K B A

ETHEREUM VIRTUAL MACHINE

INTRODUCTION TO ETHEREUM VIRTUAL MACHINE

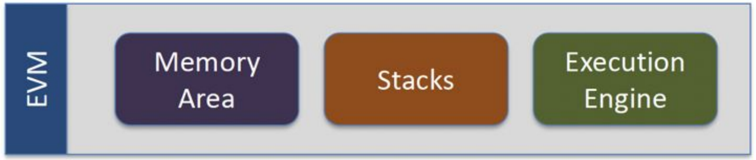
The objective of this session is to get a basic understanding of Ethereum Virtual Machine & some basic concepts, before getting into application development

- What is a Virtual Machine
- Why a Virtual Machine is necessary
- Ethereum state & accounts
- How the Ethereum Virtual Machine operates
- Block & Transaction
- Ethereum transaction life cycle

What is EVM

"Runtime environment for smart contracts in Ethereum"

- It is sandboxed and isolated; meaning the code running inside EVM has no access to network, filesystem, or other process
- It is a software that can execute Ethereum bytecode
- Smart contracts executed in EVM have limited access to other smart contracts (yet vulnerabilities are still rampant)
- Follows the EVM spec (Eth protocol)
- Runs on a computer or server (in the node)
- Implemented in multiple languages



EVM

Executes code of arbitrary algorithmic complexity

- Smart contracts are compiled into bytecodes > EVM reads and executes
- Developers create apps in familiar languages (Javascript, Python) that interact with smart contracts stored in ethereum, through transactions, which are executed by the EVM
- EVM has low level operations defined as bytecodes, which is what smart contract are compiled to.
- ADD, MUL, CREATE, BALANCE etc are some opcode representation of low level operations

Why EVM

Why is a Virtual Machine needed?

Anything that runs on a blockchain needs to be immutable and must have the ability to run through multiple nodes without compromising on its integrity.

To accomplish this, 3 things are needed:

- Deterministic
- Terminable
- Isolated

Deterministic

A program is deterministic if it gives the same output to a given set of inputs every single time, from any number of computers.

When does a program act in an un-deterministic manner?

Example: 3+1 = 4

Any computation of 3 + 1 will = 4

- When calling un-deterministic system functions
- **Un-deterministic data resources:** If a program acquires data during runtime and that data source is un-deterministic then the program becomes un-deterministic
    - Eg. Suppose a program that acquires the top 10 google searches of a particular query. The list may keep changing.
- **Dynamic Calls:** When a program calls a second program it is called dynamic calling. Since the call target is determined only during execution, it is un-deterministic in nature

Turing Completeness

- Machine that can simulate any computer algorithm
- Allows for loops, makes Ethereum vulnerable to halting problem
- Cannot determine whether a program will run infinitely
- System fees (gas) prevent this from occurring > protects network from malicious actors trying to put network in infinite loop

Terminable

Halting Problem: Unknowing whether or not a program can execute its function within a given time limit.

With smart contracts, there needs to be a way to externally “kill” the contract and to not enter into an endless loop which would drain network resources:

- Step and Fee Meter: A program can simply keep track and limit the number “steps” it has taken
- Fee-meter. Here the contracts are executed with a prepaid fee. Every instruction execution requires a particular amount of fee. If the fee spent exceeds the pre-paid fee then the contract is terminated.
- Timer: Here a predetermined timer is kept. If the contract execution exceeds the time-limit then it is externally aborted.
- Turing Incompleteness: A Turing Incomplete blockchain will have limited functionality and not be capable of making jumps and/or loops. Hence they can’t enter an endless loop.

Isolated

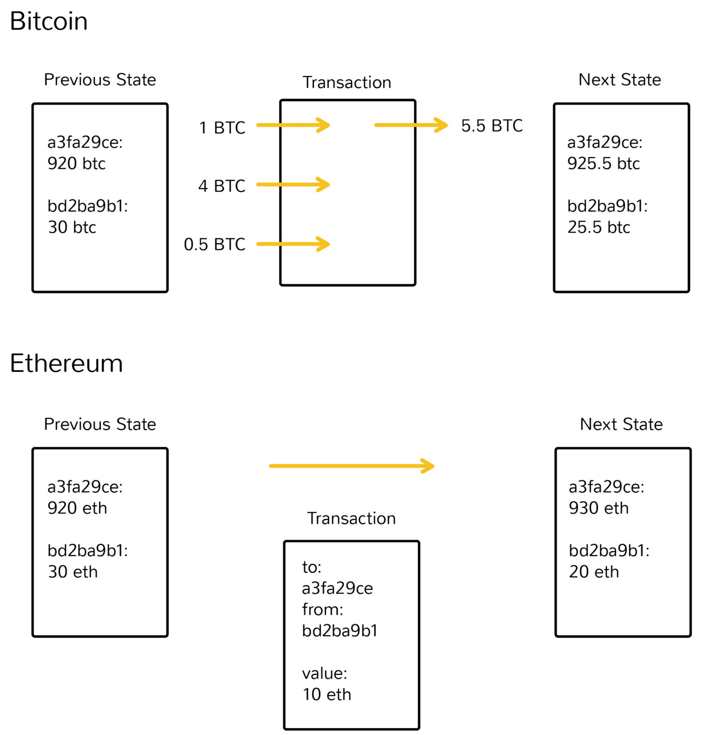
- In a public blockchain, anyone and everyone can upload a smart contract.
- However, because of this the contracts may, knowingly and unknowingly contain virus and bugs.
- If the contract is not isolated, it may have the ability to hamper the whole system.
- Hence, it is critical for a contract to be kept isolated in a sandbox to save the entire ecosystem from any negative effects.
- It is common in public blockchains to only allow certain functions to be called by contracts.

State

Network keeps track of ”state”:

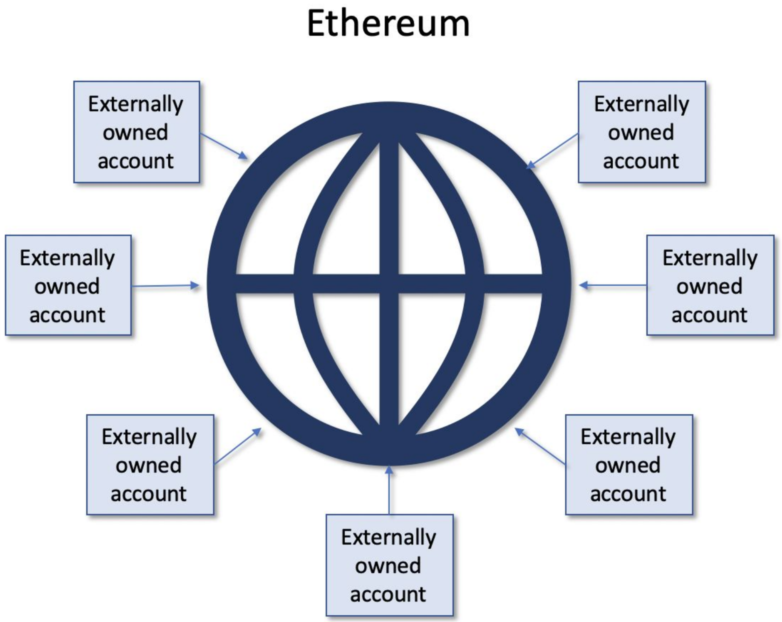
- *State can be defined as the present condition of a program/application/database etc.*
- *For ethereum, it is the current information for all applications and transactions, balances, etc.*

Instead of transactions being the basic unit (bitcoin), Ethereum uses the account as the base unit



Accounts

Ethereum State = Contains millions of objects called “accounts” that have the ability to maintain an internal database, execute code and speak to each other



Accounts

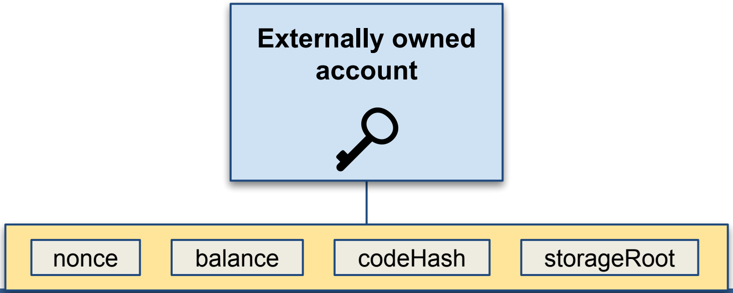
Global shared state of Ethereum is made of small objects (accounts)

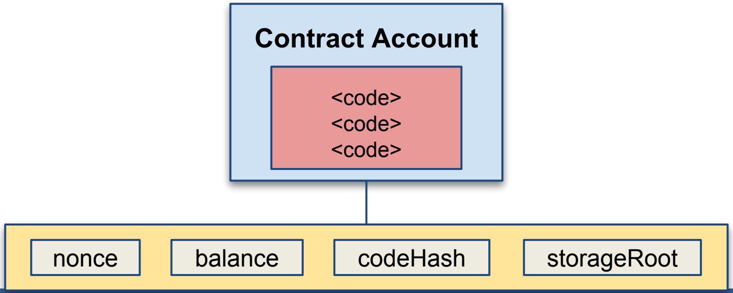
- Two types of accounts
    - EOA – externally owned account controlled by private key (no code associated)
    - Contract – account has its own code, and is controlled by it

Each account is uniquely identified by its 20-byte address

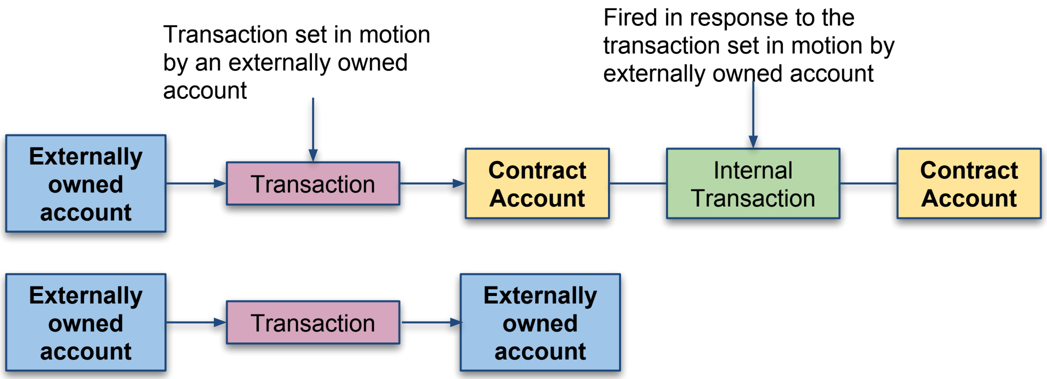
What does an account contain?

1. **Nonce:** count of the number of outbound transactions, starting with 0. for contract accounts it's the number of contracts created by the account.
2. **Balance:** the amount of ether in the account.
3. **StorageRoot:** the hash associated with the storage of the account (root node of Merkle Patricia) empty by default. Stays empty for EOA.
4. **CodeHash:** the hash of the code governing the account. For contracts, it is the hash of the codebase, for EOA this field is the hash of the empty string ""





Accounts Transaction Process



Any actions that occur on Ethereum are always set in motion from Externally Owned Accounts!

Execution environment is lifeless until external user triggers action and sets Ethereum in motion

EOA to EOA : transfer of funds

EOA to Contract: contract activates based on input and runs its code

Contract to Contract: send and receive messages to execute internal processes. Cannot initiative a transaction by themselves

Accounts vs Wallet

Ethereum Account = has a private key and account address pair

- Native to ethereum & resides in ethereum state
- Private key can make transactions that proves ownership of the account using a digital signature
- Does not require gas to create an account

Accounts vs Wallet

Ethereum wallet = software written for ethereum

- Not native to ethereum
- Interacts with a node (directly or indirectly)
- Can be used to manage ethereum accounts
- The software provides key management and other services

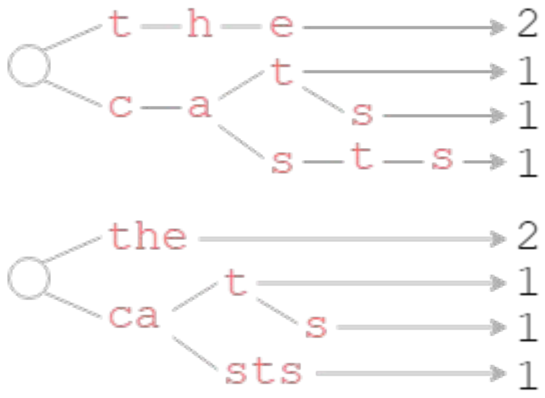
Merkle Patricia Trie vs Merkle Tree

Trie and Tree are used synonymously

Merkle Patricia Trie is just a more compact version of a Merkle Tree

The compact MPT (bottom) contains fewer nodes, as it decomposes keys only at points of difference.

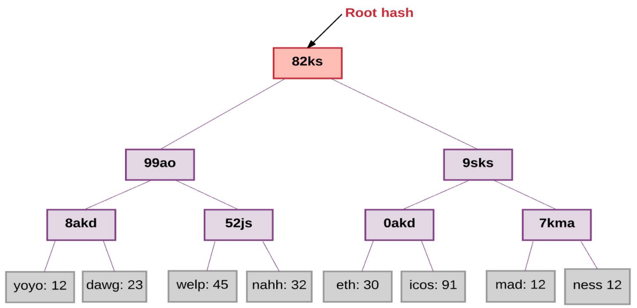
Since storage is so expensive on Ethereum, this allows for more efficient use of data.

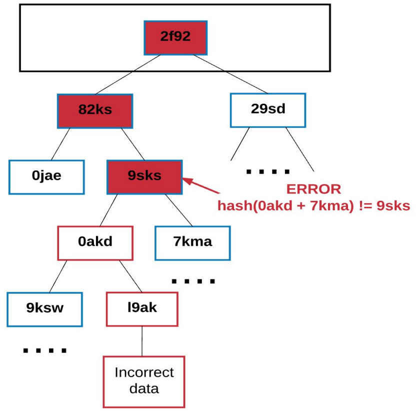


Ethereum World State

Global State = mapping between account addresses and its states

- Stored in a Merkle Patricia Tree (data structure)

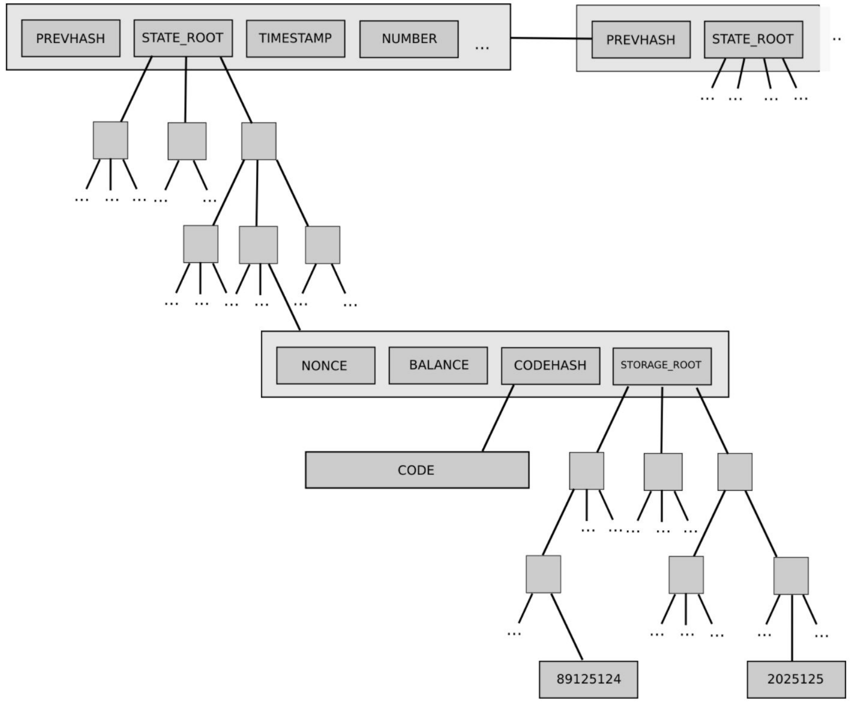




The leaf nodes are accounts and contains the details nonce, balance, storage root and code hash

Ethereum World State … Cont’d

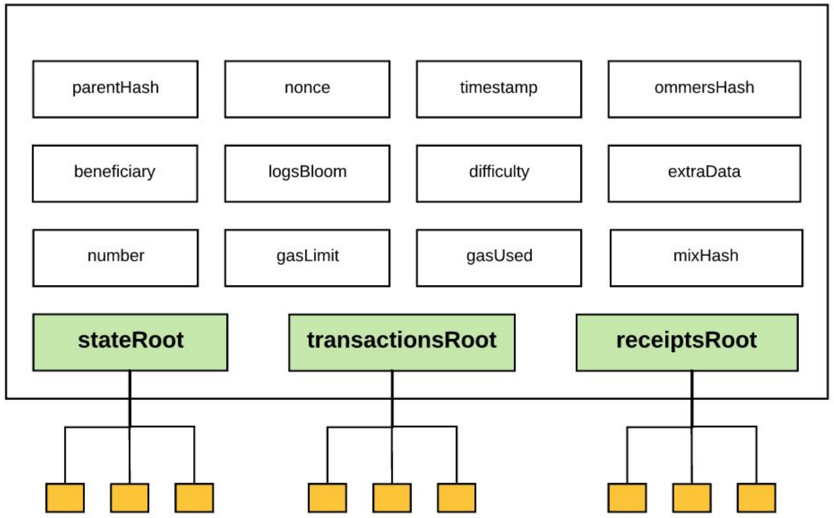
- Key is required for each value stored within tree
- Starting from root node, key tells which child node to follow to get the corresponding value, which is stored in the leaf node
- Key/value mapping for state tree is between addresses and their accounts (including 4 aspects we described earlier)
- Storageroot is root of a tree in itself



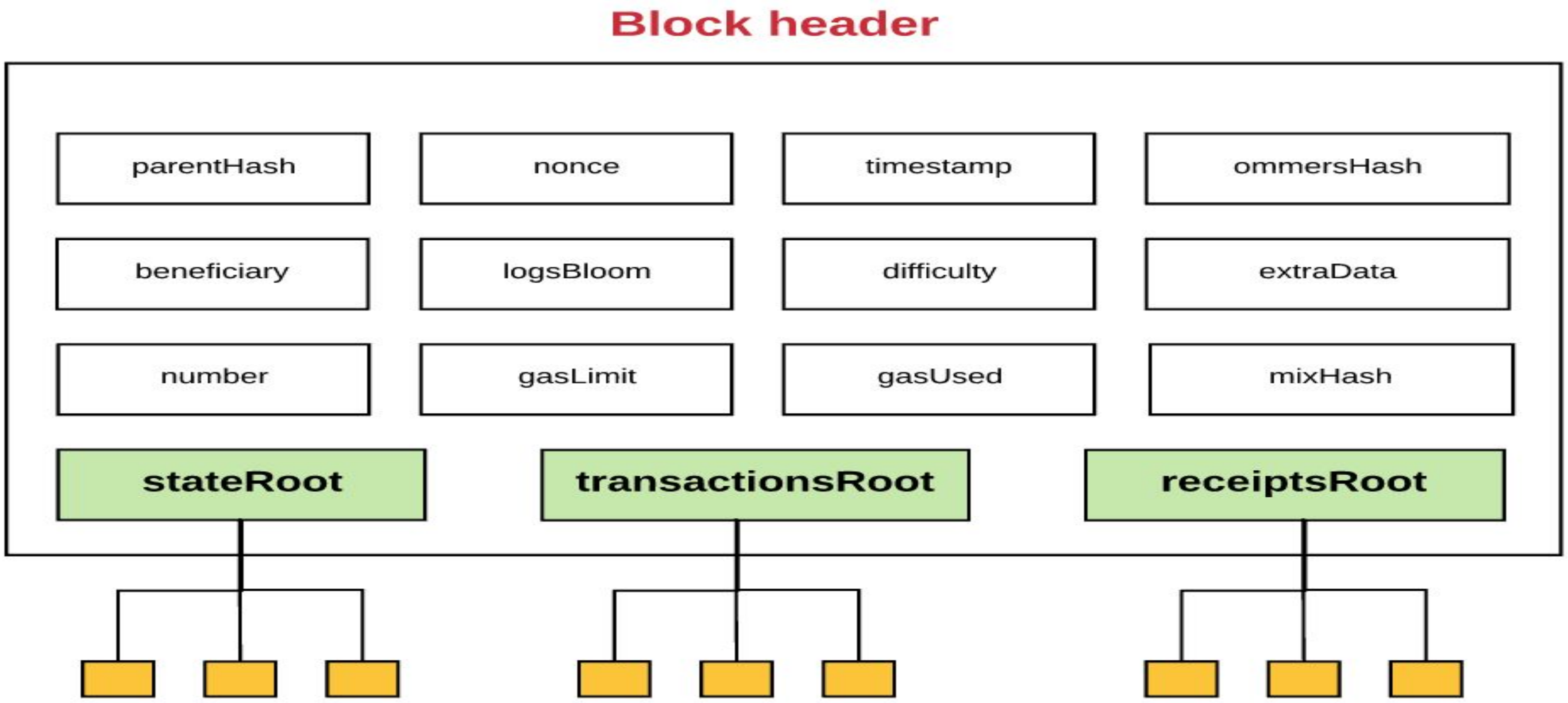
Ethereum World State … Cont'd

- Same structure is used to store transactions and receipts
- Every block has a header – stores hash of root node of three different merkle trees including (state trie, transactions trie, receipts trie)

Block header



Ethereum Block



Ether (ETH)

Native currency of ethereum, used to pay the computational resources needed to run an app on the decentralized network

- Digital bearer asset
- Not a security, although it ICO'd
    - Ex. taxi cab badges
- Used to read/write to Ethereum, known as digital "oil"
    - Costs are based on computational requirements

Gas

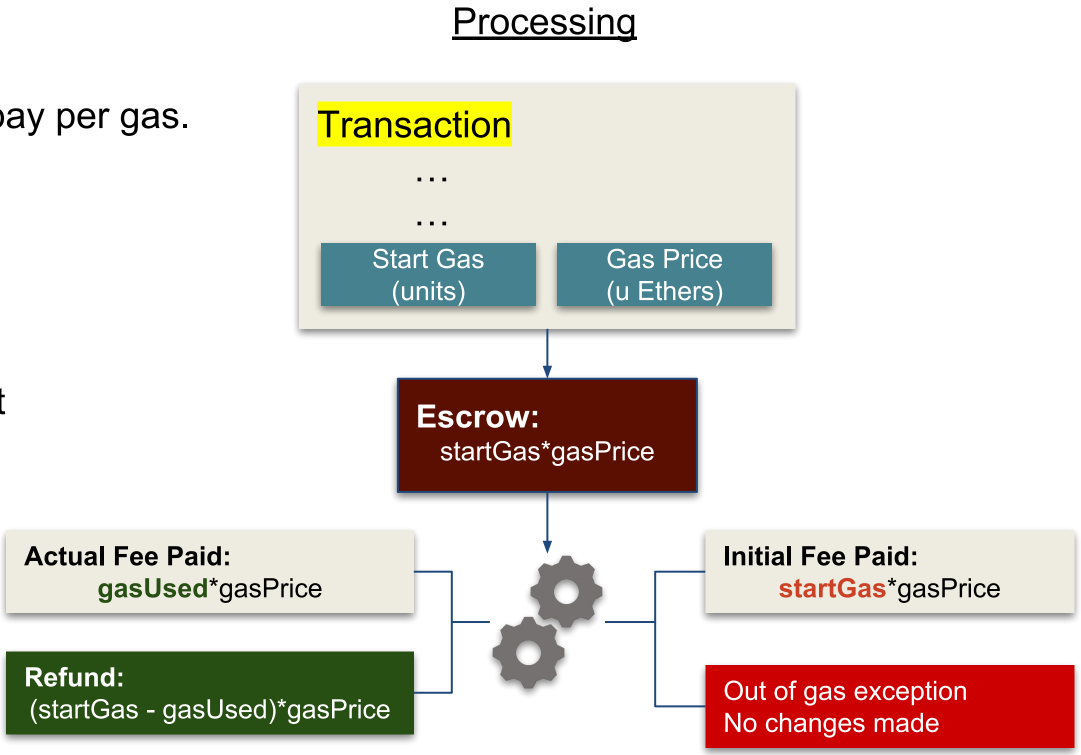
- Gas is the resource we need, to do payment transactions/run smart contract code/store data etc in ethereum
- Gas cost for each single operation is fixed (as part of the ethereum protocol, and changed as part of improvements)
- We need to “buy” gas (indirectly), when making transactions, using ether.
- Txn fee (payed in ether) = total gas required * gas price

Gas Prices

- Gas price is dynamic – subject to market conditions
- Measure of how badly user wants to pay per gas.
- Gas price is specified in Gwei/gas (1 Gwei = 0.000000001 ETH)
- These fees are paid out to the miner
- Miners prioritize higher gas prices first (they make more money)

gasUsed:
summed cost of executed instructions

gasPrice:
User specified amount


Processing
Transaction
…
…
Start Gas (units)
Gas Price (u Ethers)
Escrow: startGas*gasPrice
Actual Fee Paid: gasUsed*gasPrice
Refund: (startGas - gasUsed)*gasPrice
Initial Fee Paid: startGas*gasPrice
Out of gas exception
No changes made

Calculating Gas Costs

OPCODES = low level operation in EVM
Ex. ADD – add two integers (3 gas)
BALANCE – pull balance of an account
CREATE – creates a new contract
MUL – multiply two integers (5 gas)

Each has a gas costs associated in order to function
All transactions cost 21000 gas as a base

- Gas OPCodes and cost: https://github.com/djrtwo/evm-opcode-gas-costs

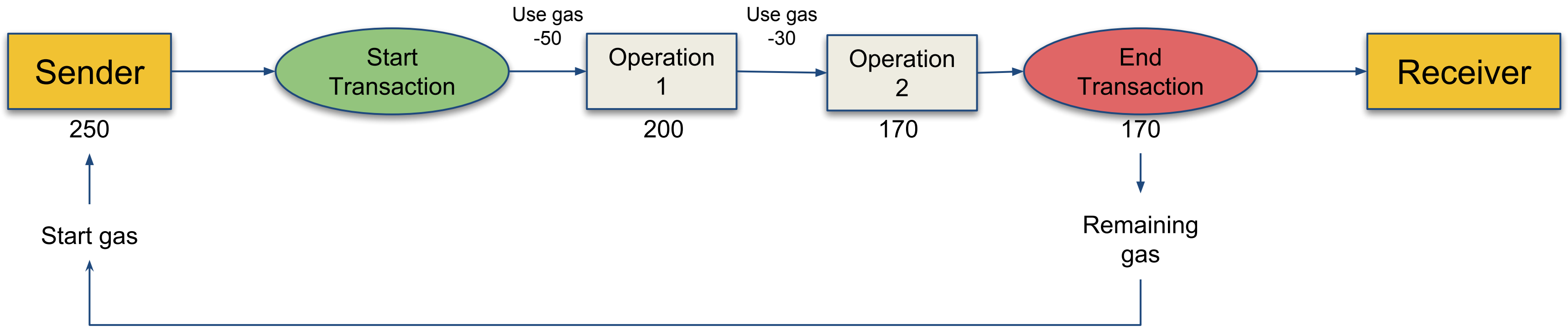
Fee Calculation

gasUsed	=	Instructions executed (summed up gas)
gasPrice	=	User specified in the transaction
		Miners decides the minimal acceptable price

Transaction Fee = gasUsed * gasPrice

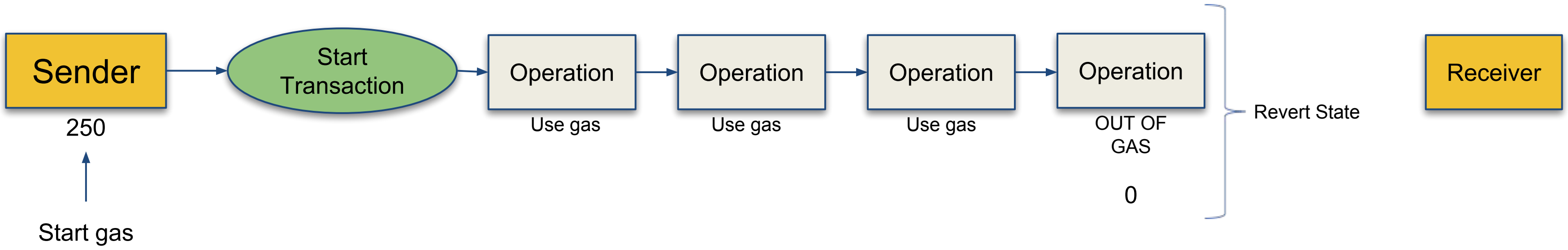
Wei

- "Wei" is the smallest unit of Ether, where 10^{18} Wei represents 1 Ether. One gwei is 1,000,000,000 Wei.
- Gas Limit * Gas Price = Max Transaction Fee
- When Gas Fee Is calculated, user must have enough ETH in acct to cover the max. when the actual transaction happens, user is refunded the difference.



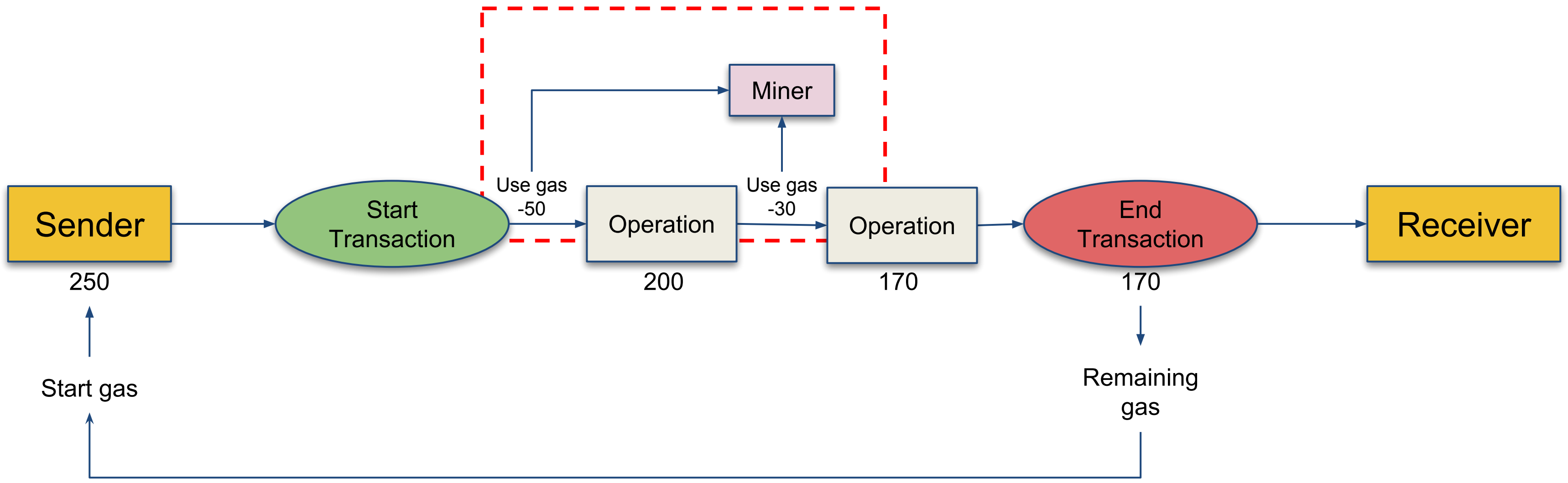
Running out of Gas

- If the sender does not provide necessary gas to execute the transaction, it will run out of gas and be rendered invalid
- State reverses to before transaction occurred (all changes reversed)
- Failing record gets recorded that attempt happened and where it failed
- Since work was done, gas is consumed and NOT REFUNDED



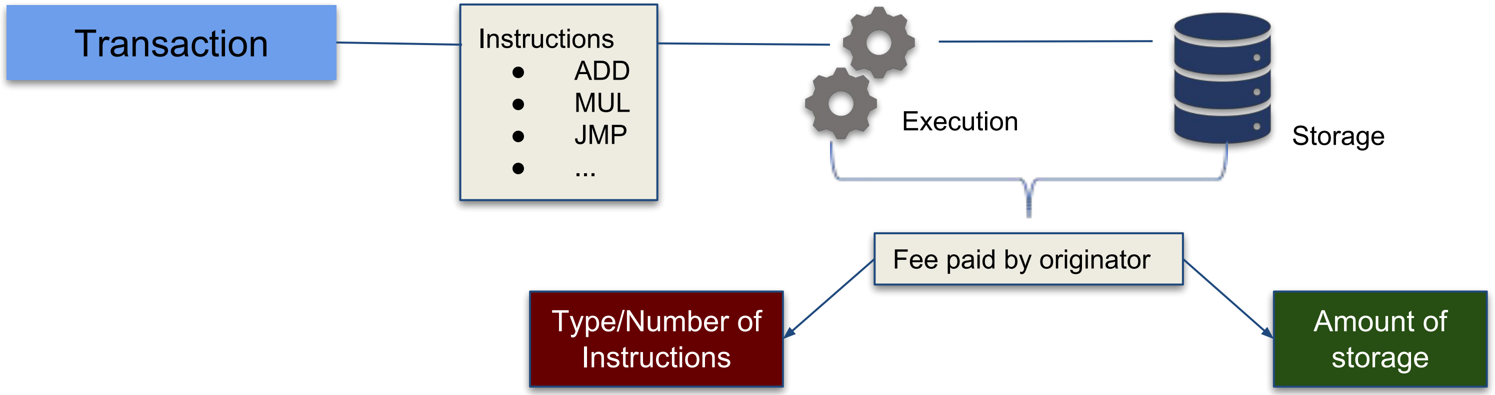
Running out of Gas … Cont’d

- Gas consumed goes to the Miner, since they were the ones doing the work validating the transaction



Current Status (Mainnet)

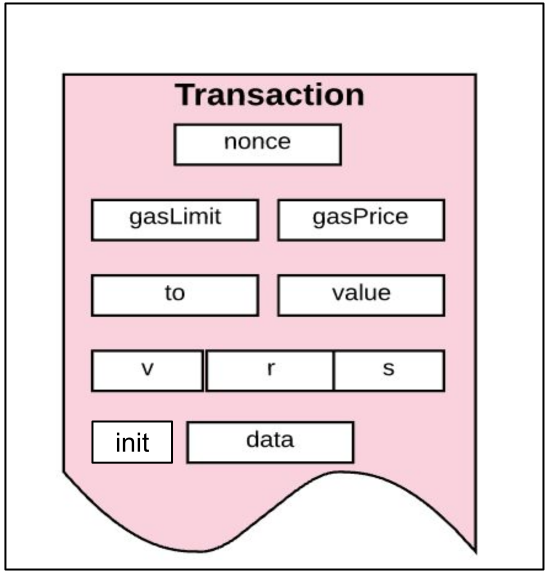
- Eth Gas station
- High level costs: spreadsheet
- ETH Converter: https://converter.murkin.me/



Transactions and Messages

- Transaction = cryptographically signed instructions generated by EOA
- It is then serialized and submitted to the Ethereum blockchain
- Two types: message calls and contract creation
- Message calls: when “data” or “value” is passed between accounts
- Txn has same data structure regardless of the type

- Nonce
- Gas Price
- Gas Limit
- To
- Value
- v,r,s
- Init
- Data

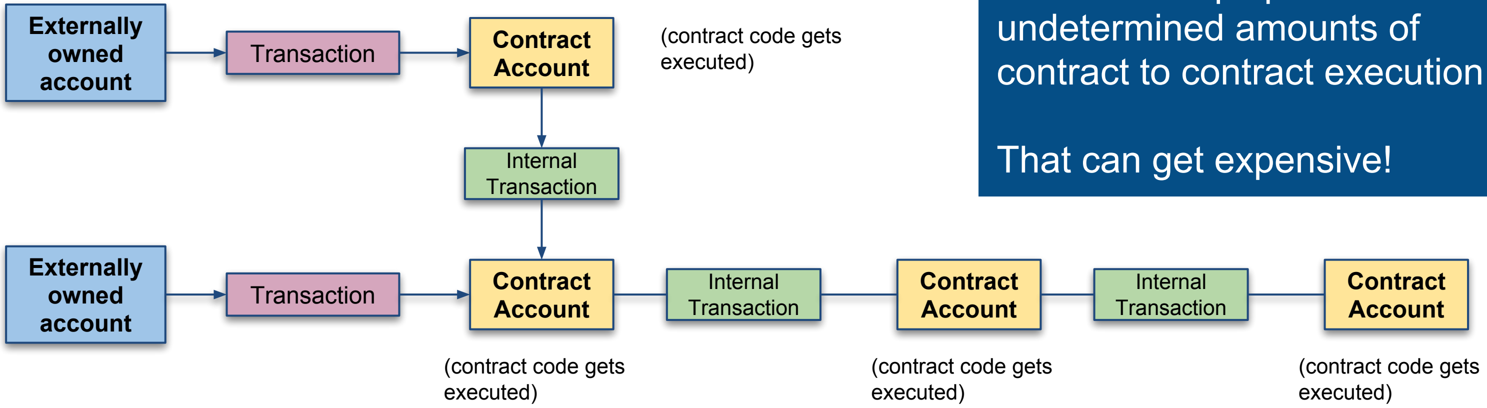


Contract to Contract Messages

- Communicate via “internal transactions”
- Not generated by EOA
- Virtual objects – not serialized and exist only with Ethereum
- Internal transactions do not contain a gas limit

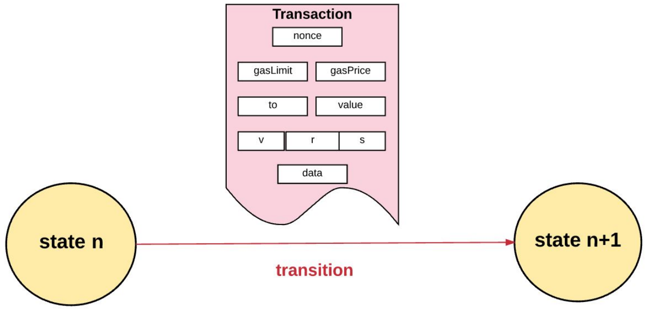
Gas Limit helps prevent undetermined amounts of contract to contract execution.

That can get expensive!



Transaction Execution - Validity

1. Must meet initial set of requirements in order to be executed
2. RLP format = Recursive Length Prefix
3. Valid transaction Signature
4. Valid Transaction Nonce
5. Gas Limit >= Intrinsic Gas Used
6. Sender's Account Balance covers Upfront gas cost



Intrinsic gas = Predefined gas fee **21,000** + Storage fee **4(X) + 68(Y)** + Contract creation **32,000**

Upfront cost = Gas Limit **50,000** x Gas Price **20 gwei** + Value **0.05 Ether**

Transaction Execution process - Message Calls

1. Upfront execution cost deducted from account
2. Nonce increases by 1
3. Remaining Gas Calculated
4. Execution: Transactional Computations
5. State Finalized (pending no invalid)
6. Unused gas refunded to sender + refund balance
7. Gas consumed given out to miner
8. Gas used added to block gas counter
9. All accounts in self destruct are deleted
10. Final State Reached and logs created for transaction

$$\textbf{Gas remaining =} \boxed{\begin{matrix}\text{Gas Limit}\\ 50{,}000\end{matrix}} - \overbrace{\boxed{\begin{matrix}\text{Predefined gas fee}\\ 21{,}000\end{matrix}} + \boxed{\begin{matrix}\text{Storage fee}\\ 4(X) + 68(Y)\end{matrix}} + \boxed{\begin{matrix}\text{Contract creation}\\ 32{,}000\end{matrix}}}^{\textbf{Intrinsic gas}}$$

Substate:

- Self Destruct Test
- Log Series
- Refund Balance

Transaction Execution process - Contract Creation

- New Contract Address Generated
- Set nonce to zero
- Ether value added to this new account balance, deducted from sender’s balance
- Set storage as empty
- Set contract’s codeHash as the hash of an empty string
- Account initialized; init code command used to create contract account
- Constructor executed
- Codehash is set
- If successful, a final contract creation cost is paid
- Unspent Gas refunded; achieve final state

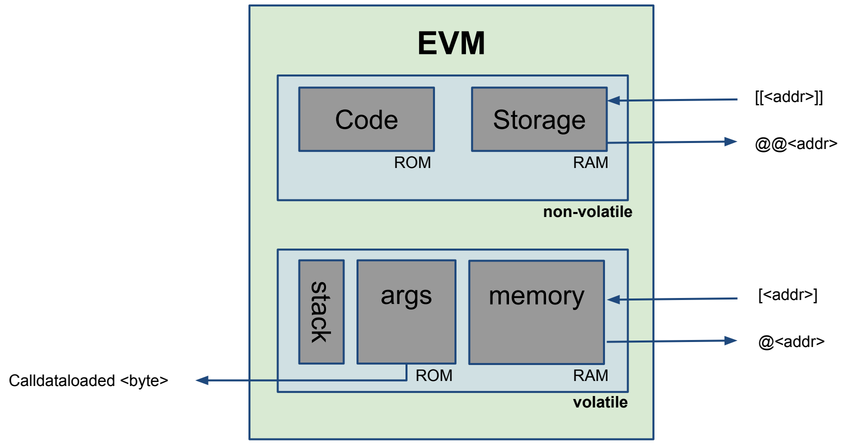
Message calls vs Contract creation

Message Calls have:

- No init code
- May contain input data (if provided by sender)
- After execution: contain output information

EVM - Memory, storage and the Stack

- The part of the protocol that handles processing (EVM)
- Stack based architecture: stack item size is 256 bit, stack max size 1024
- EVM has memory: items stored as word addressed byte arrays; volatile
- EVM has storage: non volatile, maintained as part of state
- Program code stored separately, in virtual ROM



Storage

- Fees for storage as well. Total fee is proportional to the smallest multiple of 32 bytes used
- Increased storage increases the size of Ethereum state database on all nodes, therefore there is incentive to keep the data stored small
- Thus, if a transaction has a step that clears an entry in the storage, fee for executing that operation is waived and a refund is given for freeing up space

Transaction Lifecycle

- Create transaction with appropriate values/data
- Sign transaction with private key
- Send transaction to node
- Transaction is verified & propagated to the network
- Transaction gets picked up by miner & put in block
- Block is broadcasted to the network
- Block is added by other nodes to their chain

Transaction execution

- Transaction executions happens in a node :
  - When a is node is trying to create a candidate block, by picking transaction from pool and executing it
  - A node receives a block and verifies the transactions by executing it, before adding the block to the blockchain

Execution: System State

EVM checks that the following information is available and valid:

- System state (the global state)
- Remaining gas for computation
- Account address of executed code owner
- Account address of sender
- Address of account that caused the code to execute (could be different than original sender)
- Gas price of the transaction that originated execution
- Input data for execution
- Value (in Wei) passed to this account as part of current execution
- Machine code to be executed
- Block header of current block
- Depth of present message call or contract creation stack

Execution 2: Transaction Execution

- EVM then executes transaction recursively
- Computes system state and machine state (state of current execution), for each loop
- Each cycle, gas reduced from remaining gas and the program counter increments
- 3 options at end of loop:
- Exception state
- Process continues to next state
- Process finishes and reaches end state

Finalizing Blocks

Requirements:

- Validate Ommers/Uncles
- Validate transactions
- Apply Rewards (only if mining)
- Verify State and Nonce

New = block is mined

Existing = block is validated

THANK YOU

QUESTIONS ?