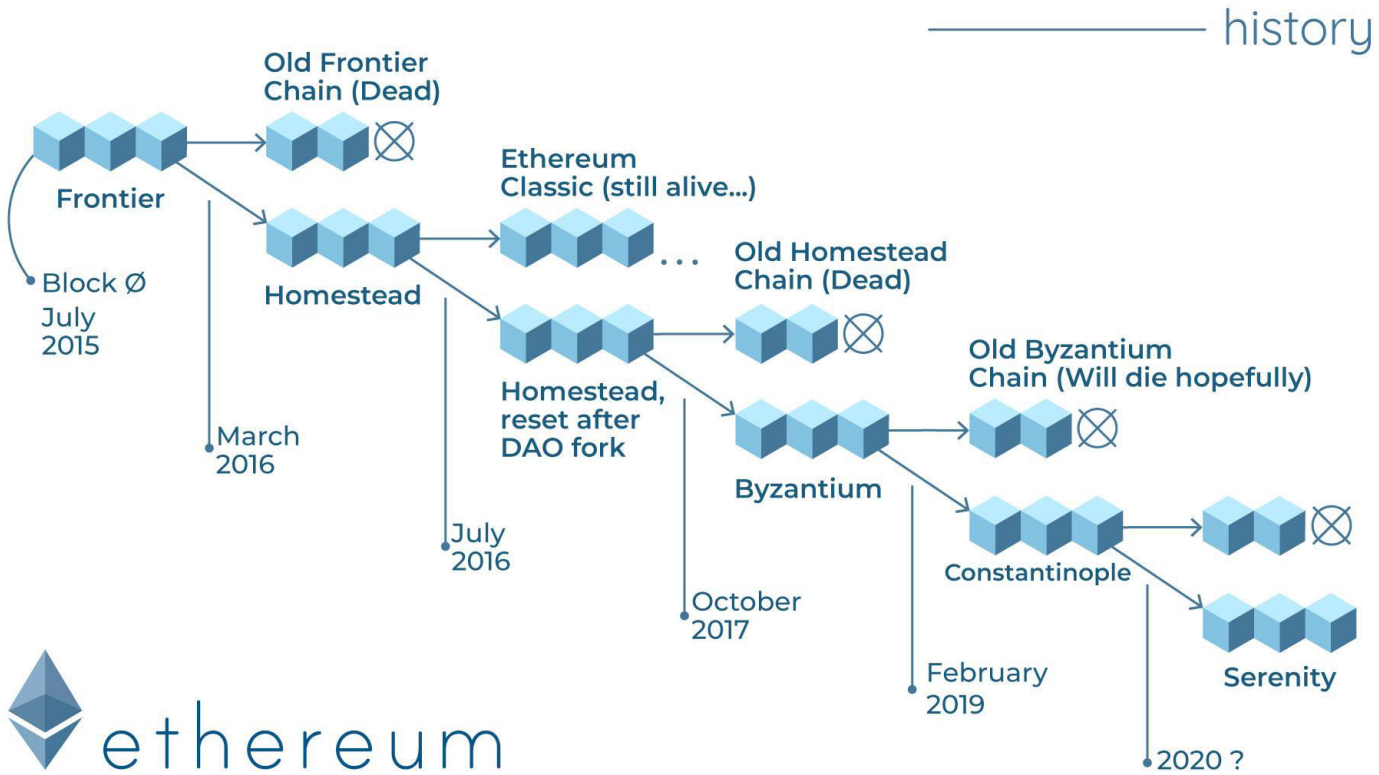
Ethereum

Many slides are from Campbell Harbey@Duke et al.
Some materials from Preethi Kasireddy@TruStory

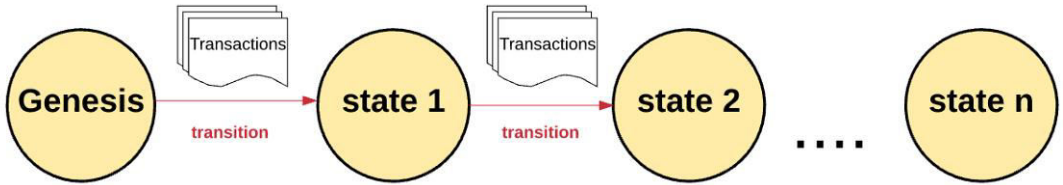
Roadmap by Ethereum Foundation



Source: Draglet

Important primitives

- Cryptography (similar to Bitcoin)
- Data structure/Algorithm in Computer Science
  - Patricia tree, Recursive length prefix, Bloom filter,...
- Blockchain
  - Accounts (Two types) and Wallets
  - Transactions
- Smart Contracts
  - Solidity
    - Language Used for Smart Contract Development



Hash Functions

- Bitcoin (BTC) uses SHA-256
- Ethereum uses Keccak-256
  - Similar to SHA-3 (variant)
  - Used for all hashing in Ethereum
  - Different from SHA-1 & SHA-2

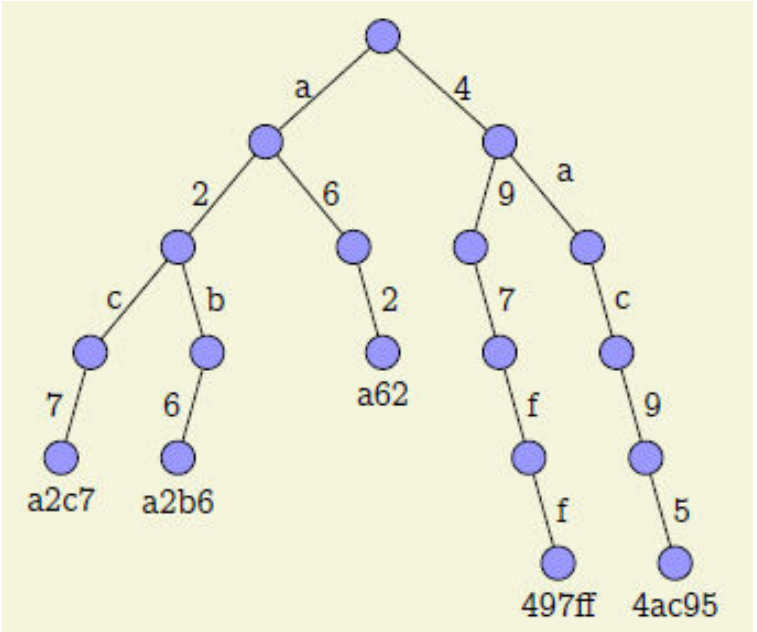
Digital Signatures (Digital Proof)

- Same use-case/cryptographic method (ECDSA) as BTC
  - Private key 256 bits
  - Public key 512 bits
- Signer uses private key to generate a signed message
- Signed message can be verified using the signer's public key
- Hashes are signed in Ethereum, not the data itself

Ethereum Blockchain

Key differences

- Blocks keep track of balances – not “unspent transaction outputs (UTXOs)” like BTC
- Merkle Patricia Tree (aka Trie)
- Will transition from PoW to PoS



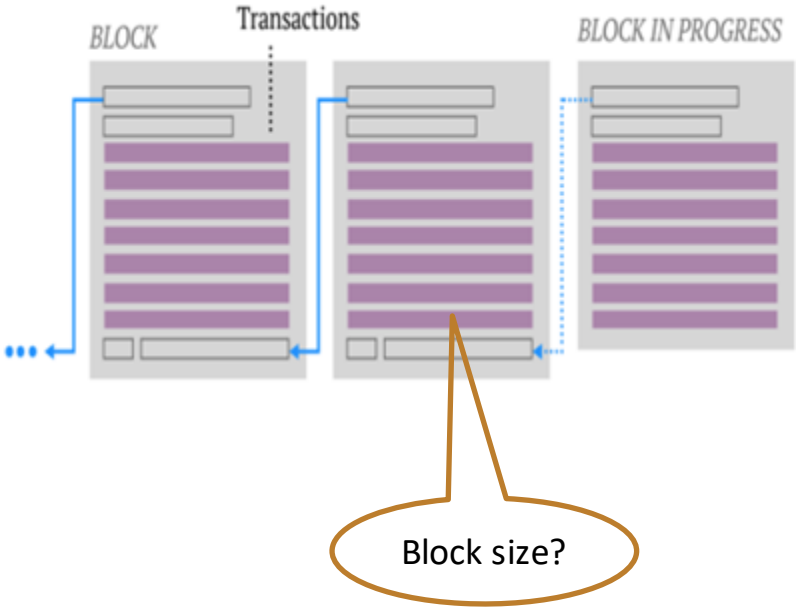
Source: Beige paper, Micah Dameron

Blockchain

Fully Distributed Database like BTC

Advantages:

- Highly Secure
- Transparent
- Immutable



Disadvantages:

- Scaling
- Performance

Ethereum Blockchain

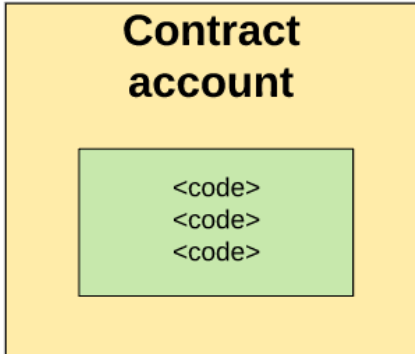
- All blocks visible like BTC
- However, blocks have a different structure than BTC
- Blocks faster than BTC and reward is different
  - Every 10~15 seconds
    - Difficulty field in block header
  - 5 -> 3 -> 2 ETH main reward
  - Miners can make a bit more by including uncle blocks (1/32 of an ETH each) up to maximum of two
  - Miners also get TX fee as gas

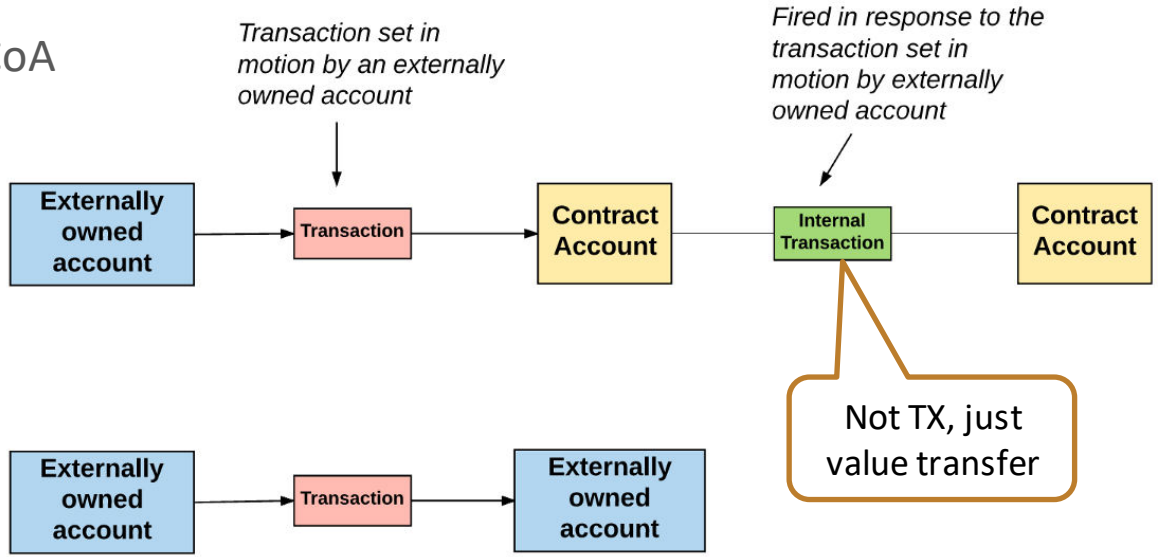
Smart Contracts

- Executable code
- Turing Complete
  - More precisely, Quasi-Turing complete (gas-limited)
- Function like an external account
  - Hold funds
  - Can interact with other accounts and smart contracts
  - Contain code
- Can be called through transactions (TXs)

2 kinds of accounts

- Each account has a state & a 20byte address
- Externally owned account (EOA)
  - Public/private key pair, no code associated
  - Send messages to other EOA or CoA
- Contract account (CoA)
  - Associated code
  - No private key
  - Can't initiate TXs
  - respond to incoming TXs





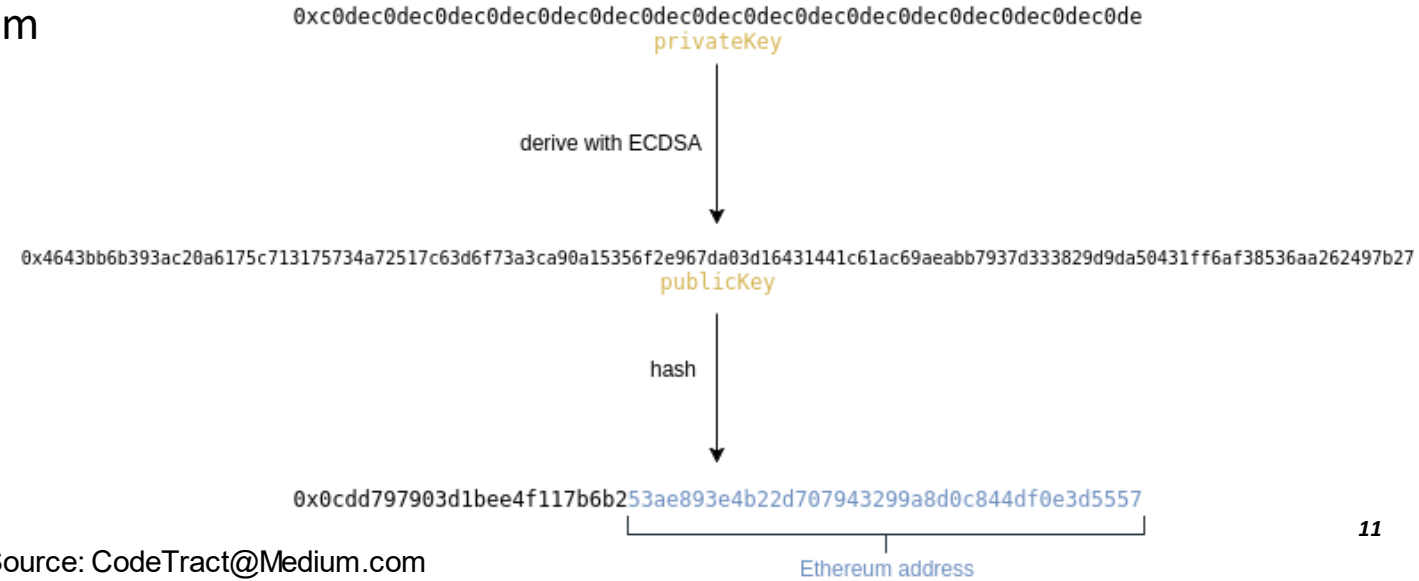
Example: EoA

Private Key: 0x2dcef1bfb03d6a950f91c573616cdd778d9581690db1cc43141f7cca06fd08ee

- Ethereum Private keys are 66 character strings (with 0x appended). Case is irrelevant. Same derivation through ECDSA as BTC.

Address: 0xA6fA5e50da698F6E4128994a4c1ED345E98Df50

- Ethereum Private keys map to addresses directly. Simply the last 40 characters of the Keccak-256 hash of the public key. Address is 42 characters total (append 0x to front)
- Beware! No checksum



Source: CodeTract@Medium.com

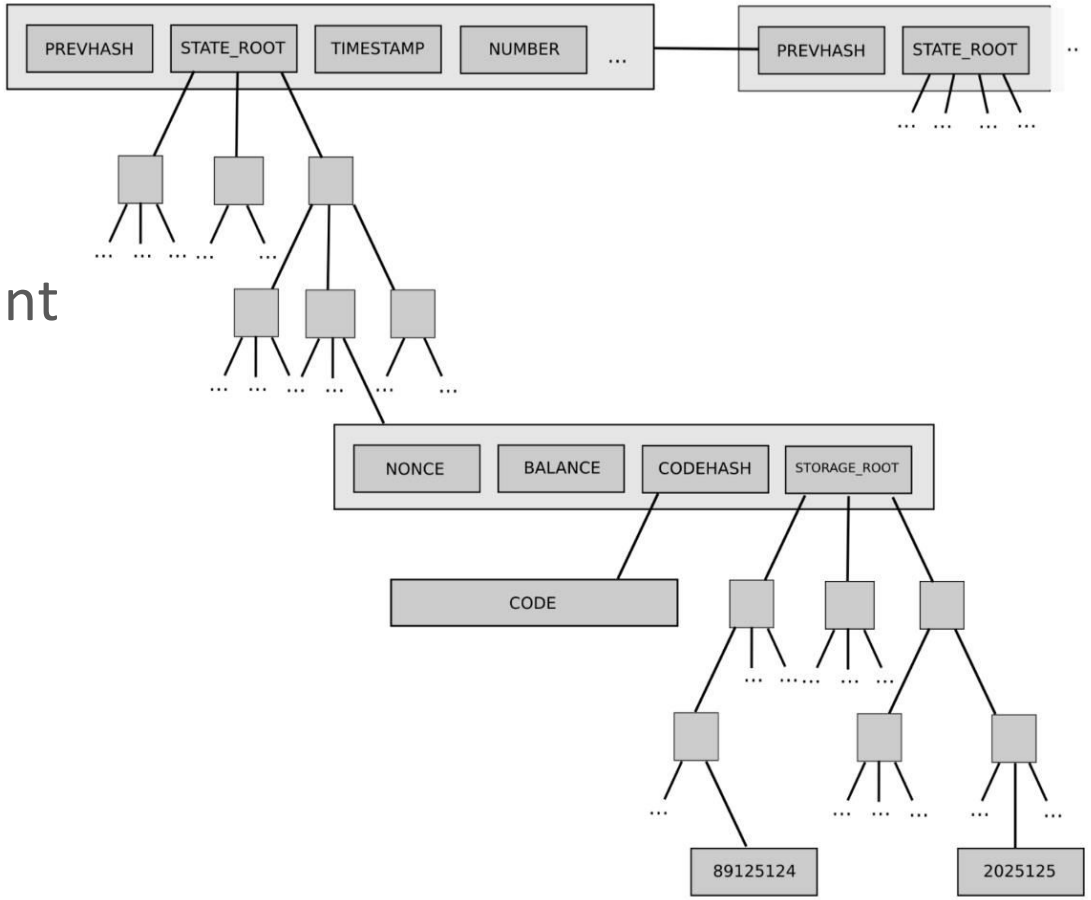
Account state

Different from PoW nonce

- Nonce: # of TXs sent from this address, # of contracts created by this address
  - Prevents replay attack
- Balance
- storageRoot: hash of the root node of a Merkle Patricia tree of “data of the contract”
- codeHash: hash of EVM code

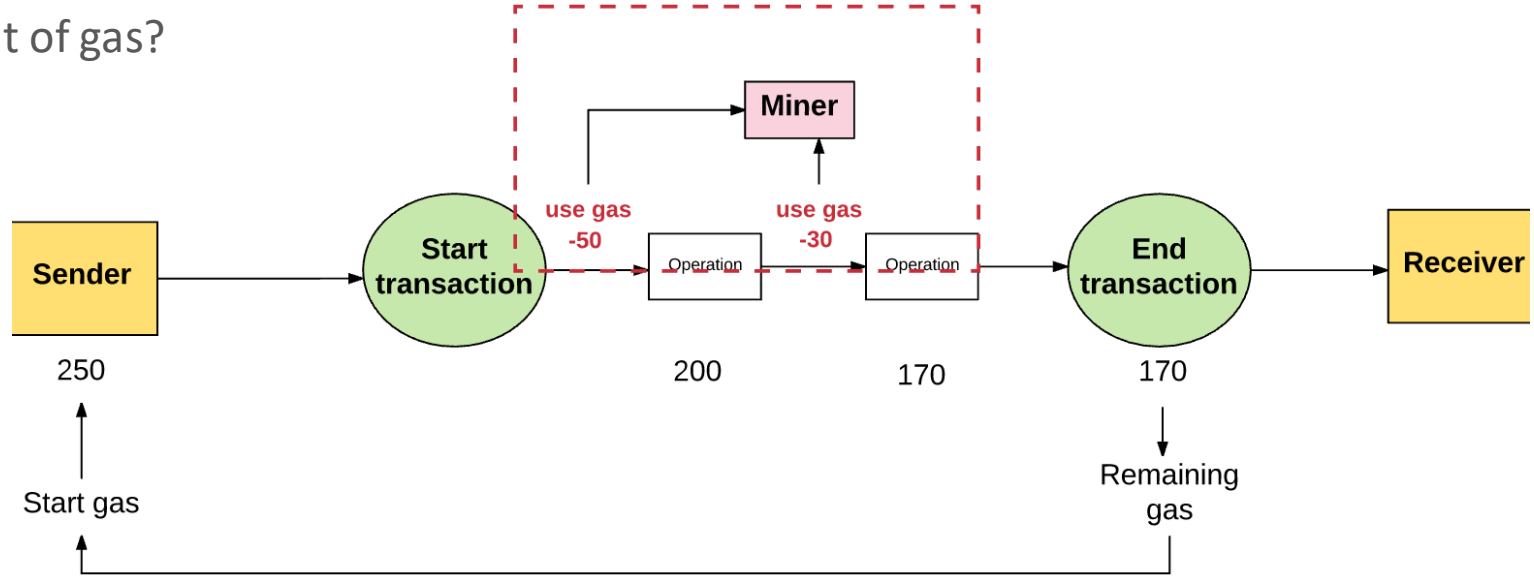
World state

- Aka system state
- Mapping between an account address and its state
- State trie is illustrated
  - E.g. code and data of a contract is stored



gas

- Every operation that occurs as a result of a transaction incurs a fee
  - Prevents DoS attack
- Gas is the unit used to measure the fees required for a particular operation
- Gas price is the amount of Ether you are willing to spend on every unit of gas
- The product of gas price and gas limit represents the maximum amount of Wei that the sender is willing to pay for executing a transaction
- out of gas?



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

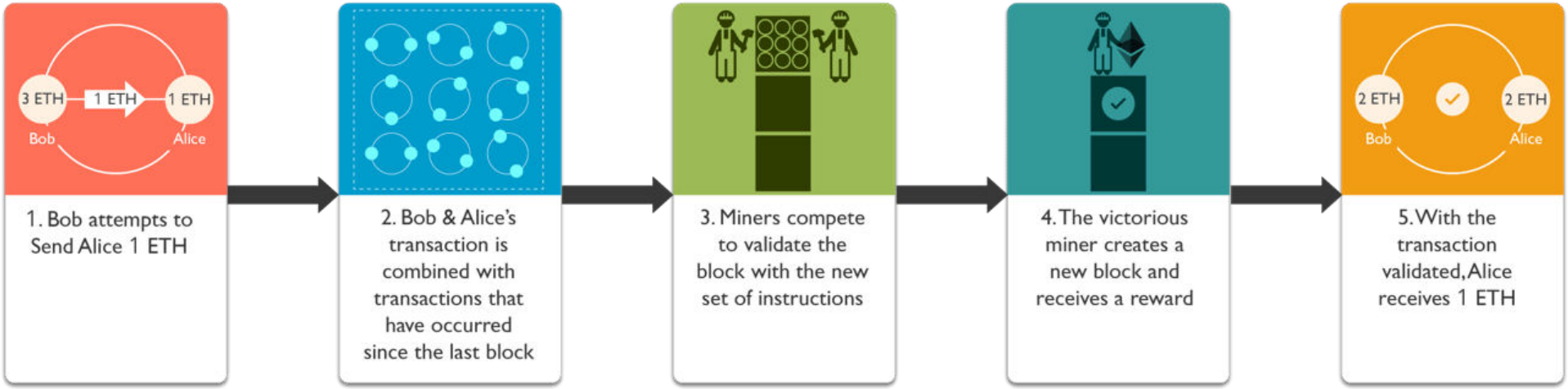
Gas Cost

- 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)

Transactions

- A request to modify the state of the blockchain
  - Can run code (contracts) which change global state
    - Contrasts with balance updates only in BTC
- Signed by originating account
- Types:
  - Send value from one account to another account
  - Create smart contract
  - Execute smart contract code

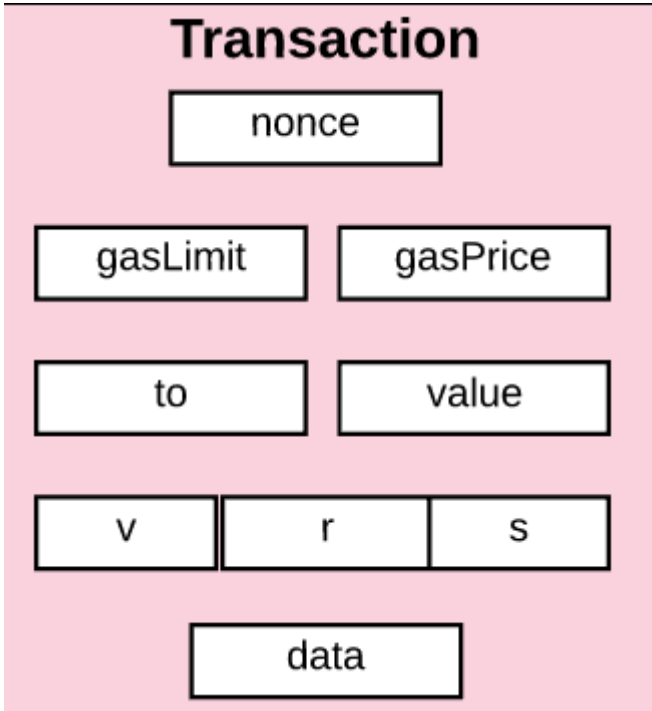
TX can transfer Ether: an illustration



Source: Paul@edureka

transactions

- 2 types: Message calls and contract creations
- Each TX has these components
  - Nonce: # of TXs sent by the sender
  - gasPrice, gasLimit
  - to
  - Value: amount of Ether to send
  - Signature (of the sender): v,r,s
  - Data
    - contract bytecode if contract creation TX
      - also called init
    - function selector and arguments if contract call TX



TX examples

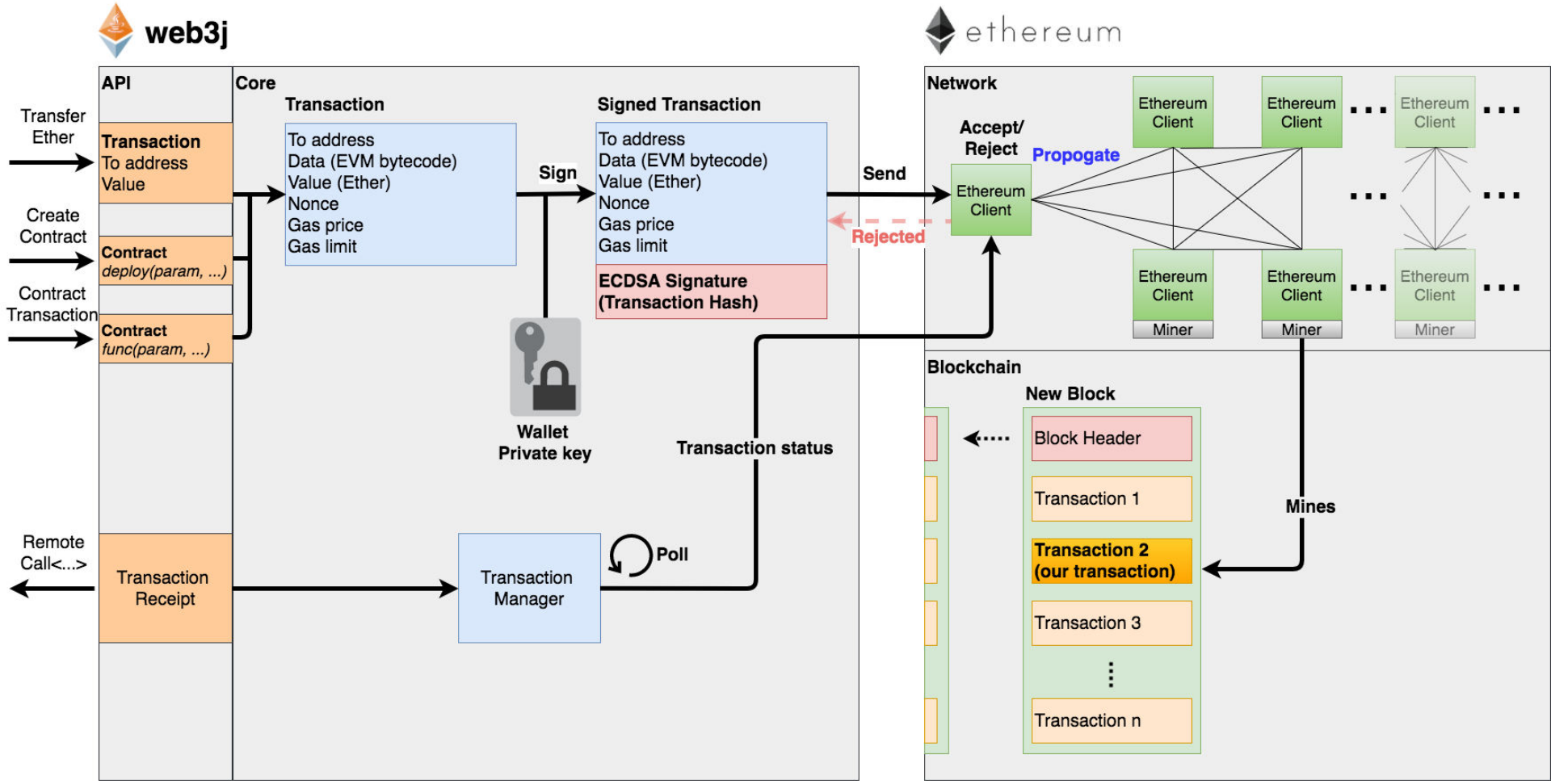
From	Fund sender, an EOA (20-byte address)
To	Fund recipient, another EOA (20-byte address)
Value	Amount, in weis
Data / Input	Empty
Gas Limit	Larger enough for an ether transfer transaction
Gas Price	To be determined by transaction initiator

From	Contract deployer, an EOA (20-byte address)
To	Empty
Value	Amount, in weis (if required by contract constructor)
Data / Input	Bytecode, plus any encoded arguments if required by constructor
Gas Limit	Larger enough for contract deployment
Gas Price	To be determined by transaction initiator

From	Function executor, an EOA (20-byte address)
To	Contract Address (20-byte address)
Value	Amount, in weis (if needed in contract function)
Data / Input	Function selector, plus any encoded arguments required by function
Gas Limit	Larger enough for contract function execution
Gas Price	To be determined by transaction initiator

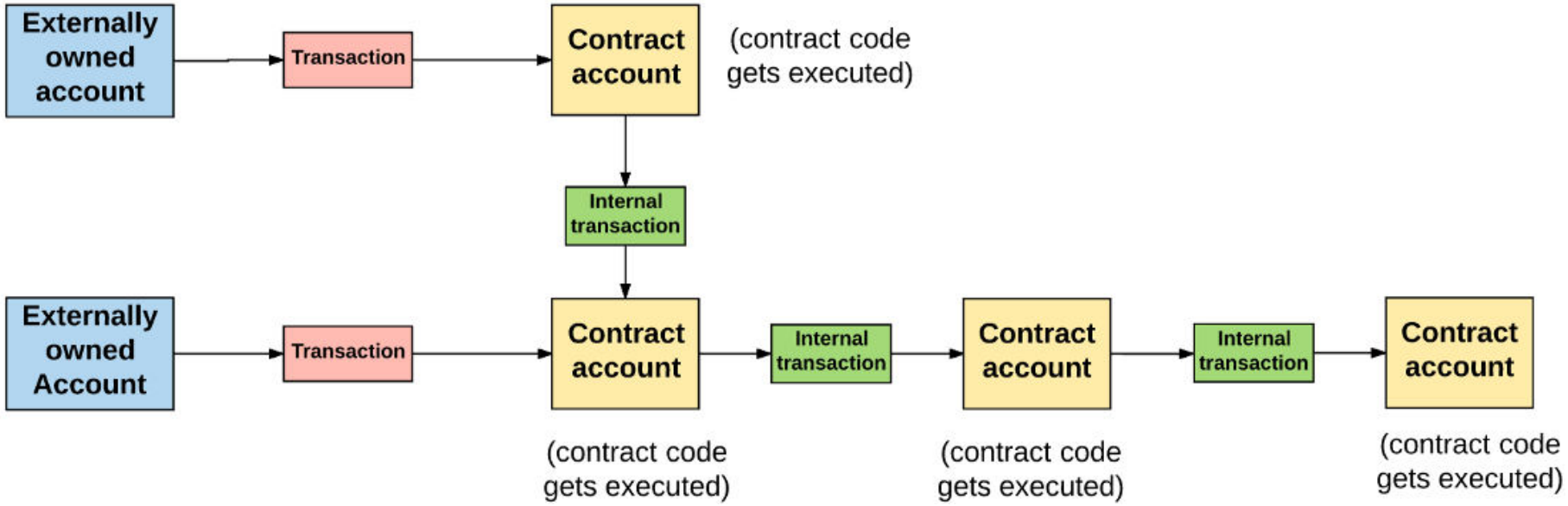
From address is derived from the public key, which is calculated from signature (v, r, s)

Source: web3j 4.1.0



TXs can interact

- Contracts interact one another via “messages” or “internal transactions” to other contracts

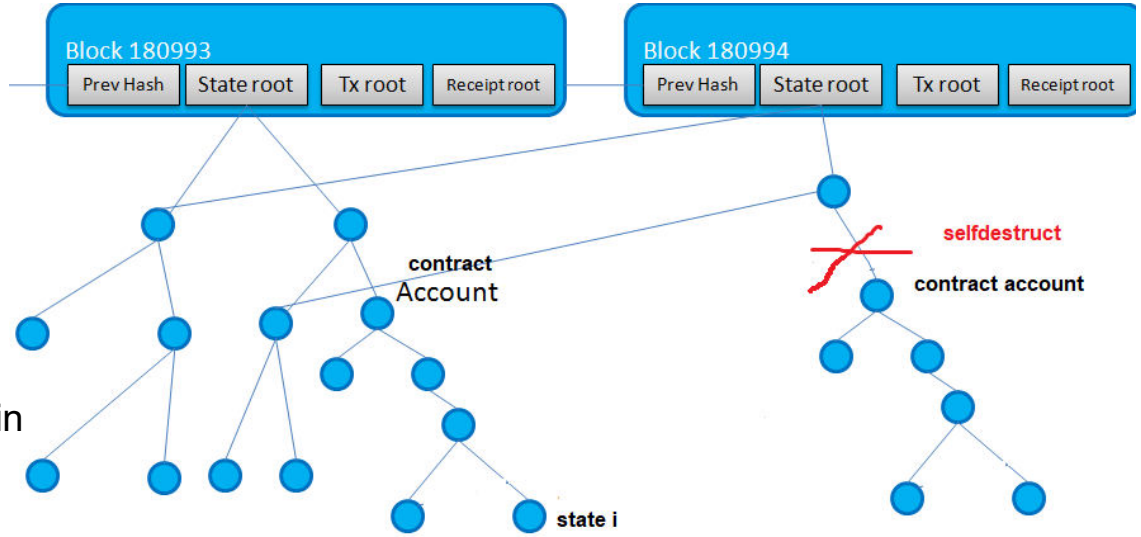


log

- Logs track and checkpoint TXs
- A contract provides pointers in logs by defining events
- A log entry has
  - Logger's account address
  - A series of topics that represent events
    - E.g. topic for a function Hello(uint256 some-name) is keccak256('Hello(uint256)')
  - Any data associated with events
- Logs are recorded in TX receipts

TX execution

- While executing a TX, Ethereum keeps track of substate
  - The substate changes for each operation
- Substate has
  - Self-destruct set: account to be discarded after execution
  - Log series: checkpoints of EVM's code execution
  - Refund balance



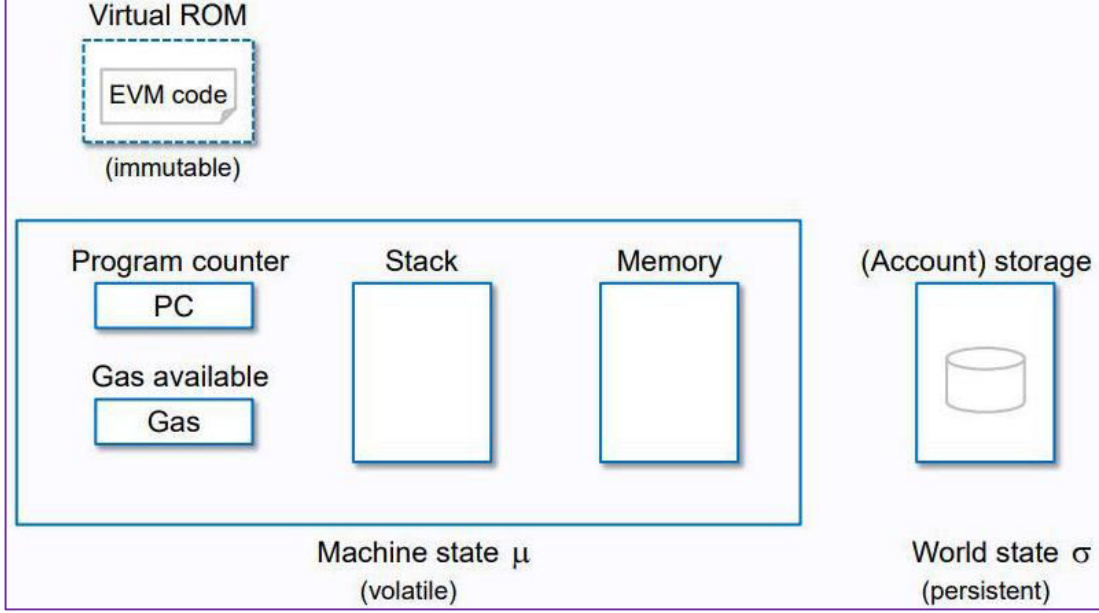
Source:Badr Bellaj@MChain

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

Execution model

- EVM is a Turing complete VM
- Bound by gas
- Stack-based
- at every operation, EVM checks
  - System state
  - Remaining gas
  - the account owning the code
  - Sender of the TX who triggers
  - Block header
  - ...
- EVM computes system state and machine state
  - Machine state: available gas, PC, memory contents, stack contents,...

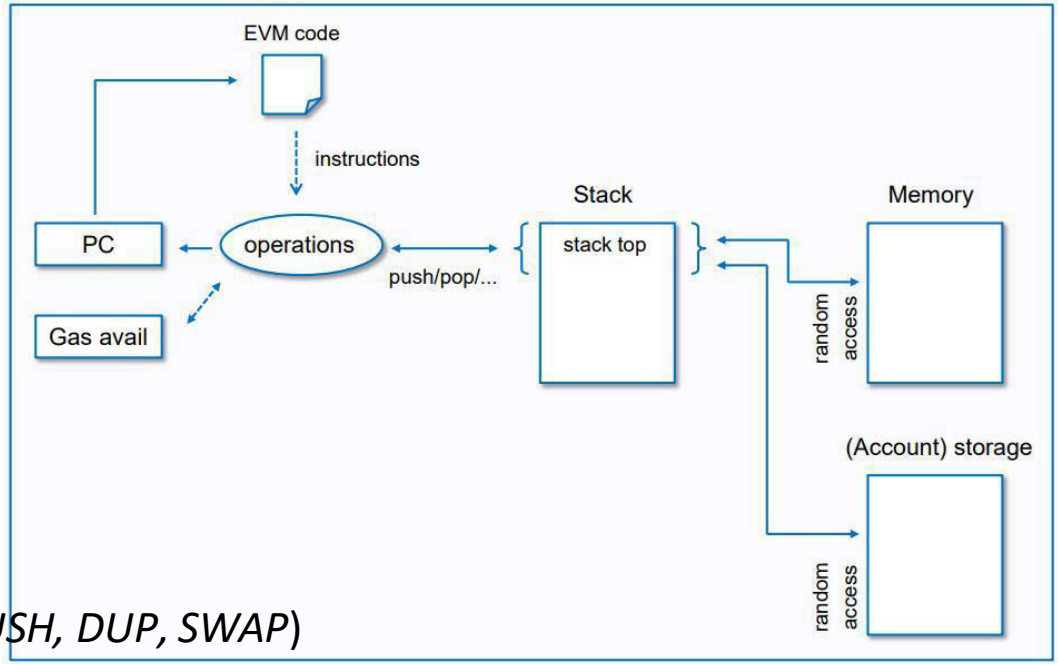


Source: AMBCrypto

EVM operation

- Bytecode (or Opcode)
- Volatile memory
- Non volatile storage
- Operands are in stack to be processed

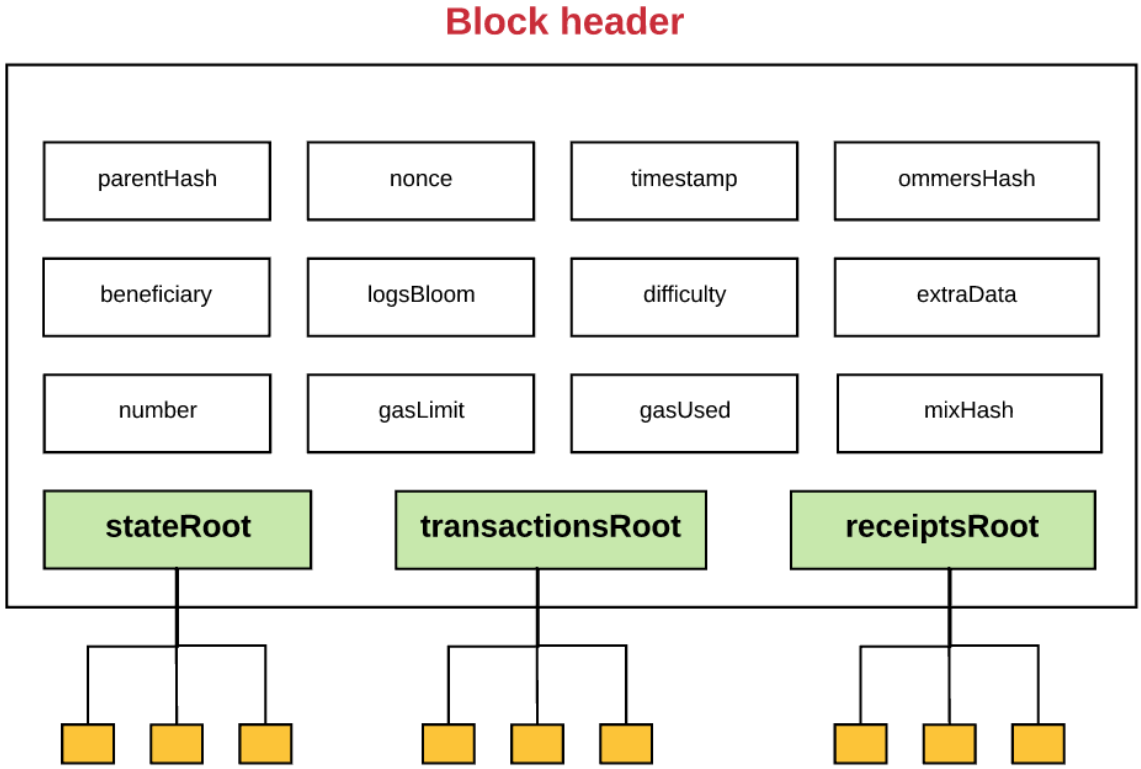
- Opcodes
  - Stack-manipulating opcodes (*POP, PUSH, DUP, SWAP*)
  - Arithmetic/comparison/bitwise opcodes (*ADD, SUB, GT, LT, AND, OR*)
  - Environmental opcodes (*CALLER, CALLVALUE, NUMBER*)
  - Memory-manipulating opcodes (*MLOAD, MSTORE, MSTORE8, MSIZE*)
  - Storage-manipulating opcodes (*SLOAD,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



Source: AMBCrypto

Block structure

- parentHash
- ommersHash
- Beneficiary: miner
- stateRoot: from state trie
- transactionsRoot: from TX trie
- receiptsRoot
- logsBloom: bloom filter
- Difficulty
- Number: count of current block
- gasLimit
- gasUsed
- Timestamp
- extraData
- Nonce, mixHash: prove the block has done computation



Ethereum Blockchain

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
- Aimed to decrease centralization and reward work

Ethereum Blockchain

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:
  - $(U_n + (8 - B_n)) * Current\_Reward / 8$, where U_n and B_n are uncle and block numbers respectively.
  - Example (1333 + 8 - 1335) * 2/8 = 1.75 ETH

Ethereum Blockchain

Blocks faster than BTC and reward is different

- Uses Ethash mining algorithm (different from Bitcoin's)
  - a large, randomly generated dataset (order of GBs)
    - Directed acyclic graph (DAG)
  - fetch random data from DAG, compute randomly selected transactions from any block & return the hash
  - Memory-hard or memory-bound
  - Helps mitigate ASIC and GPU advantages
- Difficulty is adjusted every block (not every two weeks)

Ethereum Nodes

- Validate all transactions and new blocks
- Operate in a P2P fashion
- Each contains a copy of the entire Blockchain
- Light clients - store only block headers
  - verify the proof of work on the block headers
  - Ask a full node to download only the "branches" associated with TXs relevant

Ether Denominations

- Wei - lowest denomination
  - Named after Wei Dai - author of b-money paper (1998), many core concepts used in BTC implementation
  - 1/1,000,000,000,000,000,000 (quintillion)
- Szabo - next denomination
  - Named after Nick Szabo - author of Bit-Gold
- Finney – 2nd highest denomination
  - Named after Hal Finney - received first Tx from Nakamoto

Multiplier	Name
10^0	Wei
10^{12}	Szabo
10^{15}	Finney
10^{18}	Ether

http://www.weidai.com/bmoney.txt

PoW vs. PoS

Ethereum in the process of moving to Proof of Stake

- This approach does not require large expenditures on computing and energy
- Miners are now “validators” and post a deposit in an escrow account
- The more escrow you post, the higher the probability you will be chosen to nominate the next block
- If you nominate a block with invalid transactions, you lose your escrow

PoW vs. PoS

Ethereum in the process of moving to Proof of Stake

- One issue with this approach is that those that have the most ether will be able to get even more
- This leads to centralization eventually
- On the other hand, it reduces the chance of a 51% attack and allows for near instant transaction approvals
  - Mining power can be hidden in PoW
  - With PoS, we can track who is the winner (traceability)
- The protocol is called Casper and this will be a hard fork

https://blockonomi.com/ethereum-casper/