



Blockchain Honor Degree Sem VII

HBCC 601 : Blockchain Platforms

**Module - 3 : Ethereum Blockchain
(10 Hours)**

Instructor : Mrs. Lifna C S





Topics to be covered



- Introduction to Ethereum (Already covered in Module 1)
- **Ethereum and Its Components:**
 - Mining, Gas, Ethereum, Ether, Ethereum Virtual Machine, Transaction, Accounts.
- **Architecture of Ethereum,**
- Smart Contract (Already covered in Module 1)
 - Remix IDE,
 - Developing smart contracts for Ethereum blockchain,
 - Applications using smart contracts (Already covered in Module 1)
- **Dapp Architecture.**
- Types of test networks used in Ethereum,
- Transferring Ethers Using MetaMask,
- **Mist Wallet (Deprecated)**
- Case study of Ganache for Ethereum blockchain.
- Ethereum Frameworks
- Ethereum 2.0
 - POS (Proof of Stake),
 - Sharding of Chain
 - Concept of Beacon chain.

Self-learning Topics: Study case study on any Ethereum blockchain





Benefits of Ethereum

1. **Availability:**
 - As the Ethereum network is decentralized so **there is no downtime**.
 - **Even if one node goes down other computing nodes are available.**
2. **Privacy:** **Users don't need to enter their personal credentials** while using the network for exchanges, thus **allowing them to remain anonymous.**
3. **Security:** Ethereum is **designed to be unhackable**, as the **hackers have to get control of the majority of the network nodes to exploit the network.**
4. **Less ambiguity:** The **smart contracts that are used as a basis for trade and agreement** on Ethereum **ensure stronger contracts** than traditional contracts which require follow-through and interpretation.
5. **Rapid deployment:** On Ethereum decentralized networks, enterprises can easily deploy and manage private blockchain networks instead of coding blockchain implementation from scratch.
6. **Network size:** Ethereum network **can work with hundreds of nodes and millions of users.**
7. **Data coordination:** Ethereum **decentralized architecture better allocates information** so that the **network participants don't have to rely on a central entity** to manage the system and mediate transactions.



- **Complicated programming language:** Learning solidity from programming smart contracts on Ethereum can be challenging and one of the main concerns is the scarcity of beginner-friendly classes.
- **Volatile cryptocurrency:** Ethereum investing can be risky as the price of Ether is very volatile, resulting in significant gains as well as a significant loss.
- **Low transaction rate:**
 - a. Bitcoin has an average transaction rate of 7TPS
 - b. Ethereum has an average speed of 15 TPS which is almost double that of bitcoin but it is still not enough.





Components of Ethereum Network



1. Ethereum Node
2. Ethereum Client
3. Ether
4. Gas
5. Ethereum Accounts
6. Nonce
7. Storage Root
8. Ehash
9. Transactions
10. Ethereum Virtual Machine (EVM)



Components of Ethereum Network - (1) Ethereum Node

- a computer that is running the software client.
- Nodes communicate with one another in order to validate transactions and record data about the status of the blockchain.
- Responsible for storing, validating, and trading data.
- Each node keeps its own copy of the blockchain and strives to verify that it matches the copies of all the other nodes.
- Every node on the network must process any action that requires a new block to be added to the blockchain.
- This network of continually communicating nodes allows us to avoid relying on a single source of truth and all of the challenges it entails.
- A new block is added based on whether or not the majority of nodes accept it.



Components of Ethereum Network - (1) Ethereum Node (Types)

1. Full Node:

- verify and validate each and every transaction that takes place inside the network
- Maintain the state / a full copy of the blockchain.
- When a smart contract transaction occurs,
 - Full nodes also execute all of the instructions in the smart contract.
 - It determines whether or not the smart contract execution is producing the expected results.
- It keeps receiving copies of the entire blockchain including its transactions which are stored locally and keeps the latest state of transaction with itself.
- Eg: Person A performs a transaction to person B,
 - transaction is added to the blockchain,
 - Full nodes verify whether the transaction complies with all the Ethereum specifications,
 - Maintain the latest state of the blockchain by storing or removing the specification if it does not comply.
- Example of a discarding transaction is when a person transfers X ETH to another person but their account contains less ETH.



2. Archive Nodes:

- complete nodes that have the “archive mode” option enabled.
- While a **Full Node only stores the latest state of the transaction,**
- the Archive nodes hold all of the blockchain’s history data dating back to the genesis block.
- **used when blocks prior to the latest 128 blocks are required.**
- Eg : using functions like **eth_getBalance** of a historic address would require an archive node, as will interacting with smart contracts launched far earlier in the blockchain.
- Archive Nodes memory requirement :
 - **require more than 6 Terabytes of space**, contrary to Full node which only requires a little over 500 Gigabytes of disk space.
 - **are not useful for average people,**
 - they are effective in the application of block exploring, wallet vending, and chain analytics.



3. Light Nodes:

- does not hold the complete current blockchain state
- stores only the block header.
- suitable for low memory and computational devices since maintaining a light node involve the least investment in hardware, running costs, and technical skill.
- Light nodes rely on full nodes to function.
- These nodes do not need to run continuously or read and publish a large amount of data on the blockchain.
- It provides an easy way to create a wallet, especially for beginners.
- Eg : **Solid-State Drives (SSD)** cannot afford to store the gigabytes of data that other nodes take.
- But there are some limitations of light nodes which cannot be denied, there is no guarantee that the light wallet provider will be online when it is needed.



Components of Ethereum Network - (2) Ethereum Client

- **software program** that is used to **implement the Ethereum specification**
- **connect itself with other Ethereum clients** over a peer-to-peer network.
- Different Ethereum clients can communicate with one another if they follow the reference specification and the defined communication protocols.
- These interactions among different clients in the network take place using various programming languages
 - like Geth (Go), OpenEthereum (Rust),
 - Nethermind (C#, .NET), Besu (Java),
 - Erigon (Go/Multi).
- The yellow paper is the Ethereum protocol that allows anybody to run a client to construct a node.
- **Ethereum sets standard behaviors that all Ethereum clients must adhere to**
- Ethereum's specs enabled the blockchain to allow for different, but interoperable, software implementations of an Ethereum client by providing standard documentation and simple language.



Components of Ethereum Network - (2) Ethereum Client (Types)

1. Full Client:

- **save the complete Ethereum blockchain,**
- which **might take several days to synchronize** and
- takes a **massive amount of disc space** – more than 1 Terabyte, according to the most recent estimates.
- **Enable connected nodes to conduct all network functions**, including as
 - mining,
 - transaction
 - block-header validation,
 - smart contract execution.



Components of Ethereum Network - (2) Ethereum Client (Types)

2. Light Client:

- do not always need to necessarily keep all of the data,
- when data storage and performance are concerns, developers utilize the “light clients”.
- Light clients provide a portion of full client capability.
- they can provide quick delivery and free up data storage space.
- The functionality of a light client is adapted to the purposes of the Ethereum client.
- **widely used within wallets to maintain private keys and Ethereum addresses.**
- manage smart contract interactions and transaction broadcasts.
- useful for web3 instances within JavaScript objects, Dapp browsers
- **obtaining the exchange rate data.**



Components of Ethereum Network - (2) Ethereum Client (Types)

3. Remote Client:

- A remote client is much like a light client.
- The primary distinction is that a remote client does not keep its own copy of the blockchain or validate transactions or block headers.
- Remote clients, on the other hand, rely entirely on a full or light client to have access to the Ethereum blockchain network.
- These clients are mostly used as wallets for transmitting and receiving transactions.



Components of Ethereum Network - Node Vs Client

Ethereum Node	Ethereum Client
A machine running Ethereum client software is referred to as an “Ethereum Node”	A client is an Ethereum implementation that validates all transactions in each block, ensuring the network’s security and data accuracy
The three types of Ethereum Nodes are Full, Light, Archive, and Miner Nodes.	The three types of Ethereum Clients are Full, Light, and Remote Clients
The Ethereum node operating system allows us to access the internet	The Ethereum client computer allows a user to access the node operating system



Components of Ethereum Network - Node Vs Client

FULL CLIENT



Enables the nodes to **perform all major network operations**.

LIGHT CLIENT



Performs a **subset of network operations**.
Can **validate block headers**.
Can **use Merkle proofs to verify transaction inclusion**.

REMOTE CLIENT



Relyes on **other types of clients** for network access.
Usually **offers wallet functionalities**.

DIFFERENT TYPES OF ETHEREUM NODES

are **computational devices** that participate in the Ethereum network

ETHEREUM CLIENTS

are **software applications** that implements the ethereum specifications

FULL NODE



Stores the **complete Blockchain**. Can **validate and verify** blocks.

ARCHIVE NODE



Stores the **complete Blockchain**. Stores the **blockchain state** at each block level.

LIGHT NODE



Stores the **block headers**. Can participate in **data validation**.



Components of Ethereum Network -

	Pros	Cons
Light nodes	<ul style="list-style-type: none">• Portable• Resource-efficient• User-friendly	<ul style="list-style-type: none">• Do not validate the network• Do not propagate blocks• Do not maintain consensus• Less secure
Full nodes	<ul style="list-style-type: none">• Validate the network• Propagate blocks• Maintain consensus• More secure	<ul style="list-style-type: none">• Resource-heavy• Harder to maintain• Less user-friendly
Pruned	Flexible storage	Need to revalidate old blocks
Archive	Carry full history	Resource and storage heavy
Mining	<ul style="list-style-type: none">• Easily trackable involvement• Can pool with others to increase reward rate	<ul style="list-style-type: none">• High and wasteful energy consumption• High equipment cost and barrier to entry
Staking	<ul style="list-style-type: none">• Low barrier to entry• Low energy consumption	<ul style="list-style-type: none">• Reward system based on luck• Low transparency in staking pools
Masternodes	<ul style="list-style-type: none">• Balanced network benefits and rewards• Lower maintenance costs	<ul style="list-style-type: none">• High initial investment• Difficult setup process





Components of Ethereum Network - (3) Ether

- type of **cryptocurrency used in the Ethereum network**
- It is a peer-to-peer currency
- It **tracks and promotes each transaction** in the network.
- It is the second-largest cryptocurrency in the world.
- Other cryptocurrencies can be used to get ether tokens, but vice versa is not true.
 - It means that **ether tokens can't be interchanged by other cryptocurrencies** to render computing power for Ethereum transactions.
- paid as a commission for any execution that affects the state in Ethereum.
- used in the Ethereum algorithm as an incentive for miners to blocks to the blockchain using PoW
- only currency that can be used to pay transaction costs, which go to miners as well.
- Aside from paying for transactions, **ether is often used to purchase gas**, which is used to pay for the computation of any transaction on the Ethereum network.



Components of Ethereum Network - (4) Gas



- **An internal currency** of the Ethereum network.
- We need gas **to run applications on the Ethereum network**, much as we need gas to run a vehicle.
- To complete every transaction on the Ethereum network, a consumer must first make a payment—send out ethers—and **the intermediate monetary value is known as gas**.
- Gas is a unit of measurement on the Ethereum network **for the computing power used to execute a smart contract or a transaction**.
- The price of gas is **very low compared to Ether**.
- **The execution and resource utilization costs are predetermined in Ethereum** in terms of Gas units, called **gwei**.





Components of Ethereum Network - Ether Denominations

Value (in wei)	Exponent	Common Name	SI Name
1	10^0	wei	wei
1,000	10^3	babbage	kilowei or femtoether
1,000,000	10^6	lovelace	megawei or picoether
1,000,000,000	10^9	shannon	gigawei or nanoether
1,000,000,000,000	10^{12}	szabo	microether or micro
1,000,000,000,000,000	10^{15}	finney	milliether or milli
1,000,000,000,000,000,000	10^{18}	ether	ether
1,000,000,000,000,000,000,000	10^{21}	grand	kiloether
1,000,000,000,000,000,000,000,000	10^{24}		megaether



Components of Ethereum Network - (5) Ethereum Accounts

- similar to a bank account,
- but for ethers or ETH, where Ethereum can be held, transferred to other accounts.
- can also be used to execute smart contracts.
- An entity that is composed of **an Ethereum address along with a private key.**
 - The first 20 bytes of the SHA3 hashed public key is the Ethereum address.
- Ethereum has two types of accounts:
 - **Externally owned account (EOA):**
 - **Contract Account:**



Ethereum has two types of accounts:

1. **Externally owned account (EOA):**

- controlled by private keys.
- Each EOA **has a public-private key pair.**
- The **users can send messages by creating and signing transactions.**

Advantages of EOA

1. Transactions from an external account to a contract account **can trigger code that can execute many different actions.** such as transferring tokens or even creating a new contract.
2. Externally Owned Accounts **cannot list incoming transactions.**



2. Contract Account:

- **controlled by contract codes.**
- These **codes are stored with the account.**
- Each contract account **has an ether balance associated with it.**
- The contract code of these accounts **gets activated every time a transaction from an EOA or a message from another contract is received by it.**
- When the contract code activates, **it allows to read/write the message to the local storage, send messages and create contracts.**
- **Types of Contract Accounts :**
 - i. **Simple Account:** The account is created and owned by a single account holder.
 - ii. **Multisig (multisignature) Account:** A Multisig Wallet contains several owner Accounts, one of which is also the creator Account.



Advantages of CA:

1. A contract account **can list incoming transactions**.
2. Contract accounts **can be set up as Multisig Accounts**.
3. A Multisig Account can be structured such that it has a daily limit that you specify, and **only if the daily limit is exceeded will multiple signatures be required**.

Disadvantages of CA:

1. Creating **contract accounts costs gas** because they use the valuable computational and storage resource of the network.
2. 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 either from an externally owned account or from another contract account.



Components of Ethereum Network - (5) Ethereum Account (EOA Vs CA)

Externally Owned Accounts	Contract Account
Controlled by third party	Controlled by Contract Code
Private Key is needed to access EOAs	No key is needed to access Contract Accounts
EOAs are created automatically on creating wallet	CA require EOAs to be activated
EOA doesn't have a code associated with it	CA have their own associated code
No execution fee is associated with EOAs	Execution fee is associated with CAs
Code hash = empty string	Code hash represents the code associated with the account

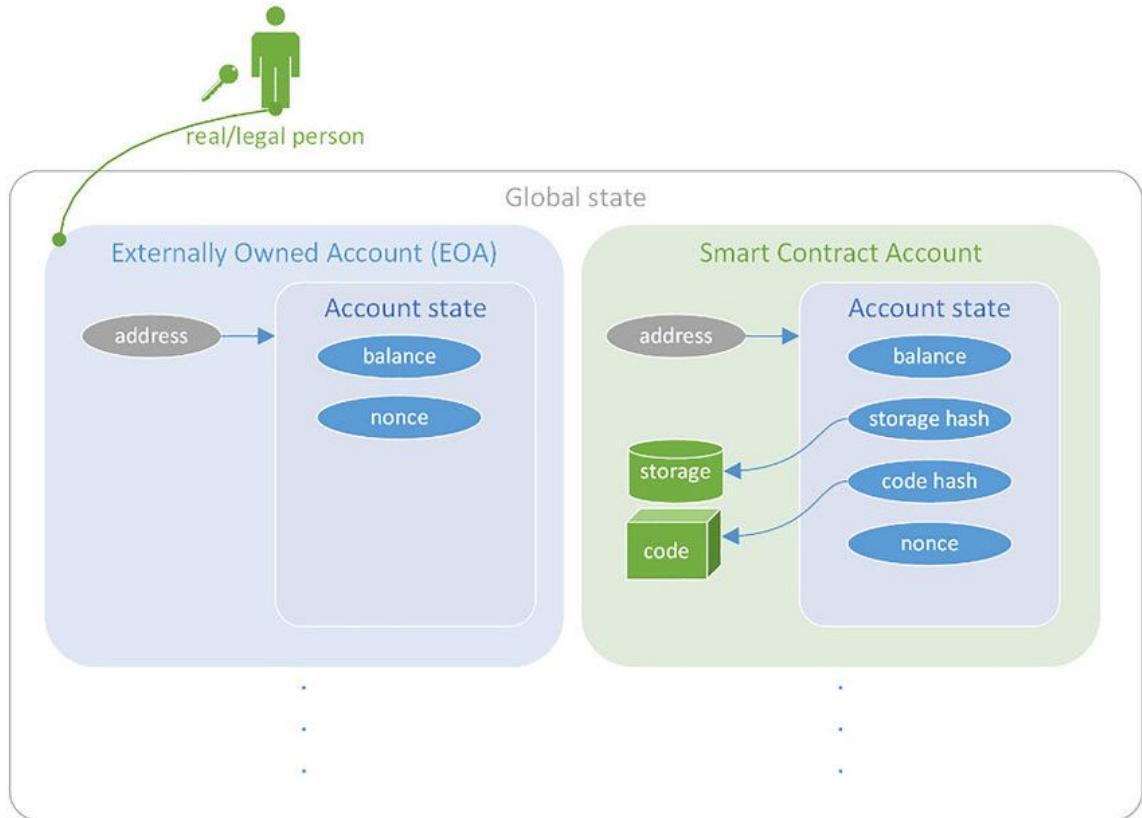


Different Fields in Ethereum Accounts

1. **Nonce:**
 - The nonce in an Ethereum account indicated the number of transactions that have been sent from that account.
 - This ensures that each transaction is made only once by taking count every time it takes place.
2. **Ether Balance:** the amount of ether present in an ether repository of the current ether account.
3. **Contract Code:**
 - This is non-mandatory to fill, in case it is present since not all accounts have a contract code.
 - But note, that they cannot be altered once executed.
4. **Storage:** This field remains not filled unless mentioned.
5. **Code Hash:**
 - hash that refers to the code present in that Ethereum account
 - since no code is associated with externally owned Ethereum accounts, ⇒ **code hash = empty string.**



Components of Ethereum Network - (5) Ethereum Account (Field)



An Ethereum account is a **private-public key pair** that may be linked to a **blockchain address**.

Private Key

- It is a “owned” or “externally owned” account if the **private key is known and controlled by someone**.
- **Contract accounts** do not have a private key connected with them, although EOAs have.
- Control and access to one’s assets and contracts are granted through the EOA private key.
- The user keeps the private key safe.

Public Key

- Account’s **public key is open**.
- This key serves as the account’s identity.
- A one-way cryptographic function is used to produce the public key from the private key.

For example, if you create an account on Ethereum,

- **Retain the private key**
- **Share the public key**. As transactions between accounts are completed using public keys.



6. Nonce

- For **EOAs**, nonce means the number of transactions via this account.
- For **CA**, nonce means the number of contracts generated via this account.

7. Storage Root

- It is the **main root node of a Merkle tree**.
- **Hash of all details of the account** is stored here.
- The **root of the Merkle tree** is the verification of all transactions.

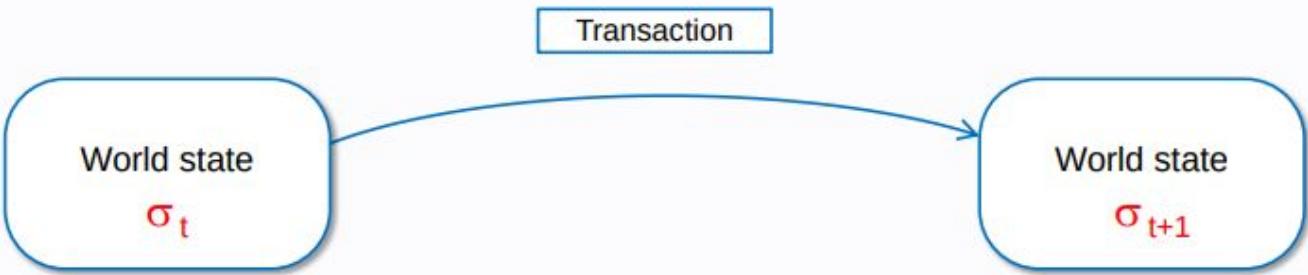
8. Ehash

- **PoW algorithm for Ethereum 1.0**
- most recent version of Dagger-Hashimoto



Components of Ethereum Network - Transactions

A transaction-based state machine



Ethereum can be viewed as a transaction-based state machine.



Components of Ethereum Network - Transactions

A transaction-based state machine

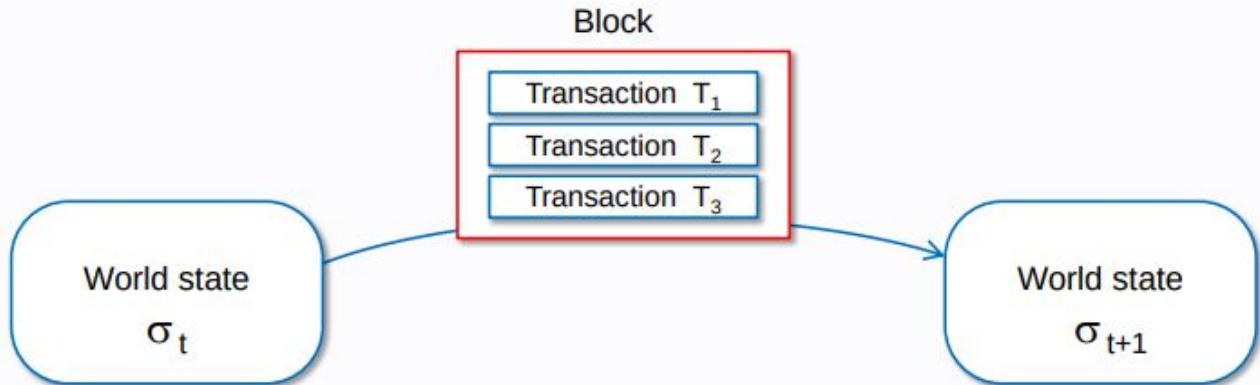


A transaction represents a valid arc between two states.



Components of Ethereum Network - Transactions

Block and transactions

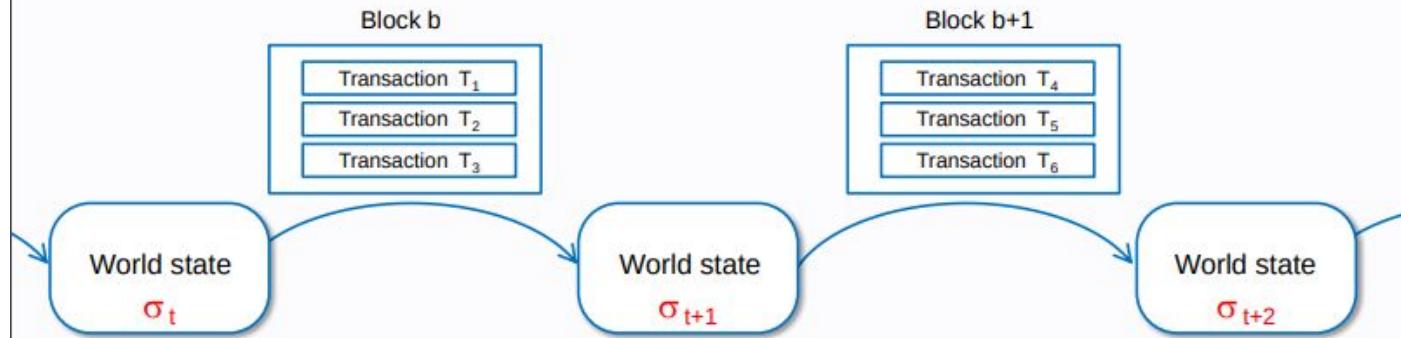


Transactions are collated into blocks.
A block is a package of data.



Components of Ethereum Network - Transactions

Chain of states

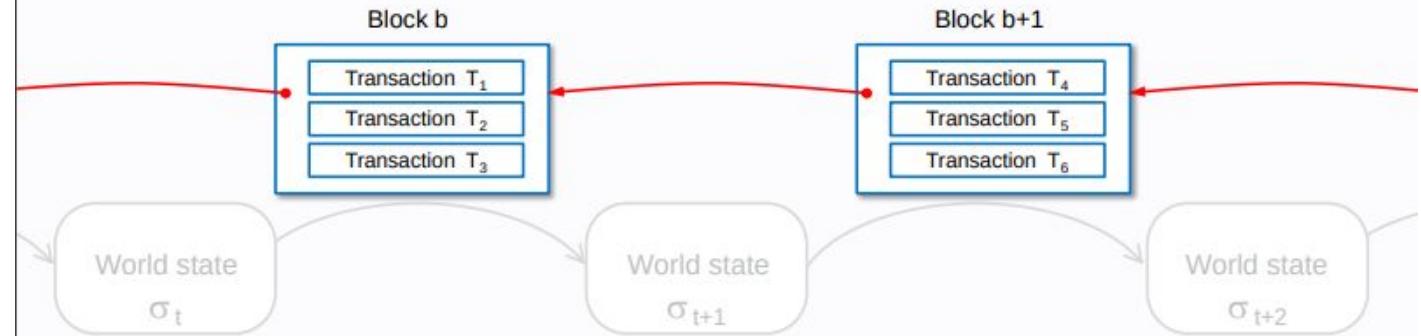


From the viewpoint of the states,
Ethereum can be seen as a state chain.



Components of Ethereum Network - Transactions

Chain of blocks: Blockchain

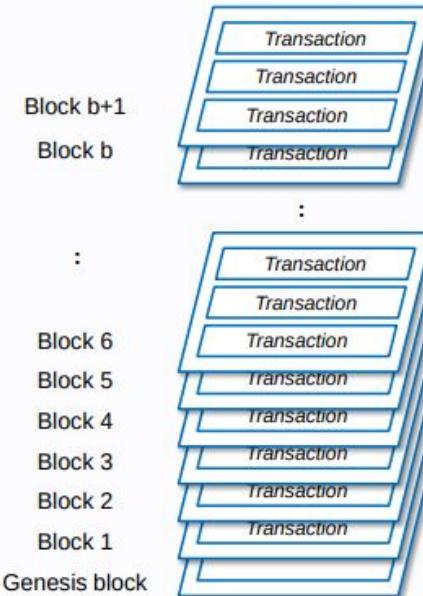


From the viewpoint of the implementation,
Ethereum can also be seen as a chain of blocks, so it is `BLOCKCHAIN`.



Components of Ethereum Network - Transactions

Stack of transactions : Ledger



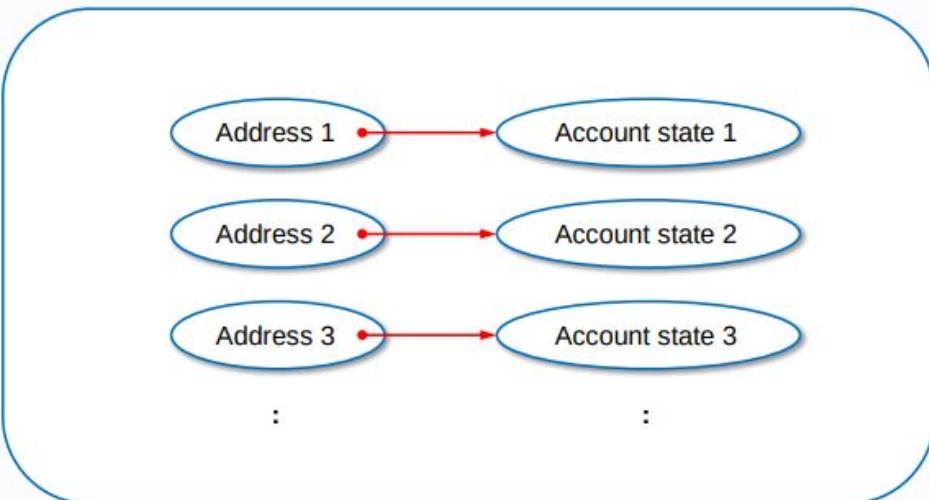
From the viewpoint of the ledger,
Ethereum can also be seen as a stack of transactions.



Components of Ethereum Network - Transactions

World state

World state σ_t



The world state is a mapping between address and account state.



Components of Ethereum Network - Transactions

Several views of world state

Mapping view

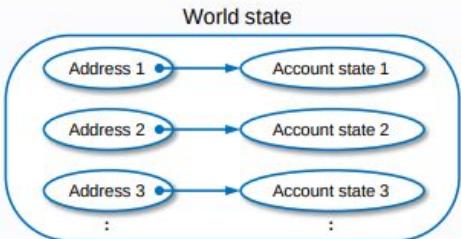
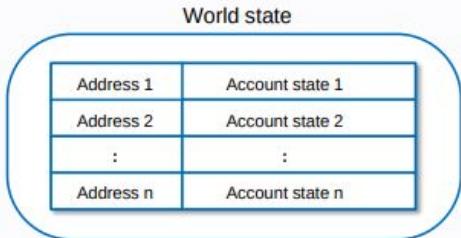
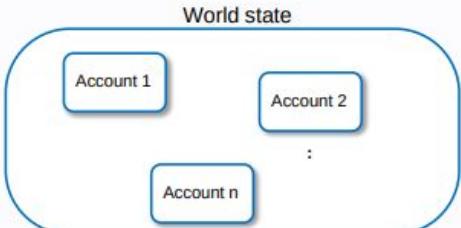


Table view



Object view



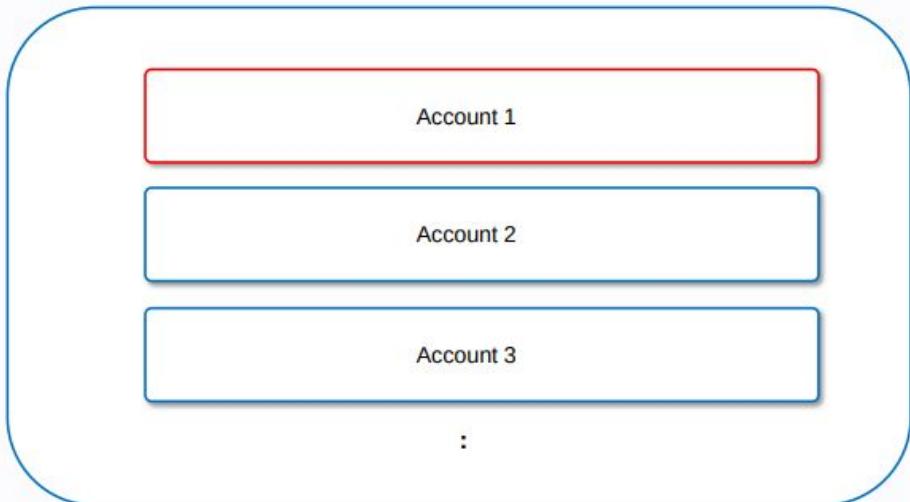
QUESTION



Components of Ethereum Network - Transactions

Account

World state



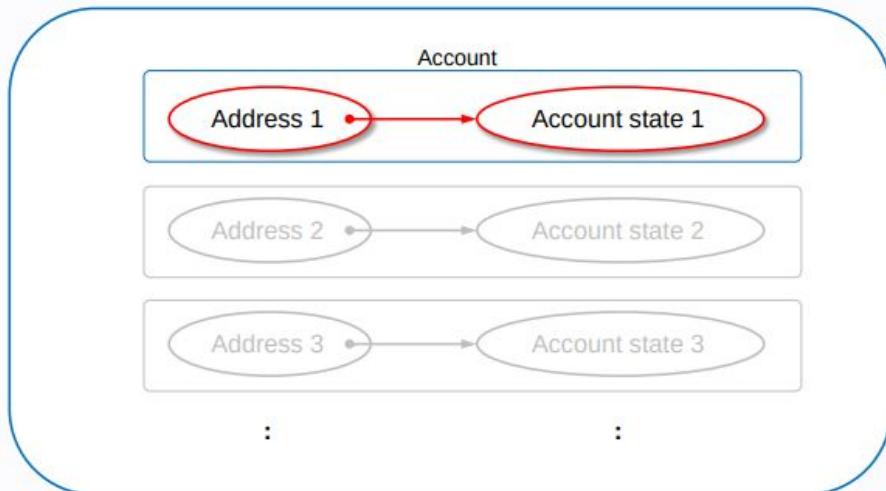
An account is an object in the world state.



Components of Ethereum Network - Transactions

Account

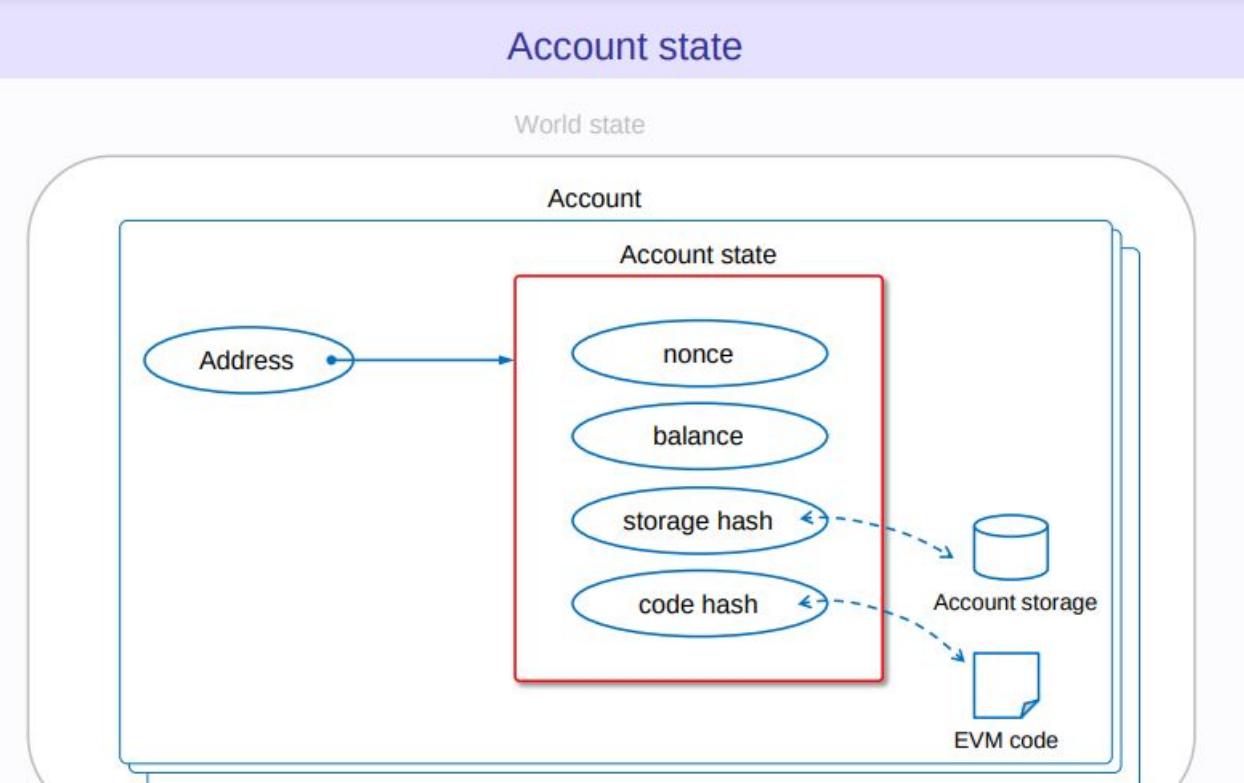
World state



An account is a mapping between address and account state.



Components of Ethereum Network - Transactions



An account state could contain EVM code and storage.

QUESTION



Components of Ethereum Network - Transactions

Two practical types of account



EOA is controlled by a private key.

Contract account contains EVM code.



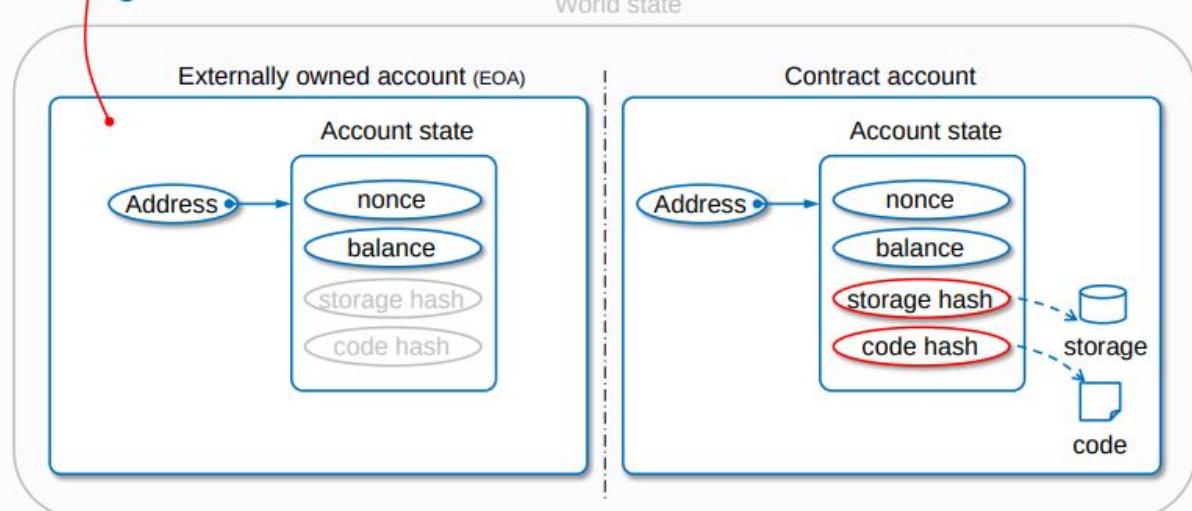
Components of Ethereum Network - Transactions

Two practical types of account

External actor



World state

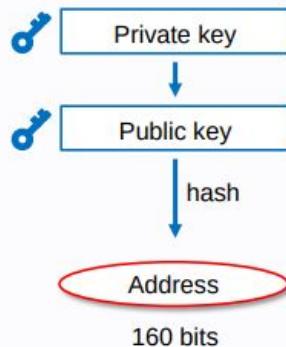


EOA is controlled by a private key.
EOA cannot contain EVM code.

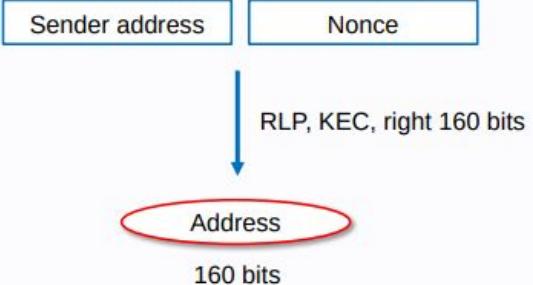
Contract contains EVM code.
Contract is controlled by EVM code.

Address of account

Externally owned account (EOA)



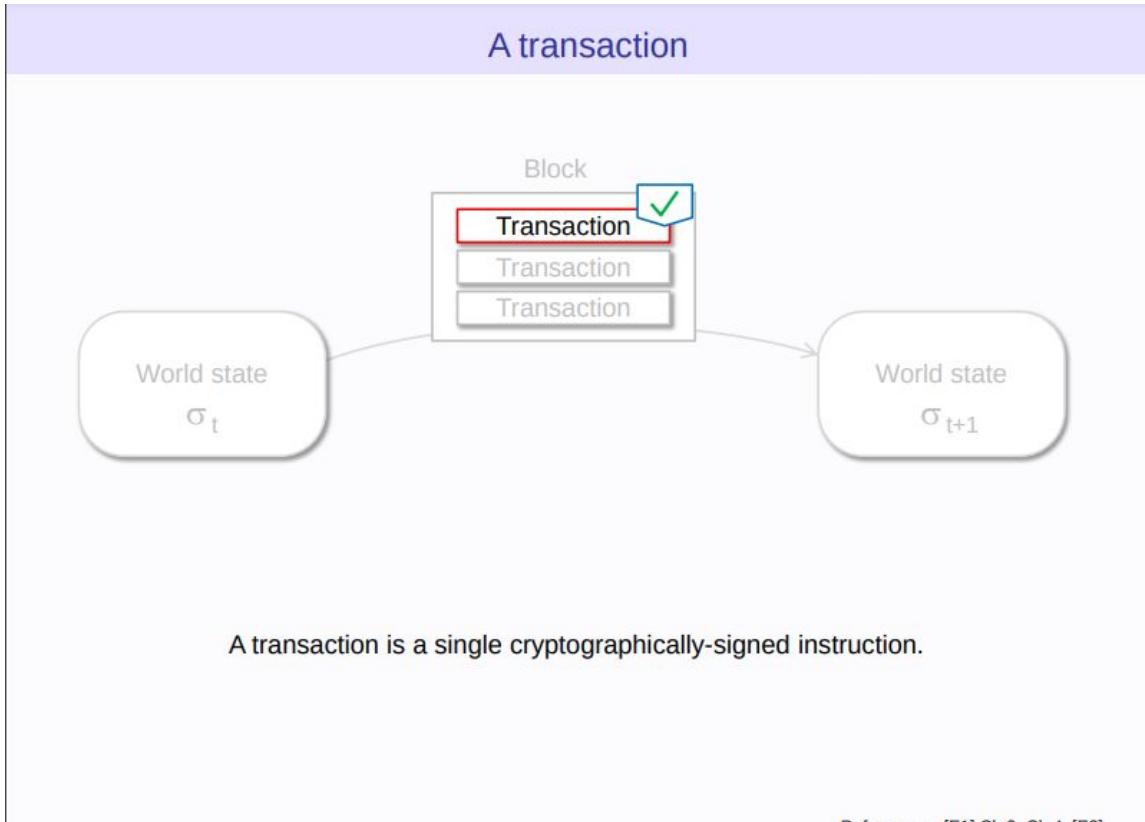
Contract account



A 160-bit code used for identifying accounts.



Components of Ethereum Network - Transactions

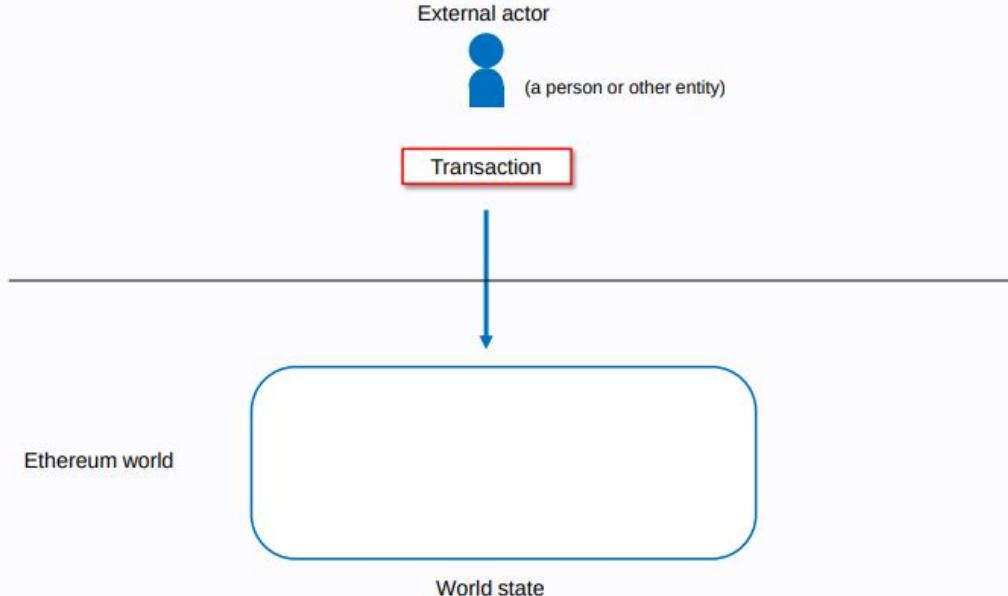


BLOCKCHAIN TECHNOLOGY



Components of Ethereum Network - Transactions

A transaction to world state



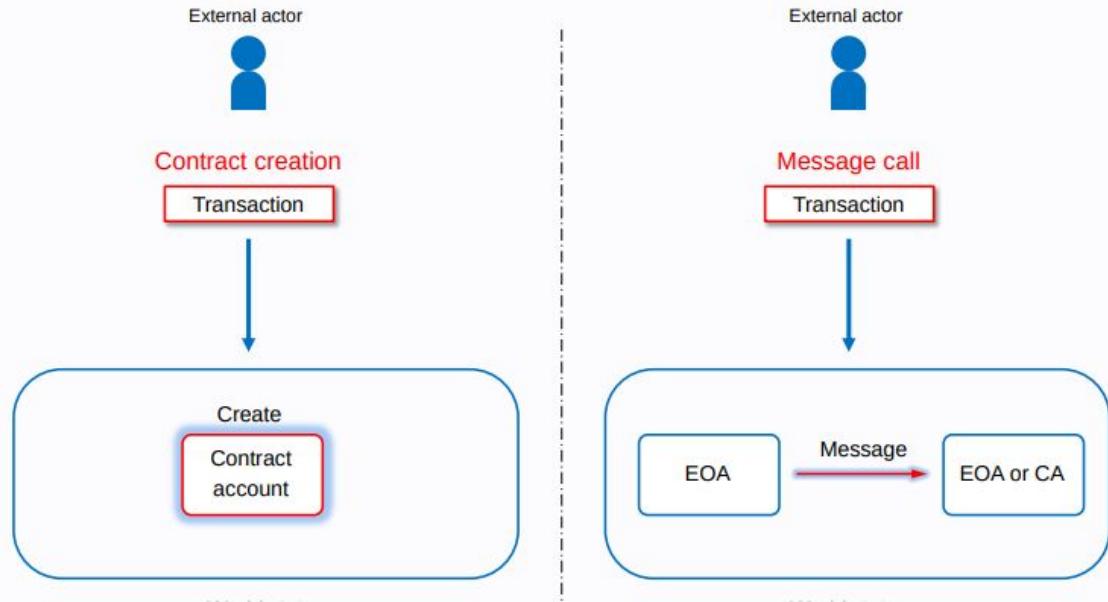
A transaction is submitted by external actor.

https://ethresear.ch/tutorials/understanding-ethereum-transaction-world-state/



Components of Ethereum Network - Transactions

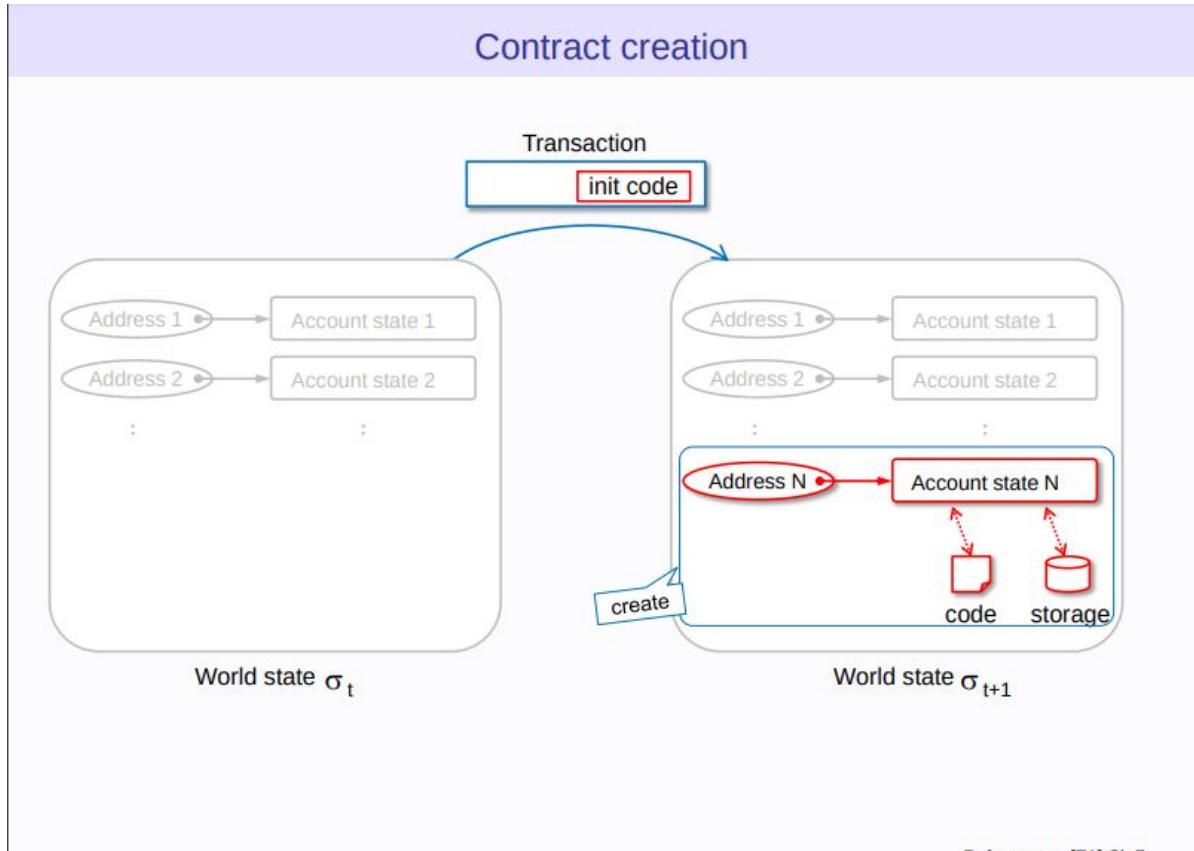
Two practical types of transaction



There are two practical types of transaction, contract creation and message call.

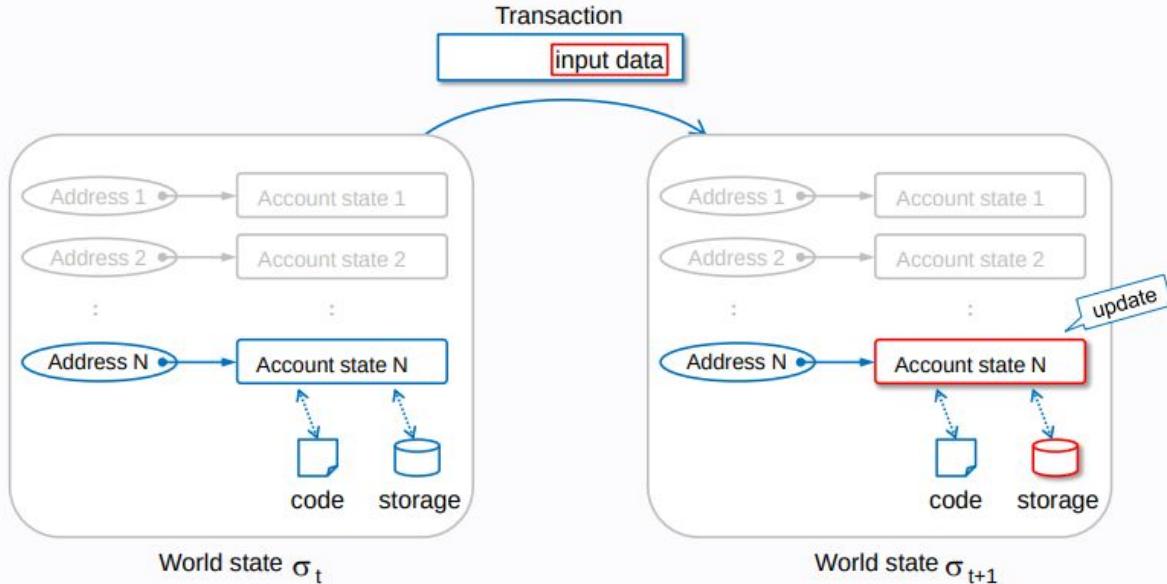


Components of Ethereum Network - Transactions



Components of Ethereum Network - Transactions

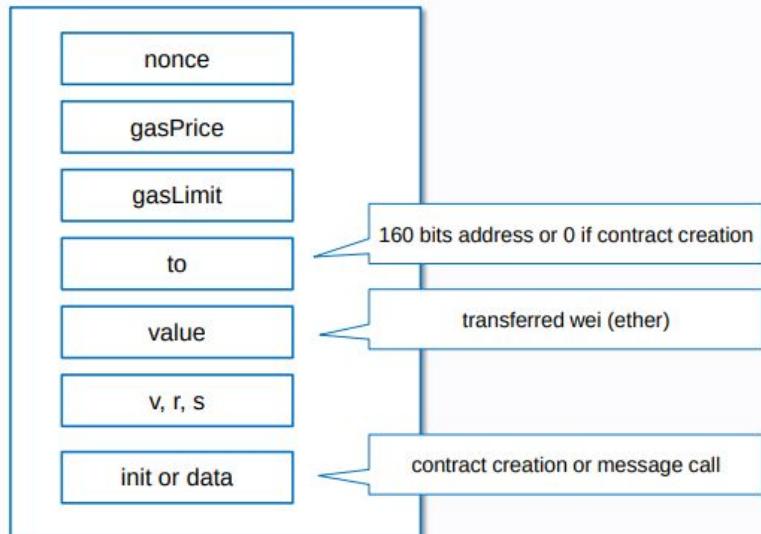
Message call



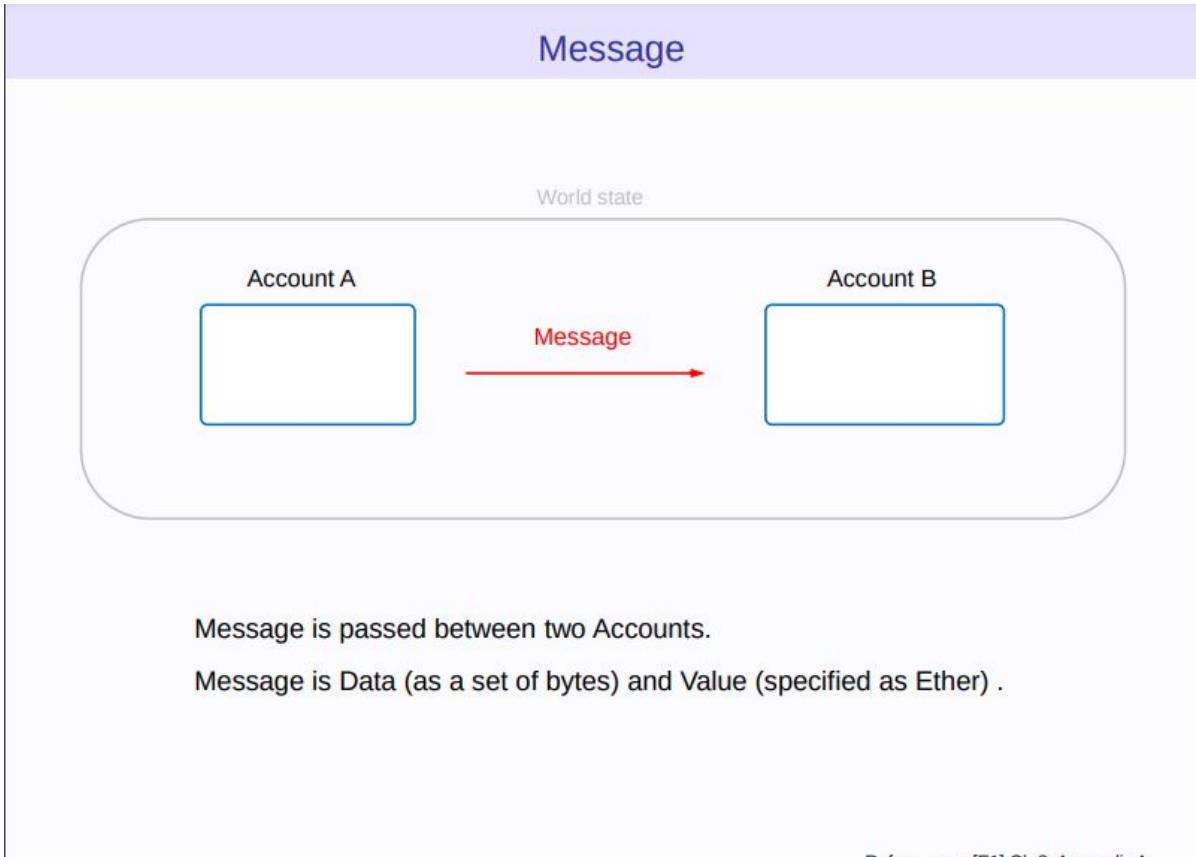
Components of Ethereum Network - Transactions

Field of a transaction

Transaction



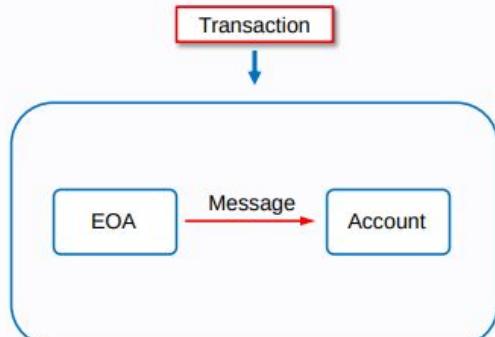
Components of Ethereum Network - Transactions



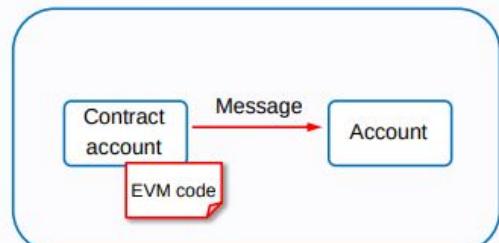
Components of Ethereum Network - Transactions

Message

Triggered by transaction



Triggered by EVM code



Transaction triggers an associated message.

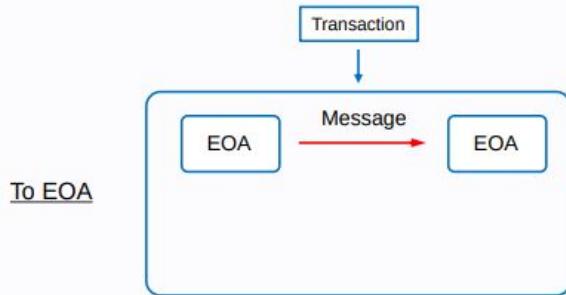
EVM can also send a message.



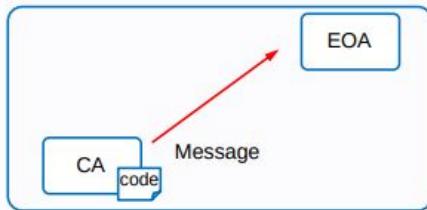
Components of Ethereum Network - Transactions

Four cases of message

By Transaction From EOA



By EVM code From CA



To EOA

To CA

Transaction

Message

EOA

CA

EOA

Message

Message

CA

Transaction

Message

EOA

CA

Message

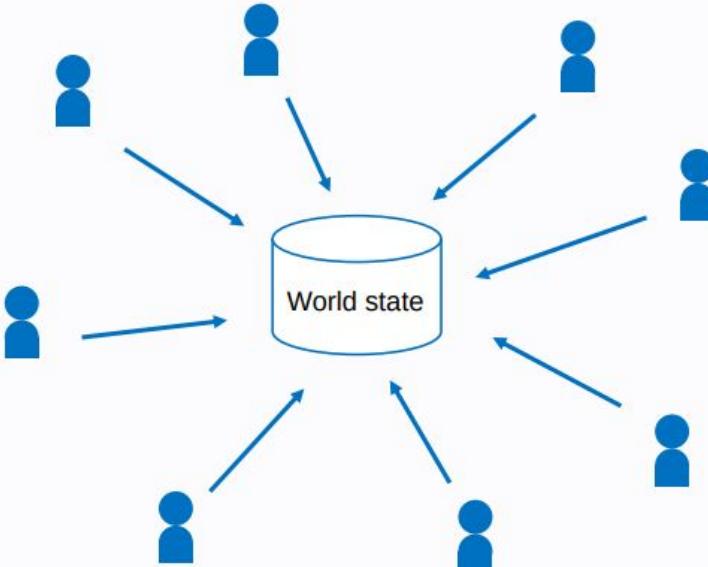
CA

QUESTION PAPER



Components of Ethereum Network - Transactions

Globally shared, transactional database



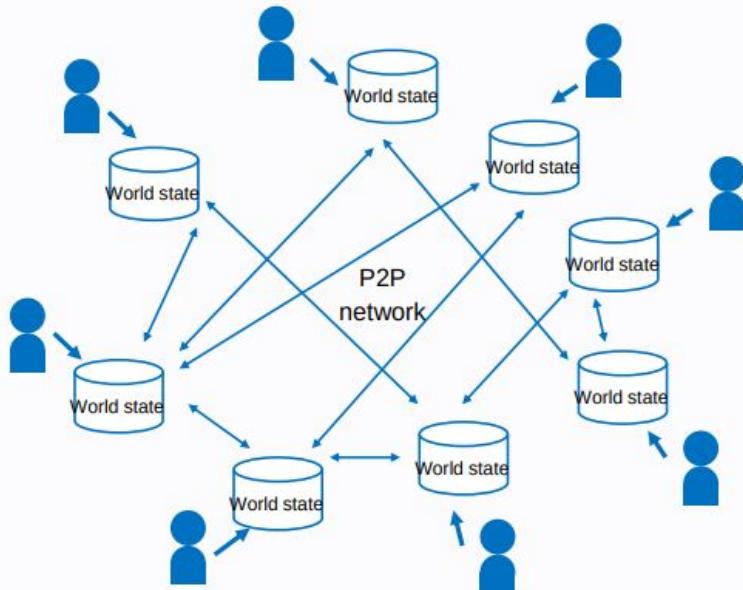
A blockchain is a globally shared, transactional database.

POLYCENTRIC BLOCKCHAIN



Components of Ethereum Network - Transactions

Decentralised database



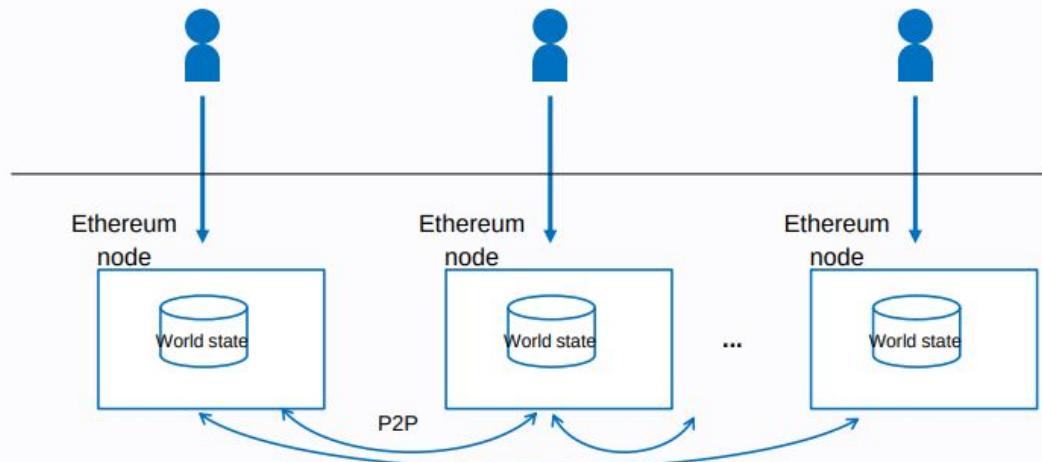
A blockchain is a globally shared, **decentralised**, transactional database.

Blockchain Development



Components of Ethereum Network - Transactions

P2P network inter nodes

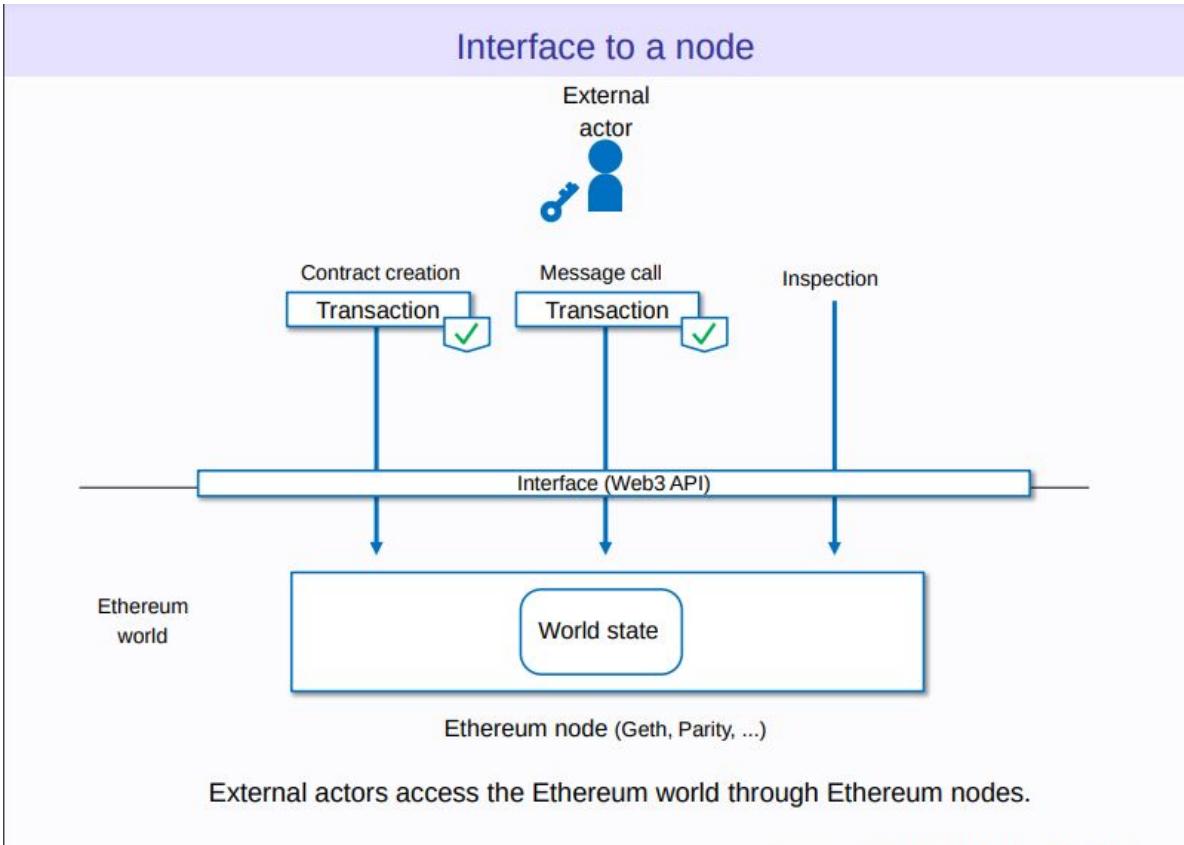


Decentralised nodes constitute Ethereum P2P network.

References : [REDACTED]



Components of Ethereum Network - Transactions



References : EIP-1 Appendix A, Ch 4, Ch 7, Ch 8



Components of Ethereum Network - Transactions

Atomicity of transaction



A transaction is **an atomic operation**. Can't divide or interrupt.

Transaction

or

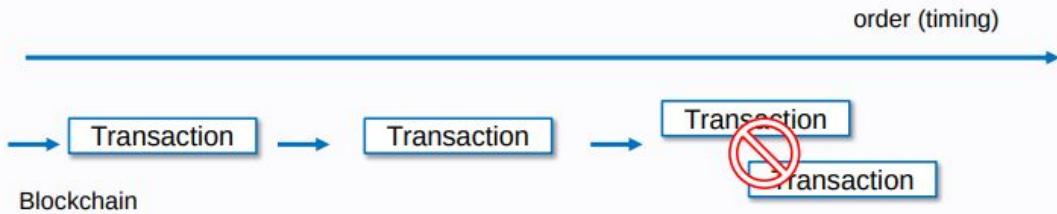
Transaction

That is, **All** (complete done) or **Nothing** (zero effect).



Components of Ethereum Network - Transactions

Order of transactions



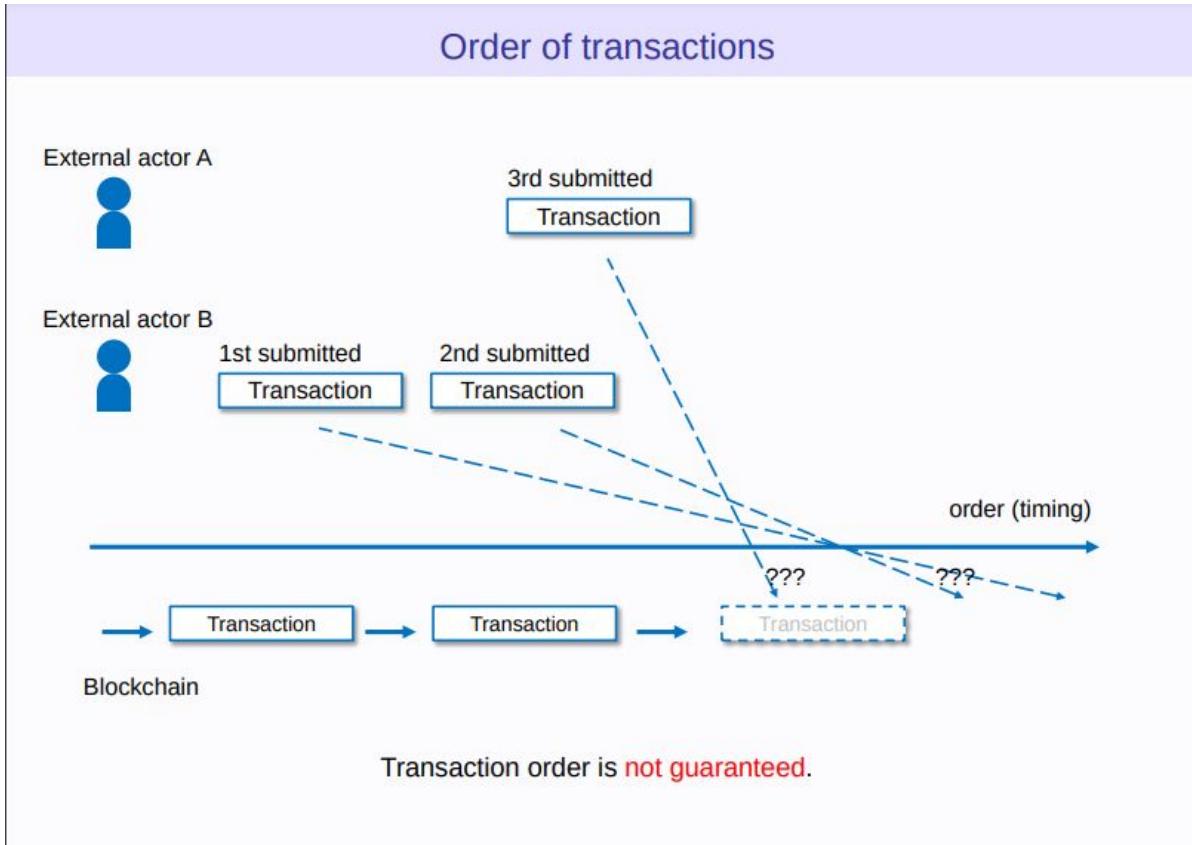
Blockchain

Transactions **cannot be overlapped**.

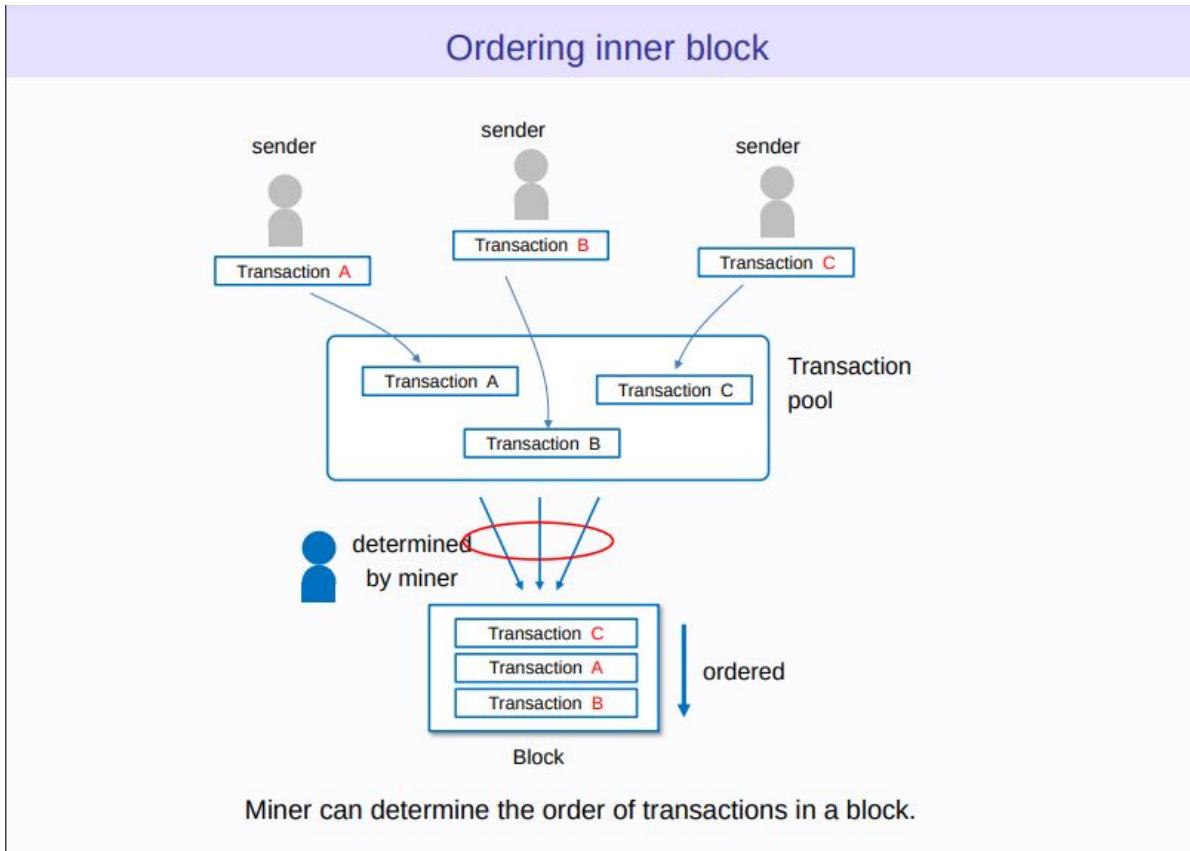
Transactions must be executed sequentially.



Components of Ethereum Network - Transactions

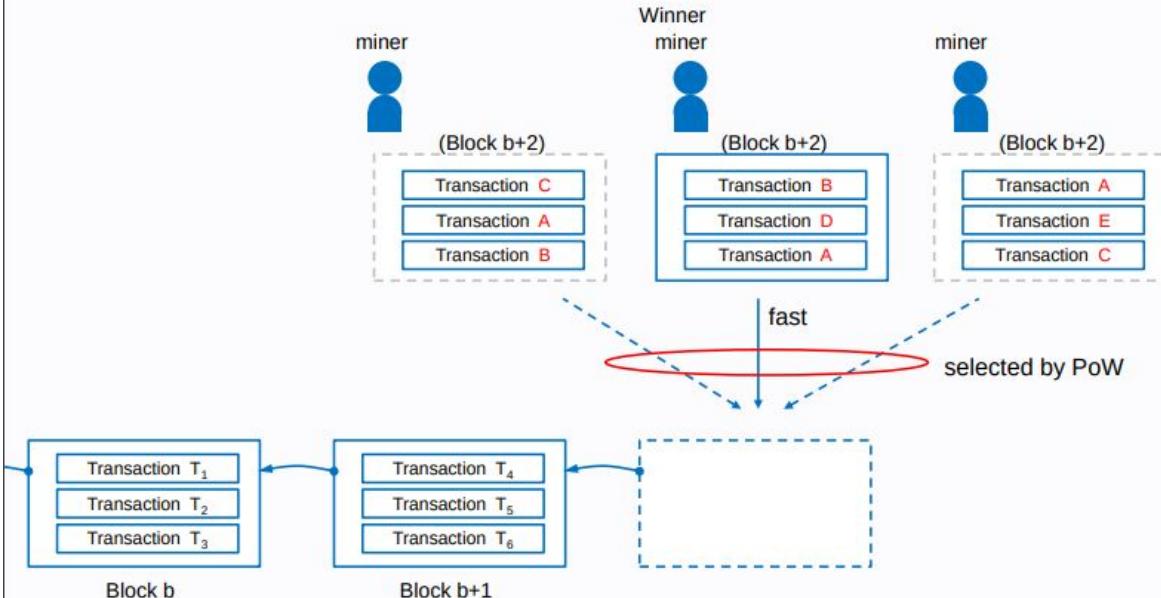


Components of Ethereum Network - Transactions



Components of Ethereum Network - Transactions

Ordering inter blocks

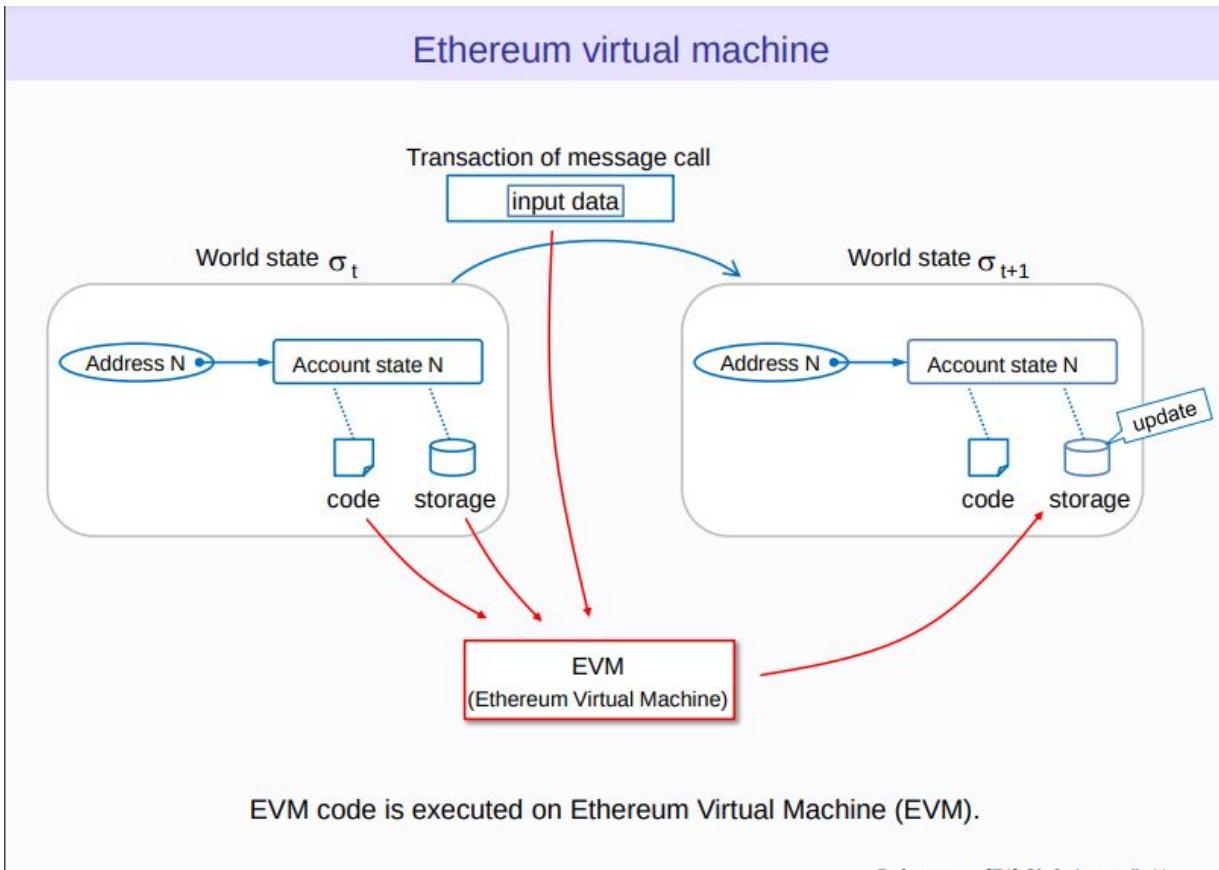


The order between blocks is determined by a consensus algorithm such as PoW.

References : EP11 Ch 3, Ch 4



Components of Ethereum Network - Transactions & EVM





Components of Ethereum Network - EVM

Ethereum virtual machine

Code

EVM code

Virtual machine

EVM (Ethereum Virtual Machine)

The Ethereum Virtual Machine is the runtime environment for smart contracts in Ethereum.

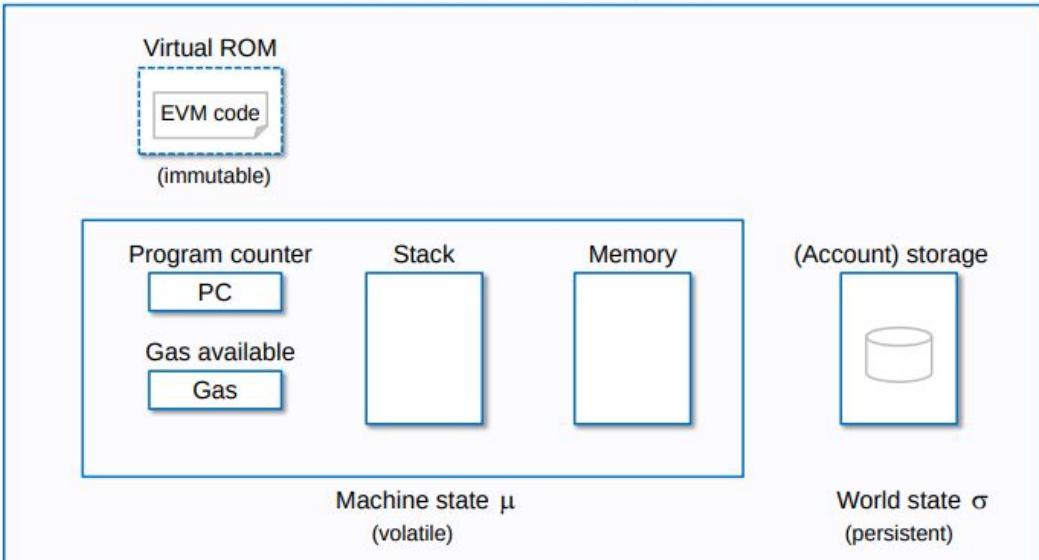
DOCUMENTATION & SUPPORT



Components of Ethereum Network - EVM Architecture

EVM architecture

Ethereum Virtual Machine (EVM)



The EVM is a simple stack-based architecture.

PRESENTED BY: [REDACTED]



Components of Ethereum Network - EVM Architecture

Machine space of EVM



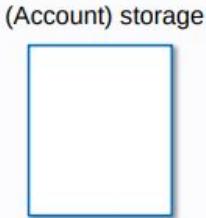
stack memory

256 bits x 1024 elements



volatile memory

byte addressing
linear memory



persistent memory

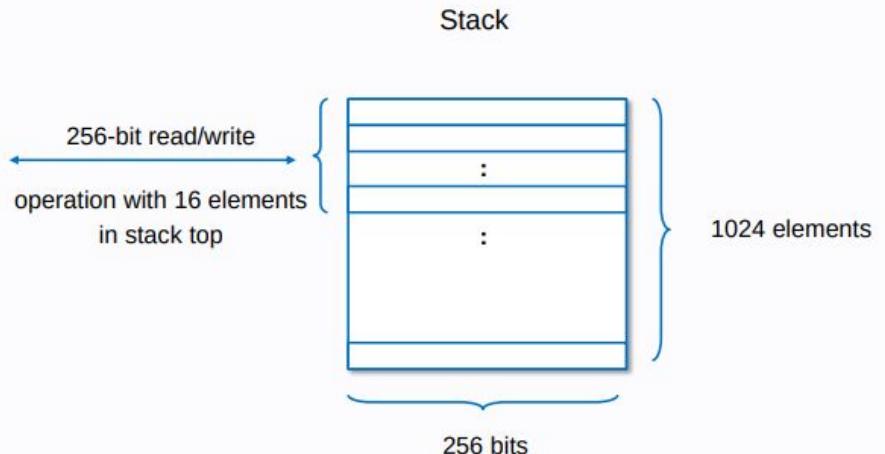
256 bits to 256 bits
key-value store

There are several resources as space.



Components of Ethereum Network - EVM Architecture

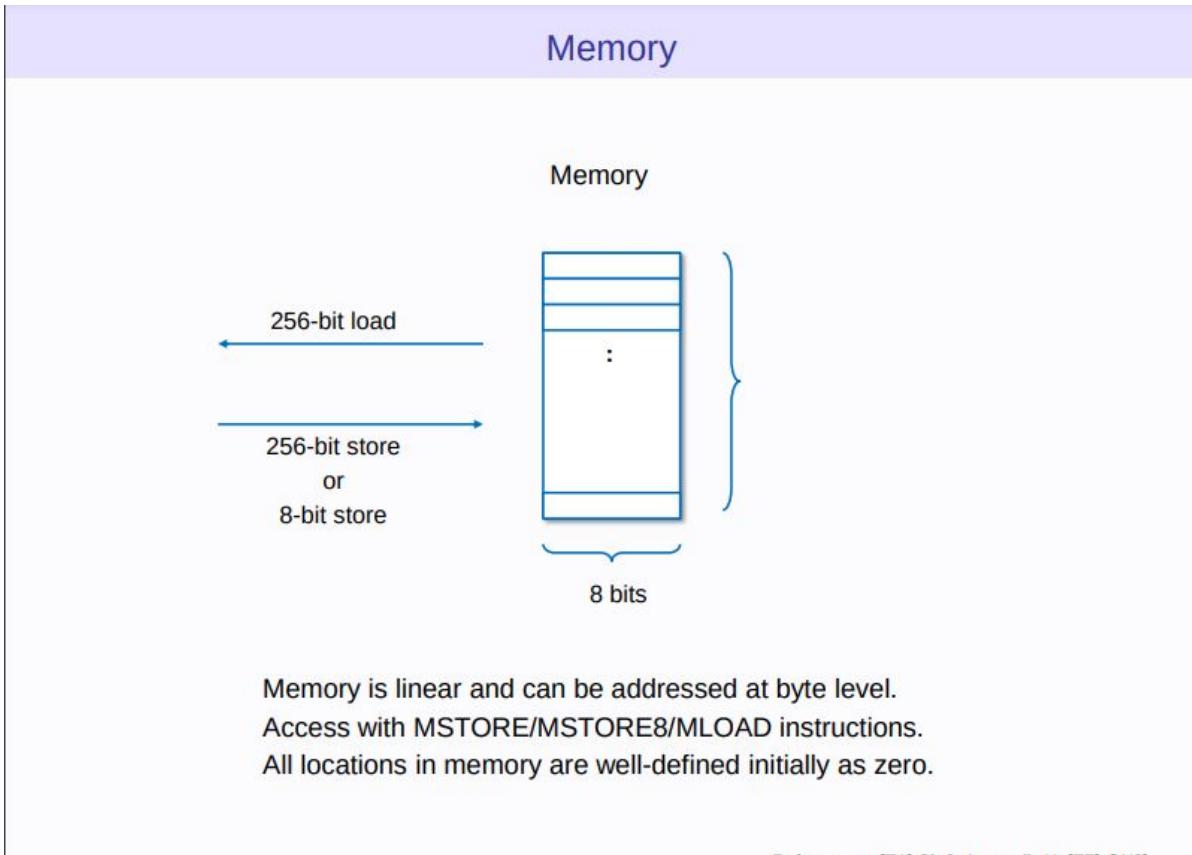
Stack



All operations are performed on the stack.

Access with many instructions such as PUSH/POP/COPY/SWAP, ...

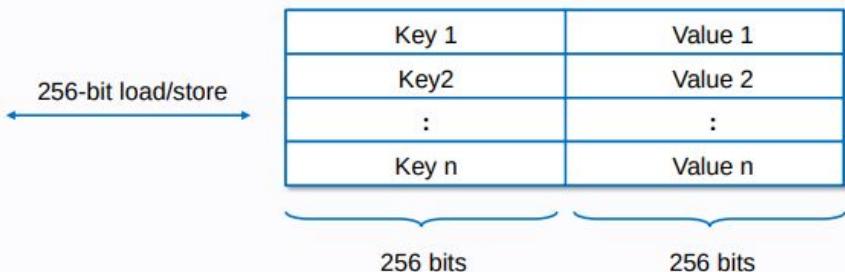




Components of Ethereum Network - EVM Architecture

Account storage

(Account) storage



Storage is a key-value store that maps 256-bit words to 256-bit words.
Access with SSTORE/SLOAD instructions.
All locations in storage are well-defined initially as zero.





Components of Ethereum Network - EVM Architecture

EVM code

Assembly view

```
PUSH1 e0
PUSH1 02
EXP
PUSH1 00
CALLDATALOAD
:
```

Bytecode view

```
0x60e060020a600035...
```

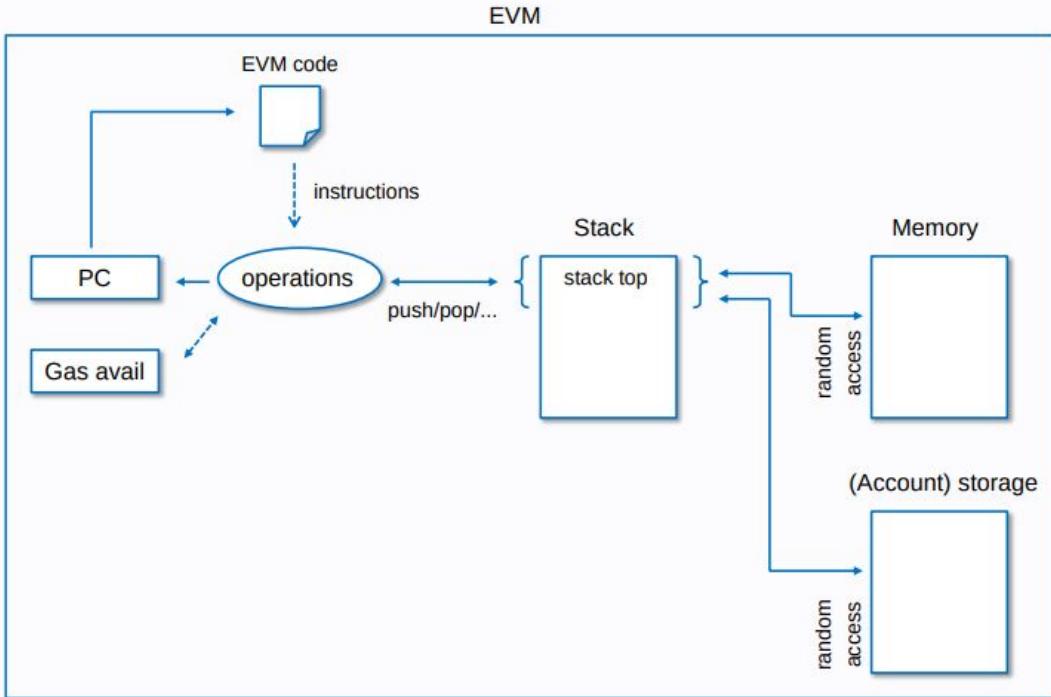
EVM Code is the bytecode that the EVM can natively execute.

QUESTION ANSWER



Components of Ethereum Network - EVM Architecture

Execution model

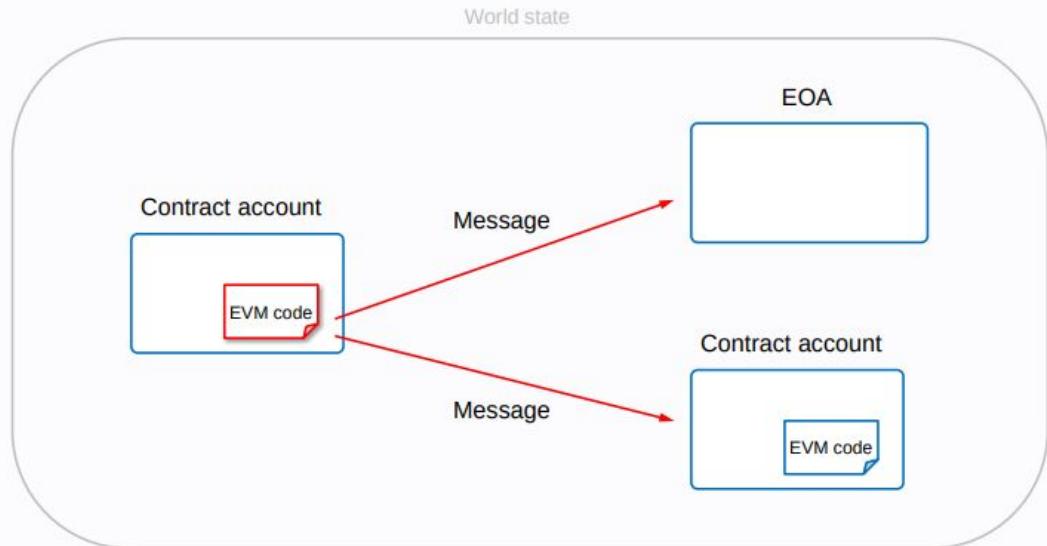


DATA STRUCTURE & ALGORITHMS



Components of Ethereum Network - Transactions

Message call



EVM can send a message to other account.

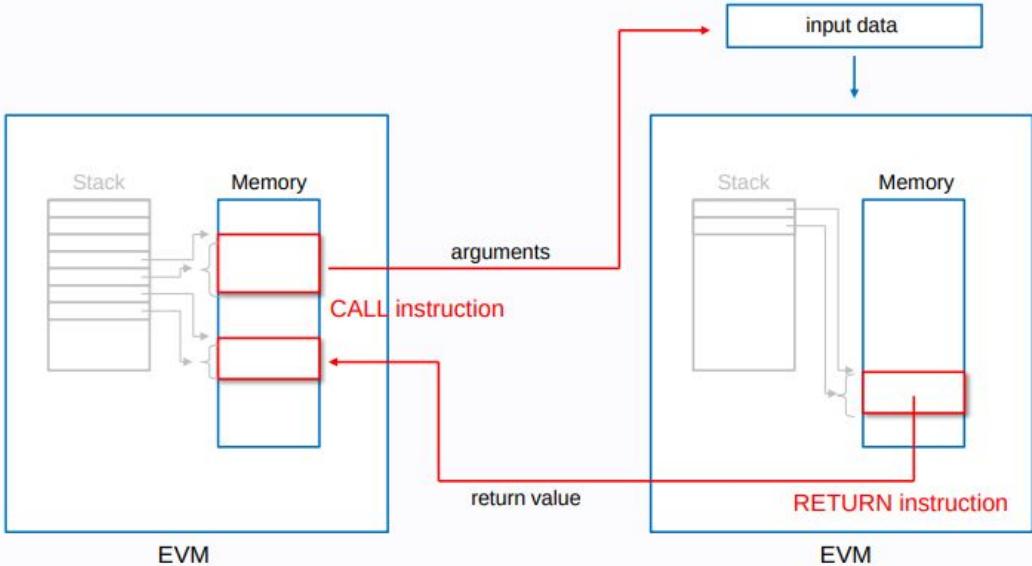
The depth of message call is limited to less than 1024 levels.

QUESTION



Components of Ethereum Network - Transactions

Instructions for Message call



Message call is triggered by CALL instruction.
Arguments and return values are passed using memory.

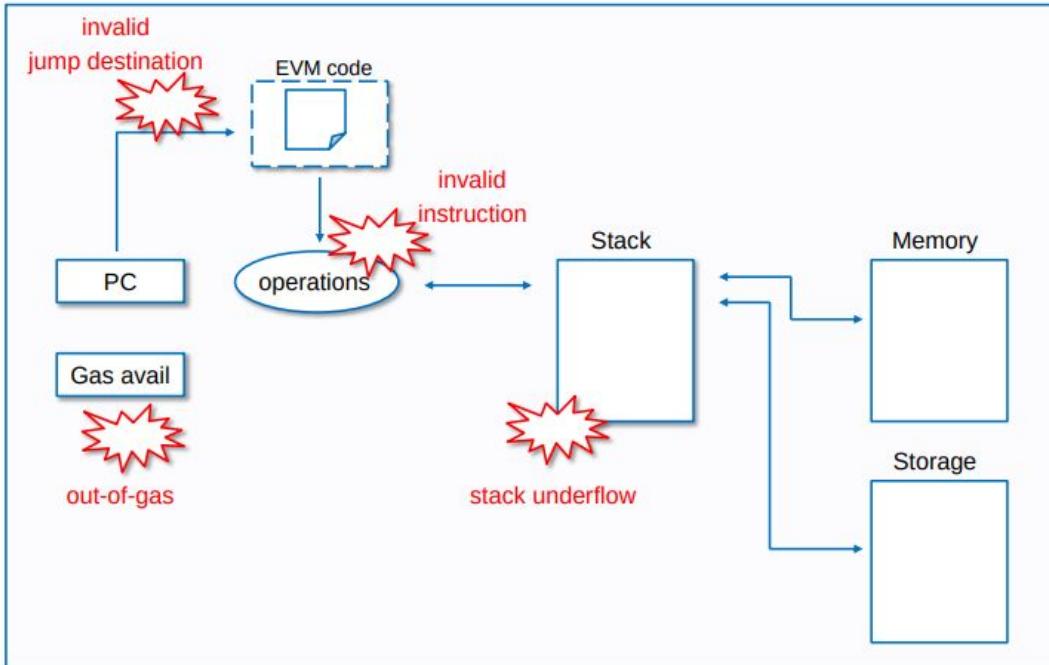
QUESTION PAPER



Components of Ethereum Network - Transactions

Exception

EVM

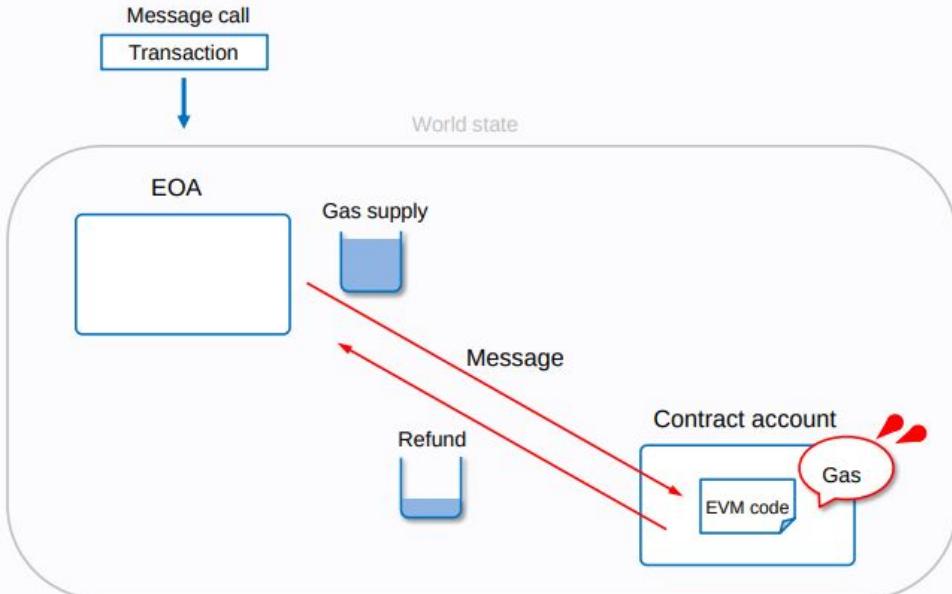


Source: https://ethresear.ch



Components of Ethereum Network - Transactions

Gas and fee

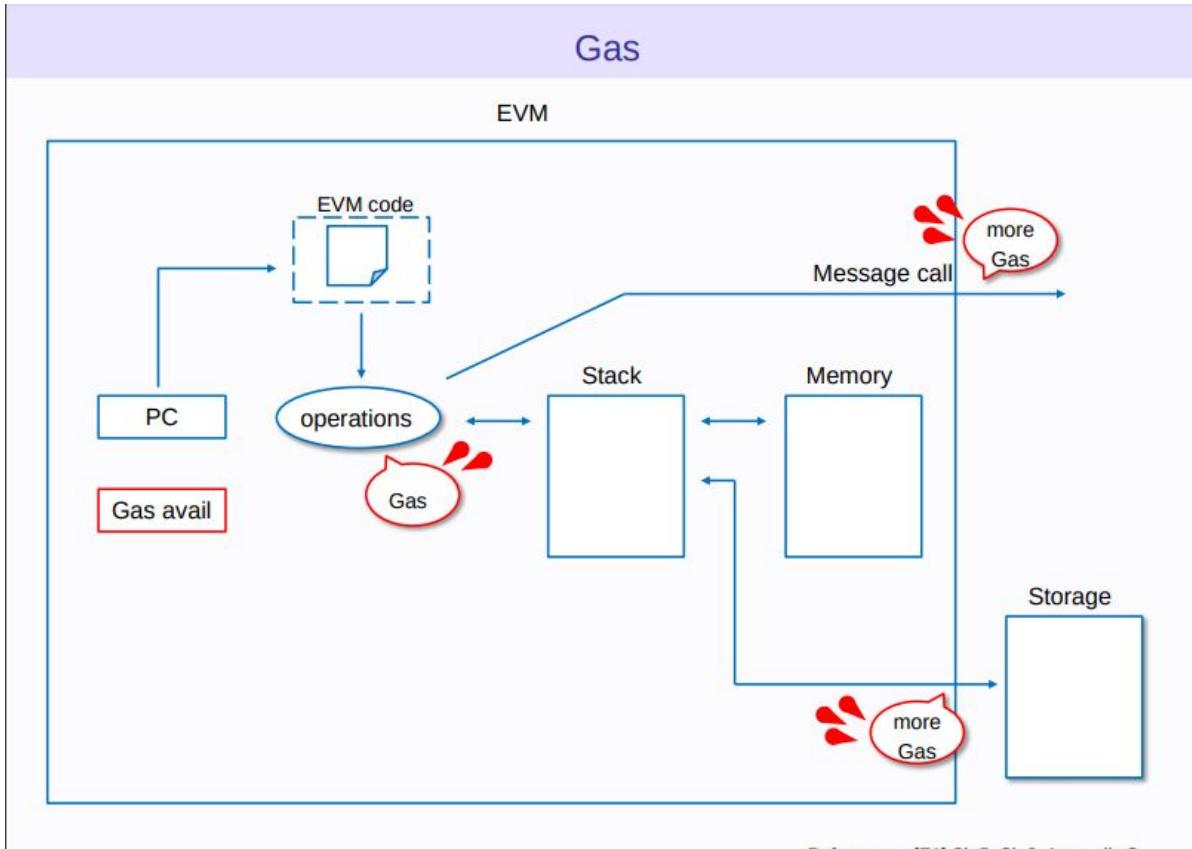


All programmable computation in Ethereum is subject to fees (denominated in gas).

PRINCIPLES OF COMPUTATION

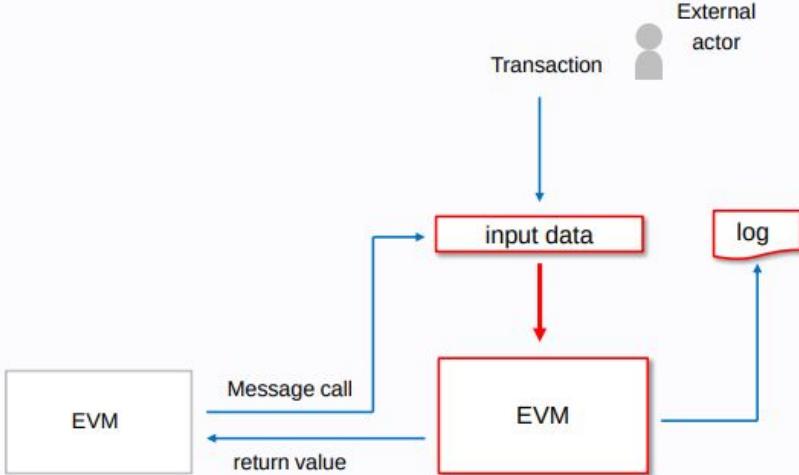


Components of Ethereum Network - Transactions



Components of Ethereum Network - Transactions

Input and Output of EVM



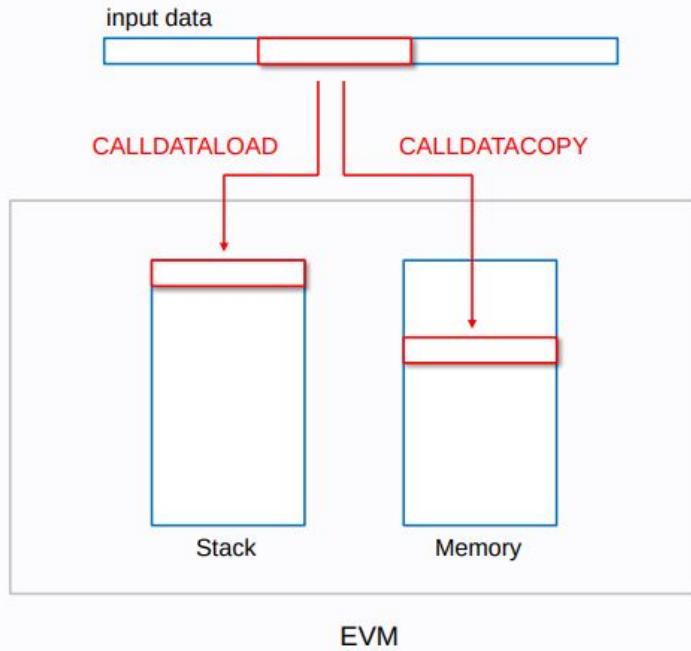
EVM can input external data from a message call.

EVM can output log. EVM can also return values to Caller EVM.



Components of Ethereum Network - Transactions

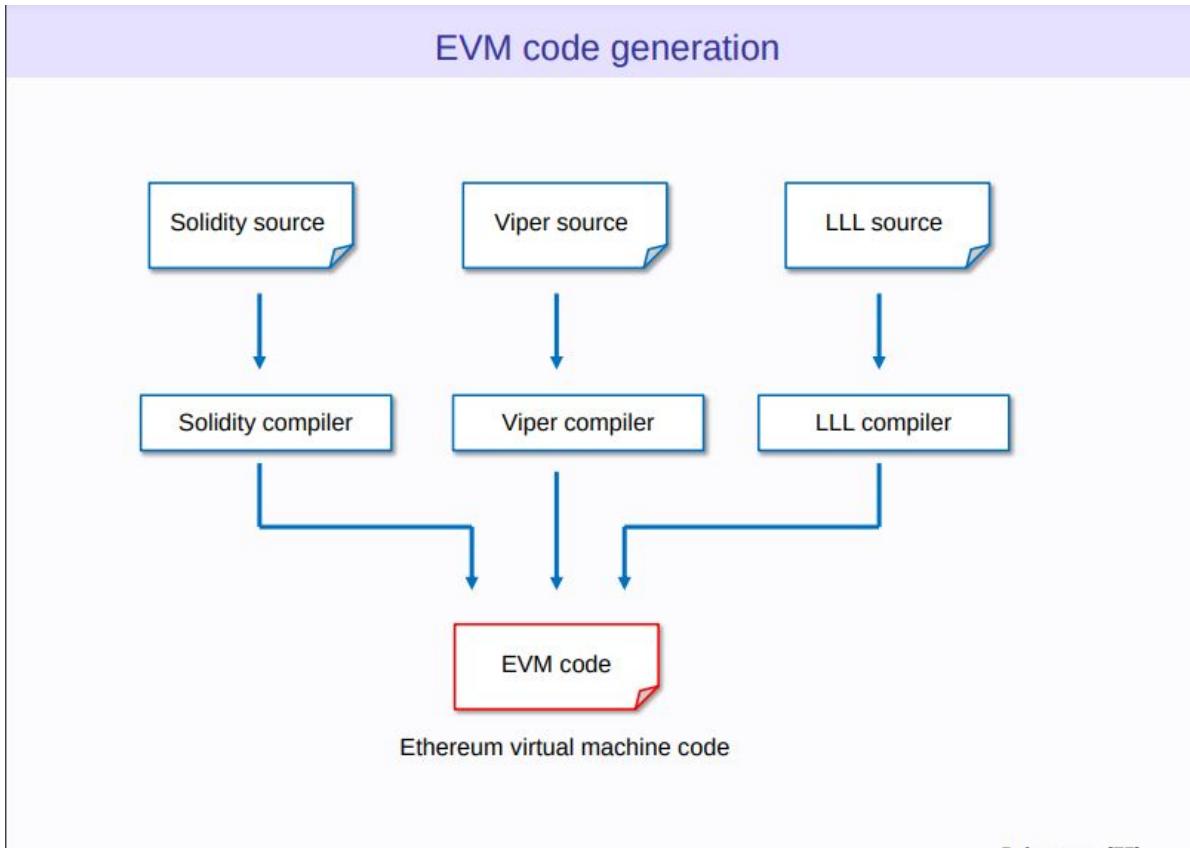
Instructions for input data



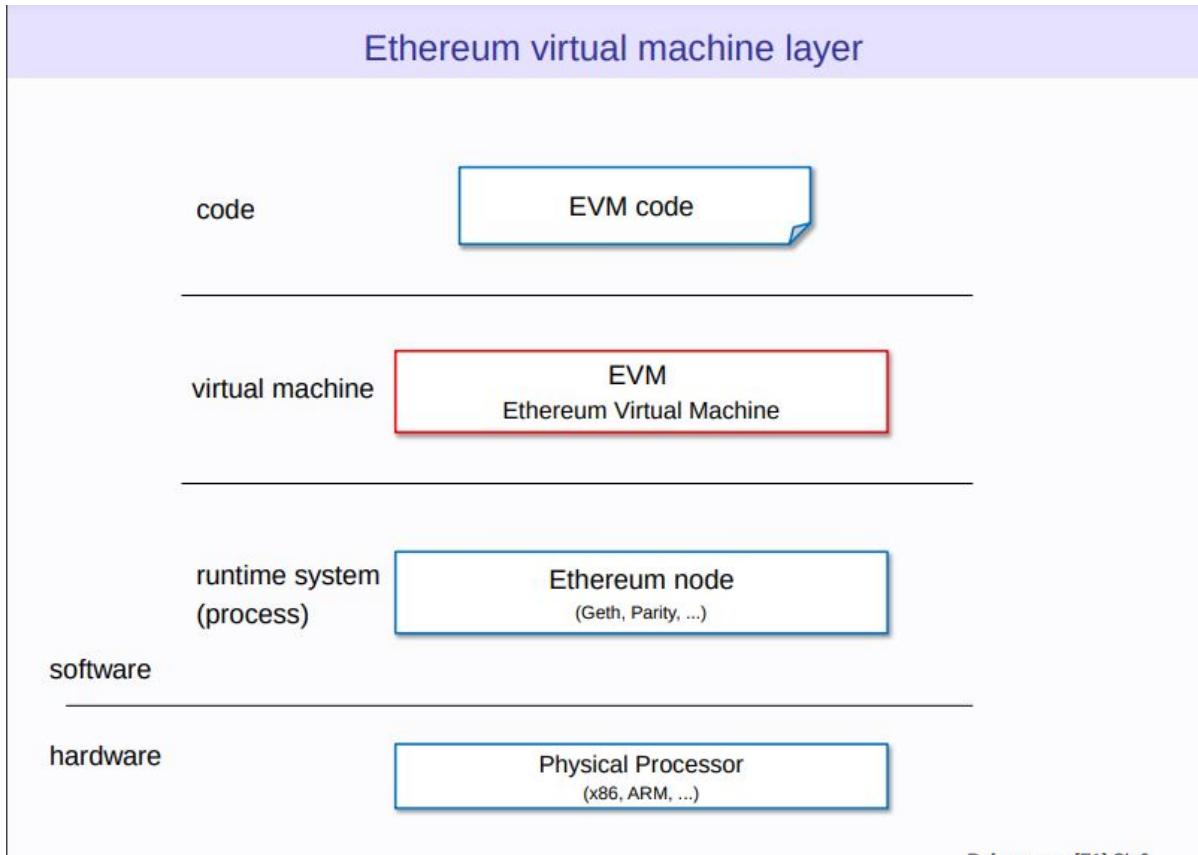
DATASTRUCTURE



Components of Ethereum Network - EVM



Components of Ethereum Network - EVM



References : EP11 Ch 0



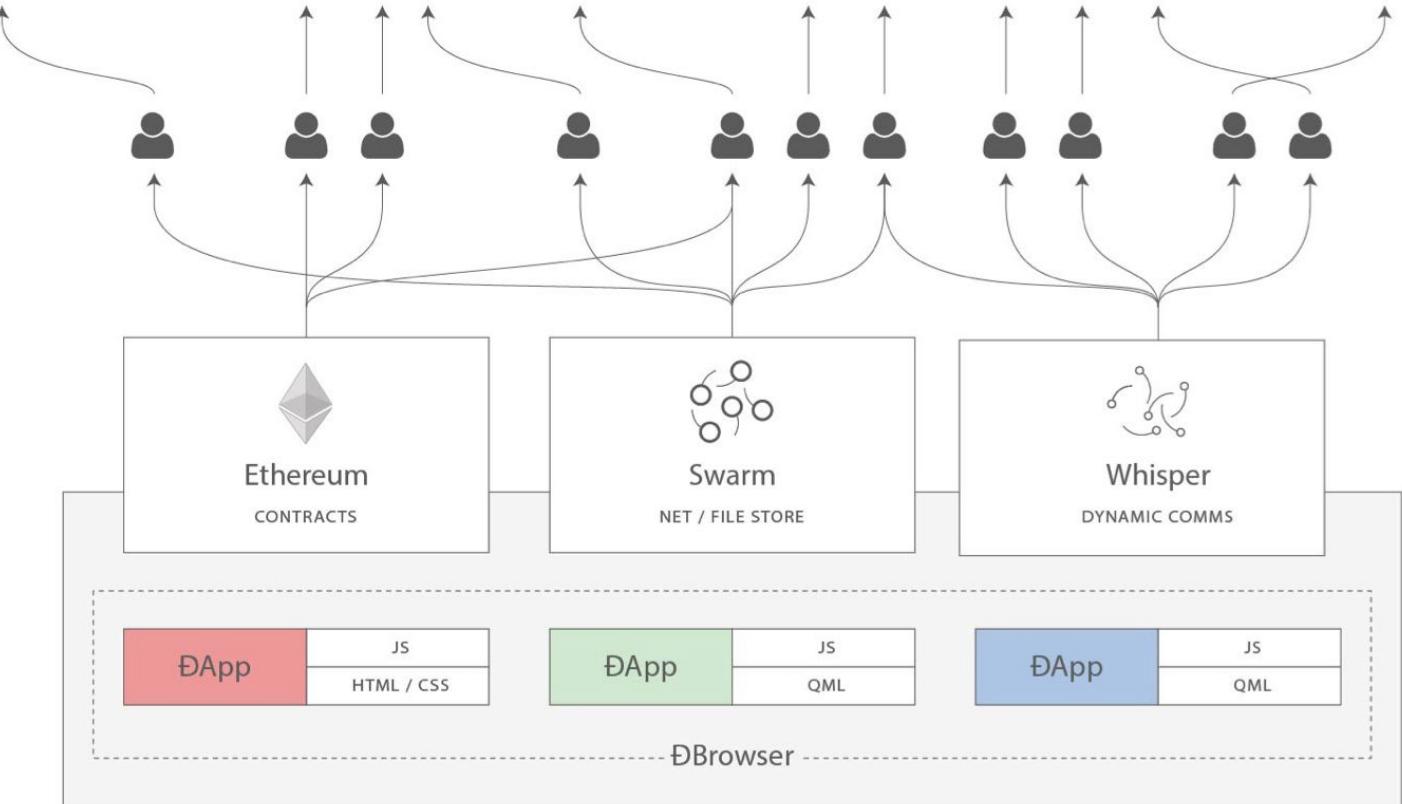
Components of Ethereum Network - Swarm & Whisper



- Swarm and Whisper are complementary technologies contributing to the vision of Ethereum as a "world computer".
- When imagining Ethereum as a metaphor for a shared computer, it should be noted that computation alone is not enough.
- For a computer to be fully useful, it also needs storage to "remember" things and bandwidth to "communicate" them. This could be summarised as such:
 - **Contracts:** decentralized logic
 - **Swarm:** decentralized storage
 - **Whisper:** decentralized messaging



Components of Ethereum Network - Swarm & Whisper



Interaction including Ethereum contracts, Swarm storage, Whisper comms



How Does Ethereum Work?



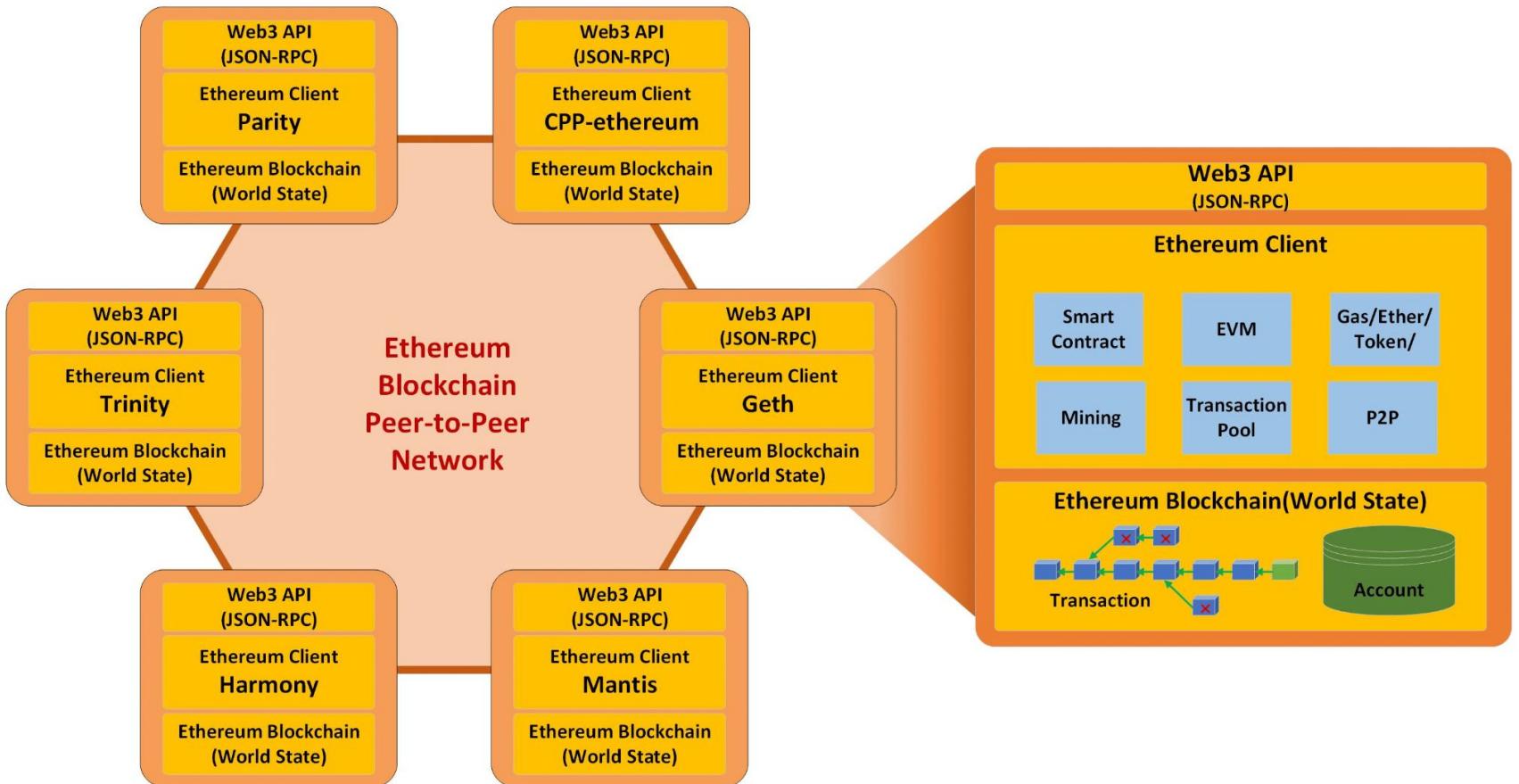
Ethereum implements an execution environment called **Ethereum Virtual Machine (EVM)**.

1. When a transaction triggers a smart contract all the nodes of the network will execute every instruction.
2. All the nodes will run EVM for block verification, where the nodes will go through the transactions listed in the block and runs the code as triggered by the transaction in the EVM.
3. All the nodes on the network must perform the same calculations for keeping their ledgers in sync.
4. Every transaction must include:
 - a. Gas limit.
 - b. Transaction Fee that the sender is willing to pay for the transaction.
5. If total amount of gas needed to process the transaction \leq the gas limit then the transaction will be processed
6. if the total amount of the gas $>$ the gas limit then the transaction will not be processed and the fees are still lost.

Thus it is safe to send transactions with the gas limit above the estimate to increase the chances of getting it processed



Ethereum Architecture





Ethereum Architecture

Ethereum blockchain network :

- decentralized Peer-to-Peer (P2P) network of Ethereum clients, representing network nodes.

Ethereum client

- any node that verify the new transaction, execute the smart contract, and process new blocks of the chain.
- EVM residing in thousands of computers or devices on the internet, and connected through the Ethereum P2P network.
- Ethereum Virtual Machine or EVM and the runtime environment in the P2P network for smart contract execution.
- Ethereum clients run the EVM and can technically be written in any popular programming language.
- There are many different implementations of Ethereum clients.
- Ethereum client implementations are defined in the [Ethereum Yellow Paper](#).
- provide [a set of web3 APIs over JSON-RPC for DApps](#) interacting with an Ethereum blockchain.





Ethereum Architecture



Advantages with a variety of Ethereum client implementations

- makes the network more resilient against bugs.
- prevents the centralization of developer resources.
- competitions between teams help to find the best solutions to common and challenging issues.
- Each client may have a different focus, strength, and weakness in mining, prototyping, DApp development, and more.

From web or wallet application,

- one can use the web3 object provided by the web3.js library to communicate with the Ethereum network.
- It works with any Ethereum client.
- it connects to a local or remote Ethereum node and makes RPC calls.



Ethereum Architecture



DApp developers or private Ethereum blockchain operators may choose the ones fitting their own special needs.

Client	Language	Developers	Where to download?
Geth	Go	Ethereum Foundation	https://geth.ethereum.org/downloads/
Parity	Rust	Parity	https://wwwparity.io
cpp-ethereum	C++	Ethereum Foundation	https://github.com/ethereum/aleth
Trinity	Python	Ethereum Foundation	https://github.com/ethereum/trinity
Harmony	Java	Ether Camp	https://github.com/ethercamp/ethereum-harmony
Mantis	Scala	IOHK's Team Grothendieck	https://mantis.readthedocs.io/en/latest/





In short, **client-server model**,

- Client : **DApps**
- Server : **entire Ethereum network**
- To DApps, the Ethereum network is just like a giant world computer, assembled together with thousands of computing devices throughout the internet.
- Once connected to the network, one could connect to any node in the decentralized network,



Ethereum client provides all blockchain components to maintain

- Managing transaction and state transition with the Ethereum blockchain
- Maintaining world state and account state
- Managing P2P communication Block finalization with mining
- Managing transaction pool
- Managing cryptoassets, gas, ether, and tokens

Ethereum design based on the white paper intends

- to build a simple, efficient and extensible blockchain platform,
- have a Turing-complete program language to support more sophisticated and complex computations.
- Incorporate all of the benefits of a blockchain
- serve as the framework for supporting all types of digital assets and value transfers as well.





Ethereum Architecture

DApp (Decentralized Applications)

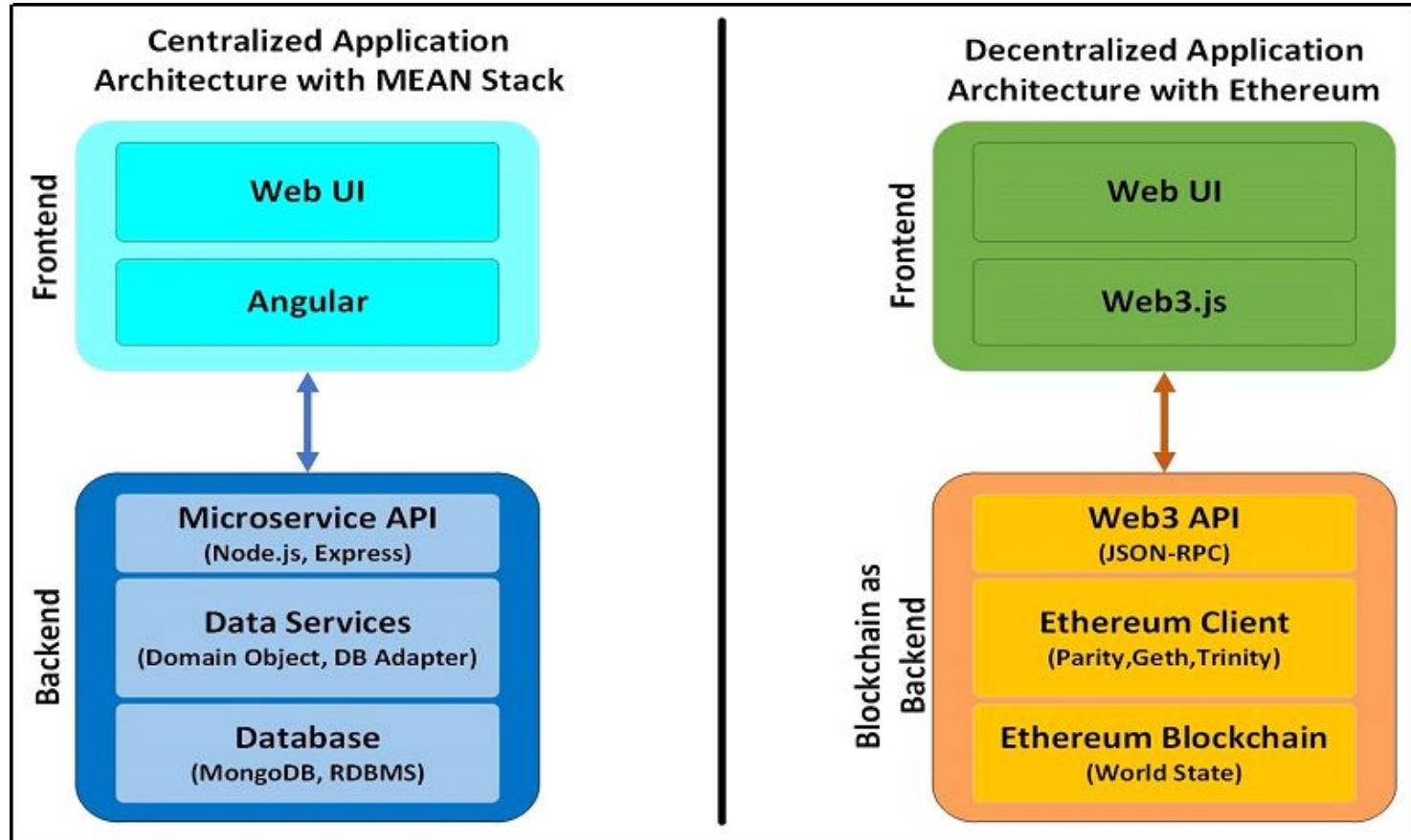
- application or service that runs on a blockchain network
- enables direct interaction between consumers and providers.
- Eg: Connecting buyers and sellers in a decentralized marketplace.
- use and generate cryptographic tokens.
- often start with a whitepaper and a working prototype.

DApp involves

- Decentralized backend that runs on the blockchain network
- Centralized frontend that allows end users to access their wallets and make a transaction



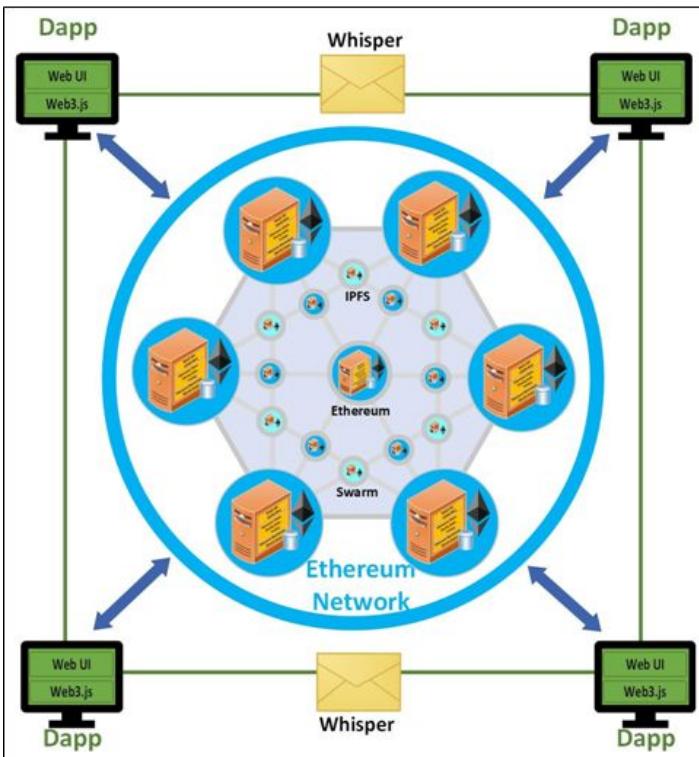
Ethereum Architecture - Centralized Vs Decentralized Applications



Ethereum Architecture

Ethereum provides

- a platform for anyone to write smart contracts and decentralized applications based on your business needs and value propositions.
- It is intended as the world computer for the decentralized world.
- Ethereum provides **four decentralized computing facilities**,
 1. Ethereum blockchain for decentralized state
 2. Smart contracts for decentralized computing
 3. Swarm and IPFS for decentralized storage
 4. Whispers for P2P messaging





Smart Contract - Remix IDE Environment - Key Features



- popular web-based integrated development environment (IDE) for smart contract development on blockchain platforms, especially Ethereum.

1. Web-Based Environment:

- Accessible through a web browser, making it platform-independent
- Easy to use without the need for local installations.

2. Solidity Support:

- Solidity is the primary programming language for writing smart contracts on Ethereum

3. Code Editor:

- with syntax highlighting and auto-completion features,
- making it easier for developers to write and edit Solidity code.

4. Smart Contract Compilation:

- allows users to compile their Solidity smart contracts directly within the IDE.
- It provides information about compilation status, errors, and warnings.





Smart Contract - Remix IDE Environment - Key Features



5. Contract Deployment:

- Developers can deploy their smart contracts to various Ethereum networks (local, testnet, or mainnet) directly from Remix.

6. Debugging Tools:

- allows developers to step through their code, set breakpoints, inspect variables, and troubleshoot issues.

7. Test Environment:

- enables developers to write and run unit tests for their smart contracts.

8. Plugin System:

- allows developers to extend its functionality by integrating additional tools and features.

9. Version Control Integration:

- Developers can integrate Remix with version control systems (such as GitHub) to manage and track changes to their smart contract projects.





Smart Contract - Remix IDE Environment - Key Features



10. Ethereum Virtual Machine (EVM) Integration:

- allows developers to test and deploy contracts on Ethereum-based networks.

11. Security Analysis:

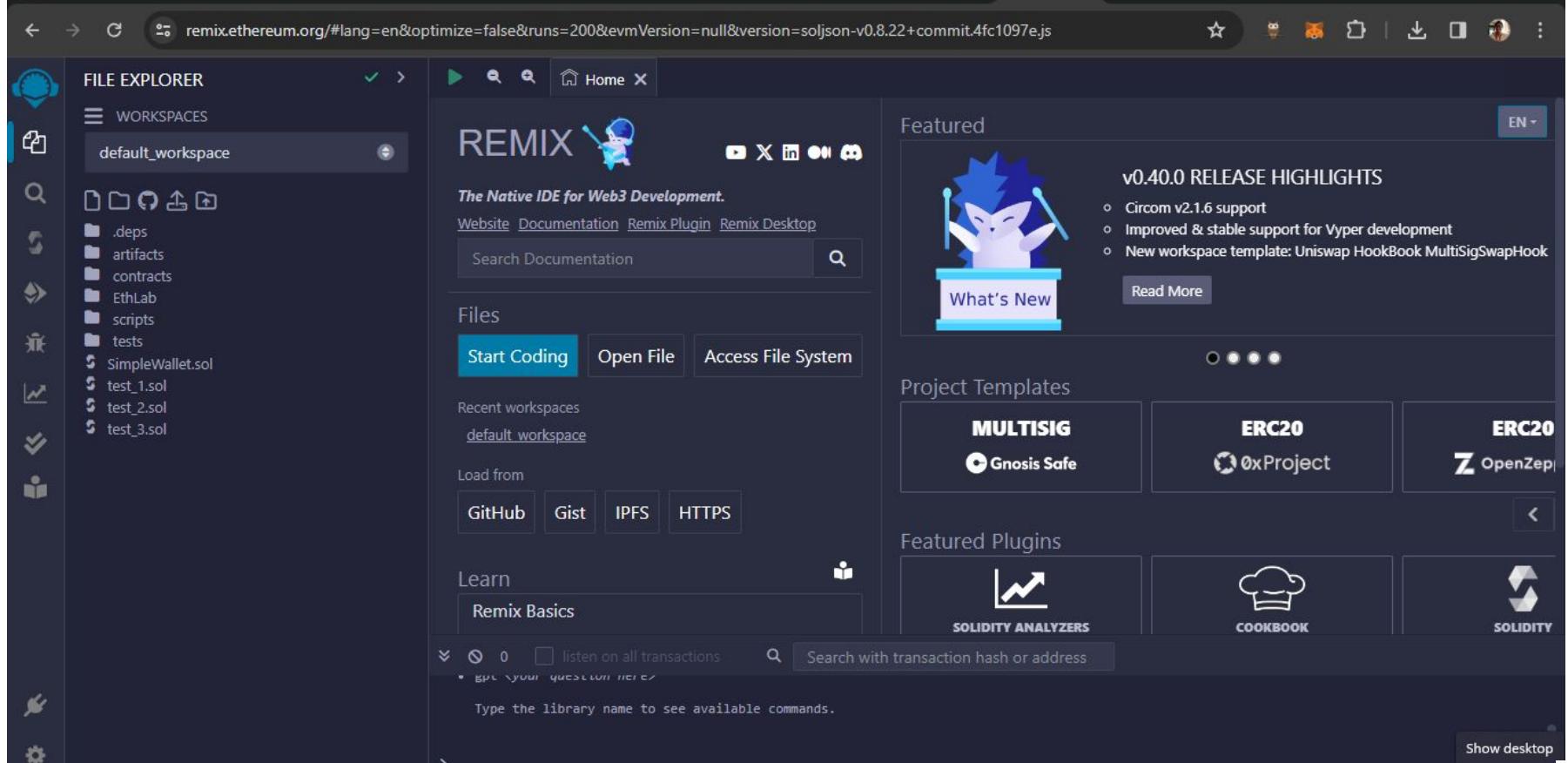
- helps developers identify potential vulnerabilities in their smart contracts.

12. Documentation and Community Support:

- makes it easier for developers to learn and troubleshoot issues.



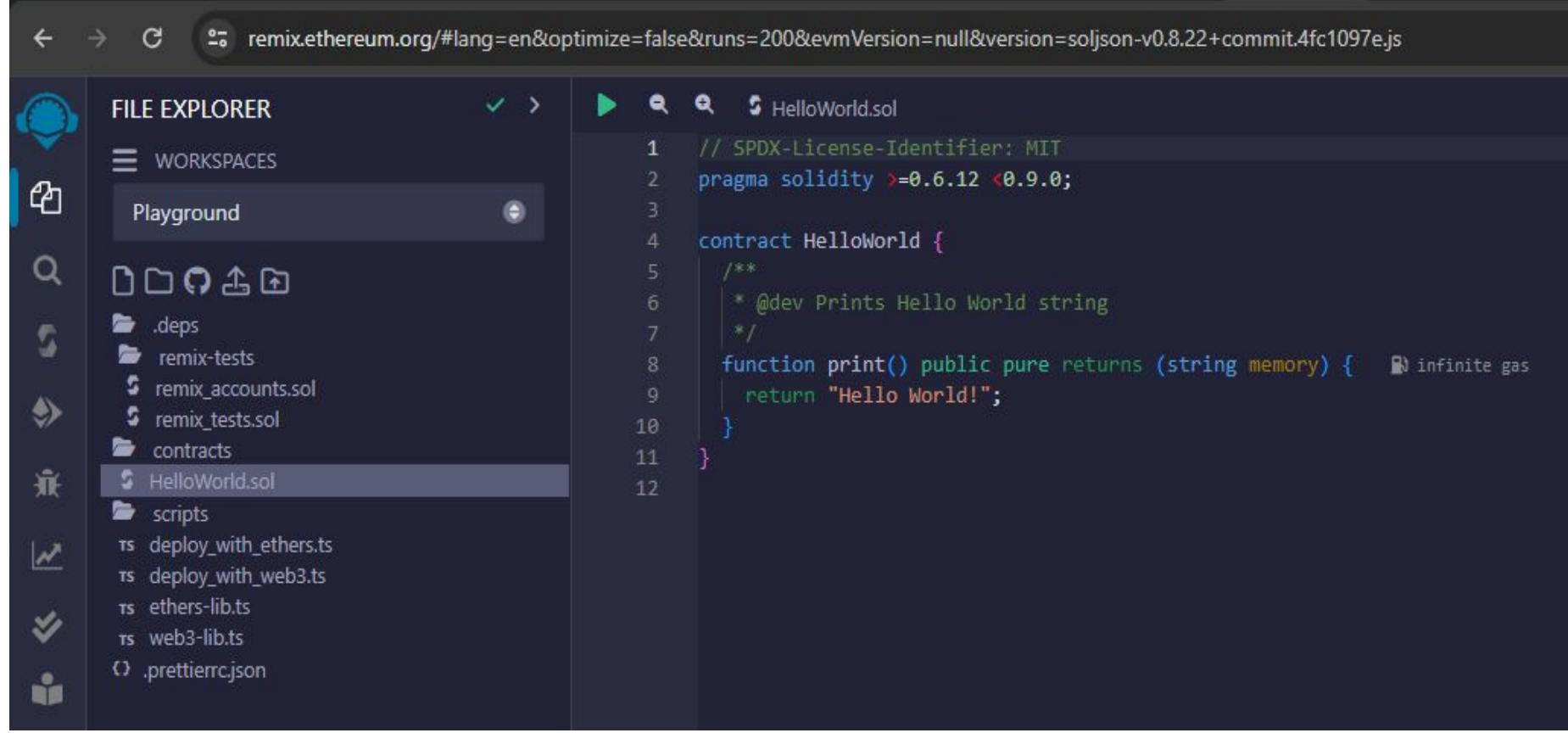
Smart Contract - Remix IDE Environment



The screenshot shows the Remix IDE interface. On the left is the File Explorer sidebar with a list of workspaces and files, including `.deps`, `artifacts`, `contracts`, `EthLab`, `scripts`, `tests`, `SimpleWallet.sol`, `test_1.sol`, `test_2.sol`, and `test_3.sol`. The main workspace is titled "default_workspace". The central area features the "REMIX" logo and the tagline "The Native IDE for Web3 Development". It includes buttons for "Start Coding", "Open File", and "Access File System". Below these are sections for "Recent workspaces" (with "default workspace" listed) and "Load from" (with options for GitHub, Gist, IPFS, and HTTPS). A "Learn" section provides links to "Remix Basics" and "Solidity". At the bottom, there's a command-line interface for running commands like "geth" and "ganache". To the right, the "Featured" section highlights the "v0.40.0 RELEASE HIGHLIGHTS" which include Circom v2.1.6 support, improved Vyper development, and a new workspace template for Uniswap HookBook MultiSigSwapHook. Other sections include "Project Templates" for Multisig, ERC20, and OpenZepplin, and "Featured Plugins" for Solidity Analyzers, Cookbook, and Solidity itself. A "What's New" section with a cartoon character is also present.



Smart Contract - Remix IDE (HelloWorld.sol)



The screenshot shows the Remix IDE interface with the following details:

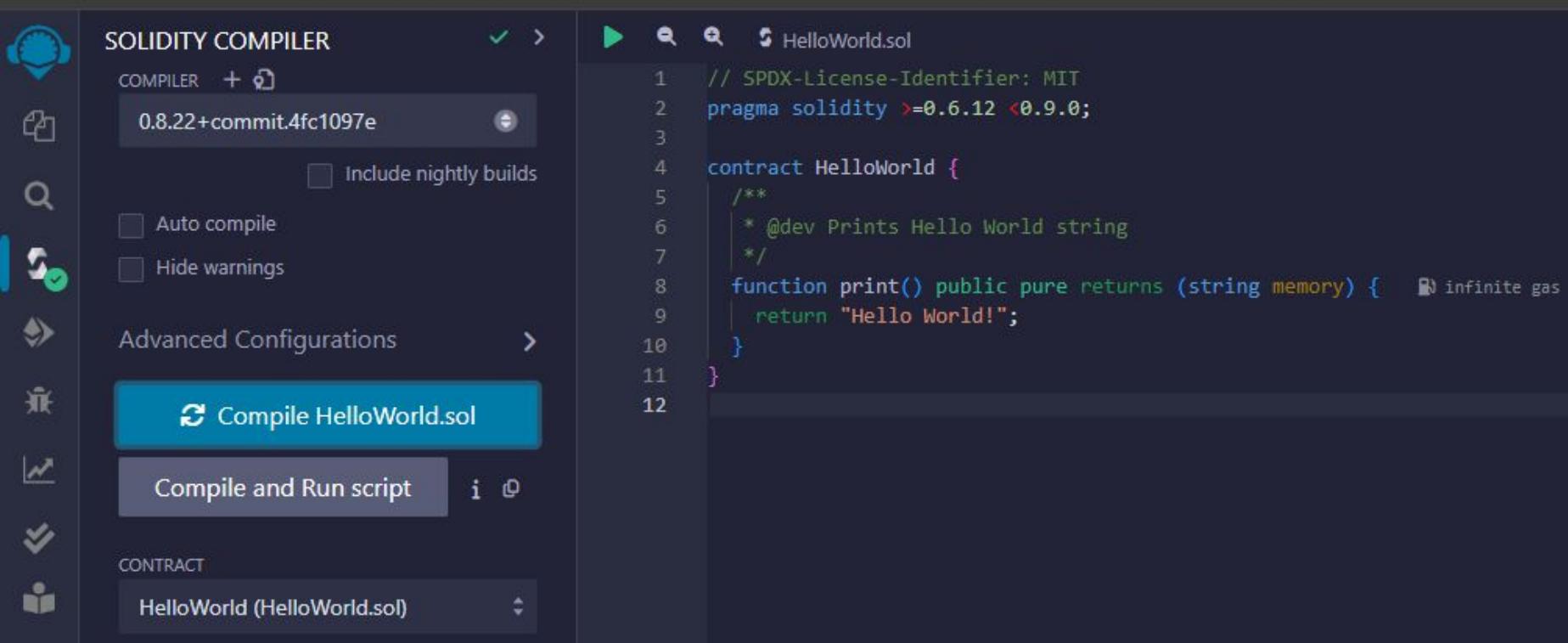
- Header:** remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.22+commit.4fc1097e.js
- File Explorer:** Shows the project structure:
 - WORKSPACES: Playground
 - .deps
 - remix-tests
 - remix_accounts.sol
 - remix_tests.sol
 - contracts
 - >HelloWorld.sol
 - scripts
 - deploy_with_ethers.ts
 - deploy_with_web3.ts
 - ethers-lib.ts
 - web3-lib.ts
 - .prettierrc.json
- Code Editor:** Displays the Solidity code for the HelloWorld contract:

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract HelloWorld {
    /**
     * @dev Prints Hello World string
     */
    function print() public pure returns (string memory) {
        return "Hello World!";
    }
}
```



Smart Contract - Remix IDE (Compile HelloWorld.sol)



The screenshot shows the Remix IDE interface with the Solidity Compiler tab selected. The code editor displays the following Solidity code:

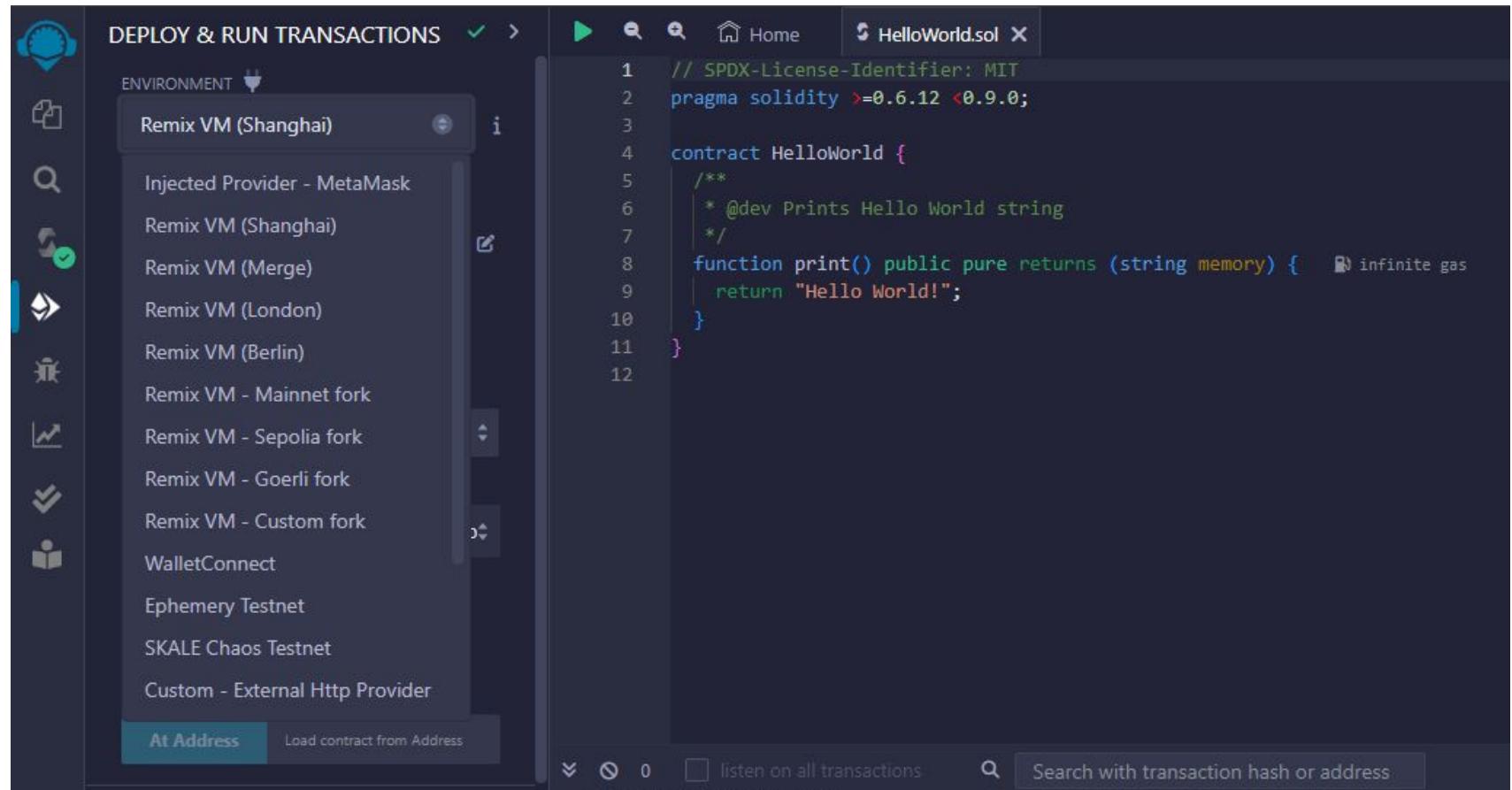
```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract HelloWorld {
    /**
     * @dev Prints Hello World string
     */
    function print() public pure returns (string memory) { infinite gas
        return "Hello World!";
    }
}
```

The interface includes various toolbars and settings on the left side, such as Compiler version (0.8.22+commit.4fc1097e), Auto compile, Hide warnings, Advanced Configurations, and buttons for Compile (highlighted in blue), Compile and Run script, and Deploy.



Smart Contract - Remix IDE (Deploy HelloWorld.sol)



The screenshot shows the Remix IDE interface with the following details:

- Deploy & Run Transactions:** A sidebar on the left with a gear icon.
- Environment:** A dropdown menu currently set to "Remix VM (Shanghai)". Other options include Injected Provider - MetaMask, Remix VM (Merge), Remix VM (London), Remix VM (Berlin), Remix VM - Mainnet fork, Remix VM - Sepolia fork, Remix VM - Goerli fork, Remix VM - Custom fork, WalletConnect, Ephemery Testnet, SKALE Chaos Testnet, and Custom - External Http Provider.
- Contract Name:** HelloWorld.sol
- Contract Code:**

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract HelloWorld {
    /**
     * @dev Prints Hello World string
     */
    function print() public pure returns (string memory) { infinite gas
        return "Hello World!";
    }
}
```
- Bottom Bar:** Buttons for "At Address" and "Load contract from Address", a transaction history, a checkbox for "listen on all transactions", a search bar, and a "Search with transaction hash or address" input field.





The screenshot shows the Truffle UI interface. On the left, there's a sidebar with various icons: a blue headphones icon, a file icon, a magnifying glass icon, a gear icon with a checkmark, a diamond icon, a bar chart icon, and a downward arrow icon. The main area has tabs for "DEPLOY & RUN TRANSACTIONS" (selected), "Home", and "HelloWorld.sol". The "HelloWorld.sol" tab is active, displaying the following Solidity code:

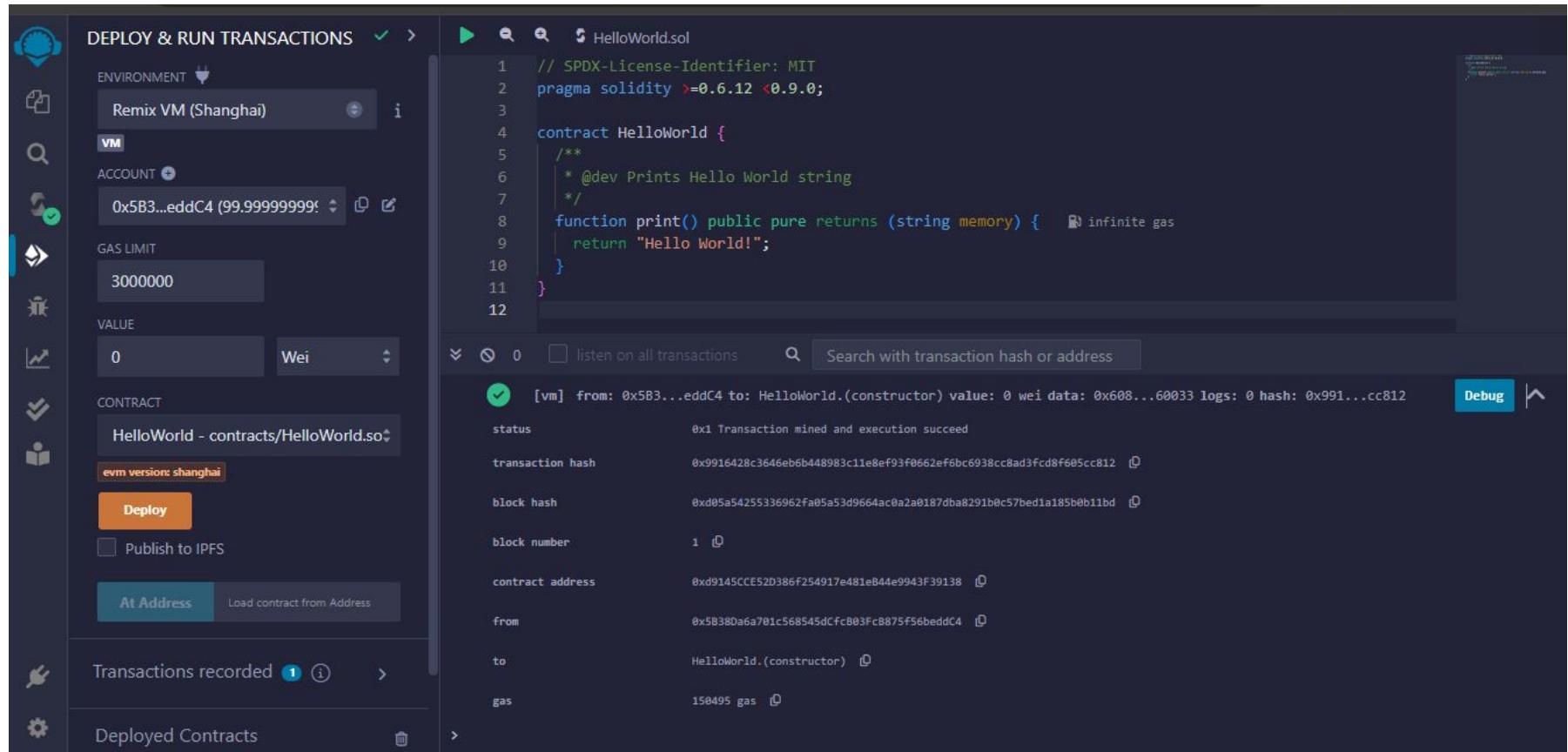
```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract HelloWorld {
    /**
     * @dev Prints Hello World string
     */
    function print() public pure returns (string memory) {
        return "Hello World!";
    }
}
```

On the left side of the main area, there's a "ACCOUNT" dropdown menu with a plus sign. It lists several accounts, each with a balance of 100 ether. The account "0x5B3...eddC4 (100 ether)" is currently selected. A tooltip box appears over the dropdown, stating: "Select a compiled contract to deploy or to use with At Address." The bottom of the screen shows the file path "HelloWorld.sol" and a small "1" icon.



Smart Contract - Remix IDE (Deploy HelloWorld.sol)



The screenshot shows the Remix IDE interface with the following details:

- Environment:** Set to "Remix VM (Shanghai)".
- Gas Limit:** Set to 3000000.
- Value:** Set to 0 Wei.
- Contract:** HelloWorld - contracts/HelloWorld.sol
- Deploy Button:** An orange "Deploy" button is visible.
- Transactions Recorded:** A message indicates 1 transaction recorded.
- Deployed Contracts:** A section for deployed contracts is shown.
- Code Editor:** The code for HelloWorld.sol is displayed:

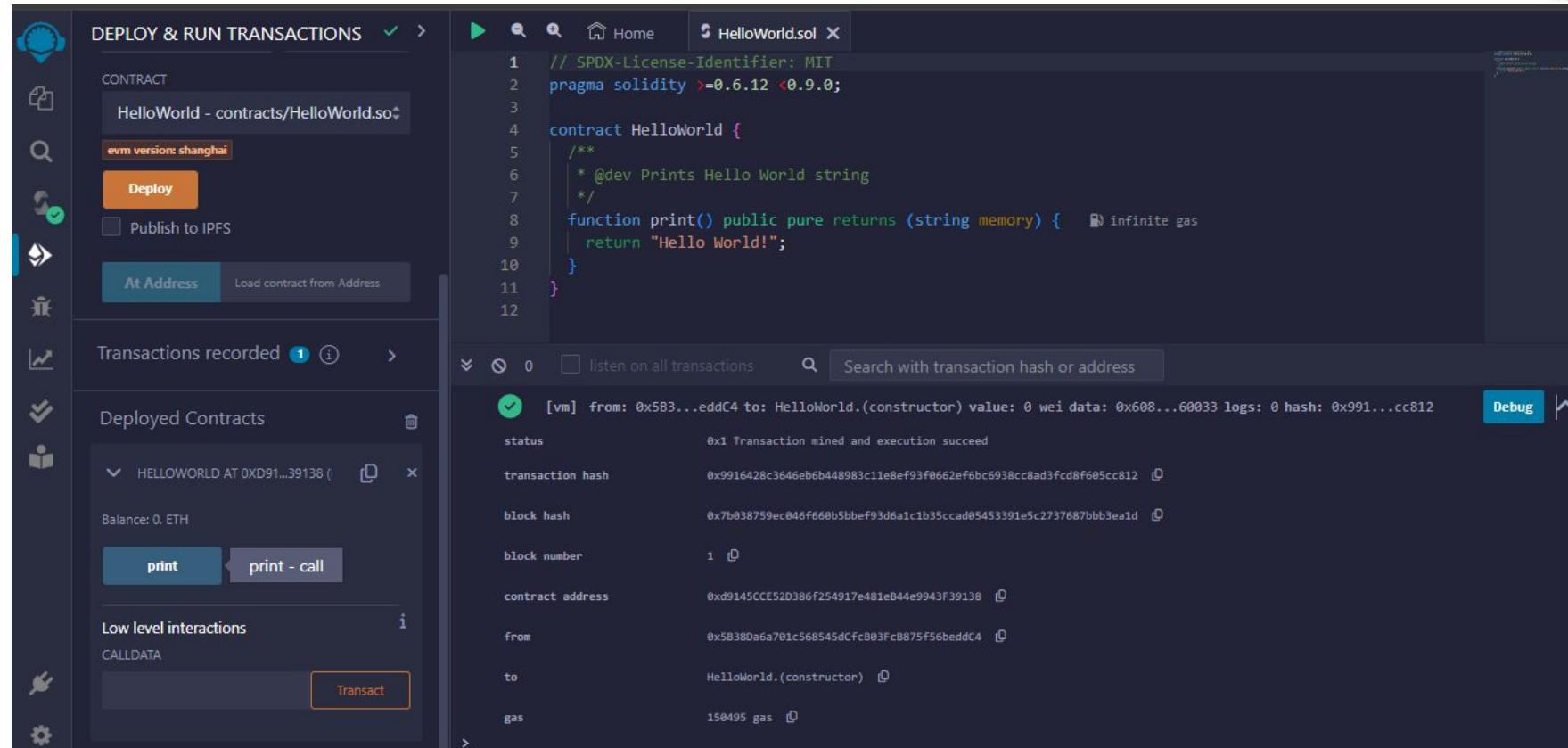
```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract HelloWorld {
    /**
     * @dev Prints Hello World string
     */
    function print() public pure returns (string memory) { infinite gas
        return "Hello World!";
    }
}
```
- Deployment Results:** A table shows the deployment details:

Parameter	Value
status	0x1 Transaction mined and execution succeed
transaction hash	0x9916428c3646eb6b448983c11e8ef93f0062ef6bc6938cc8ad3fcdb05cc812
block hash	0xd05a54255336962fa05a53d9664ac0a2a0187dba8291b0c57bed1a185b0b1bd
block number	1
contract address	0xd9145CCE52D386F254917e481e844e9943F39138
from	0x58380a6a701c568545dCfcB03FcB875f56beddC4
to	HelloWorld.(constructor)
gas	150495 gas



Smart Contract - Remix IDE (Deploy HelloWorld.sol)



The screenshot shows the Remix IDE interface with the following details:

- Left Sidebar:** Includes icons for Home, Deploy & Run Transactions, Contracts, EVM Version (set to Shanghai), Deploy button, Publish to IPFS, At Address (selected), Load contract from Address, Transactions recorded (1), and Deployed Contracts.
- Middle Panel:** Displays the Solidity code for the HelloWorld contract:

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract HelloWorld {
    /**
     * @dev Prints Hello World string
     */
    function print() public pure returns (string memory) {
        return "Hello World!";
    }
}
```
- Right Panel:** Shows the transaction details for the deployed contract:

Index	From	To	Value	Data	Logs	Hash
0	[vm] from: 0x5B3...eddC4	HelloWorld.(constructor)	0 wei	0x608...60033	0 logs	0x991...cc812

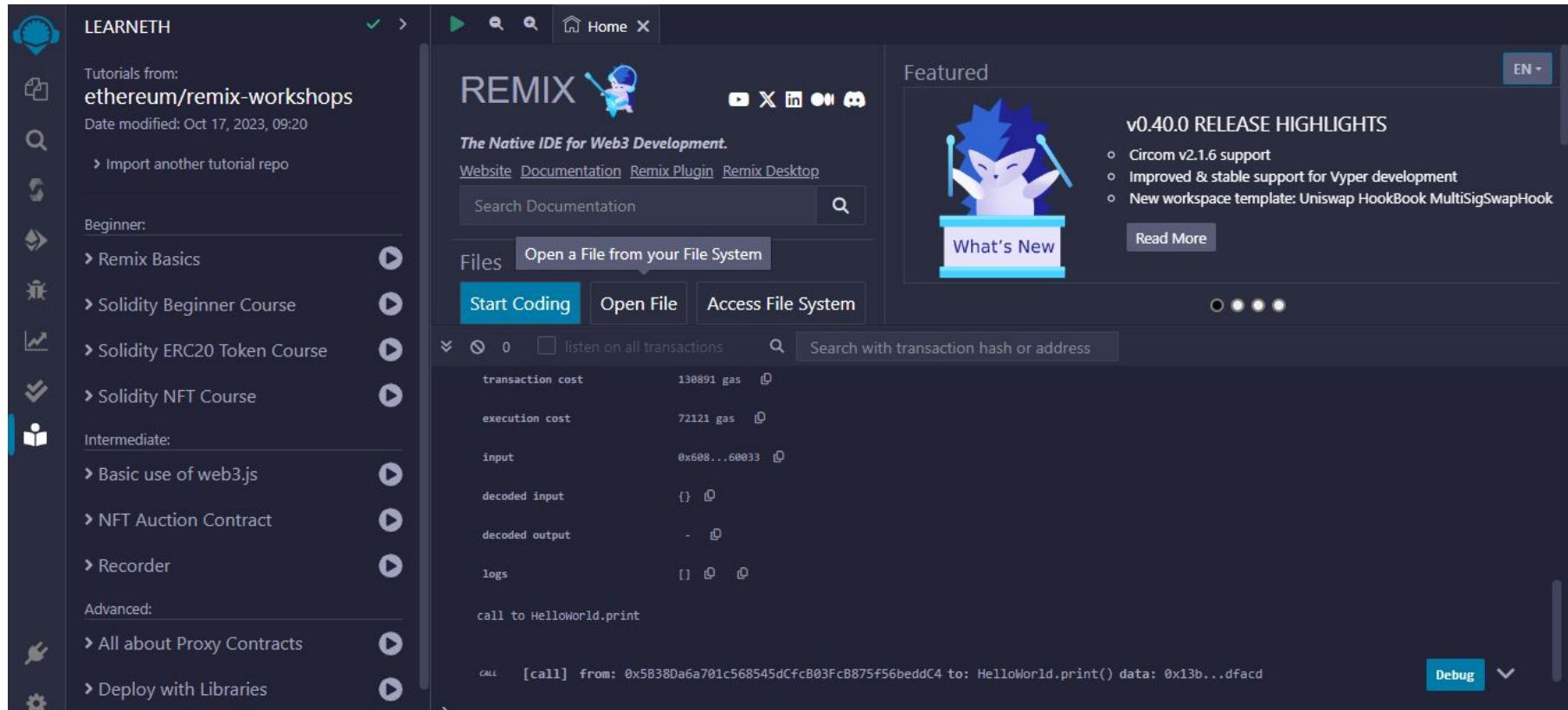
Details for the first transaction:

 - status: 0x1 Transaction mined and execution succeed
 - transaction hash: 0x9916428c3646eb6b448983c11e8ef93f0662ef6bc6938cc8ad3fcdbf605cc812
 - block hash: 0xb038759ec046f660b5bbef93d6a1c1b35ccad05453391e5c2737687bbb3ea1d
 - block number: 1
 - contract address: 0xd9145CCE52D386F254917e481e844e9943F39138
 - from: 0x5B380a6a701c568545dCfc803FcB875f56beddC4
 - to: HelloWorld.(constructor)
 - gas: 150495 gas





Smart Contract - Remix IDE (LearnETH)



The screenshot shows the Remix IDE interface. On the left, there's a sidebar titled "LEARNEETH" with a list of Ethereum remix workshops. The main area displays a Solidity contract named "HelloWorld". The contract has the following code:

```
pragma solidity ^0.8.0;
contract HelloWorld {
    function print() public pure {
        string memory message = "Hello World";
        return message;
    }
}
```

Below the code, it shows deployment details:

- transaction cost: 130891 gas
- execution cost: 72121 gas
- input: 0x608...60033
- decoded input: {}
- decoded output: -
- logs: []

A call log is also present:

```
call [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: HelloWorld.print() data: 0x13b...dfacd
```

At the bottom right, there's a "Debug" button.





Introduction to Dapp (Decentralized Application)



- A software application that operates on a decentralized network, typically using blockchain technology.
- Compared to traditional applications that rely on centralized servers, Dapps leverage the security, transparency, and trustlessness of blockchain to function.
- Salient Features of DApps
 1. **Decentralization:**
 - Dapps run on a decentralized network of computers, often referred to as a blockchain.
 - This network is distributed across many nodes, making it resistant to single points of failure and censorship.
 2. **Smart Contracts:**
 - Dapps typically use smart contracts, which are self-executing contracts with the terms of the agreement directly written into code.
 - Smart contracts automatically execute actions when specific conditions are met, without the need for intermediaries.
 3. **Transparency:**
 - All transactions and actions within a Dapp are recorded on a public ledger (the blockchain).
 - This transparency ensures that anyone can verify transactions and data, enhancing trust in the application.



Introduction to Dapp (Decentralized Application)

4. Security:

- Blockchain technology, with its cryptographic techniques, provides a high level of security.
- Transactions are immutable, meaning once recorded on the blockchain, they cannot be altered.
- This makes Dapps resilient to fraud and hacking.

5. Trustlessness:

- Dapps aim to operate without relying on a central authority or intermediary.
- Users can interact with the application and each other directly without needing to trust a third party.

6. Token Economy:

- Many Dapps have their own native tokens or use established cryptocurrencies like Ethereum's Ether.
- These tokens can be used for various purposes within the application, such as paying for services, participating in governance, or earning rewards.

7. Use Cases:

- Dapps have a wide range of use cases, including decentralized finance (DeFi), non-fungible tokens (NFTs), supply chain management, voting systems, gaming, and more.
- Each Dapp is designed to solve specific problems or provide new functionalities in a decentralized manner.





Introduction to Dapp (Decentralized Application)



8. User Interface:

- Dapps often have user interfaces (UIs) similar to traditional apps or websites, making them accessible to a broader user base.
- Users may not even be aware that they are interacting with blockchain technology.

9. Challenges:

- While Dapps offer numerous advantages, they also face challenges, including scalability issues, user adoption barriers, and regulatory compliance.
- These challenges are actively being addressed by the blockchain community.

10. Development:

- Building a Dapp typically involves programming smart contracts using languages like Solidity (for Ethereum) and developing a frontend interface.
- Popular Dapp development frameworks like Truffle and web3.js make the development process more accessible.



1. Ethereum blockchain:

- Securely executes and verifies application code, called smart contracts.
- DApps use the Ethereum blockchain for data storage.

2. Smart Contracts:

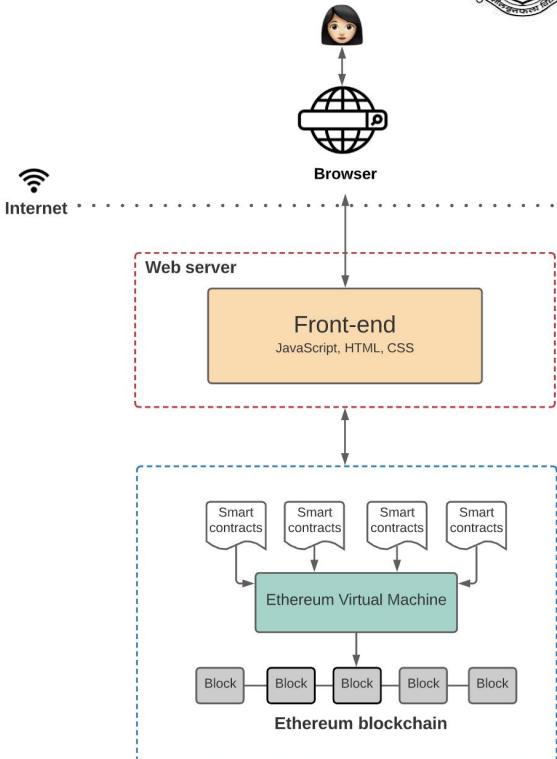
- DApps use smart contracts to define the state changes happening on the blockchain.
- A smart contract is a collection of code and data that resides at a specific address on the Ethereum Blockchain and runs on the Ethereum blockchain.

3. Ethereum Virtual Machine(EVM):

- Global virtual computer that executes the logic defined in the smart contracts and processes the state changes that happen on this Ethereum network.

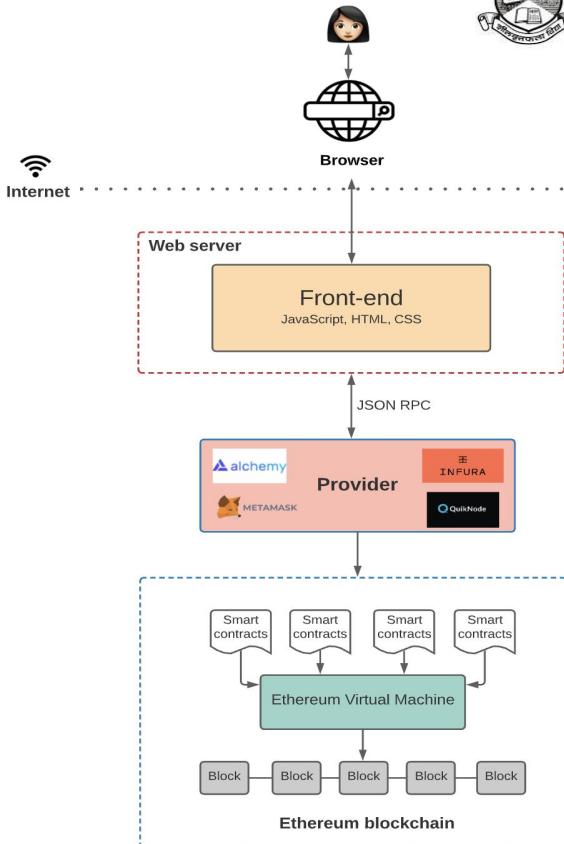
4. Front-end:

- The part of the DApps, that users can see and interact with such as the graphical user interface(GUI),
- Communicates with the application logic defined in smart contracts.



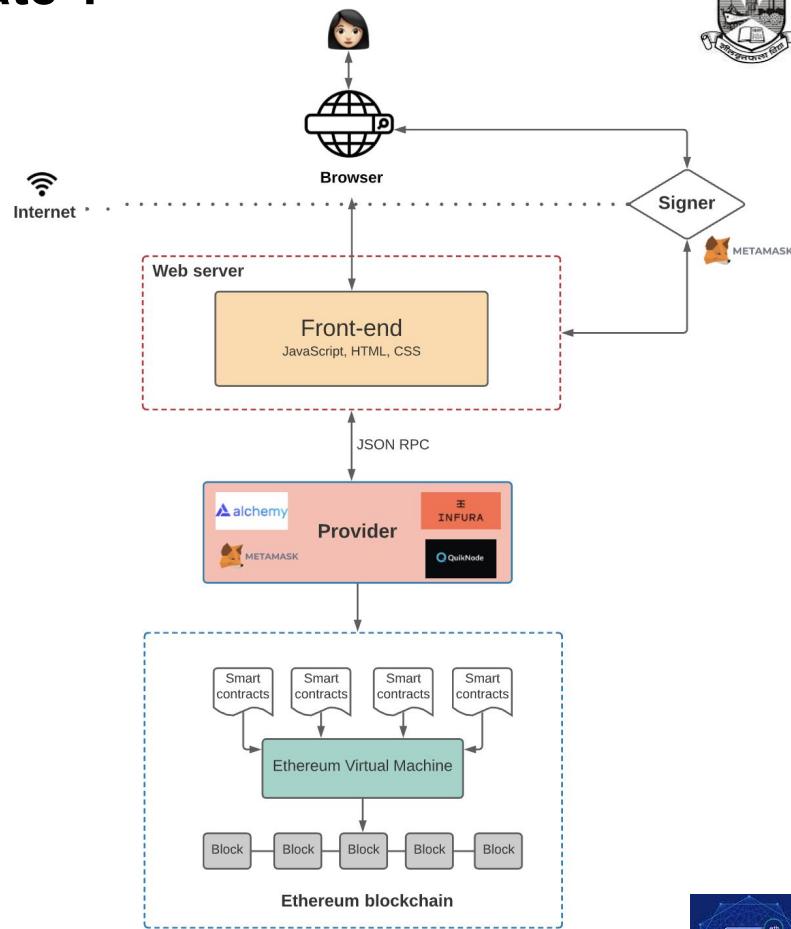
DApp - How Frontend Code Communicate ?

- When we need to interact with the data and code on a blockchain, we need to interact with one of these nodes.
- As any node can broadcast a request for a transaction to be executed on the EVM.
- A miner will then execute the transaction and propagate the resulting state change to the rest of the network.
- There are two ways to broadcast a new transaction:
 1. Set up your own node which runs the Ethereum blockchain software
 2. Use nodes provided by third-party services like [Infura](#), [Alchemy](#), and [Quicknode](#)
- Every Ethereum client (i.e. provider) implements a JSON-RPC specification. This ensures that there's a uniform set of methods when frontend applications want to interact with the blockchain.



DApp - How Frontend Code Communicate ?

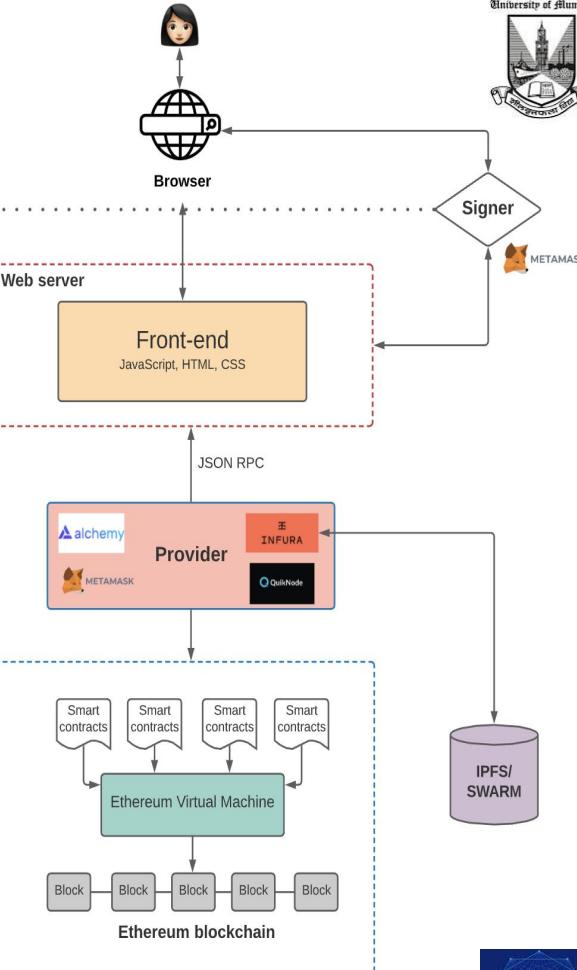
- When a user wants to publish a new post onto the chain, our DApp would ask the user to “sign” the transaction using their private key — only then would the DApp relay the transaction to the blockchain.
- Otherwise, the nodes wouldn’t accept the transaction.
- This “signing” of transactions is where [Metamask](#) typically comes in.



DApp - Storage on the Blockchain

- Storing everything on the blockchain gets really expensive.
- Users to pay extra for using your DApp every time their transaction requires adding a new state is not the best user experience.
- One way to combat this is to **use a decentralized off-chain storage solution**, like [IPFS](#) or [Swarm](#).
- IPFS : Distributed file system for storing and accessing data.
 - distributes and stores the data in a peer-to-peer network.
 - Has an incentive layer known as “Filecoin.” This layer incentivizes nodes around the world to store and retrieve this data.
 - IPFS providers
 - **Infura** (which provides you with an IPFS node) or
 - Pinata (“pin” your files to IPFS and take the IPFS hash and store that on the blockchain).
- **Swarm** : A decentralized storage network,
 - **Difference :** Swarm’s incentive system is built-in and enforced through smart contracts on the Ethereum blockchain for storing and retrieving data.

Internet



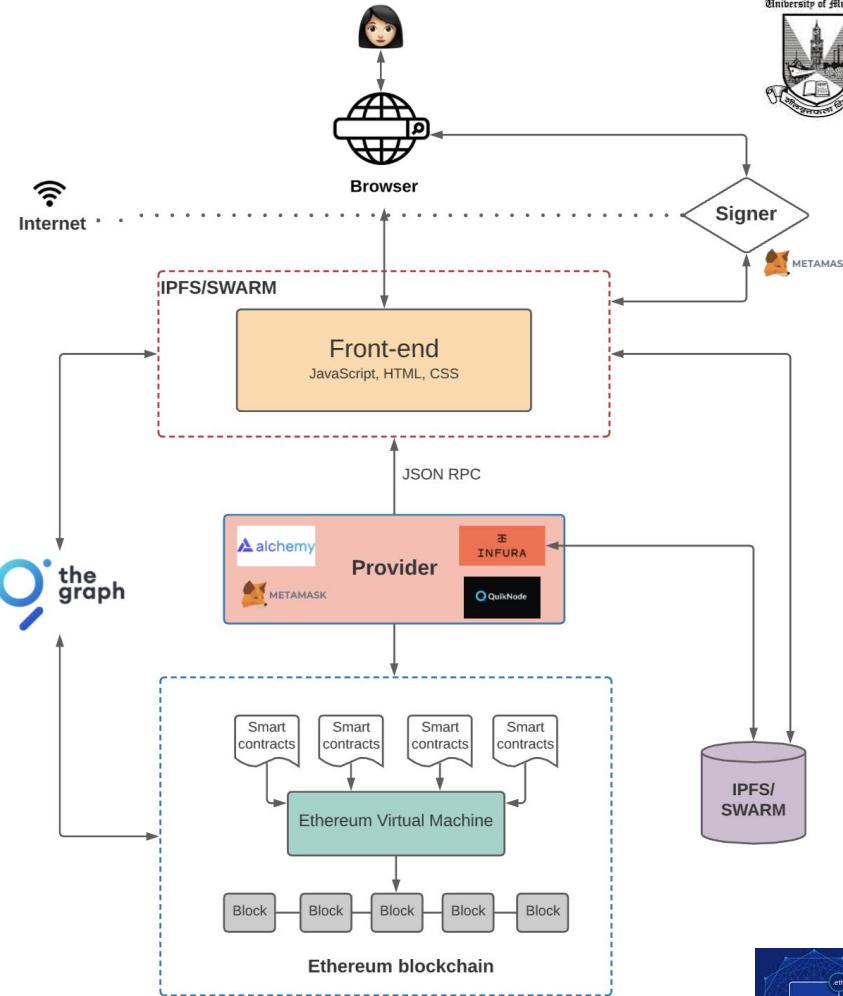
DApp Querying the Blockchain

1. Smart Contract Events :

- Use the Web3.js library to query and listen for smart contract events.
- Here, we need to listen to specific events and specify a callback every time the event is fired.
- using callbacks to handle various UI logic gets very complex
- Issue: when you deploy a smart contract and later realize you need an event emitted that you didn't originally include.

2. The Graph

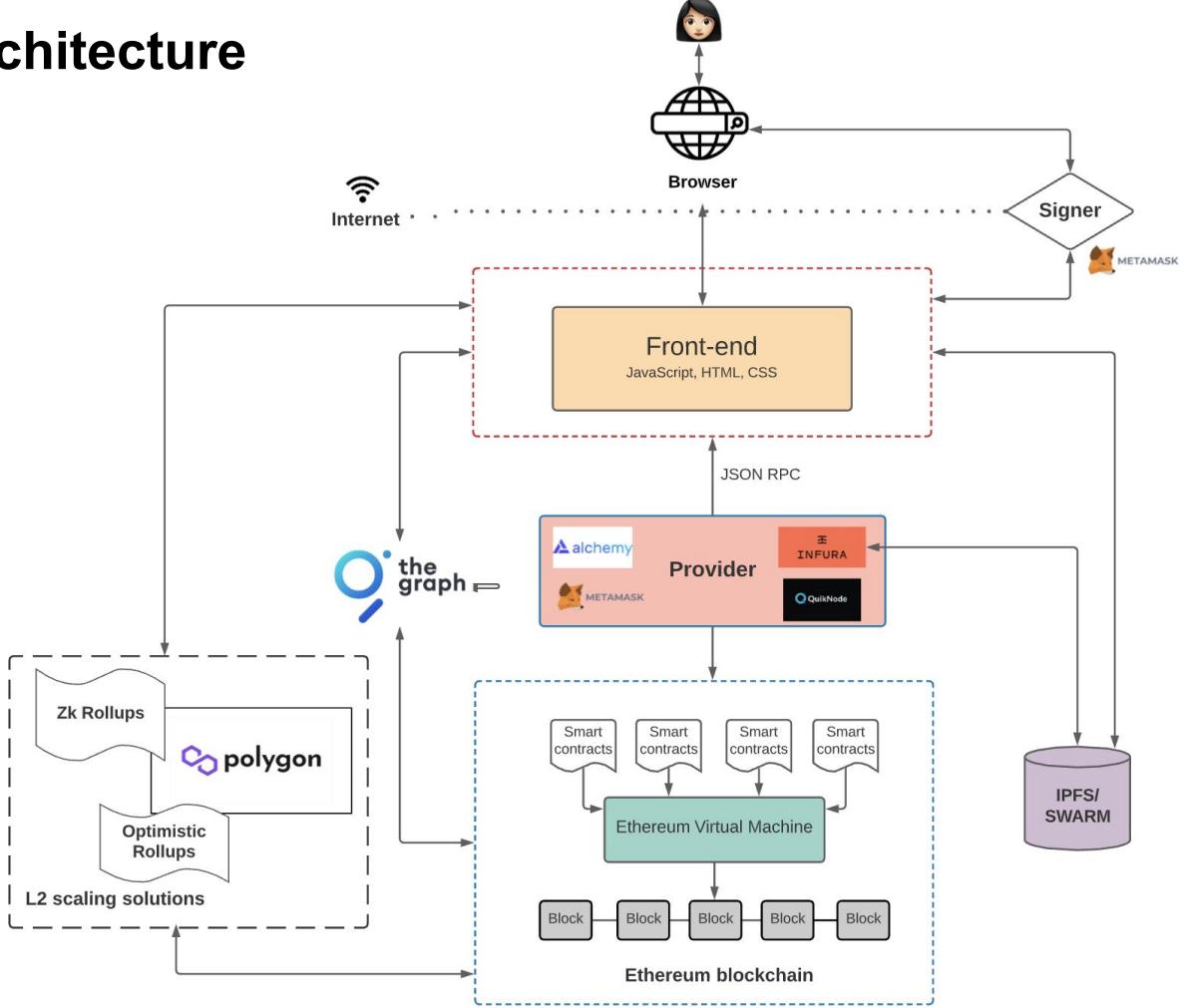
- An off-chain indexing solution that makes it easier to query data on the Ethereum blockchain.
- Allows to define which smart contracts to index, which events and function calls to listen to, and how to transform incoming events into entities that the frontend logic can consume.
- It uses **GraphQL** as a query language,
- By indexing blockchain data, The Graph lets us query on-chain data in our application logic with low latency



Overall DApp Architecture

Addressing

- Storage,
- Querying
- Scalability





Ethereum Networks

- groups of connected computers that communicate using the Ethereum protocol.
- There is **only one Ethereum Mainnet**,
- But **independent networks**
 - conforms to the **same protocol rules** can be created
 - for testing and development purposes.
 - These independent "networks" that conform to the protocol **without interacting with each other**.

Note :

- One can even start one locally on your own computer for testing your smart contracts and web3 apps.
- Eg : Geth & Ganache Private Networks created during the Lab Experiments
- Your **Ethereum account will work across the different networks**,
- But your account balance and transaction history won't carry over from the main Ethereum network.

For testing purposes :

- it's useful to know which networks are available
- how to get testnet ETH to play around with.

For security considerations : it's **not recommended to reuse mainnet accounts on testnets or vice versa**.



- **PUBLIC NETWORKS**

- Public networks are accessible to anyone in the world with an internet connection.
- Anyone can read or create transactions on a public blockchain and validate the transactions being executed.
- The consensus among peers decides on the inclusion of transactions and the state of the network.

- **Ethereum Mainnet**

- Mainnet is the primary public Ethereum production blockchain,
- where actual-value transactions occur on the distributed ledger.
- When people and exchanges discuss ETH prices, they're talking about Mainnet ETH.





Ethereum Networks



- **Ethereum Testnets**

- These are networks used by protocol developers or smart contract developers to test both protocol upgrades as well as potential smart contracts in a production-like environment before deployment to Mainnet.
- Test any contract code you write on a testnet before deploying to Mainnet.
- Among dapps that integrate with existing smart contracts, most projects have copies deployed to testnets.
- Most testnets started by using a permissioned proof-of-authority consensus mechanism.
- This means a small number of nodes are chosen to validate transactions and create new blocks – staking their identity in the process.
- Alternatively, some testnets feature an open proof-of-stake consensus mechanism where everyone can test running a validator, just like Ethereum Mainnet.
- **ETH on testnets is supposed to have no real value;**
- Most people get testnet ETH for free from faucets.
- Most faucets are webapps where you can input an address which you request ETH to be sent to.



- Which Testnet should I use?
 - The two public testnets that client developers are **Sepolia** and **Goerli**.
 - **Sepolia**
 - recommended default testnet for application development.
 - Features :
 - Closed validator set, controlled by client & testing teams
 - New testnet, less applications deployed than other testnets
 - Fast to sync and running a node requires minimal disk space
 - **Goerli (long-term support)**
 - the Goerli testnet is deprecated and will be replaced by Holesovice in 2023.
 - Features :
 - Open validator set, stakers can test network upgrades
 - Large state, useful for testing complex smart contract interactions
 - Longer to sync and requires more storage to run a node



- **PRIVATE NETWORKS**

- An Ethereum network is a private network if its nodes are not connected to a public network (i.e. Mainnet or a testnet).
- Private only means reserved or isolated, rather than protected or secure.

- **Development networks**

- To develop an Ethereum application, run it on a private network to see how it works before deploying.
- Create a local blockchain instance to test your dapp.
- This allows for much faster iteration than a public testnet.

- **Consortium networks**

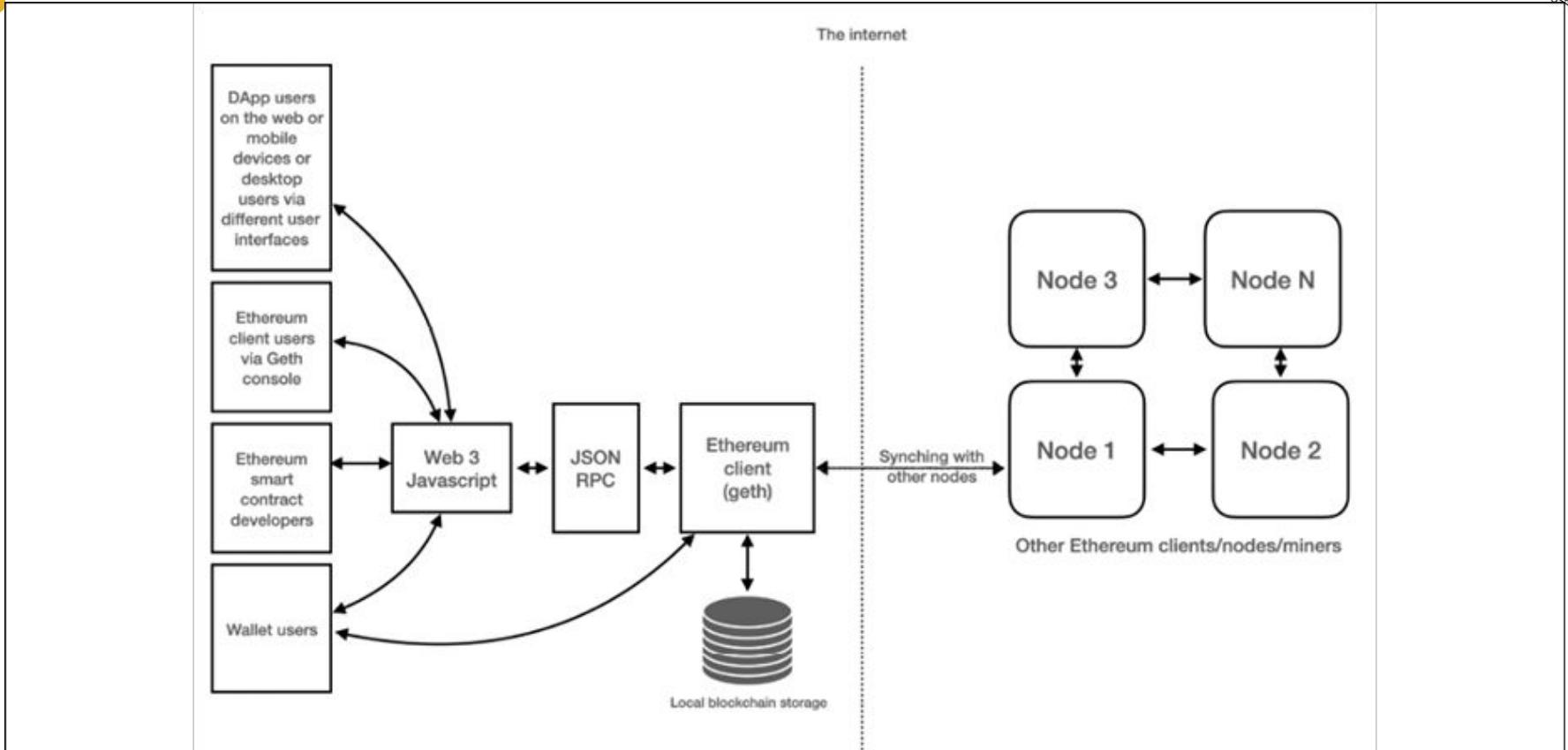
- The consensus process is controlled by a pre-defined set of nodes that are trusted.
- For example :
 - a private network of known academic institutions that each govern a single node,
 - and blocks are validated by a threshold of signatories within the network.

Note :

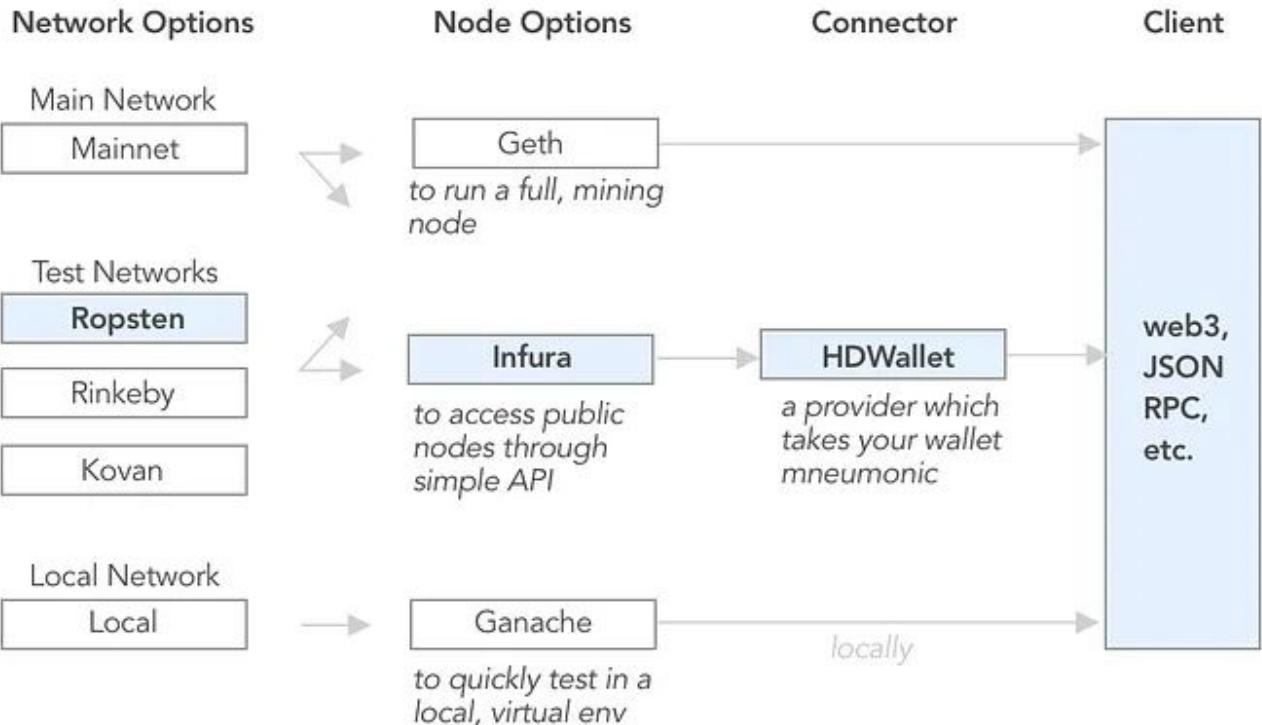
If a **public Ethereum network ⇒ public internet**, a **consortium network ⇒ private intranet**.



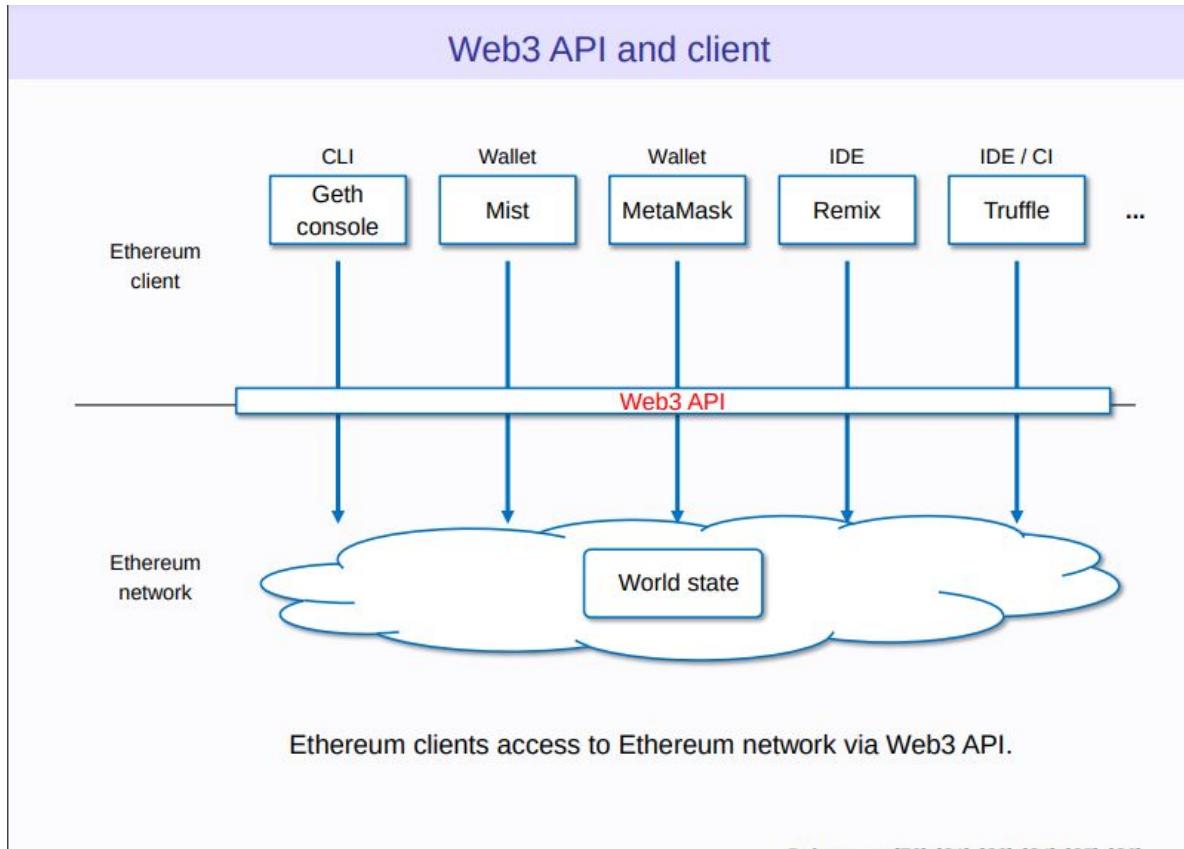
Ethereum High level Ecosystem



Ethereum Smart Contract Deployment Options



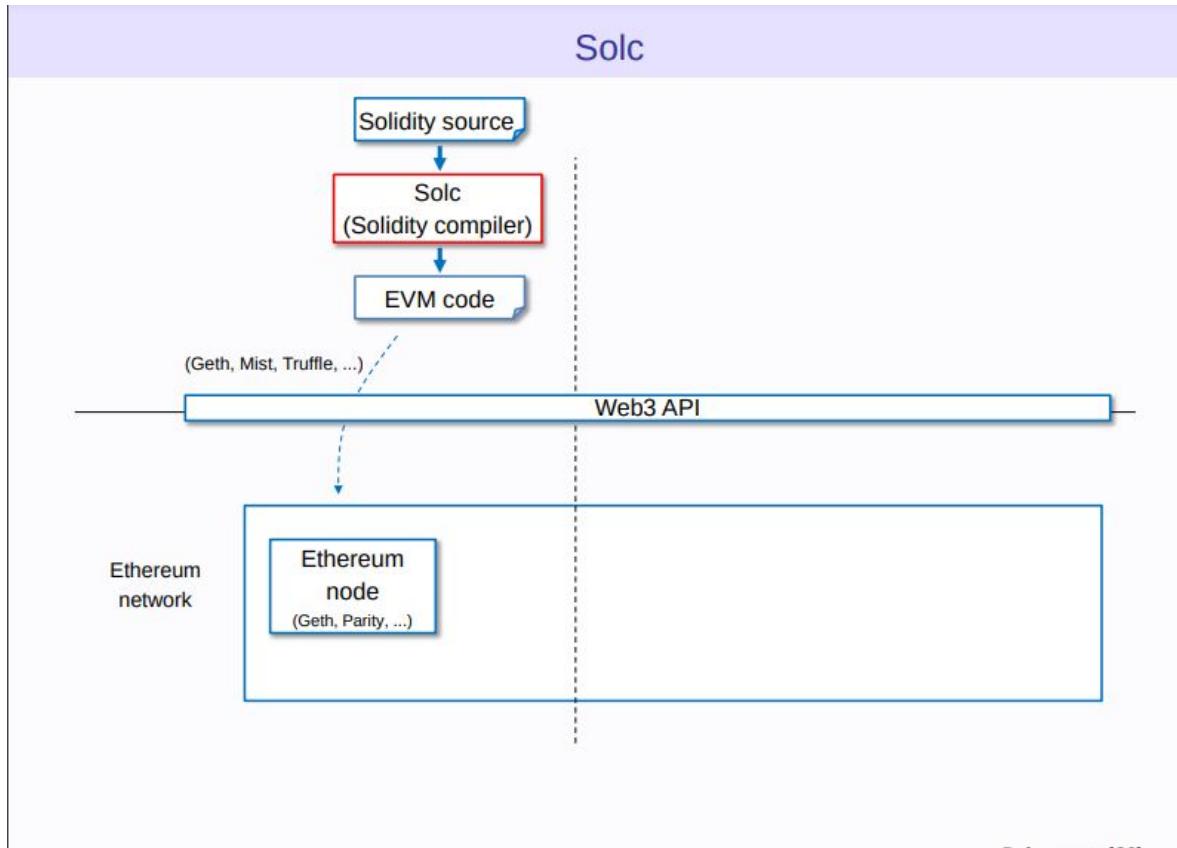
Ethereum Blockchain Deployment via Web3 API



Navigation icons: back, forward, search, etc.



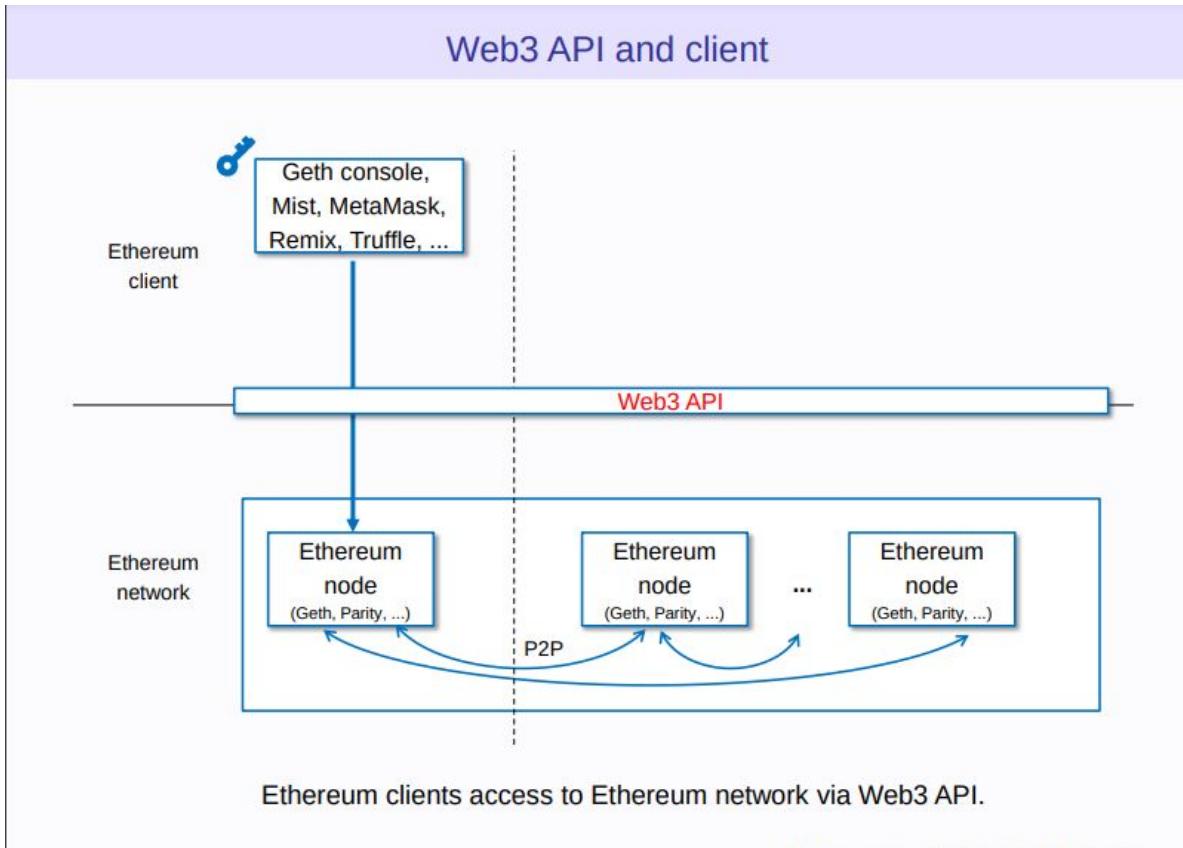
Ethereum Blockchain Deployment via Web3 API



References [60]



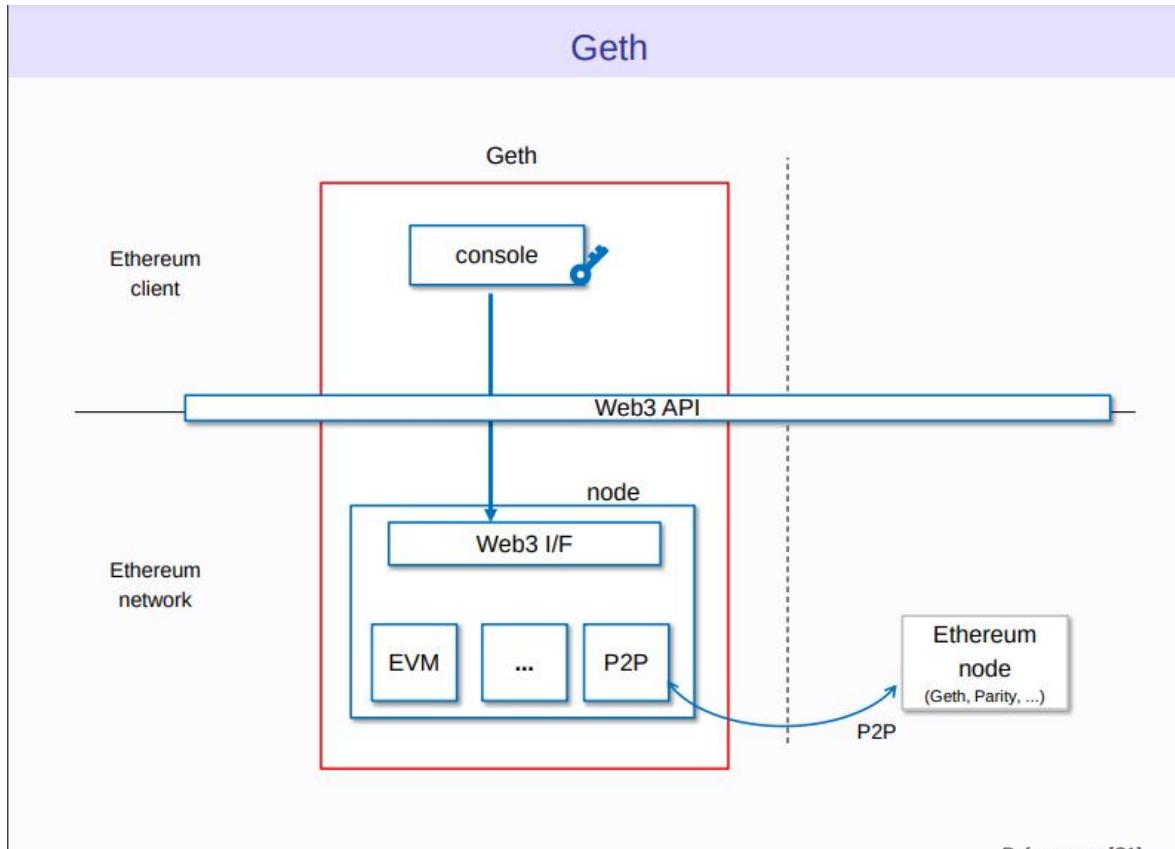
Ethereum Blockchain Deployment via Web3 API and Client



QUESTION ANSWERED



Ethereum Blockchain Deployment via Web3 API and Geth





STO Metamask (Ethereum Wallet)

- STO Metamask - Security Token Offerings using Metamask (Similar to ICO)
- a **free crypto wallet software** that people can use to interact in the crypto world.
- **free to use** and **can be installed as an extension on internet browsers**, Google Chrome, Firefox, Brave, and Edge, or downloaded as a smartphone application both on iOS and Android.
- With over **30 million users**, **MetaMask is one of the most popular cryptocurrency wallets** today.
- used for managing Ethereum and Ethereum-based tokens.
- Metamask allows users to:
 - Buy, receive, send and swap Ether, the main token on Ethereum
 - Buy, receive, send and swap nonfungible tokens (NFTs) in marketplaces
 - Connect to Ethereum dapps
 - Connect to other crypto wallets
 - Play blockchain-based games
 - Access different networks such as the BNB Smart Chain and other testnets



Benefits of using Metamask (Ethereum Wallet)

Its massive global user and follower base make it a vital part of the Ethereum community.

Utilizes local key storage, giving users full control over their keys. No personal information is stored either.

Allows in-app coin purchasing – users can buy Ether and ERC-20 directly from Coinbase and ShapeShift.

Allows multiple accounts, which is great for people who want to keep their finances separate.

Backs up the user account with hierarchical deterministic settings.

Has an easy-to-use interface that is good for crypto beginners.



Easily connects with popular exchanges like Coinbase and Binance.



METAMASK



Advantages of Metamask (Ethereum Wallet)



1. **User-Friendly Interface:** MetaMask provides a relatively intuitive and user-friendly interface, making it accessible to both beginners and experienced users.
2. **Security:** MetaMask is considered secure, with features like seed phrases for wallet recovery, password protection, and integration with hardware wallets like Ledger and Trezor for enhanced security.
3. **Cross-Platform:** It is available as a browser extension for Chrome, Firefox, Brave, and Edge, as well as a mobile app for iOS and Android, ensuring accessibility across multiple platforms.
4. **Compatibility:** MetaMask supports Ethereum and Ethereum-based tokens (ERC-20 and ERC-721 tokens), making it versatile for managing various assets within the Ethereum ecosystem.
5. **DApp Integration:** It allows users to interact with a wide range of Ethereum-based DApps directly through their browser extension, simplifying the DApp experience.
6. **Privacy:** While Ethereum is a public blockchain, MetaMask enables users to have a degree of privacy by not revealing their real-world identity when transacting or interacting with DApps.
7. **Community Support:** MetaMask has a strong community and active development, which means regular updates, bug fixes, and improvements.

Disadvantages of Metamask (Ethereum Wallet)



- Security Risks:** While MetaMask itself is secure, users can still fall victim to phishing scams and malicious websites. Users must be cautious and ensure they are using the official MetaMask extension or app.
- Limited to Ethereum:** As of my last knowledge update in September 2021, MetaMask primarily supports Ethereum and Ethereum-based tokens. While there are efforts to expand to other blockchains, it may not be the best choice for managing assets on other networks.
- Transaction Fees:** Ethereum transaction fees (gas fees) can be high during times of network congestion, making it costly to perform transactions or interact with DApps.
- Learning Curve:** For individuals new to cryptocurrencies and blockchain technology, MetaMask may have a learning curve, particularly when it comes to understanding gas fees, wallet management, and security practices.
- Lack of Customer Support:** MetaMask is an open-source project, and while it has a strong community, there is no official customer support. Users must rely on community resources for assistance.
- Centralization Concerns:** Some users may have concerns about the centralization of DApp interactions through a browser extension like MetaMask, as it relies on centralized infrastructure to communicate with the Ethereum network.



Setup a Development Environment using Metamask



1. Install MetaMask:

- install the MetaMask browser extension or mobile app on your preferred browser or device.
- You can find MetaMask on Chrome, Firefox, Brave, Edge, and as a mobile app for iOS and Android.

2. Create or Import a Wallet:

- Follow the on-screen instructions to set up a wallet, including creating a strong password and securely storing your recovery seed phrase.
- If you already have a MetaMask wallet, you can import it using your existing seed phrase or private key.

3. Switch to a Test Network:

- In a development environment, it's common to use test networks like the Ropsten, Rinkeby, or Kovan testnets to avoid spending real Ether on testing transactions.
- To switch to a test network in MetaMask:
 - Click on your MetaMask extension or app to open it.
 - Click on the network name (usually "Main Ethereum Network" by default) at the top.
 - Select "Custom RPC" if your desired test network is not listed, and then enter the network's details (such as the network name, RPC URL, and chain ID).



Setup a Development Environment using Metamask



4. Fund Your Test Wallet:

- To interact with the test network, you'll need test Ether (tEther) instead of real Ether.
- tEther can be obtained from a faucet specific to the test network you're using.
- **Faucets** are websites that provide free test Ether for development purposes. Search online for the test network's faucet and follow the instructions to get some tEther.

5. Start Developing and Testing:

- With your MetaMask wallet configured for the test network and funded with tEther, you can now start developing and testing your decentralized applications (DApps) or smart contracts.
- Use tools like Remix, Truffle, or Hardhat for Ethereum development and connect them to MetaMask to interact with your wallet and the blockchain.
- When interacting with your DApps or smart contracts, MetaMask will prompt you to confirm transactions and sign messages as needed. Remember that transactions on the test network are not real and won't affect your actual cryptocurrency holdings.
- By setting up a development environment with MetaMask, you can test your applications and smart contracts in a safe and controlled environment before deploying them to the Ethereum mainnet.

Setup a Development Environment using Metamask

C  github.com/LifnaJos/Embedding-Metamask-wallet-with-Remix-IDE-and-perform-transactions?tab=readme-ov-file#step---1--set-up

 README

Step - 1 : Set Up MetaMask:

1. Install MetaMask :

- Follow the [Link](#) to install the Google Chrome Extension.

2. Create or Import a Metamask Wallet

- To create a Metamask Wallet: Follow the [Link](#)

Note :

- While creation of the Metamask Wallet, you will be asked to select 12 pass phrase
- Keep the passphrase safely
- It becomes handy to recover your password

- To import an existing Metamask Wallet : Watch the [video](#)

3. Fund Your Wallet from any Testnet:

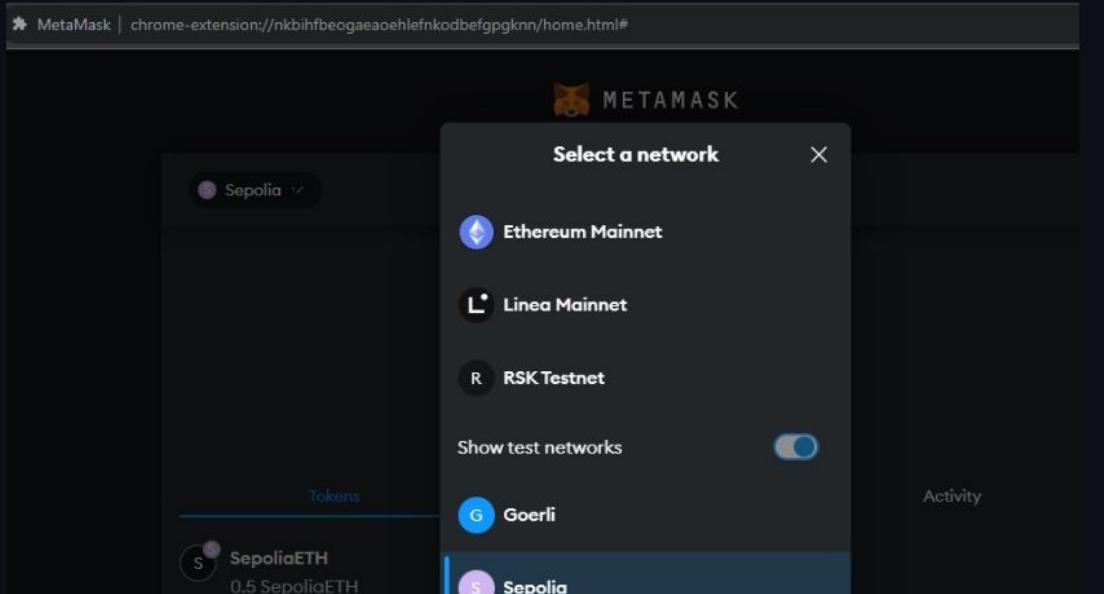
- Go through the steps to fund your wallet from [Sepolia Testnet](#) (only 0.5 ETH per day)
- Go through the steps to fund your wallet from [RSK Testnet](#) (only 0.05 RBTC per day)

Getting Funds from testnets to Metamask Wallet

□ README

Steps to get funds from Sepolia Testnet

1. Select Sepolia Testnet in Metamask @ <https://chrome-extension://nkbihfbeogaeaoehlefknkodbefgpgknn/home.html#>



The screenshot shows the Metamask extension running in a browser. The main interface has a dark theme. A modal window titled "Select a network" is displayed in the center. It lists several blockchain networks: Ethereum Mainnet, Linea Mainnet, RSK Testnet, Goerli, and Sepolia. The "Sepolia" option is currently selected, indicated by a blue bar at the bottom of the list. To the left of the modal, a sidebar shows some tokens: SepoliaETH (0.5 SepoliaETH). On the right side of the modal, there's a "Show test networks" toggle switch which is turned on (blue), and a "Activity" section.

Introduction to Ganache for Ethereum blockchain

- Ganache is a personal blockchain for rapid Ethereum and Filecoin distributed application development.
- One can use Ganache across the entire development cycle;
- enabling you to develop, deploy, and test your dApps in a safe and deterministic environment.
- Ganache comes in two flavors: a UI and CLI.
 - Ganache UI is a desktop application supporting Ethereum and Filecoin technology.
 - More robust command-line tool, ganache, is available for Ethereum development. It offers:
 - console.log in Solidity
 - Zero-config Mainnet and testnet forking
 - Fork any Ethereum network without waiting to sync
 - Ethereum JSON-RPC support
 - Snapshot/revert state
 - Mine blocks instantly, on demand, or at an interval
 - Fast-forward time
 - Impersonate any account (no private keys required!)
 - Listens for JSON-RPC 2.0 requests over HTTP/WebSockets
 - Programmatic use in Node.js
 - Pending Transactions

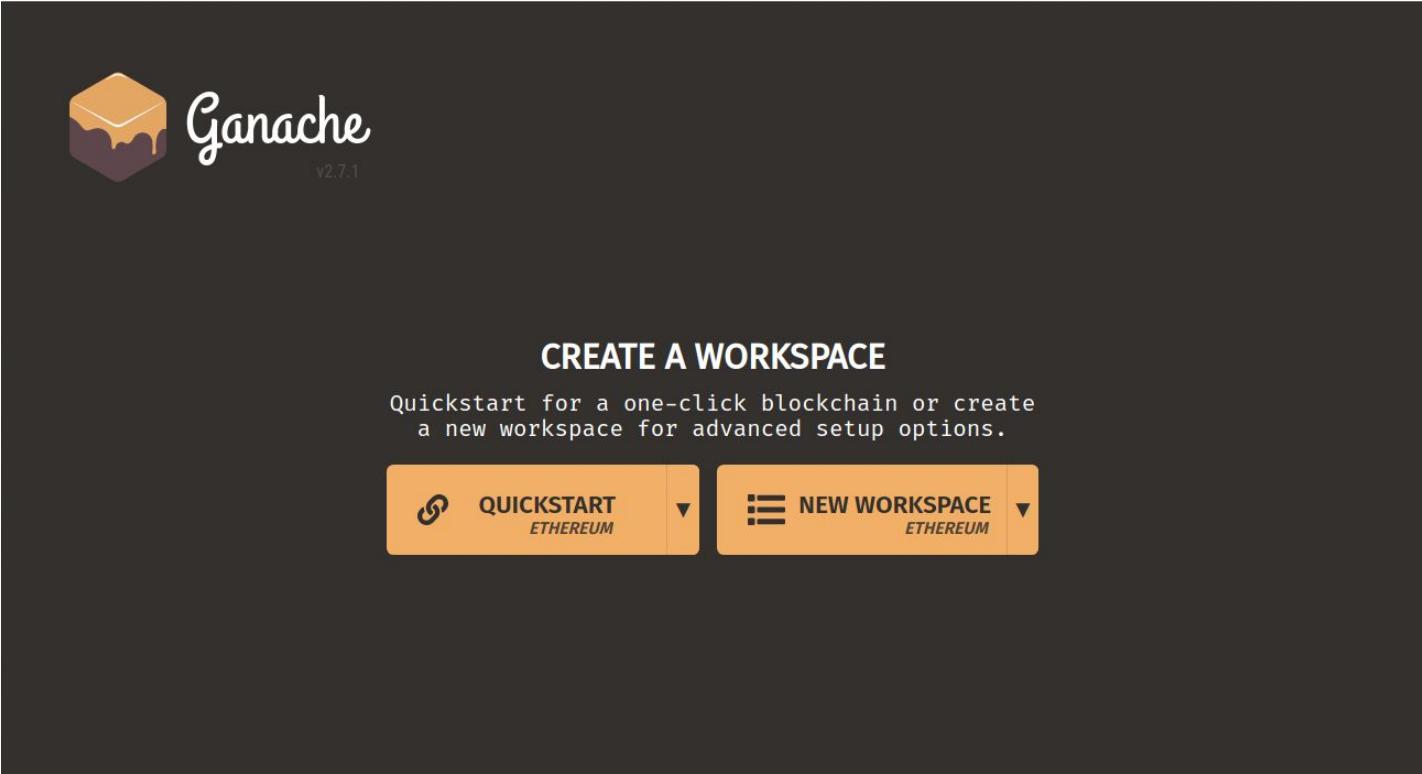
All versions of Ganache are available for Windows, Mac, and Linux.





Introduction to Ganache for Ethereum blockchain

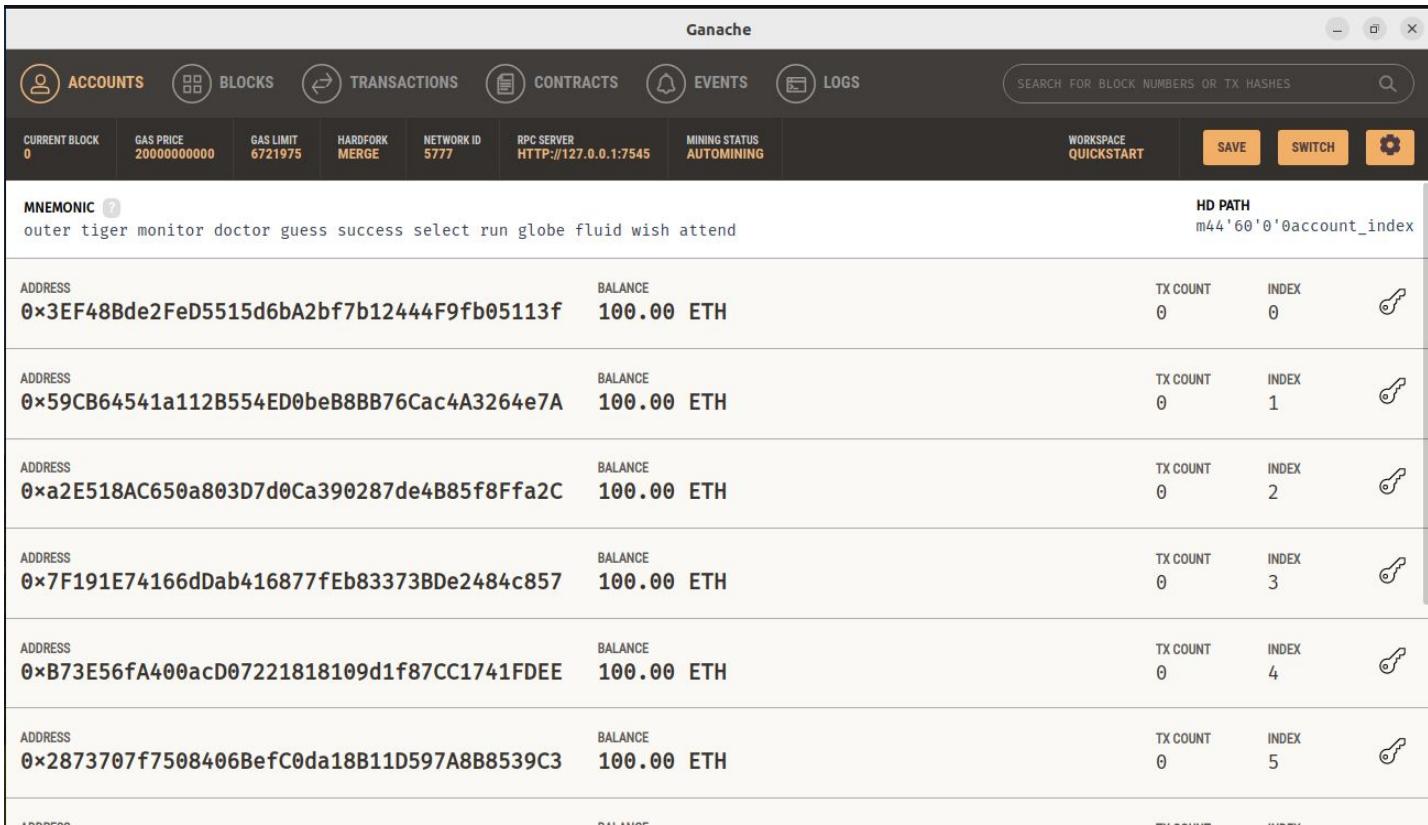
Flash screen of the Ganache after installation, Click on **Quickstart** to create a blockchain





Introduction to Ganache for Ethereum blockchain

On the Accounts tab, enlists the 10 Accounts created each with 100 Ethers



The screenshot shows the Ganache interface with the 'ACCOUNTS' tab selected. At the top, there are tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below the tabs, there are several configuration parameters: CURRENT BLOCK (0), GAS PRICE (20000000000), GAS LIMIT (6721975), HARDFORK (MERGE), NETWORK ID (5777), RPC SERVER (HTTP://127.0.0.1:7545), and MINING STATUS (AUTOMINING). There are also buttons for WORKSPACE (QUICKSTART), SAVE, SWITCH, and SETTINGS. A search bar at the top right allows for searching block numbers or tx hashes.

MNEMONIC	HD PATH
outer tiger monitor doctor guess success select run globe fluid wish attend	m44'60'0'0account_index
ADDRESS 0x3EF48Bde2FeD5515d6bA2bf7b12444F9fb05113f	BALANCE 100.00 ETH
ADDRESS 0x59CB64541a112B554ED0beB8BB76Cac4A3264e7A	BALANCE 100.00 ETH
ADDRESS 0xa2E518AC650a803D7d0Ca390287de4B85f8Ffa2C	BALANCE 100.00 ETH
ADDRESS 0x7F191E74166dDab416877fEb83373BDe2484c857	BALANCE 100.00 ETH
ADDRESS 0xB73E56fA400acD07221818109d1f87CC1741FDEE	BALANCE 100.00 ETH
ADDRESS 0x2873707f7508406BefC0da18B11D597A8B8539C3	BALANCE 100.00 ETH



Introduction to Ganache for Ethereum blockchain

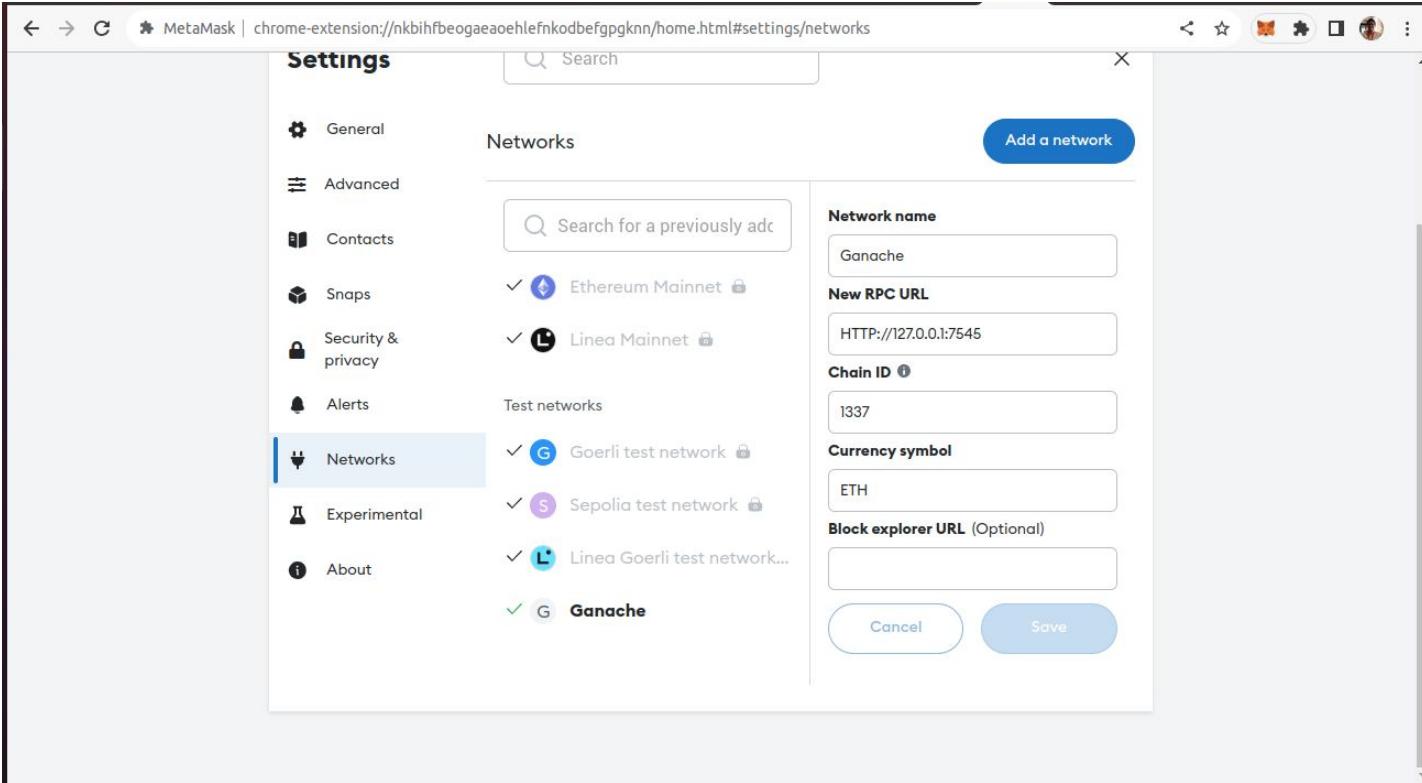
On the Blocks Tab, we could see the Genesis Block created

The screenshot shows the Ganache application window. At the top, there is a navigation bar with tabs: ACCOUNTS, BLOCKS (which is selected), TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. A search bar is located at the top right. Below the navigation bar, there is a header with various configuration settings: CURRENT BLOCK (0), GAS PRICE (20000000000), GAS LIMIT (6721975), HARDFORK (MERGE), NETWORK ID (5777), RPC SERVER (HTTP://127.0.0.1:7545), and MINING STATUS (AUTOMINING). To the right of these settings are buttons for WORKSPACE (QUICKSTART), SAVE, SWITCH, and SETTINGS. The main content area displays the Genesis Block details: BLOCK (0), MINED ON (2023-09-18 06:43:14), GAS USED (0), and NO TRANSACTIONS.



Introduction to Ganache for Ethereum blockchain

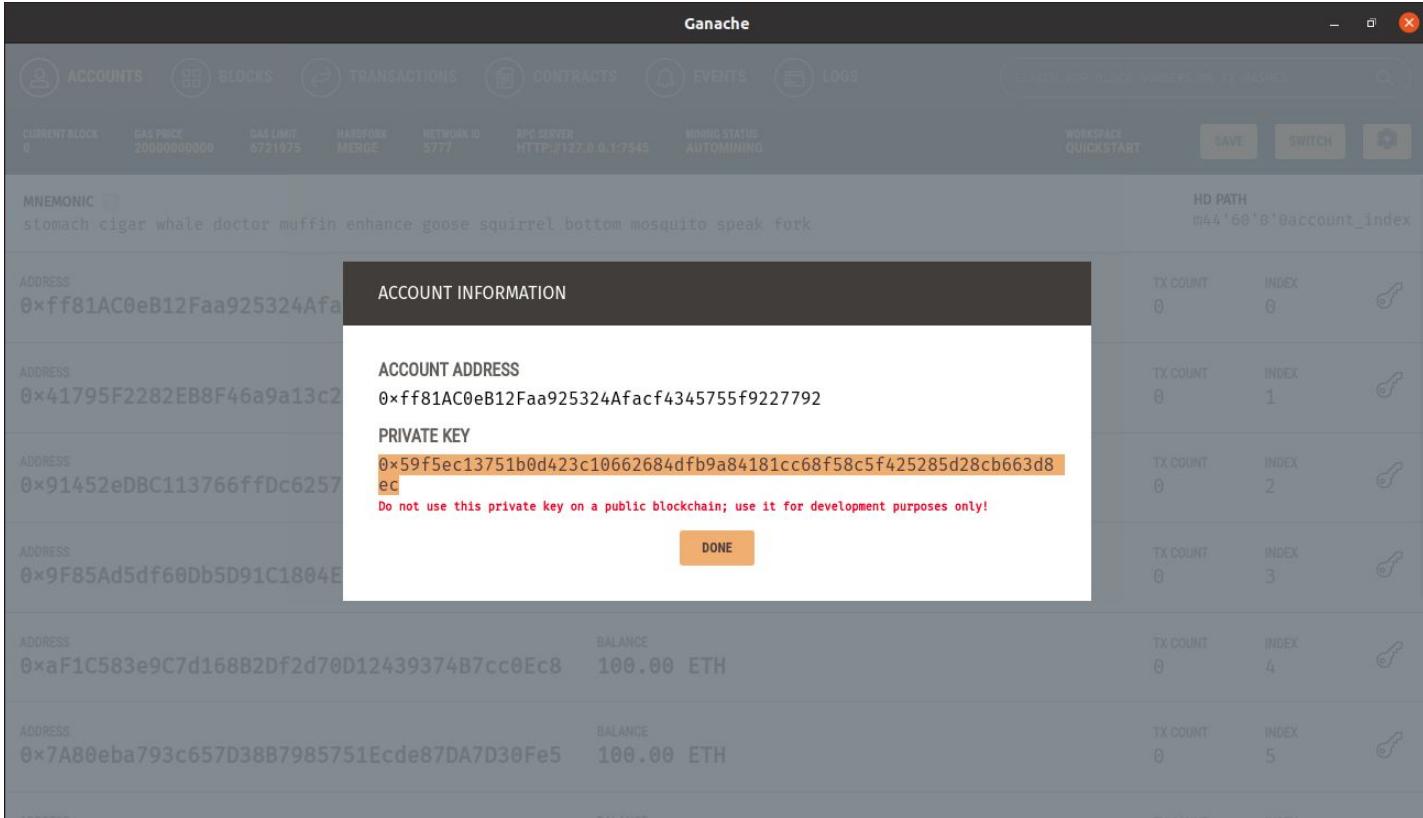
On the Metamask, add Ganache Network as follows:



The screenshot shows the MetaMask extension settings page. The left sidebar has a 'Networks' section highlighted. The main area displays a list of networks: Ethereum Mainnet, Lined Mainnet, Goerli test network, Sepolia test network, Lined Goerli test network..., and Ganache. A modal dialog box is open over the list, titled 'Add a network'. It contains fields for 'Network name' (Ganache), 'New RPC URL' (HTTP://127.0.0.1:7545), 'Chain ID' (1337), 'Currency symbol' (ETH), and 'Block explorer URL (Optional)'. At the bottom of the dialog are 'Cancel' and 'Save' buttons.

Introduction to Ganache for Ethereum blockchain

From one account, copy the Private key



The screenshot shows the Ganache application interface. At the top, there are tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below the tabs, there are status indicators: CURRENT BLOCK (0), GAS PRICE (20000000000), GAS LIMIT (6721975), HARDFORK MERGE, NETWORK ID (5777), RPC SERVER (HTTP://127.0.0.1:7545), and MINING STATUS (AUTOMINING). On the right side, there are buttons for WORKSPACE QUICKSTART, SAVE, SWITCH, and a gear icon.

In the center, a modal window titled "ACCOUNT INFORMATION" displays the following details:

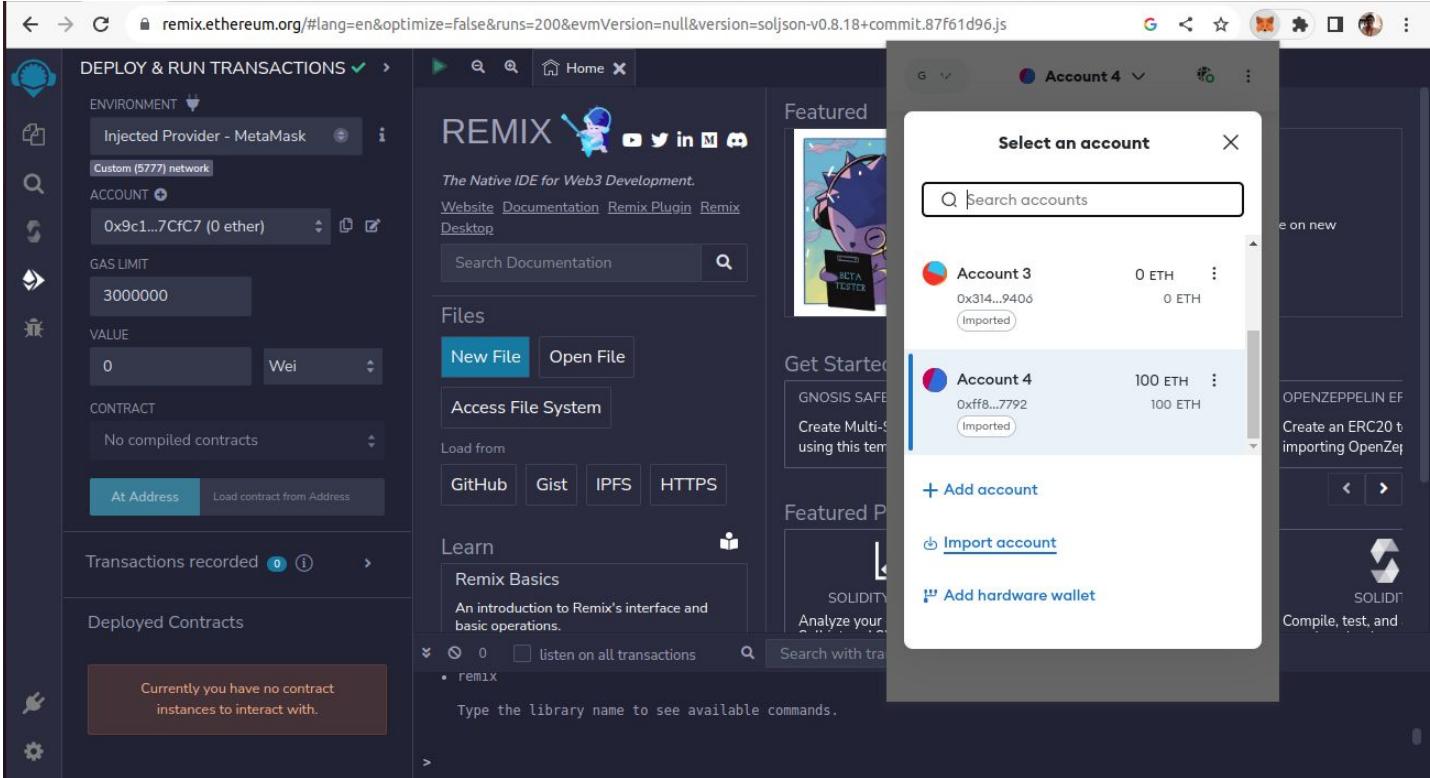
- ACCOUNT ADDRESS: 0x4f81AC0eB12Faa925324Afa
- PRIVATE KEY: 0x59f5ec13751b0d423c10662684dfb9a84181cc68f58c5f425285d28cb663d8
ec (highlighted in orange)
- Warning message: "Do not use this private key on a public blockchain; use it for development purposes only!"
- A "DONE" button at the bottom right of the modal.

Below the modal, the main table lists other accounts:

ADDRESS	BALANCE	TX COUNT	INDEX
0xaF1C583e9C7d168B2Df2d70D12439374B7cc0Ec8	100.00 ETH	0	4
0x7A80eba793c657D38B7985751Ecde87DA7D30Fe5	100.00 ETH	0	5

Introduction to Ganache for Ethereum blockchain

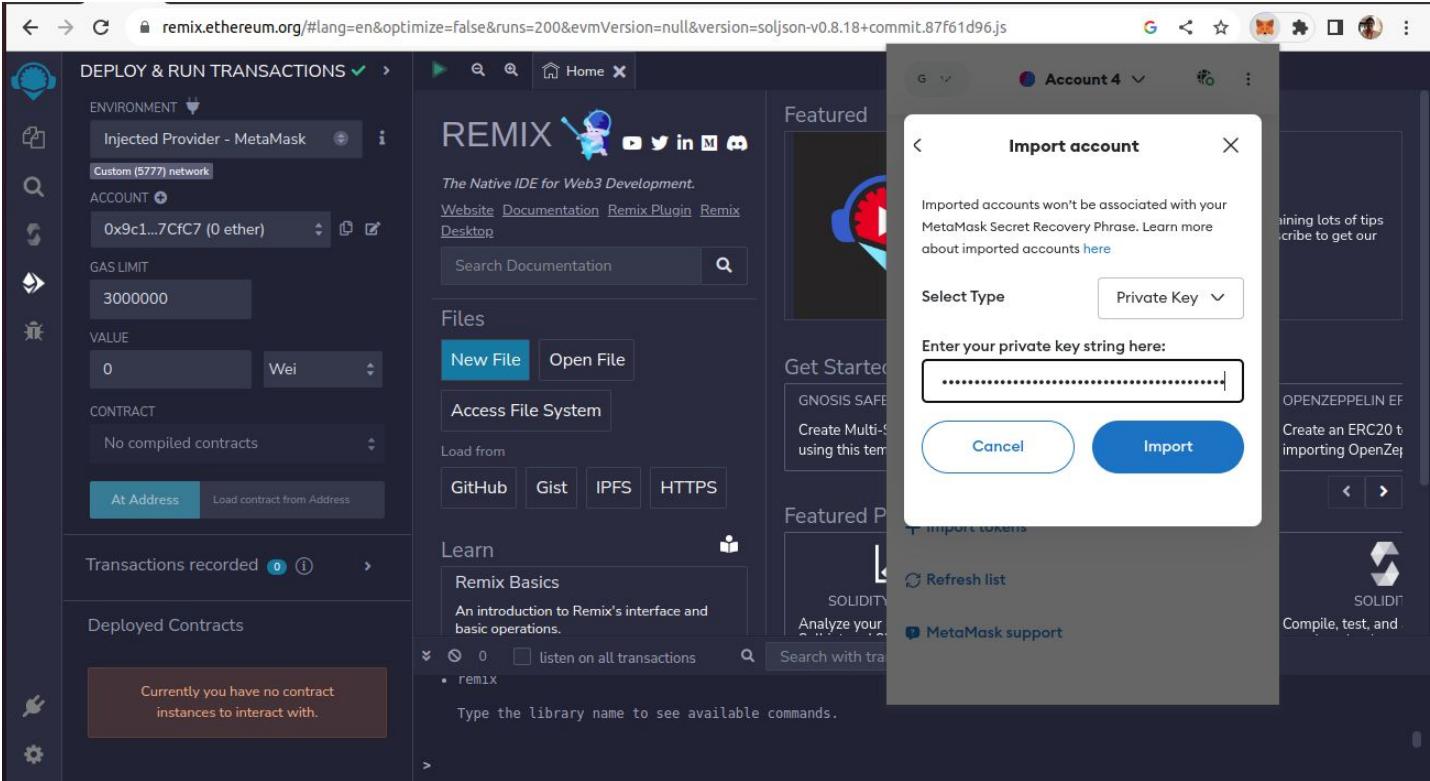
On the Metamask, select Import Account



The screenshot shows the Remix IDE interface on a web browser. The left sidebar contains options for deploying and running transactions, including environment selection (Injected Provider - MetaMask), account selection (0x9c1...7Cfc7 (0 ether)), gas limit (3000000), value (0 Wei), and contract compilation. The main area features the REMIX logo and a search bar. A modal window titled "Select an account" is open, listing two accounts: "Account 3" (0 ETH) and "Account 4" (100 ETH). Both accounts are marked as "Imported". Below the accounts, there are buttons for "+ Add account", "+ Import account", and "+ Add hardware wallet". The background of the Remix interface shows featured content like Gnosis Safe and OpenZeppelin.

Introduction to Ganache for Ethereum blockchain

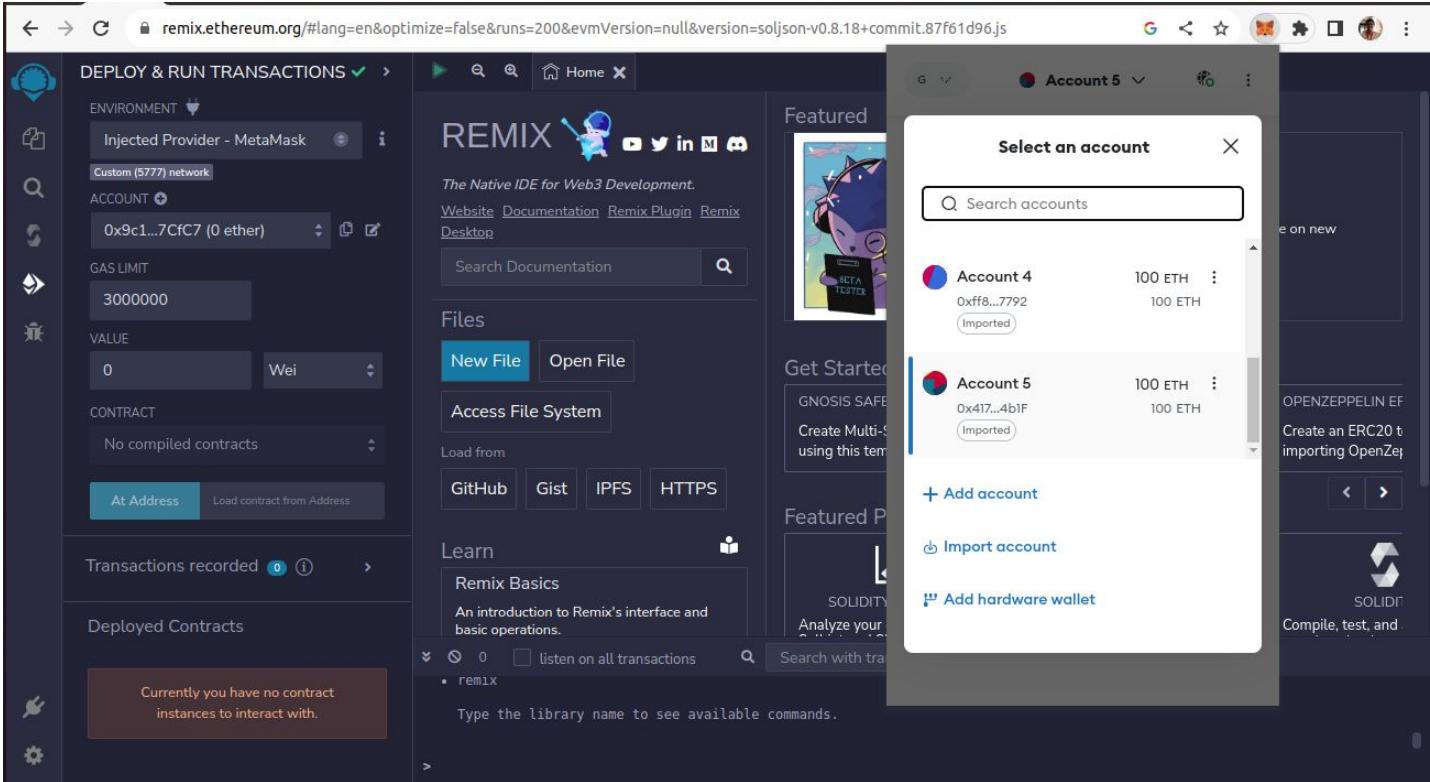
While importing Account on the Metamask, paste the Private key



The screenshot shows the Remix IDE interface on a web browser. The URL in the address bar is `remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js`. On the left, there's a sidebar with options like 'DEPLOY & RUN TRANSACTIONS', 'ENVIRONMENT' (set to 'Injected Provider - MetaMask'), 'ACCOUNT' (showing address 0x9c1...7Cfc7), 'GAS LIMIT' (3000000), 'VALUE' (0 Wei), and 'CONTRACT' (No compiled contracts). Below these are sections for 'Transactions recorded' and 'Deployed Contracts', both currently empty. The main area is titled 'REMIX' and contains sections for 'Featured', 'Get Started', 'Learn', and 'MetaMask support'. A central modal window is open, titled 'Import account'. It contains a message: 'Imported accounts won't be associated with your MetaMask Secret Recovery Phrase. Learn more about imported accounts [here](#)'. Below this is a dropdown 'Select Type' set to 'Private Key' and a text input field labeled 'Enter your private key string here:' containing a redacted private key string. At the bottom of the modal are 'Cancel' and 'Import' buttons.

Introduction to Ganache for Ethereum blockchain

Enlists the Account imported

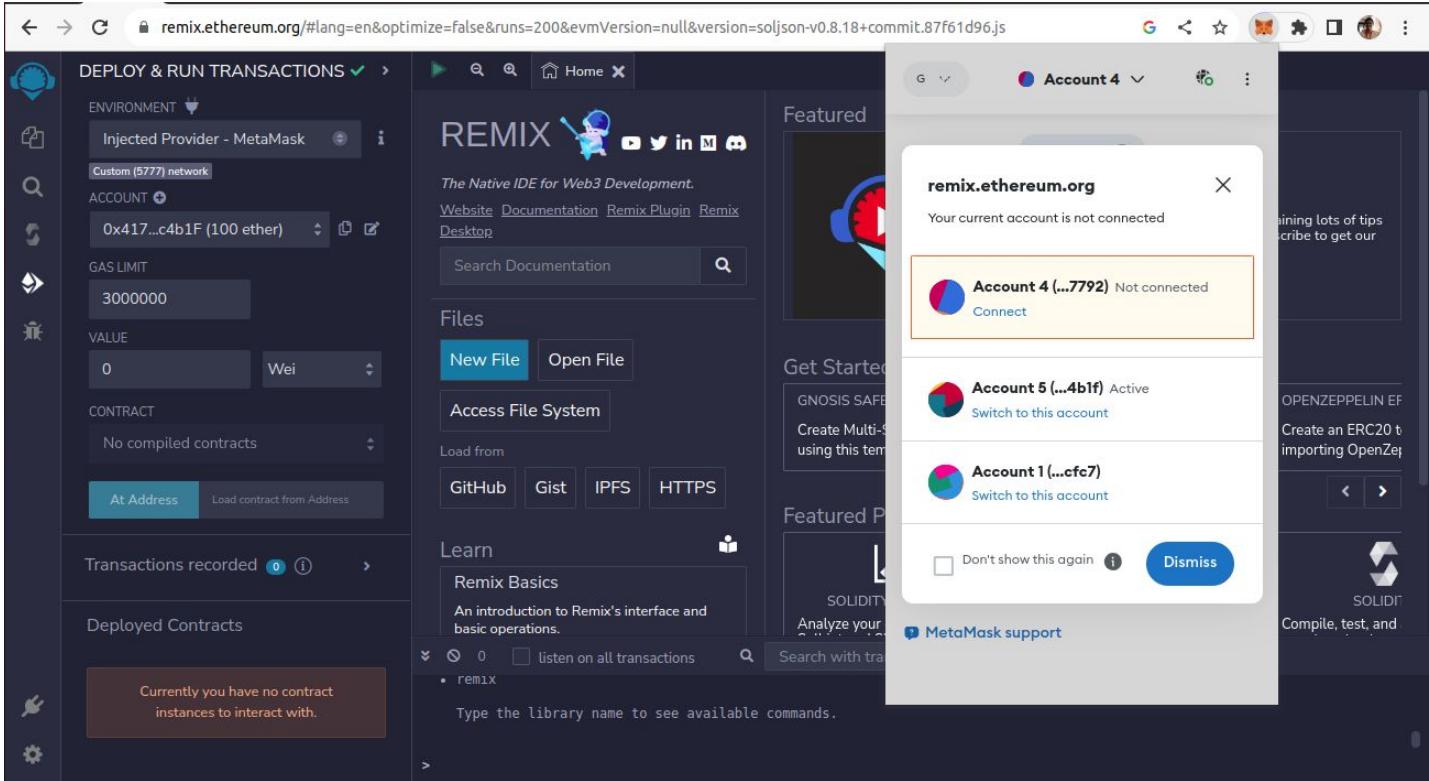


The screenshot shows the Remix IDE interface. On the left, there's a sidebar for "DEPLOY & RUN TRANSACTIONS" with fields for ENVIRONMENT (Injected Provider - MetaMask), ACCOUNT (0x9c1...7Cfc7 (0 ether)), GAS LIMIT (3000000), and VALUE (0 Wei). Below this are sections for CONTRACT (No compiled contracts) and Transactions recorded (0). Under Deployed Contracts, it says "Currently you have no contract instances to interact with." The main area is titled "REMIX" and features a search bar, documentation links, and a "Featured" section with a unicorn icon. A modal window titled "Select an account" is open in the center-right, listing two accounts: "Account 4" (0xffff...) and "Account 5" (0x417...4b1f), both with 100 ETH and labeled as "Imported". There are buttons for "+ Add account", "Import account", and "Add hardware wallet".



Introduction to Ganache for Ethereum blockchain

Select the Account from Metamask to be connected with Remix IDE

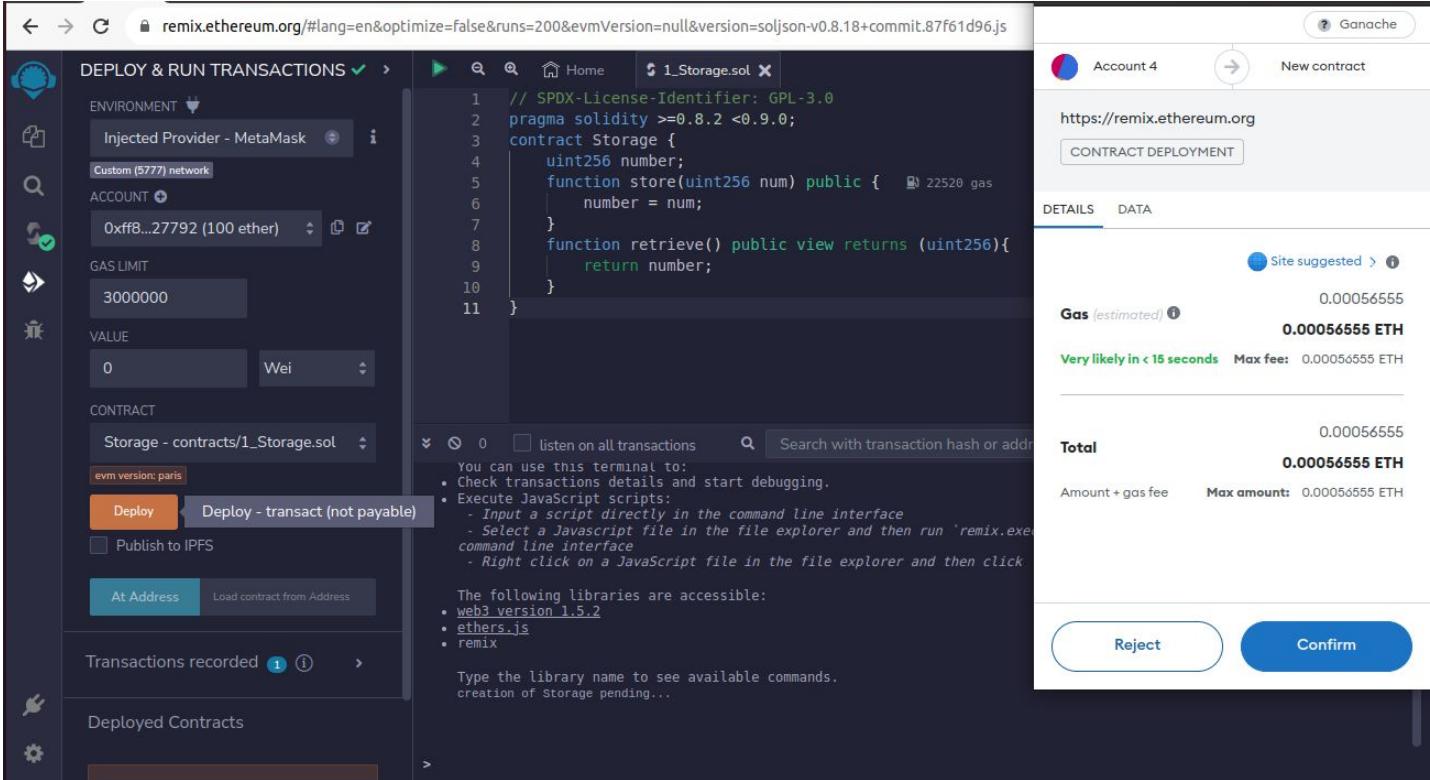


The screenshot shows the Remix IDE interface with the following details:

- Left Sidebar (DEPLOY & RUN TRANSACTIONS):**
 - ENVIRONMENT dropdown set to "Injected Provider - MetaMask".
 - ACCOUNT dropdown showing "0x417...c4b1F (100 ether)".
 - GAS LIMIT set to 3000000.
 - VALUE set to 0 Wei.
 - CONTRACT dropdown showing "No compiled contracts".
 - Buttons: "At Address" and "Load contract from Address".
 - Transactions recorded: 0.
 - Deployed Contracts: Currently you have no contract instances to interact with.
- Middle Panel:**
 - REMIX logo and search bar.
 - Featured section with links to remix.ethereum.org.
 - Get Started section with links to Gnosis Safe and Create Multi-Sig.
 - Learn section with "Remix Basics" and a link to MetaMask support.
- Right Panel (Account Selection Dialog):**
 - Header: "remix.ethereum.org" with a close button.
 - Message: "Your current account is not connected".
 - List of accounts:
 - Account 4 (...7792)** Not connected (highlighted with an orange border). [Connect](#).
 - Account 5 (...4b1f)** Active. [Switch to this account](#).
 - Account 1 (...fcf7)**. [Switch to this account](#).
 - Checkboxes: "Don't show this again" and "Dismiss".

Introduction to Ganache for Ethereum blockchain

Deploy the Smart Contract



The screenshot shows the Remix IDE interface for deploying a Solidity smart contract named `1_Storage.sol`. The code defines a `Storage` contract with two functions: `store(uint256 num)` and `retrieve()`.

Left Panel (DEPLOY & RUN TRANSACTIONS):

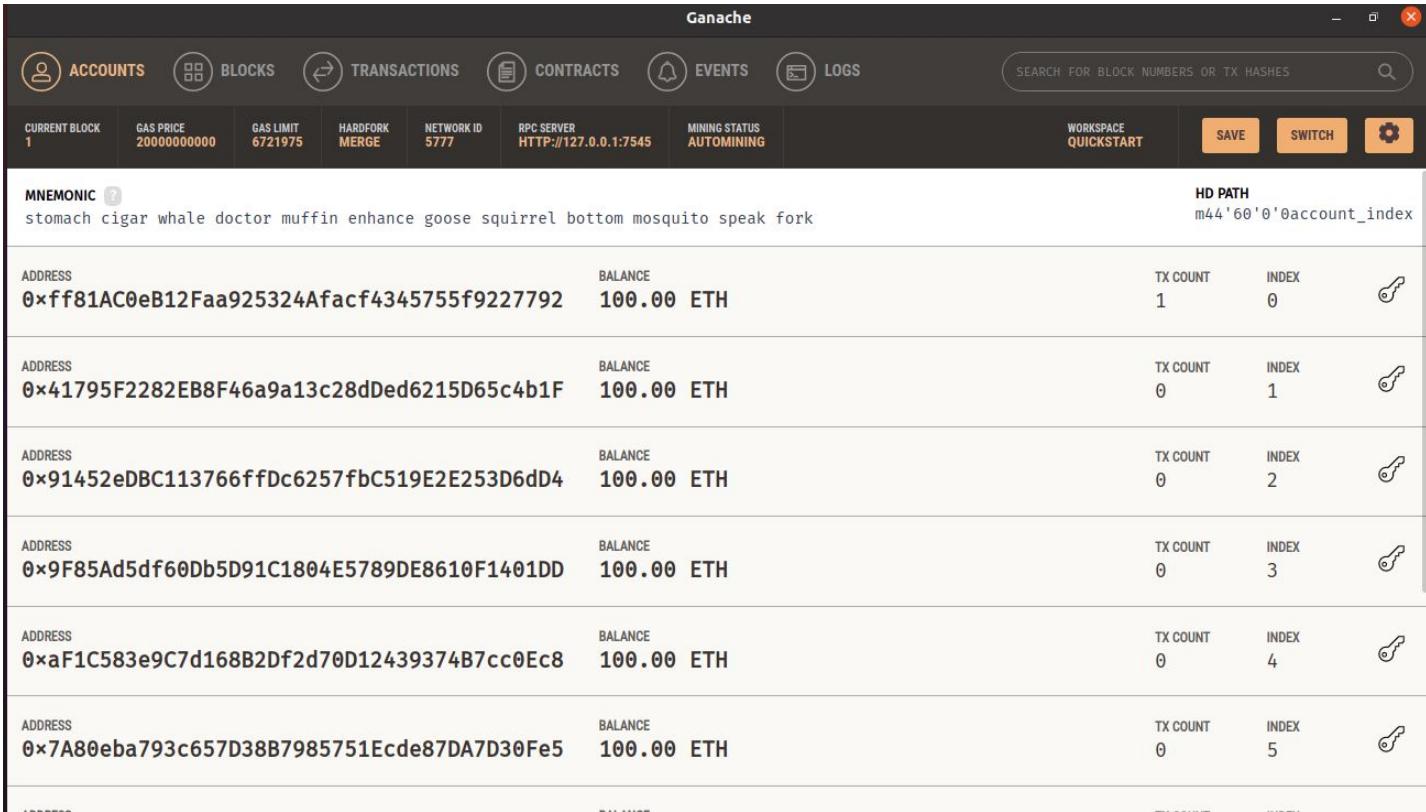
- ENVIRONMENT:** Injected Provider - MetaMask, Custom (5777) network.
- ACCOUNT:** 0x...27792 (100 ether).
- GAS LIMIT:** 3000000.
- VALUE:** 0 Wei.
- CONTRACT:** Storage - contracts/1_Storage.sol, evm version: paris.
- Buttons:** Deploy (highlighted), Deploy - transact (not payable), Publish to IPFS.
- Terminal:** You can use this terminal to:
 - Check transactions details and start debugging.
 - Execute JavaScript scripts:
 - Input a script directly in the command line interface
 - Select a Javascript file in the file explorer and then run 'remix.execute' command line interface
 - Right click on a JavaScript file in the file explorer and then click
- Information:** The following libraries are accessible:
 - web3 version 1.5.2
 - ethers.js
 - remix
- Text:** Type the library name to see available commands.

Right Panel (Contract Deployment):

- Account 4:** Ganache.
- URL:** https://remix.ethereum.org
- Buttons:** CONTRACT DEPLOYMENT, DETAILS, DATA.
- Gas (estimated):** 0.00056555 ETH.
- Total:** 0.00056555 ETH.
- Amount + gas fee:** Max amount: 0.00056555 ETH.
- Buttons:** Reject, Confirm.

Introduction to Ganache for Ethereum blockchain

On the Ganache Environment, the Transaction count is updated



The screenshot shows the Ganache interface with the following account details:

ADDRESS	BALANCE	TX COUNT	INDEX
0xff81AC0eB12Faa925324Afacf4345755f9227792	100.00 ETH	1	0
0x41795F2282EB8F46a9a13c28dDed6215D65c4b1F	100.00 ETH	0	1
0x91452eDBC113766ffDc6257fbC519E2E253D6dD4	100.00 ETH	0	2
0x9F85Ad5df60Db5D91C1804E5789DE8610F1401DD	100.00 ETH	0	3
0xaF1C583e9C7d168B2Df2d70D12439374B7cc0Ec8	100.00 ETH	0	4
0x7A80eba793c657D38B7985751Ecde87DA7D30Fe5	100.00 ETH	0	5



Introduction to Ganache for Ethereum blockchain

Block 1 is added to the Blockchain which displays the Contract Created

The screenshot shows the Ganache interface with the following details:

Block 1 Details:

GAS USED	GAS LIMIT	MINED ON	BLOCK HASH
125677	6721975	2023-09-18 11:51:42	0x180ca30613d3bb735b09dc9d0d11f2e9178c06ce1335ebe797c74540168a8545

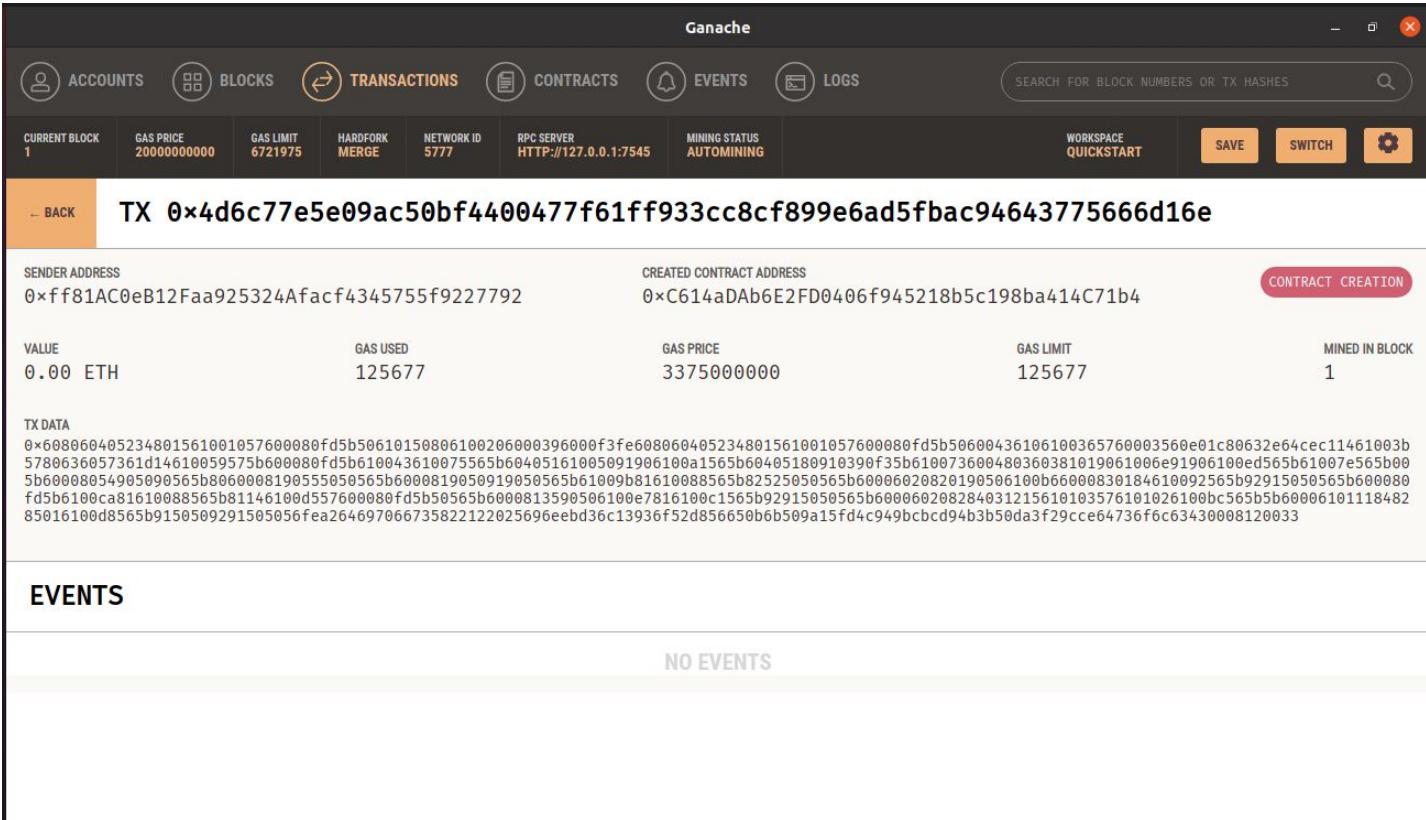
Contract Creation:

FROM ADDRESS	CREATED CONTRACT ADDRESS	GAS USED	VALUE
0xff81AC0eB12Faa925324Afacf4345755f9227792	0xC614aDAb6E2FD0406f945218b5c198ba414C71b4	125677	0



Introduction to Ganache for Ethereum blockchain

Transaction details of the Contract is displayed on Ganache Environment



The screenshot shows the Ganache application interface. At the top, there are tabs for ACCOUNTS, BLOCKS, TRANSACTIONS (which is selected), CONTRACTS, EVENTS, and LOGS. Below the tabs, there are several status indicators: CURRENT BLOCK (1), GAS PRICE (20000000000), GAS LIMIT (6721975), HARDFORK MERGE, NETWORK ID (5777), RPC SERVER (HTTP://127.0.0.1:7545), MINING STATUS (AUTOMINING), and WORKSPACE QUICKSTART. There are also buttons for SAVE, SWITCH, and SETTINGS.

TX 0x4d6c77e5e09ac50bf4400477f61ff933cc8cf899e6ad5fbac94643775666d16e

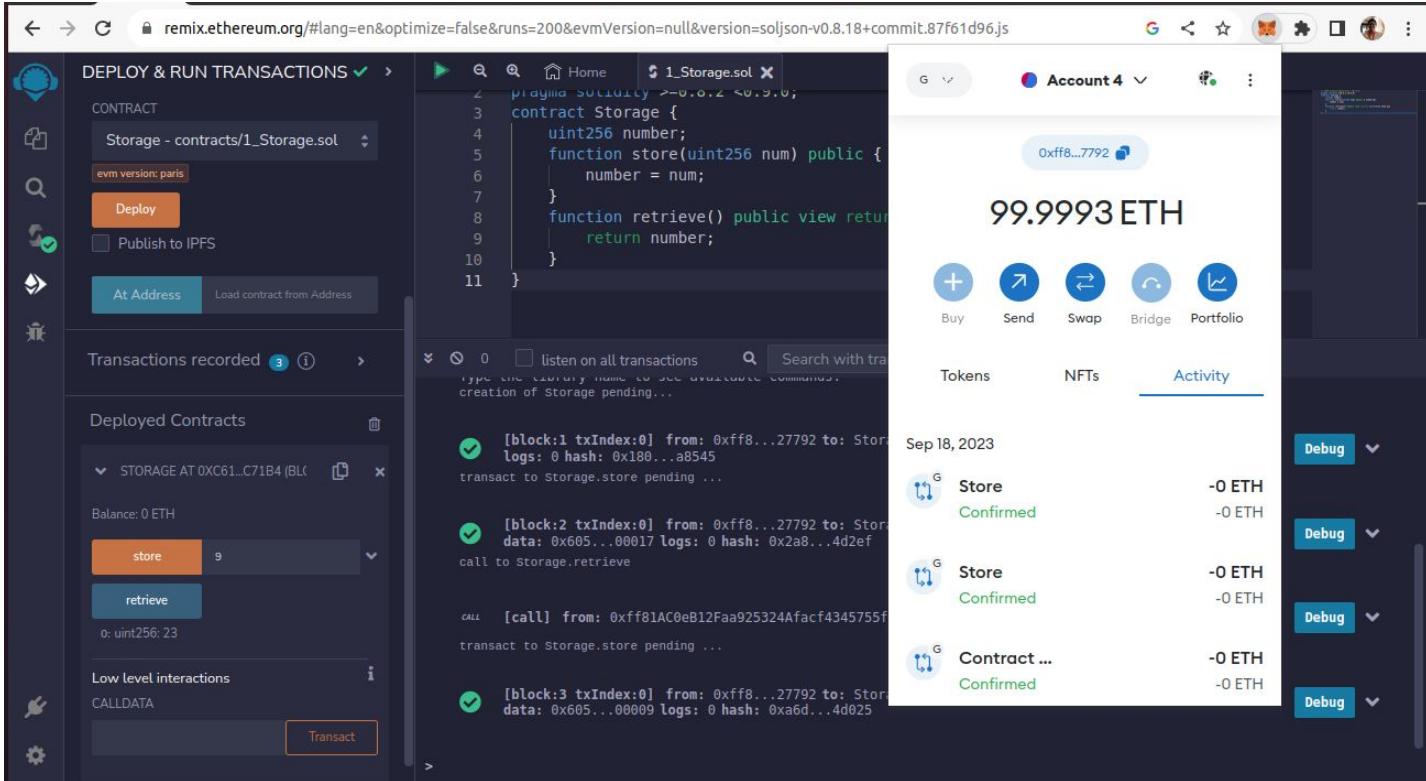
SENDER ADDRESS	CREATED CONTRACT ADDRESS	CONTRACT CREATION		
0xff81AC0eB12Faa925324Afacf4345755f9227792	0xC614aDAbE2FD0406f945218b5c198ba414C71b4			
VALUE	GAS USED	GAS PRICE	GAS LIMIT	MINED IN BLOCK
0.00 ETH	125677	3375000000	125677	1

TX DATA
0x608060405234801561001057600080fd5b50610150806100206000396000f3fe608060405234801561001057600080fd5b50600436106100365760003560e01c80632e64cec11461003b5780636057361d14610059575b600080fd5b610043610075565b60405161005091906100a1565b60405180910390f35b61007360048036038101906100e91906100ed565b61007e565b005b60008054905090565b8060008190555050565b6000819050919050565b61009b81610088565b82525050565b60006020820190506100b66000830184610092565b92915050565b600080fd5b6100ca81610088565b81146100d557600080fd5b50565b6000813590506100e7816100c1565b92915050565b600060208284031215610103576101026100b565b56000610111848285016100d8565b9150509291505056fea264697066735822122025696eebd36c13936f52d856650b6b509a15fd4c949bc94b3b50da3f29cce64736f6c63430008120033

EVENTS
NO EVENTS

Introduction to Ganache for Ethereum blockchain

After interacting with the Smart Contract, funds are updated on the Metamask



The screenshot shows the Ganache interface on the left and the Metamask extension on the right.

Ganache Interface:

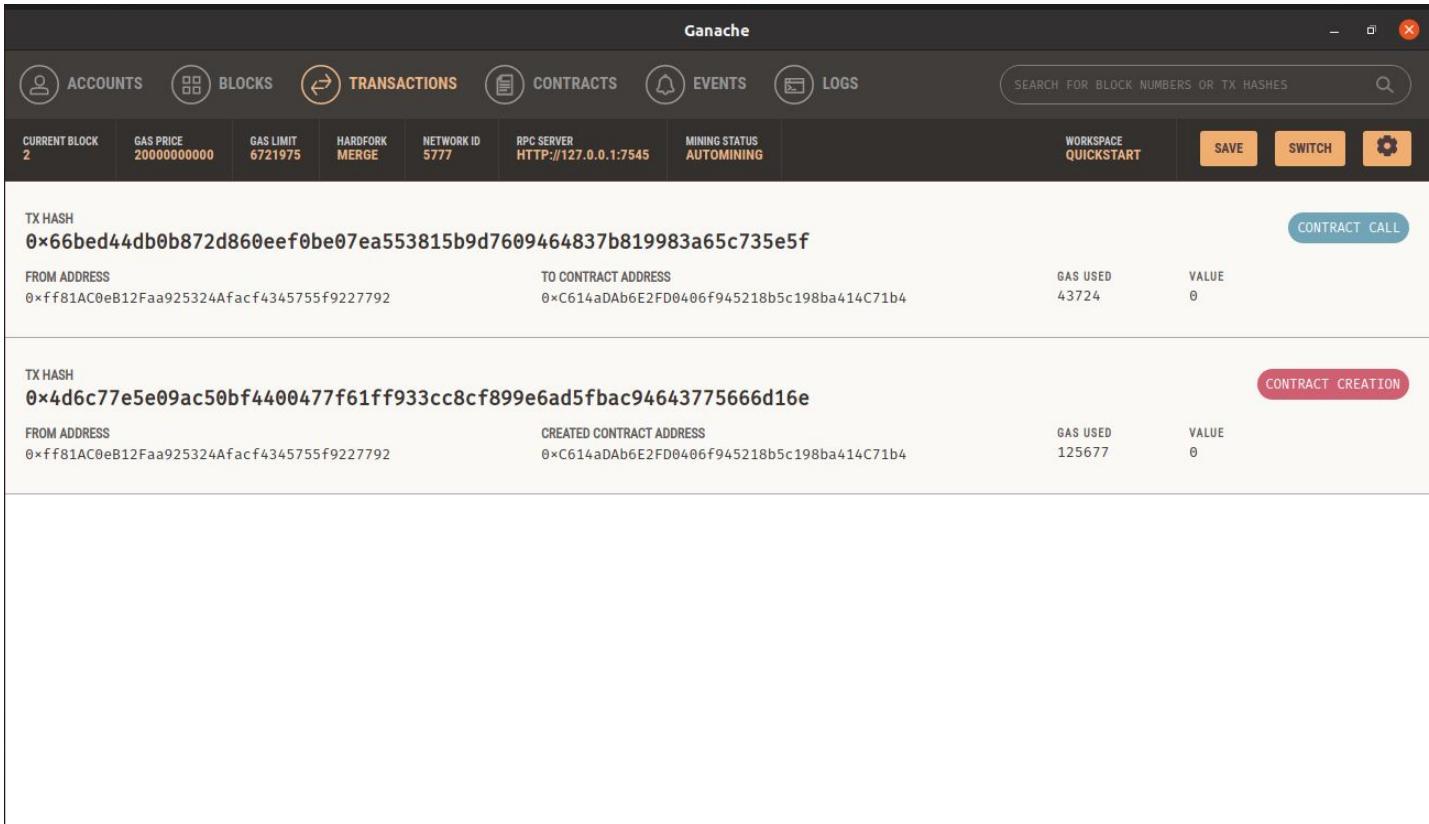
- Deploy & Run Transactions:** Shows the deployed Storage contract at address 0x61...C71B4.
- Transactions recorded:** Lists three transactions:
 - [block:1 txIndex:0] from: 0xff8...27792 to: Storage store pending ...
 - [block:2 txIndex:0] from: 0xff8...27792 to: Storage retrieve call to Storage.retrieve
 - [call] from: 0x81AC0eB12Faa925324Afacf4345755f transact to Storage.store pending ...
- Low level interactions:** Shows a store operation with value 9.

Metamask Extension:

- Account 4:** Address 0xff8...7792, Balance: 99.9993 ETH.
- Activity:** Shows the three transactions listed in the Ganache interface as confirmed operations.

Introduction to Ganache for Ethereum blockchain

On the Ganache, the Contract call is listed



The screenshot shows the Ganache interface with the following details:

TX HASH: 0x66bed44db0b872d860eef0be07ea553815b9d7609464837b819983a65c735e5f (Contract Call)

FROM ADDRESS	TO CONTRACT ADDRESS	GAS USED	VALUE
0xff81AC0eB12Faa925324Afacf4345755f9227792	0xC614aDAb6E2FD0406f945218b5c198ba414C71b4	43724	0

TX HASH: 0x4d6c77e5e09ac50bf4400477f61ff933cc8cf899e6ad5fbac94643775666d16e (Contract Creation)

FROM ADDRESS	CREATED CONTRACT ADDRESS	GAS USED	VALUE
0xff81AC0eB12Faa925324Afacf4345755f9227792	0xC614aDAb6E2FD0406f945218b5c198ba414C71b4	125677	0

ETHEREUM

Development tools, Frameworks,
Libraries



 OpenZeppelin





Ethereum Frameworks



1. Truffle Suite:

- a. **Truffle**: A development environment, testing framework, and asset pipeline for Ethereum.
- b. **Ganache**: A personal blockchain for Ethereum development & used to deploy contracts, develop your applications, and run tests.

2. **Hardhat**: A development environment for Ethereum that allows to compile, deploy, test, and debug your smart contracts.

3. **Embark**: A framework for building and deploying decentralized applications (DApps) on Ethereum.

4. **Brownie**: A Python-based framework for Ethereum smart contract development, testing, and deployment.

5. **Waffle**: A testing and development framework for Ethereum smart contracts that focuses on simplicity and extensibility.

6. OpenZeppelin:

- a. A library for secure smart contract development.
- b. It provides reusable, secure smart contract components.





Since 1962

Ethereum Frameworks

University of Mumbai



7. ethers.js:

- A JavaScript library for interacting with the Ethereum blockchain.
- It simplifies the process of interacting with smart contracts and sending transactions.

8. Web3.js:

- Another JavaScript library for interacting with the Ethereum blockchain.
- It is widely used for building decentralized applications.

9. Drizzle:

- A front-end library for integrating Ethereum decentralized applications with React.

10. Remix:

- An open-source web and desktop application that helps developers write, test, and deploy smart contracts.

11. Infura:

- A cloud infrastructure service that provides Ethereum nodes as a service.
- It allows developers to connect to the Ethereum network without running their own nodes.

12. ENS (Ethereum Name Service):

- A decentralized domain name system built on the Ethereum blockchain.
- It simplifies the process of associating human-readable names with Ethereum addresses.





Ethereum Frameworks



1. Truffle Suite:

- **Features:**

- Development environment, testing framework, and asset pipeline for Ethereum.
- Built-in smart contract compilation, linking, deployment, and binary management.
- TestRPC for local testing and debugging.

- **Applications:**

- Rapid smart contract development and testing.
- Automated testing of Ethereum contracts.

- **Pros:**

- Comprehensive development suite.
- Large community support.

- **Cons:**

- Steeper learning curve for beginners.



2. Embark:

- **Features:**

- Framework for building and deploying decentralized applications (dApps).
- Smart contract development, deployment, and testing.
- Integrated web server for easy dApp development.

- **Applications:**

- Full-stack decentralized application development.
- Simplified deployment process.

- **Pros:**

- Easy to use for both beginners and experienced developers.
- Built-in web server for streamlined development.

- **Cons:**

- Smaller community compared to Truffle.





Ethereum Frameworks



3. Hardhat:

- **Features:**

- Ethereum development environment with task automation.
- Advanced development environment with support for TypeScript.
- Extensive plugin system for additional functionality.

- **Applications:**

- Ethereum smart contract development.
- Task automation and extensibility.

- **Pros:**

- Solid TypeScript support.
- Flexible and extensible through plugins.

- **Cons:**

- May be less beginner-friendly.





Ethereum Frameworks



4. Brownie:

- **Features:**

- Python-based framework for Ethereum development.
- Simple and Pythonic syntax for smart contract development.
- Built-in testing and deployment tools.

- **Applications:**

- Ethereum smart contract development using Python.
- Rapid prototyping and testing.

- **Pros:**

- Python-friendly syntax.
- Easy to learn for Python developers.

- **Cons:**

- Smaller community compared to more established frameworks.



5. OpenZeppelin:

- **Features:**

- Open-source framework for building secure smart contracts.
- Provides reusable, community-audited smart contract components.
- Follows best practices for secure contract development.

- **Applications:**

- Building secure and audited smart contracts.
- Reusing tested and secure components.

- **Pros:**

- Emphasis on security.
- Large library of reusable smart contract components.

- **Cons:**

- Not a complete development suite; often used in conjunction with other frameworks.



Ethereum 2.0 / ETH 2.0 / Serenity - Key features

- aims to address scalability, security, and sustainability issues.
- **Beacon Chain:**
 - first phase of Ethereum 2.0 and was launched in December 2020.
 - Introduces a PoS consensus mechanism
 - where validators are chosen to create new blocks and validate transactions based on the amount of cryptocurrency they "stake" as collateral.
- **Shard Chains:**
 - will run in parallel to the existing Ethereum mainnet.
 - aim to improve scalability by processing transactions and smart contracts in smaller groups, or shards, rather than on a single chain.





Since 1962

Ethereum 2.0 / ETH 2.0 / Serenity - Key features



- **eWASM (Ethereum WebAssembly):**
 - replace the current Ethereum Virtual Machine (EVM) with eWASM.
 - expected to bring improved efficiency, faster execution of smart contracts, and greater flexibility for developers.
- **Crosslinks:**
 - connect shard chains to the Beacon Chain,
 - ensuring communication and data transfer between different shards.
 - **Data Availability:** Helps maintain the security of the network by providing a mechanism for validators to agree on the state of different shard chains.
- **Migration from PoW to PoS:**
 - The transition from PoW to PoS will occur in multiple phases.
 - PoS is more energy-efficient and allows users to earn rewards by locking up cryptocurrency as collateral, contributing to network security.



Ethereum 2.0 / ETH 2.0 / Serenity - Key features

- **Docking and Withdrawal:**
 - Allows Ethereum users to "dock" their existing ETH into the Ethereum 2.0 system, ***converting it into a special token called "BETH."***
 - Users can withdraw their ETH from the Ethereum 2.0 system back to the Ethereum 1.0 network.
- **Timeline:**
 - ***Ongoing Development:*** Ethereum 2.0 is a complex and multi-phase upgrade, and development is ongoing. Different phases will be rolled out incrementally as they are ready.
 - ***Full Deployment:*** Full deployment of Ethereum 2.0 is expected to take several years, with continuous updates and improvements.
- **Community and Research:**
 - open source, with contributions from a large community of developers and researchers.
 - Research-Oriented: The upgrade is the result of extensive research and collaboration within the Ethereum community.



Ethereum 2.0 / ETH 2.0 / Serenity - Objectives



Scalability
Security
Eco-friendly

"SERENITY"
ETHEREUM 2.0



ETH2

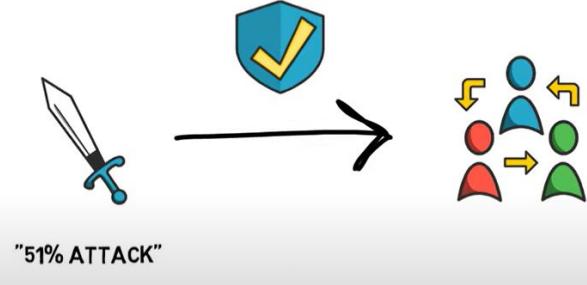


Ethereum 2.0 / ETH 2.0 / Serenity - Objectives

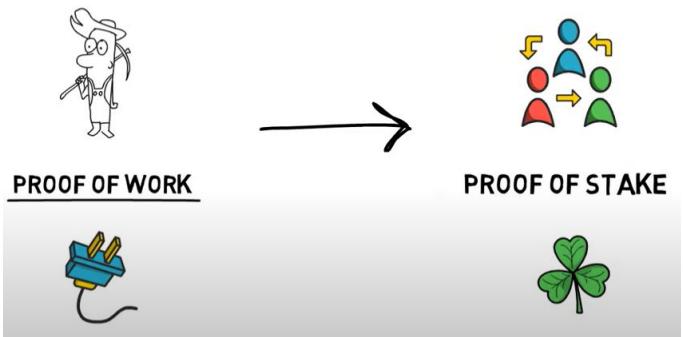
SCALABILITY 🛒

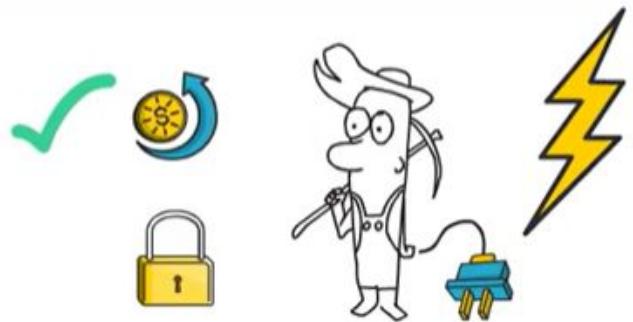


SECURITY 🔒



SUSTAINABILITY 🌱





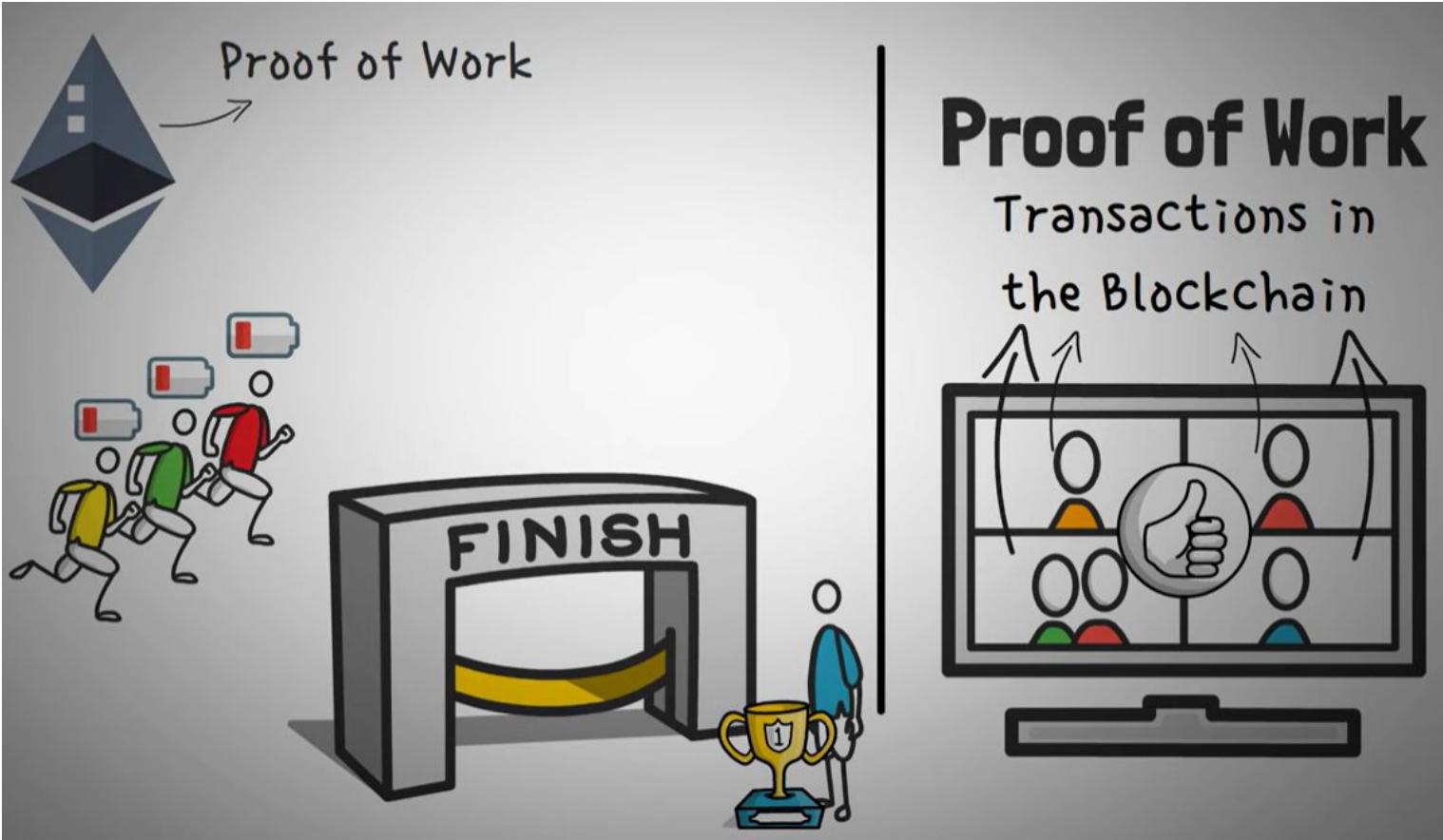
PROOF OF WORK



PROOF OF STAKE

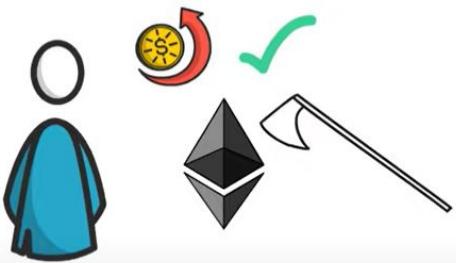


Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS



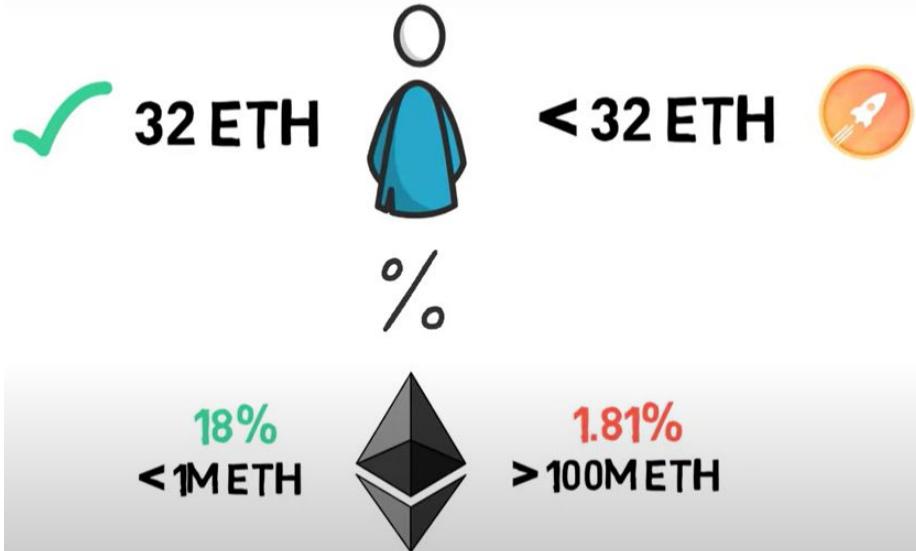
Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

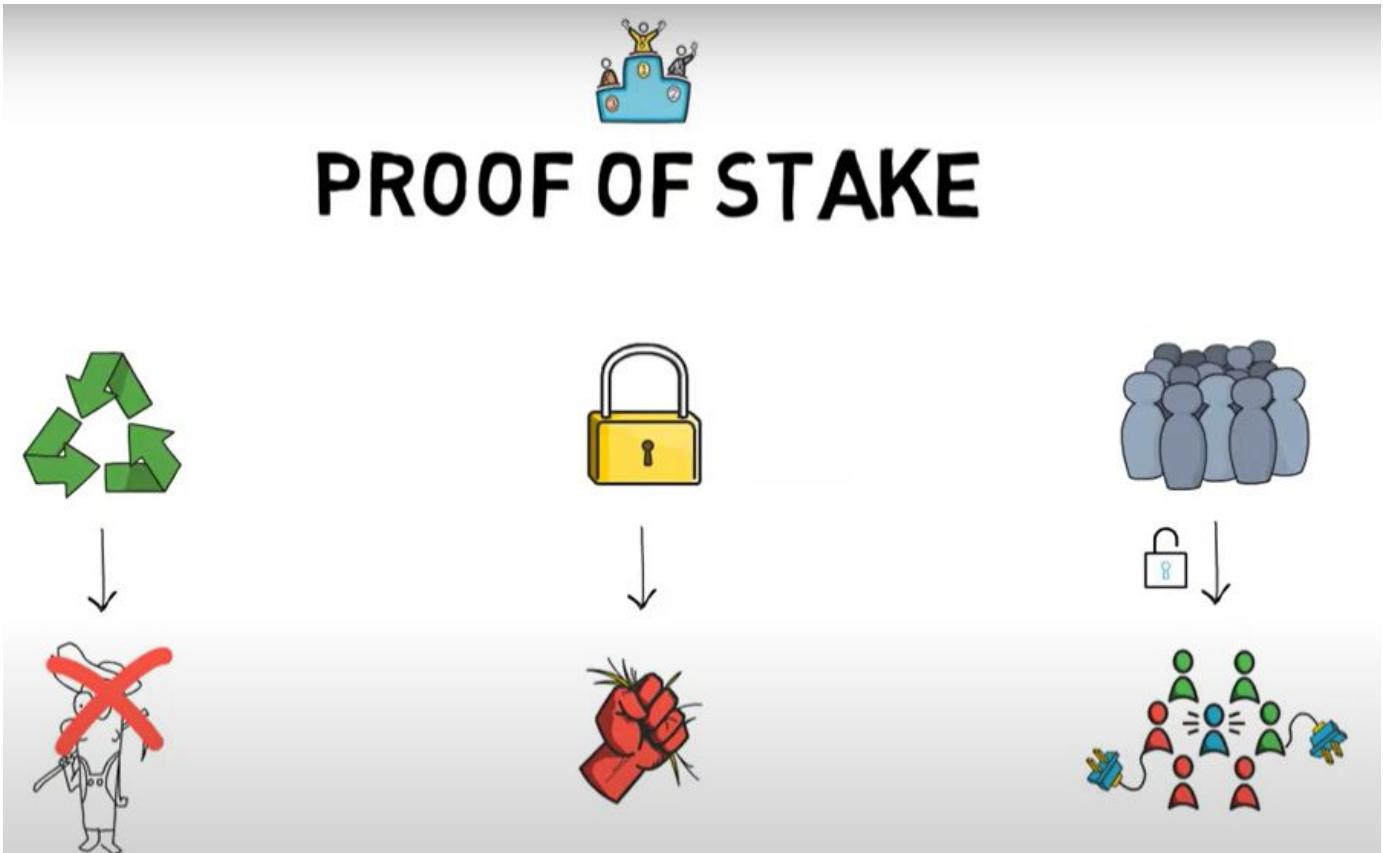
"51% ATTACK"



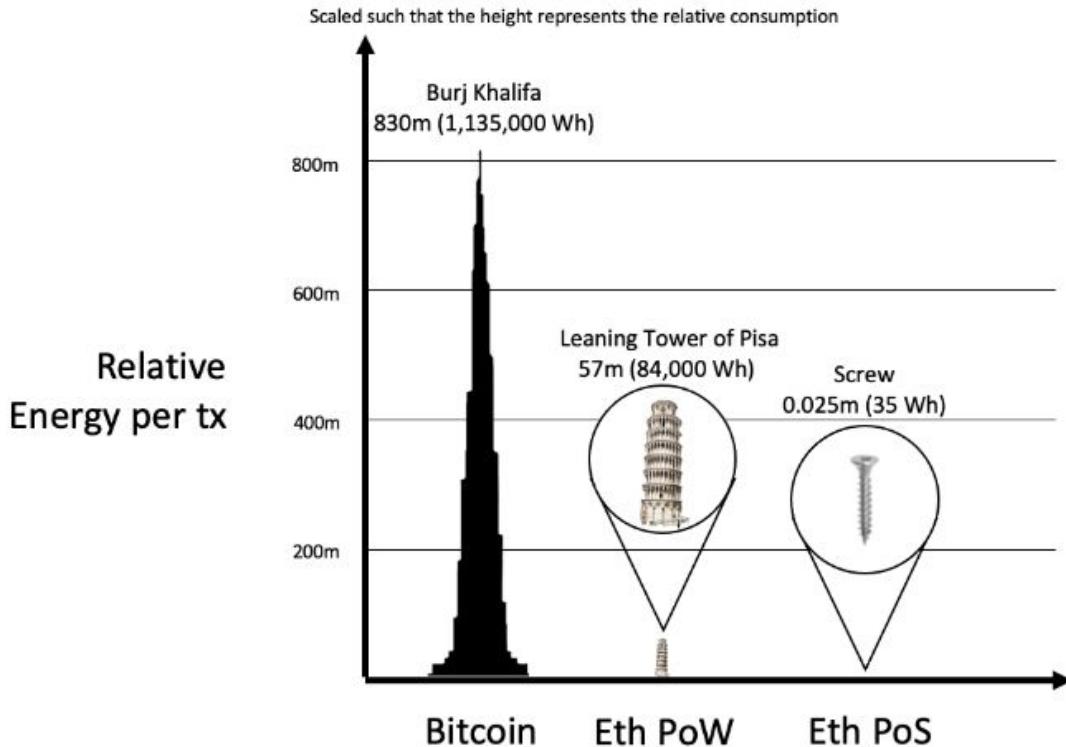
51% VALIDATORS

51% STAKED ETH





Relative energy consumption per transaction

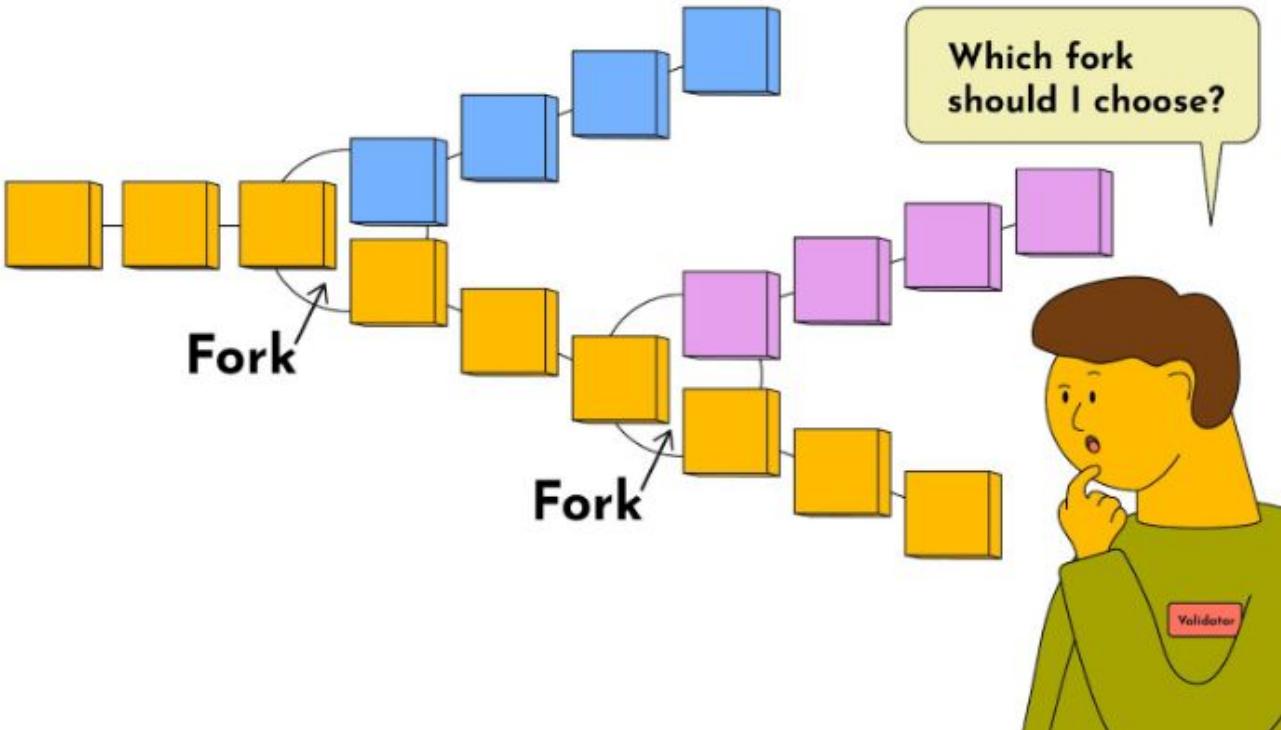


Proof-of-Work hash,

- each "vote" is backed by hardware units.

Proof-of-Stake,

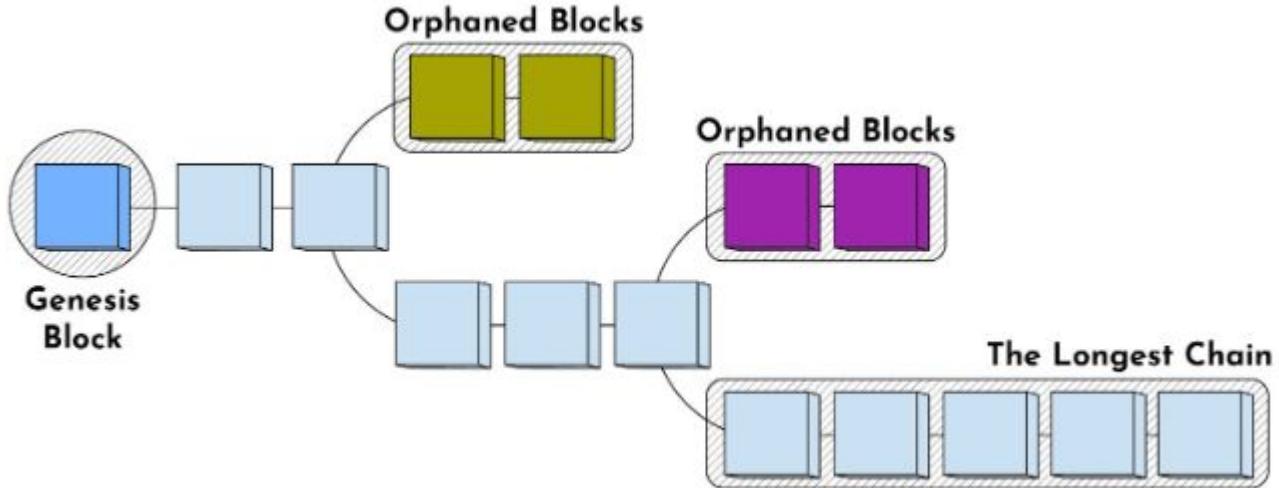
- uses money as a Sybil resistance mechanism.
- each vote is backed by money.



Fork-choice rule is how a consensus algorithm chooses the "correct" chain.

Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

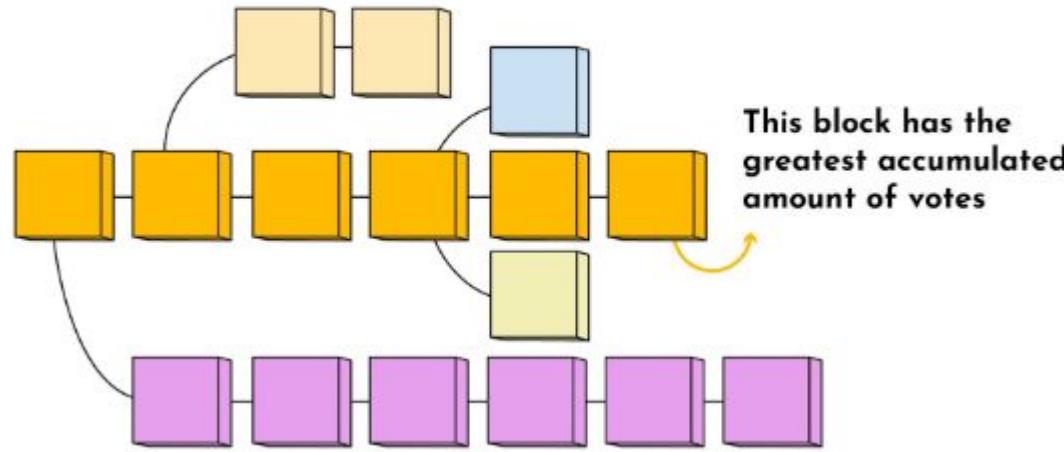
Bitcoin uses the "longest chain" rule, which means that whichever blockchain is the longest will be the one the rest of the nodes accept as the valid chain. The longest chain is determined by the cumulative Proof-of-Work difficulty on that chain.



Proof-of-Stake,

- Ethereum now looks at the “weight” of the chain, which is determined by the accumulated sum of validator votes.

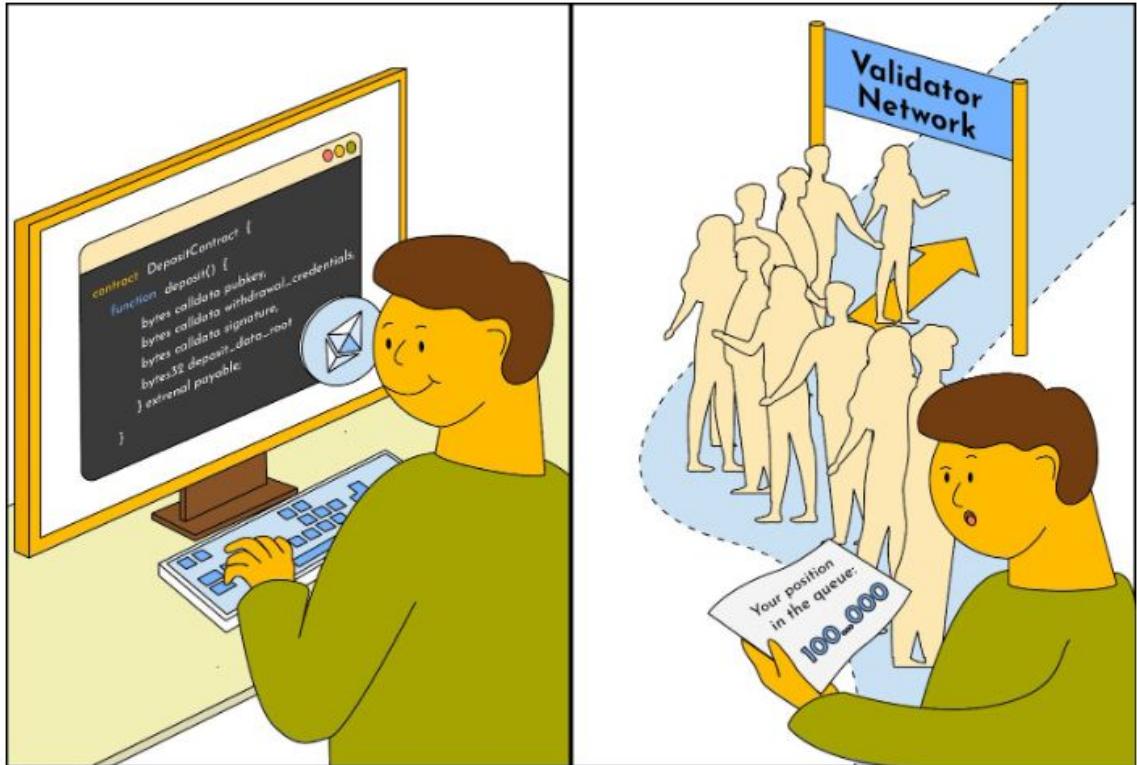
In Gasper, the fork-choice rule is called "LMD-GHOST," which stands for "latest message-driven greedy heaviest observed sub-tree." This means we select the fork with the greatest accumulated weight of attestations to be the canonical one.



Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

"Validators" are the users that are responsible for voting on which block to add next. Anyone can become a validator by depositing 32 ETH into the "deposit contract" (which is a special contract designated to collect and track validators) and then running two separate pieces of software: an execution client and a consensus client.

After depositing their ETH into the deposit contract, the user joins an "activation queue" where they are basically in line to become a validator.

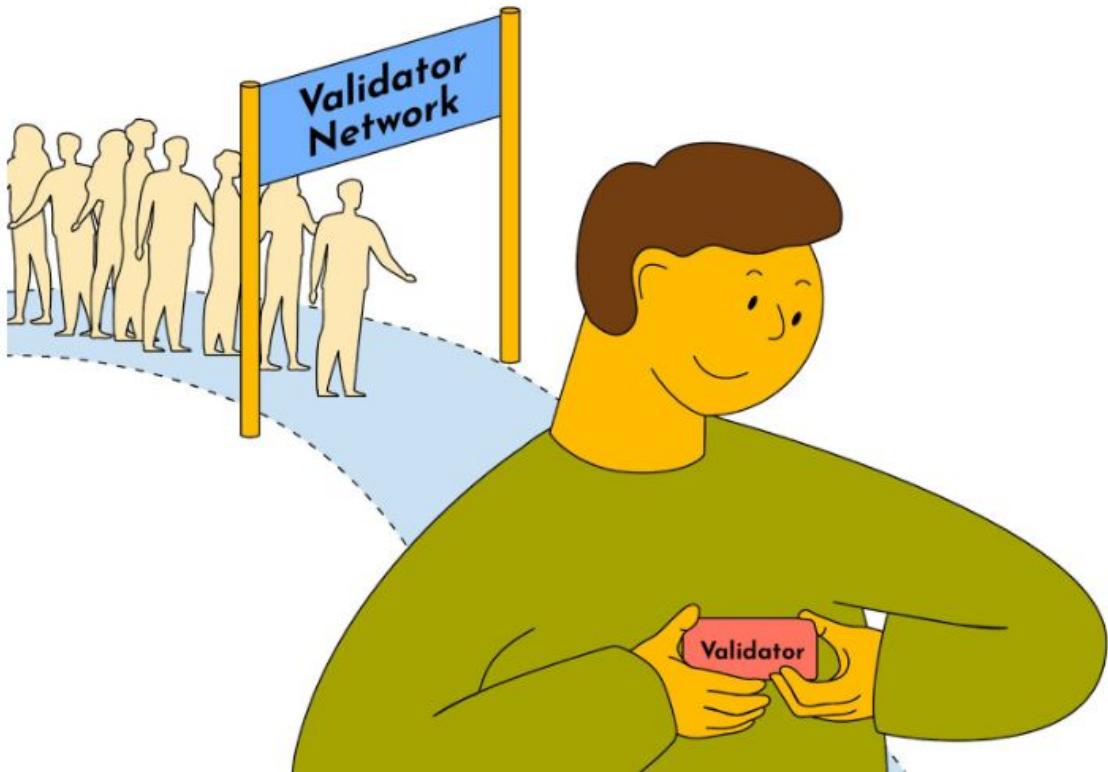


Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

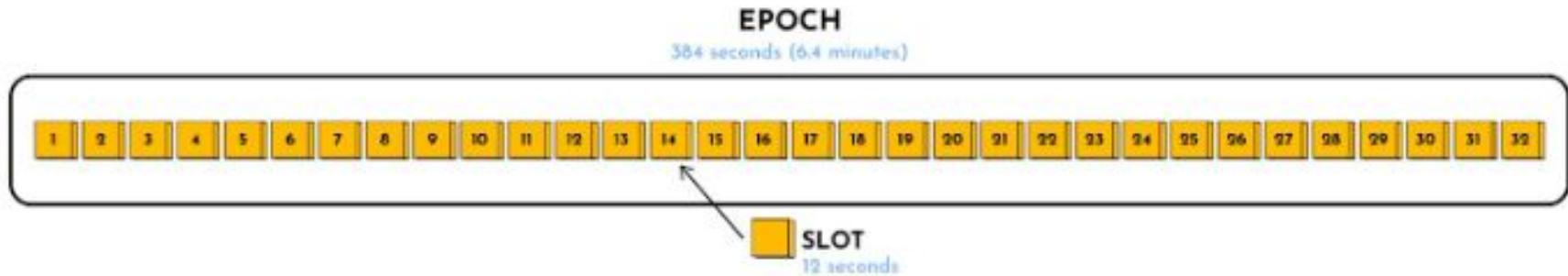
This is done to limit the rate of new validators entering the network at any one time. The goal of rate-limiting entry and exit is two-fold:

1. The communication overhead of validators casting their votes is very high, and there's a limit to how many validators the protocol can handle while ensuring votes are propagated on time.
2. To prevent a large number of malicious validators from performing some malicious action and then leaving the network to escape penalties.

Once a validator is activated, the validator's responsibilities begin.



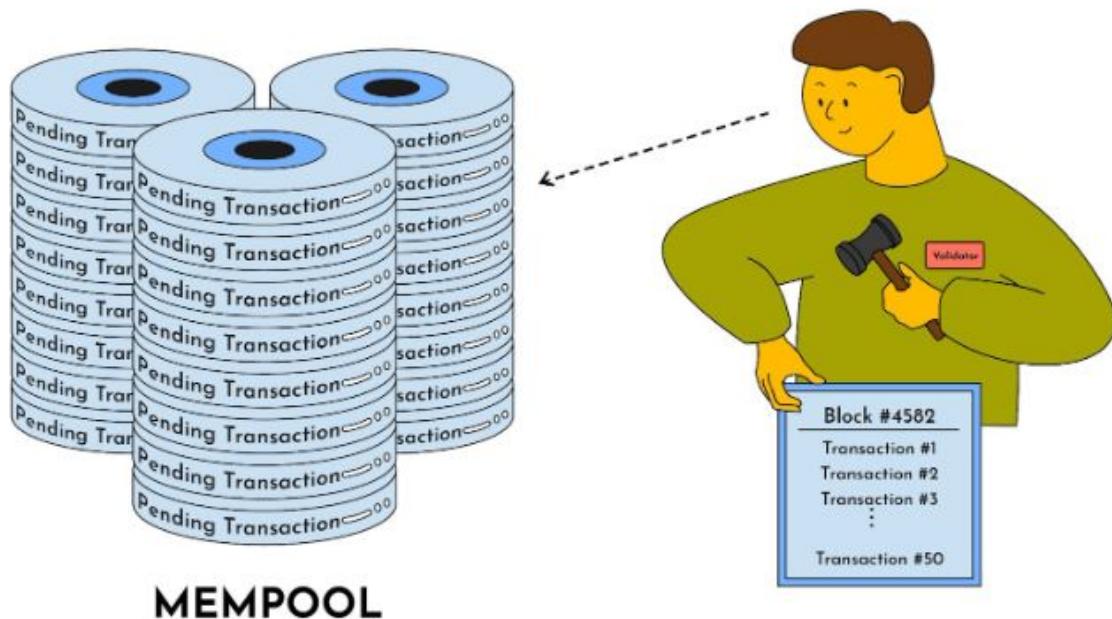
Every consensus algorithm needs a sense of time to order events in the network, such as votes. For Gasper, the algorithm divides time into "slots" and "epochs." Slots are 12 seconds long, and each epoch consists of 32 slots (thus 6.4 minutes).



Note: Although not used as much, a period of 2048 epochs (-9.1 days) is called an eek ("Ethereum week"). Operations that take a long time can be measured in eeks.

Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

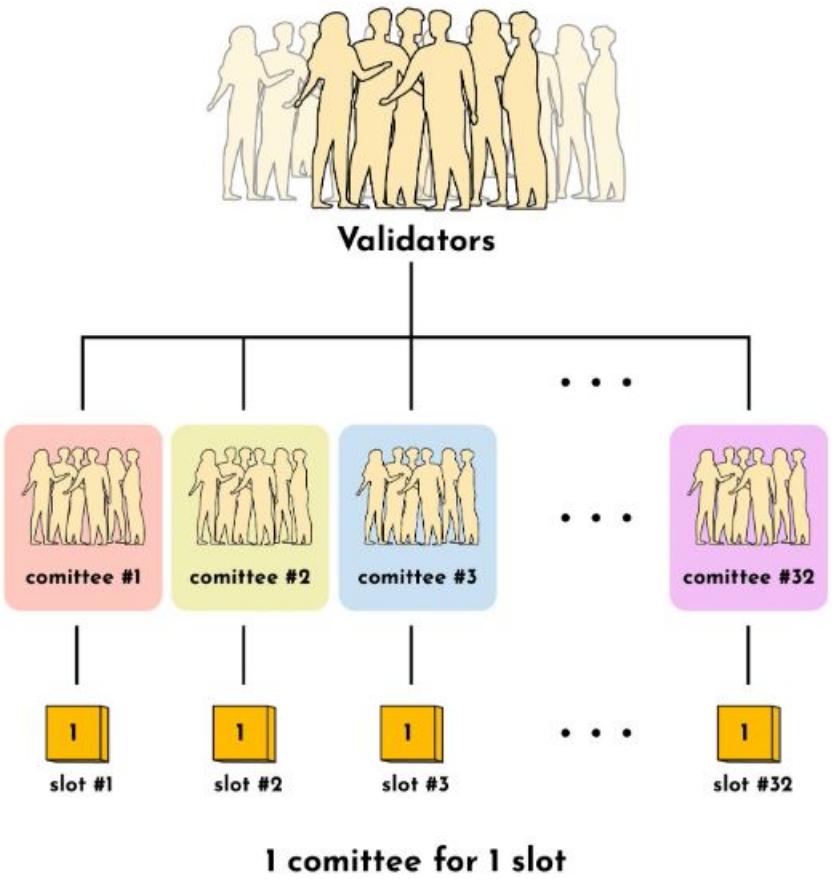
For every slot, one validator is randomly selected to be a "block proposer." The block proposer validator is responsible for constructing a new block out of pending transactions. It then sends the block out to other validators in the network, which vote on whether that block is valid.



Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

Not every validator votes for every block, though. Instead, for each epoch, a validator is assigned to a "committee." Each committee is randomly assigned one slot where they need to attest to determine whether the proposed block is valid.

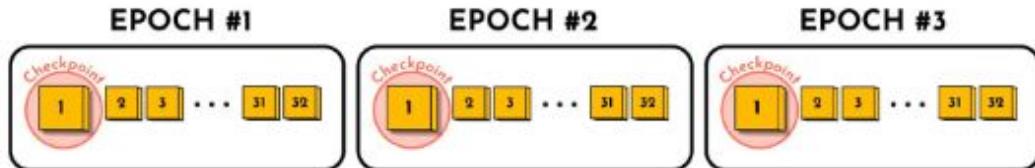
Committees are spread among the 32 slots in an epoch, which means 1/32 of the total validator set attests to the validity of each slot or block, with the maximum allowed validators per committee being 2048. In other words, each validator only attests one block per epoch for the slot their committee was assigned to.



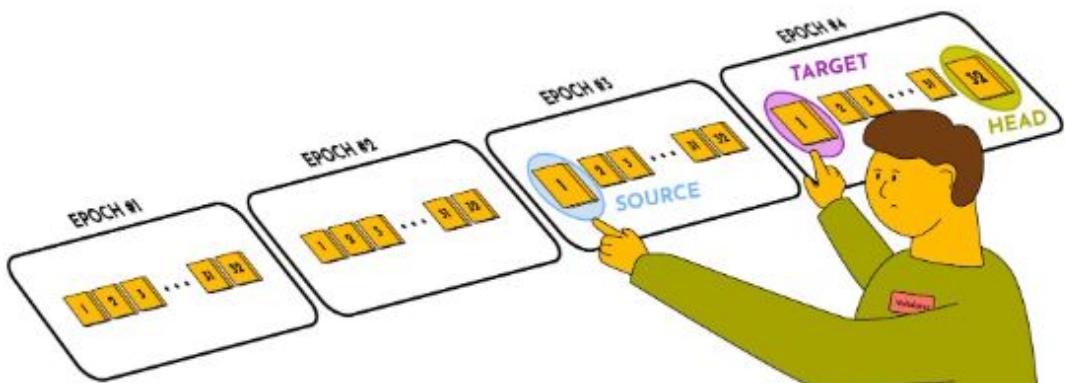
Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

Note that a small set of validators are also randomly chosen to join sync committees, which are different from the committees mentioned above. Being in a sync committee requires validators to help light clients sync up and determine the head of the chain, which they earn additional rewards for. However, as a validator, you are only part of a sync committee once every ~22 months, so it's not a responsibility carried on a daily basis.

Validators don't only attest to the current head block. They also attest to two others called "checkpoint" blocks. Every epoch has one checkpoint block that identifies the latest block at the start of that epoch.



Validators attest to their view of two checkpoints every epoch: the "source" and "target" blocks.





Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS



In addition to the head, source, and target blocks, validators also include the following information in their attestation:

- aggregation_bits: a bitlist of validators where the position maps to the validator index in their committee; the value (0/1) indicates whether the validator signed the data (i.e., whether they are active and agree with the block proposer)
- data: details relating to the attestation (defined below)
- signature: a BLS signature that aggregates the signatures of individual validators

The data contains the following information:

- slot: the slot number that the attestation refers to
- index: the committee # that the validator belongs to in a given slot
- beacon_block_root: root hash of the block the validator sees at the head of the chain
- source: what the validators see as the most recent justified block
- target: what the validators see as the first block in the current epoch

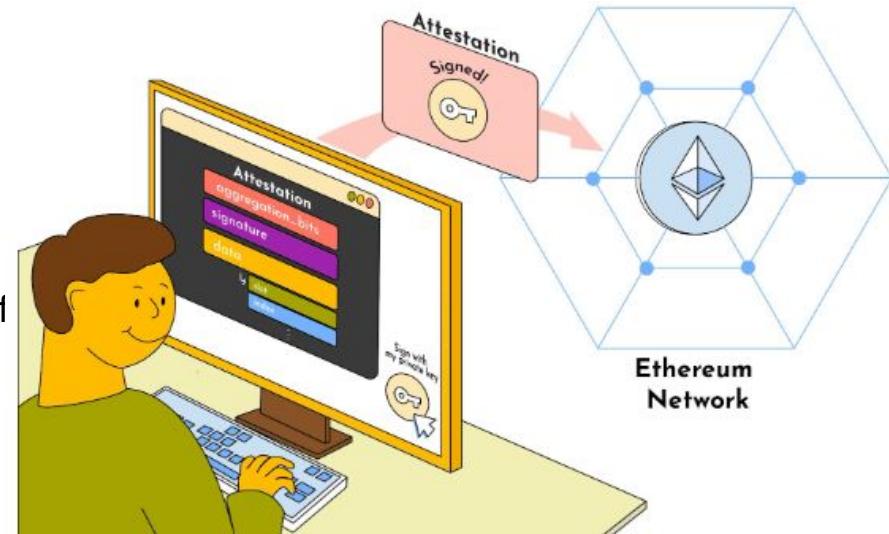


Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

Once the data is built, the validator can flip the bit in aggregation_bits corresponding to their own validator index from 0 to 1 to show that they participated in the vote. Finally, the validator signs the attestation using a private key and broadcasts it to the network.

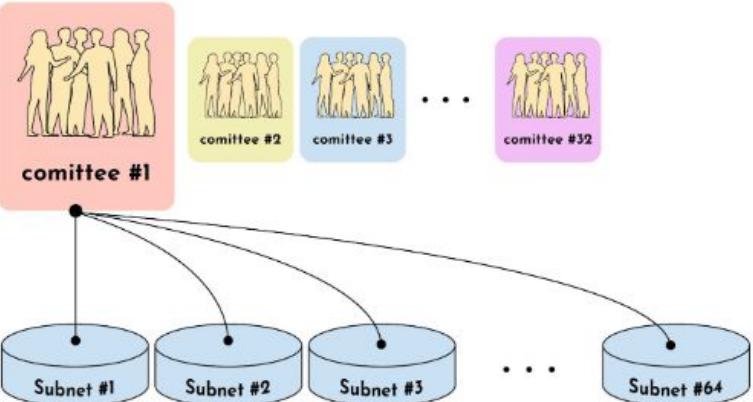
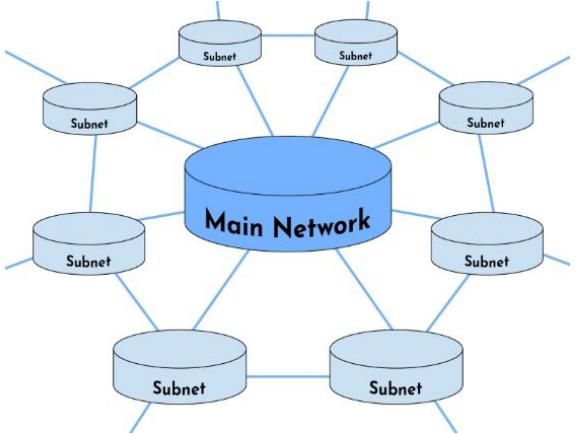
But wouldn't passing attestation data created by every validator to every other validator in the network create a lot of overhead? Yes. That's why, instead of having every validator listen to every other validator, attestations from individual validators are aggregated within "subnets" before they're broadcasted.

If you divide a network into smaller, individual networks, those networks become subnets. It's just a network in a network. Subnets make communication much more efficient by collecting and shortening the route that data has to travel to get to its destination.



Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

Remember when we learned that validators are divided into committees? Each of those committees is further divided into **64 subnets**. This lets aggregation happen.



Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

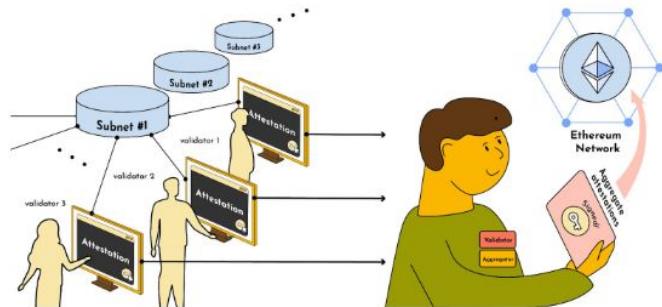
In each epoch, a validator in each subnet is selected to be the aggregator. The aggregator collects all the attestations that have equivalent data to its own. The sender of each matching attestation is recorded in the aggregation_bits. That means a single signature is formed by combining the signatures of all validators that agree with the aggregator's data. The aggregator then broadcasts the aggregate attestations to the broader network.

When a validator is selected to be a block proposer, they package aggregate attestations from the subnets up to the latest slot in the new block.

So, in summary, the responsibilities of a validator include the following:

- Attesting on a block (once per epoch)
- Aggregating attestations from validators in the same committee (occasionally when selected to be an aggregator)
- Creating blocks when selected (infrequently when selected to be a validator)

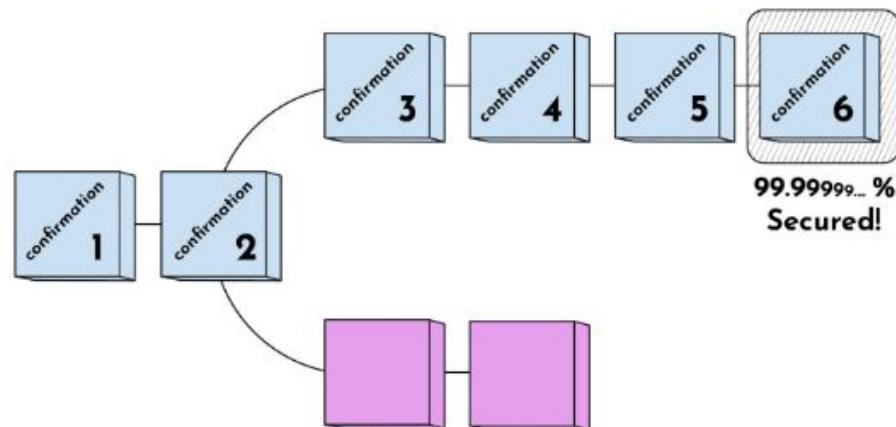
As a result of these attestations, the blockchain comes to a consensus.

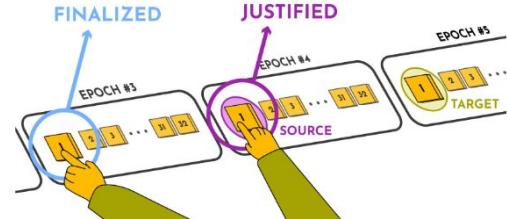


Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

A quick recap: validators vote on blocks that a block proposer creates — if they believe it's a valid block. But at what point do we determine that a block should be included in the chain? This is what we mean when we talk about "finality." If a block reaches "finality," it cannot be reverted unless there has been a critical consensus failure.

In Bitcoin, we know a block is "final" by waiting for six confirmations. In other words, if six blocks get built on top of the current block, we assume the current block is finalized. The probability of a block being reorged after six confirmations is so small that it's statistically negligible.





In Gasper, we determine a finalized block (or "finality") differently. Rather than using six confirmations, we use the following logic:

- If $\frac{2}{3}$ of the total staked ether votes in favor of that block, then it becomes "justified." Justified blocks are unlikely to be reverted, but they can be under certain conditions.
- When another block is justified on top of a justified block, it's upgraded to "finalized." Finalizing a block is a commitment to include the block in the canonical chain. It can't be reverted unless an attacker destroys millions of ether.

Blocks do not reach the "justified" and "finalized" states in every slot. Instead, they happen to the first block in every epoch, which is called a "checkpoint" block.

When a checkpoint block gets upgraded to justified, it must have a link to the previous checkpoint. That is, two-thirds of the total staked ether must vote that checkpoint B is the correct descendant of checkpoint A. As a result, the previous checkpoint block becomes finalized, and the more recent block is justified.



Therefore, an attacker cannot create an alternative finalized chain without:

- Owning or manipulating $\frac{2}{3}$ of the total staked ether (because $\frac{2}{3}$ of total ether is required to finalize a chain)
- Destroying at least $\frac{1}{3}$ of the total staked ether (because if $\frac{2}{3}$ of total ether was used to finalize two different chains, that means $\frac{1}{3}$ of the ether voted maliciously, which would be slashed)

Now that you understand how validator votes help determine which blocks reach finality, we can move on to the fork-choice rule.

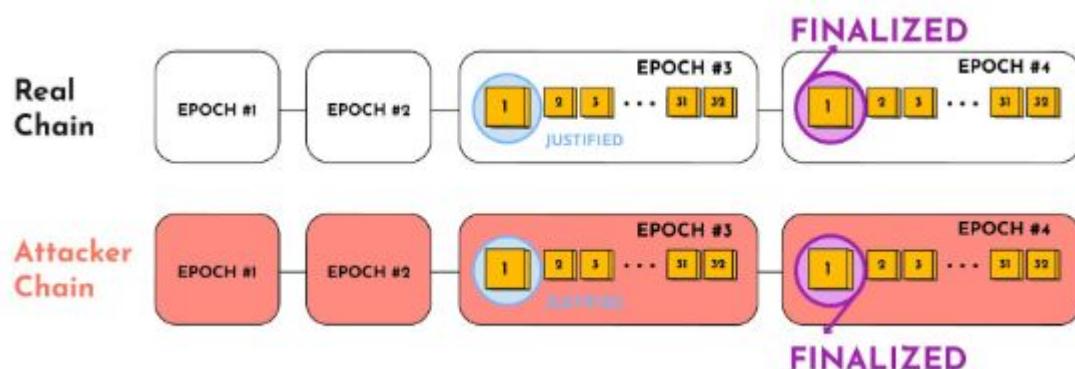


Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

So far, we understand how blocks are voted on, how they become "finalized," and how validators know which fork to build on top of. This gets us a blockchain where a decentralized set of actors can create and agree on what block to add to the blockchain next.

But the big issue with Proof-of-Stake consensus mechanisms is that they're susceptible to "**long-range attacks**" — that is, to a group of attackers can build a fork from genesis, hide it from users for a while, and then suddenly publish it. This could lead to **two different chains with "finalized" blocks** and end users wouldn't know which is the correct chain.

This wouldn't be possible in the Proof-of-Work protocol because it would require an insurmountable amount of hash power to generate all of the previous blocks and outpace the main chain. But in Proof-of-Stake, there's no such thing as hash power since votes are just digital representations of data, so it's entirely possible for a subset of validators to create a new blockchain that looks and acts like the main chain and deceive others into believing in this alternative blockchain.



4. Weak Subjectivity

There are certain checkpoints where every node in every network agrees that a block (and each that came before it) belongs in the canonical chain. This is called a "**weak subjectivity checkpoint**." If a node sees a block that conflicts with a weak subjectivity checkpoint, then it rejects that block. The latest weak subjectivity checkpoint is like a new genesis block for the entire network.

Recall that in any blockchain, the genesis block is typically the first block that everyone builds future blocks on top of. Similarly, weak subjectivity checkpoints are like genesis blocks that everyone agrees belongs to the canonical chain — as such, everyone builds on top of that weak subjectivity point.

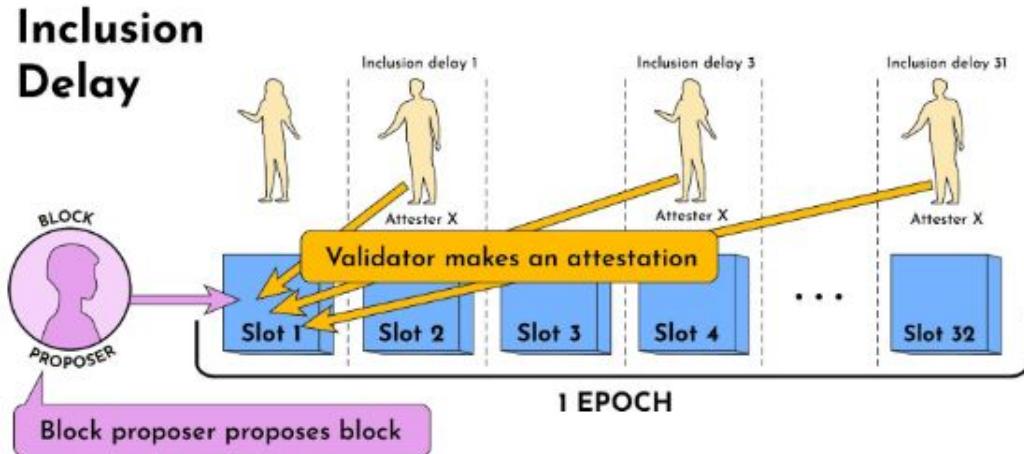
New nodes that are trying to sync to the canonical head of the Beacon chain must use the latest weak subjectivity checkpoint as the starting point of their sync.

This is where the major criticism of weak subjectivity comes in: that nodes just have to trust that they're hearing about the latest weak subjectivity checkpoint from trusted sources (e.g., block explorers or other nodes). For those interested, Vitalik goes into his rationale for why he's okay with this in this [post](#).

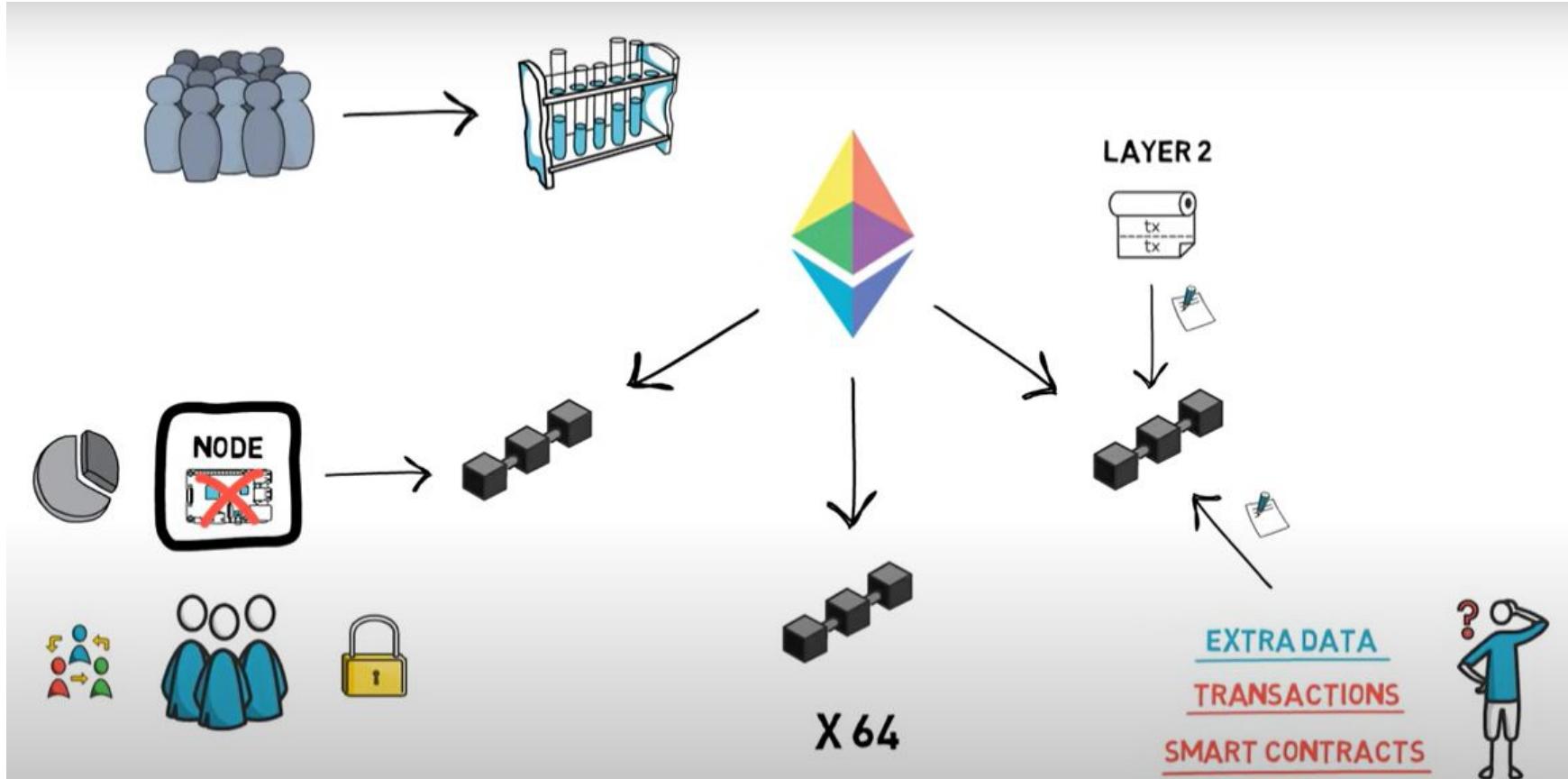


Ethereum 2.0 / ETH 2.0 / Serenity - Key features : PoS

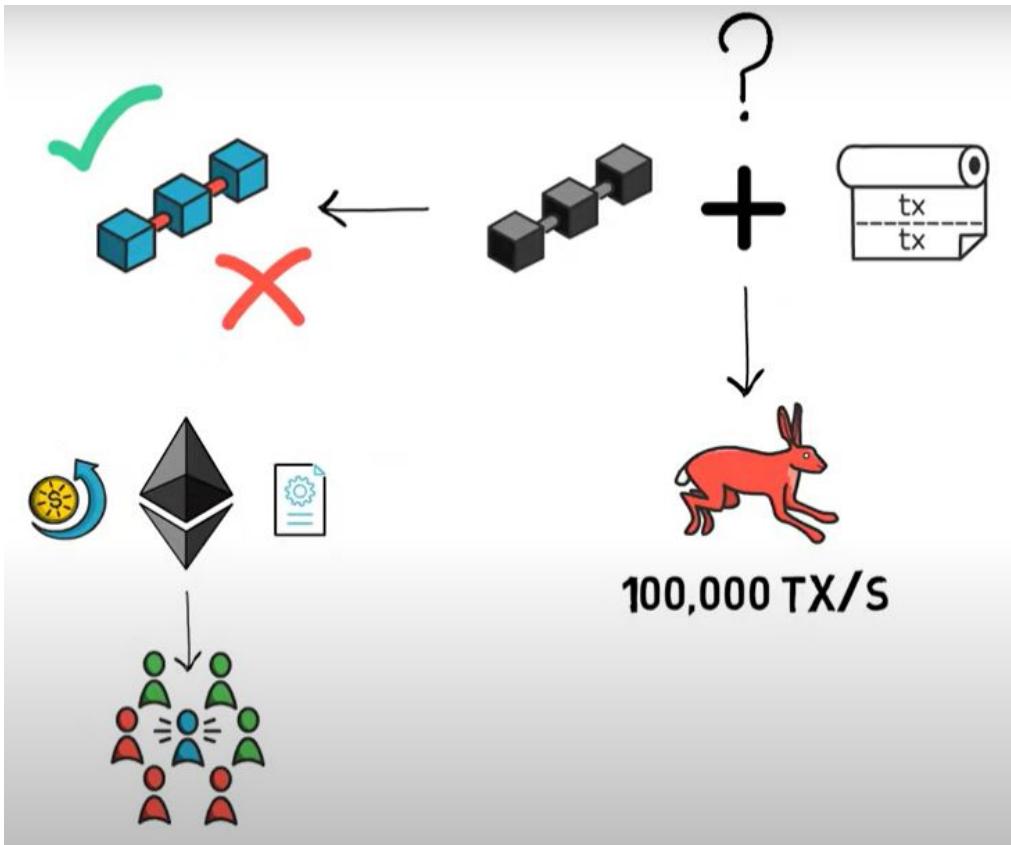
Recall that an attestation contains three votes (source, target, head). Each vote is eligible for a reward. Validators only receive rewards for "correct" attestations — meaning their votes agree with the current block proposer. Moreover, there is an additional weight based on how quickly a validator makes attestations after a block has been proposed, where the base reward gets divided by the "inclusion delay."



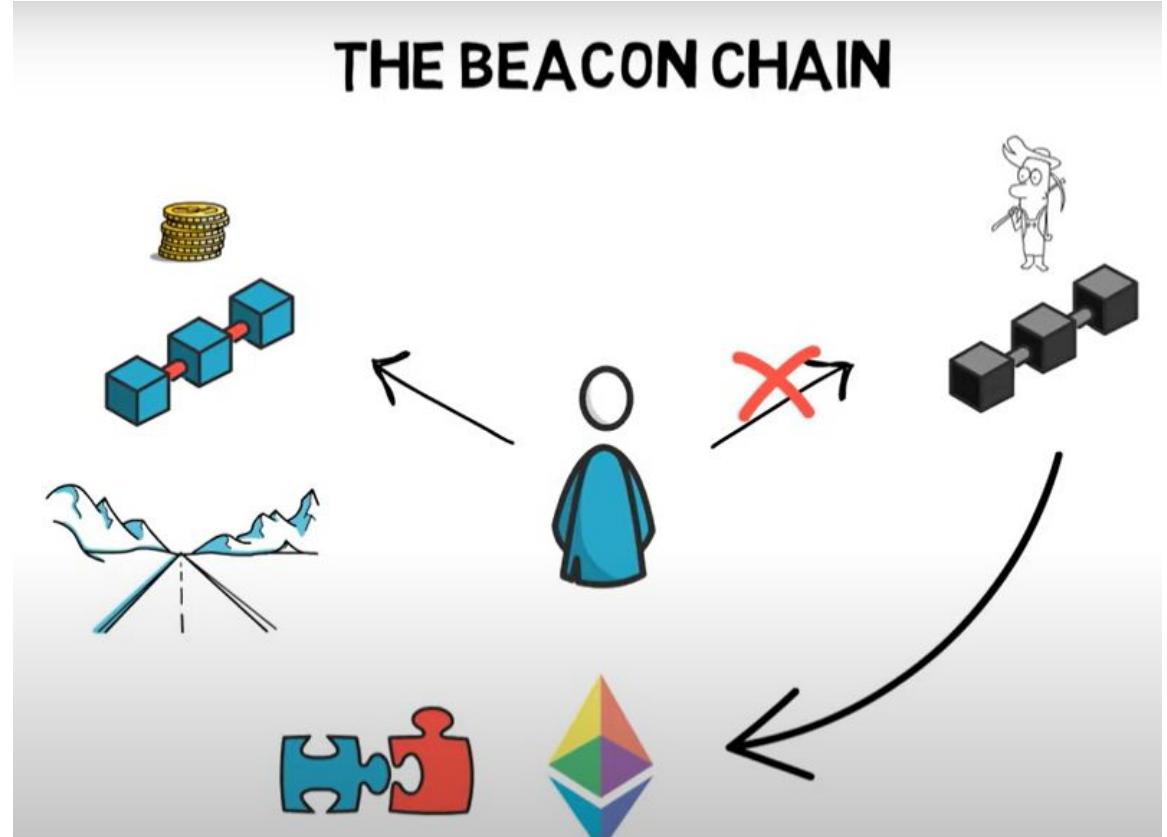
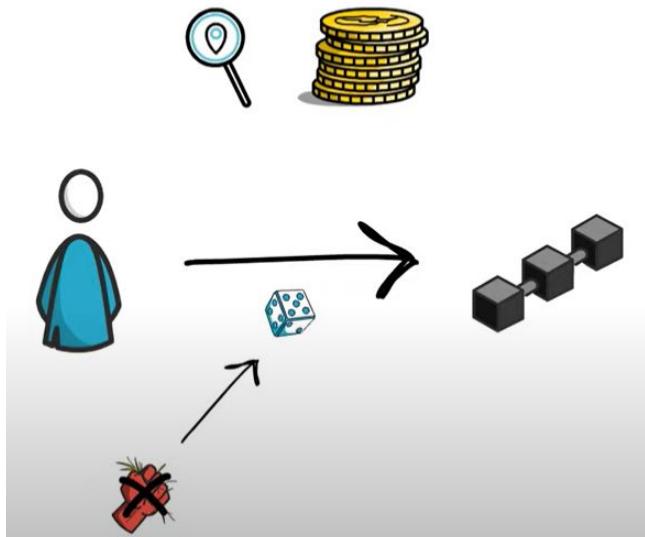
Ethereum 2.0 / ETH 2.0 / Serenity - Key features : Sharding



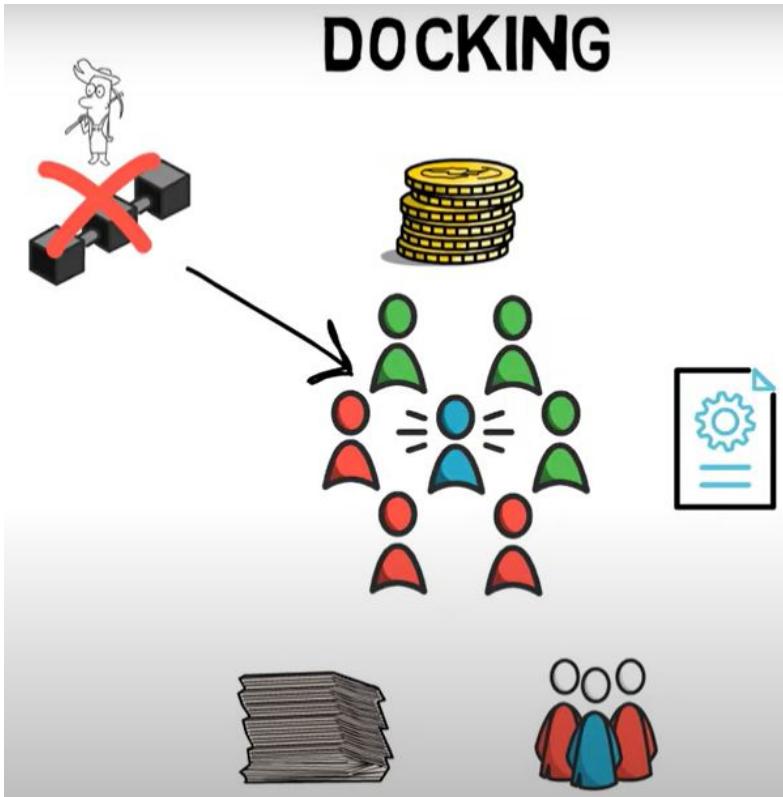
Ethereum 2.0 / ETH 2.0 / Serenity - Key features : Sharding



THE BEACON CHAIN



Ethereum 2.0 / ETH 2.0 / Serenity - Key features : Docking



Timeline for Ethereum 2.0 / ETH 2.0 / Serenity

