Peer-to-peer networking

- In peer-to-peer systems, anything goes: users have no control over which other computer they connect with
- Problem of Trust: no reason to believe that the other side is trustworthy in any sense
  - Assume one has received data, how can one verify the integrity of these data ?

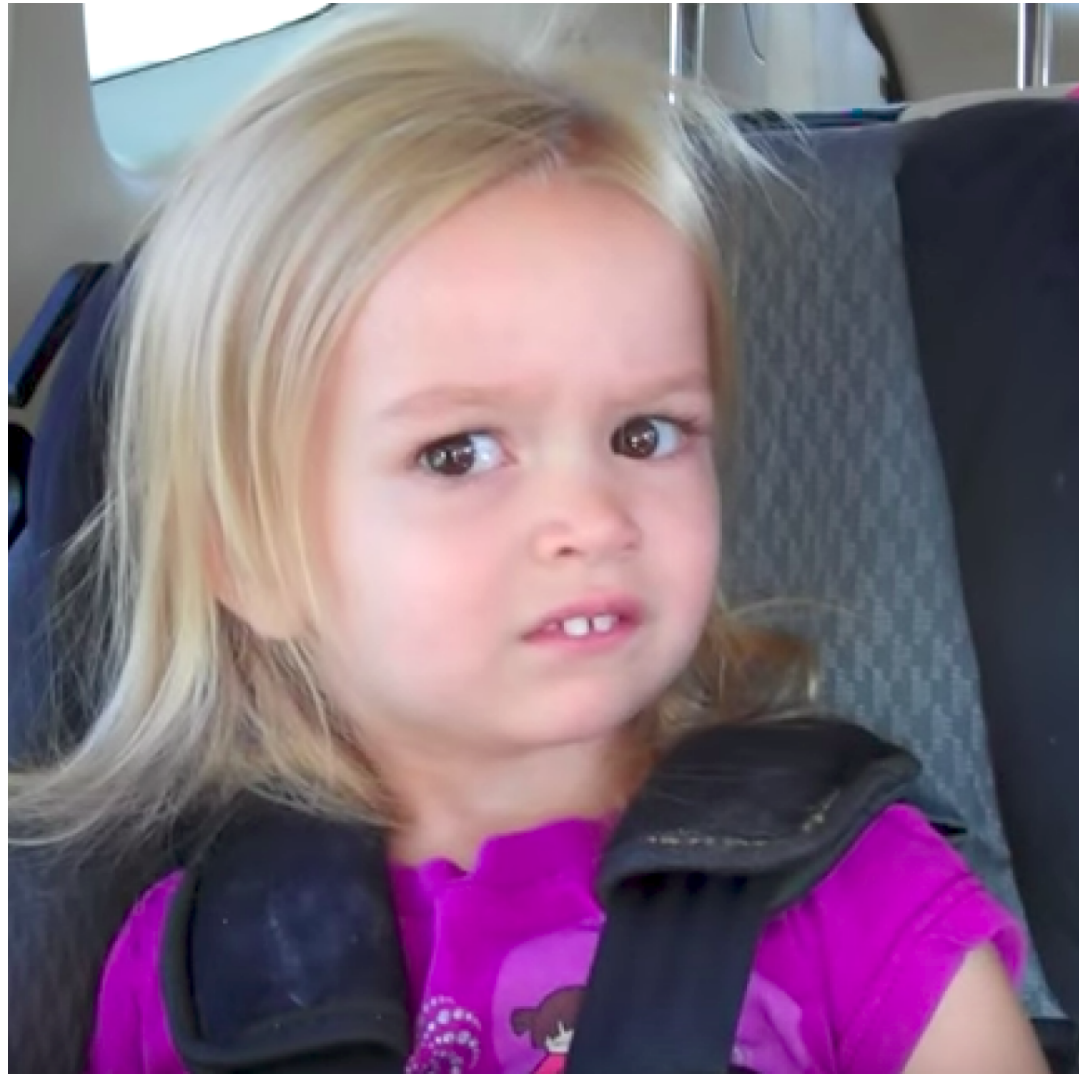
Peer-to-peer networking

- In peer-to-peer systems, anything goes: users have no control over which other computer they connect with
- Problem of Trust: no reason to believe that the other side is trustworthy in any sense
  - Assume one has received data, how can one verify the integrity of these data ?
  - A very costly way would be to download the same data from several other sources as well and compare what was received

Peer-to-peer networking

- In peer-to-peer systems, anything goes: users have no control over which other computer they connect with
- Problem of Trust: no reason to believe that the other side is trustworthy in any sense
  - Assume one has received data, how can one verify the integrity of these data ?
  - A very costly way would be to download the same data from several other sources as well and compare what was received
- A much better way is to use a ... (if someone guess that she/he get +1 for the final exam)

Cryptographic Hash Function

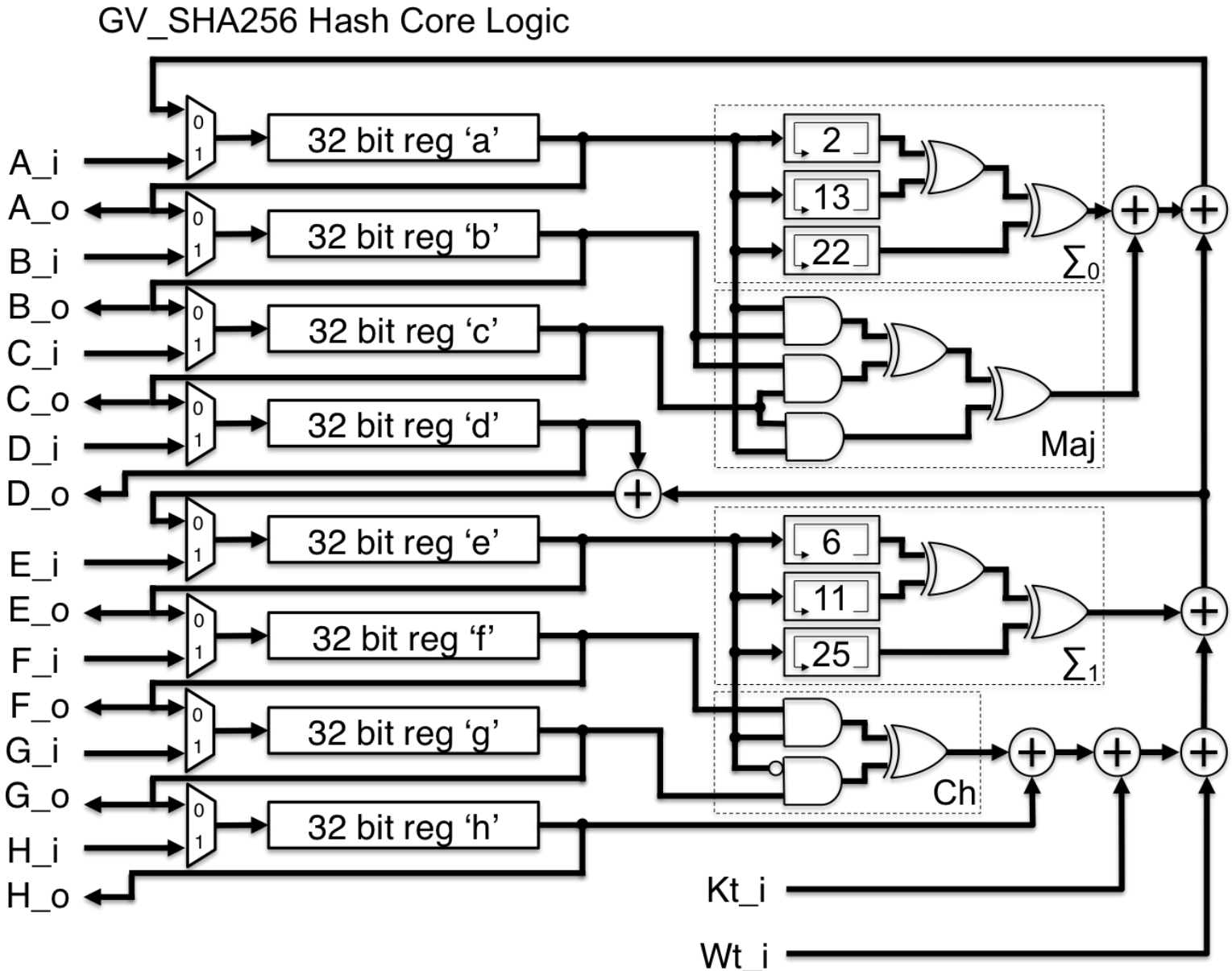


Cryptographic Hash Function [Try here]

$$h(m) = a$$

- A cryptographic hash function is a function which can be fed arbitrary data as input and spits out some fingerprint of the data:
  - the same data must give always the same hash, i.e. the function must be fully deterministic
  - the set of potential input data should be mapped as uniformly as possible to the set of possible hashes
  - the hashes should be easy to compute and much shorter than the data
  - impossible to reverse engineer

Cryptographic Hash Function: SHA-256



Cryptographic Hash Function: Example

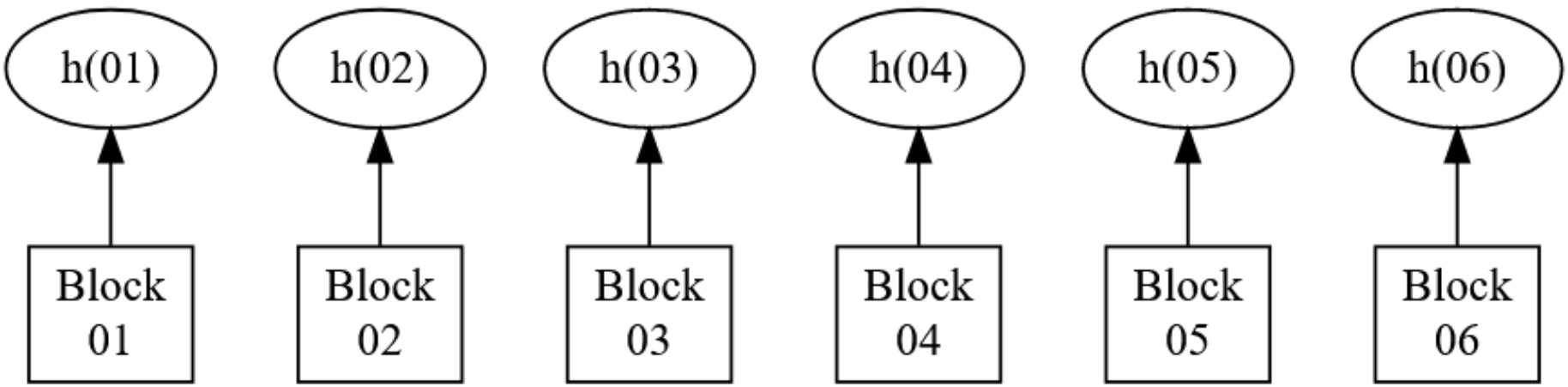
```
import hashlib

# prints an MD5 hash
def print_with_hash(data):
  print (data, " | MD5 hash: ", hashlib.md5(data).hexdigest())

print_with_hash(b'The world is big.')
print_with_hash(b'The word is big. ')
```

Output :

```
b'The world is big.'  | MD5 hash:  fe3c1c076552b3e6c61a2d3d4b288340
b'The word is big. '  | MD5 hash:  f78c604d621c6b11436c95c5152b2a3d
```

First Idea: Hash Lists

- In peer-to-peer networks, some blocks of the same file are provided to you by one peer, other blocks by another (entire movies are splitted in small chunk of data)
- Hence, it's ideal to know a list of the correct hashes of all the data blocks upfront

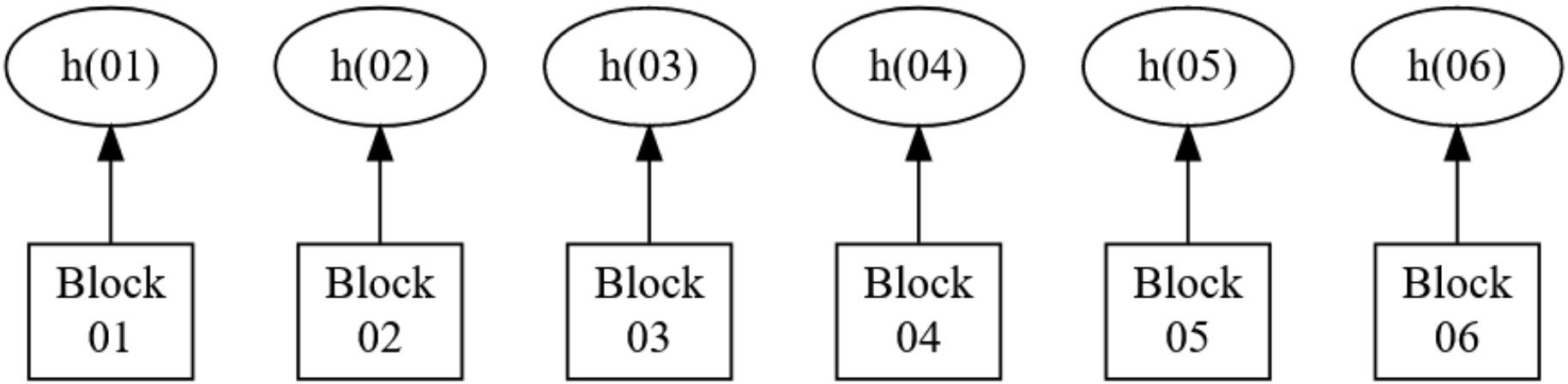


- That's for example how Bittorrent works. A `.torrent` file contains a list of block hashes, and your torrent software validates incoming data against these hashes

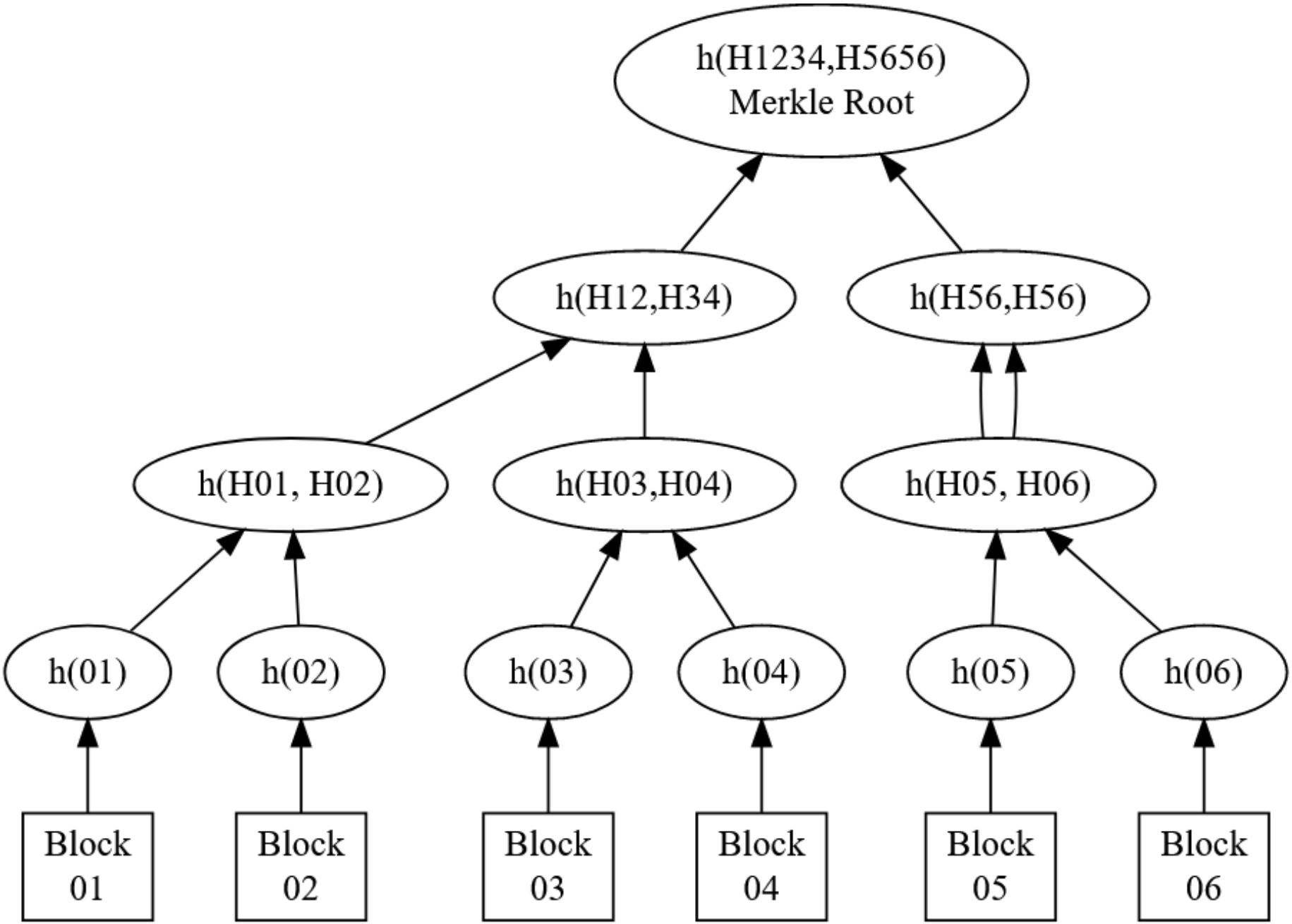
Better than Hash Lists, Merkle trees!

- The first to describe hash trees was Ralph Merkle (1980)
- The idea is simple: For any pair of hashes, join their hashes together and hash over these hash data again. Do so until you reach one single top node, which is called the Merkle root
- The Merkle root hash is the unique signature of the whole file

Better than lists, Merkle trees!



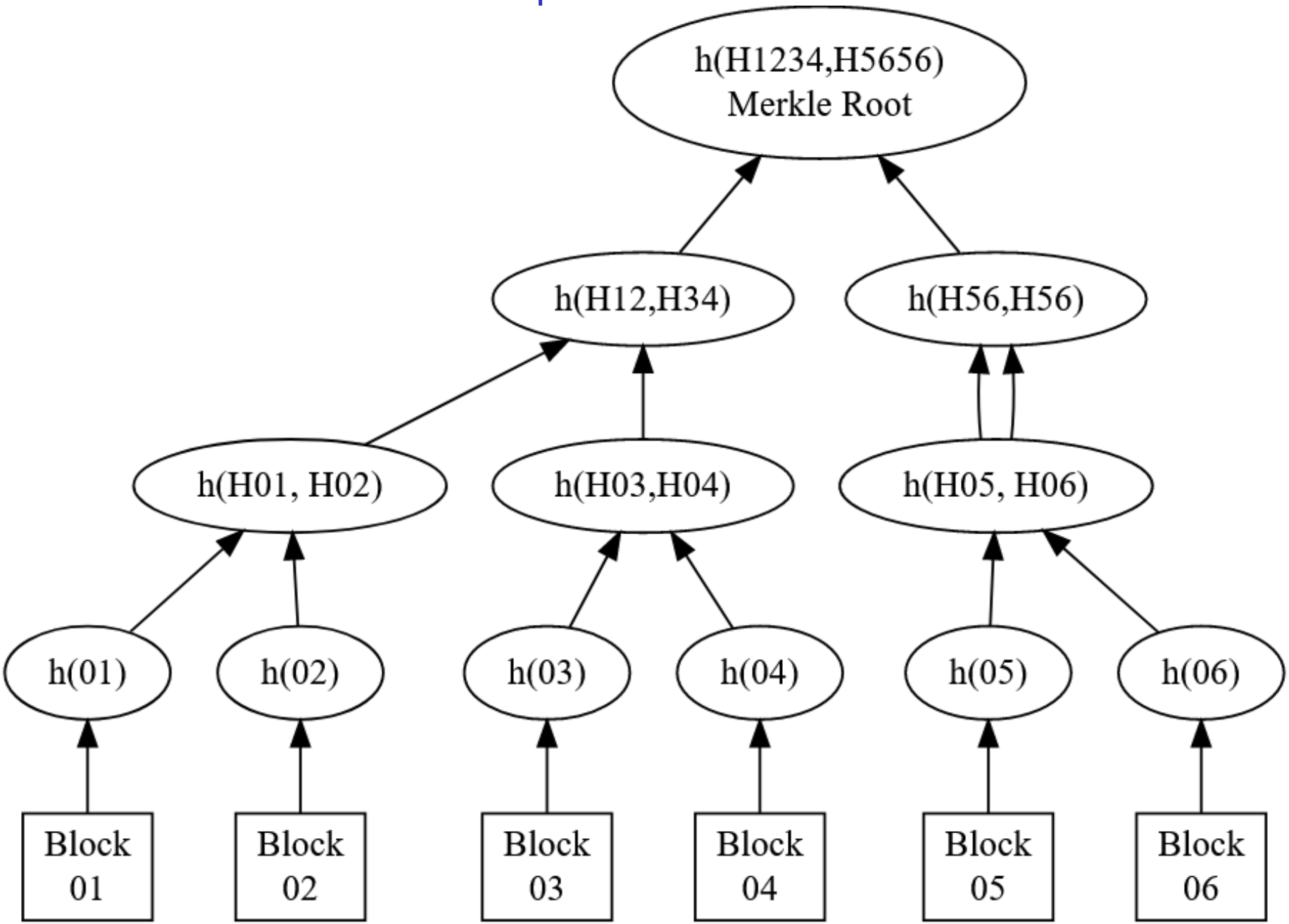
Better than lists, Merkle trees!



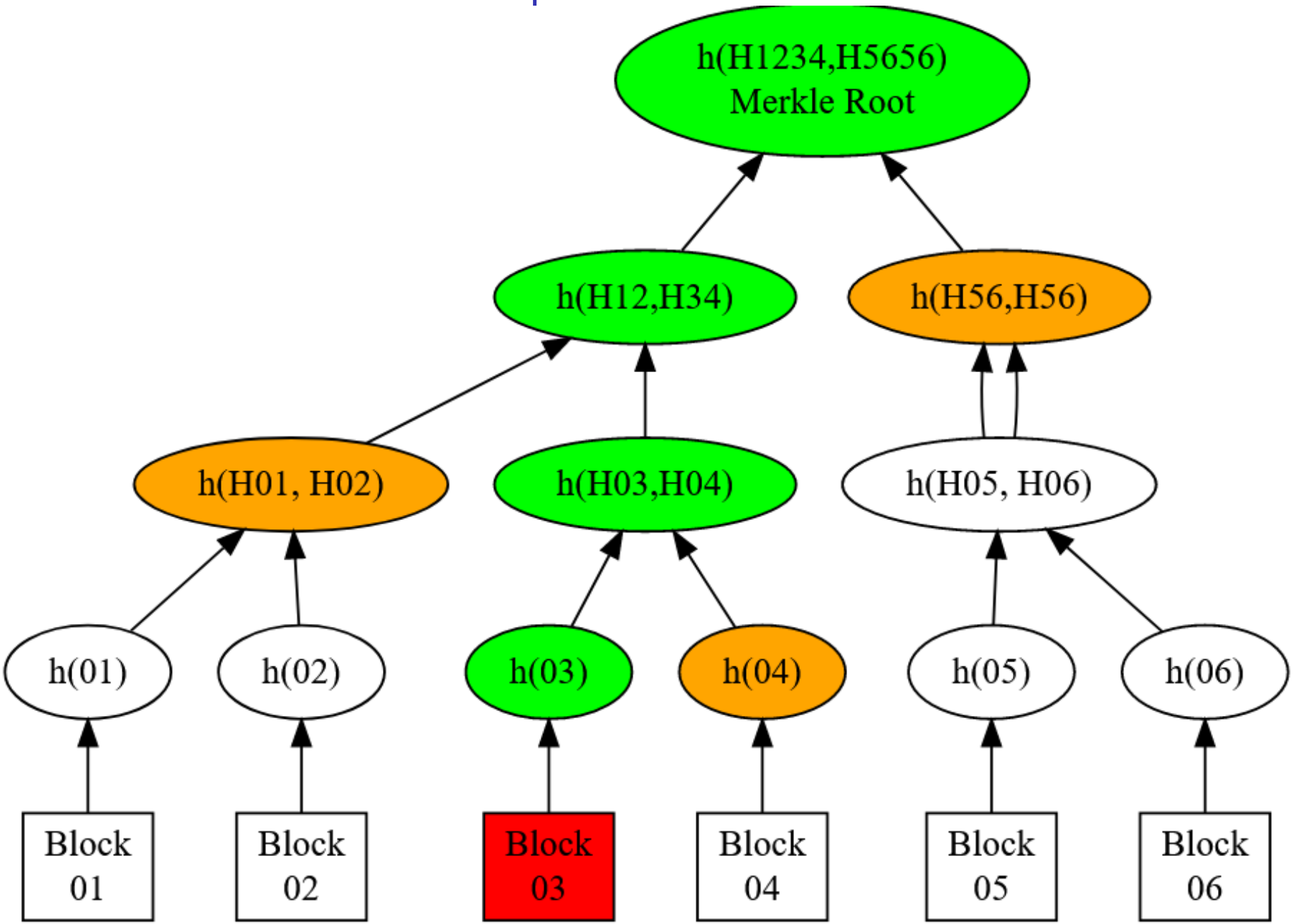
Merkle trees and Merkle proof

- Change even a bit in the file, and the Merkle root will change
- Merkle proof: Given the Merkle root of a file, it is very inexpensive to verify whether a given piece of data is a valid data block
- Assume that you know the file's Merkle root from a trusted source:
  - You've just received block 03 but don't know any of the other blocks
  - How much data do you need to have proof that the block which you received is genuine and an element of the file ? Does this data has to be trusted?

Merkle trees and Merkle proof



Merkle trees and Merkle proof



Merkle trees for daily use

- Dropbox or GIT: make sure that 2 files are the same when uploading or downloading (corruption happens unintentionally) and spotting where the corrupted data is
- BitTorrent: (i) verify the integrity of the data from an unsafe source (ii) make sure that a chunk of data coming from a unsafe source matches the one you want to download (iii) select only the uncorrupted data from an unsafe node.
- Trading firm: storing the history of all transactions (you compute new Merkle trees for each new transaction)

$\rightarrow$ Learn more technical details here: great blog post

Merkle roots in the Blockchain

- Merkle trees are central in Blockchain (more on this later, wait a few slides)
- They're of crucial importance because they enable the efficient verification of a single payment record in a sea of data without having to download all the other payments.
- To prove that a payment has actually happened, it suffices to have the transaction and obtain from the network the (few) hashes for completing the Merkle proof all the way up to the trusted Merkle root.

Second Ingredient: Cryptography 1-0-1

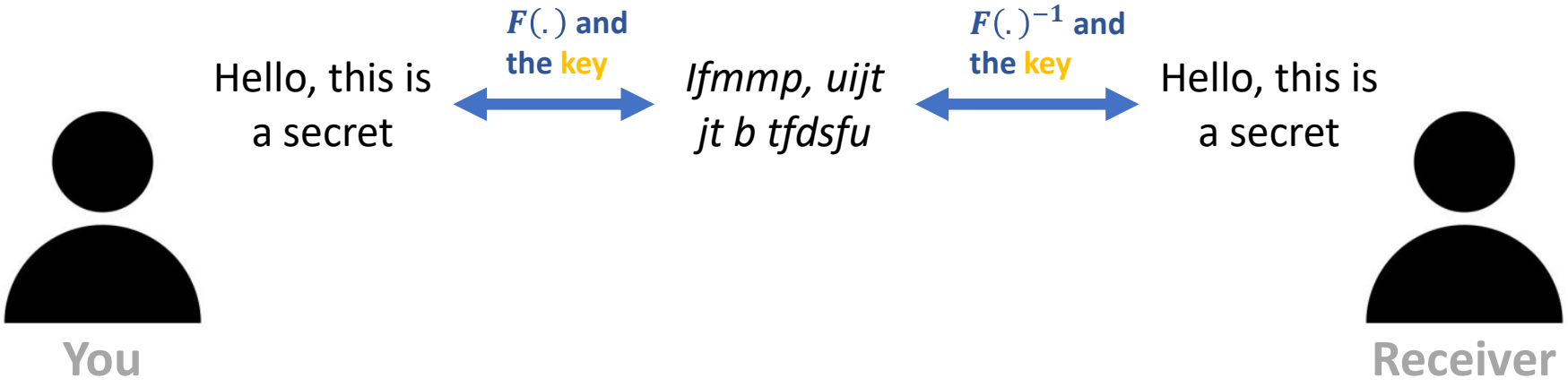
Cryptography: symmetric key

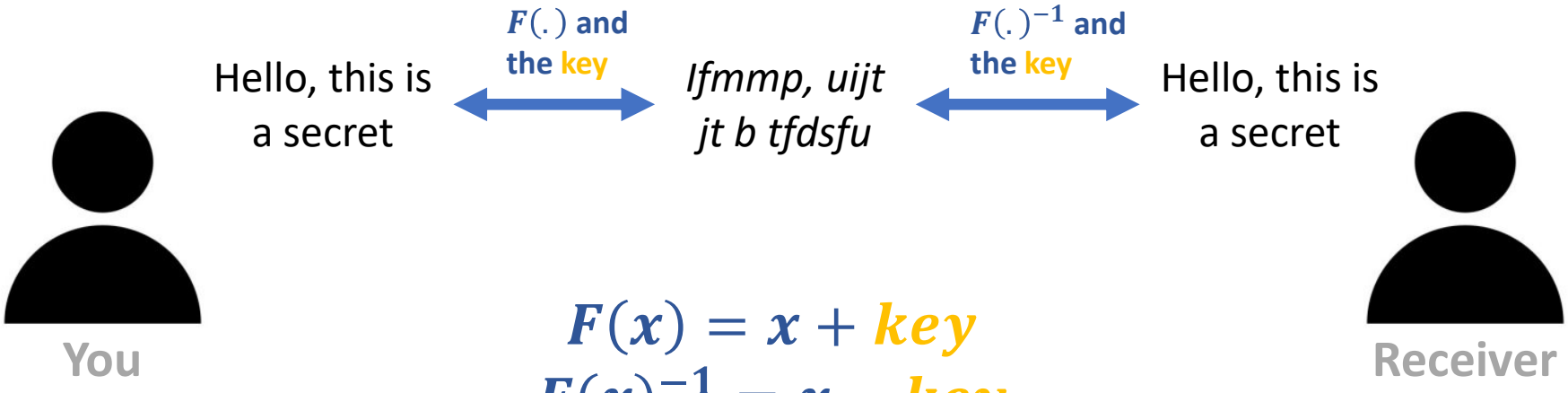
- Already 2000 years ago people were using cryptography to transmit military information
- The idea of a symmetric encryption cipher is simple :

$$\text{Crypted Text} = f^{key}(\text{Text})$$

- Only a person knowing the key should be able to retrieve the plain text without unreasonable effort

$$\text{Text} = (f^{key})^{-1}(\text{Crypted Text})$$


Hello, this is a secret
$F(.)$ and the key
Ifmmp, uijt jt b tfdsfu
$F(.)^{-1}$ and the key
Hello, this is a secret
You
Receiver



$$F(x) = x + key$$
$$F(x)^{-1} = x - key$$

and $key = 1$

Cryptography: symmetric key

- Julius Caesar used to shift all letters of the alphabet by a certain offset, writing c instead of a, d instead of b, etc.

Cryptography: symmetric key

- Julius Caesar used to shift all letters of the alphabet by a certain offset, writing c instead of a, d instead of b, etc.
- Can you break this code?

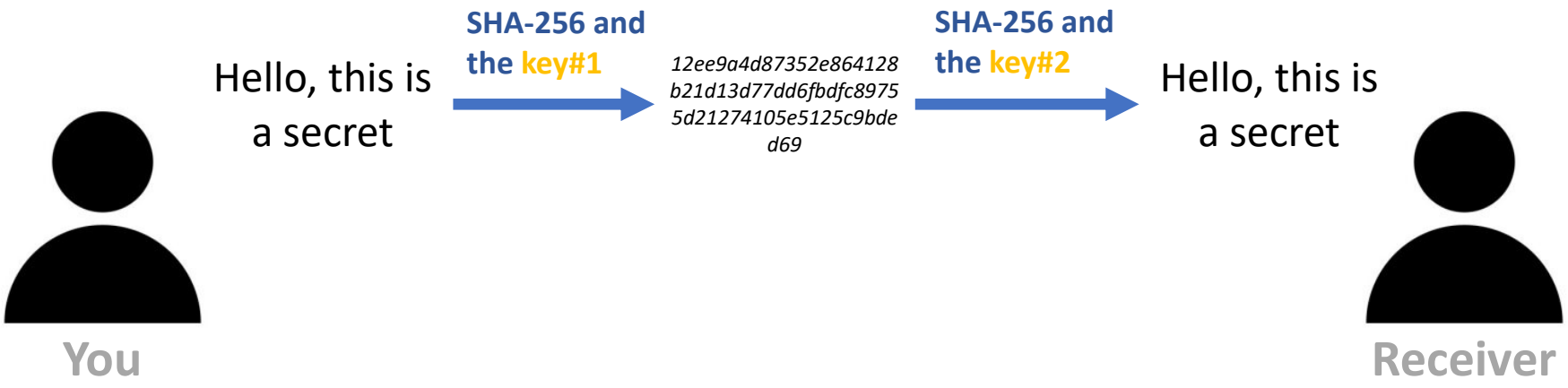
```
Tevsec Mkockb kvgkic econ oxmbizdsyx dy zbydomd rsc
zbsfkdo mywwexsmkdsyx. Dro Mkockb mszrob coowc k qyyn
snok kd psbcd. Led sd bokvvi sc xyd. Dro uoi czkmo sc
vswsdon dy (x-1) grobo x sc dro xewlob yp voddobc sx
dro kvzrklod. Drsc voddob, pyb ohkwzvo, gkc oxmbizdon
wkzzsxq k dy u, cy mryycsxq k dbkxcvkdsyx yppcod yp
dox. S qeocc iye pyexn yed li dbisxq kvv zyccslvo
uoic, nsn iye?
```

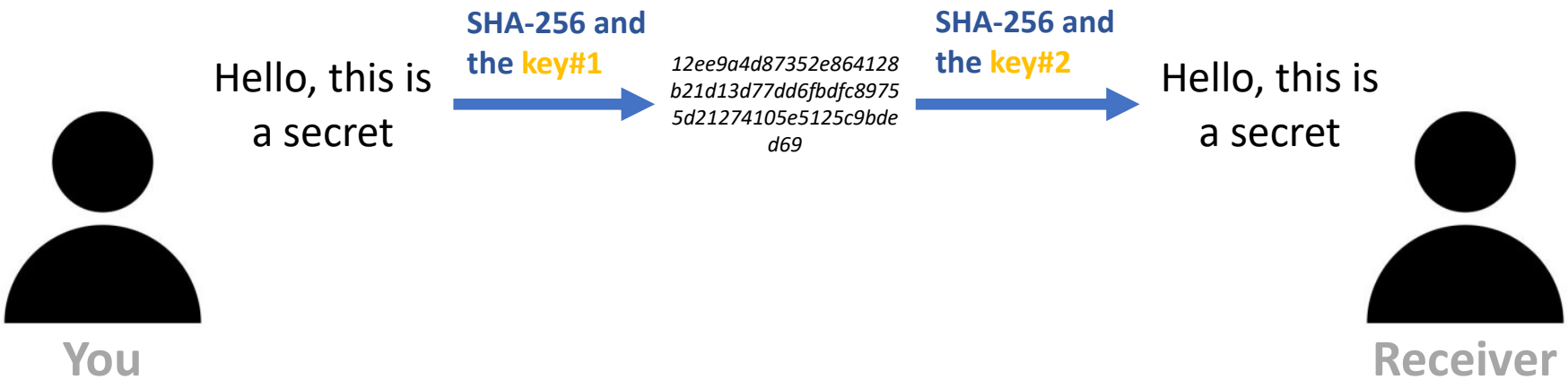
Cryptography: symmetric key

- Modern symmetric key ciphers can be very powerful, but
- For the receiver to be able to read the message, its encryption key must already have been communicated via some secure channel to the receiver
- Solution (Diffie and Hellman, 1976): two keys, a public and a private

Cryptography: asymmetric key

- Distribute the public key to the general public, and make sure everyone knows your public key. Keep the secret key private.
- Anyone wanting to send an encrypted message to you can encrypt it using your public key
- The message can now only be decrypted by the secret key (see here for a great tutorial)


Hello, this is a secret
SHA-256 and the key#1
12ee9a4d87352e864128b21d13d77dd6fbdfc89755d21274105e5125c9bded69
SHA-256 and the key#2
Hello, this is a secret
You
Receiver



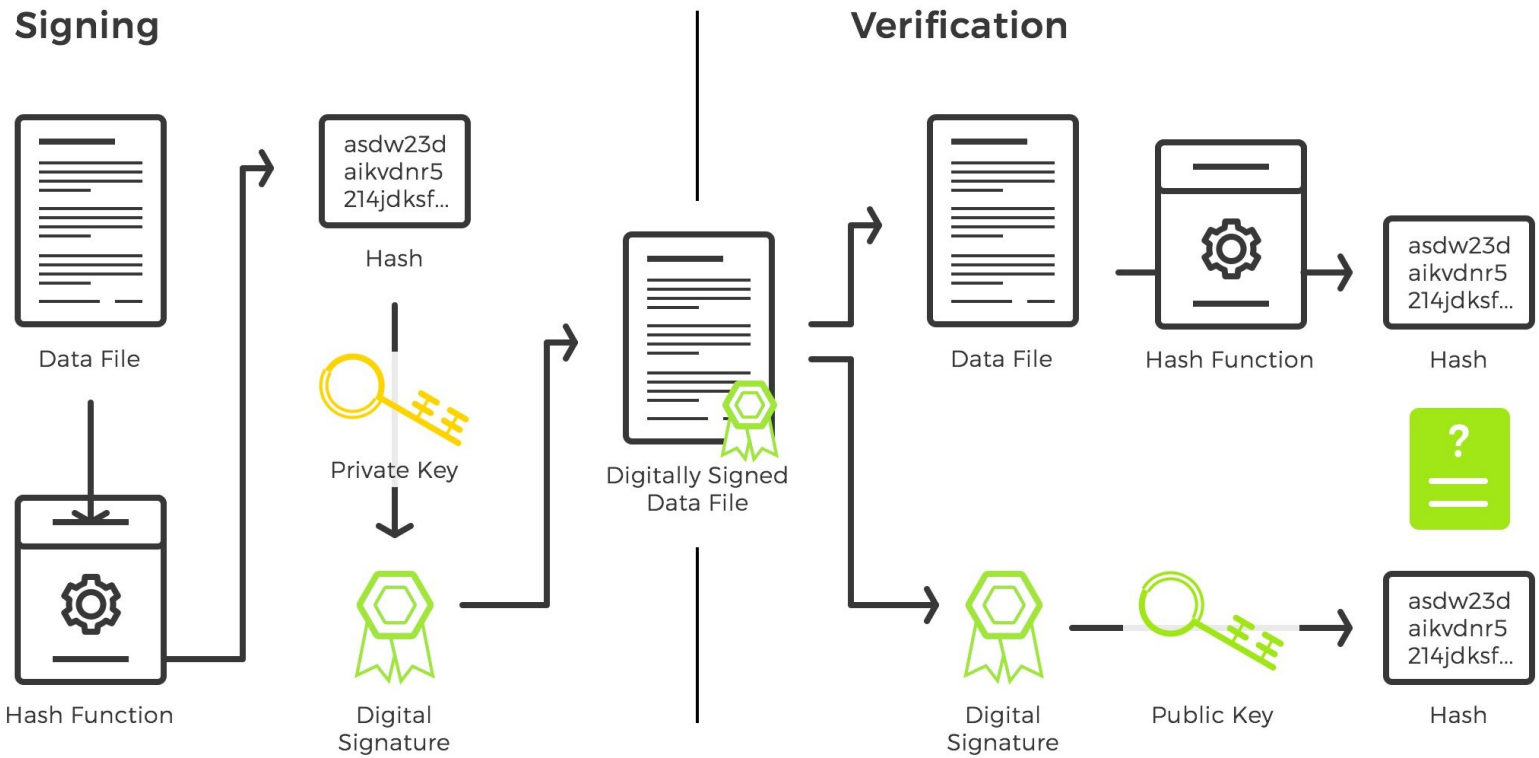
- You use the public key (key#1) of the receiver to encode your message
- Only the receiver can read it (with his private key#2)

Digital signatures

- Digital signatures $\approx$ asymmetric key encryption + hash function
- Canonical way to digitally sign un-encrypted data by add a digital signature at the end of an un-encrypted message

Digital signatures

- Digital signatures $\approx$ asymmetric key encryption + hash function
- Canonical way to digitally sign un-encrypted data by add a digital signature at the end of an un-encrypted message



Digital signatures

- Naively, we might think that digital signatures provide a natural way for creating digital coins that can be passed on from one to another

Digital signatures

- Naively, we might think that digital signatures provide a natural way for creating digital coins that can be passed on from one to another
- A coin would be a file which carries the digital signature of the payer agreeing to give the coin to the payee (identified by a public key).
- The digital signature indeed proves that the payee rightfully received the coin from the payer. The payee could spend the coin by amending his/her own signature and the public key of the next receiver

Digital signatures

- Naively, we might think that digital signatures provide a natural way for creating digital coins that can be passed on from one to another
- A coin would be a file which carries the digital signature of the payer agreeing to give the coin to the payee (identified by a public key).
- The digital signature indeed proves that the payee rightfully received the coin from the payer. The payee could spend the coin by amending his/her own signature and the public key of the next receiver

Digital signatures

- Naively, we might think that digital signatures provide a natural way for creating digital coins that can be passed on from one to another
- A coin would be a file which carries the digital signature of the payer agreeing to give the coin to the payee (identified by a public key).
- The digital signature indeed proves that the payee rightfully received the coin from the payer. The payee could spend the coin by amending his/her own signature and the public key of the next receiver
- However, this native currency doesn't work:

Digital signatures

- Naively, we might think that digital signatures provide a natural way for creating digital coins that can be passed on from one to another
- A coin would be a file which carries the digital signature of the payer agreeing to give the coin to the payee (identified by a public key).
- The digital signature indeed proves that the payee rightfully received the coin from the payer. The payee could spend the coin by amending his/her own signature and the public key of the next receiver
- However, this native currency doesn't work: the initial owner of the coin can always retain a copy of the coin before digitally signing it over to the payee

$\rightarrow$ Problem known as the double spending problem.

Digital signatures

- Naively, we might think that digital signatures provide a natural way for creating digital coins that can be passed on from one to another
- A coin would be a file which carries the digital signature of the payer agreeing to give the coin to the payee (identified by a public key).
- The digital signature indeed proves that the payee rightfully received the coin from the payer. The payee could spend the coin by amending his/her own signature and the public key of the next receiver
- However, this native currency doesn't work: the initial owner of the coin can always retain a copy of the coin before digitally signing it over to the payee

$\rightarrow$ Problem known as the double spending problem. A very smart person solved it 14 years ago.


Secret Ingredient:
Genius

The first Blockchain

- On 31st of October 2008, an anonymous developer posted unter the pseudonym Satoshi Nakamoto a paper on a cryptographic mailing list
- It describes a decentralized currency, Bitcoin
- The paper (available on BBoard) remains the single best source to understand the idea behind Bitcoin, and it's not very technical at all.
- We've already learned about Merkle trees, Merkle proofs and digital signatures, so you can understand everything that Satoshi is talking about.

Satoshi Nakamoto mystery

Dorian Nakamoto

Craig Wright

Satoshi Nakamoto

Half Finney

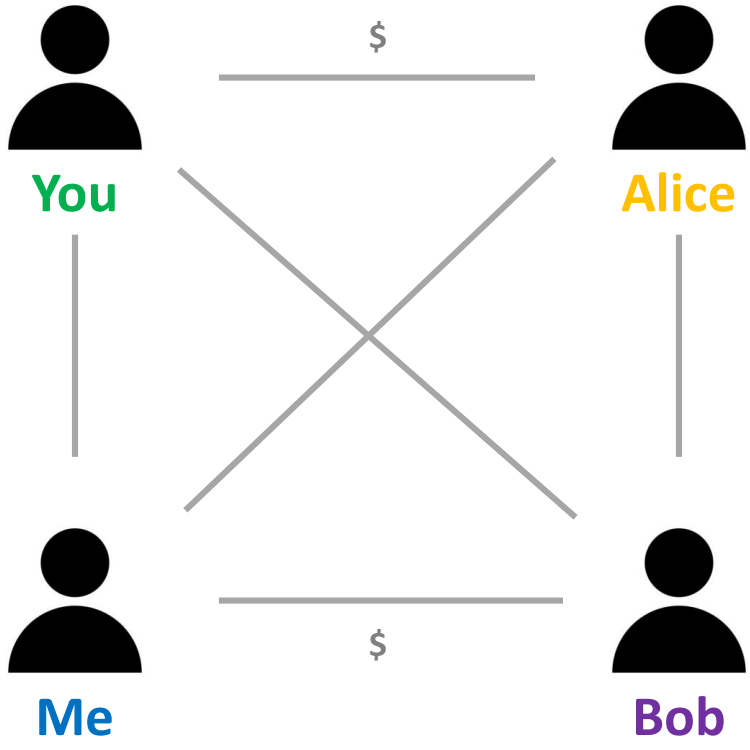
Nick Szabo

Back to the basics: What do we need?

- A digital currency that is both:

Decentralized
Secured and Immutable
Scalable

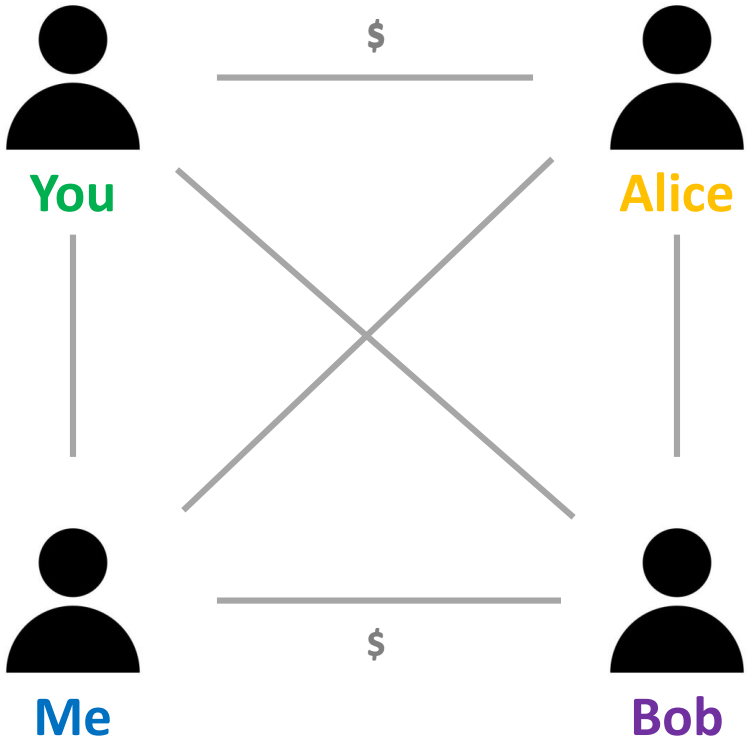
Starting point: we want to pay and be paid



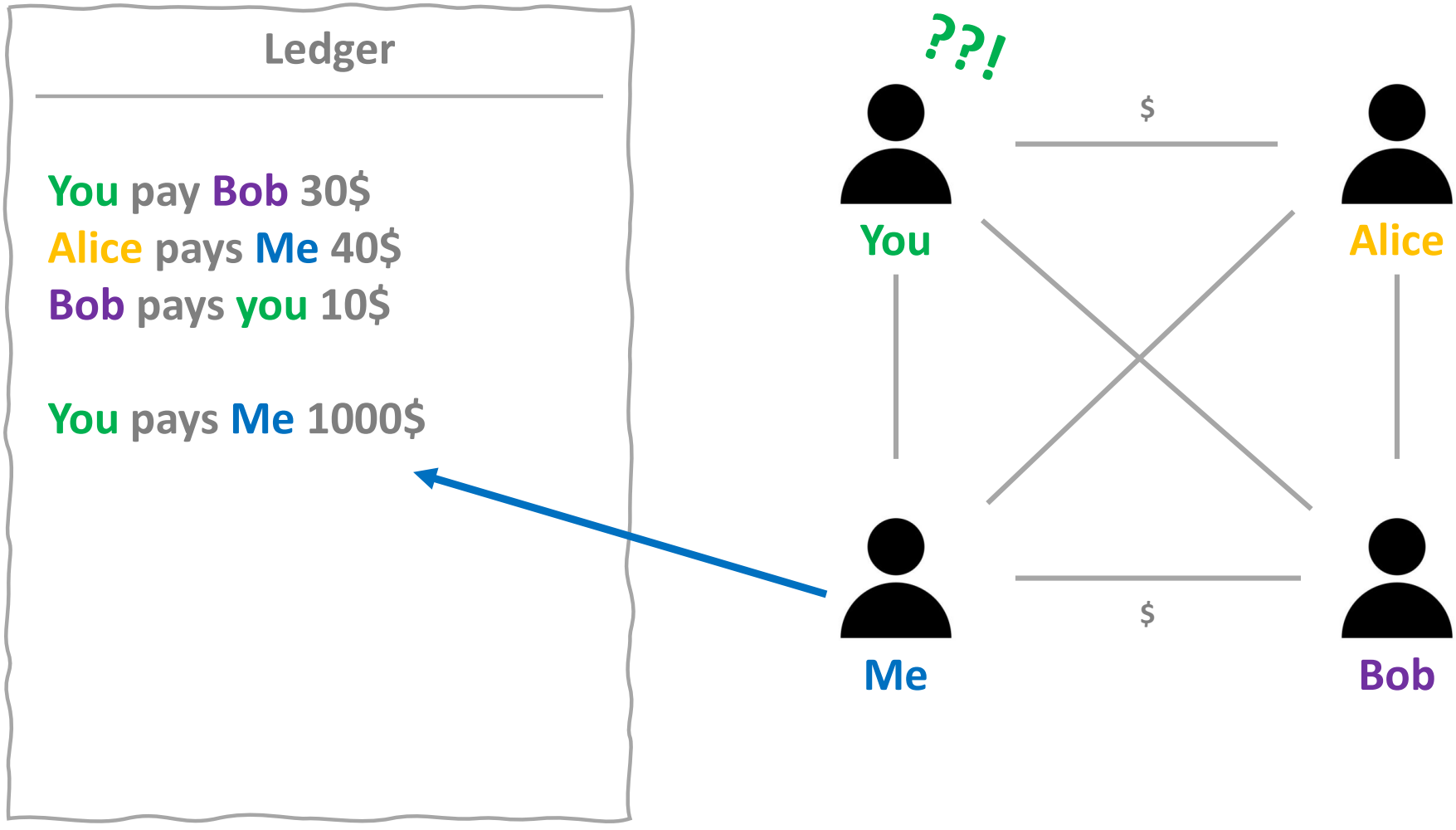
We create a **ledger** to keep track of all transactions

Ledger

You pay **Bob** 30$
Alice pays **Me** 40$
Bob pays **you** 10$



What if someone tries to fool you?

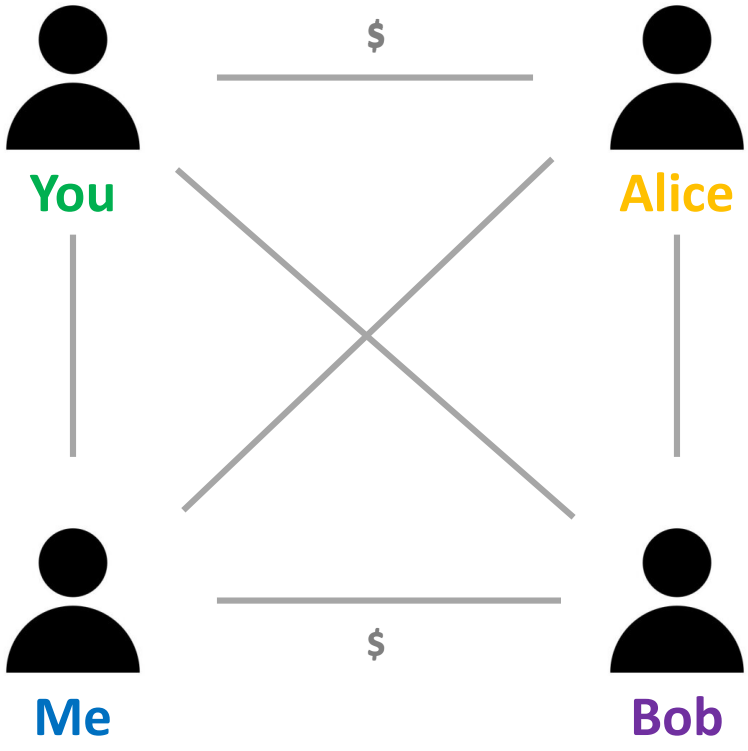


Solution: signatures - but can be reproduced (?)

Ledger

You pay **Bob** 30$ *You*
Alice pays **Me** 40$ *Alice*
Bob pays **you** 10$ *Bob*

Signatures!

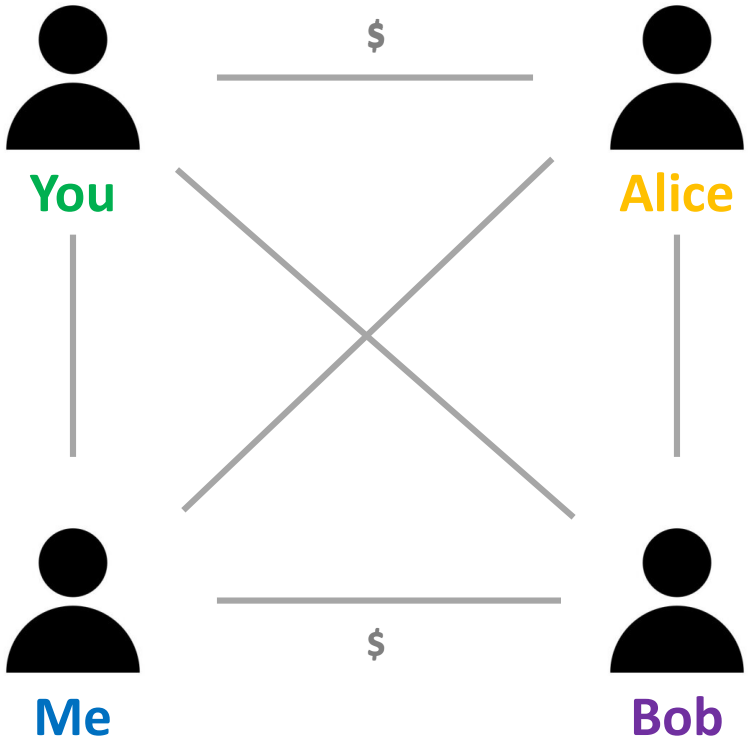


Better solution: digital signatures using 2 keys

Ledger

You pay **Bob** 30$ `f2d81a2c9t3r`
Alice pays **Me** 40$ `ca978112ca1`
Bob pays **you** 10$ `3e23e8160039`

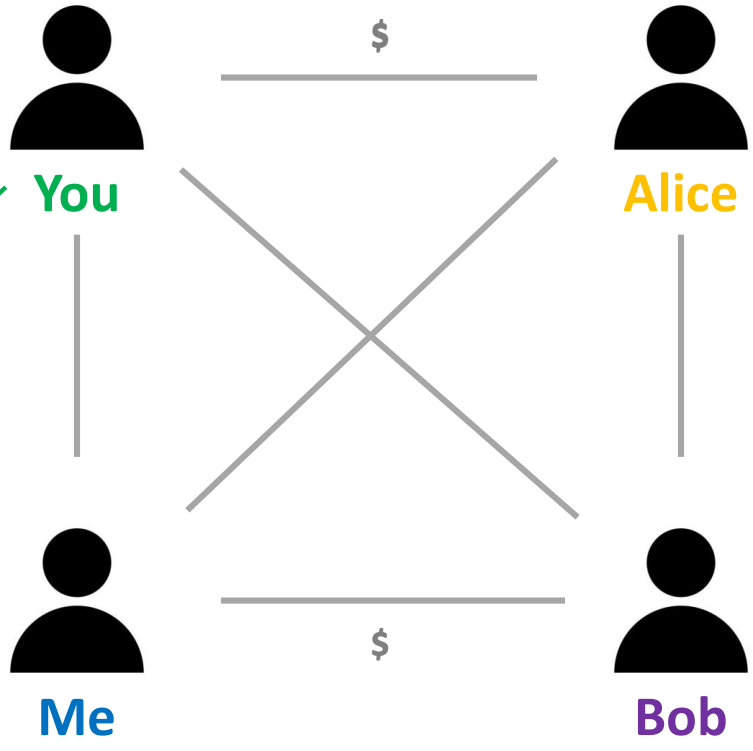
Digital signatures!



Best solution: digital signatures using 2 keys + unique ID

Ledger

You pay **Bob** 30$ `f2d81a2c9t3r`
Alice pays **Me** 40$ `ca978112ca1`
Bob pays **you** 10$ `3e23e8160039`
Bob pays **you** 10$ `3e23e8160039`
Bob pays **you** 10$ `3e23e8160039`

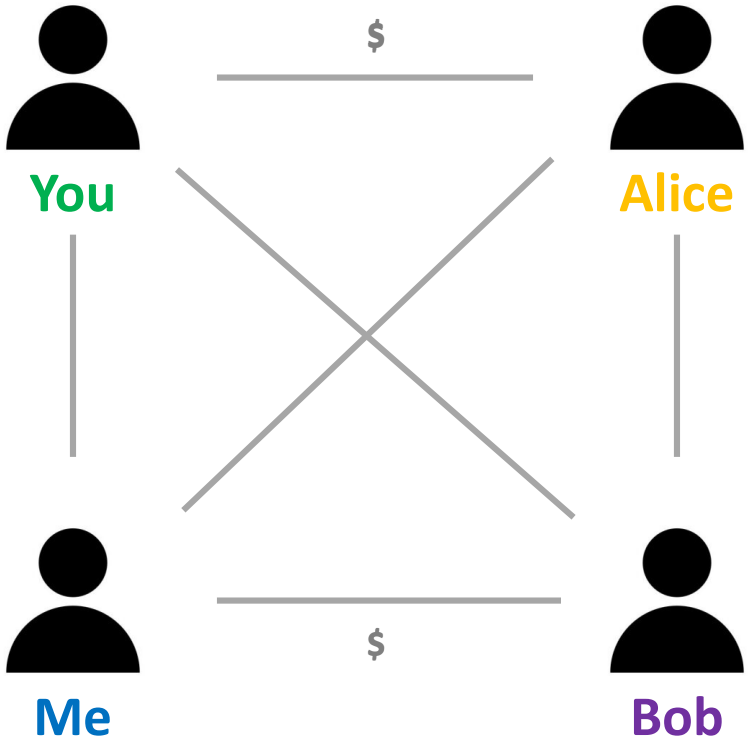


Best solution: digital signatures using 2 keys + unique ID

Ledger

You pay **Bob** 30$ `f2d81a2c9t3r`
Alice pays **Me** 40$ `ca978112ca1`
Bob pays **you** 10$ `3e23e8160039`
Bob pays **you** 10$ `594e519ae49t`

Digital signatures using private and public keys + changing the message

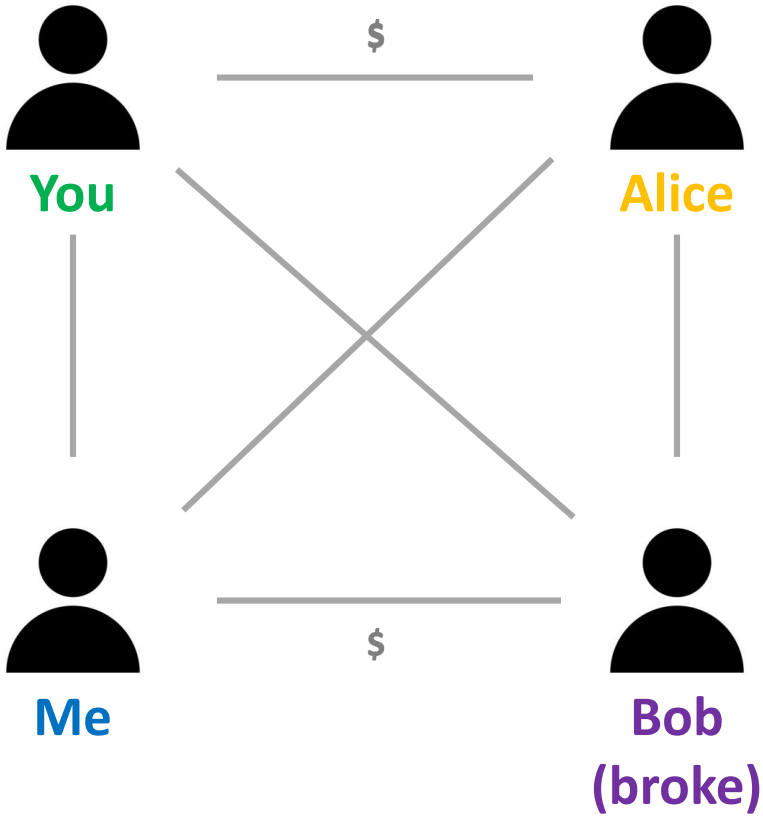


How can we avoid overspending?

Ledger

You pay **Bob** 30$ `f2d81a2c9t3r`
Alice pays **Me** 40$ `ca978112ca1`
Bob pays **you** 5K$ `3e23e8160039`

What if Bob doesn't have 5K?

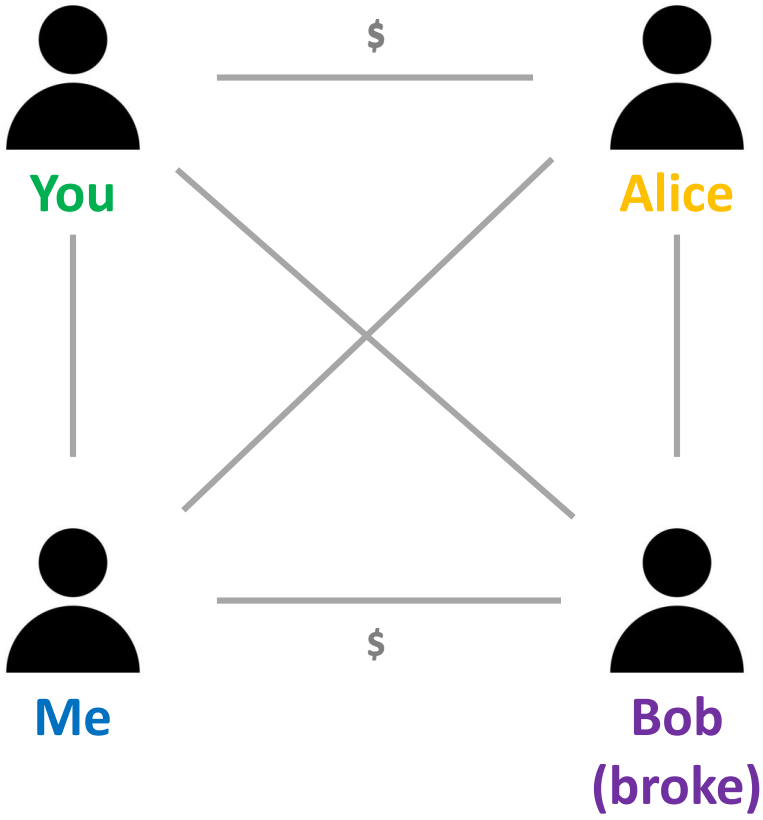


Adding the current state of the world at the top!

Ledger

You own 100$
Alice own 100$
Bob own 100$
You own 100$

You pay **Bob** 30$ `f2d81a2c9t3r`
Alice pays **Me** 40$ `ca978112ca1`
Bob pays **you** 5K$ `3e23e8160039`

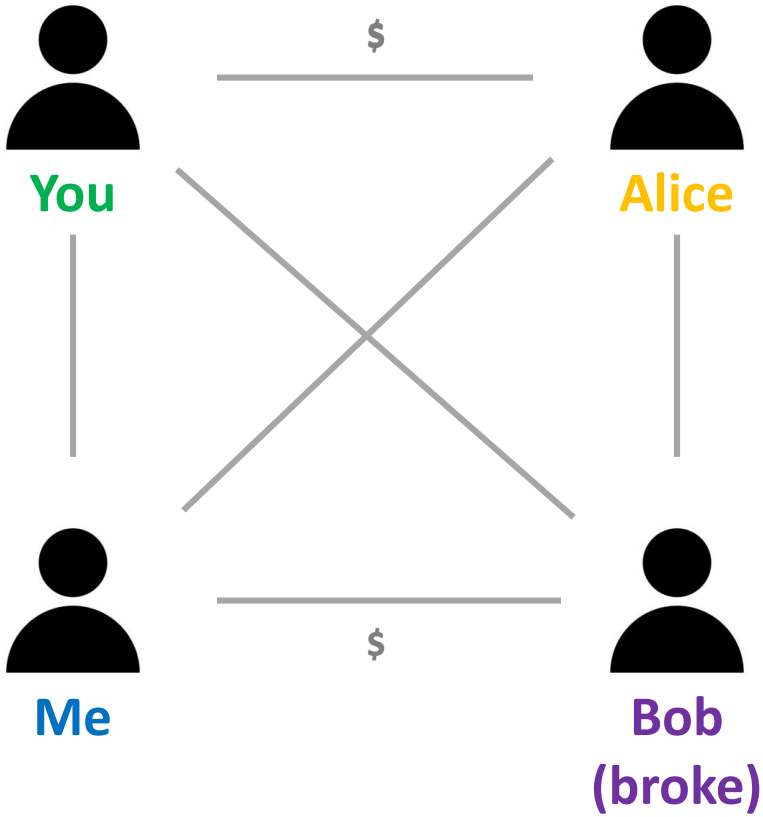


Ledger

You own 100$
Alice own 100$
Bob own 100$
You own 100$

You pay Bob 30$ f2d81a2c9t3r
Alice pays Me 40$ ca978112ca1
Bob pays you 5K$ 3e23e8160039

invalid

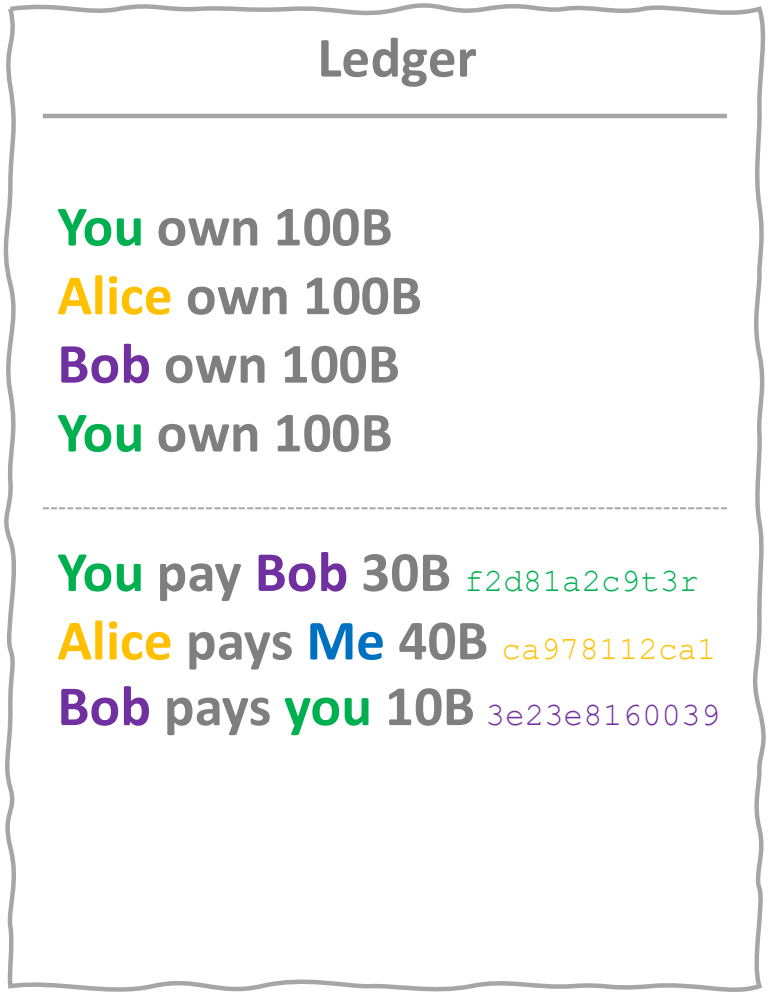


Ledger

You own 100B
Alice own 100B
Bob own 100B
You own 100B

You pay Bob 30B `f2d81a2c9t3r`
Alice pays Me 40B `ca978112ca1`
Bob pays you 10B `3e23e8160039`

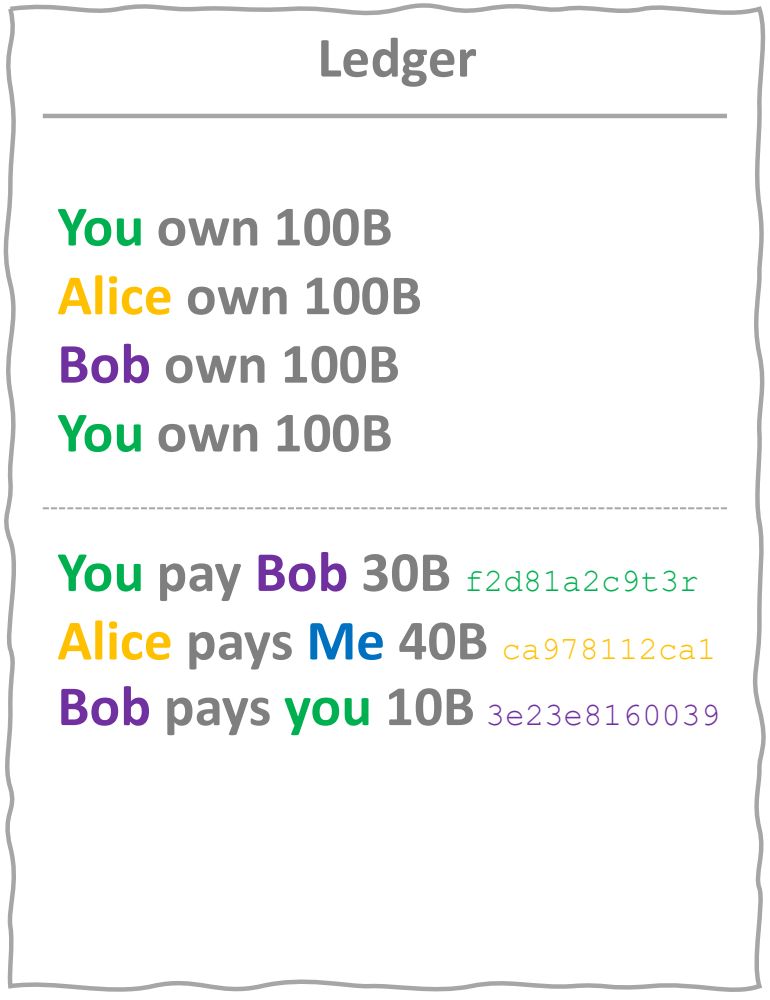
If you use $ to buy a digital coin named B, and that everyone use B, then you don't need dollars anymore: you'll use the digital ledger to pay and be paid



=

Crypto

= Transaction History!

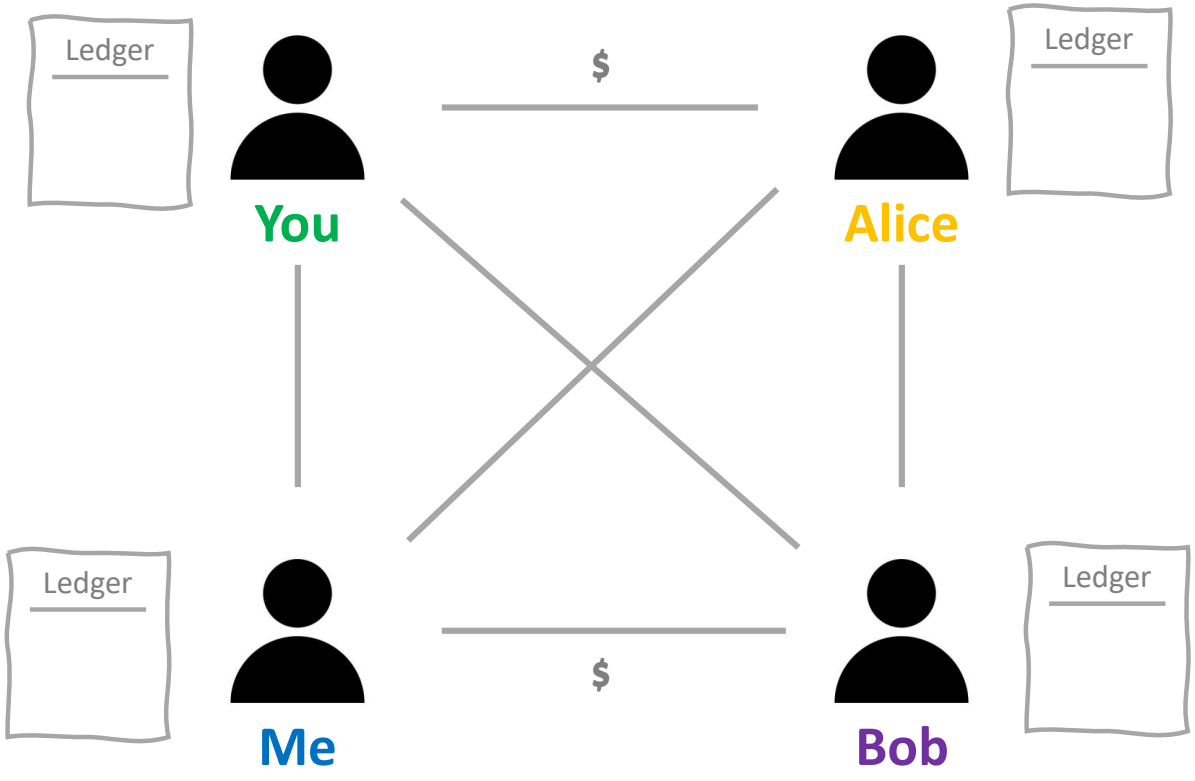


Who ows this?

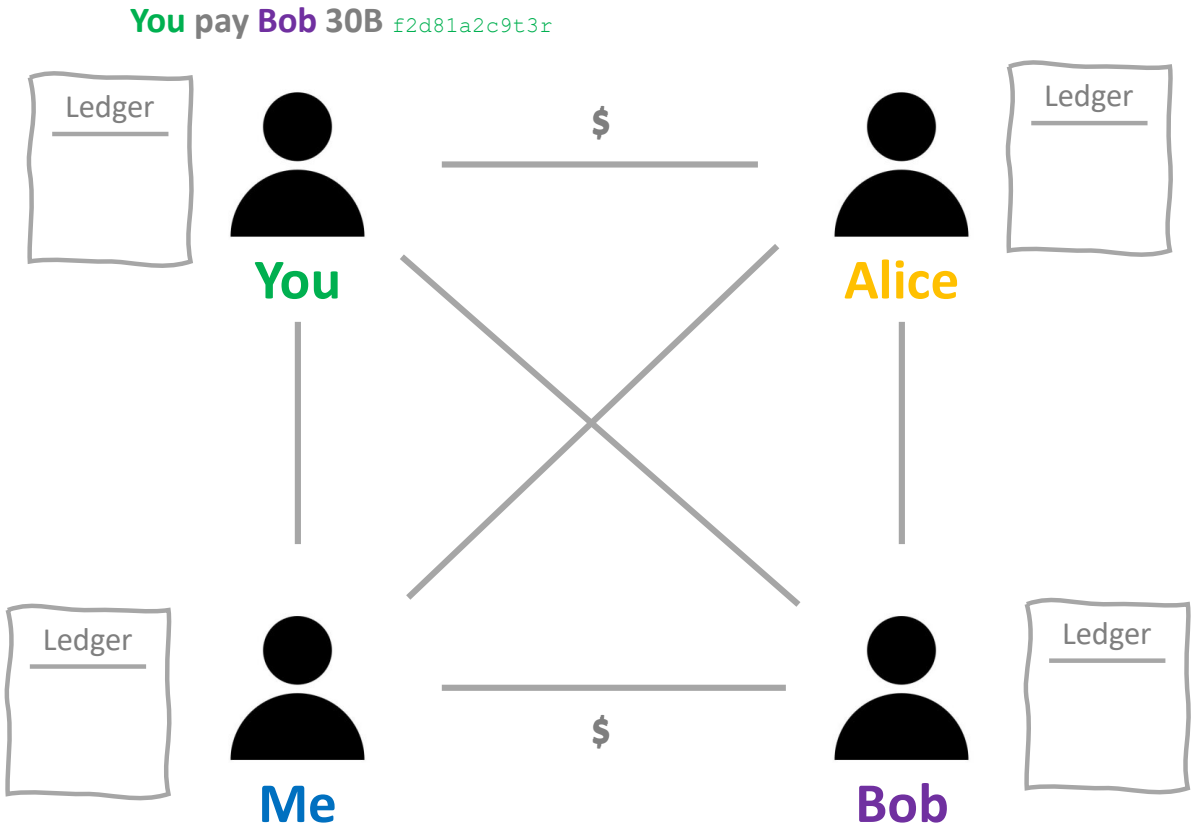
Where is it stored?

Who sets the rules

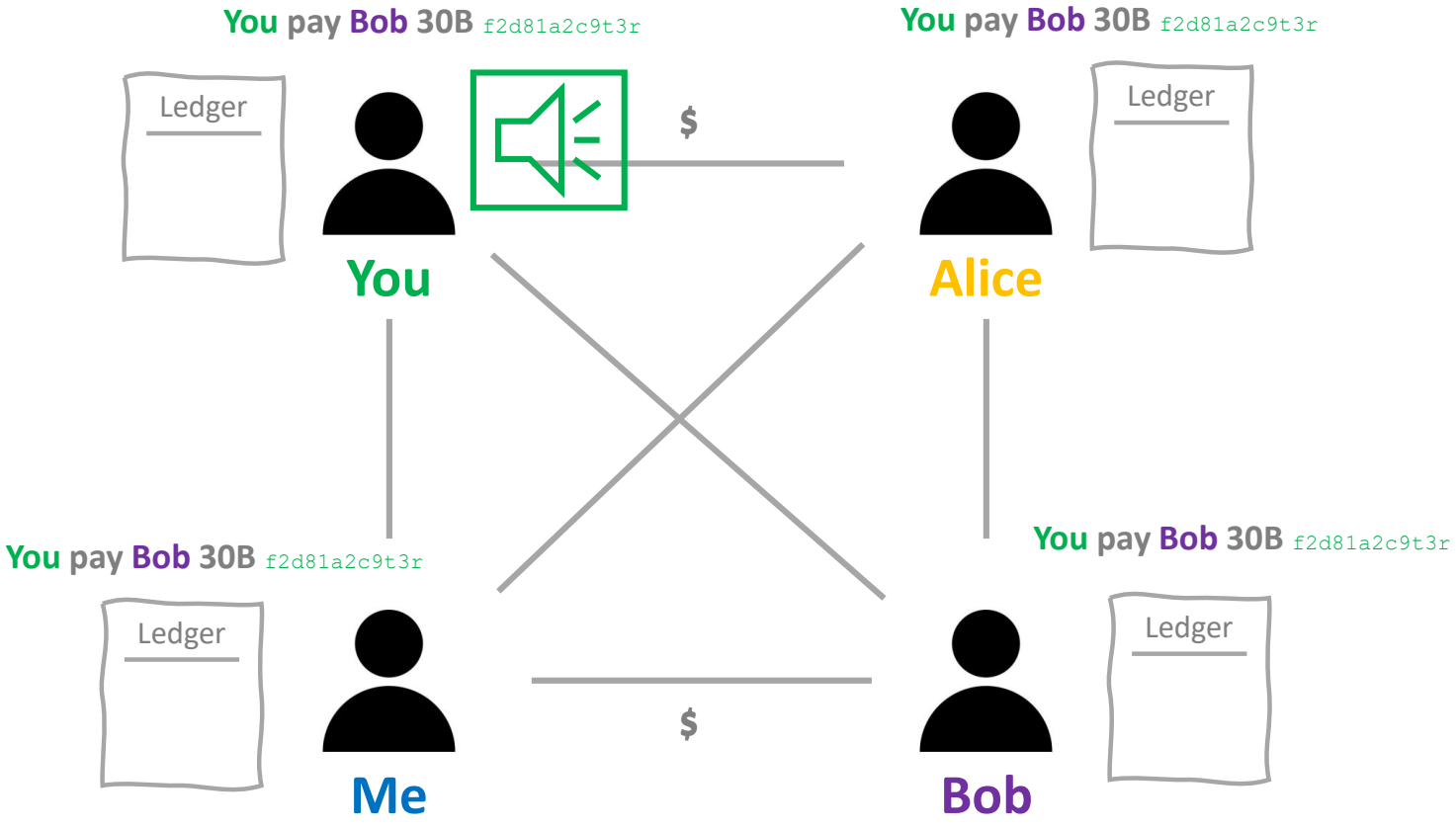
Solution: a decentralized P2P network



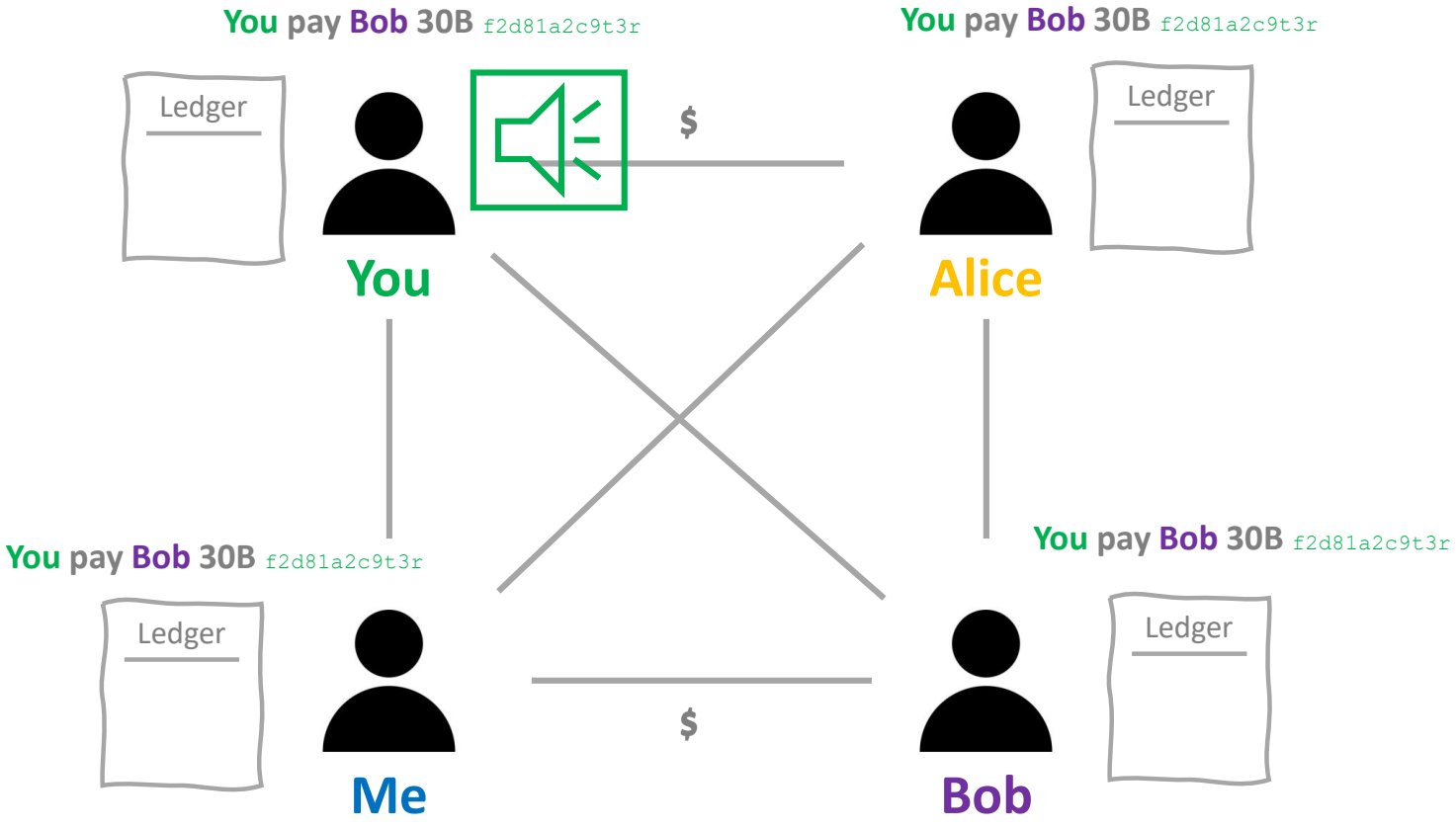
Broadcasting transactions



Broadcasting transactions: a terrible idea?

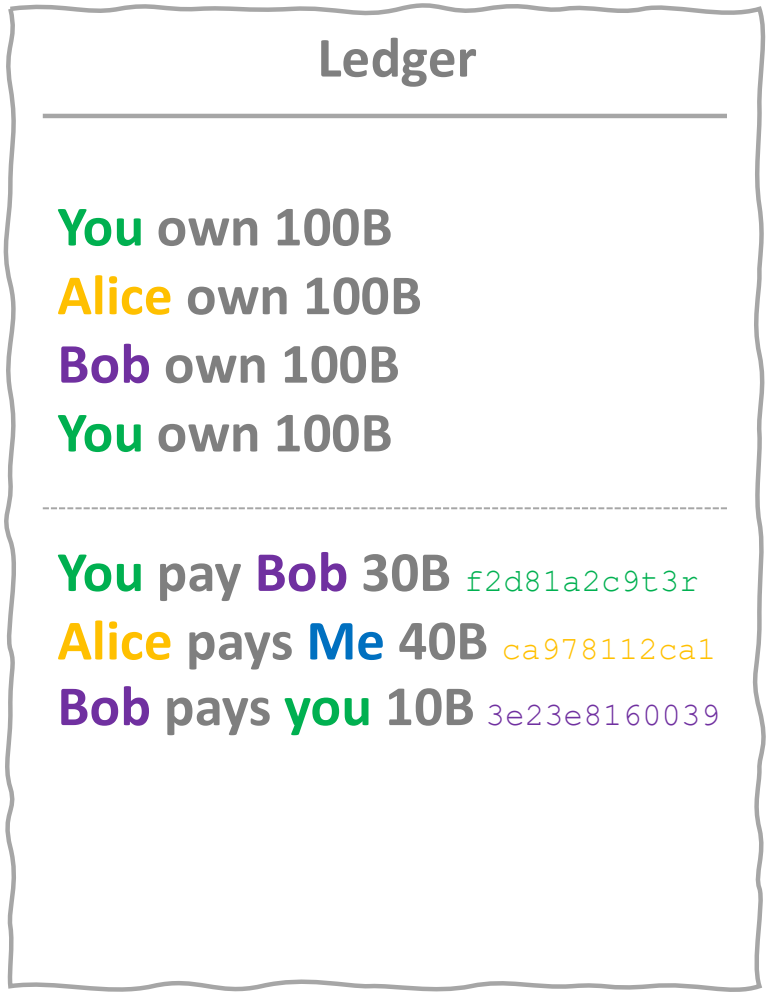


How to avoid **double spending** ?



Which ledger to trust? Satoshi's solution: the one with the most computational work

Satoshi's solution: Proof of Work



+ Cryptographic hash functions

Satoshi's solution: Proof of Work (invented in 1993 to deal with spam emails)

Ledger

You own 100B
Alice own 100B
Bob own 100B
You own 100B

You pay Bob 30B `f2d81a2c9t3r`
Alice pays Me 40B `ca978112ca1`
Bob pays you 10B `3e23e8160039`

13452782625

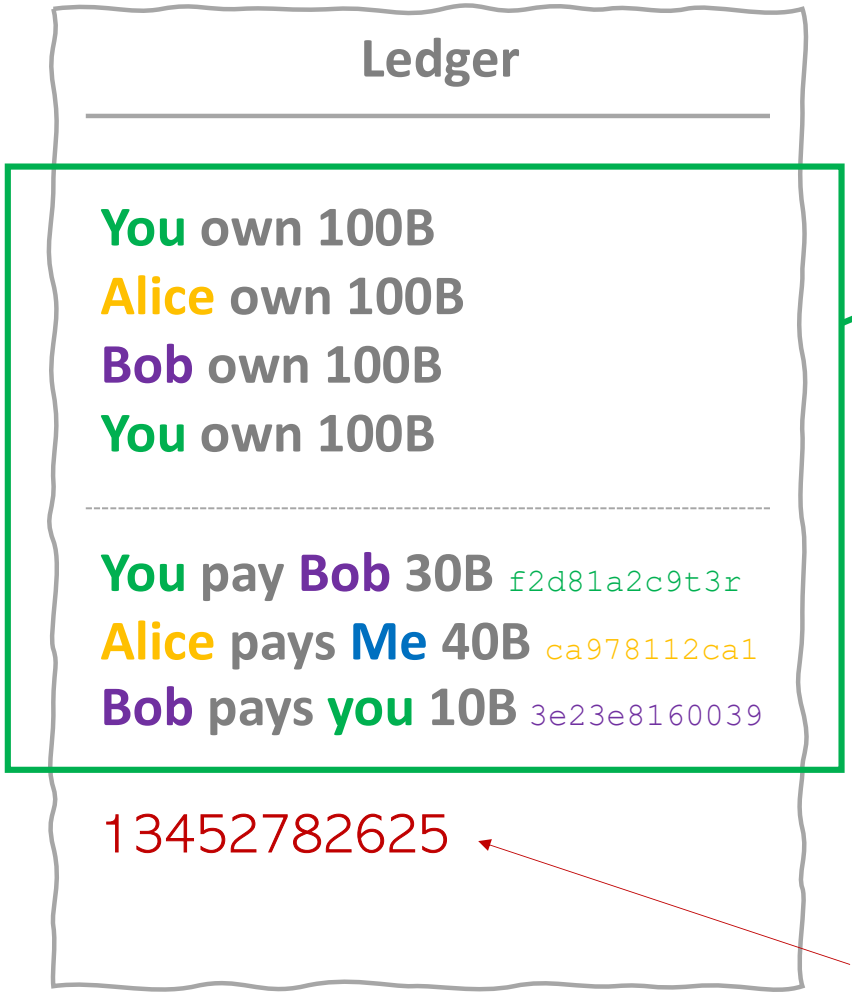
« nounce »

+ Cryptographic hash functions

X zeros

Hash(Ledger+nounce) = 000000drse58

Satoshi's solution: Proof of Work + Merkle Trees



Hash()

Merkle root:
594e519ae499312b2
9433b7Dd8a97ff068
defcba9755b6d5d00e
84c524d67b06

\+ Cryptographic hash functions

X zeros

Hash(Merkle root+nounce) = 000000drse58

A Bitcoin block [see here]

Block Detail

Block Hash:	0000000000000000fcd876c8812931414058606190 8fa141fcbc57f1132cc49a
Height:	**103890**
Previous Block:	0000000000000001b98b75cf1f446975b60703a0de766887bd360bdd54527f6d
Merkle Root:	f55d6fe0d3a6a2ce04c75866dca14b40c87a25dab76efd00f47aa1605a2cb5b6

A Bitcoin block [see here]

Block Hash

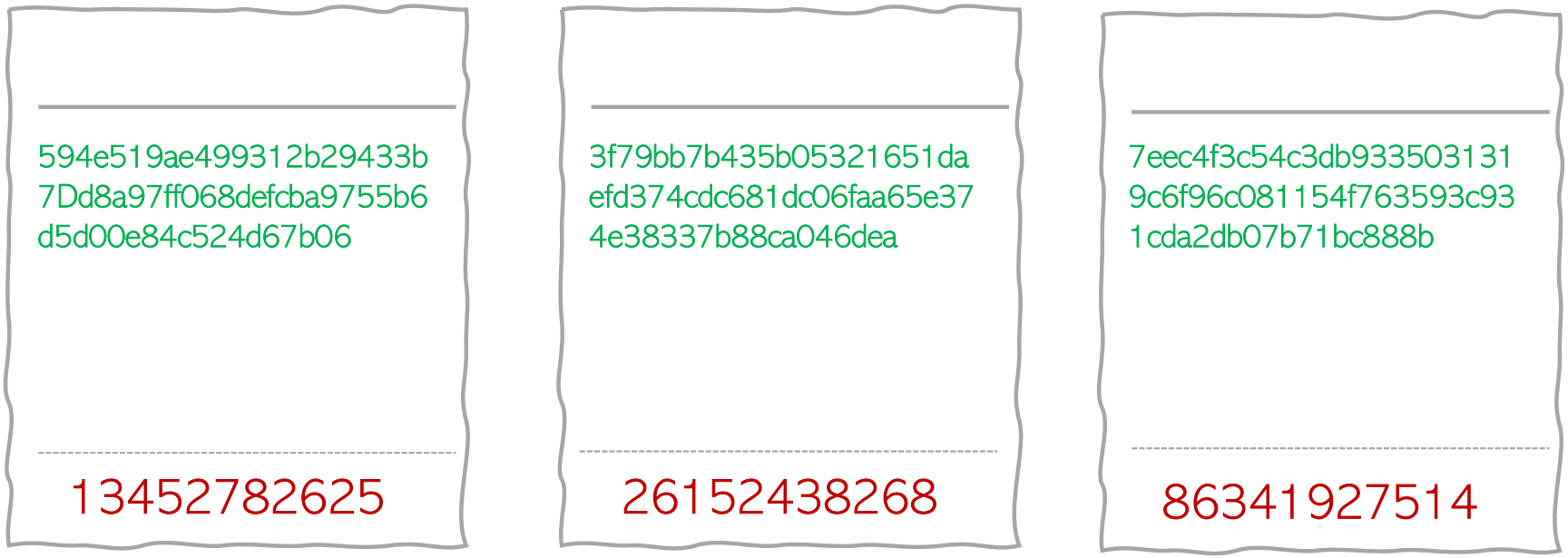
000000000000000000001f81f8d6fb77f396df76f15062d6764a9b03f42b0ac22

Summary

Height	◀ 764,014 ▶	Relayed By	F2Pool
Confirmations	5	Difficulty	142.93 T / 36.76 T
Block Size	1,645,947 Bytes	Block Reward	6.25000000 BTC
Stripped Size	784,071 Bytes	Fee Reward	0.28072551 BTC
Weight	3,998,160	Tx Count	2,826
Time	2022-11-20 15:20:36	Tx Volume	22,356.53918626 BTC

Merkle Root	a2a3f9244382ed9cd7a8f245d8676e225f7d2a7d2d82a115a61931b5423aeab0
Version	0x2dadc004
Nonce	0xee4c3853

Not one ledger: a sequence of ledgers



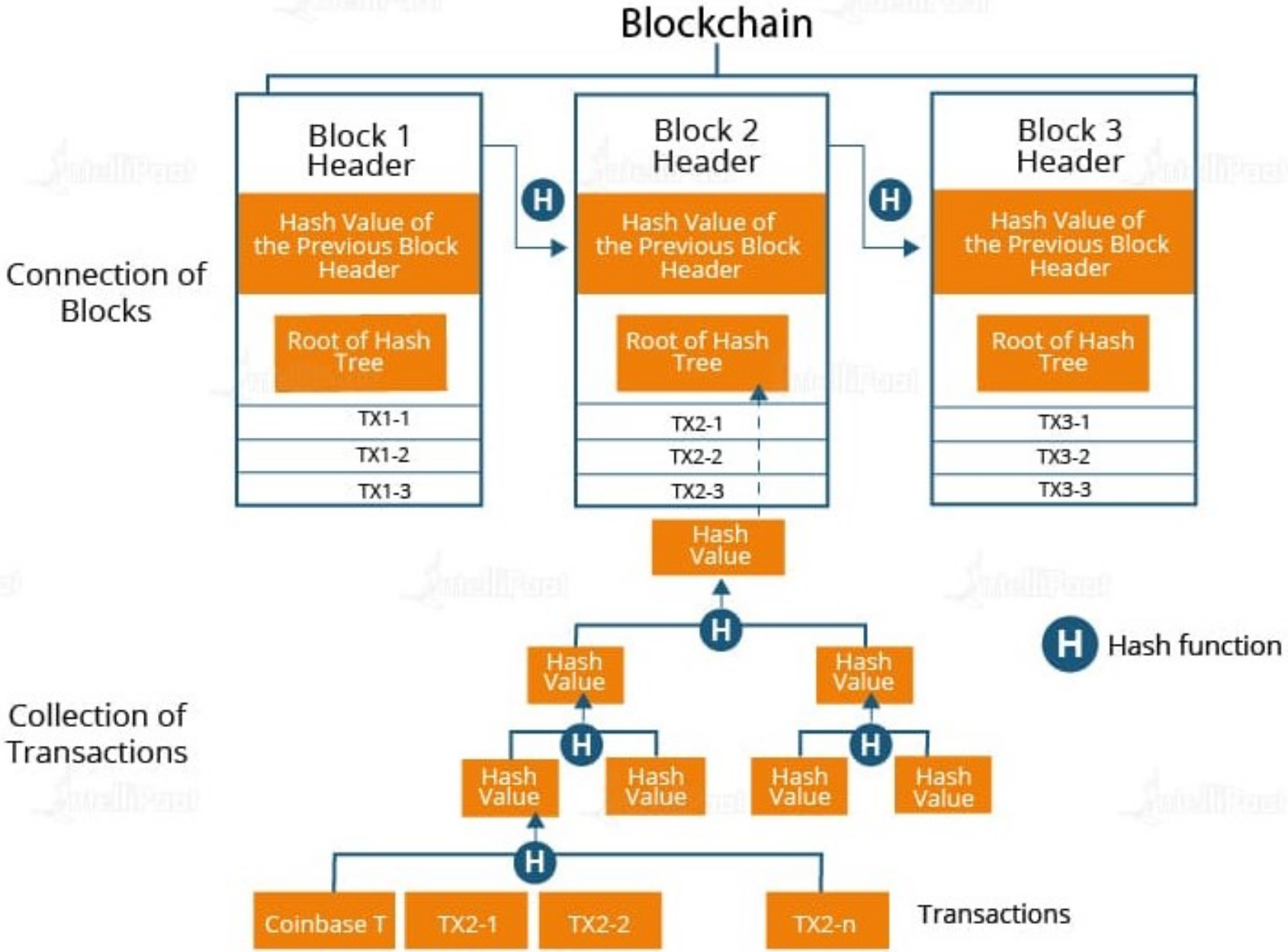
Hash() =
000000drse58

Hash() =
000000t6arf6

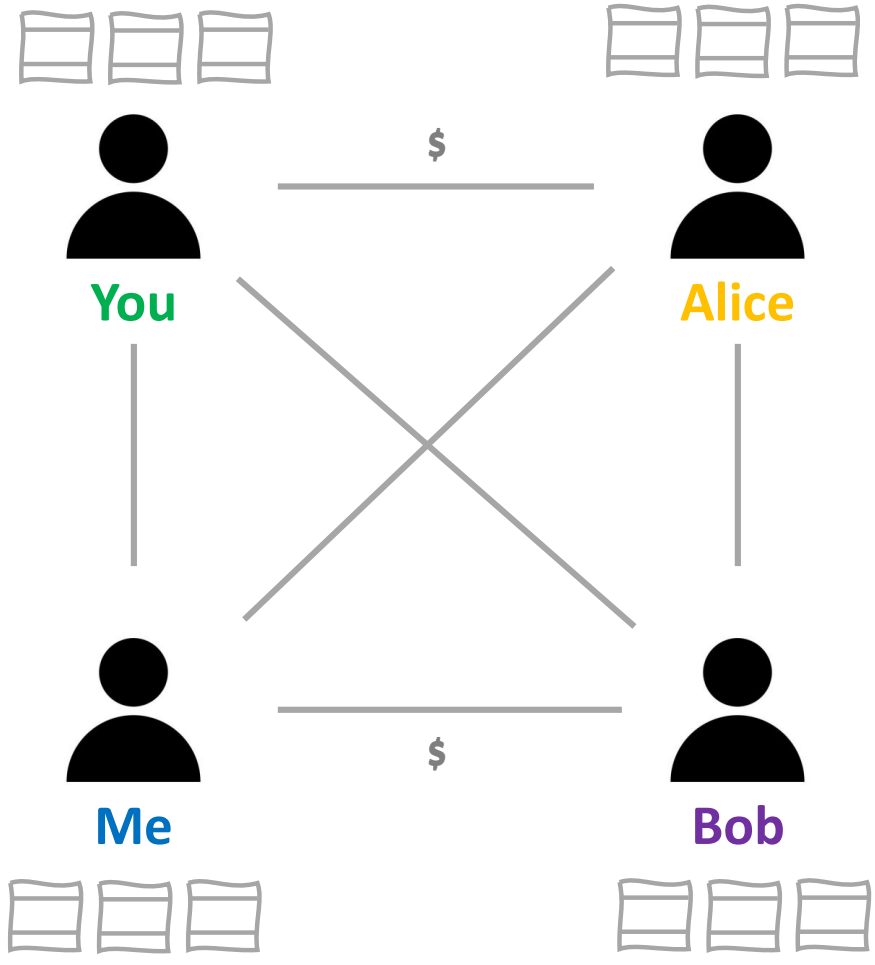
Hash() =
000000ty50o6l

A sequence of ledgers : The **blockchain**

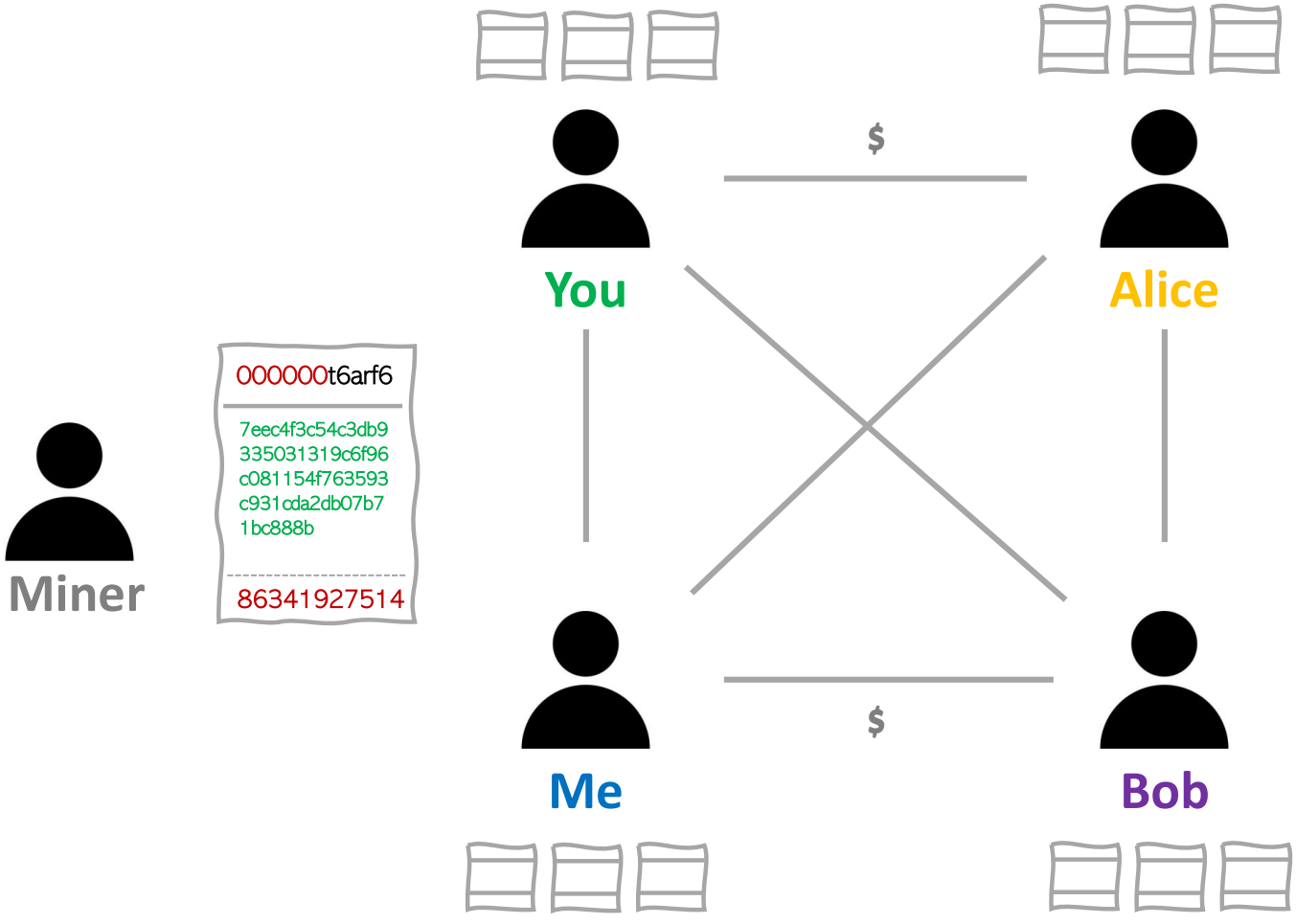
A sequence of ledgers : The **blockchain**



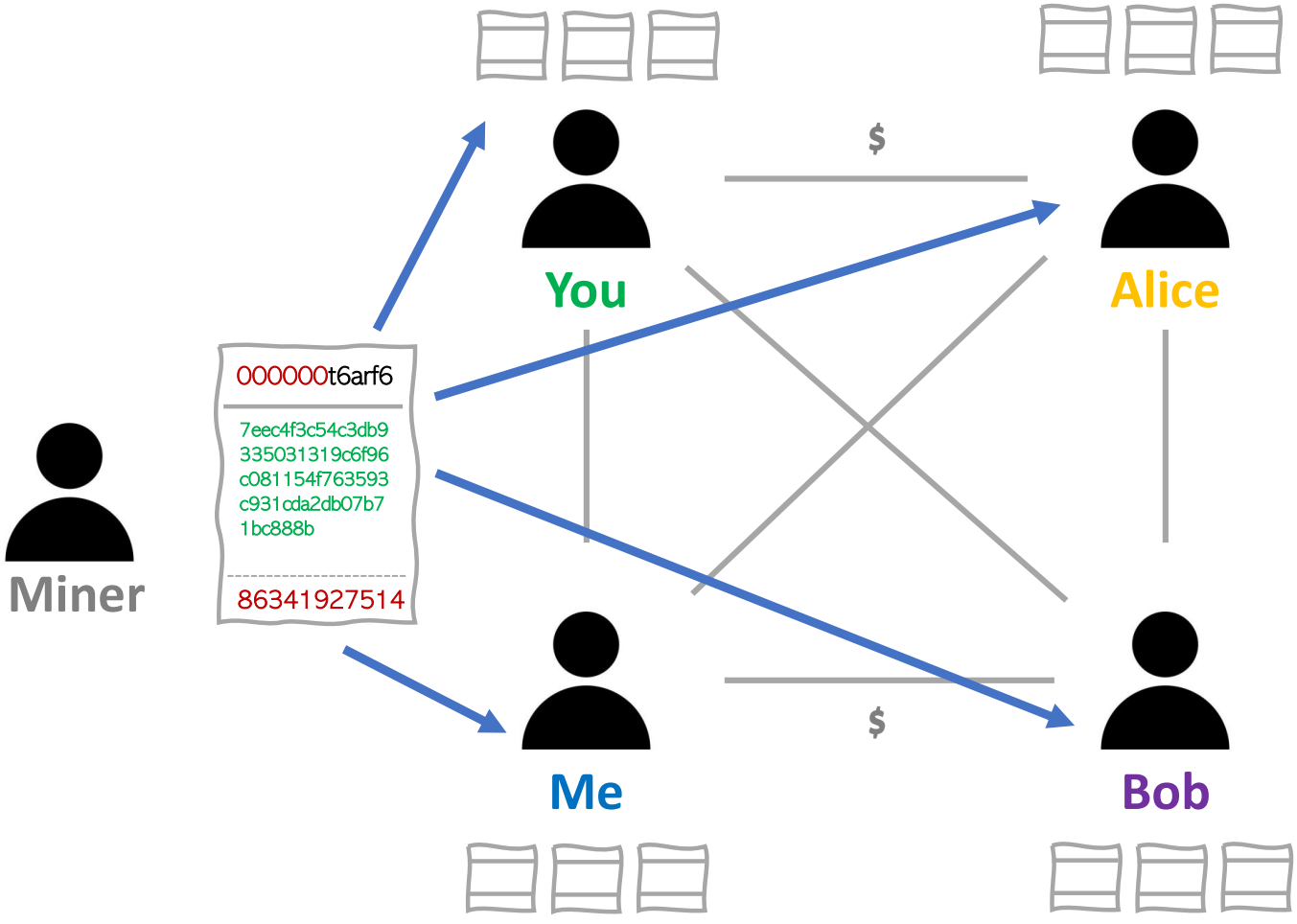
Miners and the production of blocks



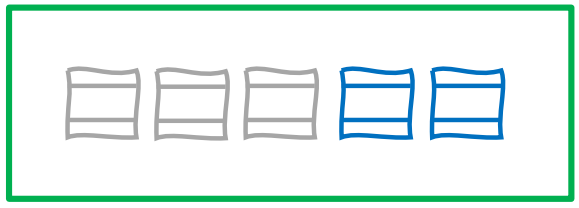
Miners and the production of blocks

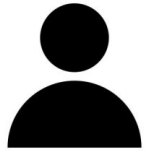


Miners and the production of blocks



Proof of work in a nutshell: the longer the better

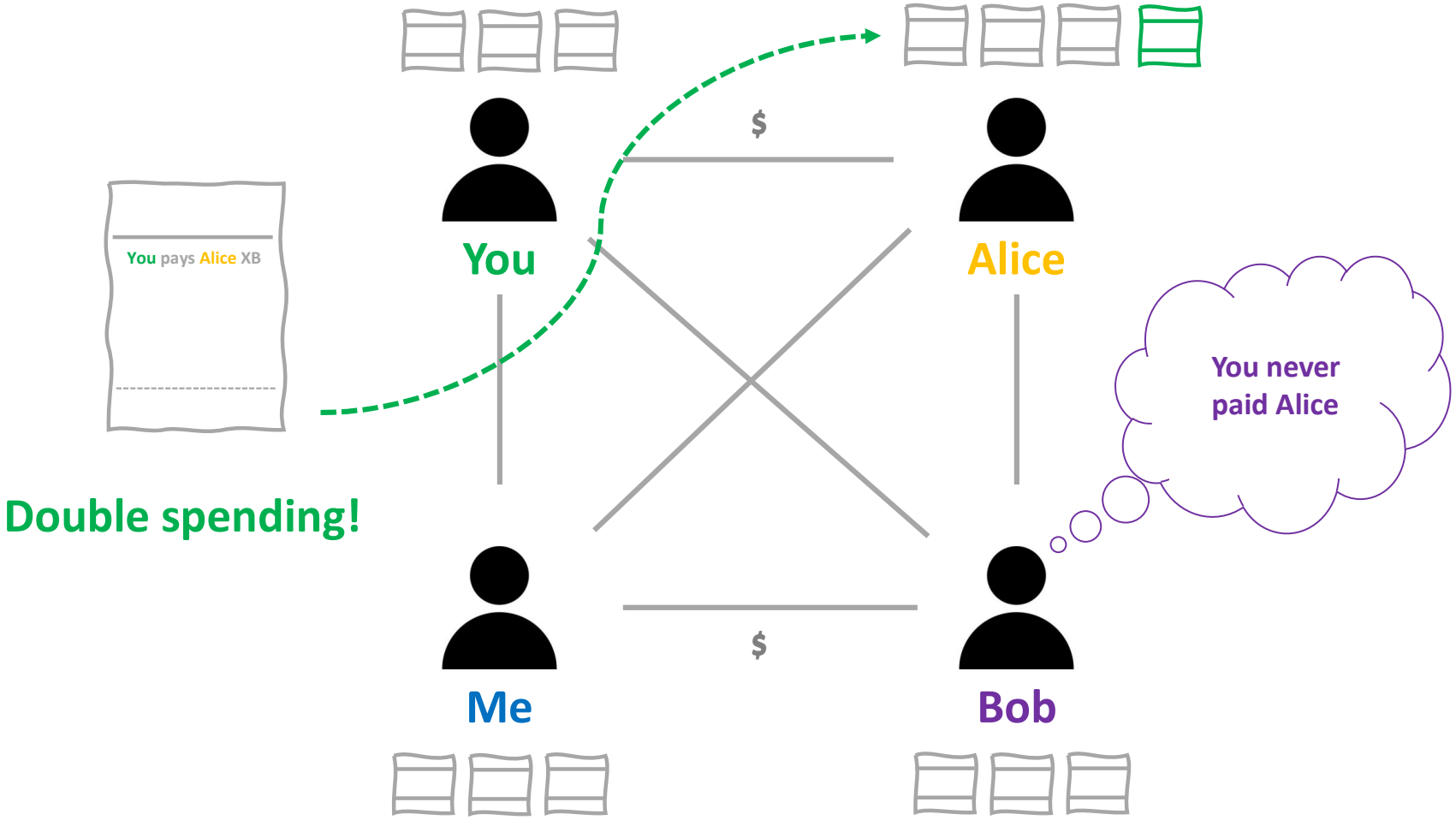




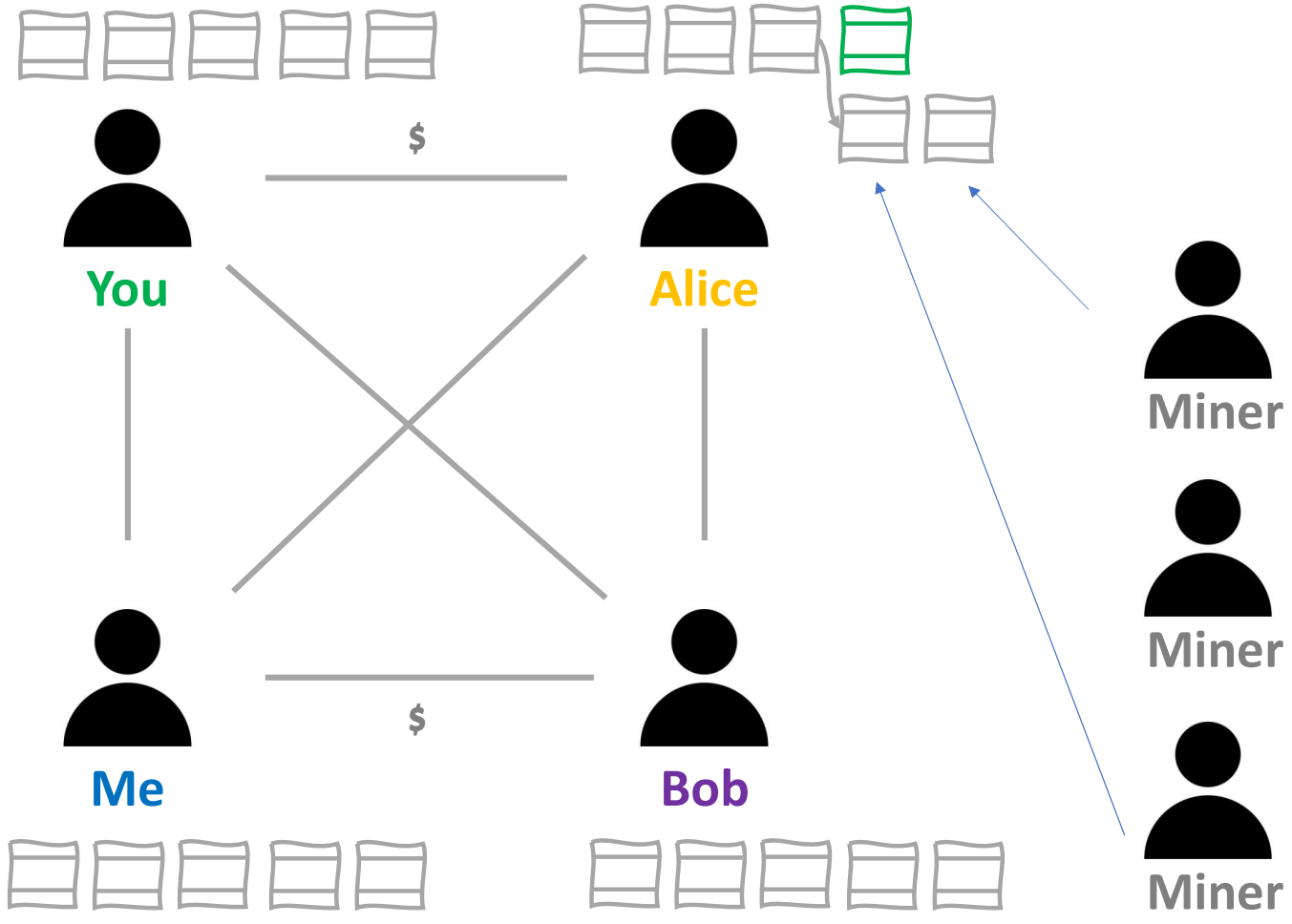
You

Always accept the longest one = the one with more work!

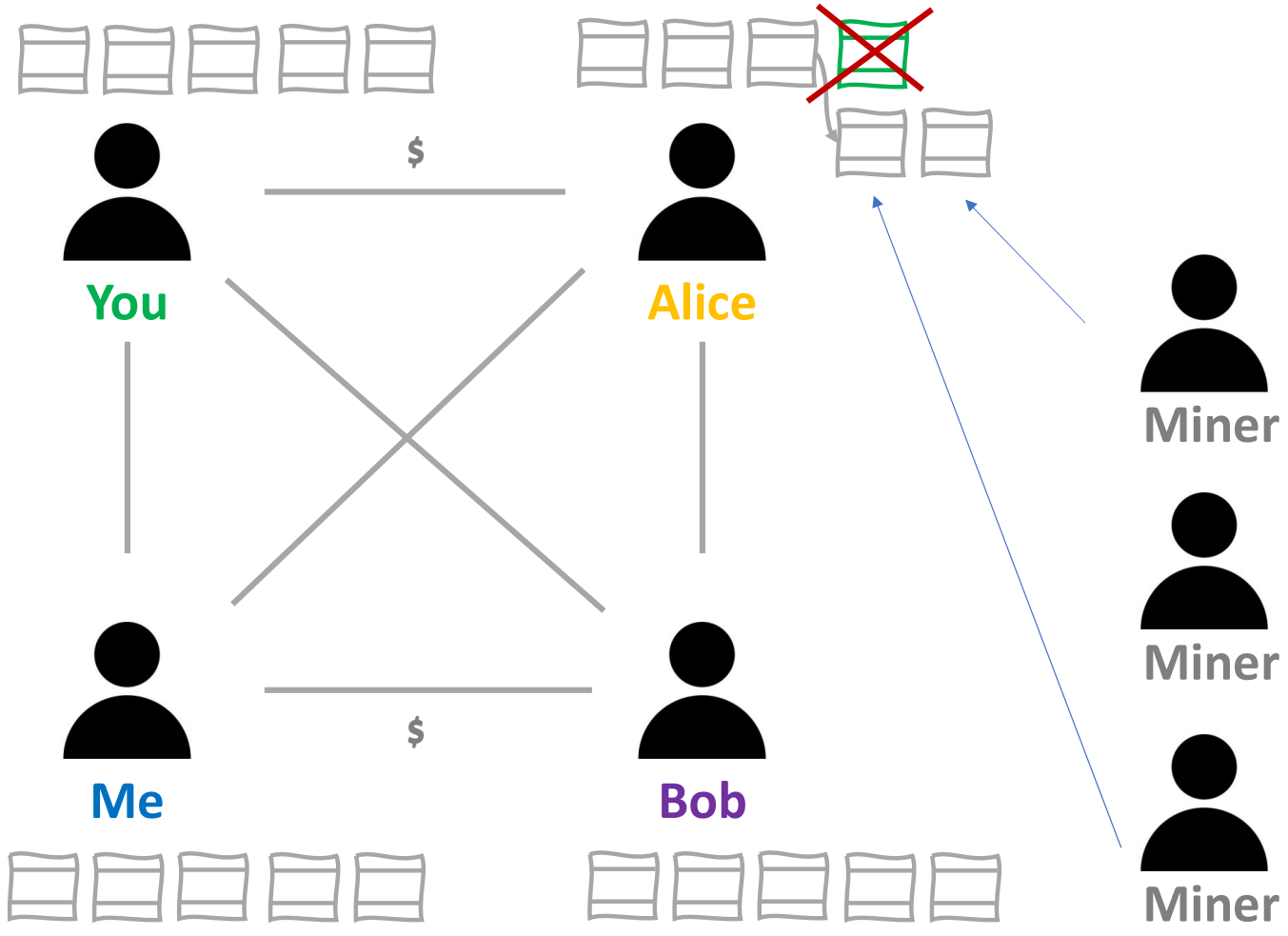
Let's try to fool Alice by double spending



Let's try to fool Alice by double spending



Let's try to fool Alice by double spending: **fail!**



Going back to our definitions

- Blockchain:

- Cryptocurrency:

Going back to our definitions

- Blockchain: digital database or ledger distributed among the nodes of a peer-to-peer network (Haber and Stornetta, 1991)

- Cryptocurrency:

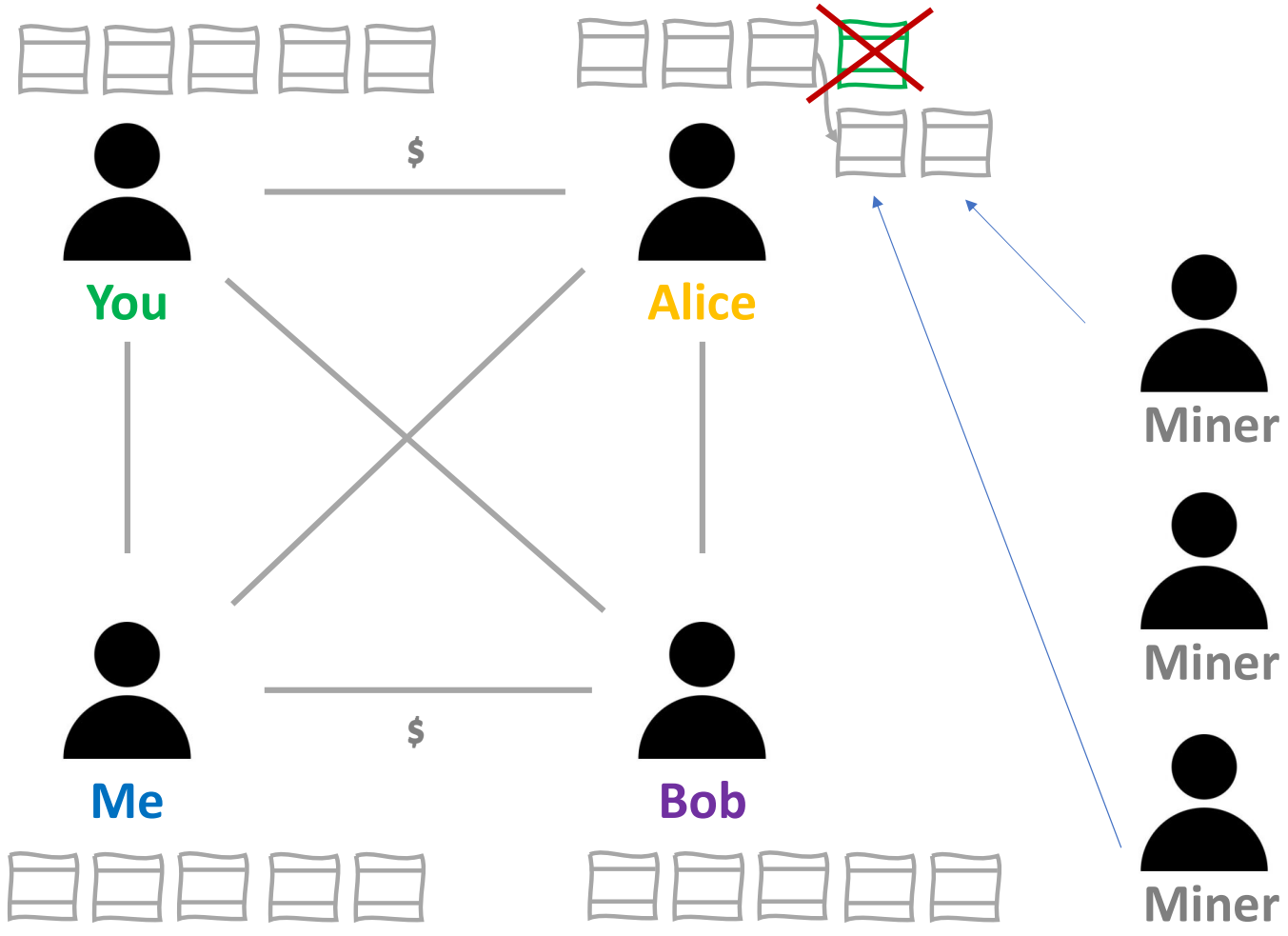
Going back to our definitions

- Blockchain: digital database or ledger distributed among the nodes of a peer-to-peer network (Haber and Stornetta, 1991)
  - Walmart, Pfizer, AIG, Siemens, Unilever, etc.
  - IBM has created its Food Trust blockchain to trace the journey that food products take to get to their locations
- Cryptocurrency:

Going back to our definitions

- Blockchain: digital database or ledger distributed among the nodes of a peer-to-peer network (Haber and Stornetta, 1991)
  - Walmart, Pfizer, AIG, Siemens, Unilever, etc.
  - IBM has created its Food Trust blockchain to trace the journey that food products take to get to their locations
- Cryptocurrency: a digital or virtual currency secured by cryptography and based on a decentralized network (a blockchain)

Let's think about it for next week: **limitations?**

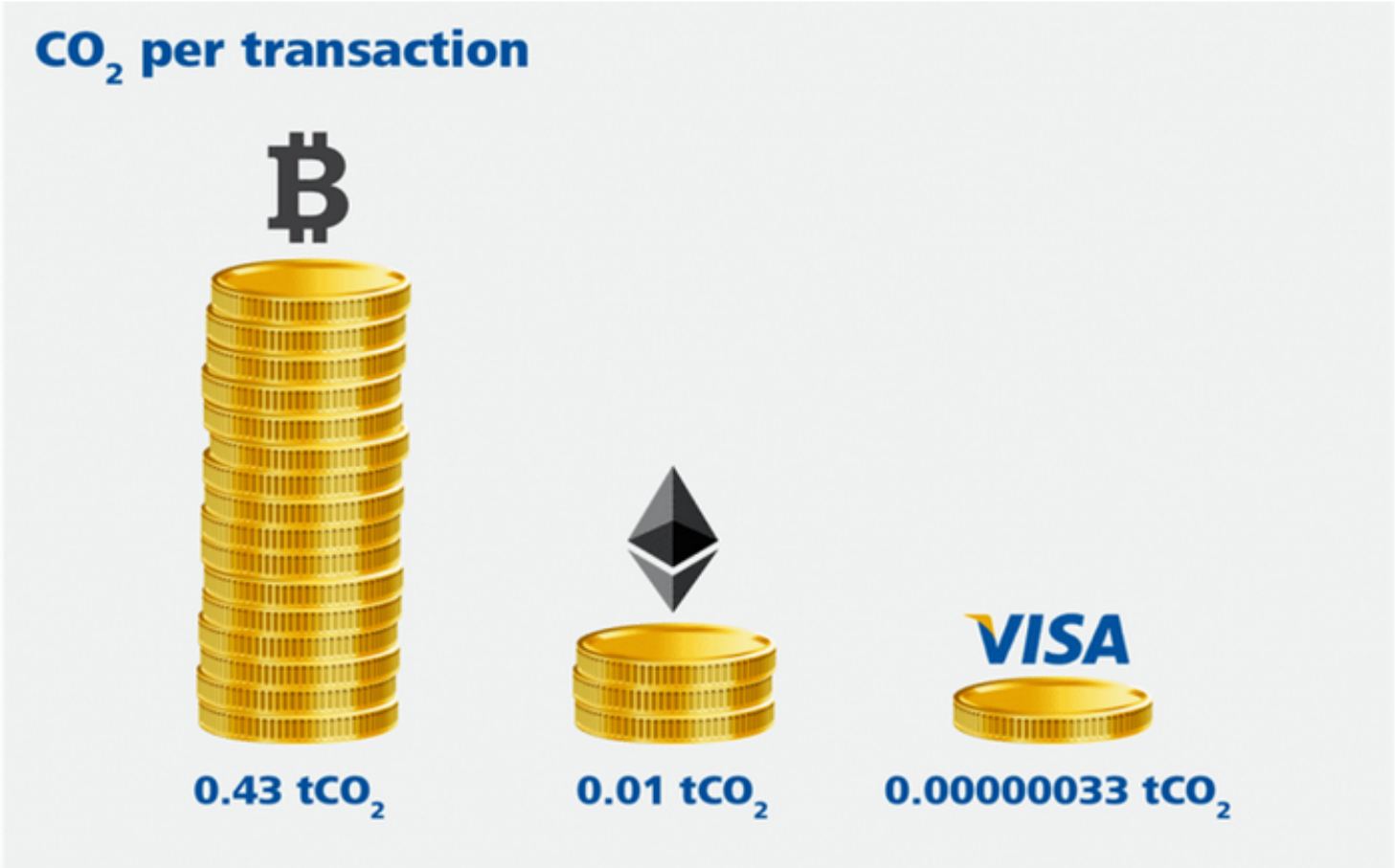


Limitation 1: Costly

- Mining is costly

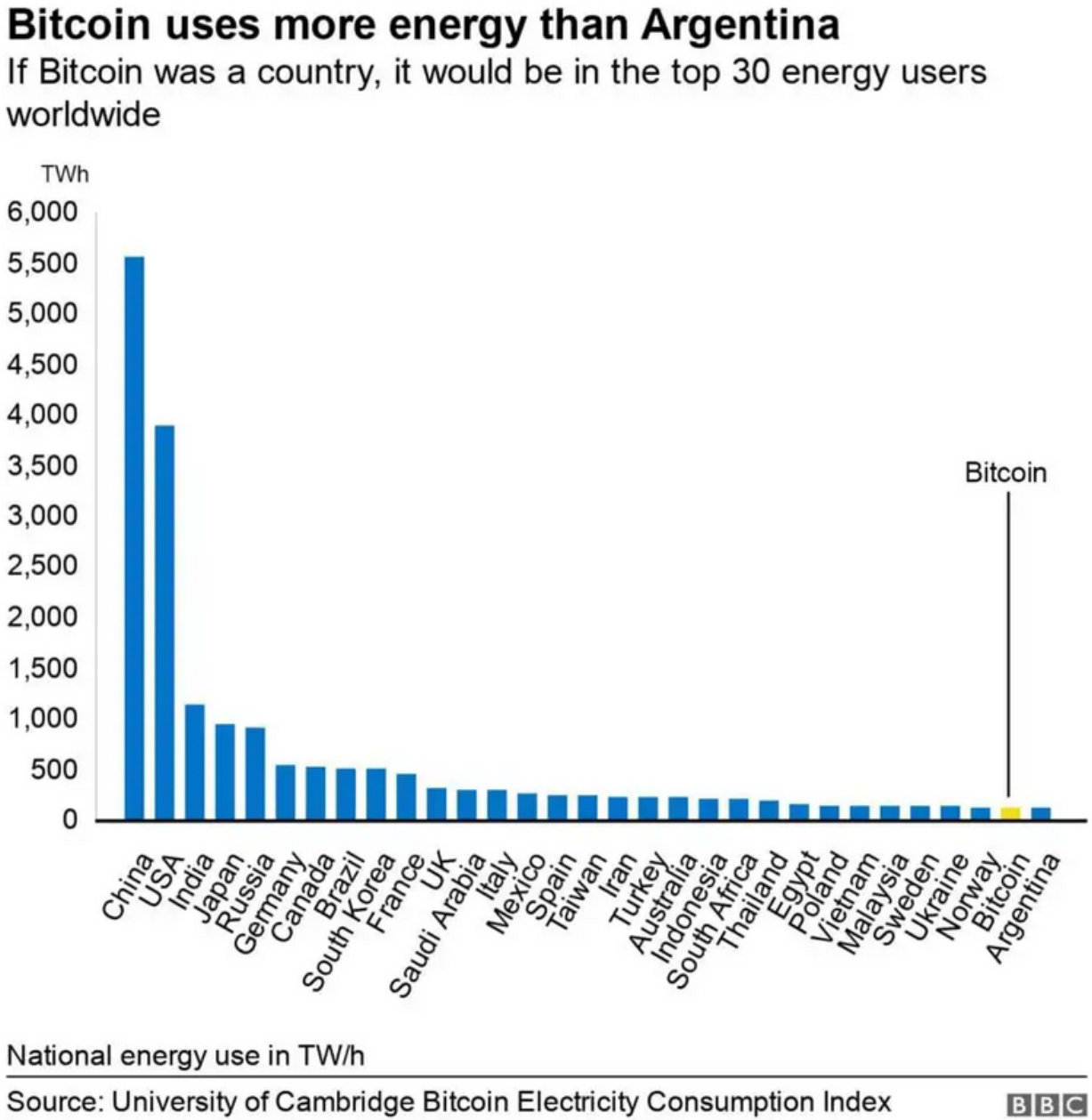
Limitation 1: Costly

- Mining is costly (huge negative externality like CO2 emissions)
- 200 million tons of CO2 since its launch



Future of Payments- Carbon Footprint of Bitcoin vs Ethereum vs Visa. Source: cleancoins.io

Limitation 1: Costly

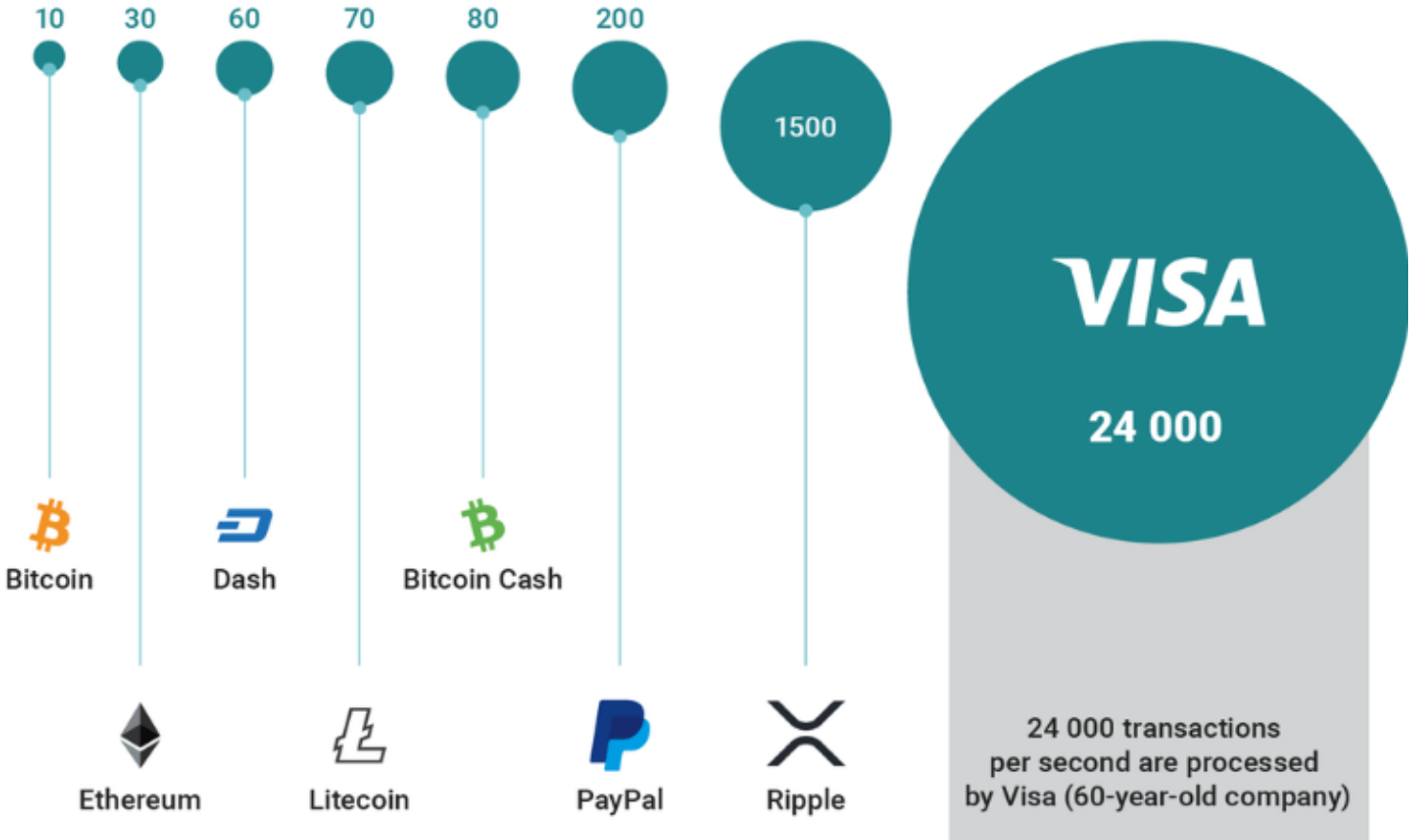


Limitation 2: Slow

- Mining is slow (limit in number of transactions)

Limitation 2: Slow

- Mining is slow (limit in number of transactions)
- Max $\approx$ 2400 transactions per block

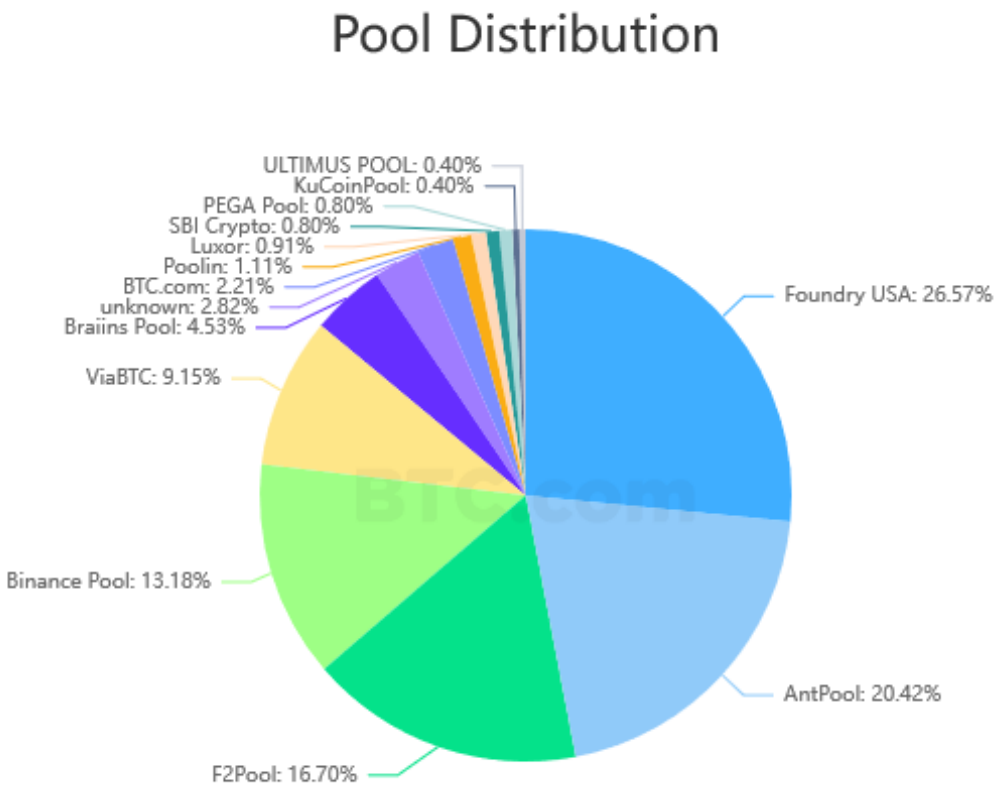


Limitation 3: Limited supply and centralization

- Monetary policy (93.1% of BTC already mined)
- What if a miner has more than 51% of CPU? See here in real-time.

Limitation 3: Limited supply and centralization

- Monetary policy (93.1% of BTC already mined)
- What if a miner has more than 51% of CPU? See here in real-time.



More Bitcoin weaknesses: `Bitcoin Script`

- Part of Bitcoin that we haven't talked about yet (because almost no-one sees it): `Bitcoin Script`
- It's a minimalist programming language that handles the details of a transaction such as signature verification
- Why was an entire mini-language created for Bitcoin?
  - to support transactions which require m out of n people to sign to become valid
  - to support slightly more advanced payment logic, such as i inputs, j outputs
- The details of the Bitcoin script specification can be found here

More Bitcoin weaknesses: `Bitcoin Script`

- `Bitcoin Script` has a **huge** problem: it's Turing-incomplete

More Bitcoin weaknesses: `Bitcoin Script`

- `Bitcoin Script` has a **huge** problem: it's Turing-incomplete

Compare the following lines of code. This code will run and halt:

```
sleep(1)
```

This code will run forever:

```
while True:
    continue
```

But what will this code do:

```
  while j < i:
    j = exp(i)-sin(i)*i**2
    i = tanh(j)
```

General question: Is there a general algorithm which can determine whether this (or any other) code will come to a final stop, looking just at the code and the initial values of its variables (e.g. i and j in the last example)?

More Bitcoin weaknesses: `Bitcoin Script`

- Alan Turing wrote in 1936: *there exists no general algorithm that can decide whether a program will or will not come to a stop*

More Bitcoin weaknesses: `Bitcoin Script`

- Alan Turing wrote in 1936: *there exists no general algorithm that can decide whether a program will or will not come to a stop*

- Direct consequences for the design of Blockchain scripting languages like `Bitcoin script:`

  - either they are by purpose made Turing-incomplete (e.g. by taking away the possibility to have loops)

  - or they can potentially run forever (on the miners' machines !)

More Bitcoin weaknesses: `Bitcoin Script`

- Since the scripts must be executed by the network's validator nodes (not on the computer which submitted them), this is a big security problem
- Satoshi Nakamoto implemented Bitcoin script as a Turing-incomplete language
- It lacks the instructions for loops such that it will always be executed sequentially and come to a halt
- The price for this is that only the most simple algorithms can be implemented in `Bitcoin script`

Discussion