NATIONAL INSTITUTE OF TRANSPORT

Blockchain Technology Short Training

Introduction to Ethereum

Facilitator: Dr. Cleverence Kombe (PhD)

Introduction to Ethereum | Overview

- Ethereum Blockchain
- The Ethereum networks
- Components of the Ethereum ecosystem

Ethereum | Definition

- Ethereum is a decentralized blockchain platform that enables developers to build DApps and execute smart contracts on a global network of computers.

- Ethereum is designed to be programmable and flexible, allowing developers to create custom tokens, implement complex business logic, and more.

- Ethereum is programmable with a **high-level language** called **Solidity**, allowing developers to create custom logic and complex applications on the blockchain.

Ethereum | Definition

- Ethereum has its own cryptocurrency, called Ether (ETH), which is used as a means of exchange for transactions on the network and as a reward for miners who validate transactions.

- Ethereum uses a consensus algorithm called Proof of Stake (PoS) to secure the network and validate transactions.

- In PoS, validators (also known as "stakers") are chosen to create new blocks and validate transactions based on the amount of ether they hold and are willing to "stake" as collateral.

Ethereum | Definition

- Ethereum has a unique feature called gas, which is a unit of measurement for the computational work that is needed to execute a specific operation on the Ethereum network.

- Each operation has a predefined gas cost, which reflects the complexity and resource requirements of that operation.

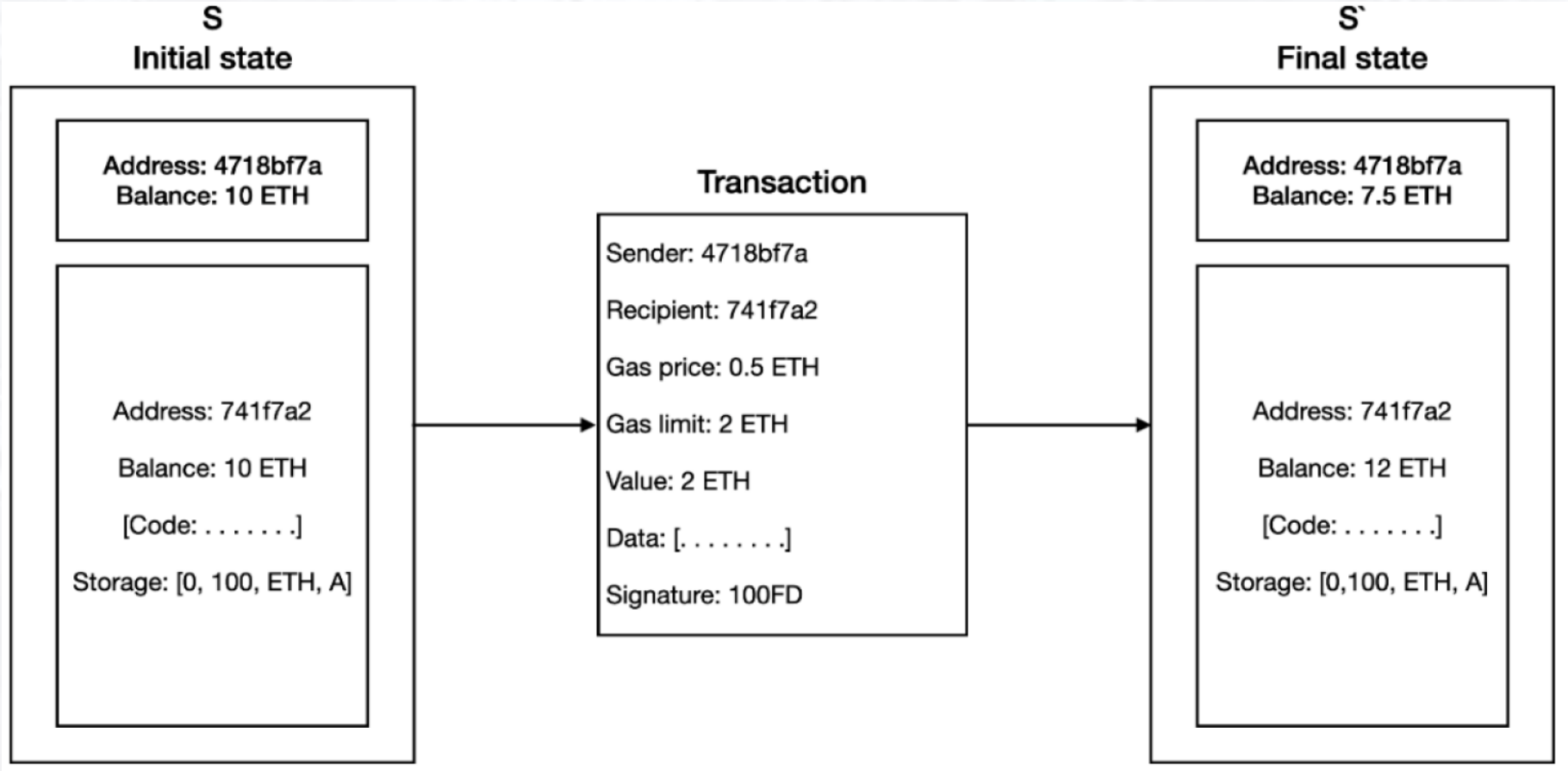
- Gas fees are paid by users to incentivize validators to process their transactions and execute their smart contracts.

Ethereum | Definition

- Ethereum is a **transaction-based state machine**.
- The process of executing transactions on the Ethereum blockchain is incremental, gradually transforming the **genesis state** into a **final state**.
- Transactions on Ethereum can trigger smart contracts, which are programmed in Solidity and automatically execute when certain conditions are met.
- The incremental execution of transactions on Ethereum enables the creation of complex decentralized applications that can automate business processes and facilitate secure peer-to-peer transactions.

Ethereum | Definition

- The final transformation is then accepted as the absolute undisputed version of the state.



Ethereum Networks | Three Types of Networks

The Ethereum network is a peer-to-peer network where nodes participate in order to maintain the blockchain and contribute to the consensus mechanism.

- **The mainnet**
  - The mainnet is the current live network of Ethereum.
  - Its network ID is 1 and its chain ID is also 1. The network and chain IDs are used to identify the network.
  - A block explorer that shows detailed information about blocks and other relevant metrics is available at https://etherscan.io. This can be used to explore the Ethereum blockchain.

Ethereum Networks | Three Types of Networks

The Ethereum network is a peer-to-peer network where nodes participate in order to maintain the blockchain and contribute to the consensus mechanism.

- **Testnets**
  - There is a number of testnets available for Ethereum testing. The aim of these test blockchains is to provide a testing environment for smart contracts and DApps before being deployed to the production live blockchain.
  - Being test networks, they also allow experimentation and research.
  - The main testnet is called Ropsten, which contains all the features of other smaller and special-purpose testnets that were created for specific releases. Other testnets include Kovan, Rinkeby, Goerli, and Sepolia.

Ethereum Networks | Three Types of Networks

The Ethereum network is a peer-to-peer network where nodes participate in order to maintain the blockchain and contribute to the consensus mechanism.

- **Private nets**
  - As the name suggests, these are private networks that can be created by generating a new genesis block. This is usually the case in private blockchain networks, where a private group of entities start their blockchain network and use it as a permissioned or consortium blockchain.
  - Network IDs and chain IDs are used by Ethereum clients to identify the network. Chain ID was introduced in EIP155 as part of replay protection mechanism. EIP155 is detailed at https://github.com/ethereum/EIPs/blob/master/EIPS/eip-155.md

Ethereum | Components of the Ethereum ecosystem

The Ethereum blockchain stack consists of various components:

- **Ethereum Blockchain**
  - At the core, there is the Ethereum blockchain running on the peer-to-peer Ethereum network.
- **Ethereum client**
  - There's an Ethereum client (usually Geth) that runs on the nodes and connects to the peer-to-peer Ethereum network from where blockchain is downloaded and stored locally.
  - The local copy of the blockchain is synchronized regularly with the network.

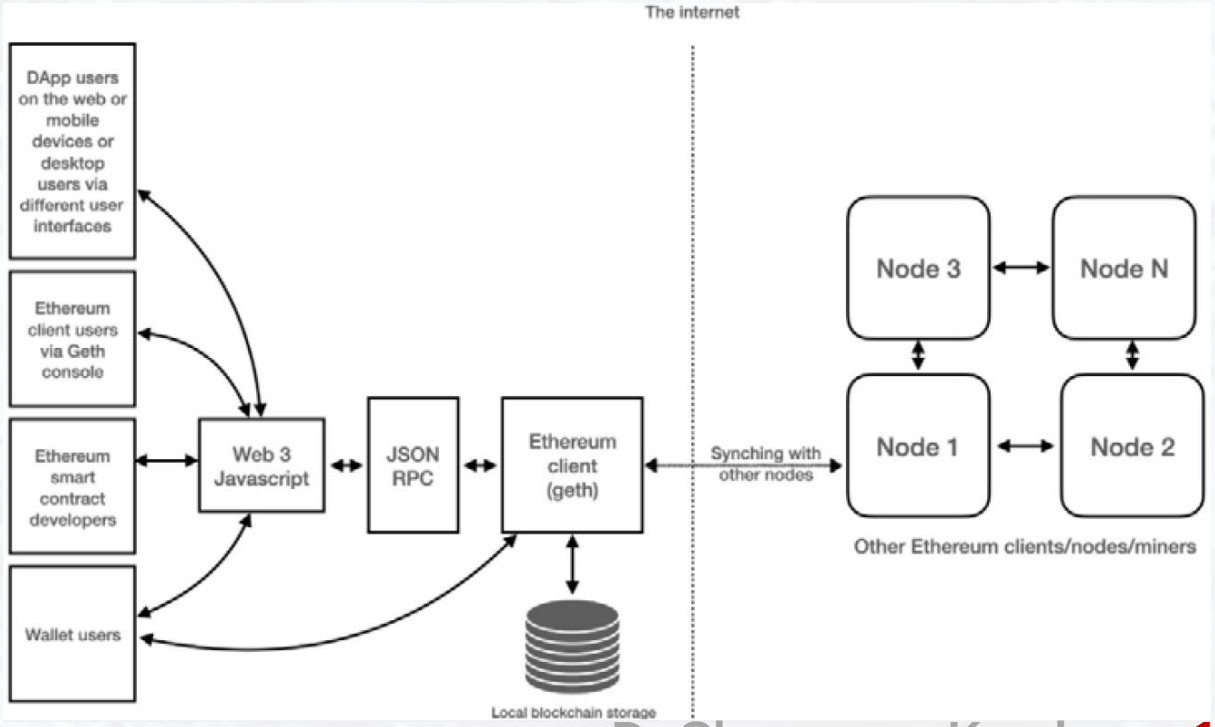
Ethereum | Components of the Ethereum ecosystem

The Ethereum blockchain stack consists of various components:

►web3.js library

- Another component is the web3.js library that allows interaction with the geth client via the Remote Procedure Call (RPC) interface.

- Ethereum high-level ecosystem:



Ethereum Blockchain | List of Elements

- Keys and addresses
- Accounts
- Transactions and messages
- The Ethereum Virtual Machine (EVM)
- Smart contracts and native contracts

Ethereum Blockchain Elements | Keys and Addresses

Ethereum Blockchain Elements | Keys and Addresses

- **Keys and Addresses** are used in the Ethereum blockchain to represent **ownership** and **transfer ether**
- The **keys** used are made up of **pairs of private** and **public parts**
- The **private key** is generated **randomly** and is **kept secret**, whereas a **public key** is derived from the **private key**
- **Addresses** are derived from **public keys** and are **20-byte codes** used to **identify accounts**

Ethereum Blockchain Elements | Keys and Addresses

The process of **key generation** and **address** derivation is as follows:

1. First, a **private key** is randomly chosen (a 256-bit positive integer) under the rules defined by the **elliptic curve secp256k1** specification (in the range [1, secp256k1n − 1]).

2. The public key is then derived from this private key using the Elliptic Curve Digital Signature Algorithm (ECDSA) recovery function.

3. An **address** is derived from the **public key**, specifically, from the rightmost 160 bits of the Keccak hash of the public key.

Ethereum Blockchain Elements | Keys and Addresses

An example of how keys and addresses look in Ethereum is shown as follows:

- A private key:

b51928c22782e97cca95c490eb958b06fab7a70b9512c38c36974f47b954ffc4

- A public key:

3aa5b8eefd12bdc2d26f1ae348e5f383480877bda6f9e1a47f6a4afb35cf998ab
847f1e3948b1173622dafc6b4ac198c97b18fe1d79f90c9093ab2ff9ad99260

- An address:

0x77b4b5699827c5c49f73bd16fd5ce3d828c36f32

Ethereum Blockchain Elements | Accounts

Ethereum Blockchain Elements | Accounts

- In the Ethereum blockchain, an **account** is a fundamental component that enables users to interact with the network. It can represent either **a human user** or a **smart contract** deployed on the blockchain.
- Each **account** is defined by a **pair of private** and **public keys**, which are generated using specialized software called a **key pair generator**.
- The **private key** is used to **digitally sign transactions** and **prove ownership of the account**, while the **public key** is used to **receive incoming transactions** and **verify signatures**.

Ethereum Blockchain Elements | Accounts

- **Transactions** are the means by which **users interact** with the Ethereum blockchain. Before submitting a transaction to the network via a node, it must first be **signed with the private key** associated with the **sending account.**
- This **digital signature** is mathematical proof that the transaction was **authorized by the account owner** and ensures that the **transaction cannot be altered** or **tampered** with during transmission.
- Once the transaction is submitted, it is processed by the Ethereum network according to a set of consensus rules that govern how the blockchain operates.

Ethereum Blockchain Elements | Accounts

- Ethereum, being **a transaction-driven state machine**, the state is created or updated as a result of the **interaction between accounts** and **transaction executions**.
- **All accounts** have a **state that**, when combined together, represents the **state of the Ethereum network**. With **every new block**, the state of the Ethereum network **is updated**.
- Operations performed **between** and **on the accounts** represent **state transitions**.

Ethereum Blockchain Elements | Accounts

The **state transition** is achieved using what's called the **Ethereum state transition function**, which works as follows:

1. Confirm the transaction **validity** by **checking the syntax**, **signature validity**, and nonce.

2. The transaction fee is calculated, and the sending address is resolved using **the signature**.

   Furthermore, the sender's account balance is checked and subtracted accordingly, and the nonce is incremented.

   An error is returned if the account balance is insufficient.

The **state transition** is achieved using what's called the **Ethereum state transition function**, which works as follows:

3. Provide enough ETH (the gas price) to cover the cost of the transaction. This is **charged per byte** and is **incrementally proportional to the size of the transaction**.

   In this step, the actual transfer of value occurs. The flow is from the sender's account to the receiver's account.

   The account is created automatically if the destination account specified in the transaction does not exist yet. Moreover, if the destination account is a contract, then the contract code is executed.

   This also depends on the amount of gas available. If enough gas is available, then the contract code will be executed fully; otherwise, it will run up to the point where it runs out of gas.

The **state transition** is achieved using what's called the **Ethereum state transition function**, which works as follows:

4. In cases of transaction failure due to insufficient account balance or gas, **all state changes are rolled back** except for the fee payment, which is paid to the miners.

5. Finally, the remainder (if any) of the fee is sent back to the sender as change and the fee is paid to the miners accordingly. At this point, the **function returns the resulting state**, which is also stored on the blockchain.

Ethereum Blockchain Elements | Accounts

Externally Owned Accounts (EOAs) and Contract Accounts (CAs)

- There are two types of accounts in Ethereum: **Externally Owned Accounts (EOAs)** and **Contract Accounts (CAs)**.
- **EOAs** are accounts that are controlled by an individual who holds the **private key** associated with the account. The private key allows the owner to send and receive ether, interact with smart contracts, and participate in the consensus process.
- **CAs** are accounts that have **code associated with them**, in addition to a private key. The code enables the contract to perform certain functions when certain conditions are met. These functions may include sending ether to other accounts, interacting with other smart contracts, and executing complex computations.

Ethereum Blockchain Elements | Accounts

Externally Owned Accounts (EOAs) and Contract Accounts (CAs)

The properties of **EOAs** are:

1. They have a state.
2. They are associated with a human user, hence are also called user accounts.
3. EOAs have an ether balance.
4. They are capable of sending transactions.
5. They have no associated code.
6. They are controlled by private keys.
7. EOAs cannot initiate a call message.
8. Accounts contain a key-value store.
9. EOAs can initiate transaction messages.

Ethereum Blockchain Elements | Accounts

Externally Owned Accounts (EOAs) and Contract Accounts (CAs)

The properties of **CAs** are:

1. They have a state.
2. They are not intrinsically associated with any user or actor on the blockchain.
3. CAs have an ether balance.
4. They have associated code that is kept in memory/storage on the blockchain. They have access to storage.
5. They can get triggered and execute code in response to a transaction or a message from other contracts. It is worth noting that due to the Turing-completeness property of the Ethereum blockchain, the code within CAs can be of any level of complexity.

The code is executed by the EVM by each mining node on the Ethereum network.

Ethereum Blockchain Elements | Accounts

Externally Owned Accounts (EOAs) and Contract Accounts (CAs)

The properties of **CAs** are:

6. CAs can maintain their permanent states and can call other contracts. It is envisaged that in the Serenity (Ethereum 2.0) release, the distinction between EOAs and CAs may be eliminated.
7. CAs cannot start transaction messages.
8. CAs can initiate a call message.
9. CAs contain a key-value store.
10. CAs' addresses are generated when they are deployed. This address of the contract is used to identify its location on the blockchain.

NB: Accounts allow interaction with the blockchain via transactions.

Ethereum Blockchain Elements | Transactions and messages

Ethereum Blockchain Elements | Transactions and messages

- **A transaction** in Ethereum consists of **digitally signed data** that contains instructions to either **create a new contract** or **execute a message call**.
- Transactions are signed using a private key, and the resulting output depends on the type of transaction.
- There are two types of transactions in Ethereum:
  - **Message call transactions**, which allow passing messages between existing contract accounts.
  - **Contract creation transactions**, which result in the creation of a new contract account. When executed successfully, these transactions create an account with associated code.

Ethereum Blockchain Elements | Transactions and messages

Message call and Contract creation transactions are composed of some standard fields, which are described as follows:

- **Nonce**
  - The nonce is a number that is incremented by one every time a transaction is sent by the sender. It must be equal to the number of transactions sent and is used as a unique identifier for the transaction. A nonce value can only be used once. This is used for replay protection on the network.
- **Gas price**
  - The gas price field represents the amount of Wei required to execute the transaction. In other words, this is the amount of Wei you are willing to pay for this transaction. This is charged per unit of gas for all computation costs incurred as a result of the execution of this transaction.

Ethereum Blockchain Elements | Transactions and messages

Message call and Contract creation transactions are composed of some standard fields, which are described as follows:

- **Gas limit**
  - The gas limit field contains the value that represents the maximum amount of gas that can be consumed to execute the transaction. This is the fee amount, in ether, that a user (for example, the sender of the transaction) is willing to pay for computation.
- **To**
  - The To field is a value that represents the address of the recipient of the transaction. This is a 20 byte value.
- **Value**
  - Value represents the total number of Wei to be transferred to the recipient; in the case of a CA, this represents the balance that the contract will hold.

Message call and Contract creation transactions are composed of some standard fields, which are described as follows:

- **Signature**
  - The signature consists of three fields: V, R, and S.
  - V represents the recovery ID and is used to derive the public key from the private key. The values R and S form the digital signature and are calculated using the ECDSA scheme and the secp256k1 curve.
  - R is a 32-byte sequence derived from a calculated point on the curve, while S is also a 32-byte sequence calculated by multiplying R with the private key and adding it into the hash of the message to be signed.

Ethereum Blockchain Elements | Transactions and messages

Message call and Contract creation transactions are composed of some standard fields, which are described as follows:

- **Signature**
  - To sign a transaction, the ECDSASIGN function is used, taking the message and private key as inputs to produce V, R, and S. These values can be used to recover the public key and determine the sender of the transaction.

ECDSASIGN (Message, Private Key) = (V, R, S)

Message call and Contract creation transactions are composed of some standard fields, which are described as follows:

- **Init**
  - The **Init** field is used only in transactions that are intended to create contracts, that is, contract creation transactions. This represents a byte array of unlimited length that specifies the EVM code to be used in the account initialization process.
  - The code contained in this field is executed only once when the account is created for the first time, it (init) gets destroyed immediately after that.
  - Init also returns another code section called the body, which persists and runs in response to message calls that the CA may receive. These message calls may be sent via a transaction or an internal code execution).

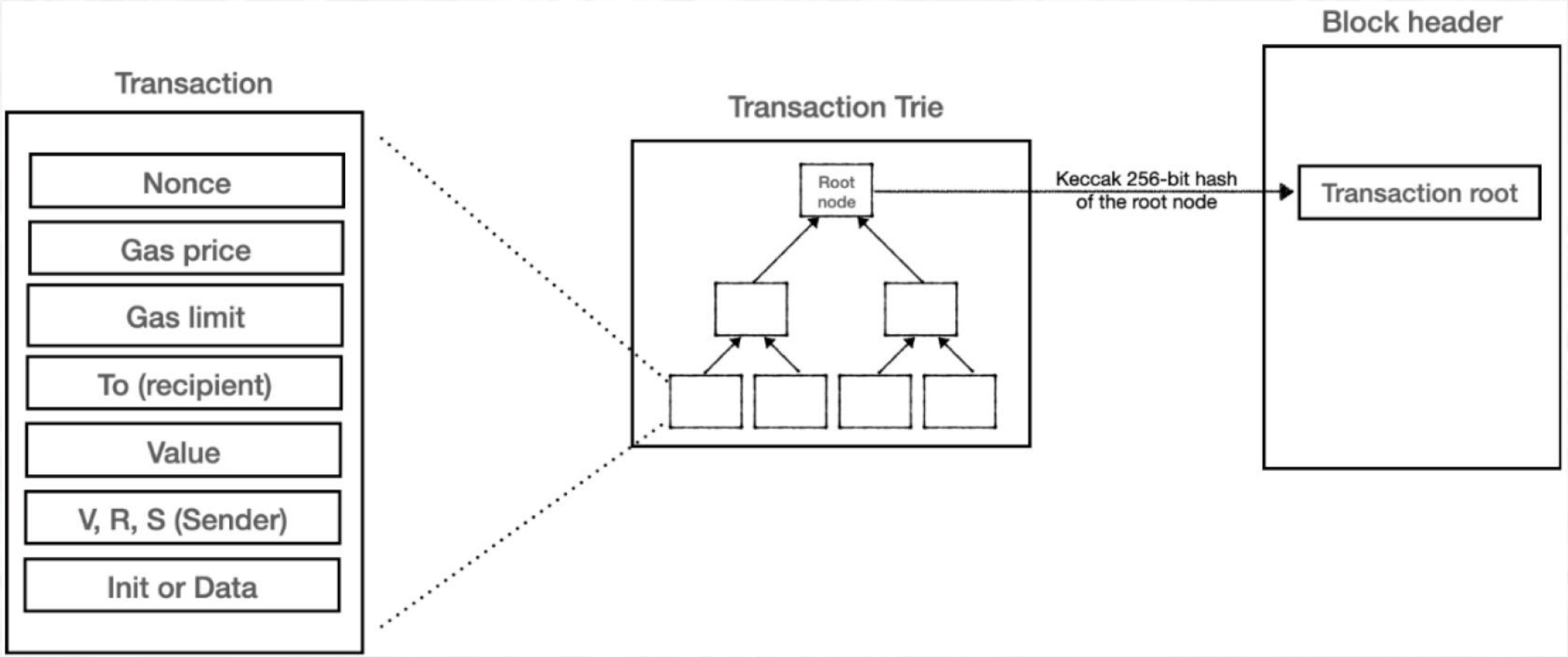
Ethereum Blockchain Elements | Transactions and messages

Message call and Contract creation transactions are composed of some standard fields, which are described as follows:

- **Data**
  - If the transaction is a message call, then the **Data field** is used instead of **init**, and represents the input data of the message call. It is also unlimited in size and is organized as a byte array.

Ethereum Blockchain Elements | Transactions and messages

Message call and Contract creation transactions are composed of some standard fields, which are described as follows:



Ethereum Blockchain Elements | Transactions and messages

- A block is a data structure that contains batches of transactions in the Ethereum blockchain.
- Transactions can be found either in transaction pools or blocks. In transaction pools, they wait for verification by a node, and in blocks, they are added after successful verification.
- When a mining node starts to verify blocks, it starts with the highest-paying transactions in the transaction pool and executes them one by one.
- Once the gas limit is reached, or no more transactions are left to be processed in the transaction pool, the mining process starts.

Ethereum Blockchain Elements | Transactions and messages

- In the mining process, the block is repeatedly hashed until a valid nonce is found. The nonce, once hashed with the block, results in a value less than the difficulty target.
- Mining requires a lot of computational power and energy, as it involves solving complex mathematical problems.
- Once the block is successfully mined, it will be broadcasted immediately to the network, claiming success, and will be verified and accepted by the network.
- Miners receive a reward in the form of Ether for successfully mining a block, which incentivizes them to keep the network secure and decentralized.

Recursive Length Prefix - RLP

- For the purpose of storage and transmission, Ethereum data is encoded with a new encoding scheme developed by Ethereum developers.
- This scheme is called **Recursive Length Prefix (RLP).**
- To define **RLP**, we first need to understand the concept of serialization**.**
- **Serialization** is simply a mechanism commonly used in computing to encode data structures into a format (a sequence of bytes) that is suitable for storage and/or transmission over the communication links in a network.
- Once a receiver receives the serialized data, it de-serializes it to obtain the original data structure

Recursive Length Prefix - RLP

- Serialization and deserialization are also referred as marshaling and un-marshaling, respectively. Some commonly used serialization formats include XML, JSON, YAML, protocol buffers, and XDR.
- There are two types of serialization formats, namely **text** and **binary.**
- In a blockchain, there is a need to serialize and deserialize different types of data such as blocks, accounts, and transactions to support transmission over the network and storage on clients**.**
- RLP is used also to save the state in a Patricia tree on storage media.

Recursive Length Prefix - RLP

- Why do we need a new encoding scheme when there are so many different serialization formats already available?.
  - The answer to this question is that RLP is a **deterministic scheme**, whereas other schemes may produce different results for the same input, which is absolutely unacceptable on a blockchain.
  - Even a small change will lead to a totally different hash and will result in data integrity problems that will render the entire blockchain useless.
- RLP is a minimalistic and simple-to-implement serialization format that does not define any data types and simply stores structures as nested arrays.

Contract creation transactions

- A contract creation transaction is used to create smart contracts on the blockchain. There are a few essential parameters required for a contract creation transaction:
  - The sender.
  - The transaction originator.
  - Available gas.
  - Gas price.
  - Endowment, which is the amount of ether allocated.
  - A byte array of an arbitrary length.

Contract creation transactions

- A contract creation transaction is used to create smart contracts on the blockchain. There are a few essential parameters required for a contract creation transaction:
  - Initialization EVM code.
  - The current depth of the message call/contract-creation stack (current depth means the number of items that are already present in the stack).
- Addresses generated as a result of a contract creation transaction are 160 bits in length. They are the rightmost 160 bits of the Keccak hash of the RLP encoding of the structure containing only the sender and the nonce.

Contract creation transactions

- Initially, the nonce in the account is set to zero. The balance of the account is set to the value passed to the contract. The storage is also set to empty. The code hash is a Keccak 256-bit hash of the empty string.

- The new account is initialized when the EVM code (the initialization EVM code) is executed. In the case of any exception during code execution, such as not having enough gas (running Out of Gas, or OOG), the state does not change.

- If the execution is successful, then the account is created after the payment of appropriate gas costs. The result of a contract creation transaction is either a new contract with its balance, or no new contract is created with no transfer of value.

Message call transactions

- A message call requires several parameters for execution, which are listed as follows:
  - The sender.
  - The transaction originator.
  - The recipient.
  - The account whose code is to be executed (usually the same as the recipient).
  - Available gas.
  - The value.

Message call transactions

- A message call requires several parameters for execution, which are listed as follows:
  - The gas price.
  - An arbitrary-length byte array.
  - The input data of the call.
  - The current depth of the message call/contract creation stack.
- Message calls result in a state transition. Message calls also produce output data, which is not used if transactions are executed. In cases where message calls are triggered by EVM code, the output produced by the transaction execution is used.

Message call transactions

- A message call is the act of passing a message from one account to another.
- If the destination account has an associated EVM code, then the EVM will start upon the receipt of the message to perform the required operations. If the message sender is an autonomous object (external actor), then the call passes back any data returned from the EVM operation.
- The state is altered by transactions. These transactions are created by external factors (users) and are signed and then broadcasted to the Ethereum network.
- **Messages are passed between accounts using message calls**.

Messages

- Messages are the data and values that are passed between two accounts.
- A message is a data packet passed between two accounts. This data packet contains data and a value (the amount of ether). It can either be sent via a smart contract (autonomous object) or from an external actor (an Externally Owned Account, or EOA) in the form of a transaction that has been digitally signed by the sender.
- Contracts can send messages to other contracts.
- Messages only exist in the execution environment and are never stored.

Messages

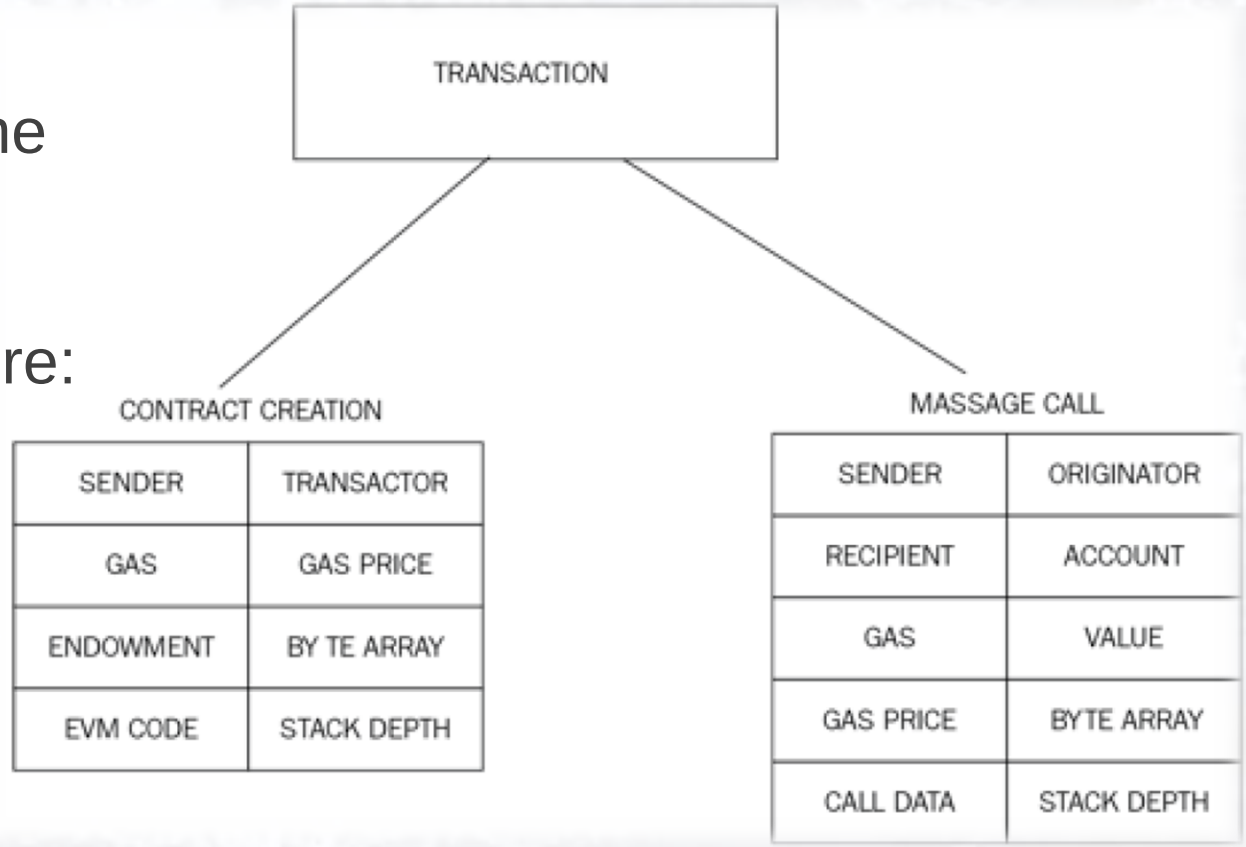
- Messages are similar to transactions; however, the main difference is that they are produced by the contracts, whereas transactions are produced by entities external to the Ethereum environment (EOAs).
- A message consists of the following components:
  - The sender of the message
  - The recipient of the message
  - Amount of Wei to transfer and the message to be sent to the contract address
  - An optional data field (input data for the contract).
  - The maximum amount of gas (startgas) that can be consumed.

Ethereum Blockchain Elements | Transactions and messages

Messages

- Messages are generated when the CALL or DELEGATECALL opcodes are executed by the contract running in the EVM.

- The segregation between the two types of transactions (*contract creation and message calls*) is shown here:



Calls

- A call does not broadcast anything to the blockchain; instead, it is a local call and executes locally on the Ethereum node. It is almost like a local function call. It does not consume any gas as it is a read-only operation. It is akin to a dry run or a simulated run.
- Calls also do not allow ether transfer to CAs.
- Calls are executed locally on a node EVM and do not result in any state change because they are never mined.
- Calls are processed synchronously and they usually return the result immediately.

Calls

- Do not confuse **a call** with **a message call transaction**, which in fact results in a state change.
- **A call** basically runs message call transactions locally on the client and never costs gas nor results in a state change. It is available in the web3.js JavaScript API and can be seen as almost a simulated mode of the message call transaction.
- **A message call transaction** is a write operation and is used for invoking functions in a CA (Contract Account, or smart contract), which does cost gas and results in a state change.

Transaction validation and execution

- Transactions are executed after their validity has been verified. The initial checks are listed as follows:
  - A transaction must be well formed and RLP-encoded without any additional trailing bytes.
  - The digital signature used to sign the transaction must be valid.
  - The transaction nonce must be equal to the sender's account's current nonce.
  - The gas limit must not be less than the gas used by the transaction.
  - The sender's account must contain sufficient balance to cover the execution cost.

The transaction substate

- A transaction substate is created during the execution of the transaction and is processed immediately after the execution completes. This transaction substate is a tuple that is composed of four items:

1. **Suicide set or self-destruct set:** This element contains the list of accounts (if any) that are disposed of after the transaction executes.

2. **Log series:** This is an indexed series of checkpoints that allows the monitoring and notification of contract calls to the entities external to the Ethereum environment, such as application frontends. It works like a trigger mechanism that is executed every time a specific function is invoked, or a specific event occurs.

   Logs are created in response to events occurring in the smart contract. It can also be used as a cheaper form of storage.

The transaction substate

- A transaction substate is created during the execution of the transaction and is processed immediately after the execution completes. This transaction substate is a tuple that is composed of four items:

3. **Refund balance:** This is the total price of gas for the transaction that initiated the execution. Refunds are not immediately executed; instead, they are used to offset the total execution cost partially.

4. **Touched accounts:** Touched accounts can be defined as those accounts which are involved any potential state changing operation. Empty accounts from this set are deleted at the end of the transaction. The final state is reached after deleting accounts in the self-destruct set and emptying accounts from the touched accounts set.

State storage in the Ethereum blockchain

- At a fundamental level, the Ethereum blockchain is a transaction- and consensus-driven state machine.
- The state needs to be stored permanently in the blockchain. For this purpose, the world state, transactions, and transaction receipts are stored on the blockchain in blocks.
- **The world state**
  - This is a mapping between Ethereum addresses and account states. The addresses are 20 bytes (160 bits) long. This mapping is a data structure that is serialized using RLP.

State storage in the Ethereum blockchain

- **The account state**
  - The account state consists of four fields: nonce, balance, storage root, and code hash, and is described in detail here:

    1. **Nonce:** This is a value that is incremented every time a transaction is sent from the address. In the case of CAs, it represents the number of contracts created by the account.

    2. **Balance:** This value represents the number of weis, which is the smallest unit of the currency (ether) in Ethereum, held by the given address.

State storage in the Ethereum blockchain

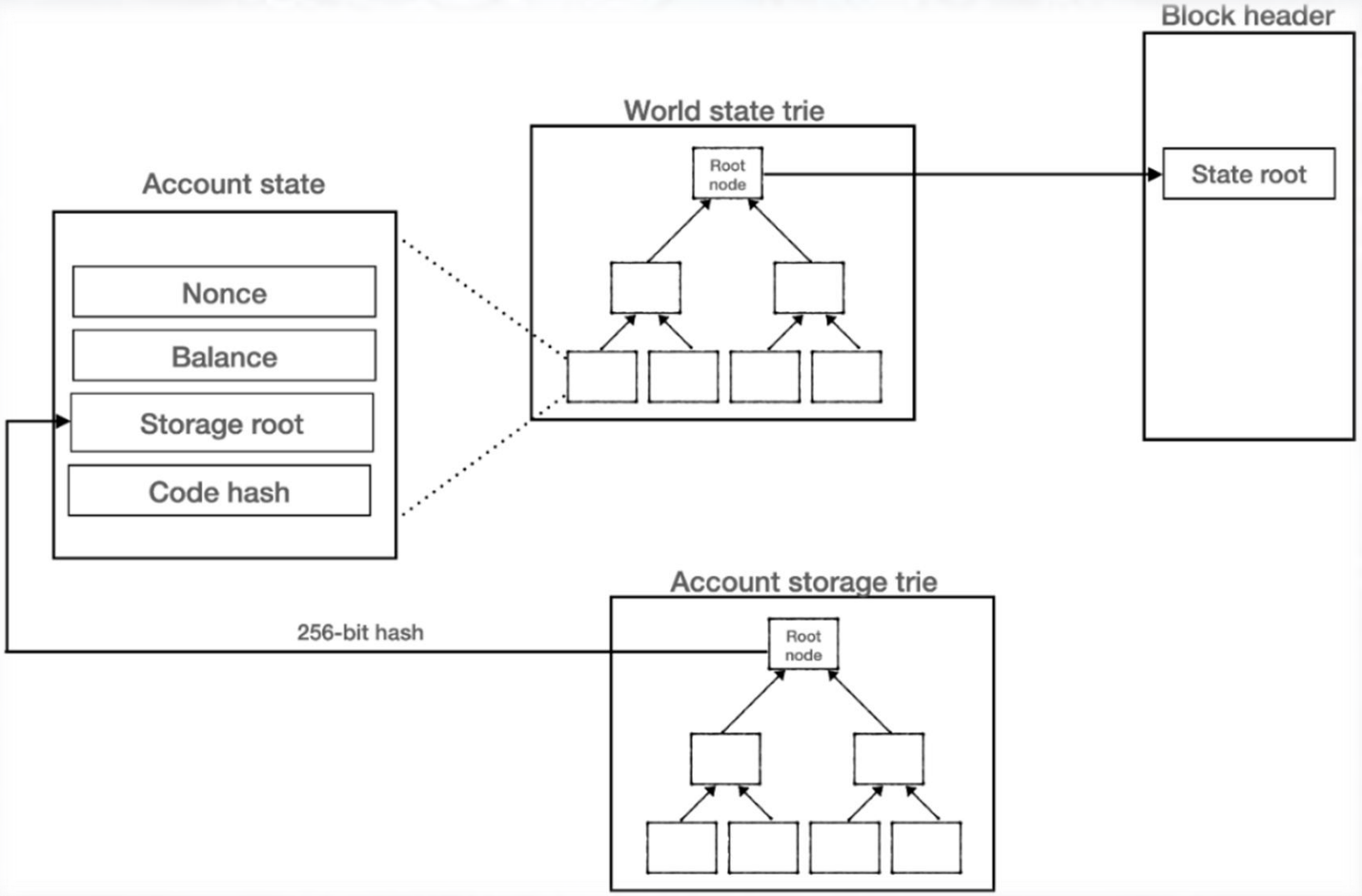
- **The account state**
  - The account state consists of four fields: nonce, balance, storage root, and code hash, and is described in detail here:

3. **Storage root:** This field represents the root node of an MPT that encodes the storage contents of the account.

4. **Code hash:** This is an immutable field that contains the hash of the smart contract code that is associated with the account. In the case of normal accounts, this field contains the Keccak 256-bit hash of an empty string. This code is invoked via a message call.

Ethereum Blockchain Elements | Transactions and messages

State storage in the Ethereum blockchain



State storage in the Ethereum blockchain

- The **world state** and its relationship with the **accounts trie**, **accounts**, and **block header** are visualized in the previous diagram.
- It shows **the account state**, or **data structure**, which contains a **storage root hash** derived from **the root node** of the **account storage trie** shown on the left.
- The **account data structure** is then used in the **world state trie**, which is a **mapping** between **addresses** and **account states**.
- The **accounts trie** is an **MPT** used to encode the storage contents of an account. The contents are stored as a mapping between Keccak 256-bit hashes of 256-bit integer keys to the RLP-encoded 256-bit integer values.
- Finally, the **root node of the world state trie** is hashed using the Keccak 256-bit algorithm and made part of the **block header data structure**, which is shown on the right-hand side of the diagram as the **state root hash**.

Transaction receipts

- Transaction receipts are used as a mechanism to store the state after a transaction has been executed.
- In other words, these structures are used to record the outcome of the transaction execution.
- **It is produced after the execution of each transaction**.
- All receipts are stored in an index-keyed trie.
- The hash (a 256-bit Keccak hash) of the root of this trie is placed in the block header as the receipt's root.

Transaction receipts

- Transaction receipts composed of four elements as follows:

1. **The post-transaction state:** This item is a trie structure that holds the state after them transaction has been executed. It is encoded as a byte array.
2. **Gas used:** This item represents the total amount of gas used in the block that contains the transaction receipt. The value is taken immediately after the transaction execution is completed.

   The total gas used is expected to be a non-negative integer.

Transaction receipts

- Transaction receipts composed of four elements as follows:

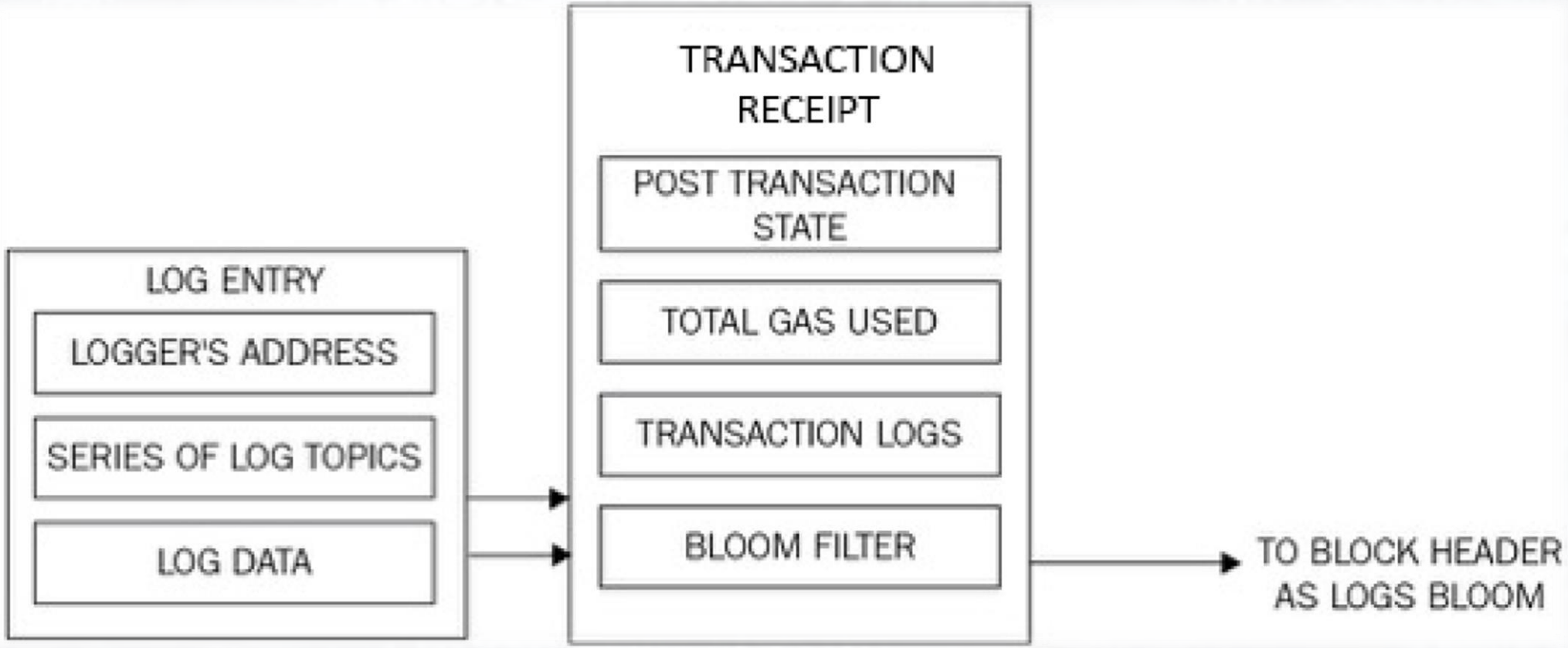
3. **Set of logs:** This field shows the set of log entries created as a result of the transaction execution. Log entries contain the logger's address, a series of log topics, and the log data.

4. **The bloom filter:** A bloom filter is created from the information contained in the set of logs discussed earlier. A log entry is reduced to a hash of 256 bytes, which is then embedded in the header of the block as the logs bloom. A log entry is composed of the logger's address, log topics, and log data.

   Log topics are encoded as a series of 32-byte data structures. The log data is made up of a few bytes of data.

Transaction receipts

- This process of transaction receipt generation is visualized in the following diagram:



Introduction to Ethereum | Summary

►In this session, we discussed:

- Ethereum Blockchain
- The Ethereum networks
  - Mainnet
  - Testnet
  - Private networks
- Components of the Ethereum ecosystem
- Ethereum Blockchain Elements
  - Key and Addresses
  - Accounts
  - Transactions and Messages


THANK
YOU