# Blockchain #7

## Ethereum

Prof. Byung Il Kwak

# Review

- Wallet for cryptocurrency
  - Brain Wallet
  - Vanity Wallet
  - Non-deterministic Wallet
  - Deterministic Wallet
  - Hierarchical Deterministic Wallet
  - BIP-39

# CONTENTS

- Basic of Ethereum

# Basic of Ethereum

- Ethereum's Yellow Paper

**ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER**

**ISTANBUL VERSION 80085f7 – 2021-07-11**

DR. GAVIN WOOD
FOUNDER, ETHEREUM & PARITY
GAVIN@PARITY.IO

ABSTRACT. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, with Bitcoin being one of the most notable ones. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

1. INTRODUCTION

With ubiquitous internet connections in most places of the world, global information transmission has become incredibly cheap. Technology-rooted movements like Bitcoin have demonstrated through the power of the default, consensus mechanisms, and voluntary respect of the social contract, that it is possible to use the internet to make a decentralised value-transfer system that can be shared across the world and virtually free to use. This system can be said to be a very specialised version of a cryptographically secure, transaction-based state machine. Follow-up systems such as Namecoin adapted this original "currency application" of the technology into other applications albeit rather simplistic ones.

Ethereum is a project which attempts to build the generalised technology; technology on which all transaction-based state machine concepts may be built. Moreover it aims to provide to the end-developer a tightly integrated end-to-end system for building software on a hitherto unexplored compute paradigm in the mainstream: a trustful object messaging compute framework.

is often lacking, and plain old prejudices are difficult to shake.

Overall, we wish to provide a system such that users can be guaranteed that no matter with which other individuals, systems or organisations they interact, they can do so with absolute confidence in the possible outcomes and how those outcomes might come about.

1.2. **Previous Work.** Buterin [2013a] first proposed the kernel of this work in late November, 2013. Though now evolved in many ways, the key functionality of a blockchain with a Turing-complete language and an effectively unlimited inter-transaction storage capability remains unchanged.

Dwork and Naor [1992] provided the first work into the usage of a cryptographic proof of computational expenditure ("proof-of-work") as a means of transmitting a value signal over the Internet. The value-signal was utilised here as a spam deterrence mechanism rather than any kind of currency, but critically demonstrated the potential for a basic data channel to carry a *strong economic signal*, allowing a receiver to make a physical assertion without

[Source: https://ethereum.github.io/yellowpaper/paper.pdf]

# Basic of Ethereum

- Vitalik Buterin
  - Ethereum의 공동 설립자
  - 1994년 1월 31일생
  - 2013년, 19살에 Ethereum을 제안하는 백서 발행

# Ethereum vs. Bitcoin

| | 이더리움 | 비트코인 |
|---|---|---|
| 사용 목적 | 스마트 계약 블록체인 | 결제용 블록체인 |
| 화폐 | Ether 사용 | Bitcoin 사용 |
| 블록체인 모형 | 계정 기반 (외부 계정, 계약 계정) | UTXO 모형 |
| 채굴 알고리즘 | 현재는 Ethash Proof-of-Work (PoW)로 블록 생성 (향후 Proof-of-Stake (PoS)으로 변경 예정) | SHA-256-기반 Proof-of-Work |
| 블록 생성 시간 | 3 ~ 30 sec | ~ 10 min |
| 블록 크기 | 30KB (2KB/s) | 1MB (1.67KB/s) |

# Ethereum price


<2021.09.11>

# Ethereum accounts

**Bitcoin:**

Bob owns private keys to set of UTXOs

5 BTC ⇒ Bob

3 BTC ⇒ Bob

2 BTC ⇒ Bob

**Ethereum:**

Alice owns private keys to an account

address: “0xfa38b...”

balance: 10 ETH

code: c := a + b

# Ethereum accounts

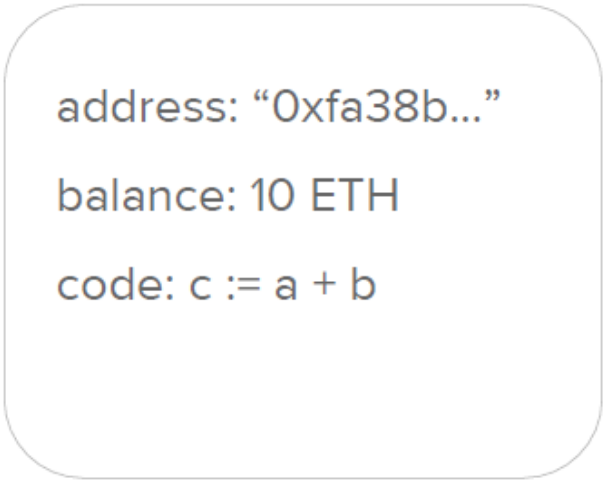
**Bitcoin:**

Bob owns private keys to set of UTXOs


- 이중 지불 문제를 해결이 쉬움
- 거래 기록 생성이 비교적 쉬움

**Ethereum:**

Alice owns private keys to an account




- ‘balance’를 통한 잔액을 효율적으로 업데이트
- 프로그래밍 시 계정 간 잔액 조회 및 송금 편리

# Ethereum

- 계정 구조
  - 소유 계정 (Externally Owned Accounts)
    - 이더리움 사용자를 위한 계정
    - 주소와 연결하여 잔액 정보를 기록
    - **개인 키로 제어**되는 것으로서 코드를 저장할 수 없음

  - 계약 계정 (Contract Accounts)
    - 계약 증명이 있는 계정
    - 주소와 연결해 코드 정보 및 잔액 정보를 기록
    - **스마트 컨트랙트 코드 (Solidity 등의 언어로 짜여진 스크립트) 에 의해 제어**되며 특정 CA에 코드를 저장할 수 있음

# Ethereum

- 계정 구조
  - Externally Owned Accounts (EOA)
    - EOA는 자신 이외의 **다른 EOA** 또는 **다른 CA**에 **메시지를 보낼 수 있음**
      - 이를 위해, 해당 EOA의 개인키 (Private Key)를 사용하여 transaction을 생성하고 서명
    - 두 개의 EOA 사이에서 발생하는 메시지는 단순하게 ETH만을 전송(송금)
      - 하지만, EOA에서 CA로 보내는 메시지는 CA에 저장된 코드를 활성화시키고, 해당 코드에 수반되는 다양한 기능을 활성화함
      - e.x) 토큰 전송, 내부 저장소에 작성, 계산 수행, 새로운 contract 생성 등

# Ethereum

- 계정 구조
  - Contract Accounts (CA)
    - CA는 스스로 새로운 transaction을 개시할 수 없음
      - CA는 자신 이외의 CA, 또는 EOA로부터 받은 transaction에 대한 **응답으로만 transaction을 실행할 수 있음**
    - transaction의 "시작점"을 정리하자면, **이더리움 블록체인의 모든 기능은 항상 "EOA"가 날린 transaction으로부터 시작**됨
    - 하지만, CA는 어떤 transaction을 '직접' 전송하는 계정이 아님

# Ethereum

## 계정 구조

- 블록 크기에 제한을 두지 않지만, 모든 이더리움 사용자의 잔액 정보를 블록에 넣게 될 경우
  - 블록 크기가 상상 이상으로 커지기 때문에 이를 위한 방법 필요

** 계정 정보, 이더 잔액 정보는 **상태 트리 (State Tree)**에 저장

  - 상태 트리는 블록 외부에 보관하고, 각 블록에는 상태 트리의 루트 노드 값만 저장

**상태 트리에 저장하는 계정 정보**

| 계정 정보 | 설명 |
|---|---|
| nonce | - 계정에서 보낸 거래 횟수 (거래 성립시 1씩 증가) |
| balance | - 계정의 잔액 정보를 wei로 나타냄 |
| storageRoot | - 계정과 연결된 저장소 트리 (Storage Tree)의 루트 노드를 나타내는 256비트 해시 |
| codeHash | - Ethereum Virtual Machine (EVM) 코드의 해시값<br>- 계약 계정은 실행 코드를 저장하고, 외부 계정은 공백 문자를 해시값으로 저장 |

# Ethereum smart contract

- 이더리움의 스마트 계약은 이더리움 네트워크 내부에 있는 **자율 에이전트**와 같다고 볼 수 있음

  ※ **자율 에이전트**는 **설정된 목표를 달성하기 위해** 특정 환경 내에서 자동으로 기능을 수행하는 프로세스
  예) 로봇, 인공 생명, 컴퓨터 바이러스 등

- 특정 함수를 호출하는 transaction에 의해 외부 세계에 대한 반응 수행
  - 직접 제어
    - Ether balance
    - Contract state

# Ethereum smart contract

- Smart contract의 목적
  - 데이터 저장 및 유지
    - 데이터는 사용자 또는 기타 계약에 유용
      - 예: 토큰 통화 또는 조직의 멤버십
    - 신뢰할 수 없는 사용자 간의 계약 또는 관계 관리
      - 예: 금융 계약, 에스크로, 보험
    - 다른 계약에 기능 제공
      - 소프트웨어 라이브러리 역할
    - 복잡한 인증
      - 예: 다중 서명 액세스

# Ethereum virtual machine

- 이더리움은 "분산 컴퓨터"
  - 모든 노드가 이더리움 스마트 계약을 실행한 다음 새로운 네트워크 상태에 대한 합의에 도달함

- 이더리움의 분산 합의 프로토콜 (PoW)
  - 마이닝은 Ethereum Virtual Machine (EVM) 코드를 실행하고, 채굴 문제의 솔루션을 검색하여 경쟁적으로 블록을 생성
  - PoW는 경쟁적으로 수행됨
    - 즉, 한 명의 채굴자만이 이더리움 블록체인에 블록을 추가하고 보상을 받음 (Gas + transaction fee)를 받음

# Ethereum virtual machine

- Ethereum node들은 생성된 블록들의 검증 절차 일부로써 EVM을 실행
  - 또한, 블록체인 네트워크 합의 알고리즘은 신뢰할 수 있는 제3자에 대한 필요성을 삭제함
  - 계약 및 조항들은 블록체인에 계속 존재하기 때문에, 스마트 컨트랙트의 계약 위반을 하기 위해서는 전체 네트워크를 파괴해야 함

# Ethereum virtual machine

- Contract code 실행
  - Contract code는 실제로 모든 노드에서 실행되는 EVM byte code를 나타냄
    - EVM에서 Contract가 실행되려면, Solidity로 작성된 Contract가 Op code 를 거쳐 EVM Bytecode로 컴파일 되어야 함

```
pragma solidity ^0.4.19;

contract example {

  address contractOwner;

  function example() {
    contractOwner = msg.sender;
  }
}
```

Contract

```
PUSH1 0x60 PUSH1 0x40 MSTORE CALLVALUE ISZERO PUSH1 0xE
REVERT JUMPDEST CALLER PUSH1 0x0 DUP1 PUSH2 0x100 EXP DU
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF MUL NOT AND S
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND MUL OR SU
0x35 DUP1 PUSH1 0x5B PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN
0x40 MSTORE PUSH1 0x0 DUP1 REVERT STOP LOG1 PUSH6 0x627A
0xb9 SWAP14 0xcb 0x1e 0xdd RETURNDATACOPY 0xec 0xe0 0x1+
0x9C5ABCC14A NUMBER 0x5e INVALID EXTCODESIZE 0xdb 0xcf E
EXTCODESIZE 0xe2 0xb8 SWAP10 0xed 0x
```

Op code

```
/* "Example.sol":26:132  contract example {... */
  mstore(0x40, 0x60)
    /* "Example.sol":74:130  function example() {... */
  jumpi(tag_1, iszero(callvalue))
  0x0
  dup1
  revert
tag_1:
    /* "Example.sol":115:125  msg.sender */
  caller
    /* "Example.sol":99:112  contractOwner */
  0x0
  dup1
    /* "Example.sol":99:125  contractOwner = msg.sender *
  0x100
  exp
  dup2
  sload
  dup2
  0xffffffffffffffffffffffffffffffffffffffff
  mul
  not
  and
  swap1
```

EVM Bytecode

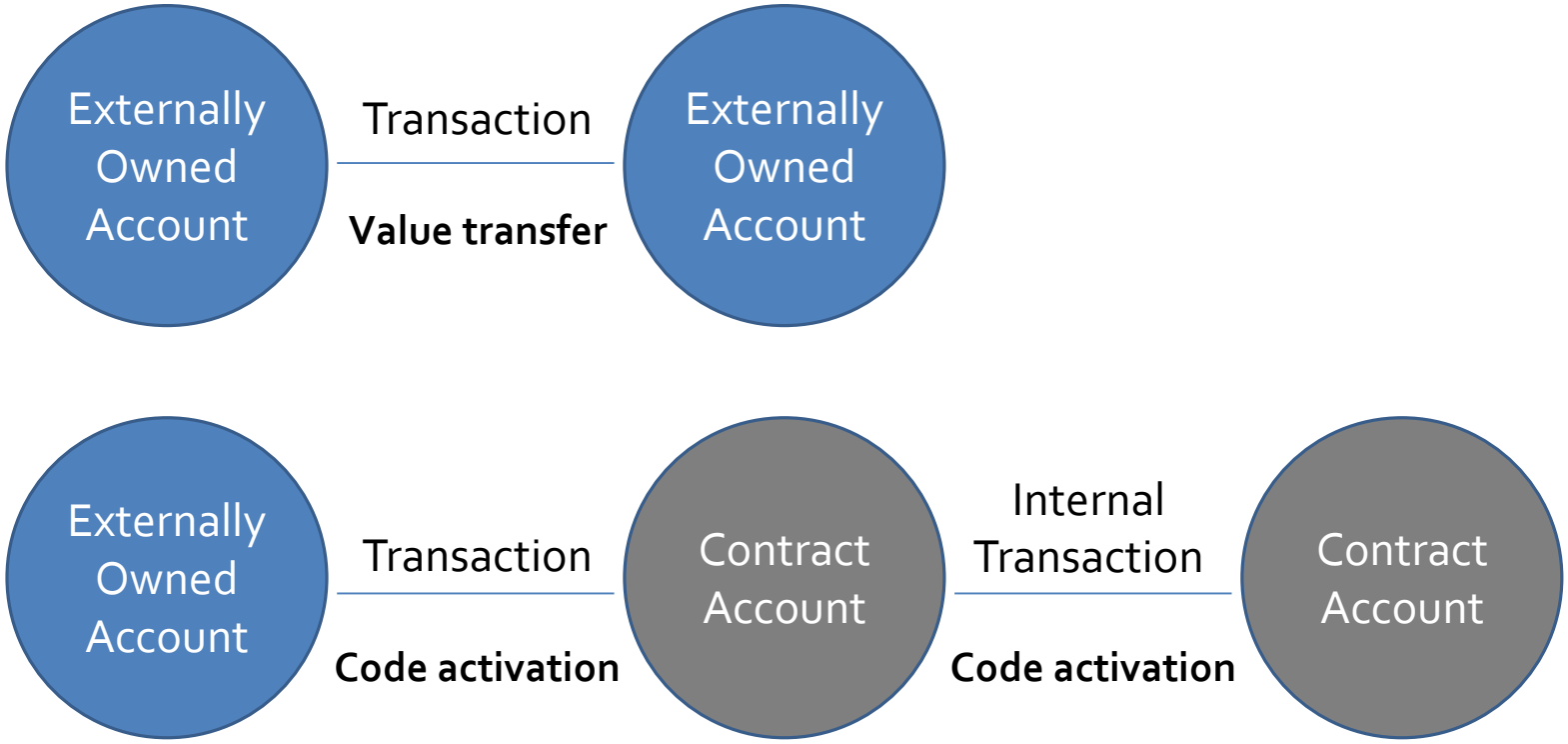
# Ethereum virtual machine

**Externally Owned Account**

transaction을 작성하고 Private key로 서명함으로써,
다른 EOA나 CA에게 메시지를 보낼 수 있음

**Contract Account**

자체적으로 새로운 transaction을 발생시킬 수 없음
타 계정에서 받은 메시지에 대한 응답으로 transaction을 실행




e.g., 토큰 전송, 스토리지/메모리에 데이터 저장, 새로운 토큰 발행, 새로운 Contract 생성 등

# EVM gas and fees

- 이슈사항
  - 스마트 계약에 아래와 같이 무한루프가 있으면 어떻게 될까?

```
function foo()
{
        while (true) {
                /* Loop forever! */
        }
}
```

# EVM gas and fees

- 이슈사항
  - 스마트 계약에 무한루프가 있을 경우
    - 네트워크 상의 모든 노드들이 무한 루프에 빠지게 됨
      - 무한 루프에 빠지게 되면, 해당 프로세스를 종료해 다른 프로세스를 실행할 수 없음.
      - Denial of Service (DoS) - 서비스 거부 공격이 가능
    - 이러한 문제에 대해, 거래 및 계약에 부과되는 수수료 (fee)가 채굴자들의 활동 촉진 및 DoS 공격을 방지함
      - Fee가 충분치 못하면 계약 이행 못함

# EVM gas and fees

- 이더리움의 화폐 단위
  - 내부 화폐 단위로 '이더'가 있음
  - 암호화폐나 애플리케이션을 실행하는 비용으로 'Gas'를 사용

| 통화 단위 | 최소 단위(wei) 기준 환산 금액 |
|---|---|
| wei | 1 |
| kwei, ada, babbage, femtoether | 1,000 |
| mwei, lovelace, picoether | 1,000,000 |
| gwei, Shannon, nanoether, nano | 1,000,000,000 |
| szabo, microether, micro | 1,000,000,000,000 |
| finney, milliether, milli | 1,000,000,000,000,000 |
| ether | 1,000,000,000,000,000,000 |
| keteher, grand | 1,000,000,000,000,000,000,000 |
| mether | 1,000,000,000,000,000,000,000,000 |
| gether | 1,000,000,000,000,000,000,000,000,000 |
| tether | 1,000,000,000,000,000,000,000,000,000,000 |

# EVM gas and fees

- Transaction fee - 'gas'(가스)
  - 비트코인은 블록에 transaction을 저장할 때, 즉 블록을 생성할 때 수수료를 지급해야 함
  - 이더리움 transaction은 네트워크에 속한 노드들에게 연산을 시킴
    - 노드들에게 비용을 지불해야함
    - 이더리움의 수수료는 프로그램 연산을 위한 연료로 'gas'라 함

# EVM gas and fees

- Transaction fee - 'gas'(가스)
  - gas를 상한선까지 모두 사용하는 건 아니며, 사용하지 않은 gas는 돌려받음
    - gas에 대한 상한선을 설정하는 이유는 무한루프를 막기 위함
    - 이더리움에서 스마트 계약을 구현하는 프로그래밍 언어는 '**튜링 완전**' 이라는 특징을 가짐
      - while문과 for문을 실행할 수 있으며, 무한 루프에 빠지게 되면 해당 프로세스를 종료해 다른 프로세스를 실행할 수 없음

**** 튜링 완전**

- 어떤 컴퓨터 알고리즘이든 이론적으로 시뮬레이션 할 수 있는 프로그래밍 언어를 뜻함
- 튜링 기계와 같은 능력이 있다고 가정하며, 현실에 맞는 속도로 동작하는지는 고려하지 않음
- '**아주 작은 단위로 이루어진 명령문의 반복 실행 가능함**'을 의미함

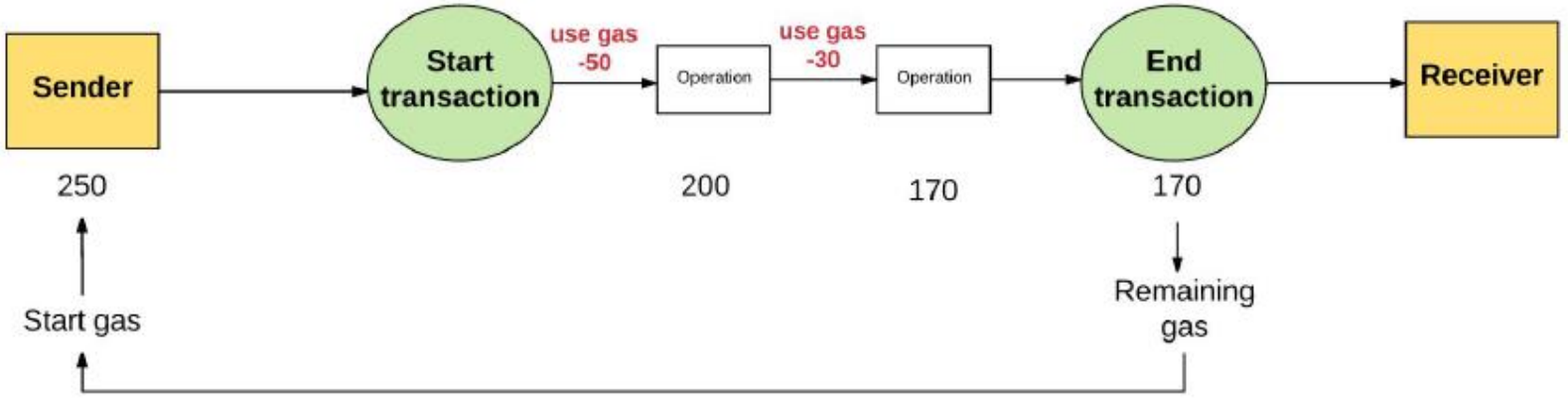
# EVM gas and fees

- Transaction fee - 'gas'(가스)
  - 이더리움 네트워크에서 transaction의 결과로 발생하는 모든 계산에는 gas라는 수수료가 발생함
    - gas는 특정 계산에 대한 수수료를 측정하는 데 사용되는 단위
      - gas 가격은 지출할 의향이 있는 Ether의 양을 의미
      - $1\ \text{Eth} = 10^{18}\ wei, \quad 1\ gwei = 10^{9}\ wei$
    - 모든 transaction 에서 발신자는 가스 한도와 가스 가격을 설정
      - **최대값 = gas price * gas limit,**
      - 최대값은 발신자가 transaction에 대해 지불할 의향이 있는 wei 금액임

# EVM gas and fees

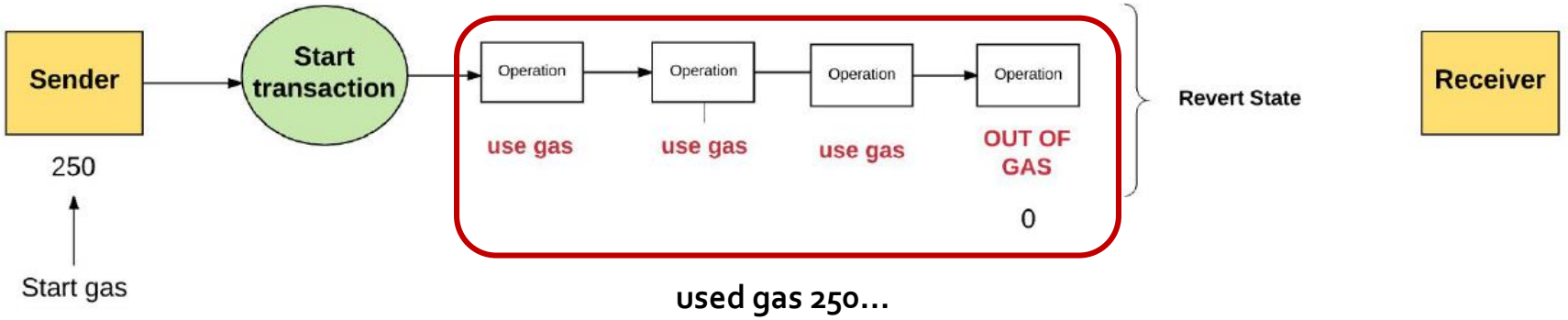
- Transaction fee - 'gas'(가스)
  - 예시
    - gas limit: 50,000 gas
    - gas price: 20 gwei
    - 최대 수수료: 50,000 * 20 gwei

      = 1,000,000,000,000,000,000 = 0.001 eth



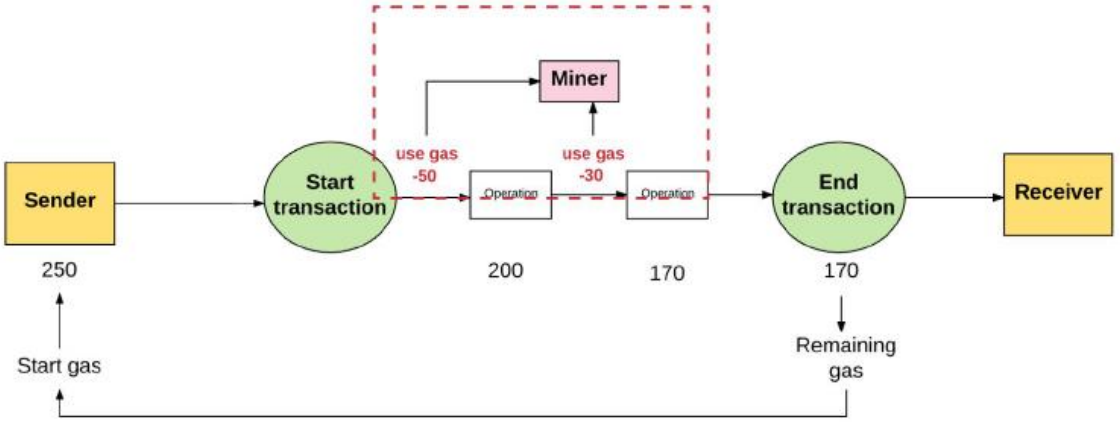
# EVM gas and fees

- Transaction실행을 위한 gas가 충분하지 않을 경우
  - **gas 부족으로 해당 transaction은 유효하지 않음**
  - **상태 변경이 취소**되고 **실패한 transaction으로 기록**됨
  - 네트워크에서 transaction 포함을 위한 연산 과정 중에 gas를 소비했기 때문에 **사용된 gas는 환불되지 않음**



# EVM gas and fees

- 수수료는 어디로 가나요?
  - 수수료는 채굴자의 이더리움 주소로 들어 감
    - 계산 실행과 transaction 검증을 수행 (Reward)
  - 발신자가 지불하고자 하는 gas 가격이 높을수록 채굴자가 얻는 가치가 상승
  - 따라서, 채굴자가 검증 or 무시할 transaction 을 선택
    - 높은 gas 비용을 지불하게 되면 블록에서 선택할 가능성이 높아짐 (즉, 비트코인에서의 거래 수수료와 동일한 개념)



# Use cases of smart contract

- 토큰 시스템 구현
  - **Alice가 거래 시작을 위한 충분한 비용 (gas)을 가지고 있는지 확인**
  - Alice (Sender)의 계정에서 **value** 만큼 빼고 Bob에게 **value**만큼 전달

```
def send(to, value):
    if self.storage[msg.sender] >= value:
        self.storage[msg.sender] = self.storage[msg.sender] - value
        self.storage[to] = self.storage[to] + value
```

# Use cases of smart contract

- Solidity 스크립팅 언어 기반의 스마트 계약 코드

```
pragma solidity ^0.4.24;

계약  contract Helloworld {
        변수  string message = "hello world"

        함수  function setMessage(string msg_) public {
                        message = msg_;
              }

        함수  function getMessage() public view returns (string) {
                        return message;
              }
}
```

# Use cases of smart contract

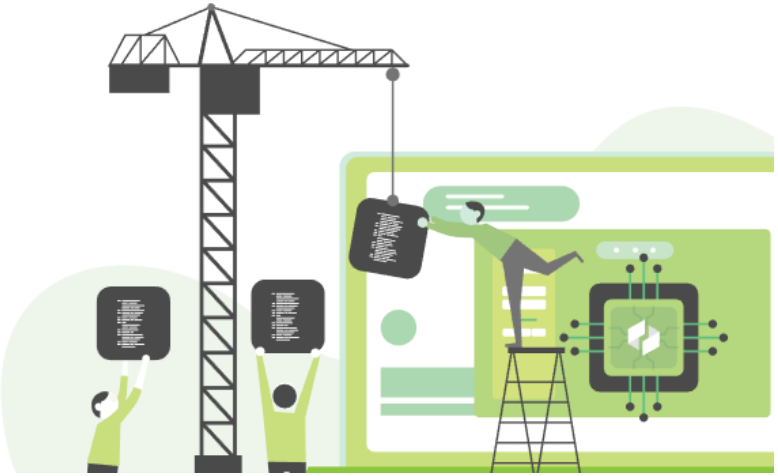
- Standard bounties
  - ETHDenver
    - StackOverflow Q의 답변에 인센티브 제공
    - 표준 현상금 계약을 사용

# CONTENTS

- Ethereum’s consensus

# PoW

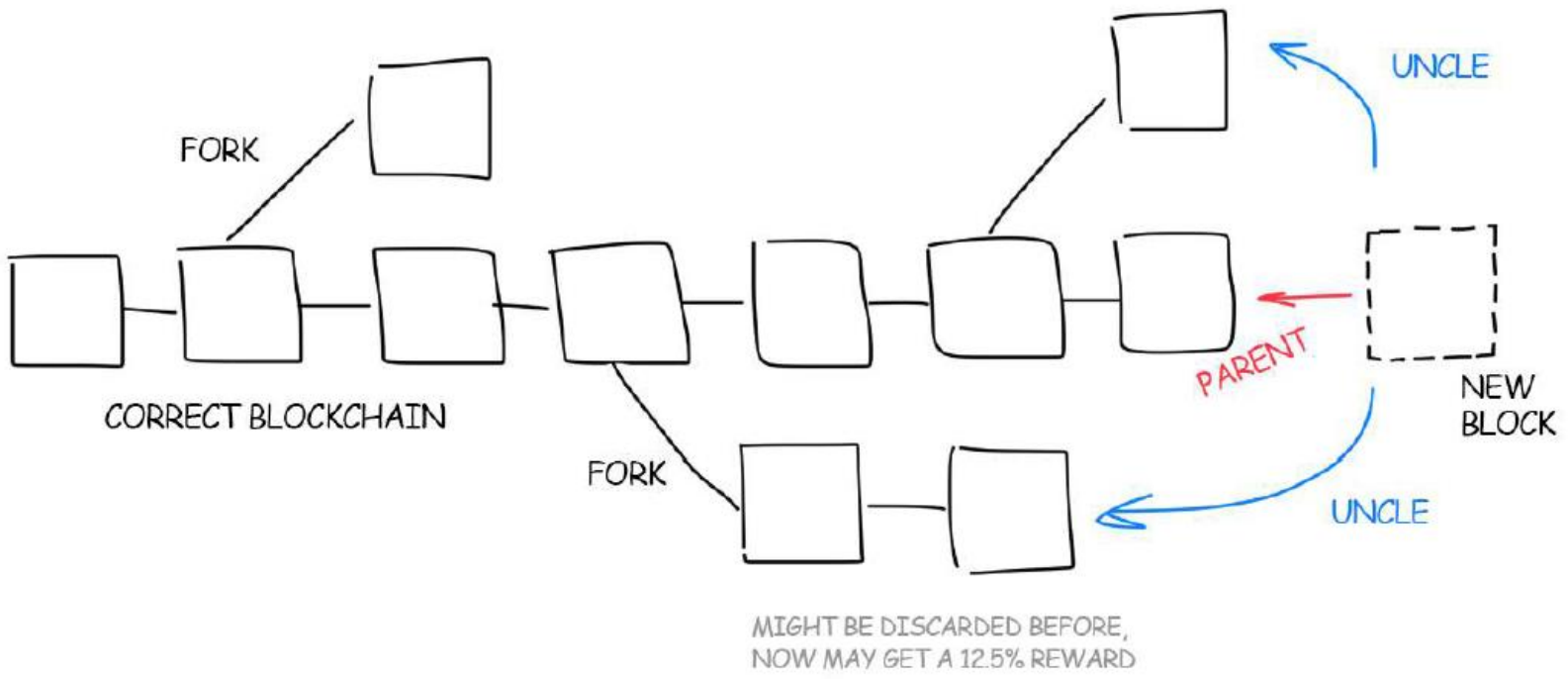
- Ethash needs 2GB to generate hash values



https://ubiqsmart.com/mining

# Uncle blocks

- 이더리움은 Ghost 프로토콜을 사용하여 블록 생성자 삼촌 블록에게 보상을 제공

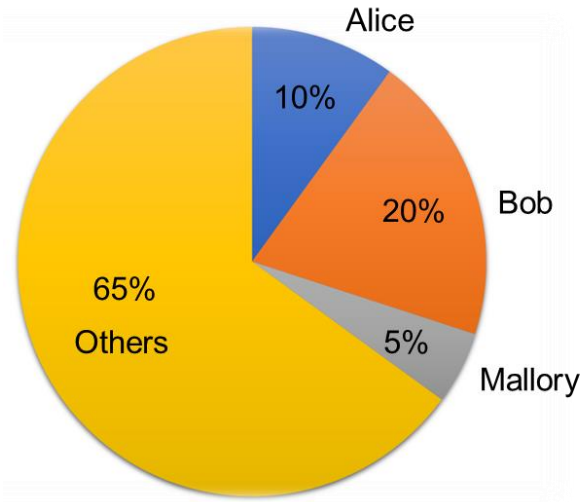


# Proof-of-Stake (PoS)

- PoS 기반 합의 알고리즘에서 블록체인 노드들의 집합은 다음 블록에서 차례로 생성 및 유효성 검사를 수행
  - 각 블록체인 노드의 가중치 (weight)는 보증금 (즉, 지분)의 크기에 따라 다르게 설정됨
- PoS 알고리즘
  - 블록체인은 블록체인 노드에서 선택된 검증자들에 대한 그룹을 추적.
  - 블록체인 기반 토큰 (예: 암호 화폐)을 보유한 모든 온라인 검증자들은 토큰을 예치금으로 잠그는 특별한 유형의 거래를 전송하여 검증인이 될 수 있음
  - 그 후, 새로운 블록 생성과 동의과정들은 모든 검증인들이 참여할 수 있는 합의 알고리즘을 통해 이뤄짐

# Proof-of-Stake (PoS)

- PoS는 Stake 검증 노드들이 얼마나 많은 지분을 가지고 있는지에 따라 블록 생성에서의 결정론적 방법을 가짐

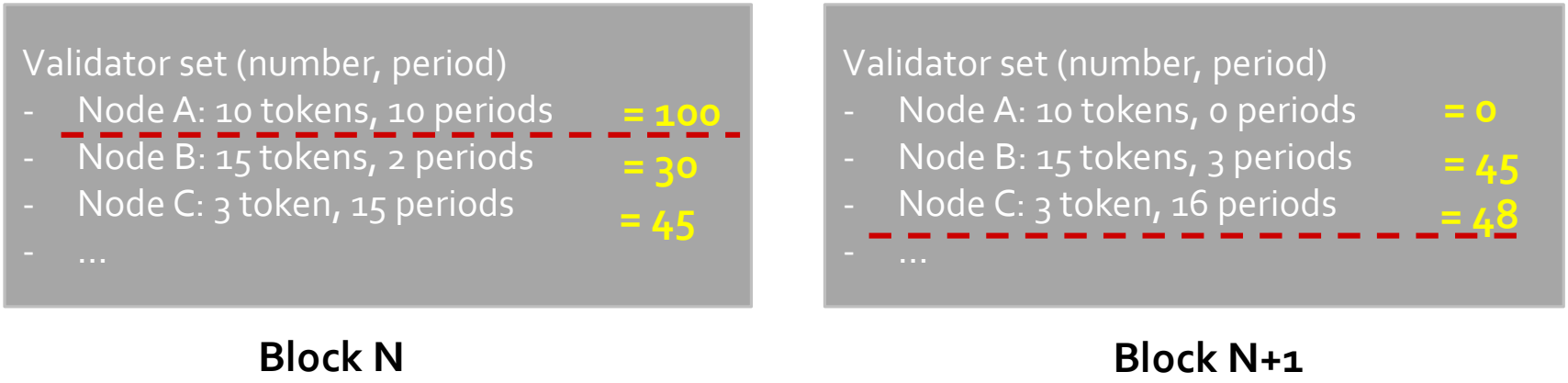



- 블록을 생성하도록 선택된 검증자는 블록체인 시스템이 보유한 토큰으로 보상을 받음
  - 예) Bob은 블록 생성자로 선정될 확률이 높지만, 블록 생성 노드는 중앙 집중화를 방지하기 위해 무작위로 선택 되어야함.
    - 그렇지 않을 경우, 시스템에서 가장 부유한 개인이 항상 다음 블록을 생성하고 지속적으로 자신의 부를 증가시켜 결과적으로 시스템에 대한 통제력을 갖게 될 수 있음

# Proof-of-Stake (PoS)

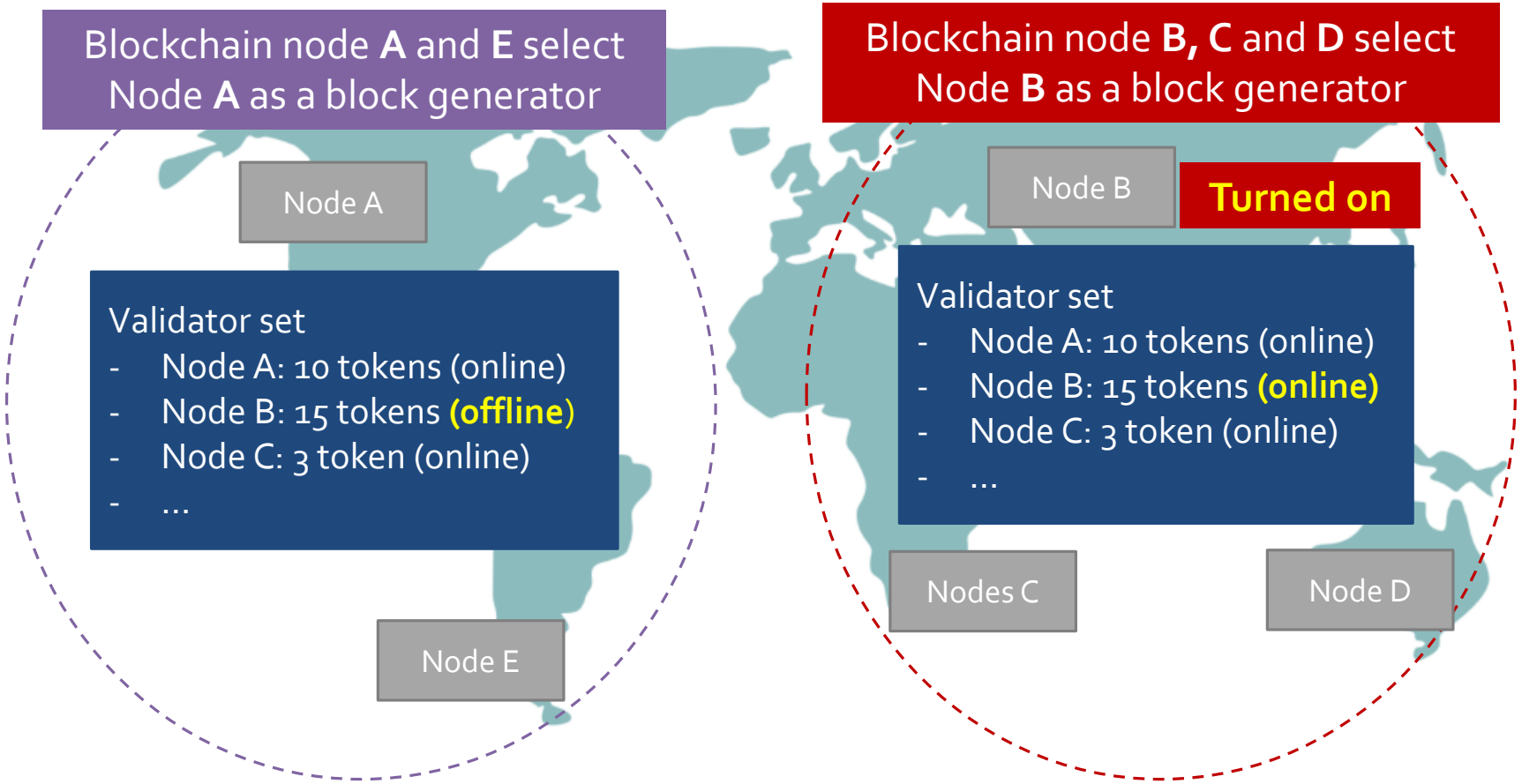
- 블록 생성 노드는 무작위 선택을 통해 검증 노드 집합에서 선택됨
  - 일반적으로 토큰의 수와 토큰 보유 기간을 적용

- 예: Age-based selection
  - 토큰 수 x 토큰 보유 일수 => 얼마나 오래 토큰을 가지고 있었는지



# Proof-of-Stake (PoS)

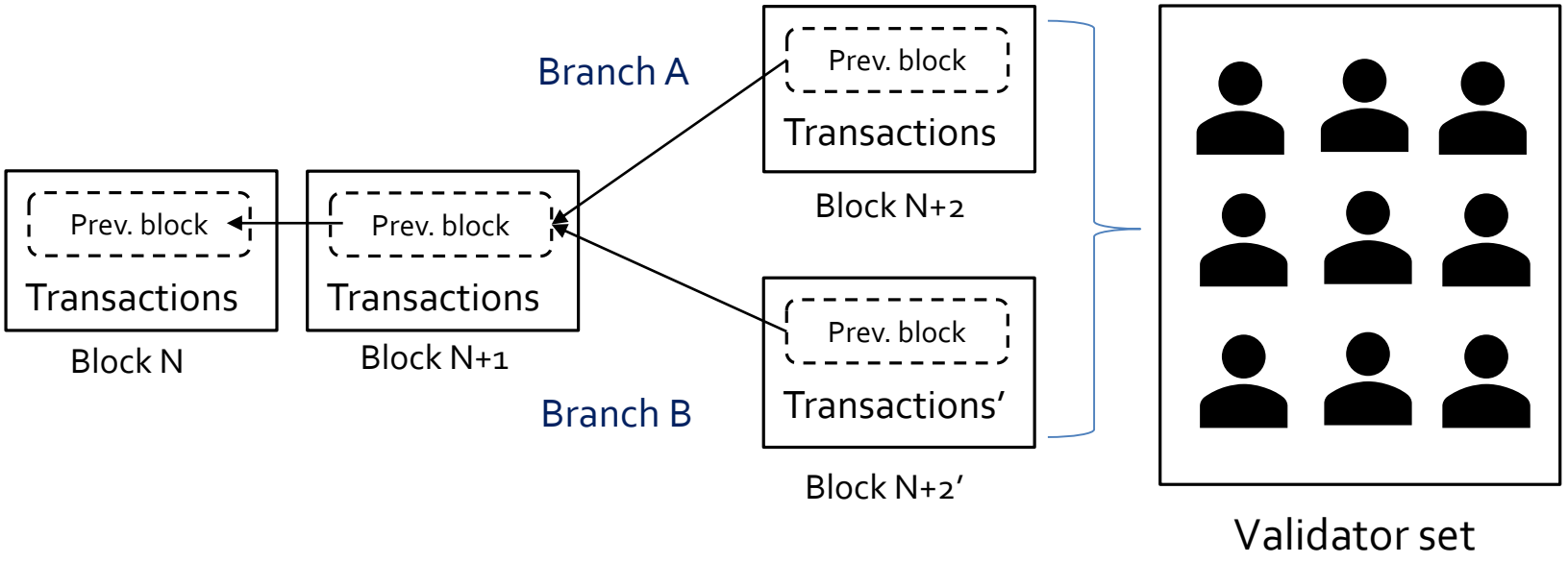
- Forked chain of PoS based blockchain



# Proof-of-Stake (PoS)

- 분기된 체인의 처리를 위한 유효성 검사 프로세스
  - 사전 정의된 규칙으로 선택된 모든(온라인) 검증 노드들은 블록 검증을 위해 특별한 유형의 transaction을 전송
    - 블록 검증에 참여한 검증 노드들은 블록체인 시스템에서 보상을 받음

**모든 검증 노드들이 1개의 분기를 선택**



# Proof-of-Stake (PoS)

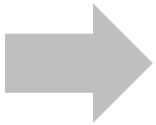
- 블록의 구조에 이전 블록의 유효성 검사 프로세스에 대한 새로운 필드를 포함 해야함

Block #N

**Block ID**
*7dfa3b…*

**Previous Block #N-1**
*e4ab41…*

**Transactions**
1. *Transaction_1*
2. *Transaction_2*
3. *Transaction_3*



Block #N

**Block ID**
*7dfa3b…*

**Previous Block #N-1**
*e4ab41…*

**Transactions**
1. *Transaction_1*
2. *Transaction_2*
3. *Transaction_3*

…

**Validation for Block #N-1**
*Validator 1's signature*
*Validator 2's signature*

…

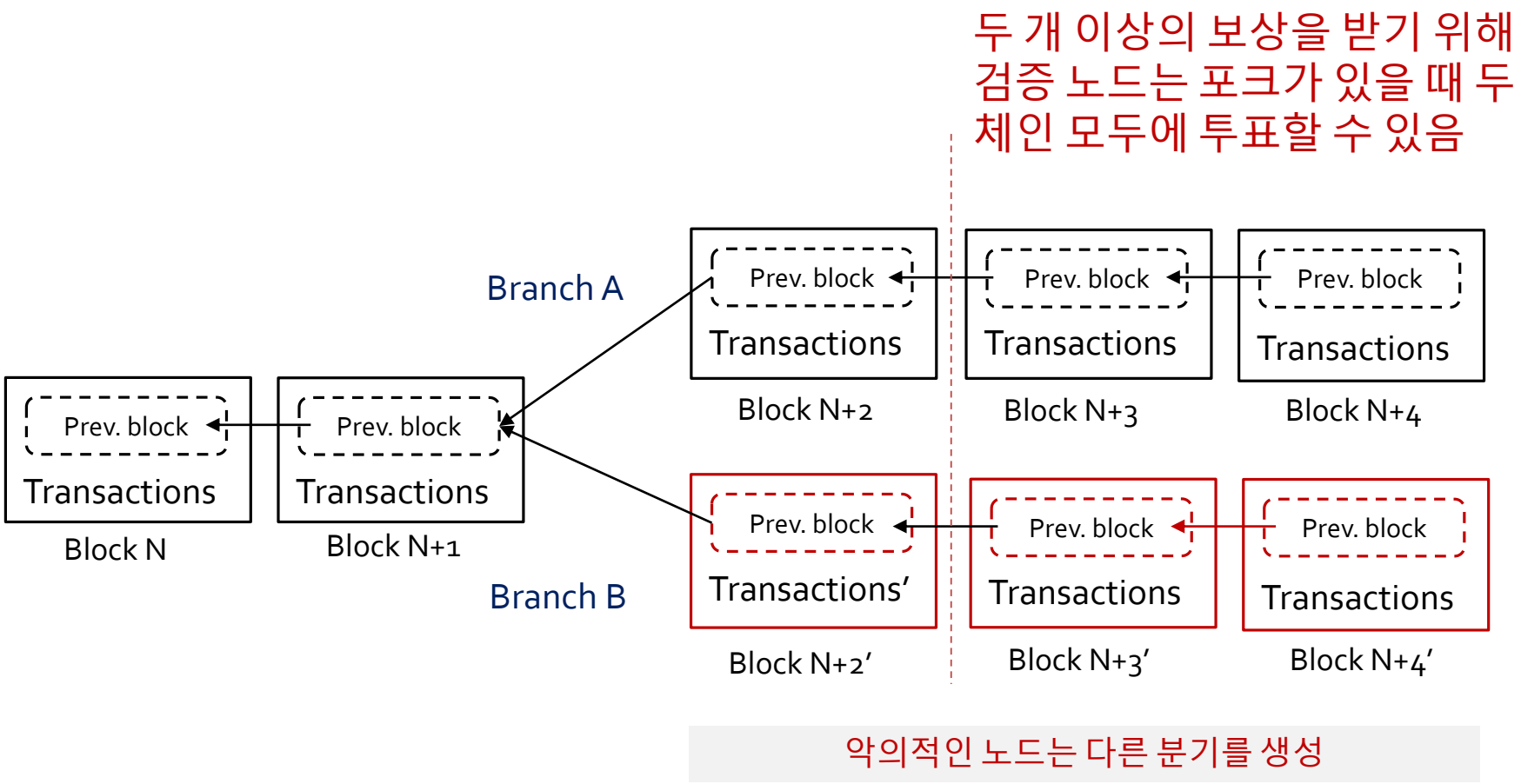
*(Source: J. Kwon. Tendermint: Consensus without Mining)*

# Proof-of-Stake (PoS)

- 매우 낮은 전력 소비
  - PoS 기반 합의의 주요 장점은 PoW 기반 합의 시스템보다 훨씬 적은 에너지를 사용한다는 것
    - 즉, PoS가 PoW보다 비용에서 보다 효율적임
- 51% 공격 확률 감소
  - 51% 공격을 수행하려면 개인 또는 그룹이 블록체인 시스템에서 대부분의 토큰 (예: 암호화폐)을 소유해야 함.
  - 또한, 51%의 공격자가 보유하고 있는 토큰의 가치를 크게 떨어뜨리므로 네트워크를 공격하는 것은 역효과
- PoS 기반의 합의는 "nothing at stake" 문제 고려
  - 두 개 분기 중 어느 것을 선택하더라도 손해가 없는 경우 (선택이 어려움)

# Proof-of-Stake (PoS)

- Nothing at stake problem



# Proof-of-Stake (PoS)

- Nothing at stake problem -> simultaneous slasher
  - 모든 검증 노드들은 블록을 검증하기 위해 일정량의 지분을 예치해야 함
    - 블록이 임계값 이상으로 검증되면 블록을 검증한 검증 노드가 보상을 받음
  - 악의적인 검증자가 두 개 분기에 동시 서명하여 악의적인 블록체인 노드 또는 잘못된 거래를 하는 클라이언트가 있을 때 더 많은 보상을 얻는다면, 미래의 올바른 검증 노드는 미래의 블록에 이중 서명의 증거를 포함하고 악의적인 검증인의 보상을 줄일 수 있음
    - 미래의 정직한 검증 노드에게 보상을 주기 위해 악성 검증 노드의 예치금을 잠금

# References

- Lecture slides from BLOCKCHAIN @ BERKELEY
- Mastering Bitcoin, https://github.com/bitcoinbook/bitcoinbook
- https://steemit.com/blockchain/@niipoong

# Q & A