P2P Systems and Blockchains

Spring 2018,

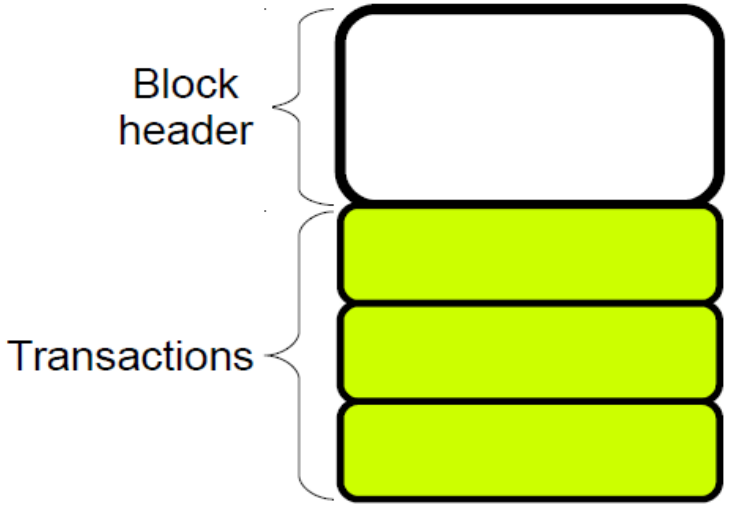
instructor: Laura Ricci
laura.ricci@unipi.it

Lesson 15: BITCOIN: BLOCKCHAIN, P2P PROTOCOL

27/4/2018

TRANSACTIONS AND BLOCKS

Transaction (tx)



- can be created by everyone who owns currency units
- transfer currency units

- encapsulates a number of transactions
- created by *miners*, include proof-of-work information
- Blocks are chained together in the blockchain

Block and transactions are disseminated to all Bitcoin users in the P2P network via a gossiping protocol

THE COINBASE

- the first transaction in the block
- does not consume (spend) previous unspent outputs contained in the blockchain
- a single, dummy input, called coinbase : no linked to any output
  - creates bitcoins from nothing.
    - Outputs value : the sum of outputs is equal to the block reward + the transaction fees.
    - Output addresses : one or more of the miner's own bitcoin address.
- Satoshi included a message in the coinbase of the genesis block **"The Times 03/Jan/2009: Chancellor on brink of second bailout for banks."**

STRUCTURE OF A BLOCK

- a header (80 bytes) consisting of three sets of metadata.
  - version: a way for miners to signal readiness for a soft fork (protocol change)
  - hash of the block header of the previous (last added) block of the blockchain.
    - calculated by taking all the fields of the header (version, nonce etc) together and applying a cryptographic function (SHA-256) twice
  - information related to the PoW:
    - difficulty
    - timestamp, Unix epoch time when the miner started hashing the header (according to the miner).
    - nonce

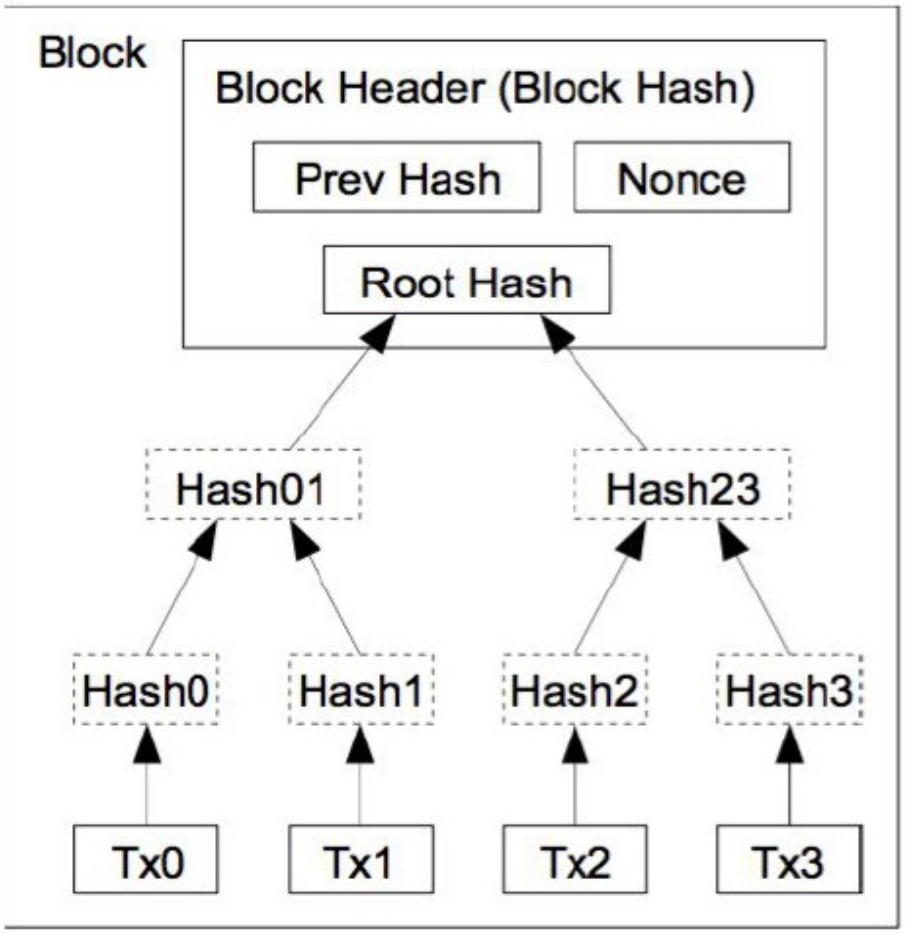
(follows in the next slide...)

STRUCTURE OF A BLOCK

- a header (80 bytes) consisting of three sets of metadata.
  - root of the Merkle tree summarizing all the transaction in the block
    - a binary balanced tree of hashes
    - constructed by recursively hashing pairs of transactions until there is only one hash, called the root, or Merkle root.
    - ensures that none of those transactions can be modified without modifying the header
    - cryptographic hash algorithm used in is SHA256 applied twice, also known as double-SHA256
    - no explicit representation of the Merkle tree in the block, it is built "on demand"

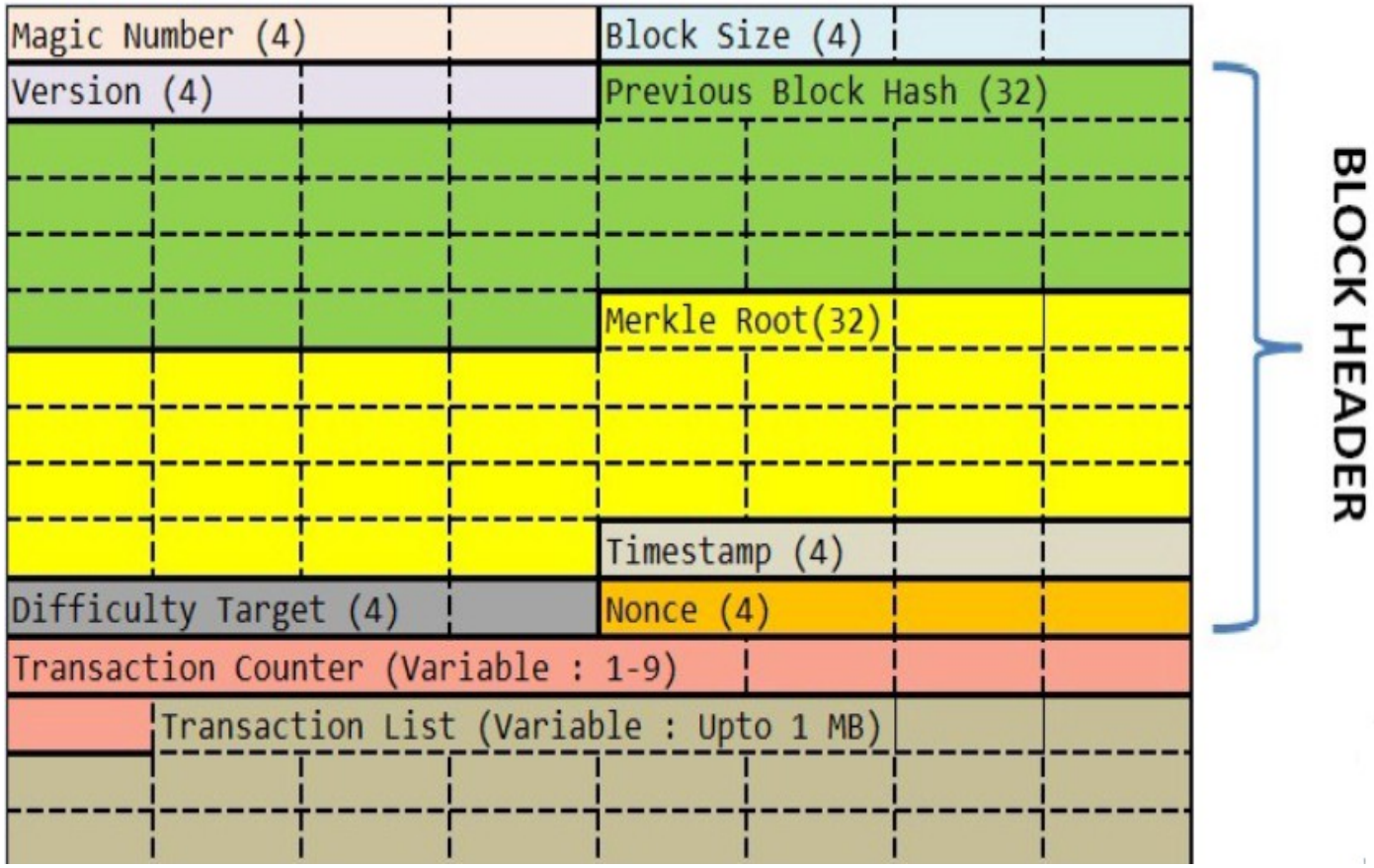
- a list of transactions that make up the bulk of its size: almost 1.000 times larger than the block header.

THE MERKLE TREE OF THE TRANSACTIONS



Transactions Hashed in a Merkle Tree

STRUCTURE OF A BLOCK



IDENTIFYING BLOCKS

- How to identify a block uniquely within the blockchain?
  - by the block hash:
    - computed by each node as the block is received from the network.
    - not actually included inside the block's data structure,
    - not computed when the block is transmitted on the network neither when it is stored on a node's persistence storage as part of the blockchain.
    - may be stored in a separate database to facilitate block indexing and retrival
  - by the block height: the position of the block in the blockchain
    - first block ever created is at block height 0 (zero)
    - blockheight 26 april 2018: 520056

WHY BLOCKS?

- miners construct a candidate block filling it with transactions.
- transaction are inserted in the block as they are received from the network
- executes the PoW: calculates the hash of this block header, including the root of the Merkle tree, the hash of the previous block and the nonce and checks if it is smalles of the current target
- block are the unit of work for miners
- if miners would compute consensus on each transaction individually:
  - the rate at which new transactions are accepted by the system would be smaller
  - a hash chain of blocks is shorter than a hash chain of transactions: the block chain data structure can be verified faster.

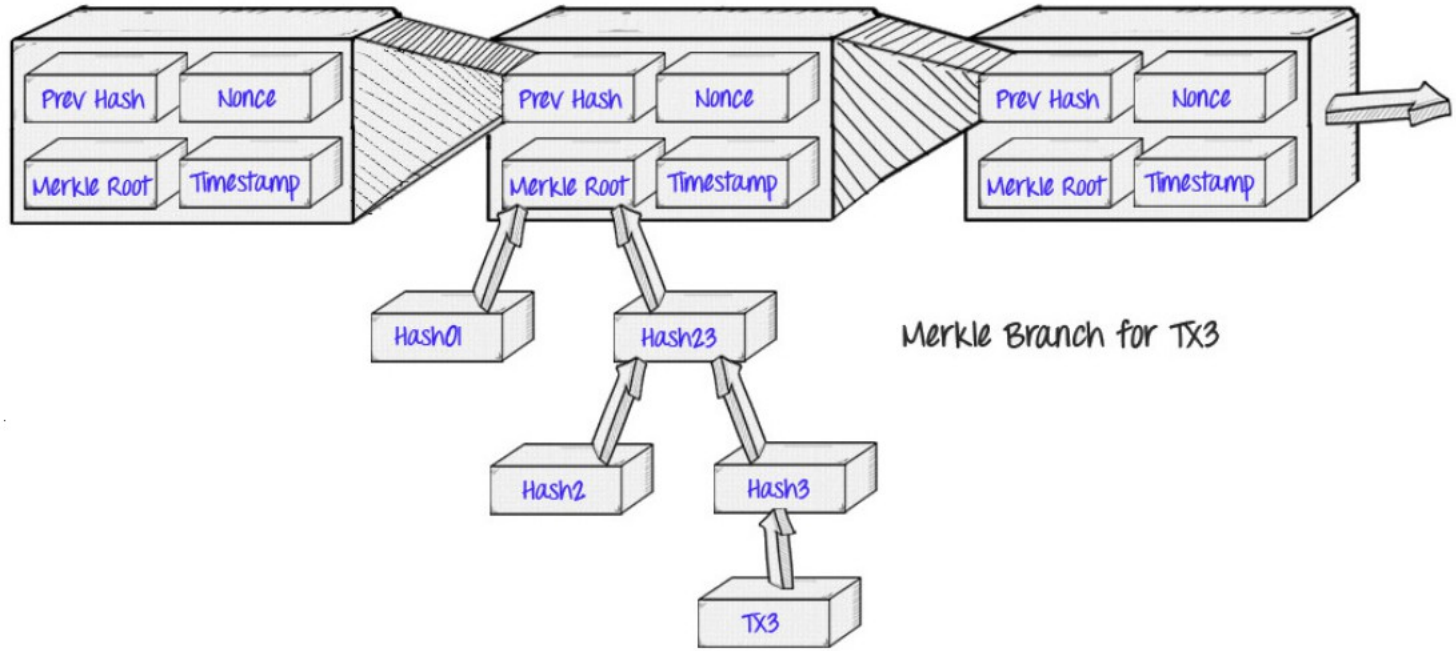
WHY MERKLE TREES?

- Efficient hashing of the block
  - PoW computed on block header instead of on all the transactions: the Merkle root is a digest of the transactions.
  - rehasing more efficient when a new transaction is received
- at each mining round, the miner takes into consideration the very latest transactions received and add them to the block
- instead of reahashing all the transactions, compute the new root of the Merkle tree and execute the PoW by considering this new root
  - computing the new root requires *log(n)*, where n is the amount of transactions in the block
  - only a few operations to recompute all the hashes along the branches on the path from the new transaction to the root.

TYPE OF NODES IN BITCOIN

- Miner
  - run on powerful or specialized hardware
  - mine new blocks by solving Proof of Work
- Full node.
  - validate blocks mined by miners and verify transactions.
  - store the whole copy of blockchain.
  - perform such routing operations, like helping other nodes to discover each other
  - stores a complete and up-to-date copy of the blockchain on the disk (about 140 G, December 2017), with all the transactions
- SPV: Simplified Payment Verification
  - don't store a full copy of blockchain, only the header
  - verify transactions a subset, for example, those that were sent to specific address
  - depend on a full node and get data from it.

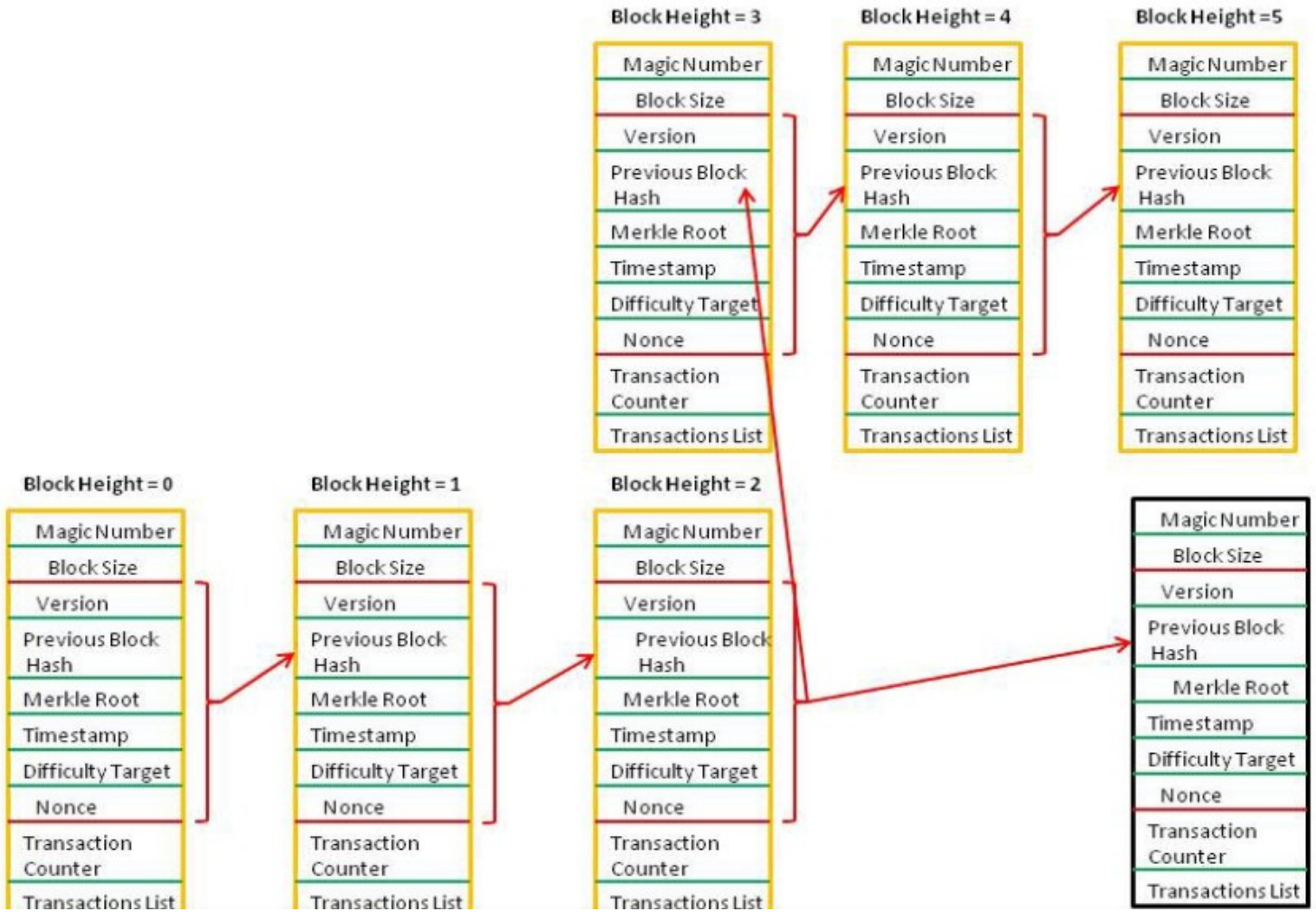
WHY MERKLE TREES?



Merkle trees provide a very efficient process to verify whether a transaction is included in a block

- the SPV has the block header (verified by the miners)
- asks to the full node for the Merkle verification branch for TX3
- computes the root hash from TX3 and the Merkle verification branch and then compare with the root hash in the header

THE BITCOIN BLOCKCHAIN



BLOCKCHAIN TAMPER FREENESS

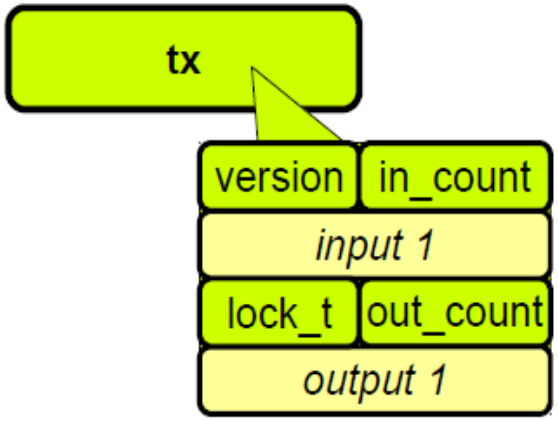
- the blockchain: an append only distributed data structure which is immutable, tamper free
- imaging an attacker changing the content of a block, for instance a transaction
  - it also change the root of the Merkle tree
  - the nonce of thet block is no more valid
  - a new PoW has to be executed to re-compute the right nonce
- the hash pointer in the successor block has to be changed, because now the header of the block is changed
- and then the pointer of the successor of the successor must be changed and so on
- an attacker would have to recompute the PoW for the entire chain
  - this would require an enormous computing power.

THE BITCOIN PROTOCOL: RECAP

Alice wants to send bitcoins to Bob

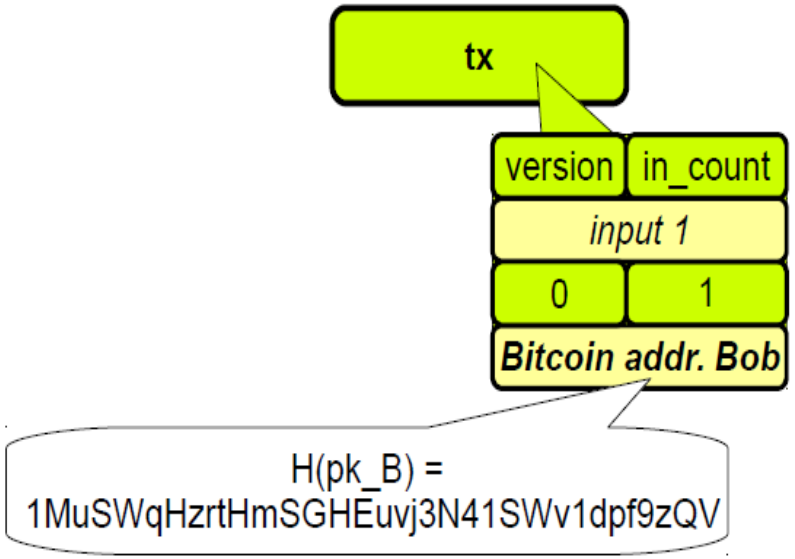
THE BITCOIN PROTOCOL: RECAP

Alice wants to send bitcoins to Bob



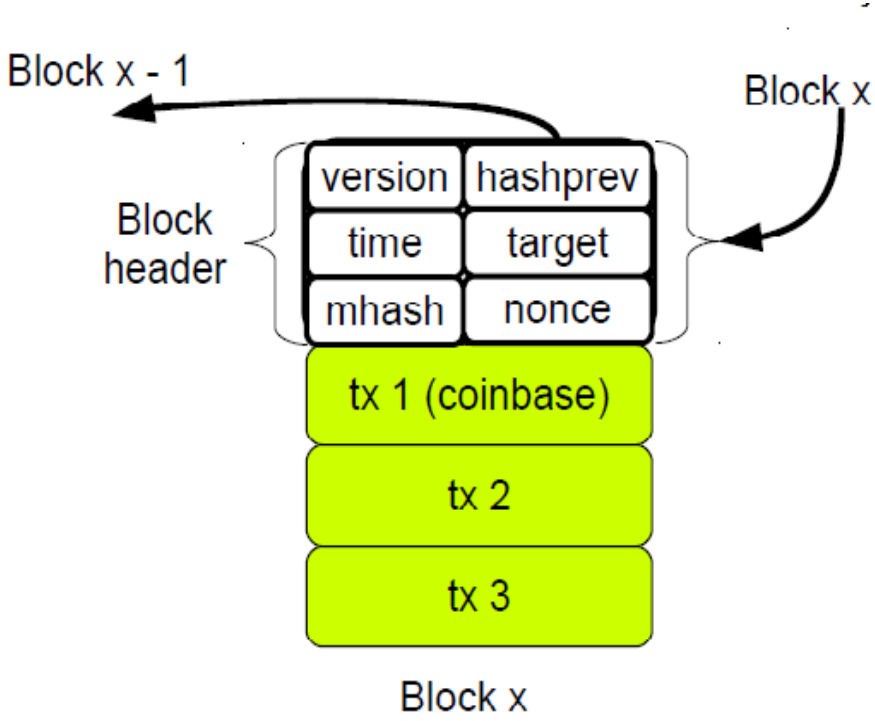
THE BITCOIN PROTOCOL: RECAP

Alice wants to send bitcoins to Bob

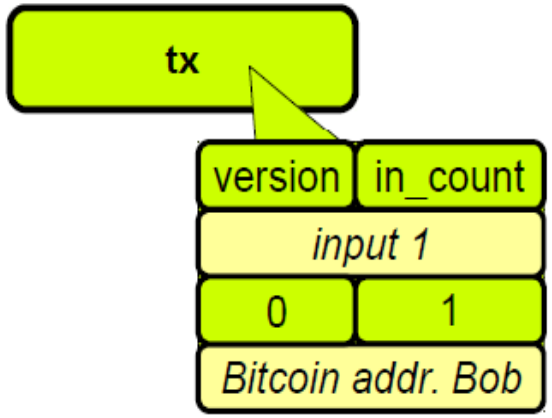


THE BITCOIN PROTOCOL: RECAP

Alice wants to send bitcoins to Bob

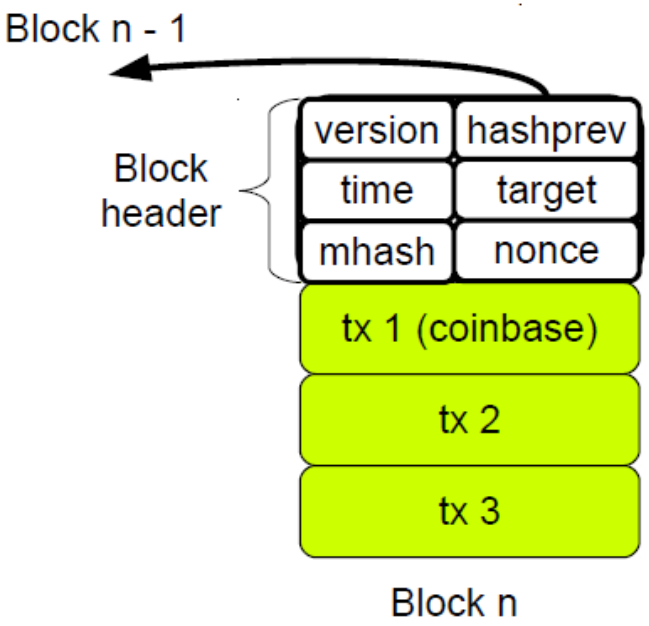


- search block with tx that has an output to Alice
- can be anywhere in the blockchain



THE BITCOIN PROTOCOL: RECAP

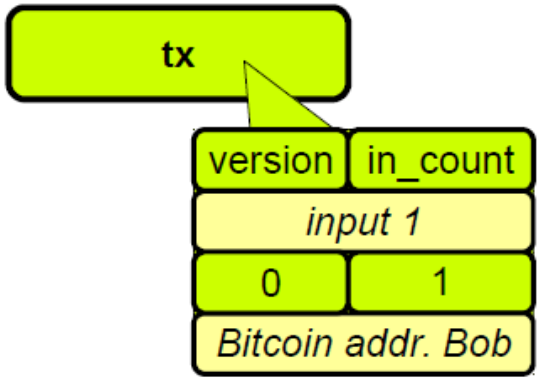
Alice wants to send bitcoins to Bob



in this case in block n

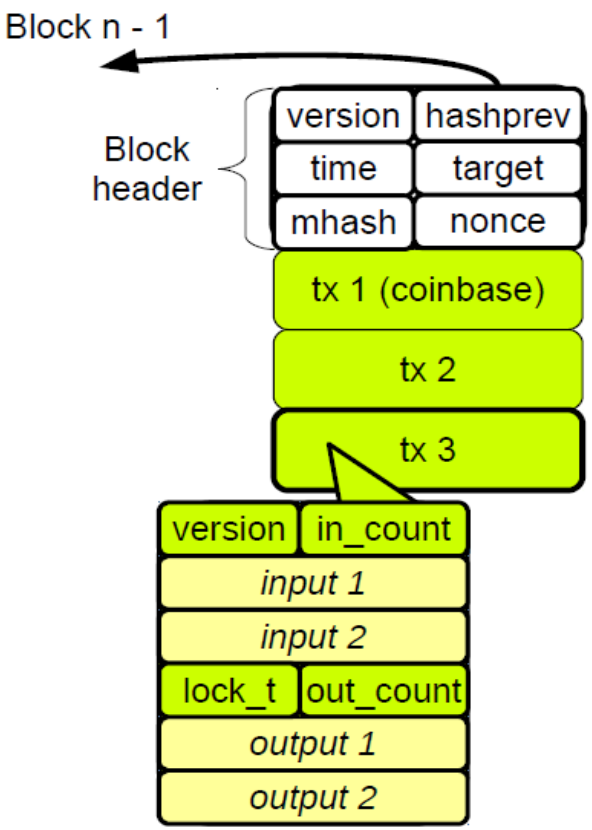
i.e. the current head of

the chain

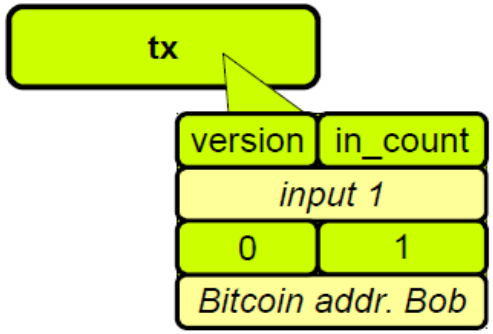


THE BITCOIN PROTOCOL: RECAP

Alice wants to send bitcoins to Bob

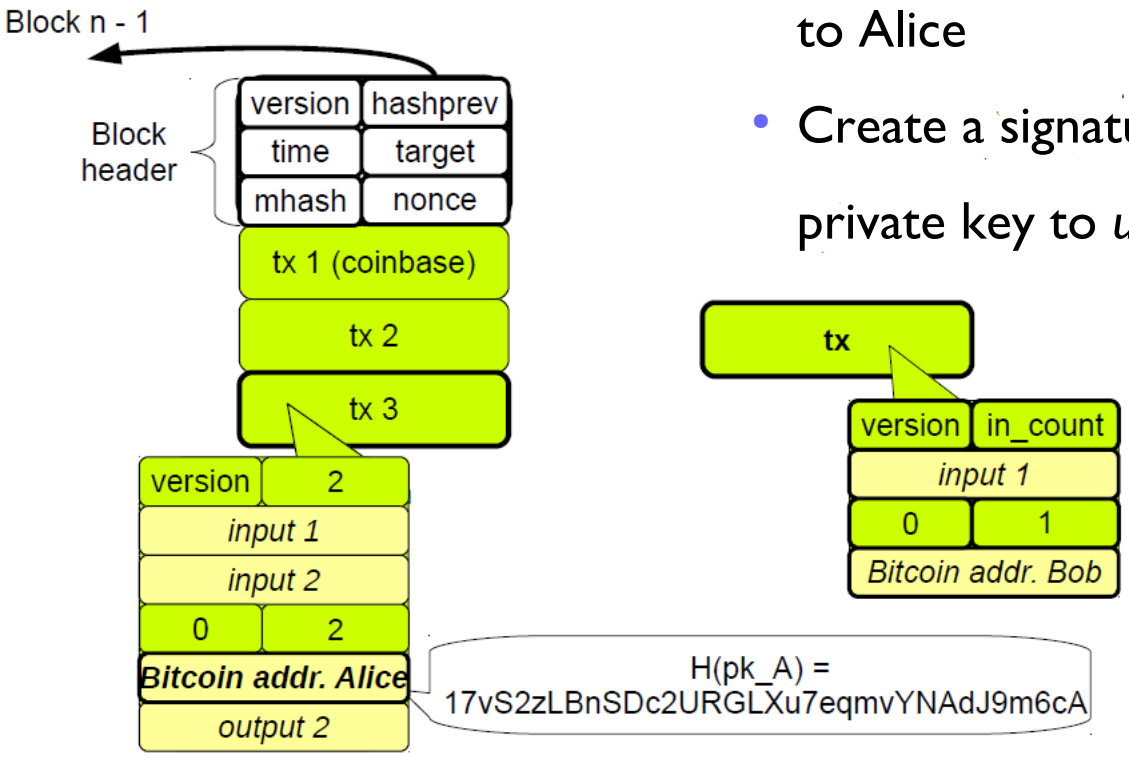


- Needs to *unlock* this previous output to Alice



THE BITCOIN PROTOCOL: RECAP

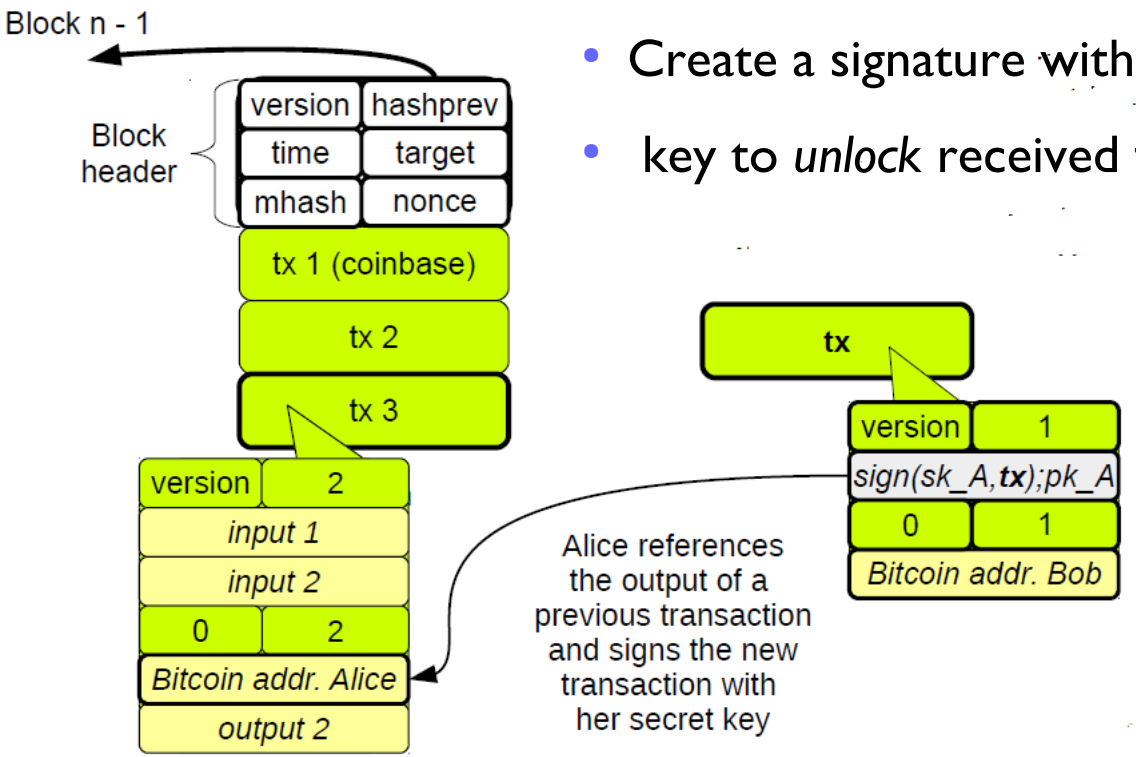
Alice wants to send bitcoins to Bob



- Need to *unlock* this previous output to Alice
- Create a signature with Alice private key to *unlock* received funds

THE BITCOIN PROTOCOL: RECAP

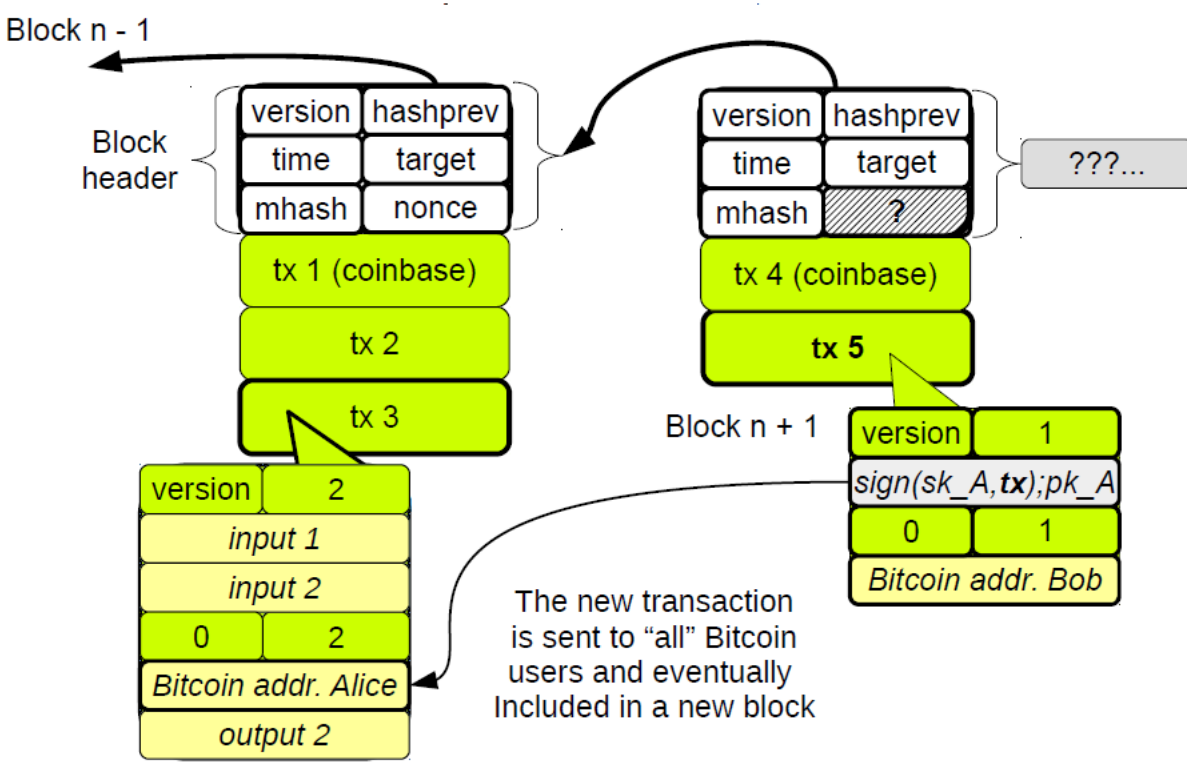
Alice wants to send bitcoins to Bob



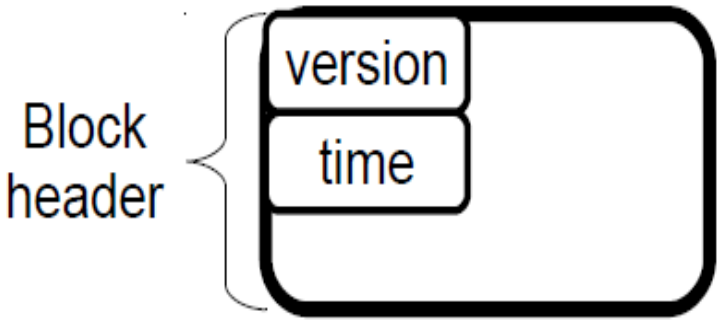
- Need to *unlock* this previous output to Alice
- Create a signature with Alice private
- key to *unlock* received funds

THE BITCOIN PROTOCOL: RECAP

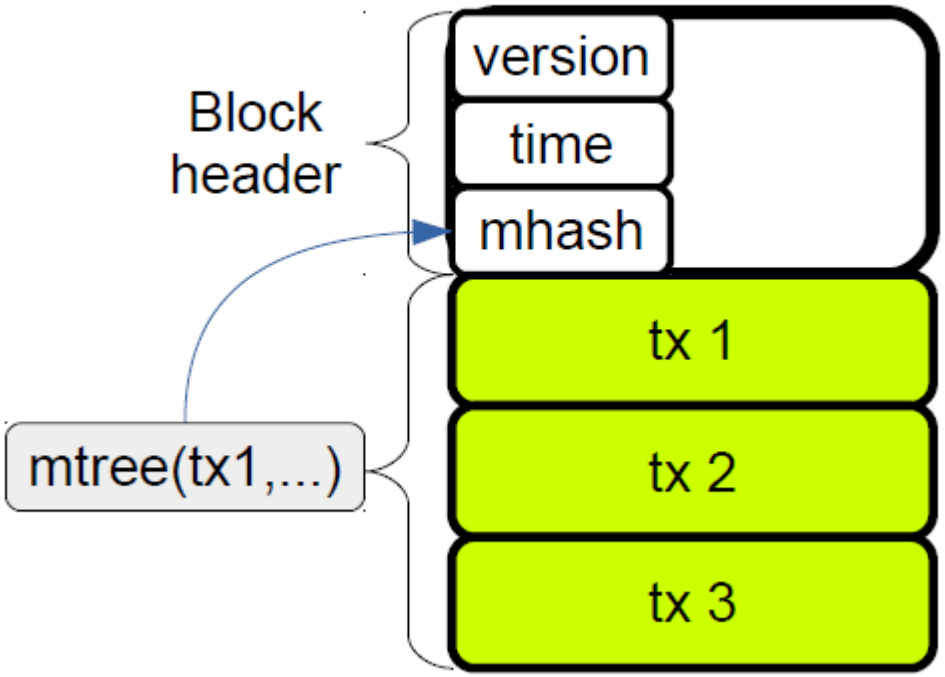
Alice wants to send bitcoins to Bob



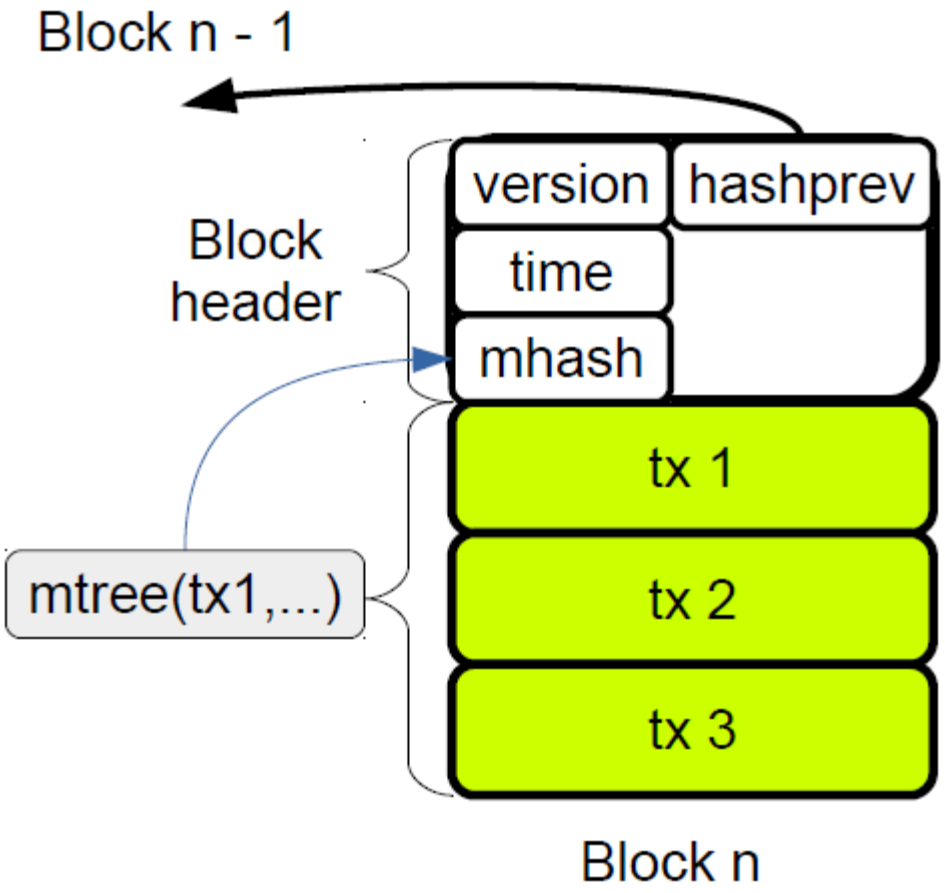
THE BITCOIN PROTOCOL: RECAP



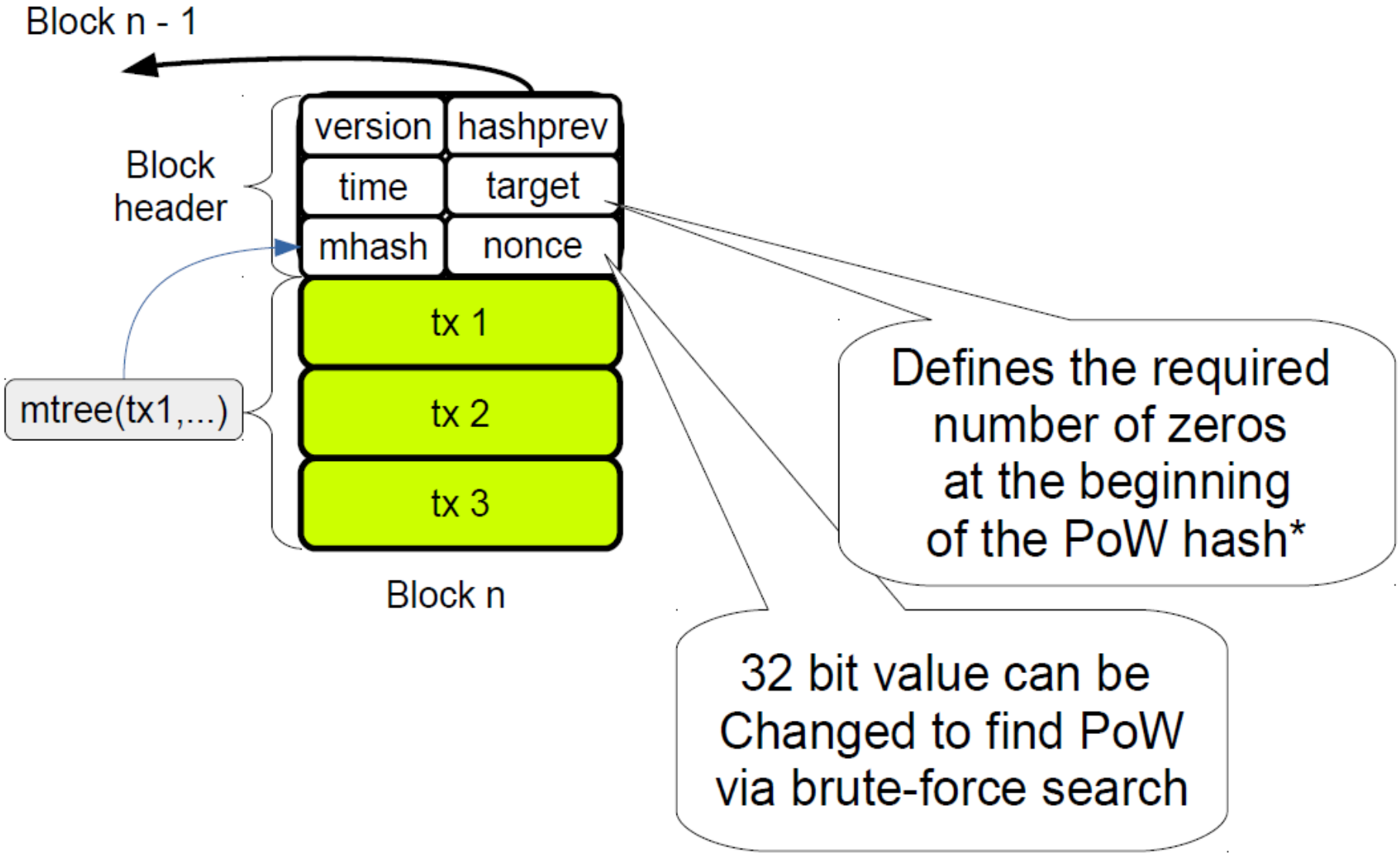
THE BITCOIN PROTOCOL: RECAP



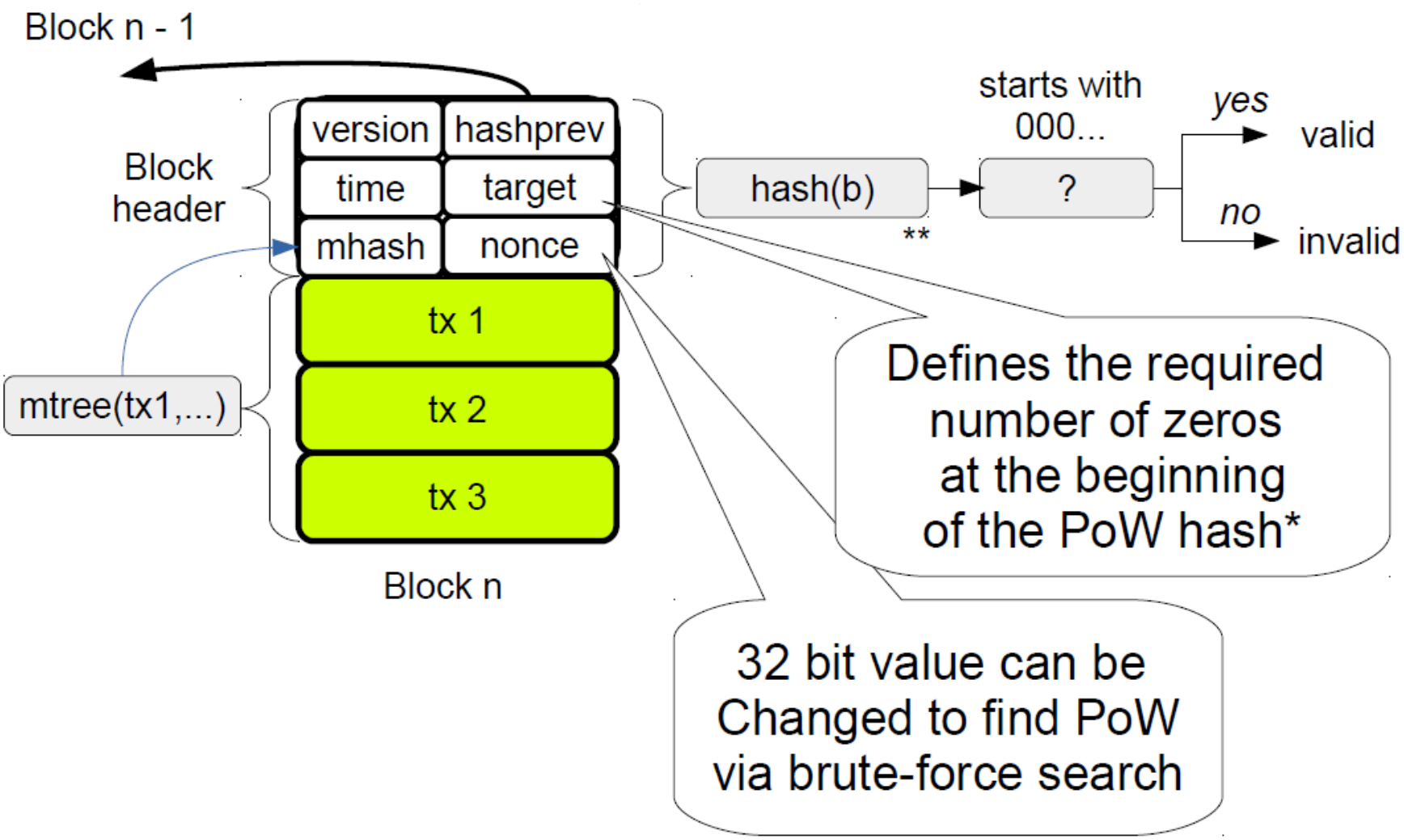
THE BITCOIN PROTOCOL: RECAP



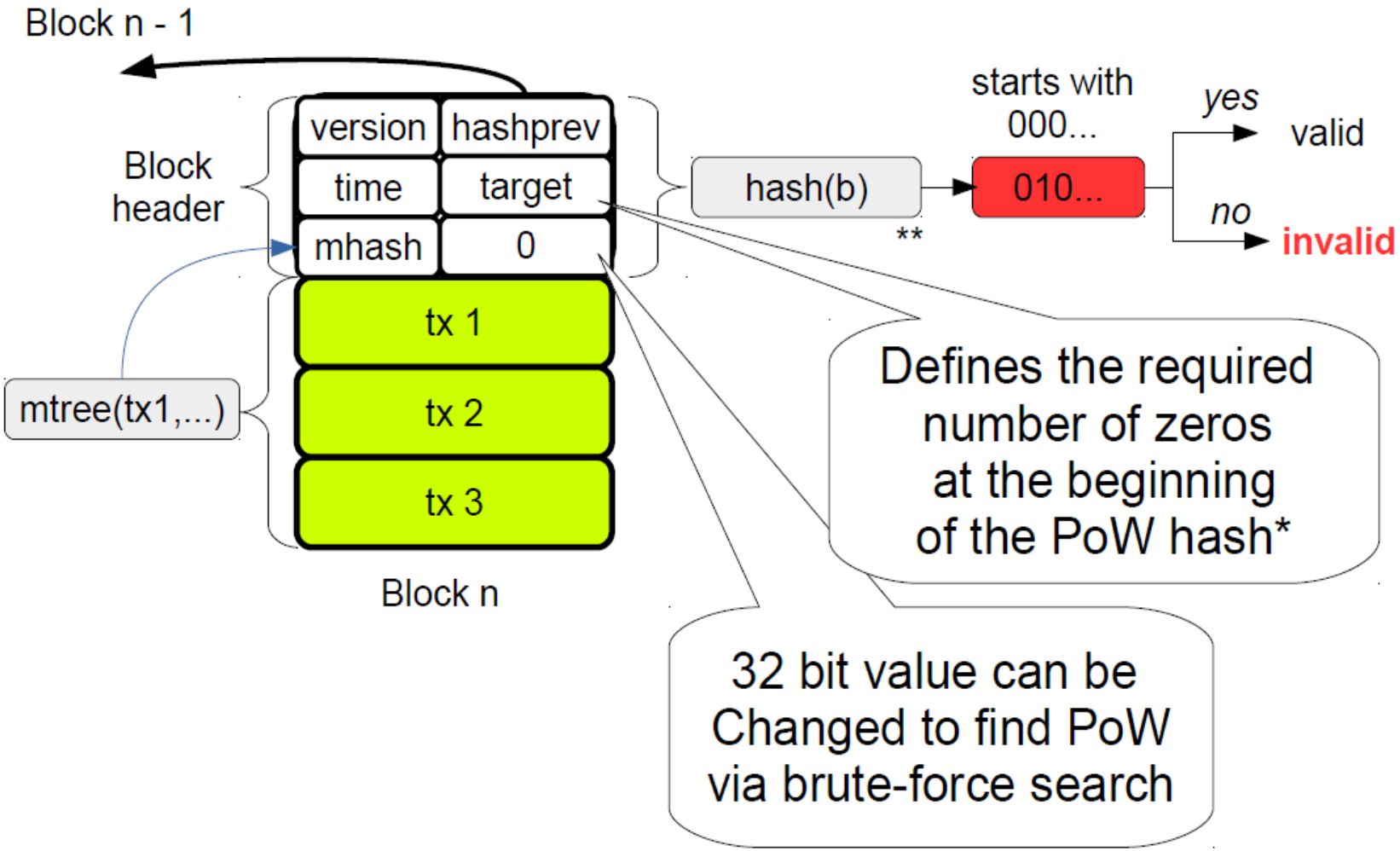
THE BITCOIN PROTOCOL: RECAP



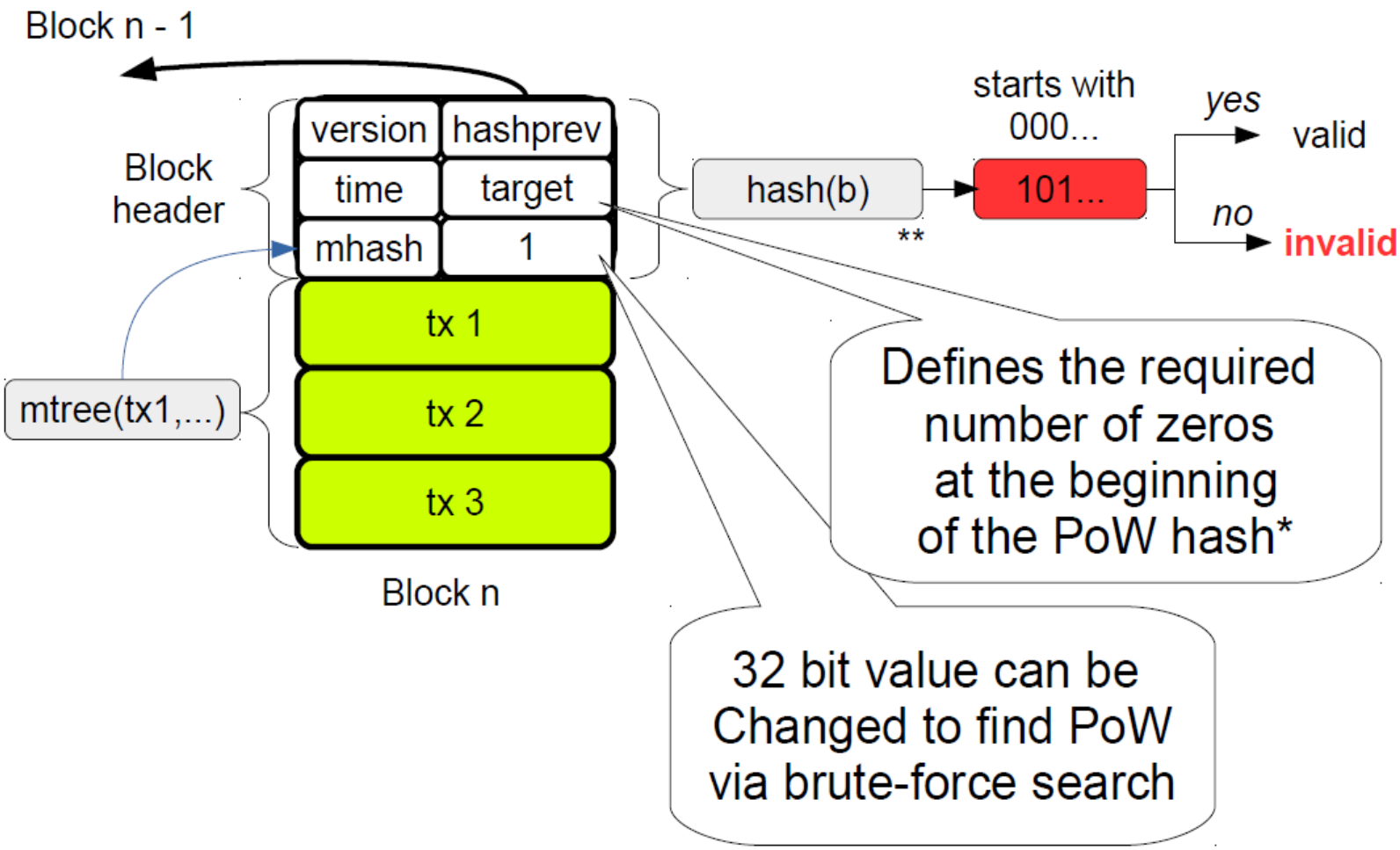
THE BITCOIN PROTOCOL: RECAP



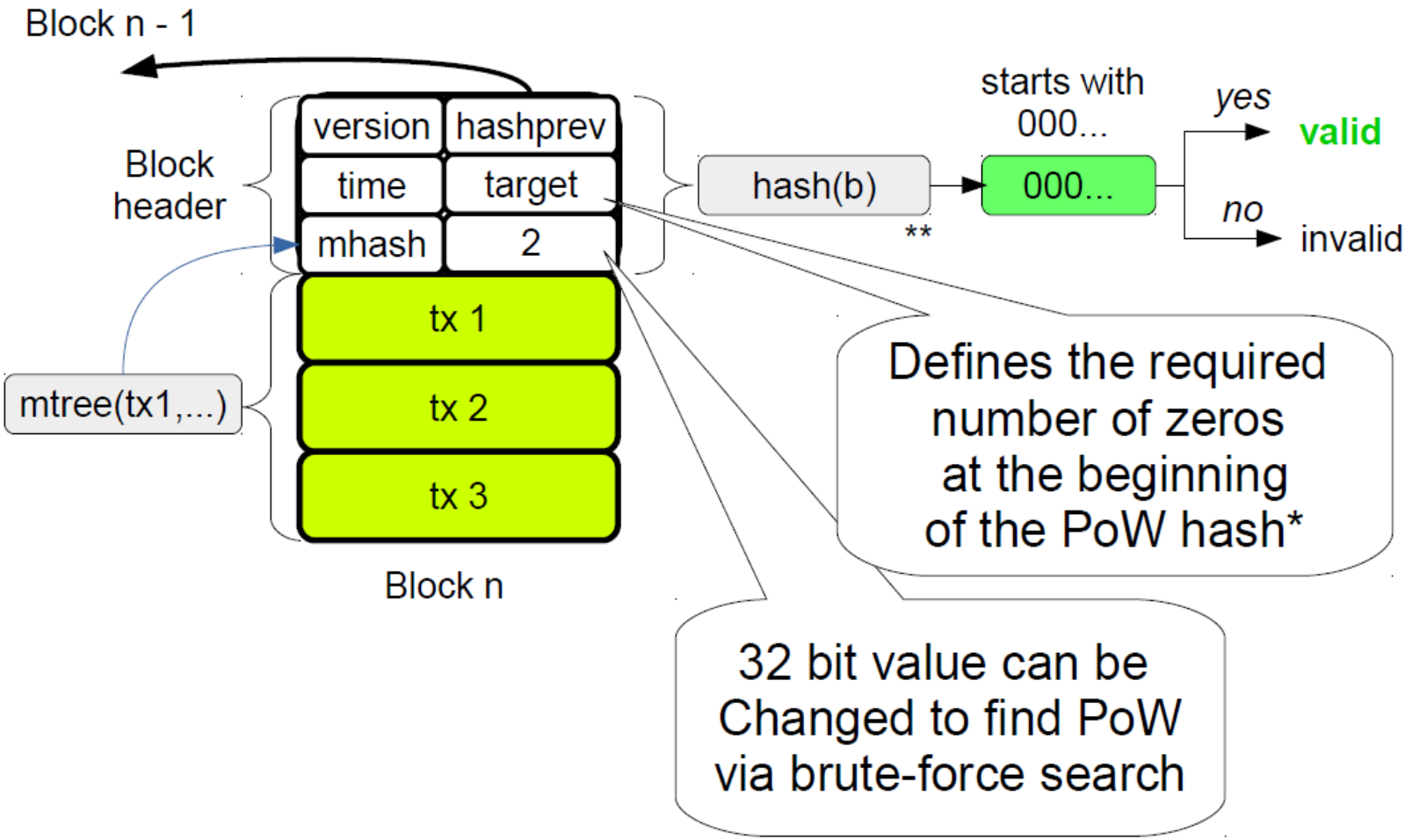
THE BITCOIN PROTOCOL: RECAP



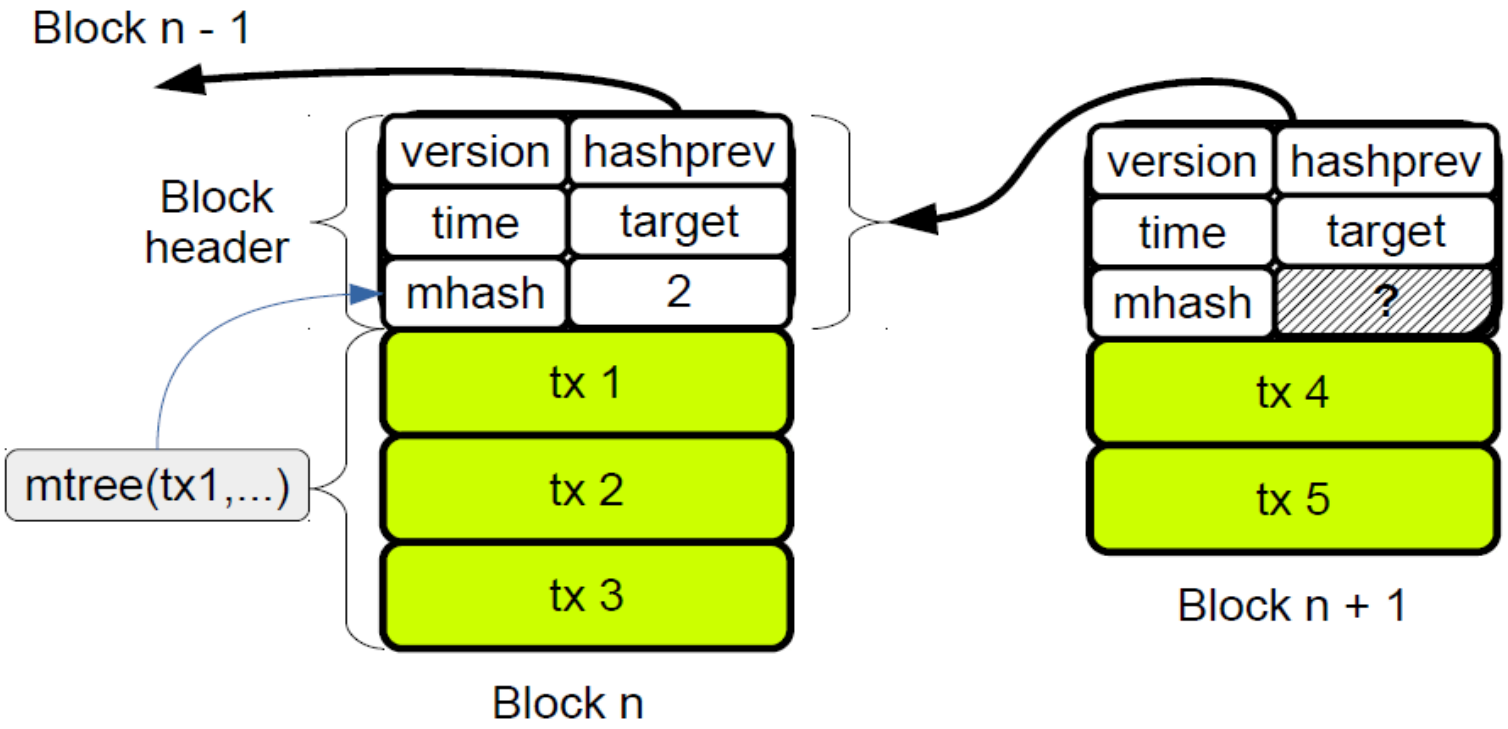
THE BITCOIN PROTOCOL: RECAP



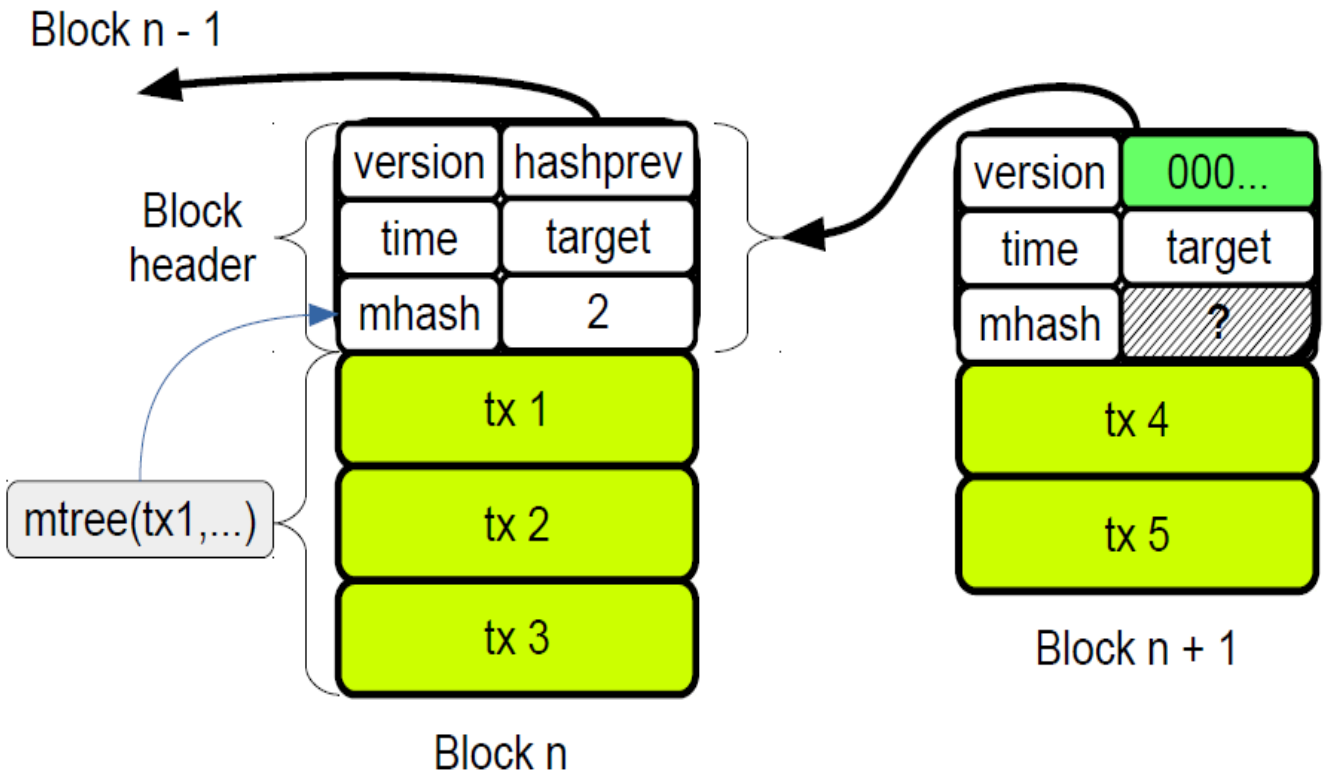
THE BITCOIN PROTOCOL: RECAP



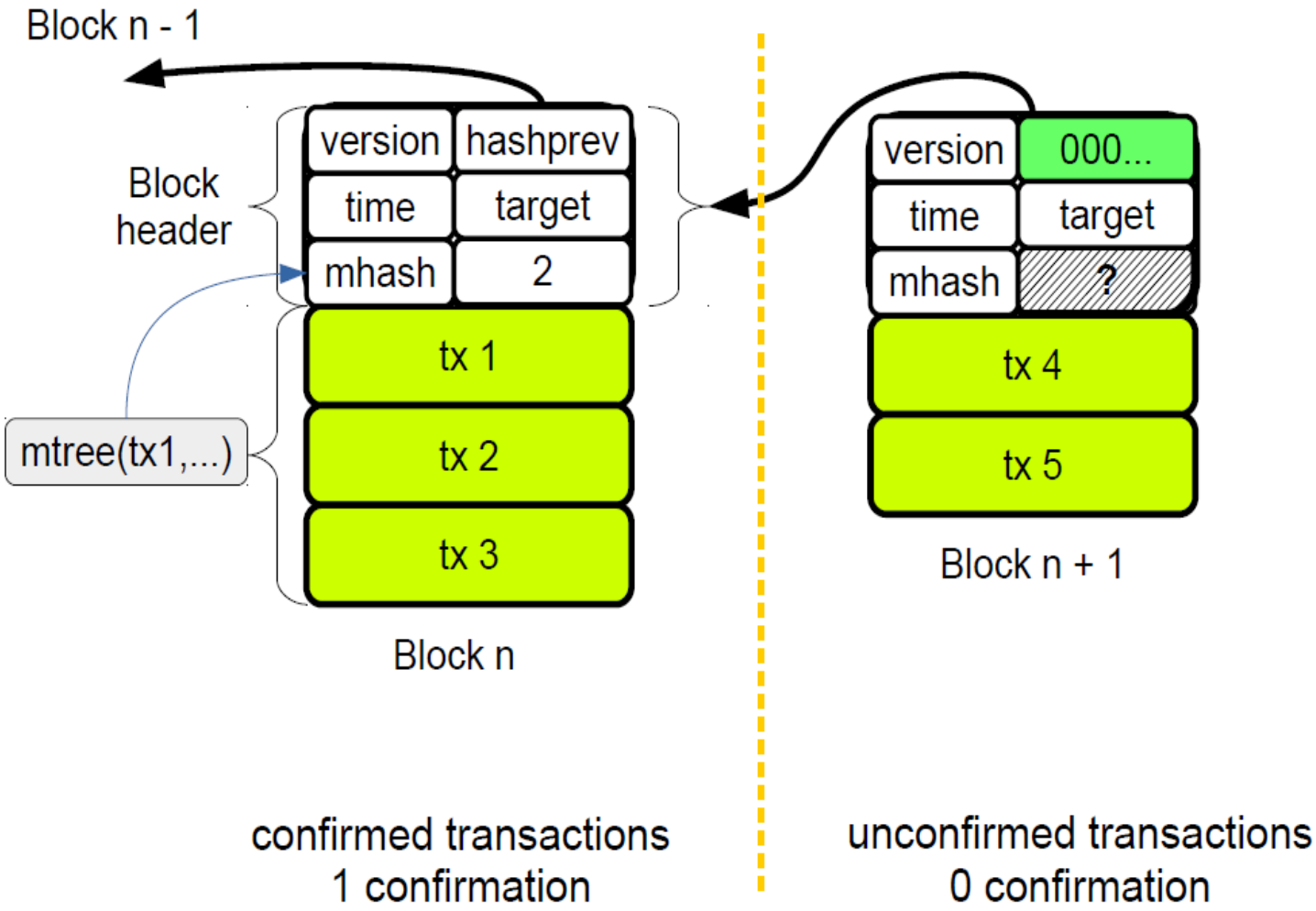
THE BITCOIN PROTOCOL: RECAP



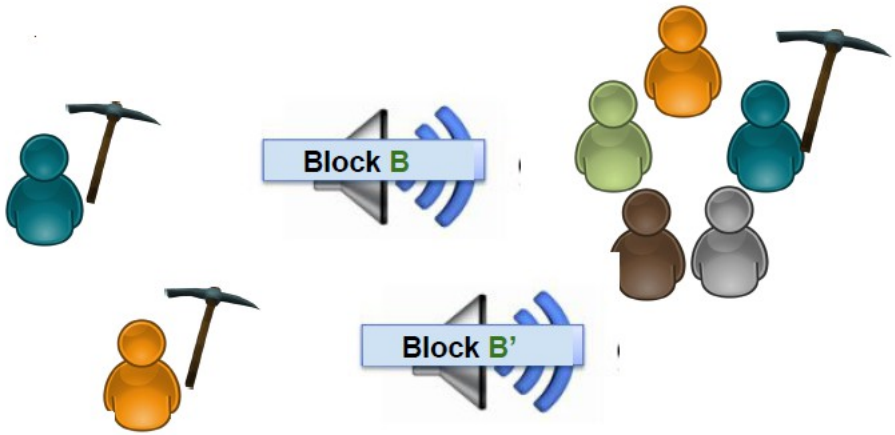
THE BITCOIN PROTOCOL: RECAP

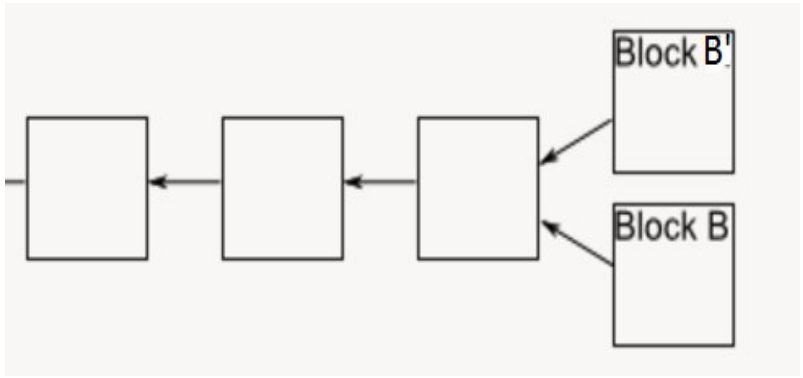


THE BITCOIN PROTOCOL: RECAP



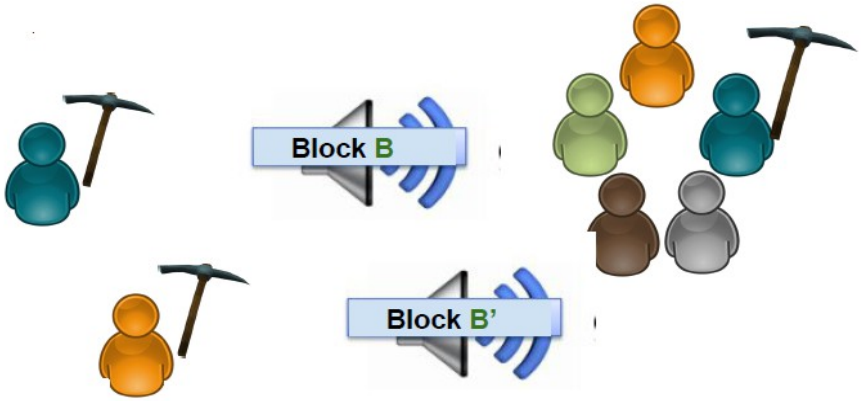
BLOCKCHAIN FORKS

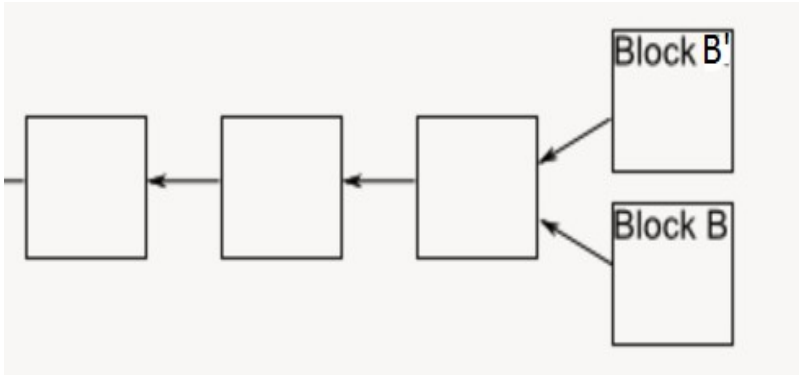




- it may happen that two miners validate a block of transactions near-simultaneously
  - the new blocks extend the same block of the blockchian and both contain an hash pointer to the same previous block of the blockchain
- The two miners both broadcast their newly-validated block:
  - a fork occurs in the block chain
  - no total order of transactions: it may not be clear who owns which bitcoins.

BLOCKCHAIN FORKS

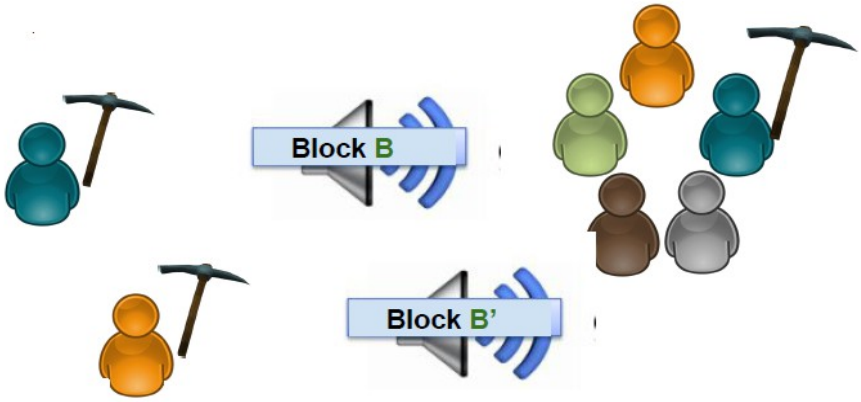


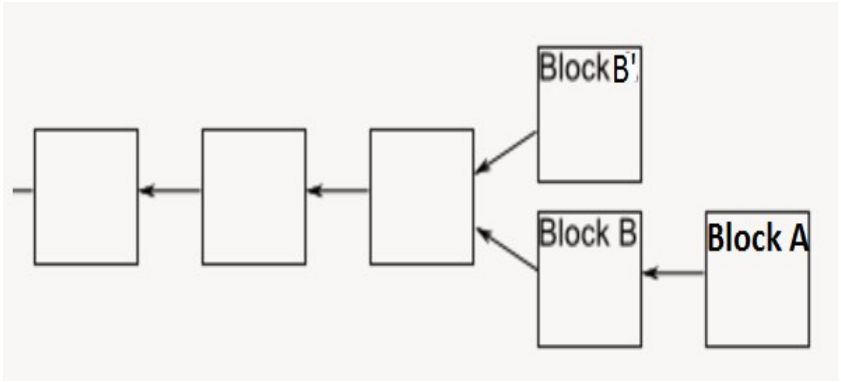


The rule to remove forks

- nodes keep track of the blocks in both the forks
- miners only work to extend the longest fork in their copy of the block chain
  - miners receiving block B first will continue mining along that fork, while the others will mine along fork B'

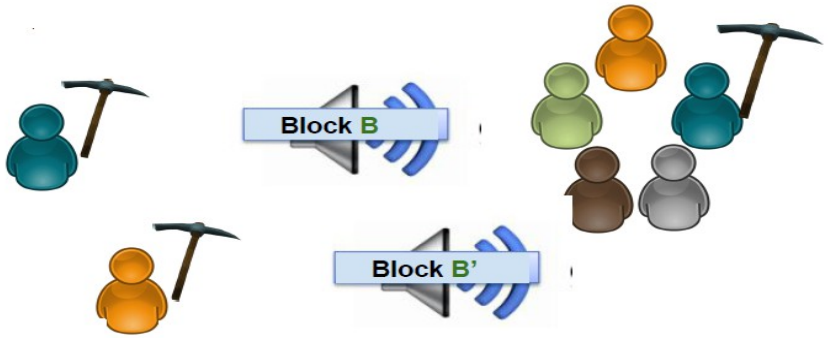
BLOCKCHAIN FORKS

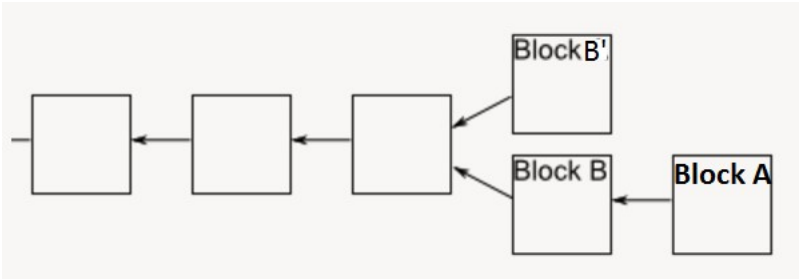




- if one of the miners working on B is the next one to successfully mine a new block
- miners working on B' receive the new block
  - since the fork including B is now longer they switch to working on that branch of the fork
  - in short time, the branch containing B' will cease to work and B' will become a orphaned block
  - everyone will be working on the same linear chain,

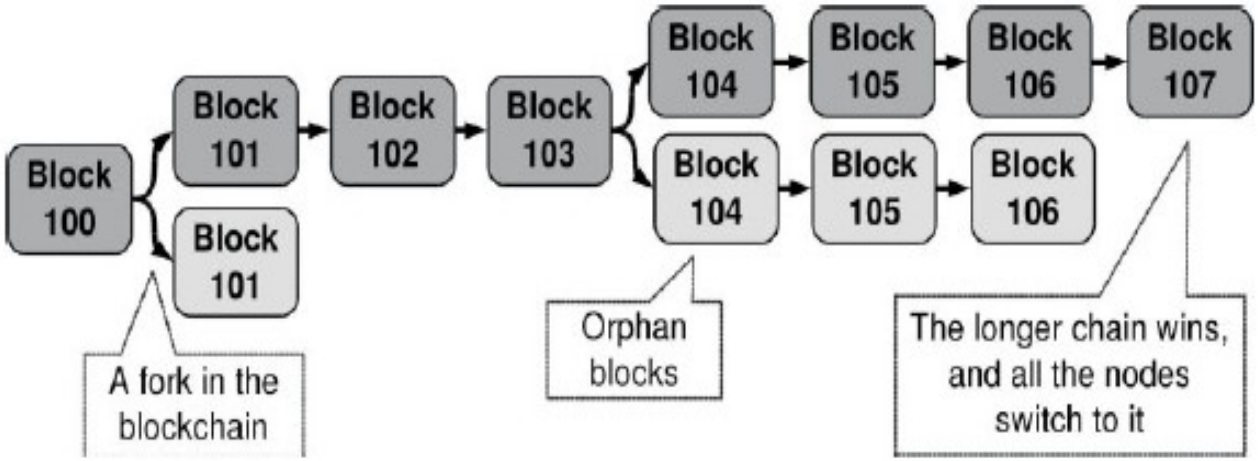
BLOCKCHAIN FORKS





- when miners working on branch B' receive a new mined block
  - any still-pending transactions in B' are still be pending in the queues of the miners working on branch B
  - all transactions will eventually be validated.

THE GENERAL STRUCTURE



- actually a tree of blocks, instead of a blckchain
  - the path in the tree corresponding to the longest chain is the block chain
- dead path contain orphan blocks
- block height: the order of a block in the longest path, starting from the genesis block (first block)
- blockchain head: the last block added to the blockchain

NAKAMOTO CONSENSUS

Theorem: forks are eventually resolved and all nodes eventually agree on which is the longest blockchain. The system therefore guarantees eventual consistency.

Proof sketch:

- in order for the fork to continue to exist, pairs of blocks need to be found in close succession, extending distinct branches
- otherwise the nodes on the shorter branch would switch to the longer one
- the probability of branches being extended almost simultaneously decreases exponentially with the length of the fork
- hence there will eventually be a time when only one branch is being extended, becoming the longest branch.

ALGORITHM EXECUTED WHEN RECEIVING A BLOCK

Receive block b

For this node the current head is block b_{max} at height h_{max}

Connect block b in the tree as child of its parent p at height

$h_b = h_p + 1$

if h_b > hmax then

$h_{max} = h_b$

$b_{max} = b$

compute UTXO for the path leading to b_{max}

cleanup memory pool

end if

UTXO: Unspent Transaction Cache (see previous lesson)

ATTACKS TO THE PROTOCOL

What can a malicious node do?

- Stealing bitcoins
- Denial of service
- Double spending

WHAT CAN A MALICIOUS NODE DO? STEALING BITCOINS

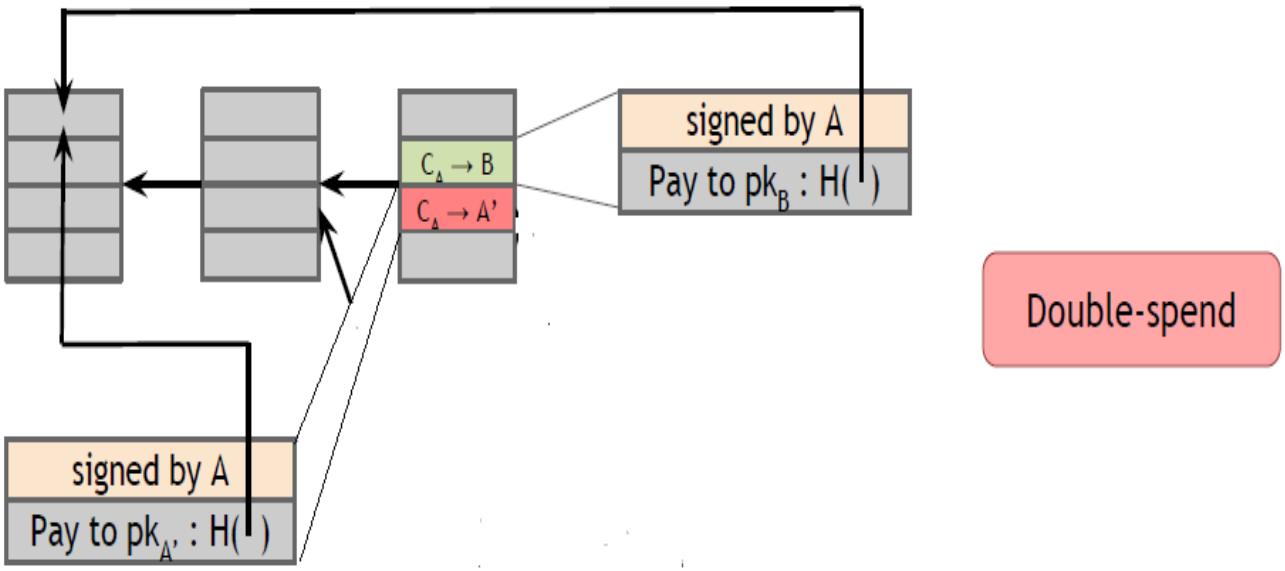
Miners can steal bitcoins of transactions belonging to the block they have created?

- this would require to create and insert a valid transaction that spend those coins.
- to do this, the miner would forge the bitcoin owners' signatures
- impossible if a secure digital signature scheme is used
- cryptography guarantees that this attack cannot succeed

DENIAL OF SERVICE

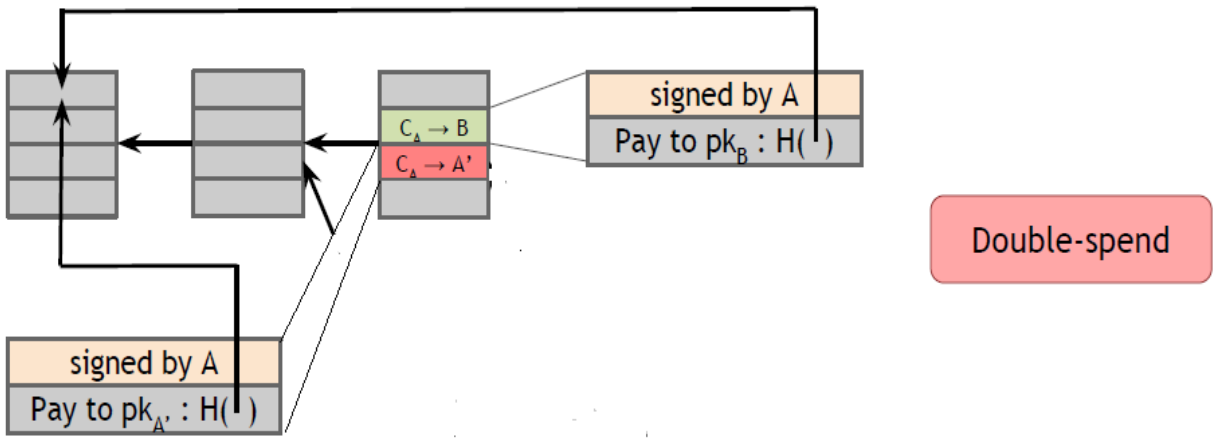
- a miner, say Alice, really dislikes some other user, say Bob, and denies service to him
- Alice decides that she will not include any transactions from Bob's addresses in any block that she proposes for the block chain
- but...if Bob's transaction is not inserted into the next block that Alice proposes, it remains in the pool of unconfirmed transactions
- Bob
  - will just wait until that an honest node proposes a new block,
  - his transaction will get into that block, so being confirmed
  - the block will be inserted in the blockchain.

DOUBLE SPENDING BY MINING A FRAUDOLENT BLOCK



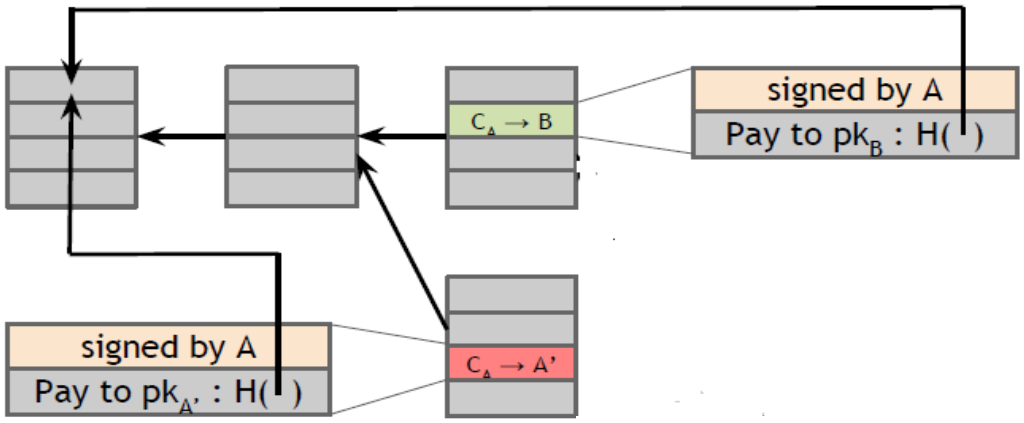
- be aware of pointers and do not confuse the two different types of hash pointers in the figure !
  - blocks include a hash pointer to the previous block that they are extending
  - transactions include one or more hash pointers to previous transaction outputs that they are now spending

DOUBLE SPENDING BY MINING A FRAUDOLENT BLOCK



- a naive double spending attack: Alice(A) spends the same coins with B and with herself (A' one of A's addresses)
- A is a miner and validates a block B that includes both transactions
  - other nodes they check the validity of B and reject it.
  - the attack does not suceed, despite A has solved the proof-of-work
    - no rational miner will perform this attack!

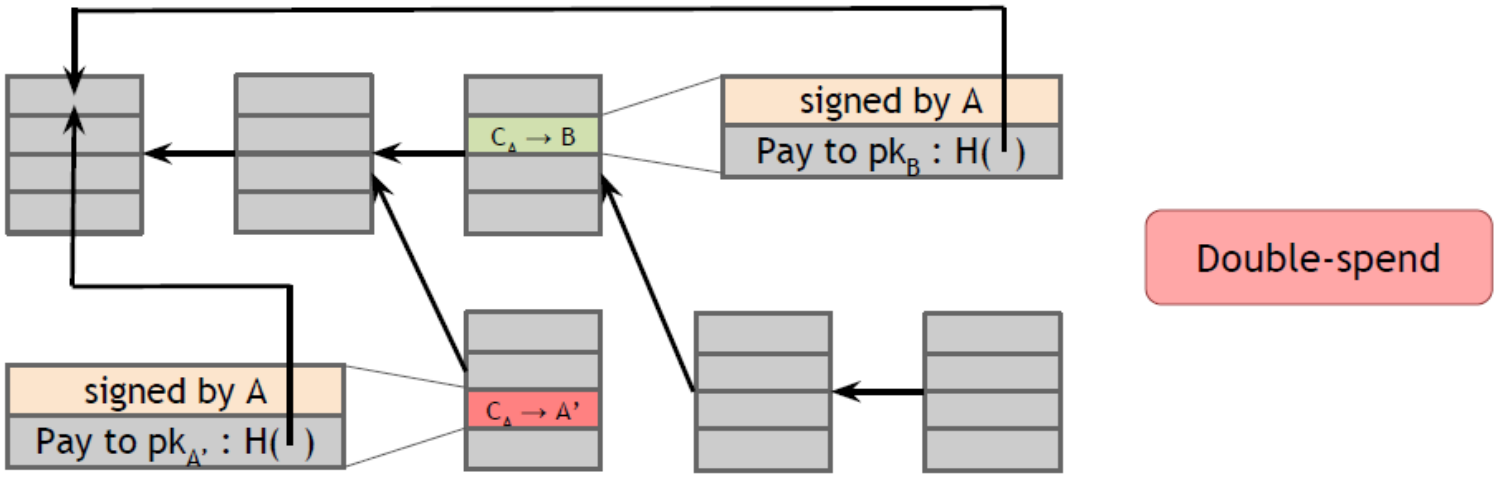
DOUBLE SPENDING BY CHAIN FORKING



- the latest block generated by an honest node includes a transaction in which Alice (A) pays Bob (B) for some service.

- A is a miner and proposes next block, and deliberately forks the chain to perform double spending
  - ignores the block that contains the payment to B
  - proposes a block including a pointer to the same previous transaction outputs
  - the transaction transfers to herself of the same bitcoins spent with B

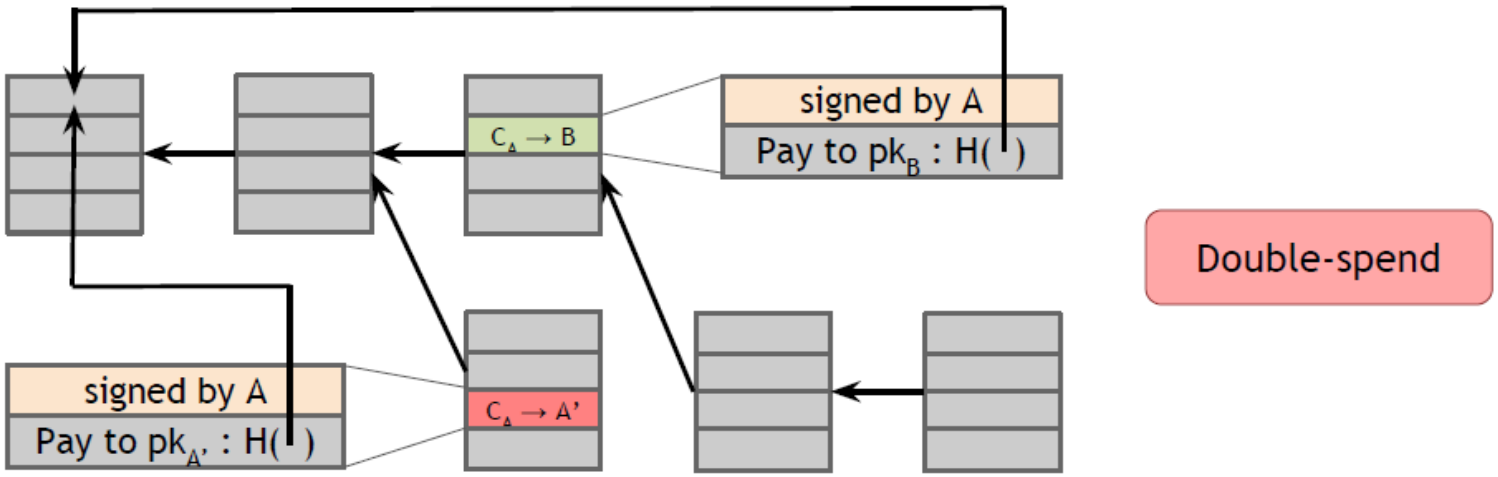
DOUBLE SPENDING BY CHAIN FORKING



The attack does not succeed!

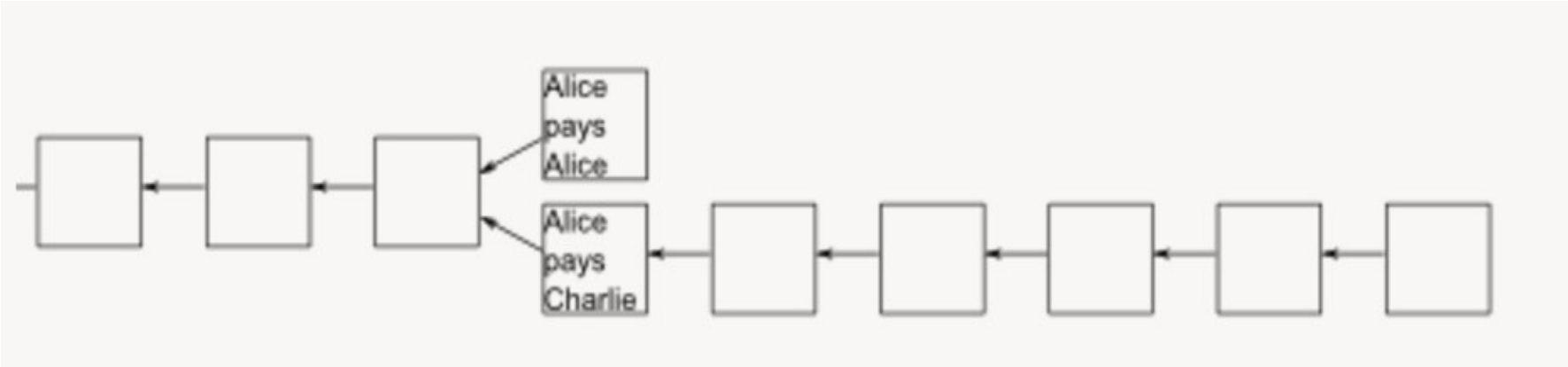
- only one of the two blocks will ultimately be inserted on the long-term consensus chain, depending from which fork will be extended
- the two blocks are completely identical and valid from the point of view of the protocol
  - the succesfull chain may contain the transfer of the bitcoin to A herself

DOUBLE SPENDING BY CHAIN FORKING



- if the fork containing the transaction to A herself is extended, B will not receive the bitcoins
- B must be careful to not deliver the service to A until he is sure that the transaction will be included in the longest chain
  - wait before delivering the service!

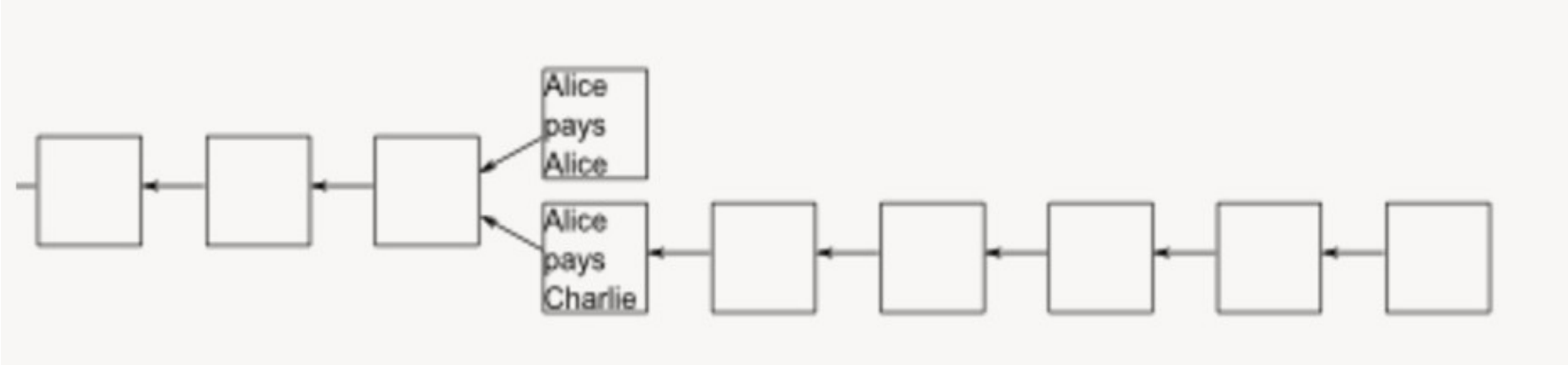
SIX CONFIRMATION RULE



in Bitcoin, a transaction is not considered confirmed until:

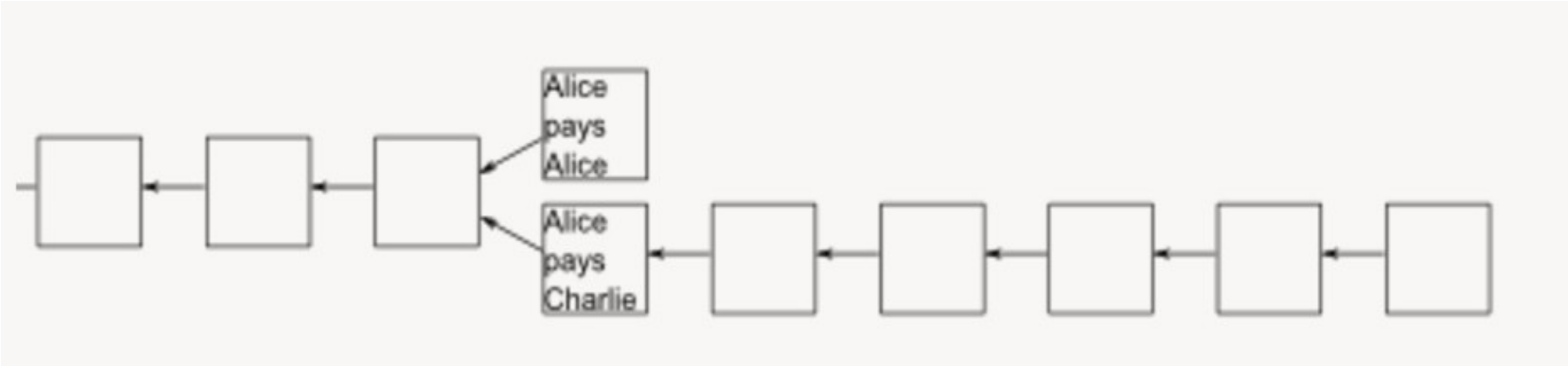
- 6 confirmations rule: at least 5 blocks follow it in the longest fork
- 6 as default value, but the number of confirmations may also be decided through the client
- giving the network time to come to an agreement upon the ordering of the blocks
- Charlie delivers the service to Alice, because it has received 6 blocks after that including the transaction from Alice

SIX CONFIRMATION RULE



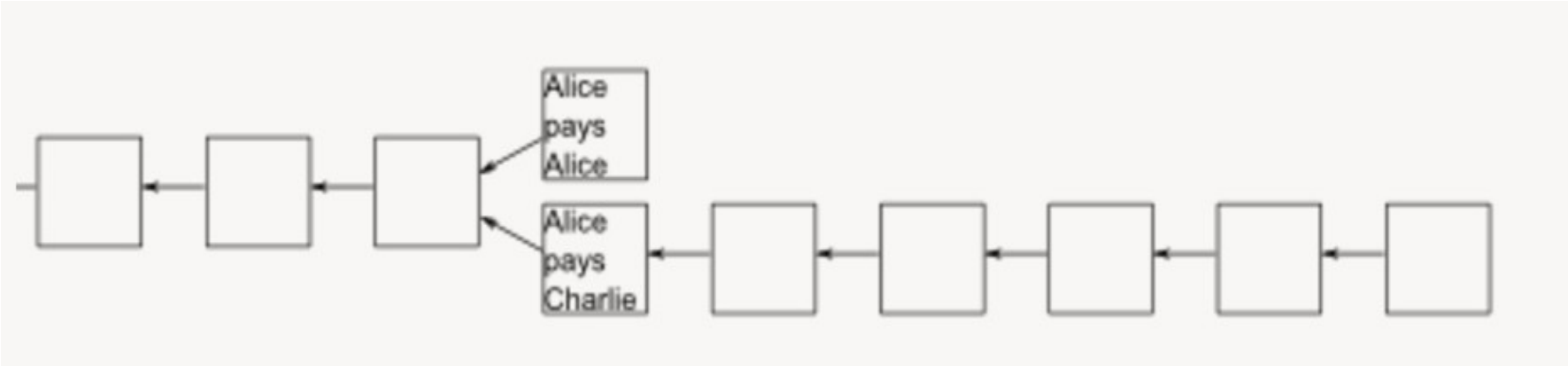
- Does Alice's attack have still some chances to succeed?
- Alice's may wait until Charlie accepts the bitcoin: this happens after the transaction has been confirmed 6 times in the longest chain.
  - then, attempt to fork the chain before the transaction with Charlie, adding a block which includes a transaction in which she pays herself
  - try to make the chain with the self payment longer than the other one

SIX CONFIRMATION RULE



- for Alice it is now very difficult to make the first fork longer than the other one
- Alice should be luck and mine six blocks in the first fork, before the rest of the network has found any extra block:
  - this depends from the computational power that Alice controls
  - Alice's probability of suceeding is infinitesimal, unless she is able to solve PoW puzzles at a rate approaching all other miners combined
  - this never happen in the Bitcoin network

SIX CONFIRMATION RULE



- no a rigorous security analysis showing that Alice cannot double spend in the original paper
- security communities are is still analysing Bitcoin, and trying to understand possible vulnerabilities.

BITCOIN: THE P2P NETWORK LAYER

Acts like a public "broadcast" network: any-to-all communication:

- forms a random well-connected overlay network
- propagates new transactions and blocks
- provides "blockchain download" for new nodes
- provides "Simplified Payment Verification service" for lightweight mobile clients

THE P2P NETWORK: LEARN NEW NODES

- expoit a set of seed DNS predefined in the Bitcoin network
- the DNS seed nodes return a list of IP addresses of Bitcoin clients that are running (or were recently running), they are known to be more or less permanently available

```
for(String s : BitConstants.DNS)
 {try
    {InetAddress [] adresses = InetAddress.getAllByName(s);
     for(InetAddress addr : adresses)
       {if(!peers.containsKey(addr.getHostAddress()))
         { Peer p = new Peer(addr,BitConstants.PORT);
           peers.put(addr.getHostAddress(),p);
         }
     }
  } catch (UnknownHostException e)
       {  e.printStackTrace(); }}
```

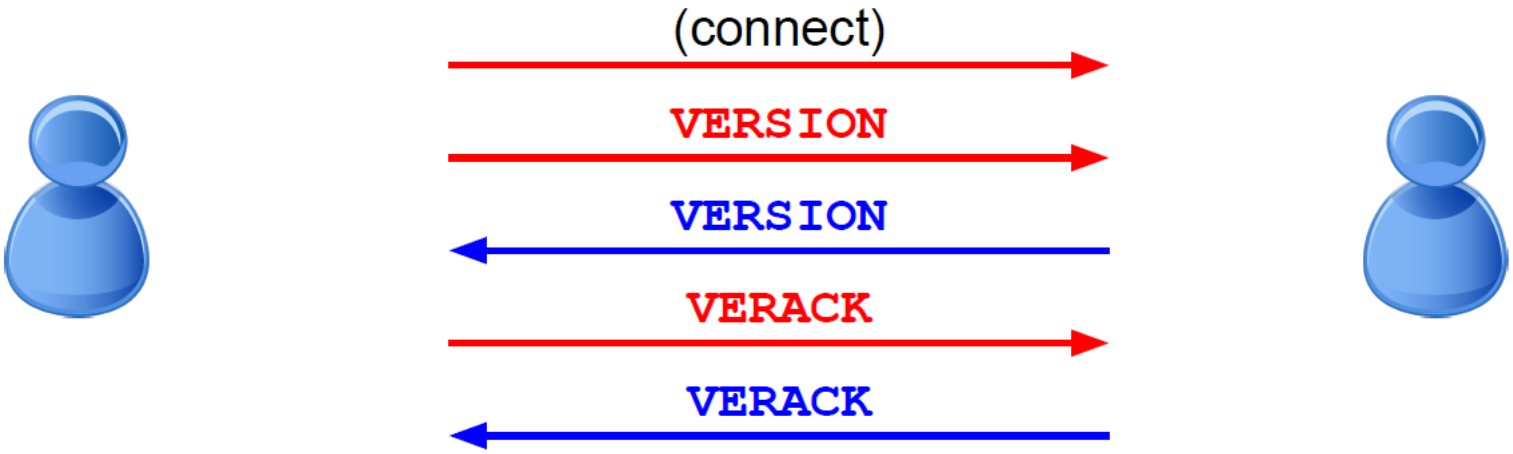
```
178.193.38.151
89.1.102.222
166.111.72.242
206.123.112.180
180.106.132.164
107.182.230.230
122.24.91.59
183.247.219.102
203.98.175.39
37.182.138.43
2.28.245.145
149.56.240.75
91.216.149.28
103.232.209.140
104.236.64.173
73.204.44.97
108.29.189.79
```

THE P2P NETWORK: LEARN NEW NODES

- Hard coded "seed address": long-running stable nodes
  - only used as a last resort, if no other method has produced any addresses at all.
  - trying to not overoad these nodes
- after bootstrapping, a node will remember the addresses of its most recent successful connections
- if it is rebooted it can quickly reestablish connections with its former peer network.
- after the bootstrap:
  - Form 8 outgoing connections
  - Accept up to 125 total connections (incoming and outgoing)
  - Propagate information about nodes to each other

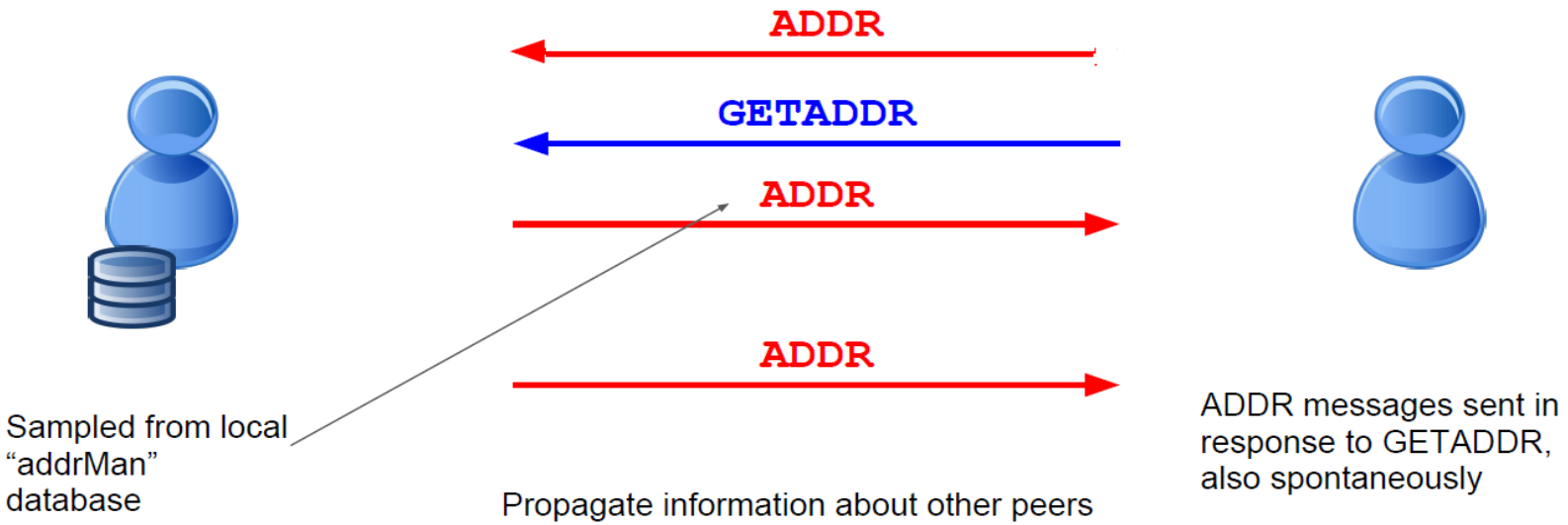
THE P2P NETWORK: HANDSHAKE

- to connect to a known peer, establish a TCP connection, usually to port 8333, or an alternative port if provided.
- upon establishing a connection, the node will start a "handshake"
- VERSION message contains basic identifying information, among which:
  - bestHeight, a node's current blockchain height (number of blocks).
  - compare if the node's blockchain is shorter than the value of BestHeight in the version msg, and, in that case, request and download missing blocks.



THE P2P NETWORK: EXPLORING THE NETWORK

- The new node
  - must advertise its existence on the network
    - sends an ADDR message with its own IP address to its neighbors.
    - the neighbors will, in turn, forward it to their neighbors,
  - may find further peers to connect
    - send GETADDR message to the neighbors, asking them to return a list of IP addresses of other peers.

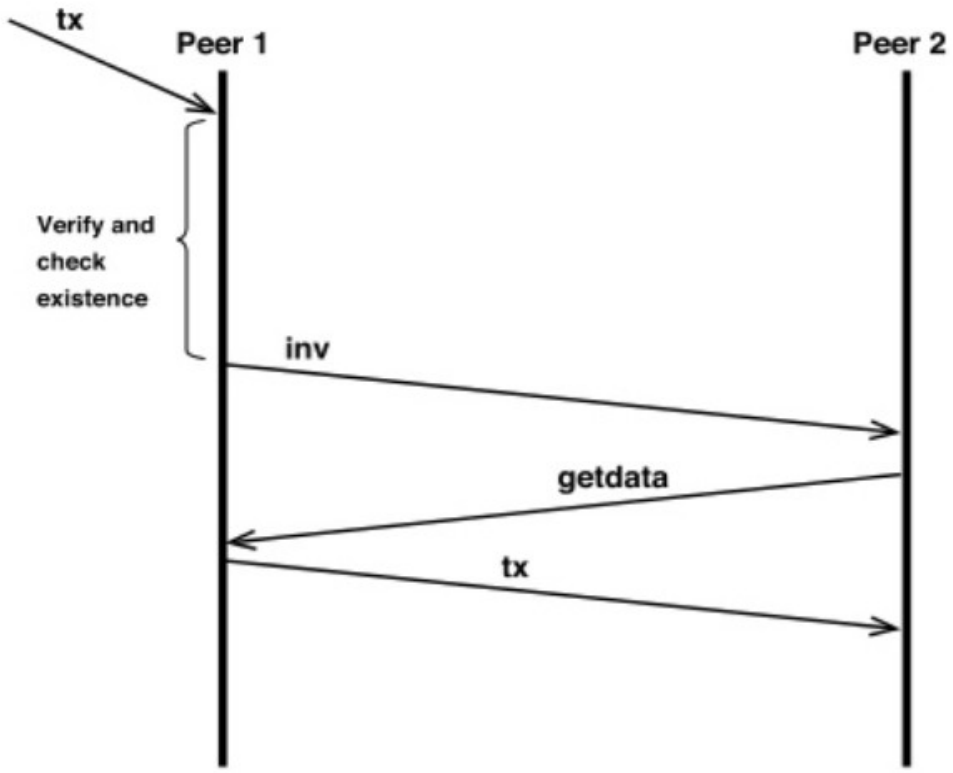


THE P2P NETWORK: EXCHANGING DATA

Messages for the transactions/blocks exchange:

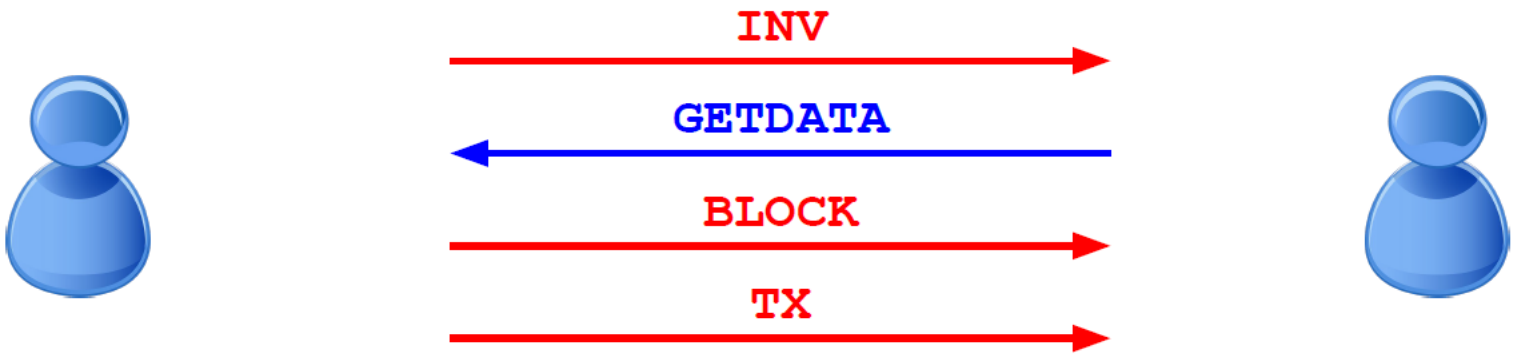
- **INV**
  - an "announcement" message
  - send the hash of blocks or transactions owned by a node, does not contain whole nodes or transactions.
  - hash and type of the advertised block
  - every time a node receives a transaction or a block it announce the transaction to each neighbour, by an inv message,
- **GETDATA**
  - request for a block/transaction
  - sent from peers which do not already have the transaction
- **BLOCK, TRANSACTION**: contain the actual block/transaction

THE P2P NETWORK: EXCHANGING DATA



- a node does not propagate transactions if the are invalid or if it already knows te transaction.
- block message is handled in the same manner

THE P2P NETWORK: EXCHANGING DATA



Only sends GETDATA if it doesn't have it and hasn't already asked

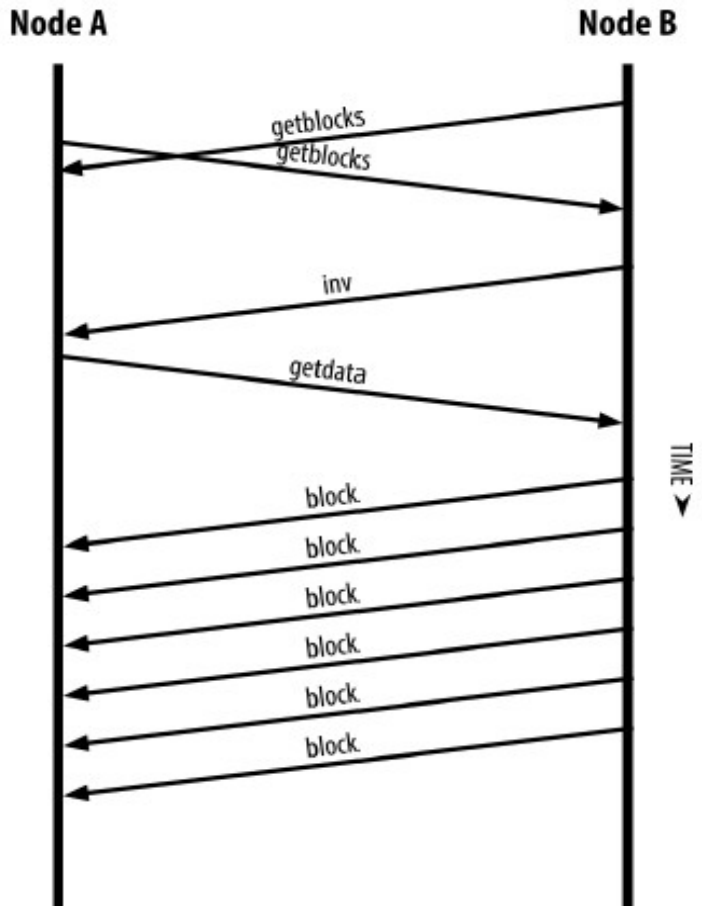
Learn about new transactions

THE P2P NETWORK: INITIAL DOWNLOAD

GETBLOCK

- allows a peer P which has been disconnected or started for the first time to get block it needs to build the updated blockchain.
- ask a neighbour for its local vision of the blockchain
- the neighbour replies with the hash of a set of blocks at various heights on their local chain.
- P can find the first common hash and ask for following blocks through the same message, GETDATA
- iterative process: after having downlaoded all the blocks another GETBLOCK is sent

THE P2P NETWORK: SYNCING THE BLOCKCHIAN



THE P2P NETWORK: SPV MESSAGES

- Bloom filters are used to filter the transactions (and blocks containing them) that an SPV node receives from full nodes
- SPV node
  - creates a Bloom filter that matches only the addresses interesting for it (contained in the node's wallet)
  - sends a message to the full node, containing the Bloom filter
- The full node test each transaction's outputs against the bloom filter. and sends only transactions that match the filter to the SPV
- Bloom filters:
  - SPV Address obfuscation
  - Bandwith saving

THE P2P NETWORK: SPV MESSAGES

- in response to a GetData message from the SPV, the full node
  - sends a merkle block message containing a merkle path for each matching transaction.
  - send messages containing the transactions matched by the filter.

- The SPV:
  - can interactively add patterns to the filter by sending a filteradd message.
  - can clear the bloom filter, sending a filterclear message.
  - because it is not possible to remove a pattern from a bloom filter, a node has to clear and resend a new bloom filter if a pattern is no longer desired.