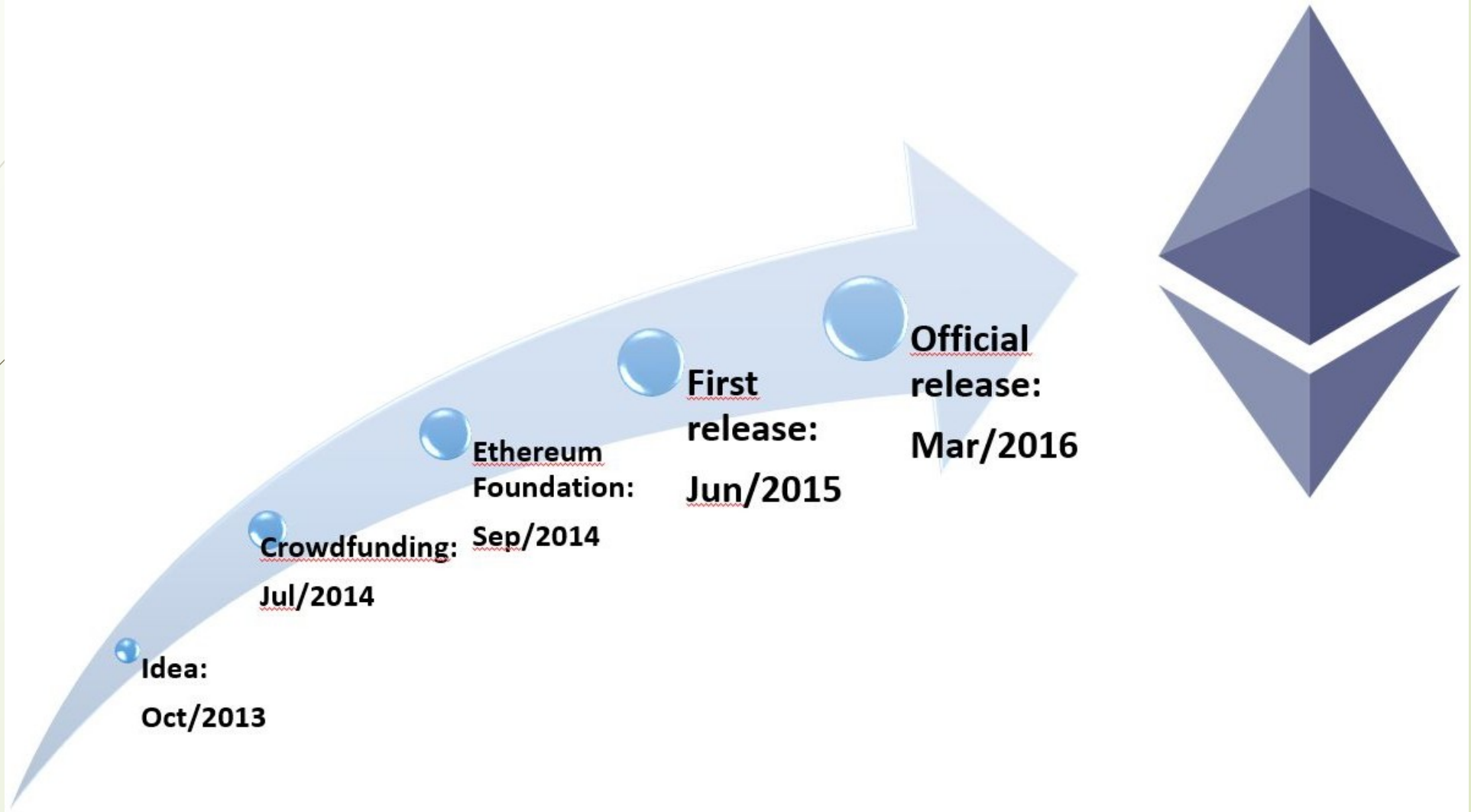
Ethereum 101

Introduction

- Ethereum was conceptualized by *Vitalik Buterin* in November 2013.
- key idea proposed was the development of a Turing-complete language
  - 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.

History of Ethereum - Timeline



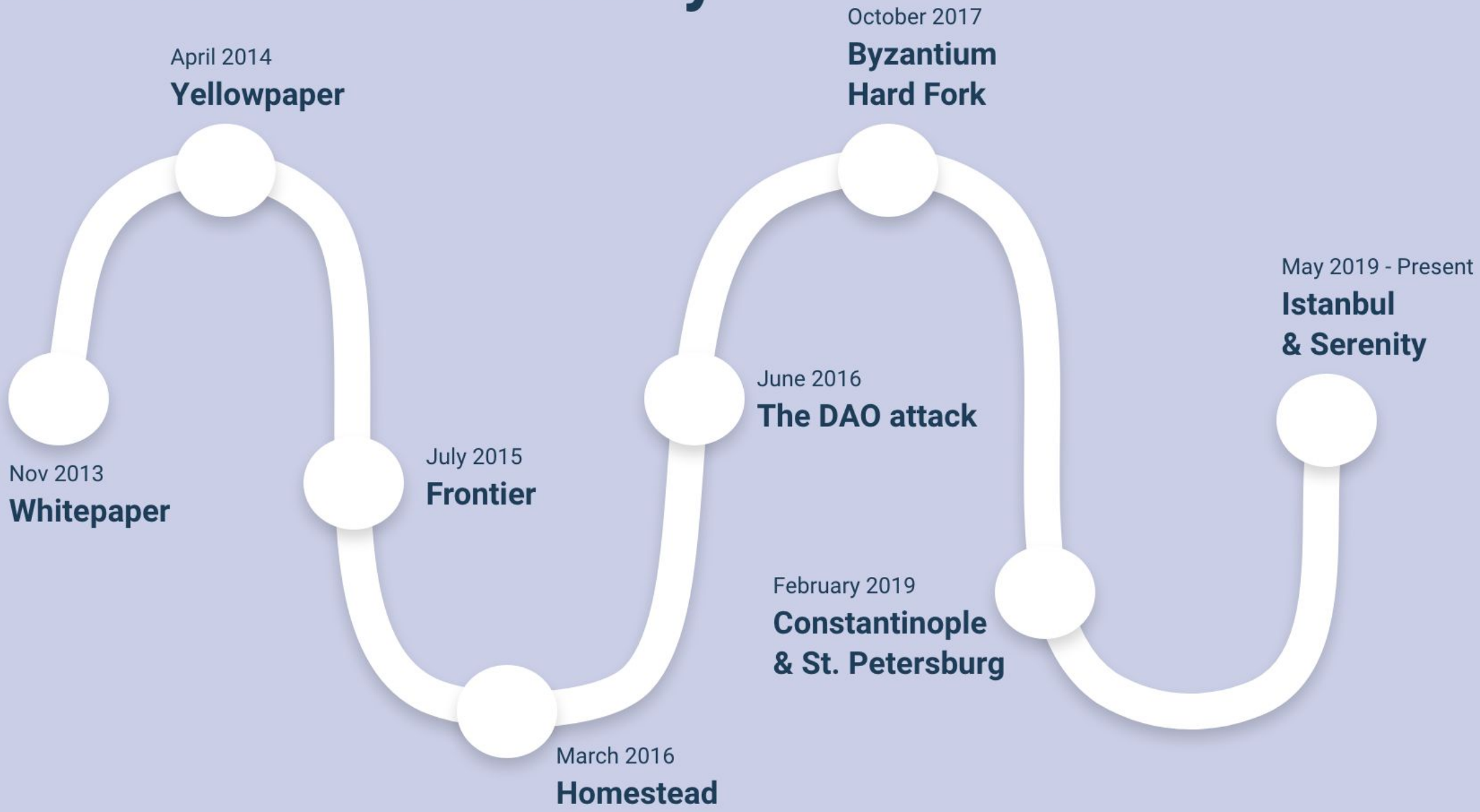
Ethereum clients and releases

- Various Ethereum clients have been developed using different languages
  - Most popular are go-Ethereum and parity.
- **go-Ethereum** was developed using Golang,
- **parity** was built using Rust.
- go-Ethereum client known as ***geth***
  - 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.

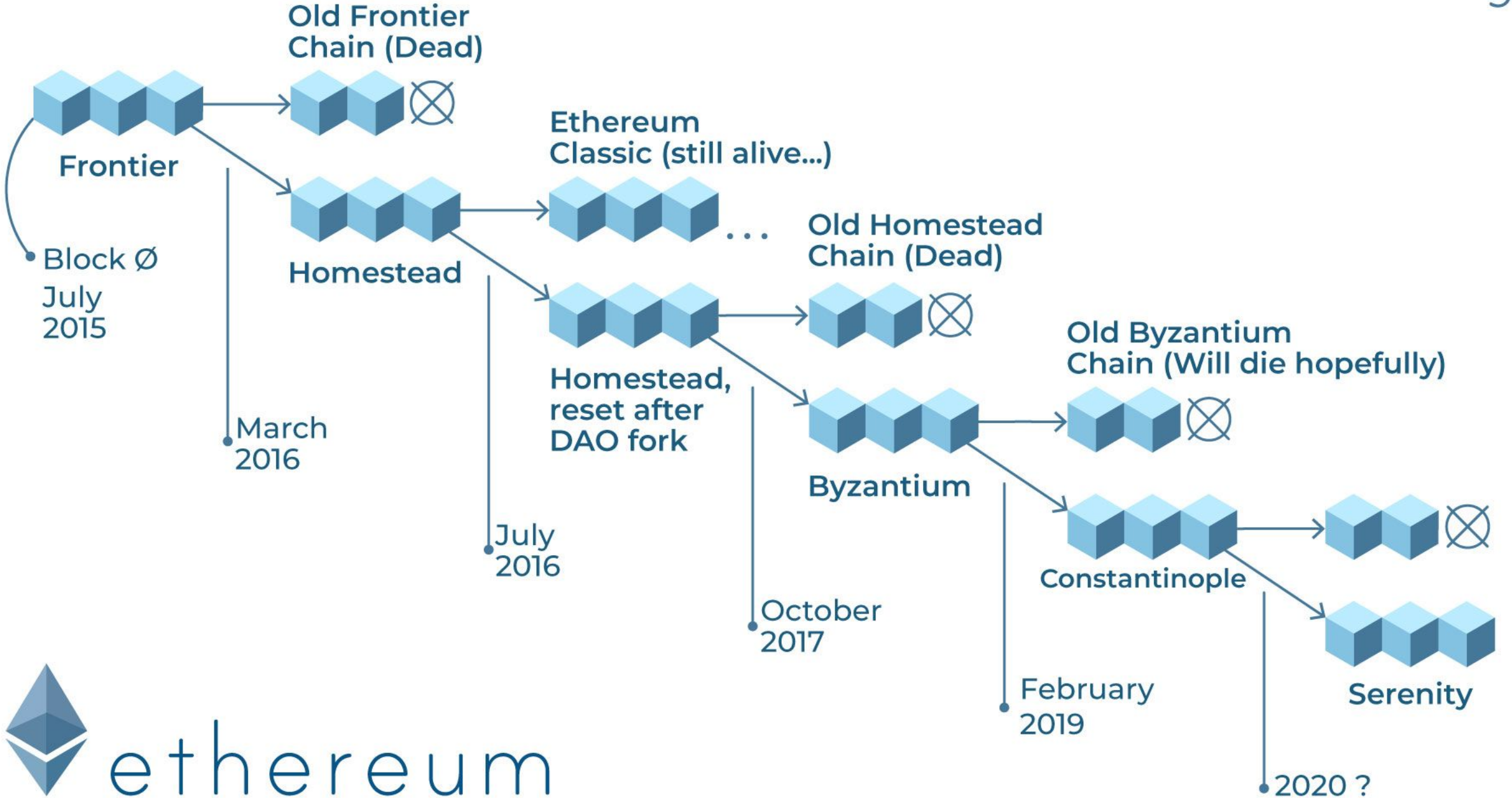
Ethereum clients and releases

- The first release of Ethereum was known as ***Frontier***
- current release of Ethereum is called ***St. Petersburgh*** *release*.
- The next release is named **Istanbul** .
- The final release is named ***serenity***, which is envisaged to have a **Proof of Stake** algorithm (Casper) implemented with it.
- Other areas of research targeted with serenity include
  - Scalability
  - Privacy
  - **Ethereum virtual machine** (**EVM**) upgrade.

Ethereum's Brief History



history



ethereum

Ethereum clients and releases

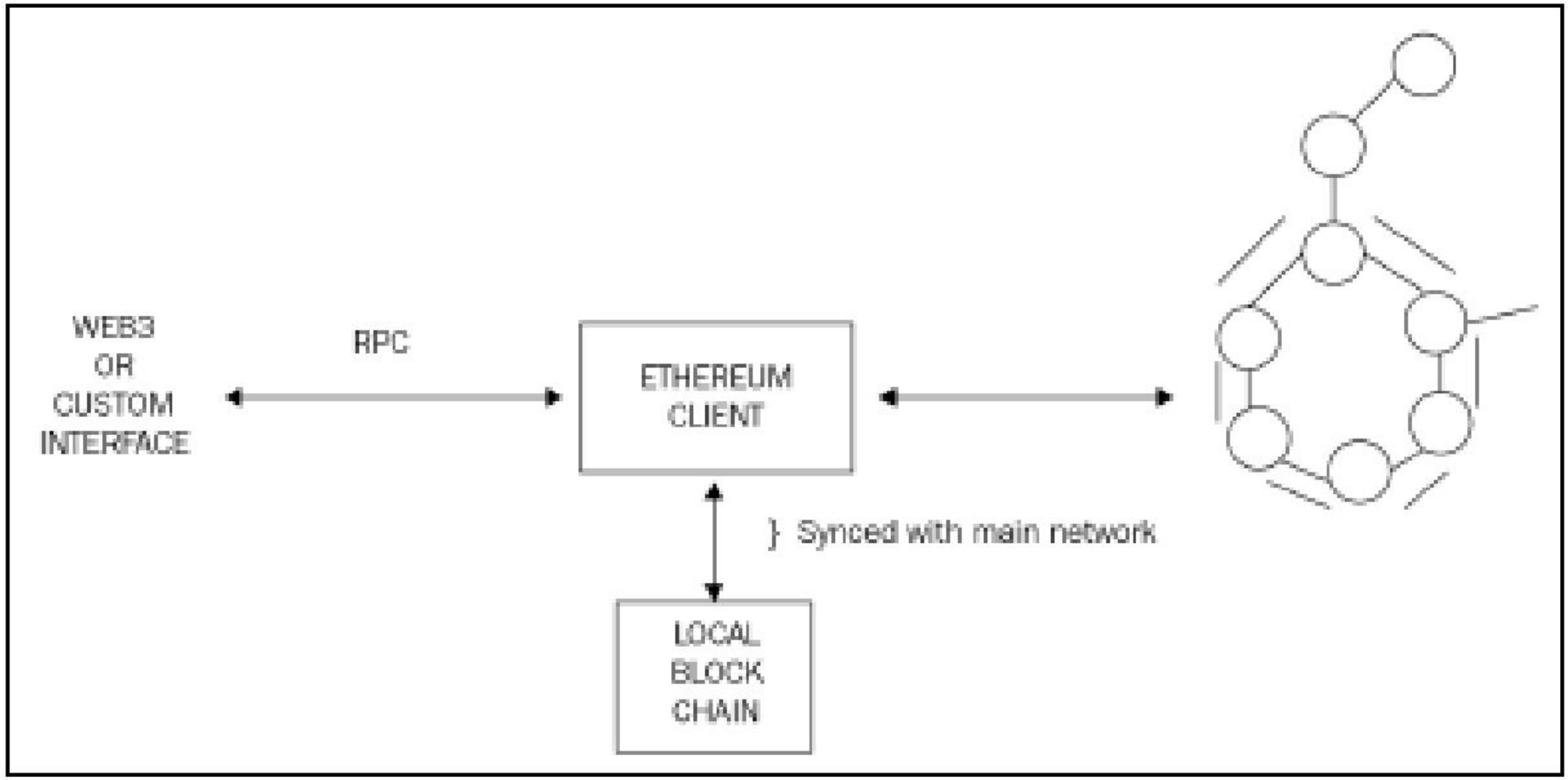
- Ethereum ecosystem will undergo constant improvement and development
- **serenity** should not really be considered a *final* version but a major milestone in a long journey of continuous improvement.
- Further releases are envisaged but have not been named yet.
- Vision of ***web 3.0*** has already been proposed and is being discussed in the community.

Ethereum clients and releases

- **Web 3.0** is a concept that basically proposes a semantic and intelligent web as an evolution of the existing web 2.0 technology.
  - 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
- All major services, such as DNS, search engines, and identity on the Internet will be decentralized in web 3.0.
  - Ethereum is being envisaged as a platform that can help realize this vision.

The Ethereum stack

- 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
    - blockchain is **downloaded** and **stored locally**.
    - It provides various functions, such as **mining** and **account management**.
    - ocal 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.

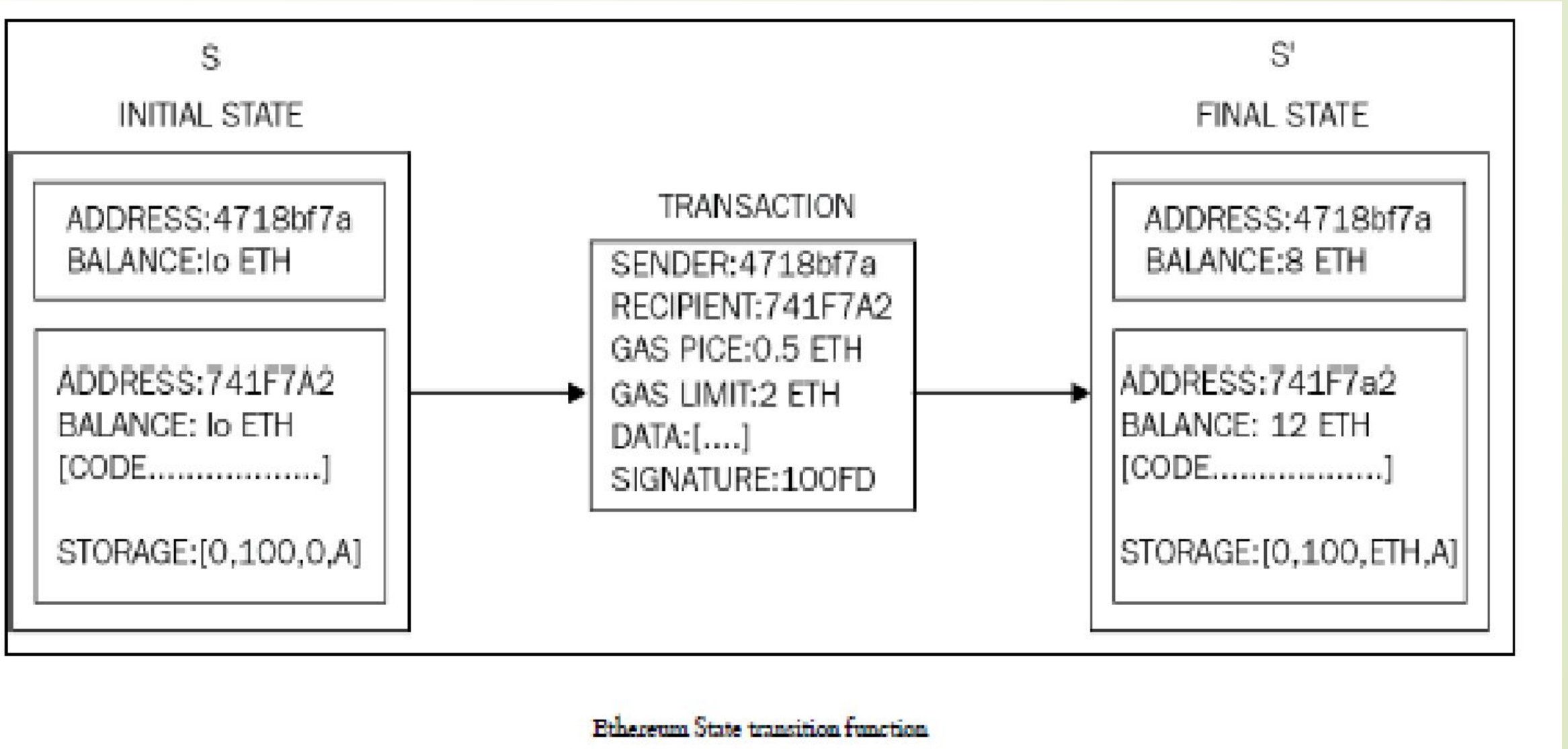


The Ethereum stack showing various components

Ethereum blockchain

- Ethereum, can be visualized as a **transaction-based state machine.**
  - **Genesis state is transformed into a final state** by executing transactions incrementally.
  - **Final transformation** is then accepted as the absolute undisputed version of the state.

Transfer of 2 Ether from Address 4718bf7a to Address 741f7a2 is initiated



Ethereum State 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 a hard fork was proposed in order to mitigate the issue;
  - There are now two Ethereum blockchains:
    - **Ethereum classic** and its currency is represented by **ETC**
    - Hard-forked version is **ETH**, which continues to grow
  - ETC, has a dedicated community that is further developing ETC, which is the nonforked original version of Ethereum.

Forks

- With the release of **homestead,** due to major protocol upgrades, it resulted in a **hard fork**.
- 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.**

Forks

- Recent 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.
- Cost is covered by something called ***gas*** or ***crypto fuel***,
- Gas as ***execution fee*** is paid upfront by the transaction originators.
- ***fuel*** is consumed with each operation.

Gas

- **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**.
- If the transaction is successfully executed, then any **remaining gas is refunded** to the transaction originator.

The consensus mechanism

- Consensus mechanism in Ethereum is based on the GHOST protocol originally proposed by *Zohar* and *Sompolinsky* in December 2013
- Ethereum uses a simpler version of GHOST protocol,
  - where the **chain** that has **most computational effort spent** on it in order to build it is identified as the **definite version**.
  - Another way is to find the **longest chain**, as the longest chain must have **been built by consuming adequate mining effort**.

The consensus mechanism

- **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.
- **Stale blocks :**
  - **B**locks that were propagated to the network and verified by some nodes as being correct but eventually being cast off as a longer chain achieved dominance, or Forking
  - Orphan, or stale block, is created when two nodes find a block at the same time. **Both say we've found the solution to this block** and send off their block to be verified and included in others block chains.
- 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.

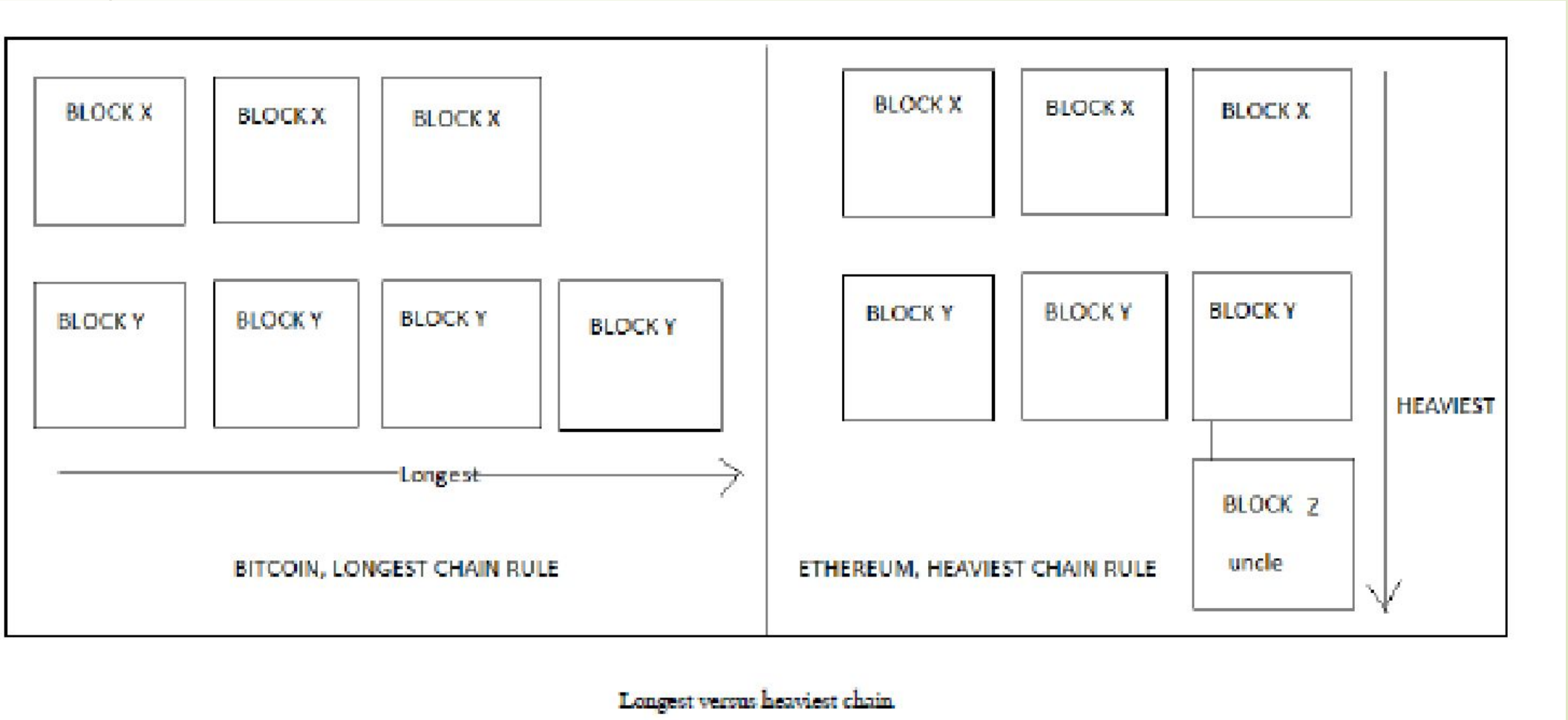
The consensus mechanism

- GHOST includes stale blocks – or Uncles
  - These are included in the calculation of which chain is longest or has the highest cumulative difficulty.
- Centralisation is solved by giving block rewards to stales of 87.5%
  - Nephew (child of the Uncle block) also receives a reward of 12.5% of the block reward.

The consensus mechanism

- The Ethereum version of Ghost only goes down seven levels – or back seven levels in the height of the block chain.
  - A block must specify its parents and its number of Uncles.
  - An Uncle included in a block must be a direct child of the new block and less than seven blocks below it in terms of height
  - It cannot be the direct ancestor of the block being formed.
  - An Uncle must have a valid block header.
  - An Uncle must be different from all other Uncles in previous blocks and the block being formed.
  - For every Uncle included in the block the miner gets an additional 3.125% of a standard block reward.
  - Miner of of the Uncle receives 93.75% of a standard block reward.

- Blocks with less difficulty add less work
  Two blocks in the same difficulty period always add the same amount of work to the chain



Longest versus heaviest chain

The world state

- World state in Ethereum represents the **global state** of the Ethereum blockchain.
  - **Mapping between Ethereum addresses and account states.**
  - Addresses are 20 bytes long.
- This mapping is a data structure that is serialized using **Recursive Length Prefix (RLP)**.
- RLP is a **specially developed encoding scheme** that is used in Ethereum to
  - serialize binary data for storage
  - transmission over the network
  - save the state in a Patricia tree.
- RLP function takes an item as an input, which can be a string or a list of items
  - 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

- Account state consists of four fields:
  - nonce
  - balance
  - storageroot
  - codehash

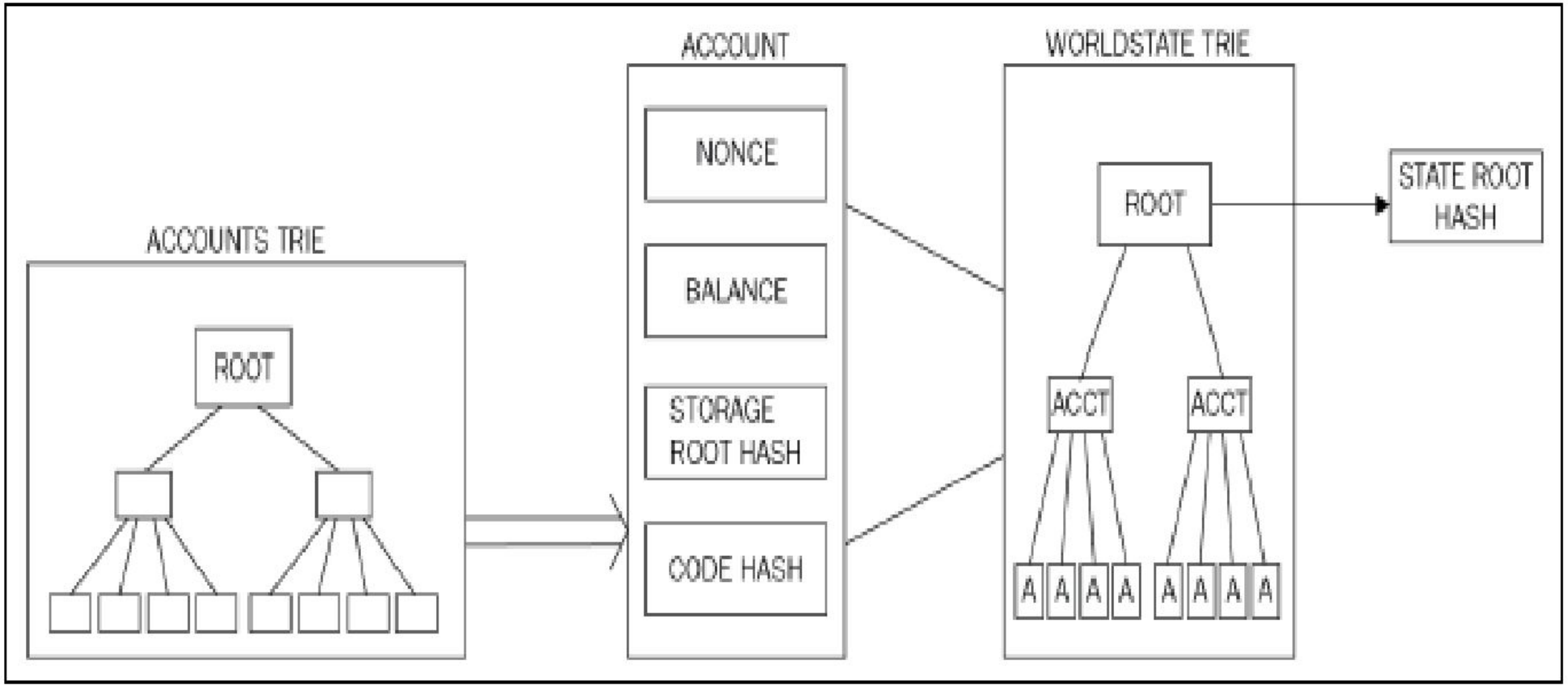
Account state

- **Nonce**
  - Value that is incremented every time a transaction is sent from the address.
  - In case of contract accounts, it represents the number of contracts created by the account.
    - Contract accounts are one of the two types of accounts that exist in Ethereum
- **Balance**
  - Represents the number of Weis which is the smallest unit of the currency (Ether) in Ethereum held by the address.

The account state

- **Storageroot**
  - 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.
  - 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.

world state and its relationship with accounts trie, accounts, and block header



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

world state and its relationship with accounts trie, accounts, and block header

- It shows the account data structure in the middle of the diagram,
  - which contains a storage root hash derived from the root node of the account storage trie shown on the left.
- Account data structure is then used in the world state trie,
  - which is a mapping between addresses and account states.
- 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 state root hash.

World state and its relationship with accounts trie, accounts, and block header

- Accounts trie is basically a Merkle Patricia tree used to encode the storage contents of an account.
- Contents are stored as a mapping between keccak 256-bit hashes of 256-bit integer keys to the RLP-encoded 256-bit integer values.

Transactions

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

Transactions

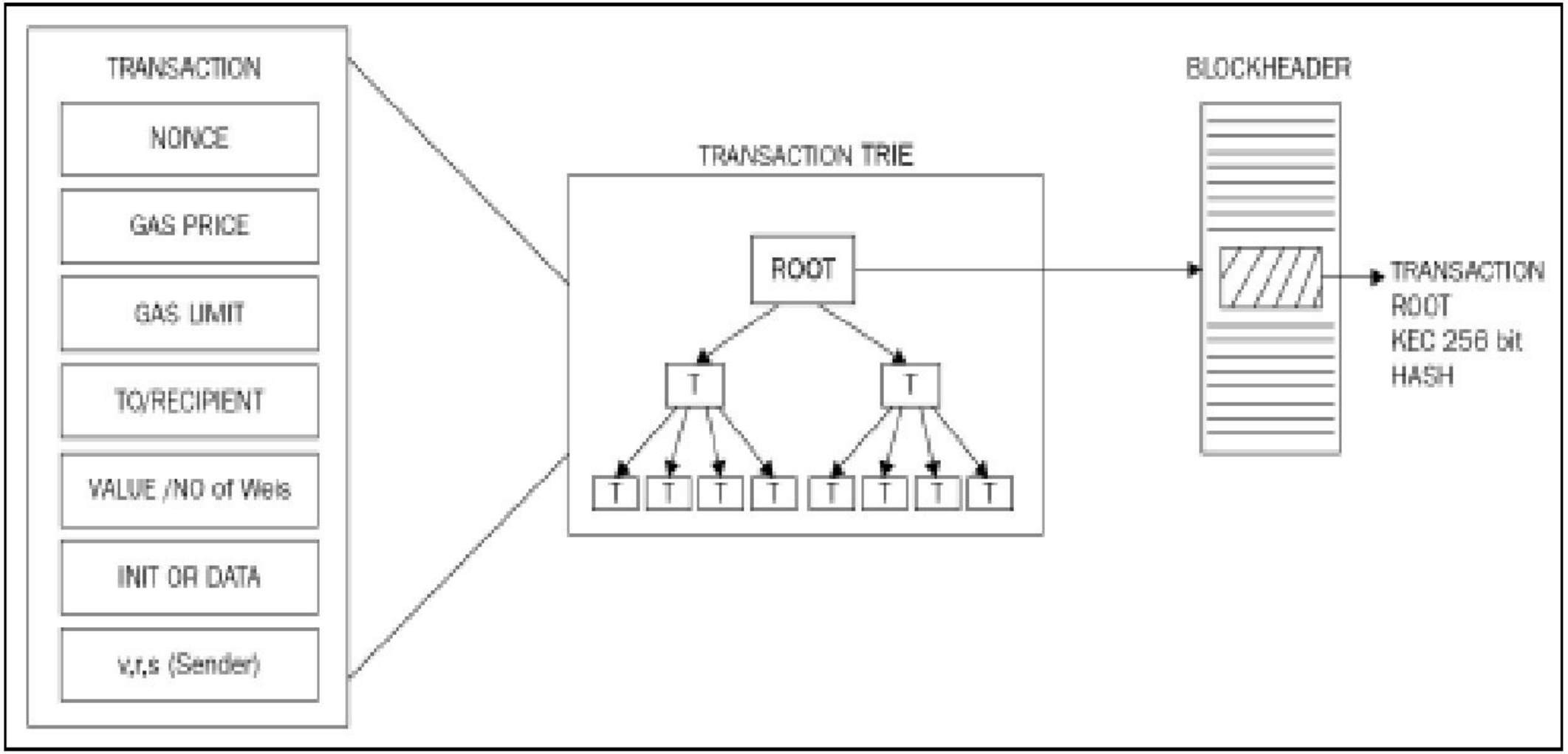
- Both transactions are composed of a number of common fields
- **Nonce**
  - Nonce is a number that is incremented by one every time a transaction is sent by the sender.
  - It must be equal to the number of transactions sent and is used as a unique identifier for the transaction.
  - Nonce value can only be used once.
- **gasPrice**
  - gasPrice field represents the amount of Wei required in order to execute the transaction.

- **gasLimit**
  - gasLimit field contains the value that represents the maximum amount of gas that can be consumed in order to execute the transaction.
  - This is the amount of fee in **Ether** that a user is willing to pay for computation
- **To**
  - **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**
  - Signature is composed of three fields, namely v, r, and s.
  - Represent the **digital signature (R, S)** and some information that can be used to recover the **public key (V).**
  - **Sender of the transaction** can also be determined
  - Signature is based on ECDSA scheme and makes use of the SECP256k1 curve.

- **Init**
  - **Init field** is used only in transactions that are intended to create **contracts.**
  - Byte array of unlimited length that specifies the **EVM code** to be used in the account initialization process.
  - **Code contained in this field is executed only once,** when the account is created for the first time, and 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.



Relationship between transaction, transaction trie and block header

Transactions

- Transaction is a tuple of the fields, which is then included in a transaction trie (modified Merkle-Patricia tree)
- **Root node of transaction trie is hashed using a Keccak 256-bit algorithm** and is included in the block header along with a list of transactions in the block.
- Transactions can be found in either **transaction pools or blocks.**
- 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.

Transactions

- In Mining process, the **block is repeatedly hashed until a valid nonce is found** that, once hashed with the block, 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.
  - Similar to Bitcoin's mining process
  - The only difference is that Ethereum's Proof of Work algorithm is **ASIC-resistant**, known as ***Ethash***, where finding a nonce requires large memory.

Contract creation transaction

- There are a few essential parameters that are required when creating an account
- Parameters :
  - Sender
  - Original transactor
  - Available gas
  - Gas price
  - Endowment -amount (ether) allocated initially
  - 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

Contract creation transaction

- Addresses generated as a result of contract creation transaction are 160-bit in length.
  - Rightmost 160-bits of the Keccak hash of the RLP encoding of the structure containing only the sender and the nonce.
- Initially, nonce in the account is set to zero.
- 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.

Contract creation transaction

- Account is initialized when the EVM code (Initialization EVM code) is executed.
- Exception during code execution, such as not having enough gas
  - **state does not change.**
- Execution is successful
  - **account is created after the payment of appropriate gas costs**.
- Result of contract transaction is either
  - **new contract with its balance**
  - **no new contract is created with no transfer of value.**

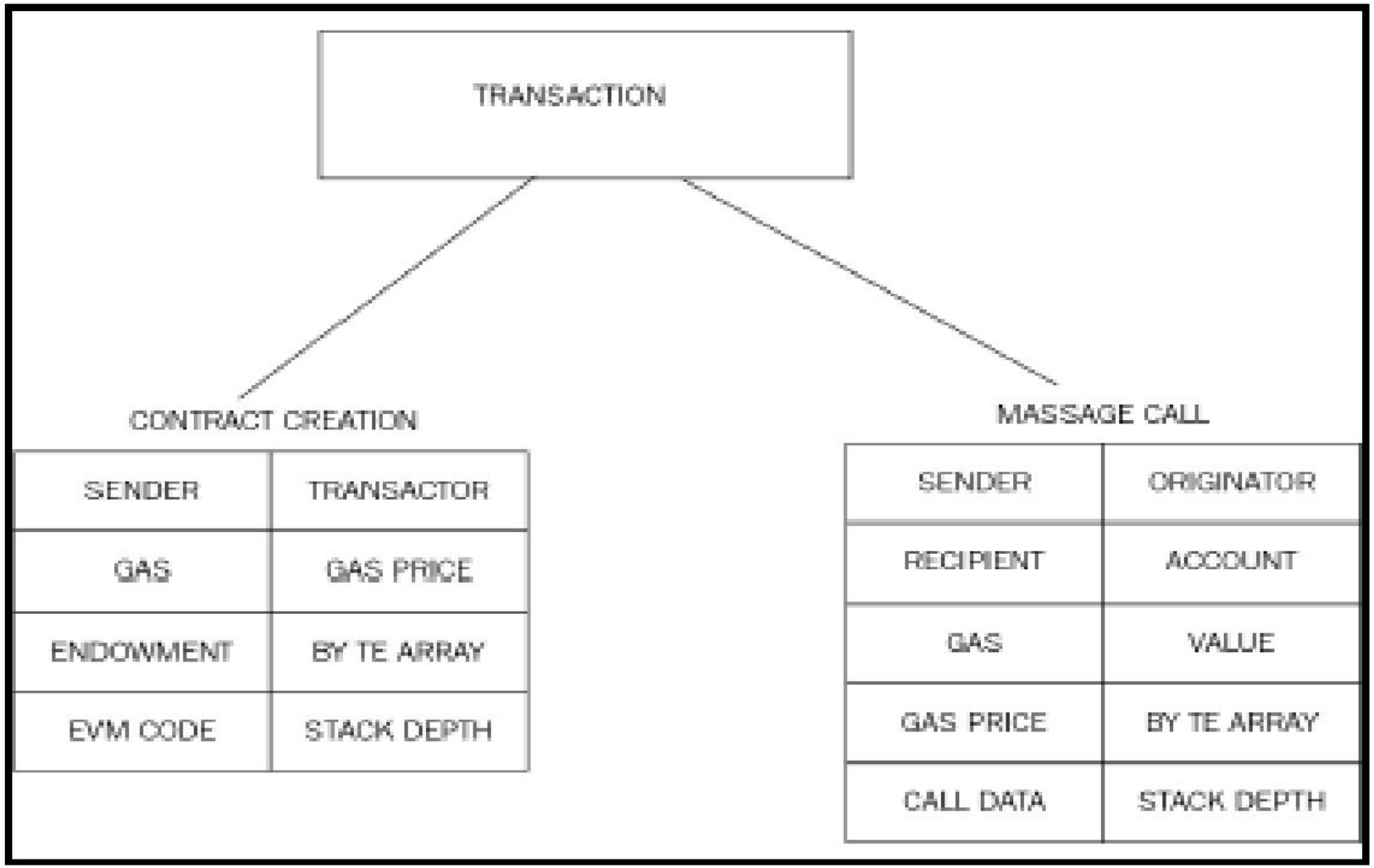
Message call transaction

- Message call requires several parameters for execution
  - Sender
  - Transaction originator
  - Recipient
  - 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

Message call transaction

- Message calls result in state transition.
- Message calls also produce output data, which is not used if transactions are executed.
- If the message calls are triggered by VM code, the output produced by the transaction execution is used.

Two types of transaction



Types of transactions, required parameters for execution

Elements of the Ethereum blockchain - Ethereum virtual machine (EVM)

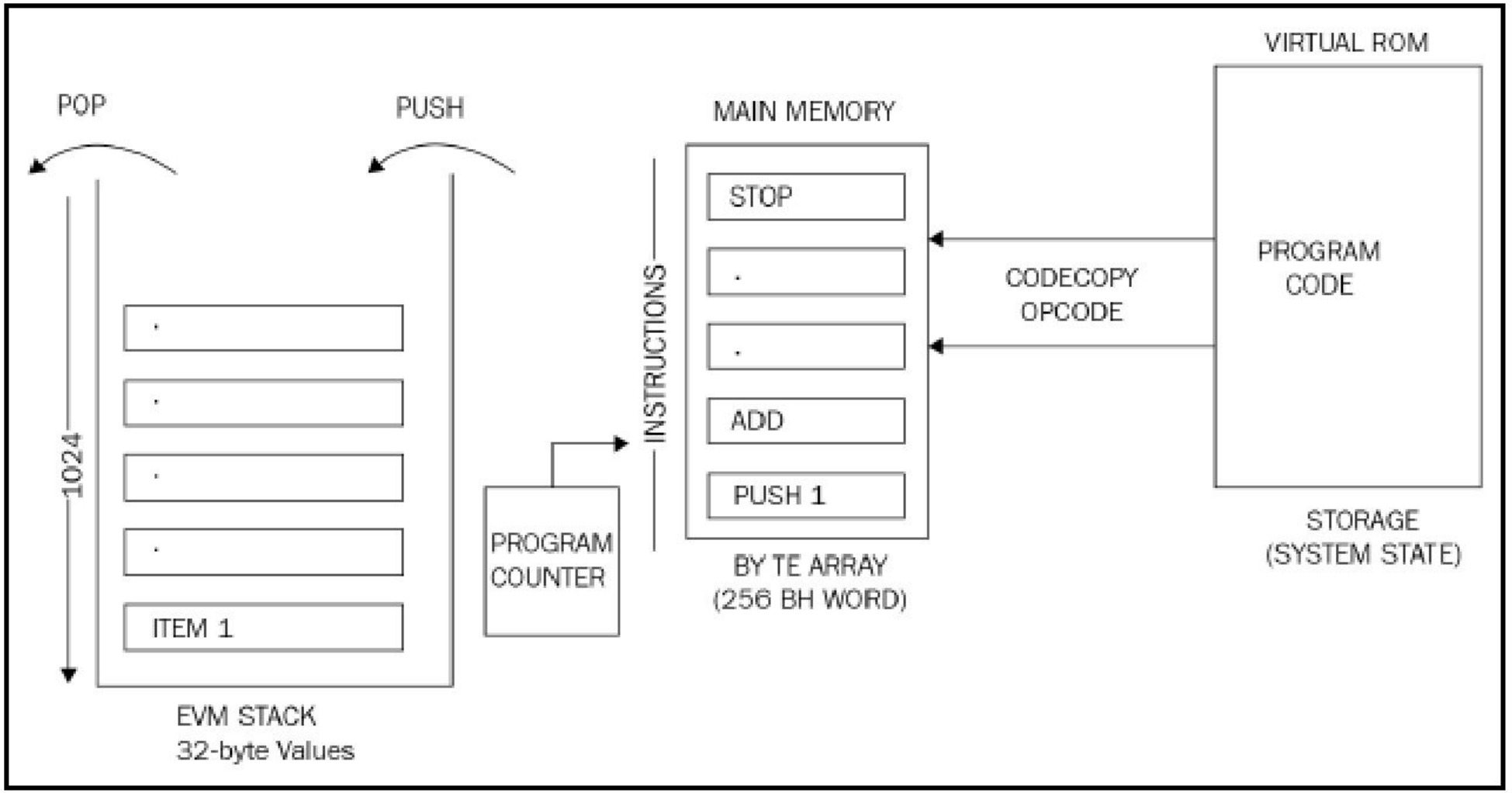
- EVM is a simple **stack-based execution machine** that **runs bytecode instructions**
  - transform the system state from one state to another.
- **Word size** of the virtual machine is set to 256-bit.
- **Stack size** is limited to 1024 elements and is based on the **LIFO** (**Last in First Out**) queue.
- EVM is a **Turing-complete machine** but is **limited by the amount of gas** that is required to **run any instruction**.
  - **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,
  - **Machine would immediately halt and return the error to the executing agent**.

Ethereum virtual machine (EVM)

- EVM is a fully **isolated** and **sandboxed** runtime environment.
- Code that runs on the EVM **does not have access to any external resources**, such as a network or file system.
- EVM is a **stack-based architecture.**
- EVM is **big-endian by design** and it uses 256-bit wide words.
- Word size allows for Keccak 256-bit hash and elliptic curve cryptography computations.
- Two types of storage available to contracts and EVM.
  - Memory, which is a byte array.
    - When a contract finishes the code execution, the memory is cleared.
  - Storage, is permanently stored on the blockchain.
    - It is a key value store.

Ethereum virtual machine (EVM)

- Memory is **unlimited** but constrained by gas fee requirements.
- Storage associated with the virtual machine is a **word array** that is nonvolatile and is maintained as part of the system state.
- Keys and value are 32 bytes in size and storage.
- Program code is stored in a **virtual read-only memory** (virtual ROM) that is accessible using the **CODECOPY** instruction.
- **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.



EVM operation

Ethereum virtual machine (EVM)

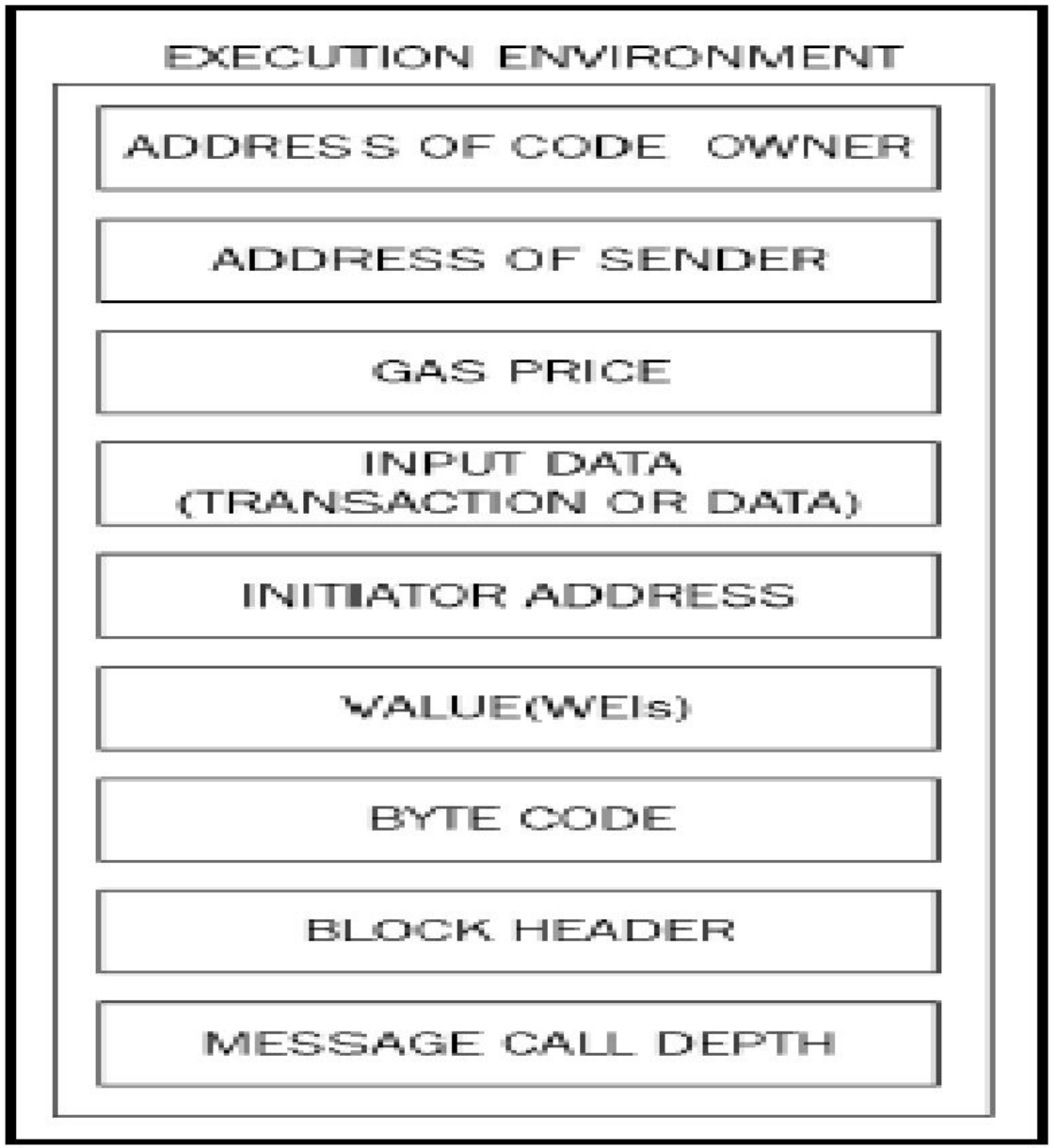
- Virtual ROM stores the program code that is copied into main memory using CODECOPY.
- Main memory is then read by the EVM by referring to the program counter and executes instructions step by step.
- Program counter and EVM stack are updated accordingly with each instruction execution.
- EVM optimization is an active area of research
  - to achieve high performance.

Execution environment

- Key parameters are provided by the execution agent
  - Address of the account that owns the executing code.
  - Address of the sender of the transaction and the originating address of this execution.
  - Gas price in the transaction that initiated the execution.
  - Input data or transaction data depending on the type of executing agent.
    - 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

- Address of the account that initiated the code execution or transaction sender.
- value or transaction value.
  - Amount in Wei. If the execution agent is a transaction, then it is the transaction value.
- Code to be executed presented as a **byte array** that the iterator function picks up in each execution cycle.
- Block header of the current block
- Number of message calls or contract creation transactions currently in execution.
  - Number of CALLs or CREATEs currently in execution.



Execution environment Tuple

Execution environment

- In addition to the nine fields, **system state** and the **remaining gas** are also provided to the execution environment.
- Execution results in producing
  - Resulting state
  - Gas remaining after the execution
  - Self-destruct or suicide set
  - Log series
  - Any gas refunds.

Gas

- Halting problem (infinite loop) – reason for Gas
  - Problem: Cannot tell whether or not a program will run infinitely from compiled code
  - Solution: charge fee per computational step to limit infinite loops and stop flawed code from executing
- Every transaction needs to specify an **estimate of the amount of gas it will spend**
  - Essentially a measure of how much one is willing to spend on a transaction, even if buggy

Gas Cost

- Gas Price: current market price of a unit of Gas (in Wei)
  - Check gas price here: https://ethgasstation.info/
  - Is always set before a transaction by user
- Gas Limit: maximum amount of Gas user is willing to spend
  - Helps to regulate load on network
- Gas Cost (used when sending transactions) is calculated by gasLimit*gasPrice.
  - All blocks have a Gas Limit (maximum Gas each block can use)

Code Execution

- Every node contains a virtual machine (similar to Java)
  - Called the Ethereum Virtual Machine (EVM)
  - **Compiles** code from high-level language to bytecode
  - Executes smart contract code and broadcasts state
- *Every full-node on the blockchain processes every transaction and stores the entire state*

Machine state

- Machine state is also maintained internally by the EVM.
- Machine state is updated after each execution cycle of EVM.
- 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:
  - Available gas
  - Program counter, which is a positive integer up to 256
  - Memory contents
  - Active number of words in memory
  - Contents of the stack

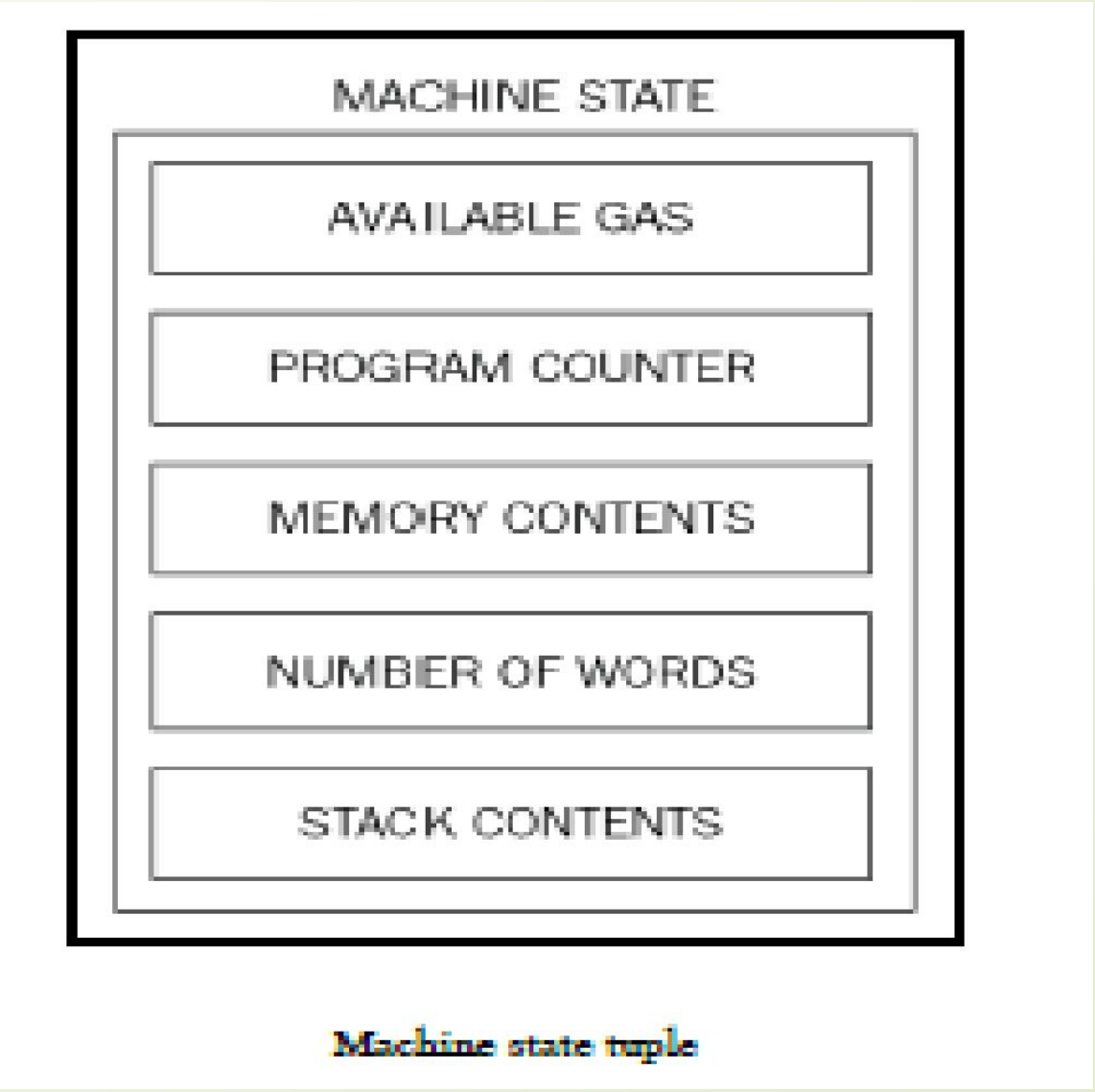
Machine state

- 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 op codes
  - Invalid stack size (greater than 1024)

The iterator function

- Iterator function 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**).

Machine state can be viewed as a tuple



Machine state tuple

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

simple program in solidity

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

- This program is converted into bytecode
- https://remix.ethereum.org/

Opcodes and their meaning

- Different opcodes that have been introduced in the EVM.
- Opcodes are divided into multiple categories based on the operation they perform.

Arithmetic operations

Mnemonic	Value	POP	PUSH	Gas	Description
STOP	0x00	0	0	0	Halts execution
ADD	0x01	2	1	3	Adds two values
MUL	0x02	2	1	5	Multiplies two values
SUB	0x03	2	1	3	Subtraction operation
DIV	0x04	2	1	5	Integer division operation
SDIV	0x05	2	1	5	Signed integer division operation
MOD	0x06	2	1	5	Modulo remainder operation
SMOD	0x07	2	1	5	Signed modulo remainder operation
ADDMOD	0x08	3	1	8	Modulo addition operation
MULMOD	0x09	3	1	8	Module multiplication operation
EXP	0x0a	2	1	10	Exponential operation (repeated multiplication of the base)
SIGNEXTEND	0x0b	2	1	5	Extends the length of 2s complement signed integer

Logical operations

Mnemonic	Value	POP	PUSH	Gas	Description
LT	0x10	2	1	3	Less than
GT	0x11	2	1	3	Greater than
SLT	0x12	2	1	3	Signed less than comparison
SGT	0x13	2	1	3	Signed greater than comparison
EQ	0x14	2	1	3	Equal comparison
ISZERO	0x15	1	1	3	Not operator
AND	0x16	2	1	3	Bitwise AND operation
OR	0x17	2	1	3	Bitwise OR operation
XOR	0x18	2	1	3	Bitwise exclusive OR (XOR) operation
NOT	0x19	1	1	3	Bitwise NOT operation
BYTE	0x1a	2	1	3	Retrieve single byte from word

Cryptographic operations

Mnemonic	Value	POP	PUSH	Gas	Description
SHA3	0x20	2	1	30	Used to calculate Keccak 256-bit hash.

Environmental information

Mnemonic	Value	POP	PUSH	Gas	Description
ADDRESS	0x30	0	1	2	Used to get the address of the currently executing account
BALANCE	0x31	1	1	20	Used to get the balance of the given account
ORIGIN	0x32	0	1	2	Used to get the address of the sender of the original transaction
CALLER	0x33	0	1	2	Used to get the address of the account that initiated the execution
CALLVALUE	0x34	0	1	2	Retrieves the value deposited by the instruction or transaction
CALLDATALOAD	0x35	1	1	3	Retrieves the input data that was passed a parameter with the message call
CALLDATASIZE	0x36	0	1	2	Used to retrieve the size of the input data passed with the message call

Environmental information

CALLDATACOPY	0x37	3	0	3	Used to copy input data passed with the message call from the current environment to the memory.
CODESIZE	0x38	0	1	2	Retrieves the size of running the code in the current environment
CODECOPY	0x39	3	0	3	Copies the running code from current environment to the memory
GASPRICE	0x3a	0	1	2	Retrieves the gas price specified by the initiating transaction.
EXTCODESIZE	0x3b	1	1	20	Gets the size of the specified account code
EXTCODECOPY	0x3c	4	0	20	Used to copy the account code to the memory.

Block Information

Mnemonic	Value	POP	PUSH	Gas	Description
BLOCKHASH	0x40	1	1	20	Gets the hash of one of the 256 most recently completed blocks
COINBASE	0x41	0	1	2	Retrieves the address of the beneficiary set in the block
TIMESTAMP	0x42	0	1	2	Retrieves the time stamp set in the blocks
NUMBER	0x43	0	1	2	Gets the block's number
DIFFICULTY	0x44	0	1	2	Retrieves the block difficulty
GASLIMIT	0x45	0	1	2	Gets the gas limit value of the block

Stack, memory, storage and flow operations

Mnemonic	Value	POP	PUSH	Gas	Description
POP	0x50	1	0	2	Removes items from the stack
MLOAD	0x51	1	1	3	Used to load a word from the memory.
MSTORE	0x52	2	0	3	Used to store a word to the memory.
MSTORE8	0x53	2	0	3	Used to save a byte to the memory
SLOAD	0x54	1	1	50	Used to load a word from the storage
SSTORE	0x55	2	0	0	Saves a word to the storage
JUMP	0x56	1	0	8	Alters the program counter
JUMPI	0x57	2	0	10	Alters the program counter based on a condition
PC	0x58	0	1	2	Used to retrieve the value in the program counter before the increment.
MSIZE	0x59	0	1	2	Retrieves the size of the active memory in bytes.
GAS	0x5a	0	1	2	Retrieves the available gas amount
JUMPDEST	0x5b	0	0	1	Used to mark a valid destination for jumps with no effect on the machine state during the execution.

Push operations

- PUSH operations used to place items on the stack.
- Range of these instructions is from 0x60 to 0x7f.
- There are 32 PUSH operations available in total in the EVM.
- PUSH operation, which reads from the byte array of the program code.

Mnemonic	Value	POP	PUSH	Gas	Description
PUSH1 . . . PUSH 32	0x60 … 0x7f	0	1	3	Used to place *N* right-aligned big-endian byte item(s) on the the stack. *N* is a value that ranges from 1 byte to 32 bytes (full word) based on the mnemonic used.

Duplication operations

- Duplication operations are used to duplicate stack items.
- Range of values is from 0x80 to 0x8f.
- There are 16 DUP instructions available in the EVM.
- Items placed on the stack or removed from the stack also change incrementally with the mnemonic used;
  - for example, DUP1 removes one item from the stack and places two items on the stack, whereas DUP16 removes 16 items from the stack and places 17 items.

Mnemonic	Value	POP	PUSH	Gas	Description
DUP1 . . . DUP16	0x80 … 0x8f	X	Y	3	Used to duplicate the nth stack item, where *N* is the number corresponding to the DUP instruction used. *X* and *Y* are the items removed and placed on the stack, respectively.

Exchange operations

- SWAP operations provide the ability to exchange stack items.
- There are 16 SWAP instructions available and with each instruction, the stack items are removed and placed incrementally up to 17 items depending on the type of Opcode used.

Mnemonic	Value	POP	PUSH	Gas	Description
SWAP1 . . . SWAP16	0x90 … 0x9f	X	Y	3	Used to swap the nth stack item, where *N* is the number corresponding to the SWAP instruction used. *X* and *Y* are the items removed and placed on the stack, respectively.

Logging operations

- Logging operations provide opcodes to append log entries on the sub-state tuple's log series field.
- There are four log operations available in total and they range from value 0x0a to 0xa4.

Mnemonic	Value	POP	PUSH	Gas	Description
LOG0 . . . LOG4	0x0a … 0xa4	X	Y (0)	375, 750, 1125, 1500, 1875	Used to append log record with *N* topics, where *N* is the number corresponding to the LOG Opcode used. For example, LOG0 means a log record with no topics, and LOG4 means a log record with four topics. *X* and *Y* represent the items removed and placed on the stack, respectively. *X* and *Y* change incrementally, starting from 2, 0 up to 6, 0 according to the LOG operation used.

System operations

- System operations are used to perform various system-related operations, such as account creation, message calling, and execution control.

Mnemonic	Value	POP	PUSH	Gas	Description
CREATE	0xf0	3	1	32000	Used to create a new account with the associated code.
CALL	0xf1	7	1	40	Used to initiate a message call into an account.
CALLCODE	0xf2	7	1	40	Used to initiate a message call into this account with an alternative account's code.
RETURN	0xf3	2	0	0	Stops the execution and returns output data.
DELEGATECALL	0xf4	6	1	40	The same as CALLCODE but does not change the current values of the sender and the value.
SUICIDE	0xff	1	0	0	Stops (halts) the execution and the account is registered for deletion later

Accounts

- Accounts are one of the main building blocks of the Ethereum blockchain.
- 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 Ethereum state transition function

Ethereum state transition function-Works as follows

- Confirm the transaction validity by checking the syntax, signature validity, and nonce.
- Transaction fee is calculated and the sending address is resolved using the signature.
  - Sender's account balance is checked and subtracted accordingly and nonce is incremented.
  - Error is returned if the account balance is not enough.
- Provide enough ether (gas price) to cover the cost of the transaction.
  - Charged per byte incrementally according to the size of the transaction.

Ethereum state transition function-Works as follows

- In the next step, the actual transfer of value occurs.
  - Flow is from the sender's account to receiver's account.
  - 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.

Ethereum state transition function-Works as follows

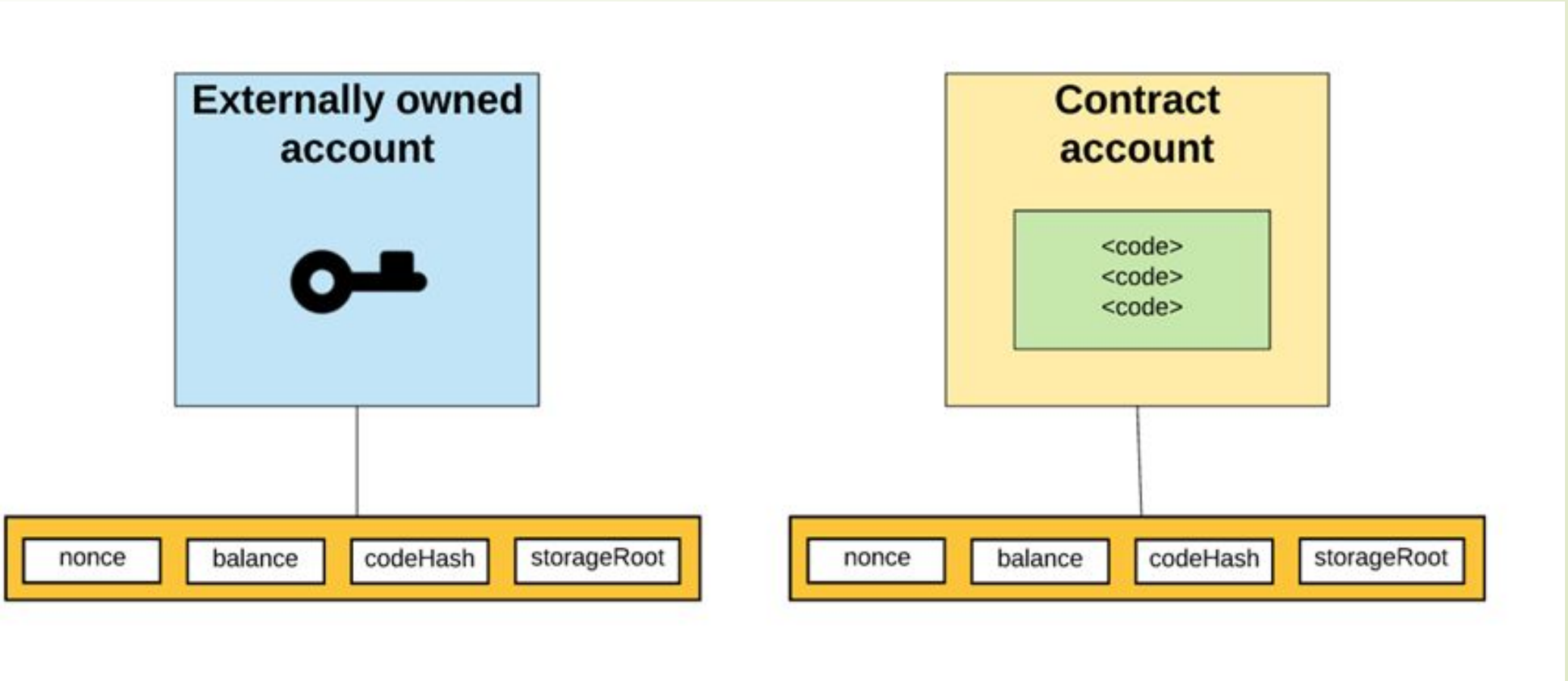
- In cases of transaction failure due to insufficient account balance or gas
  - all state changes are rolled back with the exception of fee payment, which is paid to the miners.
- Finally, the remainder of the fee is sent back to the sender as change and fee is paid to the miners accordingly.
- At this point, the function returns the resulting state.

Types of accounts

- Externally owned accounts
  - EOAs are similar to accounts that are controlled by a private key in bitcoin
- Contract accounts
  - Contract accounts are the accounts that have code associated with them along with the private key.

Types of Account

- EOA has **ether balance**, is able to send transactions, and has no associated code
- **Contract Account** (CA) has **ether balance**, **associated code,** and the ability to get triggered and execute code in response to a transaction or a message.
  - Code within contract accounts can be of any level of complexity.
  - Code is executed by EVM by each mining node on the Ethereum network.
  - contract accounts are able to maintain their own permanent state and can call other contracts.
- **Serenity release-D**istinction between externally owned accounts and contract accounts may be eliminated.


Externally owned account
nonce
balance
codeHash
storageRoot
Contract account
<code>
<code>
<code>
nonce
balance
codeHash
storageRoot

Accounts

External Account (EOA, Valid Ethereum Address)

- Has an associated nonce (amount of transactions sent from the account) and a balance
- codeHash - Hash of associated account code, i.e. a computer program for a smart contract (hash of an empty string for external accounts, EOAs)
- Storage Root is root hash of Merkle-Patricia trie of associated account data

Accounts

Contract Account

- Ethereum accounts can store and execute code
  - Has an associated nonce and balance
  - codeHash - hash of associated account code
  - storageRoot contains Merkle tree of associated storage data

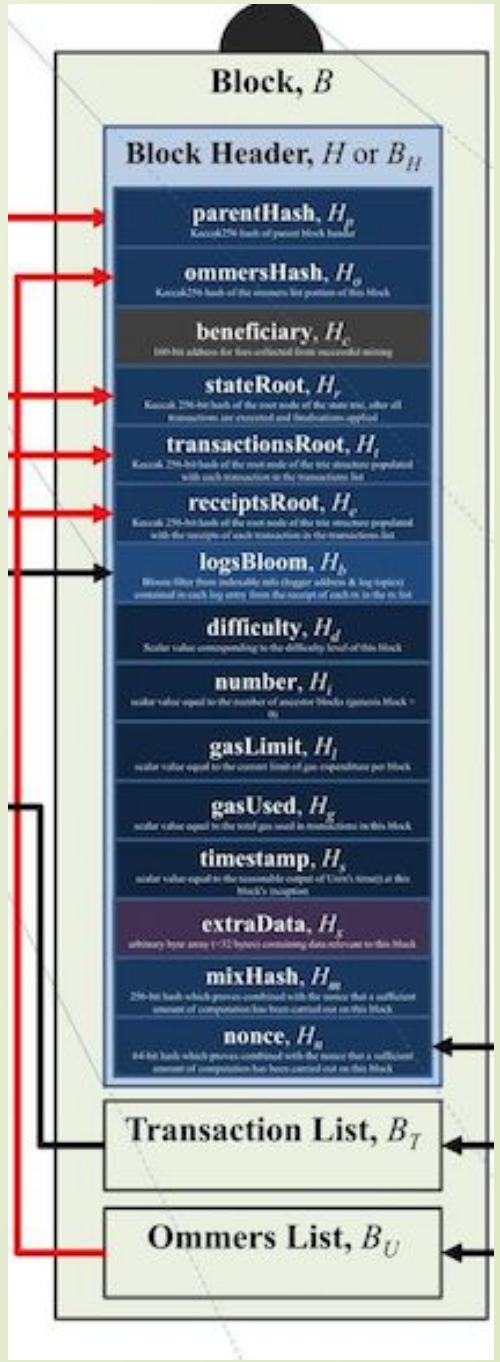
Block

- Blocks are the main building blocks of a blockchain.
- Ethereum blocks consist of various components
  - Block header
  - Transactions list
  - List of headers of Ommers or Uncles
- Transaction list is simply a list of all transactions included in the block.
- List of headers of Uncles is also included in the block.
- Most important and complex part is the block header

Ethereum Block

Blocks consist of 3 elements

- Transaction List
  - List of all transactions included in a block
- Block Header
  - Group of 15 elements
- Ommer List
  - List of all Uncle blocks included



Uncles/Ommers

- Sometimes valid block solutions don't make main chain
  - Any broadcast block (up to 6 previous blocks back) with valid PoW and difficulty can be included as an uncle
  - Maximum of two can be included per block
- Uncle block transactions are not included just header are included
- Aimed to decrease centralization and reward work

Uncles/Ommers Rewards:

- Uncle headers can be included in main block for 1/32 of the main block miner's reward given to said miner
- Miners of uncle blocks receive percent of main reward according to:
  - $(U_n + (8 - B_n)) * 5 / 8$, where U_n and B_n are uncle and block numbers respectively.
  - Example $(1333 + 8 - 1335) * \frac{5}{8} = 3.75$ ETH

Block header

- Block headers are the most critical and detailed components of an Ethereum block.
- Header contains valuable information
  - **Parent hash**
    - Keccak 256-bit hash of the parent (previous) block's header.
  - **Ommers hash**
    - Keccak 256-bit hash of the list of Ommers (Uncles) blocks included in the block.
  - **Beneficiary**
    - 160-bit address of the recipient that will receive the mining reward once the block is successfully mined.

Block header

- **State root**
  - Keccak 256-bit hash of the root node of the state trie.
  - Calculated after all transactions have been processed and finalized.
- **Transactions root**
  - 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**
  - 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.

Block header

- **Logs bloom**
  - 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**
  - Difficulty level of the current block.
- **Number**
  - Total number of all previous blocks; the genesis block is block zero.
- **Gas limit**
  - Contains the value that represents the limit set on the gas consumption per block.

Block header

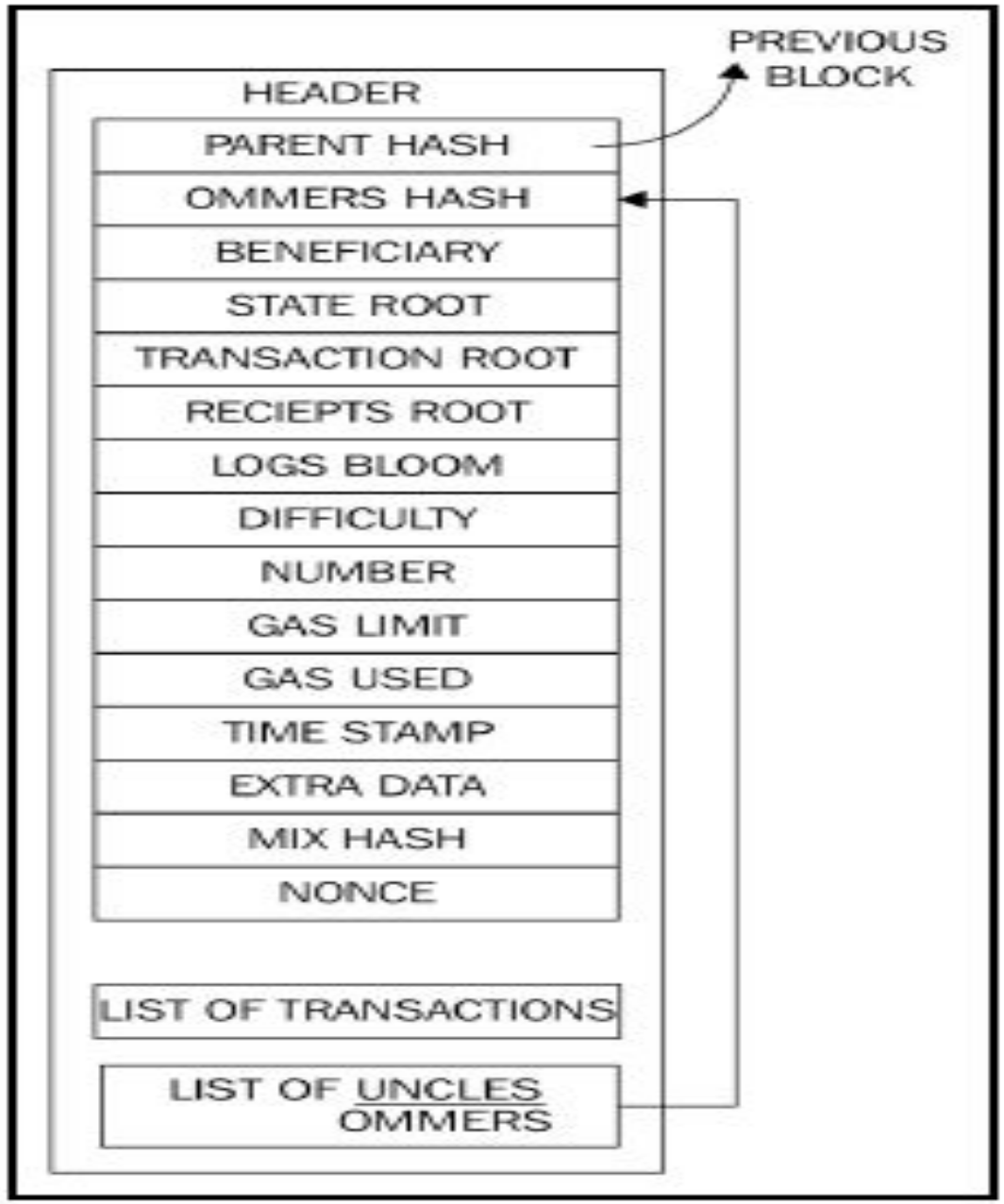
- **Gas used**
  - Total gas consumed by the transactions included in the block.
- **Timestamp**
  - Epoch Unix time of the time of block initialization.
- **Extra data**
  - Used to store arbitrary data related to the block.

Block header

- **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.

Extra Points

- **World state:** It is a mapping between addresses and account states.
- **Account state:** stores the state of each one of Ethereum's accounts. It also contains the storageRoot of the account state trie, that contains the storage data for the account.
- **Transaction:** represents a state transition in the system. It can be a funds transfer, a message call or a contract deployment.



Detailed diagram of block structure with block header

Example

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.
- 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.**
- It is composed of four elements

Transaction receipts- four elements

- **Post-transaction state**
  - trie structure that holds the state after the transaction has executed.
  - It is encoded as a byte array.
- **Gas used**
  - Represents the total amount of gas used in the block that contains the transaction receipt.
  - Value is taken immediately after the transaction execution is completed.
  - Total gas used is to be a non-negative integer.
- **Set of logs**
  - 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- four elements

- **Bloom filter**
  - Bloom filter is created from the information contained in the set of logs
  - 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.

Bloom Filter

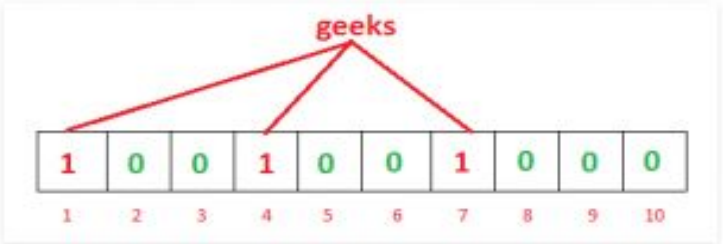
- Bloom filter is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set.
- For example, checking availability of username is set membership problem, where the set is the list of all registered username.

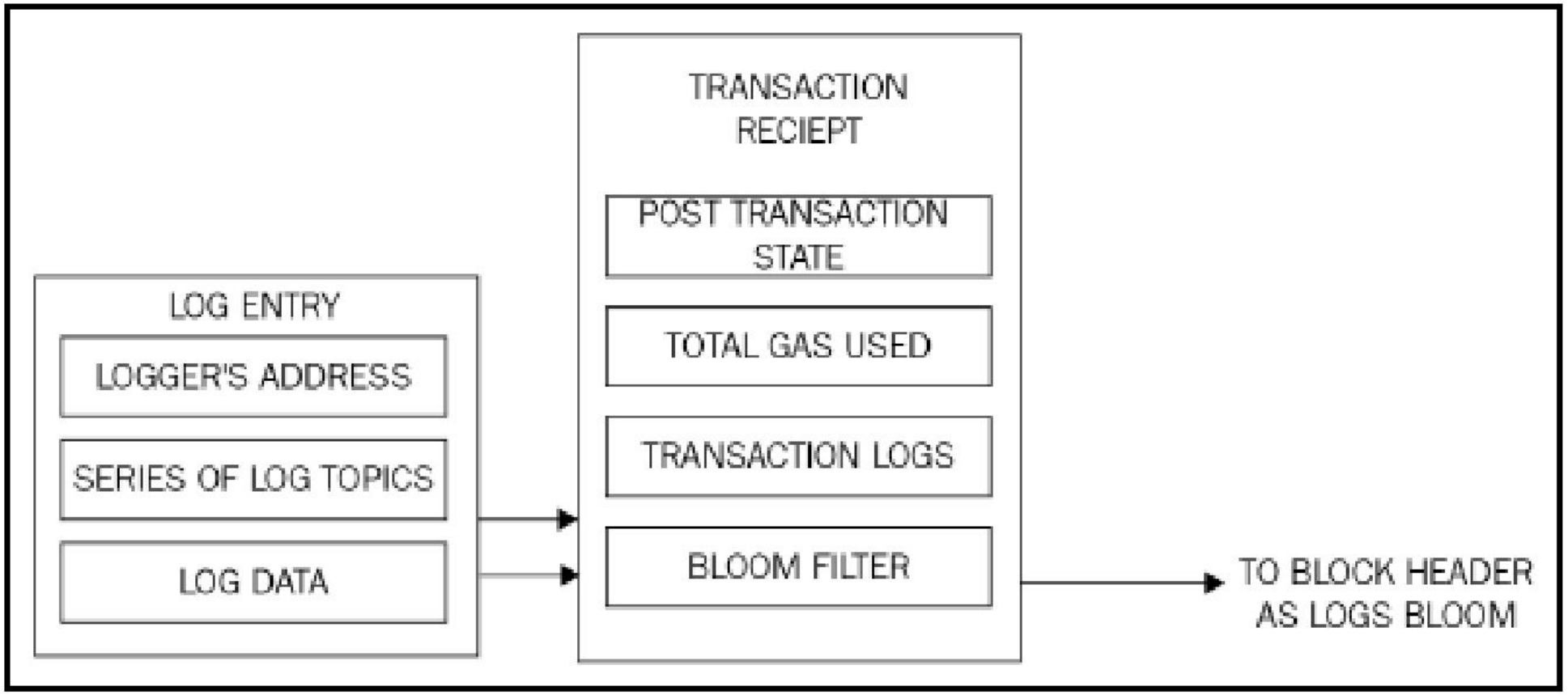
Example – Suppose we want to enter "geeks" in the filter, we are using 3 hash functions and a bit array of length 10, all set to 0 initially. First we'll calculate the hashes as following :

```
h1("geeks") % 10 = 1
h2("geeks") % 10 = 4
h3("geeks") % 10 = 7
```

Note: These outputs are random for explanation only.

Now we will set the bits at indices 1, 4 and 7 to 1





Transaction receipts and logs bloom

Transaction validation and execution

- Transactions are executed after verifying the transactions for validity.
- Initial tests are
  - Transaction must be well-formed and RLP-encoded without any additional trailing bytes
  - 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
  - Sender's account contains enough balance to cover the execution cost

Transaction sub state

- **Transaction sub-state** is created during the execution of the transaction that is processed immediately after the execution completes.
- Transaction sub-state is a tuple that is composed of three items
  - **Suicide set**
    - Contains the list of accounts that are disposed of after the transaction is executed.
  - **Log series**
    - **indexed series of checkpoints** that allow the monitoring and notification of contract calls to the entities external to the Ethereum environment
    - 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.

- **Refund balance**
  - 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.

Sub-state tuple

The block validation mechanism

- Ethereum block is considered valid if it passes the following checks:
  - Consistent with Uncles and transactions.
    - 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.
    - current block's timestamp must be higher than the parent block's timestamp.
    - It should be less than 15 minutes into the future.
    - All block times are calculated in epoch 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
- **Ommers validation**
  - Validate Ommers
  - In case of Miner, Determine Ommers.
  - 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.
  - block can contain a maximum of two Uncles.
- **State and nonce validation**
  - Verify the state and nonce.
  - In the case of mining, compute a valid state and nonce.

Block finalization

- **Transaction validation**
  - Validate transactions.
  - In the case of mining, determine transactions.
  - 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.
  - current block reward is 5 Ether

Block difficulty

- Block difficulty is increased if the time between two blocks decreases,
  - To maintain a roughly consistent block generation time.
- 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))
```

Block difficulty

- 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 (last line of the algorithm) 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
  - This will encourage users to switch to **Proof of Stake** as mining on the POW chain will eventually become prohibitively difficult.
- Block generation time will become significantly high during the year 2021, it will become so high that it will be virtually impossible to mine on the POW chain.
  - Miners will have no choice but to switch to the Proof of Stake scheme proposed by Ethereum called Casper.

Below is step by step process how difficulty of new block gets created.

- 1. First, difference between time of formation of parent block and new block is calculated.
- 2. Output of step 1 is then divided by 10 and integer of it is stored.
  - This is done to create ranges. If output of step 1 is between 1 – 9 then output of this step will be 0. If output of step 1 is between 10 – 19 then output of this step will be 1. If output of step 1 is between 20 – 29 then output of this step will be 2 and so on.
- 3. In order to create three ranges we will subtract 1 from output of step 2.
  - The three ranges will be either +ve, zero or –ve.
  - If you see it carefully then output of this step will be +ve when output of step 1 is between 0 – 9, zero when output of step 1 is between 10 – 19 and –ve when output of step 1 is anything more than 20.

- 4. Now compare output of previous step with -99 and if it is even lesser than -99 then set it as -99. This is done to limit the smallest possible value of step 3, otherwise keep the value of output of previous step as is.
- 5. Next we divide the parent block difficulty by the difficulty bound divisor, whose value is 2048.
- 6. Multiply output of step 4 with step 5. This will give the difference of difficulty of new block with old parent block. Depending is this value is +ve then difficulty will increase and if this value is –ve then then new difficulty will decrease.
- 7. Now add output of step 6 to parent difficulty and result will be the difficulty of the new block.
- 8. Once difficulty is calculated, a check is made to make sure that difficulty calculated is at least more than the minimum threshold value of 131072.

- 9. Before returning the difficulty, check is done that if block number is more than 200,000 then “The Bomb” logic is applied to increase the difficulty exponentially.
- 10. In order to increase the difficulty exponentially, new block number is calculated by adding one to the parent block number.
- 11. New block number is divided by 100,000.
- 12. If new block number is more than 200,000 then output of step 11 is subtracted from 2.
- 13. Now exponentially increased difficulty delta is calculated by doing following calculation: 2 ^ output of step 12.
  - Current Block Number 12605795
- 14. And new difficulty is calculated by adding output of previous step to the difficulty calculated in step 7.

Ether

- Ether is minted by miners as a currency reward for the **computational effort**
  - spent 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.
- Fees are charged for each computation performed by the EVM on the blockchain

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.
  - mechanism to ensures that infinite loops cannot cause the whole blockchain to stall due to the Turing-complete nature of the EVM.
- **Transaction fee** is charged as some amount of Ether and is **taken from the account balance of the transaction originator**.
- 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 the chances that the transactions will be picked up by the miners for inclusion in the block.
- 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*
- *gasUsed* is the total gas that is supposed to be used by the transaction during the execution
- *gasPrice* is specified by the transaction originator as an incentive to the miners to include the transaction in the next block. This is specified in Ether.
- 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 less operations than originally intended and can result in consuming more or fewer gas.
- if the execution is successful and there is some remaining gas, then it is returned to the transaction originator.

Example

- calculation of the SHA3 operation can be calculated as follows:
- SHA3 costs 30 gas
- Current gas price is 25 GWei, which is 0.000000025 Ether
- Multiplying both: *0.000000025 * 30 = 0.00000075* Ether

Operation Name	Gas Cost
step	1
stop	0
suicide	0
sha3	30
sload	20
txdata	5
transaction	500
contract creation	53000

Fee schedule

- Gas is charged in three scenarios as a prerequisite to the execution of an operation:
  - Computation of an operation
  - Contract creation or message call
  - Increase in the usage of memory

Messages

- Messages are the **data and value that are passed between two accounts.**
- Message is a **data packet** passed between two accounts.
- Data packet contains data and value (amount of ether).
  - It can either be sent via a smart contract (autonomous object) or from an external actor (externally owned account) in the form of a transaction that has been digitally signed by the sender.
- **Contracts can send messages to other contracts**.
- Messages only exist in the execution environment and are never stored.
- Messages are similar to transactions;
  - Messages are produced by the contracts,
  - Transactions are produced by entities external (externally owned accounts)

Messages

- Message consists of following components
  - Sender of the message
  - Recipient of the message
  - Amount of Wei to transfer
  - message to the contract address
  - Optional data field (Input data for the contract)
  - Maximum amount of gas that can be consumed
- Messages are generated when CALL or DELEGATECALL Opcodes are executed by the contracts.

Calls

- Call does not broadcast anything to the blockchain; instead**, it is a local call to a contract function** and runs locally on the node.
  - It is almost like a local function call.
- It does not consume any gas as it is a read-only operation.
- Calls are executed locally on a node and generally do not result in any state change.
  - this is the act of passing a message from one account to another.
- If the destination account has an associated EVM code, then the virtual machine will start upon the receipt of the message to perform the required operations.
- If the message sender is an autonomous object, then the call passes any data returned from the virtual machine operation.
- State is altered by transactions.

Mining

- Mining is the process by which new currency is added to the blockchain.
  - Incentive for the miners to validate and verify blocks made up of transactions.
- Mining process helps secure the network by verifying computations.

Mining

- At a theoretical level, a miner performs the following functions:
  - Listens for the transactions broadcasted on the Ethereum network and determines the transactions to be processed.
  - Determines stale blocks called Uncles or Ommers and includes them in the block.
  - Updates the account balance with the reward earned from successfully mining the block.
  - Finally, a valid state is computed and block is finalized, which defines the result of all state transitions.

Mining

- The current method of mining is based on Proof of Work, 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 Proof of Work for a given difficulty.
- Proof of Work algorithm is due to be replaced with the Proof of Stake algorithm with the release of serenity.
- Algorithm named **Casper** has been developed, which will replace the existing Proof of Work algorithm in Ethereum.
  - 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.**

Ethash

- **Ethash** is the name of the Proof of Work algorithm used in Ethereum.
  - Originally, this was proposed as the Dagger-Hashimoto algorithm
- Core idea behind mining is to find a nonce that once hashed the result in a predetermined difficulty level.
- Initially, 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.
- Ethash is a **memory-hard algorithm**, which makes it difficult to be implemented on specialized hardware

Ethash

- As in bitcoin, **ASICs have been developed**, which have resulted in mining centralization over the years
  - But memory-hard Proof of Work algorithms are one way of thwarting this threat
  - Ethereum implements Ethash to discourage ASIC development for mining.
- This algorithm requires choosing subsets of a fixed resource called **DAG** (**Directed Acyclic Graph**) depending on the nonce and block headers.
- DAG is around 2 GB in size and changes every 30000 blocks.
- Mining can only start when DAG is completely generated the first time a mining node starts.
- Time between every 30000 blocks is around 5.2 days and is called epoch.
- DAG is used as a seed by the Proof of Work algorithm called Ethash.

Ethash

- **Current reward scheme is 5 Ether for successfully finding a valid nonce**.
- In addition to receiving 5 Ethers, the successful miner also receives the cost of the gas consumed within the block and an additional reward for including stale blocks (Uncles) in the block.
- Maximum of two Uncles are allowed per block and are rewarded 7/8 of the normal block reward.
- In order to achieve a **12 second block time**, block difficulty is adjusted at every block.
- Rewards are directly proportional to the miner's hash rate, which basically means how fast a miner can hash.
- Mining can be performed by simply joining the Ethereum network and running an appropriate client.
- Key requirement is that the node should be fully synced with the main network before mining can start.

CPU mining

- **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 mine switch in order to start mining:
  - **geth --mine --minerthreads <n>**
- CPU mining can also be started using the web 3 geth console.
- 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**
  - The preceding command will start the miner with four threads.
- **Miner.stop**
  - **True**
  - The preceding command will stop the miner.

GPU mining

- 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 the farm mode to contribute to mining pools.
- It can be downloaded from h t t p s ://g i t h u b . c o m /G e n oi l /c p p - e t h e r e u m /t r e e /m a s t e r /r e l e a s e s :
  - Ethminer -G

GPU mining

- GPU mining requires an AMD or Nvidia graphics card and an applicable OpenCL SDK.
  - For NVidia chipset, it can downloaded from h t t p s ://d e v e l o p e r . n v i d i a . c o m /c u d a - d o w n l o a ds .
  - For AMD chipsets, it is available at h t t p ://d e v e l o p e r . a m d . c o m /t o o l s - a n d - s d k s /o p e n cl - z o n e /a m d - a c c e l e r a t e d - p a r a l l e l - p r o c e s s i n g - a p p - s d k
- **CPU benchmarking**
  - **$ ethminer -M -C**
- **GPU benchmarking**
  - **$ ethminer -M -G**

The following screenshot example is shown for 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
```

CPU benchmarking

GPU mining

- 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.
  - In order to discourage the development of specialized hardware for mining.

Mining rigs

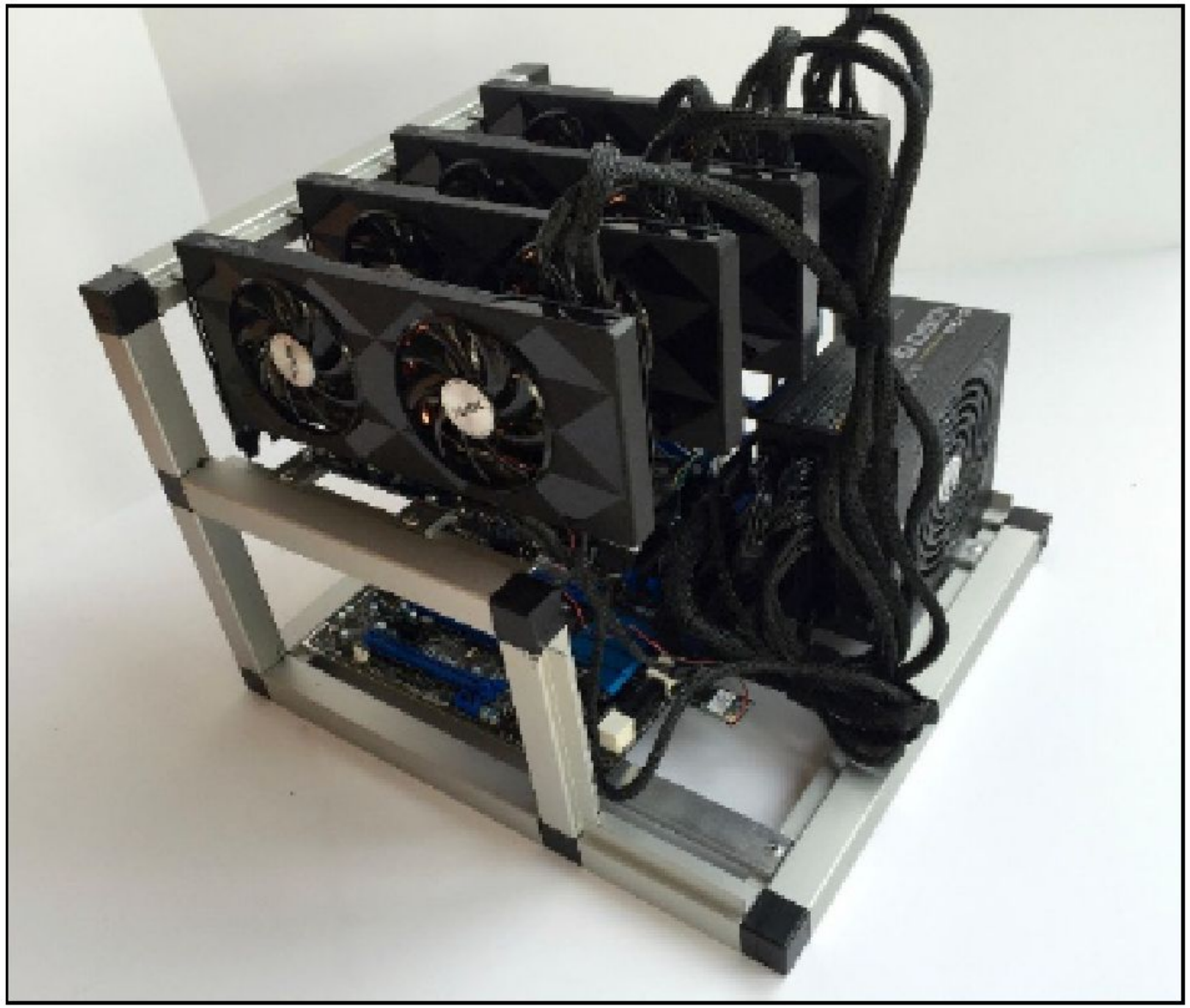
- As difficulty increased over time for mining Ether,
  - Mining rigs with multiple GPUs were starting to be built by the miners.
- Mining rig usually contains around five GPU cards, and all of them work in parallel for mining
  - thus improving the chances of finding valid nonces for mining.
- Mining rigs can be built with some effort and are also available commercially from various vendors.

Typical mining rig configuration

- **Motherboard**
  - Specialized motherboard with multiple PCI-E x1 or x16 slots,
- **SSD hard drive**
  - SSD drive is recommended because of its much faster performance over the analog equivalent.
  - mainly used to store the blockchain
- **GPU**
  - GPU is the most important component of the rig as it is the main workhorse that will be used for mining.
  - For example, it can be a Sapphire AMD Radeon R9 380 with 4 GB RAM.

Typical mining rig configuration

- Linux Ubuntu's latest version is usually chosen as the operating system for the rig.
  - There is also another variant of Linux available, called EthOS
- Mining software such as Ethminer and geth are installed.
- Remote monitoring and administration software is also installed so that rigs can be monitored and managed remotely,
- Air conditioning or cooling mechanisms in place as running multiple GPUs can generate a lot of heat



A mining rig for Ethereum for sale at eBay

Mining pools

- Online mining pools that offer Ethereum mining.
- Ethminer can be used to connect to a mining pool using the following command.
  - ethminer -C –F http://ethereumpool.co/?miner=0.1@0x024a20cc5feba7f3dc3776075b3e60c20eb1459c@DrEquinox
- Each pool publishes its own instructions, but generally, the process of connecting to a pool is similar.

Clients and wallets

- **Geth**
  - Go implementation of the Ethereum client.
- **Eth**
  - C++ implementation of the Ethereum client.
- **Pyethapp**
  - Python implementation of the Ethereum client.
- **Parity**
  - Built using Rust and developed by EthCore.

Light clients

- 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.
- SPV clients validate the execution of transactions.
- SPV clients are also called light clients.
- There is a wallet available from Jaxx (h t t p s ://j a x x . i o /),
  - which can be installed on iOS and Android, which provides the **SPV** (**Simple Payment Verification**) functionality.

Installation

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
```

- Geth can be launched simply by issuing the geth command at the command prompt
  - as it comes preconfigured with all the required parameters to connect to the live Ethereum network (mainnet):
- **> geth**

Installation

Accounts can also be added via the command line using the geth or parity command-line interface. This process is shown in the next section.

Geth

```
$ geth account new
Your new account is locked with a password. Please give a password. Do not
forget this password.
Passphrase:
Repeat passphrase:
Address: {21c2b52e18353a2cc8223322b33559c1d900c85d}
drequinox@drequinox-OP7010:~$
```

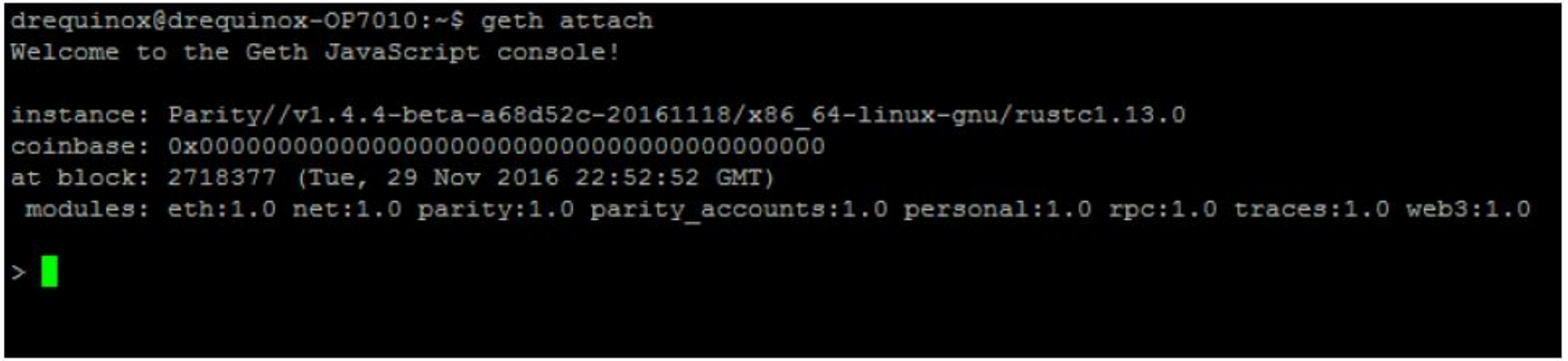
The list of accounts can be shown using geth using the following command:

```
$ geth account list

Account #0: {11bcc1d0b56c57aefc3b52d37e7d6c2c90b8ec35}
/home/drequinox/.ethereum/keystore/UTC--2016-05-07T13-04-15.175558799Z-
-11bcc1d0b56c57aefc3b52d37e7d6c2c90b8ec35

Account #1: {e49668b7ffbf031bbbdab7a222bdb38e7e3e1b63}
/home/drequinox/.ethereum/keystore/UTC--2016-05-10T19-16-11.952722205Z--
e49668b7ffbf031bbbdab7a222bdb38e7e3e1b63

Account #2: {21c2b52e18353a2cc8223322b33559c1d900c85d}
/home/drequinox/.ethereum/keystore/UTC--2016-11-29T22-48-09.825971090Z-
-21c2b52e18353a2cc8223322b33559c1d900c85d
```

The geth console

- geth JavaScript console can be used to perform various functions

Geth can be attached with the running daemon, as shown in the following figure:



Once geth is successfully attached with the running instance of the ethereum client (in this case, parity), it will display command prompt '>', which provides an interactive command line interface to interact with the ethereum client using JavaScript notations.

The yellow paper

- The Ethereum yellow paper has been written by *Dr. Gavin Wood*
  - serves as a formal definition of the Ethereum protocol.
- Anyone can implement an Ethereum client by following the protocol specifications defined in the paper.
- Abstract
  - The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, with Bitcoin being one of the most notable ones. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state. Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

Useful symbols

Symbol	Meaning
≡	Is defined as
=	Is equal to
≠	Is not equal to
‖ … ‖	Length of
∈	Is an element of
∉	Is not an element of
∀	For all
∪	Union
∧	Logical AND

Useful symbols

:	Such that
{}	Set
()	Function of tuple
[]	Array indexing
∨	Logical OR
>	Is greater than
+	Addition
–	Subtraction
Σ	Summation
{	Describing various cases of if , otherwise
⌊...⌋	Floor, lowest element

Useful symbols

⌈...⌉	Ceiling, highest element	
|...|	No of bytes	
⊕	Exclusive OR	
(a,b)	Real numbers >= a and < b	
∅	Empty set, null	

Useful symbols

Symbol	Meaning
$\leq$	Less than or equal to
σ	Sigma, World state
μ	Mu, Machine state
Υ	Upsilon, Ethereum state transition function
Π	Block level state transition function
.	Sequence concatenation
$\exists$	There exists
$\wedge$	Contract creation function
Δ	Increment

The Ethereum network

- 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 homestead.

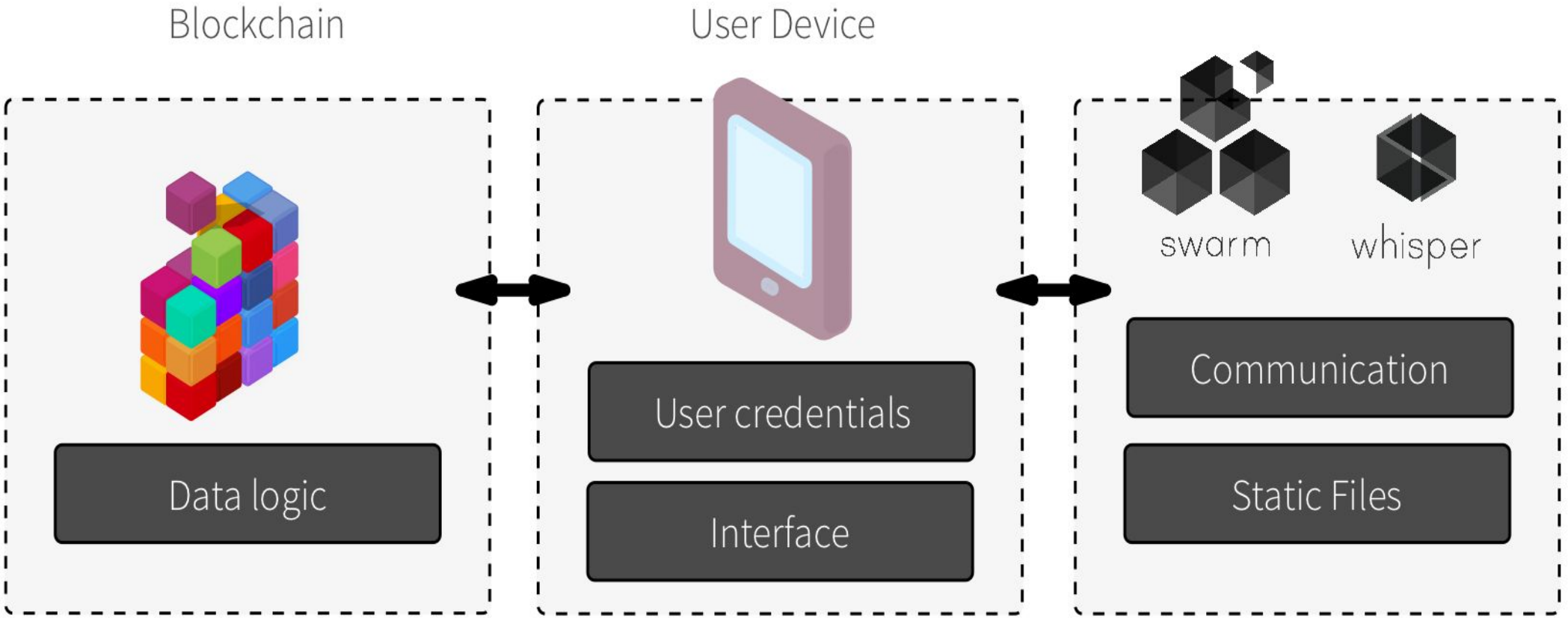
Latest release Muir Glacier / 1 January 2020

The Ethereum network

- **TestNet**
  - TestNet is also called Ropsten and is the test network for the Ethereum blockchain.
  - Blockchain is used to test smart contracts and DApps before being deployed to the production live blockchain.
  - It allows experimentation and research.
- **Private net(s)**
  - private network that can be created by generating a new genesis block.
  - This is usually the case in distributed ledger networks, where a private group of entities start their own blockchain and use it as a permissioned blockchain.

Supporting protocols

- Various supporting protocols are development in order to support the complete decentralized ecosystem.
  - whisper and Swarm protocols.
- In addition to the contracts layer, which is the core blockchain layer, there are **additional layers that need to be decentralized** in order to achieve a complete decentralized ecosystem.
  - This includes decentralized storage and decentralized messaging.
- **Whisper**, being developed for ethereum, is a decentralized messaging protocol,
- **Swarm** is a decentralized storage protocol.


Blockchain
User Device
swarm
whisper
Data logic
User credentials
Interface
Communication
Static Files

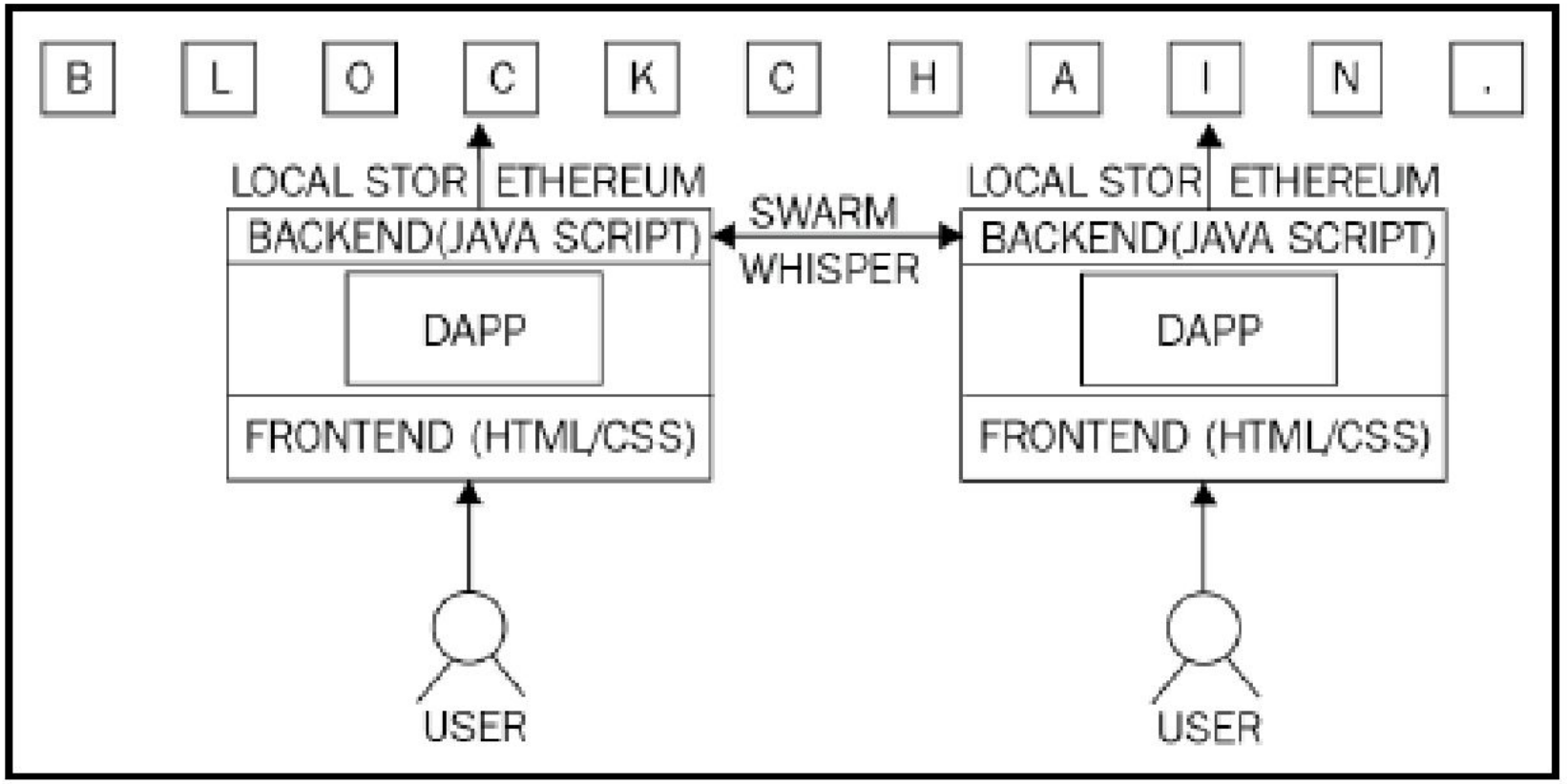
Whisper

- Whisper provides decentralized peer-to-peer messaging capabilities to the Ethereum network.
- **whisper is a communication protocol** that nodes use in order to communicate with each other.
- Data and routing of messages are encrypted within whisper communications.
  - used for smaller data transfers and in scenarios where real-time communication is not required.
- Whisper is also designed to provide a communication layer that cannot be traced and provides "**dark communication**" between parties.
- Blockchain can be used for communication, but that is expensive
  - **consensus is not really required for messages exchanged between nodes.**
    - whisper can be used as a protocol that allows
- Whisper is available with geth

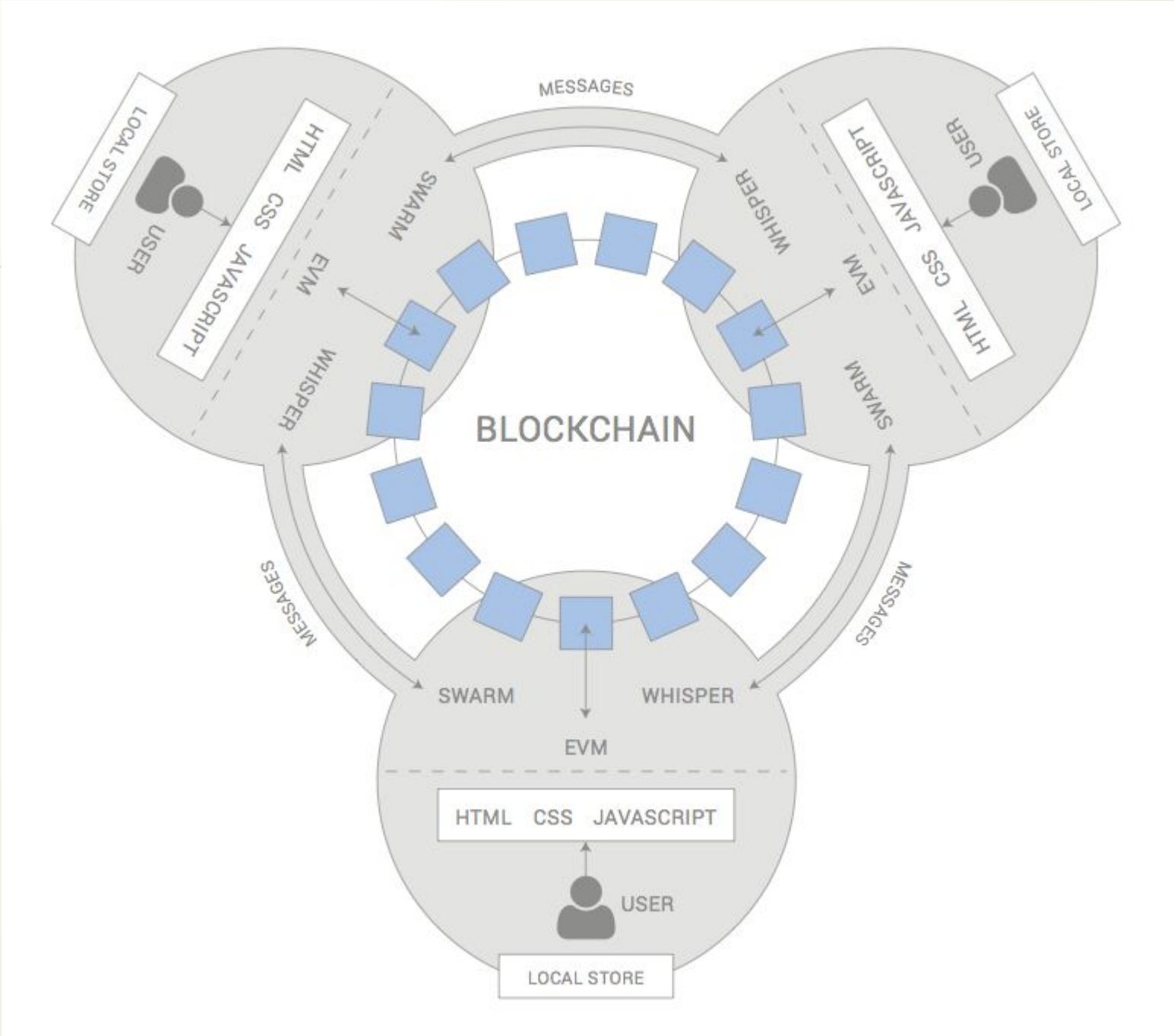
Swarm

- Swarm is being developed as a **distributed file storage platform.**
  - decentralized, distributed, and peer-to-peer storage network.
- Files in this network are addressed by the hash of their content.
  - This is in contrast to the traditional centralized services, where storage is available at a central location only.
- Developed as a **native base layer service** for the Ethereum web 3.0 stack.
- Swarm is integrated with DevP2P, which is the multiprotocol network layer of Ethereum.
- Swarm is envisaged to provide a **DDOS (Distributed Denial of service)-resistant** and fault-tolerant distributed storage layer for Ethereum Web 3.0.

High level overview of Swarm and whisper



Diagrams shows blockchain, whisper and Swarm


MESSAGES
LOCAL STORE
USER
HTML CSS JAVASCRIPT
SWARM
EVM
WHISPER
WHISPER
EVM
SWARM
JAVASCRIPT CSS HTML
USER
LOCAL STORE
BLOCKCHAIN
MESSAGES
MESSAGES
SWARM
WHISPER
EVM
HTML CSS JAVASCRIPT
USER
LOCAL STORE

Applications developed on Ethereum

- There are various implementations of DAOs and smart contracts in Ethereum,
  - *DAO*, which was recently hacked and required a hard fork in order for funds to be recovered.
- DAO was created to serve as a decentralized platform to collect and distribute investments.
- Augur is another DAPP that has been implemented on Ethereum, which is a decentralized prediction market

Scalability and security issues

- Scalability in any blockchain is a fundamental issue.
  - Network has a maximum capacity of 15 transactions per second, if the status quo remains, the industry's infrastructure will be unable to cope.
  - Most people in the crypto world agree that frameworks and scalability need to be addressed, coming up with solutions takes time and a lot of effort.
    - Any proposal has to have the support of miners, developers, businesses and other stakeholders before it can be enforced
    - Process which can take months and, even then, end in disagreement.
- Security is also of paramount importance.
  - Issues such as privacy and confidentiality have caused some adaptability issues, especially in the financial sector.

ON-CHAIN SCALING

- Sharding
  - Sharding is the process of splitting a database horizontally to spread the load.
  - sharding will reduce network congestion and increase transactions per second by creating new chains, known as “shards.”
  - This will also lighten the load for each validator who will no longer be required to process the entirety of all transactions across the network.

OFF-CHAIN SCALING

- Layer 2 Scaling
  - Layer 2 solutions are centered around a server or cluster of servers, each of which may be referred to as a node, validator, operator, sequencer, block producer, or similar term.
  - Transactions are submitted to these layer 2 nodes instead of being submitted directly to layer 1 (mainnet);
  - Layer 2 instance then batches them into groups before anchoring them to layer 1, after which they are secured by layer 1 and cannot be altered.

OFF-CHAIN SCALING

- State Channels
  - State channels utilize multisig contracts to enable participants to transact quickly and freely off-chain, then settle finality with mainnet.
  - Minimizes network congestion, fees, and delays.
  - Two types of channels are currently state channels and payment channels.

OFF-CHAIN SCALING

- Sidechains
  - Sidechain is an independent EVM-compatible blockchain which runs in parallel to mainnet.
  - Compatible with Ethereum via two-way bridges, and run under their own chosen rules of consensus, and block parameters
- Plasma
  - Plasma chain is a separate blockchain that is anchored to the main Ethereum chain

WHY ARE SO MANY SCALING SOLUTIONS NEEDED?

- Multiple solutions can help to reduce the overall congestion on any one part of the network, and also prevents single points of failure.
- Different solutions can exist and work in harmony, allowing for an exponential effect on future transaction speed and throughput.
- Not all solutions require utilizing the Ethereum consensus algorithm directly, and alternatives can offer benefits that would otherwise be difficult to obtain

References

- Imran Bashir. “Mastring BlockChain”, Packt
- Web Materials