EXTROPY.IO

EVM

EXTROPY.IO

Lesson Plan

Ethereum storage
Ethereum accounts
Ethereum transactions
EVM languages
Ethereum clients and mining
Connecting to a test network

References

DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Mastering Ethereum by Andreas Antonopoulos

White paper

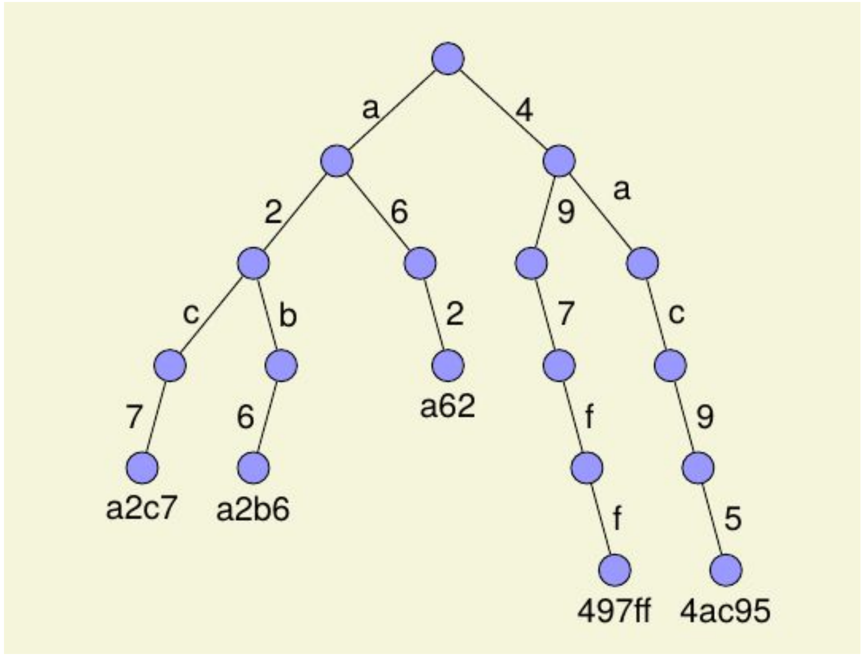
Beige Paper

Yellow Paper

EVM languages

Ethereum Data Structures

Ethereum uses Merkle Patricia Tries / Radix Tries for their searching performance and low memory footprint



EXTROPY.IO

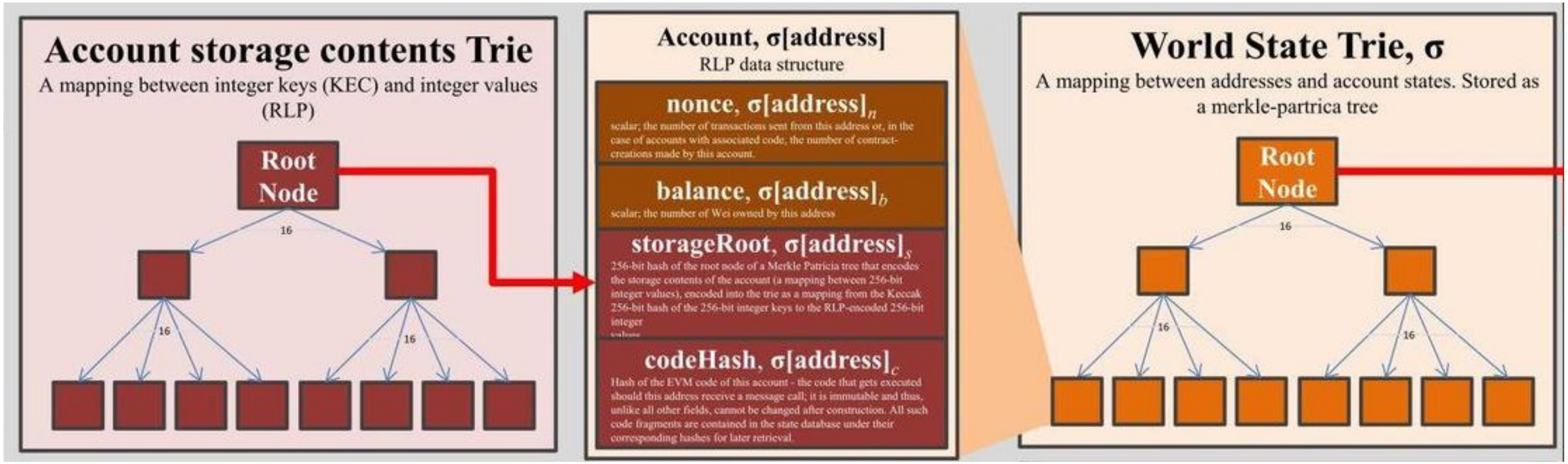
Data Structures

There are 3 Tries

- World State
- Transaction
- Transaction Receipt

See : Ethereum State Trie Architecture Explained

World and Account State



Account storage contents Trie
A mapping between integer keys (KEC) and integer values (RLP)
Root Node
16
16
16
Account, σ[address]
RLP data structure
World State Trie, σ
A mapping between addresses and account states. Stored as a merkle-partrica tree
Root Node
16
16
16

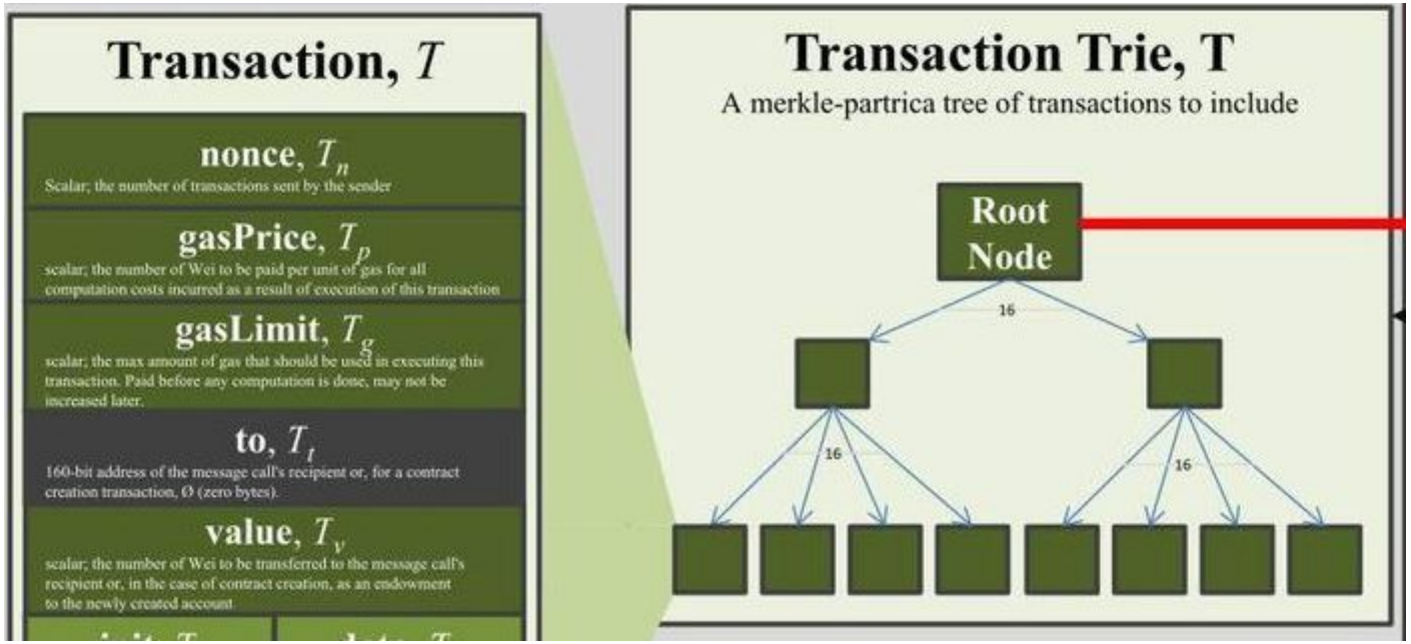
nonce, $\sigma[\text{address}]_n$
scalar; the number of transactions sent from this address or, in the case of accounts with associated code, the number of contract-creations made by this account.

balance, $\sigma[\text{address}]_b$
scalar; the number of Wei owned by this address

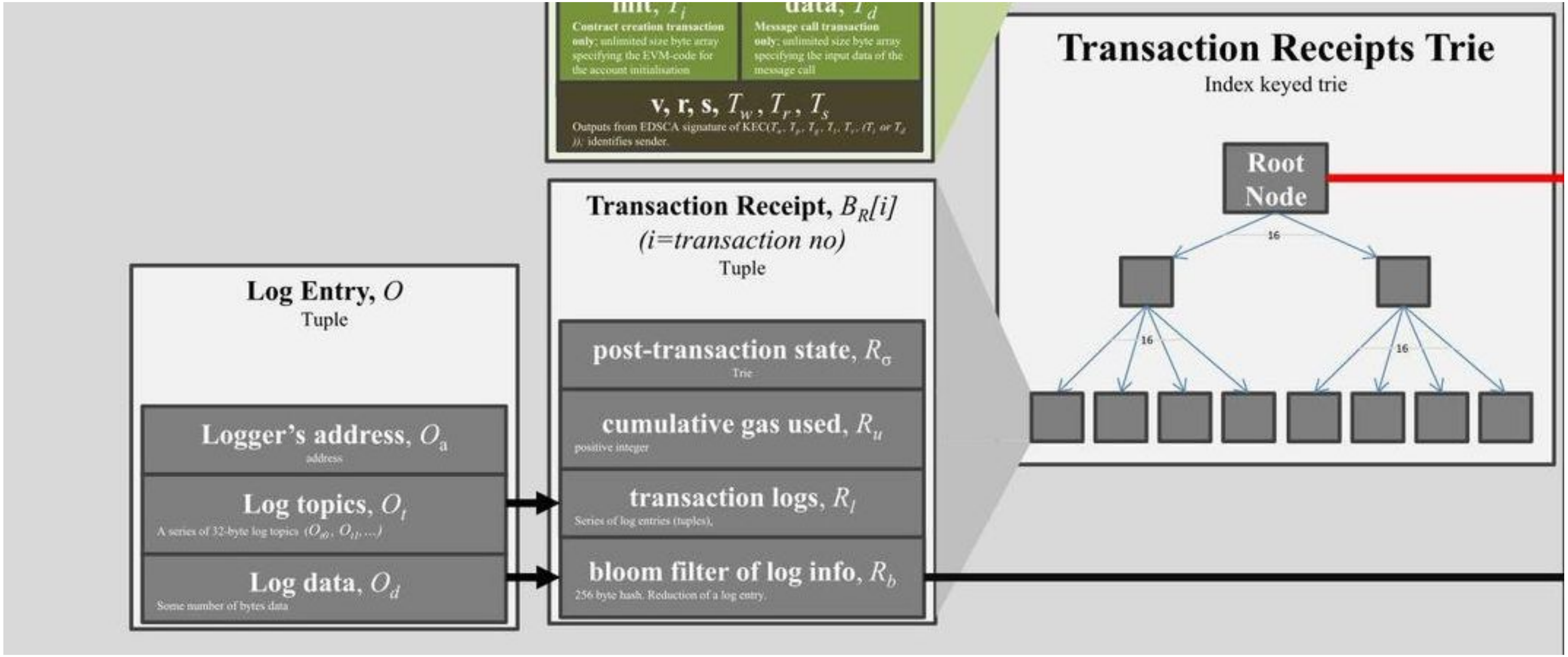
storageRoot, $\sigma[\text{address}]_s$
256-bit hash of the root node of a Merkle Patricia tree that encodes the storage contents of the account (a mapping between 256-bit integer values), encoded into the trie as a mapping from the Keccak 256-bit hash of the 256-bit integer keys to the RLP-encoded 256-bit integer values

codeHash, $\sigma[\text{address}]_c$
Hash of the EVM code of this account - the code that gets executed should this address receive a message call; it is immutable and thus, unlike all other fields, cannot be changed after construction. All such code fragments are contained in the state database under their corresponding hashes for later retrieval.

Transactions


Transaction, T
nonce, T_n
Scalar; the number of transactions sent by the sender
gasPrice, T_p
scalar; the number of Wei to be paid per unit of gas for all computation costs incurred as a result of execution of this transaction
gasLimit, T_g
scalar; the max amount of gas that should be used in executing this transaction. Paid before any computation is done, may not be increased later.
to, T_t
160-bit address of the message call's recipient or, for a contract creation transaction, Ø (zero bytes).
value, T_v
scalar; the number of Wei to be transferred to the message call's recipient or, in the case of contract creation, as an endowment to the newly created account
Transaction Trie, T
A merkle-partrica tree of transactions to include
Root Node
16
16
16

Transaction Receipts and Logs


init, T_i
Contract creation transaction only: unlimited size byte array specifying the EVM-code for the account initialisation
data, T_d
Message call transaction only: unlimited size byte array specifying the input data of the message call
v, r, s, T_w, T_r, T_s
Outputs from EDSCA signature of KEC(T_n, T_p, T_g, T_t, T_v, (T_i or T_d)); identifies sender.
Transaction Receipts Trie
Index keyed trie
Root Node
16
16
16
Transaction Receipt, $B_R[i]$
(i=transaction no)
Tuple
post-transaction state, R_σ
Trie
cumulative gas used, R_u
positive integer
transaction logs, R_l
Series of log entries (tuples),
bloom filter of log info, R_b
256 byte hash. Reduction of a log entry.
Log Entry, O
Tuple
Logger's address, O_a
address
Log topics, O_t
A series of 32-byte log topics (O_{t0}, O_{t1}, ...)
Log data, O_d
Some number of bytes data

Transaction and Transaction Receipt Tries

Purpose:

- **Transaction Tries**: records transaction request vectors
- **Transaction Receipt Tries**: records the transaction outcome

Content:

Parameters used in composing a **Transaction Trie** [details in section 4.3 of the yellow paper]:

- nonce,
- gas price,
- gas limit,
- recipient,
- transfer value,
- transaction signature values, and
- account initialization (if transaction is of contract creation type), or transaction data (if transaction is a message call)

Parameters used in composing a **Transaction Receipt Trie** [details in section 4.4.1 of the yellow paper]:

- post-transaction state,
- the cumulative gas used,
- the set of logs created through execution of the transaction, and
- the Bloom filter composed from information in those logs

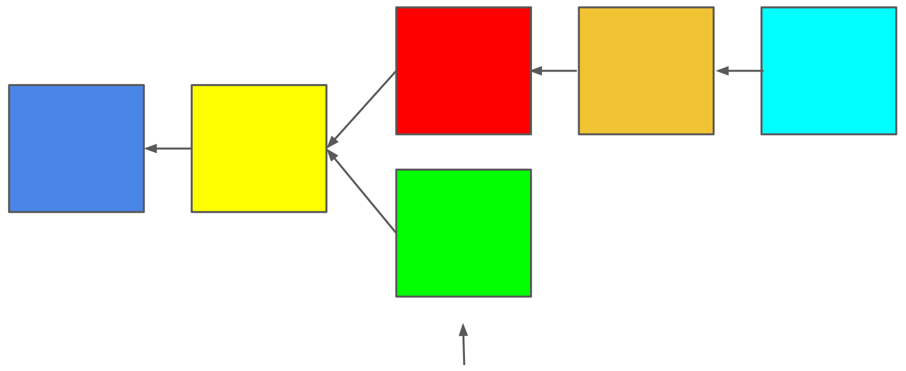
Ethereum Block Fields

- Header
  - Parent Hash
  - Ommers Hash
  - Beneficiary
  - State Root
  - Transaction Root
  - Receipts Root
  - Logs Bloom
  - Difficulty
  - Block Number
  - Gas Limit
  - Gas Used
  - Timestamp
  - Extra Data
  - Mix Hash
  - Nonce
- Ommer Block Headers
- Transactions

See Beige Paper

Ommer (Uncle) Blocks

Unlike Bitcoin, Ethereum gives rewards to blocks that have been produced and are valid, but do not form part of the canonical chain. Has to be within the last six blocks



Orphan in Bitcoin (No reward)
Ommer in Ethereum (Small reward)

Ommer block defined as “The child of an ancestor that is not itself an ancestor”

- Smaller reward given for ommer blocks
- Transactions from ommer blocks are not included

2 Types of accounts in Ethereum

Externally Owned Accounts (EOA)

Controlled by people + private keys

Contract accounts (Smart Contracts)

Controlled by smart contract code + storage

Note that because a contract account does not have a private key, it cannot initiate a transaction. Only EOAs can initiate transactions, but contracts can react to transactions by calling other contracts, building complex execution paths.

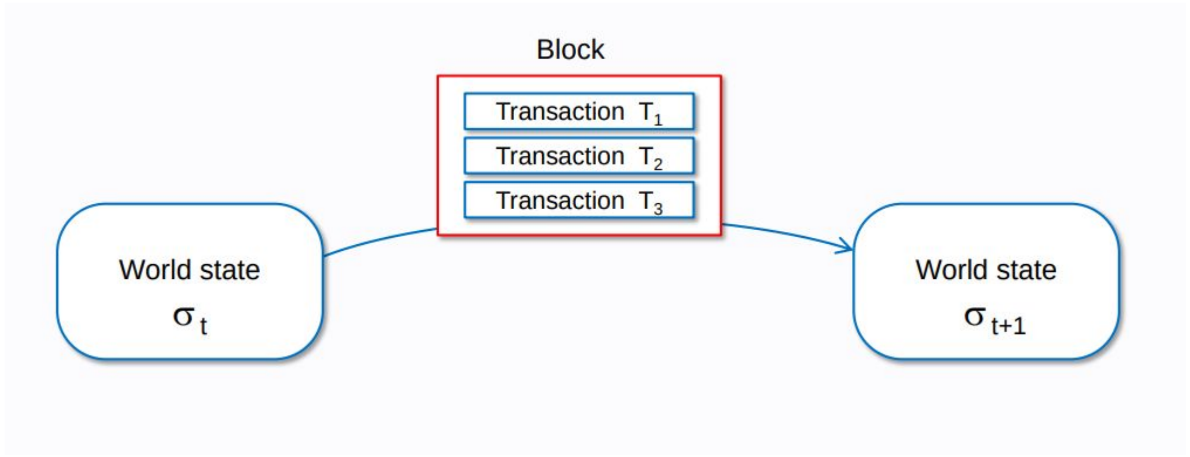
EXTROPY.IO

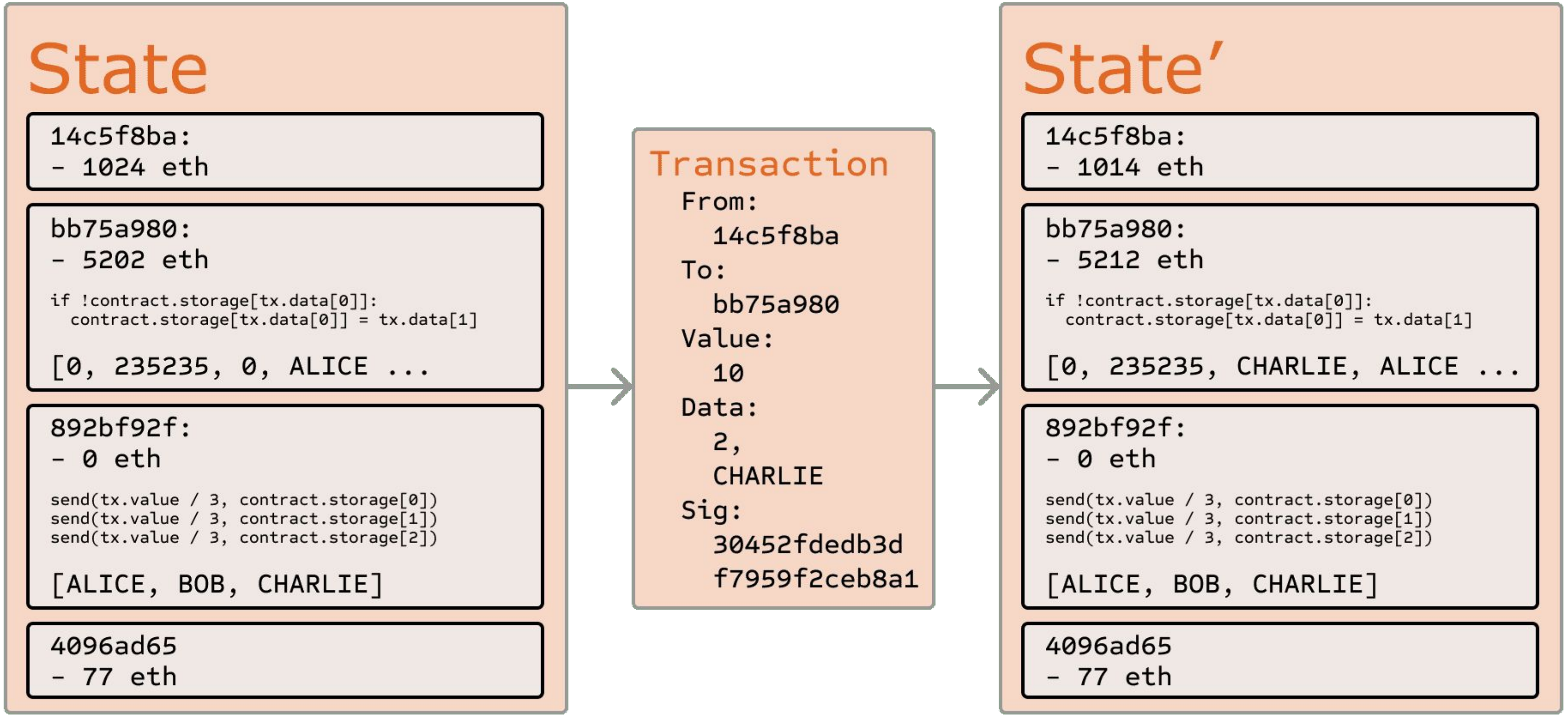
The Account State

The account state comprises the following four fields:

- balance: A scalar value equal to the number of Wei owned by this address.
- storageRoot: A 256-bit hash of the root node of a Merkle Patricia tree that encodes the storage contents of the account (a mapping between 256-bit integer values)
- codeHash: The hash of the EVM code of this account
- nonce: A scalar value equal to the number of transactions sent from this address or, in the case of accounts with associated code, the number of contract-creations made by this account.

Ethereum Transactions





```
if !contract.storage[tx.data[0]]:
  contract.storage[tx.data[0]] = tx.data[1]
```

```
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
```

```
if !contract.storage[tx.data[0]]:
  contract.storage[tx.data[0]] = tx.data[1]
```

```
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
```

The Ethereum state transition function, `APPLY(S,TX) -> S'` can be defined as follows:

1. Check if the transaction is well-formed (ie. has the right number of values), the signature is valid, and the nonce matches the nonce in the sender's account. If not, return an error.
2. Calculate the transaction fee as `STARTGAS * GASPRICE`, and determine the sending address from the signature. Subtract the fee from the sender's account balance and increment the sender's nonce. If there is not enough balance to spend, return an error.
3. Initialize `GAS = STARTGAS`, and take off a certain quantity of gas per byte to pay for the bytes in the transaction.
4. Transfer the transaction value from the sender's account to the receiving account. If the receiving account does not yet exist, create it. If the receiving account is a contract, run the contract's code either to completion or until the execution runs out of gas.
5. If the value transfer failed because the sender did not have enough money, or the code execution ran out of gas, revert all state changes except the payment of the fees, and add the fees to the miner's account.
6. Otherwise, refund the fees for all remaining gas to the sender, and send the fees paid for gas consumed to the miner.

A submitted transaction includes the following information:

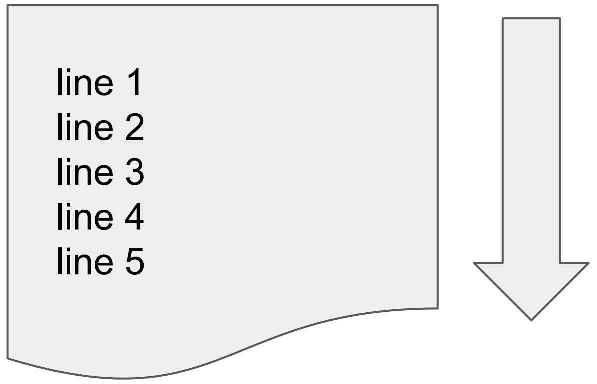
- `recipient` – the receiving address (if an externally-owned account, the transaction will transfer value. If a contract account, the transaction will execute the contract code)
- `signature` – the identifier of the sender. This is generated when the sender's private key signs the transaction and confirms the sender has authorised this transaction
- `value` – amount of ETH to transfer from sender to recipient (in WEI, a denomination of ETH)
- `data` – optional field to include arbitrary data
- `gasLimit` – the maximum amount of gas units that can be consumed by the transaction. Units of gas represent computational steps
- `maxPriorityFeePerGas` - the maximum amount of gas to be included as a tip to the miner
- `maxFeePerGas` - the maximum amount of gas willing to be paid for the transaction (inclusive of `baseFeePerGas` and `maxPriorityFeePerGas`)

Some practical points about transactions

- Miners choose which transactions to include in a block
- Miners can add their own transactions to a block
- Miners choose the order of transactions in a block
- Your transaction is in competition with other transactions for inclusion in the block

Processing a transaction

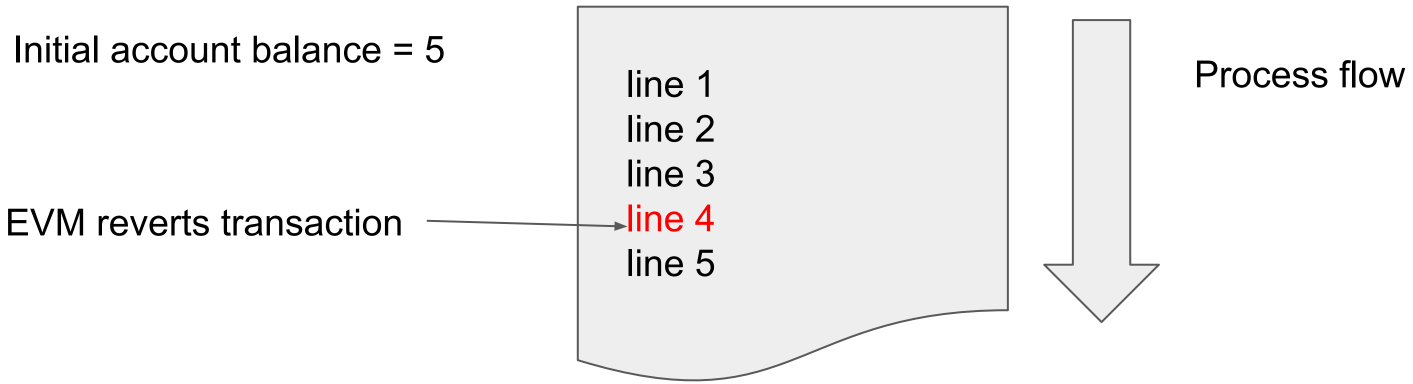
Initial account balance = 5



Process flow

Final account balance = 10

Transactions are atomic

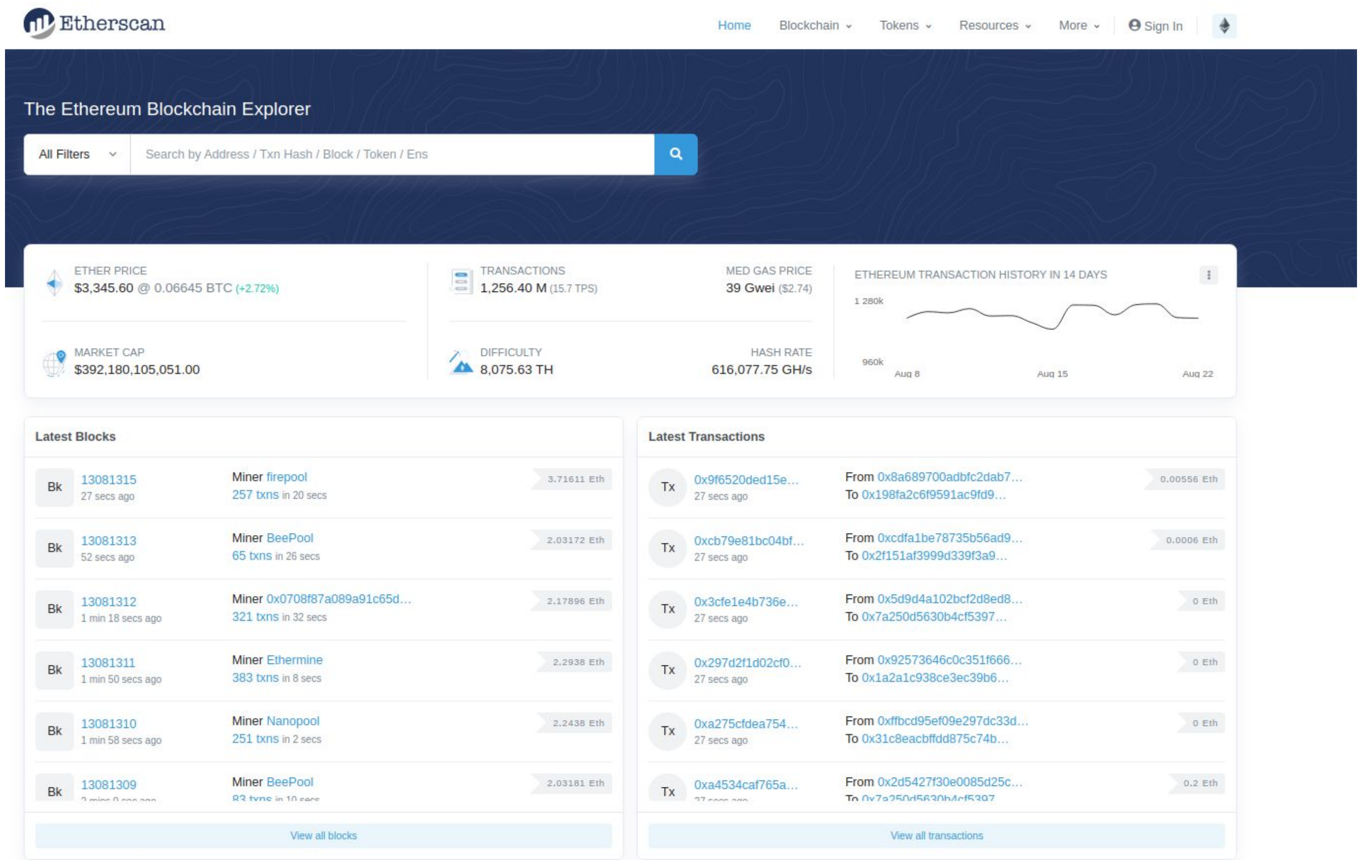


Final account balance = 5

EXTROPY.IO

Analysing Ethereum Data

Block Explorer



Etherscan

The Ethereum Blockchain Explorer

ETHER PRICE
$3,345.60 @ 0.06645 BTC (+2.72%)

MARKET CAP
$392,180,105,051.00

TRANSACTIONS
1,256.40 M (15.7 TPS)

MED GAS PRICE
39 Gwei ($2.74)

DIFFICULTY
8,075.63 TH

HASH RATE
616,077.75 GH/s

ETHEREUM TRANSACTION HISTORY IN 14 DAYS

Latest Blocks

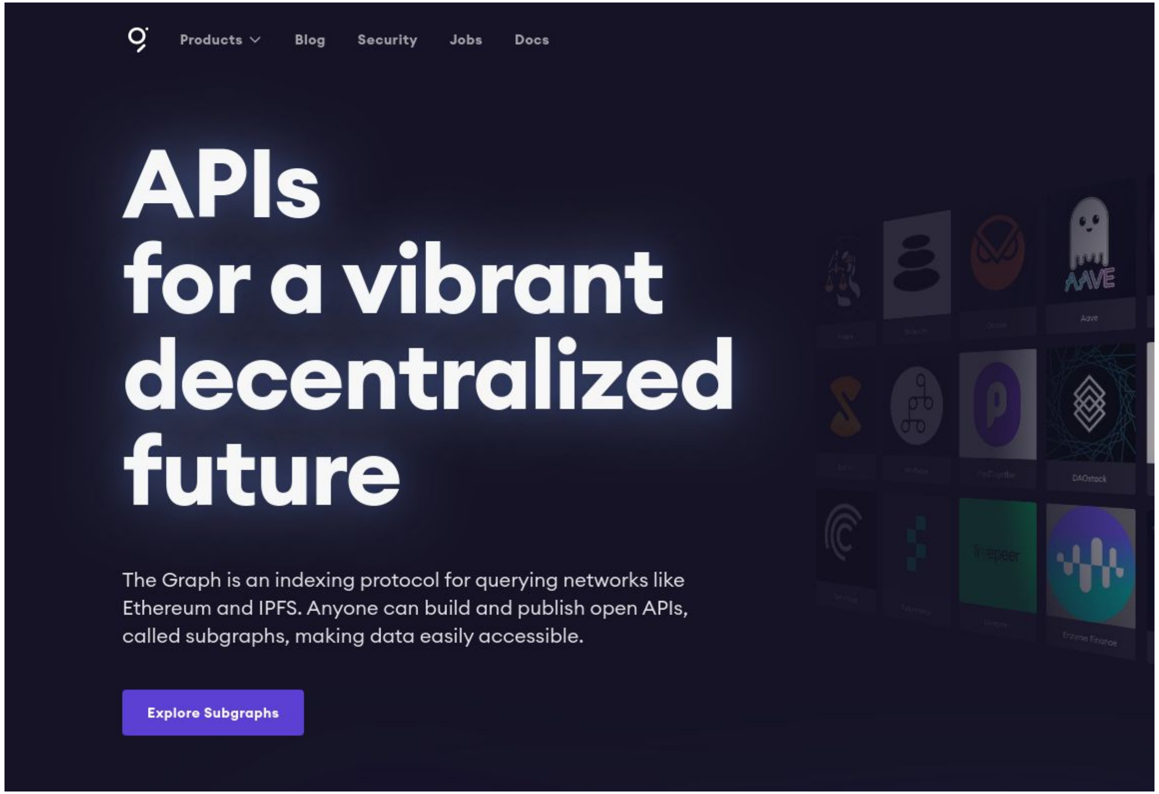
Bk	13081315 27 secs ago	Miner firepool 257 txns in 20 secs	3.71611 Eth
Bk	13081313 52 secs ago	Miner BeePool 65 txns in 26 secs	2.03172 Eth
Bk	13081312 1 min 18 secs ago	Miner 0x0708f87a089a91c65d… 321 txns in 32 secs	2.17896 Eth
Bk	13081311 1 min 50 secs ago	Miner Ethermine 383 txns in 8 secs	2.2938 Eth
Bk	13081310 1 min 58 secs ago	Miner Nanopool 251 txns in 2 secs	2.2438 Eth
Bk	13081309	Miner BeePool	2.03181 Eth

View all blocks

Latest Transactions

Tx	0x9f6520ded15e… 27 secs ago	From 0x8a689700adbfc2dab7… To 0x198fa2c6f9591ac9fd9…	0.00556 Eth
Tx	0xcb79e81bc04bf… 27 secs ago	From 0xcdfa1be78735b56ad9… To 0x2f151af3999d339f3a9…	0.0006 Eth
Tx	0x3cfe1e4b736e… 27 secs ago	From 0x5d9d4a102bcf2d8ed8… To 0x7a250d5630b4cf5397…	0 Eth
Tx	0x297d2f1d02cf0… 27 secs ago	From 0x92573646c0c351f666… To 0x1a2a1c938ce3ec39b6…	0 Eth
Tx	0xa275cfdea754… 27 secs ago	From 0xffbcd95ef09e297dc33d… To 0x31c8eacbffdd875c74b…	0 Eth
Tx	0xa4534caf765a…	From 0x2d5427f30e0085d25c…	0.2 Eth

View all transactions



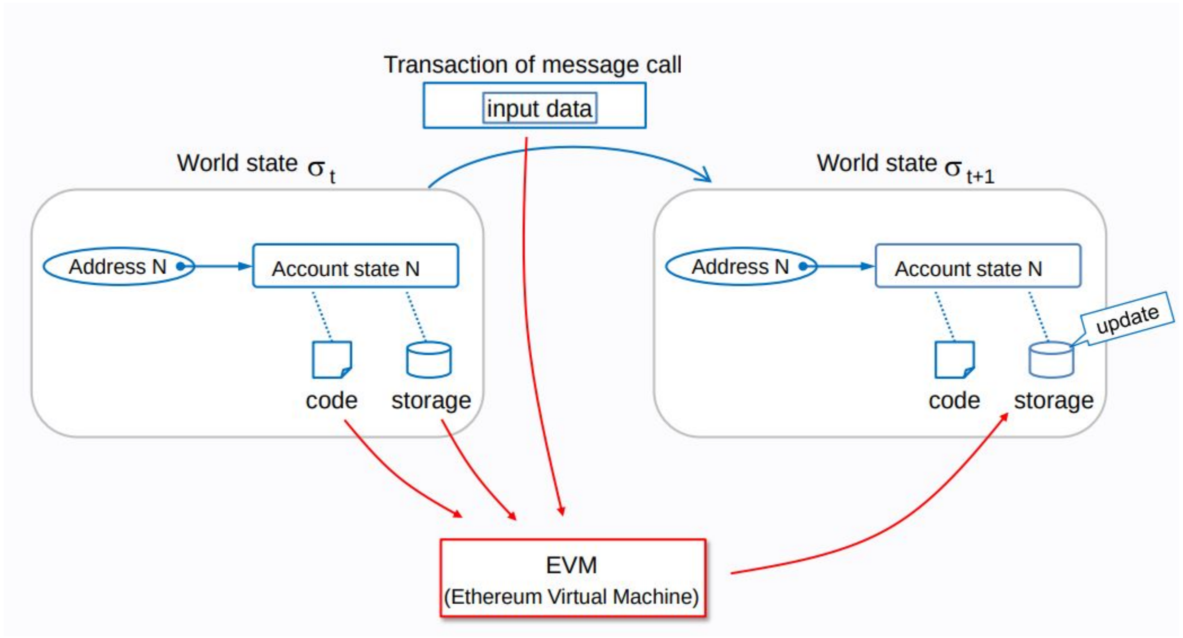
APIs for a vibrant decentralized future

The Graph is an indexing protocol for querying networks like Ethereum and IPFS. Anyone can build and publish open APIs, called subgraphs, making data easily accessible.

The Graph

Ethereum Virtual Machine

Ethereum is a distributed state machine.
Ethereum's state is a large data structure which holds not only all accounts and balances, but a *machine state*, which can change from block to block according to a pre-defined set of rules, and which can execute arbitrary machine code.



From : Ethereum EVM

The EVM is a stack machine , the stack has a maximum size of 1024.

Stack items have a size of 256 bits; in fact, the EVM is a 256-bit word machine (this facilitates Keccak256 hash scheme and elliptic-curve computations).

During execution 2 areas are available for variables

- memory - a transient memory which does not persist between transactions
- storage - part of a Merkle Patricia storage trie associated with the contract’s account, part of the global state

EXTROPY.IO

Execution / OP codes

A compiled smart contract bytecode executes as a number of EVM opcodes

There is a maximum of 256 possible op codes : Reference sheet for op codes

Op codes can be categorised as

- **Stack-manipulating opcodes** (*POP, PUSH, DUP, SWAP*)
- **Arithmetic/comparison/bitwise opcodes** (*ADD, SUB, GT, LT, AND, OR*)
- **Environmental opcodes** (*CALLER, CALLVALUE, NUMBER*)
- **Memory-manipulating opcodes** (*MLOAD, MSTORE, MSTORE8, MSIZE*)
- **Storage-manipulating opcodes** (*SLOAD, SSTORE*)
- **Program counter related opcodes** (*JUMP, JUMPI, PC, JUMPDEST*)
- **Halting opcodes** (*STOP, RETURN, REVERT, INVALID, SELFDESTRUCT*)

EVM Languages

See : https://ethereum.org/en/developers/docs/smart-contracts/languages/

- Solidity
  - The most popular programming language for Ethereum contracts
- LLL
  - *Low-level Lisp-like Language*
- Vyper
  - A language with overflow-checking, numeric units but without unlimited loops
- Yul / Yul+
  - An intermediate language that can be compiled to bytecode for different backends. Support for EVM 1.0, EVM 1.5 and Ewasm is planned, and it is designed to be a usable common denominator of all three platforms.
- FE
  - Statically typed language Inspired by Rust and Python

- Pyramid Scheme (experimental)
  - A Scheme compiler into EVM that follows the SICP compilation approach
- Flint
  - A language with several security features: e.g. asset types with a restricted set of atomic operations
- LLLL
  - An LLL-like compiler being implemented in Isabelle/HOL
- HAseembly-evm
  - An EVM assembly implemented as a Haskell DSL
- Bamboo (experimental)
  - A language without loops

Vyper

- Pythonic programming language
- Strong typing
- Small and understandable compiler code
- Deliberately has less features than Solidity with the aim of making contracts more secure and easier to audit. Vyper does not support:
    - Modifiers
    - Inheritance
    - Inline assembly
    - Function overloading
    - Operator overloading
    - Recursive calling
    - Infinite-length loops
    - Binary fixed points

Example Vyper Code

```
beneficiary: public(address)
auctionStart: public(uint256)
auctionEnd: public(uint256)

pendingReturns: public(HashMap[address, uint256])

# Create a simple auction with `_bidding_time`
@external
def __init__(_beneficiary: address, _bidding_time: uint256):
    self.beneficiary = _beneficiary
    self.auctionStart = block.timestamp
    self.auctionEnd = self.auctionStart + _bidding_time
```

```
@external
@payable
def bid():

    assert block.timestamp <
self.auctionEnd
    # Check if bid is high enough
    assert msg.value > self.highestBid
self.pendingReturns[self.highestBidder]
+= self.highestBid
    # Track new high bid
    self.highestBidder = msg.sender
    self.highestBid = msg.value
```

Yul

- Intermediate language for Ethereum.
- Supports the EVM and Ewasm, an Ethereum flavored WebAssembly, and is designed to be a usable common denominator of both platforms.
- Good target for high-level optimisation stages that can benefit both EVM and Ewasm platforms equally.

Yul+

- A low-level, highly efficient extension to Yul.
- Initially designed for an optimistic rollup contract.
- Yul+ can be looked at as an experimental upgrade proposal to Yul, adding new features to it.

Yul Example

```
{
    function power(base, exponent) -> result
    {
        switch exponent
        case 0 { result := 1 }
        case 1 { result := base }
        default
        {
            result := power(mul(base, base), div(exponent, 2))
            if mod(exponent, 2) { result := mul(base, result) }
        }
    }
    let res := power(calldataload(0), calldataload(32))
    mstore(0, res)
    return(0, 32)
}
```

FE

See : https://ethereum.org/en/developers/docs/smart-contracts/languages/#fe

- Statically typed language for the Ethereum Virtual Machine (EVM).
- Inspired by Python and Rust.
- Aims to be easy to learn -- even for developers who are new to the Ethereum ecosystem.
- Fe development is still in its early stages, the language had its alpha release in January 2021.

FE Example

```
type BookMsg = bytes[100]

contract GuestBook:
    pub guest_book: map<address, BookMsg>

    event Signed:
        book_msg: BookMsg

    pub def sign(book_msg: BookMsg):
        self.guest_book[msg.sender] = book_msg

        emit Signed(book_msg=book_msg)

    pub def get_msg(addr: address) -> BookMsg:
        return self.guest_book[addr].to_mem()
```

Comparison / Cheat Sheet for Solidity and Vyper

https://reference.auditless.com/cheatsheet/

Ethereum Clients

Name	Language	Sync Strategy	State Pruning
Geth	Go	Fast, Full	Archive, Pruned
Open Ethereum	Rust	Warp, Full	Archive, Pruned
Nethermind	C#	Fast, Full	Archive, Pruned
Besu	Java	Fast, Full	Archive, Pruned
Erigon	Go	Fast, Full	Archive, Pruned

Sync Modes

- Full – downloads all blocks (including headers, transactions and receipts) and generates the state of the blockchain incrementally by executing every block.
- Fast (Default) – downloads all blocks (including headers, transactions and receipts), verifies all headers, and downloads the state and verifies it against the headers.
- Light – downloads all block headers, block data, and verifies some randomly.
- Warp sync – Every 5,000 blocks, nodes will take a consensus-critical snapshot of that block’s state. Any node can fetch these snapshots over the network, enabling a fast sync.
- Beam sync – A sync mode that allows you to get going faster. It doesn't require long waits to sync, instead it back-fills data over time.
- Header sync – you can use a trusted checkpoint to start syncing from a more recent header and then leave it up to a background process to fill the gaps eventually

Mining on Ethereum

Miners run Ethash (PoW algorithm)

- Uses a 4 GB dataset, which is updated every 30,000 blocks

Ethash is intended to satisfy the following goals:

1. IO saturation: The algorithm should consume nearly the entire available memory access bandwidth (this is a strategy toward achieving ASIC resistance)
2. GPU friendliness: We try to make it as easy as possible to mine with GPUs. Targeting CPUs is almost certainly impossible, as potential specialization gains are too great, and there do exist criticisms of CPU-friendly algorithms that they are vulnerable to botnets, so we target GPUs as a compromise.
3. Light client verifiability: a light client should be able to verify a round of mining in under 0.01 seconds on a desktop in C, and under 0.1 seconds in Python or Javascript, with at most 1 MB of memory (but exponentially increasing)
4. Light client slowdown: the process of running the algorithm with a light client should be much slower than the process with a full client, to the point that the light client algorithm is not an economically viable route toward making a mining implementation, including via specialized hardware.
5. Light client fast startup: a light client should be able to become fully operational and able to verify blocks within 40 seconds in Javascript.

From Ethereum Wiki

EXTROPY.IO

Ethereum Test Networks

The Ropsten test network is a Proof-of-Work testnet for Ethereum. To acquire ETH on Ropsten, one can mine on the network.

The Kovan test network is a Proof-of-Authority testnet for Ethereum, originally started by the Parity team. To acquire ETH on Kovan, one can request it from a faucet.

The Rinkeby test network is a Proof-of-Authority testnet for Ethereum, originally started by the Geth team. To acquire ETH on Rinkeby, one can request it from a faucet.

The Görli test network is a Proof-of-Authority testnet for Ethereum, originally proposed by Chainsafe and Afri Schoedon. To acquire ETH on Görli, one can use the one-way throttled bridge from any of the other three test networks.