# Ethereum

## *Data Layer*

**Prof. James Won-Ki Hong**

**Distributed Processing & Network Management Lab.**
**Dept. of Computer Science and Engineering**
**POSTECH**

**http://dpnm.postech.ac.kr**
**jwkhong@postech.ac.kr**

# Table of Contents

# Data Layer

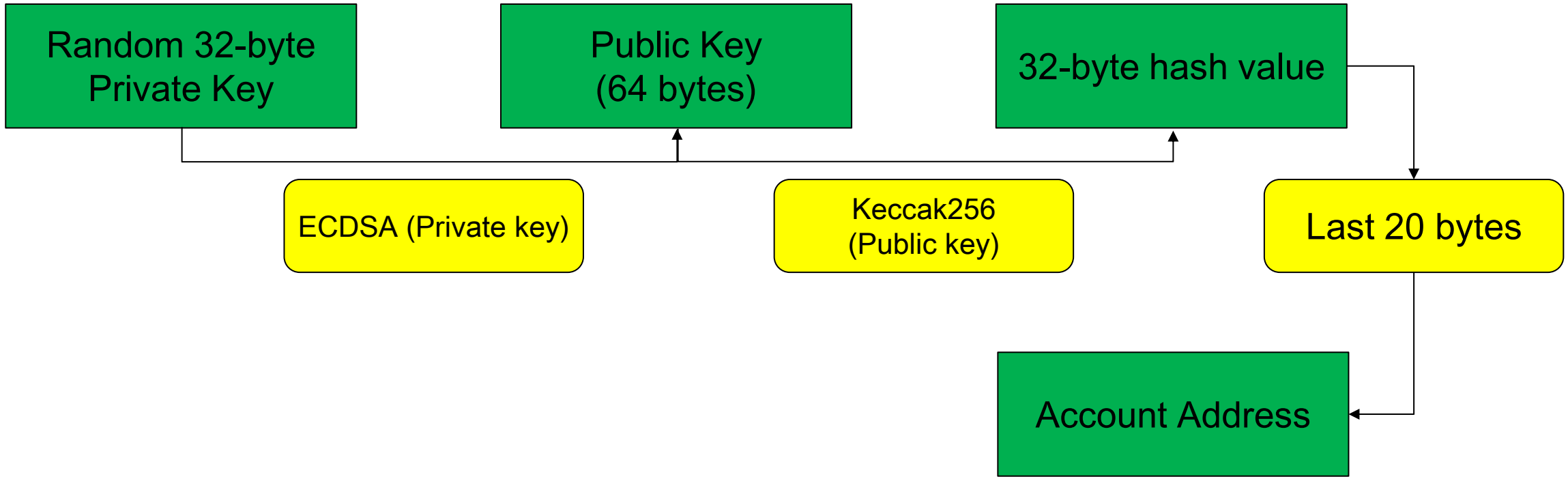
## Account

- Each account has an address of 20 bytes and account information
  - e.g., (0x)517d601e0b20483732d04c22246dfec575e1e6ad
- Ethereum has two types of account
  - **Externally Owned Account** (EOA), **Contract Account** (CA)
- Account information consists of 4 fields

| Field | Description |
|---|---|
| **Nonce** | The number of transactions sent from this account |
| **Balance** | Ether balance(Wei) of this account |
| **Root** | Merkle root of the storage trie |
| **CodeHash** | Byte code on smart contract of this account |

# Data Layer

- **Account Generation**
  - Public Key = ECDSA (Private Key)
  - Account Address = The last 20 bytes of Base58Check(Keccak256(Public Key))



# Data Layer

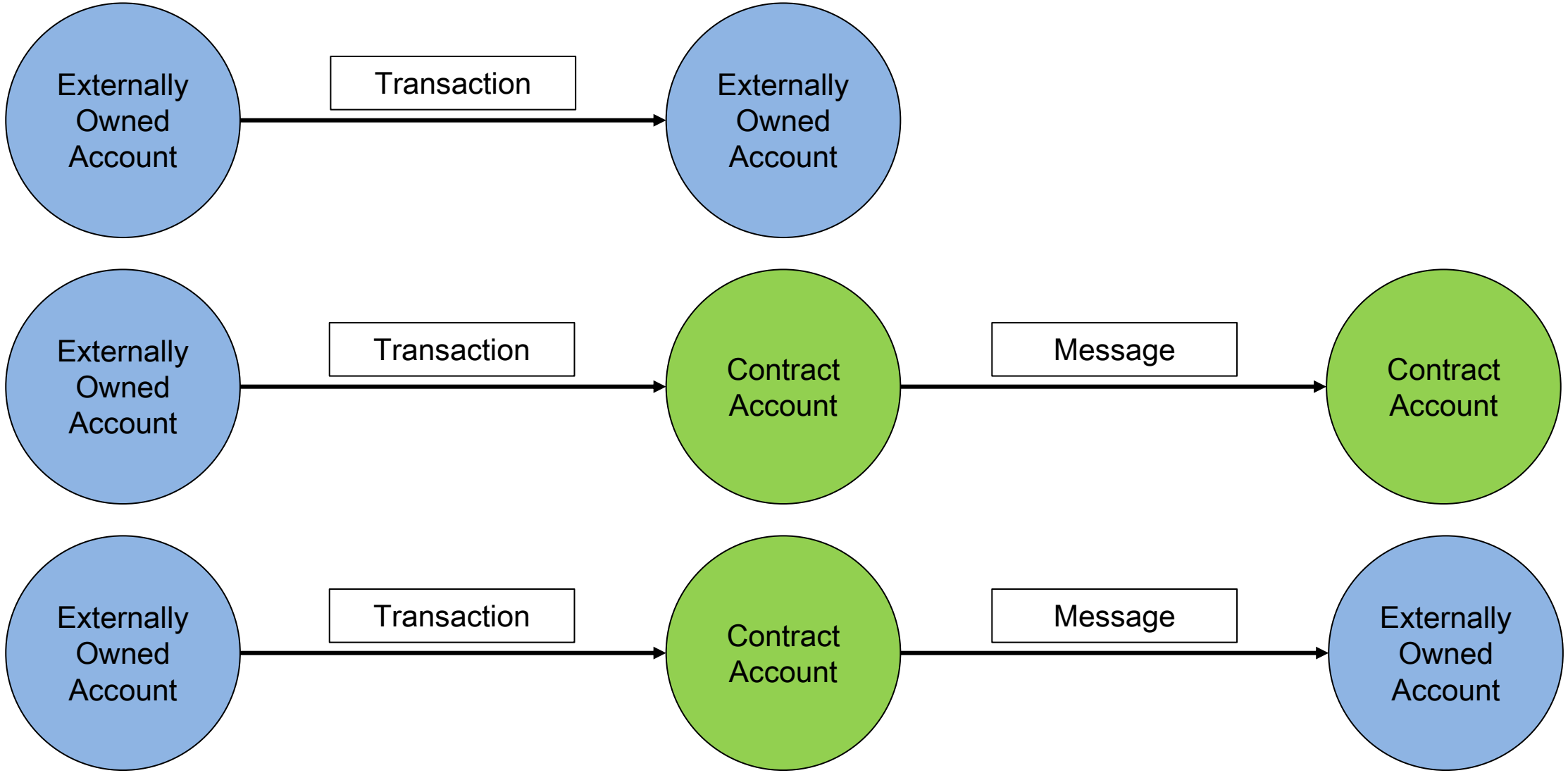
## Transaction

- Data structure sent to other account or contract
- Encrypted with a digital signature
- Used to create new contract and distribute it to nodes in Ethereum

| Field | Description |
|---|---|
| **AccountNonce** | The number of transactions sent from the sender |
| **Price** | Cost which transaction's sender will pay for each step of execution (Wei) |
| **GasLimuit** | The maximum range that can be paid for a transaction |
| **Recipient** | Address of transaction's recipient |
| **Amount** | The amount of Ether sent from the sender to the recipient (Wei) |
| **Payload** | (Optional) Parameter to be transferred to the contract / Contract code |
| **V, R, S** | Values related to ECDSA signature to identify the sender |

# Data Layer

- **Transaction Cases**



# Data Layer

- **Receipt**
  - The result of execution of transaction
  - All records for the transaction's execution
  - Save information of every transaction properly included in the block

| Field | Description |
|---|---|
| **PostState** | State information after processing the transaction |
| **Failed** | Failure after processing the transaction |
| **CumulativeGasUsed** | Gas cost cumulatively used at the block includes this transaction and receipt |
| **Bloom** | Bloom filter to quickly search for log information in Logs |
| **Logs** | Logs generated during execution of the transaction |
| **TxHash** | Hash value to identify a transaction |
| **ContractAddress** | Address of smart contract (if this transaction is created from smart contract) |
| **GasUsed** | Gas cost used in order to execute this transaction |

# Data Layer

| Field | Description |
|---|---|
| **ParentHash** | Hash value of parent block header |
| **UncleHash** | Hash value of uncle blocks of current block |
| **Coinbase** | Account address to receive Ether (Miner's address) |
| **Root** | Hash of root node of Merkle Patricia Tree containing state information of accounts |
| **TxHash** | Hash of root node of Merkle Tree that includes all transactions in current block |
| **ReceiptHash** | Hash of root node of Merkle Tree that includes all receipts in current block |
| **Bloom** | Bloom Filter used to search for log information |
| **Difficulty** | Difficulty of current block |
| **Number** | Current block number |
| **GasLimit** | Maximum gas acceptable to current block |
| **GasUsed** | The sum of gas used by transactions in current block |
| **Time** | Record on initial generation time of current block |
| **Extra** | Additional information related to current block |
| **MixDigest, Nonce** | Values used to make enough calculations during mining work creating new block |

# Data Layer

- **Genesis Block**
  - Initial (First) Block in a blockchain
  - Number of Block is “0”
  - Does not include any transactions
  - Can be generated using ethereum client with genesis.json file

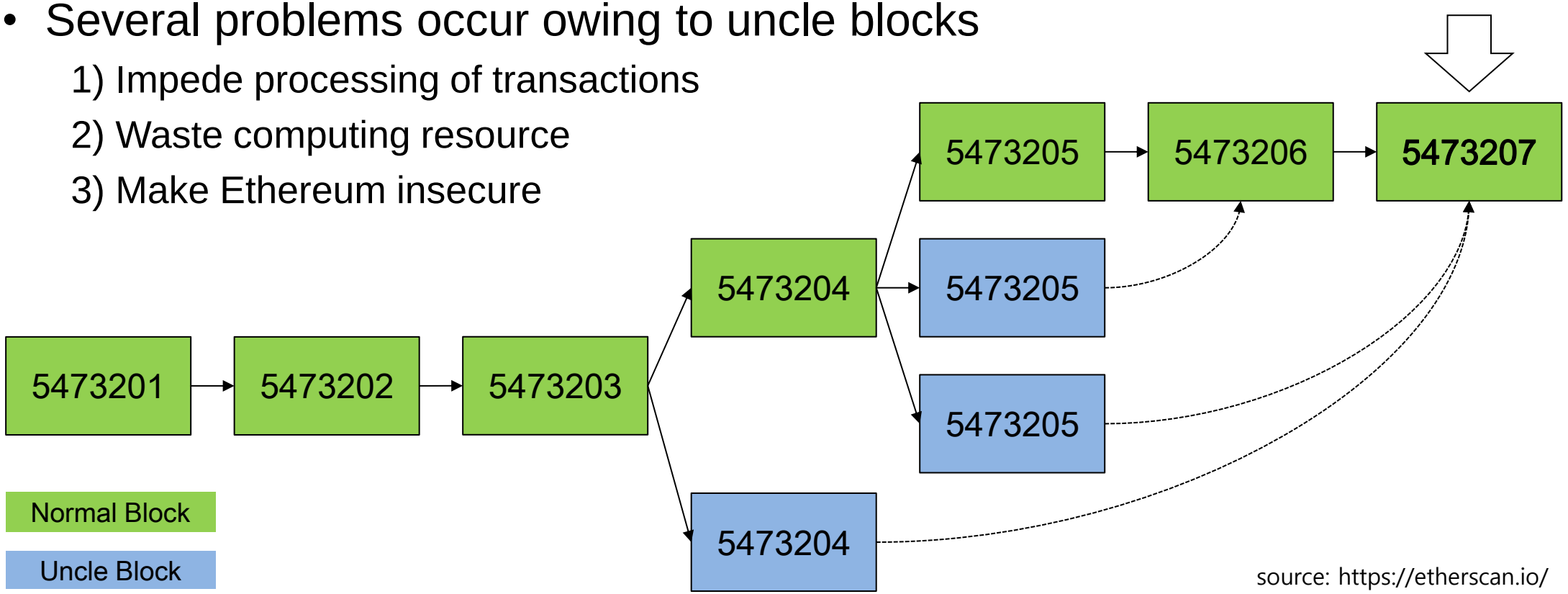
```
{
  "config": {
    "chainId": 0,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0
  },
  "alloc": {},
  "coinbase": "0x0000000000000000000000000000000000000000",
  "difficulty": "0x100",
  "extraData": "",
  "gasLimit": "0x8000000",
  "nonce": "0x0000000000000033",
  "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "timestamp": "0x0"
}
```

genesis.json

# Data Layer

## Uncle Block

- Block which cannot be included in normal (main) blockchain
  - Due to lower difficulty of the blockchain
  - Ethereum selects a blockchain which has higher difficulty
- Several problems occur owing to uncle blocks
  1) Impede processing of transactions
  2) Waste computing resource
  3) Make Ethereum insecure




source: https://etherscan.io/

# Data Layer

- **Ghost (Greedy Heaviest Observed Subtree)**

https://etherscan.io/

**[Normal Block]**

block Height: **5473207**
Transactions: 178 transactions and 6 contract internal transactions in this block
Difficulty: 3,013,212,963,063,350
Total Difficulty: 3,741,615,489,063,530,911,921
Block Reward: 3.288458365548649104 Ether (3 + 0.100958365548649104 + 0.1875)
Uncle Reward: 4.125 Ether (2 Uncles at Position 0, Position 1)

**[Uncle Block #0]**

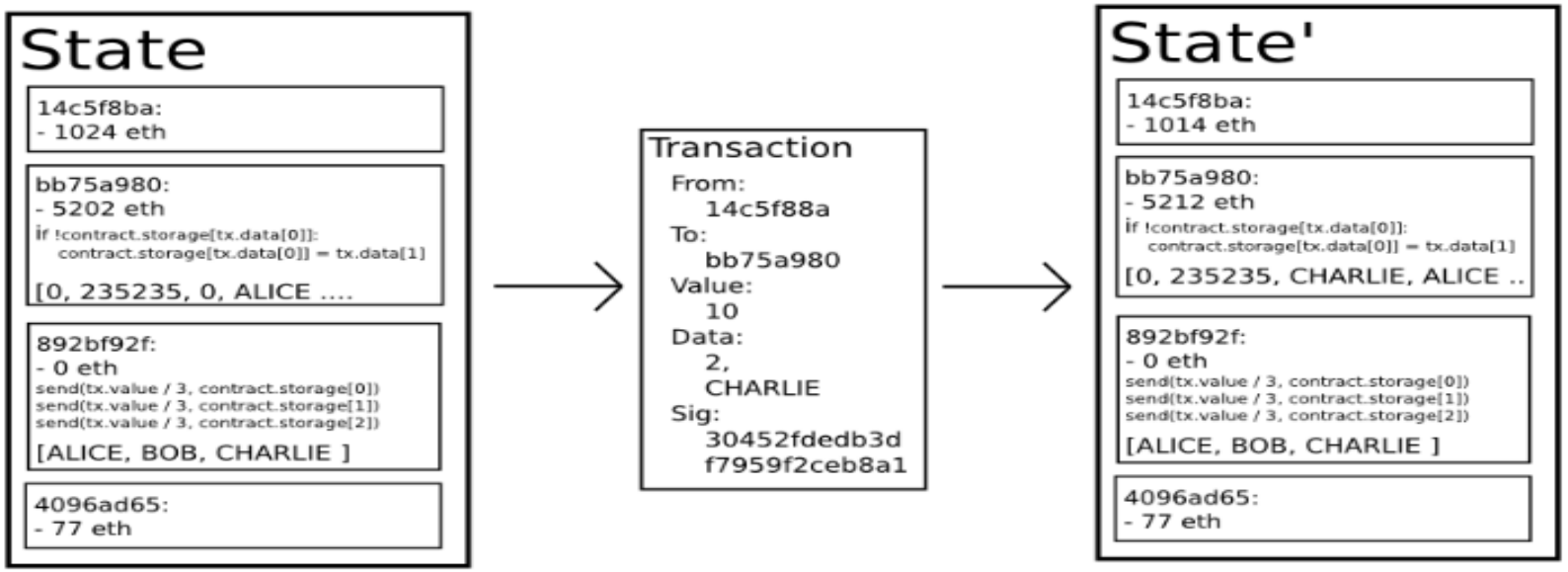
Uncle Height: **5473205**
block Height: 5473207
Difficulty: 3,017,636,198,039,301
Transaction: X
Block Reward: X
Uncle Reward: 2.25 Ether

**[Uncle Block #1]**

Uncle Height: **5473204**
block Height: 5473207
Difficulty: 3,020,585,984,845,821
Transaction: X
Block Reward: X
Uncle Reward: 1.875 Ether

# Data Layer

- **Ethereum State Transition Function**
  - The Ethereum state transition function, APPLY(S,TX) -> S'

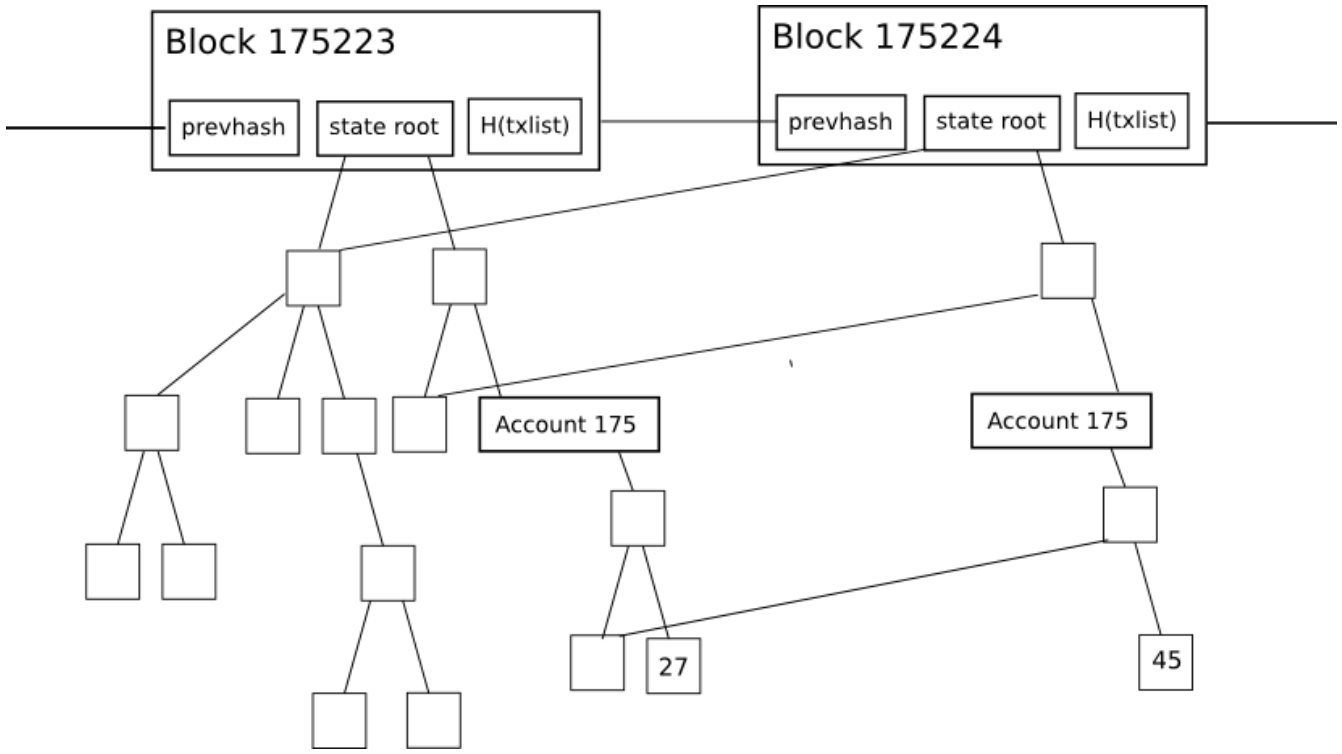




source: https://github.com/ethereum/wiki/wiki/White-Paper

# Data Layer

- **Merkle Patricia Tree (MPT) in Ethereum (1)**
  - Why MPT is needed?
    1) Improve efficiency for inserts, deletes, updates and lookups
    2) Bound the depth of the tree (Merkle Patricia Tree)




source: https://blog.ethereum.org/2015/06/26/state-tree-pruning/

# Data Layer

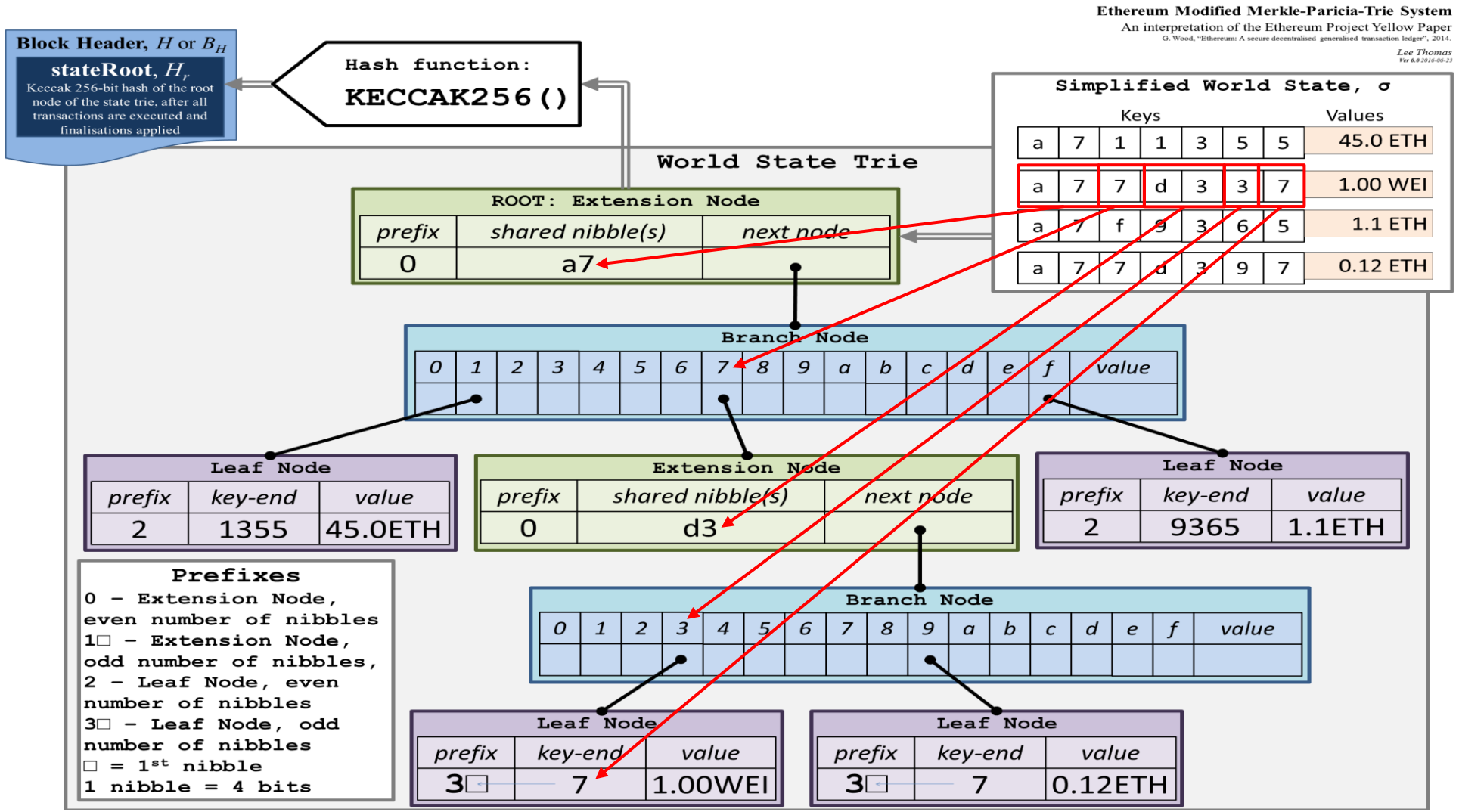
## Merkle Patricia Tree (MPT) in Ethereum (2)

- Main features
  - All elements in the MPT are encoded in **Recursive Length Prefix (RLP)**
  - Paths to each node in the MPT are stored in **LevelDB** after they are RLP-encoded and hashed in Keccak256
  - Tree has **4 types** of node:
    - Empty node, Leaf node, Extension node, Branch node
  - **Hex-Prefix** is used to distinguish between the leaf and extension node

| Prefix | Node Type | Length of path(Nibble) |
|---|---|---|
| 0 | Extension node | Even |
| 1 | Extension node | Odd |
| 2 | Leaf node | Even |
| 3 | Leaf node | Odd |

# Data Layer

- **Merkle Patricia Tree (MPT) in Ethereum (3)**




source: https://ethereum.stackexchange.com/questions/6415/eli5-how-does-a-merkle-patricia-trie-tree-work?rq=1

# Data Layer

- **Ether Units**
  - Used as a cryptocurrency in Ethereum
    - Users can transfer and receive Ether
    - Most exchanges provide Ether
  - Used to pay for EVM calculation fees
  - **1** Ether = **1***10^18 Wei

https://converter.murkin.me/

| | |
|---|---|
| 1 | Wei |
| 0.001 | Kwei |
| 0.000001 | Mwei |
| 0.000000001 | Gwei |
| 0.000000000001 | Szabo |
| 0.000000000000001 | Finney |
| 0.000000000000000001 | Ether |
| 0.000000000000000000001 | Kether |
| 0.000000000000000000000001 | Mether |
| 0.000000000000000000000000001 | Gether |
| 0.000000000000000000000000000001 | Tether |

# Data Layer

- **Gas**
  - Operational **token** of Ethereum
    - Used to pay for the usage of Ethereuem platform
  - Precaution against abuse of transactions
    - To prevent DoS attack
    - To prevent smart contract from executing infinite loop
  - Cost for executing a transaction in Ethereum:

    Max Transaction Fee = Gas Price x Gas Limit

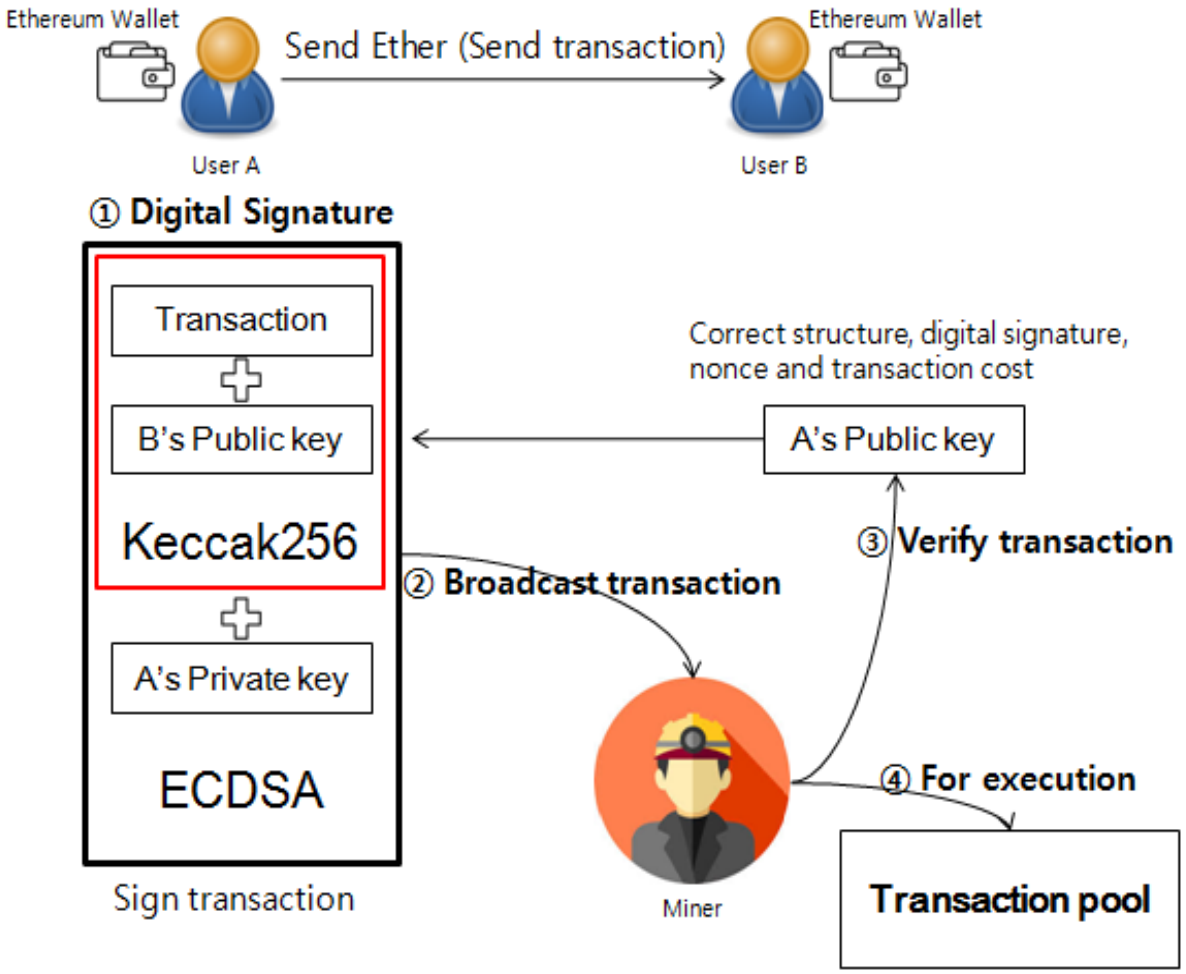

**Fuel tank** 50 Liter × **Price per liter** 2,000 Won = **Cost for full** 100,000 Won

**Gas limit** 20,000 Gas × **Gas Price per 1000 Gas** 5000 Gwei = **Max Transaction Fee** 0.0001 ETH

1Gwei = 0.000000001 ETH

# Data Layer

## Transaction Processing




(1) User A performs ECDSA digital signature encryption on the transaction with its own private key

(2) Ethereum client (geth) broadcasts signed transactions to all nodes currently connected

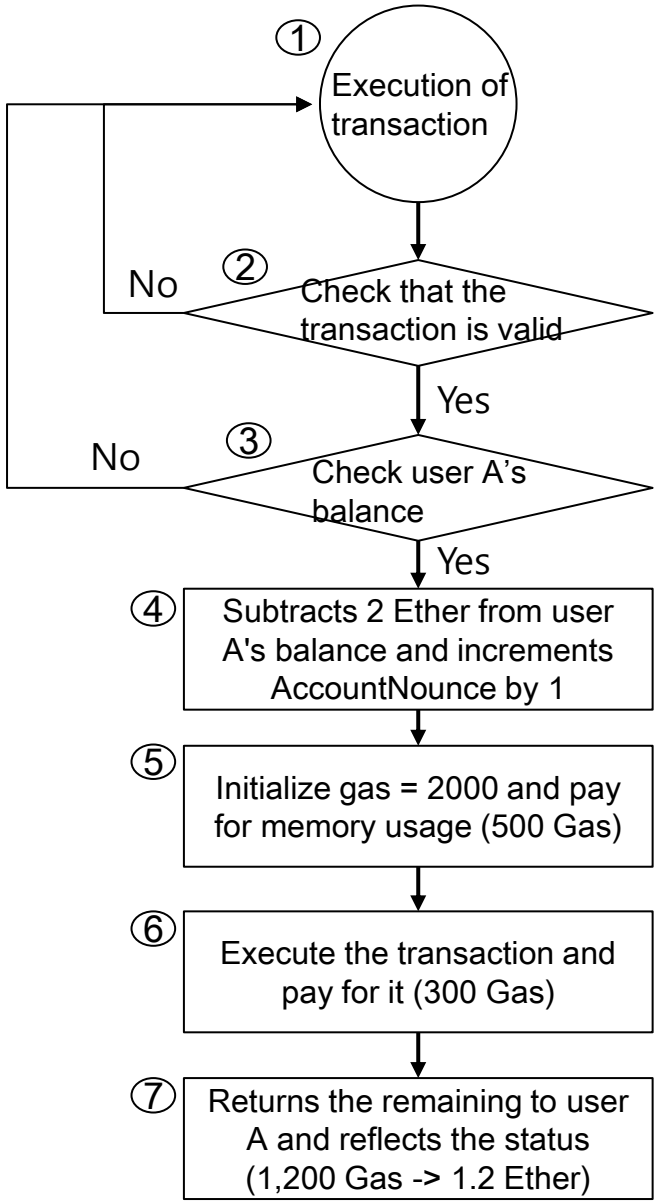
(3) Miner validates the transaction, such as digital signature verification, and puts it in the transaction pool if there is no problem

(4) Miner pulls out the transaction with the highest cost of execution in the transaction pool, executes it, adds the block to blockchain by mining it

# Data Layer

## Transaction Cost Processing

- Suppose that
  - GasLimit: 2,000 Gas
  - GasPrice: 0.0001 Ether
  - Data size used in transaction: 100 bytes
  - Processing cost per 1 byte: 5 Gas
  - User A's transaction cost: 300 Gas
  - 1 Gas = 0.001 Ether




Check that the transaction sender has at least 2000 * 0.001 = 2 ether

2,000 Gas – (100*5 Gas) = 1,500 Gas

1,500 Gas – 300 Gas = 1,200 Gas

1,200 Gas * 0.0001 = 1.2 Ether (Return)

# Summary

- **Data Layer**
  - Account
  - Transaction
  - Receipt
  - Block
  - Merkle Patricia Tree
  - Ether / Gas
  - Transaction / Cost Processing

# References


- https://www.ethereum.org/
- https://namu.wiki/w/Ethereum
- https://github.com/ethereum/wiki/wiki/White-Paper
- https://easythereentropy.wordpress.com/2014/06/04/understanding-the-ethereum-trie/
- https://blog.ethereum.org/2015/06/26/state-tree-pruning/
- Jaehyun Park, **core ethereum programming**, Jpub, 2018