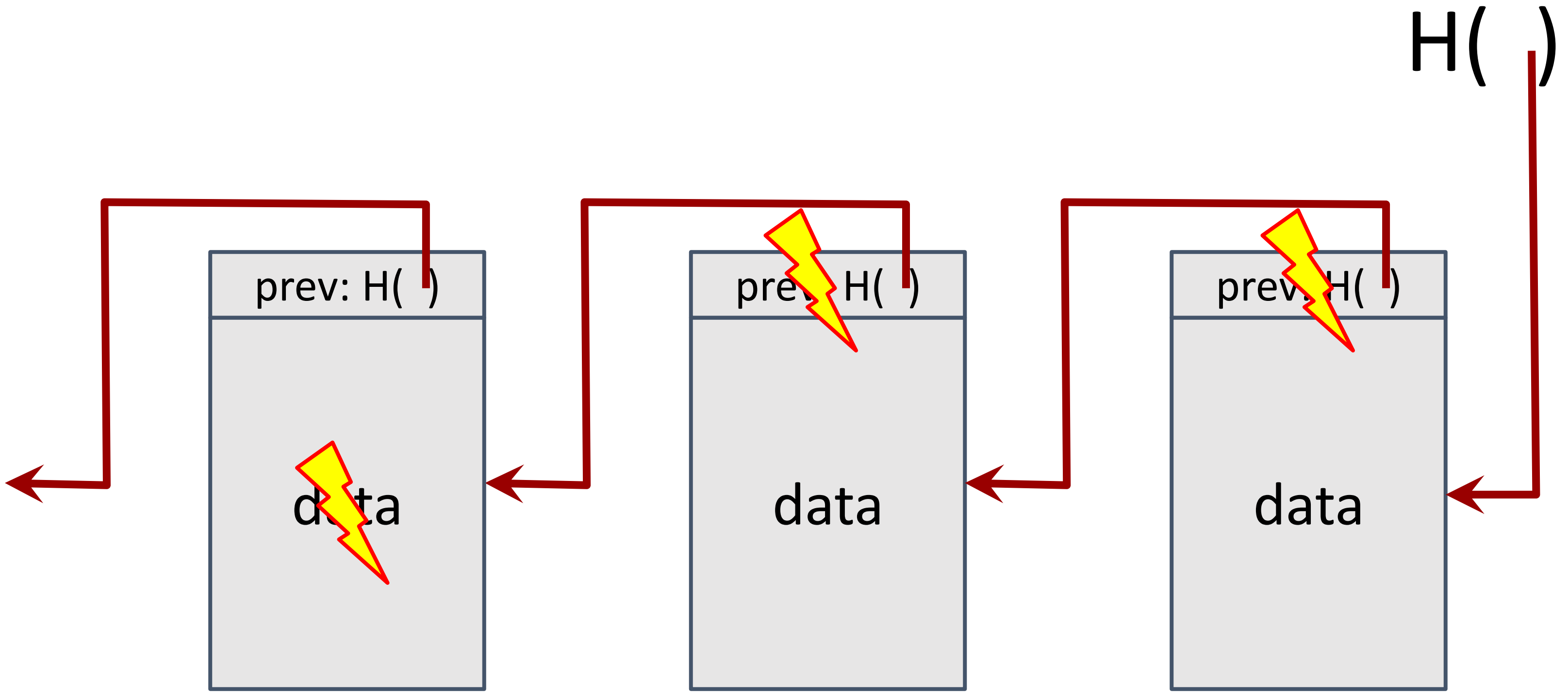
Build data structures with hash pointers.

Linked List with Hash Pointers: “Block Chain”



use case: tamper-evident log

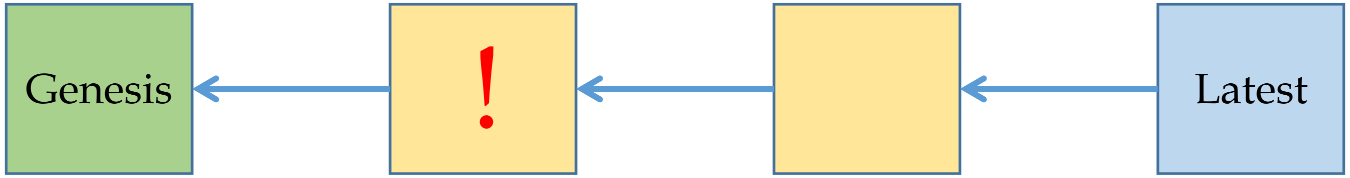
Detecting Tampering in Block Chains



use case: tamper-evident log

Blockchain is Tamper Resistant

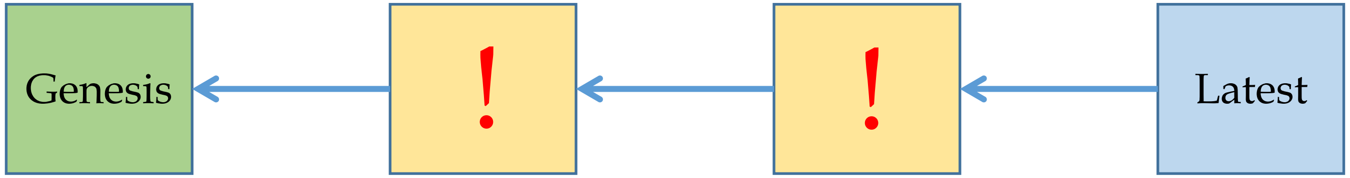
- Once a transaction has been recorded inside a blockchain, it is very difficult to change it.
- If some transaction gets changed, then the hash of the block and the hash pointer contained in the following block won’t match.
- It is easy to identify where the modification happens.



Hash	183F	3T7A	568A	0V67
Previous Hash	0000	183F	7B29	568A

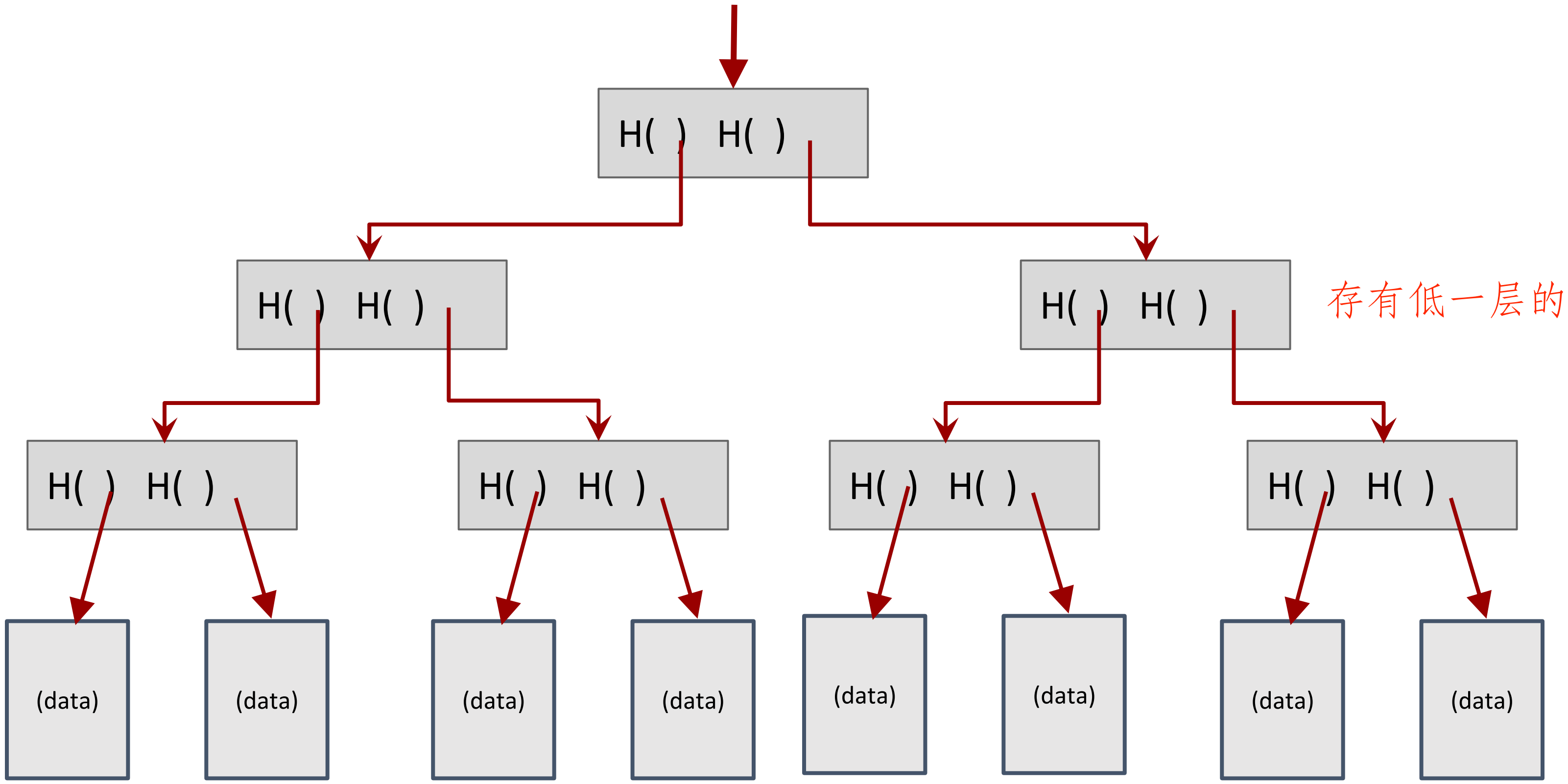
Chained Reactions

- A change of hash pointer will cause the hash of the whole block to change.
- In order to make the blockchain valid again, all following blocks of the downstream of the blockchain have to be altered.
- In reality, technically infeasible!



Hash	183F	3T7A	9527	0V67
Previous Hash	0000	183F	3T7A	568A

Binary Trees with Hash Pointers: “Merkle Tree”



存有低一层的 Hash point 值

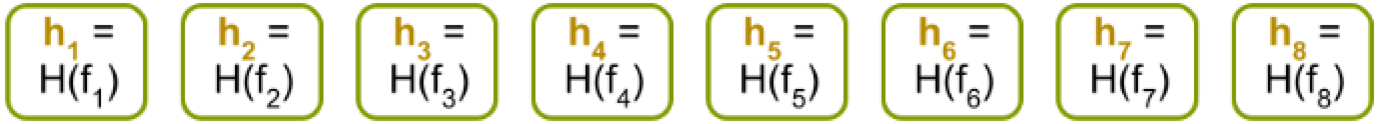
Used in file systems (IPFS, Btrfs, ZFS), BitTorrent, Apache Wave, Git, various backup systems, Bitcoin, Ethereum, and database systems.

Merkle Tree Example and Construction

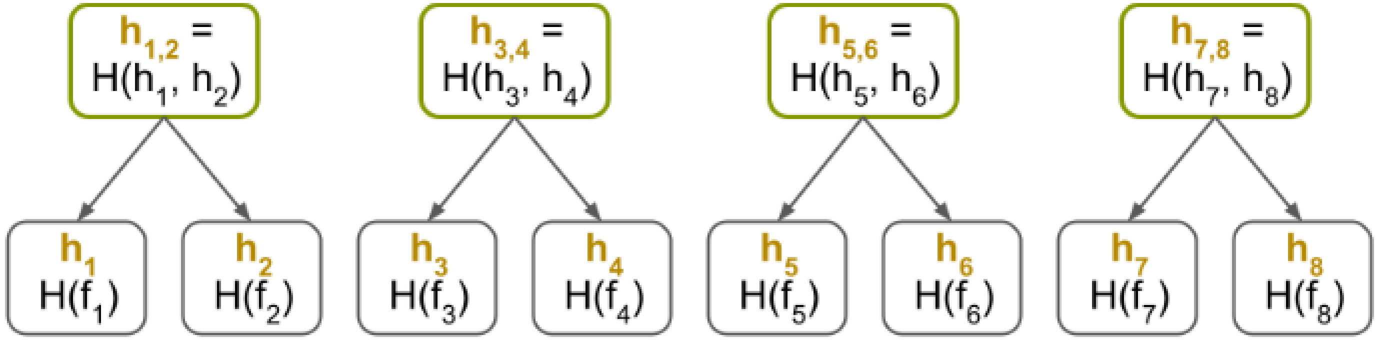
Suppose you have $n = 8$ files, denoted by $(f_1, f_2, \ldots, f_8)$, you have a collision-resistant hash function H and you want to have some fun.

You start by hashing each file as $h_i = H(f_i)$:

文件作为最底层叶子节点
那如果不是偶数个文件呢?



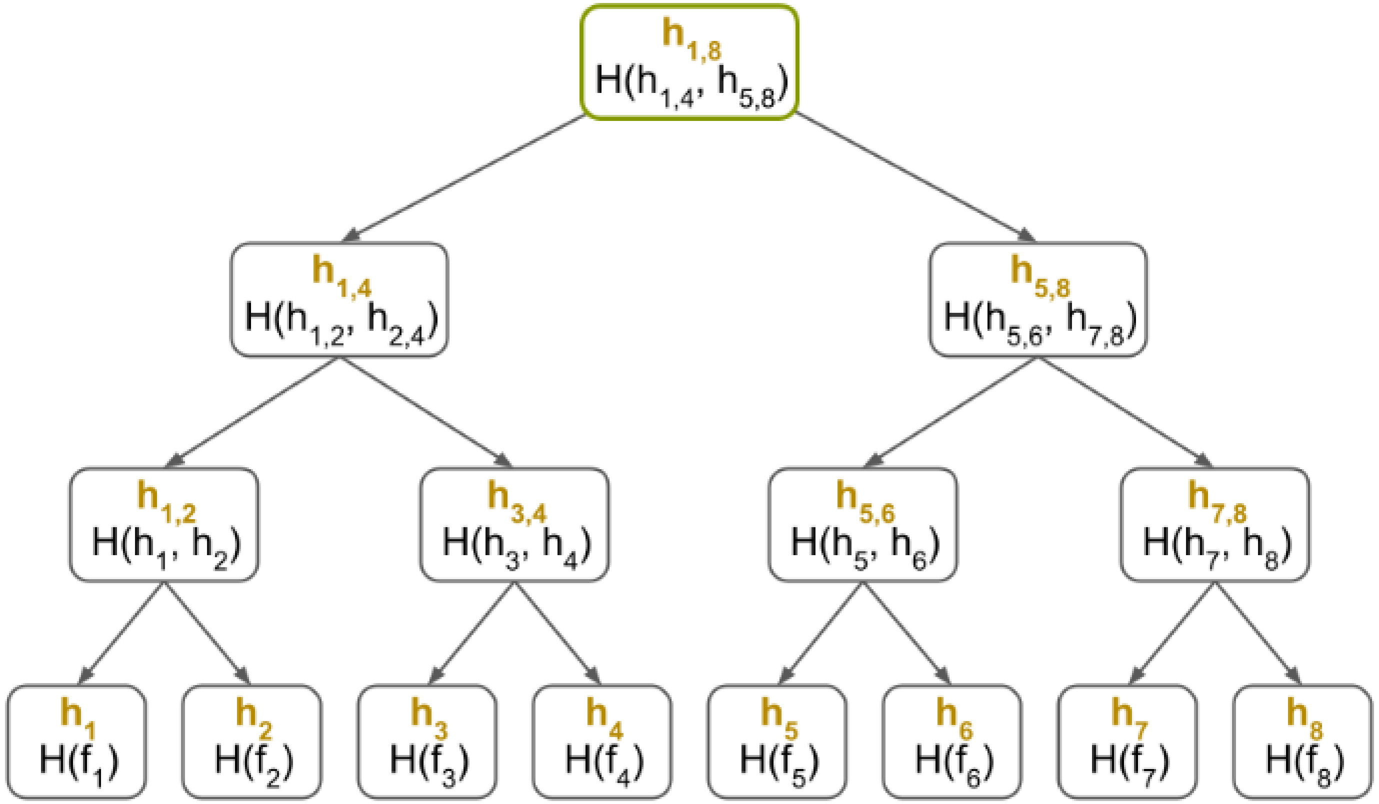
You could have a bit more fun by continuing to hash every two adjacent hashes:



Merkle Tree Example and Construction

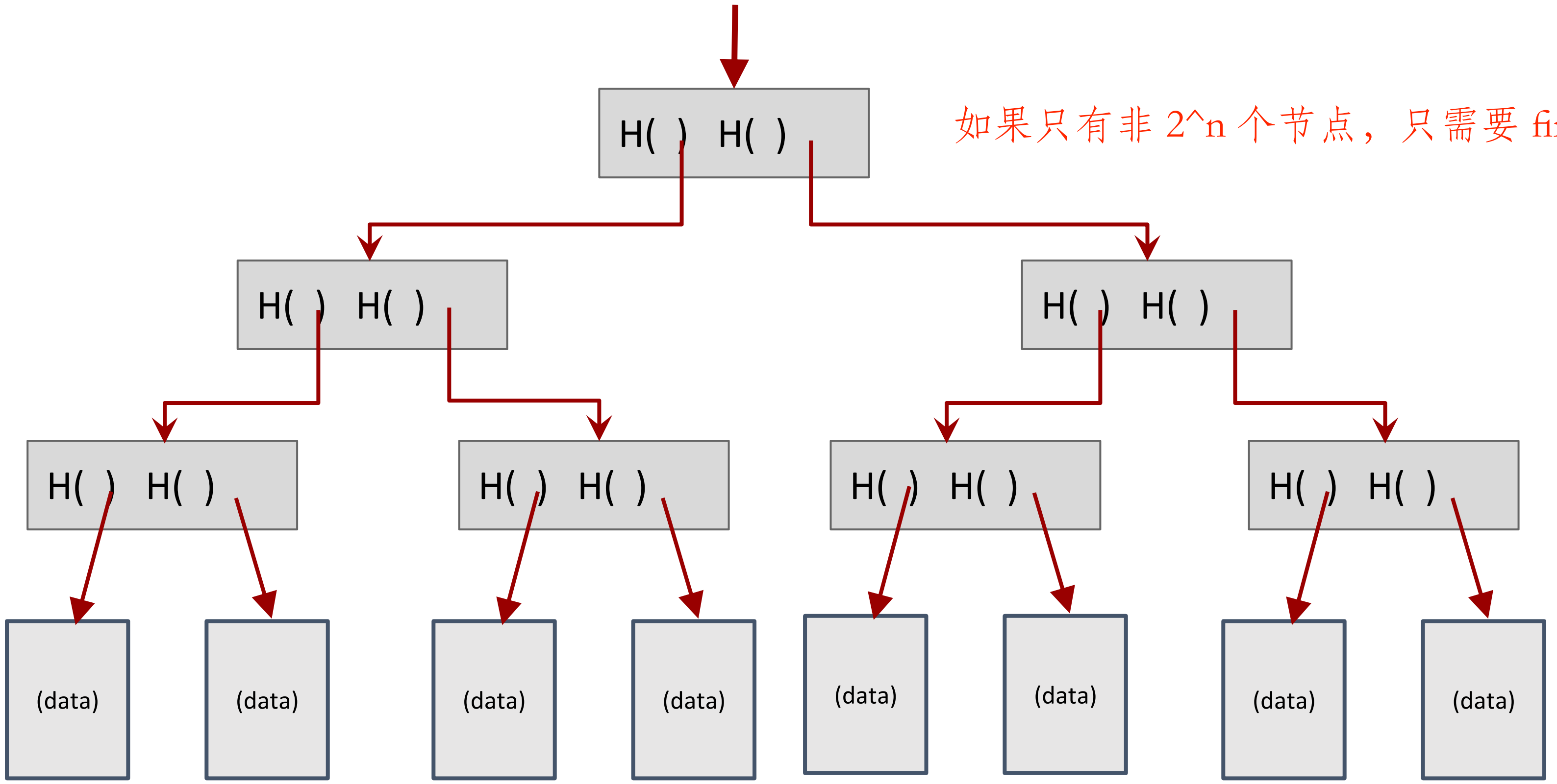
In the end, you only live once, so you'd better hash these last two hashes as $h_{1,8} = H(h_{1,4}, h_{4,8})$:

多层 Hash 结构，每一层都是对 Concat 后再 Hash



根节点的值不仅依赖于文件的内容，还依赖于文件的顺序。
因此，根节点就可以看作这些文件的数字签名，作为唯一标识

Binary Trees with Hash Pointers: “Merkle Tree”

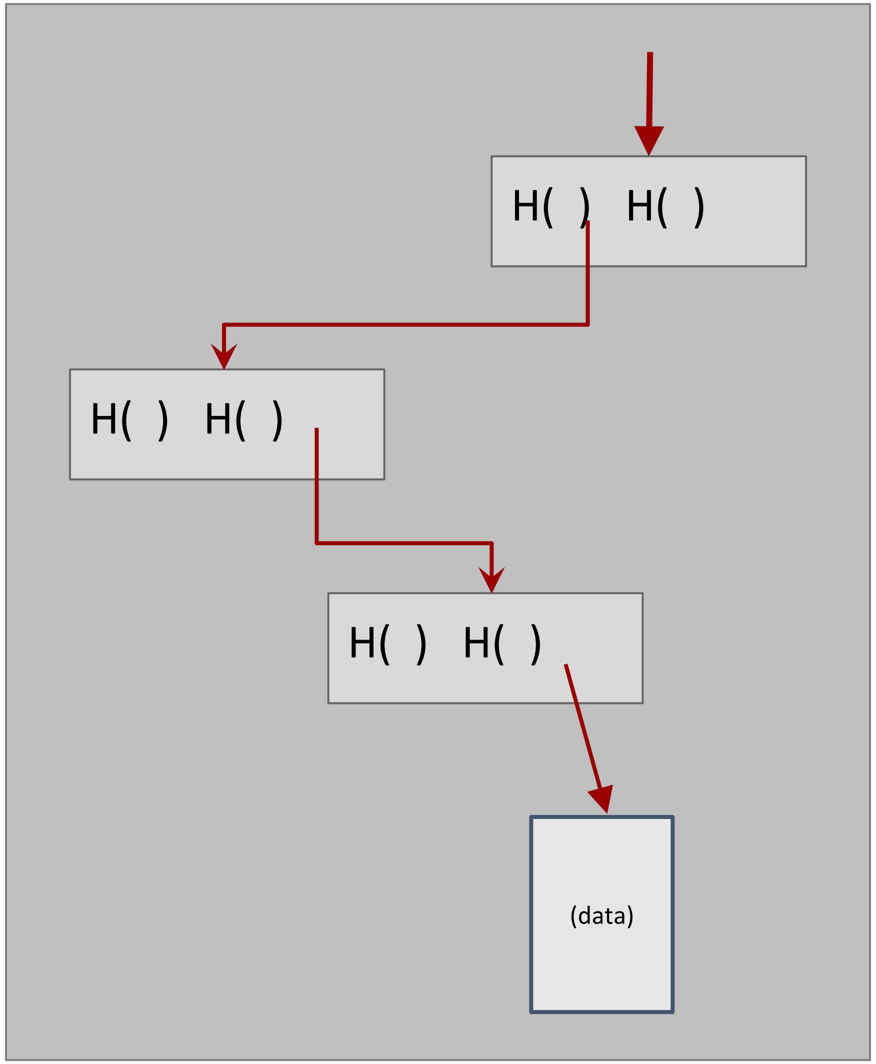


如果只有非 2^n 个节点，只需要 first fill 就可以了

Used in file systems (IPFS, Btrfs, ZFS), BitTorrent, Apache Wave, Git, various backup systems, Bitcoin, Ethereum, and database systems.

在 Block Chain 的视角下，每一个 data 是各个合约的集合。

Proving Membership in a Merkle Tree

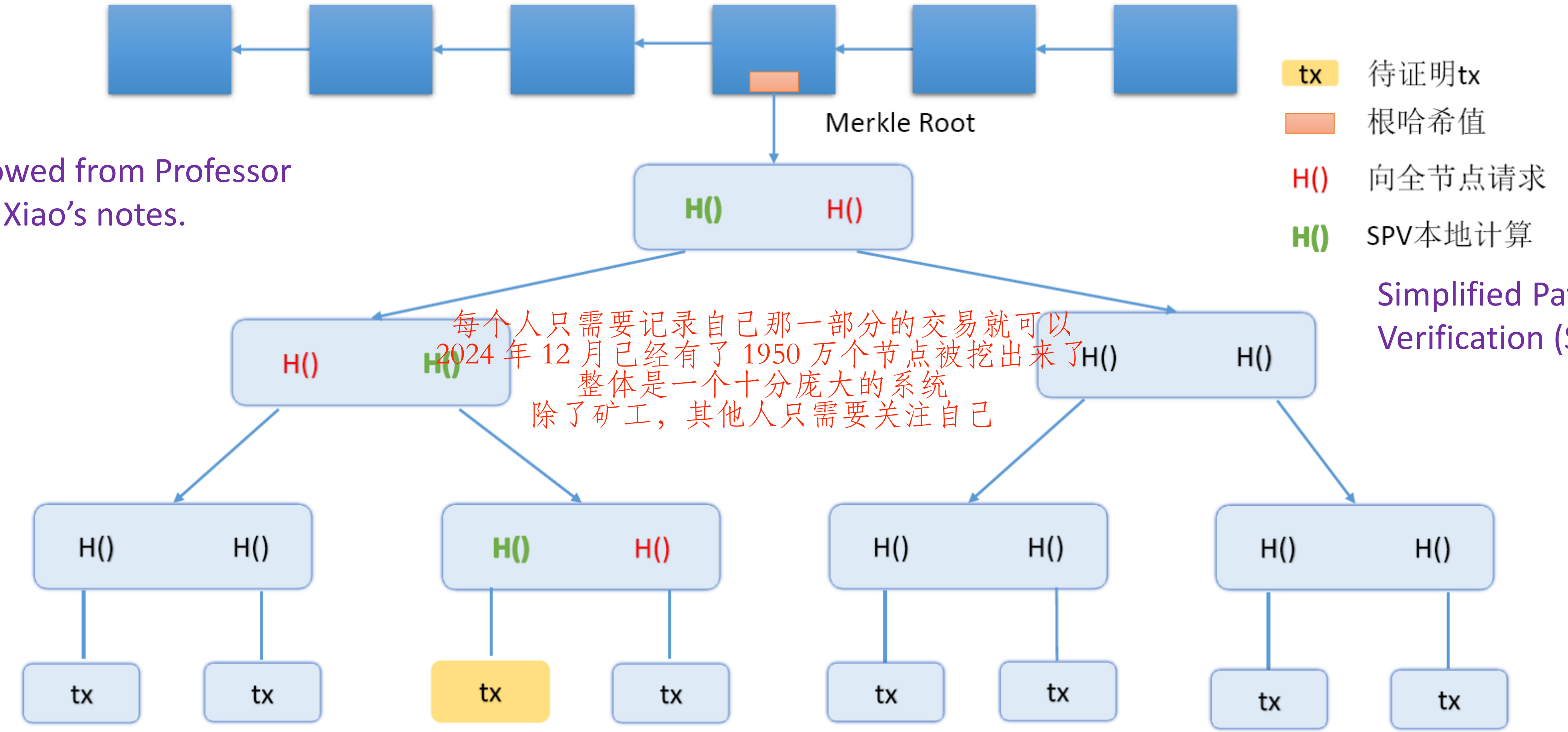


Single branches of the tree can be downloaded at a time.

To prove that a data block is included in the tree only requires showing blocks in the path from that data block to the root.

Full Node vs Light Node 证明交易节点存在的计算

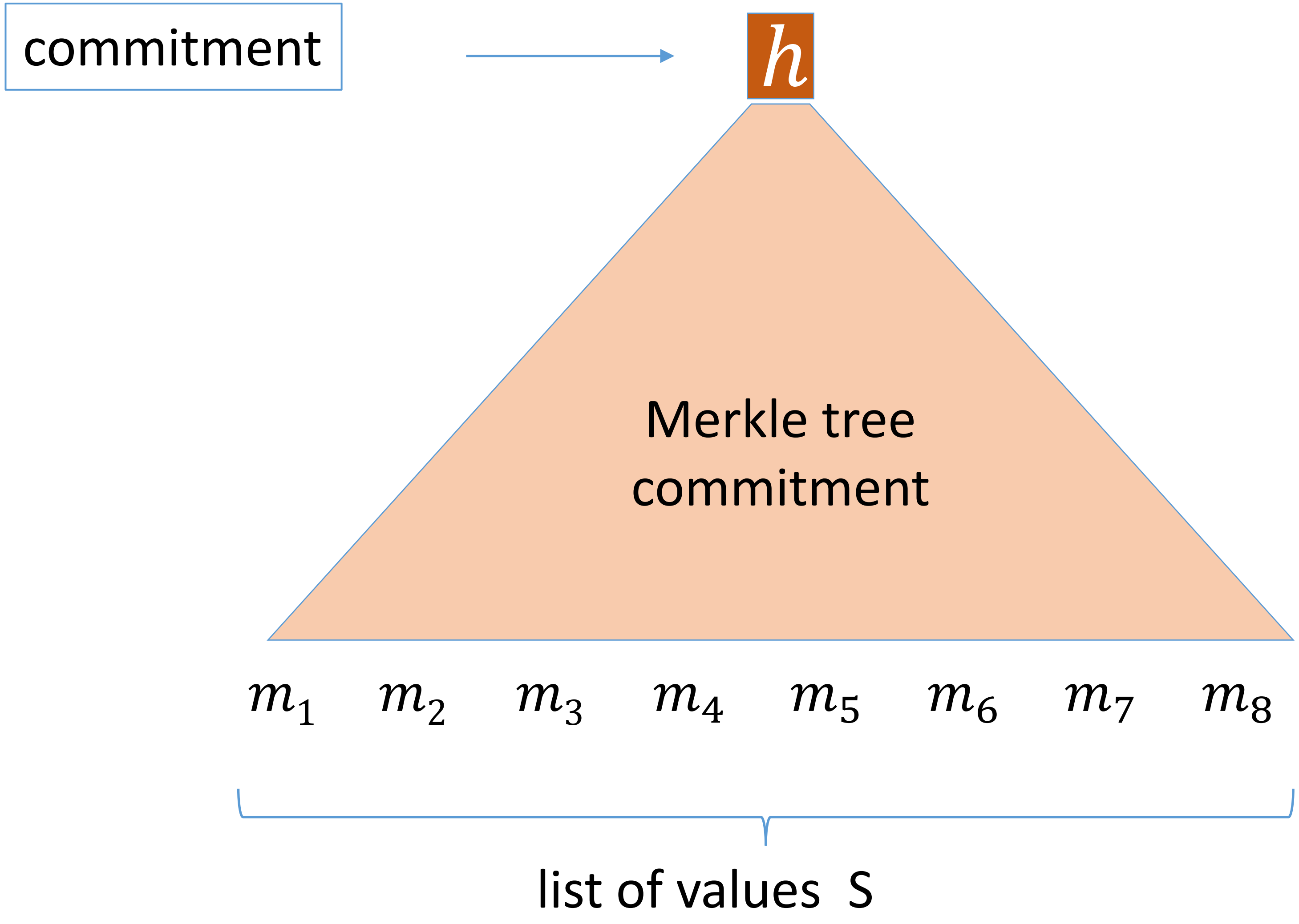
Borrowed from Professor
Zhen Xiao’s notes.



每个人只需要记录自己那一部分的交易就可以
2024 年 12 月已经有了 1950 万个节点被挖出来了
整体是一个十分庞大的系统
除了矿工，其他人只需要关注自己

Simplified Payment
Verification (SPV)

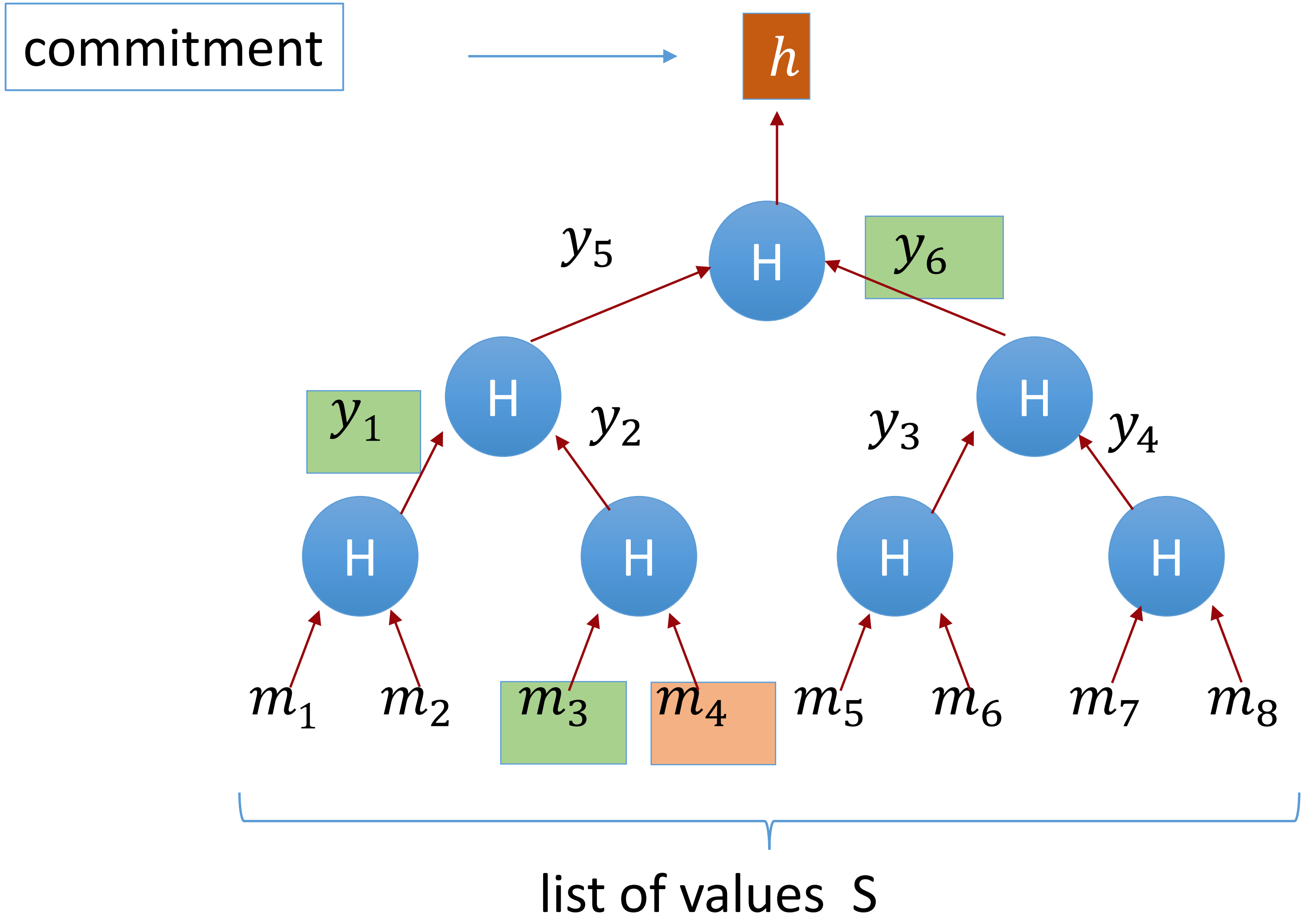
Merkle tree (Merkle 1989)



Goal:

- commit to list S of size n
- Later prove $S[i] = m_i$

Merkle tree (Merkle 1989) [simplified]



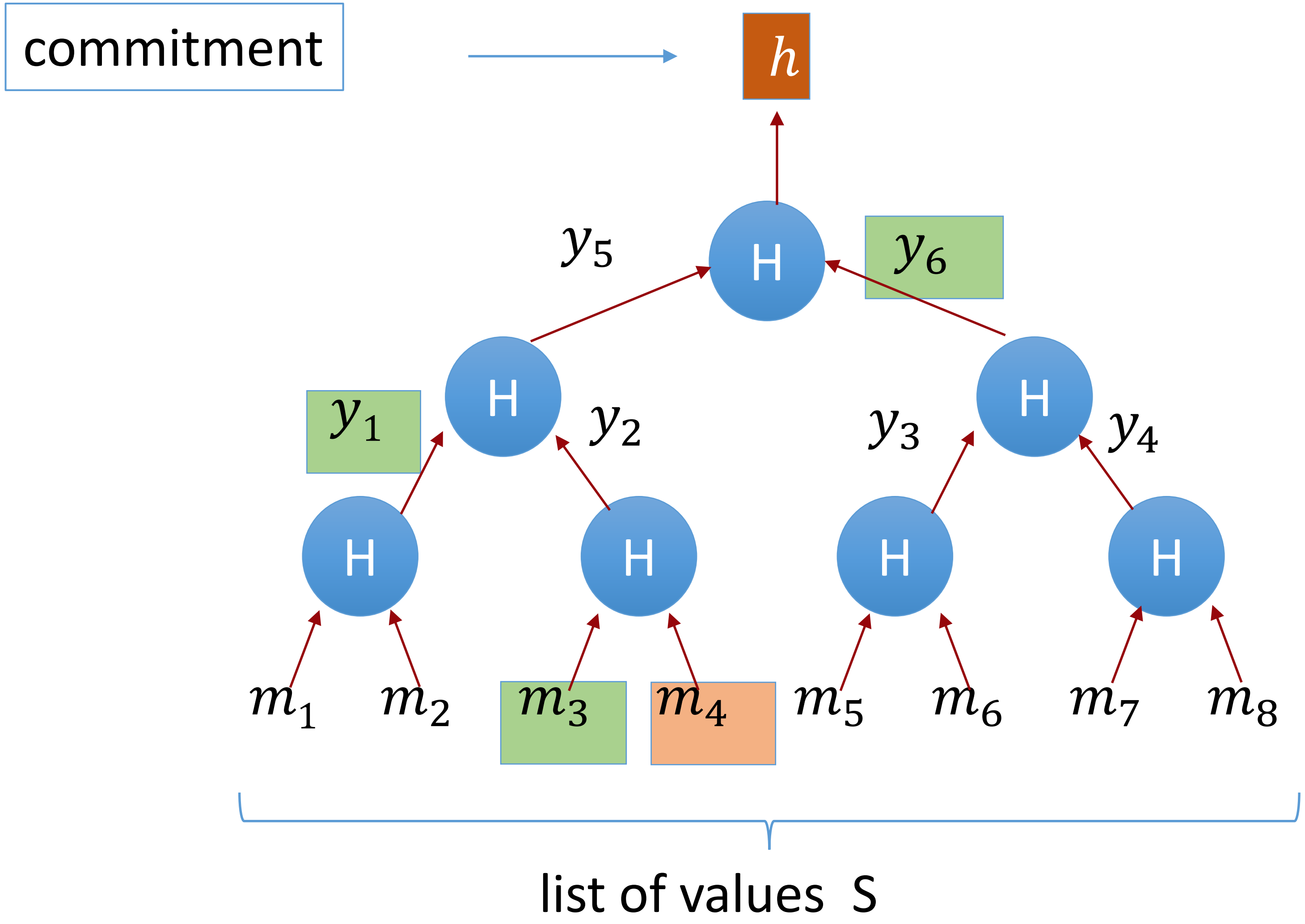
Goal:
- commit to list S of size n
- Later prove $S[i] = m_i$

To prove $S[4] = m_4$,

proof $\pi = (m_3, y_1, y_6)$

length of proof: $\log_2 n$

Merkle tree (Merkle 1989) [simplified]



To prove $S[4] = m_4$,

proof $\pi = (m_3, y_1, y_6)$

Do the following in sequence:

$y_2 \leftarrow H(m_3, m_4)$

$y_5 \leftarrow H(y_1, y_2)$

$h' \leftarrow H(y_5, y_6)$

accept if $h = h'$

Benefits of Merkle Trees

Tree holds many items . . .

. . . but just need to remember the root hash

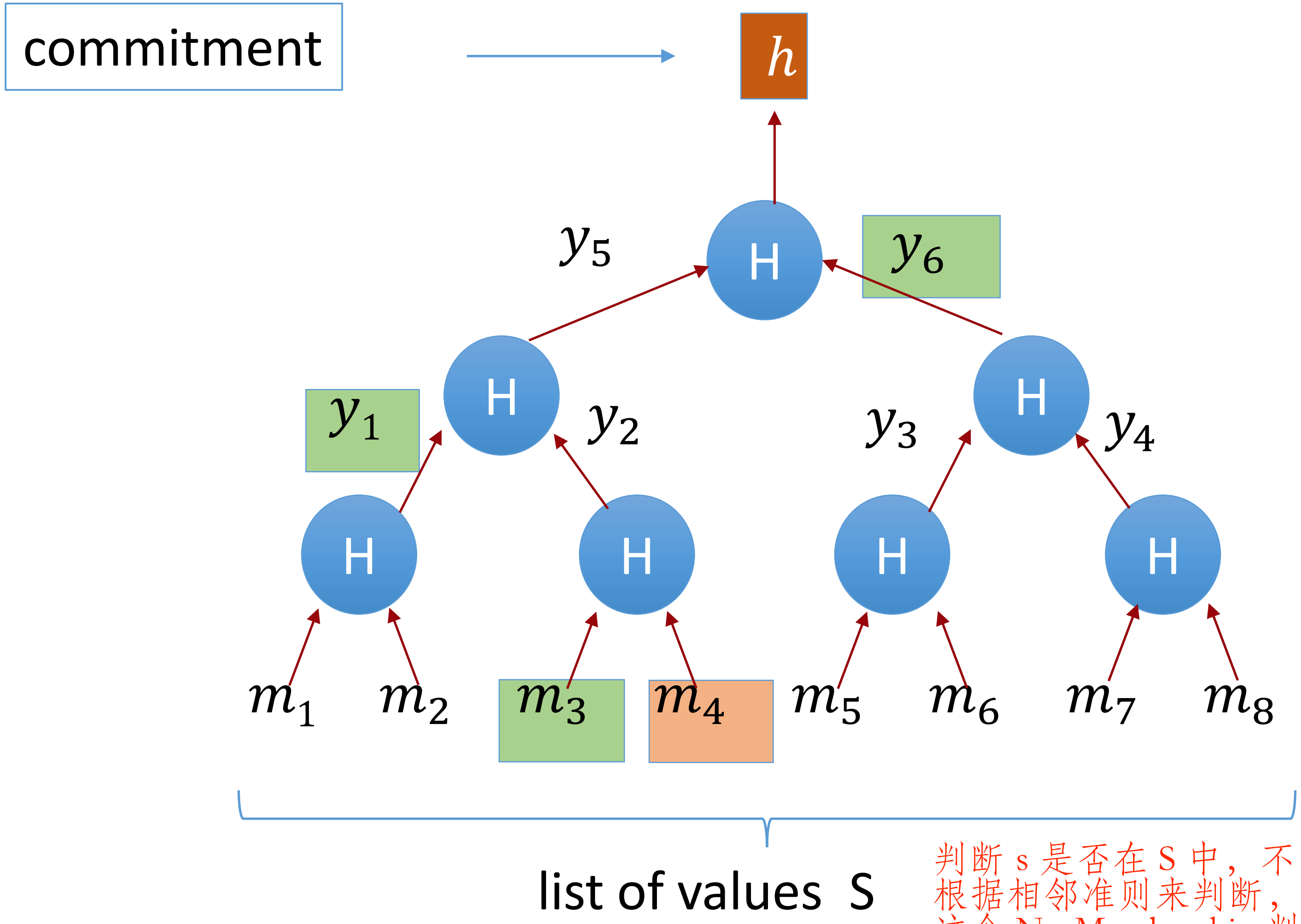
Can verify membership in *O(log n)* time/space

Variant: **sorted Merkle tree**

can verify non-membership in *O(log n)*

(show items before, after the missing one)

Sorted Merkle tree (Merkle 1989) [simplified]



To prove whether s in S is inside the Merkle tree,

Need *two consecutive items* m_i *and* m_j,

where $m_i \le s \le m_j$,

$$\pi = (m_i, y_a, y_b)$$
$$\pi' = (m_j, y_c, y_d)$$

判断 s 是否在 S 中，不需要知道所有的 S 中的内容。
根据相邻准则来判断，中间没有元素。
这个 No-Membership 判断是 Merkle tree 的一个重要应用。

More About Merkle Trees ..

More information about Merkle tree can be found at https://decentralizedthoughts.github.io/2020-12-22-what-is-a-merkle-tree/

We can use hash pointer in any pointer-based data structure that has no cycles.