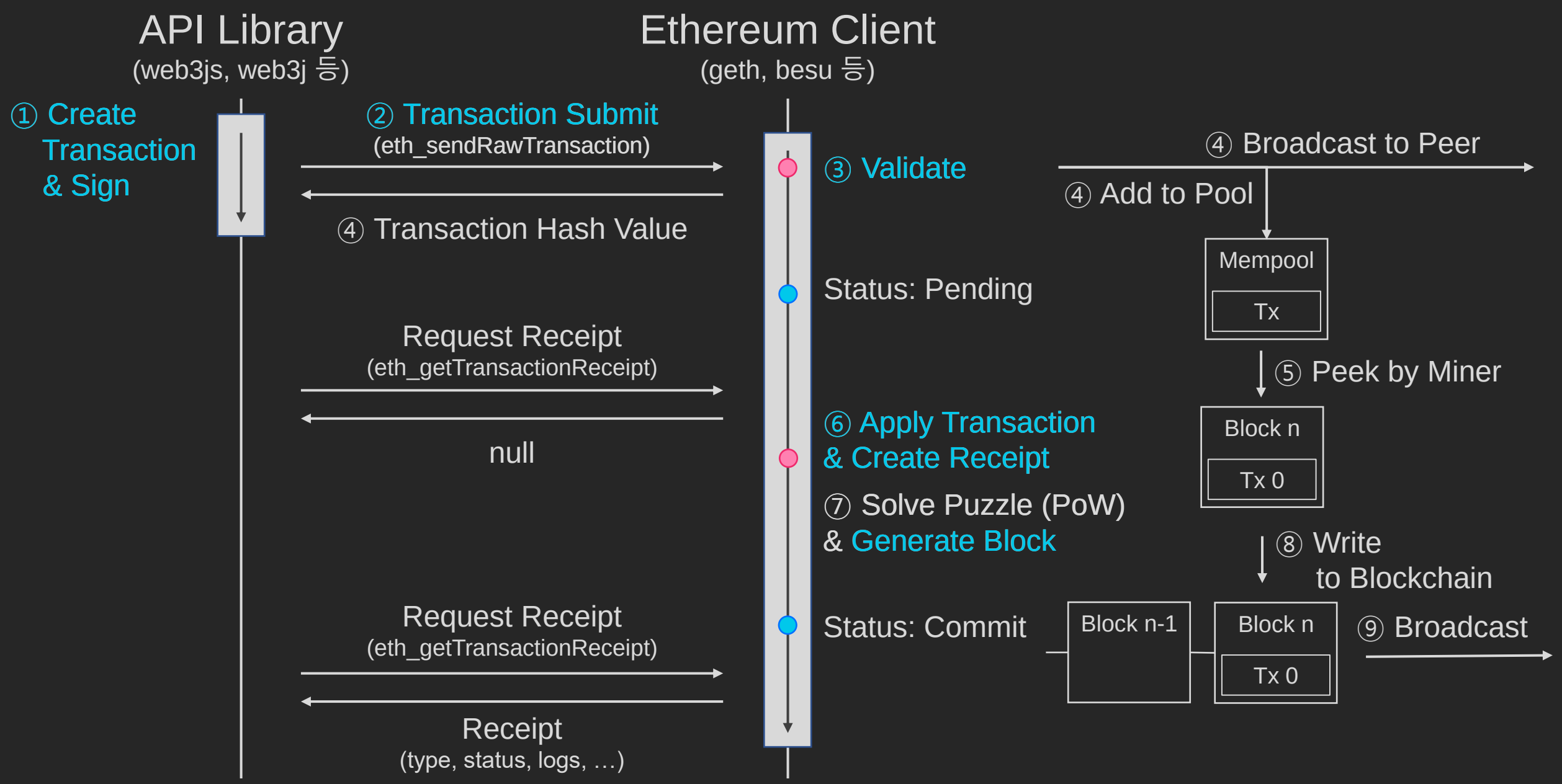
# Ethereum Deep Dive

이더리움의 트랜잭션부터 블록까지

- 목차
  - 1. 네트워크 구성 및 샘플 Solidity 컨트랙트
  - 2. ECDSA, ABI, RLP
  - 3. 트랜잭션 생성, 제출, 적용 및 영수증
  - 4. 블록과 상태 관리

- 챕터 1 : 네트워크 구성 및 샘플 Solidity 컨트랙트


API Library
(web3js, web3j 등)
Ethereum Client
(geth, besu 등)
① Create Transaction & Sign
② Transaction Submit
(eth_sendRawTransaction)
③ Validate
④ Broadcast to Peer
④ Add to Pool
④ Transaction Hash Value
Mempool
Tx
Status: Pending
Request Receipt
(eth_getTransactionReceipt)
⑤ Peek by Miner
Block n
Tx 0
null
⑥ Apply Transaction & Create Receipt
⑦ Solve Puzzle (PoW) & Generate Block
⑧ Write to Blockchain
Request Receipt
(eth_getTransactionReceipt)
Status: Commit
Block n-1
Block n
Tx 0
⑨ Broadcast
Receipt
(type, status, logs, …)

- OS : Ubuntu 20.04
- geth (go-ethereum)
  - Official Go implementation of the Ethereum protocol
  - Ehtereum co-founder Jeffrey Wilcke 개발
- 필요 프로그램 설치, 소스코드 다운로드 및 geth 빌드

```
$ sudo apt update && sudo apt upgrade
$ sudo apt install golang make gcc git tree
$ git clone -b "release/1.10" --single-branch https://github.com/ethereum/go-ethereum.git
$ git clone https://github.com/ethereum/go-ethereum.git
$ cd go-ethereum
$ sudo make geth
```

# genesis.json – Ethash (PoW)

```
{
  "config": {
    "chainId": 4693,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "muirGlacierBlock": 0,
    "berlinBlock": 0,
    "londonBlock": 0,
    "arrowGlacierBlock": 0,
    "ethash": {}
  },
  "difficulty": "131072",
  "extraData": "",
  "gasLimit": "29970705",
  "alloc": {
    "942F397B7f4391B43115395F469c63072aEd6E41": { "balance": "1000000000000000000000" }
  }
}
```

- `"chainId": 4693` — ChainId (EIP-155)
- `"homesteadBlock": 0` ~ `"arrowGlacierBlock": 0` — Hardfork 적용 블록번호
- `"ethash": {}` — 합의 알고리즘: ethash(PoW)
- `"difficulty": "131072"` — 채굴 난이도
- `"extraData": ""` — Extra Data
- `"gasLimit": "29970705"` — Block GasLimit (Block Size)
- `"alloc": { ... }` — Account 초기 Balance (1000 ETH)

- Geth Database 초기화

```
$ geth init --datadir data genesis.json
```

  - genesis.json 설정으로 data 폴더에 초기화

- Geth 구동

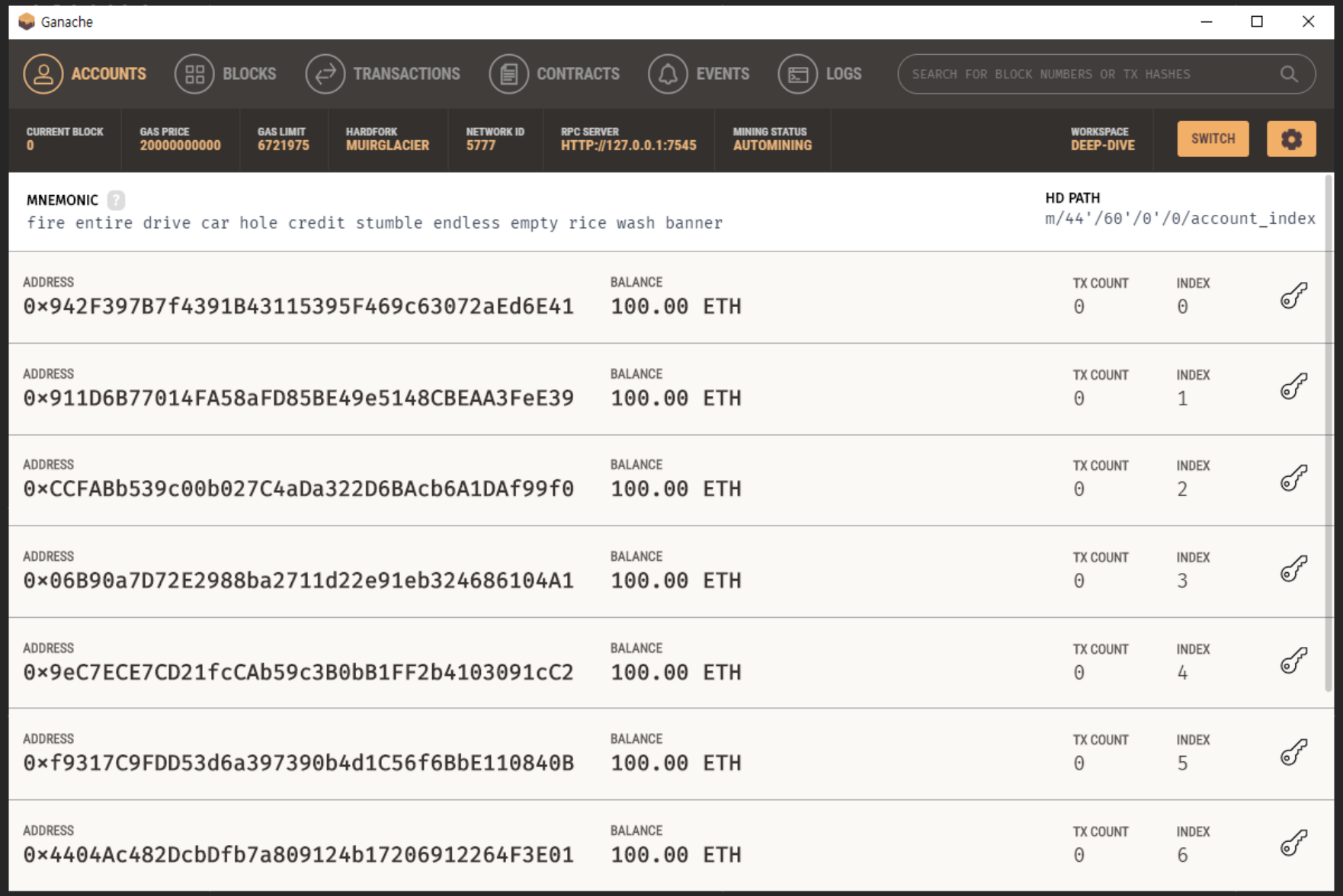
```
$ geth \
  --datadir data \
  --networkid 5666 \
  --nodiscover \
  --ipcdisable \
  --http --http.port 7545 \
  --http.api personal,eth,net,web3 \
  --http.corsdomain '*' \
  --allow-insecure-unlock \
  console
```

  - --nodiscover : peer 탐색하지 않음
  - --ipcdisable : ipc 통신 비활성화 (Windows 10 WSL2 환경)
  - --allow-insecure-unlock : account 연관 API 오픈
  - console : console 오픈

## geth console 기본 명령어

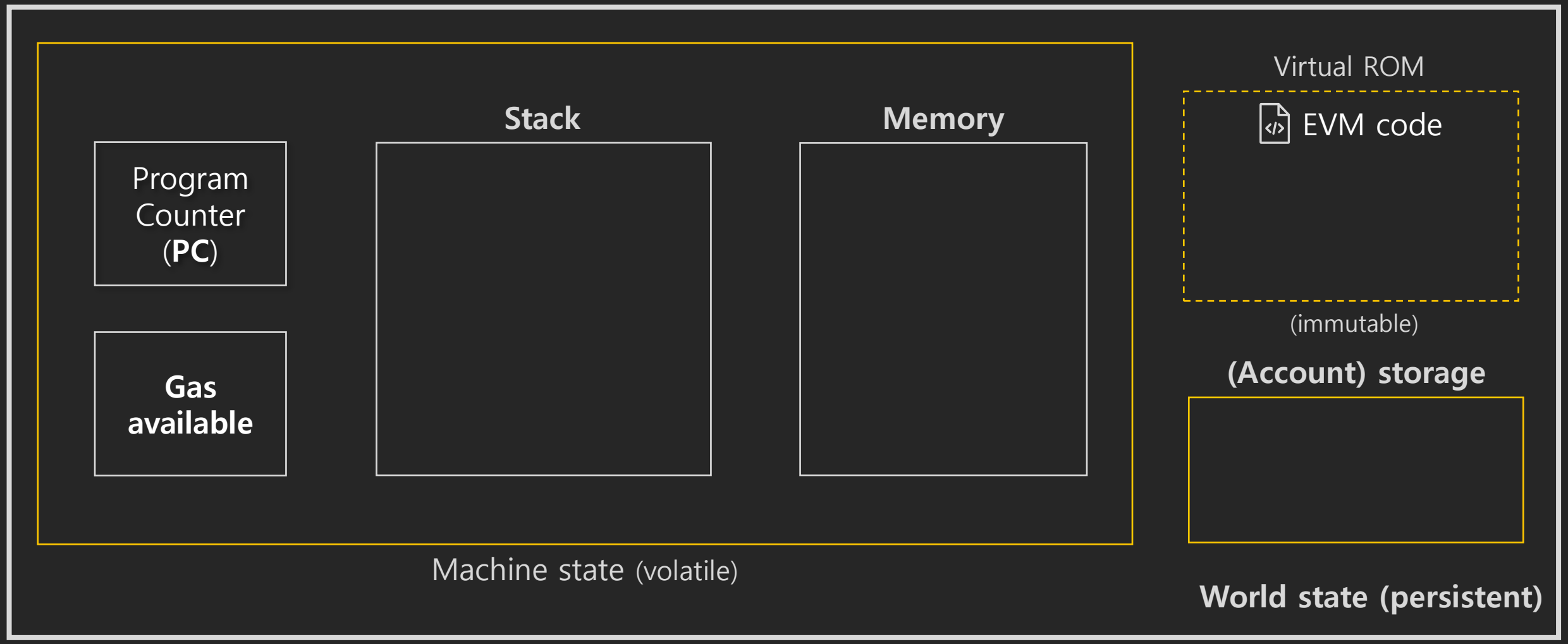
```
# 현재 블록번호 조회
> eth.blockNumber
# 채굴여부 조회
> eth.mining
# 채굴 보상 수신 계정 설정
> miner.setEtherbase("0x942F397B7f4391B43115395F469c63072aEd6E41")
# 채굴 보상 수신 계정 조회
> eth.coinbase
# 채굴 시작(스레드 수)
> miner.start(1)
# 채굴 중지
> miner.stop()

# 블록 조회(블록 번호)
> eth.getBlock(0)
# 트랜잭션 조회(트랜잭션 해시)
> eth.getTransaction("0xaf1ad2f6b02595cfcf8fdbb78e69be2f15e762c915dcf708fdfe3d1f02c80848")
# 트랜잭션 영수증 조회(트랜잭션 해시)
> eth.getTransactionReceipt("0xaf1ad2f6b02595cfcf8fdbb78e69be2f15e762c915dcf708fdfe3d1f02c80848")
```

# Ganache : 로컬 개발용 이더리움 클라이언트 Download



# EVM (Ethereum Virtual Machine)

## Smart Contract Runtime Environment



## EVM 스마트 컨트랙트 프로그래밍 언어 Docs

- Solidity(ECMAScript 문법 기반), Vyper (Python 기반)
- 결정론적(deterministic) : 동일 상태, 동일 코드에 같은 입력 값의 실행은 항상 같은 결과
  - 예) Random 없음
  - 예) timestamp  Block을 참조 ➡ 블록에 속한 트랜잭션들의 컨트랙트 실행 시 timestamp는 동일
- 컴파일 결과
  - Bytecode : EVM 에서 실행되는 컨트랙트 OpCode (배포 대상)
  - ABI (Application Binary Interface) : EVM과 상호작용을 위한 컨트랙트 인터페이스

## Solidity

- EVM 스마트 컨트랙트 프로그래밍 언어
- 2014년 08월 Gavin Wood 에 의해 제안
- ECMAScript 문법 기반
- HelloWorld.sol

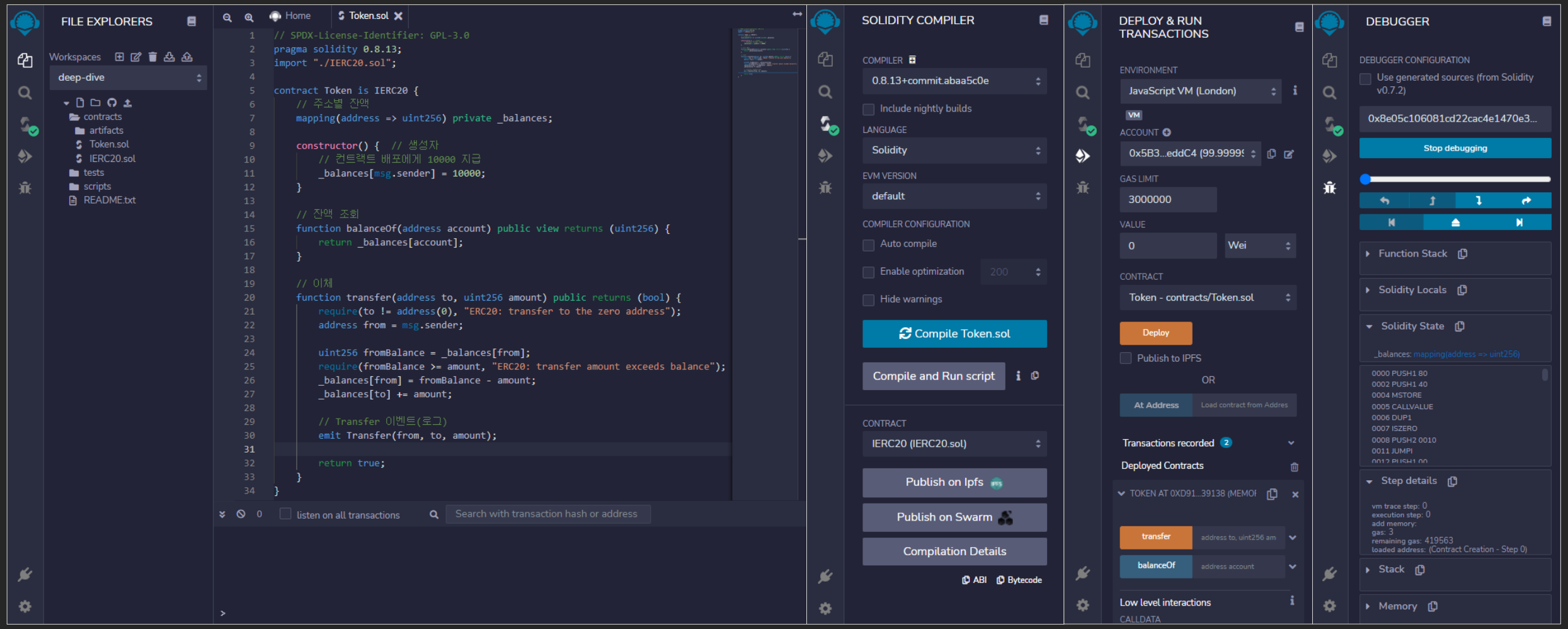
```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.13;

contract HelloWorld {
    function hello() public pure returns(string memory) {
        return "Hello, world";
    }
}
```

- Truffle : Command-line EVM IDE(Integrated Development Environment, 통합 개발 환경)

```
$ npm install -g truffle
$ mkdir deep-dive
$ cd deep-dive
# 프로젝트 초기화
$ truffle init
$ tree
.
├── contracts
│   └── Migrations.sol
├── migrations
│   └── 1_initial_migration.js
├── test
└── truffle-config.js
```

```
# 컴파일
$ truffle compile

# 테스트
$ truffle test

# 배포
$ truffle migrate
```

- Remix : EVM GUI IDE
  - Web : https://remix.ethereum.org/ , Desktop : https://github.com/ethereum/remix-desktop/releases
- 코드 작성, 컴파일, 배포 및 실행 , 디버그

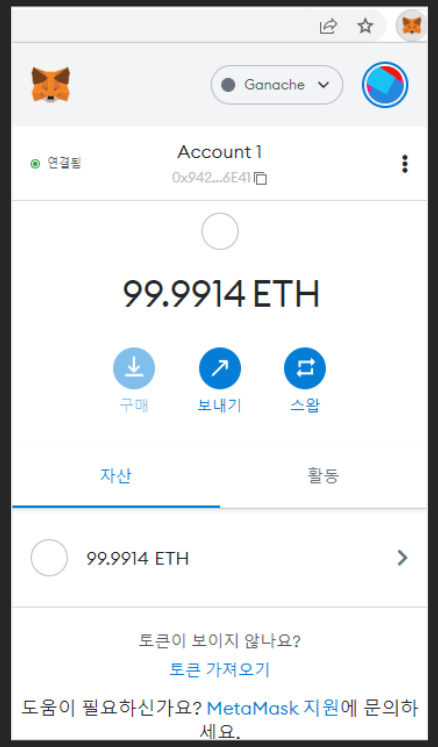


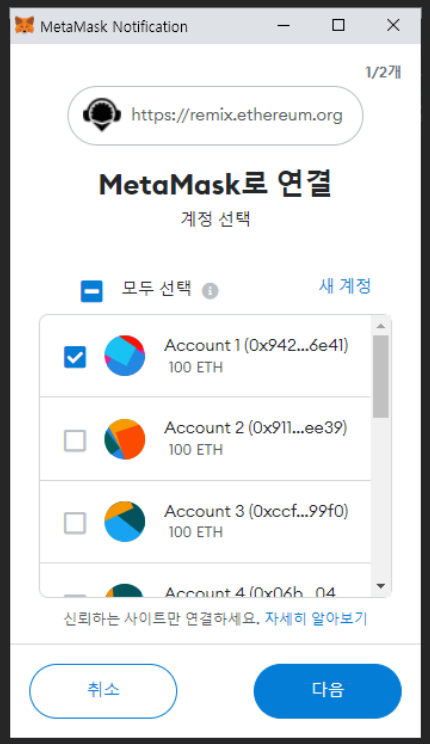
- 이더리움 JSON-RPC
  - Ethereum JSON-RPC Specification (JSON-RPC)
  - Http, WebSocket - CORS 지원

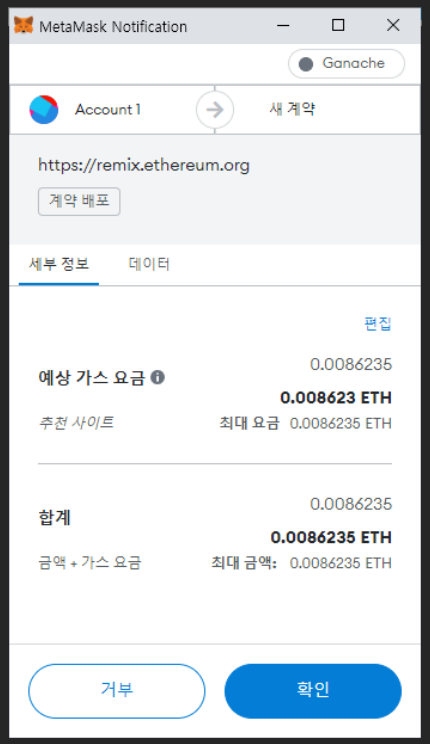
| Method | Description |
|---|---|
| eth_getBalance | 이더리움 잔액 조회 |
| eth_call | 조회 |
| eth_sendRawTransaction | 트랜잭션 제출 |

- 이더리움 API library & Wallet API
  - web3js (javascript), web3j (Java), web3.py (python), Nethereum (C#)

- 지갑 및 트랜잭션 서명
  - 메타마스크 (integration with web3js)



# • 이더리움 JSON-RPC API

| Method | Description |
|---|---|
| eth_getBlockByHash | Returns information about a block by hash. |
| eth_getBlockByNumber | Returns information about a block by number. |
| eth_getBlockTransactionCountByHash | Returns the number of transactions in a block from a block matching the given block hash. |
| eth_getBlockTransactionCountByNumber | Returns the number of transactions in a block matching the given block number. |
| eth_getUncleCountByBlockHash | Returns the number of uncles in a block from a block matching the given block hash. |
| eth_getUncleCountByBlockNumber | Returns the number of transactions in a block matching the given block number. |
| eth_protocolVersion | Returns the current Ethereum protocol version. |
| eth_chainId | Returns the chain ID of the current network. |
| eth_syncing | Returns an object with data about the sync status or false. |
| eth_coinbase | Returns the client coinbase address. |
| eth_accounts | Returns a list of addresses owned by client. |
| eth_blockNumber | Returns the number of most recent block. |
| eth_call | Executes a new message call immediately without creating a transaction on the block chain. |
| eth_estimateGas | Generates and returns an estimate of how much gas is necessary to allow the transaction to complete. |
| eth_gasPrice | Returns the current price per gas in wei. |
| eth_feeHistory | |
| eth_newFilter | Creates a filter object, based on filter options, to notify when the state changes (logs). |
| eth_newBlockFilter | Creates a filter in the node, to notify when a new block arrives. |
| eth_newPendingTransactionFilter | Creates a filter in the node, to notify when new pending transactions arrive. |
| eth_uninstallFilter | Uninstalls a filter with given id. |
| eth_getFilterChanges | Polling method for a filter, which returns an array of logs which occurred since last poll. |
| eth_getLogs | Returns an array of all logs matching filter with given id. |

# • 이더리움 JSON-RPC API

| Method | Description |
|---|---|
| eth_getLogs | Returns an array of all logs matching filter with given id. |
| eth_mining | Returns whether the client is actively mining new blocks. |
| eth_hashrate | Returns the number of hashes per second that the node is mining with. |
| eth_getWork | Returns the hash of the current block, the seedHash, and the boundary condition to be met (“target”). |
| eth_submitWork | Used for submitting a proof-of-work solution. |
| eth_submitHashrate | Used for submitting mining hashrate. |
| eth_hashrate | Returns the number of hashes per second that the node is mining with. |
| eth_getWork | Returns the hash of the current block, the seedHash, and the boundary condition to be met (“target”). |
| eth_submitWork | Used for submitting a proof-of-work solution. |
| eth_submitHashrate | Used for submitting mining hashrate. |
| eth_sign | Returns an EIP-191 signature over the provided data. |
| eth_signTransaction | Returns an RLP encoded transaction signed by the specified account. |
| eth_getBalance | Returns the balance of the account of given address. |
| eth_getStorageAt | Returns the value from a storage position at a given address. |
| eth_getTransactionCount | Returns the number of transactions sent from an address. |
| eth_getCode | Returns code at a given address. |
| eth_sendTransaction | Signs and submits a transaction. |
| eth_sendRawTransaction | Submits a raw transaction. |
| eth_getTransactionByHash | Returns the information about a transaction requested by transaction hash. |
| eth_getTransactionByBlockHashAndIndex | Returns information about a transaction by block hash and transaction index position. |
| eth_getTransactionByBlockNumberAndIndex | Returns information about a transaction by block number and transaction index position. |
| eth_getTransactionReceipt | Returns the receipt of a transaction by transaction hash. |

# web3.js - Ethereum JavaScript API

web3.js is a collection of libraries that allow you to interact with a local or remote ethereum node using HTTP, IPC or WebSocket.

## helloWorld.js

```
import Web3 from 'web3';
import { createRequire } from "module";
const require = createRequire(import.meta.url);
const helloWorldAbi = require("./contract/HelloWorld.json");

const web3 = new Web3(new
Web3.providers.HttpProvider('http://localhost:7545'));

(async () => {
  const to = '0x63c246Cdf8e5cF930bc988B466E44802FEeeb154';
  const contract = new web3.eth.Contract(helloWorldAbi, to);
  const result = await contract.methods.hello().call();
  console.log(result);
  // Hello, world
})();
```

## HelloWorld.sol

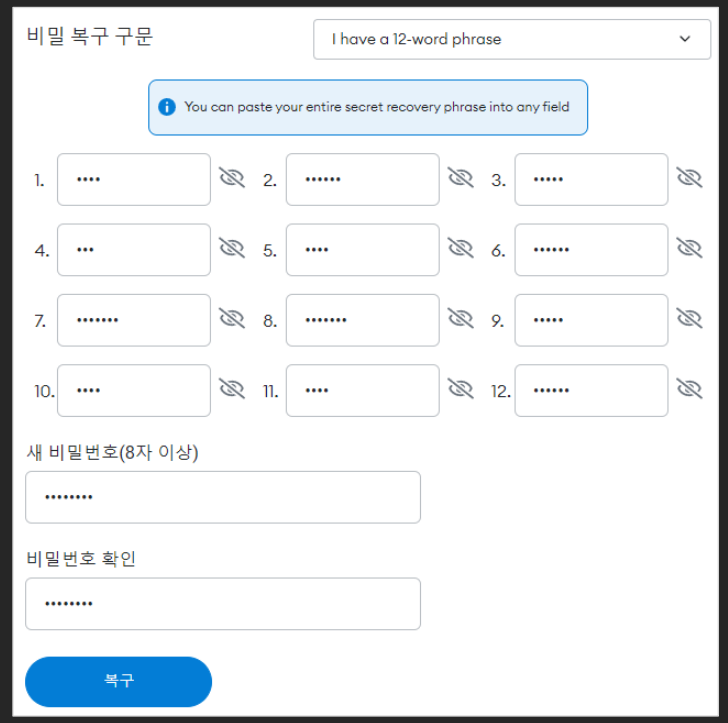
```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.13;

contract HelloWorld {
    function hello() public pure returns(string memory) {
        return "Hello, world";
    }
}
```

## HelloWorld.json, ABI(Application Binary Interface)

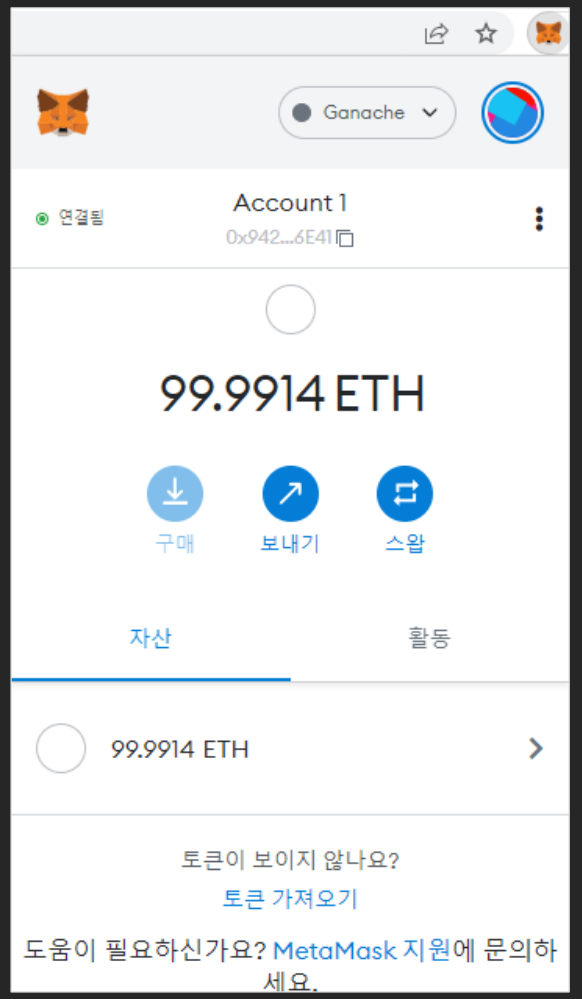
```
[
  {
    "type": "function",
    "name": "hello",
    "inputs": [],
    "outputs": [
      {
        "internalType": "string",
        "name": "",
        "type": "string"
      }
    ],
    "stateMutability": "pure"
  }
]
```

- Metamask : EVM Compatible Blockchain Wallet
  - 브라우저 확장 프로그램

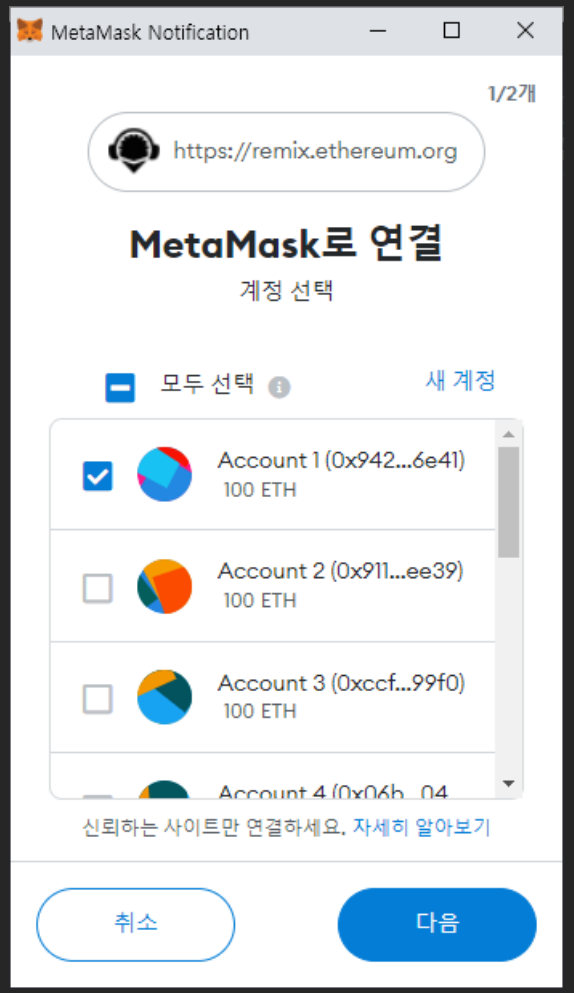
## HD Wallet (mnemonic)



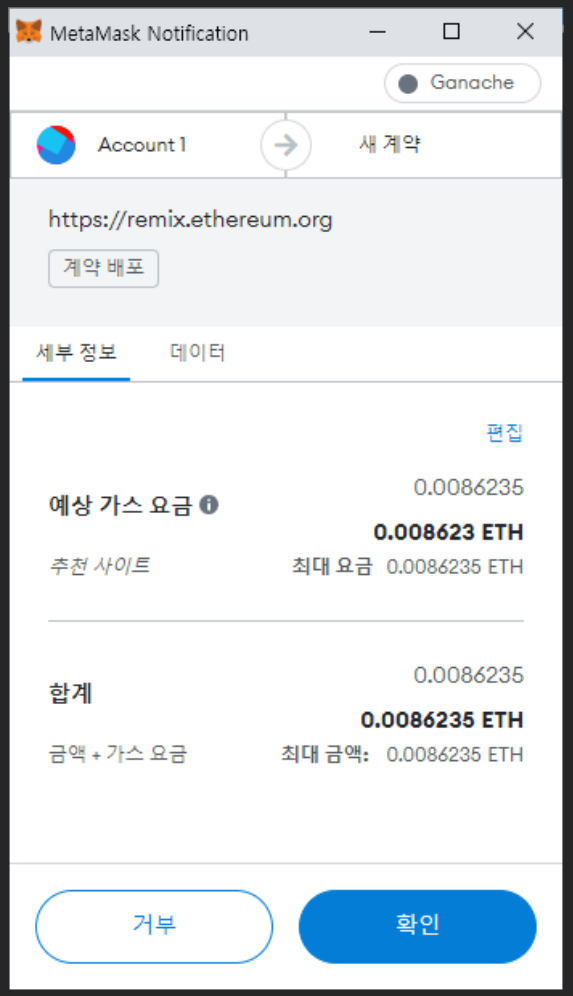

## Connect to Node (http)

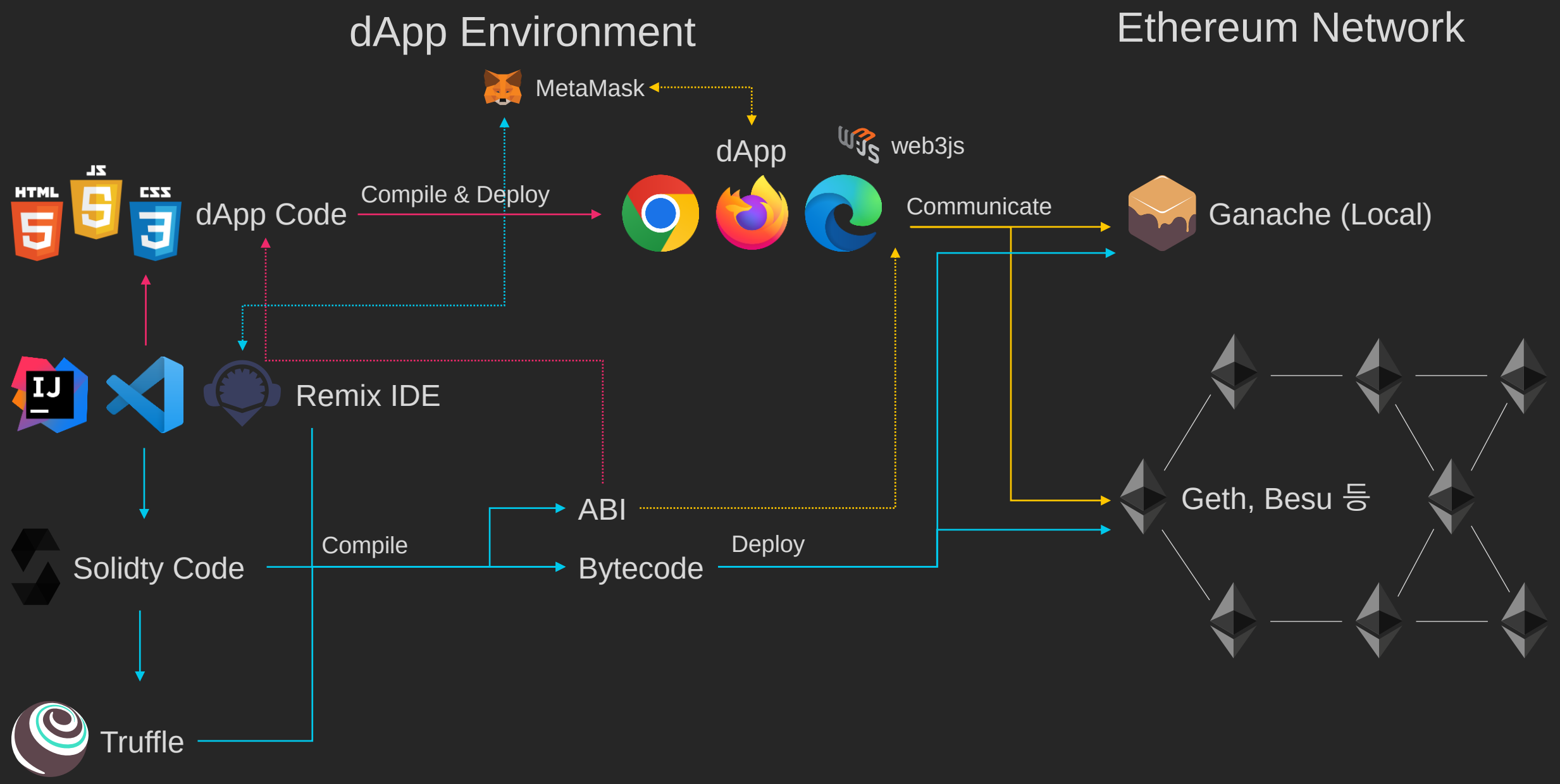



## Web3js Provider (dApp)




## Transaction Sign




dApp Environment
Ethereum Network
MetaMask
dApp
web3js
dApp Code
Compile & Deploy
Communicate
Ganache (Local)
Remix IDE
ABI
Compile
Deploy
Solidty Code
Bytecode
Geth, Besu 등
Truffle

- ERC-20 : 이더리움 (Fungible) Token Standard
  - 최소 기능 인터페이스 (잔액조회, 전송)

```
interface IERC20 {
    event Transfer(address indexed from, address indexed to, uint256 value);  // 전송 이벤트

    function balanceOf(address account) external view returns (uint256);      // 잔액 조회
    function transfer(address to, uint256 amount) external returns (bool);    // 전송
}
```

  - 전체 인터페이스

```
interface IERC20 {
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address to, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address from, address to, uint256 amount) external returns (bool);
}
```

## 샘플 스마트 컨트랙트 [Token.sol]

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.13;
import "./IERC20.sol";

contract Token is IERC20 {
    mapping(address => uint256) private _balances;  // 주소별 잔액

    constructor() {  // 생성자
        _balances[msg.sender] = 10000;  // 컨트랙트 배포자에게 10000 지급
    }

    function balanceOf(address account) public view returns (uint256) {  // 잔액 조회
        return _balances[account];
    }

    function transfer(address to, uint256 amount) public returns (bool) {  // 이체
        require(to != address(0), "ERC20: transfer to the zero address");
        address from = msg.sender;

        uint256 fromBalance = _balances[from];
        require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
        _balances[from] = fromBalance - amount;
        _balances[to] += amount;

        emit Transfer(from, to, amount);  // Transfer 이벤트(로그)
        return true;
    }
}
```

## Bytecode

```
6080604052348015610010576000 80fd5b50612
7106000803373ffffffffffffffffffffffffff
ffffffffffffff1673ffffffffffffffffffffff
ffffffffffffffffffff16815260200190815260
20016000208190555061067c806100656000396
000f3fe6080604052348015610010576000 80fd
5b506004361061003657600035 60e01c806370a
082311461003b578063a9059cbb1461006b575b
600080fd5b61005560048036038101906100509
190610355565b61009b565b604051610062919 0
61039b565b604051809103 90f35b61008560048
0360381019061008091906103e2565b6100e356
5b604051610092919061043d565b604051809 10
390f35b60008060008373ffffffffffffffffff
ffffffffffffffffffffff1673ffffffffffffff
ffffffffffffffffffffffffff168152602001
90815260200160002054905091905056 5b60008
073ffffffffffffffffffffffffffffffffffff
ffff168373ffffffffffffffffffffffffffff
ffffffffff1603610153576040517f08c379a0
000000000000000000000000000000000000000
000000000000000081526004016 1014a906104
db565b604051809103 90fd5b600033905060008
060008373ffffffffffffffffffffffffff…
```

## Assembly View

```
0000   60  PUSH1 0x80
0002   60  PUSH1 0x40
0004   52  MSTORE
0005   34  CALLVALUE
0006   80  DUP1
0007   15  ISZERO
0008   61  PUSH2 0x0010
000B   57  *JUMPI
000C   60  PUSH1 0x00
000E   80  DUP1
000F   FD  *REVERT
0010   5B  JUMPDEST
0011   50  POP
0012   61  PUSH2 0x2710
0015   60  PUSH1 0x00
0017   80  DUP1
0018   33  CALLER
0019   73  PUSH20 0xffffffffffffffffffffffffffffffffffffffff
002E   16  AND
002F   73  PUSH20 0xffffffffffffffffffffffffffffffffffffffff
0044   16  AND
0045   81  DUP2
0046   52  MSTORE
0047   60  PUSH1 0x20
0049   01  ADD
004A   90  SWAP1
004B   81  DUP2
004C   52  MSTORE
004D   60  PUSH1 0x20
004F   01  ADD
0050   60  PUSH1 0x00
0052   20  SHA3
0053   81  DUP2
0054   90  SWAP1
0055   55  SSTORE
0056   50  POP
0057   61  PUSH2 0x067c
005A   80  DUP1
005B   61  PUSH2 0x0065
005E   60  PUSH1 0x00
0060   39  CODECOPY
0061   60  PUSH1 0x00
0063   F3  *RETURN
0064   FE    *ASSERT
0065   60    PUSH1 0x80
0067   60    PUSH1 0x40
0069   52    MSTORE
006A   34    CALLVALUE
006B   80    DUP1
006C   15    ISZERO
006D   61    PUSH2 0x0010
0070   57    *JUMPI
0071   60    PUSH1 0x00
0073   80    DUP1
0074   FD    *REVERT
0075   5B    JUMPDEST
0076   50    POP
0077   60    PUSH1 0x04
0079   36    CALLDATASIZE
007A   10    LT
007B   61    PUSH2 0x0036
007E   57    *JUMPI
007F   60    PUSH1 0x00
0082   60    PUSH1 0xe0
0084   1C    SHR
0085   80    DUP1
0086   63    PUSH4 0x70a08231
                ⋮
```

https://ethervm.io/decompile

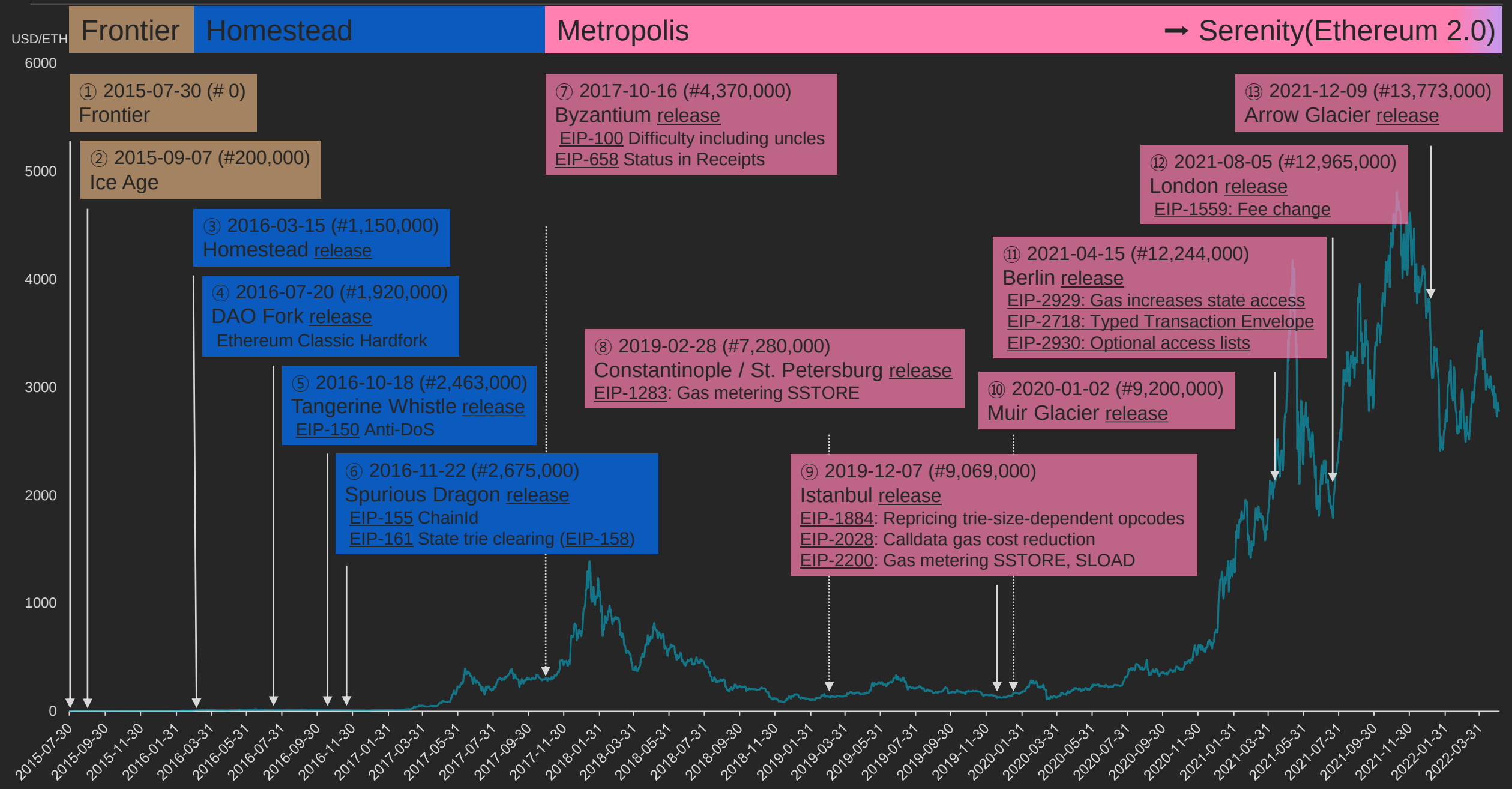
# Contract ABI (Application Binary Interface) docs

```
[
  {
    "type": "event",
    "name": "Transfer",
    "inputs": [
      { "type": "address", "name": "from", "indexed": true, "internalType": "address" },
      { "type": "address", "name": "to", "indexed": true, "internalType": "address" },
      { "type": "uint256", "name": "value", "indexed": false, "internalType": "uint256" }
    ],
    "anonymous": false
  },
  {
    "type": "function",
    "name": "balanceOf",
    "inputs": [
      { "type": "address", "name": "account", "internalType": "address" }
    ],
    "outputs": [
      { "type": "uint256", "name": "", "internalType": "uint256" }
    ],
    "stateMutability": "view"
  },
  {
    "type": "function",
    "name": "transfer",
    "inputs": [
      { "type": "address", "name": "to", "internalType": "address" },
      { "type": "uint256", "name": "amount", "internalType": "uint256" }
    ],
    "outputs": [
      { "type": "bool", "name": "", "internalType": "bool" }
    ],
    "stateMutability": "nonpayable"
  }
]
```

```
interface IERC20 {
  event Transfer(address indexed from, address indexed to, uint256 value);

  function balanceOf(address account) external view returns (uint256);

  function transfer(address to, uint256 amount) external returns (bool);
}
```

# Ethereum Protocol Release (Hard Fork)

https://github.com/ethereum/execution-specs#ethereum-protocol-releases



## [Frontier]

**Frontier** [ 2015-07-30 15:26:13 +UTC, #0 ]

**Ice Age** [ 2015-09-07 21:33:09 +UTC, #200,000 ]

## [Homestead]

**Homestaed** release [ 2016-03-14 18:49:53 +UTC, #1,150,000 ]

- EIP-2: Homestead Hard-fork Changes
- EIP-7: DELEGATECALL
- EIP-8: Networking layer: devp2p Forward Compatibility Requirements for Homestead

**DAO Fork** release [2016-07-20 13:20:40 +UTC, #1,920,000 ]

Ethereum Classic Hardfork

**Tangerine Whistle** release [ 2016-10-18 13:19:31 PM +UTC, #2,463,000 ]

- EIP-150: Gas cost changes for IO-heavy operations

**Spurious Dragon** release [ 2016-11-22 16:15:44 +UTC, #2,675,000 ]

- EIP-155: Simple replay attack protection
- EIP-160 : EXP cost increase
- EIP-161: State trie clearing (EIP-158)
- EIP-170: Contract code size limit

## [Metropolis]

**Byzantium** release [ 2017-10-16 05:22:11 +UTC, #4,370,000 ]

- EIP-100: Change difficulty adjustment to target mean block time including uncles
- EIP-140 : REVERT instruction
- EIP-196, EIP-197 : Precompiled contracts for addition and scalar multiplication, optimal ate pairing check on the elliptic curve alt_bn128
- EIP-198: Precompiled contract for bigint modular exponentiation
- EIP-211 : RETURNDATASIZE,RETURNDATACOPY
- EIP-214: STATICCALL
- EIP-649 : Difficulty Bomb Delay and Block Reward Reduction
- EIP-658 : Embedding transaction status code in receipts

**Constantinople/St. Petersburg** release [2019-02-28 19:52:04 +UTC, #7,280,000 ]

- EIP-145: Bitwise shifting instructions in EVM
- EIP-1014: Skinny CREATE2
- EIP-1052: EXTCODEHASH Opcode
- EIP-1234: Delay difficulty bomb, adjust block reward
- EIP-1283: Net gas metering for SSTORE without dirty maps

**Istanbul** release [ 2019-12-08 12:25:09 +UTC, #9,069,000 ]

- EIP-152: Add Blake2 compression function F precompile
- EIP-1108: Reduce alt_bn128 precompile gas costs
- EIP-1344: Add ChainID opcode
- EIP-1884: Repricing for trie-size-dependent opcodes
- EIP-2028: Calldata gas cost reduction
- EIP-2200: Rebalance net-metered SSTORE gas cost with consideration of SLOAD gas cost change

**Muir Glacier** release [ 2020-01-02 08:30:49 +UTC, #9,200,000 ]

- EIP-2384: Istanbul/Berlin Difficulty Bomb Delay

**Berlin** release [ 2021-04-15 10:07:03 +UTC, #12,244,000 ]

- EIP-2565: ModExp Gas Cost
- EIP-2929: Gas cost increases for state access opcodes
- EIP-2718: Typed Transaction Envelope
- EIP-2930: Optional access lists

**London** release [ 2021-08-05 12:33:42 +UTC, #12,965,000 ]

- EIP-1559: Fee market change for ETH 1.0 chain
- EIP-3198: BASEFEE opcode
- EIP-3529: Reduction in refunds
- EIP-3541: Reject new contracts starting with the 0xEF byte
- EIP-3554: Difficulty Bomb Delay to December 1st 2021

**Arrow Glacier** release [ 2021-12:09 19:55:23 +UTC, #13,773,000 ]

- EIP-4345: Difficulty Bomb Delay to June 2022

- 챕터 2 : Keccak-256, ECDSA, RLP, ABI

# 해시(Hash ) 함수

임의의 길이를 갖는 임의의 데이터에 대해 고정된 길이의 데이터로 매핑하는 단방향 함수

$$f(x) = x \% 7$$

임의의 정수에 대해 0 ~ 6 사이의 숫자로 매핑하는 단방향 함수

## Hash Table

해시 함수: $f(x) = x \% 7$ , 값 : 2, 7, 15, 33, -1, 14

- 2 % 7 = 2
- 7 % 7 = 0
- 15 % 7 = 1
- 33 % 7 = 5
- -1 % 7 = 6
- 14 % 7 = 0 : 해시 충돌(Hash Collision) (해시 함수가 서로 다른 입력 값(7,14)에 대해 동일한 출력 값(0)는 내는 상황)

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Value | 7 | 15 | 2 | | | 33 | -1 |

# 암호화 해시 함수(Cryptographic Hash Function)

해시 함수의 일종으로, 해시 값으로부터 원래의 입력 값과의 관계를 찾기 어려운 성질을 가지는 해시 함수

- 역상 저항성(preimage resistance): 주어진 해시 값에 대해, 그 해시 값을 생성하는 입력 값을 찾는 것이 계산상 어렵다.
- 제 2 역상 저항성(second preimage resistance): 입력 값에 대해, 그 입력의 해시 값을 바꾸지 않고 입력을 변경하는 것이 계산상 어렵다.
- 충돌 저항성(collision resistance): 해시 충돌에 대해 안전해야 한다. 같은 해시 값을 생성하는 두 개의 입력 값을 찾는 것이 계산상 어렵다.

**Hash(message) = digest**

**역상 저항성**

Hash(?) = digest

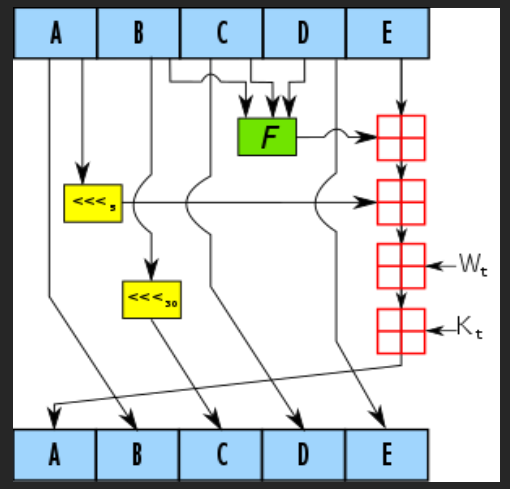
**제 2 역상 저항성**

Hash(message) = digest
Hash(?) = digest

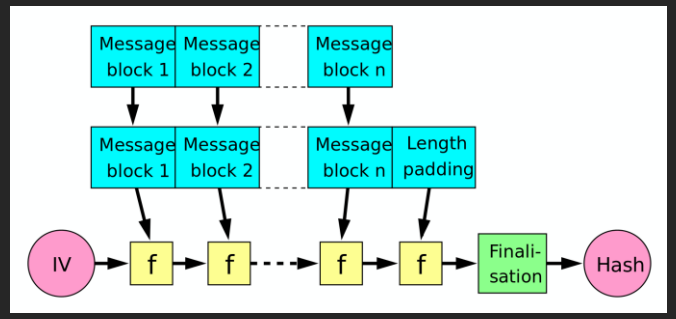
**충돌 저항성**

Hash(1?) = Hash(2?)

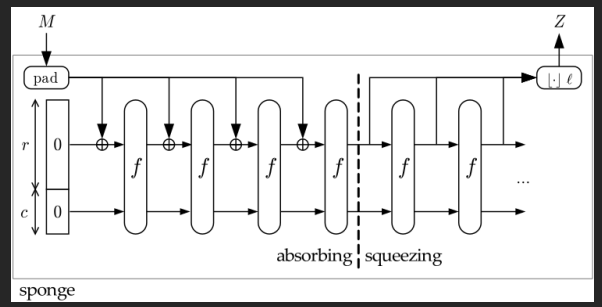
- SHA(Secure Hash Algorithm) (SHA-0) [Input : $2^{64}$bit → Output:160bit]
  - 미국 국가안보국(NSA)가 1993년 처음 설계했으면 미국 국가 표준으로 지정
- SHA-1 [Input : $2^{64}$bit → Output:160bit]
  - 1995년 SHA-0 변형 발표 (보안 강화)
  - 2017-02-23 Google Announcing the first SHA1 collision 링크
- SHA-2
  - 2001년 SHA1 변형 발표
  - SHA-224, SHA-256, SHA-384, SHA-512
  - 비트코인 : SHA-256(SHA-256(message)) , Genesis Block (2009-01-03)
    - 2번: Merkle-Damgard construction 기반 알고리즘(MD5, SHA1, SHA2)은 Length extension attack 가능성 존재
- Keccak
  - 2007-11-02 NIST이 SHA2의 잠재적 충돌 우려로 공개적으로 SHA-3 대회 개최, SHA-3 Project
  - 2012-10-02 Keccak이 우승 알고리즘으로 선정, Announcement
  - 고정길이: keccak-224, keccak-256, keccak-384, keccak-512 | 가변길이: SHAKE-128, SHAKE-256
  - 이더리움 : keccak256(message) , Genesis Block (2015-07-30)
- SHA-3
  - 2015-08-05 NIST가 Keccak을 수정하여 발표, Announcement
  - 고정길이: SHA3-224, SHA3-256, SHA3-384, SHA3-512 | 가변길이: SHAKE-128, SHAKE-256



SHA-1 압축 함수가 블록 하나를 처리하는 과정



Merkle–Damgård hash construction



Keccak (sponge function)

# 이더리움, Keccak, SHA-3

- 2007년 NIST SHA-3 암호화 해시 함수 경쟁 대회 개최
- Keccak은 SHA-3 후보 알고리즘으로 설계 및 우승
- 이더리움을 개발하는 기간동안 미국 표준기술연구소(NIST)의 표준화 미완료
- NIST 주장 지연 사유 : 표준 프로세스를 완료한 후 Keccak의 일부 파라미터를 조정하여 효율성을 향상
- 내부 고발 : 에드워드 스노든(Deward Snowden) 이 미국 국가안보국(National Security Agency)의 부적절한 영향을 받아 Dual_EC_DRBG 난수 생성기 표준을 의도적으로 약화시켜 표준 난 수 생성기에 백도어를 효과적으로 배치했다는 문서 공개
- 변경에 대한 반발로 인해 SHA-3의 표준화에 상당한 지연을 초래
- 이더리움 재단은 NIST에서 수정한 SHA-3 표준이 아닌,
  알고리즘 발명가들이 제안한 원래 Keccak 알고리즘을 구현하기로 결정
- Keccak은 SHA-3로 2015년 FIPS 202로 표준화
- 중요한 점 : Keccak ≠ SHA-3

# 쇄도 효과 (avalanche effect) (산사태 효과)

어떤 암호 알고리즘이 입력값에 미세한 변화를 줄 경우 출력값에 상당한 변화가 일어나는 성질

```
message1              : {"from":"A","to":"B","value:100}
message2              : {"from":"A","to":"C","value:100}


SHA-256(message1)     : 0x57e08177d123681fe427bd5ca289fdcb68d199e1fbd0e48cbd9e8766454f448e   58/64 = 90.6%
SHA-256(message2)     : 0x3da09727786e150ffd594bc0e6fe671c2ca2db115bac663a3f54a36da9ec62e9


Keccak-256(message1)  : 0x284693edfb1d4aa0201888bc2e03084e08c708d9f7936efa136a060c88c85757   62/64 = 96.9%
Keccak-256(message2)  : 0xc19a6194cdbf9f29bc290f58f00f3cf6a246d41b866cd761a2566c24fcc47fd8


SHA3-256(message1)    : 0xfb2ca149b1938f3d1a7d1b3be530cef98244c8ce99cee49ce2b91c8653a43d98   63/64 = 98.4%
SHA3-256(message2)    : 0xa47b2604a0da8b9abd23f8d252927956e55bae95250867dd558f6237357cfb44


Keccak-256(message1)  : 0x284693edfb1d4aa0201888bc2e03084e08c708d9f7936efa136a060c88c85757   63/64 = 98.4%
SHA3-256(message1)    : 0xfb2ca149b1938f3d1a7d1b3be530cef98244c8ce99cee49ce2b91c8653a43d98
```

online-tools

- $E(\mathbb{F}_p)$ : 소수 유한체 상 타원곡선
  - $\mathbb{F}_p$ : $p$가 홀수 소수인 소수 유한체(prime finite field)
  - $a, b \in \mathbb{F}_p$ , $4a^3 + 27b^2 \not\equiv 0 \ (mod\ p)$ : 비특이 타원 곡선 만족(x축 대칭)

$$y^2 \equiv x^3 + ax + b \ (mod\ p)$$

$$\|$$

$$y^2\ mod\ p = (x^3 + ax + b)\ mod\ p$$

$$\|$$

$$y^2 = (x^3 + ax + b)\ over\ (\mathbb{F}_p)$$

# K = k·G (G는 알려진 점)

k를 알면 K를 빠르게 계산 가능

K를 알아도 k를 계산하는 것은 어려움

k : 개인키 (G를 더하는 횟수)

K: 공개키 (G를 k번 더한 좌표)

# Elliptic Curve Domain Parameter

- ECC 를 사용하려면 타원곡선을 정의하는 "Domain Parameter" 공유 필요
- Prime Field 의 Domain Parameter 는 $(p, a, b, G, n, h)$ 로 구성 $E(\mathbb{F}_p): y^2 \equiv x^3 + ax + b \ (mod\ p)$
  - $p$ : Modulo Prime Number
  - $a$ : 타원곡선 방정식에 사용되는 계수
  - $b$ : 타원곡선 방정식에 사용되는 계수
  - $G$ : Base point 또는 Generator Point, $G$ 는 $\mathrm{E}(\mathbb{F}_p)$ 에 속해 있는 점 (생성자 점)
  - $n$ : the order of point $G$ ($G$를 n번 더하면, 무한원점이 되는 값 : $\mathrm{n}G = 0$)
  - $h$ : cofactor, $\mathrm{h} = \#E(\mathbb{F}^p)/n$ (공개키 복구에 사용)
- 여러 표준 단체에서 각 Field Size에 맞는 타원곡선에 대한 Domain Parameter 발표
  - value $(p, a, b, G, n, h)$ 나 name(p-256 or secp256k1)로 표시
  - FIPS 186-4 및 SEC2 v2.0 에서는 5개의 Prime Field를 recommend
    - p size 5개 : 192bit , 224bit , 256bit , 384bit , 521bit
    - 각 Prime Field에 대하여 하나의 타원곡선을 recommend

- secp256k1 : 비트코인, 이더리움 사용 EC Domain Parameter

### 2.4.1 Recommended Parameters secp256k1

The elliptic curve domain parameters over $\mathbb{F}_p$ associated with a Koblitz curve `secp256k1` are specified by the sextuple $T = (p, a, b, G, n, h)$ where the finite field $\mathbb{F}_p$ is defined by:

$$p = \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F}$$
$$= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

The curve $E$: $y^2 = x^3 + ax + b$ over $\mathbb{F}_p$ is defined by:

$$a = \texttt{00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000}$$
$$b = \texttt{00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007}$$

The base point $G$ in compressed form is:

$$G = \texttt{02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798}$$

and in uncompressed form is:

$$G = \texttt{04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8}$$

Finally the order $n$ of $G$ and the cofactor are:

$$n = \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141}$$
$$h = \texttt{01}$$

- Domain Parameter
  - a = 0
  - b = 7
  - p = $2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
  - G =
    (79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798,
    483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8)
  - n =
    FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141
- 타원 곡선 방정식

$$y^2 \equiv x^3 + 7 \ (mod\ p)$$

- K = kG
  - k = 개인키 ( 0 < k < n)
  - K = kG = 공개키

- 개인키 k = 0x1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
- 공개키 K = k * G = (x, y)

  x = 0xF028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A

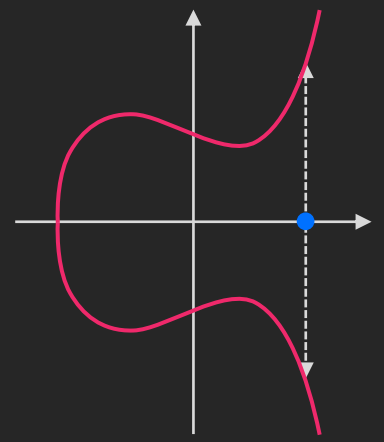
  y = 0x07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB

- 공개키 Serialize (Standards for Efficient Cryptography 1, SEC 1, www.secg.org/sec1-v2.pdf)

| 접두어 | 의미 | 길이 (접두어 포함 바이트 수) |
|---|---|---|
| 0x00 | 무한원점 | 1 |
| 0x04 | 압축되지 않은 점 | 65 |
| 0x02 | y값이 짝수인 압축된 점 | 33 |
| 0x03 | y값이 홀수인 압축된 점 | 33 |

K = (x, y) → 0x04{x}{y}

- y가 짝수 → 0x02{x}
- y가 홀수 → 0x03{x}



압축된 점은 y좌표의 부호 저장 필요(양수 또는 음수)
타원곡선 상 y좌표는 짝수 또는 홀수이며 각각 양수, 음수에 해당

압축되지 않은 점 : 0x04{x}{y}

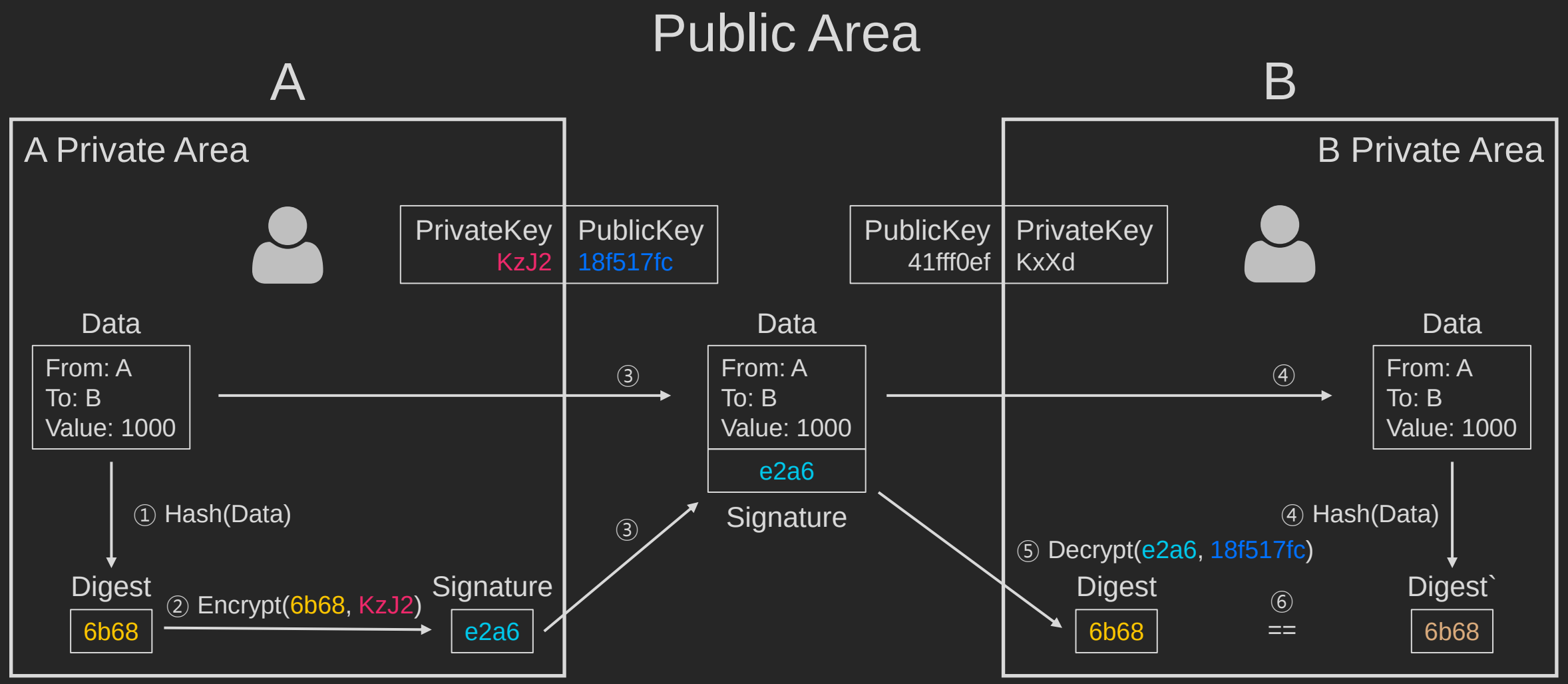
0x04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB

압축된 점 : {prefix}{x} - y값 마지막 hex ➡ 0xB = 11 ➡ 홀수

0x03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A

① Data를 Hash 함수로 Digest 생성
② Digest를 PrivateKey로 암호화하여 Signature 생성
③ Data와 Signature를 같이 전송

④ 전송받은 Data를 Hash 함수로 Digest` 생성
⑤ Signature를 PublicKey로 복호화하여 Digest 획득
⑥ Digest == Digest` 가 성립하면 위·변조 되지 않음



- ECDSA /w secp256k1

private key  Public Key

m = 서명 없는 Tx

$$Sig = F_{sig}(Hash(m), privateKey)$$

Hash
비트코인: SHA-256(SHA-256(m))
이더리움: keccak-256(m)

$$Sig = (r, s)$$

Tx
(r, s, Public Key)

(r, s)를
Public Key로 Verify

## ECDSA Signing

1. 일시적(ephemeral ) 개인키를 안전한 방법으로 생성
   (같은 일시적 개인키를 2번 사용하면 안된다, 개인키 유출)
2. $q$ : 일시적 개인키
3. $Q$ : $q$ 로 생성된 일시적 공개키 $Q$
4. $r$ : $Q$ 의 x 좌표 (서명 결과 중 하나)
5. $s$ 를 계산

$$s \equiv q - 1(Hash(m) + r * k) \bmod n$$

- $q$ : 일시적 개인키
- $r$ : 일시적공개키의 x 좌표
- $k$ : 서명자의 개인키
- $m$ : 트랜잭션 데이터
- $n$ : 타원 곡선의 소수 차수

## ECDSA Verifying

$r, s$ 및 서명자의 공개키를 사용하여 $Q$ 를 계산 후,
$r$ 과 $Q$ 의 x 좌표가 동일하면 서명은 유효

1. 모든 입력 유효성 확인
2. $w = s^{-1} \bmod n$
3. $u_1 = Hash(m) * w \bmod n$
4. $u_2 = r * w \bmod n$
5. $Q$ 계산

$$Q \equiv u_1 * G + u_2 * K \pmod n$$

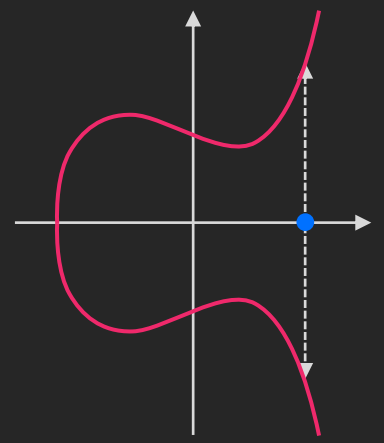
- $r, s$ : 서명 값
- $K$ : 서명자 공개키
- $m$ : 트랜잭션 데이터
- $G$ : 타원 곡선상 생성자 점
- $n$ : 타원 곡선의 소수 차수

# 공개키 복구(The Public Key Recovery) : r, s 로 2개의 가능한 공개키 계산 가능

1. x 좌표인 $r$ 값의 2개의 타원 곡선 점 $R$과 $R'$ 계산
2. $r^{-1}$ : $r$ 의 곱셈의 역함수
3. $z$ : 메시지 Hash의 최하위 n bits (n은 타원 곡선 차수)
4. 가능한 2개의 공개키 ($K_1$ : Q의 y좌표 홀수, $K2$ : Q의 y좌표 짝수 )

$$K_1 = r^{-1}(sR - zG)$$
$$K_2 = r^{-1}(sR' - zG)$$

- $K_1$ 과 $K_2$는 서명자의 2개의 가능한 공개키
- $r^{-1}$ : 서명 $r$ 값의 곱셈 역함수
- $s$ : 서명 s
- $R$ 과 $R'$ : 2개의 가능한 일시적 공개키
- $z$ : 메시지 해시의 n 최하위 bits
- $G$ : 타원 곡선의 생성자 점



압축된 점은 y좌표의 부호 저장 필요(양수 또는 음수)
타원곡선 상 y좌표는 짝수 또는 홀수이며 각각 양수, 음수에 해당

5. 이더리움은 트랜잭션에 Q의 y좌표 짝수, 홀수 여부를 포함(yParity, 0: 짝수, 1: 홀수)

# RLP

# (Recursive Length Prefix Serialize)

- 이더리움 객체 직렬화
- 임의 중첩된 Item을 binary data로 인코딩
- 인코딩할 데이터 앞에 binary data의 길이를 추가하는 방식으로 동작하기 때문에 Length Prefix 이름을 붙음
- Item
  - 문자열 (String, byte array)
  - 배열 (Array, list of items)
- 양의 정수는 선행 0이 없는 big-endian 이진 형으로 표현
  - 1 ➞ 0x01 ➞ RLP(1) = 0x01, 255 ➞ 0xff ➞ RLP(255) = 0x81ff
  - 0 ➞ 0x00 ➞ 0x ➞ [ ] (empty byte array) = "" (empty string) ➞ RLP(0) = RLP("") = 0x80

| Prefix | 유형 | 설명 | 포맷 |
|---|---|---|---|
| `[0x00, 0x7f]` | 문자열 | 문자 한 개 (ASCII 코드) | `[0x00, 0x7f]`, 그대로 사용 |
| `[0x80, 0xb7]` | 문자열 | 길이 55 bytes 이하 문자열 | (`0x80` + 문자열 길이) + 문자열 |
| `[0xb8, 0xbf]` | 문자열 | 길이 56 bytes 초과 문자열 | (`0xb7` + 문자열 길이 표현 바이트 수) + 문자열 길이 + 문자열 |
| `[0xc0, 0xf7]` | 배열 | 전체 길이 55 bytes 이하 배열 | (`0xc0` + 배열 바이트길이) + 배열 바이트 (아이템들의 RLP 인코딩 값) |
| `[0xf8, 0xff]` | 배열 | 전체 길이 55 bytes 초과 배열 | (`0xf7` + 배열 바이트 길이 표현 바이트 수) + 배열 바이트 길이 + 배열 바이트 |

[ 0, 1, "0", "1", "00", "01" ] (LP : Length Prefix)

배열 아이템 Serialize [ LP(0x80) , LP(0x01) , LP(0x30) , LP(0x31) , LP(0x82) Data(0x3030) , LP(0x82) Data(0x3031) ]

배열 Serialize
0xc0[배열] + 0x0a(10)

LP(0xca) Data(0x80013031823030823031)

0xca80013031823030823031

배열 Deserialize [ LP(0x80) , LP(0x01) , LP(0x30) , LP(0x31) , LP(0x82) Data(0x3030) , LP(0x82) Data(0x3031) ]

배열 아이템 Deserialize [ 0, 1, "0", "1", "00", "01" ]

| Prefix | 유형 | 설명 | 포맷 |
|---|---|---|---|
| [0x00, 0x7f] | 문자열 | 문자 한 개 (ASCII 코드) | [0x00, 0x7f], 그대로 사용 |
| [0x80, 0xb7] | 문자열 | 길이 55 bytes 이하 문자열 | (0x80 + 문자열 길이) + 문자열 |
| [0xb8, 0xbf] | 문자열 | 길이 56 bytes 초과 문자열 | (0xb7 + 문자열 길이 표현 바이트 수) + 문자열 길이 + 문자열 |
| [0xc0, 0xf7] | 배열 | 전체 길이 55 bytes 이하 배열 | (0xc0 + 배열 바이트길이) + 배열 바이트 (아이템들의 RLP 인코딩 값) |
| [0xf8, 0xff] | 배열 | 전체 길이 55 bytes 초과 배열 | (0xf7 + 배열 바이트 길이 표현 바이트 수) + 배열 바이트 길이 + 배열 바이트 |

1. [0x00, 0x7f] , 문자열 , 길이 1bytes and [0x00, 0x7f]
   : 그대로 사용
   - RLP("a") = RLP(0x61) ➞ 길이 : 1 bytes, 0x00 <= 0x61 <= 0x7f
     ➞ 0x61

2. [0x80, 0xb7] , 문자열 , 길이 55 bytes 이하 문자열
   : (0x80 + 문자열 길이) + 문자열
   - RLP("abc") = RLP(0x616263) = RLP([0x61, 0x62, 0x63]) ➞ 길이 : 3 bytes <= 55 bytes
     ➞ (0x80 + 3) + 0x616263
     ➞ 0x83616263
   - RLP("") = RLP([ ]) = RLP(0x) ➞ 길이: 0 bytes ➞ [0x80 + 0] + 0x = 0x80

| Prefix | 유형 | 설명 | 포맷 |
|---|---|---|---|
| `[0x00, 0x7f]` | 문자열 | 문자 한 개 (ASCII 코드) | `[0x00, 0x7f]`, 그대로 사용 |
| `[0x80, 0xb7]` | 문자열 | 길이 55 bytes 이하 문자열 | (`0x80` + 문자열 길이) + 문자열 |
| `[0xb8, 0xbf]` | 문자열 | 길이 56 bytes 초과 문자열 | (`0xb7` + 문자열 길이 표현 바이트 수) + 문자열 길이 + 문자열 |
| `[0xc0, 0xf7]` | 배열 | 전체 길이 55 bytes 이하 배열 | (`0xc0` + 배열 바이트길이) + 배열 바이트 (아이템들의 RLP 인코딩 값) |
| `[0xf8, 0xff]` | 배열 | 전체 길이 55 bytes 초과 배열 | (`0xf7` + 배열 바이트 길이 표현 바이트 수) + 배열 바이트 길이 + 배열 바이트 |

## 3. [0xb8, 0xbf] , 문자열 , 길이 55 bytes 초과 문자열

: (0xb7 + 문자열 길이 표현 바이트 수) + 문자열 길이 + 문자열

- RLP("abcdefghijabcdefghijabcdefghijabcdefghijabcdefghijabcdefghij") = RLP("abcdefghij" × 6)

= RLP(0x6162636465666768696a6162636465666768696a6162636465666768696a6162636465666768696a6162636465666768696a6162636465666768696a)

→ 길이 : 60 bytes > 55 bytes , 60 = 0x3c → 1 bytes

→ (0xb7 + 1) + 0x3c + 0x6162636465666768696a…6162636465666768696a

→ 0xb83c6162636465666768696a6162636465666768696a6162636465666768696a6162636465666768696a6162636465666768696a

| Prefix | 유형 | 설명 | 포맷 |
|---|---|---|---|
| `[0x00, 0x7f]` | 문자열 | 문자 한 개 (ASCII 코드) | `[0x00, 0x7f]`, 그대로 사용 |
| `[0x80, 0xb7]` | 문자열 | 길이 55 bytes 이하 문자열 | (0x80 + 문자열 길이) + 문자열 |
| `[0xb8, 0xbf]` | 문자열 | 길이 56 bytes 초과 문자열 | (0xb7 + 문자열 길이 표현 바이트 수) + 문자열 길이 + 문자열 |
| `[0xc0, 0xf7]` | 배열 | 전체 길이 55 bytes 이하 배열 | (0xc0 + 배열 바이트길이) + 배열 바이트 (아이템들의 RLP 인코딩 값) |
| `[0xf8, 0xff]` | 배열 | 전체 길이 55 bytes 초과 배열 | (0xf7 + 배열 바이트 길이 표현 바이트 수) + 배열 바이트 길이 + 배열 바이트 |

## 4. [0xc0, 0xf7] , 배열 , 전체 길이 55 bytes 이하 배열

: (0xc0 + 배열 바이트길이) + 배열 바이트 (아이템들의 RLP 인코딩 값)

- RLP(["a", "abc"])
  - → RLP("a") = 0x61, RLP("abc") = 0x83616263
  - → 0x61 + 0x83616263 = 0x6183616263 → 길이 : 5 byes <= 55 bytes
  - → (0xc0 + 5) + 0x6183616263
  - → 0xc56183616263

| Prefix | 유형 | 설명 | 포맷 |
|---|---|---|---|
| `[0x00, 0x7f]` | 문자열 | 문자 한 개 (ASCII 코드) | `[0x00, 0x7f]`, 그대로 사용 |
| `[0x80, 0xb7]` | 문자열 | 길이 55 bytes 이하 문자열 | (`0x80` + 문자열 길이) + 문자열 |
| `[0xb8, 0xbf]` | 문자열 | 길이 56 bytes 초과 문자열 | (`0xb7` + 문자열 길이 표현 바이트 수) + 문자열 길이 + 문자열 |
| `[0xc0, 0xf7]` | 배열 | 전체 길이 55 bytes 이하 배열 | (`0xc0` + 배열 바이트길이) + 배열 바이트 (아이템들의 RLP 인코딩 값) |
| `[0xf8, 0xff]` | 배열 | 전체 길이 55 bytes 초과 배열 | (`0xf7` + 배열 바이트 길이 표현 바이트 수) + 배열 바이트 길이 + 배열 바이트 |

## 5. [0xf8, 0xff] , 배열 , 전체 길이 55 bytes 초과 배열 :
## (0xf7 + 배열 바이트 길이 표현 바이트 수) + 배열 바이트 길이 + 배열 바이트

- RLP(["a", "abc“, "abcdefghij" × 6])

➞ RLP("a") = 0x61, RLP("abc") = 0x83616263, RLP("abcdefghij" × 6) = 0xb83c616263…68696a

➞ 0x61 + 0x83616263 + 0xb83c616263…68696a = 0x6183616263b83c616263…68696a

➞ 길이 : 1 + 4 + 62 = 67 > 55 bytes , 67 = 0x43 ➞ 1 bytes

➞ (0xf7 + 0x01) + 0x43 + 0x6183616263b83c616263…68696a

➞ 0xf8436183616263b83c6162636465666768696a6162636465666768696a6162636465666768696a6162636465666768696a6162636465666768696a

| Prefix | 유형 | 설명 | 포맷 |
|---|---|---|---|
| `[0x00, 0x7f]` | 문자열 | 문자 한 개 (ASCII 코드) | `[0x00, 0x7f]`, 그대로 사용 |
| `[0x80, 0xb7]` | 문자열 | 길이 55 bytes 이하 문자열 | (`0x80` + 문자열 길이) + 문자열 |
| `[0xb8, 0xbf]` | 문자열 | 길이 56 bytes 초과 문자열 | (`0xb7` + 문자열 길이 표현 바이트 수) + 문자열 길이 + 문자열 |
| `[0xc0, 0xf7]` | 배열 | 전체 길이 55 bytes 이하 배열 | (`0xc0` + 배열 바이트길이) + 배열 바이트 (아이템들의 RLP 인코딩 값) |
| `[0xf8, 0xff]` | 배열 | 전체 길이 55 bytes 초과 배열 | (`0xf7` + 배열 바이트 길이 표현 바이트 수) + 배열 바이트 길이 + 배열 바이트 |

- RLP Deserialize 0xc56183616263
  - 0xc5 → [0xc0, 0xf7] → 전체 길이 55 bytes 이하 배열
  - 배열 바이트 길이 = 0xc5 - 0xc0 = 0x05 = 5
  - 배열 바이트 = 0x6183616263
    - 0x61 → [0x00, 0x7f] → ASCII 코드 문자 한 개
      → "a"
    - 0x83 → [0x80, 0xb7] → 길이 55 bytes 이하 문자열
      → 길이 = 0x83 - 0x80 = 0x03 = 3
      → 문자열 바이트 = 0x616263 = "abc"
  - 결과 = ["a", "abc"]

| Prefix | 유형 | 설명 | 포맷 |
|---|---|---|---|
| `[0x00, 0x7f]` | 문자열 | 문자 한 개 (ASCII 코드) | `[0x00, 0x7f]`, 그대로 사용 |
| `[0x80, 0xb7]` | 문자열 | 길이 55 bytes 이하 문자열 | (`0x80` + 문자열 길이) + 문자열 |
| `[0xb8, 0xbf]` | 문자열 | 길이 56 bytes 초과 문자열 | (`0xb7` + 문자열 길이 표현 바이트 수) + 문자열 길이 + 문자열 |
| `[0xc0, 0xf7]` | 배열 | 전체 길이 55 bytes 이하 배열 | (`0xc0` + 배열 바이트길이) + 배열 바이트 (아이템들의 RLP 인코딩 값) |
| `[0xf8, 0xff]` | 배열 | 전체 길이 55 bytes 초과 배열 | (`0xf7` + 배열 바이트 길이 표현 바이트 수) + 배열 바이트 길이 + 배열 바이트 |

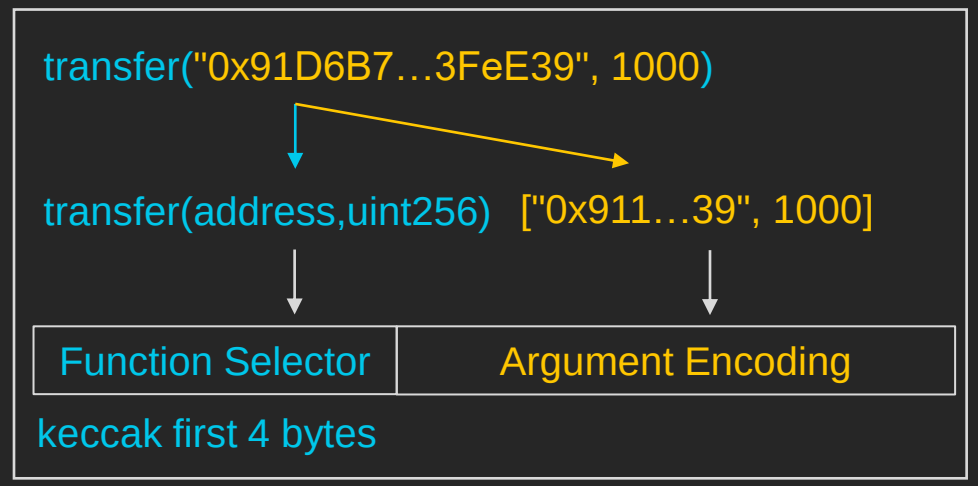
# ABI
# (Application Binary Interface)

# Contract ABI (Application Binary Interface) docs

```
[
  {
    "type": "event",
    "name": "Transfer",
    "inputs": [
      { "type": "address", "name": "from", "indexed": true, "internalType": "address" },
      { "type": "address", "name": "to", "indexed": true, "internalType": "address" },
      { "type": "uint256", "name": "value", "indexed": false, "internalType": "uint256" }
    ],
    "anonymous": false
  },
  {
    "type": "function",
    "name": "balanceOf",
    "inputs": [
      { "type": "address", "name": "account", "internalType": "address" }
    ],
    "outputs": [
      { "type": "uint256", "name": "", "internalType": "uint256" }
    ],
    "stateMutability": "view"
  },
  {
    "type": "function",
    "name": "transfer",
    "inputs": [
      { "type": "address", "name": "to", "internalType": "address" },
      { "type": "uint256", "name": "amount", "internalType": "uint256" }
    ],
    "outputs": [
      { "type": "bool", "name": "", "internalType": "bool" }
    ],
    "stateMutability": "nonpayable"
  }
]
```

```
interface IERC20 {
  event Transfer(address indexed from, address indexed to, uint256 value);

  function balanceOf(address account) external view returns (uint256);
  function transfer(address to, uint256 amount) external returns (bool);
}
```

- ABI : 이더리움 컨트랙트와 상호작용하기 위한 표준 방법 Specification
  - 컨트랙트의 함수 및 이벤트와 각각의 파라미터에 대한 MetaData를 정의
- ABI Encoding
  - Function Selector
    - 트랜잭션이 호출할 컨트랙트의 함수 선택
    - 함수 시그니처(signature)의 keccak-256 결과의 처음 4 bytes(32 bits, 8 nibbles)
  - Argument Encoding
    - 규칙에 따라 32 bytes(256 bits, 64 nibbles)의 배수의 bytes로 변환하여 function selector 뒤에 추가
  - 타입(Type)
    - 고정 크기 (fixed-size) 타입, 비고정 크기(non-fixed-size) 타입
    - 정적(static) 타입, 동적(dynamic) 타입



- ABI Encoding 예제
  - function transfer(address to, uint256 value)
    → transfer("0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39", 1000);

  - Function Selector (함수 시그니처 keccak-256 결과의 처음 4 bytes)
    - keccak-256("transfer(address,uint256)")
      = 0xa9059cbb2ab09eb219583f4a59a5d0623ade346d962bcd4e46b11da047c9049b
      → 0xa9059cbb
  - Argument Encoding
    - (address, uint256)
    - (0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39, 1000)
    - → 0x000000000000000000000000911d6b77014fa58afd85be49e5148cbeaa3fee39000000000000000000
      000000000000000000000000

- ABI Encoding 예제
  - function transfer(address to, uint256 value)
    → transfer("0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39", 1000);
  - Function Selector
    - keccak-256("transfer(address,uint256)")
      = 0xa9059cbb2ab09eb219583f4a59a5d0623ade346d962bcd4e46b11da047c9049b
      → 0xa9059cbb
  - Argument Encoding
    - address : 0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39
      → 0x000000000000000000000000911d6b77014fa58afd85be49e5148cbeaa3fee39
    - uint256 : 1000 → 0x3e8
      → 0x0000000000000000000000000000000000000000000000000000000000000003e8
  - ABI Encoding = Function Selector + Argument Encoding
    - 0xa9059cbb000000000000000000000000911d6b77014fa58afd85be49e5148cbeaa3fee39000000000000000000000000000000000000000000000000000000000000003e8

# • 정적 타입(Static Type)

| Type | Description |
|---|---|
| uint<M> | M bits 부호 없는 정수, 0 < M <= 256, M % 8 == 0. 예) uint8, uint16, … , uint256 [left padding] |
| int<M> | M bits 부호 있는 정수, 0 < M <= 256, M % 8 == 0. 예) int8, int16, … , int256 [left padding] |
| uint, int | uint == uint256, int == int256 |
| address | 주소. address ➞ uint160 으로 처리 |
| bool | 불린. 참(true) 또는 거짓(false). bool ➞ uint8 으로 처리하고 값을 0(false) 또는 1(true)로 제한 |
| bytes<M> | M bytes binary. 0 < M <= 32. 예) bytes1, bytes2, … , bytes32 [right padding] |
| T[M] | 정적(static) 타입 T의 고정길이 배열, M >= 0 |
| fixed<M>x<N> | 부호 있는 고정 소수점 숫자 [solidity 미지원] |
| ufixed<M>x<N> | 부호 없는 고정 소수점 숫자 [solidity 미지원] |
| function | address(20bytes) + function selector(4bytes) ➞ bytes24 으로 처리 [solidity 미지원] |
| $(T_1,\ldots,T_k)$ | 튜플(Tuple). 임의 타입의 묶음<br>모든 타입 T가 정적 타입이면 Tuple도 정적 타입<br>1개 이상 타입이 동적 타입이면 Tuple도 동적 타입<br>예) transfer(address to, uint256 amount) ➞ (address, uint256) ➞ 정적 타입<br>예) (string, uint256) ➞ string은 동적타입 ➞ 동적 타입 |

- 정적 타입(Static Type) Encoding 예제

| Type | Value | Encoding Value |
|---|---|---|
| uint8 | 255 | 0x00000000000000000000000000000000000000000000000000000000000000ff |
| uint256 | 255 | 0x00000000000000000000000000000000000000000000000000000000000000ff |
| bool | false | 0x0000000000000000000000000000000000000000000000000000000000000000 |
| bool | true | 0x0000000000000000000000000000000000000000000000000000000000000001 |
| address | 0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39 | 0x000000000000000000000000911d6b77014fa58afd85be49e5148cbeaa3fee39 |
| bytes1 | 0xab | 0xab0000000000000000000000000000000000000000000000000000000000000000 |
| bytes32 | 0xabcdef | 0xabcdef0000000000000000000000000000000000000000000000000000000000 |
| uint[3] | [1,2,3] | 0x0000000000000000000000000000000000000000000000000000000000000010<br>0000000000000000000000000000000000000000000000000000000000000020<br>0000000000000000000000000000000000000000000000000000000000000003 |

- 동적(dynamic) Type
  - bytes : 동적 크기 바이트 시퀀스
  - string : 동적 크기 utf-8 캐릭터 시퀀스
  - T[ ] for any T : 임의 타입 T의 동적길이 배열, 예) uint256[ ]
  - T[k] for any dynamic T and k >= 0 : 동적 타입 T의 고정길이 배열, 예) string[2]
  - $(T_1,...,T_n)$ if $T_i$ is dynamic, 1 <= i < = k : 하나 이상이 동적 타입인 경우
    - 예) (address, uint256) ➞ 모두 정적 타입 ➞ tuple은 정적 타입
    - 예) (string, uint256) ➞ string은 동적 타입 ➞ tuple은 동적 타입

Solidity to ABI 타입 매핑

| Solidity | ABI |
|---|---|
| address payable | address |
| contract | address |
| enum | uint8 |
| struct | tuple |
| user defined value types | 기반 타입을 따름<br>예) type C is V ➞ V |

- bytes encoding [ enc(X) = enc(k) pad_right(X), k = number of bytes ]
  - {number of bytes : uint256} + {bytes with right padding : multiple of 32bytes}
  - 예) 0x616263 = [ 0x61, 0x62, 0x63 ]
    - length(0x616263) = 3
      → 0x0000000000000000000000000000000000000000000000000000000000000003
    - 0x616263 → right padding
      → 0x6162630000000000000000000000000000000000000000000000000000000000
    - 0x0000000000000000000000000000000000000000000000000000000000000003616263000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
- string encoding [ enc(X) = enc(enc_utf8(X) ]
  - string → convert to utf-8 bytes → bytes encoding
  - 예) "abc" → [ 0x61, 0x62, 0x63 ]
    - 0x0000000000000000000000000000000000000000000000000000000000000003616263000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

- T[ ] for any T [ enc(X) = enc(k) enc(X[0], … , X[k-1]) , k = size of array ]
  임의(정적, 동적) 타입 T의 동적길이 배열
  - {size of array: uint256} + {encoding(X[0])} + … + {encoding(X[k-1])}
  - 예) uint256[] = [1, 2, 3]
    - size of array = 3 → 0x0000000000000000000000000000000000000000000000000000000000000003
    - encoding(1) = 0x0000000000000000000000000000000000000000000000000000000000000001
    - encoding(2) = 0x0000000000000000000000000000000000000000000000000000000000000002
    - encoding(3) = 0x0000000000000000000000000000000000000000000000000000000000000003
    - 0x0000000000000000000000000000000000000000000000000000000000000003000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000003

- T[k] for static type T and k >= 0 [ enc(X) = enc((X[0], ..., X[k-1]))]
  정적 타입 T의 정적 길이 배열
  - {encoding(X[0])} + … + {encoding(X[k-1])}
  - 예) uint256[3] = [1, 2, 3]
    - encoding(1) = 0x0000000000000000000000000000000000000000000000000000000000000001
    - encoding(2) = 0x0000000000000000000000000000000000000000000000000000000000000002
    - encoding(3) = 0x0000000000000000000000000000000000000000000000000000000000000003
    - 0x0000000000000000000000000000000000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000000003

- len(a) : binary string a 의 byte 수 (type: uint256)
- 임의 ABI 값 X 에 대해, enc(X) 를 재귀적으로 정의
  - $X = (T_1, \ldots, Tk)$ k >= 0, T1, … , Tk 은 임의 타입

$$enc(X) = head(X(1)) \ldots head(X(k)) tail(X(1)) \ldots tail\ (X(k))$$

*transfer(address to, uint256 value) ➞ X = (address, uint256), k = 2*

$$enc((address, uint256)) = head(address) head(uint256) tail(address) tail\ (uint256)$$

- X(i) 가 i 번째 값 인 경우 ( 1 <= i <= k ) , end(X(i)) 는 X(i) 의 encoding 결과

| Type | head(X(i)) | tail(X(i)) |
|---|---|---|
| 정적(static) 타입 | enc(X(i)) | ""(empty) |
| 동적(dynamic) 타입 | enc(len( head(X(1)) … head(X(k)) tail(X(1)) … tail(X(i-1)) ))<br>= 전체 Argument Encoding **시작 부터 tail(X(i)) 까지** bytes 수 | enc(X(i)) |

$$enc(X) = head(X(1)) \ldots head(X(k)) tail(X(1)) \ldots tail\ (X(k))$$

function baz(uint32 x, bool y) ➞ baz(69, true)

- Function Selector : keccak-256("baz(uint32,bool)") = 0xcdcd77c0…
- Argument Encoding : (uint32, bool) ➞ (69, true)

| Type | Value | head | tail |
|---|---|---|---|
| uint32 | 69 | 0x0000000000000000000000000000000000000000000000000000000000000045 | "" |
| bool | true | 0x0000000000000000000000000000000000000000000000000000000000000001 | "" |

➞ "0x" + "000…045" + "000…001" + "" + "" = 0x000…045000…001

- ABI Encoding of baz(69, true)

0xcdcd77c0000000000000000000000000000000000000000000000000000000000000045000000000000000000000000000000000000000000000000000000000000001

$$enc(X) = head(X(1)) \dots head(X(k)) tail(X(1)) \dots tail\ (X(k))$$

function bar(bytes3[2] memory) → bar(["abc", "def"])

- Function Selector : keccak-256("bar(bytes3[2])") = 0xfce353f6…
- Argument Encoding : (byte3[2]) → (["abc", "def"])

| Type | Value | head | tail |
|---|---|---|---|
| bytes3[2] | ["abc", "def"] | abc : 0x616263000000000000000000000000000000000000000000000000000000<br>def : 646566000000000000000000000000000000000000000000000000000000 | "" |

→ "0x" + "616263…000000646566…000000" + "" = 0x616263…000000646566…000000

- ABI Encoding of bar(["abc", "def"])

0xfce353f6616263000000000000000000000000000000000000000000000000000000000000646566000000000000000000000000000000000000000000000000000000

$$enc(X) = head(X(1)) \ldots head(X(k)) tail(X(1)) \ldots tail\ (X(k))$$

function sam(bytes memory, bool, uint[] memory) → sam("dave", true, [1,2,3])

- Function Selector : keccak-256("sam(bytes,bool,uint[])") = 0xa5643bf2…
- Argument Encoding : (bytes, bool, uint[]) → ("dave", true, [1,2,3])

| Type | Value | head | tail |
|---|---|---|---|
| bytes | "dave" | 0x000…060 | 0x0000000000000000000000000000000000000000000000000000000000000004<br>6461766500000000000000000000000000000000000000000000000000000000 |
| bool | true | 0x000…001 | "" |
| uint[] | [1,2,3] | 0x000…0a0 | 0x0000000000000000000000000000000000000000000000000000000000000003<br>0000000000000000000000000000000000000000000000000000000000000001<br>0000000000000000000000000000000000000000000000000000000000000002<br>0000000000000000000000000000000000000000000000000000000000000003 |

$$head(X(i)) = enc(len(\ head(X(1)) \ldots head(X(k)) tail\ X(1))\ \ldots tail(X((i-1))\ ))$$

→ "0x" + "000…060" + "000…001" + "000…0a0" + "000…00464617665…000" + "" + "000…003000…001000…002000…003"

head(1) = 32 + 32 + 32 = 96 = 0x60

head(3) = 32 + 32 + 32 + 64 + 0 = 160 = 0xa0

- ABI Encoding of bar(["abc", "def"])

0xa5643bf2000…060000…001000…0a0000…004000…003000…001000…002000…3

$$enc(X) = head(X(1)) \dots head(X(k)) tail(X(1)) \dots tail\ (X(k))$$

function transfer(address to, uint256 value)
→ transfer("0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39", 1000)

- Function Selector : keccak-256("transfer(address,uint256)") = 0xa9059cbb…
- Argument Encoding : (address, uint256) → ("0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39", 1000)

| Type | Value | head | tail |
|---|---|---|---|
| address | 0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39 | 0x000000000000000000000000911d6b77014fa58afd85be49e5148cbeaa3fee39 | "" |
| uint256 | 1000 | 0x0000000000000000000000000000000000000000000000000000000000000003e8 | "" |

→ "0x" + "000000000000000000000000911d6b77014fa58afd85be49e5148cbeaa3fee39" +
"0000000000000000000000000000000000000000000000000000000000000003e8" + "" + ""

- ABI Encoding of transfer("0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39", 1000)

0xa9059cbb000000000000000000000000911d6b77014fa58afd85be49e5148cbeaa3fee39000000000000000000000000000000000000000000000000000000000000003e8

- 챕터 3 : World State, EVM, 트랜잭션(Transaction), 영수증(Receipt)

# World State

4.1. **World State.** The world state (*state*), is a mapping between addresses (160-bit identifiers) and account states (a data structure serialised as RLP, see Appendix B). Though not stored on the blockchain, it is assumed that the implementation will maintain this mapping in a modified Merkle Patricia tree (*trie*, see Appendix D). The trie requires a simple database backend that maintains a mapping of byte arrays to byte arrays; we name this underlying database the state database. This has a number of benefits; firstly the root node of this structure is cryptographically dependent on all internal data and as such its hash can be used as a secure identity for the entire system state. Secondly, being an immutable data structure, it allows any previous state (whose root hash is known) to be recalled by simply altering the root hash accordingly. Since we store all such root hashes in the blockchain, we are able to trivially revert to old states.

The account state, $\boldsymbol{\sigma}[a]$, comprises the following four fields:

**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. For account of address $a$ in state $\boldsymbol{\sigma}$, this would be formally denoted $\boldsymbol{\sigma}[a]_{\mathrm{n}}$.

**balance:** A scalar value equal to the number of Wei owned by this address. Formally denoted $\boldsymbol{\sigma}[a]_{\mathrm{b}}$.

**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), 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. The hash is formally denoted $\boldsymbol{\sigma}[a]_{\mathrm{s}}$.

**codeHash:** The hash of the EVM code of this account—this is 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. This hash is formally denoted $\boldsymbol{\sigma}[a]_{\mathrm{c}}$, and thus the code may be denoted as $\mathbf{b}$, given that $\mathtt{KEC}(\mathbf{b}) = \boldsymbol{\sigma}[a]_{\mathrm{c}}$.

Since we typically wish to refer not to the trie's root hash but to the underlying set of key/value pairs stored within, we define a convenient equivalence:

$$\mathtt{TRIE}\big(L_{\mathrm{I}}^{*}(\boldsymbol{\sigma}[a]_{\mathbf{s}})\big) \equiv \boldsymbol{\sigma}[a]_{\mathrm{s}} \tag{7}$$

The collapse function for the set of key/value pairs in the trie, $L_{\mathrm{I}}^{*}$, is defined as the element-wise transformation of the base function $L_{\mathrm{I}}$, given as:

$$L_{\mathrm{I}}\big((k, v)\big) \equiv \big(\mathtt{KEC}(k), \mathtt{RLP}(v)\big) \tag{8}$$

where:

$$k \in \mathbb{B}_{32} \quad \wedge \quad v \in \mathbb{N} \tag{9}$$

It shall be understood that $\boldsymbol{\sigma}[a]_{\mathbf{s}}$ is not a 'physical' member of the account and does not contribute to its later serialisation.

If the **codeHash** field is the Keccak-256 hash of the empty string, i.e. $\boldsymbol{\sigma}[a]_{\mathrm{c}} = \mathtt{KEC}\big(()\big)$, then the node represents a simple account, sometimes referred to as a "non-contract" account.

Thus we may define a world-state collapse function $L_{\mathrm{S}}$:

$$L_{\mathrm{S}}(\boldsymbol{\sigma}) \equiv \{p(a) : \boldsymbol{\sigma}[a] \neq \varnothing\} \tag{10}$$

where

$$p(a) \equiv \big(\mathtt{KEC}(a), \mathtt{RLP}\big((\boldsymbol{\sigma}[a]_{\mathrm{n}}, \boldsymbol{\sigma}[a]_{\mathrm{b}}, \boldsymbol{\sigma}[a]_{\mathrm{s}}, \boldsymbol{\sigma}[a]_{\mathrm{c}})\big)\big) \tag{11}$$

This function, $L_{\mathrm{S}}$, is used alongside the trie function to provide a short identity (hash) of the world state. We assume:

$$\forall a : \boldsymbol{\sigma}[a] = \varnothing \ \vee \ (a \in \mathbb{B}_{20} \ \wedge \ v(\boldsymbol{\sigma}[a])) \tag{12}$$

where $v$ is the account validity function:

$$v(x) \equiv x_{\mathrm{n}} \in \mathbb{N}_{256} \wedge x_{\mathrm{b}} \in \mathbb{N}_{256} \wedge x_{\mathrm{s}} \in \mathbb{B}_{32} \wedge x_{\mathrm{c}} \in \mathbb{B}_{32} \tag{13}$$

An account is *empty* when it has no code, zero nonce and zero balance:

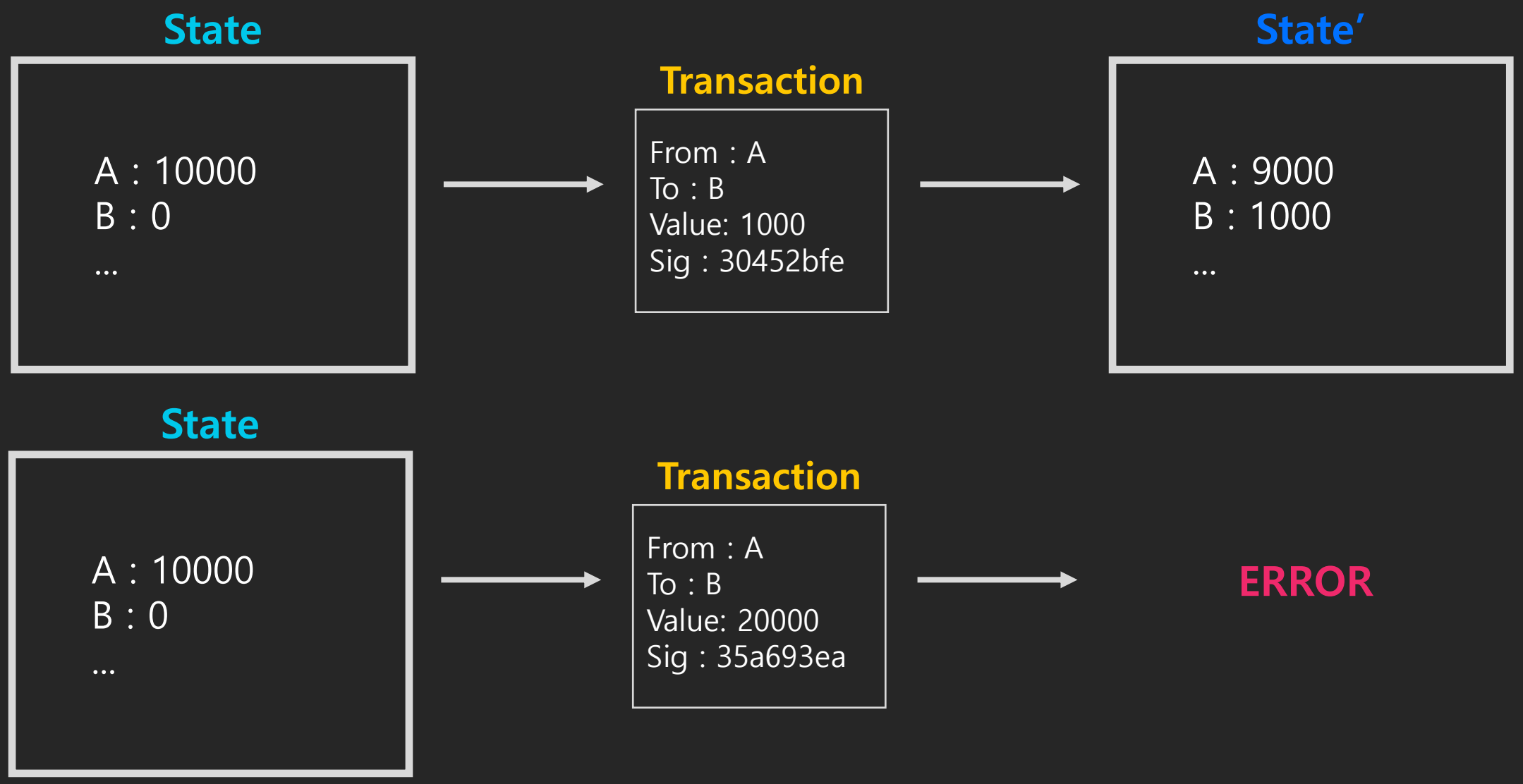
$$\mathtt{EMPTY}(\boldsymbol{\sigma}, a) \quad \equiv \quad \boldsymbol{\sigma}[a]_{\mathrm{c}} = \mathtt{KEC}\big(()\big) \wedge \boldsymbol{\sigma}[a]_{\mathrm{n}} = 0 \wedge \boldsymbol{\sigma}[a]_{\mathrm{b}} = 0 \tag{14}$$

Even callable precompiled contracts can have an empty account state. This is because their account states do not usually contain the code describing its behavior.

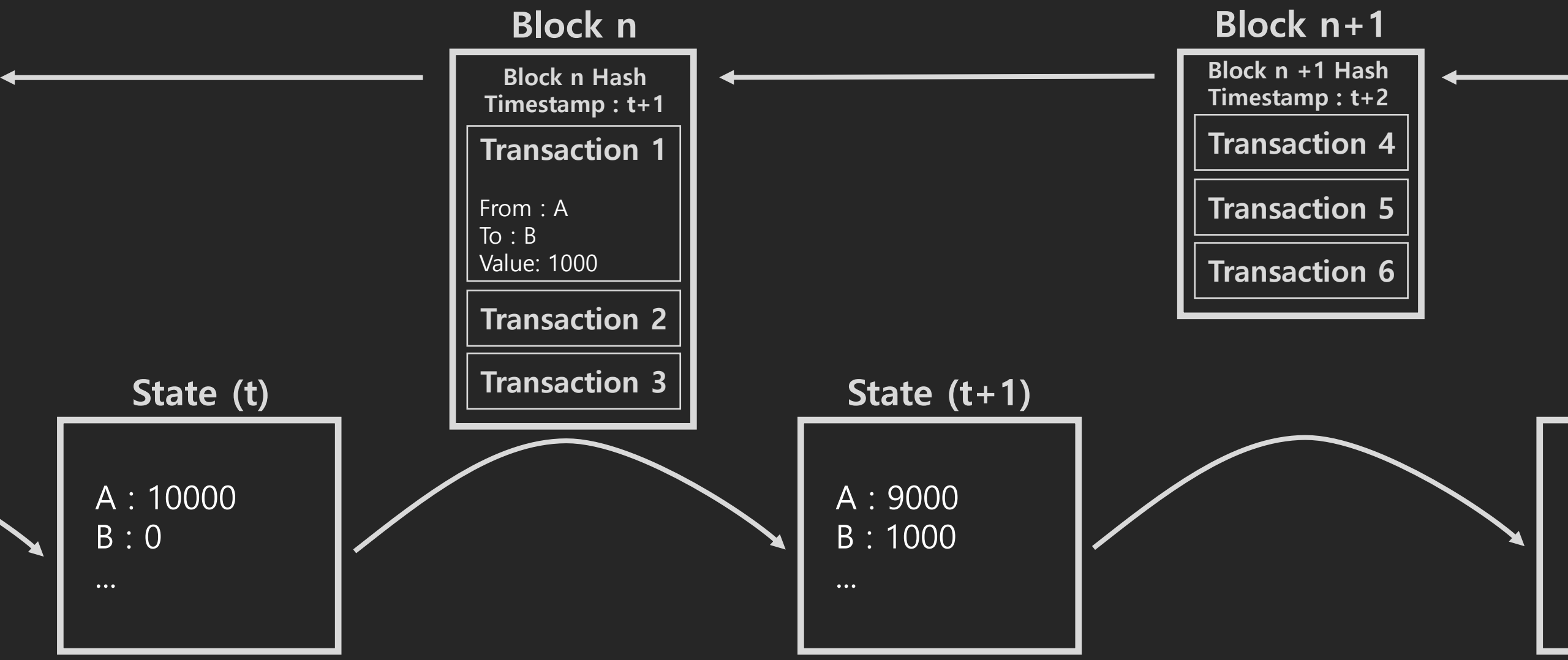
An account is *dead* when its account state is non-existent or empty:

$$\mathtt{DEAD}(\boldsymbol{\sigma}, a) \quad \equiv \quad \boldsymbol{\sigma}[a] = \varnothing \vee \mathtt{EMPTY}(\boldsymbol{\sigma}, a) \tag{15}$$

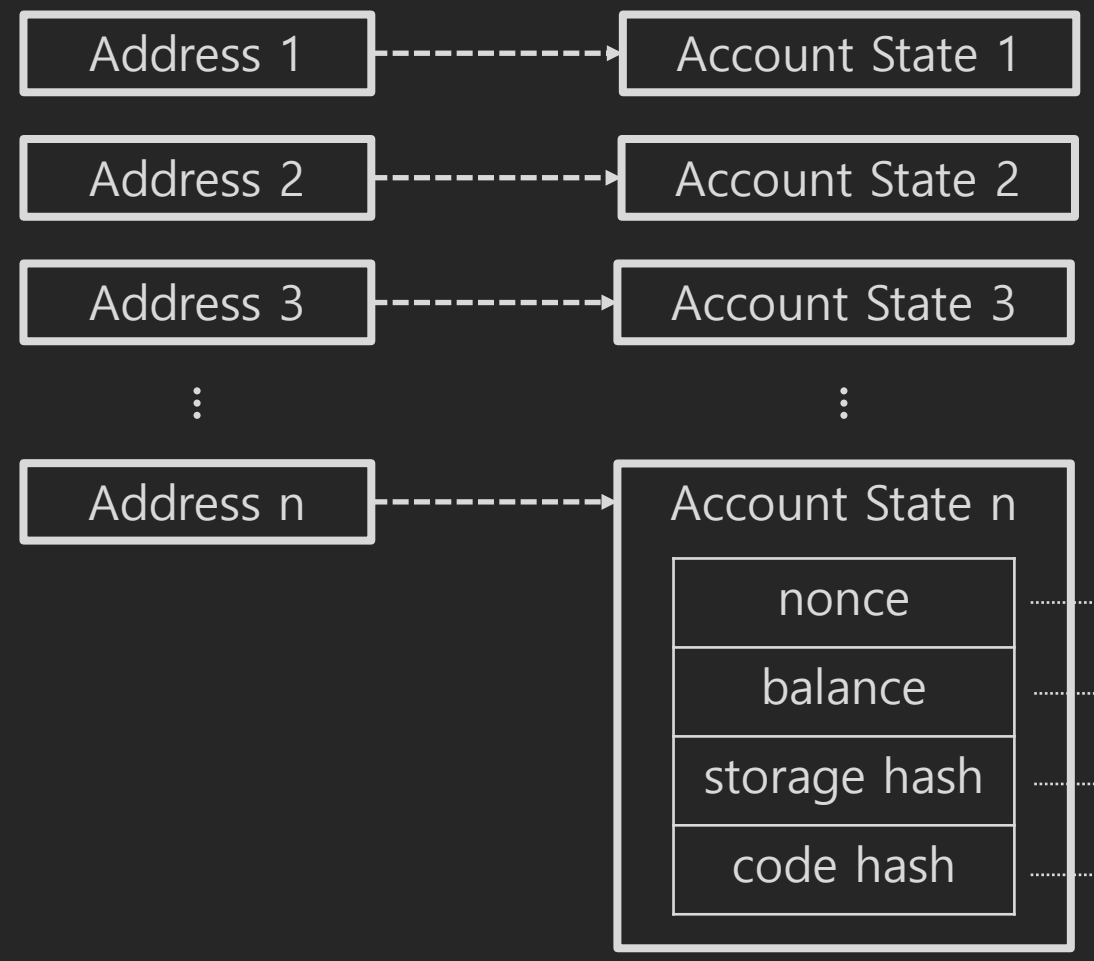
# APPLY(S,TX) → S' or ERROR



# Blockchain


Block n
Block n Hash
Timestamp : t+1
Transaction 1
From : A
To : B
Value: 1000
Transaction 2
Transaction 3
Block n+1
Block n +1 Hash
Timestamp : t+2
Transaction 4
Transaction 5
Transaction 6
State (t)
A : 10000
B : 0
...
State (t+1)
A : 9000
B : 1000
...

# World State




1 wei
1 Gwei = $1*10^9$ wei
1 szabo = $1*10^{12}$ wei
1 finney = $1*10^{15}$ wei
1 ether(ETH) = $1*10^{18}$ wei

제출한 트랜잭션 수 (중복, 건너뛰기 불가), 0 → 1 → ···

ETH 잔액 (wei 단위) (1 ether = $1\times10^{18}$ wei)

Account storage

EVM bytecode

## External Owned Account (EOA)

외부 사용자 → Private Key

ECDSA (secp256k1)
random 256 bits < n

↓

Public Key

↓

keccak256(public key) → right 160bits

Address

↓

Account State

| |
|---|
| nonce (default: 0) |
| balance |
| storage hash (null) |
| code hash ("") |

## Contract Account (CA)

Sender address | nonce | init code

↓

CREATE
keccak256(RLP([sender, nonce])) → right 160bits

CREATE2 (EIP-1014)
keccak256(0xff+sender+salt+keccak256(init code)) → right 160bits

Address

↓

Account State

| |
|---|
| nonce (default: 1) |
| balance |
| storage hash |
| code hash |

storage hash → storage

code hash → bytecode

```
CreateDataGas        uint64 = 200   //
```

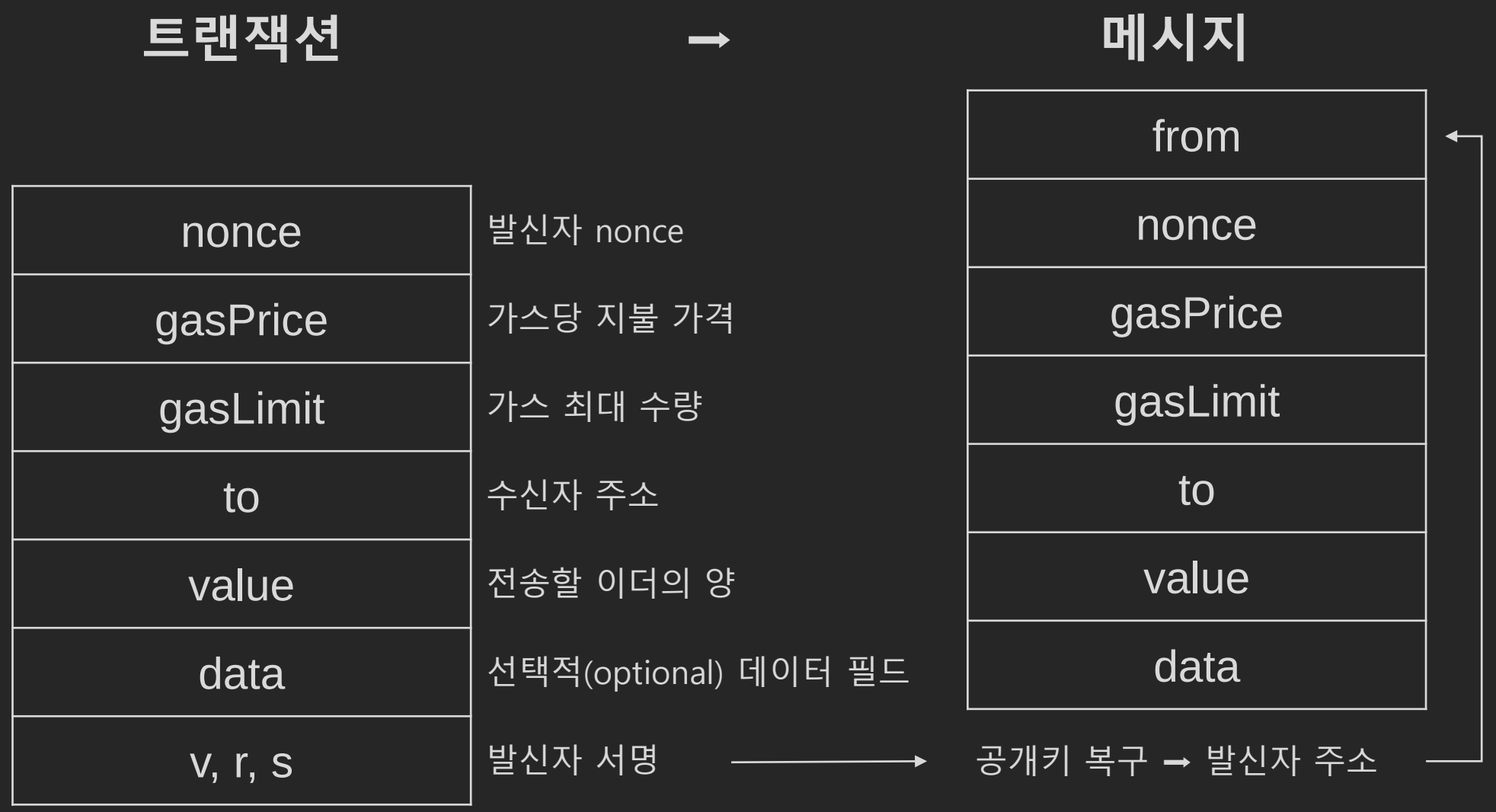
- Contract Create Data는 Runtime byte당 200 가스?

```
// Create a new account on the state
snapshot := evm.StateDB.Snapshot()
evm.StateDB.CreateAccount(address)
if evm.chainRules.IsEIP158 {
   evm.StateDB.SetNonce(address, 1)
}
evm.Context.Transfer(evm.StateDB, caller.Address(), address, value)
```

```
once := evm.StateDB.GetNonce(caller.Address())
f nonce+1 < nonce {
  return nil, common.Address{}, gas, ErrNonceUintOverflow

vm.StateDB.SetNonce(caller.Address(), nonce+1)
/ We add this to the access list _before_ taking a snapshot. Even if the creation fails,
/ the access-list change should not be rolled back
f evm.chainRules.IsBerlin {
  evm.StateDB.AddAddressToAccessList(address)

/ Ensure there's no existing contract already at the designated address
ontractHash := evm.StateDB.GetCodeHash(address)
f evm.StateDB.GetNonce(address) != 0 || (contractHash != (common.Hash{}) && contractHash != emptyCodeHash) {
  return nil, common.Address{}, 0, ErrContractAddressCollision

/ Create a new account on the state
napshot := evm.StateDB.Snapshot()
vm.StateDB.CreateAccount(address)
f evm.chainRules.IsEIP158 {
  evm.StateDB.SetNonce(address, 1)

vm.Context.Transfer(evm.StateDB, caller.Address(), address, value)

/ Initialise a new contract and set the code that is to be used by the EVM.
/ The contract is a scoped environment for this execution context only.
ontract := NewContract(caller, AccountRef(address), value, gas)
ontract.SetCodeOptionalHash(&address, codeAndHash)
```

```
err := evm.interpreter.Run(contract, nil, false)

eck whether the max code size has been exceeded, assign err if the
 == nil && evm.chainRules.IsEIP158 && len(ret) > params.MaxCodeSiz
 = ErrMaxCodeSizeExceeded


ject code starting with 0xEF if EIP-3541 is enabled.
 == nil && len(ret) >= 1 && ret[0] == 0xEF && evm.chainRules.IsLon
 = ErrInvalidCode


 the contract creation ran successfully and no errors were returned
culate the gas required to store the code. If the code could not
 stored due to not enough gas set an error and let it be handled
 the error checking condition below.
 == nil {
eateDataGas := uint64(len(ret)) * params.CreateDataGas
 contract.UseGas(createDataGas) {
 evm.StateDB.SetCode(address, ret)
lse {
 err = ErrCodeStoreOutOfGas
```

- 트랜잭션 : EOA 가 보낼 메시지를 포함하는 서명된 데이터 패키지
- 메시지 : Account 에서 Account로 전달되어 상태 전이(State Transition) 발생



# 트랜잭션 유형 2가지 [ Message call, Contract creation ]

triggered by EOA

## Message call

Transaction

EOA —Message→ EOA or CA

World State

Transaction

value | input data

Address 1 → Account State 1
Address 2 → Account State 2
⋮
Address n → Account State n

code | storage

World State (t)

Address 1 → Account State 1
Address 2 → Account State 2
⋮
Address n → Account State n

update

code | storage

World State (t + 1)

## Contract creation

Transaction

Create

Contract account

World State

Transaction

init code

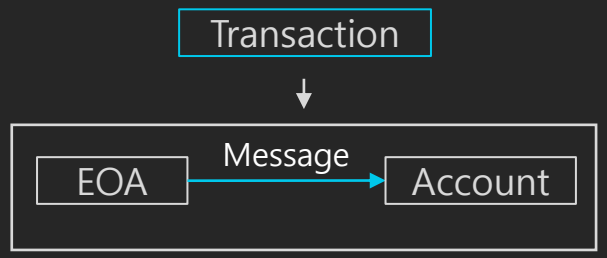
Address 1 → Account State 1
Address 2 → Account State 2
⋮

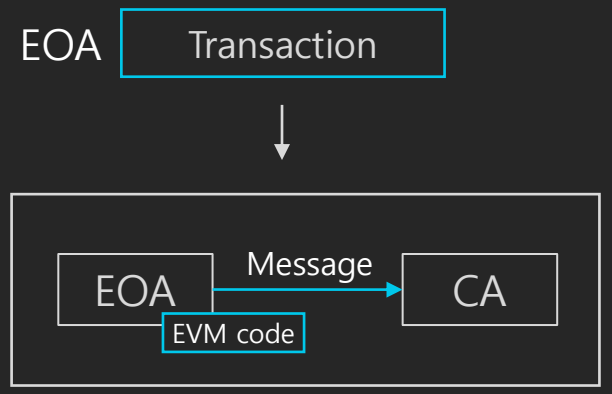
World State (t)

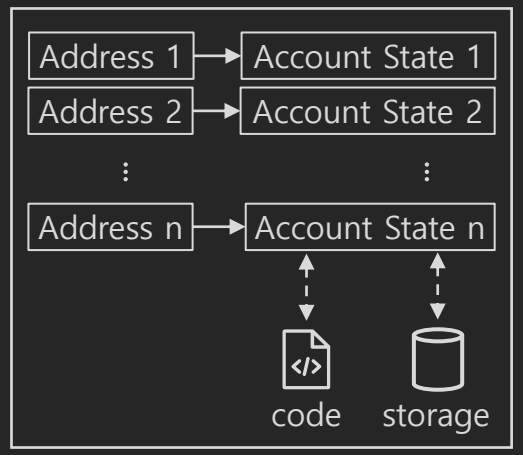
Address 1 → Account State 1
Address 2 → Account State 2
create ⋮
Address n → Account State n

code | storage

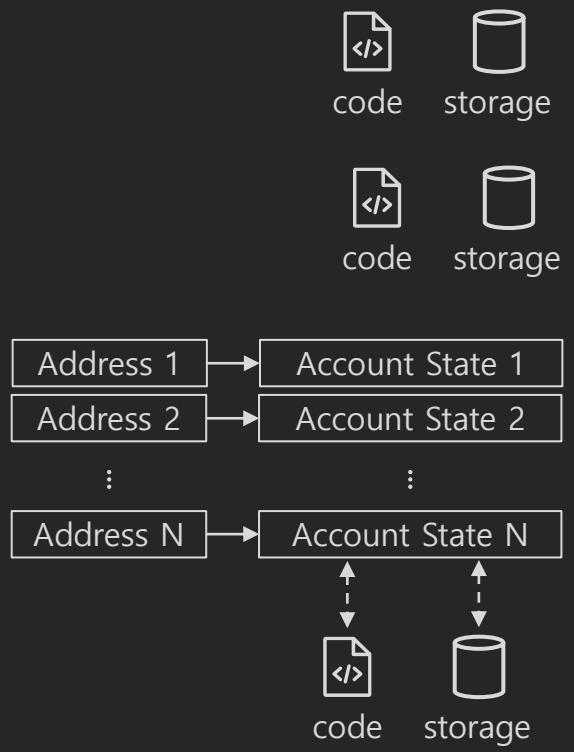
World State (t + 1)

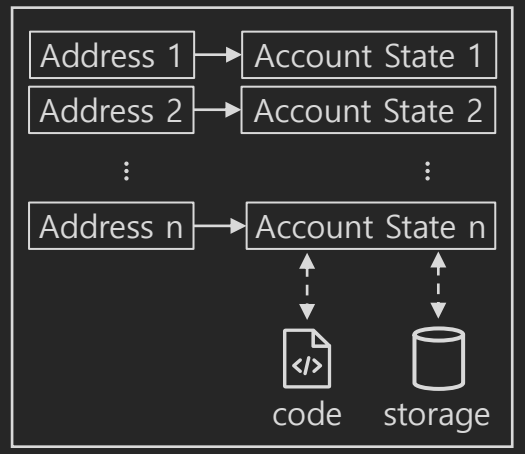


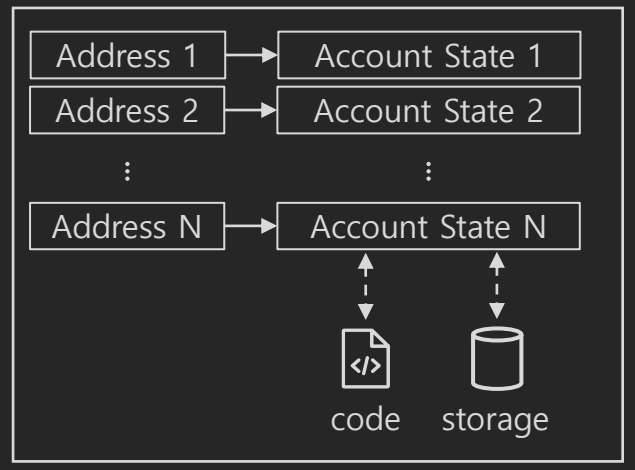




World State (t)



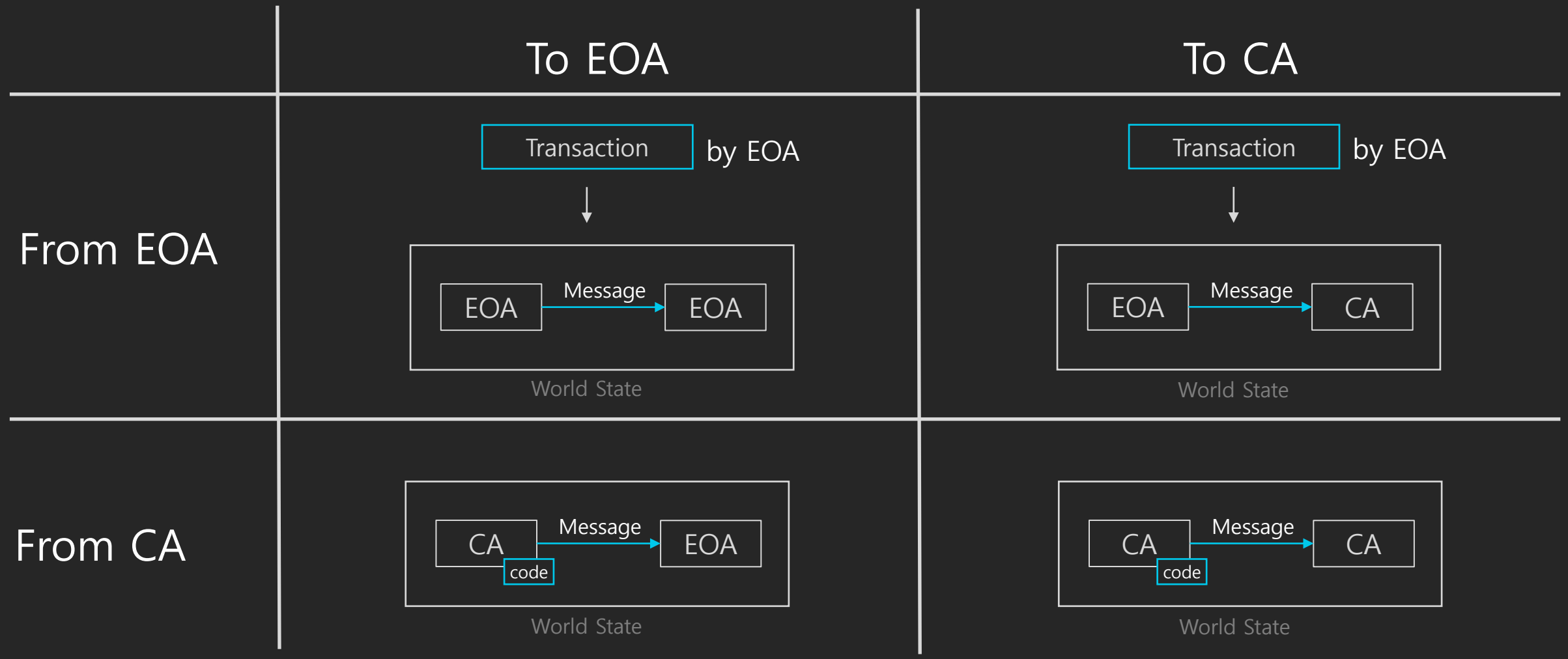




World State (t)

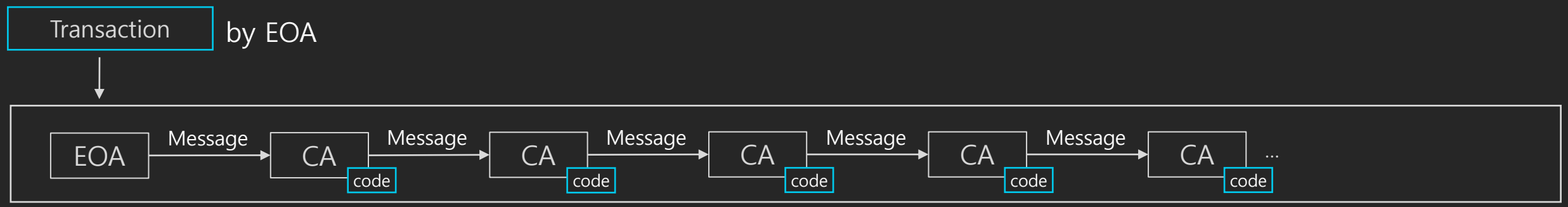



World State (t)

- 메시지는 Account 에서 Account로 전달
- Data(컨트랙트 실행 bytes)와 Value(Ether)로 구성
- CA도 code에서 메시지를 생성해서 다른 CA 또는 EOA 호출(call) 가능



# 메시지 (CA → CA)



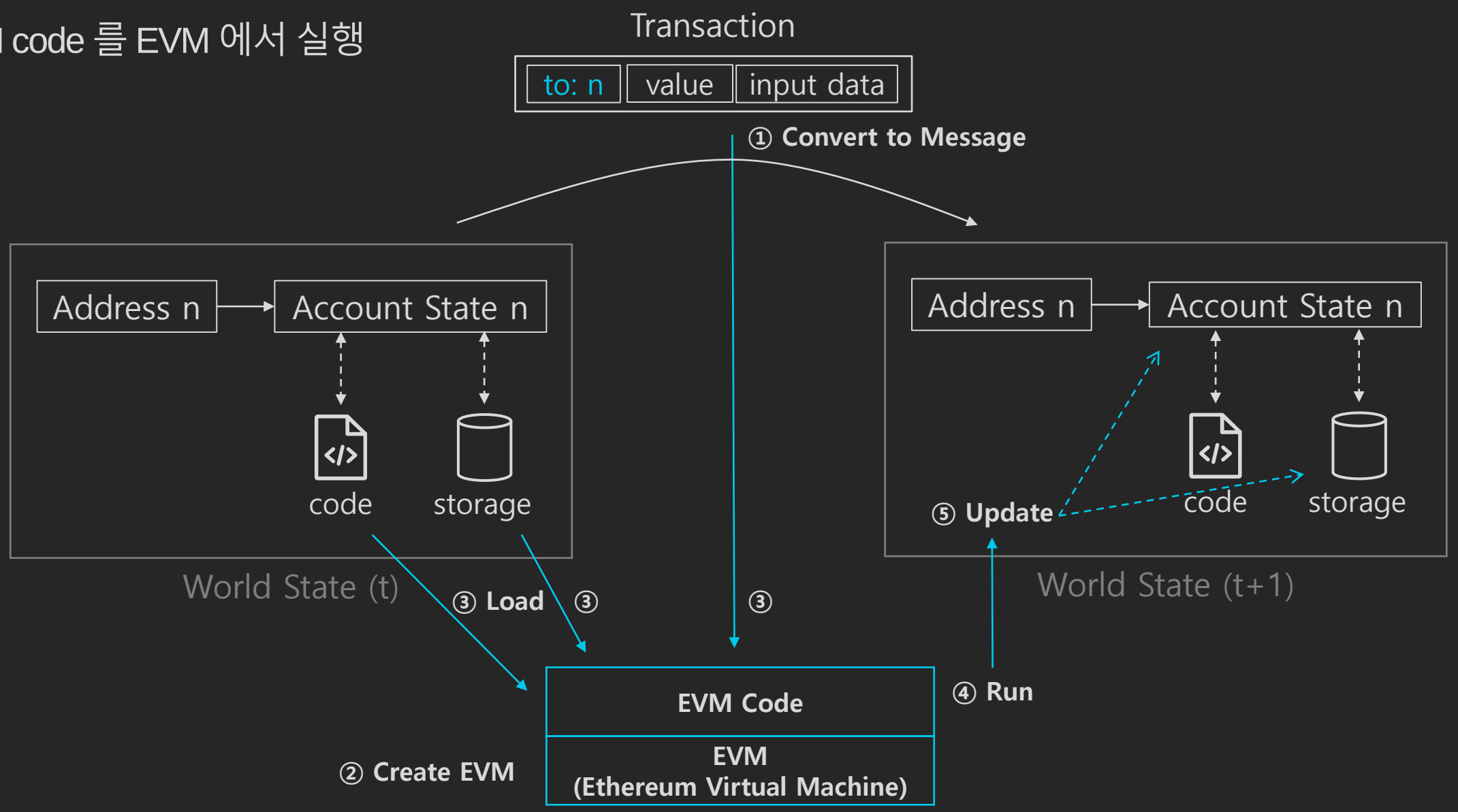

## Add(+) Contract , Subtract(-) Contract → ' a + b – c ' Contract ?

```
// Add.sol, 0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8
// SPDX-License-Identifier: MIT
pragma solidity 0.8.13;

contract Add {
    function add(uint a, uint b) public pure returns(uint) {
        return a + b;
    }
}
```

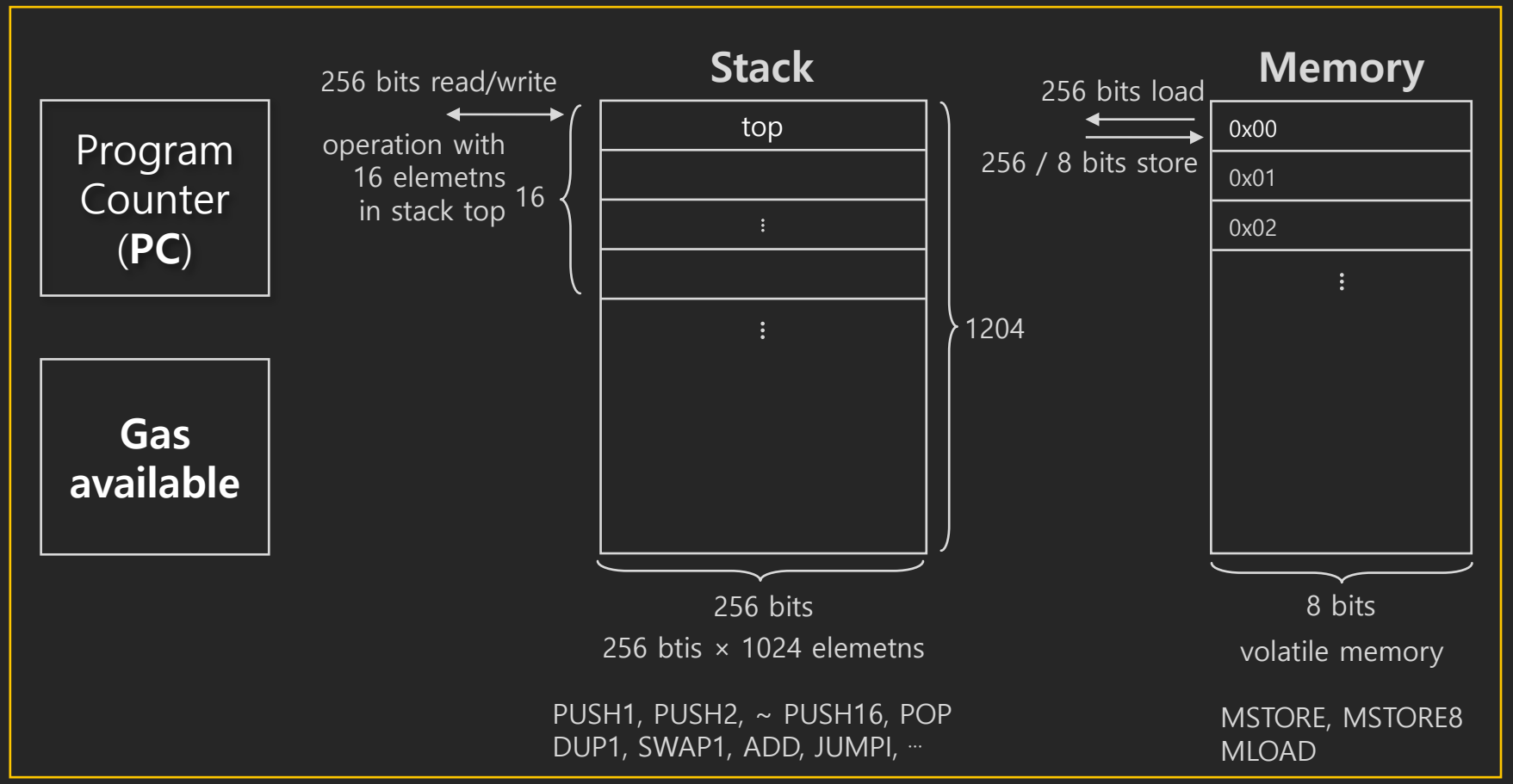
```
// Sub.sol, 0xd9145CCE52D386f254917e481eB44e9943F39138
// SPDX-License-Identifier: MIT
pragma solidity 0.8.13;

contract Sub {
    function sub(uint a, uint b) public pure returns(uint) {
        return a - b;
    }
}
```

```
// Calculator.sol
// SPDX-License-Identifier: MIT
pragma solidity 0.8.13;
import "./Add.sol";
import "./Sub.sol";

contract Calculator {
    // a + b - c
    function calc(uint a, uint b, uint c) public pure returns(uint) {
        Add addContract =
            Add(address(0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8));
        uint sum = addContract.add(a, b);

        Sub subContract =
            Sub(address(0xd9145CCE52D386f254917e481eB44e9943F39138));
        return subContract.sub(sum, c);
    }
}
```

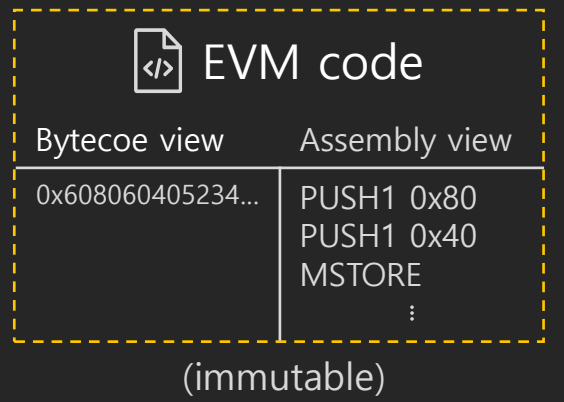
- Smart Contract Runtime Environment (like JVM)
- EVM code 를 EVM 에서 실행



# EVM architecture
## (Simple Stack Machine)




Machine state (volatile)




(immutable)

**(Account) storage**

| Key 1 | Value 1 |
|---|---|
| Key 2 | Value 2 |
| ⋮ | ⋮ |
| Key n | Value n |

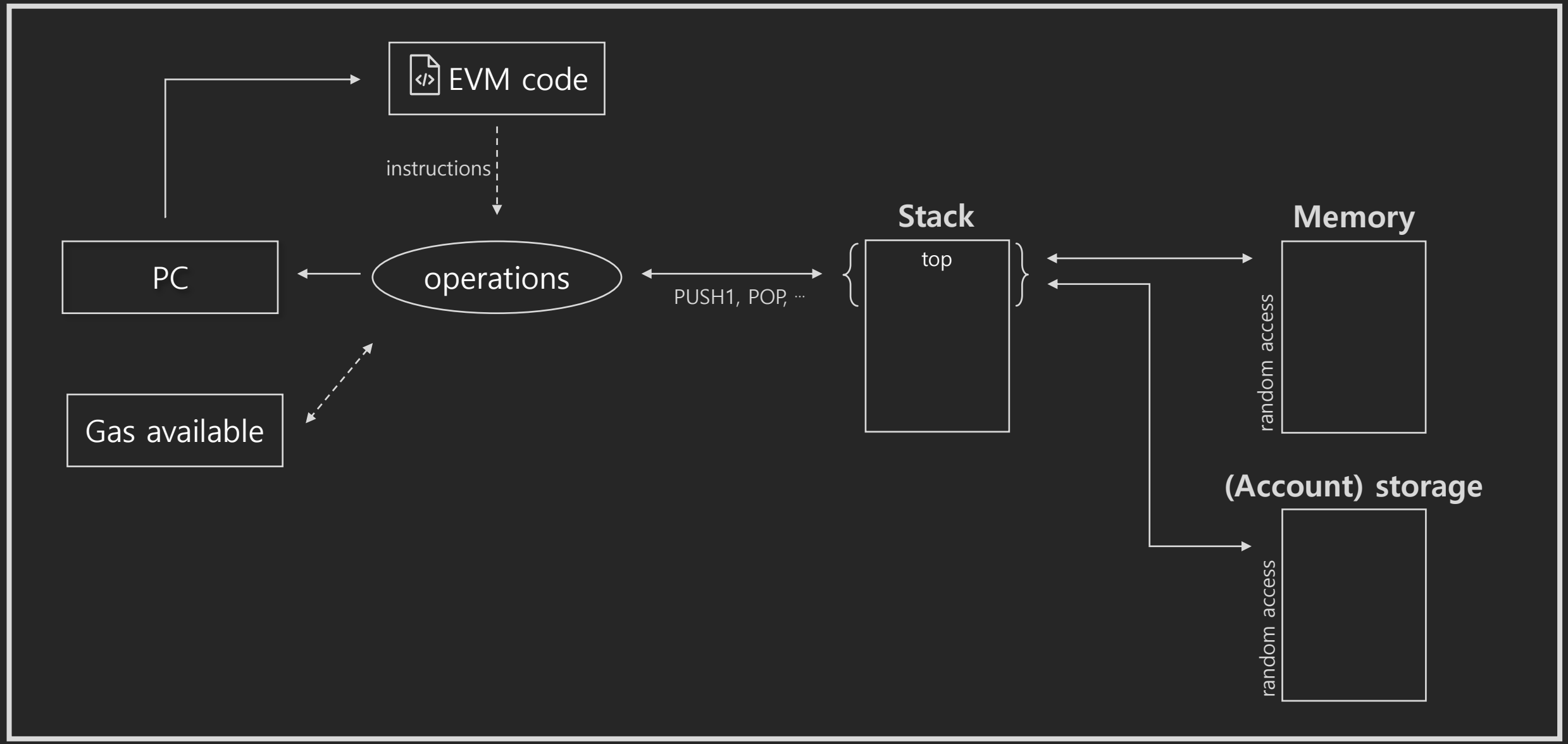
256 bits 256 bits

persistent memory

SLOAD, SSTORE

**World state (persistent)**

# EVM Execution Model



- 가스 (Gas)
  - 작업 수행에 필요한 계산 및 스토리지 자원 측정 단위
  - 연산자(OpCode) 수행, 데이터 조회, 저장 등에서 가스를 소비
  - 트랜잭션 : 최대 사용 가스 (Gas Limit)지정 및 가스 구매 필요 (Gas Fee = Gas Limit * Gas Price)
    ➞ 트랜잭션 수행간 가스 전부 소모 시 에러, 미사용 가스는 환불
  - 블록 크기 지정 : 블록 크기에 지정된 Gas 범위 내에서 트랜잭션을 블록에 포함 가능
  - 정지 문제 해결 및 DoS 공격 (Denial of service attack) 대응
  - 채굴자 보상 ➞ Gas Fee 를 채굴자에게 제공하여 동기부여
    ➞ EIP-1559(London) 이후 BaseFee를 제외한 Gas Fee만 보상으로 받음

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.13;
contract DosAttack {
    uint256 value;
    function infiniteLoop() public {
        for (uint256 i = 0; i < 9999999999999; i++) {
            for (uint256 j = 0; j < 9999999999999; j++) {
                value += i % 2 + j % 2;
            }
        }
    }
}
```

# EVM 실행 모델 with Gas

# 예제) '2 + 3' 결과를 key 0에 저장

Postfix => 2,3,+

## Transaction

GasPrice : 30 * $10^9$ wei (30 gwei)

GasLimit: 51144

## EVM code

```
0000 PUSH1 2
0002 PUSH1 3
0004 ADD
0005 PUSH1 1
0007 SSTORE
0008 RETURN
```

## Opcodes

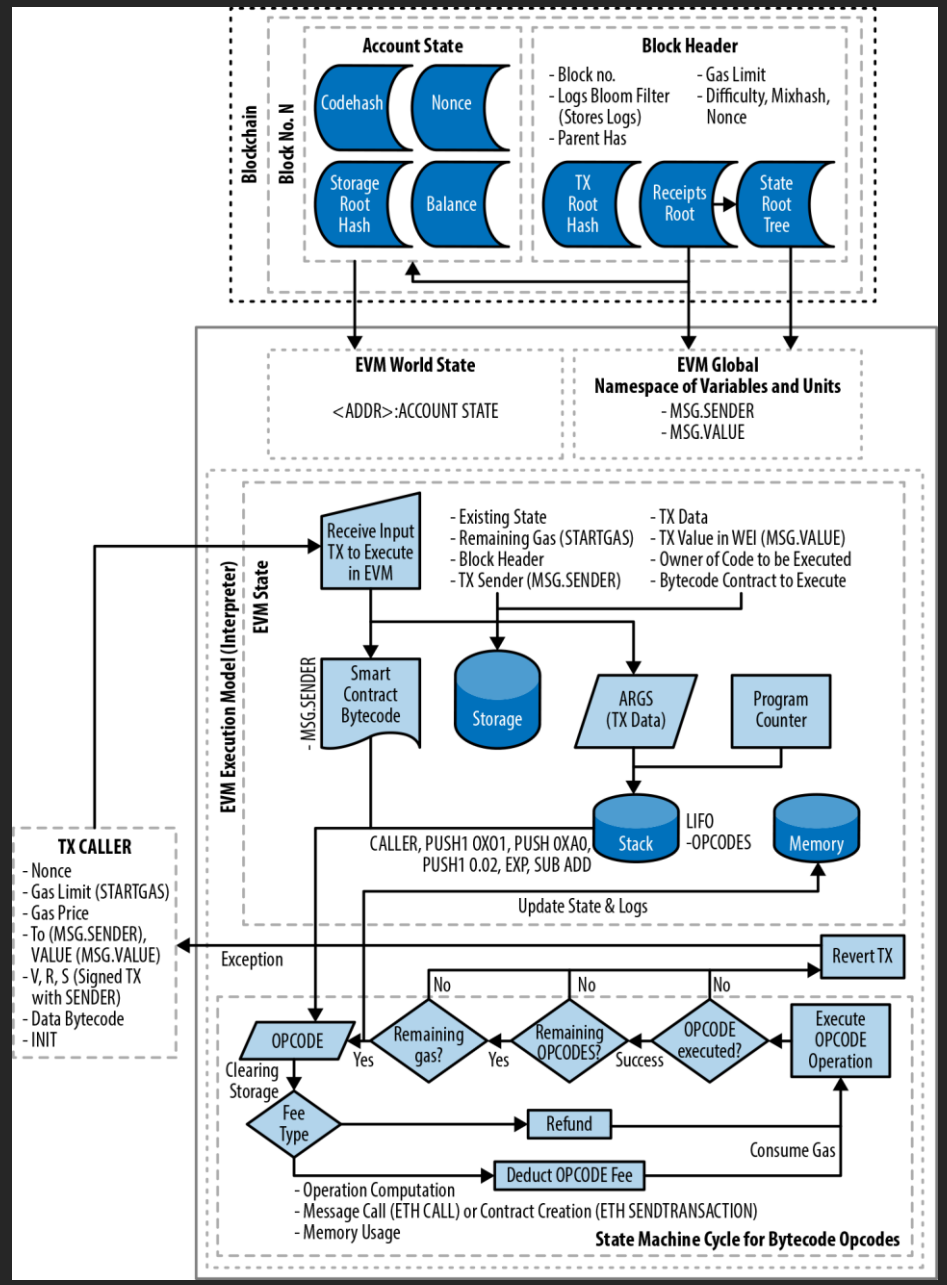
| Opcode | Name | Gas |
|---|---|---|
| 0x01 | ADD | 3 |
| 0x55 | SSTORE | 20000 |
| 0x60 | PUSH1 | 3 |
| 0xf3 | RETURN | 0 |

| PC | Opcde | Stack | GasUsed | GaAvailable | Storage |
|---|---|---|---|---|---|
| 0 | PUSH1 2 | ⋮ | 0 | *30000 | |
| 2 | PUSH1 3 | 2<br>⋮ | 3 | 29997 | |
| 4 | ADD | 3<br>2<br>⋮ | 6 | 29994 | |
| 5 | PUSH1 1 | 5<br>⋮ | 9 | 29991 | |
| 7 | SSTORE | 1<br>5<br>⋮ | 12 | 29988 | Key: 0x000...001, Value: 0x000...005 |
| 8 | RETURN | ⋮ | 22112 | 7876 | |

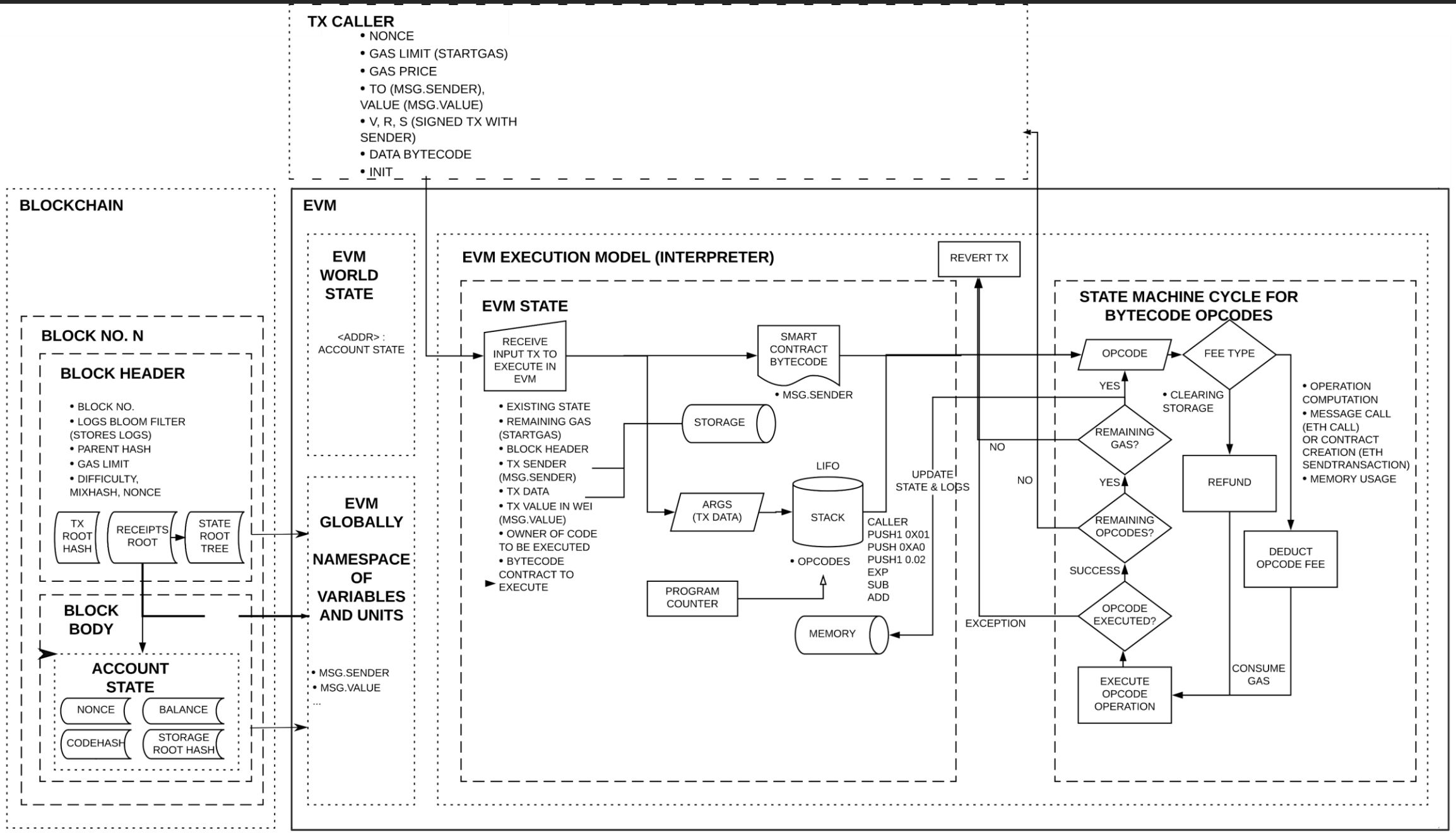
* 트랜잭션 기본 비용 : 21000, non-zero byte당 16 ➞ 51144 – 21000 - 144(16*9) = 30000

- GasUsed = 21000 + 16*9bytes + 22112 = 43256 gas
- Refund Gas = 51144 - 43256 = 7888 gas
- Fee = 43256 gas * 30 gwei = 1297680 gwei
- 1 ETH = 10**9 gwei = 2,500,000 원 ➞ 3,244.2 원 (ETH 전송 비용: 21000 gas ➞ 1575 원)

# EVM Architecture



# EVM Context


TX CALLER
• NONCE
• GAS LIMIT (STARTGAS)
• GAS PRICE
• TO (MSG.SENDER), VALUE (MSG.VALUE)
• V, R, S (SIGNED TX WITH SENDER)
• DATA BYTECODE
• INIT
BLOCKCHAIN
BLOCK NO. N
BLOCK HEADER
• BLOCK NO.
• LOGS BLOOM FILTER (STORES LOGS)
• PARENT HASH
• GAS LIMIT
• DIFFICULTY, MIXHASH, NONCE
TX ROOT HASH
RECEIPTS ROOT
STATE ROOT TREE
BLOCK BODY
ACCOUNT STATE
NONCE
BALANCE
CODEHASH
STORAGE ROOT HASH
EVM
EVM WORLD STATE
<ADDR> : ACCOUNT STATE
EVM GLOBALLY
NAMESPACE OF VARIABLES AND UNITS
• MSG.SENDER
• MSG.VALUE
...
EVM EXECUTION MODEL (INTERPRETER)
REVERT TX
EVM STATE
RECEIVE INPUT TX TO EXECUTE IN EVM
• EXISTING STATE
• REMAINING GAS (STARTGAS)
• BLOCK HEADER
• TX SENDER (MSG.SENDER)
• TX DATA
• TX VALUE IN WEI (MSG.VALUE)
• OWNER OF CODE TO BE EXECUTED
• BYTECODE CONTRACT TO EXECUTE
SMART CONTRACT BYTECODE
• MSG.SENDER
STORAGE
ARGS (TX DATA)
LIFO
STACK
• OPCODES
CALLER
PUSH1 0X01
PUSH 0XA0
PUSH1 0.02
EXP
SUB
ADD
PROGRAM COUNTER
MEMORY
UPDATE STATE & LOGS
STATE MACHINE CYCLE FOR BYTECODE OPCODES
OPCODE
FEE TYPE
YES
• CLEARING STORAGE
• OPERATION COMPUTATION
• MESSAGE CALL (ETH CALL) OR CONTRACT CREATION (ETH SENDTRANSACTION)
• MEMORY USAGE
REMAINING GAS?
NO
REFUND
YES
REMAINING OPCODES?
DEDUCT OPCODE FEE
SUCCESS
OPCODE EXECUTED?
EXCEPTION
EXECUTE OPCODE OPERATION
CONSUME GAS

# Transaction

4.2. **The Transaction.** A transaction (formally, $T$) is a single cryptographically-signed instruction constructed by an actor externally to the scope of Ethereum. The sender of a transaction cannot be a contract. While it is assumed that the ultimate external actor will be human in nature, software tools will be used in its construction and dissemination[1]. EIP-2718 by Zoltu [2020] introduced the notion of different transaction types. As of the Berlin version of the protocol, there are two transaction types: 0 (legacy) and 1 (EIP-2930 by Buterin and Swende [2020b]). Further, there are two subtypes of transactions: those which result in message calls and those which result in the creation of new accounts with associated code (known informally as 'contract creation'). All transaction types specify a number of common fields:

**type:** EIP-2718 transaction type; formally $T_{\mathrm{x}}$.

**nonce:** A scalar value equal to the number of transactions sent by the sender; formally $T_{\mathrm{n}}$.

**gasPrice:** A scalar value equal to the number of Wei to be paid per unit of *gas* for all computation costs incurred as a result of the execution of this transaction; formally $T_{\mathrm{p}}$.

**gasLimit:** A scalar value equal to the maximum amount of gas that should be used in executing this transaction. This is paid up-front, before any computation is done and may not be increased later; formally $T_{\mathrm{g}}$.

**to:** The 160-bit address of the message call's recipient or, for a contract creation transaction, $\varnothing$, used here to denote the only member of $\mathbb{B}_0$ ; formally $T_{\mathrm{t}}$.

**value:** A scalar value equal to 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; formally $T_{\mathrm{v}}$.

**r, s:** Values corresponding to the signature of the transaction and used to determine the sender of the transaction; formally $T_{\mathrm{r}}$ and $T_{\mathrm{s}}$. This is expanded in Appendix F.

EIP-2930 (type 1) transactions also have:

**accessList:** List of access entries to warm up; formally $T_{\mathbf{A}}$. Each access list entry $E$ is a tuple of an account address and a list of storage keys: $E \equiv (E_{\mathrm{a}}, E_{\mathbf{s}})$.

**chainId:** Chain ID; formally $T_{\mathrm{c}}$. Must be equal to the network chain ID $\beta$.

**yParity:** Signature Y parity; formally $T_{\mathrm{y}}$.

Legacy transactions do not have an **accessList** ($T_{\mathbf{A}} = ()$), while **chainId** and **yParity** for legacy transactions are combined into a single value:

**w:** A scalar value encoding Y parity and possibly chain ID; formally $T_{\mathrm{w}}$. $T_{\mathrm{w}} = 27 + T_{\mathrm{y}}$ or $T_{\mathrm{w}} = 2\beta + 35 + T_{\mathrm{y}}$ (see EIP-155 by Buterin [2016b]).

Additionally, a contract creation transaction (regardless whether legacy or EIP-2930) contains:

**init:** An unlimited size byte array specifying the EVM-code for the account initialisation procedure, formally $T_{\mathbf{i}}$.

**init** is an EVM-code fragment; it returns the **body**, a second fragment of code that executes each time the account receives a message call (either through a transaction or due to the internal execution of code). **init** is executed only once at account creation and gets discarded immediately thereafter.

In contrast, a message call transaction contains:

**data:** An unlimited size byte array specifying the input data of the message call, formally $T_{\mathbf{d}}$.

Appendix F specifies the function, $S$, which maps transactions to the sender, and happens through the ECDSA of the SECP-256k1 curve, using the hash of the transaction (excepting the latter three signature fields) as the datum to sign. For the present we simply assert that the sender of a given transaction $T$ can be represented with $S(T)$.

(16)
$$L_{\mathrm{T}}(T) \equiv \begin{cases} (T_{\mathrm{n}}, T_{\mathrm{p}}, T_{\mathrm{g}}, T_{\mathrm{t}}, T_{\mathrm{v}}, \mathbf{p}, T_{\mathrm{w}}, T_{\mathrm{r}}, T_{\mathrm{s}}) & \text{if } T_{\mathrm{x}} = \\ (T_{\mathrm{c}}, T_{\mathrm{n}}, T_{\mathrm{p}}, T_{\mathrm{g}}, T_{\mathrm{t}}, T_{\mathrm{v}}, \mathbf{p}, T_{\mathbf{A}}, T_{\mathrm{y}}, T_{\mathrm{r}}, T_{\mathrm{s}}) & \text{if } T_{\mathrm{x}} = \end{cases}$$

where

(17)
$$\mathbf{p} \equiv \begin{cases} T_{\mathbf{i}} & \text{if } T_{\mathrm{t}} = \varnothing \\ T_{\mathbf{d}} & \text{otherwise} \end{cases}$$

Here, we assume all components are interpreted by the RLP as integer values, with the exception of the access list $T_{\mathbf{A}}$ and the arbitrary length byte arrays $T_{\mathbf{i}}$ and $T_{\mathbf{d}}$.

(18)
$$\begin{array}{llllll} T_{\mathrm{x}} \in \{0,1\} & \wedge & T_{\mathrm{c}} = \beta & \wedge & T_{\mathrm{n}} \in \mathbb{N}_{256} & \wedge \\ T_{\mathrm{p}} \in \mathbb{N}_{256} & \wedge & T_{\mathrm{g}} \in \mathbb{N}_{256} & \wedge & T_{\mathrm{v}} \in \mathbb{N}_{256} & \wedge \\ T_{\mathrm{w}} \in \mathbb{N}_{256} & \wedge & T_{\mathrm{r}} \in \mathbb{N}_{256} & \wedge & T_{\mathrm{s}} \in \mathbb{N}_{256} & \wedge \\ T_{\mathrm{y}} \in \mathbb{N}_{1} & \wedge & T_{\mathbf{d}} \in \mathbb{B} & \wedge & T_{\mathbf{i}} \in \mathbb{B} & \end{array}$$

where

(19)
$$\mathbb{N}_{\mathrm{n}} = \{P : P \in \mathbb{N} \wedge P < 2^{n}\}$$

The address hash $T_{\mathbf{t}}$ is slightly different: it is either a 20-byte address hash or, in the case of being a contract-creation transaction (and thus formally equal to $\varnothing$), it is the RLP empty byte sequence and thus the member of $\mathbb{B}_0$:

(20)
$$T_{\mathbf{t}} \in \begin{cases} \mathbb{B}_{20} & \text{if } T_{\mathrm{t}} \neq \varnothing \\ \mathbb{B}_{0} & \text{otherwise} \end{cases}$$

# Legacy Transaction

## Frontier
(default: Error over RPC)

- nonce
- gasPrice
- gasLimit
- to
- value
- data
- v (27 + yParity)
- r
- s

## Replay attack Protection

- nonce
- gasPrice
- gasLimit
- to
- value
- data
- v (chainId*2 + 35 + yParity)
- r
- s

# Typed Transaction

## Type 1
(Optional Access List)

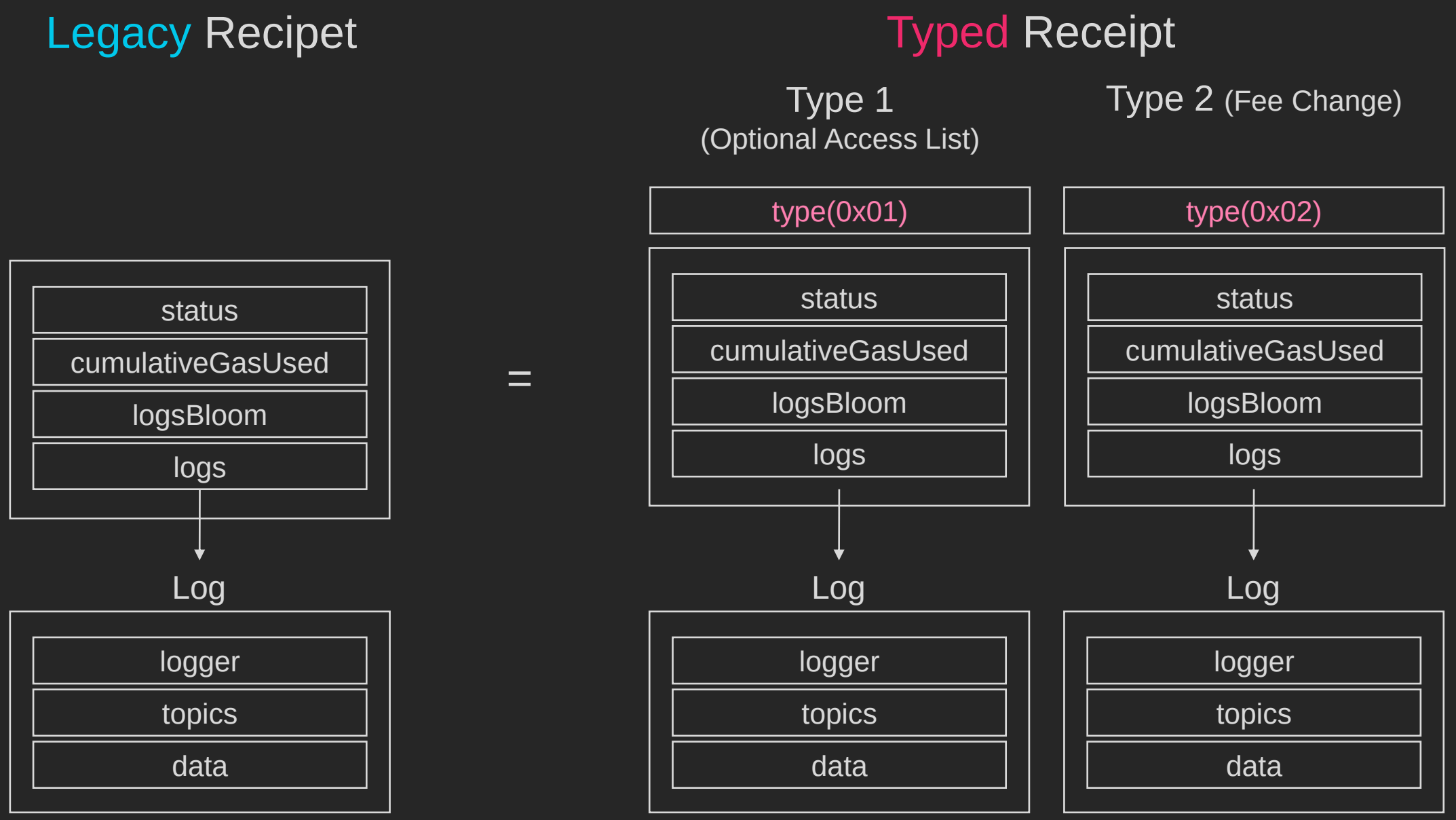
type(0x01)

- chainId
- nonce
- gasPrice
- gasLimit
- to
- value
- data
- accessList
- signatureYParity
- signatureR
- signatureS

## Type 2 (Fee Change)

type(0x02)

- chainId
- nonce
- maxPriorityFeePerGas
- maxFeePerGas
- gasLimit
- destination (= to)
- amount (= value)
- data
- accessList
- signatureYParity
- signatureR
- signatureS


Legacy Recipet
Typed Receipt
Type 1
(Optional Access List)
Type 2 (Fee Change)
type(0x01)
type(0x02)
status
cumulativeGasUsed
logsBloom
logs
=
status
cumulativeGasUsed
logsBloom
logs
status
cumulativeGasUsed
logsBloom
logs
Log
logger
topics
data
Log
logger
topics
data
Log
logger
topics
data

- 트랜잭션 유형 (EIP-2718: Typed Transaction Envelope, Berlin, #12,244,000, 2021-04-15)
  - 트랜잭션 유형 추가 시, 하위 호환성 보장 어려움 해결
  - RLP 외 Serialize 지원 (예) SSZ, LEB128 등)
- Transaction , Receipt
  - Transaction : Legacy Transaction 또는 Typed Transaction
  - Receipt : Legacy Receipt 또는 Typed Receipt
- Legacy와 Typed 구분 (첫번째 바이트)
  - [0x00, 0x7f] Typed
  - [0xc0, 0xfe] Legacy , [0xff] is reserved for future use.
- Typed Transaction : TransactionType || TransactionPayload → ex) Type2 : 0x02f9...
  - TransactionType : unsigned 8bit 숫자 [0x00, 0x7f]
  - TransactionPayload : TransactionType 별 EIP에서 정의한 byte array
- Typed Receipt : TransactionType || ReceiptPayload
  - TransactionType: unsigned 8bit 숫자 [0x00, 0x7f]
  - ReceiptPayload : TransactionType 별 EIP에서 정의한 byte array

RLP Serialize

| Prefix | 유형 |
|---|---|
| [0x00, 0x7f] | 문자열 |
| [0x80, 0xb7] | 문자열 |
| [0xb8, 0xbf] | 문자열 |
| [0xc0, 0xf7] | 배열 |
| [0xf8, 0xff] | 배열 |

- Legacy (Frontier, #0, 2015-07-30) , (EIP-155 : Simple replay attack protection, Spurious Dragon, #2,675,000, 2016-11-22)
  - Transaction
    RLP([nonce, gasPrice, gasLimit, to, value, data, v, r, s])
  - Receipt
    RLP([status, cumulativeGasUsed, logsBloom, logs])

- Type 1 (EIP-2930 : Optional access lists, Berlin, #12,244,000, 2021-04-15)
  - Transaction = 0x01 || Type 1 Transaction Payload
    0x01 || RLP([chainId, nonce, gasPrice, gasLimit, to, value, data, accessList, signatureYParity, signatureR, signatureS])
  - Receipt = 0x01 || Type 1 Receipt Payload
    0x01 || RLP([status, cumulativeGasUsed, logsBloom, logs])

| | |
|---|---|
| Typed Transaction | : TransactionType \|\| TransactionPayload |
| Typed Receipt | : TransactionType \|\| ReceiptPayload |

- Type 2 (EIP-1599 : Fee market change for ETH 1.0 chain (London, #12,965,000, 2021-08-05)
  - Transaction = 0x02 || Type2 Transaction Payload
    0x02 || RLP([chainId, nonce, maxPriorityFeePerGas, maxFeePerGas, gasLimit, destination, amount, data, accessList, signatureYParity, signatureR, signatureS])
  - Receipt = 0x02 || Type2 Receipt Payload
    0x02 || RLP([status, cumulativeGasUsed, logsBloom, logs])

| 항목 | 타입 | 설명 |
|---|---|---|
| **nonce** | uint256 | 발신자(sender)가 전송한 트랜잭션 수 (트랜잭션 순서 보장, 이중사용 방지 등) |
| **gasPrice** | uint256 | gas 당 지불 가격 (단위: wei, 1ETH = 1 * $10^{18}$wei, 1 gWei = 1 * $10^{9}$wei) |
| **gasLimit** | uint256 | 트랜잭션 수행에 사용할 최대 gas |
| **to** | address | 메시지 콜 수신자(recipient), 0x00: 컨트랙트 생성 |
| **value** | uint256 | 전송할 wei (1 ether = 1 * $10^{18}$wei) |
| **data** | bytes | to != 0x00 : 메시지 콜 input data, to == 0x00: EVM bytecode(init code)<br>컨트랙트 호출: ABI Encoding 결과 입력 |
| **v** | uint256 | 27 + yParity [Frontier] or (chainId)*2 + 35 + yParity [EIP-155]<br>• chainId : 블록체인 chainId (EIP-155: Simple replay attack protection), chainlist.org<br>• yParity : ECDSA 서명 결과, 임시 public key y 좌표 값 parity (0: 짝수, 1: 홀수) |
| **r** | uint256 | ECDSA 서명 결과 r, 임시 공개키 x 좌표 |
| **s** | uint256 | ECDSA 서명 결과 s |

## **논스(Nonce):** Account 가 전송한 트랜잭션 수

- 0 부터 1씩 증가
- 순서대로 트랜잭션 적용 + 중복 불가, 누락 불가 ➞ 트랜잭션 순서 보장 ➞ 이중 사용(Double Spending) 방지
  예) 유효: 0 ➞ 1 ➞ 2 , 중복 불가: 0 ➞ ~~1~~ ➞ 1 , 누락 불가 : 0 ➞ ~~1~~ ➞ 3
- 유효한 트랜잭션 Nonce : Account Nonce (include Pending) 이상
- Pool에 Pending 상태의 동일 Nonce 트랜잭션 존재하는 경우 ➞ Gas Price 체크
  - Compare Gas Price = Old Gas Price + Price Bump (default: 10%))
  - New Gas Price >= Compare Price ➞ 교체(Replace)
  - Compare Price > New Gas Price ➞ 에러(ERROR)

- 트랜잭션 교체 (Price Bump Percentage: 10%)
  - Old Gas Price : 30 Gwei ➞ Compare Gas Price = Old Gas Price × 1.1 = 33 Gwei
  - New Gas Price : 33 Gwei ➞ 교체
  - New Gas Price : 31 Gwei ➞ 에러
- 수정 또는 가속화
  - 동일 Nonce, New Gas Price >= Old Gas Price × (100% + Price Bump Percentage)
- 취소 (Cancel) : 0 ETH를 본인에게 전송하는 수정 트랜잭션 제출
  - 기본 트랜잭션 Fee 소모 ➞ 21000 gas × New Gas Price
- 현재 nonce 조회
  - JSON-RPC : eth_getTransactionCount(address, quantity)
  - web3js : web3.eth.getTransactionCount(address [, defaultBlock] [, callback ])

# Tx Nonce >= Current State Nonce (include Pending)

Tx 1

① nonce: 0

② ERROR

Tx Nonce(0) < Current State Nonce(1)

Tx 2

③ nonce: 1

④ Success

Tx Nonce(1) == Current State Nonce(1)

Tx 3

⑤ nonce: 3

⑥ Success

Tx Nonce(3) > Current State Nonce(2)

## Ethereum Client

Transaction Pool

④ Tx2 (nonce: 1)

⑥ Tx3 (nonce: 3)

⑦ Tx3 은 제외 (nonce: 2 Tx 필요)

Current Account State

nonce : 1

④

nonce : 2

Block

Tx2 (nonce: 1)

- Legacy Transaction (Frontier)
  - Hash for sign : keccak-256 ( RLP [ nonce, gasPrice, gasLimit, to, value, data, **0x**, 0x, 0x ] )
  - v : signatureYParity (0:짝수 or 1:홀수) **+ 27**
  - Signed Tx : RLP [ nonce, gasPrice, gasLimit, to, value, data, v, r, s ]
  - Replay attack 가능 (트랜잭션 정보로 Chain 구분 불가능)

Tx (address: A, nonce: 2) ➞ Ropsten [ Testnet ] (address: A, nonce : 2)

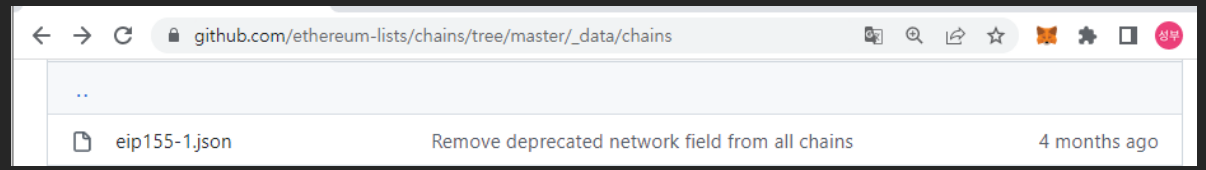
⬇ 다른 chain에서 replay attack 가능

Tx (address: A, nonce: 2) ➞ Ethereum [ Mainnet ] (address: A, nonce: 2)

- EIP-155 : Simple replay attack protection (Spurious Dragon, 2016-11-22)
  - Hash for sign : keccak-256 ( RLP [ nonce, gasPrice, gasLimit, to, value, data, **chainId**, 0x, 0x ] )
  - v : signatureYParity (0 or 1) **+ (chainId * 2 ) + 35**
  - Signed Tx : RLP [ nonce, gasPrice, gasLimit, to, value, data, v, r, s ]

# 트랜잭션 [ Chain List ]

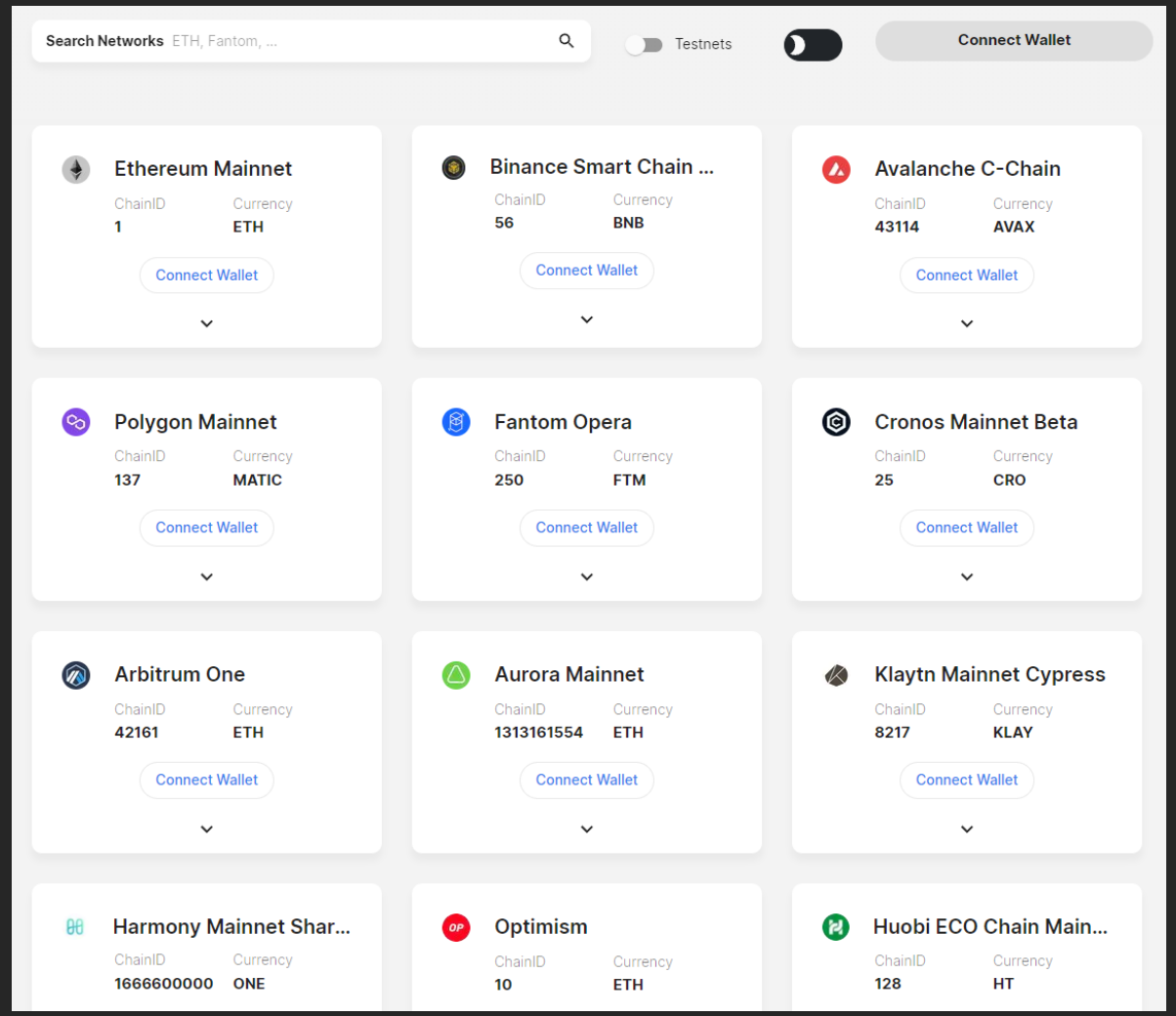
https://github.com/ethereum-lists/chains





```
{
  "name": "Ethereum Mainnet",
  "chain": "ETH",
  "icon": "ethereum",
  "rpc": [
    "https://mainnet.infura.io/v3/${INFURA_API_KEY}",
    "wss://mainnet.infura.io/ws/v3/${INFURA_API_KEY}",
    "https://api.mycryptoapi.com/eth",
    "https://cloudflare-eth.com"
  ],
  "faucets": [],
  "nativeCurrency": {
    "name": "Ether",
    "symbol": "ETH",
    "decimals": 18
  },
  "infoURL": "https://ethereum.org",
  "shortName": "eth",
  "chainId": 1,
  "networkId": 1,
  "slip44": 60,
  "ens": {
    "registry":"0x00000000000C2E074eC69A0dFb2997BA6C7d2e1e"
  },
  "explorers": [{
    "name": "etherscan",
    "url": "https://etherscan.io",
    "standard": "EIP3091"
  }]
}
```

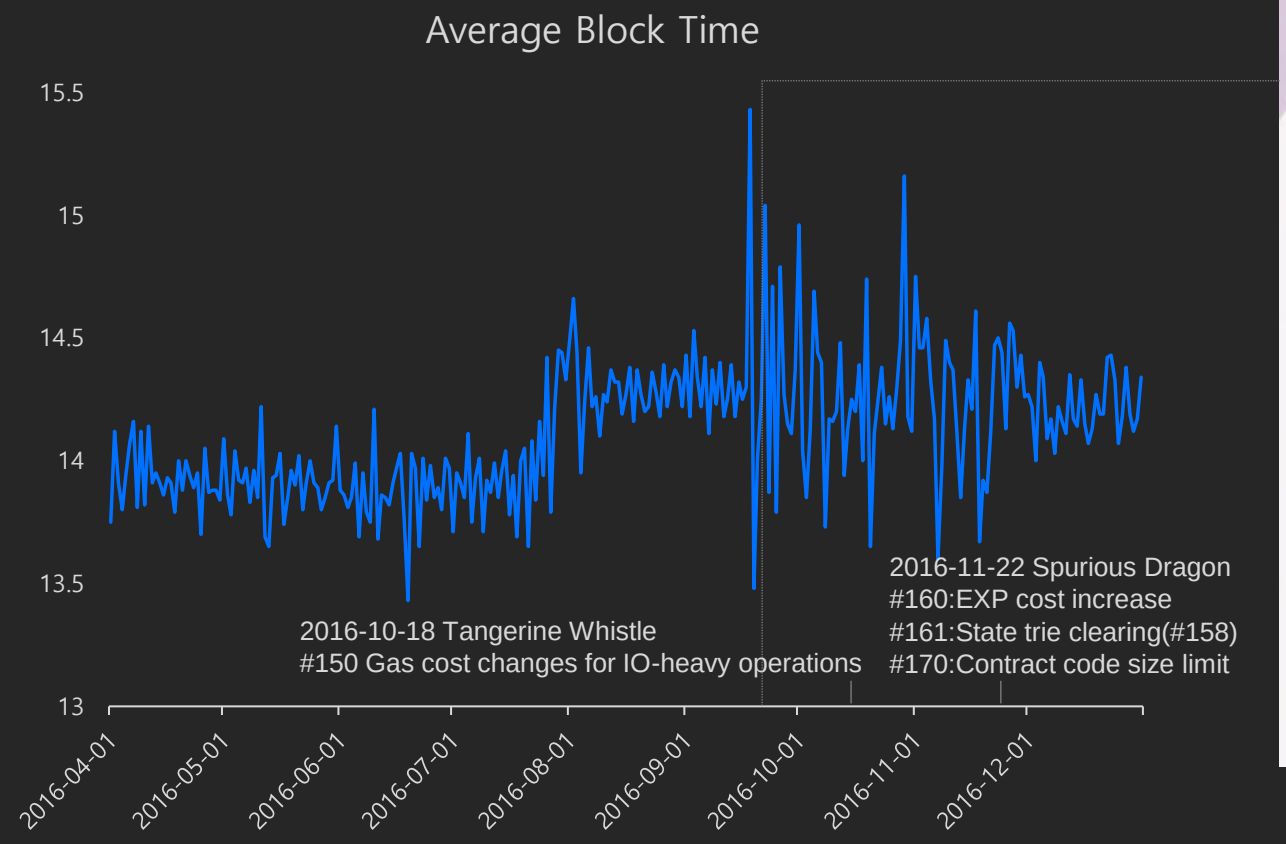

https://chainlist.org/



트랜잭션 Type 1 (EIP-2930: Optinal Access List, Berlin, #12,244,000, 2021-04-15)

| 항목 | 타입 | 설명 |
|---|---|---|
| **chainId** | uint256 | chainId (EIP-155: Simple replay attack protection, chainlist.org) |
| **nonce** | uint256 | 발신자(sender)가 전송한 트랜잭션 수 (트랜잭션 순서 보장, 이중지불 방지 등) |
| **gasPrice** | uint256 | gas 당 지불 가격 (단위: wei, 1ETH = 1 * $10^{18}$wei, 1 gwei = 1 * $10^{9}$wei) |
| **gasLimit** | uint256 | 트랜잭션 수행에 사용할 최대 gas |
| **to** | address | 메시지 콜 수신자(recipient), 0x0: 컨트랙트 생성 |
| **value** | uint256 | 전송할 wei (1 ether = 1 * $10^{18}$wei) |
| **data** | bytes | to != 0x00 : 메시지 콜 input data, to == 0x00: EVM bytecode |
| **accessList** | Access[] | Warn 상태 접근, Access = [ address (20bytes), [ StorageKey (32bytes) ] ] |
| **signatureYParity** | uint256 | ECDSA 서명 결과, 임시 public key y 좌표 값 parity (0: 짝수, 1 : 홀수) |
| **signatureR** | uint256 | ECDSA 서명 결과, r |
| **signatureS** | uint256 | ECDSA 서명 결과, s |

- 2016년 9월 이더리움 네트워크에 지속적인 DOS
- Disk I/O OpCode Gas Price가 저렴한 점을 이용
  - 무작위 계정(Account) 생성 및 접근 공격
  - 무작위 거래(Transaction) 수행 공격
- 네트워크 불안정 → 트랜잭션 속도 감소 → 채굴 제약




# The Ethereum network is currently undergoing a DoS attack

Posted by Jeffrey Wilcke on September 22, 2016

Research & Development

URGENT ALL MINERS: The network is under attack. The attack is a computational DDoS, ie. miners and nodes need to spend a very long time processing some blocks. This is due to the EXTCODESIZE opcode, which has a fairly low gasprice but which requires nodes to read state information from disk; the attack transactions are calling this opcode roughly 50,000 times per block. The consequence of this is that the network is greatly slowing down, but there is NO consensus failure or memory overload. We have currently identified several routes for a more sustainable medium-term fix and have developers working on implementation.

https://blog.ethereum.org/2016/09/22/ethereum-network-currently-undergoing-dos-attack/

- 2016-10-16 (Tangerin Whistle)
  - EIP-150 : Gas cost changes for IO-heavy operations
- 2016-11-22 (Spurious Dragon)
  - EIP-160 : EXP cost increase
  - EIP-161 : State trie clearing
- 2019-02-28 (Constantinople/St. Petersburg)
  - EIP-1283 : Net gas metering for SSTORE without dirty maps
- 2019-12-08 (Istanbul)
  - EIP-1108 : Reduce alt_bn128 precompile gas costs
  - EIP-1884: Repricing for trie-size-dependent opcodes
  - EIP-2028: Calldata gas cost reduction
  - EIP-2200: Rebalance net-metered SSTORE gas cost with consideration of SLOAD gas cost change
- 2021-04-15 (Berlin)
  - EIP-2565: ModExp Gas Cost
  - EIP-2929: Gas cost increases for state access opcodes
  - EIP-2930: Optional access lists

# EIP 2929 : Gas cost increases for state access opcodes (Berlin, 2021-04-15)

- 처음 사용하는 SLOAD, *CALL, BALANCE, EXT*,SELFDESTRUCT 의 Gas 증가
  - 이미 접근한 주소 및 스토리지 슬롯은 비용을 적게 소모
  - precompiled된 코드는 비용을 적게 소모
  - *CALL : CALL, CALLCODE, DELEGATECALL, STATICCALL
  - EXT* : EXTCODESIZE, EXTCODECOPY, EXTCODEHASH

| OpCode | Before Berlin | After Berlin | |
|---|---|---|---|
| | | Cold(Not accessed) | Warm(Accessed) |
| SLOAD | 800 | 2100 | 100 |
| SSTORE 0 ➞ 1 | 20000 | 22100 | 20000 |
| SSTORE 1 ➞ 2 | 5000 | 5000 | 2900 |
| SLOAD + SSTORE* | 5800 | 5000 | 3000 |
| SSOTRE* + SLOAD | 5800 | 5100 | 3000 |
| SSTORE an already written slot | 800 | 100 | |
| SSTORE 0 ➞ 1 : 슬롯 값이 0에서 1(또는 0이 아닌 값)으로 변경<br>SSTORE 1 ➞ 2 : 슬롯 값이 1에서 2(또는 1이 아닌 값)으로 변경 - 0으로 변경 시 (환불)<br>SSTORE* = SSTORE 1 ➞ 2 | | | |

## Access List

- AccessList = [ Access ]
- Access = [ adress(20bytes) , [ StorageKey ] ]
- StroageKey = 32 bytes
- Access List 지정 시, Warn 상태로 트랜잭션 수행
  - Address 당 2400 Gas (Cold : 2600 Gas, 200 절약)
  - Storage Key 당 1900 Gas (Cold : 2100 Gas, 100 절약)

```
// Access List
[
  [  // Access
    "0xde0b295669a9fd93d5f28d9ec85e40f4cb697bae", // address (20 bytes)
    [  // [ Storage Key (32 bytes) ]
      "0x0000000000000000000000000000000000000000000000000000000000000003",
      "0x0000000000000000000000000000000000000000000000000000000000000007"
    ]
  ],
  [
    "0xbb9bc244d798123fde783fcc1c72d3bb8c189413",
    []
  ]
]
```

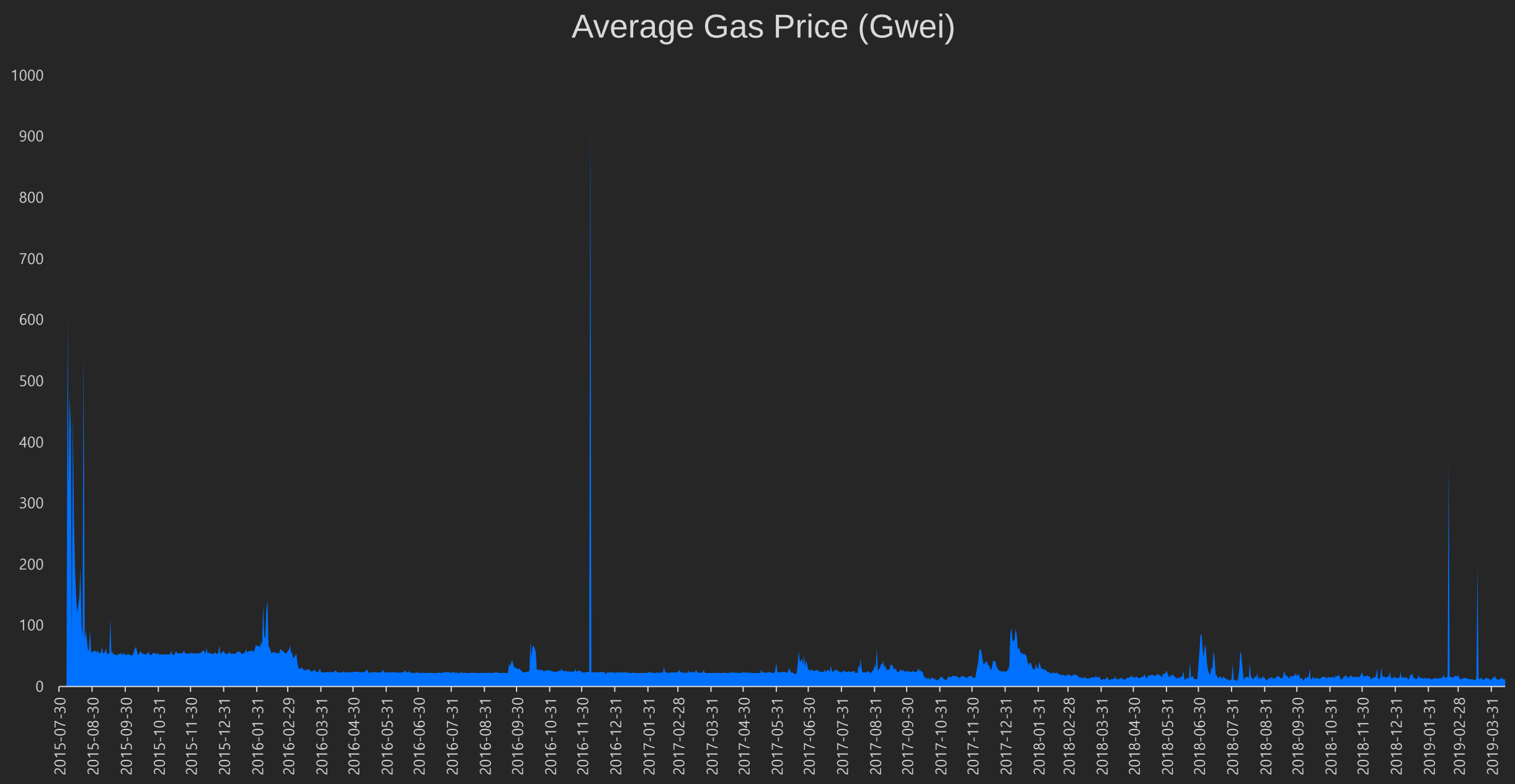
- 예상치 못한 Gas 소모로 트랜잭션 실패를 막기 위함, AccessList Gas는 트랜잭션 수행 전 계산
- eth_createAccessList : 트랜잭션의 AccessList 및 사용 가스량 반환

| OpCode | Before Berlin | After Berlin | |
|---|---|---|---|
| | | Cold(Not accessed) | Warm(Accessed) |
| SLOAD | 800 | 2100 | 100 |
| SSTORE 0 → 1 | 20000 | 22100 | 20000 |
| SSTORE 1 → 2 | 5000 | 5000 | 2900 |
| SLOAD + SSTORE* | 5800 | 5000 | 3000 |
| SSOTRE* + SLOAD | 5800 | 5100 | 3000 |
| SSTORE an already written slot | 800 | 100 | |

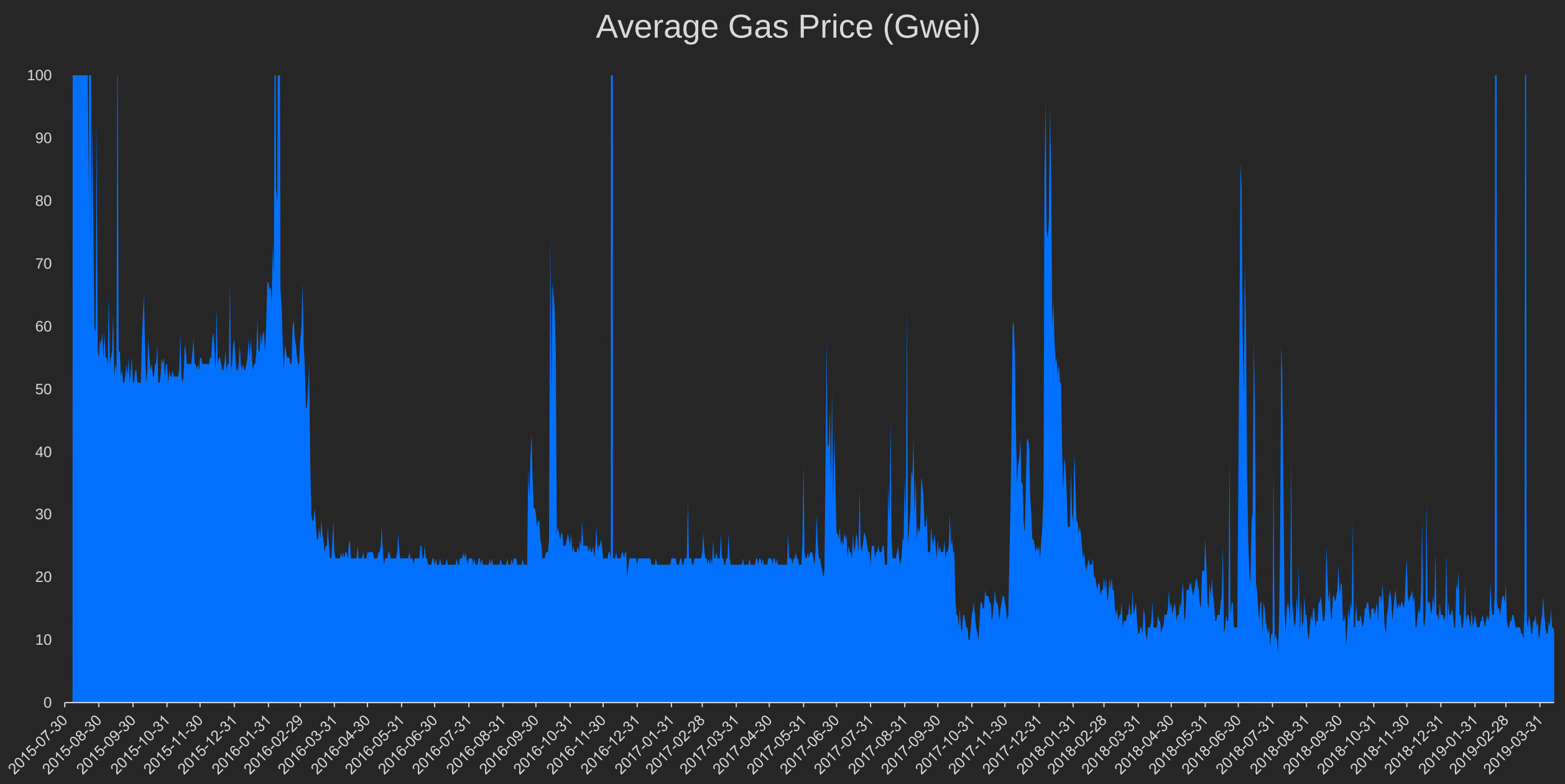
트랜잭션 Type 2 (EIP-1559: Fee market change for ETH 1.0 chain, London, #12,965,000, 2021-08-05)

| 항목 | 타입 | 설명 |
|---|---|---|
| **chainId** | uint256 | chainId (EIP-155: Simple replay attack protection, chainlist.org) |
| **nonce** | uint256 | 발신자(sender)가 전송한 트랜잭션 수 (트랜잭션 순서 보장, 이중지불 방지 등) |
| **maxPriorityFeePerGas** | uint256 | gas당 최대 priority fee |
| **maxFeePerGas** | uint256 | gas당 최대 fee (maxFeePerGas >= maxPriorityFeePerGas) |
| **gasLimit** | uint256 | 트랜잭션 수행에 사용할 최대 gas |
| **destination** | address | 메시지 콜 수신자(recipient), 0x0: 컨트랙트 생성 (= legacy.to) |
| **amount** | uint256 | 전송할 wei (1 ether = 1 * $10^{18}$wei) (= legacy.value) |
| **data** | bytes | to != 0x00 : 메시지 콜 input data, to == 0x00: EVM bytecode |
| **accessList** | Access[] | |
| **signatureYParity** | uint256 | ECDSA 서명 결과, 임시 public key y 좌표 값 parity (0: 짝수, 1 : 홀수) |
| **signatureR** | uint256 | ECDSA 서명 결과, r |
| **signatureS** | uint256 | ECDSA 서명 결과, s |

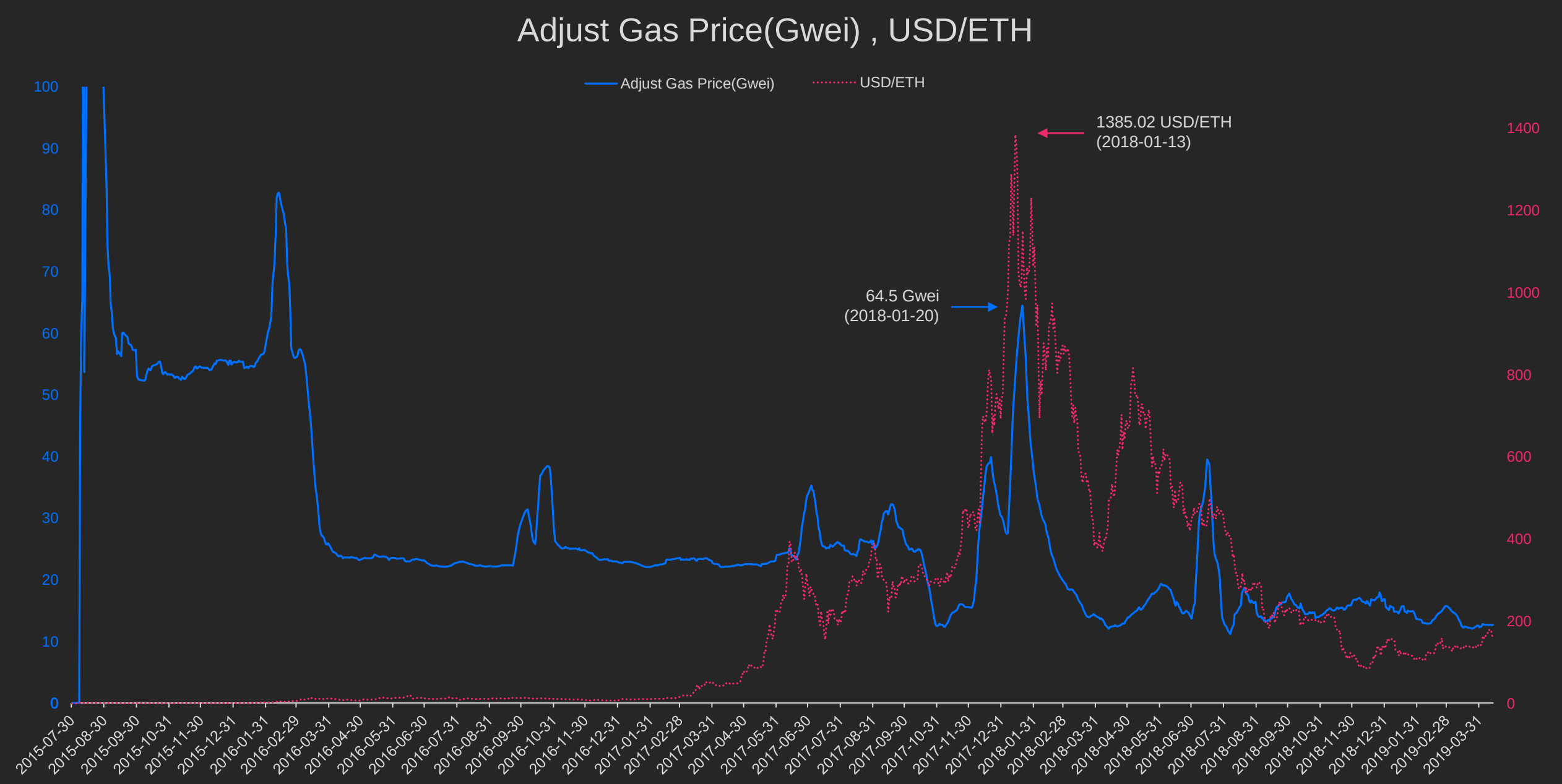
EIP-1559 [ Average Gas Price ( Genesis ~ 2019-04-13 ) ]



EIP-1559 [ Average Gas Price ( Genesis ~ 2019-04-13 ) ]

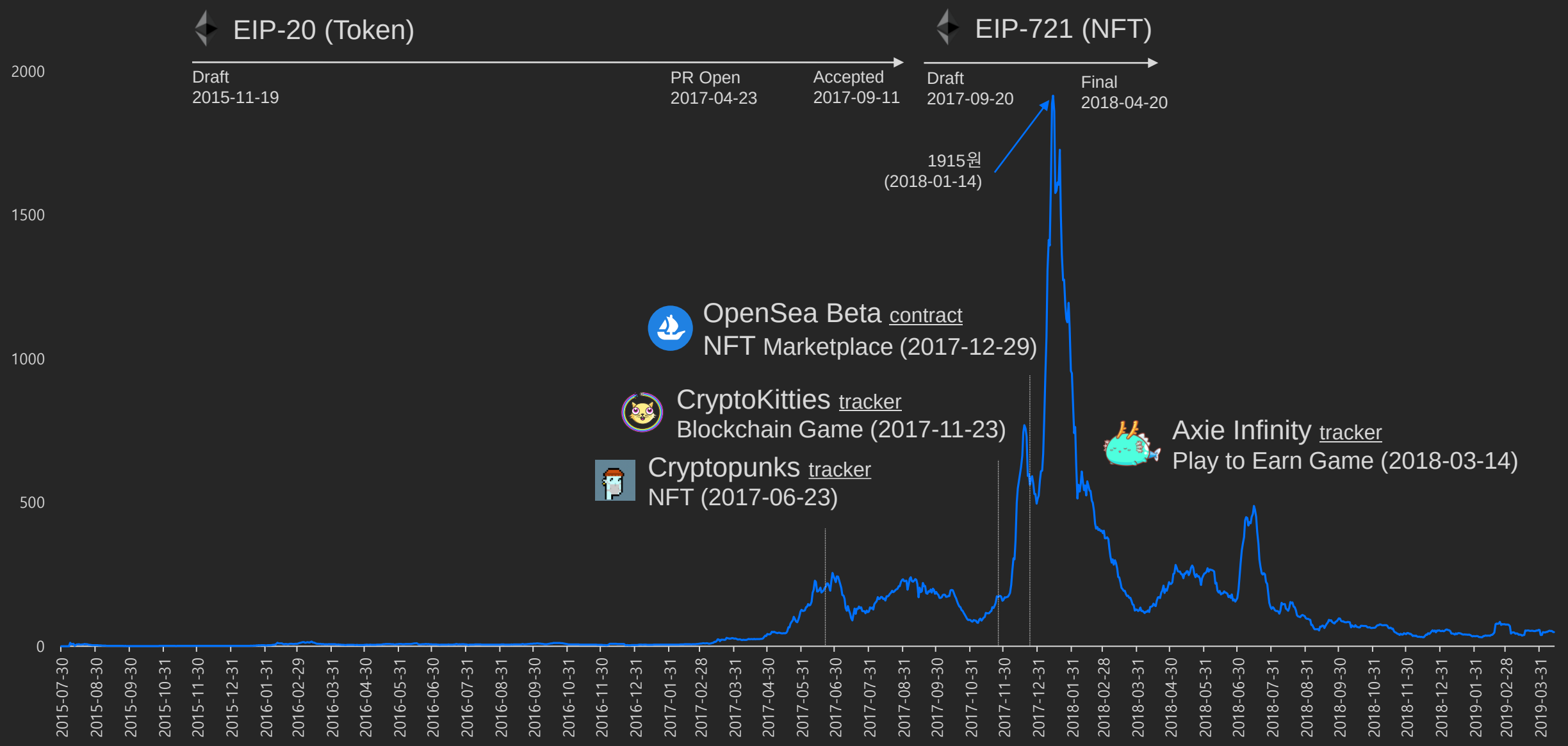


EIP-1559 [ Adjust Gas Price , USD/ETH ( Genesis ~ 2019-04-13 ) ]



EIP-1559 [ ETH Transfer Cost (원) (KTW/USD = 1100) ( Genesis ~ 2019-04-13 )]

# ETH Transfer Cost (원, 21000 gas) (원/달러 = 1100)



**EIP-1559** : Fee market change for ETH 1.0 chain (London, 2021-08-05)

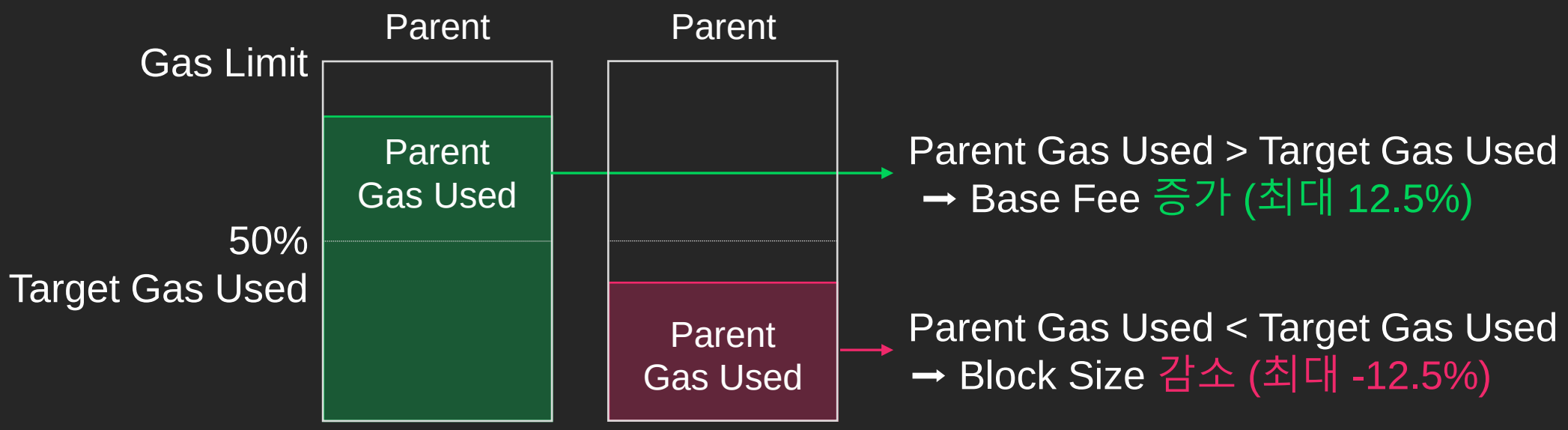
- 가스비 급등 억제 - 블록 크기 증가, Base Fee 도입
- 블록 크기
  - 최대 블록 크기 2배로 증가 (15,000,000 Gas ➞ 30,000,000 Gas)
- 기본 수수료 (Base Fee) : 부모 블록 사용율에 따라 자동으로 조정
  - 부모 블록 기본 수수료 ±12.5% ➞ 변화 폭 제한, 가스비 급등 방지
  - Gas Price 계산 방식 변경 - Max Priority Fee Per Gas, Max Fee Per Gas
  - Base Fee 소각(Burn)

## **기본 수수료(Base Fee)** : 부모 블록 기본 수수료 ±12.5%

- 부모 블록 사용률
  - 50% 이상 : 증가 ➞ 이전 블록 100% 사용 12.5% 증가
  - 50% 미만 : 하락 ➞ 이전 블록 0% 사용 12.5% 감소
- Max Priority Fee : 채굴자 팁(Base Fee에서 추가 지불할 최대 Fee) ➞ 트랜잭션 우선순위 증가
- MaxFee : 최대 수수료, MaxFee >= MaxPriorityFee
- Gas Price : 가스 비, MIN ( MaxFee, BaseFee + MaxPriorityFee )
- 소각(Burn) : 채굴자들이 Base Fee를 일부러 높게 만드는 것을 막기 위함
  - 채굴자에게 집중되어 있는 보상을 네트워크 사용자 전체에게 돌아가게 하는 효과
  - 소각량은 네트워크 사용량이 많아지면 BaseFee가 비례해서 조금씩 증가하므로 같이 증가
  - 사용량 증가 ➞ 소각량 증가 ➞ 이더리움 가치 증가 ➞ 사용량 증가 의 선순환

# Base Fee

$$Base\ Fee = Parent\ Base\ Fee\ * (1 + \left(\frac{1}{8} * \frac{Parent\ Gas\ Used\ - Parent\ Target\ Gas\ Used}{Parent\ Target\ Gas\ Used}\right))$$



Tx Gas Price —① All Gas Price→ Miner

After EIP-1559

Parent Block

Base Fee
Gas Limit
Gas Used

Transaction

② Priority Fee Per Gas
③ Max Fee Per Gas

① Base Fee 계산

④ Gas Price 계산
1.Base Fee + Priority Fee
2.Max Fee

—1,2 중 Min→ Gas Price

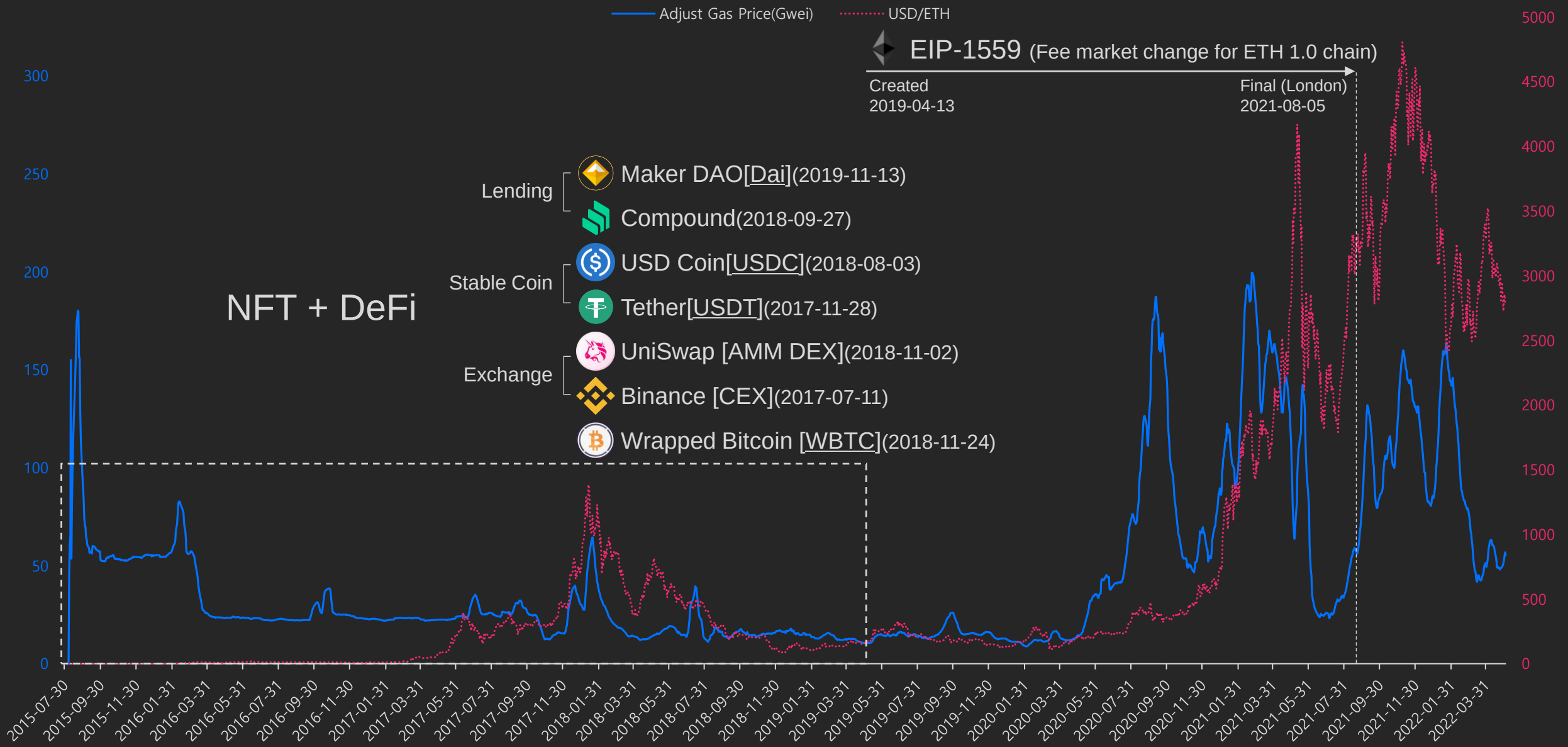
⑤ Gas Price – Base Fee → Miner

⑤ Base Fee → Burn

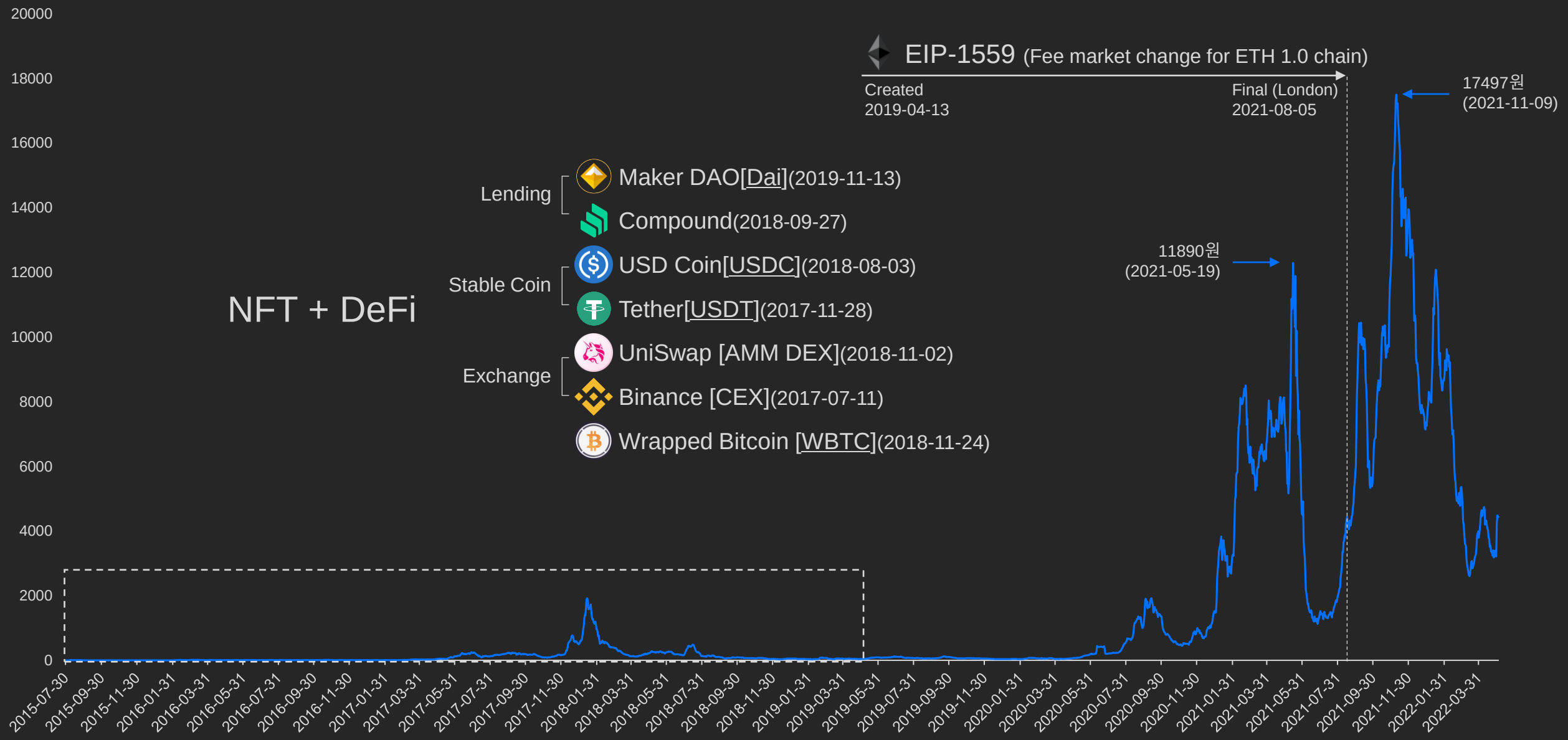
# Gas Price
# MIN(MaxFee, BaseFee + MaxPriorityFee)

| MaxFee | MaxPriorityFee | BaseFee | Gas Price | Burn | PriorityFee (채굴자 팁) |
|---|---|---|---|---|---|
| 50 | 0 | 1 | 1 | 1 | 0 |
| 50 | 10 | 1 | 11 | 1 | 10 |
| 50 | 10 | 40 | 50 | 40 | 10 |
| 50 | 10 | 45 | 50 | 45 | 50 - 45 = 5 |
| 50 | 10 | 50 | 50 | 50 | 50 - 50 =0 |
| 50 | 10 | 60 | 50 | | Max Fee < Base Fee<br>처리 되지 않음<br>Base Fee가 Max Fee 와 같거나 작아질 때 까지 대기 |

# Adjust Gas Price(Gwei) , USD/ETH



# ETH Transfer Cost (원, 21000 gas) (원/달러 = 1100)



서명 없는 Tx 구성
RLP(서명 없는 Tx)
Typed Tx면 Type Prefix 추가
keccak-256(RLP 및 Prefix 추가한 Tx)
m = 서명 없는 Tx Hash
ECDSA Sign(m, privateKey)
r, s, yParity
Tx에 서명정보 추가
Sign Tx
RLP(Sign Tx)
Typed Tx면 Type Prefix 추가
Raw Tx
keccak-256
Tx Hash

# 트랜잭션 생성 [ Legacy ]

Tx : RLP([nonce, gasPrice, gasLimit, to, value, data, v, r, s])

서명 없는 Tx : RLP([nonce, gasPrice, gasLimit, to, value, data, 0x, 0x, 0x])

```
import RLP from 'rlp';
import { createRequire } from 'module';
const require = createRequire(import.meta.url);
import util from 'ethereumjs-util';
const secp256k1 = require("ethereum-cryptography/secp256k1");
import Web3 from 'web3';

function hex(buffer, prefix = true) {
    return (prefix ? '0x' : '') + buffer.toString('hex');
}

(async () => {
// Prepare
const web3 =
    new Web3(new Web3.providers.HttpProvider('http://localhost:7545'));
const privateKey =
   '8d46f2260091f89d63a73a2171234835d272e0cab9fb630b4306aaf72f22e830';
const account = web3.eth.accounts.privateKeyToAccount(privateKey);

//// 1.서명을 위한 트랜잭션 구성 및 RLP serialize 결과의 Hash값 생성
const nonce = await web3.eth.getTransactionCount(account.address, 'pending');
const gasPrice = "0x06fc23ac00"; // 30 * 10 ** 9 wei
const gasLimit = "0x0f4240"; // 100,000 gas
const to = "0x60e69B73db38D52C70690a8EfCeE30383190CDFA";
const value = "0x";  // 0 wei
// data: ABI Encoding of transfer('0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39',
1000)
const data =
"0xa9059cbb000000000000000000000000911d6b77014fa58afd85be49e5148cbeaa3fee390000
000000000000000000000000000000000000000000000000000000000003e8";
let v = '0x';  // 0
let r = '0x';  // 0
let s = '0x';  // 0
const txItems = [nonce, gasPrice, gasLimit, to, value, data, v, r, s];
const txRlp = RLP.encode(txItems);
const txHash = util.keccak256(txRlp);

//// 2. Hash값과 개인키로 서명 값(recId, r, s) 생성 및 v 값 계산
let [signature, recId] =
    secp256k1.signSync(txHash, privateKey, {recovered: true, der: false});
// 27 + signatureRParity
v = new util.BN(27 + recId);
r = Buffer.from(signature.slice(0, 32))
s = Buffer.from(signature.slice(32, 64))

//// 3. 서명된 트랜잭션 구성 후 RLP serialize
let signedTxItems = [nonce, gasPrice, gasLimit, to, value, data, hex(v),
hex(r), hex(s)];
let signedTxRlp = RLP.encode(signedTxItems);
let signedTxHash = util.keccak256(signedTxRlp);

//// Send Transaction
web3.eth.sendSignedTransaction(hex(signedTxRlp)).on('receipt', console.log);
})();
```

# 트랜잭션 생성 [ Legacy EIP-155 ]

Tx : RLP([nonce, gasPrice, gasLimit, to, value, data, v, r, s])

서명 없는 Tx : RLP([nonce, gasPrice, gasLimit, to, value, data, chainId, 0x, 0x])

```javascript
import RLP from 'rlp';
import { createRequire } from 'module';
const require = createRequire(import.meta.url);
import util from 'ethereumjs-util';
const secp256k1 = require("ethereum-cryptography/secp256k1");
import Web3 from 'web3';

function hex(buffer, prefix = true) {
    return (prefix ? '0x' : '') + buffer.toString('hex');
}

(async () => {
// Prepare
const web3 =
    new Web3(new Web3.providers.HttpProvider('http://localhost:7545'));
const privateKey =
   '8d46f2260091f89d63a73a2171234835d272e0cab9fb630b4306aaf72f22e830';
const account = web3.eth.accounts.privateKeyToAccount(privateKey);

//// 1.서명을 위한 트랜잭션 구성 및 RLP serialize 결과의 Hash값 생성
const nonce = await web3.eth.getTransactionCount(account.address, 'pending');
const gasPrice = "0x06fc23ac00"; // 30 * 10 ** 9 wei
const gasLimit = "0x0f4240"; // 100,000 gas
const to = "0x60e69B73db38D52C70690a8EfCeE30383190CDFA";
const value = "0x";  // 0 wei
// data: ABI Encoding of transfer('0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39',
1000)
const data =
"0xa9059cbb000000000000000000000000911d6b77014fa58afd85be49e5148cbeaa3fee390000
000000000000000000000000000000000000000000000000000000000003e8";
```

```javascript
// EIP-155: chainId를 포함하여 서명을 위한 Hash 생성
let v = '0x' + chainId.toString(16);
let r = '0x';  // 0
let s = '0x';  // 0
const txItems = [nonce, gasPrice, gasLimit, to, value, data, '0x', '0x', '0x'];
const txRlp = RLP.encode(txItems);
const txHash = util.keccak256(txRlp);

//// 2. Hash값과 개인키로 서명 값(recId, r, s) 생성 및 v 값 계산
let [signature, recId] =
  secp256k1.signSync(txHash, privateKey, {recovered: true, der: false});
// 27 + signatureRParity
v = new util.BN(27 + recId);
r = Buffer.from(signature.slice(0, 32))
s = Buffer.from(signature.slice(32, 64))

//// 3. 서명된 트랜잭션 구성 후 RLP serialize
let signedTxItems =
  [nonce, gasPrice, gasLimit, to, value, data, hex(v), hex(r), hex(s)];
let signedTxRlp = RLP.encode(signedTxItems);
let signedTxHash = util.keccak256(signedTxRlp);

//// Send Transaction
web3.eth.sendSignedTransaction(hex(signedTxRlp)).on('receipt', console.log);

})();
```

Tx : 0x01 || RLP([chainId, nonce, gasPrice, gasLimit, to, value, data, accessList, yParity, signatureR, signatureS])
서명 없는 Tx : RLP([chainId, nonce, gasPrice, gasLimit, to, value, data, accessList])

```
import RLP from 'rlp';
import { createRequire } from 'module';
const require = createRequire(import.meta.url);
import util from 'ethereumjs-util';
const secp256k1 = require("ethereum-cryptography/secp256k1");
import Web3 from 'web3';

function hex(buffer, prefix = true) {
    return (prefix ? '0x' : '') + buffer.toString('hex');
}

(async () => {
// Prepare
const web3 =
    new Web3(new Web3.providers.HttpProvider('http://localhost:7545'));
const chainId = '0x' + (await web3.eth.getChainId()).toString(16);  // 4693
const privateKey =
    '8d46f2260091f89d63a73a2171234835d272e0cab9fb630b4306aaf72f22e830';
const account = web3.eth.accounts.privateKeyToAccount(privateKey);

const type1 = Buffer.from('01', 'hex');
const nonce = await web3.eth.getTransactionCount(account.address, 'pending');
const to = "0x60e69B73db38D52C70690a8EfCeE30383190CDFA".toLowerCase();
// data: ABI Encoding of transfer(address, uint256) =>
transfer("0xCCFABb539c00b027C4aDa322D6BAcb6A1DAf99f0", 2000)
const data =
"0xa9059cbb000000000000000000000000ccfabb539c00b027c4ada322d6bacb6a1daf99f00000
0000000000000000000000000000000000000000000000000000007d0";
const gasPrice = "0x06fc23ac00"; // 30 * 10 ** 9 wei
const gasLimit = "0x0f4240"; // 100,000 gas
const value = "0x";  // 0 wei
const accessList = [];
```

```
let signatureYParity = "0x";  // 0
let signatureR = '0x';  // 0
let signatureS = '0x';  // 0

// 0x01 || rlp([chainId, nonce, gasPrice, gasLimit, to, value, data,
accessList, signatureYParity, signatureR, signatureS])
const txItems = [chainId, nonce, gasPrice, gasLimit, to, value, data,
accessList];
const txRlp = RLP.encode(txItems);

// 0x02 || rlp(items)
const txHash = util.keccak256(Buffer.concat([type1, txRlp]));

//// ECDSA sign 생성
const [signature, recId] =
    secp256k1.signSync(txHash, privateKey, {recovered: true, der: false});
signatureYParity = recId === 0 ? '0x' : '0x1';
signatureR = Buffer.from(signature.slice(0, 32))
signatureS = Buffer.from(signature.slice(32, 64))

//// Signed Tx with v,r,s
const signedTxItems = [
    chainId, nonce, gasPrice, gasLimit,
    to, value, data, accessList,
    signatureYParity, hex(signatureR), hex(signatureS)];
const signedTxRlp = RLP.encode(signedTxItems);
const signedTx = Buffer.concat([type1, signedTxRlp]);
const signedTxHash = util.keccak256(signedTx);

//// Send Transaction
web3.eth.sendSignedTransaction(hex(signedTx)).on('receipt', console.log);
})();
```

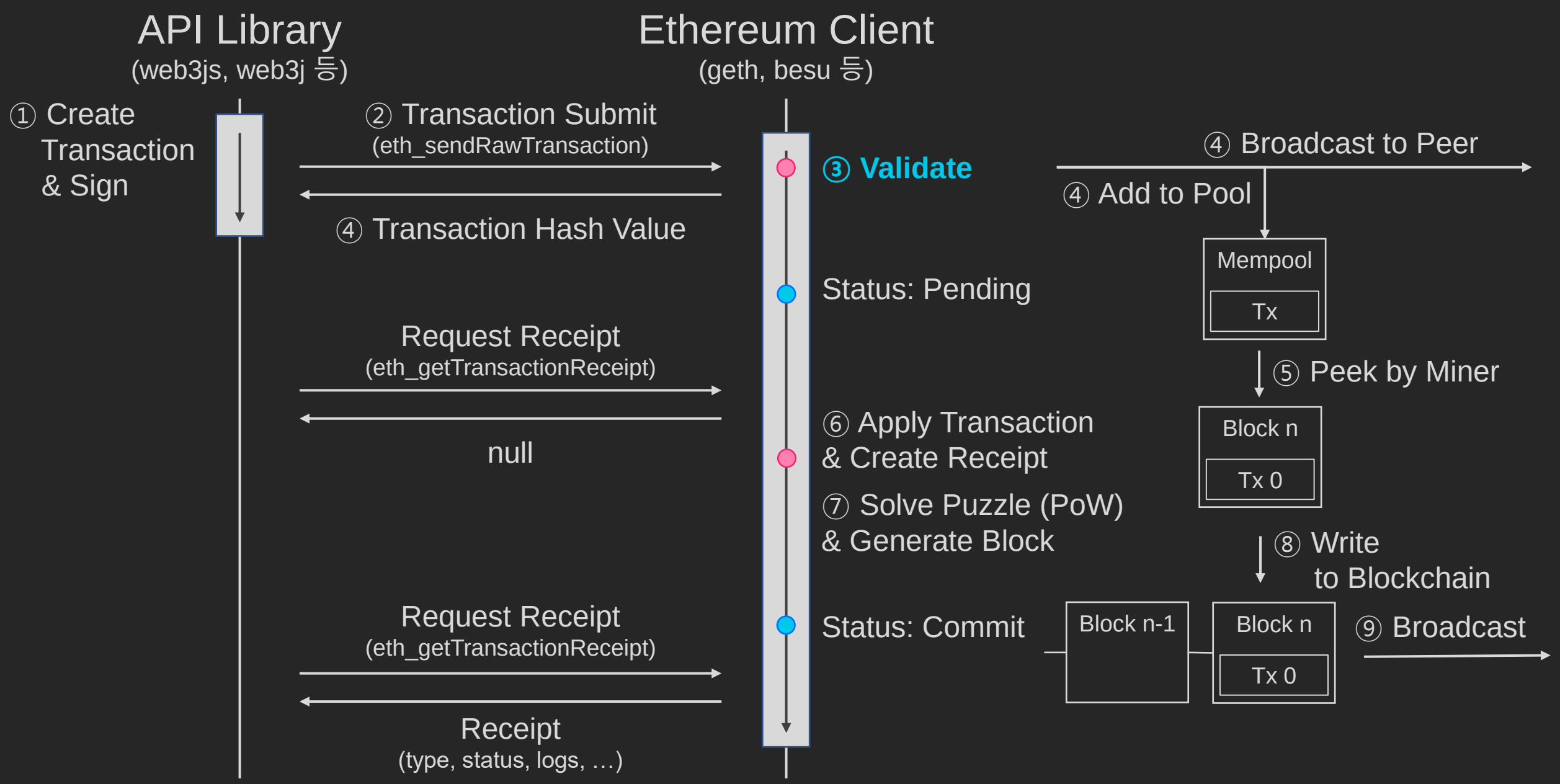
Tx : 0x02 || RLP([chainId, nonce, maxPriorityFeePerGas, maxFeePerGas, gasLimit, destination, amount, data, accessList, signatureYParity, signatureR, signatureS])

서명 없는 Tx : RLP([chainId, nonce, maxPriorityFeePerGas, maxFeePerGas, gasLimit, destination, amount, data, accessList])

```
import RLP from 'rlp';
import { createRequire } from 'module';
const require = createRequire(import.meta.url);
import util from 'ethereumjs-util';
const secp256k1 = require("ethereum-cryptography/secp256k1");
import Web3 from 'web3';
function hex(buffer, prefix = true) {
    return (prefix ? '0x' : '') + buffer.toString('hex');
}

(async () => {
// Prepare
const web3 =
    new Web3(new Web3.providers.HttpProvider('http://localhost:7545'));
const chainId = '0x' + (await web3.eth.getChainId()).toString(16);  // 4693
const privateKey =
    '8d46f2260091f89d63a73a2171234835d272e0cab9fb630b4306aaf72f22e830';
const account = web3.eth.accounts.privateKeyToAccount(privateKey);
const type2 = Buffer.from('02', 'hex');
const nonce = await web3.eth.getTransactionCount(account.address, 'pending');
const destination = "0x60e69B73db38D52C70690a8EfCeE30383190CDFA".toLowerCase();
// data: ABI Encoding of transfer(address, uint256) =>
transfer("0x06B90a7D72E2988ba2711d22e91eb324686104A1", 3000)
const data =
"0xa9059cbb000000000000000000000000006b90a7d72e2988ba2711d22e91eb324686104a10000
00000000000000000000000000000000000000000000000000000000000000000bb8";

const maxPriorityFeePerGas = '0x9502f900';  // 2500000000 wei
const maxFeePerGas = "0x06fc23ac00"; // 30 * 10 ** 9 wei
const gasLimit = "0x0f4240"; // 100,000 gas
const amount = "0x";  // 0 wei
const accessList = [];
```

```
let signatureYParity = "0x";  // 0
let signatureR = '0x';  // 0
let signatureS = '0x';  // 0

// 0x02 || rlp([chain_id, nonce, max_priority_fee_per_gas, max_fee_per_gas,
gas_limit, destination, amount, data, access_list, signature_y_parity,
signature_r, signature_s])
const txItems = [ chainId, nonce, maxPriorityFeePerGas, maxFeePerGas, gasLimit,
    destination, amount, data, accessList];
const txRlp = RLP.encode(txItems);

// 0x02 || rlp(items)
const txHash = util.keccak256(Buffer.concat([type2, txRlp]));

//// ECDSA sign 생성
const [signature, recId] =
    secp256k1.signSync(txHash, privateKey, {recovered: true, der: false});
signatureYParity = recId === 0 ? '0x' : '0x1';
signatureR = Buffer.from(signature.slice(0, 32))
signatureS = Buffer.from(signature.slice(32, 64))

//// Signed Tx with v,r,s
const signedTxItems = [ chainId, nonce, maxPriorityFeePerGas, maxFeePerGas,
    gasLimit, destination, amount, data, accessList,
    signatureYParity, hex(signatureR), hex(signatureS)];
const signedTxRlp = RLP.encode(signedTxItems);
const signedTx = Buffer.concat([type2, signedTxRlp]);
const signedTxHash = util.keccak256(signedTx);

//// Send Transaction
web3.eth.sendSignedTransaction(hex(signedTx)).on('receipt', console.log);
})();
```

# Validate Transaction


API Library
(web3js, web3j 등)
Ethereum Client
(geth, besu 등)
① Create Transaction & Sign
② Transaction Submit
(eth_sendRawTransaction)
④ Transaction Hash Value
③ Validate
④ Broadcast to Peer
④ Add to Pool
Mempool
Tx
Status: Pending
Request Receipt
(eth_getTransactionReceipt)
null
⑤ Peek by Miner
Block n
Tx 0
⑥ Apply Transaction & Create Receipt
⑦ Solve Puzzle (PoW) & Generate Block
⑧ Write to Blockchain
Request Receipt
(eth_getTransactionReceipt)
Status: Commit
Block n-1
Block n
Tx 0
⑨ Broadcast
Receipt
(type, status, logs, …)

## Gas Price, Max Fee, Max Priority Fee

- Gas Price → Max Fee, Max Priority Fee
- Max Fee, Max Priority Fee → Gas Price

```
if ( Gas Price exists )  // [ Legacy , Type 1 ]
  Max Fee = Gas Price
  Max Priority Fee = Gas Price

else  // Max Fee, Max Priority Fee [ Type 2 ]
  Gas Price = MIN ( Base Fee + Max Priority Fee, Max Fee)
```

- 참고
  - Max Fee Per Gas == Gas Fee Cap
  - Max Priority Fee Per Gas == Gas Tip Cap

## Intrinsic Gas ( Tx Gas + Data Gas + Access List Gas)

```
gas = 0

if ( Contract Creation && after Homestead )
  gas += 53000 (TxGasContractCreation)
else
  gas += 21000 (TxGas)

if ( data exists )
  // 0x00(z: zero)와 0x00 아닌 바이트(nz: non-zero) 구분
  z = zeroByte Count
  nz = nonZeroByte Count

  nonZeroGas = 68
  if ( isApply EIP-2028 [after Istanbul (2019-12-08)] )
    nonZeroGas = 16

  gas += nz * nonZeroGas
  gas += z * 4 (TxDataZeroGas)

if ( accessList exists )
  gas += accessList size * 2400 (TxAccessListAddressGas)
  gas += storageKey size * 1900 (TxAccessListStorageKeyGas)

return gas
```

# Validate Transaction & Add to Pool

1. Tx Hash 존재 확인
2. Tx 유효성 검사(Validate Tx)
    1. Type Support Check (Legacy, Type 1, Type 2)
    2. Tx Size < 128KB  (to prevent DOS attacks)
    3. Tx Gas Limit <= Block Gas Limit
    4. Max Fee >= Max Priority Fee
    5. Validate Signature
    6. Max Priority Fee >= Miner Minimum Gas Price
    7. Tx Nonce >= Current State Nonce
    8. Current State Balance >= Value + GasPrice * GasLimit
    9. Tx Gas Limit >= Intrinsic Gas
3. Pending 상태 동일 Nonce Tx 존재 확인
    1. Replace IF ( New Gas Price >= Old Gas Price + Price Bump(10%) )
4. Pool 에 추가

## Gas Price, Max Fee, Max Priority Fee

```
if ( Gas Price exists )  // [ Legacy , Type 1 ]
  Max Fee = Gas Price
  Max Priority Fee = Gas Price

else  // Max Fee, Max Priority Fee [ Type 2 ]
  Gas Price = MIN ( Base Fee + Max Priority Fee, Max Fee)
```

## Intrinsic Gas ( Tx Gas + Data Gas + Access List Gas)

```
gas = 0

if ( Contract Creation && after Homestead )
  gas += 53000 (TxGasContractCreation)
else
  gas += 21000 (TxGas)

if ( data exists )
  // 0x00(z: zero)와 0x00 아닌 바이트(nz: non-zero) 구분
  z = zeroByte Count
  nz = nonZeroByte Count

  nonZeroGas = 68
  if ( isApply EIP-2028 [after Istanbul (2019-12-08)] )
    nonZeroGas = 16

  gas += nz * nonZeroGas
  gas += z * 4 (TxDataZeroGas)

if ( accessList exists )
  gas += accessList size * 2400 (TxAccessListAddressGas)
  gas += storageKey size * 1900 (TxAccessListStorageKeyGas)

return gas
```

# 트랜잭션 Set Gas [ transaction_args.ToMessage() ]

https://github.com/ethereum/go-ethereum/blob/master/internal/ethapi/transaction_args.go#L178

```go
// ToMessage converts the transaction arguments to the Message type used by the core evm. This method is used in calls and traces that do not require a real live transaction.
func (args *TransactionArgs) ToMessage(globalGasCap uint64, baseFee *big.Int) (types.Message, error) {
   // Reject invalid combinations of pre- and post-1559 fee styles
   if args.GasPrice != nil
         && (args.MaxFeePerGas != nil || args.MaxPriorityFeePerGas != nil) {
      return types.Message{}, errors.New(
         "both gasPrice and (maxFeePerGas or maxPriorityFeePerGas) specified")
   }
   // Set sender address or use zero address if none specified.
   addr := args.from()

   // Set default gas & gas price if none were set
   gas := globalGasCap
   if gas == 0 {
      gas = uint64(math.MaxUint64 / 2)
   }
   if args.Gas != nil {
      gas = uint64(*args.Gas)
   }
   if globalGasCap != 0 && globalGasCap < gas {
      log.Warn(
         "Caller gas above allowance, capping",
            "requested", gas, "cap", globalGasCap)
      gas = globalGasCap
   }
   var (
      gasPrice  *big.Int
      gasFeeCap *big.Int
      gasTipCap *big.Int
   )
   if baseFee == nil {
      // If there's no basefee, then it must be a non-1559 execution
      gasPrice = new(big.Int)
      if args.GasPrice != nil {
         gasPrice = args.GasPrice.ToInt()
      }
      gasFeeCap, gasTipCap = gasPrice, gasPrice
   } else {
      // A basefee is provided, necessitating 1559-type execution
      if args.GasPrice != nil {
         // User specified the legacy gas field, convert to 1559 gas typing
         gasPrice = args.GasPrice.ToInt()
         gasFeeCap, gasTipCap = gasPrice, gasPrice
      } else {
         // User specified 1559 gas feilds (or none), use those
         gasFeeCap = new(big.Int)
         if args.MaxFeePerGas != nil {
            gasFeeCap = args.MaxFeePerGas.ToInt()
         }
         gasTipCap = new(big.Int)
         if args.MaxPriorityFeePerGas != nil {
            gasTipCap = args.MaxPriorityFeePerGas.ToInt()
         }
         // Backfill the legacy gasPrice for EVM execution, unless we're all zeroes
         gasPrice = new(big.Int)
         if gasFeeCap.BitLen() > 0 || gasTipCap.BitLen() > 0 {
            gasPrice = math.BigMin(new(big.Int).Add(gasTipCap, baseFee), gasFeeCap)
         }
      }
   }
   value := new(big.Int)
   if args.Value != nil {
      value = args.Value.ToInt()
   }
   data := args.data()
   var accessList types.AccessList
   if args.AccessList != nil {
      accessList = *args.AccessList
   }
   msg := types.NewMessage(
      addr, args.To, 0, value, gas, gasPrice, gasFeeCap, gasTipCap, data,
accessList, true)
   return msg, nil
}
```

Before EIP-1559(London)

After EIP-1559(London)

# 트랜잭션 IntrinsicGas [ state_transition.IntrinsicGas () ]

https://github.com/ethereum/go-ethereum/blob/master/core/state_transition.go#L118

```go
// IntrinsicGas computes the 'intrinsic gas' for a message with the given data.
func IntrinsicGas(data []byte, accessList types.AccessList, isContractCreation bool, isHomestead, isEIP2028 bool) (uint64, error) {
   // Set the starting gas for the raw transaction
   var gas uint64
   if isContractCreation && isHomestead {
      gas = params.TxGasContractCreation
   } else {
      gas = params.TxGas
   }
   // Bump the required gas by the amount of transactional data
   if len(data) > 0 {
      // Zero and non-zero bytes are priced differently
      var nz uint64
      for _, byt := range data {
         if byt != 0 {
            nz++
         }
      }
      // Make sure we don't exceed uint64 for all data combinations
      nonZeroGas := params.TxDataNonZeroGasFrontier
      if isEIP2028 {
         nonZeroGas = params.TxDataNonZeroGasEIP2028
      }
      if (math.MaxUint64-gas)/nonZeroGas < nz {
         return 0, ErrGasUintOverflow
      }
      gas += nz * nonZeroGas

      z := uint64(len(data)) - nz
      if (math.MaxUint64-gas)/params.TxDataZeroGas < z {
         return 0, ErrGasUintOverflow
      }
      gas += z * params.TxDataZeroGas
   }
   if accessList != nil {
      gas += uint64(len(accessList)) * params.TxAccessListAddressGas
      gas += uint64(accessList.StorageKeys()) * params.TxAccessListStorageKeyGas
   }
   return gas, nil
}
```

Base Gas

Data

Access List

https://github.com/ethereum/go-ethereum/blob/master/params/protocol_params.go

```go
// Per transaction not creating a contract.
// NOTE: Not payable on data of calls between transactions.
TxGas                 uint64 = 21000

// Per transaction that creates a contract.
// NOTE: Not payable on data of calls between transactions.
TxGasContractCreation uint64 = 53000

// Per byte of data attached to a transaction that equals zero.
// NOTE: Not payable on data of calls between transactions.
TxDataZeroGas         uint64 = 4


// Per byte of data attached to a transaction that is not equal to zero.
// NOTE: Not payable on data of calls between transactions.
TxDataNonZeroGasFrontier  uint64 = 68

// Per byte of non zero data attached to a transaction after EIP 2028
// (part in Istanbul)
TxDataNonZeroGasEIP2028   uint64 = 16

// Per address specified in EIP 2930 access list
TxAccessListAddressGas    uint64 = 2400

// Per storage key specified in EIP 2930 access list
TxAccessListStorageKeyGas uint64 = 1900
```

# 트랜잭션 Add [ tx_pool.add() ]

https://github.com/ethereum/go-ethereum/blob/master/core/tx_pool.go#L655

```go
// add validates a transaction and inserts it into the non-executable queue for later pending promotion and execution. If the transaction is a replacement for an already
// pending or queued one, it overwrites the previous transaction if its price is higher.
//
// If a newly added transaction is marked as local, its sending account will be added to the allowlist, preventing any associated transaction from being dropped
// out of the pool due to pricing constraints.
func (pool *TxPool) add(tx *types.Transaction, local bool) (replaced bool, err error) {
   // If the transaction is already known, discard it
   hash := tx.Hash()
   if pool.all.Get(hash) != nil {
      log.Trace("Discarding already known transaction", "hash", hash)
      knownTxMeter.Mark(1)
      return false, ErrAlreadyKnown
   }
   // Make the local flag. If it's from local source or it's from the network but
   // the sender is marked as local previously, treat it as the local transaction.
   isLocal := local || pool.locals.containsTx(tx)

   // If the transaction fails basic validation, discard it
   if err := pool.validateTx(tx, isLocal); err != nil {
      log.Trace("Discarding invalid transaction", "hash", hash, "err", err)
      invalidTxMeter.Mark(1)
      return false, err
   }

   // If the transaction pool is full, discard underpriced transactions
   ........

   // Try to replace an existing transaction in the pending pool
   from, _ := types.Sender(pool.signer, tx) // already validated
   if list := pool.pending[from]; list != nil && list.Overlaps(tx) {
      // Nonce already pending, check if required price bump is met
      inserted, old := list.Add(tx, pool.config.PriceBump)
      if !inserted {
         pendingDiscardMeter.Mark(1)
         return false, ErrReplaceUnderpriced
      }
      // New transaction is better, replace old one
      if old != nil {
         pool.all.Remove(old.Hash())
         pool.priced.Removed(1)
         pendingReplaceMeter.Mark(1)
      }
```

Check Already Known

Validate

Try Replace

```go
      pool.all.Add(tx, isLocal)
      pool.priced.Put(tx, isLocal)
      pool.journalTx(from, tx)
      pool.queueTxEvent(tx)
      log.Trace("Pooled new executable transaction", "hash", hash, "from", from, "to",
tx.To())

      // Successful promotion, bump the heartbeat
      pool.beats[from] = time.Now()
      return old != nil, nil
   }
   // New transaction isn't replacing a pending one, push into queue
   replaced, err = pool.enqueueTx(hash, tx, isLocal, true)
   if err != nil {
      return false, err
   }
   // Mark local addresses and journal local transactions
   if local && !pool.locals.contains(from) {
      log.Info("Setting new local account", "address", from)
      pool.locals.add(from)
      // Migrate the remotes if it's marked as local first time.
      pool.priced.Removed(pool.all.RemoteToLocals(pool.locals))
   }
   if isLocal {
      localGauge.Inc(1)
   }
   pool.journalTx(from, tx)

   log.Trace("Pooled new future transaction", "hash", hash, "from", from, "to",
tx.To())
   return replaced, nil
}
```

# 트랜잭션 Validate [ tx_pool.validateTx () ]

https://github.com/ethereum/go-ethereum/blob/master/core/tx_pool.go#L586

```go
const (
   …
   // txSlotSize is used to calculate how many data slots a single transaction
   // takes up based on its size. The slots are used as DoS protection, ensuring
   // that validating a new transaction remains a constant operation (in reality
   // O(maxslots), where max slots are 4 currently).
   txSlotSize = 32 * 1024

   // txMaxSize is the maximum size a single transaction can have. This field has
   // non-trivial consequences: larger transactions are significantly harder and
   // more expensive to propagate; larger transactions also take more resources
   // to validate whether they fit into the pool or not.
   txMaxSize = 4 * txSlotSize // 128KB
)

// validateTx checks whether a transaction is valid according to the consensus
// rules and adheres to some heuristic limits of the local node (price and size).
func (pool *TxPool) validateTx(tx *types.Transaction, local bool) error {
   // Accept only legacy transactions until EIP-2718/2930 activates.
   if !pool.eip2718 && tx.Type() != types.LegacyTxType {
      return ErrTxTypeNotSupported
   }
   // Reject dynamic fee transactions until EIP-1559 activates.
   if !pool.eip1559 && tx.Type() == types.DynamicFeeTxType {
      return ErrTxTypeNotSupported
   }
   // Reject transactions over defined size to prevent DOS attacks
   if uint64(tx.Size()) > txMaxSize {
      return ErrOversizedData
   }
   // Transactions can't be negative. This may never happen using RLP decoded
   // transactions but may occur if you create a transaction using the RPC.
   if tx.Value().Sign() < 0 {
      return ErrNegativeValue
   }
   // Ensure the transaction doesn't exceed the current block limit gas.
   if pool.currentMaxGas < tx.Gas() {
      return ErrGasLimit
   }
   // Sanity check for extremely large numbers
   if tx.GasFeeCap().BitLen() > 256 {
      return ErrFeeCapVeryHigh
   }
   if tx.GasTipCap().BitLen() > 256 {
      return ErrTipVeryHigh
   }
   // Ensure gasFeeCap is greater than or equal to gasTipCap.
   if tx.GasFeeCapIntCmp(tx.GasTipCap()) < 0 {
      return ErrTipAboveFeeCap
   // Make sure the transaction is signed properly.
   from, err := types.Sender(pool.signer, tx)
   if err != nil {
      return ErrInvalidSender
   }
   // Drop non-local transactions under our own minimal accepted gas price or tip
   if !local && tx.GasTipCapIntCmp(pool.gasPrice)
      return ErrUnderpriced
   }
   // Ensure the transaction adheres to nonce ordering
   if pool.currentState.GetNonce(from) > tx.Nonce() {
      return ErrNonceTooLow
   }
   // Transactor should have enough funds to cover the costs
   // cost == V + GP * GL
   if pool.currentState.GetBalance(from).Cmp(tx.Cost()) < 0 {
      return ErrInsufficientFunds
   }
   // Ensure the transaction has more gas than the basic tx fee.
   intrGas, err := IntrinsicGas(
      tx.Data(), tx.AccessList(), tx.To() == nil, true, pool.istanbul)
   if err != nil {
      return err
   }
   if tx.Gas() < intrGas {
      return ErrIntrinsicGas
   }
   return nil
}
```

Type Support Check

Tx Size < 128KB

Tx Gas Limit <= Block Gas Limit

MaxFee >= MaxPriorityFee

Validate Signature

Tx Gas Price >= Miner Minimum Gas Price

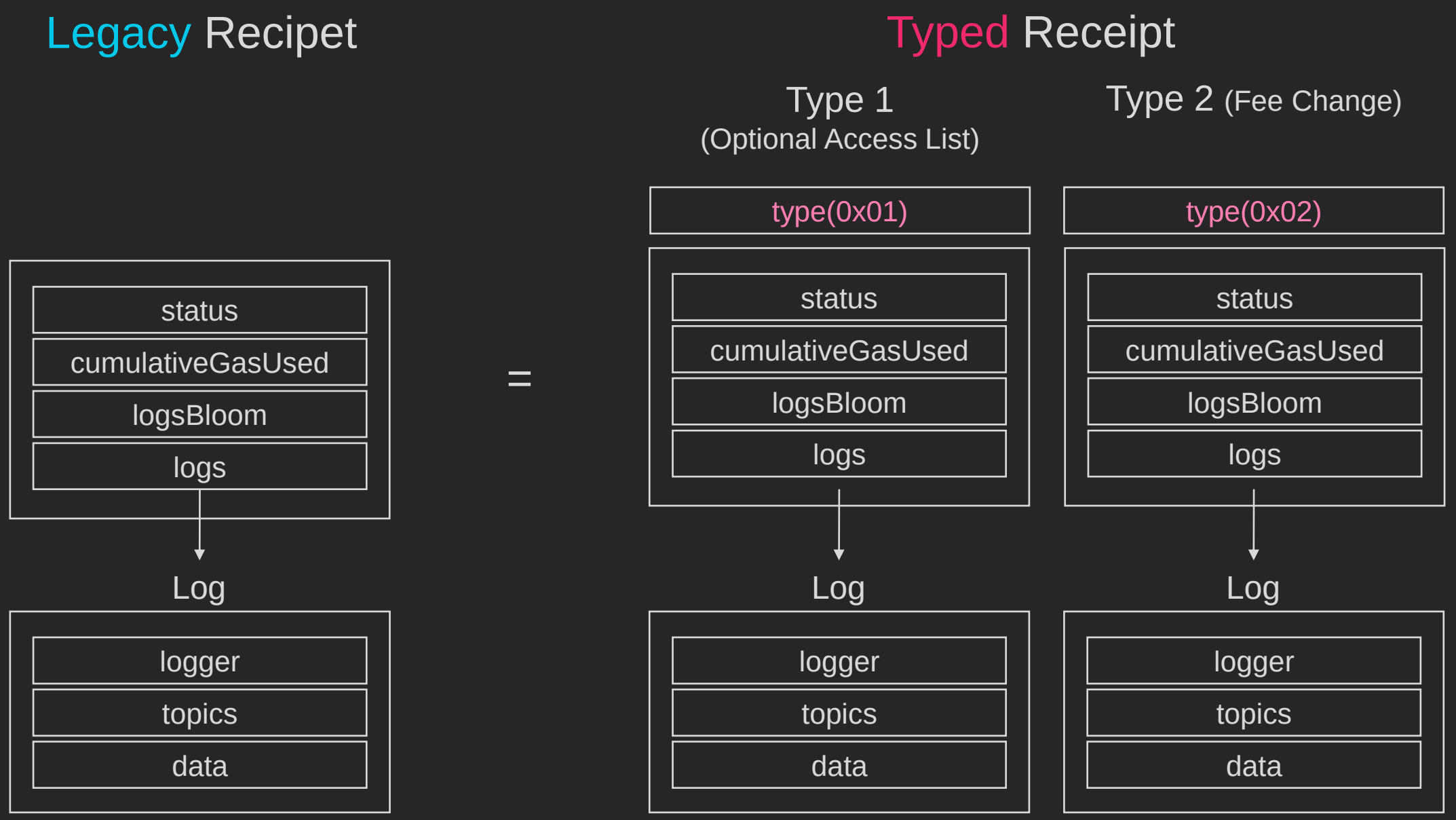
Tx Nonce >= Current State Nonce

Current State Balance >= Value + Gas Price * Gas Limit

Calc Intrinsic Gas

Gas >= Intrinsic Gas

# Receipt


Legacy Recipet
Typed Receipt
Type 1
(Optional Access List)
Type 2 (Fee Change)
type(0x01)
type(0x02)
status
cumulativeGasUsed
logsBloom
logs
=
status
cumulativeGasUsed
logsBloom
logs
status
cumulativeGasUsed
logsBloom
logs
Log
logger
topics
data
Log
logger
topics
data
Log
logger
topics
data

- RECEIPT = [ status, cumulativeGasUsed, bloomFilter, LOGS ]

| 항목 | 타입 | 설명 |
|---|---|---|
| **status** | uint256 | 성공 여부 (0: 실패, 1: 성공) |
| **cumulativeGasUsed** | uint256 | 누적 Gas 사용량 (이전 트랜잭션 누적 Gas 사용량 + 현재 트랜잭션 Gas 사용량) |
| **bloomFilter** | 2048bits | logger, topics 블룸 필터(Bloom Filter)<br>StateDB 저장 X |
| **LOGS** | LOG[] | 로그 목록 |

- LOG = [ logger, TOPICS, data ] ➞ Solidity emit event 시 Log 기록

| 항목 | 타입 | 설명 |
|---|---|---|
| **logger** | address | 로그를 기록한 (컨트랙트) 주소, BloomFilter 대상 |
| **TOPICS** | 256bits | TOPIC 목록 (MAX: 4), BloomFilter 대상<br>keccak-256(event Signature) or RLP(indexed parameter value) |
| **data** | bytes | indexed parameter가 아닌 value들의 RLP 결과 |

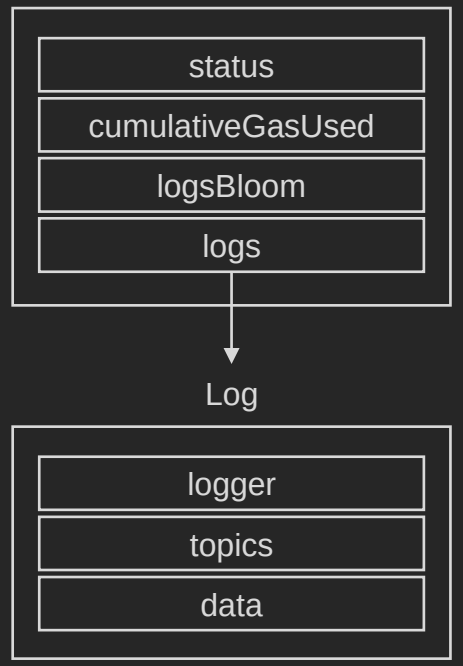
- TOPICS 구성 예 : [ event ABI Signature, indexed value1, indexed value2, indexed value3 ]
  - 블룸필터를 생성하여 Filter API 에 사용 (상태 변경 탐지)
  - eth_newFilter, eth_getFilterChanges, eth_getFilterLogs

```
event Transfer(address indexed from, address indexed to, uint256 value);

emit Transfer(0x942F397B7f4391B43115395F469c63072aEd6E41, 0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39, 1000);
```

# Receipt

RECEIPT = [ status, cumulativeGasUsed, bloomFilter, LOGS ]
LOGS = [ logger, TOPICS, data ]
TOPICS = [{32bytes}] (MAX: 4)

```
[
  "0x01",                                                                  // status, 1(success), 0(fail)
  "0xcc16",                                                                // cumulativeGasUsed, 52246 gas
  [                                                                        // logs
    [                                                                      // └ log
      "0x60e69b73db38d52c70690a8efcee30383190cdfa",                        //   ├ logger, contract address
      [                                                                    //   ├ topics
        "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef", //   │ ├ topic0, event signature
        "0x000000000000000000000000942f397b7f4391b43115395f469c63072aed6e41", //   │ ├ topic1, indexed from
        "0x000000000000000000000000911d6b77014fa58afd85be49e5148cbeaa3fee39"  //   │ └ topic2, indexed to
      ],                                                                   //   │
      "0x0000000000000000000000000000000000000000000000000000000000000003e8"  //   └ data, value
    ]
  ]
]
```



- **Bloom Filter** : 원소가 집합에 속하는지 여부를 검사하는데 사용되는 확률적 자료 구조
- 주어진 data에 대해서 2가지 상황 중 하나를 판단
  - 존재하지 않는다.
  - 존재할 수도 있다.

[ 0, 8, 12 ] →

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

0 % 10 = 0

| **0** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

8 % 10 = 8

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | **8** | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

18 % 10 = 8

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | **8** | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

Bloom Filter

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

1 존재? → 1 % 10 = 1

존재하지 않는다.

18 존재? → 18 % 10 = 8

존재할 수 있다.

| Name | Gas | Description |
|---|---|---|
| $G_{log}$ | 375 | log 호출 gas |
| $G_{logtopic}$ | 375 | topic당 gas |
| $G_{logdata}$ | 8 | byte당 gas |

- Log
  - OpCode : LOG0(375 gas), LOG1(750) , LOG2(1125) , LOG3(1500) , LOG4(1875)
  - Solidity emit event → Log 기록

```
event Transfer(address indexed from, address indexed to, uint256 value);

emit Transfer(0x942F397B7f4391B43115395F469c63072aEd6E41, 0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39, 1000);
```

- Topic (Max : 4)
  - keccak-256(event signature) 예) keccak-256 ("Transfer(address,address,uint256)")
  - RLP(indexed parameter value) → 32bytes 이하 데이터만 저장, 32 bytes 초과시 keccak-256 값이 저장되어 효용 감소
  - 예) LOG3 Topics : [ keccak-256(event Signature), RLP(indexed value1), RLP(indexed value2) ]
    [ "0xddf252ad1be...a4df523b3ef", "0x0000000...aed6e41", "0x0000000...a3fee39" ]
- anonymous event : Topics 에서 event Signature 제외 → indexed 4개 사용 가능

```
event Transfer(address indexed from, address indexed to, uint256 value) anonymous;
```

- Data
  - Indexed Parameter가 아닌 Value들의 RLP 결과

- Receipt Bloom Filter : 모든 Log의 Logger와 Topics 로 2048 bits (256 bytes) Bloom Filter 생성
  - Contract에서 발생하는 많은 이벤트들의 존재 가능성 여부 확인
  - 이벤트 및 파라미터 값을 filtering 하는데 사용 (eth_newFilter, eth_getFilterChanges, eth_getFilterLogs)
- keccak-256(Topic) 결과의 처음 2바이트 씩 3개의 값을 사용
  - Topic : 0x60e69b73db38d52c70690a8efcee30383190cdfa
  - Keccak-256(Topic) : 0x44e229de97d6dcb0cff85acd0eabdd23b939f8cc1cc1ca0595aca9135b1243b9
  - 0x44e2, 0x29de, 0x97d6
- 2개 바이트의 하위 11bit 를 Bloom Filter index로 사용 ( & 0x07ff) ➞ 2048 bits 매핑을 위한 '% 2048' 와 동일
  - 0x44e2 & 0x07ff = 0x4e2 = 1250
  - 0x29de & 0x07ff = 0x1de = 478
  - 0x97d6 & 0x07ff = 0x7d6 = 2006
- 가장 오른쪽 bit를 bit index 0 으로 하여서 1로 표시

| bit index | 2047 | … | 2006 | … | 1250 | … | 478 | … | 0 |
|---|---|---|---|---|---|---|---|---|---|
| value | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

- 검색 : 검색 단어 ➞ 검색 Bloom Filter
  - 검색 Bloom Filter & Receipt Bloom Filter == 검색 Bloom Filter ➞ 존재 할 수 있다.
  - 검색 Bloom Filter & Receipt Bloom Filter != 검색 Bloom Filter ➞ 존재하지 않는다.

```
event Transfer(address indexed from, address indexed to, uint256 value);
emit Transfer(0x942F397B7f4391B43115395F469c63072aEd6E41, 0x911D6B77014FA58aFD85BE49e5148CBEAA3FeE39, 1000);
```

```
"0x60e69b73db38d52c70690a8efcee30383190cdfa"                          // logger
"0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef"  // topic0
"0x000000000000000000000000942f397b7f4391b43115395f469c63072aed6e41"  // topic1
"0x000000000000000000000000911d6b77014fa58afd85be49e5148cbeaa3fee39"  // topic2
```

## Keccak-256(Value) ➞ 왼쪽 2bytes 3개 ➞ 하위 11 bits (Bit Index)

```
keccak-256("0x60e6…cdfa") = 0x44e229de97d6… ➞ 0x44e2,0x29de,0x96d6 & 0x7ff = 1250,478,2006
keccak-256("0xddf2…b3ef") = 0xada389e1fc24… ➞ 0xada3,0x89e1,0xfc24 & 0x7ff = 1443,481,1060
keccak-256("0x0000…6e41") = 0x1bb39cf094aa… ➞ 0x1bb3,0x9cf0,0x94aa & 0x7ff = 947,1264,1194
keccak-256("0x0000…ee39") = 0x8582e9ad66d8… ➞ 0x8582,0xe9ad,0x66d8 & 0x7ff = 1410,429,1752
```

## 256 Bytes(2048 bits)에 매핑을 위한 연산 : Bit Index ➞ (Byte Index, Bit Value)

- Byte Index = 256 - 1 - Math.floor(Bit Index / 8) // 256 = 2048 / 8
- Bit Value = 1 << (bitIndex % 8)

## Bloom Filter 연산

```
let logsBloom = Buffer.alloc(256); // 256bytes(2048 bits)
for [(99,4),(196,64),(5,64),(75,8),(195,2),(123,16),(137,8),(97,1),(106,4),(79,4),(202,32),(36,1)] {
    logsBloom[Byte Index] |= Bit Value;
}
```

# ApplyTransaction

https://github.com/ethereum/go-ethereum/blob/master/core/state_processor.go#L95

```go
func applyTransaction(msg types.Message, config *params.ChainConfig, bc ChainContext, author *common.Address, gp *GasPool, statedb *state.StateDB, blockNumber *big.Int,
      blockHash common.Hash, tx *types.Transaction, usedGas *uint64, evm *vm.EVM) (*types.Receipt, error) {
   // Create a new context to be used in the EVM environment.
   txContext := NewEVMTxContext(msg)
   evm.Reset(txContext, statedb)

   // Apply the transaction to the current state (included in the env).
   result, err := ApplyMessage(evm, msg, gp)
   if err != nil {
      return nil, err
   }

   // Update the state with pending changes.
   var root []byte
   if config.IsByzantium(blockNumber) {
      statedb.Finalise(true)
   } else {
      root = statedb.IntermediateRoot(config.IsEIP158(blockNumber)).Bytes()
   }
   *usedGas += result.UsedGas

   // Create a new receipt for the transaction, storing the intermediate root and gas used by the tx.
   receipt := &types.Receipt{Type: tx.Type(), PostState: root, CumulativeGasUsed: *usedGas}
   if result.Failed() {
      receipt.Status = types.ReceiptStatusFailed
   } else {
      receipt.Status = types.ReceiptStatusSuccessful
   }
   receipt.TxHash = tx.Hash()
   receipt.GasUsed = result.UsedGas

   // If the transaction created a contract, store the creation address in the receipt.
   if msg.To() == nil {
      receipt.ContractAddress = crypto.CreateAddress(evm.TxContext.Origin, tx.Nonce())
   }

   // Set the receipt logs and create the bloom filter.
   receipt.Logs = statedb.GetLogs(tx.Hash(), blockHash)
   receipt.Bloom = types.CreateBloom(types.Receipts{receipt})
   receipt.BlockHash = blockHash
   receipt.BlockNumber = blockNumber
   receipt.TransactionIndex = uint(statedb.TxIndex())
   return receipt, err
}
```

New EVM

Apply Message

CumulativeGasUsed

Receipt

Status

Bloom Filter

# Bloom Filter [ bloom9 ]

```go
// CreateBloom creates a bloom filter out of the give Receipts (+Logs)
func CreateBloom(receipts Receipts) Bloom {
   buf := make([]byte, 6)
   var bin Bloom
   for _, receipt := range receipts {
      for _, log := range receipt.Logs {
         bin.add(log.Address.Bytes(), buf)
         for _, b := range log.Topics {
            bin.add(b[:], buf)
         }
      }
   }
   return bin
}
```

loop receipts
loop logs
add logger
add topics

```go
// add is internal version of Add, which takes a scratch buffer for reuse (needs to be
at least 6 bytes)
func (b *Bloom) add(d []byte, buf []byte) {
   i1, v1, i2, v2, i3, v3 := bloomValues(d, buf)
   b[i1] |= v1
   b[i2] |= v2
   b[i3] |= v3
}
```

calc 3 (byte index, bit value)
or to bloomFilter

```go
// bloomValues returns the bytes (index-value pairs) to set for the given data
func bloomValues(data []byte, hashbuf []byte) (uint, byte, uint, byte, uint, byte) {
   sha := hasherPool.Get().(crypto.KeccakState)
   sha.Reset()
   sha.Write(data)
   sha.Read(hashbuf)
   hasherPool.Put(sha)
   // The actual bits to flip
   v1 := byte(1 << (hashbuf[1] & 0x7))
   v2 := byte(1 << (hashbuf[3] & 0x7))
   v3 := byte(1 << (hashbuf[5] & 0x7))
   // The indices for the bytes to OR in
   i1 := BloomByteLength - uint((binary.BigEndian.Uint16(hashbuf)&0x7ff)>>3) - 1
   i2 := BloomByteLength - uint((binary.BigEndian.Uint16(hashbuf[2:])&0x7ff)>>3) - 1
   i3 := BloomByteLength - uint((binary.BigEndian.Uint16(hashbuf[4:])&0x7ff)>>3) - 1

   return i1, v1, i2, v2, i3, v3
}
```

first 2 bytes 3
right 11 bits

byte index, bit value

# Block

4.3. **The Block.** The block in Ethereum is the collection of relevant pieces of information (known as the block *header*), $H$, together with information corresponding to the comprised transactions, $\mathbf{T}$, and a set of other block headers $\mathbf{U}$ that are known to have a parent equal to the present block's parent's parent (such blocks are known as *ommers*[2]). The block header contains several pieces of information:

**parentHash:** The Keccak 256-bit hash of the parent block's header, in its entirety; formally $H_{\mathrm{p}}$.

**ommersHash:** The Keccak 256-bit hash of the ommers list portion of this block; formally $H_{\mathrm{o}}$.

**beneficiary:** The 160-bit address to which all fees collected from the successful mining of this block be transferred; formally $H_{\mathrm{c}}$.

**stateRoot:** The Keccak 256-bit hash of the root node of the state trie, after all transactions are executed and finalisations applied; formally $H_{\mathrm{r}}$.

**transactionsRoot:** The Keccak 256-bit hash of the root node of the trie structure populated with each transaction in the transactions list portion of the block; formally $H_{\mathrm{t}}$.

**receiptsRoot:** The Keccak 256-bit hash of the root node of the trie structure populated with the receipts of each transaction in the transactions list portion of the block; formally $H_{\mathrm{e}}$.

**logsBloom:** The Bloom filter composed from indexable information (logger address and log topics) contained in each log entry from the receipt of each transaction in the transactions list; formally $H_{\mathrm{b}}$.

**difficulty:** A scalar value corresponding to the difficulty level of this block. This can be calculated from the previous block's difficulty level and the timestamp; formally $H_{\mathrm{d}}$.

**number:** A scalar value equal to the number of ancestor blocks. The genesis block has a number of zero; formally $H_{\mathrm{i}}$.

**gasLimit:** A scalar value equal to the current limit of gas expenditure per block; formally $H_{\mathrm{l}}$.

**gasUsed:** A scalar value equal to the total gas used in transactions in this block; formally $H_{\mathrm{g}}$.

**timestamp:** A scalar value equal to the reasonable output of Unix's time() at this block's inception; formally $H_{\mathrm{s}}$.

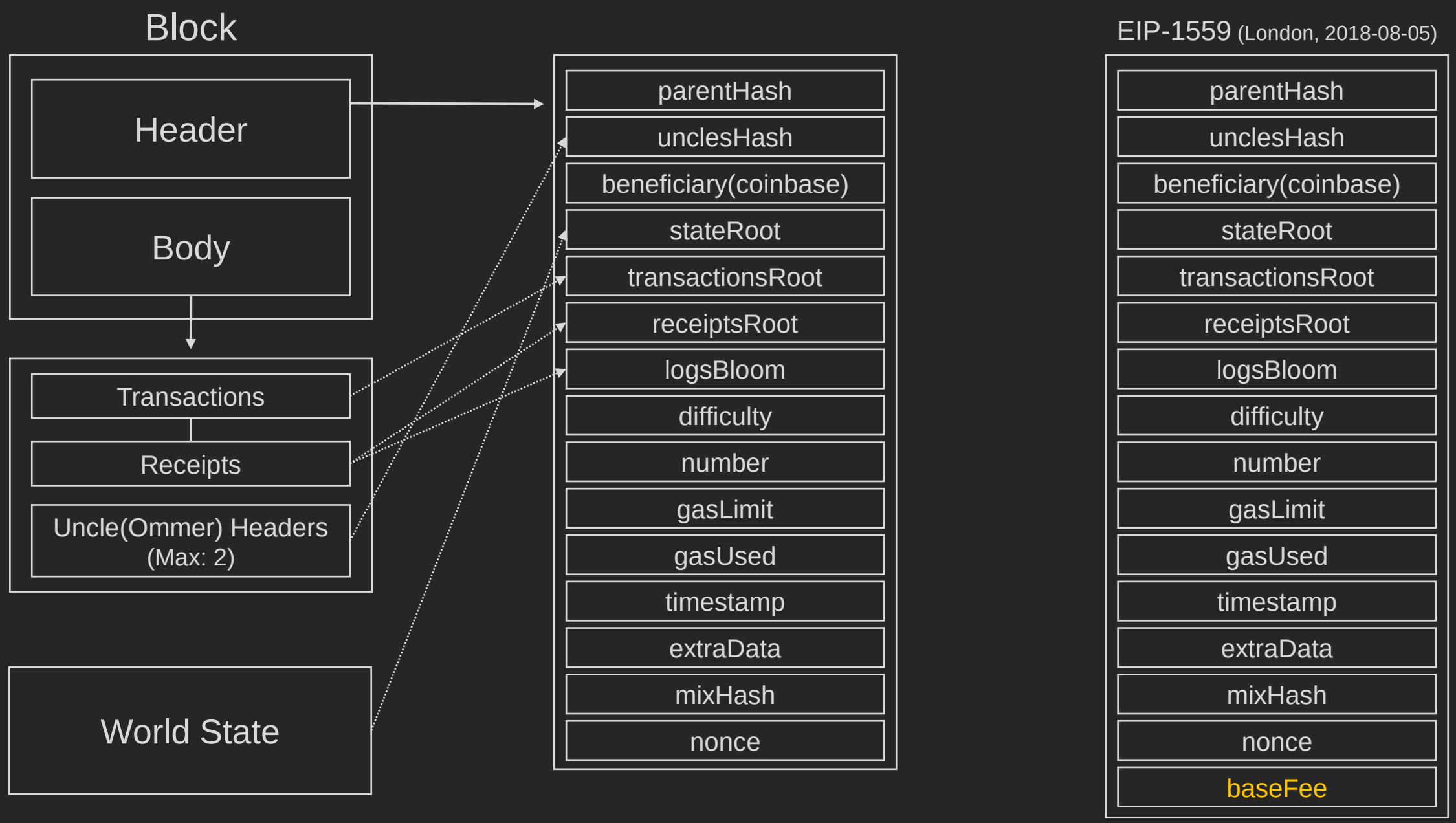
**extraData:** An arbitrary byte array containing data relevant to this block. This must be 32 bytes or fewer; formally $H_{\mathrm{x}}$.

**mixHash:** A 256-bit hash which, combined with the nonce, proves that a sufficient amount of computation has been carried out on this block; formally $H_{\mathrm{m}}$.

**nonce:** A 64-bit value which, combined with the mixhash, proves that a sufficient amount of computation has been carried out on this block; formally $H_{\mathrm{n}}$.

The other two components in the block are simply a list of ommer block headers (of the same format as above), $B_{\mathbf{U}}$ and a series of the transactions, $B_{\mathbf{T}}$. Formally, we can refer to a block $B$:

$$B \equiv (B_{\mathrm{H}}, B_{\mathbf{T}}, B_{\mathbf{U}}) \tag{21}$$


Block
Header
Body
Transactions
Receipts
Uncle(Ommer) Headers
(Max: 2)
World State
parentHash
unclesHash
beneficiary(coinbase)
stateRoot
transactionsRoot
receiptsRoot
logsBloom
difficulty
number
gasLimit
gasUsed
timestamp
extraData
mixHash
nonce
EIP-1559 (London, 2018-08-05)
parentHash
unclesHash
beneficiary(coinbase)
stateRoot
transactionsRoot
receiptsRoot
logsBloom
difficulty
number
gasLimit
gasUsed
timestamp
extraData
mixHash
nonce
baseFee

| 항목 | 타입 | 설명 |
|---|---|---|
| **parentHash** | 256 bits | 부모 블록 Hash |
| **unclesHash** | 256 bits | Uncle Headers의 Hash 값 |
| **beneficiary** | address | Block Reward, Uncle Inclusion Reward 및 Transaction Gas Tip을 받는 address |
| **stateRoot** | 256 bits | (Account) State Trie Root Node Key |
| **transactionsRoot** | 256 bits | Transactions Trie Root Node Key |
| **receiptsRoot** | 256 bits | Receipts Trie Root Node Key |
| **logsBloom** | 2048 bits | 모든 Receipt Bloom Filter의 or(\|) 연산 결과 |
| **difficulty** | uint256 | PoW(ethash) 퍼즐 난이도 |
| **number** | uint256 | Block 번호 |
| **gasLimit** | uint64 | 블록의 사용가능 최대 Gas (= Block Size) |
| **gasUsed** | uint64 | 블록의 실제 사용 Gas (전체 트랜잭션 Gas 사용량의 합) |
| **timestamp** | uint64 | 블록 생성 시점 Unix Timestamp (초) |
| **extraData** | 32 bytes | 블록 생성자가 기록할 수 있는 임의 32bytes 이하 데이터 (일반적으로 채굴자 기록) |
| **mixHash** | 256 bits | nonce가 충분한 양의 계산이 수행되었음을 증명<br>Dagger Hashimoto : https://eth.wiki/concepts/dagger-hashimoto |
| **nonce** | uint64 | ethash 퍼즐 정답 |
| **baseFee** | uint256 | BaseFee by EIP-1559, ignore before EIP-1559 |

- **Block Hash** = keccak-256 ( RLP ( Block Header ) )
- Block Header
  - Frontier
    [ parentHash, ommersHash, coinbase, stateRoot, transactionRoot, receiptRoot, logsBloom, difficulty, number, gasLimit, gasUsed, timestamp, extraData, mixHash, nonce ]
  - after EIP-1559 (London, 2021-08-05)
    [ parentHash, ommersHash, coinbase, stateRoot, transactionRoot, receiptRoot, logsBloom, difficulty, number, gasLimit, gasUsed, timestamp, extraData, mixHash, nonce, baseFee ]

- **Uncles Hash** = keccak-256 ( RLP ( Uncle Hash List ) )
- Uncle Hash List (Max: 2)
  - [ ]
  - [ Ommer0 Hash ]
  - [ Ommer0 Hash, Ommer1 Hash ]

- Beneficiary (Coinbase)
  - Reward 및 Tip 수령 주소
  - Block Reward, Uncle Inclusion Reward, Gas Tip
- Block Reward
  - Frontier : 5 ETH
  - Byzantium (2017-10-16) : 3 ETH
  - Constantinople (2019-02-28) : 2 ETH
- Uncle Inclusion Reward (MAX : 2)
  - Block Reward * Uncles Size / 32
  - Uncle Header 당 Block Reward 3.125%
- Transaction Gas Tip
  - Before EIP-1559 : Gas Price * Gas Used
  - After EIP-1559 : (Gas Price - Base Fee) * Gas Used

- Uncle Block : 고아(orphan) 블록 중 일정 조건 만족
  - Block의 2 ~ 7번째 조상의 직접적인 자식, 부모의 자식은 불가
  - 유효한 Header Block
  - 중복 불가, 최대 2개
  - 용도: PoW (ethash 및 mGHOST) 로직에 사용
- 삼촌 블록 보상(Uncle Block Reward)
  - (Uncle Block Height – Block Height + 8) * Block Reward / 8
  - 가까운 Block 일수록 높은 Reward (Block Reward 25% ~ 87.5%)

| AH – BH | UH - BH | UH – BH +8 | (UH - BH +8) / 8 |
|---|---|---|---|
| -7 | -6 | 2 | 0.25 |
| -6 | -5 | 3 | 0.375 |
| -5 | -4 | 4 | 0.5 |
| -4 | -3 | 5 | 0.625 |
| -3 | -2 | 6 | 0.75 |
| -2 | -1 | 7 | 0.875 |

BH: Block Height
AH: Ancestor Height
UH: Uncle Height

- Miner Reward
  - Uncle 당 Block Reward 3.125% (÷ 32)

# Uncle Selection

Manage uncles

```go
case ev := <-w.chainSideCh:
   // Short circuit for duplicate side blocks
   …
   // Add side block to possible uncle block set depending on the author.
   if w.isLocalBlock != nil && w.isLocalBlock(ev.Block.Header()) {
      w.localUncles[ev.Block.Hash()] = ev.Block
   } else {
      w.remoteUncles[ev.Block.Hash()] = ev.Block
   }
   // If our sealing block contains less than 2 uncle blocks,
   // add the new uncle block if valid and regenerate a new
   // sealing block for higher profit.
   if w.isRunning() && w.current != nil && len(w.current.uncles) < 2 {
      start := time.Now()
      if err := w.commitUncle(w.current, ev.Block.Header()); err == nil {
         w.commit(w.current.copy(), nil, true, start)
      }
   }
```

Local Block

Remote Block

```go
env, err := w.makeEnv(parent, header, genParams.coinbase)
if err != nil {
   log.Error("Failed to create sealing context", "err", err)
   return nil, err
}
```

```go
// makeEnv creates a new environment for the sealing block.
func (w *worker) makeEnv(parent *types.Block, header *types.Header, coinbase
common.Address) (*environment, error) {
   …
   // Note the passed coinbase may be different with header.Coinbase.
   env := &environment{
      signer:    types.MakeSigner(w.chainConfig, header.Number),
      state:     state,
      coinbase:  coinbase,
      ancestors: mapset.NewSet(),
      family:    mapset.NewSet(),
      header:    header,
      uncles:    make(map[common.Hash]*types.Header),
   }
   // when 08 is processed ancestors contain 07 (quick block)
   for _, ancestor := range w.chain.GetBlocksFromHash(parent.Hash(), 7) {
      for _, uncle := range ancestor.Uncles() {
         env.family.Add(uncle.Hash())
      }
      env.family.Add(ancestor.Hash())
      env.ancestors.Add(ancestor.Hash())
   }
   // Keep track of transactions which return errors so they can be removed
   env.tcount = 0
   return env, nil
}
```

~ 7 ancestors

family = ancestor + uncles
check for duplicate

Find uncles

```go
// Accumulate the uncles for the sealing work only if it's allowed.
if !genParams.noUncle {
   commitUncles := func(blocks map[common.Hash]*types.Block) {
      for hash, uncle := range blocks {
         if len(env.uncles) == 2 {
            break
         }
         if err := w.commitUncle(env, uncle.Header()); err != nil {
            log.Trace("Possible uncle rejected", "hash", hash, "reason", err)
         } else {
            log.Debug("Committing new uncle to block", "hash", hash)
         }
      }
   }
   // Prefer to locally generated uncle
   commitUncles(w.localUncles)
   commitUncles(w.remoteUncles)
}
```

Max : 2

Commit uncle

```go
// commitUncle adds the given block to uncle block set, returns error if failed to add.
func (w *worker) commitUncle(env *environment, uncle *types.Header) error {
   if w.isTTDReached(env.header) {
      return errors.New("ignore uncle for beacon block")
   }
   hash := uncle.Hash()
   if _, exist := env.uncles[hash]; exist { return errors.New("uncle not unique") }
   if env.header.ParentHash == uncle.ParentHash { return errors.New("uncle is sibling")
}
   if !env.ancestors.Contains(uncle.ParentHash) {
      return errors.New("uncle's parent unknown")
   }
   if env.family.Contains(hash) { return errors.New("uncle already included") }
   env.uncles[hash] = uncle
   return nil
}
```

Check unique, not parent's child, proper ancestor(2~7), not duplicated

# Beneficiary (Coinbase), Uncle Reward, Transaction Gas Tip

https://github.com/ethereum/go-ethereum/blob/master/consensus/ethash/consensus.go#L645

```go
var (
   // Block reward in wei for successfully mining a block
   FrontierBlockReward           = big.NewInt(5e+18)
   // Block reward in wei for successfully mining a block upward
   ByzantiumBlockReward          = big.NewInt(3e+18) from Byzantium
   // Block reward in wei for successfully mining a block upward from Constantinople
   ConstantinopleBlockReward     = big.NewInt(2e+18)
)

// AccumulateRewards credits the coinbase of the given block with the mining
// reward. The total reward consists of the static block reward and rewards for
// included uncles. The coinbase of each uncle block is also rewarded.
func accumulateRewards(config *params.ChainConfig, state *state.StateDB, header
*types.Header, uncles []*types.Header) {
   // Select the correct block reward based on chain progression
   blockReward := FrontierBlockReward
   if config.IsByzantium(header.Number) {
      blockReward = ByzantiumBlockReward
   }
   if config.IsConstantinople(header.Number) {
      blockReward = ConstantinopleBlockReward
   }
   // Accumulate the rewards for the miner and any included uncles
   reward := new(big.Int).Set(blockReward)
   r := new(big.Int)
   for _, uncle := range uncles {
      r.Add(uncle.Number, big8)
      r.Sub(r, header.Number)
      r.Mul(r, blockReward)
      r.Div(r, big8)
      state.AddBalance(uncle.Coinbase, r)

      r.Div(blockReward, big32)
      reward.Add(reward, r)
   }
   state.AddBalance(header.Coinbase, reward)
}
```

Block Reward

Uncle Reward

Uncle Inclusion Reward

https://github.com/ethereum/go-ethereum/blob/master/core/state_transition.go#L343

Transaction Gas Tip

```go
func (st *StateTransition) TransitionDb() (*ExecutionResult, error) {
   …
   effectiveTip := st.gasPrice
   if rules.IsLondon {
   effectiveTip =
      cmath.BigMin(st.gasTipCap,
         new(big.Int).Sub(st.gasFeeCap, st.evm.Context.BaseFee))
   }
   st.state.AddBalance(
      st.evm.Context.Coinbase,
      new(big.Int).Mul(new(big.Int).SetUint64(st.gasUsed()), effectiveTip))
  …
}
```

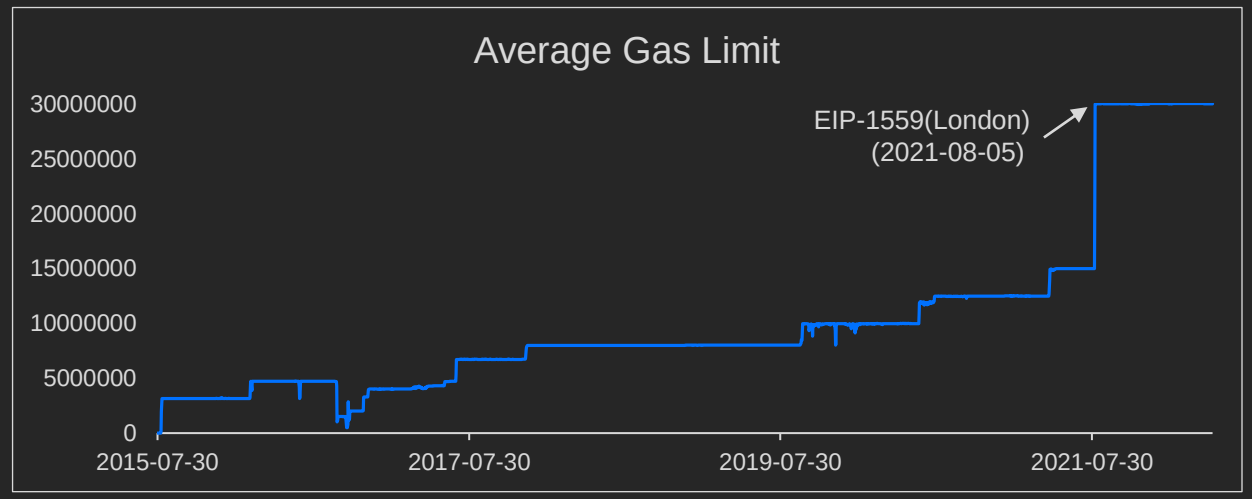
- EIP-1559 (London) 이전 Target Size : 15,000,000 gas
- EIP-1559 (London) 이후 : 30,000,000 gas (2배)
  - Hardfork 시 Gas Limit 2 배로 설정

```
// Set baseFee and GasLimit if we are on an EIP-1559 chain
if w.chainConfig.IsLondon(header.Number) {
   header.BaseFee = misc.CalcBaseFee(w.chainConfig, parent.Header())
   if !w.chainConfig.IsLondon(parent.Number()) {
      parentGasLimit := parent.GasLimit() * params.ElasticityMultiplier // * 2
      header.GasLimit = core.CalcGasLimit(parentGasLimit, w.config.GasCeil)
   }
}
```

## Gas Limit 계산

- Parent Gas Limit / 1024 범위 내에서 변경
  Delta = ( Parent Gas Limit / 1024 ) - 1
- Parent Gas Limit 와 Node Gas Ceiling(한계) 를 비교
  - Parent Gas Limit = Node Gas Ceiling ➞ Parent Gas Limit
  - Parent Gas Limit > Node Gas Ceiling ➞ Parent Gas Limit - Delta
  - Parent Gas Limit < Node Gas Ceiling ➞ Parent Gas Limit + Delta

```
parentGasLimit, nodeGasCeiling, gasLimit
delta = (parentGasLimit / 1024) - 1

if (parentGasLimit > nodeGasCeiling) {
  gasLimit = parentGasLimit - delta
  if (gasLimit > nodeGasCeiling) {
    gasLimit = nodeGasCeiling
  }
}

if (parentGasLimit < nodeGasCeiling) {
  gasLimit = parentGasLimit + delta
  if (gasLimit > nodeGasCeiling) {
    gasLimit = nodeGasCeiling
  }
}

return gasLimit
```



- Parent Gas Limit / 1024 범위 내에서 변경
  Delta = Parent Gas Limit / 1024 - 1
- Parent Gas Limit 와 Node Gas Ceiling(한계) 를 비교
  - Parent Gas Limit = Node Gas Ceiling ➞ Parent Gas Limit
  - Parent Gas Limit > Node Gas Ceiling ➞ Parent Gas Limit - Delta
  - Parent Gas Limit < Node Gas Ceiling ➞ Parent Gas Limit + Delta

```
parentGasLimit, nodeGasCeiling, gasLimit
delta = (parentGasLimit / 1024) - 1

if (parentGasLimit > nodeGasCeiling) {
  gasLimit = parentGasLimit - delta
  if (gasLimit > nodeGasCeiling) {
    gasLimit = nodeGasCeiling
  }
}

if (parentGasLimit < nodeGasCeiling) {
  gasLimit = parentGasLimit + delta
  if (gasLimit > nodeGasCeiling) {
    gasLimit = nodeGasCeiling
  }
}

return gasLimit
```

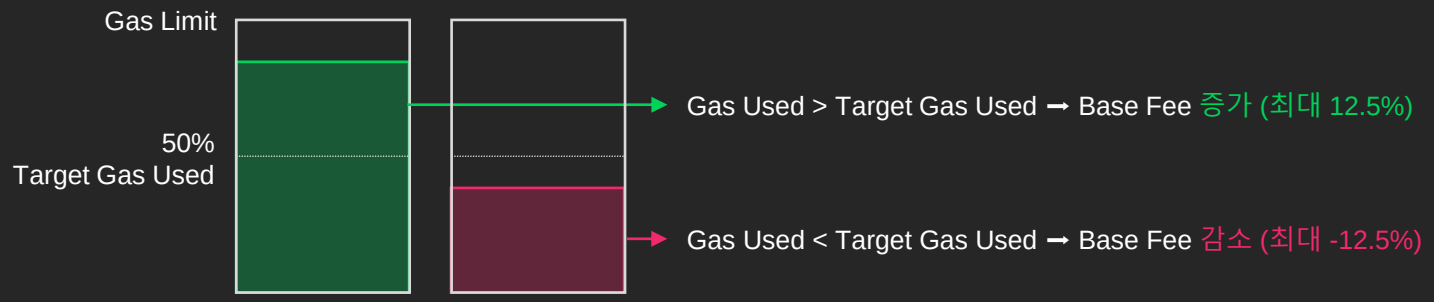
https://github.com/ethereum/go-ethereum/blob/master/params/protocol_params.go#L22

```
// The bound divisor of the gas limit, used in update calculations.
GasLimitBoundDivisor uint64 = 1024
// Minimum the gas limit may ever be.
MinGasLimit          uint64 = 5000
// Maximum the gas limit (2^63-1).
MaxGasLimit          uint64 = 0x7fffffffffffffff
```

https://github.com/ethereum/go-ethereum/blob/master/core/block_validator.go#L108

```
// CalcGasLimit computes the gas limit of the next block after parent. It aims
// to keep the baseline gas close to the provided target, and increase it towards
// the target if the baseline gas is lower.
func CalcGasLimit(parentGasLimit, desiredLimit uint64) uint64 {
   delta := parentGasLimit/params.GasLimitBoundDivisor - 1
   limit := parentGasLimit
   if desiredLimit < params.MinGasLimit {
      desiredLimit = params.MinGasLimit
   }
   // If we're outside our allowed gas range, we try to hone towards them
   if limit < desiredLimit {
      limit = parentGasLimit + delta
      if limit > desiredLimit {
         limit = desiredLimit
      }
      return limit
   }
   if limit > desiredLimit {
      limit = parentGasLimit - delta
      if limit < desiredLimit {
         limit = desiredLimit
      }
   }
   return limit
}
```

- BaseFee = Parent Base Fee * (0.875 ~ 1.125) [ -12.5% ~ +12.5% ]
- Parent Block의 Gas Used에 따라 결정
  Parent Target Gas Used 초과: 증가, Target 미만: 감소




| 항목 | 엑셀 수식 |
|---|---|
| Used | 직접 입력 Etherscan 참고, Gas Used |
| Limit | 직접 입력 Etherscan 참고, Gas Limit |
| Base Fee | 직접 입력 Etherscan 참고 |
| Target | Limit / 2 |
| Usage | ROUNDDOWN((Used / Target) * 100, 2) |
| Change Ratio | (Used – Target) / (Target * 8) |
| Next Base Fee | ROUNDDOWN(Base Fee * (1+Change Ratio), 9) |

$$Target\ Gas\ Used = \frac{Gas\ Limit}{2}$$

$$Change\ Ratio = \frac{Gas\ Used\ - Target\ Gas\ Used}{Target\ Gas\ Used} * \frac{1}{8}$$

$$\text{Next}\ Base\ Fee = Base\ Fee * (1 + Chage\ Ratio)$$

| Number | Used | Limit | Base Fee(Gwei) | Target | Usage | Change Ratio | Next Base Fee |
|---|---|---|---|---|---|---|---|
| 12965000 | 30025257 | 30029122 | 1.000000000 | 15014561 | 199% | 0.124967823 | 1.124967822 |
| 12965001 | 29985144 | 29999798 | 1.124967822 | 14999899 | 199% | 0.124877883 | 1.265451421 |
| 12965002 | 29950343 | 29970503 | 1.265451421 | 14985252 | 199% | 0.124831835 | 1.423420043 |
| 12965003 | 29,933,895 | 29,941,236 | 1.423420054 | 14970618 | 199% | 0.124938705 | 1.601260312 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 14742500 | 28,232,824 | 30,000,000 | 18.433433459 | 15000000 | 188% | 0.110273533 | 20.466153297 |
| 14742501 | 10,889,171 | 30,000,000 | 20.466153297 | 15000000 | 72% | -0.034256908 | 19.765046159 |
| 14742502 | 1,781,880 | 30,000,000 | 19.765046160 | 15000000 | 11% | -0.110151000 | 17.587906560 |
| 14742503 | 29,996,604 | 30,000,000 | 17.587906561 | 15000000 | 199% | 0.124971700 | 19.785897143 |
| 14742504 | 13,582,663 | 30,000,000 | 19.785897143 | 15000000 | 90% | -0.011811142 | 19.552203108 |

# Base Fee

https://github.com/ethereum/go-ethereum/blob/master/params/protocol_params.go#L122

```
// Bounds the amount the base fee can change between blocks.
BaseFeeChangeDenominator = 8

// Bounds the maximum gas limit an EIP-1559 block may have.
ElasticityMultiplier     = 2

// Initial base fee for EIP-1559 blocks.
InitialBaseFee           = 1000000000
```

https://github.com/ethereum/go-ethereum/blob/master/consensus/misc/eip1559.go#L55

```go
// CalcBaseFee calculates the basefee of the header.
func CalcBaseFee(config *params.ChainConfig, parent *types.Header) *big.Int {
   // If the current block is the first EIP-1559 block, return the InitialBaseFee.
   if !config.IsLondon(parent.Number) {
      return new(big.Int).SetUint64(params.InitialBaseFee)
   }

   var (
      parentGasTarget          = parent.GasLimit / params.ElasticityMultiplier
      parentGasTargetBig       = new(big.Int).SetUint64(parentGasTarget)
      baseFeeChangeDenominator = new(big.Int).SetUint64(params.BaseFeeChangeDenominator)
   )
   // If the parent gasUsed is the same as the target, the baseFee remains unchanged.
   if parent.GasUsed == parentGasTarget {
      return new(big.Int).Set(parent.BaseFee)
   }
   if parent.GasUsed > parentGasTarget {
      // If the parent block used more gas than its target, the baseFee should increase.
      gasUsedDelta := new(big.Int).SetUint64(parent.GasUsed - parentGasTarget)
      x := new(big.Int).Mul(parent.BaseFee, gasUsedDelta)
      y := x.Div(x, parentGasTargetBig)
      baseFeeDelta := math.BigMax(
         x.Div(y, baseFeeChangeDenominator),
         common.Big1,
      )

      return x.Add(parent.BaseFee, baseFeeDelta)
   } else {
      // Otherwise if the parent block used less gas than its target, the baseFee should decrease.
      gasUsedDelta := new(big.Int).SetUint64(parentGasTarget - parent.GasUsed)
      x := new(big.Int).Mul(parent.BaseFee, gasUsedDelta)
      y := x.Div(x, parentGasTargetBig)
      baseFeeDelta := x.Div(y, baseFeeChangeDenominator)

      return math.BigMax(
         x.Sub(parent.BaseFee, baseFeeDelta),
         common.Big0,
      )
   }
}
```

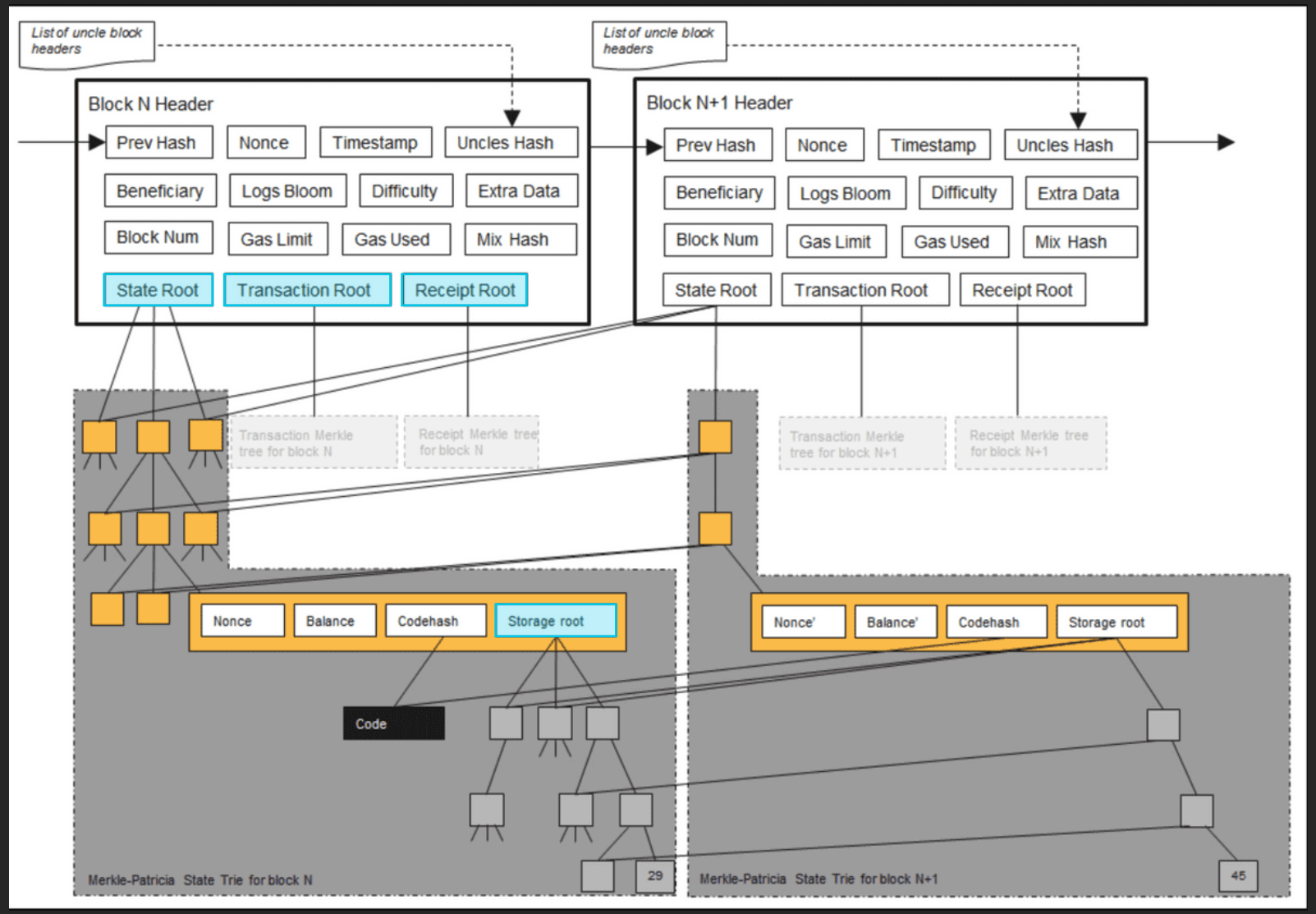
$$Target\ Gas\ Used = \frac{Gas\ Limit}{2}$$

$$Change\ Ratio = \frac{Gas\ Used\ - Target\ Gas\ Used}{Target\ Gas\ Used} * \frac{1}{8}$$

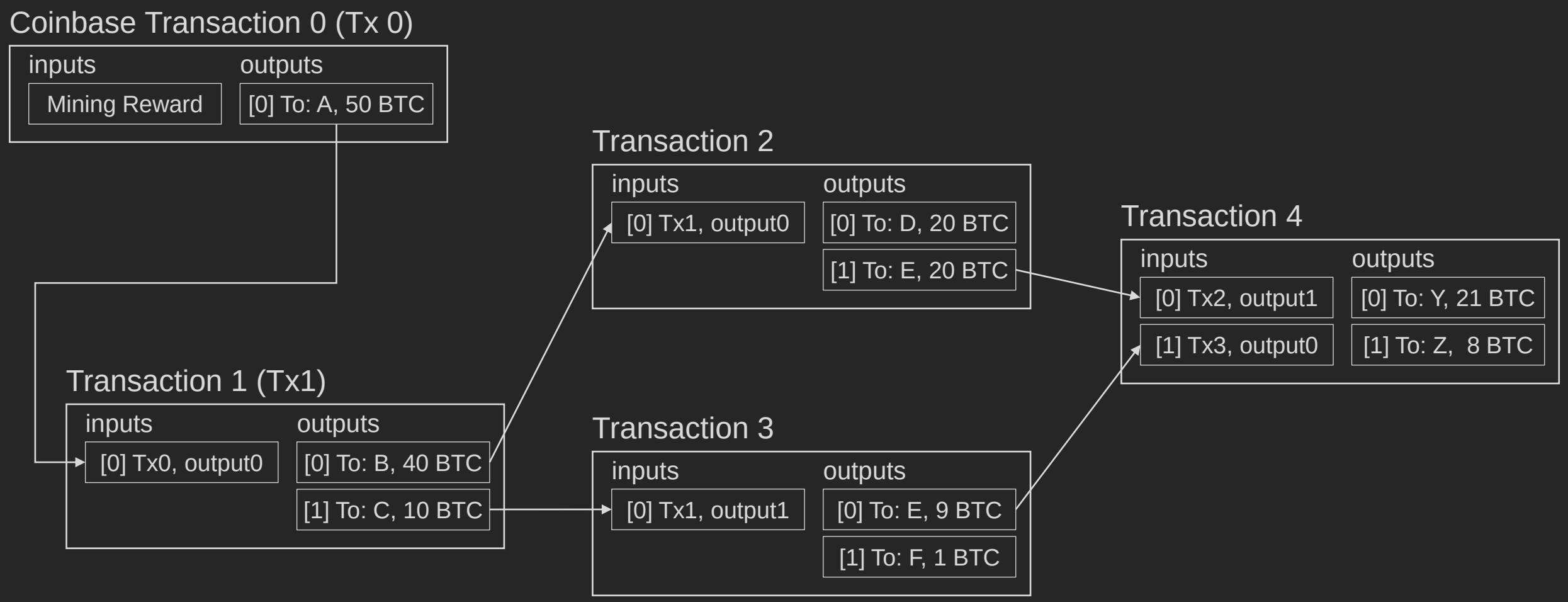
$$\text{Next}\ Base\ Fee = Base\ Fee * (1 + Chage\ Ratio)$$

# 확장 머클 패트리샤 트리 (The modified Merkle Patricia Trie)

## 머클 트리(Merkle Tree)와 패트리샤 트리(Patricia Tree)를 조합 및 확장한 자료구조

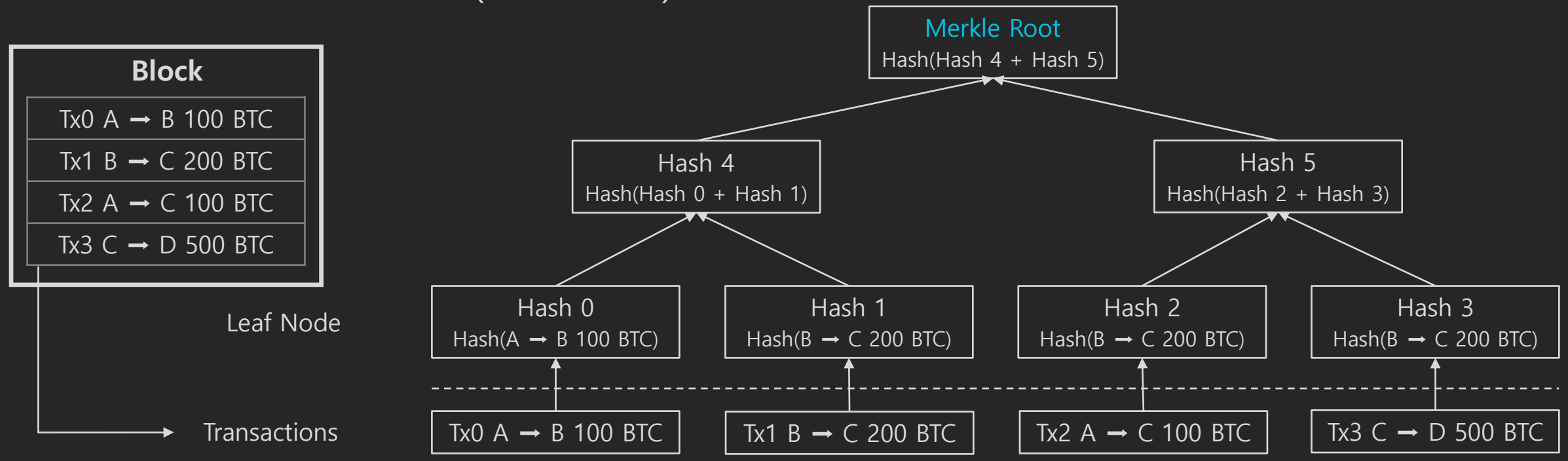


# UTXO (Unspent Transaction Output)



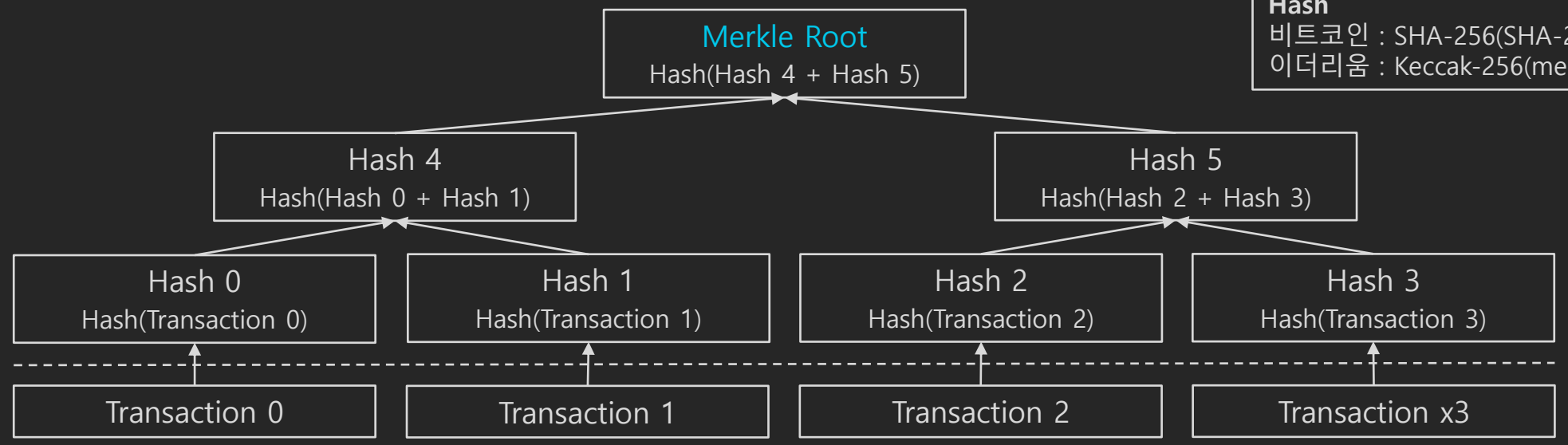
https://patents.google.com/patent/US4309569A/en

- 블록은 트랜잭션(거래)의 묶음
- 머클 트리(Merkle tree) [해시 트리(Hash tree)]
  - 데이터 공간 절약
  - 블록에 포함된 내역을 트리 형태로 요약한 것
  - leaf(잎) 가 아닌 모든 노드가 자식 노드들의 해시로 구성된 트리
  - 1979년 랄프 머클(Ralph Merkle)이 개발
  - 비트코인 : 바이너리 머클 트리 (자식 노드 2개)



# 바이너리 머클 트리 (Binary Merkle Tree)

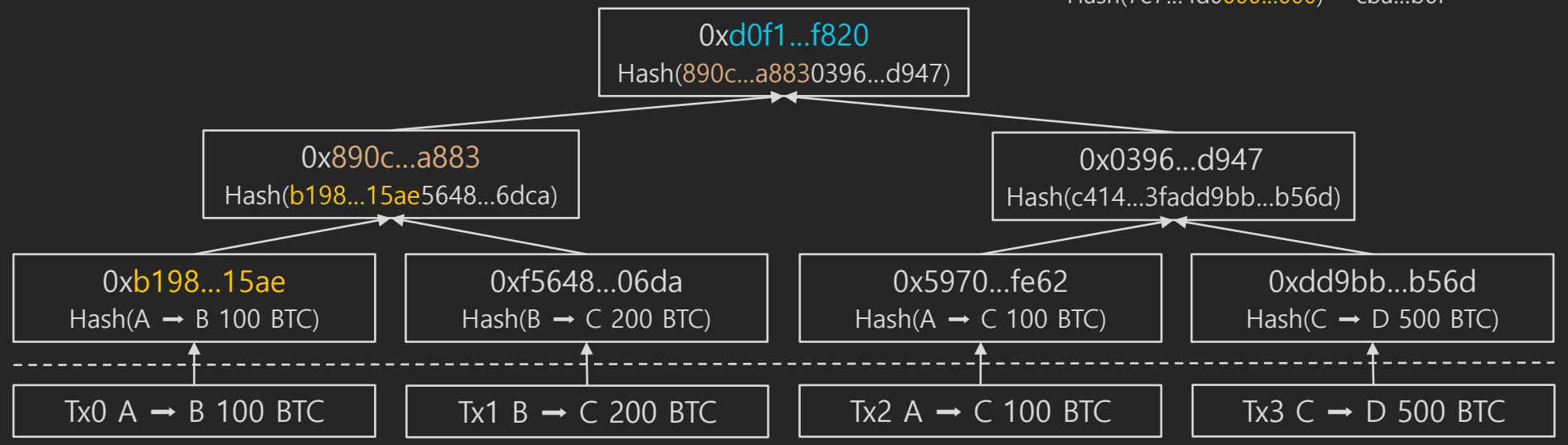
**Hash**
비트코인 : SHA-256(SHA-256(message))
이더리움 : Keccak-256(message)



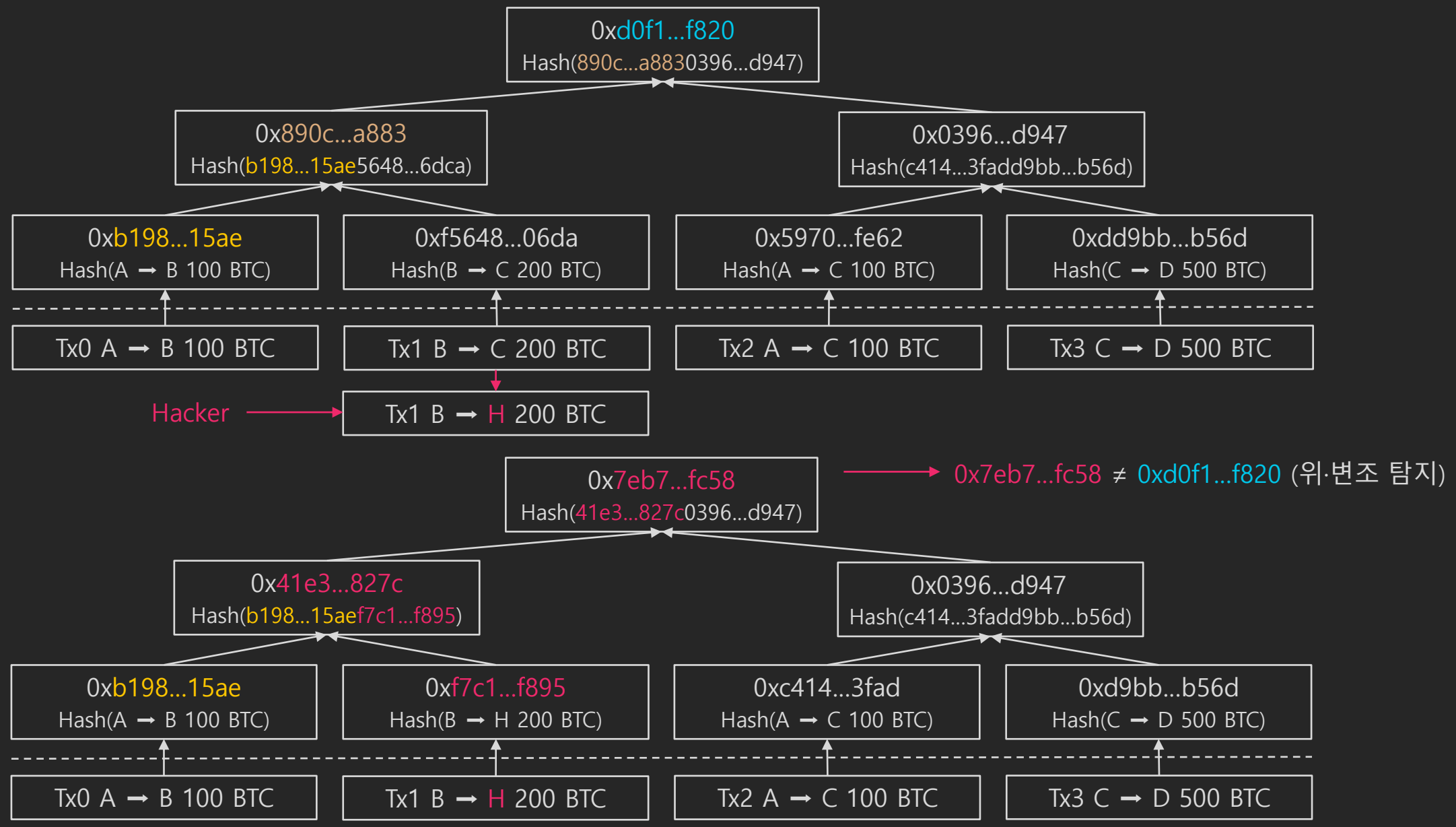

Hash(A → B 100 BTC) = SHA-256(SHA-256(A → B 100 BTC))
= b1986b79781eb8b52d3159e41c73a26565297bf5aaa753c25e9520d81f4b15ae
= b198...15ae

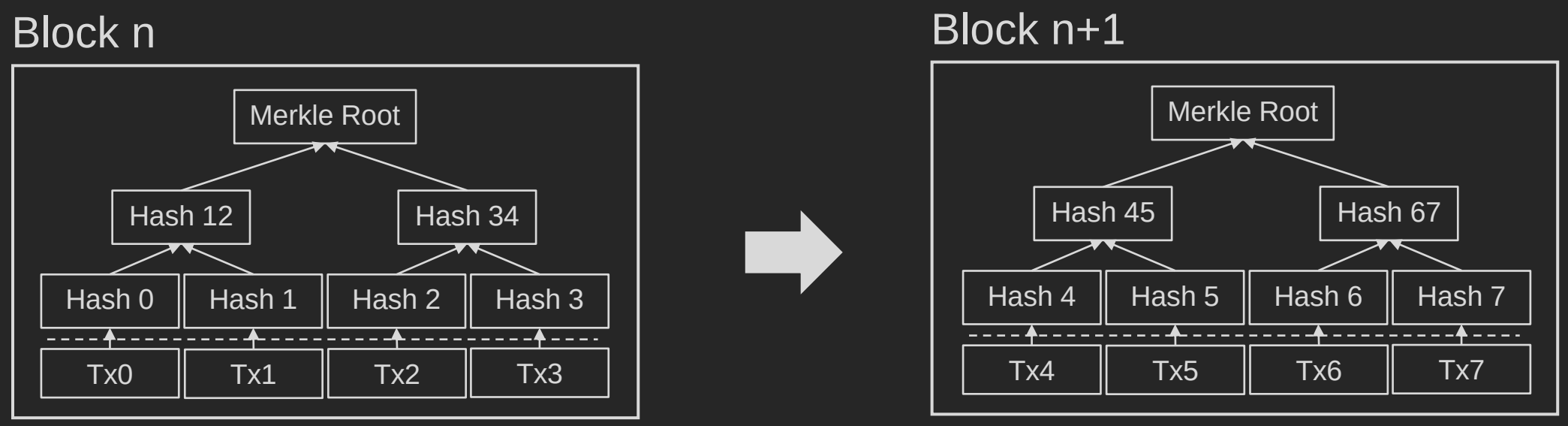
## Bitcoin

※ 마지막 노드의 짝이 없는 경우 0x000...000 추가 후 Hash
Hash(7e7...4d0000...000) = cba...b6f



# 머클 증명 (Merkle Proof) [ 트랜잭션 위·변조 탐지 ]



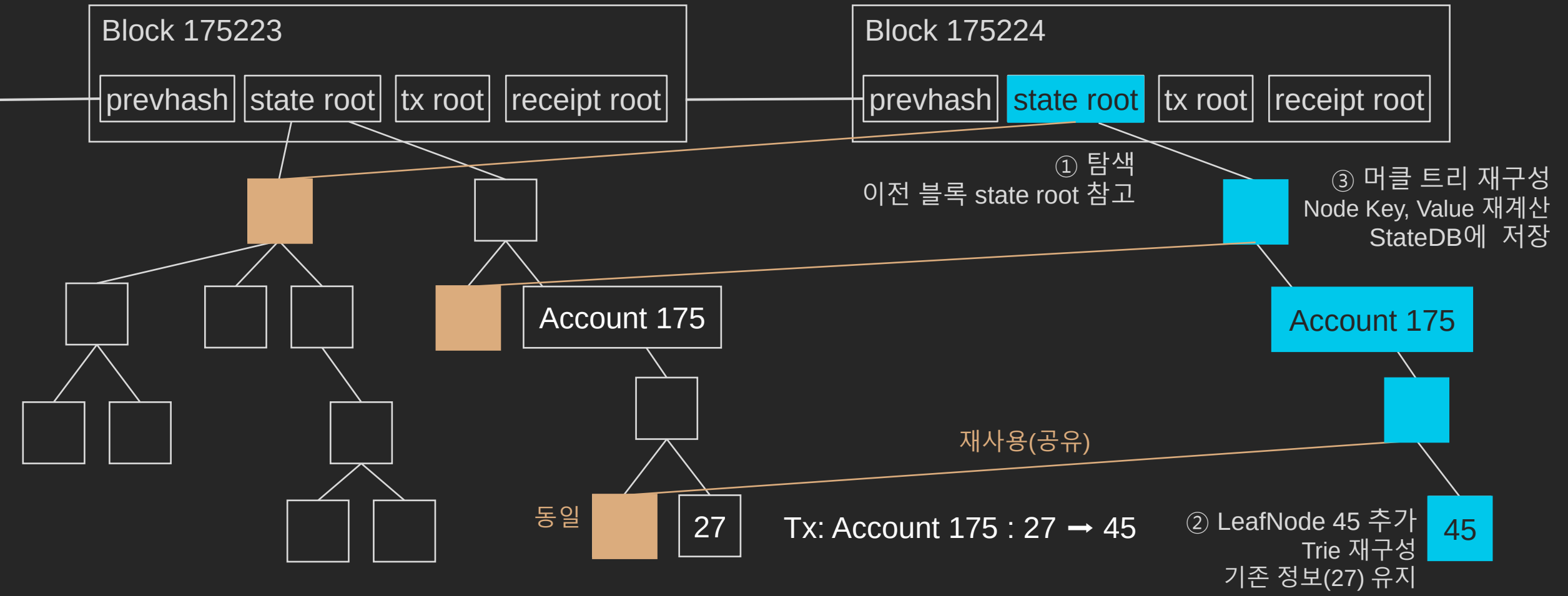



기존 것을 사용 후, 새롭게 생성하여 교체

재사용 하지 않음

https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/

- Non-Binary
- 머클 트리 공유를 통해 저장 공간 절약
  - 상태 전이(state transition) 시, 변경 된 Leaf Node 추가 후 Root 까지 머클 트리 재구성
  - 수정 불필요 머클 트리는 공유 (상태 공유)



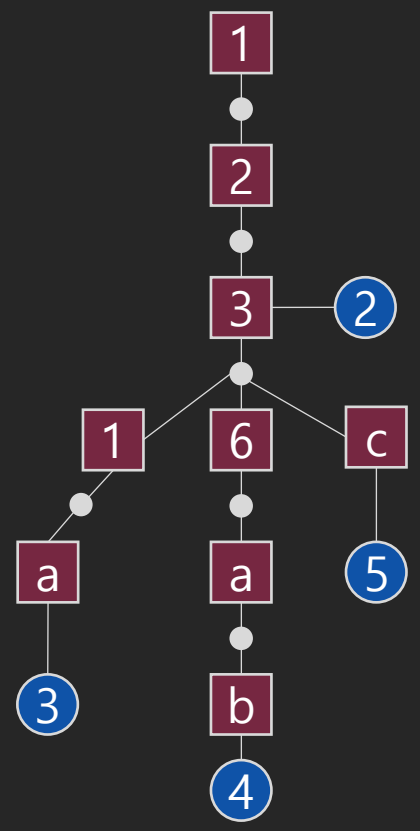
# Trie

- 특정 키 집합의 탐색 트리의 한 유형
- 일반적으로 문자열 저장 및 효율적 탐색을 위한 탐색 트리
- retrieval tree

| Key | Value |
|---|---|
| 123 | 2 |
| 1231a | 3 |
| 1236ab | 4 |
| 123c | 5 |

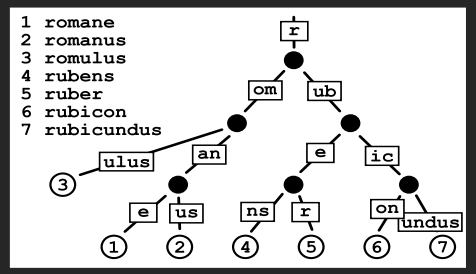
**Key (세로)**

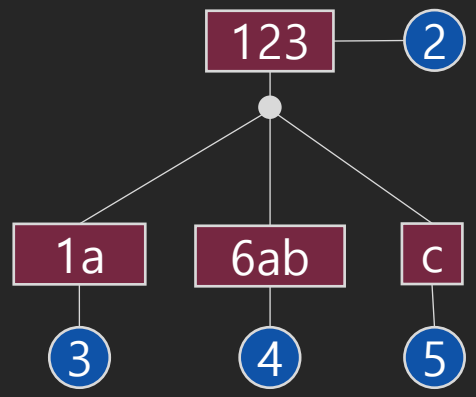
```
1 1 1 1
2 2 2 2
3 3 3 3
  1 6 c
  a a
    b
```



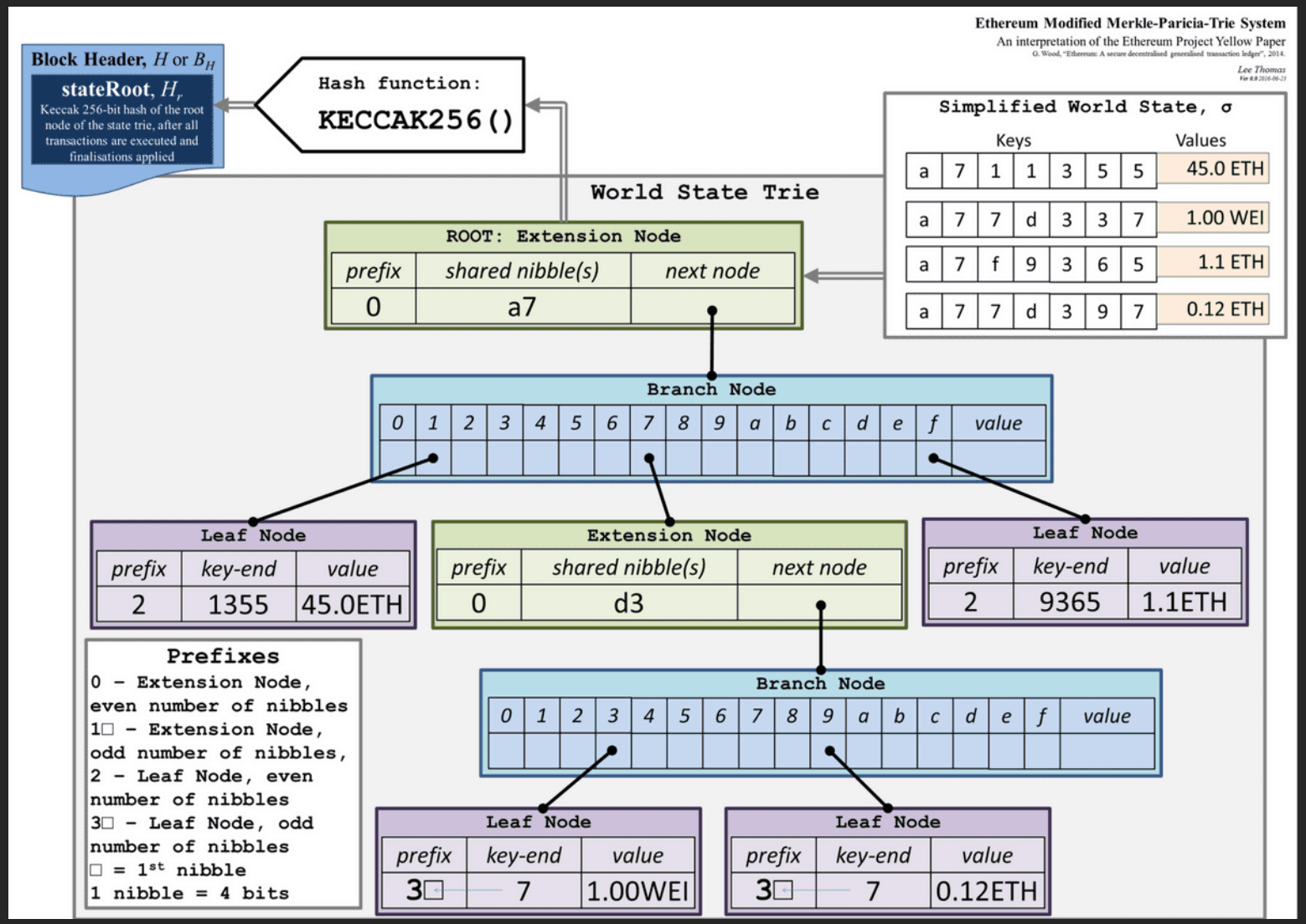

# Patricia Trie

- Radix Tree
- 데이터 저장 시 공간 절약
- 공통부분을 공유하여 공간 절약



- 머클 트리(Merkle Tree)와 패트리샤 트리(Patricia Trie)를 조합 및 확장한 자료구조
- 목적
  - 상태 Key-Value 의 효율적인 탐색 및 관리
  - 동일 머클 트리 재사용(공유)
  - 거래 정보 검증 (Merkle Proof)
- Path
  - Key ➡ Path 전환 : Trie별 정의
    예) StateTrie: keccak-256(address)
  - nibble(4 bits) 당 path 1개
- Node
  - ExtensionNode : 공통 Path 압축
  - BranchNode : Path 분기
  - LeafNode : Value 저장
- Node는 DB에 (Key, Value)로 저장
  - Storage Key = keccak-256(Storage Value)
  - Storage Value = RLP(Node)



- **Extension Node** : 공통 path 압축 노드, [ encodedPath , value ]

| encodedPath | value |
|---|---|
| HP Encoding(Extension, shared path) | Branch Node Key |

- **Branch Node** : 분기 노드, [ $i_0$, $i_1$, …, $i_{15}$, value ] (17)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | 현재 path의 Value |

index와 path 1개가 매칭되는 Next Node Key 또는 Null Node

- **Leaf Node** : Value 저장 노드, [ encodedPath , value ]

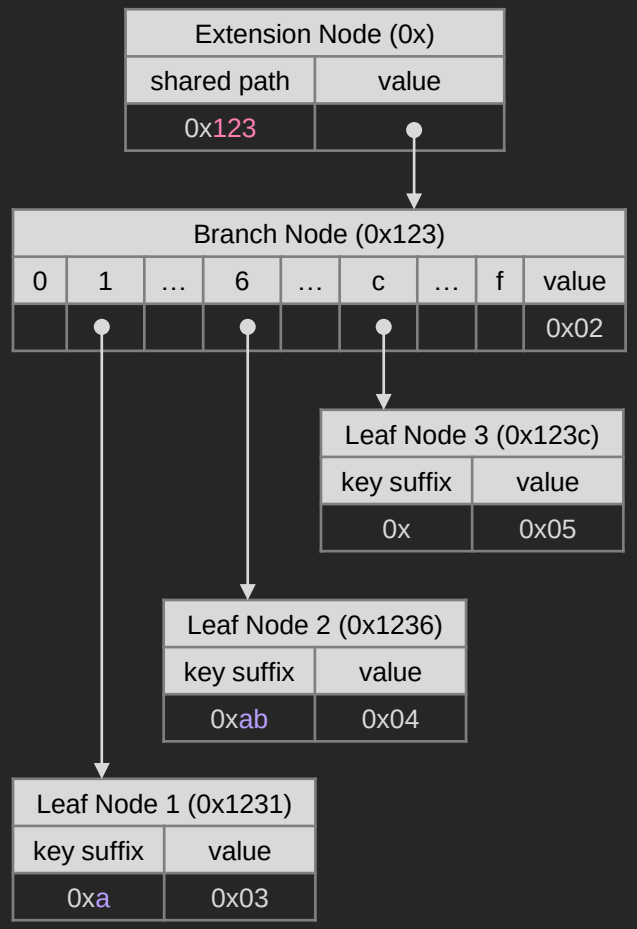
| encodedPath | value |
|---|---|
| HP Encoding(Leaf, key suffix) | Value |

- Null Node : "" (빈 문자열)

| Key | Value |
|---|---|
| 0x123 | 0x02 |
| 0x1231a | 0x03 |
| 0x1236ab | 0x04 |
| 0x123c | 0x05 |

Key (세로)
1 1 1 1
2 2 2 2
3 3 3 3
1 6 c
a a
b

HP Encoding 제외



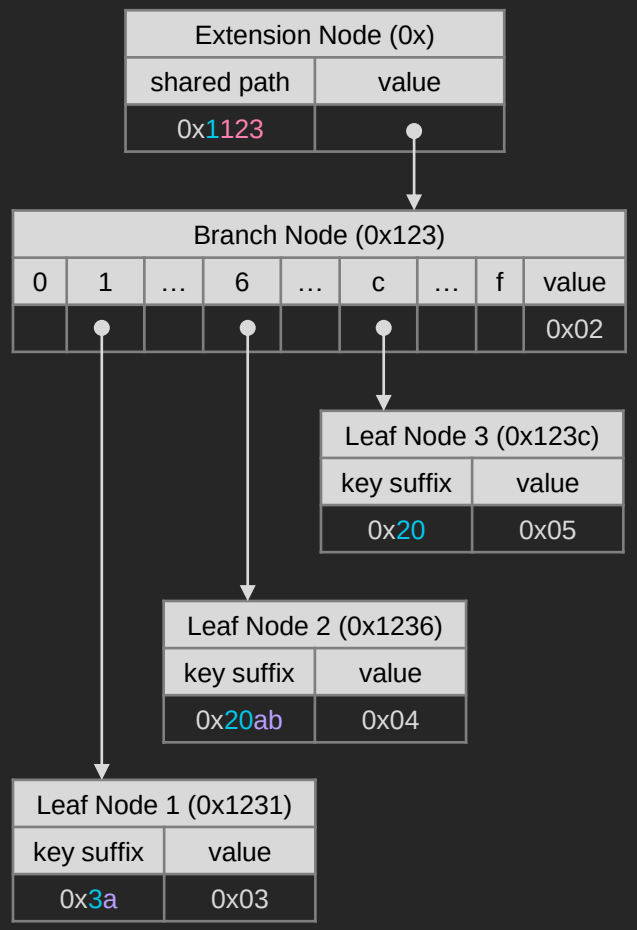
- HP Encoding (Hex-Prefix Encoding)
  - Extension Node와 Leaf Node의 path Encoding
- 목적
  - Extension Node 와 Leaf Node 구분
  - nibble (4 bits) 시퀀스를 byte (8 byte) 시퀀스로 변환
    path 길이가 홀수 인 경우 path의 첫번째 nibble을 prefix에 append (1byte 절약)

| Node | Format |
|---|---|
| Extension Node | [ encodedPath , value ] |
| Branch Node | [ i0, i1, …, i15, value ] |
| Leaf Node | [ encodedPath , value ] |

| Hex | Bits | Node Type | Path 길이 | HP Prefix | 예제 |
|---|---|---|---|---|---|
| 0 | 0000 | Extension | 짝수(even) | 0x00 | HP(Extension, 0x1234) = 0x001234 |
| 1 | 0001 | Extension | 홀수(odd) | 0x1□ | HP(Extension, 0x12345) = 0x112345 |
| 2 | 0010 | Leaf | 짝수(even) | 0x20 | HP( Leaf, 0x1234) = 0x201234 |
| 3 | 0011 | Leaf | 홀수(odd) | 0x3□ | HP( Leaf, 0x12345) = 0x312345 |

| Key | Value |
|---|---|
| 0x123 | 0x02 |
| 0x1231a | 0x03 |
| 0x1236ab | 0x04 |
| 0x123c | 0x05 |

Key (세로)
1 1 1 1
2 2 2 2
3 3 3 3
1 6 c
a a
b



- Node는 DB에 (Node Key, Node Value) pair로 저장
  - Node Key = keccak-256 ( Node Value )
  - Node Value = RLP ( Node )

| Node | Format |
|---|---|
| Extension Node | [ encodedPath , value ] |
| Branch Node | [ i0, i1, …, i15, value ] |
| Leaf Node | [ encodedPath , value ] |

| Prefix | Node Type | Path 길이 |
|---|---|---|
| 0x00 | Extension | 짝수 |
| 0x1□ | Extension | 홀수 |
| 0x20 | Leaf | 짝수 |
| 0x3□ | Leaf | 홀수 |

1.Key로 Trie 구성

2.Leaf 부터 머클 트리 계산

| Key | Value |
|---|---|
| 0x123 | 0x02 |
| 0x1231a | 0x03 |
| 0x1236ab | 0x04 |
| 0x123c | 0x05 |

**Key (세로)**

```
1 1 1 1
2 2 2 2
3 3 3 3
  1 6 c
  a a
    b
```

①

| Extension Node (0x) | |
|---|---|
| shared path | value |
| 0x1123 | ⑪ 0x9e…a5 |

⑫ Key = 0x2d1cabee0ff5f59ec746028747d908e7f651c416b63c3d1facaf28cb46d925d9 **(Merkle Root)**

Value

```
= RLP([01123,0x9e21a34e4349dcd5eccddd1f99eb07ceb3ab97ce15789baee9eec7b14a9c32a5])
=  0xe4821123a09e21a34e4349dcd5eccddd1f99eb07ceb3ab97ce15789baee9eec7b14a9c32a5
```

②

| Branch Node (0x123) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | ⑨ 1 | … | ⑨ 6 | … | ⑨ c | … | f | value |
| | 0x9a…38 | | 0x1c…c5 | | 0xd2…c0 | | | 0x02 |

⑩ Key = 0x9e21a34e4349dcd5eccddd1f99eb07ceb3ab97ce15789baee9eec7b14a9c32a5

Value

```
= RLP([0x,0x9a…38,0x,0x,0x,0x,0x1c…c5,0x,0x,0x,0x,0x,0xd2…c0,0x,0x,0x,0x02 ])
= 0xf87180a09ad2b2688efb6abe61e1d4af3464eac90dd5cb1ea4114cfc29a85a2c6b84803880
808080a01ced0505978bc31e134f58ef3e8a15bf089f031921ddb3a3533c6f97e2572fc5808080
8080a0d21b3a1ada9987e50e2b4dc805be4bea8ce445e56f20ff1536188edb1ef28bc080808002
```

⑤

| Leaf Node 3 (0x123c) | |
|---|---|
| key suffix | value |
| 0x20 | 0x05 |

⑧ Key = 0xd21b3a1ada9987e50e2b4dc805be4bea8ce445e56f20ff1536188edb1ef28bc0

Value = RLP( [ 0x20, 0x05 ] ) = 0xc22005

④

| Leaf Node 2 (0x1236) | |
|---|---|
| key suffix | value |
| 0x20ab | 0x04 |

⑦ Key = 0x1ced0505978bc31e134f58ef3e8a15bf089f031921ddb3a3533c6f97e2572fc5

Value = RLP( [ 0x20ab, 0x04 ] ) = 0xc48220ab04

③

| Leaf Node 1 (0x1231) | |
|---|---|
| key suffix | value |
| 0x3a | 0x03 |

⑥ Key = 0x9ad2b2688efb6abe61e1d4af3464eac90dd5cb1ea4114cfc29a85a2c6b848038

Value = RLP( [ 0x3a, 0x03 ] ) = 0xc23a03

1. (Account) State Trie
   Key = account address
   Path = keccak-256(address)

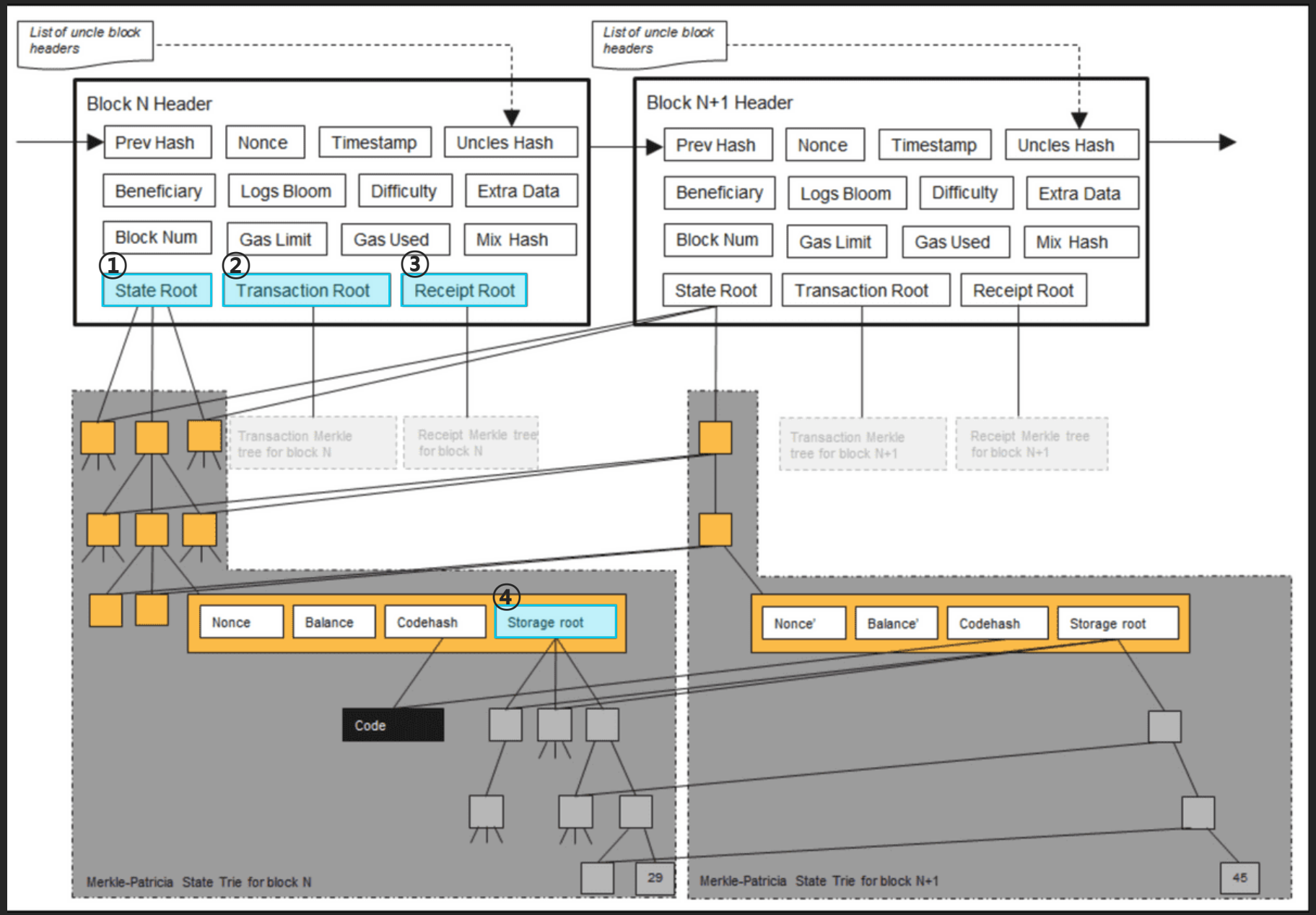
2. CA Storage Trie
   Key :
   - static : position
   - dynamic : key + position
   Path = Keccak-256( Key )

3. Block Transactions Trie
   Key = Transaction index
   Path = RLP(tx index)

4. Block Receipts Trie
   Key = Receipt Index
   Path = RLP(receipt index)

Path 최대 길이 : 64

256 bits = 64 nibbles



1. (Account) State Trie
   - Key : address → Path : keccak-256(address)
   - Value : RLP [ nonce, balance, storageHash, codeHash ]
2. CA Storage Trie (position : 변수 정의 순서에 따라 결정)
   - Key : static = position, dynamic : key || position → Path : keccak-256( Key )
   - Value : RLP (value)
3. Transactions Trie
   - Key : index of transaction → Path : RLP( index of transaction )
   - Value
     - Leagcy : RLP [nonce, gasPrice, gasLimit, to, value, data, v, r, s]
     - Type 1: 0x01 || RLP [chainId, nonce, gasPrice, gasLimit, to, value, data, accessList, signatureYParity, signatureR, signatureS]
     - Type 2: 0x02 || RLP [chainId, nonce, maxPriorityFeePerGas, maxFeePerGas, gasLimit, destination, amount, data, accessList, signatureYParity, signatureR, signatureS]
4. Receipts Trie
   - Path : RLP( index of receipt ) → Key : index of receipt
   - Value
     - Legacy: RLP(RECEIPT)
     - Type 1 : 0x01+RLP(RECEIPT)
     - Type 2 : 0x02+RLP(RECEIPT)

RECEIPT = [ status, cumulativeGasUsed, bloomFilter, LOGS ]
LOGS = [ logger, TOPICS, data ]
TOPICS = [{32bytes}] (MAX: 4)

## CA Storage Trie (position : 변수 정의 순서에 따라 결정)

- Key : static = position, dynamic : key || position
- Value : value
- Path : keccak-256( Key )

```
contract ContractPosition {
    uint256 position0 = 1;
    address position1 = 0x942F397B7f4391B43115395F469c63072aEd6E41;
    mapping(address => uint256) private position2;
    uint256[3] position345 = [2,3,4];
}
```

**[0] uint256 position0 = 1;**

```
Key: 0
Path: keccak-256(0x0000000000000000000000000000000000000000000000000000000000000000)
Value : 1
```

**[1] address position1 = 0x942F397B7f4391B43115395F469c63072aEd6E41;**

```
Key: 1
Path: keccak-256(0x0000000000000000000000000000000000000000000000000000000000000001)
Value: 0x942f397b7f4391b43115395f469c63072aed6e41
```

**[2] mapping(address => uint256) private position2; → position2[0x942F397B7f4391B43115395F469c63072aEd6E41]**

```
Key:address + 2
Path:keccak-256
(0x000000000000000000000000942F397B7f4391B43115395F469c63072aEd6E41000000000000000000000000000000000000000000000000000000000
```

# Difficulty

$$\text{diff} = \text{parent diff} + \left(\frac{\text{parent diff}}{2048} * \max\left((1\ or\ 2) - \frac{(block\ timestamp\ - parent\ timestamp)}{9}, -99\right)\right)$$

- 난이도 조정 단위 : 부모 난이도 / 2048
- (1 or 2) : 부모 블록의 Uncle Block 포함 여부
  - 1 : 미포함
  - 2 : 포함 (개수 상관없이 블록 하나 생성으로 간주)
- 난이도 조정 (Uncle 미포함 가정)
  - 9초 미만 : 난이도 증가
  - 18초 이상 : 9초당 난이도 1단계씩 감소 (최대 99단계로 제한)
  - (9 + 18) / 2 = 13.5
- 난이도 폭탄 (Difficulty Bomb) $diff = diff + 2^{\left(\left(\frac{parent\ block\ numer\ -\ delay}{100000}\right) - 2\right)}$
  - Arrow Glacier delay: 10,700,000
  - Block Number(2022-05-19) : 14,799,950 ➡ diff + $2^{38}$ ➡ diif = diif + 137,438,953,472 (난이도 조정 단위의 1.9% 추가 상승)
  - Current Difficulty : 14,371,129,481,582,148

# Difficulty

https://github.com/ethereum/go-ethereum/blob/master/consensus/ethash/consensus.go#L367

```go
// makeDifficultyCalculator creates a difficultyCalculator with the given bomb-delay.
// the difficulty is calculated with Byzantium rules, which differs from Homestead in how uncles affect the calculation
func makeDifficultyCalculator(bombDelay *big.Int) func(time uint64, parent *types.Header) *big.Int {
   // Note, the calculations below looks at the parent number, which is 1 below
   // the block number. Thus we remove one from the delay given
   bombDelayFromParent := new(big.Int).Sub(bombDelay, big1)
   return func(time uint64, parent *types.Header) *big.Int {
      // https://github.com/ethereum/EIPs/issues/100.
      // algorithm:
      // diff = (parent_diff +
      //         (parent_diff / 2048 * max((2 if len(parent.uncles) else 1) - ((timestamp - parent.timestamp) // 9), -99))
      //        ) + 2^(periodCount - 2)

      bigTime := new(big.Int).SetUint64(time)
      bigParentTime := new(big.Int).SetUint64(parent.Time)

      // holds intermediate values to make the algo easier to read & audit
      x := new(big.Int)
      y := new(big.Int)

      // (2 if len(parent_uncles) else 1) - (block_timestamp - parent_timestamp) // 9
      x.Sub(bigTime, bigParentTime)
      x.Div(x, big9)
      if parent.UncleHash == types.EmptyUncleHash {
         x.Sub(big1, x)
      } else {
         x.Sub(big2, x)
      }
      // max((2 if len(parent_uncles) else 1) - (block_timestamp - parent_timestamp) // 9, -99)
      if x.Cmp(bigMinus99) < 0 {
         x.Set(bigMinus99)
      }

      // parent_diff +
      // (parent_diff / 2048 *
      //  max((2 if len(parent.uncles) else 1) - ((timestamp - parent.timestamp) // 9), -99))
      y.Div(parent.Difficulty, params.DifficultyBoundDivisor)
      x.Mul(y, x)
      x.Add(parent.Difficulty, x)

     // minimum difficulty can ever be (before exponential factor)
      if x.Cmp(params.MinimumDifficulty) < 0 {
         x.Set(params.MinimumDifficulty)
      }
```

(1 or 2) : 부모 블록의 uncle 존재 여부
미존재 : 1, 존재: 2

$$\text{parent diff} + \left(\frac{\text{parent diff}}{2048} * \max\left((1\ or\ 2) - \frac{(block\ timestamp\ - parent\ timestamp)}{9},\quad -99\right)\right)$$

Using Parent Uncle Block

Difficulty Bomb Delay

```go
      // calculate a fake block number for the ice-age delay
      // Specification: https://eips.ethereum.org/EIPS/eip-1234
      fakeBlockNumber := new(big.Int)
      if parent.Number.Cmp(bombDelayFromParent) >= 0 {
         fakeBlockNumber = fakeBlockNumber.Sub(parent.Number, bombDelayFromParent)
      }
      // for the exponential factor
      periodCount := fakeBlockNumber
      periodCount.Div(periodCount, expDiffPeriod)

      // the exponential factor, commonly referred to as "the bomb"
      // diff = diff + 2^(periodCount - 2)
      if periodCount.Cmp(big1) > 0 {
         y.Sub(periodCount, big2)
         y.Exp(big2, y, nil)
         x.Add(x, y)
      }
      return x
   }
}
```

expDiffPeriod = 100000

Difficulty Bomb

$$diff + 2^{\left(\left(\frac{parent\ block\ numer\ - delay}{100000}\right) - 2\right)}$$

# MixHash, Nonce

https://github.com/ethereum/go-ethereum/blob/master/consensus/ethash/sealer.go#L132

https://github.com/ethereum/go-ethereum/blob/master/consensus/ethash/ethash.go#L47

```go
var (
   // two256 is a big integer representing 2^256
   two256 = new(big.Int).Exp(big.NewInt(2), big.NewInt(256), big.NewInt(0))
)
```

```go
// mine is the actual proof-of-work miner that searches for a nonce starting
from
// seed that results in correct final block difficulty.
func (ethash *Ethash) mine(block *types.Block, id int, seed uint64, abort chan
struct{}, found chan *types.Block) {
   // Extract some data from the header
   var (
      header  = block.Header()
      hash    = ethash.SealHash(header).Bytes()
      target  = new(big.Int).Div(two256, header.Difficulty)
      number  = header.Number.Uint64()
      dataset = ethash.dataset(number, false)
   )
   // Start generating random nonces until we abort or find a good one
   var (
      attempts  = int64(0)
      nonce     = seed
      powBuffer = new(big.Int)
   )
   …
```

Target = $2^{256}$ / Difficulty

```go
search:
   for {
      select {
      case <-abort:
         …
      default:
         // We don't have to update hash rate on every nonce,
         // so update after after 2^X nonces
         attempts++
         if (attempts % (1 << 15)) == 0 {
            ethash.hashrate.Mark(attempts)
            attempts = 0
         }
         // Compute the PoW value of this nonce
         digest, result := hashimotoFull(dataset.dataset, hash, nonce)
         if powBuffer.SetBytes(result).Cmp(target) <= 0 {
            // Correct nonce found, create a new header with it
            header = types.CopyHeader(header)
            header.Nonce = types.EncodeNonce(nonce)
            header.MixDigest = common.BytesToHash(digest)

            // Seal and return a block (if still needed)
            select {
            case found <- block.WithSeal(header):
               logger.Trace("Ethash nonce found and reported",
                  "attempts", nonce-seed, "nonce", nonce)
            case <-abort:
               logger.Trace("Ethash nonce found but discarded",
                  "attempts", nonce-seed, "nonce", nonce)
            }
            break search
         }
         nonce++
      }
```

Find Nonce

(mixHash, nonce)

nonce

mixHash

found

nonce++

# MixHash, Nonce

```go
// mine is the actual proof-of-work miner that searches for a nonce starting
from
// seed that results in correct final block difficulty.
func (ethash *Ethash) mine(block *types.Block, id int, seed uint64, abort chan
struct{}, found chan *types.Block) {
   // Extract some data from the header
   var (
      header  = block.Header()
      hash    = ethash.SealHash(header).Bytes()
      target  = new(big.Int).Div(two256, header.Difficulty)
      number  = header.Number.Uint64()
      dataset = ethash.dataset(number, false)
   )
   // Start generating random nonces until we abort or find a good one
   var (
      attempts  = int64(0)
      nonce     = seed
      powBuffer = new(big.Int)
   )
   …
```

Target

```go
search:
   for {
      select {
      case <-abort:
         …
      default:
         // We don't have to update hash rate on every nonce,
         // so update after after 2^X nonces
         attempts++
         if (attempts % (1 << 15)) == 0 {
            ethash.hashrate.Mark(attempts)
            attempts = 0
         }
         // Compute the PoW value of this nonce
         digest, result := hashimotoFull(dataset.dataset, hash, nonce)
         if powBuffer.SetBytes(result).Cmp(target) <= 0 {
            // Correct nonce found, create a new header with it
            header = types.CopyHeader(header)
            header.Nonce = types.EncodeNonce(nonce)
            header.MixDigest = common.BytesToHash(digest)

            // Seal and return a block (if still needed)
            select {
            case found <- block.WithSeal(header):
               logger.Trace("Ethash nonce found and reported",
                  "attempts", nonce-seed, "nonce", nonce)
            case <-abort:
               logger.Trace("Ethash nonce found but discarded",
                  "attempts", nonce-seed, "nonce", nonce)
            }
            break search
         }
         nonce++
      }
```

Find Nonce

(mixHash, nonce)

nonce

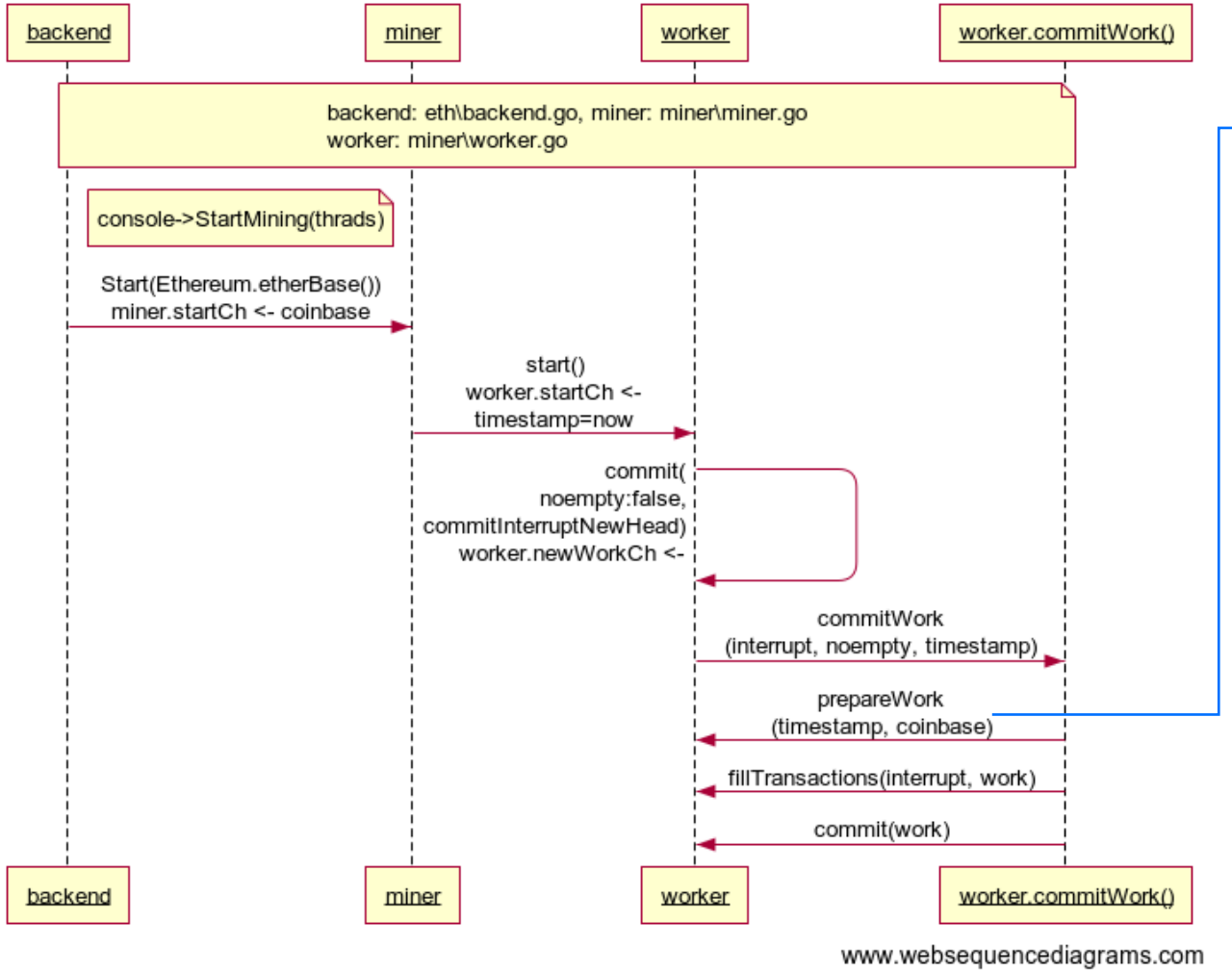
mixHash

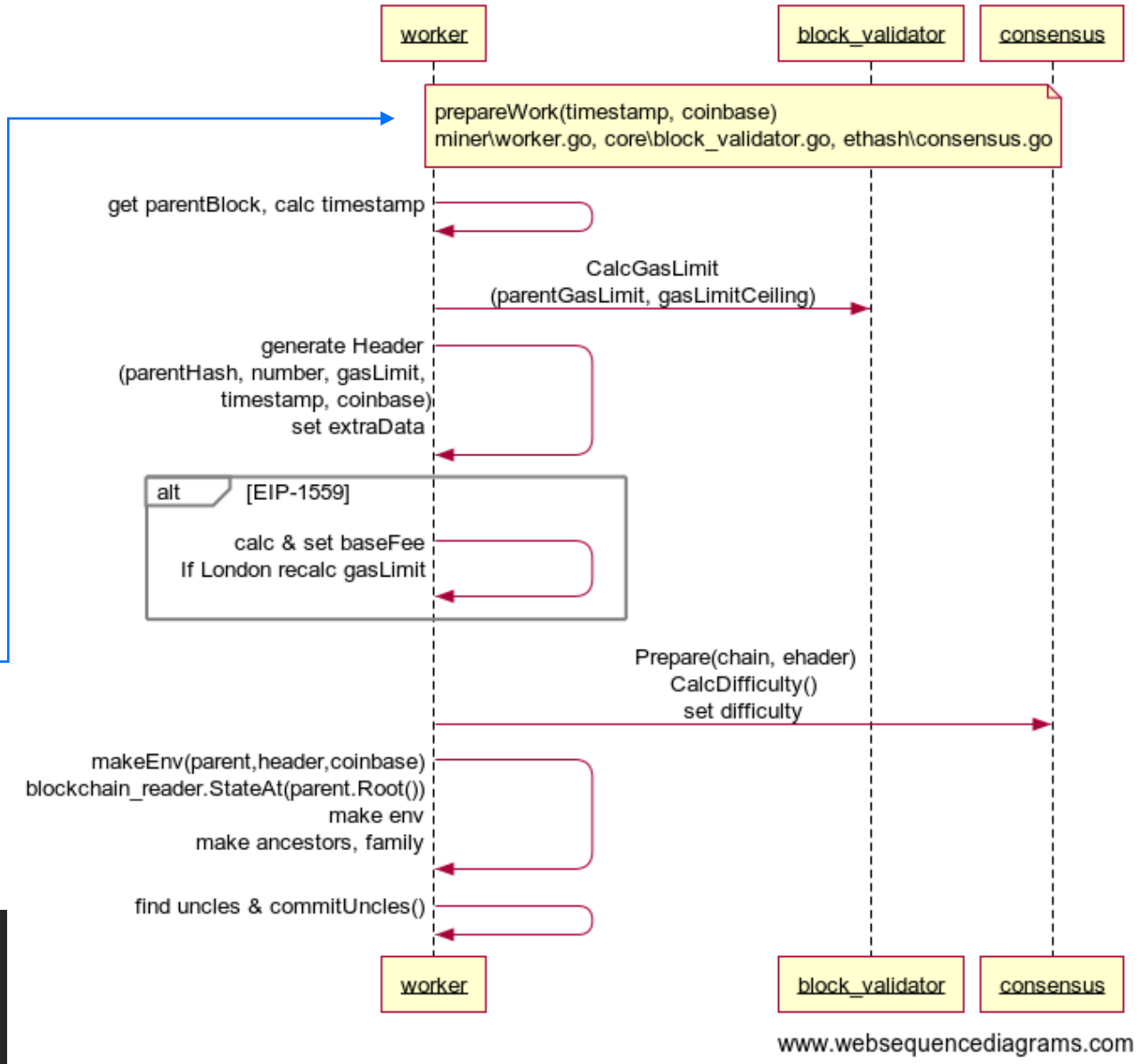
nonce++

https://github.com/ethereum/go-ethereum/blob/master/core/types/block.go#L199

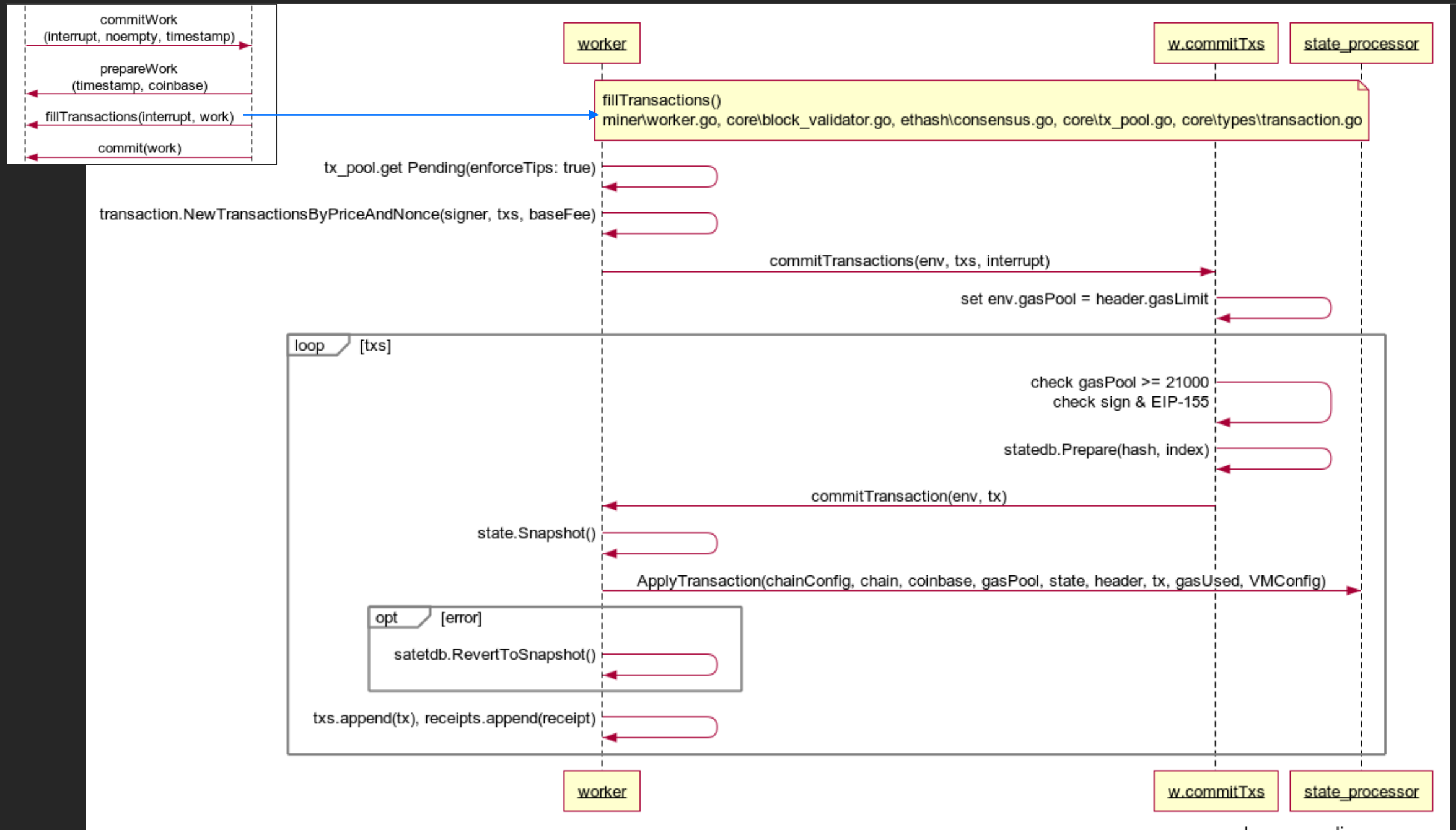
```go
// NewBlock creates a new block. The input data is copied,
// changes to header and to the field values will not affect the
// block.
//
// The values of TxHash, UncleHash, ReceiptHash and Bloom in header
// are ignored and set to values derived from the given txs, uncles
// and receipts.
func NewBlock(header *Header, txs []*Transaction, uncles []*Header, receipts []*Receipt, hasher
TrieHasher) *Block {
   b := &Block{header: CopyHeader(header), td: new(big.Int)}

   // TODO: panic if len(txs) != len(receipts)
   if len(txs) == 0 {
      b.header.TxHash = EmptyRootHash
   } else {
      b.header.TxHash = DeriveSha(Transactions(txs), hasher)
      b.transactions = make(Transactions, len(txs))
      copy(b.transactions, txs)
   }

   if len(receipts) == 0 {
      b.header.ReceiptHash = EmptyRootHash
   } else {
      b.header.ReceiptHash = DeriveSha(Receipts(receipts), hasher)
      b.header.Bloom = CreateBloom(receipts)
   }

   if len(uncles) == 0 {
      b.header.UncleHash = EmptyUncleHash
   } else {
      b.header.UncleHash = CalcUncleHash(uncles)
      b.uncles = make([]*Header, len(uncles))
      for i := range uncles {
         b.uncles[i] = CopyHeader(uncles[i])
      }
   }

   return b
}
```

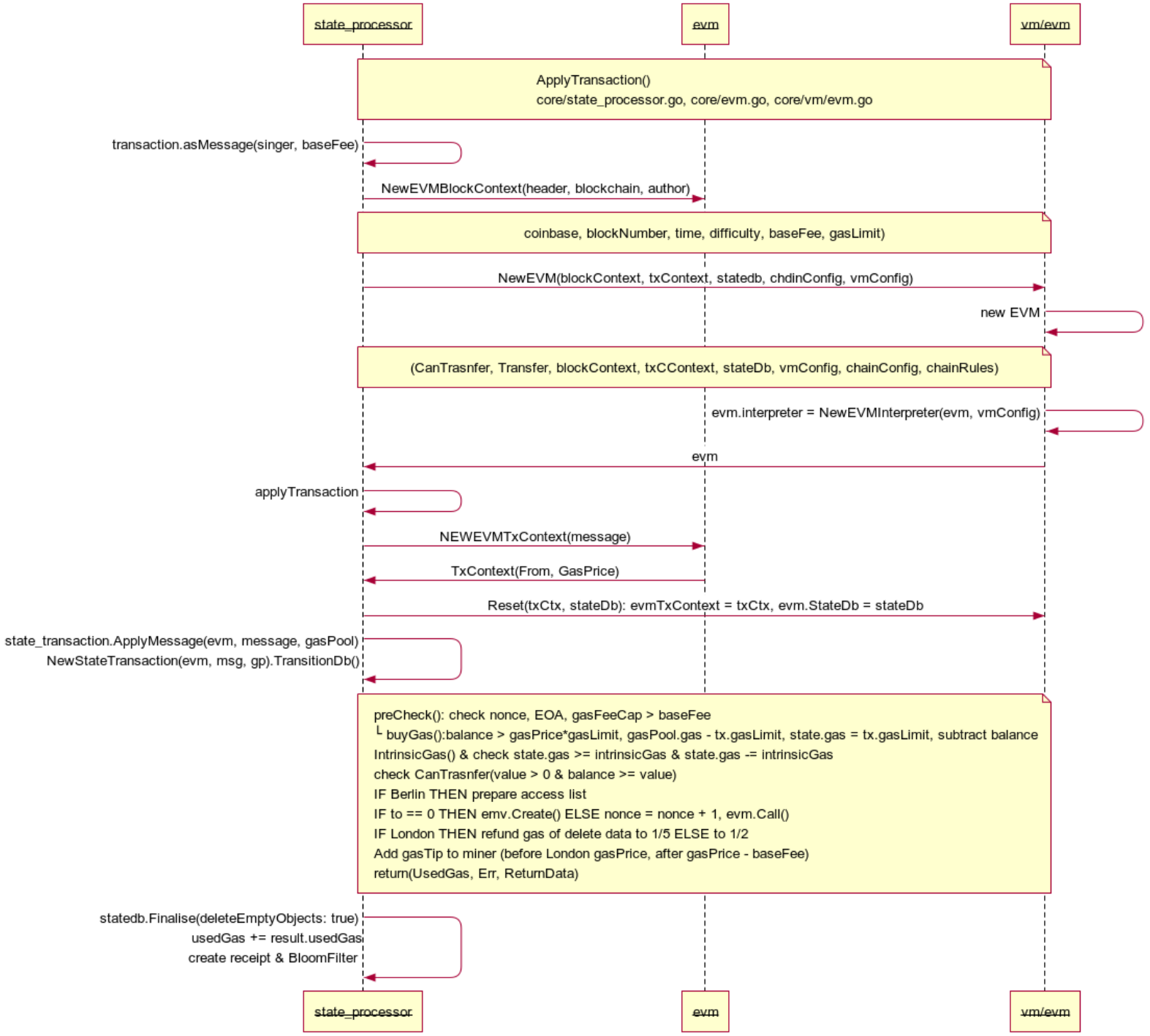
# Block Mining



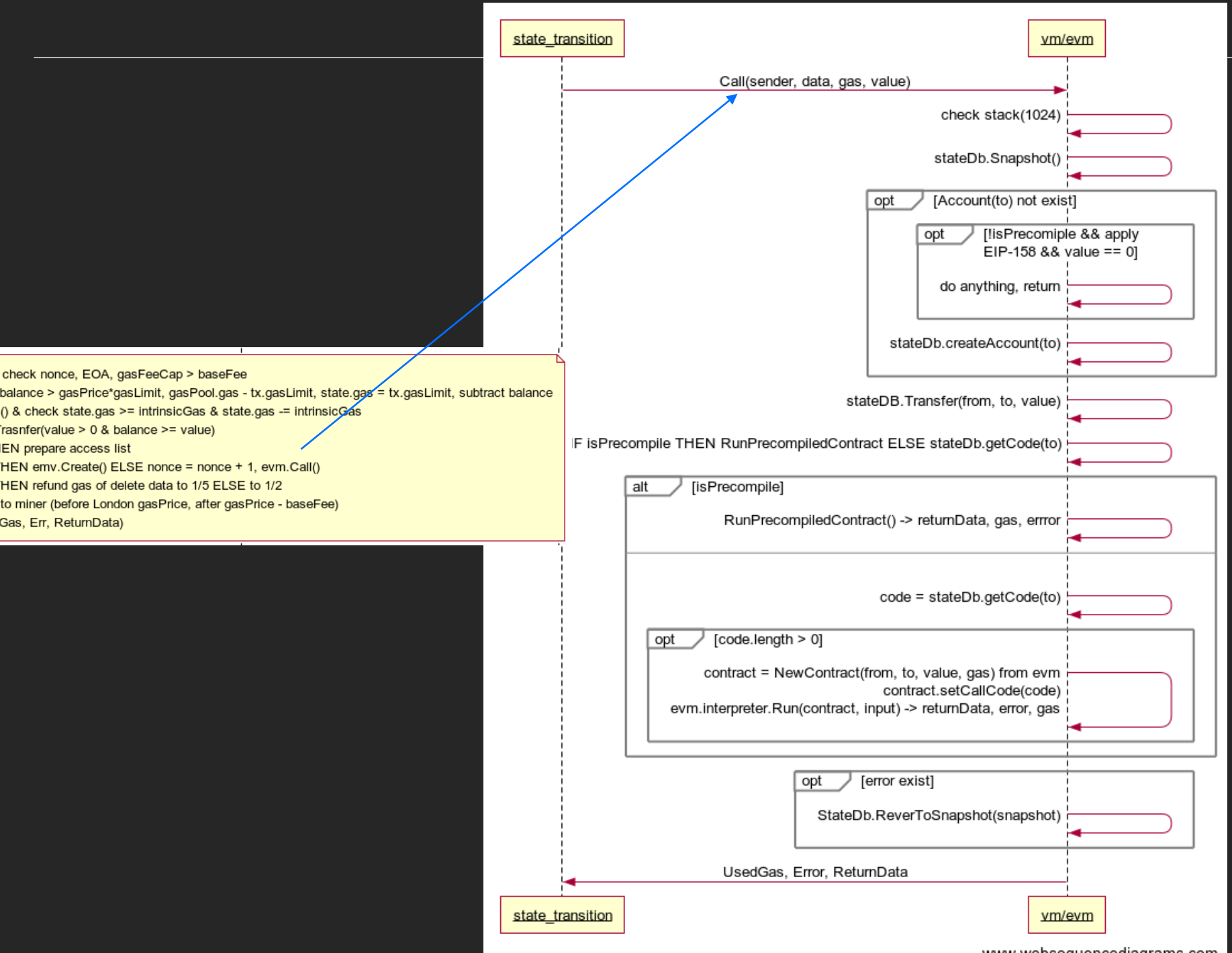



# worker.commitWork() → worker.fillTransactions()




state_processor
evm
vm/evm
ApplyTransaction()
core/state_processor.go, core/evm.go, core/vm/evm.go
transaction.asMessage(singer, baseFee)
NewEVMBlockContext(header, blockchain, author)
coinbase, blockNumber, time, difficulty, baseFee, gasLimit)
NewEVM(blockContext, txContext, statedb, chdinConfig, vmConfig)
new EVM
(CanTrasnfer, Transfer, blockContext, txCContext, stateDb, vmConfig, chainConfig, chainRules)
evm.interpreter = NewEVMInterpreter(evm, vmConfig)
evm
applyTransaction
NEWEVMTxContext(message)
TxContext(From, GasPrice)
Reset(txCtx, stateDb): evmTxContext = txCtx, evm.StateDb = stateDb
state_transaction.ApplyMessage(evm, message, gasPool)
NewStateTransaction(evm, msg, gp).TransitionDb()
preCheck(): check nonce, EOA, gasFeeCap > baseFee
└ buyGas():balance > gasPrice*gasLimit, gasPool.gas - tx.gasLimit, state.gas = tx.gasLimit, subtract balance
IntrinsicGas() & check state.gas >= intrinsicGas & state.gas -= intrinsicGas
check CanTrasnfer(value > 0 & balance >= value)
IF Berlin THEN prepare access list
IF to == 0 THEN emv.Create() ELSE nonce = nonce + 1, evm.Call()
IF London THEN refund gas of delete data to 1/5 ELSE to 1/2
Add gasTip to miner (before London gasPrice, after gasPrice - baseFee)
return(UsedGas, Err, ReturnData)
statedb.Finalise(deleteEmptyObjects: true)
usedGas += result.usedGas
create receipt & BloomFilter
state_processor
evm
vm/evm

- Contract Creation



- Geth 초기화
  - block 0
- Block 1
  - 컨트랙트 배포
  - 채굴
  - Level Db 조회
- Block 2
  - Legacy Transaction Submit
  - 채굴
  - Level Db 조회
- Block 3
  - Type 1 Transaction Submit
  - Type 2 Transaction Submit
  - Level Db 조회

- https://notes.ethereum.org/@vbuterin/serenity_design_rationale
- RLP -> SSZ
- EVM -> eWASM
- PoW(ethash + modeif GHOST) -> PoS(gasper) [ casper + modfied GHOST ]
- Sharding (Scaling) - Multi chain (Rollup)

기본 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ

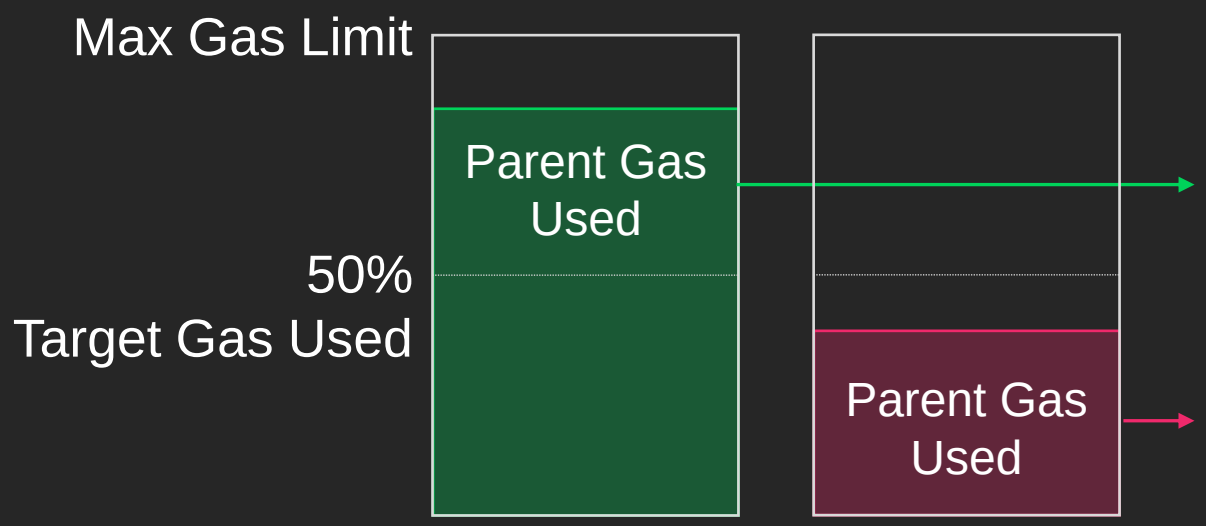
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ

포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ

비트코인 색깔 비트코인 색깔 비트코인 색깔 비트코인 색깔 비트코인 색깔
이더리움 색깔 이더리움 색깔 이더리움 색깔 이더리움 색깔 이더리움 색깔

# 채굴자는 블록 Node Gas Ceiling(한계) 지정
# Target Gas Used = Node Gas Ceiling의 50%

Node Gas Ceiling : 약 30,000,000
Target Gas Used : 약 15,000,000

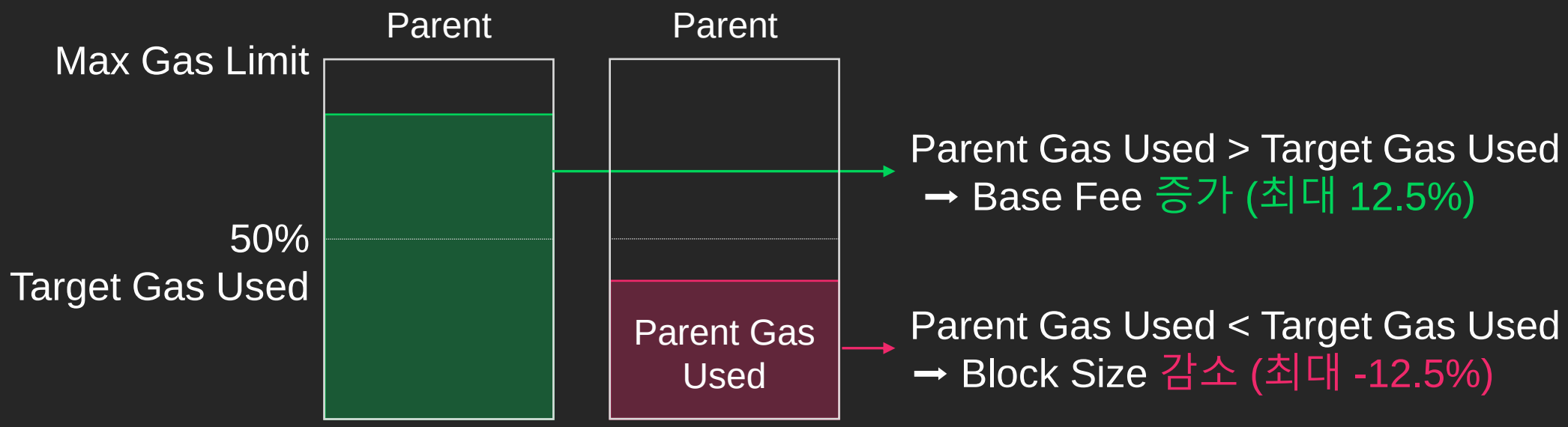



Parent Gas Used > Target Gas Used
➞ Block Size 증가 (Parent Gas Limit / 1024 범위 내)

Parent Gas Used < Target Gas Used
➞ Block Size 감소 (Parent Gas Limit / 1024 범위 내)

# 채굴자는 블록 Max Gas Limit 지정
# Target Gas Used = Max Gas Limit의 50%

채굴자(Miner) 설정
Max Gas Limit : 약 30,000,000
Target Gas Used : 약 15,000,000



- 실제 계산
  - 13047888, https://etherscan.io/block/13047888
    - Gas Used : 2,525,954 (-83%)
    - Base Fee Per Gas: 0.000000039492766985 Ether (39.492766985 Gwei)
    - Gase Limit: 29,970,705
  - 이전 블록: 13047887, https://etherscan.io/block/13047887
    - Gas Used: 29,967,198 (99.99%)
    - Base Fee Per Gas: 0.000000035105594361 Ether (35.105594361 Gwei)
    - Gas Limit: 29,970,705
  - 35.10 * (1 + 1/8 * 1) = 39.49 (소수점 세째 자리 반올림)
    - 35.10 : 이전 기본 수수료 (35.105594361 Gwei)
    - 타켓 블록 사이즈 = Gas Limit / 2 = 29,970,705 / 2 = 14985352.5
    - 1 = (이전 블록 사이즈 - 타켓 블록 사이즈) / 타켓 블록 사이즈 (29,967,198 - 14985352.5) / 14985352.5 = 0.9997659714711415697428538968302 5
  - 최대한 정확한 계산
    - 35.105594361 * (1 + 1/8 * (0.9997659714711415697428538968302 5))
    - 35.105594361 * 1.124970746433892696217856737103 8
    - 39.49276669229962429562467749757 6

- 하나의 블록은 반드시 하나의 모블록을 지정해야 하며, 0 또는 그 이상의 삼촌을 지정해야 한다.
- 블록 B에 포함된 삼촌은 다음과 같은 속성들을 가지고 있어야 한다.[4]
- B의 k번째 조상의 직접적인 자손이어야 한다. 여기서 ‘2≤k≤7’.
- B의 조상이어서는 안 된다.
- 유효한 블록 헤더여야 하지만, 이전에 확인되었을 필요도, 또는 심지어 유효한 블록일 필요도 없다.
- 이전 블록들에 포함된 모든 삼촌들, 그리고 같은 블록에 포함된 다른 삼촌들과는 달라야 한다.(중복포함방지)
- 블록 B에 있는 각 삼촌 U에 대해, B의 채굴자는 코인베이스 보상에 더해 추가로 3.125%를 받고, U의 채굴자는 기본 코인베이스 보상의 93.75%를 받는다.
- 단지 최대 7세대만 삼촌을 포함할 수 있는 수정 고스트 프로토콜을 사용하는 이유는 두 가지이다. 첫째, 무제한 고스트는 하나의 블록에 대해 어떤 삼촌이 유효한지에 대한 계산을 매우 복잡하게 만든다. 둘째, 만일 이더리움과 같은 방식의보상을 하면서도 무제한 고스트를 적용하게 되면 채굴자들이 공격자의 체인이 아니라 주체인에서 채굴을 할 동기를 잃게 될 것이다.

internal/ethapi/api.go

SendRawTransaction
SubmitTransaction

backend.go, api_backend.go

SendTx

b.eth.txPool (tx_pool.go)

AddLocl

AddLocals

addTxs - loop ( check hash, signature)

addTxsLocked - loop

add, validateTx

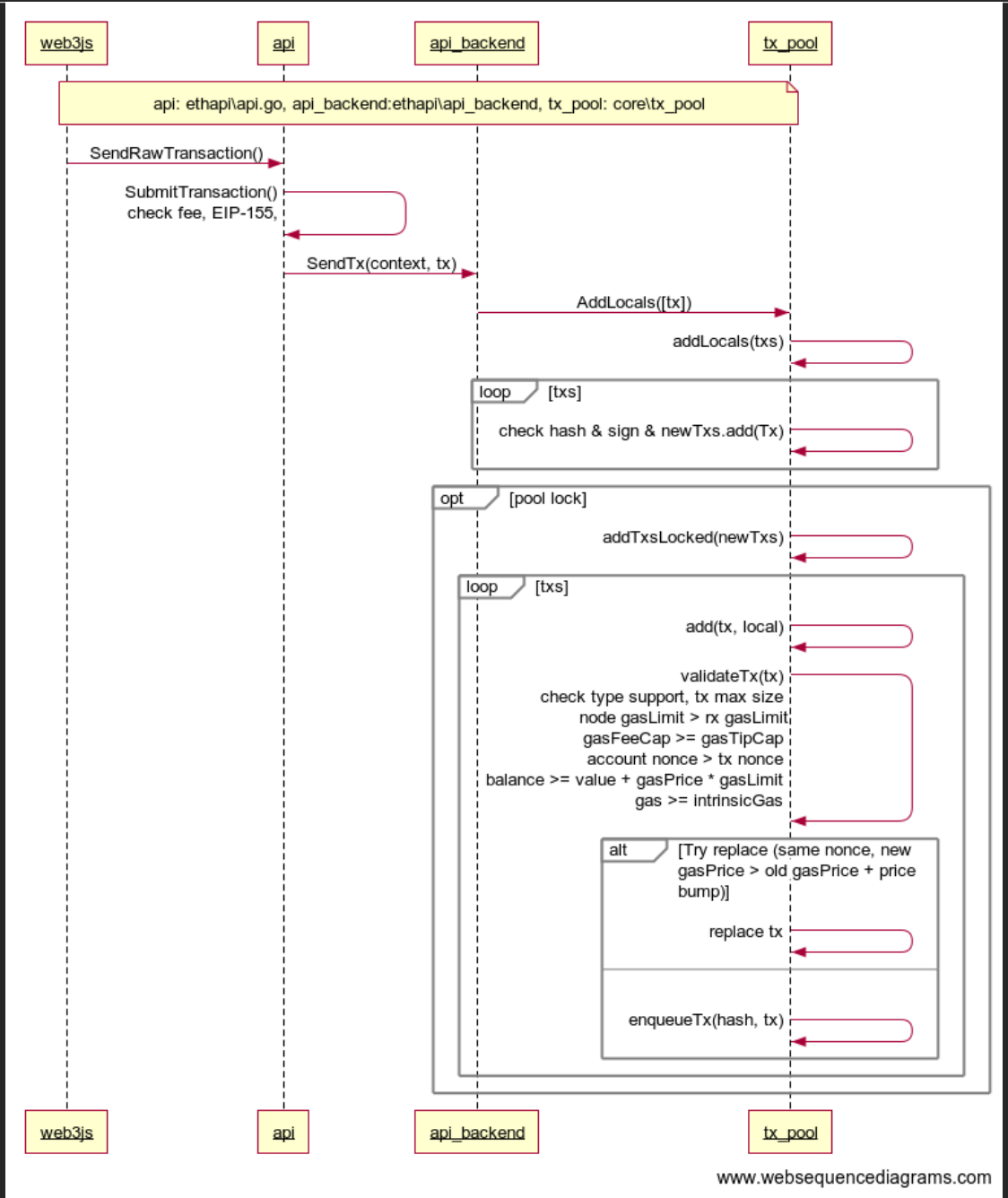
dirtyAccount.addTx

requestPromoteExecuteables

```go
// SendRawTransaction will add the signed transaction to the
transaction pool.
// The sender is responsible for signing the transaction and using
the correct nonce.
func (s *PublicTransactionPoolAPI) SendRawTransaction(ctx
context.Context, input hexutil.Bytes) (common.Hash, error) {
   tx := new(types.Transaction)
   if err := tx.UnmarshalBinary(input); err != nil {
      return common.Hash{}, err
   }
   return SubmitTransaction(ctx, s.b, tx)
}
```

https://github.com/ethereum/go-ethereum/blob/master/internal/ethapi/api.go#L1701

# Transaction Flow



- eth/backend.go startMining()
  - beneficiary(coinbase) 조회
- miner.miner.go start()
  - coinbase 계정 설정
    - sync 중이면 중단
- miner/worker.go commitNewWork()
  - parentHash 조회 및 timestamp 계산 -> timestamp > now + 1s 면 대기
  - number, gasLimit 계산
  - 670: 기본 헤더 생성 (ParentHash, Number, GasLimit, ExtraData, Timestmp)
  - 683: 헤더에 coinbase 설정
  - 685: difficulty 계산 -> w.engine.Prepare(w.chain, header)
  - DAO 하드포킹 블록 별도 검증
  - 744: pending 트랜잭션 가져옴 : w.eth.TxPool().Pending()
  - 754: 가격과 논스 기준으로 트랜잭션 재 정렬 : types.NewTransactionsByPriceAndNonce(w.current.signer, pending)
  - 755 : commitTransactions (w.current.signer, pending) -> miner/worker.go commitTransactions() -> loop 트랜잭션 목록 ➔ 왼쪽
  - 756: worker.commit
    - 772: block = w.engine.Finalize( w.chain, w.current.header, s , w.current.txs , uncles , w.current.receipts )
      - 블록 보상 accumulateRewards
      - stateRoot 에 intermediateRoot 할당
      - 블록 생성

- 755 : commitTransactions (w.current.signer, pending) -> miner/worker.go commitTransactions() -> loop 트랜잭션 목록
  - Block Gas Limit 확인, 남은 가스가 21000 보다 작으면 반복문 종료
    576 : w.current.gasPool.Gas() < params.TxGas(21000) -> break;
  - 581: 다음 트랜잭션 선택
  - singer 조회, EIP-155 처리
  - 599 : stateDB prepare() w.current.state.Prepare(tx.Hash(), common.Hash{}, w.current.tcount)
    -> commitTransaction 호출) w.commitTransaction(tx, coinbase)
  - miner/worker.go commitTransaction() : 실제 트랜잭션 처리
    - snapshop
    - 541: core.ApplyTransaction
      - AsMessage -> vm.NewEVM -> ApplyMessage
      - delete empty objects(after Byzantium
      - gasUsed += gas
      - New Receipt
      - Receipt BloomFIlter 계산
    - 트랜잭션 및 영수증 목록에 추가

- main/worker.go seal() 퍼즐 풀기
  - w.engine.Seal() -> PoW 퍼즐 정답 찾기 (mixHash, nonce)
    hashimoto(https://eth.wiki/concepts/dagger-hashimoto)
- miner/worker.go resultLoop() 블록체인에 블록 추가 및 전파
  - 433 : w.chain.WriteBlockWithState( ) – 블록과 state를 데이터베이스에 커밋
  - 439 : NewMinedBlockEvent Post => minedBroadcastLoop() 실행
  - 454: unconfirmed 블록에 새 블록을 insert