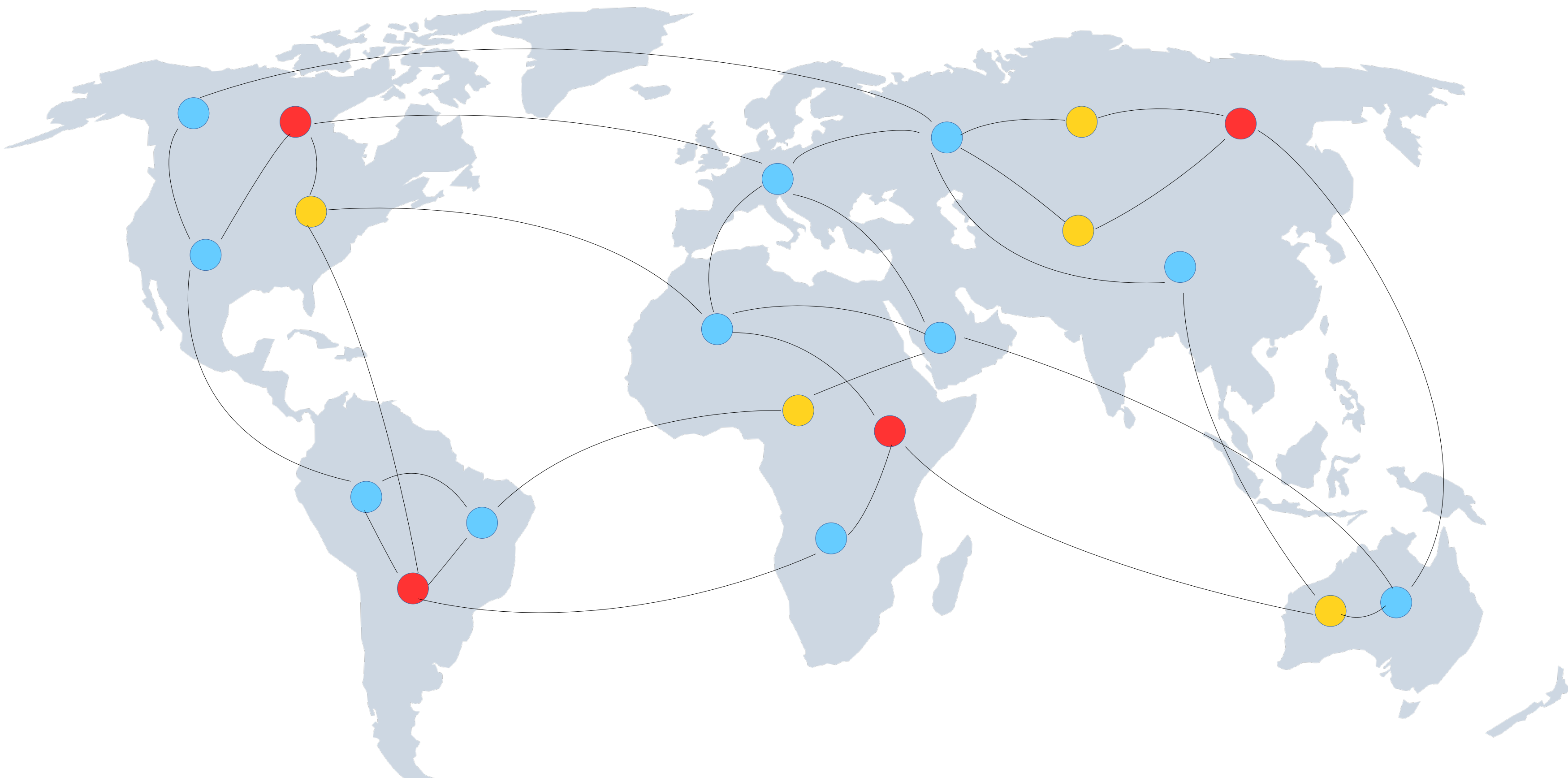
A recipe for a Distributed Ledger: Proof-of-Work, Blockchains, and Bitcoins

Stefano Leucci



Goals

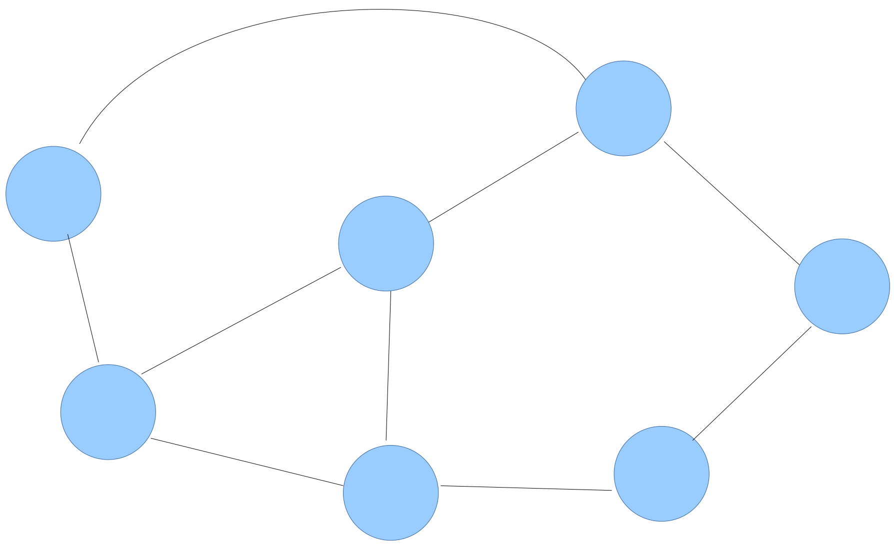
- Understand the basic cryptographic tools of Blockchains.

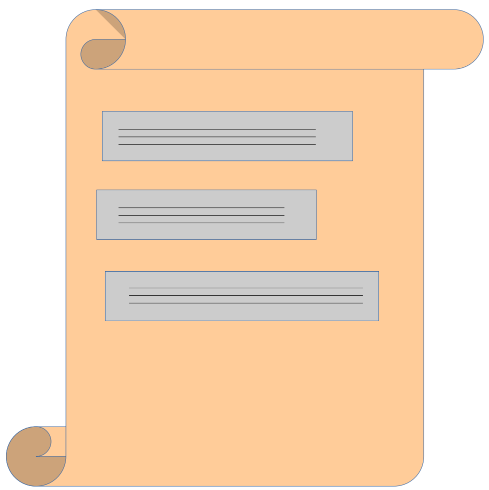
- Give an overview of how Blockhains are built using these tools.

- **Bitcoin** blockchain as an example.

The Distributed Ledger Problem

Maintain a distributed ledger containing a sequence of economic transactions





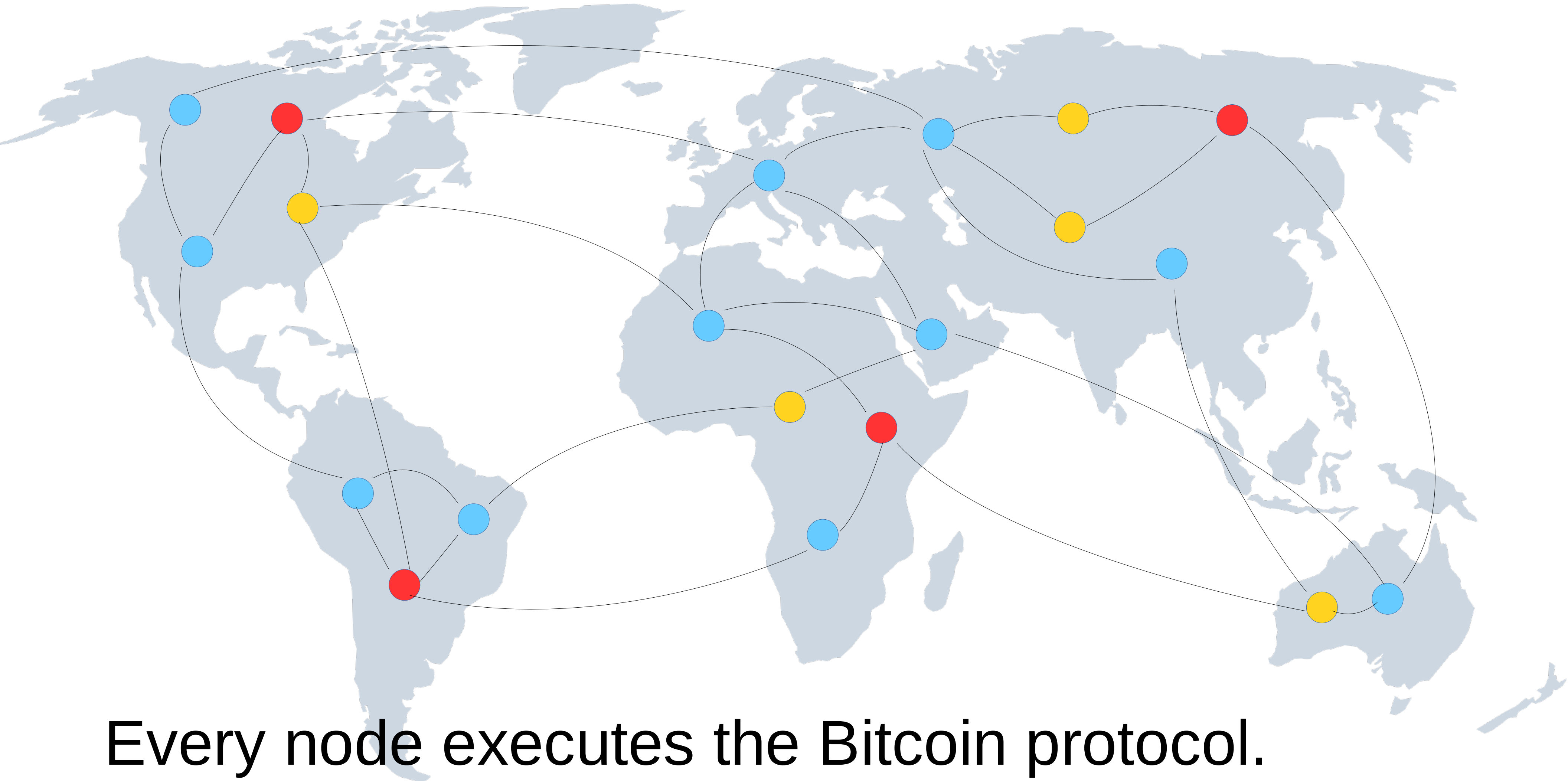
Every node can add transactions to the ledger.

All agents agree on the ledger contents.

No central authority.

The Bitcoin Network

A distributed peer-to-peer overlay network



Every node executes the Bitcoin protocol.

Bitcoin

- A digital cryptocurrency and payment system
- Invented in 2008/2009 by Satoshi Nakamoto
- Currently ≈ 18 900 000 BTC
- Bitcoin generation is on a schedule and will converge to 21 000 000 BTC

Bitcoin: A Peer-to-Peer Electronic Cash System

2008

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

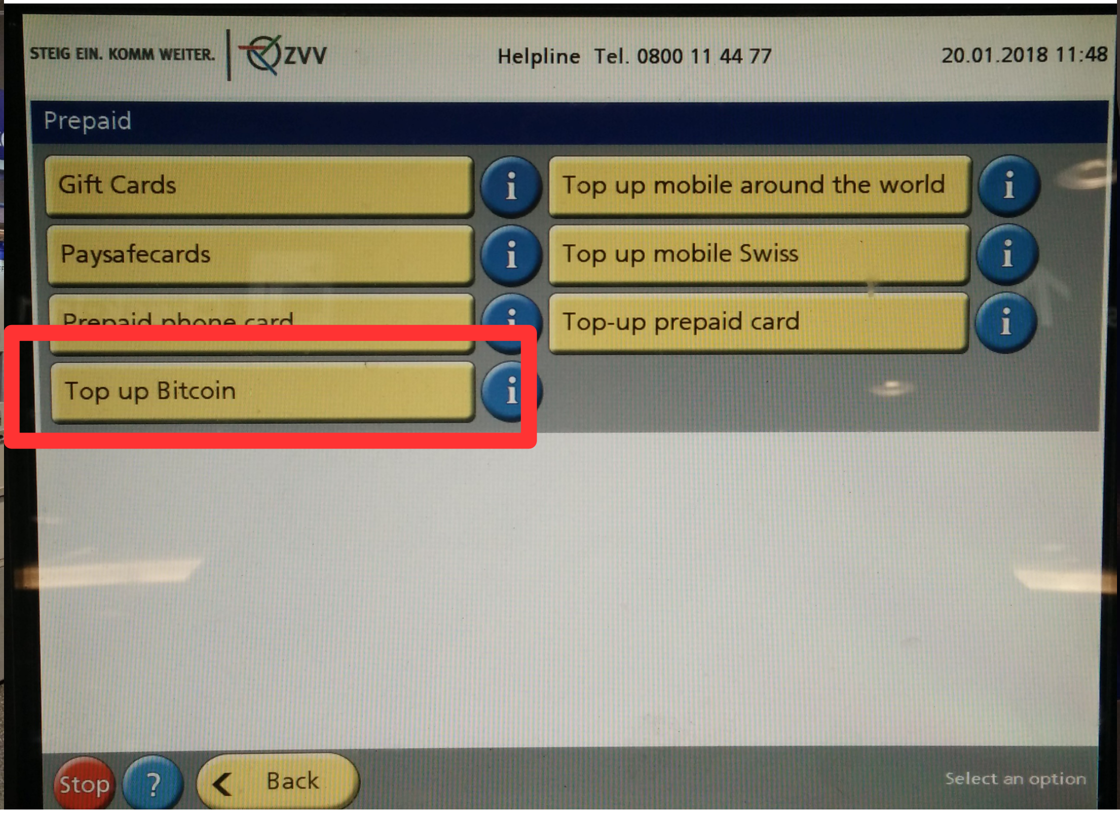
Currently 1 BTC ≈ 48 500$ ≈ 43 000€

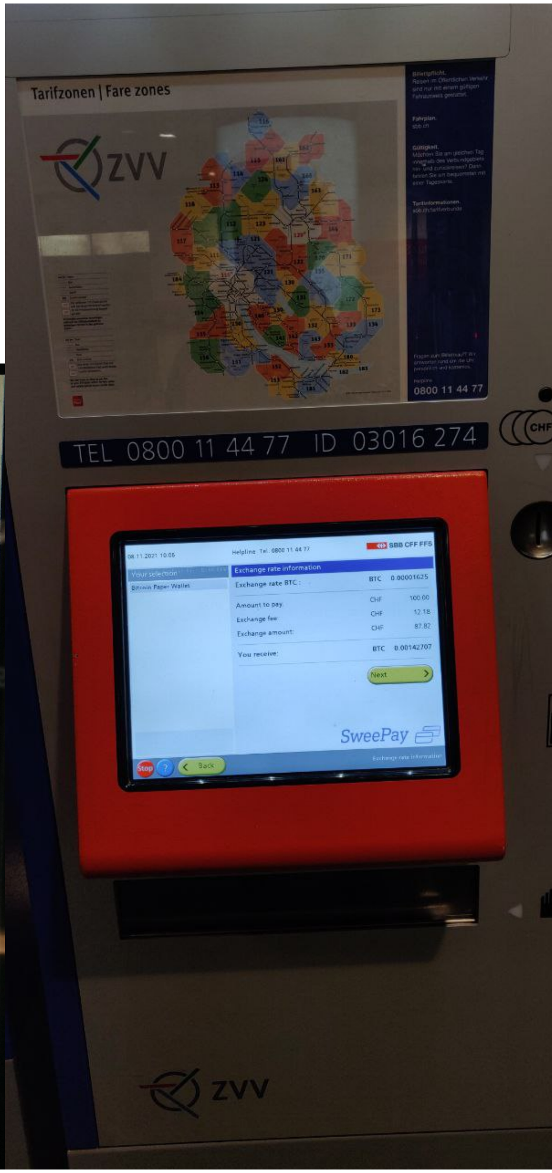
[https://coincap.io/assets/bitcoin]

1 Satoshi = 10^{-8} BTC = 0.000 000 01 BTC

Acquiring Bitcoins

- Receiving a payment for a good/service
- Exchanging with other currencies
- Mining





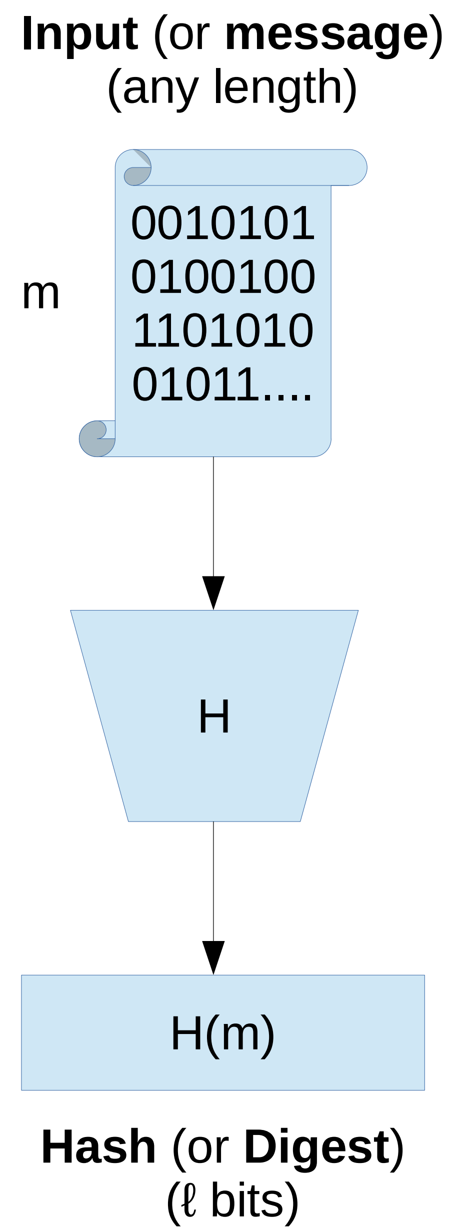
Some Basic Ingredients

Hash Functions

A function $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$

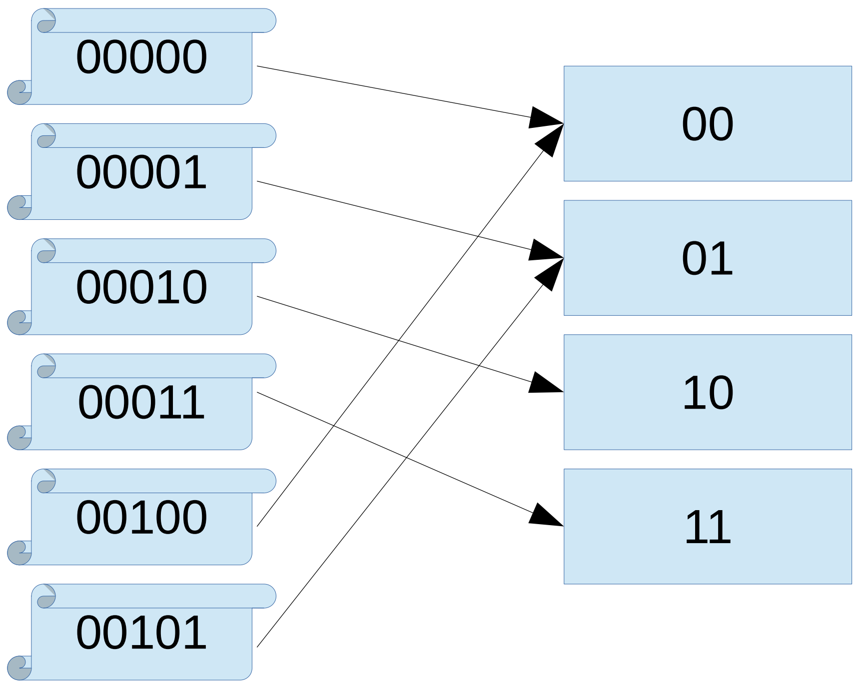
- **Deterministic:** same input ⇒ same output.
- **Uniform:** Hashes are evenly distributed in $\{0,1\}^\ell$

Example: $H(m) = m \bmod 2^\ell$



Hash Functions

Example: $H(x) = x \bmod 2^{\ell}$, $\ell=2$



⋮

Collision: H(00001) = H(00101) = 01

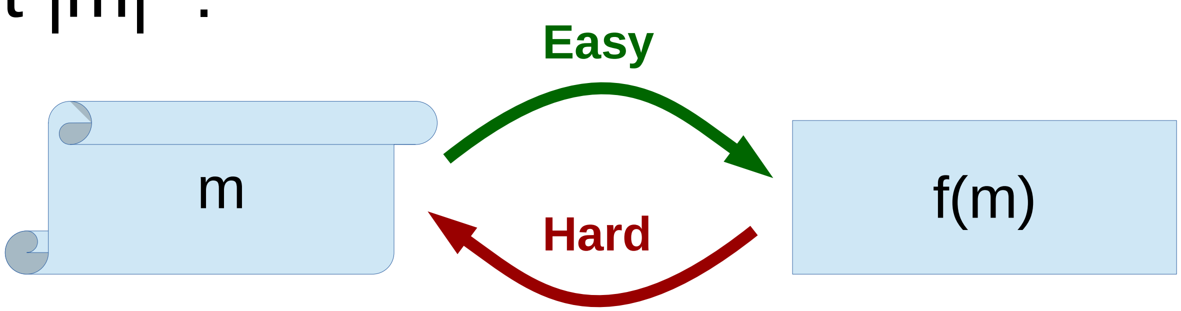
One-way functions

Function $f : \{0,1\}^* \to \{0,1\}^*$ that is:

- **Easy to compute**

  f $\in$ **FP**: Given m, h=f(m) can be computed by a deterministic polynomial-time algorithm.

- **Hard to invert**:
  - $\forall$ c>0 and sufficiently large |m|, $\nexists$ randomized polynomial-time algorithm A(f(m)) that computes x such that f(x)=f(m) with a success probability of at least $|m|^{-c}$.



Do one-way functions exist?

We don’t know!

Major open problem in computer science.

$\exists$ One way functions $\Rightarrow$ FP $\neq$ FNP $\Rightarrow$ **P ≠ NP**.

Informally: *is it true that every problem whose solution can be* ***efficiently verified*** *can also be* ***efficiently solved****?*

Millennium prize problem ($1,000,000 from Clay Institute)

One-Way Functions: Candidates

Factoring:

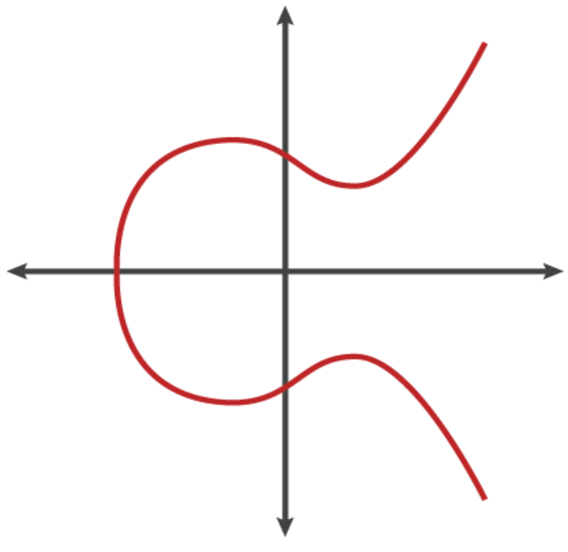
Given two primes p,q: easy to compute x=pq
Hard to factor x into p and q.

Discrete logarithm:

Given k and p, easy to compute $x=2^k$ mod p.
Hard to find k from x and p.

Elliptic Curves:

Point multiplication is easy
to compute and hard to invert.



Hash Functions (Attacks)

Preimage attack:

given h, find m such that H(m) = h.

Second preimage attack:

given m_1, find $m_2 \neq m_1$ such that $H(m_1) = H(m_2)$.

Birthday attack:

find m_1 and $m_2 \neq m_1$ such that $H(m_1) = H(m_2)$.

Cryptographic Hash Functions

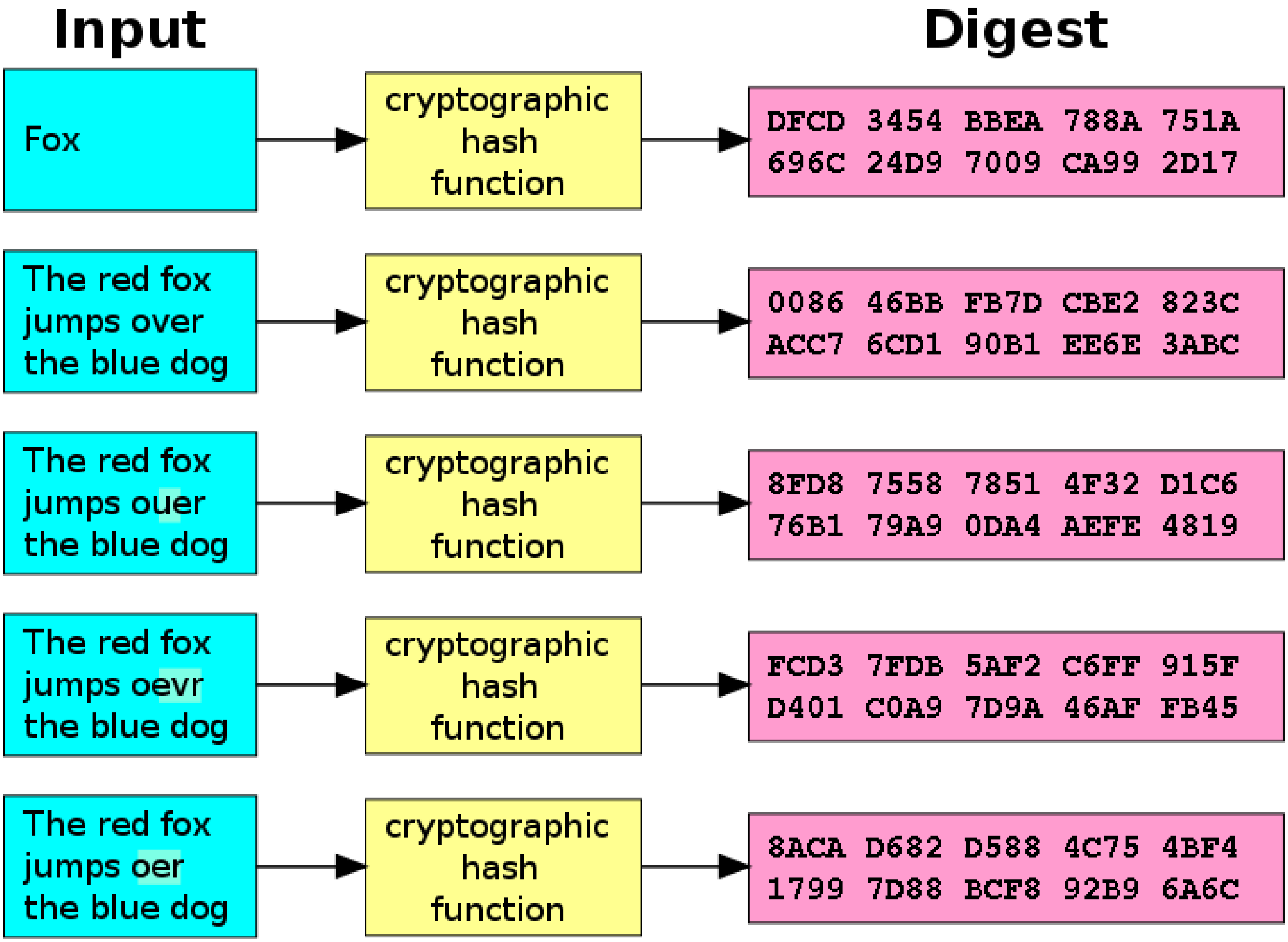
Collisions are unavoidable.

Next best thing: collisions are **hard to find**.

Cryptographic Hash Function H:

- Is a one-way function: avoids pre-image attacks.
- Resistant to second pre-image attacks.
- Collision resistant; avoids birthday attacks.
- A small change to the input produces a big change in the output.
- Resistant to other attacks (e.g., length extension).

Very Informally: H looks "random".

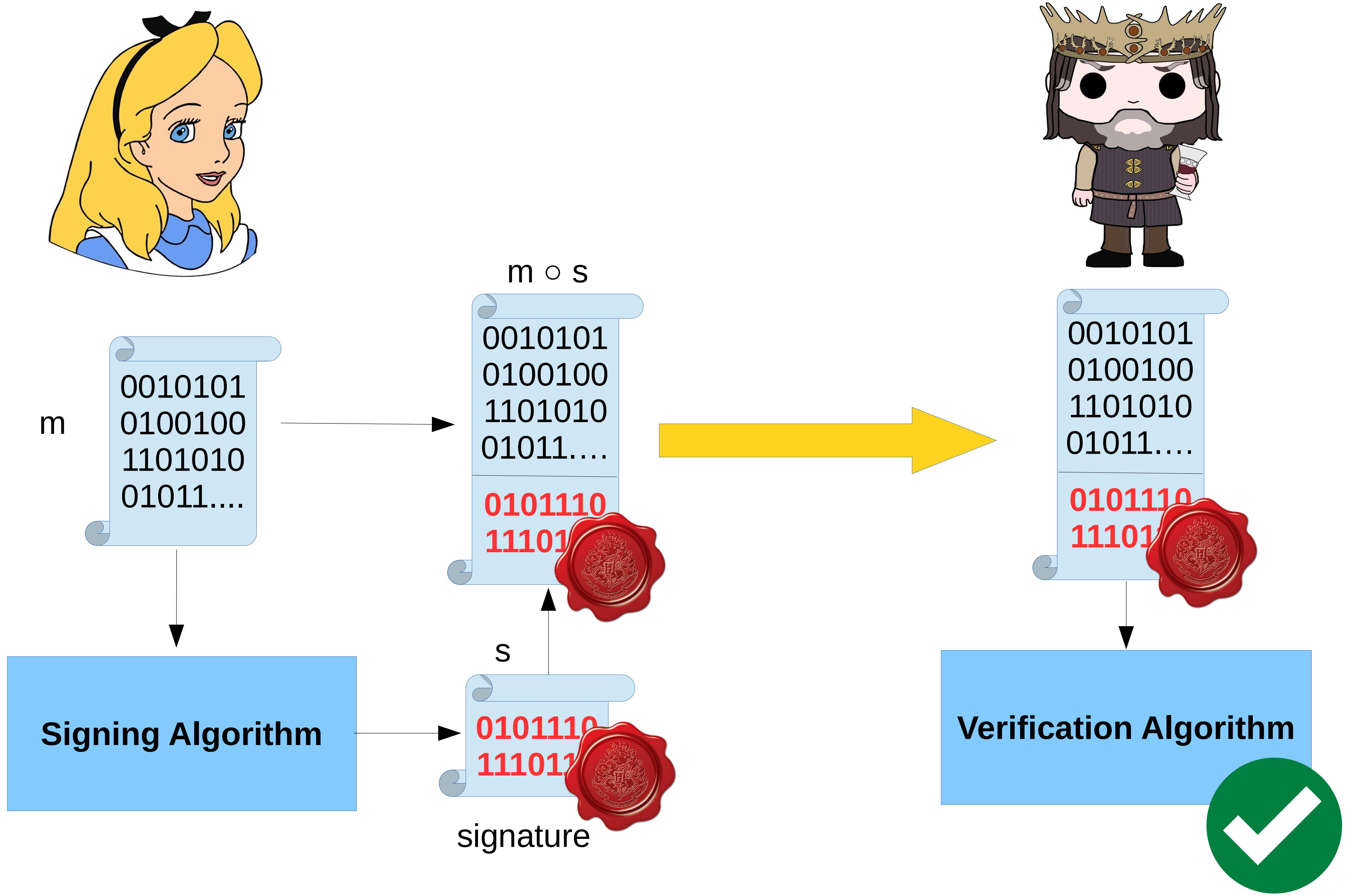


Famous Cryptographic Hash Functions

- ~~**MD4**~~ Birthday attack (μs), Preimage attack
- ~~**MD5**~~ Birthday attack (s), Preimage attack (theoretical)
- ~~**SHA0**~~ Birthday attack (< 1hour)
- ~~**SHA1**~~ Birthday attack (110 years on GPU)
- **SHA2:** SHA-256,SHA-384, SHA-512
- **SHA3**: Keccak

...

Digital Signatures



Digital Signatures

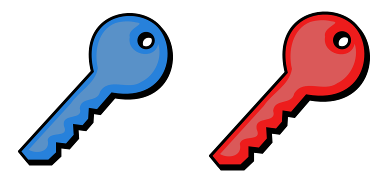
A mathematical scheme that **signs** a message to guarantee:

- **Authentication**: Bob knows Alice sent the message
- **Non-repudiation**: Alice cannot deny having sent the message
- **Integrity**: The message was not altered in transit

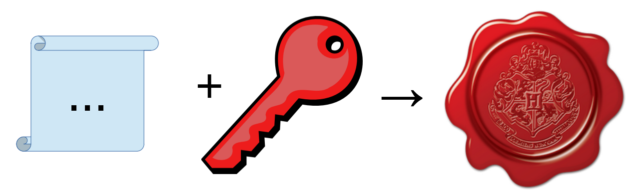
Public and Private Keys

Three ingredients (algorithms):

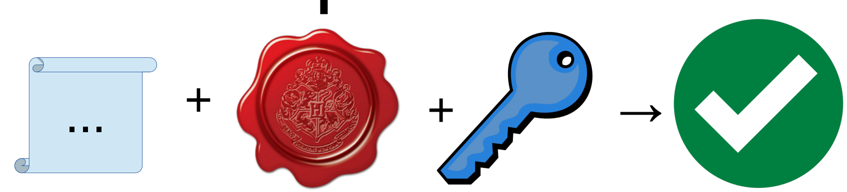
- **Key generation**: generates a pair (pk,sk) of public and private (secret) keys.



- **Signing**: Given a message m and a private key sk produces a digital signature s.

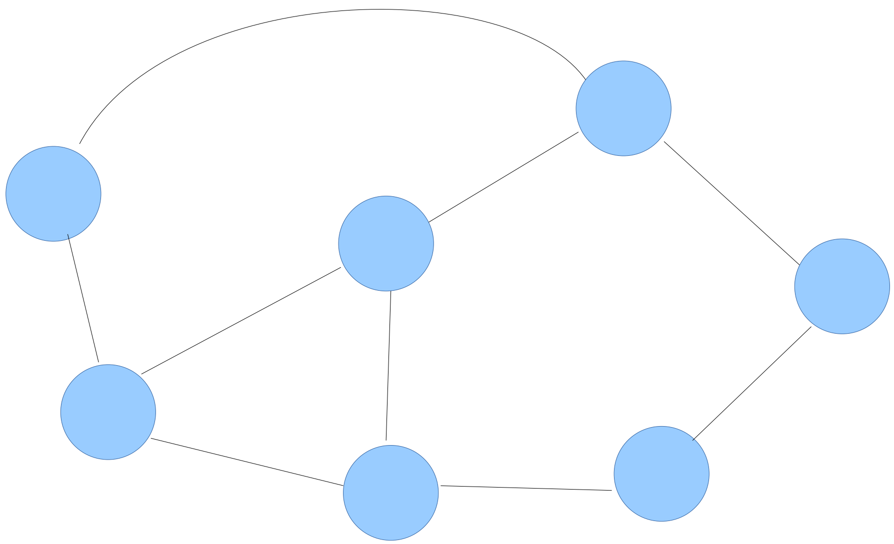


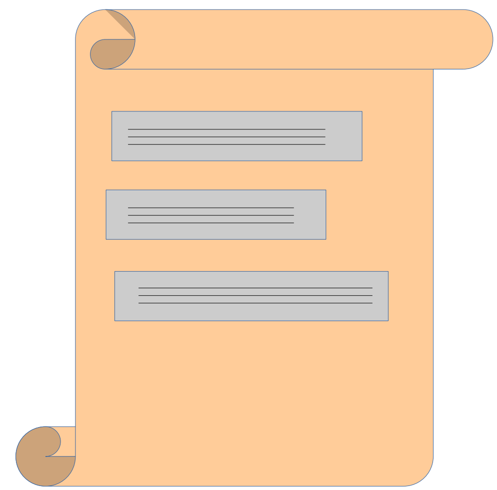
- **Signature Verification**: Given m, s, and pk, verifies that the signature s matches m has been produced using sk associated with pk.



The ~~Distributed~~ Ledger Problem

Maintain a ~~distributed~~ ledger containing a sequence of economic transactions





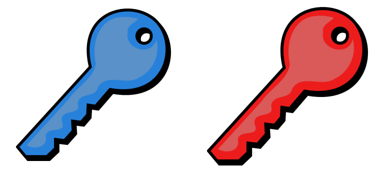
Every node can add transactions to the ledger.

All agents agree on the ledger contents.

~~No central authority.~~

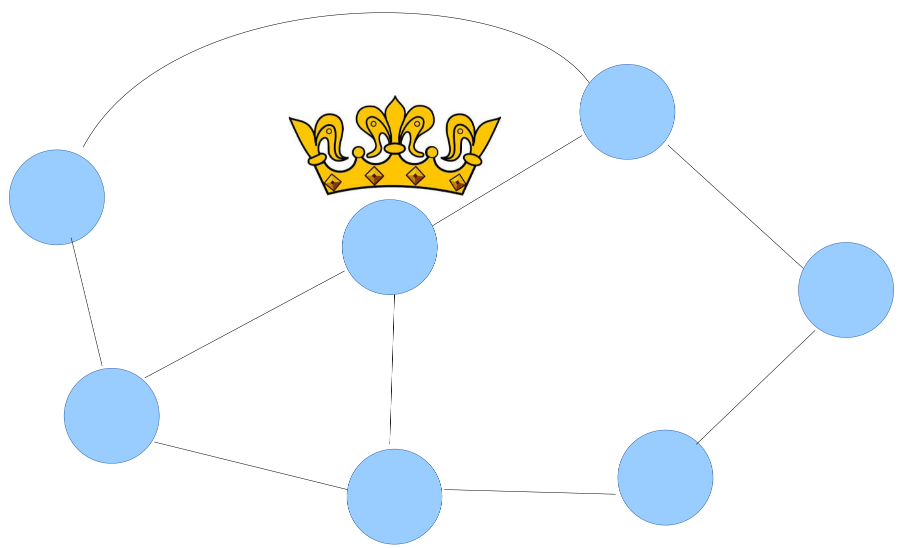
Solving the Ledger Problem with a trusted central authority

Each node owns a public/private key pair

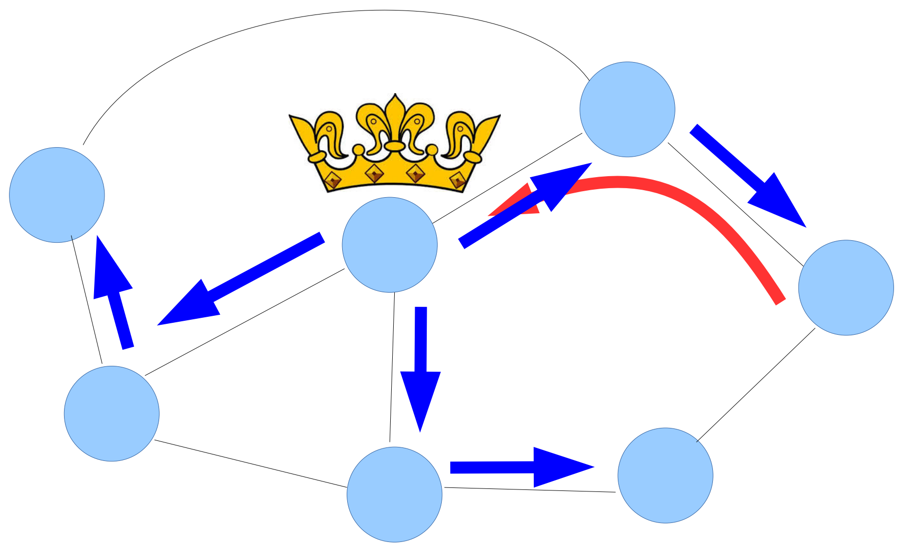


The public key of the central authority is known to all nodes of the network

The central authority knows the public key of all other nodes



Solving the Ledger Problem with a trusted central authority



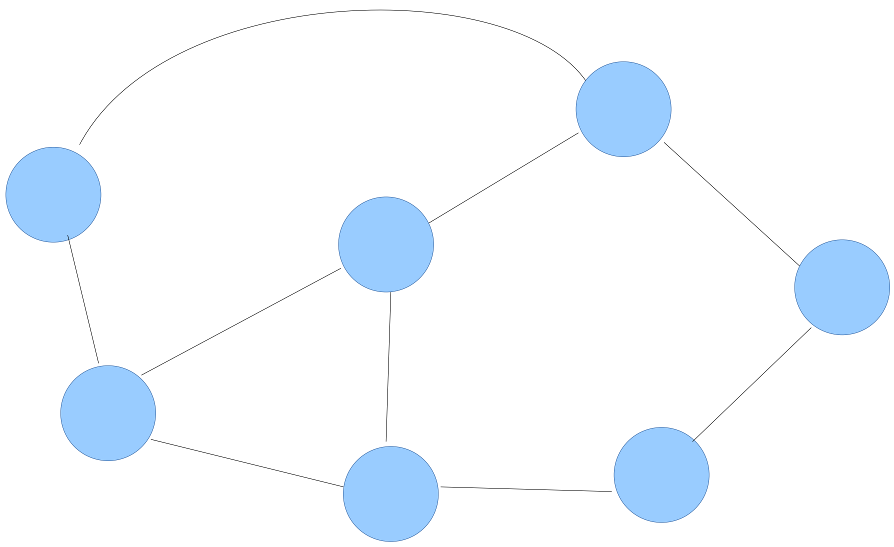
Nodes send their transactions to the authority

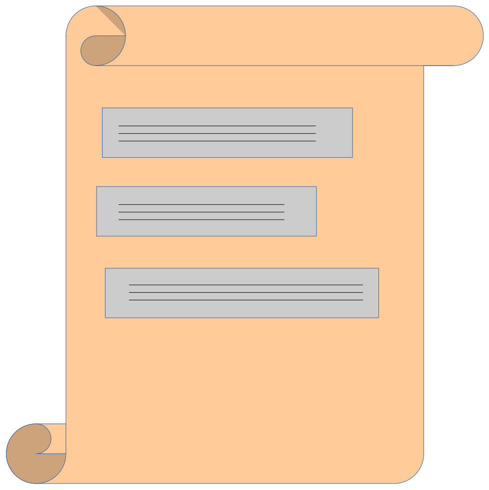
The authority publishes and updates the ledger

All identities are checked through digital signatures

The Distributed Ledger Problem

Maintain a distributed ledger containing a sequence of economic transactions





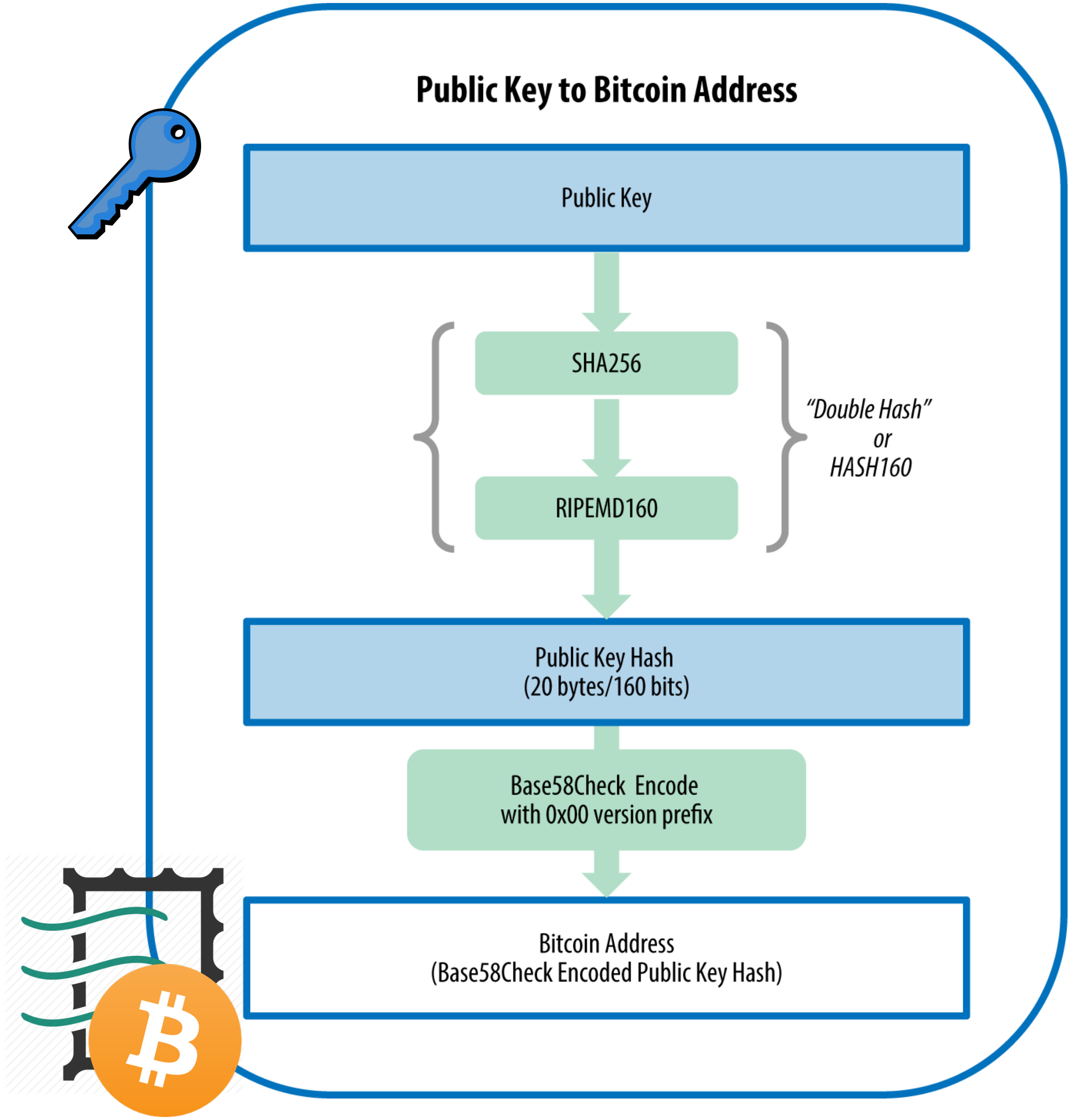
Every node can add transactions to the ledger.

All agents agree on the ledger contents.

No central authority.

The Bitcoin Architecture

Bitcoin Address



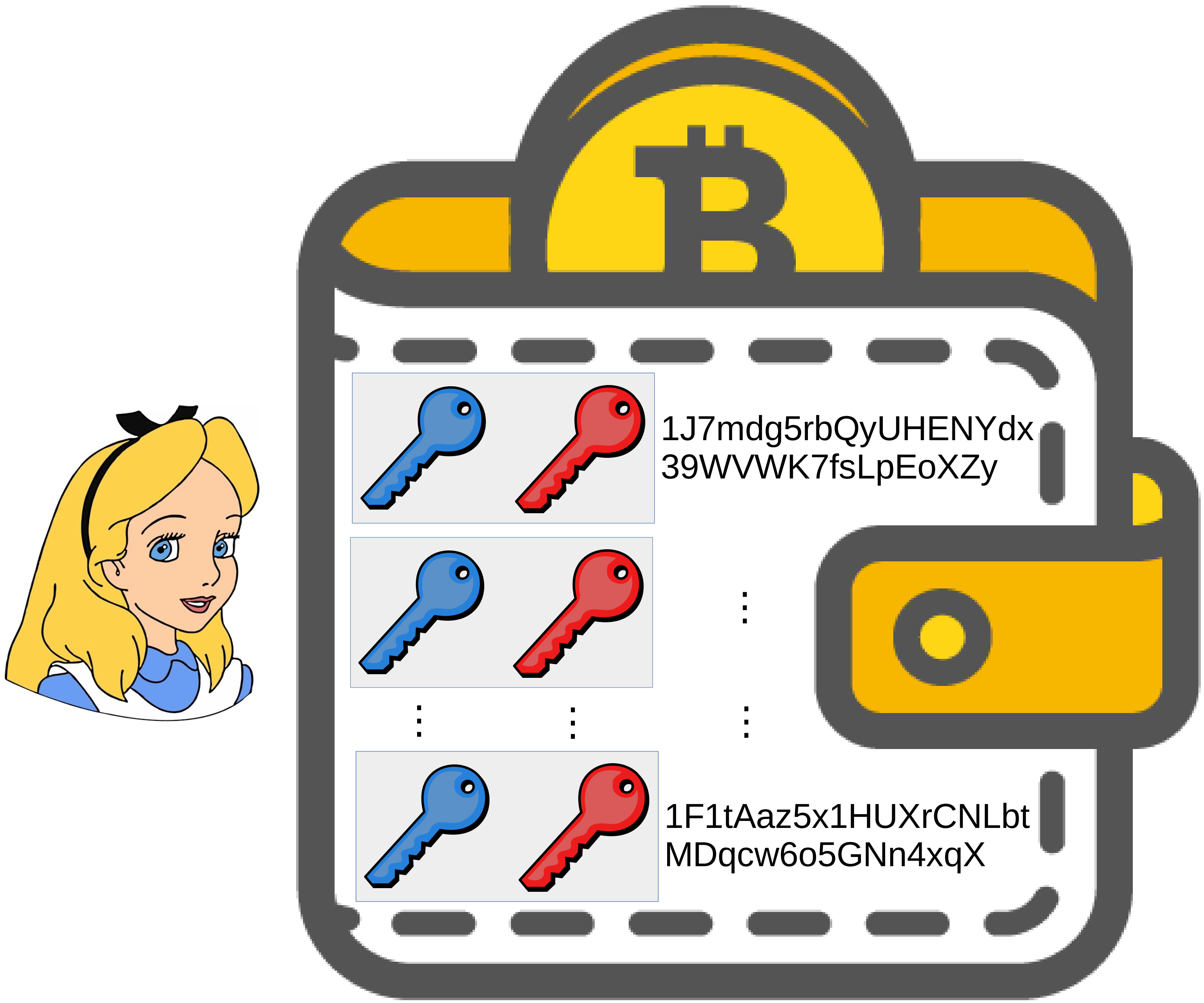
Bitcoin Address

 → 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy 

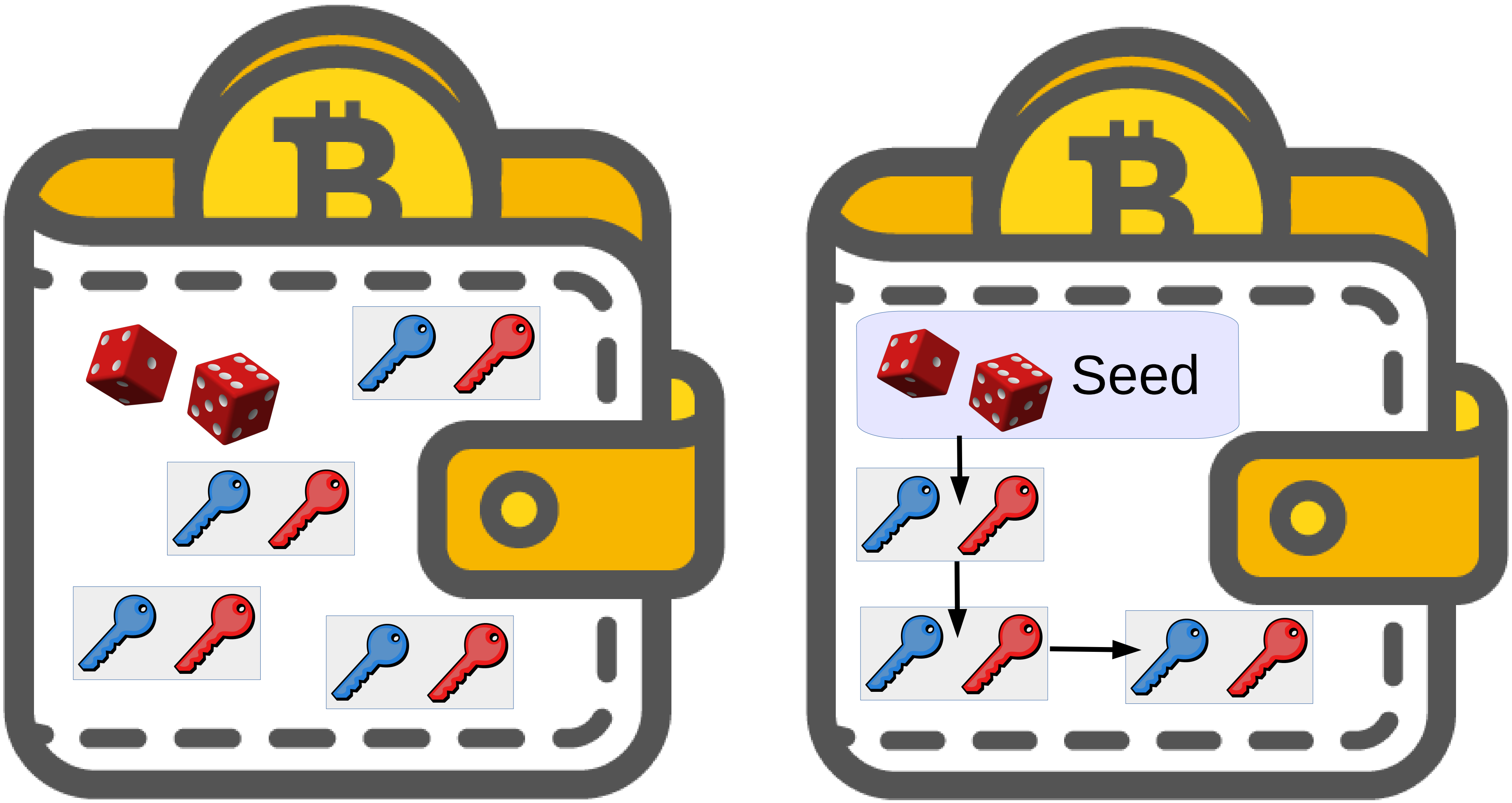
Bitcoins are “owned” by an address (a public key). 

They can be spent by whoever controls the corresponding private key. 

Bitcoin Wallet



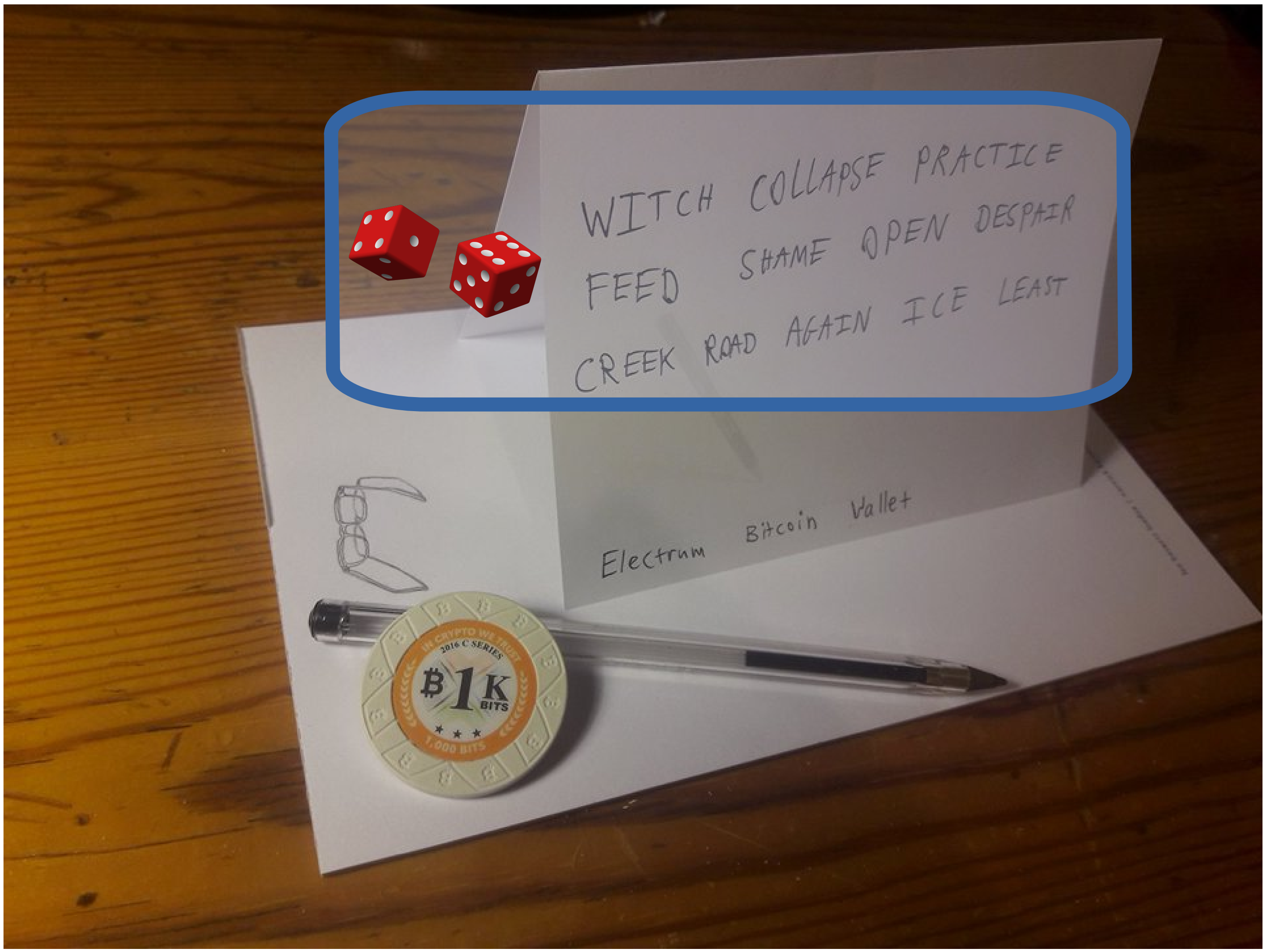
Bitcoin Wallet Types



Non-Deterministic
(Random)

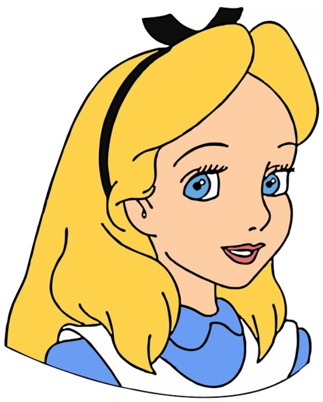
Deterministic
(Seeded)

Mnemonic Seed



Transactions

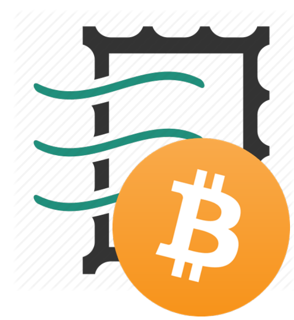
Transactions transfer Bitcoins between addresses.



INPUT(s)

OUTPUT(s)

0.0005 BTC



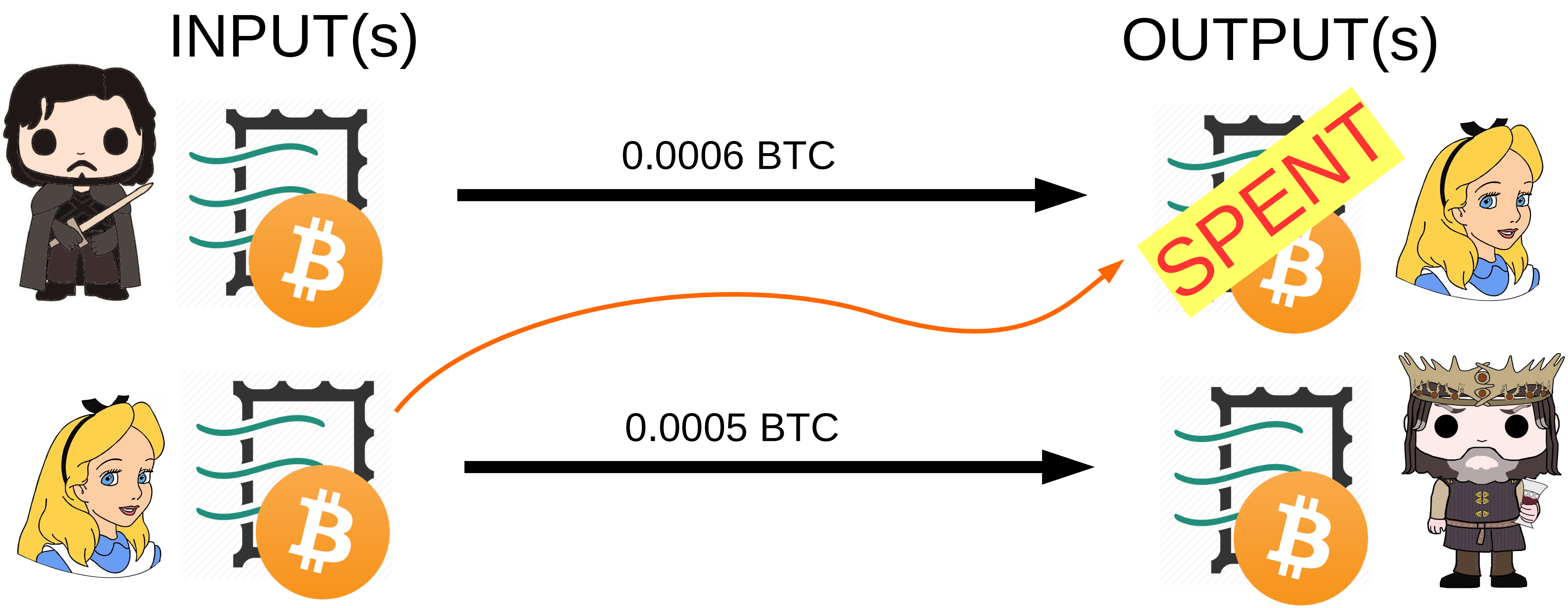
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy

1Mz7153HMuxXTuR2R1t78mGSdzaAtNbBWX

Transactions

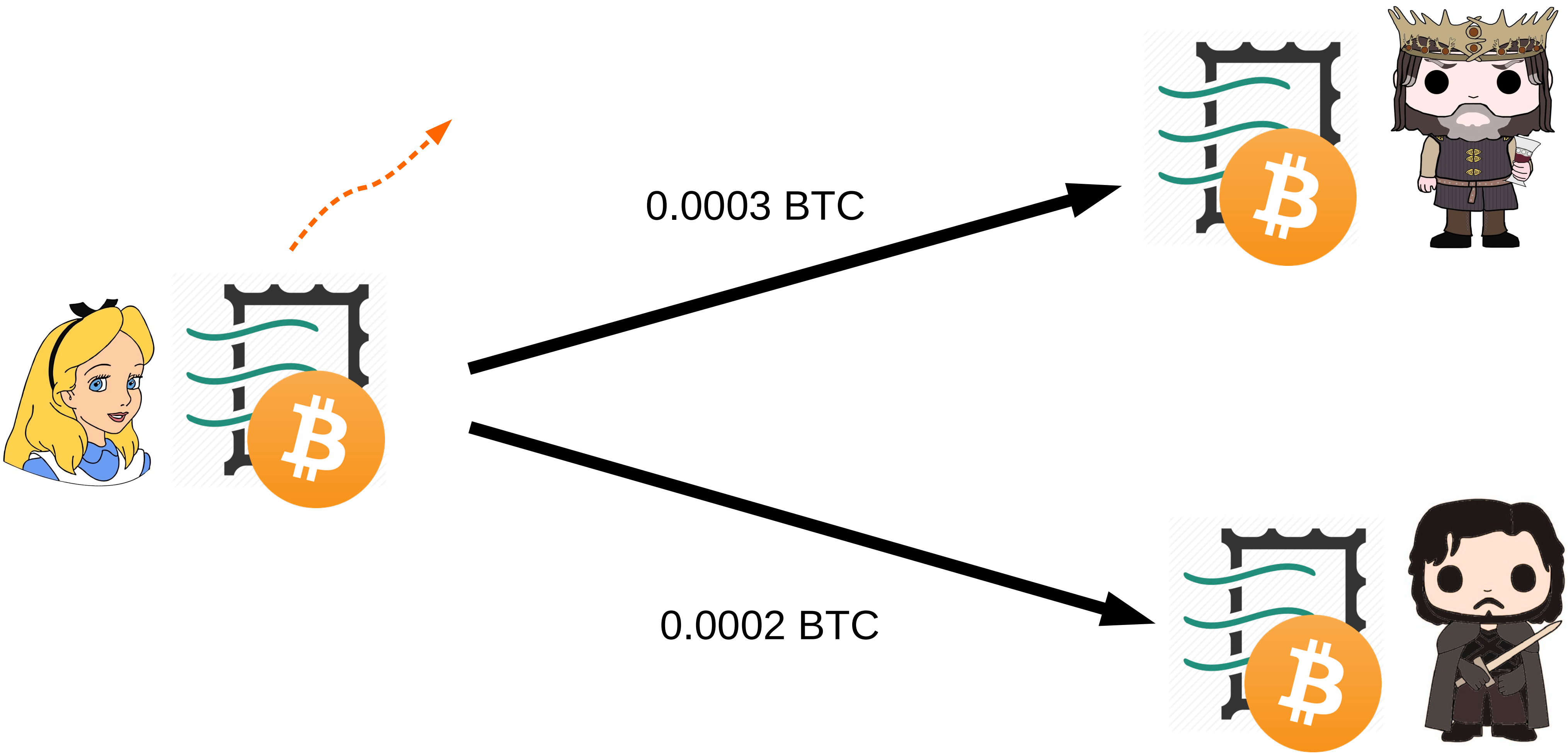
Alice must **prove** that her input address owns the funds (**no double spending**).

She does so by referencing an **unspent output** in a previous transaction.



Multiple Inputs/Outputs

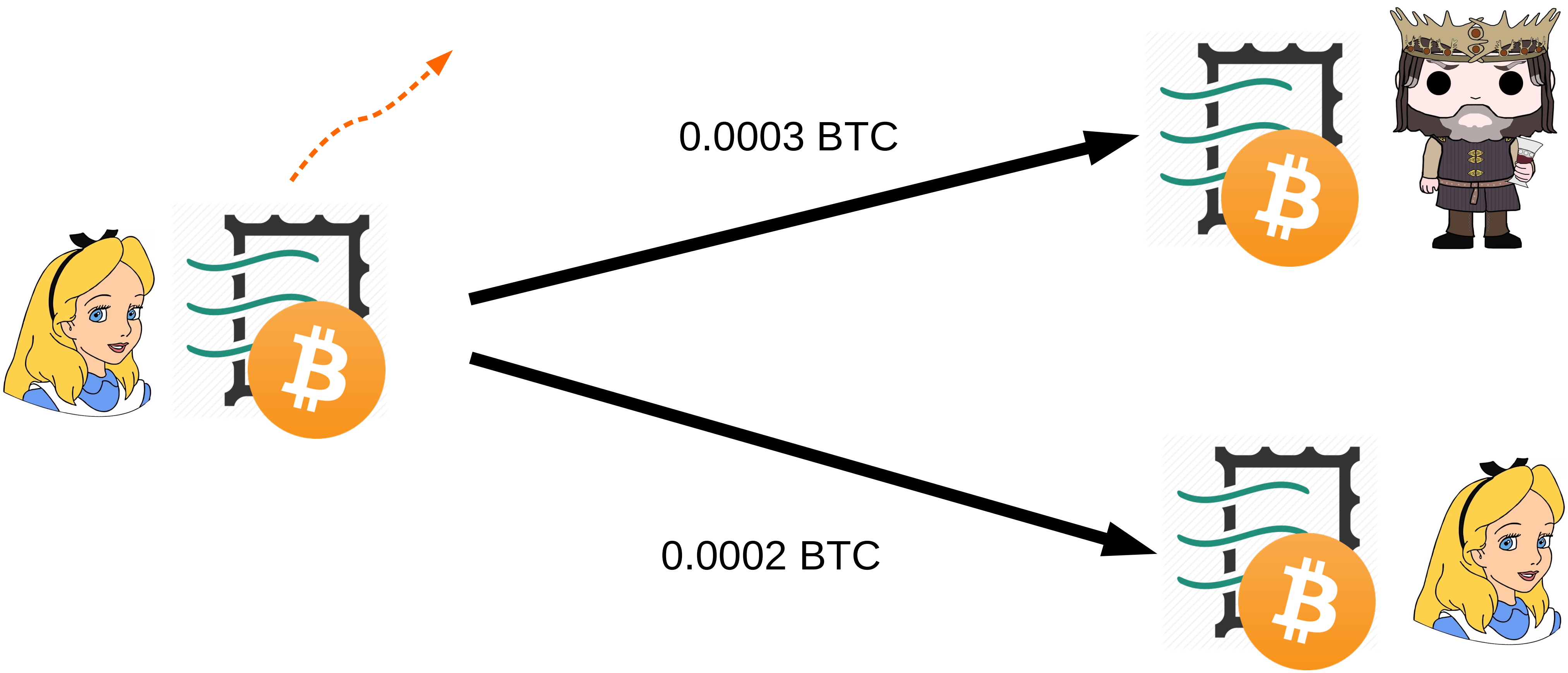
A transaction might contain multiple inputs and/or outputs.



No Change

Outputs are either unspent or **completely spent**.

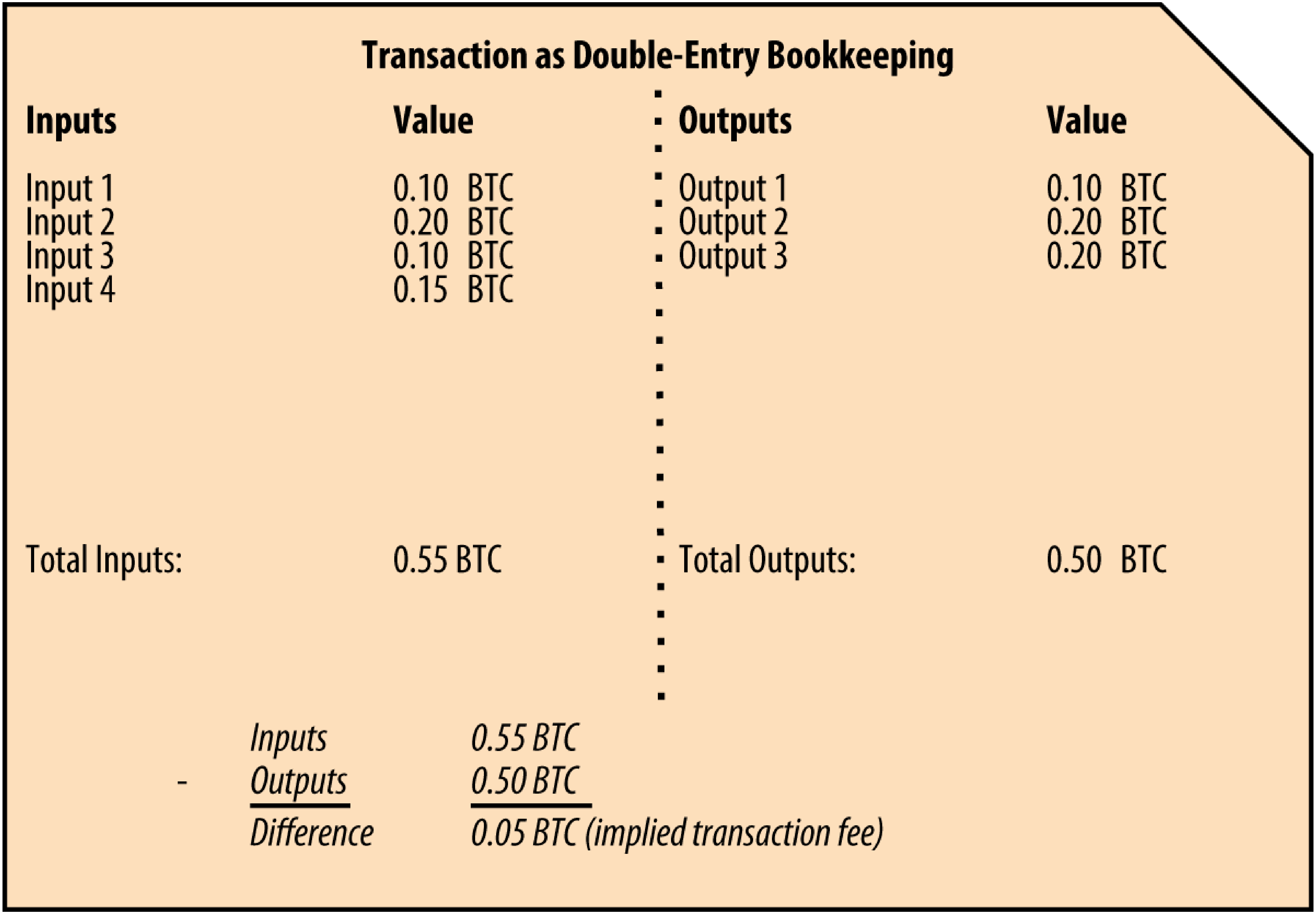
If the input is amount is too large, change can be collected by adding an additional output.



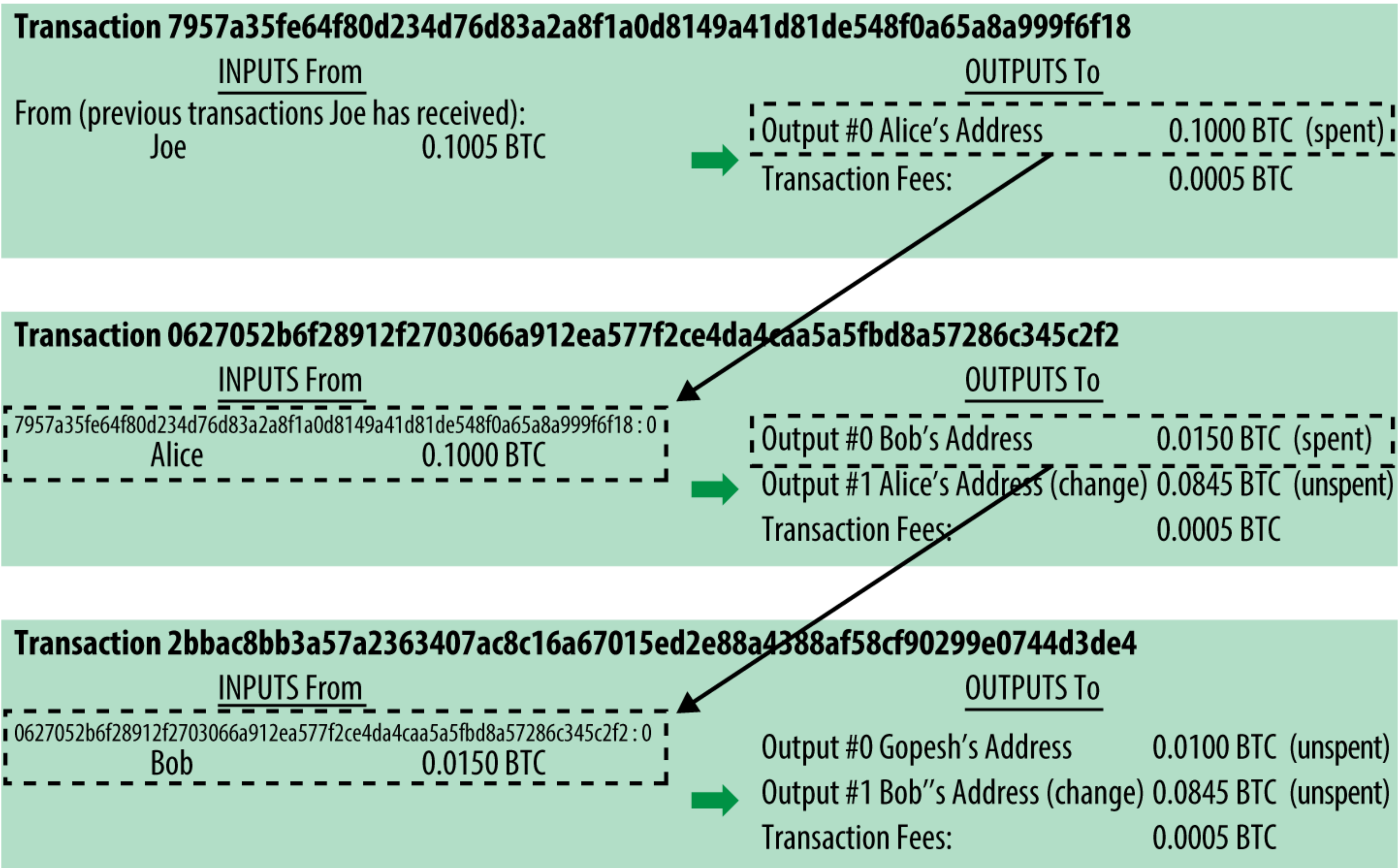
Transaction Fees

If an input is not completely spent, the difference between inputs and outputs is an implicit **transaction fee.**

Each transaction should have a non-zero transaction fee.



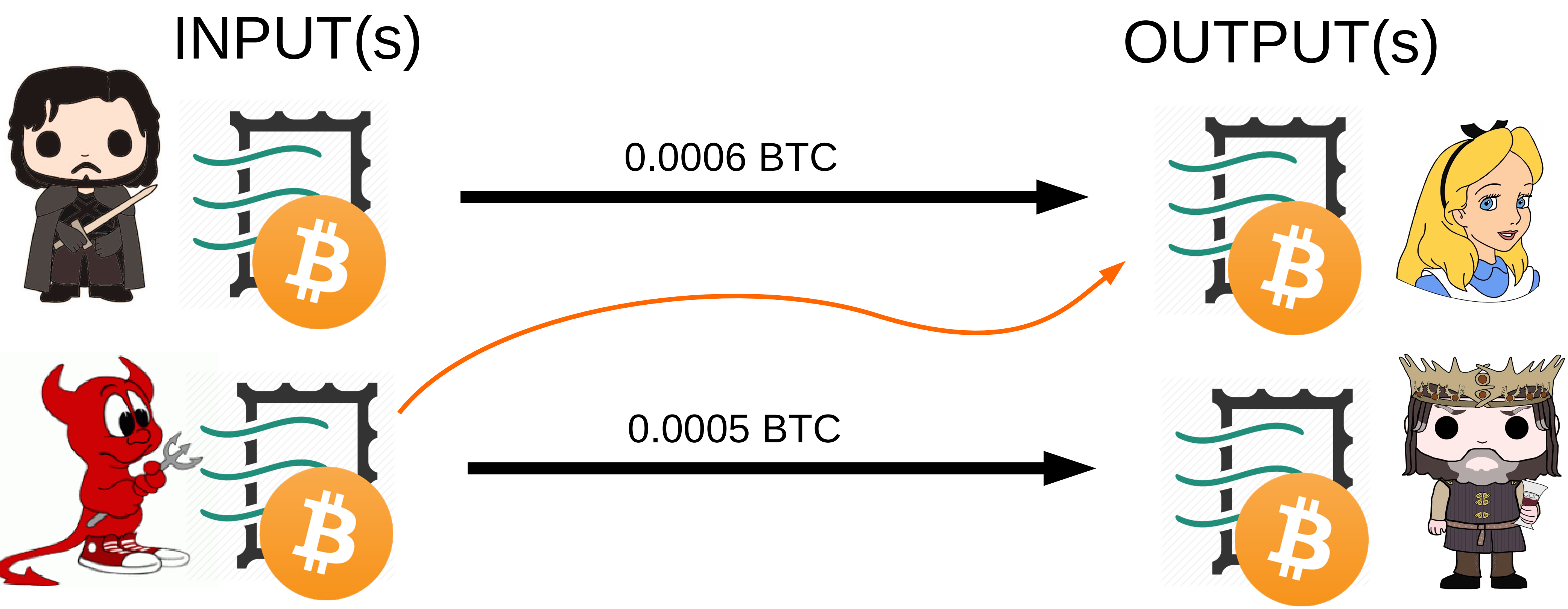
Transaction as Double-Entry Bookkeeping
Inputs
Value
Outputs
Value
Input 1
0.10 BTC
Input 2
0.20 BTC
Input 3
0.10 BTC
Input 4
0.15 BTC
Output 1
0.10 BTC
Output 2
0.20 BTC
Output 3
0.20 BTC
Total Inputs:
0.55 BTC
Total Outputs:
0.50 BTC
Inputs
0.55 BTC
- Outputs
0.50 BTC
Difference
0.05 BTC (implied transaction fee)



Transaction 7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18
INPUTS From
From (previous transactions Joe has received):
Joe 0.1005 BTC
OUTPUTS To
Output #0 Alice's Address 0.1000 BTC (spent)
Transaction Fees: 0.0005 BTC
Transaction 0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2
INPUTS From
7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18 : 0
Alice 0.1000 BTC
OUTPUTS To
Output #0 Bob's Address 0.0150 BTC (spent)
Output #1 Alice's Address (change) 0.0845 BTC (unspent)
Transaction Fees: 0.0005 BTC
Transaction 2bbac8bb3a57a2363407ac8c16a67015ed2e88a4388af58cf90299e0744d3de4
INPUTS From
0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2 : 0
Bob 0.0150 BTC
OUTPUTS To
Output #0 Gopesh's Address 0.0100 BTC (unspent)
Output #1 Bob''s Address (change) 0.0845 BTC (unspent)
Transaction Fees: 0.0005 BTC

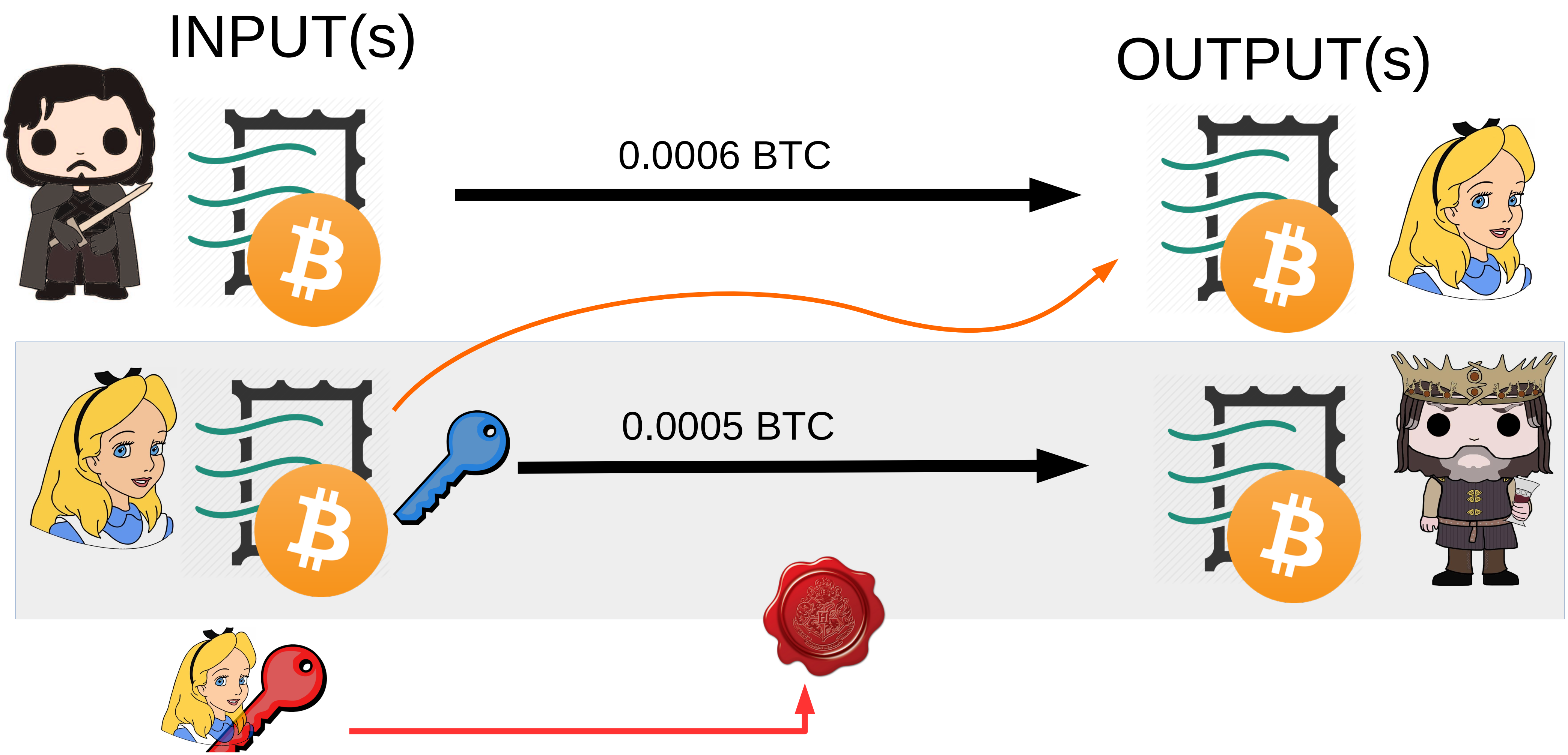
Digital Signatures

The following transaction must be rejected by the network.



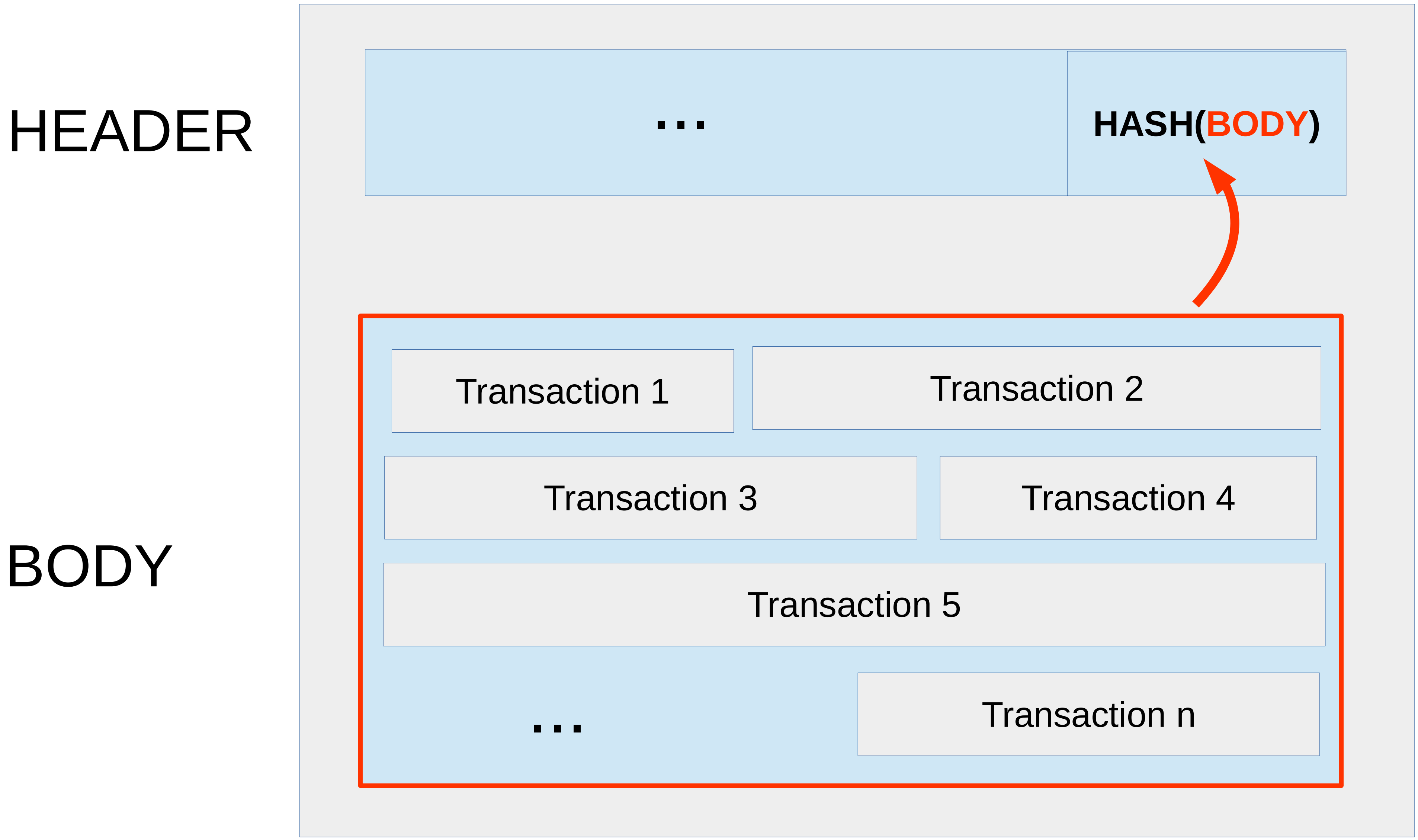
Digital Signatures to the Rescue

Alice provides the **public key** corresponding to the input address and signs the transaction with her **private key**.



Blocks

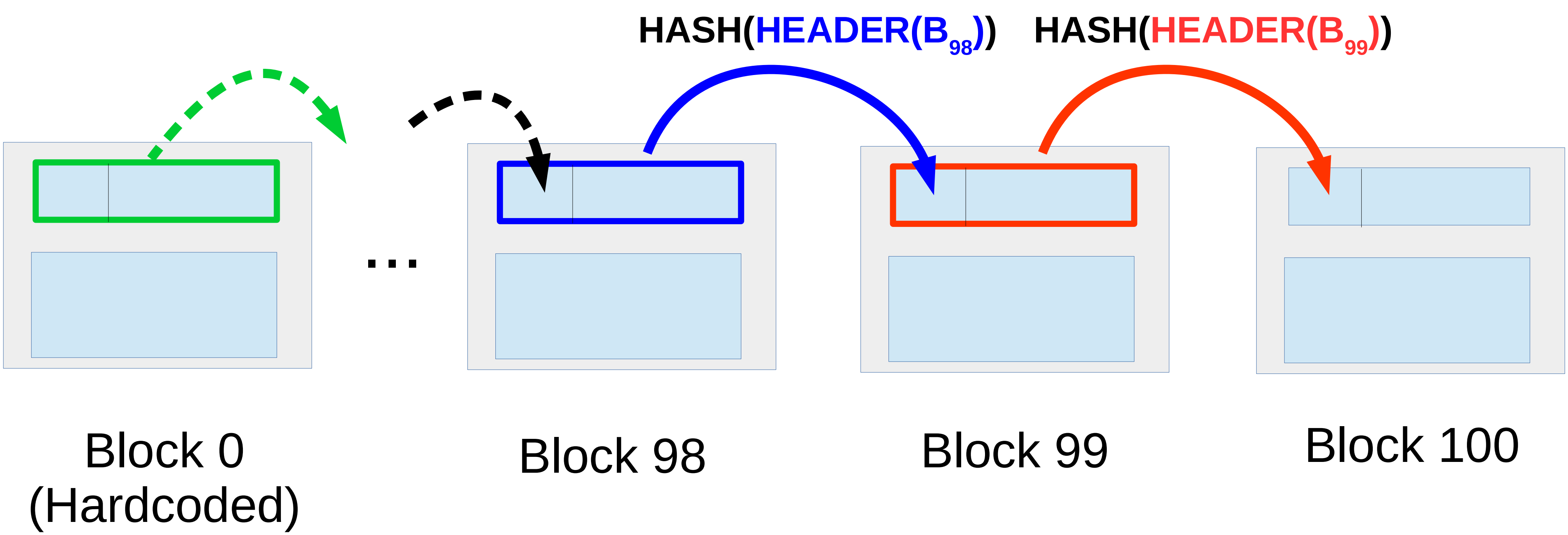
Transaction are grouped into Blocks



Blockchain

Blocks are linked together to form a chain.

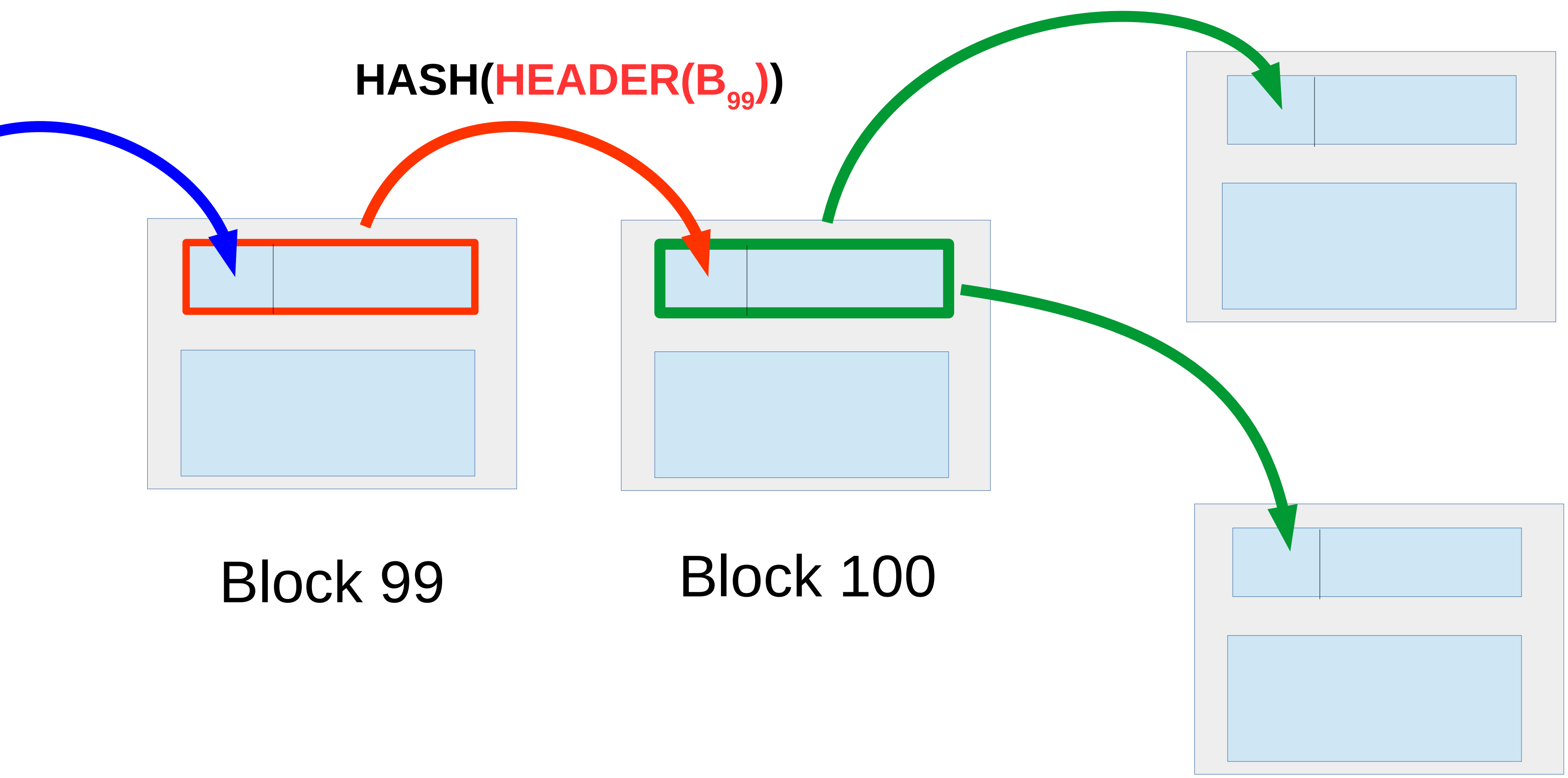
Each block stores the hash of the parent block header.



Blockchain Forks

It is possible for two blocks to extend the blockchain

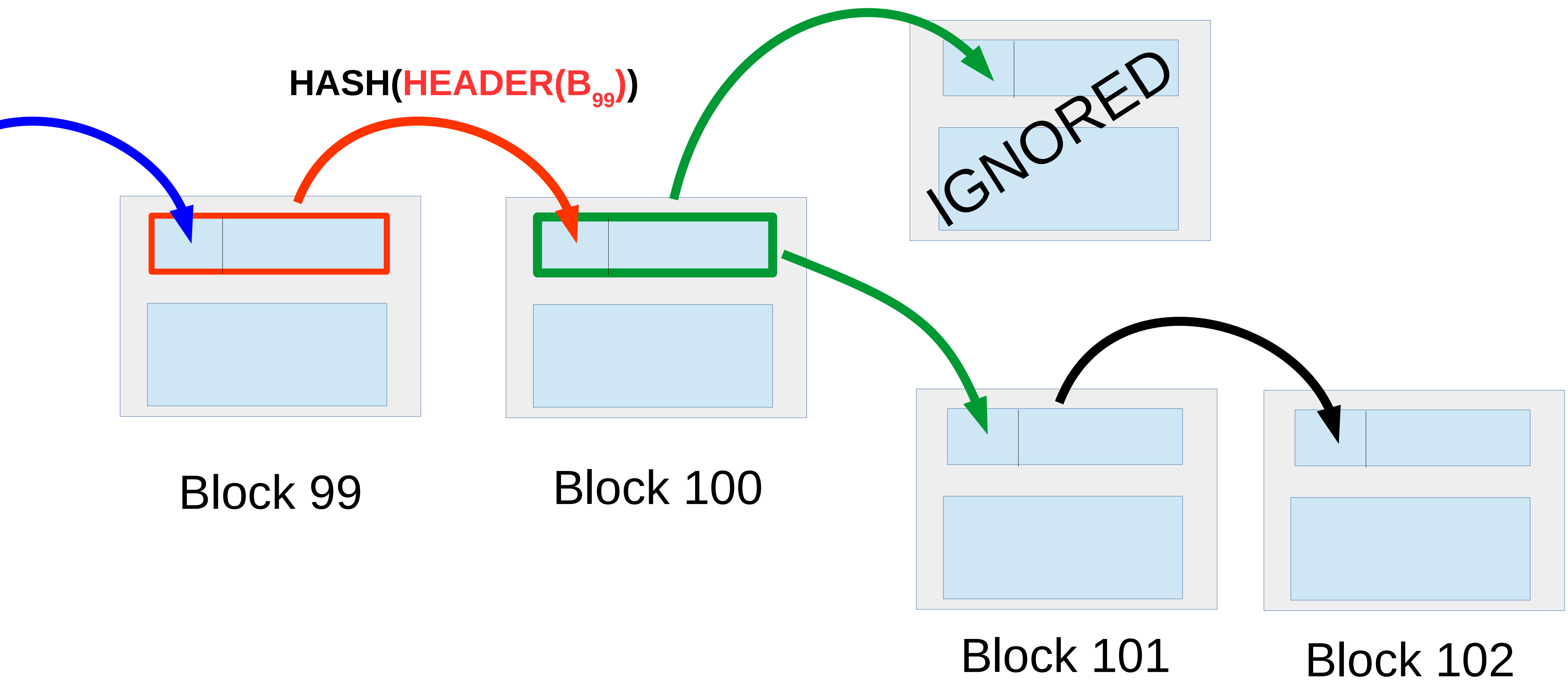
Each block stores the hash of the parent block header.



Blockchain Forks

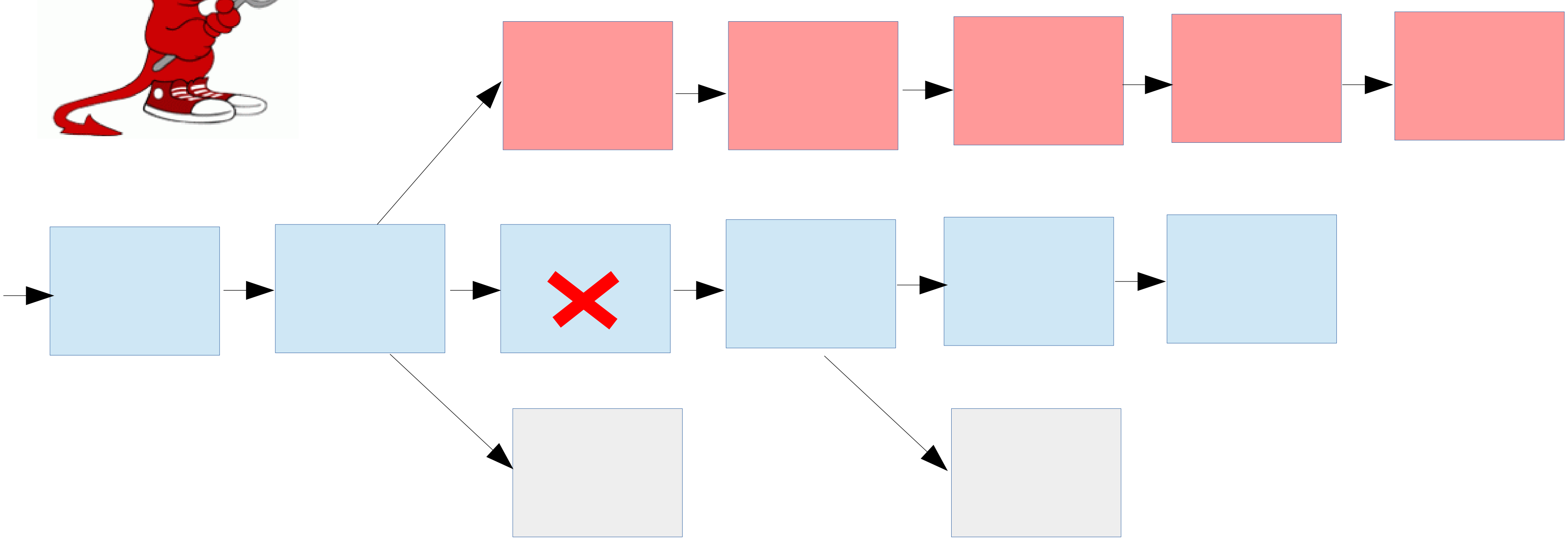
Ties are broken in favor of the longest chain.

Shorter branches are ignored.



Blockchain Forks

An attacker can only rewrite history if he controls more than half* of the computational power of the network.



*some attacks only require about ¼ of the total computational power.

Proof of Work

Creating a new block requires significant amount of work (computational effort).

Malicious peers who want to modify past blocks have to work harder than honest peers who want append blocks.

A block B is only accepted by the network iff

HASH(HEADER(B)) $\leq$ **TARGET**

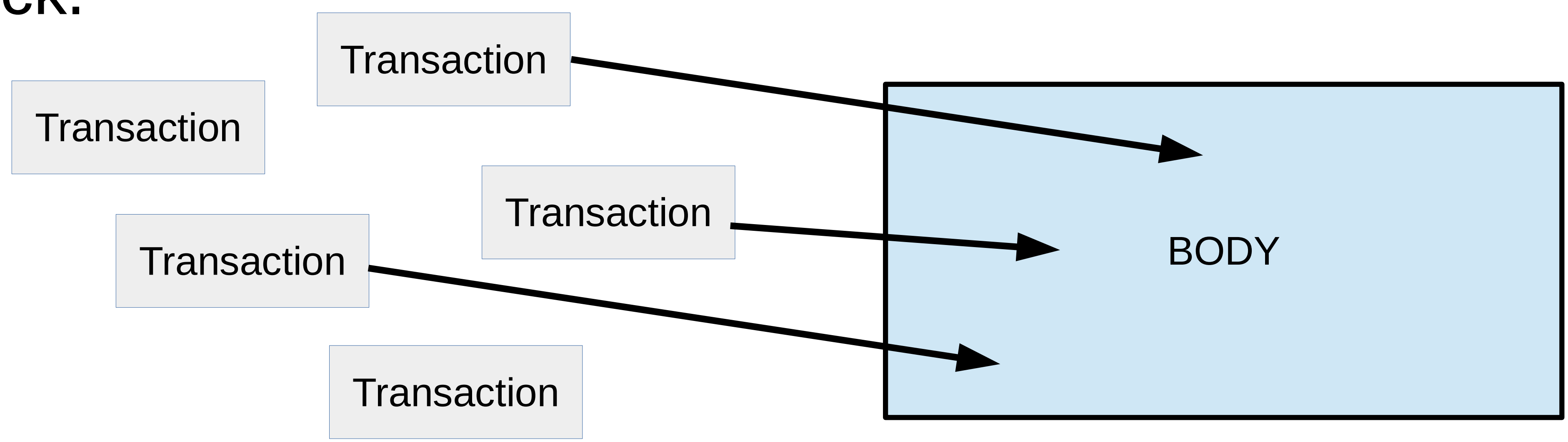
TARGET dynamically updates so that the average time to find a valid block is around 10 minutes

Mining

Mining is the process through which new blocks are created.

Mempool: set of transaction that do not belong to any block.

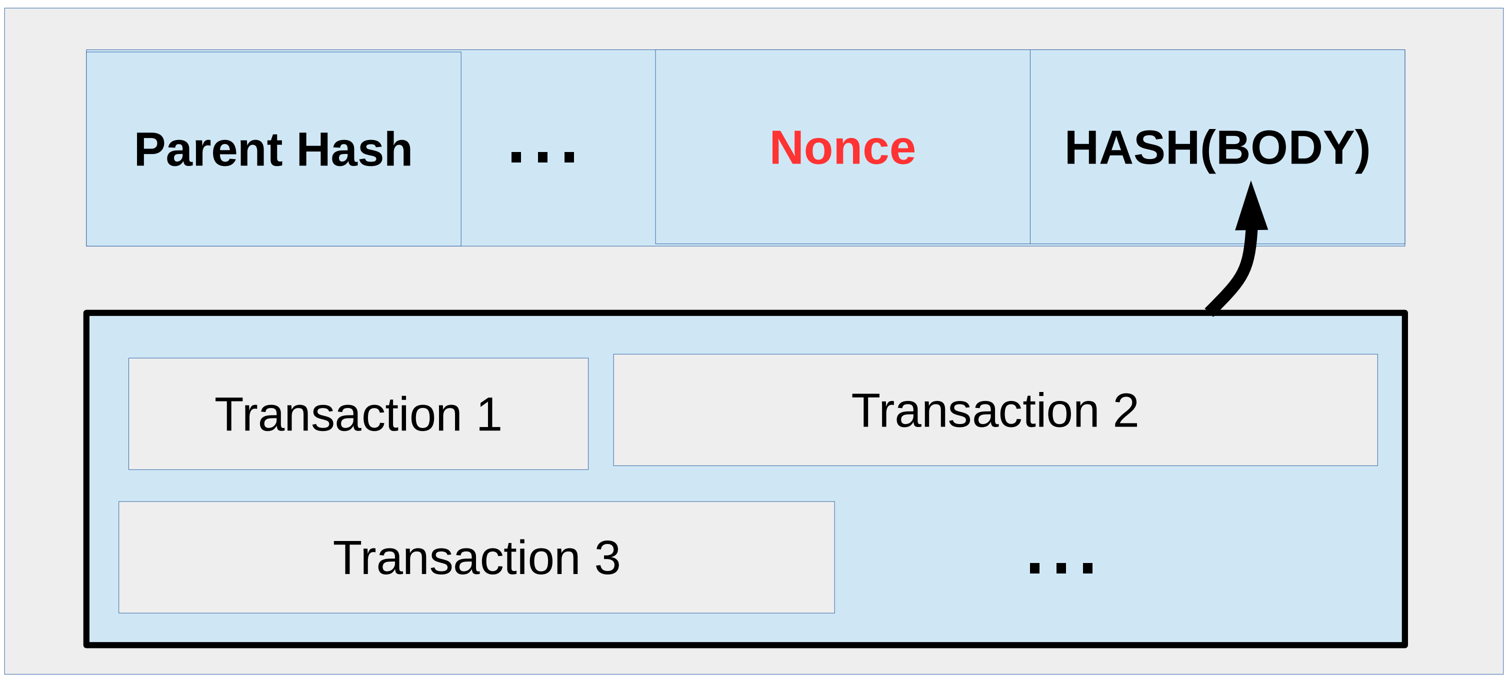
Miners select a (sub)set of mempool transactions, check for their validity, and add them to the body of the new block.



Mining

Miners generate the header for the new block, and they compute the header's hash until

HASH(Header) ≤ **TARGET**



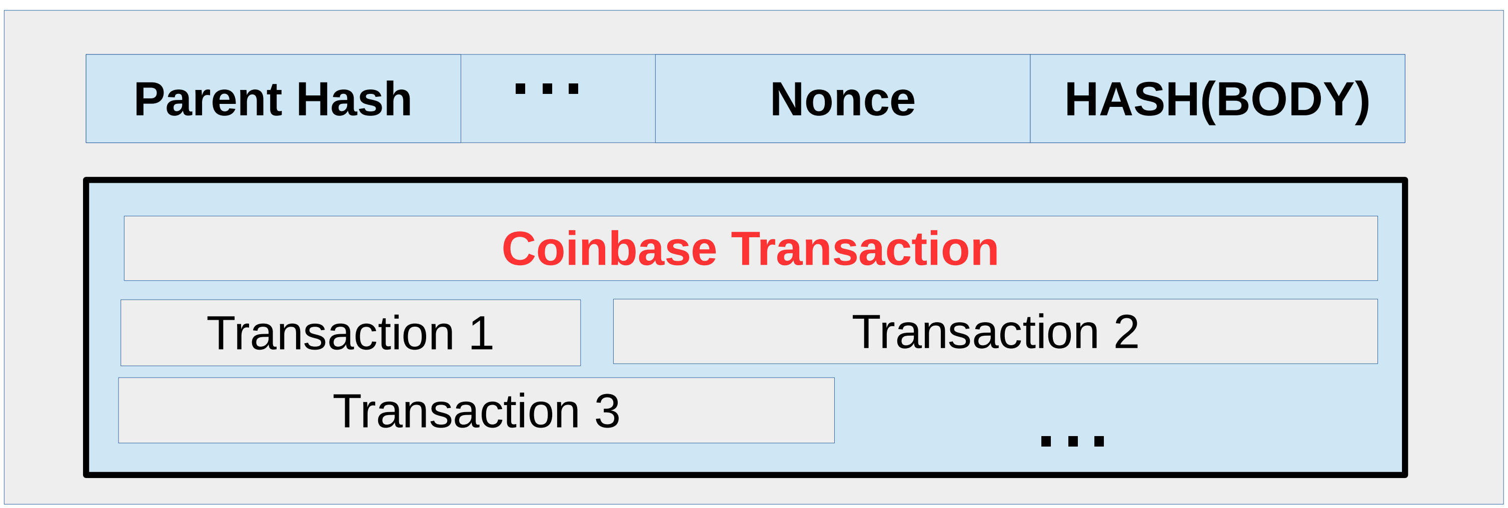
A Nonce field ensures that the header hash changes.

Mining

Miners get to add one additional "**coinbase**" transaction with no inputs.

The output(s) is usually an address owned by the miner.

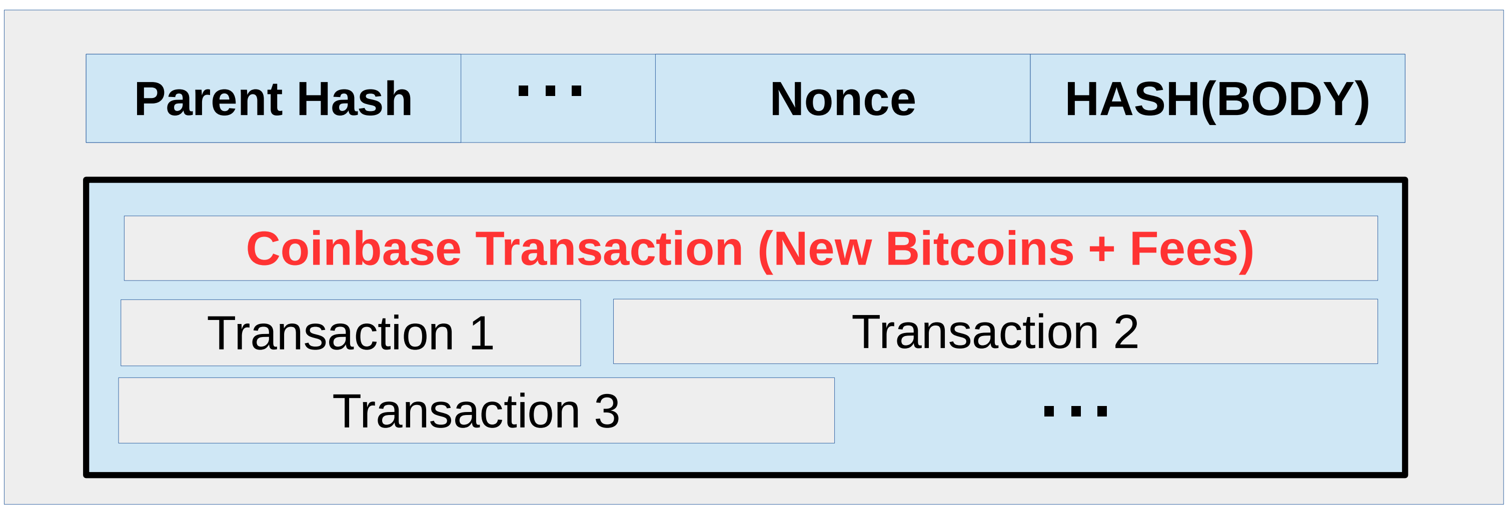
The amount is the sum of a **block subsidy** and of the **transaction fees**.



Mining

Block subsidy: only depends on the block number (i.e., length of the block-chain up to the first block). This is how new bitcoins are created.

Transaction fees: the sum of the transaction fees of the selected transactions.



This is the miner’s reward for “solving” a block.

Mining (Selecting the Transactions)

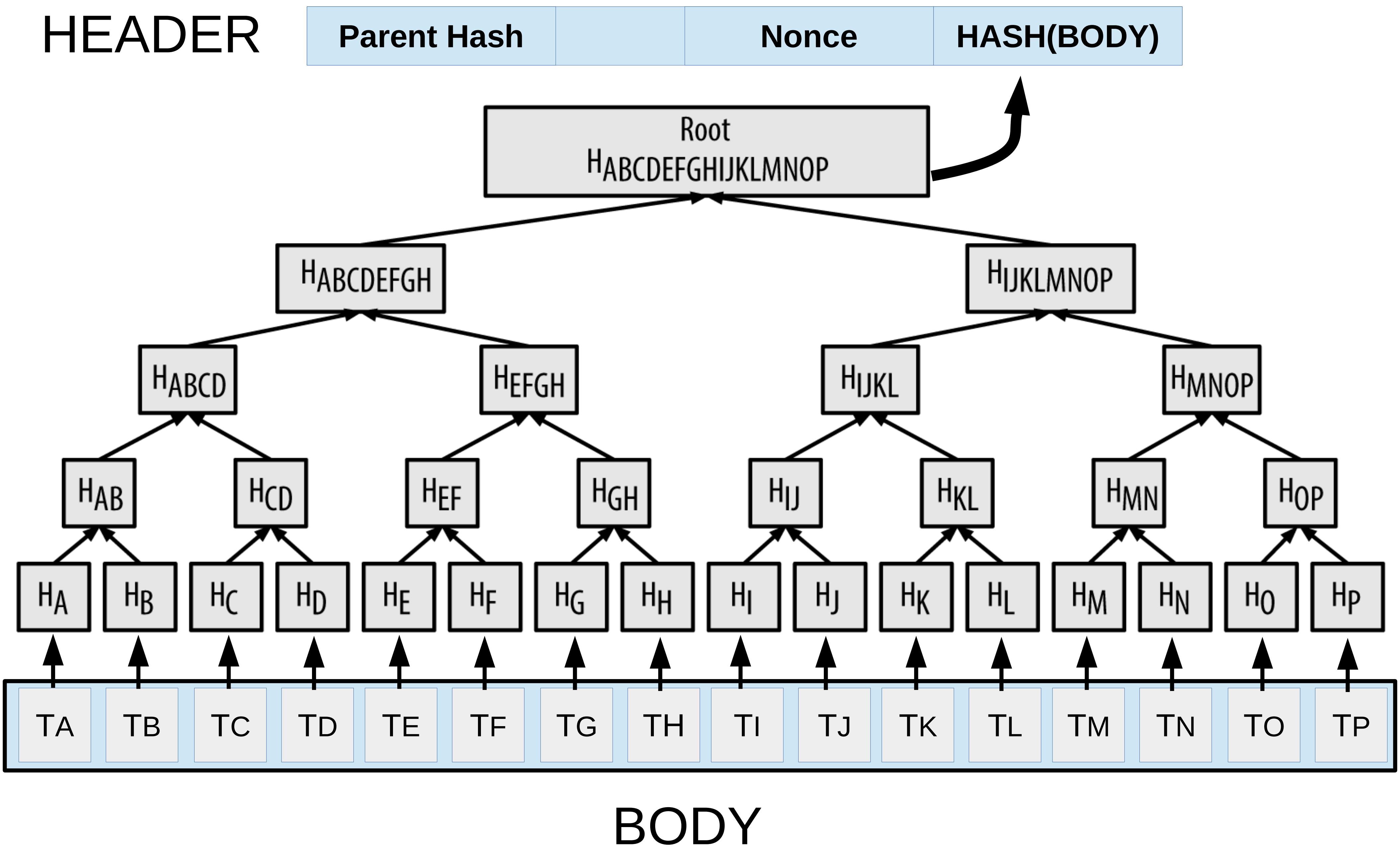
Each block has a maximum “capacity” of 1MB.

Transaction have varying lengths in bytes (depending on the number of inputs/outputs) and different fees.

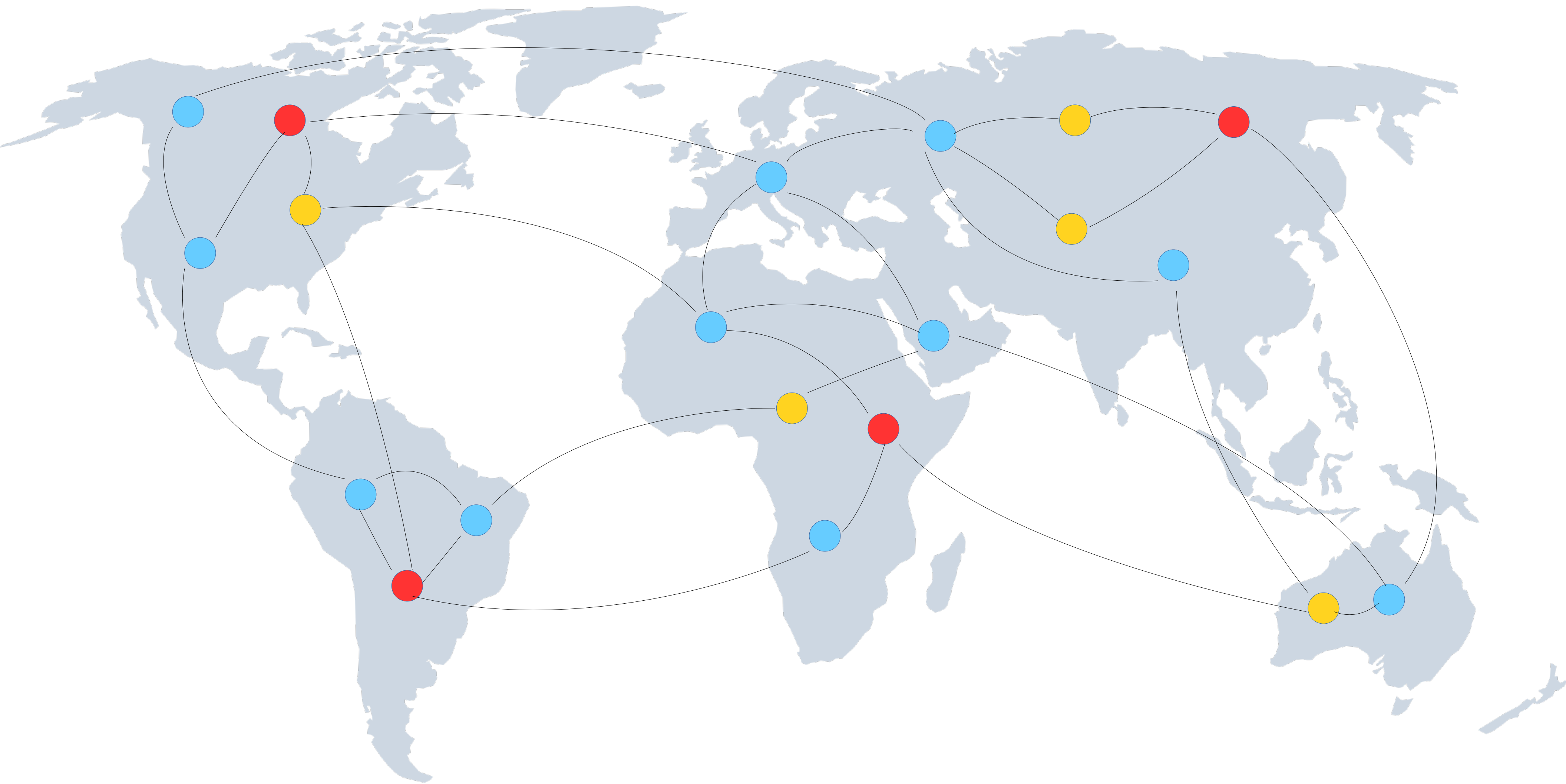
Selecting the a set of transactions to include while maximizing the miner’s revenue is a special case of the Knapsack Problem, a well known **NP-Hard** problem.

The reference miner implementation greedily selects transactions to add (in order of fee/transaction size).

The Body Hash (Merkle Trees)



The Bitcoin Network



Node Types

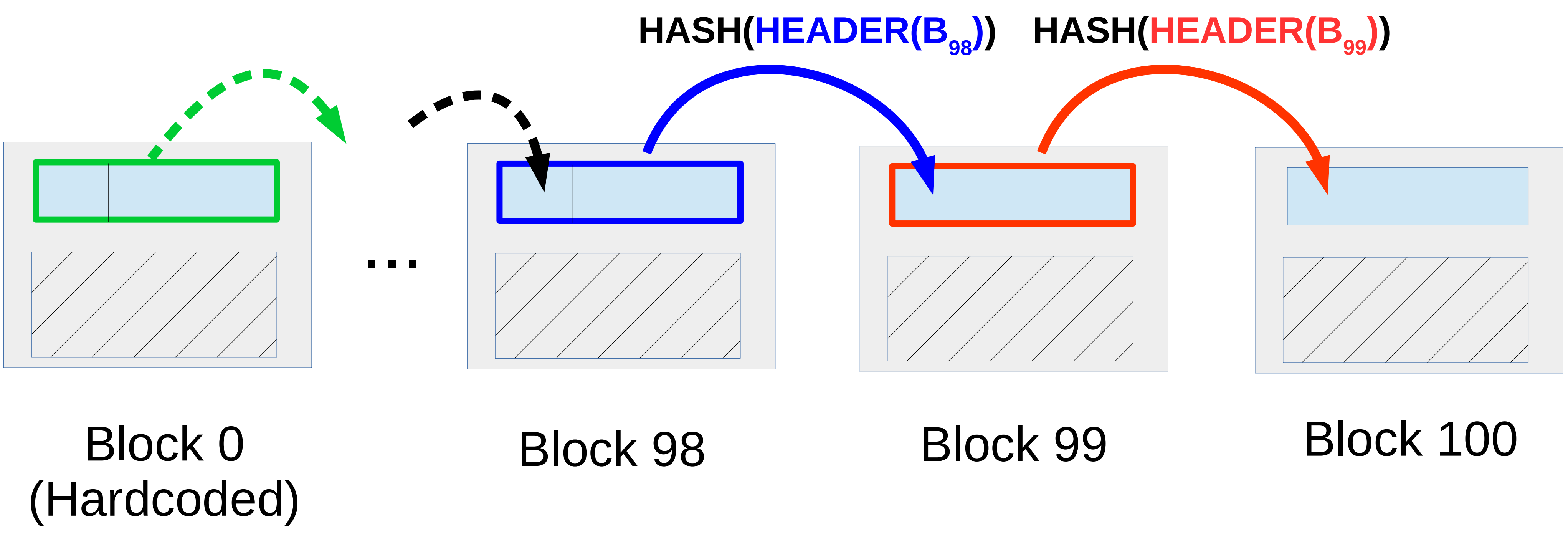
- **Full Nodes:** have a local copy of the entire blockchain. Can <u>directly</u> verify transactions. Can send/receive bitcoins.

- **Lightweight (SPV) nodes:** No local copy of the blockchain. Can <u>indirectly</u> verify transactions. Can send/receive bitcons.

- **Miners:** work to extend the blockchain, mint bitcoins, and get rewarded.

SPV Nodes

- SPV Nodes only download block headers (80B/block).
- Complete knowledge of **which** blocks are in the blockchain
- No knowledge on **what** the block contents (transactions) are.



Proof of Inclusion