Generic verifiable computation from QAP

s

(p, g, G, e, G_T)

server

Public key:
$g^{v_i(s)}, g^{w_i(s)}, g^{y_i(s)}$
for $i = 1, \dots, m$

$g^s, g^{s^2}, \dots, g^{s^m}$

client

$$\pi = g^{\sum_{i=1}^{m} c_i \times v_i(s)}, g^{\sum_{i=1}^{m} c_i \times w_i(s)}, g^{\sum_{i=1}^{m} c_i \times y_i(s)}, g^{h(s)}$$

$g^{t(s)}$

Verification: $e(\pi_1, \pi_2)/e(\pi_3, g) = e(g^{t(s)}, \pi_4)$

$$p(x) = \left(\sum_{i=1}^{m} c_i \times v_i(x)\right) \times \left(\sum_{i=1}^{m} c_i \times w_i(x)\right) - \left(\sum_{i=1}^{m} c_i \times y_i(x)\right)$$

Target polynomial: $t(x) = (x - r_1)(x - r_2)(x - r_3)$

Complexity

- Setup: O(C)
- Local storage: O(1)
- Prover: O(C log C)
- Proof size: O(1)
- Verification: O(1)+ size of IO

Summary of zkSNARK

- Circuit evaluation to satisfying assignment
- SAT to QAP
- QAP to argument (bilinear map, knowledge of exponent assumption)
- Argument to zero knowledge

Pros and Cons of zkSNARK

- ✔ Supports all functions (modeled as arithmetic circuit)
- ✔ Constant proof size
- ✔ Fast verification time

- × Function dependent **trusted setup**

Generic verifiable computation from QAP

s

(p, g, G, e, G_T)

server

Public key:
$g^{v_i(s)}, g^{w_i(s)}, g^{y_i(s)}$
for $i = 1, \dots, m$

$g^s, g^{s^2}, \dots, g^{s^m}$

client

$$\pi = g^{\sum_{i=1}^{m} c_i \times v_i(s)}, g^{\sum_{i=1}^{m} c_i \times w_i(s)}, g^{\sum_{i=1}^{m} c_i \times y_i(s)}, g^{h(s)}$$

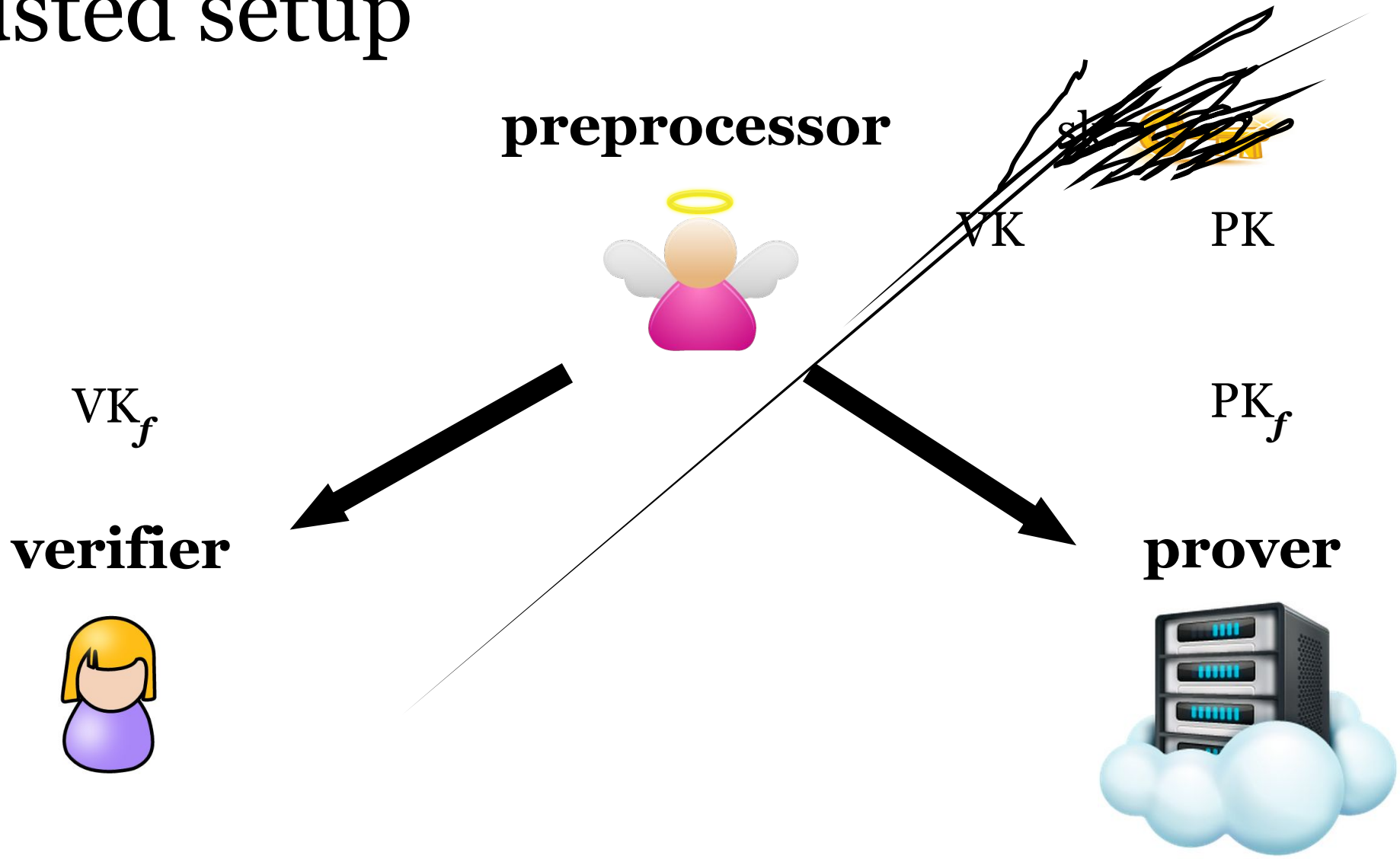
$g^{t(s)}$

Verification: $e(\pi_1, \pi_2)/e(\pi_3, g) = e(g^{t(s)}, \pi_4)$

$$p(x) = \left(\sum_{i=1}^{m} c_i \times v_i(x)\right) \times \left(\sum_{i=1}^{m} c_i \times w_i(x)\right) - \left(\sum_{i=1}^{m} c_i \times y_i(x)\right)$$

Target polynomial: $t(x) = (x - r_1)(x - r_2)(x - r_3)$

Trusted setup



Pros and Cons of zkSNARK

- ✔ Supports all functions (modeled as arithmetic circuit)
- ✔ Constant proof size
- ✔ Fast verification time

- × Function dependent **trusted setup**
- × Slow prover time (modular exponentiations for every gate)

Example: Matrix Multiplication

n = 256

Local computation time: 52ms

- Prover time: 1545s
- Memory usage: 32GB
- Proof size: 200 bytes
- Verification time: 3ms

Blockchain and Crypto-currencies

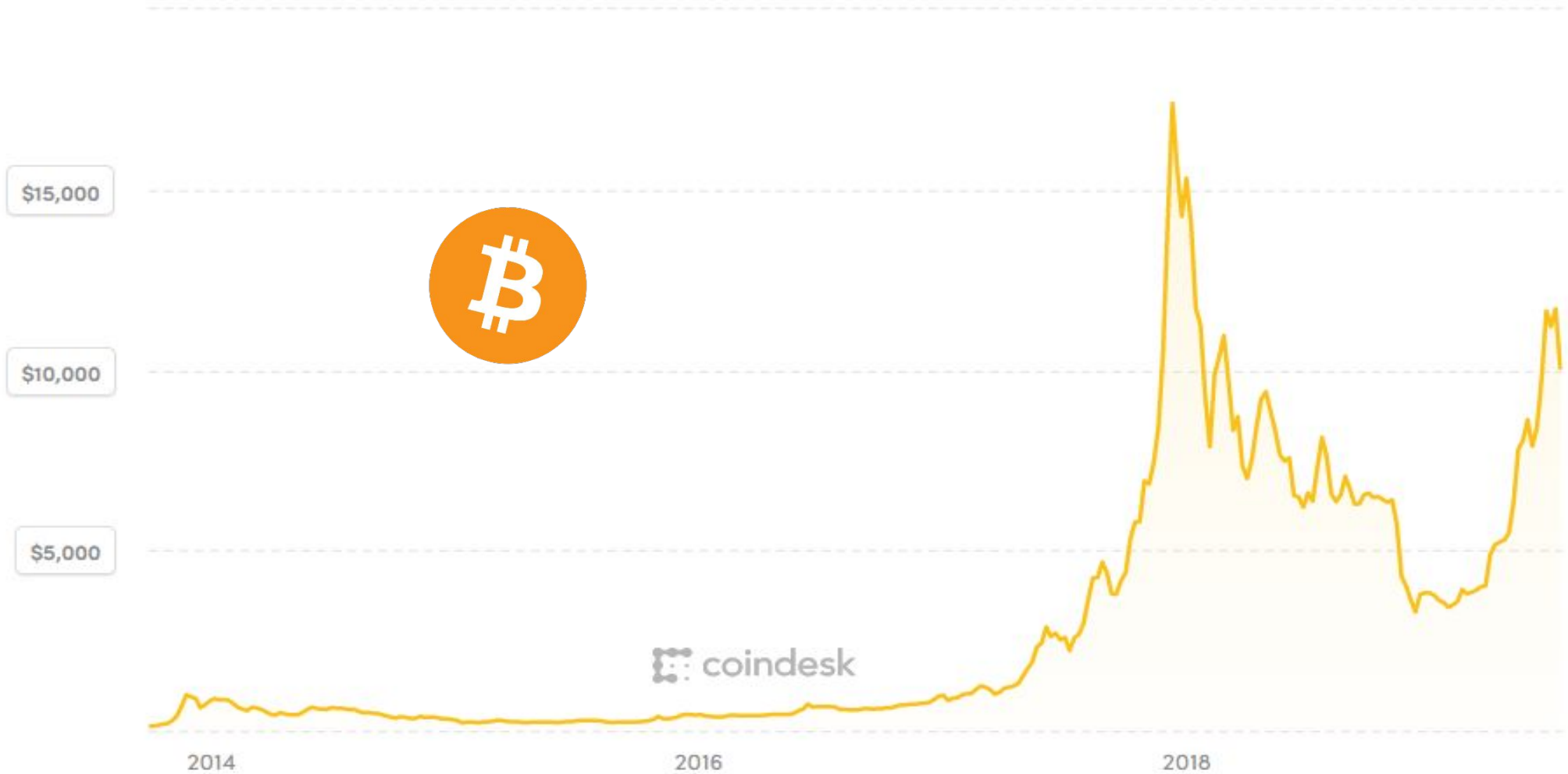
The Times 03/Jan/2009 *Chancellor on brink of second bailout for banks.*

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

```
bitcoin-0.1.0.rar
bitcoin-0.1.0.tgz
```


$15,000
$10,000
$5,000
coindesk
2014
2016
2018

Crypto-currencies

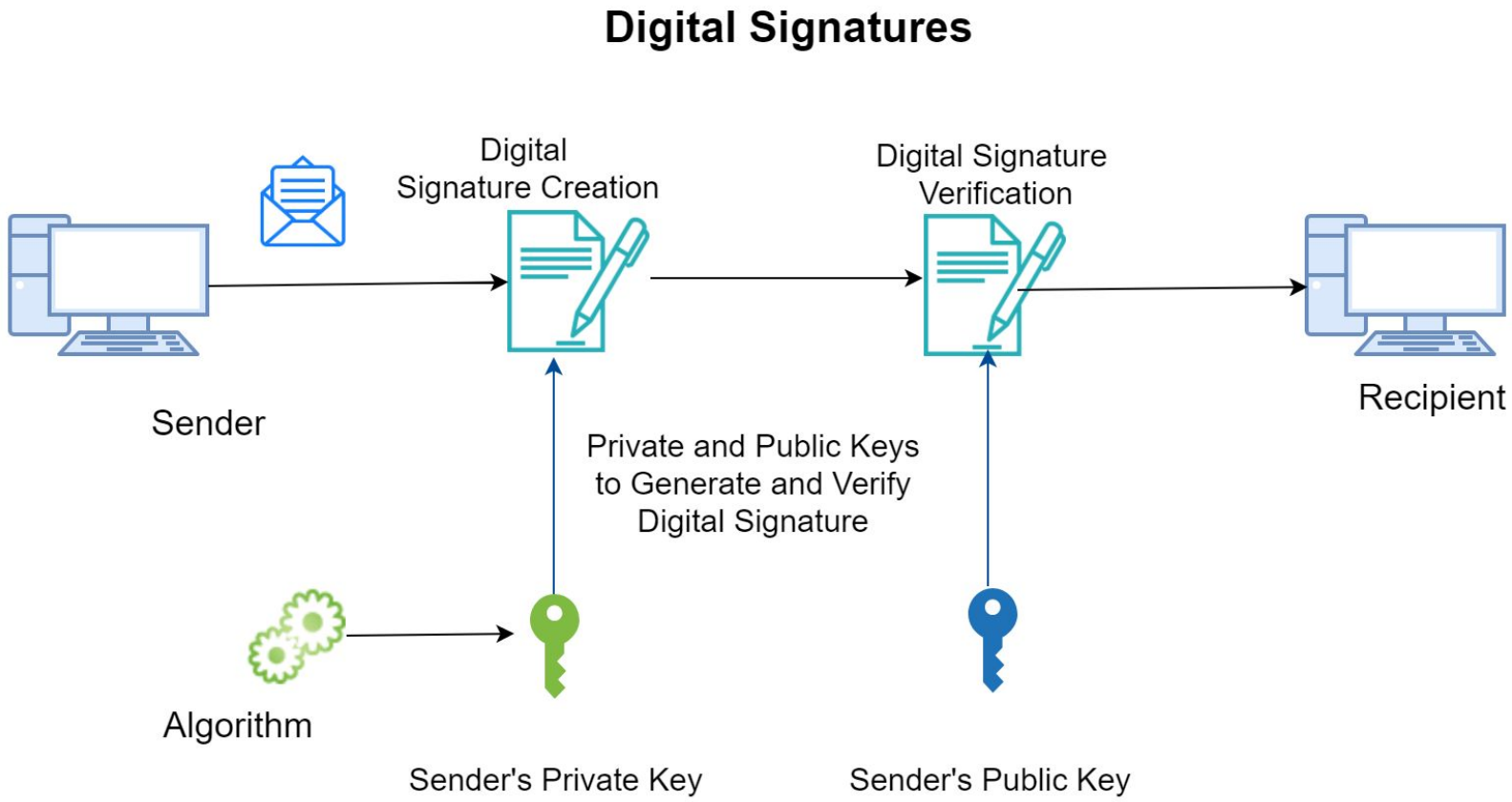
- Digital currency
- Decentralized

Challenges: authentication and double spending

vs. cash and centralized banks

Digital signature

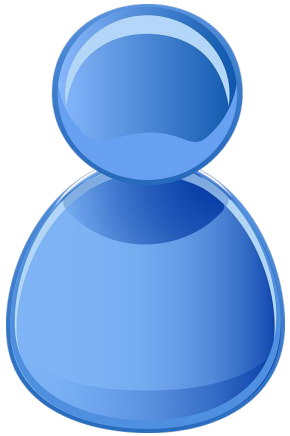
- sk, pk
- σ=Sign(sk, m)
- {0,1} = Verify(pk, m, σ)



Alice and Bob are only identified by public keys

pk_A pk_B

Transfer 10 Bitcoins from me to Bob.



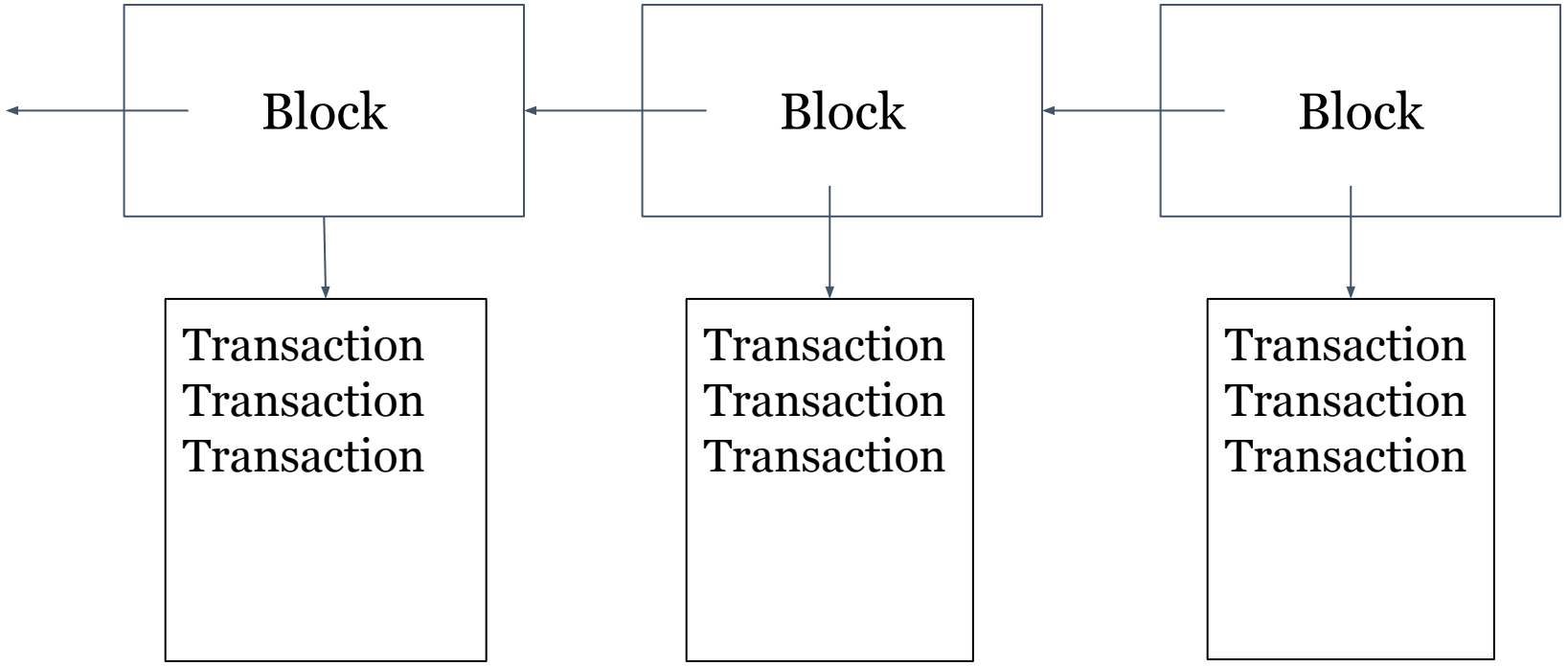
Alice

sk_A

Signed with Alice's private key

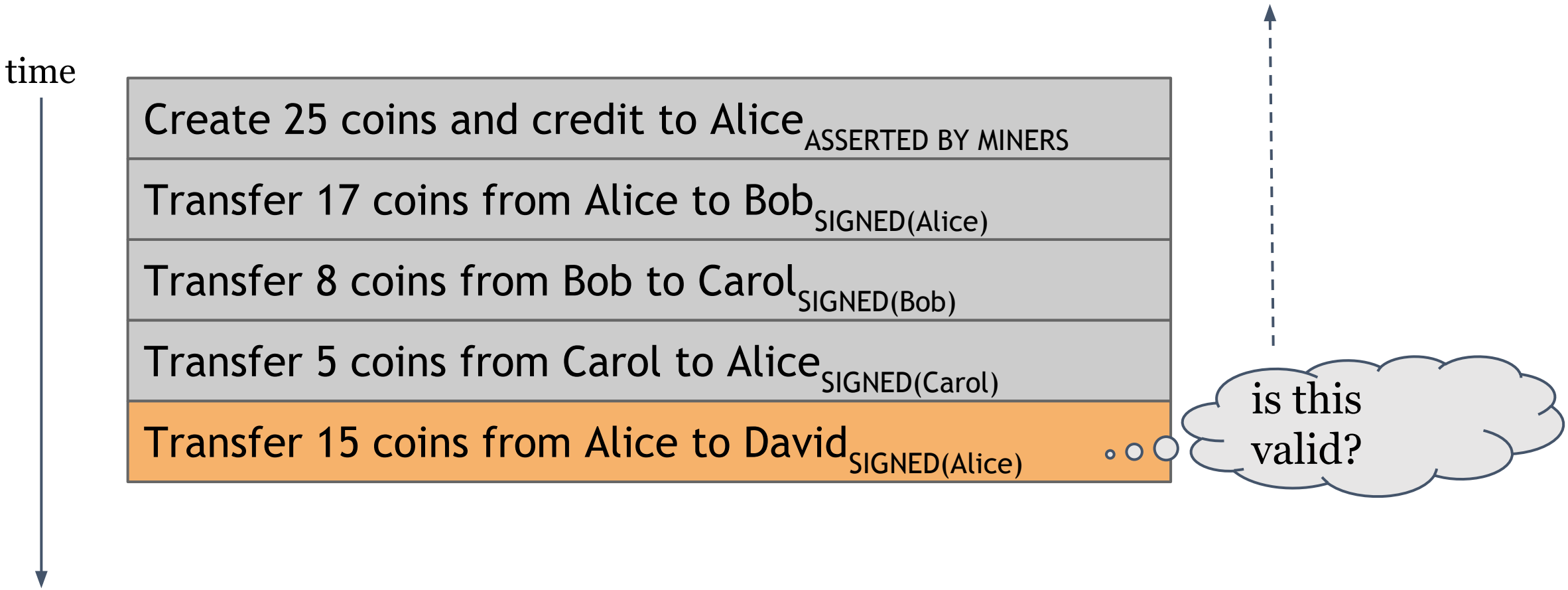
Bitcoin Network

Blockchain



Public link list / ledger / data structure

An account-based ledger (*not* Bitcoin)



SIMPLIFICATION: only one transaction per block

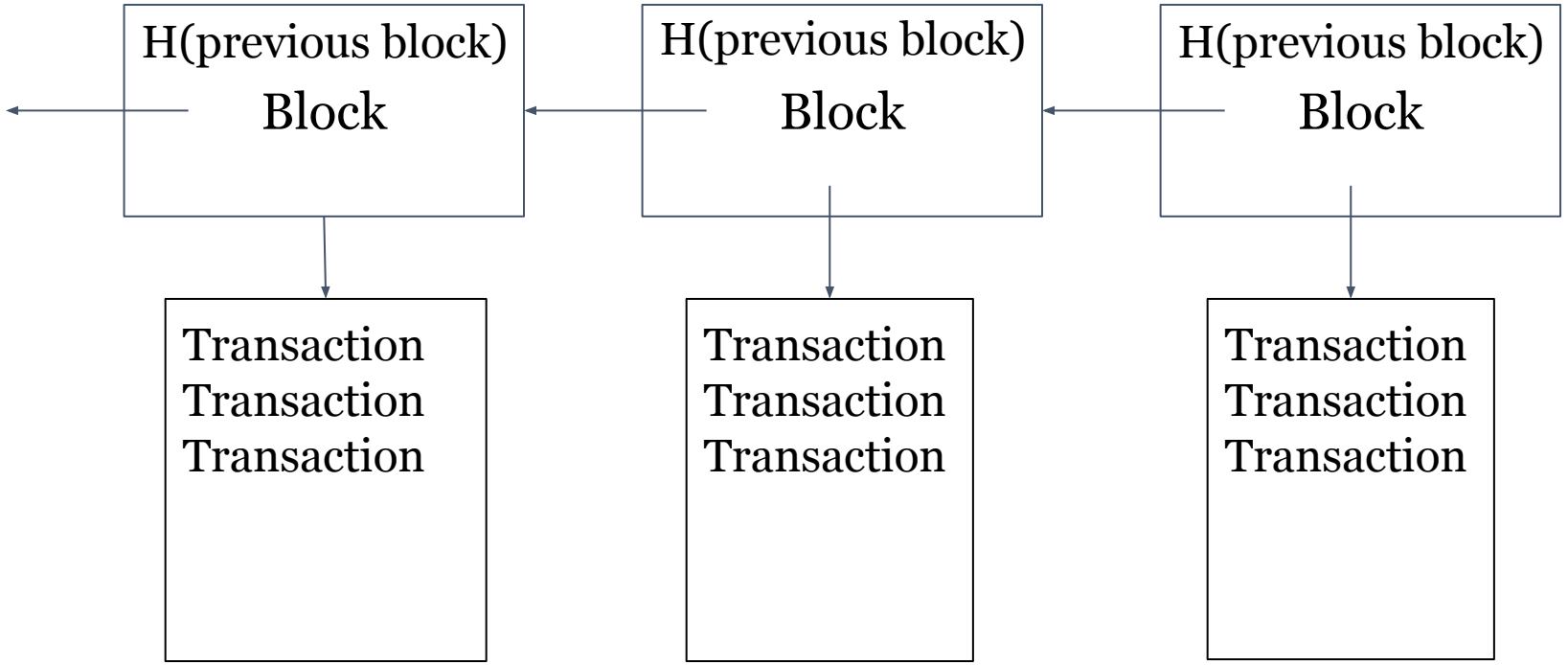
Cryptographic hash function

- $H: \{0,1\}^* \rightarrow \{0,1\}^k$ any string to 256-bit string, deterministic

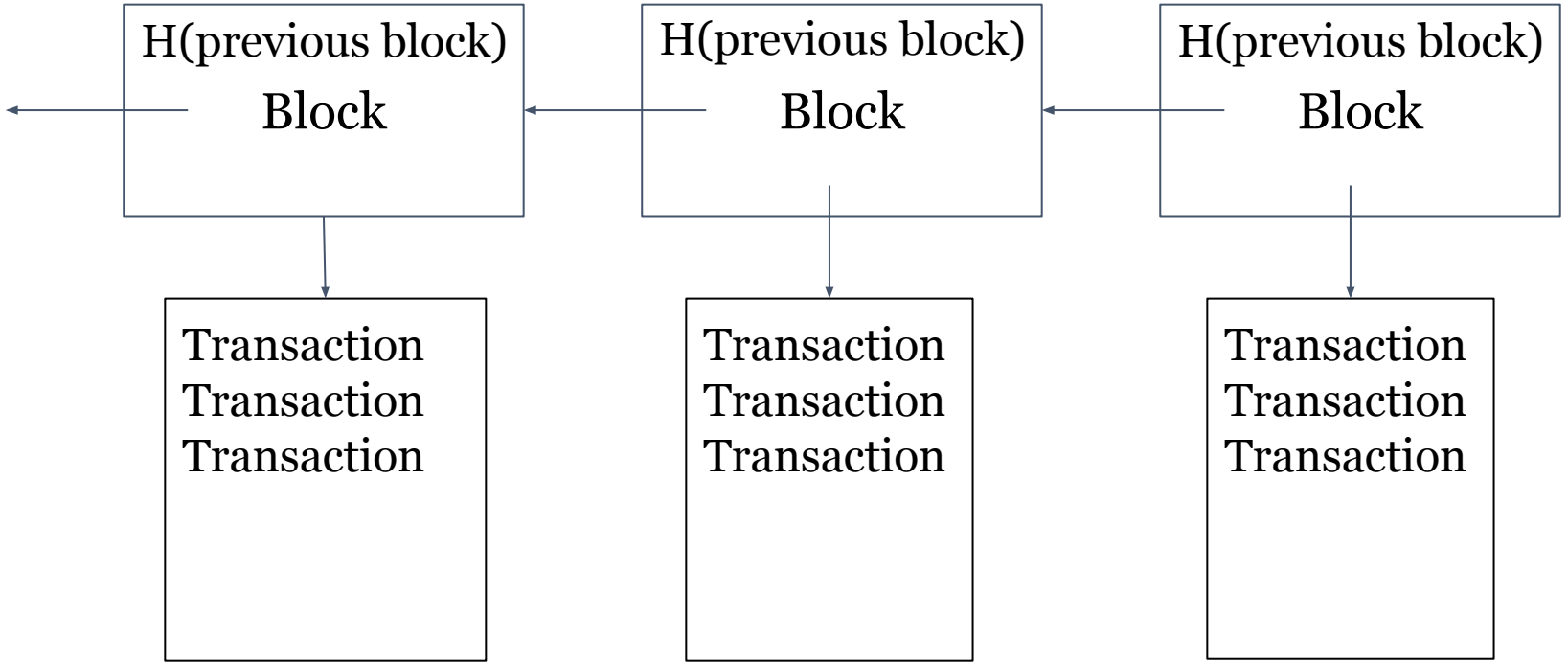
- Collision resistant: hard to find x,y such that H(x) = H(y)

- One-way: easy to compute, hard to invert
  (find x such that H(x) = y)

Hash pointer



Simple digital currency



Transfer 15 coins from Alice to David$_{\text{SIGNED(Alice)}}$

- Maintained by a trusted party

Problem: single point of failure

- Inconsistent ledger
- Unfair transactions/ denial of service

Distributed consensus

Solution

- A random party is selected to write the next block

- Everyone checks the data of the new block is valid

More than 50% honest parties → consensus without a centralized trusted party

Mining

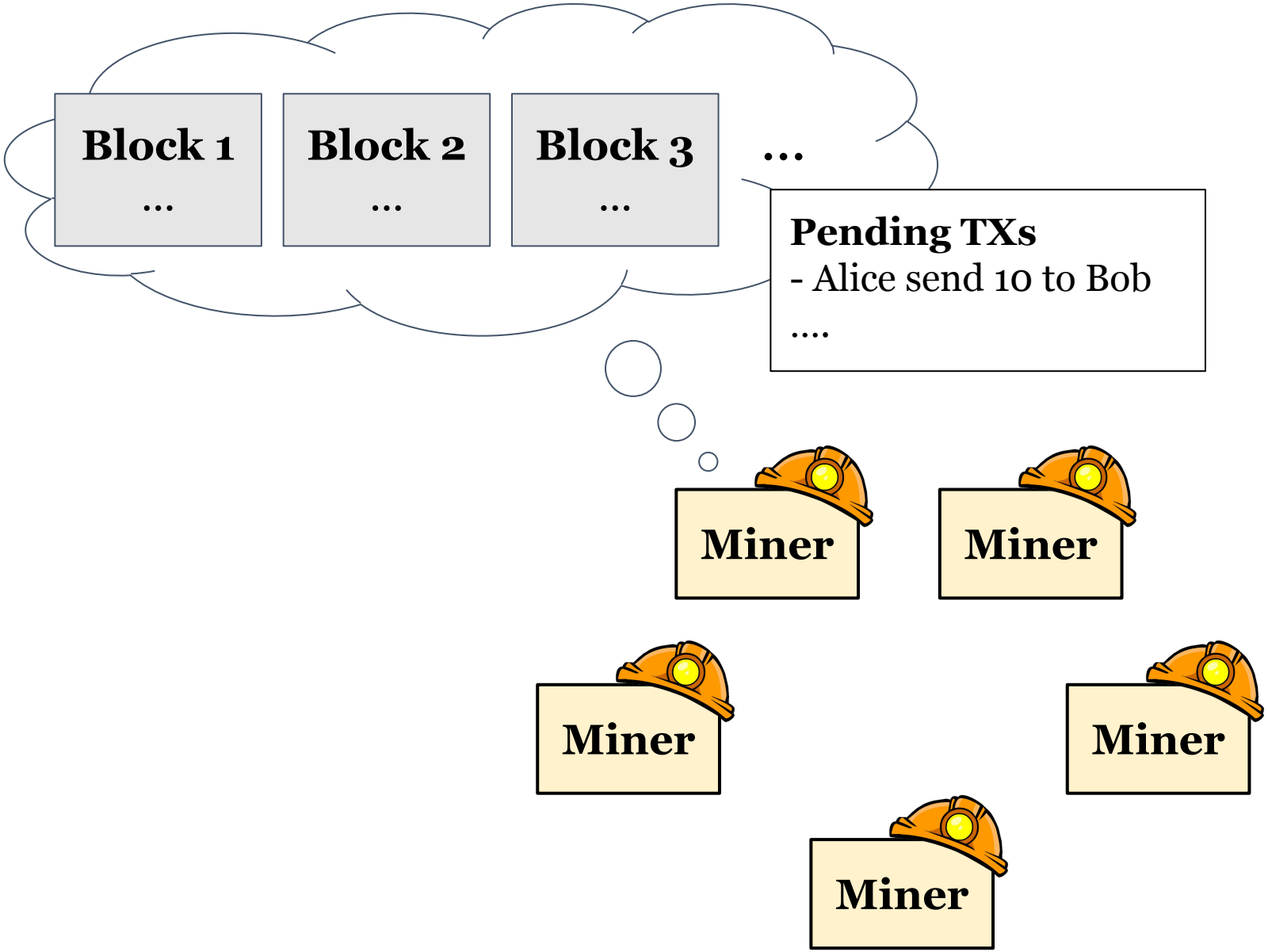
Use hash function: Find x such that $H(x) = y$

- One-way: easy to compute, hard to invert (find x such that H(x) = y)

- $H(x)$ = 000000000000... 2^{128} time

- $H(x)$ = 000000xxxxxx... tunable probability

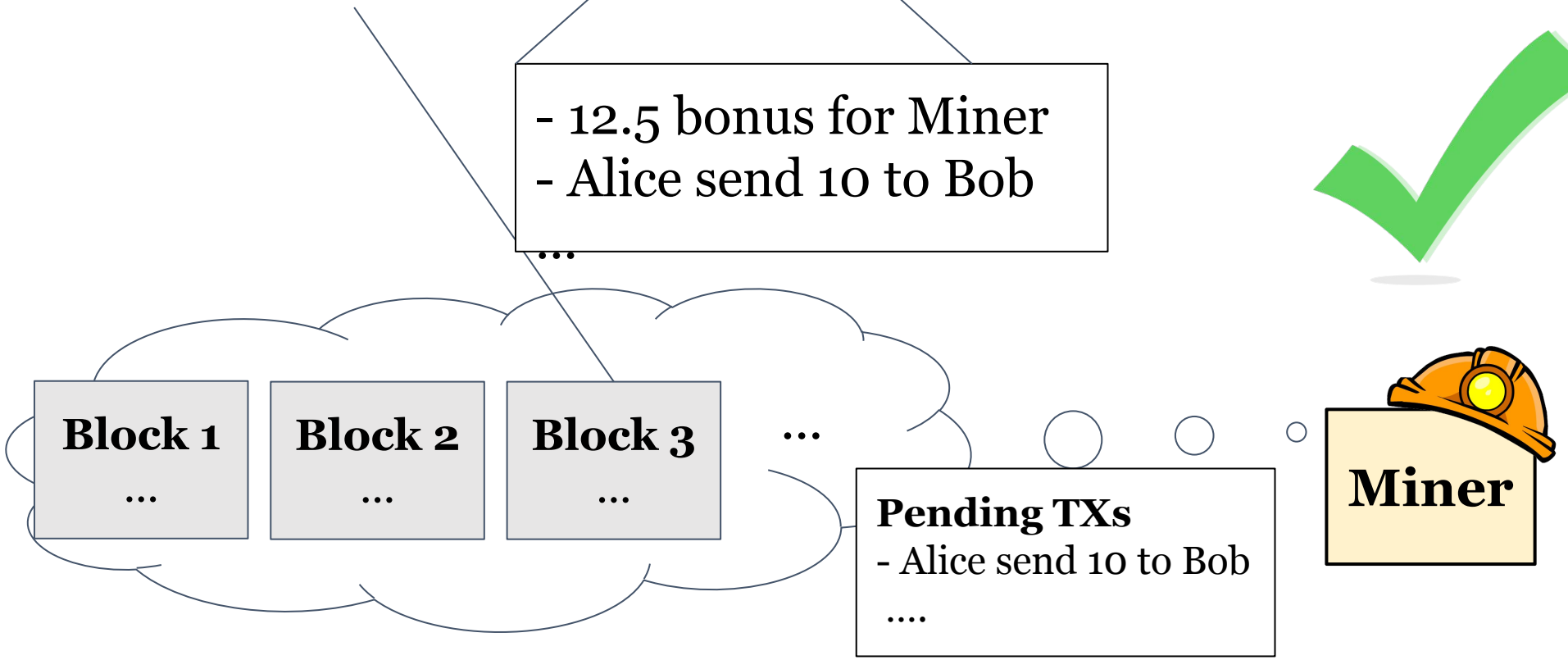
- H(transactions | nonce) = 000000xxxxxx...

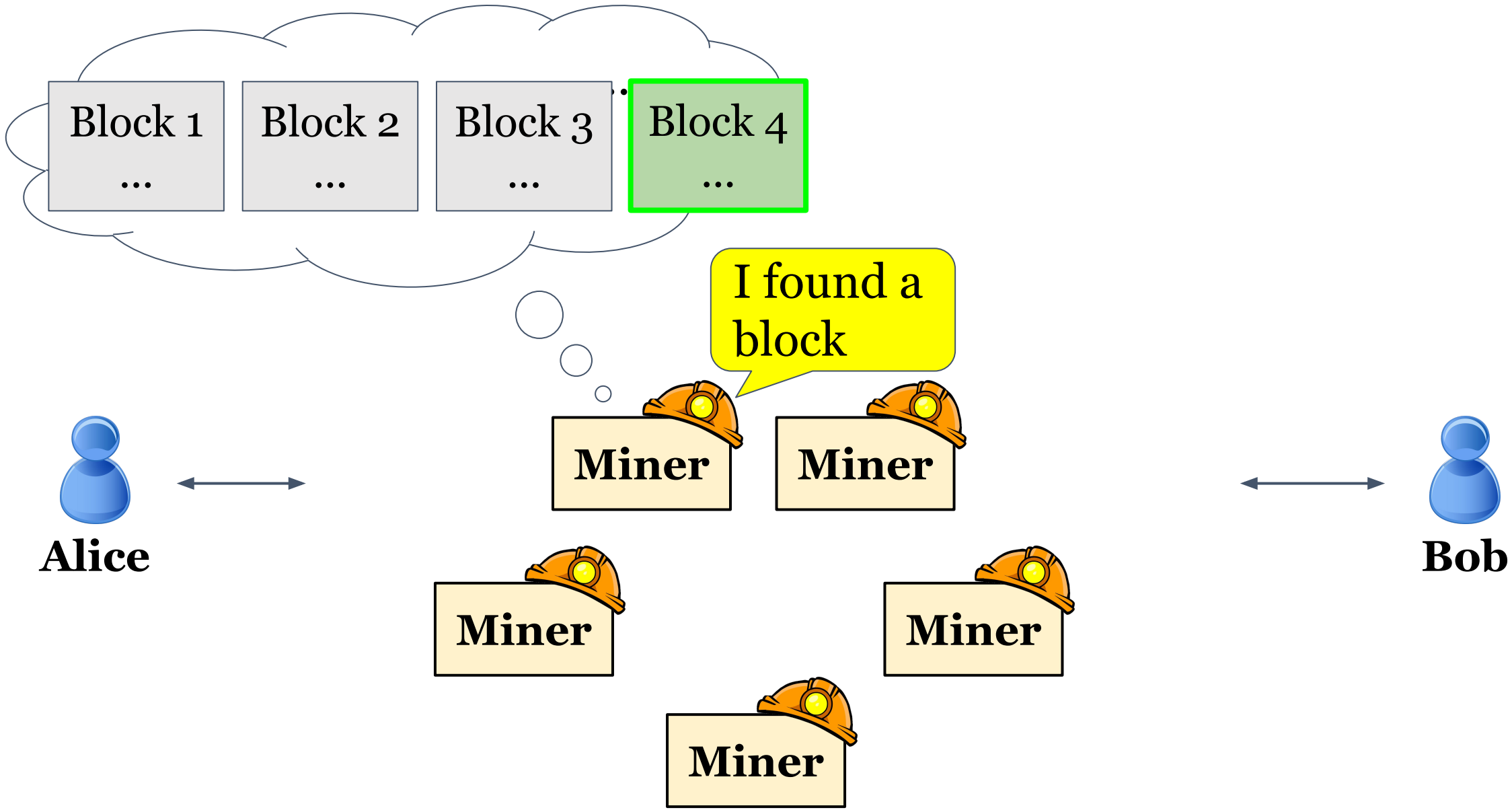

Block 1
...
Block 2
...
Block 3
...
...
Pending TXs
- Alice send 10 to Bob
....
Miner
Miner
Miner
Miner
Miner

Miners commit new transactions by solving puzzles

```
                                                    = 0x000***...

Hash(   Block 3   |  newTXs  | 0xb9824 ) = 0x000c3f...
```

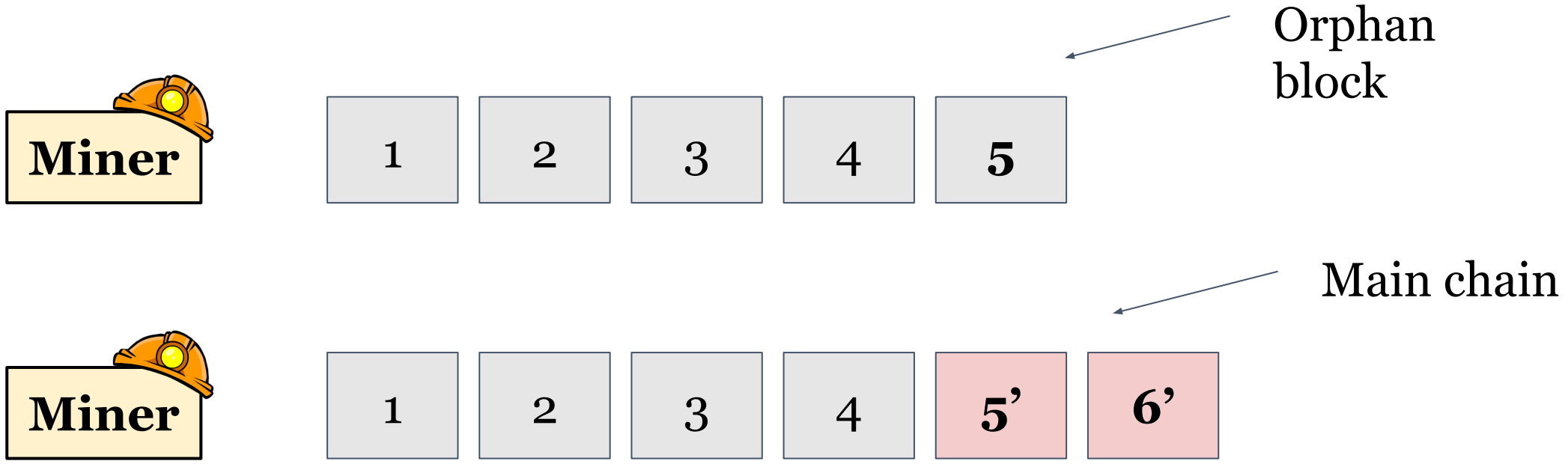



Block 1
...
Block 2
...
Block 3
...
Block 4
...
I found a block
Miner
Miner
Miner
Miner
Miner
Alice
Bob

Mining

1. Join the network, listen for transactions
    a. Validate all proposed transactions
2. Assemble a new valid block
3. Find the nonce to make your block valid

What happens if 2 blocks found at the same time?
Forks can occur



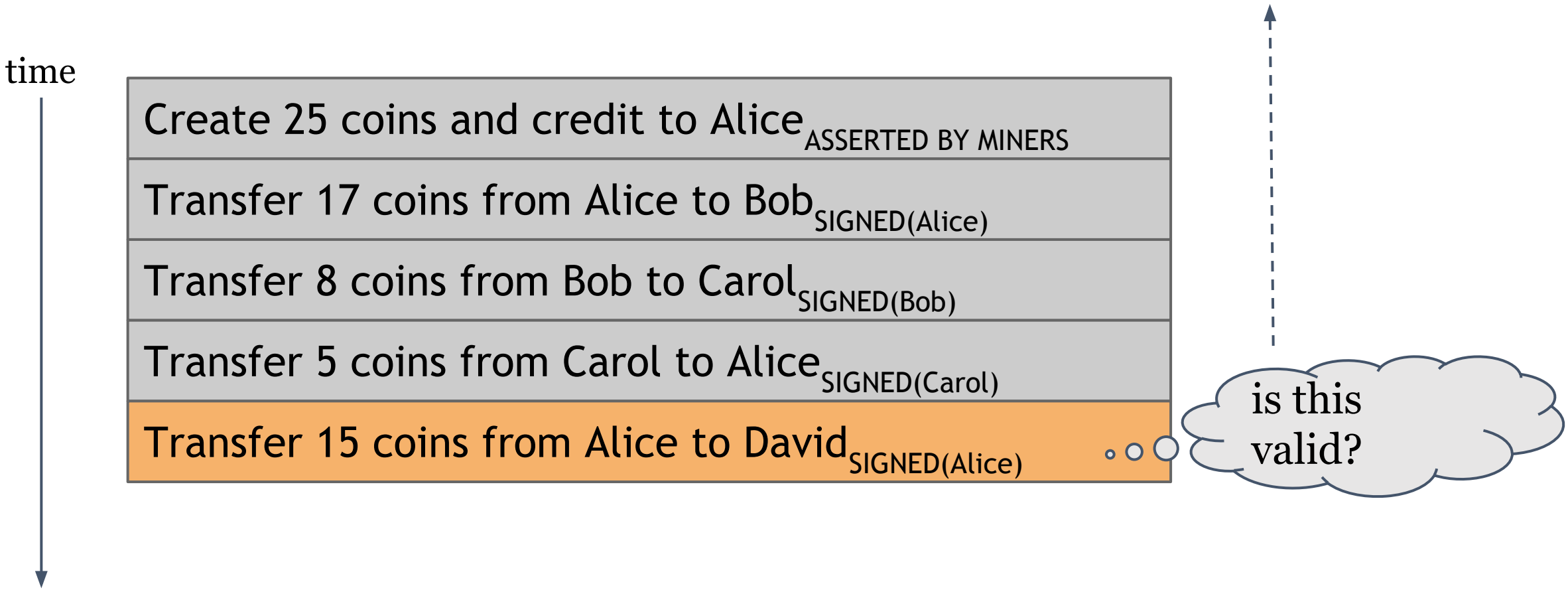
Under the “honest majority” assumption

We can prove that forks are short-lived

Why blockchain?

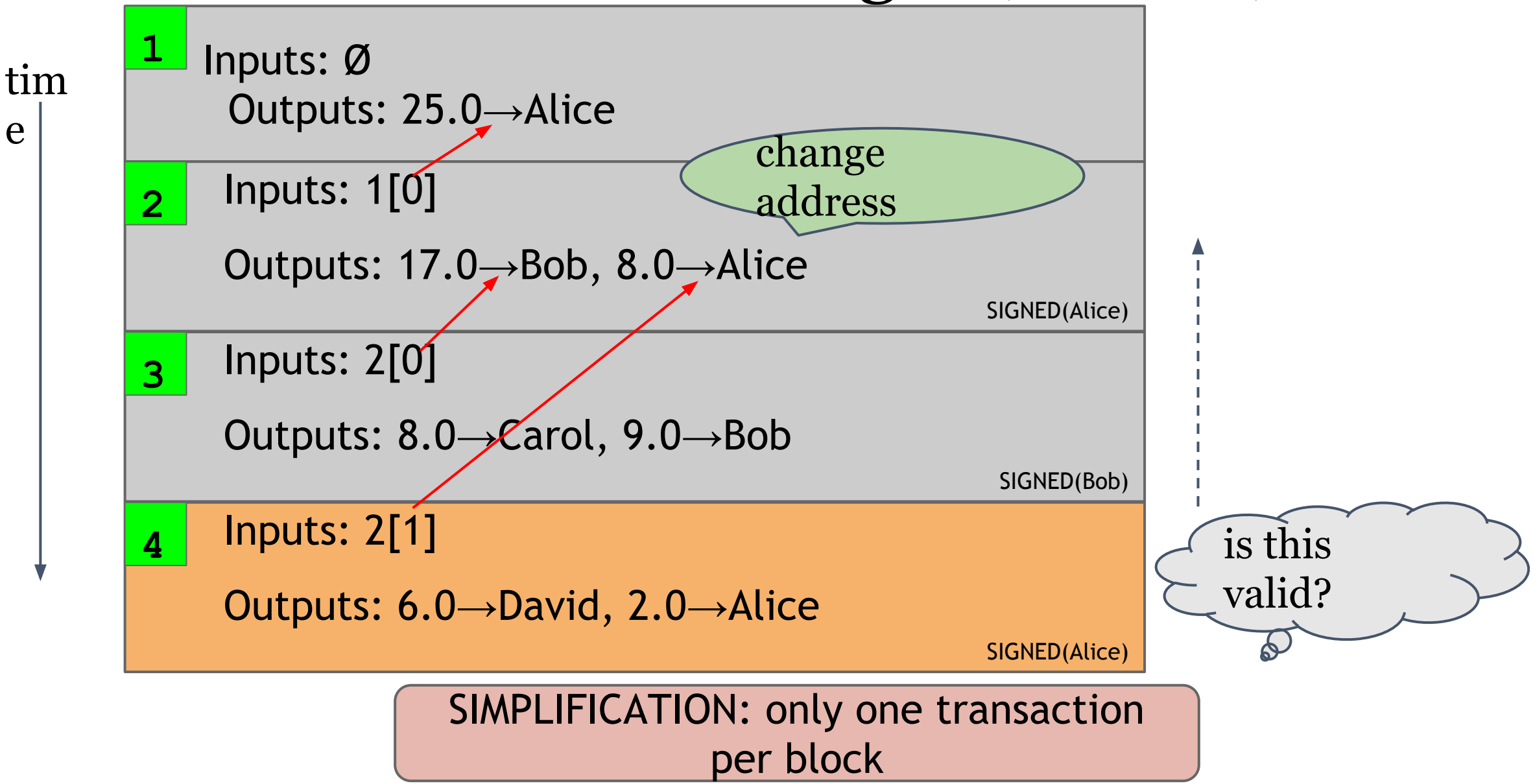
- Distributed consensus
  - New ideas
  - Assumptions
- Game theory
- Cryptography

Account model vs transaction model



SIMPLIFICATION: only one transaction per block

A transaction-based ledger (Bitcoin)



SIMPLIFICATION: only one transaction per block

Meta data for efficient validation

- Account model: account balances
- Transaction model: UTXO (unspent output of transactions)
  - A set of unspent transactions
  - Each new transaction destroys 1 (or several) elements in the set, and insert 2 elements into the set