Lecture Notes

Smart Contract Development in Ethereum
Module 1 – Ethereum Fundamentals

Session 1 – Ethereum Architecture

In this session, you have learnt about:

- The history of Ethereum
- Ether, which is the digital currency to trade in the Ethereum space
- Gas, which is a unit to measure the amount of computational effort and, hence, the execution fee for every transaction on Ethereum
- Ethereum accounts and how transactions are performed in Ethereum
- How Ethereum is different from Bitcoin

Let us summarise all the topics that we have covered in this session.

Introduction to Ethereum

Ethereum is a decentralised, open-source, distributed computing platform that enables the creation of smart contracts and decentralised applications, also known as Dapps.

Ethereum was developed by **Vitalik Buterin** in 2013. Today, Ethereum hosts more than 90% of the new tokens on its platforms. It has a market capitalisation of $70 billion.

Ethereum was developed in order to overcome some of the challenges with and inefficiencies of Bitcoin.

These challenges and inefficiencies include the following:

- **Bitcoin is specific to one purpose only:**
    - Ethereum can be used to build various applications.
- **Bitcoin mining was becoming centralised:**
    - Ethereum developers are working on shifting the mining algorithm from Proof of Work (PoW) to Proof of Stake (PoS).

The architecture of Ethereum and Bitcoin is fairly similar, except their 'Transaction Processing Model'.

The Ethereum blockchain is essentially a transaction-based state machine.

> **State machine** refers to something that would read from a series of inputs and then transition to a new state based on those inputs.

The Ethereum blockchain starts from the 'genesis state', which is the first state. And after every transaction, the state of the entire blockchain changes.

Bitcoin follows the UTXO model, and it stores everything on the chain. By contrast, Ethereum stores all accounts off-chain.

Ether

Ether is the main currency on the Ethereum blockchain:

- It is a form of payment made by the client of the platform to the machine that executes its requested operation.
- Every time a block is validated, 5 Ethers are created and awarded to the successful node.

Gas

Gas is a unit to measure the fee required for a computation:

- It refers to the fee, or the pricing value, required to successfully perform a transaction or execute a contract on the Ethereum blockchain platform.
- The exact price of gas is determined by the network's miners, who can decline from processing a transaction if the gas price does not meet their threshold.

Gas Price

Gas price is the amount of Ether paid per unit of Gas spent by a miner for running a transaction:

- This is paid by the sender of the transaction to the miner.
- Gas price is measured in Gwei:

  1 Gwei = 10^9 wei

  1 Ether = 10^{18} wei

Gas Limit

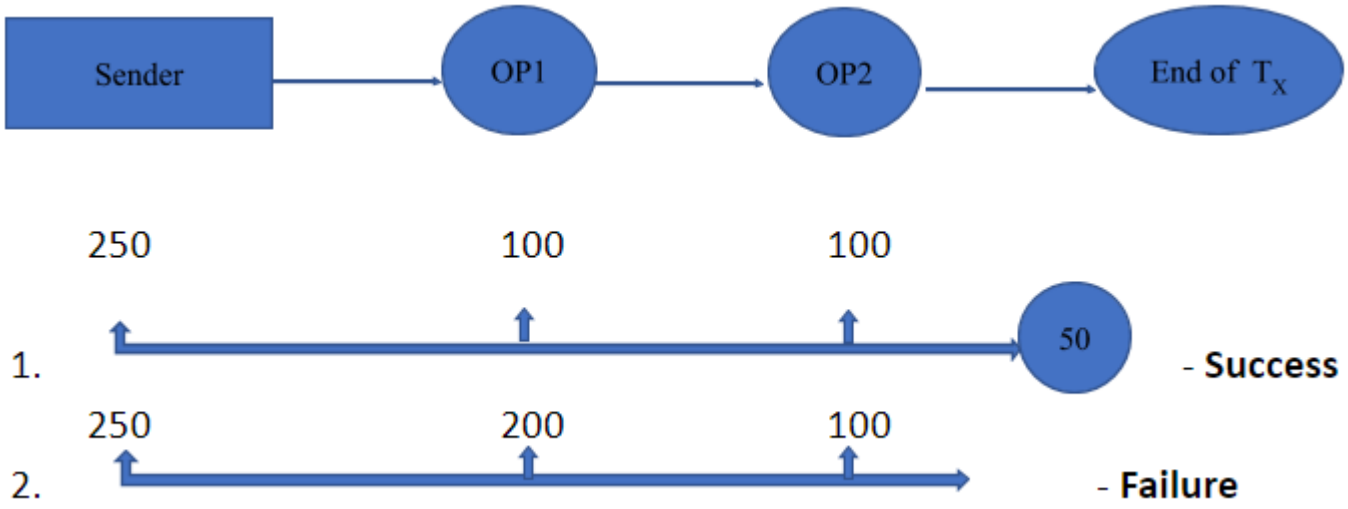
Gas limit is the maximum amount of Gas required to run a transaction.

Transaction limit is the maximum amount that the sender would need to spend in order to perform a transaction successfully. It is calculated as follows:

Transaction Limit = Gas Limit X Gas Price

Gas price and Gas limit are set by the sender of a transaction. The system would deduct this amount from the sender based on the amount of Gas consumed in running the transaction.

Let's try to understand this with two scenarios where the sender has a balance amount of 250.



- In the first scenario, the transaction is successful, since the total gas consumed in operations 1 and 2 is 200, and the remaining 50 is sent back to the sender.

- In the second scenario, the transaction has failed, since the second operation is consuming 100 gas, and the sender has only 50 gas at this point. The transaction will, therefore, not be completed, and the error message '**Transaction failed with out of Gas**' will be displayed. The amount consumed will not be sent back to the sender.

Also, if:

Gas Price = 7,000 wei and
Gas Limit = 150 gas,

Then for a transaction to be mined successfully:

The minimum balance should be 150 * 7,000 = 10,50,000 wei.

***Gas is also used to pay for storage use.**

Ethereum Accounts

The combination of Ethereum address and its private key is referred to as an **account**. Not only that, an account in Ethereum can hold balance (Ether) and can send transactions.

Accounts in Ethereum are stored in a global 'shared state' in the form of key–value pairs.

The list of all these key–value pairs representing accounts defines the state of Ethereum at that point.

Ethereum has two types of accounts: Externally owned accounts (EOAs) and Contract accounts (CAs).

1. Externally Owned Accounts (EOAs): These are combinations of public addresses and private keys, and there is no code associated with them. You can use these accounts to:

- Send and receive Ether to/from another account, and
- Send transactions to smart contracts.

2. Contract Accounts (CAs): These accounts do not have a corresponding private key. These accounts are generated when you deploy your contract on blockchain. You will see them referred to as just contracts in many places (instead of contract accounts). Some key features of contract (accounts) are as follows:

- They can send and receive Ether just like EOAs.
- Unlike EOAs, they have code associated with them.
- Transactions have to be triggered by an EOA or another contract.

In a **key–value** pair, the key is a 20-byte string, which is usually the public key of the account.

Value, on the other hand, represents the state of the account.

Value comprises of the four values described below:

- **Nonce:** For EOAs, nonce is the number of transactions that are sent from an account's address. **For CAs**, nonce is the number of contracts created by the account.
- **Balance:** It is calculated as Number of wei/Ether owned by an account.
- **Storage Root (Storage Hash):** This is a hash of the root node of a Merkle tree that encodes the hash of the storage content of an account.
- **Code Hash:** This is the hash of the EVM (Ethereum Virtual Machine) code of this account:

  For EOAs – Hash of an empty string

  For CAs – Code that gets hashed and stored as the code hash

Structure of Ethereum Accounts

Interaction Between Ethereum Accounts

The following points summarise the transactions in Ethereum:

- An EOA can send messages to other EOAs or to other CAs by creating and signing a transaction using its private key.
- A message between two EOAs is simply a value transfer.
- But a message from an EOA to a CA activates the CA's code, allowing it to perform various actions (e.g., transfer tokens, write to internal storage, mint new tokens, perform some calculation, create new contracts, etc.).
- Unlike EOAs, CAs cannot initiate new transactions on their own.
- Instead, CAs can only fire transactions in response to other transactions that they have received (from an EOA or from another CA).
- An action that occurs on the Ethereum blockchain is always set in motion by transactions that are fired from EOAs.

Note: Transactions can be triggered from an EOA only.

In Ethereum, the values of all the accounts are maintained independently by each of the nodes inside the network. And they are all maintained logically as a single shared state:

- All of the accounts capture the snapshot of the blockchain at any particular point in time and that goes to every block as an entity.
- Whenever a new block is created, a state root is stored in the header of that block.
  State root is the Merkle root of all the accounts at that moment. Simply put, all the key–value pairs that represent an account together, form a Merkle tree is known as state root.
- This state root is captured by a block at the time of its creation.
- Any change in the data would lead to the calculation of the Merkle tree all over again to match the state root in the block. This is highly impossible, and hence the immutability is maintained in the network. Along with state root, the transaction root and the receipt root are also used to capture the state of the network in every block.
- By calculating and storing the roots mentioned above, Ethereum captures the network state every time a new block is created (**state root**, **transaction root** and **receipt root**).

Transaction in Ethereum

The term ‘transaction’ is used in Ethereum to refer to the signed data package that stores a message that is to be sent from an EOA to another account on the blockchain.

Components of a Transaction

The components of a transaction include the following:

- **nonce:** A count of the number of transactions sent by the sender
- **Gas Price:** The amount of Wei that the sender is willing to pay per unit of Gas required to execute the transaction
- **Gas Limit:** The maximum amount of gas that the sender is willing to pay to execute a transaction
- **to:** The address of the recipient. In a contract-creating transaction, an empty value is used
- **value:** The amount of Wei to be transferred from the sender to the recipient. In a contract-creating transaction, this value serves as the starting balance within the newly created contract account
- **v, r, s:** Used to generate the signature that identifies the sender of the transaction
- **init:** An EVM code fragment that is used to initialise the new contract account
- **data:** The input data (i.e., parameters) of the message call. Each type of transaction has all the components above

Types of Transactions

Transactions are of the following different types:

- **Message Calls:** These are internal transactions between an EOA and a CA or between one CA and another. When one CA sends a message to another CA, the associated code that exists on the recipient CA is executed.
- **Value Transfer:** These are transfer of value from one EOA to another.
- **Contract Creation Calls:** They are initiated by EOAs, and the recipient's address is kept empty. Such transactions create a new CA and, hence, are used to create and install new Ethereum contracts.

Block Anatomy

Blocks are batches of transactions with a hash of a previous block in the chain.

A block consists of:

- The block header;
- The set of transactions included in that block; and
- A set of other block headers for the Ommers of the current block.

A block header in Ethereum is quite similar to a block in Bitcoin, and apart from components like parent hash, nonce, difficulty, mix hash and timestamp, it also contains the Gas used to mine a block and the Gas limit that is defined for that block.

Every block header also contains three important Trie structures for:

- **state (state root):** The Merkle root of the current state of all the accounts
- **transactions (transaction root):** The Merkle root of all the transactions mined in a block
- **receipts (receipt root):** The Merkle root of the receipts for all the transactions in a block. A receipt is issued by the network/system whenever a transaction is committed; hence, the receipts for all the transactions in a block would form a Merkle tree, and its root is called the receipt root.

Etherscan is a blockchain explorer on which you can view the different components of a block and a transaction, which we covered in the previous segment.

Session 2 – Consensus and Forking

In this session, you learnt about:

- The Byzantine Generals Problem – What drove a blockchain network to create a consensus?
- Consensus in Ethereum – How is Ethereum's PoW different from Bitcoin?
- Forking and ommer blocks
- Hard and soft forks

Let us summarise all the topics that we have covered in this session.

Byzantine Generals Problem

The Byzantine Generals Problem is a universal problem for any networking system. It suggests that a group of soldiers communicating with each other to decide their next move can be brought down if some of the soldiers are traitors and communicate incorrect information.

In a distributed network, we can say that the Byzantine Generals Problem states that:

If there are 3f+1 nodes in a distributed system, where ‘f’ can be any positive integer, then the maximum number of faulty nodes should not be more than ‘f’.

Let us understand this with the example below.

Let us consider that the number of nodes in the system (3f+1) is 10.

According to the formula, the maximum number of faulty nodes in the system can be 3:
$3f+1 = 10$
$f = (10-1)/3$
$f = 3$

Therefore, a distributed system with 10 nodes having fewer than 3 faulty nodes will be Byzantine fault tolerant.

Any distributed system with more than three faulty nodes will be an intolerant system, and such systems are vulnerable to attacks from faulty nodes.

Any network that is not affected by attacks from faulty nodes is said to be Byzantine fault tolerant. The extent to which a network can survive attacks from faulty nodes is said to be its tolerance percentage.

PoW is 51% tolerant as mentioned in Bitcoin.

Consensus in Ethereum

Ethereum is currently operating on a Proof-of-Work (PoW) consensus protocol; but in the future, it will be shifting to a Proof-of-Stake (PoS) protocol.

The current Ethereum blockchain uses a consensus algorithm called **Ethash**, which is built specifically for the Ethereum blockchain. The Ethash PoW algorithm introduces the property of 'memory hardness' to the Ethereum blockchain.

Memory hardness states that your computational performance is limited by how fast your machine can move data around in memory, as opposed to how rapidly it can perform computations. By doing this, the Ethereum blockchain aims to prevent large organisations and large mining pools from getting undue control over the network.

Ethereum made its mining process highly I/O intensive. This is done with the help of a DAG (directed acyclic graph) – a very large file that is passed along with a block as an input to the hashing algorithm, Ethash.

The DAG requires sufficient memory size, and so, the entire hashing process is a CPU-based rather than a GPU-based process. Therefore, any system with memory large enough to hold a DAG can now stand a chance to mine the block successfully. And this is why Ethereum has a 15-second block creation rate, as compared with 10 minutes for Bitcoin.

Forks

Forks are related to the fact that different parties need to use common rules to maintain the history of a blockchain. When parties are not in agreement, alternative chains may emerge. While most forks are short-lived, some are permanent. Short-lived forks stem from the difficulty of reaching fast consensus in a distributed system. On the other hand, permanent forks (in the sense of protocol changes) are used to add new features to a blockchain, and to reverse the effects of hacking or catastrophic bugs on a blockchain, as was the case with the Bitcoin fork on 6 August 2010, or the fork between **Ethereum** and **Ethereum Classic**.

Forks can be classified as accidental or intentional. An accidental fork is created when two or more miners find a block at nearly the same time. The fork is resolved when subsequent block(s) are added and one of the chains becomes longer than the alternative chain(s). The network abandons the blocks that are not in the longest chain (they are called orphaned blocks).

Soft Fork

A soft fork gets created as a result of multiple blocks being created at the same time. However, as the chain moves forward and a dominant chain is chosen, the other chains become irrelevant.

If a soft fork happens on the blockchain, the older version of the chain has to eventually merge into the newer version.

Hard Fork

A hard fork is a permanent split of a blockchain, and it is backward-incompatible. It divides a blockchain into two separate chains, both of which are dominant.

A hard fork is generally a change in protocol that renders the older version invalid.

One of the examples of a hard fork is the two separate chains in Ethereum:

- Ethereum and
- Ethereum Classic.

Ommer Block

Ommer/uncle blocks are created in Ethereum blockchains when two blocks are mined and submitted to the ledger at roughly the same time. Only one can enter the ledger as a block, while the other does not. They are similar to Bitcoin orphans but have an integrated use, unlike their Bitcoin counterparts. The orphan block is called the Ommer block.

An important point that differentiates Ethereum from Bitcoin is the Ommer rewards. Unlike Bitcoin, Ethereum also rewards people for creating Ommer blocks. However, the rewards for creating Ommer blocks are much less as compared with the rewards for creating the main block.

The reason to have an Ommer block in a blockchain is to give the miner who lost the opportunity a chance.

Session 3 – Transaction Flow and Challenges In Ethereum

In this session, you learnt the following topics:

- Logs and receipts – All the transaction events in Ethereum are stored in the form of logs and receipts
- State transition in Ethereum – These are the different checks that a transaction goes through and all the steps of the transaction flow
- Challenges in Ethereum
- Solutions to the listed challenges

We will take a detailed look at each of these topics in the upcoming segments.

Logs and Receipts

Ethereum maintains logs to track various transactions and messages. A contract can explicitly generate a log by defining events.

A log entry contains:

- The account address,
- Events and
- Data.

A log stored in the header comes from the log information present in the transaction receipt. Ethereum generates a receipt for every transaction.

These receipts include:

- Block number,
- Block hash,
- Transaction hash,
- Gas used,
- Cumulative gas used in the block and
- Logs of current transaction.

State Transition in Ethereum – Checks and Process

As you know, Ethereum is a State Transition Machine, which is a process that takes the system from an existing state (N1) to a new state (N2) after a transaction is given as an input to the network..

Ethereum keeps moving between these states as new transactions keep occurring on the network.

In this segment, you learnt how transactions take place inside Ethereum and what are the checks and what is the process that is involved.

Whenever a transaction is triggered by an EOA, that transaction has to go through certain checks and a specific process.

Checks

A transaction has to go through the following checks before being executed:

- It must be formatted properly.
- It should have a valid transaction signature.
- It should have a valid transaction nonce. Recall that the nonce of an account is the count of transactions that are sent from that account. To be valid, a transaction nonce must be equal to the nonce of the sender account.
- The gas limit of a transaction must be equal to or greater than the intrinsic gas used by the transaction. The intrinsic gas includes:
    - A predefined cost of 21,000 gas for executing the transaction;
    - A gas fee for the data sent with the transaction (4 gas for every byte of data or code that equals zero, and 68 gas for every non-zero byte of data or code); and
    - An additional 32,000 gas, which is to be sent if the transaction is a contract-creation transaction.
- The sender's account balance must have enough Ether to cover the 'upfront' gas costs that the sender must pay.
- If the transaction meets all of the above requirements for validity, then we move on to the next step, which is the execution of a transaction.

Process

A transaction on Ethereum is executed in the following manner:

- First, the upfront cost of execution is deducted from the sender's balance.
- Then the nonce of the sender's account is increased by 1 to account for the current transaction.
- Next, the transaction starts executing. Throughout the execution, Ethereum keeps track of the '**substate**', which is a way to record information accrued

during the transaction that would be needed immediately after the transaction is complete. Specifically, it contains:

- A self-destruct set, which is a set of accounts (if any) that would be discarded after the transaction completes;
- Log series, which are archived and indexable checkpoints of the virtual machine's code execution; and
- Refund balance, which is the amount to be refunded to the sender's account after the transaction.
- The various computations required by the transaction are processed.
- Once all the steps required by the transaction have been processed, and assuming there is no invalid state, the state is finalised by determining the amount of unused gas to be refunded to the sender.
- The sender is also refunded some allowance from the 'refund balance'.

Altcoins versus Tokens

AltCoins

Altcoins are the base coin in any blockchain network. These are real currency that is used in a blockchain.

The altcoins for Ethereum are 'ether'.

Tokens

Tokens have a specific purpose and can be used for a specific application only. They have a value in Ether but cannot be used outside. They are analogous to a card that we bought in an amusement park and which can be used inside the park only.

Turing Complete

Ethereum is a Turing complete system, like C, C++, Java, JavaScript, Python, etc.

A Turing complete system refers to a system that can compute everything that needs to be computed.

Challenges for Ethereum

Ethereum currently facing the following challenges:

- **Scalability/throughput:** The number of transactions per second is very low (<2,000 Tx/sec).
- **Cost:** The mining process is expensive.
- **Decentralisation:** In Bitcoin, 51% of the network is held by five miner groups. This could happen in Ethereum.
- **Size:** The size of the network is increasing, and nodes have to be highly equipped in order to store a blockchain of this size.
- DApps are not user-friendly yet.
- Competition is increasing, for example, EOS, NEO and Cardano.

Solutions

In a PoS consensus algorithm, users who desire to validate blocks are required to deposit a stake of their own Ether (right now, that stake is estimated to be 32 Ether). That stake is locked, and a consensus algorithm is used after staked users can participate in.

- Anyone who holds Ether can become a validator.
- Validators place a bet on a block, and if the block gets selected as the main block, then they get rewarded.
- Validators would be penalised if they place a bet on more than one block.
- Ethereum is building its own PoS, which is called the CASPER protocol.

PoS algorithms are two types: a chain-based proof and a Byzantine fault tolerance-style proof:

- A chain-based Proof-of-Stake where staked users are selected pseudo-randomly during a set time block. That user is given the right to create a single block. The block that is created must be pointed to a previous block.

- A Byzantine fault tolerance-style (BFT-style) proof randomly gives users the right to propose blocks. However, agreement on a canonical block is obtained in a multiple-round process, where each validator party to the network places a vote for a specific block during a round, and in the final round, the validators would have arrived at a permanent block to add to the chain.

There are many other smaller solutions to the other challenges mentioned in the previous segment:

- To increase the throughput of the blockchain network, a solution called off-chaining was proposed by the community. It suggests that not all the transactions are done on the blockchain. A bulk of smaller transactions can be bundled togEther to create one transaction, which can be stored in either a centralised or a decentralised data storage. The net result of all the transactions can then be committed on the blockchain.
- A new node entering the blockchain may not want to download the entire network. Storage of the entire network could be an issue for some nodes. To overcome this problem, sharding was proposed. In sharding, the blockchain network is split into smaller shards. Any node may choose to work on any particular shard without having to download the entire network.

Session 4 – Additional Topics

This session addresses in detail certain topics that could not be covered in the earlier sessions. Some of the topics covered in this session are as follows:

- How Merkle Patricia trie is implemented in Ethereum
- Ethereum block header detailing
- State machine of Ethereum

- Different types of consensus algorithms other than PoW
- Ethash – Ethereum's implementation of PoW
- A few case studies on forking

We will take a detailed look at each of these topics in the upcoming segments.

Merkle Patricia Trie

Bitcoin uses Merkle trees as a data structure to verify its transactions. It can only verify the exact point at which data was tampered with.

Ethereum stores data in a slightly different manner. It uses the properties of both Merkle trees and a radix tree called Patricia trie:

- Ethereum combines the properties of a Merkle tree and a Patricia trie to create a modified Merkle Patricia trie.
- A Merkle tree can be used only to check whEther a value is present in the tree or not.
- A Patricia trie (a radix tree) is a data structure that stores values in key–value pairs. This is optimum in finding common prefixes of the searched value.
- These combined properties of both the trees yields a structure that is more optimum for storing and checking values than Merkle tree individual.

Ethereum uses a database to store its trie structure. Some examples of such databases could include LevelDb and RocksDB. The purpose of storing this data is to go forward and backward, and identify the exact data that was tampered with.

Ethereum – A State Machine

Ethereum, unlike Bitcoin, was developed in such a way that it had to store all the states. States in the network mean that if any transaction occurs on the network, then the entire network undergoes a series of changes. These changes have to be

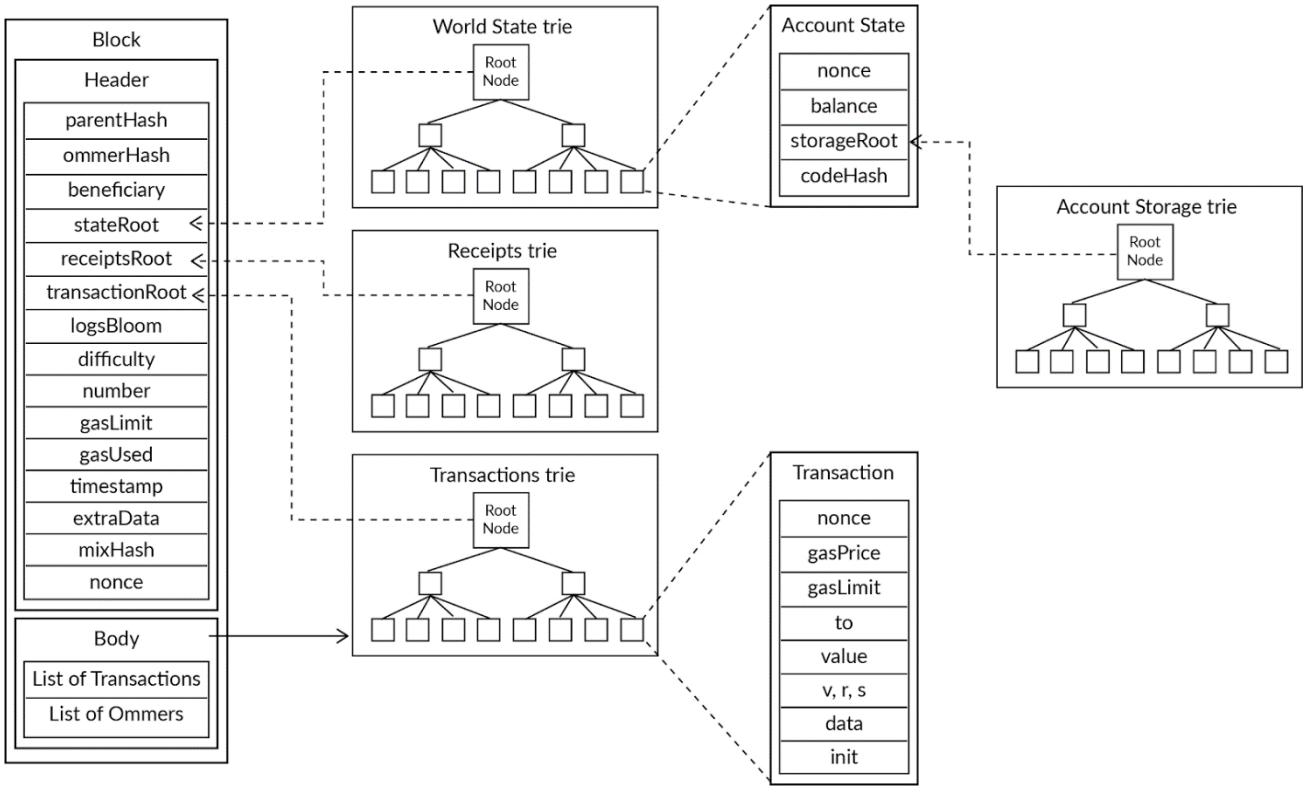
recorded and stored in a structured manner. To achieve this, Ethereum stores states.

The data to be stored in Ethereum can be of the following two types: Permanent data and ephemeral data.

Permanent data can be anything related to a transaction that has to be stored on the blockchain permanently.

Ephemeral data could include account balance, address, etc. This is also stored in the trie data structure.

The diagram below shows the entire state of Ethereum.



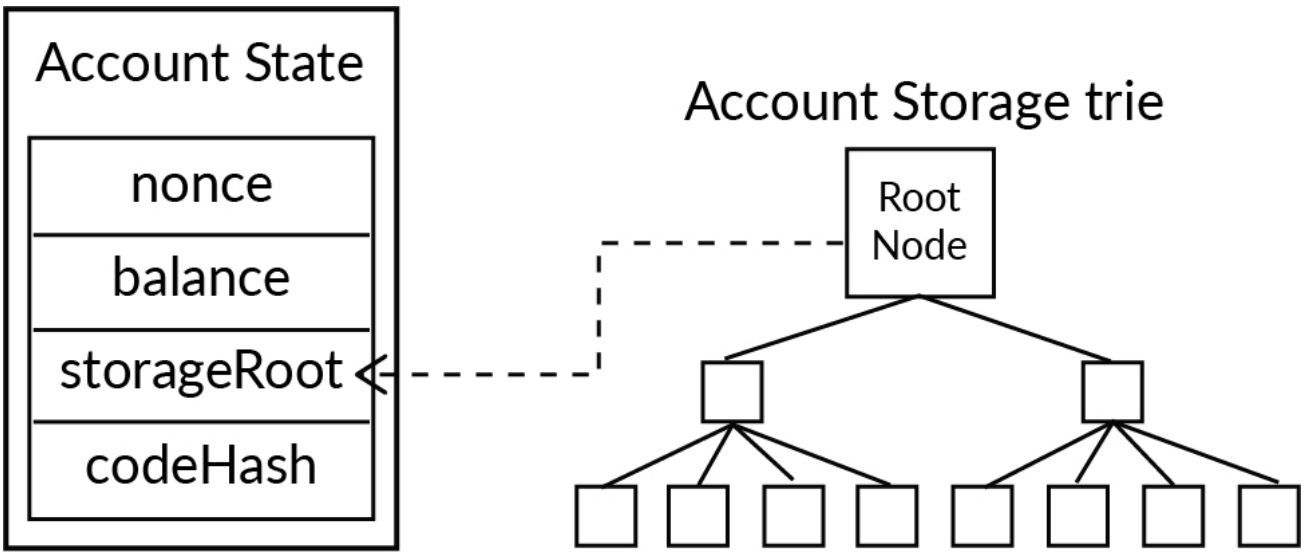
- There are three components of the trie structure has three components, as shown in the diagram:

- State trie tells us the global state of the Ethereum network at any point in it. It is updated as and when transactions take place. It contains a key–value pair, where key is the address of an Ethereum account and value has multiple data, which includes the state root of the network. The state root at any given time represents the entire network as it is a hash of the whole network.

- Transaction trie contains all the transactions of the network. The transaction trie comprises the transaction root, which represents the state of all the transactions at a given point in time.

- The purpose of the Receipt trie is to store all the transaction outcomes.

The diagram below shows the details of an account state.



1. **Nonce:** This is a simple iterator of the account, and it signifies the number of transactions that have been performed in the past. If, for example, account A is performing its fifth transaction, then the nonce would be 5.
2. **Balance:** This is the current balance of the Ethereum account. For example, account A has a balance of 10 Ethers.
3. **Storage root:** It stores the hash of all the contract data present in the account.
4. **codeHash:** Ethereum has two types of accounts, as we saw in the earlier segments: Externally owned accounts (EOAs) and contract accounts (CAs).

For a CA, the codeHash contains the bytecode of the smart contract deployed on the network. For an EOA, this is not applicable.

Types of Consensus Algorithms

In the previous session, you learnt about Proof of Work. Now, there are other consensus algorithms that are being used in other blockchain networks. These algorithms includes the following:

- **Proof of Stake (PoS):** This algorithm states that a person or a node can mine blocks in a network depending upon how much stake they own in the network. For example, in a Bitcoin network, the more Bitcoins that a miner has, the more stake that they have in the network. There are also greater chances for that miner to confirm a new block.
- **Delegate Proof of Stake (DPoS):** DPoS is another PoS mechanism to achieve consensus. It states that each node will bid on a particular transaction, where a bid is counted as a vote. The transaction or the block that gets the higher number of bids wins and is confirmed. Nodes that were bidding in the entire process get back their rewards after the block is committed.
- **Proof of Authority (PoA):** In PoA, your identity is at stake in the network. A particular set of nodes is given the authority to commit blocks to the blockchain. The process of giving authority can be a set of rules defined by the network. Anyone wanting to commit a transaction to the blockchain can do so through the set of authorised nodes.
- **Practical Byzantine Fault Tolerance (PBFT):** In this type of consensus, a leader chosen by the other nodes gets to commit a transaction or a block. The other nodes are called secondary nodes. The leader node is chosen through a random algorithm within a given period of time.

Note that these are not the only consensus algorithms that are used in the industry. There are many other algorithms that are being developed according to

specific requirements. These are just the algorithms that we have discussed in this segment.

Ethash

PoW, as we know, is computation-intensive. Vitalik Buterin proposed the Dagger–Hashimoto algorithm to make Ethereum's PoW I/O-intensive:

1. **Dagger algorithm:** Instead of using a complex computation problem, the Dagger algorithm uses a directed acyclic graph (DAG). What is a DAG? It is a huge data set that is used to store a large amount of data in the most optimised manner. There are a lot of I/O operations that are performed in the Dagger algorithm. Here, the problem was that hashing was still used, which meant difficulty in mining was still an issue.
1. **Hashimoto algorithm:** Instead of only hashing, the Hashimoto algorithm proposes a hash, shift and modulus process. The last step of the process is an Exclusive OR (XOR), the transaction that makes it more memory-intensive. The problem with this algorithm was that it was extremely memory hardened, as the block number needed to be calculated at every single iteration.
1. **Dagger–Hashimoto algorithm:** It combined the best of both the algorithms. It uses a DAG as the data set for I/O operations. The initial version of a DAG was 5 GB. After a while, it was compressed to 1 GB. After every epoch, the DAG is updated. Epoch in Ethereum occurs after the creation of 30,000 blocks.

What is a DAG?

It is a large data set that is sent as an input to Ethash, which is the mining algorithm in Ethereum.

The initial size of a DAG is **1 GB**, and it changes after every epoch (when 30,000 blocks get mined).

Epoch = Cycle of creation of 30,000 blocks

- After every epoch, the size of the DAG increases by 72%.
- The current size of a DAG is about **3 GB**.
- A DAG is calculated from an existing block in the network.

Note: There are many case studies on forking in Ethereum and Bitcoin. You can read about some case studies on Ethereum and Bitcoin through the links given below:

Ethereum https://medium.com/mycrypto/the-history-of-Ethereum-hard-forks-6a6dae76d56f

Bitcoin
https://www.investopedia.com/tech/history-bitcoin-hard-forks/