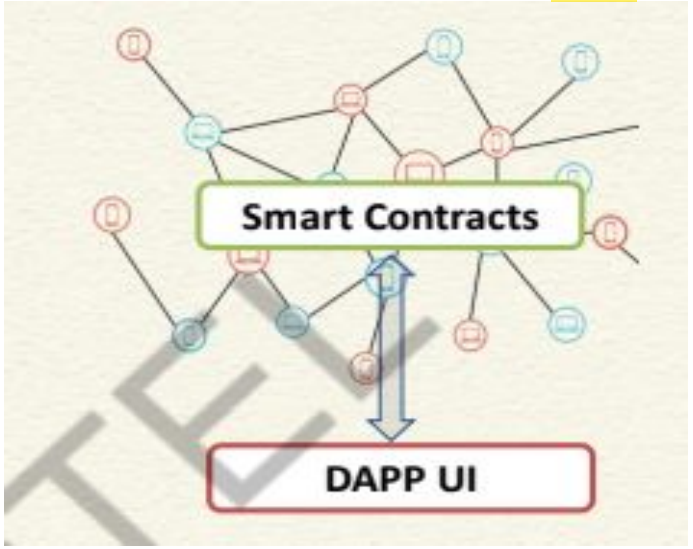
- **BLOCKCHAIN 2.0 - ETHEREUM**: Distributed applications (Dapps), Smart contracts, Ethereum Virtual Machines, Ethereum high level design, Ethereum addresses, Ethereum accounts, Transactions, Currency, Gas, Tokens, Decentralized autonomous organizations (DAOs), Bitcoin vs Ethereum – Trie- Solidity programming – writing smart contracts – remix IDE – TestNet -sample exercises - issues in solidity programming

Distributed applications (Dapps)

- DAPP - decentralized application

Application that is built for decentralized networks like Ethereum

- Decentralized applications (dApps) are digital applications or programs that run on a decentralized network rather than a single computer or server. They are built on blockchain technology and use cryptocurrency as a means of exchange.



- Combines two components:
  1. Smart Contracts
  2. User interface for executing transactions and contracts

Decentralized application

- Decentralized applications or dApps, are software programs that run on a blockchain or peer-to-peer (P2P) network of computers instead of on a single computer.
- Rather than operating under the control of a single authority, dApps are spread across the network to be collectively controlled by its users.
- They are often built on the Ethereum platform and have been developed for various purposes, including wallets, exchanges, gaming, personal finance and social media.

Understanding Decentralized Applications (dApps)

- A web app such as Uber or X (formerly Twitter) runs on a computer system that is owned and operated by a company with authority over the app and its workings. No matter how many users there are, the backend is controlled by the company.
- DApps operate a bit differently. They run on a P2P or a blockchain network. For example, BitTorrent are applications that run on computers that are part of a P2P network, which allows multiple participants to consume, feed or seed content.

Importance of dApps

- **Cost and Efficiency** : Because dApps operate on decentralized networks, there is no need for an intermediary. This can lead to reduced costs, increased efficiency, and greater accessibility
- **Security** : Blockchains make data immutable by leveraging cryptographic techniques and distributed automated consensus.
- **Accessibility** : DApps are accessible to anyone with an internet connection.
- **Transparency** : Blockchain-based dApps maintain transparent records of transactions, meaning users can verify the integrity of data without relying on centralized authorities.

Examples of practical uses for dApps

- **Financial services**: Facilitating peer-to-peer financial transactions, such as currency exchanges or asset transfers.
- **Supply chain management**: Tracking the movement of goods through a supply chain, ensuring transparency and accountability.
- **Identity verification**: Securely storing and verifying identity information, such as for voter rolls or passport applications.
- **Real estate**: Facilitating real estate transactions directly between buyer and seller, tracking property ownership and related documentation, such as deeds.

- **Healthcare**: Storing and tracking healthcare records and facilitating communications between healthcare professionals.
- **Education**: Creating decentralized learning platforms that allow students and teachers to interact and collaborate directly without the need for intermediaries.
- **Social media**: Creating decentralized social media platforms that allow users to interact and share content without being censored by a centralized authority.
- **Predictive markets**: Creating decentralized platforms for predictive markets, allowing users to make bets on any event.

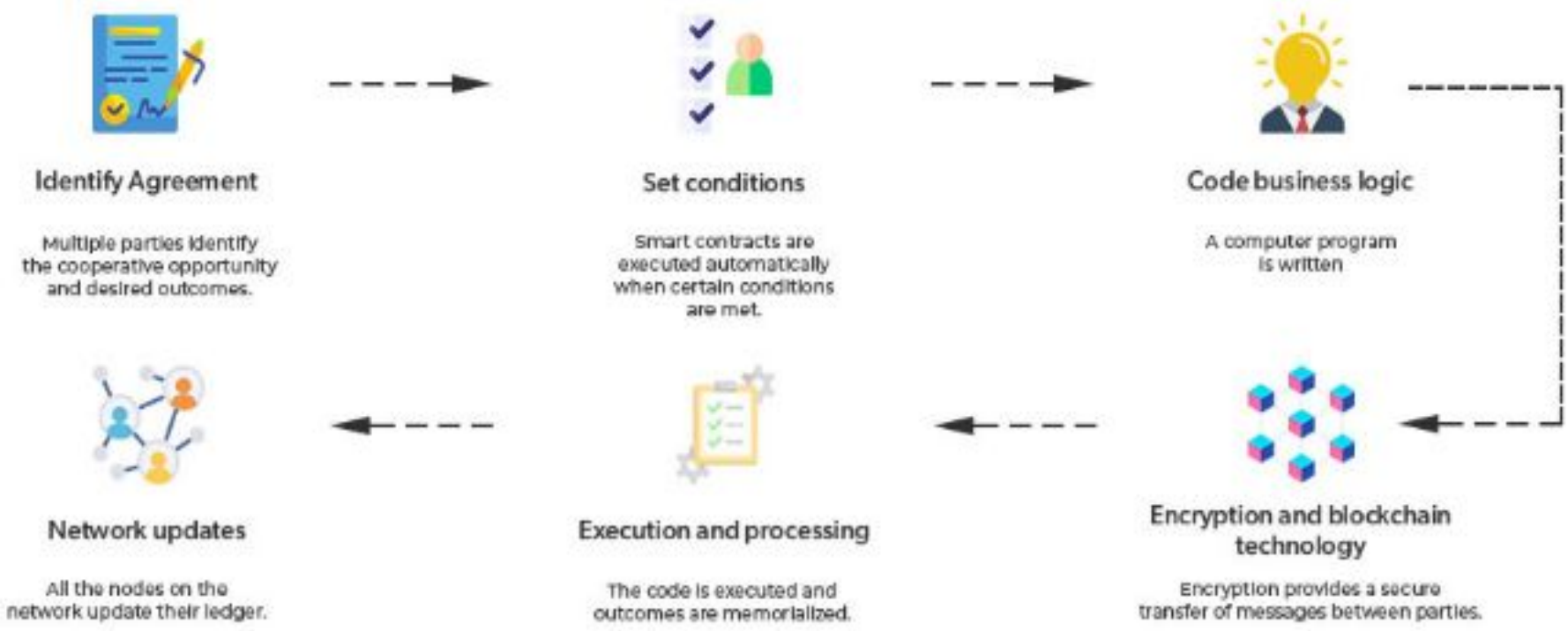
Example of dApps

- One popular example of a dApp is CryptoKitties.9 CryptoKitties is a blockchain-based virtual game that allows players to adopt, raise, and trade virtual cats. The game is one of the world's first forms of interactive blockchain dApps.

Smart contracts

- Program that is executed in a decentralized setting.
- Acts on distributed ledger data.
- Output of the program depends on consensus of independent participants executing it.
- A Smart Contract (or cryptocontract) is a computer program that directly and automatically controls the transfer of digital assets between the parties under certain conditions. A smart contract works in the same way as a traditional contract while also automatically enforcing the contract. Smart contracts are programs that execute exactly as they are set up(coded, programmed) by their creators.

Working of Smart Contract



Smart Contract

- A smart contract is a self-executing program that automates the actions required in a blockchain transaction.
- Once done, these transactions are traceable and cannot be undone.
- The best way to envision a smart contract is to think of a vending machine—when you insert the correct amount of money and push an item's button, the program (the smart contract) activates the machine to dispense your chosen item.

Advantages and Challenges of Smart Contracts

- **Efficiency**: They speed up contract execution
- **Accuracy**: There can be no human error introduced
- **Immutability**: The programming cannot be altered
- Some of the downfalls of smart contracts are:
- **Permanent**: They cannot be changed if there are mistakes
- **Human factor**: They rely on the programmer to ensure the code is programmed properly to execute the intended actions
- **Loopholes**: There may be loopholes in the coding, allowing for contracts to be executed in bad faith

Example of a Smart Contract

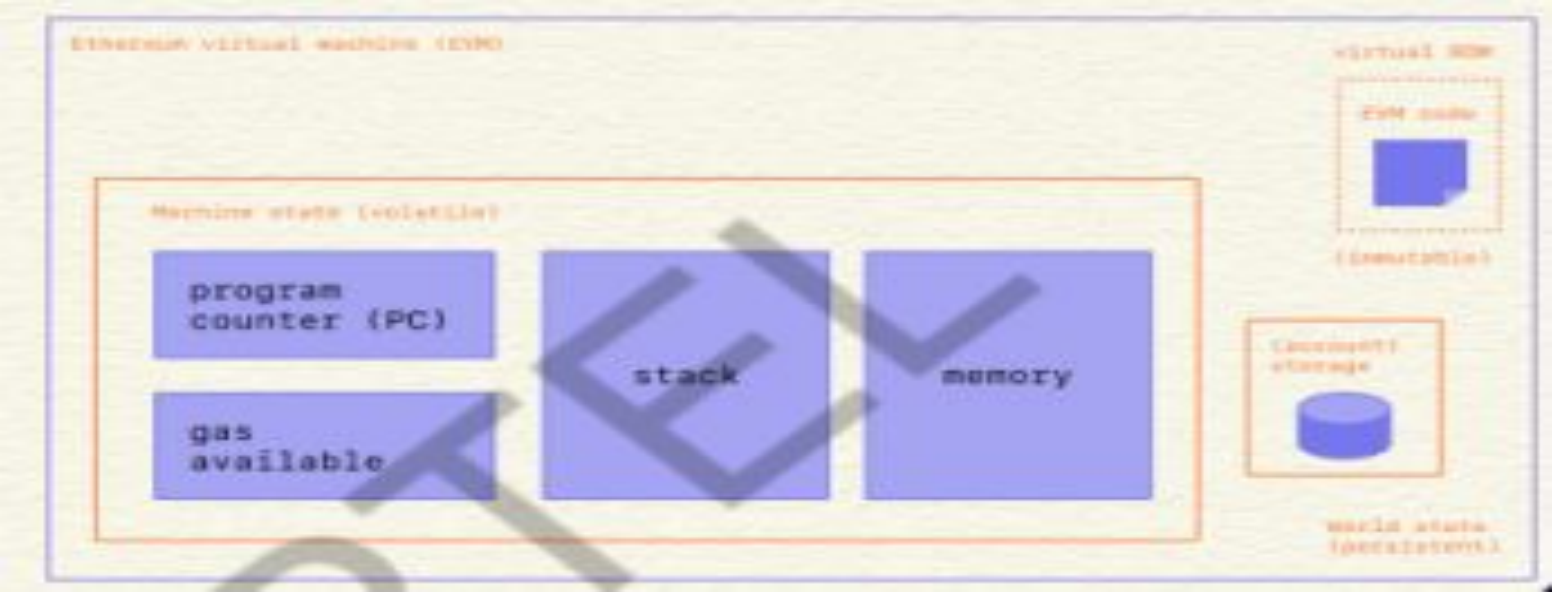
- A basic example of a smart contract is a sale transaction between a consumer and a business. The smart contract could execute the customer's payment and initiate the business's shipment process.

Ethereum Smart Contracts

- Program that reads data from the Ethereum ledger, performs operations on it, and writes back the output to the ledger.
- Smart contracts are a type of Ethereum account.
  - have a balance
  - can send transactions over the network
  - not controlled by a user, run as programmed
- User accounts interact with a smart contract by submitting transactions that execute a function defined on the smart contract.
- Smart contracts aim to eliminate the need for a trusted third party to facilitate actions between untrusting parties.

Ethereum Virtual Machine - EVM

- Ethereum implements a distributed state machine / replicated state machine.
- Ledger data holds the state - accounts, balances, and other variables.
- Transactions deterministically change the machine from one state to another.



What is Ethereum?

- Ethereum is the open source decentralized software program that allows you to create and deploy applications.
- Vitalik Buterin conceptualized Ethereum in November, 2013.
- The critical idea proposed was the development of a Turing-complete language that allows the development of arbitrary programs (smart contracts) for blockchain and decentralized applications.

What is Ethereum?

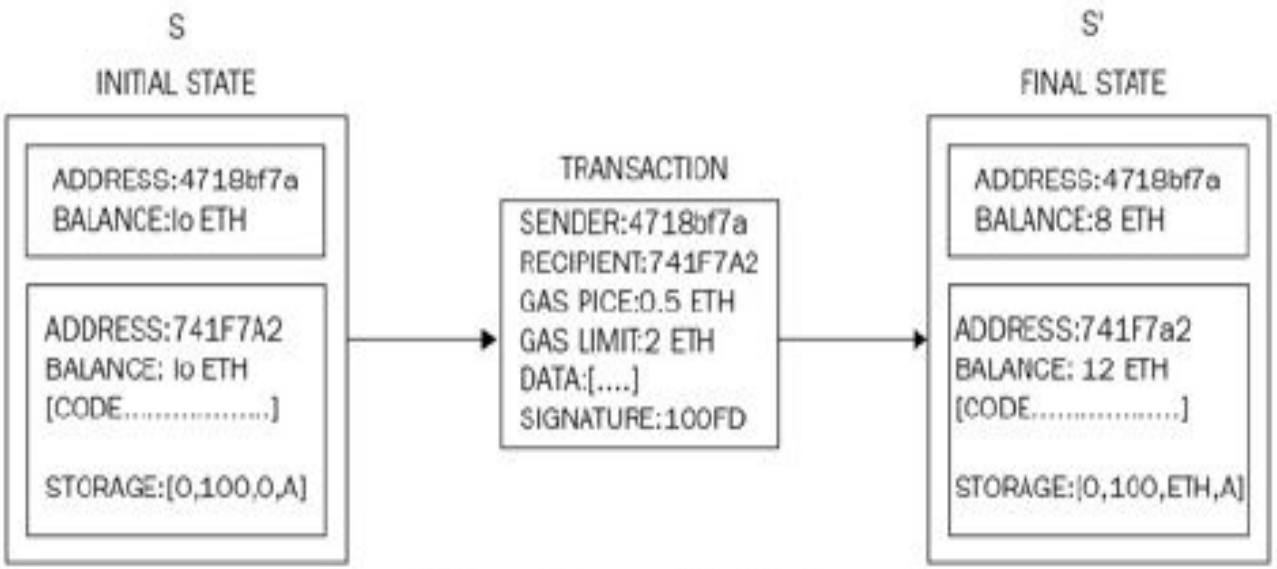
- The following table shows all the releases of Ethereum starting from the first release to the planned final release:

Version	Release date
Olympic	May, 2015
Frontier	July 30, 2015
Homestead	March 14, 2016
Byzantium (first phase of Metropolis)	October 16, 2017
Metropolis	To be released
Serenity (final version of Ethereum)	To be released

Ethereum blockchain

- Like any other blockchain, can be visualized as a **transaction-based state machine**.
- The core idea is that in Ethereum blockchain, a genesis state is transformed into a final state by executing transactions incrementally.
- The final transformation is then accepted as the absolute **undisputed** version of the state.
- The Ethereum state transition function is shown in diagram. where a transaction execution has resulted in a state transition

Ethereum blockchain

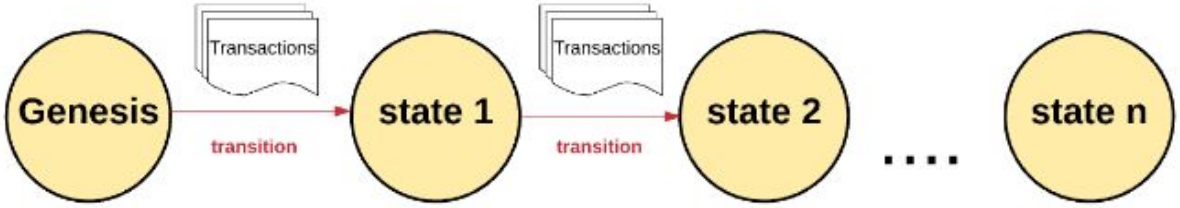


Ethereum State transition function

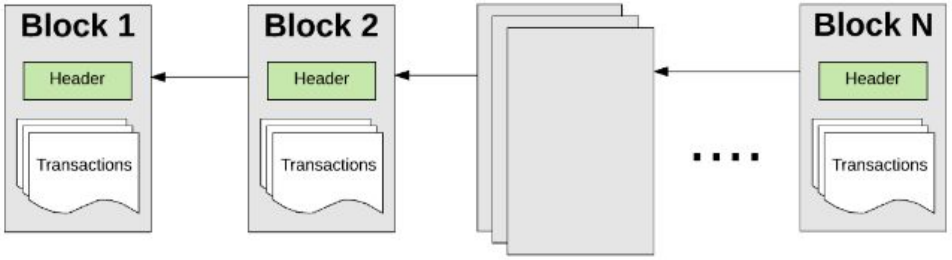
- Example: a transfer of two Ether from address **4718bf7a to address 741f7a2 is initiated.**
- **The** initial state represents the state before the transaction execution, and the final state is what the morphed state looks like.
- Mining plays a central role in state transition.
- The state is stored on the Ethereum network as the ***world state***. *This is the* ***global state of the*** Ethereum blockchain.

Ethereum Blockchain

With **Ethereum's state machine**, we begin with a "**genesis state**." This is analogous to a blank slate, before any transactions have happened on the network
When **transactions are executed**, this **genesis state transitions** into **some final state**.
At any point in time, this **final state represents the current state of Ethereum**.



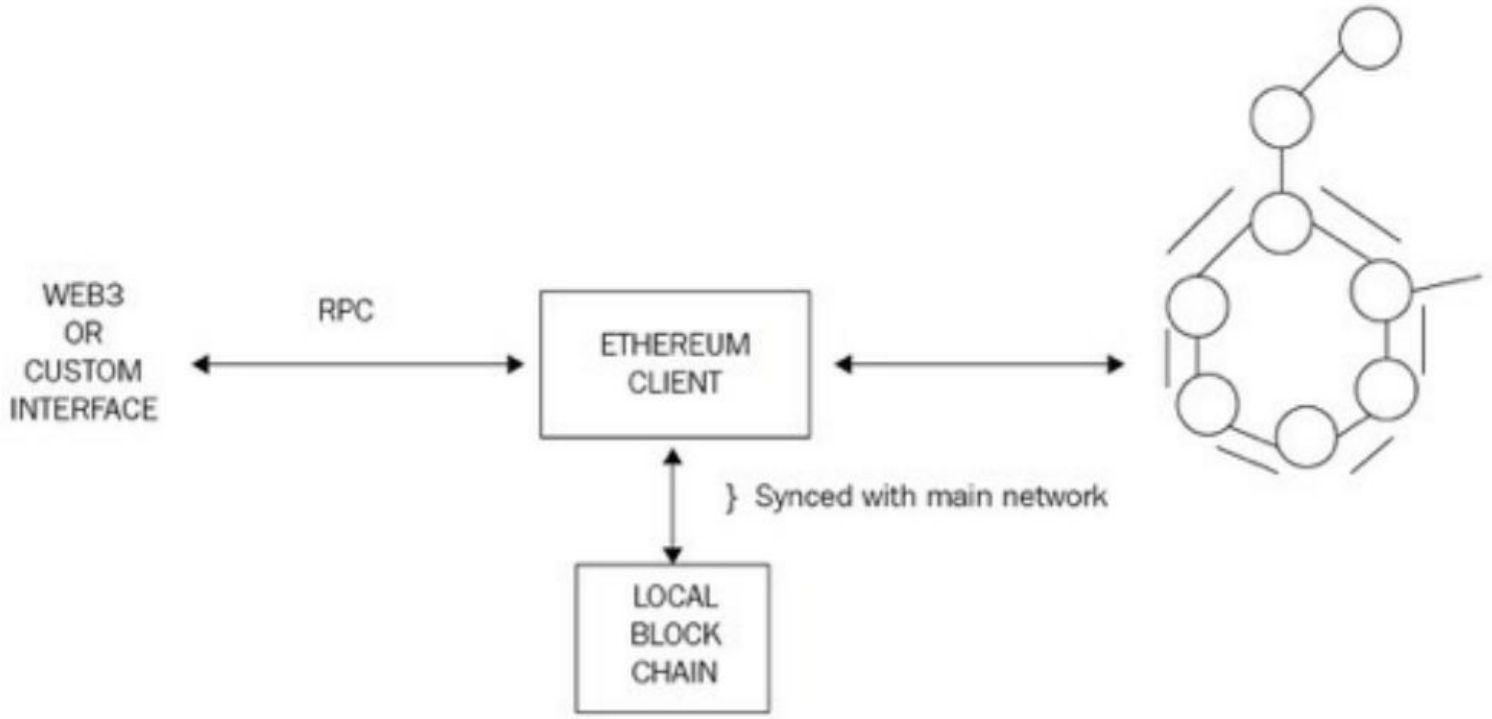
The state of Ethereum has millions of transactions. These transactions are grouped into "blocks." A block contains a series of transactions, and each block is chained together with its previous block.



Components of the Ethereum ecosystem

- The Ethereum blockchain stack consists of various **components.**
- At the core, there is the **Ethereum blockchain** running on the peer-to-peer Ethereum network.
- Secondly, 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.
  - It provides various functions, such as **mining and account management.**
  - The local copy of the blockchain is synchronized regularly with the network.
- Another component is the **web3.js** library that allows interaction with the geth client via the **Remote Procedure Call (RPC) interface.**

Ethereum Architecture



The Ethereum stack showing various components

Ethereum Architecture

- A formal list of all high-level elements present in the Ethereum blockchain is presented here:
  - Keys and addresses
  - Accounts
  - Transactions and messages
  - Ether crypto currency/tokens
  - The EVM
  - Smart contracts

Keys and addresses

- Mainly to **represent ownership and transfer of Ether.** Keys are used in pairs of private and public type.
- The private key is generated randomly and is kept secret whereas a public key is derived from the private key.
- **Addresses are derived from the public keys** which are a **20-bytes code used to identify accounts**.
- The process of key generation and address derivation is described here:

1. A private key is randomly chosen (256 bits positive integer) under the rules defined by elliptic curve secp256k1 specification (in the range [1, secp256k1n − 1]).
2. The public key is then derived from this private key using ECDSA recovery function.
3. An address is derived from the public key which is the right most 160 bits of the **Keccak hash** of the public key.

Example

1.Privatekey: b51928c22782e97cca95c490eb958b06fab7a70b9512c38c36974f47b954ffc2

2.Publickey:
3aa5b8eefd12bdc2d26f1ae348e5f383480877bda6f9e1a47f6a4afb35cf998ab847f1e394
8b1173622dafc6b4ac198c97b18fe1d79f90c9093ab2ff9ad99260

3. Address: 0x77b4b5699827c5c49f73bd16fd5ce3d828c36f32

Accounts

- Accounts are one of the **main building blocks of the Ethereum blockchain**.
- Ethereum, being a transaction driven state machine, the state is created or updated as a result of the interaction between accounts and transaction execution.
- Operations performed between and on the accounts, represent state transitions. 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, sender's account balance is checked and subtracted accordingly, and nonce is incremented. An error is returned if the account balance is not enough.

Accounts

3. Provide enough Ether (gas price) to cover the cost of the transaction. This is charged per byte incrementally proportional to the size of the transaction. here the actual transfer of value occurs.

- The flow is from the sender's account to 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.

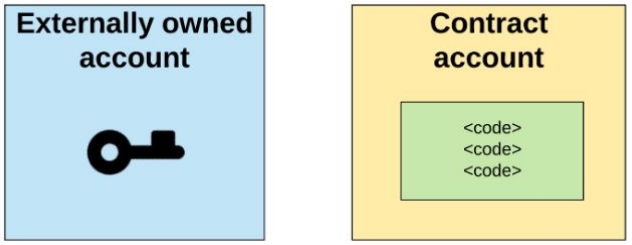
  If enough gas is available, then the contract code will be executed fully; otherwise, it will run up to the point where it runs out of gas.

4. In case of transaction failure due to insufficient account balance or gas, all state changes are rolled back **except for 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 fee are paid to the miners accordingly. At this point, the function returns the **resulting state which is also stored on the blockchain**.

Types of accounts

Two kinds of accounts exist in Ethereum:

1. **Externally Owned Accounts (EOAs)** : EOAs are similar to accounts that are controlled by a private key in Bitcoin.
2. **Contract Accounts (CAs):** the accounts that have code associated with them along with the private key.



EOAs properties are :

- EOAs has ether balance
- They are capable of sending transactions
- They have no associated code
- They are controlled by private keys
- Accounts contain a **key-value store**
- They are associated with a human use

Types of accounts

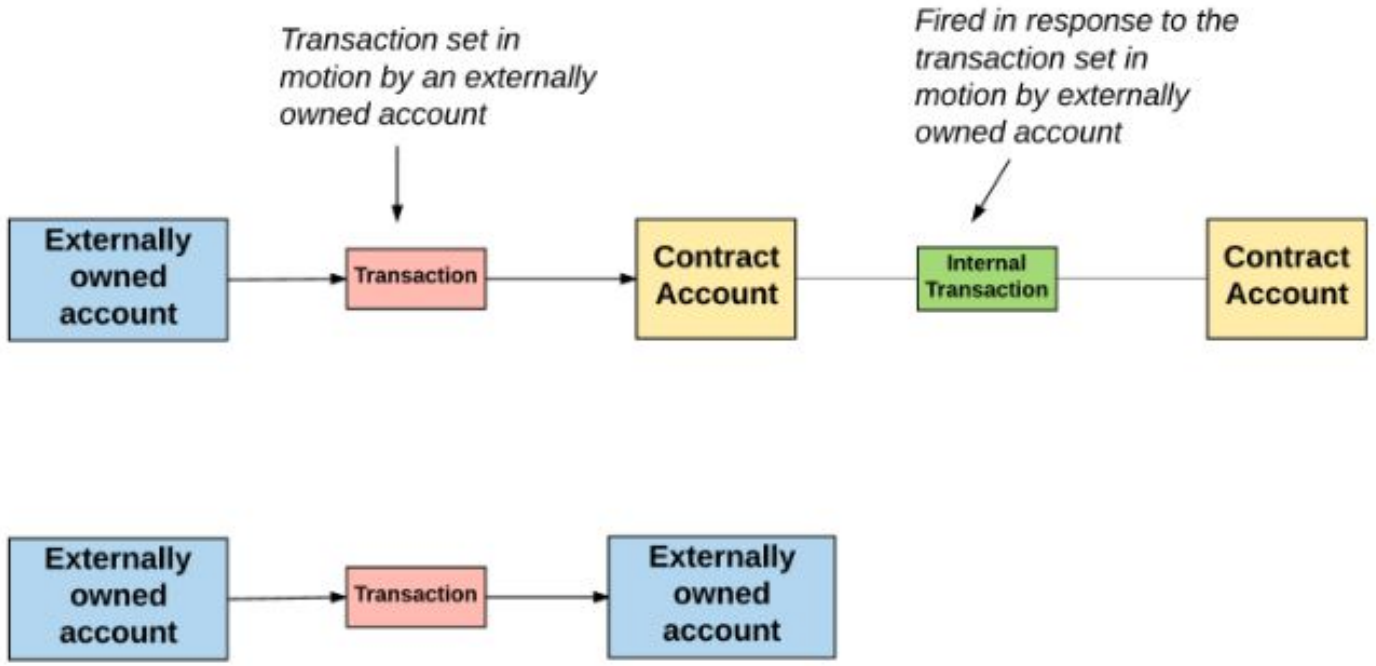
CAs properties are

- CAs have Ether balance.
- They have **associated code** that is kept in memory/storage on the blockchain.
- 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 contract accounts can be of **any level of complexity**. The code is executed by **Ethereum Virtual Machine** (EVM) by each mining node on the Ethereum network.
- Also, CAs can maintain their permanent state and can call other contracts. It is envisaged that in the **serenity release**, the distinction between externally owned accounts and contract accounts may be eliminated.
- They are **not intrinsically associated with any user or actor** on the blockchain.
- CAs contain **a key-value store**.

Externally owned accounts vs. contract accounts

- **An externally owned account can send messages to other externally owned accounts OR to other contract accounts by creating and signing a transaction using its private key.**
- A message between two externally owned accounts is simply a **value transfer.**
- But a message from an externally owned account to a contract account activates the contract account'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 externally owned accounts, contract accounts can't initiate new transactions on their own.** Instead, contract accounts can only fire transactions in response to other transactions they have received (from an externally owned account or from another contract account).

Externally owned accounts vs. contract accounts

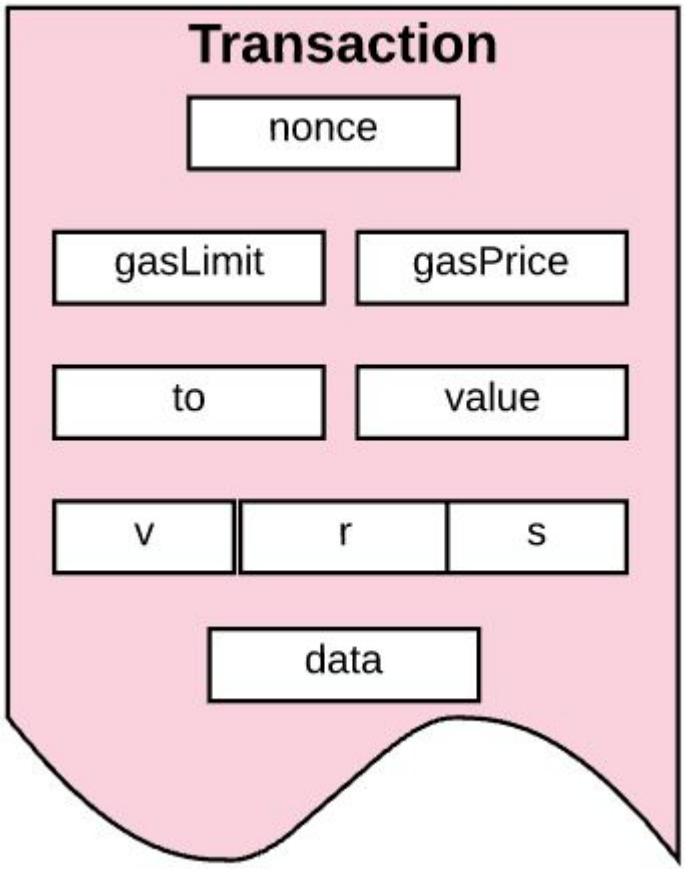


- **EOA = wallet controlled by a person's private key.**
- **CA = smart contract controlled by code, not by a private key.**
- EOAs **start transactions**, CAs **respond and execute logic**.

Transactions and messages

- A transaction in Ethereum is a **digitally signed data packet using a private key** that contains the instructions that, when completed, either result in a **message call or contract creation.**
- Transactions can be divided into two types based on the output they produce:
- **Message call transactions:** This transaction simply produces a message call that is used to pass messages from one contract account to another.
- **Contract creation transactions:** As the name suggests, these transactions result in the creation of a new contract account. This means that when this transaction is executed successfully, it creates an account with the associated code.

Transaction



Transactions and messages

- Both of these transactions are composed of some standard fields, which are described here.
- **Nonce**

✔ 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

- *Wei is the smallest denomination of ether; therefore, it is used to count ether.*

1. Gas : A unit that measures the **amount of computational work** a transaction or smart contract operation requires.

Gas = "how much work is needed"

2. Gas Price: How much **ETH you are willing to pay per unit of gas**.

- Expressed in **gwei** (1 gwei = 10^{-9} ETH).
- Gas Price = "price per liter of fuel"

3. Gas Limit: The **maximum amount of gas** you are willing to let your transaction consume.

- Gas Limit = "maximum liters of fuel you're willing to buy"
- Transaction Fee = **Gas Used × Gas Price**
- Eg: Transfer **ETH** (needs 21,000 gas).
- Set **Gas Price = 50 gwei**.
- Set **Gas Limit = 50,000** (safe buffer).
- Actual gas used = 21,000.
- Transaction fee = 21,000 × 50 gwei = **1,050,000 gwei = 0.00105 ETH**.
- (1 gwei = 10^{-9} ETH)

How Ethereum works from a user's point of view?

- Most common use case of transferring funds. from one user (Bob) to another (Alice).
- We will use two Ethereum clients, one for sending funds and the other for receiving. There are several steps involved in this process

1. First either a user requests money by sending the request to the sender, or the sender decides to send money to the receiver.

- If Alice requests money from Bob, then she can send a request to Bob by using QR code.
- Once Bob receives this request he will either scan the QR code or manually type in Alice's Ethereum address and send **Ether** to Alice's address. **Jaxx wallet**

How Ethereum works from a user's point of view?

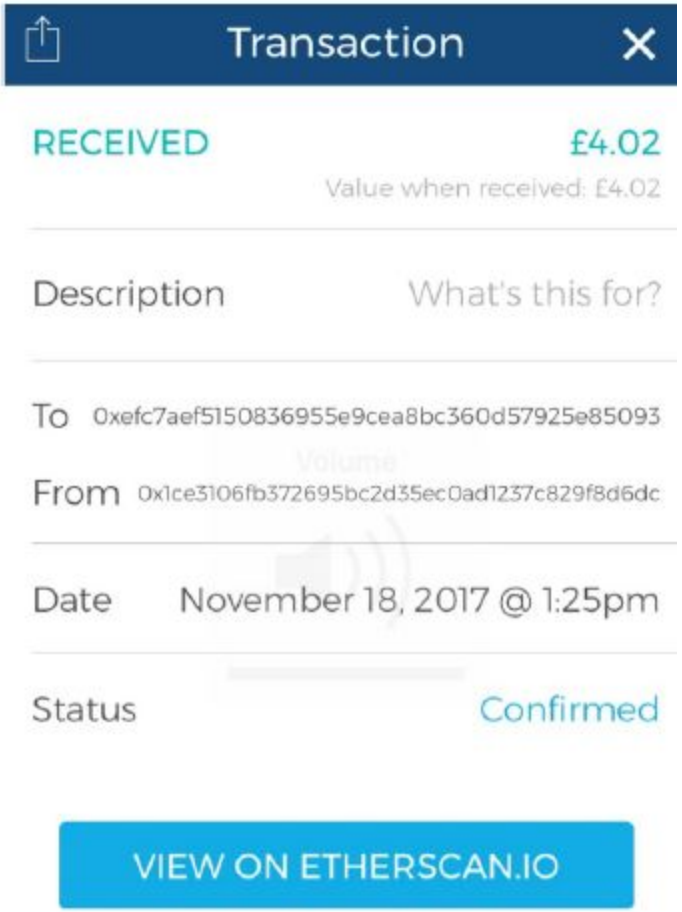
2. Once Bob receives this request he will either scan this QR code or copy the Ethereum address in the Ethereum wallet software and initiate a transaction.

- This process is shown in the following screenshot
- where the Jaxx Ethereum wallet software on iOS is used to send money to Alice.
- The following screenshot shows that the sender has entered both the amount and destination address for sending Ether.
- Just before sending the Ether the final step is to confirm the transaction.

Sending of funds confirmation in Jaxx wallet from Bashir

How Ethereum works from a user's point of view?

3. Once the request (transaction) of sending money is constructed in the wallet software, it is then broadcasted to the Ethereum network. The transaction is digitally signed by the sender as proof that he is the owner of the Ether.
4. This transaction is then picked up by nodes called **miners** on the Ethereum network for verification and inclusion in the block. At this stage, the transaction is still unconfirmed.
5. Once it is verified and included in the block, the PoW process starts.
6. Once a miner finds the answer to the PoW problem, by repeatedly hashing the block with a new nonce, this block is immediately broadcasted to the rest of the nodes which then verifies the block and PoW.
7. If all the checks pass then this block is added to the blockchain, and miners are paid rewards accordingly.
8. Finally, Alice gets the Ether, and it is shown in her wallet software. This is shown here:



https://etherscan.io/

Transactions and messages

- **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 body, which persists and runs in response to **message calls that the contract account** may receive. These message calls may be sent via a transaction or an internal code execution.

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

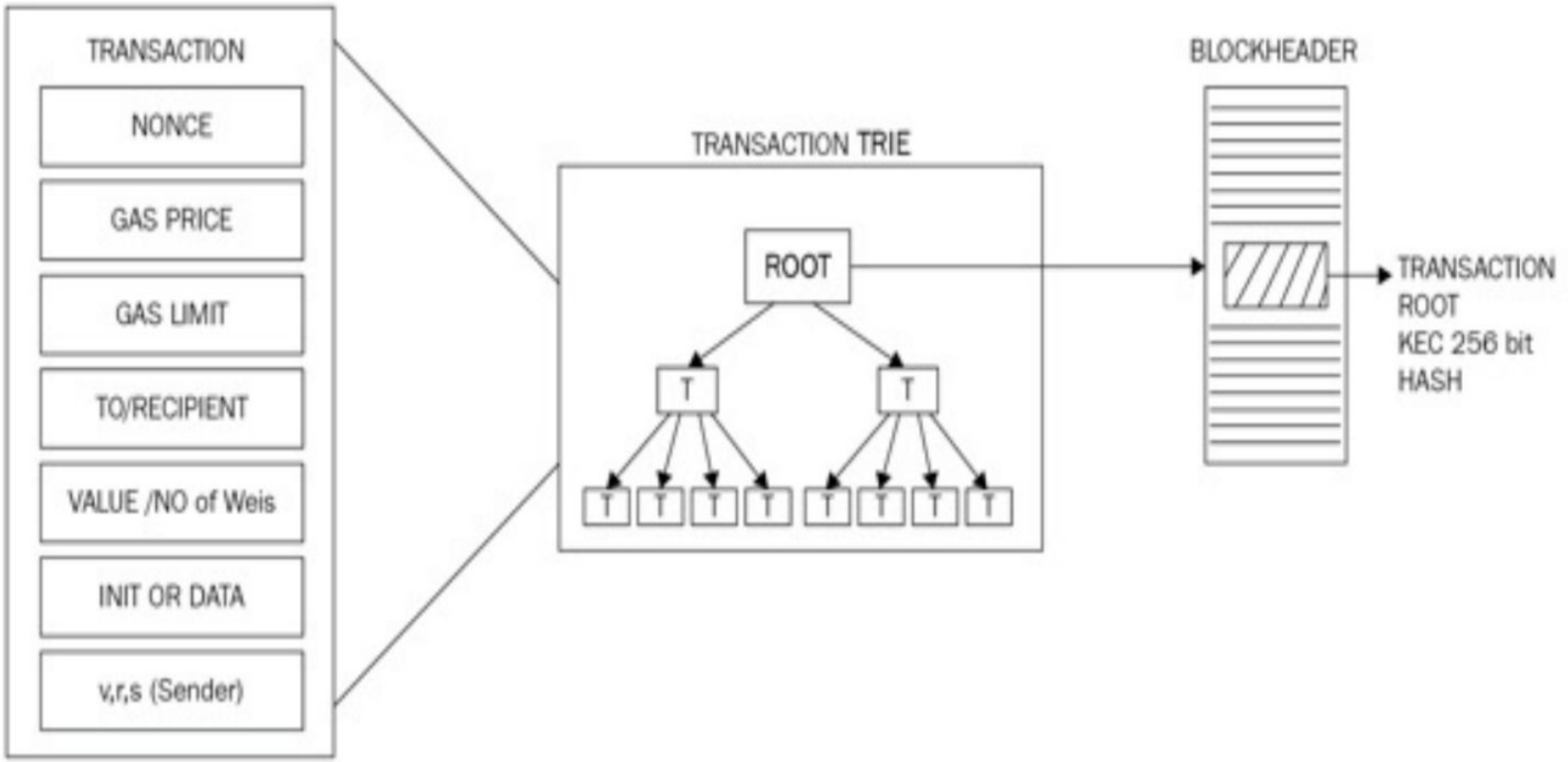
✔ This structure can be visualized in the following diagram, where a transaction is a tuple of the fields mentioned earlier, which is then included in a transaction trie (a modified Merkle-Patricia tree) composed of the transactions to be included.

✔ Finally, the root node of transaction trie is hashed using a **Keccak 256-bit algorithm** and is included in the block header along with a list of transactions in the block.

Transactions and messages

- Transactions can be found in either transaction pools or blocks.
- In transaction pools, they wait for verification by a node or in blocks, they are added after successful verification.
- When a mining node starts its operation of verifying blocks, it starts with the highest paying transactions in the transaction pool and executes them one by one.
- When the gas limit is reached, or no more transactions are left to be processed in the transaction pool, the mining starts.
- In this process, the block is repeatedly hashed until a valid nonce is found such that, once hashed with the block, it results in a value less than the difficulty target.
- 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.
- This process is similar to Bitcoin's mining process discussed in the previous chapters, The only difference is that **Ethereum's PoW algorithm is ASIC-resistant, known as Ethash, where finding a nonce requires large memory.**

Transactions and messages



The relationship between transaction, transaction trie and block header

Contract creation transaction

- There are a few essential parameters that are required when creating an account. These parameters are listed as follows:
  - Sender
  - Original transactor (transaction originator)
  - Available gas
  - Gas price
  - Endowment, which is the amount of ether allocated
  - A byte array of an arbitrary length
  - Initialization EVM code
  - Current depth of the message call/contract-creation stack (current depth means the number of items that are already there in the stack)

Contract creation transaction

- Addresses generated as a result of contract creation transaction are 160 bit 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.
- Initially, the **nonce** in the account is **set to zero.** The balance of the account is set to the value passed to the contract.
- **Storage** is also set to **empty**. Code hash is Keccak 256-bit hash of the empty string.
- The new account is initialized when the EVM code (the Initialization EVM code, mentioned earlier) is executed.
- In the case of **any exception during code execution**, such as not having enough gas (running Out Of Gas, OOG), the **state does not change**.
- If the **execution is successful**, then the **account is created after the payment of appropriate gas costs.**
- Since the **Ethereum (Homestead)** is 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 transaction

- A message call requires several parameters for execution, which are listed as follows:
  - The sender
  - The transaction originator
  - Recipient
  - The account whose code is to be executed (usually same as the recipient)
  - Available gas Value
  - Gas price
  - Arbitrary length byte array
  - Input data of the call
  - Current depth of the message call/contract creation stack

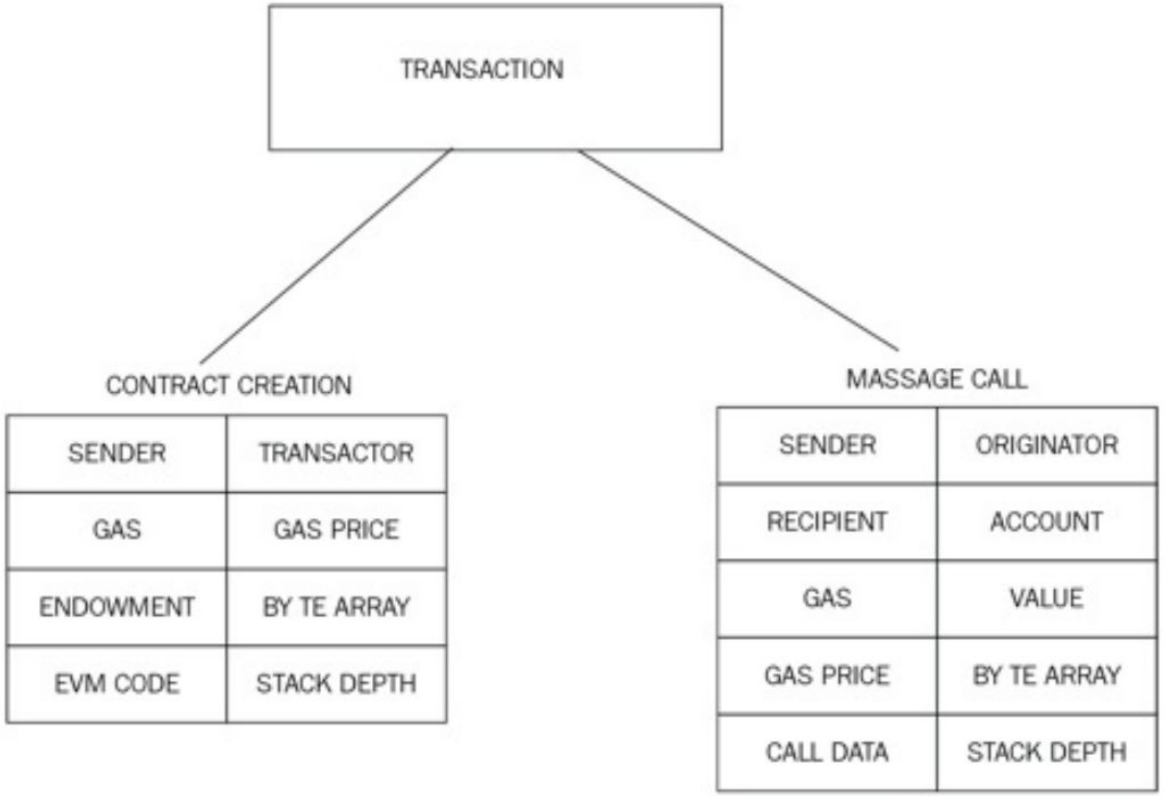
Message call transaction

- Message calls **result in a state transition**.
- The message calls also produce **output data**, which is not used if transactions are executed.
- In cases where message calls are triggered by VM code, the output produced by the transaction execution is used.
- 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 virtual machine 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 are created by external factors and are **signed and then broadcasted to the Ethereum network**. Messages are passed using message calls.
- A description of a message is shown in next slide.

Messages

- **Messages are the data and value** that are passed between two accounts.
- A message is a data packet passed between two accounts. This data packet contains data and value (amount of ether).
- It can either be sent via a smart contract (autonomous object) or from an external actor (externally owned account) 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 are similar to transactions; however, the main difference is that they are produced by the contracts, **whereas transactions are produced by entities external (externally owned accounts**) to the Ethereum environment.
- **A message consists of the components:**
  - ✔ The sender of the message
  - ✔ Recipient of the message
  - ✔ Amount of Wei to transfer and message to the contract address
  - ✔ Optional data field (Input data for the contract)
  - ✔ The maximum amount of gas (start gas) that can be consumed Messages are generated when CALL or DELEGATECALL opcodes are executed by the code in execution by the contracts.

Messages


TRANSACTION
CONTRACT CREATION
SENDER
TRANSACTOR
GAS
GAS PRICE
ENDOWMENT
BY TE ARRAY
EVM CODE
STACK DEPTH
MASSAGE CALL
SENDER
ORIGINATOR
RECIPIENT
ACCOUNT
GAS
VALUE
GAS PRICE
BY TE ARRAY
CALL DATA
STACK DEPTH

Calls

- A call does not broadcast anything to the blockchain; instead, it is a local call to a contract function and runs locally on the 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 are executed locally on a node VM and do **not result in any state change** because they are never mined.
- Call basically runs message call transactions in simulated mode and is available in the **web3.js JavaScript API.**

Transaction validation and execution

- Transactions are executed after verifying the transactions for validity. Initial tests are listed as follows:
- A transaction must be well-formed and RLP(Recursive Length Prefix)-encoded without any additional trailing bytes
- The digital signature used to sign the transaction is valid Transaction
- nonce must be equal to the sender's account's current nonce gas limit must not be less than the gas used by the transaction
- The sender's account contains enough balance to cover the execution cost.

The transaction substate

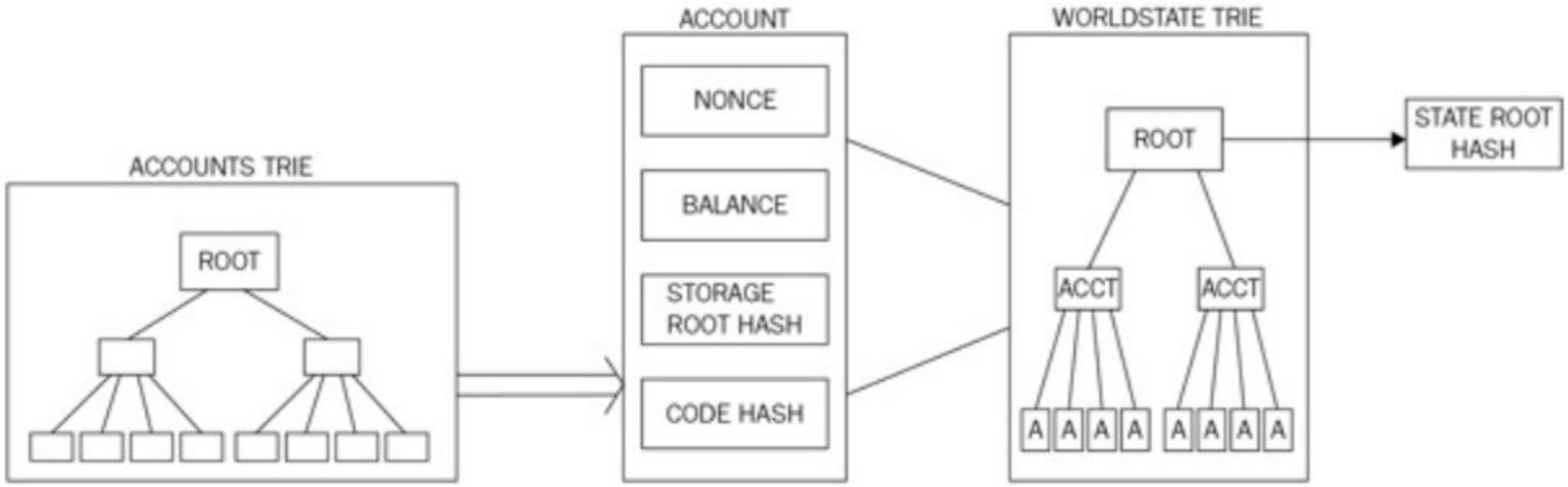
- A transaction substate is created during the execution of the transaction that is processed immediately after the execution completes.
- This transaction substate is a tuple that is composed of four items. These items are described here:
- **Suicide set or self-destruct set:** This element contains the list of accounts (if any) that are disposed of after the transaction executes.
- **Log series:** This is an indexed series of checkpoints that allow 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.
- **Refund balance:** This is the total price of gas in the transaction that initiated the execution. Refunds are not immediately executed; instead, they are used to offset the total execution cost partially.
- **Touched accounts:** This is the set of touched accounts from which empty ones are deleted at the end of the transaction.

State storage in the Ethereum blockchain

- At a fundamental level, **Ethereum blockchain** is a **transaction and consensus-driven state machine**.
- The state needs to be **stored permanently** in the blockchain.
- For this purpose, world state, transactions, and transaction receipts are stored on the blockchain in blocks.
- **The world state** It 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 **Recursive Length Prefix (RLP).** RLP is a specially developed encoding scheme that is used in Ethereum to serialize binary data for storage or transmission over the network and also to save the state in a Patricia tree on storage media.
- The RLP function takes an **item as an input**, which can be a **string or a list of items** and produces **raw bytes** that are suitable for storage and transmission over the network.
- RLP does not encode data; instead, its primary purpose is to encode structures.

The account state

- The account state consists of four fields: **nonce, balance, storage root and code hash**
- **Nonce: T**his is a value that is incremented every time a transaction is sent from the address. In case of contract accounts, it represents the **number of contracts created** by the account.
- **Balance:** This value represents the number of Weis which is the smallest unit of the currency (Ether) in Ethereum held by the address
- **Storage root:** This field represents the root node of a Merkle Patricia tree that encodes the storage contents of the account.
- **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.

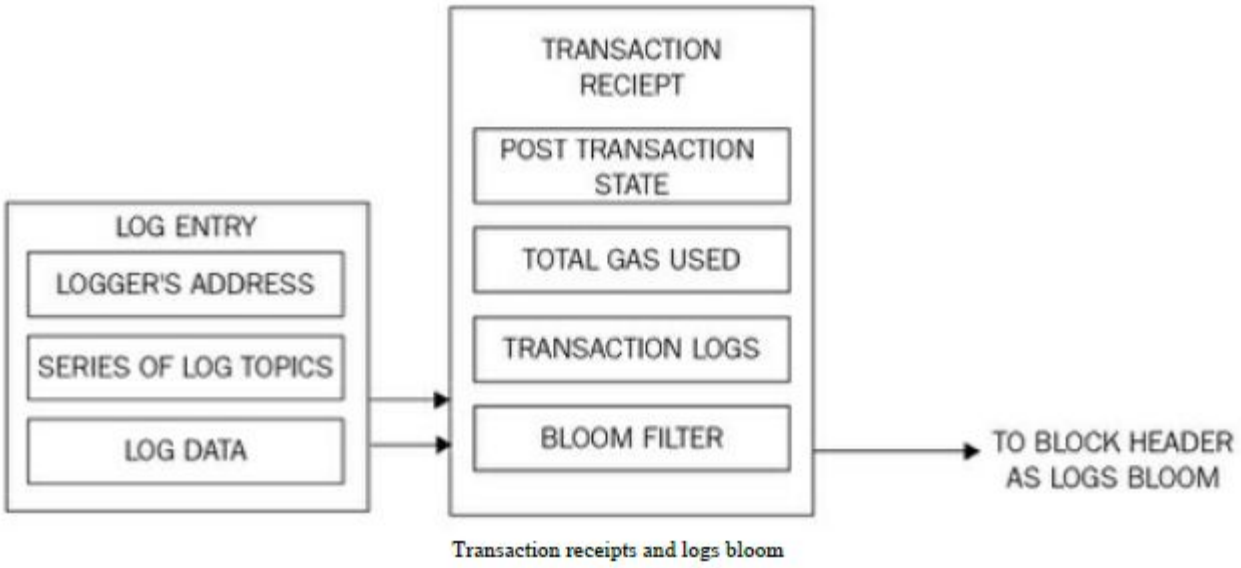


Accounts trie (storage contents of account), account tuple, world state trie, and state root hash and their relationship

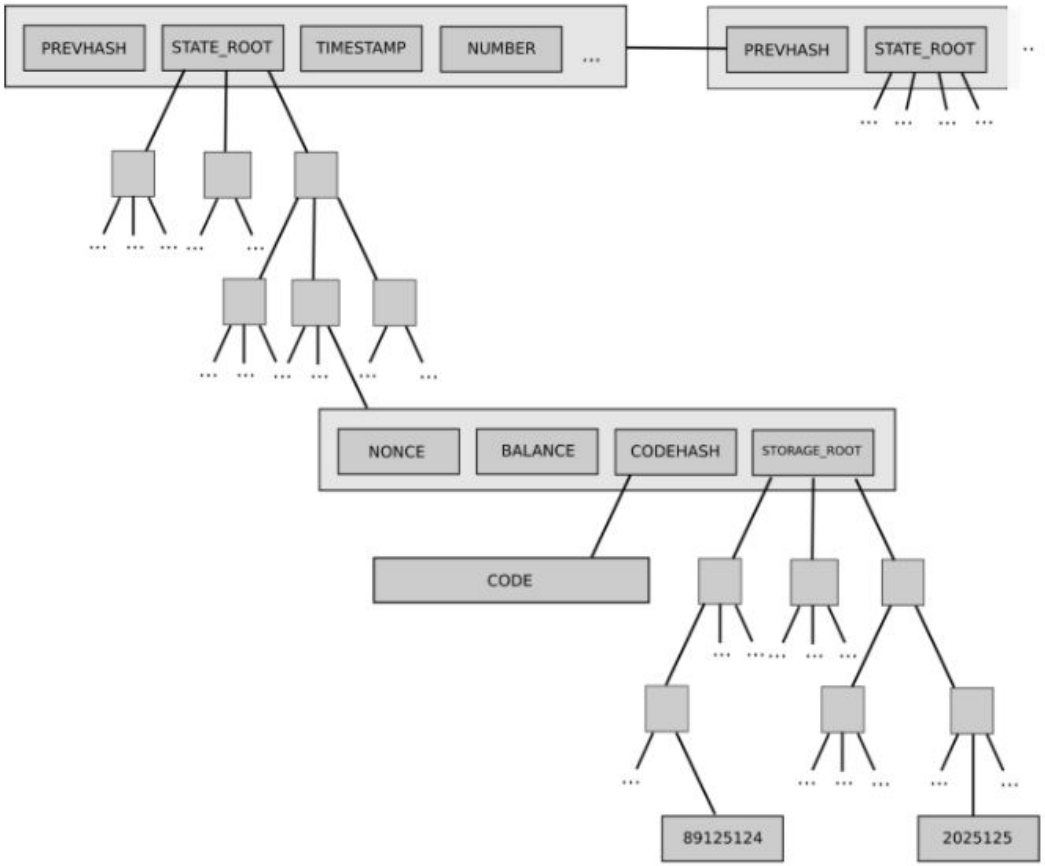
Transaction receipts

- Transaction receipts are used as a mechanism **to store the state** after a transaction has been executed. or 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 (Keccak 256-bit) of the root of this trie is placed in the block header as the receipts root.
- It is composed of four elements that are described here:
- **The post-transaction state:** This item is a trie structure that holds the state after the transaction has been executed. It is **encoded as a byte array.**
- **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.
- **Set of logs:** This field shows the set of log entries created as a result of transaction execution. Log entries contain the logger's address, a series of log topics, and the log data.
- **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. 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**. Log data is made up of a few bytes of data. With the release of Byzantium, an additional field returning the success (1) or failure (0) of the transaction is also available.
- This process of transaction receipt generation can be visualized in the following diagram: Transaction receipts and logs bloom As the result of transaction execution process, the state morphs from an initial state to a target state.
- This state needs to be stored and made available globally in the blockchain

Transaction receipts



Transaction receipts and logs bloom



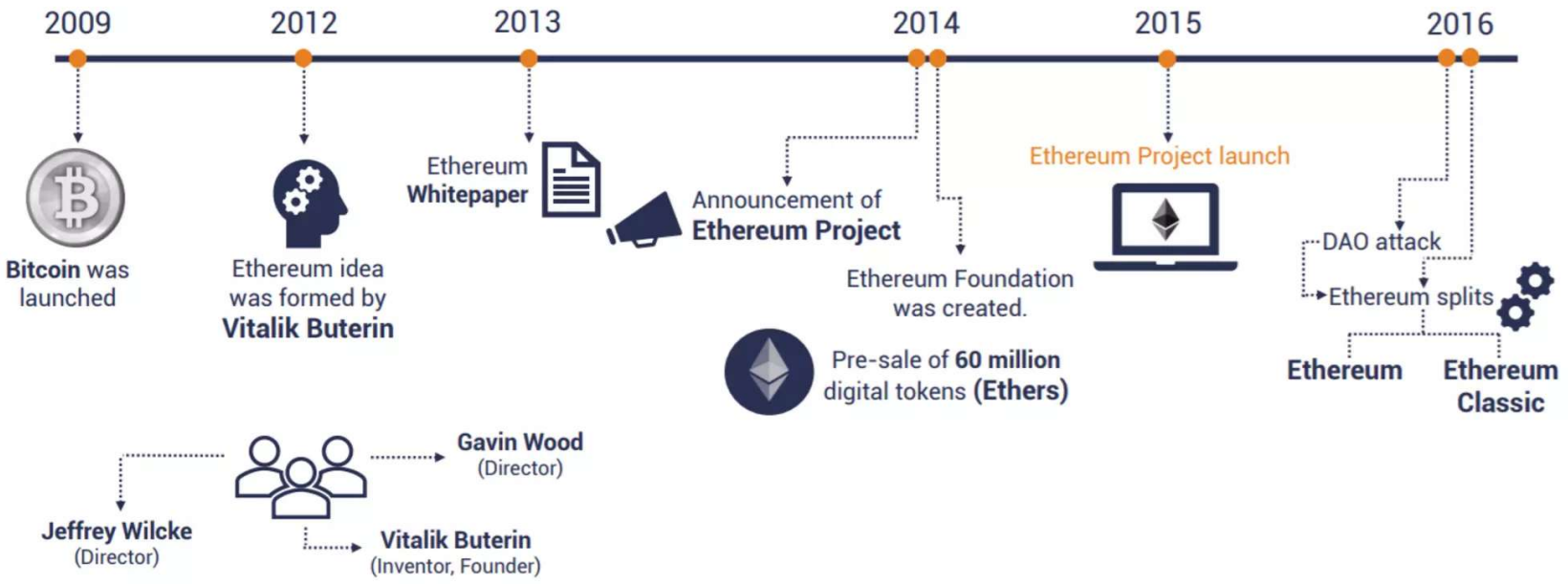
Source: Ethereum whitepaper

Ethereum

Ethereum is a decentralized platform that runs smart contracts: applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference

Ethereum

The birth of Ethereum



Ethereum

What is Ethereum

Ethereum is an open software platform based on blockchain technology that enables anyone to build and deploy decentralised applications (dapps).

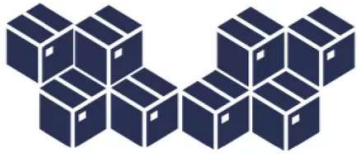
What is Dapp?

Decentralised application (Dapp)

is a new type of software program designed to exist on the internet in a way that is not controlled by any single entity.

Example

Bitcoin's Blockchain



Public ledger showing all Bitcoin transactions

- ✓ Online based
- ✓ Decentralised
- ✓ Not controlled by any single entity

Blockchain is a distributed database that is used to maintain a continuously growing list of records called blocks. = Shared record of the entire transaction history

Centralised vs Decentralised Applications

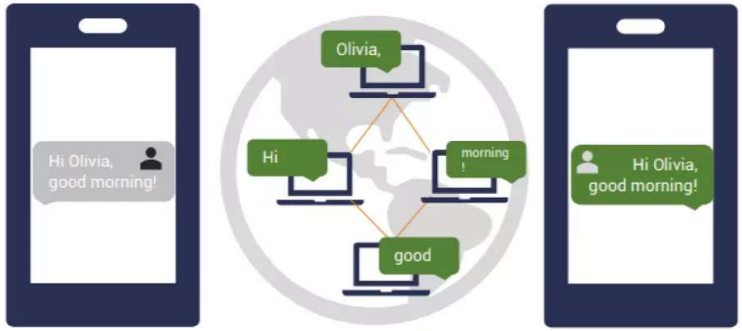
Centralised application

Message sent via messaging app

Message goes through messaging app data server or a centralized network.

Message delivered via messaging app

Decentralised application



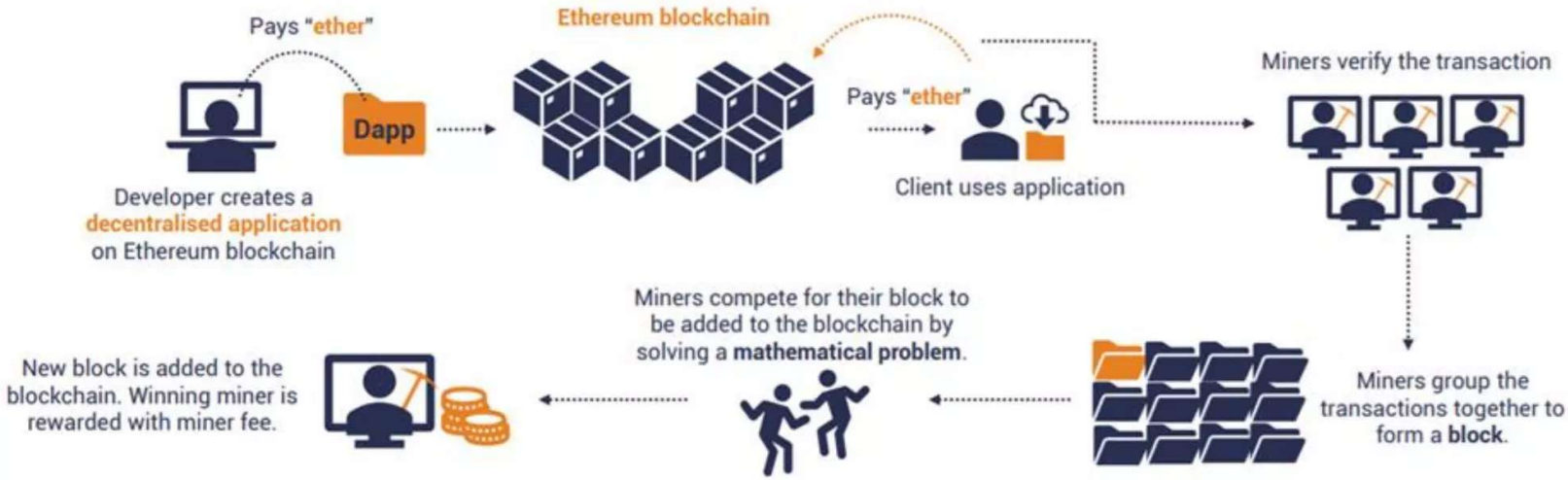
Message sent via messaging dapp

Message goes to independent computers spread all over the world.

Message delivered via messaging dapp

The message that has been sent via dapp goes to independent computers (miners) spread all over the world. To connect to the network, they need to download a program called Mist. No one can access the full message. This is an example of a decentralised network.

How does it work



Ether is Ethereum's digital currency which fuels the Ethereum platform

Miners are people who help secure the Ethereum network and verify all transactions that take place in the blockchain

How are ethers created

2014 pre-sale of 60 million ethers ~US$7.26B

20%
Development fund

80%
Ethereum Foundation contributors

5 ethers
As mining fee
every 15-17 seconds

Ethereum network

- The Ethereum network is a **peer-to-peer network** where nodes participate in order to maintain the blockchain and contribute to the consensus mechanism.
- Networks can be divided into three types, based on requirements and usage.
  - **Mainnet**
  - **Testnet**
  - **Private net**

Ethereum network

- **Main net:** is the current live network of Ethereum.
- The current version of main net is Byzantium (Metropolis) and its chain ID is 1.
- Chain ID is used to identify the network.
- A block explorer which 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 network

- **Testnet :** is also called **Ropsten** and is the widely used test network for the Ethereum blockchain.
- This test blockchain is used to test smart contracts and DApps before being deployed to the production live blockchain.
- Moreover, being a test network, it allows experimentation and research. The main **testnet is called *Ropsten***
- which contains all features of other smaller and special purpose testnets that were created for specific releases.
- For example, other testnets include **Kovan** and **Rinkeby** which were developed for testing **Byzantium** releases.
- Now the Ropsten test network contains all properties of **Kovan and Rinkeby.**

Ethereum network

Private net: As the name suggests, this is the private network that can be created by generating a new genesis block.

This is usually the case in private blockchain distributed ledger networks, **where a private group of entities start their blockchain and use it as a permissioned blockchain.**

This table shows the list of Ethereum network with their network IDs. These network IDs are used to identify the network by Ethereum clients

Network name	Network ID / Chain ID
Ethereum mainnet	1
Morden	2
Ropsten	3
Rinkeby	4
Kovan	42
Ethereum Classic mainnet	61