비트코인과 이더리움

블록체인이란

In

비트코인과 이더리움

01 비트코인 vs 이더리움

비트코인 월드맵

GLOBAL BITCOIN NODES DISTRIBUTION

Reachable nodes as of Thu Jun 10 2021 14:58:10 GMT+0900 (대한민국 표준시).

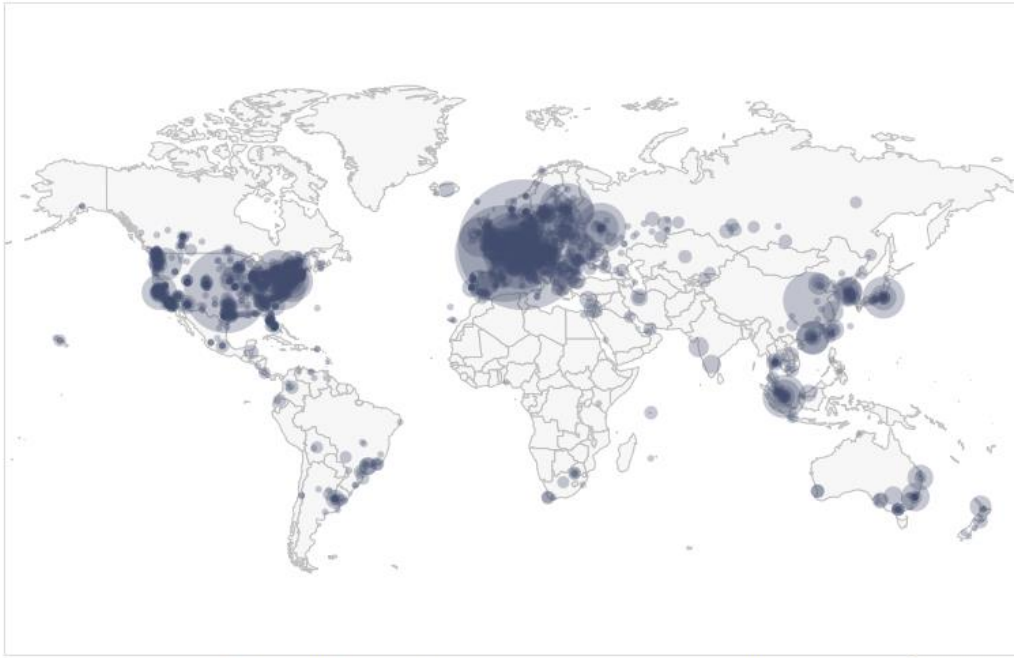
9598 NODES

24-hour charts »

Top 10 countries with their respective number of reachable nodes are as follow.

RANK	COUNTRY	NODES
1	n/a	2000 (20.84%)
2	United States	1831 (19.08%)
3	Germany	1752 (18.25%)
4	France	569 (5.93%)
5	Netherlands	415 (4.32%)
6	Canada	300 (3.13%)
7	Russian Federation	248 (2.58%)
8	United Kingdom	242 (2.52%)
9	China	176 (1.83%)
10	Switzerland	152 (1.58%)

More (92) »



Map shows concentration of reachable Bitcoin nodes found in countries around the world.

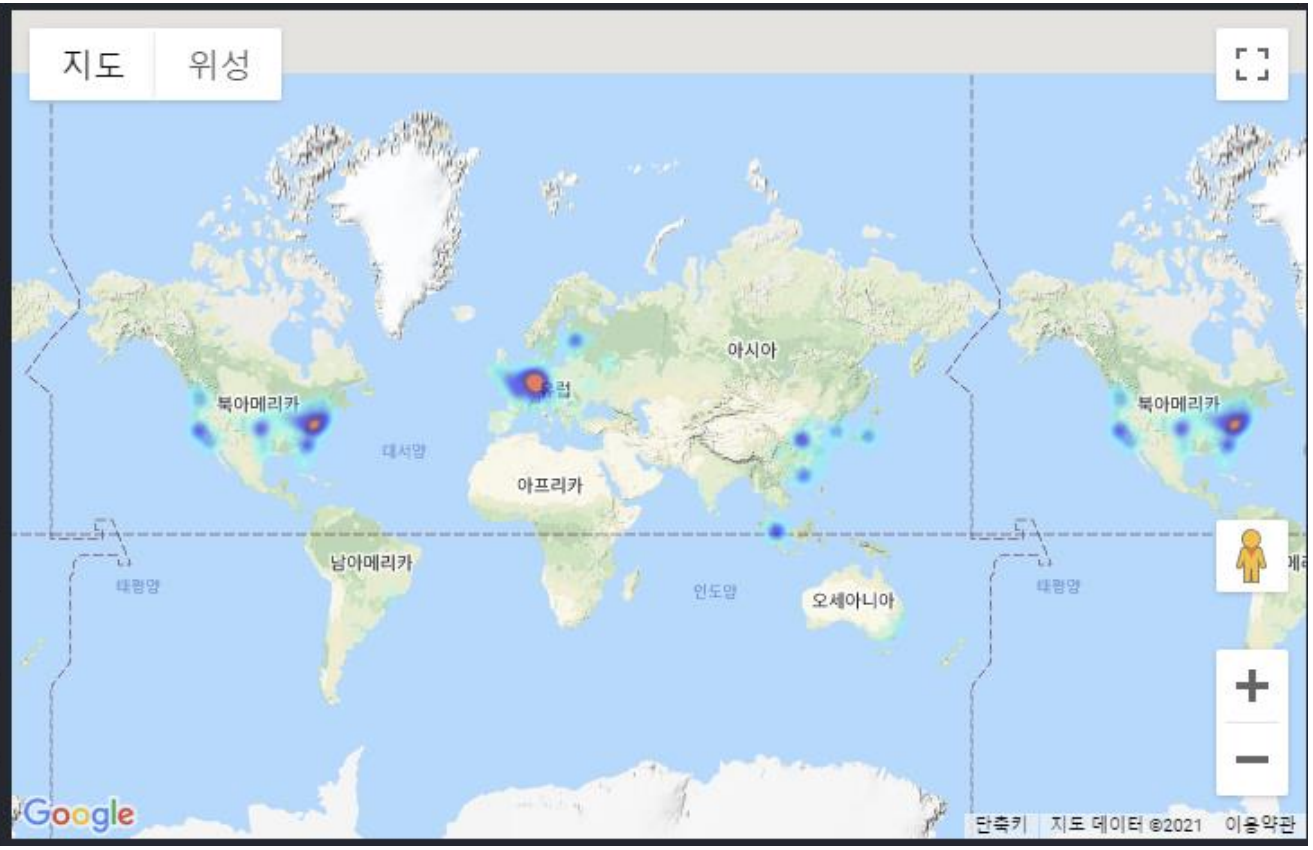
LIVE MAP

(https://bitnodes.io/)

01 비트코인 vs 이더리움

이더리움 월드맵

Total	4026 (100%)
United States	1447 (35.94%)
Germany	776 (19.27%)
China	302 (7.50%)
France	144 (3.58%)
Singapore	144 (3.58%)
United Kingdom	112 (2.78%)
Finland	111 (2.76%)
Canada	109 (2.71%)
Netherlands	88 (2.19%)

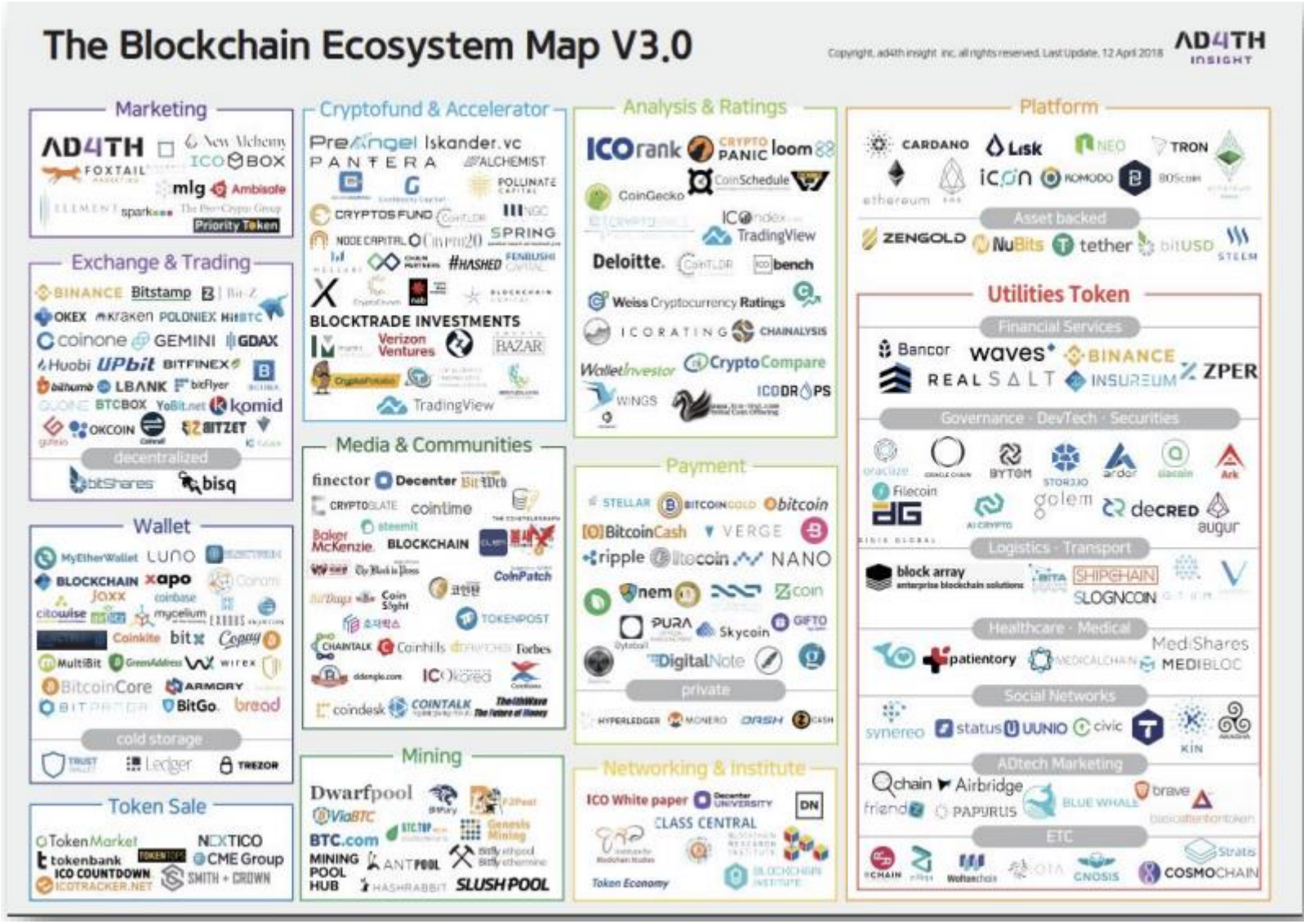


(https://www.ethernodes.org/countries)

02 블록체인의 현재

- 의료 정보 기록을 통한 효율적 건강 관리
- 중고상품 이력관리 (자동차, 부동산)
- 물류 관리 및 화물 추적
- 지적 재산권 보호 (음원, S/W)
- 금융 상품의 디지털화 (보험, 선물, 송금)
- IoT와 결합된 에너지 절약형 가전제품
- 선거 투표 시스템으로 즉각적 의사 반영

02 블록체인의 현재


The Blockchain Ecosystem Map V3.0
Copyright, ad4th insight inc, all rights reserved. Last Update, 12 April 2018
AD4TH INSIGHT
Marketing
Exchange & Trading
decentralized
Wallet
cold storage
Token Sale
Cryptofund & Accelerator
BLOCKTRADE INVESTMENTS
Media & Communities
Mining
Analysis & Ratings
Payment
private
Networking & Institute
Platform
Asset backed
Utilities Token
Financial Services
Governance · DevTech · Securities
Logistics · Transport
Healthcare · Medical
Social Networks
ADtech Marketing
ETC

03 비트코인의 한계점

- 튜링 불완전한 스크립트 언어
- 송금 외에 다른 기능 수행 불가
- PoW 합의 알고리즘

- 튜링 불완전?

03 비트코인의 한계점

튜링 완전 vs 튜링 불완전

튜링 완전(turing completeness)이란 **어떤 프로그래밍 언어나 추상 기계가 튜링 기계와 동일한 계산 능력**을 가진다는 의미이다.
이것은 **튜링 기계로 풀 수 있는 문제, 즉 계산적인 문제를 그 프로그래밍 언어나 추상 기계로 풀 수 있다는 의미**이다.
제한 없는 크기의 기억 장치를 갖는 기계를 만드는 것이 불가능하므로, 진정한 의미의 **튜링 완전 기계**는 아마도 물리적으로 불가능할 것이다. 그러나, 제한 없이 기억 장치의 크기를 늘려갈 수 있다고 가정할 수 있는 물리적인 기계 혹은 프로그래밍 언어에 대해서는, 느슨하게 튜링 완전하다고 간주한다. 이런 맥락에서, 요즘 나온 컴퓨터들은 튜링 완전하다고 여겨지고 있다.

위키백과

튜링 완전 = 무제한적인 기억장치 + 튜링 완전한 언어

튜링 완전한 언어란?

- **프로세스를 충분히 분할할 수 있을 만큼 작은 단위를 사용할 수 있어야 한다.**
- **If (조건문) + for/while(반복 – 루프 문) 이 존재 해야 한다.**

If + for/while이 존재하면 문제를 풀 때까지 영원히 멈추지 않는 알고리즘 설계가 가능하도록.

03 비트코인의 한계점

비트코인의 스크립트 언어

1. 잠금 스크립트 : 출력값을 소비하기 위해 충족되어야 하는 요건을 스크립트로 작성한 것
2. 해제 스크립트 : 잠금 스크립트가 출력값에 걸어둔 조건을 해결해 출력값이 소비될 수 있도록 하는 스크립트

UTXO(Unspent Transaction Outputs)를 기본 단위로 거래

-> 공개키 비밀키 방식과 유사?
➔ 해당 스크립트에서 내부적으로 이를 이용

[실행 중 스택 상태변화]

1. 송금자의 비밀키로 만든 서명 데이터(<sig>)를 스택에 푸시한다.
2. 송금자의 공개키(<punkey>)를 스택에 푸시한다.
3. OP_HASH160 명령으로 스택 최상위에 위치한 (<punkey>)의 HASH160 해시값을 계산하고 이 값 (<punkeyhash>)을 스택에 푸시한다.
4. OP_EQUALVERIFY 명령으로 스택 최상위에 위치한 (<punkeyhash>)을 스택에 푸시한다.
5. OP_EQUALVERIFY 명령으로 스택 최상위에 위치한<punkeyhash>와 그 아래에 위치한 <punkeyhash>를 비교한다. 두값이 일치하면 두 값을 모두 스택에서 삭제한다.
6. OP_CHECKSIG 명령으로 스택 최상위에 있는 송금자의 공개키 (<punkey>)와 그 아래에서 위치한 비밀키로 한서명(<sig>)을 스택에서 꺼내 서명 데이터를 검증한 다음 검증결과를 스택에 푸시한다.

03 비트코인의 한계점

비트코인의 스크립트 언어

- 비트코인은 **송금**만을 목적으로 만들어졌기 때문에,
  존재하는 스크립트 언어에는 for/while 등의 Loop가 필요 없다. (필요가 없음.)
- 상태를 저장하지 않음. (UTXO 기본단위)

➔ 튜링 불완전 언어이다. (과부하 걸리는걸 막기 위함)

➔ 튜링불완전 언어이다?
➔ 송금 외에 다른 문제를 해결하기 어렵다.

- 튜링 불완전한 스크립트 언어
- 송금 외에 다른 기능 수행 불가

04 이더리움의 탄생

핵심개념	전자화폐	스마트 계약
설립자	사토시 나카모토 Satoshi Nakamoto	비탈릭 부테린 Vitalik Buterin
출시년도	2009년 1월	2015년 7월
언어	스크립트 언어	튜링 완전 언어
합의 메커니즘	POW	현재: GPU 기반 POW 목표: POS(casper)
블록 생성 시간	약 10분	약 15초
노드 수	약 9,000 개	약 10,000 개

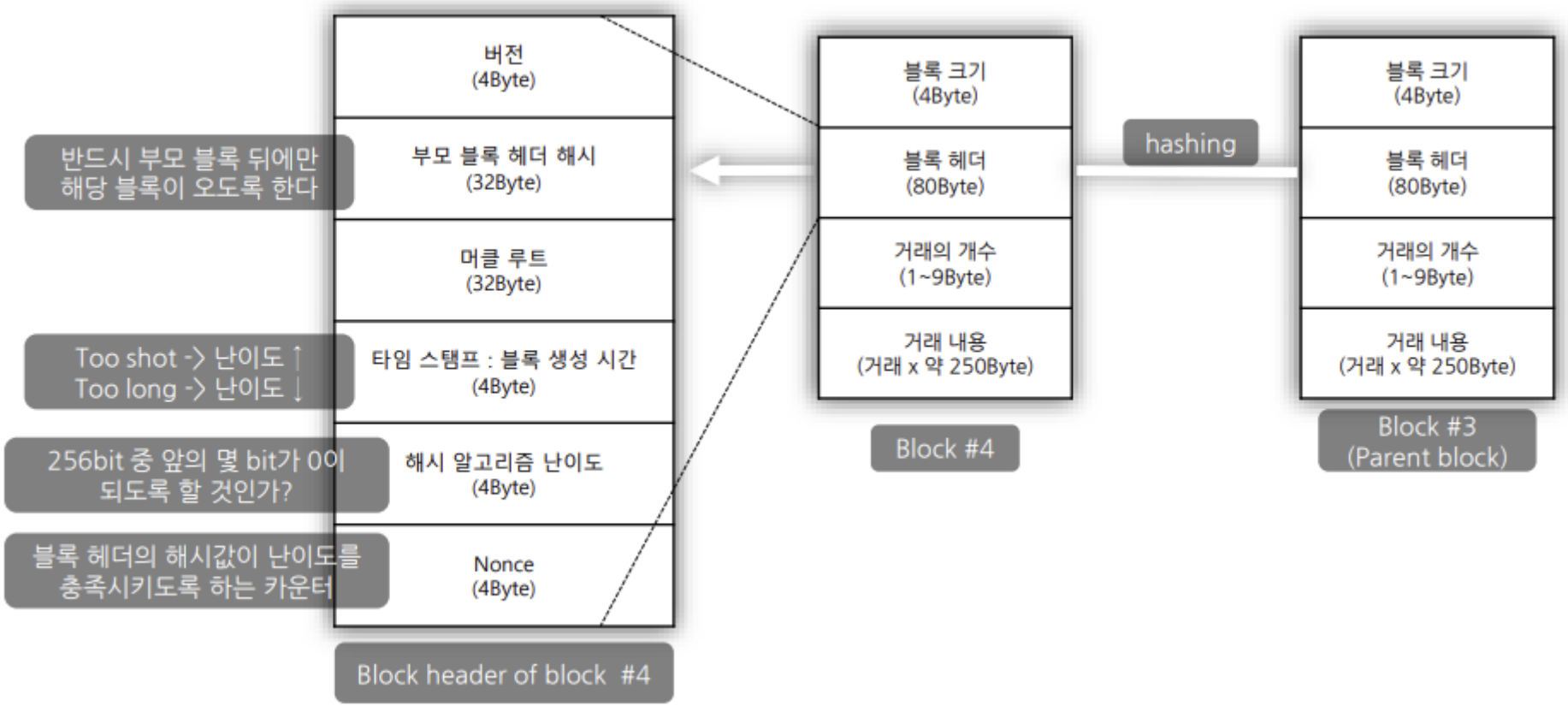
04 이더리움의 탄생

이더리움

- 비트코인의 단점을 보완하는 새로운 블록체인 플랫폼
- Vitalick Buterin이 개발
- 상태 저장 및 데이터 저장이 가능하다. + 반복문 실행이 가능하다. (튜링 완전언어)
- Solidity 등 튜링완전언어를 통해 SmartContract 생성
- 만들어진 Smart Contract는 Dapp을 거쳐 블록체인을 통해 배포 되고 실행된다.

04 이더리움의 탄생

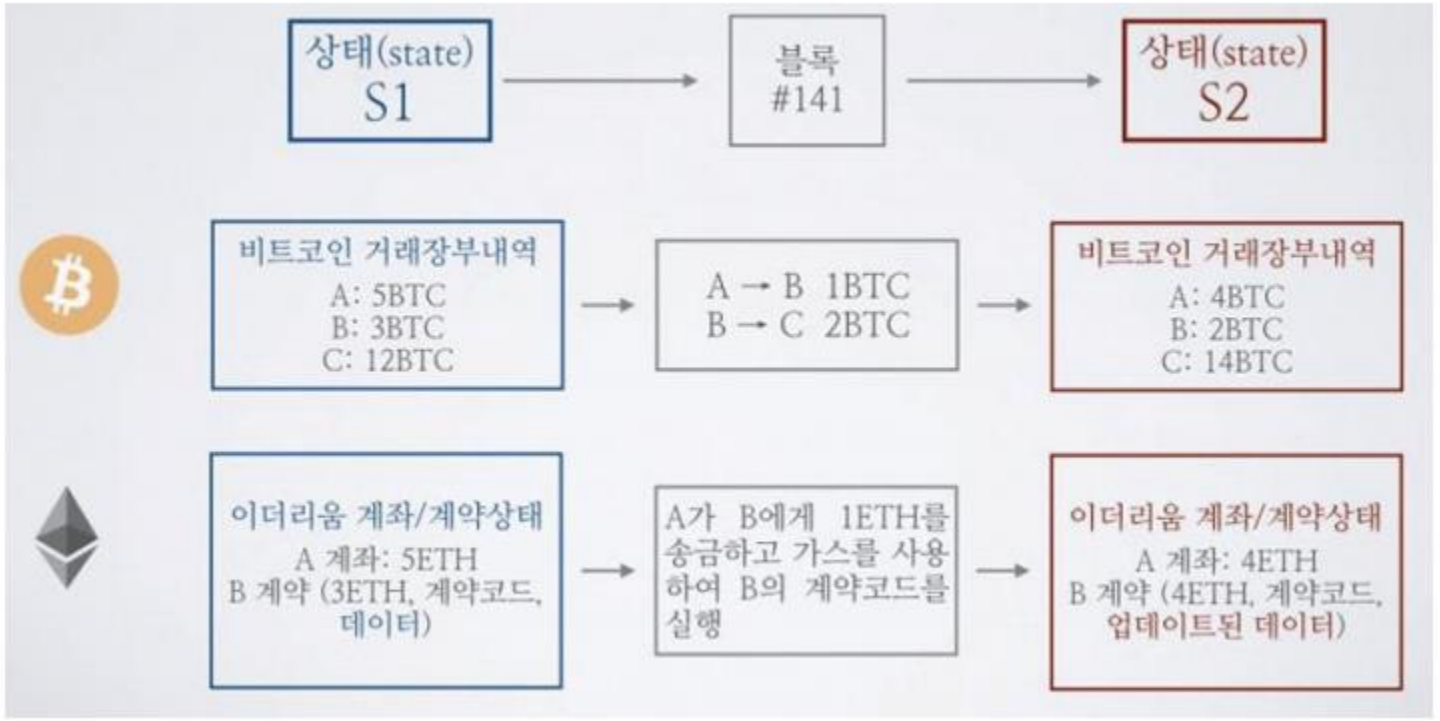
비트코인 구조



04 이더리움의 탄생

이더리움 구조

- 상태 저장 머신
- UTXO 사용 하지 않고 Account 모델



05 이더리움 플랫폼

블록체인 기반의 분산 컴퓨팅 플랫폼 (Distributed Computing Platform)

- 튜링 완전한 프로그래밍 언어를 내장(built-in) 하고 있어 스마트 계약(smart contract) 과 분산 어플리케이션 (Dapp: decentralized applications) 를 구현할 수 있음

- 이더리움 가상머신 (EVM: Ethereum Virtual Machine) 을 통해 모든 참가자(node) 들의 컴퓨터에서 동일한 연산을 수행하며 이를 통해 동일한 상태(state) 에 합의. 전 세계 모든 참가자가 동일한 하나의 컴퓨터를 돌리는 것과 같기 때문에 “세계 컴퓨터 (world computer)” 라 고 불리기도 함.

05 이더리움 플랫폼

이더리움의 화폐 단위

값(웨이)	이름	
1	wei (웨이)	wei
10^3	babbage (배비지)	kwei
10^6	lovelace (러브레이스)	mwei
10^9	shannon (사넌)	gwei
10^{12}	szabo (사보)	micro ether
10^{15}	finney (피니)	milli ether
10^{18}	ether (이더)	ether

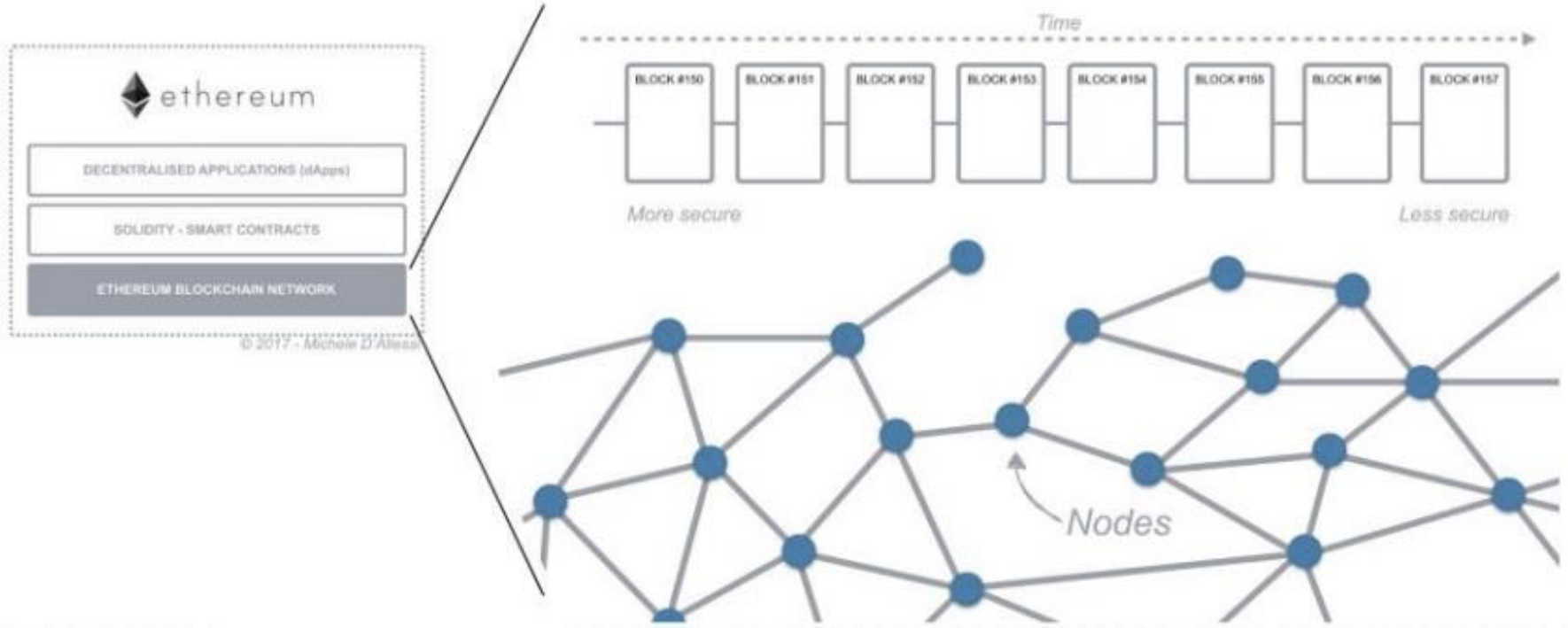
05 이더리움 플랫폼

이더리움의 계층구조

05 이더리움 플랫폼

이더리움의 계층구조

- 이더리움 블록체인 네트워크



05 이더리움 플랫폼

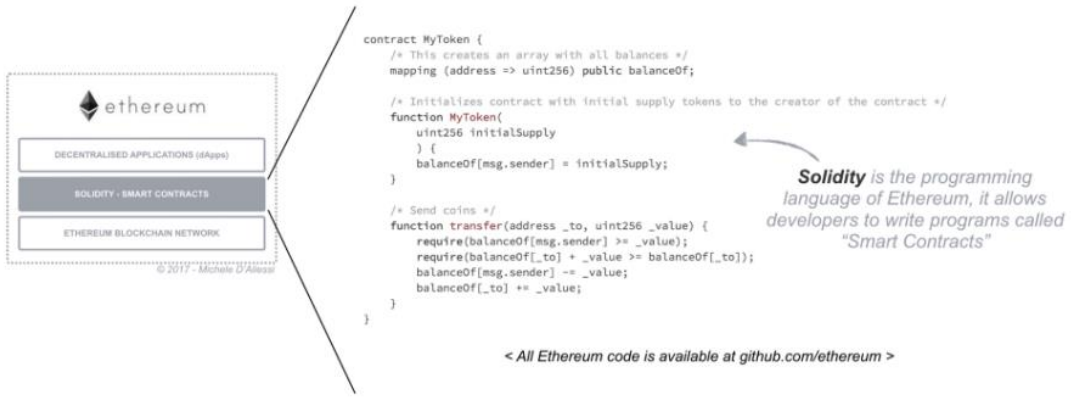
이더리움의 계층구조

- Smart Contract Layer(네트워크 소프트웨어가 올라가는 Layer)

05 이더리움 플랫폼

이더리움의 계층구조

- Smart Contract Layer(네트워크 소프트웨어가 올라가는 Layer)



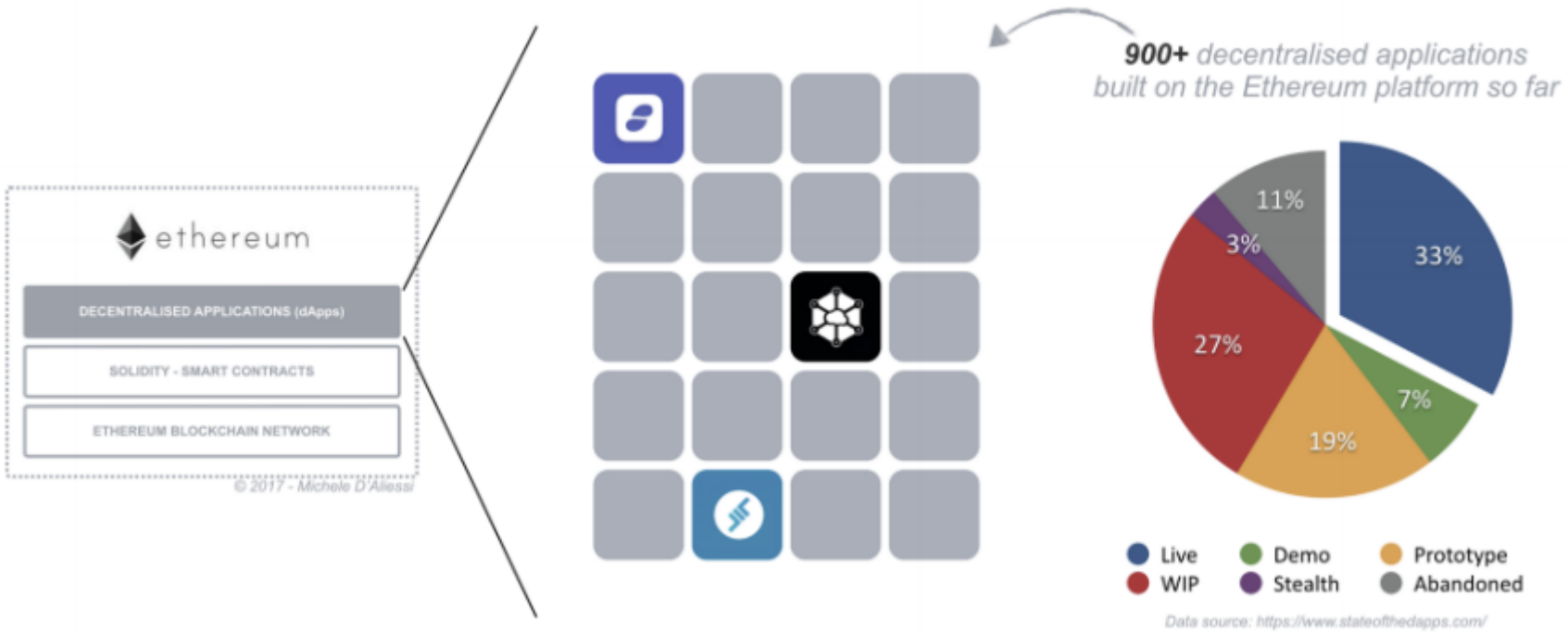
- 이더리움에서 Smart Contract를 작성하기 위한 언어
- 언어의 종류로는 Solidity, Vyper 등이 있음.

05 이더리움 플랫폼

이더리움의 계층구조

- DECENTRALISED APPLICATION LAYER (dApp Layer)

다양한 Front application을 다양한 언어를 통해 작성.

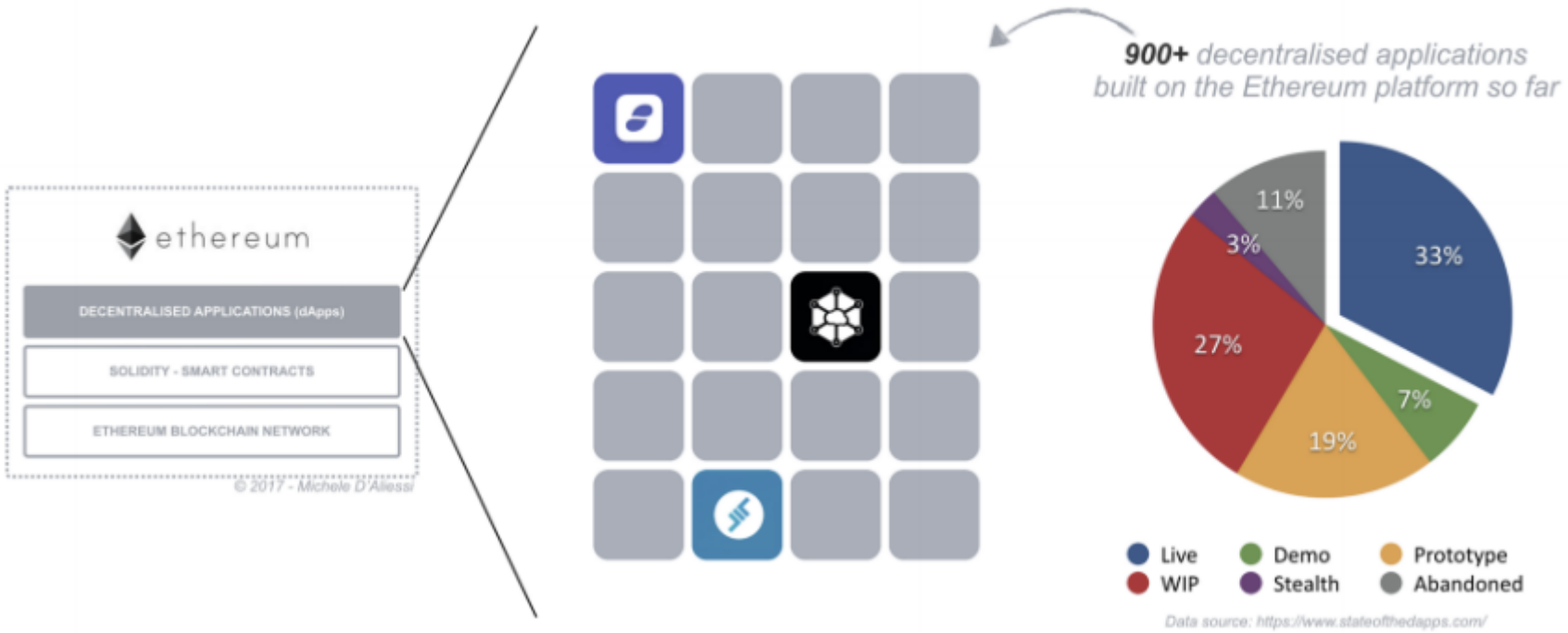


05 이더리움 플랫폼

이더리움의 계층구조

- DECENTRALISED APPLICATION LAYER (dApp Layer)

다양한 Front application을 다양한 언어를 통해 작성.



01 이더리움 주요 개념

01 이더리움 Account

이더리움의 Account (계정)

- 이더리움 송수신에 사용되는 고유한 160bit 주소
- Keccak256(해시함수 -공개키) = ed4f3d … 33dd0ac5f8e4d21040cbb

이더리움 계정의 관리

- 계정에 대한 소유권은 개인키로 증명
- 개인키는 암호화된 특수 파일로 PC에 저장
- 지갑 프로그램은 키를 보관하고 거래 생성, 배포를 도와줌
  - 예) MetaMask, My Ether Wallet 등
  - 지갑을 사용하면 이더리움 설치 없이 이용 가능
  - mnemonic 등을 이용하여 쉽게 이동, 복원이 가능함.

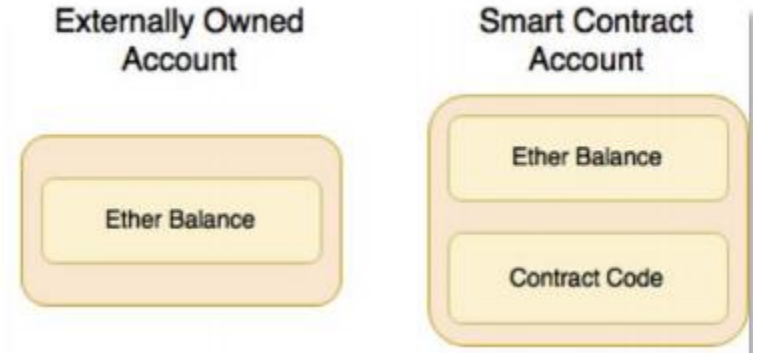
01 이더리움 Account 종류

외부 소유 계좌 (EOA : Externally Owned Account)

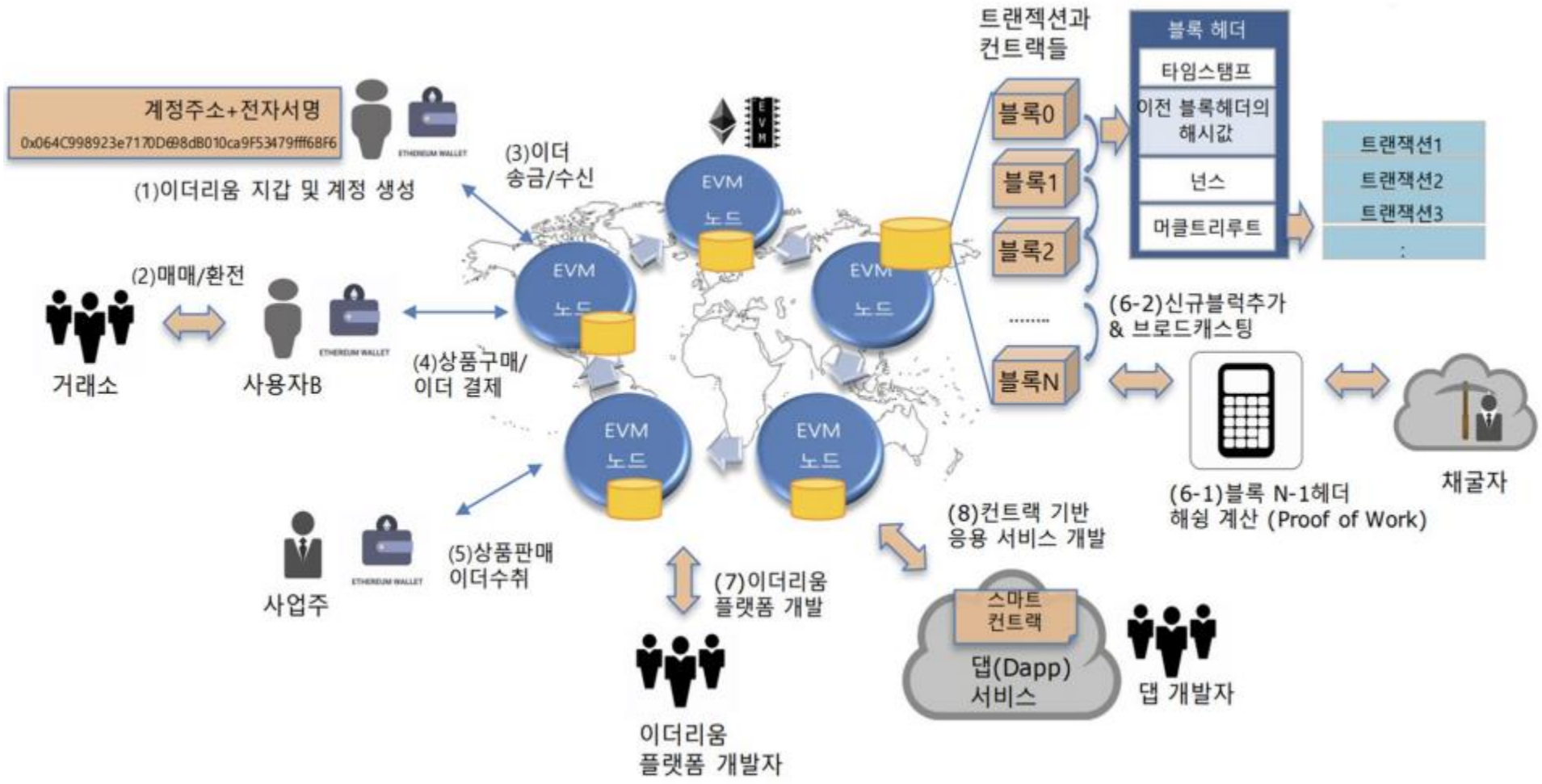
- 개인키 (Private Key) 를 가지고 통제할 수 있는 계좌
- 다른 계좌로의 Ether 송금, 메시지 전송 가능

계약 계좌 (CA: Contract Account)

- EOA + α (Smart Contract Code + Storage)
- 계약 계좌가 특정 메시지를 받게 되면 계좌내 코드가 실행됨.
- 내부 저장공간에 정보를 읽고 쓸 수 있음.



02 이더리움 전개도


계정주소+전자서명
0x064C998923e7170D6f8dB010ca9F53479fff68F6
(1)이더리움 지갑 및 계정 생성
(2)매매/환전
거래소
사용자B
(3)이더
송금/수신
(4)상품구매/
이더 결제
(5)상품판매
이더수취
사업주
EVM
노드
(7)이더리움
플랫폼 개발
이더리움
플랫폼 개발자
트랜젝션과
컨트랙들
블록0
블록1
블록2
.....
블록N
블록 헤더
타임스탬프
이전 블록헤더의
해시값
넌스
머클트리루트
트랜잭션1
트랜잭션2
트랜잭션3
(6-2)신규블럭추가
& 브로드캐스팅
(6-1)블록 N-1헤더
해쉬 계산 (Proof of Work)
채굴자
(8)컨트랙 기반
응용 서비스 개발
스마트
컨트랙
댑(Dapp)
서비스
댑 개발자

03 스마트 컨트랙트

스마트 컨트랙트(Smart Contract)

- 블록체인 안에 저장되는 프로그램
- Solidity로 만들어진 프로그램
- World Computer(Miner)에 의해서 실행되는 프로그램

dAPP (Decentralized Application)

- Smart Contract와 연동되어 동작하는 Application
- Backend가 P2P 분산 네트워크 안에 존재하는 Application
- Front-End 는 일반 Application과 같음.

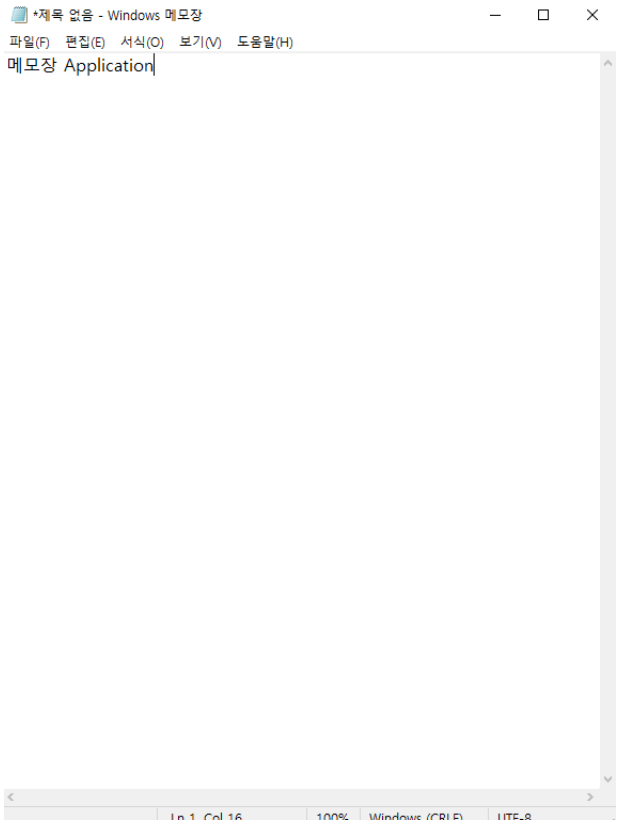
- “소리바다”, uTorrent라는 프로그램도 일종의 dAPP으로 볼 수 있다.

03 스마트 컨트랙트

dApp vs PC Application vs Web Application

핵심 작업의 실행주체?

PC Application

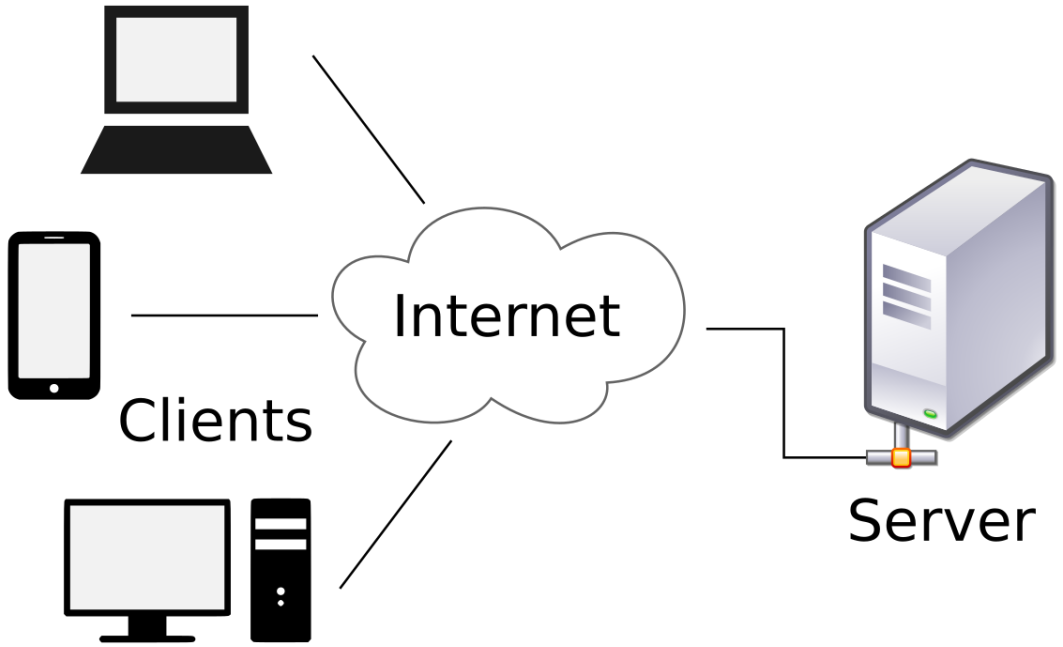


03 스마트 컨트랙트

dApp vs PC Application vs Web Application

핵심 작업의 실행주체?

Web Application



03 스마트 컨트랙트

dApp vs PC Application vs Web Application

핵심 작업의 실행주체?

dApp

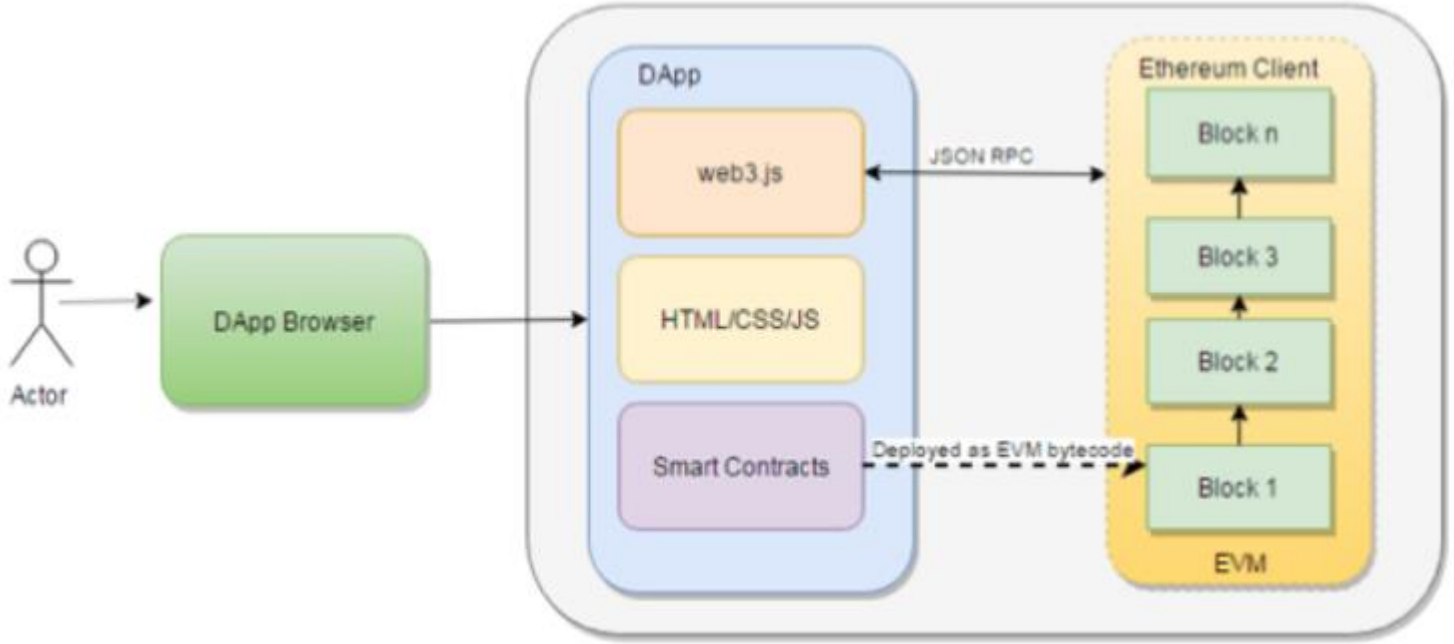
디앱(DApp) 또는 댑이란 Decentralized Application의 약자로서, 이더리움, 큐텀, 이오스 같은 플랫폼 블록체인 위에서 작동하는 **탈중앙화 분산 애플리케이션**을 말한다.
간략히 분산앱이라고도 한다. 플랫폼 위에서 작동하는 **디앱의 암호화폐는 코인(coin)이라고 하지 않고 토큰(token)이라고 구별하여 부르기도 한다**.
‘Dapp’ 또는 ‘dApp’이라고도 쓴다. 단수형이 아니라 복수형으로 표현하여, 디앱스(DApps) 또는 댑스(dApps)라고도 한다.

03 스마트 컨트랙트

dApp vs PC Application vs Web Application

핵심 작업의 실행주체?

dApp



03 스마트 컨트랙트

dApp vs PC Application vs Web Application

핵심 작업의 실행주체?

이더리움

- 작업: Transaction을 처리하는 행위
  TX(A → B, 10ETH), TX(A->B, 10R, 어떤 메시지 기록)
- 작업실행: 발생한 Transaction을 모아 새로은 블록을 만드는 것
- 작업주체: 블록 생성자(= Minor(Node))

➔ 이더리움은 개인(특정 단체)이 아닌 마이너가 핵심 작업을 처리
➔ 요청한 작업이 처리될 때 까지 대기(다음 블록이 생성되고 승인될 때까지)
➔ 작업을 수행하기 위해서 모든 상태 데이터는 BlockChain내에 기록

04 Solidity (솔리디티)

SmartContract를 작성하기 위한 프로그래밍 언어

Solidity

- 2014년 8월 최초 제안
- Smart Contract 지향 프로그래밍 언어 (튜링 완전 언어)
- EVM(Ethereum Vritual Machine)위에서 동작
- Remix, MS Visual Studio 등에서 편집 및 개발 가능

04 Solidity (솔리디티)

SmartContract의 구성요소

데이터 (상태)

- 블록체인의 Smart Contract 안에 담아놓을 자료
  예) 주소록, 매매기록, 자산의 상태 등

함수 (기능)

- Smart Contract 내에 데이터를 조작, 가공하는 기능의 명세
- 일반적으로 생성, 수정, 검색, 삭제 등이 존재

04 Solidity (솔리디티)

토큰

- 특정 비즈니스 조직 등에서 제한된 기능을 가지고 사용되는 재화
- 이더는 이더리움 플랫폼 어디서나 사용 가능.
- 토큰은 특정 Smart Contract 내에서만 사용 가능
- ERC20 표준, ERC233 표준 등등 토큰에 대한 표준이 존재함.

04 Solidity (솔리디티)

```
pragma solidity >=0.4.22 <0.6.0;

contract MyToken {

    mapping (address => uint256) public balanceOf;

    constructor (uint256 initialSupply) public {
        balanceOf[msg.sender] = initialSupply;
    }

    function transfer(address _to, uint256 _value) public returns (bool success){
        require(balanceOf[msg.sender] > _value);
        require(balanceOf[_to] + _value > balanceOf[_to]);
        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;
        return true;
    }
}
```

데이터

함수

05 Smart Contract LifeCycle

LifeCycle?

생애주기(생명주기)
- 어떤 객체가 실행되고 소멸되는 일련의 모든 과정

SmartContract LifeCycle

1. **생성**
   - 컨트랙트 생성 TX(Transaction)에 의해 Ethereum Platform에 배포

2. **실행**
   - 컨트랙트 Call TX에 의해 호출되어 요청받은 기능을 실행
   - 실행됨에 따라 데이터(상태)변화

3. **삭제**
   - 컨트랙트 작성자가 Selef Destruction 기능을 수행하면 코드와 내부 State가 삭제됨.

06 Gas

Gas

이더리움 내의 수수료
- 마이너의 자원(Computing, Storage)을 사용하는 대가로 지불
- Computing을 Gas라는 단위로 측정
- 모든 작업은 자신의 고유한 Gas 소모량 가짐.

Smart Contract를 조심해서 잘 작성해야 하는 이유
➔ 보안문제 + 가스 비용 문제

06 Gas

Gas Limit
- 사용자가 허용하는 최대 Gas 사용량

Gas Price
- 사용자가 지불하는 1Gas 당 가격
- 사용자가 지불할 용의가 있는 최대 수수료

Gas Use
- 실제 소모된 Gas량

Gas Fee
- Gas Use * Gas Price
- 실제 지불될 수수료

Max Tx Fee: (Gas Limit * Gas Price)

07 이더리움의 Transaction

비트코인 vs 이더리움의 TX 비교

비트코인 거래	
From	송신자 주소
To	수신자 주소
Value	보내는 양
Sign	송신자의 서명

이더리움 거래	
From	송신자 주소
To	수신자 주소
Value	보내는 이더의 양
Init	컨트랙트 생성에 사용
Data	컨트랙트 호출에 사용
Sign	송신자의 서명

07 이더리움의 Transaction

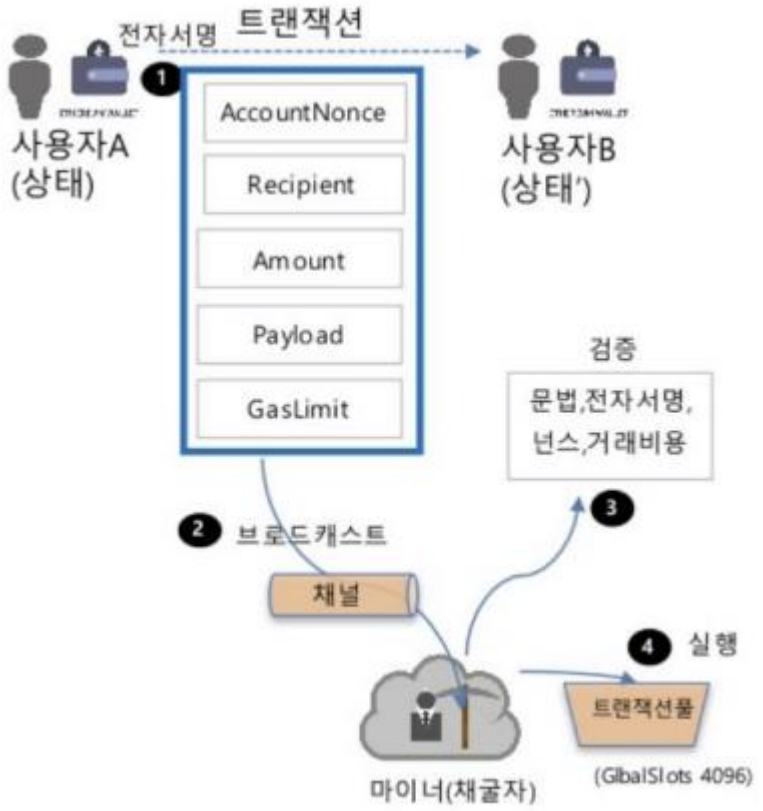
작업에 따른 TX 내용

거래 내용	일반 송금	컨트랙트 생성	컨트랙트 호출
From	O	O	O
To	O	0x00	O
Value	O	컨트랙트에 송금 (최초 자본금)	컨트랙트에 송금
Init	X	컨트랙트의 코드	X
Data	담고 싶은 메시지 (선택)	X	컨트래트 호출에 사용될 파라미터
Sign	O	O	O

Init 과 Data는 둘 중 하나만 포함 가능.

07 이더리움의 Transaction

이더리움의 Transaction 과정

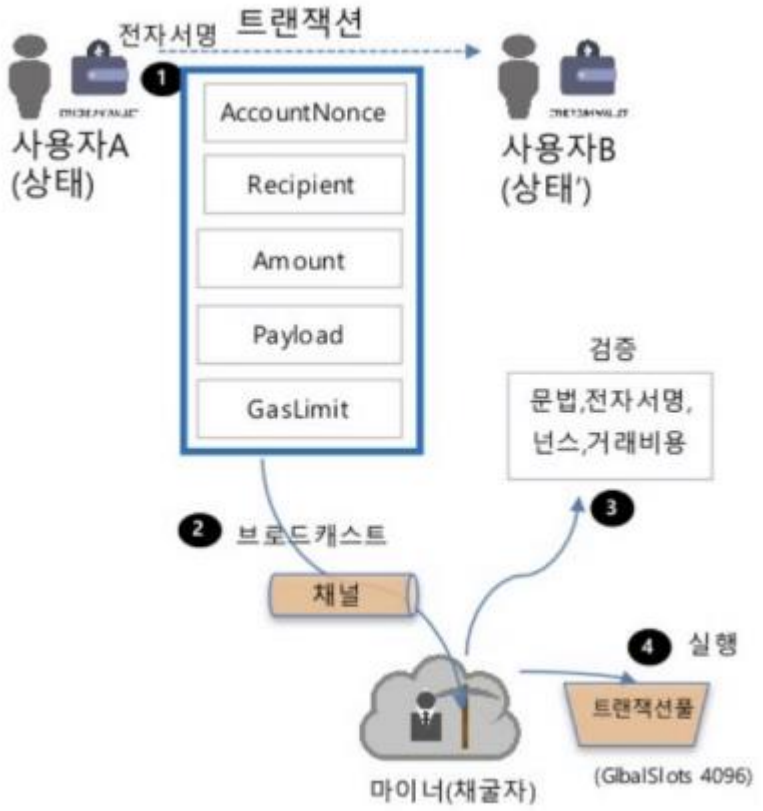


[A → B] Transaction

1. 사용자 A는 전자서명 후 사용자 B에게 송금하는 TX 실행
2. 해당 TX는 마이너를 포함 모든 노드들에게 BroadCasting

07 이더리움의 Transaction

이더리움의 Transaction 과정

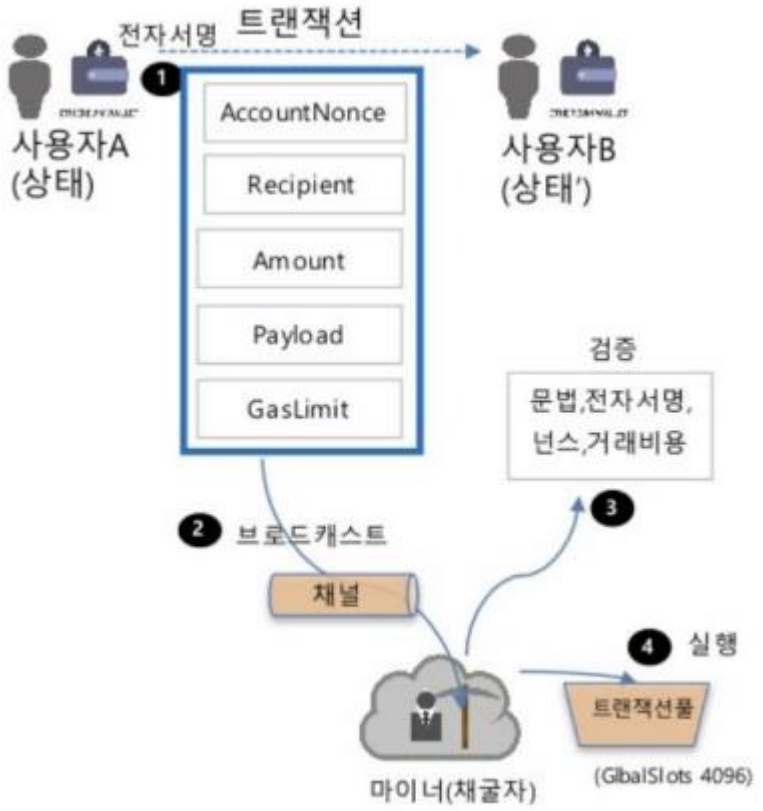


[A → B] Transaction

3. 마이너는 채널을 통해 TX를 전달 받은 후 유효성 검사(문법 체크, 서명 검증, Nonce 검증) 후 TX 비용 계산, 수신자 B의 주소 유효성 검사

07 이더리움의 Transaction

이더리움의 Transaction 과정



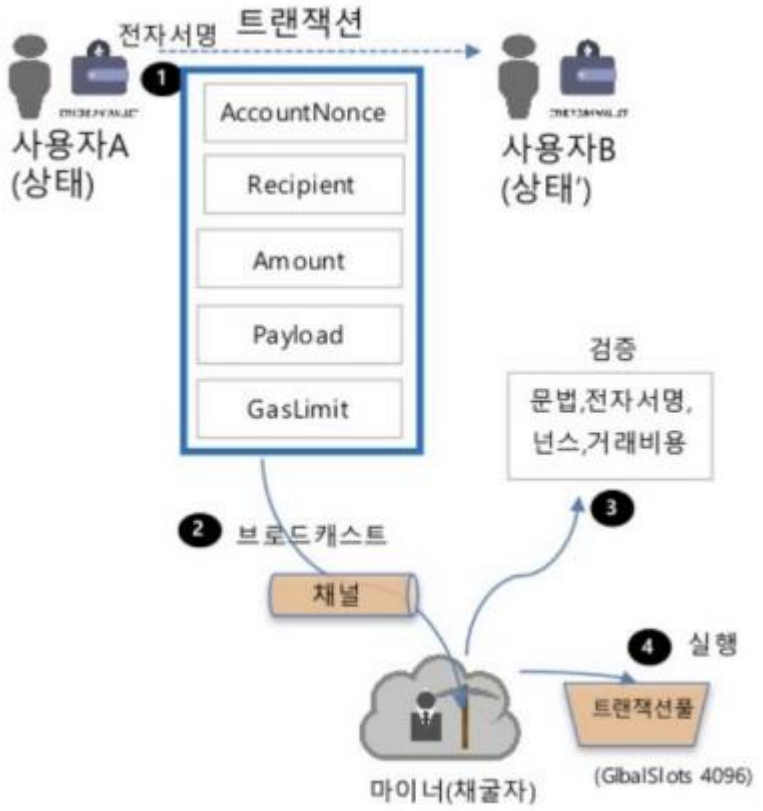
[A → B] Transaction

4. 사용자 A의 계정 잔액에서 TX 비용 빼고, AccountNonce 1증가시킴,

- 만약 잔액이 충분하지 않을 경우 에러 반환
- 오류 없을 시 gas used * gas price 계산 및 실행을 위해 Transaction Pool에 등록

07 이더리움의 Transaction

이더리움의 Transaction 과정



[A → B] Transaction

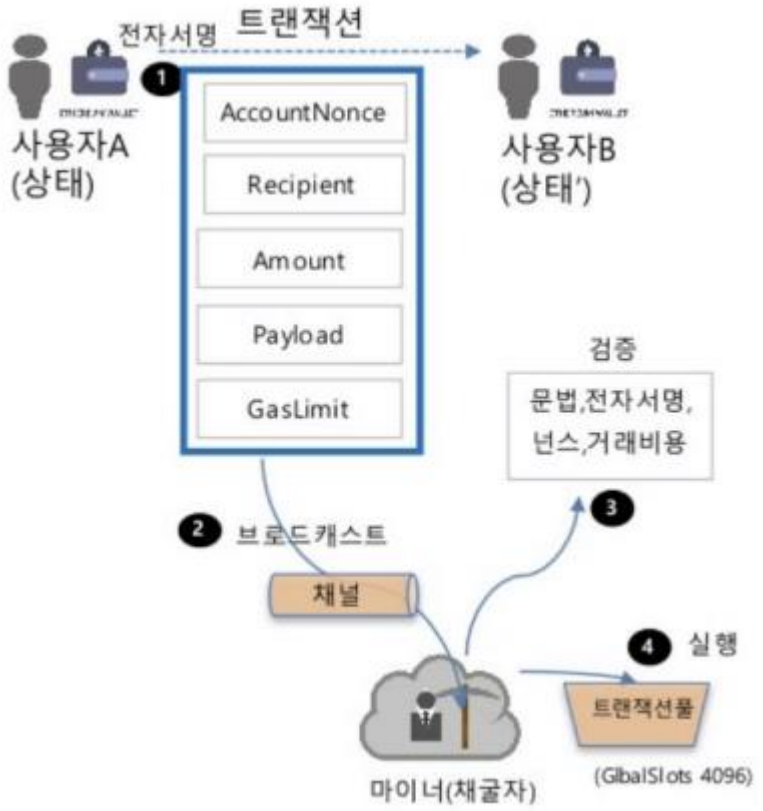
5. 마이너는 Transaction Pool에서 Transaction 처리비용이 높은 순으로 Transaction을 처리

- B계정이 Contract Account일 경우 해당 Contract Code를 실행
- B계정이 EOA(일반 Account)일 경우 송금 처리

➔ SmartContract일 경우 Contract가 완료되거나 실행비용이 모두 소진될 때 까지 실행.

07 이더리움의 Transaction

이더리움의 Transaction 과정



[A → B] Transaction

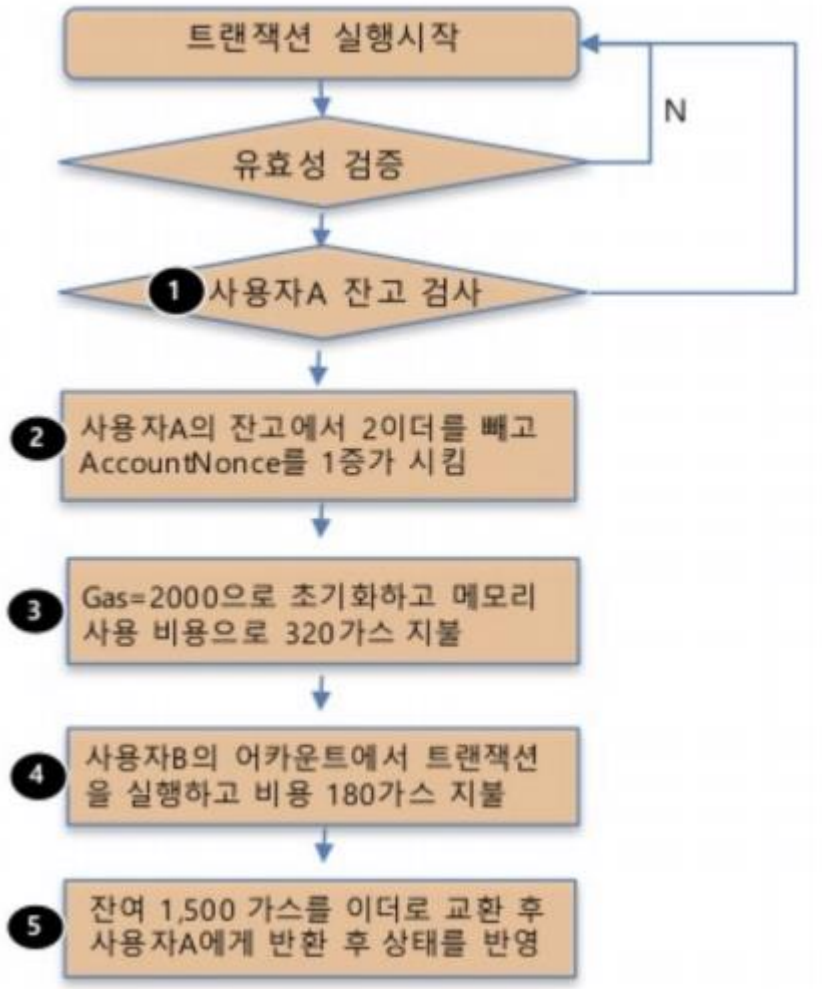
만약 사용자 A에게 충분한 양의 Account가 없거나 정해진 가스가 모두 소비되어 코드 실행을 할 수 없을 경우 등의 이유로 TX 처리가 실패한다면?

➔ TX 처리 비용을 제외하고 모든 상태를 원래대로 rollback(원상복구)하고, TX 처리 대가를 마이너 계정에 추가함

➔ TX가 성공하면 처리 결과를 State에 반영하고 남은 모든 Gas를 이더로 환산 후 A에게 반환 + TX 비용을 마이너에게 전송

07 이더리움의 Transaction

이더리움의 Transaction 과정



08 비트코인 vs 이더리움

비트코인 블록 구조

구분명	내용
버전(Version)	소프트웨어 버전
비츠(Bits)	난이도
이전 블록 해시(Prev Block Hash)	이전 블록 해시
시간(Time)	블록 생성 시간
머클 해시(Merkle Hash)	거래 정보 해시
논스(Nonce)	난수[2]
바디(Body)	

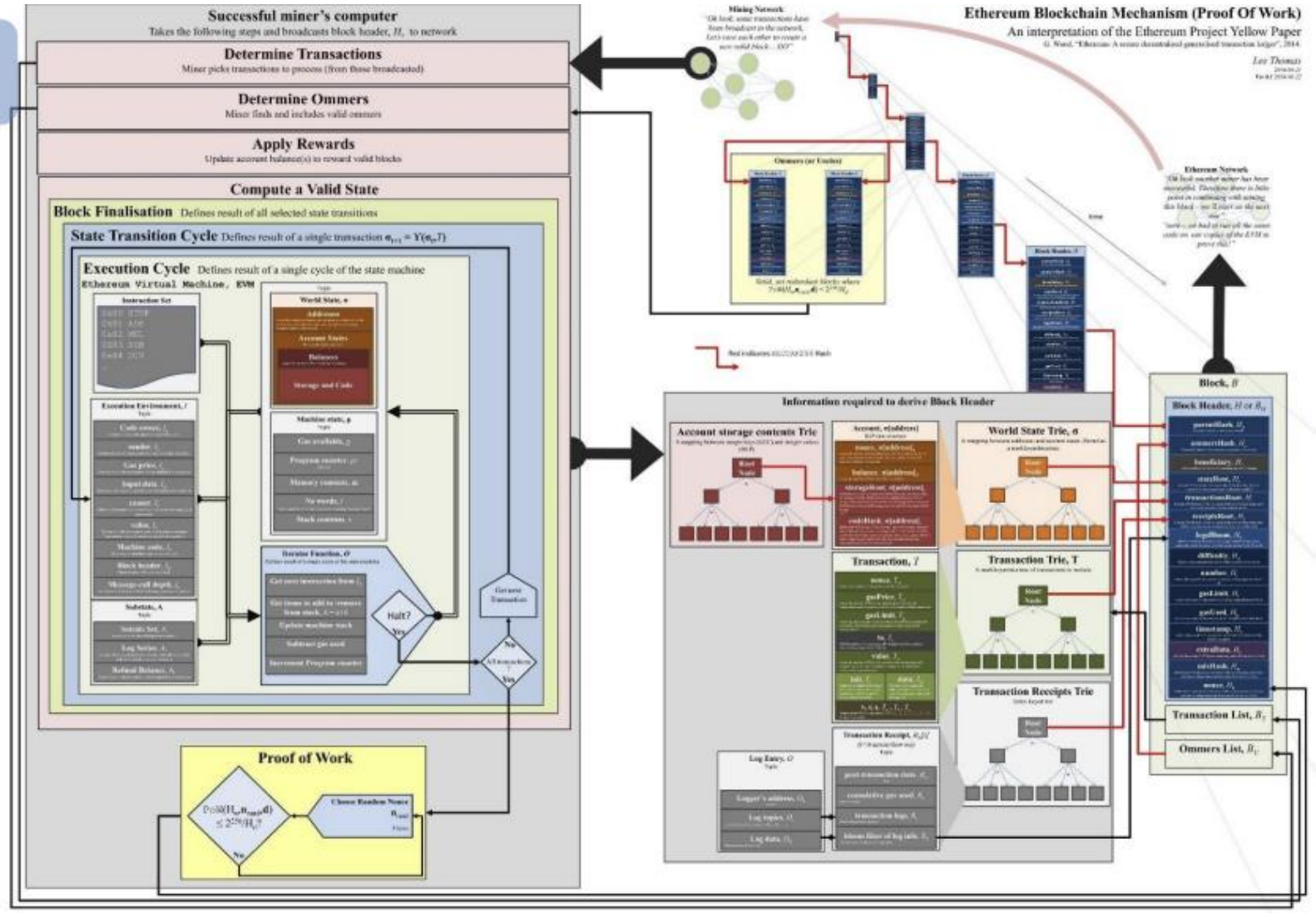
이더리움 블록 구조

- 이더리움 블록헤더의 구성요소
  1. 부모해시(parentHash) : 부모 블록의 해시값
  2. 엉클해시(uncleHash) : 현재 블록의 엉클 블록들의 해시값
  3. 주소(beneficiary) : 채굴 후 해당 트랜잭션의 수수료를 받을 계정 주소
  4. 상태루트(stateRoot) : 계정의 상태정보가 모여있는 머클 패트리시아 트리의 루트 노드 해시값
  5. 트랜잭션루트(transactionsRoot) : 블록의 모든 트랜잭션에 대한 머클트리의 루트노드 해시값
  6. 영수증루트(receiptsRoot) : 해당 블록 내 모든 트랜잭션에 대한 일종의 영수증 머클트리의 루트노드 해시값
  7. 로그블룸(logsBloom) : 로그 정보를 사용하는데 필요한 32바이트 블룸필터
  8. 난이도(difficulty) : 블록 생성 난이도
  9. 블록번호(number) : 해당 블록 번호
  10. 가스한도(gasLimit) : 블록 당 지급 가능한 최대 가스의 제한량
  11. 사용된 가스(gasUsed) : 해당 블록 내 트랜잭션에 사용된 가스의 총합
  12. 타임테이블(time : 해당 블록의 최초 생성시간
  13. 믹스해시, 논스(mixHash, nonce : 해당 블록이 충분한 연산을 했음을 입증하는 해시값
  14. 기타(extra) : 블록의 기타 정보

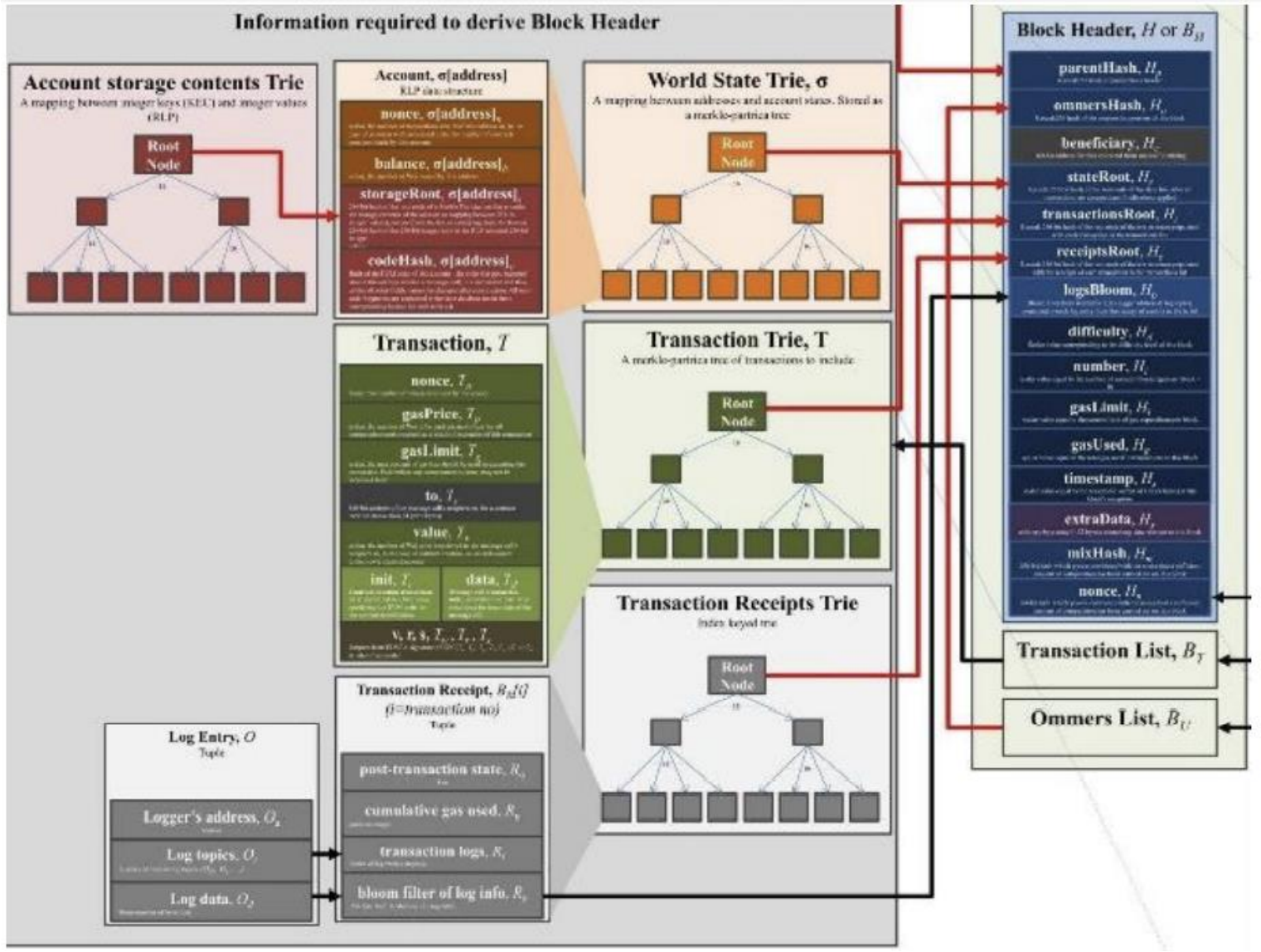
01 이더리움 구조

01 이더리움 구조

이더리움 구조도



01 이더리움 구조



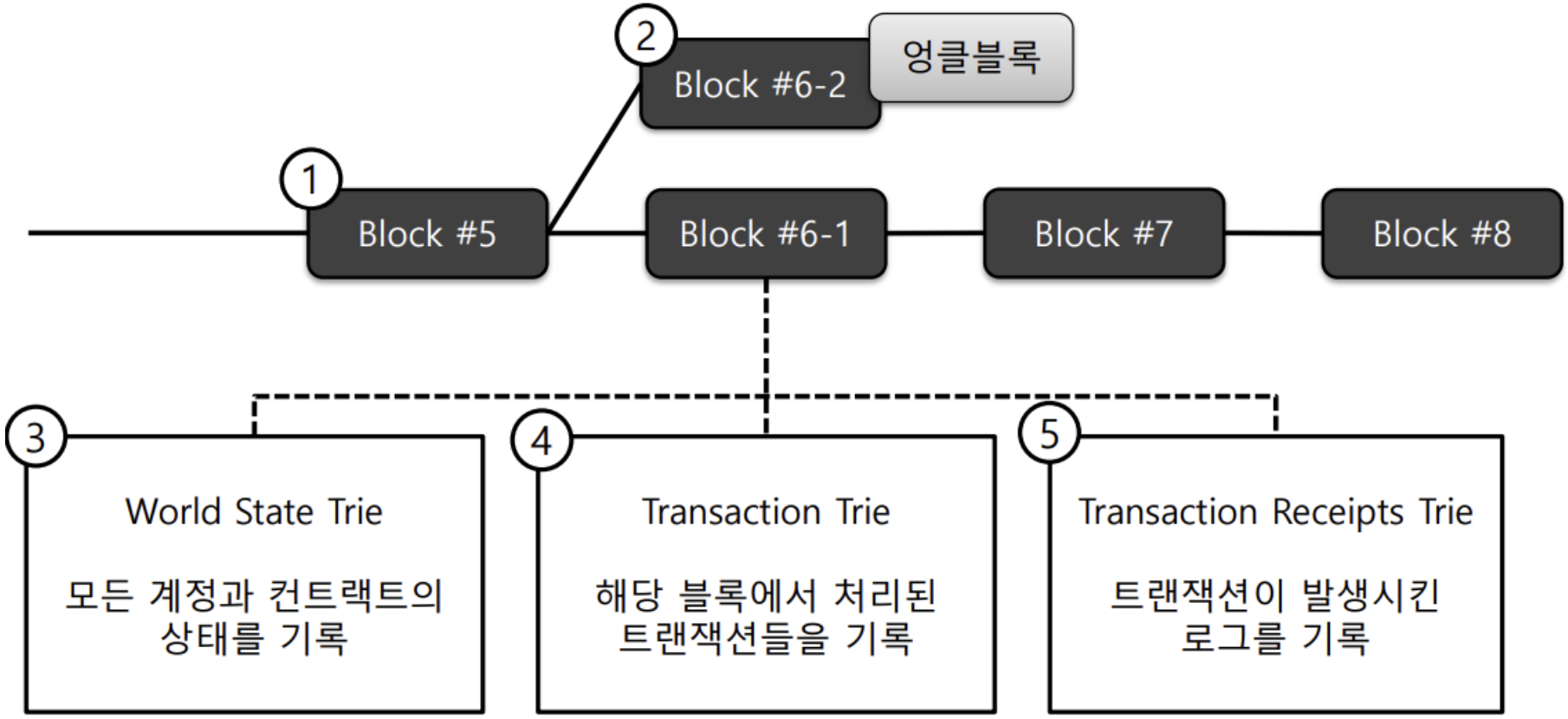
Ommer:
orphaned blocks (현재 블록의 부모와 동일한 부모를 두고 있는 블록)

→ 빠른 블록 생성주기(15초 이하)➔ 경쟁 심화 ➔ 블록의 충분한 공유가 되기 전에 발견된 새로운 블록

→ GHOST(Greedy Heaviest Observed Subtree) 프로토콜로 해결

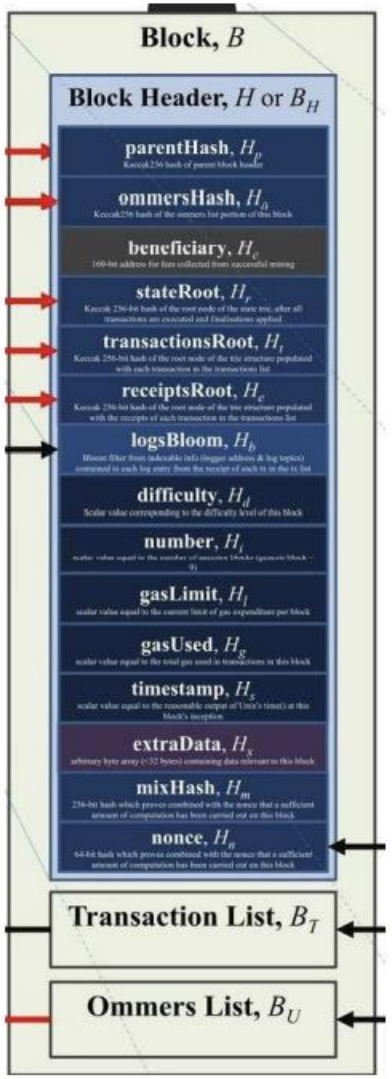
→ 가장 긴 X => 가장 무거운

01 이더리움 구조



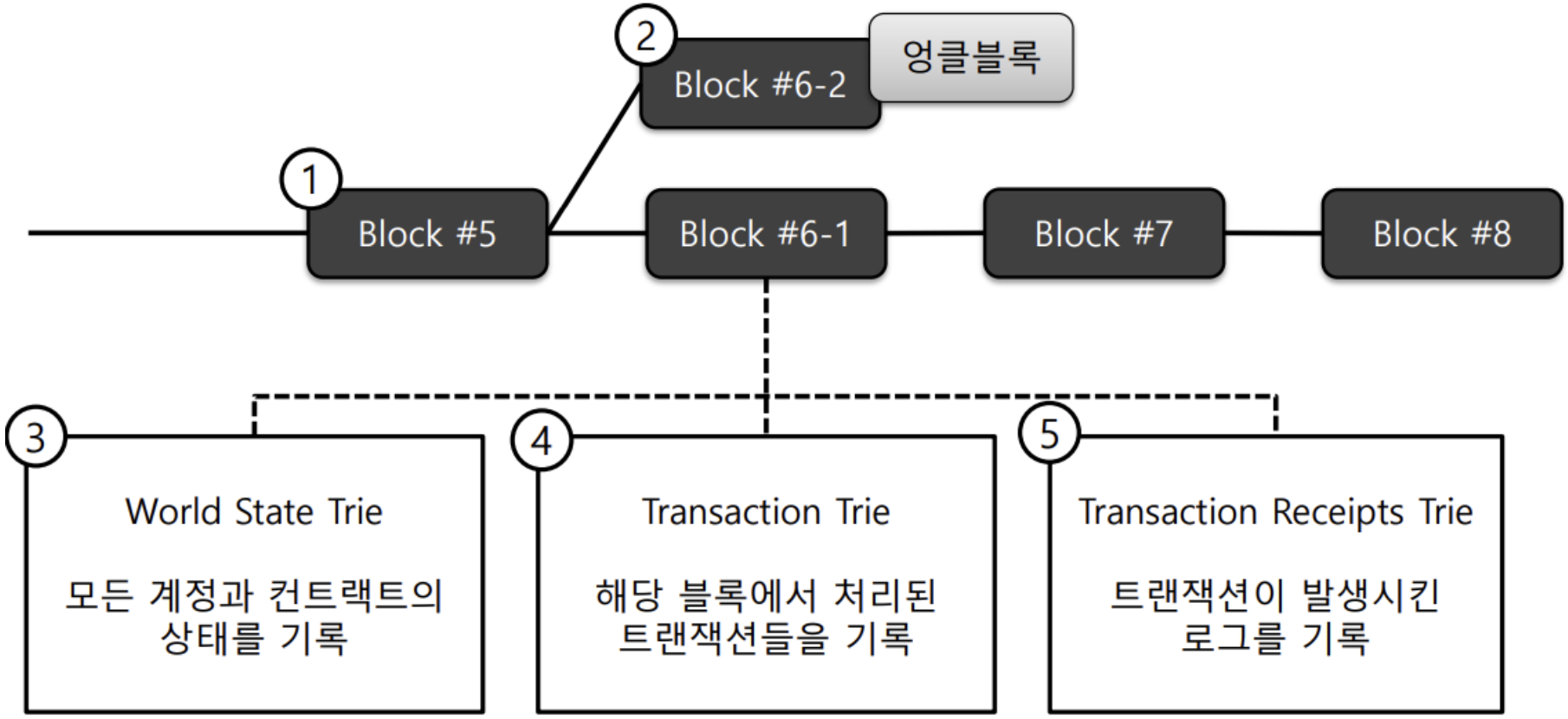
이더리움 블록체인 네트워크 구성요소

01 이더리움 구조



항 목	설 명
Prev Hash	이전 블록의 해시 값을 지칭(Parent Hash)
Nonce	작업증명 시 이용되는 64비트 해시로써, 충분한 양의 계산을 위해 이용
Timestamp	유닉스 time() 함수의 출력 값으로 생성 시간을 의미
Uncles Hash (ommer[1]shash)	블록의 엉클 목록의 SHA-3 해시(256비트)
Beneficiary	블록의 채굴 성공 시 모든 수수료(Fees)가 전송되는 160비트 주소
Logs Bloom	블룸필터(Bloom filter)[2]는 거래 목록의 각 거래 영수증에서 각 로그 항목에 포함된 색인 정보(로그 주소 및 주제를 기록)로 구성
Difficulty	블록의 난이도에 해당되는 값으로 이전 블록의 난이도 및 타임스탬프로부터 계산
Extra Data	블록과 관련된 데이터를 포함하는 임의의 바이트 배열
Block Num	상위 블록의 수
Gas Limit	블록 당 가스 비용을 제한하는 값
Gas Used	블록에서 트랜잭션에 사용된 총 가스 값
Mix Hash	충분한 양의 계산을 위해 블록에 수행되는 넌스와 작업 증명을 하는데 이용되는 256비트 해시
State Root	모든 트랜잭션이 실행된 후 완결 짓는데 적용되는 상태 트리의 루트 노드(SHA-3 해시 값)
Transaction Root	블록의 거래 목록 부분에 각 트랜잭션으로 채워진 상태 트리의 루트 노드(SHA-3 해시 값)
Receipt Root	블록의 거래 목록 부분에 각 거래 영수증으로 채워진 상태 트리의 루트 노드(SHA-3 해시 값)

01 이더리움 구조



메인 블록 보상: 기본 보상 (2Eth) + 수수료
엉클블록 보상: 1.75ETH (기본 보상의 7/8), 1.5ETH, …

01 이더리움 구조

엉클 블록이 많아지면..?

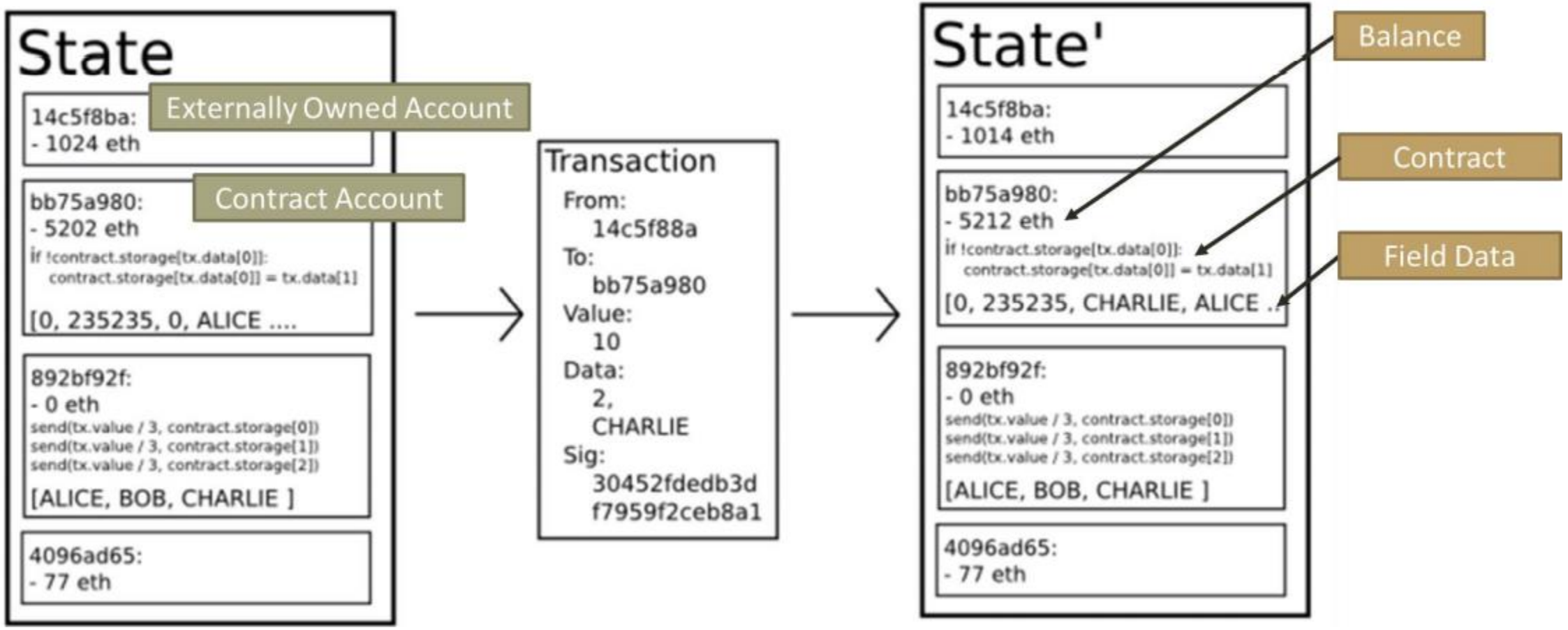
- **TX 처리를 지연**
  동시에 두 마이너가 블록을 생성하면 서로 다른 블록들이 전파.
  서로 다른 TX이 처리
  포함되지 않은 TX는 처리X

- **Computing Power 낭비**
  엉클블록의 생성과정상 불필요한 Computing Power 낭비

- **보안문제**
  블록체인의 폭이 넓어진다 → 컴퓨팅 파워 강한 마이너에 의해 독점 가능

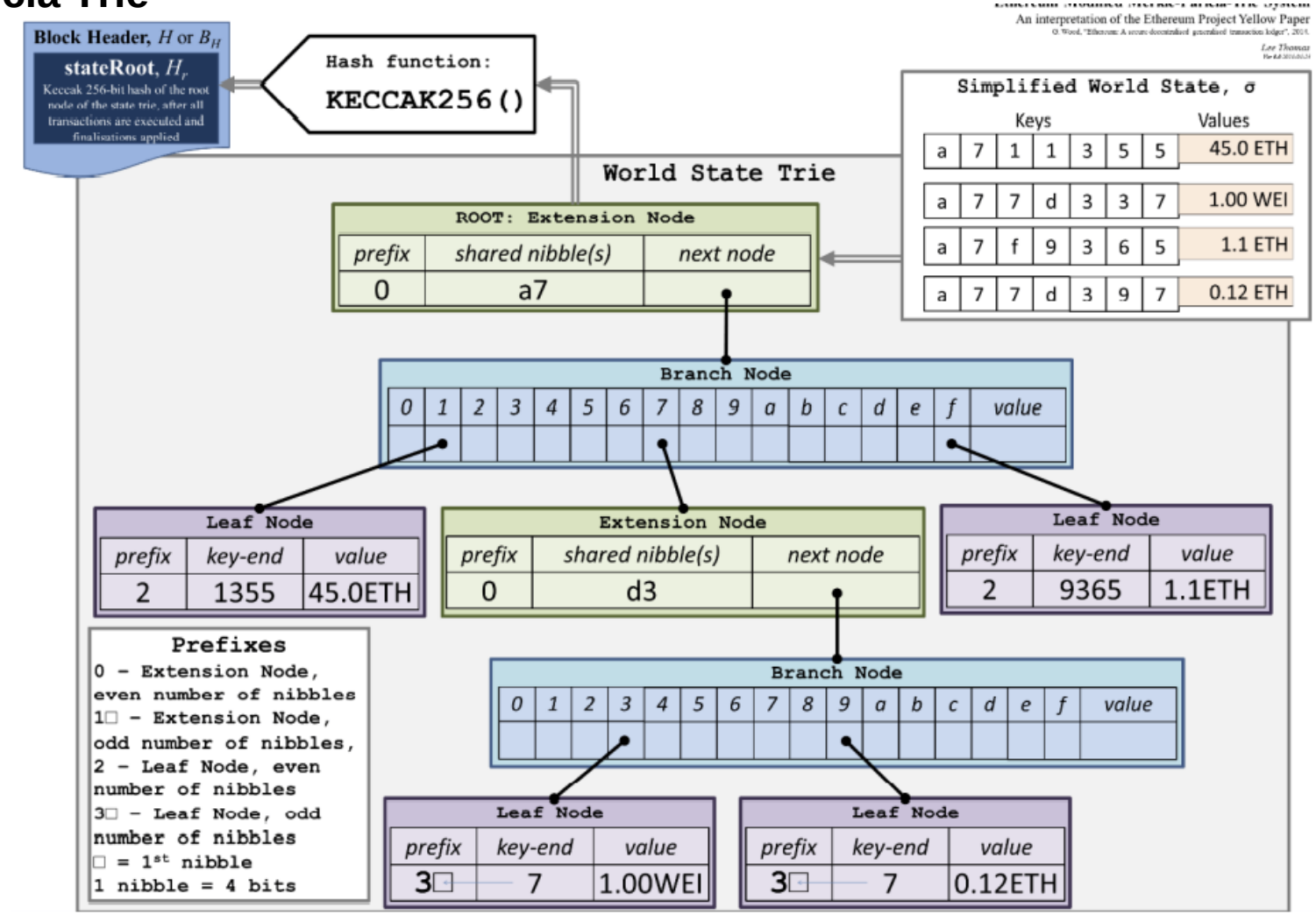
01 이더리움 구조

World State Trie



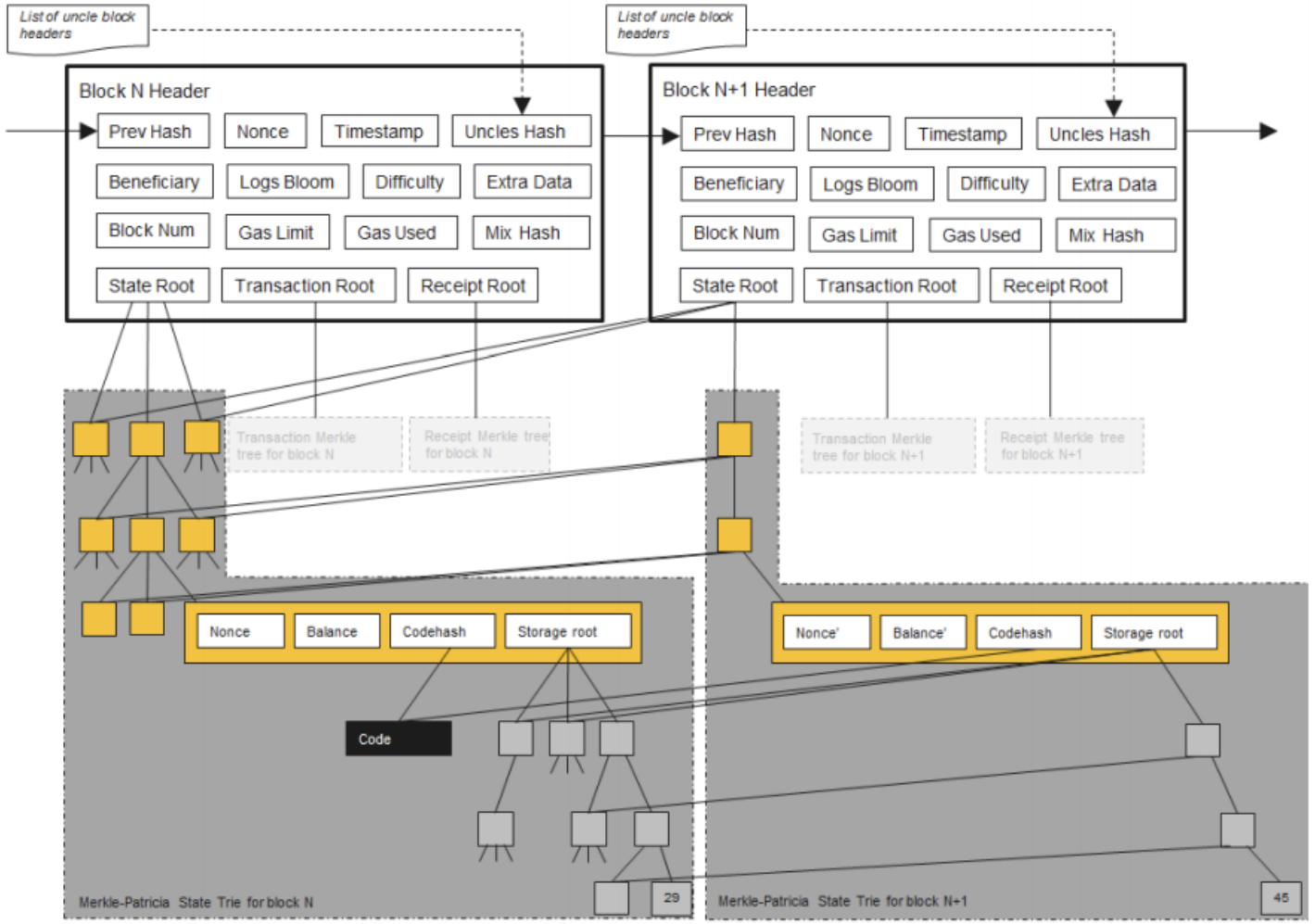
01 이더리움 구조

Merkle Patrcia Trie



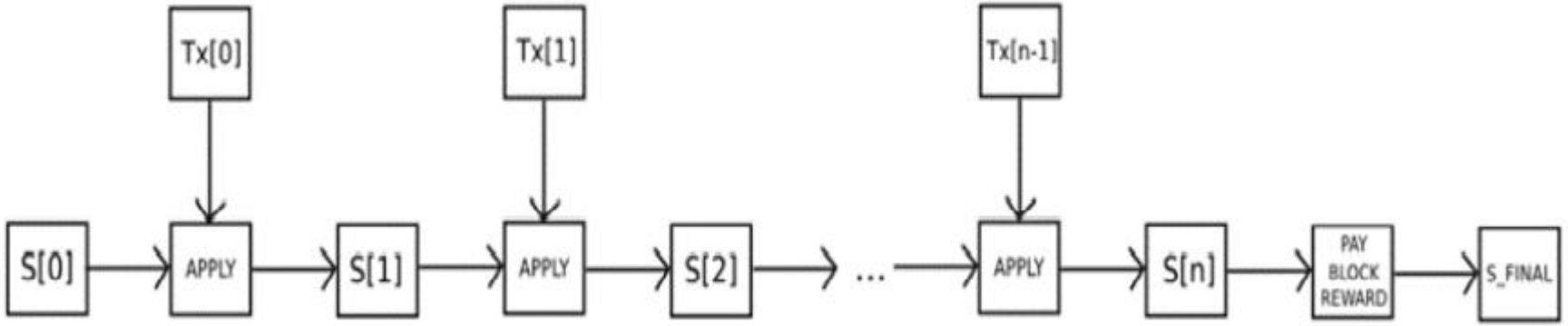
01 이더리움 구조

Merkle Patrcia Trie



01 이더리움 구조

Transaction Trie



- S[0]: 초기 상태 거래 내용
- Tx[0], Tx[1], … : 을 적용하여 상태 업데이트
  --> S[1], S[2], …
- 상태 변화에 따른 Balance 기록
- S_FINAL 상태의 Account 내용이 담긴 블록 배포
- Block에서 처리된 Tx들이 머클트리 형태로 Tx Tree에 기록

01 이더리움 구조

Receipt

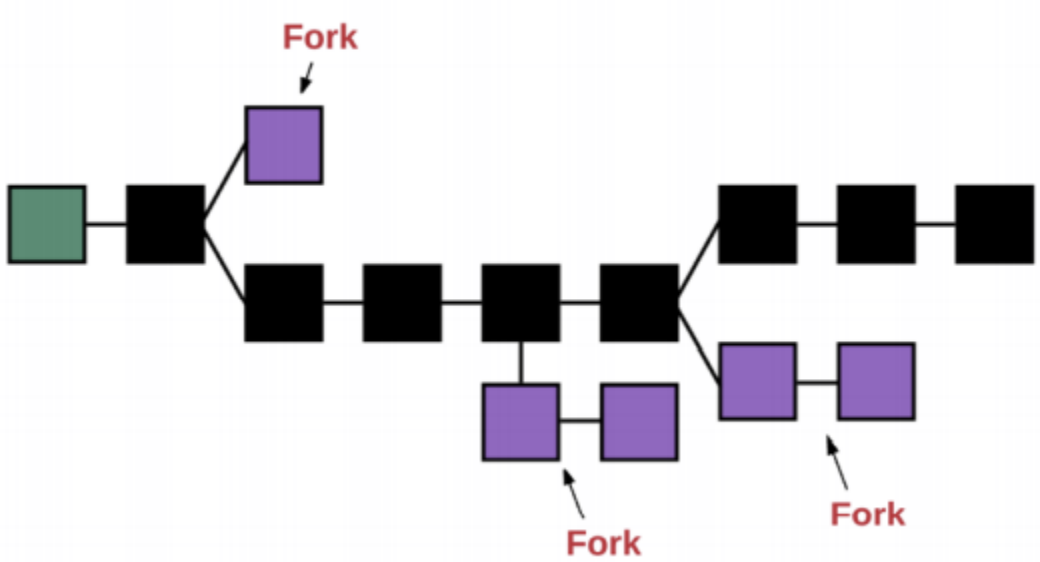
```
// Receipt represents the results of a transaction.
type Receipt struct {
    // Consensus fields
    PostState         []byte `json:"root"`
    Status            uint   `json:"status"`
    CumulativeGasUsed uint64 `json:"cumulativeGasUsed" gencodec:"required"`
    Bloom             Bloom  `json:"logsBloom"         gencodec:"required"`
    Logs              []*Log `json:"logs"              gencodec:"required"`

    // Implementation fields (don't reorder!)
    TxHash          common.Hash    `json:"transactionHash" gencodec:"required"`
    ContractAddress common.Address `json:"contractAddress"`
    GasUsed         uint64         `json:"gasUsed" gencodec:"required"`
}
```

- PostState : 트랜잭션이 실행된 후의 상태정보
- Status : 트랜잭션 실행후의 상태결과
- CumulativeGasUsed : 트랜잭션이 포함된 블록에서 사용된 전체 가스비용
- Bloom : 로그정보를 빠르게 검색하기 위한 블룸필터
- Logs : 생성된 로그
- TxHash : 트랜잭션 hash id
- ContractAddress : 스마트 컨트랙트에서 생성된 트랜잭션인 경우 해당 스 마트 컨트랙트의 주소
- GasUsed : 트랜잭션 실행에 사용된 가스비용

01 이더리움 구조

Ghost 프로토콜 (Greedy Heaviest Observed Subtree)



이더리움에서도 다른 블록체인처럼 포크가 발생한다.
이 때 어떤 path가 가장 유효한지 선택하여 여러 체인들이 생겨나는 것을 방지하기 위해서 이더리움은 **GHOST 프로토콜**이라는 메커니즘을 사용

➔ 블록생성이 짧으므로 많은 블록들이 존재.
➔ 가장 큰 무게를 가진 subtree 선택
➔ 가장 큰 difficulty를 가진 가장 최신의 블록넘버