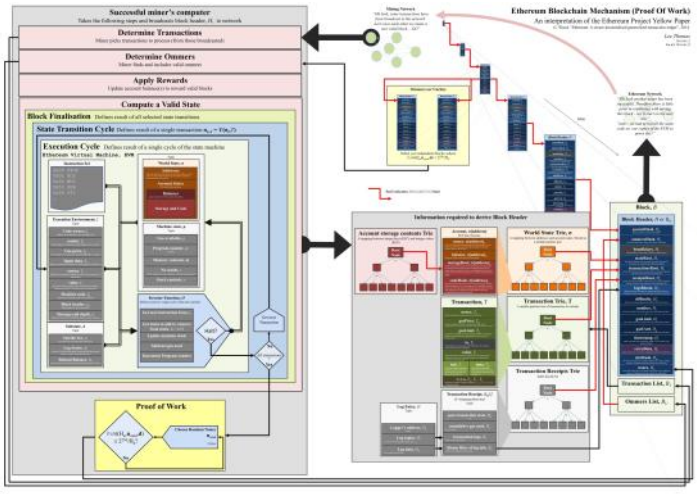
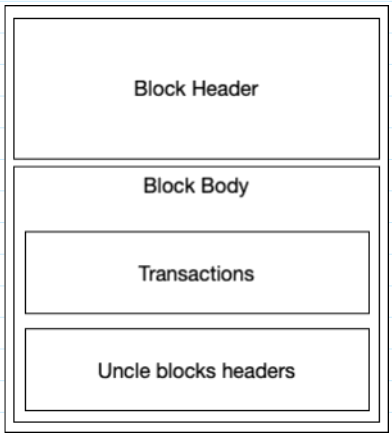


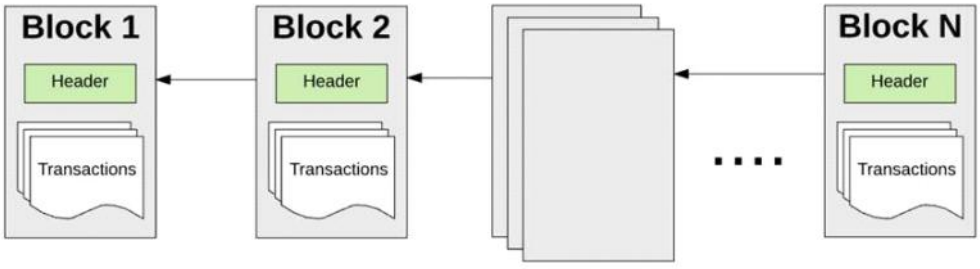
Ethereum architecture formally-mathematically is presented in Ethereum **Yellow Paper**
Ethereum Yellow Paper: a formal specification of Ethereum, a programmable blockchain
The Ethereum Yellow Paper serves as the definitive technical document for the Ethereum protocol. It was initially written by **Gavin Wood** and is now maintained by **Andrew Ashikhmin** along with contributions from various experts globally.

Ethereum architecture structurally is presented in:
blockchain - Ethereum block architecture - Ethereum Stack Exchange

A high level diagram of the Ethereum block.



Ethereum block is divided in two parts, the **block header** and the **block body**.
The block header is the blockchain part of Ethereum. This is the structure that contains the hash of its predecessor block (also known as parent block), building a cryptographically guaranteed chain.
The block body contains a **list of transactions** that have been included in this block and a **list of uncle (ommer) blocks headers** (if you want to know more about uncle blocks recommend this post).



All transactions are grouped together into “blocks.” A blockchain contains a series of such blocks that are chained together.

5.2.2. Ethereum Block content.

In Ethereum, a block consists of:

- the **block header**
- information about the **set of transactions** included in that block
- a **set of other block headers for the current block’s ommers**.

Fields in the block header:

- **parentHash**
  Hash of the block header from the previous block. Each block contains a hash of the previous block, all the way to the first block in the chain. This is how all the data is protected against modifications (any modification in a previous block would change the hash of all blocks after the modified block).
- **ommersHash**
  Hash of the uncle (ommer) blocks headers part of the block body.
- **beneficiary**
  Ethereum account that will get fees for validating this block.
- **stateRoot**
  Hash of the root node of the world state trie (after all transactions are executed).
- **transactionsRoot**
  Hash of the root node of the transactions trie. This trie contains all transactions in the block body.
- **receiptsRoot**
  Every time a transactions is executed, Ethereum generates a transaction receipt that contains information about the transaction execution. This field is the hash of the root node of the transactions receipt trie.
- **logs**
  Bloom filter that can be used to find out if logs were generated on transactions in this block (if you want more details check this Stack Overflow answer). This avoids storing of logs in the block (saving a lot of space).
- **Difficulty**
  Only in Proof of Work - **PoW** consensus mechanism for mining is used. It is revoked in Ethereum in 2022.
  Difficulty level of this block. This is a measure of how hard it was to mine this block.
  In recent consensus mechanism the Proof of Stake- **PoS** is used instead.
- **number**
  Number of ancestor blocks. This represents the height of the chain (how many blocks are in the chain). The genesis block has number zero.
- **gasLimit**
  Each transaction consumes gas. The gas limit specifies the maximum gas that can be used by the transactions included in the block. It is a way to limit the number of transactions in a block.
- **gasUsed**
  Sum of the gas cost of each transaction in the block.
- **timestamp**
  Unix timestamp when the block was created. Note that due to the decentralized nature of Ethereum, we can't trust in this value (especially when implementing smart contracts that have time related business logic).
  extraData
  Arbitrary byte array that can contain anything. When a validator is creating the block, it can choose to add anything in this field.
- **mixHash**
  Hash used to verify that a block has been mined properly by placing a validator's signature on it.
- **nonce**
  Same as the mixHash, this value is used to verify that a block has been mined properly.

Ethereum Logs

Allows for logs to make it possible to track various transactions and messages. A contract can explicitly generate a log by defining “events” that it wants to log.

A log entry contains:

- the logger’s account address,
- a series of topics that represent various events carried out by this transaction, and

- any data associated with these events.

Logs are stored in a bloom filter, which stores the endless log data in an efficient manner.

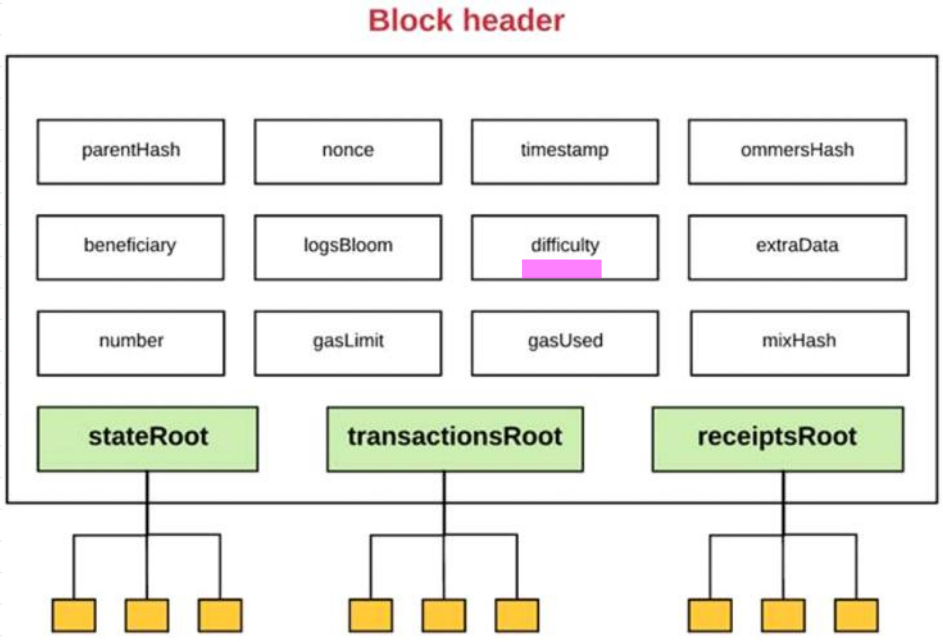
Transaction receipt

Logs stored in the header come from the log information contained in the transaction receipt. Just as you receive a receipt when you buy something at a store, Ethereum generates a receipt for every transaction. Like you'd expect, each receipt contains certain information about the transaction. This receipt includes items like:

- the block number
- block hash
- transaction hash
- gas used by the current transaction
- cumulative gas used in the current block after the current transaction has executed
- logs created when executing the current transaction

Block header contains three main Merkle-Patricia trie structures for:

- state (**stateRoot**)
- transactions (**transactionsRoot**)
- receipts (**receiptsRoot**)



Ethereum transitioned to a **proof-of-stake** consensus mechanism on **September 15, 2022**.

This significant upgrade, known as **The Merge**, marked the shift from a proof-of-work (**PoW**) system
to proof-of-stake (**PoS**),
resulting in a reduction of energy consumption by 99.98%.

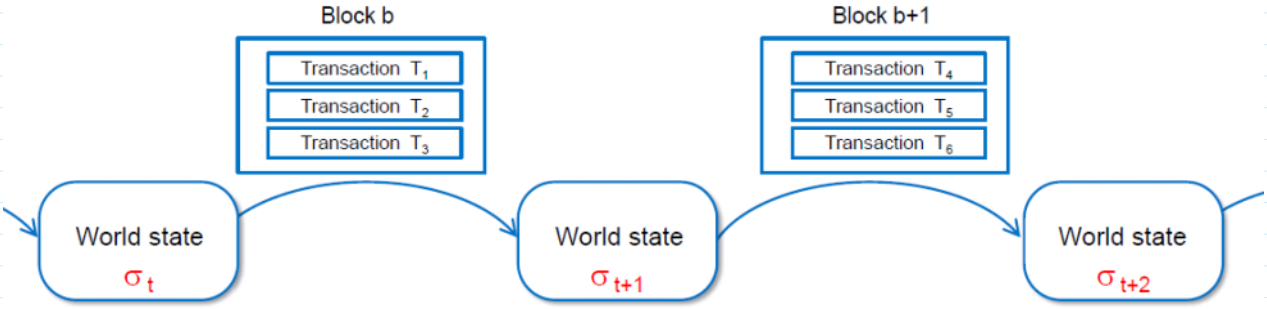
From <https://www.bing.com/search?q=ethereum+proof+of+stake+date&qs=UT&pq=ethereum+proof+of+stake&sk=HS1&sc=11-23&cvid=C0D9675E158B43FDA60561A99734B6F2&FORM=QBRE&sp=2&lq=0>

As a result the field **difficulty** lost it sence.

An **ommer** is a block whose parent is equal to the current block's parent's parent.
Why a block contains the block headers for ommers.
Because of the way Ethereum is built, block times are much lower (~15 seconds) than those of other blockchains, like Bitcoin (~10 minutes). This enables faster transaction processing. However, one of the downsides of shorter block times is that more competing block solutions are found by miners. These competing blocks are also referred to as "orphaned blocks" (i.e. mined blocks do not make it into the main chain).
The purpose of ommers is to help reward validators for including these orphaned blocks. The ommers that miners include must be "valid," (in the case of **PoW** meaning within the sixth generation or smaller of the present block. To prevent forgered blocks mining. (See so called **51% attack**). After six children, stale orphaned blocks can no longer be referenced (because including older transactions would complicate things a bit).
Ommer blocks receive a smaller reward than a full block. Nonetheless, there's still some incentive for miners to include these orphaned blocks and reap a reward.

5.2.3. World State.

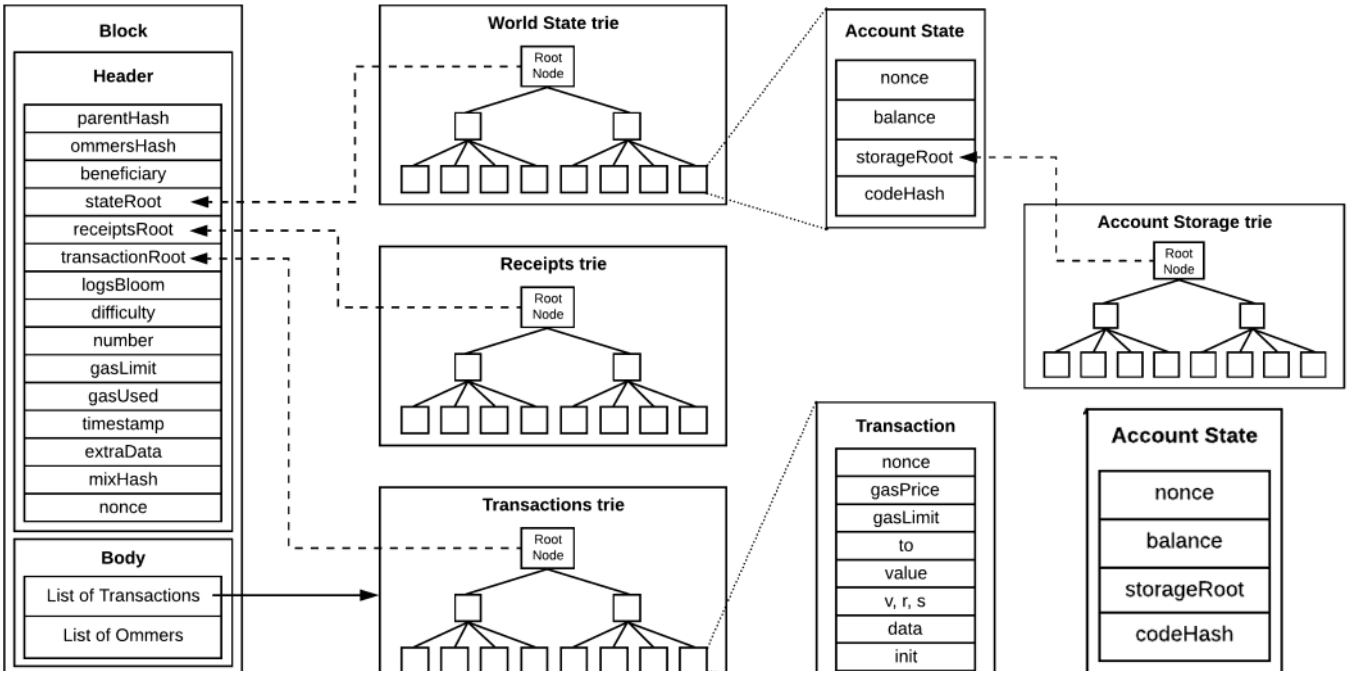
Chain of states: Ethereum can be seen as a state chain.

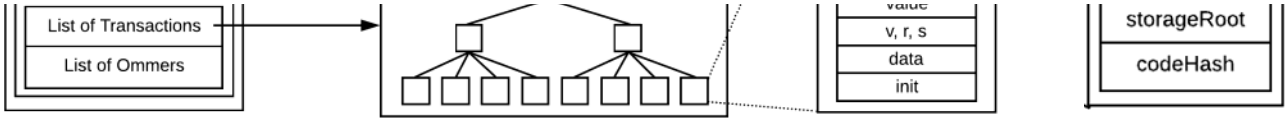


Notations $б_t$, $б_{t+1}$ and $б_{t+2}$ means the state transitions in time instances t, t+1 and t+2.

Ethereum has 4 types of tries:

1. **The world state trie contains the mapping between addresses and account states**. The hash of the root node of the world state trie is included in a block (in the **stateRoot** field) to represent the current state when that block was created. We only have one world state trie.
2. **The account storage trie contains the data associated to a smart contract**. The hash of the root node of the Account storage trie is included in the account state (in the **storageRoot** field). We have one Account storage trie for each account.
3. **The transaction trie contains all the transactions included in a block**. The hash of the root node of the Transaction trie is included in the block header (in the **transactionsRoot** field). We have one transaction trie per block.
4. **The transaction receipt trie contains all the transaction receipts for the transactions included in a block**. The hash of the root node of the transaction receipts trie is included in also included in the block header (in the **receiptsRoot** field); We have one transaction receipts trie per block.





And the objects depicted above are:

1. **World state trie**: the hard drive of the distributed computer that is Ethereum Virtual Machine (**EVM**). It is a mapping between addresses and account states.
2. **Account state trie**: stores the state of each one of Ethereum's accounts. It also contains the storageRoot of the account state trie, that contains the storage data for the account.
3. **Transaction**: represents a state transition in the system. It can be a funds transfer, a message call or a contract deployment.
4. **Block**: contains the link to the previous block (parentHash) and contains a group of transactions that, when executed, will yield the new state of the system. It also contains the **stateRoot**, the **transactionRoot** and the **receiptsRoot**, the hash of the root nodes of the world state trie, the transaction trie and the transaction receipts trie, respectively.

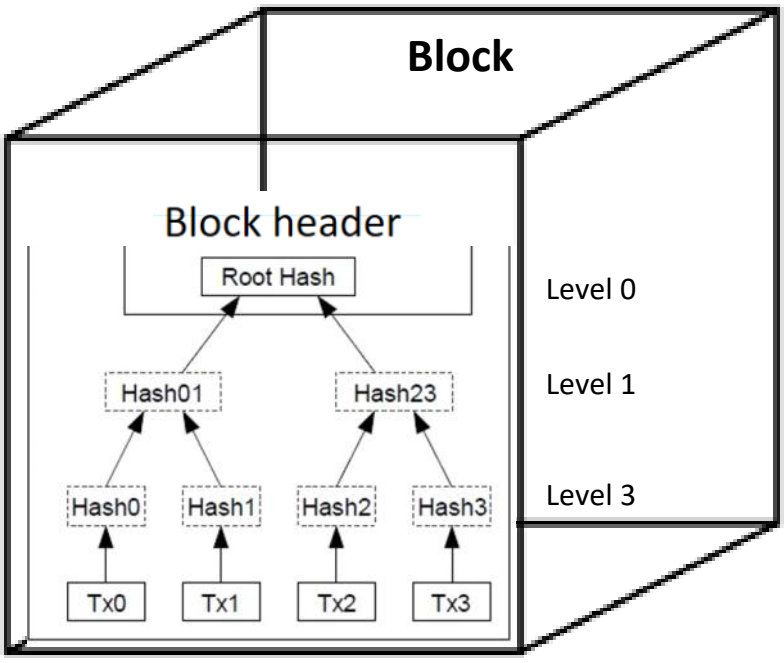
World State trie, also known as **Global State trie**, **is one of the most critical data structures in Ethereum**.

It serves as a snapshot of the current state of the entire network. The trie stores a mapping between account addresses (both Externally Owned Accounts - **EOA**s and smart contracts accounts **CA**) and their respective account states.

Understanding Ethereum Structures: World State Trie, Transaction Trie, Receipts, and Account Storage Trie, MPT | by Ricardo Santos | Medium

Transaction Trie is a data structure that stores all the transactions contained within a specific block. Every block has its own distinct Transaction Trie, which serves to store details related to transactions and their sequence within the block.Transactions are grouped into blocks before being added to the Transaction Trie. This means each block holds a set of transactions, and together they make up a series of blocks known as blockchain. The Transaction Trie is linked to the World State Trie and other tries through block hashes, ensuring data integrity and immutability.

Transaction Trie is included in Block header.

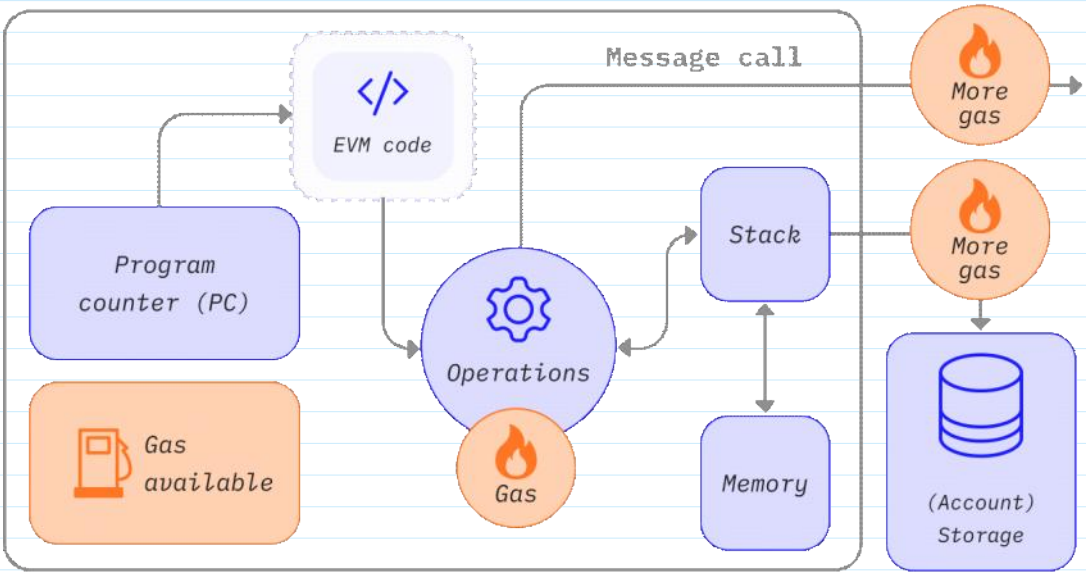


As it was mentioned in Account Storage trie, Transactions trie consist of **leafs**, **edges** and **vertices**.
A simplified Transactions trie is presented in picture having 4-**leafs** corresponding to 4 **transactions**, 7-**vertices** corresponding to h-values and 10-**edges** connecting leafs and vertices.

Then using a H-function **Keccak-256** the h-values has 256 bit length in the **vertices** are computed in the following way:
Hash0=Keccak-256(Tx0)
Hash1=Keccak-256(Tx1)
Hash2=Keccak-256(Tx2)
Hash3=Keccak-256(Tx3)
Hash01=Keccak-256(Hash0||Hash1)
Hash23=Keccak-256(Hash2||Hash3)
RootHash=Keccak-256(Hash01||Hash23)

The illustrative Merkle-Patricia trie presented above is so called binary tree.
Every vertex has 2 incoming edges of 2 verices and 1 outcomming edge.
In general trees are very effective for data representation, data authenticity and integrity ensurance.
In the case of transaction tree above we can see 3 levels of representation.
In Level 0 we can represent 1 data unit, i.e. RootHash vertice, computing the number of represented vertices by $2^0=1$.
In Level 1 we can represent 2 vertices, since in binary tree case $2^1=2$.
In Level 2 we can represent 4 vertices, since in binary tree case $2^2=4$.
Question 1: how many vertices corresponding to leafs representing transactions we can represent by **L** levels?
Answer: $\mathbf{2^L}$.
Question 2: how many levels **L** we must have to place 1024 transactions in binary trie?
Answer: L=10, since $\mathbf{2^{10}}=1024$.
As we see the representation amount is increasing **exponentially** by increasing the number of levels **L**.

Ethereum's world state is a large data structure which holds not only all accounts and balances, but a
EVM state, which can change from block to block according to a pre-defined set of rules, and which can execute arbitrary machine code, i.e., **EVM** is Turing complete mashine.
The specific rules of changing state from block to block are defined by the **EVM**.
The **EVM** also implements a number of blockchain-specific stack operations, such as ADDRESS, BALANCE, BLOCKHASH, etc.



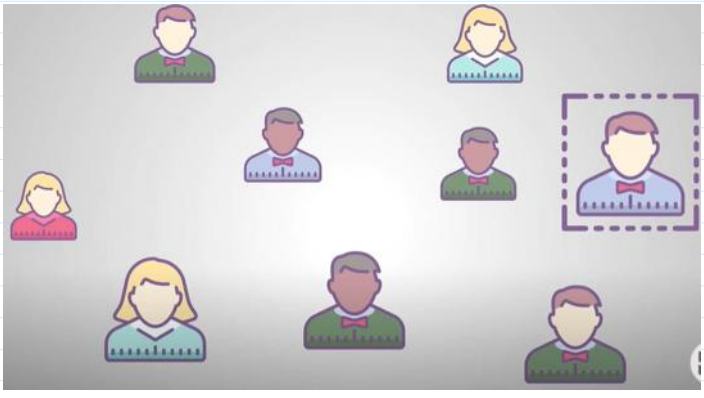
References.
https://ethereum.org/developers/docs/evm/
https://parfin.io/en/blog/the-ethereum-guide-understanding-ethereums-structures

5.2.3. PoW vs PoS.

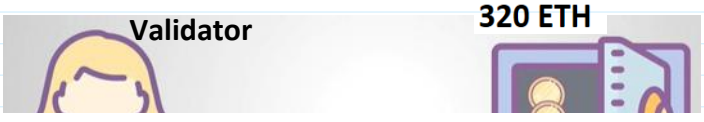
B111 2024-Pa

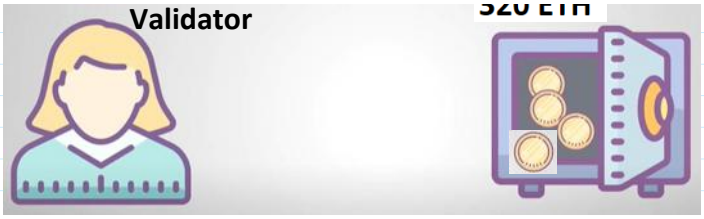
The initial concensus mechanism of blocks validation in the most blockchains was **Proof of Work** - (**PoW**).
As it was mentioned above, on **September 15, 2022, PoW** mechanism was changed to a **Proof-of-Stake** consensus mechanism.

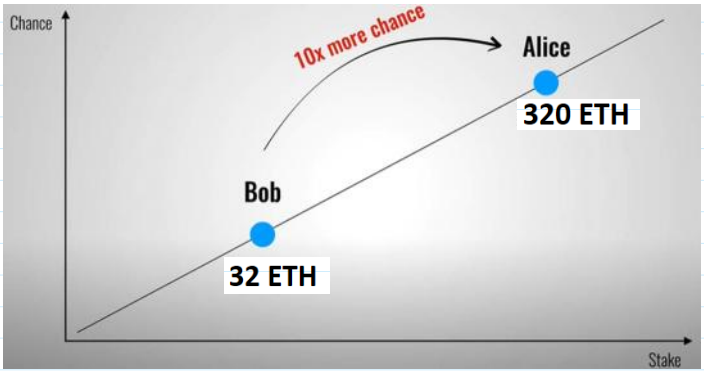
vs



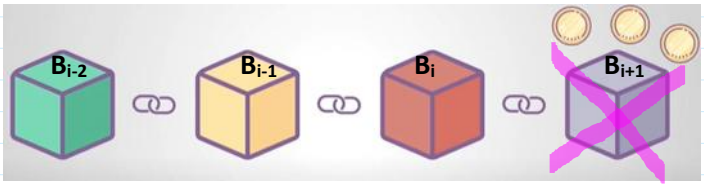
To mine a block according to **PoS** consensus mechanism, the validator must put the predetermined sum as a deposit in a "Shell".
This sum some time ago was at least 32 ETH.







According to **PoS** mechanism, the more money is put into the "Shell" the more chances is to get right to mine a Block and to get a reward.
A Block consist of a lot of Transactions having a gas for validator as an incentive for validation.
Let Alice puts 32 ETH and Bob puts 32 ETH into the "Shell".
Then Alice has 10x more chances to get to get a block for validation than Bob.

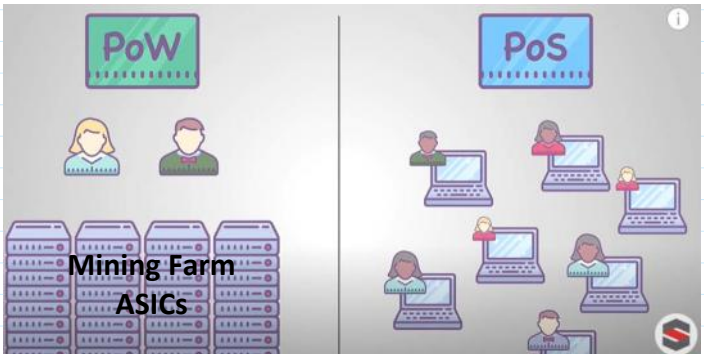


Mistakenly validated Block

If nodes find that some Block is validated mistakenly, then according to **PoS** mechanism the certain sum of such validator is confiscated.

If Alice was mistakenly validated Block then her deposit was reduced.

Validator can empty his/her deposit after some time.



In **PoW** mechanism Validators are called Miners and they are using a lot of electrical energy to mine a block. For this purpose they are using Mining Farm connected to high-voltage energy souces and equiped with Application Specific Integrated Circuits (**ACIC**s) to increase the mining probability and to win the mining competition.
In **PoS** mechanism it is needed to have a

competition.
In **PoS** mechanism it is needed to have a deposit and ordinary computer.

A lot of Validators can get a reward with probability depending on deposit amount.

The End