

Blockchain Technology (BCS613A)

Module 4: Ethereum

Course Notes

Syllabus:

Ethereum 101, Introduction, Ethereum clients and releases, The Ethereum stack, Ethereum blockchain, Currency (ETH and ETC), Forks, Gas, The consensus mechanism, The world state, Transactions, Contract creation transaction, Message call transaction, Elements of the Ethereum blockchain , Ethereum virtual machine (EVM), Accounts, Block, Ether, Messages, Mining, The Ethereum network. Hands-on: Clients and wallets –Geth..

Ethereum 101

Ethereum – an overview

Ethereum Clients and Releases

- Vitalik Buterin (<https://vitalik.ca>) conceptualized Ethereum (<https://ethereum.org>) in November, 2013.
- The core idea proposed was the development of a Turing-complete language that allows the development of arbitrary programs (smart contracts) for blockchain and Decentralized Applications (DApps).
- This concept is in contrast to Bitcoin, where the scripting language is limited and only allows necessary operations.
- The first version of Ethereum, called Olympic, was released in May, 2015.
- Two months later, a version of Ethereum called Frontier was released in July.
- Another version named Homestead with various improvements was released in March, 2016.
- The latest Ethereum release is called Muir Glacier, which delays the difficulty bomb (<https://eips.ethereum.org/EIPS/eip-2384>).
- A major release before that was Istanbul, which included changes around privacy and scaling capabilities.
- The difficulty bomb is a difficulty adjustment mechanism that eventually forces all miners to stop mining on Ethereum 1 and move to Ethereum 2.
- Once activated, over time, it makes mining on Ethereum 1.x so prohibitively slow that mining becomes infeasible and results in a so called "ice age".
- This mechanism exponentially increases the Proof-of-Work (PoW) mining difficulty to a level where block generation becomes impossible, thus forcing the miners to migrate to Ethereum 2.0's Proof-of-Stake (PoS) system.

Major upgrades of Ethereum:

Release	Description	Release date	Details and original announcement
Olympic	Pre-release	May 9, 2015	https://blog.ethereum.org/2015/05/09/olympic-frontier-pre-release/
Frontier	First live network	July 30, 2015	https://blog.ethereum.org/2015/03/12/getting-to-the-frontier/
Frontier Thawing	Initial protocol adjustments	September 08, 2015	https://blog.ethereum.org/2015/08/04/the-thawing-frontier/
Homestead	Second major version	March 15, 2016	https://blog.ethereum.org/2016/02/29/homestead-release/
Spurious Dragon	EIP55	November 23, 2016	https://blog.ethereum.org/2016/11/18/hard-fork-no-4-spurious-dragon/
Byzantium	Metropolis part 1	October 16, 2017	https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/
Constantinople St. Petersburg	Metropolis part 2	February 28, 2019	https://blog.ethereum.org/2019/02/22/ethereum-constantinople-st-petersburg-upgrade-announcement/
Istanbul	Metropolis part 3	December 08, 2019	https://blog.ethereum.org/2019/11/20/ethereum-istanbul-upgrade-announcement/
Muir Glacier	Unplanned release	January 01, 2020	https://blog.ethereum.org/2019/12/23/ethereum-muir-glacier-upgrade-announcement/
Berlin	Second phase of Istanbul	Q3 2020	https://eips.ethereum.org/EIPS/eip-2070
Serenity	ETH 2.0	To be implemented in different phases	

The Ethereum blockchain

- Ethereum, just like any other blockchain, can be visualized as a transaction-based state machine.
- The core idea is that in the Ethereum blockchain, a genesis state is transformed into a final state by executing transactions incrementally.
- The final transformation is then accepted as the absolute undisputed version of the state.

- In the following diagram, the Ethereum state transition function is shown, where a transaction execution has resulted in a state transition:

Ethereum state transition function

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

Ethereum – a user's perspective

The most common use case of transferring funds:

1. First, either a user requests money by sending the request to the sender, or the sender decides to send money to the receiver. We can use the Jaxx Ethereum wallet software on iOS –the Jaxx wallet can be downloaded from <https://jaxx.io>. The request can be sent by sending the receiver's Ethereum address to the sender by using a QR code.

Tap to copy this address. Share it with the sender via email or text.



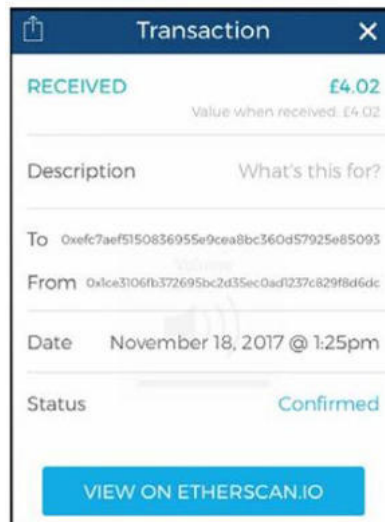
0xFc7aEF5150836955e9CEa8Bc360D57925e850...

2. Once sender receives this request he can scan this QR code or copy the Ethereum address in the Ethereum wallet software and initiate a transaction. The amount and the destination address to be entered and confirm the transaction.



3. Once the request for money to be sent is constructed in the wallet software, it is broadcasted to the Ethereum network. The transaction is digitally signed by the sender as proof.
4. This transaction is then picked up by nodes called miners for verification and inclusion in the block. At this stage, the transaction is still unconfirmed.
5. Once it is verified and included in the block, the PoW process starts.

6. Once a miner finds the answer to the PoW problem by repeatedly hashing the block with a new nonce, this block is immediately broadcasted to the rest of the nodes, which then verifies the block and PoW.
7. If all the checks pass then this block is added to the blockchain, and miners are paid rewards accordingly.
8. Finally, receiver gets the ether, and it is shown in her wallet software.



The details of this transaction can be visualized as:

Overview	State Changes	Comments
Transaction Hash:	0xc63dce6747e1640abd63ee63027c3352aed8cdb92b6a02ae25225666e171009e	
Status:	Success	
Block:	4576084	4659657 Block Confirmations
Timestamp:	780 days 7 hrs ago (Nov-18-2017 01:25:54 PM +UTC)	
From:	0x1ce3106fb372695bc2d35ec0ad1237c829f8d6dc	
To:	0xefc7aef5150836955e9cea8bc360d57925e85093	
Value:	0.015927244142974896 Ether	(\$2.29)
Transaction Fee:	0.000441 Ether	(\$0.06)
Gas Limit:	21,000	
Gas Used by Transaction:	21,000 (100%)	
Gas Price:	0.000000021 Ether (21 Gwei)	
Nonce	Position	1 4

The Ethereum network

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

The mainnet

- The mainnet is the current live network of Ethereum. Its network ID is 1 and its chain ID is also 1.
- Network IDs and chain IDs are used by Ethereum clients to identify the network.
- Chain ID was introduced in EIP155 as part of replay protection mechanism.
- A block explorer that shows detailed information about blocks and other relevant metrics is available at <https://etherscan.io>.
- This can be used to explore the Ethereum blockchain.

Testnets

- There is a number of testnets available for Ethereum testing. The aim of these test blockchains is to provide a testing environment for smart contracts and DApps before being deployed to the production live blockchain.
- They also allow experimentation and research.
- The main testnet is called Ropsten, which contains all the features of other smaller and special-purpose testnets that were created for specific releases.
- Other testnets include Kovan and Rinkeby, which were developed for testing Byzantium releases.
- The changes that were implemented on these smaller testnets have also been implemented in Ropsten. Now the Ropsten test network contains all properties of Kovan and Rinkeby.

Private nets

- These are private networks that can be created by generating a new genesis block.

- This is usually the case in private blockchain networks, where a private group of entities start their blockchain network and use it as a permissioned or consortium blockchain.

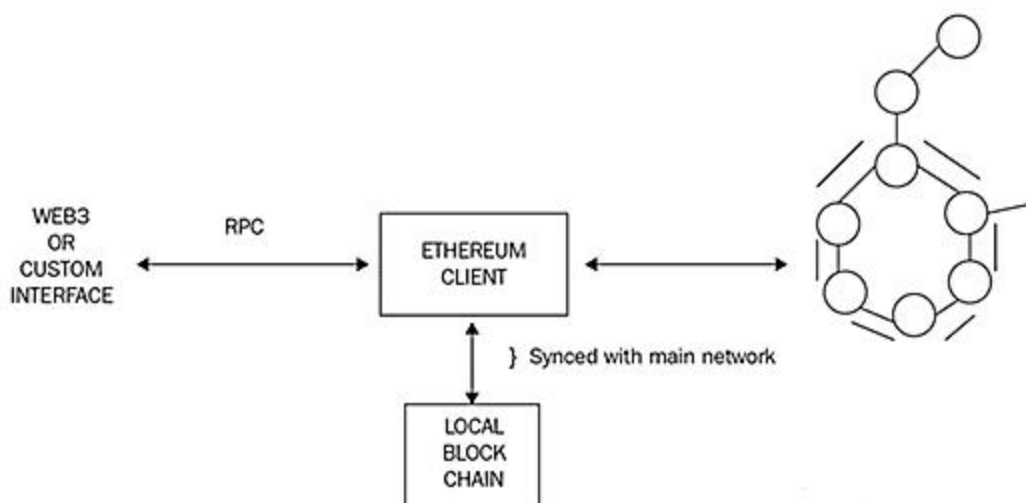
The Ethereum Stack

The Ethereum stack consists of various components. At the core, there is the Ethereum blockchain running on the P2P Ethereum network.

Secondly, there's an Ethereum client (usually geth) that runs on the nodes and connects to the peer-to-peer Ethereum network from where blockchain is downloaded and stored locally.

It provides various functions, such as mining and account management. The local copy of the blockchain is synchronized regularly with the network.

Another component is the web3.js library that allows interaction with geth via the Remote Procedure Call (RPC) interface. This can be visualized in the following diagram



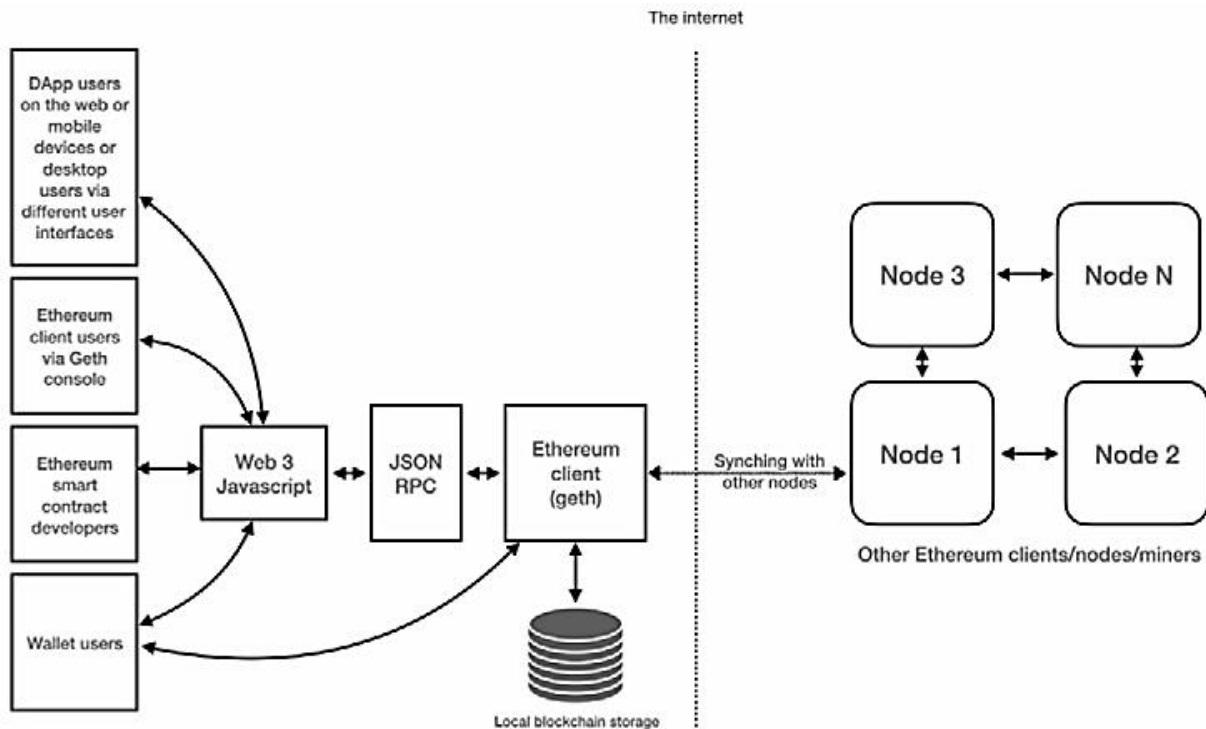
The Ethereum stack is essentially a suite of tools and technologies designed to develop and interact with Ethereum-based applications. It's the foundation for building decentralized applications (dApps) on the Ethereum blockchain. Here's a breakdown:

1. **Ethereum Virtual Machine (EVM):** This is the computation engine that executes smart contracts on the blockchain. Every Ethereum node runs the EVM.

2. **Smart Contracts:** These are self-executing contracts with predefined rules written in code. They are usually written in programming languages like Solidity and then deployed to the blockchain.
3. **Solidity:** The most popular programming language for writing Ethereum smart contracts. It's high-level and specifically tailored for the Ethereum Virtual Machine.
4. **Web3.js:** A JavaScript library that enables developers to interact with the Ethereum blockchain. Through this, developers can send transactions, fetch blockchain data, and interact with smart contracts.
5. **Ganache:** A personal blockchain for Ethereum development. It helps developers simulate the blockchain environment locally, making it easier to test and deploy contracts.
6. **Truffle:** A development framework for Ethereum that simplifies tasks like compiling smart contracts, deploying them, and testing them. It's widely used in dApp development.
7. **Infura:** A cloud-based service that provides scalable access to Ethereum nodes. Developers use it to connect their dApps to the blockchain without needing to run their own node.
8. **MetaMask:** A browser extension that acts as a wallet and interface for Ethereum applications. It allows users to interact with dApps by managing keys and signing transactions.

Components of the Ethereum ecosystem

- The Ethereum blockchain stack consists of various components.
- At the core, there is the Ethereum blockchain running on the peer-to-peer Ethereum network.
- Secondly, there's an Ethereum client (usually Geth) that runs on the nodes and connects to the peer-to-peer Ethereum network from where blockchain is downloaded and stored locally.
- It provides various functions, such as mining and account management.
- The local copy of the blockchain is synchronized regularly with the network.
- Another component is the web3.js library that allows interaction with the geth client via the Remote Procedure Call (RPC) interface.



Ethereum high-level ecosystem

A list of elements present in the Ethereum blockchain:

- Keys and addresses
- Accounts
- Transactions and messages
- Ether cryptocurrency/tokens
- The EVM
- Smart contracts and native contracts

Keys and addresses

- Keys and addresses are used in the Ethereum blockchain to represent ownership and transfer ether.
- The keys used are made up of pairs of private and public parts.
- The private key is generated randomly and is kept secret, whereas a public key is derived from the private key.

- Addresses are derived from public keys and are 20-byte codes used to identify accounts.

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

- First, a private key is randomly chosen (a 256-bit positive integer) under the rules defined by the elliptic curve secp256k1 specification (in the range $[1, \text{secp256k1n} - 1]$).
- The public key is then derived from this private key using the Elliptic Curve Digital Signature Algorithm (ECDSA) recovery function.
- An address is derived from the public key, specifically, from the rightmost 160 bits of the Keccak hash of the public key.

Accounts

- Accounts are one of the main building blocks of the Ethereum blockchain. They are defined by pairs of private and public keys.
- Accounts are used by users to interact with the blockchain via transactions.
- A transaction is digitally signed by an account before submitting it to the network via a node.
- Ethereum, being a transaction-driven state machine, the state is created or updated as a result of the interaction between accounts and transaction executions.
- All accounts have a state that, when combined together, represents the state of the Ethereum network.
- With every new block, the state of the Ethereum network is updated. Operations performed between and on the accounts represent state transitions.

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

1. Confirm the transaction validity by checking the syntax, signature validity, and nonce.
2. The transaction fee is calculated, and the sending address is resolved using the signature. The sender's account balance is checked and

subtracted accordingly, and the nonce is incremented. An error is returned if the account balance is insufficient.

3. Provide enough ETH (the gas price) to cover the cost of the transaction. This is charged per byte and is incrementally proportional to the size of the transaction. In this step, the actual transfer of value occurs. The flow is from the sender's account to the receiver's account. The account is created automatically if the destination account specified in the transaction does not exist yet. If the destination account is a contract, then the contract code is executed. This also depends on the amount of gas available. If enough gas is available, then the contract code will be executed fully; otherwise, it will run up to the point where it runs out of gas.
4. In cases of transaction failure due to insufficient account balance or gas, all state changes are rolled back except for the fee payment, which is paid to the miners.
5. Finally, the remainder (if any) of the fee is sent back to the sender as change and the fee is paid to the miners accordingly. At this point, the function returns the resulting state, which is also stored on the blockchain.

Types of accounts

Two kinds of accounts exist in Ethereum:

1. Externally Owned Accounts (EOAs)
2. Contract Accounts (CAs)

EOAs are similar to accounts that are controlled by a private key in Bitcoin. CAs are the accounts that have code associated with them along with the private key.

Properties of account:

EOAs

- They have a state.
- They are associated with a human user, hence are also called user accounts.
- EOAs have an ether balance.

- They are capable of sending transactions.
- They have no associated code.
- They are controlled by private keys.
- EOAs cannot initiate a call message.
- Accounts contain a key-value store.
- EOAs can initiate transaction messages.

CAs

- They have a state.
- They are not intrinsically associated with any user or actor on the blockchain.
- CAs have an ether balance.
- They have associated code that is kept in memory/storage on the blockchain. They have access to storage.
- They can get triggered and execute code in response to a transaction or a message from other contracts. The code within CAs can be of any level of complexity. The code is executed by the EVM by each mining node on the Ethereum network.
- Also, CAs can maintain their permanent states and can call other contracts. It is envisaged that in the Serenity (Ethereum 2.0) release, the distinction between EOAs and CAs may be eliminated.
- CAs cannot start transaction messages.
- CAs can initiate a call message.
- CAs contain a key-value store.
- CAs' addresses are generated when they are deployed. This address of the contract is used to identify its location on the blockchain.

Transactions and messages

A transaction in Ethereum is a digitally signed data packet using a private key that contains the instructions that, when completed, either result in a message call or contract creation.

Transactions can be divided into two types based on the output they produce:

1. **Message call transactions:** This transaction simply produces a message call that is used to pass messages from one CA to another.
2. **Contract creation transactions:** These transactions result in the creation of a new CA. This means that when this transaction is executed successfully, it creates an account with the associated code.

The standard fields of transactions:

- **Nonce:** The nonce is a number that is incremented by one every time a transaction is sent by the sender. It must be equal to the number of transactions sent and is used as a unique identifier for the transaction. A nonce value can only be used once. This is used for replay protection on the network.
- **Gas price:** The gas price field represents the amount of Wei you are willing to pay for this transaction. This is charged per unit of gas for all computation costs incurred as a result of the execution of this transaction.
- **Gas limit:** The gas limit field contains the value that represents the maximum amount of gas that can be consumed to execute the transaction. This is the fee amount, in ether, that a user is willing to pay for computation.
- **To:** The To field is a value that represents the address of the recipient of the transaction. This is a 20 byte value.
- **Value:** Value represents the total number of Wei to be transferred to the recipient; in the case of a CA, this represents the balance that the contract will hold.
- **Signature:** The signature is composed of three fields, namely V, R, and S. These values represent the digital signature (R, S) and some information that can be used to recover the public key (V). Also, the sender of the transaction can also be determined from these values. The signature is based on the ECDSA scheme and makes use of the secp256k1 curve.
- **Init:** The Init field is used only in transactions that are intended to create contracts, that is, contract creation transactions. This represents a byte array of unlimited length that specifies the EVM code to be used in the account initialization process. It (init) gets destroyed immediately after that. Init also returns another code section called the body, which persists and runs in response to message calls that the CA

may receive. These message calls may be sent via a transaction or an internal code execution.

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

The relationship between the transaction, transaction trie, and block header

- A transaction is a tuple of the fields, which is then included in a transaction trie (a modified Merkle-Patricia tree (MPT)) composed of the transactions to be included.
- Finally, the root node of the transaction trie is hashed using a Keccak 256-bit algorithm and is included in the block header along with a list of transactions in the block:
- A block is a data structure that contains batches of transactions. Transactions can be found in either transaction pools or blocks.
- In transaction pools, they wait for verification by a node, and in blocks, they are added after successful verification.
- When a mining node starts its operation of verifying blocks, it starts with the highest-paying transactions in the transaction pool and executes them one by one.
- When the gas limit is reached, or no more transactions are left to be processed in the transaction pool, the mining starts.
- In this process, the block is repeatedly hashed until a valid nonce is found, such that once hashed with the block, it results in a value less than the difficulty target.

- Once the block is successfully mined, it will be broadcasted immediately to the network, claiming success, and will be verified and accepted by the network.
- This process is similar to Bitcoin's mining process.
- Ethereum's PoW algorithm is called Ethash, and it was originally intended to be ASIC-resistant where finding a nonce requires large amounts of memory.
- However, now some ASICs are available for Ethash too. Consequently, there is also an agreement among Ethereum core developers to implement ProgPow, a new, more ASIC-resistant PoW consensus mechanism.

RLP

- Serialization is simply a mechanism commonly used in computing to encode data structures into a format (a sequence of bytes) that is suitable for storage and/or transmission over the communication links in a network.
- Once a receiver receives the serialized data, it de-serializes it to obtain the original data structure.
- Serialization and deserialization are also referred as marshaling and un-marshaling, respectively.
- Some commonly used serialization formats include XML, JSON, YAML, protocol buffers, and XDR. There are two types of serialization formats, namely text and binary.
- In a blockchain, there is a need to serialize and deserialize different types of data such as blocks, accounts, and transactions to support transmission over the network and storage on clients.
- RLP is an encoding scheme developed by Ethereum developers to serialize binary data for storage or transmission over the network and also to save the state in a Patricia tree on storage media.
- It is a deterministic and consistent binary encoding scheme used to serialize objects on the Ethereum blockchain, such as account state, transactions, messages, and blocks.

- It operates on strings and lists to produce raw bytes that are suitable for storage and transmission.
- RLP is a minimalistic and simple-to-implement serialization format that does not define any data types and simply stores structures as nested arrays.

Contract creation transactions

A contract creation transaction is used to create smart contracts on the blockchain. There are a few essential parameters required for a contract creation transaction.

The parameters are as follows:

- The sender
- The transaction originator
- Available gas
- Gas price
- Endowment, which is the amount of ether allocated
- A byte array of an arbitrary length
- Initialization EVM code
- The current depth of the message call/contract-creation stack (current depth means the number of items that are already present in the stack)
- Addresses generated as a result of a contract creation transaction are 160 bits in length. Precisely they are the rightmost 160 bits of the Keccak hash of the RLP encoding of the structure containing only the sender and the nonce.
- Initially, the nonce in the account is set to zero. The balance of the account is set to the value passed to the contract. The storage is also set to empty. The code hash is a Keccak 256-bit hash of the empty string.
- The new account is initialized when the EVM code (the initialization EVM code) is executed.
- In the case of any exception during code execution, such as not having enough gas (running Out of Gas, or OOG), the state does not change.

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

Message call transactions

A message call requires several parameters for execution:

- The sender
 - The transaction originator
 - The recipient
 - The account whose code is to be executed (usually the same as the recipient)
 - Available gas
 - The value
 - The gas price
 - An arbitrary-length byte array
 - The input data of the call
 - The current depth of the message call/contract creation stack
-
- Message calls result in a state transition. Message calls also produce output data, which is not used if transactions are executed.
 - In cases where message calls are triggered by EVM code, the output produced by the transaction execution is used.
 - A message call is the act of passing a message from one account to another. If the destination account has an associated EVM code, then the EVM will start upon the receipt of the message to perform the required operations.
 - If the message sender is an autonomous object (external actor), then the call passes back any data returned from the EVM operation.

- The state is altered by transactions. These transactions are created by external factors (users) and are signed and then broadcasted to the Ethereum network.
- Messages are passed between accounts using message calls.

Messages

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

A message consists of the following components:

- The sender of the message
- The recipient of the message
- Amount of Wei to transfer and the message to be sent to the contract address
- An optional data field (input data for the contract)
- The maximum amount of gas (startgas) that can be consumed

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

Types of transactions and the required parameters for execution

Calls

- A call does not broadcast anything to the blockchain; instead, it is a local call and executes locally on the Ethereum node.
- It is almost like a local function call. It does not consume any gas as it is a read-only operation. It is akin to a dry run or a simulated run.
- Calls also do not allow ether transfer to CAs. Calls are executed locally on a node EVM and do not result in any state change because they are never mined.
- Calls are processed synchronously and they usually return the result immediately
- A call basically runs message call transactions locally on the client and never costs gas nor results in a state change. A message call transaction, results in a state change.
- It is available in the web3.js JavaScript API and can be seen as almost a simulated mode of the message call transaction.

- On the other hand, a message call transaction is a write operation and is used for invoking functions in a CA (Contract Account, or smart contract), which does cost gas and results in a state change.

Transaction validation and execution

Transactions are executed after their validity has been verified. The initial checks are listed as follows:

- A transaction must be well formed and RLP-encoded without any additional trailing bytes
- The digital signature used to sign the transaction must be valid
- The transaction nonce must be equal to the sender's account's current nonce
- The gas limit must not be less than the gas used by the transaction
- The sender's account must contain sufficient balance to cover the execution cost

The transaction substate

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

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

Log series: Logs are created in response to events occurring in the smart contract. It can also be used as a cheaper form of storage. It works like a trigger mechanism that is executed every time a specific function is invoked, or a specific event occurs.

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

Touched accounts: Touched accounts can be defined as those accounts which are involved any potential state changing operation. Empty accounts from this set are deleted at the end of the transaction. The final state is reached after

deleting accounts in the self-destruct set and emptying accounts from the touched accounts set.

State storage in the Ethereum blockchain

- At a fundamental level, the Ethereum blockchain is a transaction- and consensus-driven state machine.
- The state needs to be stored permanently in the blockchain. For this purpose, the world state, transactions, and transaction receipts are stored on the blockchain in blocks.

The world state

- This is a mapping between Ethereum addresses and account states.
- The world state is a huge mapping (like a database) that holds the current status of all accounts on Ethereum — both externally owned accounts (EOAs) and smart contracts.
- The addresses are 20 bytes (160 bits) long. This mapping is a data structure that is serialized using RLP.

The account state

The account state consists of four fields: nonce, balance, storage root, and code hash:

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

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

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

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

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

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

Transaction receipts

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

It is composed of four elements:

The post-transaction state: This item is a trie structure that holds the state after the transaction has been executed. It is encoded as a byte array.

Gas used: This item represents the total amount of gas used in the block that contains the transaction receipt. The total gas used is expected to be a non-negative integer.

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

The bloom filter: A bloom filter is created from the information contained in the set of logs. A log entry is reduced to a hash of 256 bytes, which is then embedded in the header of the block as the logs bloom. A log entry is composed of the logger's address, log topics, and log data. Log topics are encoded as a series of 32-byte data structures. The log data is made up of a few bytes of data.

Transaction receipts and logs bloom

Ether cryptocurrency/tokens (ETC and ETH)

- As an incentive to the miners, Ethereum rewards its own native currency called ether (ETH).
- After the Decentralized Autonomous Organization (DAO) hack, Smart Contracts, a hard fork was proposed in order to mitigate the issue; therefore, there are now two Ethereum blockchains:
- one is called Ethereum Classic, and its currency is represented by ETC, whereas the hard-forked version is ETH, which continues to grow and on which active development is being carried out.
- ETC, however, has its own following with a dedicated community that is further developing ETC, which is the unforked original version of Ethereum.
- ETH is currently the most active and official Ethereum blockchain. Ether is minted by miners as a currency reward for the computational effort they spend to secure the network by verifying transactions and blocks.
- Ether is used within the Ethereum blockchain to pay for the execution of contracts on the EVM.
- Ether is used to purchase gas as crypto fuel, which is required to perform computation on the Ethereum blockchain.

Unit	Alternative name	Wei value	Number of weis
Wei	Wei	1 Wei	1
KWei	Babbage	1e3 Wei	1,000
MWei	Lovelace	1e6 Wei	1,000,000
GWei	Shannon	1e9 Wei	1,000,000,000
microether	Szabo	1e12 Wei	1,000,000,000,000
milliether	Finney	1e15 Wei	1,000,000,000,000,000
ether	ether	1e18 Wei	1,000,000,000,000,000,000

The denomination table

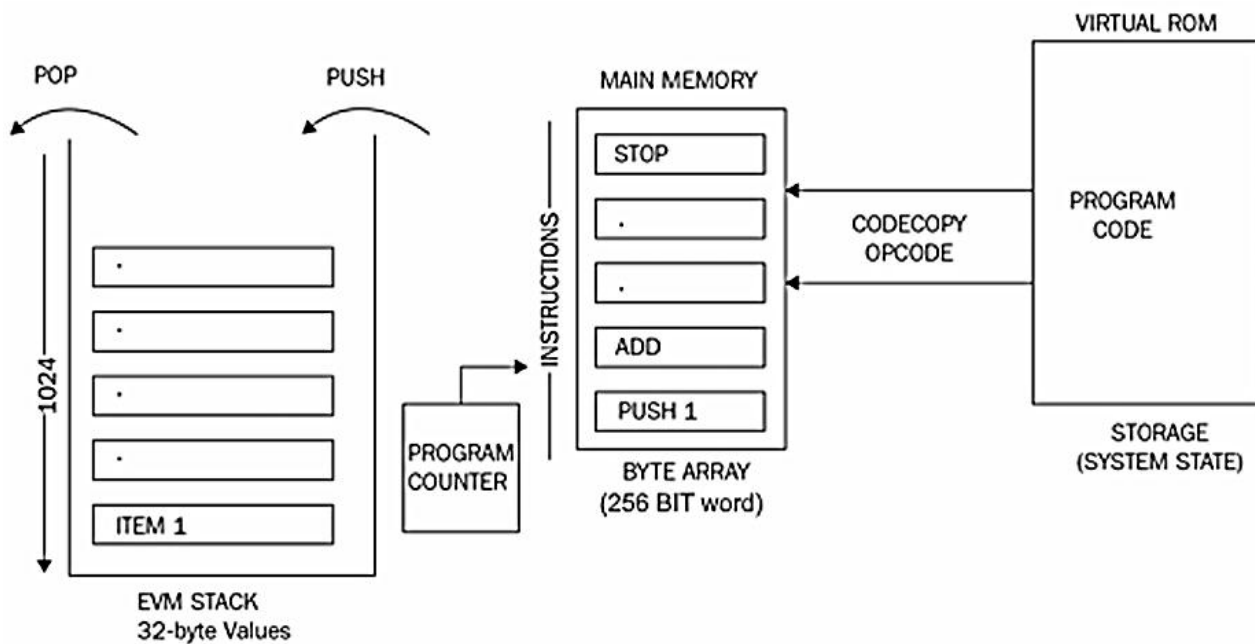
The Ethereum Virtual Machine (EVM)

- The EVM is a simple stack-based execution machine that runs bytecode instructions to transform the system state from one state to another.
- The word size of the EVM is set to 256-bit. The stack size is limited to 1,024 elements and is based on the Last In, First Out (LIFO) queue.

- The EVM is a Turing-complete machine but is limited by the amount of gas that is required to run any instruction.
- This means that infinite loops that can result in denial-of-service attacks are not possible due to gas requirements.
- The EVM also supports exception handling should exceptions occur, such as not having enough gas or providing invalid instructions, in which case the machine would immediately halt and return the error to the executing agent.
- The EVM is an entirely isolated and sandboxed runtime environment. The code that runs on the EVM does not have access to any external resources such as a network or filesystem.
- This results in increased security, deterministic execution, and allows untrusted code (code that can be run by anyone) to be executed on Ethereum blockchain.
- The EVM is big-endian by design, and it uses 256-bit-wide words. This word size allows for Keccak 256-bit hash and ECC computations.

There are three main types of storage available for contracts and the EVM:

1. **Memory:** The first type is called memory or volatile memory, which is a word-addressed byte array. When a contract finishes its code execution, the memory is cleared. write operations to the memory can be of 8 or 256 bits, whereas read operations are limited to 256-bit words. Memory is unlimited but constrained by gas fee requirements.
 2. **Storage:** The other type is called storage, which is a key-value store and is permanently persisted on the blockchain. Keys and values are each 256 bits wide. It is allocated to all accounts on the blockchain. As a security measure, storage is only accessible by its own respective CAs. It can be thought of as hard disk storage.
 3. **Stack:** EVM is a stack-based machine, and thus performs all computations in a data area called the stack. All in-memory values are also stored in the stack. It has a maximum depth of 1024 elements and supports the word size of 256 bits.
- The program code is stored in virtual read-only memory (virtual ROM) that is accessible using the CODECOPY instruction. The CODECOPY instruction copies the program code into the main memory. Initially, all storage and memory are set to zero in the EVM.
 - The main memory is then read by the EVM by referring to the program counter and executes instructions step by step.
 - The program counter and EVM stack are updated accordingly with each instruction execution.



EVM operation

Execution environment

There are some key elements that are required by the execution environment to execute the code.

The key parameters are provided by the execution agent; for example, a transaction. These are listed as follows:

- The system state.
- The remaining gas for execution.
- The address of the account that owns the executing code.
- The address of the sender of the transaction. This is the originating address of this execution (it can be different from the sender).
- The gas price of the transaction that initiated the execution.
- Input data or transaction data depending on the type of executing agent. This is a byte array; in the case of a message call, if the execution agent is a transaction, then the transaction data is included as input data.
- The address of the account that initiated the code execution or transaction sender. This is the address of the sender in case the code execution is initiated by a transaction; otherwise, it is the address of the account.
- The value or transaction value. This is the amount in Wei. If the execution agent is a transaction, then it is the transaction value.

- The code to be executed presented as a byte array that the iterator function picks up in each execution cycle.
- The block header of the current block.
- The number of message calls or contract creation transactions (CALLs, CREATEs or CREATE2s) currently in execution.
- Permission to make modifications to the state.

The execution environment can be visualized as a tuple of ten elements, as follows:

Address of code owner
Sender address
Gas price
Input data
Initiator address
Value
Bytecode
Block header
Message call depth
Permission

The execution results in producing the resulting state, the gas remaining after the execution, the self-destruct or suicide set, log series, and any gas refunds.

The machine state

- The machine state is also maintained internally, and updated after each execution cycle of the EVM.
- An iterator function runs in the EVM, which outputs the results of a single cycle of the state machine.

The machine state is a tuple that consists of the following elements:

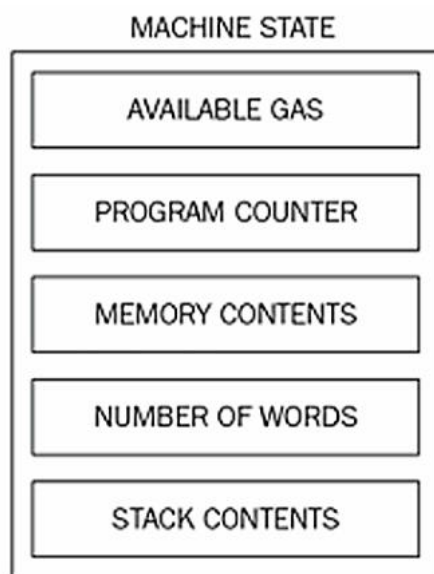
- Available gas
- The program counter, which is a positive integer of up to 256
- The contents of the memory (a series of zeroes of size 2256)
- The active number of words in memory (counting continuously from position 0)

- The contents of the stack

The EVM is designed to handle exceptions and will halt (stop execution) if any of the following exceptions should occur:

- Not having enough gas required for execution
- Invalid instructions
- Insufficient stack items
- Invalid destination of jump opcodes
- Invalid stack size (greater than 1,024)

Machine state can be viewed as a tuple:



The iterator function

The iterator function performs various vital functions that are used to set the next state of the machine and eventually the world state.

These functions include the following:

- It fetches the next instruction from a byte array where the machine code is stored in the execution environment.
- It adds/removes (PUSH/POP) items from the stack accordingly.
- Gas is reduced according to the gas cost of the instructions/opcodes. It increments the Program Counter (PC).

The EVM is also able to halt in normal conditions if STOP, SUICIDE, or RETURN opcodes are encountered during the execution cycle.

Blocks and blockchain

Blocks are the main building structure of a blockchain. Ethereum blocks consist of various elements:

- The block header
- The transactions list
- The list of headers of stale blocks

Block header:

The Block contains various elements:

Parent hash: This is the Keccak 256-bit hash of the parent (previous) block's header.

Ommers hash: This is the Keccak 256-bit hash of the list of ommers blocks included in the block.

The beneficiary: The beneficiary field contains the 160-bit address of the recipient that will receive the mining reward once the block is successfully mined.

State root: The state root field contains the Keccak 256-bit hash of the root node of the state trie. It is calculated once all transactions have been processed and finalized.

Transactions root: The transaction root is the Keccak 256-bit hash of the root node of the transaction trie. The transaction trie represents the list of transactions included in the block.

Receipts root: The receipts root is the Keccak 256-bit hash of the root node of the transaction receipt trie. This trie is composed of receipts of all transactions included in the block.

Logs bloom: The logs bloom is a bloom filter that is composed of the logger address and log topics from the log entry of each transaction receipt of the included transaction list in the block.

Difficulty: The difficulty level of the current block.

Number: The total number of all previous blocks; the genesis block is block zero.

Gas limit: This field contains the value that represents the limit set on the gas consumption per block.

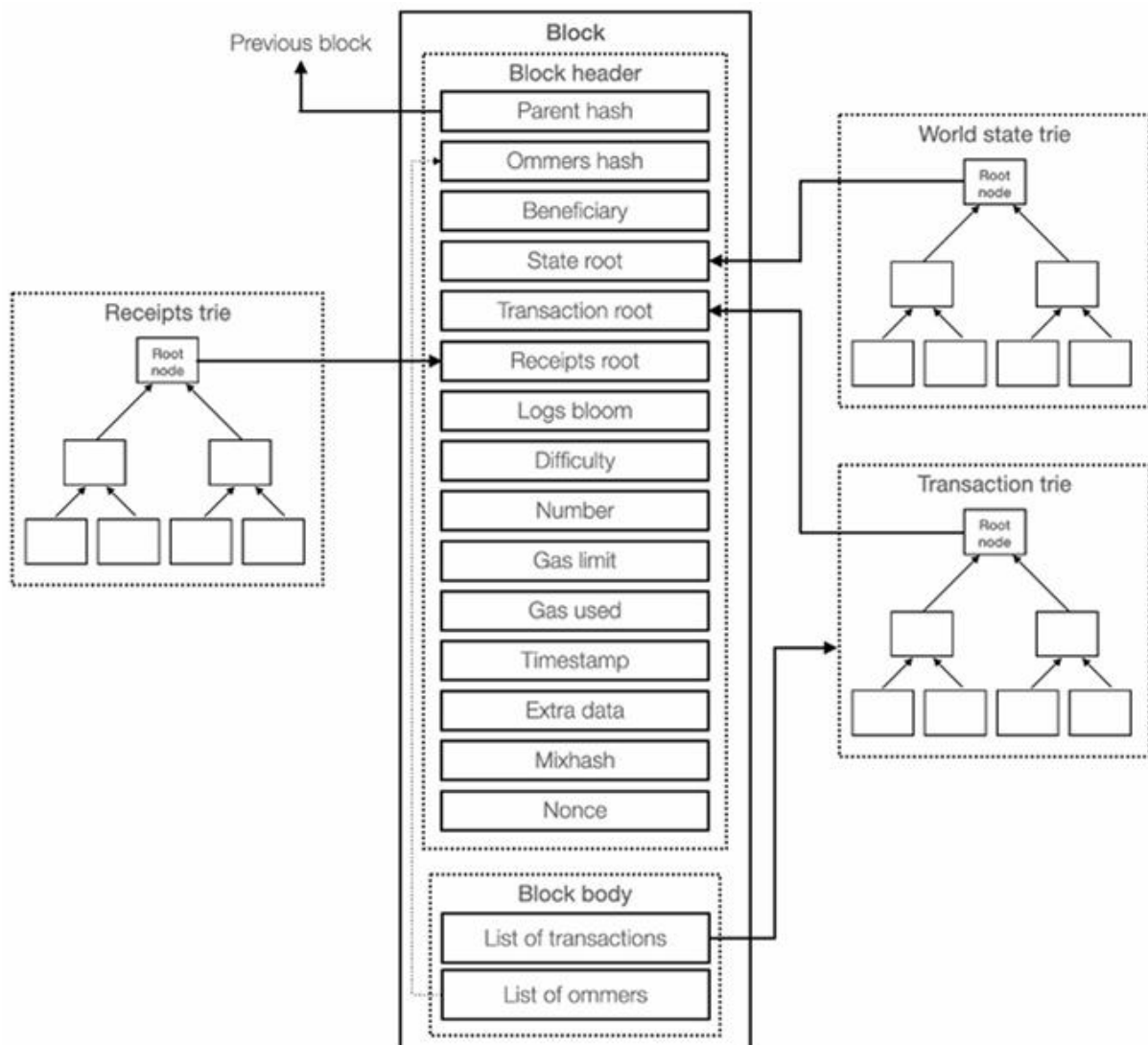
Gas used: This field contains the total gas consumed by the transactions included in the block.

Timestamp: The timestamp is the epoch Unix time of the time of block initialization.

Extra data: The extra data field can be used to store arbitrary data related to the block. Only up to 32 bytes are allowed in this field.

Mixhash: The mixhash field contains a 256-bit hash that, combined with the nonce. It is used to prove that adequate computational effort (Proof of Work, or PoW) has been spent in order to create this block.

Nonce: Nonce is a 64-bit hash (a number). Miners change this value repeatedly in order to try to find a valid block hash that meets the difficulty target.



Block structure with a block header and relationship with tries

The genesis block

- The genesis block is the first block in a blockchain network. It varies slightly from normal blocks due to the data it contains and the way it has been created.
- It contains 15 items:

The block validation mechanism

An Ethereum block is considered valid if it passes the following checks:

- If it is consistent with ommers and transactions. This means that all ommers satisfy the property that they are indeed uncles and also if the PoW is valid.
- If the previous block (parent) exists and is valid.
- If the timestamp of the block is valid. This means that the current block's timestamp must be higher than the parent block's timestamp. Also, it should be less than 15 minutes into the future. All block times are calculated in epoch time (Unix time).

If any of these checks fails, the block will be rejected. A list of errors for which the block can be rejected is presented here:

- The timestamp is older than the parent
- There are too many ommers
- There is a duplicate ommers
- The ommers is an ancestor
- The ommers 's parent is not an ancestor
- There is non-positive difficulty
- There is an invalid mix digest

- There is an invalid PoW

Block finalization

Block finalization is a process that is run by miners to validate the contents of the block and apply rewards.

It results in four steps being executed:

1. **Ommers validation:** The validation process of the headers of stale blocks checks whether the header is valid and whether the relationship between the ommers and the current block satisfies the maximum depth of six blocks. A block can contain a maximum of two ommers.
2. **Transaction validation:** This process involves checking whether the total gas used in the block is equal to the final gas consumption after the final transaction.
3. **Reward application:** Apply rewards, which means updating the beneficiary's account with a reward balance. In Ethereum, a reward is also given to miners for stale blocks, which is $1/32$ of the block reward. Ommers that are included in the blocks also receive $7/8$ of the total block reward. The current block reward is 2 ether. A block can have a maximum of two ommers.
4. **State and nonce validation:** Verify the state and block nonce. In the case of mining, compute a valid state and block nonce.

How a block is received and processed by a node. And how it is updated in the local blockchain:

1. When an Ethereum full node receives a newly mined block, the header and the body of the block are detached from each other. The roots of the MPTs or tries are present in each block header as a state trie root node, a transaction trie root node, and a receipt trie root node.
2. A new MPT is constructed that comprises all transactions from the block.
3. All transactions from this new MPT are executed one by one in a sequence. This execution occurs locally on the node within the Ethereum Virtual Machine (EVM). As a result of this execution, new transaction receipts are generated that are organized in a new receipts MPT. Also, the global state is modified accordingly, which updates the state MPT (trie).
4. The root nodes of each respective trie, the state root, transaction root, and receipts root are compared with the header of the block that was split in the first step. If both the roots of the newly constructed tries and

the trie roots that already exist in the header are equal, then the block is verified and valid.

5. Once the block is validated, new transaction, receipt, and state tries are written into the local blockchain database.

Block difficulty mechanism

Block difficulty is increased if the time between two blocks decreases. This is required to maintain a roughly consistent block generation time. The difficulty adjustment algorithm in Ethereum's Homestead release is as follows:

```
block_diff = parent_diff + parent_diff // 2048 *  
    max(1 - (block_timestamp - parent_timestamp) // 10, -99) +  
    int(2 ** ((block.number // 100000) - 2))
```

- If the time difference between the generation of the parent block and the current block is less than 10 seconds, the difficulty goes up by $\text{parent_diff} // 2048 * 1$.
- If the time difference is between 10 and 19 seconds, the difficulty level remains the same.
- Finally, if the time difference is 20 seconds or more, the difficulty level decreases. This decrease is proportional to the time difference from $\text{parent_diff} // 2048 * -1$ to a maximum decrease of $\text{parent_diff} // 2048 * -99$.
- In addition to timestamp difference-based difficulty adjustment, there is also another part that increases the difficulty exponentially after every 100,000 blocks.
- This is called difficulty time bomb, or ice age, which will make it very hard to mine on the Ethereum blockchain at some point in the future. This will encourage users to switch to Proof of Stake (PoS).

Gas

- Gas is required to be paid for every operation performed on the Ethereum blockchain.
- This is a mechanism that ensures that infinite loops cannot cause the whole blockchain to stall due to the Turing-complete nature of the EVM.

- A transaction fee is charged as an amount of Ether and is taken from the account balance of the transaction originator.
- A fee is paid for transactions to be included by miners for mining.
- If this fee is too low, the transaction may never be picked up; the more the fee, the higher are the chances that the transactions will be picked up by the miners for inclusion in the block.
- Conversely, if the transaction that has an appropriate fee paid is included in the block by miners but has too many complex operations to perform, it can result in an out-of-gas exception if the gas cost is not enough.
- In this case, the transaction will fail but will still be made part of the block, and the transaction originator will not get any refund.
- Transaction costs can be estimated using the following formula:

$$\text{Total cost} = \text{gasUsed} * \text{gasPrice}$$

- Here, gasUsed is the total gas that is supposed to be used by the transaction during the execution, and gasPrice is specified by the transaction originator as an incentive to the miners to include the transaction in the next block.
- Each EVM opcode has a fee assigned to it. It is an estimate because the gas used can be more or less than the value specified by the transaction originator originally.
- For example, if computation takes too long or the behavior of the smart contract changes in response to some other factors, then the transaction execution may perform more or fewer operations than intended initially and can result in consuming more or less gas.
- If the execution runs out of gas, everything is immediately rolled back; otherwise, if the execution is successful and some gas remains, then it is returned to the transaction

A high-level fee schedule of a few operations:

Operation name	Gas cost
Stop	0
SHA3	30
SLOAD	800
Transaction	21000
Contract creation	32000

Example calculation of the SHA-3 operation :

- SHA-3 costs 30 gas.
- Assume that the current gas price is 25 GWei, and convert it into ETH, which is 0.000000025 ETH. After multiplying both, $0.000000025 * 30$, we get 0.00000075 ETH.
- In total, 0.00000075 ETH is the total gas that will be charged.

Fee schedule

Gas is charged in three scenarios as a prerequisite to the execution of an operation:

- The computation of an operation
- For contract creation or message calls
- An increase in the use of memory

Wallets and client software

As Ethereum is under heavy development and evolution, there are many components, clients, and tools that have been developed and introduced over the last few years.

Wallets

A wallet is a generic program that can store private keys and, based on the addresses stored within it, it can compute the existing balance of ether associated with the addresses by querying the blockchain.

It can also be used to deploy smart contracts.

Geth

This is the official Go implementation of the Ethereum client.

Eth

This is the C++ implementation of the Ethereum client.

Parity

This implementation is built using Rust and developed by Parity technologies.

Trinity

Trinity is the implementation of the Ethereum protocol. It is written in Python.

Light clients

- Simple Payment Verification (SPV) clients download only a small subset of the blockchain.
- This allows low resource devices, such as mobile phones, embedded devices, or tablets, to be able to verify the transactions.
- A complete Ethereum blockchain and node are not required in this case, and SPV clients can still validate the execution of transactions.
- SPV clients are also called light clients. This idea is similar to Bitcoin SPV clients.
- The critical difference between clients and wallets is that clients are full implementations of the Ethereum protocol, which support mining, account management, and wallet functions.
- In contrast, wallets only store the public and private keys, provide essential account management, and interact with the blockchain for usually only payment (transfer of funds) purposes.

Installation and usage

Geth

The following installation procedure describes the installation of Ethereum clients on macOS and Linux.

The Geth client can be installed by using the following command on an Ubuntu system:

```
$ sudo apt-get install -y software-properties-common
```

```
$ sudo add-apt-repository -y ppa:ethereum/ethereum
```

```
$ sudo apt-get update
```

```
$ sudo apt-get install -y Ethereum
```

On a macOS using homebrew, Geth can be installed by executing the following commands:

```
$ brew tap ethereum/ethereum
```

```
$ brew install Ethereum
```

Once installation is complete, Geth can be launched simply by issuing the geth command at the terminal. It comes preconfigured with all the required parameters to connect to the live Ethereum network (mainnet)

```
$ geth
```

When the Ethereum client starts up, it starts to synchronize with the rest of the network.

There are three types of synchronization mechanisms available, namely, full, fast, and light.

1. Full:

- In this synchronization mode, the Geth client downloads the complete blockchain to its local node.
- Currently (as at early 2020), the Ethereum blockchain size is roughly 210 GB, and downloading and maintaining that could be a problem.
- Usually, SSDs are recommended for a full node, so that disk latency cannot cause any processing delays.

2. Fast:

- The client downloads the full blockchain, but it retrieves and verifies only the previous 64 blocks from the current block.
- After this, it verifies the new blocks in full. It does not replay and verify all historic transactions since the genesis block; instead it only does the state downloads.

- This also reduces the on-disc size of the blockchain database quite significantly. This is the default mode of Geth client synchronization.

3. Light:

- This is the quickest mode and only downloads and stores the current state trie.
- In this mode, the client does not download any historic blocks and only processes newer blocks.

Synchronization mode is configurable on the geth client via the flag:

--syncmode value

Here, the value can be either fast, full, or light.

Ethereum account management using Geth

New accounts can be created via the command line using Geth or OpenEthereum command-line interface.

Creating a Geth new account

Execute the following command to add a new account:

\$ geth account new

The list of accounts can be displayed using the geth client by the following command:

\$ geth account list

How to query the blockchain using Geth

There are different methods available to query with the blockchain. First, in order to connect to the running instance of the client, either a local IPC or RPC API can be used.

There are three methods of interacting with the blockchain using Geth:

1. Geth console
2. Geth attach
3. Geth JSON RPC

Geth console

Geth can be started in console mode by running the following command:

```
$ geth console
```

This will also start the interactive JavaScript environment in which JavaScript commands can be run to interact with the Ethereum blockchain.

Geth attach

When a geth client is already running, the interactive JavaScript console can be invoked by attaching to that instance. This is possible by running the geth attach command.

Geth can be attached with the running daemon, as shown below:

```
$ geth attach
```

Once Geth is successfully attached with the running instance of the Ethereum client, it will display the command prompt >.

A new account can be added using the following command in the Geth console:

```
> personal.newAccount()
```

Ethereum keystore

- Accounts are represented by private keys. The public and private key pair generated by the new account creation process is stored in key files located locally on the disk.
- These key files are stored in the keystore directory present in the relevant path according to the OS in use.
- On the Linux OS, its location is : ~/.ethereum/keystore
- The private key is stored in an encrypted format in the keystore file. It is generated when a new account is created using the password and private key.
- The keystore file is also referred to as a UTC file as the naming format of it starts with UTC with the date timestamp therein.
- A sample name of the keystore file is shown here:

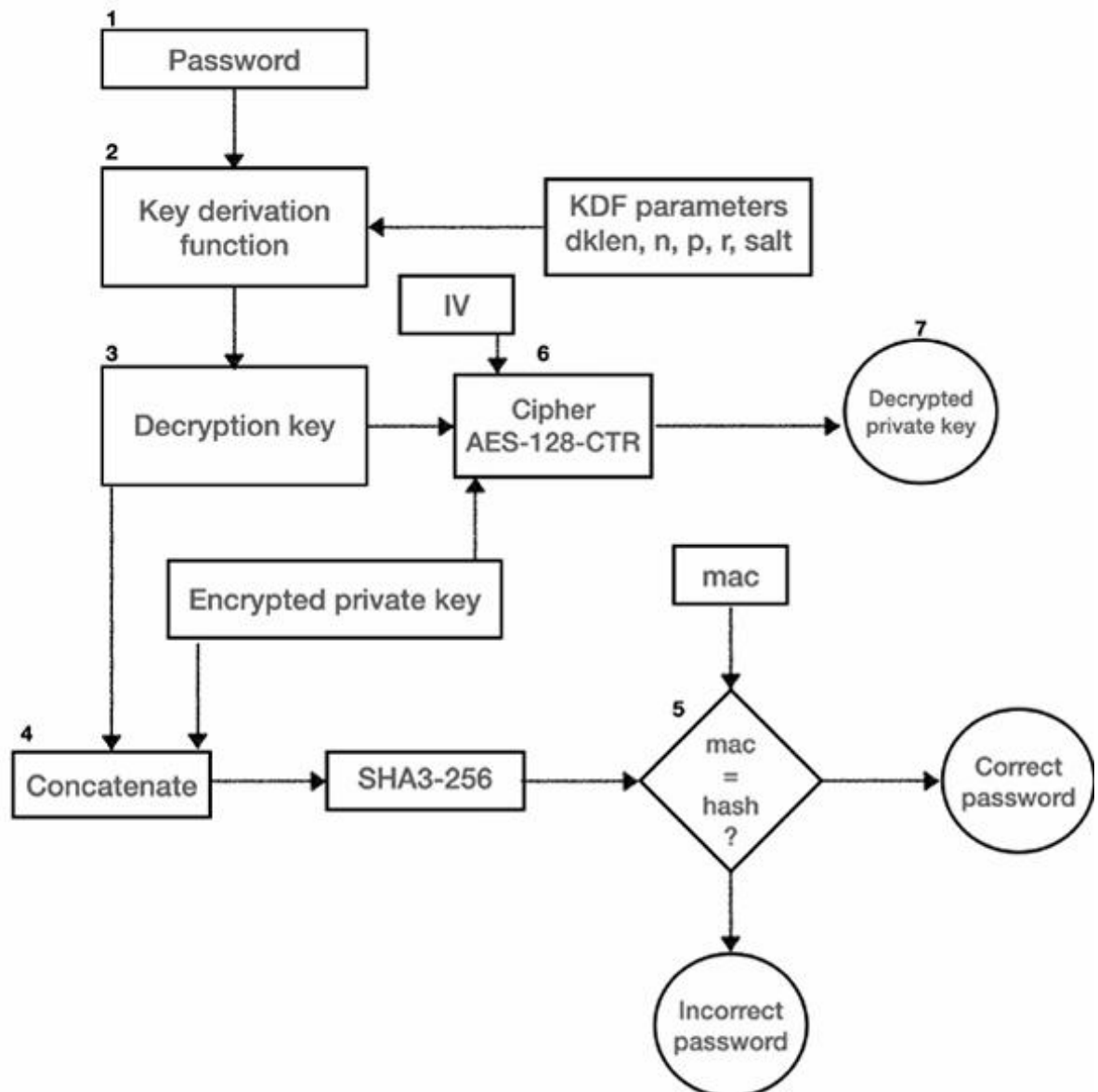
UTC--2020-

0118T174646.604174215Zba94fb1f306e4d53587fcdcd7eab8109a2e183c4

- On macOS, the keystore file is stored at the following location:
~/Library/Ethereum/keystore/
- As the keystore file is present on the disk, it is very important to keep it safe. It is recommended that it is backed up as well.
- If the keystore files are lost or overwritten or somehow corrupted, there is no way to recover them. This means that any ether associated with the private key will be irrecoverable, too.

Elements of the password keystore file:

- **Address:** This is the public address of the account that is used to identify the sender or receiver.
- **Crypto:** This field contains the cryptography parameters.
- **Cipher:** This is the cipher used to encrypt the private key. In the following diagram, AES-128-CTR indicates the Advanced Encryption Standard – 128 bit – in counter mode.
- **Ciphertext:** This is the encrypted private key.
- **Cipherparams:** This represents the parameters required for the encryption algorithm, AES-128-CTR.
- **IV:** This is the 128-bit initialization vector for the encryption algorithm.
- **KDF:** This is the key derivation function. It is scrypt in this case.
- **KDFParams:** These are the parameters for the key derivation function (KDF).
- **Dklen:** This is the derived key length. It is 32 in our example.
- **N:** This is the iteration count.
- **P:** This is the parallelization factor; the default value is 1.
- **R:** This is the block size of the underlying hash function. It is set to 8, which is the default setting.
- **Salt:** This is the random value of salt for the key derivation function, KDF.
- **Mac:** This is the Keccak-256 hash output obtained following concatenation of the second leftmost 16 bytes of the derived key together with the ciphertext.
- **ID:** This is a random identification number.
- **Version:** The version number of the file format, currently version 3.



Private key decryption process

1. First, the password is fed into the KDF.
2. KDF takes several parameters, namely, dklen, n, p, r, and salt, and produces a decryption key.
3. The decryption key is concatenated with an encrypted private key (ciphertext). The decryption key is also fed into the cipher algorithm, AES-128-CTR.
4. The concatenated decryption key and ciphertext are hashed using the SHA3-256 hash function.
5. mac is fed into the checking function where the hash produced from step 4 is compared with the mac. If both match, then the password is valid, otherwise the password is incorrect.

6. The cipher function, which is fed with the initialization vector (IV), encrypted private key (ciphertext), and decryption key, decrypts the encrypted private key.
7. The decrypted private key is produced.

Nodes and miners

- The Ethereum network contains different nodes. Some nodes act only as wallets, some are light clients, and a few are full clients running the full blockchain.
- One of the most important types of nodes are mining nodes.
- As a result of the mining operation, currency (ether) is awarded to the nodes that perform mining operations.
- These mining nodes are known as miners. Miners are paid in ether as an incentive for them to validate and verify blocks made up of transactions.
- The mining process helps secure the network by verifying computations.

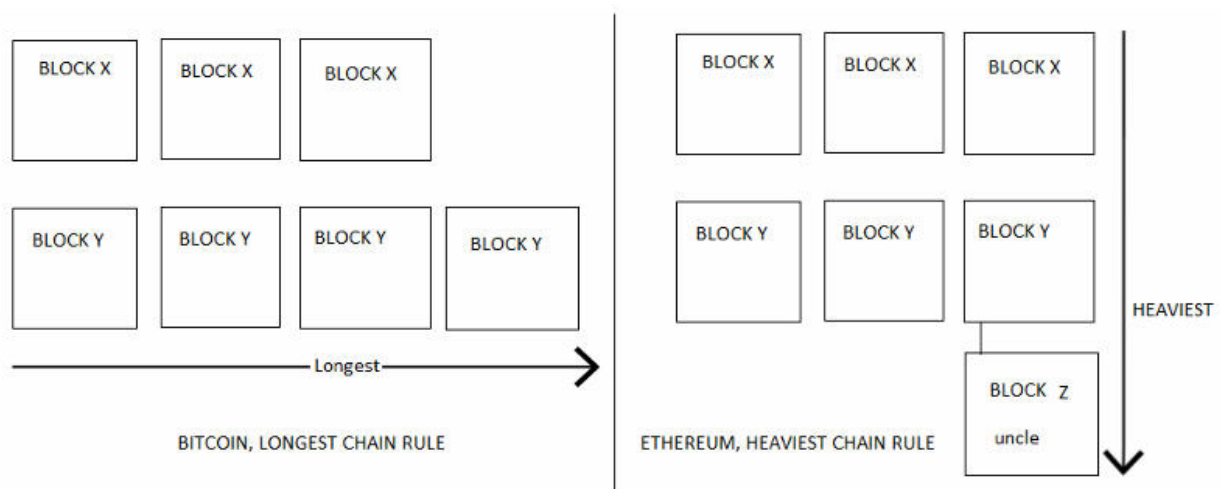
A miner node performs the following functions:

- It listens for the transactions broadcasted on the Ethereum network and determines the transactions to be processed.
- It determines stale ommer blocks and includes them in the blockchain.
- It updates the account balance with the reward earned from successfully mining the block.
- Finally, a valid state is computed, and the block is finalized, which defines the result of all state transitions.
- The current method of mining is based on PoW, which is similar to that of Bitcoin.
- When a block is deemed valid, it has to satisfy not only the general consistency requirements, but it must also contain the PoW for a given difficulty.
- The PoW algorithm is due to be replaced by the PoS algorithm with the release of Serenity.
- An algorithm named Casper has been developed that will replace the existing PoW algorithm in Ethereum.

- This is a security deposit based on the economic protocol where nodes are required to place a security deposit before they can produce blocks.
- Nodes have been named bonded validators in Casper, whereas the act of placing the security deposit is named bonding.
- Miners play a vital role in reaching a consensus regarding the canonical state of the blockchain.

The consensus mechanism

- The consensus mechanism in Ethereum is based on the Greedy Heaviest Observed Subtree (GHOST) protocol proposed initially by Zohar and Sompolinsky in December 2013.
- Ethereum uses a simpler version of this protocol, where the chain that has the most computational effort spent on it to build it is identified as the definite version.
- Another way of looking at it is to find the longest chain.
- The GHOST protocol was first introduced as a mechanism to alleviate the issues arising out of fast block generation times that led to stale or orphaned blocks.
- In GHOST, stale blocks, or ommers, are added in calculations to figure out the longest and heaviest chain of blocks.



Longest versus the heaviest chain

- The preceding diagram shows two rules of figuring out which blockchain is the canonical version of truth.
- In the case of Bitcoin, shown on the left-hand side in the diagram, the longest chain rule is applied, which means that the active chain (true chain) is the one that has the most amount of PoW done.
- In the case of Ethereum, the concept is similar from the point of view of the longest chain, but it also includes ommers, the orphaned blocks, which means that it also rewards those blocks that were competing with other blocks during mining to be selected and performed significant PoW, or were mined exactly at the same time as others but did not make it to the main chain.
- This makes the chain the heaviest instead of the longest because it also contains the orphaned blocks. This is shown on the right-hand side of the diagram.
- As the blockchain progresses (more blocks are added to the blockchain) governed by the consensus mechanism, on occasion, the blockchain can split into two. This phenomenon is called forking.

Forks in the blockchain

- A fork occurs when a blockchain splits into two. This can be intentional or non-intentional.
- Usually, as a result of a major protocol upgrade, a hard fork is created, while an unintentional fork can be created due to bugs in the software.
- It can also be temporary; in other words, the longest and heaviest chain.
- This temporary fork occurs when a block is created almost at the same time and the chain splits into two, until it finds the longest or heaviest chain to achieve eventual consistency.
- The release of Homestead involved major protocol upgrades, which resulted in a hard fork.
- The protocol was upgraded at block number 1,150,000, resulting in the migration from the first version of Ethereum, known as Frontier, to the second version.
- The latest version is called Byzantium, which is the first part of the Metropolis release.

- This was released as a hard fork at block number 4,370,000.
- An unintentional fork, which occurred on November 24, 2016, at 14:12:07 UTC, was due to a bug in Ethereum's Geth client journaling mechanism.
- As a result, a network fork occurred at block number 2,686,351.
- This bug resulted in Geth failing to prevent empty account deletions in the case of the empty out-of-gas exception.
- This was not an issue in Parity (another popular Ethereum client). This means that from block number 2,686,351, the Ethereum blockchain is split into two, one running with the Parity clients and the other with Geth.
- This issue was resolved with the release of Geth version 1.5.3. As a result of the DAO hack, Smart Contracts, the Ethereum blockchain was also forked to recover from the attack.
- To facilitate consensus, a PoW algorithm is used. In Ethereum, the algorithm used for this purpose is called Ethash.

Ethash

- Ethash is the name of the PoW algorithm used in Ethereum. Similar to Bitcoin, the core idea behind mining is to find a nonce (a random arbitrary number), which, once concatenated with the block header and hashed, results in a number that is lower than the current network difficulty level.
- Initially, the difficulty was low when Ethereum was new, and even CPU and single GPU mining was profitable to a certain extent, but that is no longer the case. Now, only either pooled mining or large GPU mining farms are used for profitable mining purposes.
- Ethash is a memory-hard algorithm, which makes it difficult to be implemented on specialized hardware.
- As in Bitcoin, ASICs have been developed, which have resulted in mining centralization over the years, but memory-hard PoW algorithms are one way of thwarting this threat, and Ethereum implements Ethash to discourage ASIC development for mining.
- Ethash is a memory-hard algorithm and developing ASICs with large and fast memories is not feasible.

- This algorithm requires subsets of a fixed resource called Directed Acyclic Graph (DAG) to be chosen, depending on the nonce and block headers.
- DAG is a large, pseudo-randomly generated dataset. This graph is represented as a matrix in the DAG file created during the Ethereum mining process.
- The Ethash algorithm expects the DAG as a two-dimensional array of 32-bit unsigned integers.
- Mining can only start when DAG is completely generated the first time a mining node starts.
- This DAG is used as a seed by the algorithm called Ethash. According to current specifications, the epoch time is defined as 30,000 blocks, or roughly 6 days.
- The Ethash algorithm requires a DAG file to work. A DAG file is generated every epoch, which is 30,000 blocks. DAG grows linearly as the chain size grows. Currently, the DAG size is around 3.5 GB (as of block 9325164) and epoch number 310.

The protocol works as follows:

1. First, the header from the previous block and a 32-bit random nonce is combined using Keccak-256.
 2. This produces a 128-bit structure called mix.
 3. mix determines which data is to be picked up from the DAG.
 4. Once the data is fetched from the DAG, it is "mixed" with the mix to produce the next mix, which is then again used to fetch data from the DAG and subsequently mixed. This process is repeated 64 times.
 5. Eventually, the 64th mix is run through a digest function to produce a 32-byte sequence.
 6. This sequence is compared with the difficulty target. If it is less than the difficulty target, the nonce is valid, and the PoW is solved. As a result, the block is mined. If not, then the algorithm repeats with a new nonce.
- The current reward scheme is 2 ETH for successfully finding a valid nonce.

- In addition to receiving 2 ether, the successful miner also receives the cost of the gas consumed within the block and an additional reward for including stale blocks in the block.
- A maximum of two ommers are allowed per block and are rewarded with 7/8 of the normal block reward.
- In order to achieve a 12-second block time, block difficulty is adjusted at every block.
- The rewards are proportional to the miner's hash rate, which means how fast a miner can hash.

CPU mining

- Even though not profitable on mainnet, CPU mining is still valuable on the test network or even a private network to experiment with mining and contract deployment.

- Geth can be started with a mine switch in order to initiate mining:

```
$ geth --mine --minerthreads
```

- CPU mining can also be started using the Web3 Geth console. The Geth console can be started by issuing the following command:

```
$ geth attach
```

- After this, the miner can be started by issuing the following command, which will return True if successful, or False otherwise.

```
miner.start(4)
```

```
true
```

- Number 4 here represents the number of threads that will run for mining. It can be any number depending on the number of CPUs. The preceding command will start the miner with four threads.

```
miner.stop()
```

```
True
```

- The preceding command will stop the miner. The command will return True if successful.

GPU mining

- At a basic level, GPU mining can be performed easily by running two commands:

geth -rpc

- Once geth is up and running, and the blockchain is fully downloaded, Ethminer can be run in order to start mining. ethminer is a standalone miner that can also be used in farm mode to contribute to mining pools:

\$ ethminer -G

- Running with the -G switch assumes that the appropriate graphics card is installed and configured correctly. If no appropriate graphics cards are found, ethminer will return an error, as shown in the following screenshot:

```
drequinox@drequinox-OP7010:~$ ethminer -G
[OPENCL]:No OpenCL platforms found
No GPU device with sufficient memory was found. Can't GPU mine. Remove the -G argument
drequinox@drequinox-OP7010:~$
```

- GPU mining requires an AMD or NVIDIA graphics card and an applicable OpenCL SDK.
- Once the graphics cards are installed and configured correctly, the process can be started by issuing the same ethminer -G command.
- Benchmarking ethminer can also be used to run benchmarking. Two modes can be invoked for benchmarking. It can either be CPU or GPU.
- CPU benchmarking:
\$ ethminer -M -C
- GPU benchmarking:
\$ ethminer -M -G

The following screenshot is shown as an example of CPU mining benchmarking:

```
drequinox@drequinox-OP7010:~$ ethminer -M -C
◇ 22:43:30.560 ethminer #00004000...
Benchmarking on platform: 8-thread CPU
Preparing DAG...
□ 22:43:30.561 miner0 Loading full DAG of seedhash: #00000000...
Warming up...
Trial 1... 0
Trial 2... DAG 22:43:38.310 miner0 Generating DAG file. Progress: 0 %
0
Trial 3... 0
Trial 4... DAG 22:43:45.336 miner0 Generating DAG file. Progress: 1 %
0
```

- The GPU device to be used can also be specified in the command line:
\$ ethminer -M -G --opencl-device 1
- As GPU mining is implemented using OpenCL AMD, chipset-based GPUs tend to work faster as compared to NVIDIA GPUs.
- Due to the high memory requirements (DAG creation), FPGAs and ASICs will not provide any major advantage over GPUs.
- This is done on purpose to discourage the development of specialized hardware for mining.

Mining rigs

- As difficulty increased over time in relation to mining ether, mining rigs with multiple GPUs started to be built by the miners.
- A mining rig usually contains around five GPU cards with all of them working in parallel for mining, thereby improving the chances of finding a valid nonce for mining.
- Mining rigs can be built with some effort and are also available commercially from various vendors.
- A typical mining rig configuration includes the components :

Motherboard: A specialized motherboard with multiple PCI-E x1 or x16 slots, for example, BIOSTAR Hi-Fi or ASRock H81, is required.

SSD hard drive: An SSD hard drive is required. The SSD drive is recommended because of its much faster performance than the analog equivalent. This will mainly be used to store the blockchain. It is

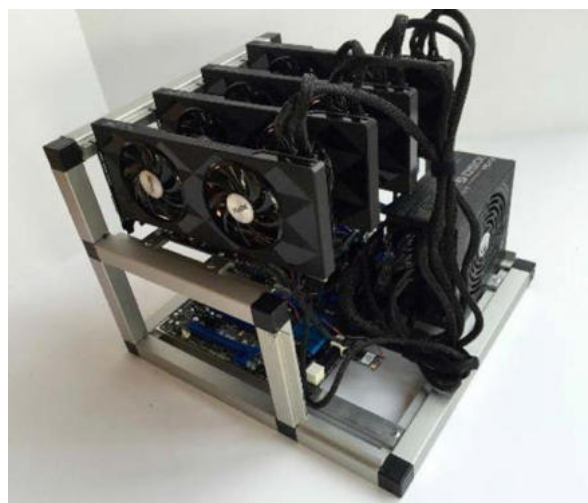
recommended that you have roughly over 350 GB of free space on your hard disk.

GPU: The GPU is the most critical component of the rig as it is the primary workhorse that will be used for mining. For example, it can be a Sapphire AMD Radeon R9 380 with 4 GB RAM.

OS: Linux Ubuntu's latest version is usually chosen as the OS for the rig because it is more reliable and gives better performance as compared to Windows. Also, it allows to run a bare minimum OS required for mining, and essential operations as compared to heavy graphical interfaces that another OS may have. There is also another variant of Linux available, called EthOS that is specially built for Ethereum mining and supports mining operations natively.

Mining software: Finally, mining software such as Ethminer and Geth are installed. Additionally, some remote monitoring and administration software is also installed so that rigs can be monitored and managed remotely if required. It is also important to put proper air conditioning or cooling mechanisms in place, since running multiple GPUs can generate a large amount of heat. This also necessitates the need to use an appropriate monitoring software that can alert users if there are any problems with the hardware, for example, if the GPUs are overheating.

Power supply units (PSUs): In a mining rig, there are multiple GPUs running in parallel. Therefore, there is a need for a constant powerful supply of electricity and it is necessary to use PSUs that can take the load and provide enough power to the GPUs in order for them to operate. Usually, 1,000 watts of power is required to be produced by PSUs.



A mining rig for Ethereum

Mining pools

Many online mining pools offer Ethereum mining. Ethminer can be used to connect to a mining pool. Each pool publishes its instructions, but generally, the process of connecting to a pool is similar.

An example from <http://ethereumpool.co> is shown here:

```
$ ethminer -C -F  
http://ethereumpool.co/?miner=0.1@0x024a20cc5feba7f3dc3776075b  
3e61234eb1459c@DrEquinox
```

This command will produce output similar to the one shown here:

```
miner 23:50:53.046 ethminer Getting work package . . .
```

This output means that the ethminer is running and receiving work from the pool to process.

ASICs

- Even though Ethereum was originally considered ASIC-resistant, there are now ASICs being built for ether mining that perform significantly better than GPUs.
- ASIC resistance in Ethereum comes from the fact that the Ethash consensus algorithm requires a large amount of memory to store the DAG.
- Since maintaining such a large memory of usually around 4 GB is quite difficult on ASICs, this feature offers some resistance against ASICs.
- However, there are some companies that are building Ethereum ASICs, but these are quite difficult to acquire as they are in very high demand.
- A transition from the existing Ethash consensus algorithm to ProgPoW has also been agreed among developers, which is also slowing down ASIC hardware adoption.
- However, the difficulty is increasing on the Ethereum network due the existence of these ASIC devices.
- Eventually, a move to a PoS system in Ethereum 2.0 will permanently address this issue.