

이더리움 개요

1. 이더리움의 탄생

- 1) 이더리움의 정의
- 2) 이더리움의 역사
- 3) 이더리움인 근황

2. 이더리움 동작과정

- 1) 소프트포크, 하드포크
- 2) 블록 구조
- 3) 구성 요소
- 4) 이더리움 도구

3. 이더리움의 기술적 요소

- 1) 키, 주소, 지갑
- 2) 합의 알고리즘
- 3) 스마트 컨트랙트 가상머신
- 4) 스마트 컨트랙트

이더리움 공식 홈페이지 (영문 Version)

Ethereum is open access to digital money and data-friendly services for everyone — no matter your background or location.

It's a community-built technology behind the cryptocurrency ether (ETH) and thousands of applications you can use today.

이더리움은 배경이나 위치에 상관없이 모든 사용자가 디지털 화폐나 데이터 친화적인 서비스에 자유롭게 접근할 수 있습니다

이더리움은 공동체 기반의 기술로서, 오늘날 당신이 사용할 수 있는 수천개의 애플리케이션과 암호화폐 이더기반으로 구성되어 있습니다

이더리움 공식 홈페이지 (한국어 Version)

이더리움은 새로운 인터넷 시대를 위한 기반입니다.

- 화폐와 결제 수단이 내장된 인터넷
- 사용자가 자신의 데이터 주권을 가지며 애플리케이션이 나의 고유한 데이터를 탈취하지 않는 인터넷
- 누구나 열린 금융 시스템에 접속할 수 있는 인터넷
- 특정 회사나 개인에 의해 통제되지 않으며, 독립적이고 개방된 인프라를 갖춘 인터넷

업비트 소개

- 이더리움(Ethereum)은 2013년 비탈릭 부테린(Vitalik Buterin)에 의해 탄생하였습니다
- 이더리움은 스마트 컨트랙트를 위한 분산 네트워크로 현재 많은 디앱(DApp)들이 이더리움 네트워크를 기반으로 하고 있습니다
- 또한 ERC-20이라는 이더리움 네트워크에서 탄생한 토큰들의 표준을 제시하여 토큰간 호환성도 증진시켰습니다
- 현재 ERC-721 기반의 수많은 NFT 또한 이더리움 네트워크를 기반으로 탄생하고 있습니다
- 이더리움(ETH)은 이더리움 네트워크에서 화폐처럼 사용되는 디지털 자산입니다.
- 이더리움은 비트코인 다음으로 시가총액이 세계에서 가장 큰 디지털 자산이며, 스마트 컨트랙트 기능의 도입으로 블록체인 2세대의 시작을 열었습니다
- 이더리움 플랫폼 위에서 수많은 비즈니스 앱이 탄생하였으며, 이더리움은 이들이 자유롭게 활동할 수 있는 분산 네트워크를 지향하고 있습니다
- NFT 시장의 확대로 이더리움의 가능성은 더 주목을 받고 있습니다. 앞으로도 많은 기능들이 이더리움에 추가될 예정이며, 이 과정에서 발생하는 비효율성의 문제를 해결하기 위해 활발한 개발이 이루어지고 있습니다.

이더리움의 역사



2013년 12월

- 이더리움 백서 공개

2015년 7월

- Frontier 패치

2016 3월

- Homestead 패치

2016년 7월

- DAO 패치

2017년 10월

- Byzantine 패치

2019 2월

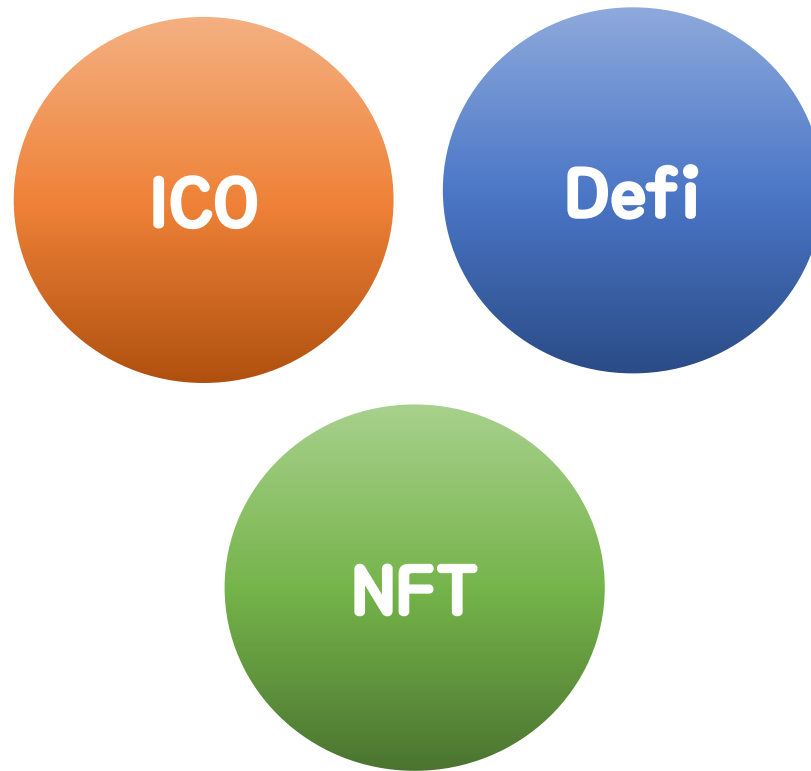
- Constantinople 패치

2021년 4월

- Berlin 패치

2021년 8월

- London 패치



디지털 자산의 소유권을 증명하는 NFT

- 기존의 현물 거래는 거래 기록을 확인 가능
- 기존의 디지털 자산은 손쉬운 파일 복사로 인해 소유주 증명 불가능
- NFT는 블록체인을 통해 거래 내역을 증명할 수 있으므로 소유주 증명 가능
- 기존에는 거래가 불가능했던 상품에 대한 새로운 거래 시장 형성

NFT의 활용

1. 디지털 예술품 거래 시장



CryptoPunk #7523



JA MORANT DUNK Legendary #42/49

- 디지털 예술품 거래가 이루어지는 시장

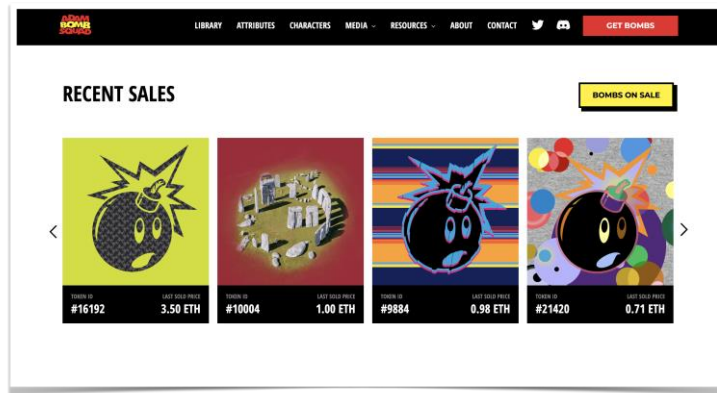
2. 커뮤니티 멤버십



- NFT 작품 시리즈를 가지고 있는 사람들끼리 모임 생성

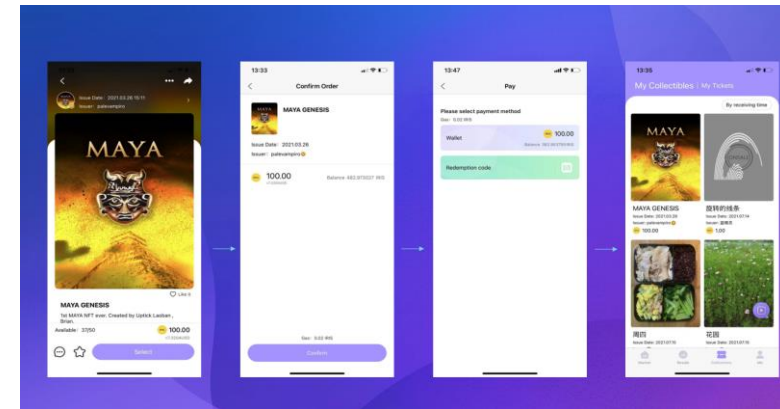
NFT의 활용

3. 브랜드 세계관 확장



- 유명 브랜드가 자신들의 팬덤 및 세계관 확장을 위해 NFT 시리즈를 발행

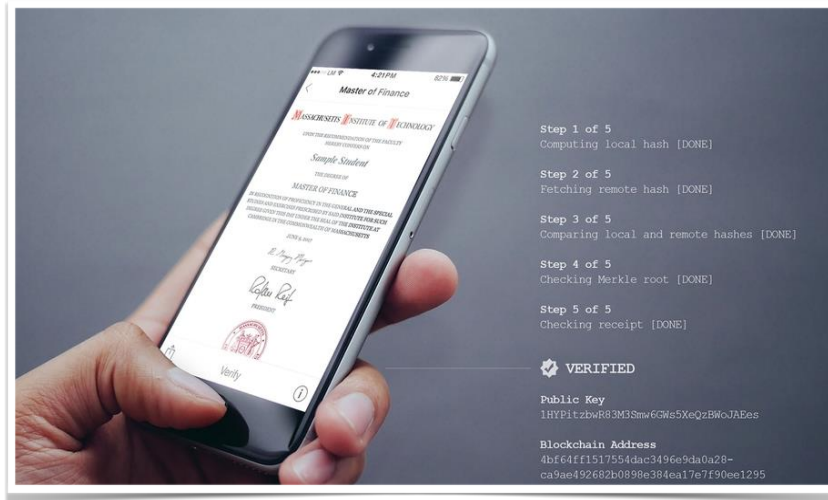
4. 실생활 응용



- 레스토랑 예약권, 멤버십 카드
- 공연 티켓을 NFT로 판매해 굿즈로서의 소장 욕구 자극

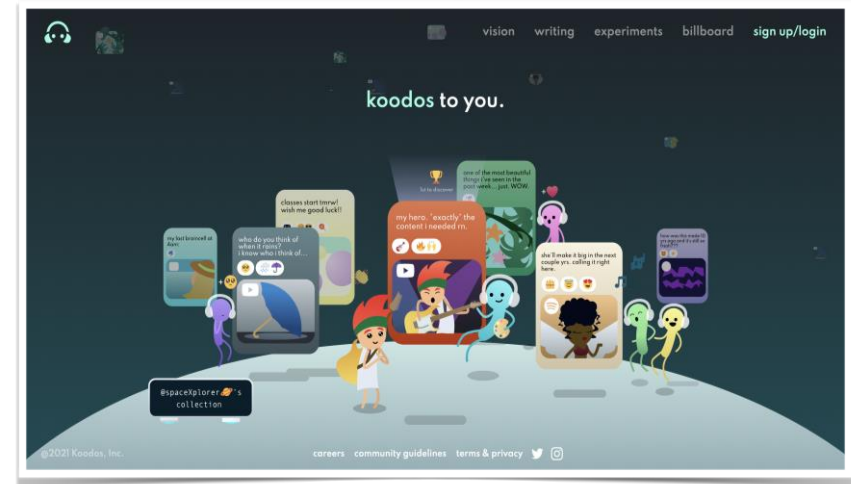
NFT의 활용

5. 양도 불가 NFT



- MIT의 새로운 졸업장 배포

6. NFT 발행 플랫폼



- NFT를 쉽게 발행 및 공유 가능한 서비스 제공

1. 이더리움의 탄생

- 1) 이더리움의 정의
- 2) 이더리움의 역사
- 3) 이더리움인 근황

2. 이더리움 동작과정

- 1) 소프트포크, 하드포크
- 2) 블록 구조
- 3) 구성 요소
- 4) 이더리움 도구

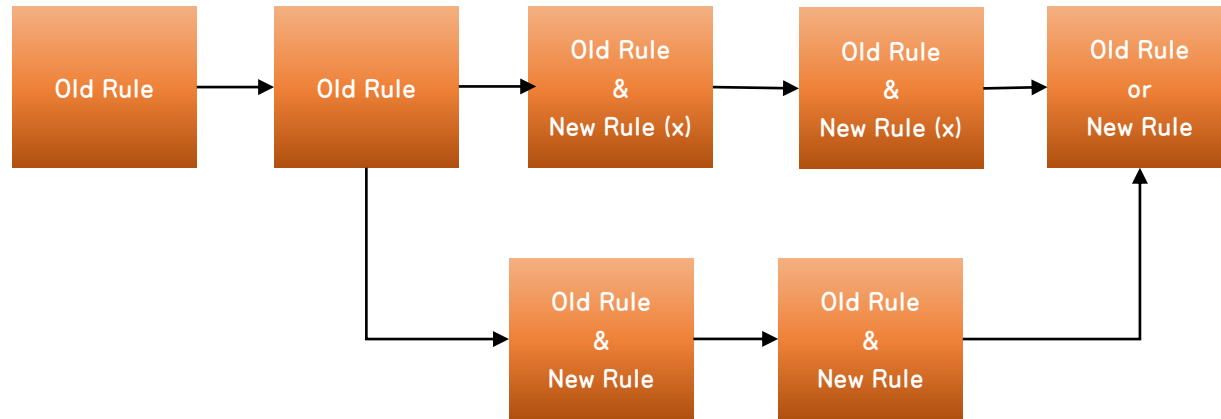
3. 이더리움의 기술적 요소

- 1) 키, 주소, 지갑
- 2) 합의 알고리즘
- 3) 스마트 컨트랙트 가상머신
- 4) 스마트 컨트랙트

소프트포크, 하드포크

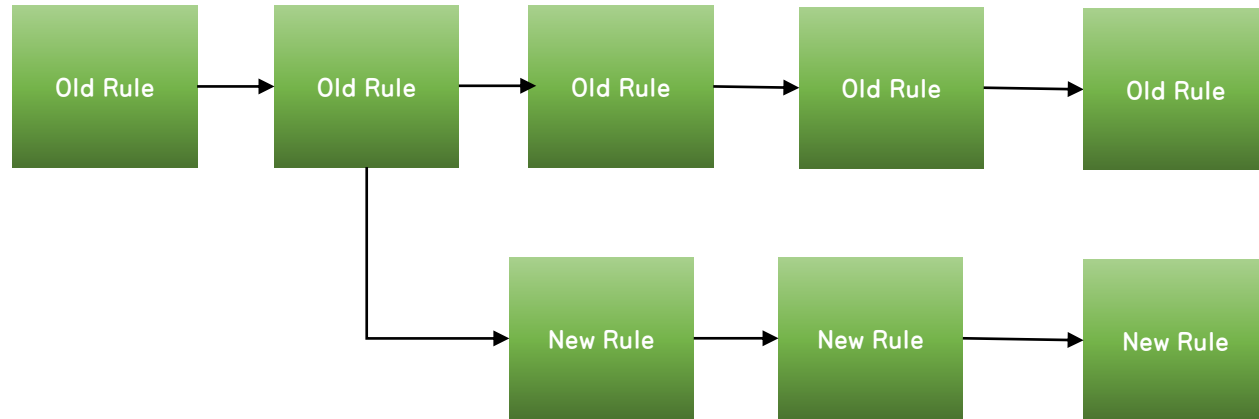
포크는 특정 커뮤니티, 개발자에 의해서 이루어 질 수 있음

소프트 포크



- 간단한 업데이트
- 기존 규칙에서 새로운 규칙을 추가 하는것
- 최종적으로 어떤 규칙이 선택될지는 네트워크 참여자 마음

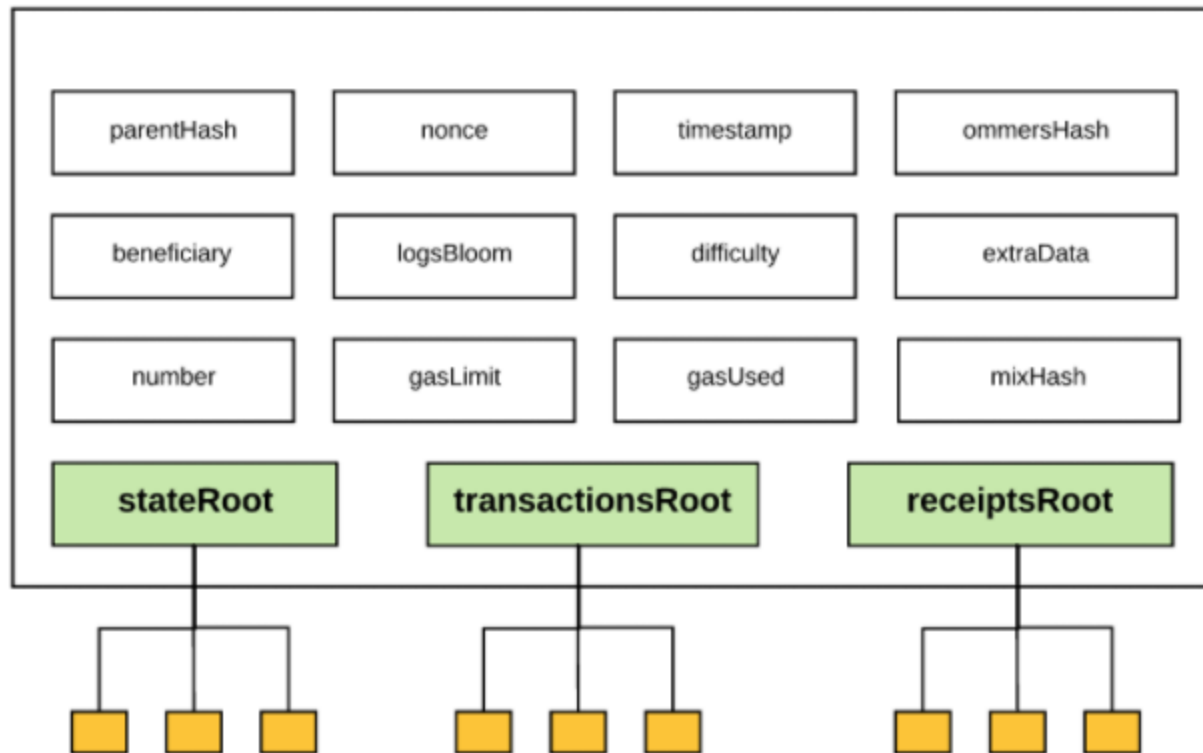
하드 포크



- 대규모 업데이트
- 기존 규칙을 지우고 완전히 새로운 규칙을 가진 블록을 생성
- 새로운 블록체인을 만드는 것
- 기존 블록과 연동될 수 없음

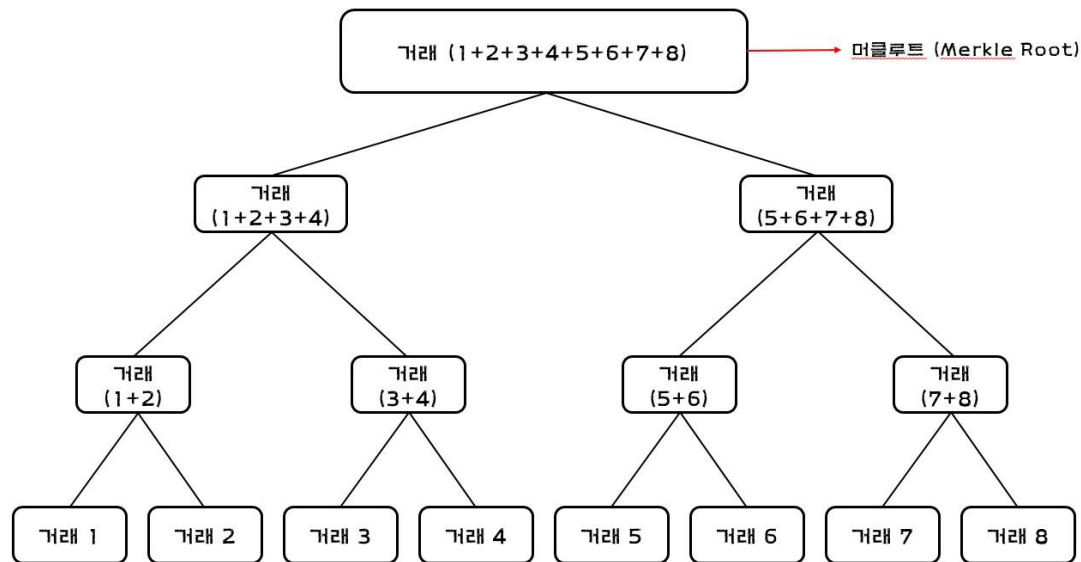
이더리움 블록	
timestamp	블록이 생성된 시간
blockNumber	블록 번호
baseFeePerGas	트랜잭션이 블록에 포함되기 위한 최소 수수료
difficulty	블록을 만드는데 필요한 난이도
mixHash	블록의 식별자
parentHash	이전 블록의 해시값
transactions	트랜잭션 리스트
stateRoot	머클 패트리샤 트리의 루트 시스템의 전체 상태, 계정 잔액, 계약 저장소, 스마트 컨트랙트 코드, 계정 논스 등
nonce	mixHash와 결합하여 블록이 작업증명을 통해 생성되었다는 것을 증명하기 위한 값

Block header

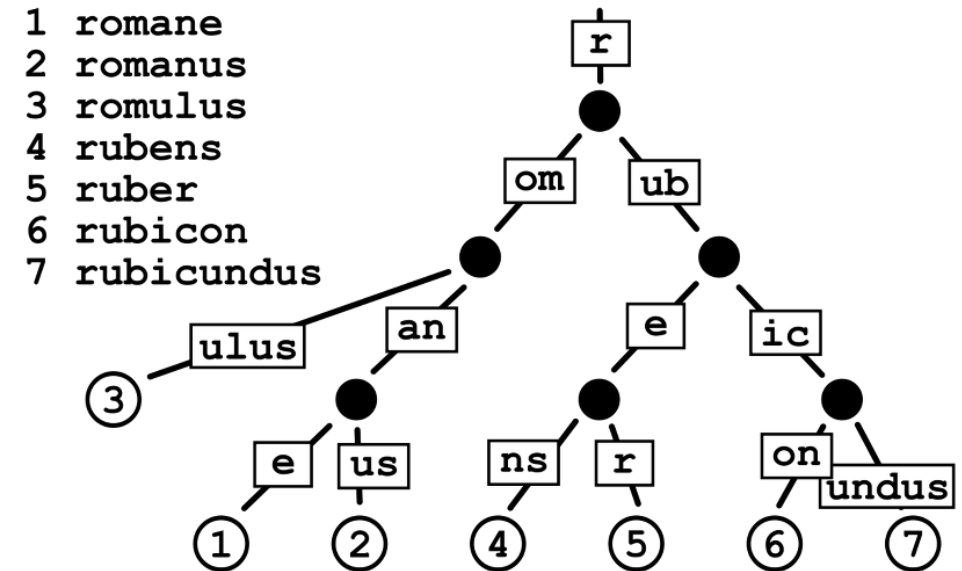


<https://hersheythings.xyz/entry/ethereumstructure>

머클 패트리샤 트리



<https://steemit.com/kr/@jsralph/merkle-trees>



<https://okky.kr/article/464145>

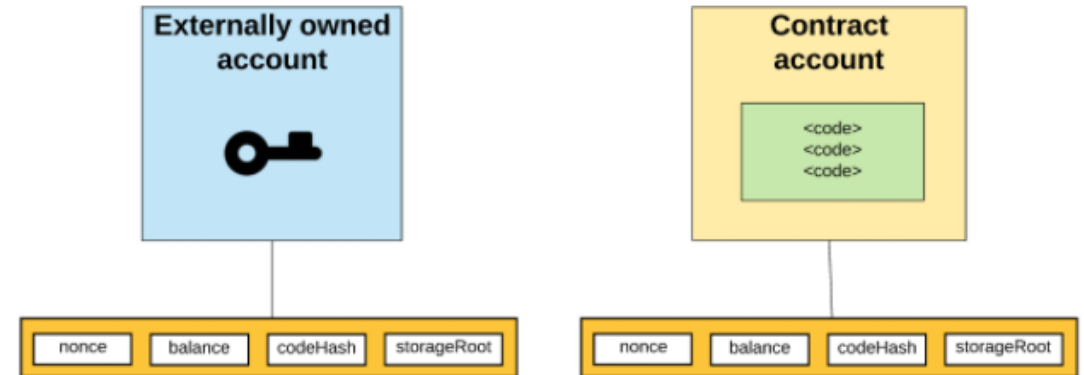
총 3개의 머클 패트리샤 트리를 활용

- State Tire: 블록을 통해서 변경된 어카운트 정보 저장
- Transaction Tire: 현재 블록의 트랜잭션 정보를 저장
- Receipts Tire: 현재 블록의 거래 영수증 정보를 저장

이더리움의 계정

계정 종류

- 외부 계정: 개인 키로 관리가 가능한 계정
- 컨트랙트: 네트워크에 배포된 스마트 컨트랙트, 코드에 의해 관리가 됨
- Nonce: 해당 어카운트 주소로부터 보내진 트랜잭션들의 수
- Balance: 해당 어카운트가 소유하고 있는 Balance의 양
- codeHash: 어카운트에 저장된 요소들을 저장한 머클 패트리샤 트리의 루트
- storageRoot: EVM에서 실행될 코드의 해시값, 외부계정에서는 비어있는 값



<https://hersheythings.xyz/entry/ethereumstructure>

이더리움의 트랜잭션

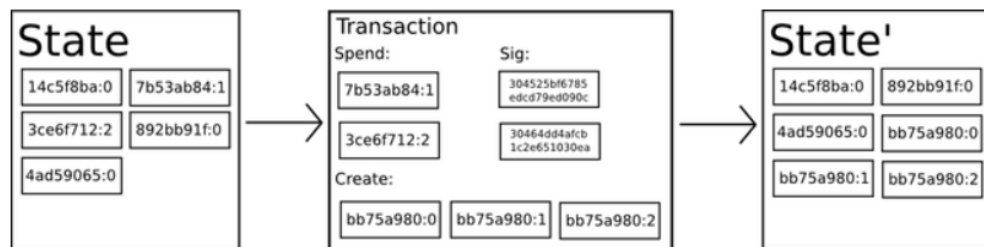
- 상태를 변환 시키는 요소
- nonce: 보내는 사람에 의해 보내진 트랜잭션의 수
- gasPrice: 트랜잭션이 실행될때 보내는 사람이 지불할 의향이 있는 가스의 가격
- gasLimit: 트랜잭션이 실행될때 보내는 사람이 지불할 의사가 있는 가스의 최대량
- to: 트랜잭션 받는사람
- value: 보내고자 하는 이더의 량
- v, r, s: 트랜잭션 보내는 사람을 식별하기 위한 서명을 생성할 때 사용되는 변수

이더리움의 가스

- 무한 루프등을 막기 위한 이더리움의 수수료
- 트랜잭션 실행을 위해서는 가스를 지불해야함

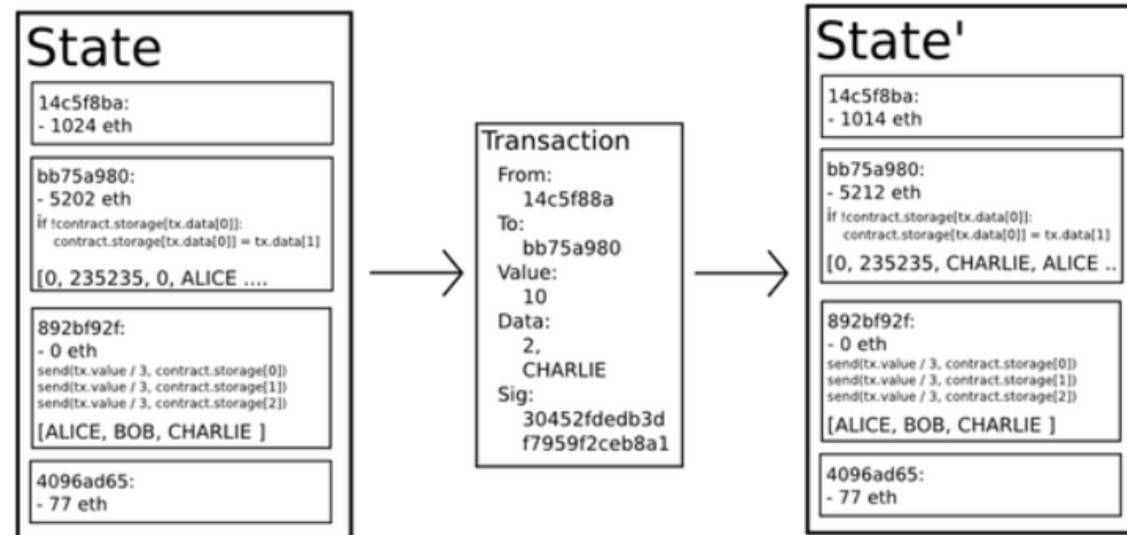
이더리움의 상태 데이터

비트코인의 상태 변환



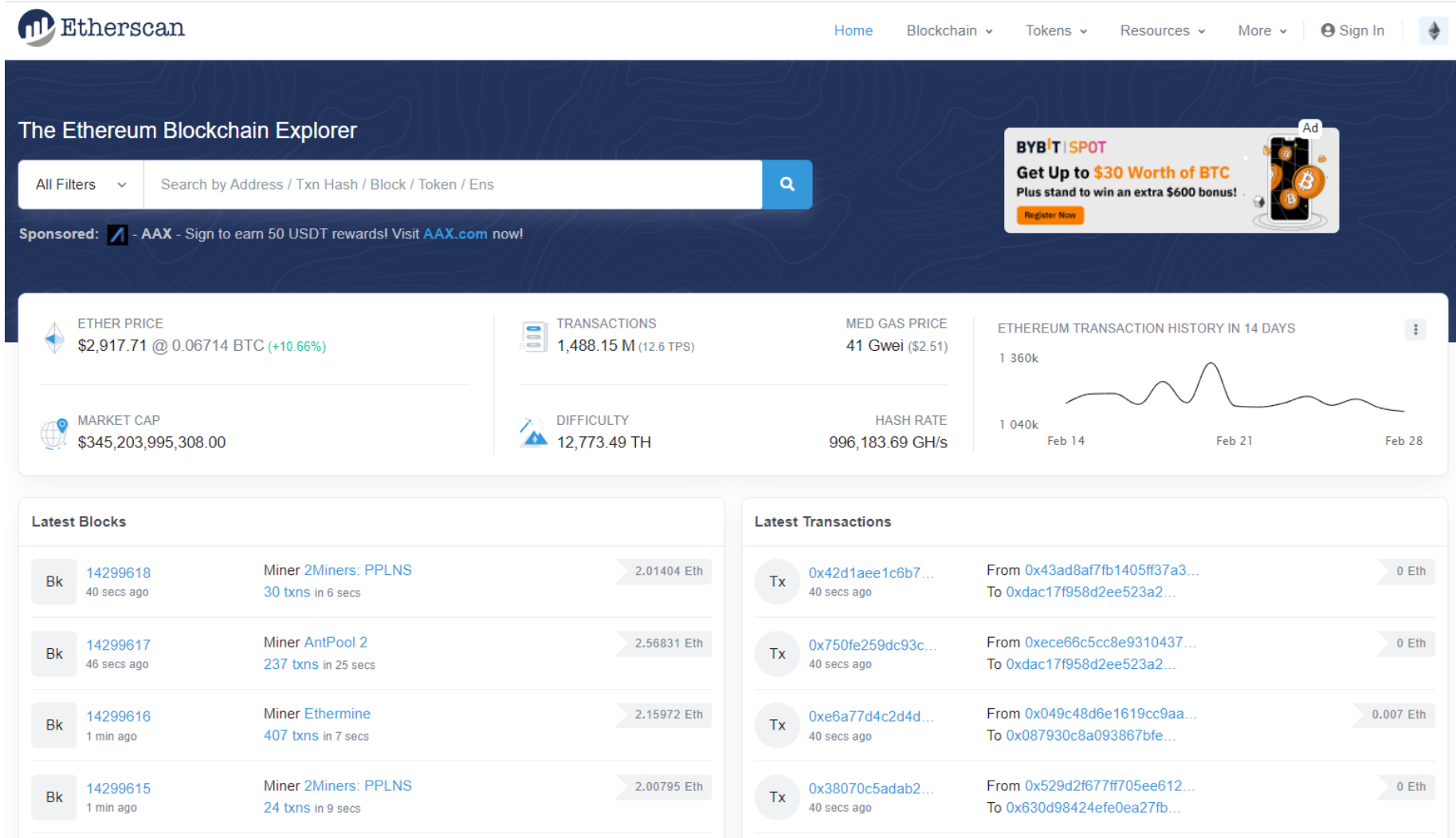
<https://steemit.com/kr-dev/@modolee/q6hpz>

- UTXO의 생성 삭제로 표현



<https://steemit.com/kr-dev/@modolee/q6hpz>

- 어카운트의 상태 변화로 표현



[Go Ethereum](#) [Install](#) [Downloads](#) [Documentation](#)

[Getting Started](#)
[Install and Build](#)
[Installing Geth](#)
[Backup & Restore](#)
[Cross-Compiling Geth](#)
[Using Geth](#)
[For dApp Developers](#)
[JSON RPC APIs](#)
[For Geth Developers](#)
[Clef](#)
[Vulnerabilities](#)

Installing Geth

You can install the Go implementation of Ethereum using a variety of ways. These include installing it via your favorite package manager; downloading a standalone pre-built bundle; running as a docker container; or building it yourself. This document details all of the possibilities to get you joining the Ethereum network using whatever means you prefer. A list of stable releases can be found [here](#).

- [Updating Geth](#)
- [Install from a package manager](#)
 - [Install on macOS via Homebrew](#)
 - [Install on Ubuntu via PPAs](#)
 - [Install on Windows](#)
 - [Install on FreeBSD via pkg](#)
 - [Install on FreeBSD via ports](#)
 - [Install on Arch Linux via pacman](#)
- [Download standalone bundle](#)
- [Run inside Docker container](#)
- [Build go-ethereum from source code](#)
 - [Most Linux systems and macOS](#)
 - [Windows](#)
 - [FreeBSD](#)
 - [Building without a Go workflow](#)

Updating Geth

Updating go-ethereum is as easy as it gets. You just need to download and install the newer version of geth, shutdown your node and restart with the new software. Geth will automatically use the data of your old node and sync the latest blocks that were mined since you shutdown the old software.

The screenshot displays the Remix IDE interface. On the left, the 'FILE EXPLORERS' sidebar shows a 'Workspaces' section with 'default_workspace' selected. Below it, a file tree lists 'contracts', 'scripts', 'tests', and 'README.txt'. The main workspace area features the 'Remix IDE' title, a 'Scam Alert' message, and a 'Featured Plugins' section with buttons for SOLIDITY, STARKNET, SOLHINT LINTER, LEARNETH, SOURCIFY, and MORE. At the bottom, there are 'File' and 'Resources' sections with links to 'New File', 'Open Files', 'Connect to Localhost', 'Documentation', 'Gitter channel', and 'Featuring website'. The bottom status bar includes a dropdown menu, a 'listen on network' checkbox, and a search bar labeled 'Search with transaction hash or address'.

FILE EXPLORERS

Workspaces

default_workspace

- contracts
- scripts
- tests
- README.txt

Remix IDE

⚠ Scam Alert: Beware of online videos promoting "liquidity front runner bots". [Learn more](#)

Featured Plugins

- SOLIDITY
- STARKNET
- SOLHINT LINTER
- LEARNETH
- SOURCIFY
- MORE

File

- New File
- Open Files
- Connect to Localhost

Resources

- [Documentation](#)
- [Gitter channel](#)
- [Featuring website](#)

0 ☐ listen on network

Search with transaction hash or address

The screenshot shows the Truffle Suite website. At the top, there's a navigation bar with the Truffle Suite logo, a search bar, and links for HOME, DOCUMENTATION, GUIDES, TUTORIAL, BOXES, BLOG, and COMMUNITY. The main content area features a large illustration of a chocolate cake on a black square base, with several smaller, outlined cake boxes floating around it, connected by dotted lines. Below the illustration, the text "Ganache" is written in a stylized font, followed by "ONE CLICK BLOCKCHAIN". To the right of this text, there's a paragraph: "Quickly fire up a personal Ethereum blockchain which you can use to run tests, execute commands, and inspect state while controlling how the chain operates." Below the paragraph, there's a button labeled "DOWNLOAD (WINDOWS)" with a Windows logo icon. To the left of the download button, there are two buttons: "GITHUB REPO" and "DOCS". Below these buttons, there's a GitHub Star button showing "4,019" stars. At the bottom right, there's a link that says "Need another OS download?".

TRUFFLE SUITE

HOME DOCUMENTATION GUIDES TUTORIAL BOXES BLOG COMMUNITY

Ganache

ONE CLICK BLOCKCHAIN

Quickly fire up a personal Ethereum blockchain which you can use to run tests, execute commands, and inspect state while controlling how the chain operates.


DOWNLOAD (WINDOWS)

GITHUB REPO DOCS

Star 4,019

Need another OS download?



 web3.js
v1.7.0

Search docs

USER DOCUMENTATION

Getting Started

Callbacks Promises Events

Glossary

API REFERENCE

Web3

web3.eth

web3.eth.subscribe

web3.eth.Contract

web3.eth.accounts

web3.eth.personal

web3.eth.ens

web3.eth.iban

web3.eth.abi

web3.*.net

web3.bzz

web3.shh

web3.utils

[Docs](#) » [web3.js - Ethereum JavaScript API](#)

[Edit on GitHub](#)

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.

The following documentation will guide you through [installing and running web3.js](#) as well as providing an API reference documentation with examples.

Contents:

[Keyword Index](#), [Search Page](#)

User Documentation

- [Getting Started](#)
 - [Adding web3.js](#)
- [Callbacks Promises Events](#)
- [Glossary](#)
 - [json interface](#)

API Reference

- [Web3](#)
 - [Web3 modules](#)

1. 이더리움의 탄생

- 1) 이더리움의 정의
- 2) 이더리움의 역사
- 3) 이더리움인 근황

2. 이더리움 동작과정

- 1) 소프트포크, 하드포크
- 2) 블록 구조
- 3) 구성 요소
- 4) 이더리움 도구

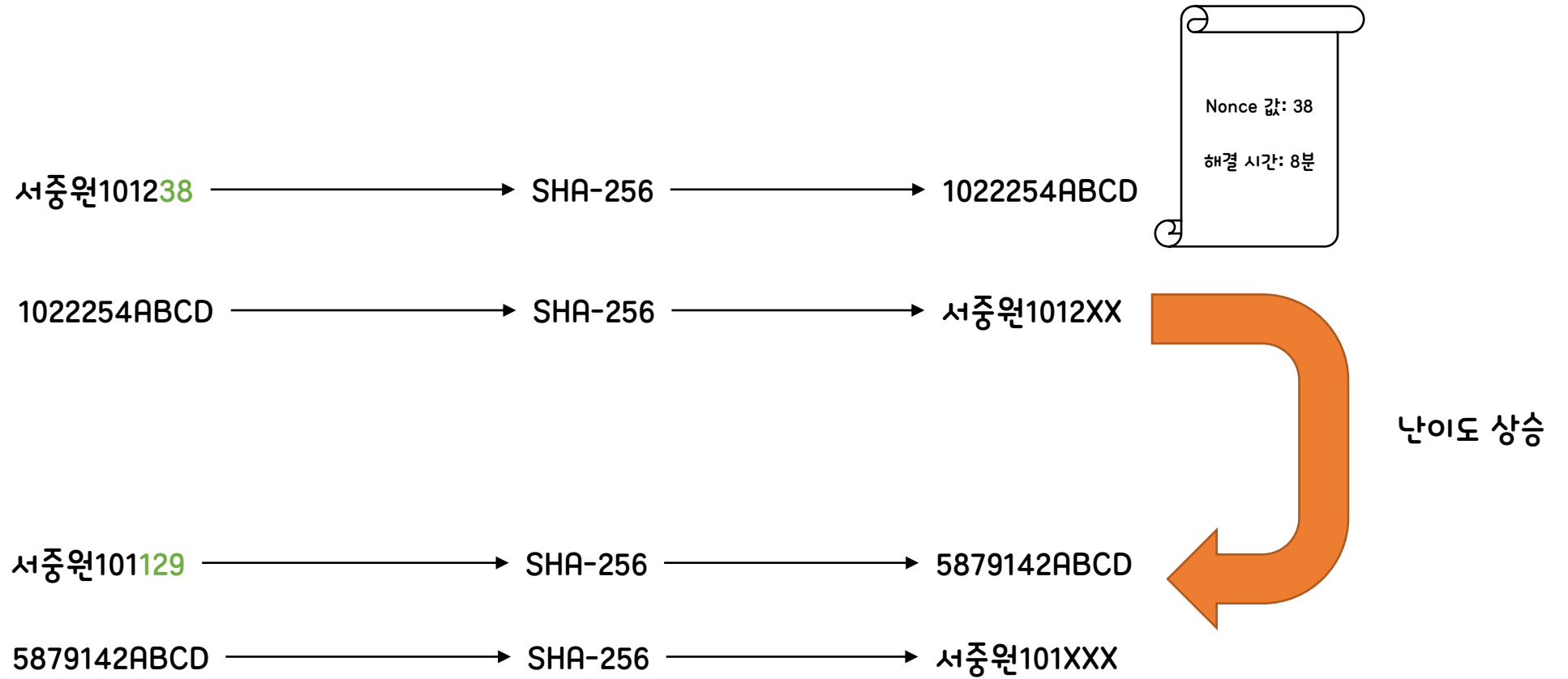
3. 이더리움의 기술적 요소

- 1) 합의 알고리즘
- 2) 스마트 컨트랙트 가상머신
- 3) 스마트 컨트랙트

작업증명 (PoW: Proof of Work, 이대시)

- 비트코인의 총량이 약 2100만 비트코인으로 고정되어있음
- 2100만 비트코인이 2140년이 될동안 작업증명 (채굴)을 통해 생성됨
- 해시알고리즘의 특징을 활용하여 사용된 알고리즘
- 10분안에 풀수 있도록 되어있음

합의 알고리즘





Author

Topic: Proof of stake instead of proof of work (Read 31823 times)

QuantumMechanic
Member
👤👤

Activity: 110
Merit: 14
👤

Proof of stake instead of proof of work #1

July 11, 2011, 04:12:45 AM
Merited by Vod (2), d5000 (1), drays (1)

I've got an idea, and I'm wondering if it's been discussed/ripped apart here yet:

I'm wondering if as bitcoins become more widely distributed, whether a transition from a proof of work based system to a proof of stake one might happen. What I mean by proof of stake is that instead of your "vote" on the accepted transaction history being weighted by the share of computing resources you bring to the network, it's weighted by the number of bitcoins you can prove you own, using your private keys.

For those that don't want to be actively verifying transactions, and so that not all private keys need to be facing the network, votes could be delegated to other addresses via some kind of nonstandard Bitcoin transaction. In this way, voting power would accumulate with trusted delegates instead of miners. New bitcoins and transaction fees could be randomly and periodically distributed to delegates, weighted by the number of votes they've accumulated, thereby incentivising diversity of the delegates and direct voters.

If the implementation could be done, it proved to maintain at least a similar level of privacy and trustworthiness, and it only minimally complicated the UX, I'm thinking that a proof of stake based fork could out-compete a proof of work one due to much lower transaction fees, since its network wouldn't need to support the cost of the miners' computing resources. (Note that the vote delegation scheme has bandwidth/storage overhead that would offset these savings by some amount which would hopefully be relatively small.)

Some other potential improvements this system could offer:

- Possibly quicker, more definite confirmation of transactions, depending on how it can be implemented.
- The "voting power" may be more trustworthy, since it would accumulate in a bottom-up fashion via a network of trust, instead of in the somewhat arbitrary way it accumulates now. (Note the potential problem of vote-buying here.)
- It would remove the physical point of failure of bitcoin mining equipment, which can be confiscated or made illegal to run.
- It could be used to provide stakeholders a means of making their voices heard (via the delegated voting system it establishes) when it comes to proposals for software updates and protocol changes.

Anyway, I just wanted to throw the idea out here to see if there are any obvious reasons why it couldn't be implemented, and to hopefully spark a discussion amongst those better qualified than me.

Cheers.

PoS 로 가기위한 Ethereum 두가지 프로젝트

1.Casper Friendly Finality Gadget (FFG)

- 비탈린 부테린 (Vitalik Buterin)이 주도
- PoW와 PoS를 합친 개념

2.Correct By Construction Consensus Protocols (CBC)

- 블라드 잠퍼 (Vlad Zamfir)가 주도
- PoS를 포함한 완전히 새로운 Consensus Algorithm을 설계하기 위한 프로젝트

FFG

- 분기를 막기위한 블록 Finalizing에 초점을 맞춤

Casper the Friendly Finality Gadget is an overlay atop a *proposal mechanism*—a mechanism which proposes blocks¹. Casper is responsible for finalizing these blocks, essentially selecting a unique chain which represents the canonical transactions of the ledger. Casper provides safety, but liveness depends on the chosen proposal mechanism. That is, if attackers wholly control the proposal mechanism, Casper protects against finalizing two conflicting checkpoints, but the attackers could prevent Casper from finalizing any future checkpoints.

Rather than deal with the full block tree, for efficiency purposes² Casper only considers the subtree of *checkpoints* forming the *checkpoint tree* (Figure 1a). The genesis block is a checkpoint, and every block whose height in the block tree (or block number) is an exact multiple of 100 is also a checkpoint. The “checkpoint height” of a block with block height $100 * k$ is simply k ; equivalently, the height $h(c)$ of a checkpoint c is the number of elements in the checkpoint chain stretching from c all the way back to root along the parent links (Figure 1b).³

CBC

- 현재의 합의는 분산네트워크 환경에서 사용되기 위한 합의이기 때문에 블록체인만을 위한 합의가 필요하다

$$\begin{aligned}
 F(\bigcup_{i=1}^n \sigma_i) \leq t &\implies \bigcap_{i=1}^n Futures_t(\sigma_i) \neq \emptyset \\
 &\iff \exists \sigma \in \Sigma_t, \sigma \in \bigcap_{i=1}^n Futures_t(\sigma_i) \\
 &\iff \exists \sigma \in \Sigma_t, \bigwedge_{i=1}^n \sigma \in Futures_t(\sigma_i) \\
 &\iff \exists \sigma \in \Sigma_t, \bigwedge_{i=1}^n \sigma \in Futures_t(\sigma_i) \\
 \wedge \bigwedge_{j=1}^n \sigma \in Futures_t(\sigma_j) &\implies (\forall p \in P_\Sigma, Decided_{\Sigma,t}(p, \sigma_j) \implies Decided_{\Sigma,t}(p, \sigma)) \\
 &\implies \exists \sigma \in \Sigma_t, \bigwedge_{i=1}^n \forall p \in P_\Sigma, Decided_{\Sigma,t}(p, \sigma_i) \implies Decided_{\Sigma,t}(p, \sigma) \\
 &\implies \exists \sigma \in \Sigma_t, \bigwedge_{i=1}^n \forall p \in Decisions_{\Sigma,t}(\sigma_i), Decided_{\Sigma,t}(p, \sigma_i) \implies Decided_{\Sigma,t}(p, \sigma) \\
 &\iff \exists \sigma \in \Sigma_t, \bigwedge_{i=1}^n \forall p \in Decisions_{\Sigma,t}(\sigma_i), Decided_{\Sigma,t}(p, \sigma) \\
 &\iff \exists \sigma \in \Sigma_t, \forall p \in \bigcup_{i=1}^n Decisions_{\Sigma,t}(\sigma_i), Decided_{\Sigma,t}(p, \sigma) \\
 &\implies \exists \sigma \in \Sigma_t, \forall p \in \bigcup_{i=1}^n Decisions_{\Sigma,t}(\sigma_i), p(\sigma) \\
 &\iff Consistent_\Sigma(\bigcup_{i=1}^n Decisions_{\Sigma,t}(\sigma_i))
 \end{aligned}$$

이더리움 가상머신

- 스마트 컨트랙트 배포 및 실행 처리를 담당
- 튜링 완전한 기계가 가능하도록 함
- 메모리 내의 모든 값을 스택에 저장하는 스택 기반 아키텍처
- 코드를 돌릴 수 있도록 만들어 줌

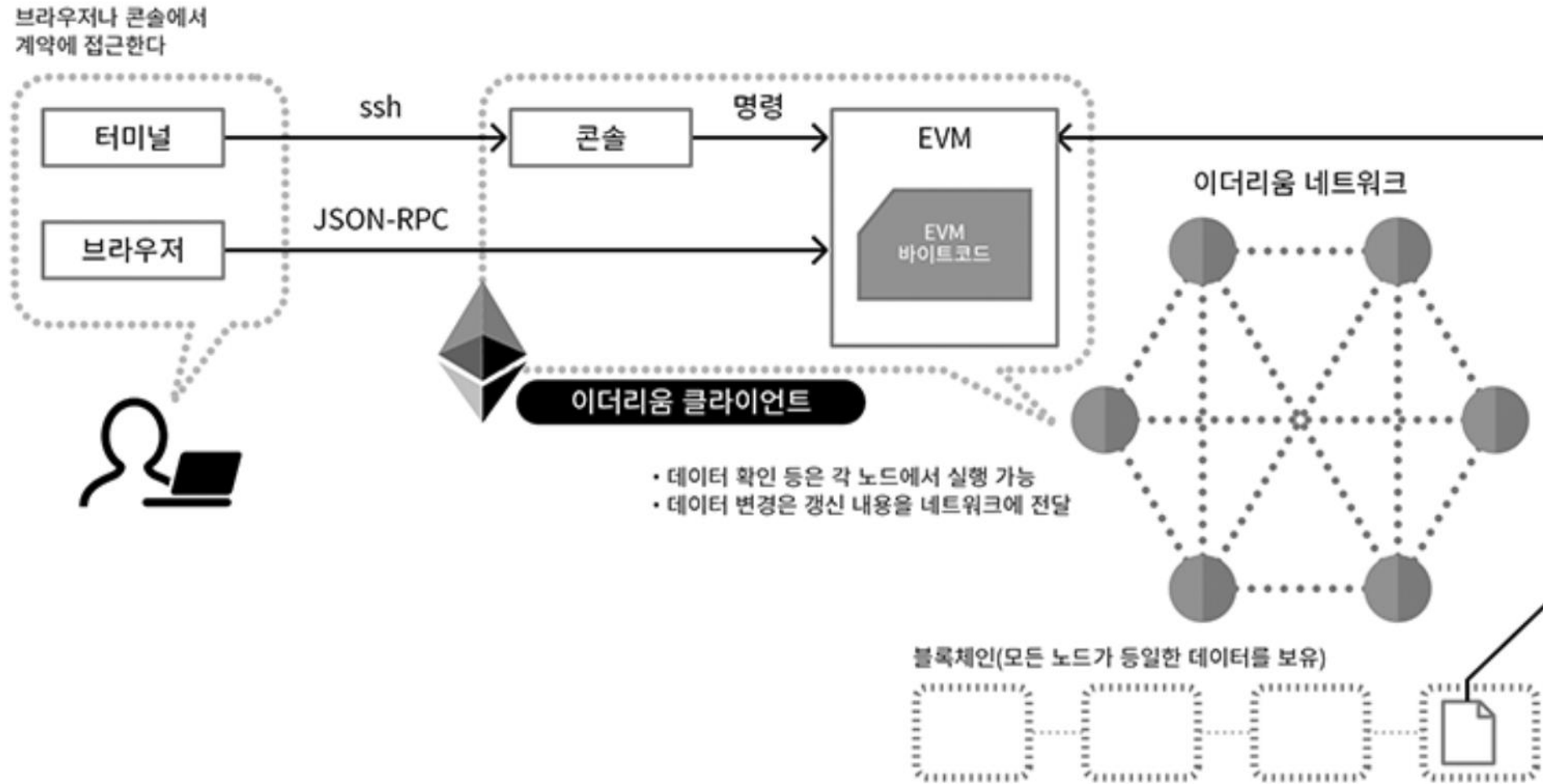


Alan Mathison Turing

튜링 완전성

무한 순환을 포함에서 상상 가능한
모든 계산을 할 수 있는 것을 의미

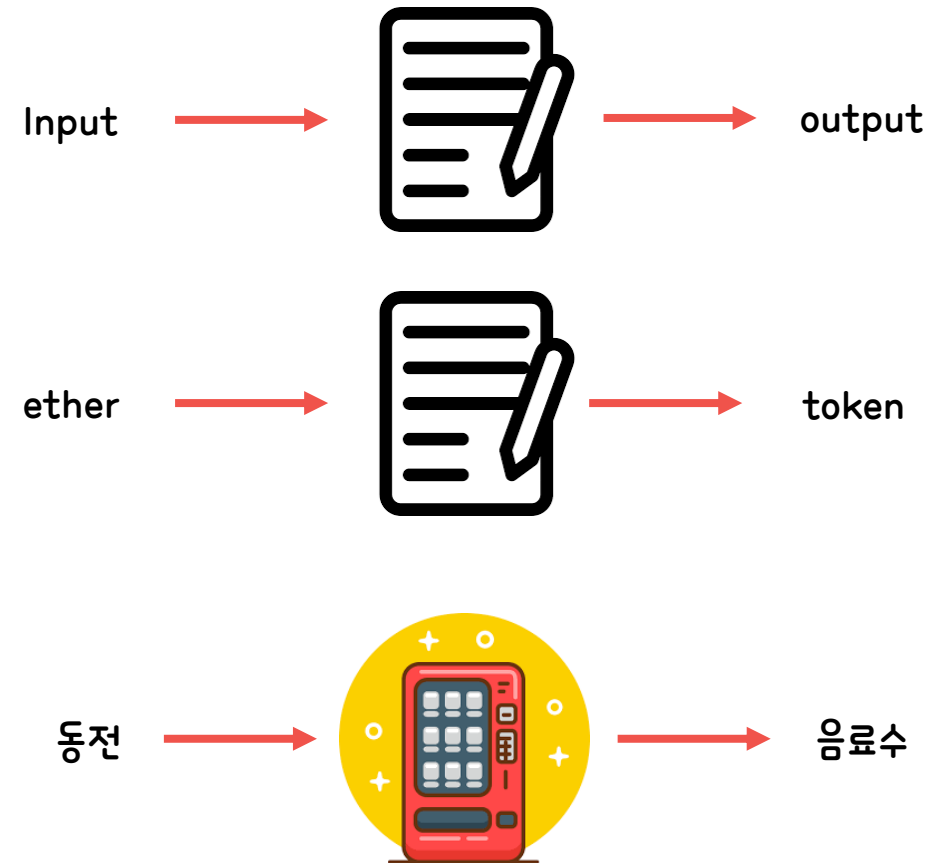
스마트 컨트랙트 가상머신



<https://opentutorials.org/course/2869/19273>



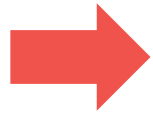
- 1994년 Nick Szabo가 제시한 개념
- 디지털화된 계약서가 해킹될 경우 문제가 발생
- 이더리움을 통해 스마트컨트랙트가 사용됨
- 코드의 조각
- 블록체인과 상호 작용하는 인터페이스



스마트 컨트랙트



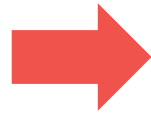
스마트 컨트랙트
생성



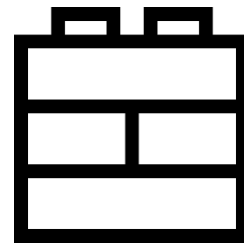
컴파일



바이트 코드 변환



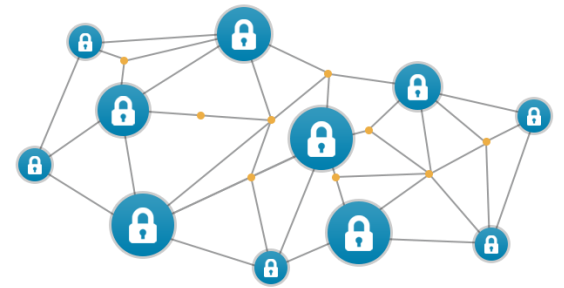
트랜잭션



블록에 저장



배포



블록체인에 저장

스마트 컨트랙트 언어의 종류

Mutan

- ?

LLL (Lisp-Like Language)

- 함수형 프로그래밍 언어, 이더리움의 첫 언어

Serpent

- 파이썬과 유사한 구문을 사용하는 프로그램 언어
- 개발 중단

Solidity

- 자바스크립트, C++, 자바와 유사한 구문을 사용하는 언어
- 가장 널리 사용되고 자주 사용되는 언어

Viper

- 서펀트와, 파이썬과 비슷한 구문을 사용함

Bamboo

- 명시적 상태 전이와 루프가 없는 새로 개발된 언어

Q & A

