Blockchain and Distributed Ledgers

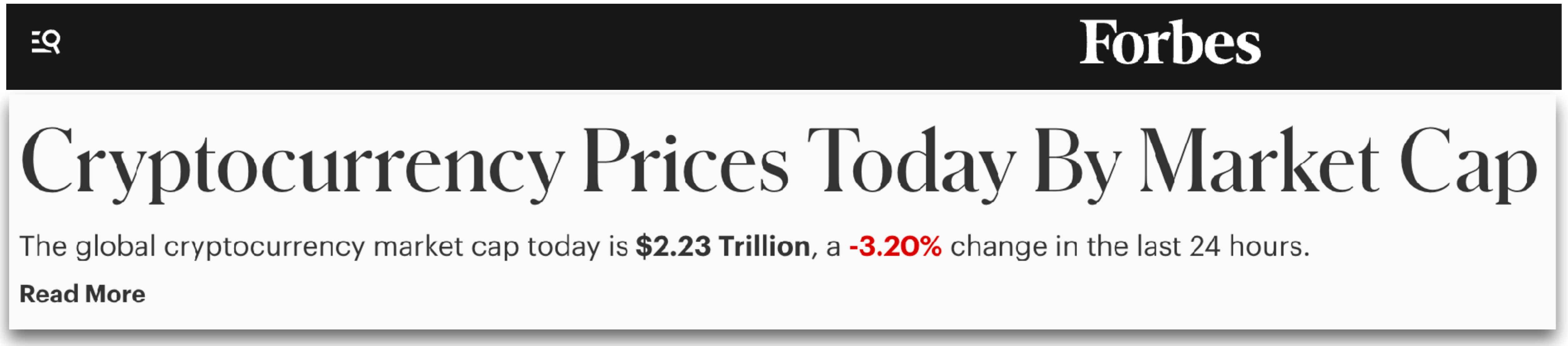
Tom Van Cutsem
KU Leuven, DistriNet

ACM Europe DARE 2026 Summer School - Milan

tvcutsem.github.io be.linkedin.com/in/tomvc github.com/tvcutsem @tvcutsem.bsky.social

Blockchain in 2026?

Digital assets are now an established asset class


Forbes

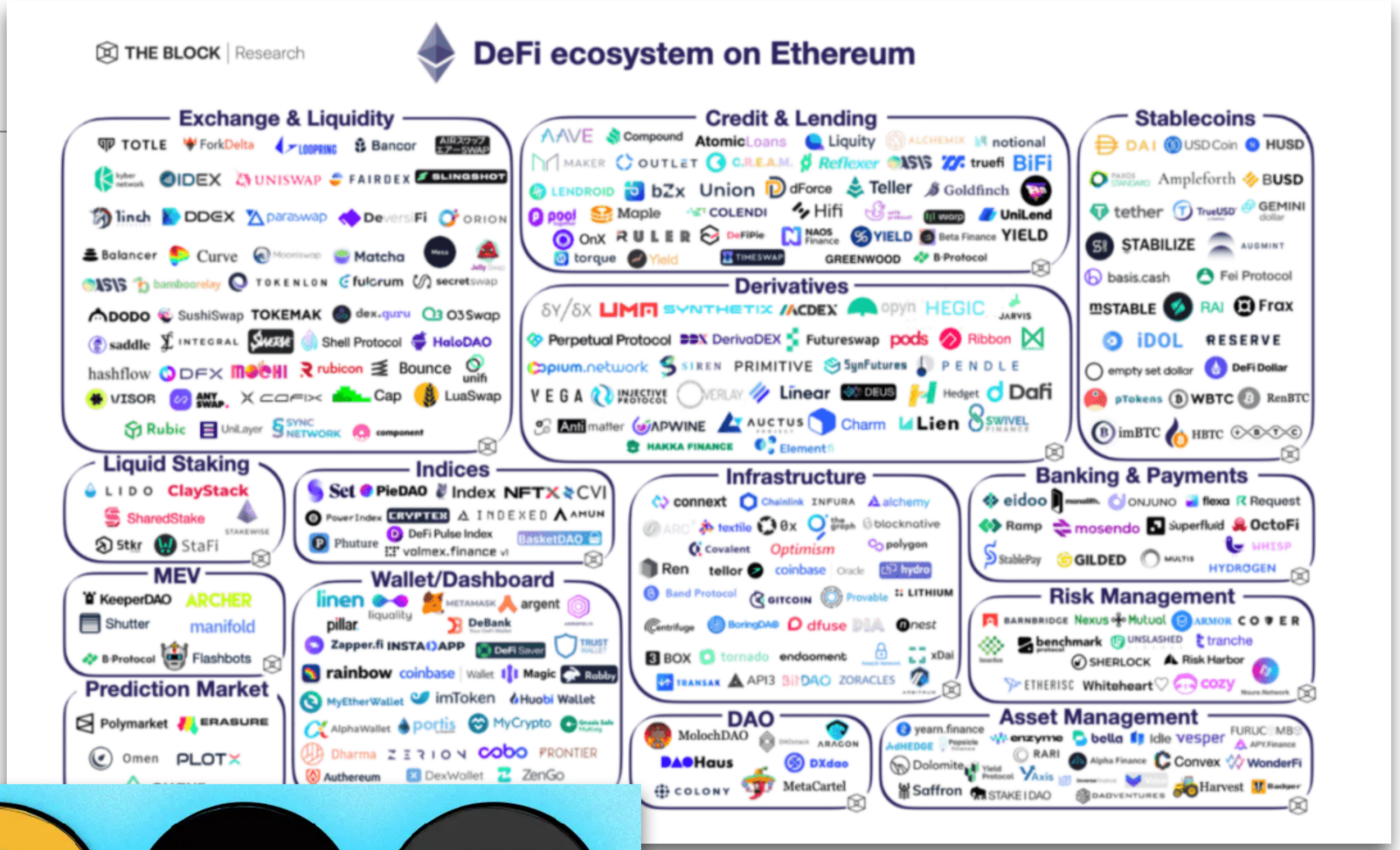
Cryptocurrency Prices Today By Market Cap

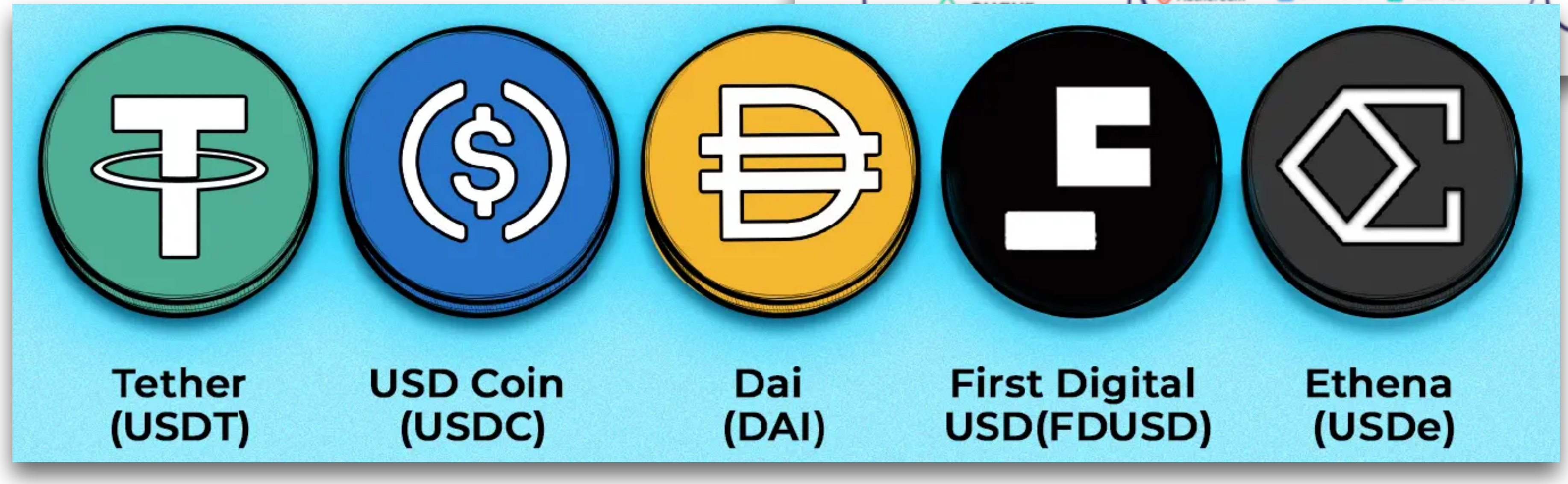
The global cryptocurrency market cap today is **$2.23 Trillion**, a **-3.20%** change in the last 24 hours.

Read More

coinbase GRAYSCALE

Ethereum & **De**centralized **Fi**nance





Stablecoins for global payments

Real-world Assets (RWA)

BlackRock

Regulatory clarity

EU MiCA, US GENIUS Act

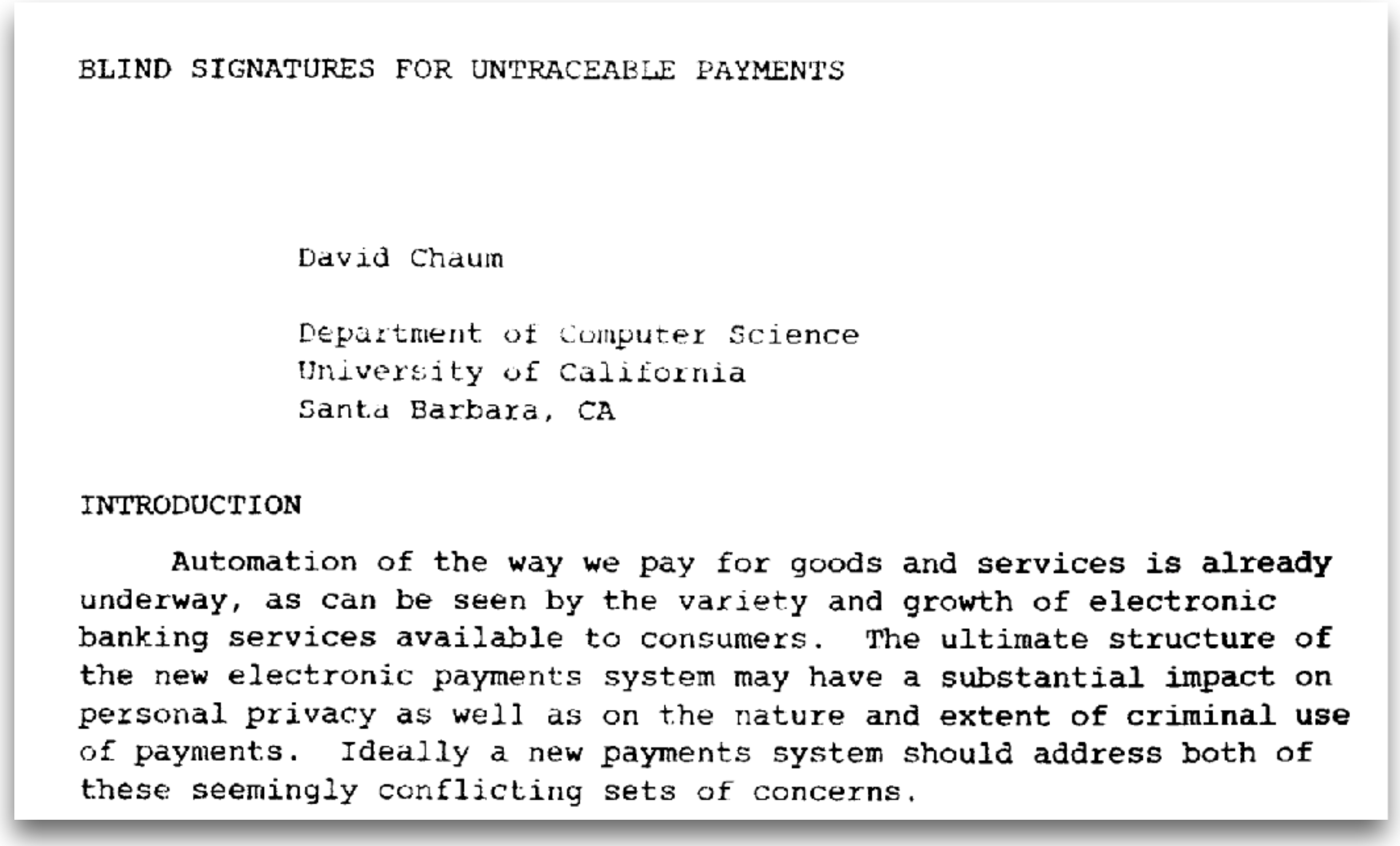
Course topics

- The **origins** of Blockchain
- What are the **cryptographic building blocks** of a blockchain?
- How does a blockchain process transactions? **Life of a blockchain transaction**.
- **Consensus** in blockchain networks: Proof-of-Work, Proof-of-Stake, BFT Consensus
- **Permissioned** versus **Permissionless** blockchain networks
- Bonus: Blockchains as trusted computers: **smart contracts** and **Ethereum**
- **Conclusion** and latest trends

Blockchain: origins

Electronic **cash**

- Since the dawn of the Internet, cryptographers have tried to create digital currencies that are more similar to physical cash or coins
- Money as a “bearer instrument” token: whoever holds the token can spend it
- Payments are anonymous and untraceable
- Example: e-cash (Digicash)



BLIND SIGNATURES FOR UNTRACEABLE PAYMENTS

David Chaum

Department of Computer Science
University of California
Santa Barbara, CA

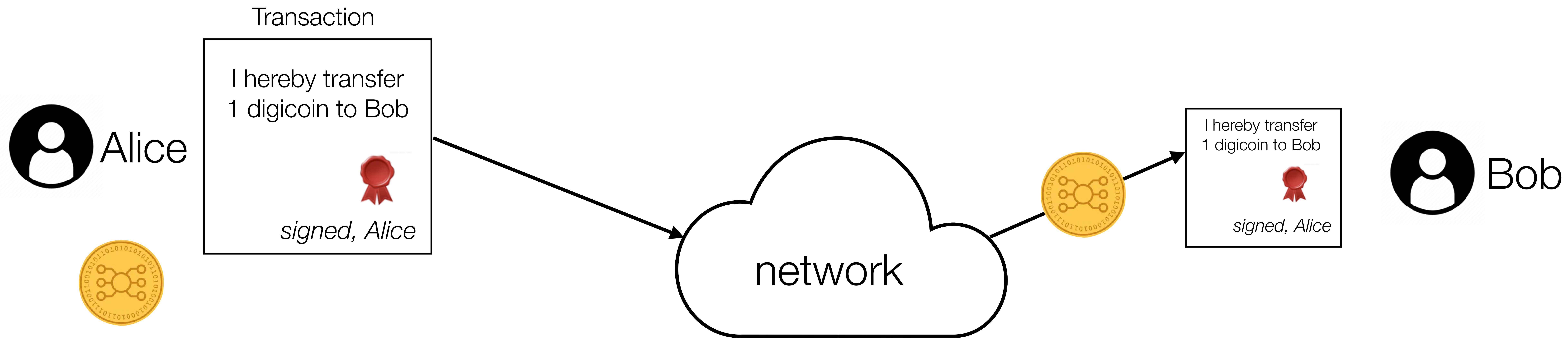
INTRODUCTION

Automation of the way we pay for goods and services is already underway, as can be seen by the variety and growth of electronic banking services available to consumers. The ultimate structure of the new electronic payments system may have a substantial impact on personal privacy as well as on the nature and extent of criminal use of payments. Ideally a new payments system should address both of these seemingly conflicting sets of concerns.

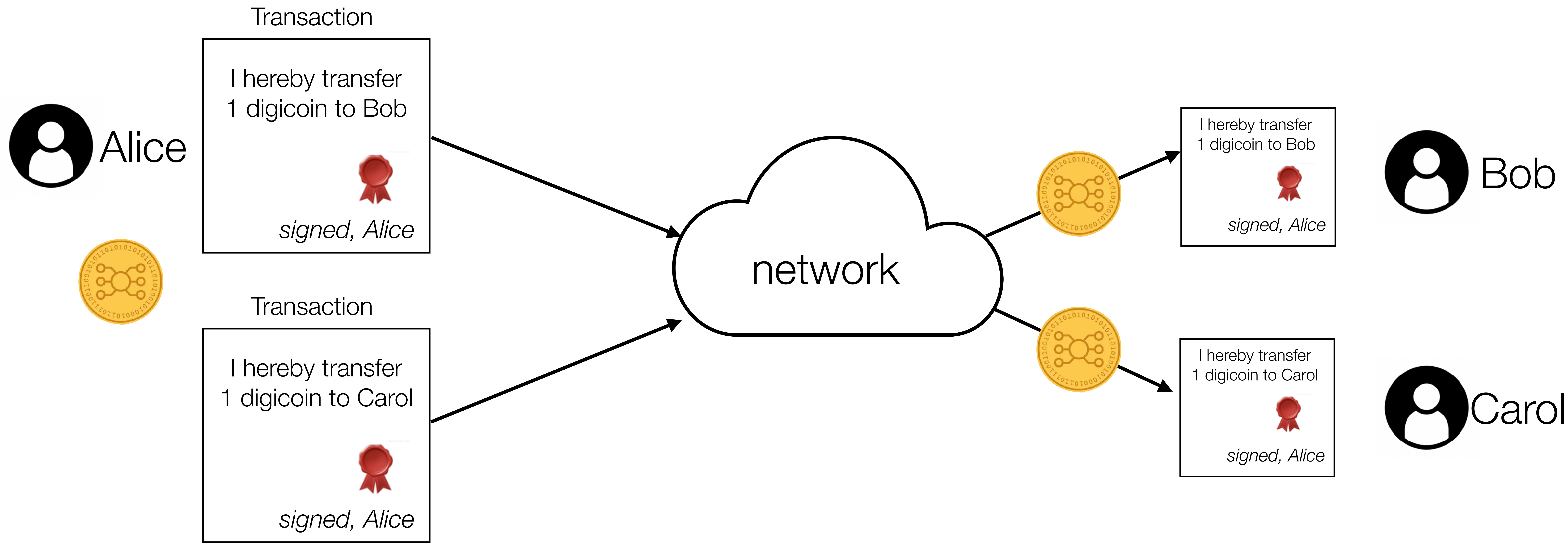
David Chaum
Electronic cash (1982)

KU LEUVEN DistriNet

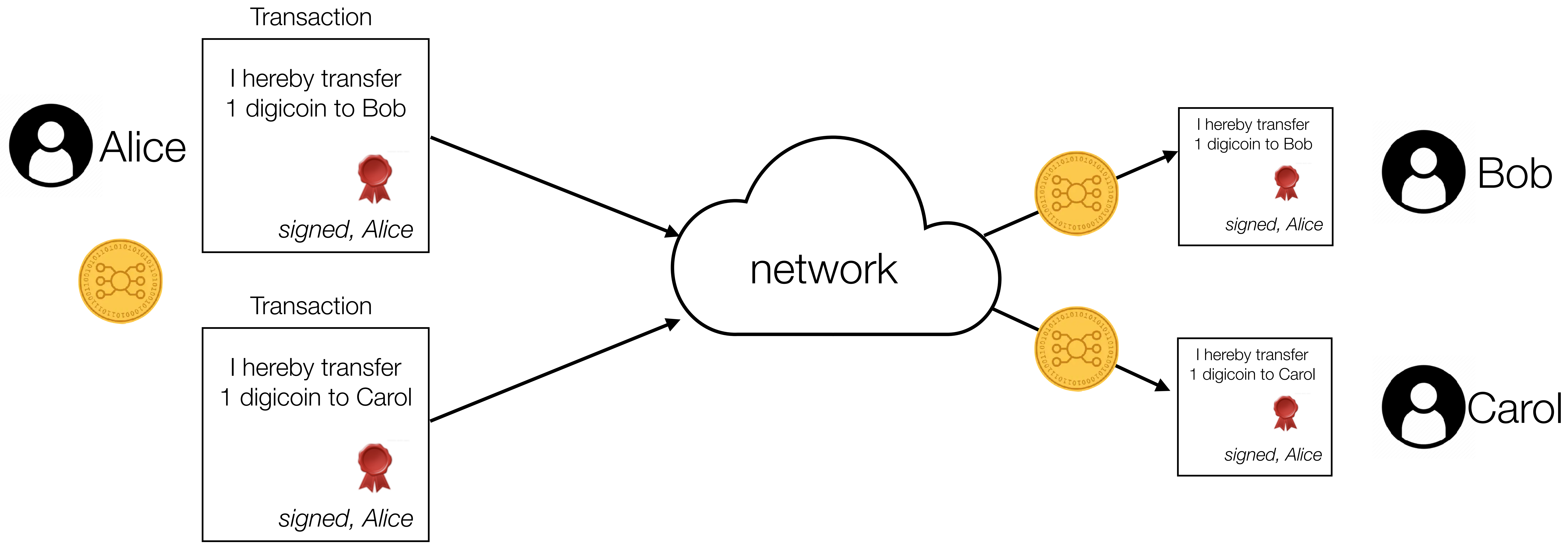
The problem with electronic cash: the Double Spending Problem



The problem with electronic cash: the Double Spending Problem



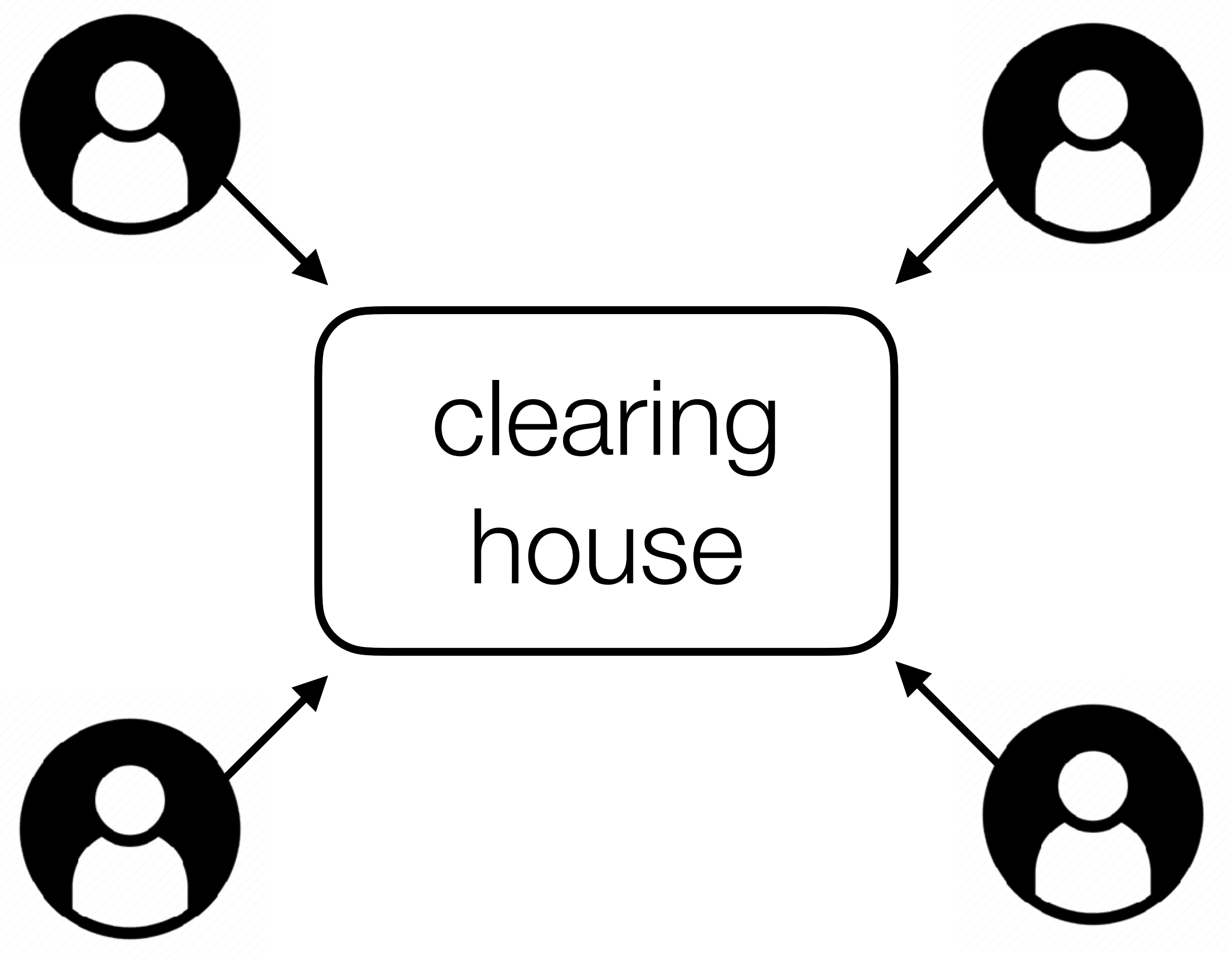
The problem with electronic cash: the Double Spending Problem



How can Bob and Carol be sure they are now the sole owner of Alice’s coin?

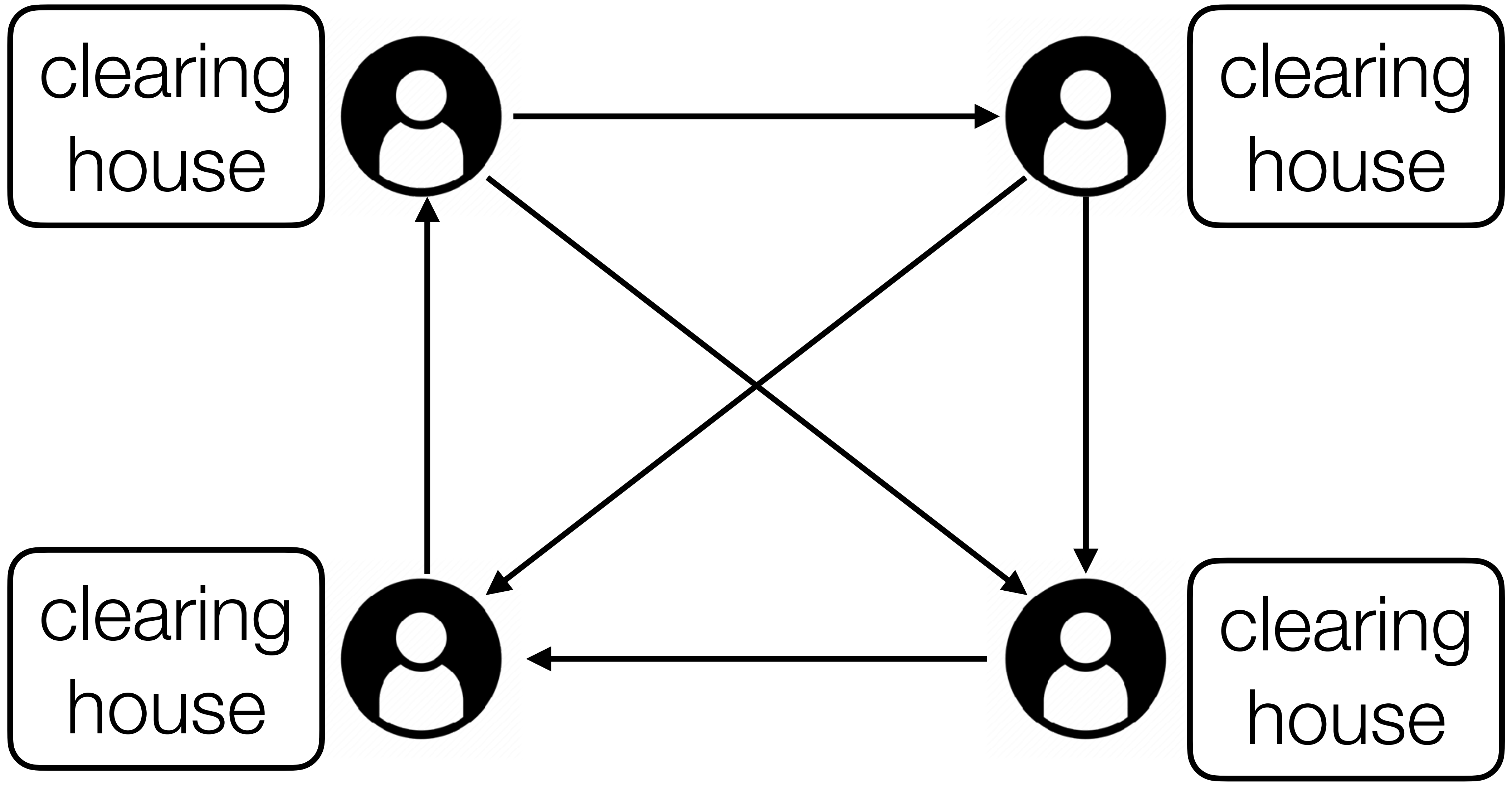
Straightforward solution: use a central clearing house

- The clearing house does the accounting of what tokens have already been spent. This **avoids "double spending"** the same token.
- The payments themselves can still be **anonymous**! We just need to record spent tokens. Privacy risks partially mitigated using blind signatures.
- Problem: everyone depends on the clearing house. **Risks**:
  - **Technical** risks: availability (what if the clearing house is unavailable?) and security (what if the clearing house gets attacked? This may include insider threats!)
  - **Economic** and **political** risks: what if the company running the clearing house goes bankrupt or is threatened in court? (E.g. Digicash actually went bankrupt in 1998)

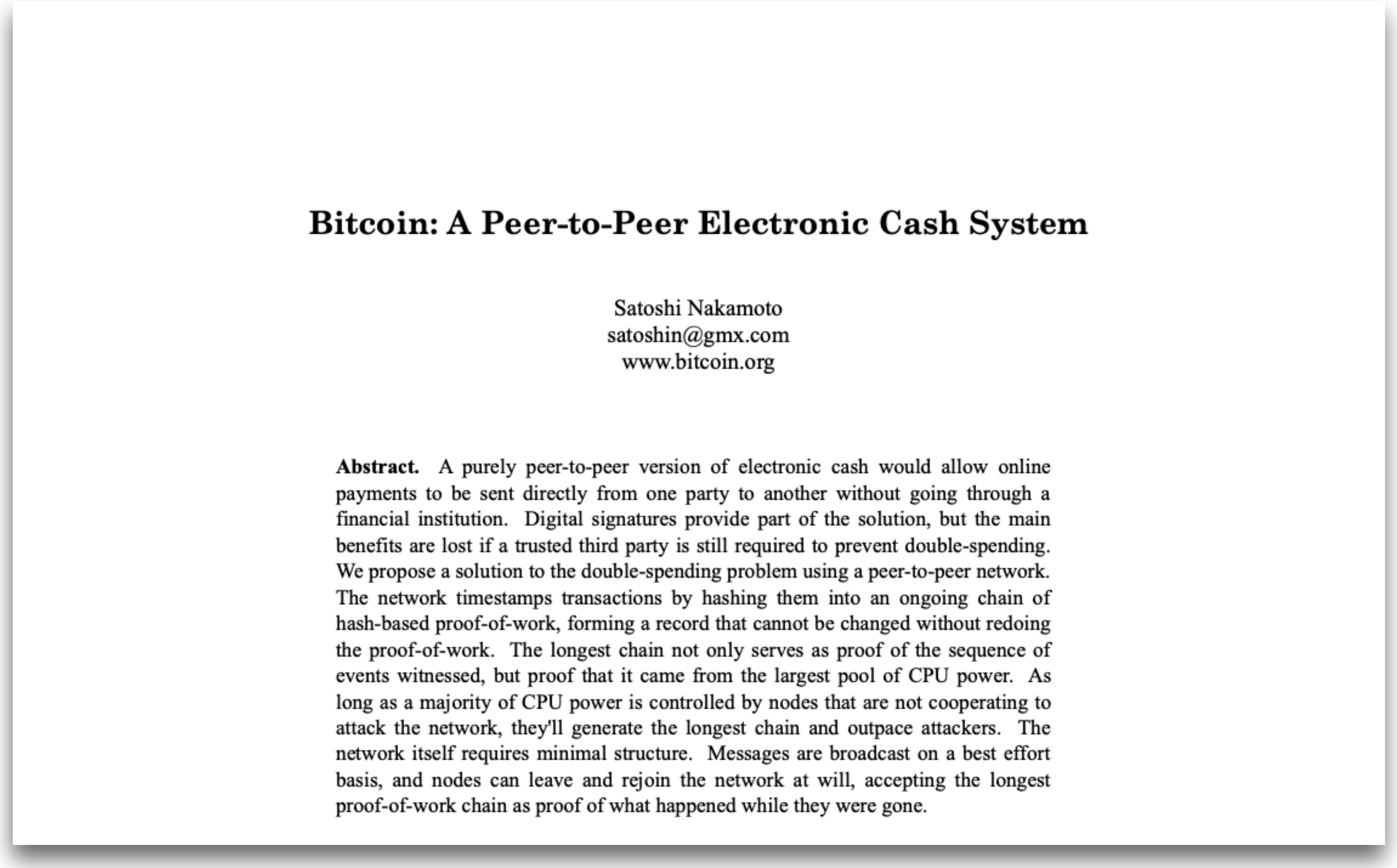


Blockchain networks

- Bitcoin's breakthrough idea: rather than having a single party record who owns what, let *everyone* collectively do the accounting of who owns what
- Store account balances in an append-only **replicated database** called a **blockchain**



Fully decentralised payment network



Bitcoin: A Peer-to-Peer Electronic Cash System

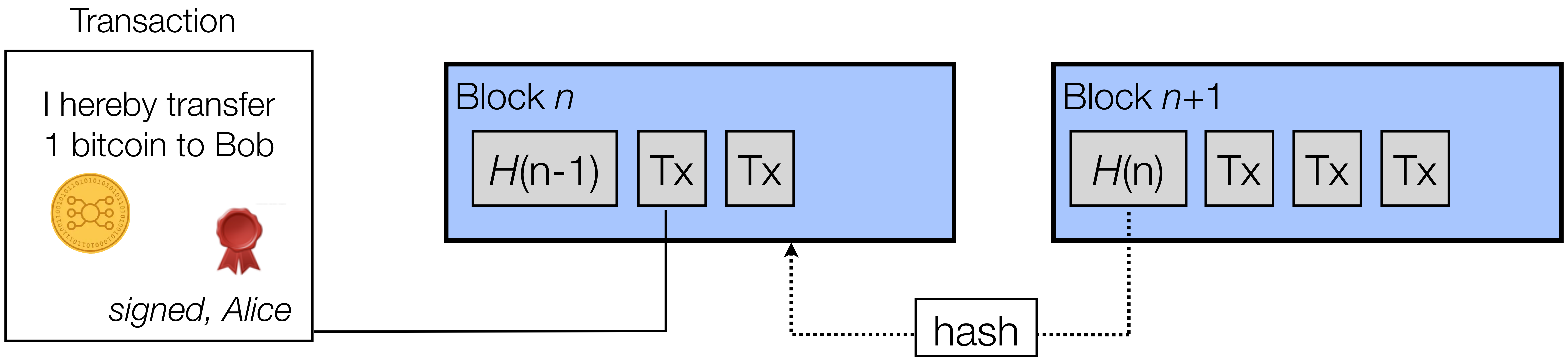
Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

The "Bitcoin whitepaper" by "Satoshi Nakamoto", 2009

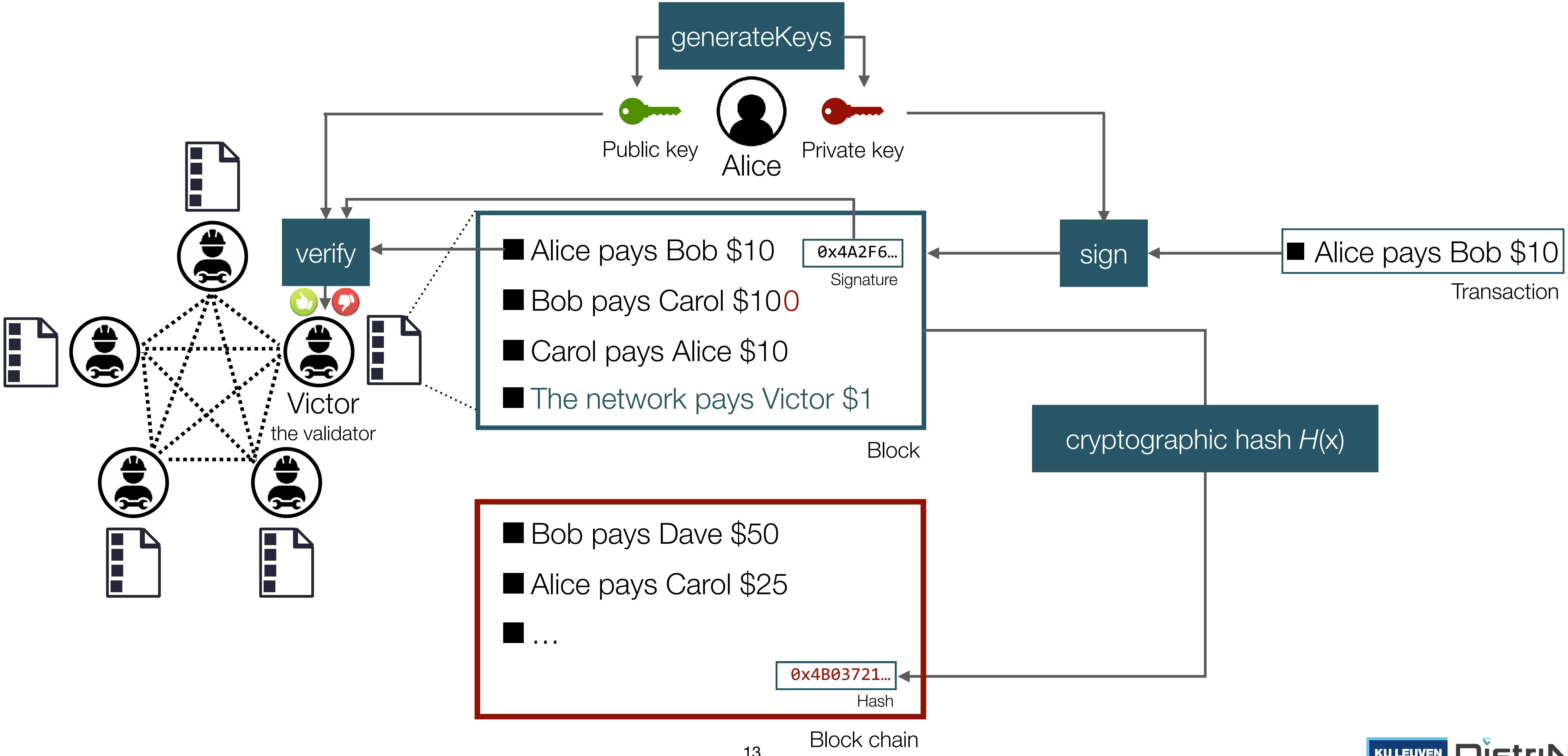
Bitcoin's blockchain

- Replace central clearing house by a **public, replicated, append-only, tamper-resistant ledger**
- Validator nodes group transactions in "blocks", "chained" together into **a linear sequence** using cryptographic hashes, secured using "Proof of Work"



What are the cryptographic building blocks of a blockchain?

How cryptography is used to securely record transactions on a blockchain

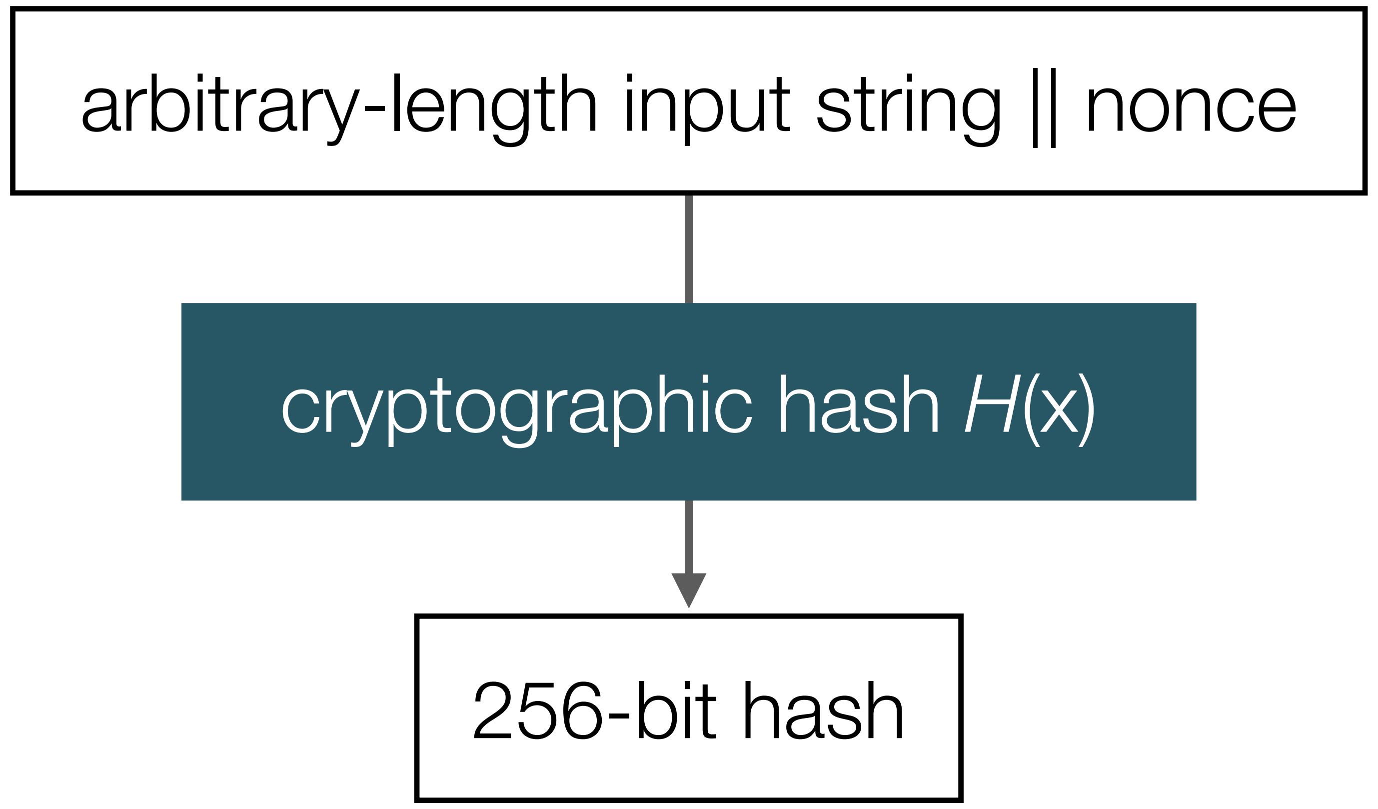


Common cryptographic algorithms used in blockchain systems

Cryptographic hashes

SHA-256 or KECCAK-256

Secure hash algorithm



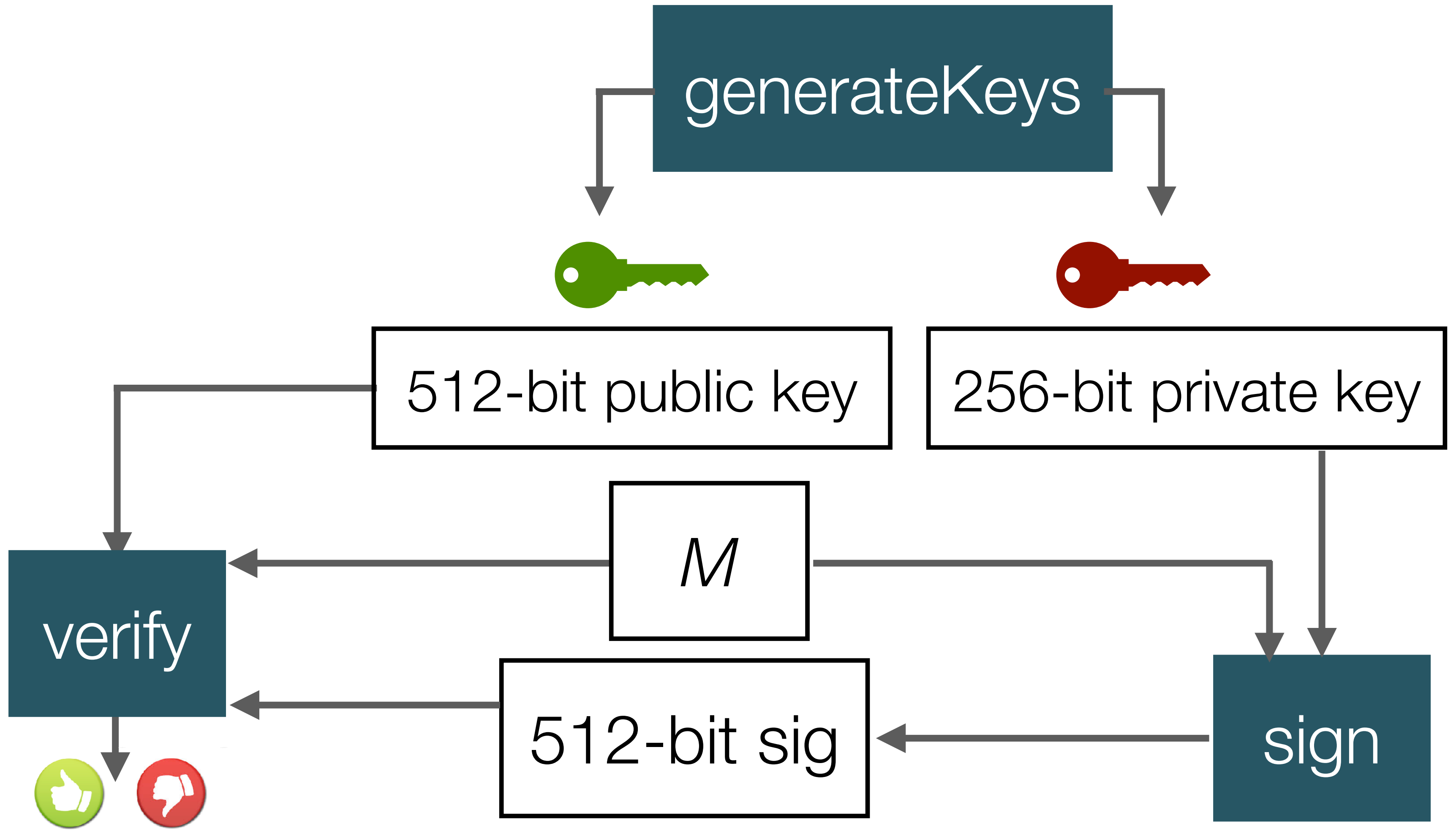
Desirable properties:

- *H* is collision-resistant
- *H* hides its input x
- *H* is “puzzle-friendly”

Digital signatures

ECDSA (secp256k1 curve)

Elliptic curve digital signature algorithm

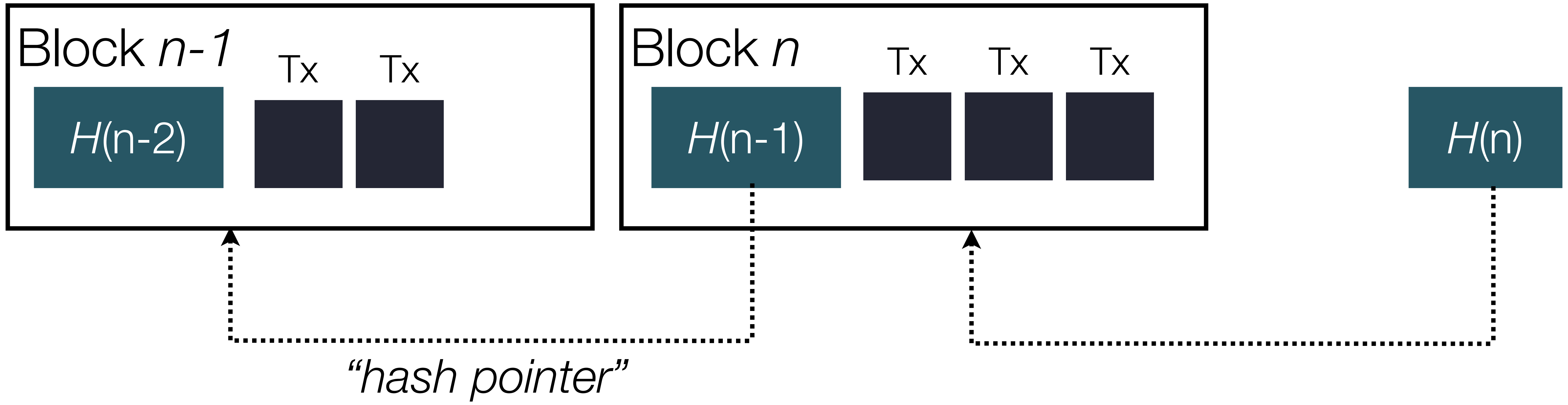


Desirable properties:

- Valid signatures must verify
- Signatures are unforgeable
- Signature is unique to *M*

Common cryptographic algorithms used in blockchain systems

Hash pointers



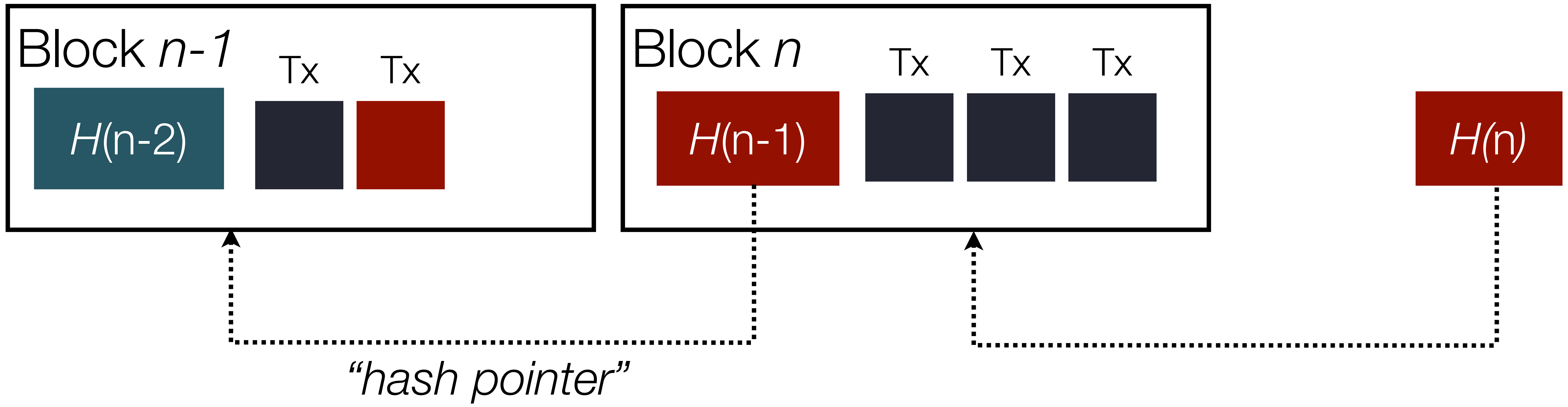
The hash is used both as:

- a unique identifier (to identify and lookup the data)
- a digest (to verify that the data has not been tampered with)

Any non-cyclical data structure can be built from hash pointers

Common cryptographic algorithms used in blockchain systems

Why use hash pointers?



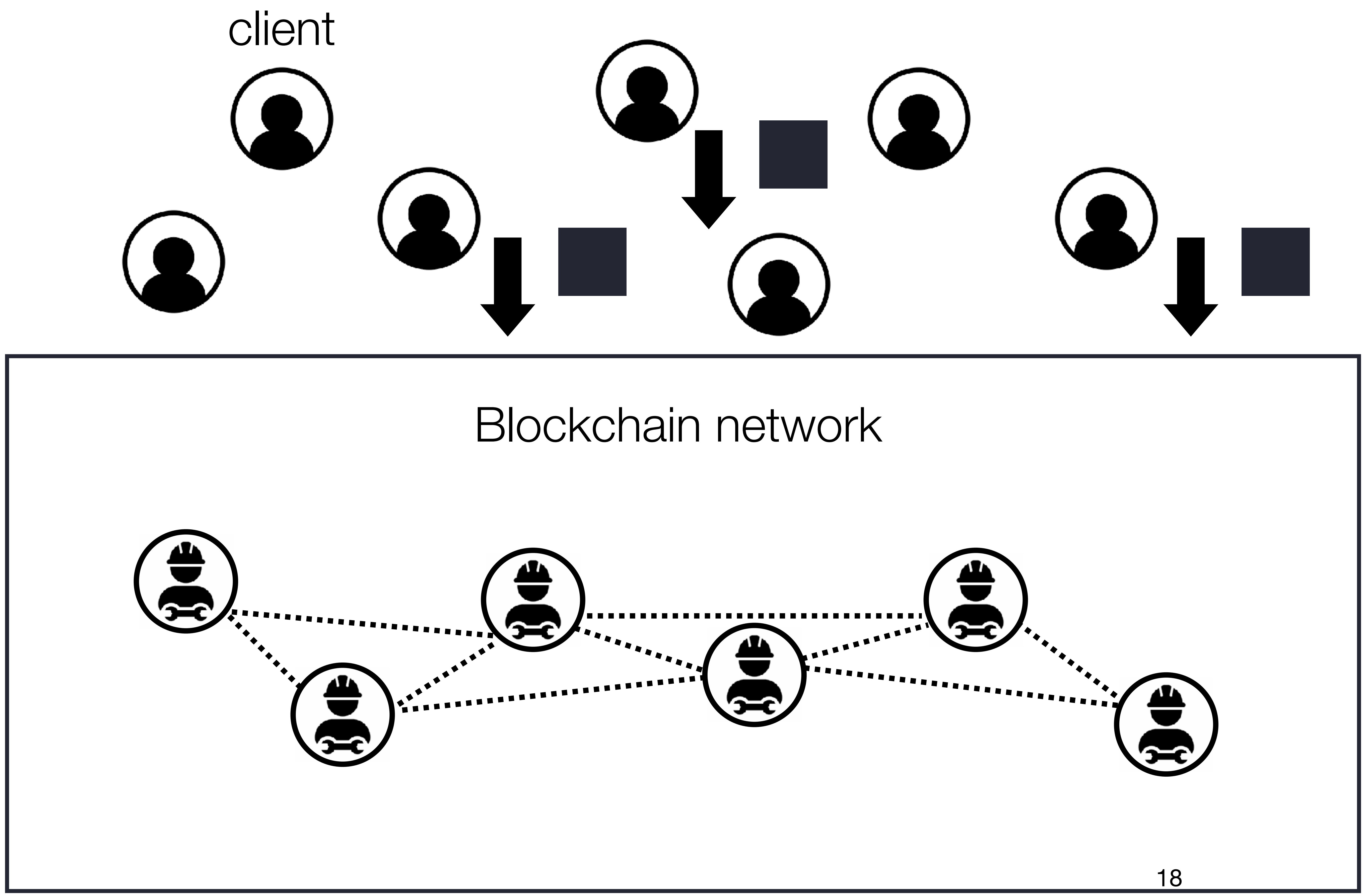
- We want the transaction log history to be immutable (i.e. only *append* new transaction, not *edit* past transactions).
- By using hash pointers, we ensure that modifying *any* data in *any* past block would invalidate the hash pointers of *all* the following blocks.
- This makes it immediately clear to anyone with a historical copy of the blockchain that data has been tampered with.
- This makes the transaction log "tamper-evident".

How does a blockchain network process transactions?

A.k.a. the “life of a blockchain transaction”

Step 1: clients submit signed transactions

- Clients **concurrently** submit signed transactions to one or more validators.



Example transactions…

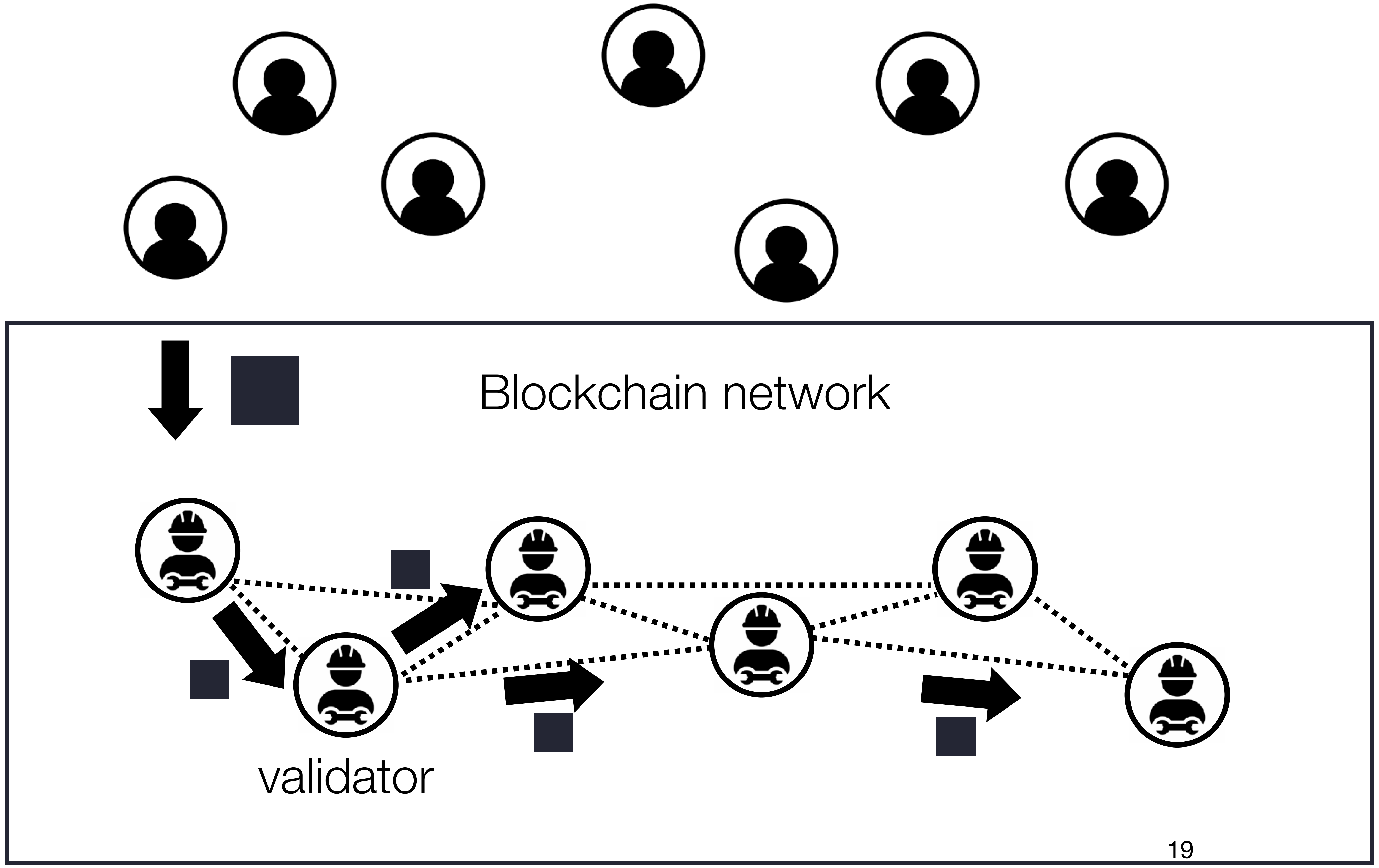
= “send x bitcoin to address *a*”

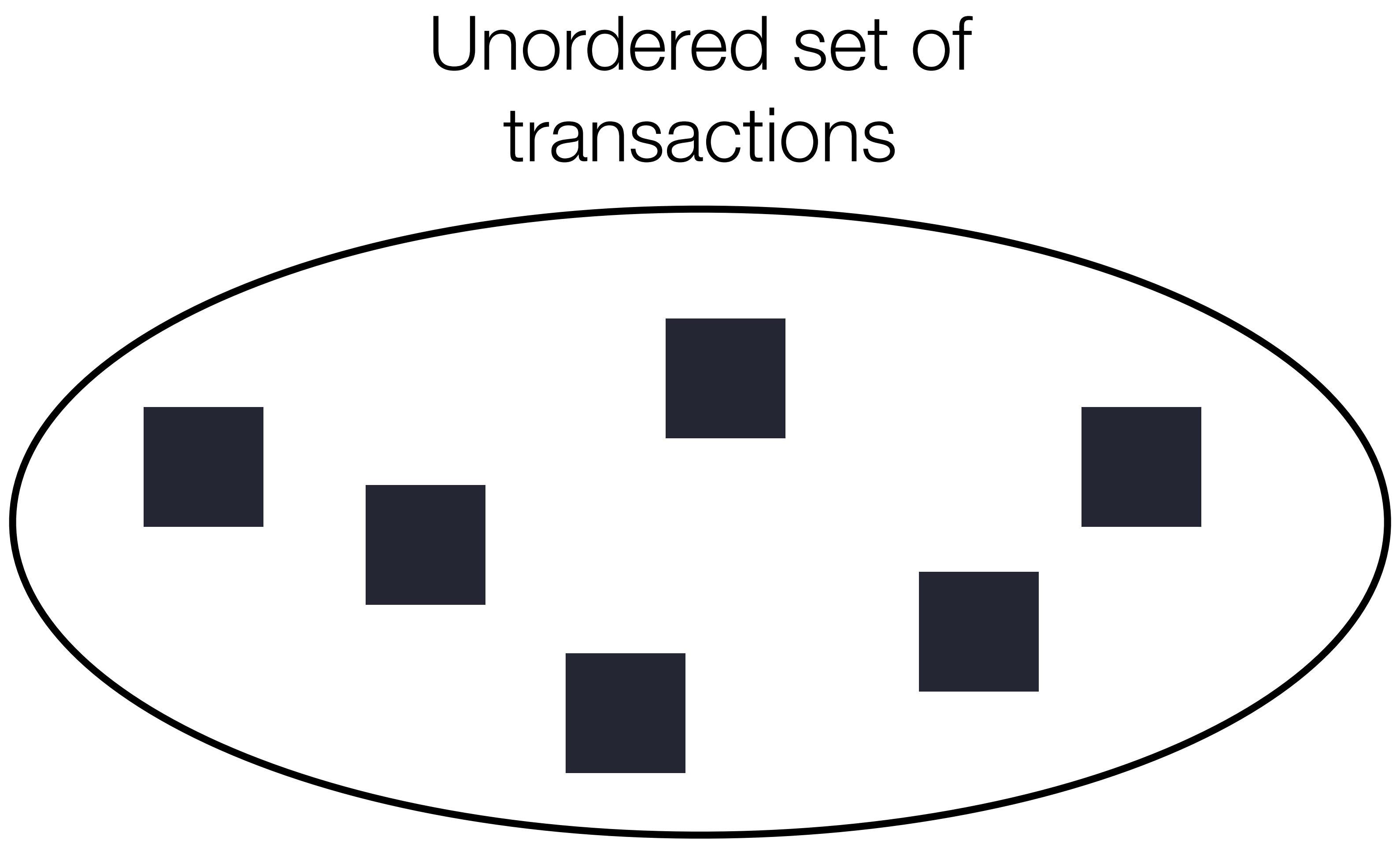
= “call function *f* on contract *a* with input *x*”

= “please store these bytes”

Step 2: validators validate and gossip transactions

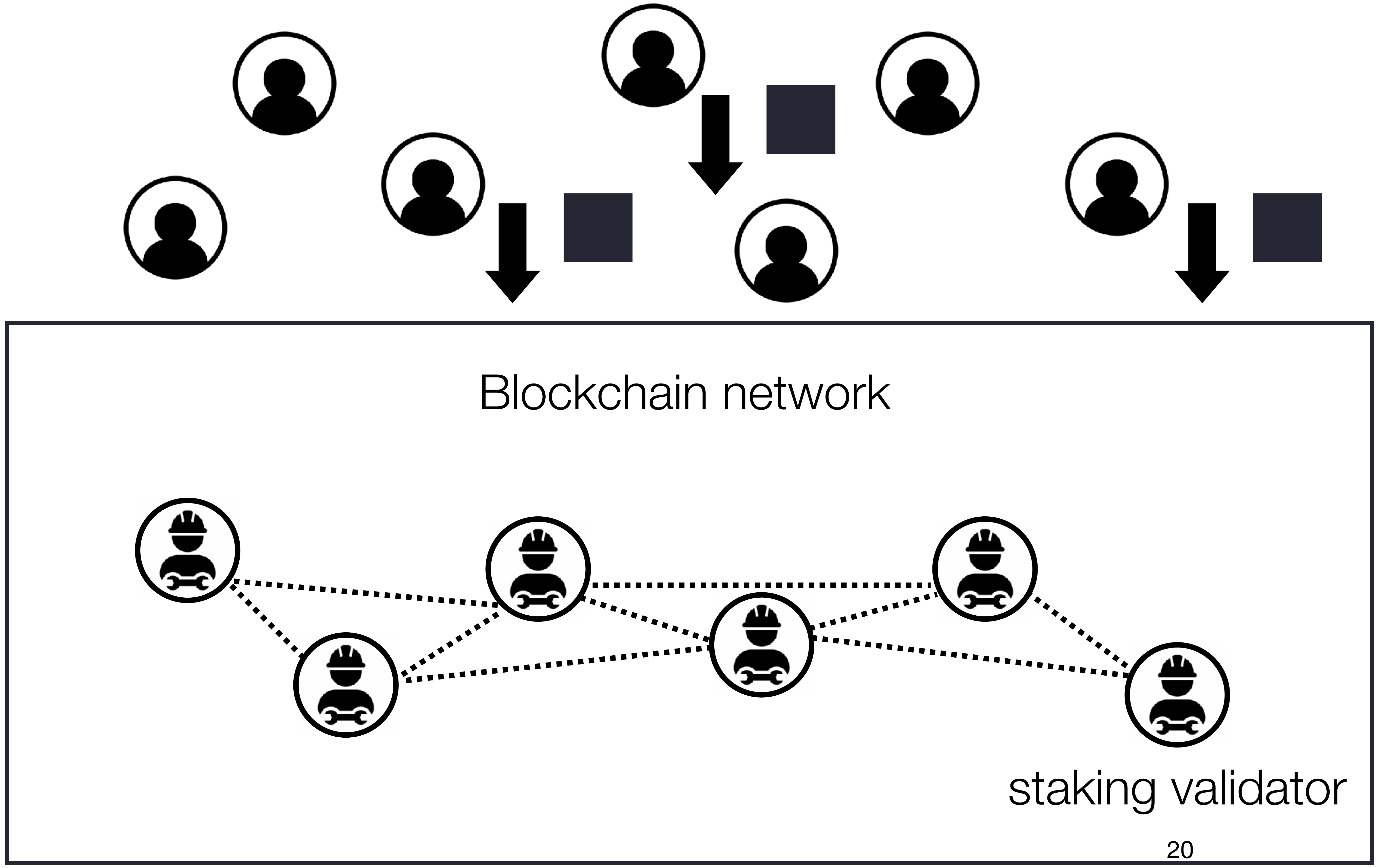
- A **validator** is a network node that maintains an **unordered set** (“mempool”) of incoming transactions. It collects, validates and broadcasts transactions to other peers (using a **gossip** broadcast protocol)

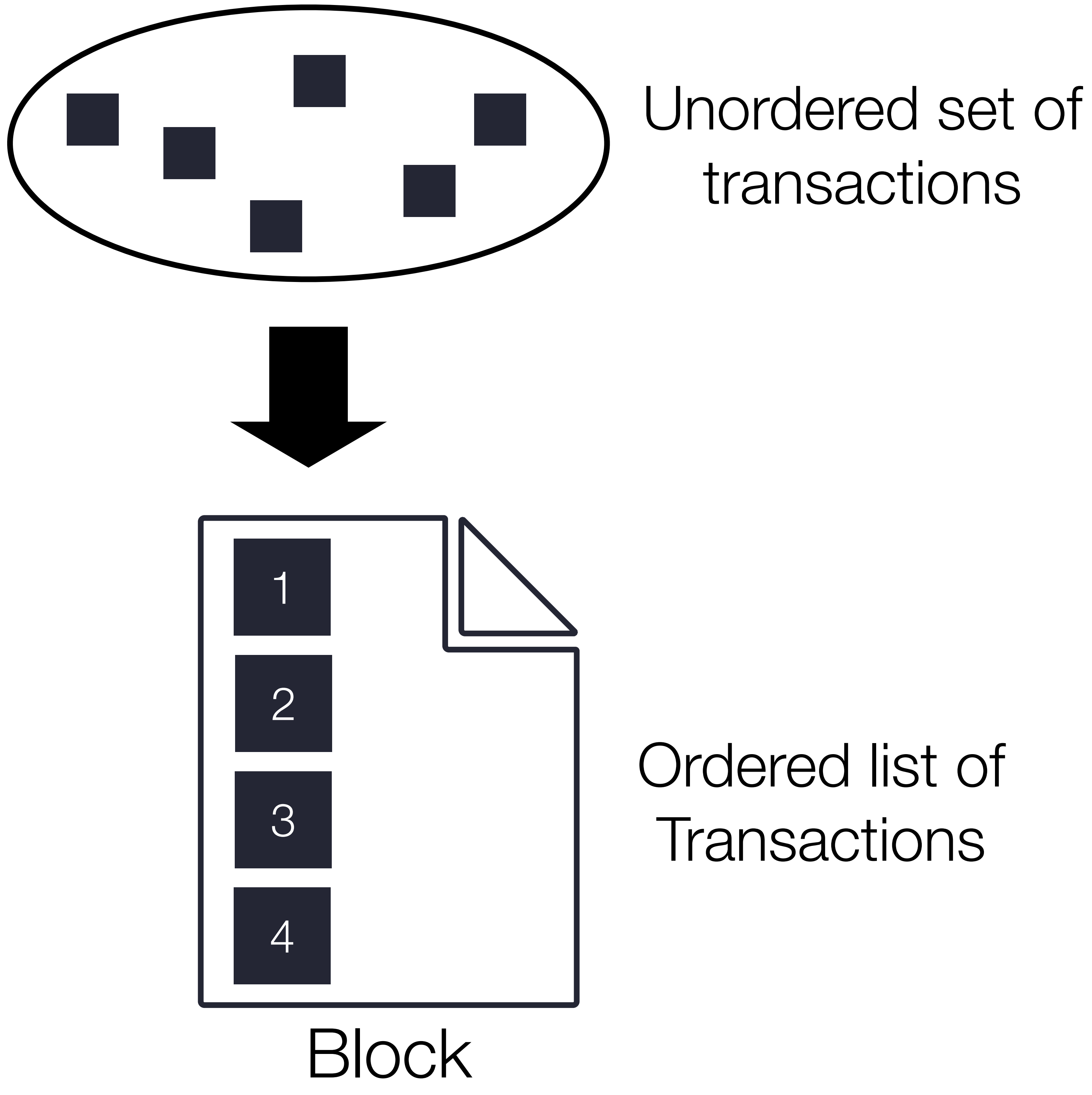




Step 3: a validator produces a block of transactions

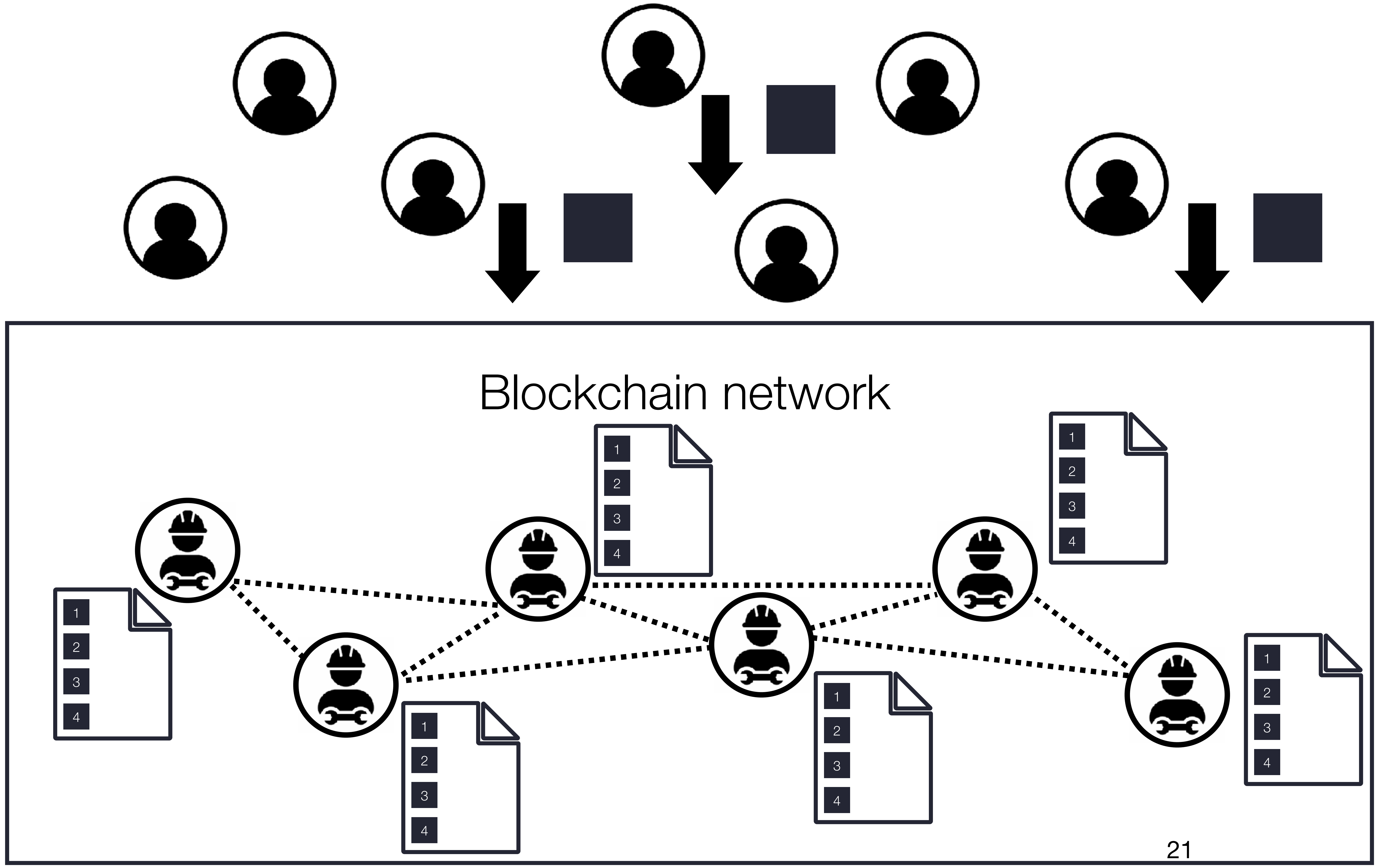
- At regular intervals, a subset of validators pick a subset of transactions from the pool and *sequence* them, thus producing an *ordered* list of transactions. These validators are sometimes called “**miners**” or “**staking validators**”. The transaction list is called a “block”

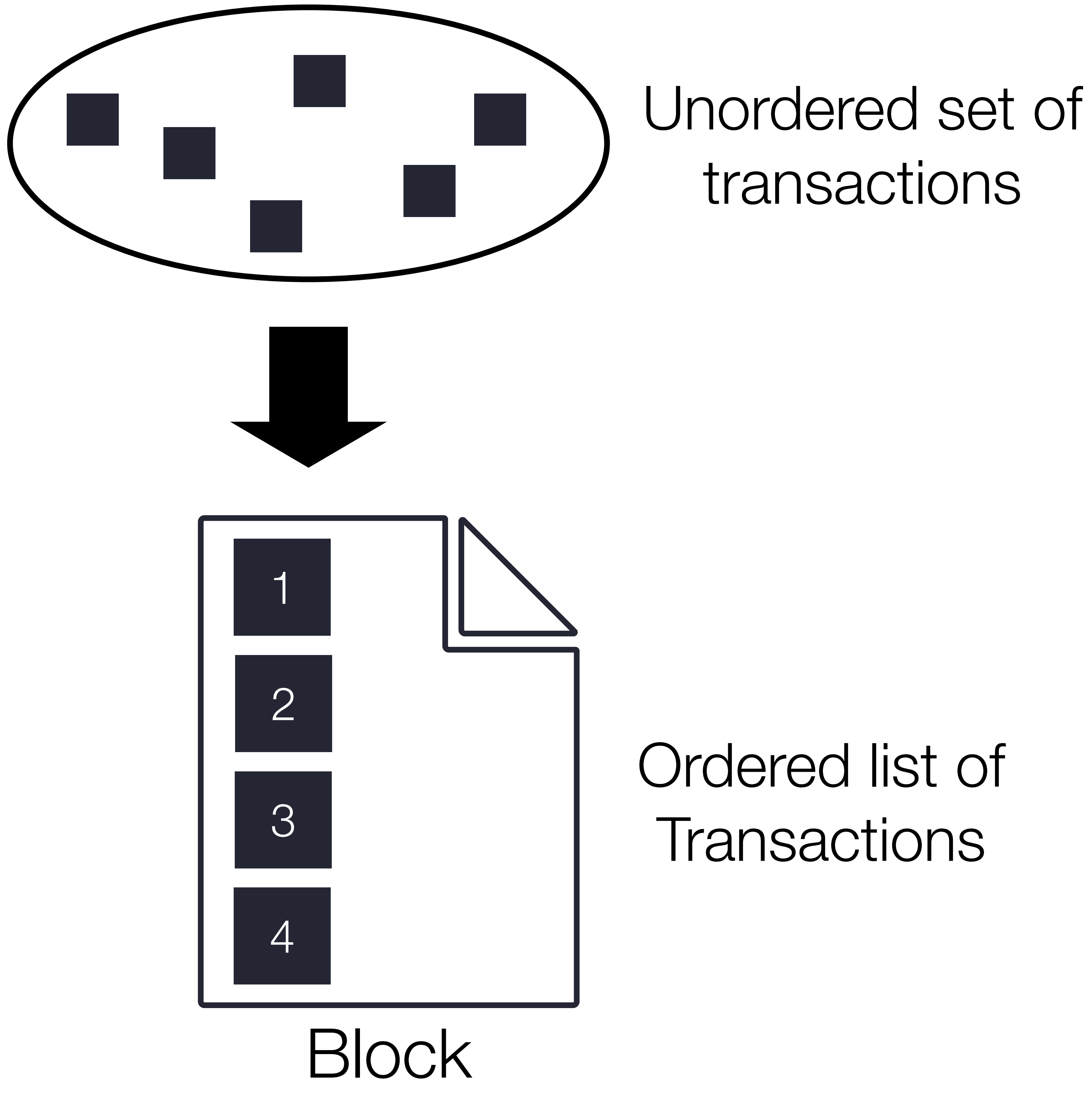




Step 4: validators gossip block and append to the blockchain

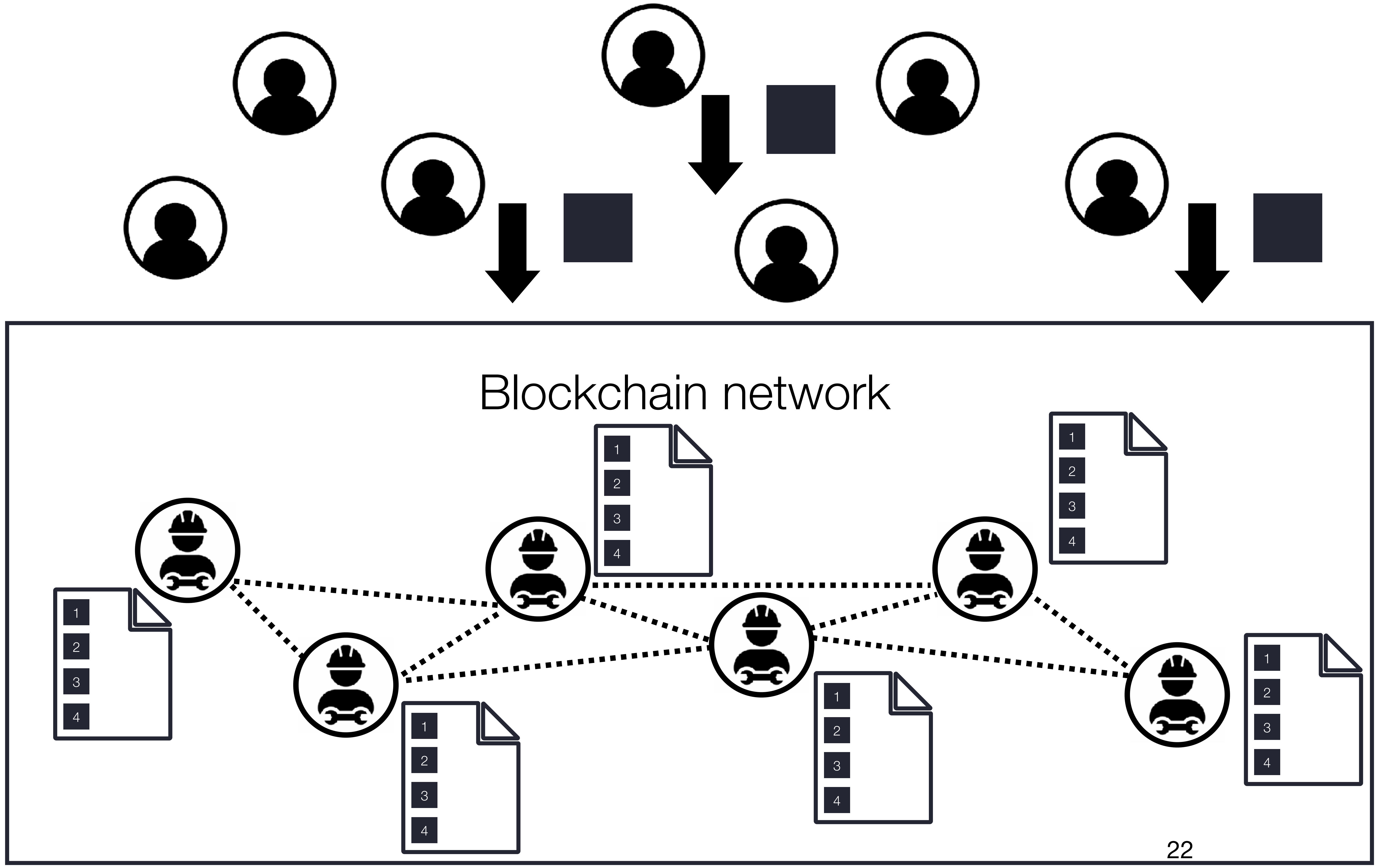
- The **block is broadcast** to all validators (again using gossip). Each validator **checks again** if all transactions in the block are valid. If yes, they **append** the block to their local transaction log (aka the blockchain).

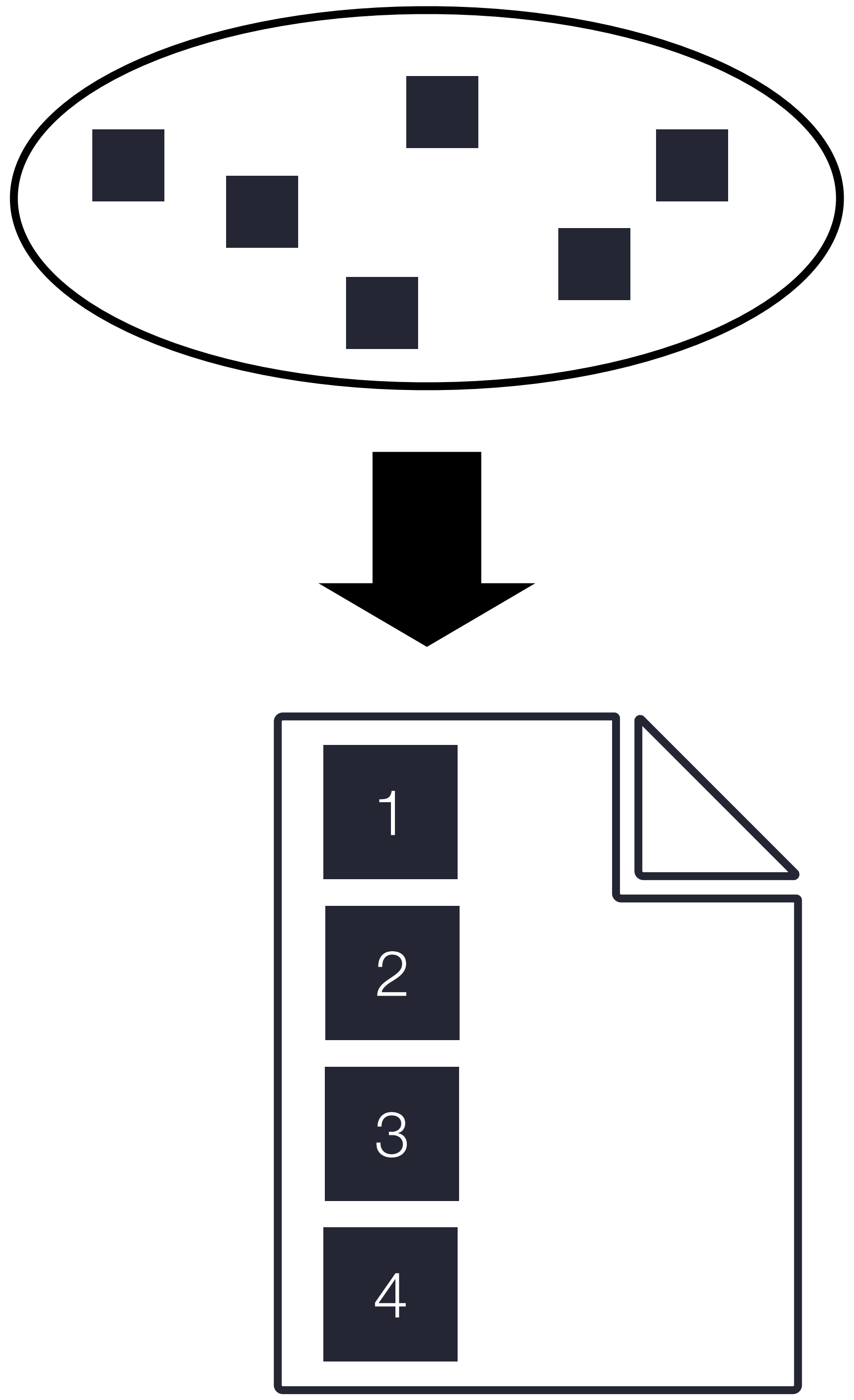




Consensus

- All validators must reach **consensus** on the exact same transaction history!
- Need to make sure that blocks get appended everywhere *in the same order*





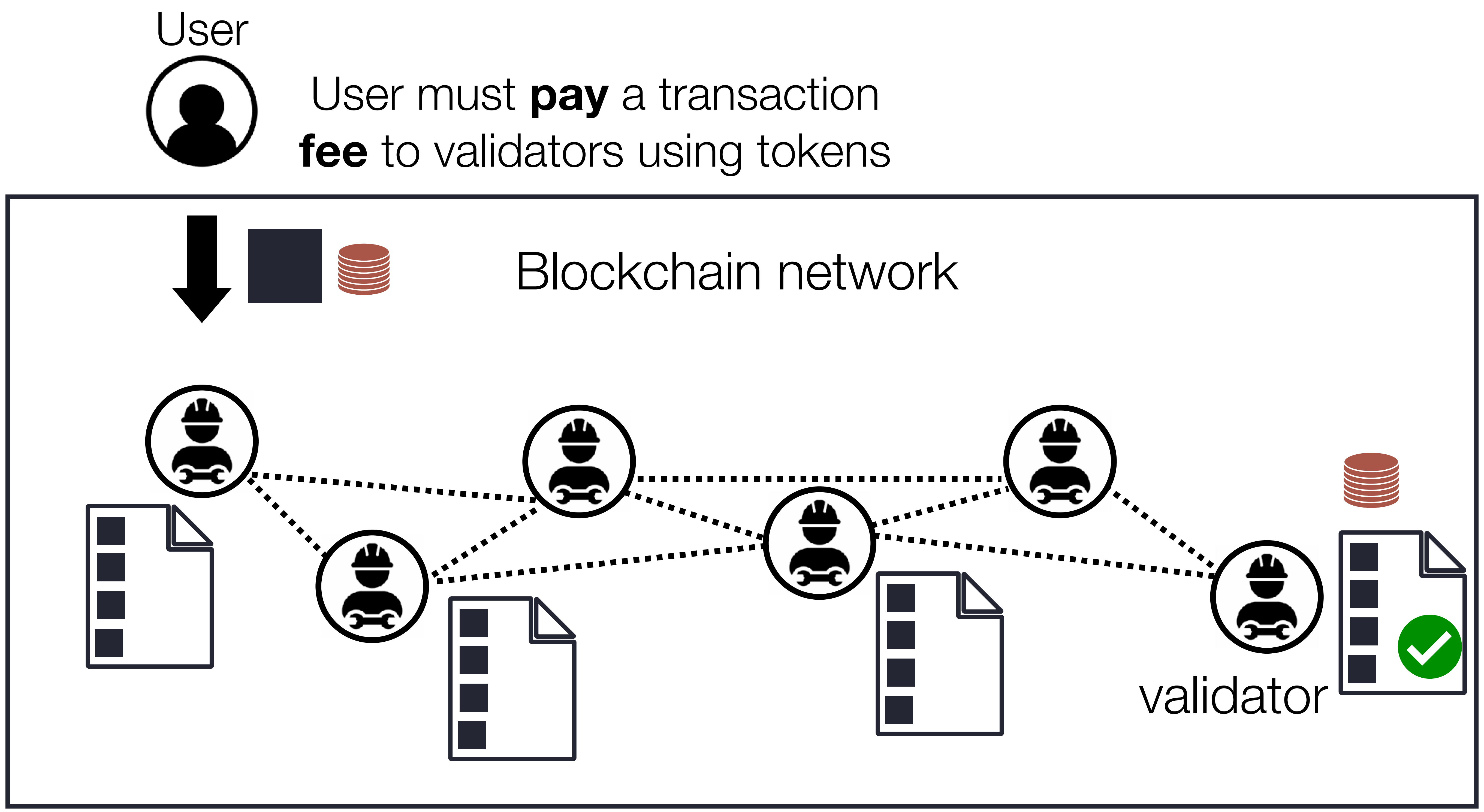
Unordered set of transactions

Ordered list of Transactions

Block

Blockchain networks: tokens, transaction fees and mining rewards

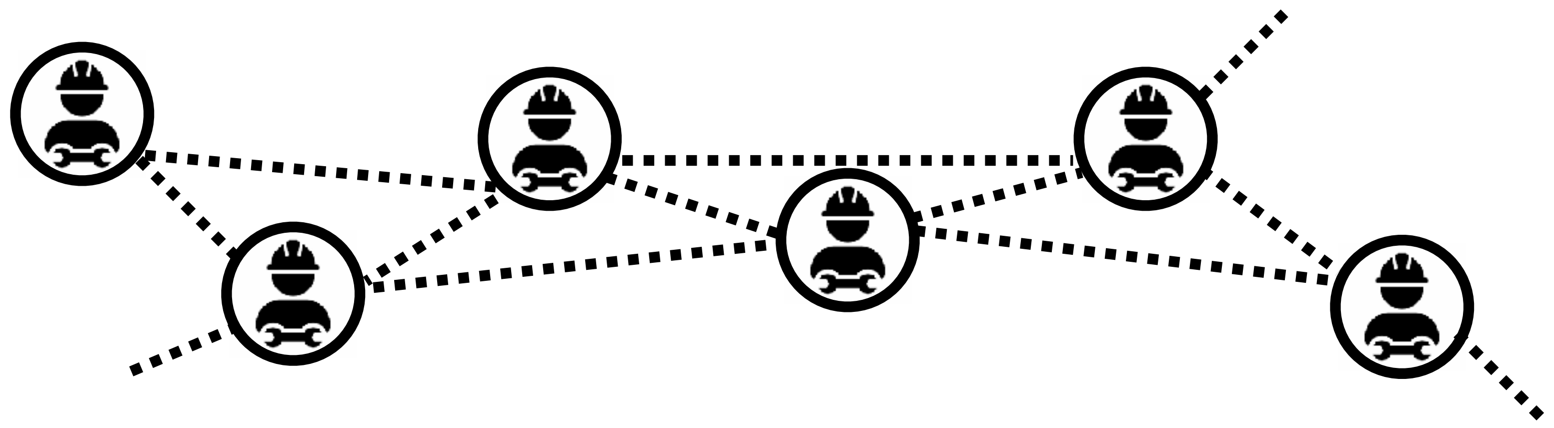
- **Tokens** are used to a) pay for transaction processing (transaction **fee**) and b) to **reward** validators for contributing hardware resources (compute, bandwidth, storage) to validate transactions. They act as an **incentive mechanism** to keep validators honest.



Validators can **earn** additional tokens
by producing valid blocks
(a process called "mining" or "staking")

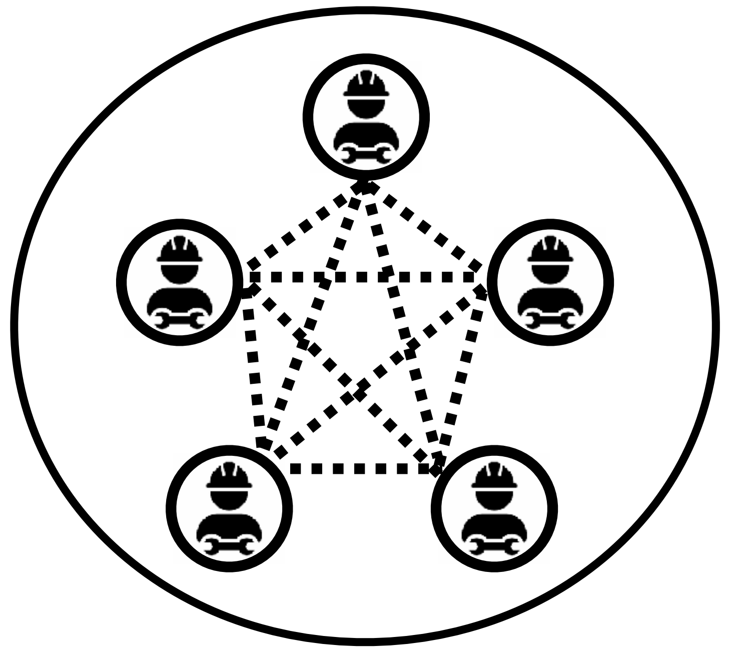
Who can be a validator?

- In **permissionless** blockchains: anyone can join the network to become a transaction validator. No need to ask for permission to anyone. Group membership is ***open***.
- In **permissioned** blockchains: must receive *permission* from a coordinator or from existing validators in order to become a transaction validator. Group membership is ***closed***.



Open

vs



Closed

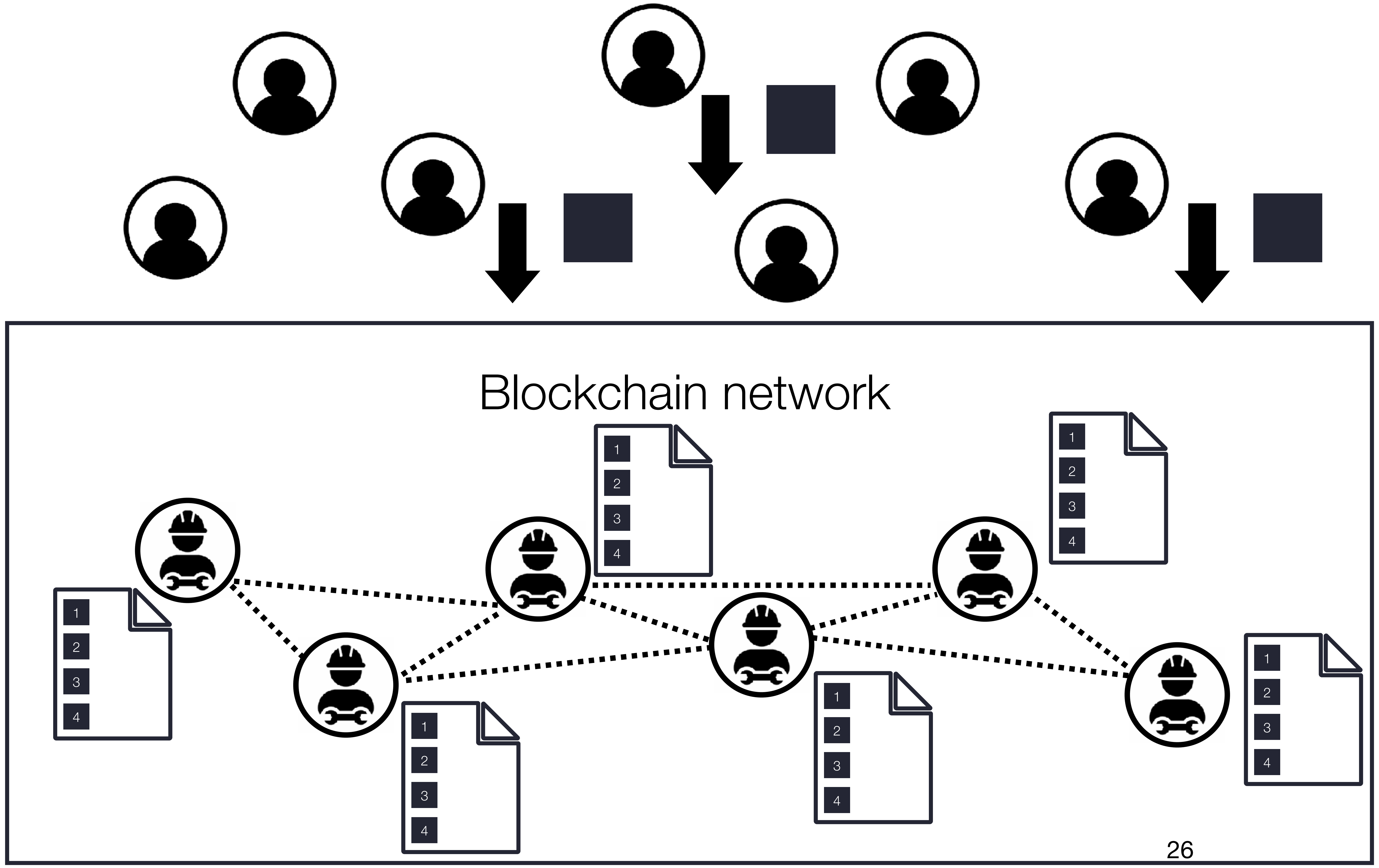
Bitcoin Ethereum Filecoin

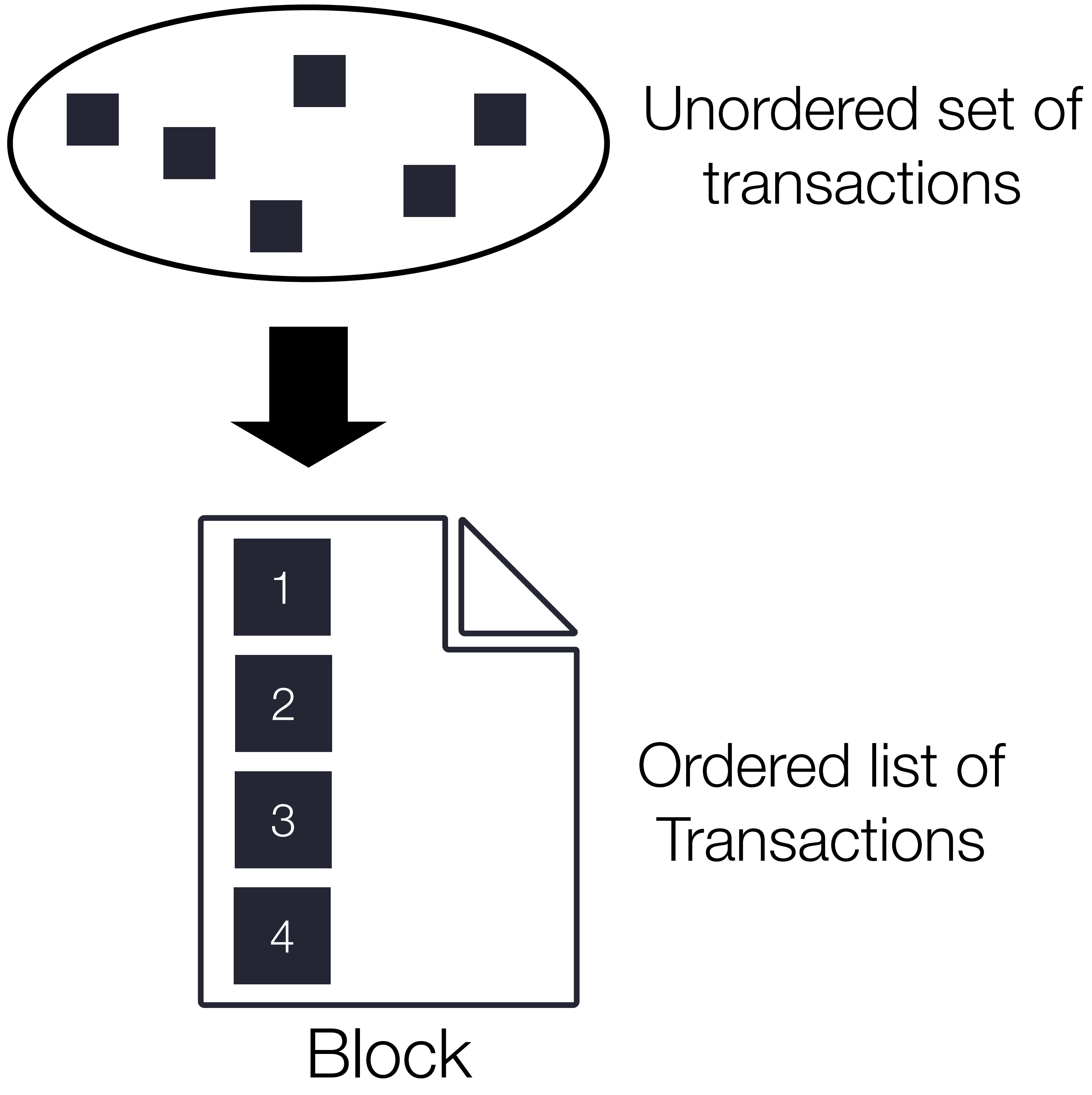
Hyperledger FABRIC HYPERLEDGER BESU corda

Consensus in Blockchain networks

Consensus in Blockchain networks: recap

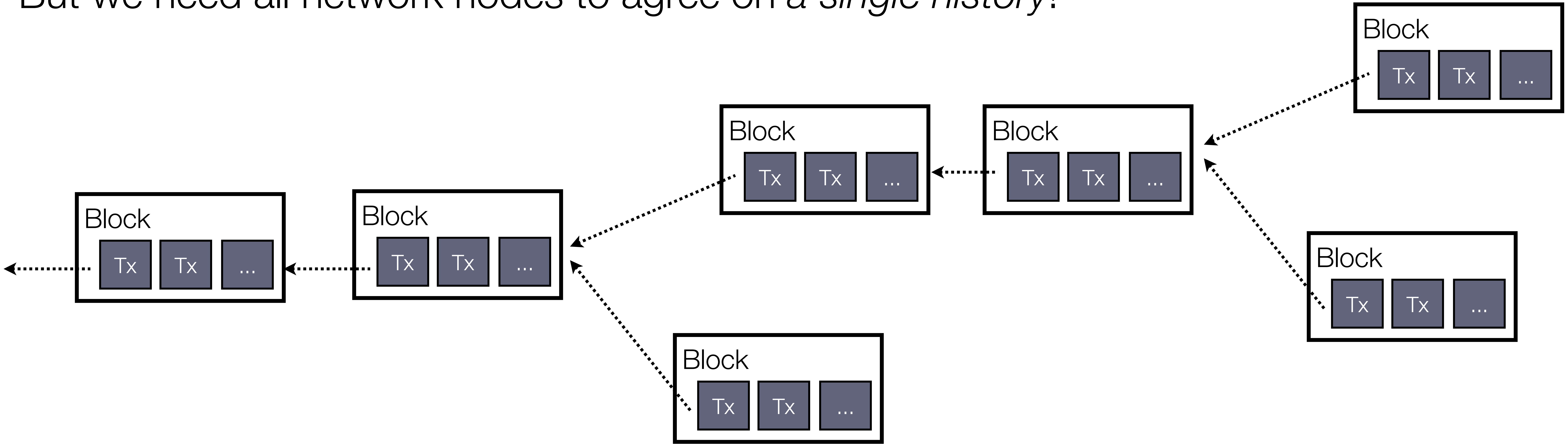
- All validators must reach **consensus** on the exact same transaction history!
- Need to make sure that blocks get appended everywhere *in the same order*





Problem: diverging histories

- In an open system, if *anyone* can easily produce a valid block and add it directly to the ledger, there is little hope that the network will end up agreeing on a *single* ledger
- More likely, we would end up with a quickly growing *tree* of blocks
- But we need all network nodes to agree on *a single history*!



How to get consensus: organize a vote?

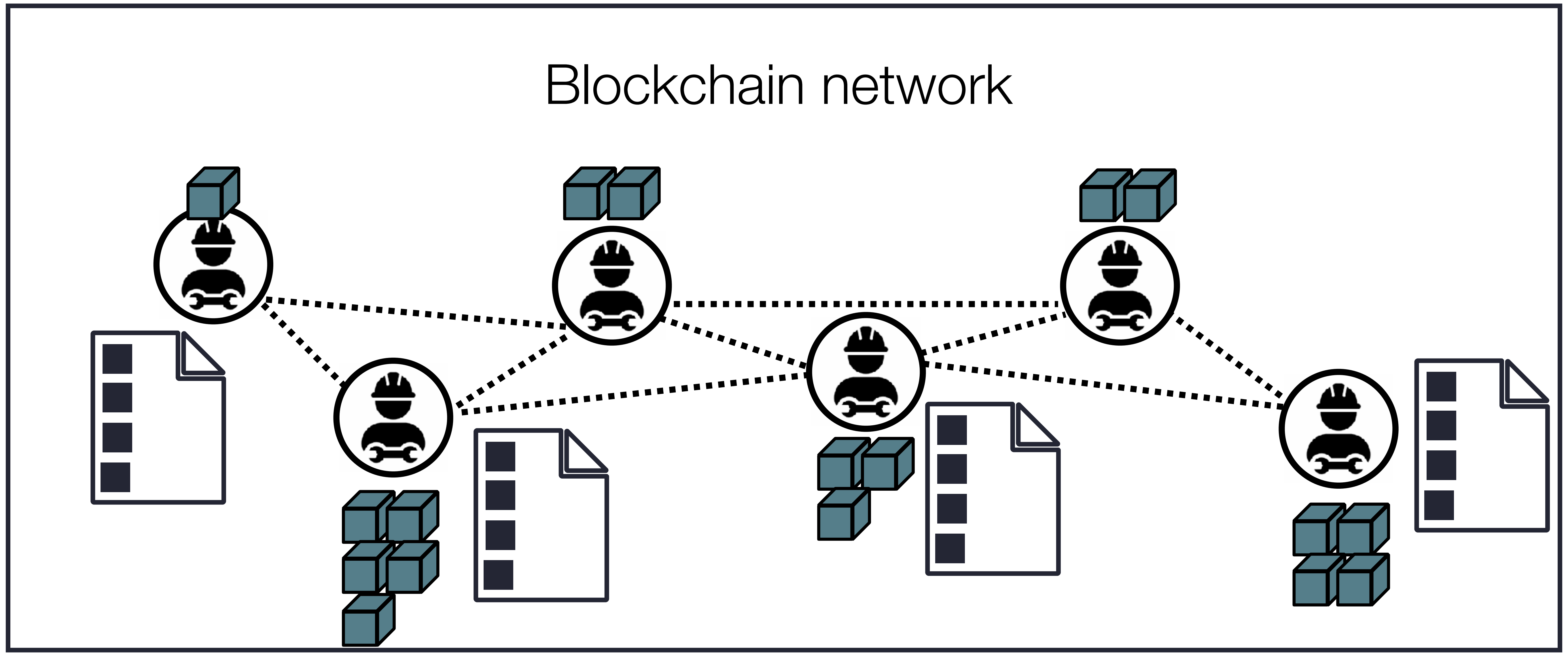
- We can let the network **vote** to **elect a single validator node** to propose the next block
- Ideally the proposer node is chosen **randomly** to avoid any bias in the election process
- But how to organize a vote in an open and permissionless network?
  - 1. We don't even have a fixed list of nodes to organize a voting poll
  - 2. Even if we would have a list of nodes, how to assign **voting rights** to each one?
- One IP address = one vote? Problem: attacker may control multiple IP addresses
- This is known as a **sybil attack**. The same problem holds for any other type of "identity" that is cheap to create (e.g. public keys)

How to get consensus: organize a lottery!

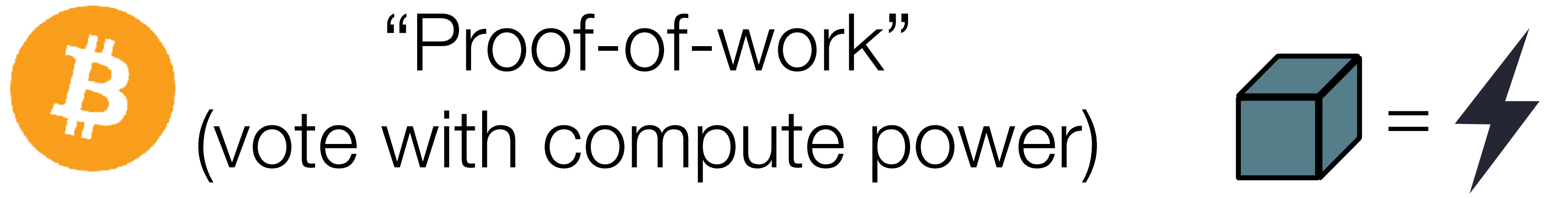
- To elect a node from an open group of participants, organize a **lottery**: each node “buys” tickets, whoever can “prove” they have the lucky ticket is the winner (and so gets to propose the next block)
- The lottery should have the following properties:
  - **Fair** - node election should be distributed across the broadest possible population of participants (i.e. “everyone can buy a ticket”)
  - **Proportional** - The cost of controlling the election process should be proportional to the value gained from it (i.e. “the more tickets bought, the higher the chance of winning”)
  - **Verifiable** - It should be relatively simple for all participants to verify that the winning node was legitimately selected (i.e. “everyone can verify whether the winning ticket is indeed a valid ticket”)

Lottery-based consensus in permissionless blockchains (“proof-of-X”)

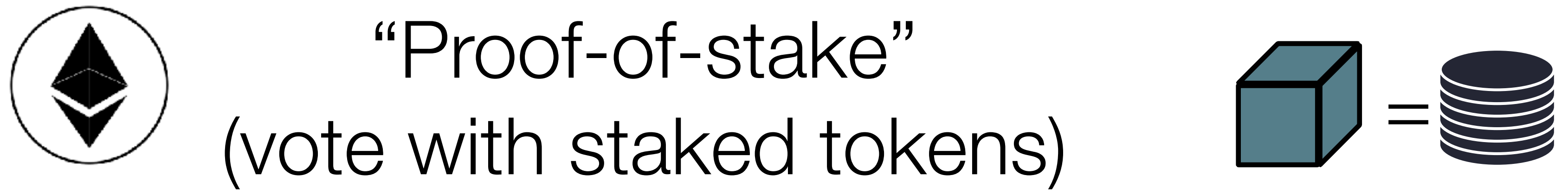
- Validators enter the lottery by proving ownership of a digital or physically **scarce resource**
- Different blockchain networks may use different kinds of resources



Example lottery-based consensus protocols:



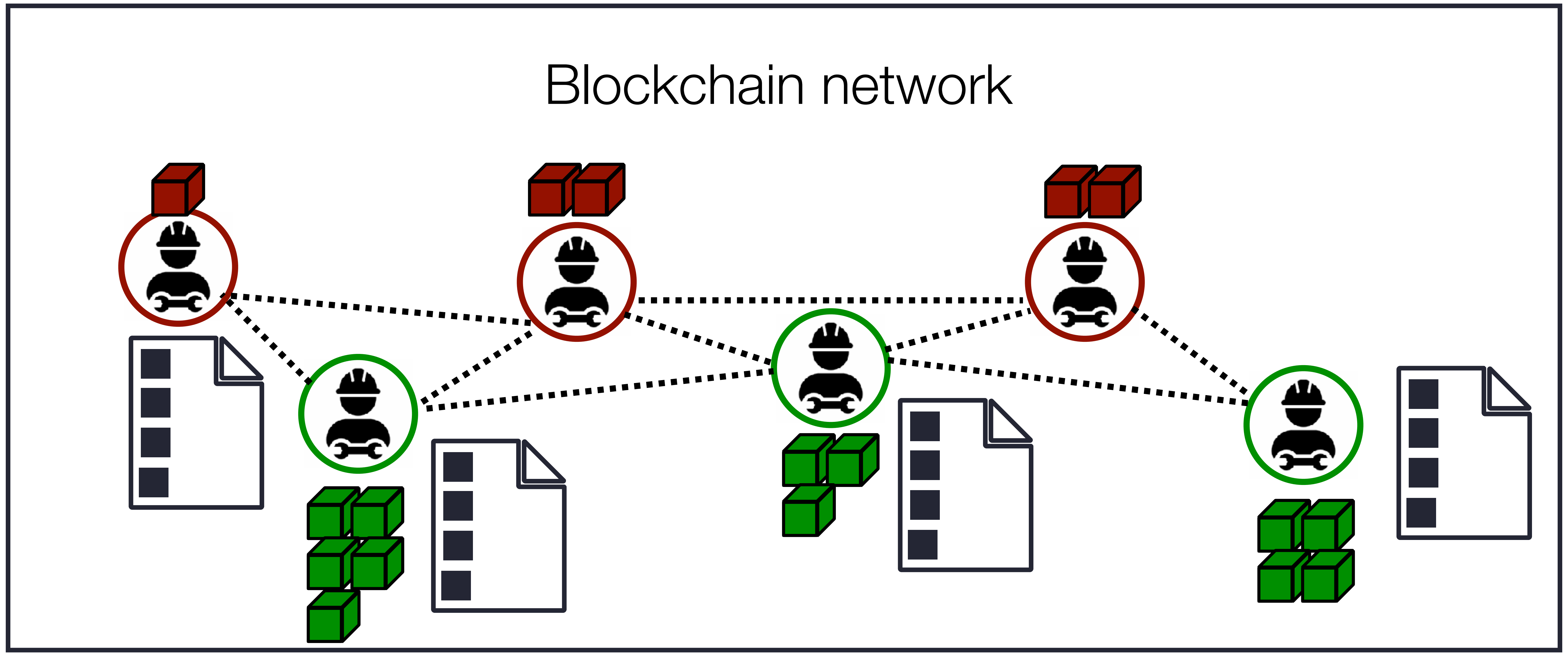
“Proof-of-work”
(vote with compute power)

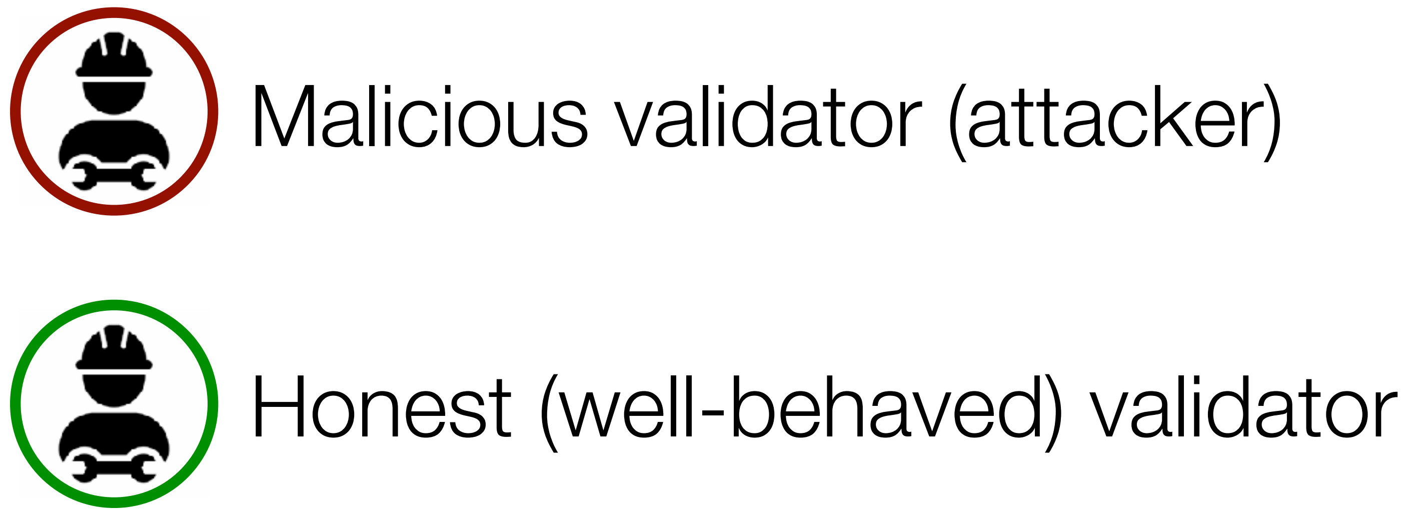


“Proof-of-stake”
(vote with staked tokens)

Lottery-based consensus in permissionless blockchains (“proof-of-X”)

- The integrity of the blockchain is guaranteed as long as a **majority** of the network, **weighted** by their resource ownership, is controlled by well-behaved validators

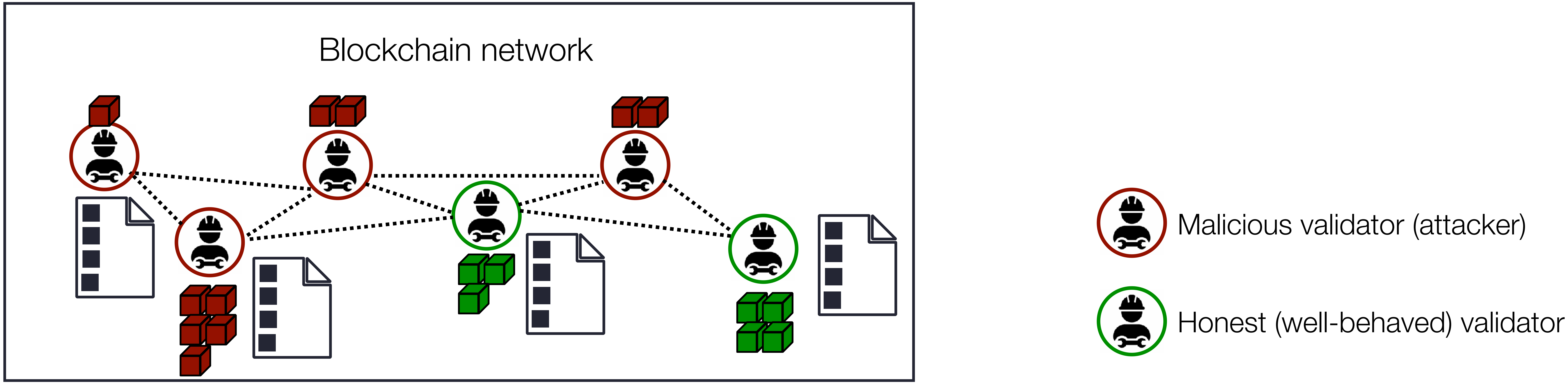




Attacking a permissionless blockchain network: “51% attack”

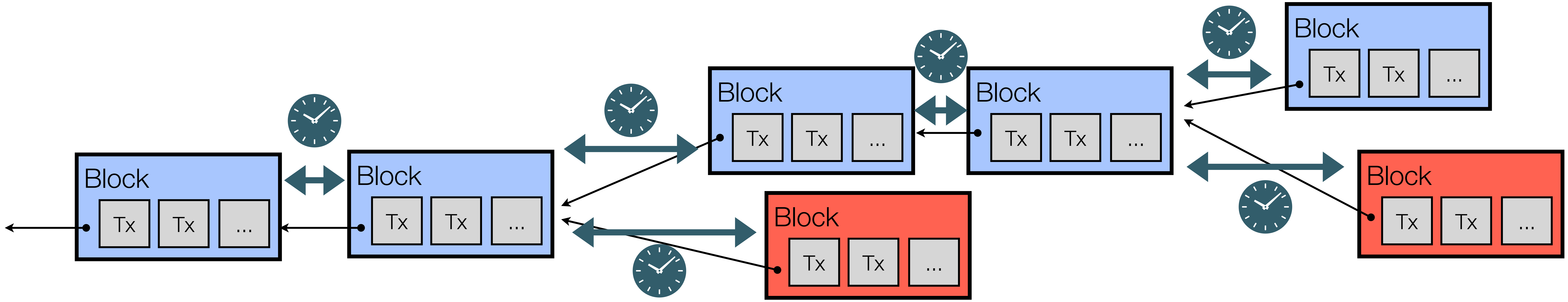
- If an attacker (or group of attackers) **controls >50% of the scarce resources**, they effectively control the production of new blocks.

- While such an attacker cannot create “fake” signed transactions (i.e. steal tokens), they can reject (**censor**) any number of transactions and can approve transactions that **double-spend** their own tokens by “forking” the blockchain and “rewriting” block history.



Proof-of-Work consensus

- Require blocks to contain a “proof-of-work”: a proof that significant (computational) work was done to find the solution to a puzzle, where the solution - once known - is easy to verify
- The purpose is to **slow down** block production
  - so that only **one node at a time** can propose a new block
  - so that there is time to **propagate the new block** across the entire P2P network to avoid disagreement on what is the latest valid block
- In Bitcoin: propose a new block on average every 10 minutes

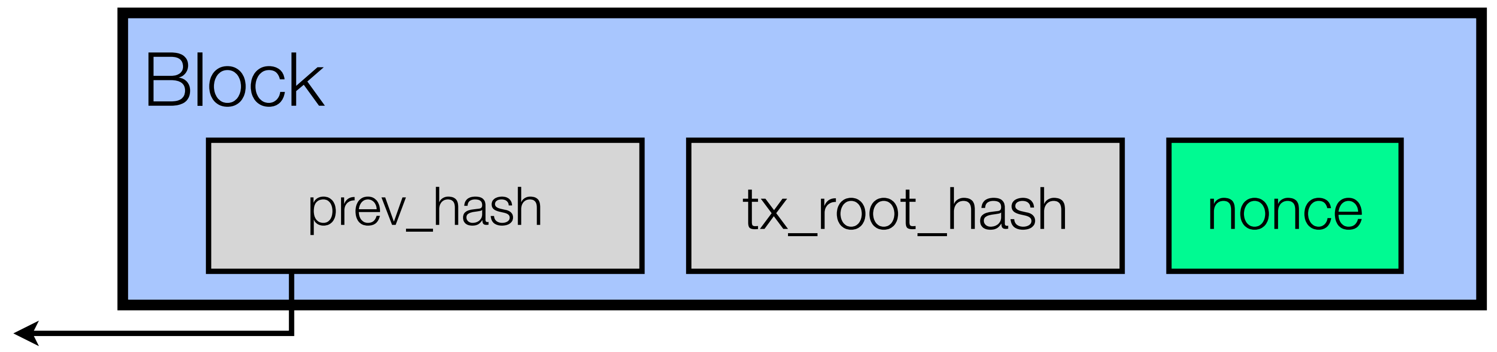


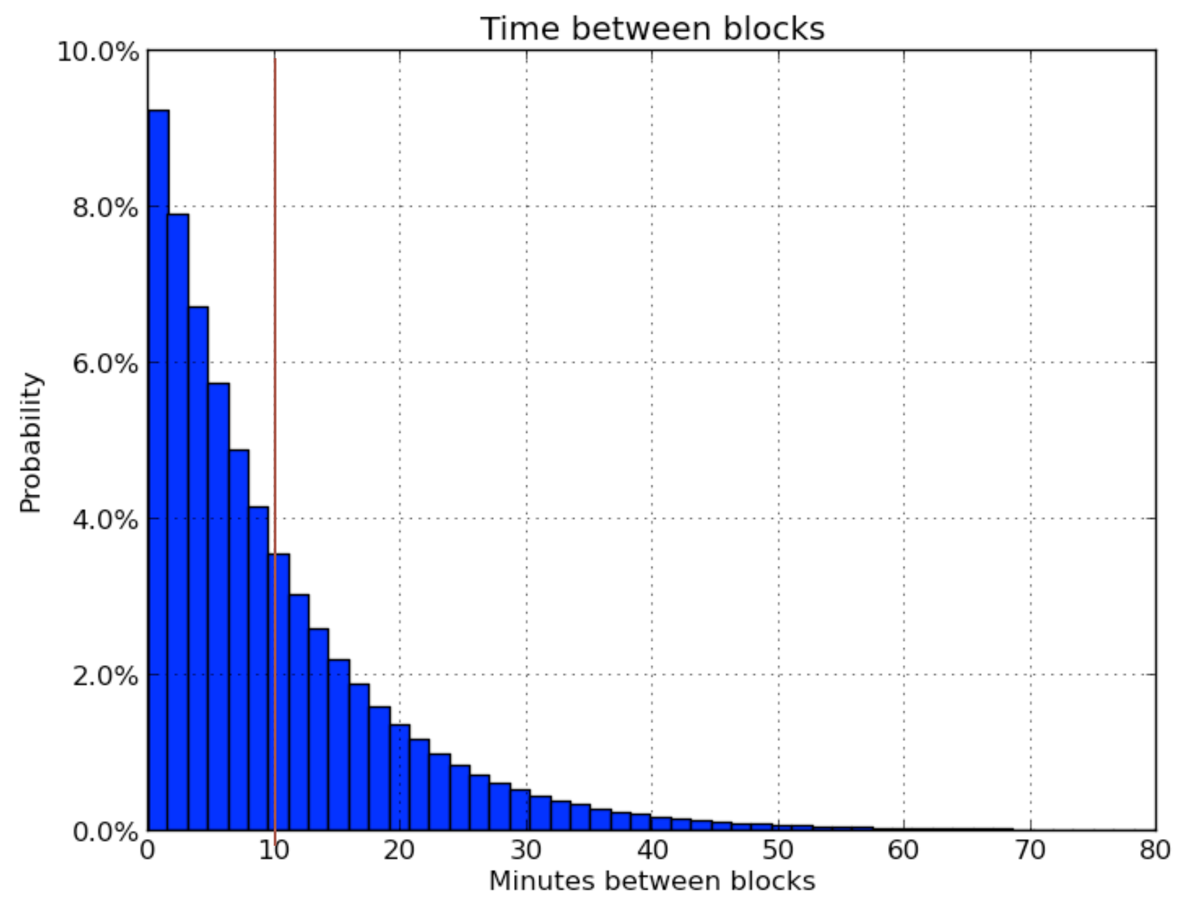
Proof-of-Work in Bitcoin

- The proof-of-work involves searching for a value *v* such that hash(*v*) is smaller than a given target threshold value (known as the **difficulty** parameter)
- Because the output of a cryptographic hash cannot be predicted, there is no known strategy better than a **brute force** search
- The search is done by incrementing a number in the block (the **nonce**) until a value is found such that the block's hash satisfies the target difficulty
- The difficulty parameter is **adjusted** every 2016 blocks such that the **average time** between blocks remains 10 minutes.

Find a nonce (a number) such that:

$$H\text{ (nonce }||\text{ prev\_hash }||\text{ tx\_root\_hash) } < \text{ target}$$



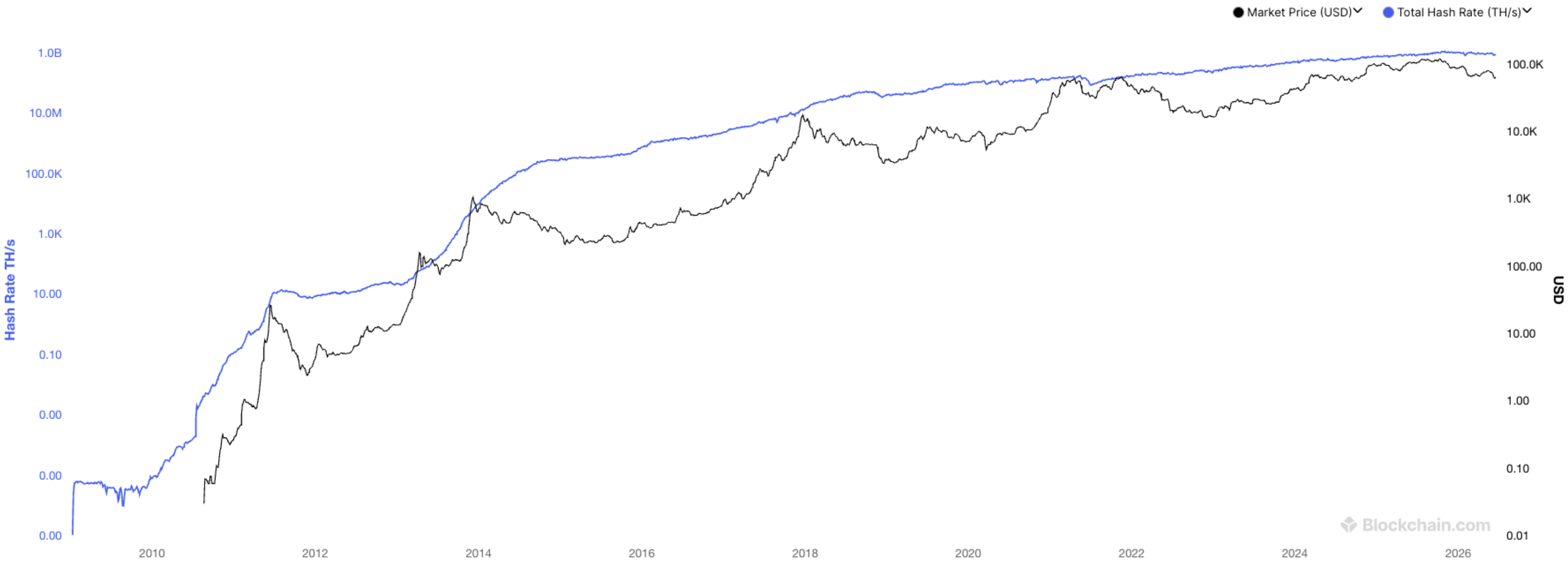


(source: Bitcoin wiki, retrieved November 2022)

Proof-of-Work: change in network compute power over time

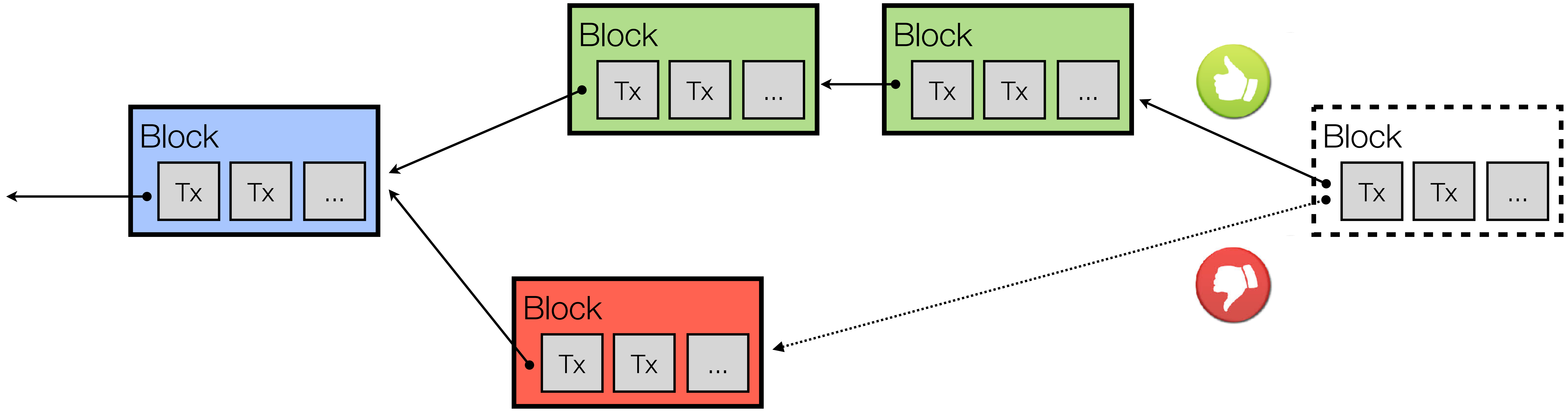
- Difficulty adjustment is needed because the total amount of compute power available to the network is constantly changing over time.

Note: **logarithmic** Y-axis!



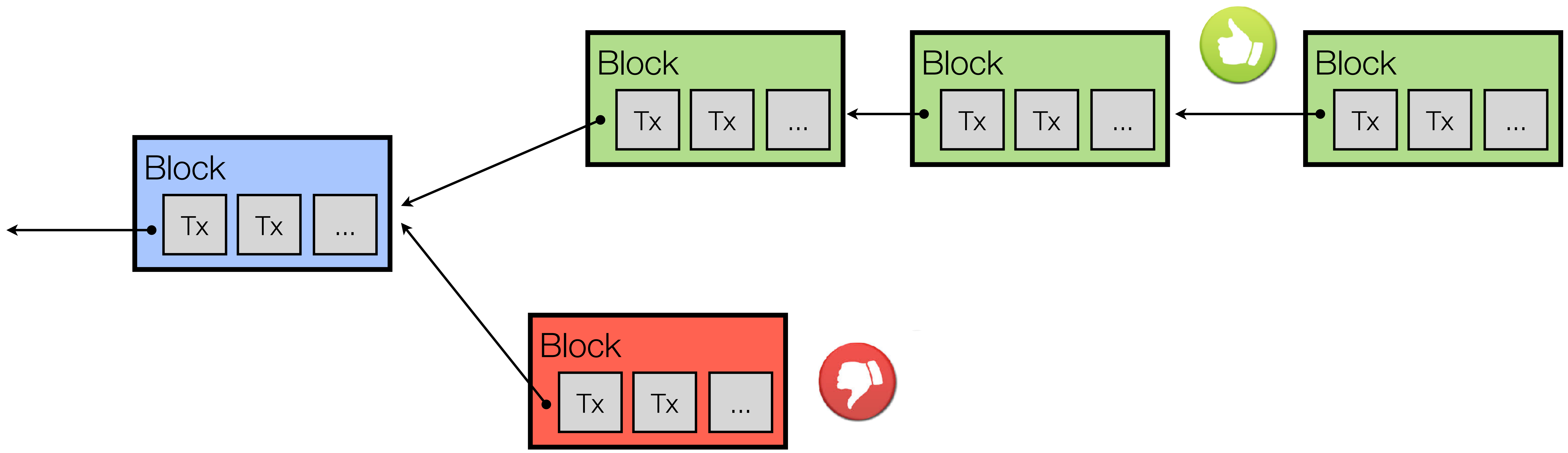
How Proof-of-Work solves the consensus problem

- Nodes implicitly “vote” with their computational power
- Nodes silently accept a block by working on extending the block (= mining)
- Nodes silently reject a block by refusing to work on it
- The majority decision is represented by **the longest chain**, which has the greatest proof-of-work effort invested in it.



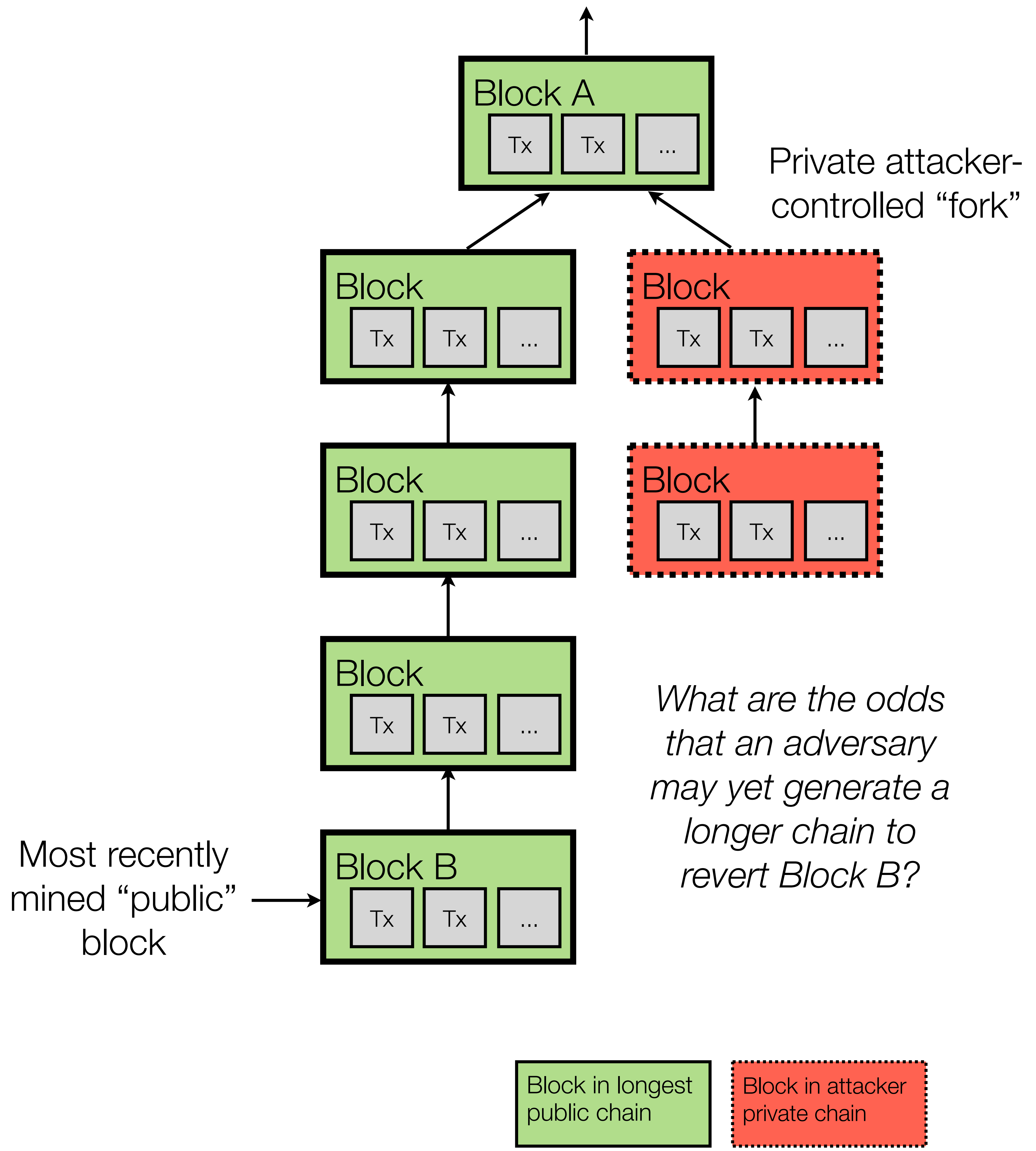
How Proof-of-Work solves the consensus problem

- Nodes implicitly “vote” with their computational power
- Nodes silently accept a block by working on extending the block (= mining)
- Nodes silently reject a block by refusing to work on it
- The majority decision is represented by **the longest chain**, which has the greatest proof-of-work effort invested in it.



Block finality under “Nakamoto Consensus”

- How do we know that we have truly seen the “longest” chain? We can’t!
- Bitcoin’s longest-chain rule provides only ***probabilistic finality*** guarantees. Block “re-orgs” near the tip of the chain are always possible.
- If we assume a **majority** of hash power is controlled by honest nodes, then the honest chain will grow the fastest and outpace any competing chains.
- Put differently: an attacker must control more compute power than **all the honest nodes combined** in order to overtake the “honest chain”



KU LEUVEN DistriNet

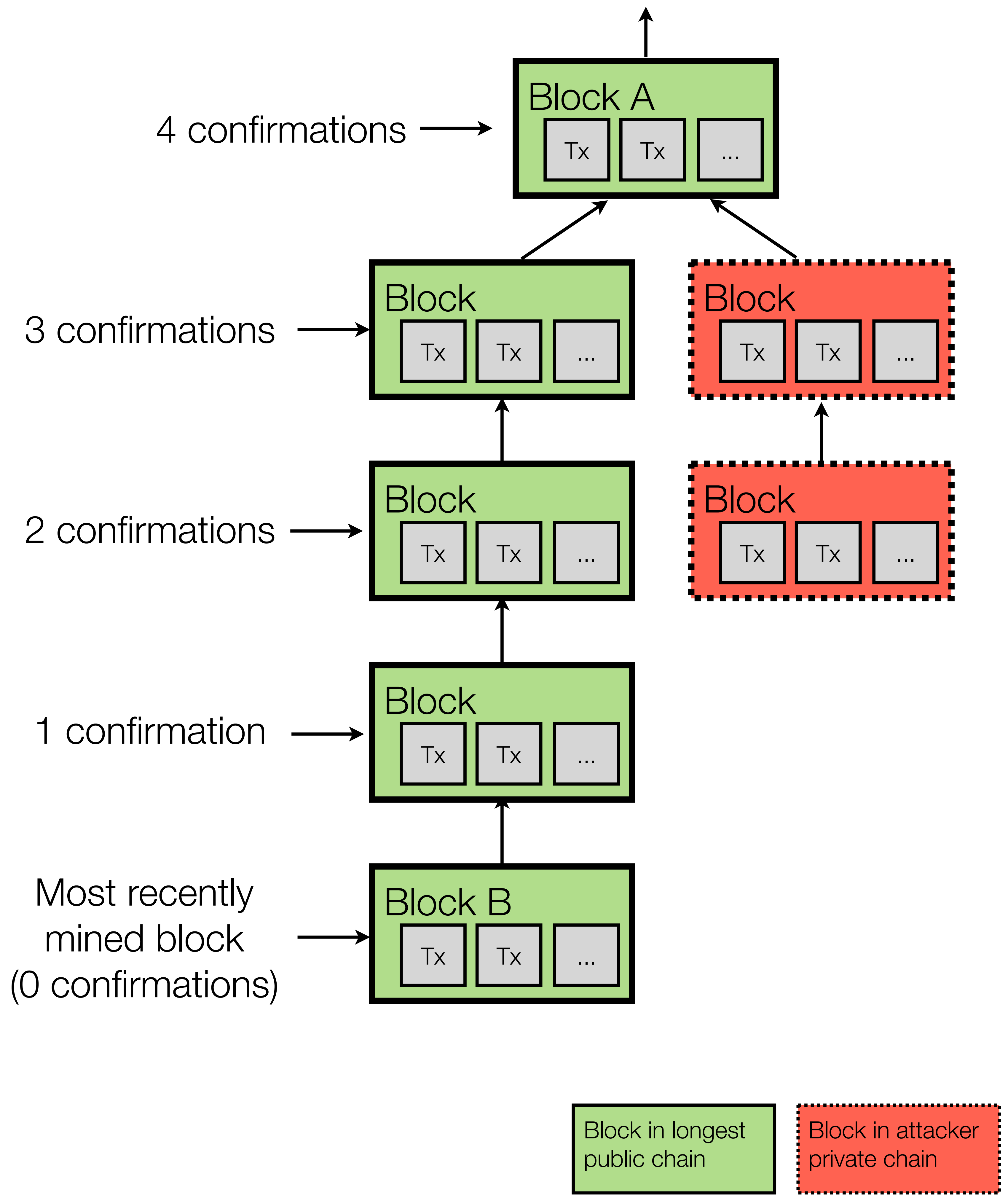
Block finality under “Nakamoto Consensus”

- Bob should only consider Alice’s transaction **settled** when her transaction is included in a block that is “buried” under multiple more recent blocks.

- Every block added *after* the block is an additional **confirmation** that the network has accepted the block as part of the longest chain.

- The more block confirmations, the smaller the chance that another, longer chain will be discovered. The probability **drops exponentially** with each confirmation.

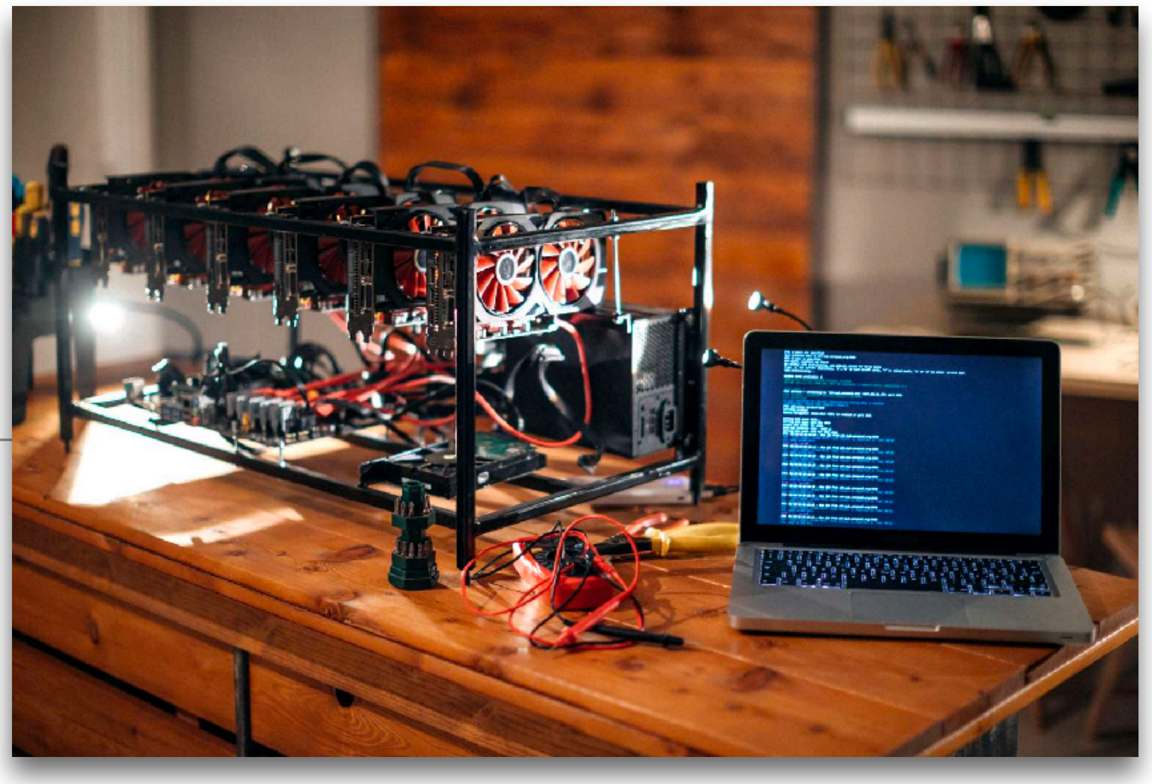
- **Six block confirmations** (~1 hour wait time) is considered the gold standard to consider a block and its transactions as final.

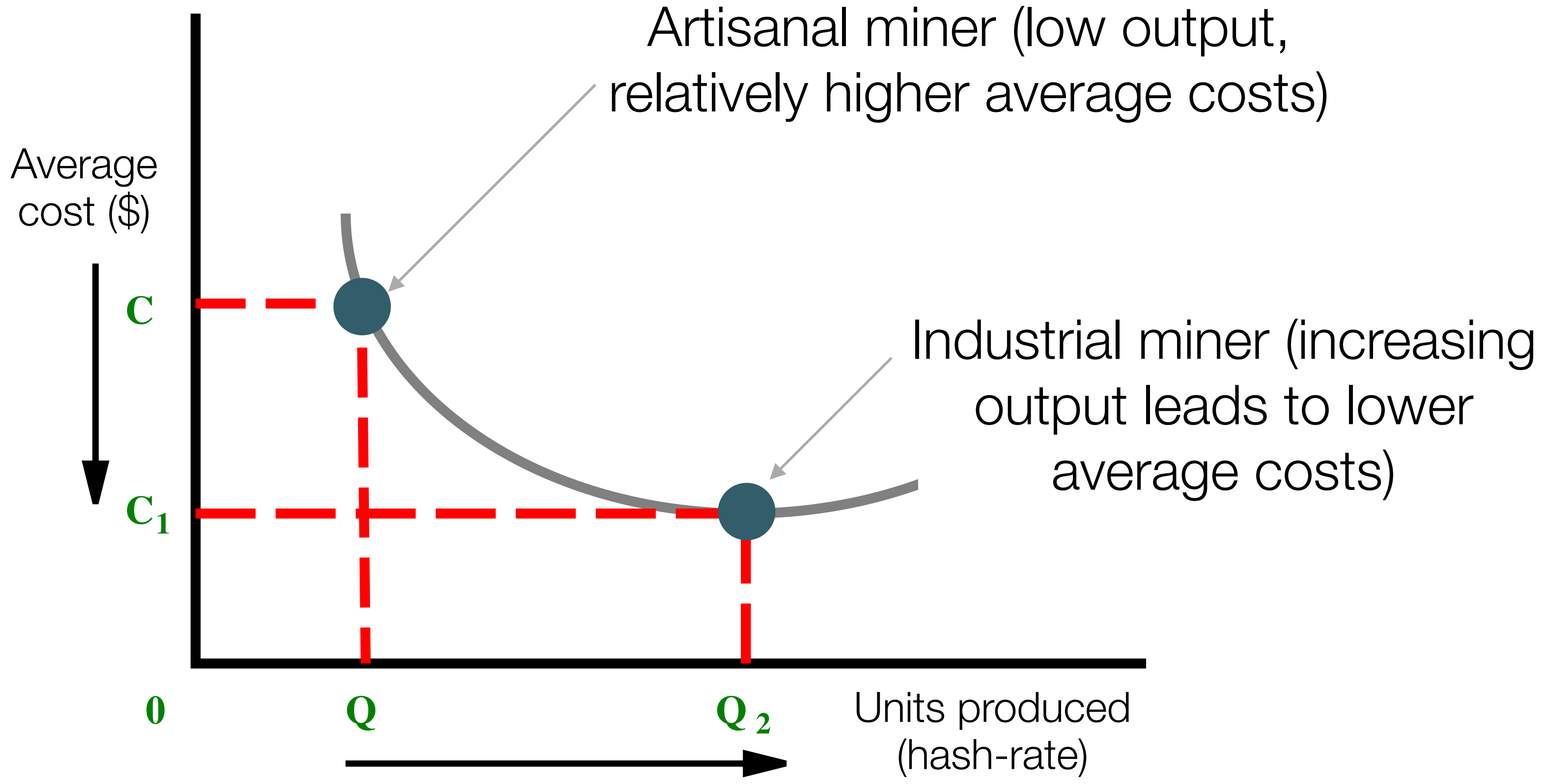


Proof-of-Work

- Miners “race” each other to find the next block. The more computational power a miner has, the higher the chance of winning the race.
- But Proof-of-Work is:
  - Slow (by design)
  - Energy-inefficient (by design)
  - Subject to centralizing economies of scale (mining pools, large-scale mining facilities)

A small-scale GPU-based Bitcoin “mining rig”
(Image credit: investopedia.com)





A large-scale ASIC-based Bitcoin “mining farm”
(Image credit: stockhouse.com)

Proof-of-Stake

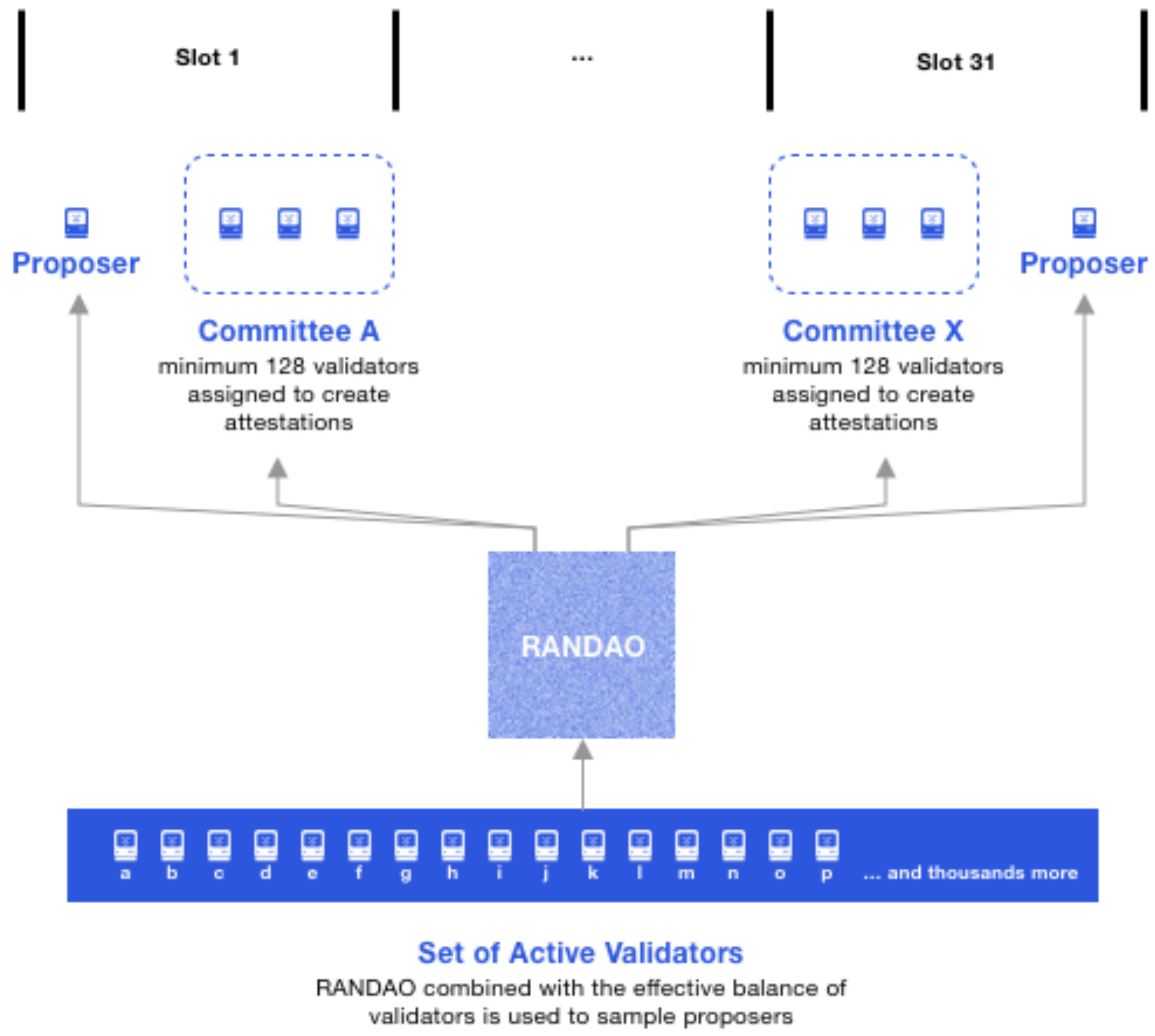
- Proof-of-Stake (PoS): the chance of proposing the next block is **proportional to the economic stake** in the system.
  - The more tokens “staked” (= locked in escrow), the higher the chance of becoming the next block proposer.
- Many **variations** of PoS **exist**. Two large families include:
  - **Lottery-based** Proof-of-Stake. Similar to Proof-of-Work. Also called **chain-based** Proof-of-Stake.
  - **Voting-based** Proof-of-Stake. Also called **BFT-based** Proof-of-Stake. Tolerate f malicious nodes among $N = 3f+1$ validators (attacker cannot control >= 1/3 of network)

Ethereum’ Proof-of-Stake consensus

- Ethereum's PoS is not a single algorithm — it combines two fundamentally different consensus primitives:

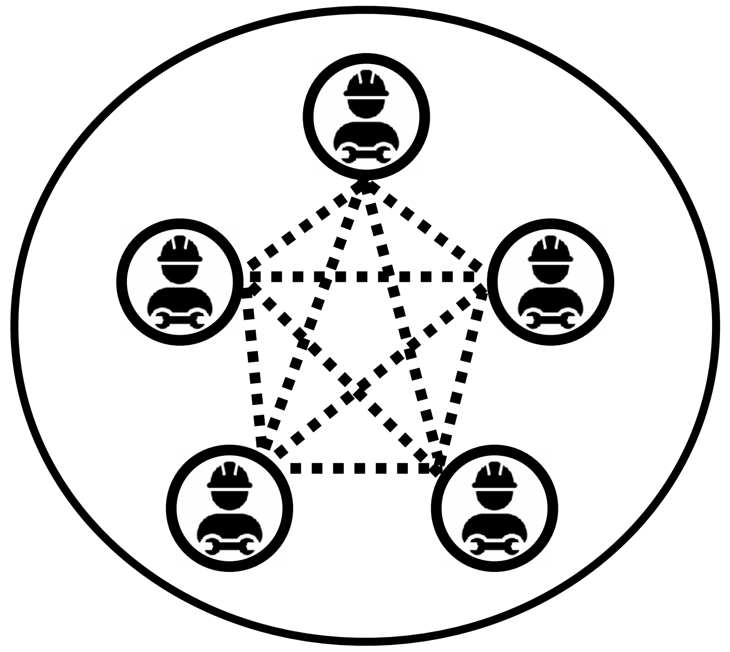
- Validator *selection* is permissionless and probabilistic, while block *finality* is permissioned and deterministic.

Layer	Problem solved	Algorithm family	Participants
Validator selection	Who gets to vote?	Lottery	All stakers (≥32 ETH) (Many 1000s)
Block finality	What is the canonical chain?	BFT voting	Selected committee (128 validators)



Permissioned Blockchains

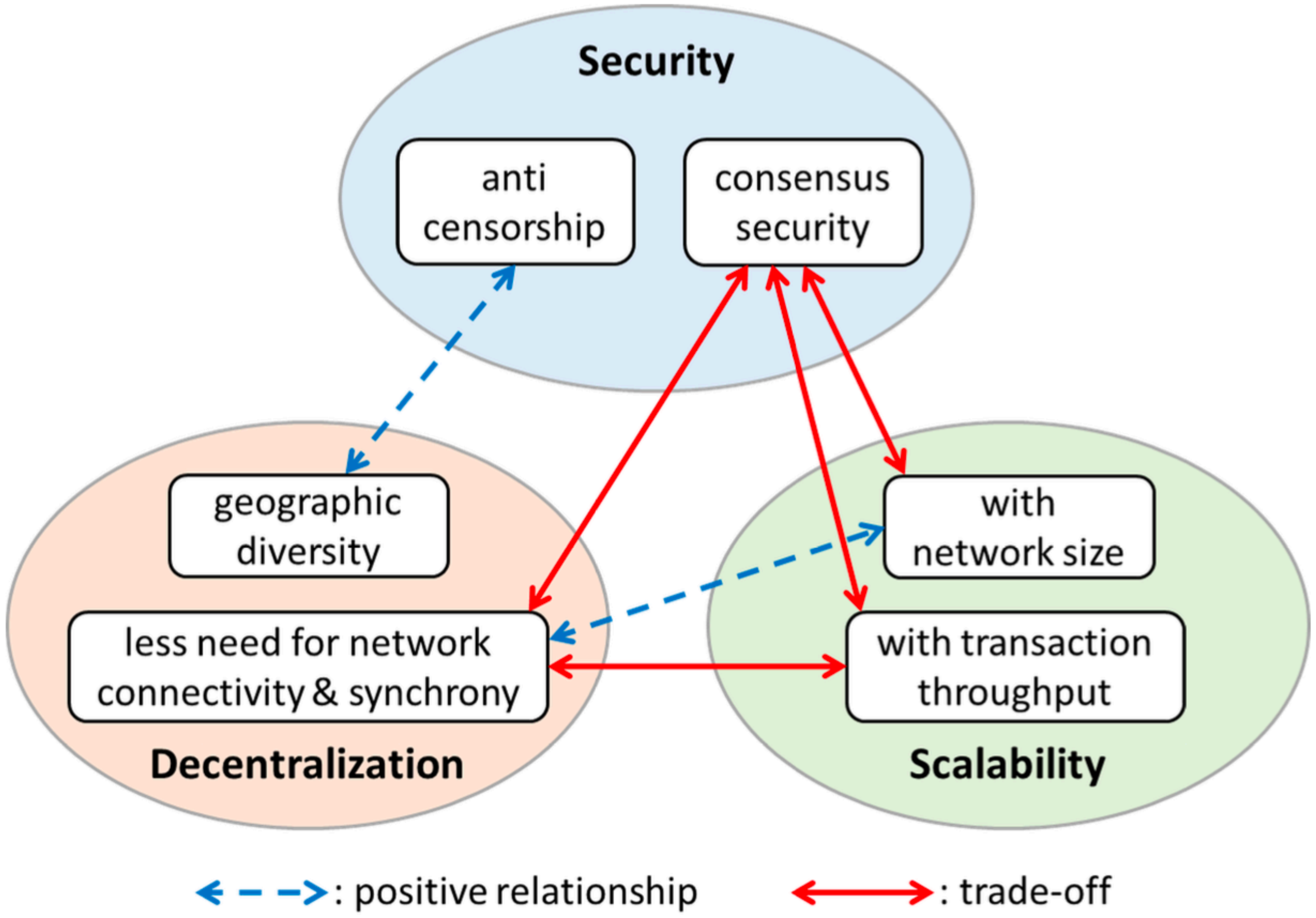
- Avoid the privacy and scalability challenges of permissionless blockchains by **limiting** readers and/or writers **to** a set of **authorised parties only**
- This **avoids** the **sybil attack** problem and the need to use "**lottery**"-based consensus (Proof-of-Work, Proof-of-Stake, …)
- Instead, use standard "**voting**"-based byzantine consensus algorithms:
    - Need $\boldsymbol{N} = 3\boldsymbol{f} + 1$ nodes to tolerate $\boldsymbol{f}$ incorrect nodes (or: a ⅔+1 honest, available supermajority)
    - Use of closed-group BFT algorithms in Blockchain sometimes called "Proof-of-Authority"



Hyperledger FABRIC HYPERLEDGER BESU corda

The Blockchain Trilemma

- "**Security**, **Decentralization**, **Scalability**: pick two" - in practice it is hard to achieve all three simultaneously.
- These are *engineering* trade-offs, not fundamental impossibilities
- Permissionless blockchains maximize security and decentralization at the cost of reduced scalability (throughput).
- Permissioned blockchains achieve high scalability (throughput) at the cost of increased centralization
- Not a binary choice, but a spectrum.



(Source: Xiao et al., "A Survey of Distributed Consensus Protocols for Blockchain Networks", IEEE COMST 2020)

Permissioned vs Permissionless Blockchains: summary

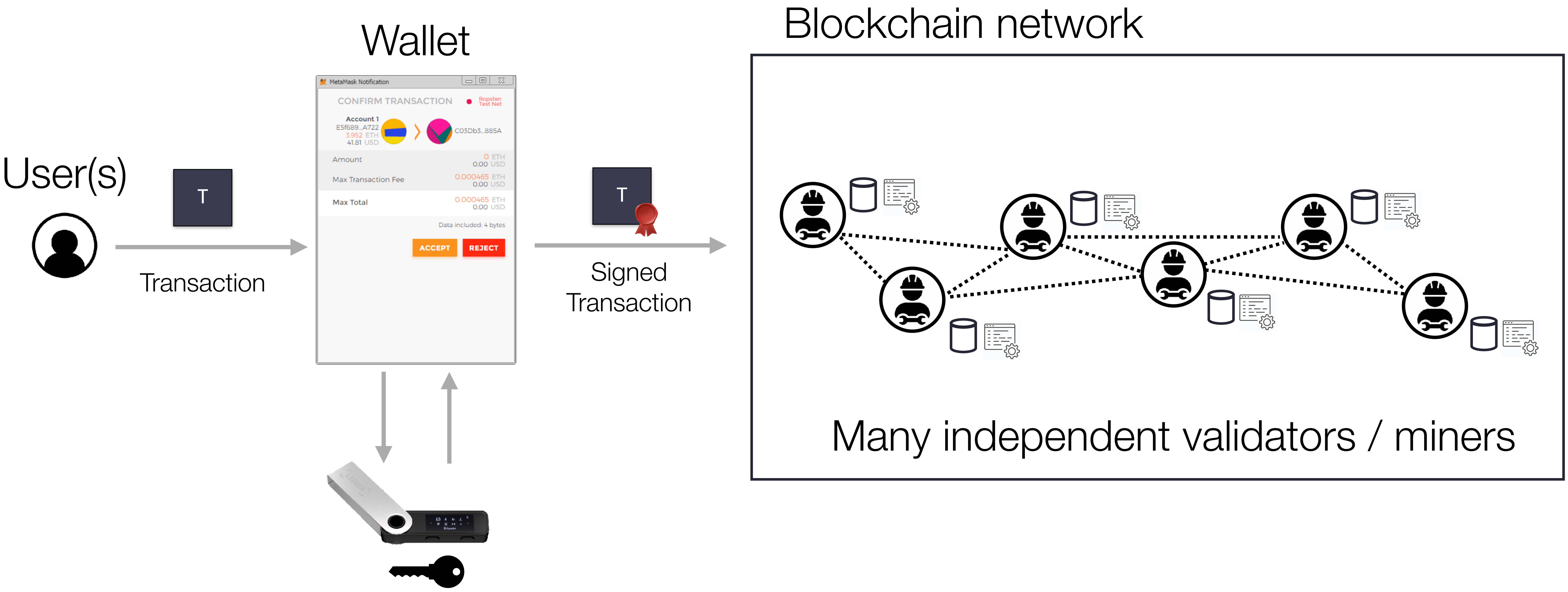
	Permissionless	Permissioned
Network peers	Validator nodes are **pseudonymous**.	Validator node **identity** is usually **revealed**.
Peer membership	**Open:** anyone can join, no need to ask "permission" to join	**Closed:** an administrator authorizes membership, or pre-existing members vote to update the membership list
Network size	Scales to **large** number of peers (>1000 nodes)	Usually **small** (< 100 nodes)
Network connectivity	**Low degree:** not all peers may be able to connect to all other peers. Use gossip broadcast.	**High degree:** often fully interconnected - direct broadcast.
Consensus achieved via	**Lottery**-based algorithms, prove ownership of scarce resource (e.g. Proof-of-Work, Proof-of-Stake)	**Voting**-based algorithms, such as Byzantine Fault-tolerant (BFT) consensus algorithms (e.g. PBFT). 1 node = 1 vote.
Transaction throughput	**Low:** ~10 TPS for Bitcoin, Ethereum. Generally: the larger the network, the lower the TPS.	**High:** >= 10,000 TPS (transactions per second)
Transaction & Block finality	**Probabilistic & slow:** blocks are considered final only after being extended by enough newer blocks, which can take minutes/hours.	**Deterministic & fast:** blocks are considered final as soon as 2/3 of validators accepted it, which may take < 1 second.
Safety threshold	At least **50% of a scarce resource** under the control of honest (correct) peers.	At least $2f + 1$ honest (correct) peers for every f byzantine peers ($N - f = 2f+1$). In other words, >2/3 or **67%** honest.
Energy-efficiency	Very **low** for Proof-of-Work. **High** for Proof-of-Stake.	**High:** comparable to a standard replicated database.

Applications of Blockchains beyond Payment ledgers

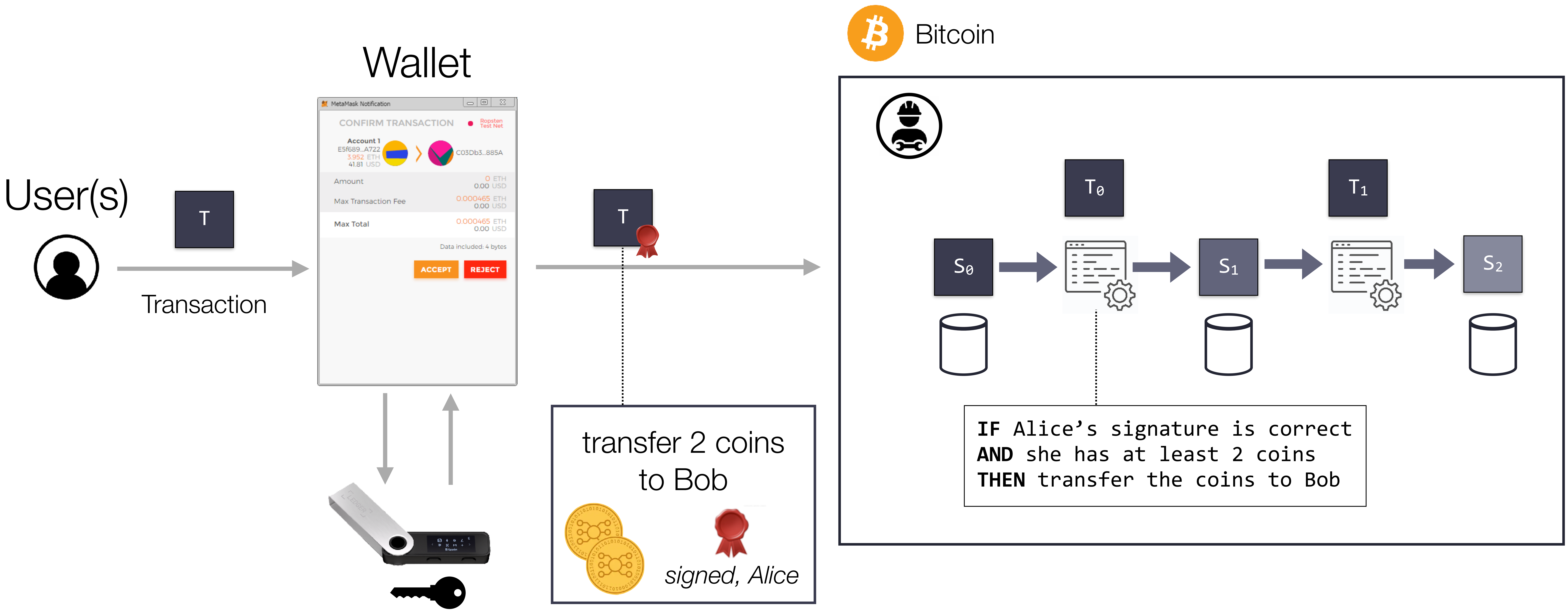
Application	Payload	Data structure	Examples
Decentralized Payment ledger	Payment transactions (transfer coins from A to B)	Blockchain blocks directly store payment transactions	Bitcoin
Decentralized Notary / timestamp service	Store document hash (not the content itself)	Store metadata on existing blockchain	OpenTimestamps (anchors into Bitcoin)
Decentralized Public Key Infrastructure (PKI)	Public key ↔ identity metadata	Host blockchain + smart contract	W3C DIDs (did:ethr, did:ion) Ethereum Attestation Service (EAS)
Decentralized Name registry	Human-readable name ↔ address	Host blockchain + smart contract	Ethereum Name Service (ENS) Unstoppable Domains

Blockchains as trusted computers: **smart contracts** and **Ethereum**

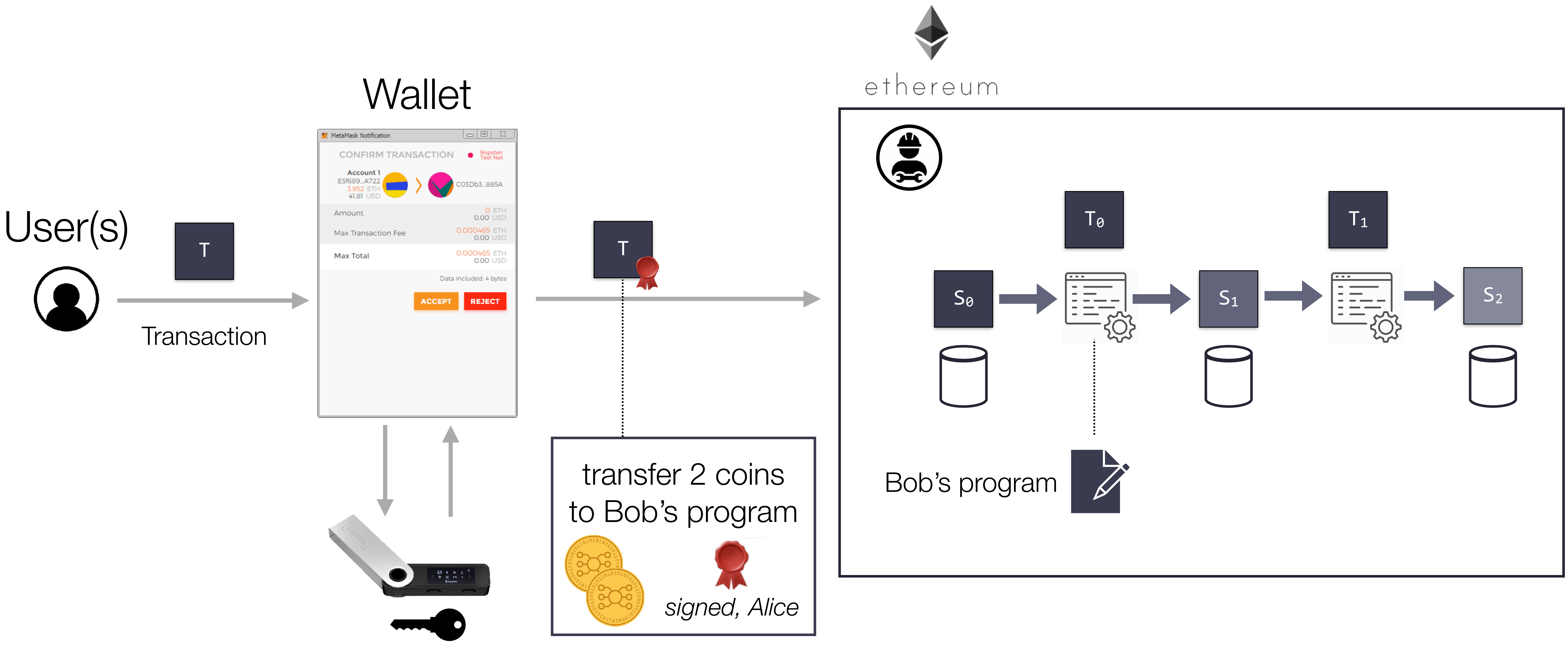
Physical view: a blockchain is a peer-to-peer network of computers



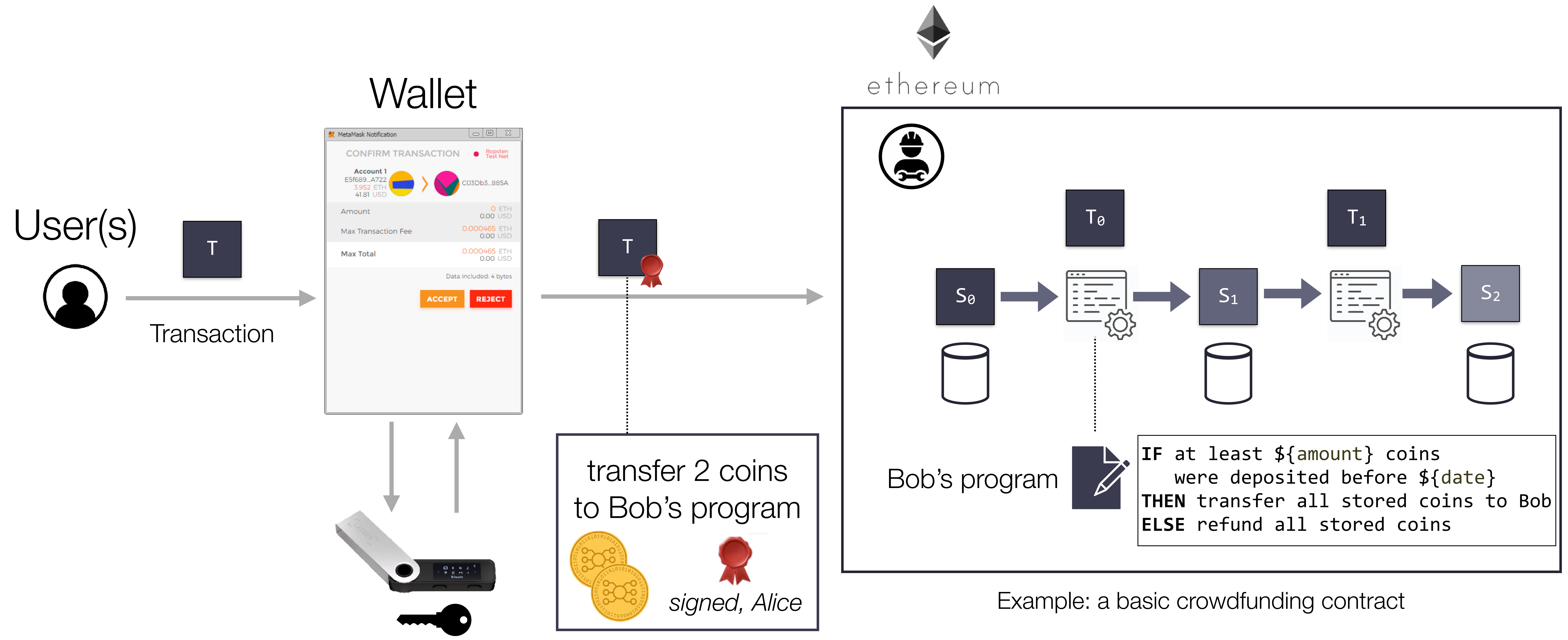
Logical view: a blockchain is a transaction processing machine



Ethereum’s innovation: make the transactions programmable!

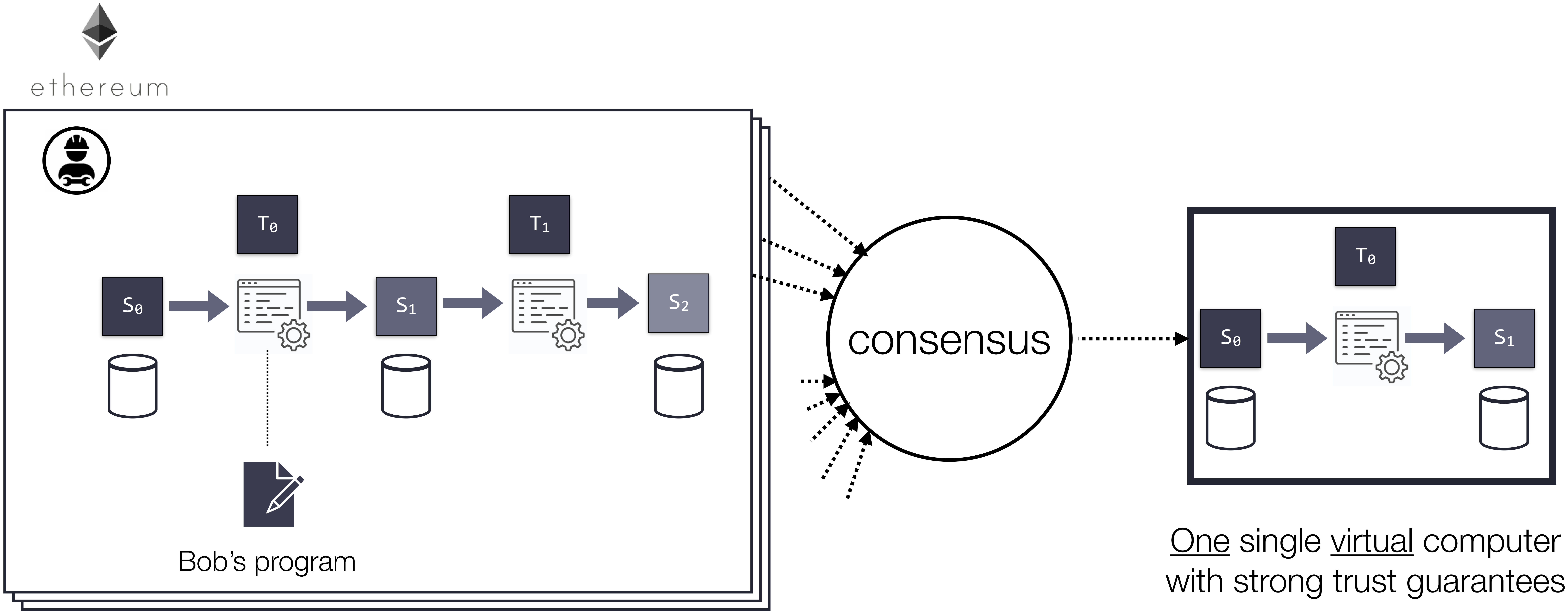


Ethereum's innovation: make the transactions programmable!



Example: a basic crowdfunding contract

Blockchains as *trusted* virtual computers



One single virtual computer with strong trust guarantees

Many (1000s) untrustworthy physical computers

Smart contracts: basic principle

- A vending machine is an **automaton** that can trade **physical** assets

1. insert coins

2. dispense drink

Smart contracts: basic principle

- A smart contract is an **automaton** that can trade **digital** assets

code

1. insert digital coins (tokens)

2. dispense other digital assets or electronic rights

But who should we trust to faithfully execute the automaton’s code?

- A smart contract is an **automaton** that can trade **digital** assets

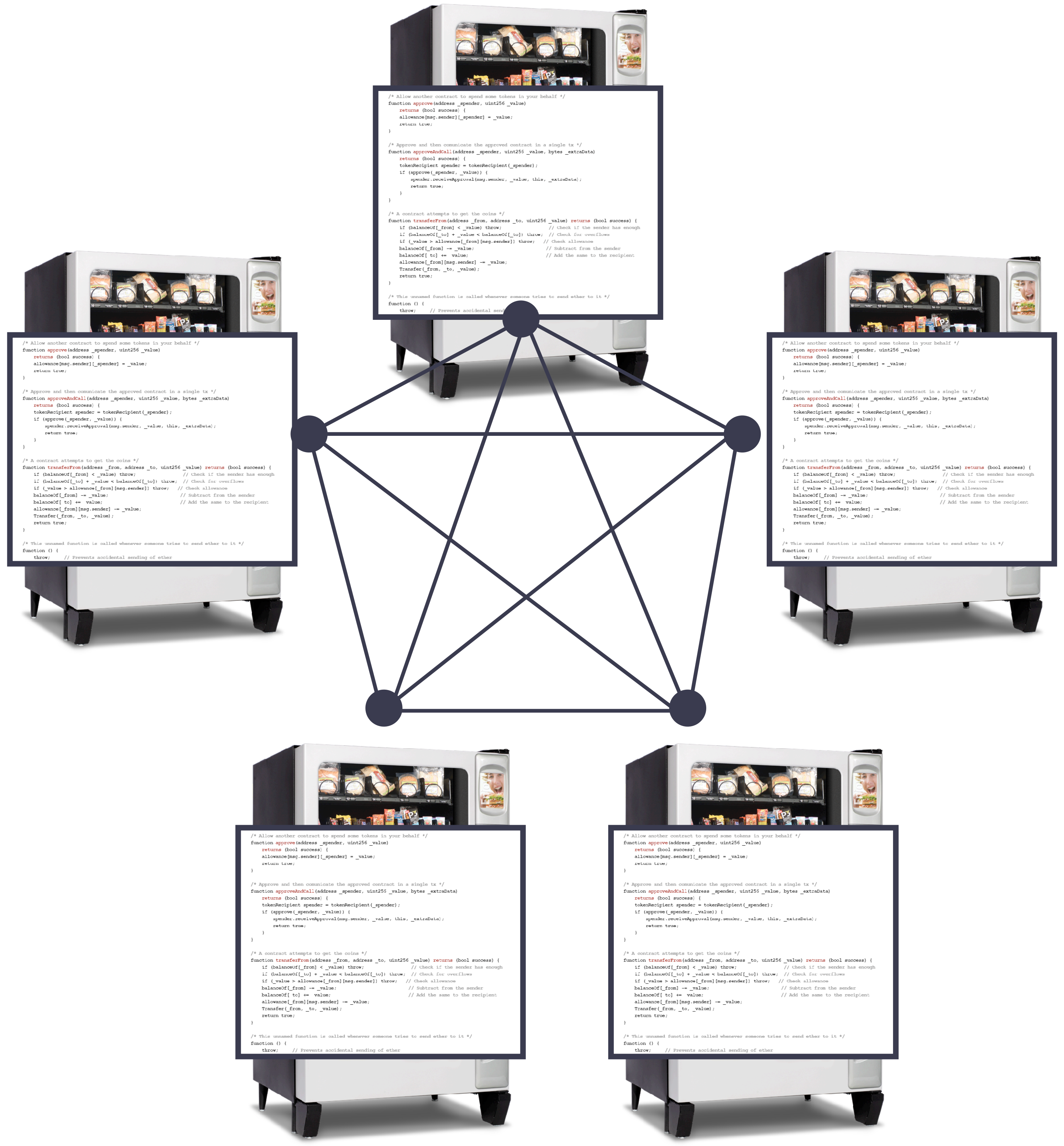
code

1. insert digital coins (tokens)

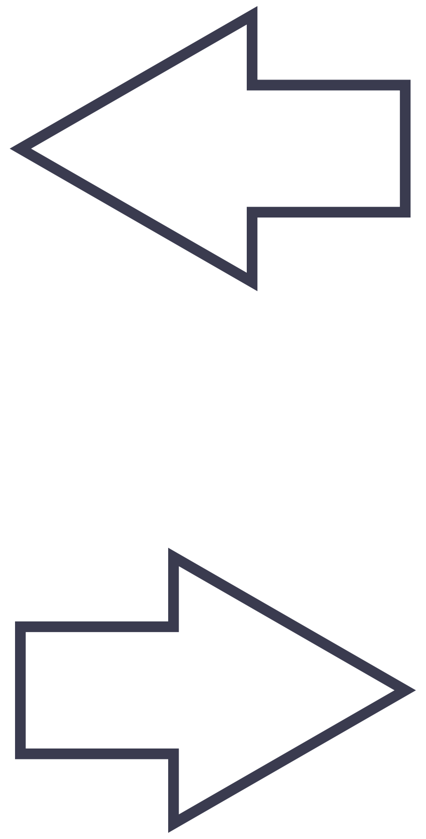
2. dispense other digital assets or electronic rights

Delegate trust to a decentralised network (= blockchain!)

- A smart contract is a **replicated automaton** that can trade **digital** assets



replicated code



1. insert digital coins (tokens)

2. dispense other digital assets or electronic rights

KU LEUVEN DistriNet

Contracts are compiled into bytecode for a simple stack machine

Bob's contract

network of validator nodes

.sol

solidity

solidity source code

Solidity compiler

EVM bytecode

Ethereum Virtual Machine

Example: crowdfunding as a smart contract

- Can we model a crowdfunding campaign as an automaton?

crowdfunding contract

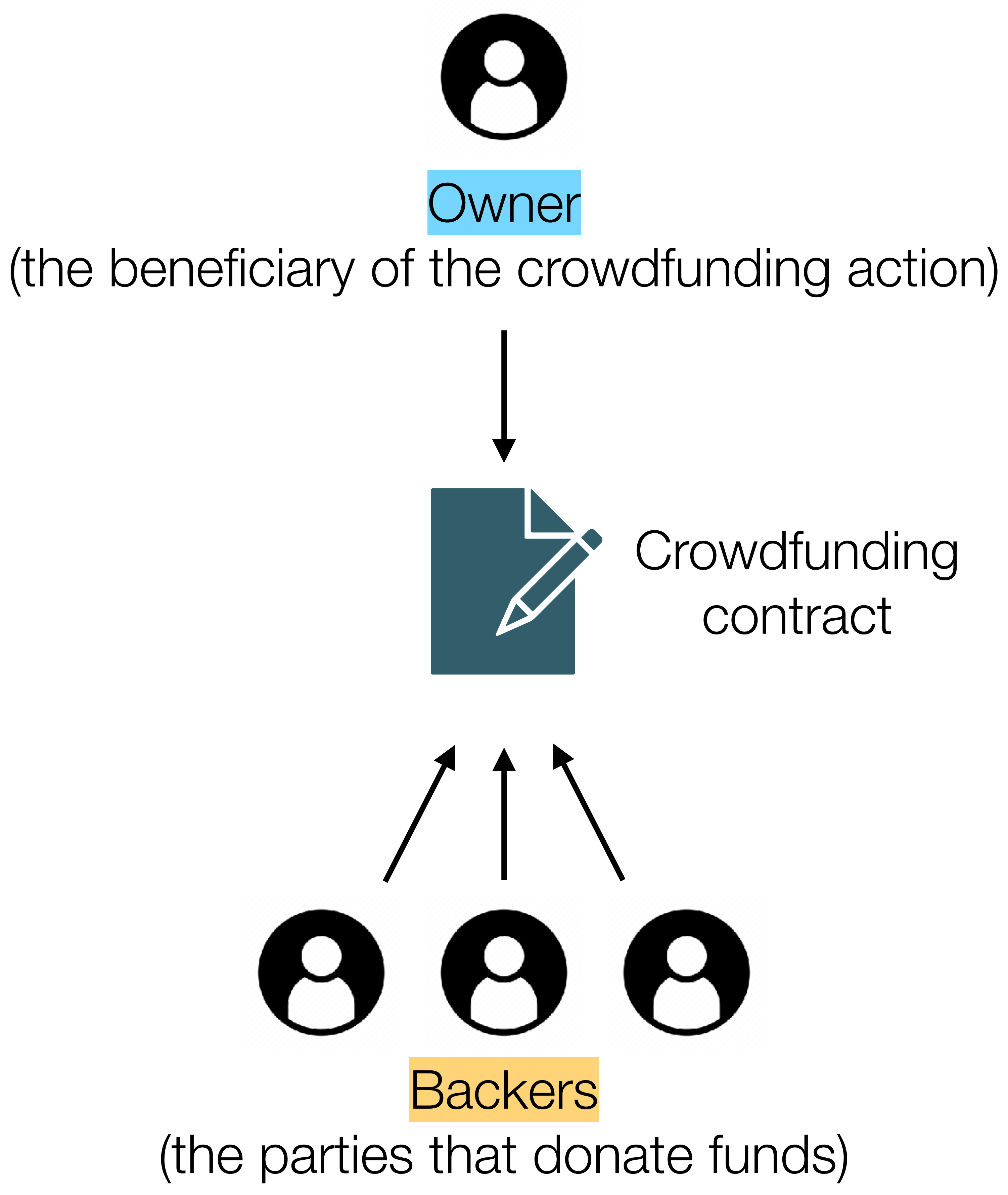
1. Backers deposit tokens (pledge support)

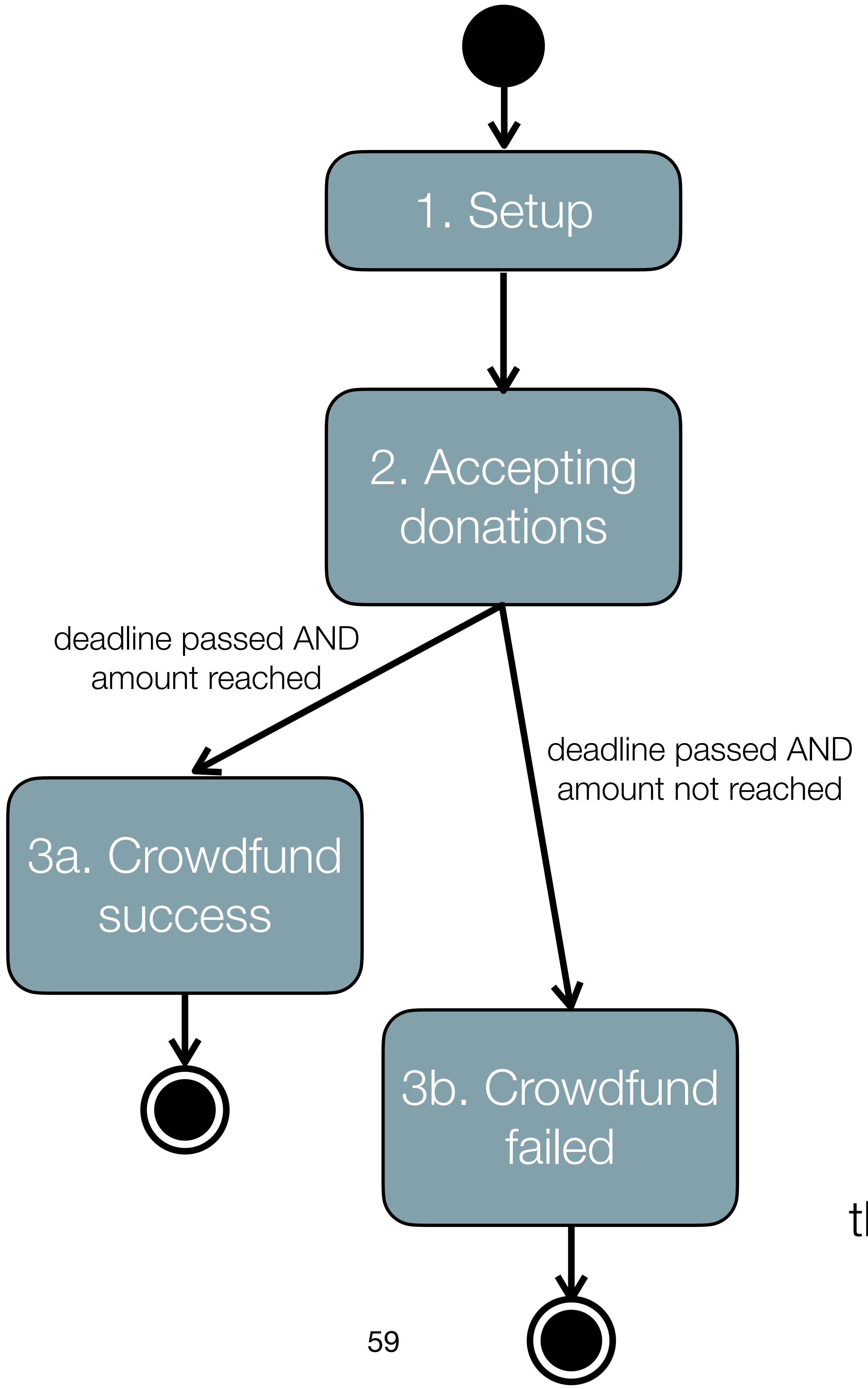
2. Wait until deadline to see if the goal was met

3a. Either the backers withdraw their share…

3b. or the owner withdraws the full deposit

Example: crowdfunding as a smart contract





Step 1: the owner creates the contract, stating target amount + funding deadline (which **cannot be changed** afterwards)

Step 2: backers can donate money (**deposit** funds into the contract) IF the funding deadline has not yet passed

Step 3a (crowdfunding successful): the owner can claim the funds (**withdraw** funds from the contract) IF the funding deadline has passed AND the minimum target amount has been met

Step 3b (crowdfunding failed): backers can reclaim their donations (**withdraw** funds from the contract) IF the funding deadline has passed AND the minimum target amount has **not** been met

Crowdfunding contract: Solidity source code

```
contract Crowdfunding {

    address public owner;    // the beneficiary address
    uint256 public deadline; // campaign deadline in number of days
    uint256 public goal;     // funding goal in ether
    mapping (address => uint256) public backers; // the share of each backer

    constructor(uint256 numberOfDays, uint256 _goal) {
        owner = msg.sender;
        deadline = block.timestamp + (numberOfDays * 1 days);
        goal = _goal;

    }
    function donate() public payable {
        require(block.timestamp < deadline); // before the fundraising deadline
        backers[msg.sender] += msg.value;

    }

    function claimFunds() public {
        require(address(this).balance >= goal); // funding goal met
        require(block.timestamp >= deadline); // after the withdrawal period
        require(msg.sender == owner);
        payable(msg.sender).transfer(address(this).balance);
    }
    function getRefund() public {
        require(address(this).balance < goal); // campaign failed: goal not met
        require(block.timestamp >= deadline); // in the withdrawal period
        uint256 donation = backers[msg.sender];
        backers[msg.sender] = 0;
        payable(msg.sender).transfer(donation);
    }
}
```

(Based on: Ilya Sergey, “The next 700 smart contract languages”, Principles of Blockchain Systems 2021)

Crowdfunding contract: Solidity source code

```
contract Crowdfunding {
    address public owner;    // the beneficiary address
    uint256 public deadline; // campaign deadline in number of days
    uint256 public goal;     // funding goal in ether
    mapping (address => uint256) public backers; // the share of each backer

    constructor(uint256 numberOfDays, uint256 _goal) {
        owner = msg.sender;
        deadline = block.timestamp + (numberOfDays * 1 days);
        goal = _goal;
    }
    function donate() public payable {
        require(block.timestamp < deadline); // before the fundraising deadline
        backers[msg.sender] += msg.value;
    }

    function claimFunds() public {
        require(address(this).balance >= goal); // funding goal met
        require(block.timestamp >= deadline); // after the withdrawal period
        require(msg.sender == owner);
        payable(msg.sender).transfer(address(this).balance);
    }
    function getRefund() public {
        require(address(this).balance < goal); // campaign failed: goal not met
        require(block.timestamp >= deadline); // in the withdrawal period
        uint256 donation = backers[msg.sender];
        backers[msg.sender] = 0;
        payable(msg.sender).transfer(donation);
    }
}
```

Declare a contract.

Similar to a class in OOP, a contract can have **state** (variables) and **behaviour** (functions)

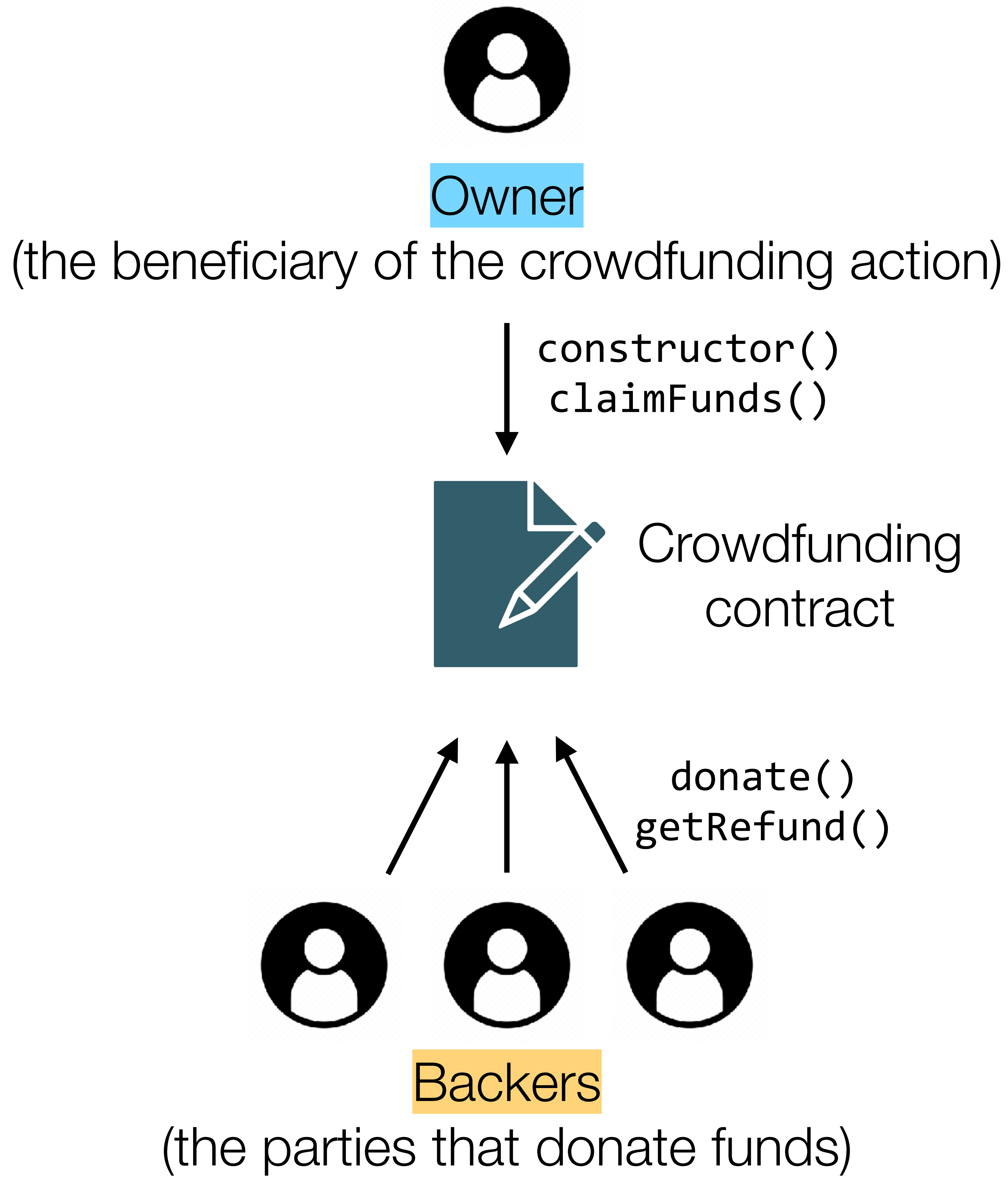
Crowdfunding contract: Solidity source code

```
contract Crowdfunding {
    address public owner;     // the beneficiary address
    uint256 public deadline; // campaign deadline in number of days
    uint256 public goal;      // funding goal in ether
    mapping (address => uint256) public backers; // the share of each backer
    constructor(uint256 numberOfDays, uint256 _goal) {
        owner = msg.sender;
        deadline = block.timestamp + (numberOfDays * 1 days);
        goal = _goal;
    }
    function donate() public payable {
        require(block.timestamp < deadline); // before the fundraising deadline
        backers[msg.sender] += msg.value;
    }
    function claimFunds() public {
        require(address(this).balance >= goal); // funding goal met
        require(block.timestamp >= deadline); // after the withdrawal period
        require(msg.sender == owner);
        payable(msg.sender).transfer(address(this).balance);
    }
    function getRefund() public {
        require(address(this).balance < goal); // campaign failed: goal not met
        require(block.timestamp >= deadline); // in the withdrawal period
        uint256 donation = backers[msg.sender];
        backers[msg.sender] = 0;
        payable(msg.sender).transfer(donation);
    }
}
```

All contract state is **replicated** and publicly **persisted** on the blockchain.

Crowdfunding contract: Solidity source code

```
contract Crowdfunding {

    address public owner;    // the beneficiary address
    uint256 public deadline; // campaign deadline in number of days
    uint256 public goal;     // funding goal in ether
    mapping (address => uint256) public backers; // the share of each backer

    constructor(uint256 numberOfDays, uint256 _goal) {
        owner = msg.sender;
        deadline = block.timestamp + (numberOfDays * 1 days);
        goal = _goal;
    }
    function donate() public payable {
        require(block.timestamp < deadline); // before the fundraising deadline
        backers[msg.sender] += msg.value;
    }

    function claimFunds() public {
        require(address(this).balance >= goal); // funding goal met
        require(block.timestamp >= deadline); // after the withdrawal period
        require(msg.sender == owner);
        payable(msg.sender).transfer(address(this).balance);
    }
    function getRefund() public {
        require(address(this).balance < goal); // campaign failed: goal not met
        require(block.timestamp >= deadline); // in the withdrawal period
        uint256 donation = backers[msg.sender];
        backers[msg.sender] = 0;
        payable(msg.sender).transfer(donation);
    }
}
```



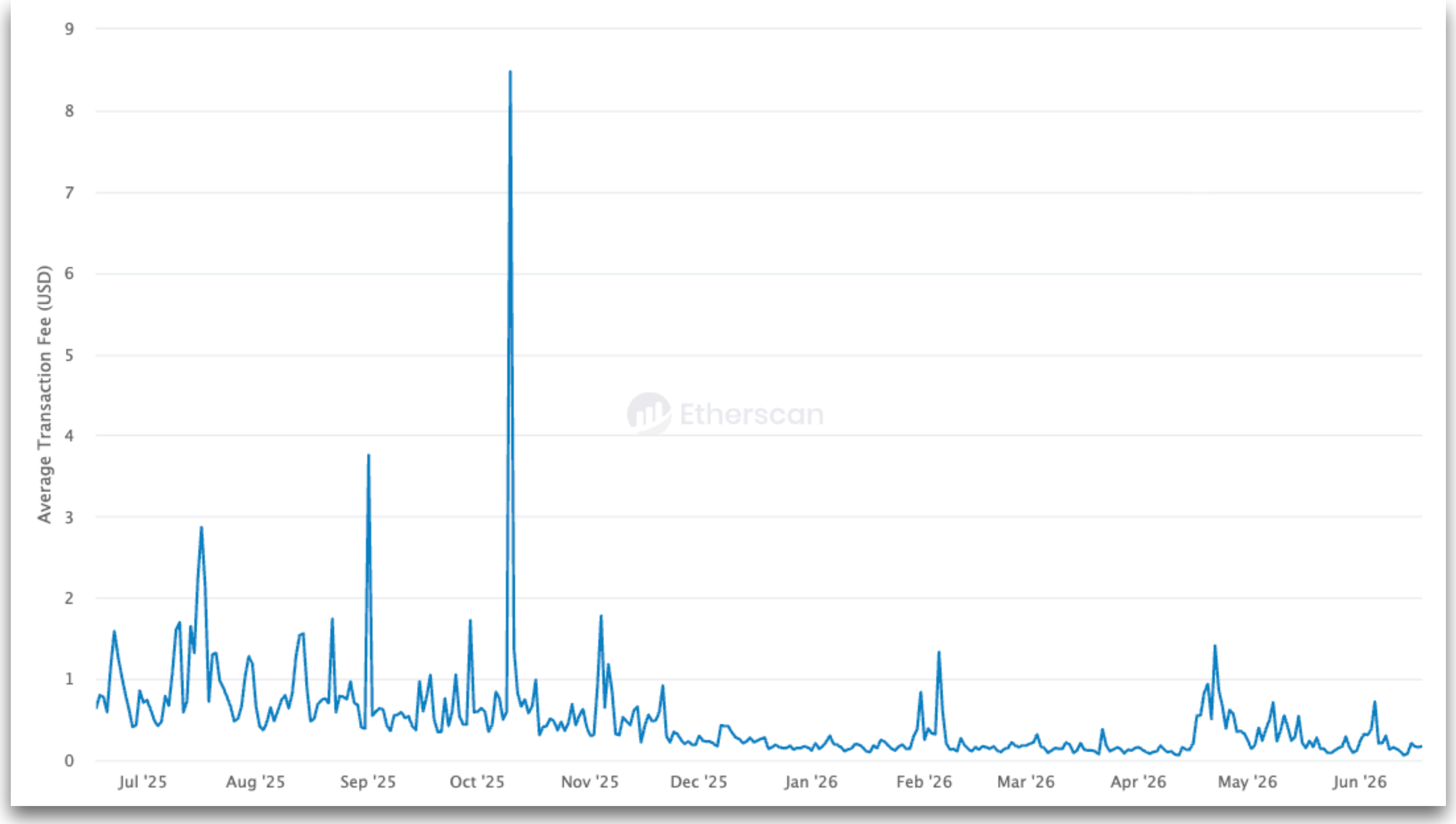
Crowdfunding contract: Solidity source code

```
contract Crowdfunding {

    address public owner;    // the beneficiary address
    uint256 public deadline; // campaign deadline in number of days
    uint256 public goal;     // funding goal in ether
    mapping (address => uint256) public backers; // the share of each backer

    constructor(uint256 numberOfDays, uint256 _goal) {
        owner = msg.sender;
        deadline = block.timestamp + (numberOfDays * 1 days);
        goal = _goal;

    }
    function donate() public payable {
        require(block.timestamp < deadline); // before the fundraising deadline
        backers[msg.sender] += msg.value;
    }

    function claimFunds() public {
        require(address(this).balance >= goal); // funding goal met
        require(block.timestamp >= deadline); // after the withdrawal period
        require(msg.sender == owner);
        payable(msg.sender).transfer(address(this).balance);
    }
    function getRefund() public {
        require(address(this).balance < goal); // campaign failed: goal not met
        require(block.timestamp >= deadline); // in the withdrawal period
        uint256 donation = backers[msg.sender];
        backers[msg.sender] = 0;
        payable(msg.sender).transfer(donation);
    }
}
```

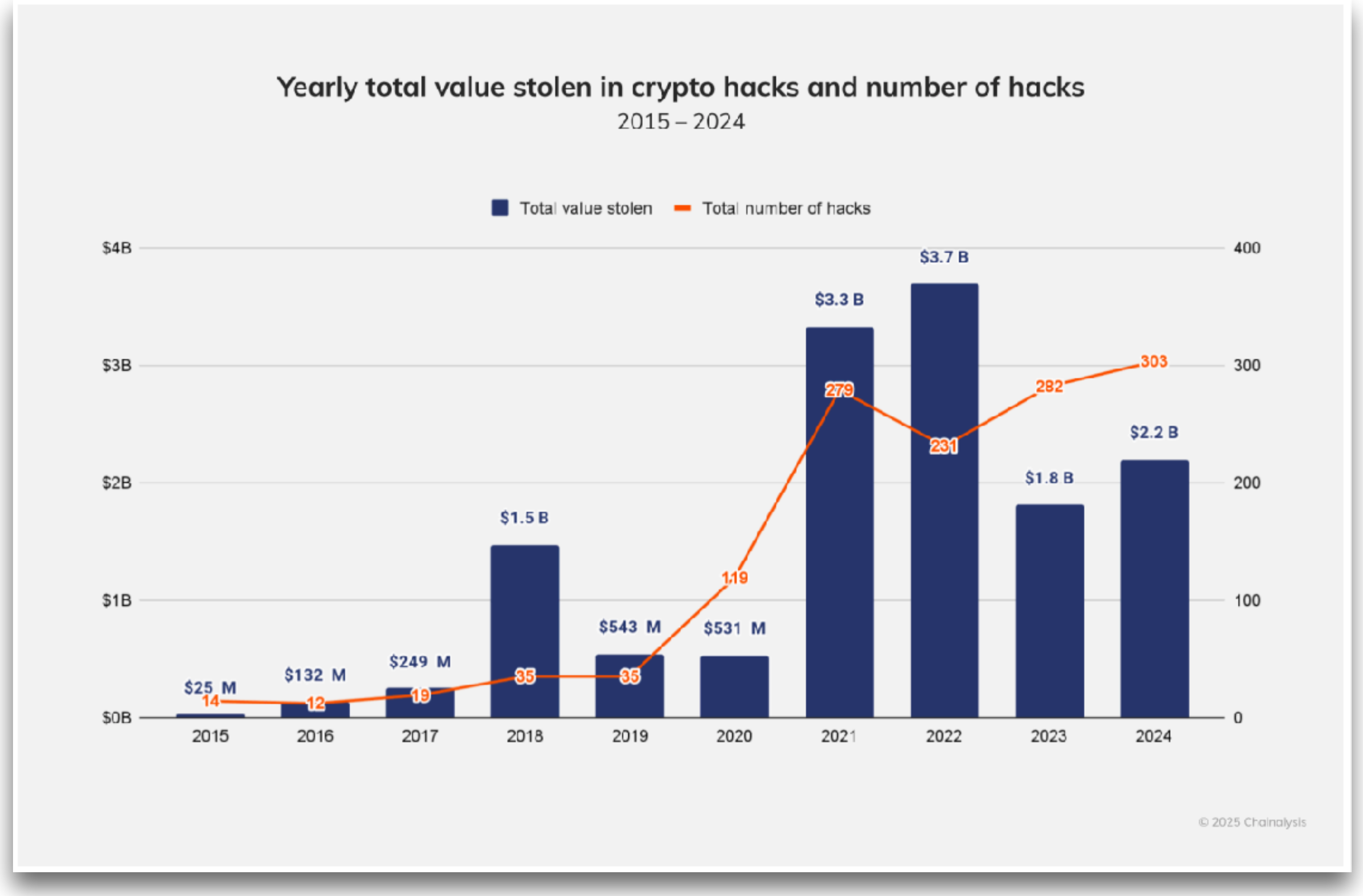
Instructions to deposit and withdraw money (ether)

Ethereum has challenges

- Slow: ~10-14 transactions per second / ~1500-2500 transactions per day
- Expensive: high transaction fees during peak times
- Bugs in smart contracts can lead to major, irreversible financial losses



(Source: Etherscan, June 2025- June 2026 data)



(Source: Chainalysis, Crypto Crime Report 2025)

“Layer 2” scaling solutions (a.k.a. “rollups”)

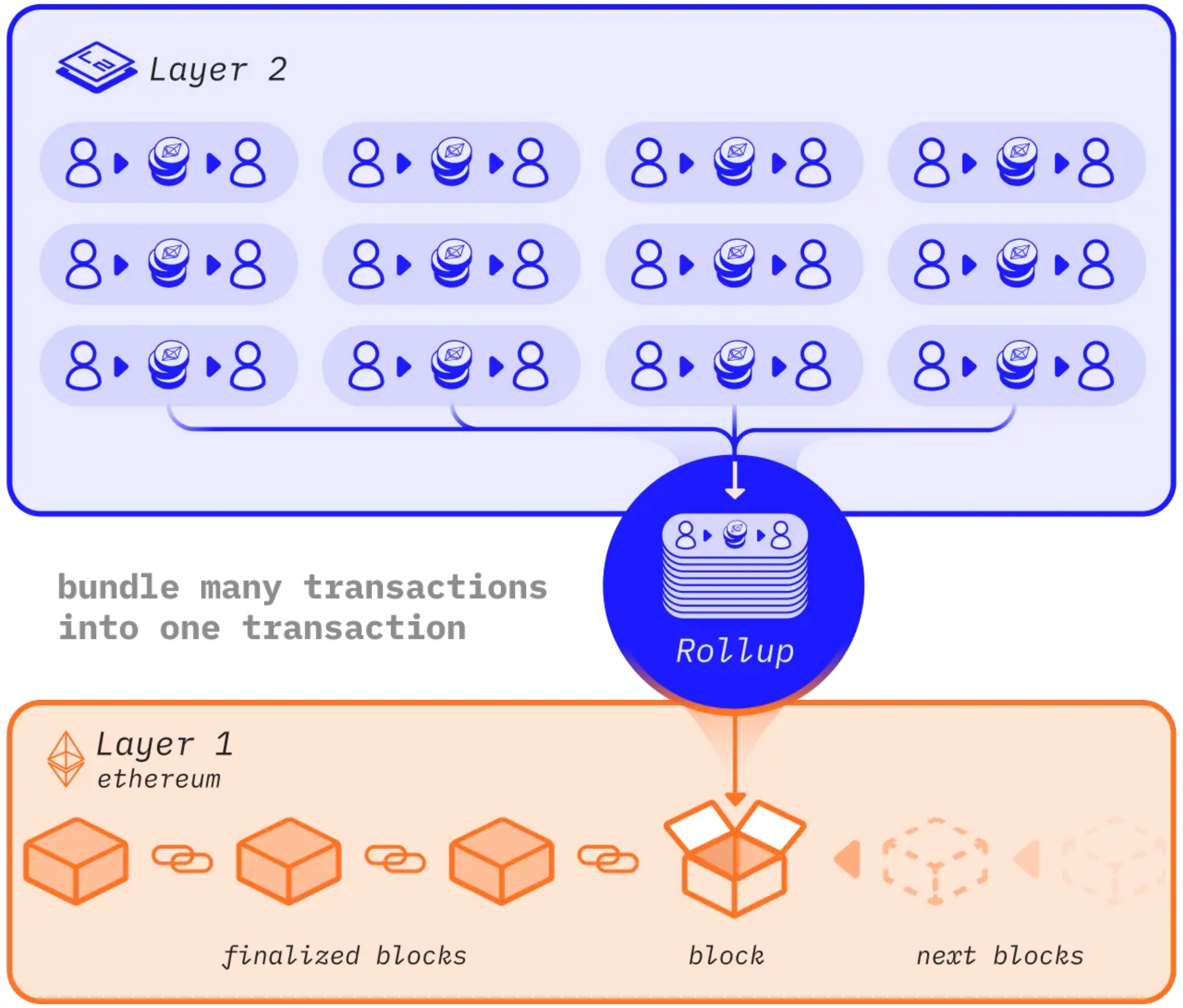
- Key idea: batch many “Layer 2” (L2) transactions into a single combined transaction stored on “Layer 1”

- Offer a way for anyone to verify that the batch of L2 transactions was correctly executed

- Two main categories, with different trust model:

  - **Optimistic rollups:** “innocent until proven guilty” — submit fraud proofs during dispute period, require long (~7 day) delays until funds can be safely withdrawn.

  - **ZK rollups:** include “cryptographic proof of correctness” (e.g. a zk-SNARK) on every batch submission. Immediate finality, but heavier cost for the rollup prover infrastructure.



(Source: https://ethereum.org/layer-2/learn)

“Layer 2” scaling solutions: benefits

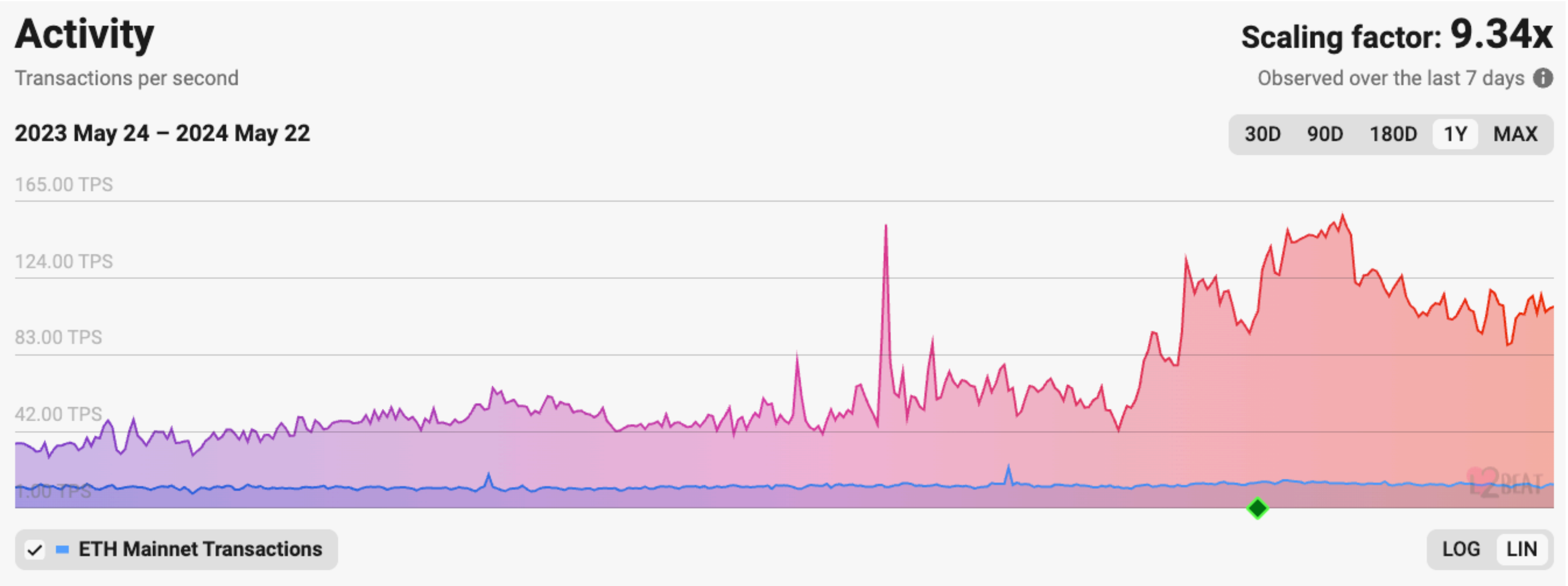
- **Lower** transaction **fees** (~ $0.10 / tx)

All L2s | Full Rollups

Name	Send ETH	Swap tokens
Metis Network ⚠	$0.04	$0.18
Loopring	$0.04	$0.59
zkSync Era	$0.07	-
zkSync Lite	$0.09	$0.22
Optimism	$0.09	$0.18
Arbitrum One	$0.09	$0.27
Boba Network	$0.15	$0.17
DeGate	$0.16	$0.18
StarkNet	$0.19	$0.57
Polygon zkEVM	$0.19	$2.75
Ethereum	$1.10	$5.48

(Source: l2fees.info, June 2026)

- **Higher** transaction **throughput** (100-1000 tps at ~13min finality)



(Source: L2Beat, 2023-2024 data)

Conclusion

Blockchain: challenges beyond the basics

Problem	Solution	Examples
Lack of Privacy: can't store secrets on a blockchain	Zero-knowledge Proofs	ZCash, Privacy Pools, StarkNet
Poor scalability: "secure, scalable, decentralized: choose two"	Layer-2 rollups, AppChains, Payment channels	Arbitrum, Optimism, Lightning Network
Security vulnerabilities in smart contracts	Use safer languages and abstractions	Move programming language (used in Sui blockchain)
(Mobile/Web) clients must **trust servers** to access the blockchain	Stateless "light clients", compact inclusion proofs, decentralized RPC protocols, …	Helios, Mina protocol, Portal network, PARP Protocol
Interoperability: assets stored on one blockchain cannot be used on another	Cross-chain "bridges", atomic swaps (hashed time-locked contracts), …	Inter-blockchain Communication (IBC) protocol, Wormhole
How can contracts get trustworthy, reliable access to real-world **off-chain** data?	"Oracle" contracts, Decentralized Oracle Networks (DONs)	Chainlink
ECDSA is vulnerable to attacks from **Quantum** Computers via Shor's algorithm	Transition to Post-Quantum signature schemes	Hash-Based Post-Quantum Signatures, EdDSA + ZK 'proof of seed'

Recap: course topics

- The **origins** of Blockchain
- What are the **cryptographic building blocks** of a blockchain?
- How does a blockchain process transactions? **Life of a blockchain transaction**.
- **Consensus** in blockchain networks: Proof-of-Work, Proof-of-Stake, BFT Consensus
- **Permissioned** versus **Permissionless** blockchain networks
- Bonus: Blockchains as trusted computers: **smart contracts** and **Ethereum**
- **Conclusion** and latest trends

Blockchain and Distributed Ledgers

Tom Van Cutsem
KU Leuven, DistriNet

Questions?
tom.vancutsem@kuleuven.be

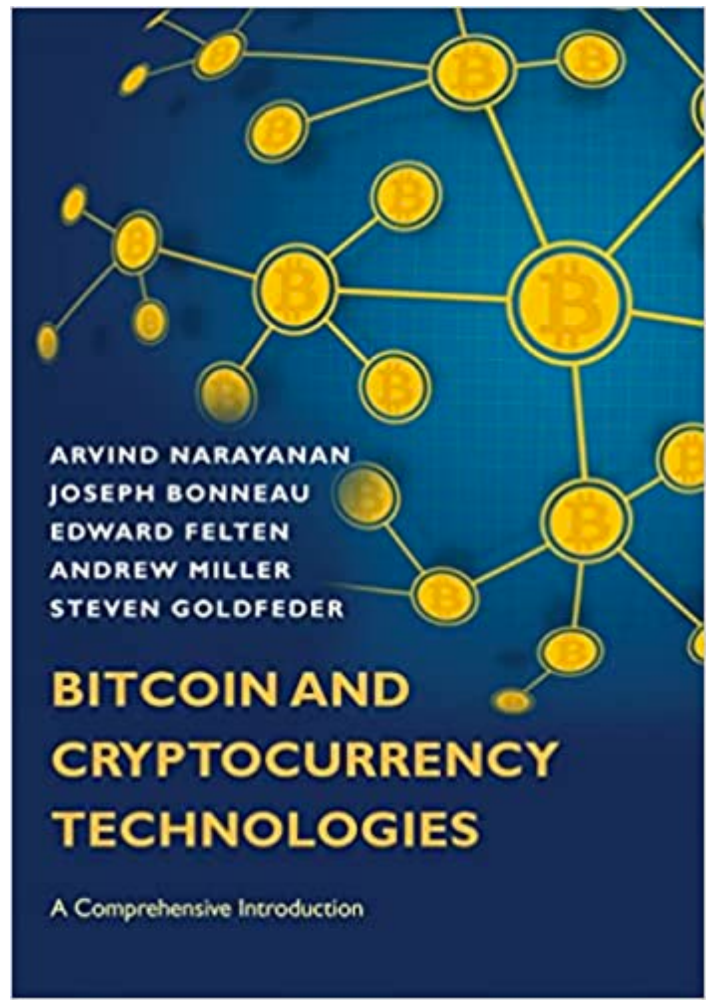
Further reading - good introductory resources on Blockchain

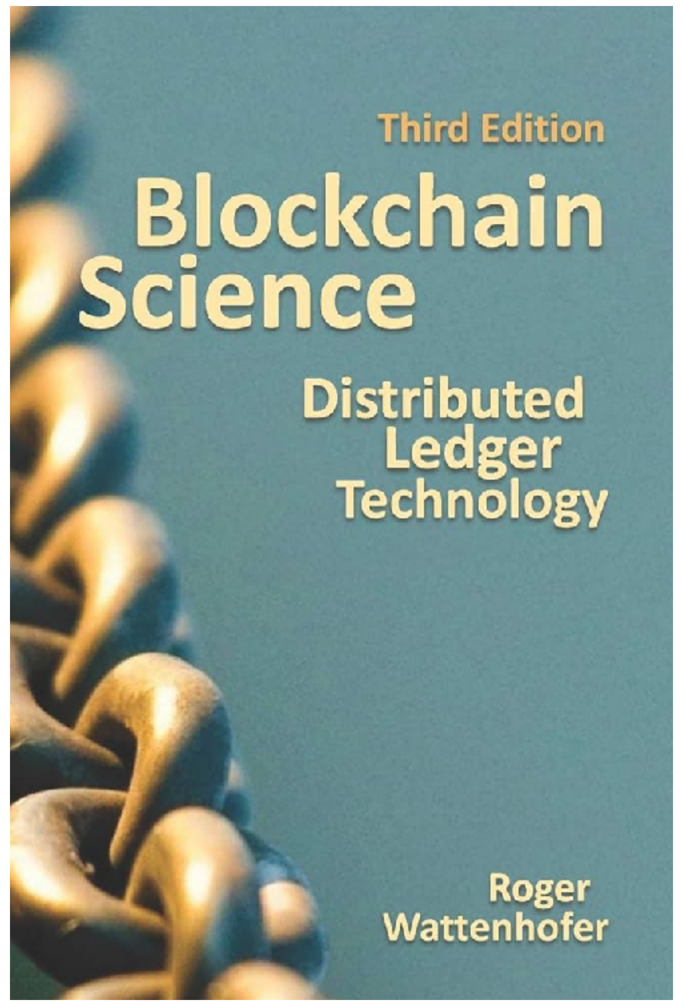
- Narayanan *et al.* “Bitcoin and Cryptocurrency Technologies” Princeton University Press, 2016 - preprint available for free online at: https://bitcoinbook.cs.princeton.edu/

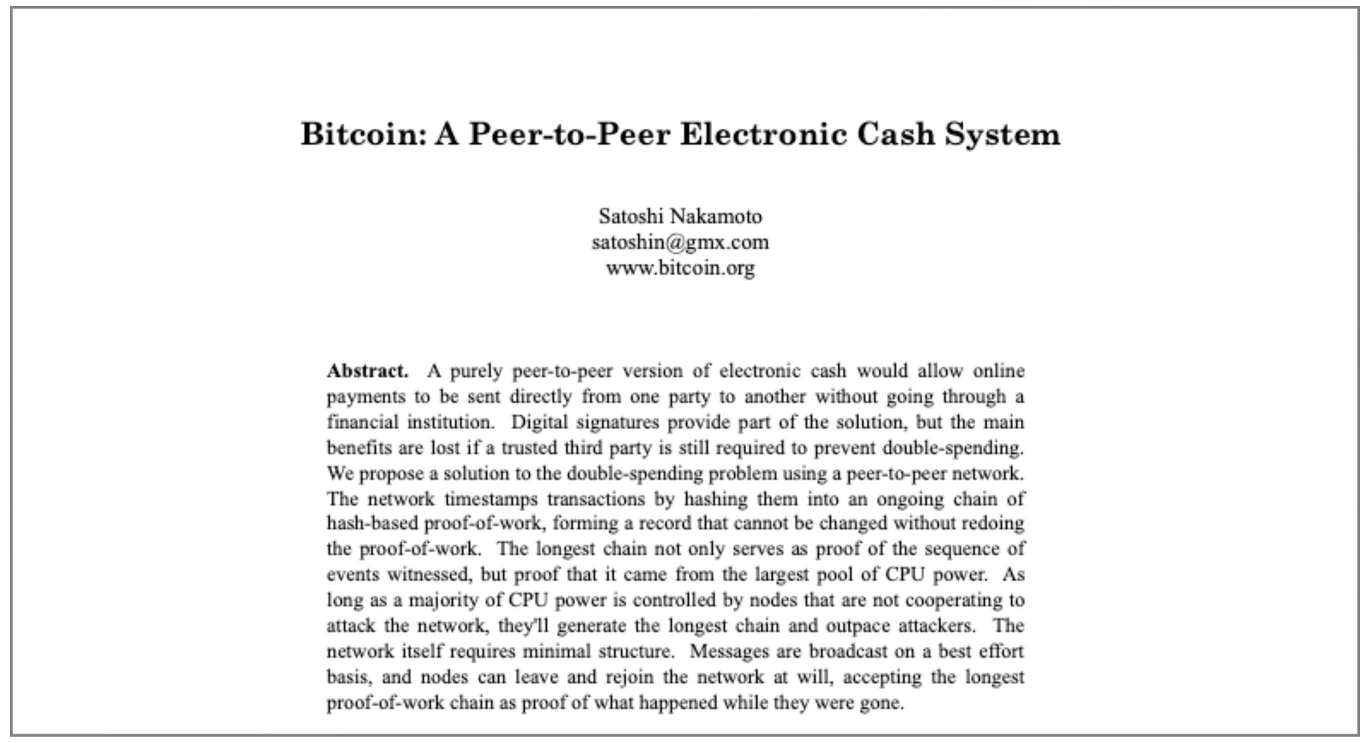
- Roger Wattenhofer (ETH Zurich), “Blockchain Science”, 2019

- Satoshi Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System” a.k.a. the “Bitcoin whitepaper” (2008)

    - Recommended: an annotated online version with helpful notes & clarifications (D. Hogg, 2021): https://blog.infocruncher.com/2021/10/31/bitcoin-whitepaper-annotated/







Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Further reading - good introductory resources on Ethereum

- Ethereum official project website: https://ethereum.org/
- Ethereum whitepaper: https://ethereum.org/en/whitepaper/
- Etherscan block explorer: https://etherscan.io/
- Remix, an online IDE and playground for Solidity: https://remix.ethereum.org/
- Solidity by Example: https://solidity-by-example.org/
- OpenZeppelin reusable contracts: https://www.openzeppelin.com/contracts
- Awesome-Ethereum: https://github.com/ttumiel/Awesome-Ethereum

Work Projects

Work projects

- Project 1: **build** an end-to-end **decentralized application** on Ethereum
- Project 2: perform a **comparative study of consensus protocols** for blockchains

Project 1: build a decentralized application on Ethereum

- Pick your own application use case, or elaborate on the Crowdfunding example
- Components to build:
  - **Front-end** (website UI, wallet integration)
  - **Back-end** (web-server, gateway to blockchain)
  - **Smart contract** (Solidity program, deployed on a test-network)
- Things to consider:
  - **Development**: use a developer framework like Hardhat
  - **Testing**: write unit tests to test the important interactions
  - **Static analysis**: use a static analysis tool for Solidity to find bugs
  - **Stretch goal:** look into **formal verification** of key safety / liveness properties

Project 2: comparative study of consensus protocols for blockchains

- Compare the consensus protocols of 5 real-world blockchains (2 classic, 3 modern):
  - **Bitcoin**'s original Proof-of-Work based protocol
  - **Ethereum**'s modern Proof-of-Stake based protocol
  - **Solana**'s Proof-of-History based protocol
  - **Internet Computer**'s ICC consensus protocol
  - **Sui**'s Mysticeti protocol
- Focus on:
  - **History**, origins
  - **Performance** (throughput, latency)
  - **Scalability** in terms of network size
  - **Security** thresholds (safety, liveness)