CHAPTER 7
ETHEREUM 101

From : Mastering Blockchain

By – Imran Bashir

www.packet.com

Introduction

- Ethereum was conceptualized by *Vitalik Buterin* in November 2013.
- The key 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 is in contrast to bitcoin, where the scripting language is very limited and allows basic and necessary operations only.

Ethereum clients

- Various Ethereum clients have been developed using different languages.
- Most popular are go-Ethereum and parity.
- go-Ethereum was developed using Golang, whereas parity was built using Rust.
- There are other clients available too, but usually, the go-Ethereum client known as *geth* is sufficient for all purposes.
- *Mist* is a user-friendly **Graphical User Interface** (**GUI**) wallet that runs geth in the background to sync with the network.

Ethereum Realeses

- The first release of Ethereum was known as *Frontier*, and the other release of Ethereum is called *homestead release*.
- The next version is named *metropolis* and it focuses on protocol simplification and performance improvement.
- The final release(at the time of writing this book) is named *serenity*, which is envisaged to have a Proof of Stake algorithm (Casper) implemented with it.

Ethereum Realeses

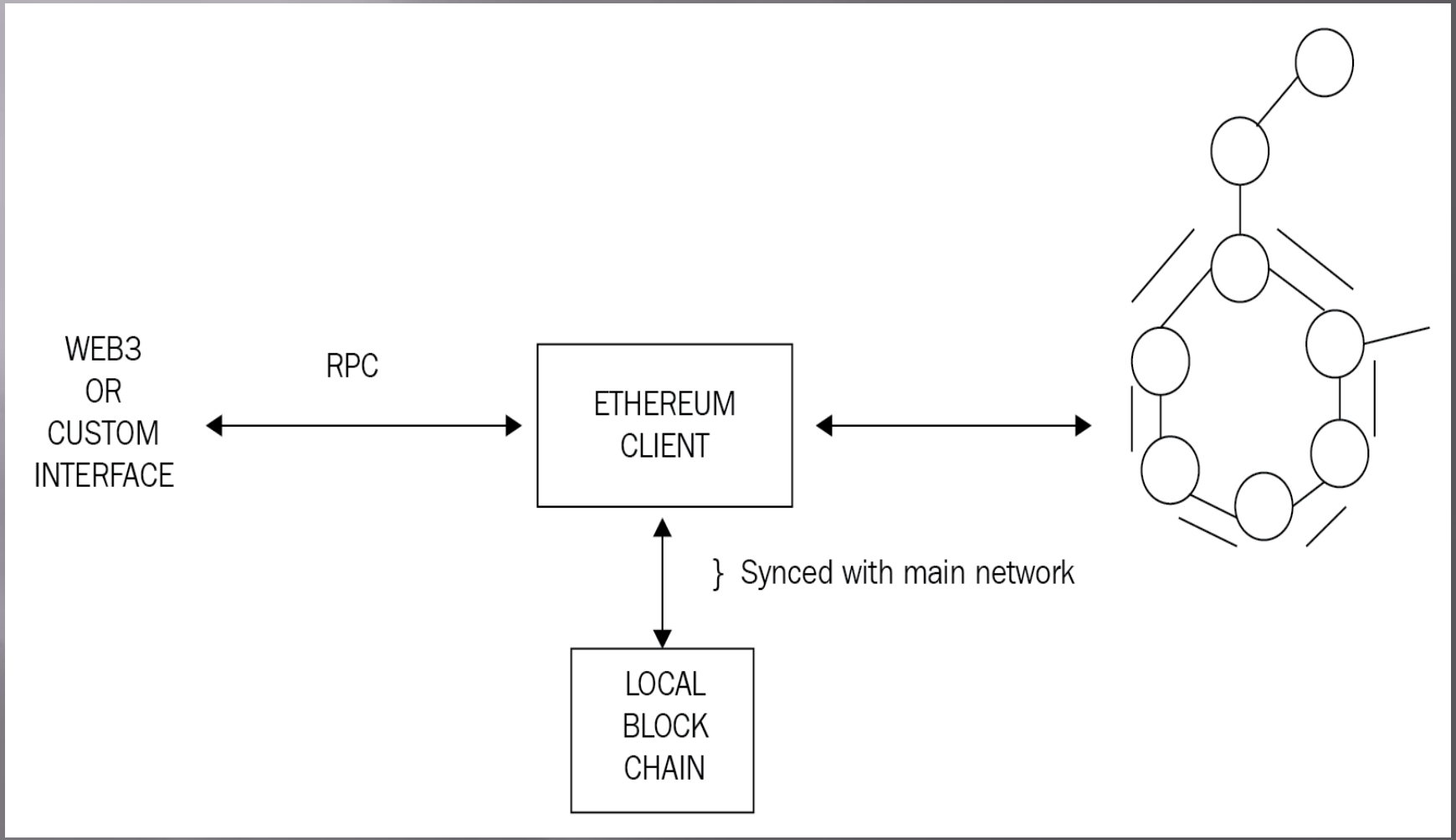
- Further releases are envisaged but have not been named yet.
- The vision of *web 3.0* has already been proposed and is being discussed in the community.
- Web 3.0 is a concept that basically proposes a semantic and intelligent web as an evolution of the existing web 2.0 technology.
- This is the vision of an ecosystem where people, applications,
- data, and web are all connected together and are able to interact with each other in an intelligent fashion.
- With the advent of the blockchain technology, an idea of decentralized web has also emerged, which in fact was the original vision of the Internet.
- The core idea is that all major services, such as DNS, search engines, and identity on the Internet will be decentralized in web 3.0.
- This is where Ethereum is being envisaged as a platform that can help realize this vision.

Ethereum stack

The Ethereum stack consists of various components.

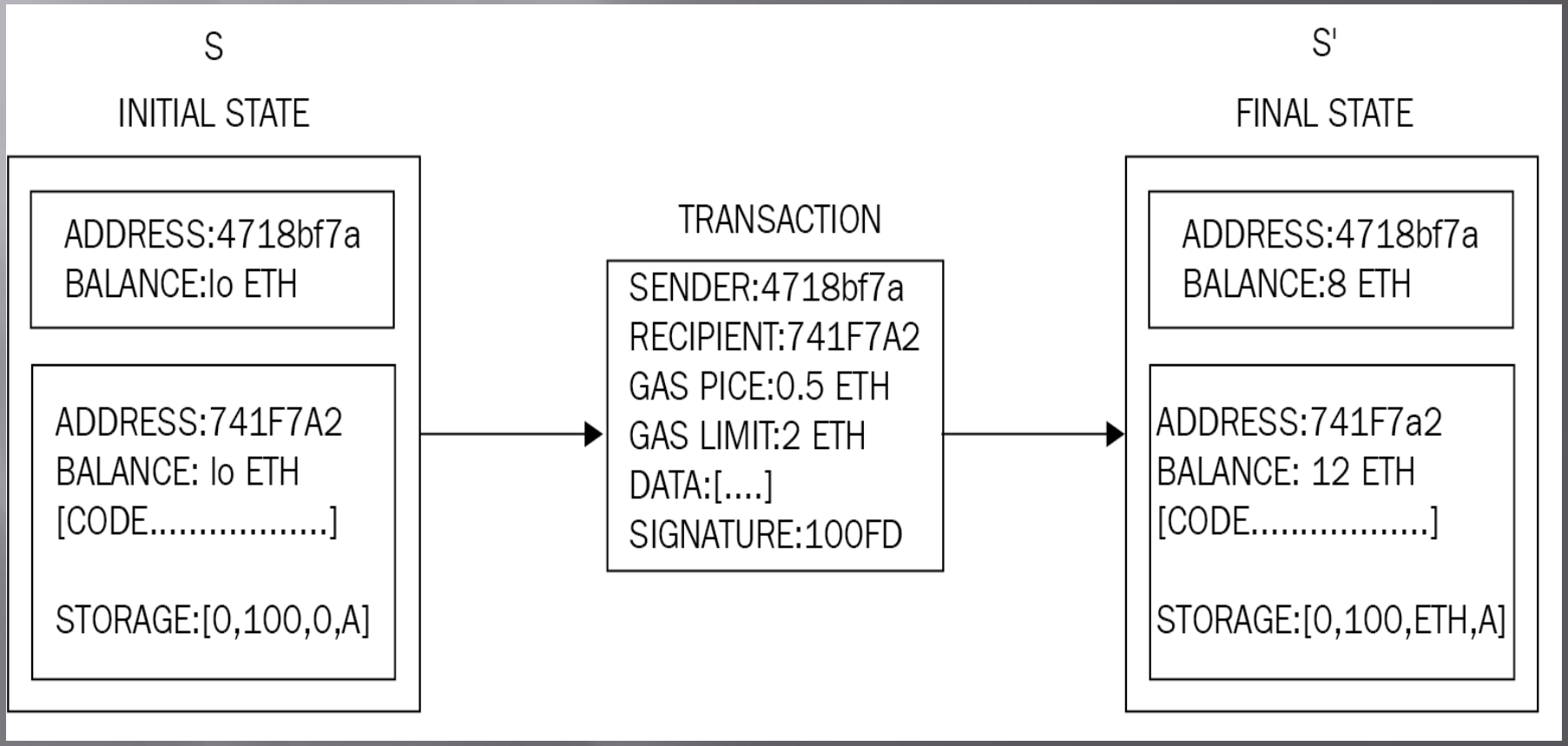
- At the core, there is the Ethereum blockchain running on the P2P Ethereum network.
- Secondly, there's an Ethereum client (usually geth) that runs on the nodes and connects to the peer-to-peer Ethereum network from where blockchain is downloaded and stored locally. It provides various functions, such 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.

Ethereum stack



Ethereum blockchain

Ethereum state machine transition function



Currency (ETH and ETC)

- As an incentive to the miners, Ethereum also rewards its native currency called Ether, abbreviated as ETH.
- After the DAO hack (described earlier), 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 developingn ETC, which is the nonforked original version of Ethereum.
- Here we will be focussing on the ETH

Forks in Ethereum - History

- With the latest release of homestead, due to major protocol upgrades, it 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 of Ethereum called homestead.
- An unintentional fork that occurred on November 24, 2016, at 14:12:07 UTC was due to a bug in the geth client's journaling mechanism. Network fork occurred at block number 2,686,351. This bug resulted in geth failing to revert empty account deletions in the case of the empty out-of-gas exception.
- This means that from block number 2686351, the Ethereum blockchain is split into two, one running with parity clients and the other with geth.
- This issue was resolved with the release of geth version 1.5.3.

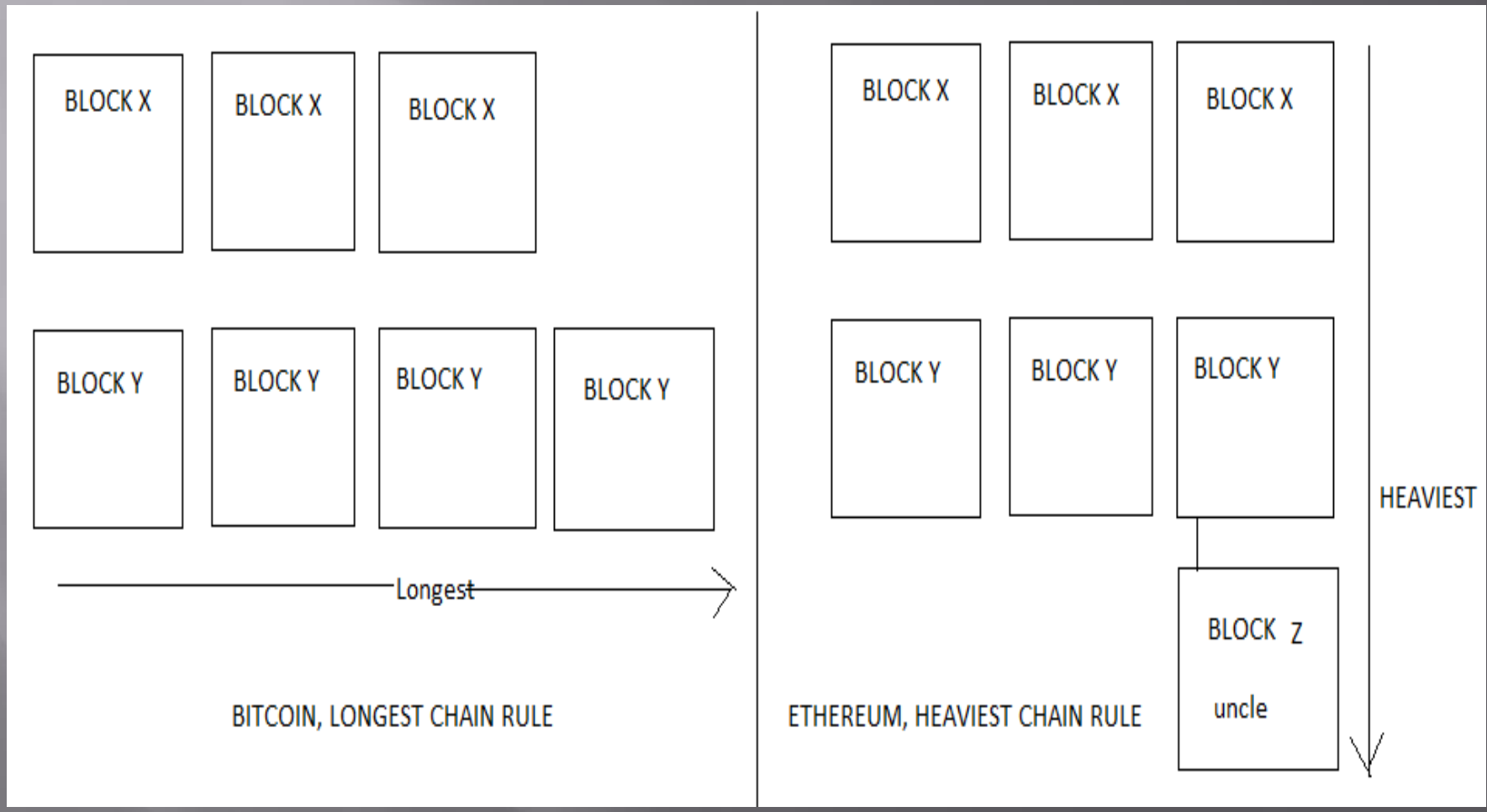
Gas

- Another key concept in Ethereum is that of gas. All transactions on the Ethereum blockchain are required to cover the cost of computation they are performing.
- The cost is covered by something called *gas* or *crypto fuel*, which is a new concept introduced by Ethereum.
- This gas as *execution fee* is paid upfront by the transaction originators.
- The *fuel* is consumed with each operation. Each operation has a predefined amount of gas associated with it.
- Each transaction specifies the amount of gas it is willing to consume for its execution.
- If it runs *out of gas* before the execution is completed, any operation performed by the transaction up to that point is rolled back. Otherwise; on successful transaction, any remaining gas is refunded to the transaction originator.

Consensus Mechanism

- Ethereum uses the simplest version of the GHOST protocol, originally proposed by *Zohar* and *Sompolinsky* in December 2013.
- **Greedy Heaviest Observed Subtree** (**GHOST**) was first introduced as a mechanism to alleviate the issues arising out of fast block generation times that led to stale or orphan blocks.
- In GHOST, stale blocks are added in calculations to figure out the longest and heaviest chain of blocks.
- Stale blocks are called Uncles or Ommers in Ethereum.

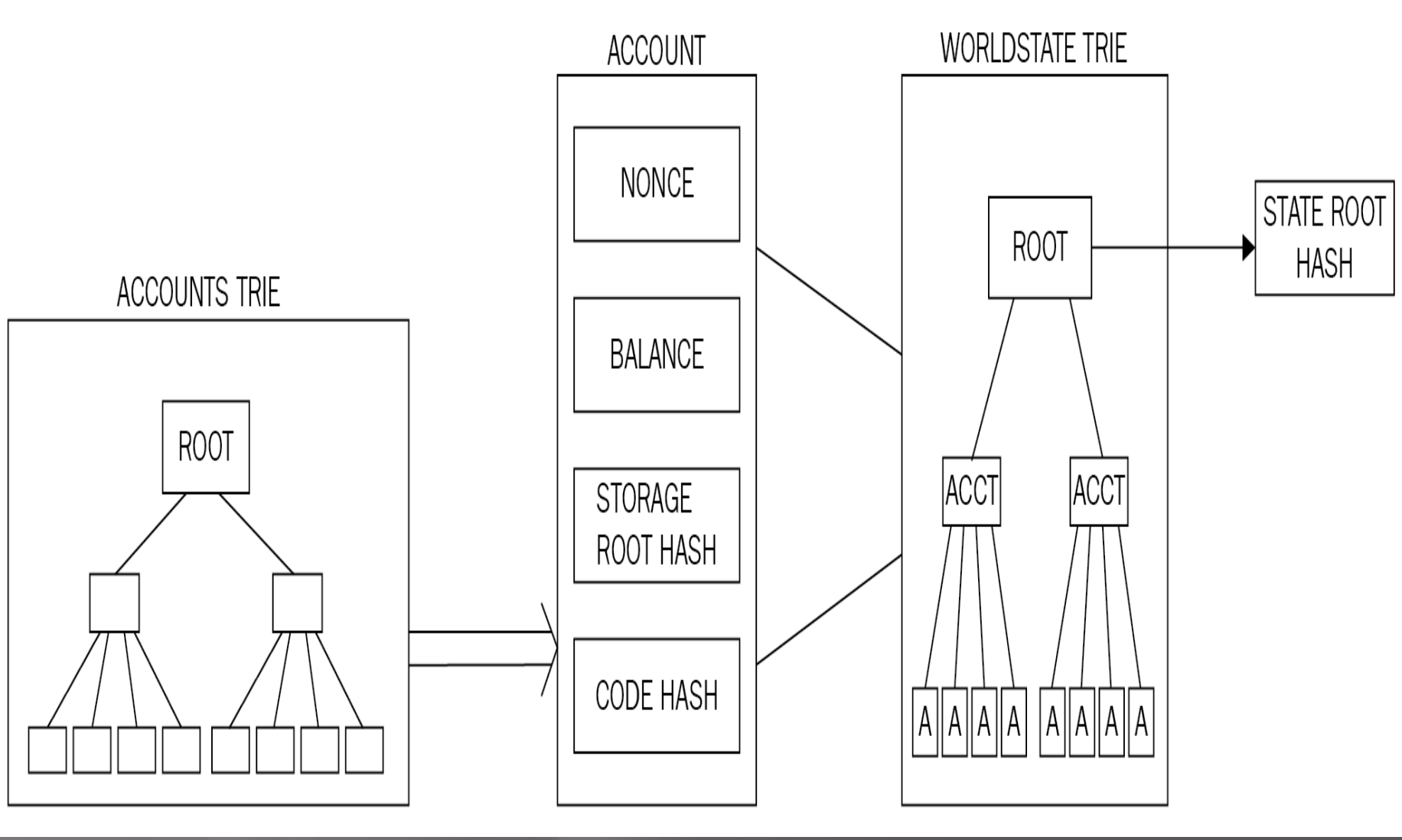
Longest Vs. Heaviest Chain



World State

- The world state in Ethereum represents the global state of the Ethereum blockchain.
- It is basically a mapping between Ethereum addresses and account states.
- RLP is a specially developed encoding scheme that is used in Ethereum to serialize binary data for storage or transmission over the network and also to save the state in a Patricia tree.
- The RLP function takes an item as an input, which can be a string or a list of items, and produces raw bytes that are suitable for storage and transmission over the network.
- RLP does not encode data; instead, its main purpose is to encode structures.

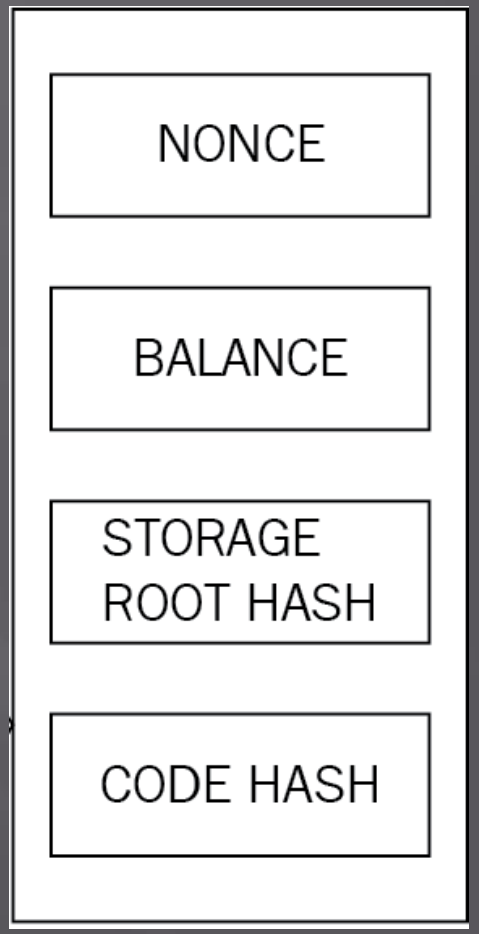
Account State



The account state

The account state consists of four fields:

- Nonce - This is a value that is incremented every time a transaction is sent from the address.
- Balance - This value represents the number of Weis which is the smallest unit of the currency (Ether) in Ethereum held by the address.
- Storageroot - This field represents the root node of a Merkle Patricia tree that encodes the storage contents of the account.
- Codehash - This is an immutable field that contains the hash of the smart contract code that is associated with the account.



Transactions

- 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 account to another.
  - ***Contract creation transactions***: As the name suggests, these transactions result in the creation of a new contract. This means that when this transaction is executed successfully, it creates an account with the associated code.

Transaction Structure

- **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.
- **gasPrice -** The gasPrice field represents the amount of Wei required in order to execute the transaction.
- **gasLimit -** The gasLimit field contains the value that represents the maximum amount of gas that can be consumed in order to execute the transaction. It is the amount of fee in Ether that a user is willing to pay for computation.

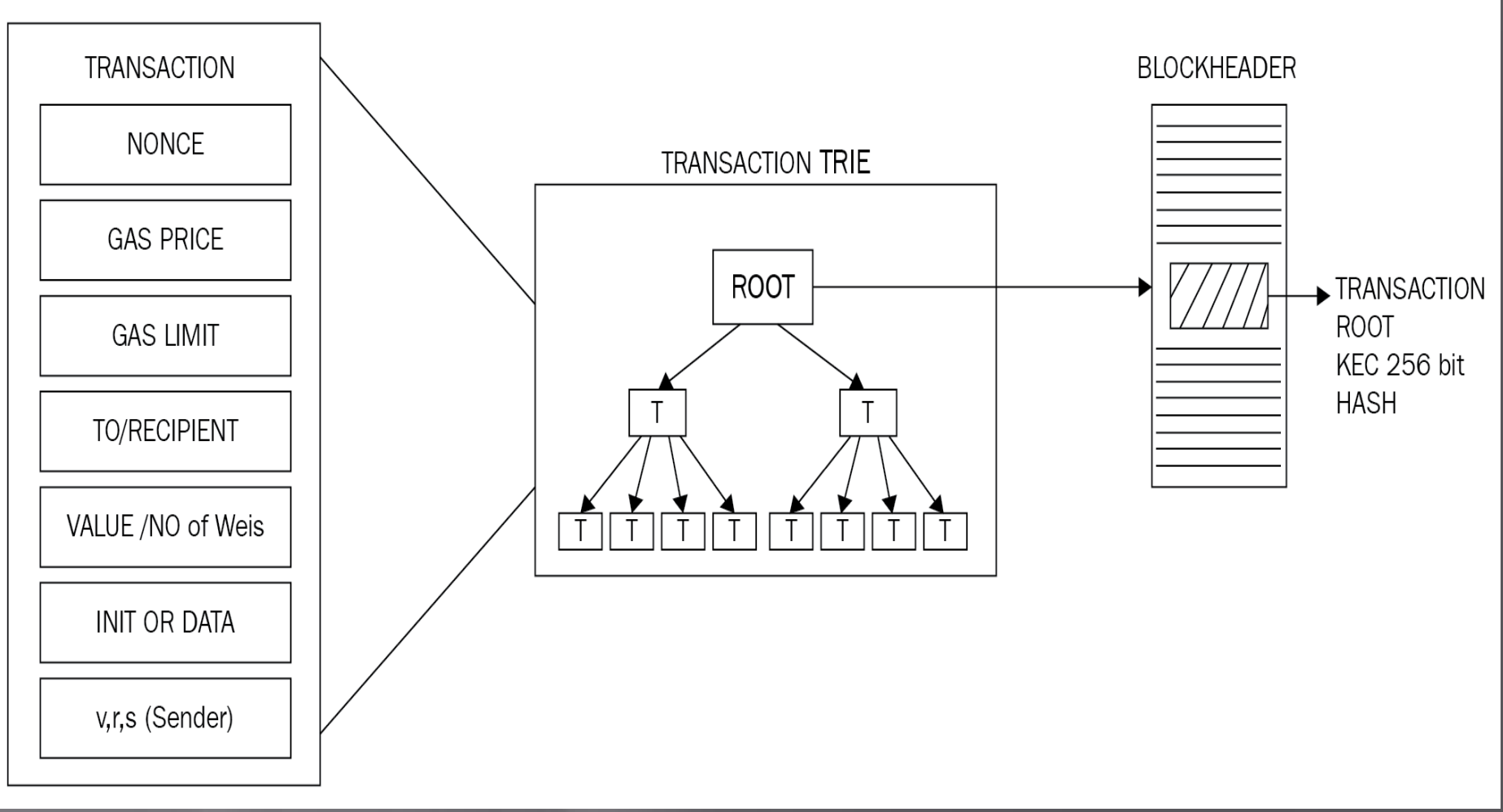
Transaction Structure

- **To -** As the name suggests, the to field is a value that represents the address of the recipient of the transaction.
- **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 –** The signature is based on ECDSA scheme and makes use of the SECP256k1 curve.

Transaction Structure

- **Init -** The Init field is used only in transactions that are intended to create contracts. This represents a byte array of unlimited length that specifies the EVM code to be used in the account initialization process.
  - *Ccde* - The code contained in this field is executed only once, when the account is created for the first time, and gets destroyed immediately after that.
  - *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.

Relationship between transaction, transaction trie and block header



Contract creation transaction

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

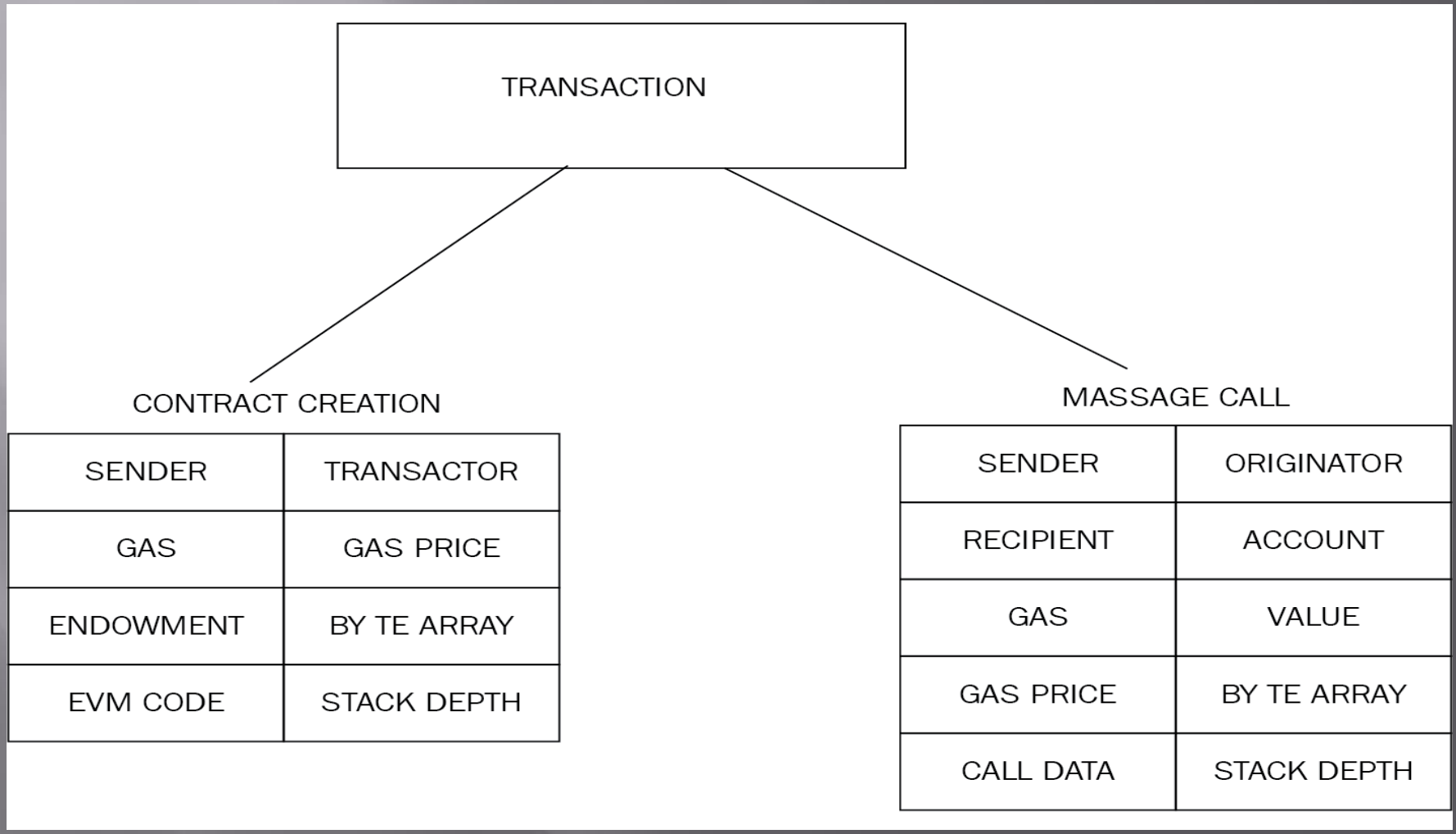
- Sender
- Original transactor
- Available gas
- Gas price
- Endowment, which is the amount of ether allocated initially
- A byte array of arbitrary length
- Initialization EVM code
- Current depth of the message call/contract-creation stack (current depth means the number of items that are already there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

Message call transaction

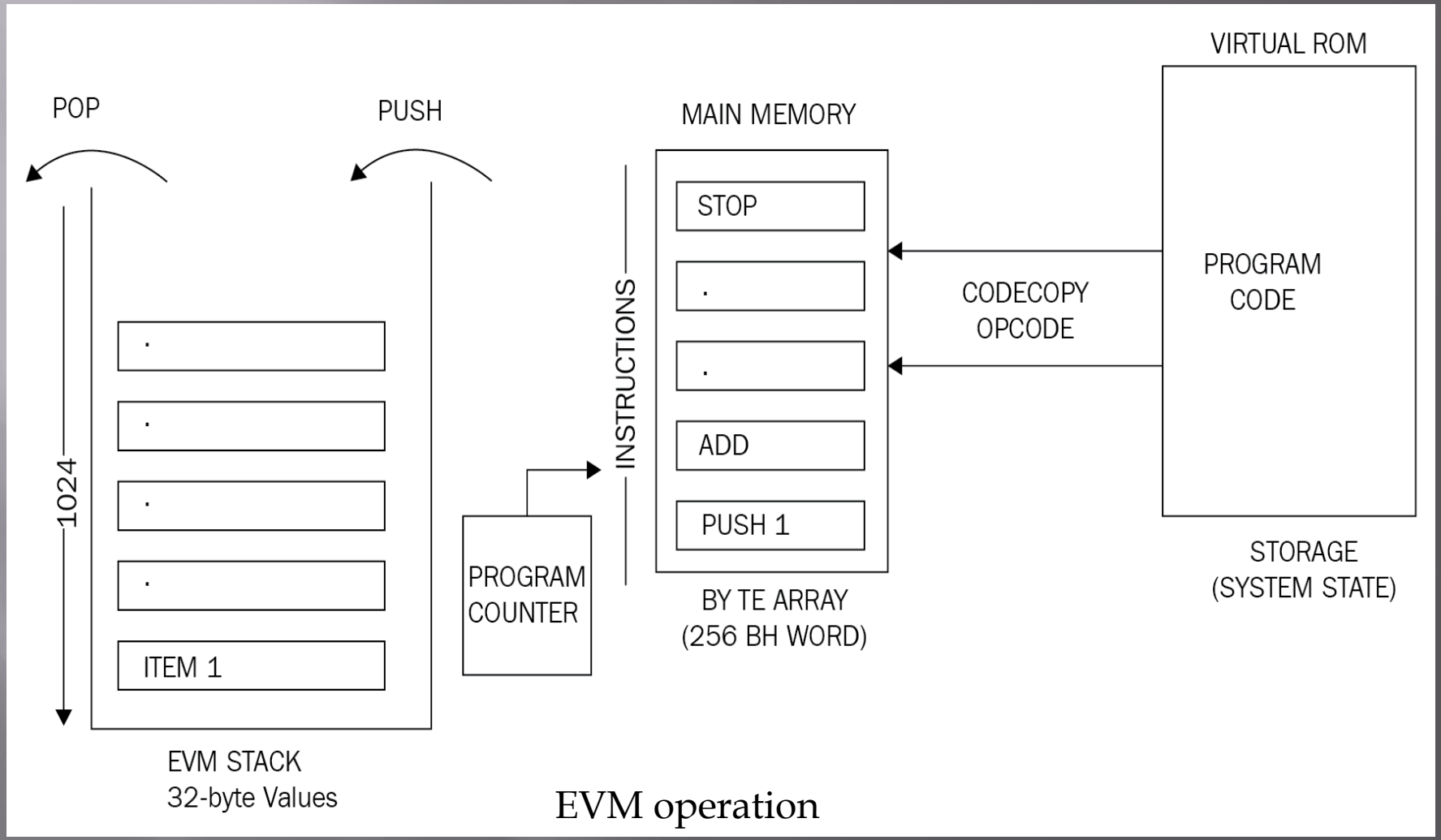
A message call requires several parameters for execution, which are listed as follows:

- Sender
- The transaction originator
- Recipient
- The account whose code is to be executed
- Available gas
- Value
- Gas price
- Arbitrary length byte array
- Input data of the call
- Current depth of the message call/contract creation stack

Types of transactions, required parameters for execution



Ethereum virtual machine (EVM)



EVM operation

EVM Architecture

- EVM 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 elliptic curve cryptography computations.

EVM Storage

- There are two types of storage available to contracts and EVM.
  1. The first one is called memory, which is a byte array. When a contract finishes the code execution, the memory is cleared. Similar to RAM.
  2. The other type, is permanent storage on the blockchain. It is a key value store.

Memory is unlimited but constrained by gas fee requirements

EVM Storage(cont…)

- The storage associated with the virtual machine is a word addressable *word array* that is non-volatile and is maintained as part of the system state.
- Keys and value are 32 bytes in size and storage.
- The program code is stored in a **virtual read-only memory** (**virtual ROM**) that is accessible using the CODECOPY instruction.
- The CODECOPY instruction is used to copy the program code into the main memory.
- Initially, all storage and memory is set to zero in the EVM.

Execution Environment

- There are some key elements that are required by the execution environment in order to execute the code.
- The key parameters are provided by the execution agent, for example, a transaction.

Execution Environment – Key parameters

1. The address of the account that owns the executing code.
2. The address of the sender of the transaction and the originating address of this execution.
3. The gas price in the transaction that initiated the execution.
4. 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.

Execution Environment – Key parameters

5. 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's the address of the account.

6. The value or transaction value. This is the amount in Wei. If the execution agent is a transaction, then it is the transaction value.

7. The code to be executed presented as a byte array that the iterator function picks up in each execution cycle.

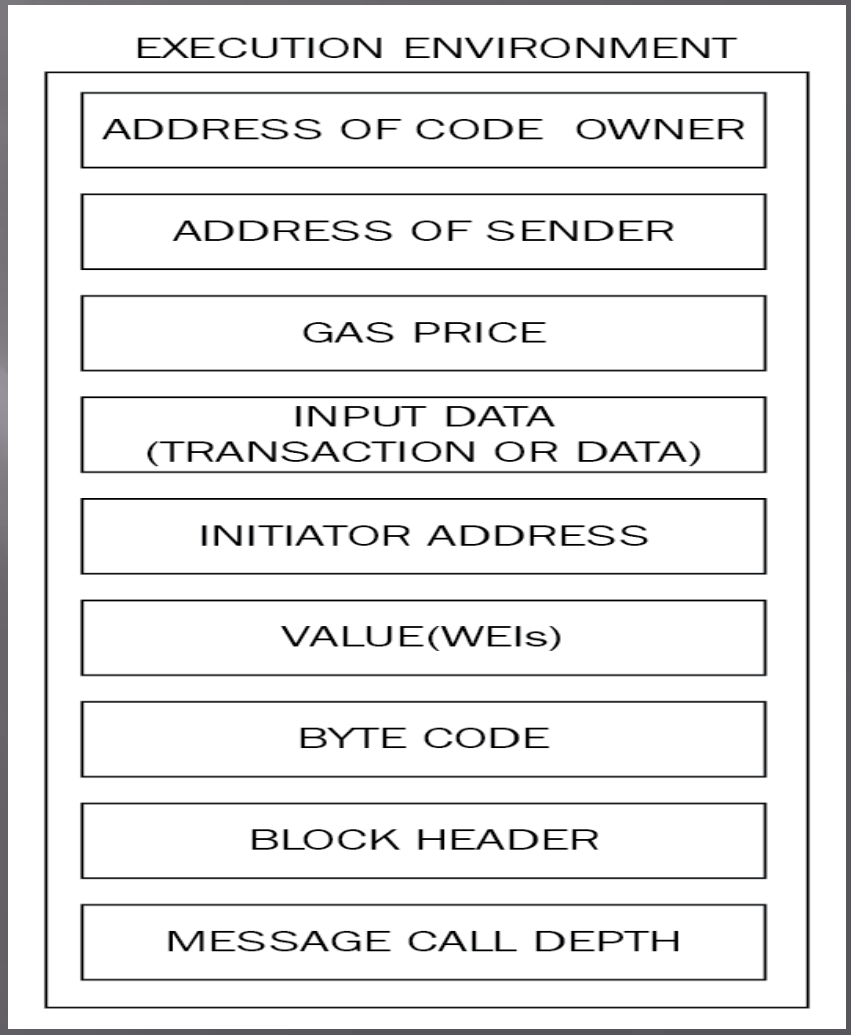
Execution Environment – Key parameters

8. The block header of the current block

9. 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.

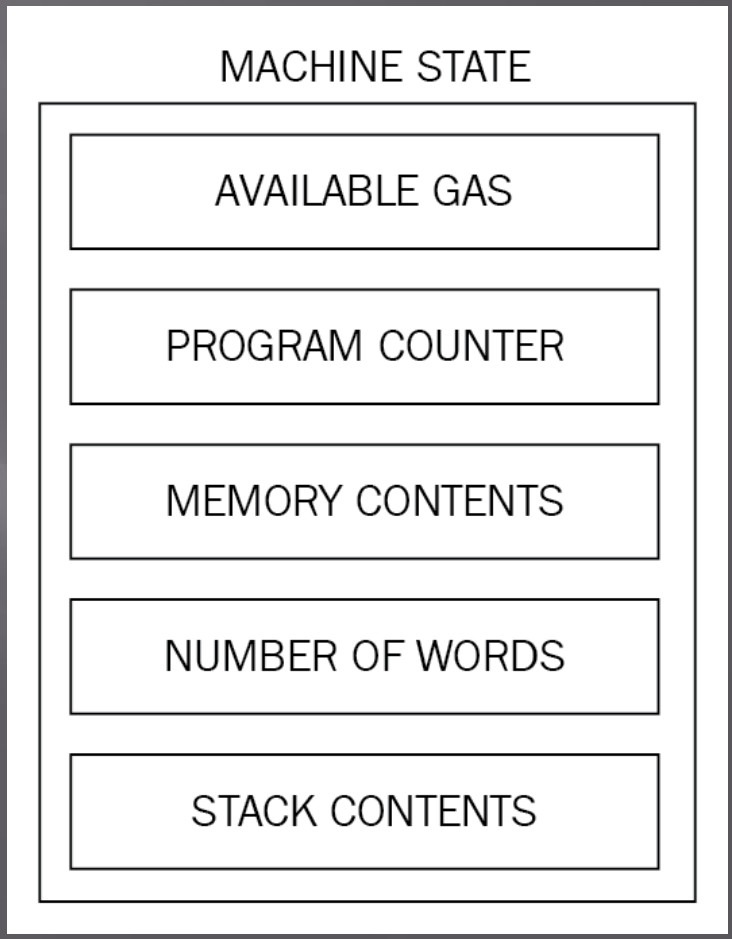
Execution Environment- 9 tuples

- In addition to these nine fields, system state and the remaining gas are also provided to the execution environment.
- The execution results in producing the resulting state, gas remaining after the execution, self-destruct or suicide set, log series, and any gas refunds.



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 consist of the following elements:
  1. Available gas
  2. The program counter, which is a positive integer up to 256
  3. Memory contents
  4. Active number of words in memory
  5. Contents of the stack



Exceptions in EVM

1. Not having enough gas required for execution
2. Invalid instructions
3. Insufficient stack items
4. Invalid destination of jump op codes
5. Invalid stack size (greater than 1024)

The virtual machine is also able to halt in normal conditions if STOP or SUICIDE or RETURN Opcodes are encountered during the execution cycle.

The iterator function

- The iterator function mentioned earlier performs various important 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**).

EVM Byte code

- Code written in a high-level language such as serpent, LLL, or Solidity is converted into the byte code that EVM understands in order for it to be executed by the EVM.

Ethereum Languages

- Solidity is the high-level language that has been developed for Ethereum with JavaScript such as syntax to write code for smart contracts. Once the code is written, it is compiled into byte code that's understandable by the EVM using the Solidity compiler called solc.
- LLL (Lisp-like Low-level language) is another language that is used to write smart contract code.
- Serpent is a Python-like high-level language that can be used to write smart contracts for Ethereum.

Solidity code and Byte code

```
pragma solidity ^0.4.0;
contract Test1
{
    uint x=2;
    function addition1(uint x)
returns (uint y) {
     y=x+2;
    }
}
```

606060405260e060020a6000350463989e17318114601c5
75b6000565b34600057602960043

5603b565b60408051918252519081900360200190f35b6
00281015b91905056

Opcodes PUSH1 0x60 PUSH1 0x40 MSTORE PUSH1 0x2 PUSH1 0x0 SSTORE CALLVALUE

PUSH1 0x0 JUMPI JUMPDEST PUSH1 0x45 DUP1 PUSH1 0x1A PUSH1 0x0 CODECOPY

PUSH1 0x0 RETURN PUSH1 0x60 PUSH1 0x40 MSTORE PUSH1 0xE0 PUSH1 0x2 EXP

PUSH1 0x0 CALLDATALOAD DIV PUSH4 0x989E1731 DUP2 EQ PUSH1 0x1C JUMPI

JUMPDEST PUSH1 0x0 JUMP JUMPDEST CALLVALUE PUSH1 0x0 JUMPI PUSH1 0x29 PUSH1

0x4 CALLDATALOAD PUSH1 0x3B JUMP JUMPDEST PUSH1 0x40 DUP1 MLOAD SWAP2 DUP3

MSTORE MLOAD SWAP1 DUP2 SWAP1 SUB PUSH1 0x20 ADD SWAP1 RETURN JUMPDEST

PUSH1 0x2 DUP2 ADD JUMPDEST SWAP2 SWAP1 POP JUMP

Precompiled contracts

There are four precompiled contracts in Ethereum

1. **The elliptic curve public key recovery function**
2. **The SHA-256 bit hash function**
3. **The RIPEMD-160 bit hash function**
4. **The identity function**

The elliptic curve public key recovery function

- **ECDSARECOVER** (**Elliptic curve DSA recover function**) is available at address 1.
- It is denoted as ECREC and requires 3000 gas for execution.
- If the signature is invalid, then no output is returned by this function.
- Public key recovery is a standard mechanism by which the public key can be derived from the private key in elliptic curve cryptography.
- The ECDSA recovery function is shown as follows:

$$ECDSARECOVER(H, V, R, S) = Public\ Key$$

- It takes four inputs: H, which is a 32 byte hash of the message to be signed and *V*, *R*, and *S*, which represent the ECDSA signature with the recovery ID and produce a 64 byte public key.

The SHA-256 bit hash function

- The SHA-256 bit hash function is a precompiled contract that is available at address 2 and produces a SHA256 hash of the input.
- Gas requirement for SHA-256 (SHA256) depends on the input data size.
- The output is a 32 byte value.

The RIPEMD-160 bit hash function

- The RIPEMD-160 bit hash function is used to provide RIPEMD 160-bit hash and is available at address 3.
- The output of this function is a 20-byte value.
- Gas requirement, similar to SHA-256, is dependent on the amount of input data.

The identity function

- The identity function is available at address 4 and is denoted by the ID.
- Whatever input is given to the ID function, it will output the same value.
- Gas requirement is calculated by a simple formula:

$$15 + 3\,[Id/32]$$

where *Id* is the input data.

- This means that at a high level, the gas requirement is dependent on the size of the input data.

Accounts

- Accounts are one of the main building blocks of the Ethereum blockchain.
- The state is created or updated as a result of the interaction between accounts.
- Operations performed between and on the accounts represent state transitions.
- State transition is achieved using what's called the Ethereum state transition function.

Ethereum state transition function

1. Confirm the transaction validity by checking the syntax, signature validity, and nonce.
2. 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 according to the size of the transaction.

Ethereum state transition function

4. 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.

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

6. Finally, the remainder (if any) of the fee is sent back to the sender as change and fee is paid to the miners accordingly. At this point, the function returns the resulting state.

Types of Accounts

There are two types of accounts in Ethereum:

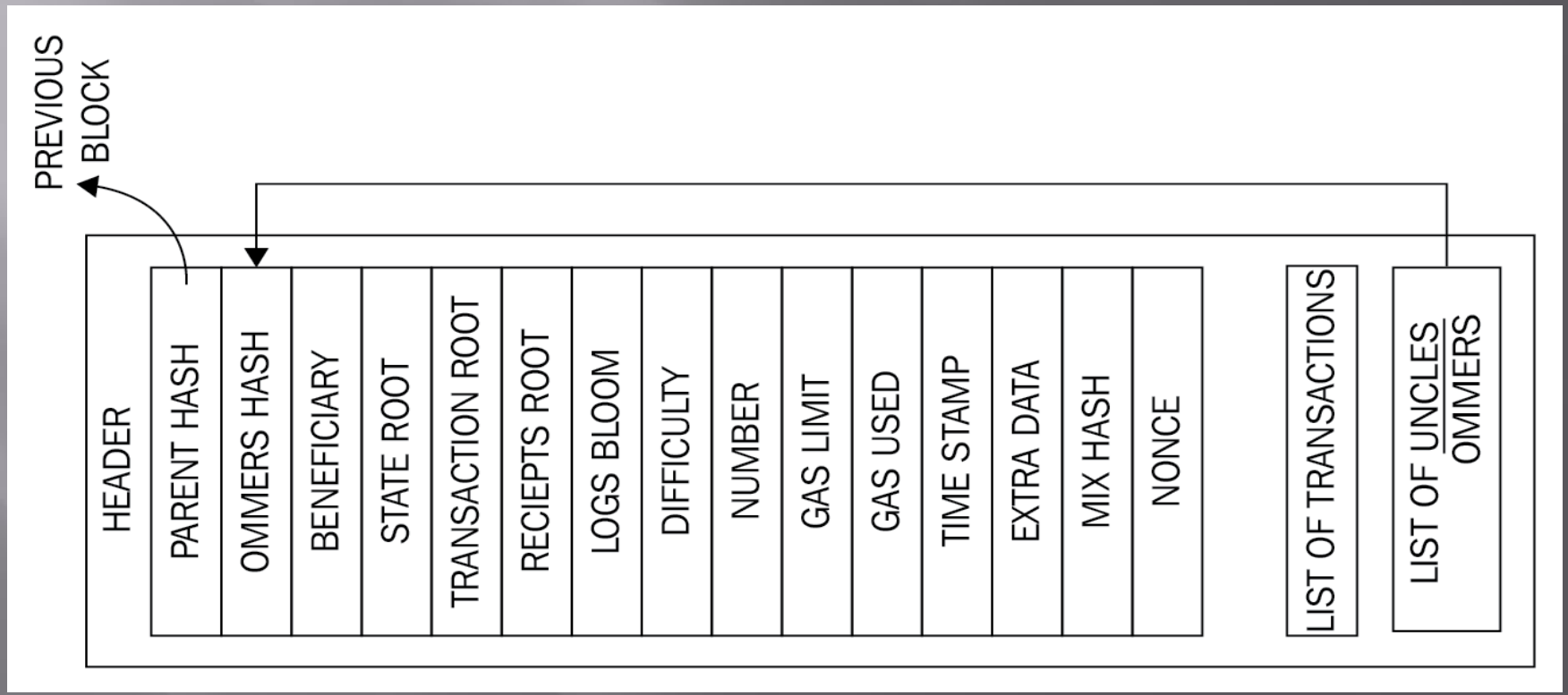
1. **Externally owned accounts** (**EOAs**) – These are similar to accounts that are controlled by a private key in bitcoin. An EOA has ether balance, is able to send transactions, and has no associated code
2. **Contract accounts (CA) -** These are the accounts that have code associated with them along with the private key. Contract Account has ether balance, associated code, and the ability to get triggered and execute code in response to a transaction or a message. Due to the Turing-completeness property of the Ethereum blockchain, the code within contract accounts can be of any level of complexity. The code is executed by EVM by each mining node on the Ethereum network. In addition, contract accounts are able to maintain their own permanent state and can call other contracts.

Block

- Blocks are the main building blocks of a blockchain.
- Ethereum blocks consist of various components, which are described as follows:
  - The block header
  - The transactions list
  - The list of headers of Ommers or Uncles
- The transaction list is simply a list of all transactions included in the block. In addition, the list of headers of Uncles is also included in the block.

Block Header

- The most important and complex part is the block header, which is discussed here.



Fields in Block Header

- **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 (Uncles) blocks included in the block.
- **Beneficiary -** 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 after all transactions have been processed and finalized.

Fields in Block Header

- **Transactions root -** The transaction root is the Keccak 256-bit hash of the root node of the transaction trie. 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. Transaction receipts are generated after each transaction is processed and contain useful posttransaction information.
- **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

Fields in Block Header

- **Difficulty -** The difficulty level of the current block.
- **Number -** The total number of all previous blocks; the genesis block is block zero.
- **Gas limit -** The field contains the value that represents the limit set on the gas consumption per block.
- **Gas used -** The field contains the total gas consumed by the transactions included in the block.
- **Timestamp -** Timestamp is the epoch Unix time of the time of block initialization.
- **Extra data -** Extra data field can be used to store arbitrary data related to the block.
- **Mixhash -** Mixhash field contains a 256-bit hash that once combined with the nonce is used to prove that adequate computational effort has been spent in order to create this block.
- **Nonce -** Nonce is a 64-bit hash (a number) that is used to prove, in combination with the mixhash field, that adequate computational effort has been spent in order to create this block.

The genesis block

Element	Description
Timestamp	(Jul-30-2015 03:26:13 PM +UTC)
Transactions	8893 transactions and 0 contract internal transactions in this block
Hash	0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3
Parent hash	0x0000000000000000000000000000000000000000000000000000000000000000
Sha3Uncles	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347
Mined by	0x0000000000000000000000000000000000000000 IN 15 secs
Difficulty	17,179,869,184
Total Difficulty	17,179,869,184
Size	540 bytes
Gas Limit	5,000
Gas Used	0
Nonce	0x0000000000000042
Block Reward	5 Ether
Uncles Reward	0
Extra Data	»èÛN4{NŒ”|ƒpäµí3³ÛiËÛz8áå ,ú (Hex:0x11bbe8db4e347b4e8c937c1c8370e4b5ed33adb3db69cbdb7a38e1e50b1b82fa)

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. Hash (Keccak 256-bit) of the root of this trie is placed in the block header as the receipts root.

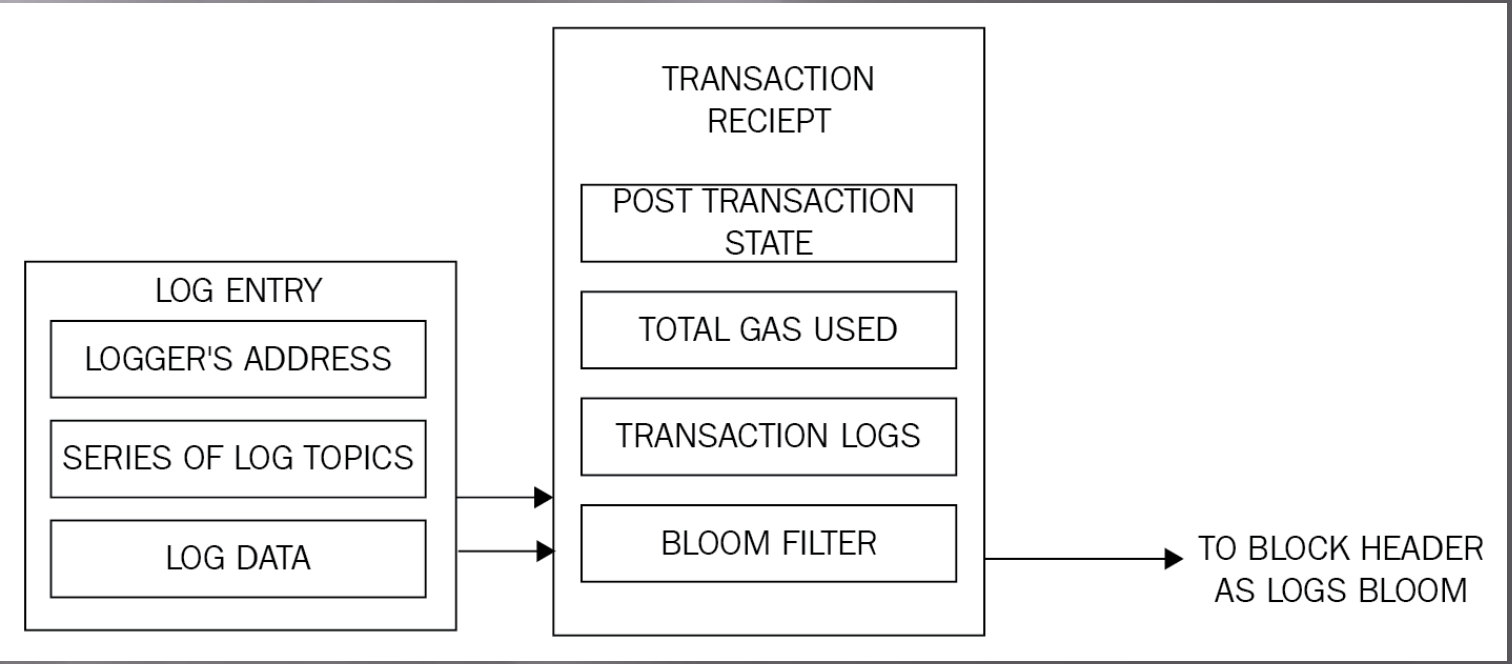
Transaction receipts

It is composed of four elements -

1. **The post-transaction state -** This item is a trie structure that holds the state after the transaction has executed. It is encoded as a byte array.
2. **Gas used -** This item represents the total amount of gas used in the block that contains the transaction receipt. The value is taken immediately after the transaction execution is completed. The total gas used is expected to be a non-negative integer.
3. **Set of logs -** This field shows the set of log entries created as a result of transaction execution. Log entries contain the logger's address, a series of log topics, and the log data.

Transaction receipts And Logs Bloom

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



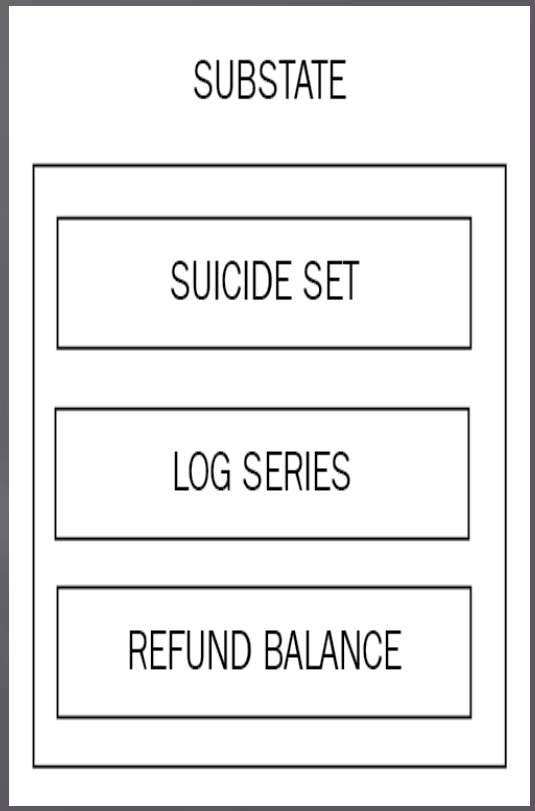
Transaction validation and execution

Transactions are executed after verifying the transactions for validity. Initial tests are listed as follows:

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

Transaction sub state

- **Transaction sub state**
  - A transaction sub-state is created during the execution of the transaction that is processed immediately after the execution completes.
  - This transaction sub-state is a tuple that is composed of three items.
    1. **Suicide Set**
    2. **Log Series**
    3. **Refund Balance**



Transaction sub state-components

- **Suicide Set**
  - This element contains the list of accounts that are disposed of after the transaction is executed.
- **Log series**
  - This is an indexed series of checkpoints that allow the monitoring and notification of contract calls to the entities external to the Ethereum environment, such as application frontends.
  - It works like a trigger mechanism that is executed every time a specific function is invoked or a specific event occurs.
  - Logs are created in response to events occurring in the smart contract.
  - It can also be used as a cheaper form of storage.
- **Refund balance**
  - This is the total price of gas in the transaction that initiated the execution.
  - Refunds are not immediately executed; instead, they are used to partially offset the total execution cost.

The block validation mechanism

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

- Consistent with Uncles and transactions. This means that all Ommers (Uncles) satisfy the property that they are indeed Uncles and also if the Proof of Work for Uncles is valid.
- If the previous block (parent) exists and is valid.
- If the timestamp of the block is valid. This basically 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.

Block finalization

- Block finalization is a process that is run by miners in order to validate the contents of the block and apply rewards.
- It results in four steps being executed.
  1. **Ommers validation**
  2. **Transaction validation**
  3. **Reward application**
  4. **State and nonce validation**

Block finalization Steps

- **Ommers validation**
  - In the case of mining, determine Ommers.
  - The validation process of the headers of stale blocks checks whether the header is valid and the relationship of the *Uncle with the current block satisfies the maximum depth of six blocks.*
  - A block can contain a maximum of two Uncles.
- **Transaction validation**
  - In the case of mining, determine transactions.
  - The process involves checking whether the total gas used in the block is equal to the final gas consumption after the final transaction.
- **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.
  - Uncles that are included in the blocks also receive 7/8 of the total block reward.
- **State and nonce validation**
  - Verify the state and nonce. In the case of mining, compute a valid state and nonce.

Block difficulty

- Block difficulty is increased if the time between two blocks decreases, whereas it increases if the block 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 shown as follows:

  *block_diff = parent_diff + parent_diff // 2048 **

  max(1 - (block_timestamp - parent_timestamp) // 10, -99) +

  *int(2**((block.number // 100000) - 2))*

- The preceding algorithm means that, if the time difference between the generation of the parent block and the current block is less than 10 seconds, the difficulty goes up.
- If the time difference is between 10 to 19 seconds, the difficulty level remains the same.
- Finally, if the time difference is 20 seconds or more, the difficultly level decreases. This decrease is proportional to the time difference.

Block difficulty

- 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 the so called *difficulty time bomb* or *Ice age* introduced in the Ethereum network, 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 as mining on the POW chain will eventually become prohibitively difficult.
- According to the latest update and estimates based on the algorithm, the block generation time will become significantly high during the second half of the year 2017 and in 2021, it will become so high that it will be virtually impossible to mine on the POW chain.
- This way, miners will have no choice but to switch to the Proof of Stake scheme proposed by Ethereum called Casper.

Ether

- Ether is minted by miners as a currency reward for the computational effort they spend in
- order 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 in order to perform
- computation on the Ethereum blockchain.

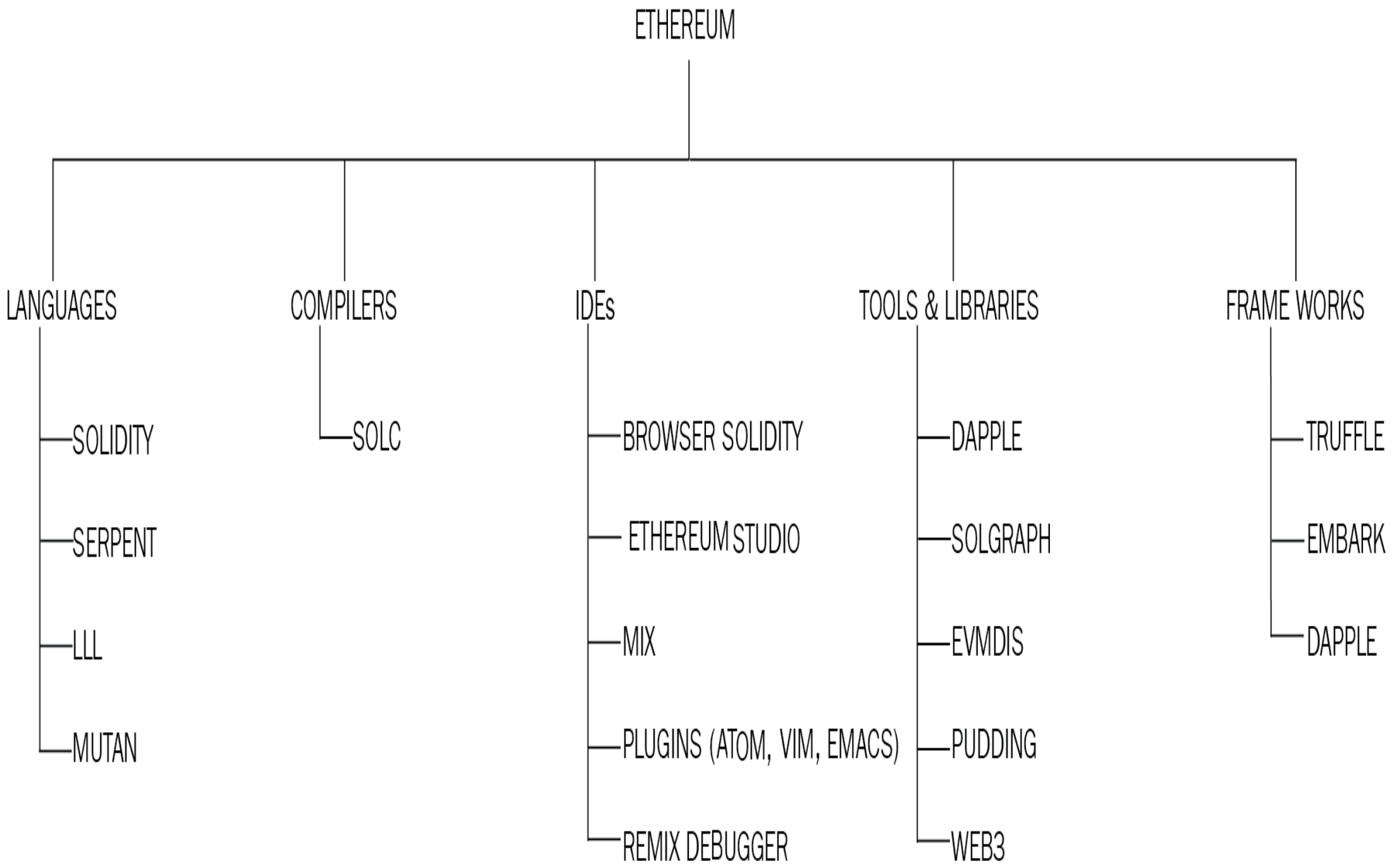
Ether denomination

Unit	Wei Value	Weis
Wei	1 Wei	1
Babbage	1e3 Wei	1,000
Lovelace	1e6 Wei	1,000,000
Shannon	1e9 Wei	1,000,000,000
Szabo	1e12 Wei	1,000,000,000,000
Finney	1e15 Wei	1,000,000,000,000,000
Ether	1e18 Wei	1,000,000,000,000,000,000

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 some 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 cost can be estimated using the following formula:

*Total cost = gasUsed * gasPrice*


ETHEREUM
LANGUAGES
SOLIDITY
SERPENT
LLL
MUTAN
COMPILERS
SOLC
IDEs
BROWSER SOLIDITY
ETHEREUM STUDIO
MIX
PLUGINS (ATOM, VIM, EMACS)
REMIX DEBUGGER
TOOLS & LIBRARIES
DAPPLE
SOLGRAPH
EVMDIS
PUDDING
WEB3
FRAME WORKS
TRUFFLE
EMBARK
DAPPLE