Bitcoin Developer Meetup

비트코인 노드 동기화의 현재와 미래:
SwiftSync로 보는 확장성의 방향

https://github.com/kcalvinalvin/2026-02-21-bitcoin-center-meetup

저에 대해서

개발자가 된 길

- 2017년 서울 비트코인 밋업 참여 (meetup.com/seoulbitcoin)

저에 대해서

개발자가 된 길

- 2017년 서울 비트코인 밋업 참여 (meetup.com/seoulbitcoin)
- 2019년 오픈소스 비트코인 기여 시작

저에 대해서

개발자가 된 길

- 2017년 서울 비트코인 밋업 참여 (meetup.com/seoulbitcoin)
- 2019년 오픈소스 비트코인 기여 시작
- 2020년 비트코인 개발 지원금 받기 시작

목차

발표 개요

1. 비트코인 노드 동기화의 본질과 현재 구조

“Of Bitcoin’s many properties, trustlessness, or the ability to use Bitcoin without trusting anything but the open-source software you run, is, by far, king”

https://bluematt.bitcoin.ninja/2017/02/28/bitcoin-trustlessness/

“비트코인의 많은 속성들 중에서, 무 신뢰성, 즉 스스로 운영하는 오픈소스 소프트웨어 외에는 아무것도 신뢰하지 않고 비트코인을 사용할 수 있는 속성이 최우선이다”

https://bluematt.bitcoin.ninja/2017/02/28/bitcoin-trustlessness/

무 신뢰성 비트코인 사용법

풀 노드 실행

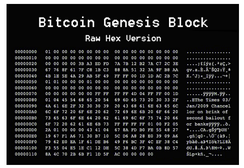
Trustlessness 도달

- Full node = fully validating
- 블록 및 트랜잭션 스스로 검증

Bitcoin Genesis Block

Raw Hex Version

```
00000000   01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000010   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000020   00 00 00 00 3B A3 ED FD  7A 7B 12 B2 7A C7 2C 3E   ....;£íýz{.²zÇ,>
00000030   67 76 8F 61 7F C8 1B C3  88 8A 51 32 3A 9F B8 AA   gv.a.È.Ã^ŠQ2:Ÿ¸ª
00000040   4B 1E 5E 4A 29 AB 5F 49  FF FF 00 1D 1D AC 2B 7C   K.^J)«_Iÿÿ...¬+|
00000050   01 01 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ................
00000060   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000070   00 00 00 00 00 00 FF FF  FF FF 4D 04 FF FF 00 1D   ......ÿÿÿÿM.ÿÿ..
00000080   01 04 45 54 68 65 20 54  69 6D 65 73 20 30 33 2F   ..EThe Times 03/
00000090   4A 61 6E 2F 32 30 30 39  20 43 68 61 6E 63 65 6C   Jan/2009 Chancel
000000A0   6C 6F 72 20 6F 6E 20 62  72 69 6E 6B 20 6F 66 20   lor on brink of
000000B0   73 65 63 6F 6E 64 20 62  61 69 6C 6F 75 74 20 66   second bailout f
000000C0   6F 72 20 62 61 6E 6B 73  FF FF FF FF 01 00 F2 05   or banksÿÿÿÿ..ò.
000000D0   2A 01 00 00 00 43 41 04  67 8A FD B0 FE 55 48 27   *....CA.gŠý°þUH'
000000E0   19 67 F1 A6 71 30 B7 10  5C D6 A8 28 E0 39 09 A6   .gñ¦q0·.\Ö¨(à9.¦
000000F0   79 62 E0 EA 1F 61 DE B6  49 F6 BC 3F 4C EF 38 C4   ybàê.aÞ¶Iö¼?Lï8Ä
00000100   F3 55 04 E5 1E C1 12 DE  5C 38 4D F7 BA 0B 8D 57   óU.å.Á.Þ\8M÷º..W
00000110   8A 4C 70 2B 6B F1 1D 5F  AC 00 00 00 00            ŠLp+kñ._¬....
```


Bitcoin Genesis Block
Raw Hex Version

Bitcoin Core

Synchronizing

42.61%

Synchronized

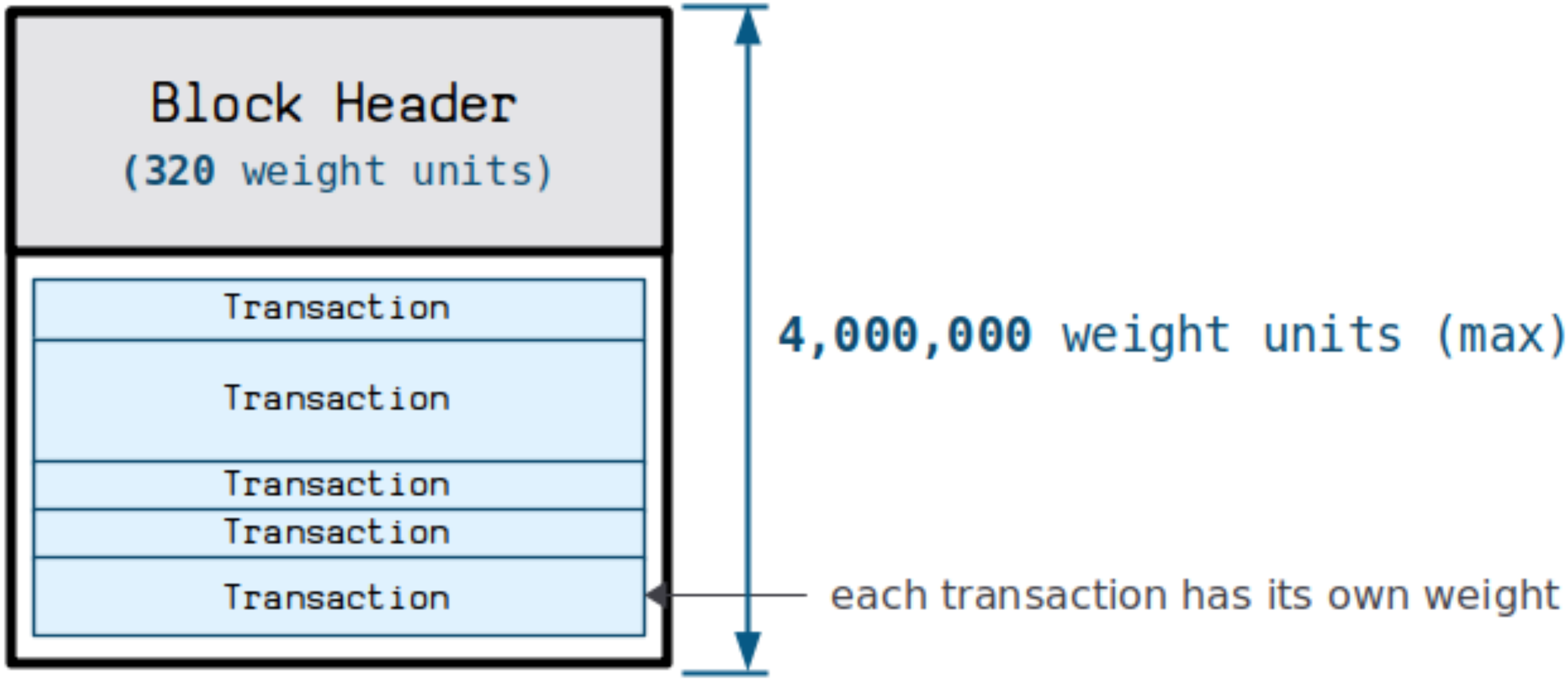
Manage

검증 과정

검증에서 시간이 제일 많이 소모되는 부분

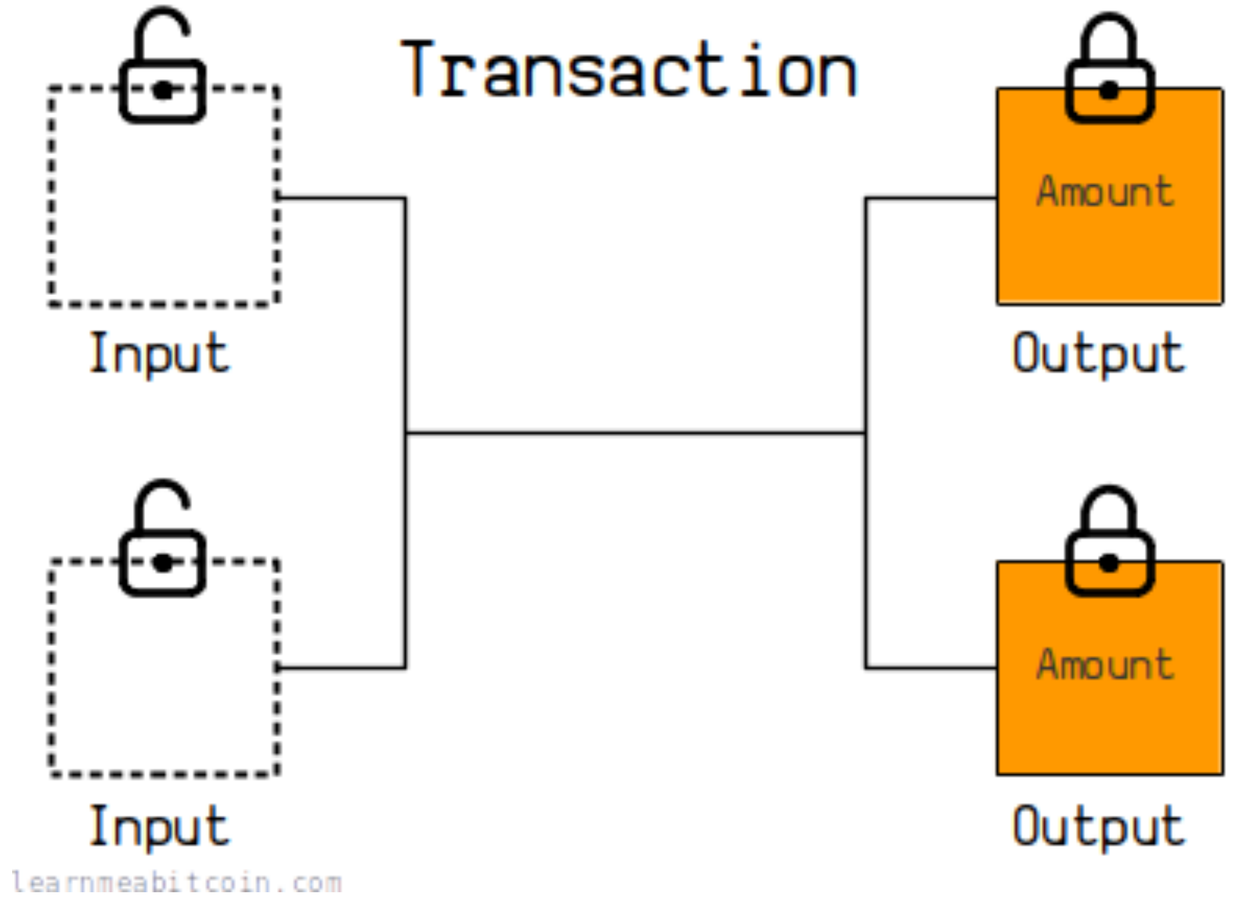
- 700GB 정도의 블록들 다운로드
- 모든 트랜잭션의 비트코인 액수 검증
- 모든 트랜잭션의 서명 검증

블록 다운로드



https://learnmeabitcoin.com/technical/block/

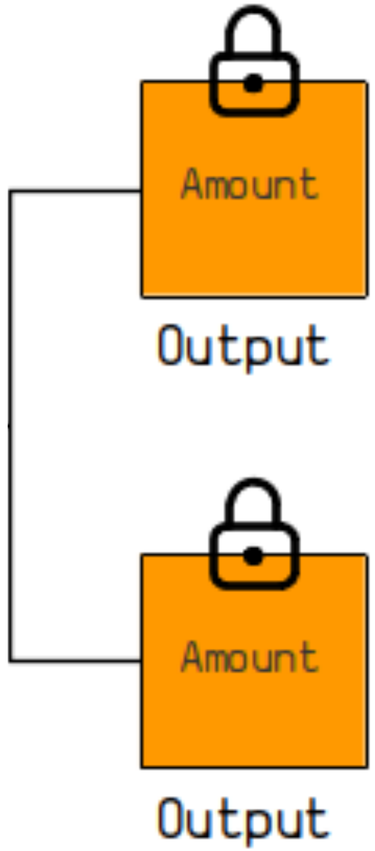
액수 검증



https://learnmeabitcoin.com/technical/transaction/

액수 검증

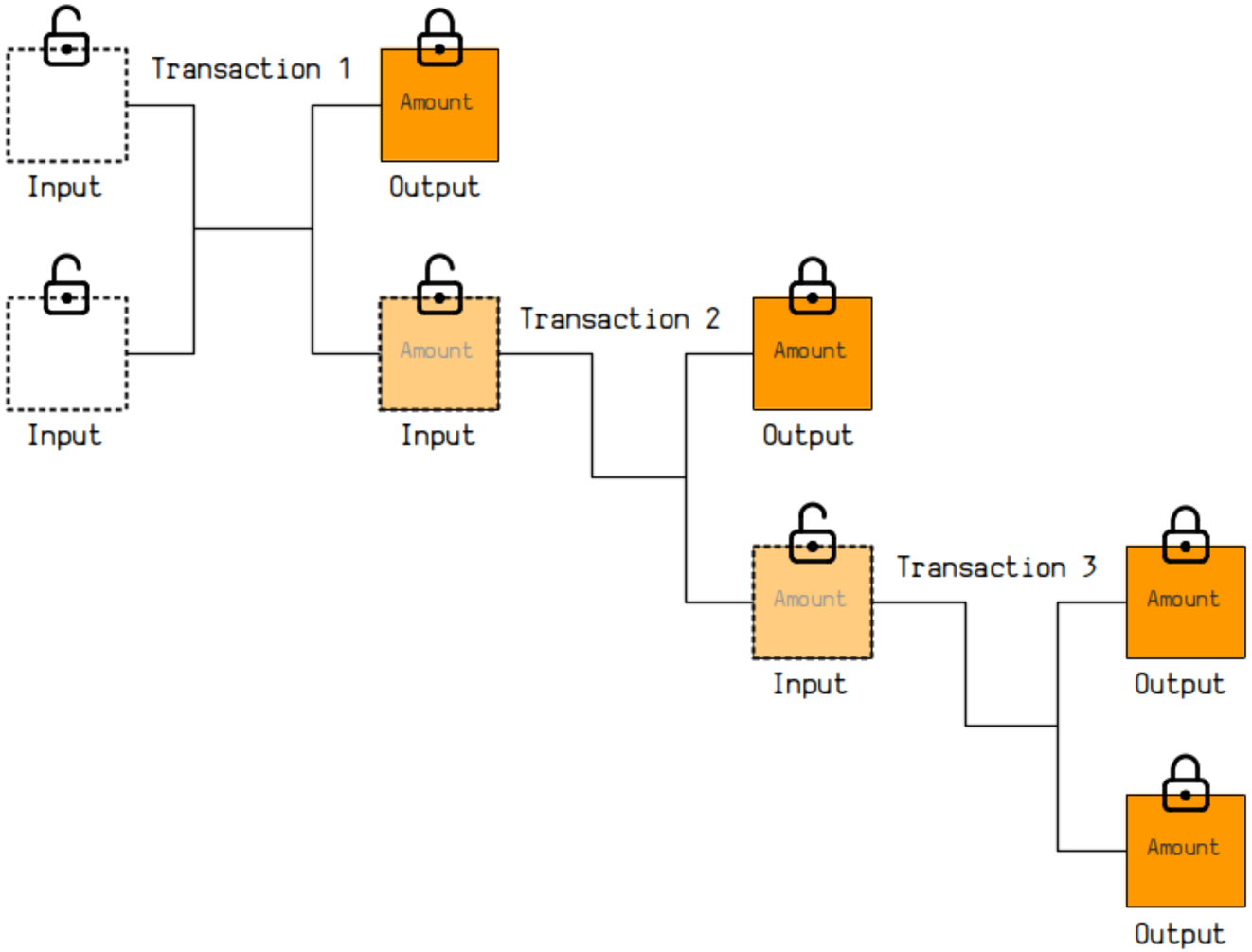
아웃풋 값들은 저장 (액수, 주소등)



https://learnmeabitcoin.com/technical/transaction/

액수 검증

추후 다른 인풋에서 사용될때 다시 읽어옴



액수 검증

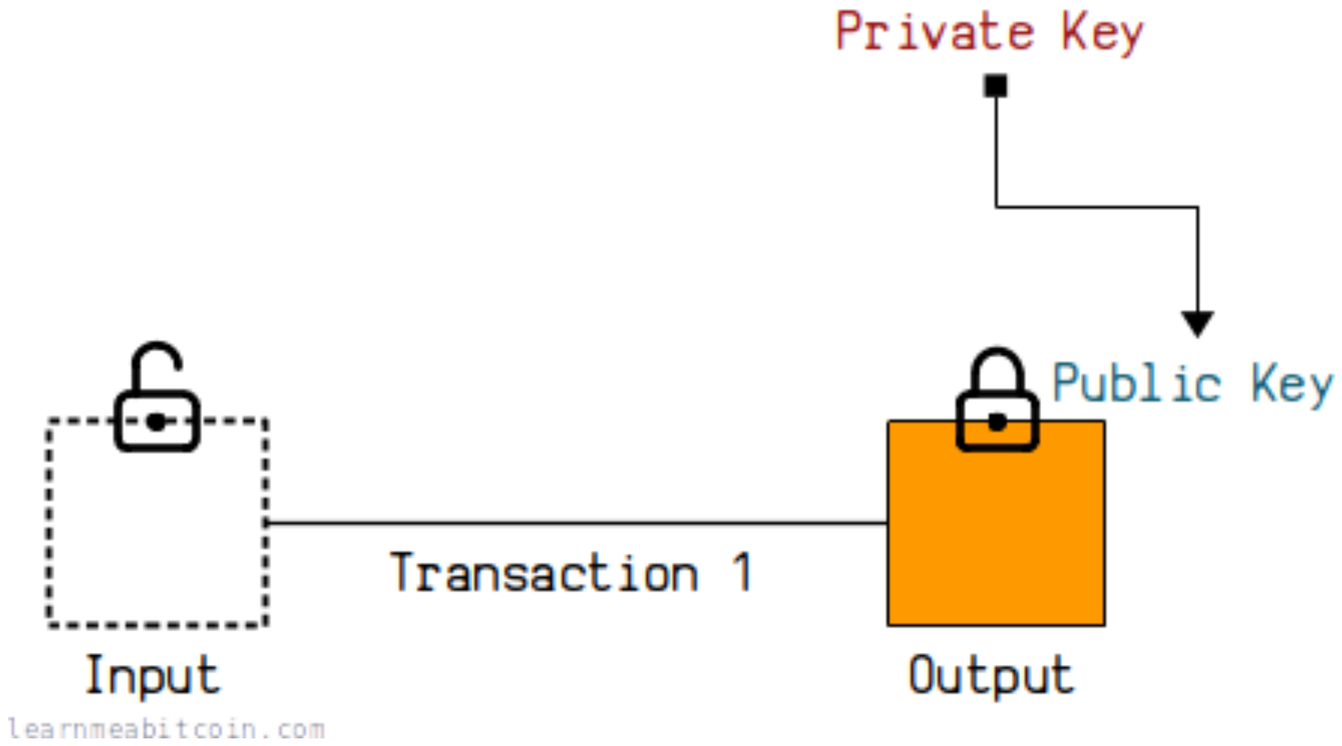
디스크 읽기/쓰기

- 트랜잭션의 인풋값을 읽기
- 트랜잭션의 아웃풋값을 쓰기

https://learnmeabitcoin.com/technical/transaction/

서명 검증

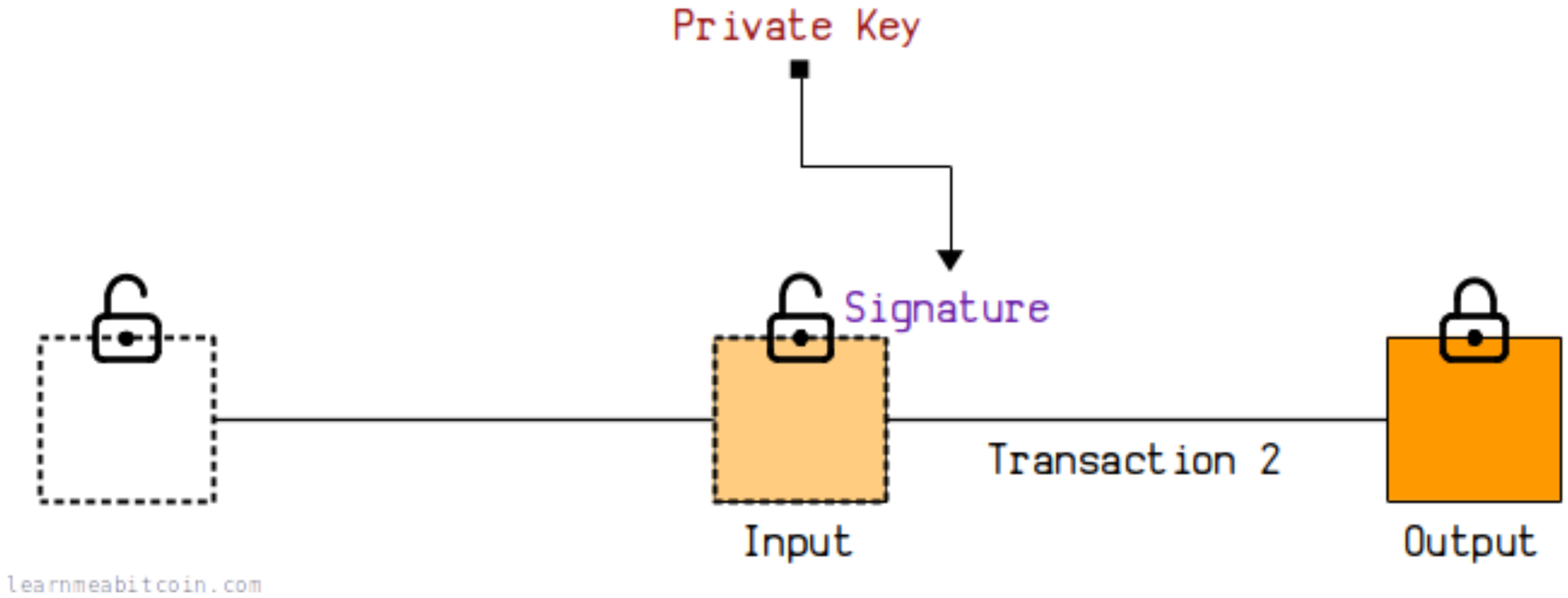
아웃풋에 내 주소를 넣음



https://learnmeabitcoin.com/technical/transaction/

서명 검증

이후 앤수 검증 후 서명 검증



https://learnmeabitcoin.com/technical/transaction/

서명 검증

CPU 사용

- 서명 검증 = 큰 숫자 많이 더하기

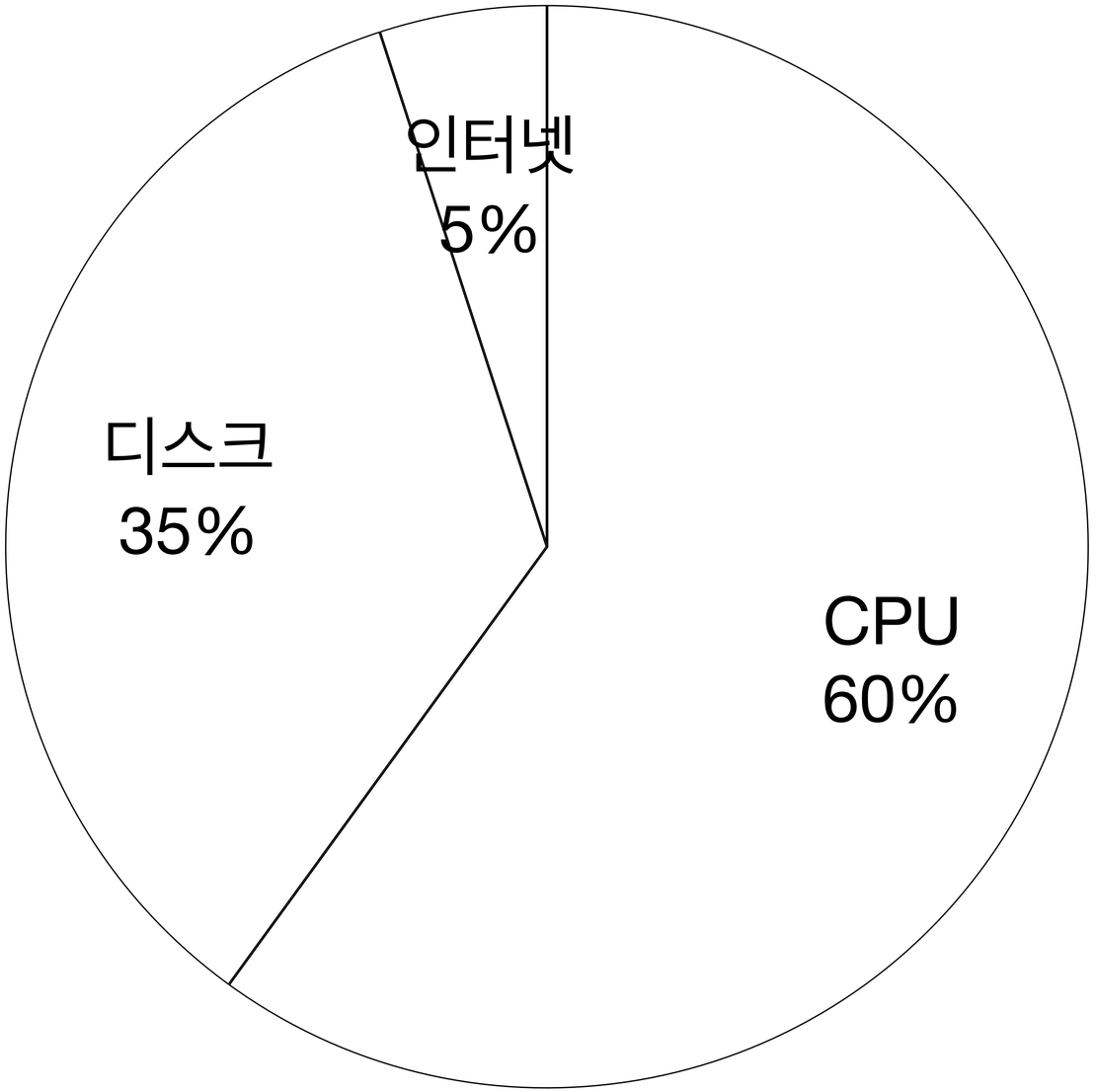
https://learnmeabitcoin.com/technical/transaction/

검증 과정

검증에서 시간이 제일 많이 소모되는 부분

- 인터넷 속도
- 디스크 성능
- CPU 성능

대략적 비율



CPU 속도 해결

2가지 방안들

- 더 빠른 코드 작성 https://github.com/bitcoin-core/secp256k1
- 트랜잭션 서명 검증 안하기

서명 검증 안하기

그 역사

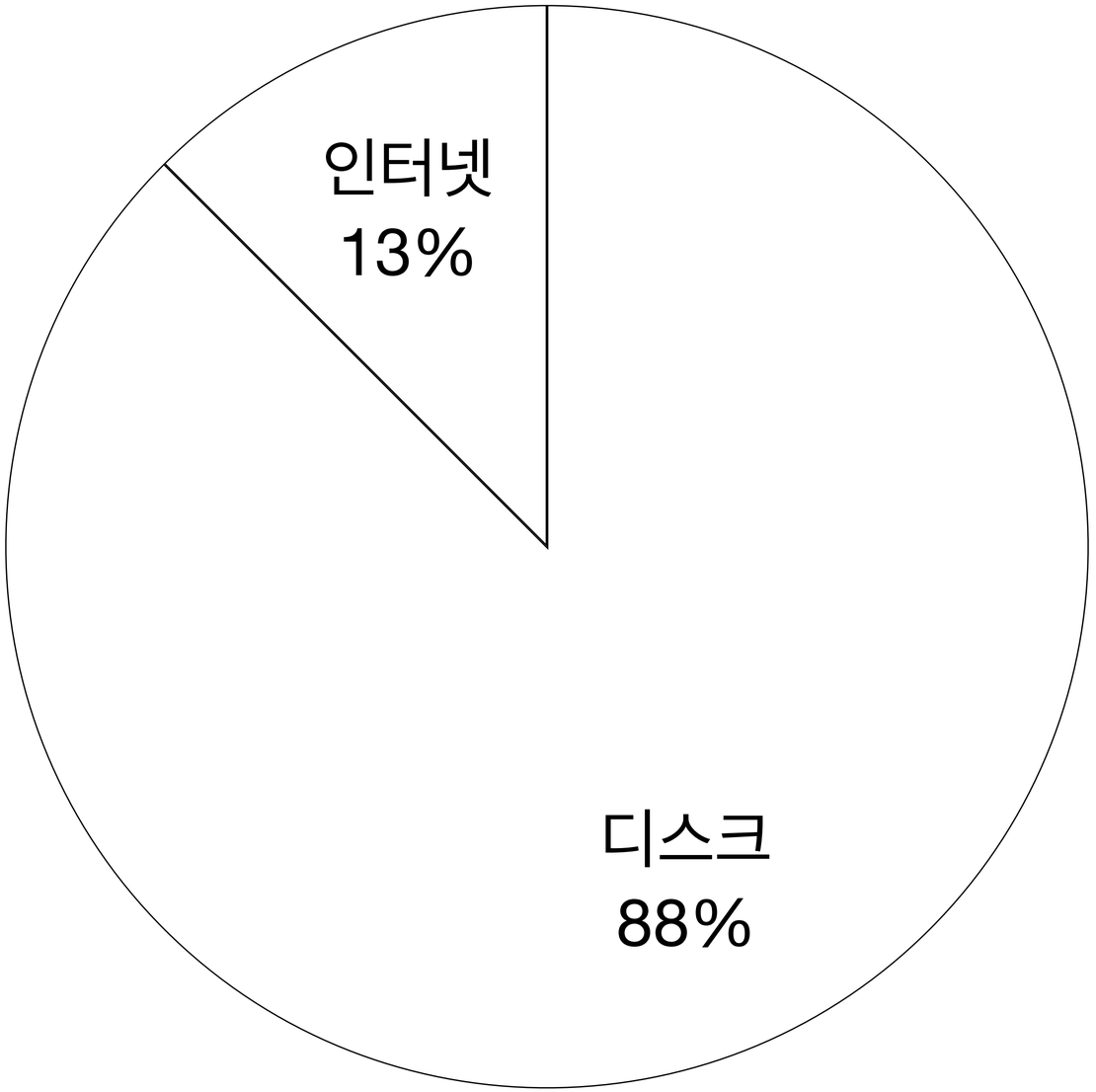
- 사토시가 처음 체크포인트를 넣음 https://bitcointalk.org/index.php?topic=437
- 이후 버전 0.14에서 assumevalid로 바뀜
- 디폴트 설정에서는 비트코인 노드가 서명 검증을 하지 않음

체크포인트란?

그 역사

- 특정 블록 높이에서 지정된 해시만 유효하도록 노드 소프트웨어에 하드코딩
- 본래 목적은 51% 공격을 막기 위함
- 이후 assumevalid라는 방식으로 바꿈. 이는 특정 블록 높이에서 지정된 해시 외 다른 해시도 유효

디폴트 설정에서의 대략적 비율



현재의 목표: 액수 검증을 빠르게 하기

2. 기존 IBD(Initial Block Download)의 한계

블록 검증 프로세스

현재의 검증

1. 블록을 다운로드 받는다
2. 각각의 트랜잭션마다
    A. 인풋 값 (액수, 주소) 디스크에서 읽어오기
    B. 인풋 액수 <= 아웃풋 액수 확인하기
    C. 아웃풋 값 (액수, 주소) 디스크에 쓰기

-dbcache

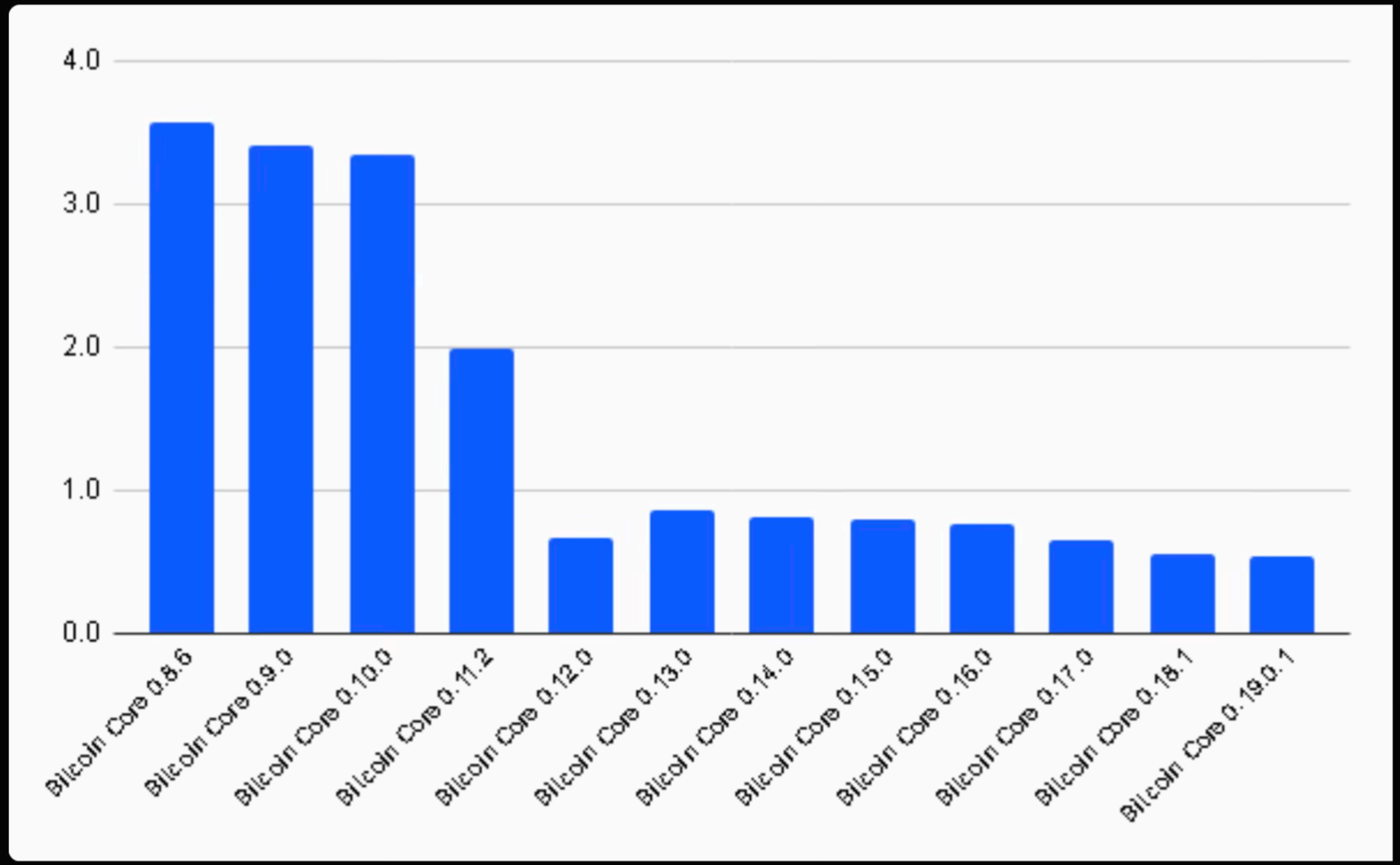
비트코인 코어 설정값

- 매번 디스크에서 읽어오는건 느리니까 기본적으로 컴퓨터 메모리에 캐싱을 해둠
- 이 캐시의 사이즈를 늘려주는 값
- 늘려주면 IBD가 많이 빨라짐

현재의 한계점들

- 트랜잭션을 검증하는 과정은 순차적이다
- 트랜잭션 하나의 검증 속도를 획기적으로 빨리 늘리기는 불가

Figure 1 – Bitcoin Initial Block Download Time (Days) – Average Of 3 Attempts



https://www.bitmex.com/blog/bitcoins-initial-block-download

2018-2020

노드 성능에 대한 여러가지 의논들

- 대부분은 성능 향상이 더 이상 불가능하다고 생각
- assumeUTXO 제안이 발표됨

OUT OF ORDER BLOCK VALIDATION WITH UTREEXO ACCUMULATORS

20 May 2021

Abstract: In this piece, BitMEX grantee Calvin Kim explains why the order in which blocks are validated does not matter under Utreexo. This is because the disk space savings make UTXO snapshots practical, which in turn allows for the parallelisation of the validation of the Bitcoin Blockchain. Calvin goes on to explain that one can check if an incoming transaction is valid by verifying the accumulator proof, getting rid of the need to access the UTXO set. This allows Utreexo nodes to eliminate disk access requirements, in exchange for more hashing, which can be an excellent trade-off. Calvin then discusses how Utreexo can change how we think about block validation.

https://www.bitmex.com/blog/faster-blockchain-validation-with-utreexo-accumulators

2021년 처음 블록을 병렬로 처리하는 기법을 공개

- 제가 처음으로 개발함
- 기존의 노드보다 63% 빨라짐
- 여러 컴퓨터를 활용하여 하나의 IBD를 더 빨리 처리하는 기법도 공개

단점

- Utreexo accumulator라는 새롭고 아직 비완성된 기술을 활용
- 기존의 IBD보다 추가로 다운로드 해야 하는 데이터가 있었음

당시엔 IBD 성능에 대한 관심이 없었음

Eric Voskuil
@evoskuil

X.com

Bitcoin Core completed to block 840k at 15.3 hours, so just over 15x longer than BN.

11:55 PM · 10/20/24 · **949** Views

3 1 11

Relevant

https://x.com/evoskuil/status/1848015101233672628

libbitcoin 충격

- libbitcoin이라는 다른 구현체가 비트코인 코어보다 15배 빨리 IBD를 끝냄
- libbitcoin도 병렬적으로 블록을 처리

블록 검증 프로세스

libbitcoin

프로세스 A

1. 블록을 다운로드 받는다
2. 각각의 트랜잭션마다
    A. 아웃풋이 사용되었는지 디스크에 쓰기
    B. 사용됐다면 사용한 트랜잭션 인풋을 디스크에 쓰기

프로세스 B

1. 프로세스 A가 처리한 블록의 트랜잭션마다
    A. 인풋에 해당하는 아웃풋이 디스크에 있는지 확인. 없으면 무효한 블록으로 처리

libbitcoin의 구현의 한계점

- 디스크 속도가 핵심. 디스크가 느린 컴퓨터에서는 느림

https://gist.github.com/RubenSomsen/a61a37d14182ccd78760e477c78133cd

SwiftSync

두 마리의 토끼를 다 잡음

- 디스크 성능이 느려도 상관없음
- 추가적으로 다운로드 받을 데이터 필요없음

3. SwiftSync의 설계 철학과 핵심 아이디어

SwiftSync

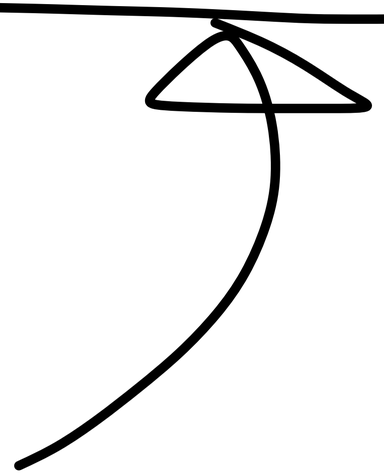
핵심 아이디어

- 동기화 하는 노드에게 미래에 사용될 아웃풋들을 알려줌
- 사용되지 않은 아웃풋들만 디스크에 저장

SwiftSync

핵심 아이디어

- 동기화 하는 노드에게 미래에 사용될 아웃풋들을 알려줌
- 사용되지 않은 아웃풋들만 디스크에 저장



이 정보가 거짓이 아닌것을 증명하는게
IBD

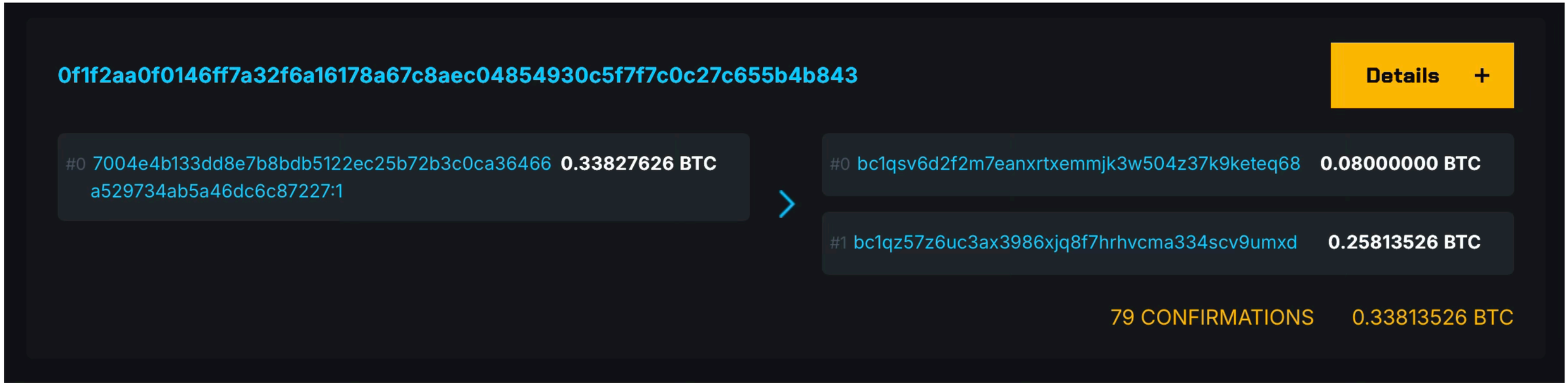
SwiftSync

핵심 아이디어

- 검증 방법은 (모든 아웃풋의 값) - (모든 인풋의 값)
- 아웃풋 리스트가 맞다면 위의 값은 0이여야함

SwiftSync

핵심 아이디어

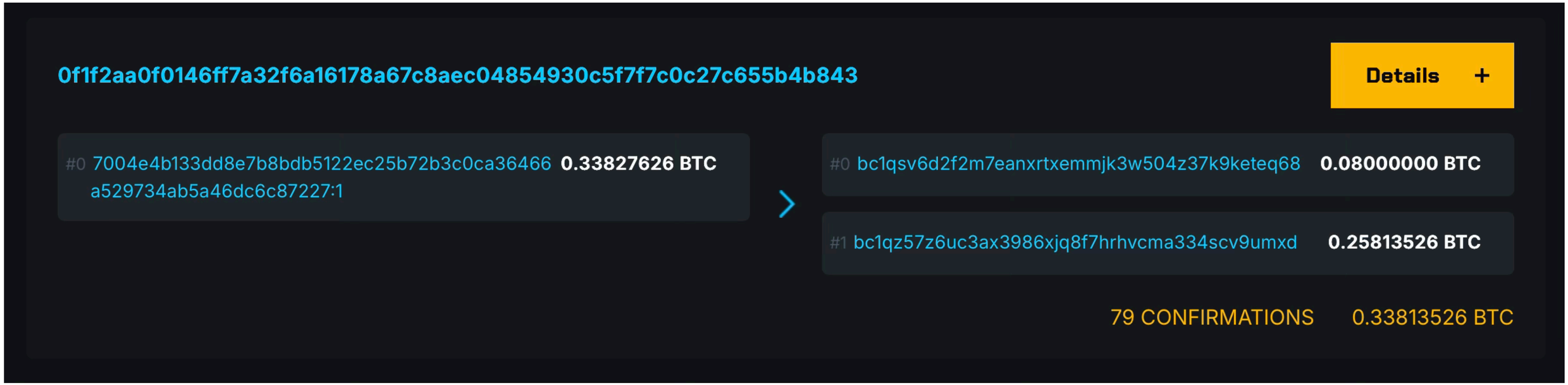


인풋:

7004e4b133dd8e7b8bdb5122ec25b72b3c0ca36466a529734ab5a46dc6c87227:1

SwiftSync

핵심 아이디어



아웃풋:

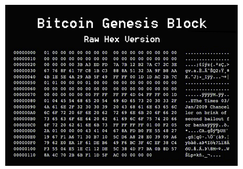
0f1f2aa0f0146ff7a32f6a16178a67c8aec04854930c5f7f7c0c27c655b4b843:0

0f1f2aa0f0146ff7a32f6a16178a67c8aec04854930c5f7f7c0c27c655b4b843:1

4. SwiftSync의 동작 방식과 기술적 트레이드오프

SwiftSync

현재 방식



TX (A)

아웃풋 1

아웃풋 2

TX (B)

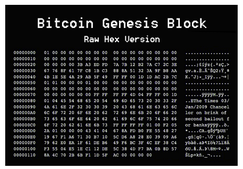
인풋 TX(A):1

TX (C)

인풋 TX(A):2

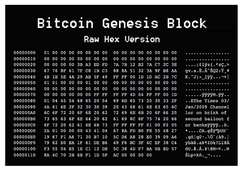
SwiftSync

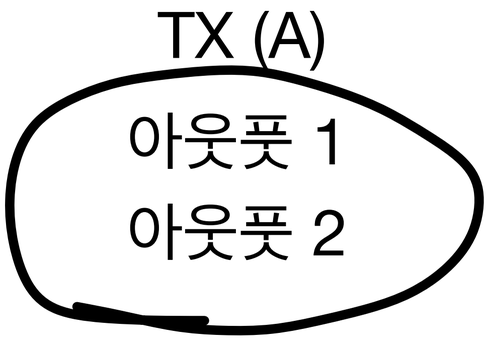
현재 방식



SwiftSync

현재 방식

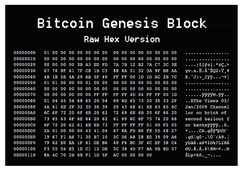




디스크에 저장

SwiftSync

현재 방식



TX (A)

아웃풋 1

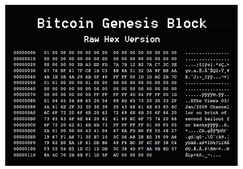
아웃풋 2

TX (B)

인풋 TX(A):1

SwiftSync

현재 방식



TX (A)

TX (B)

아웃풋 1

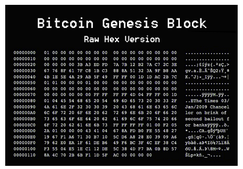
인풋 TX(A):1

아웃풋 2

디스크에서 읽어옴

SwiftSync

현재 방식



TX (A)

TX (B)

아웃풋 1

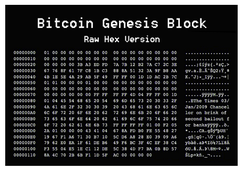
아웃풋 2

인풋 TX(A):1

디스크에서삭제

SwiftSync

현재 방식



TX (A)

아웃풋 1

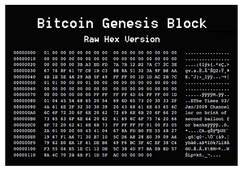
아웃풋 2

TX (B)

인풋 TX(A):1

SwiftSync

현재 방식



TX (A)

아웃풋 1

아웃풋 2

TX (B)

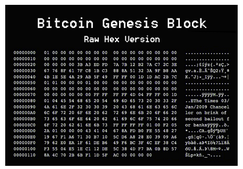
인풋 TX(A):1

TX (C)

인풋 TX(A):2

SwiftSync

현재 방식



TX (A)

아웃풋 1

아웃풋 2

TX (B)

인풋 TX(A):1

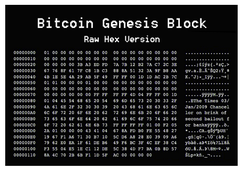
TX (C)

인풋 TX(A):2

디스크에서 읽어옴

SwiftSync

현재 방식



TX (A)

아웃풋 1

아웃풋 2

TX (B)

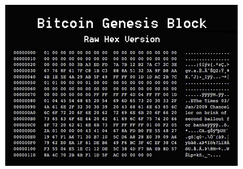
인풋 TX(A):1

TX (C)

인풋 TX(A):2

SwiftSync

현재 방식



TX (A)

아웃풋 1

아웃풋 2

TX (B)

인풋 TX(A):1

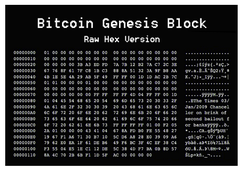
TX (C)

인풋 TX(A):2

디스크에서삭제

SwiftSync

Swiftsync 적용



TX (A)

아웃풋 1

아웃풋 2

TX (B)

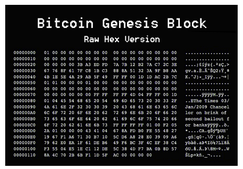
인풋 TX(A):1

TX (C)

인풋 TX(A):2

SwiftSync

Swiftsync 적용



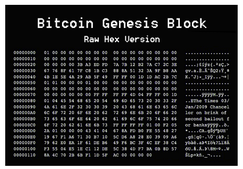
사용될 아웃풋:

TX(A):1

TX(A):2

SwiftSync

Swiftsync 적용



사용될 아웃풋:

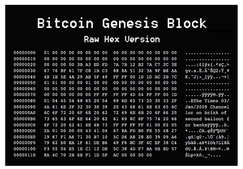
TX(A):1

TX(A):2

인풋&아웃풋 값: () - ()

SwiftSync

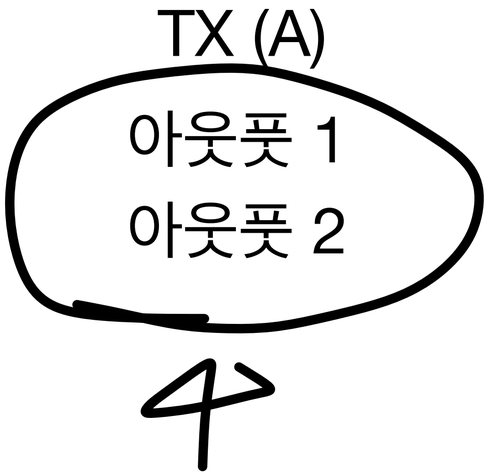
Swiftsync 적용



사용될 아웃풋:

TX(A):1

TX(A):2

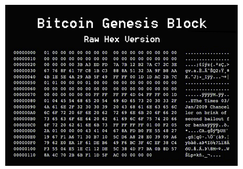


인풋&아웃풋 값: () - ()

디스크에 저장하지 않음

SwiftSync

Swiftsync 적용



→ →

사용될 아웃풋:

TX(A):1

TX(A):2

TX (A)

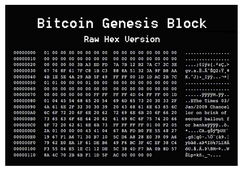
아웃풋 1

아웃풋 2

인풋&아웃풋 값: (TX(A):1 + TX(A):2) - ()

SwiftSync

Swiftsync 적용



→ → TX (A)

사용될 아웃풋:

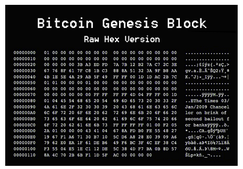
TX(A):1

TX(A):2

인풋&아웃풋 값: (TX(A):1 + TX(A):2) - ()

SwiftSync

Swiftsync 적용



→ → →

TX (A)

TX (B)

인풋 TX(A):1

사용될 아웃풋:

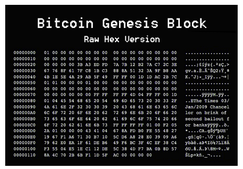
TX(A):1

TX(A):2

인풋&아웃풋 값: (TX(A):1 + TX(A):2) - ()

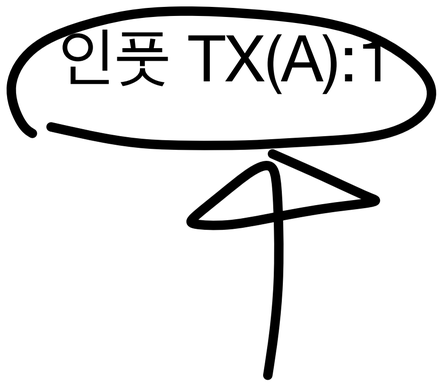
SwiftSync

Swiftsync 적용



TX (A)

TX (B)



사용될 아웃풋:

TX(A):1

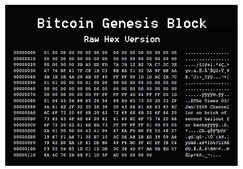
TX(A):2

인풋&아웃풋 값: (TX(A):1 + TX(A):2) - ()

인풋&아웃풋 값에 넣기

SwiftSync

Swiftsync 적용



→ → →

TX (A)

TX (B)

인풋 TX(A):1

사용될 아웃풋:

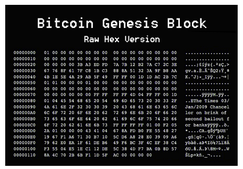
TX(A):1

TX(A):2

인풋&아웃풋 값: (TX(A):1 + TX(A):2) - (TX(A):1)

SwiftSync

Swiftsync 적용



TX (A)

TX (B)

사용될 아웃풋:

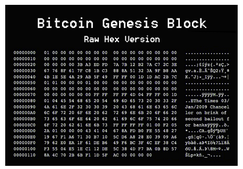
TX(A):1

TX(A):2

인풋&아웃풋 값: (TX(A):1 + TX(A):2) - (TX(A):1)

SwiftSync

Swiftsync 적용



TX (A)

TX (B)

TX (C)

인풋 TX(A):2

사용될 아웃풋:

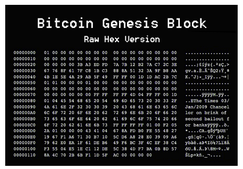
TX(A):1

TX(A):2

인풋&아웃풋 값: (TX(A):1 + TX(A):2) - (TX(A):1)

SwiftSync

Swiftsync 적용



TX (A)

TX (B)

TX (C)

사용될 아웃풋:

TX(A):1

TX(A):2

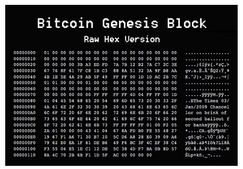
인풋 TX(A):2

인풋&아웃풋 값에 넣기

인풋&아웃풋 값: (TX(A):1 + TX(A):2) - (TX(A):1)

SwiftSync

Swiftsync 적용



TX (A)

TX (B)

TX (C)

인풋 TX(A):2

사용될 아웃풋:

TX(A):1

TX(A):2

인풋&아웃풋 값: (TX(A):1 + TX(A):2) - (TX(A):1 + TX(A):2)

아웃풋 - 인풋 계산

0 이여서 정보가 거짓이 아닌것을 확인

$$(TX(A):1 + TX(A):2) - (TX(A):1 + TX(A):2)$$

SwiftSync의 트레이드오프

- 사용된 아웃풋 리스트를 받아야함 (~100MB)
- 서명 검증을 위해선 새로운 프로토콜이 필요

5. 비트코인 네트워크 확장성과 검증의 미래

검증의 미래

네트워크 확장성

https://github.com/kcalvinalvin/2026-02-21-bitcoin-center-meetup