Module -4

Ethereum 101

Introduction

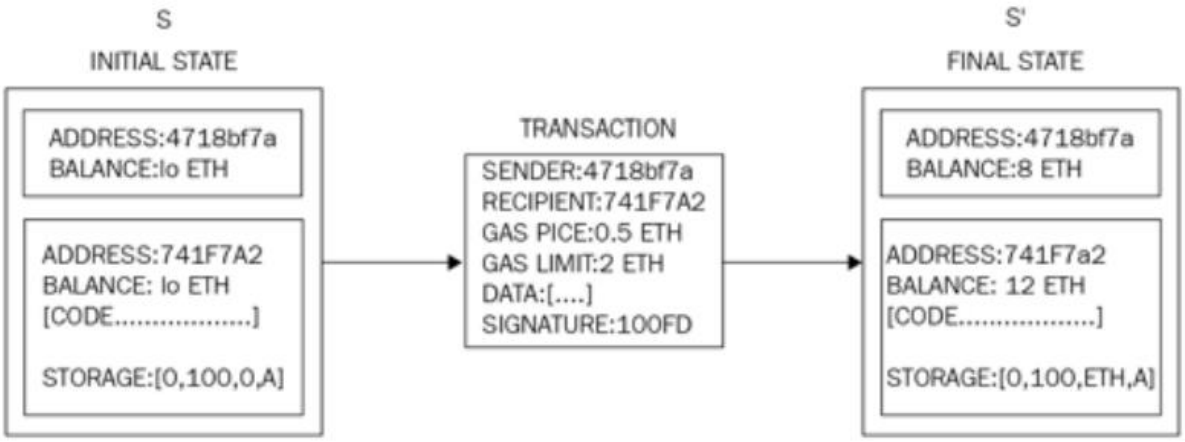
Vitalik Buterin conceptualized Ethereum in November, 2013.

-The critical idea proposed was the development of a Turing-complete language that allows the development of arbitrary programs (smart contracts) for blockchain and decentralized applications.

-This concept is in contrast to Bitcoin, where the scripting language is limited in nature and allows necessary operations only.

Ethereum blockchain

Ethereumcan be visualized as a transaction-based state machine. The core idea is that in Ethereum blockchain, a genesis state is transformed into a final state by executing transactions incrementally. The final transformation is then accepted as the absolute undisputed version of the state. 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. The state is stored on the Ethereum network as the world state. This is the global state of the Ethereumblockchain.

Ethereum – bird's eye view: How Ethereum works from a user's point of view.

The most common use case of transferring funds. In our use case, from one user (Bashir) to another (Irshad).There are several steps involved in this process:

1. First either a user requests money by sending the request to the sender, or the sender decides to send money to the receiver. The request can be sent by sending the receivers Ethereum address to the sender. For example, there are two users, Bashir and Irshad. If Irshad requests money from

Bashir, then she can send a request to Bashir by using QR code. Once Bashir receives this request he will either scan the QR code or manually type in Irshad'sEthereum address and send Ether to Irshad's address.

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

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 requirements and usage.

Mainnet:Mainnet is the current live network of Ethereum. The current version of mainnet is Byzantium (Metropolis) and its chain ID is 1. Chain ID is used to identify the network.

Testnet:Testnet is also called Ropsten and is the widely used test network for the Ethereumblockchain. This test blockchain is used to test smart contracts and DApps before being deployed to the production live blockchain. Test network allows experimentation and research. The main testnet is called Ropsten. othertestnets include Kovan and Rinkeby

Private net: This is the private network that can be created by generating a new genesis block. This is usually the case in private blockchain distributed ledger networks, where a private group of entities start their blockchain and use it as a permissioned blockchain.

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

Mainnet: The Ethereum mainnet is the production network where actual transactions and contracts are executed and stored on the blockchain. This is the network where Ether (ETH), the native cryptocurrency of the Ethereum network, has real-world value and can be traded on cryptocurrency exchanges.
Testnet: The Ethereum testnet is a test environment that allows developers to test and experiment with their applications and smart contracts before deploying them on the mainnet. The testnet uses test Ether, which has no real-world value, and transactions on the testnet do not affect the mainnet.
Private net: A private net is a custom Ethereum network created by individuals or organizations for private use. A private net is not connected to the main Ethereum network and does not use real Ether. Private nets can be used for testing and experimentation, or for running decentralized applications that are not meant for public use. They provide a secure and isolated environment for testing and development.

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

Ethereum Blockchain: The decentralized, open-source blockchain that serves as the backbone of the Ethereum network.

Ethereum Virtual Machine (EVM): A runtime environment for executing smart contracts on the Ethereum network.

Ether (ETH): The native cryptocurrency of the Ethereum network, used to pay for transactions and computational services.

Solidity: A high-level programming language used to write smart contracts on the Ethereum network.

Web3: A collection of technologies and protocols that enable decentralized applications (dApps) to interact with the Ethereum blockchain.

Decentralized Applications (dApps): Applications that run on the Ethereum network and are powered by smart contracts.

Decentralized Exchanges (DEXs): Platforms that allow for the trading of cryptocurrencies and other digital assets in a decentralized manner.

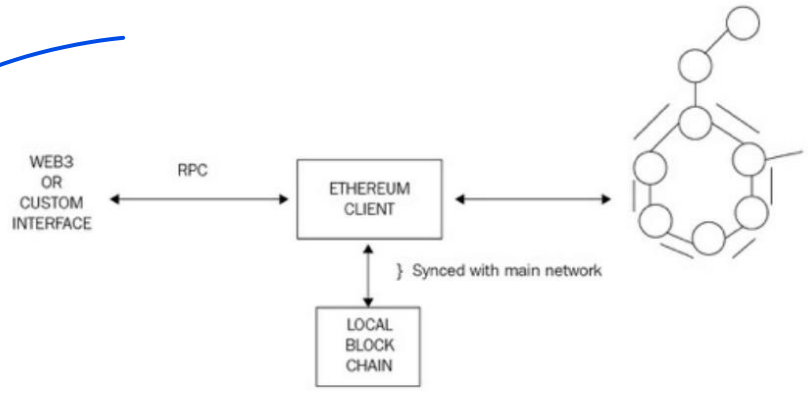
Wallets: Applications that allow users to store, send, and receive Ether and other cryptocurrencies.

Developer Tools: Tools and resources for developers to build and deploy decentralized applications on the Ethereum network.

Mining: The process of validating transactions and adding blocks to the Ethereum blockchain, performed by nodes in the network.

Components of the Ethereum ecosystem

The Ethereumblockchain stack consists of various components. At the core, there is the Ethereumblockchain 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.



The Ethereum stack showing various components

RPCs are the primary way of fetching data in a blockchain system. When new blocks are created and the old ones are updated, an RPC is needed to make sure all nodes (individual copies of the blockchain) have the same information.

A formal list of all high-level elements present in the Ethereum blockchain is presented here:

1. **Keys and addresses**:are used in Ethereumblockchain mainly to represent ownership and transfer of Ether. Keys are used in pairs of private and public type. The private key is generated randomly and is kept secret whereas a public key is derived from the private key. Addresses are derived from the public keys which are a 20-bytes code used to identify accounts.

The process of key generation and address derivation is described here:

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

2. The public key is then derived from this private key using ECDSA recovery function.

3. An address is derived from the public key which is the right most 160 bits of the Keccak hash of the public key

2. **Accounts**: Ethereum, being a transaction driven state machine, the state is created or updated as a result of the interaction between accounts and transaction execution. Operations performed between and on the accounts, represent state transitions.

State storage in the Ethereum blockchain refers to the process of storing information about the current state of the network, including information about smart contracts, transactions, and user accounts. In Ethereum, this state information is stored in a data structure known as the state trie, which is a type of Merkle tree.

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

2. The transaction fee is calculated, and the sending address is resolved using the signature. Furthermore, sender's account balance is checked and subtracted accordingly, and nonce is incremented. An error is returned if the account balance is not enough.

3. Provide enough Ether (gas price) to cover the cost of the transaction. This is charged per byte incrementally proportional to the size of the transaction. In this step, the actual transfer of value occurs. The flow is from the sender's account to receiver's account. The account is created automatically if the destination account specified in the transaction does not exist yet. Moreover, if the destination account is a contract, then the contract code is executed. This also depends on the amount of gas available. If enough gas is available, then the contract code will be executed fully; otherwise, it will run up to the point where it runs out of gas.

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

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

Types of accounts:

Two kinds of accounts exist in Ethereum:

- Externally Owned Accounts (EOAs)
- 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 each type of accounts :

External Account: These are accounts that are controlled by individuals or organizations and are used to interact with the Ethereum network. An external account can send and receive transactions, hold and transfer Ether, and execute smart contracts.

Contract Account: These are accounts that are created and controlled by a smart contract. Contract accounts hold code that can be executed when triggered by a transaction. They can also store and manage data and Ether. Contract accounts are autonomous and operate independently without the need for human intervention. They are often used to create decentralized applications (dapps) and decentralized autonomous organizations (DAOs) on the Ethereum network.

EOs:

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

CAs:

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

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

Both of these transactions are composed of some standard fields,described here.

- **Nonce**: Nonce is a number that is incremented by one every time a transaction is sent by the sender. It must be equal to the number of transactions sent and is used as a unique identifier for the transaction. A nonce value can only be used once. This is used for replay protection on the network.
- **Gas price**: The gas price field represents the amount of Wei required to execute the transaction. In other words, this is the amount of Wei you are willing to pay for this transaction. This is charged per unit of gas for all computation costs incurred as a result of the execution of this transaction
- **Gas limit**: The gas limit field contains the value that represents the maximum amount of gas that can be consumed to execute the transaction.
- **To**: As the name suggests, the to field is a value that represents the address of the recipient of the transaction. This is a 20-byte value.

Value: The " Value" field is used to specify the amount of Ether that is being transferred in the transaction.

Data: The "Data" field contains information about the function call or the data that is being stored in the contract.

Signature: The " Signature" field contains the digital signature of the sender, which is used to verify the authenticity of the transaction.

V, R, S: These are three values used in the Elliptic Curve Digital Signature Algorithm (ECDSA) that are used to verify the authenticity of the transaction and the identity of the sender.

- **Value**: Value represents the total number of Wei to be transferred to the recipient; in the case of a contract account, this represents the balance that the contract will hold.
- **Signature**: 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 ECDSA scheme and makes use of the secp256k1 curve. The theory of Elliptic Curve Cryptography (ECC) was discussed in Chapter 4, Public Key Cryptography. In this section, ECDSA will be presented in the context of its usage in Ethereum.
- **Init**: The Init field is used only in transactions that are intended to create contracts, that is, contract creation transactions. This represents a byte array of unlimited length that specifies the EVM code to be used in the account initialization process. The code contained in this field is executed only once when the account is created for the first time, it (init) gets destroyed immediately after that. Init also returns another code section called body, which persists and runs in response to message calls that the contract account may receive. These message calls may be sent via a transaction or an internal code execution.
- **Data**: If the transaction is a message call, then the data field is used instead of init, which represents the input data of the message call. It is also unlimited in size and is organized as a byte array.

Contract creation transaction:

There are a few essential parameters that are required when creating an account. These parameters are listed as follows:

- Sender
- Original transactor (transaction originator)
- Available gas
- Gas price Endowment, which is the amount of ether allocated
- A byte array of an arbitrary length
- Initialization EVM code
- Current depth of the message call/contract-creation stack (current depth means the number of items that are already there in the stack)

Addresses generated as a result of contract creation transaction are 160-bit in length. Initially, the nonce in the account is set to zero. The balance of the account is set to the value passed to the contract. Storage is also set to empty. Code hash is Keccak 256-bit hash of the empty string. The new account is initialized when the EVM code (the Initialization EVM code, mentioned earlier) is executed.

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

Message call transaction

A message call requires several parameters for execution, which are listed as follows:

- The sender
- The transaction originator
- Recipient

- The account whose code is to be executed (usually same as the recipient)
- Available gas
- Value
- Gas price
- Arbitrary length byte array
- Input data of the call
- Current depth of the message call/contract creation stack

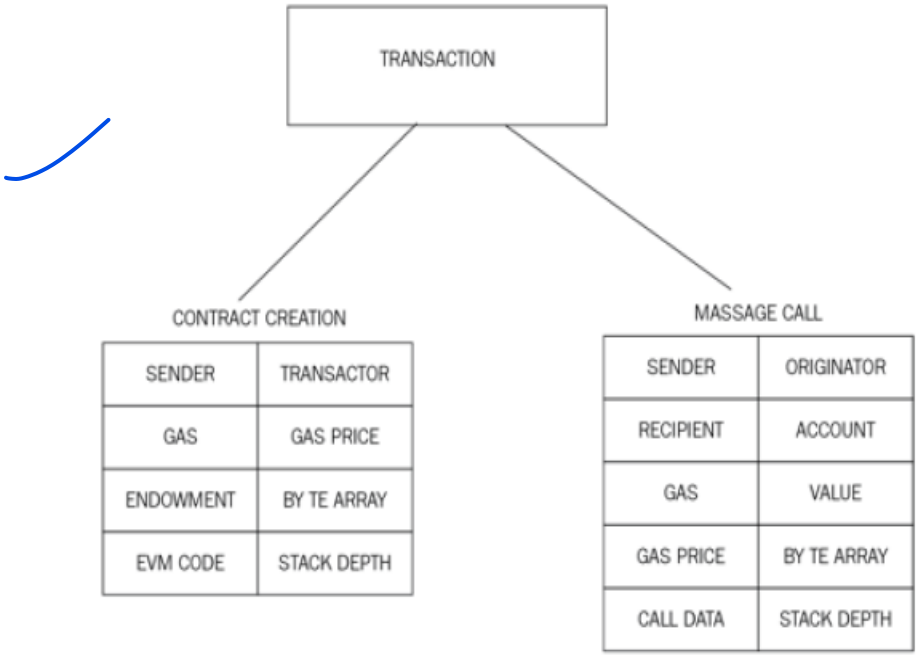
Message calls result in a state transition. The message calls also produce output data, which is not used if transactions are executed. In cases where message calls are triggered by VM code, the output produced by the transaction execution is used. If the message sender is an autonomous object (external actor), then the call passes back any data returned from the EVM operation.

Messages

A message is a data packet passed between two accounts. This data packet contains data and value (amount of ether). It can either be sent via a smart contract (autonomous object) or from an external actor (externally owned account) in the form of a transaction that has been digitally signed by the sender.

Contracts can send messages to other contracts. Messages only exist in the execution environment and are never stored.
A message consists of the components mentioned here:

- The sender of the message
- Recipient of the message
- Amount of Wei to transfer and message to the contract address
- Optional data field (Input data for the contract)
- The maximum amount of gas (startgas) that can be consumed



Types of transactions, required parameters for execution

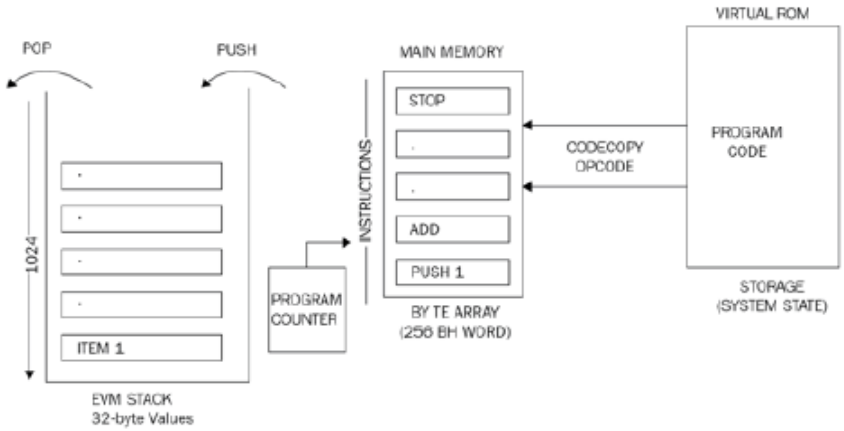
Ether cryptocurrency/tokens (ETC and ETH)

As an incentive to the miners, Ethereum also rewards its own native currency called **Ether**, abbreviated as ETH.There are 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.

Ether is minted by miners as a currency reward for the computational effort they spend to secure the network by verifying and with validation 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.
Fees are charged for each computation performed by the EVM on the blockchain.

The Ethereum Virtual Machine (EVM)

An Ethereum virtual machine is a software platform, or "virtual computer," used by developers to create decentralized applications (DApps), as well as to execute and deploy smart contracts on the Ethereum system.
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 virtual machine is set to 256-bit. The stack size is limited to 1024 elements and is based on the Last In, First Out (LIFO) queue. 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. EVM also supports exception handling, in case exceptions occur, such as not having enough gas or invalid instructions, in which case the machine would immediately halt and return the error to the executing agent.
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 (anyone can run code) to be run on Ethereum blockchain.
EVM is a stack-based architecture. EVM is big-endian by design, and it uses 256-bit wide words. his word size allows for Keccak 256-bit hash and ECC computations. is a stack-based architecture. 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 two types of storage available to contracts and EVM. The first one is called memory, which is a byte array. When a contract finishes the code execution, the memory is cleared. It is akin to the concept of RAM. The other type is called storage which is permanently stored on the blockchain. It is a key value store and can be thought of like a hard disk storage.
Memory is unlimited but constrained by gas fee requirements.



EVM operation

The preceding diagram shows an EVM stack on the left side showing that elements are pushed and popped from the stack. It also shows that a program counter is maintained which is incremented with instructions being read from main memory. Main memory gets the program code from virtual ROM / storage via the CODECOPY instruction.

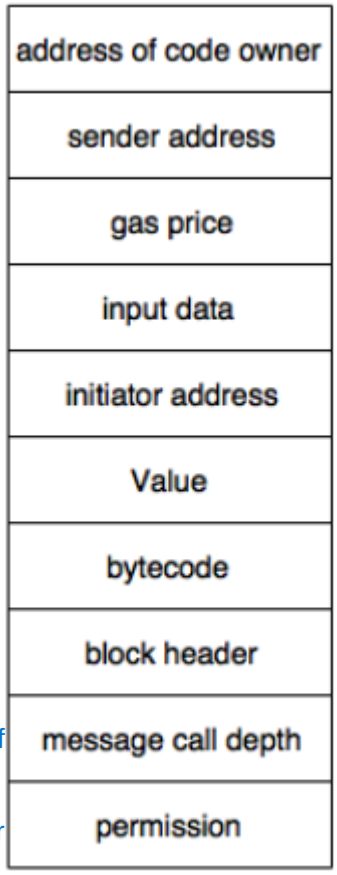
Execution environment

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:

- System state.
- Remaining gas for execution.
- The address of the account that owns the executing code.
- The address of the sender of the transaction.
- 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 currently in execution. In other words, this is the number of CALLs or CREATEs currently in execution.
- Permission to make modifications to the state.

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

A. Machine state: The machine state in Ethereum refers to the current state of all smart contracts and their associated data on the Ethereum blockchain. This includes the current balance of all accounts, the storage of all contracts, and the current values of all variables. The machine state is constantly evolving as transactions are processed and new blocks are added to the blockchain. It is critical to understand the current machine state in order to properly execute smart contracts and process transactions on the Ethereum network.
B. Iterator function: An iterator function is a function that is used to iterate over a data structure in a specific order. In Ethereum, the iterator function is used in combination with smart contracts to allow for the efficient processing of large amounts of data. For example, a smart contract that manages a large database of information could use an iterator function to process the data in a specific order, rather than processing it all at once. This can greatly improve the performance of the contract and allow it to handle larger amounts of data.
C. Smart contracts: Smart contracts are self-executing contracts with the terms of the agreement between buyer and seller being directly written into lines of code. These contracts run on the Ethereum blockchain and are stored on its decentralized ledger, ensuring that they are secure, transparent, and tamper-proof. The code of a smart contract is automatically executed when certain conditions are met, such as when a specified date is reached or when a certain number of funds have been received. This allows for automated, trustless, and secure transactions to take place without the need for intermediaries.



Execution environment tuple

Machine state

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

Machine state is a tuple that consists of the following elements:

- Available gas
- The program counter, which is a positive integer up to 256 memory contents
- Contents of the stack
- Active number of words in memory
- Contents of the stack

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

- Not having enough gas required for execution invalid instructions
- Insufficient stack items
- Invalid destination of jump opcodes
- Invalid stack size (greater than 1024)

The iterator function

The iterator function mentioned earlier 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).

Machine state can be viewed as a tuple shown in the following diagram:

Machine state tuple

Smart Contracts

• Precompiled contracts
• functions that are available natively on the blockchain to support various computationally intensive tasks
• run on the local node and are coded within the Ethereum client

Native Contracts

• The elliptic curve public key recovery function
  – Available at adders 0x1
  – Requires 3000 gas fee for execution
  – Use of public key recovery mechanism for deriving the public key
  – Considers 4 inputs - H, V, R and S
• The SHA-256-bit hash function
  – Available at address 0x2
  – Produces SHA256 hash of the input
  – Gas requirements depend on input data size
  – Output is 32 byte value
• The RIPEMD-160-bit hash function
  – Available at address 0x3
  – provide RIPEMD 160-bit hash
  – Output is a 20 byte value
  – Gas requirements depend on input data size

The identity/datacopy function
  – Available at address 0x4
  – defines output as input
  – The gas requirement is calculated by a simple formula: 15 + 3

[Id/32] where Id is the input data

- Big mod exponentiation function
  - Available at address 0x5
  - Follows RSA signature verification
  - implements a native big integer exponentialmodular operation
- Elliptic Curve (EC) point addition function
  - Available at address 0x6
  - Implements EC point addition function
- Elliptic Curve (EC) scalar multiplication
  - Available at address 0x7
- Elliptic Curve (EC) pairing
  - Available at address 0x8