Blockchain

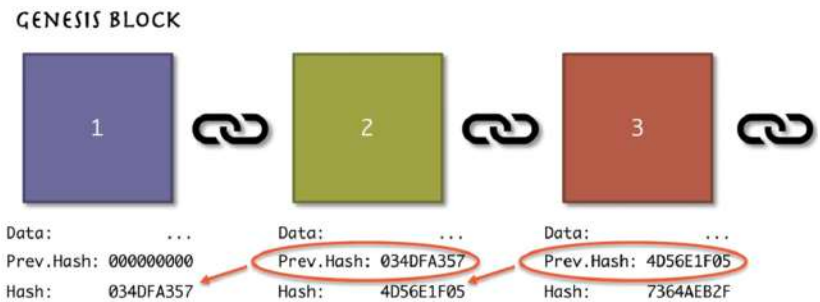
What is a blockchain?
A blockchain is a continuously growing list of records, called blocks, which are linked and secured using cryptography.

Consider a block.
1. It has a 'Data' inside it:
   - Data: "Hello World!"
2. It has a value which is called Previous Hash (the Hash value for the previous block)
   - Prev. Hash: 034DFA357
3. The block Hash, which is like a fingerprint of this block
   - Hash: 4D56E1F05

So the block hash, gets the data and value and calculates a number which represents those data. So the block Hash is a short version of Data and Prev. value which specifically 64 characters long.

Imagine we have a chain of blocks.
1. The first block is called Genesis Block. This block will never change.
It has a Data, It doesn't have a previous hash, it has its own block hash.
2. It has Data, the hash for previous block hash (as a reference), and its own block hash.
3. It has Data, the hash for previous block hash, and its own block hash…
4. …

That is why the blocks are cryptographically linked together.



Cryptography - SHA256 Hash:

There are some fingerprints for documents that are completely unique, which is called SHA256. We use SHA256 algorithm for storing passwords or check digital documents,
SHA = Secured Hash Algorithm
256 = the number of bits it takes in memory.
The hash always 64 characters long, containing letters and numbers.
The hash is Hexadecimal hash: Just numbers 0 to 9 and letters: A, B, C, D, E, F

Each character in the resulting hash takes up 4 bits.
64 character * 4 bits = 256 bits

Blockchain Demo: Hashes and Blocks:
https://tools.superdatascience.com/blockchain/hash/

Here we can write anything and instantly see its hash.
Ex)
Data: Hello this is Arman Feili
Hash:
c88caf37e4ccc229dcb919546648dca63c2ee2d77a2048336e7cd41cb0162f34

This has represents the data everywhere. So if we copy and paste the text again, it gives the same hash code as result.
If we change even 1 character of the data, the Hash code will be changed completely.

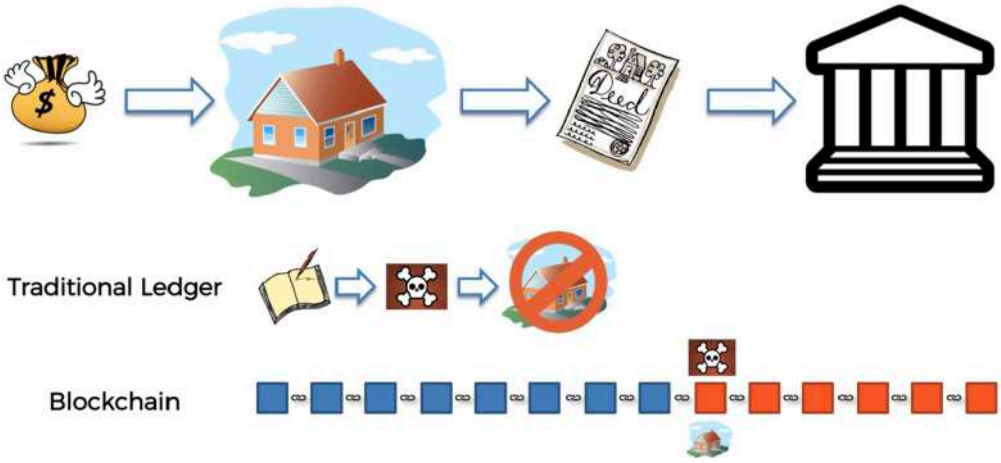
The 5 requirements for Hash Algorithms:

1. It has to be **One-Way**. You cannot go from the Hash to the document !!! So you cannot restore the document based on the hash.
2. It has to be **Deterministic**. Means that if we run the Same document and apply the SHAHash algorithm again, we get the exact same result.
3. It has to be **Fast Computation.**
4. **The Avalanche Effect**. If we apply a tiny little change to the document, then the hash will be absolutely different.
5. It **Must Withstand Collisions**. For example, 2 persons can have same fingerprints in 60 million people. So, it is too rare and unlikely to have of find 2 SHA256 hash code the same as each other. But the algorithm should withstand with it, so the pirates cannot generate a file like yours to have same hash for that. So the Collisions should **NOT** be possible.

Immutable Ledger:

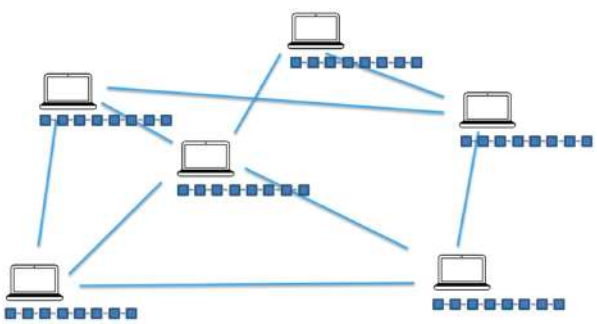
Traditional Ledger:
Let's say you want to go and buy a house. So you pay your money for your dream home. So you will have a deed as a proof that claims it is your home. For getting the deed, you should go and sign it in a city or council authority and register your ownership.
If we lose the deed or burn it, or if someone hack the SQL database of the authority, you will lose your house.

So imagine we have this data in blockchain. Anytime that a person buys that house, one block will be added to that chain.
It takes so long for a hacker to find and change the hash for the block related to your house! If you sell the house to someone else, the hacker still needs to hack and change the hash code for the next blocks that contain your hash of your block.
So the following blocks will be not valid anymore.
That's why we say the blockchain is an immutable ledger !
So it's practically impossible to change the hash code.

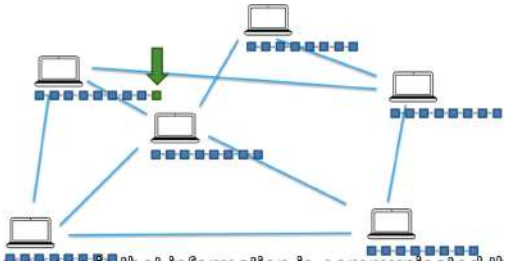


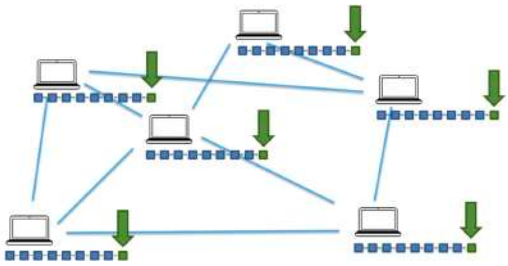
Distributed P2P Networks:

In P2P systems, we have lots of nodes (computers) that are interconnected.
In case of our blockchain, once we use the P2P network, all the computers in the P2P network have a copy of our blockchain and keep the chain updating.

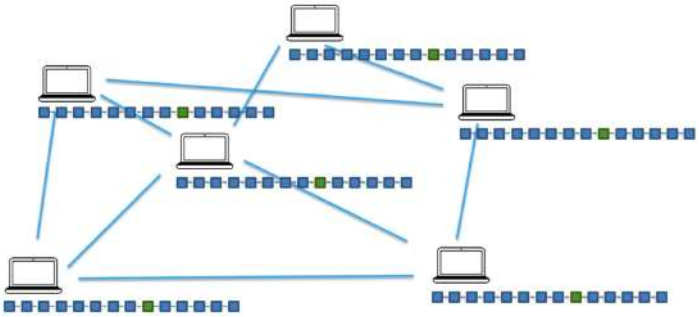


Once we added a new block to the chain in our computer, that information gets communicated throughout the network, and that block is added further throughout the network and all then all the computers will have this block. It might take some time of course.



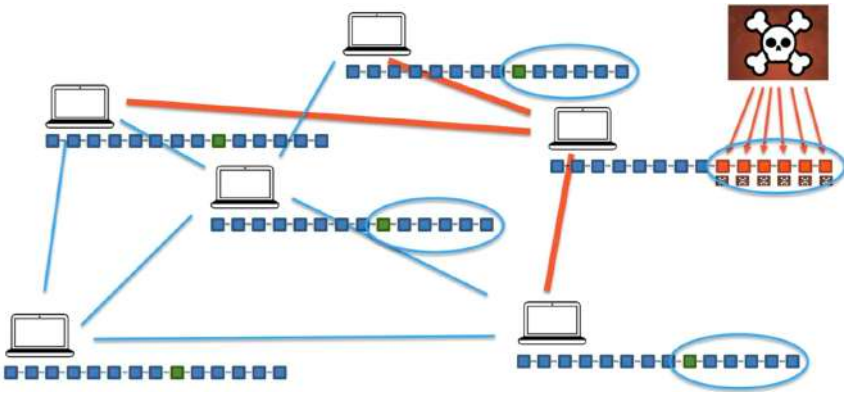


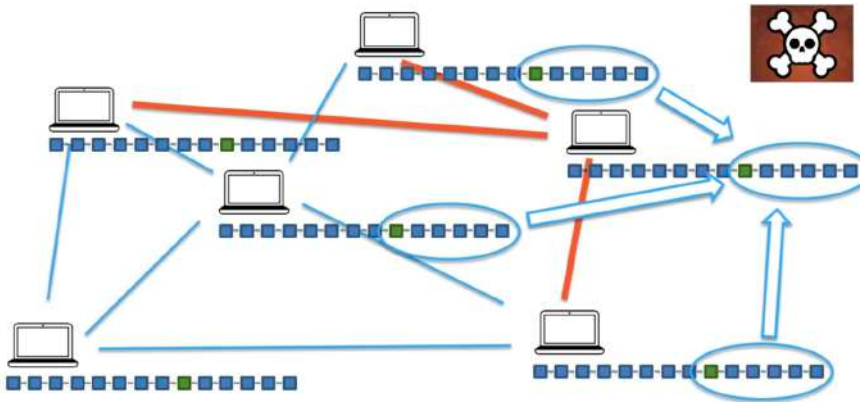
And as time passes, more transactions will be added to the chain.



Now if someone wants to hack the block that we added, they need to update all the hash codes for the following blocks, for all of the chains !!
That's why we say it's impossible.

If someone changes our block and all the following blocks, other computers will notice the difference between their blockchains with the hacked blockchain. So they will instantly copy and paste the actual real blocks in the hacked one.





Mining:

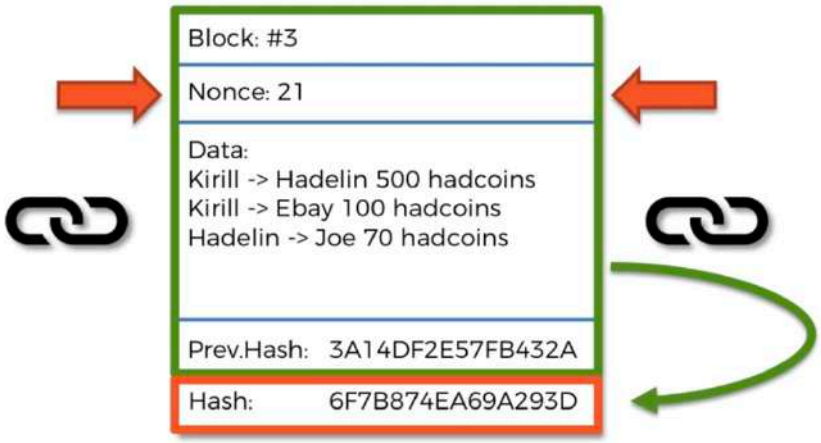
Imagine a block in a blockchain.

Note: A block doesn't just store one single transaction. It can store multiple transactions

There is another field in the block, named **Nonce** which stands for **Number used on the once**.
This is the filed that the mining is all about.

Everybody are just changing the Nonce field, all the time. Since the Hash of the block, gets 1. The number of the block 2. Nonce 3. Data and 4. Prev hash , we can change the Hash of the block, just by changing the Nonce ! So the Nonce gives us extra control.

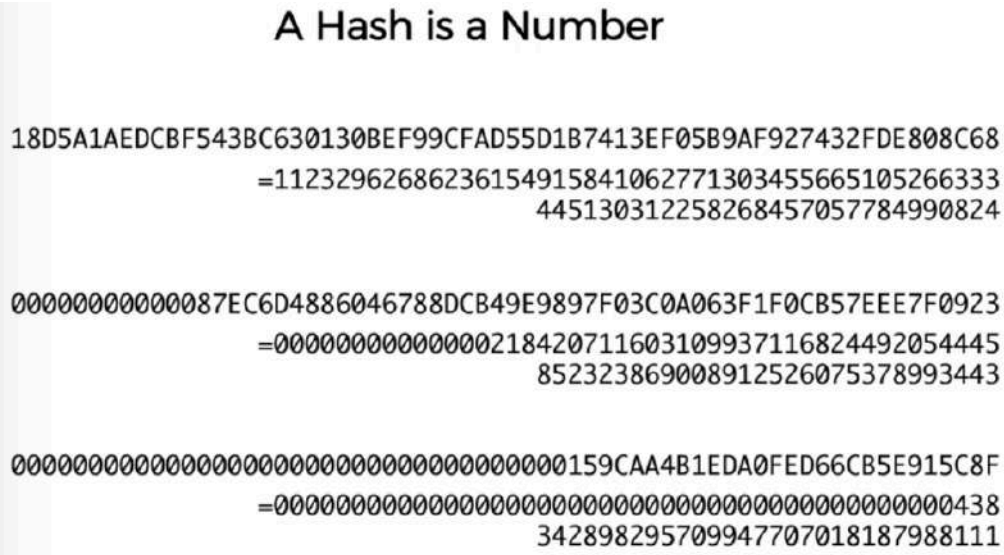
So anytime we increase the number of the Nonce, the Hash code will be changed completely.



The Hexadecimal number consists of 0 to 9 and A=10, B=11, C=12, D=13, E=14, F=15

We can convert it to a decimal number.

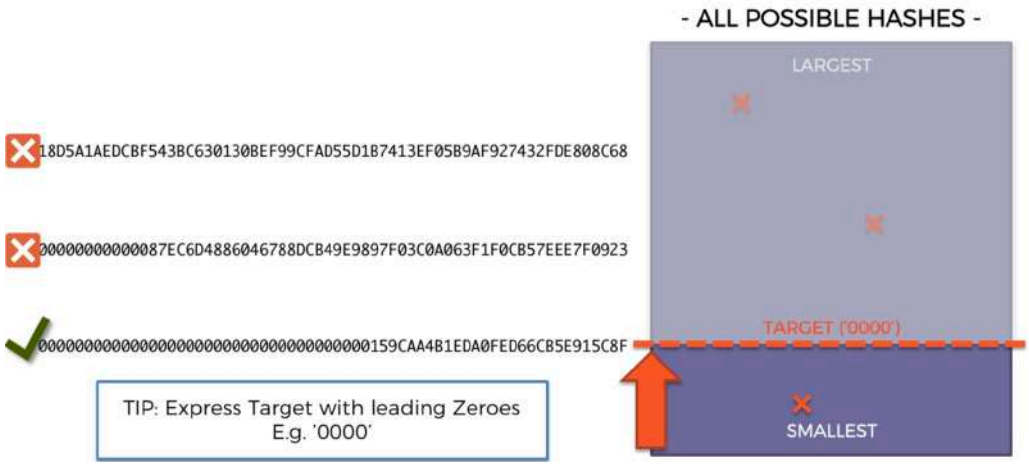
Here are some Hexadecimal number and their converted versions to decimal numbers:



The smallest hash number is: 00000000000000000000000000 -> 64 char
The largest hash number is: FFFFFFFFFFFFFFFFFFFFFFFFFF -> 64 char

There is a target which is set for miners to accomplish a certain hash. And any hash, higher than target doesn't count.
In this case, if the miner finds the first or second hash, it would not be allowed to create a block with these hash number!

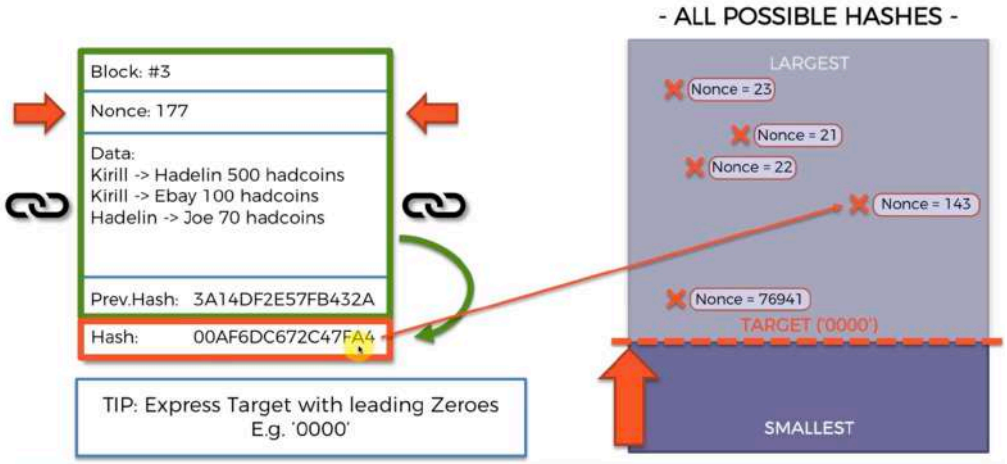
The best way to find the target is reading **zeros**.
The more we have zeros, the lower the number of the hash is and it can be calculated to see if it's under the target or not.



But what miners do?

Minders just change the Nonce value in order to get reach a Hash number which is below the target. Once they find that desired Hash number, they can create a block and add it to the blockchain and get the reward.

Besides, as you see, the position of the hash in the range, is **NOT** depended to the value of **Nonce**. Because of **The Avalanche Effect** of SHA256 algorithm, the position of the hash is random and is not predictable. So miners cannot cheat and lower the Nonce to get the lower value of hash.

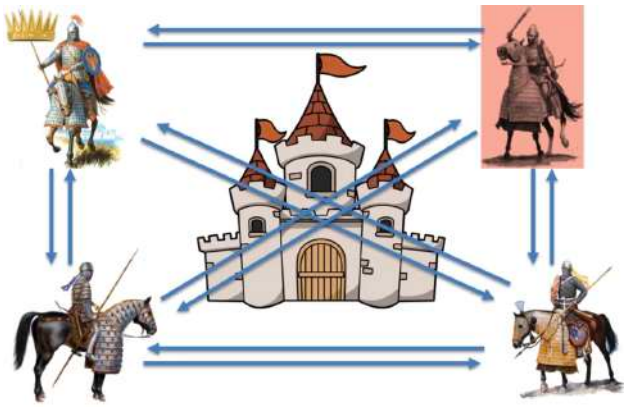


Byzantine Fault Tolerance:

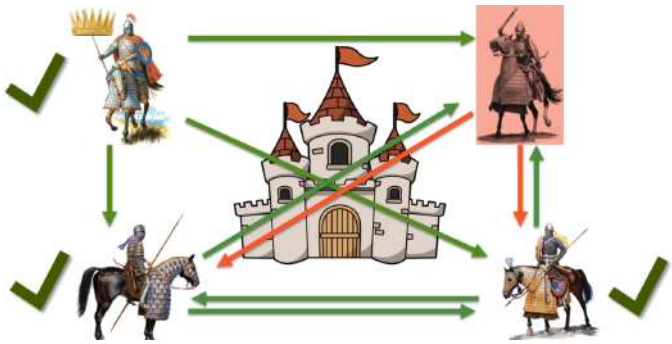
There was a castle that 4 generals and their army wanted to capture. One of them is the main General (like a king) and one of them is a traitor. They need to come to an agreement on how to attack the castle or the enemy would destroy them.

Here is the question: How do they come up with a consensus or algorithm that help them come to a decision despite there being a traitor.

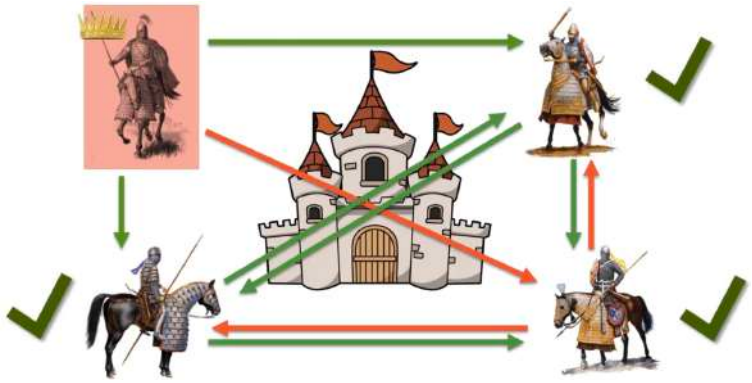
They can communicate with each other, but the since one of them is traitor, the best they can do is to (the algorithm they use) is they just look at the majority of messages that they get make decision based on them.



So if the main general commands on Attack, as you see below, the majority of commands (Green arrows) are attack. And the traitor can be easily found.



Now let's look at the case if the main general is traitor. Possibly, he will send commands of Attack to 2 of them and 1 command of retreat to one of them.



In this case also, based on the majority of commands, they will all attack.

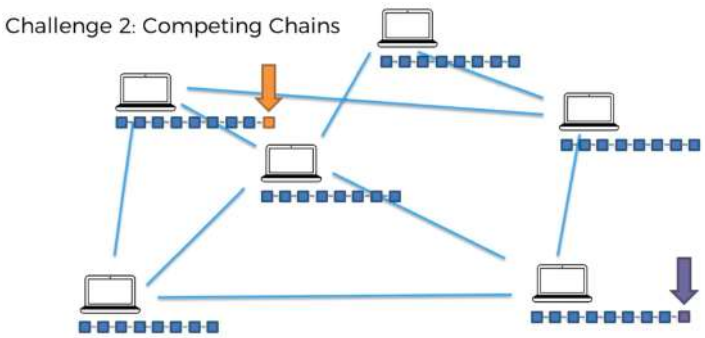
Note: in order to victory the war, no more than 33% (one third) of generals should be traitor. If the number of traitors was 2, they wouldn't be succeed. This is the **level of Tolerance**.

In blockchain, we need an algorithm like **Byzantine**, to work based on that, cause maybe one wants to hack a blockchain, so the system should protect itself.
This algorithm works in many networks. Like airplane systems.

Consensus Protocol:

Challange 1:
The main challenge for miners is: which command to listen to ?!

Challange 2:
Competing Chain: when 2 of the miners (2 nodes) could find and successfully mine a block at the same time !!!
We cannot split the fee and give it to the both of them. One should be chosen!



Consensus Protocol responsible for:
1. Proof-of-Work (PoW)
2. Proof-of-Stake (PoS)
3. Other

As a matter of fact, a miner should go thorough thousands or millions of iterations to reach a Nonce with lower hash to be allowed to create a block in the chain. It needs a huge amount of time and electricity.
The final hash that they will create is their Proof-of-Work. It is the proof that they did all this work to solve the cryptographic challenge.

Once the miner adds a block, it receives a reward from the system + the transaction fee.

How the system can find if the miner is adding a malicious block?

Just before adding the block to the chain by the miner, Every Single Node in the system, check some certain rules. If one of the checks fails, other nodes will reject the block !!!

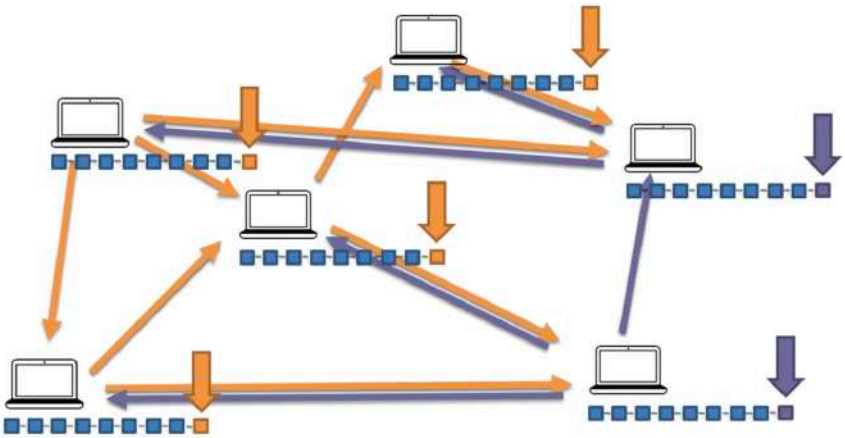
For example, the transaction shouldn't be empty. The previous hash code should be correct and so on…

1. Check syntactic correctness
2. Reject if duplicate of block we have in any of the three categories
3. Transaction list must be non-empty
4. Block hash must satisfy claimed *nBits* proof of work
5. Block timestamp must not be more than two hours in the future
6. First transaction must be coinbase (i.e. only 1 input, with hash=0, n=-1), the rest must not be
7. For each transaction, apply "tx" checks 2-4
8. For the coinbase (first) transaction, scriptSig length must be 2-100
9. Reject if sum of transaction sig opcounts > MAX_BLOCK_SIGOPS
10. Verify Merkle hash
11. Check if prev block (matching *prev* hash) is in main branch or side branches. If not, add this t
    orphan block in *prev* chain; done with block
12. Check that *nBits* value matches the difficulty rules
13. Reject if timestamp is the median time of the last 11 blocks or before
14. For certain old blocks (i.e. on initial block download) check that hash matches known values
15. Add block into the tree. There are three cases: 1. block further extends the main branch; 2. bl
    make it become the new main branch; 3. block extends a side branch and makes it the new n
16. For case 1, adding to main branch:
    1. For all but the coinbase transaction, apply the following:
        1. For each input, look in the main branch to find the referenced output transactio
        2. For each input, if we are using the *n*th output of the earlier transaction, but it ha
        3. For each input, if the referenced output transaction is coinbase (i.e. only 1 inpu
           (100) confirmations; else reject.
        4. Verify crypto signatures for each input; reject if any are bad
        5. For each input, if the referenced output has already been spent by a transactio
        6. Using the referenced output transactions to get input values, check that each ir
        7. Reject if the sum of input values < sum of output values
    2. Reject if coinbase value > sum of block creation fee and transaction fees

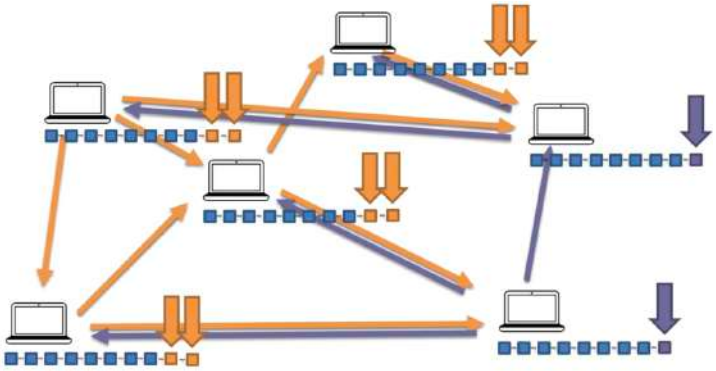
Overall, the main hard work is to find the Nonce, but for checking the hash to see if it's under the target, is an easy task to do. All it takes is to take everything in the block and put them in SHA256 algorithm and fetch the hash.

What happens if 2 nodes get to the same Nonce and add a block to their chain?

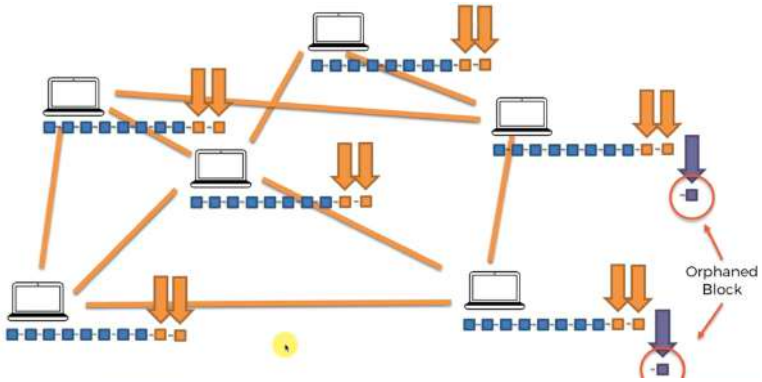
it's different from Byzantine algorithm.
As you see, here are two nodes (Orange and Purple), added their block to the chain and send the message to other nodes that they solved a transaction.
The Orange Node might be faster computer, and send the message to more nodes. But the system doesn't decide by their speed !



What the network does is to **WAIT**. It waits for the next block! Any of those 2 blocks which could make their chain longer faster can win the reward !
So which ever the chain gets longer, wins and replace the other chains !
So the part of the network which has the higher power of hashing will generate the next block faster and will win the competition.
So overall, the Orange group has a higher chance of creating the next block, cause they have higher power of hashing.



The conclusion: In Consensus Protocol, those nodes that have **more than 50%** of the hashing power will win the competition.

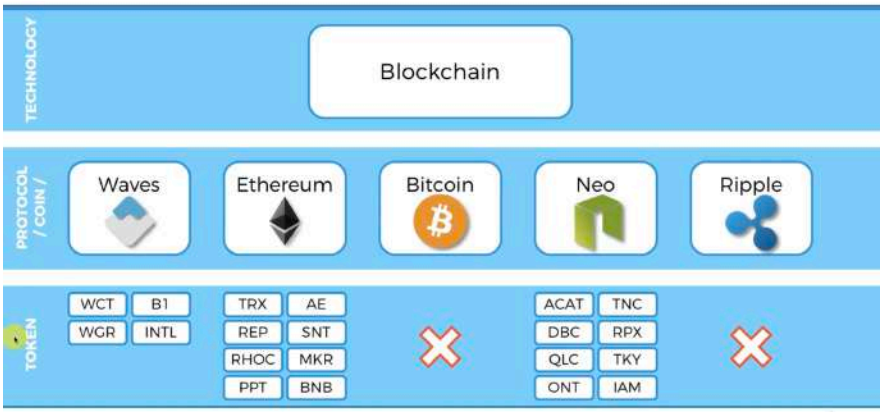


```python
# Module 1 - Create a Blockchain

# To be installed:
# Flask==0.12.2: pip install Flask==0.12.2
# Postman HTTP Client: https://www.getpostman.com/

# Importing the libraries
import datetime  # for the exact time that the block is created.
import hashlib  # use it to hash the blocks
import json  # to encode the block before hashing them

# import the Flask class for the web-application + jsonify to display the response of the HTTP request.
# also we use jsonify to return the key information of this new block that was just mined, in json format.
from flask import Flask, jsonify


# Part 1 - Building a Blockchain


class Blockchain:
    def __init__(self):
        self.chain = []
        # Generate the first block as Genesis block.
        self.create_block(proof=1, previous_hash="0")

    def create_block(self, proof, previous_hash):
        block = {
            "index": len(self.chain) + 1,   # index of the block = length of the current chain + 1
            "timestamp": str(datetime.datetime.now()),  # the time that this block is created
            "proof": proof,
            "previous_hash": previous_hash,
        }
        # append this block to our chain
        self.chain.append(block)
        # return the block to display
        return block

    def get_previous_block(self):
        return self.chain[-1]   # the last index of the chain

    # The proof work is the number (piece of data) that miners have to find in order to mine a new block.
    # The challenge is Hard to find the proof-of-work but easy to verify.
    # It's hard to find, cause if it was easy, miners could mine lots of proof-of-works and grab all rewards.
    # It's easy, cause other miners need to verify that the first miner solve the problem.
    def proof_of_work(self, previous_proof):
        # initialize the new_proof value as 1
        # new_proof is the role for Nonce - So miners will increment this variable
        # at each iteration of a While loop until we get the right proof !
        # Basically we are solving the problem with a 'Trial and Error' approach.
        new_proof = 1
        check_proof = False
        while check_proof is False:
            # We need to come up with an algorithm to generate a unique and challenging hash.
            # that's why we wrote: [ new_proof ** 2 - previous_proof ** 2 ] to make finding the hash code, challenging.
            # Thee encode() and hexdigest() will convert the string into SHA256 format with 64 chars.
            hash_operation = hashlib.sha256(
                str(new_proof ** 2 - previous_proof ** 2).encode()
            ).hexdigest()
            # we check for the Four Leading Zeros. - It's a clasic way to define a problem that miners need to solve to find the PoW
            # The more Leading Zeros you impose, the harder it would be for the miners to solve the problem.
            # Because they need to iterate more to find a hash code with higher number of zeros.
            if hash_operation[:4] == "0000": # take a range between 0 to 4
                check_proof = True
            else:
                new_proof += 1
        # Once we find the right hash code, we need to return the new_proof as the Proof-of-Work
        return new_proof

    # Define a hash function that creates a cryptographic hash for a block.
    def hash(self, block):
        # get the block dictionary and convert its data to string, using json.dumps() function.
        # [ sort_keys=True ] means that the blocks are sorted by the keys.
        # .encode() function, adds a 'b' before the data which is nesseccary for SHA256 algorithm.
        encoded_block = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(encoded_block).hexdigest()

    # here is 2 methods for checking our blockchain. We are gonna check 2 essential things:
    # 1. Check if each block in the blockchain has a PoW ?
    # 2. Check if the Prev. Hash of each block is equal to the hash of previous block?
    def is_chain_valid(self, chain):
        # initialize prev block and the looping variable, block_index
        previous_block = chain[0]
        block_index = 1
        while block_index < len(chain):
            # Check if the Prev. Hash of each block is equal to the hash of previous block?
            block = chain[block_index]
            if block["previous_hash"] != self.hash(previous_block):
                return False
            # Check if each block in the blockchain has a valid PoW ?
            previous_proof = previous_block["proof"]
            proof = block["proof"]
            hash_operation = hashlib.sha256(
                str(proof ** 2 - previous_proof ** 2).encode()
            ).hexdigest()
            # Check if the hash has the first 4 leading zeros
            if hash_operation[:4] != "0000":
                return False
            # update the previous block and the looping variable
            previous_block = block
            block_index += 1
        # if everythong was right, return True
        return True
```

```
# Part 2 - Mining our Blockchain

# Creating a Web App
app = Flask(__name__)
app.config['JSONIFY_PRETTYPRINT_REGULAR'] = False


# Creating a Blockchain
blockchain = Blockchain()

# Mining a new block
@app.route("/mine_block", methods=["GET"])
def mine_block():
    previous_block = blockchain.get_previous_block() # get the last block of current chain
    previous_proof = previous_block["proof"]
    # the proof of our future new block that will be added to the blockchain
    # the following code, will get the PoW of last block (The Nonce), then increment it and return a new PoW for next new Block.
    proof = blockchain.proof_of_work(previous_proof)
    # we need to restore the hash for the last block, to add it into the next new block.
    previous_hash = blockchain.hash(previous_block)
    # create and add the block just after mining the correct hash code under the target.
    block = blockchain.create_block(proof, previous_hash)
    # in order to display the new block, we need to fetch the data of block
    response = {
        "message": "Congratulations, you just mined a block!",
        "index": block["index"],
        "timestamp": block["timestamp"],
        "proof": block["proof"],
        "previous_hash": block["previous_hash"],
    }
    return jsonify(response), 200


# Getting the full Blockchain
@app.route("/get_chain", methods=["GET"])
def get_chain():
    response = {"chain": blockchain.chain, "length": len(blockchain.chain)}
    return jsonify(response), 200


# Checking if the Blockchain is valid
@app.route("/is_valid", methods=["GET"])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)
    if is_valid:
        response = {"message": "All good. The Blockchain is valid."}
    else:
        response = {
            "message": "Houston, we have a problem. The Blockchain is not valid."
        }
    return jsonify(response), 200


# Running the app
# Run on server "0.0.0.0" to make the app, publicly available.
app.run(host="0.0.0.0", port=5000)

# if we run the file, we can send the HTTP GET request by Postman:
# GET -> http://127.0.0.1:5000/get_chain
# GET -> http://127.0.0.1:5000/mine_block
# GET -> http://127.0.0.1:5000/is_valid
```

What is bitcoin?

In the subject of cryptography, there are 3 main layers:
1. Technology -> Blockchain
2. Protocol /Coin/ -> Bitcoin, Ethereum, Neo, Ripple
3. Token -> TRX, BNB, PPT, SNT



Protocol:
A Protocol is a set of rules or procedures for transmitting data between electronic devices, such as computers.

In order for computers to exchange information, there must be a preexisting agreement as to how the information will be structured and how each side will send and receive it. Without a protocol, a transmitting computer, for example, could be sending its data in 8-bit packets while the receiving computer might expect the data in 16-bit packets.

If one computer uses the Internet Protocol (IP) and a second computer does as well, they will be able to communicate.
Among the most important sets of Internet protocols are TCP/IP (Transmission Control Protocol/Internet Protocol), HTTPS (Secure HyperText Transmission Protocol), SMTP (Simple Mail Transfer Protocol), and DNS (Domain Name System).

Bitcoin, Ethereum, Neo, Ripple are not just coins or currencies. They are actual **protocols**.

For example, the protocol dictates how they should come to consensus of things that are needed.
The Protocol also dictates how public keys and signatures should be used for authentication.
The Protocol also dictates how how they agree on updates to the protocol itself and lots of more things.

Each mentioned Protocols have their own **Coins**.
- A Coin is an innate asset of the protocol, which facilitates the interaction of nodes.
- Coins are used to reward people for mining the blockchain and adding blocks.
- Coins are used for people to buy things from each other.

ICO = Initial Coin Offering

Tokens rely on smart contracts which are built on top of the different protocols.
The Ethereum is the most popular protocol for creating smart contracts. It has many tokens associated with Protocols.

Bitcoin has no tokens

https://coinmarketcap.com/

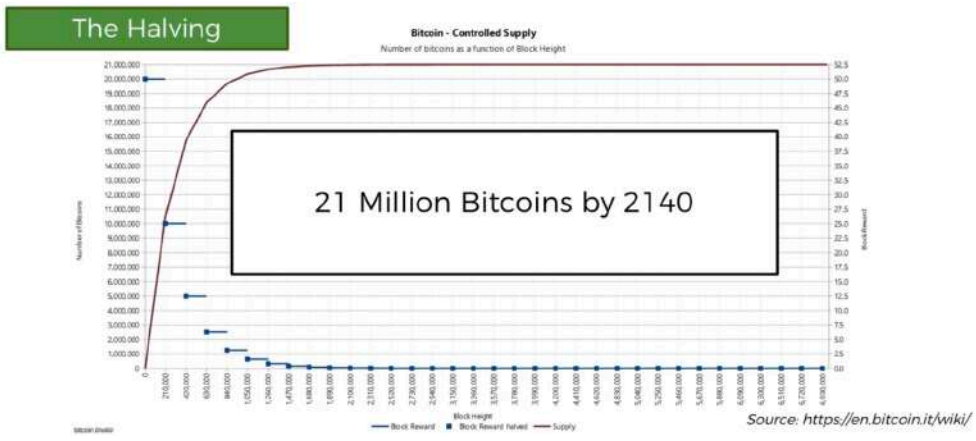
Bitcoin's Monetary Policy:

1) **The Halving**
2) **Block Frequency**

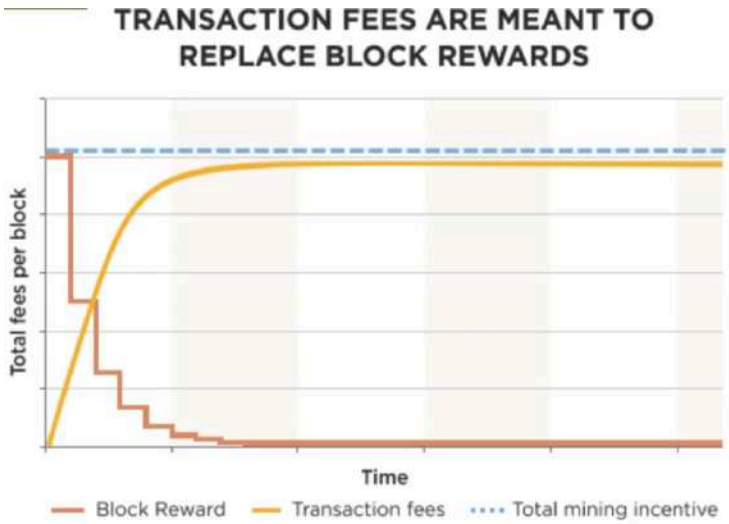
The Halving:

The number of bitcoins, released into system (to be mined) is halved every single 4 years.
As you see, the number of bitcoins per block is reduced by 2 every 4 years.

Date reached	Block	Reward Era	BTC/block
2009-01-03	0	1	50.00
2010-04-22	52500	1	50.00
2011-01-28	105000	1	50.00
2011-12-14	157500	1	50.00
2012-11-28	210000	2	25.00
2013-10-09	262500	2	25.00
2014-08-11	315000	2	25.00
2015-07-29	367500	2	25.00
2016-07-09	420000	3	12.50
2017-06-23	472500	3	12.50



So the amount of Bitcoins for mining will be close to Zero , and at last, there won't be any bitcoin to mine and the fees that the miners will get will be just from transaction fees. However the fees will be increased !!!



Block Frequency:
Block time is the measure of the time it takes the miners or validators within a network to verify transactions within one block and produce a new block in that blockchain.

Cryptocurrency	Average block time
bitcoin	10 min
ethereum	15 sec
ripple	3.5 sec
litecoin	2.5 min

It's the number of times a block executes. How often do these blocks come in that bring the reward? Which is right now, 2.5 bitcoins per block.
It varies in different protocols.

The block time on the bitcoin blockchain is 10 minutes.
This means that every 10 minutes a new block of transactions is added to the blockchain and transactions within the block are considered to be 'processed'. However this block time isn't set in stone and can oscillate between a few seconds and a few days!

Mining Difficulty:

What is the current target and how that feel?
By requesting 1 leading Zero, we're effectively reducing the pool size by 10 in a Hexadecimal number!

XXXXX : 0 - 99.999 (100,000 options)
0XXXX : 0 - 9.999 (10,000 options)

Imagine a target with 18 leading zeros !!!
Let's calculate its probability.

Current target = 000000000000000005d97dc0000000000000000000000000000000000000000

18 zeros

Let's do some estimations:

Probability:
Total possible 64-digit hexadecimal numbers: $16 \times 16 \times \ldots \times 16 = 16^{64} \approx 1.1579 \times 10^{77} \approx 10^{77}$
Total valid hashes (with 18 leading zeros): $16 \times 16 \times \ldots \times 16 = 16^{64-18} \approx 2.4519 \times 10^{55} \approx 2 \times 10^{55}$

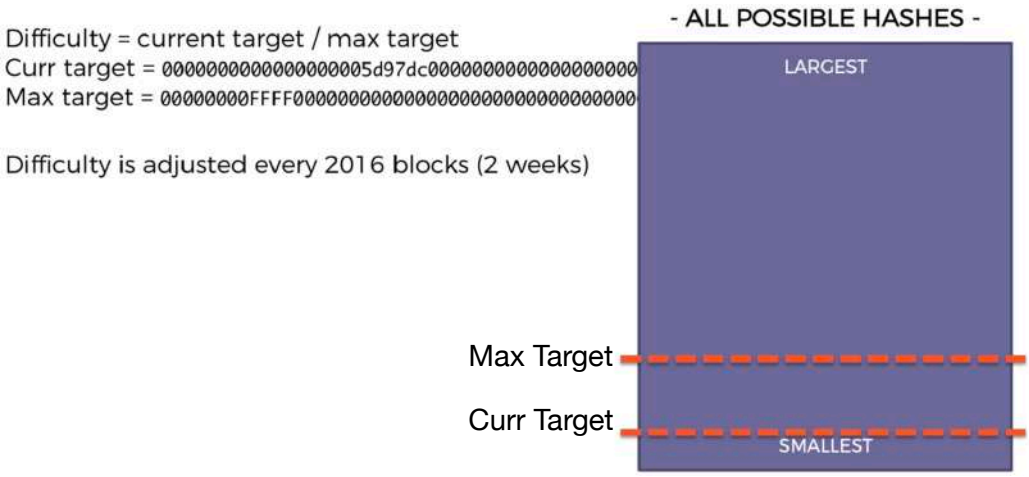
Probability that a Randomly picked hash is valid: $2 \times 10^{55} / 10^{77} = 2 \times 10^{-22} = 0.00000000000000000002\%$

So the probability is quite low, that's why miners machines are continuously working.

How is Mining Difficulty calculated?

Difficulty = Current target / max target
Difficulty is adjusted every 2016 blocks (2 weeks)
Every block should be released every 10 minutes. So in 2 weeks, 2016 blocks will be released.

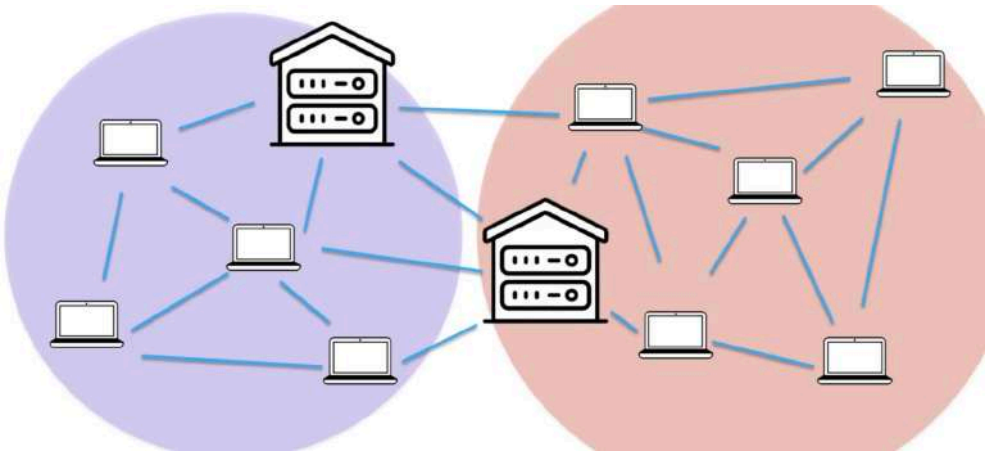
The Max target is not Maximum number of hash.



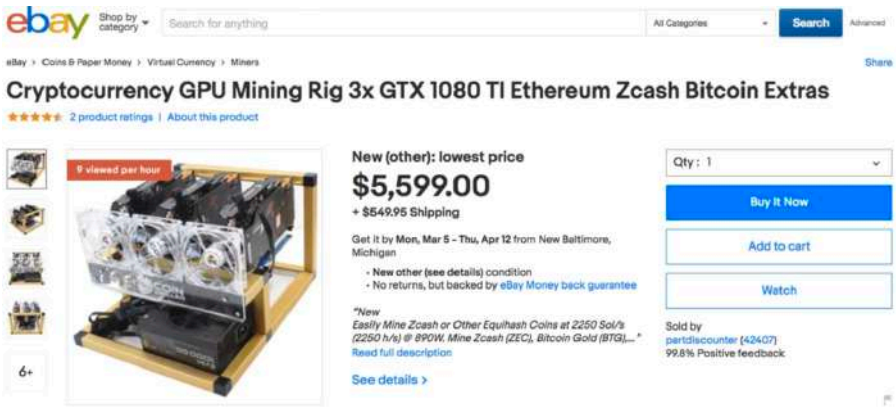
Mining Pool

If you want to compete in the distributed system of mining, you cannot win the prize when there are huge companies that are solving the hash algorithms.
So you can join to a mining pool ! The nodes in a mining pool combine their hashing power with each other.
The other thing that mining pool does is to allocate ranges of Nonce to different mining nodes. So one is responsible to look after 0 to 1Billion, the other one is responsible for 1B to 2B and so on.
Once one of the nodes, found the Nonce, that mining pool, wins.

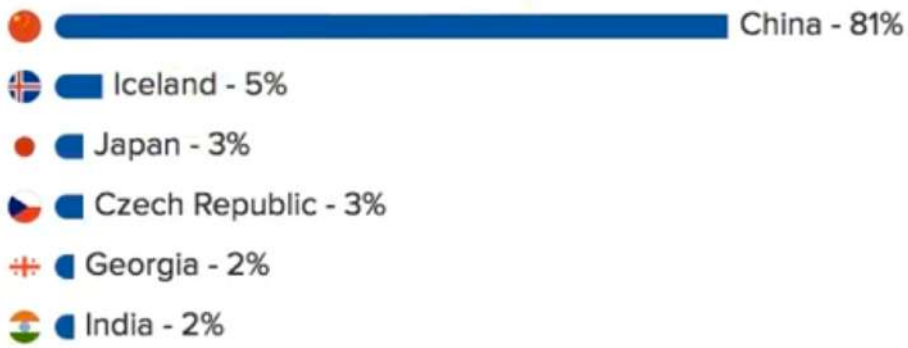
How the reward split among them? The reward will split based on the hashing power they introduced into the mining pool. So if you share 1G of GPU and your friend shares 5G of his GPU, and your mining pool wins, you get sth like 10$ and your friend gets 50$.



So you can buy a Cryptographic GPU mining Rig and download a software of a mining pool and instantly redirect all the calculations to that mining pool.



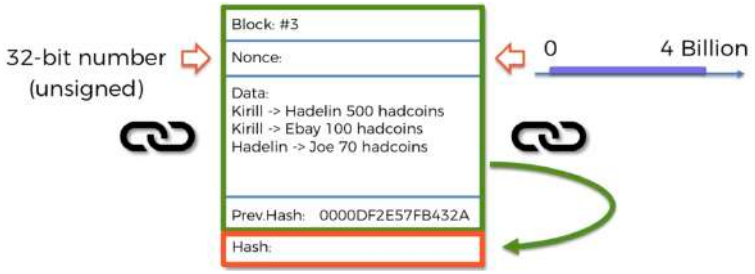
A lot of mining processes are happening in China, cause the price of Electricity is quite High over there.



If Bitcoin miners where a country they'd rank 61st in the world in terms of electricity consumption.

Nonce Range

Note: The Nonce is not Infinite. We cannot keep incrementing the Nonce forever. The Nonce is a 32-bit number. There's only 32 bits of memory allocated in every single block for the Nonce.
So the Nonce is an integer which has a range. 0 to 4 Billion.



Is the range of the Nonce enough for finding the correct hash?

So if we still go through all the Nonce numbers of its range, the probability of finding a valid hash is quite low.

Let's do some estimations:

Difficulty:
Total possible 64-digit hexadecimal numbers: $16 \times 16 \times .... \times 16 = 16^{64} \approx 10^{77}$
Total valid hashes (with 18 leading zeros): $16 \times 16 \times ... \times 16 = 16^{64-18} \approx 2 \times 10^{55}$
Probability that a Randomly picked hash is valid: $2 \times 10^{55} / 10^{77} = 2 \times 10^{-22} = 0.00000000000000000002\%$

Nonce:
The Nonce is a 32-bit number, the Max Nonce = $2^{32} = 4{,}294{,}967{,}296 = 4 \times 10^{9}$
Assuming no collisions, this means 4×10^{9} different hashes
Probability that ONE of them will be valid: $4 \times 10^{9} \times 2 \times 10^{-22} = 8 \times 10^{-13} \approx 10^{-12} = 0.0000000001\%$

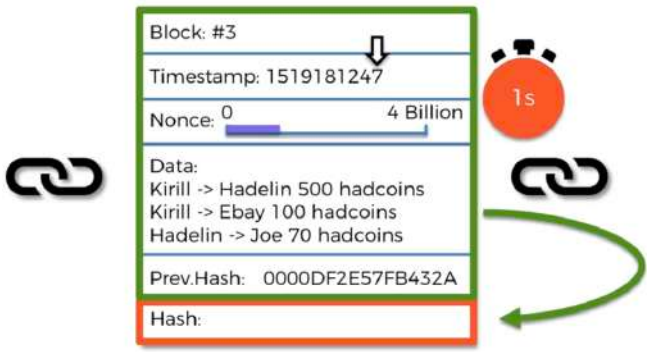
Conclusion: One Nonce Range is not enough

The modest minnde does 100 Million hashes per second = 100 M hashes
4 Billion / 100 million = 40 seconds.

Note: Timestamp

There is another data in a block which is **Timestamp.** The value of Timestamp changes in every second ! So the Block's hash changes every second.

Conclusion: The miner have only 1 second to go through Nonce range from 0 to 4 Billion. But it can't. So once 1 second passed, all of the Nonce values in the range, are valid again for testing.
Since 40 seconds are needed for looping through Nonce range, the miner will actually iterate the 1/40 of the Nonce range.

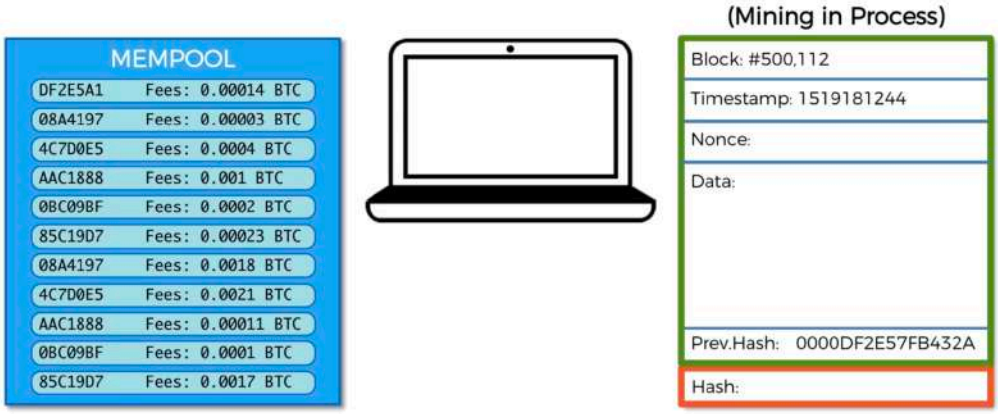


Since miners are a part of mining pool, Mining is still cost benefit. Cause the mining pool will get the correct Nonce, in a time even way less than a second.

The current hashes that are testing is: 22 Million Trillion hashes / Second.

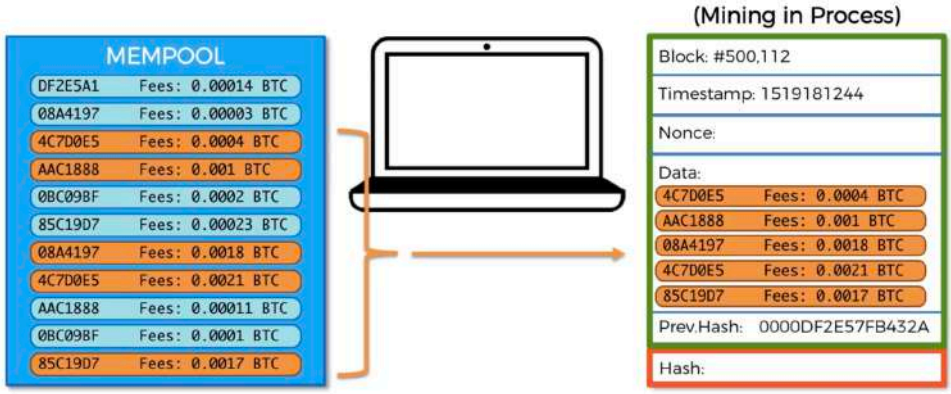
How miners pick the list of Transactions?

The real transactions come from a **Memory Pool** (Mempool), which is attatched to every node (every miner).
Basically these transactions are all the **unconfirmed transactions** stored in memory before being included to the block as Data.
In Bitcoin, blocks are added about every 10 minutes but transactions happen all the time, that's why they are **staging** in the Mempool.



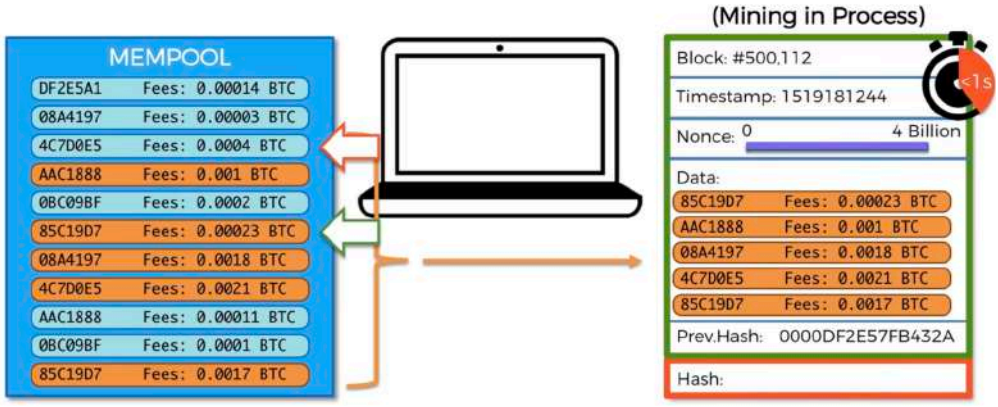
Let's say a miner is allowed to add a maximum of 5 transactions in a block, (in blockchain world the limit is 1MB which is about 2000 transactions)

Any transaction has a hash number and a fee assigned to it.
Fees are Non-Compulsory and they're specified by users themselves.
However the miner will pick the transaction with the highest fee.



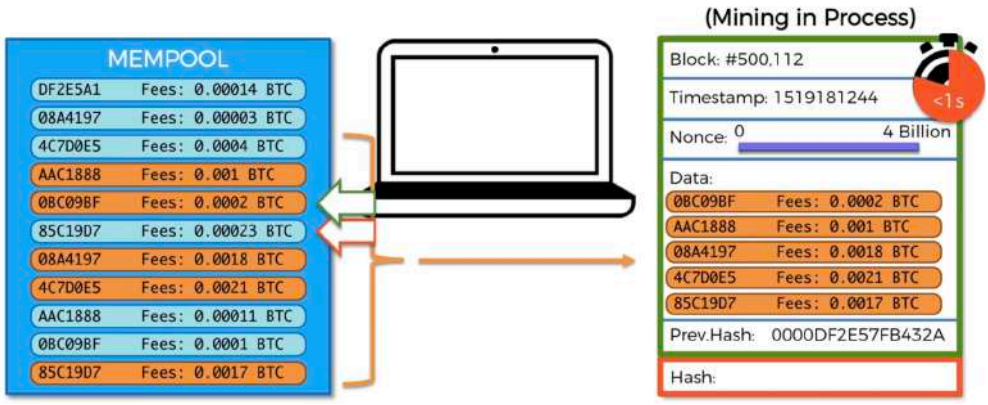
Question)
If the whole whole mining pool can resolve the hash in less than a second, what the Nodes should do at the rest of that second? Do they really have to wait until the timestamp updates? (Cause they don't need to increment the Nonce anymore, the hash number is already solved.)

The answer is: **Change the Block Configuration**.
In this situation the node changes the Data. Since the node has already chosen the transactions with highest fees and placed them in the Data of the block, it can now, remove one of the chosen transactions with lowest fee and replace it with another transaction of Mempool with the highest fee.

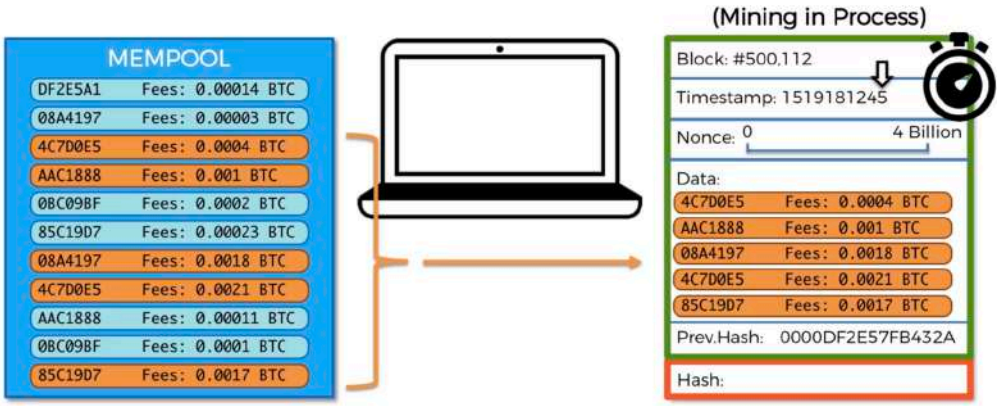


So, by changing the Data, the correct hash for the block will be changed and the node can again reuses the Nonce range in the rest of the second for resolving the hash number.

Then, if the mining pool again resolved the hash, the node can remove one of the chosen transactions with the lowest fee and replace it with the next transaction with the highest fee from Mempool.



Note:
Once that one second is passed, the timestamp will be updated and the node replaces the first chosen transaction with highest fee, back in the block again!



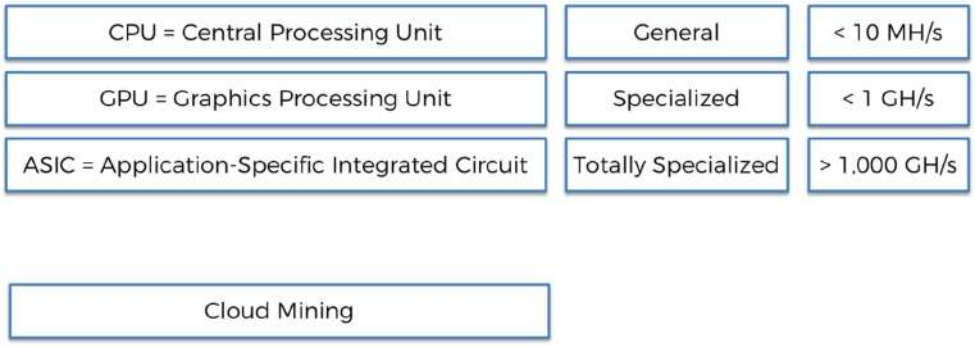
There are about 7000 transactions in the Mempool that the algorithm algorithm combines the best transactions with the best possible way, in order to get the maximum of the fees and also in order for the miners not to duplicate their work for different miners in the pool.
So the algorithm allocates the transactions with miners, it allocates the transaction fees, allocates the Nonces.

Note: If you specify a low fee like Zero, it will never be resolved and be stock in the Mempool.

Which Hardware? CPIs vs GPUs vs ASICs :

CPU = Central Processing Unit
It's very powerful but it's very general. It's not specialized to one specific task.
It is limited to solve 10 Mega hashes per second.
GPU = Graphic Processing Unit
It is less pwoerfull than CPU with lower functionality. it's specialized,
It is limited to solve 1 Giga Hashes hashes per second.
ASIC = Application-Specific Integrated Circuit
It is totally specialized to solve just the SHA256 hash. The device is doesn't responsible to do anything else. It can solve up to 1000 Giga Hashes per second.

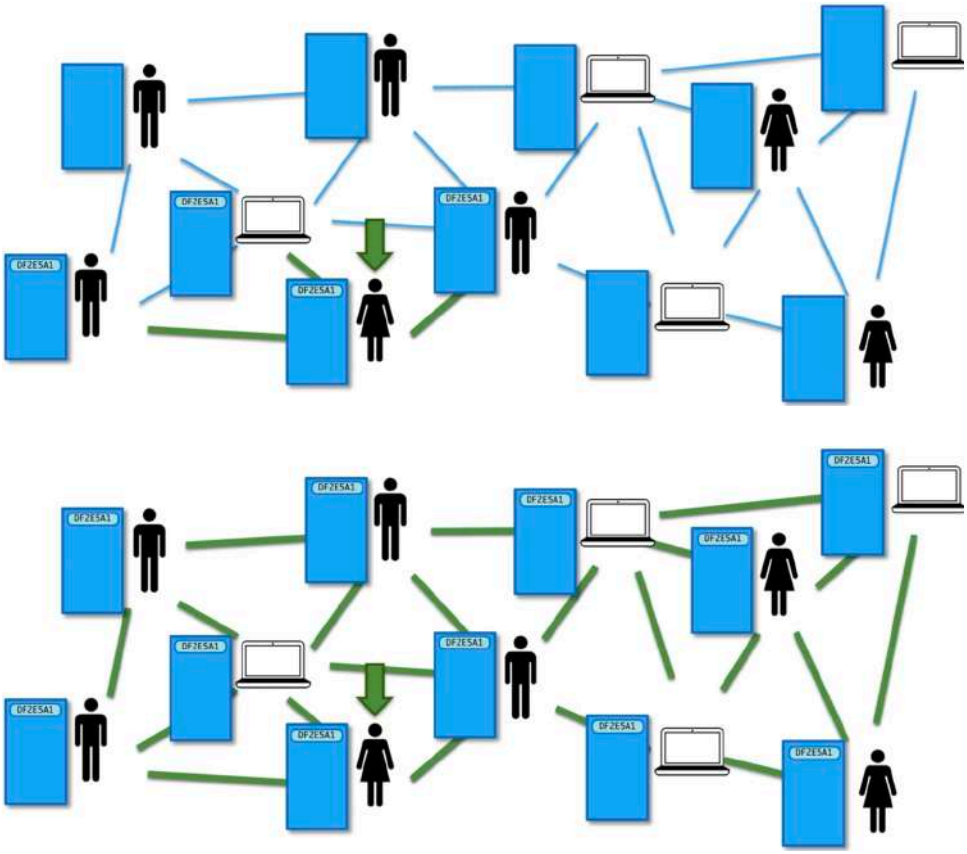
You can also rent a machine, named Cloud Mining



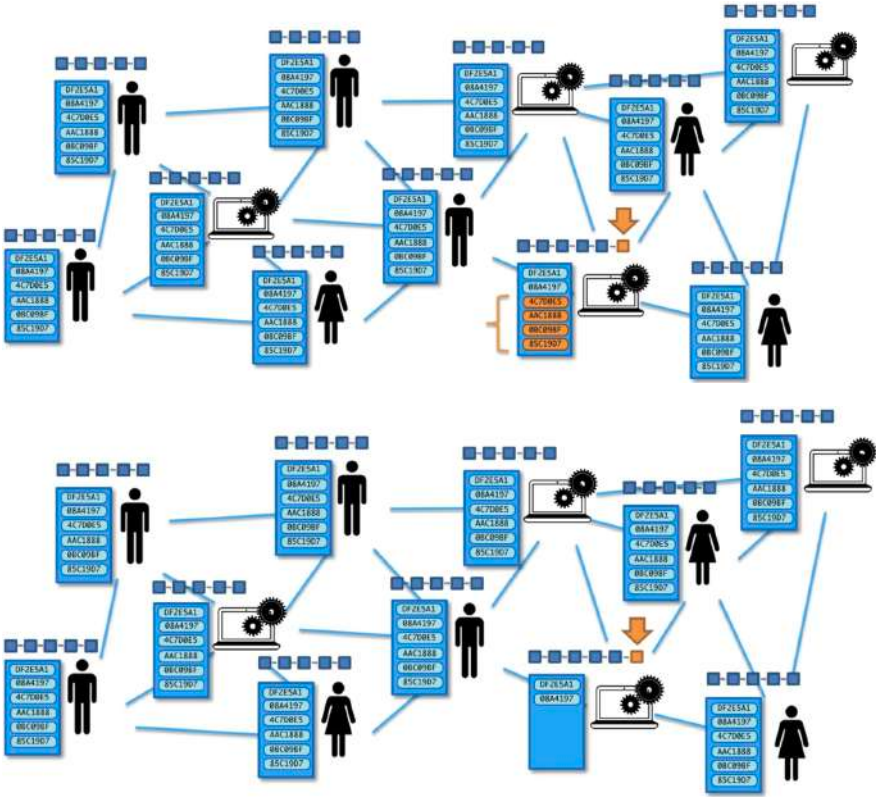
How Mempools work?

There is a Mempool for each participant: a Node or miner. There is NOT a mempool, shared with everybody because in Peer2Peer Network, there is nothing central about it !
Mempools are a staging area for transactions go before they're added to a block.

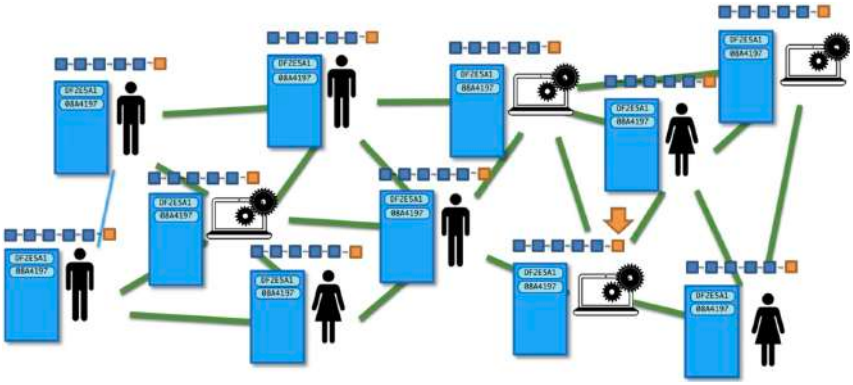
1. Imagine a user wants to do a transaction, so that transaction gets added to their Mempool.
2. Then, that transaction gets broadcasted across the Network. So that transaction gets added to their mempools.
3. Finally that transaction gets added to every single Mempool through the Network.
4. Then, another user or miner adds a transaction, and broadcast it and every mempool will finally has it.



Imagine a miner found the hash for a block. It has already selected some of the transactions from Mempool to be added to the last block. After solving the hash, the miner should remove those selected transactions from mempool



Then, that block broadcasts through the network, and then all nodes will add the block to their chain and remove the related transactions from their Mempool.

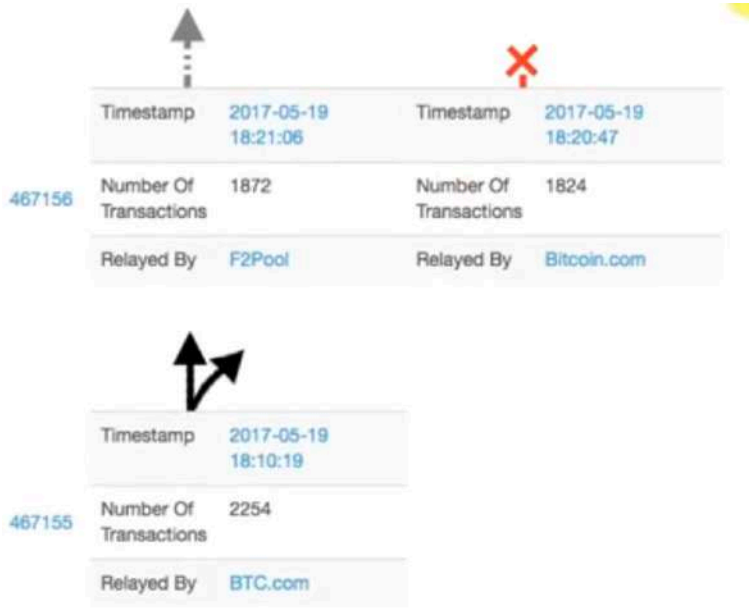


Double Spend Problem:

Generally it is a rule to wait for 6 confirmations before considering our transaction to be successful.

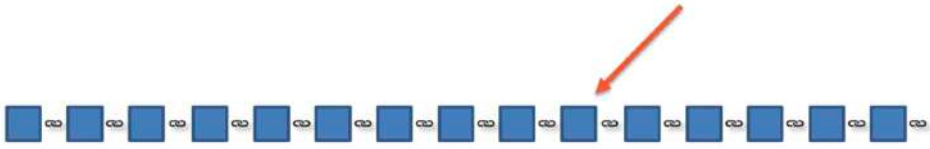
Orphaned Blocks:

Sometimes if 2 miners produce blocks at similar times, they both claim that the next block will be their block, even if they completely have different transactions. that's why we need to wait for other confirmations and then add the right block.



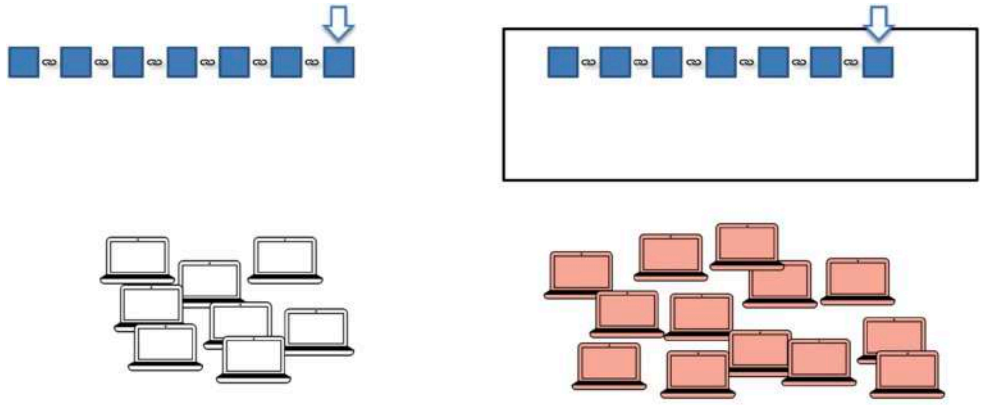
The 51% Attack:

The 51% Attack is not an attack designed to tamper with a selected block

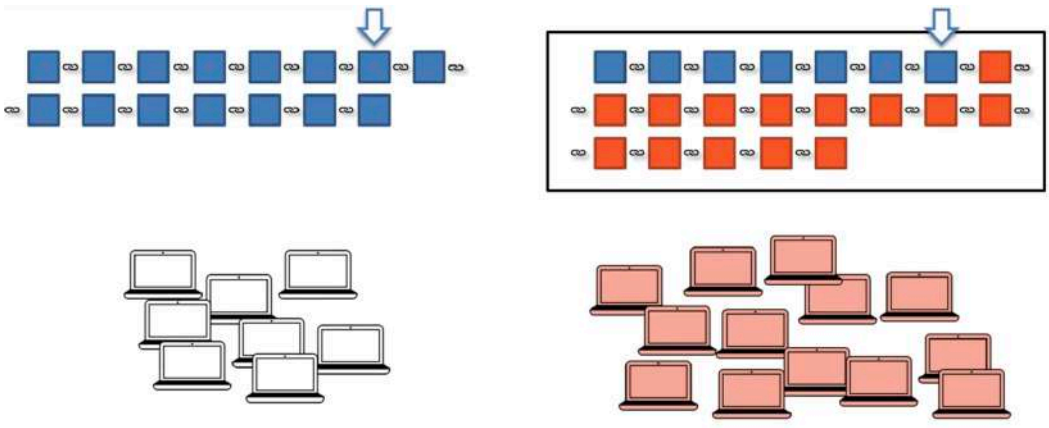


This is NOT the 51% attack

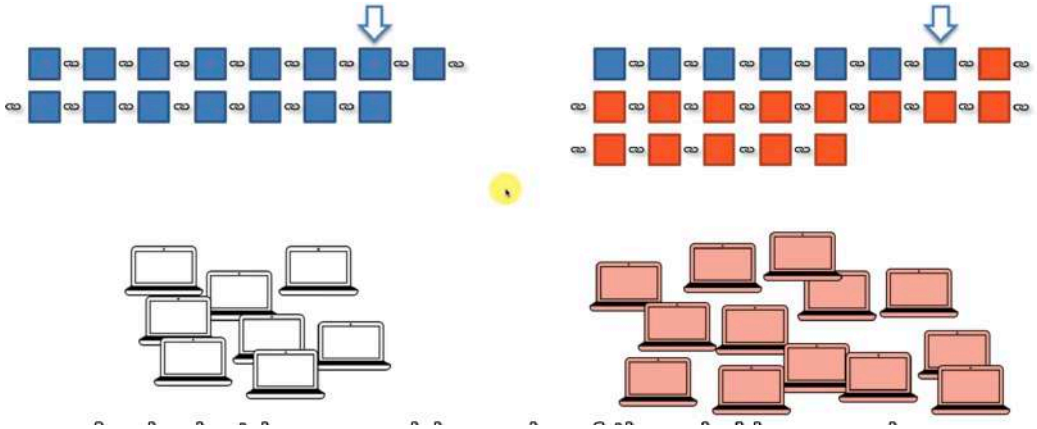
The 51% Attack is the time that a group of nodes (which are malicious) come and join the network. So they get a copy of the chain and get their mempools and setup everything.
Once they got the chain, they will close the connection with other nods, and **don't broadcast** anything back to the network and then keep mining.



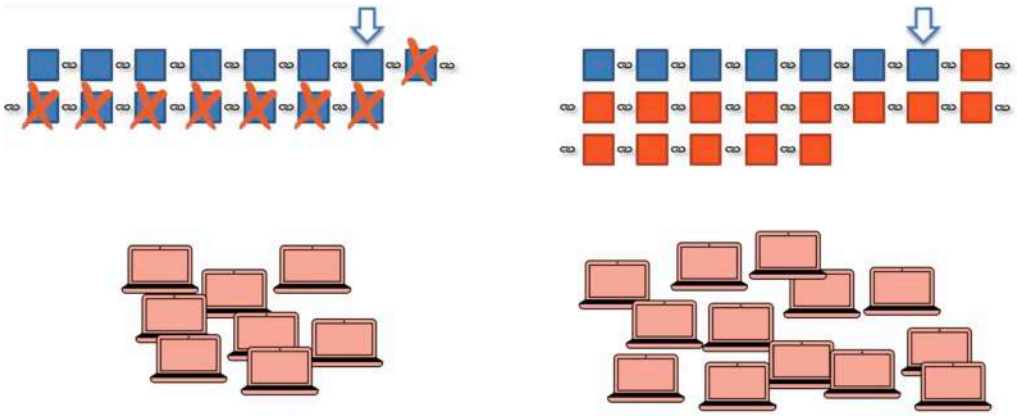
Finally because they have more nodes than the previous mining pool, as time passes, they solve more blocks, and the difference between them, increases.



Then, out of a sudden, they will open up the connection and **start to broadcast** with other nodes.
Since the golden rule for blockchain system is that the longest chain wins,



Therefore the blocks of previous mining pool will be invalid. And their nodes will change their chain into the new chain.



Then, all of the transactions of the invalid blocks, will be returned to the Mempool !!!

So basically those malicious nodes didn't do anything illegal, but the crash the network.

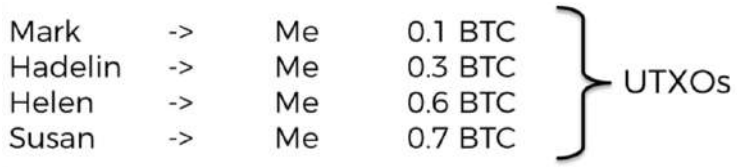
This kind of attack is kind of not possible in system like Blockchain, but it can be done in smaller blockchains.

Transactions and UTXOs (Unspent Transaction Outputs):

In banking system, once you sent a transaction, the transaction is done and it's recorded in the Database of the bank.

In Bitcoin, a transaction lives on after it's been executed until another transaction builds of the UTXOs from that transaction.

In Bitcoin or any cryptocurrency, there is no account for storing your balance. Everything is based on blockchain. And any movement of money is based on transaction on the blockchain.



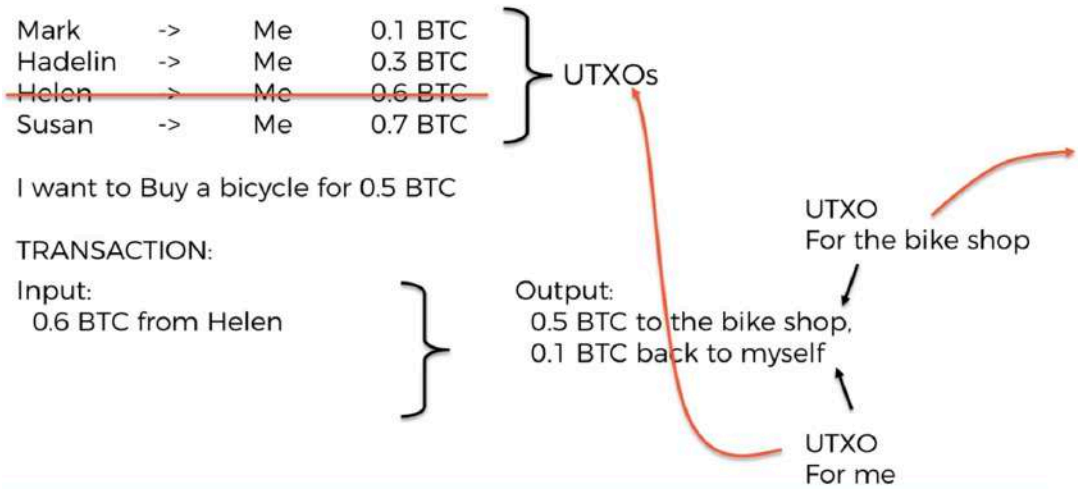
Since there is no account and all you have is a set of transactions that people sent you money in. (Called UTXOs = Unspent Transaction Outputs)

Now imagine you want to **buy a bicycle for 0.5 BTC** .

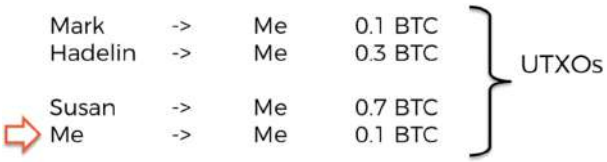
You need to select one of the transactions in your UTXOs as input.
(like: Helen -> Me : 0.6 BTC)

Then create 2 other transactions. One for buying the bicycle (0.5 BTC) and since you cannot split the amount of the input transaction, you need to create one transaction for sending (0.1 BTC) to yourself.

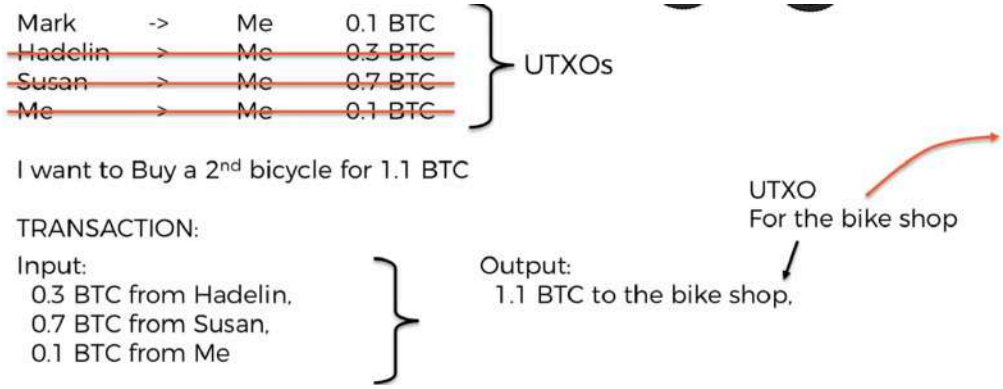
The used transaction is no longer a UTXOs and should be removed from that list.



New UTXOs will be:

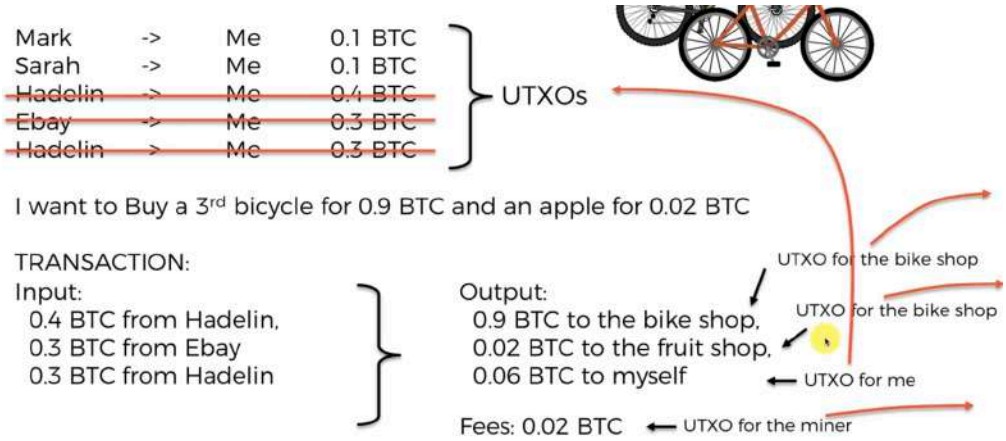


Now imagine we want to buy a bicycle which is more expensive than a single transaction of ours.



Where do transaction fees come from?

Anything that we don't account for, becomes the fee for this transaction to be included in a block of the blockchain.
The higher fee is, the more likely the question will be accepted or sooner. The fee is calculated automatically.



How wallets work?

The wallet does an analysis in all transactions and check for the specific transactions that came back to ourselves. And then the wallet checks that which one of these transactions still has the UTXO.
So all it needs to do is just to add up the UTXO amounts of them.

So the number (value) that is shown in our wallet does NOT exist ! The thing that exists are the transactions to ourselves which are UTXO. And the Balance value is just a summation of them.

Signatures: Private & Public keys

For any node, there is a unique **Private Key** that is assigned to it. It is a secret unique identifier that no one never knows about it. (Just like a password to our bank account.)
From that Private key, we can generate a **Public key**. This is the code we can share with others to send and receive money.
Then we can send a message. (It can be any sort of data, but in case of cryptocurrency, it is transaction or transactions)

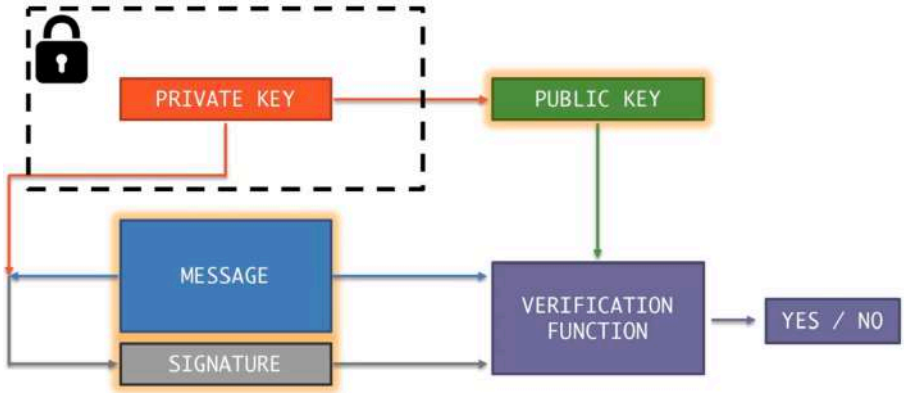
The Private key combines with the message and create a signature. (The Private key is used to sign the message.)
Since only we know the private key, so the created signature will be unique to us.

So the message and its signature always be sent with each other.

Question :

How other nodes can verify that it was you that sent this message with this signature?

In Blockchain, there is a Verification function that anybody who has the message + your signature + your public key, can put them in that function to see if you are the owner of the message or not. (Use the public key to assess if the signature is from the Private key which generated the Public key.)

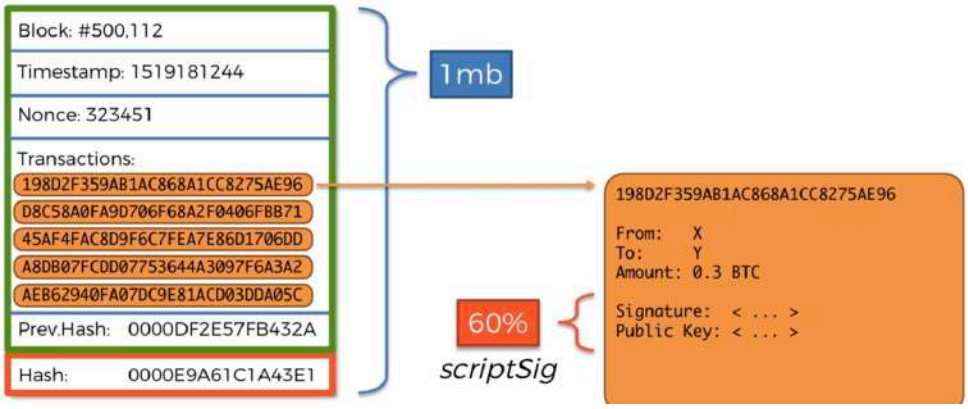


Segregated Witness (SegWit):

There is size limit for any block. For example a 1mb. If the size of a block increases, it takes more time to go through the network, so:

1. Since the size is too large, approving the transactions will be slow.
2. Attackers can have more time to hack the system.
3. Chance of orphaned blocks increases (the ones that stuck in the)

Any transaction, is not just a number. It includes **From, To, Amount, Signature, Public Key.**
And the **signature** and **Public key**, contain 60% of the size of the whole transaction.
The **Verification Function** takes this 60% part of transaction (called scriptSig) and remove it from the message and send it tho the network. It is still connected to the transaction but it will go through the network, separately.
This is how we reduce the size of the transactions and can fill up more transactions in the block.

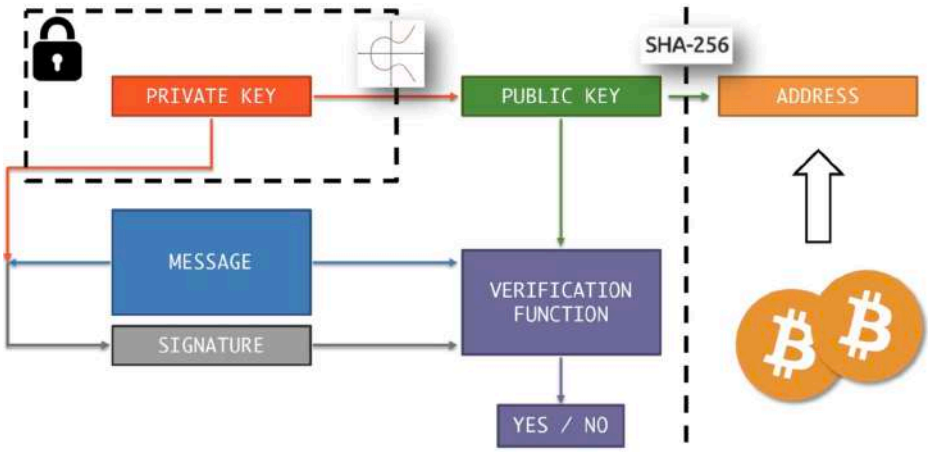


Segregated Witness (SegWit) refers to **a change in Bitcoin's transaction format where the witness information was removed from the input field of the block**. The stated purpose of Segregated Witness is to prevent non-intentional Bitcoin transaction malleability and allow for more transactions to be stored within a block.

Public Key vs Bitcoin Address:

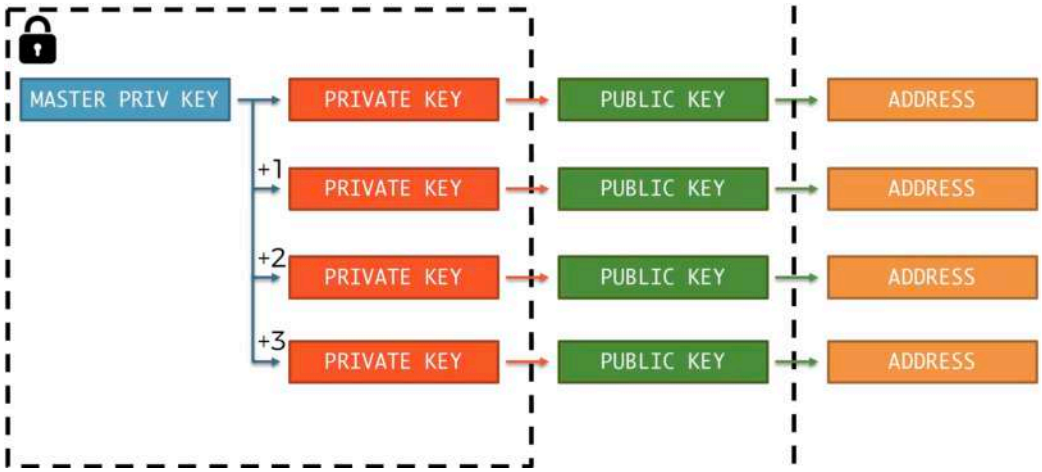
There is another address just like the public key that we can share with others, so others can identify where they can send you bitcoin or other tokens.
Since the public key and this address is similar to each other, others can send you bitcoins just by having the Public key, however, we want to separate the public key and the address.
Of course when we want to send money (create a transaction) we need to expose the public key in order to verify the owner of the transaction.

But why we should avoid exposing the public key when we can? Cause if someone could find a way to go from Public key to Private key, they can hack our transaction. that's why we use SHA256 algorithm to generate another address.

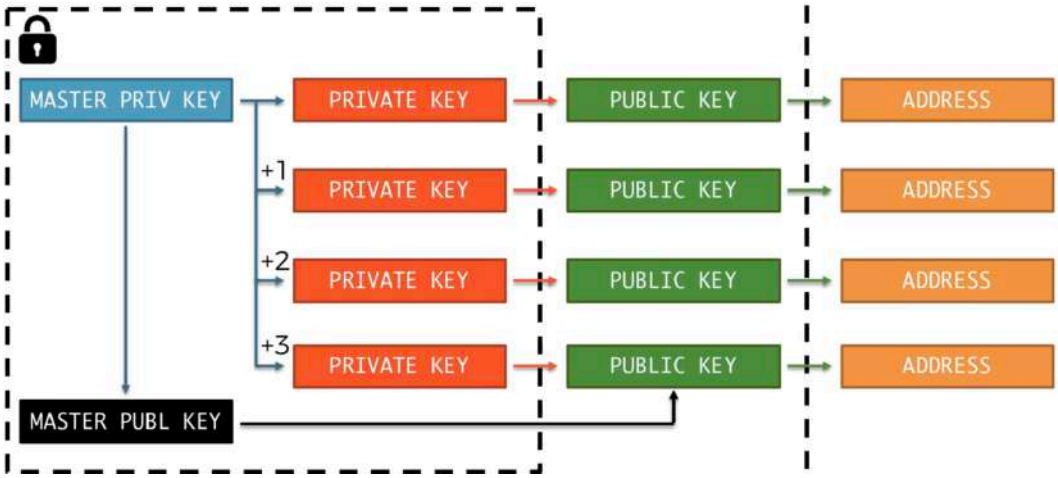


Hierarchically Deterministic (HD) Wallets:

With an HD wallet, you will have a Master Private Key. Form that Master Private Key, a Private Key will be generated. Then it is used to generate the Public Key and the Public key is used to generate the address.
The difference here is that the Master Private Key can generate multiple Private keys to increase the security.



We also have Master Public Key, can recreate any Public key above. The Master Public Key should not necessary be located privately. so the picture below is wrong.



So how do we keep the Master Private Key safe?

Most of the time, the wallets give us 12 words to memorize. Those 12 words are responsible to generate our Master Private Key. So by knowing those 12 words, we can recreate the Master Private Key.