Ethereum

Limitations of Bitcoin

UTXO contains (hash of) ScriptPK

a simple script: indicates conditions when UTXO can be spent

- Unable to determine from spent UTXo how many other UTXO were spent during a time/that day

Limitations:

- Difficult to maintain state in multi-stage contracts
- Difficult to enforce global rules on assets

A simple example: rate limiting My wallet manages 100 UTXOs.

- Desired policy: can only transfer 2BTC per day out of my wallet

An example: NameCoin(NMC)

Namecoin: is a blockchain system and cryptocurrency which allows users to protect domain name servers (DNS) by embedding them on a distributed ledger

Namecoin (NMC) is a cryptocurrency that represents ownership of domain names

NMC: a peer-to-peer naming system based on Bitcoin

Domain name system on the blockchain: [google.com → IP addr]

Need support for three operations:

- **Name.new**(OwnerAddr, DomainName): intent to register
- **Name.update**(DomainName, newVal, newOwner, OwnerSig)
- **Name.lookup**(DomainName)

In NMC token holders can claim a specific name for their websites.

Note: also need to ensure no front-running on **Name.new**()

A broken implementation

Name.new() and Name.upate() create a UTXO with ScriptPK:

DUP HASH256 <OwnerAddr> **EQVERIFY CHECKSIG**
VERIFY <NameCoin> <DomainName> <IPaddr> <1>

only owner can "spend" this UTXO to update domain data

Contract: (should be enforced by miners)

if domain google.com is registered,
no one else can register that domain

Problem: this contract cannot be enforced using Bitcoin script

What to do?

NameCoin: fork of Bitcoin that implements this contract

A fork is a change in a blockchain that makes the new branch noncompatible to the original branch.

Namecoin was developed as a blockchain and token for a decentralized domain name system (DNS).

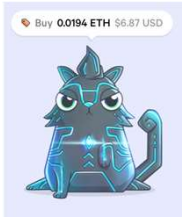
Can we build a blockchain that natively supports generic contracts like this?

⇒ Ethereum

Ethereum: enables many applications

over 3000 Ethereum Decentralized Apps (DAPPs) built on its network

- New coins: ERC-20 interface to DAPP
- DeFi: exchanges, lending, stablecoins, derivatives, etc.
- Insurance
- Games: assets managed on chain (e.g. CryptoKitties)
- Managing distinguished assets (ERC-821)

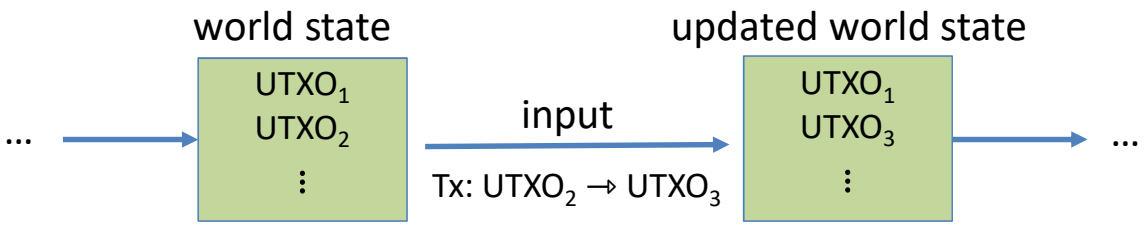


stateofthedapps.com, dapp.review

ERC-20

- Ethereum Request for Comment 20 **(ERC-20)**
  - protocol standard for creating Ethereum-based tokens, which can be utilized and deployed in the Ethereum network.
  - It is **the implemented standard for fungible tokens created using the Ethereum blockchain**

Bitcoin as a state transition system

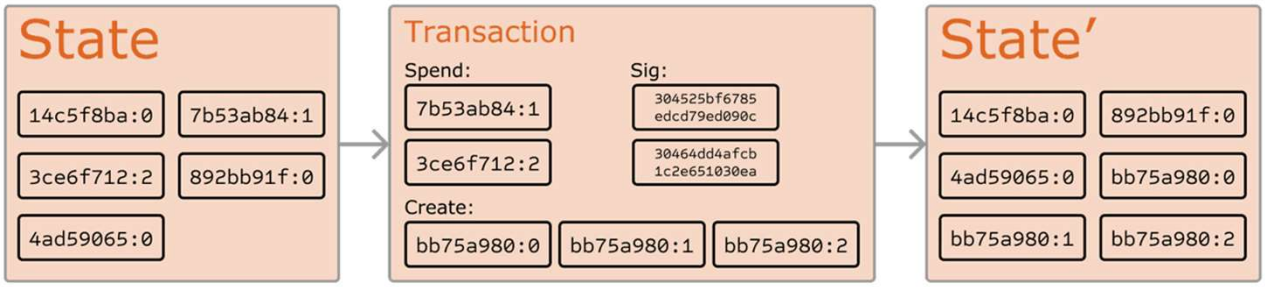


Bitcoin rules: $F_{bitcoin} : S \times I \rightarrow S$

S: set of all possible world states, $s_0 \in S$ genesis state
I: set of all possible inputs

Bitcoin as a State Transition System

- As a technical standpoint
  - ledger of a cryptocurrency such as Bitcoin can be thought of as a state transition system, where there is a "state" consisting of the ownership status of all existing bitcoins and
  - a "state transition function" that takes a state and a transaction and outputs a new state which is the result



Ethereum as a state transition system

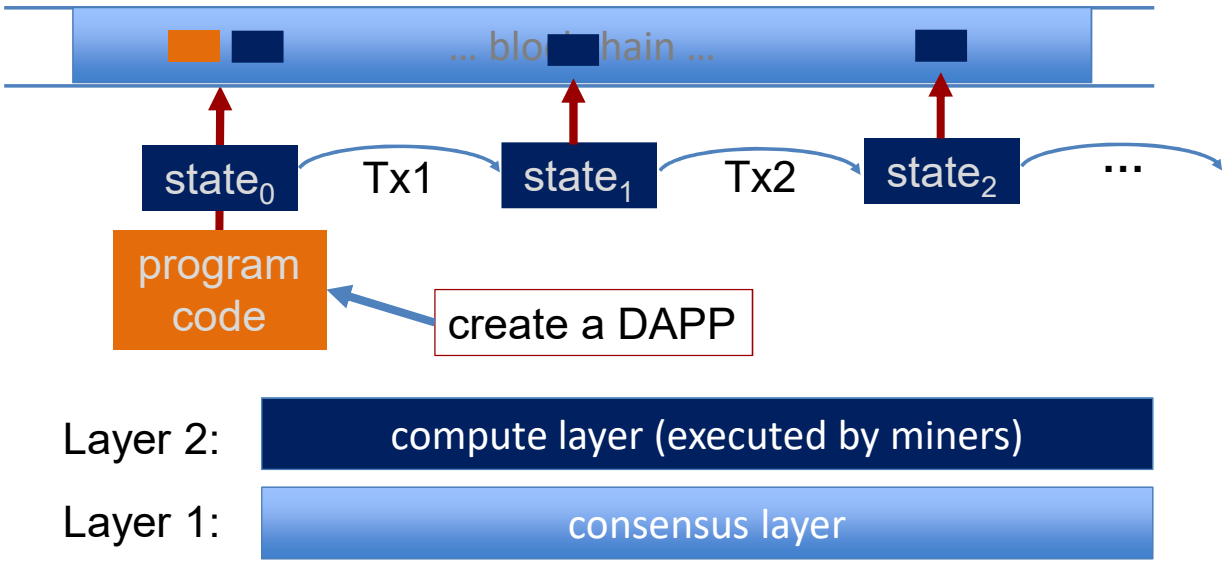
Much richer state transition functions

⇒ one transition executes an entire program

In Ethereum,

- state is made up of objects called "accounts",
- each account having a 20-byte address
- state transitions being direct transfers of value and information between accounts.

Running a program on a blockchain (DAPP)

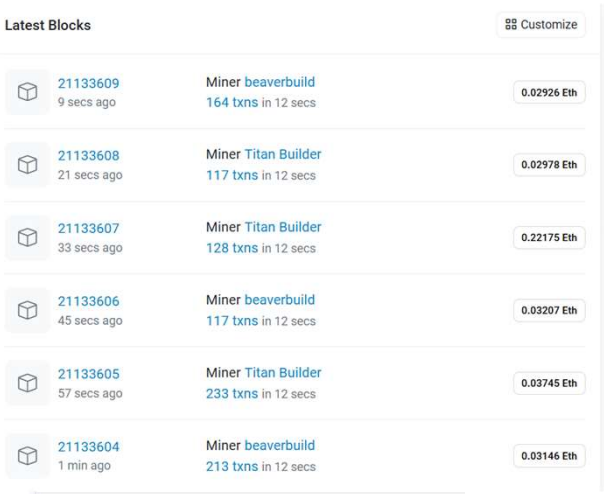


Layer 2: compute layer (executed by miners)

Layer 1: consensus layer

The Ethereum system

Layer 1: PoW consensus. Block reward = 2 ETH + Tx fees (gas)



avg. block rate ~= 12 seconds. (variant of Nakamoto)

about 150 Tx per block.

Ethereum's block time adjustment made in each block

unlike Bitcoin's, which adjusts difficulty after 2,016 blocks

Ethereum Layer 1.5: compute layer

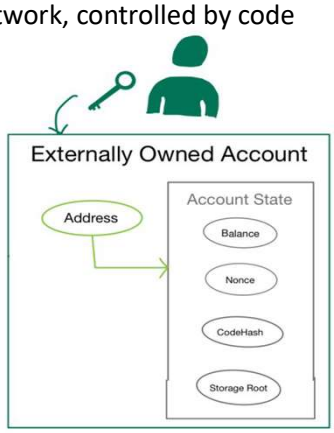
World state: set of **accounts** identified by 160-bit address.

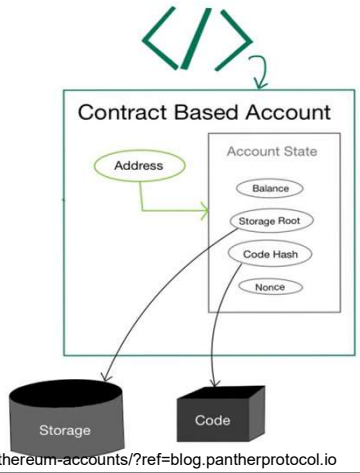
Two types of accounts in Ethereum:

- **Externally owned accounts**: controlled by private keys and with only Ether balance; has a public address and a private key
  - controlled by ECDSA signing key pair (PK,SK). SK known only to account owner
- **Contracts accounts** : that has executable code; controlled by code.
  - code set at account creation time, does not change
  -will define the nature of transactions the contract account can complete.

Ethereum Accounts

- With Owned account you can initiate transactions, send, and receive ETH or any other Ethereum-based token (ERC-20, ERC-721, etc), and interact with smart contracts is an externally owned account (EOA).
- The average Ethereum user owns an EOA, and through wallets, account interacts with blockchain.
- CA is a smart contract deployed to the network, controlled by code
- CAs do not initiate transactions
- CA can only send transactions in response to a transaction received.
- Creating a contract has a cost because of using network storage
- Both account types have the ability to:
  - Receive, hold and send ETH and tokens
  - Interact with deployed smart contracts





source@https://www.geeksforgeeks.org/what-are-ethereum-accounts/?ref=blog.pantherprotocol.io

Ethereum Accounts

An Ethereum account (address) contains four key fields:

- The **nonce**, a counter used to make sure each transaction can only be processed once
  - It tracks the number of transactions an account sends or contracts an account creates, depending on the account type.
- account's current **ether balance** : it is the amount of wei owned by an account.
- account's **contract code (codeHash)**, *if present*
  - the code tied to that account on the Ethereum Virtual Machine;
  - contract accounts are tied in code to the EVM
- The account's **storage (storageRoot)** (empty by default)
  - the hash that encodes the account's storage content.
  - Also called as the storage hash, it is a 256-bit hash of a Merkle Patricia trie's root node.
  - A Patricia Merkle Trie is one of the key data structures for Ethereum's storage layer and provides a cryptographically authenticated data structure that can be used to store all (key, value) bindings.

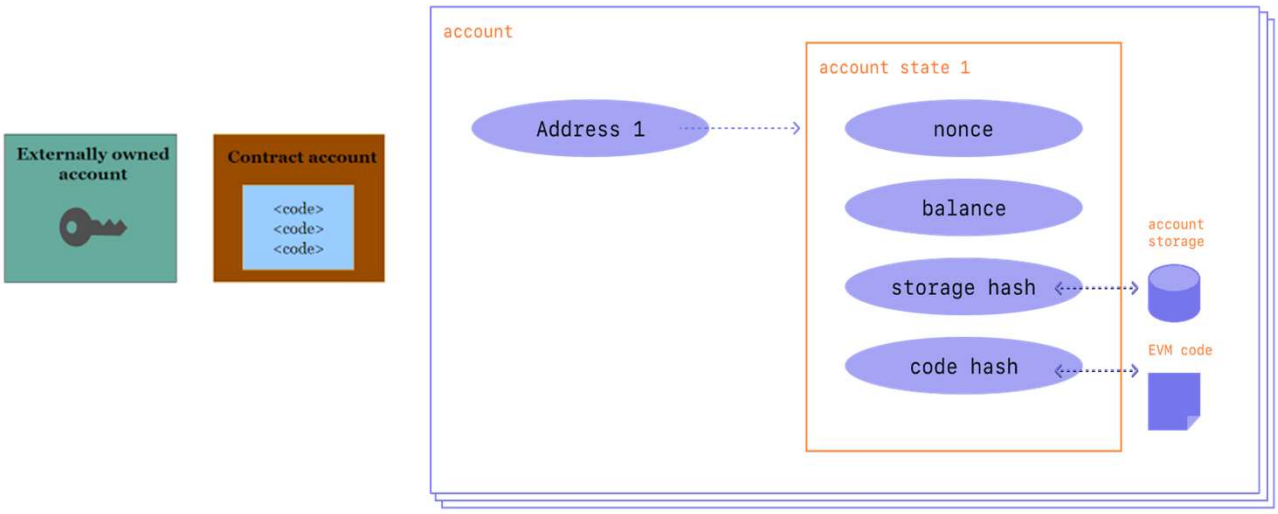
- Create an Ethereum account in order to use Ethereum network. which has a public address and a private key.
- Ethereum accounts and Ethereum wallets are not same
  - Ethereum wallets is only an interface that helps you interact with tokens on the blockchain.

Data associated with an account

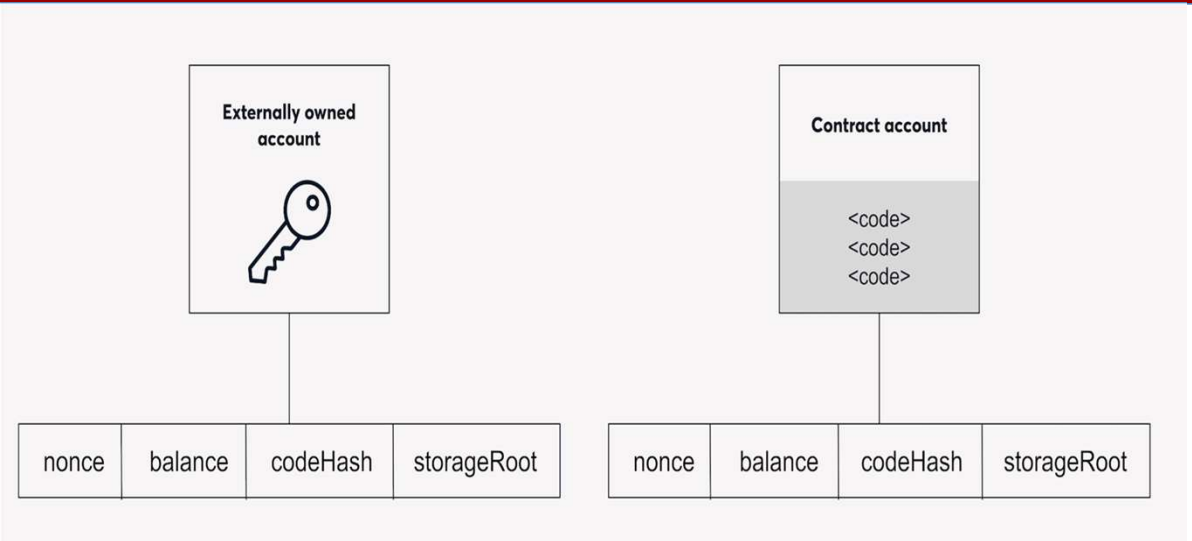
Account data	Owned	Contracts	
address (computed):	H(PK)	H(CreatorAddr, CreatorNonce)	
code:	⊥	CodeHash	
This hash refers to the *code* of an account on the Ethereum virtual machine (EVM).			
storage root (state):	⊥	StorageRoot	
Hash of root node of a Merkle Patricia trie that encodes the storage contents of the account			
balance (in Wei):	balance	balance	(10^{18} Wei = 1 ETH)
nonce:	nonce	nonce	

(#Tx sent) + (#accounts created): anti-replay mechanism

Data associated with an account

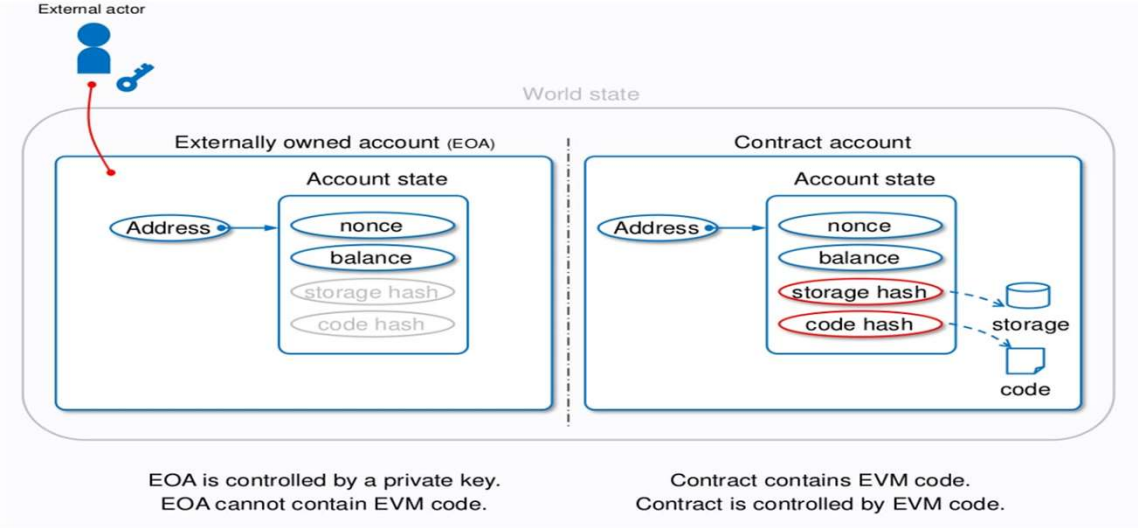


Data associated with an account



Data included in each account type

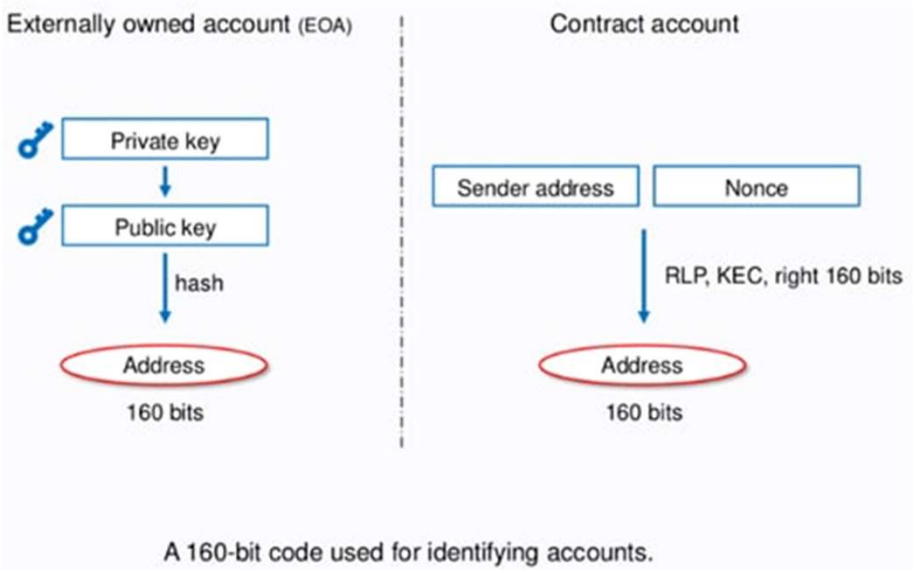
- **Each account consists of balance, nonce, bytecode and stored data (storage)**



Sub-units of Ether

- **wei** is a smallest denomination of ether, Ethereum's native currency
- 1 Ether = 10^{18} wei 1wei =0.000000000000000001 ETH (i.e. 10^-18 ETH)

Unit	wei value	wei	ether value
wei	1 wei	1	10^-18 ETH
kwei	10^3 wei	1,000	10^-15 ETH
mwei	10^6 wei	1,000,000	10^-12 ETH
gwei	10^9 wei	1,000,000,000	10^-9 ETH
microether	10^12 wei	1,000,000,000,000	10^-6 ETH
milliether	10^15 wei	1,000,000,000,000,000	10^-3 ETH
ether	10^18 wei	1,000,000,000,000,000,000	1 ETH



A 160-bit code used for identifying accounts.

Account state: persistent storage

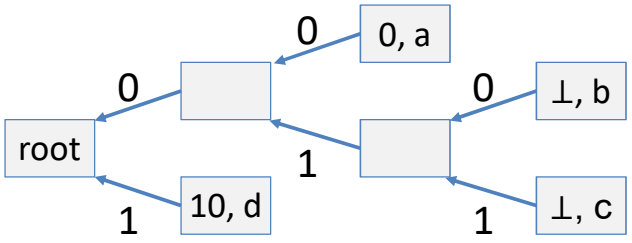
Every contract has an associated **storage array S**[]:

S[0], S[1], … , S[2^{256}-1]: each cell holds 32 bytes, init to 0.

Account storage root: **Merkle Patricia Tree hash** of S[]

- Cannot compute full Merkle tree hash: 2^{256} leaves

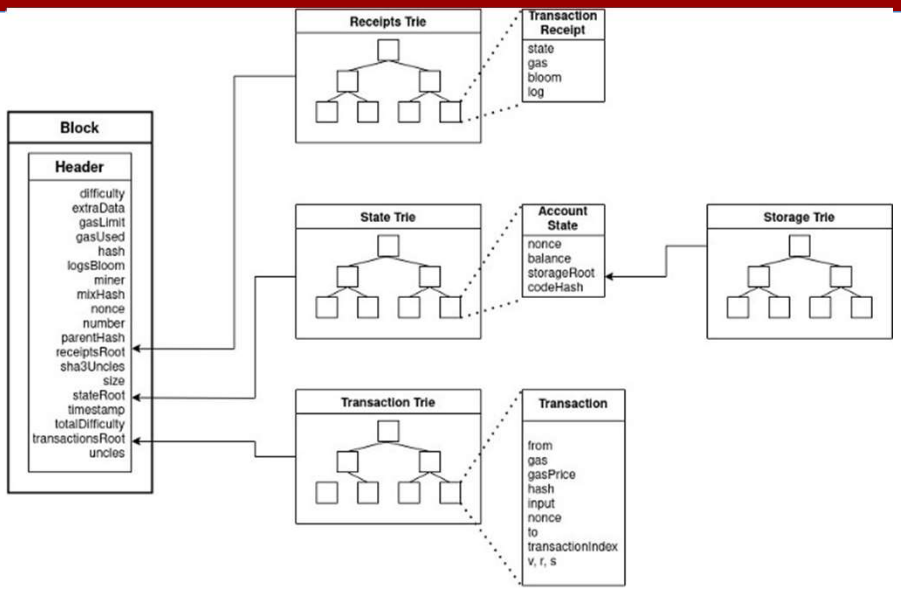
S[000] = a
S[010] = b
S[011] = c
S[110] = d



time to compute root hash:
≤ 2 × |S|

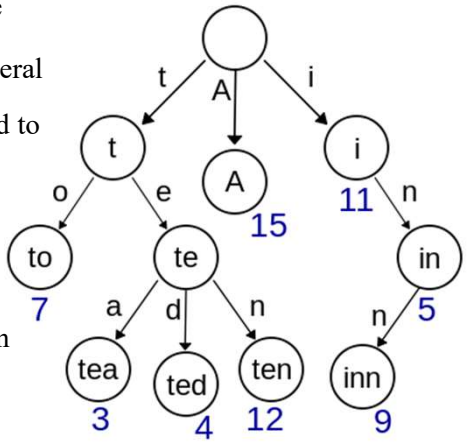
|S| = # non-zero cells

Merkle patricia trie in ethereum



Merkle Patricia Trie

- Ethereum is a state machine where transactions modify states on the Ethereum network.
- A state can be expressed as a key-value pair. Although there are several ways of representing a key-value pair, the Ethereum specification defines the Modified Merkle Patricia Trie (a.k.a MPT) as the method to save states.
- MPT is a combination of Patricia trie and Merkle tree, with few additional optimizations that fit the characteristics of Ethereum.



Patricia trie

- is a data structure which is also called Prefix tree, radix tree or trie.
- Trie uses a key as a path so the nodes that share the same prefix can also share the same path.
- This structure is fastest at finding common prefixes, simple to implement, and requires small memory.
- it is commonly used for implementing routing tables, systems that are used in low specification machines like router.
- But Merkle tree is a tree of hashes. Leaf nodes store data

State transitions: Tx and messages

Transactions: signed data by initiator

- **To:** 32-byte address of target (0 ⇾ create new account)
- **From**, **Signature**: initiator address and signature on Tx
- **Value**: # Wei being sent with Tx
- **gasPrice, gasLimit**: Tx fees
- if To = 0: create new contract **code = (init, body)**
- if To ≠ 0: **data** (what function to call & arguments)
- **nonce**: must match current nonce of sender (prevents Tx replay)

Term	Description
Gas	Measures how much computation is done
Gas price	How much you're willing to pay **per** gas of work
Gas limit	Max gas you're willing to use for a transaction
Tx cost	Gas used * Gas price
Gas block limit	Max gas allowed in a block

- Gas points and gas point limits are to address denial of service attacks and whimsical codes to be deployed on the blockchain.
- Also own code may not have tested and may mal function.
- Ethers/Wei.. Other denominations: For payment towards transaction fees, also as cryptocurrency between accounts

Transaction (stored on blockchain)

- Transaction has a message→ snapshot from Remix for the Ballot register()

status	0x1 Transaction mined and execution succeed
transaction hash	0xef8df4a73cac62d056d54572469b539b99cbf3944a2fb6b62830a4173b54cc11
from	0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to	BallotV3.register(address) 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a
gas	3000000 gas
transaction cost	49000 gas
execution cost	26320 gas
hash	0xef8df4a73cac62d056d54572469b539b99cbf3944a2fb6b62830a4173b54cc11
input	0x442...c160c
decoded input	{ "address voter": "0x14723A09ACff6D2A60DcdF7aA4AFf308FDDC160C" }
decoded output	{}
logs	[]
value	0 wei

transaction object

```
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
  gasLimit: "21000",
  maxFeePerGas: "300",
  maxPriorityFeePerGas: "10",
  nonce: "0",
  value: "10000000000"
}
```

Messages and Transactions

- "transaction" term in Ethereum refer to the signed data package that stores a message to be sent from an externally owned account.

Components of Transactions:

- The recipient of the message
- A signature identifying the sender
- The amount of ether to transfer from the sender to the recipient
- An optional data field
- A STARTGAS value, representing the maximum number of computational steps the transaction execution is allowed to take
- A GASPRICE value, representing the fee the sender pays per computational step

- The data field has no function by default, but the virtual machine has an opcode using which a contract can access the data

Messages

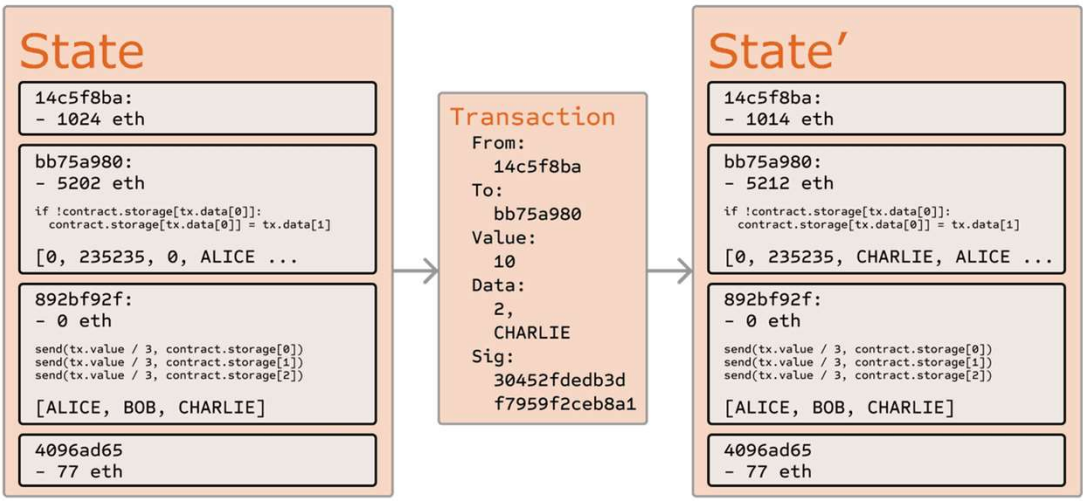
- Contracts have the ability to send "messages" to other contracts.
- Messages are virtual objects that are never serialized and exist only in the Ethereum execution environment.

A message contains:

- The sender of the message (implicit)
- The recipient of the message
- The amount of ether to transfer alongside the message
- An optional data field
- A STARTGAS value

- Essentially, a message is like a transaction, except it is produced by a contract and not an external actor
- A message is produced when a contract currently executing code executes the CALL opcode, which produces and executes a message

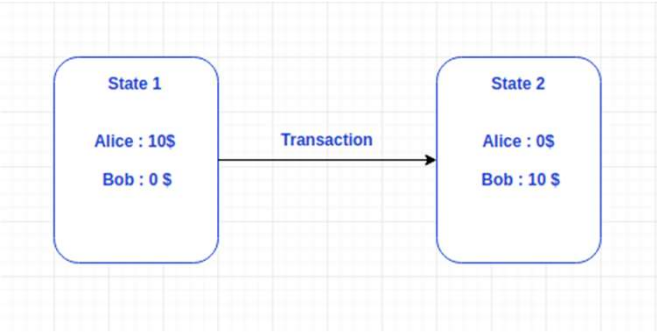
Ethereum State Transition Function



Storage of states and transactions

- Let see what all things we need to store for making the blockchain system work when Alice giving Bob 10$.

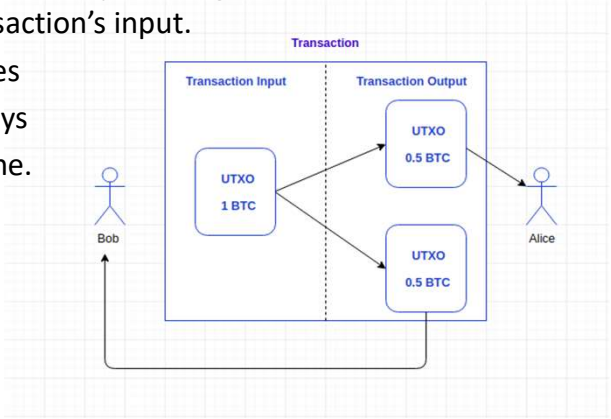
here we can change the state by executing a transaction on it →



- we have to keep track of the balances and other details of different people(**states**) and the details of what happens between them on blockchain(**transactions**).
- Different platforms handle this differently.

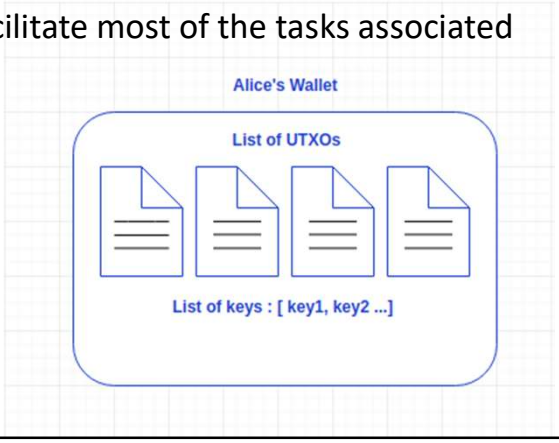
Storage of states and transactions

- The transfer of value in bitcoin is actioned through transactions.
- Bitcoin's "state" is represented by its global collection of Unspent Transaction Outputs (UTXOs).
- a bitcoin user can spend one or more of their UTXOs by creating a transaction and adding one or more of their UTXOs as the transaction's input.
- bitcoin does not maintain user account balances
- With bitcoin, a user simply holds the private keys to one or more UTXO at any given point in time.
- bitcoin wallets hold keys to UTXOs
- bitcoin UTXOs can not be partially spent.



visualization of how wallets work in bitcoin

- *Digital wallets make it seem like the bitcoin blockchain automatically stores and organizes user account balances and so forth.*
- UTXO system in bitcoin works well
  - as digital wallets are able to facilitate most of the tasks associated with transactions.

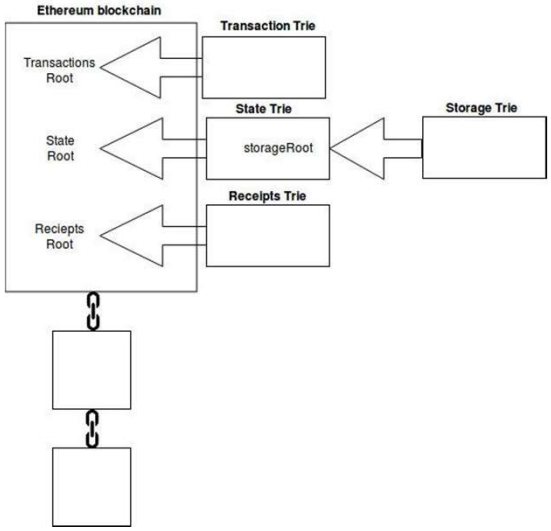


Ethereum world state

- Ethereum world state is able to manage account balances, and more.
- The state of Ethereum is part of Ethereum's base layer protocol.
- Ethereum is a transaction-based "state" machine; a technology on which all transaction-based state machine concepts may be built.
- Ethereum blockchain begins life at its own genesis block.
  - From this point (genesis state at block 0) onward, activities such as transactions, contracts, and mining will continually change the state of the Ethereum blockchain.
- In Ethereum, for example an account balance (stored in the state trie) which changes every time a transaction, in relation to that account, takes place.

Ethereum state

- data such as account balances are not stored directly in the blocks of the Ethereum blockchain.
- Only the root node hashes of the transaction trie, state trie and receipts trie are stored directly in the blockchain.
- the root node hash of the storage trie (where all of the smart contract data is kept) actually points to the state trie, which in turn points to the blockchain.



types of data in Ethereum

1. permanent data

   Example: a mined transaction→ once a transaction fully confirmed, it is recorded in the transaction trie and never altered.

2. ephemeral data

   Example: balance of a particular Ethereum account address → balance stored in the state trie and is altered whenever transactions against that particular account occur

- Both stored separately
- Ethereum uses trie data structures to manage data

State transitions: Tx and messages

Transaction types:

owned ⇀ owned: transfer ETH between users

owned ⇀ contract: call contract with ETH & data

Example (block #10993504)

Explore ethereum on https://etherscan.io/

From		To	msg.value	Tx fee (ETH)
0xa4ec1125ce9428ae5...	→	0x2cebe81fe0dcd220e...	0 Ether	0.00404405
0xba272f30459a119b2...	→	Uniswap V2: Router 2	0.14 Ether	0.00644563
0x4299d864bbda0fe32...	→	Uniswap V2: Router 2	89.839104111882671 Ether	0.00716578
0x4d1317a2a98cfea41...	→	0xc59f33af5f4a7c8647...	14.501 Ether	0.001239
0x29ecaa773f052d14e...	→	CryptoKitties: Core	0 Ether	0.00775543
0x63bb46461696416fa...	→	Uniswap V2: Router 2	0.203036474328481 Ether	0.00766728
0xde70238aef7a35abd...	→	Balancer: ETH/DOUGH...	0 Ether	0.00261582
0x69aca10fe1394d535f...	→	0x837d03aa7fc09b8be...	0 Ether	0.00259936
0xe2f5d180626d29e75...	→	Uniswap V2: Router 2	0 Ether	0.00665809

Messages: virtual Tx initiated by a contract

Same as Tx, but no signature (contract has no signing key)

contract ⇀ owned: contract sends funds to user

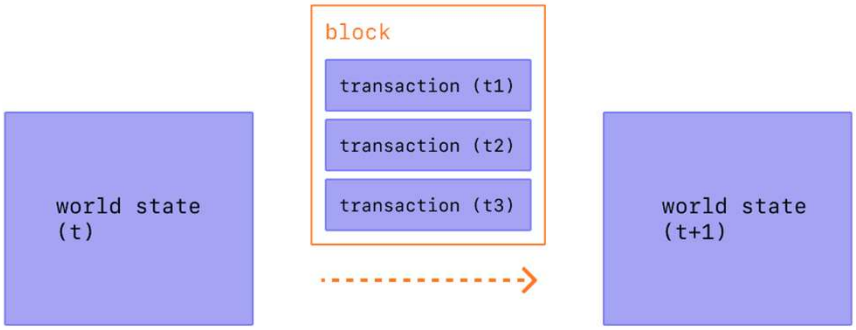
contract ⇀ contract: one program calls another (and sends funds)

One Tx from user: can lead to many Tx processed. Composability!

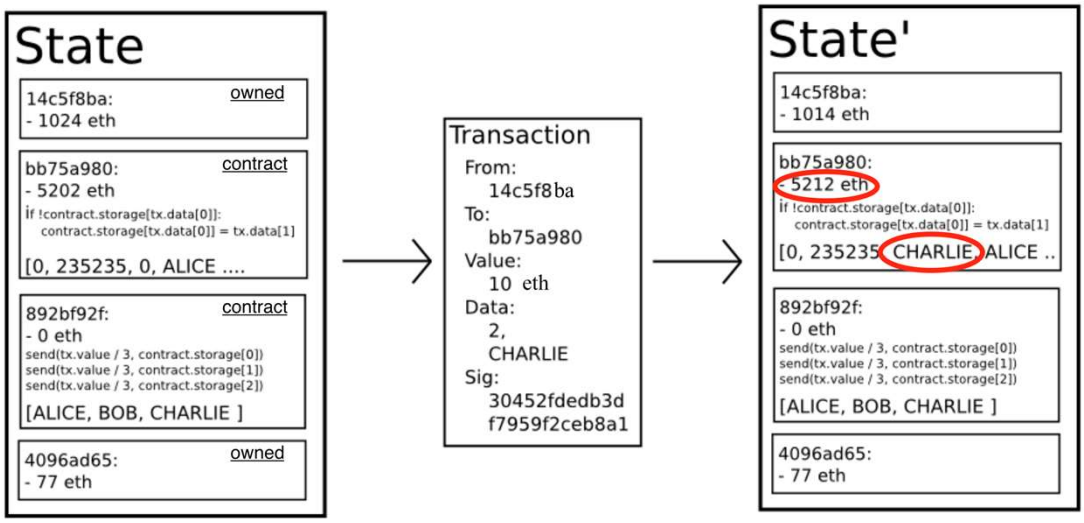
Tx from owned addr ⇀ contract ⇀ another contract

↳ another contract ⇀ different owned

Ethereum world state



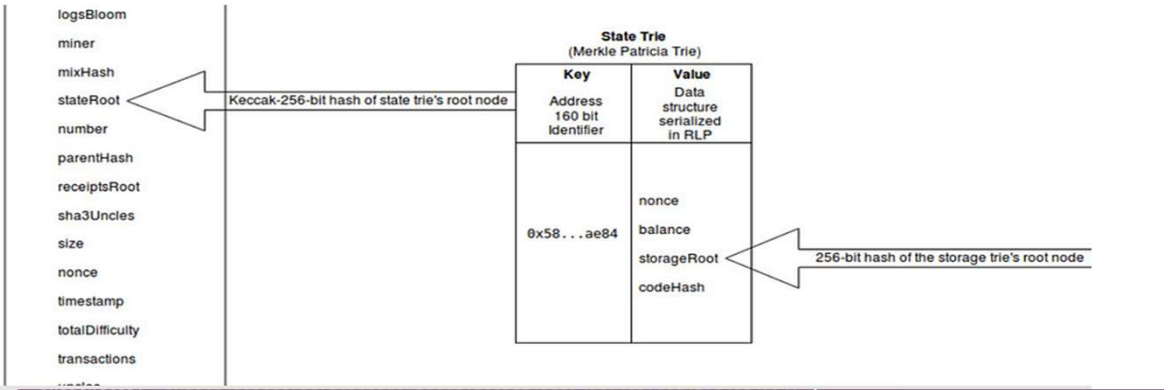
Example Tx



world state (four accounts)

State Trie and an Ethereum block

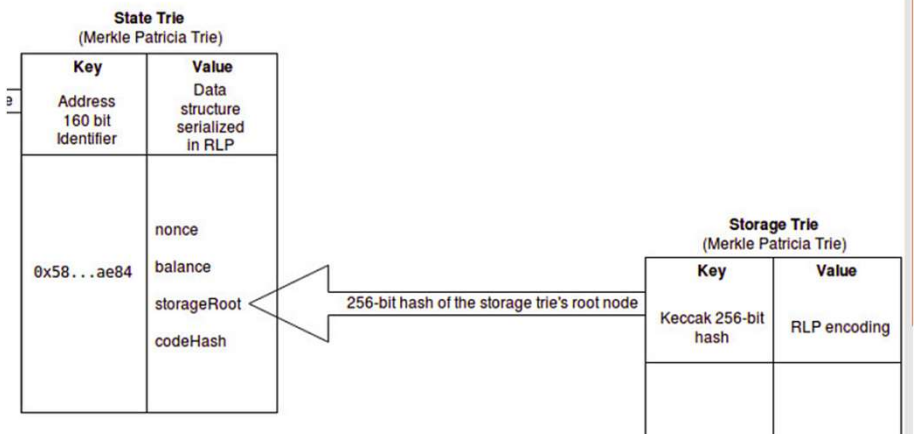
- trie (or tree) is a well known data structure which is used for storing sequences of characters
- There is one, and one only, global state trie in Ethereum.




```
size: 533,
stateRoot: "0x8c77785e3e9171715dd34117b047dffe44575c32ede59bde39fbf5dc074f2976",
timestamp: 1517107395,
totalDifficulty: 3021121,
transactions: [],
transactionsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
uncles: []
```

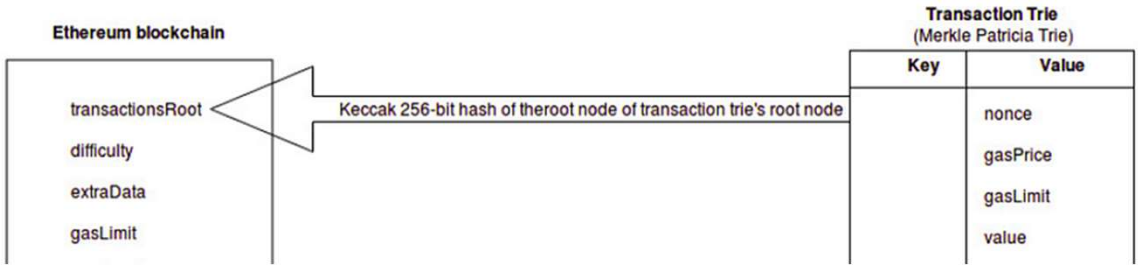

Storage trie

- where all of the contract data lives
- Each Ethereum account has its own storage trie. A 256-bit hash of the storage trie's root node is stored as the storageRoot value in the global state trie



Transaction trie

- Each Ethereum block has its own separate transaction trie. A block contains many transactions.
- The order of the transactions in a block decided by the miner who assembles the block
- Mined blocks are never updated; the position of the transaction in a block is never changed.
  - it means that once you locate a transaction in a block's transaction trie, you can return to the same path over and over to retrieve the same result.



- Ethereum clients use two different database software solutions to store their tries. Ethereum's Rust client Parity uses rocksdb. whereas Ethereum's Go, C++ and Python clients all use leveldb.

An Ethereum Block

Miners collect Txs from users ⇒ leader creates a block of n Tx

- Miner does:
  - for i=1,…,n: execute state change of Tx_i
    (can change state of >n accounts)
  - record updated world state in block

Other miners re-execute all Tx to verify block

- Miners should only build on a valid block
- Miners are not paid for verifying block (note: verifier's dilemma)

Block header data

(1) consensus data: parent hash, difficulty, PoW solution, etc.

(2) address of gas beneficiary: where Tx fees will go

(3) world state root: updated world state

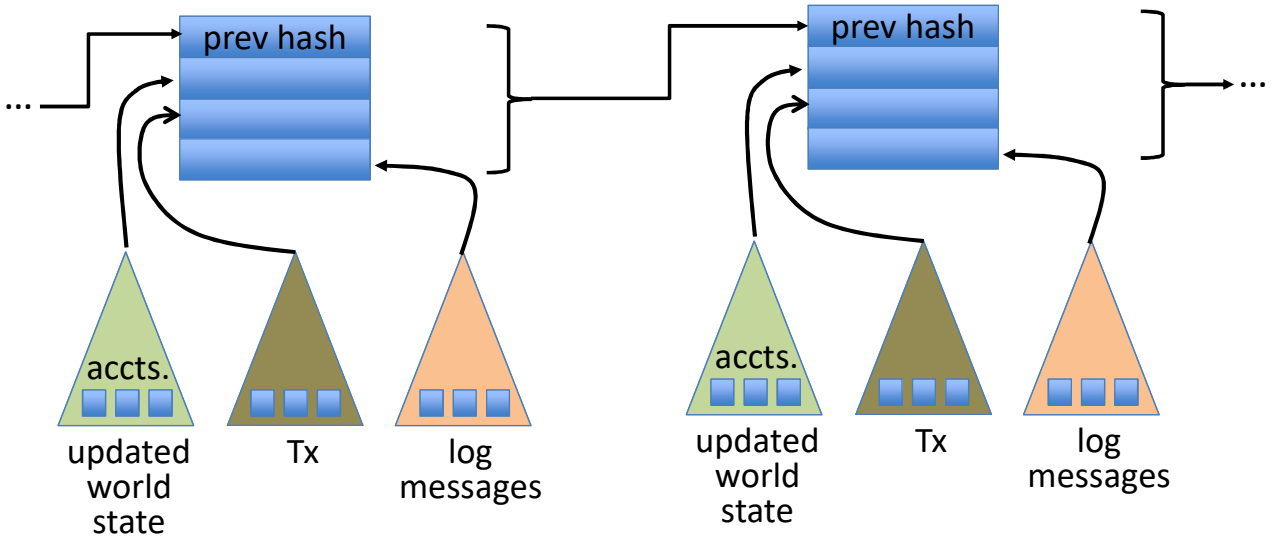
Merkle Patricia Tree hash of <u>all</u> accounts in the system

(4) **Tx root**: Merkle hash of all Tx processed in block

(5) **Tx receipt root**: Merkle hash of log messages generated in block

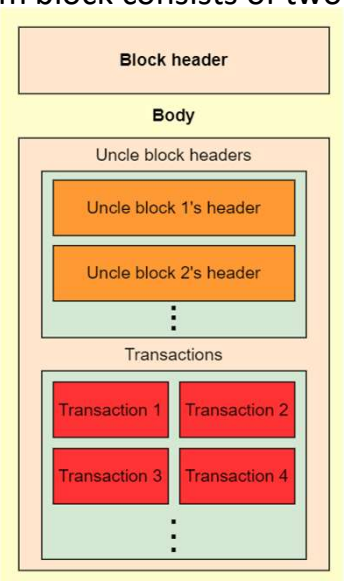
(5) Gas used: tells verifier how much work to verify block

The Ethereum blockchain: abstractly



Structure of blocks

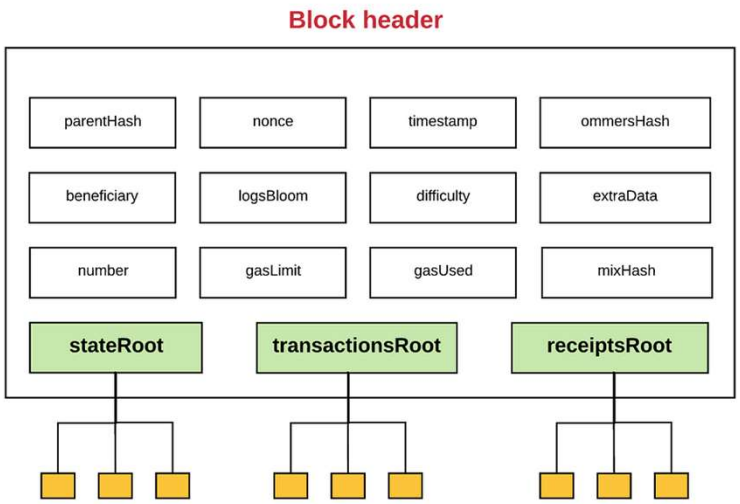
- Every Ethereum block consists of two main parts:
  - Header
  - Body

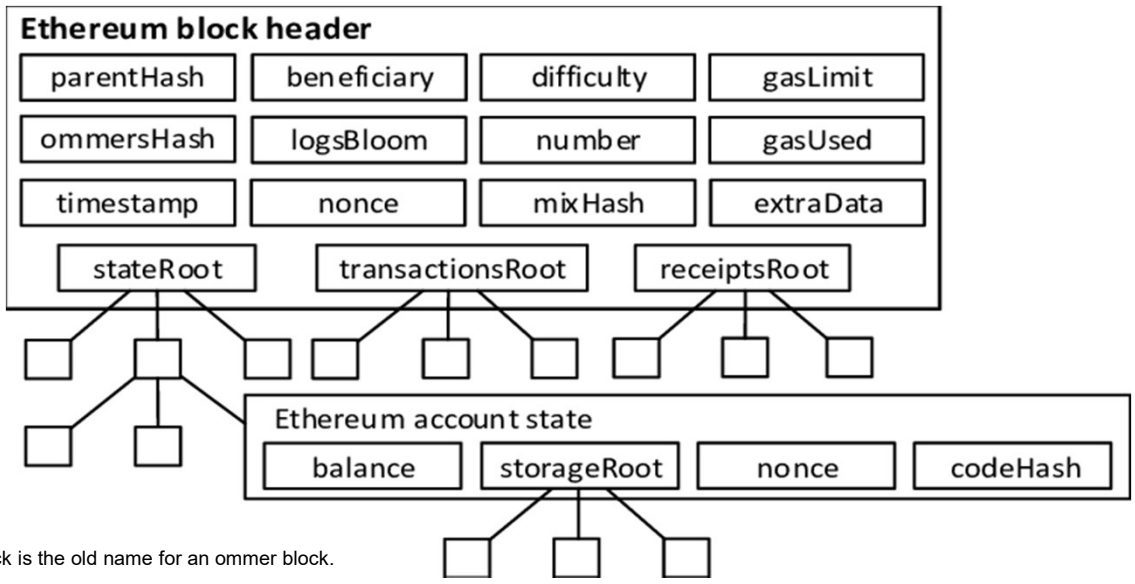


Ethereum block → header structure

An Ethereum Block Header

- *A **trie** is a data structure that efficiently **stores data** for quick retrieval. **The state trie hence** has info about the **state of transactions** in the block without having to look at the transactions*
- Ommer blocks are created in the Ethereum blockchain when two blocks are created and submitted to the ledger at roughly the same time.





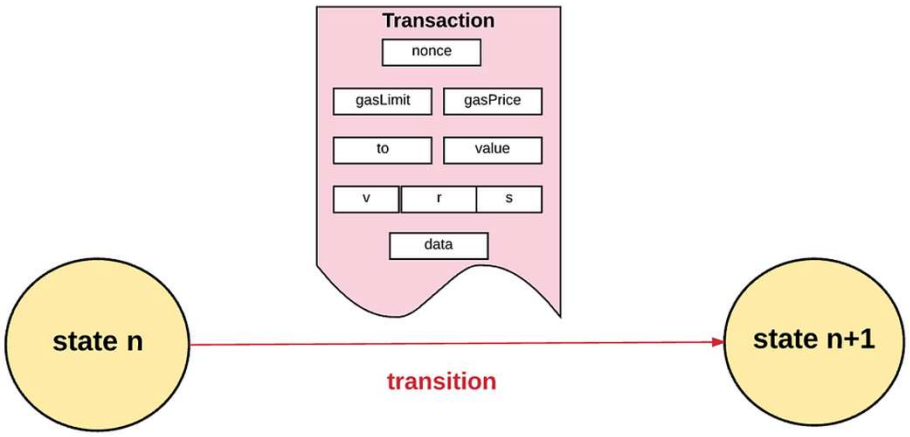
Uncle block is the old name for an ommer block.

uncle blocks

- Due to many miners work on same blocks concurrently, it's common for several miners to solve the hash puzzle at almost the same time.
- In other blockchains, these blocks are discarded as **orphans**.
- In Ethereum, these blocks are called **uncle blocks**.
- Ethereum accepts uncle blocks and even provides a reward to the miner, but one that's smaller than the accepted block.
- **Uncle rewards provide an incentive for smaller miners to participate. Otherwise, mining would be profitable only for large pools that could eventually take over all mining.**

An Ethereum Block Body — Transaction Set

- An Ethereum Block Body has a set of transactions — each with a unique set of properties



Transaction Set

*For transfers, the **To** address is the **account** that will receive the transfer. For calling functions, the **To** address is the address of the **smart contract**.*

- **Nonce**: Each Ethereum account keeps track of the number of transactions it executes. This field is the **latest transaction**, based on the account's counter. The transaction nonce is used by the network to ensure that transactions are executed in the proper order.
- **Signature**: The **digital signature of the account owner.**
- **Gas price**: The unit **price you're willing to pay** to execute this transaction.
- **Gas limit**: The **max amount you're willing to pay** to execute this transaction.
- **To**: The **recipient address** of this transaction.
- **Value**: The total amount of ether you want to send to the recipient.
- **Data**: The actual data submitted as the transaction body. **For calling functions (Smart Contracts), the data might contain parameters.**

block's header fields

Every block header contains the following fields:

Parent block's hash: It is Keccak 256-bit hash of its parent block's header. used to chain this block to previous block making blockchain more secure.

Uncle block's hash: It is the Keccak 256-bit hash of the list of uncle block headers present in the block's body.

Beneficiary address: It is the 160-bit account address of the miner of this block to which all the mining fees from this block will be transferred.

State root: It is the Keccak 256-bit hash of the root node of the state trie after all the transactions are executed and finalizations are applied.

Transaction root: It is the Keccak 256-bit hash of the root node of Merkle root trie populated by all the transactions present in that block's body.

Receipt root: The Keccak 256-bit hash of the root of the Merkel root trie that is populated by the receipts of all the transactions in that block.

Logs bloom: A Bloom filter consisting of indexable log entries from receipt of each transaction from list of transactions present in body of block.

Block number: The length of the blockchain or the number of blocks in the blockchain. A positive whole number equals the number of ancestor blocks of this block where the genesis block is block 0.

Gas limit: The current upper limit on the gas used in the block.

Gas used: Total gas used in transactions in this block.

Difficulty: It is the difficulty of the network to mine this block. Miners' have to mine a block whose header's hash is less than the network's difficulty at that time. Miners do this by changing the nonce value until they find a value with which the hash of the block is less than the network's difficulty.

Mix hash: unique identifier for block. When combined with nonce proves that a sufficient amount of work has been done by miner (PoW).

Timestamp: The time in Unix format when the block was mined.

Base fee per gas: The minimum fee required per gas for the transaction to be included in the block.

Extra data: A byte array containing data relevant to this block. This must be less than 32 bytes.

body of an Ethereum block

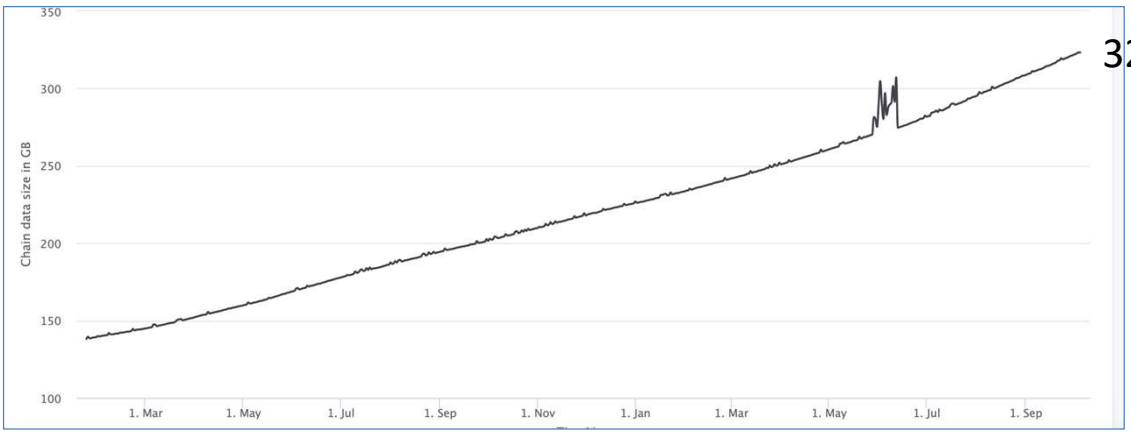
- body of an Ethereum block contains the list of

  1. Uncle block headers: This is a list of uncle block headers (same format as a block header).

  2. Transactions: This is a list of actual Ethereum transactions stored in the block.

- Every miner is trying to mine a block at the same time.
- In the end, only one block will be accepted and other blocks will become **uncle blocks**.
- In Ethereum uncle blocks also get paid.

Amount of memory to run a node (in GB)



323GB

ETH total blockchain size: 5.2 TB (Oct. 2020)

An example contract: NameCoin

```
contract nameCoin {        // Solidity code

    struct nameEntry {
        address owner;     // address of domain owner
        bytes32 value;     // IP address
    }

    // array of all registered domains
    mapping (bytes32 => nameEntry)  data;
```

An example contract: NameCoin

```
function nameNew(bytes32 name) {
    // registration costs is 100 Wei
    if (data[name] == 0   &&   msg.value >= 100) {
        data[name].owner = msg.sender    // record domain owner
        emit Register(msg.sender, name)   // log event
}}
```

Code ensures that no one can take over a registered name

An example contract: NameCoin

```
function nameUpdate(
            bytes32 name, bytes32 newValue, address newOwner) {
  // check if message is from domain owner,
  //          and update cost of 10 Wei is paid
  if (data[name].owner == msg.sender   &&   msg.value >= 10) {
      data[name].value = newValue;          // record new value
      data[name].owner = newOwner;       // record new owner
}}}
```

An example contract: NameCoin

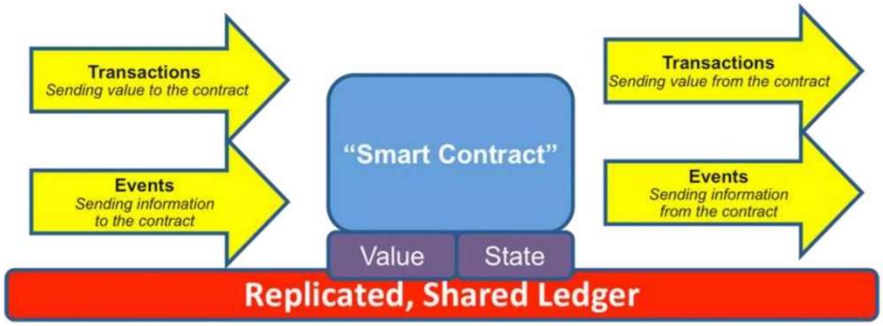
```
    function nameLookup(bytes32 name) {
        return data[name];
    }

} // end of contract
```

In context to Blockchain

Code is law

Smart Contract

- A piece of code i.e. event driven program deployed to the shared replicated ledger which can maintain its own state , control its own assets and which responds to the arrival of external information or receipts of assets

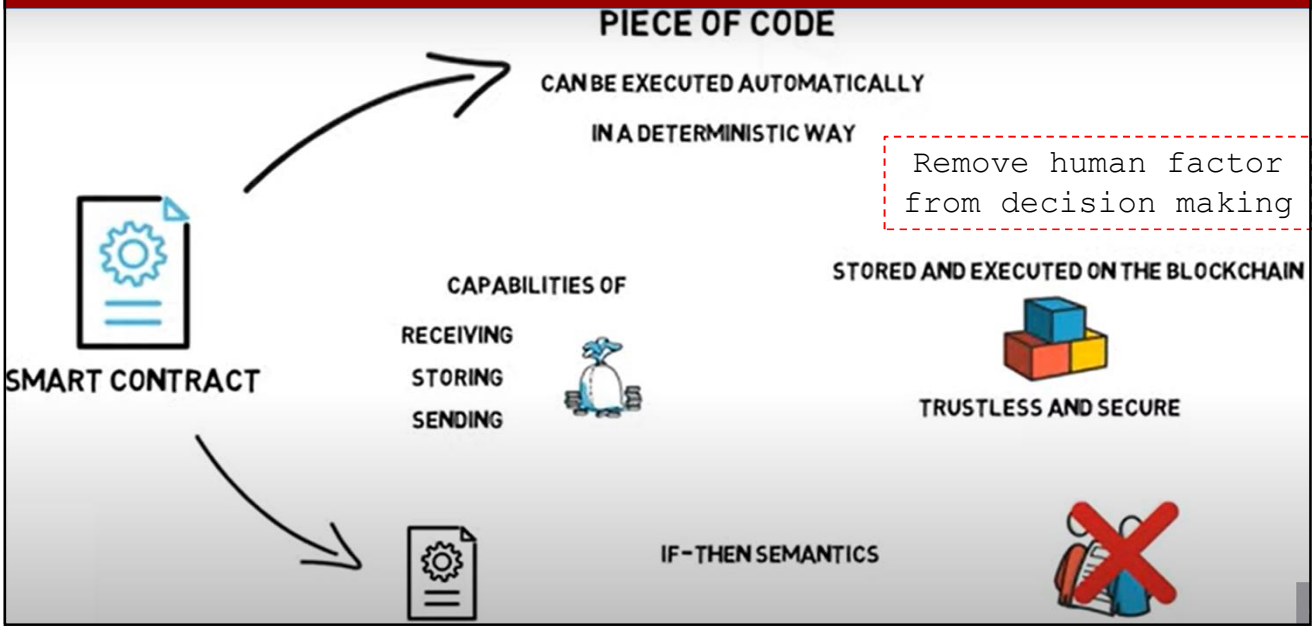


- It's a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain.

Smart contracts

- Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met.
- Smart contracts digitize agreements by turning the terms of an agreement into computer code that automatically executes when the contract terms are met.
- They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss.
- Authority-less autonomous program, that directly controls numeric securities (digital assets), based on mutually agreed terms
- Looks like “if-then” instructions that automatically evaluate predefined conditions and do transactions
- It an owner and a life cycle, and is executed on Ethereum Virtual Machine (EVM)
- They can perform computations, create currency, store data, mint NFTs, send communications and even generate graphics.

Smart contract



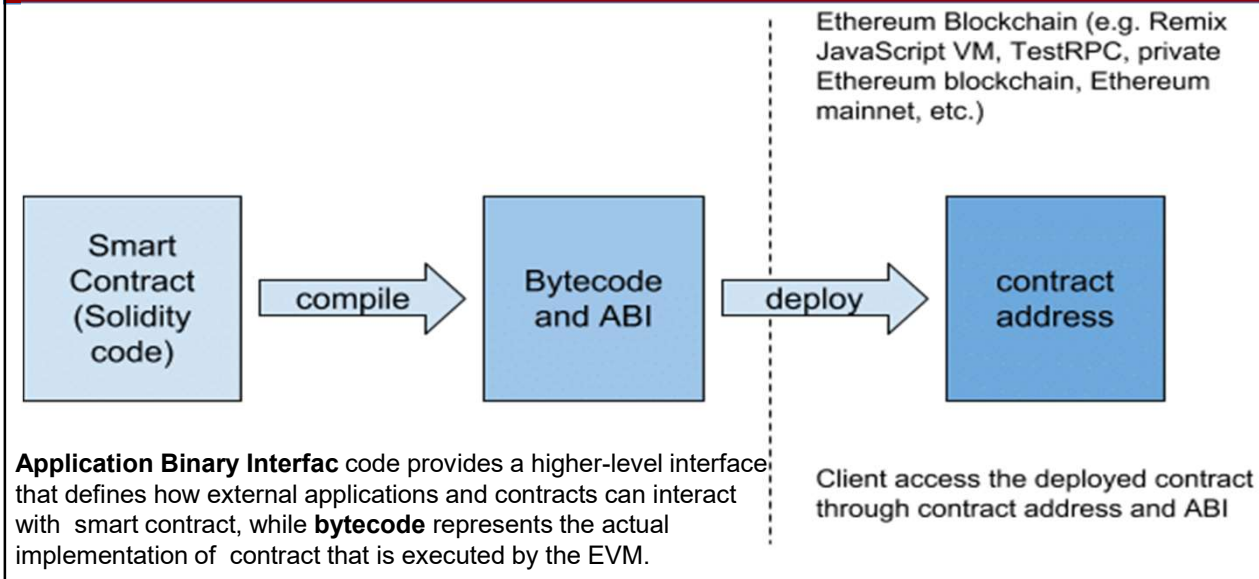
Smart Contract

- A "smart contract" is simply a program that runs on the Ethereum blockchain.
- Smart contracts are a type of Ethereum account i.e. have a balance and can be target of Txs
- not controlled by a user, instead they are deployed to the network and run as programmed
- User accounts can then interact with a smart contract by submitting transactions that execute a function defined on the smart contract.
- It can define rules, like a regular contract, and automatically enforce them via the code.
- Smart contracts cannot be deleted by default, and interactions with them are irreversible.

Benefits

- Immutability characteristic ➔ Contract terms will not change
  - How about when a bug is introduced on a smart contract?
- Autonomy and automaticity characteristics ; Reduce audit and execution cost, and fraud
  - Allows to restrict actions as for acquisition schedule, where an individual owns actions but cannot dispose of them before a given date

Smart Contract Lifecycle



- A self-executing digital contract where the terms of the agreement between buyer and seller are directly written into lines of code.
- Smart contract is not always self-executable
- It requires an external call to be executed. Otherwise, it is pending till one calls one of its implemented functions
- Once it is executed, transactions resulting from the execution are transcribed on the blockchain and eventually smart contract's meta data are updated

Analogy to a DIGITAL VENDING MACHINE

- With the right inputs, a certain output is guaranteed.

Example: To get a snack from a vending machine:

money + snack selection = snack dispensed

- This logic is programmed into the vending machine.
- A smart contract, like a vending machine which has logic programmed into it.
- how this vending machine logic would look if it were a smart contract written in Solidity:

smart contract :vending machine

```
pragma solidity 0.8.7;

contract VendingMachine {

    // Declare state variables of the contract
    address public owner;
    mapping (address => uint) public cupcakeBalances;

    // When 'VendingMachine' contract is deployed:
    // 1. set the deploying address as the owner of the contract
    // 2. set the deployed smart contract's cupcake balance to 100
    constructor() {
        owner = msg.sender;
        cupcakeBalances[address(this)] = 100;

  // Allow the owner to increase the smart contract's cupcake balance
  function refill(uint amount) public {
      require(msg.sender == owner, "Only the owner can refill.");
      cupcakeBalances[address(this)] += amount;
  }

  // Allow anyone to purchase cupcakes
  function purchase(uint amount) public payable {
      require(msg.value >= amount * 1 ether, "You must pay at least 1 ETH per cupcake");
      require(cupcakeBalances[address(this)] >= amount, "Not enough cupcakes in stock to complete this purchase");
      cupcakeBalances[address(this)] -= amount;
      cupcakeBalances[msg.sender] += amount;
  }
}
```

- Ethereum has developer-friendly languages for writing smart contracts:
  - **Solidity**
  - **Vyper**

- a vending machine removes the need for a vendor employee,
- Similarly; smart contracts can replace intermediaries in many industries

Example contract: a sense of what Solidity contract syntax is like

```
pragma solidity >= 0.8.7;
contract Coin {
   // The keyword "public" makes variables
   // accessible from other contracts
   address public minter;
   mapping (address => uint) public balances;

  // Events allow clients to react to specific
  // contract changes you declare
event Sent(address from, address to, uint amount);

   // Constructor code is only run when the contract is created
   constructor() {
      minter = msg.sender;
   }

  // Sends an amount of newly created coins to an address
  // Can only be called by the contract creator
  function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
    }
   // Sends an amount of existing coins  from any caller to an address
  function send(address receiver, uint amount) public
{   require(amount <= balances[msg.sender],
                    "Insufficient balance.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Solidity

docs: https://docs.soliditylang.org/en/develop/

IDE: https://remix-ide.readthedocs.io/en/latest/

ETHEREUM VIRTUAL MACHINE

- EVM is an execution environment for the Ethereum blockchain.
- It allows us to run the smart contract code by compiling it into EVM bytecode.

Solidity → Byte Code → Opcode

- Solidity source code must be compiled into bytecode before being deployed on the Ethereum network.
- This bytecode corresponds to a series of opcode instructions that the EVM interprets.

EVM mechanics: execution environment

Write code in Solidity (or another front-end language)

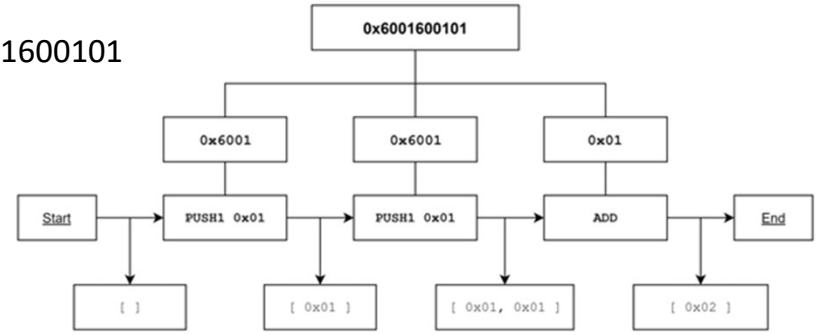
⇒ compile to EVM bytecode
(recent projects use WASM or BPF bytecode)

⇒ miners use the EVM to execute contract bytecode
in response to a Tx

Byte code

- In order to store opcodes efficiently, they are encoded into a byte code.
- Each operation code is allocated a byte (for example, STOP — 0x00).
- During execution, the byte code is split into bytes (1 byte equals 2 hexadecimal characters)

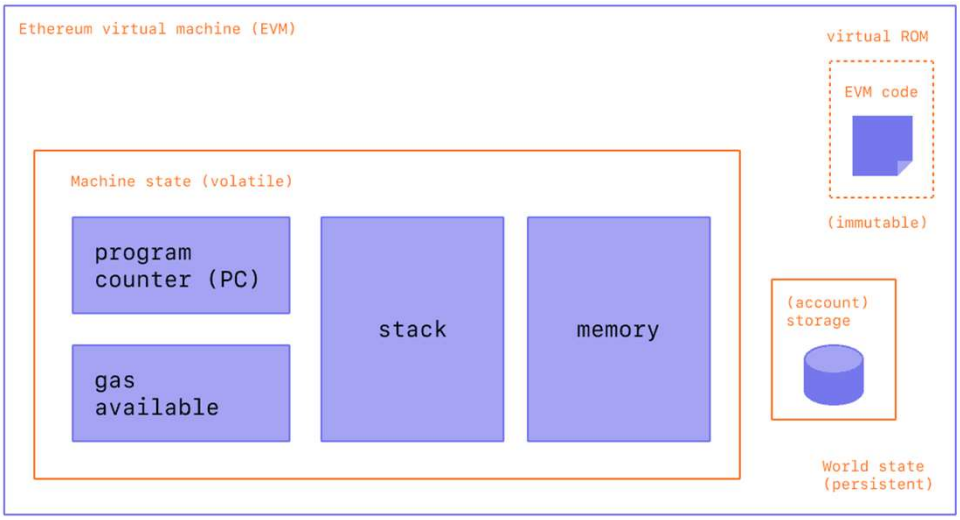
For Example: byte code: 0x6001600101



- Ethereum bytecode is an assembly language made up of multiple opcodes.
- Each opcode performs a certain action on the Ethereum blockchain.

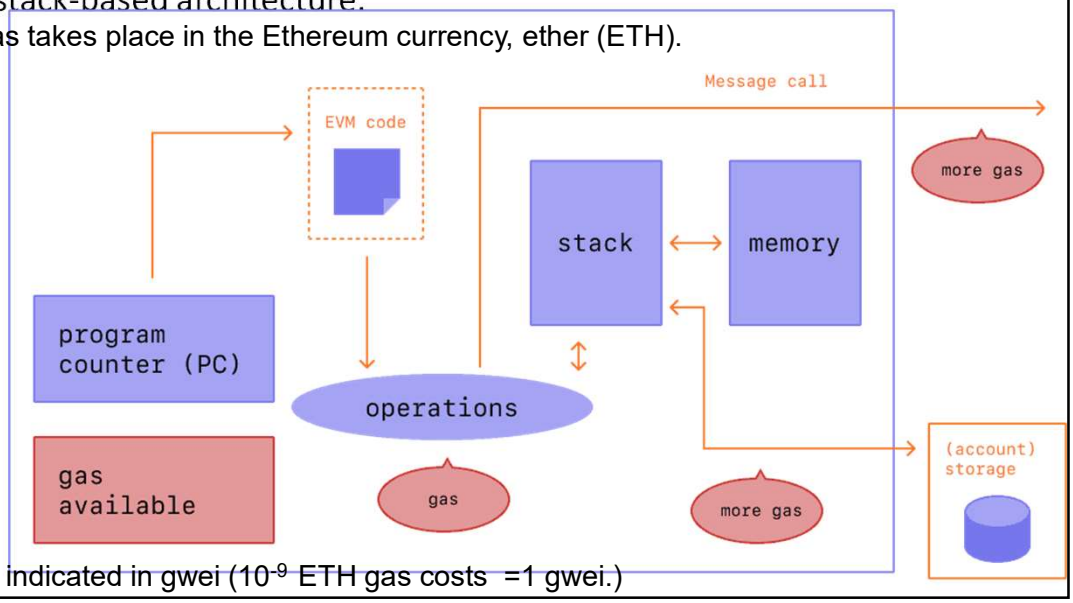
ETHEREUM VIRTUAL MACHINE (EVM)

- The EVM behaves as a mathematical function would: Given an input, it produces a deterministic output



EVM Architecture

- EVM uses a stack-based architecture.
- payment for gas takes place in the Ethereum currency, ether (ETH).



- Gas prices are indicated in gwei (10^{-9} ETH gas costs =1 gwei.)

The EVM

Stack machine (like Bitcoin) but with JUMP

- max stack depth = 1024
- program aborts if stack size exceeded; miner keeps gas
- contract can create or call another contract

In addition: two types of zero initialized memory

- Persistent storage (on blockchain): SLOAD, SSTORE (expensive)
- Volatile memory (for single Tx): MLOAD, MSTORE (cheap)
- LOG0(data): write data to log

Gas prices: examples

SSTORE addr (32 bytes), **value** (32 bytes)

- zero → non-zero: 20,000 gas
- non-zero → non-zero: 5,000 gas
- non-zero → zero: 15,000 gas refund

SELFDESTRUCT addr (SUICIDE): kill current contract. 24,000 gas refund

Refund is given for reducing size of blockchain state

Gas calculation

Tx fees (gas) prevents submitting Tx that runs for many steps
Every EVM instruction costs gas:

- Tx specifies **gasPrice**: conversion: gas → Wei
  gasLimit: max gas for Tx

ETH Gas Fees:

Total Gas Fee = Gas units (limit) x (Base fee + Tip)

gas refund

- is a reward you get for freeing storage on the blockchain.
- done by setting a value back to zero or its default value. When do this, you get a fixed refund of 4,800 units of gas.

Every block has a "baseFee": the minimum gasPrice for all Tx in the block

Gas calculation

Tx specifies **gasPrice**: conversion gas ⇀ Wei
gasLimit: max gas for Tx

(0) if gasPrice < baseFee: abort
(1) if **gasLimit × gasPrice** > msg.sender.balance: abort
(2) deduct **gasLimit × gasPrice** from msg.sender.balance
(3) set Gas = gasLimit
(4) execute Tx: deduct gas from Gas for each instruction
at end if (Gas < 0): abort, miner keeps **gasLimit × gasPrice**
(5) Refund **Gas × gasPrice** to msg.sender.balance
(6) gasUsed ↼ gasLimit – Gas
6 a) BURN gasUsed × baseFee
6 b) Send gasUsed × (gasPrice – baseFee) to block producer

gas for each instruction defined by a hard coded table.

Creating an account

- **EVM processes 160-bit addresses.**
- **The account consists of a cryptographic key pair**: public and private.
  - public key is generated from the private key using the ECDSA algorithm.
- **public address of an Externally-owned account is formed as :**
  - the last 20 bytes from Keccak-256(public key) are taken and 0x is added to the beginning.
- **Contract address is usually specified when deploying a contract in the Ethereum blockchain.**
  - address is formed from the Externally-owned address of the creator and the number of transactions sent from this address (“nonce”).
  - The last 20 bytes from Keccak are 256 (RLP(Externall-owned; nonce))

Creating an account

For a given private key, p_r, the Ethereum address $A(p_r)$ (a 160-bit value) to which it corresponds is defined as the rightmost 160-bits of the Keccak hash of the corresponding ECDSA public key:

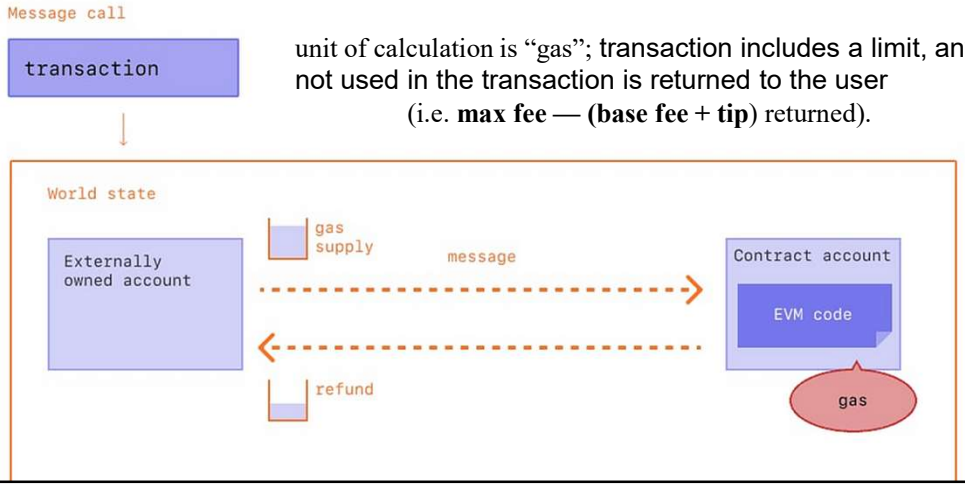
(309) $$A(p_r) = \mathcal{B}_{96..255}\left(\mathtt{KEC}\left(\mathtt{ECDSAPUBKEY}(p_r)\right)\right)$$

The address of the new account is defined as being the rightmost 160 bits of the Keccak hash of the RLP encoding of the structure containing only the sender and the account nonce. Thus we define the resultant address for the new account a:

(77) $$a \equiv \mathcal{B}_{96..255}\left(\mathtt{KEC}\left(\mathtt{RLP}\left(\ (s, \boldsymbol{\sigma}[s]_{\mathrm{n}} - 1)\ \right)\right)\right)$$

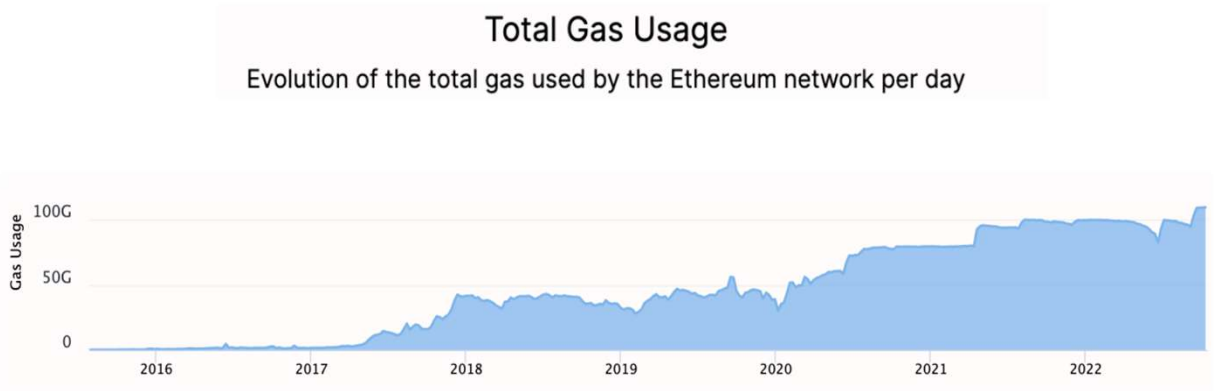
Why do we need gas

- gas fee helps maintain the security of the Ethereum network. By demanding a fee for each calculation performed on the network, do not allow attackers to send spam on the network
- to avoid random or hostile infinite loops or other computational losses in the code, each transaction should set a limit on the number of computational steps of code execution that it can use.



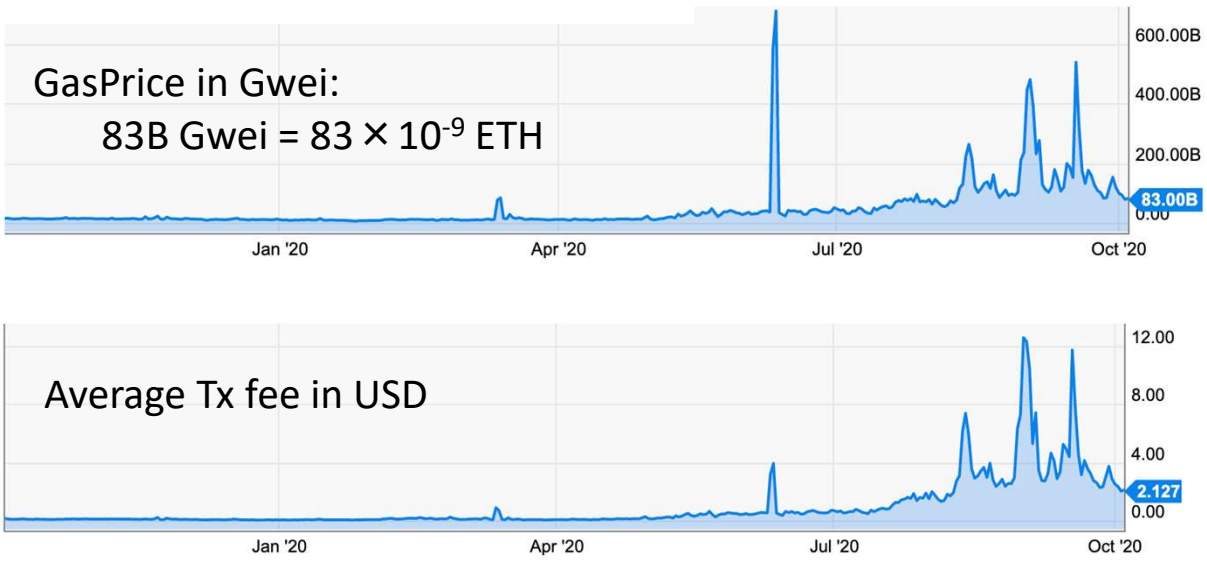
unit of calculation is "gas"; transaction includes a limit, any gas not used in the transaction is returned to the user (i.e. **max fee — (base fee + tip)** returned).

Transactions are becoming more complex



Gas usage is increasing ⇒ each Tx takes more instructions to execute

Gas prices: spike during congestion

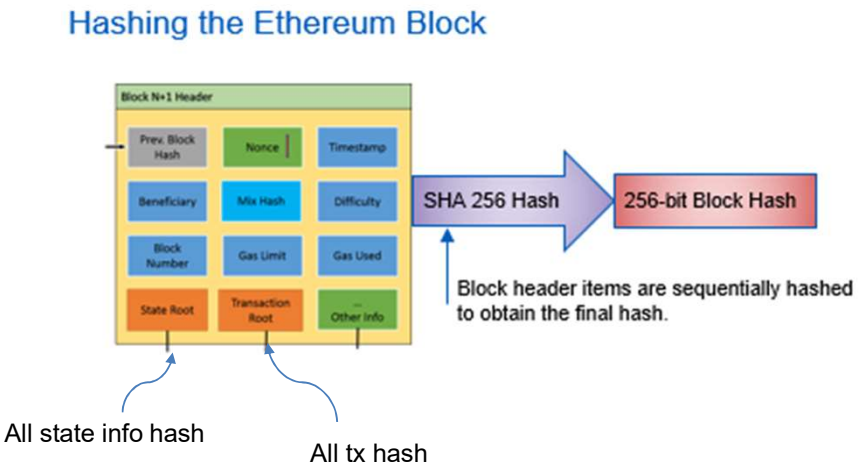


Key Points

- *Externally-owned account*:
  - Creating an account costs nothing;
  - Can initiate transactions;
  - Transactions between Externally-owned can only be transfers of ETH/tokens.
- *Contract*:
  - Creating a contract requires costs because network storage is used;
  - Can only send transactions in response to received transactions;
  - Transactions from an Externally-owned account to a Contract account can run code that can perform many different actions, such as transferring tokens or even creating a new contract.

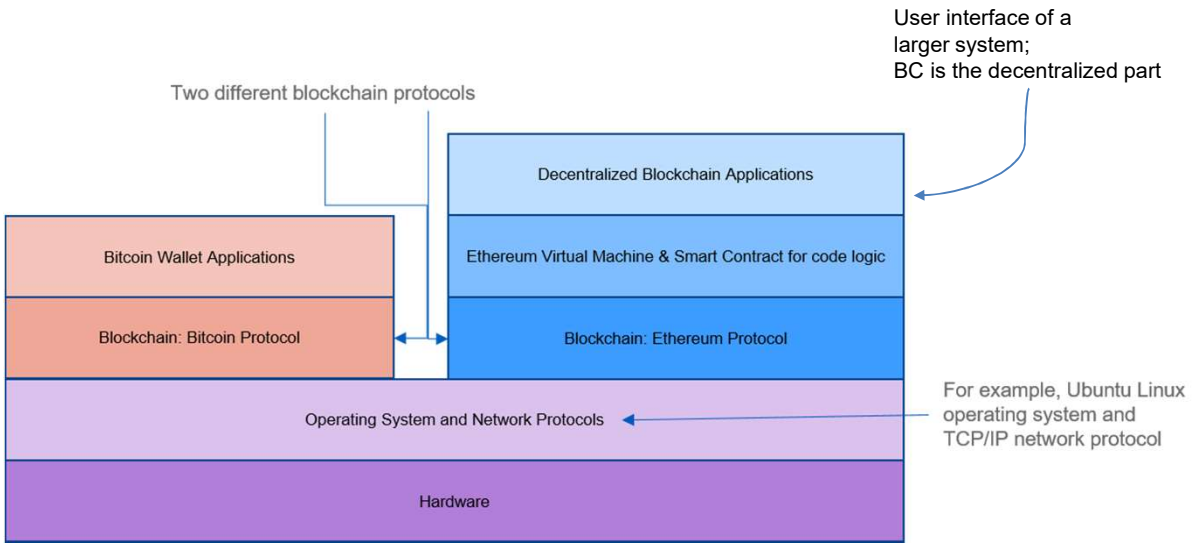
- Tx fees (gas) prevents submitting Tx that runs for many steps.
- During high load: *block proposer chooses Tx from mempool that maximize its income.*

Ethereum block

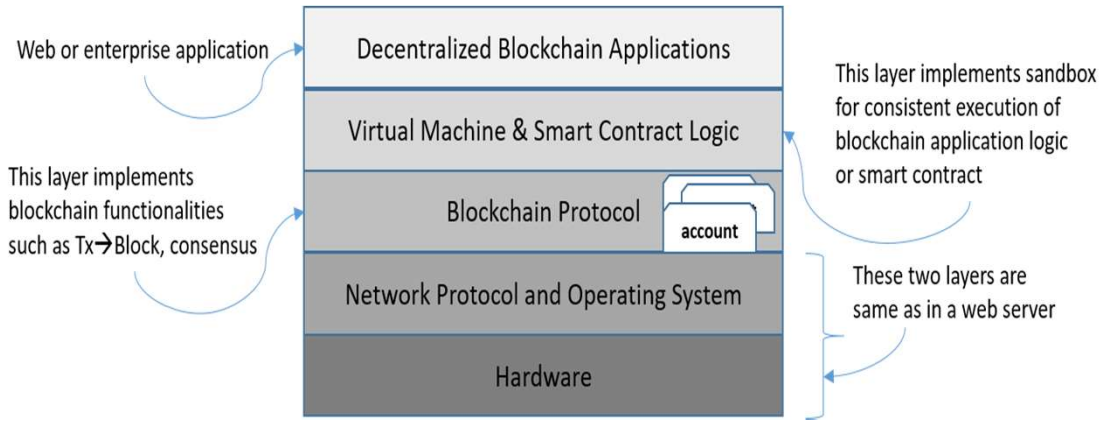


Explore what's in block @https://etherscan.io/

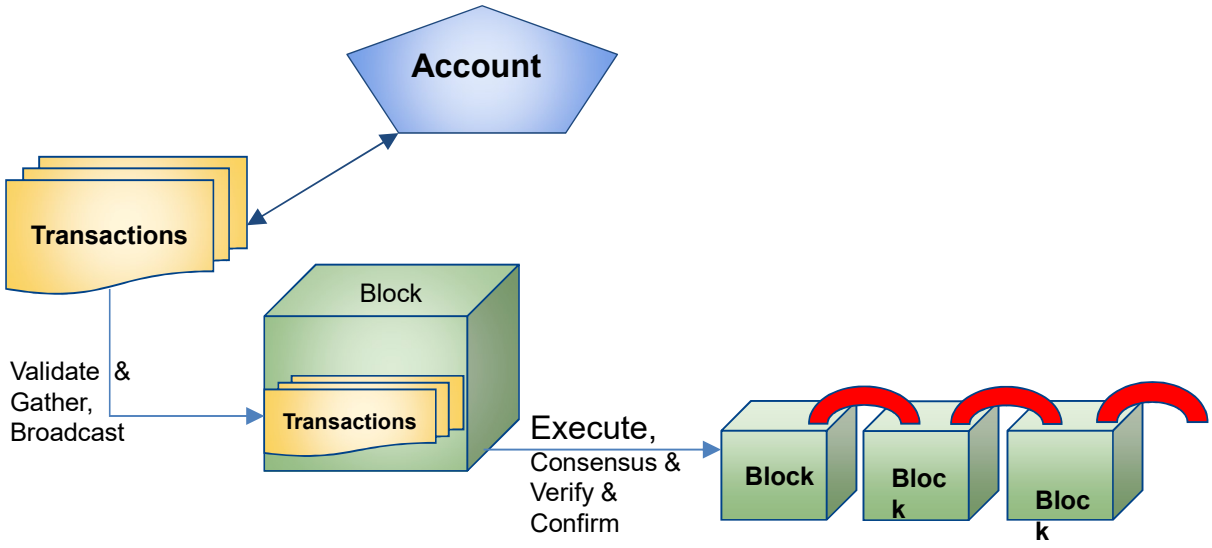
Bitcoin vs Ethereum Stack



Ethereum stack

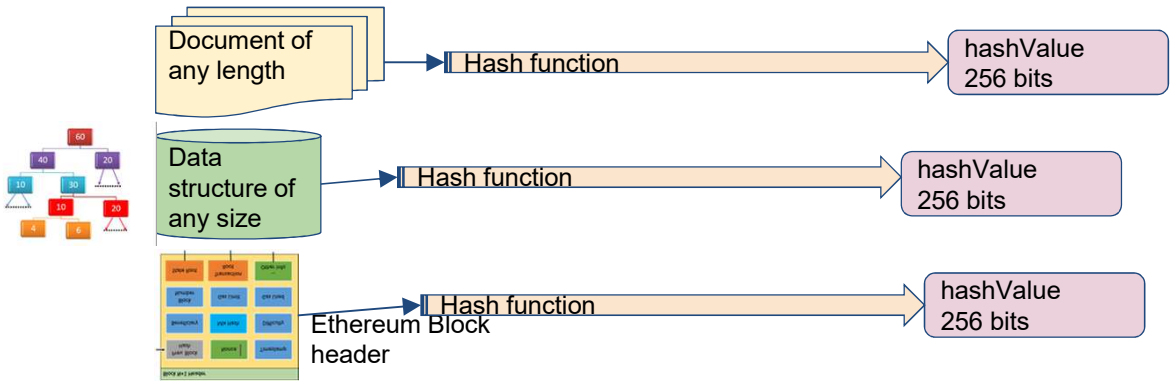


Ethereum: Account-based (not UTXO-based)

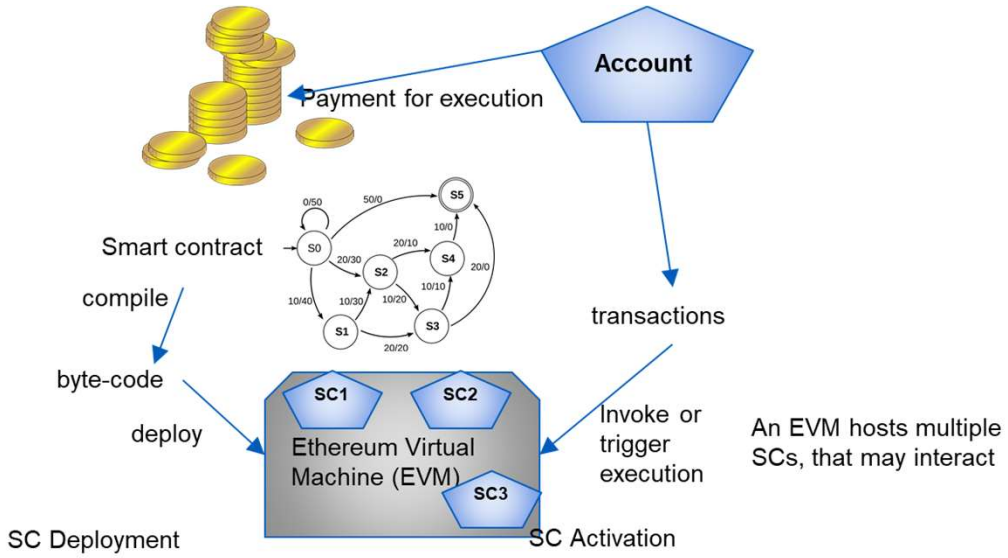


hashing

hashing i.e. a hash function transforms/maps an arbitrary length of input data value to a value of fixed length output value.



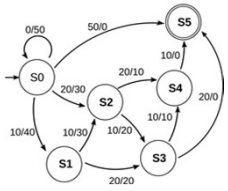
Ethereum system



Expansion of Ethereum Ecosystem

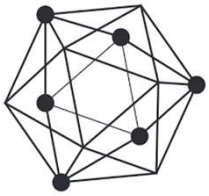
Type 1: only cryptocurrency
Example: **Bitcoin**

Type 2: Cryptocurrency + Smart contract business logic
Example: **Ethereum**



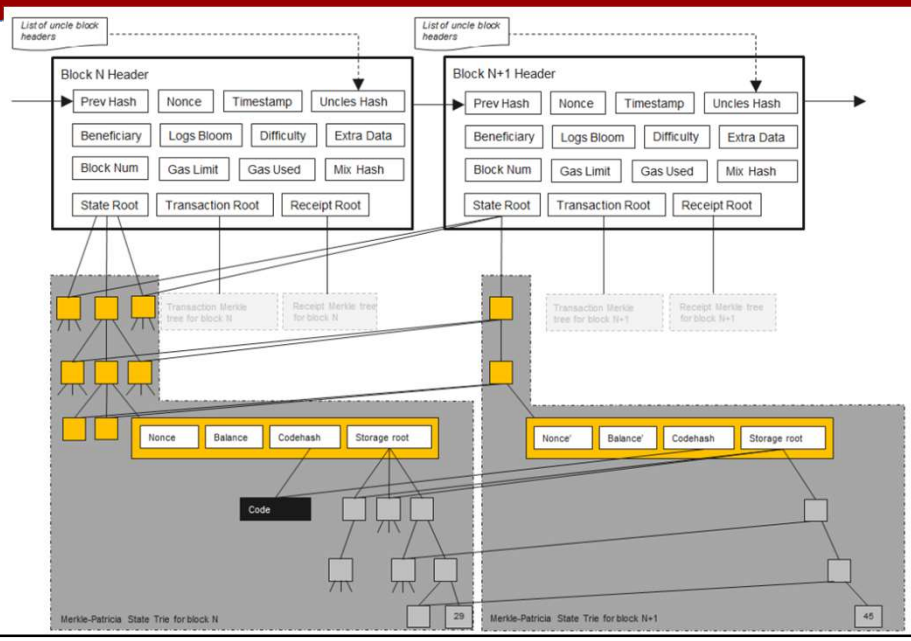
Type 3: Only business logic + NO cryptocurrency
Example: **Hyperledger**

Type 4: Blockchain platform as a service:
Example: **Microsoft Azure**

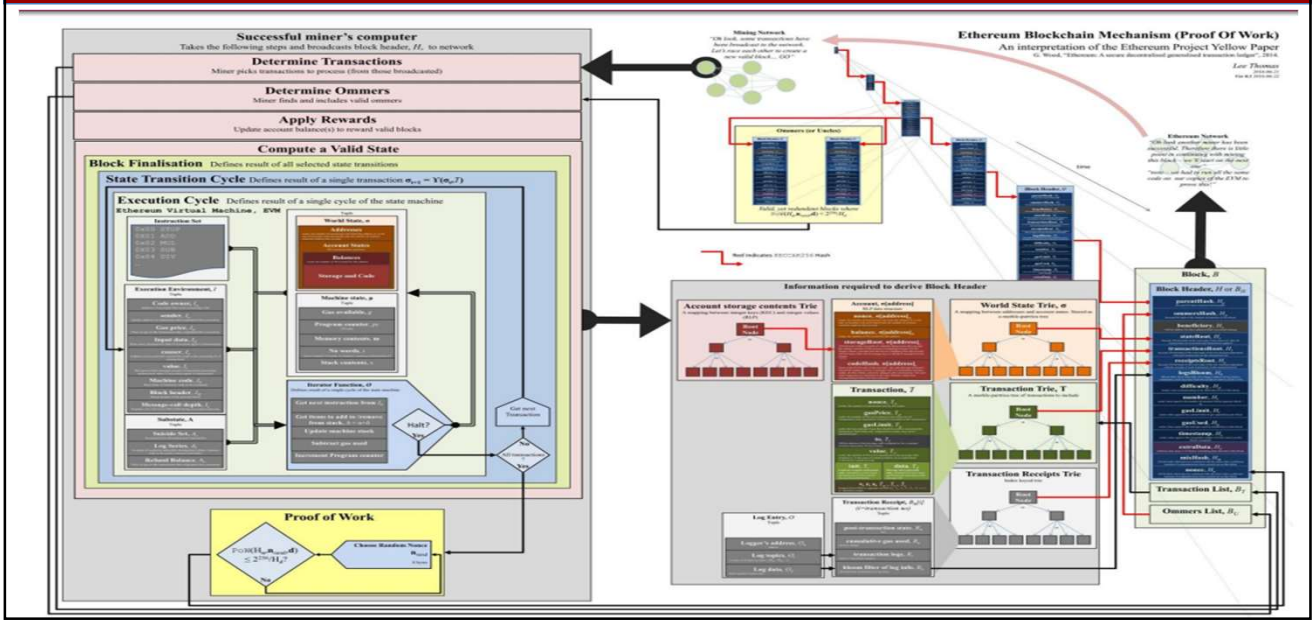


Miscellaneous for reference

Ethereum Block Structure



Ethereum Blockchain Archtecture



Ethereum Blockchain Archtecture