DSRV©

When you send ERC20 tokens

in-house seminar

Jeongho Jeon

2022-05-03

DSRV

Metamask wallet & Uniswap

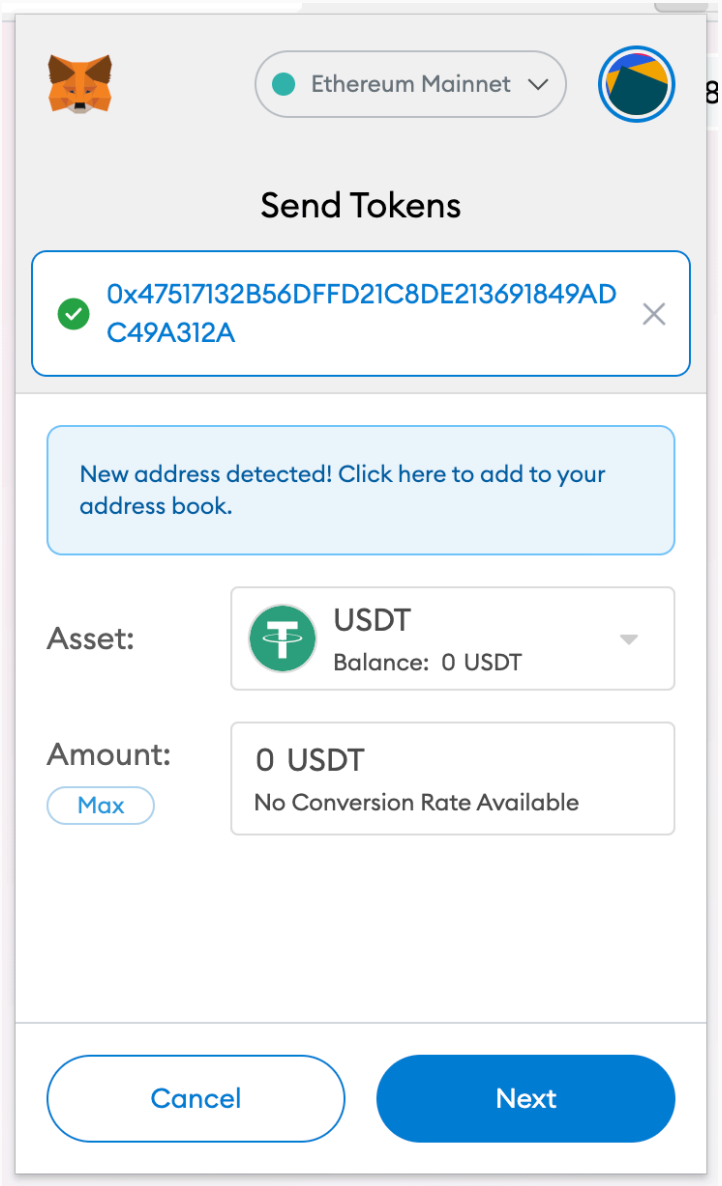


Figure 1: *

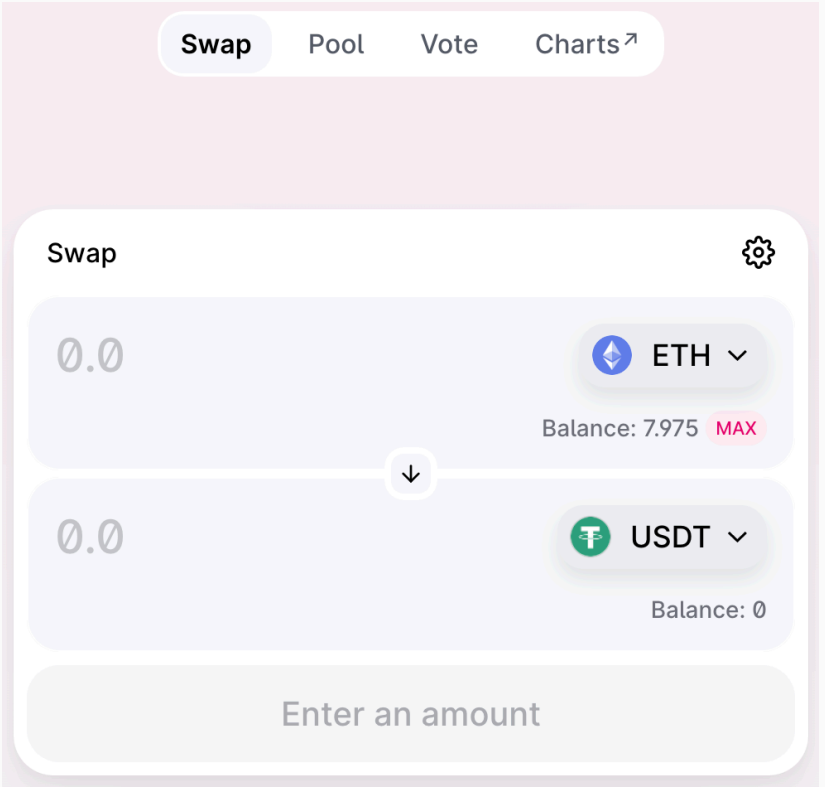


Figure 2: *

sign tx that DApp has built

Transaction (tx)

type 0x02	0x02 = EIP-1559 tx
Array	
chainId	0x01 = mainnet
nonce	the number of txs that this account has sent. No gap between serial numbers
maxPriority	maximum gas price tip that tx is willing to pay (for miners, unit wei)
maxFee	maximum gas price (tip included, unit wei)
gasLimit	maximum gas amount that tx can use
to	token contract address. not token receipient. empty when creating contracts
value	transferred ether (unit wei = 10^{-18} ether). zero (empty $\varnothing$) for ERC20 transfers
data	smart contract function selector. see next slide
accessList	for future optimizations
parity	for public key recovery from ECDSA signature, $r^{-1}(sR - zG)$
r	ECDSA signature, kG
s	ECDSA signature, $k^{-1}(z + rd_A)$

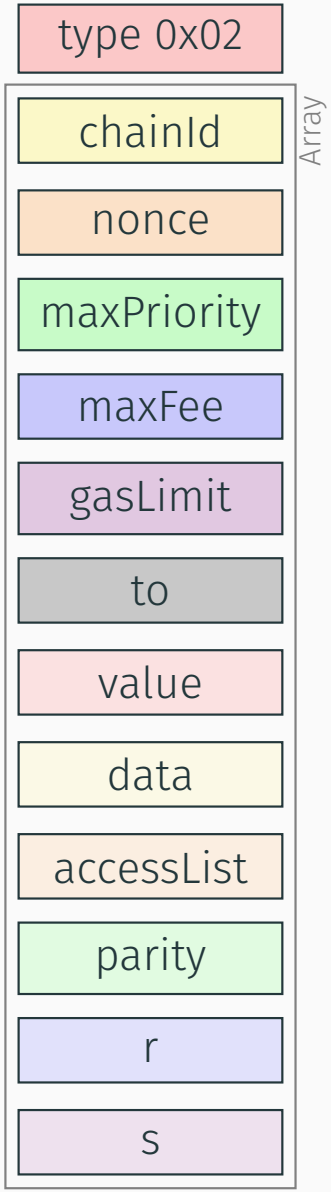
Transaction `data` field

- `data` field may contain arbitrary data such as constitution and love letters. When you call smart contracts, `data` decides a function with parameters and follows an ABI convention: 4 bytes selector + padded 32 bytes parameters.
- If you send 326.183863 USDT to 0x0437da...88,
- `0xa9059cbb` = first 4 bytes of `keccak("transfer(address,uint256)")`
- `0000000000000000000000000437dab2d4801d1e9100adebb39e95e8f9973288` recipient address
- `0000000000000000000000000000000000000000000000000000000013712bb7` =326183863
- USDT's `decimals` is 6. Internal representation of 326.183863 USDT is 326183863. Almost all ERC20s' `decimals` is 18 like ethers. 1 UNI becomes 1000000000000000000.

6 digits

18 zeros

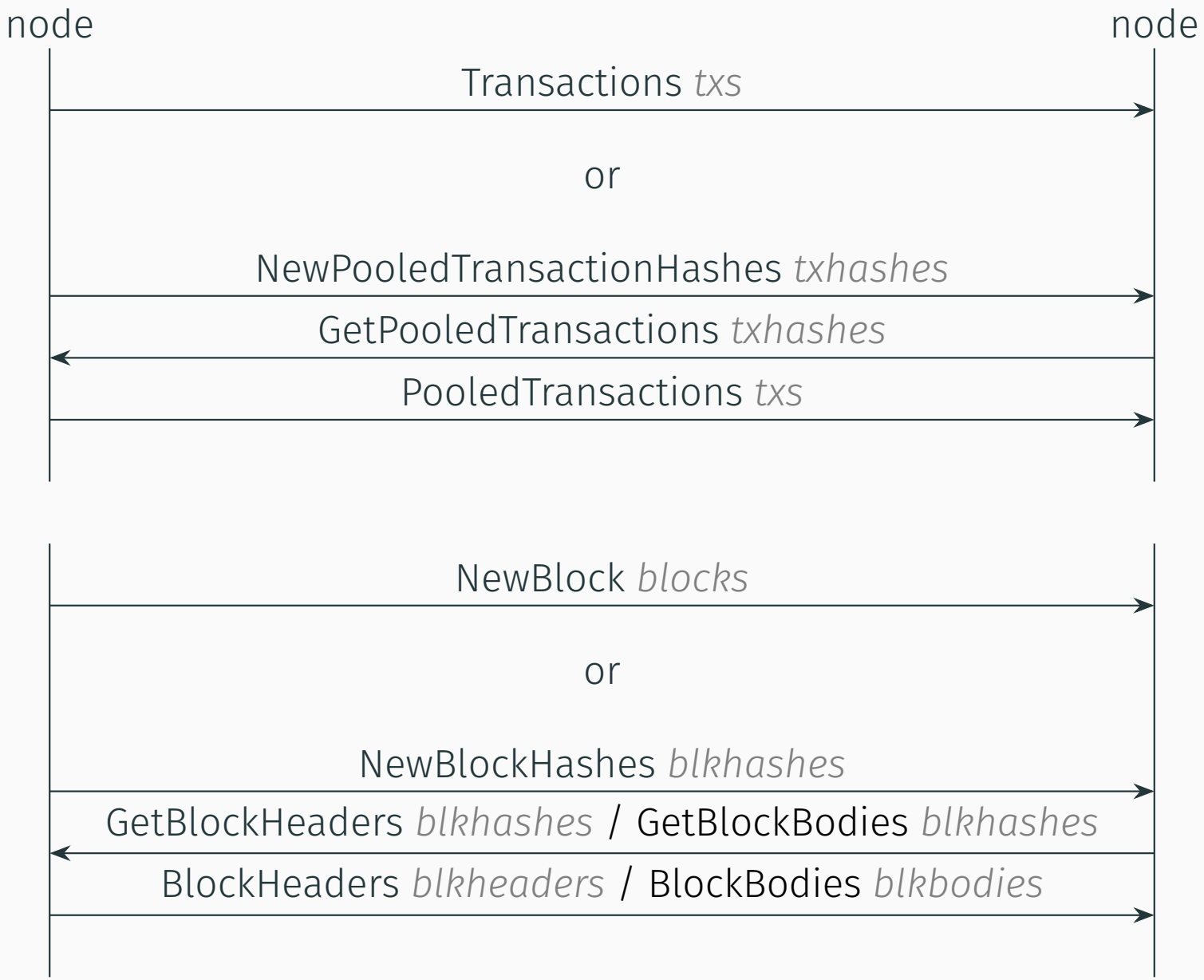
Transaction serialization by RLP (Recursive Length Prefix)



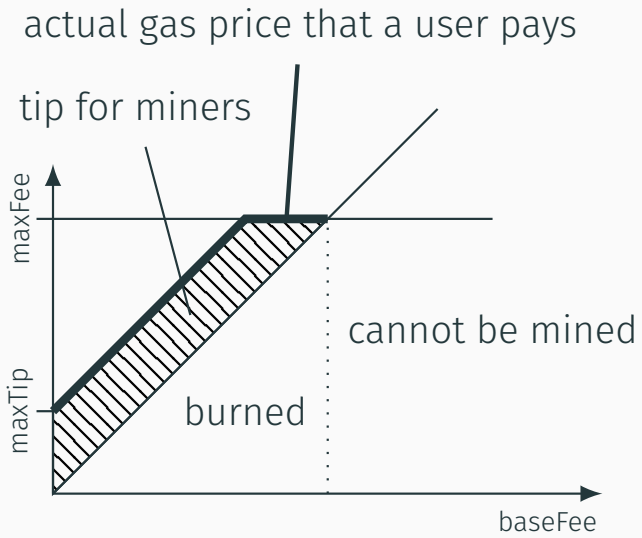

```
02f8b1010d8459682f00851053a1eac88301724b
94dac17f958d2ee523a2206206994597c13d831e
c780b844a9059cbb000000000000000000000000
0437dab2d4801d1e9100adebb39e95e8f9973288
0000000000000000000000000000000000000000
00000000000000013712bb7c001a03e97ca5ecf
2bdcdbe7f77eebc0e0c19533ee9e248d2d6cb134
3f68cd060dbd91a067e444367283b0deef089ba9
a590fdf9e5fa77207133529cb8c59a55b4ebdc73
```

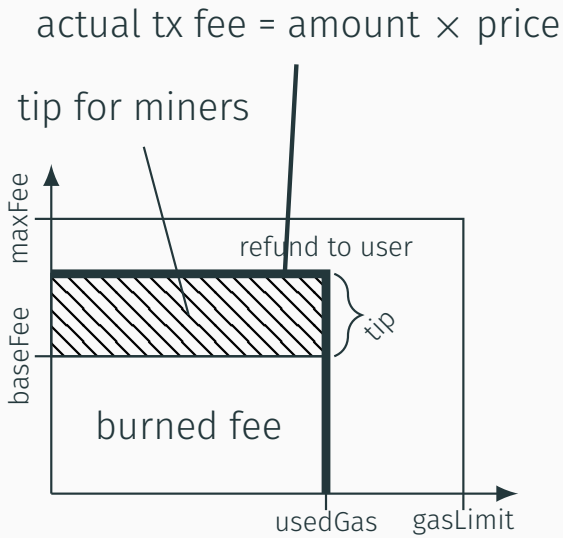
`00-7f` literal value; `01` means 1
`80-b7` (□-80) bytes string; `a0` means 20 bytes string
`b8-bf` string whose length is written in next (□-b7) bytes;
  `b8` means string and next 1 byte (`44`) is its length
`c0-f7` (□-c0) bytes array; `c0` means 0 bytes array
`f8-ff` array whose bytes length is written in next (□-f7) bytes;
  `f8` means array and next 1 byte (`b1`) is its bytes length

p2p protocol, devp2p



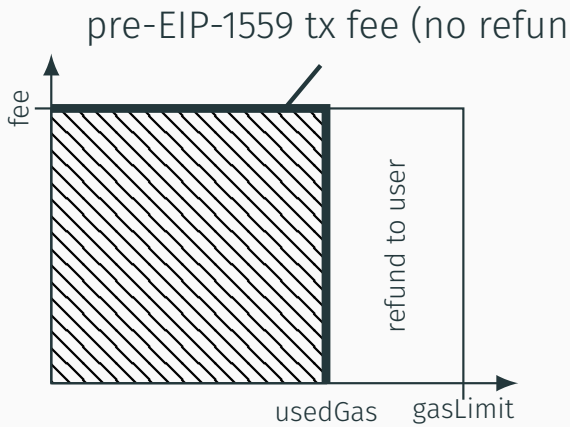
EIP-1559 gas fee





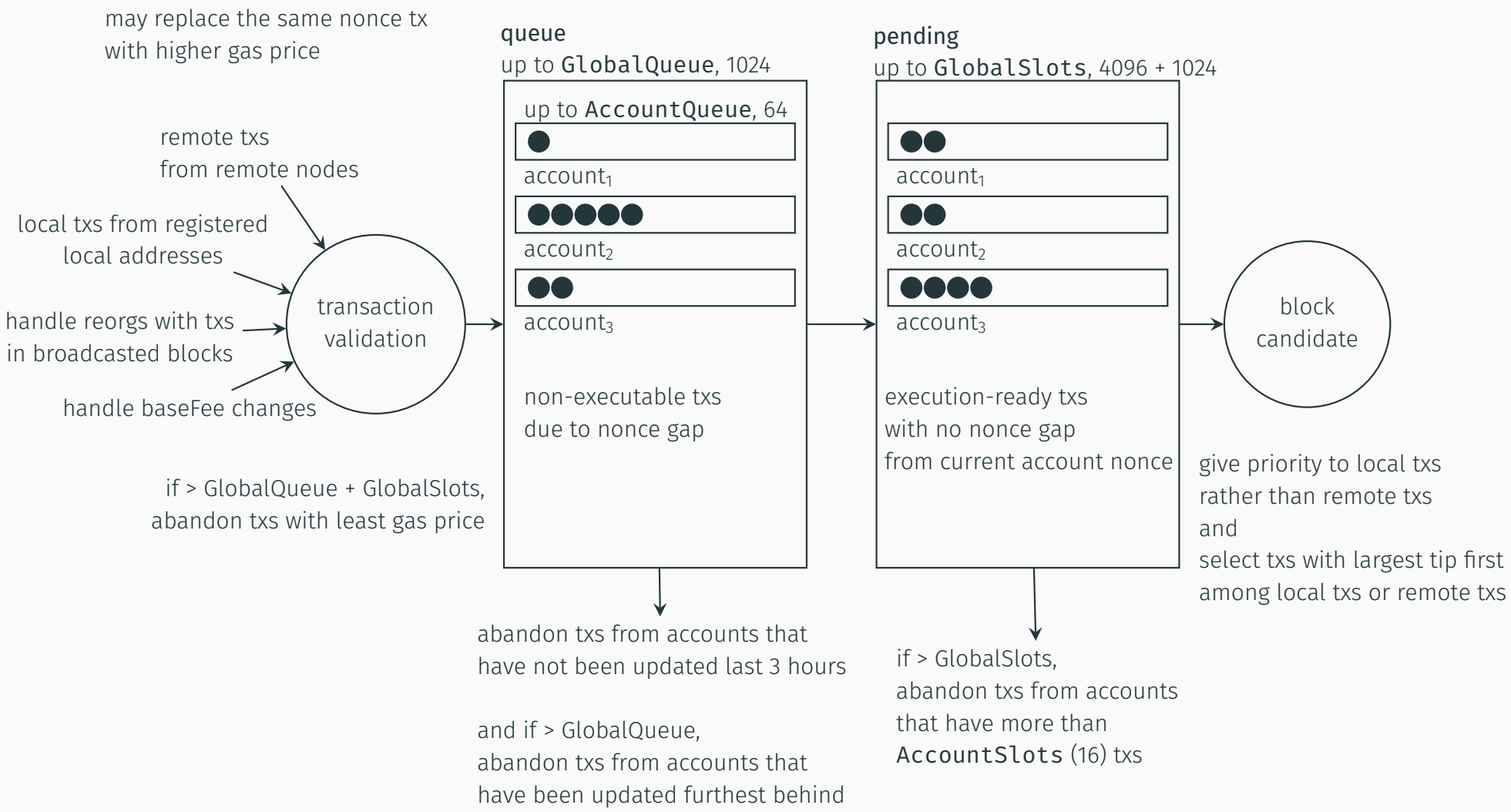
baseFee is determined from used gas ratio each block. If maximum block gas limit is used, **baseFee** increases by +12.5%.
If half of gas limit is used, it is not changed.
If no gas is used, it decreases by 12.5%.

The maximum block gas limit is decided by miners within $\pm\frac{1}{1024}$ (about 0.1%) difference from the previous block.

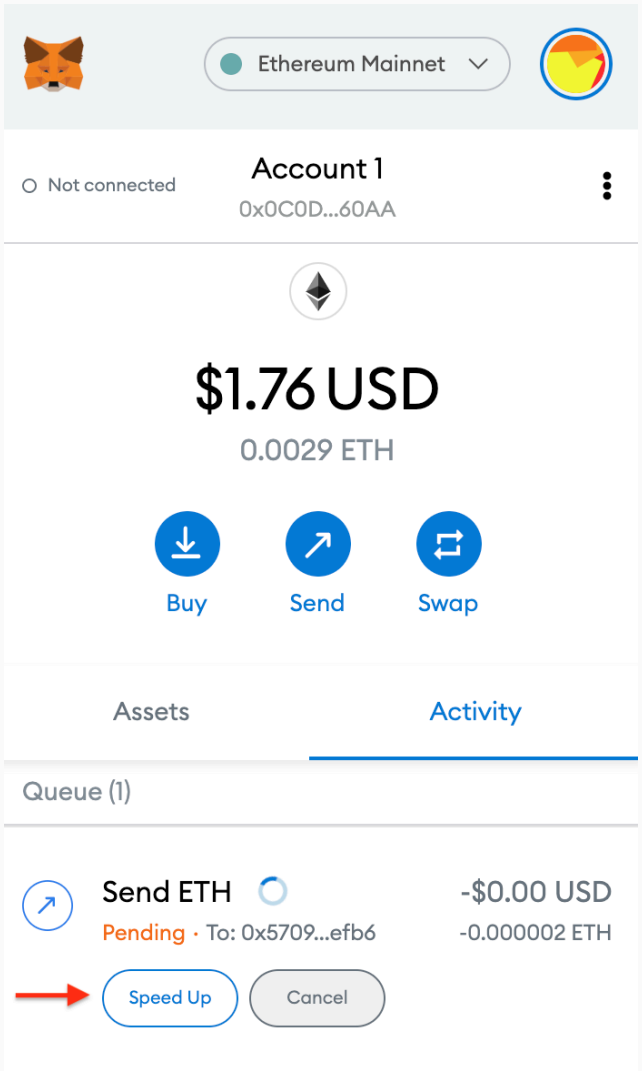


mempool, a.k.a. Dark Forest

simplified Geth mempool (not effective mining pool mempool)



Metamask Speed Up & Cancel



- You may “speed up” or “cancel” a pending transaction. Both require higher gas price.
- “Speed Up” resends the same transaction except the gas price.
- “Cancel” resends “transfer zero ethers to self” transaction with the same nonce and higher gas price.
- Metamask advises to use at least 10% higher `maxPriority` and at least 30% higher `maxFee`.
- Geth replaces a transaction with the same nonce if both `maxPriority` and `maxFee` are at least 10% (`PriceBump` parameter) higher.
- But transaction replacement is not guaranteed all times.

EVM Execution

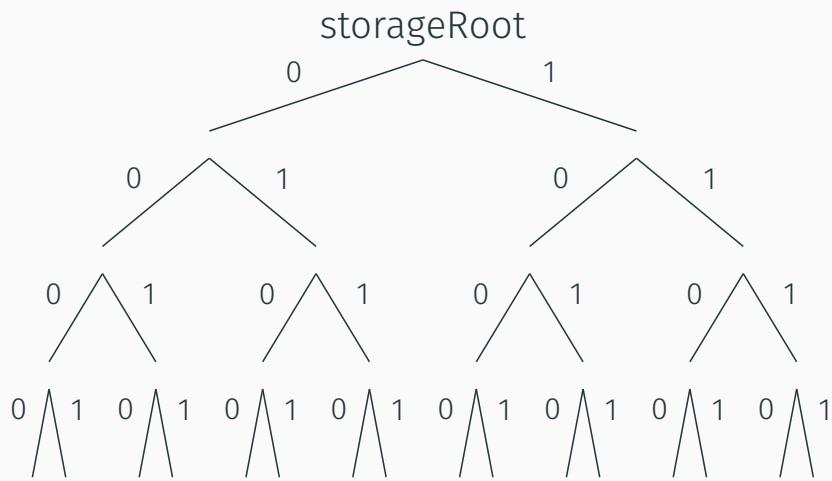
First, jump to `transfer` function by a selector,

```
…
dup1
0x95d89b41
eq
tag_22
jumpi
dup1
0xa9059cbb
eq
tag_23
jumpi
dup1
0xc0324c77
eq
…
```

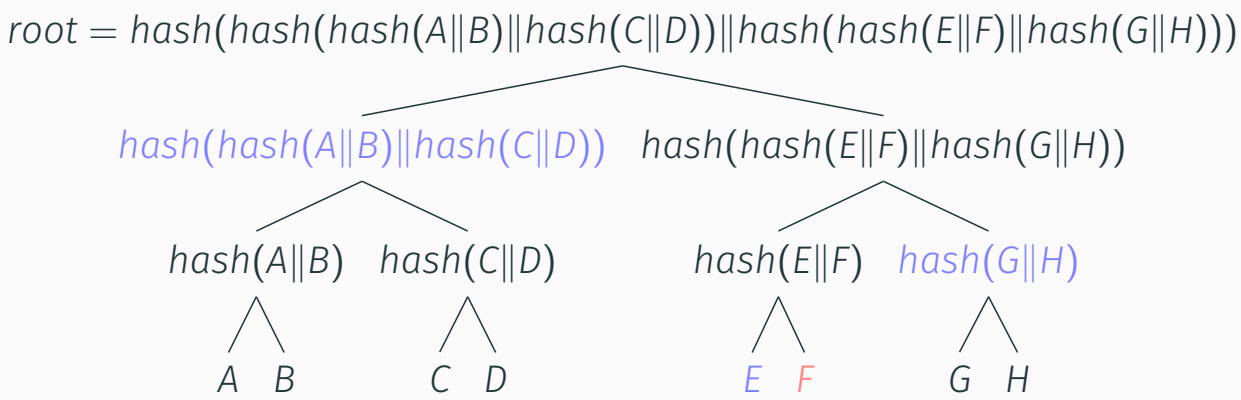
```
address public owner; // at storage 0x00
bool public paused; // at 0x00 next to owner
uint public _totalSupply; // at 0x01
mapping(address => uint) public balances; // at keccak(address||2)
uint public basisPointsRate; // at 0x03
uint public maximumFee; // at 0x04
mapping (address1 => mapping (address2 => uint)) public allowed; // at keccak(address2||keccak(address1||5))
mapping (address => bool) public isBlackListed; at keccak(address||6)
string public name; // at 0x07
string public symbol; // at 0x08
uint public decimals; // at 0x09
address public upgradedAddress; // at 0x0a
bool public deprecated; // at 0x0a next to upgradedAddress
…
balances[msg.sender] = balances[msg.sender].sub(_value);
balances[_to] = balances[_to].add(sendAmount);
…
```

```
…
sub(exp(0x2, 0xa0), 0x1)
caller /* = msg.sender */
and
0x0
swap1
dup2
mstore
0x2
0x20
mstore
0x40
swap1
keccak256
sload /* storage load */
…
```

storage in tree style
(not consecutive space)



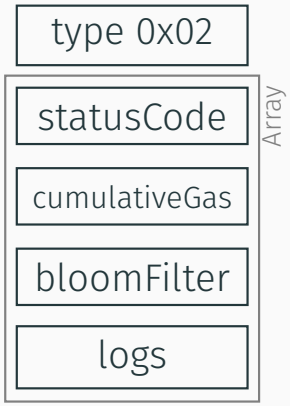
Merkle Tree



Integrity
root is changed if any leaf is changed.
You can compare roots only.

Inclusion Proof (Merkle Proof)
If Alice knows a root already, you can convince her of the existence of any leaf, F.
You send her a Merkle proof: F, E (left), $hash(G||H)$ (right), $hash(hash(A||B)||hash(C||D))$ (left).
Alice may compute a root with a Merkle proof only, and compare two roots.
You cannot fool her with nonexistent leaves.

Receipt (execution output)



0x02 = EIP-1559 tx

tx failure 0, tx success 1

used gas from the start of block up to the end of this tx

Bloom filter for quick searches on **contract address** and **log topics**

array of logs, log = (contract address, array of log topics (max 4), log data)

Consensus requires that all nodes make the same receipts after transaction execution. Receipts are not block data. Only Merkle tree root of receipts (`receiptRoot`) is stored in block header.

```
event Transfer(address indexed from, address indexed to, uint value);
```

writes a log:

`0xdac17f958d2ee523a2206206994597c13d831ec7` contract address

`0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef` first topic, *keccak*("*Transfer*(*address*, *address*, *uint*)")

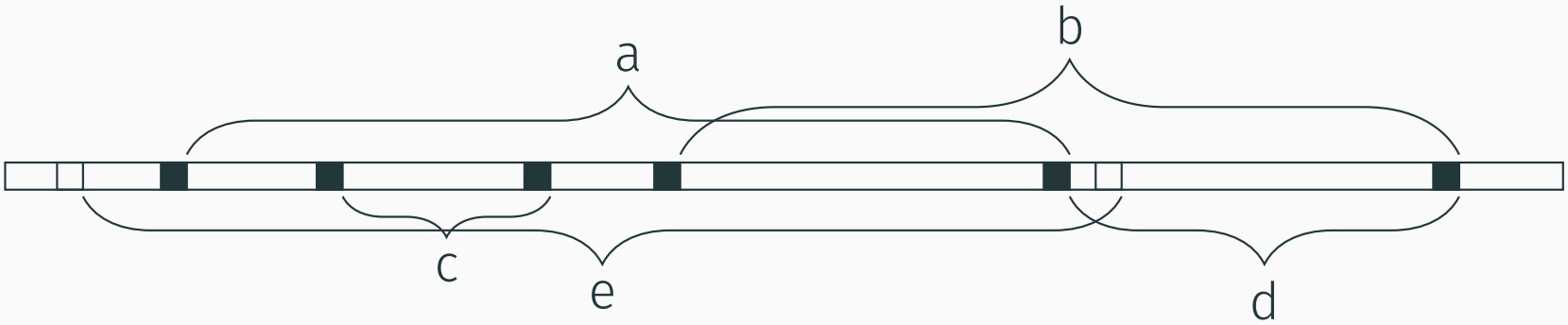
`0x000000000000000000000000014d9df840ab5ea7f7120c35931138462bec70a8a` second topic (first `indexed`) `from`

`0x0000000000000000000000000437dab2d4801d1e9100adebb39e95e8f9973288` third topic (second `indexed`) `to`

`0x00000000000000000000000000000000000000000000000000000013712bb7` cheaper log data (non-`indexed` fields) `value`

You may use web3.js to monitor log events: `web3.eth.getPastLogs()`, `TetherContract.getPastEvents()`, `TetherContract.events.Transfer()`, `TetherContract.events.allEvents()`, and `web3.eth.subscribe('logs')`

Bloom filter

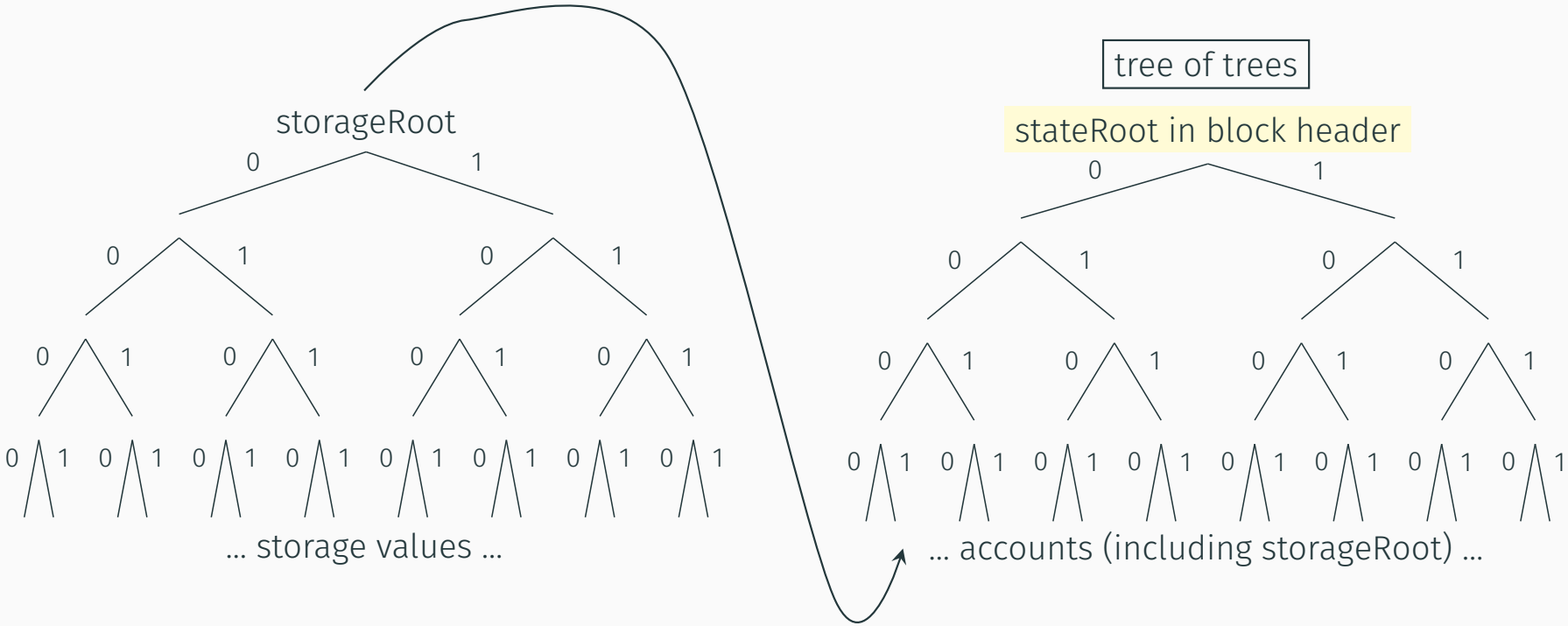


Set bits from many hash functions of values. Then we can test bits from hash functions of new values. Bloom filter may tell that not-included-values (“d”) are included by small chance (false positive). But Bloom filter **never** tell that included-values are not included (no false negative). When we want to know whether any data is included without large data transfer, we use it.

Ethereum uses three bit ranges of keccak instead of three hash functions. It makes 2048 (2^{11}) bits Bloom filter with 0-10th, 16-26th, and 32-42nd bits (11 bits from 0-1st, 2-3rd, and 4-5th bytes, respectively) from LSB of *keccak*() results.

Account

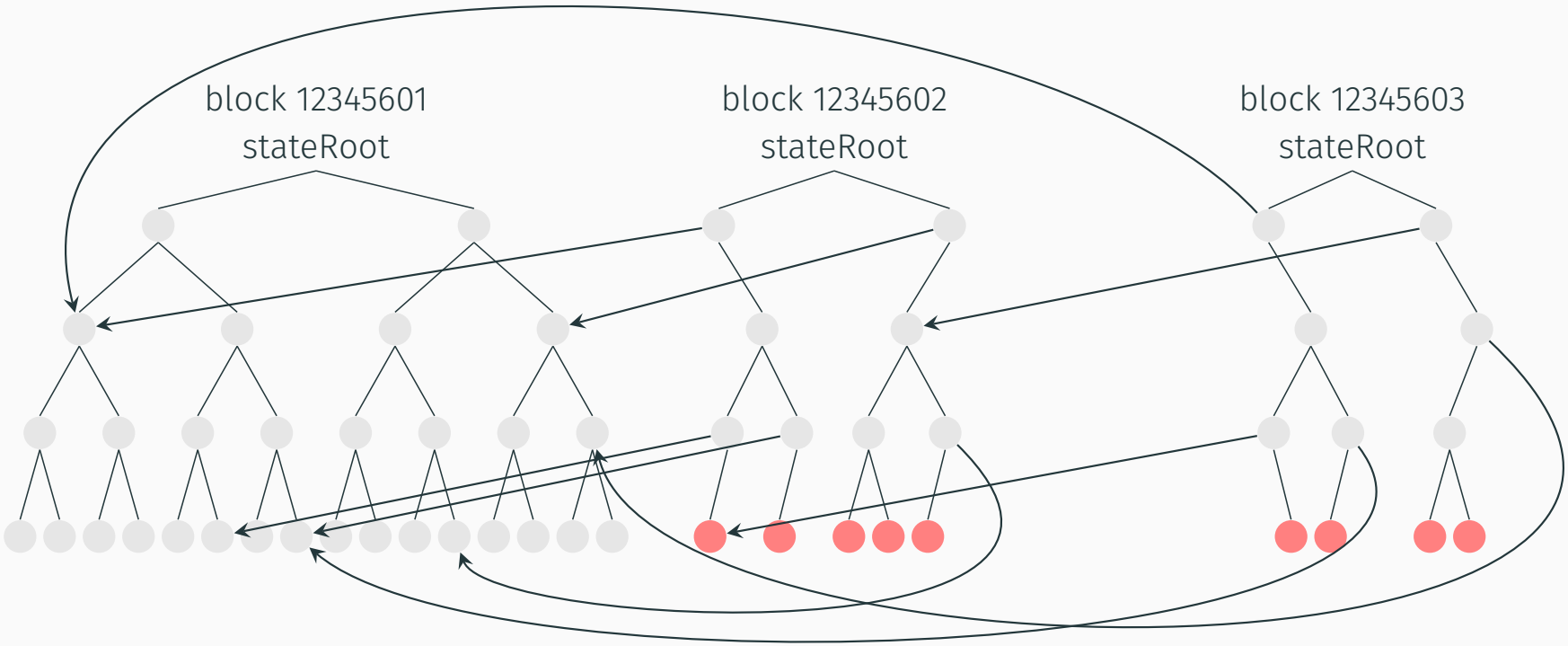
balance	ethers that this account holds (unit wei)
storageRoot	root of storage tree
nonce	the number of txs that this account has sent
codeHash	hash of smart contract code. empty in the case of user account (EOA)



Block Header

parentHash	hash of parent block header
unclesHash	hash of uncle block headers
beneficiary	miner address (coinbase)
stateRoot	Merkle tree root of all accounts
txRoot	Merkle tree root of all transactions. (key: sequence, value: transaction)
receiptRoot	Merkle tree root of all receipts (execution outputs). (key: sequence, value: receipt)
logsBloom	Bloom filter for quick searches on logs in all receipts
difficulty	PoW (Ethash) difficulty
number	block number
gasLimit	maximum gas amount that all transactions in this block can use
gasUsed	gas amount that all transactions in this block used actually
timestamp	when this block was mined
extraData	miner identity in many cases
mixHash	calculated PoW target from nonce
nonce	PoW value that miner found. cf. account nonce
baseFee	EIP-1559 base fee that is burnt

Archive Node

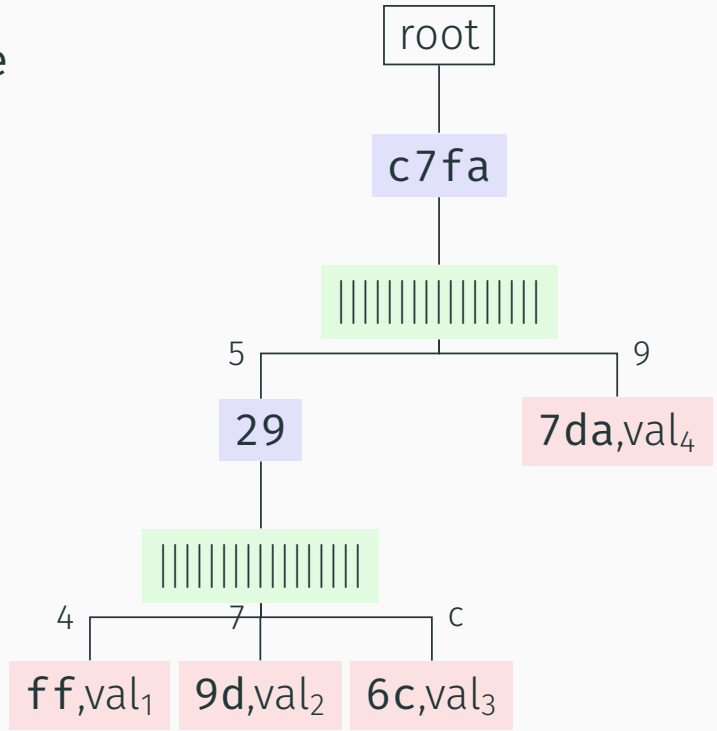


Red circles are changed. State (and storage) trees share unchanged parts.
Archive nodes store whole states from the genesis block. It takes about 10 Tb.
Normal (Geth) nodes need the latest state only, but store recent 128 blocks
(about 25 minutes, over 600 Gb) for handling reorg.

In fact, state is not in a binary tree. It is in a Patricia tree.

2^{256} keys are more than the number of atoms in universe. Ethereum state is sparse (almost empty).
We can compress common parts of keys into single tree nodes. We use series of nibbles, 4 bits key fragments.
The cons is complex code that splits and combines nodes when nodes are inserted and deleted.

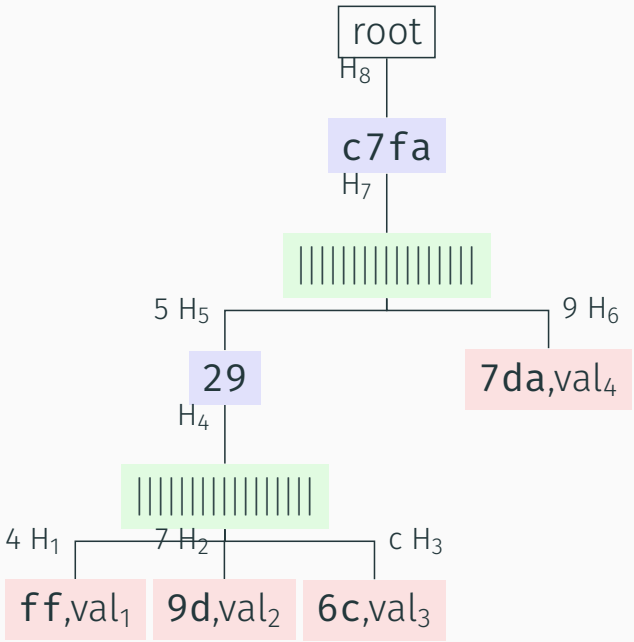
key	value
c7fa5294ff	val_1
c7fa5297 9d	val_2
c7fa529c6c	val_3
c7fa97da	val_4



Extension node represents common part of keys.
Branch node has up to 16 (nibble) children.
Also it may have its own value if there is a key that ends at this node.
Leaf node contains both remaining part of a key and a value.

And state is stored in a disk (not a memory).

RLP serializes nodes from bottom to up.
Results up to 32 bytes are used intact, but keccak results if they are longer than 32 bytes.
i.e., key: keccaked value (longer than 32 bytes) or value (otherwise) & value: RLP-serialized node
(We assume that val_1-val_4 are 32 bytes here.)
The odd and even number of nibbles in extension nodes and leaf nodes determines a prefix.



H_1 = *keccak*(20, *ff*, val_1), 20 = leaf node with even nibbles
H_2 = *keccak*(20, 9*d*, val_2)
H_3 = *keccak*(20, 6*c*, val_3)
H_4 = *keccak*(∅, ∅, ∅, ∅, H_1, ∅, ∅, H_2, ∅, ∅, ∅, ∅, H_3, ∅, ∅, ∅, ∅)
H_5 = *keccak*(00, 29, H_4), 00 = extension node with even nibbles
H_6 = *keccak*(37, *da*, H_4), 3□ = leaf node with odd nibbles
H_7 = *keccak*(∅, ∅, ∅, ∅, ∅, H_5, ∅, ∅, ∅, H_6, ∅, ∅, ∅, ∅, ∅, ∅, ∅)
H_8 = *keccak*(00, *cf*, *fa*, H_7), (1□ = extension node with odd nibbles)

Nodes and other objects are stored in physical LevelDB key-value store.
Geth uses one byte key-prefix to avoid collisions. For example,
  “h” + blockNumber + “n” (for block header hash) and “c” + codeHash.
  But no prefix is required to find tree nodes (tries).
Geth may store data older than 90000 blocks (about 2 weeks) that is considered immutable
  in slow but less expensive disk. This “freezer” (or “ancient”) database uses append-only flat files.

Closing

Ethereum will use SSZ (Simple Serialize, see Jiyun's cpp_ssz) instead of RLP and more efficient accumulator (such as Verkle) instead of Patricia tree.

Special thanks to Korea Ethereum Research Group colleagues and our captain, Jay Park!