Blockchain?!

블록체인은 어떻게 동작하는가? (feat. 비트코인, 이더리움)

Cypherpunk

Cipher + Cyberpunk

https://www.activism.net/cypherpunk/manifesto.html

Privacy

모든 사람들이 알게 되는 것을 원하지 않는 것

↓

자신을 선택적으로 세상에 드러내는 힘

Secret

어느 누구도 알지 못하게 하는 것

https://www.activism.net/cypherpunk/manifesto.html

Privacy → Open Society

필수적

보호 ↓

거래 당사자는
거래에 직접적으로 필요한 정보만 알아야 함

↓

(자신이 원하는 만큼 자신의 정보 공개)
익명 거래 시스템 필요

누구나 자유롭게 정보를 열람하고
자신의 의견을 말하는 개방적인 사회

열린 사회(open society) : 사회 계급이나 계층에 상관 없이 누구나 자유롭게 정보를 열람하고 자신의 정치·경제·사회적 의견을 말할 수 있는 개방적인 사회

익명 거래 시스템

(anonymous transaction systems)

암호기술(cryptography)

+

우리 스스로
코드 작성

개발 →

메일링 리스트
디지털 서명
전자화폐
⋮

무료 배포
및 사용 →

무정지 분산 시스템

사이퍼펑크 선언문 (A Cypherpunk's Manifesto) 요약

https://www.activism.net/cypherpunk/manifesto.html

프라이버시(Privacy)는 전자 시대에서 열린 사회를 위해 필수적이다. 프라이버시는 비밀과 다르다. 프라이버시는 세상의 모든 사람들이 알게 되는 것을 원하지 않는 것이고, 비밀은 어느 누구도 알지 못하게 하는 것이다. 프라이버시는 자신에 대해 선택적으로 세상에 드러낼 수 있는 힘이다.

프라이버시를 보호하려면, 거래 당사자는 오직 해당 거래에 직접적으로 필요한 정보만 알아야 한다. 만약 가게에서 잡지를 구매하고 점원에게 현금을 건네준다면, 내가 누구인지에 대해 알려줄 필요가 없다. 만약 그걸 알려줘야 한다면, 프라이버시가 보호되지 않는 것이다.

열린 사회에서 프라이버시를 보호하려면 익명의 거래 시스템이 필요하다. 지금까지는 현금이 그런 역할을 했다. 익명의 거래 시스템은 비밀 거래 시스템이 아니다. 익명의 거래 시스템에서 개인들은 자신이 원하는 만큼 자신에 대한 정보를 공개할 수 있다.

열린 사회에서 프라이버시를 보호하려면 암호기술이 필요하다. 내가 한 말은 내가 공개하고 싶은 사람들에게만 공개되어야 한다. 만약 내가 한 말이 전 세계 누구에게나 알려질 수 있다면, 프라이버시가 보호되지 않는 것이다.

정부나 기업 또는 다른 거대 조직들이 우리의 프라이버시를 지켜줄 것이라고 기대할 수 없다. 프라이버시 보호를 원한다면, 우리 스스로 지켜야 한다. 우리는 익명의 거래가 이루어질 수 있는 시스템을 만들기 위해 힘을 합쳐야 한다.

사이퍼펑크는 익명의 시스템을 만들기 위해 노력한다. 우리는 암호기술과 익명의 메일링 리스트 시스템, 디지털 서명, 그리고 전자화폐를 사용하여 우리의 프라이버시를 보호한다.

사이퍼펑크는 코드를 개발한다. 프라이버시를 보호하기 위한 소프트웨어를 개발해야 하는데, 우리 모두가 하지 않으면 프라이버시를 지킬 수 없기 때문에, 우리가 직접 개발할 것이다. 우리가 짠 코드는 동료 사이퍼펑크 개발자들이 이용할 수 있도록 전 세계에 무료로 배포될 것이다. 이 소프트웨어는 결코 파괴되지 않을 것이고, 광범위하게 분산된 시스템은 절대 정지되지 않을 것이다.

사이퍼펑크는 암호기술에 대한 규제를 반대한다. 암호기술을 규제하는 법률은 국가의 경계선을 벗어날 수 없다. 암호기술은 글로벌하게 퍼질 것이며, 그와 함께 익명의 거래 시스템도 전 세계로 확산될 것이다.

사이퍼펑크는 프라이버시를 안전하게 지켜주는 네트워크를 만들기 위해 적극 참여한다. 우리 함께 앞으로 힘차게 전진하자!

에릭 휴즈(Eric Hughes), hughes@soda.berkeley.edu, 1993년 3월 9일

암호화폐와 스마트 컨트랙트 역사

1983년 · 1990년 · 1992년 · 1994년 · 1997년 · 1998년 · 2004년 · 2009년 · 2015년

ECASH

- Privacy 보호
- 최초 암호화폐

데이비드 차움
(David Chaum)

DigiCash

- ECASH 실현
- 중앙화

데이비드 차움
(David Chaum)

1989년 설립
1998년 파산

Cypherpunks

에릭 휴즈
(Eric Hughes)

티모시 메이
(Timothy May)

존 길모어
(John Gilmore)

Smart Contract

닉 재보
(Nick Szabo)

Hashcash

- 작업 증명(PoW)

아담 백
(Adam Back)
[2002년: 논문]

B-Money

- 분산 합의
- 계산 퍼즐
- 전자화폐

웨이 다이
(Wei Dai)

Bit gold

- 타임스탬프 서버
- 분산 전자 화폐
- 스마트 계약

닉 재보
(Nick Szabo)

RPOW

- 재사용 PoW

할 피니
(Hall finney)

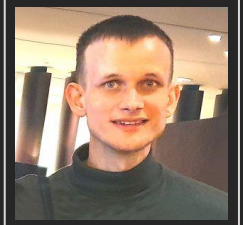
Bitcoin

- 익명성 보장
  암호화폐 시스템

사토시 나카모토
(Satoshi Nakamoto)

Ethereum

- 스마트 컨트랙트
  플랫폼



비탈릭 부테린
(Vitalik Buterin)

2008-10-31 Cypherpunk mailing list

Bitcoin P2P e-cash paper

Satoshi Nakamoto satoshi at vistomail.com
Fri Oct 31 14:10:00 EDT 2008

- Previous message: Fw: SHA-3 lounge
- **Messages sorted by:** [date] [thread] [subject] [author]

```
I've been working on a new electronic cash system that's fully
peer-to-peer, with no trusted third party.

The paper is available at:
http://www.bitcoin.org/bitcoin.pdf

The main properties:
 Double-spending is prevented with a peer-to-peer network.
 No mint or other trusted parties.
 Participants can be anonymous.
 New coins are made from Hashcash style proof-of-work.
 The proof-of-work for new coin generation also powers the
    network to prevent double-spending.

Bitcoin: A Peer-to-Peer Electronic Cash System

Abstract.  A purely peer-to-peer version of electronic cash would
allow online payments to be sent directly from one party to another
without the burdens of going through a financial institution.
Digital signatures provide part of the solution, but the main
benefits are lost if a trusted party is still required to prevent
double-spending.  We propose a solution to the double-spending
problem using a peer-to-peer network.  The network timestamps
transactions by hashing them into an ongoing chain of hash-based
proof-of-work, forming a record that cannot be changed without
redoing the proof-of-work.  The longest chain not only serves as
proof of the sequence of events witnessed, but proof that it came
from the largest pool of CPU power.  As long as honest nodes control
the most CPU power on the network, they can generate the longest
chain and outpace any attackers.  The network itself requires
minimal structure.  Messages are broadcasted on a best effort basis,
and nodes can leave and rejoin the network at will, accepting the
longest proof-of-work chain as proof of what happened while they
were gone.

Full paper at:
http://www.bitcoin.org/bitcoin.pdf

Satoshi Nakamoto

--------------------------------------------------------------------
The Cryptography Mailing List
Unsubscribe by sending "unsubscribe cryptography" to majordomo at metzdowd.com
```

- Previous message: Fw: SHA-3 lounge
- **Messages sorted by:** [date] [thread] [subject] [author]

More information about the cryptography mailing list

https://satoshi.nakamotoinstitute.org/emails/cryptography/

Bitcoin Paper

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

1

https://bitcoin.org/bitcoin.pdf , 한글

1. Introduction

인터넷 기반 상거래는 전자 결제를 처리할 신뢰받는 제 3 자 역할을 거의 전적으로 금융기관에 의존해 왔다. 이 시스템은 대다수 거래에 충분히 잘 동작하지만, 여전히 신뢰 기반 모델의 태생적 약점을 극복하지 못한다.

Commerce on the Internet has come to *rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments*. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model.

…

필요한 것은 신뢰 대신 암호학적 증명(cryptographic proof)에 기반해), 거래 의사가 있는 두 당사자가 신뢰받는 제 3 자를 필요로 하지 않고 서로 직접 거래하게 해주는 전자 화폐 시스템이다.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party.

…

12. Conclusion

우리는 신뢰에 의존하지 않는 전자거래용 시스템을 제안했다. …

We have proposed a system for electronic transactions without relying on trust. …

Bitcoin Genesis Block, 2009-01-03 18:15 UTC

Current
0

Next
1

Hash
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

General info

Mined on	Jan 3, 2009 6:15 PM UTC	Miner	Unknown
Transaction count	1	Fee per kB	0 BTC · 0 USD
Witness tx count	0	Fee per kWU	0 USD
Input count	1	Output count	1
Input total	0 BTC · 0 USD	Output total	50.00 BTC · 1 USD
Fee total	0 BTC · 0 USD	Coindays destroyed	0
Generation	50.00 BTC · 1 USD	Reward	50.00 BTC · 1 USD
Size	285	Weight	1,140
Stripped size	285	Median time	Jan 3, 2009 6:15 PM UTC
Version	1_{10} 1_{16}	Version [bits]	$00000000000000000000000000001_2$
Merkle root	4a5e1e ••• eda33b	Difficulty	1
Nonce	2,083,236,893	Bits	486,604,799
Chainwork	000000 ••• 010001		
Coinbase data	··┌EThe Times 03/Jan/2009 Chancellor on brink of second bailout for banks		

https://blockchair.com/bitcoin/block/0

The Times 03/Jan/2009 Chancellor on brink of second bailout for banks
더 타임스 2009-01-03 은행에 대한 두번째 구제금융에 임박한 재무장관

THE TIMES

Max 5C, min -5C | Saturday January 3 2009 timesonline.co.uk | No 69523 | £1.50

Eat Out from £5

More than 900 great restaurants, including four **Gordon Ramsay** favourites from £15

Start collecting tokens today Pullout Inside

Israel prepares to send tanks and troops into Gaza

Israel allowed foreigners to flee the Gaza Strip as it prepared for a ground offensive. At least 430 Palestinians were killed in a week of airstrikes News, page 3

Chancellor on brink of second bailout for banks

Billions may be needed as lending squeeze tightens

Francis Elliott Deputy Political Editor
Gary Duncan Economics Editor

Alistair Darling has been forced to consider a second bailout for banks as the lending drought worsens.

The Chancellor will decide within weeks whether to pump billions more into the economy as evidence mounts that the £37billion part-nationalisation last year has failed to keep credit flowing. Options include cash injections, offering banks cheaper state guarantees to raise money privately or buying up "toxic assets", The Times has learnt.

The Bank of England revealed yesterday that, despite intense pressure, the banks curbed lending in the final quarter of last year and plan even tighter restrictions in the coming months. Its findings will alarm the Treasury.

The Bank is expected to take yet more aggressive action this week by cutting the base rate from its current level of 2 per cent. Doing so would reduce the cost of borrowing but have little effect on the availability of loans.

Whitehall sources said that ministers planned to "keep the banks on the boil" but accepted that they need more help to restore lending levels. Formally, the Treasury plans to focus on state-backed guarantees to encourage private finance, but a number of interventions are on the table, including further injections of taxpayers' cash.

Under one option, a "bad bank" would be created to dispose of bad debts. The Treasury would take bad loans off the hands of troubled banks, perhaps swapping them for government bonds. The toxic assets, blamed for poisoning the financial system, would be parked in a state vehicle or "bad bank" that would manage them and attempt to dispose of them while "detoxifying" the mainstream banking system.

The idea would mirror the initial proposal by Henry Paulson, the US Treasury Secretary, to underpin the American banking system by buying

Continued on page 6, col 1
Leading article, page 2

99p

Pub chain cuts the price of a pint from £1.69 to 1989 levels
Business, page 47

Michael Sheen Frost, Nixon and me
Magazine

Working mums So that's how she does it
Body&Soul

Detox in style The best spas on the planet
Travel

Salmon Rushdie I Won't Marry Again
Pages 22, 23

Giant Killing? Guide to the FA Cup Third Round
Sport

2009-01-08, Bitcoin v0.1 released

Bitcoin v0.1 released

Satoshi Nakamoto satoshi at vistomail.com
Thu Jan 8 14:27:40 EST 2009

- Previous message: [tmoore at seas.harvard.edu: [fc-announce] F
- Next message: MD5 considered harmful today, SHA-1 consider
- **Messages sorted by:** [date] [thread] [subject] [author]

Announcing the first release of Bitcoin, a new electronic cash system that uses a peer-to-peer network to prevent double-spending. It's completely decentralized with no server or central authority.

See bitcoin.org for screenshots.

Download link:
http://downloads.sourceforge.net/bitcoin/bitcoin-0.1.0.rar

Windows only for now. Open source C++ code is included.

- Unpack the files into a directory
- Run BITCOIN.EXE
- It automatically connects to other nodes

If you can keep a node running that accepts incoming connections, you'll really be helping the network a lot. Port 8333 on your firewall needs to be open to receive incoming connections.

The software is still alpha and experimental. There's no guarantee the system's state won't have to be restarted at some point if it becomes necessary, although I've done everything I can to build in extensibility and versioning.

You can get coins by getting someone to send you some, or turn on Options->Generate Coins to run a node and generate blocks. I made the proof-of-work difficulty ridiculously easy to start with, so for a little while in the beginning a typical PC will be able to generate coins in just a few hours. It'll get a lot harder when competition makes the automatic adjustment drive up the difficulty. Generated coins must wait 120 blocks to mature before they can be spent.

There are two ways to send money. If the recipient is online, you can enter their IP address and it will connect, get a new public key and send the transaction with comments. If the recipient is not online, it is possible to send to their Bitcoin address, which is a hash of their public key that they give you. They'll receive the transaction the next time they connect and get the block it's in. This method has the disadvantage that no comment information is sent, and a bit of privacy may be lost if the address is used multiple times, but it is a useful alternative if both users can't be online at the same time or the recipient can't receive incoming connections.

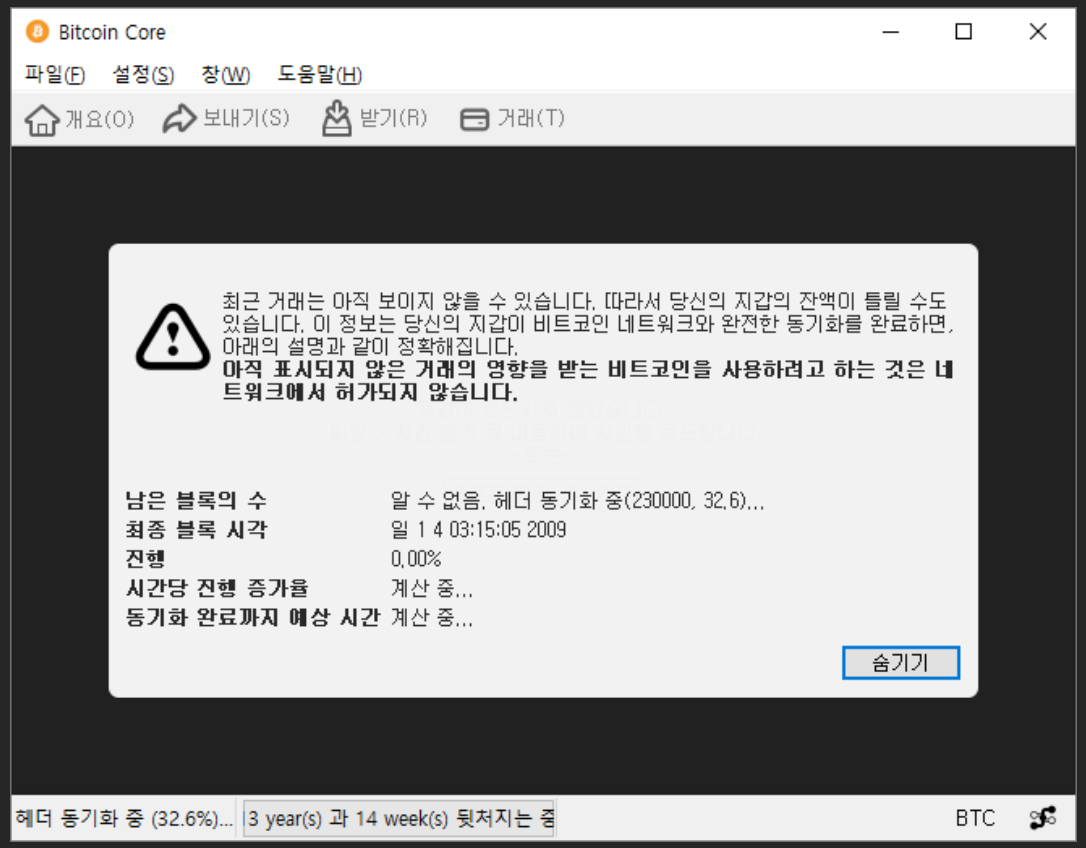
Total circulation will be 21,000,000 coins. It'll be distributed to network nodes when they make blocks, with the amount cut in half every 4 years.

first 4 years: 10,500,000 coins
next 4 years: 5,250,000 coins
next 4 years: 2,625,000 coins
next 4 years: 1,312,500 coins
etc...

When that runs out, the system can support transaction fees if needed. It's based on open market competition, and there will probably always be nodes willing to process transactions for free.

Satoshi Nakamoto

Bitcoin Core 22.0



2013-11-27, 비탈릭 부테린 : 이더리움 백서, 개발 제안

Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.
By Vitalik Buterin (2014).

When Satoshi Nakamoto first set the Bitcoin blockchain into motion in January 2009, he was simultaneously introducing two radical and untested concepts. The first is the "bitcoin", a decentralized peer-to-peer online currency that maintains a value without any backing, intrinsic value or central issuer. So far, the "bitcoin" as a currency unit has taken up the bulk of the public attention, both in terms of the political aspects of a currency without a central bank and its extreme upward and downward volatility in price. However, there is also another, equally important, part to Satoshi's grand experiment: the concept of a proof of work-based blockchain to allow for public agreement on the order of transactions. Bitcoin as an application can be described as a first-to-file system: if one entity has 50 BTC, and simultaneously sends the same 50 BTC to A and to B, only the transaction that gets confirmed first will process. There is no intrinsic way of determining from two transactions which came earlier, and for decades this stymied the development of decentralized digital currency. Satoshi's blockchain was the first credible decentralized solution. And now, attention is rapidly starting to shift toward this second part of Bitcoin's technology, and how the blockchain concept can be used for more than just money.

Commonly cited applications include using on-blockchain digital assets to represent custom currencies and financial instruments ("colored coins"), the ownership of an underlying physical device ("smart property"), non-fungible assets such as domain names ("Namecoin") as well as more advanced applications such as decentralized exchange, financial derivatives, peer-to-peer gambling and on-blockchain identity and reputation systems. Another important area of inquiry is "smart contracts" - systems which automatically move digital assets according to arbitrary pre-specified rules. For example, one might have a treasury contract of the form "A can withdraw up to X currency units per day, B can withdraw up to Y per day, A and B together can withdraw anything, and A can shut off B's ability to withdraw". The logical extension of this is decentralized autonomous organizations (DAOs) - long-term smart contracts that contain the assets and encode the bylaws of an entire organization. What Ethereum intends to provide is a blockchain with a built-in fully fledged Turing-complete programming language that can be used to create "contracts" that can be used to encode arbitrary state transition functions, allowing users to create any of the systems described above, as well as many others that we have not yet imagined, simply by writing up the logic in a few lines of code.

Page 1
ethereum.org

https://ethereum.org/en/whitepaper/ , 한글

2015-07-30 15:25:13 UTC, Genesis Block 생성

Block #0

Overview | Comments

Block Height:	0
Timestamp:	2453 days 21 hrs ago (Jul-30-2015 03:26:13 PM +UTC)
Transactions:	8893 transactions and 0 contract internal transaction in this block
Mined by:	0x0000000000000000000000000000000000000000(Null Address: 0x000…000) in 15 secs
Block Reward:	5 Ether
Uncles Reward:	0
Difficulty:	17,179,869,184
Total Difficulty:	17,179,869,184
Size:	540 bytes
Gas Used:	0 (0.00%)
Gas Limit:	5,000
Extra Data:	◀◆◆◆N4{N◆◆|└◆p◆◆3◆◆◆i◆◆z8◆◆♂←◆◆ (Hex:0x11bbe8db4e347b4e8c937c1c8370e4b5ed33adb3db69cbdb7a38e1e50b1b82fa)
Ether Price:	N/A
Hash:	0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3
Parent Hash:	0x0000000000000000000000000000000000000000000000000000000000000000
Sha3Uncles:	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347
StateRoot:	0xd7f8974fb5ac78d9ac099b9ad5018bedc2ce0a72dad1827a1709da30580f0544
Nonce:	0x0000000000000042

https://etherscan.io/block/0

차세대 스마트 컨트랙트와 탈중앙화된 어플리케이션 플랫폼

A Next-Generation Smart Contract and Decentralized Application Platform

사토시 나카모토에 의해 2009 년 개발된 비트코인은 종종 화폐와 통화분야에서 매우 근본적인 혁신으로 묘사되어 왔는데, 이것은 비트코인이 어떤 담보나 내재적인 가치를 가지지 않으며, 중앙화된 발행기관이나 통제기관도 없는 디지털 자산의 첫 번째 사례였기 때문이다. 하지만 비트코인 실험의 더욱 중요한 측면은 비트코인을 떠받치고 있는 분산 합의 수단으로서의 블록체인 기술이며, 이에 대한 관심이 급격하게 늘어나고 있다.

Satoshi Nakamoto's development of Bitcoin in 2009 has often been hailed as a radical development in money and currency, being the first example of a digital asset which simultaneously has no backing or "intrinsic value" and no centralized issuer or controller. However, another, arguably more important, part of the Bitcoin experiment is the underlying blockchain technology as a tool of distributed consensus, and attention is rapidly starting to shift to this other aspect of Bitcoin.

…

이더리움이 제공하려는 것은 완벽한 튜링완전(turing-complete) 프로그래밍 언어가 심어진 블록체인이다. 이 프로그래밍 언어는, 코딩 된 규칙에 따라 '어떤 상태'를 다르게 변환시키는 기능(arbitrary state transition functions)이 포함된 "계약(contracts)"을 유저들이 작성할 수 있게 함으로써 앞서 설명한 시스템들을 구현 가능하게 할 뿐만 아니라 우리가 아직 상상하지 못한 다른 많은 어플리케이션들도 매우 쉽게 만들 수 있도록 도와줄 것이다. …

What Ethereum intends to provide is a blockchain with a built-in fully fledged Turing-complete programming language that can be used to create "contracts" that can be used to encode arbitrary state transition functions, allowing users to create any of the systems described above, as well as many others that we have not yet imagined, simply by writing up the logic in a few lines of code. …

Conclusion

이더리움 프로토콜은 이러한 어플리케이션들을 직접적으로 제공하는 것이 아니라, 튜링완전언어(Turing-complete programming language)를 통해 이론적으로 거의 모든 형태의 이체방식이나 어플리케이션을 만들어 낼 수 있도록 지원한다. …

The Ethereum protocol would not "support" any of the applications directly, but the existence of a Turing-complete programming language means that arbitrary contracts can theoretically be created for any transaction type or application. …

비트코인

신뢰가 아닌 암호학적인 증명에 기반한 전자 화폐 시스템

탈중앙화 디지털 자산의 첫 번째 사례

분산 합의 수단으로서의 블록체인 기술

이더리움

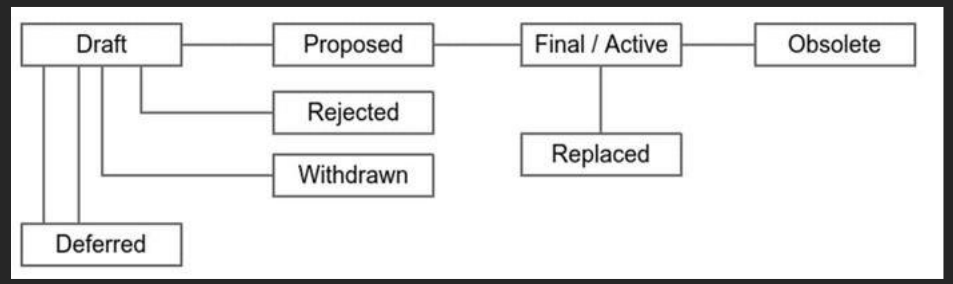
스마트 컨트랙트와 탈중앙화된 어플리케이션 플랫폼

튜링완전(turing-complete) 프로그래밍 언어 지원 블록체인

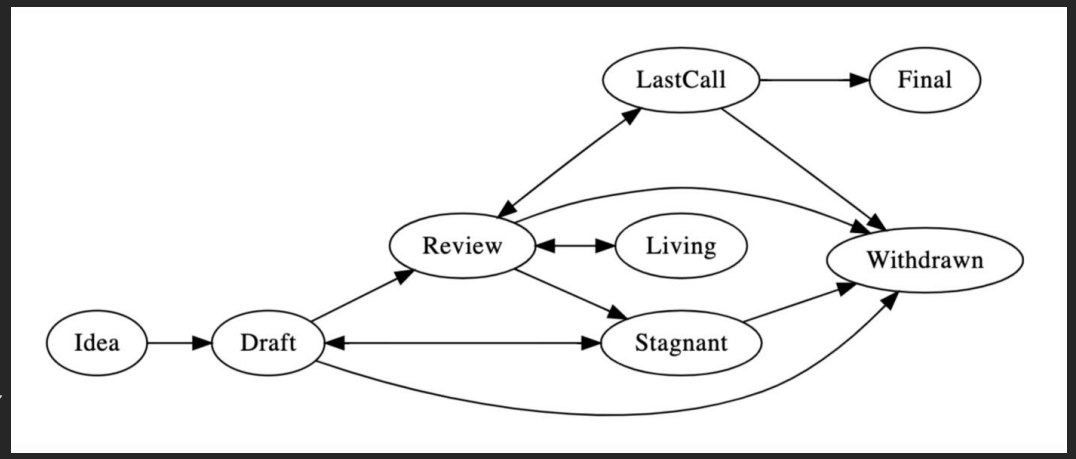
규칙에 따라 '상태'를 변환시키는 기능이 포함된 "계약(contracts)" 작성 가능

거의 모든 형태의 어플리케이션을 만들어낼 수 있도록 지원

- BIP (Bitcoin Improvement Proposals)
  - 비트코인 개선 제안
  - https://github.com/bitcoin/bips



- EIP (Ethereum Improvement Proposals)
  - 이더리움 개선 제안
  - https://github.com/ethereum/EIPs
- ERC (Ethereum Request for Comment)
  - RFC(Request For Comment)의 이더리움 버전
  - EIP 중 Application(예) Smart Contract Format)과 연관된 것
  - ERC-20(Token), ERC-721(NFT), ERC-1155(Multi-Token)
  - https://eips.ethereum.org/erc



- NIST (National Institute of Standards and Technology)
  - 미국 국립표준기술연구소
  - 공식적인 임무는 경제 안보를 강화하고, 삶의 질을 개선하는 방식으로 측정 과학, 표준 및 기술을 진보시켜 미국 혁신과 산업 경쟁력을 증진시키는 것

암호학

10진수, 2진수, 16진수

나머지(modulo) 연산

해시(Hash) 함수

대칭키 암호, 비대칭키 암호(공개키 암호)

ECC(타원 곡선 암호) & 비트코인, 이더리움 주소

10진수(decimal) : [0, 9] : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, …

2진수(binary) : [0,1]

10진수	0	1	2	3	4	5	6	7	8	9	10
2진수	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010

2진수 자릿수별 10진수

2진수	1	10	100	1000	10000
자릿수	2^0	2^1	2^2	2^3	2^4
10진수	1	2	4	8	16

2진수 ➡ 10진수

(2진수) 1011 = 1000 + 10 + 1 ➡ (10진수) 8 + 2 + 1 = 11

10진수 ➡ 2진수

(10진수) 11

2 | 11

2 | 5 ---------- 1

2 | 2 ---------- 1

1 ---------- 0

➡ (2진수) 1011

16진수(hex) : [0, f]

10진수	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2진수	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
16진수	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f

bit : 0 or 1 = 2진수 한자리, **1 nibble** = 4bits, 1 byte = 8bits, 1KB = 1024 bytes, 1MB = 1024 KB, 1GB = 1024 MB, …

큰 10진수(decimal) ➞ 2진수(binary)(bit) ➞ 16진수(hex)**(nibble)**

10진수

5616731559252230693496373227192969113164579 60

2진수 (160 bits)

0000000000011001001011111011000100001101111100110111110010011111101100100110100000101001111010110010110011000110001
0001111001101000110111111010110011001111101000

16진수 (160 bits = 40 nibbles) (Prefix : 0x)

0x00192Fb10dF37c9FB26829eb2CC623cd1BF599E8

```
11110101100110011110100 0
   F   5   9   9   E   8
```

나머지(modular) 연산

- 10 나누기 3 의 몫은 3, 나머지는 1
- 몫 : 10 / 3 = 3
- 나머지 : 10 % 3 = 1 ➡ 10 mod 3 = 1

	-4	-3	-2	-1	0	1	2	3	4	5
% 3	2	0	1	2	0	1	2	0	1	2

$$1 \,\%\, 3 = 1 \,, -2 \,\%\, 3 = 1$$

$$1 \,\%\, 3 = -2 \,\%\, 3$$

$$1 \; mod \; 3 = -2 \; mod \; 3$$

$$1 \;\equiv -2 \; (mod \; 3)$$

해시(Hash) 함수

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

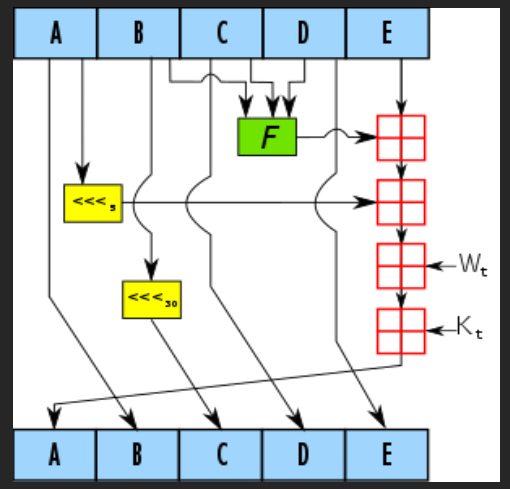
제 2 역상 저항성

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

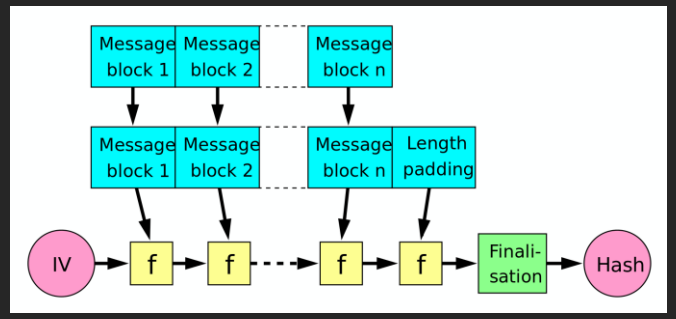
충돌 저항성

Hash(1?) = Hash(2?)

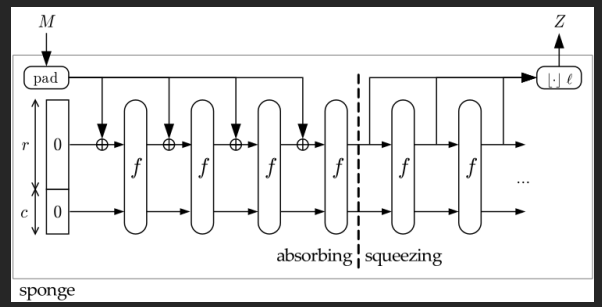
- SHA(Secure Hash Algorithm) (SHA-0) [Input : 2^{64}bit → Output:160bit]
  - 미국 국가안보국(NSA)가 1993년 처음 설계했으면 미국 국가 표준으로 지정
- SHA-1 [Input : 2^{64}bit → Output:160bit]
  - 1995년 SHA-0 변형 발표 (보안 강화)
  - 2017-02-23 Google Announcing the first SHA1 collision 링크
- SHA-2
  - 2001년 SHA1 변형 발표
  - SHA-224, **SHA-256**, 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
```

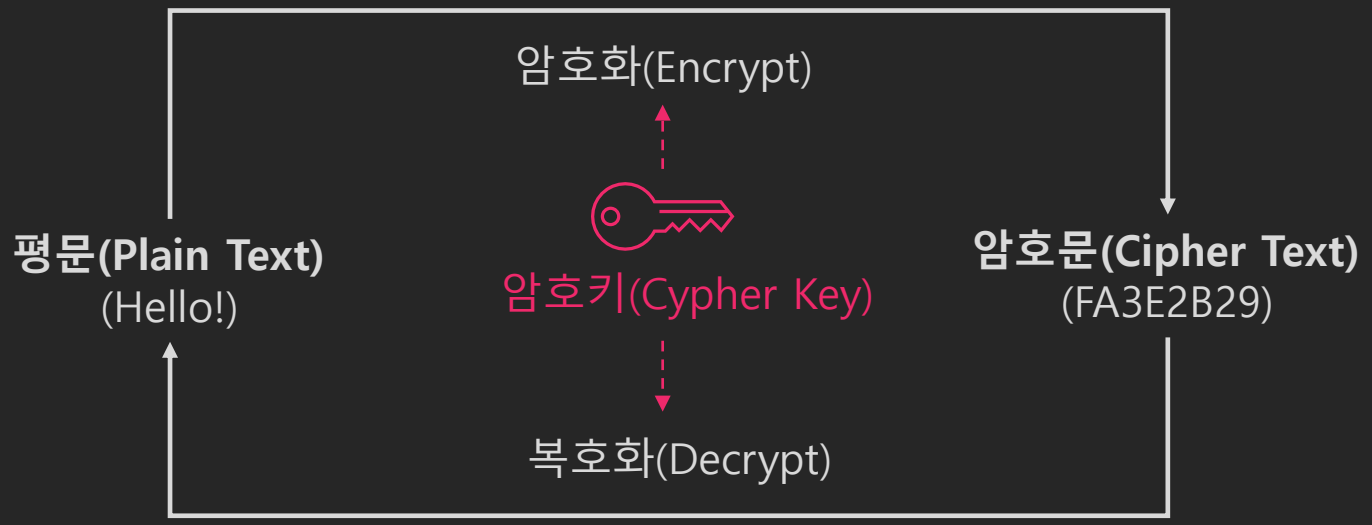
online-tools

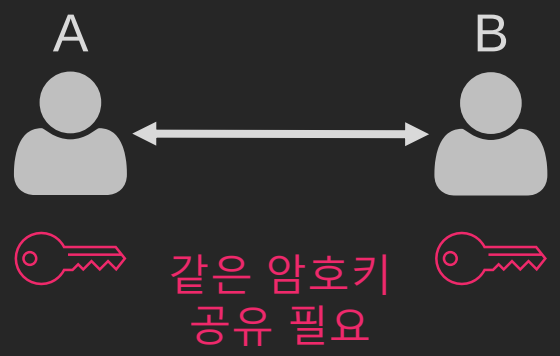
대칭키 암호
(symmetric-key algorithm)

비대칭키 암호 (공개키 암호)
(public-key cryptography)

대칭 키 암호(symmetric-key algorithm)

암호화와 복호화에 같은 키를 쓰는 알고리즘

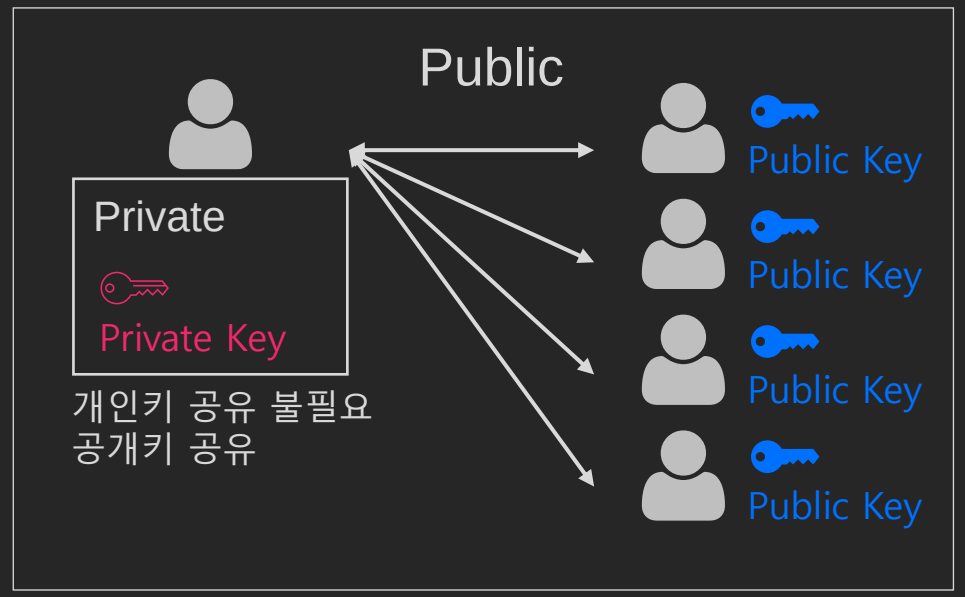




- AES(Advanced Encryption Standard) : 2001-11-26 NIST에 의해 제정된 암호화 방식, Announcement
  - 1997-01-02 NIST가 DES를 대체할 목적으로 개선된 암호 기법 공모, Announcement
  - 빈슨트 레이믄(Vencent Rijment)과 요안 대믄(Joan Daemen)의 개발한 레인달(Rijndael) 알고리즘이 선정
  - AES-126, AES-192, AES-256

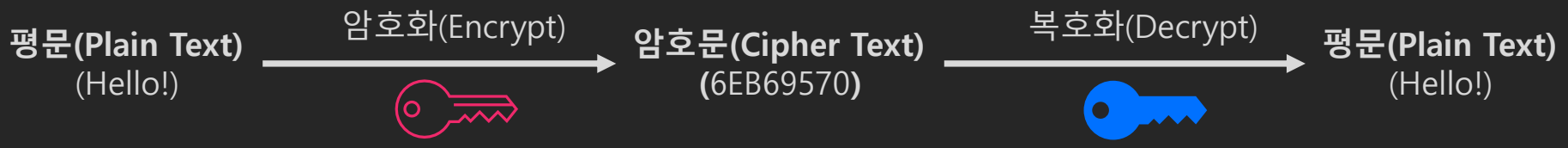
비대칭키 암호 (공개키 암호, public-key cryptography)

암호화와 복호화에 다른 키를 쓰는 알고리즘

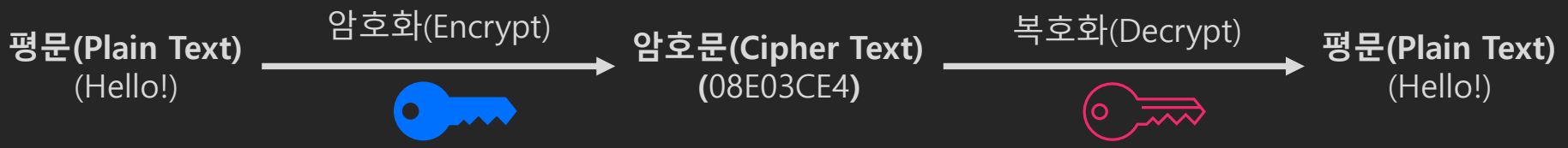


1. 개인키로 공개키 생성 가능, 공개키로 개인키 생성 불가능

2. 개인키로 암호화한 암호문을 공개키로 복호화 가능



3. 공개키로 암호화한 암호문을 개인키로 복호화 가능



	연산 속도	비밀(개인) 키 공유
대칭 키 암호	빠름	필요
비대칭키 암호 (공개키 암호)	느림	불필요

암호 방식

- 공개 키 암호 : 특정한 비밀 키를 가지고 있는 사용자만 내용을 열어 볼 수 있음
- 공개 키 서명 : 특정한 비밀 키로 만들었다는 것을 누구나 확인할 수 있음

원리

- 소인수 분해 : 2 개의 큰 소수의 곱은 간단. 큰 소수의 곱의 소인수 분해는 어려움. (RSA)
- 이산로그 문제 : a 와 c를 알 때, 방정식 $a^b = c$ 의 해 b를 구하는 문제 (디피-헬만 키 교환, 타원 곡선 암호)

주요 공개키 암호화 방식

- 디피-헬만 키 교환(Diffie-Hellman key exchange) : 공개 채널 상 공유 보안 키를 설정하는 최초의 실용적인 방법
- RSA Key Cryptosystem : 암호화뿐만 아니라 전자서명이 가능한 최초의 알고리즘
- 타원 곡선 암호(ECC; Elliptic curve cryptography) : RSA에 비해 짧은 키 길이와 빠른 연산속도로 동일 수준 보안 강도 제공
  - 256 bits 타원 곡선 암호 키는 3072 bits RSA 키와 유사한 보안 강도 제공

용도

- 기밀 내용 전달 (TLS 키 교환)
  - A가 자신만 알고 있는 기밀을 B 에게 전달하고자 할 때 사용. B 를 제외한 타인은 이 내용을 알 수 없어야 함.
    1. B 가 자신의 공개키를 공개
    2. A 는 이 공개키로 문서를 암호화
    3. 암호화된 문서를 B 에게 전달
    4. B 는 자신만이 가진 개인키로 이 문서를 해독
  - 공개키 암호화 처리 속도는 매우 느림으로 공개키 암호화는 TLS의 키 교환 같이 간단한 데이터를 전달하는 용도로만 사용
- 발행자의 증명 및 문서의 변조 방지 (공인인증서)
  - 어떤 문서를 '자신이 작성했음'을 증명하는 용도로 사용. 위·변조 탐지.
    1. A 는 자신의 공개키를 공개
    2. A 는 어떤 문서를 자신의 개인키로 암호화
    3. A 는 암호화된 문서를 일반에 공개하면서,이 문서를 자신이 만들었음을 선포
    4. 타인은 공개된 공개키로 해당 문서를 해독하여 내용을 볼 수 있음
  - 타인은 A의 공개키로 복호화 가능한 문서 생성 불가능. 해당 문서는 A 만이 발행할 수 있다는 강력한 증거
  - 문서가 변조되지 않았음을 확인 가능
- 부인 방지
  - B가 가진 문서에 있는 A의 서명을 A가 부인할 수 있음. B가 문서의 서명이 A 의 것이 맞다는 것을 확인 하는 것
    1. B 는 A 에게 개인키/공개키를 생성한 뒤 공개키를 공개하도록 요구
    2. B 는 A 에게 '문서'를 개인키로 암호화할 것으로 요구
    3. B 는 이 '암호화된 문서'를 수령
    4. B 는 '암호화된 문서'를 A 의 공개키로 해독하여, 이 문서가 A 의 개인키로 제대로 암호화 되었음을 검증 가능
  - '암호화된 문서'는 A의 공개키로만 해독 가능. A만 발행할 수 있다는 증거 및 변조되지 않았음을 증명

- 1976년 마틴 헬먼(Martin Hellman)과 휫필드 디피(Whitfield Diffie)가 공개
- 최초로 발표된 (일종의) 공개키 암호화 방식
- 단순 키 교환 알고리즘일 뿐이므로 암호화/복호화에 대한 내용은 없음
- 공개된 통신 채널상에서 사전 키 공유없이 공유 보안 키를 설정할 수 있는 최초의 실용적인 방법

동작 방식

1. 앨리스(Alice)가 소수 p , 그리고 1 부터 $p-1$ 까지의 정수 g 를 선택하여 사전에 밥(Bob)과 공유
2. 앨리스가 정수 a 를 선택한다. 이 정수는 외부에 공개되지 않으며, 밥 또한 알 수 없음
3. 앨리스가 $A = ga \bmod p$, 즉 g^a 를 p 로 나눈 나머지를 계산
4. 밥이 마찬가지로 정수 b를 선택하여 $\mathrm{B} = \mathrm{gb} \bmod \mathrm{p}$ 를 계산
5. 앨리스와 밥이 서로에게 A 와 B 를 전송
6. 앨리스가 $B^a \bmod p$ 를, 밥이 $A^b \bmod p$ 를 계산

마지막 단계를 통해 공통의 $g^{ab} \bmod p$ 비밀 키를 공유

앨리스 : $B^a \bmod p = (g^b)^a \bmod p = gab \bmod p$
밥 : $A^b \bmod p = (g^a)^b \bmod p = gab \bmod p$

앨리스와 밥 외 인물은 a, b 를 알 수 없고, $g, p, ga \bmod p$, $gb \bmod p$ 를 알 수 있음
p 가 충분히 클 경우, $g^a \bmod p$와$g^b \bmod p$를 통해서 $g^{ab} \bmod p$ 를 알아 내기 어렵다고 알려져 있음
(p : 최소 300자리 소수, a, b 각각 100자리 이상 정수)

ECC(타원 곡선 암호)
(Elliptic Curve Cryptography)

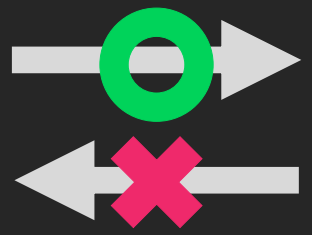
비트코인, 이더리움 주소 (Address)

Key & Address

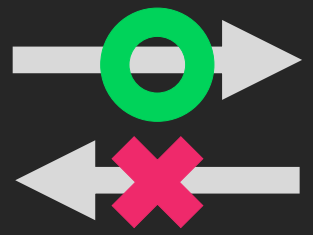
Elliptic Curve Multiplication
(단방향, One-way)

Hashing Function
(단방향, One-way)

Private Key
반복 횟수



Public Key
좌표(x,y)



Address
공개키로 생성

유한체 (Finite Field)

- 집합에 소해 있는 원소의 수가 한정되어 있으며, 덧셈, 곱셈 연산에 대하여 닫혀 있는 집합
  - 닫혀있다 : 연산의 결과 값도 집합에 속해있다
  - 갈로이스 체(Galois Field) 라고로 불림 (갈로아 이론: 체의 대칭성 구조를 군의 구조로 바라다볼 수 있게 한 이론)
  - 원소의 개수가 소수 p (prime)개 인 유한체 : 원소는 " 0, 1, 2, …, (p-1)" 중 하나
  - 표기 : $G\mathbb{F}(p)$ 또는 $\mathbb{F}_p$
- 암호학에서 사용하는 유한체
  - Prime(소수) Field : 원소의 개수인 p 가 홀수 소수(prime)인 유한체 ➡ secp256k1(비트코인, 이더리움 사용)
  - Binary Field : 원소의 개수가 2^m. m-bit string

$$p = 5,\ \mathbb{F}_5 = [0,1,2,3,4]$$

$$y = x^3\ over\ (\mathbb{F}_5)$$

$$y\ mod\ 5 = (x^3)\ mod\ 5$$

x	0	1	2	3	4
x^3	0	1	8	27	64
y	0	1	3	2	4
(x, y)	(0,0)	(1,1)	(2,3)	(3,2)	(4,4)

- 유한체(Finite Filed) : 집합에 혹해 있는 원소의 수가 한정되어 있으며, 덧셈, 곱셈 연산에 대하여 닫혀 있는 집합
  - 닫혀있다 : 연산의 결과 값도 집합에 속해있다
  - 갈로이스 체(Galois Field) 라고로 불림 (갈로아 이론: 체의 대칭성 구조를 군의 구조로 바라다볼 수 있게 한 이론)
  - 원소의 개수가 p 인 유한체 : $G\mathbb{F}(p)$ 또는 $\mathbb{F}_p$ 또는 $(\mathbb{F}_p)^n$ 표시, 원소는 " 0, 1, 2, …, (p-1)" 중 하나
  - 표기 : $G\mathbb{F}(p)$ 또는 $\mathbb{F}_p$ 또는 $(\mathbb{F}_p)^n$
  - GF(p^n) : p = 갈로이스체(Galois Field) 라고로 불림
- 암호학에서 유한체
  - 그래프와 부딪히는 값을 추측하기 어렵도록 정의역과 치역을 소수(prime) p 체계로 한정
  - 연산의 결과 값에 대해 특정 수[암호학에선 소수(prime)) p]를 가지고 mod 연산
- 타원곡선이 암호학에서 적합한 이유: 실수(Real Number) 상에서나 유한체 상에서 연산 모두 동일 수학법칙 적용 가능

ECC (타원 곡선 암호, Elliptic Curve Cryptography)

- 타원 곡선 군의 연산에서 정의되는 이산대수 문제의 어려움에 기반
- 1985년에 닐 코블리츠(Neal Koblitz)와 빅터 밀러(Victor Miller)가 각각 독립적으로 제안
- 기본 공개키 암호 방식(RSA/DSA/Rabin/ElGamal) 대비
  짧은 키 길이와 빠른 연산속도를 가지면서 동일한 수준의 보안 강도를 제공
- ECC는 EC(Elliptic Curve, 타원 곡선)를 사용한 암호 기술의 총체적인 이름
  - 디지털서명: ECDSA(Elliptic Curve Digital Signature Algorithm)
  - 키교환: ECDH(Elliptic-curve Diffie–Hellman)
- 유한체 상의 타원곡선(Elliptic Curve over Finite Field)의 수학적 성질을 이용
  유한체 위에서 타원곡선의 대수적 구조를 기반으로 한 이산로그 문제에 착안
  - 그래프와 부딪히는 값을 추측하기 어렵도록 정의역과 치역을 소수(prime) p 체계로 한정
  - 연산의 결과 값에 대해 소수(prime)) p 로 mod 연산
- 타원곡선이 암호학에서 적합한 이유: 실수(Real Number) 상에서나 유한체 상에서 연산 모두 동일 수학법칙 적용 가능

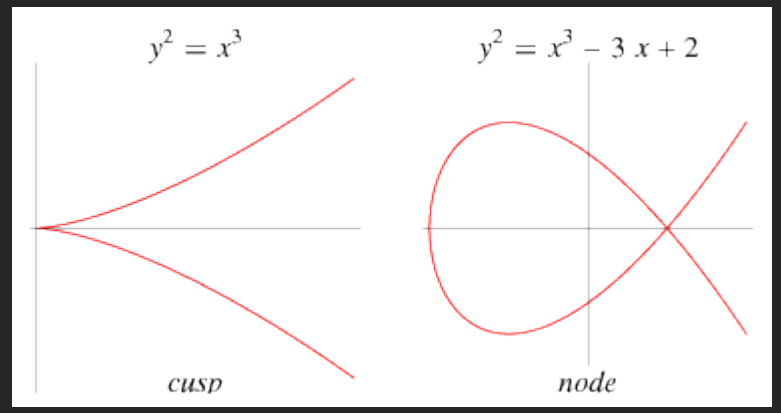
보안 강도 : 키 크기(key size)

RSA	ECC
1024	160
2048	224
3072	256
7680	384
15368	512

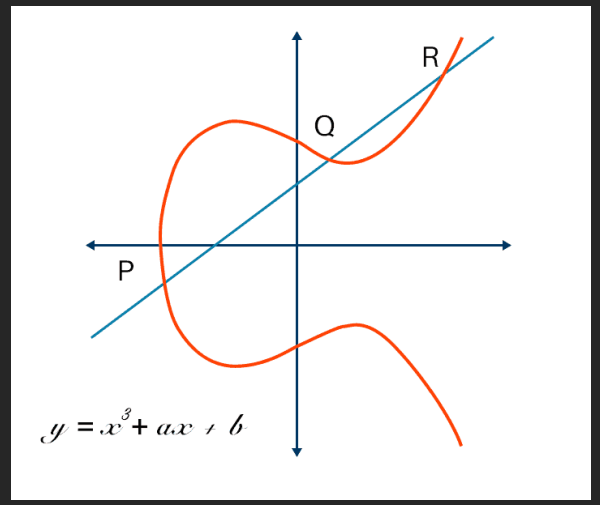
- 타원 곡선(EC, Elliptic Curve)

$$y^2 = x^3 + ax + b, \qquad a, b \in K$$

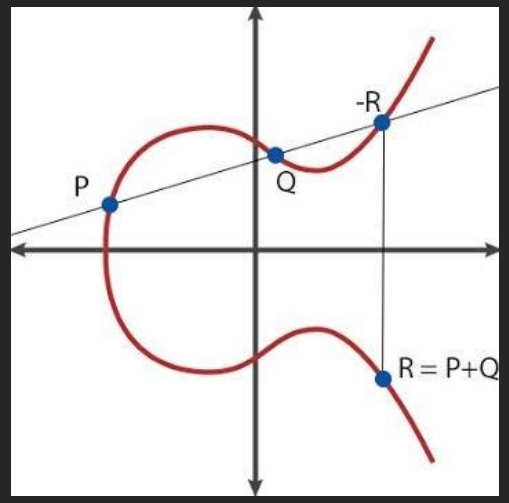
- 비특이 타원 곡선(nonsingular elliptic curve)
  - $4a^3 + 27b^2 \neq 0$
  - 첨점(sharp point), 교차점(intersection point)가 존재하지 않는 타원 곡선
  - $y^2 = x^3 \rightarrow a = 0, b = 0 \rightarrow 4*0^3 + 27*0^2 = 0$
  - $y^2 = x^3 - 3x + 2 \rightarrow a = -3, b = 2 \rightarrow 4*(-3)^3 + 27*2^2 = -108 + 108 = 0$



- 비특이 타원곡선 특징
  - x 축 상에서 y의 값이 정확하게 대칭
  - 두 점 P와 Q를 연결하는 직선은 반드시 다른 한 점(R)을 연결
  - P와 Q가 동일할 때도 성립 : 접선(tangent)

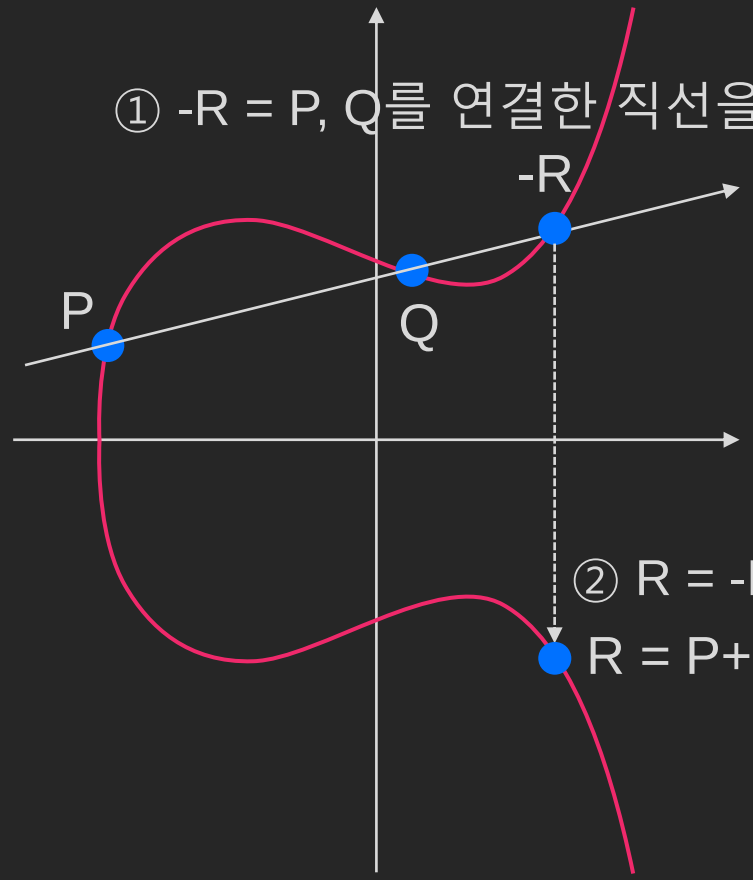


- 타원 곡선 점(point)
  - 점 P = (x, y) , x는 x축의 값, y는 y축의 값
  - 점 0 ($\mathcal{O}$) = (x, 0), y 축의 값이 0인 점, 무한원점(the point at infinity)
  - 점 -P = (x, y), P의 x축 상의 대칭점 (비특이 타원 곡선에서 반드시 존재)
- 점 덧셈 연산
  - 직선의 연결되는 점 P, Q, R에 대해
  - P + Q + R = 0
  - P + Q = -R (R의 대칭점)
  - R + (-R) = R - R = 0
  - ➡ P와 Q를 연결한 직선과 만나는 점 -R의 X축 대칭 점이 R



$$y^2 = x^3 + ax + b$$

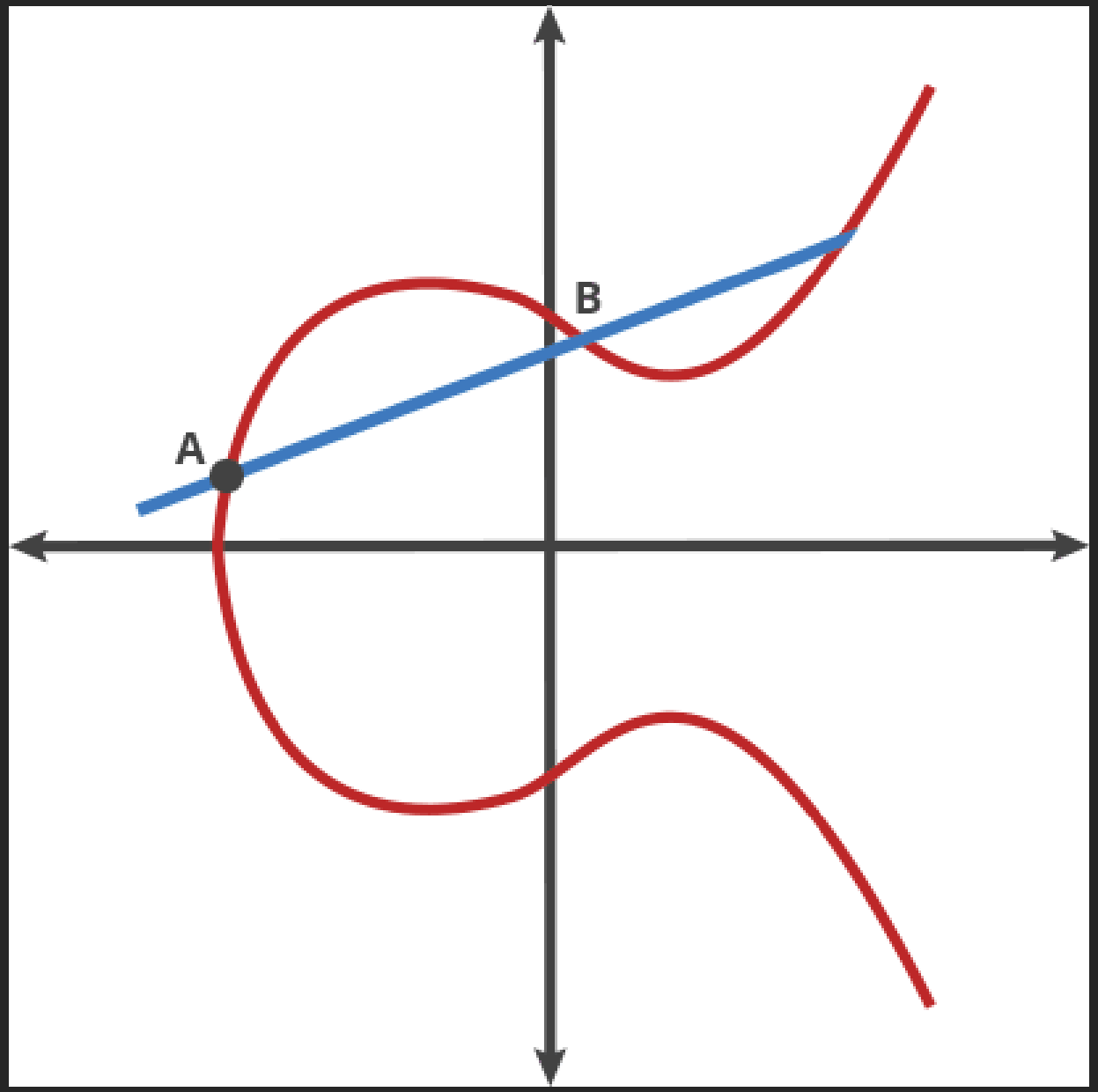
P + Q = R



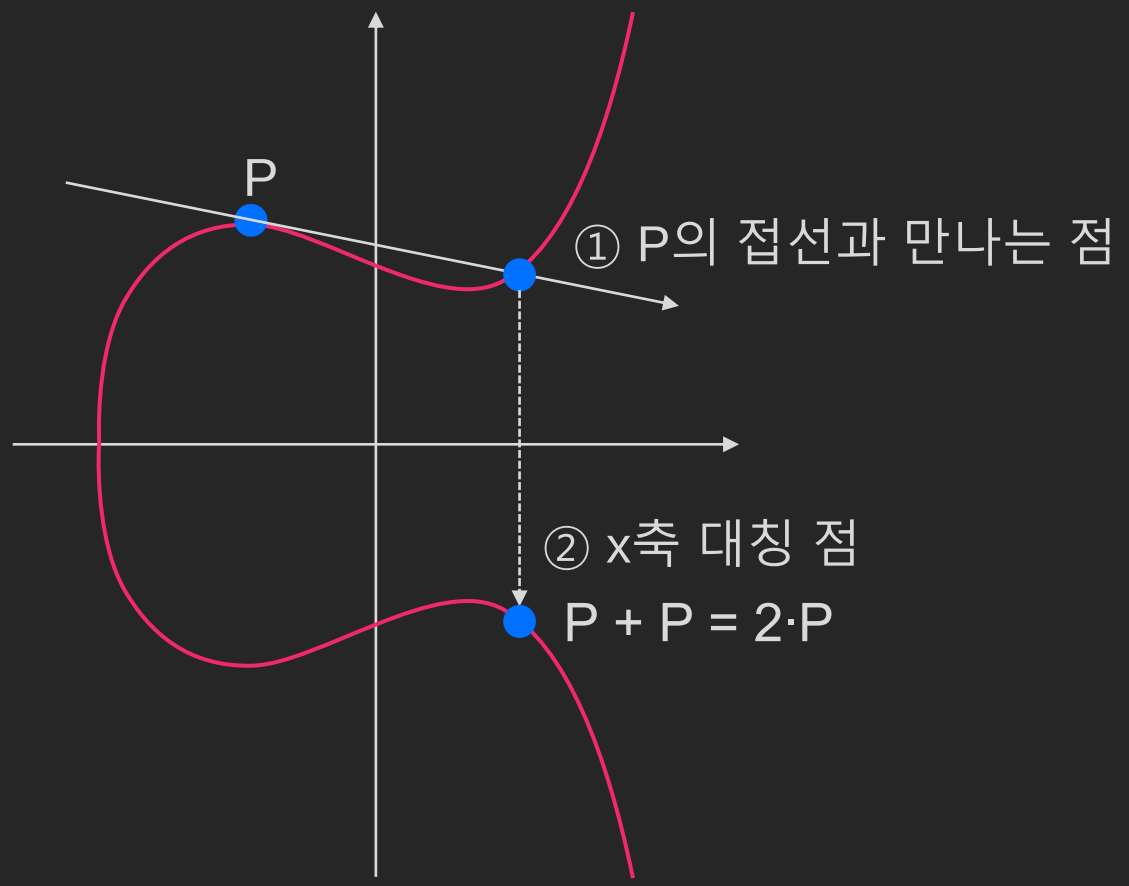
$$A + B = C$$

$$A + C = D$$

$$A + D = E$$



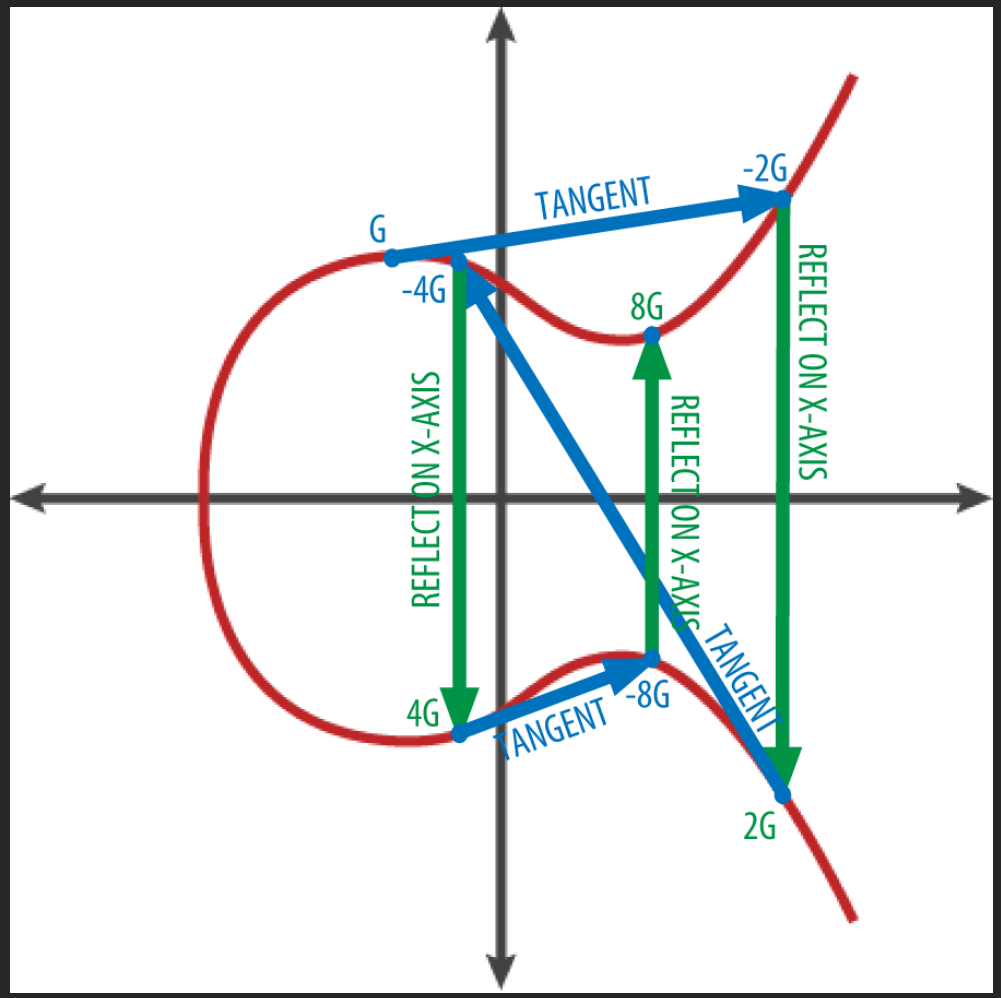
- 동일 점의 덧셈
  - P + P = 2P, P의 접선과 만나는 점의 대칭점



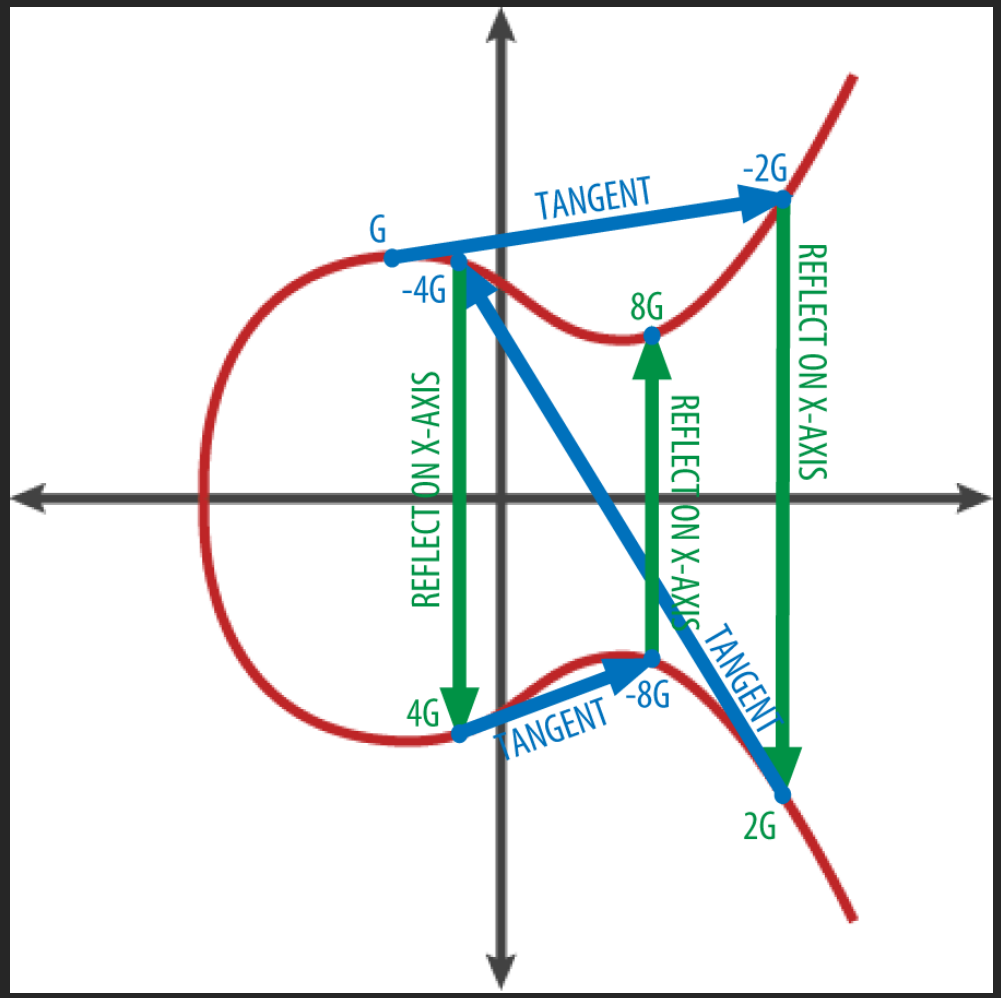
- 동일점 덧셈과 스칼라 곱셈 (scalar multiplication) 연산
  - P + P + … + P = kP (점 P의 k 스칼라 곱), k는 덧셈 횟수
  - 3G = G + G + G
- 타원 곡선 점 곱셈 = 점의 스칼라 곱셈
- 결합법칙, 교환법칙, 분배법칙 성립
  - 3G = (G+ G) + G = G + (G + G), 덧셈 결합법칙
  - 3G = 2G + G = G + 2G , 덧셈 교환법칙
  - (3 + 2)G = 3G + 2G, 덧셈과 스칼라 곱셈 분배법칙
  - 6G = 3(2G) = 2(3G), 스칼라 곱셈 교환법칙
  - 24G = 3(2(4G)) = 4(2(3G)), 스칼라 곱셈 결합법칙
- 항등원, 역원
  - 덧셈 항등원 : 0 ($\mathcal{O}$) $\quad P + \mathcal{O} = P$
  - P의 덧셈 역원 : -P $\quad P - P = \mathcal{O}$
  - 곱셈 항등원 : 1 $\quad 1 \cdot P = P$
  - P의 곱셈 역원 : P^{-1} $\quad P \cdot P^{-1} = 1$

※ 항등원 : 임의의 수 a에 대하여 어떤 수를 연산했을 때 처음의 수 a가 되도록 만들어 주는 수
※ 역원 : 집합 S 와 이항연산자 $*$ 에 대해, 만약 항등원 e 가 존재한다고 할 때, S 의 원소 a 에 대해 $a * b = b * a = e$ 를 만족하는 S 의 원소 b 가 유일하게 존재할 때 b 가 a 의 역원

- 배가 연산(doubling operation)
  - 2G = G + G, G doubling
  - 4G = (2 + 2)G = 2G + 2G, 2G doubling
  - 2^nG 덧셈 횟수 = $n(\log_2 2^n)$
- 빠른 덧셈 연산 : 배가와 덧셈(double-add) 알고리즘
  - k_{max} = n비트 이진수의 최대값 = $2^{n-1} + 2^{n-2} + \ldots + 2^0$
  - $k_{max} = (2^{n-1} + 2^{n-2} + \ldots + 2^0)G = 2^{n-1}G + 2^{n-2}G + \ldots + G$
  - k_{max}G 연산 횟수 = 배가연산 (n-1)회 + 덧셈 연산 (n-1)회 = 2(n-1)회
- k가 256 bits 인 경우
  - kG의 최대 연산 횟수 = 255 + 255 = 510회
  - 배가 연산을 미리 계산하면 최대 덧셈 255회로 줄일 수 있음
- 비특이 타원 곡선에서 알려진 점 G에 대한 kG는 쉽게 계산 가능 (G : Generation Point)



- 타원 곡선 이산 대수 문제
  - K = kG 에서 K와 G가 주어질 때 k를 구하는 문제
- 타원 곡선 이산 대수 문제의 해법
  - 점 K로부터 점 G를 빼는 연산을 반복 수행
  - 알려진 빠른 계산 방법 없음
  - 전통적인 지수 연산의 이산 대수 문제에 비해 훨씬 어려움
- k가 n비트 값인 경우
  - 최대 2^n회의 뺄셈 연산 수행
  - k의 값이 커지면 매우 어려운 문제



- $E(\mathbb{F}_p)$: 소수 유한체 상 타원곡선
  - $\mathbb{F}_p$: p가 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 (G를 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

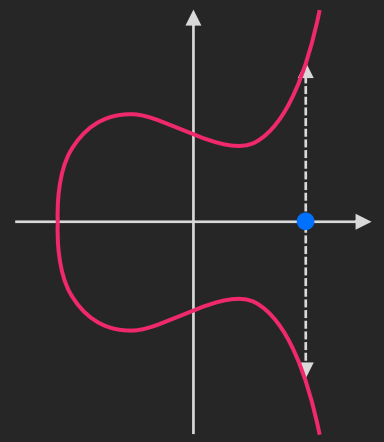
  y = 0x07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB

- 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

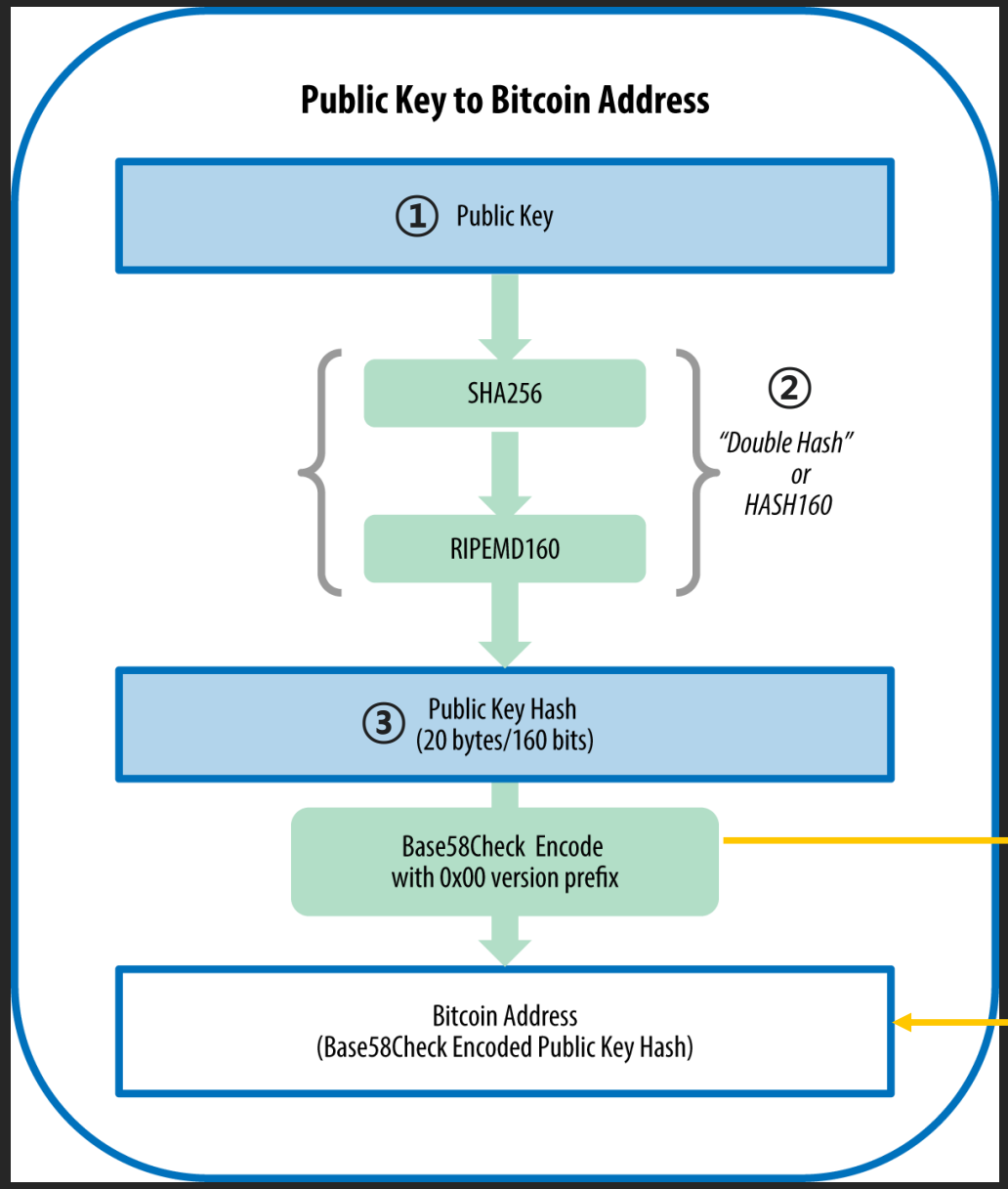
RIPMED-160

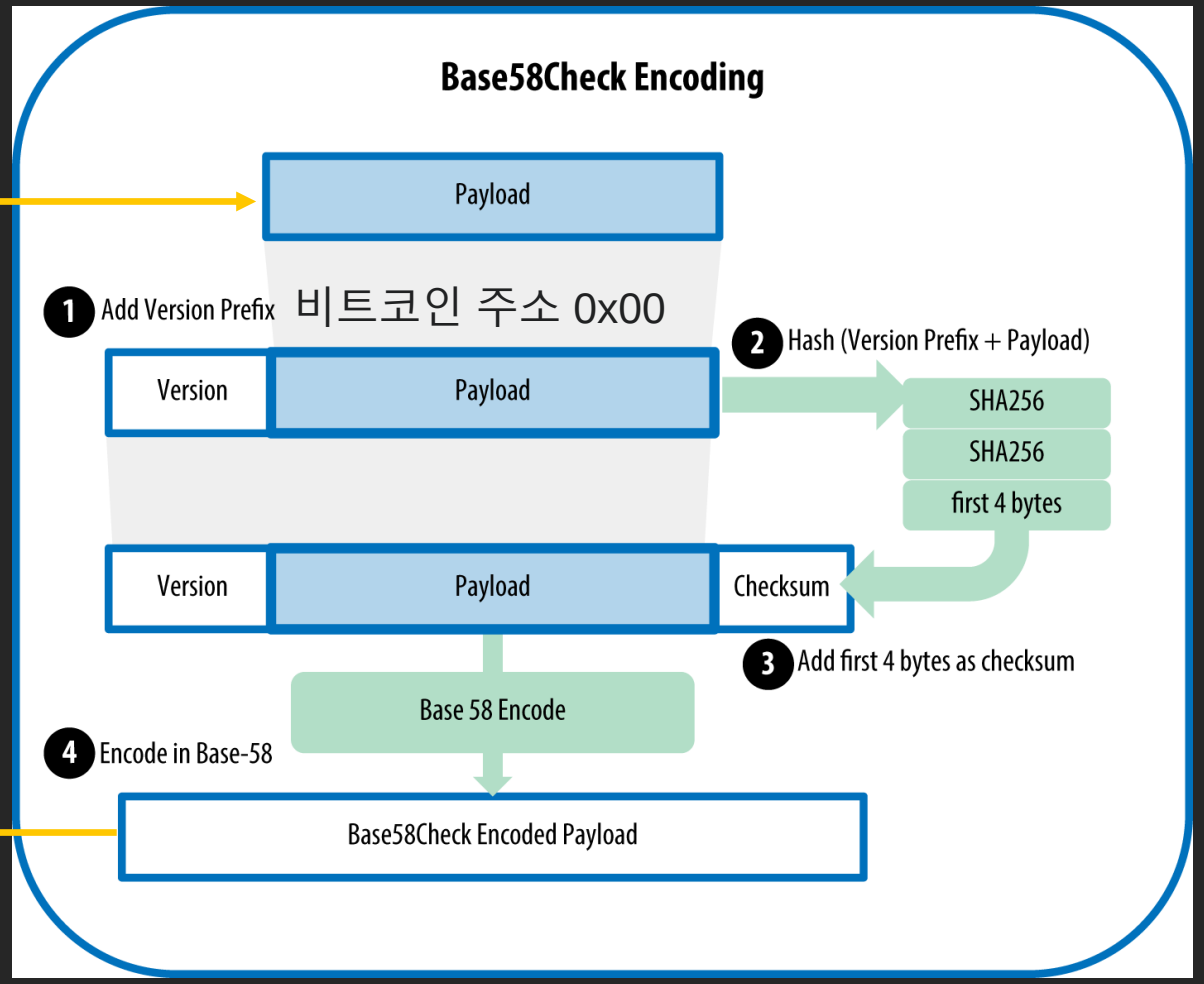
- 임의의 길이의 입력값을 160 bits로 압축하는 암호화 해시 함수
- 유럽의 RIPE 프로젝트 주관 하에 개발
- RIPE : RACE Integrity Primitives Evaluation
- RIPMED : RIPE Evaluation Message Digest

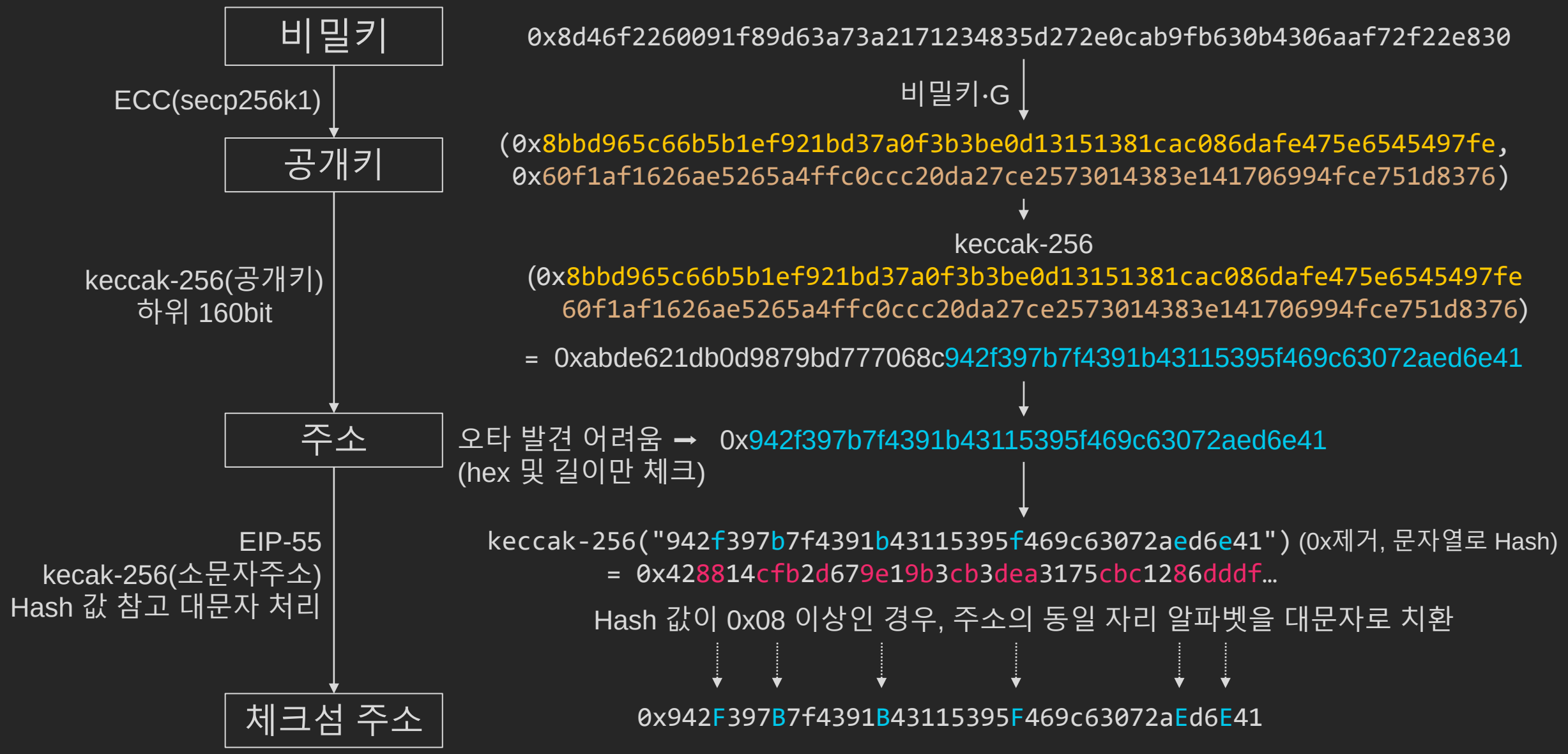
BASE58Check encoding

- 비트코인 주소 인코딩에 사용
- Base64 대신 사용 이유
  - 사람에게 혼동 되는 문자 0OIl 제외 (숫자 영, 영문 대문자 O, 영문 대문자 I, 영문 소문자 l)
  - 더블 클릭 시 전체 선택 방해하는 문자 존재 (+ , /)
- 특징
  - 대문자와 소문자 구분이 쉬운 58개 영숫자 기호 집합 (oOIl+/ 제외)
  - 버전/애플리케이션 정보 1byte Prefix (비트코인 주소 0x00)
  - 4bytes SHA256 체크섬 추가

https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch04.asciidoc






비밀키
0x8d46f2260091f89d63a73a2171234835d272e0cab9fb630b4306aaf72f22e830
ECC(secp256k1)
비밀키·G
공개키
(0x8bbd965c66b5b1ef921bd37a0f3b3be0d13151381cac086dafe475e6545497fe,
0x60f1af1626ae5265a4ffc0ccc20da27ce2573014383e141706994fce751d8376)
keccak-256
keccak-256(공개키)
하위 160bit
(0x8bbd965c66b5b1ef921bd37a0f3b3be0d13151381cac086dafe475e6545497fe
60f1af1626ae5265a4ffc0ccc20da27ce2573014383e141706994fce751d8376)
= 0xabde621db0d9879bd777068c942f397b7f4391b43115395f469c63072aed6e41
주소
오타 발견 어려움 → 0x942f397b7f4391b43115395f469c63072aed6e41
(hex 및 길이만 체크)
EIP-55
kecak-256(소문자주소)
Hash 값 참고 대문자 처리
keccak-256("942f397b7f4391b43115395f469c63072aed6e41") (0x제거, 문자열로 Hash)
= 0x428814cfb2d679e19b3cb3dea3175cbc1286dddf…
Hash 값이 0x08 이상인 경우, 주소의 동일 자리 알파벳을 대문자로 치환
체크섬 주소
0x942F397B7f4391B43115395F469c63072aEd6E41

주소

- 비밀키 -> 공개키 -> 계정의 주소
- 비트코인 주소 생성 방법
  - 비밀키 -> ECC(secp256k1) -> 공개키 -> SHA-256 -> RIPEMD160 (20바이트)
    -> prefix 0x00 (1바이트) ->
    postfix checksum(0x00 + 공개키 해시값의 SHA256 -> SHA256 -> 결과의 맨앞 4바이트
    -> 0x00 + 공개키 해쉬값 + checksum => 총 25바이트 => Base58
  - RIPEMD160 -> 학계에서 만든 output 160bit 암호화 해쉬 함수 (많이 사용되지 않음)
  - Base58 Check 인코딩
    - 0 ~ 57 을 1~9,A~Z,a~z 까지 매핑
      - BASE64의 0(숫자 0), O(대문자 O), I(대문자 아이), l(소문자 엘)처럼 모양이 비슷해 혼동을 일으키는 몇 글자를 제외하고 만든
      - Base64는 공식 매핑 테이블이 있으나 base58은 응용프로그램마다 다를 수 있다. 여기 표는 비트코인 base58 매핑표
    - 예전 이메일이 이진 파일을 처리하지 못하던 시절 이메일에 이진 파일을 첨부하기 위해서 사용
    - 매핑표를 이용하여 58진수 처리로 생각하면 된다.
      - 주의: 맨 앞에 0이 연속한 경우: 맨 앞의 0을 바이트 단위로 분리하여 모두 1로 변환
      - 나머지는 58진번 변환을 활용해 두 수를 합친다
      - 3으로 인해 항상 맨 앞에 최소 1바이트의 0이 존재
      - 0의 바이트 수 만큼 1 + Base58 결과
    - 특징
      - 비트코인 주소는 항상 1로 시작한다.
      - 더 이상 몫이 없을 때까지 반복해 그 나머지를 매핑하는 방식 -> 숫자에 따라 나누는 횟수가 달라질 수 있고, 그 결과 비트코인 주소의 길이는 가변이다.
        최대 34자
  - 2012년 소프트 포크로 P2PH 새로운 트랜잭션 방식 추가 -> 3으로 시작하는 주소 존재
    - P2PKH : original, pay to public key hash
    - P2SH : current, pay-to-script hash (대부분의 월렛) – now standard (더 많은 기능 제공)
- 이더리움 주소 생성 방법
  - 비밀키 -> ECC(secp256k1) -> 공개키 -> Keccak256 -> 하위 160bit
  - EIP-55 (Mixed-case checksum address encoding)
    - 주소 -> 소문자로 변환 -> keccak256 -> keccak256 결과의 앞쪽 160bit를 사용 -> keccak256 결과의 nibbl이 8이상이고, 주소가 alphabet인 경우 대문자로 치환

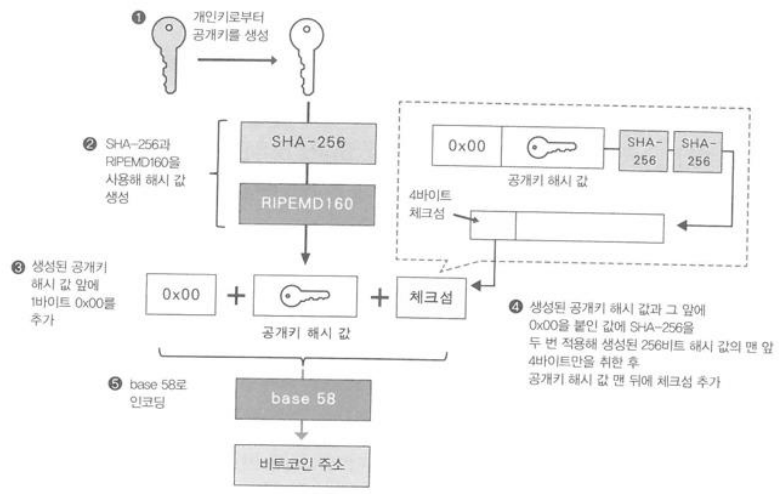


그림 3-1 비트코인 주소의 생성

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	18	19	20
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	J	K	M

21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
N	P	Q	R	S	T	U	V	W	X	Y	Z	a	b	c	d	e	f	g	h

41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57
i	j	k	m	n	o	p	q	r	s	t	u	v	w	x	y	z

그림 3-2 비트코인 base 58 매핑

NIST SP 800-57 Part 1 (Recommendation for Key Management: Part 1 – General)

보안강도(비트)	Hash		대칭키 암호	비대칭키 암호		허용기간
	SHA2	SHA3	AES	IFC(RSA)	ECC(ECDSA)	
≤ 80				1024	160-223	미허용
112	224	224		2048	224-255	~2030년
128	256	256	128	3072	256-383	2031년 ~
192	384	384	192	7680	384-511	2031년 ~
≥ 256	512	512	256	15360	512+	2031년 ~

비트코인

- Hash : SHA256 (256 bits)
- ECC(ECDSA) : secp256k1 (256 bits)

이더리움

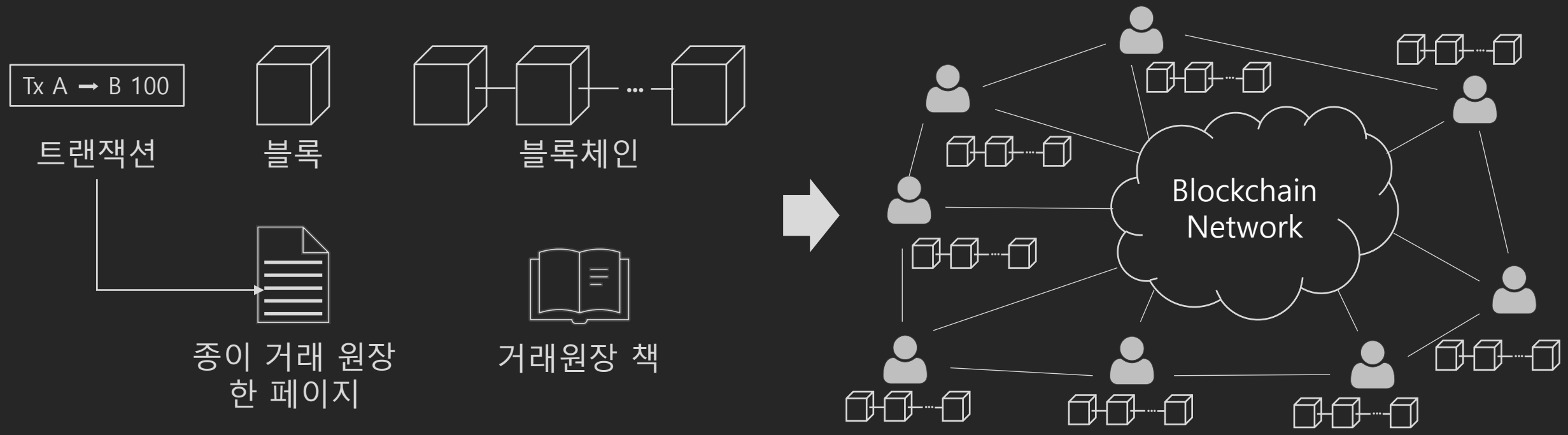
- Hash : keccak-256 (256 bits) [SHA3 Original Proposal]
- ECC(ECDSA) : secp256k1 (256 bits)

Blockchain

Transaction
Block
Blockchain

원장(Ledger) : 주요 거래(Transaction)를 계정(Account)과 잔고별로 모은 장부 문서

- 트랜잭션 = 거래 한 건(Transaction)
- 블록(Block) = 한 페이지(Page)
- 블록체인(Blockchain) = 원장(Ledger)
- 분산 원장 (Distributed Ledger) : 네트워크 참여자 모두 블록체인 복사본 저장



트랜잭션(Transaction, Tx) : 거래 한 건

- A가 B에게 100 비트코인 이체 | Tx0 A → B 100 BTC |
- B가 C에게 200 비트코인 이체 | Tx1 B → C 200 BTC |

블록 : 트랜잭션의 묶음

Block

Tx0 A → B 100 BTC
Tx1 B → C 200 BTC

블록체인 : 블록의 묶음 - Genesis Block : 첫번째 블록(0번 블록)

Block 0

Tx0 A → B 100 BTC
Tx1 B → C 200 BTC

Genesis Block
첫번째 블록

Block 1

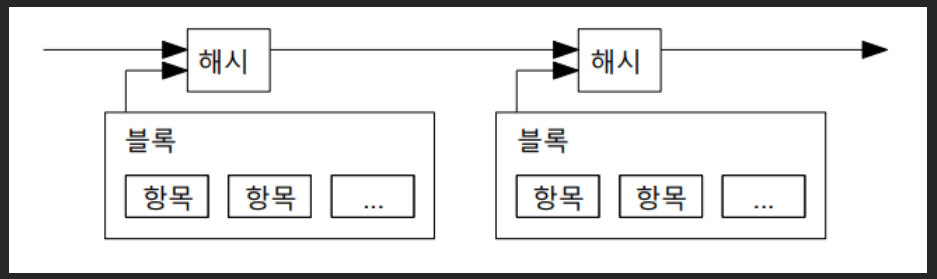
Tx0 C → D 300 BTC
Tx1 A → D 100 BTC

Block 2

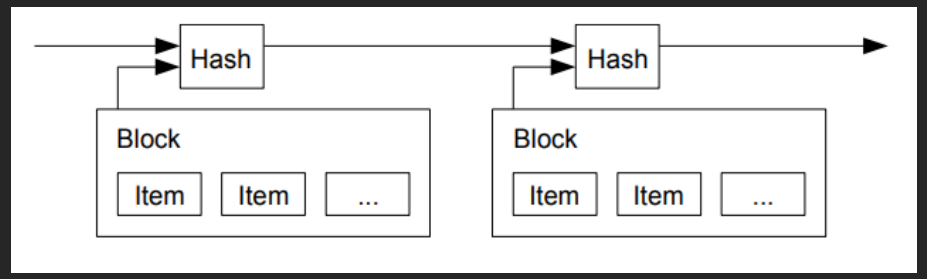
Tx0 E → A 100 BTC
Tx1 D → B 400 BTC

Bitcoin - 3.Timestamp Server

우리가 제안하는 해법은 타임스탬프 서버로 시작한다. 타임스탬프 서버는 타임스탬프가 찍힌 항목 블록의 해시를 가져가 그 해시를 신문이나 유즈넷 게시물[2-5]처럼 널리 배포하는 식으로 작동한다. 이 타임스탬프는 그 데이터가, 명백히, 해시(과정)에 들어가기 위해 해당 시각부터 존재했음을 증명한다. 각 타임스탬프는 그 해시 안 에 먼젓번 타임스탬프를 포함하고, 그에 앞선 것들을 하나씩 연장하는(reinforcing the ones) 타임스탬프가 찍힌 사슬을 생성한다.

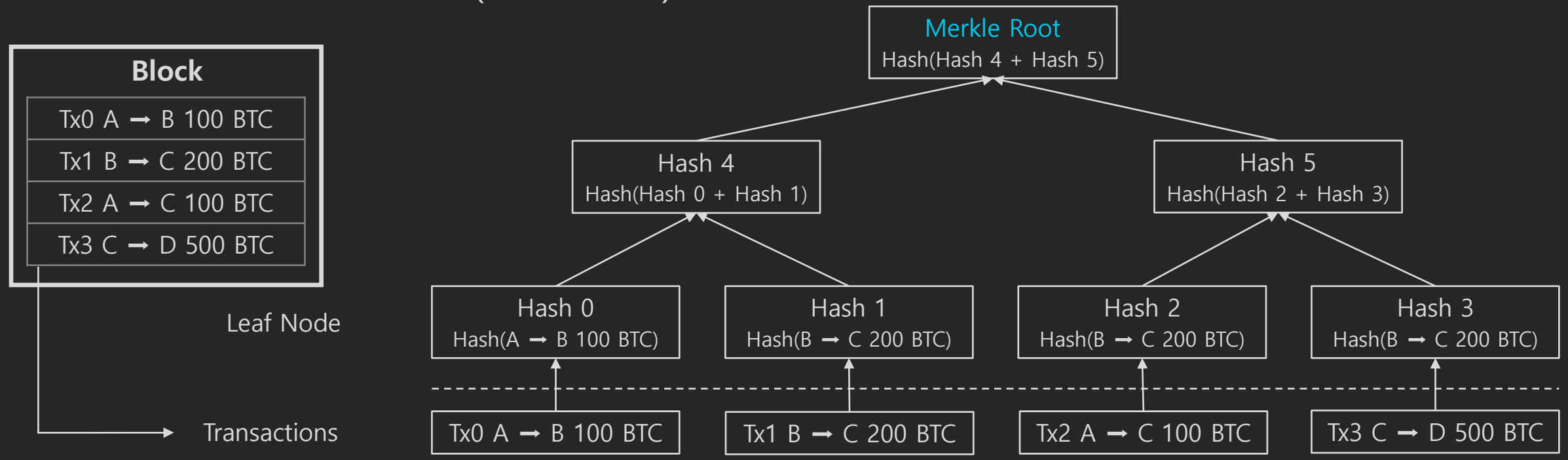


The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



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)

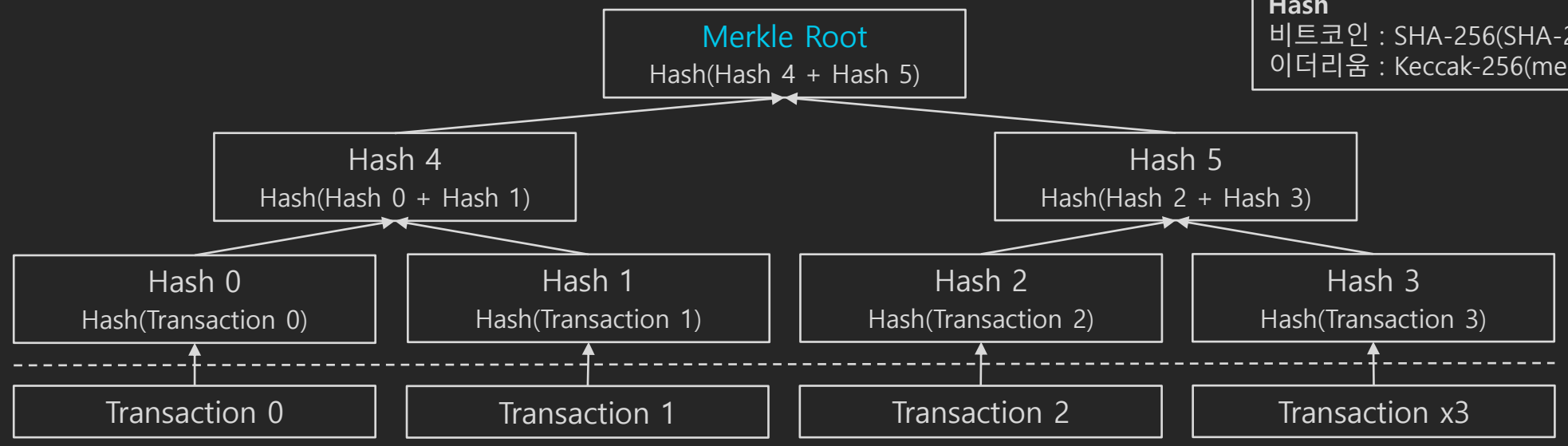


Bitcoin Hash

Hash = SHA-256(SHA-256(message))

블록과 트랜잭션 [바이너리 머클 트리(Merkle tree) Root]

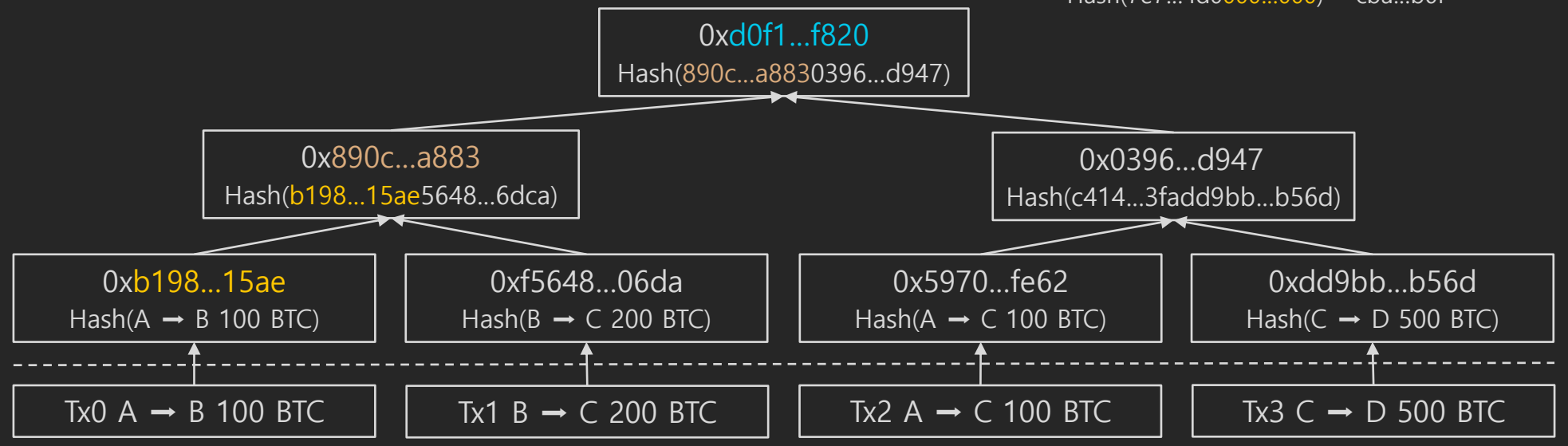
Hash
비트코인 : SHA-256(SHA-256(message))
이더리움 : Keccak-256(message)



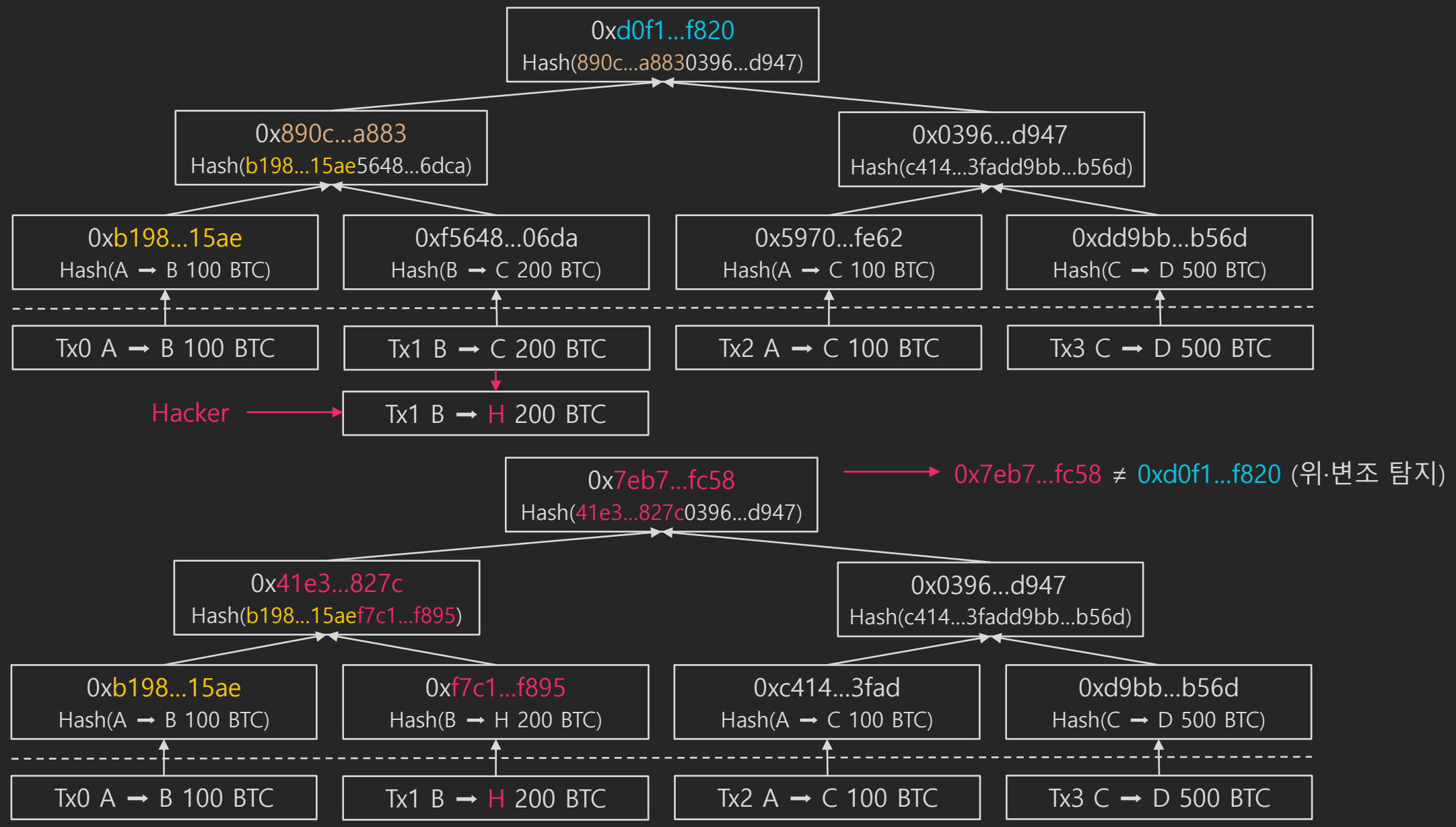
Hash(A → B 100 BTC) = SHA-256(SHA-256(A → B 100 BTC))
= b1986b79781eb8b52d3159e41c73a26565297bf5aaa753c25e9520d81f4b15ae
= b198...15ae

Bitcoin

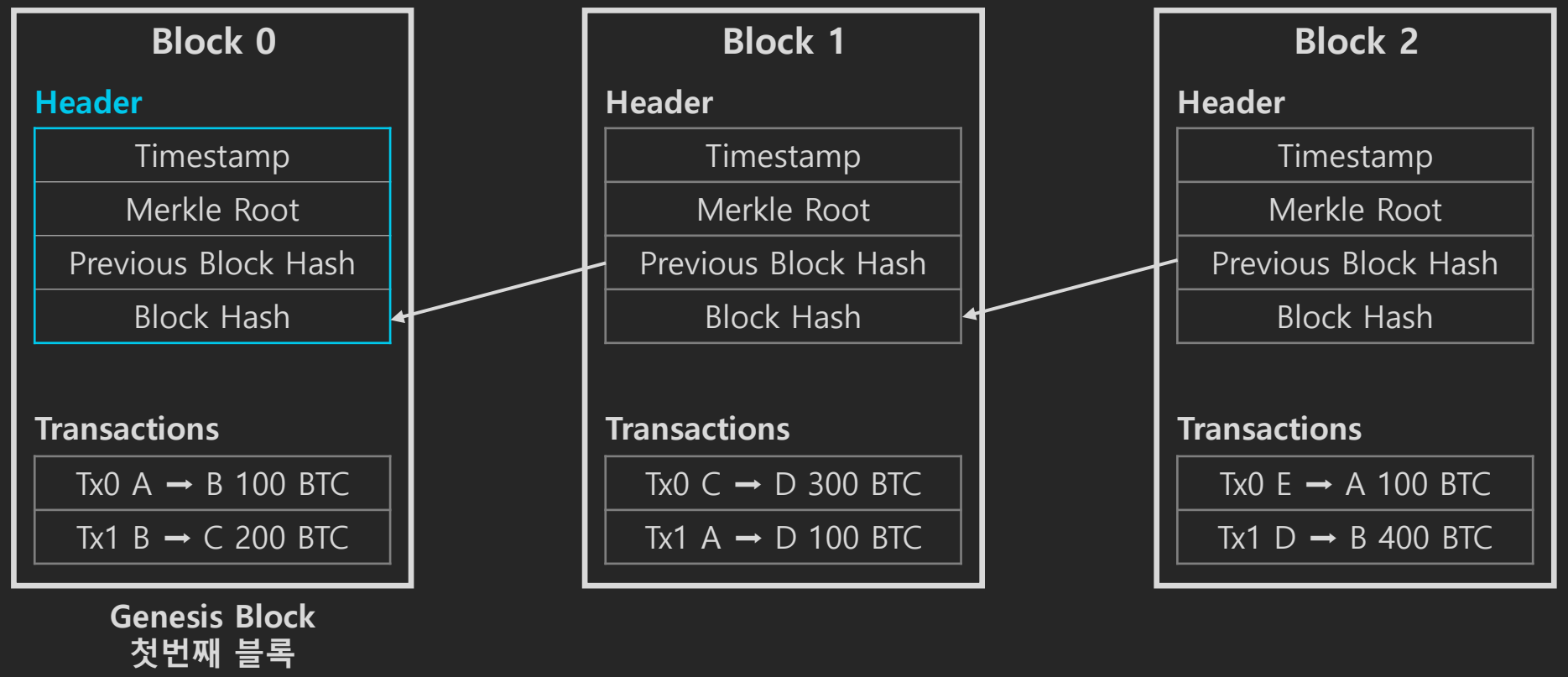
※ 마지막 노드의 짝이 없는 경우 0x000...000 추가 후 Hash
Hash(7e7...4d0000...000) = cba...b6f



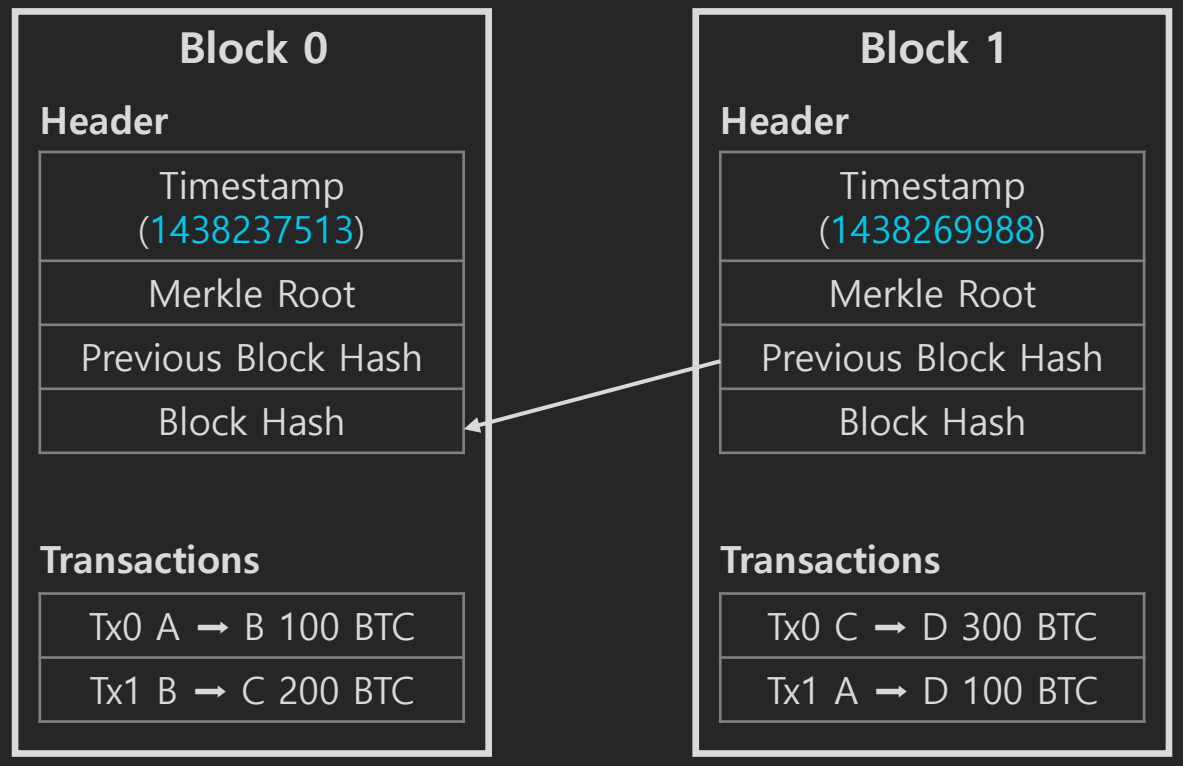
블록과 트랜잭션 [머클 증명 (Merkle Proof),트랜잭션 위·변조 탐지]



- 블록 헤더(Header)
  - 타임스탬프 (Timestamp) : 블록 생성 시간 (= 트랜잭션 존재 시간)
  - 머클 루트 (Merkle Root) : 트랜잭션 목록 머클 트리의 머클 루트
  - 이전 블록 해시 (Previous Block Hash)
  - 블록 해시 (Block Hash) : Hash (Timestamp + Merkle Root + Previous Block Hash)



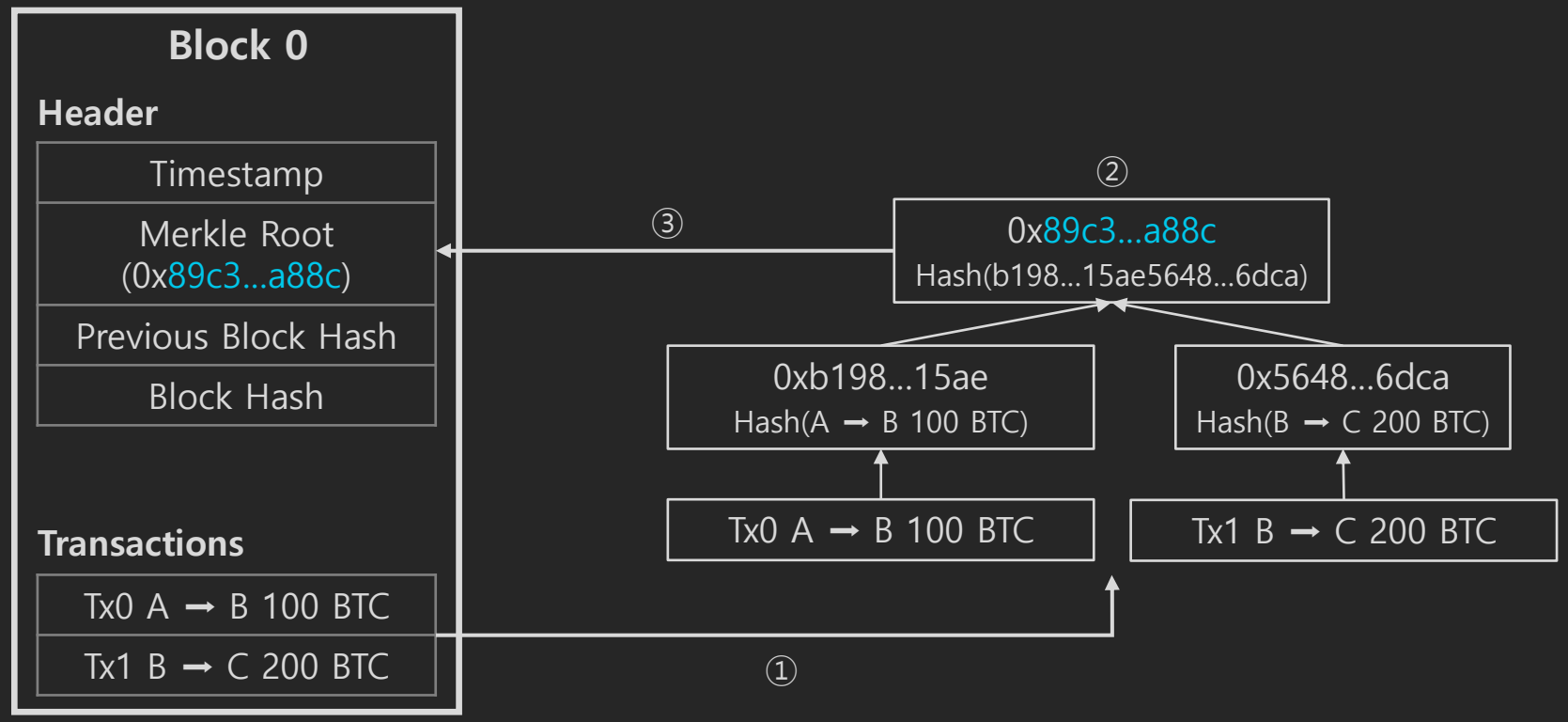
- 타임스탬프 (Timestamp) - '시간(time)' + '도장을 찍다(stamp)' 의 합성어
  - 시간 표기방법
  - 1970-01-01 00:00:00.000 을 0으로 시작해서 초(second) 또는 밀리초(millisecond) 로 표시 (1s = 1000ms)
    - 초 (s) 기준 : 1 = 1970-01-01 00:00:01
    - 밀리초(ms) 기준 : 1 = 1970-01-01 00:00:00.001
  - 비트코인, 이더리움 : 초 단위 사용
- 블록체인 타임스탬프
  - 블록 생성 (시작) 시간
    - 트랜잭션 존재 시간 : 실제 실행 시간이 아님
  - 블록의 순서
    - 새 블록은 이전 블록보다 항상 미래
  - 존재 증명 : 데이터가 특정 시점에 존재하고 있었다



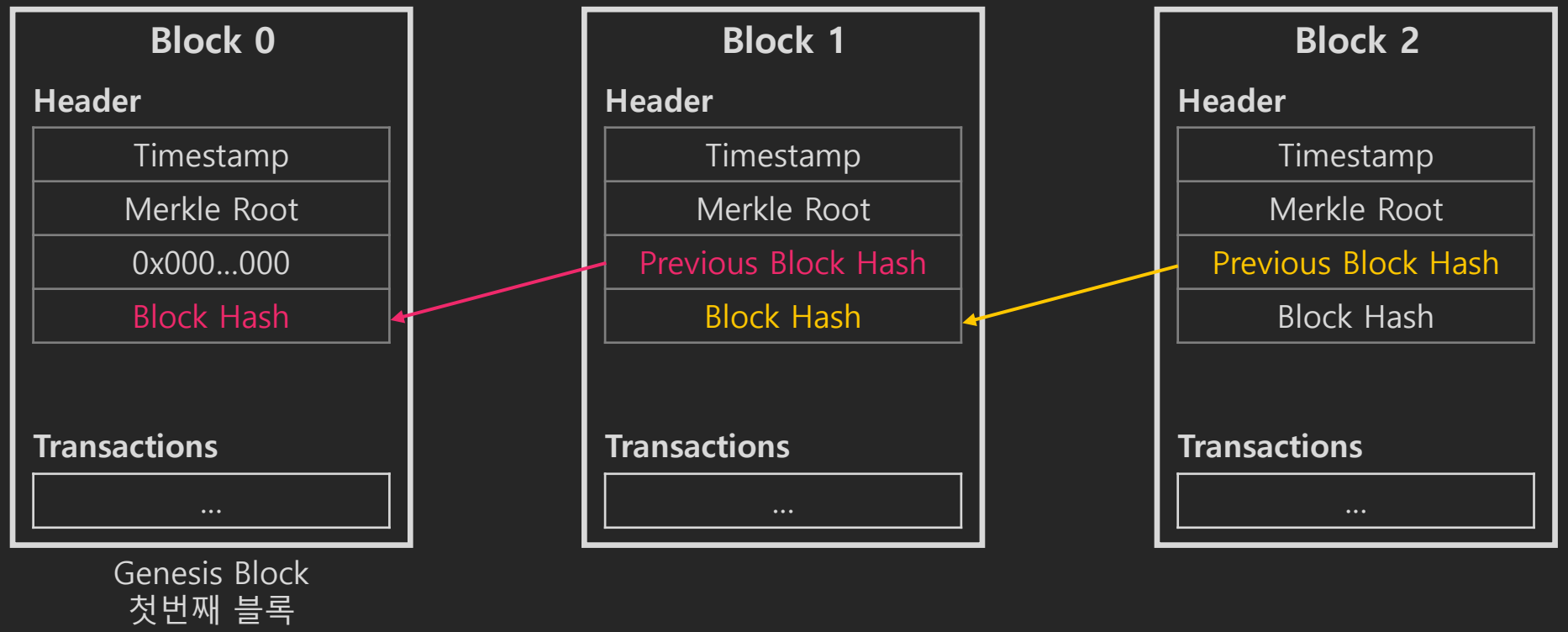
Genesis Block
첫번째 블록

1438237513 : 2015-07-30 03:26:13 PM +UTC
1438269988 : 2015-07-30 03:26:28 PM +UTC

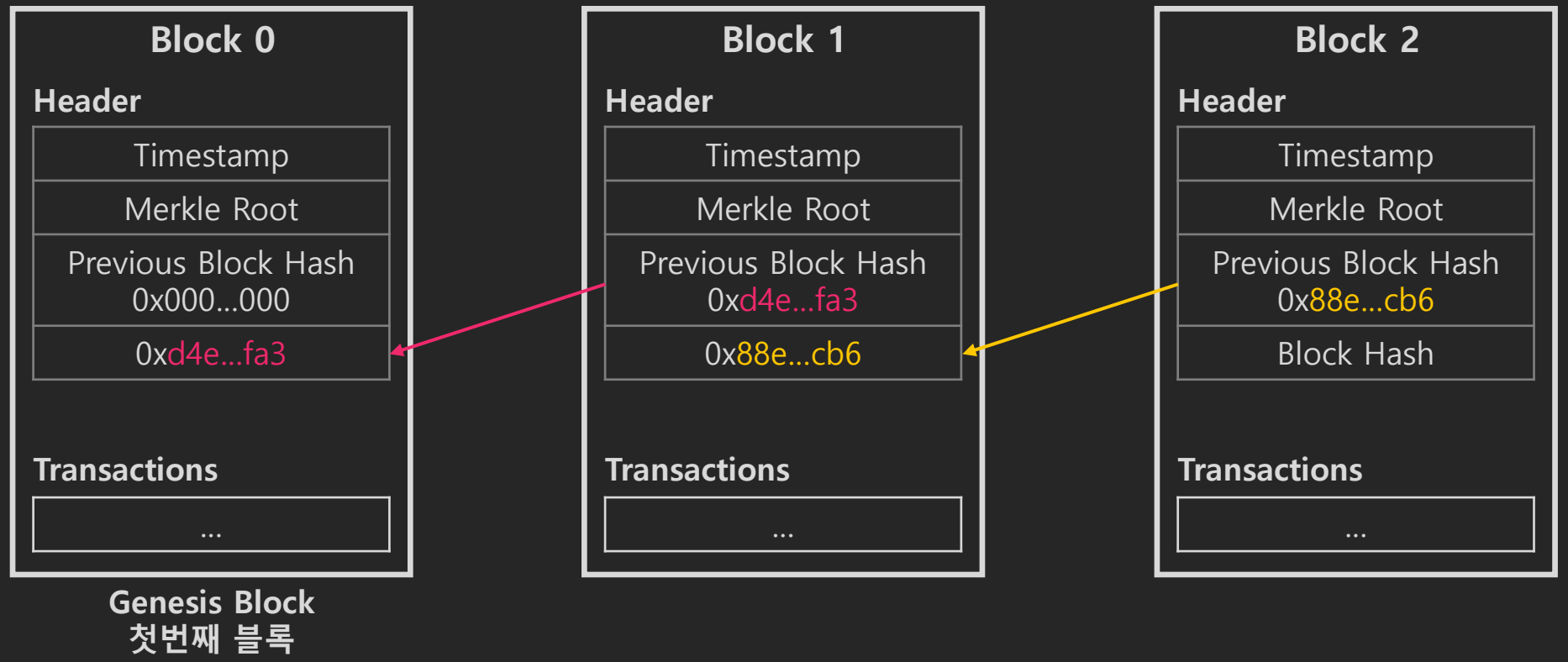
- 머클 루트 (Merkle Root)
  - 트랜잭션들의 머클 트리 루트
  - 내용 증명 : 트랜잭션 데이터가 변경되지 않았다



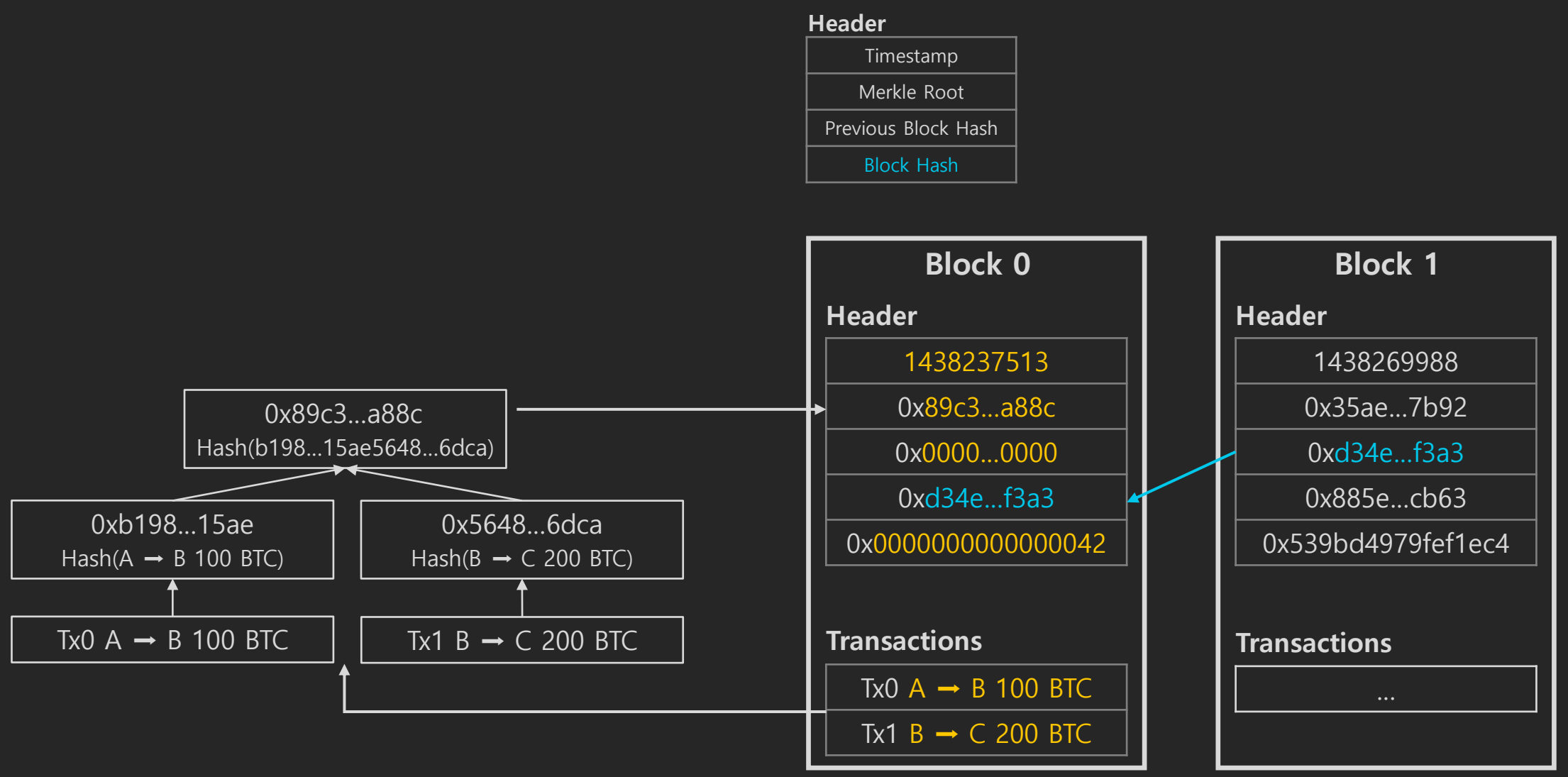
- 이전 블록 해시 (Previous Block Hash)
  - 이전 블록과 관계 형성
  - Genesis Block의 Previous Block Hash:
    0x0000000000000000000000000000000000000000000000000000000000000000

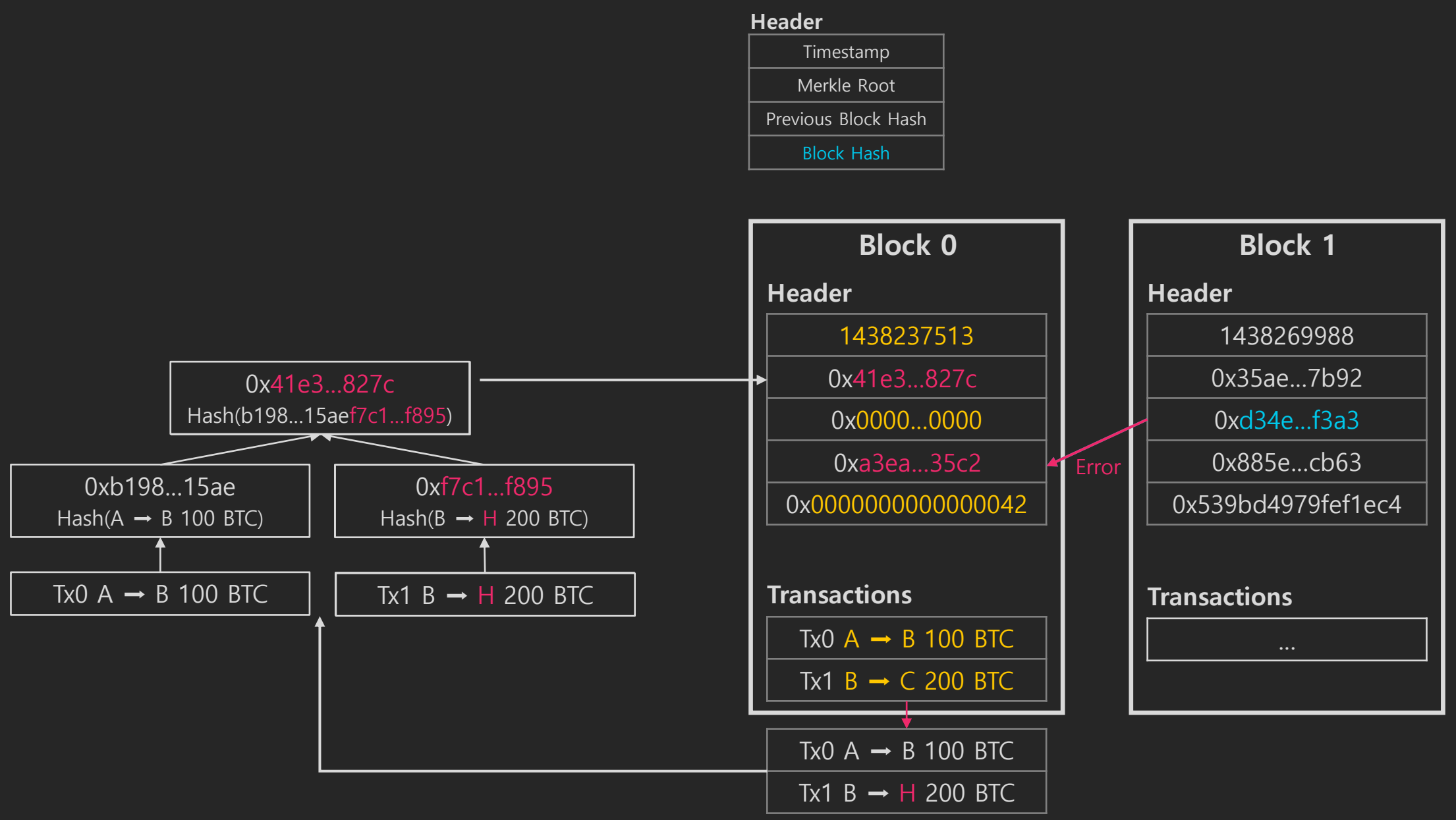


- 블록 해시 (Block Hash)
  - Timestamp, Merkle Root, Previous Block Hash 등의 Block 정보를 Hash로 변환한 값
  - Block Hash = Hash(Timestamp + Merkle Root + Previous Block Hash + 기타 정보)
  - 내용 증명 : 블록 데이터가 변경되지 않았다

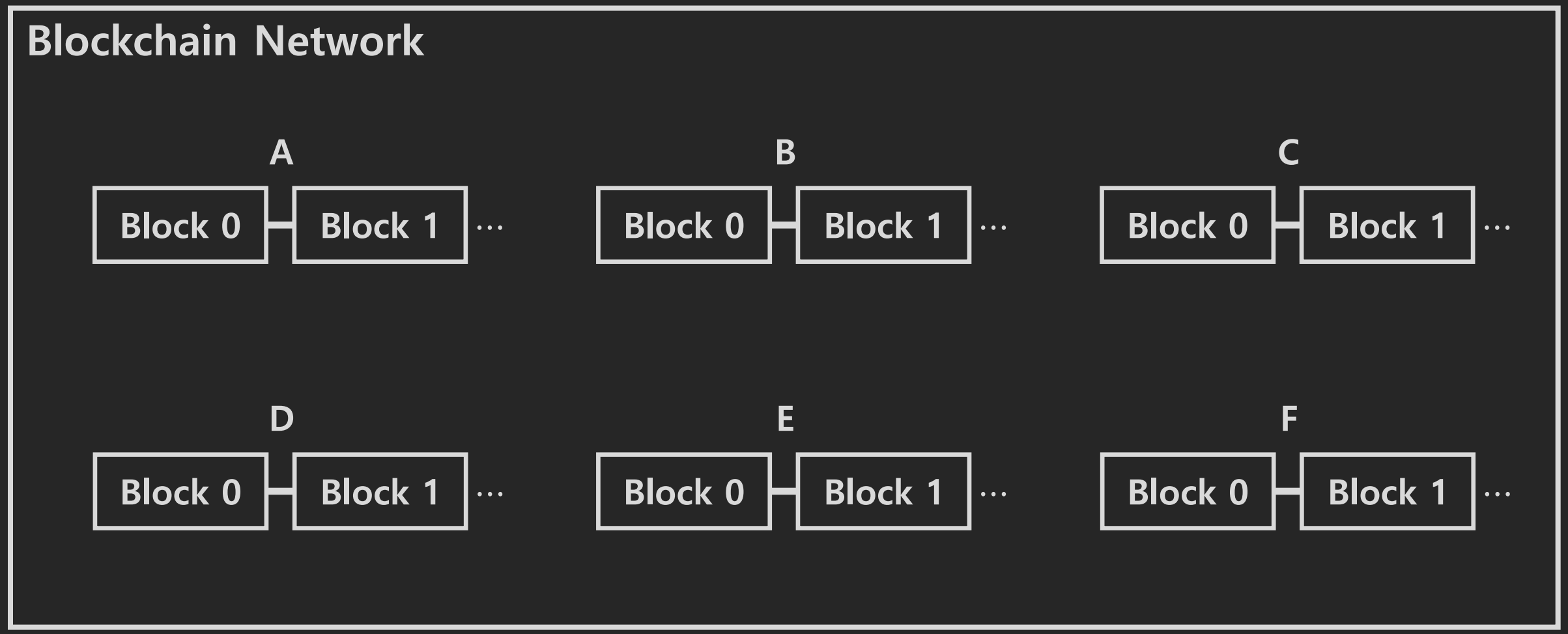


블록과 블록체인 [머클 루트, 블록 해시, 이전 블록 해시]

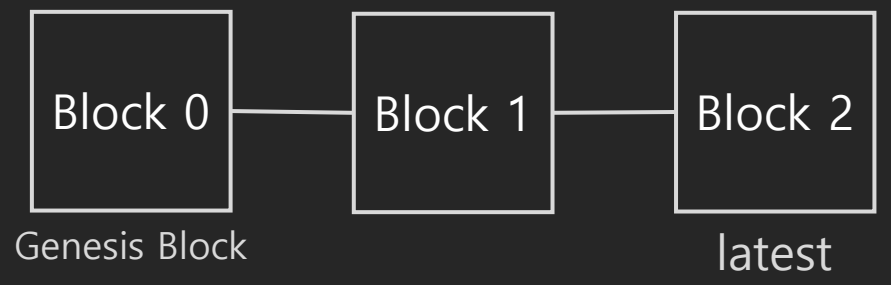



Header
Timestamp
Merkle Root
Previous Block Hash
Block Hash
Block 0
Header
1438237513
0x41e3...827c
0x0000...0000
0xa3ea...35c2
0x0000000000000042
Transactions
Tx0 A → B 100 BTC
Tx1 B → C 200 BTC
Block 1
Header
1438269988
0x35ae...7b92
0xd34e...f3a3
0x885e...cb63
0x539bd4979fef1ec4
Transactions
...
Error
0x41e3...827c
Hash(b198...15aef7c1...f895)
0xb198...15ae
Hash(A → B 100 BTC)
0xf7c1...f895
Hash(B → H 200 BTC)
Tx0 A → B 100 BTC
Tx1 B → H 200 BTC
Tx0 A → B 100 BTC
Tx1 B → H 200 BTC

- 블록체인은 분산원장 기술 (DTL; Distributed Ledger Technology)
  - 참여자들 모두 블록체인의 복사본을 가짐
  - 참여자들은 각자 트랜잭션, 블록, 블록체인을 검증

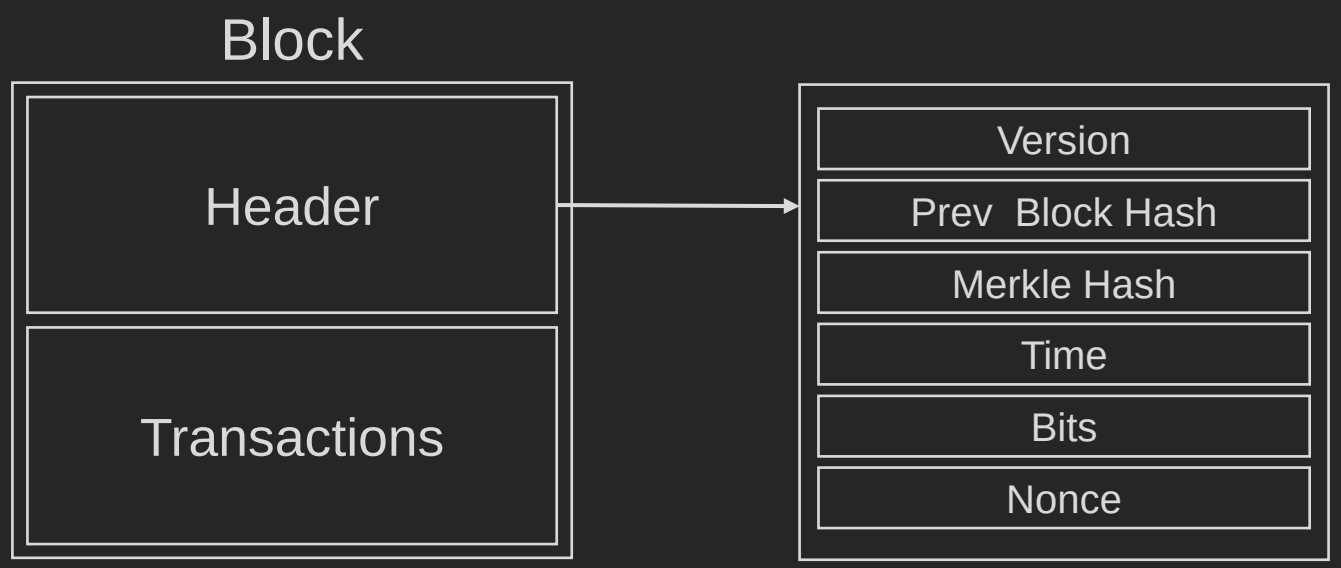


- 블록 높이(height), 깊이(depth), 컨펌(confirmation)

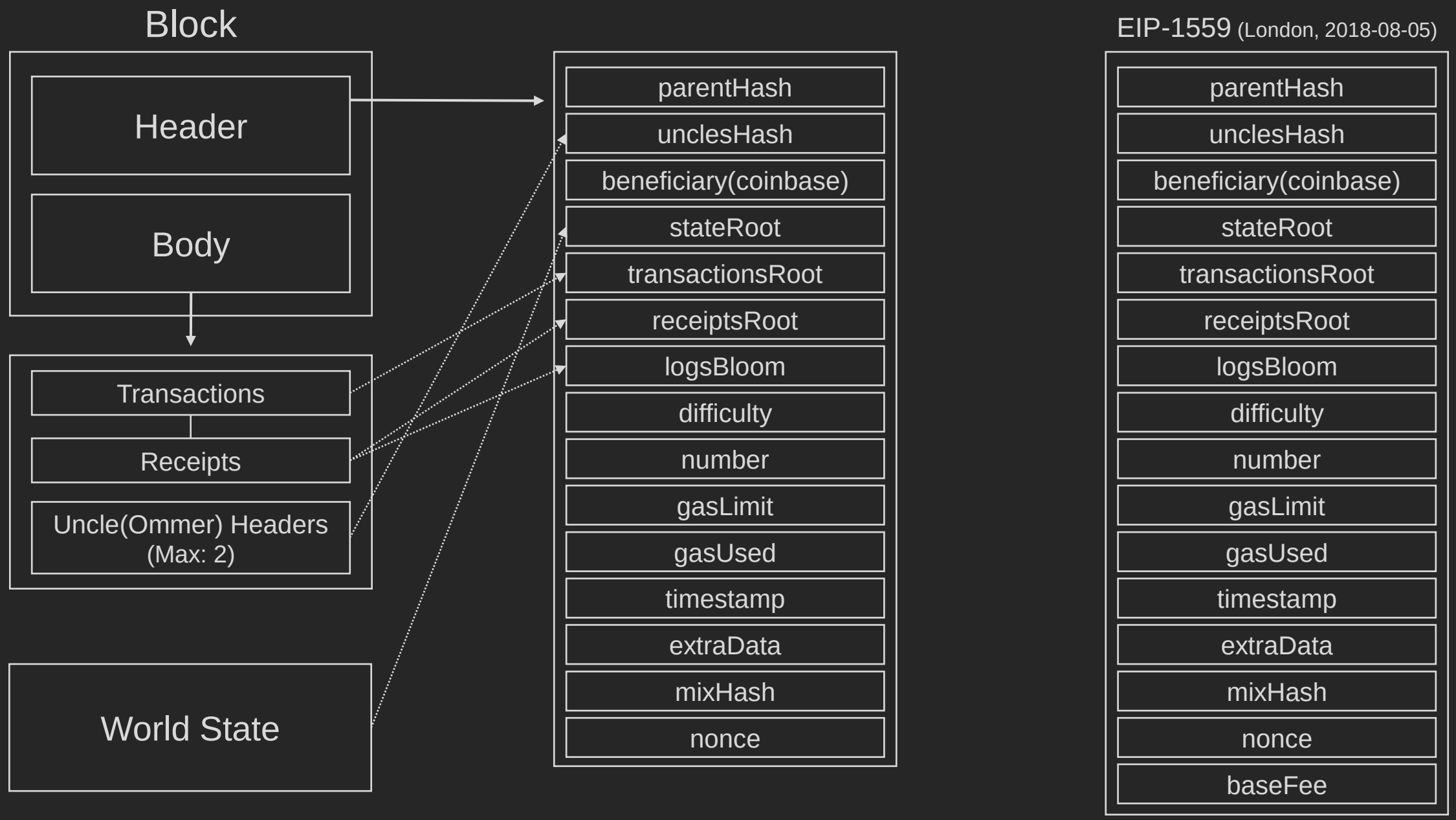


	높이 (불변)	깊이 (가변) = 컨펌
latest Block 2	2	1
Block 1	1	2
Block 0	0	3

	높이	깊이
latest Block 3	3	1
Block 2	2	2
Block 1	1	3
Block 0	0	4



항목		Field Size	Data Type	설명
Version	버전	4	int32_t	블록의 버전, 현재 이 블록 헤더를 만든 비트코인 프로그램의 버전 현재 버전: 4 (1,2,3,4)
Prev Block Hash	이전 블록 해시	32	char[32]	이전 블록 해시 정보 이전 블록헤더를 SHA256 해시로 두 번 해시한 값
Merkle Hash	머클 해시	32	char[32]	Transactions Trie Root Node Key (= Transactions Trie Merkle Root)
Time	시간	4	uint32_t	블록의 대략적인 생성 시간, Timestamp 초 단위
Bits	비츠	4	uint32_t	작업 난이도, 논스 값 계산이 어려운 정도
Nonce	논스	4	uint32_t	임의의 수 (PoW[Proof of Work,작업 증명] 퍼즐 정답)


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

블록체인 구조

비잔티움 장군 문제

P2P 네트워크

전자서명

합의 알고리즘

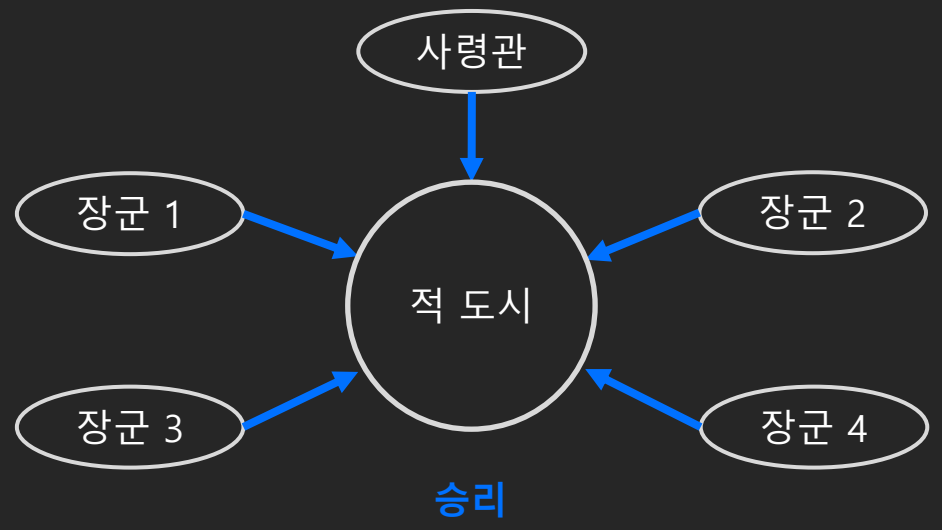
지갑

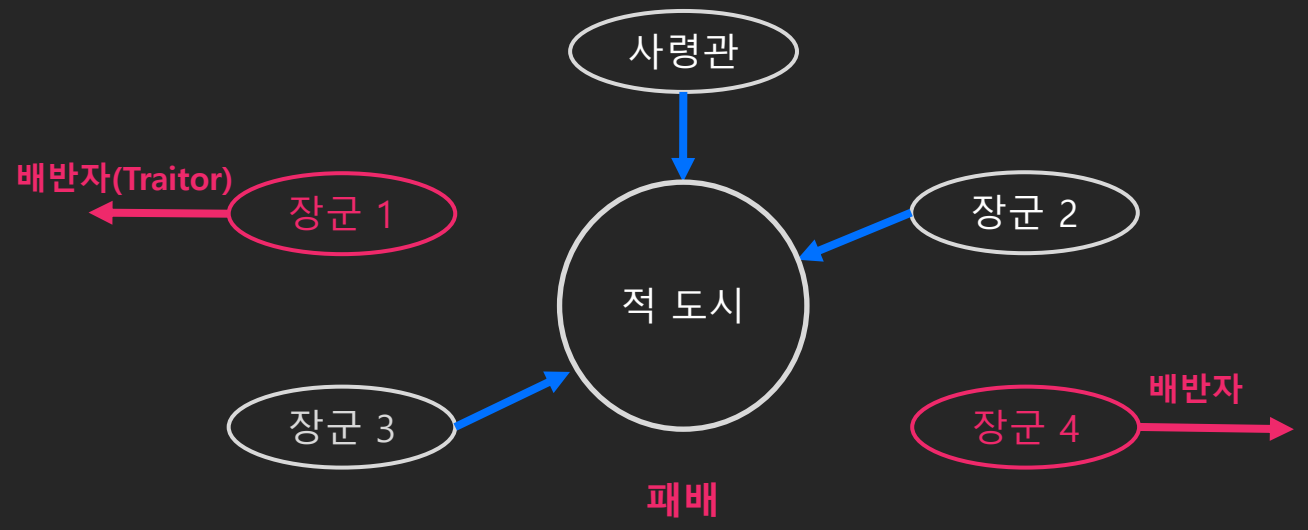
비잔티움 장군 문제
(The Byzantine Generals Problem)

- 신뢰할 수 없는 분산 네트워크 환경에서 악의적인 노드 존재 시에도
  안전하게 메시지 전송 및 데이터 상태 공유 방법은 무엇인가?
- 비잔티움 장군 문제 (The Byzantine Generals Problem) – Paper
  - 1982년 레슬리 램포트(Leslie Lamport), 쇼스탁(Robert Shostak), 피스(Marshall Pease)가 공저한 논문
  - 적군의 도시를 공격하려는 비잔티움 제국군의 여러 부대가 지리적으로 떨어진 상태에서 각 부대의 장군들이 (중간에 잡힐지도 모르는) 전령을 통해 교신하면서 공격 계획을 함께 세우는 상황을 가정 – 위키백과
  - 배신자가 존재할 경우에도 합의에 성공하려면 배신자를 포함하여 총 몇 명의 장군이 네트워크에 있어야 할 지를 정하는 것
  - 한 체계 내에서 연결된 다양한 시스템들이 그 중 일부가 에러 코드, 혹은 잘못된 명령어 전달하는 상황에서 어떻게 시스템들의 기능을 정상으로 유지시키고, 체계를 정상 작동 시킬수 있는지 고민하는 일종의 사고 실험 - 나무위키
- Bitcoin mailing list #014847 – James A. Donald, 2008-11-13 06:16:31
  - It is not sufficient that everyone knows X.
    We also need everyone to know that everyone knows X,
    and that everyone knows that everyone knows that everyone knows X
    - which, as in the Byzantine Generals problem, is the classic hard problem of distributed data processing.
  - 모두가 X를 안다는 사실로는 충분하지 않습니다.
    모두가 X를 안다는 사실도 모두가 알아야 하며,
    또 모두가 X를 안다는 사실을 우리가 모두 다 안다는 사실도 전원이 알아야 한다.
    이는 비잔틴 장군 문제처럼, 분산 데이터 시스템에서 오래된 난제이다.

비잔티움 장군 문제 (The Byzantine Generals Problem) - Paper , 한글 분석

- Base Conditions
  - 적은 모두 동시에 공격을 해야 이길 수 있는 상대
  - 적진을 둘러싼 장군들은 합의를 통해 공격 시간 및 공격 여부 결정
  - 장군들은 전령을 통해서만 서로 연락 가능. 한 곳에 모이기 어려움.
  - 장군들 중 한 명의 사령관이 "공격“, "후퇴"를 결정하여 메시지를 모든 장군에게 전달
  - 장군들은 사령관으로부터 받은 메시지를 나머지 장군들에게 전달 (장군들은 서로 연결되어 있다.)
  - 장군들은 자신이 받은 메시지를 종합하여 "공격“, "후퇴"를 결정하여 명령 이행 (동기화)
- Risk Factors : 합의를 방해하는 요소들
  - 전령은 메시지 전달을 실패할 수 있다. (적군에게 포획,살해 또는 기타 사고 등) ➡ 메시지를 보낸 장군은 메시지 도달 여부를 알 수 없다.
  - 장군 및 전령들 중에 배신자가 섞여 있어 메시지를 왜곡할 수 있다. (사령관도 배신자가 될 수 있다.)



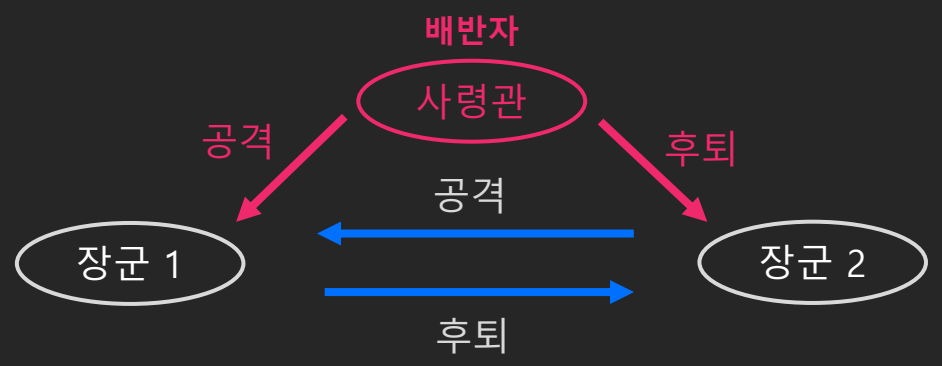


전쟁 승리 조건은?

- 사령관이 "공격" 명령 시, 메시지를 받은 장군들이 모두 "공격" 해야 전쟁에서 승리.
- 배신자의 방해가 있더라도 모두 같은 결정을 할 수 있어야 함
- 아래의 두 가지 추가 조건 고려 (IC, Interactive Consistency)
  - IC1: 배신자가 아닌 모든 장군들은 같은 명령을 이행
  - IC2: 만약 사령관이 배신자가 아니라면, 배신자를 제외한 모든 장군들은 사령관의 명령을 이행
- BGP 발생 조건 : 합의 대상이 3명 이상
- V_i : i 번째 장군이 받은 메시지의 집합. 배신자 외 장군들은 메시지 내용으로 "공격"과 "후퇴" 중 하나를 이행. (과반수 우선 선택 기준)

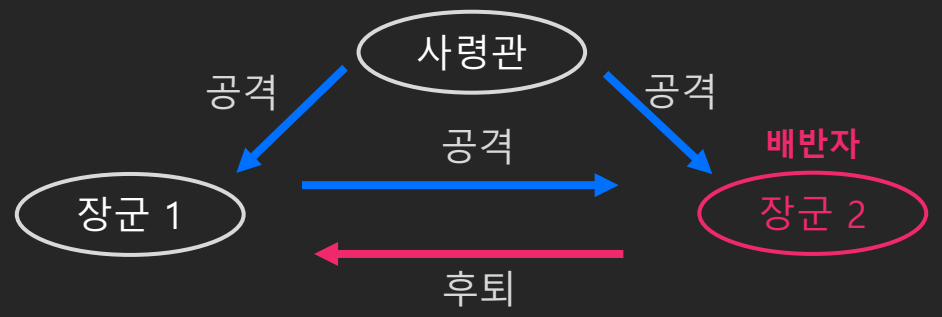
1. 사령관이 배신자일 경우 : **사령관 메시지 Bold**

- V_1 = {**"공격"**, "후퇴"} : 장군 1은 "공격" 으로 ➡ 장군 1만 공격
- V_2 = {**"후퇴"**, "공격"} : 장군 2는 "후퇴" 로 결정



2. 장군 2가 배신자일 경우 : **사령관 메시지 Bold**

- V_1 = {**"공격"**, "후퇴"} : 장군 1은 "공격" 으로 결정
  ➡ 사령관과 장군 1만 공격



- 부정한 노드가 f개 일 때, 전체 노드 수는 3f+1개, 확정에 2f + 1개 노드 필요

f (부정한 노드 수)	3f + 1 (최소 노드 수)	2f (회신에 필요한 수신 명령 수)	f + 1 (확정에 필요한 노드 수)
1	4	2	2
2	7	4	3
3	10	6	4
4	13	8	5
5	16	10	6

P2P 네트워크

전자서명

합의 알고리즘

스마트 컨트랙트

5. 네트워크

네트워크를 실행하는 단계는 다음과 같다:

1) 새로운 거래가 모든 노드에 브로드캐스트된다.
2) 각 노드가 새로운 거래를 블록에 수집한다.
3) 각 노드가 그 블록에 맞는 난도의 작업증명을 찾아 나선다.
4) 노드가 작업증명을 찾은 시점에, 거기서 모든 노드로 그 블록을 브로드캐스트한다.
5) 노드는 모든 거래가 유효하며 아직 지불되지 않았다는 조건에 맞을 경우에만 그 블록을 승인한다.
6) 노드는 블록 승인을 표현하기 위해 먼젓번 해시로 승인된 블록의 해시를 사용해 사슬 안에 다음 블록을 생성한다.

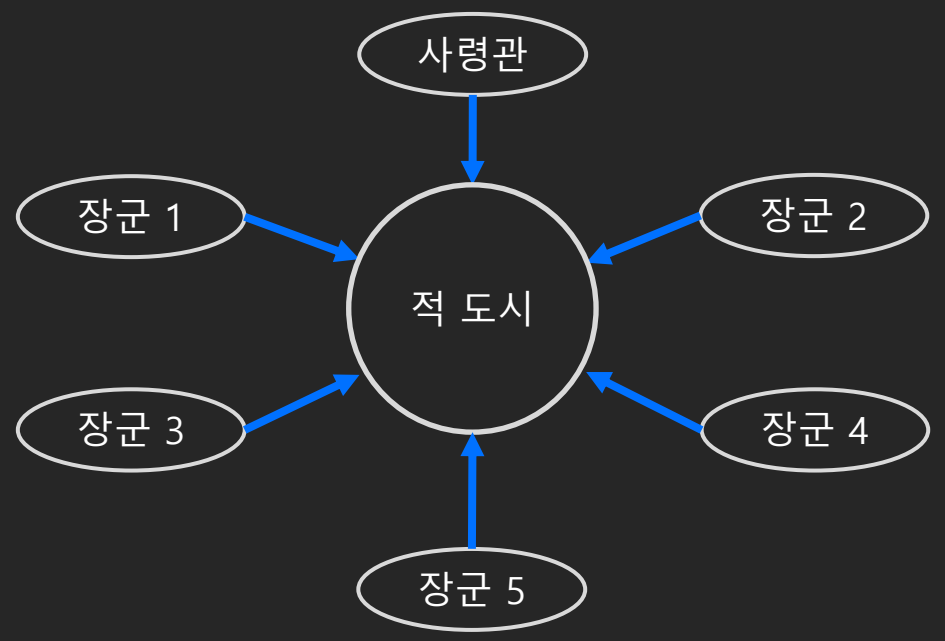
5. Network

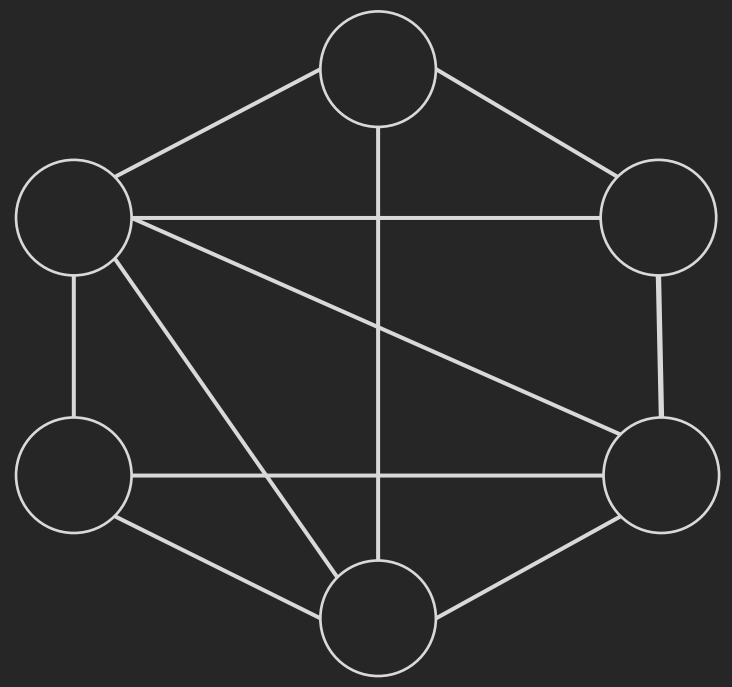
The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

P2P(Peer-to-Peer) Network

- 인터넷에 연결된 다수의 컴퓨터(노드(Node))가 중개 기관을 거치지 않고 직접 데이터를 주고 받는 것
- Peer : 동료, 대등한 자, 동등한 자
- 블록체인 Node : 블록체인 클라이언트를 실행하는 컴퓨터

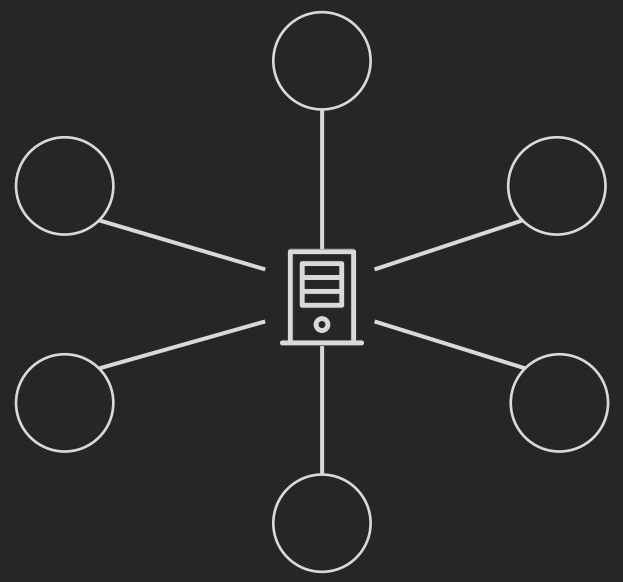




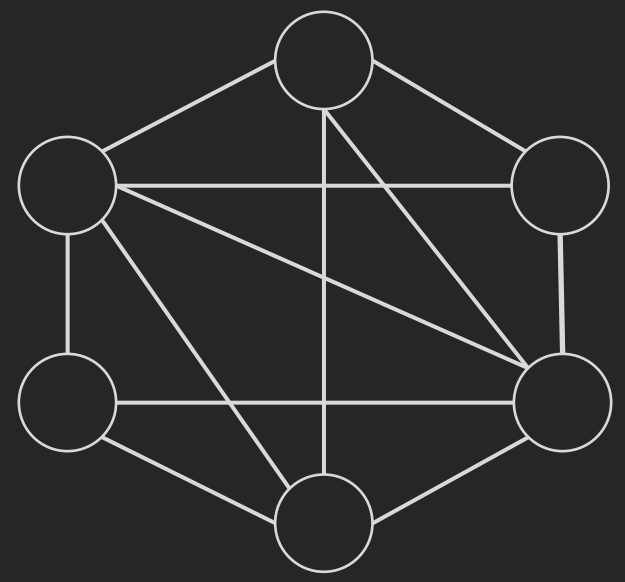
P2P Network

클라이언트-서버 와 P2P Network

	클라이언트-서버	P2P Network
특징	중앙집중, 상이한 역할과 지위	분산 노드, 동등한 역할과 지위
장점	시스템 설계 및 유지보수 쉬움 데이터 통일성	노드의 자원과 네트워크 회선을 이용해 부하 분산 노드가 증가해도 시스템 유지할 수 있는 확장성
단점	SPOF(Single Point Of Failure) 문제 다수 클라이언트 접근시 고사양 서버 및 큰 대역폭 필요 관리기관의 도덕적 헤이	구축 및 다른 노드를 찾는 기능 설계에 많은 노력 필요 네트워크 전송 시간으로 인한 동기화 문제 및 성능 저하 발생 쉬움



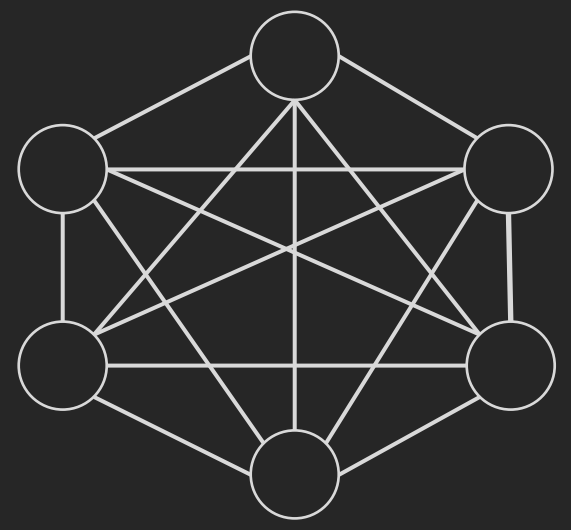
클라이언-서버



P2P Network

퓨어 P2P (Pure P2P)

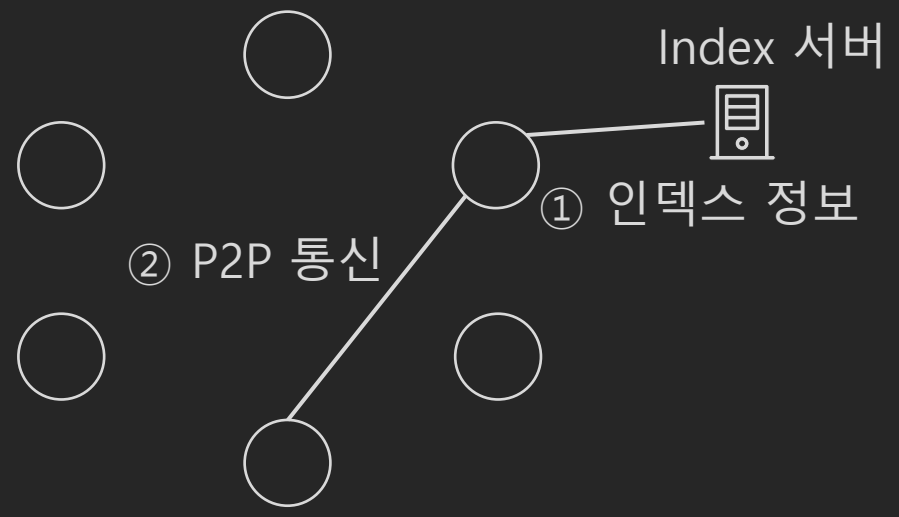
- 모든 노드가 동등한 입장에서 네트워크에 연결
- 장점 : 새로운 노드 추가 등 확장성 높음
- 단점: 시스템 전반을 설계하고 관리하기 어려움
- 단점 : 다른 노드들을 발견 및 탐색할 수 있는 알고리즘 구현 필요
- 비트코인, 이더리움



Pure P2p

하이브리드 P2P (Hybrid P2P)

- 노드 탐색을 위해 인덱스 서버를 활용
- 각 노드가 보유한 데이터의 정보가 인덱스 서버에 기록
  노드들의 검색 및 발견 등을 인덱스 서버를 통함
- 데이터는 노드들간 직접 주고 받음
- 장점: 클라이언트-서버 처럼 설계와 관리가 용이
- 단점: 확장성이 떨어짐

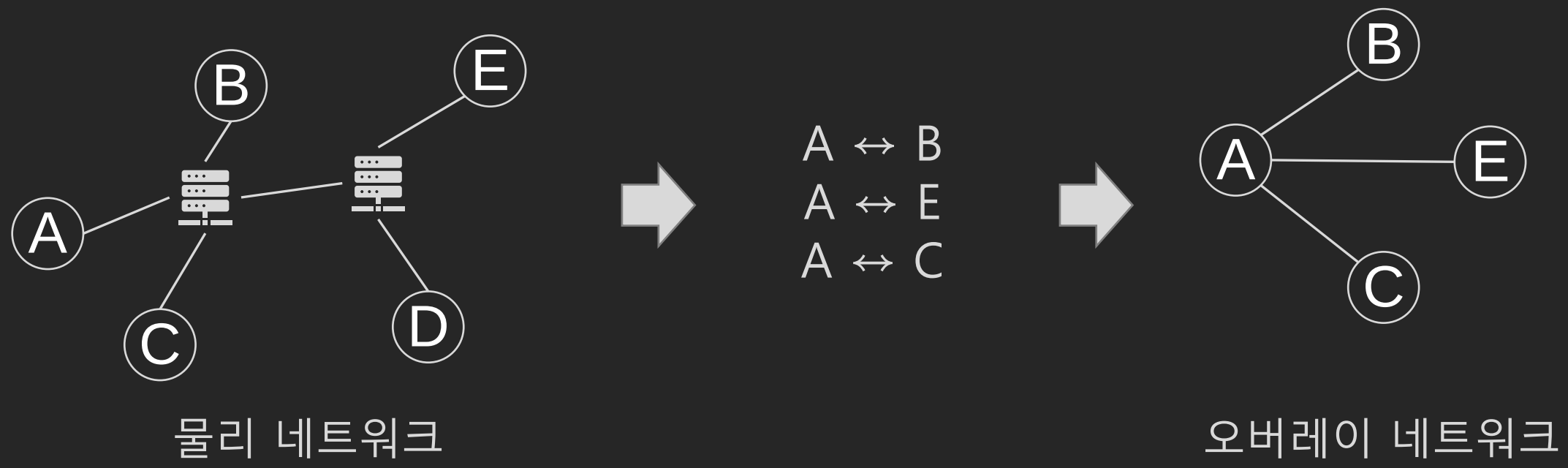


하이브리드 P2P

퓨어 P2P은 중앙서버 없이 노드 탐색 실현을 위해 프로그램 수준의 네트워크 구축 필요

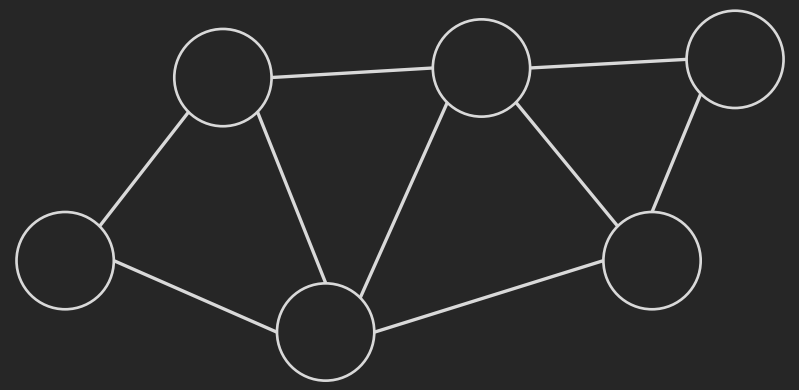
오버레이 네트워크(Overlay Network)

- 물리 네트워크 위에 성립되는 가상의 네트워크
- 노드와 노드간 경로로 구성돼 실제 네트워크와 다른 네트워크 토폴로지 구성
- 토폴로지와 노드의 검색방법 따른 분류
  - 비구조화 오버레이
  - 구조화 오버레이

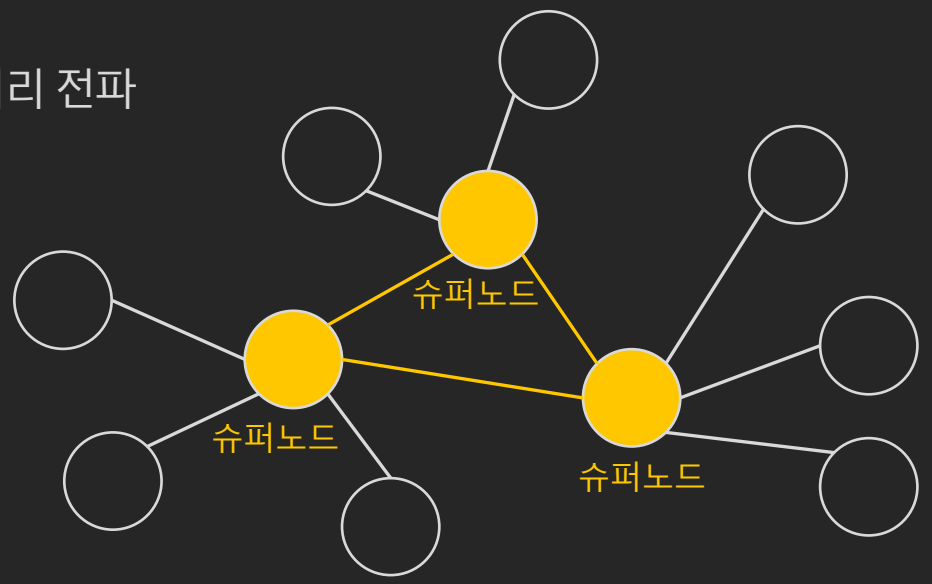


비구조화 오버레이 (비트코인, 이더리움)

- 각 노드가 인접 노드를 선택 시 제약 없음
- 노드 탐색은 메시지를 인접 노드에 차례로 전파해 확산시키는 방법을 사용
- 장점 : 메시지에 요청 데이터의 메타 정보를 포함, 메타 정보에 맞는 데이터를 가진 노드를 탐색하는 등 유연한 탐색 가능
- 단점 : 목적 노드에 메시지 전달 보장 불가, 노드 수 증가 시 네트워크에 메시지가 많아지는 확장성 문제 발생
- 슈퍼 노드 : 메시지 전파 문제 해결을 위한 개념 (일부 시스템)
  - 일부 노드는 일반 노드의 상위 노드인 슈퍼 노드로 구분
  - 슈퍼 노드는 다수의 일반 노드를 밑에 두고 메시지를 슈퍼노드 끼리 전파



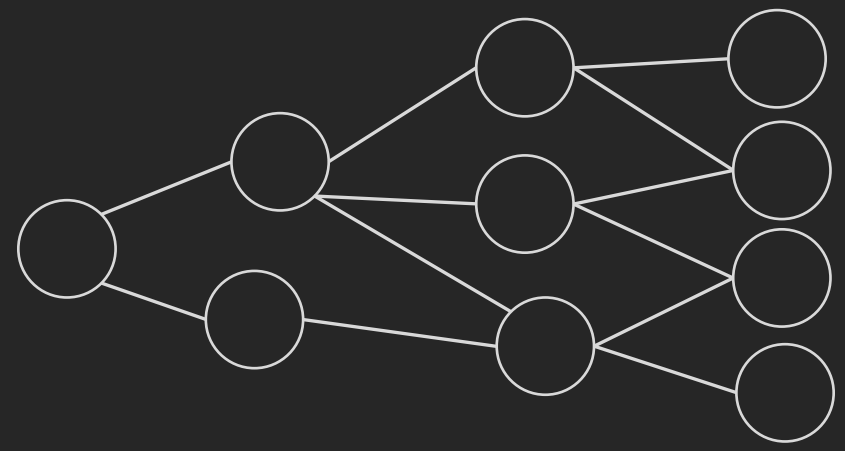
비구조화 오버레이



비구조화 오버레이 /w 슈퍼 노드

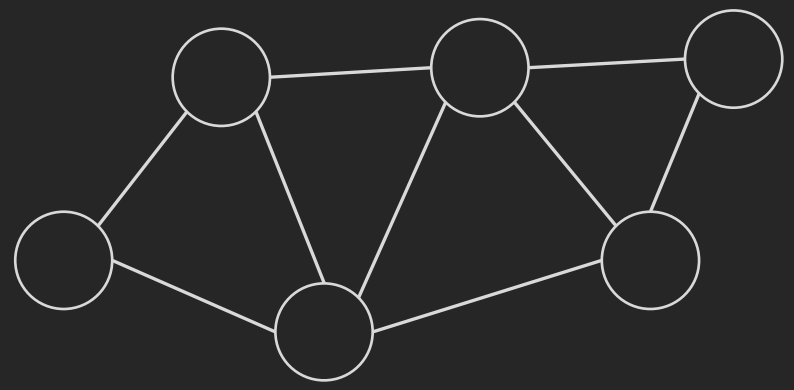
구조화 오버레이

- 노드 별로 연결할 때 상대가 미리 결정, 엄격하게 설계된 네트워크 토폴로지
- 각 노드에는 ID가 할당되고 ID에 따라 연결할 상대가 결정
- 링크형 또는 트리형 구조의 오버레이 네트워크 구축
- 데이터: 노드와 동일한 ID가 할당. 각 데이터는 자신의 ID와 가장 가까운 ID를 가진 노드에 저장
- 메시지 전송 : ID를 사용한 경로 선택(라우팅 - Routing). 메시지의 대상 ID에 가까운 인접 노드 선택을 반복
- 장점 : 메시지 도착 가능성 및 확장성 높음
  - 목적지 노드가 네트워크에 존재하고 있다면 반드시 메시지 전달
  - 엄격한 네트워크 토폴로지 설계로 노드 수 증가에도 메시지의 전송횟수는 크게 늘어나지 않음
- 단점 : ID 등을 기반으로한 메시지 전송이기에 유연한 탐색 불가. 대부분 데이터 ID를 사용한 완전 일치 검색만 가능



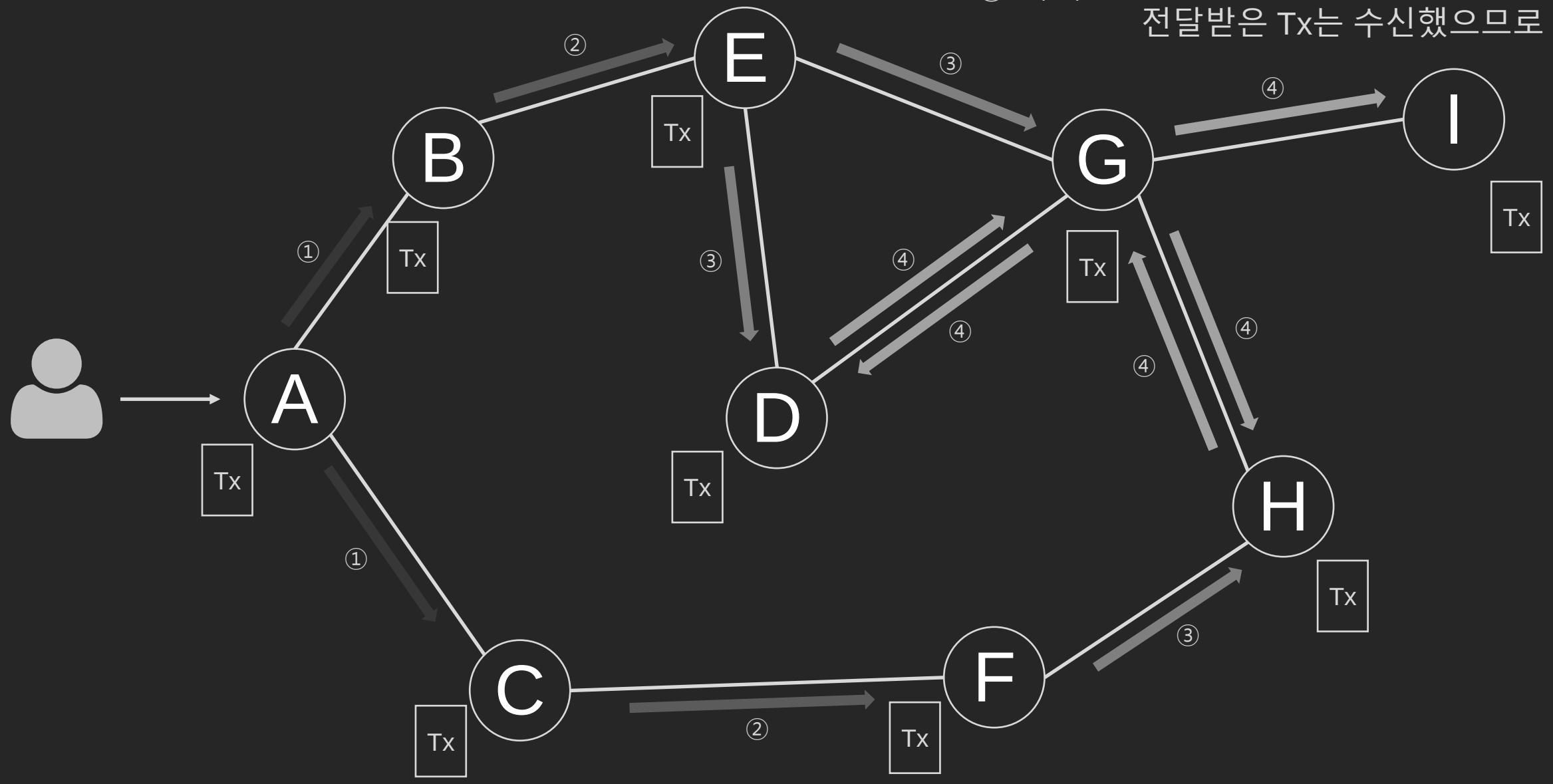
비트코인, 이더리움 P2P Network

	P2P 네트워크	오버레이 네트워크
BitCoin	퓨어 P2P	비구조화 오버레이
이더리움	퓨어 P2P	비구조화 오버레이



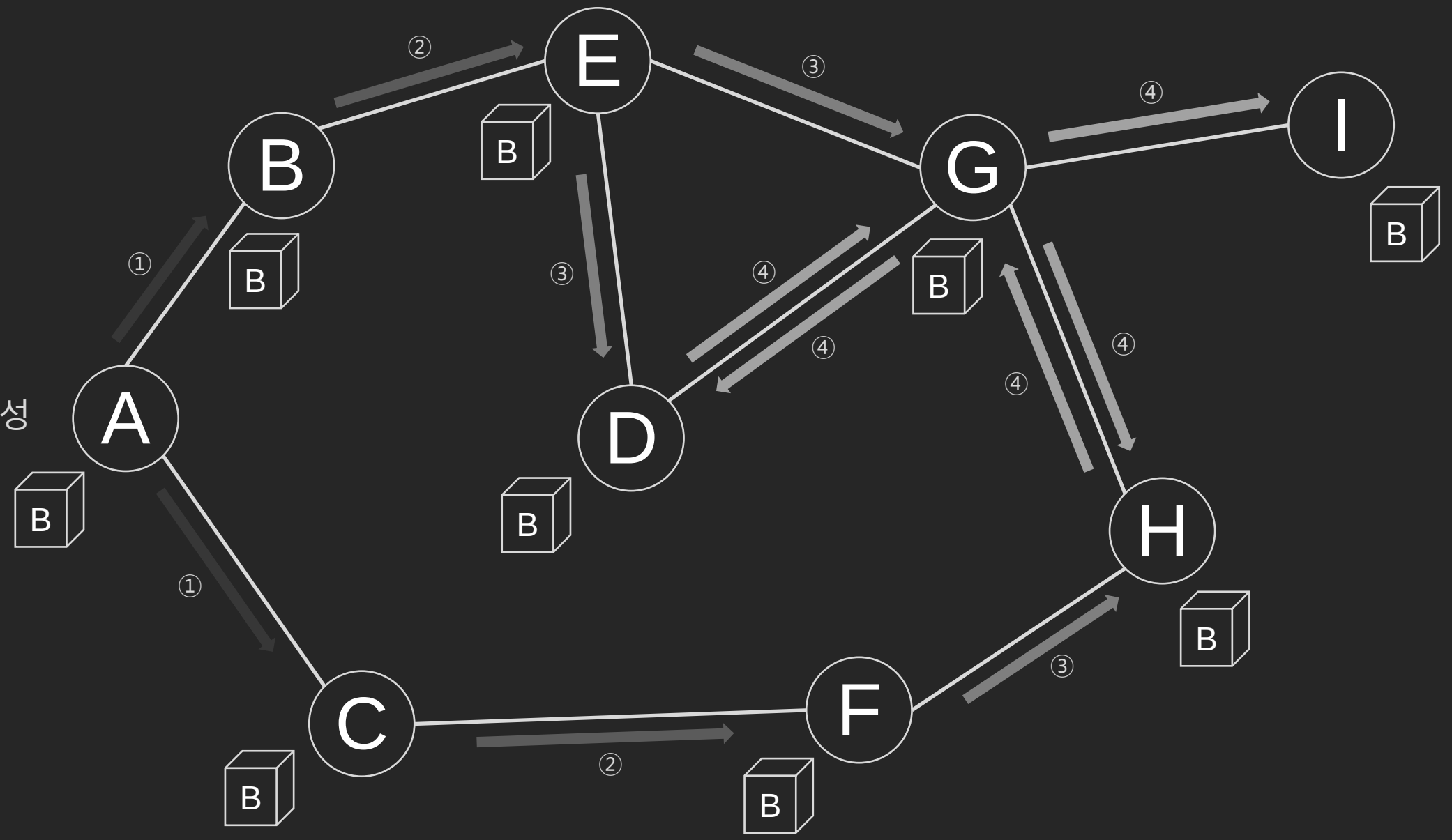
노드는 유효성 검사를 통과한 Tx를 Peer에 전파

④ D,G,H는 각각의 Tx 수신 사실을 알 수 없음
전달받은 Tx는 수신했으므로 무시



노드는 유효성 검사를 통과한 Block를 Peer에 전파

A가 블록 생성



전자 서명
(Digital Signature)

비대칭(공개키) 암호와 해시의 응용

인증(authentication)

개인키의 소유자에 의해 생성되었음을 증명

부인 방지(non-repudiation)

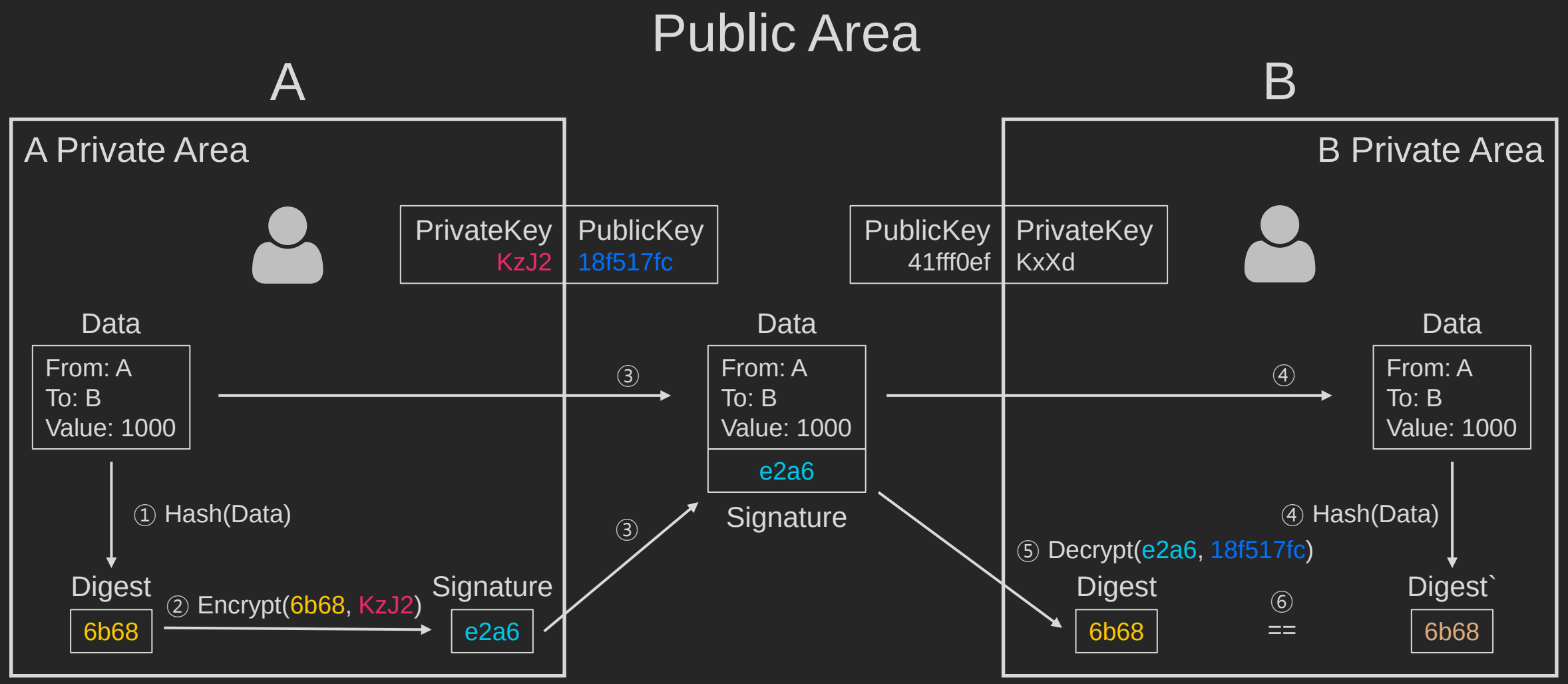
보낸 사람이 메시지를 보내지 않았음을 부인할 수 없음.

무결성(integrity)

위조·변조 탐지 근거 제공

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

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

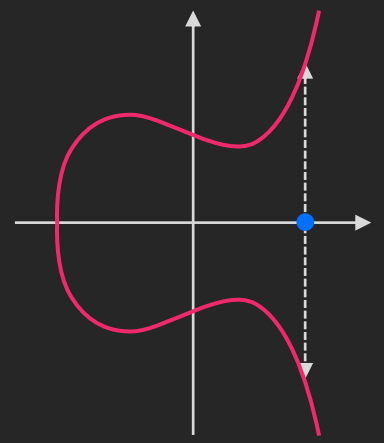
- r, s : 서명 값
- K : 서명자 공개키
- m : 트랜잭션 데이터
- G : 타원 곡선상 생성자 점
- n : 타원 곡선의 소수 차수

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

1. x 좌표인 r 값의 2개의 타원 곡선 점 R과 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_2는 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)

- 비트코인
  - m = SHA-256(SHA-256(Tx))
  - (r, s) = Fsig(m, privateKey)
  - Tx에 r, s, Public Key 포함
  - r, s, Public Key를 이용하여 검증
- 이더리움
  - m = keccak-256(Tx)
  - (r, s, yParity) = F_{sig}(m, privateKey)
    - yParity = 일시적 공개키 y 좌표의 짝수(0), 홀수(1) 여부
  - Tx에 r, s, yParirty 포함
  - 공개키 복구
  - r, s, 복구한 공개키를 이용하여 검증

합의 알고리즘(Consensus Algorithm)

분산 환경에서 다른 컴퓨터들간 같은 데이터에 동의하는 메커니즘

블록체인 합의 알고리즘

비신뢰 분산 환경에서 신뢰가능한 블록?

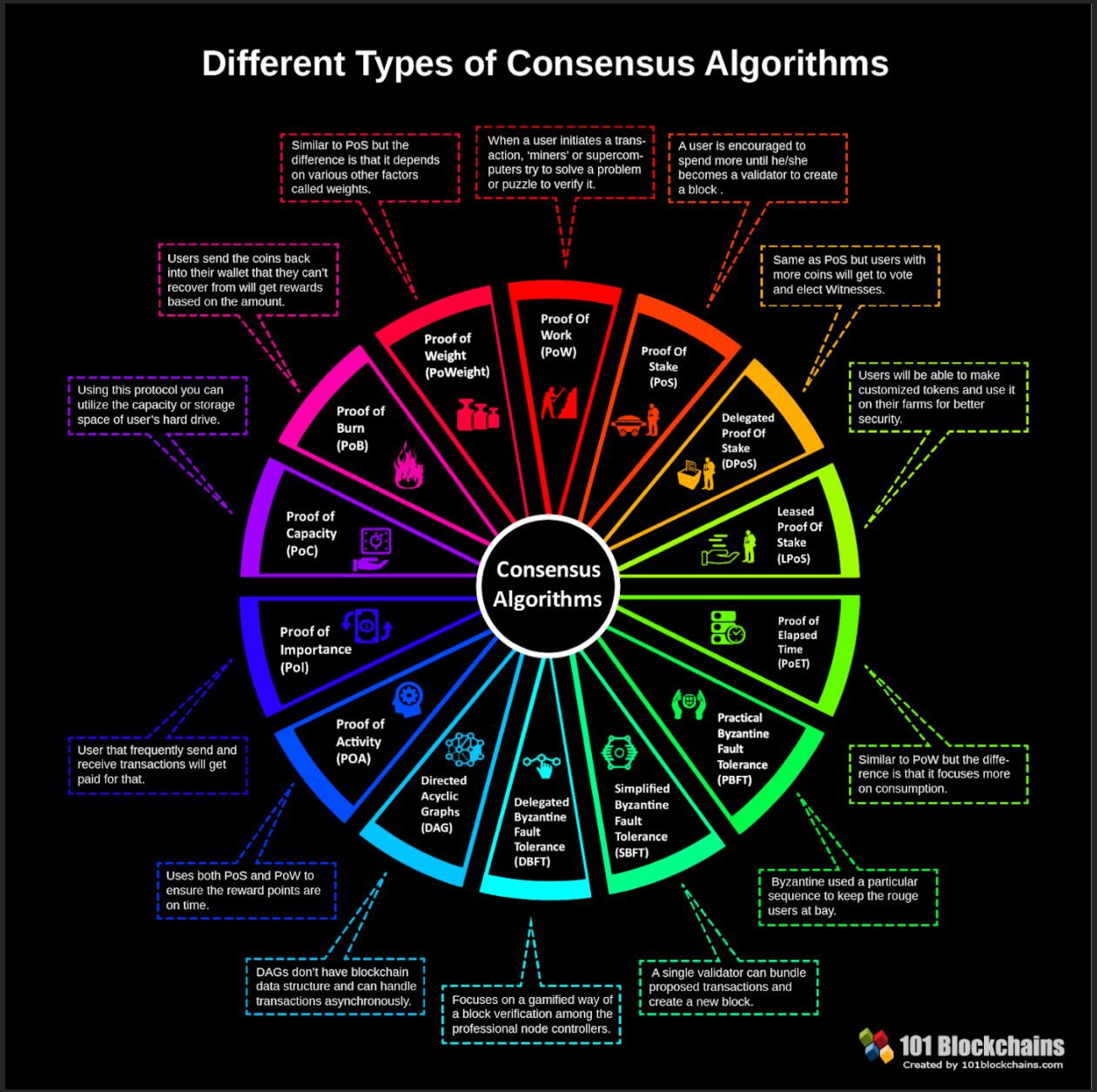
동일 Height 블록이 2개 이상인 경우?

⬇

누가 블록을 만들 것인가?

어떤 블록을 선택할 것인가?

합의 알고리즘 (Consensus Algorithm)



https://101blockchains.com/wp-content/uploads/2018/08/Different_Consensus_Algorithms.png

PoW (Proof of Work) → Bitcoin, Ethereum

PoS (Proof of State) → Ethereum 2.0

PBFT → Klaytn, Line

Raft → Hyperledger Fabric

4. 작업증명

개인 대 개인 기반으로 분산 타임스탬프 서버를 구현하기 위해 우리는 신문이나 유즈넷 게시물 대신 애덤 백의 해시캐시(Adam Back's Hashcash)와 유사한 작업증명 시스템[6]을 사용할 필요가 있다. 작업증명은 SHA-256 같은걸로 해시 연산을 거친 결과가 0 비트(zero bits) 여러 개로 시작할, 특정 값을 찾는 작업을 수반한다. 여기에 평균적으로 필요한 연산 작업은 결과값에 필요한 0 비트 개수에 따라 지수적으로 달라지며 해시 연산을 한 번 실행하는 걸로 검증될 수 있다.

타임스탬프 네트워크용으로, 우리는 블록의 해시에 필요한 0 비트를 주는 값이 발견될 때까지 블록 안에 임시값을 증분(incrementing a nonce)하는 것으로 작업증명을 구현했다. CPU 동작(CPU effort)이 한 번 작업증명을 충족하는 데 동원됐다면, 그 블록은 해당 작업을 재수행하지 않고는 변경될 수 없다. 그 뒤에 이어서 나중에 생성된 블록이 연결되는만큼, 그 블록을 변경하는 재수행 작업은 그 뒤 모든 블록을 생성하는 연산까지 포함한다.

작업증명은 다수결(majority decision making)의 대표성 문제도 해결한다. IP 주소당 1 표에 기반한 다수 조건이라 면 누구든지 많은 IP 를 할당할 수 있는 이에 의해 장악될(subverted) 수 있다. 작업증명은 기본적으로 CPU 당 1 표다. 다수의사는 최다 작업증명 동작이 투입된 가장 긴 사슬로 대표된다. 만일 다수 CPU 파워가 정직한 노드에 의해 통제된다면, 가장 정직한 사슬이 가장 빠르게 늘어나 다른 경쟁 사슬을 압도(outpace)할 것이다. 과거 블록을 변경하려면 공격자는 그 블록과 그 뒤를 잇는 모든 블록의 작업증명을 재수행해야 하고 그러면서 가장 정직한 노드들의 작업을 따라잡아 앞질러야 한다. 뒤에서 우리는 이어지는 블록이 추가될수록 더 느린 공격자가 따라잡을 확률이 지수적으로 감소함을 보이겠다.

시간이 지날수록 노드를 구동하는 하드웨어의 속도 증가와 변화하는 관여도(interest)를 보상하기 위해, 작업증명 난도(difficulty)는 시간당 평균 블록 수에 따른 평균 목표치를 조정해 결정된다. 그것들(블록)이 너무 빨리 생성되면 난도가 증가한다.

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proofof-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.

The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

PoW(Proof of Work, 작업증명) : 누가 블록을 만들것인가?

- 퍼즐 문제를 빨리 푼 노드가 블록 생성
- 퍼즐 : Block Hash값이 일정 값 이하가 되는 Nonce를 찾는 문제
- 논스 (Block Nonce) : Block Hash 값을 변경시키기 위한 임의의 값
- 채굴(Mining) : 퍼즐을 풀어 블록을 생성하고 전파하는 것
- 블록 생성 시간 & 난이도 조정 (Target Block Time & Difficulty Target)
  - 비트코인 : 블록당 10분 타켓, 2016 블록 (약 2주) 마다 난이도 조정
  - 이더리움 : 블록당 12~14초 타켓, 블록마다 난이도 조정
- 보상(Reward) : 블록생성(채굴)에 대한 보상 및 트랜잭션 수수료
  - 비트코인 : 블록생성 보상과 트랜잭션 수수료
  - 이더리움 : 블록생성 보상, 삼촌 블록 포함 보상 및 트랜잭션 수수료

비트코인 블록 Reward

- 블록생성 보상 및 트랜잭션 수수료
- Block의 첫번째 Transaction은 Miner에게 Bitcoin Reward 지급 ➞ 화폐 유통
  - Bitcoin 생성 트랜잭션 ➞ Coinbase Transaction ➞ 100 Block 동안 사용 불가
- 50 BTC 시작, 21만 블록 (약 4년)마다 절반으로 감소, 총 21,000,000 BTC
- 트랜잭션 수수료는 byte 당 지불.

이더리움 블록 Reward

- 블록 생성 보상, 삼촌 블록 포함 보상 및 가스 수수료
- Block의 Beneficiary(Coinbase)를 Miner로 지정하고, Miner의 상태를 수정하여 지급
- 5 ETH (Frontier) ➞ 3 ETH(Byzantium, 2017-10-16) ➞ 2 ETH (Constantinople, 2019-02-28)
- 삼촌 블록 포함 시, 블록 당 블록 보상의 3.125% (최대 2개)
- 가스 수수료 (가스 : 트랜잭션 처리 비용 측정 단위. ADD: 3, DIV: 5)
  - Before EIP-1559 : Gas Price * Gas Used
  - After EIP-1559 : (Gas Price - Base Fee) * Gas Used

1 ~ 8

8이하를 뽑을 확률 ➞ 100%

4이하를 뽑을 확률 ➞ 50%

2이하를 뽑을 확률 ➞ 25%

1이하를 뽑을 확률 ➞ 12.5%

대상 숫자 감소

↓

난이도 증가

Hash 결과의 처음 0길이가 증가하는 숫자 찾기

0 길이	숫자	Hash
1	7	0x0e9b80a0e46a4624d5f9d3ae02c9a3180f263d332bc109e61456d155de29b5c4
2	604	0x00bb9f5f6617cc83190bfcae6431184e58d887c1062f1db3f65c77f5a1ae4e0a
4	8181	0x0000e8d780492c1b9cdaa17edb6144a1ef37f6332df3a22e700c8c89ef48f012
5	311931	0x00000c8e3778bc57ec33b2cc8116436db7ba63574d8bfac4bdabbdba021911cb
6	16079576	0x000000 3d9d700667b0ef527c6dfdd1301db42e935e9ed5c6504ad12ee28a1af5

Hash 결과는 랜덤 숫자 뽑기

숫자 찾기 프로그램, hash = SHA-256(SHA-256(tx))

```
import util from 'ethereumjs-util';

(async () => {
  let target = "0";
  for (let nonce = 0; nonce < 100000000; nonce++) {
    let tx = `{"from":"A","to":"B","value":1000,"nonce":${nonce}}`;
    const hash = util.sha256(util.sha256(Buffer.from(tx))).toString('hex');
    if (hash.startsWith(target)) {
      let zeroLength = 0;
      for (let j = 0; j< hash.length; j++) {
        if (!(hash.charAt(j) === '0')) { break; }
        zeroLength++;
      }
      console.log(`number: ${nonce}, 0 length: ${zeroLength}, hash: 0x${hash}`);
      target = "";
      for (let j = 0; j <= zeroLength; j++) { target += "0"; }
    }
  }
})();
```

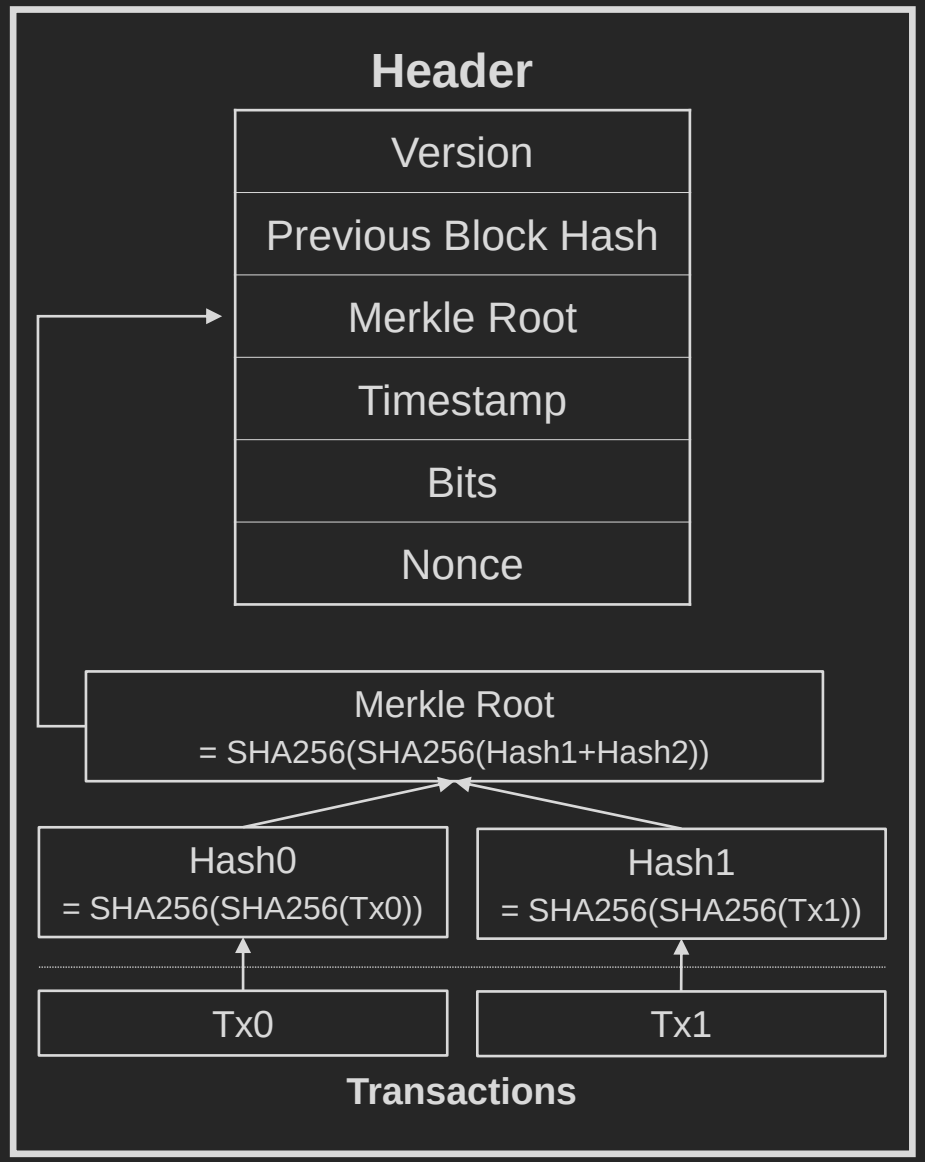
비트코인 PoW Puzzle

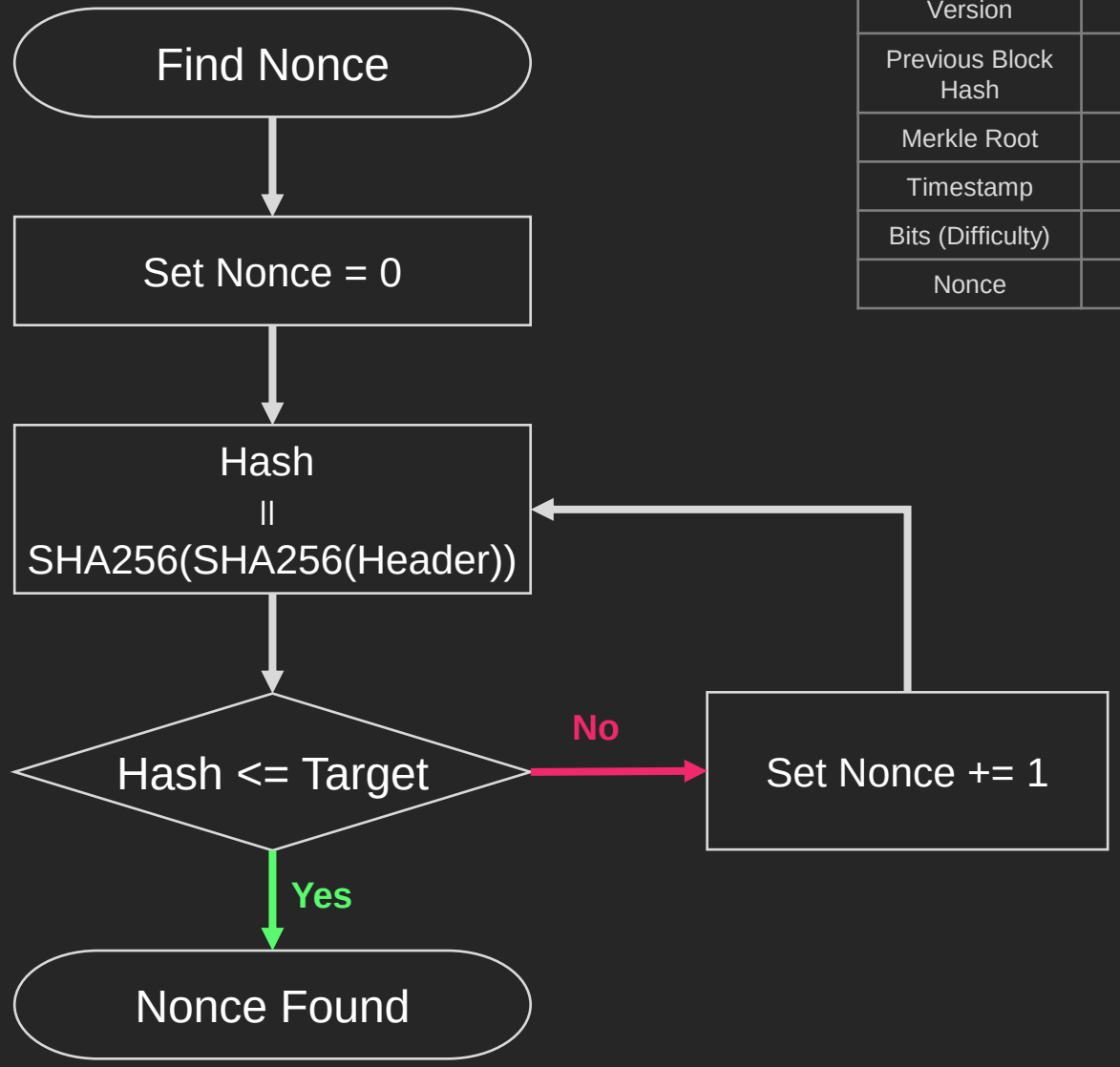
숫자 범위 : $0 \sim 2^{256} - 1$

숫자 뽑기 : Block Hash

Block

Difficulty Level = Difficulty Target / Current Target





Header

Field	Size (bits)	Type
Version	32	int32_t
Previous Block Hash	256	char[32]
Merkle Root	256	char[32]
Timestamp	32	uin32_t
Bits (Difficulty)	32	uin32_t
Nonce	32	uin32_t

Difficulty

- 주어진 Target 보다 작은 hash 를 찾는 것이 어려운 정도
- $difficulty = \frac{difficulty\ 1\ target}{current\ target} = \frac{maximum\ target}{current\ target}, \quad current\ target = \frac{maximum\ target}{difficulty}$
- 조정 주기 (Epoch) : 2016블록 (약 2주)
- 난이도 조정 비율 = 2주 / 2016 블록 채굴 소요 시간
  - 평균 블록 생성시간이 10분 미만이면 난이도 감소, 10분 초과면 난이도 증가
  - 2주 = 2016 블록 * 10분 * 60초 = 1,209,600 초
- 다음 난이도 = 현재 난이도 * 난이도 조정 비율
  - 최소값 = 1, 최대값: 제한 없음
- 난이도 보정 계수
  - 최대 4배(+300%), 최소 1/4 배(-75%) ➞ 급격한 변화 방지
- Target 값은 압축되어 Block Header의 Bits로 저장 (Bits ➞ Current Target ➞ Difficulty)

Block Header - Bits 로 부터 Target 및 Difficulty 계산

$$difficulty = \frac{difficulty\ 1\ target}{current\ target}$$

- Bits : 0x1b0404cb

Target : 0x0404cb * 2**(8*(0x1b - 3))
= 0x0000000000404CB000000000000000000000000000000000000000000000000

- Difficulty 1 (maximum target) : 0x1d00ffff
  = 0x00ffff * 2**(8*(0x1d - 3)) = 0x00000000FFFF0000000000000000000000000000000000000000000000000000

- Difficulty (0x1b0404cb) :
  0x00000000FFFF0000000000000000000000000000000000000000000000000000 /
  0x0000000000404CB000000000000000000000000000000000000000000000000
  = 16307.420938523983

Difficulty

- 주어진 Target 보다 작은 hash 를 찾는 것이 어려운 정도
- $difficulty = \frac{difficulty\ 1\ target}{current\ target} = \frac{maximum\ target}{current\ target}, \quad current\ target = \frac{maximum\ target}{difficulty}$
- Difficulty는 Block Header의 Bits로 저장
- Bits 로 부터 Target 및 Difficulty 계산 방법

Btis : `0x1b0404cb`

Target : `0x0404cb * 2**(8*(0x1b - 3))`
= `0x0000000000404CB000000000000000000000000000000000000000`

- Difficulty 1 (maximum target) : `0x1d00ffff`

= `0x00ffff * 2**(8*(0x1d - 3))` = `0x00000000FFFF0000000000000000000000000000000000000000000000000000`

- Difficulty (0x1b0404cb) :

`0x00000000FFFF0000000000000000000000000000000000000000000000000000` /
`0x0000000000404CB000000000000000000000000000000000000000`
= `16307.420938523983`

Difficulty

- 조정 주기 (Epoch) : 2016블록 (약 2주)
- 난이도 조정 비율 = 2016 블록 채굴 소요 시간 / 2주
  - 2주 = 2016 블록 * 10분 * 60초 = 1,209,600 초
- 다음 난이도 = 현재 난이도 * 난이도 조정 비율
  - 최소값 = 1, 최대값: 제한 없음
- 난이도 보정 계수 : 최대 4배(+300%), 최소 1/4 배(-75%) ➞ 급격한 변화 방지

Header - Bits

- $difficulty = \frac{difficulty\ 1\ target}{current\ target} = \frac{maximum\ target}{current\ target}, \quad current\ target = \frac{maximum\ target}{difficulty}$
- Bits 로 부터 difficulty 계산 방법
- Btis : `0x1b0404cb`
- Bits : 0x1d00ffff

Target : `0x0404cb * 2**(8*(0x1b - 3))`

- 2016 블록 (Difficulty Epoch) 마다 난이도 조정 `0x00000000000404CB000000000000000000000000000000000000000000000000`
- 조정 주기 (Epoch) : 2016블록 (약 2주)
- 난이도 조정 비율 = $\frac{2016\ 블록\ 채굴\ 소요\ 시간(초)}{1209600}$ (2016 블록 * 10분 * 60초 = 1,209,600 초)
- 다음 난이도 = 현재 난이도 * 난이도 조정 비율
  - 최소값 = 1, 최대값: 제한 없음
- 난이도 보정 계수 : 최대 4배(+300%), 최소 ¼배(-75%) ➡ 급격한 변화 방지
- $difficulty = \frac{difficulty\ 1\ target}{current\ target} = \frac{maximum\ target}{current\ target}, \quad current\ target = \frac{maximum\ target}{difficulty}$
  - current target 이 작으면 난이도가 높다
  - difficulty가 높을 수록 current target은 작다.
- difficulty 1 = 0x1d00ffff

Difficulty

- 새 블록을 찾기 어려운 정도는 나타냄
- 조정 주기 (Epoch) : 2016블록 (약 2주)
- 난이도 조정 비율 = $\frac{2016\text{ 블록 채굴 소요 시간(초)}}{1209600}$ (2016 블록 * 10분 * 60초 = 1,209,600 초)
- 다음 난이도 = 현재 난이도 * 난이도 조정 비율
  - 최소값 = 1, 최대값: 제한 없음
- 난이도 보정 계수 : 최대 4배(+300%), 최소 ¼배(-75%) ➞ 급격한 변화 방지
- $difficulty = \frac{difficulty\ 1\ target}{current\ target} = \frac{maximum\ target}{current\ target}$, $current\ target = \frac{maximum\ target}{difficulty}$
  - current target 이 작으면 난이도가 높다
  - difficulty가 높을 수록 current target은 작다.
- difficulty 1 = 0x1d00ffff

Difficulty

- 2016 블록 (Difficulty Epoch) 마다 난이도 조정
- 조정 주기 (Epoch) : 2016블록 (약 2주)
- 난이도 조정 비율 = $\frac{2016\text{ 블록 채굴 소요 시간(초)}}{1209600}$ (2016 블록 * 10분 * 60초 = 1,209,600 초)
- 다음 난이도 = 현재 난이도 * 난이도 조정 비율
  - 최소값 = 1, 최대값: 제한 없음
- 난이도 보정 계수 : 최대 4배(+300%), 최소 ¼배(-75%) ➞ 급격한 변화 방지
- $difficulty = \frac{difficulty\ 1\ target}{current\ target} = \frac{maximum\ target}{current\ target}$, $current\ target = \frac{maximum\ target}{difficulty}$
  - current target 이 작으면 난이도가 높다
  - difficulty가 높을 수록 current target은 작다.
- difficulty 1 = 0x1d00ffff

Difficulty

- 2016 블록 (Difficulty Epoch) 마다 난이도 조정
- 조정 주기 (Epoch) : 2016블록 (약 2주)
- 난이도 조정 비율 = $\frac{\text{2016 블록 채굴 소요 시간(초)}}{1209600}$ (2016 블록 * 10분 * 60초 = 1,209,600 초)
- 다음 난이도 = 현재 난이도 * 난이도 조정 비율
  - 최소값 = 1, 최대값: 제한 없음
- 난이도 보정 계수 : 최대 4배, 최소 ¼배 ➡ 급격한 변화 방지 ➡ Epoch 당 +300%(4x) 또는 -75%(1/4) 로 제한
- bdiff -> 대략적인 난이도
- difficulty = difficulty_1_target / current_target = maximum_target / current_target
  - current_target 이 작으면 난이도가 높이진다.
  - difficulty_1_target (pool difficulty, pdiff) = 선행 32bit 0, 나머지 1인 해시
    0x1d00ffff -> maximum_target = 0x00ffff * 2**(8*(0x1d - 3)) = 0x00000000FFFF0000000000000000000000000000000000000000000000000000
    종종 잘리지 않은 0x00000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF 를 사용함을 유의
- difficulty target : targe의 압축된 표현 -> 0x1b0404db (little endian)
- 0x0404cb * 2**(8*(0x1b - 3)) = 0x00000000000404CB000000000000000000000000000000000000000000000000
- 0x00000000FFFF0000000000000000000000000000000000000000000000000000 /
0x00000000000404CB000000000000000000000000000000000000000000000000
= 16307.420938523983(bdiff)
- 0x00000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF /
0x00000000000404CB000000000000000000000000000000000000000000000000
= 16307.669773817162 (pdiff)(pool difficulty)

- Difficulty 1 = 0x1d00ffff
- maximum_target = 0x00ffff * 2**(8*(0x1d - 3)) = 0x00000000FFFF0000000000000000000000000000000000000000000000000000
- Current Target 보다 작은 것을 찾기

- chainwork - 현재 체인을 생성하기 위해 필요한 hash을 총 수
- https://learnmeabitcoin.com/technical/longest-chain
- chainwork: https://bitcoin.stackexchange.com/questions/26869/what-is-chainwork/26894#26894

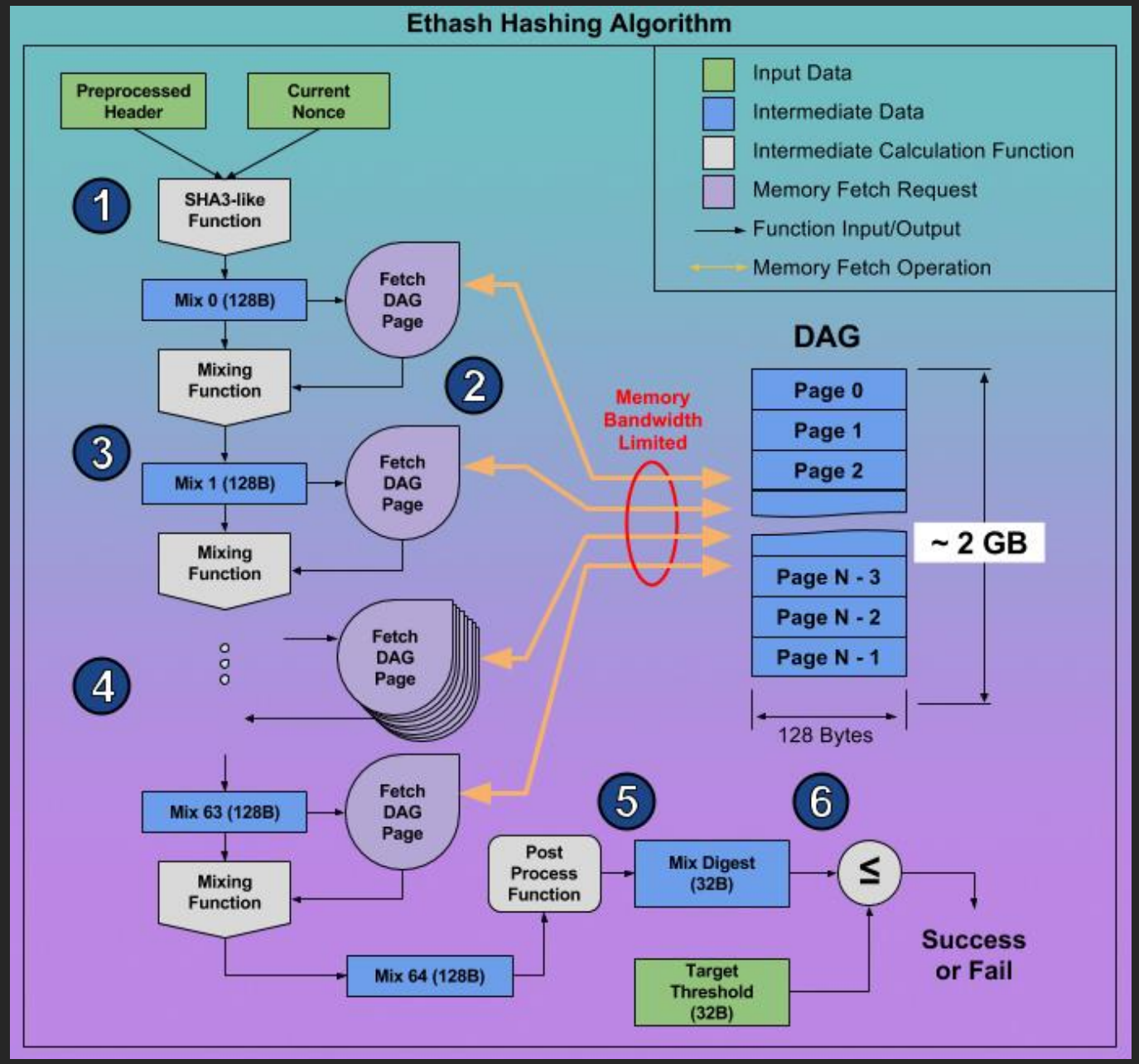
- [0x7e00,0x7e00,0x7e00] = 32256 * [1.0] * [0x01,0001,0001]

- "difficulty": 1.182899534312841

- [chainwork value] = [previous chainwork value] + [difficulty] * [0x01,0001,0001]
- [0x7e01,acd4,2dd2] = [0x7e00,0x7e00,0x7e00] + [1.182899534312841] * [0x01,0001,0001]
- 7e00,0x7e00,0x7e00 : 138540579061248
- 4295032833(0x0100010001) *1.182899534312841 = 5080592338.014062
- 138540579061248 + 5080592338.014062 = 138545659653586 (0x 7E01,ACD4,2DD2)

- hash rate
- CPU vs GPU vs ASIC

이더리움 PoW [ethash]



6. 인센티브

관례상 블록 안의 첫 거래는 블록을 만든 이의 몫이 될 새 화폐로 시작하는 특별한 거래다. 이는 화폐를 발행하는 중앙기관 없이, 노드가 네트워크를 지원할 인센티브를 더해 주며 초기에 발행한 화폐를 유통할 방법을 제공한다. 새 화폐를 일정량 꾸준히 추가하는 것은 금 채굴자가 유통하는 금을 추가하기 위해 자원을 소비하는 것과 유사하다. 우리의 경우 소비되는 것은 CPU 시간과 전기다.

이 인센티브는 거래 수수료(transaction fees)로 충당될 수도 있다. 만일 거래에서 도출된 가치가 투입된 가치보다 작다면, 그 차이가 거래를 포함한 블록의 인센티브 가치에 더해질 거래 수수료다. 애초 정해 놓은 수 만큼의 화폐가 유통되면, 인센티브는 전부 거래 수수료로 바뀌어 인플레이션에서 완전히 자유로워질 수 있다.

이 인센티브는 노드가 계속 정직하게 행동하길 유도하는 데 도움을 줄 수 있다. 만일 탐욕스러운 공격자가 모든 정직한 노드보다 더 많은 CPU 파워를 모을 수 있다면, 그는 그걸 자신의 결제를 도로 훔쳐 사람들을 속이는 데 쓰는 것, 또는 새로운 화폐를 만들어내는 데 쓰는 것 사이에서 선택해야 한다. 그는 규칙대로 움직이는 게 더 이득임을 알게 돼 있는데, 규칙은 그에게 다른 모두의 몫을 합친 것보다, 시스템과 그가 보유한 부의 유효성을 해치는 것보다 더 많은 새 화폐를 베푼다.

6. Incentive

By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

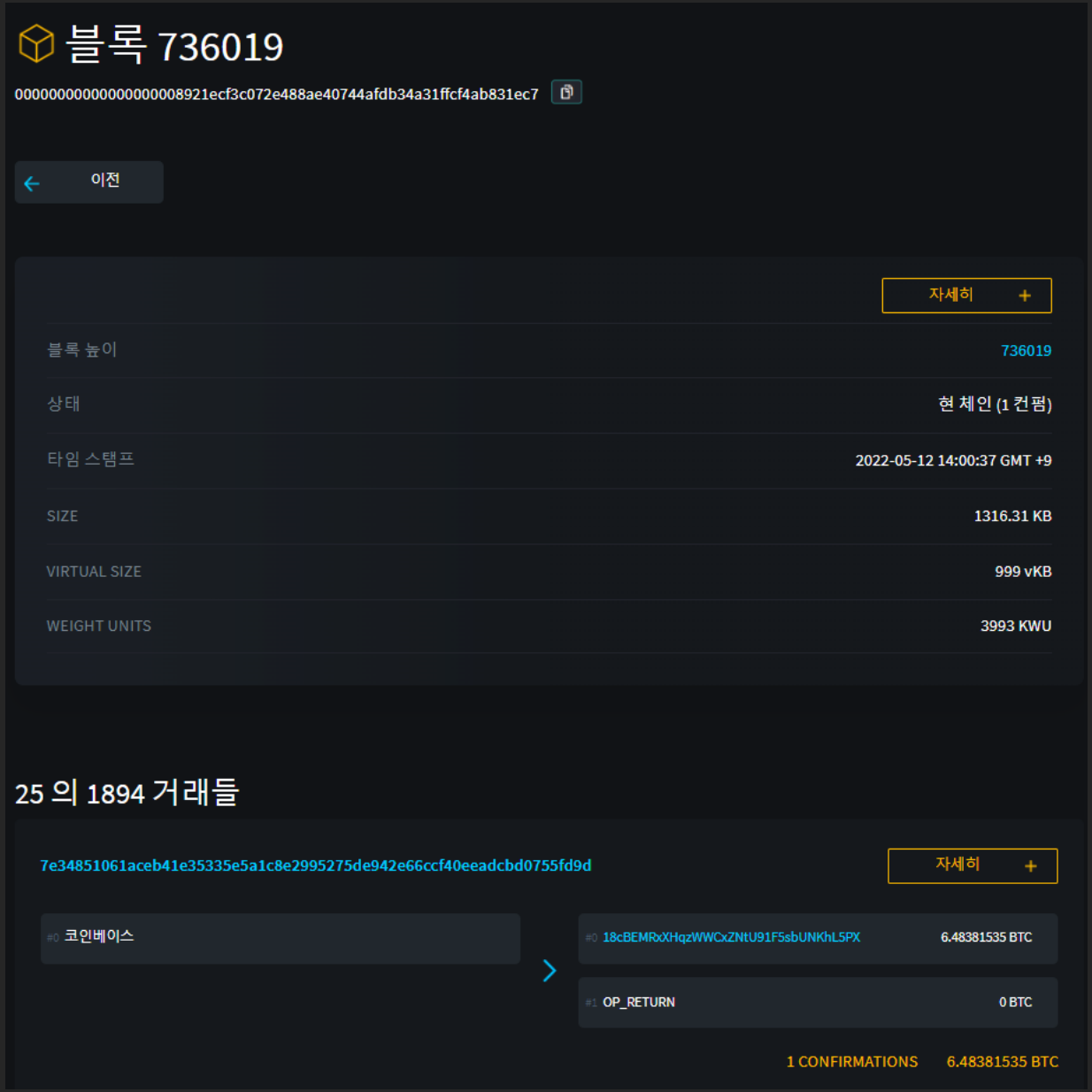
Block

Header

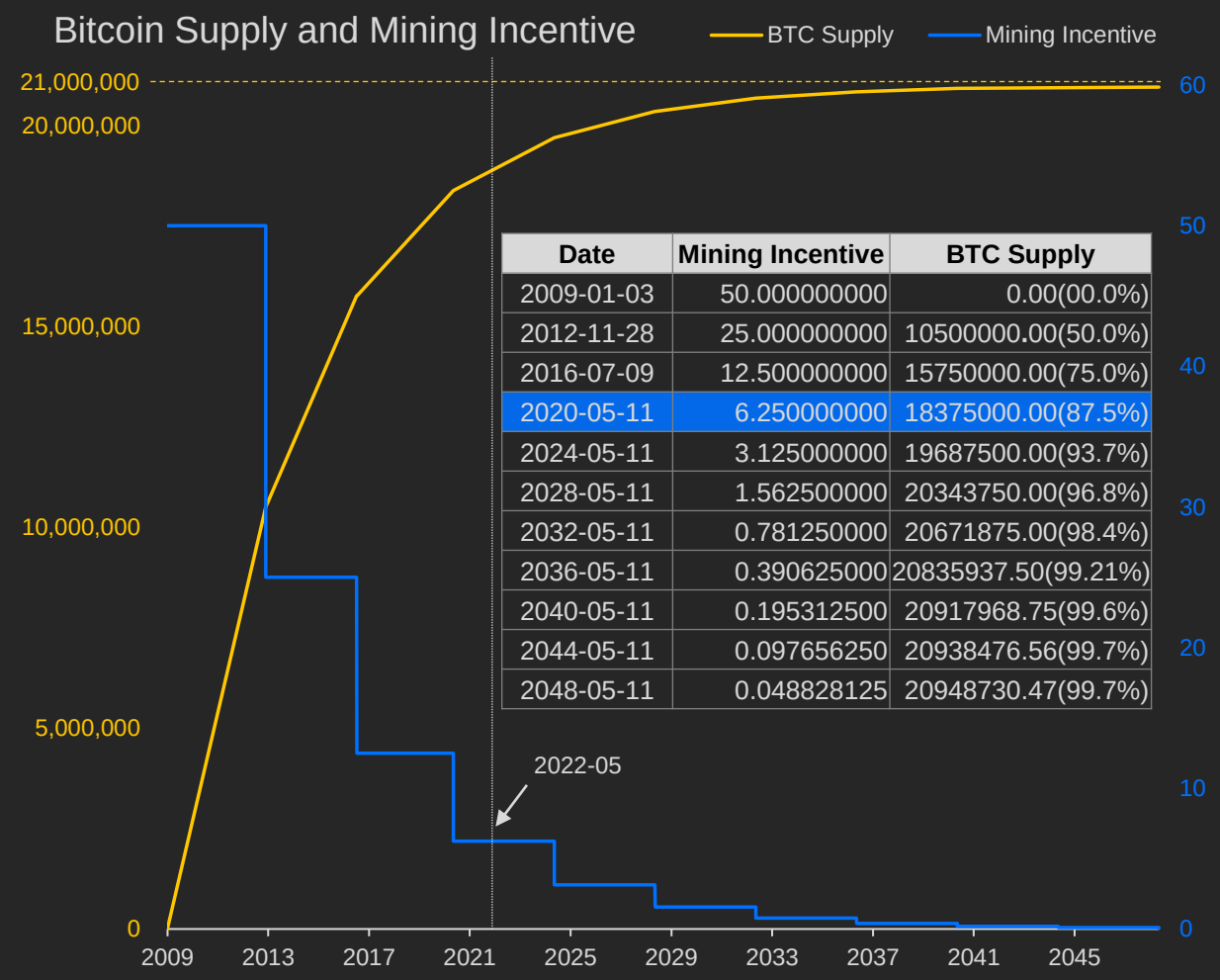
Version
Previous Block Hash
Merkle Root
Timestamp
Difficulty Target
Nonce

Transactions

Tx0 Coinbase Transaction Coinbase ➞ Miner (Mining Incentive + Txs Fee)
⋮



- 최대 21,000,000 BTC 공급
- 최초 보상은 Block 당 50 BTC,
- 21만 블록(약 4년)마다 절반으로 감소



Date	Mining Incentive	BTC Supply
2009-01-03	50.000000000	0.00(00.0%)
2012-11-28	25.000000000	10500000.00(50.0%)
2016-07-09	12.500000000	15750000.00(75.0%)
2020-05-11	6.250000000	18375000.00(87.5%)
2024-05-11	3.125000000	19687500.00(93.7%)
2028-05-11	1.562500000	20343750.00(96.8%)
2032-05-11	0.781250000	20671875.00(98.4%)
2036-05-11	0.390625000	20835937.50(99.21%)
2040-05-11	0.195312500	20917968.75(99.6%)
2044-05-11	0.097656250	20938476.56(99.7%)
2048-05-11	0.048828125	20948730.47(99.7%)

Bitcoin v0.1 released Satoshi Nakamoto, 2009-01-08 19:27:40 UTC
...

Total circulation will be 21,000,000 coins. It'll be distributed to network nodes when they make blocks, with the amount cut in half every 4 years.

first 4 years: 10,500,000 coins
next 4 years: 5,250,000 coins
next 4 years: 2,625,000 coins
next 4 years: 1,312,500 coins
etc...

When that runs out, the system can support transaction fees if needed. It's based on open market competition, and there will probably always be nodes willing to process transactions for free.

Satoshi Nakamoto

https://satoshi.nakamotoinstitute.org/emails/cryptography/16/#selection-123.0-151.16

PoW [Bitcoin Code - Block Hash]

https://github.com/bitcoin/bitcoin/blob/master/src/rpc/mining.cpp#L117

```
// block.h - CBlcockHeader
```

https://github.com/bitcoin/bitcoin/blob/master/src/primitives/block.h#L20

Block Header

```
class CBlockHeader
{
public:
    // header
    int32_t nVersion;
    uint256 hashPrevBlock;
    uint256 hashMerkleRoot;
    uint32_t nTime;
    uint32_t nBits;
    uint32_t nNonce;

    CBlockHeader()
    {
        SetNull();
    }
```

Serialize

```
    SERIALIZE_METHODS(CBlockHeader, obj) { READWRITE(
        obj.nVersion, obj.hashPrevBlock, obj.hashMerkleRoot,
        obj.nTime, obj.nBits, obj.nNonce); }
…
}
```

https://github.com/bitcoin/bitcoin/blob/master/src/primitives/block.cpp#L11

```
// block.cpp - CBlcock
```

Block Hash

```
uint256 CBlockHeader::GetHash() const
{
    return SerializeHash(*this);
}
```

```
// src/hash.h
class CHashWriter
{
…
public:
…
    /** Compute the double-SHA256 hash of all data written to this object.
     *
     * Invalidates this object.
     */
```

Double SHA256

```
    uint256 GetHash() {
        uint256 result;
        ctx.Finalize(result.begin());
        ctx.Reset().Write(
            result.begin(),CSHA256::OUTPUT_SIZE).Finalize(result.begin());
        return result;
    }

    template<typename T>
    CHashWriter& operator<<(const T& obj) {
        // Serialize to this stream
        ::Serialize(*this, obj);
    return (*this);
    }
…
}

/** Compute the 256-bit hash of an object's serialization. */
template<typename T>
uint256 SerializeHash(const T& obj, int nType=SER_GETHASH, int
nVersion=PROTOCOL_VERSION)
{
    CHashWriter ss(nType, nVersion);
    ss << obj;
    return ss.GetHash();
}
```

PoW [Bitcoin Code - Generate Block, PoW]

```cpp
// mining.cpp                                   https://github.com/bitcoin/bitcoin/blob/master/src/rpc/mining.cpp#L117

static bool GenerateBlock(ChainstateManager& chainman, CBlock& block,
        uint64_t& max_tries, uint256& block_hash)
{
    block_hash.SetNull();
    block.hashMerkleRoot = BlockMerkleRoot(block);                                  Merkle Root

    CChainParams chainparams(Params());

    while (max_tries > 0 && block.nNonce < std::numeric_limits<uint32_t>::max()     Find Nonce
            && !CheckProofOfWork(block.GetHash(), block.nBits,
chainparams.GetConsensus())
            && !ShutdownRequested()) {
        ++block.nNonce;
        --max_tries;
    }
    if (max_tries == 0 || ShutdownRequested()) {
        return false;
    }
    if (block.nNonce == std::numeric_limits<uint32_t>::max()) {
        return true;
    }

    std::shared_ptr<const CBlock> shared_pblock =
        std::make_shared<const CBlock>(block);
    if (!chainman.ProcessNewBlock(chainparams, shared_pblock, true, nullptr)) {     New Block
        throw JSONRPCError(RPC_INTERNAL_ERROR,
            "ProcessNewBlock, block not accepted");
    }

    block_hash = block.GetHash();
    return true;
}
```

```cpp
// pow.cpp                                      https://github.com/bitcoin/bitcoin/blob/master/src/pow.cpp#L74
bool CheckProofOfWork(uint256 hash, unsigned int nBits,
    const Consensus::Params& params) {
    bool fNegative;
    bool fOverflow;
    arith_uint256 bnTarget;

    bnTarget.SetCompact(nBits, &fNegative, &fOverflow);                             Calculate Target

    // Check range
    if (fNegative || bnTarget == 0
            || fOverflow || bnTarget > UintToArith256(params.powLimit))
        return false;

    // Check proof of work matches claimed amount
    if (UintToArith256(hash) > bnTarget)                                            Hash < Target
        return false;

    return true;
}
```

```cpp
// arith_uint256.cpp  https://github.com/bitcoin/bitcoin/blob/master/src/arith_uint256.cpp#L198

// This implementation directly uses shifts instead of going
// through an intermediate MPI representation.
arith_uint256& arith_uint256::SetCompact(
        uint32_t nCompact, bool* pfNegative, bool* pfOverflow) {
    int nSize = nCompact >> 24;
    uint32_t nWord = nCompact & 0x007fffff;
    if (nSize <= 3) {
        nWord >>= 8 * (3 - nSize);
        *this = nWord;
    } else {
        *this = nWord;
        *this <<= 8 * (nSize - 3);
    }
    if (pfNegative)
        *pfNegative = nWord != 0 && (nCompact & 0x00800000) != 0;
    if (pfOverflow)
        *pfOverflow = nWord != 0 && ((nSize > 34) ||
                                     (nWord > 0xff && nSize > 33) ||
                                     (nWord > 0xffff && nSize > 32));
    return *this;
}
```

PoW [Bitcoin Code - Difficulty]

https://github.com/bitcoin/bitcoin/blob/master/src/pow.cpp#L13

```
unsigned int GetNextWorkRequired(const CBlockIndex* pindexLast, const CBlockHeader *pblock, const Consensus::Params& params)
{
    assert(pindexLast != nullptr);
    unsigned int nProofOfWorkLimit = UintToArith256(params.powLimit).GetCompact();

    // Only change once per difficulty adjustment interval
    if ((pindexLast->nHeight+1) % params.DifficultyAdjustmentInterval() != 0)
    {
        if (params.fPowAllowMinDifficultyBlocks)
        {
            // Special difficulty rule for testnet:
            // If the new block's timestamp is more than 2* 10 minutes
            // then allow mining of a min-difficulty block.
            if (pblock->GetBlockTime() > pindexLast->GetBlockTime() + params.nPowTargetSpacing*2)
                return nProofOfWorkLimit;
            else
            {
                // Return the last non-special-min-difficulty-rules-block
                const CBlockIndex* pindex = pindexLast;
                while (pindex->pprev
                        && pindex->nHeight % params.DifficultyAdjustmentInterval() != 0
                        && pindex->nBits == nProofOfWorkLimit)
                    pindex = pindex->pprev;
                return pindex->nBits;
            }
        }
        return pindexLast->nBits;
    }

    // Go back by what we want to be 14 days worth of blocks
    int nHeightFirst = pindexLast->nHeight - (params.DifficultyAdjustmentInterval()-1);
    assert(nHeightFirst >= 0);
    const CBlockIndex* pindexFirst = pindexLast->GetAncestor(nHeightFirst);
    assert(pindexFirst);

    return CalculateNextWorkRequired(pindexLast, pindexFirst->GetBlockTime(), params);
}
```

per Epoch(2016 Blocks)

Find first block of epoch

Calculate Next Difficulty(Bits)

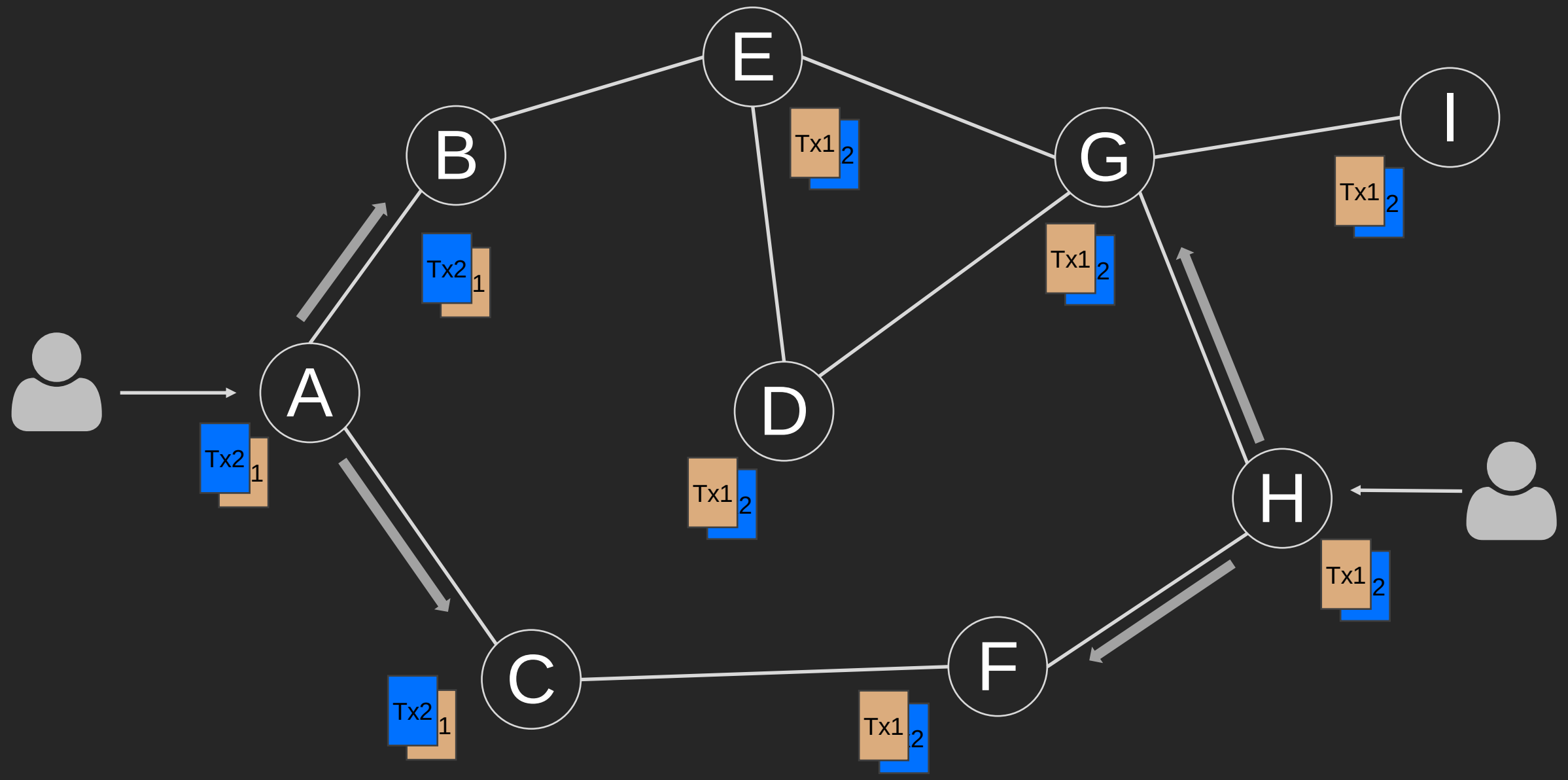
```
unsigned int CalculateNextWorkRequired(const CBlockIndex* pindexLast,
    int64_t nFirstBlockTime, const Consensus::Params& params)
{
    if (params.fPowNoRetargeting)
        return pindexLast->nBits;

    // Limit adjustment step
    int64_t nActualTimespan = pindexLast->GetBlockTime() - nFirstBlockTime;
    if (nActualTimespan < params.nPowTargetTimespan/4)
        nActualTimespan = params.nPowTargetTimespan/4;
    if (nActualTimespan > params.nPowTargetTimespan*4)
        nActualTimespan = params.nPowTargetTimespan*4;

    // Retarget
    const arith_uint256 bnPowLimit = UintToArith256(params.powLimit);
    arith_uint256 bnNew;
    bnNew.SetCompact(pindexLast->nBits);
    bnNew *= nActualTimespan;
    bnNew /= params.nPowTargetTimespan;

    if (bnNew > bnPowLimit)
        bnNew = bnPowLimit;

    return bnNew.GetCompact();
}
```

LastBlock Time - FirstBlock Time

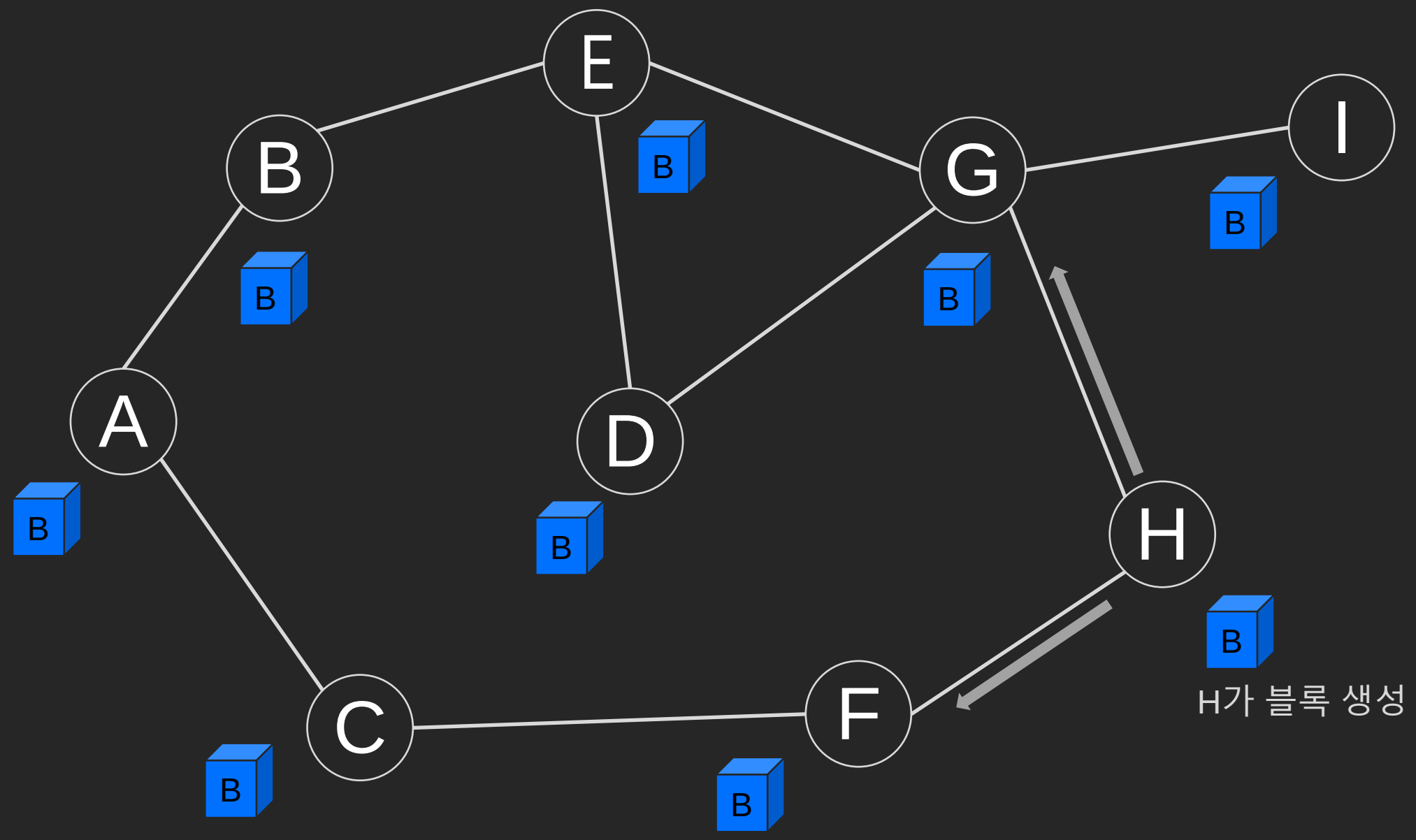
Limit /4(25%)

Limit *4(400%)

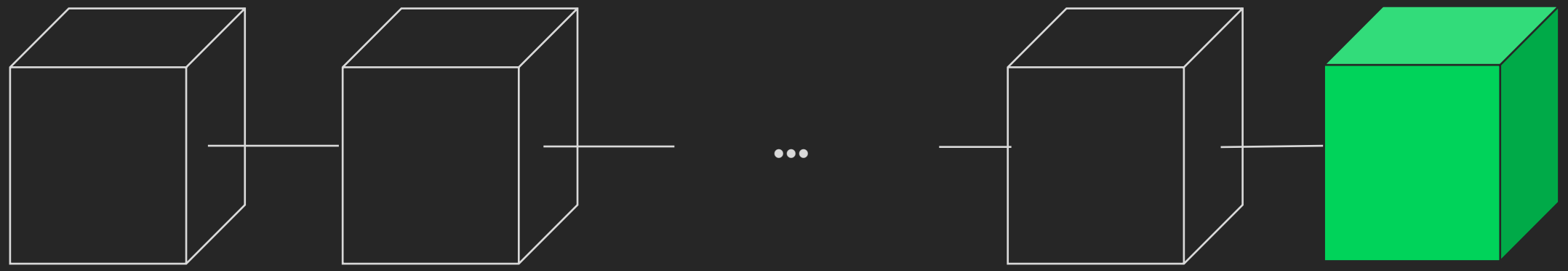
트랜잭션 도착 순서는 네트워크 지연, 노드 다운 등으로 달라질 수 있음



PoW [블록 전파 (Block Broadcast)]



모든 노드는 블록을 검증
검증을 통과하면 블록체인에 추가



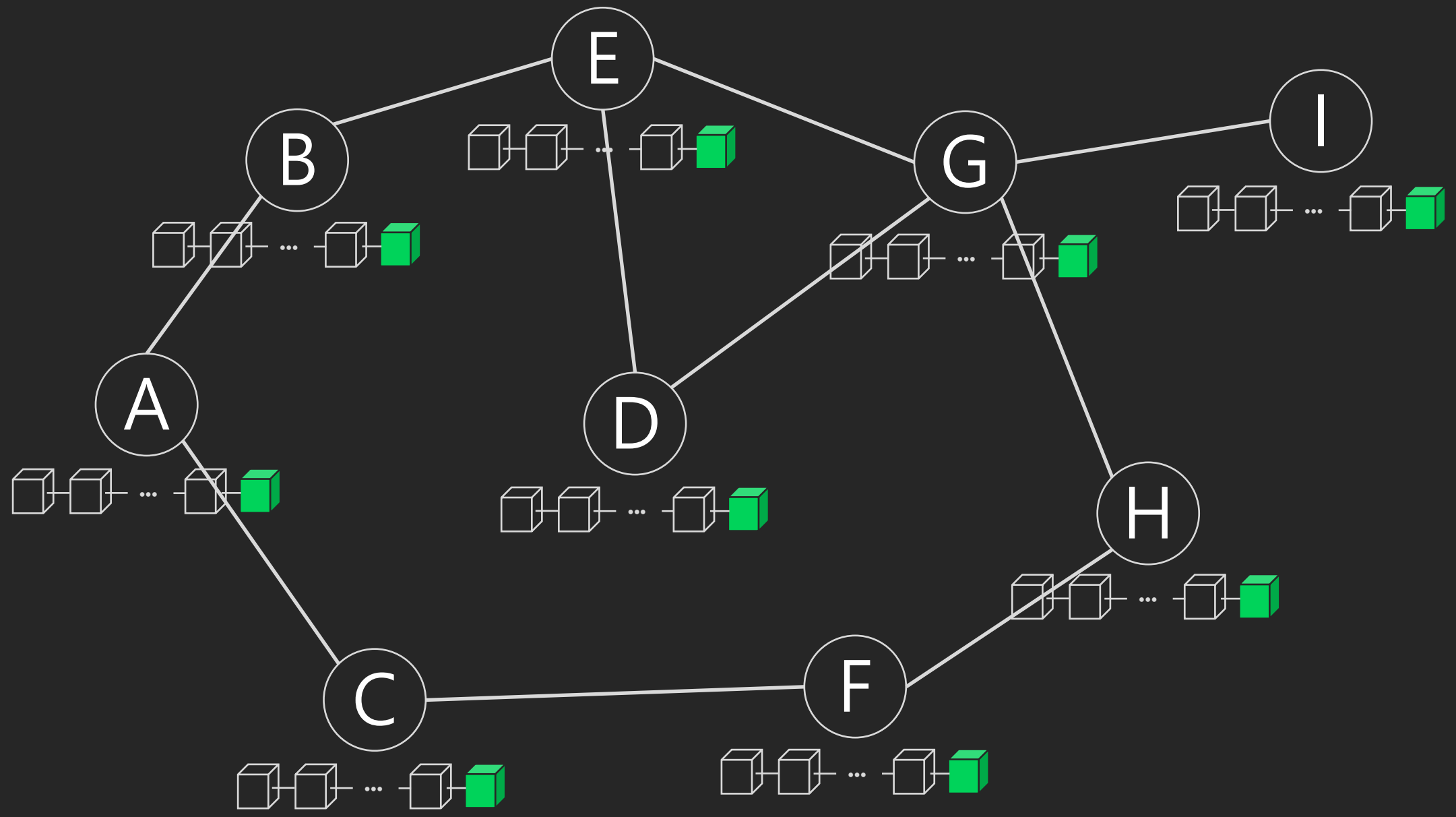
블록 자체 검증
1. 블록 구조
2. PoW(Hash < Difficulty Target)
3. Merkle Root
4. Size <= 4000000 bytes
5. 첫번째 트랜잭션만 Coinbase 여부
6. 모든 트랜잭션 검증
7. Sigop Count <= 80000

블록체인 연관 검증
1. bits(difficulty) 유효성 (이전 epoch와 비교)
2. blockTime > 이전 블록 blockTime (mediumTime 체크 필요*)
3. blockTime < adjustedBlockTime + 2 hours
4. version 유효성
   - 1: height < 227931(BIP-34, Height in Coinbase)
   - 2: height < 363725(BIP-66, String DER signatures)
   - 3: height < 388381(BIP-65, OP_CHECKLOCKTIMEVERIFY)

트랜잭션 검증
1. inputs not empty
2. outputs not empty
3. Size <= 4000000 bytes
4. 0 <= outputs total value <= 21000000 BTC
5. input 중복 사용 체크
6. IF coinbase : 2 <= scriptsigSize <= 100
7. IF !conbase : in's out not empty

Size : 트랜잭션 전체 Size <= 1000000 bytes, 블록 전체 Size <= 4000000 bytes (수정 필요)

검증을 통과한 블록을 기존 블록체인에 추가

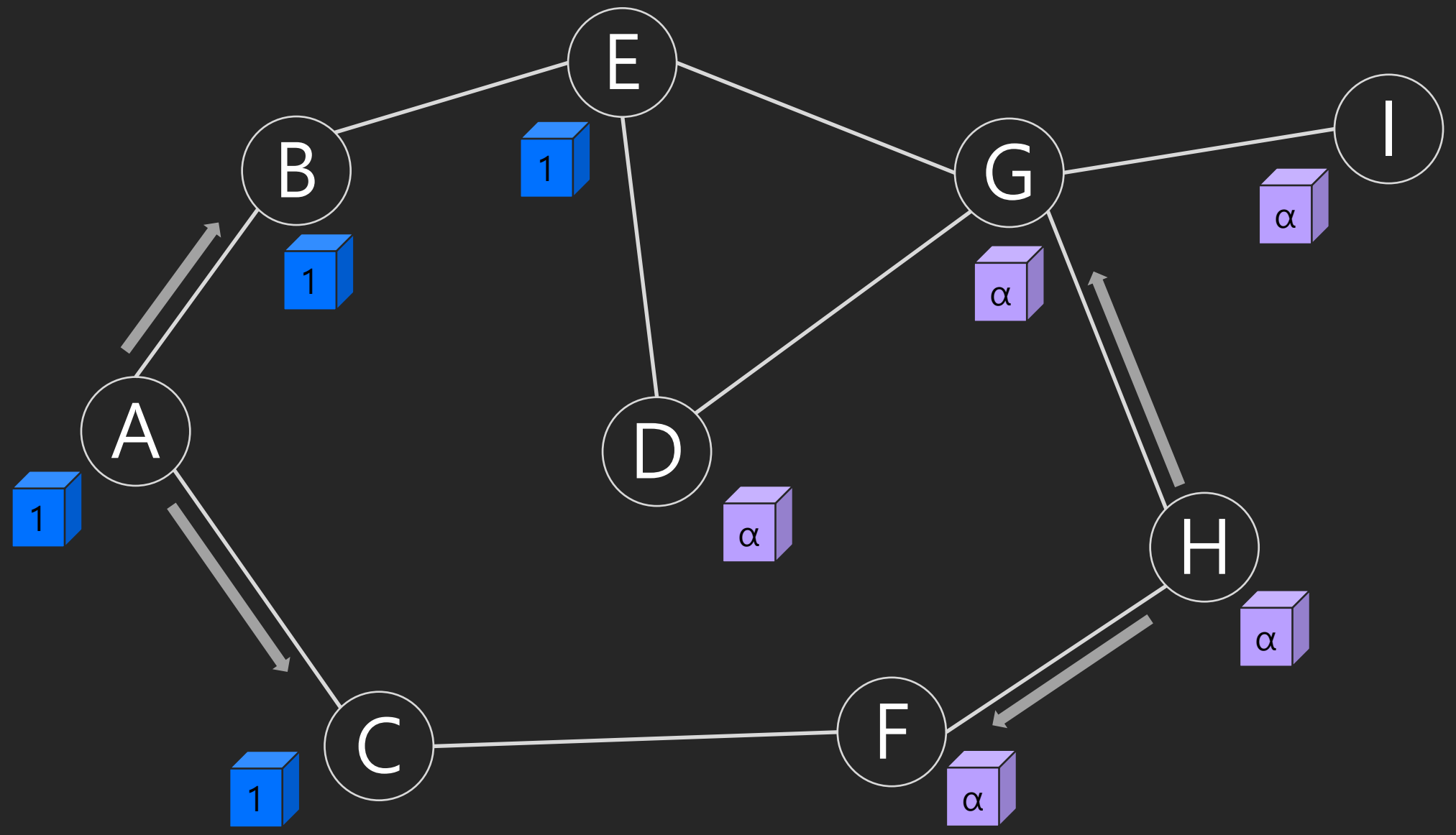


PoW [Bitcoin Contextual Check Block Header]

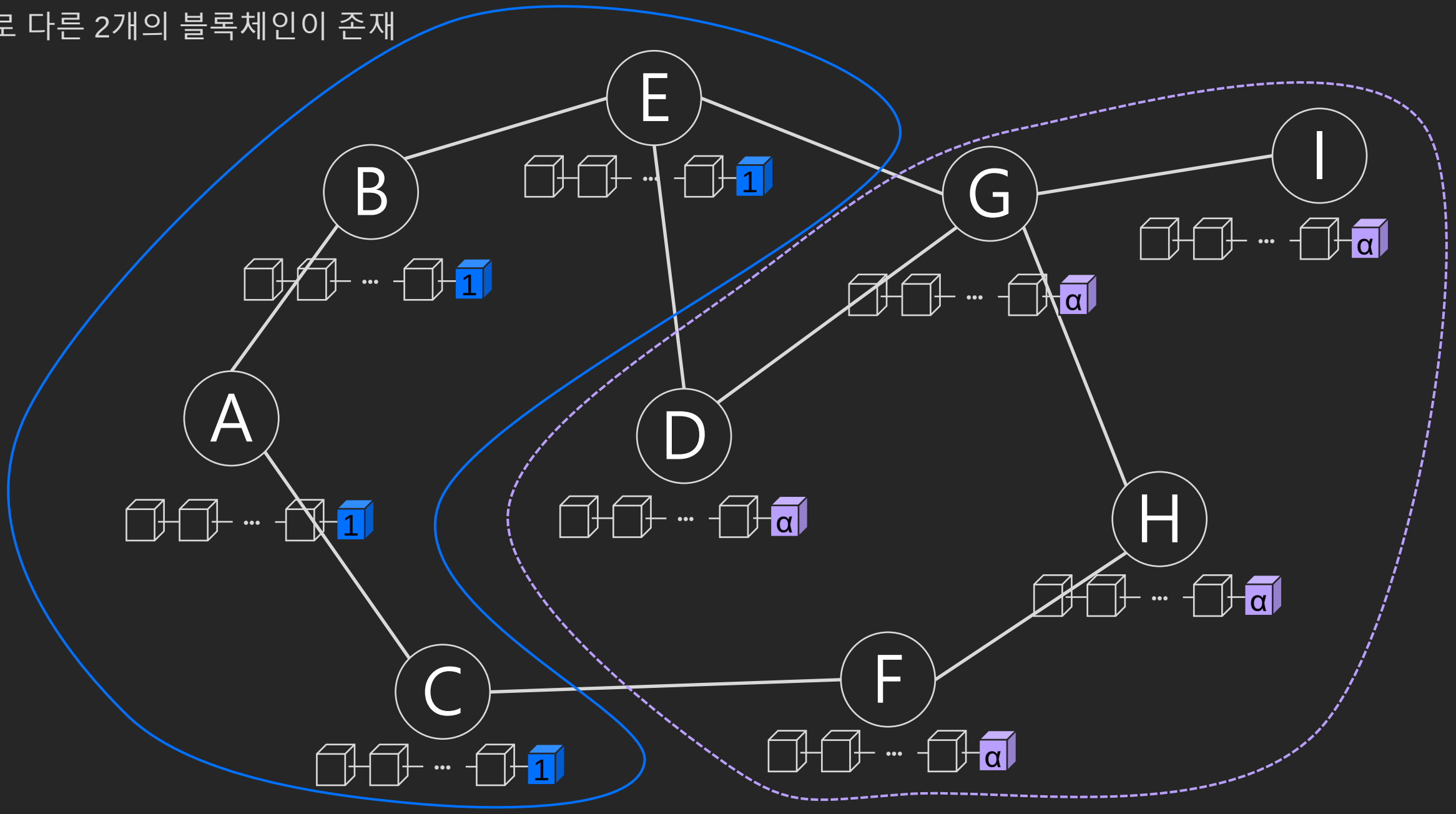
```cpp
/** Context-dependent validity checks.
 *  By "context", we mean only the previous block headers, but not the UTXO
 *  set; UTXO-related validity checks are done in ConnectBlock().
 *  NOTE: This function is not currently invoked by ConnectBlock(), so we
 *  should consider upgrade issues if we change which consensus rules are
 *  enforced in this function (eg by adding a new consensus rule). See comment
 *  in ConnectBlock().
 *  Note that -reindex-chainstate skips the validation that happens here!
 */
static bool ContextualCheckBlockHeader(const CBlockHeader& block,
BlockValidationState& state, BlockManager& blockman, const CChainParams& params,
const CBlockIndex* pindexPrev, int64_t nAdjustedTime)
EXCLUSIVE_LOCKS_REQUIRED(::cs_main)
{
    AssertLockHeld(::cs_main);
    assert(pindexPrev != nullptr);
    const int nHeight = pindexPrev->nHeight + 1;

    // Check proof of work
    const Consensus::Params& consensusParams = params.GetConsensus();
    if (block.nBits != GetNextWorkRequired(pindexPrev, &block, consensusParams))
        return state.Invalid(BlockValidationResult::BLOCK_INVALID_HEADER,
            "bad-diffbits", "incorrect proof of work");

    // Check against checkpoints
    if (fCheckpointsEnabled) {
        // Don't accept any forks from the main chain prior to last checkpoint.
        // GetLastCheckpoint finds the last checkpoint in MapCheckpoints
        // that's in our BlockIndex().
        const CBlockIndex* pcheckpoint =
            blockman.GetLastCheckpoint(params.Checkpoints());
        if (pcheckpoint && nHeight < pcheckpoint->nHeight) {
            LogPrintf(
                "ERROR: %s: forked chain older than last checkpoint (height %d)\n",
                __func__, nHeight);
            return state.Invalid(BlockValidationResult::BLOCK_CHECKPOINT,
                "bad-fork-prior-to-checkpoint");
        }
    }

    // Check timestamp against prev
    if (block.GetBlockTime() <= pindexPrev->GetMedianTimePast())
        return state.Invalid(BlockValidationResult::BLOCK_INVALID_HEADER,
            "time-too-old", "block's timestamp is too early");

    // Check timestamp
    if (block.GetBlockTime() > nAdjustedTime + MAX_FUTURE_BLOCK_TIME)
        return state.Invalid(BlockValidationResult::BLOCK_TIME_FUTURE,
            "time-too-new", "block timestamp too far in the future");

    // Reject blocks with outdated version
    if ((block.nVersion < 2 && DeploymentActiveAfter(
                pindexPrev, consensusParams, Consensus::DEPLOYMENT_HEIGHTINCB)) ||
        (block.nVersion < 3 && DeploymentActiveAfter(
                pindexPrev, consensusParams, Consensus::DEPLOYMENT_DERSIG)) ||
        (block.nVersion < 4 && DeploymentActiveAfter(
                pindexPrev, consensusParams, Consensus::DEPLOYMENT_CLTV))) {
            return state.Invalid(
                BlockValidationResult::BLOCK_INVALID_HEADER,
                strprintf("bad-version(0x%08x)", block.nVersion),
                strprintf("rejected nVersion=0x%08x block", block.nVersion));
    }

    return true;
}
```

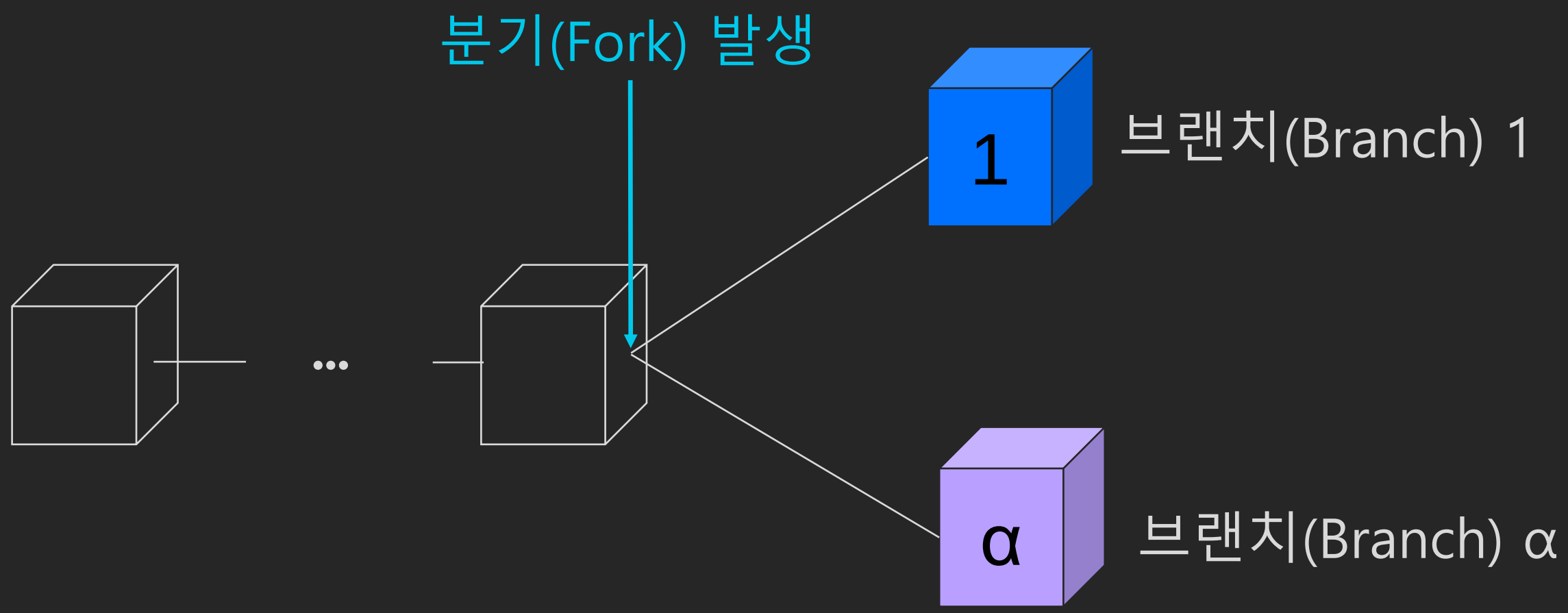
```cpp
int64_t GetMedianTimePast() const
{
    int64_t pmedian[nMedianTimeSpan];
    int64_t* pbegin = &pmedian[nMedianTimeSpan];
    int64_t* pend = &pmedian[nMedianTimeSpan];

    const CBlockIndex* pindex = this;
    for (int i = 0; i < nMedianTimeSpan && pindex; i++, pindex = pindex->pprev)
        *(--pbegin) = pindex->GetBlockTime();

    std::sort(pbegin, pend);
    return pbegin[(pend - pbegin) / 2];
}
```

A, H가 동시에 블록 생성 성공



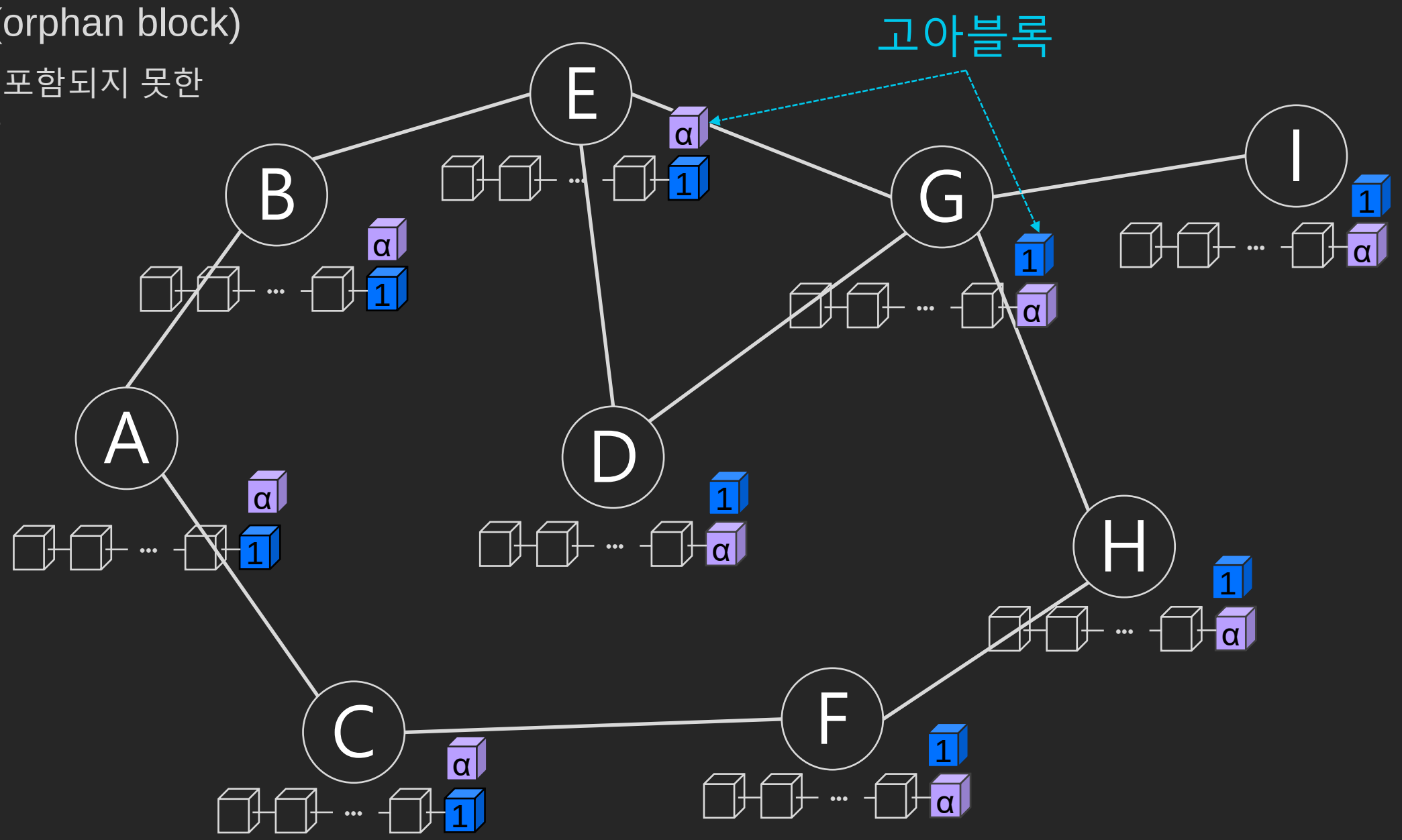
서로 다른 2개의 블록체인이 존재




분기(Fork) 발생
1
브랜치(Branch) 1
α
브랜치(Branch) α

고아블록(orphan block)

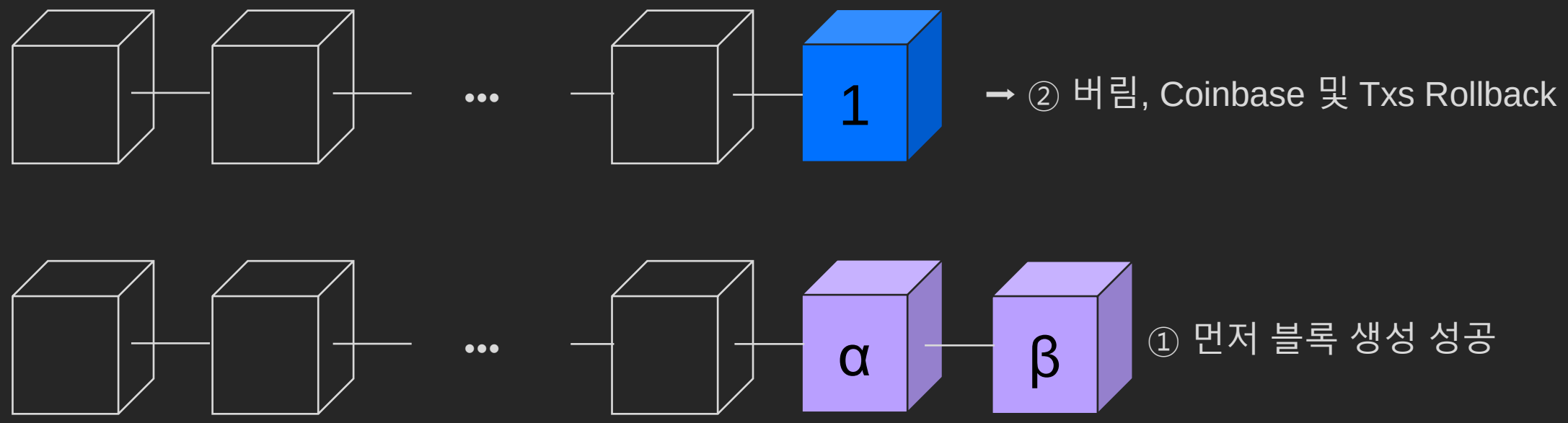
메인체인에 포함되지 못한
유효한 블록



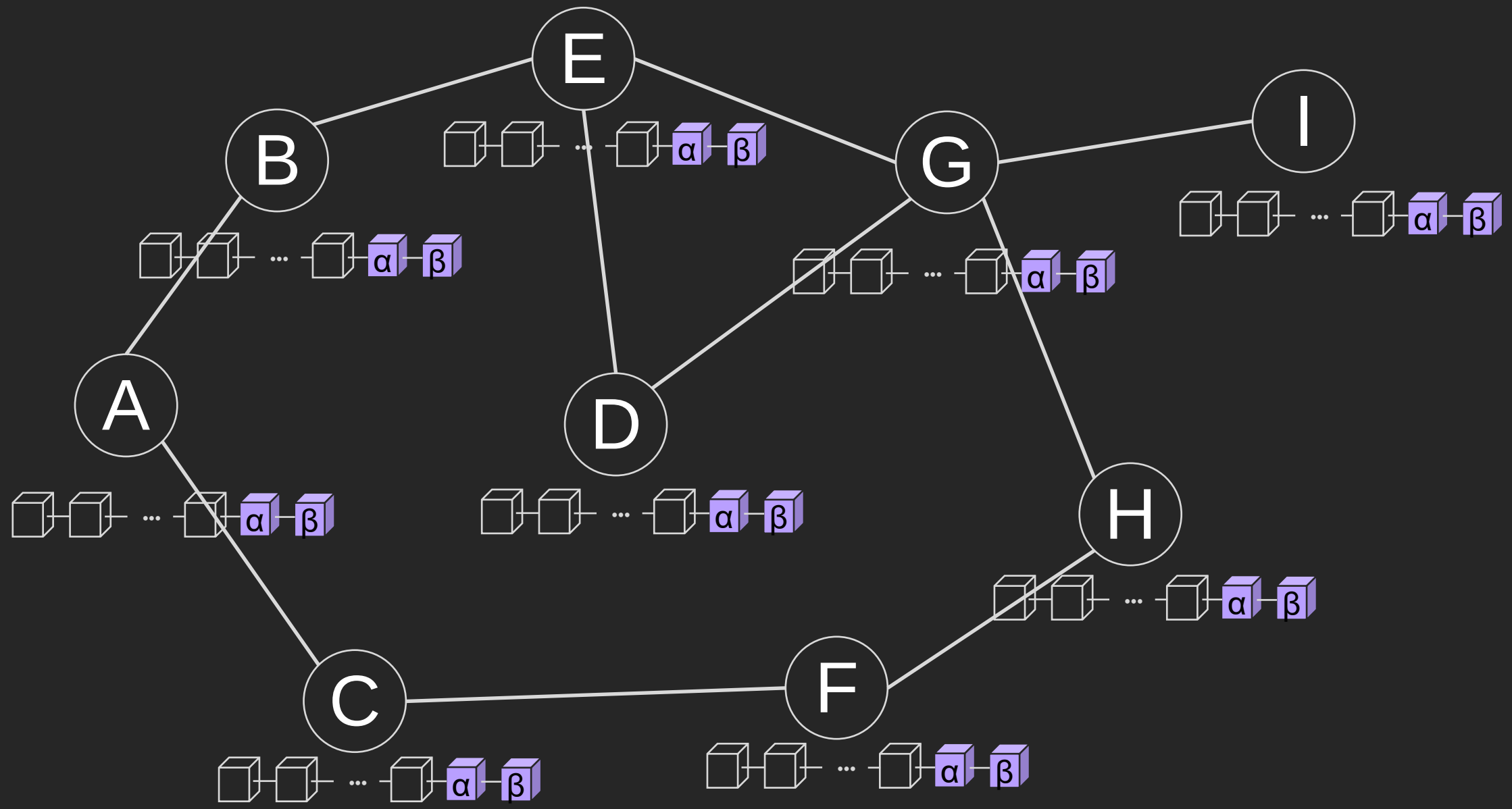
어떤 블록을 선택할 것인가?

가장 긴 체인을 선택
(Longest Chain Selection)

- 각 체인은 더 긴 체인을 생성하기 위한 경쟁 시작
- 긴 체인을 생성한 체인을 메인 체인으로 선택, 나머지 체인은 버림
  - 버려진 블록의 Coinbase 및 트랜잭션은 Rollback
  - 버려진 블록의 트랜잭션이 메인 체인에 포함되지 않은 경우 트랜잭션 풀로 들어감
- The Longest Chain: 누적 Difficulty 비교 (블록 생성에 더 많은 에너지를 소모한 체인)



하나의 메인 체인으로 네트워크가 합의



이더리움 (Heaviest Chain Selection)

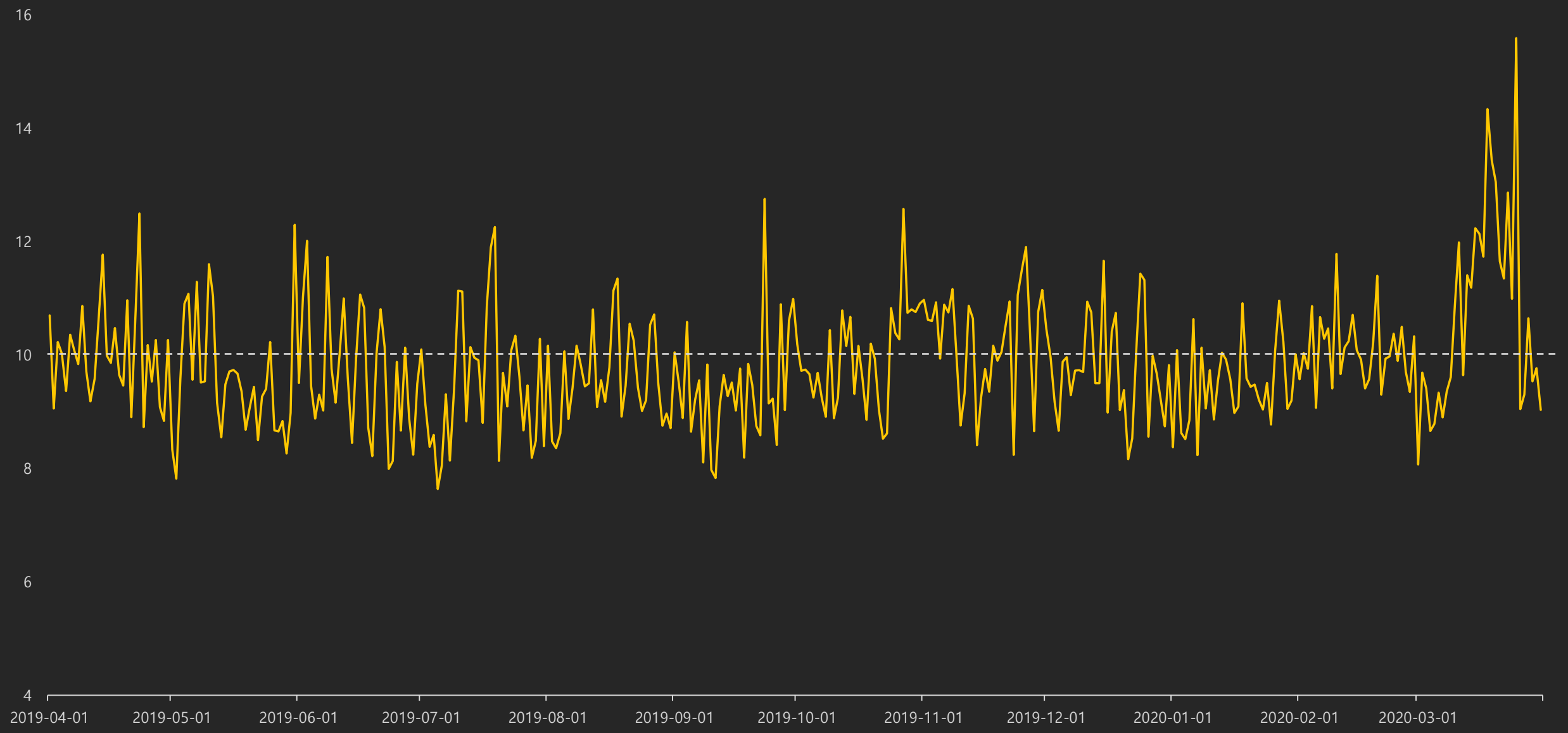
- Heaviest : 각 블록을 Root로 하는 Subtree의 크기
- 누적 difficulty를 비교
- modified GHOST protocol
  - GHOST 프로토콜 (Greedy Heaviest Object subtree protocol)의 7단계 레벨만 포함
    요나탄 솜폴린스키(Yonatan Sompolinsky)와 아비브 조우하르(Aviv Zohar)에 의해 2013년 12월 소개된 프로토콜
  - difficulty 계산에 Uncle block(Ommer Block) 을 포함 (최대 2개, 중복 불가)
    - Uncle Block : 2~7번째 조상의 직접 자손
    - Reward
      - Uncle block을 포함한 Block : Block Reward 3.125%
      - Uncle Block : Block Reward 25% ~ 87.5% [(Uncle Block Height – Block Height + 8) * Block Reward / 8]
    - Uncle Block이 포함되면 Block 1개를 추가 채굴한 것으로 간주하고 난이도 조정
- 짧은 Block Time (12 ~14초)에 따른 빈번한 분기(Fork)로 인한 네트워크 리소스 낭비를 막기 위함

Bitcoin Average Block Time (2019-04-01 ~ 2020-03-31)

https://data.bitcoinity.org/bitcoin/block_time/5y?r=day&t=l

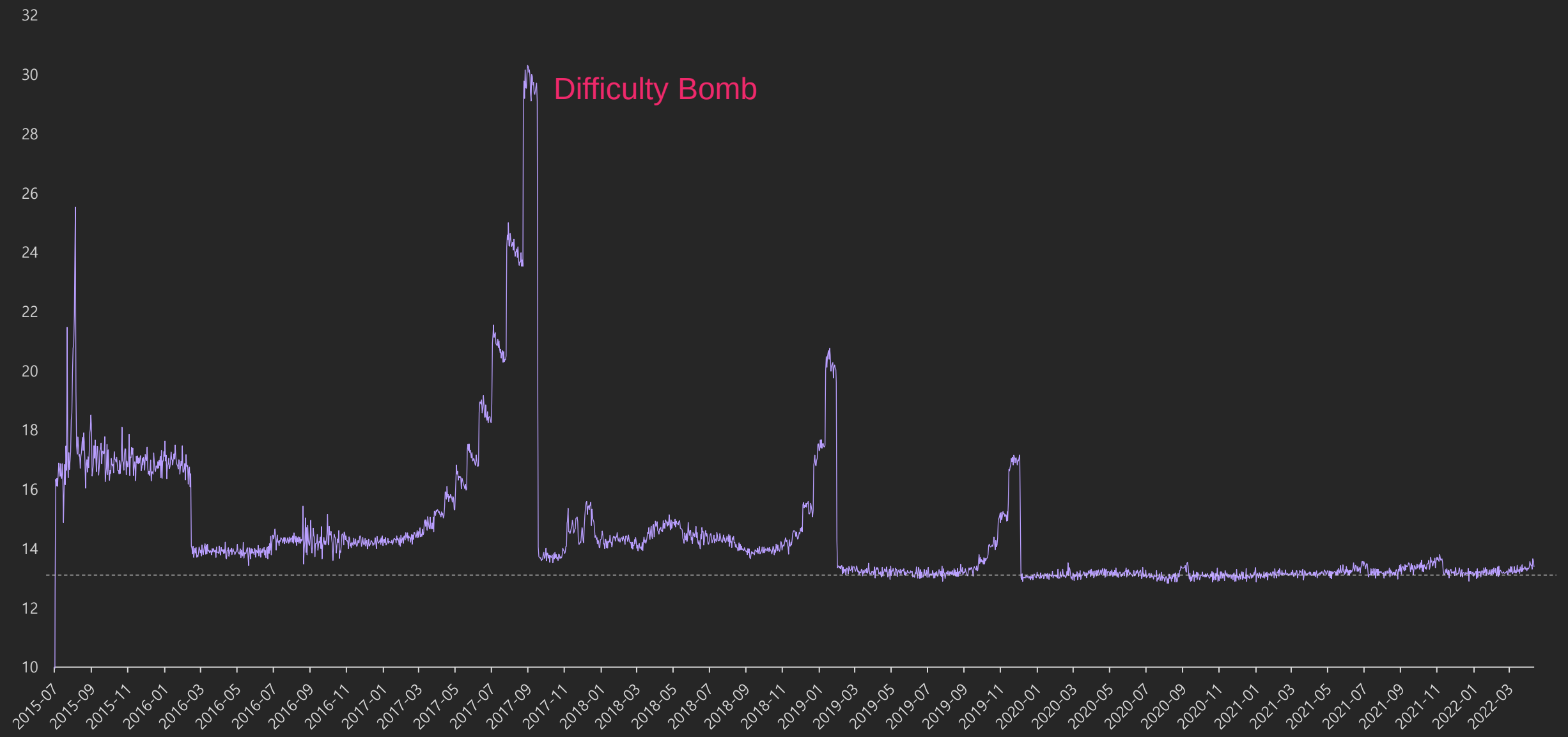
비트코인 블록타임 타켓 : 10분

Bitcoin Average Block Time(m)



이더리움 블록타임 타켓 : 13초

Ethereum Average Block Time



이중 지불

51% 문제

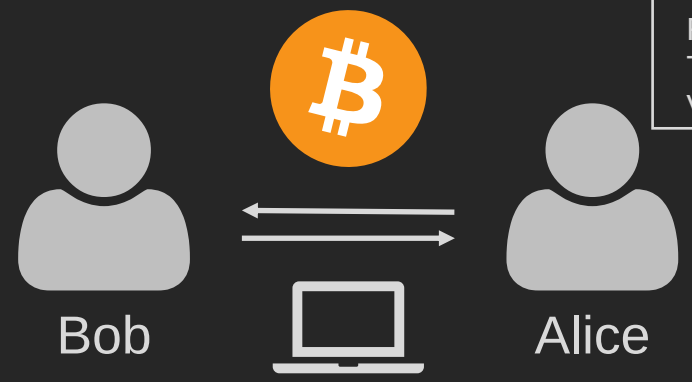
완결성(Finality)

성능한계

블록체인 용량

이중 지불(Double-spending) : 동일 디지털 토큰을 두 번 이상 사용

레이스 어택(Race Attack)



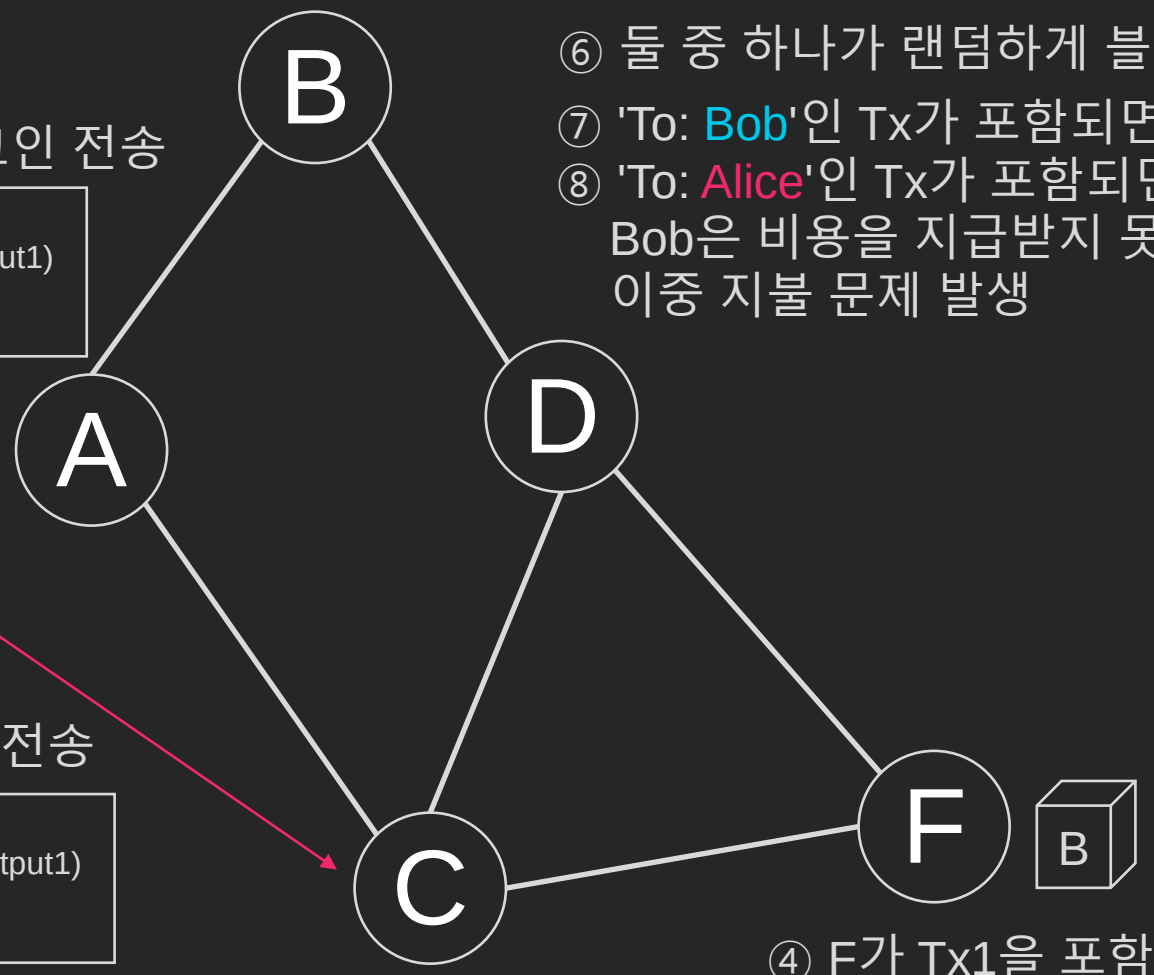
① 판매자(Bob)에게 코인 전송

Tx1
From: Alice (output1)
To: Bob
Value: 1000

② Bob이 전송내역 확인 후 떠남

③ 본인(Alice)에게 코인 전송

Tx1
From: Alice (output1)
To: Alice
Value: 1000



④ F가 Tx1을 포함한 블록 생성

⑤ 어느 Transaction이 블록에 포함되는가?

⑥ 둘 중 하나가 랜덤하게 블록에 포함

⑦ 'To: Bob'인 Tx가 포함되면 문제 없음

⑧ 'To: Alice'인 Tx가 포함되면
Bob은 비용을 지급받지 못하는
이중 지불 문제 발생

레이스 어택 (Race Attack)

- 두 개의 충돌하는 트랜잭션을 보내서 더 빠른 하나의 트랜잭션만 받아 들여짐
- 공격자는 서비스를 제공받기 위해 지불한 코인을 다시 한번 본인에게 전송
- 50% 확률로 본인에게 전송한 트랜잭션이 메인 체인에 포함되면 공격 성공

피니 어택 (Finney Attack) - 타켓 : 0 컨펌을 믿는 대상, Miner 필요

0 컨펌: 트랜잭션 전송만 한 상태 (블록에 포함되지 않음)

- 본인에게 코인을 전송하는 트랜잭션을 포함한 블록을 채굴
- 본인에게 전송하는 코인을 이용하여 타켓과 서비스 거래 시도
- 타켓이 0 컨펌을 믿고 서비스 거래를 완료하면, 공격자는 블록을 네트워크에 전파
- 공격자의 트랜잭션이 메인체인에 포함되고, 타켓은 코인을 전송받지 못 함.
- 단, 다른 Miner가 먼저 채굴에 성공하면 공격 실패 및 블록 채굴 보상의 손해

- **벡터 76 어택** - 타켓 : 1컨펌을 믿을 대상, Miner 필요
  - 타켓과 서비스 거래 시도 후, 타켓에게 코인을 전송하는 트랜잭션이 포함된 블록을 채굴, 트랜잭션은 전파하지 않음
  - 블록을 전파하지 않고, 제3자의 채굴 성공 대기 (제3자의 블록에는 타켓에게 코인을 전송하는 트랜잭션 미포함)
  - 제3자가 채굴을 성공과 동시에 타켓에게 본인 블록의 생성을 알림
  - 타켓은 1컨펌을 믿고 서비스 거래를 완료하고, 공격자는 제3자의 체인과 Fork 상태가 되어서 경쟁 시작
    - 제3자 체인이 승리 : 타켓에게 코인을 전송한 트랜잭션 무효
    - 공격자 체인이 승리 : 블록 보상 획득

- **리플레이 어택** (Replay Attack)
  - 포크가 일어나 블록체인이 분리되어도 트랜잭션은 모든 체인에서 유효함을 이용
  - 악의적으로 포크한 체인을 통해 타켓에게 코인을 전송
  - 타켓은 이 코인을 다른 지갑에 전송 -> 메인 체인과 포크 체인 모두에게 전송
  - 포크체인이 패배하면, 타켓 입장에서는 받은 코인은 없고, 코인을 보낸 기록에 메인 체인에 남음

51% 공격

- 51% ➡ 과반수 ➡ 네트워크를 장악할 수 있는 충분한 힘을 표현
- 해커가 합의 의결권의 51% 이상 이상을 점유할 경우 3가지 공격 가능
  - 51% 점유기간동안 일어난 해커 본인의 전송 내역을 뒤집어 이중 지불을 할 수 있다.
  - 특정/모든 전송이 승인 받지 못하게 할 수 있다.
  - 특정/모든 채굴자가 확정 블록을 채굴하지 못하게 할 수 있다.
- 51% 이상의 의결권을 점유한다고 하더라도 할 수 없는 일
  - 타인의 전송 내역을 그들의 협조 없이 뒤집을 수는 없다.
  - 블록당 생성되는 보상 코인의 수를 바꿀 수는 없다.
  - 세상에 없던 코인을 새로 발행하거나 타인의 코인을 임의로 자신에게 송금할 수는 없다.
- 방어를 위해서는 구성원이 풀노드 구성 필요
  - 라이트 노드는 특정 채굴자나 풀노드를 신뢰해야하기 때문에 공격에 피해를 받을 수 있다.

이클립스 어택

- 하나의 노드를 대상
- 특정 노드와 통신하는 다수의 노드가 공격자의 허위전송내역을 담은 블록을 타켓 노드에 전파

완결성(Finality)

- 거래(Transaction)가 더 이상 되돌려지지 않는 상태
- Fork 발생과 Longest Chain Selection 규칙 때문에 Finality를 보장할 수 없음
- 블록체인 길이가 길어질수록 Finality 확률 증가 (100% 불가)

컨펌 (Confirmation)

- Finality 가 성립했다고 기대하는 블록 수
- Satoshi Nakamoto: 6 confirmations (1 hour)

긴 Block Time

- Fork 가능성 ⬇ (감소)
- Finality 가능성 ⬆ (증가)

Bitcoin P2P e-cash paper Satoshi Nakamoto, 2009-01-08 19:27:40 UTC

…

When there are multiple **double-spent** versions of the same transaction, one and **only one will become valid**.

The receiver of a payment **must wait an hour** or so before believing that **it's valid**. The network will resolve any possible double-spend races by then.

The guy who received the double-spend that became invalid never thought he had it in the first place. His software would have shown the transaction go from "unconfirmed" to "invalid". If necessary, the UI can be made to hide transactions until they're sufficiently deep in the block chain.

…

Sorry if I didn't make that clear. The **target time** between blocks will probably be **10 minutes**.

Every block includes its creation time. If the time is off by more than 36 hours, other nodes won't work on it. If the timespan over the last 6*24***30** blocks is less than **15 days**, blocks are being generated too fast and the **proof-of-work difficulty doubles**. Everyone does the same calculation with the same chain data, so they all get the same result at the same link in the chain.

https://satoshi.nakamotoinstitute.org/emails/cryptography/threads/1/#014843

PoW [컨펌(Confirmations) 사례 - 가상자산 거래소]

bithumb 거래소

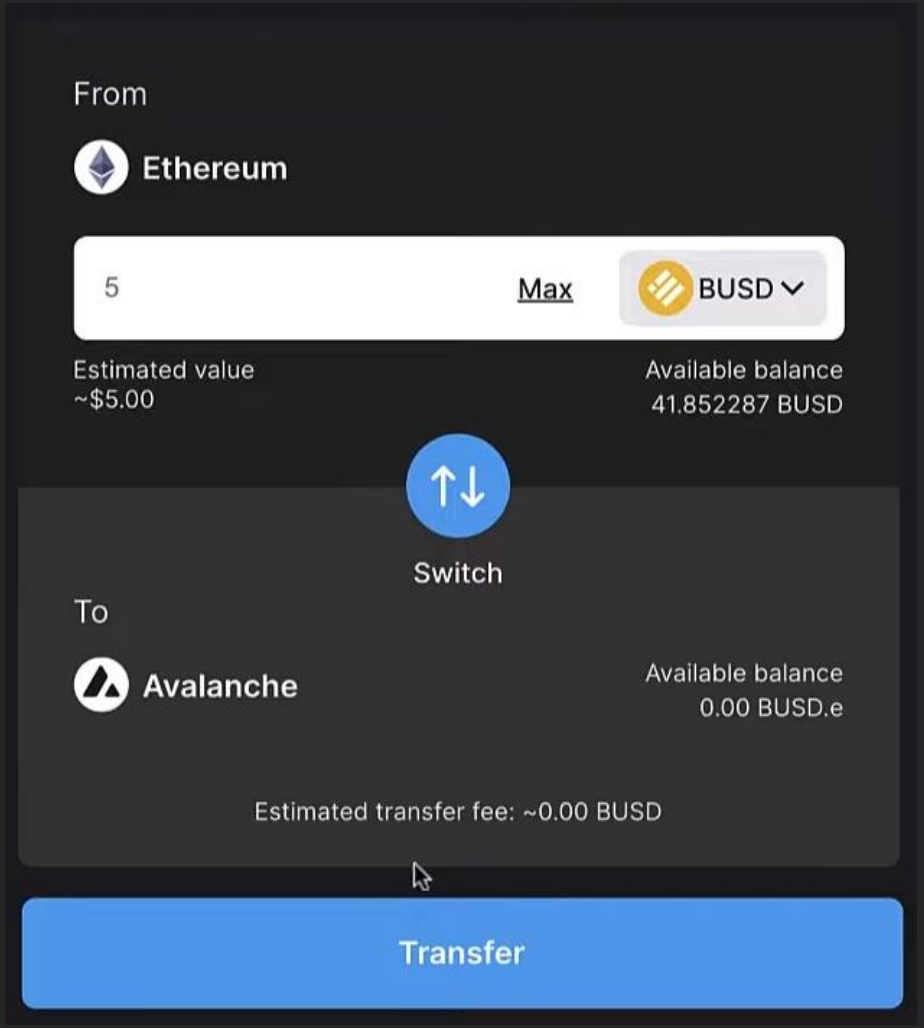
입출금 현황

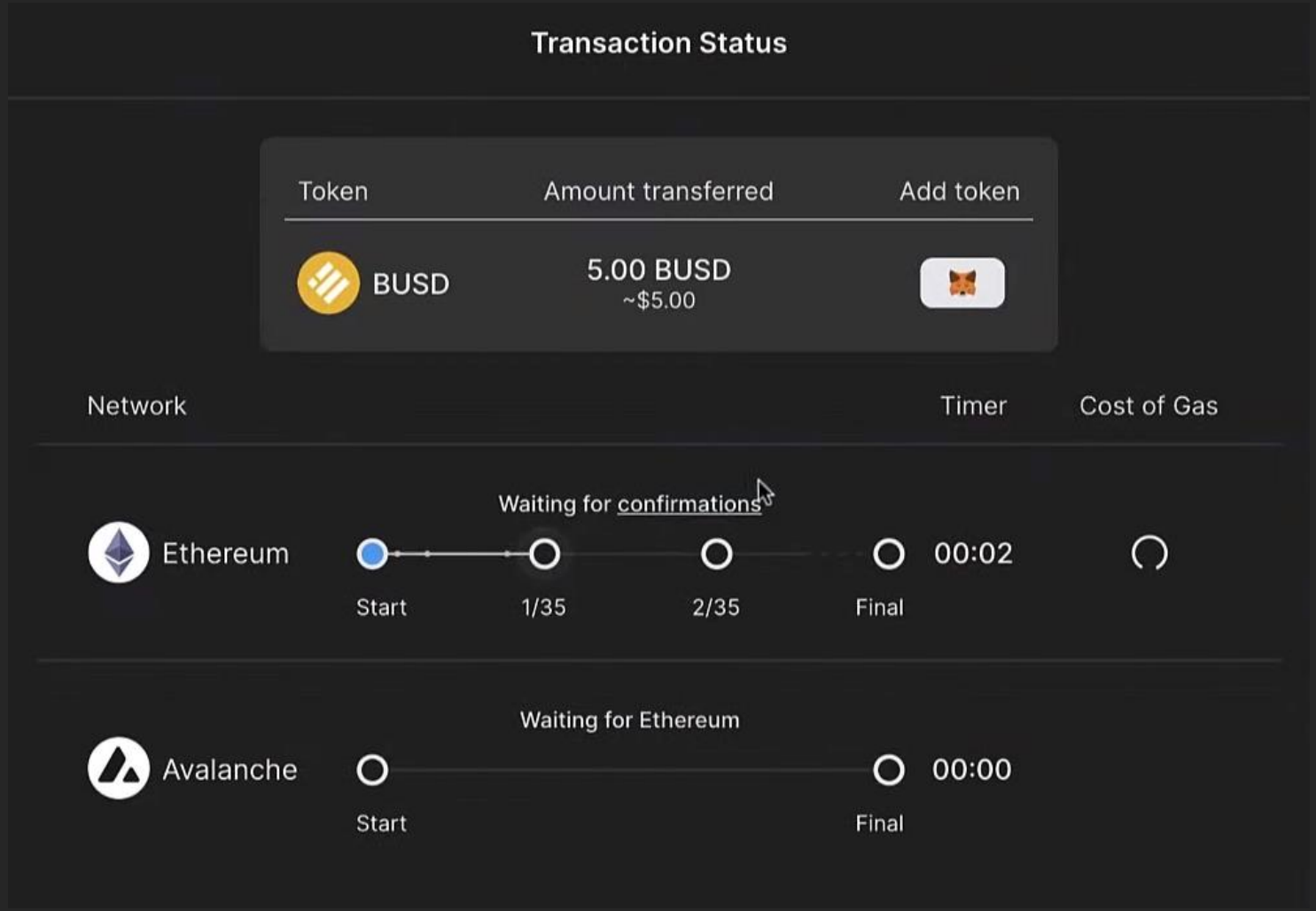
전체 정상 중단

자산	네트워크	입금 컨펌	입금	출금
비트코인(BTC)	Mainnet	1	정상	정상
이더리움(ETH)	Mainnet	30	정상	정상
라이트코인(LTC)	Mainnet	6	정상	정상
이더리움 클래식(ETC)	Mainnet	1000	정상	정상
리플(XRP)	Mainnet	1	정상	정상
비트코인 캐시(BCH)	Mainnet	60	정상	정상

Avalanche Bridge

블록체인간 Token Swap시 출발 블록체인의 Confirmations 대기





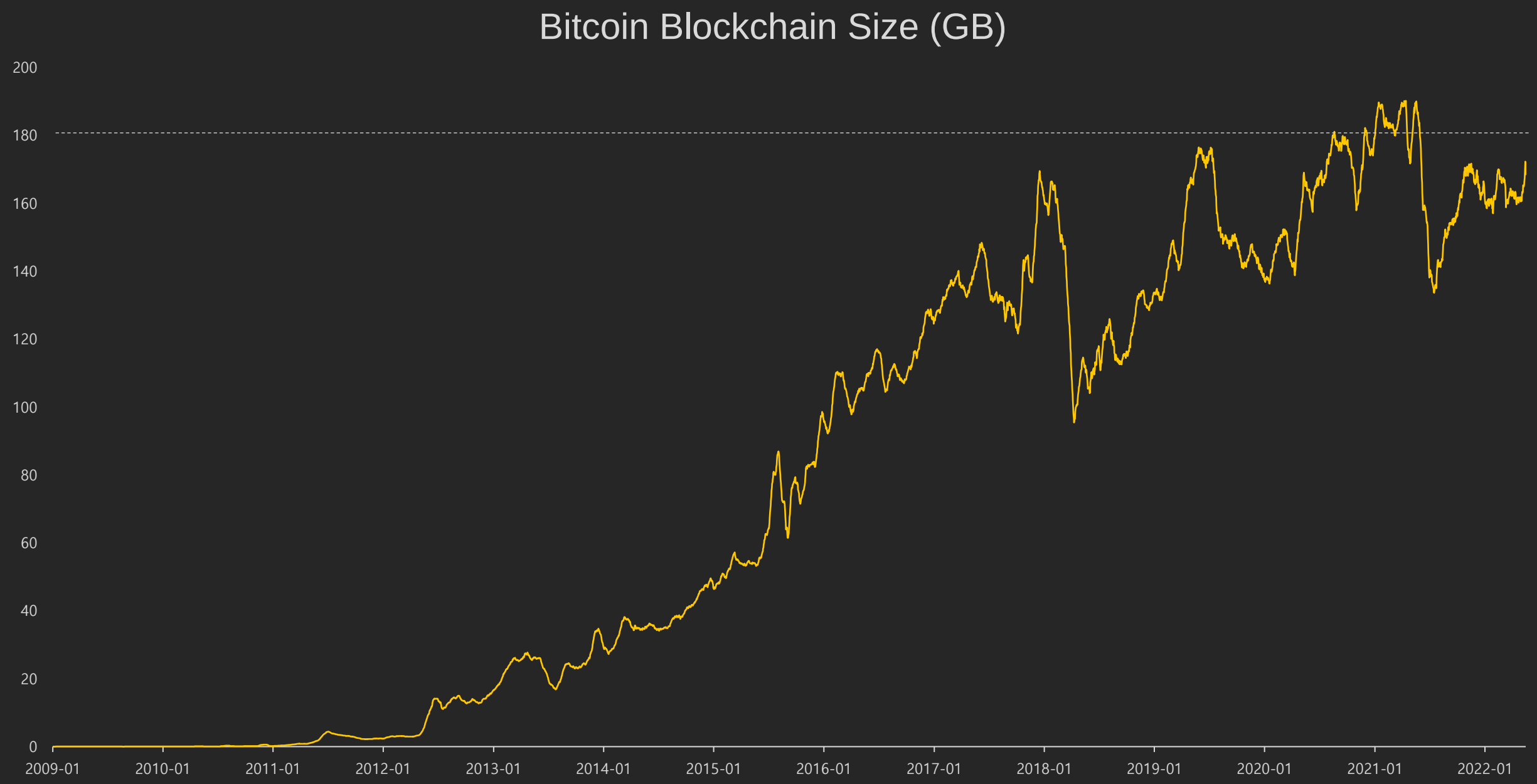
https://bridge.avax.network/

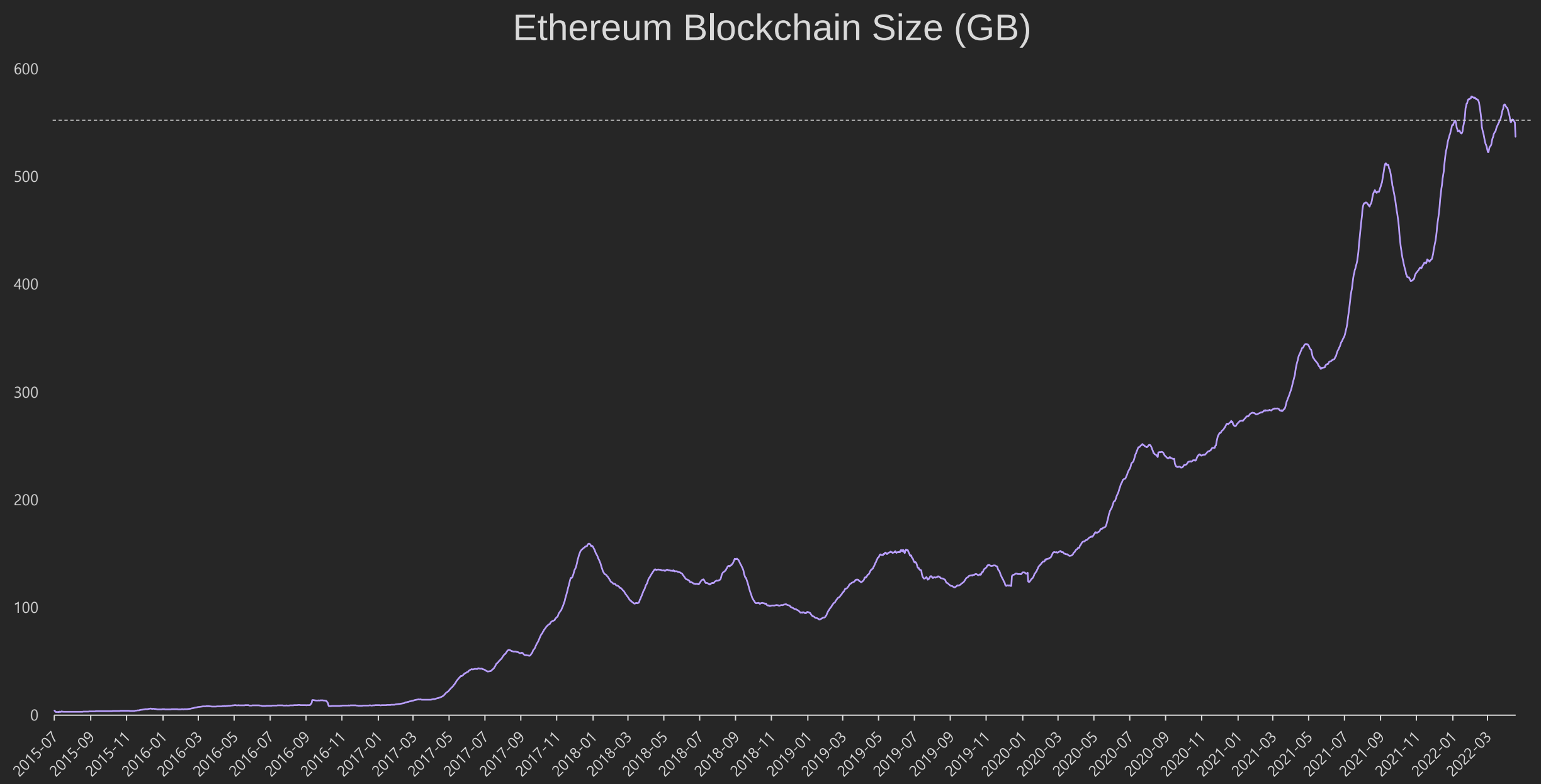
성능 한계

- P2P 네트워크에서 정보 확산 시간 제거 불가능
- 여러 노드간 합의에 필요한 시간 필요
- 성능 (응답 시간과 처리량)을 올리는 것은 어려움
  - 비트코인
    - 블록 생성 시간 : 약 10분
    - Body 최대 크기 : 1MB (Block: 4MB)
  - 이더리움
    - 블록 생성 시간 : 약 12 ~ 14초
    - 최대 블록 크기 : 30,000,000 Gas
- 실시간 처리 업무에 적합하지 않다.

블록체인 용량

- 참가자 전원이 트랜잭션 실행 결과를 검토해 신뢰성 확보
- 모든 블록 정보를 각 노드가 보유 필요
- 비트코인 : 약 170G, 이더리움 : 약 550G

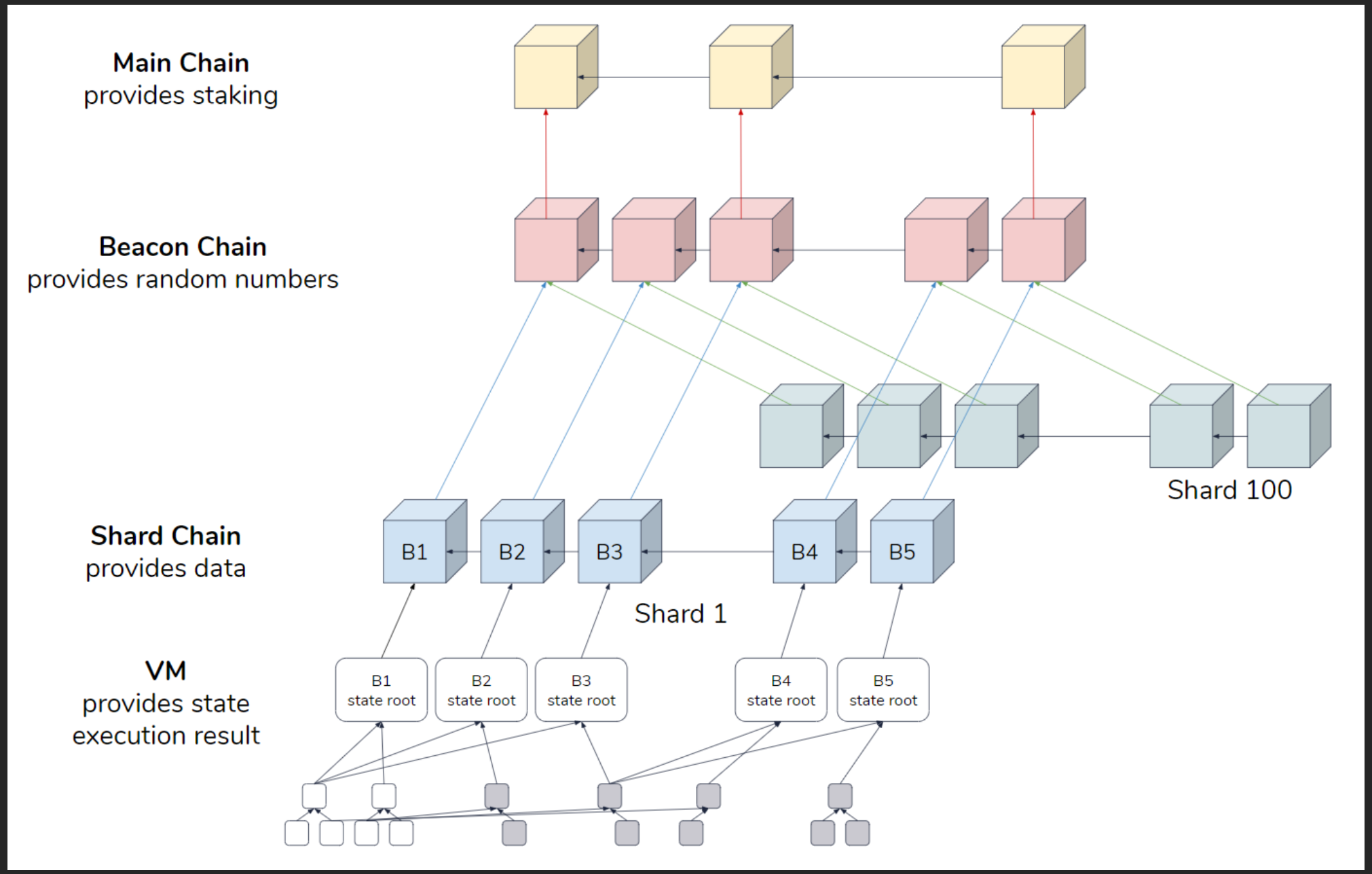

Bitcoin Blockchain Size (GB)
200
180
160
140
120
100
80
60
40
20
0
2009-01
2010-01
2011-01
2012-01
2013-01
2014-01
2015-01
2016-01
2017-01
2018-01
2019-01
2020-01
2021-01
2022-01


Ethereum Blockchain Size (GB)
600
500
400
300
200
100
0
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01
2017-03
2017-05
2017-07
2017-09
2017-11
2018-01
2018-03
2018-05
2018-07
2018-09
2018-11
2019-01
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07
2021-09
2021-11
2022-01
2022-03

PoS (Proof of Stake) : 지분 증명

- 블록체인의 지분인 코인이 많으면 합의를 도출할 기회가 더 많이 생기는 구조
- PoW 네트워크 장악을 위해서는 해시파워가 필요
- PoS 네트워크 장악을 위해서는 자금이 필요
- 자금
  - 비정상 노드가 침입하지 못하게 막는 방어벽
  - 지분이 있는 참가자는 나쁜 행위를 하여 가치를 떨어뜨리지 않을 것이라 기대
- 문제점, Nothing at stake
  - 서로 다른 브랜치가 존재하는 포크 상황에서 지분을 보유한 블록 생성자들이 모든 브랜치에 배팅하는 문제
  - 블록 생성자들이 어떤 브랜치가 메인체인에 포함되든 관계없이 보상을 얻으려고 할 때 발생하는 문제
    블록 생성이 완결되지 않고 브랜치가 계속해서 늘어남으로써 체인이 수십, 수백개 브랜치가 될 수도 있기 때문
    다중 투표 등 네트워크 참여자의 악의적인 행동에 벌을 주는 여러 규칙이 필수적으로 포함

합의 알고리즘 [PoS - Ethereum 2.0]



https://pmg.csail.mit.edu/papers/osdi99.pdf

BFT(Byzantine Fault Tolerance, 비잔티움 장애 허용)

- 다른 피어에 잘못된 데이터를 전파하려는 악의적인 노드의 시도에도 불구하고 네트워크가 합의에 도달할 수 있는 능력

FLP Impossibility

- 어떤 합의 알고리즘이 네트워크에서 통용되기 위해선 Safety와 Liveness 특성을 가져야 함
- Safety : 노드 간 합의가 발생했다면, 어느 노드가 접근하든 그 값은 동일해야 한다. (블록체인: Finality, 같은 높이의 블록 미발생)
- Liveness: 합의대상에 문제가 없다면, 네트워크 내에서 반드시 합의가 이루어진다.
  (블록체인: 제출된 Transaction은 네트워크의 모든 노드의 Blockchain에 추가)
- 비동기 네트워크 내에서 Safety와 Liveness를 모두 완벽히 만족하는 합의 알고리즘 설계는 불가능
  Impossibility of Distributed Consensus with One Faulty Process
- Bitcoin 이전: Safety 우선 > Liveness 손실 최소화 ➡ Bitcoin : Liveness 극대화 > Safety 보완: Fork 발생 시 Longest Chain Selection)

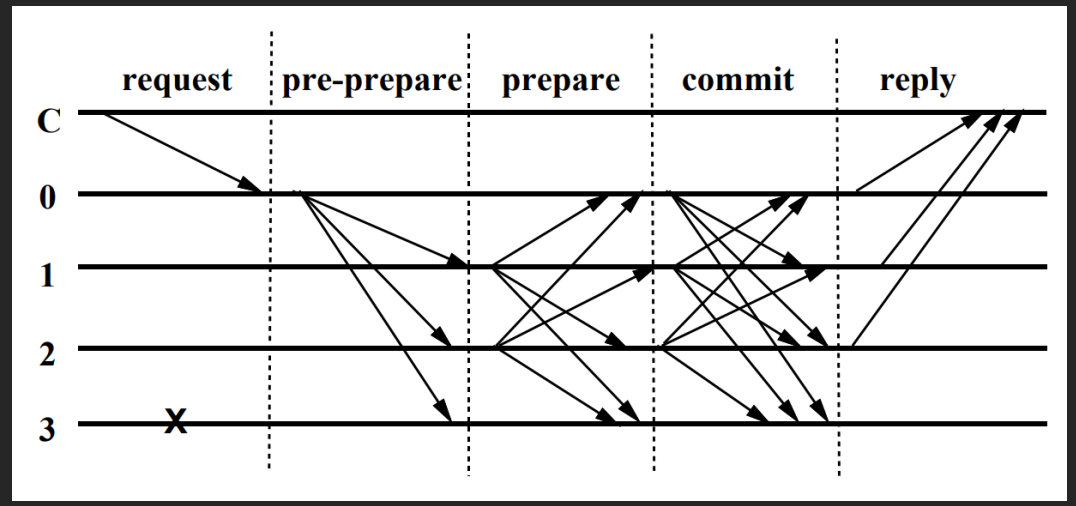
비동기 네트워크

- 동기 네트워크: 노드가 메시지를 보내는 시간과 수신 노드가 수신하는 시간 사이에 상한선 존재하고, 그 시간 안에 반드시 메시지가 수신 노드에게 도착한다는 것을 보증하는 네트워크
- 비동기 네트워크 : 노드가 메시지를 보내는 시간과 수신 노드가 수신하는 시간 사이에 상한선 미존재 (인터넷, 블록체인)
- 부분 동기 네트워크 : 노드가 메시지를 보내는 시간과 수신 노드가 메시지를 수신하는 시간 사이에 상한 선이 존재하지만, 그 상한선이 얼마인지는 알 수 없는 네트워크

https://pmg.csail.mit.edu/papers/osdi99.pdf

PBFT (Practical Byzantine Fault Tolerance, 실용적 비잔틴 장애 허용)

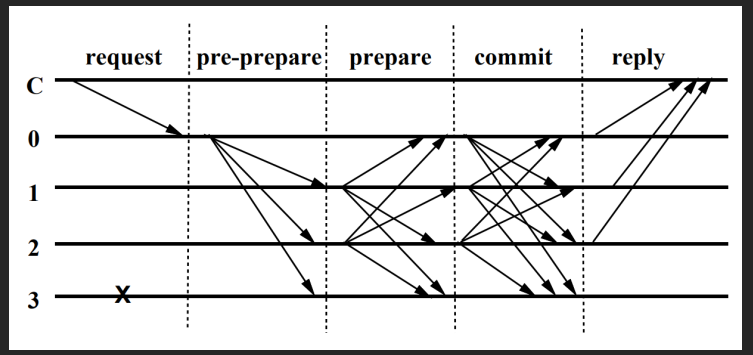
- Miguel Castro와 Barbara Liskov가 1999년 제안
- 네트워크의 모든 참여자를 미리 알고 있어야 한다.
- 부정한 노드가 f개 일 때, 전체 노드 수는 3f+1개, 확정에 f + 1개 노드 필요
  수학적으로 증명, Leslie B. Lamport , The Byzantine Generals Problem



C : Client
0 : Primary
3 : faulty

f (부정한 노드 수)	3f + 1 (최소 노드 수)	2f (회신에 필요한 수신 명령 수)	f + 1 (확정에 필요한 노드 수)
1	4	2	2
2	7	4	3
3	10	6	4
4	13	8	5
5	16	10	6

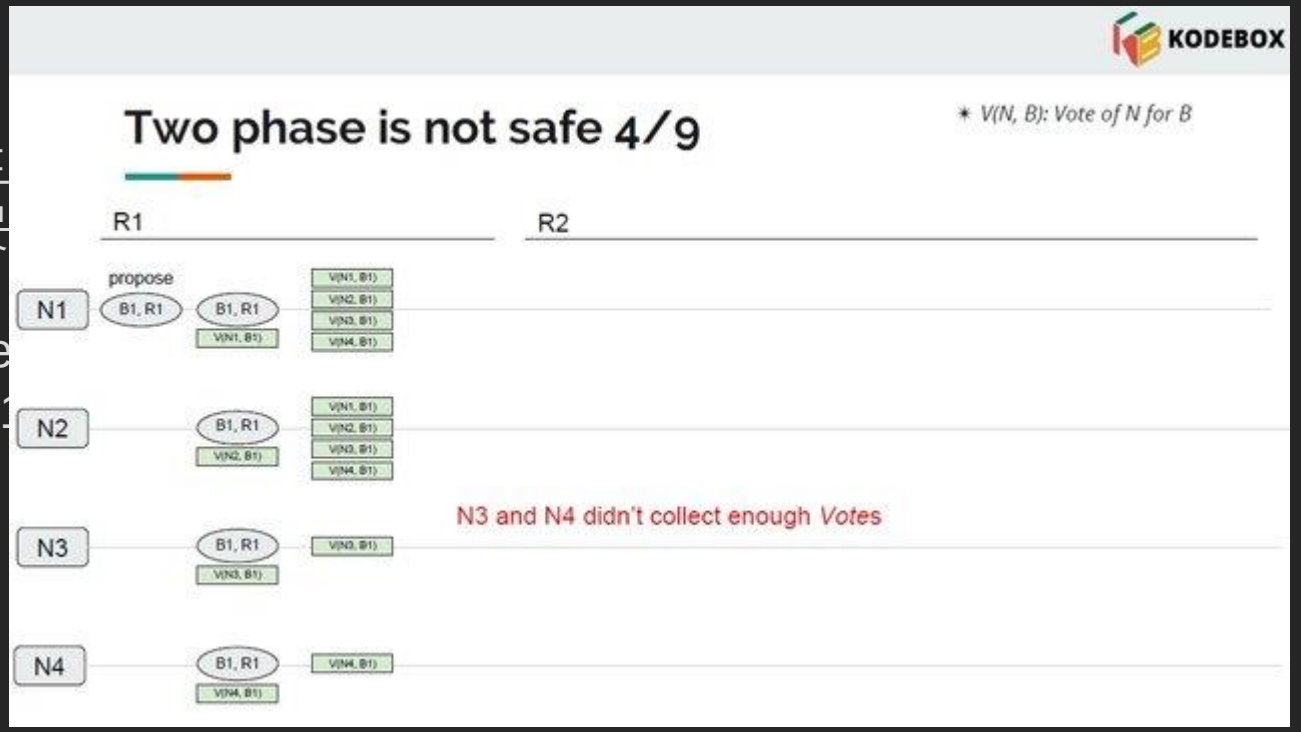
1. **request** : client가 Request 메시지 m을 임의 노드에 전송, Request 받은 노드는 Primary(리더) 노드, 나머지는 Backup Node
2. **pre-request** : Sequential number "N" 생성, 네트워크의 나머지 노드에 Pre-prepare 메시지<Pre-prepare, V, N, D(m)> 전송
    1. V: 메시지가 전송되는 View (View in which the message is being sent)
    2. N: Sequential umber
    3. D(m): 메시지 m의 요약본



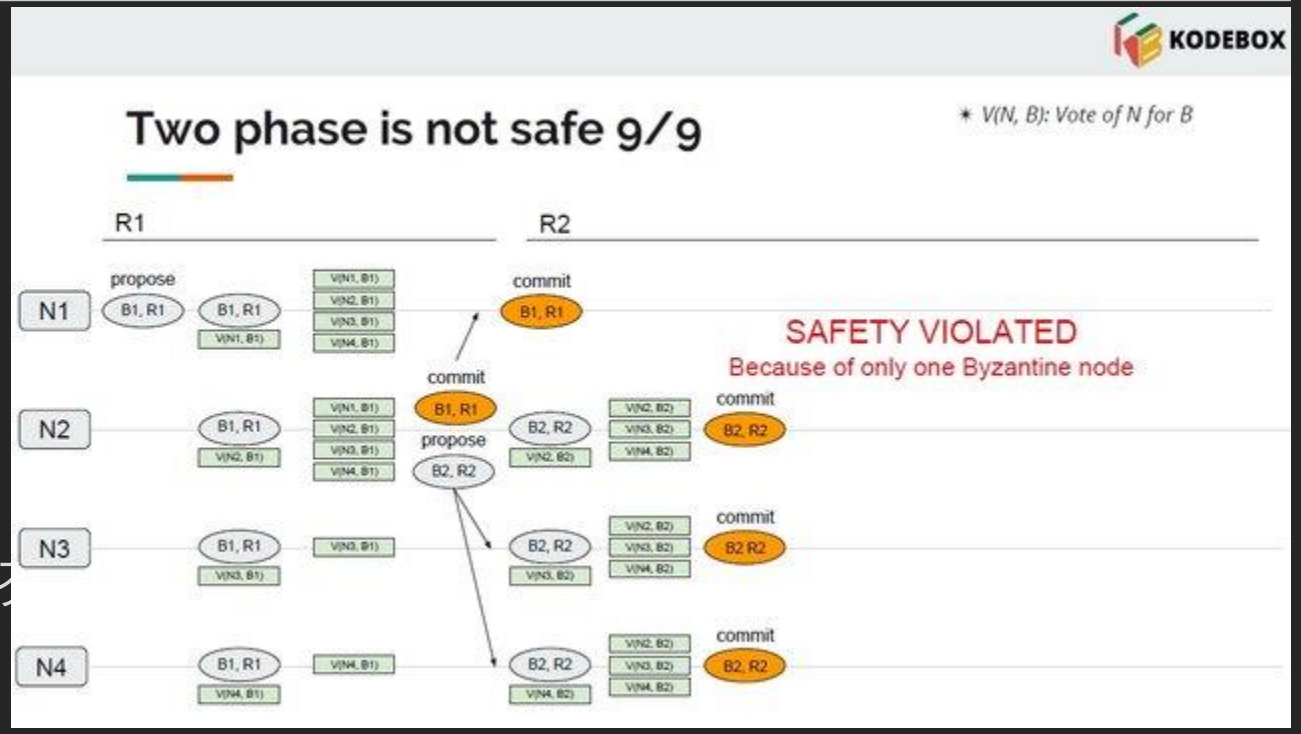
3. **prepare**: Pre-prepare 메시지를 받은 Backup i 노드는 D(m), V, N 값을 검증하고,
   유효하면 Prepare 메시지<Prepare, V, N, D(m), i>를 모든 노드에 전송
4. 각 노드는 Pre-prepare 메시지와 Prepare 메시지 수집
   prepared certificate : Pre-prepared 메시지 개수 2f + 1 이상, Prepare 메시지 2f개 이상 ➞ 노드: "prepared the request" 상태
5. **commit** : "prepared certificate" 조건을 만족한 노드는 모든 노드에게 Commit 메시지<Commit, V, N, i> 전송
6. 각 노드는 Commit 메시지를 수집. Commit 메시지가 2f+1개가 모이면 노드는 "**commit certificate**" 상태가 됨
7. "prepared certificate"와 "commit certificate" 두 가지 모두 있을 경우 해당 노드는 "**committed certificate**"
   클라이언트가 요청한 request를 수용해 상태 변화 함수를 실행
   즉, 네트워크의 합의 알고리즘이 작동하여 합의를 도출
   Request를 수용한 경우 네트워크 어느 노드에서도 상태 변화를 수용했기 때문에 Safety를 충족
   단, 합의 요건을 충족하지 못한 request는 기각하기 때문에 Liveness를 일부 희생

- 두 번의 절차를 걸쳐서 합의해야하는 이유
- 비동기 네트워크에서 한 번의 절차로만 합의를 시행할 경우, 배신자 노드가 합의 알고리즘의 Safety를 파괴할 수 있음
- 4개 노드가 존재하는 네트워크
  - 노드 1이 블록 B1을 생성하고 네트워크에 전파
    노드1과 노드2는 해당 블록을 검증한 결과를 노드
    노드3과 노드4는 나머지 노드들의 검증을 받지 못

    PBFT 알고리즘에서 노드3과 노드4는 prepared ce
    commit 메시지가 과반수에 미치지 못하므로 블록



- certificate 한 번 만으로 commit을 할 수 있다고 가정
- 노드2개 배신자 노드인 경우

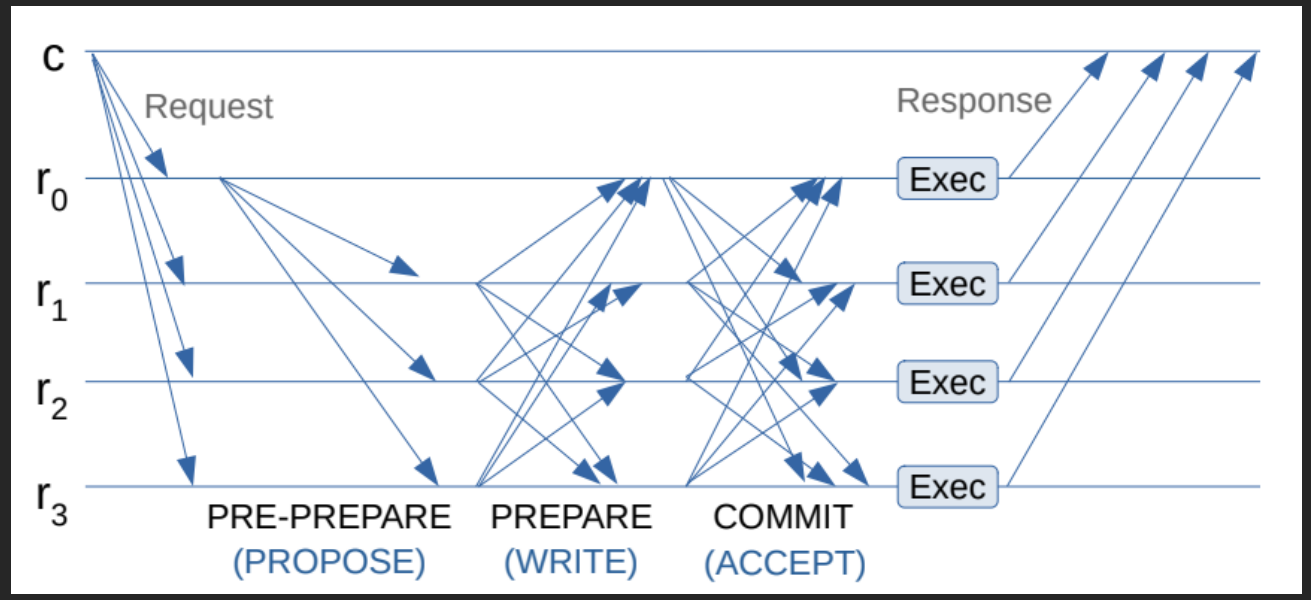


- 노드1에게는 블록1을 Commit
노드1는 R2단계에서의 블록은 블록1이라 간주
- 합의에 도달할 투표수가 부족한 노드3, 노드4에게는 새로운 블록2를 제안
노드3과 4는 블록1이 유효성 검증을 받지 못해 탈락한 것
노드2와 같이 블록2의 유효성 검증을 진행
노드 2,3,4 모두 블록 2의 유효성 검증을 마치고 certificate를 얻었기에 블록2를 Commit
노드 3,4는 R2 단계의 블록을 블록2라 간주
- 이 결과 R2 단계에서 Commit 인증을 받은 블록은 블록1과 블록2 두개
PBFT시스템의 최대 장점은 배신자가 발생할 수 있는 비동기 네트워크에서 Safety를 보장하는 것
- 투표 절차가 한 번인 경우, 한 명의 배신자 노드가 네트워크 합의 시스템의 Safety를 무력화
- 따라서 prepared certificate와 commit certificate라는 두 번의 절차를 활용하여 배신자 노드가 존재하는 상황에서도 네트워크 합의 도출

- BFT 3f+1인 이유
- https://velog.io/@dik654/BFT-3f1%EC%9D%B8-%EC%9D%B4%EC%9C%A0
- 어떤 노드가 byzantine failure일 때 발생할 수 있는 문제 상황은 크게 두 가지다. 첫 번째 문제 상황은 메시지를 보내지 않는 것이다. 즉, 분산 시스템이 N개의 노드 중 byzantine failure f개가 정상적으로 동작하기 위해서는 N - f개의 메시지만으로 합의가 이루어져야 한다. 이것을 다른 말로 quorum을 위해서 N - f개의 노드가 필요하다고 말한다.

- 두 번째 문제 상황은 byzantine failure인 노드가 악의적으로 다른 메시지를 보내는 것이다. 극단적으로는 quorum을 이룬 N - f개의 노드 중에서 f개가 byzantine failure가 보낸 메시지인 경우를 생각해볼 수 있다. 이 경우도 정상 동작해야 하므로 (N - f) - f 개의 메시지가 byzantine failure인 노드가 보낸 f개의 메시지보다 많아야 한다.

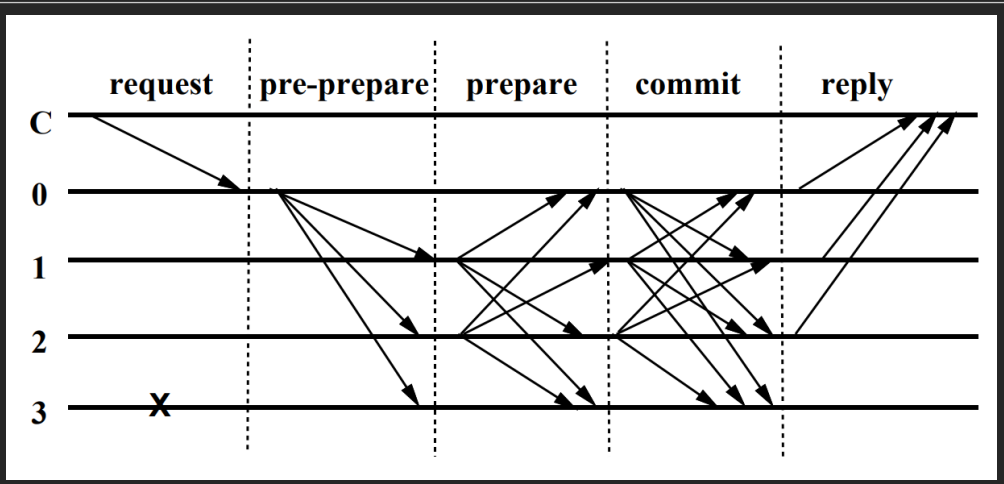
- 위의 두 경우를 대비하기 위해서 (N - f) - f > f. 즉, N > 3f이므로 f개의 byzantine failure인 노드가 있을 때 최소 3f개보다 많은 노드가 있어야 byzantine fault tolerance한 시스템이고, 이때 가장 작은 N은 3f + 1이다. 따라서 3f + 1개의 노드로 구성된 시스템에서 견딜 수 있는 faulty 노드의 최대 수는 f다.



f (부정한 노드 수)	3f + 1 (최소 노드 수)	2f (회신에 필요한 수신 명령 수)	f + 1 (확정에 필요한 노드 수)
1	4	2	2
2	7	4	3
3	10	6	4
4	13	8	5
5	16	10	6
6	19	12	7

1. Request : client 가 모든 노드(replica)에 요청(Request)을 전파(Broadcast)
2. Pre-Prepare : 임의의 replica0이 Primary(리더)가 되고 순차적으로 명령을 다른 노드에 전달
3. Prepare : 각 노드는 2의 명령을 받으면 Primary(replica0)을 포함한 모든 노드에 회신
4. Commit : 각 노드는 3에서 전달된 명령을 일정 수 이상(2f) 수신하면 primary를 포함한 모든 노드에 수신한 신호를 전송
5. 각 노드는 4에서 보낸 명령을 일정 수 이상(2f) 수신하면 명령을 실행하고 블록을 등록해 client에 reply를 반환

합의 알고리즘 - PBFT(Practical Byzantine Fault Tolerance)



- 클라이언트가 모든 노드에 요청을 브로드 캐스트
- Replica0이 primary(리더)가 되고 순차적으로 명령을 다른 노드에 전달
- 각 노드는 2의 명령을 받으면 primary(Replica0)을 포함한 모든 노드에 회신
- 각 노드는 3에서 전달된 명령을 일정 수 이상(2f) 수신하면 Primary(Replica0)을 포함한 모든 노드에 수신한 신호를 전송
- 각 노드는 4에서 보낸 명령을 일정 수 이상(2f) 수신하면 명령을 실행하고 블록을 등록해 client에 reply를 반환

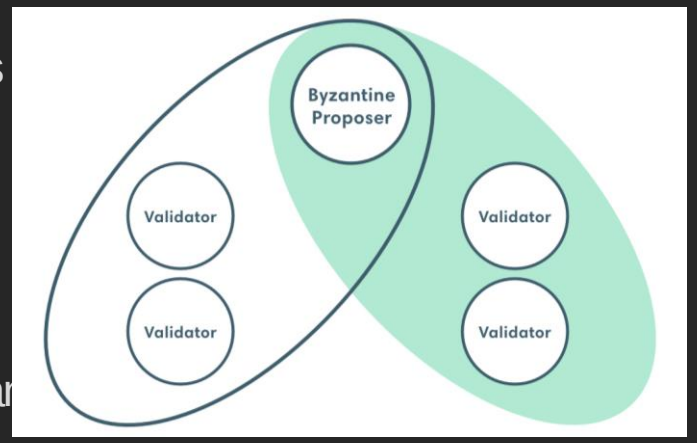
1. 2/3 이상의 노드가 합의하면 검증되기 때문에 속도 향상
2. 노드 수가 증가하면 복잡도가 증가해 확장성 확보가 어렵고, 합의 도출도 어렵다.
3. 노드 수가 작아야 한다. -> 장점(보안에 유리), 단점(누구나 접근할 수 없다.)

- 다수결로 의사결정한 뒤 블록을 만들기 때문에 블록체인의 분기 미발생
- 생성된 블록은 1Confirmation으로 Finality 확보
- PoW의 퍼즐 문제가 없기 때문에 고속으로 동작

- Tendermint, IBFT

IBFT(Istanbul Byzantium Fault Tolerant) - EIP-650

- 블록체인에서 동작하는 PBFT 구현
- 특징
  - 즉각적인 완결성(Finality) 보장 : Fork, 체인 재구성 및 트랜잭션 취소 없음
  - 블록간 시간 단축: 많은 트랜잭션 처리량 증가
    블록 구성 및 검증 비용 감소 (PoW 같은 반복적 계산 없음)
  - 높은 데이터 무결성 및 내결함성
    검증자 그룹(Validator Set)을 사용하여 제안된 블록의 무결성 보장(66% 이상의 서명 필요)
    '리더십'도 시간에 따라 돌아가면 변경하여 결함 있는 노드의 영향을 제한
    투표를 통해서 리더 퇴출 가능
  - 동적 Validator Set (PBFT: 정적 Validator Set) : 수정 가능하며, 완전히 신뢰할 수 있는 노드만 포함
- Issues: Correctness Analysis of IBFT
  - Safety issue : two correct processes could decide different values
  - Liveness issue : some executions could lead to a deadlock
    in the presence of just one Byzantine node
    an IBFT 1.0 network
    can reach a state where
    nodes are locked on different blocks and
    no agreement can be reached on any of these blocks which mean



5 Nodes
3 node for block
1 Proposer
2 Validator Set
2 Blocks generate
with same height
to the chain.

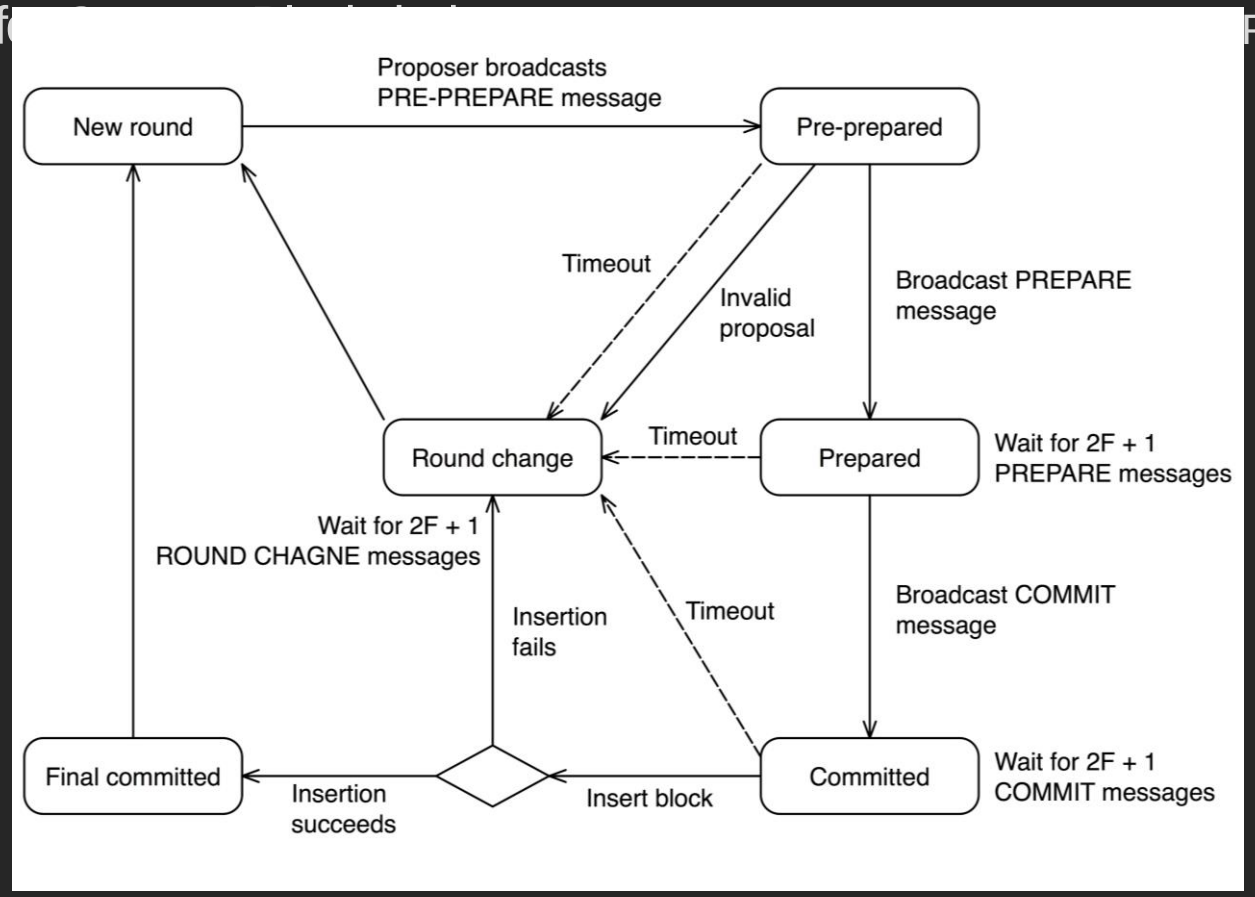
Algorithm	IBFT	PBFT
client	all validators	specific clients
validator list	dynamic	static
epoch	block/round	view
fault-tolerance degress	F = (N-1)/3 (F= faulty nodes, N = validator nodes)	F = (N-1)/3 (F= faulty nodes, N = validator nodes)
proposer selection	1. round robin (in every block and round change) 2. sticky proposer (only when a round change happens)	each view
result	block generation	read/write operations

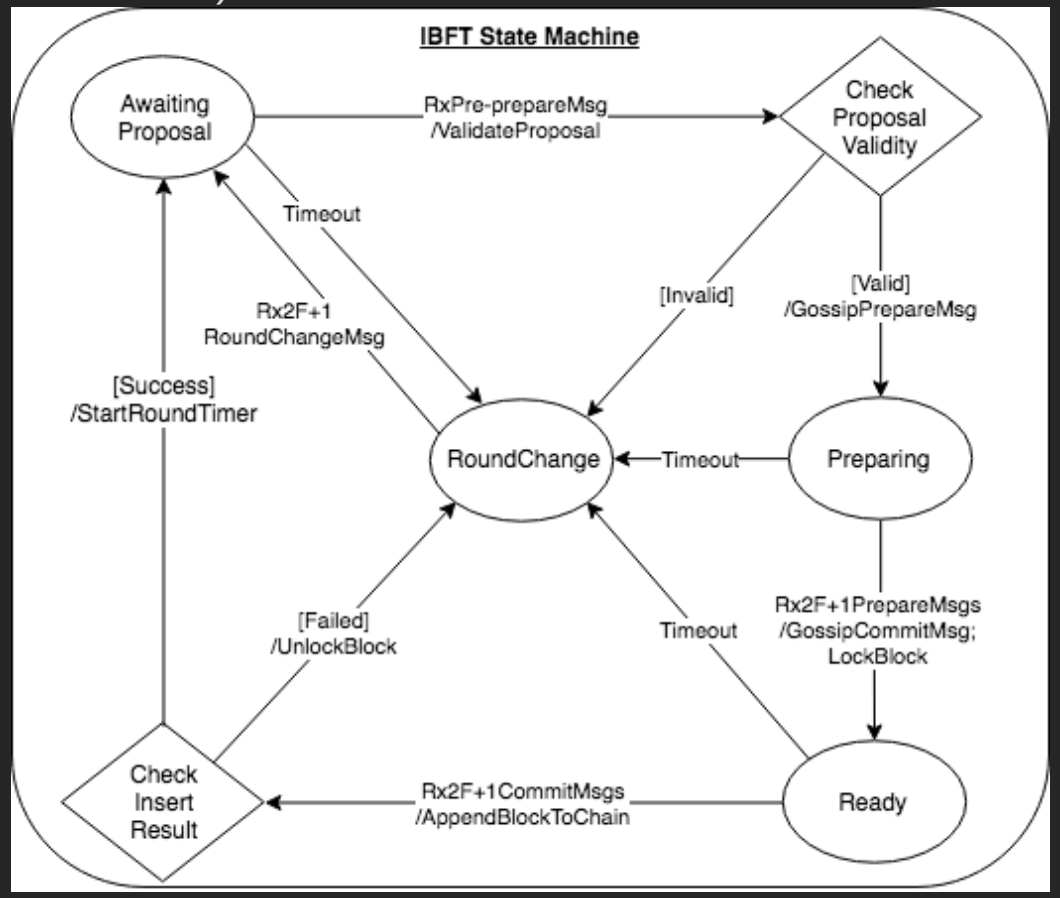
IBFT 2.0: https://arxiv.org/pdf/1909.10194v1.pdf

IBFT 2.0 - IBFT 1.0 개선 버전

- PoA(Proof of Authority) BFT 합의 알고리즘

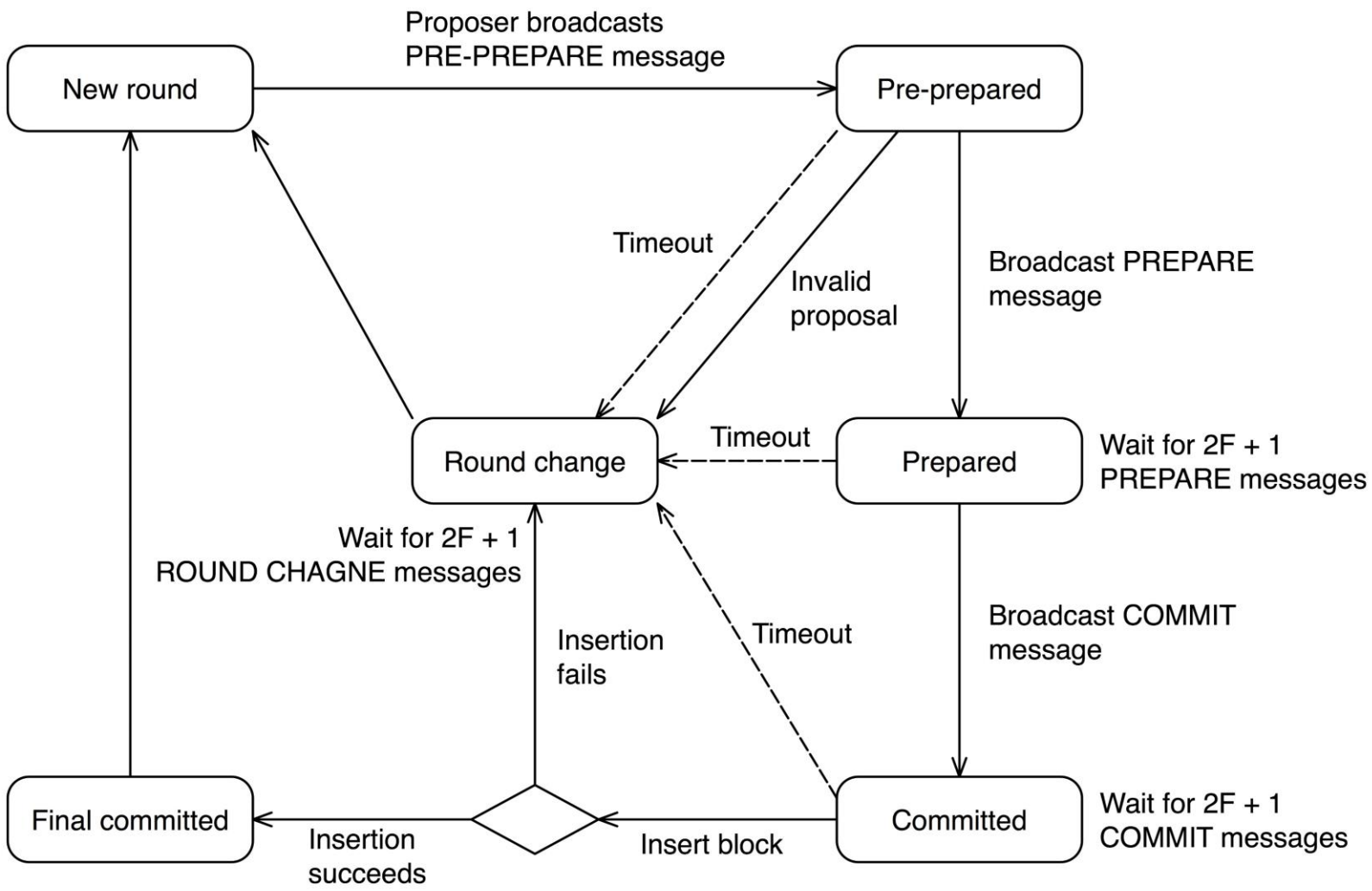
f Permissioned)



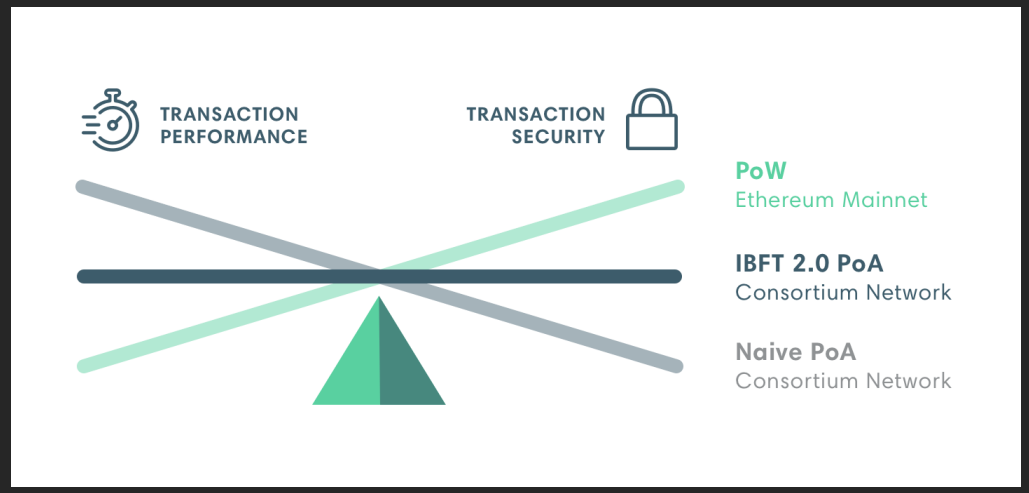


• 대다수 수정은 쿼럼에 도달하는 데 필요한 노드 수를 변경합니다. 이 변경 사항은 비잔틴 노드가 과반수를 요구하여 두 가지 별개의 검증자 세트와 합의에 도달할 수 있는 문제를 해결합니다. 예를 들어, 5명의 검증자가 있는 경우 IBFT 2.0은 블록체인에 추가할 블록에 대한 블록에 대한 합의에 도달하기 위해 3명이 아닌 4명의 검증자가 필요합니다. 이 변경으로 비잔틴 유효성 검사기는 두 세트의 정직한 유효성 검사기를 통해 서로 다른 블록에 대한 합의에 도달할 수 없습니다.

• 라운드 변경 수정은 IBFT 1.0의 활성 문제를 해결합니다. IBFT 1.0에서 라운드 변경은 항상 새로운 블록 제안으로 이어집니다. IBFT 2.0에서 향상된 라운드 변경 프로토콜은 유효성 검사기가 라운드에서 커밋하는 경우 연속 라운드에서 제안된 블록이 커밋된 블록과 일치하도록 합니다. PBFT 보기 변경 프로토콜에서 영감을 받은 이 메커니즘은 활성 문제를 일으킨 IBFT 1.0에서 사용된 블록 잠금 메커니즘의 필요성을 제거합니다.


Proposer broadcasts
PRE-PREPARE message
New round
Pre-prepared
Timeout
Invalid
proposal
Broadcast PREPARE
message
Round change
Timeout
Prepared
Wait for 2F + 1
PREPARE messages
Wait for 2F + 1
ROUND CHAGNE messages
Insertion
fails
Timeout
Broadcast COMMIT
message
Final committed
Insertion
succeeds
Insert block
Committed
Wait for 2F + 1
COMMIT messages

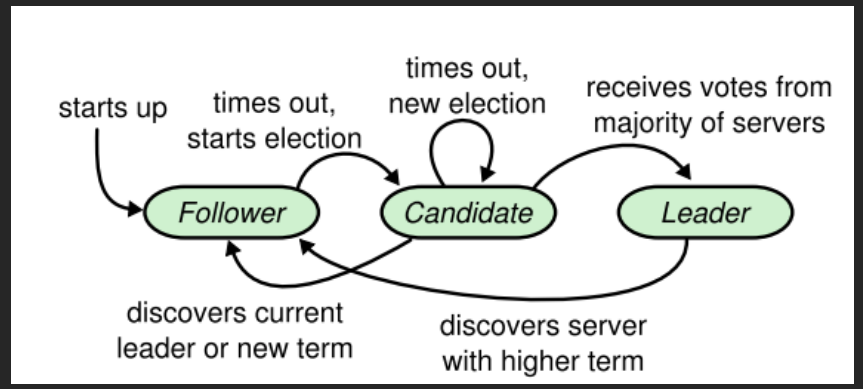
- *보안*이라는 용어는 악의적인 행위자를 견디고 의도한 대로 작동하는 합의 알고리즘의 능력
- 합의 알고리즘을 사용하면 트랜잭션 보안과 성능 간에 균형이 유지
- 블록체인의 성능은 일반적으로 트랜잭션 처리량으로 측정
- PoW를 사용하는 이더리움 메인넷은 뛰어난 보안성을 제공하지만 낮은 성능을 제공
- 초기 PoA 알고리즘을 사용하는 컨소시엄 블록체인 솔루션은 보안이 거의 또는 전혀 없는 고성능을 제공
- IBFT 2.0과 같은 향상된 권한 증명(PoA) 합의 알고리즘은 이러한 두 극단 사이에 위치하여 향상된 성능의 균형을 제공하는 동시에 악의적인 행위자로부터 보호합니다. Goldilocks 호출: 너무 느리고, 너무 불안정하고, 딱 맞습니다.

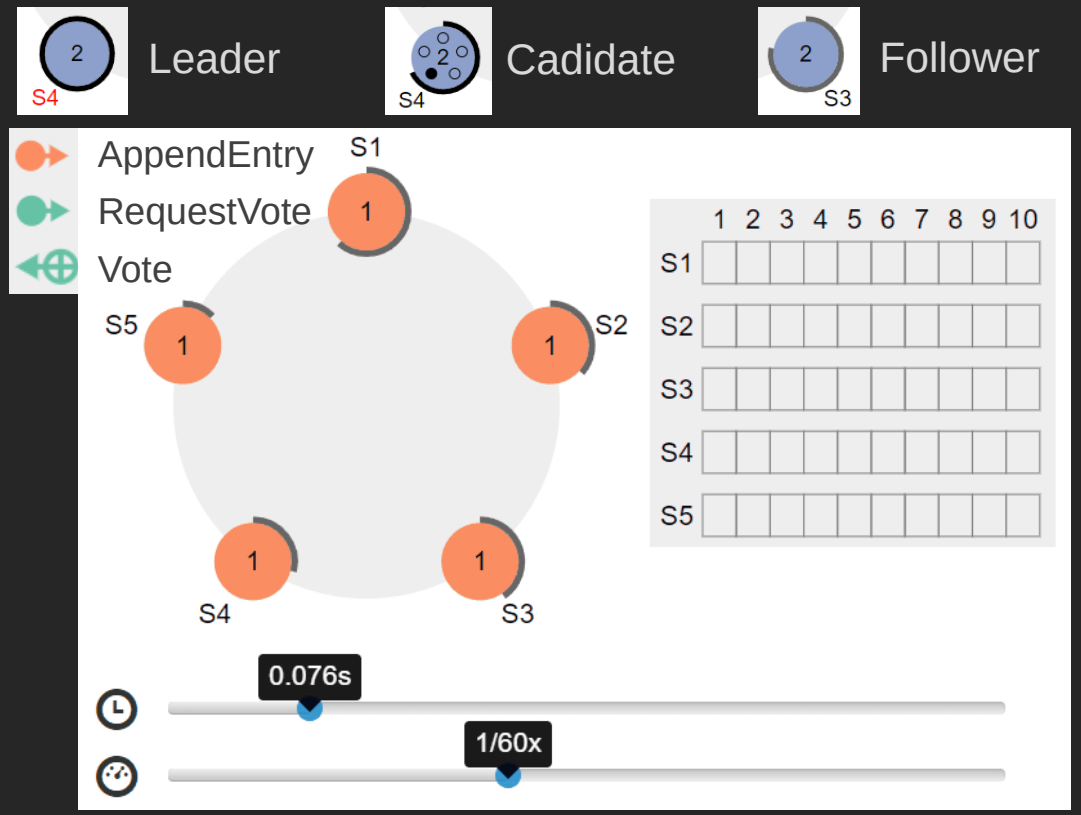


https://raft.github.io/

Raft Paper

- 신뢰 환경
- 서버는 리더(leader), 팔로워(follower), 후보자(candidate) 중 하나
- 팔로워는 리더의 요청에 따라 상태 업데이트
  변경 내역이 없어도 일정 시간마다 하트비트로 더미 요청 전송
- 리더 선출
  1. 팔로워가 일정 시간 요청을 받지 못하면 후보자로 변경
  2. Term 증가
  3. 다른 서버에 투표 요청 → **과반수** 투표를 받으면 리더
- Term number
  - 팔로워는 Term당 한 번만 투표
  - 높은 Term의 리더를 따름
- 클라이언트는 리더와 통신
  - 클라이언트가 팔로워를 호출하면 리더로 리다이렉트



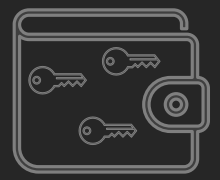


Wallet

개인 키 관리

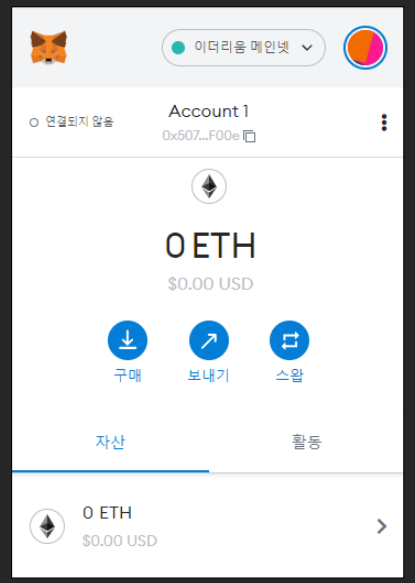
좁은 의미(개발자) 의 지갑

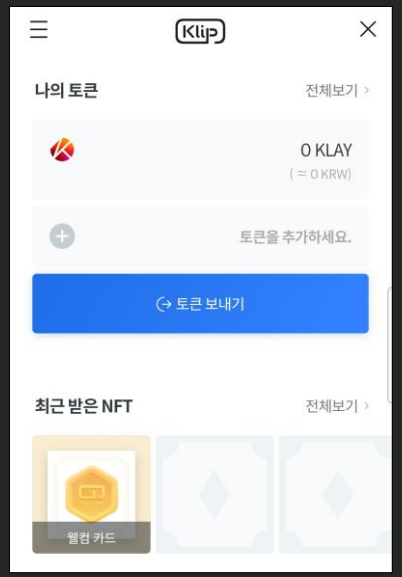
- 개인키 관리



넓은 의미의 지갑 (사용자 상호작용 애플리케이션)

- 개인키 관리 + 주소 생성
- 트랜잭션 서명 (Transaction Signing)
- Native Currency (BTC, ETH) 조회 및 전송
- Token(ERC20), NFT(ERC721) 토큰 같은 컨트랙트와 상호 작용





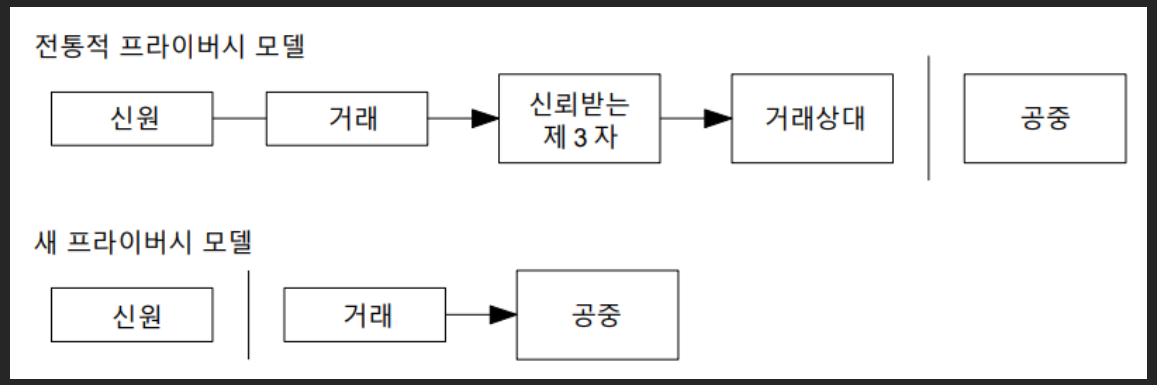
Private Key를 이용하여
트랜잭션 서명을 통해
블록체인 상 자산의 소유자임을 증명

Bockchain

A : 1000	B : 2000	C : 6000	D : 1000
E : 30000	F : 0	G : 5000	H : 2000
I : 1000	J : 4000	K : 7000	L : 11000
M : 53000	N : 93000	O : 35000	P : 6100
Q : 0	R : 9000	S : 152000	T : 73000

Bitcoin - 10. 프라이버시(Privacy)

전통적인 은행 모델은 참여 당사자(the parties involved)와 신뢰받는 제 3 자에게 정보 접근을 제한함으로써 일정 수준 프라이버시를 달성한다. 이 방법은 모든 거래를 공개할 필요성에 따라 배제되지만, 공개키 익명성을 보존해 다른 장소에서 정보의 흐름을 끊는 걸로 여전히 프라이버시가 보장될 수 있다. 공중(the public)은 누군가가 다른 누군가에게 보내는 금액을 볼 수 있지만, 그 거래에 연결된 누군가에 대한 정보는 볼 수 없다. 이는 증권거래소에서 공개되는 정보 수준과 비슷하게, 개별 거래 시각과 규모를 나타내는 "테이프(tape)"는 공개되지만, 그 거래 당사자가 누구인지 알지는 못하는 것이다.



부가적인 방책으로, 각 거래마다 새로운 키 쌍이 사용돼야 그게 어떤 공통된 소유자에게 연결되는 일을 계속 피할 수 있다. 여러 입금이 동일 소유자의 소유임을 부득이 드러내는 다중입금 거래에서 어떤 연결은 여전히 불가피하다. 그 거래의 키 소유자가 드러나면, 연결이 동일 소유자에게 속한 다른 거래까지 노출할 위험이 있다.

하나의 주소보다는 여러 개의 주소를 가지고 입출금 하는 것이 보다 안전

- 블록체인은 공개되어 있음 → 특정 주소에 대한 모든 거래 추적 가능
- 복수의 주소 사용은 하나의 주소 사용보다 감시(추적) 어려움 → 익명성에 도움
- 문제 : 복수개의 주소는 복수개의 키/주소를 저장 및 관리 필요 → 어려움

복수 개의 Key 관리를 위한 지갑

- HD 지갑 (BIP-32)
- 니모닉 코드 (BIP-39)
- HD 지갑 구조 (BIP-43 / BIP-44)

Private Key (256 bits)

0x8d46f2260091f89d63a73a2171234835d272e0cab9fb630b4306aaf72f22e830

0x11bd63b2fa9375ae72be1483e749e4fcfee12503ec06254d513ac2d6c442fa90

0x5c59ce0bdf6bafd44203ee201eb601cc80cc23d603dc1190dd11e836e25fb0bf

0xdb1a11a2fcb7f56801f44091269427eb1fe9ba977b8d5ed733a141187ec184c7

0x321f4747eae52197734cfc95b04c0717776bc73b279c506cc7d5c63b46aadbf3

0xd5d7a68e6e295cf780ec9addfbe60977866fb44c39752fbb3b24a0f7b0e1f444

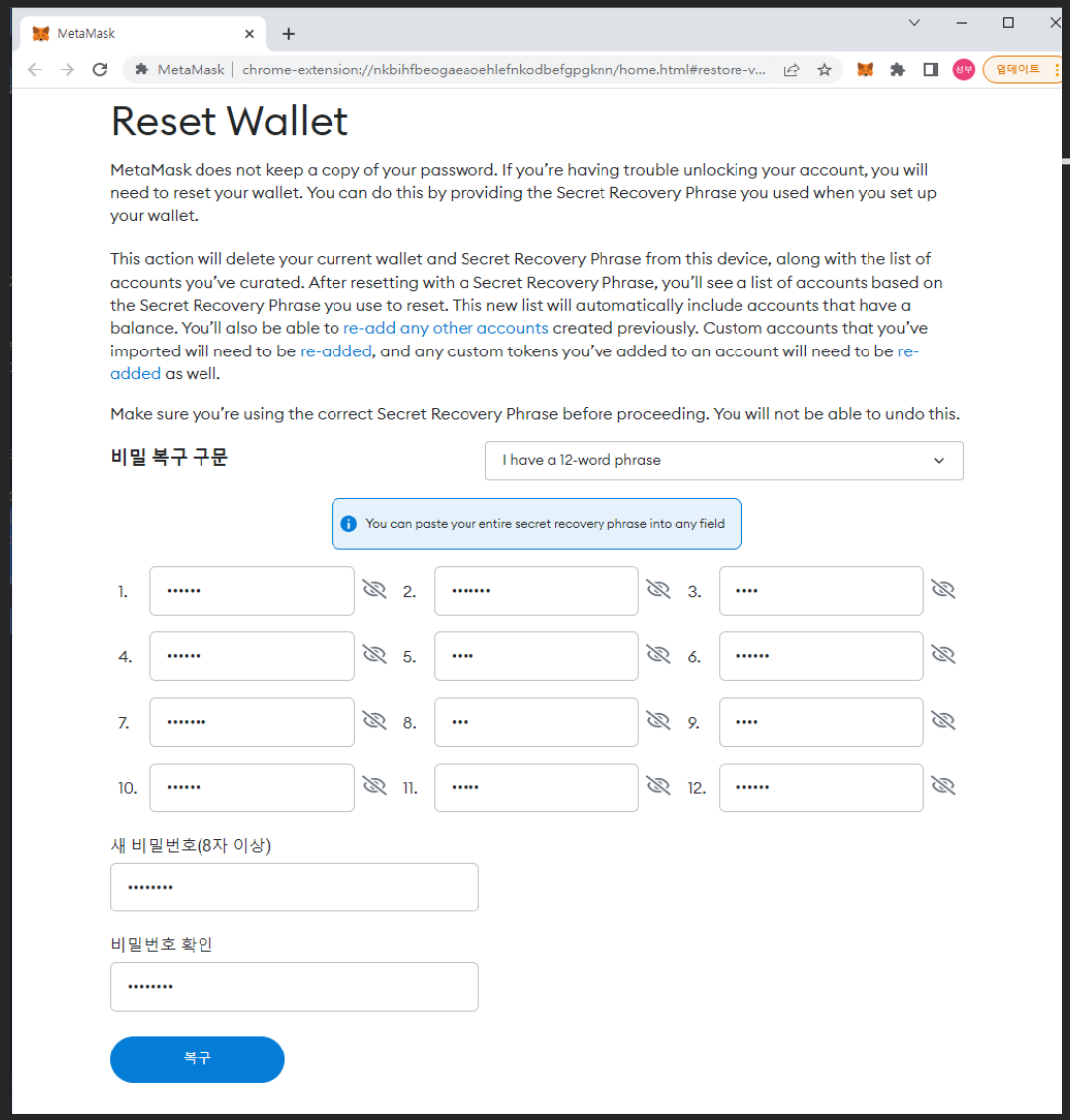
0xe060d3d30306df031bb6646c76a41effd63b5a9e80a8557794a302a85940b10f

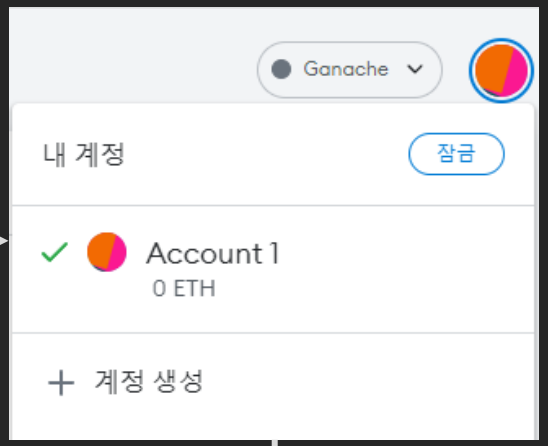
0x45ea754b57f4ce3cb27f1b4808c796b5fb0ccf35bbacc12055d2db6ee15b1ee4

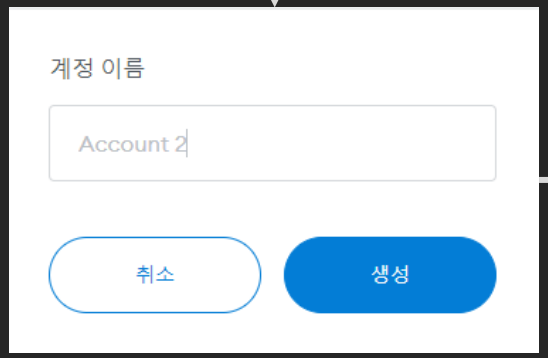
0xc87eba80c969f382d2161bbec57d36ca28387aa5305de1611e4c662535eaebae

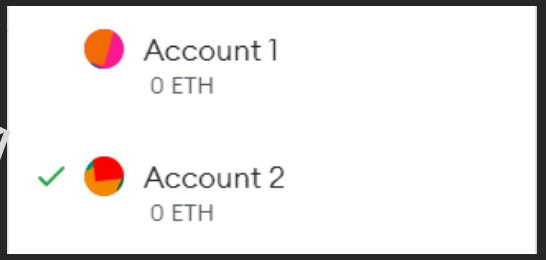
0xdec5419b6d72c398ad0a379645d6b9081f75090261daddb1ec27564b6e51338f

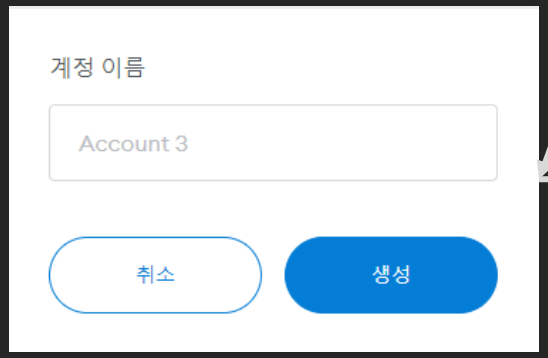
지갑 [MetaMask Reset Wallet]

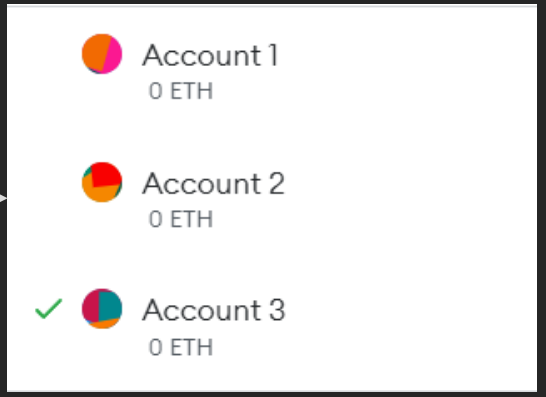


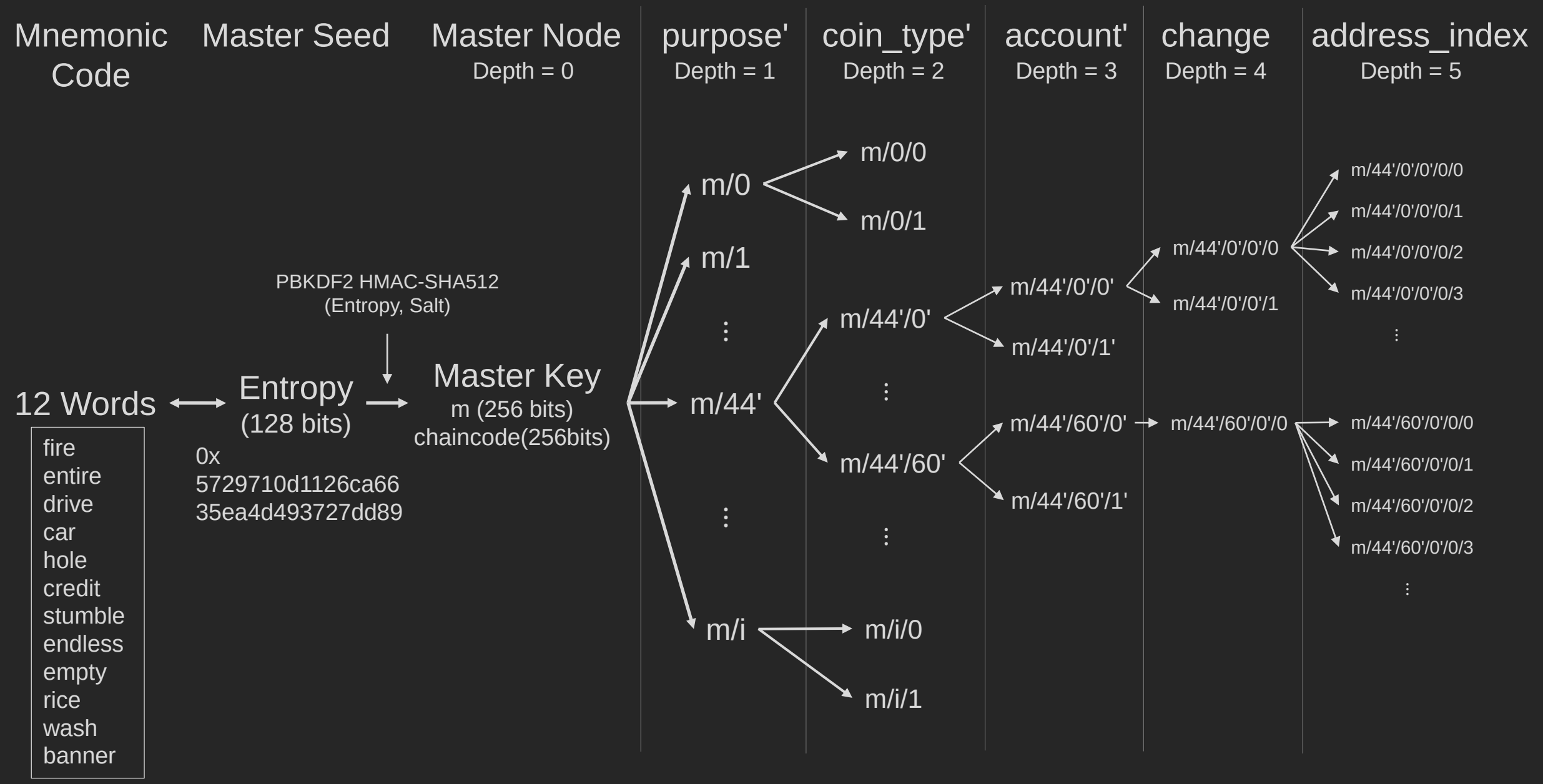











Mnemonic Code
Master Seed
Master Node
Depth = 0
purpose'
Depth = 1
coin_type'
Depth = 2
account'
Depth = 3
change
Depth = 4
address_index
Depth = 5
12 Words
fire
entire
drive
car
hole
credit
stumble
endless
empty
rice
wash
banner
Entropy
(128 bits)
0x
5729710d1126ca66
35ea4d493727dd89
PBKDF2 HMAC-SHA512
(Entropy, Salt)
Master Key
m (256 bits)
chaincode(256bits)
m/0
m/1
m/44'
m/i
m/0/0
m/0/1
m/44'/0'
m/44'/60'
m/i/0
m/i/1
m/44'/0'/0'
m/44'/0'/1'
m/44'/60'/0'
m/44'/60'/1'
m/44'/0'/0'/0
m/44'/0'/0'/1
m/44'/60'/0'/0
m/44'/0'/0'/0/0
m/44'/0'/0'/0/1
m/44'/0'/0'/0/2
m/44'/0'/0'/0/3
m/44'/60'/0'/0/0
m/44'/60'/0'/0/1
m/44'/60'/0'/0/2
m/44'/60'/0'/0/3

HD 지갑 (Hierarchical Deterministic Wallets)

BIP-32, https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

- Entropy(Master Seed) 로 부터 키 트리 파생
- 개인키 없이 공개키로 자식 공개키 파생 가능

니모닉 코드 (Mnemonic code for generating deterministic keys)

BIP-39, https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki

- 지갑(HD Wallet Seed)을 복구하기 위한 12개의 단어

다목적 HD 지갑 구조(Purpose Field for Deterministic Wallets)

BIP-43, https://github.com/bitcoin/bips/blob/master/bip-0043.mediawiki

- 강화된 첫 번째 자식 인덱스를 트리 구조의 '목적(purpose)'을 나타내는 식별자로 사용

복수화폐(multi currency) 및 복수계정(multi account) 지갑 (Multi-Account Hierarchy for Deterministic Wallets)

BIP 44 - https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki

- '목적' 번호를 44'로 설정하여 복수 화폐, 복수 계정 구조 제안
- 미리 정의된 다섯 가지 트리 레벨로 구성된 구조를 지정 (m / purpose' / coin_type / account' / change / address_index)

니모닉 코드 생성 (Mnemonic code for generating deterministic keys) [BIP-39]

https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki

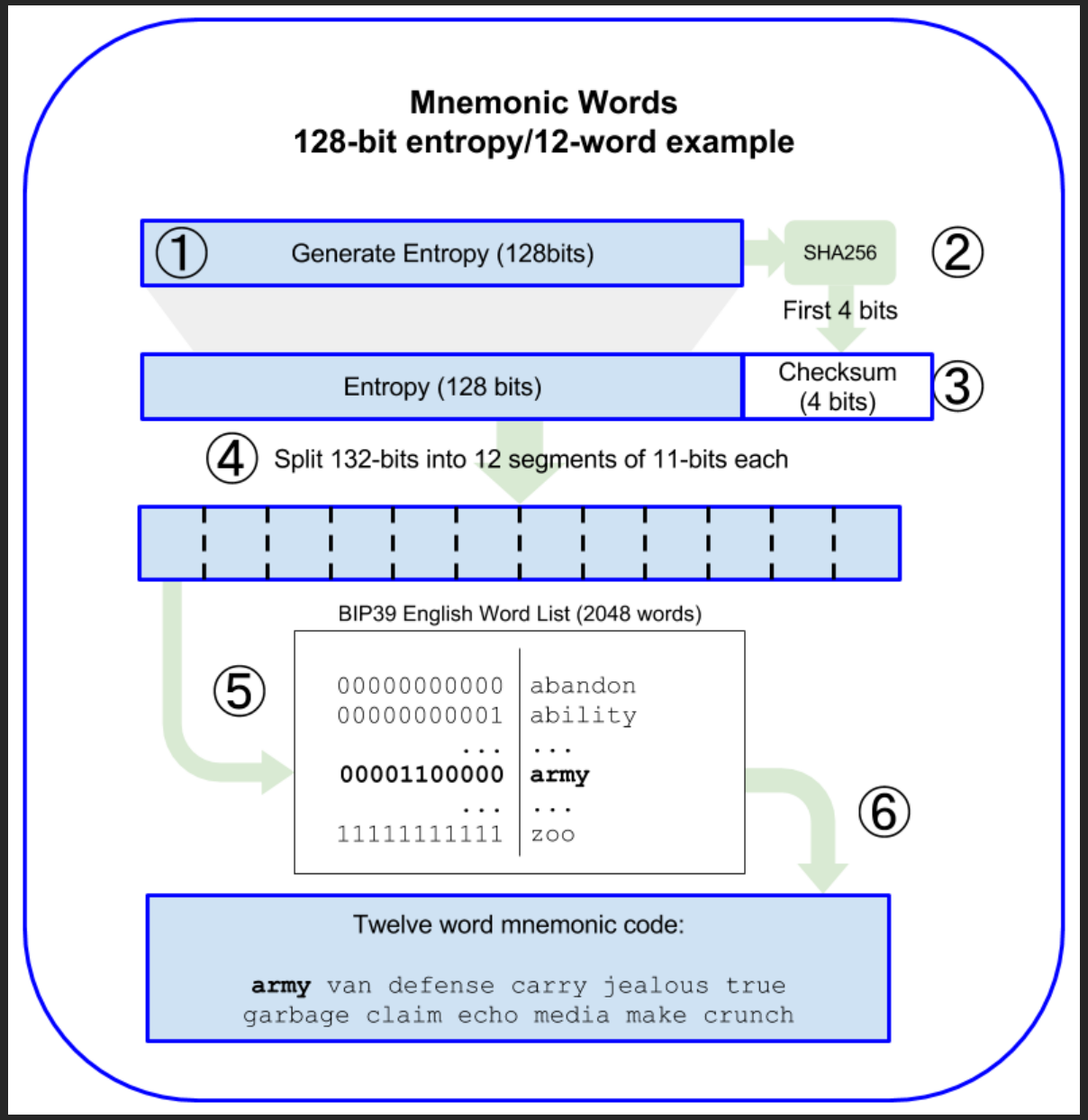
1. 128 ~ 256 bits의 무작위 암호화 시퀀스 S(Seed) 생성
2. S를 SHA-256 해시한 값의 첫 4 bits를 체크섬으로 생성
3. 무작위 시퀀스 S 뒤에 체크섬 추가
4. 시퀘스와 체크섬을 연결한 것을 11 bits 단위 나누기
5. 각각의 11 bits를 사전 정의된 2048 단어과 매핑
6. 단어의 시퀀스 순서를 유지하며 니모닉 코드 생성

Entropy

- (P)RNG : (Pseudo Random Number Generator)
- CSPRNG : 암호학적으로 안전한 유사난수 생성기 (Cryptographically Secure Pseudo Random Number Generator)
- DRBG : Deterministic Random Bit generator, NIST

* 256 bits recommended

Entropy (bits)	Checksum (bits)	Entropy + checksum (bits)	Mnemonic length (words)
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256*	8	264	24



니모닉 코드 생성 (Mnemonic code for generating deterministic keys) [BIP-39]

① 0x5729710d1126ca6635ea4d493727dd89 → ② SHA-256(0x57…89) = 0x2…

③ 0x5729710d1126ca6635ea4d493727dd892

```
01010111001010010111000100001101000100010010011011001010011001100 0
11010111101010010011010100100100110111001001111101110110001001001 0
```

④

01010111001	697
01001011100	604
01000011010	538
00100010010	274
01101100101	869
00110011000	408
11010111101	1725
01001001101	589
01001001001	585
10111001001	1481
11110111011	1979
00010010010	146

⑤

146	banner
274	car
408	credit
538	drive
585	empty
589	endless
604	entire
697	fire
869	hole
1481	rice
1725	stumble
1979	wash

⑥

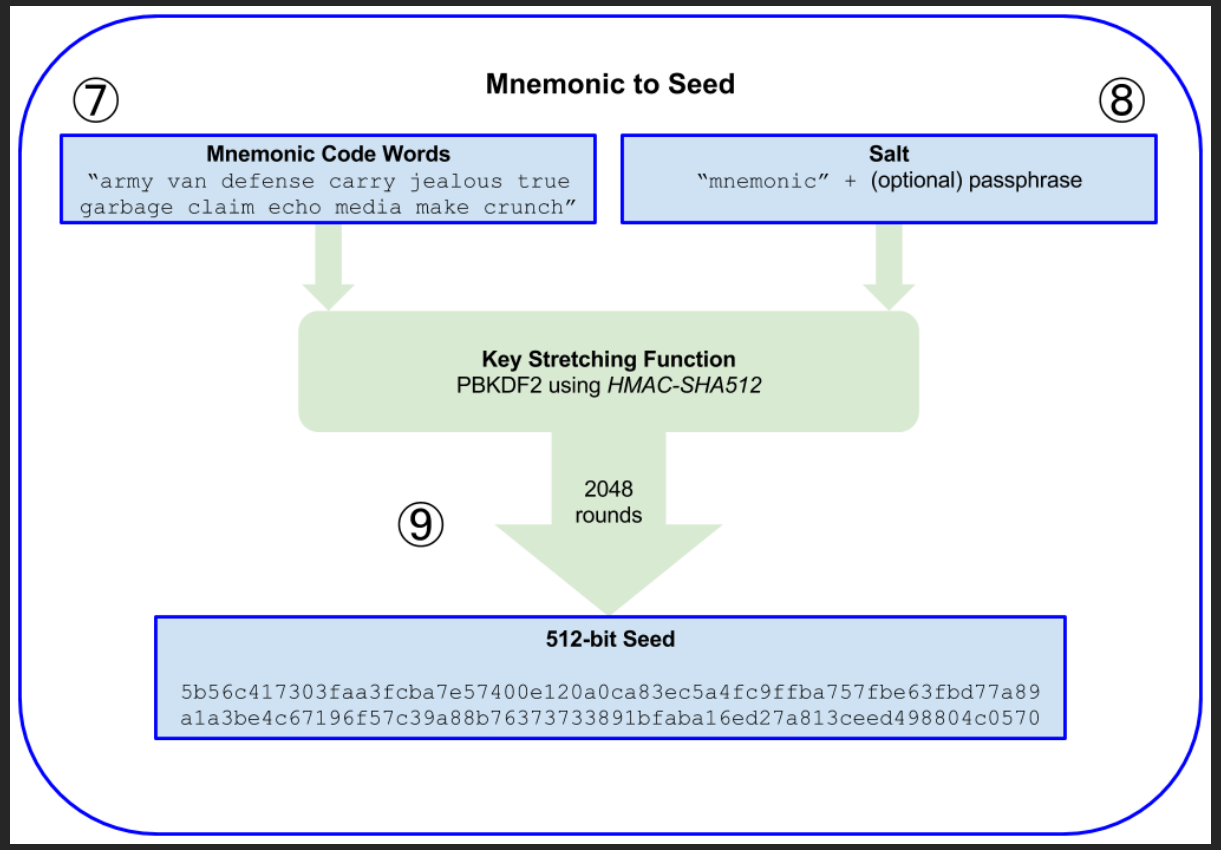
fire
entire
drive
car
hole
credit
stumble
endless
empty
rice
wash
banner

https://raw.githubusercontent.com/bitcoin/bips/master/bip-0039/english.txt

PBKDF2 : 키 스트레칭 함수(key-stretching function)

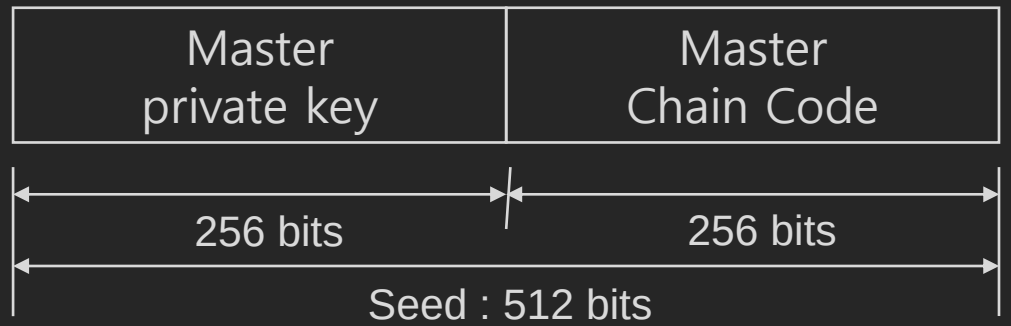
- PBKDF2 사용하여 512 bits 시드(Seed) 파생
- 파라미터 2개 : 니모닉 코드(Mnemonic Code), 솔트(Salt)
  - salt : 무차별 대입 공격을 가능하게 하는 조회 테이블(lookup table) 생성을 어렵게 함

7. PBKDF2 의 첫 번째 파라미터는 6. 에서 생성한 니모닉
8. PBKDF2 의 두 번째 파라미터는 솔트(Salt) "mnemonic" 문자열에 선택적 사용자 입력 암호문을 연결하여 구성
9. PBKDF2는 HMAC-SHA512 알고리즘을 사용하여 니모닉과 솔트 파라미터을 512 bits로 확장



512 bits Seed를 256 bits 씩 2개로 분리

- 왼쪽 256 bits : Master private Key
- 오른쪽 256 bits : Master Chain Code
- parse256(IL) == 0 or parse256(IL) >= n 인 경우 비유효



니모닉 코드 생성기

https://iancoleman.io/bip39/#english

v0.5.4

Mnemonic Code Converter

Mnemonic

You can enter an existing BIP39 mnemonic, or generate a new random one. Typing your own twelve words will probably not work how you expect, since the words require a particular structure (the last word contains a checksum).

For more info see the BIP39 spec.

Generate a random mnemonic: GENERATE 12 words, or enter your own below.

☐ Show entropy details

☐ Hide all private info

☑ Auto compute

Mnemonic Language English 日本語 Español 中文(简体) 中文(繁體) Français Italiano 한국어 Čeština Português

BIP39 Mnemonic

```
finger sustain crack smooth phone fatal endless bean diagram toilet erase awake
```

☐ Show split mnemonic cards

BIP39 Passphrase (optional)

BIP39 Seed

```
7370deba02d7f42a9656de7f6ef19b02b42d93de350e9f6f70594f567d6372d0af523d1faba7e1a3788ff4e7fafb6a34ff07b31
073289ae8e2395a75ac7e167d
```

Coin BTC - Bitcoin

BIP32 Root Key

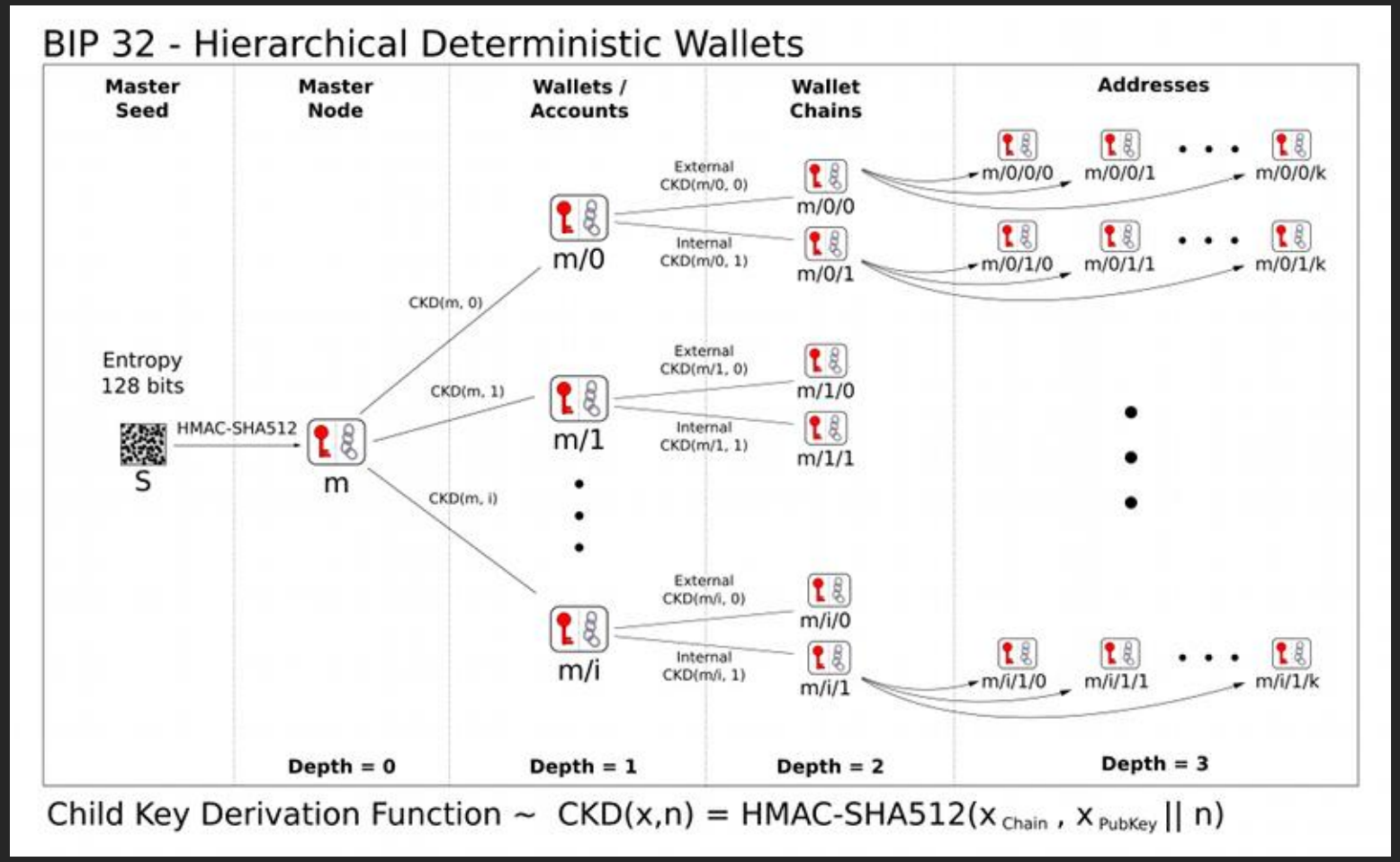
```
xprv9s21ZrQH143K2rT9YobeyaQFMtdybPVSGwHfi7W3kVrQFuJ8UoZpU6n2yNiC7Rtm9FZpRHxMbWSh5gHLXpUfniDo
LQ1dYYMWUhk13Ctz9xj
```

☐ Show BIP85

https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

BIP-32 HD Wallet (Hierarchical Deterministic Wallets, 결정적 계층 구조 지갑)

- 시드 (seed, 128 ~ 256 bits)로부터 생성된 키 트리
- 부모키로 여러 자식키를 파생
- 부모 개인키 없이
  부모 공개키로 자식 공개키 파생 가능
- 분기별로 용도를 다르게 사용 가능



- 타원 곡선 전자 서명(ECDSA) secp256k1 이용
  - n : 타원 곡선 차수(order)
    (0x FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C)
  - 두 좌표의 덧셈(+) : EC 그룹 연산의 적용, $a \star G + b \star G = (a + b) \star G \ (mod\, p)$
- || : byte sequence들을 연결 , 0x1234 || 0x5678 = 0x12345678
- 기호 설명
  - i : index
  - 소문자 p : private key ➞ 개인키, 대문자 P : point(p) = p * G = 타원곡선상 좌표 ➞ 공개키
  - ser : 직렬화(serialize)

변환 함수

- $point(p)$: Private Key p 의 타원 곡선 상 Public Key 좌표, $\mathrm{point(p)} = p * G$
- $ser_{32}(i)$: 32 bits unsigned integer i 를 4 bytes로 직렬화(serialize) (Big Endian) , 예) $ser_{32}(0)$ = 0x00000000
- $ser_{256}(p)$: 정수 p를 32 bytes로 직렬화(serialize) (Big Endian)
- $ser_p(P)$: 좌표 $P = (x, y)$ 를 SEC1의 압축 형식을 사용하여 직렬화
  - $(0x02\ or\ 0x03) \parallel ser_{256}(x)$ ➞ 0x02 : y 좌표 짝수, 0x03 : y 좌표 홀수
- $parse_{256}(p)$: 32 bytes p를 256 bits 숫자로 해석 (Big Endian)

확장 키 (Extended key) : (k, c) = (key, chain code)

- 체인코드(chain code) : 키 파생 시, 키 자체에만 의존하는 것을 방지하기 위한 추가 엔트로피(256 bits)
- 확장 개인 키(xprv, extended private key) = (k, c) , k = 개인키 c = 체인 코드
- 확장 공개 키(xpub, extended public key) = (K, c), K = point(k) (공개키) , c = 체인 코드

확장 키는 일반 자식 키, 강화 자식 키로 구분 (normal child key, hardened child key)

- 일반 자식 키(2^{31}개) - Index: 0 ~ 2^{31} - 1
- 강화 자식 키(2^{31}개) - Index: 2^{31} ~ 2^{32} - 1 ➡ $i + 2^{31} = i_H = i'$

자식 키 파생 함수 (CKD, Child Key Derivation Function), $CKD(x, n) = HMAC\text{-}SHA512(key = x_{chaincode}, data = x_{PublicKey} \| n)$

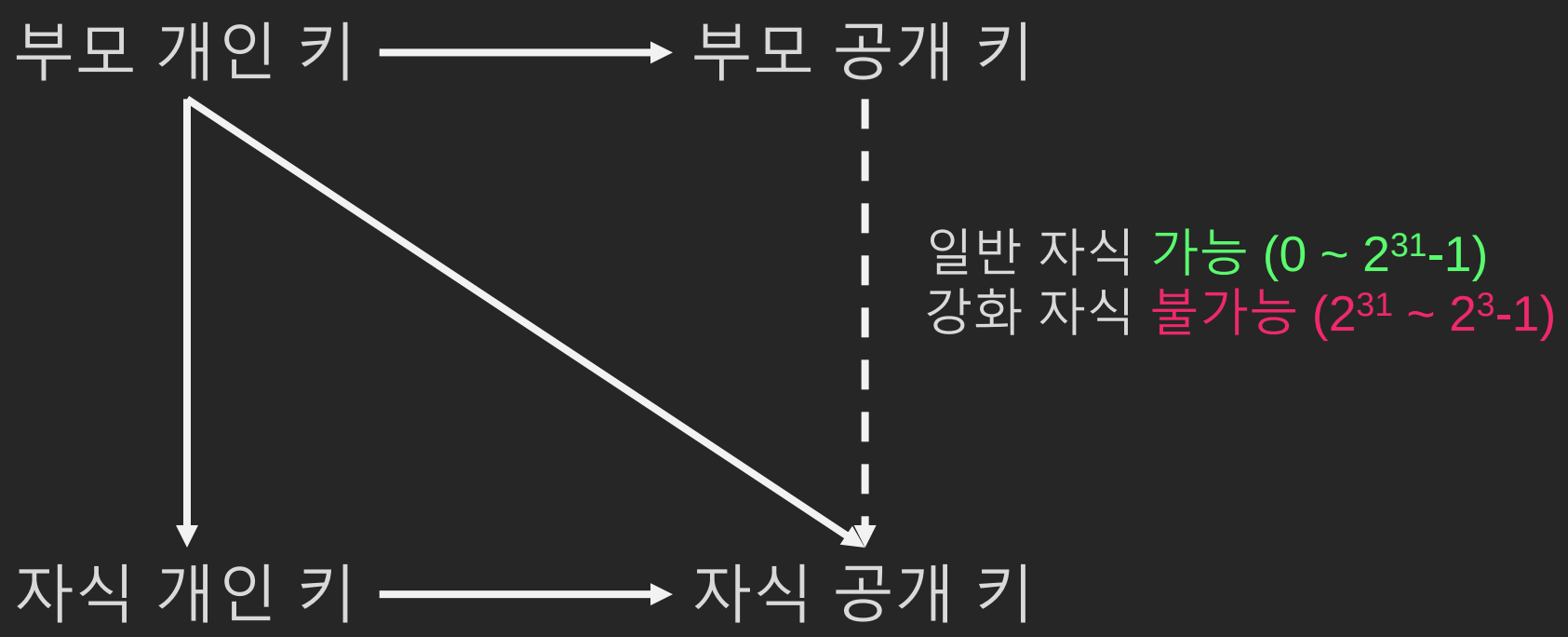
- 부모 확장 키와 인덱스 i가 주어지면 자식 확장 키 파생(derivation) 가능
- 자식의 강화 키 여부와 개인 키 또는 공개키 여부에 따라 다른 알고리즘 사용

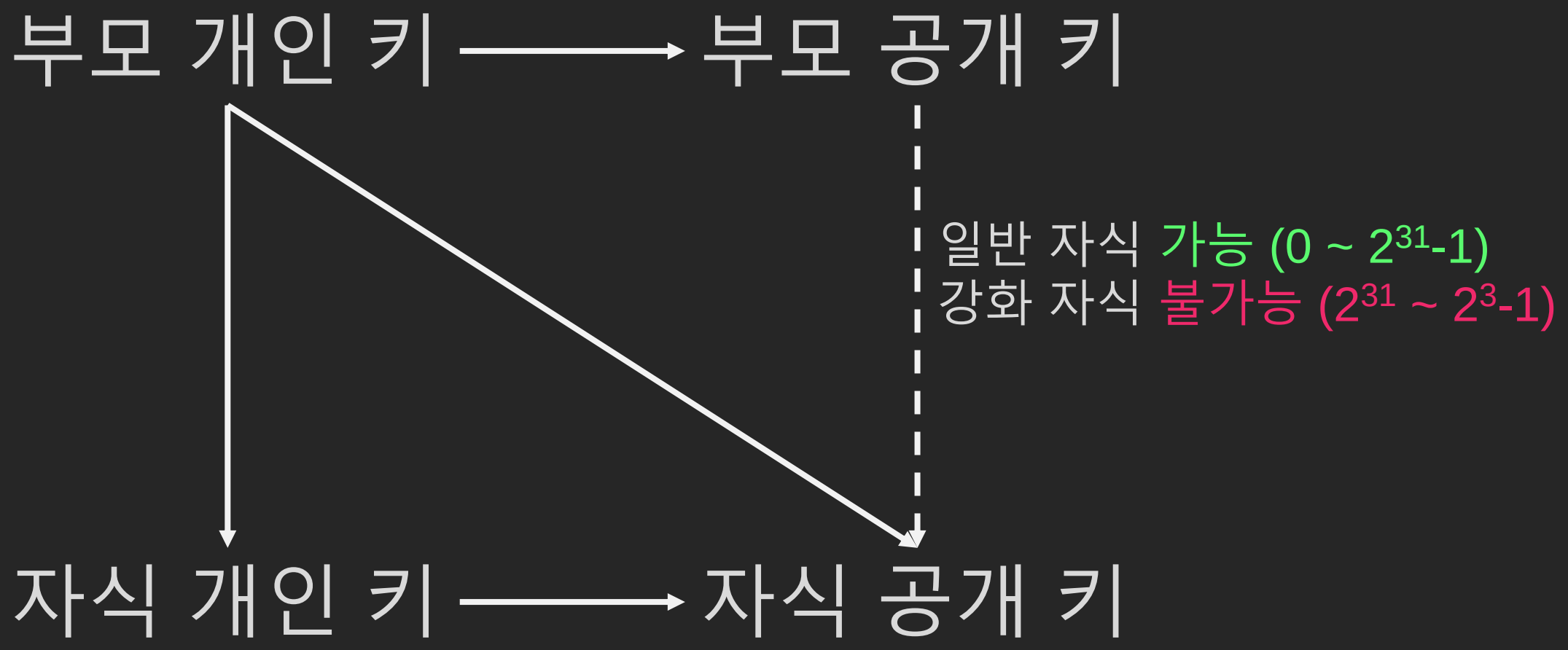
부모 개인 키 →
- 자식 개인 키 : $CKDpriv((k_{par}, c_{par}), i) \rightarrow (k_i, c_i)$
- 자식 공개 키 : $N((k, c)) \rightarrow (K, c)$ = 부모 공개 키,

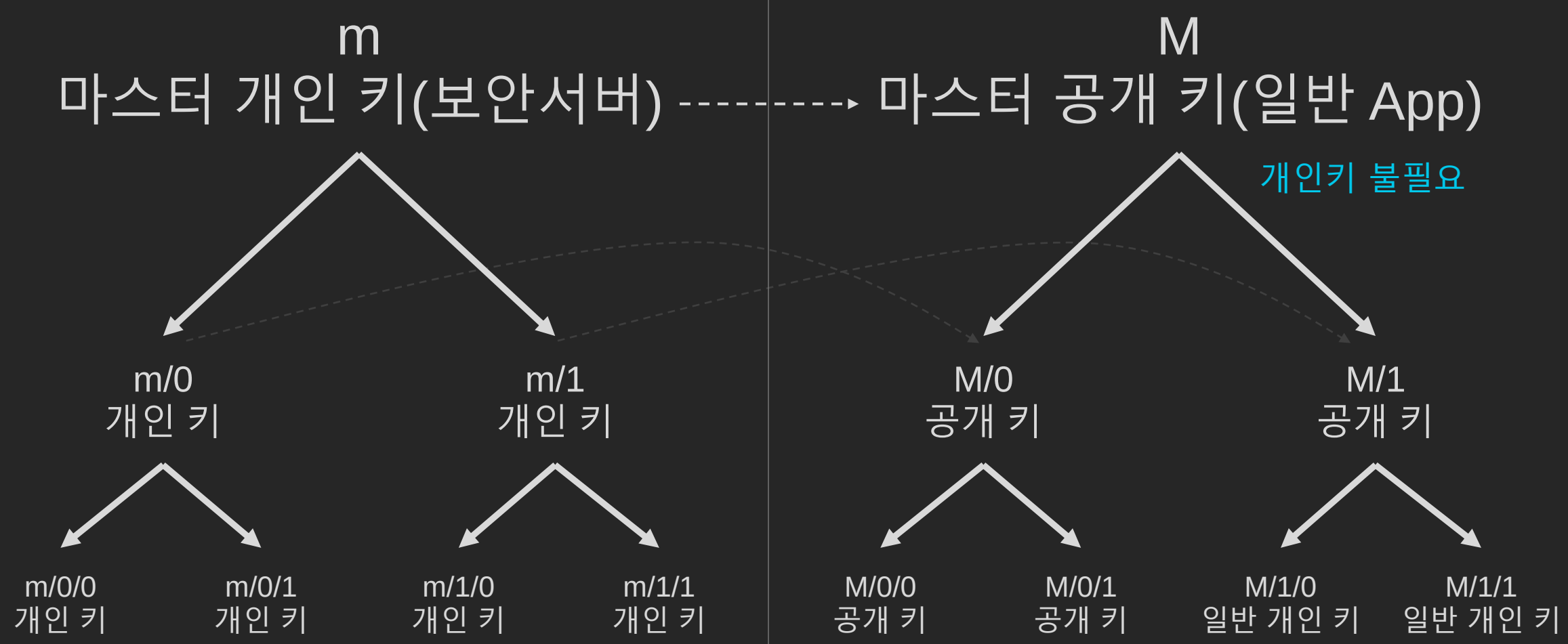
부모 공개 키 →
- ✗ 자식 개인 키 : 불가능
- 자식 공개 키 : $CKDpub((K_{par}, c_{par}), i) \rightarrow (K_i, c_i)$ (강화 자식 키 : 불가능)

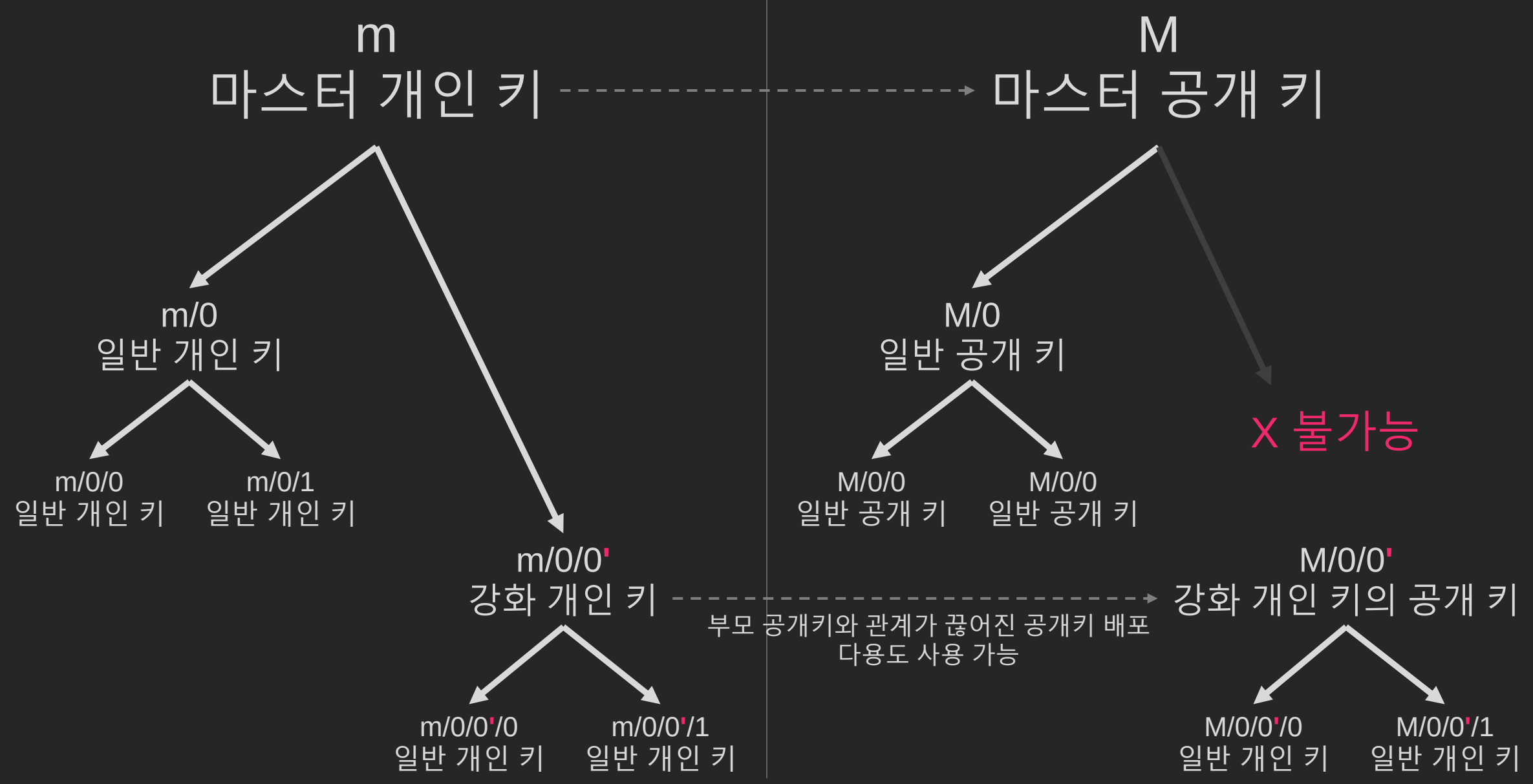
자식 키 파생 함수 (CKD, Child Key Derivation Function), $CKD(x, n) = HMAC\text{-}SHA512(key = x_{chaincode}, data = x_{PublicKey} \| n)$

- 부모 확장 키와 인덱스 i가 주어지면 자식 확장 키 파생(derivation) 가능
- 자식의 강화 키 여부와 개인 키 또는 공개키 여부에 따라 다른 알고리즘 사용




부모 개인 키
부모 공개 키
일반 자식 가능 (0 ~ 2^{31}-1)
강화 자식 불가능 (2^{31} ~ 2^{3}-1)
자식 개인 키
자식 공개 키


m
마스터 개인 키(보안서버)
M
마스터 공개 키(일반 App)
개인키 불필요
m/0
개인 키
m/1
개인 키
M/0
공개 키
M/1
공개 키
m/0/0
개인 키
m/0/1
개인 키
m/1/0
개인 키
m/1/1
개인 키
M/0/0
공개 키
M/0/1
공개 키
M/1/0
일반 개인 키
M/1/1
일반 개인 키


m
마스터 개인 키
M
마스터 공개 키
m/0
일반 개인 키
m/0/0
일반 개인 키
m/0/1
일반 개인 키
m/0/0'
강화 개인 키
m/0/0'/0
일반 개인 키
m/0/0'/1
일반 개인 키
M/0
일반 공개 키
M/0/0
일반 공개 키
M/0/0
일반 공개 키
X 불가능
M/0/0'
강화 개인 키의 공개 키
부모 공개키와 관계가 끊어진 공개키 배포
다용도 사용 가능
M/0/0'/0
일반 개인 키
M/0/0'/1
일반 개인 키

부모 개인 키 ➞ 자식 개인 키 : $CKDpriv((k_{par}, cp_{ar}), i) \rightarrow (ki, ci)$

1. i >= 2^{31} (강화 자식 키 유도 여부 확인)
   - true (강화 자식)

$$I = HMAC - SHA512(Key = c_{par}, Data = ser_{256}(k_{par}) \parallel ser_{32}(i))$$

부모 체인코드 부모 개인키 Index

   - false (일반 자식)

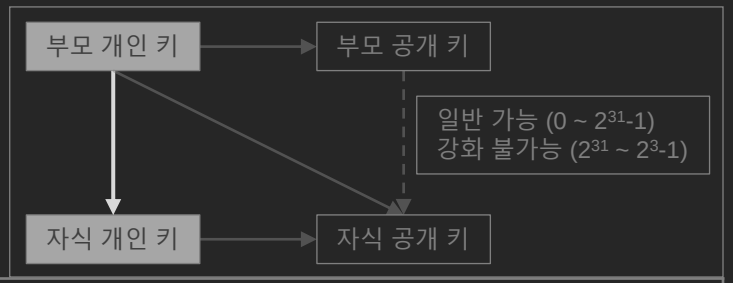
$$I = HMAC - SHA512(Key = c_{par}, Data = serp(point(k_{par})) \parallel ser_{32}(i))$$

부모 체인코드 부모 공개키 Index

CDKpriv	자식 개인 키 유도 함수
CDKpub	자식 공개 키 유도 함수
k_{par}	부모 개인키
K_{par}	부모 공개키
c_{par}	부모 체인코드
i	Index
k_i	index i의 자식 개인 키
K_i	index i의 자식 공개 키
c_i	index i의 자식 체인코드

2. I 를 2개의 32 bytes 시퀀스로 나눔 I_L, IR
3. 자식 키 $k_i = parse_{256}(I_L) + kpar\ (mod\ n)$
4. 자식 체인코드 $c_i = IR$
5. $parse_{256}(I_L) > n$ 또는 $k_i = 0$ 인 경우 비유효(Invalid)

   다음 index를 사용



$point(p)$	Private Key p 의 타원 곡선 상 Public Key 좌표 $point(p) = p * G$
$ser_{32}(i)$	32 bits unsigned integer i 를 4 bytes로 직렬화(serialize) (Big Endian)
$ser_{256}(p)$	정수 p를 32 bytes로 직렬화(serialize) (Big Endian)
$ser_p(P)$	좌표 $P = (x, y)$ 를 SEC1의 압축 형식을 사용하여 직렬화 $(0x02\ or\ 0x03) \parallel ser_{256}(x)$ ➞ 0x02 : y 좌표 짝수, 0x03 : y 좌표 홀수
$parse_{256}(p)$	32 bytes p를 256 bits 숫자로 해석 (Big Endian)

부모 공개 키 ➞ 자식 공개 키 : $CKDpub((K_{par}, cp_{ar}), i) \rightarrow (Ki, ci)$

1. i >= 2^{31} (강화 자식 키 유도 여부 확인)
   - true (강화 자식)

     불가능

   - false (일반 자식)

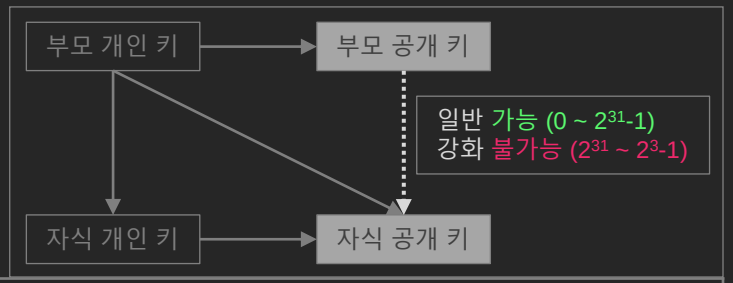
$$I = HMAC - SHA512(Key = c_{par}, Data = serp(K_{par}) \parallel ser_{32}(i))$$

부모 체인코드 부모 공개키 Index

CDKpriv	자식 개인 키 유도 함수
CDKpub	자식 공개 키 유도 함수
k_{par}	부모 개인키
K_{par}	부모 공개키
c_{par}	부모 체인코드
i	Index
k_i	index i의 자식 개인 키
K_i	index i의 자식 공개 키
c_i	index i의 자식 체인코드

2. I 를 2개의 32 bytes 시퀀스로 나눔 I_L, IR
3. 자식 키 $K_i = point(parse_{256}(I_L)) + Kpar$
4. 자식 체인코드 $c_i = IR$
5. $parse_{256}(I_L) > n$ 또는 $K_i = \mathcal{O}$ 인 경우 비유효(Invalid)

   다음 index를 사용



$point(p)$	Private Key p 의 타원 곡선 상 Public Key 좌표 $\mathrm{point(p)} = p * G$
$ser_{32}(i)$	32 bits unsigned integer i 를 4 bytes로 직렬화(serialize) (Big Endian)
$ser_{256}(p)$	정수 p를 32 bytes로 직렬화(serialize) (Big Endian)
$ser_p(P)$	좌표 $P = (x, y)$ 를 SEC1의 압축 형식을 사용하여 직렬화 $(0x02\ or\ 0x03) \parallel ser_{256}(x)$ ➞ 0x02 : y 좌표 짝수, 0x03 : y 좌표 홀수
$parse_{256}(p)$	32 bytes p를 256 bits 숫자로 해석 (Big Endian)

개인 키 ➞ 공개 키

$N((k,c)) \rightarrow (K,c)$

- 공개키 $K = point(k)$
- 체인코드 : c 로 동일

부모 개인 키 ➞ 자식 공개 키

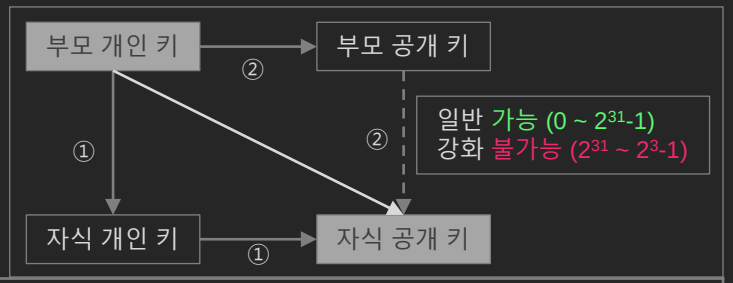
1. $N\left(CKDpriv((k_{par}, cp_{ar}), i)\right)$ (항상 가능)

   자식 개인 키 계산 ➞ 자식 공개키 계산

2. $CDKpub(N(k_{par}, cpar), i)$ (강화 자식 불가)

   부모 공개키 계산 ➞ 자식 개인키 계산

CDKpriv	자식 개인 키 유도 함수
CDKpub	자식 공개 키 유도 함수
k_{par}	부모 개인키
K_{par}	부모 공개키
c_{par}	부모 체인코드
i	Index
k_i	index i의 자식 개인 키
K_i	index i의 자식 공개 키
c_i	index i의 자식 체인코드



$point(p)$	Private Key p 의 타원 곡선 상 Public Key 좌표 $\text{point(p)} = p * G$
$ser_{32}(i)$	32 bits unsigned integer i 를 4 bytes로 직렬화(serialize) (Big Endian)
$ser_{256}(p)$	정수 p를 32 bytes로 직렬화(serialize) (Big Endian)
$ser_p(P)$	좌표 $P = (x, y)$ 를 SEC1의 압축 형식을 사용하여 직렬화 $(0x02\ or\ 0x03) \parallel ser_{256}(x)$ ➞ 0x02 : y 좌표 짝수, 0x03 : y 좌표 홀수
$parse_{256}(p)$	32 bytes p를 256 bits 숫자로 해석 (Big Endian)

키 경로(key path)

- m : master private key , M : Master Public Key
- i : Index
  - i : 일반 자식 ($0 \sim 2^{31} - 1$) 예) 1 = 1
  - i_H or i' : 강화 자식 ($2^{31} \sim 2^{32} - 1$) → $2^{31} + i$ 예) 1' = 2^{31} + 1
- CKDpriv(m, i) = m / i , CKDpub(M, i) = M / i , N(private key) = Public Key

Master Node			
Depth = 0	Depth = 1	Depth = 2	Depth = 3
Master Key m →	m/1 →	m/1/3 →	m/1/3/2'

N(m/a/b/c)
= N(m/a/b)/c
= N(m/a)/b/c
= N(m)/a/b/c
= M(a/b/c)

N(m/a'/b/c)
= N(m/a'/b)/c
= N(m/a')/b/c

N(m/a')
≠ N(m)/a' (계산 불가)

	private key		Public Key	
CKDpriv(m, 1)	= m / 1		= N(m / 1)	= M / 1
CKDpriv(CKDpriv(m, 1), 3)	= m / 1 / 3	→	= N(m / 1 / 3)	= M / 1 / 3
CKDpriv(CKDpriv(CKDpriv(m, 1), 3), 2')	= m / 1 / 3 / 2'		= N(m / 1 / 3 / 2')	= M / 1 / 3 / 2'

BIP-43 다목적 HD지갑 트리 구조 표준 (Purpose Field for Deterministic Wallets)

- HD Wallet은 너무 많은 자유도를 제공
- 트리 구조 Depth = 1 을 "목적(purpose)"으로 사용

m / purpose' / * (' : 강화 파생 사용)

- m / 0' / * 는 BIP-32(기본 계정)에서 사용 중
- 10001 ~ 19999 : SLIP용으로 예약, https://github.com/satoshilabs/slips

  SLIP: SatoshiLabs Improvement Proposals

BIP-44 복수화폐(multicurrency) 및 복수계정(multiaccount) 지갑 (Multi-Account Hierarchy for Deterministic Wallets)

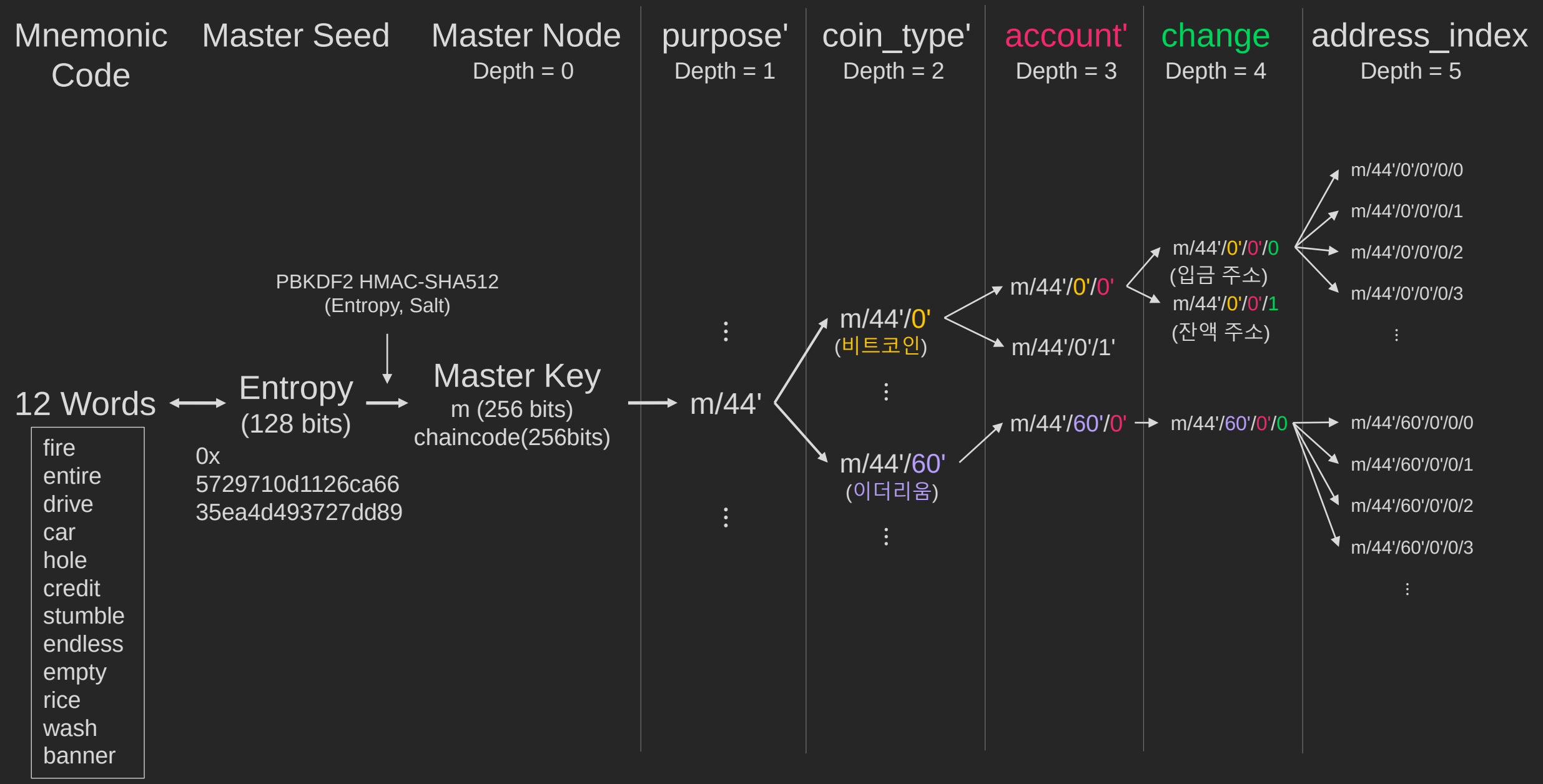
- BIP-43 확장, '목적' 번호 44'를 복수 화폐 복수 계정 구조로 사용
- 모든 HD 지갑 구조는 하나의 트리 분리(m/44'/*)만 사용하여 식별
- 다섯 가지 트리 레벨로 구성된 구조를 지정

m / 44' / coin_type' / account' / change / address_index (' : 강화 파생 사용)

Depth	유형(Type)	설명(Description)
1	purpose'	44' 고정 값
2	coin_type'	암호화폐 유형, 비트코인(m/44'/0'), 이더리움(m/44'/60') - SLIP0044
3	account'	사용 목적 (회계, 기부, 저축, 공동 비용 등)
4	change	비트코인 : 0 - 수신 주소, 1 - 잔액 주소(change address,) 이더리움 : 수신 주소(0)만 사용, 잔액 주소(1) 불필요 일반 파생 사용 ➡ 비보안 환경에서 사용할 수 있도록 확장된 공개키를 트리의 계정 수준에서 내보낼 수 있음
5	address_index	실제 사용 주소

m / 44' / coin_type' / account' / change / address_index (' : 강화 파생 사용)

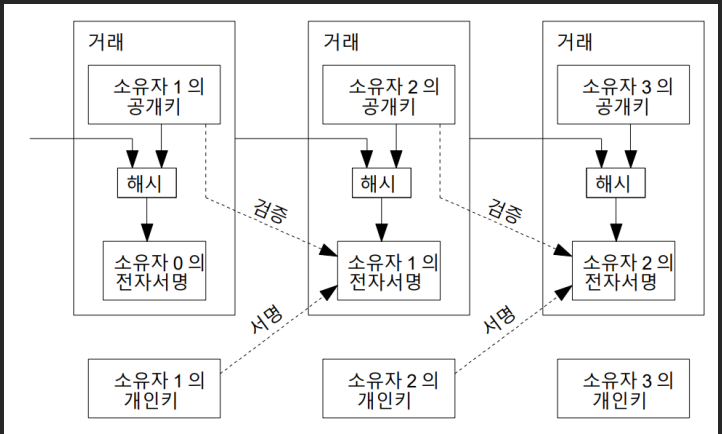
HD 경로	설명
M/44'/0'/0'/0/0	비트코인 첫 번째 수신 주소 공개키 M(공개키)/44'/0'(비트코인)/0'/0(수신 계정)/0(첫번째)
M/44'/0'/0'/1/2	비트코인 세 번째 잔액 주소 공개키 M(공개키)/44'/0'(비트코인)/0'/1(잔액 계정)/2(세번째)
m/44'/60'/0'/0/5	이더리움 수신 주소 여섯 번째 개인키 m(비밀키)/44'/60'(이더리움)/0'/0(수신 계정)/5(여섯번째)


Mnemonic Code
Master Seed
Master Node
Depth = 0
purpose'
Depth = 1
coin_type'
Depth = 2
account'
Depth = 3
change
Depth = 4
address_index
Depth = 5
PBKDF2 HMAC-SHA512
(Entropy, Salt)
12 Words
fire
entire
drive
car
hole
credit
stumble
endless
empty
rice
wash
banner
Entropy
(128 bits)
0x
5729710d1126ca66
35ea4d493727dd89
Master Key
m (256 bits)
chaincode(256bits)
m/44'
m/44'/0'
(비트코인)
m/44'/60'
(이더리움)
m/44'/0'/0'
m/44'/0'/1'
m/44'/60'/0'
m/44'/0'/0'/0
(입금 주소)
m/44'/0'/0'/1
(잔액 주소)
m/44'/60'/0'/0
m/44'/0'/0'/0/0
m/44'/0'/0'/0/1
m/44'/0'/0'/0/2
m/44'/0'/0'/0/3
m/44'/60'/0'/0/0
m/44'/60'/0'/0/1
m/44'/60'/0'/0/2
m/44'/60'/0'/0/3

Transaction

2. 거래

우리는 디지털 서명의 사슬로써 전자적 화폐(electronic coin)를 정의한다. 각 소유자는 화폐를 송금할 때 먼젓번 거래 내역 및 다음 소유자 공개키의 해시값에 전자적으로 서명을 하고 이 정보를 이 화폐의 끝에 첨가한다. 수금자(payee)는 그 소유권(ownership)의 사슬을 검증할 서명을 검증할 수 있다.

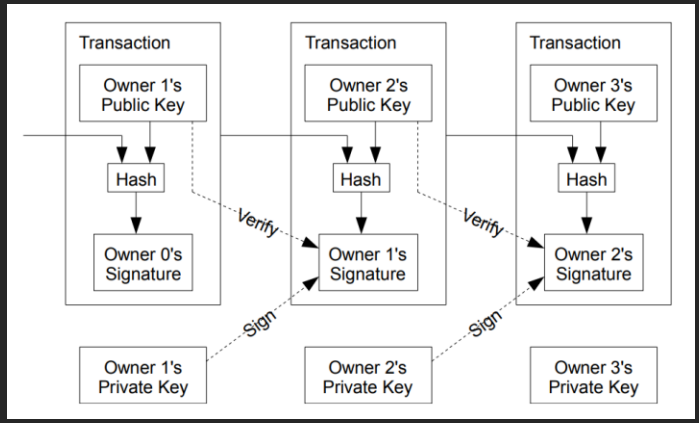


이 과정상 문제는 수금자가 소유자 가운데 누군가 화폐를 이중지불하지 않았는지 검증할 수 없다는 점이다. 통상적인 해법은 신뢰받는 중앙통제기관(trusted central authority)이나 조폐국(mint)을 세우고 모든 거래마다 이중지불 여부를 점검하는 것이다. 거래를 마칠 때마다 이 화폐는 조폐국으로 회수돼 새로운 화폐로 발행돼야 하고, 조폐국에서 직접 발행된 화폐만이 이중지불되지 않았다는 신뢰를 받는다. 이 해법을 적용시 문제는 전체 통화체계(the entire money system)가, 은행처럼 모든 거래가 거쳐가야하는, 조폐국을 운영하는 회사에 의존한다는 점이다.

우리에게는 먼젓번 소유자가 이전에 어떤 거래에도 서명하지 않았음을 수금자에게 알릴 수단이 필요하다. 이런 목적에서 우리는 가장 앞선 거래 하나를 인정하고, 이후 이중지불 시도에는 신경쓰지 않는다. 그런 (이중지불된) 거래가 없음을 확인할 유일한 방법은 모든 거래를 인식하는 것뿐이다. 조폐국 기반 모델에서, 조폐국은 모든 거래를 인식했고 최초로 받은 거래를 (승인 대상으로) 결정했다. 어떤 신뢰받는 자도 없이 이 방식을 실현하려면, 거래는 공개적으로 알려져야 하고[1], 참가자들에게는 그걸 받은 순서의 단일한 이력에 합의하는 시스템이 필요하다. 수금자는 각 거래의 시점에 그게 최초로 받은 거래임을 노드 다수가 동의했다는 증명을 요한다.

2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.



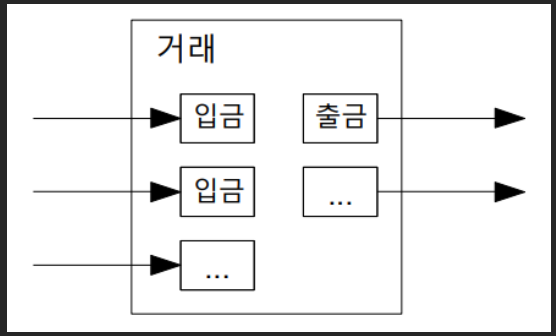
The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

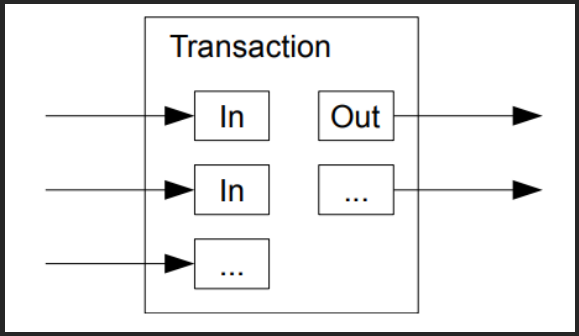
We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

9. 가치 합치기와 나누기

화폐를 독립적으로 다루는 것은 가능하더라도, 송금에 모든 푼돈(every cent)을 별도 거래로 만드는 건 무리한 일이다. 가치를 나누고 합칠 수 있도록, 거래는 복수의 입출금을 포함한다. 일반적으로 입금은 더 큰 먼젓번 거래의 단수 입금 또는 더 작은 양을 결합한 복수 입금이며, 출금은 지불용 출금 하나와 만일 있다면 송금인 (sender)에게 돌려줄 거스름돈 출금 하나, 이렇게 많아야 둘이다.

펼친 부채꼴(fan-out)처럼, 거래가 여러 거래에 의존하고 그 여러 거래가 더 많은 거래에 의존하는 것은 문제가 되지 않는다는 것에 주목해야 한다. 완전 독립된(standalone) 거래 내역 사본을 추출해야 할 필요는 전혀 없다.





9. Combining and Splitting Value

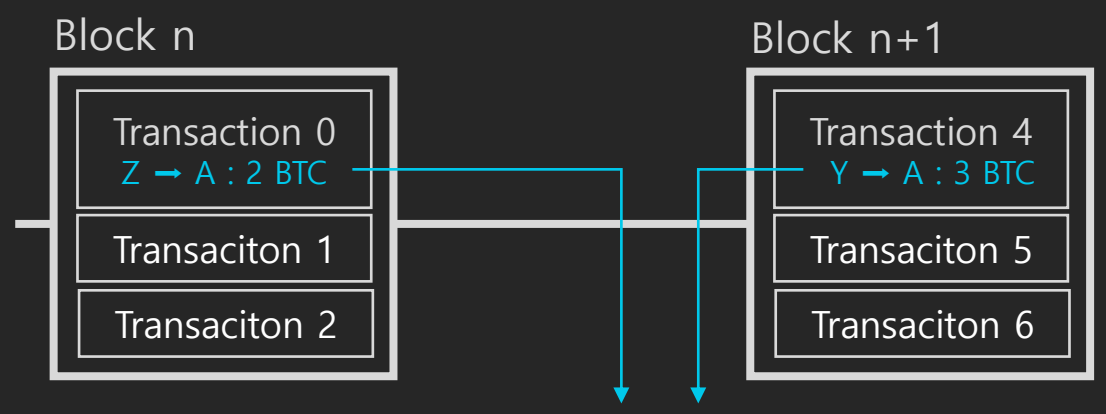
Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.

It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

UTXO

(Unspent Transaction Output)

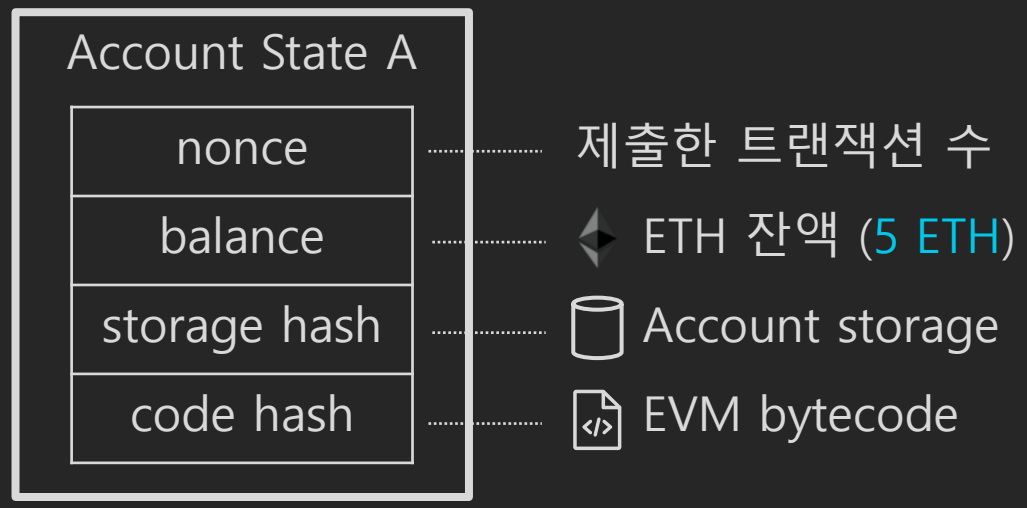
- 미사용 거래 출력 모델
- 블록체인 (Distributed Ledger)
- UTXO를 Input으로 사용하여 화폐 전송
- 비트코인 : 모든 블록의 트랜잭션 조회 필요



- 잔액 = UTXO의 합, A의 잔액 : 2 + 3 = 5 BTC

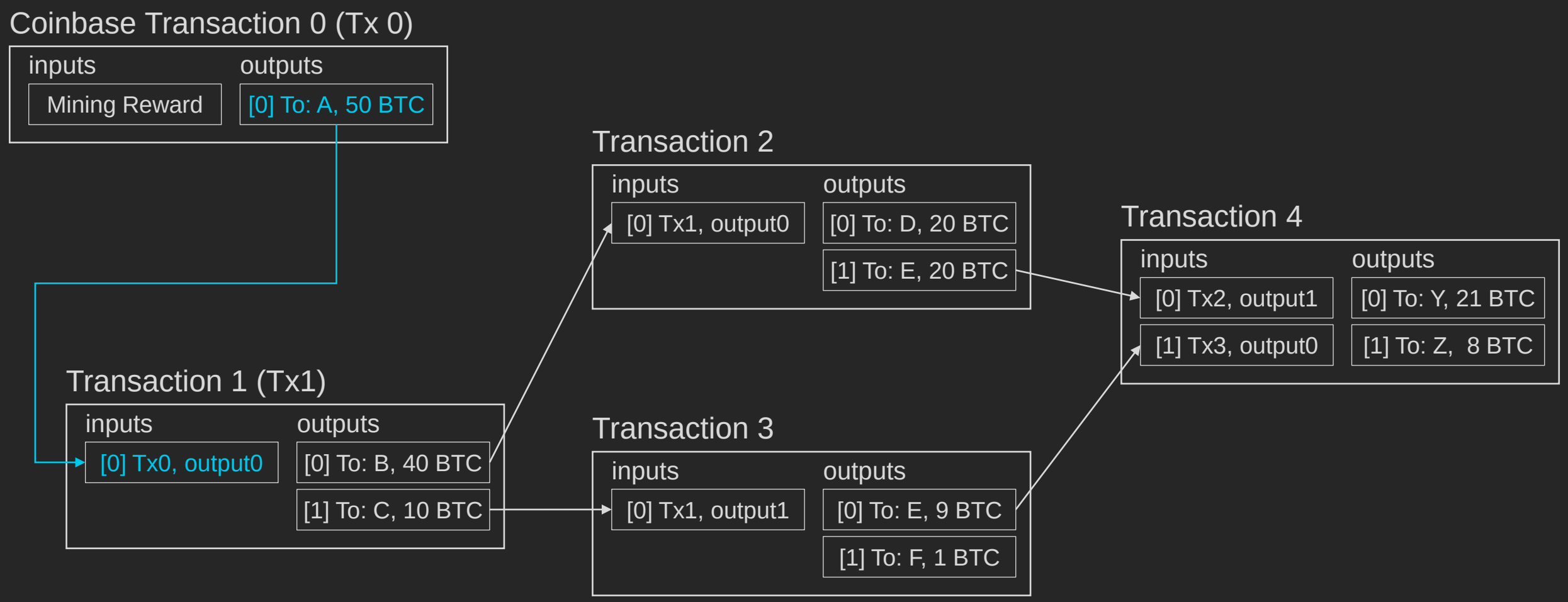
Account-based

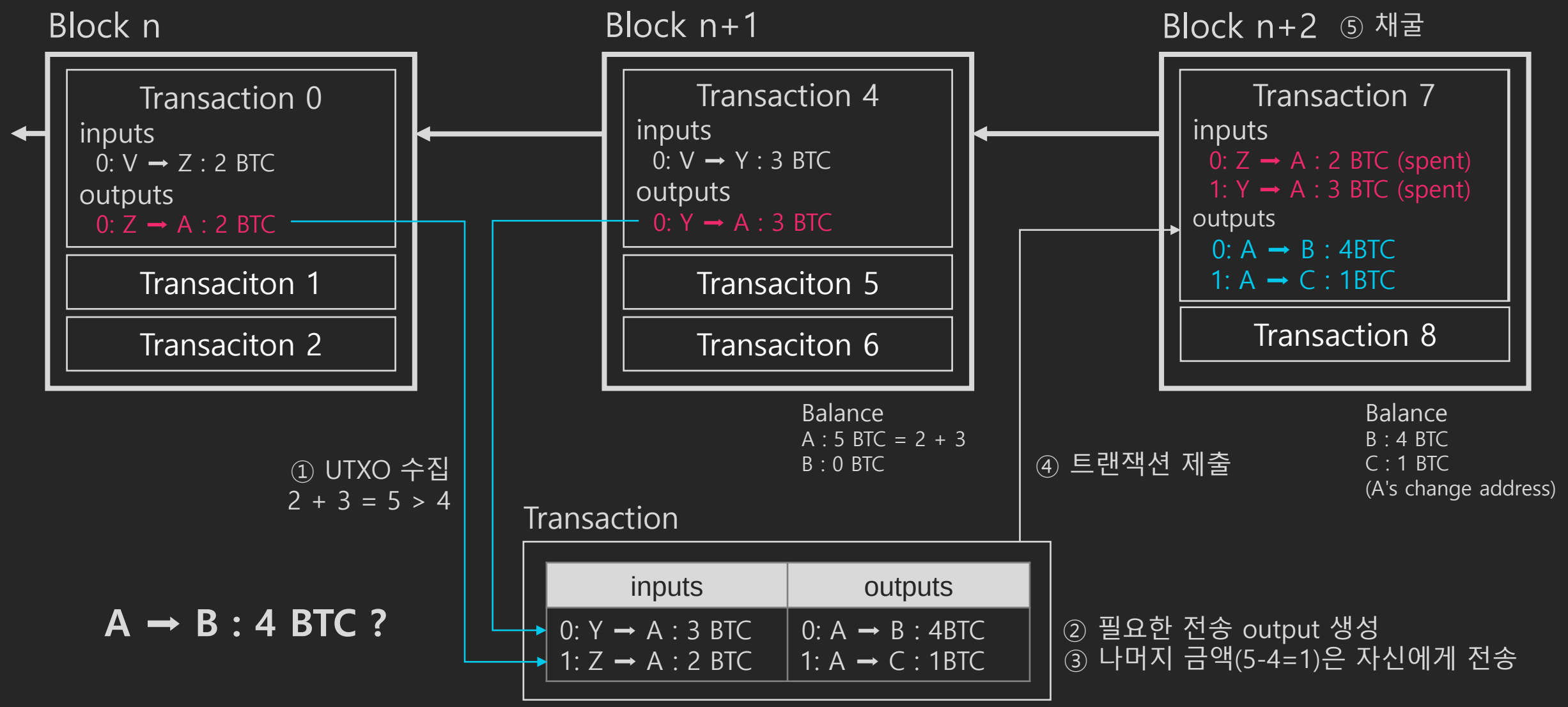
- 계정 기반 모델
- 블록체인 + World State (Distributed State Machine)
  - 블록체인 : 상태 탐색 및 검증 정보
  - World State : 계정 상태 목록
- 이더리움 : 계정 상태 조회



UTXO

(Unspent Transaction Output)


Coinbase Transaction 0 (Tx 0)
inputs
Mining Reward
outputs
[0] To: A, 50 BTC
Transaction 1 (Tx1)
inputs
[0] Tx0, output0
outputs
[0] To: B, 40 BTC
[1] To: C, 10 BTC
Transaction 2
inputs
[0] Tx1, output0
outputs
[0] To: D, 20 BTC
[1] To: E, 20 BTC
Transaction 3
inputs
[0] Tx1, output1
outputs
[0] To: E, 9 BTC
[1] To: F, 1 BTC
Transaction 4
inputs
[0] Tx2, output1
[1] Tx3, output0
outputs
[0] To: Y, 21 BTC
[1] To: Z, 8 BTC

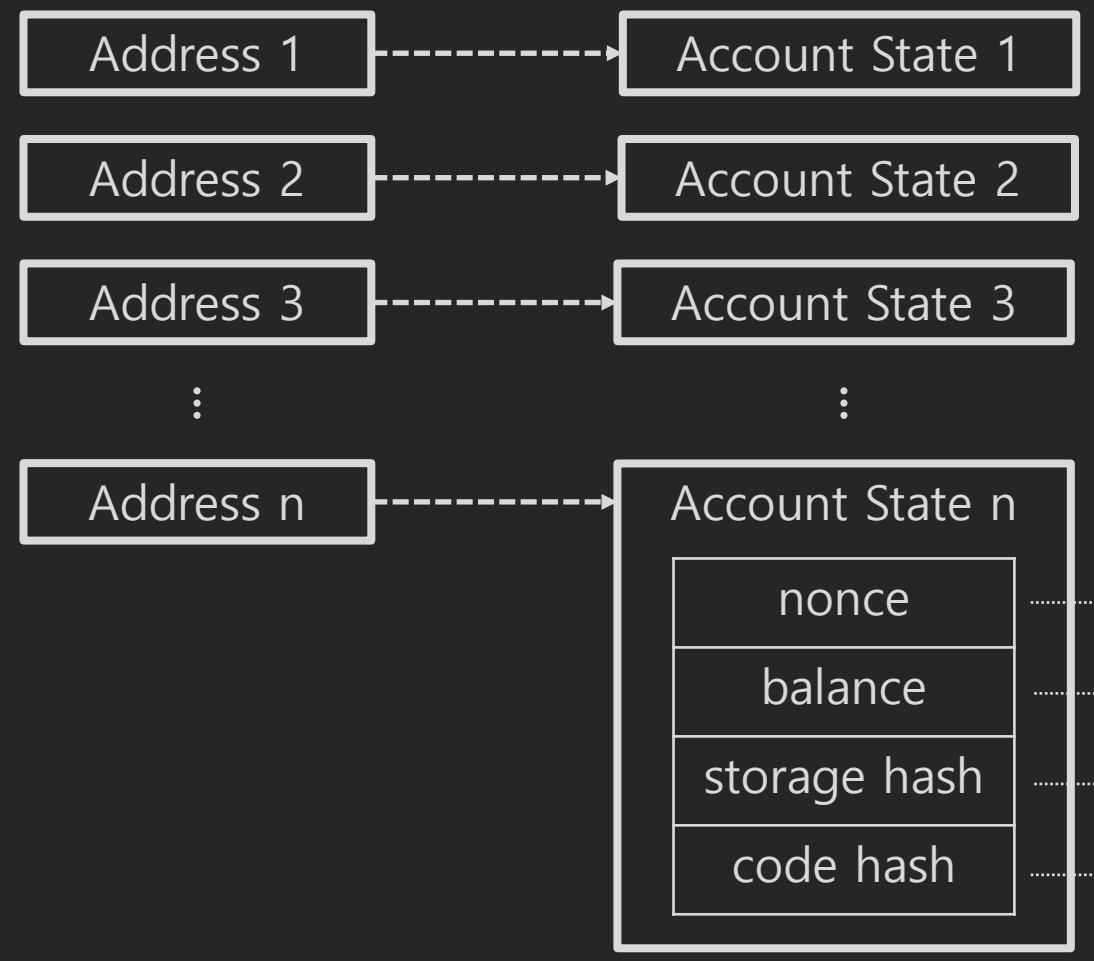


A → B : 4 BTC ?

기존 것을 무효화(spent)하고 새 것 생성 (unspent)하여 교체

Account-based

World State



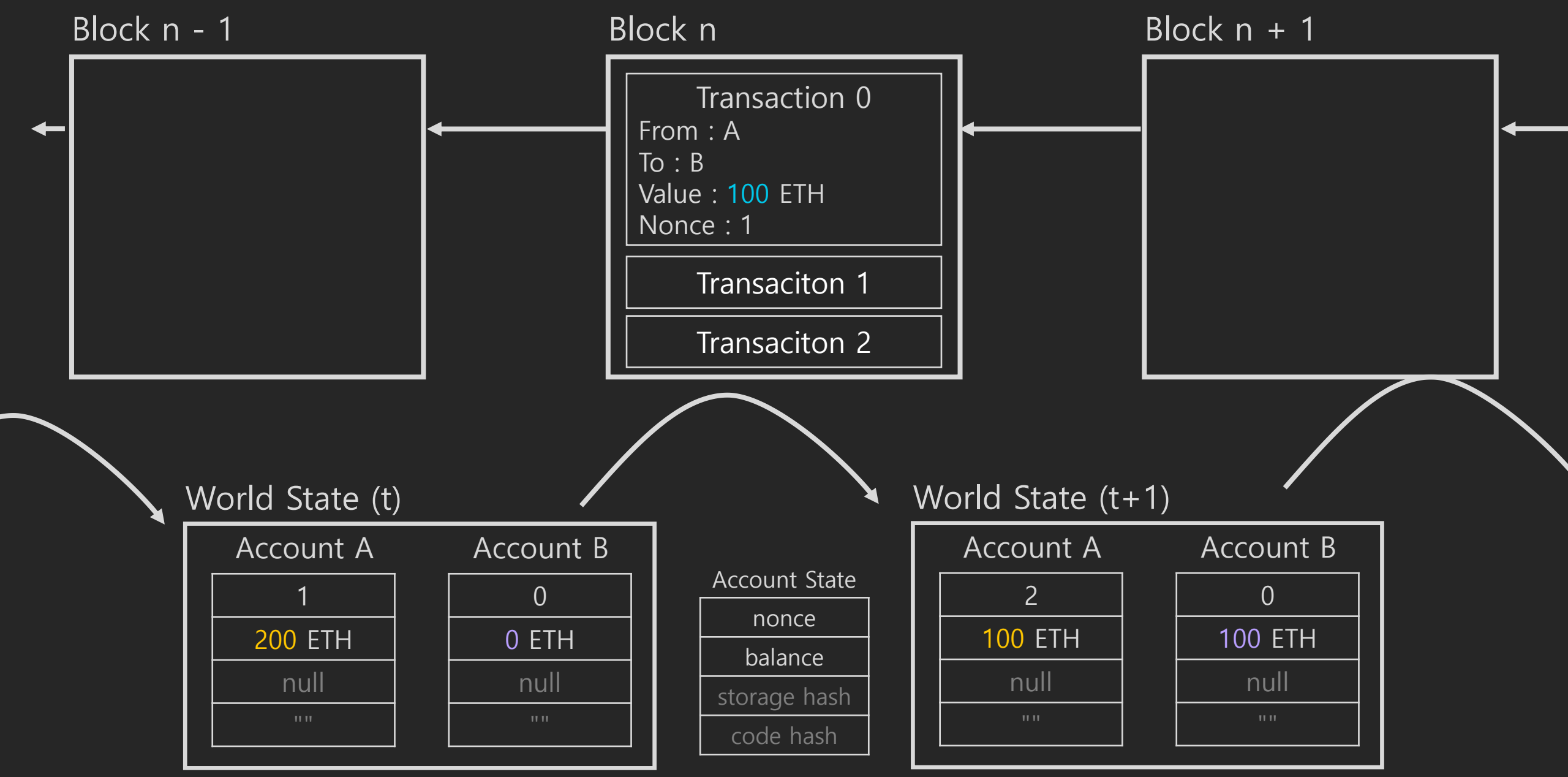
1 wei
1 Gwei = $1*10^9$ wei
1 szabo = $1*10^{12}$ wei
1 finney = $1*10^{15}$ wei
1 ether(ETH) = $1*10^{18}$ wei

제출한 트랜잭션 수 (중복, 건너뛰기 불가), 0 → 1 → ···

ETH 잔액 (wei 단위) (1 ether = 1×10^{18} wei)

Account storage

EVM bytecode


Block n - 1
Block n
Transaction 0
From : A
To : B
Value : 100 ETH
Nonce : 1
Transaciton 1
Transaciton 2
Block n + 1
World State (t)
Account A
1
200 ETH
null
""
Account B
0
0 ETH
null
""
Account State
nonce
balance
storage hash
code hash
World State (t+1)
Account A
2
100 ETH
null
""
Account B
0
100 ETH
null
""

Smart Contract

블록체인 상에서 동작하는 프로그램

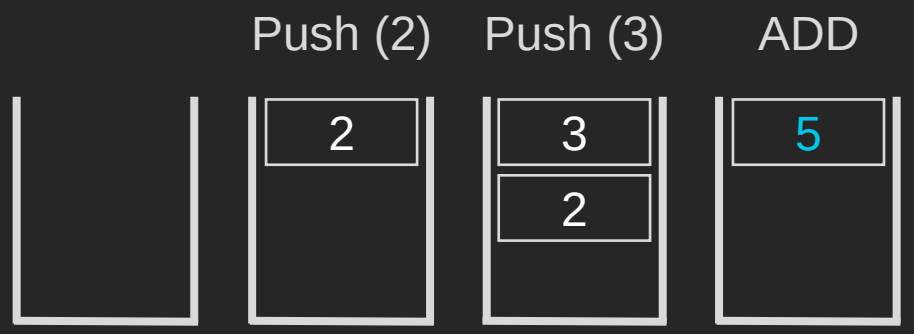
- 블록체인 상에서 동작하는 프로그램
- 1994년 닉 재보(Nick Szabo)가 처음 고안한 개념
  - 계약에 필요한 요소를 코드를 통해 스스로 실행되게 하는 전산화된 거래 약속
    일정한 조건이 충족되는 자동으로 계약이 실행
    예) 자동 판매기 : 미리 정해진 액수 이상의 돈을 투입 ➡ 자동으로 원하는 상품을 구매
  - 계약 조건을 실행하는 컴퓨터화된 거래 프로토콜 (Smart Contracts, 1994)
    a computerized transaction protocol that executes the terms of a contract
  - 프로토콜을 포함하는 약속 집합, 프로그램으로 구현된 계약은 종이 기반 계약보다 "스마트" (Smart Contracts Glossary, 1995)
    A set of promises, including protocols within which the parties perform on the other promises. The protocols are usually implemented with programs on a computer network, or in other forms of digital electronics, thus these contracts are "smarter" than their paper-based ancestors. No use of artificial intelligence is implied.
    Protocol: A sequence of messages between multiple agents.
- 스마트 계약 설계의 기본 원칙(Some Basic Principals of Contract Design)
  (Smart Contracts: Building Blocks for Digital Markets, Nick Szabo, 1996)
  - 관측 가능성(observability) : 서로의 계약 이행 가능성을 관찰하거나 성과를 입증할 수 있어야 한다.
  - 검증 가능성(verifiability) : 계약을 이행 또는 위반했을 때 이를 알 수 있어야 한다.
  - 사생활 보호(privacy) : 계약 내용은 계약에 관련된 당사자들에게만 알려져야 한다.
  - 강제 가능성(enforceability) : 계약을 강제로 이행할 수 있는 구속력이 있어야 한다. 단, 강제 가능성은 최소화해야 한다.

2 + 3 을 컴퓨터가 계산하는 방법?

Stack Machine을 이용하는 방법

- Stack (LIFO : Last-in First-out) : Push(item), Pop()

- 2 + 3 : 중위법 ➞ 2 3 + : 후위법
- PUSH 2 : 2를 PUSH 해라
  PUSH 3 : 3을 PUSH 해라
  ADD : 2개를 POP 후 더하고 PUSH 해라



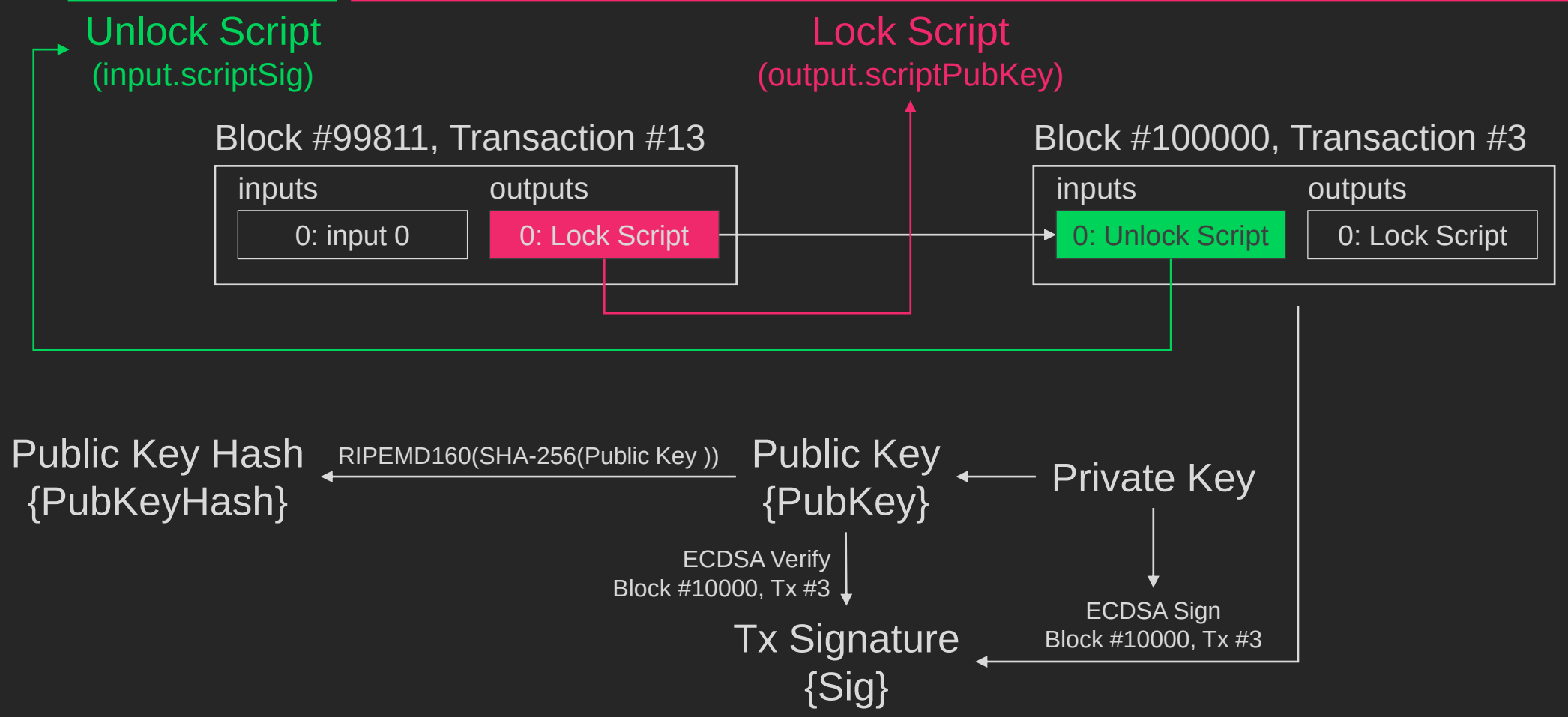
Bitcoin Script

전자 화폐 기능

- 비트코인 : 전자화폐 기능 제공, 비트코인 스크립트로 구현

P2PKH : Pay to Public Key Hash

{Sig} {PubKey} OP_DUP OP_HASH160 {PubKeyHash} OP_EQUALVERIFY OP_CHECKSIG



P2PKH : Pay to Public Key Hash

{Sig} {PubKey} OP_DUP OP_HASH160 {PubKeyHash} OP_EQUALVERIFY OP_CHECKSIG

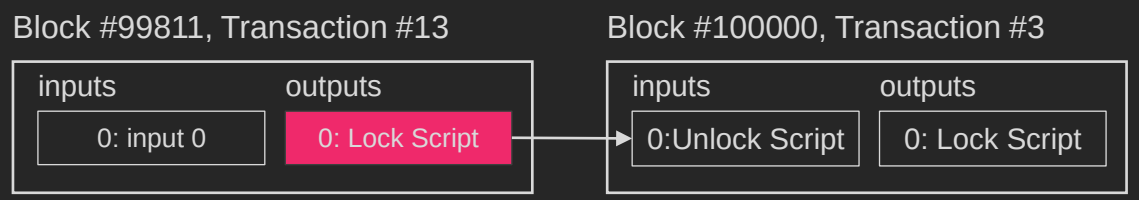
Bitcoin Script (https://en.bitcoin.it/wiki/Script)

연산자	Opcode	Hex	Input	Output	설명
OP_DUP	118	0x76	x	x x	Top 스택 아이템을 복사
OP_HASH160	1690	0xa9	in	hash	RIPEMD-160 (SHA-256 (in)) , pop()한 in 값을 hash 계산 후 push한다.
OP_EQUAL	135	0x87	x1 x2	True / false	두 값이 같으면 True(1), 아니면 false(0)
OP_VERIFY	105	0x69	True / false	Nothing / fail	Top 스택 아이템이 True(1)가 아니면 트랜잭션은 비유효(fail) 탑 스택 아이템은 제거한다.
OP_EQUALVERIFY	136	0x88	x1 x2	Nothing / fail	OP_EQUAL, OP_VERIFY를 순서대로 실행
OP_CHECKSIG	172	0xac	sig pubkey	True / false	sig가 Tx의 hash와 pubkey에 유효하면 True(1), 아니면 false(0)

{Sig}	output을 사용하는 트랜잭션 서명
{PubKey}	output을 사용하는 트랜잭션 서명 증명 및 {PubKeyHash}를 생성할 수 있는 공개 키
{PubKeyHash}	수신자 공개키 Hash (주소 생성 과정의 Hash 값과 동일 Hash = RIPEMD160(SHA-256(Public Key))

Unlock Script [Bitcoin Block #99811, Transaction #13]

https://bitcoinexplorer.org/tx/f4515fed3dc4a19b90a317b9840c243bac26114cf637522373a7d486b372600b



inputs

input 0

Lock script


```
{
  "txid": "f4515fed3dc4a19b90a317b9840c243bac26114cf637522373a7d486b372600b",
  "hash": "f4515fed3dc4a19b90a317b9840c243bac26114cf637522373a7d486b372600b",
  "version": 1,
  "size": 224,
  "vsize": 224,
  "weight": 896,
  "locktime": 0,
  "vin": [
    {
      "txid": "c691e521b77f7e59425d9f51b98c9897f2f40bb6165bc595156fb8e69ef3fce7",
      "vout": 1,
      "scriptSig": {
        "asm": "3045022100ac572b43e78089851202cfd9386750b08afc175318c537f04eb364bf5a0070d402203f0e829d4baea982feaf987cb9f14c85097d2fbe89fba3f283f6925b3214a97e[ALL]
048922fa4dc891f9bb39f315635c03e60e019ff9ec1559c8b581324b4c3b7589a57550f9b0b80bc72d0f959fddf6ca65f07223c37a8499076bd7027ae5c325fac5",
        "hex":
"483045022100ac572b43e78089851202cfd9386750b08afc175318c537f04eb364bf5a0070d402203f0e829d4baea982feaf987cb9f14c85097d2fbe89fba3f283f6925b3214a97e0141048922fa4dc891f9bb39f315635c03e6
0e019ff9ec1559c8b581324b4c3b7589a57550f9b0b80bc72d0f959fddf6ca65f07223c37a8499076bd7027ae5c325fac5"
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 c4eb47ecfdcf609a1848ee79acc2fa49d3caad70 OP_EQUALVERIFY OP_CHECKSIG",
        "desc": "addr(1JxDJCyWNakZ5kECKdCU9Zka6mh34mZ7B2)#pk74vhp9",
        "hex": "76a914c4eb47ecfdcf609a1848ee79acc2fa49d3caad7088ac",
        "address": "1JxDJCyWNakZ5kECKdCU9Zka6mh34mZ7B2",
        "type": "pubkeyhash"
      }
    }
  ],
  "hex":
"0100000001e7fcf39ee6b86f1595c55b16b60bf4f297988cb9519f5d42597e7fb721e591c6010000008b483045022100ac572b43e78089851202cfd9386750b08afc175318c537f04eb364bf5a0070d402203f0e829d4baea982
feaf987cb9f14c85097d2fbe89fba3f283f6925b3214a97e0141048922fa4dc891f9bb39f315635c03e60e019ff9ec1559c8b581324b4c3b7589a57550f9b0b80bc72d0f959fddf6ca65f07223c37a8499076bd7027ae5c325fac
5ffffffff0140420f00000000001976a914c4eb47ecfdcf609a1848ee79acc2fa49d3caad7088ac00000000",
  "blockhash": "0000000000004273d89d3f220a9d3dbd45a4fd1c028b51e170f478da11352187",
  "confirmations": 636544,
  "time": 1293528528,
  "blocktime": 1293528528
}
```

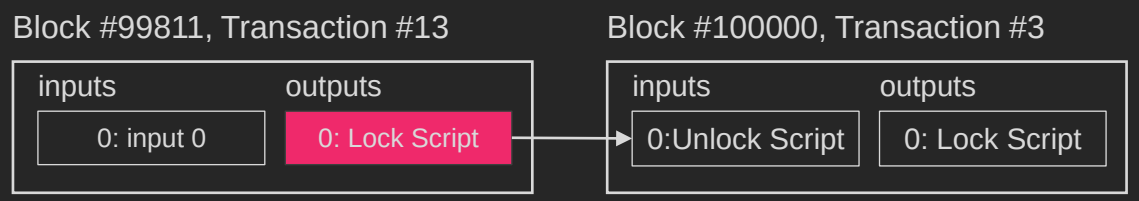

outputs

output 0

Lock script

Unlock Script [Bitcoin Block #99811, Transaction #13]

https://bitcoinexplorer.org/tx/f4515fed3dc4a19b90a317b9840c243bac26114cf637522373a7d486b372600b



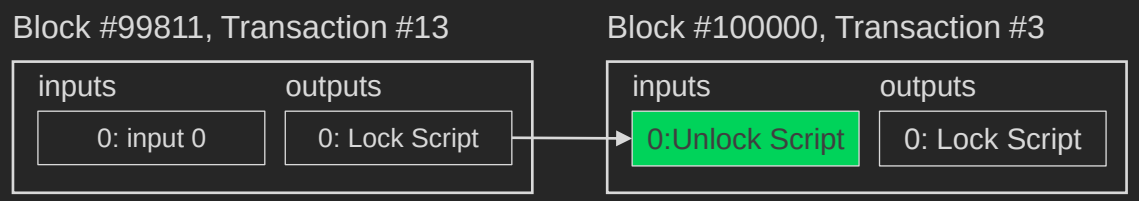

```
"vout": [
  {
    "value": 0.01,      // 전송할 bitcoin
    "n": 0,             // output index
    "scriptPubKey": {   // lock script
      "asm": "OP_DUP OP_HASH160 c4eb47ecfdcf609a1848ee79acc2fa49d3caad70 OP_EQUALVERIFY OP_CHECKSIG",
      "desc": "addr(1JxDJCyWNakZ5kECKdCU9Zka6mh34mZ7B2)#pk74vhp9",
      "hex": "76a914c4eb47ecfdcf609a1848ee79acc2fa49d3caad7088ac",
      "address": "1JxDJCyWNakZ5kECKdCU9Zka6mh34mZ7B2",
      "type": "pubkeyhash"
    }
  }
]
```


Lock Script (ouput.script.PubKey)

OP_DUP OP_HASH160 c4eb47ecfdcf609a1848ee79acc2fa49d3caad70 OP_EQUALVERIFY OP_CHECKSIG

Lock Script [Bitcoin Block #100000, Transaction #3]

https://bitcoinexplorer.org/tx/e9a66845e05d5abc0ad04ec80f774a7e585c6e8db975962d069a522137b80c1d




```
{
  "in_active_chain": true,
  "txid": "e9a66845e05d5abc0ad04ec80f774a7e585c6e8db975962d069a522137b80c1d",
  "hash": "e9a66845e05d5abc0ad04ec80f774a7e585c6e8db975962d069a522137b80c1d",
  "version": 1,
  "size": 225,
  "vsize": 225,
  "weight": 900,
  "locktime": 0,
  "vin": [
    {
      "txid": "f4515fed3dc4a19b90a317b9840c243bac26114cf637522373a7d486b372600b",
      "vout": 0,
      "scriptSig": {
        "asm": "3046022100bb1ad26df930a51cce110cf44f7a48c3c561fd977500b1ae5d6b6fd13d0b3f4a022100c5b42951acedff14abba2736fd574bdb465f3e6f8da12e2c5303954aca7f78f3[ALL]
04a7135bfe824c97ecc01ec7d7e336185c81e2aa2c41ab175407c09484ce9694b44953fcb751206564a9c24dd094d42fdbfdd5aad3e063ce6af4cfaaea4ea14fbb",
        "hex":
"493046022100bb1ad26df930a51cce110cf44f7a48c3c561fd977500b1ae5d6b6fd13d0b3f4a022100c5b42951acedff14abba2736fd574bdb465f3e6f8da12e2c5303954aca7f78f3014104a7135bfe824c97ecc01ec7d7e336
185c81e2aa2c41ab175407c09484ce9694b44953fcb751206564a9c24dd094d42fdbfdd5aad3e063ce6af4cfaaea4ea14fbb"
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 39aa3d569e06a1d7926dc4be1193c99bf2eb9ee0 OP_EQUALVERIFY OP_CHECKSIG",
        "desc": "addr(16FuTPaeRSPVxxCnwQmdyx2PQWxX6HWzhQ)#r33agg0y",
        "hex": "76a91439aa3d569e06a1d7926dc4be1193c99bf2eb9ee088ac",
        "address": "16FuTPaeRSPVxxCnwQmdyx2PQWxX6HWzhQ",
        "type": "pubkeyhash"
      }
    }
  ],
  "hex":
"01000000010b6072b386d4a773235237f64c1126ac3b240c84b917a3909ba1c43ded5f51f4000000008c493046022100bb1ad26df930a51cce110cf44f7a48c3c561fd977500b1ae5d6b6fd13d0b3f4a022100c5b42951acedff
14abba2736fd574bdb465f3e6f8da12e2c5303954aca7f78f3014104a7135bfe824c97ecc01ec7d7e336185c81e2aa2c41ab175407c09484ce9694b44953fcb751206564a9c24dd094d42fdbfdd5aad3e063ce6af4cfaaea4ea14
fbbffffffff0140420f00000000001976a91439aa3d569e06a1d7926dc4be1193c99bf2eb9ee088ac00000000",
  "blockhash": "000000000003ba27aa200b1cecaad478d2b00432346c3f1f3986da1afd33e506",
  "confirmations": 636355,
  "time": 1293623863,
  "blocktime": 1293623863
}
```


inputs

input 0

Unlock script

outputs

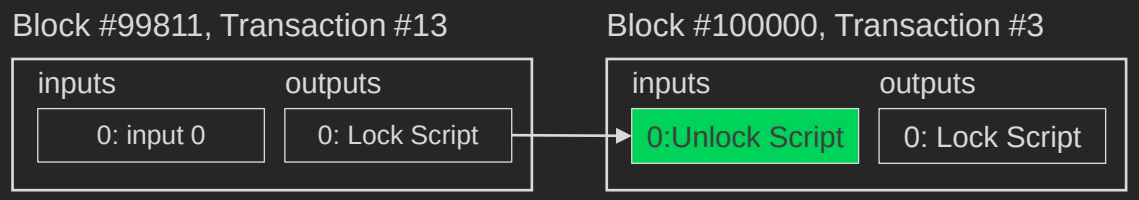
output 0

Lock script

Lock Script [Bitcoin Block #100000, Transaction #3]

https://bitcoinexplorer.org/tx/e9a66845e05d5abc0ad04ec80f774a7e585c6e8db975962d069a522137b80c1d


```
"vin": [
  {
    "txid": "f4515fed3dc4a19b90a317b9840c243bac26114cf637522373a7d486b372600b", // output tx hash
    "vout": 0,                                                                  // output index
    "scriptSig": {                                                              // unlock script
      "asm": "3046022100bb1ad26df930a51cce110cf44f7a48c3c561fd977500b1ae5d6b6fd13d0b3f4a022100c5b4295
              1acedff14abba2736fd574bdb465f3e6f8da12e2c5303954aca7f78f3[ALL]
              04a7135bfe824c97ecc01ec7d7e336185c81e2aa2c41ab175407c09484ce9694b44953fcb751206564a9c24
              dd094d42fdbfdd5aad3e063ce6af4cfaaea4ea14fbb",
      "hex": "493046022100bb1ad26df930a51cce110cf44f7a48c3c561fd977500b1ae5d6b6fd13d0b3f4a022100c5b42
              951acedff14abba2736fd574bdb465f3e6f8da12e2c5303954aca7f78f3014104a7135bfe824c97ecc01ec7
              d7e336185c81e2aa2c41ab175407c09484ce9694b44953fcb751206564a9c24dd094d42fdbfdd5aad3e063c
              e6af4cfaaea4ea14fbb"
    },
    "sequence": 4294967295
  }
],
```

Unlock Script (input.scriptSig)

3046022100bb1ad26df930a51cce110cf44f7a48c3c561fd977500b1ae5d6b6fd13d0b3f4a022100c5b42951acedff14abba2736fd574bdb465f3e6f8da12e2c5303954aca7f78f3
04a7135bfe824c97ecc01ec7d7e336185c81e2aa2c41ab175407c09484ce9694b44953fcb751206564a9c24dd094d42fdbfdd5aad3e06

P2PKH : Pay to Public Key Hash

{Sig} {PubKey} OP_DUP OP_HASH160 {PubKeyHash} OP_EQUALVERIFY OP_CHECKSIG

Unlock Script
(input.scriptSig)

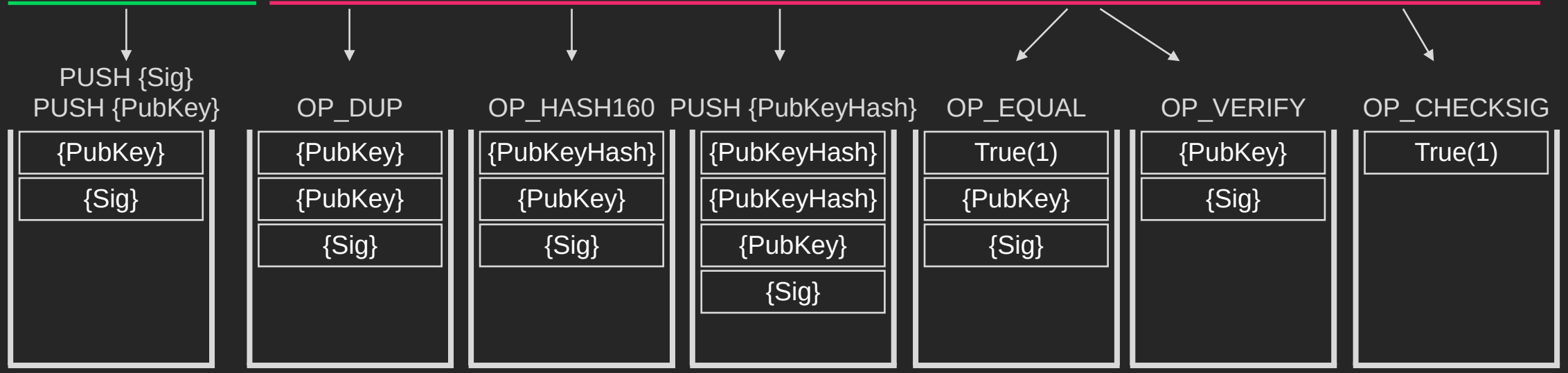
Lock Script
(output.scriptPubKey)

Sig	3046022100bb1ad26df930a51cce110cf44f7a48c3c561fd977500b1ae5d6b6fd13d0b3f4a022100c5b42951acedff14abba2736fd574bdb465f3e6f8da12e2c5303954aca7f78f3
PubKey	04a7135bfe824c97ecc01ec7d7e336185c81e2aa2c41ab175407c09484ce9694b44953fcb751206564a9c24dd094d42fdbfdd5aad3e06
PubkeyHash	c4eb47ecfdcf609a1848ee79acc2fa49d3caad70

연산자	Opcode	Hex	Input	Output	설명
OP_DUP	118	0x76	x	x x	Top 스택 아이템을 복사
OP_HASH160	1690	0xa9	in	hash	RIPEMD-160 (SHA-256 (in)) , pop()한 in 값을 hash 계산 후 push한다.
OP_EQUAL	135	0x87	x1 x2	True / false	두 값이 같으면 True(1), 아니면 false(0)
OP_VERIFY	105	0x69	True / false	Nothing / fail	Top 스택 아이템이 True(1)가 아니면 트랜잭션은 비유효(fail) 탑 스택 아이템은 제거한다.
OP_EQUALVERIFY	136	0x88	x1 x2	Nothing / fail	OP_EQUAL, OP_VERIFY를 순서대로 실행
OP_CHECKSIG	172	0xac	sig pubkey	True / false	sig가 Tx의 hash와 pubkey에 유효하면 True(1), 아니면 false(0)

P2PKH : Pay to Public Key Hash

{Sig} {PubKey} OP_DUP OP_HASH160 {PubKeyHash} OP_EQUALVERIFY OP_CHECKSIG

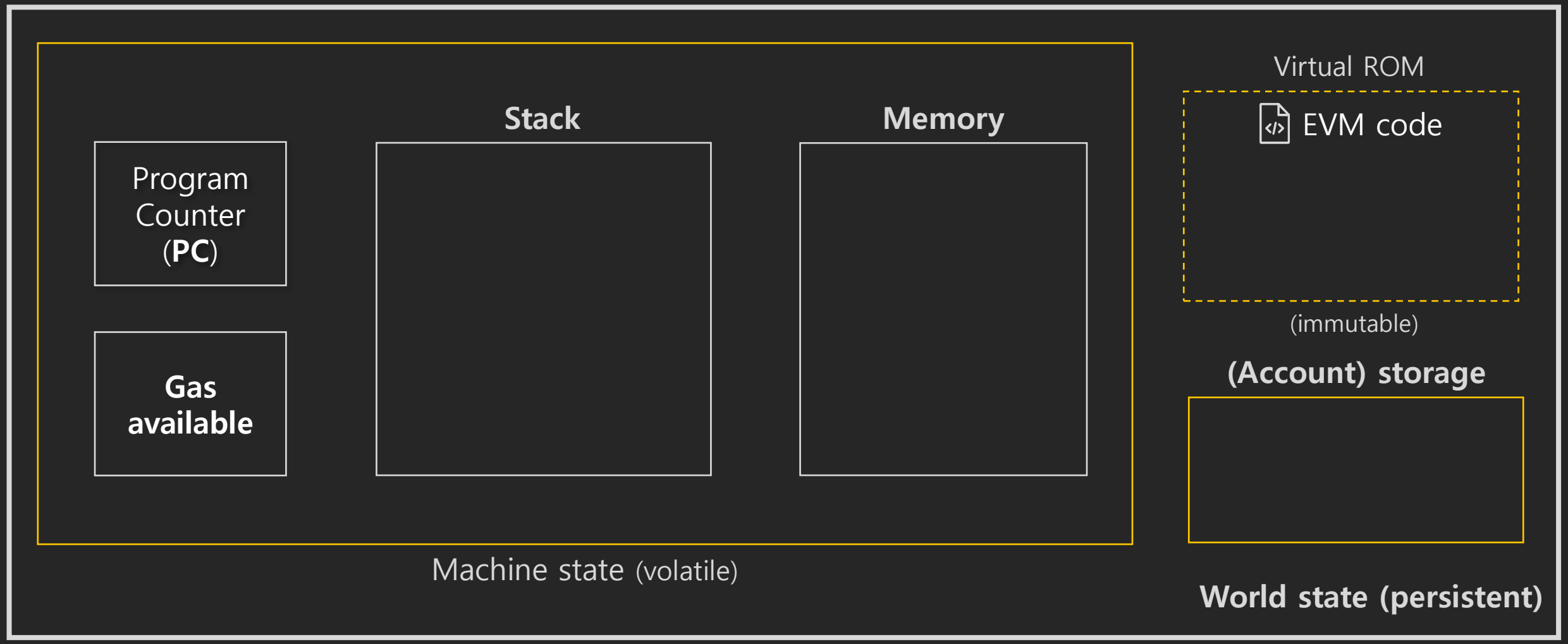


연산자	Opcode	Hex	Input	Output	설명
OP_DUP	118	0x76	x	x x	Top 스택 아이템을 복사
OP_HASH160	1690	0xa9	in	hash	RIPEMD-160 (SHA-256 (in)) , pop()한 in 값을 hash 계산 후 push한다.
OP_EQUAL	135	0x87	x1 x2	True / false	두 값이 같으면 True(1), 아니면 false(0)
OP_VERIFY	105	0x69	True / false	Nothing / fail	Top 스택 아이템이 True(1)가 아니면 트랜잭션은 비유효(fail) 탑 스택 아이템은 제거한다.
OP_EQUALVERIFY	136	0x88	x1 x2	Nothing / fail	OP_EQUAL, OP_VERIFY를 순서대로 실행
OP_CHECKSIG	172	0xac	sig pubkey	True / false	sig가 Tx의 hash와 pubkey에 유효하면 True(1), 아니면 false(0)

Etheretum Smart Contract

EVM (Ethereum Virtual Machine)

Smart Contract Runtime Environment



- Solidity
  - EVM 에서 동작하는 bytecode 작성 프로그래밍 언어

종이 계약서

스마트 계약(Token.sol)

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

contract Token {
    mapping(address => uint256) balances;

    function transfer(address to, uint256 value) public returns(bool success) {
        require(balances[msg.sender] > value, "balance is insufficient");

        balances[msg.sender] -= value;
        balances[to] += value;

        emit Transfer(msg.sender, to, value);

        return true;
    }

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

이더리움 스마트 컨트랙트

- 배포 되면 변경 불가
- 불변적(immutable) ➞ EVM 상에서 결정론적(deterministically) 작동
  - 결정론적(deterministic) : 동일 상태 & 동일 코드 & 동일 입력 ➞ 항상 동일 결과
  - 시스템 전체는 탈중앙화된 단일 월드 컴퓨터
  - 병렬적으로 실행되지 않는다 ➞ 단일 스레드 컴퓨터
  - 원자성 : 모든 실행 성공해야 변경사항 기록 ➞ 오류 ➞ rollback (단, 실행을 위해 가스로 소비된 이더는 차감)
- EVM
  - 유사 튜링 완전 상태 머신(quasi-Turing-complete state machine)
    ➞ 어떤 종류의 프로그램도 실행 가능
  - 매우 제한적인 실행 컨텍스트에서 작동
    ➞ 자신의 상태, 호출한 트랜잭션의 컨텍스트 및 가장 최근 블록의 일부 정보에 접근
  - 정지 문제(halting problem) : 프로그램이 영원히 실행 되는 문제, 프로그램만으로 영원히 실행되는지 여부를 알 수 없음

- 가스 (Gas)
  - 작업 수행에 필요한 계산 및 스토리지 자원 측정 단위
  - 연산자(OpCode) 수행, 데이터 조회, 저장 등에서 가스를 소비
  - 트랜잭션 : 최대 사용 가스 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;
            }
        }
    }
}
```

ERC-20

(Fungible) Token Standard

ERC-721

NFT, Non-fungible Token

Marketplace

(/w ERC-20, ERC-721)

ERC-20 (Fungible) Token Standard

- (대체 가능) 토큰 표준 인터페이스
- 대체 가능 토큰
  - 예) 화폐. 같은 금액 지폐 2장은 실제로 다른 물건이나 같은 가치로 교환 가능

```
//// Method
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256 balance)
function transfer(address _to, uint256 _value) public returns (bool success)
function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)
function approve(address _spender, uint256 _value) public returns (bool success)
function allowance(address _owner, address _spender) public view returns (uint256 remaining)

//// Event
event Transfer(address indexed _from, address indexed _to, uint256 _value)
event Approval(address indexed _owner, address indexed _spender, uint256 _value)
```

ERC-20 Methods

Method	Description
name()	토큰 명 조회. 예) Wrapped Ether
symbol()	토큰 심볼 조회. 예) WETH
decimals()	소수점 자리. 예) 18 데이터 1 = 0.000000000000000001 WETH 데이터 1000000000000000000 = 1 WETH
totalSupply()	총 공급 량. 예) 5834832821676581767963986
balanceOf(address owner)	owner의 잔액 조회
transfer(address to, uint256 value)	to에게 value 만큼 전송
transferFrom(address from, address to, uint256 value)	from으로부터 to에게 value 만큼 전송
approve(address spender, uint256 value)	spender가 value만큼 사용하도록 승인
allowance(address owner, address spender)	spender가 사용가능한 owner은 토큰 수량 조회

balances[{address}] = {balance}

balances[A] = 1000

A의 잔액(balance)는 1000

allowances[{owner}][{spender}] = {value}

allowances[A][C] = 1000

C는 A의 토큰을 1000 만큼 사용가능

balances[{address}] = {balance}
allowances[{owner}][{spender}] = {amount}

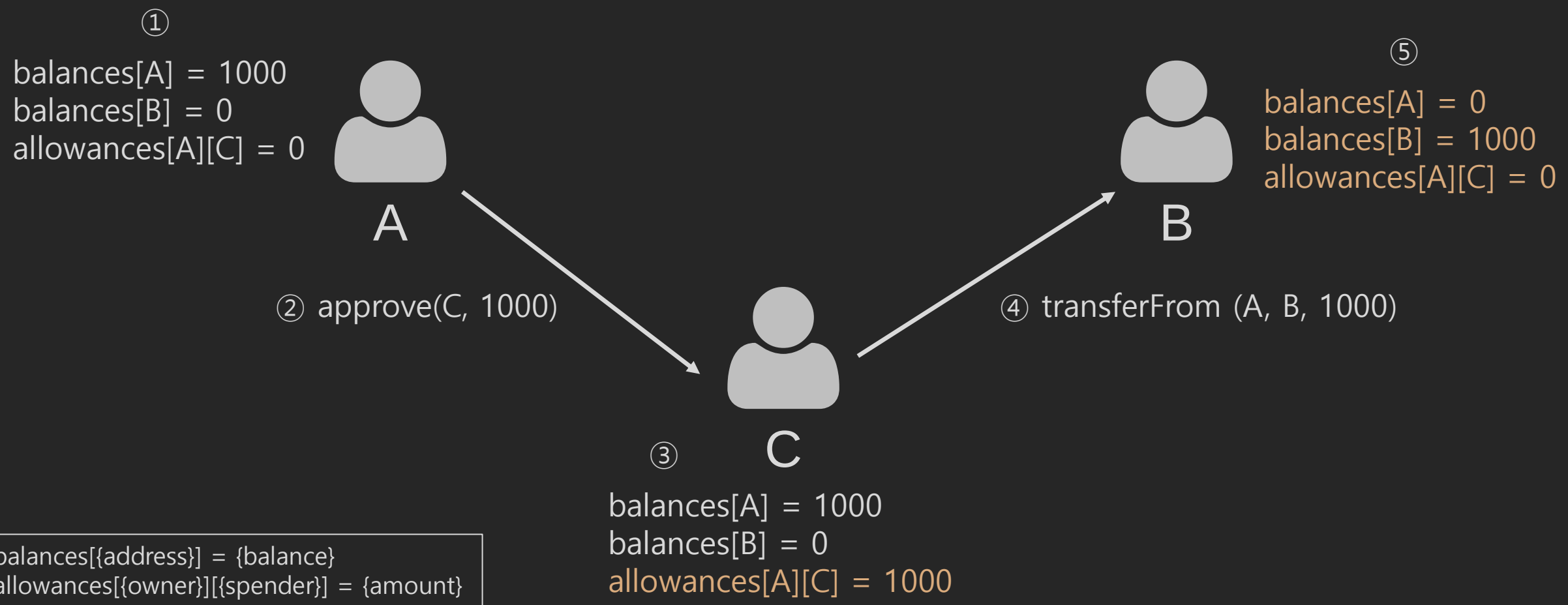
balances[A] = 1000 ➞ A의 잔액(balance)는 1000
allowances[A][C] = 1000 ➞ C는 A의 토큰을 1000 만큼 사용가능

transfer(to, value) ➞ A : transfer(B, 1000)

balances[{address}] = {balance}
allowances[{owner}][{spender}] = {amount}

approve(spender, value) ➞ A : approve(C, 1000)

transferFrom(from, to, value) ➞ C : transferFrom(A, B, 1000)



balances[{address}] = {balance}
allowances[{owner}][{spender}] = {amount}

ERC-721 NFTs(Non-Fungible) Token Standard

- 대체 불가 토큰 표준 인터페이스
  - 예) 그림, 이미지, 게임 아이템 등
- TokenId 별 소유자 관리. TokenId 발급 예) 0부터 1씩 증가하면 발급

```
//// Methods
function balanceOf(address _owner) external view returns (uint256);
function ownerOf(uint256 _tokenId) external view returns (address);
function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external payable;
function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;
function transferFrom(address _from, address _to, uint256 _tokenId) external payable;
function approve(address _approved, uint256 _tokenId) external payable;
function setApprovalForAll(address _operator, bool _approved) external;
function getApproved(uint256 _tokenId) external view returns (address);
function isApprovedForAll(address _owner, address _operator) external view returns (bool);

//// Event
event Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);
event Approval(address indexed _owner, address indexed _approved, uint256 indexed _tokenId);
event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);
```

ERC-721 Methods

Method	Description
balanceOf(address owner)	소유 토큰 개수
ownerOf(uint256 tokenId)	TokenId 소유자 조회
safeTransferFrom (address from, address to, uint256 tokenId, bytes data)	컨트랙트로 토큰 전송. 전송받는 컨트랙트 ERC721TokenReceiver 구현 필요 전송 후 ERC721TokenReceiver.onERC721Received() 호출
safeTransferFrom(address from, address to, uint256 tokenId)	data가 "" 인 safeTransferFrom()
transferFrom(address from, address to, uint256 tokenId)	from으로 부터 to로 토큰 Id가 tokenId인 토큰 전송
approve(address approved, uint256 tokenId)	approved에게 tokenId 토큰 사용 승인
setApprovalForAll(address operator, bool approved)	operator에게 전체 사용 승인 여부 설정
getApproved(uint256 tokenId)	tokenId의 사용 승인된 approved 조회
isApprovedForAll(address owner, address operator)	owner의 operator에 대한 전체 사용 승인 여부 조회

owners[{tokenId}] = {address}

owners[0] = A

tokenId 0의 소유자는 A

approvals[{tokenId}] = {address}

approvals[0] = C

C는 tokenId 0의 사용을 승인 받음

owners[{tokenId}] = {address}

approvals[{tokenId}] = {address}

owners[0] = A ➞ tokenId 0의 소유자는 A

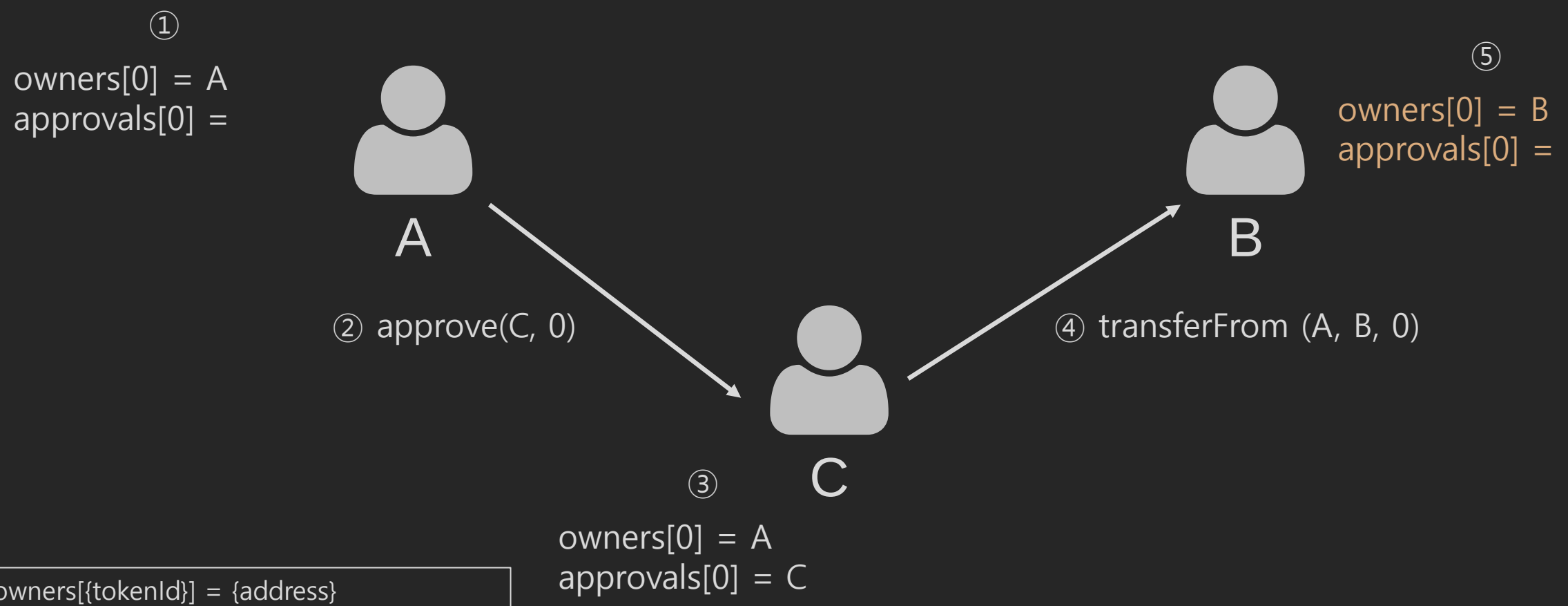
approvals[0] = C ➞ C는 tokenId 0의 사용을 승인 받음

transferFrom(from, to, tokenId) ➞ A : transferFrom(A, B, 0)

owners[{tokenId}] = {address}
approvals[{tokenId}] = {address}

approve(approved, tokenId) ➞ A : approve(C, 0)

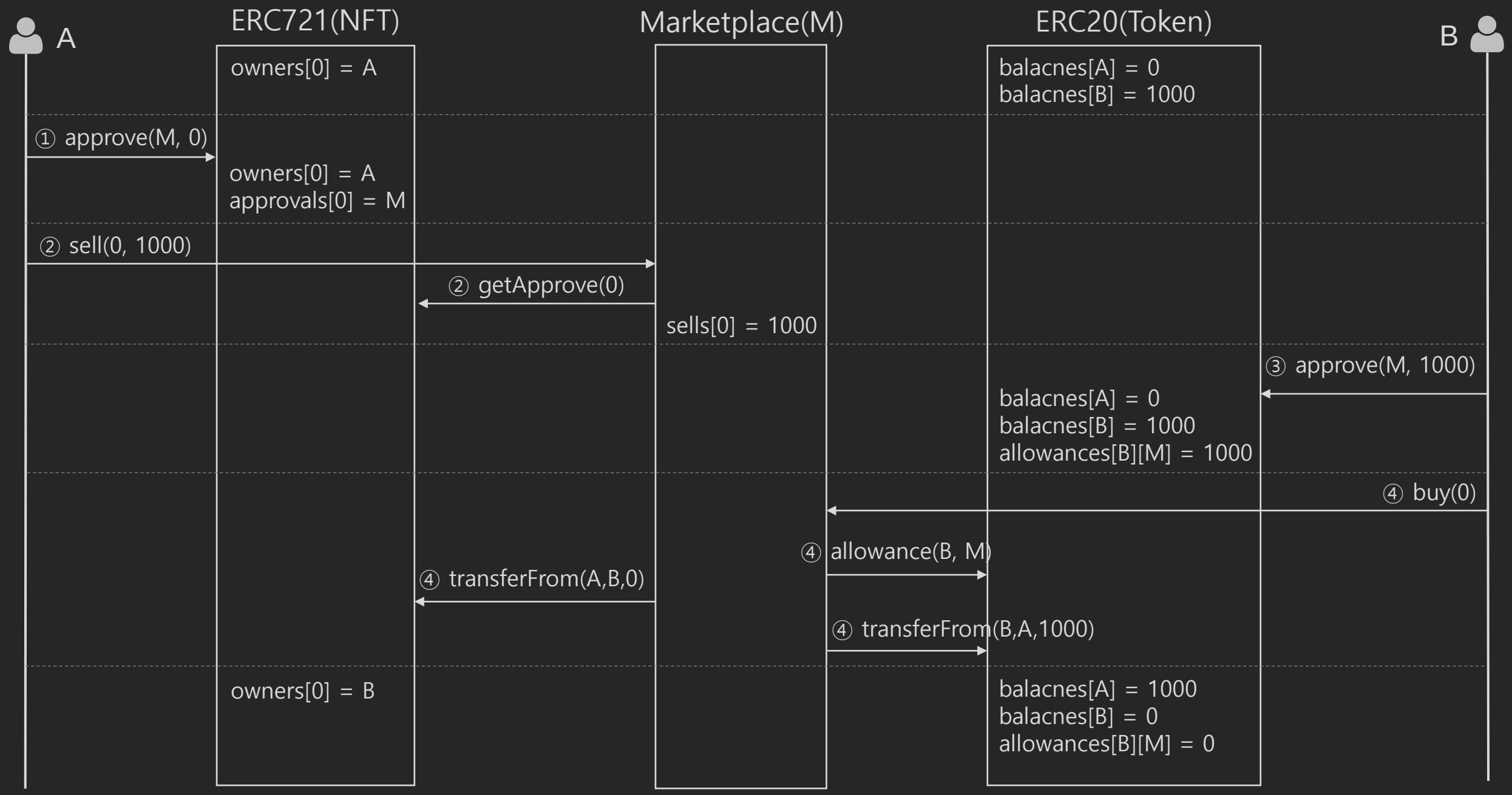
transferFrom(from, to, tokenId) ➞ C : transferFrom(A, B, 0)



owners[{tokenId}] = {address}
approvals[{tokenId}] = {address}

Marketplace

- ERC20(Token)와 ERC721(NFT)의 활용
- NFT 소유자가 Sell(tokenId, price)을 하면 판매 목록에 등록
- Token 소유자는 Buy(tokenId)로 NFT 구매
- A는 NFT(tokenId = 0) 을 Token 1000에 판매 희망
- B는 Token 1000을 가지고 있고, NFT(tokenId = 0) 을 구매 희망


A
ERC721(NFT)
Marketplace(M)
ERC20(Token)
B
owners[0] = A
balacnes[A] = 0
balacnes[B] = 1000
① approve(M, 0)
owners[0] = A
approvals[0] = M
② sell(0, 1000)
② getApprove(0)
sells[0] = 1000
③ approve(M, 1000)
balacnes[A] = 0
balacnes[B] = 1000
allowances[B][M] = 1000
④ buy(0)
④ allowance(B, M)
④ transferFrom(A,B,0)
④ transferFrom(B,A,1000)
owners[0] = B
balacnes[A] = 1000
balacnes[B] = 0
allowances[B][M] = 0

dApp
(Decentralized Application)

dApp (Decentralized Application)

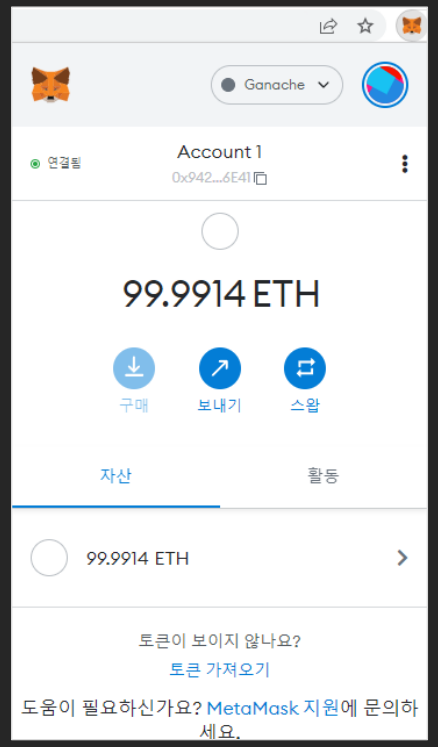
- 탈중앙화 분산 애플리케이션
- 블록체인을 기반으로 동작하는 애플리케이션
- 좁은 의미 : 블록체인 상에서만 동작하는 스마트 컨트랙트
- 넓은 의미 : 스마트 컨트랙트와 상호작용하는 애플리케이션을 포함

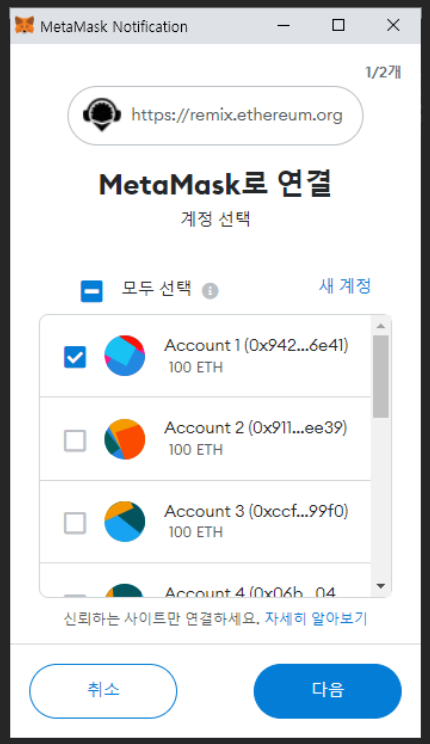
- 이더리움 JSON-RPC
  - Ethereum JSON-RPC Specification (JSON-RPC)
  - Http, WebSocket - CORS 지원

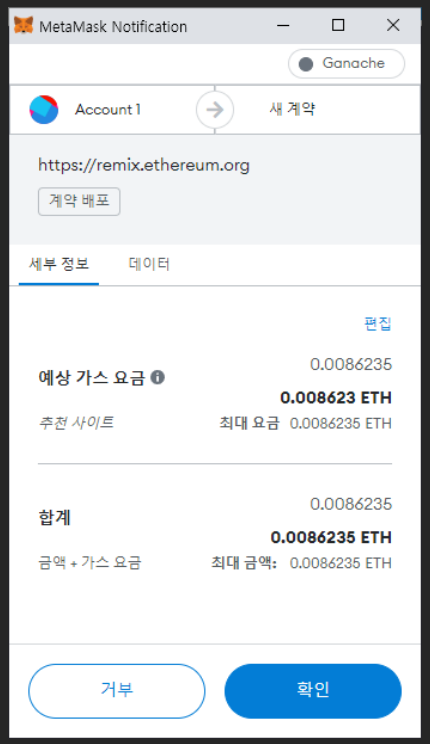
Method	Description
eth_getBalance	이더리움 잔액 조회
eth_call	조회
eth_sendRawTransaction	트랜잭션 제출

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

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







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

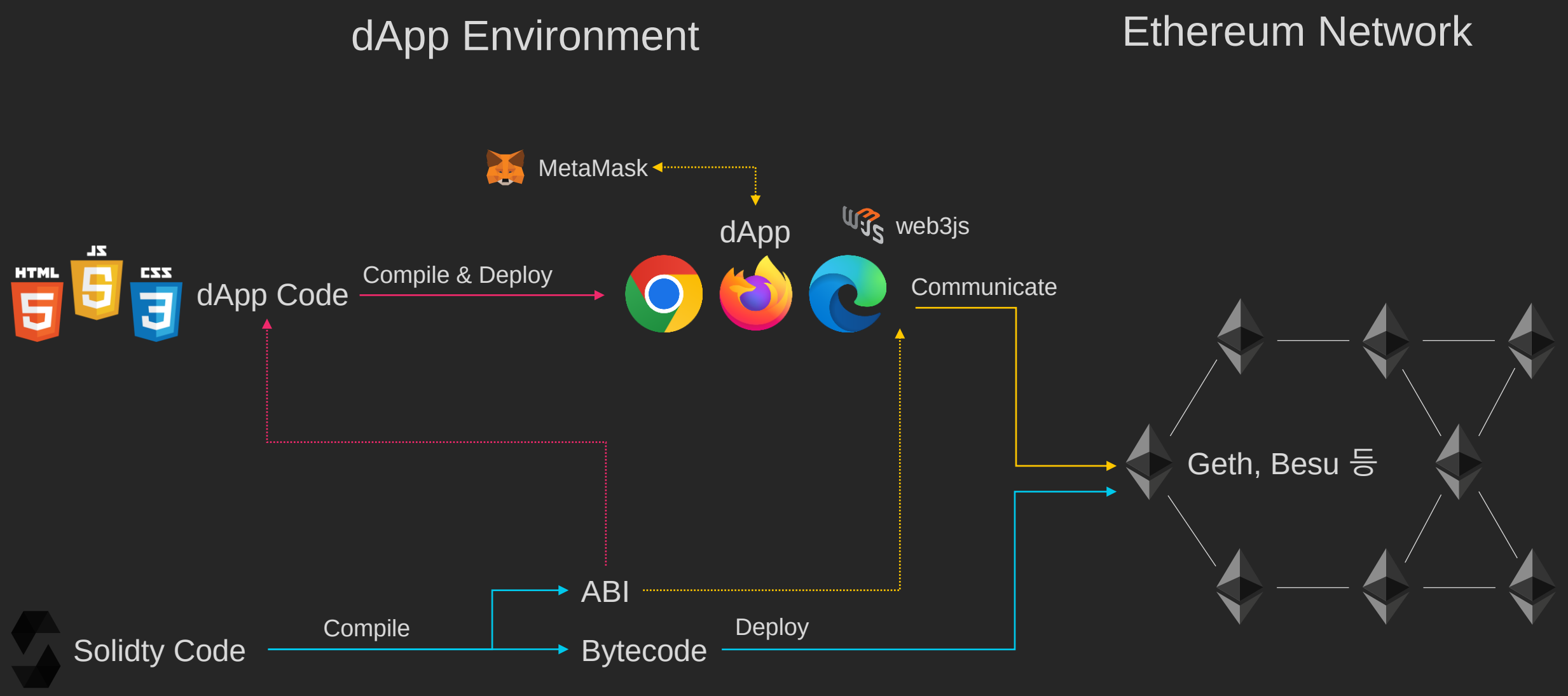
```javascript
import Web3 from 'web3';
import { createRequire } from "module";
const require = createRequire(import.meta.url);
const helloWorldAbi = require("./contract/HelloWorld.json");

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

```solidity
// 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)

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


dApp Environment
Ethereum Network
MetaMask
dApp
web3js
HTML
JS
CSS
dApp Code
Compile & Deploy
Communicate
Geth, Besu 등
ABI
Solidty Code
Compile
Bytecode
Deploy

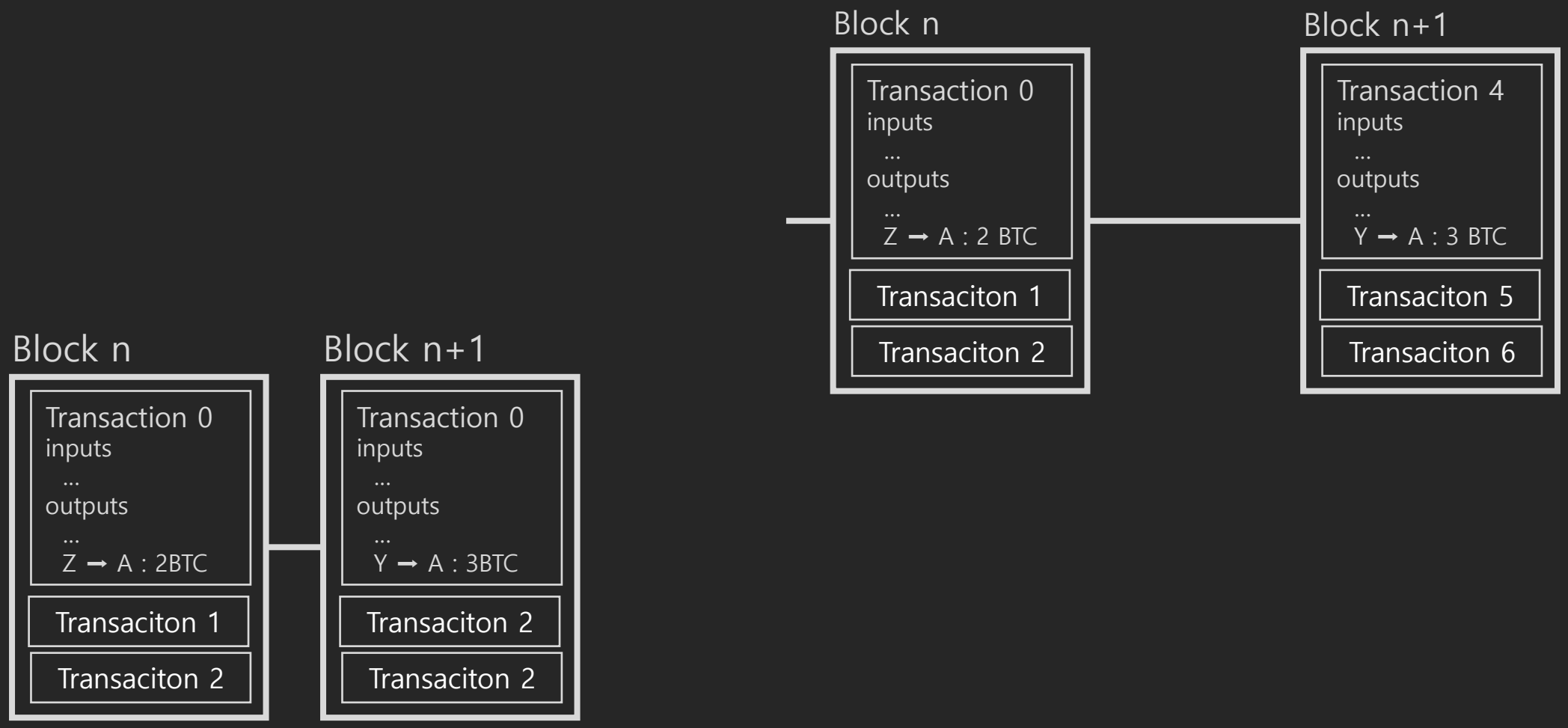
- 가상자산 거래소에서 무엇을 매수, 매도하는 것인가?
  - 업비트 이용약관 - 제 2조 (정의)
    - 5. "디지털 자산"이라 함은 서비스에서 거래할 수 있는 비트코인, 이더리움 등 블록체인에서 전자적으로 존재하는 정보로 서비스의 대상물을 말합니다. 회원은 회사에 대하여 계정 내 "디지털 자산"의 수량만큼의 출금 청구권을 보유합니다.

기본 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ

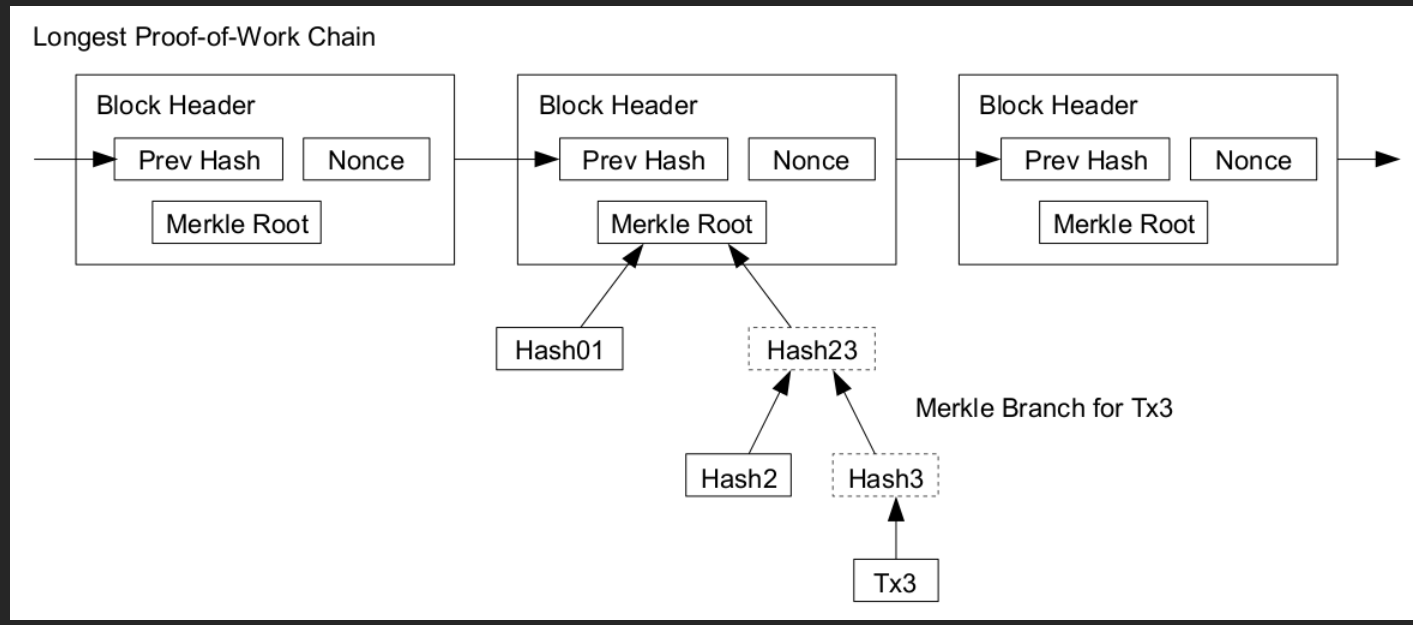
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
포인트 색깔 가나다라마바사아자차카타파하 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ

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


Block n
Transaction 0
inputs
...
outputs
...
Z → A : 2BTC
Transaciton 1
Transaciton 2
Block n+1
Transaction 0
inputs
...
outputs
...
Y → A : 3BTC
Transaciton 2
Transaciton 2
Block n
Transaction 0
inputs
...
outputs
...
Z → A : 2 BTC
Transaciton 1
Transaciton 2
Block n+1
Transaction 4
inputs
...
outputs
...
Y → A : 3 BTC
Transaciton 5
Transaciton 6

비트코인 블록헤더, 이더리움 블록헤더



이더리움 besu total difficulty

```
private Difficulty calculateTotalDifficulty(final BlockHeader blockHeader) {
  if (blockHeader.getNumber() == BlockHeader.GENESIS_BLOCK_NUMBER) {
    return blockHeader.getDifficulty();
  }

  final Difficulty parentTotalDifficulty =
      blockchainStorage
          .getTotalDifficulty(blockHeader.getParentHash())
          .orElseThrow(
              () -> new IllegalStateException("Blockchain is missing total difficulty
data."));
  return blockHeader.getDifficulty().add(parentTotalDifficulty);
}
```