Merkle trees and Blockchains

Dor Cohen

Alice and Bob

- Bob stores a set of items for Alice.
- Alice keeps a single value.
- Alice can validate the Items returned to her.

Basics - Cryptographic Hash

- Arbitrary Input size.
- Output size is fixed.
- **H**(x) is easy to compute.
- But finding any x, x′ s.t. **H**(x) = **H**(x′), should be computationally hard.
- The output should also appear “random”.

First Solution

- Alice keeps the hash of the entire set.

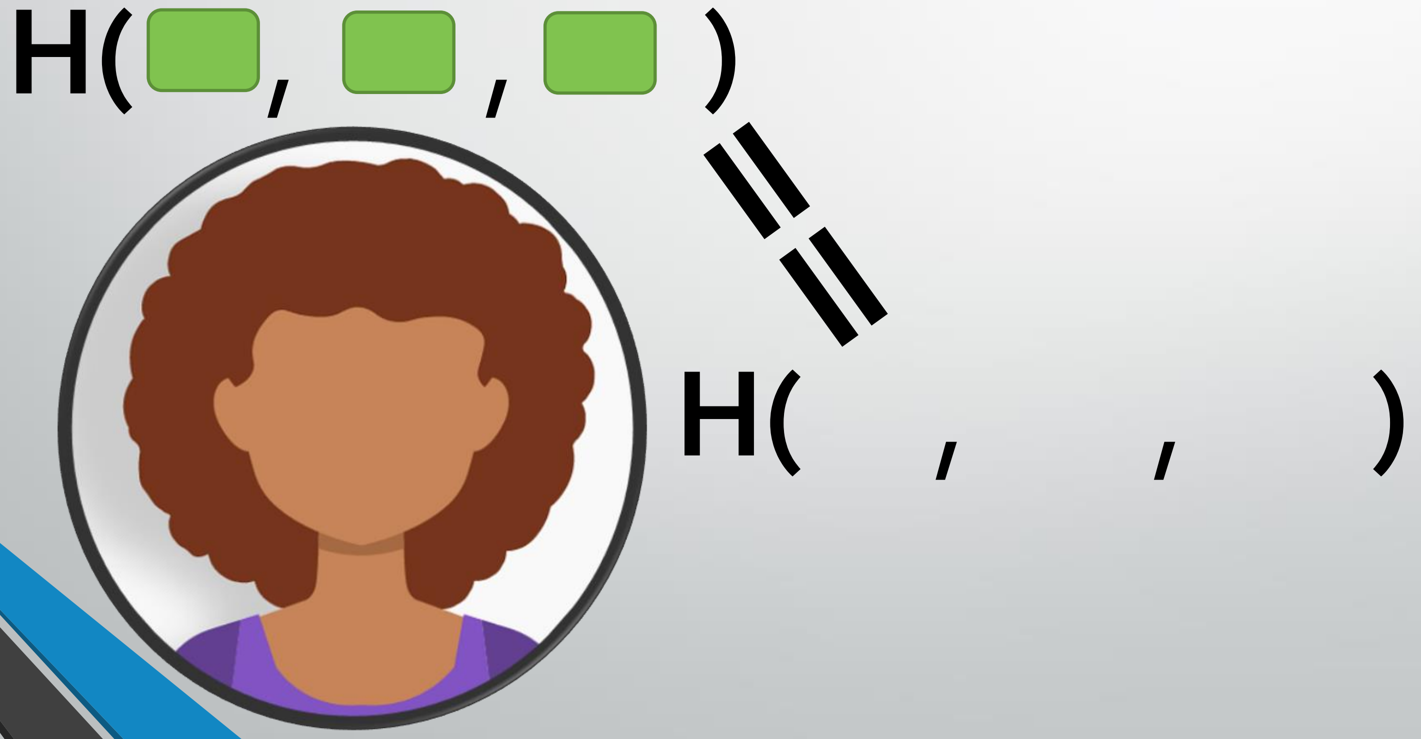
$$H(\square, \square, \square)$$

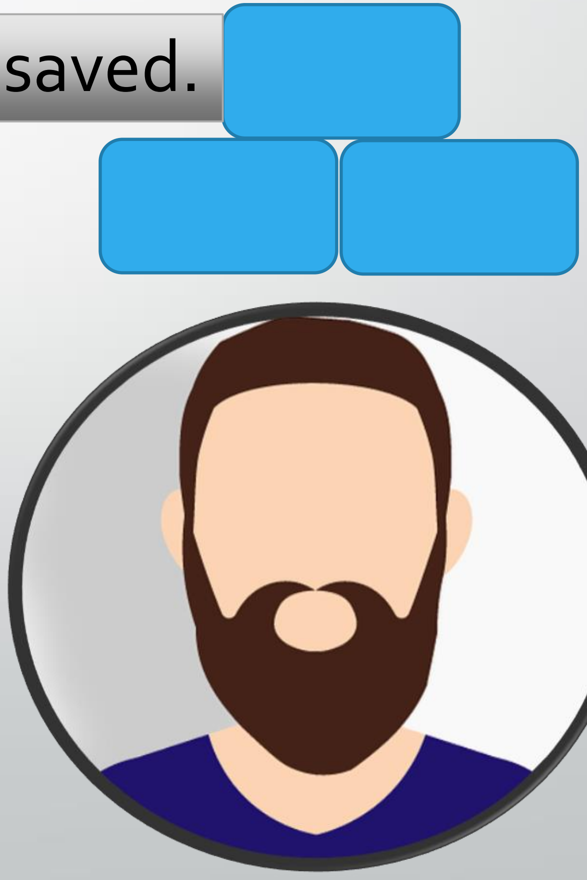
Validation Of An Item

Bob sends all of the items to Alice.

Alice computes the hash of the items.

Alice compares the result to the value she has saved.





Problems With First Solution

- Bob must send Alice the entire set for validation.

Problems With First Solution

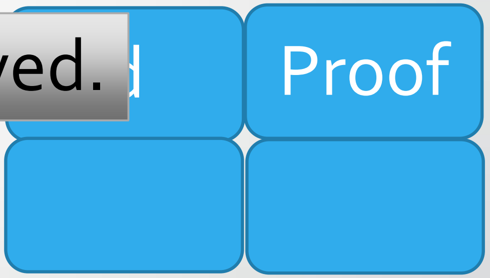
- Bob must send Alice the entire set for validation.
- Denote m to be the size of the set.
- We have O(m) network traffic for validating a single item.
- Can we do better?

Validating An Item

Bob sends Alice an item d and a logarithmic size proof.

Alice computes a function of the item and proof.

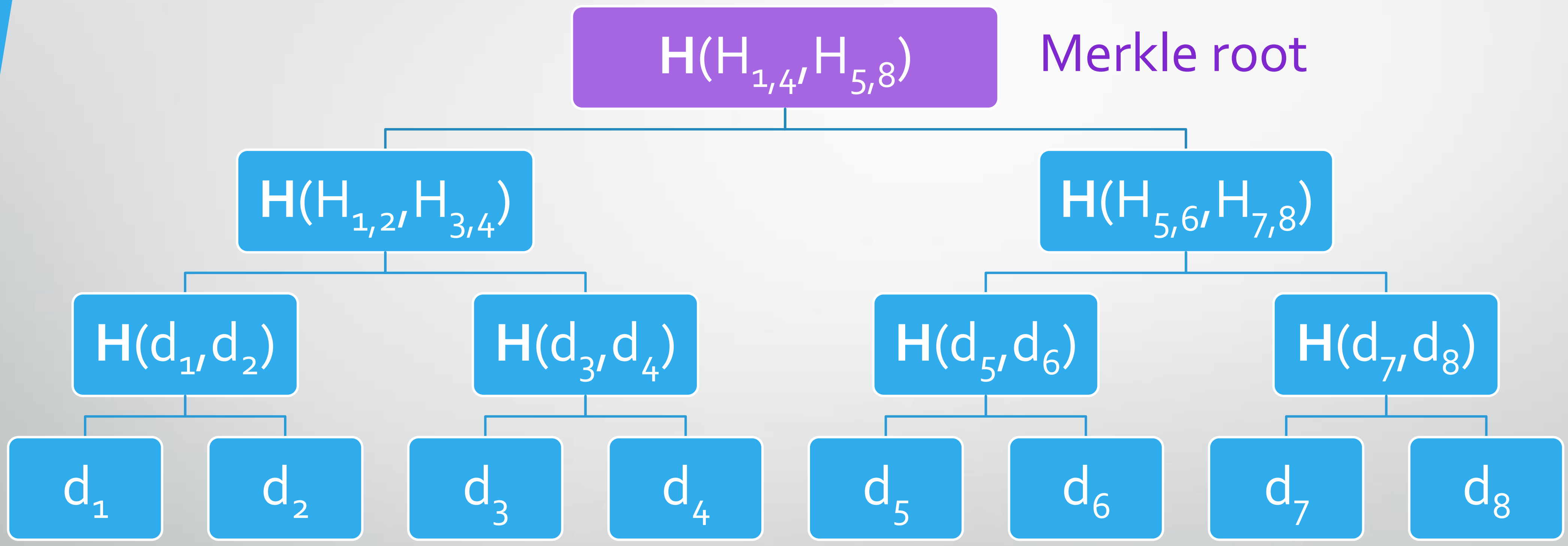
Alice compares the result to the value she has saved.



ce86b7dde40...

$$= F(\quad , \quad)$$

Merkle Tree

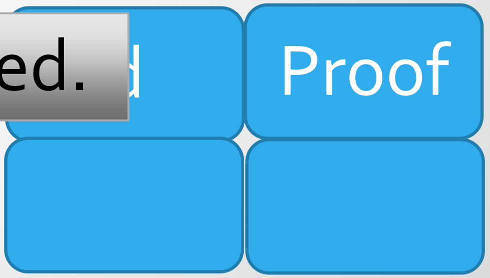

$H(H_{1,4}, H_{5,8})$
Merkle root
$H(H_{1,2}, H_{3,4})$
$H(H_{5,6}, H_{7,8})$
$H(d_1, d_2)$
$H(d_3, d_4)$
$H(d_5, d_6)$
$H(d_7, d_8)$
d_1
d_2
d_3
d_4
d_5
d_6
d_7
d_8

Validating An Item

Bob sends Alice an item d and a logarithmic size proof.

Alice uses the proof to compute the Merkle root.

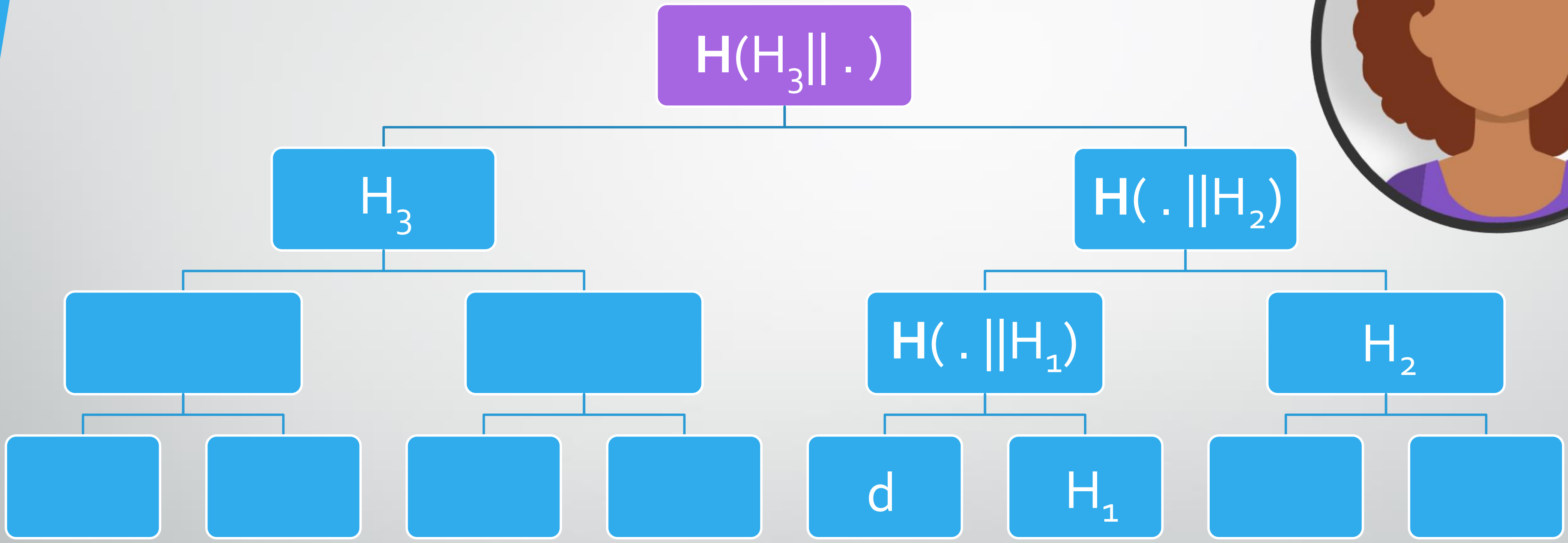
Alice compares the result to the root she has saved.

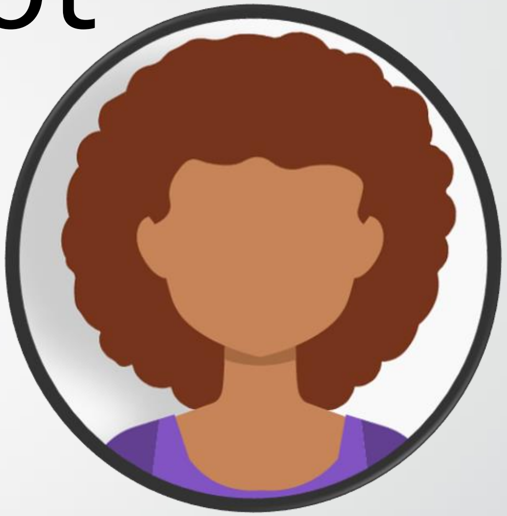


Merkle root

Alice Computes The Root

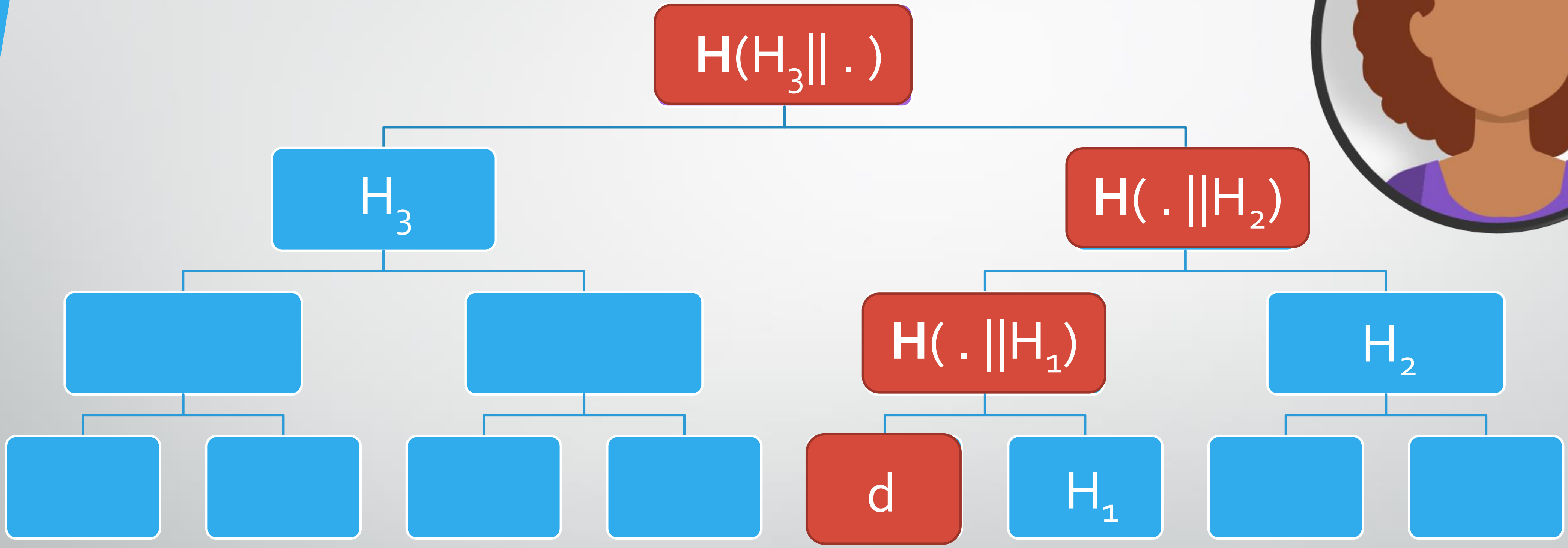
Given an item d and H_1,H_2,H_3 hash values





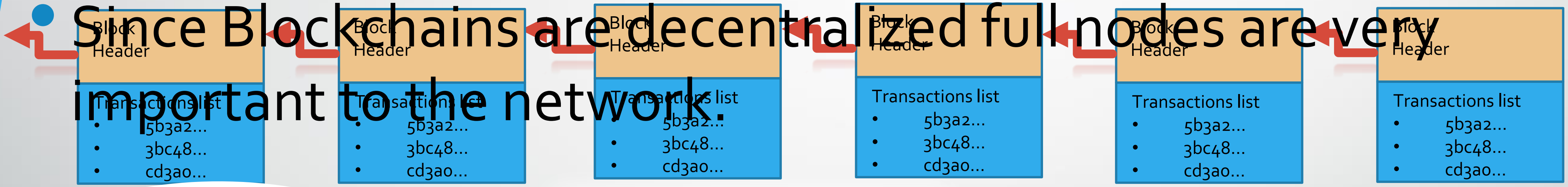
What If d Isn't Valid?

Given an item d and H_1,H_2,H_3 hash values



Blockchain– Full nodes

- Full nodes are nodes in the Blockchain network that store the entire Blockchain in order to validate new transactions.
- Since Blockchains are decentralized full nodes are very important to the network.



Motivation - Storing the Blockchain

- The problem is the Blockchain can take up a lot of memory.

Database size	
Bitcoin	149 GB (December 2017)
Ethereum	400 GB (February 2018)

- And these numbers are constantly growing.
- Some devices such as mobile can't spare that much space.

A Block in the Blockchain

- Block's are consisted of two main parts:

Block Header

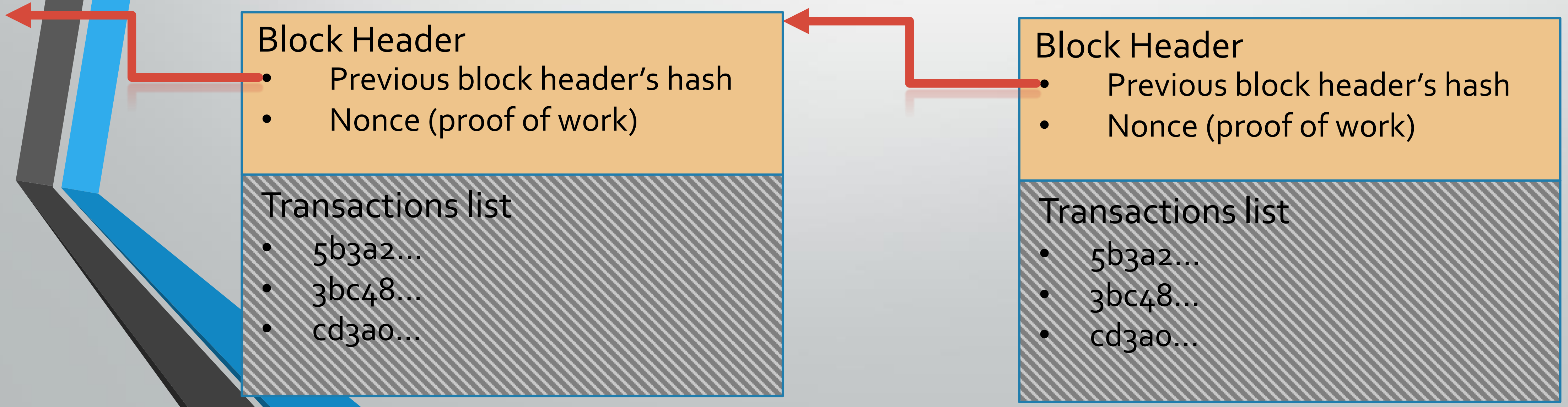
- Previous block header's hash
- Nonce (proof of work)

Transactions list

- 5b3a2…
- 3bc48…
- cd3a0…

Light Nodes

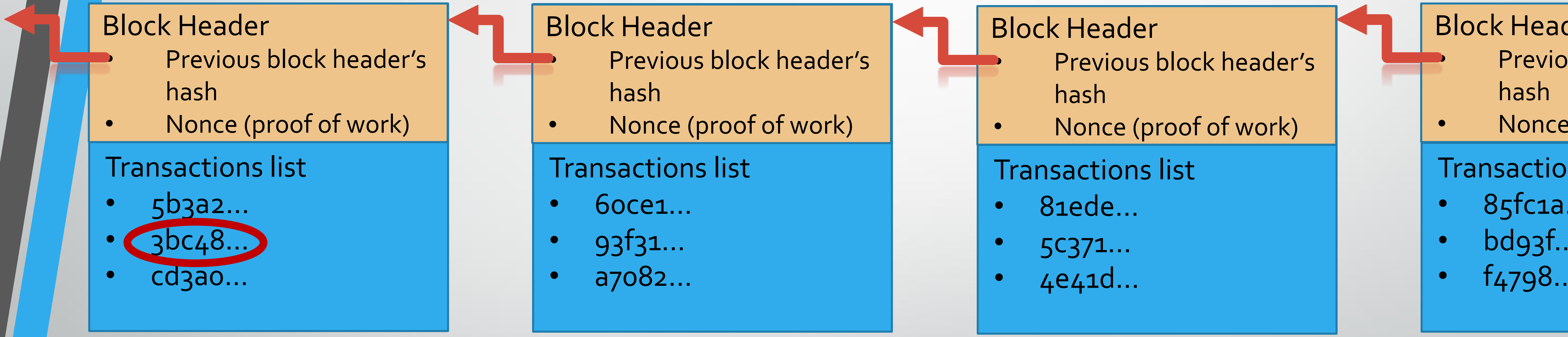
- Light nodes were created for simple clients who want to save on storage.
- They only store the chain of block headers.
- They follow the longest chain rule, without validating transactions.



Paying With Light Nodes

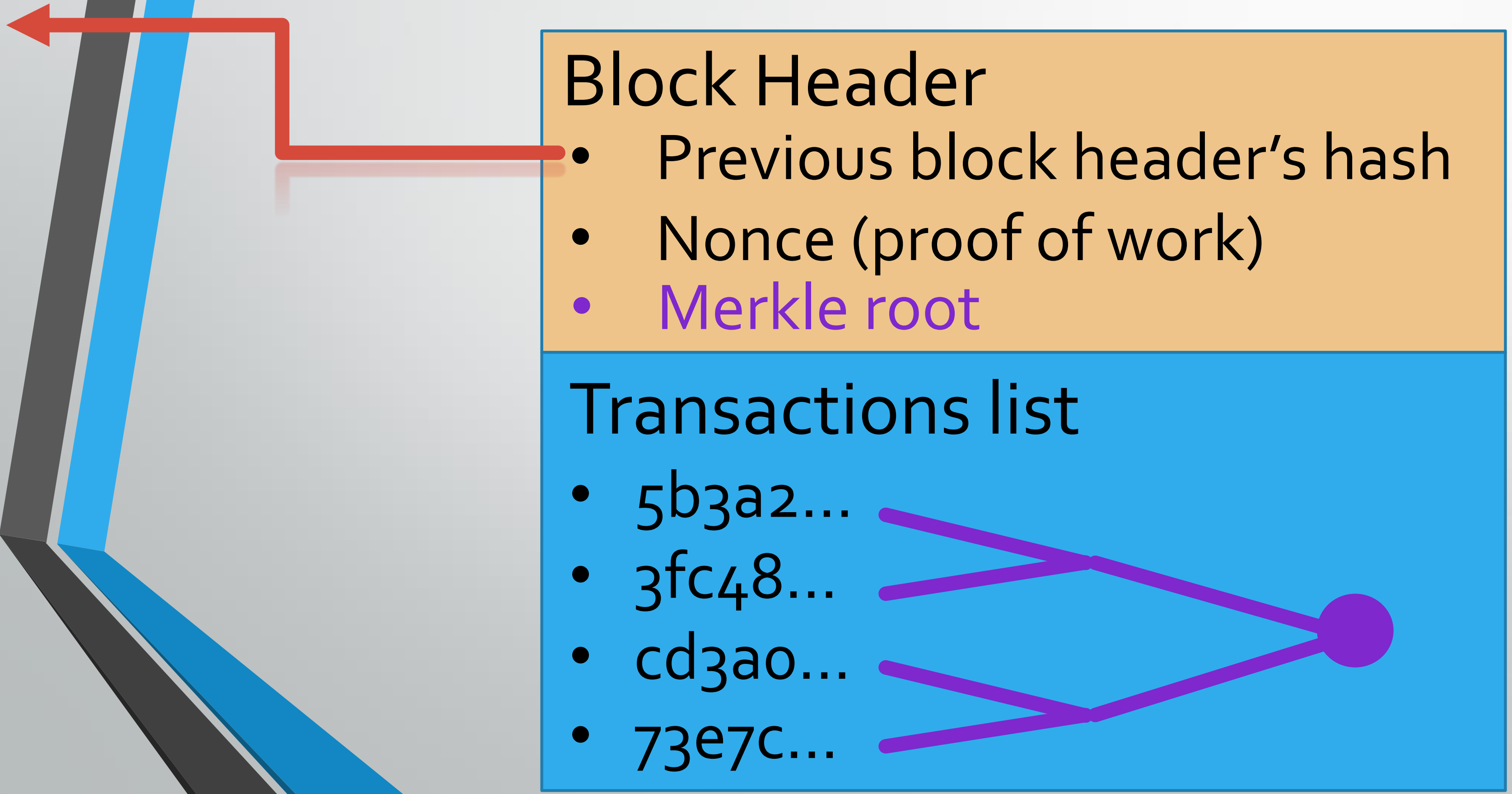
- Same as full nodes.

Accepting Payment



Merkle Root In Header

- Put the Merkle root of transactions in the header.



Merkle Root In Header

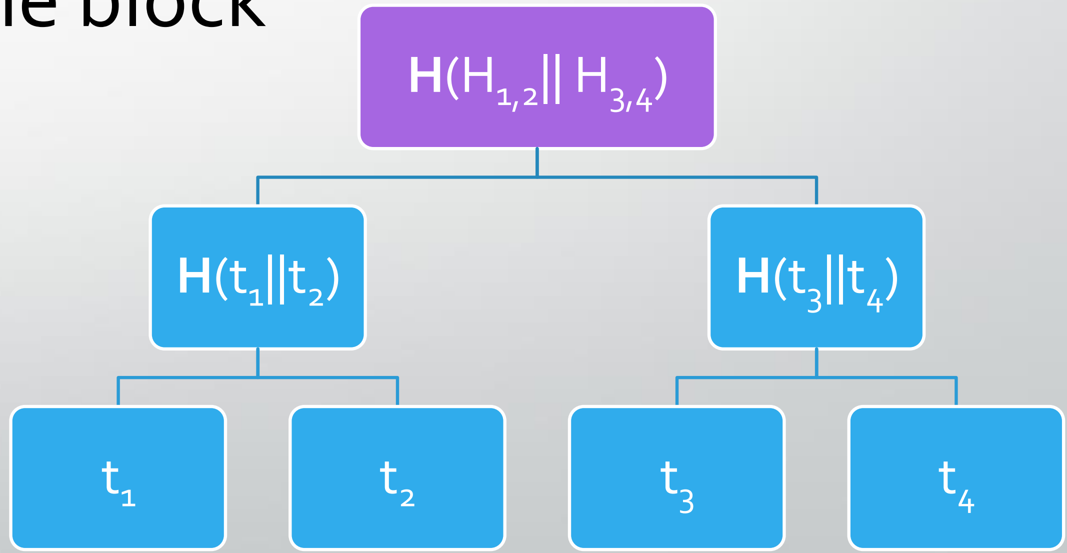
- Now light nodes can request transactions from full nodes, and know that they were from a block.
- Just like Alice did with Bob.

Forgetting Spent Transactions

- Having the Merkle root in the header has another interesting perk.
- Freeing storage in full nodes by forgetting transactions that are already spent.

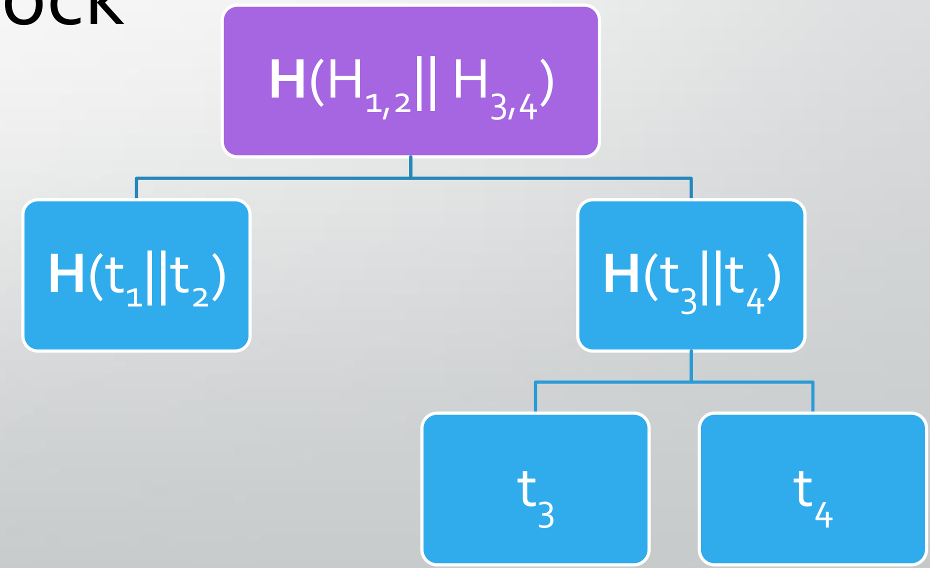
Forgetting Spent Transactions

- Suppose that t_1, t_2, t_3 have all been spent.
- We can get rid of most of them.
- Now we're storing $\mathbf{H}(t_1||t_2)$ in the block instead of t_1 and t_2.



Forgetting Spent Transactions

- Suppose that t_1, t_2, t_3 have all been spent.
- We can get rid of most of them.
- Now we're storing $\mathbf{H}(t_1 || t_2)$ in the block instead of t_1 and t_2.



Conclusion

- Merkle trees are a smart way to hash.
- They allow for easier storage of Blockchains, allowing headers to represent the entire block in a concise way.
- They even allows us to forget the transaction IDs of spent transactions.

Questions?