
*CS1699: Blockchain Technology and Cryptocurrency*

# 3. Hash Pointers and Related Data Structures

Bill Laboon

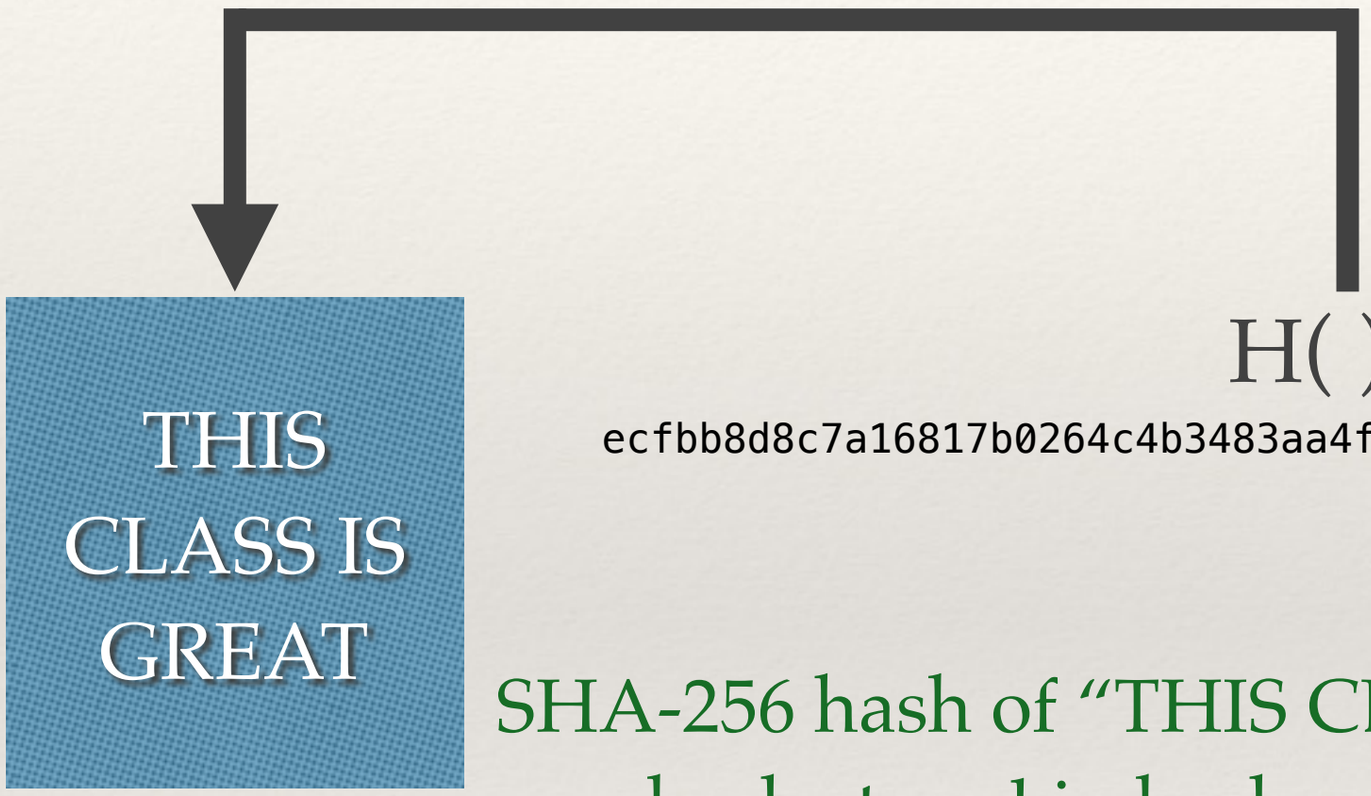
# What is a hash pointer?

- Two parts:
  - A pointer to some object or data set
  - A cryptographic hash of that data or representation of that object

# What good are hash pointers?

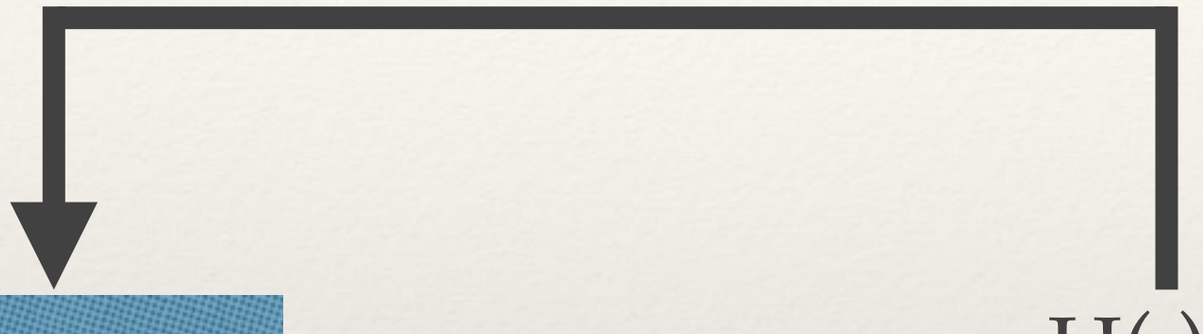
- Regular pointers (or object references in Java) can tell you where data is
- Hash pointers tell you that, plus let you verify that it has not been modified

# Hash Pointer




SHA-256 hash of "THIS CLASS IS GREAT" matches hash stored in hash pointer - ALL IS WELL

# Hash Pointer



THIS CLASS IS BAD

H( )

ecfbb8d8c7a16817b0264c4b3483aa4f7d3134e52de010af560d7c1d3d104a79

SHA-256 hash of “THIS CLASS IS BAD” is fa862cdde7d6e7f63ac843e8380f319629a9d718d3482cf6624e1d956497ba99 - DOES NOT MATCH!

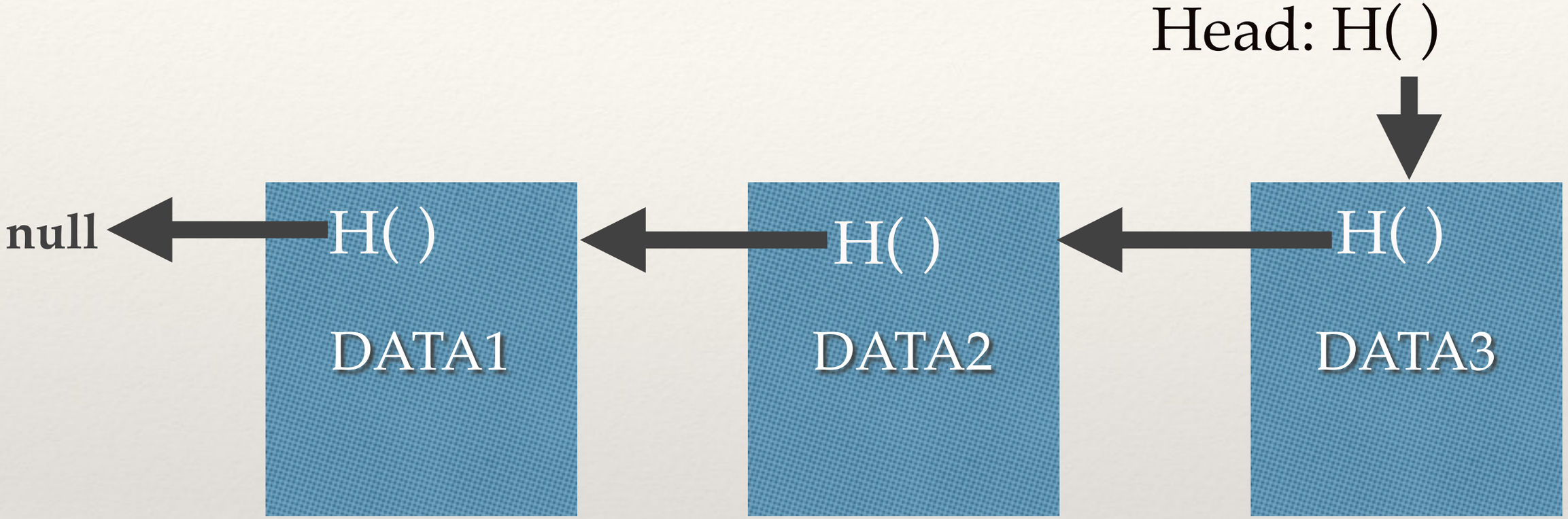
# Example

See /sample_code/hash_pointer for Java code

# Data Structures with Hash Pointers

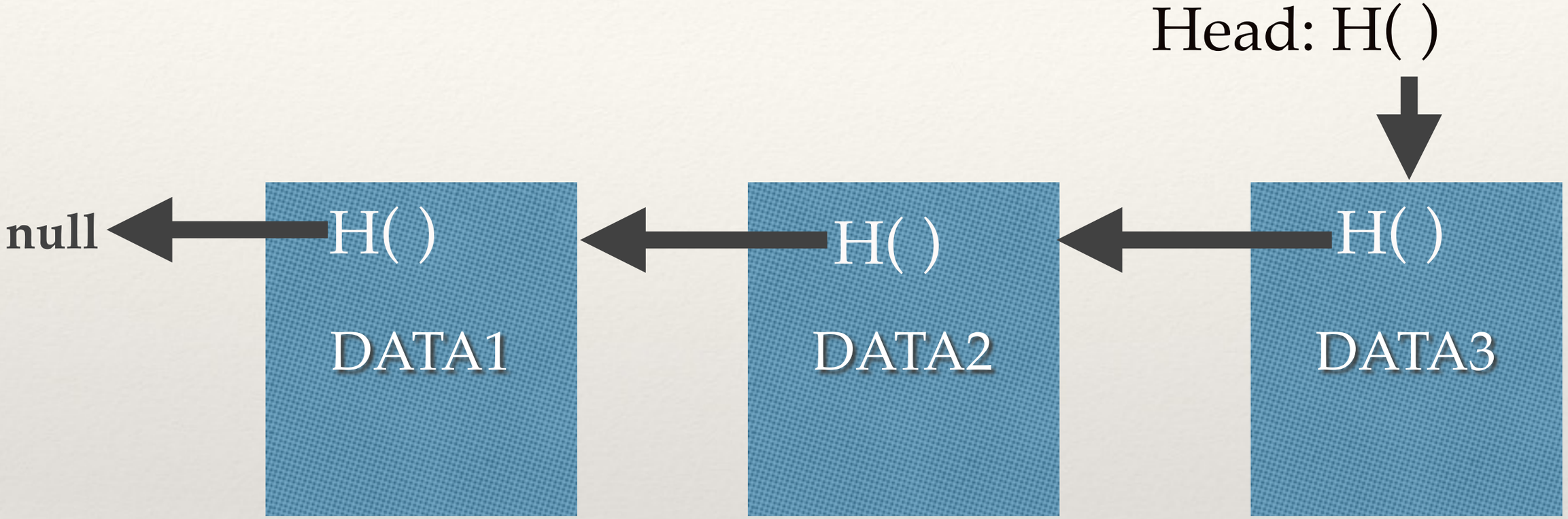
- Many data structures which use pointers/references can have their pointers/references replaced with hash pointers
- This makes them tamper-resistant versions of the "naive" data structure

# Linked List with Hash Pointers



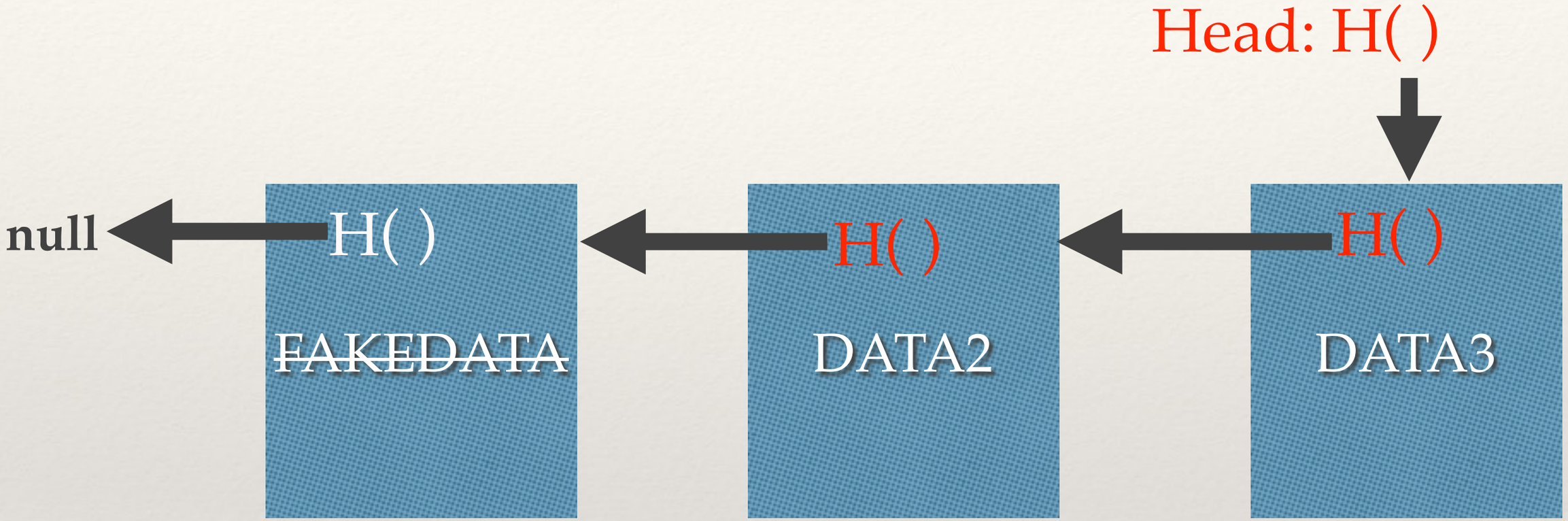

*A basic blockchain*

# Tamper-Resistant




***Hash pointer of previous data includes both data in node AND hash of preceding node.***

# Tamper-Resistant




*Strikethrough = modified data*

*White = Valid hash pointer*

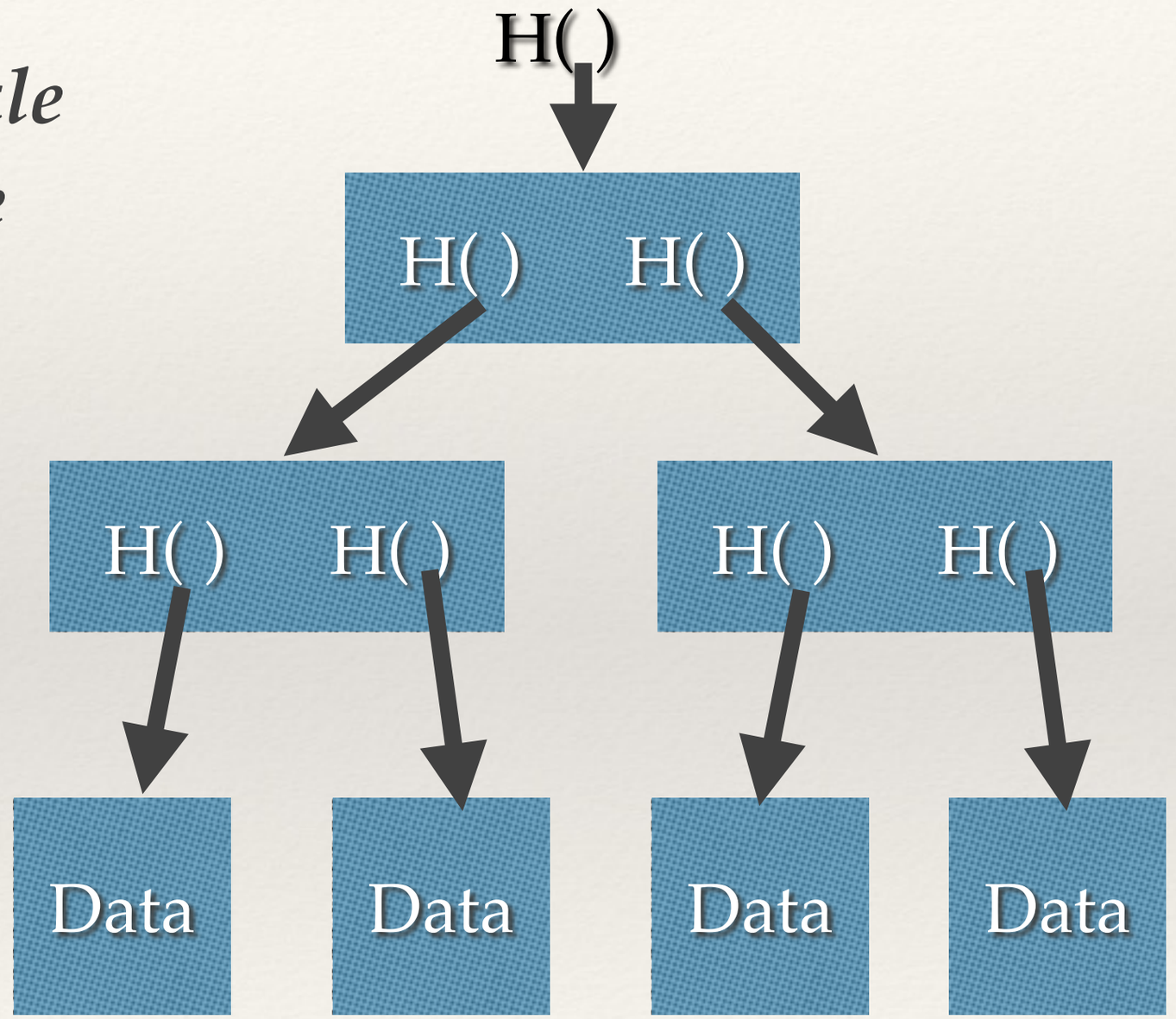
*Red = Invalid hash pointer*

# Example

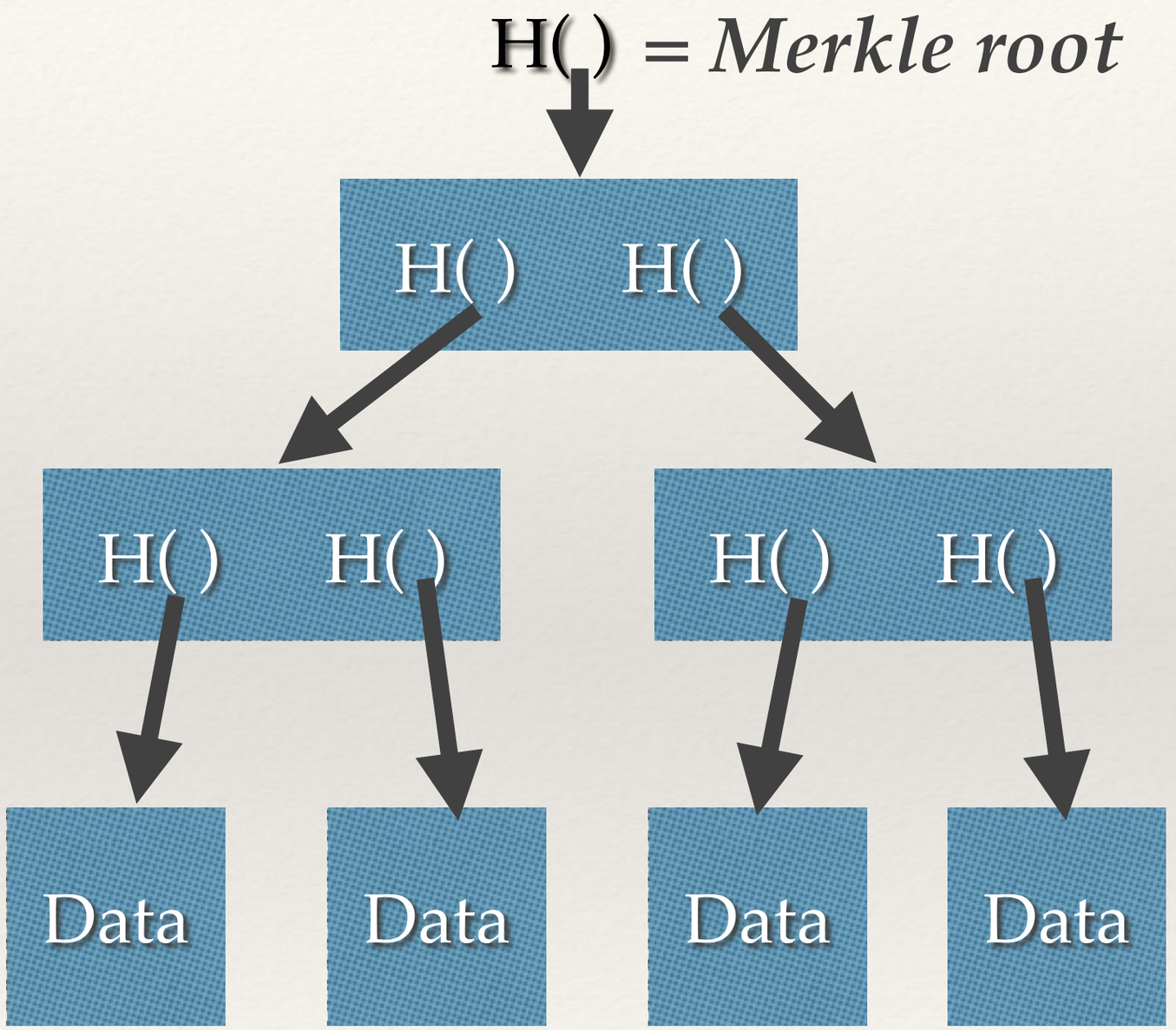
See /sample_code/basic_blockchain for Java code

# Binary Tree with Hash Pointers

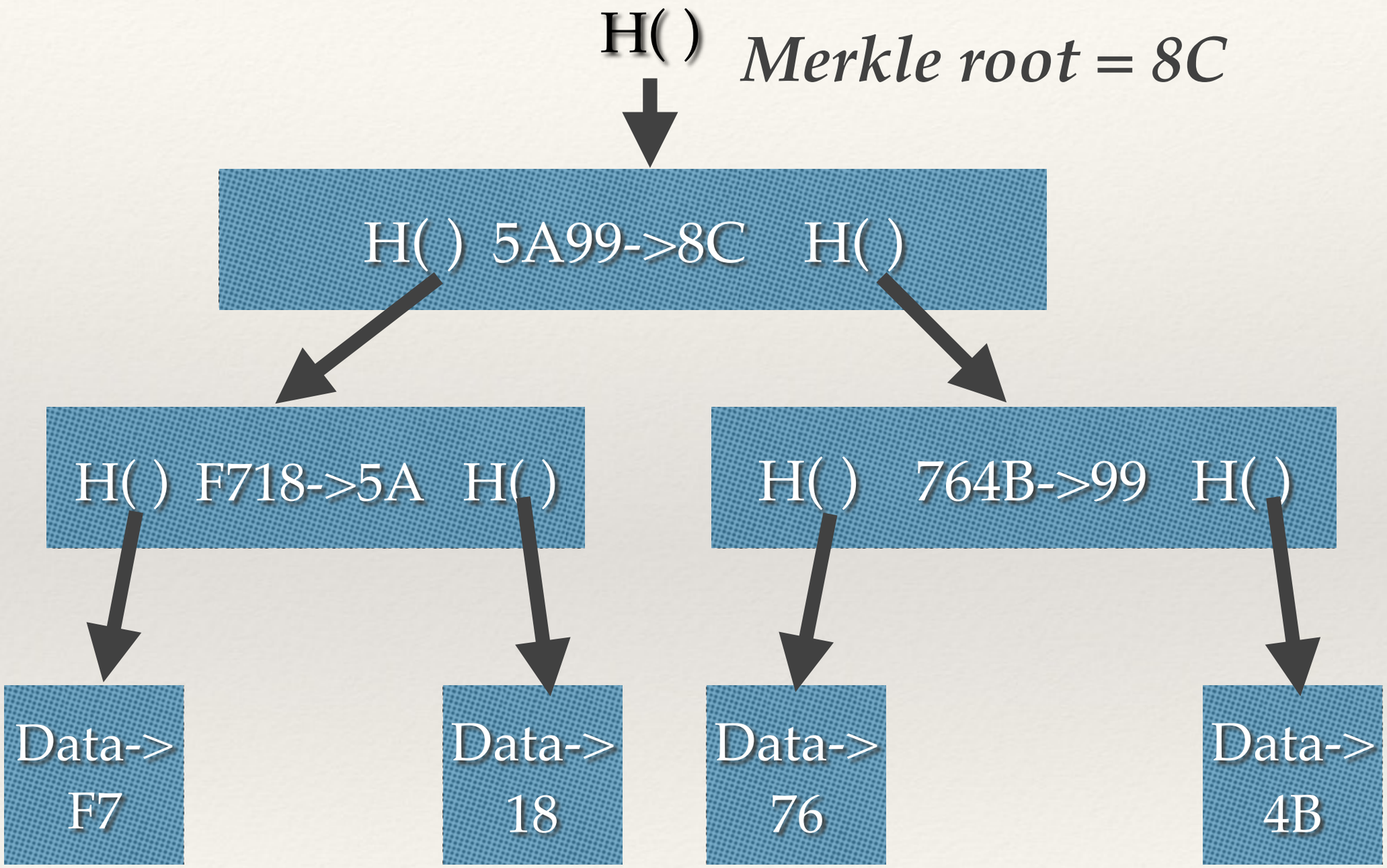
*Merkle tree*



# Merkle Root

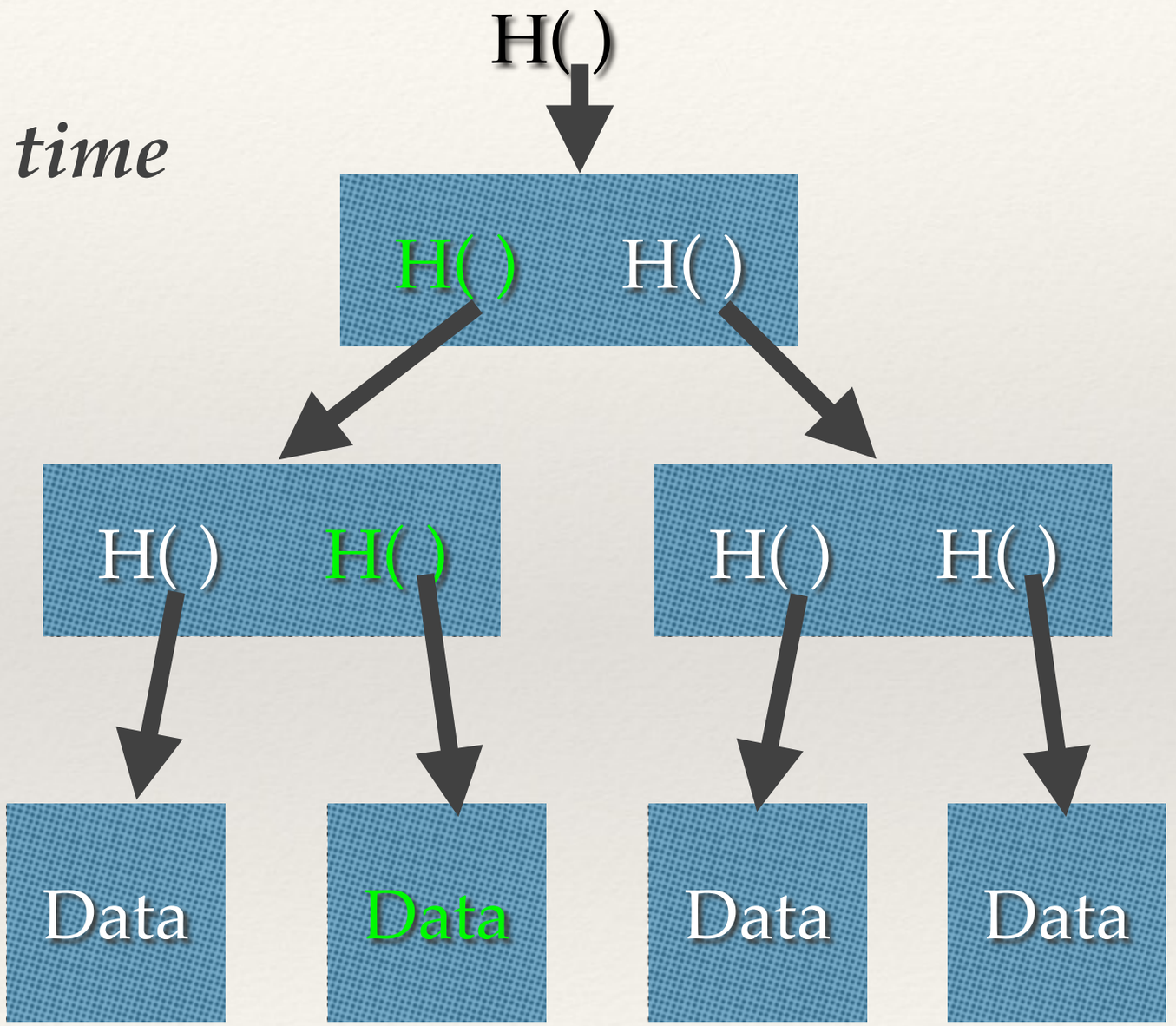


# Hashes of Hashes

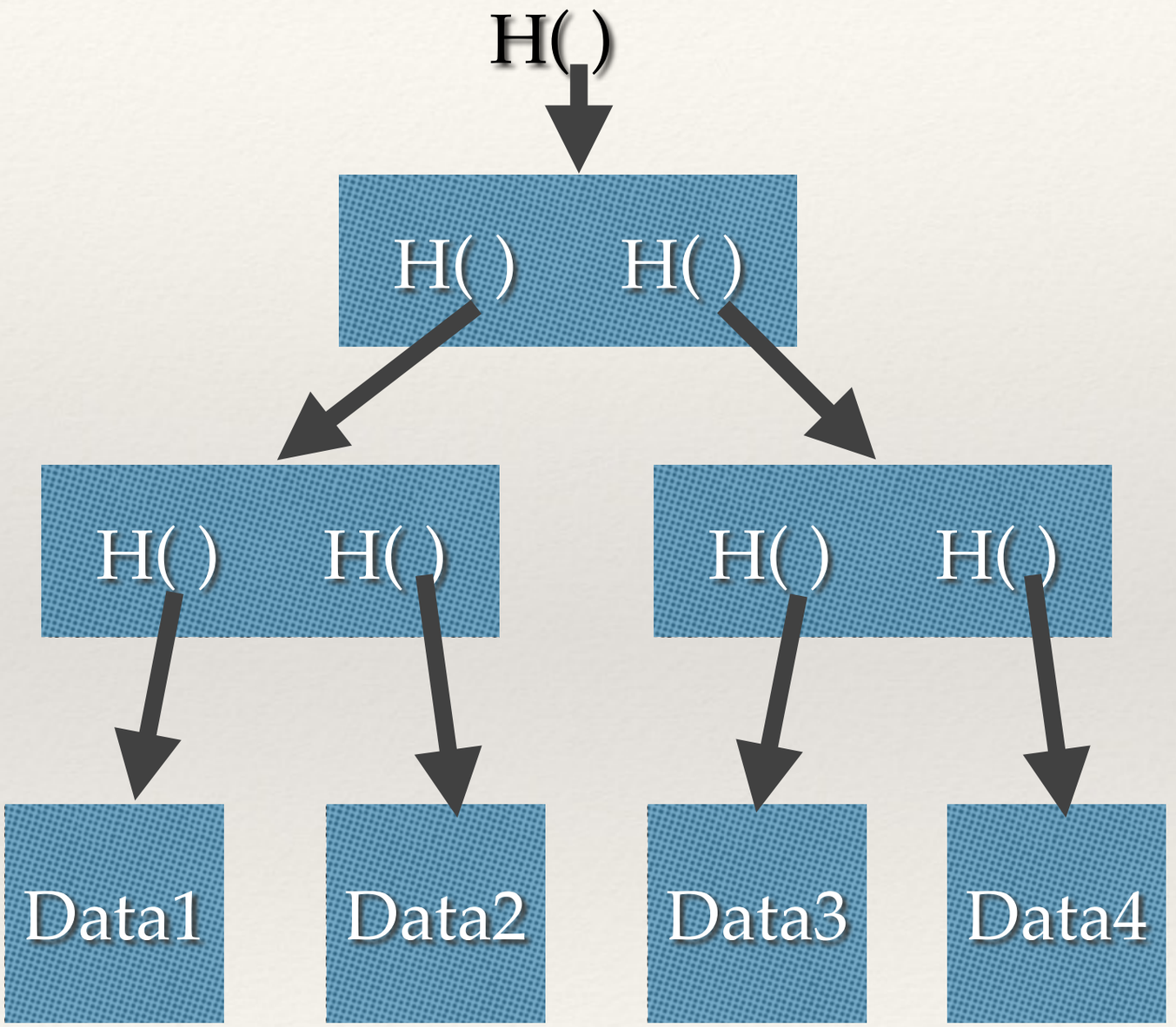


# Proving Membership with a Merkle Tree

*O(log n) time*

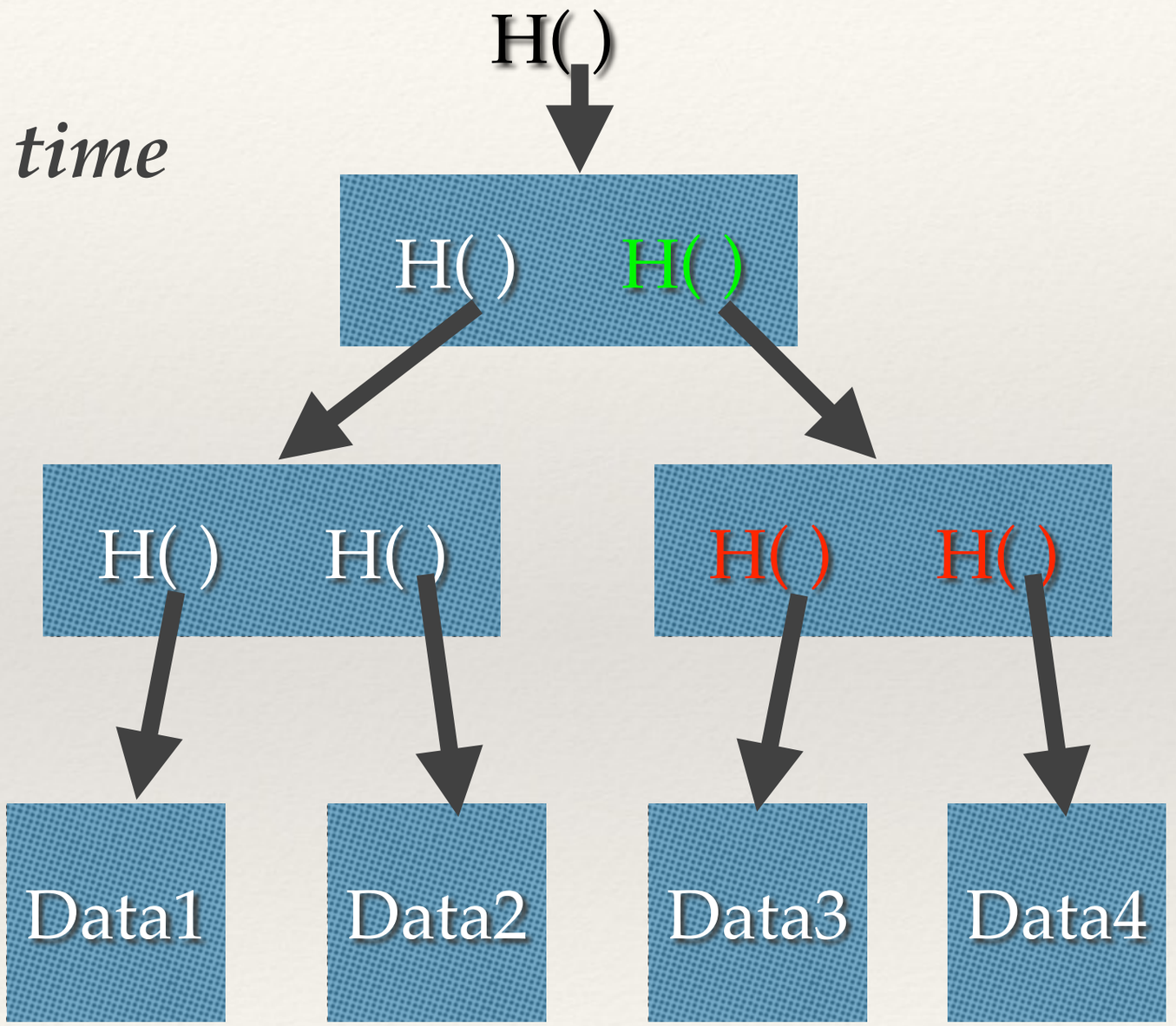


# Sorted Merkle Tree



# Disproving Membership in a Sorted Merkle Tree

*O(log n) time*



# Why Merkle Trees?

- Good access time
- Can just store Merkle root to verify entire tree
- Verify membership (or non-membership, for sorted Merkle trees) in O(log n) time

# Where Are Merkle Trees Used in Bitcoin?

- The Merkle root in a block is the Merkle root of all transactions in that block
- Allows mining to be approximately equally difficult no matter how many transactions are included
  - And thus incentivizes miners to add more transactions, so they get transaction fees!

# Block #540822

| Summary | |
|---|---|
| Number Of Transactions | 3193 |
| Output Total | 10,219.4089608 BTC |
| Estimated Transaction Volume | 789.12450391 BTC |
| Transaction Fees | 0.15175297 BTC |
| Height | 540822 (Main Chain) |
| Timestamp | 2018-09-10 16:46:00 |
| Received Time | 2018-09-10 16:46:00 |

| Hashes | |
|---|---|
| Hash | 00000000000000000000b26d77f0f823ccdd0e3b097b6c03b9789e43f90ed0ef2 |
| Previous Block | 00000000000000000023fc653842c9a7e4019f31a615a26b40fd5c59334f33fa |
| Next Block(s) | 00000000000000000000035213b4e39dabb4e596bf3503028587f9dab5a4c66f92 |
| Merkle Root | 6d03f773cb2f53d68eebf3f594ee27b5988a4c33251ba3d8241c17a441e551f1 |

# Where Can't We Use Hash Pointers?

- Data structures without explicit pointers / references (e.g. arrays)
- Data structures which can include cycles
  - Cyclic structures = no starting point for hashes