Blockchain Technologies

Internet of Things A.Y. 19-20

Prof. Chiara Petrioli
Dept. of Computer Science
Sapienza University of Rome

Speaker Michele Martinelli

PART I

How many of you:

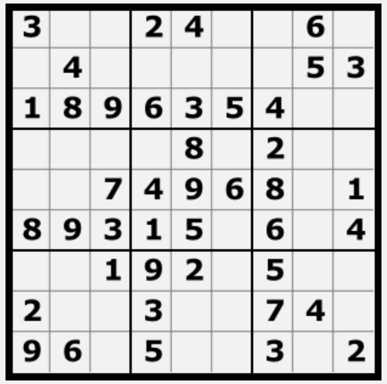
- Have heard about bitcoins?
- Own cryptocurrency?
- Feel you understand the underlying blockchain technology?

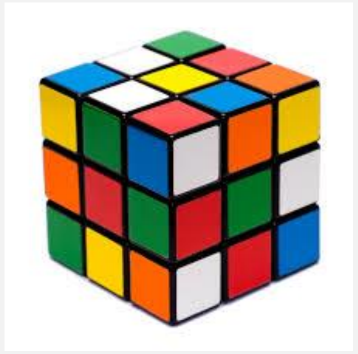
Block(chain) idea

1. Everyone **tries to solve** a puzzle

2. The **first one** who solve the puzzle **gets a reward ("a coin")**

3. The solution of **puzzle *i*** **defines puzzle *i+1***
(better: the puzzle i+1 "depends" on the puzzle i)





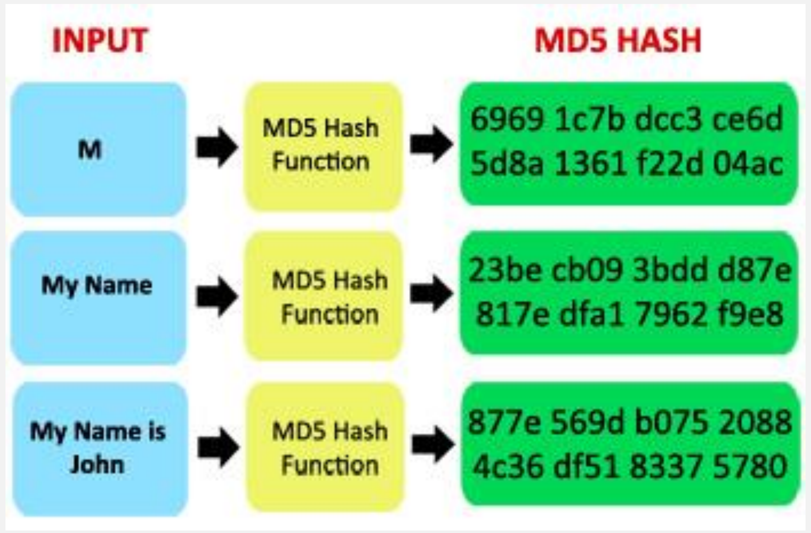
It's EASY to VERIFY the solution
But it's HARD to find it!

Hash functions

Hashing: you turn anything (as long as you can represent it as a string) into a fixed length of bit string

A cryptographic hash function is a special class of hash functions which has various properties making it ideal for cryptography.

1. **Deterministic**: no matter how many times you hash a particular input, you will always get the same result
2. **Quick Computation:** The hash function should be capable of returning the hash of an input quickly
3. **Pre-Image Resistance**: given H(A) it is infeasible to determine A, where A is the input and H(A) is the output hash.
4. **Avalanche effec:** Small Changes In The Input Changes the Hash.
5. **Puzzle Friendly**: For every output "Y", if k is chosen from a distribution with high min-entropy it is infeasible to find an input x such that H(k|x) = Y.


INPUT
MD5 HASH
M
MD5 Hash Function
6969 1c7b dcc3 ce6d 5d8a 1361 f22d 04ac
My Name
MD5 Hash Function
23be cb09 3bdd d87e 817e dfa1 7962 f9e8
My Name is John
MD5 Hash Function
877e 569d b075 2088 4c36 df51 8337 5780

SHA256 hash function

Here how you can try it in Python:

```
>>> import hashlib


>>> hashlib.sha256(“hello world”).hexdigest()
‘B94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9’

>>> hashlib.sha256("helli world").hexdigest()
'bd6952606ca18ccc9ff86bb8874ce0c61d7aa4fb72363e323f9eb2d3e783a487'
```

The puzzle

I want to find a particular hash…
For example, I want two ZEROs at the end of the hash (“00”)…
What string I have to use?

```
>>> import hashlib


>>> hashlib.sha256(“hello world”).hexdigest()
‘B94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9’

>>> hashlib.sha256("hello world 1").hexdigest()
'063dbf1d36387944a5f0ace625b4d3ee36b2daefd8bdaee5ede723637efb1cf4’

…

>>> hashlib.sha256("hello world 238").hexdigest()
‘d622375f29c4b35867479461040 4bdd1f4e060a7244568b5549eed9754dfa400’
```

The puzzle

…and 3 ZEROs at the end of the hash (“000”)…?

```
>>> import hashlib


>>> hashlib.sha256(“hello world 3198”).hexdigest()
‘838ee925a5eff45a3337363e9a2e993e02eddcc2edd90cdd4fec14b0c6e59000’



…and 6?
>>> hashlib.sha256(“hello world 28114982”).hexdigest()
3d28877e8af972f6732de4591245a87545b0221c26e446f09e98665a6d000000
```

It’s getting harder to find a solution, but easy to verify it!

The puzzle – python code

```
import hashlib

i=0
while True:
    res = hashlib.sha256("hello world "+str(i)).hexdigest()
    if res.endswith("000000"):
        print i
        print res
        break
    i += 1
```

The puzzle – create a chain

How can the puzzle “depends” on the previous one?

```
>>> hashlib.sha256(“hello world 3198”).hexdigest()
‘838ee925a5eff45a3337363e9a2e993e02eddcc2edd90cdd4fec14b0c6e59000’
```

The puzzle – create a chain

How can the puzzle “depends” on the previous one?

```
>>> hashlib.sha256(“hello world 3198”).hexdigest()
‘838ee925a5eff45a3337363e9a2e993e02eddcc2edd90cdd4fec14b0c6e59000’
```

A new string!

```
>>> hashlib.sha256(“
SECOND hello world + 838ee925a5eff45a3337363e9a2e993e02eddcc2edd90cdd4fec14b0c6e59000
  ”).hexdigest()
‘9b582a1b5fd41f36cc0a396d4747507148f3bdde58a4bff18891406a9158a72e’
```

But this hash doesn’t end with “000”…

The puzzle – create a chain

How can the puzzle “depends” on the previous one?

```
>>> hashlib.sha256(“hello world 3198”).hexdigest()
‘838ee925a5eff45a3337363e9a2e993e02eddcc2edd90cdd4fec14b0c6e59000’
```

A new string!

```
>>> hashlib.sha256(“
SECOND hello world + 838ee925a5eff45a3337363e9a2e993e02eddcc2edd90cdd4fec14b0c6e59000
   ”).hexdigest()
‘9b582a1b5fd41f36cc0a396d4747507148f3bdde58a4bff18891406a9158a72e’
```

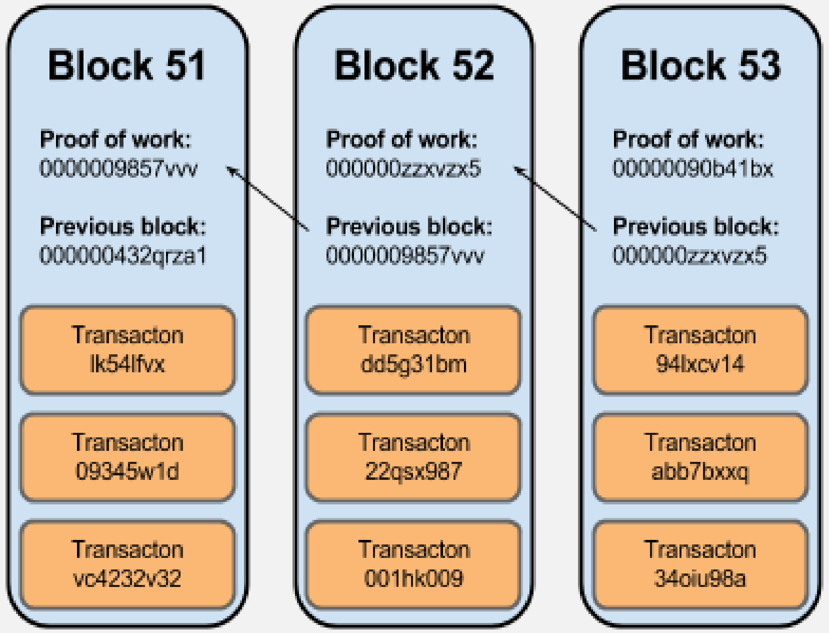
But this hash doesn’t end with “000”… let’s do the puzzle again:

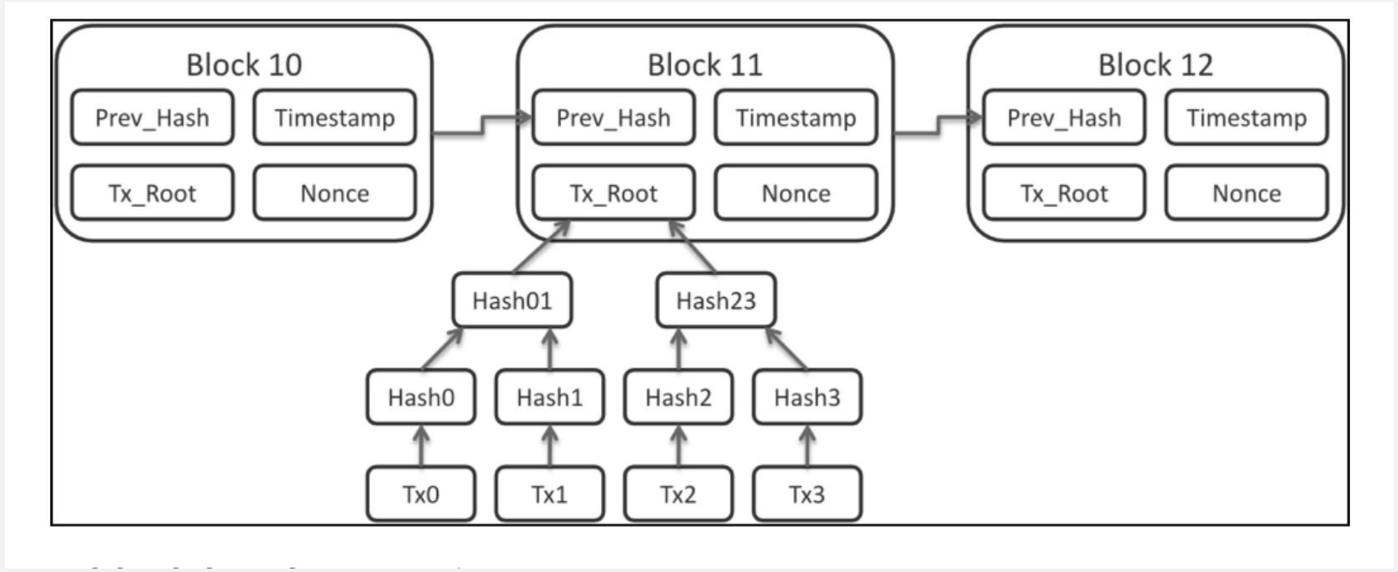
```
SECOND hello world +
838ee925a5eff45a3337363e9a2e993e02eddcc2edd90cdd4fec14b0c6e59000 + 1659
‘46eb8a72e26d849f9b755c39f0caa08013b61b4592f02ab446c4fa1b498d4000’
```

What is the Blockchain

The blockchain is basically a linked list which contains data and a hash pointer to the previous block, hence creating a persistent, ever-growing, "blockchain" that constantly updates to represent the latest state of the ledger.

What is a hash pointer? **A hash pointer is similar to a pointer, but instead of just containing the address of the previous block it contains the hash of the data inside the previous block**.



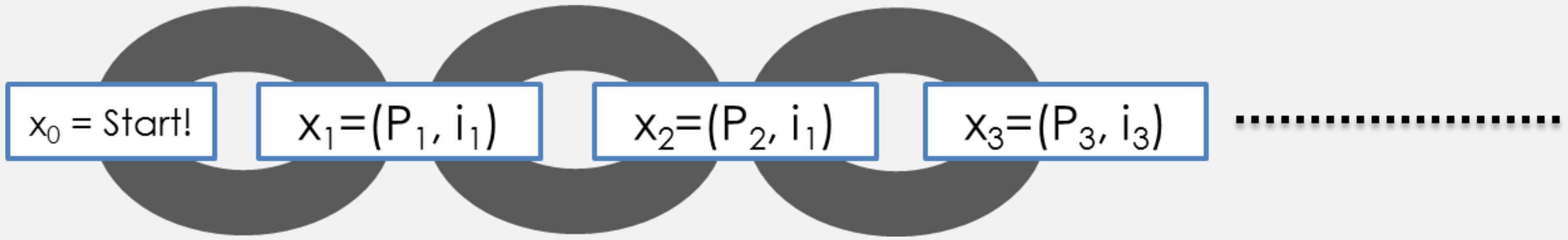


What is the Blockchain

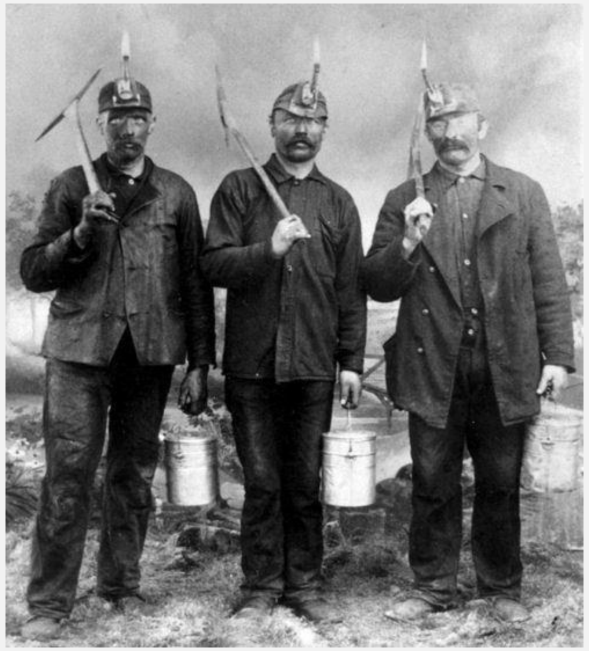
The blockchain is a decentralized, distributed and public digital ledger that is used to record information across many computers so that any involved record cannot be altered retroactively, without the alteration of all subsequent blocks.

Key concepts are:

- **Decentralization:** the information is not stored in a privileged central server but in all the individual computers of each participant
- **Immutability:** no participant can modified a transaction after it has been recorded on the ledger
- **Cryptography:** integrity and security of the information on the blockchain are ensured with cryptographic functions
- **Transparency:** participants know where the assets came from and how its ownership has changed over time



Proof of work (mining)



When you want to set a transaction this is what happens behind the scenes:

- Transactions are bundled together into what we call a block;
- Miners verify that transactions within each block are legitimate;
- miners solve a mathematical puzzle known as proof-of-work problem;
- A reward is given to the first miner who solves each blocks problem;
- Verified transactions are stored in the public blockchain

This “mathematical puzzle” has a key feature: **asymmetry**: hard on the requester side but easy to check for the network

All the network miners compete to be the first to find a solution for the mathematical problem that concerns the candidate block**, a problem that cannot be solved in other ways than through brute force** so that essentially requires a huge number of attempts.

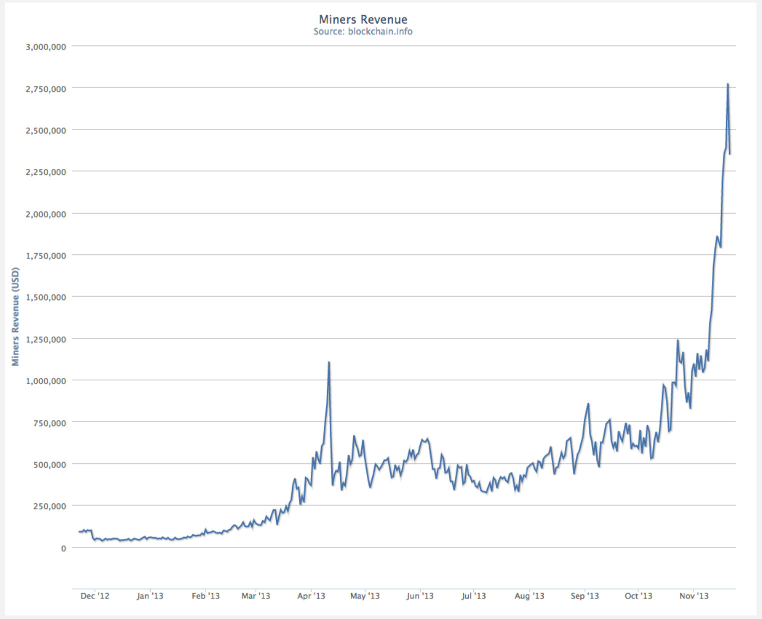
When a miner finally finds the right solution, he/she announces it to the whole network at the same time, receiving a cryptocurrency prize (the reward) provided by the protocol.

Reward

mining process is an operation of inverse hashing: it determines a number (nonce), so the cryptographic hash algorithm of block data results in less than a given threshold.

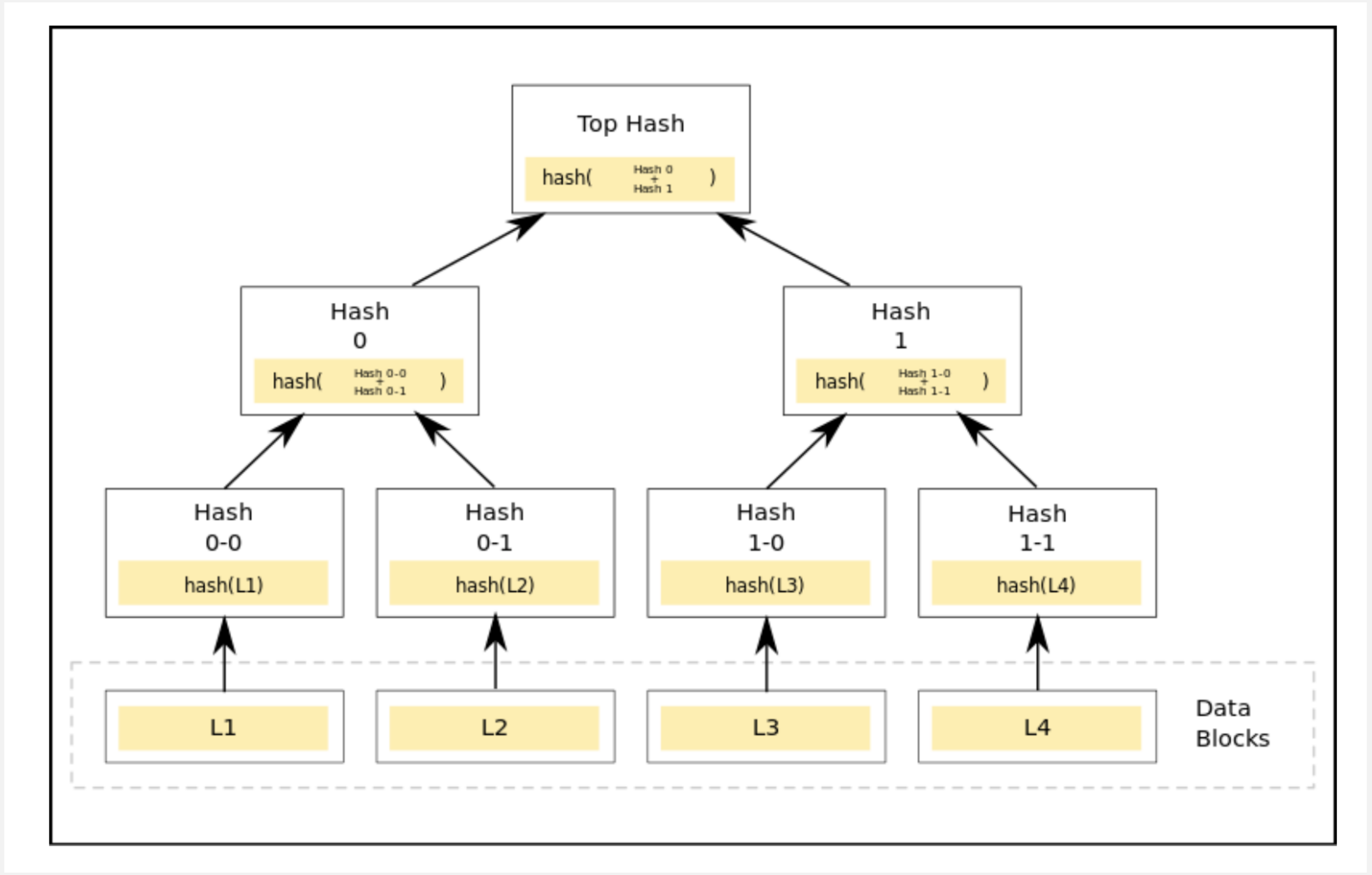
This threshold, called difficulty, is what determines the competitive nature of mining: more computing power is added to the network, the higher this parameter increases, increasing also the average number of calculations needed to create a new block.

This method also increases the cost of the block creation, pushing miners to improve the efficiency of their mining systems to maintain a positive economic balance.



Merkle Tree

A Merkle tree is a type of binary tree:
- each node is the hash of its two children
- a single root node, also formed from the hash of its two children, representing the "top" of the tree.

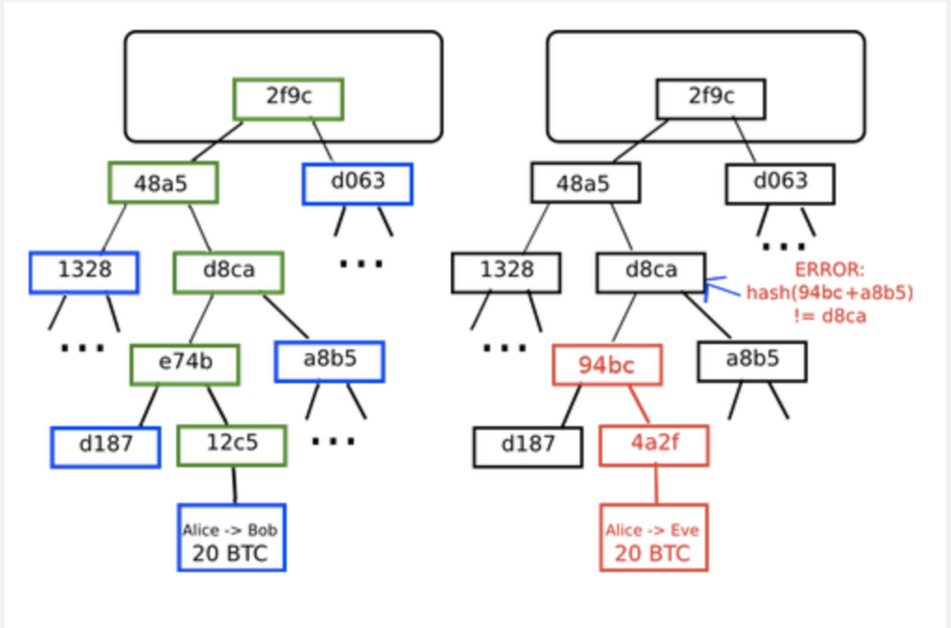


Scalability - Merkle Trees

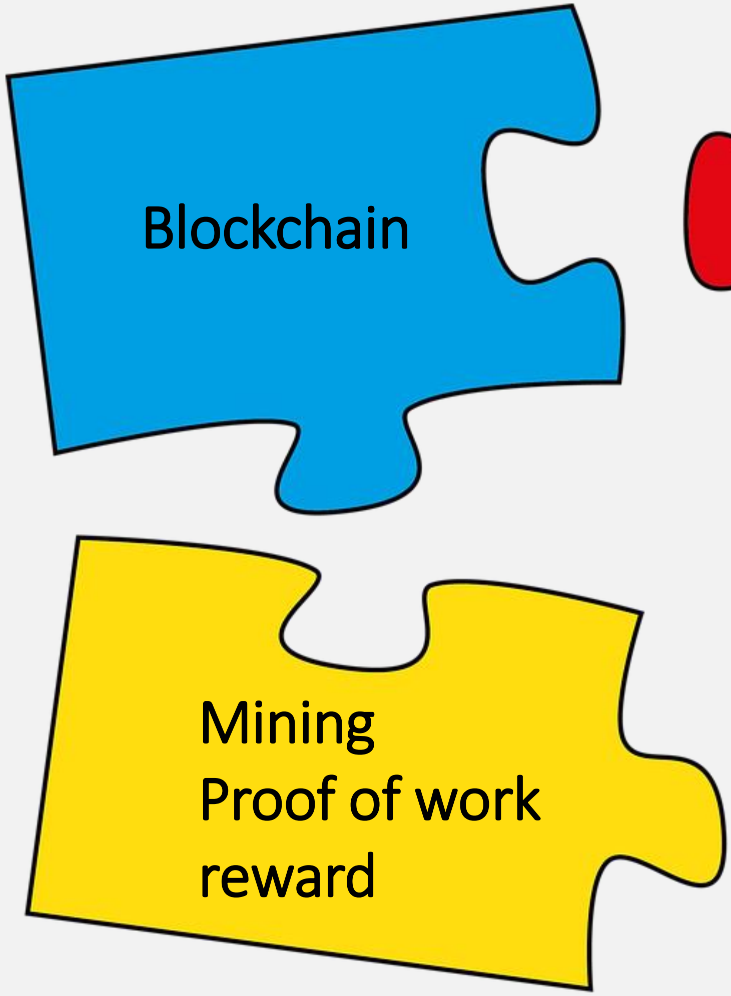
a node can download just the header of a block from one source, the small part of the tree relevant to them from another source, and still be assured that all of the data is correct.

1) if a malicious user attempts to swap in a fake transaction into the bottom of a Merkle tree, this change will cause a change in the node above, and then a change in the node above that, finally changing the root of the tree and therefore the hash of the block, causing the protocol to register it as a completely different block

2) "light nodes" downloads the block headers, verify the proof of work on the block headers, and then download only the "branches" associated with transactions that are relevant to them. This allows light nodes to determine with a strong guarantee of security what the status of any Bitcoin transaction, and their current balance is while downloading only a very small portion of the entire blockchain.



So much things... ☹



Be patient! Let’s see a real case:
BITCOIN

Bitcoin

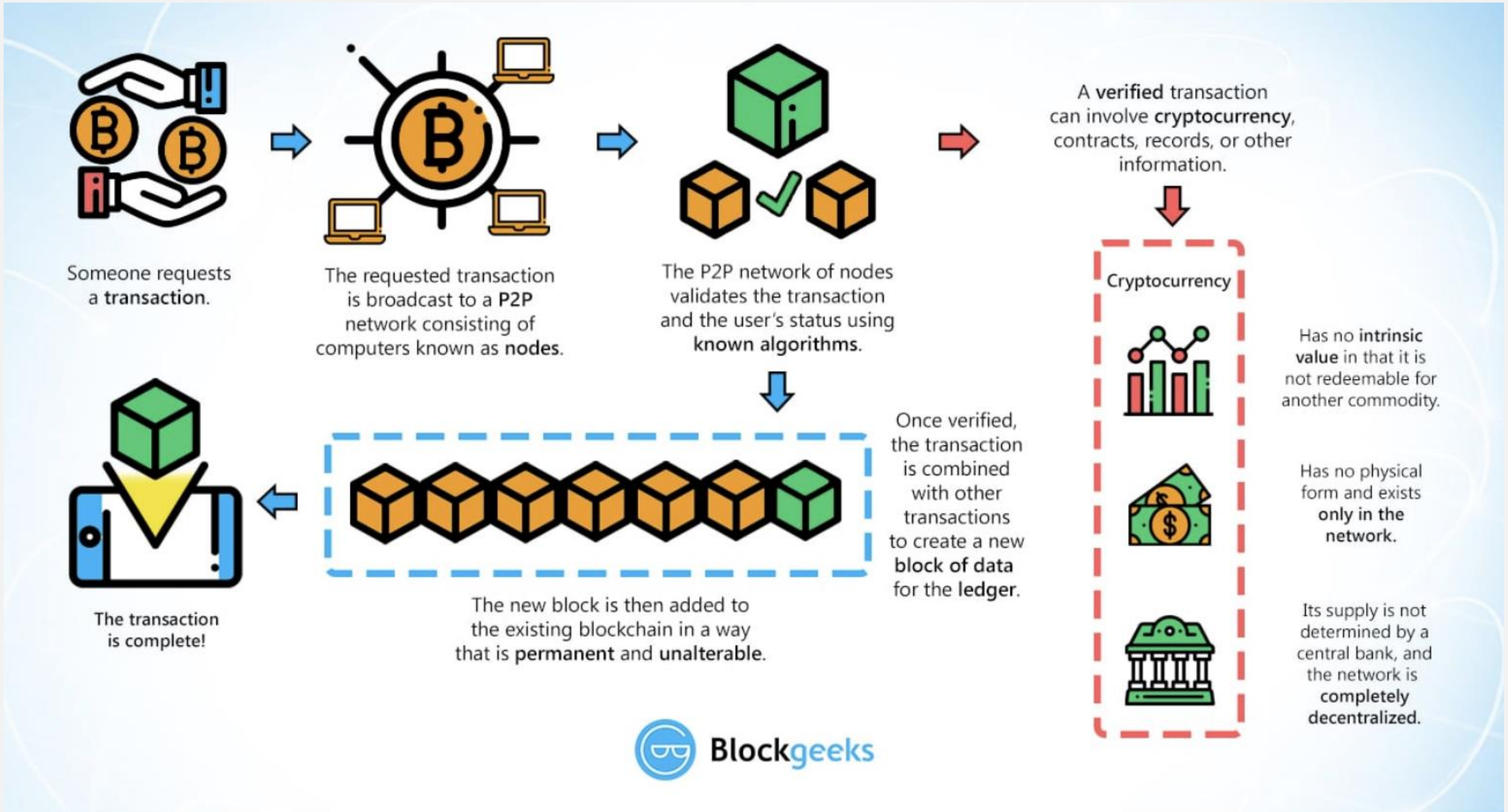
Blockchain technology was first introduced in a whitepaper entitled:
"Bitcoin: A Peer-to-Peer Electronic Cash System," by Satoshi Nakamoto in 2008.

Bitcoin is a collection of concepts and technologies creating a digital money ecosystem.

- A decentralized peer-to-peer network (the bitcoin protocol)
- A public transaction ledger (the blockchain) (roughly one block created every ten minutes)
- A decentralized mathematical and deterministic currency issuance (distributed mining)
- A decentralized transaction verification system (transaction script)

Bitcoin

Satoshi Nakamoto: Bitcoin: A Peer-to-Peer Electronic Cash System

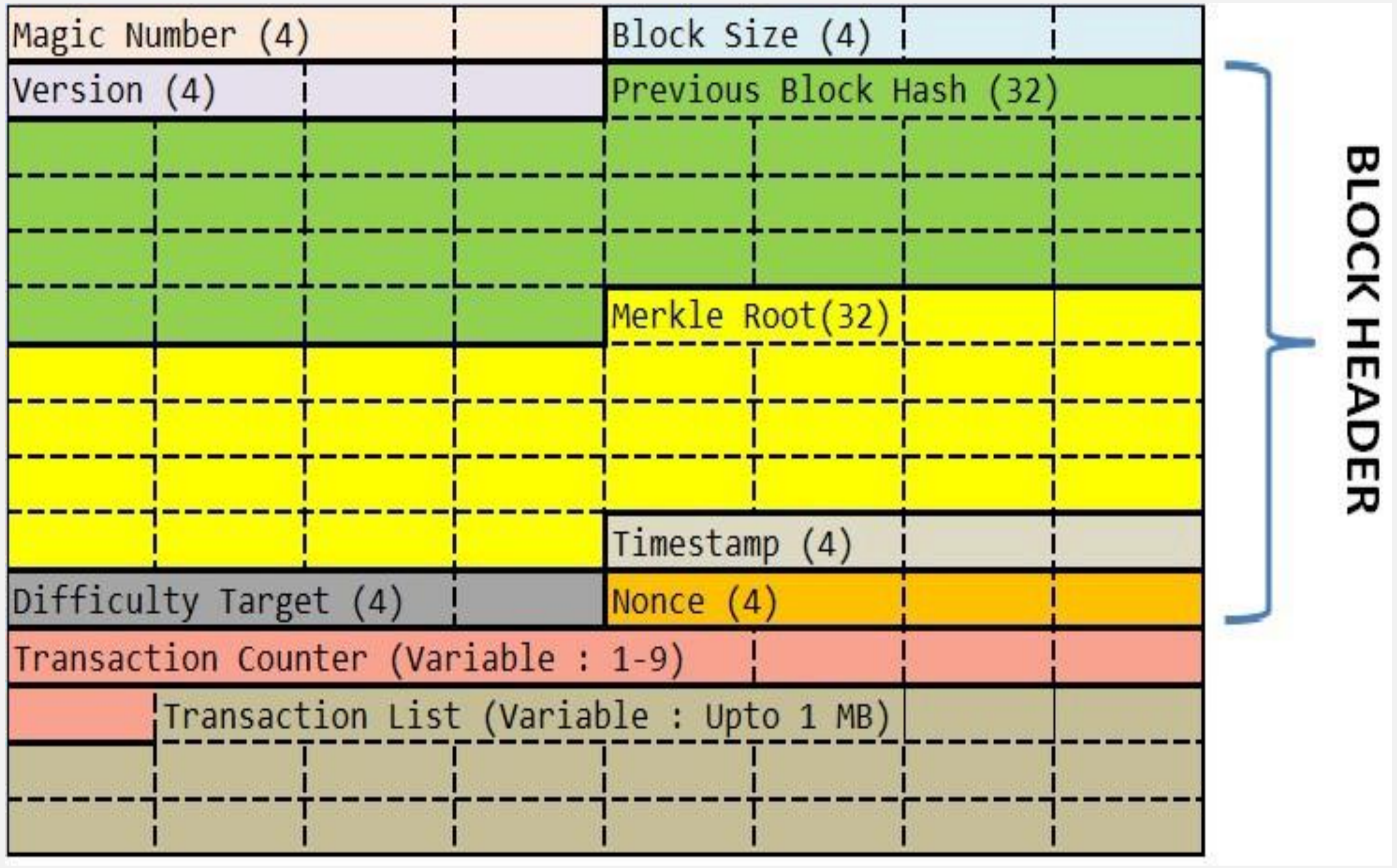

Someone requests a **transaction**.
The requested transaction is broadcast to a **P2P** network consisting of computers known as **nodes**.
The P2P network of nodes validates the transaction and the user's status using **known algorithms**.
A **verified** transaction can involve **cryptocurrency**, contracts, records, or other information.
Cryptocurrency
Has no **intrinsic value** in that it is not redeemable for another commodity.
Has no physical form and exists **only in the network**.
Its supply is not determined by a central bank, and the network is **completely decentralized**.
Once verified, the transaction is combined with other transactions to create a new **block of data** for the **ledger**.
The new block is then added to the existing blockchain in a way that is **permanent** and **unalterable**.
The transaction is complete!
Blockgeeks

Blockchain

A block header contains:

Magic: 0xD9B4BEF9
The protocol version number.
Time: the current timestamp.
The current difficult target.
Hash of the previous block.
The nonce.
The Merkle Root.

4byte timestamp = unix timestamp («seconds» precision) =
What if you check all the available nonce in less than 1 second???



difficulty

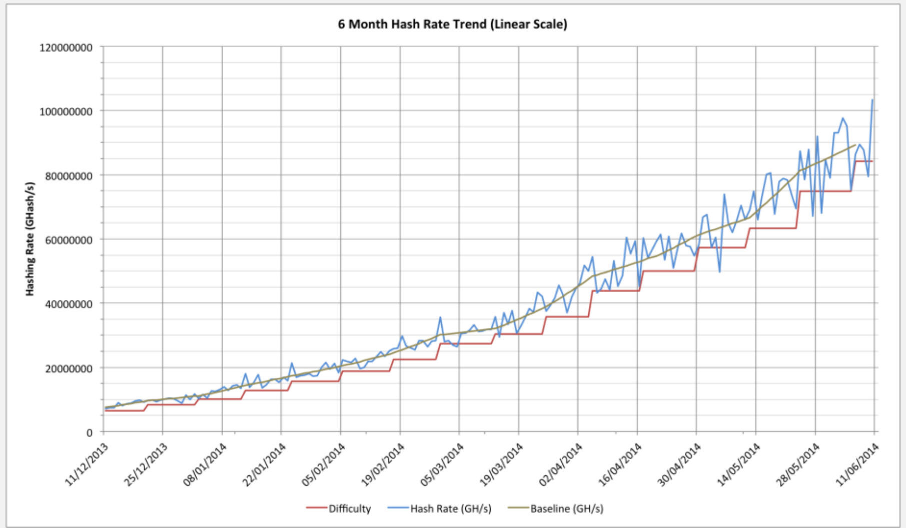
Mining is like a game, you solve the puzzle and you get rewards.

Setting higher difficulty makes that puzzle much harder to solve and hence more time-consuming.

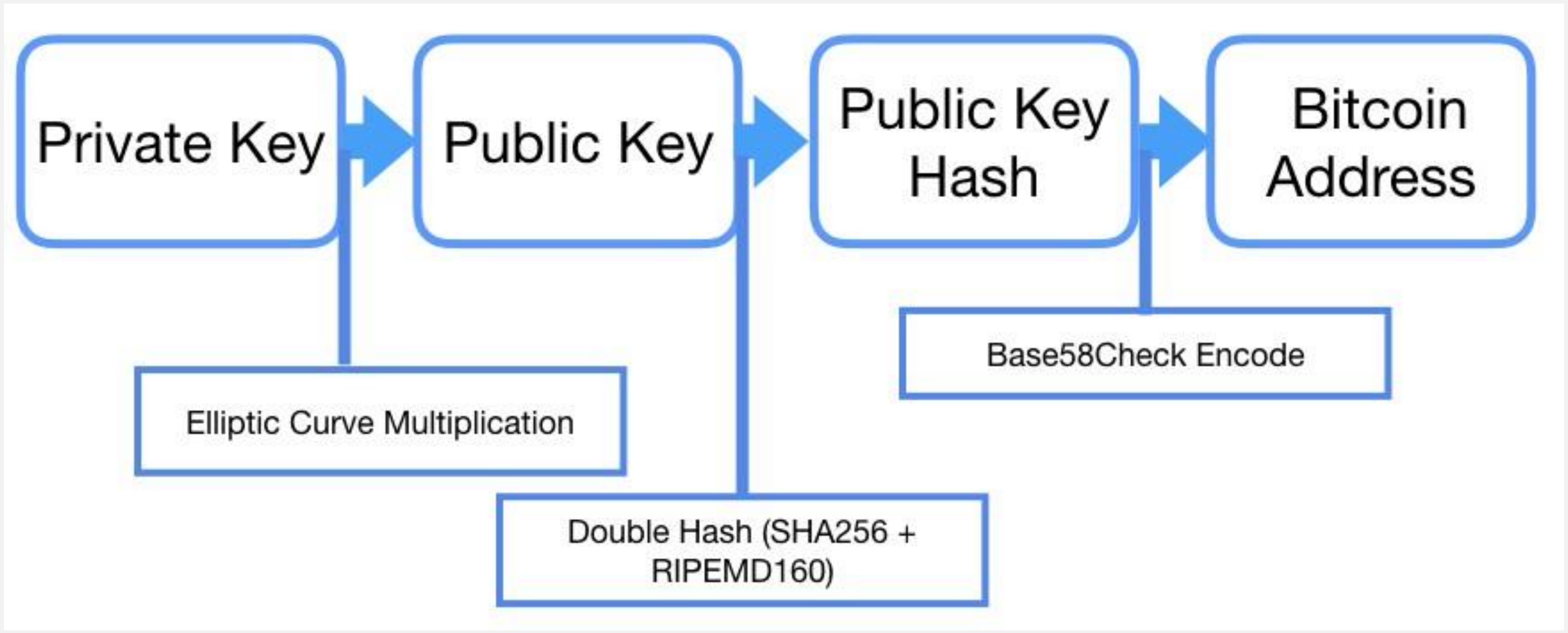
the difficulty target is a 64-character string (which is the same as a SHA-256 output) which begins with a bunch of zeroes.
A number of zeroes increases as the difficulty level increases.
The difficulty level changes after every 2016th block to produce a block every (about) 10 minutes.

The miner of every block is entitled to include a transaction giving themselves ~~12.5 BTC~~ out of nowhere (MINING).
Additionally, there are "transaction fee".

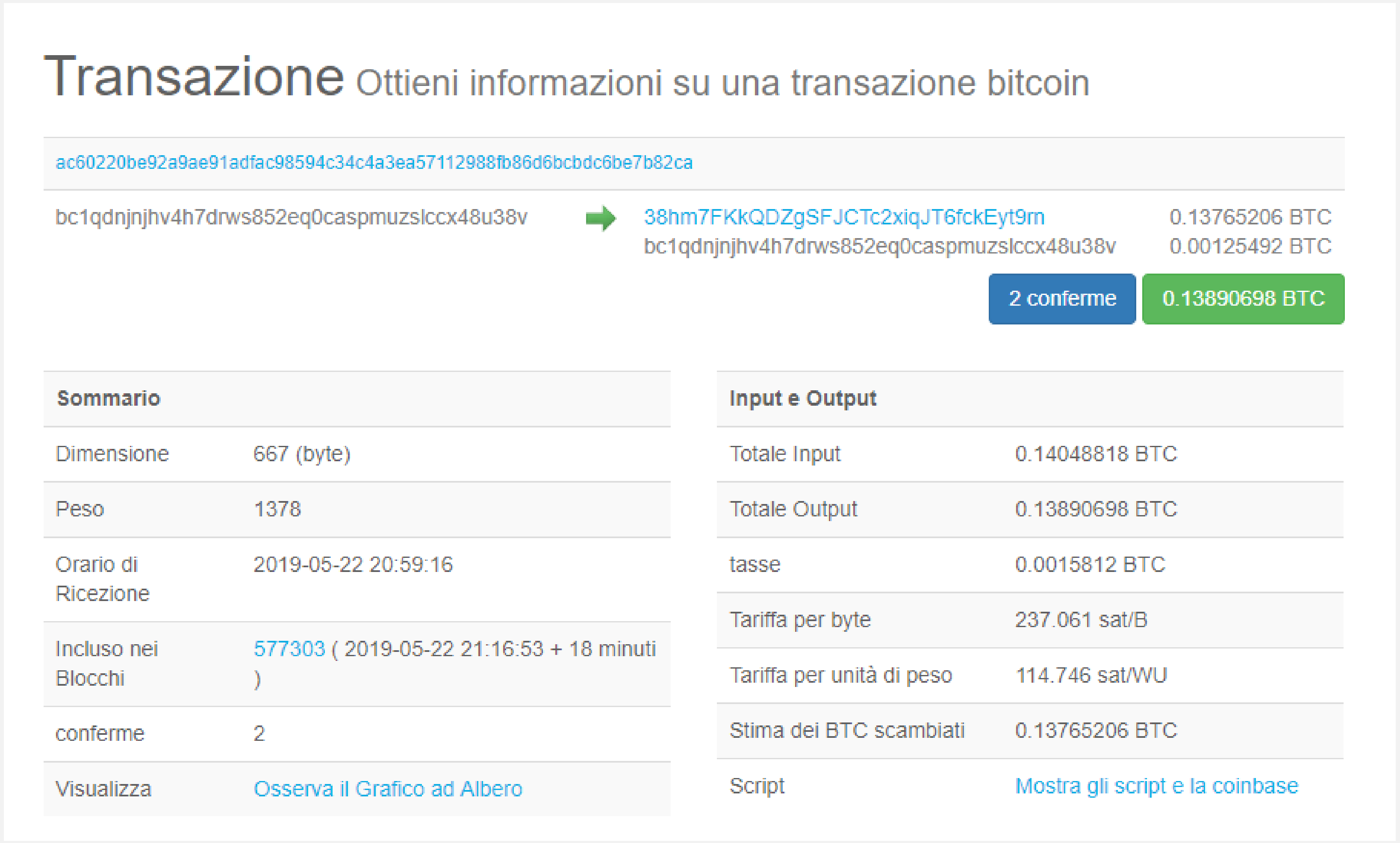
This is just changed (may 11 2020) to 6.25BTC («halving»)



BTC Address


Private Key
Public Key
Public Key Hash
Bitcoin Address
Elliptic Curve Multiplication
Double Hash (SHA256 + RIPEMD160)
Base58Check Encode

BTC transaction



Transazione Ottieni informazioni su una transazione bitcoin

ac60220be92a9ae91adfac98594c34c4a3ea57112988fb86d6bcbdc6be7b82ca

bc1qdnjnjhv4h7drws852eq0caspmuzslccx48u38v → 38hm7FKkQDZgSFJCTc2xiqJT6fckEyt9m 0.13765206 BTC
bc1qdnjnjhv4h7drws852eq0caspmuzslccx48u38v 0.00125492 BTC

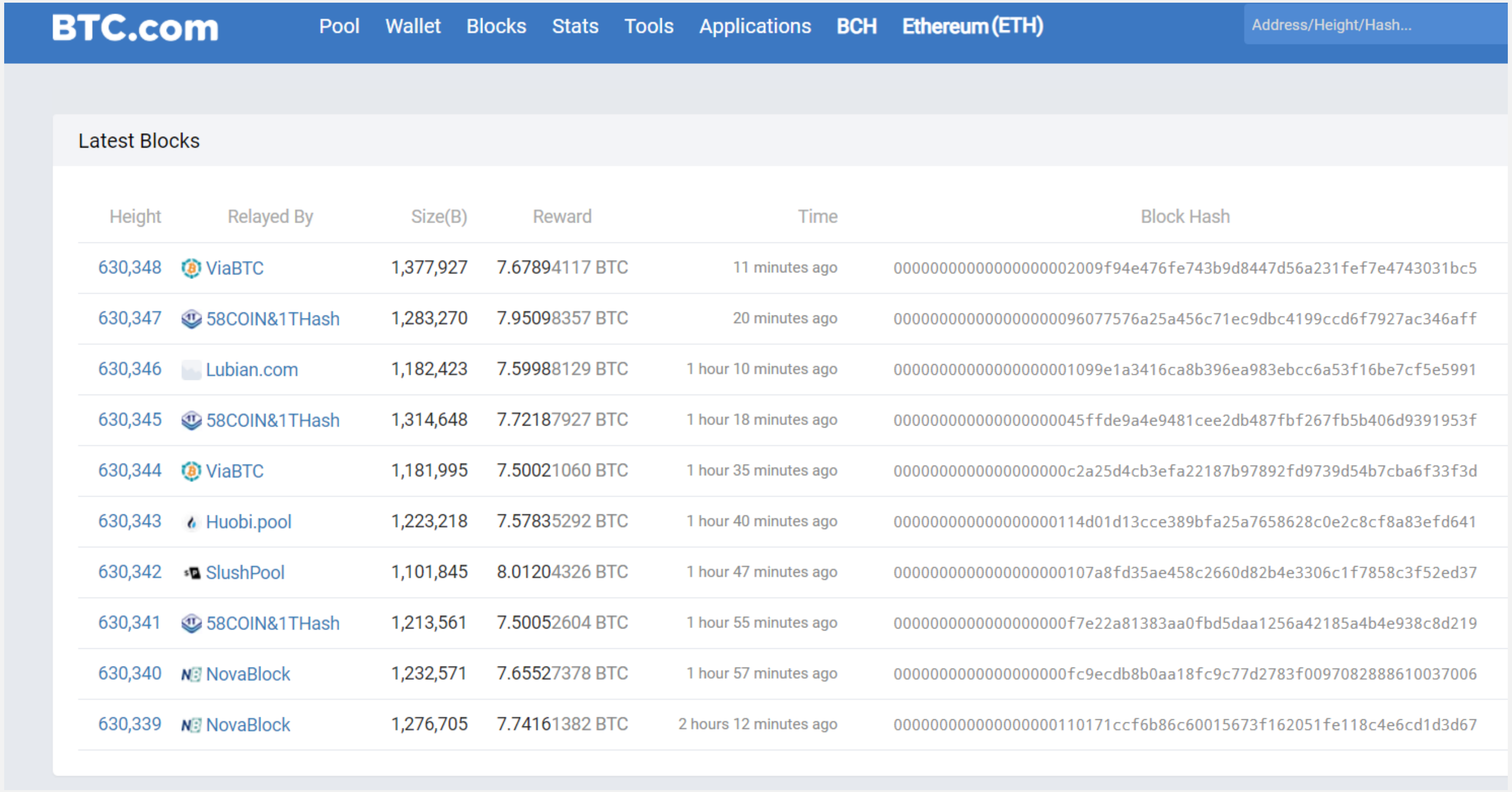
2 conferme 0.13890698 BTC

Sommario	
Dimensione	667 (byte)
Peso	1378
Orario di Ricezione	2019-05-22 20:59:16
Incluso nei Blocchi	577303 (2019-05-22 21:16:53 + 18 minuti)
conferme	2
Visualizza	Osserva il Grafico ad Albero

Input e Output	
Totale Input	0.14048818 BTC
Totale Output	0.13890698 BTC
tasse	0.0015812 BTC
Tariffa per byte	237.061 sat/B
Tariffa per unità di peso	114.746 sat/WU
Stima dei BTC scambiati	0.13765206 BTC
Script	Mostra gli script e la coinbase

Way more complex than this…

BTC - BlockChain



Latest Blocks

Height	Relayed By	Size(B)	Reward	Time	Block Hash
630,348	ViaBTC	1,377,927	7.67894117 BTC	11 minutes ago	00000000000000000002009f94e476fe743b9d8447d56a231fef7e4743031bc5
630,347	58COIN&1THash	1,283,270	7.95098357 BTC	20 minutes ago	000000000000000000096077576a25a456c71ec9dbc4199ccd6f7927ac346aff
630,346	Lubian.com	1,182,423	7.59988129 BTC	1 hour 10 minutes ago	00000000000000000001099e1a3416ca8b396ea983ebcc6a53f16be7cf5e5991
630,345	58COIN&1THash	1,314,648	7.72187927 BTC	1 hour 18 minutes ago	000000000000000000045ffde9a4e9481cee2db487fbf267fb5b406d9391953f
630,344	ViaBTC	1,181,995	7.50021060 BTC	1 hour 35 minutes ago	0000000000000000000c2a25d4cb3efa22187b97892fd9739d54b7cba6f33f3d
630,343	Huobi.pool	1,223,218	7.57835292 BTC	1 hour 40 minutes ago	000000000000000000114d01d13cce389bfa25a7658628c0e2c8cf8a83efd641
630,342	SlushPool	1,101,845	8.01204326 BTC	1 hour 47 minutes ago	0000000000000000000107a8fd35ae458c2660d82b4e3306c1f7858c3f52ed37
630,341	58COIN&1THash	1,213,561	7.50052604 BTC	1 hour 55 minutes ago	0000000000000000000f7e22a81383aa0fbd5daa1256a42185a4b4e938c8d219
630,340	NovaBlock	1,232,571	7.65527378 BTC	1 hour 57 minutes ago	0000000000000000000fc9ecdb8b0aa18fc9c77d2783f0097082888610037006
630,339	NovaBlock	1,276,705	7.74161382 BTC	2 hours 12 minutes ago	000000000000000000110171ccf6b86c60015673f162051fe118c4e6cd1d3d67

Crypto Puzzles – mining bitcoin

When the Bitcoin mining software wants to add a new block to the blockchain, this is the procedure it follows.

Whenever a new block arrives, all the contents of the blocks are first hashed.

If the hash is less than the difficulty target, then it is added to the blockchain and everyone in the community acknowledges the new block.

However, it is not as simple as that. You will have to be extremely lucky to get a new block just like that. This is where the nonce comes in.

The nonce is an arbitrary string which is concatenated with the hash of the block.

After that this concatenated string is hashed again and compared to the difficulty level. If it is not less than the difficulty level, then the nonce is changed and this keeps on repeating a million times until finally, the requirements are met. When that happens the block is added to the blockchain.

Crypto Puzzles – mining bitcoin

Remember property number 6 of hash functions? The puzzle friendliness?
For every output "Y", if k is chosen from a distribution with high min-entropy it is infeasible to find an input x such that H(k|x) = Y.

So, when it comes to bitcoin mining:
K = Nonce
X = the hash of the block
Y = the difficulty target (hash of the block + nonce hashed togheter)

The entire process is completely random, there is no thought process behind the selection of the nonces. It is just pure brute-force where the software keep on randomly generating strings till they reach their goal.

The entire process follows the Proof Of Work protocol : The puzzle solving should be difficult.

Checking the answer should, however, be easy for everyone. This is done to make sure that no underhanded methods were used to solve the problem.

Mining bitcoin 2009 - CPU

the first bitcoin miners used standard multi-core CPUs

If you had a couple computers lying around with decent specs you could have earned about five euros a day.

The difficulty of mining was so low then it was worth it for hobbyists and crypto nerds to participate.

Mining bitcoin 2010 - GPU

in October 2010 the code for mining bitcoin with GPUs was released to the general public.

Mining bitcoin 2011 - FPGA

mining difficulty continued to rise, and with it, the power requirements would soon become too steep for your average hobbyist to make any money.

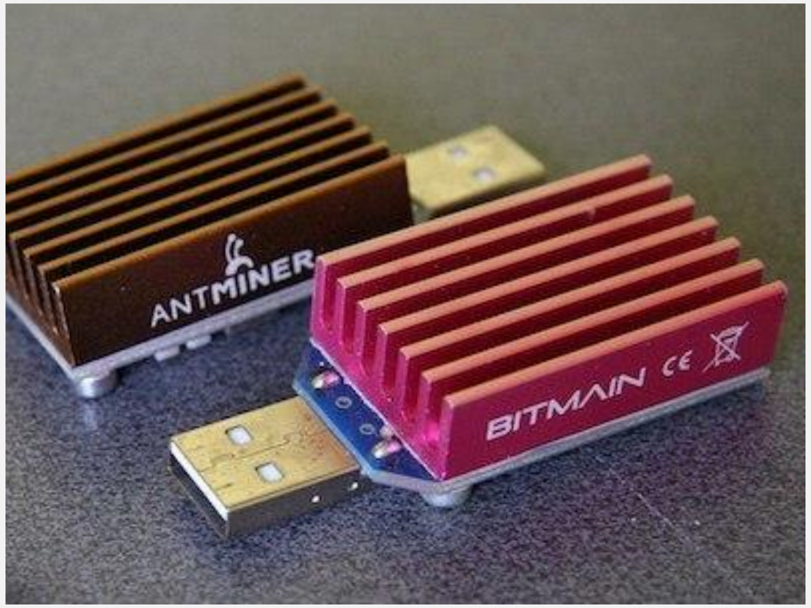
By June 2011 field-programmable gate arrays (FPGAs) were becoming all the rage.

The biggest draw to this hardware was the fact that it used three times less power than simple GPU setups to effectively accomplish the same task.

Mining bitcoin 2013 – cheap ASIC

Where FPGA requires tweaking after purchase (the field-programmable part of FPGA), an ASIC is created for a specific use, in this case mining cryptocurrency. This is why ASIC miners remain the standard.

330 Mh/s



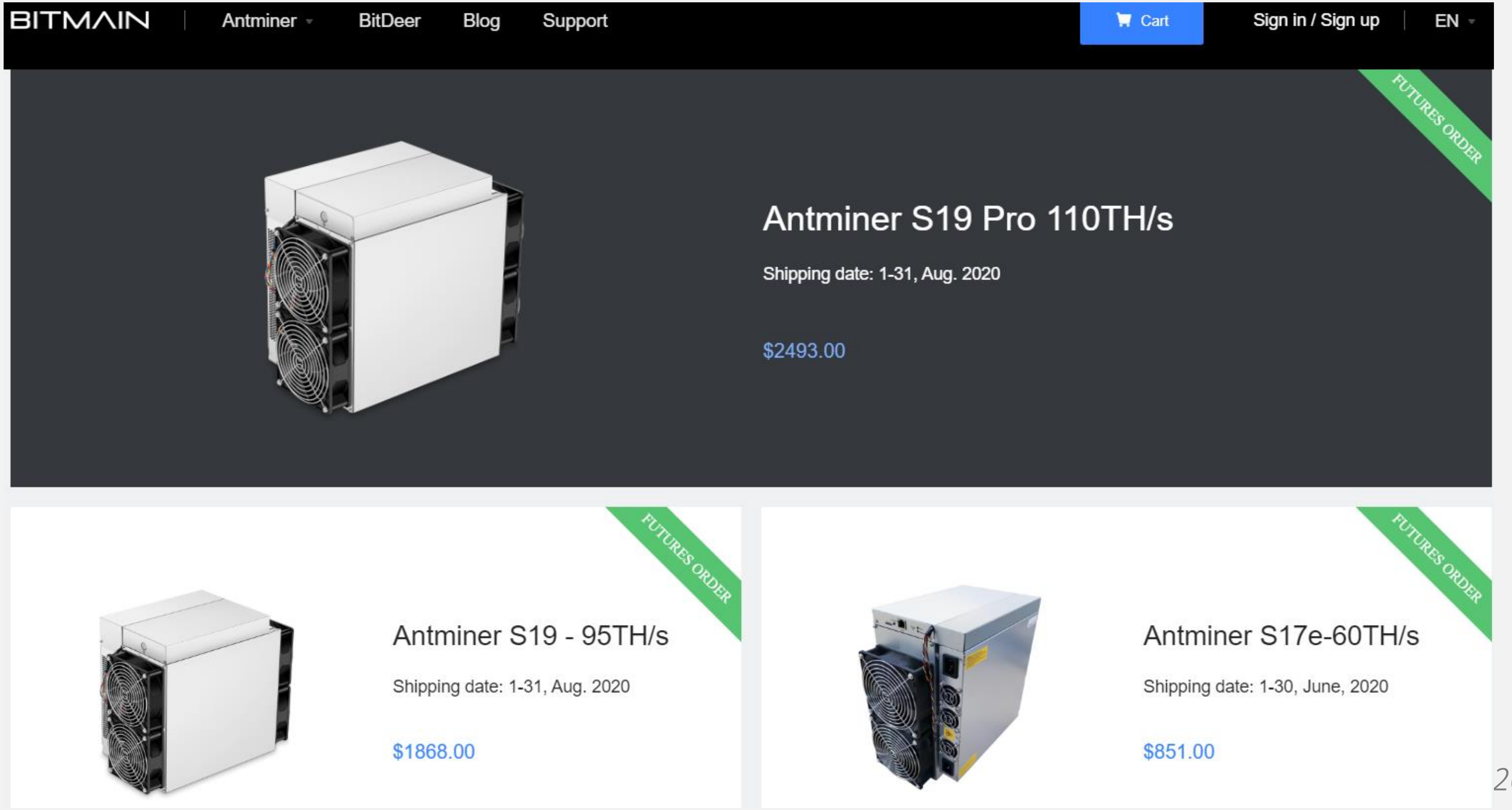
1.8 Gh/s

Mining bitcoin 2013 – mining pool

Mining bitcoin last year – expensive ASIC

14 Th/s

Mining bitcoin now – expensive ASIC



Antminer S19 Pro 110TH/s

Shipping date: 1-31, Aug. 2020

$2493.00

Antminer S19 - 95TH/s

Shipping date: 1-31, Aug. 2020

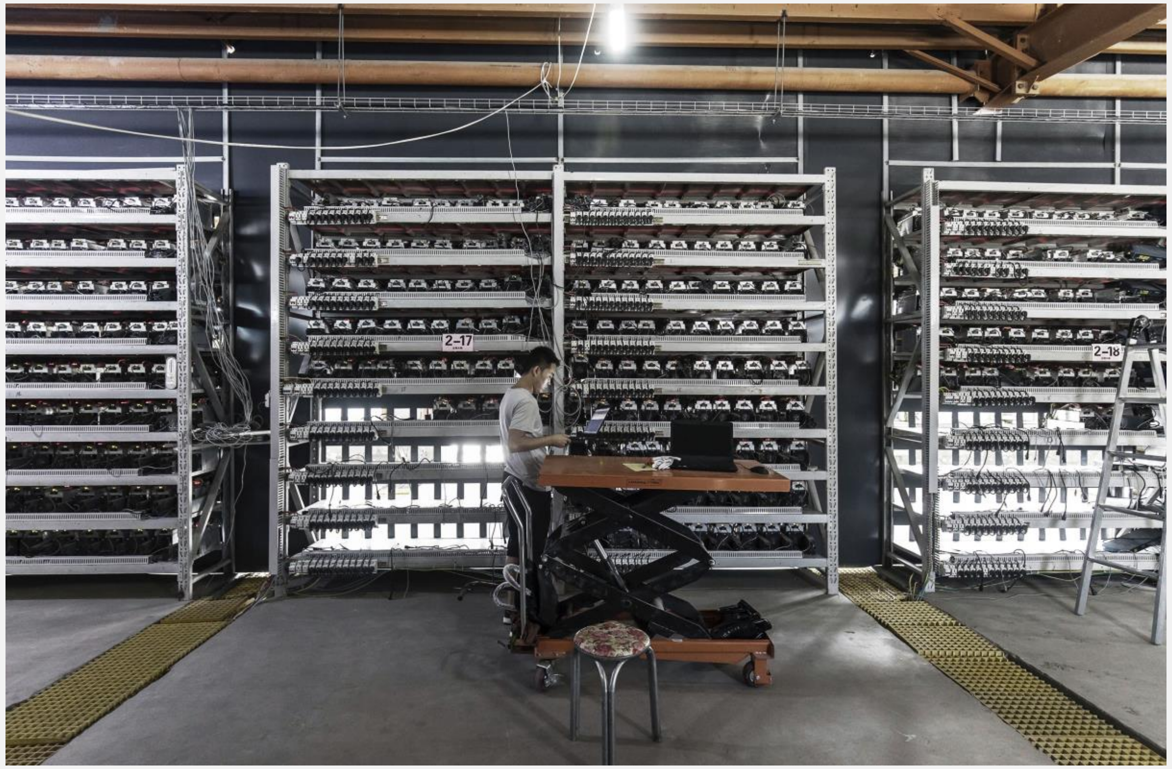
$1868.00

Antminer S17e-60TH/s

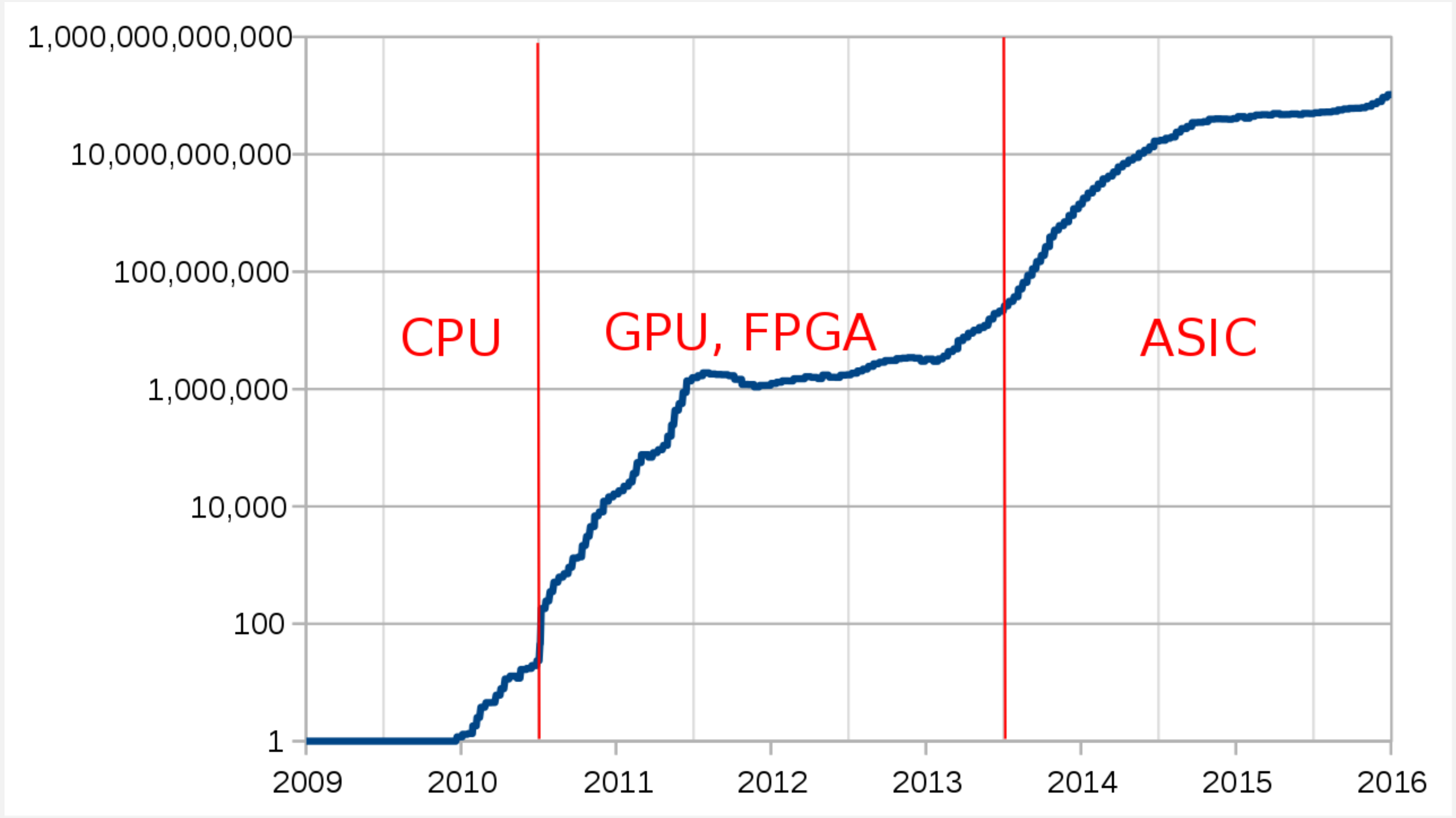
Shipping date: 1-30, June, 2020

$851.00

Mining bitcoin now



Mining bitcoin - difficulty



Bitcoin PIZZA

I'll pay 10,000 bitcoins for a couple of pizzas.. like maybe 2 large ones so I have some left over for the next day. I like having left over pizza to nibble on later. You can make the pizza yourself and bring it to my house or order it for me from a delivery place …

https://bitcointalk.org/index.php?topic=137.0

Eventually someone took him up on the offer and Hanyecz ended up eating a meal that, only some years later, would be worth $71 million.

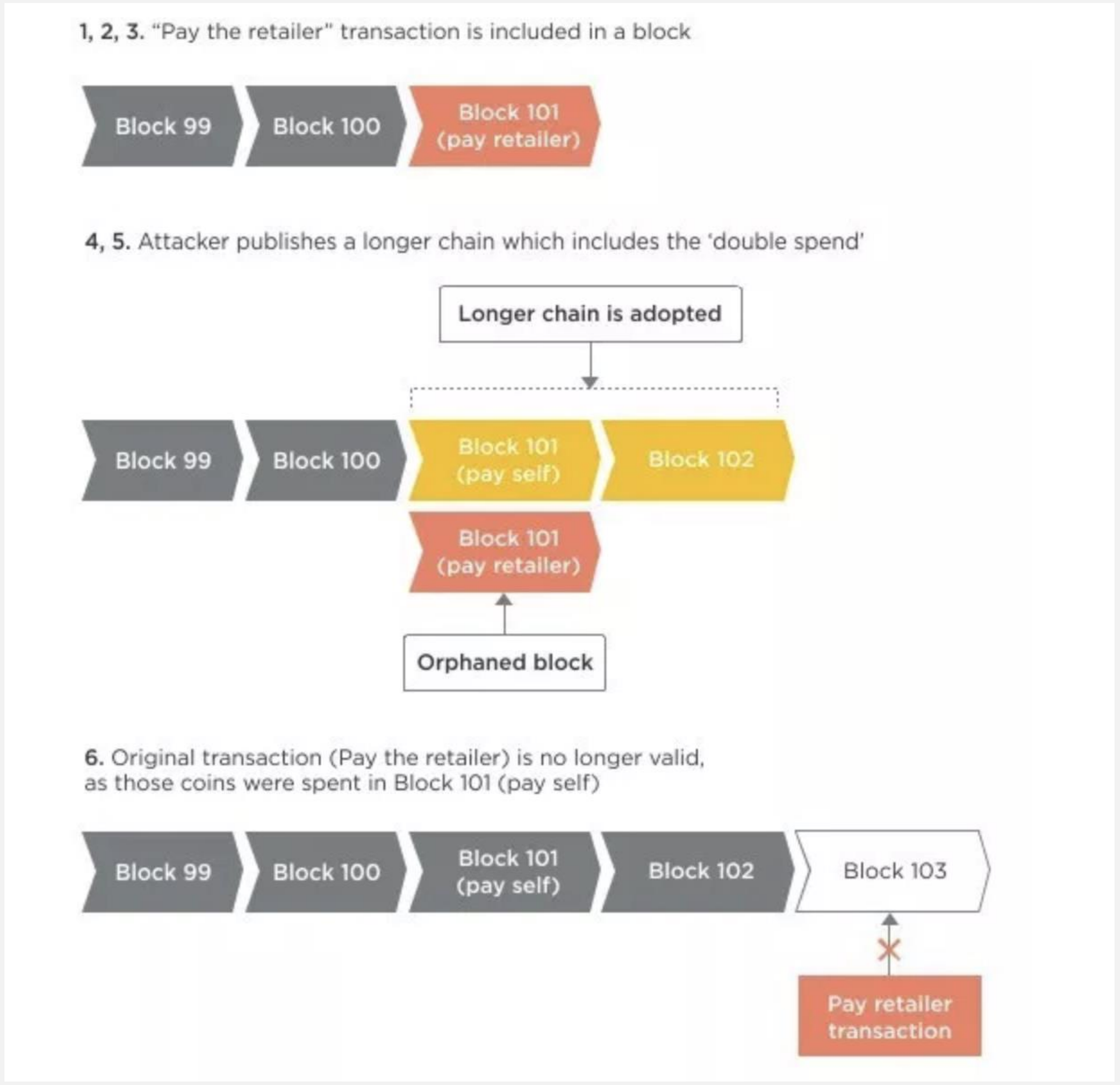
Every year, the crypto community celebrates 'Bitcoin Pizza Day' on May 22.

Double spending attack

Attacks the part lof the system not covered directly by cryptography: order of transactions.

Attacker strategy

- Send 100 BTC to a merchant/retailer in exchange for some product (Block 101-pay retailer);
- Wait for the delivery of the product
- Produce another transaction sending the same 100 BTC back to the attacker (Block 101, Pay self)
- Try to convince the network that this latter transaction was the one that came first.


1, 2, 3. "Pay the retailer" transaction is included in a block
Block 99
Block 100
Block 101 (pay retailer)
4, 5. Attacker publishes a longer chain which includes the 'double spend'
Longer chain is adopted
Block 99
Block 100
Block 101 (pay self)
Block 102
Block 101 (pay retailer)
Orphaned block
6. Original transaction (Pay the retailer) is no longer valid, as those coins were spent in Block 101 (pay self)
Block 99
Block 100
Block 101 (pay self)
Block 102
Block 103
Pay retailer transaction

Double spending attack

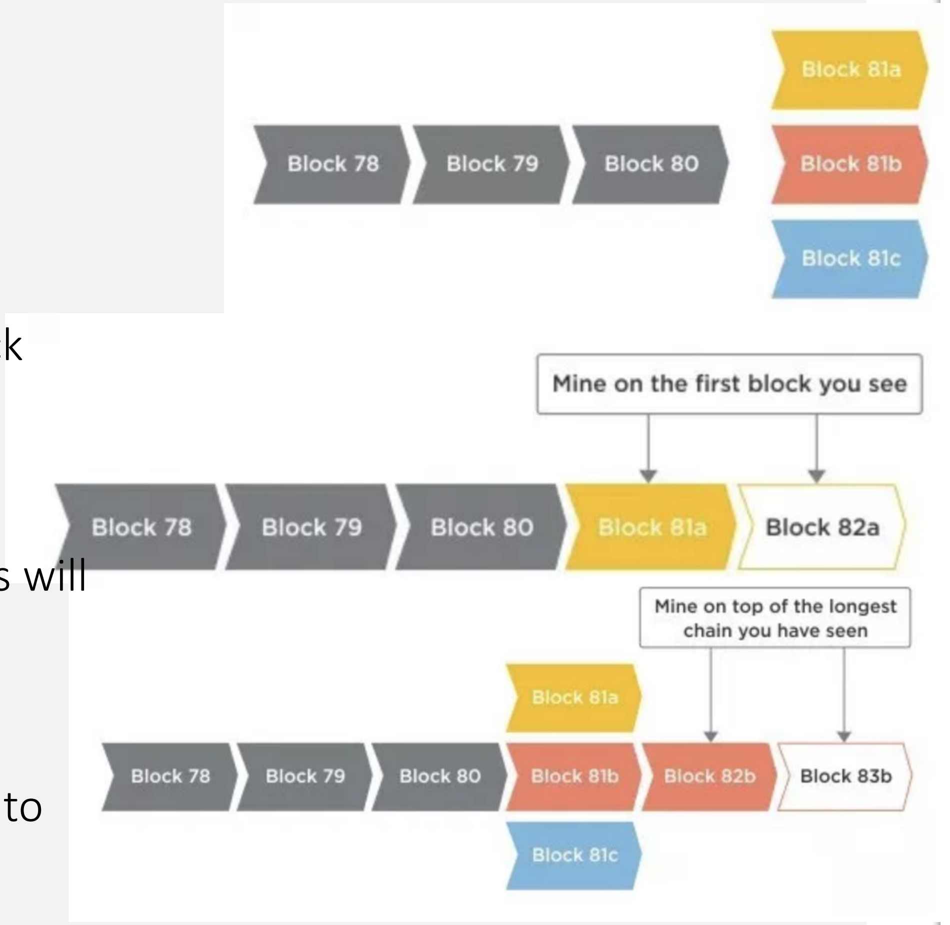
-The attacker creates another transaction sending the 100 BTC back to himself.

the transaction will not be processed as miners will notice that funds are already spent

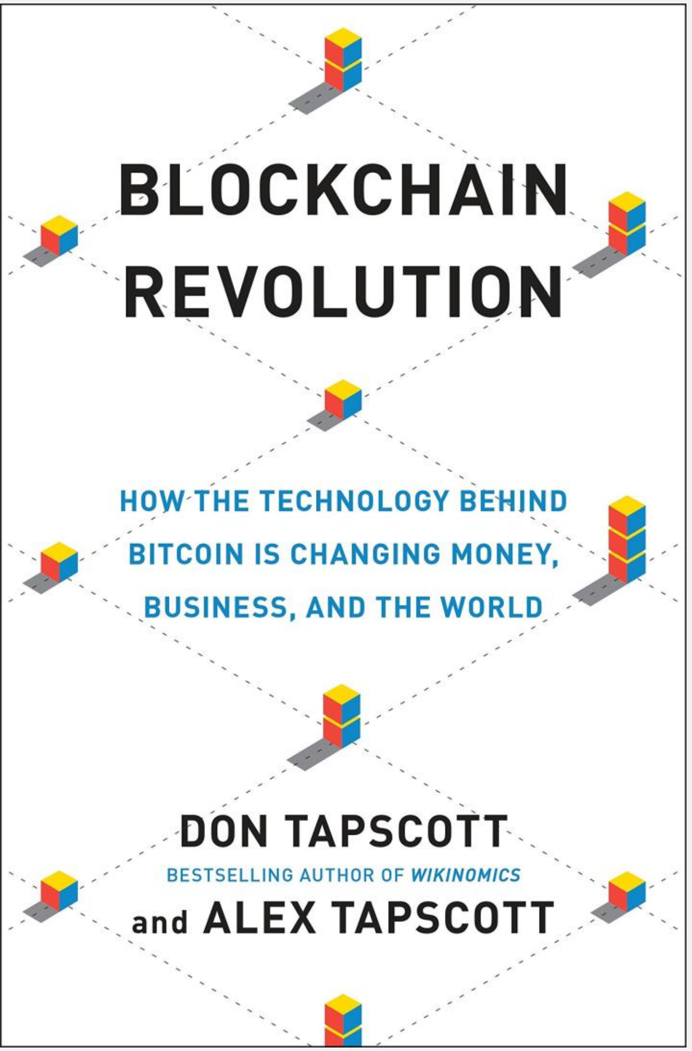
-the attacker creates a "fork" of the blockchain, by mining another version of block 270 pointing to the same block 269 as a parent but with the new transaction in place of the old one

-In case of a fork the longest blockchain is taken to be the truth: legitimate miners will work on the 275 chain while the attacker is working on the 270 chain.

- In order for the attacker to make his blockchain the longest, he would need to have more computational power than the rest of the network combined in order to catch up (hence, "51% attack").

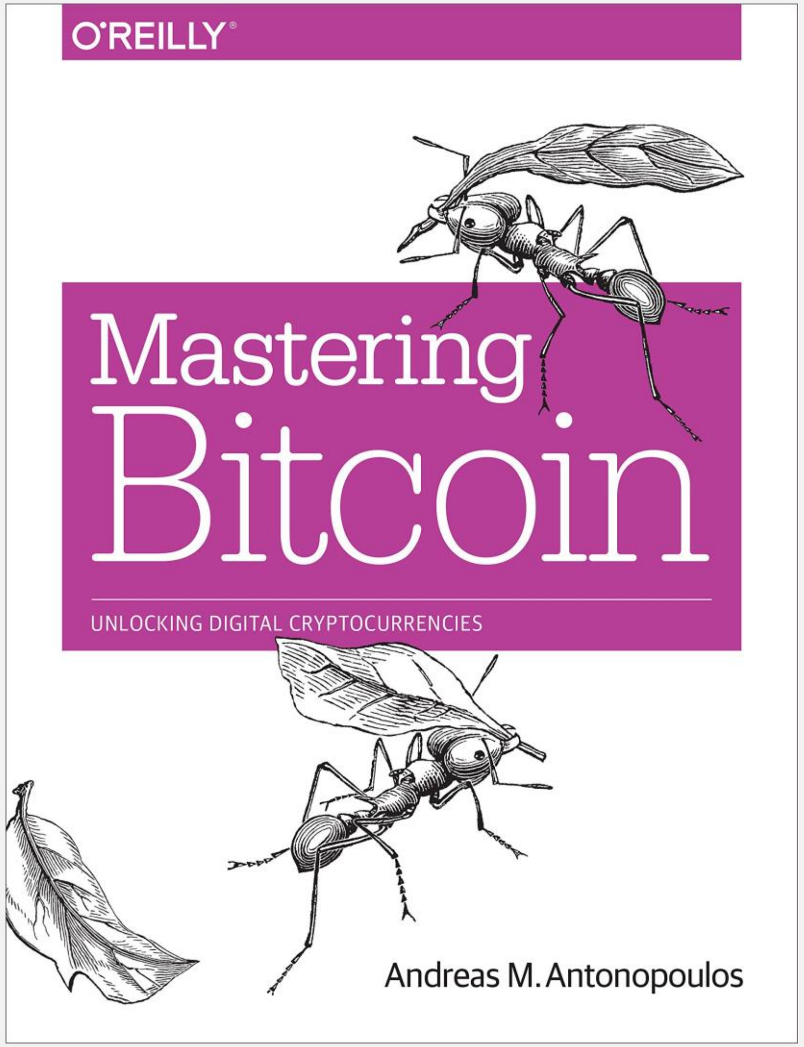


For more infos…


BLOCKCHAIN REVOLUTION

HOW THE TECHNOLOGY BEHIND BITCOIN IS CHANGING MONEY, BUSINESS, AND THE WORLD

DON TAPSCOTT
BESTSELLING AUTHOR OF *WIKINOMICS*
and ALEX TAPSCOTT


O'REILLY

Mastering Bitcoin

UNLOCKING DIGITAL CRYPTOCURRENCIES

Andreas M. Antonopoulos

https://github.com/bitcoinbook/bitcoinbook

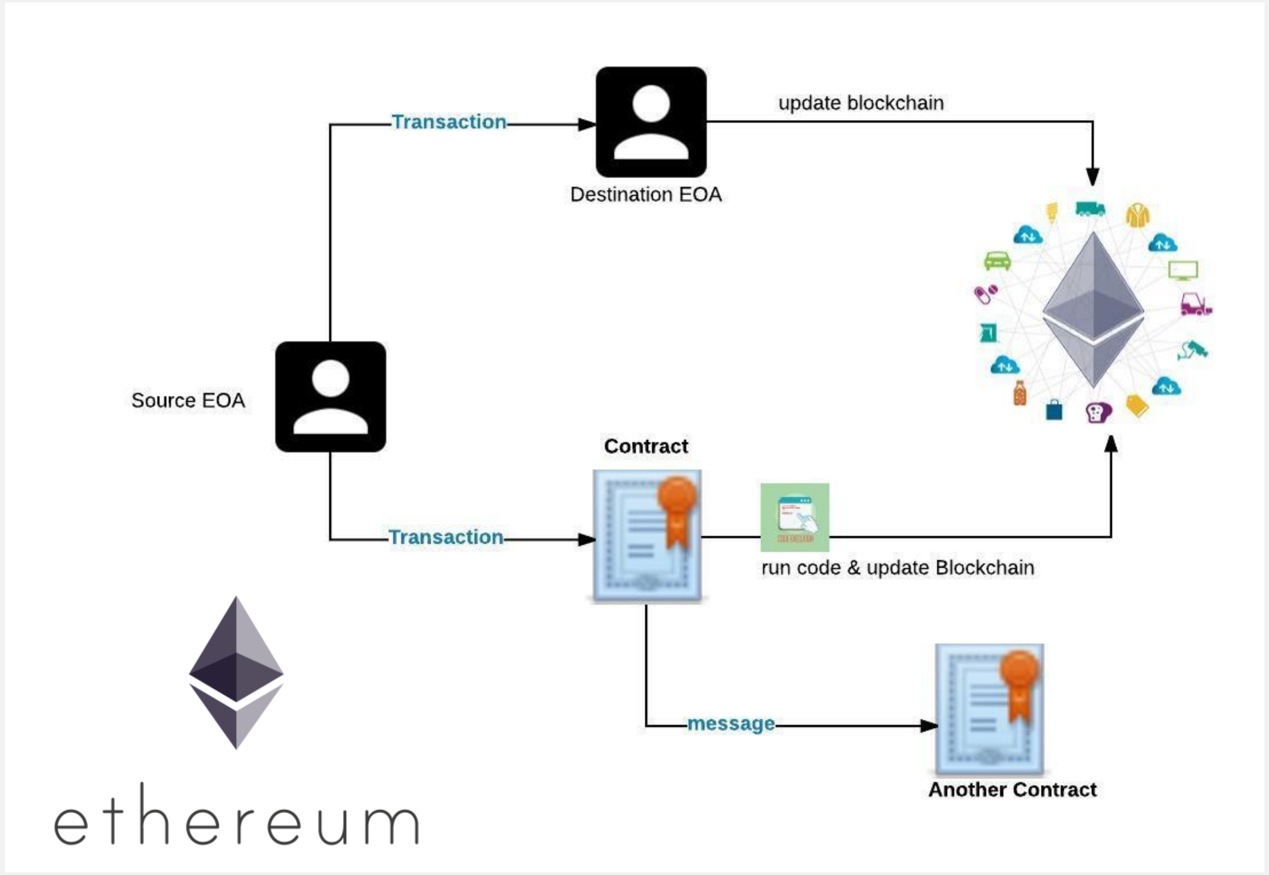
BlockChain + Smart Contracts: Ethereum

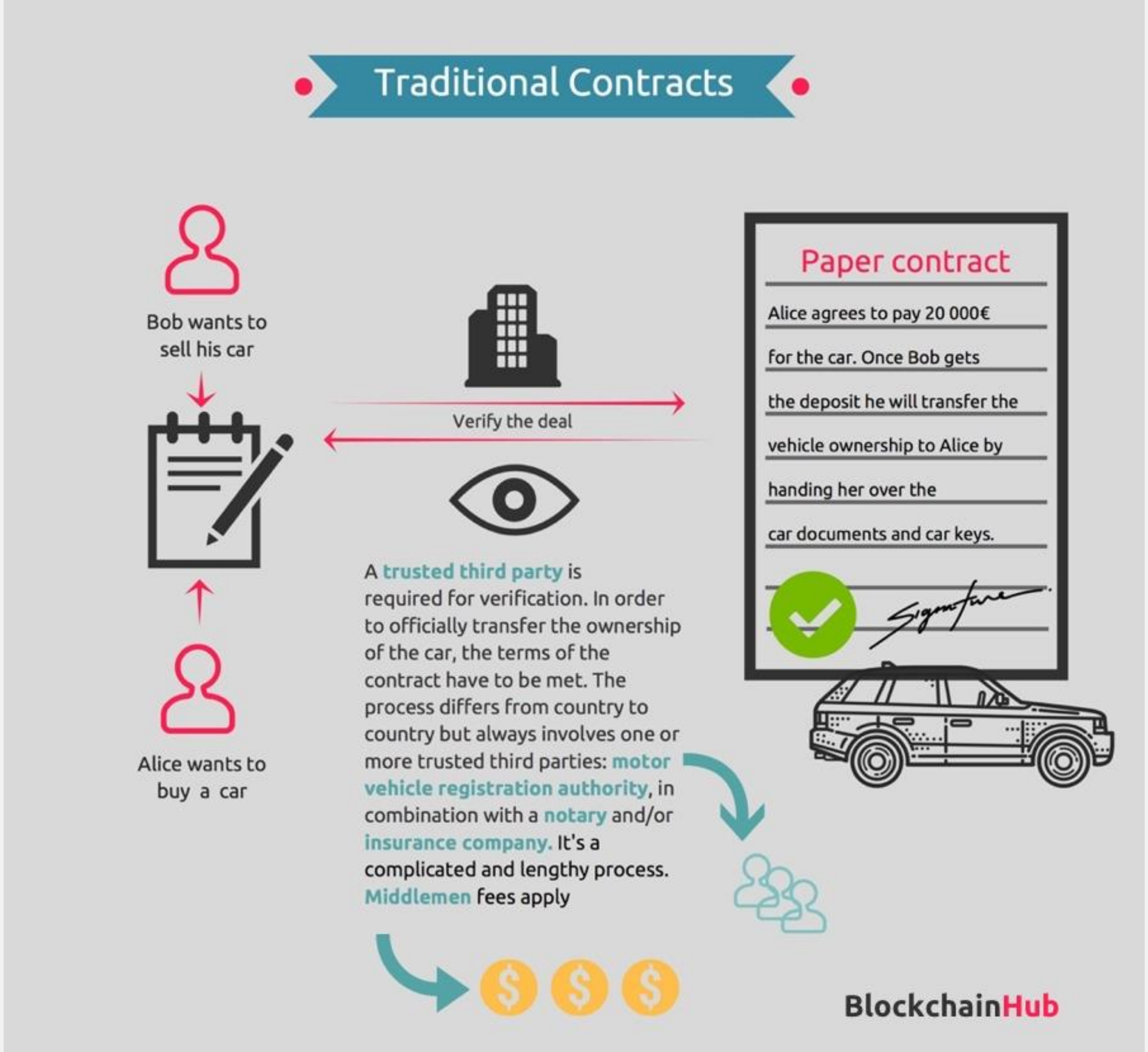
Ethereum is a blockchain with a built-in fully Turing-complete programming language that can be used to create «smart contracts»: a piece of code implementing arbitrary rules controlling the digital assets on the blockchain.

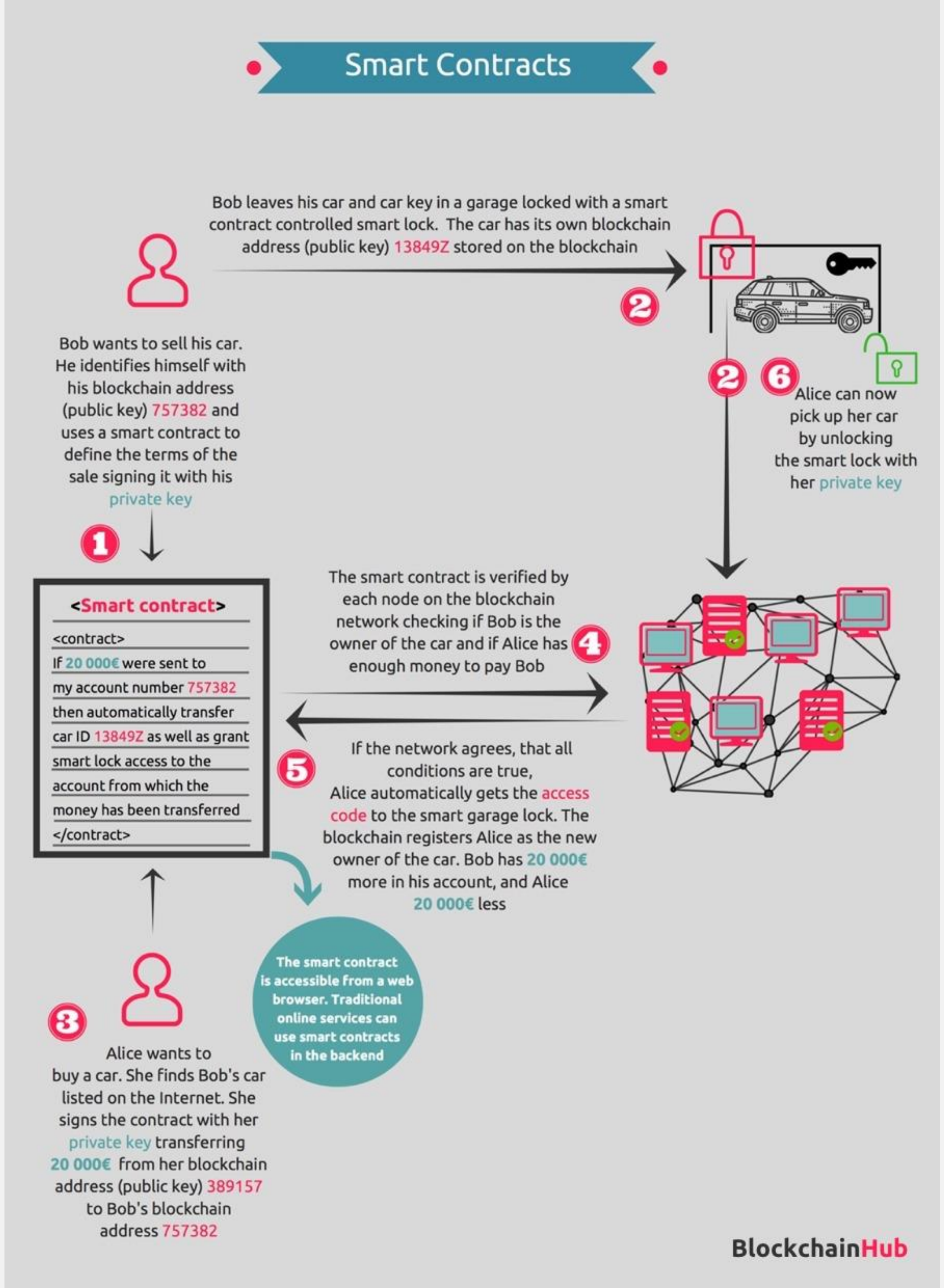
The structure of the ethereum blockchain is very similar to bitcoin's, in that it is a shared record of the entire transaction history. Every node on the network stores a copy of this history.

The big difference with ethereum is that **its nodes store the most recent state of each smart contract, in addition to all of the ether transactions.**

For each ethereum application, the network needs to keep track of the 'state', or the current information of all of these applications, including each user's balance, all the smart contract code and where it's all stored.




Traditional Contracts
Bob wants to sell his car
Verify the deal
Alice wants to buy a car
Paper contract
Alice agrees to pay 20 000€ for the car. Once Bob gets the deposit he will transfer the vehicle ownership to Alice by handing her over the car documents and car keys.
A trusted third party is required for verification. In order to officially transfer the ownership of the car, the terms of the contract have to be met. The process differs from country to country but always involves one or more trusted third parties: motor vehicle registration authority, in combination with a notary and/or insurance company. It's a complicated and lengthy process. Middlemen fees apply
BlockchainHub



Smart Contracts

Bob leaves his car and car key in a garage locked with a smart contract controlled smart lock. The car has its own blockchain address (public key) 13849Z stored on the blockchain

Bob wants to sell his car. He identifies himself with his blockchain address (public key) 757382 and uses a smart contract to define the terms of the sale signing it with his private key

<Smart contract>

```
<contract>
If 20 000€ were sent to
my account number 757382
then automatically transfer
car ID 13849Z as well as grant
smart lock access to the
account from which the
money has been transferred
</contract>
```

Alice wants to buy a car. She finds Bob's car listed on the Internet. She signs the contract with her private key transferring 20 000€ from her blockchain address (public key) 389157 to Bob's blockchain address 757382

The smart contract is accessible from a web browser. Traditional online services can use smart contracts in the backend

The smart contract is verified by each node on the blockchain network checking if Bob is the owner of the car and if Alice has enough money to pay Bob

If the network agrees, that all conditions are true, Alice automatically gets the access code to the smart garage lock. The blockchain registers Alice as the new owner of the car. Bob has 20 000€ more in his account, and Alice 20 000€ less

Alice can now pick up her car by unlocking the smart lock with her private key

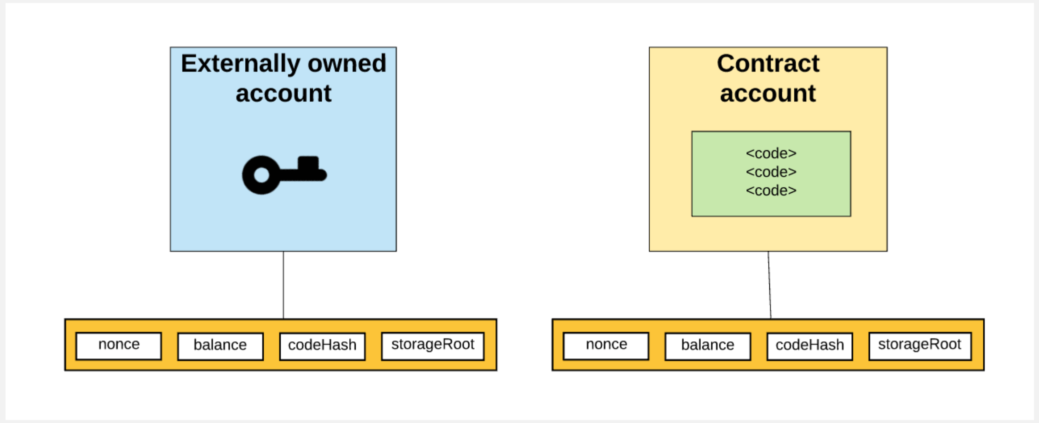
Ethereum

In Ethereum, the state is made up of objects called "accounts", with each account having a 20-byte address and state transitions being direct transfers of value and information between accounts.

An Ethereum account contains four fields:
The **nonce**, a counter used to make sure each transaction can only be processed once **(THIS IS NOT THE BLOCK NONCE!)**
The account's current **ether balance**
The account's **contract code**, if present
The account's **storage** (empty by default)

"Ether" is the main internal crypto-fuel of Ethereum, and is used to pay transaction fees.

In general, there are two types of accounts:
externally owned accounts, controlled by private keys: account has no code, and one can send messages from an externally owned account by creating and signing a transaction;
contract accounts, controlled by their contract code: every time the contract account receives a message its code activates, allowing it to read and write to internal storage and send other messages or create contracts in turn.

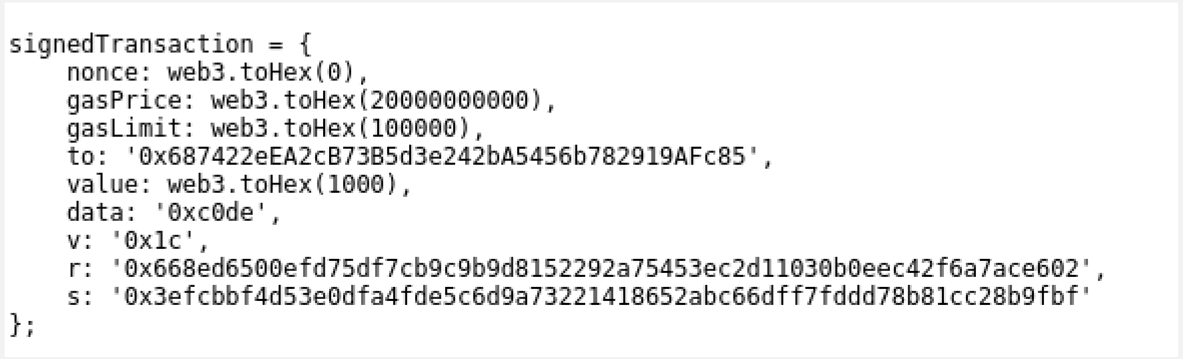


Ethereum contracts

"contracts" are like "autonomous agents" that live inside of the Ethereum execution environment
Always executing a specific piece of code when "poked" by a message or transaction
Have direct control over their own ether balance and their own key/value store to keep track of persistent variables.

The term "transaction" is used in Ethereum to refer to the signed data package that stores a message to be sent from an externally owned account.

Transactions contain:
The recipient of the message
A signature identifying the sender
The **amount of ether to transfer** from the sender to the recipient
An optional **data field**
A **STARTGAS value**, representing the maximum number of computational steps the transaction execution is allowed to take
A **GASPRICE value**, representing the fee the sender pays per computational step

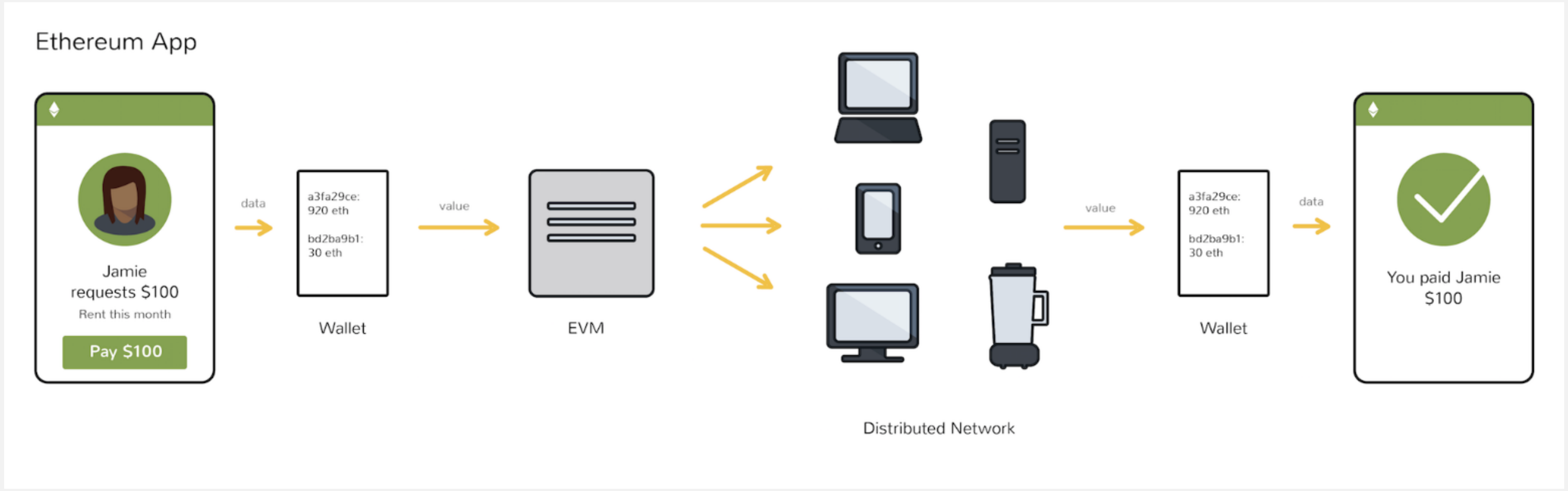


```
signedTransaction = {
    nonce: web3.toHex(0),
    gasPrice: web3.toHex(20000000000),
    gasLimit: web3.toHex(100000),
    to: '0x687422eEA2cB73B5d3e242bA5456b782919AFc85',
    value: web3.toHex(1000),
    data: '0xc0de',
    v: '0x1c',
    r: '0x668ed6500efd75df7cb9c9b9d8152292a75453ec2d11030b0eec42f6a7ace602',
    s: '0x3efcbbf4d53e0dfa4fde5c6d9a73221418652abc66dff7fddd78b81cc28b9fbf'
};
```

Ethereum

Contracts written in a smart contract-specific programming languages (Solidity and Serpent) are compiled into ‘bytecode’, which a feature called the ‘ethereum virtual machine’ (EVM) can read and execute.

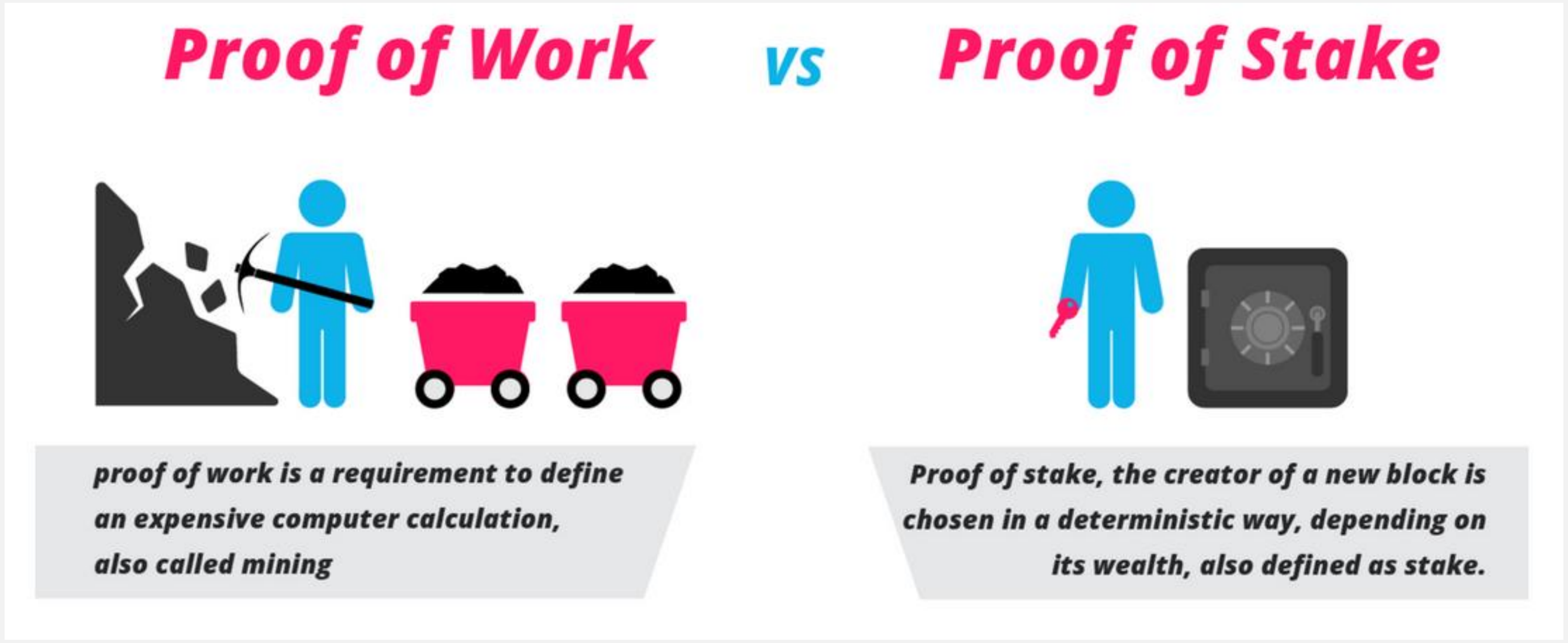
The goal here is for the network of miners and nodes to take responsibility for transferring the shift from state to state, rather than some authority such as PayPal or a bank.



Proof of stake

Unlike the proof-of-Work, where the algorithm rewards miners who solve mathematical problems with the goal of validating transactions and creating new blocks, with the proof of stake, **the creator of a new block is chosen in a deterministic way, depending on its wealth, also defined as stake.**

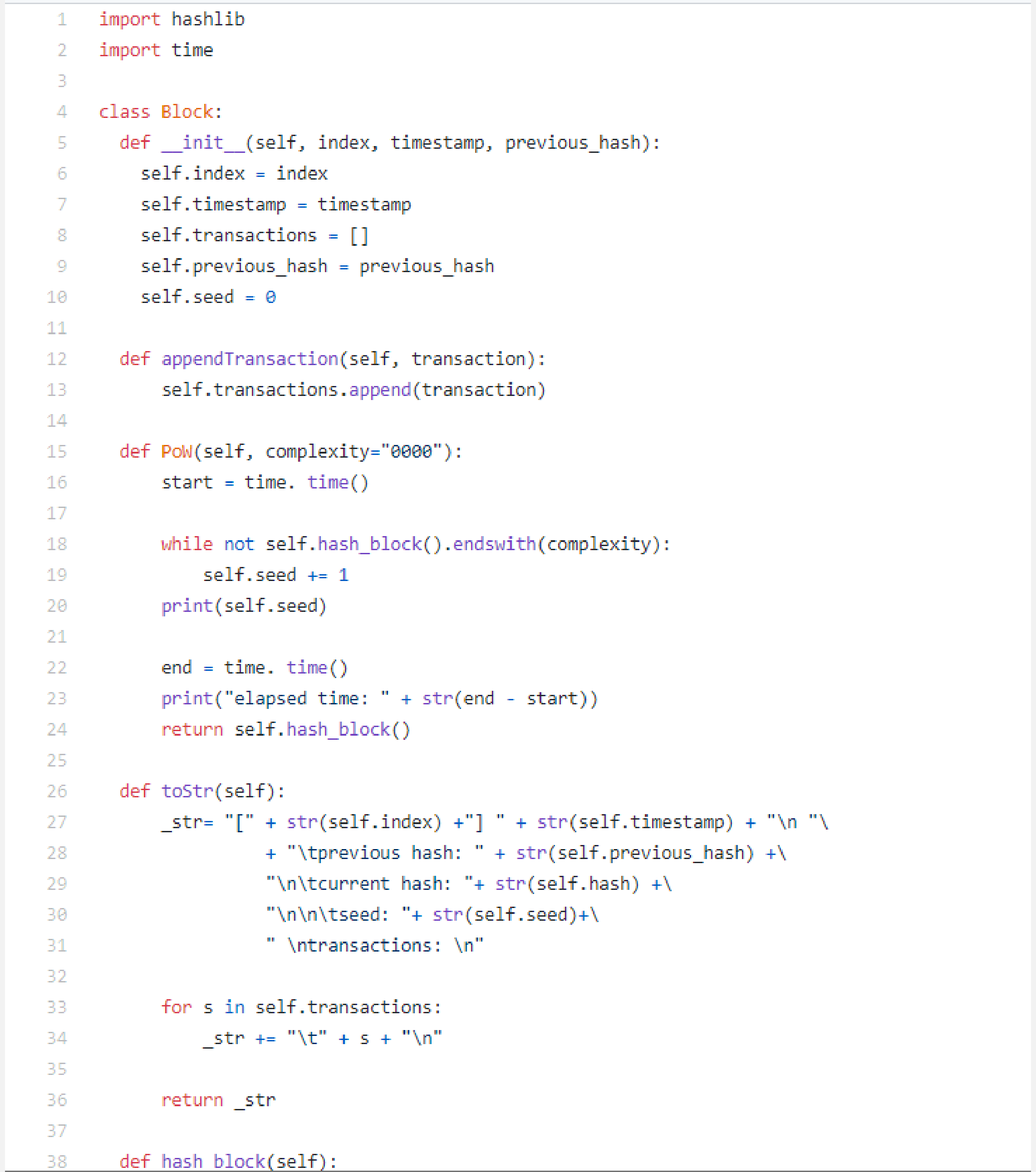
If Casper (the new proof of stake consensus protocol) will be implemented, there will exist a validator pool. Users can join this pool to be selected as the forger.


Proof of Work
vs
Proof of Stake
proof of work is a requirement to define an expensive computer calculation, also called mining
Proof of stake, the creator of a new block is chosen in a deterministic way, depending on its wealth, also defined as stake.

Let’s play!

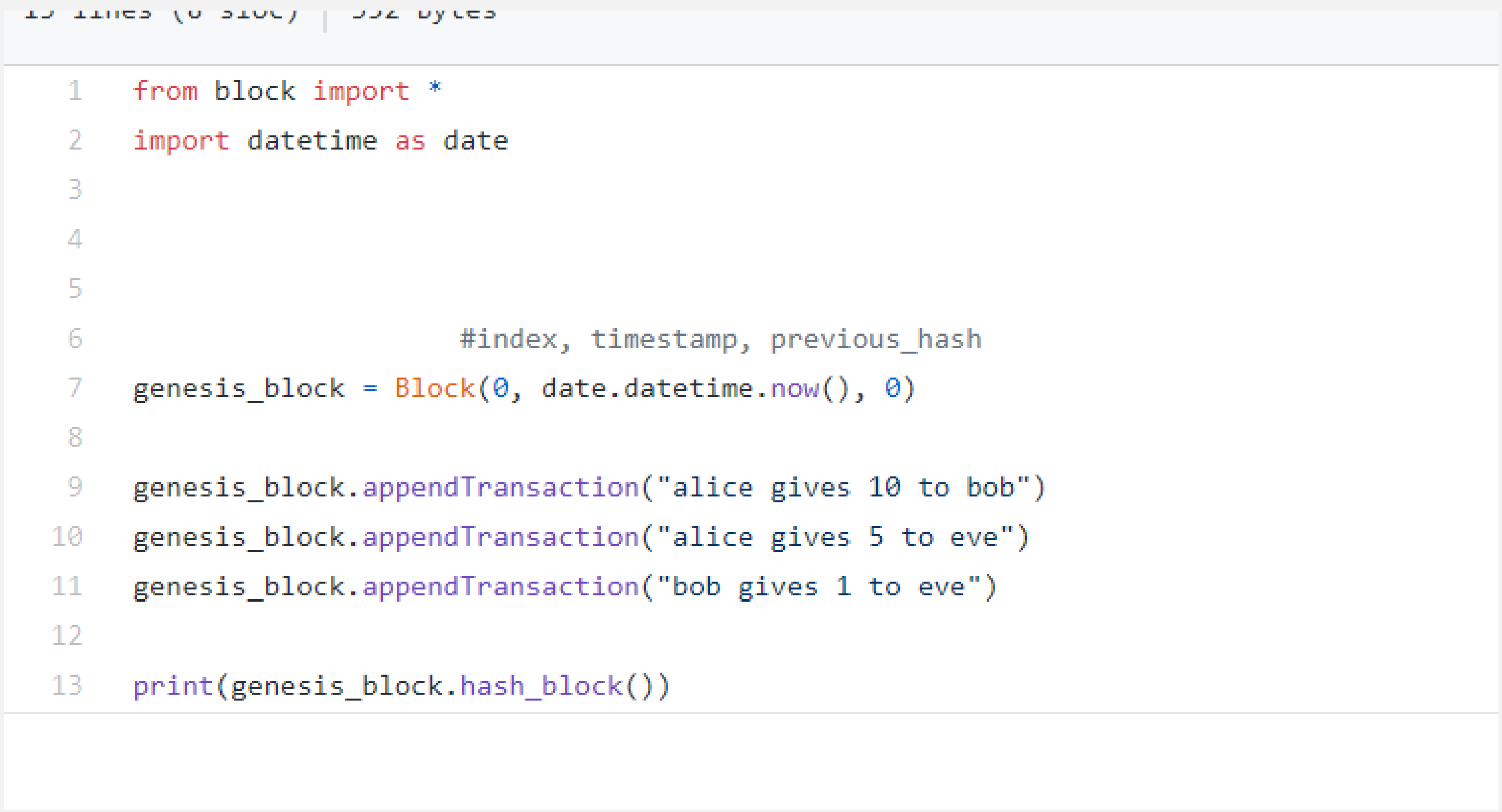
We will implement a simple (but working) blockchain in Python
tinyurl.com/y4sms6ls

Block



```
import hashlib
import time

class Block:
 def __init__(self, index, timestamp, previous_hash):
  self.index = index
  self.timestamp = timestamp
  self.transactions = []
  self.previous_hash = previous_hash
  self.seed = 0

 def appendTransaction(self, transaction):
   self.transactions.append(transaction)

 def PoW(self, complexity="0000"):
   start = time. time()

   while not self.hash_block().endswith(complexity):
     self.seed += 1
   print(self.seed)

   end = time. time()
   print("elapsed time: " + str(end - start))
   return self.hash_block()

 def toStr(self):
   _str= "[" + str(self.index) +"] " + str(self.timestamp) + "\n "\
        + "\tprevious hash: " + str(self.previous_hash) +\
        "\n\tcurrent hash: "+ str(self.hash) +\
        "\n\n\tseed: "+ str(self.seed)+\
        " \ntransactions: \n"

   for s in self.transactions:
     _str += "\t" + s + "\n"

   return _str

 def hash_block(self):
```

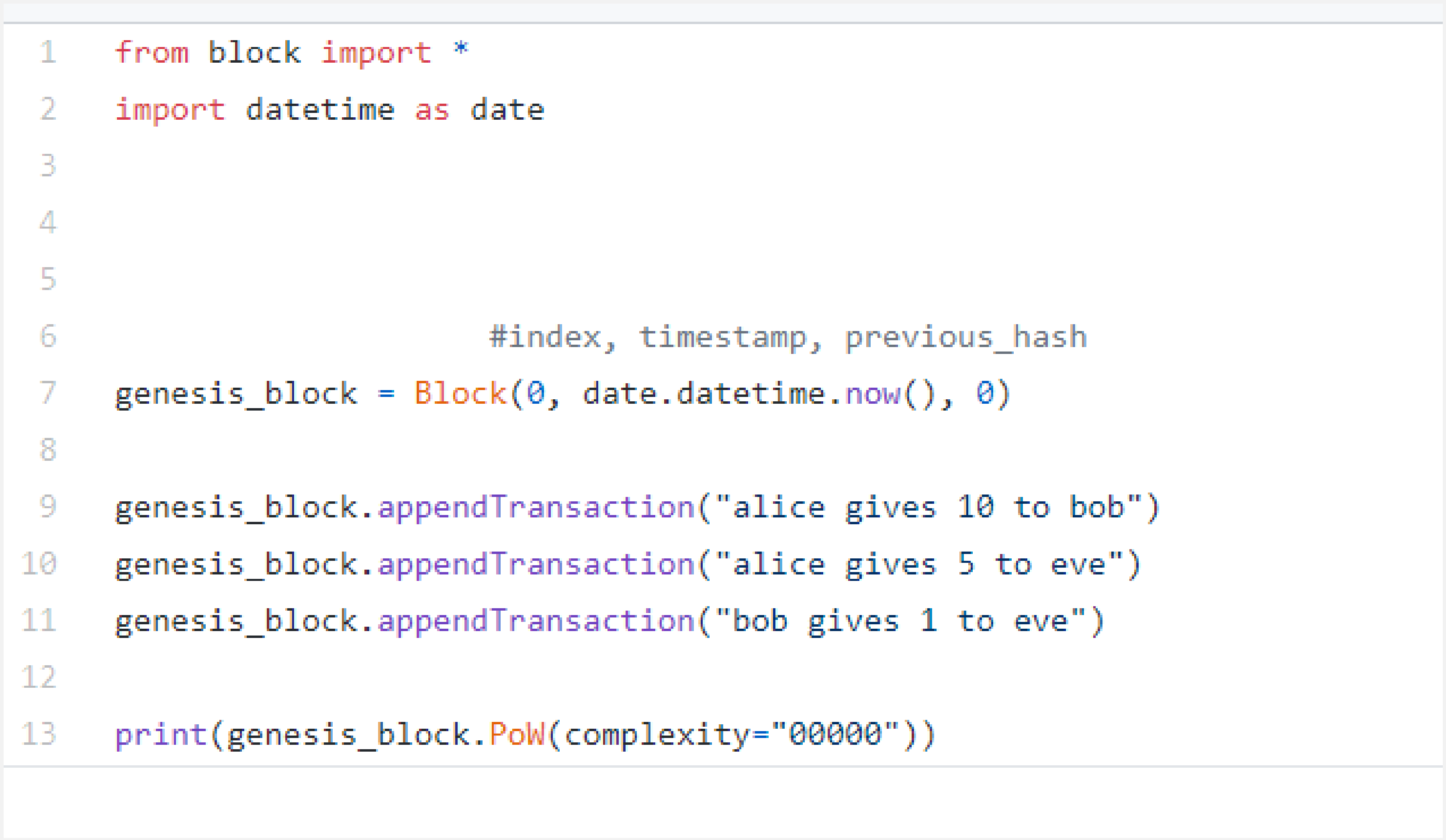
Run1



[illegible]

```python
from block import *
import datetime as date



                    #index, timestamp, previous_hash
genesis_block = Block(0, date.datetime.now(), 0)

genesis_block.appendTransaction("alice gives 10 to bob")
genesis_block.appendTransaction("alice gives 5 to eve")
genesis_block.appendTransaction("bob gives 1 to eve")

print(genesis_block.hash_block())
```

Run2



```python
from block import *
import datetime as date



                    #index, timestamp, previous_hash
genesis_block = Block(0, date.datetime.now(), 0)

genesis_block.appendTransaction("alice gives 10 to bob")
genesis_block.appendTransaction("alice gives 5 to eve")
genesis_block.appendTransaction("bob gives 1 to eve")

print(genesis_block.PoW(complexity="00000"))
```

Run3

```python
from block import *
import datetime as date
import random



                    #index, timestamp, previous_hash
genesis_block = Block(0, date.datetime.now(), 0)
latest_block = genesis_block
latest_hash = 0
latest_index = 0


while True:


    latest_hash = latest_block.PoW(complexity="00000")

    latest_block.appendTransaction("alice gives "+str(random.randint(1,100))+" to bob")
    latest_block.appendTransaction("alice gives "+str(random.randint(1,100))+" to eve")
    latest_block.appendTransaction("bob gives "+str(random.randint(1,100))+" to eve")

    print(latest_block.toStr())

    latest_block = Block(latest_index, date.datetime.now(), latest_hash)
    latest_index += 1
```

Questions?

Michele.martinelli@wsense.it