



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.

$H(\text{[green box]}, \text{[green box]}, \text{[green box]})$



Validation Of An Item

← Bob sends all of the items to Alice.

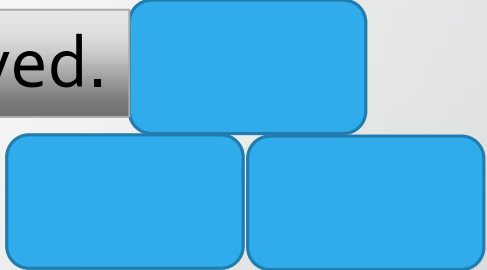
Alice computes the hash of the items.

Alice compares the result to the value she has saved.

$$H(\text{■}, \text{■}, \text{■})$$



$$= H(\text{■}, \text{■}, \text{■})$$



Problems With First Solution

- Bob must send Alice the entire set for validation.

OK?



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.

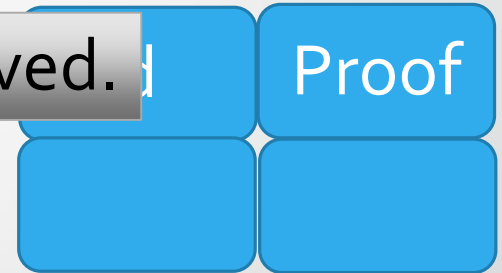
Alice computes a function of the item and proof.

Alice compares the result to the value she has saved.

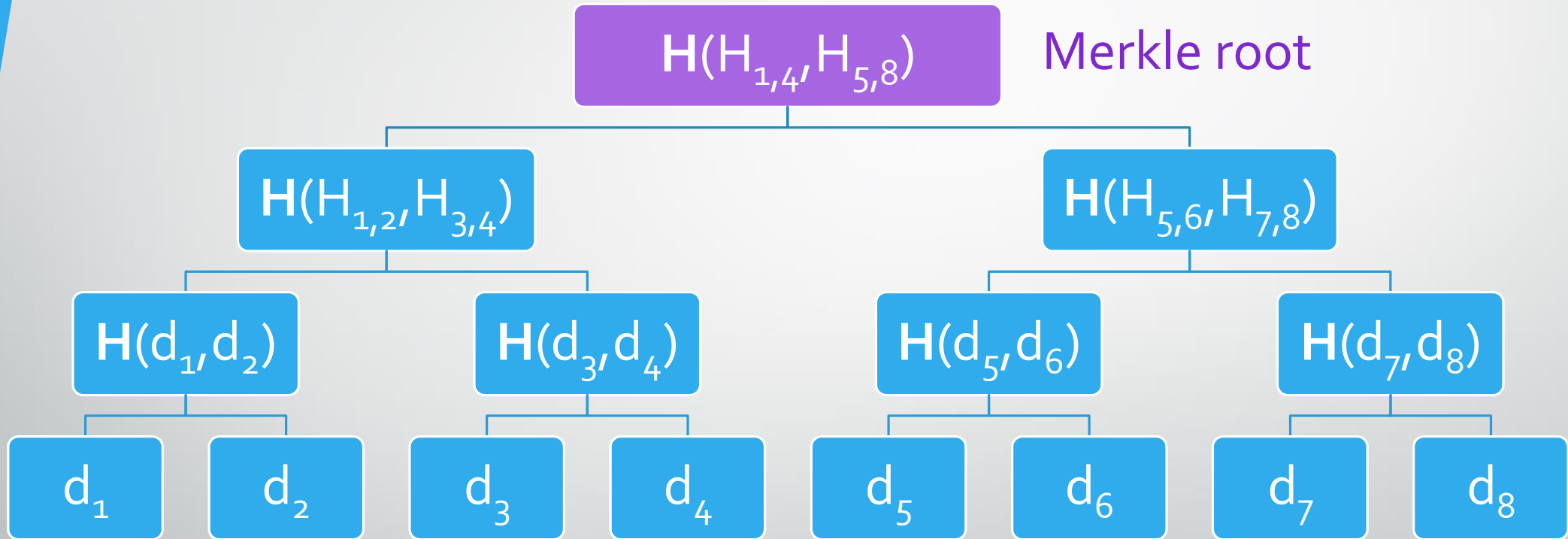
ce86b7dde40...



$F(d, \text{Proof})$



Merkle Tree



Validating An Item

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

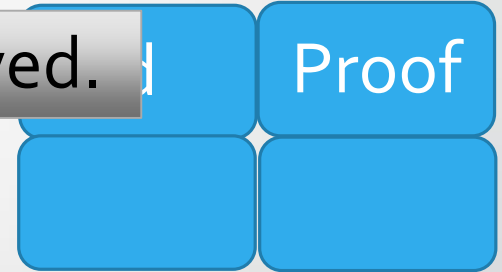
Alice uses the proof to compute the Merkle root.

Alice compares the result to the root she has saved.

Merkle root

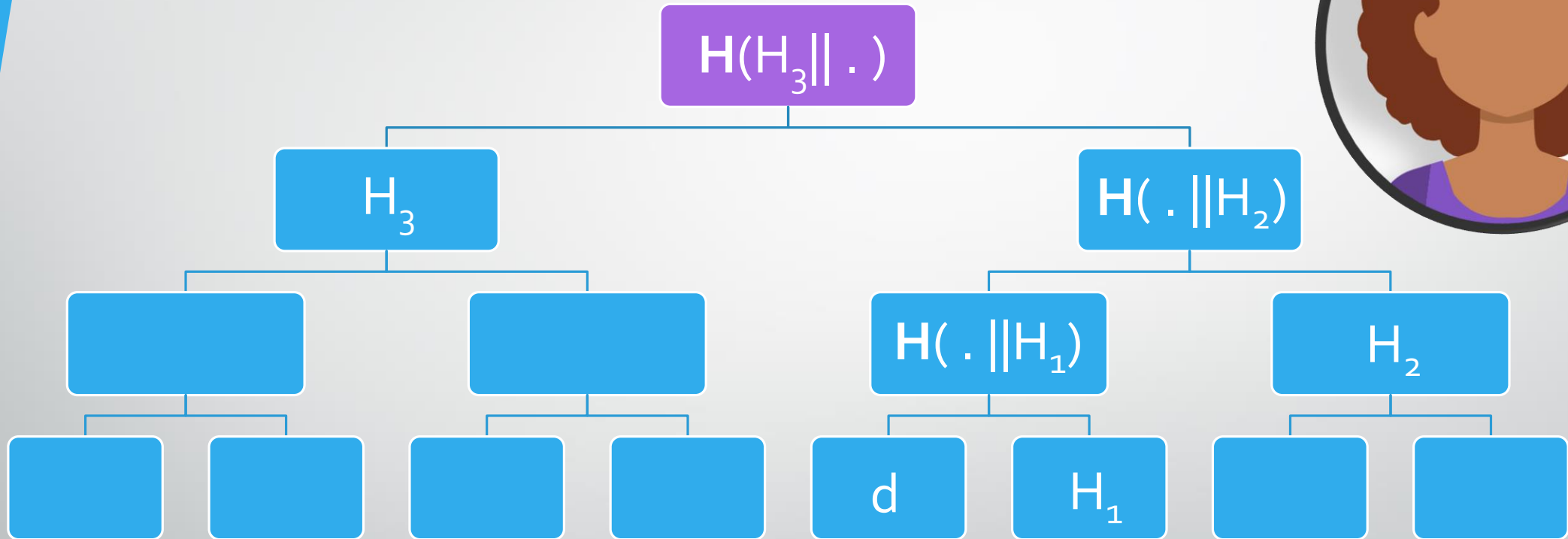


$F(\quad , \quad)$



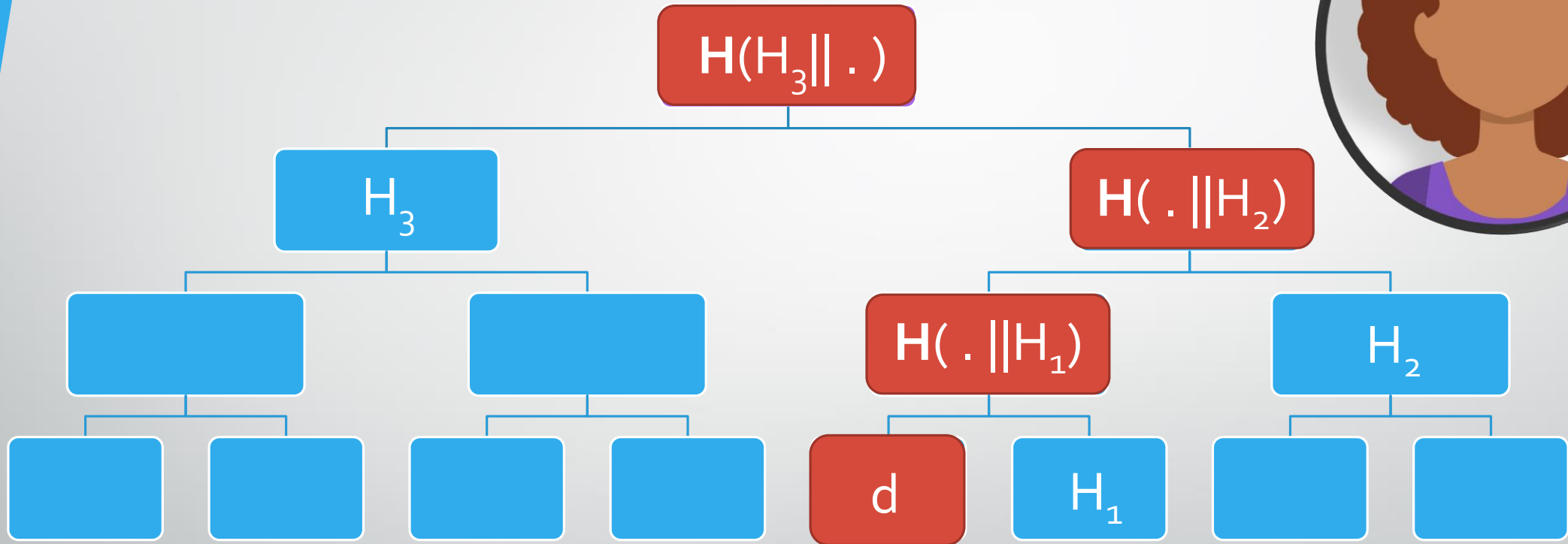
Alice Computes The Root

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



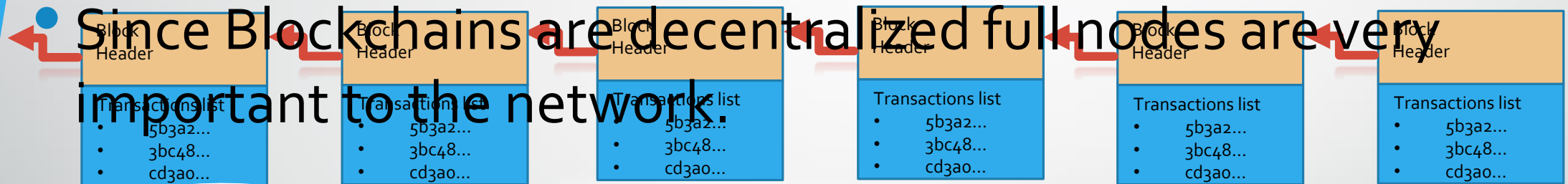
What If d Isn't Valid?

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.



No double spending here



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:

A diagram showing a block structure. On the left, a vertical blue bar with a grey shadow is connected to a horizontal blue bar. A red arrow points from the horizontal bar to the 'Block Header' section of a block. The block is a rectangle divided into two horizontal sections: an orange top section and a blue bottom section. The orange section is titled 'Block Header' and contains two bullet points: 'Previous block header's hash' and 'Nonce (proof of work)'. The blue section is titled 'Transactions list' and contains three bullet points: '5b3a2...', '3bc48...', and 'cd3a0...'.

Block Header

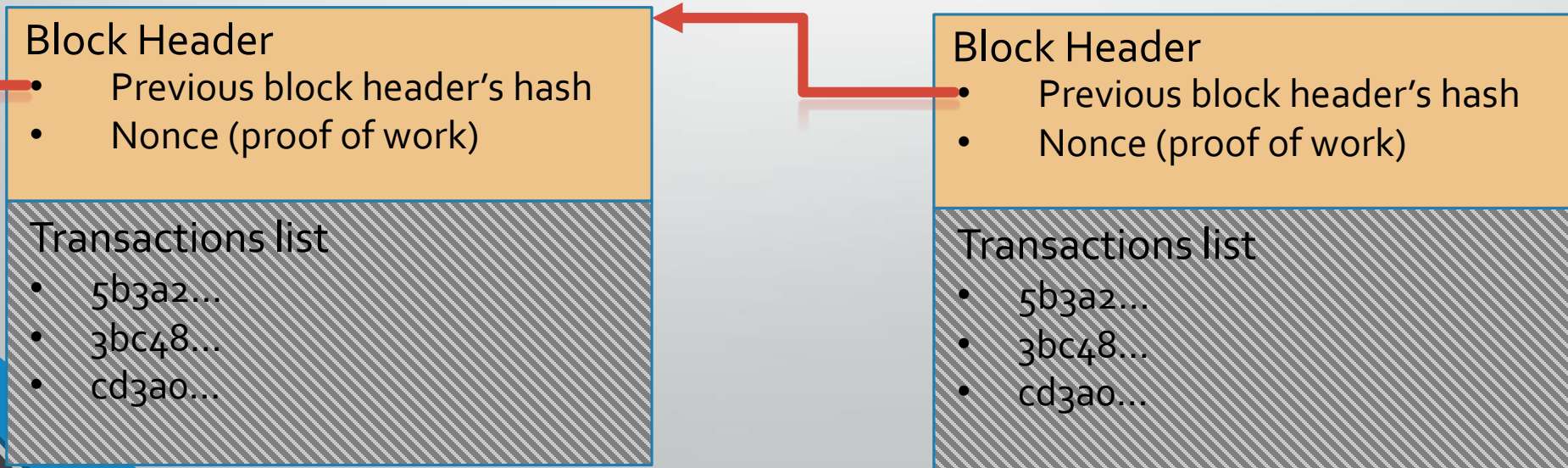
- Previous block header's hash
- Nonce (proof of work)

Transactions list

- 5b3a2...
- 3bc48...
- cd3a0...

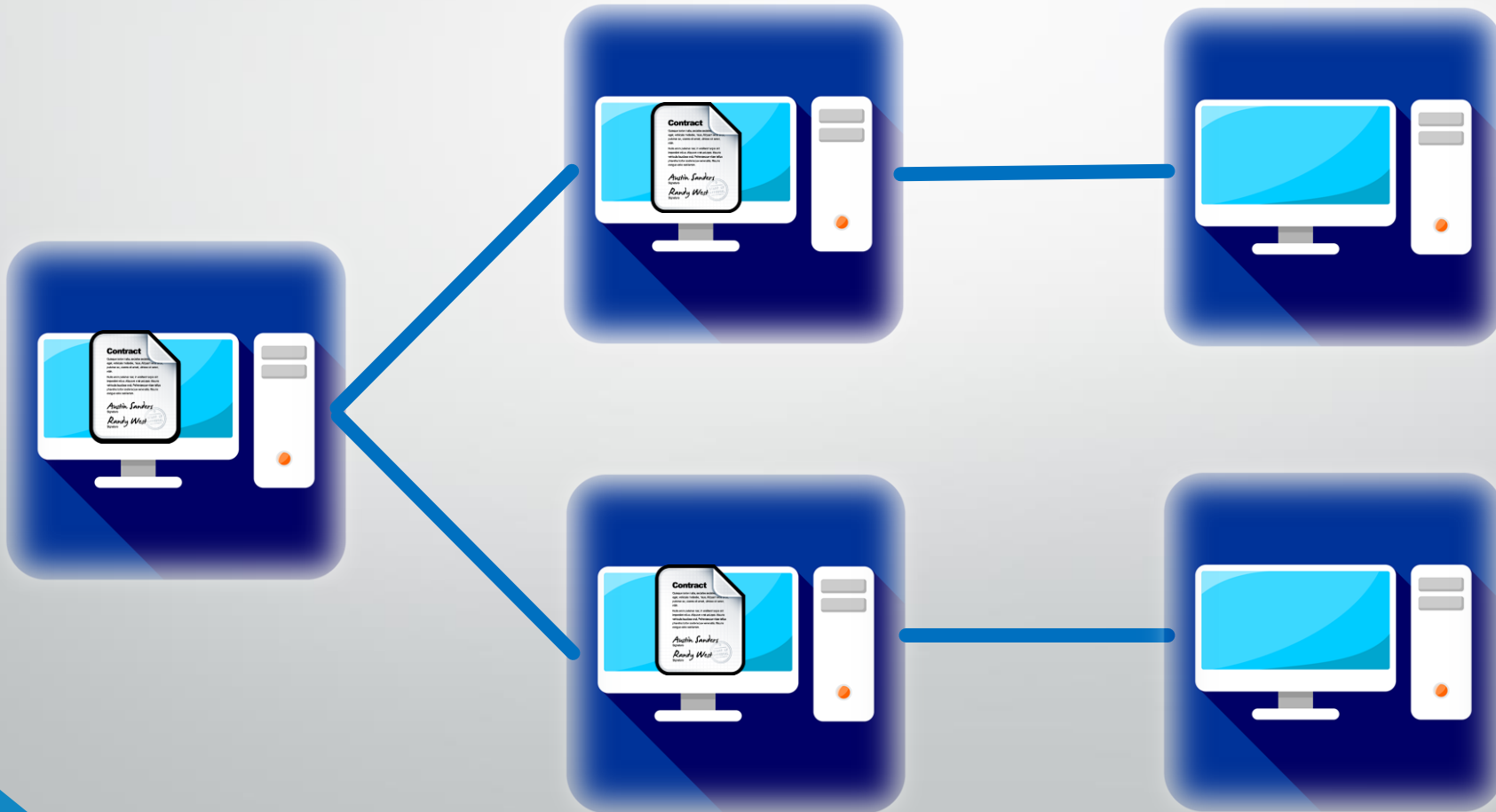
Light Nodes

- 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.

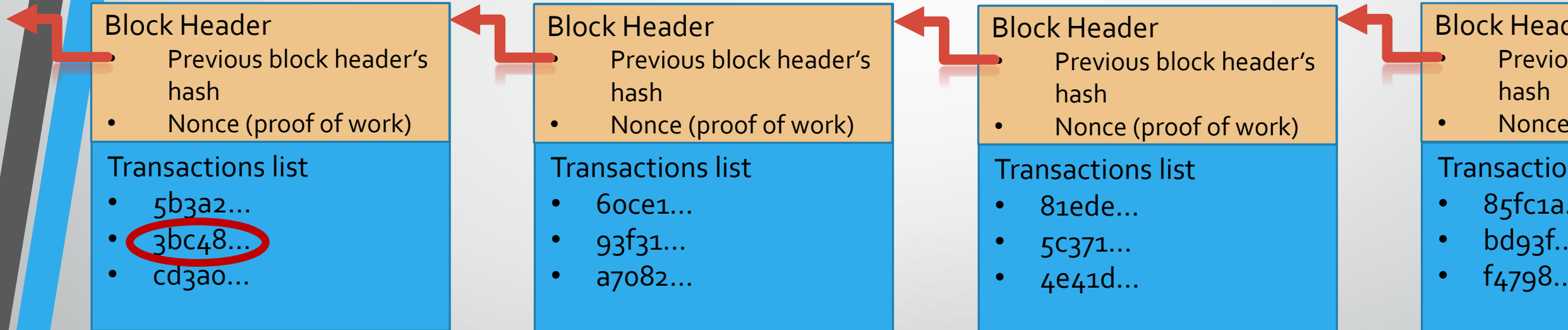


Paying With Light Nodes

- Same as full nodes.



Accepting Payment



Merkle Root In Header

- Put the Merkle root of transactions in the header.

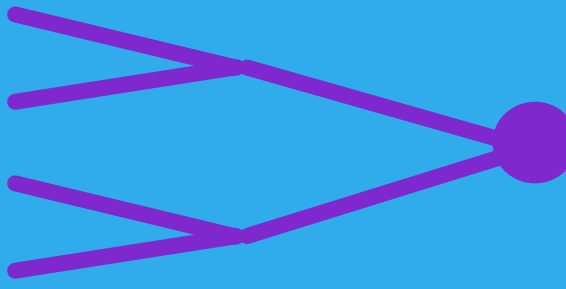


The diagram shows a block structure on the left, represented by a blue and grey vertical bar. A red arrow points from the 'Block Header' section of a box to the left, indicating its position within the block. The box is divided into two sections: an orange 'Block Header' and a blue 'Transactions list'. The 'Block Header' contains three items: 'Previous block header's hash', 'Nonce (proof of work)', and 'Merkle root'. The 'Transactions list' contains four transaction hashes: '5b3a2...', '3fc48...', 'cd3ao...', and '73e7c...'. Purple lines connect these four hashes to a single purple circle on the right, representing the Merkle root.

Block Header

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

Transactions list

- 5b3a2...
 - 3fc48...
 - cd3ao...
 - 73e7c...
- 
- A Merkle tree diagram showing four transaction hashes on the left. Purple lines connect the first two hashes to a node, and the last two hashes to another node. These two nodes then connect to a final purple circle on the right, representing the Merkle root.

Merkle Root In Header

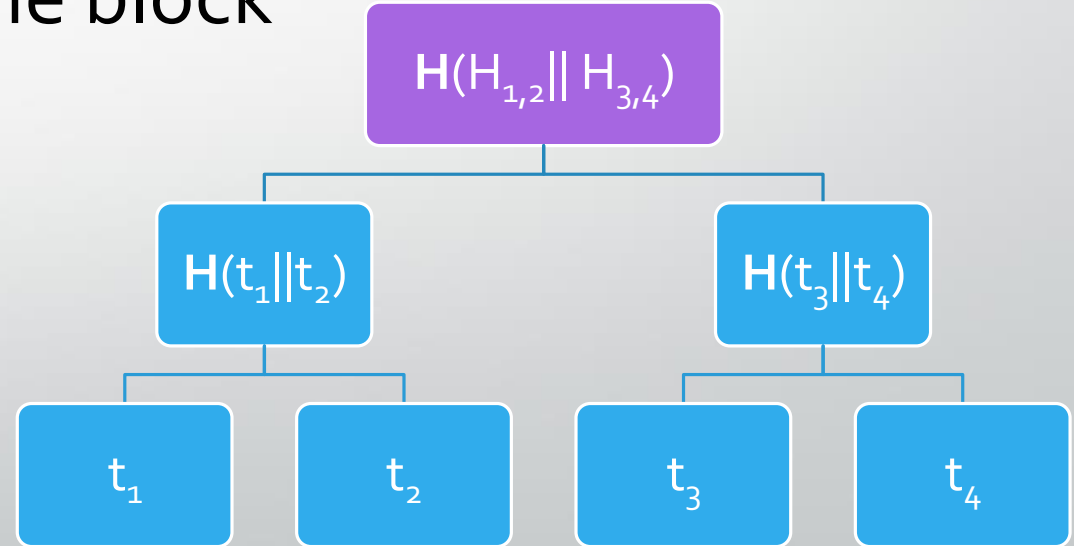
- Now light nodes can request transactions from full nodes, and know that they were from a block.
- Just like Alice did with Bob.

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.

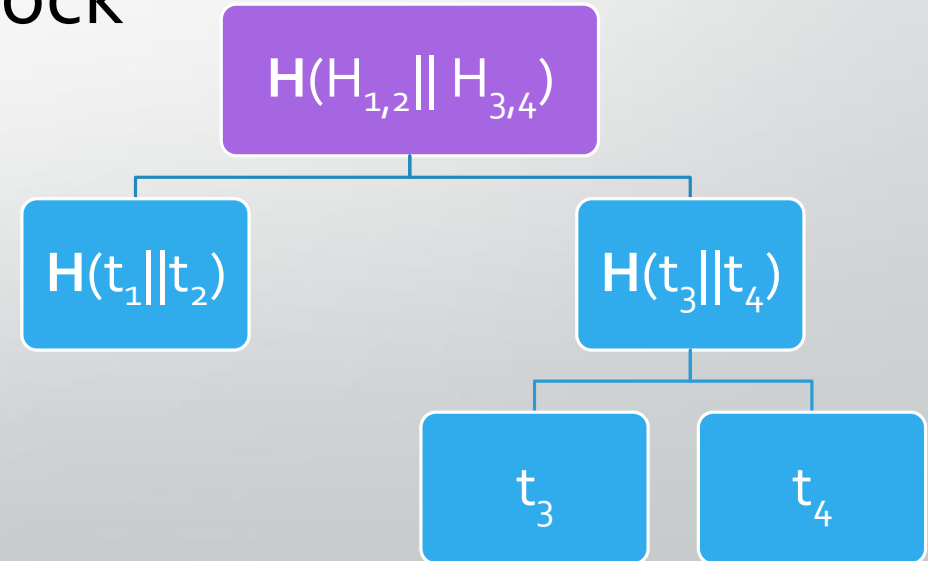
Forgetting Spent Transactions

- Suppose that t_1, t_2, t_3 have all been spent.
- We can get rid of most of them.
- Now we're storing $H(t_1 || t_2)$ 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 $H(t_1 || t_2)$ 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?