

너보스(Nervos Network)

해당 번역본은 이용자가 디지털자산 관련 정보를 편리하게 확인할 수 있도록 단순 참고용으로 번역된 것이며, 투자 권유를 목적으로 하지 않습니다. 그 내용이 정확하지 않을 수 있으므로 정확한 정보 습득을 위해서는 원문을 참고하시거나 원문 작성 주체 측에 문의하시기 바랍니다. 두나무는 해당 번역본 내용의 진실성 및 정확성을 보장하지 않고, 제공된 정보에 의한 투자 결과에 대하여 어떠한 책임도 지지 않으며, 투자 손실은 투자자에게 귀속됩니다

CKB 작동 원리

블록체인 초보자든 기술적 배경이 있는 사람이든, 이 가이드는 CKB가 어떻게 작동하는지 이해하는 데 도움이 될 것이다. 우리는 Cell 모델, 스크립트(Script), 트랜잭션(Transaction), 그리고 CKB-VM의 기본 구성 요소를 쉽게 이해할 수 있는 방식으로 살펴볼 것이다.

CKB 소개

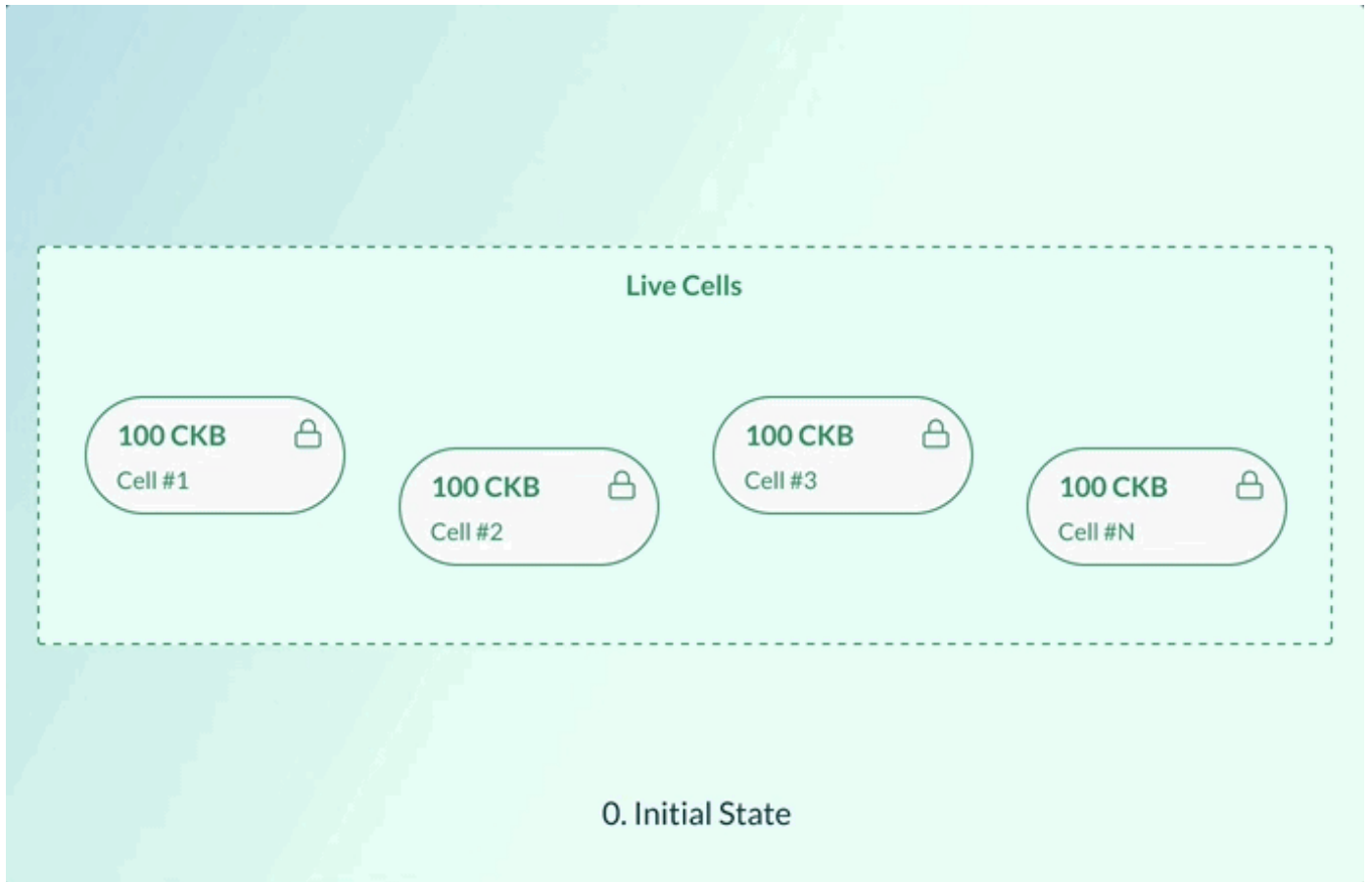
블록체인은 네트워크 전반에 걸쳐 트랜잭션을 기록하는 탈중앙화되고 변경 불가능한 원장으로, 중개자 없이 상호작용을 가능하게 한다. CKB(Common Knowledge Base)는 Nervos 네트워크의 기초 계층으로, 작업 증명(Proof of Work, PoW) 합의에 기반해 운영된다. 이때 채굴자들은 암호 퍼즐을 해결함으로써 네트워크를 보호하고 트랜잭션을 검증하여 강력한 보안성과 탈중앙화를 보장한다.

트랜잭션의 작동 방식

CKB 블록체인은 'Cell'이라 불리는 상자로 가득 찬 저장소처럼 작동한다. 이러한 Cell은 CKB의 네이티브 토큰인 CKByte를 보유할 수 있으며, 이는 가치와 저장 용량을 동시에 나타낸다. 즉, 더 많은 CKByte를 소유할수록 블록체인에 더 많은 데이터를 저장할 수 있다.

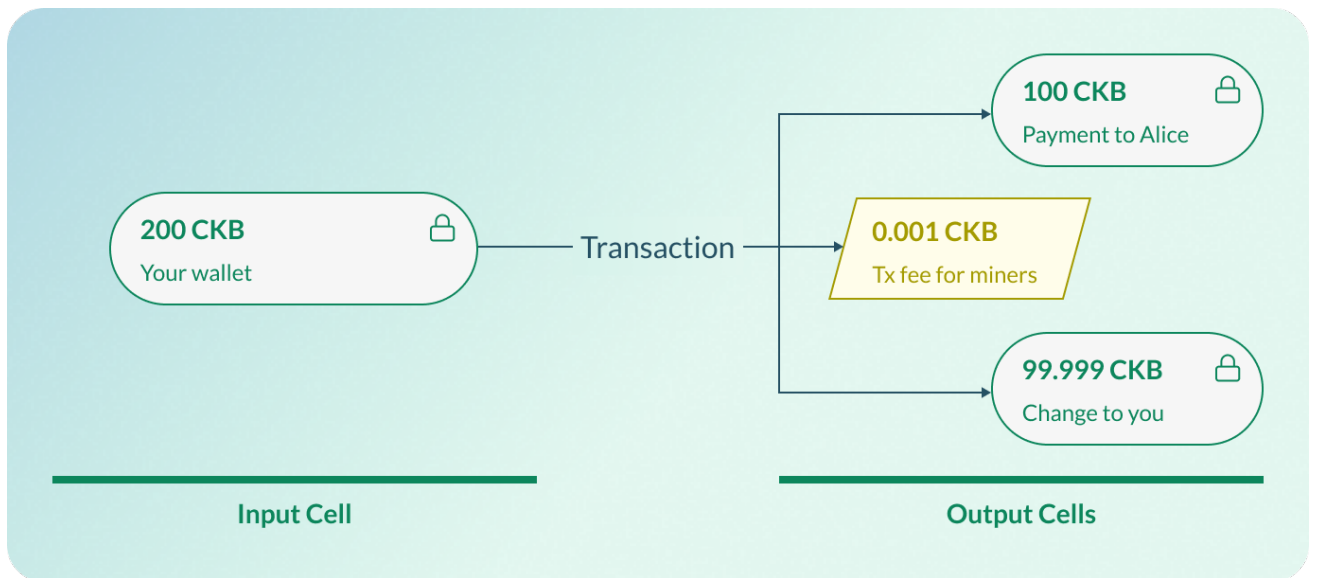
각 Cell은 기능에 필요한 기본 데이터를 담기 위해 최소 61 CKBytes가 필요하다. 실제로는 트랜잭션 수수료와 원활한 처리를 위해 **62 CKBytes** 이상을 할당하는 것이 권장된다.

Cell 모델은 이러한 Cell의 동작 방식을 정의하며, 스크립트(Script)에 의해 Cell이 사용되거나 접근될 수 있는 규칙이 정해진다. 일단 블록체인에 추가된 Cell은 불변적이므로 수정할 수 없다. Cell의 데이터를 업데이트하려면 해당 Cell을 소비(consume)해야 한다 — 데이터를 추출하고 수정한 후, 업데이트된 정보를 담은 새로운 Cell을 생성해야 한다.

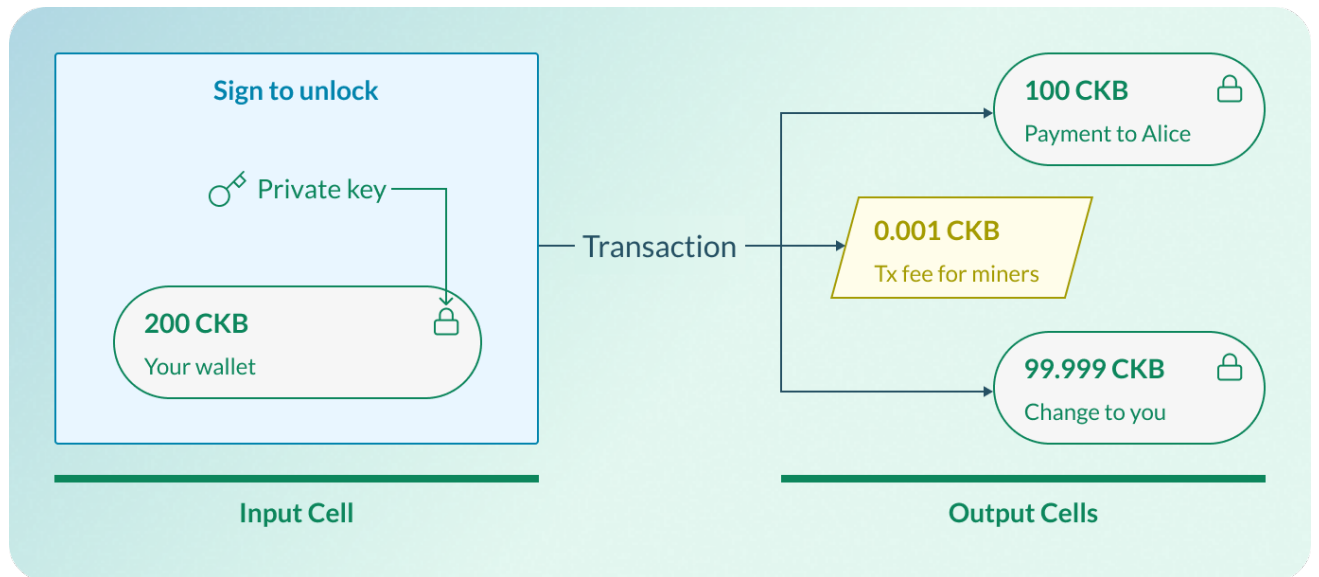


아직 소비되지 않은 Cell을 Live Cell이라 부르며, 이는 향후 트랜잭션에서 사용할 수 있다. 예를 들어, Live Cell에 200 CKBytes가 들어 있고, 이 중 100 CKBytes를 Alice에게 전송하고 싶다고 하자.

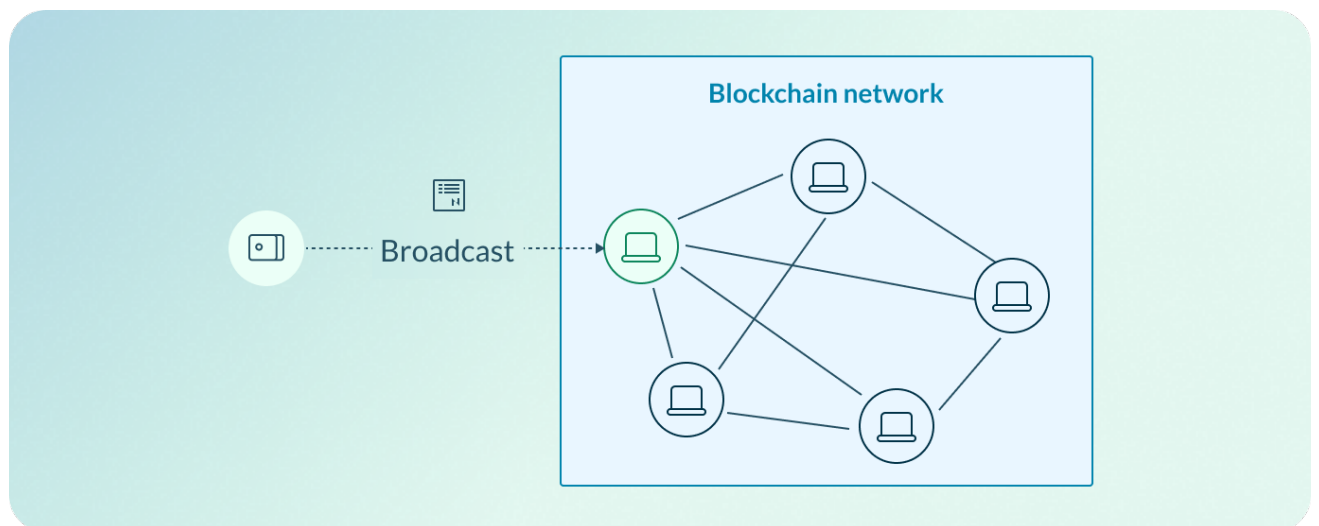
1. **생성(Create):** 우선 200 CKBytes 용량의 Live Cell을 선택한다. 이후 출력으로 두 개의 새로운 Live Cell이 생성된다. 하나는 100 CKBytes로, 오직 Alice만 잠금 해제할 수 있도록 설정되고, 다른 하나는 99.999 CKBytes로, 자신이 소유한 잔돈 Cell이다. 0.001 CKBytes 차이는 트랜잭션 수수료로 사용된다.



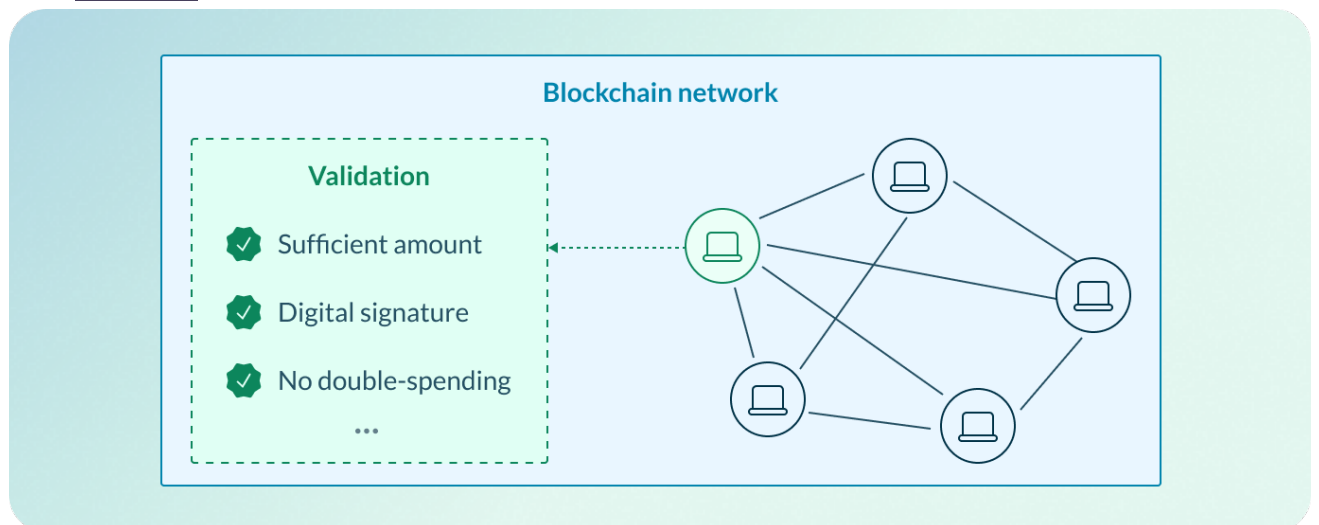
2. **서명(Sign):** 개인 키로 트랜잭션에 서명하여, 해당 Cell 내 CKBytes를 사용할 권한이 있음을 증명한다.



3. **전파(Broadcast)**: 서명 후, 지갑에서 가장 가까운 CKB 네트워크 노드로 트랜잭션을 브로드캐스트한다.

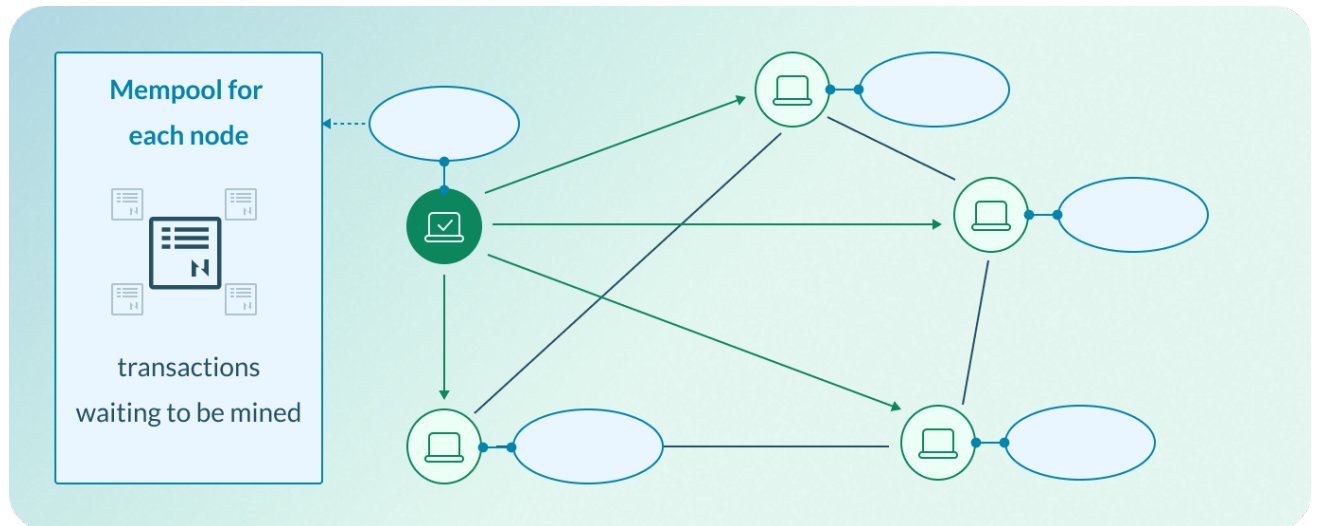


4. **검증(Validate)**: 수신 노드는 여러 검증 절차를 수행하여 트랜잭션의 유효성을 확인한다. 자세한 내용은 검증 과정을 참고할 수 있다.

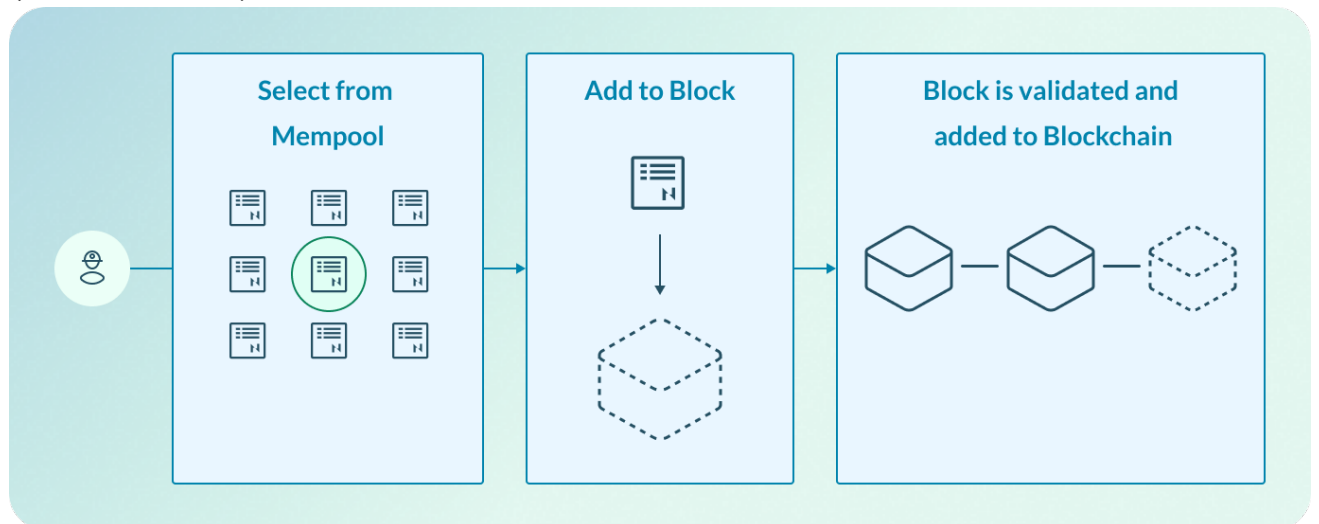


5. **전파(Propagate)**: 검증된 트랜잭션은 네트워크의 다른 노드로 전파되어 mempool에 저장된다. mempool은 채굴자들이 트랜잭션을 선택하기 전까지 미확인 트랜잭션을 임시로 보관하는 공간이

다.



6. **확정(Confirm):** 채굴자들은 mempool에서 트랜잭션을 선택해 새로운 블록에 포함시킨다. 트랜잭션이 포함된 블록이 채굴되면 블록체인에 추가되고, 이 시점에서 트랜잭션은 1회의 확인(confirmation)을 갖게 된다.



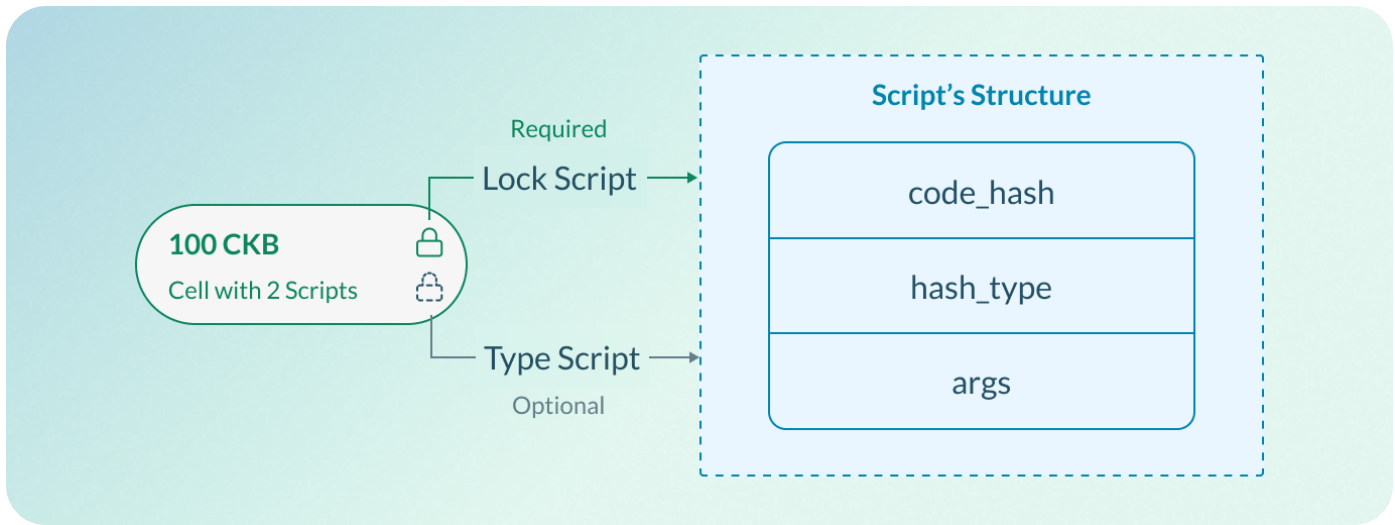
그 결과:

- 기존의 200 CKBytes Live Cell은 소비되어 Dead Cell로 표시된다.
- 새로 생성된 두 개의 Live Cell(100 CKBytes의 Alice Cell, 99.999 CKBytes의 사용자 Cell)은 블록체인 상에서 활성 상태가 된다.
- 이후 새 블록이 채굴될 때마다 트랜잭션은 추가적인 확인을 받아 블록체인 내에서의 영속성과 보안성이 강화된다.

스크립트(Scripts)

스크립트란?

CKB에서 스크립트(Script)는 온체인에서 실행될 수 있는 바이너리 실행 파일이다. 이는 CKB-VM이라 불리는 RISC-V 명령어 집합 기반 가상 머신에서 실행되는 프로그램으로, Cell을 보호하고 제어하기 위한 임의의 논리를 수행할 수 있다. 스마트 계약(Smart Contract)과 유사하게 생각할 수 있다.



- `code_hash` : CKB-VM에 로드할 스크립트 코드를 식별한다.
- `hash_type` : CKB-VM이 스크립트 코드나 스크립트를 찾는 방식을 지정한다.
- `args` : 사용자 공개키 해시 등, 스크립트를 구별하기 위한 인수를 제공한다.

스크립트에는 두 가지 주요 유형이 있다:

- **Lock Script**: Cell의 소유권 및 접근을 제어하는 필수 스크립트로, 권한이 있는 사용자만 Cell을 소비할 수 있도록 한다.
- **Type Script**: 트랜잭션에서 Cell이 어떻게 사용되거나 수정될 수 있는지를 정의하는 선택적 스크립트이다.

CKB 시스템 스크립트

가장 중요한 시스템 스크립트 중 하나는 `secp256k1_blake160_sighash_all`로 알려진 기본 Lock Script이다. 이 스크립트는 `secp256k1` 타원곡선과 Blake2b 해시 알고리즘을 사용하여, 해당 개인 키 소유자만이 Cell에 연결된 CKBytes를 잠금 해제하고 소비할 수 있도록 보장한다.

예를 들어, Alice가 `secp256k1_blake160_sighash_all` Lock Script로 보호된 Cell에서 CKBytes를 전송하려는 경우 다음 과정이 수행된다:

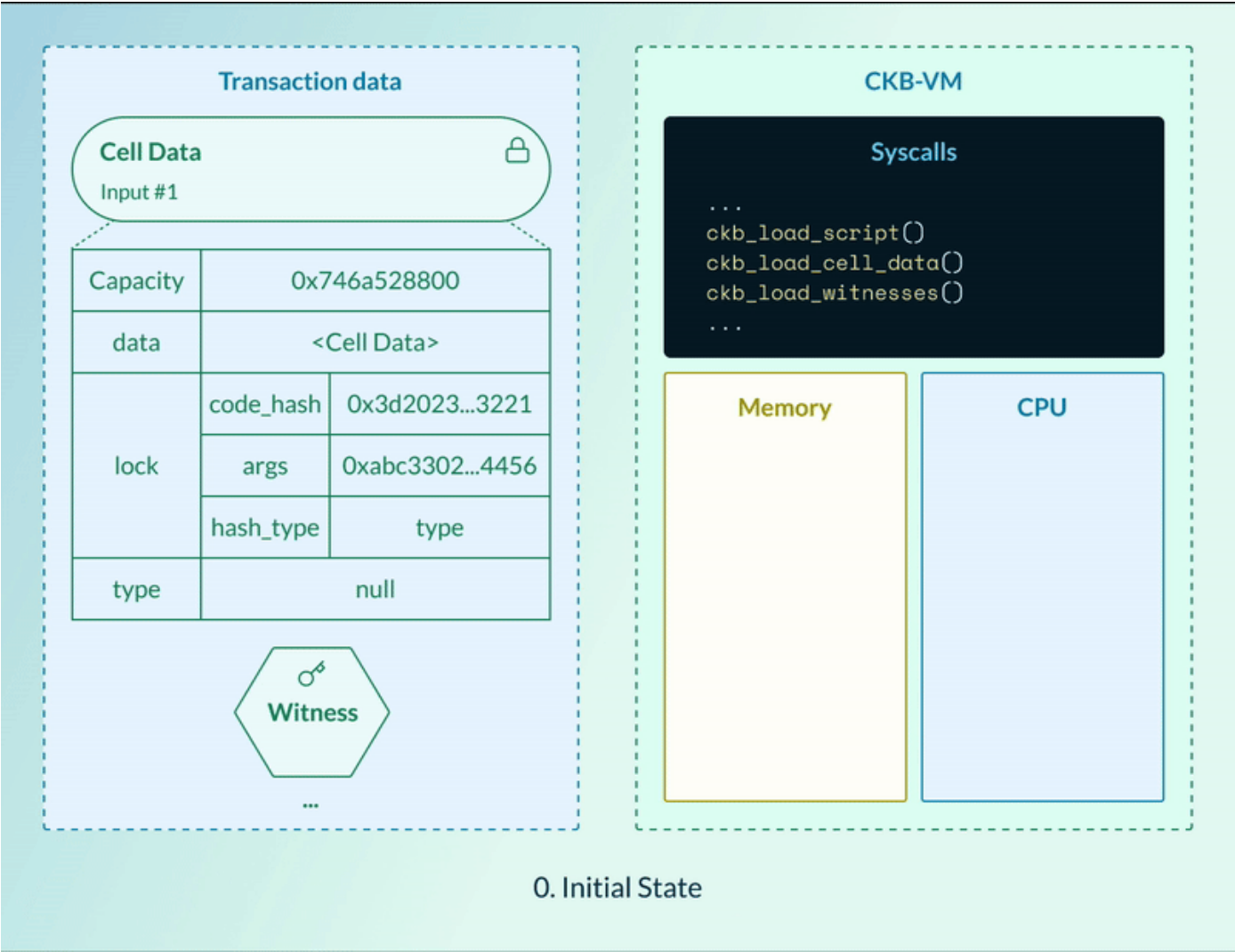
1. 트랜잭션은 Alice의 개인 키로 서명된다. 이 서명은 Alice가 Cell의 CKBytes를 사용할 권한이 있음을 증명한다.
2. 트랜잭션의 서명에서 나온 공개키를 Blake2b로 해시하여 새로운 Blake160 해시를 생성한다. Lock Script는 이 해시를 Cell의 `script_args`에 저장된 해시와 비교한다. 두 해시가 일치하면, 스크립트는 `secp256k1` 곡선을 사용해 서명을 검증하여 서명이 실제 해당 공개키의 개인 키로 생성되었음을 확인한다.

CKB-VM

CKB-VM은 CKB 상에서 스크립트를 실행하는 가상 머신이다. 현대적이고 오픈소스인 RISC-V 명령어 집합을 사용하며, CPU에 대한 저수준 접근을 제공해 높은 효율성과 유연성을 확보한다.

스크립트 실행

스크립트 실행은 트랜잭션이 CKB 네트워크에 제출될 때 시작된다. CKB-VM은 먼저 `code_hash`와 `hash_type`으로 참조되는 스크립트 코드를 `syscalls`를 통해 메모리에 로드한다. 다음으로 `ckb_load_cell_data()`와 `ckb_load_witnesses()` 등의 `syscall`을 사용하여 트랜잭션 내의 Cell 및 Witness에서 필요한 데이터를 불러온다. 필요한 모든 요소가 준비되면, CKB-VM은 스크립트를 실행하여 공개키 해시, 서명 검증, 저장된 값과의 비교 등을 수행함으로써 트랜잭션을 검증한다. 실행 후 스크립트의 반환값이 0이면 성공적인 실행을 의미하며, 이는 유효한 트랜잭션이다. 반대로 0이 아닌 값을 반환하면 실행 실패로 간주되어 트랜잭션은 무효 처리된다.



사이클(Cycles)

CKB-VM은 루프나 분기문 등 다양한 제어 흐름 구조를 지원하도록 설계되었다. 사이클(cycle)은 스크립트 실행의 계산 비용을 측정하는 단위로, 각 명령어나 syscall은 일정한 수의 사이클을 소비한다. 그러나 이러한 유연성은 무한 루프와 같은 악의적 행위를 방지하기 위한 안전장치가 필요함을 의미한다. 이를 위해 합의 계층에서 `max_block_cycles`라는 필드를 통해 사이클 한도를 설정한다. 이 필드는 하나의 블록 내에서 모든 스크립트가 사용할 수 있는 총 사이클 수의 상한을 정하며, 블록의 사이클 합이 이를 초과하면 블록은 거부된다.

현재 CKB 메인넷 MIRANA의 `max_block_cycles`는 3,500,000,000이다.

중요하게도, 개별 트랜잭션이나 스크립트에는 별도의 사이클 제한이 없다. 단지 블록 전체가 `max_block_cycles` 범위 내에 있는 한, 필요한 만큼의 사이클을 사용할 수 있다.

각 명령어나 syscall의 사이클 측정 방식에 대한 자세한 내용은 [사이클 한도를 통한 스크립트 규제](#)를 참고하라.

다음 단계

이론적 지식을 습득했다면, 이제 실전으로 나아갈 준비가 되었다.

- [Quick Start](#)를 통해 첫 번째 CKB 프로젝트를 설정하고 실행해보자.
- 또는 [dApp 튜토리얼](#)로 바로 이동해 CKB 위에서 실제로 빌드하는 실무 지식을 익힐 수 있다.

CKB 기본 개념

Nervos 블록체인

Nervos 블록체인이란?

Nervos 블록체인은 Common Knowledge Base라고도 하며, Nervos 생태계의 가장 근본적인 계층이다. 기반 계층으로서 Nervos 블록체인은 상위 계층 전체에 신뢰를 제공한다. 이는 탈중앙화를 극대화하면서도 최소한의 구조로 유지되고, 유연하고 안전하게 설계되었다. 주요 목적은 저장된 모든 데이터와 자산을 신뢰성 있게 보존하는 것이다.

다층 구조(Multi-Layer Architecture)

탈중앙화 시스템을 설계할 때는 항상 트레이드오프가 존재한다. 대표적인 예는 확장성, 보안성, 탈중앙화 간의 균형이다. 단일 계층에서 이 세 가지를 모두 높은 수준으로 달성하기는 어렵지만, 계층을 나누면 각기 다른 문제를 독립적으로 해결할 수 있다.

계층화를 통해 향상되는 예시는 다음과 같다:

- 1계층(Layer 1)은 보안성과 탈중앙화에 초점을 맞추어 상위 계층에 신뢰를 제공한다.
- 2계층(Layer 2)은 확장성에 초점을 맞추어 수백만 명의 사용자에게 거의 즉각적인 트랜잭션을 제공한다.

이 두 계층은 함께 작동하여 탈중앙화, 보안성, 확장성의 높은 수준을 달성한다.

Nervos 블록체인은 보안성과 탈중앙화를 핵심 설계 원칙으로 하는 다층 구조의 1계층을 나타낸다.

CKByte란?

Nervos의 네이티브 토큰은 CKByte 또는 간단히 CKB라고 한다. 하나의 CKByte 토큰은 Nervos 상에서 1바이트의 데이터 저장 공간을 소유할 권리를 부여한다. CKByte는 트랜잭션 및 연산에 필요한 수수료를 지불할 때도 사용된다.

Nervos에 100바이트의 데이터를 저장하려면 반드시 100 CKBytes를 보유해야 한다. 데이터가 Nervos 상의 공간을 차지하는 동안, 해당 CKBytes는 잠긴 상태로 유지된다. 데이터가 더 이상 필요 없어 삭제되면, 100 CKBytes는 다시 다른 용도로 사용할 수 있게 된다.

또한 CKBytes는 Nervos DAO에 예치할 수 있으며, 스테이킹과 유사한 방식으로 보상을 얻을 수 있다.

CKByte에 대한 추가 정보는 [Cell 모델](#) 및 [경제 구조](#) 섹션에서 다룬다.

Nervos에서의 프로그래밍

Nervos는 Javascript, Rust, C 등 점점 더 많은 범용 프로그래밍 언어를 지원하여 스마트 계약 프로그래밍 환경을 제공한다.

Nervos 상의 모든 프로그램은 온체인에서 데이터와 상태를 저장할 수 있으므로 복잡한 애플리케이션이나 맞춤형 토큰을 단순하고 직관적으로 만들 수 있다.

모든 코드는 CKB-VM에서 실행된다. CKB-VM은 고성능 RISC-V 가상 머신으로, 개발자에게 안전하고 일관되며 유연한 환경을 제공한다. 여러 CKB-VM 인스턴스가 서로 다른 스마트 계약을 동시에 실행할 수 있어, 대규모 병렬화를 통해 확장성이 크게 향상된다.

Nervos에서의 프로그래밍에 대한 자세한 내용은 [Cell 모델](#)과 [CKB-VM](#) 섹션에서 다룬다.

합의(Consensus)

Nervos는 NC-MAX라 불리는 작업 증명(PoW) 기반 합의 알고리즘을 사용한다. PoW는 보안 인센티브를 제공하는 최적의 솔루션으로 여러 차례 검증되었다.

Nervos는 비트코인의 나카모토 합의(Nakamoto Consensus)를 기반으로 하며, 보안성이나 탈중앙화를 훼손하지 않으면서 초당 트랜잭션 수를 비약적으로 늘리고 확정 시간을 단축시킨다.

현재 Nervos는 Ethereum 대비 약 10배의 처리량을 제공하며, 향후 Layer 2 솔루션이 본격화됨에 따라 기하급수적으로 확장될 것으로 예상된다.

Nervos 합의 알고리즘의 설계에 대한 자세한 내용은 [합의\(Consensus\)](#) 섹션에서 다룰 예정이다.

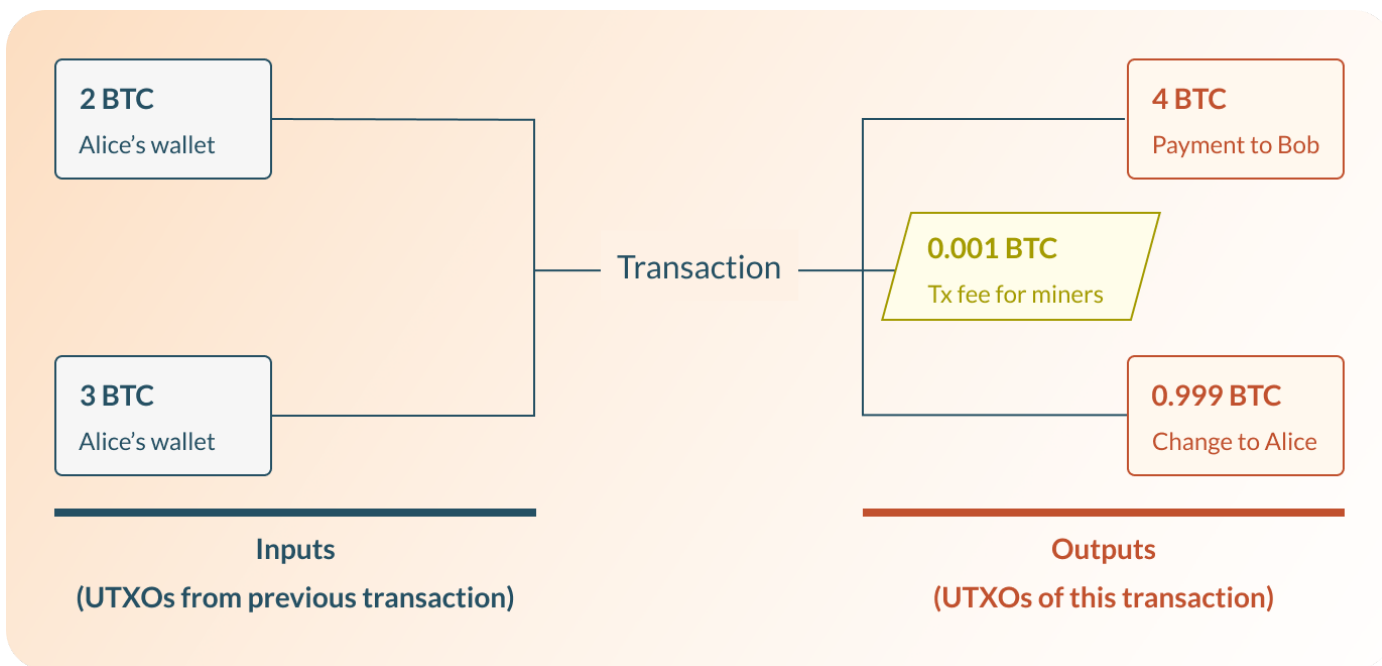
CKB vs. BTC

CKB는 블록체인의 개척자인 비트코인에서 영감을 받아 UTXO와 작업증명(Proof of Work) 같은 비트코인의 근본적 혁신을 계승하면서, 특히 비트코인을 위한 계약 유연성과 레이어 2 솔루션을 강화하는 데 고유하게 초점을 맞춘다. 이 가이드는 Nervos CKB 여정을 시작하는 데 필요한 핵심 개념과 비교를 안내하며, [기본 단위](#), [가상 머신\(VM\)](#), [스크립트](#), [트랜잭션 구조](#), 그리고 [검증 과정](#)에 집중한다.

UTXO vs Cell

BTC의 UTXO

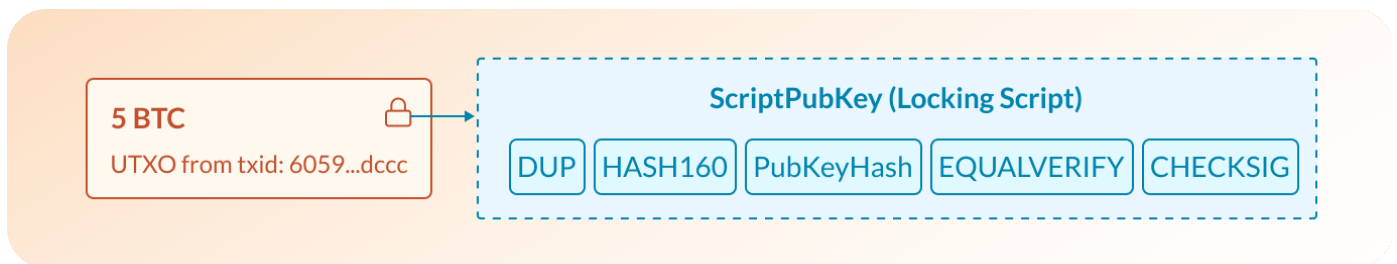
UTXO(Unspent Transaction Output)는 트랜잭션이 완료된 후 남은 디지털 화폐의 양을 의미한다. 예를 들어, Alice에게 5 BTC가 있고 Bob에게 4 BTC를 보낸다면, 두 개의 새로운 UTXO가 생성된다: Bob을 위한 4 BTC와 Alice를 위한 0.999 BTC. UTXO들의 총액이 입력 금액보다 적은 것에 유의해야 하는데, 그 차이는 채굴자에게 트랜잭션 수수료로 지불되기 때문이다.



UTXO에는 `scriptPubKey`로 알려진 단일 락이 포함되어 있으며, 이는 UTXO를 소비하기 위해 충족되어야 하는 조건을 설정한다.

UTXO의 데이터 구조는 다음과 같다:

```
{
  "value": 5, // The amount of Bitcoin
  "scriptPubKey": "OP_DUP OP_HASH160 <PubKeyHash> OP_EQUALVERIFY
OP_CHECKSIG" // The locking Script that sets the conditions for spending
the UTXO
}
```



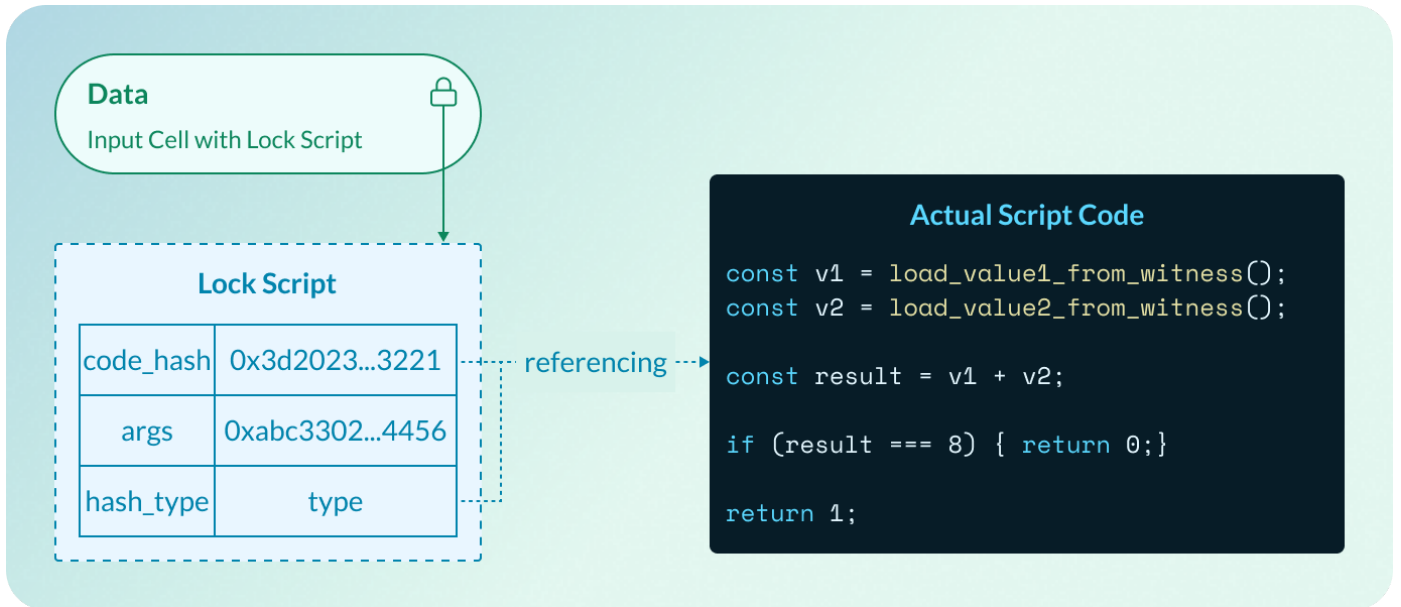
CKB의 Cell

UTXO와 달리, Cell은 더 높은 유연성을 제공한다. Cell은 CKB의 기본 단위이며 암호화페뿐 아니라 이미지, 비디오, 코드 같은 다른 유형의 데이터도 저장할 수 있다. Cell에는 Lock Script와 Type Script가 모두 포함된다:

- **Lock Script:** 비트코인의 `scriptPubKey`처럼 작동하며, Cell을 잠금 해제하고 소비할 수 있는 조건을 정의한다.
- **Type Script:** 선택적 스크립트로, Cell 변환 또는 상태 전이에 대한 조건을 정의한다.

Cell의 데이터 구조는 다음과 같다:

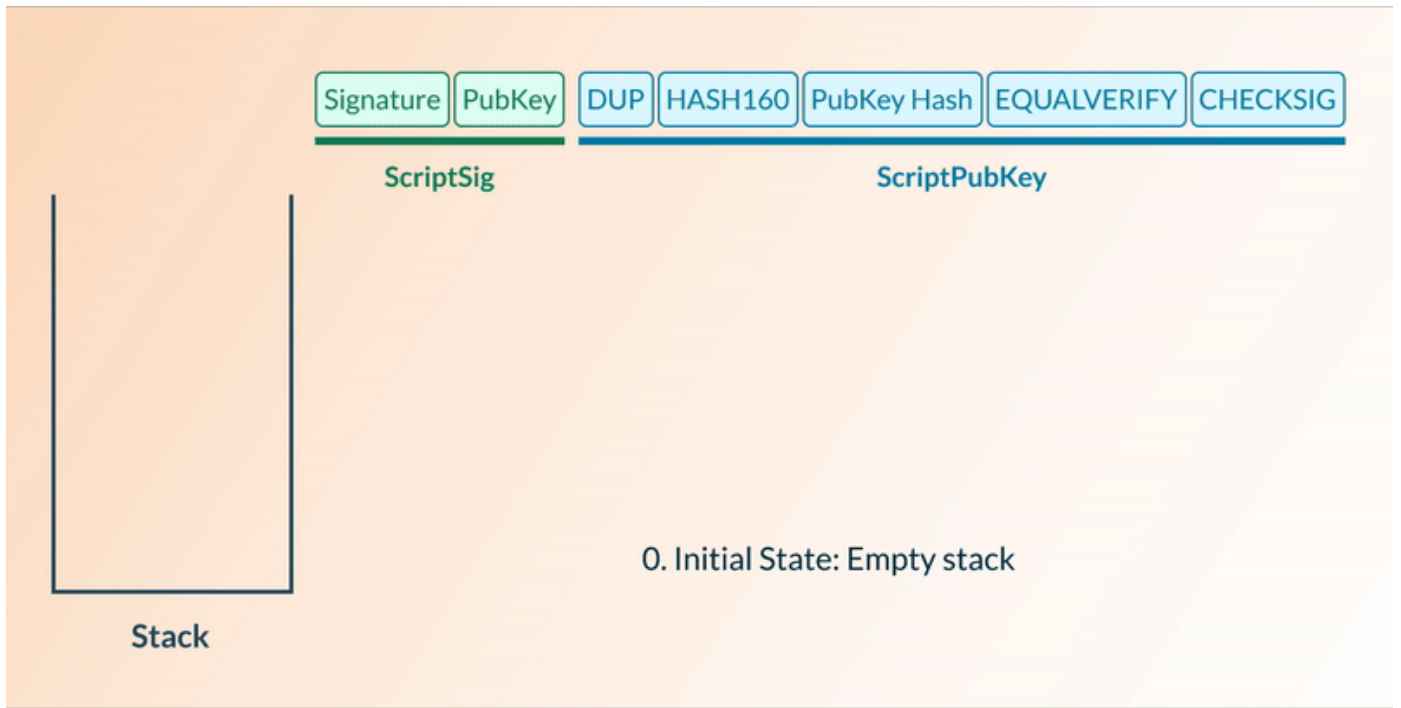
```
{
  "capacity": "0x19995d0ccf", // The size of the Cell (in shannons)
  "lock": {
    // A Script that defines the ownership of the Cell
    "code_hash":
    "0x9bd7e06f3ecf4be0f2fcd2188b23f1b9fcc88e5d4b65a8637b17723bbda3cce8",
    "args": "0x0a486fb8f6fe60f76f001d6372da41be91172259",
    "hash_type": "type"
  },
  "type": null // An optional Script that defines the type of the Cell.
}
```



Bitcoin Script Interpreter vs. CKB-VM

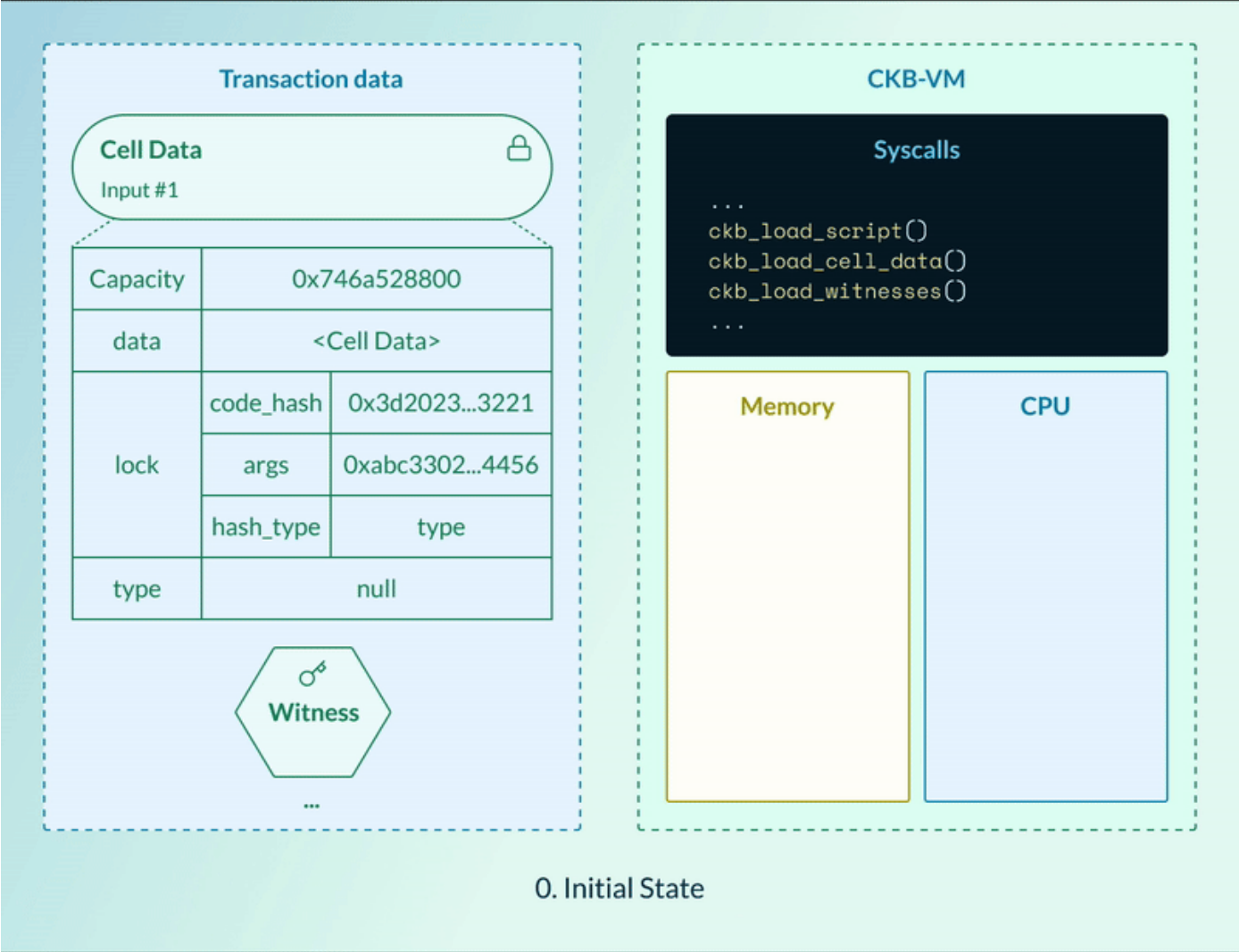
Bitcoin Script Interpreter

Bitcoin Script Interpreter는 **스택 기반**으로, 스크립트 실행 중 임시 데이터를 저장하기 위해 스택 자료 구조를 사용한다. 비트코인 스크립트의 연산은 데이터를 스택에 푸시(추가)하거나 팝(제거)한다. 다음 GIF는 단순 P2PKH(Pay to Public Key Hash) 같은 전형적인 비트코인 스크립트 동작을 묘사한다:



CKB-VM

CKB-VM은 현대적이고 오픈소스인 RISC-V 명령어 집합을 사용한다. 이러한 설계는 CPU에 대한 저수준 접근을 제공하여 매우 효율적인 실행과 유연성을 가능하게 한다. RISC-V를 대상으로 컴파일할 수 있는 모든 프로그래밍 언어는 CKB에서 네이티브하게 개발에 사용할 수 있다. 다음 GIF는 전형적인 CKB-VM 프로세스를 묘사하며, 스크립트가 트랜잭션 데이터에서 값을 로드하고 유효한 트랜잭션에는 0, 무효한 트랜잭션에는 1을 반환해 검증하는 과정을 보여준다.



BTC Script vs. CKB Script

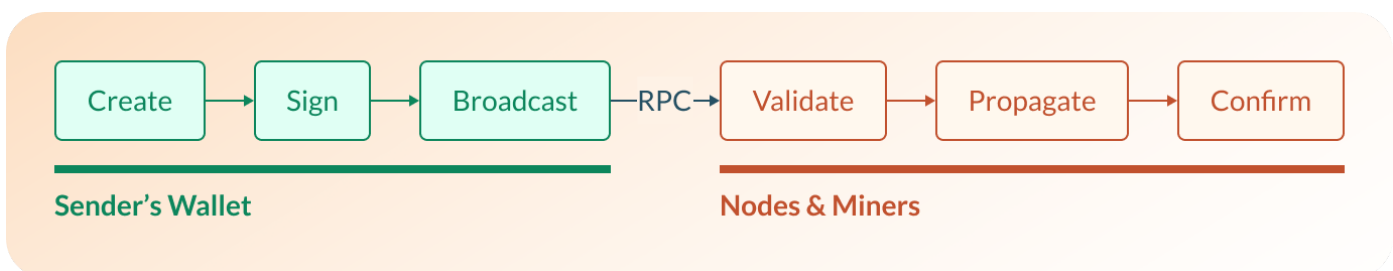
다음은 BTC Script와 CKB Script의 상세 비교이다:

Aspect	BTC Script	CKB Script
Statefulness	Stateless	Stateful; Can maintain and update state (data) across multiple transactions
Introspection	Limited. Proposed opcodes like OP_CTV to impose future spending constraints	Extensive. With Lock Script and Type Script to inspect and interact with other Scripts and the entire transaction.
Programmability	Limited by stack-based scripting and predefined opcodes	RISC-V based VM enables extensive programmability and multiple high-level languages
Script Structure	Series of opcodes operating on a stack, limiting its scope to basic transaction conditions	Written in high-level languages and compiled to the RISC-V instruction set
Use Cases	Transaction validation, basic contracts	Transaction validation, Smart contracts, dApps
Interoperability	Limited	Can run other VMs like EVM, enhancing cross-chain compatibility
Upgradeability	Requires network-wide hard forks for changes	Allows upgrades without hard forks

BTC 트랜잭션 vs CKB 트랜잭션

트랜잭션 라이프사이클

블록체인 시스템에서 트랜잭션 라이프사이클은 생성부터 확정까지 여러 단계를 거친다.



1. **Create:** 전송할 디지털 자산과 필요한 입력 및 출력을 지정해 트랜잭션을 구성한다.
2. **Sign:** 송신자가 자신의 개인 키를 사용해 트랜잭션을 인증하여 자산 이전의 소유권 또는 권한을 검증

한다.

3. **Broadcast**: 송신자의 지갑에서 가장 가까운 네트워크 노드로 트랜잭션을 브로드캐스트한다.
4. **Validate**: 노드는 무결성, 서명의 진위, 블록체인 프로토콜 준수, 그리고 사용자 정의 규칙을 검증하여 트랜잭션을 확인한다.
5. **Propagate**: 검증되면, 노드는 트랜잭션을 네트워크의 다른 노드로 전파한다. 트랜잭션은 블록에 포함되기를 기다리는 트랜잭션의 임시 저장소인 mempool에 진입한다.
6. **Confirm**: 트랜잭션이 포함된 블록이 채굴되어 블록체인에 추가되면, 해당 트랜잭션은 확정된 것으로 간주된다. 이후 이 블록을 참조하는 각 후속 블록은 확인 횟수를 증가시켜 트랜잭션을 보다 안전하게 만든다.

BTC 트랜잭션

비트코인 트랜잭션은 각 트랜잭션 출력이 향후 트랜잭션의 입력으로 사용될 수 있는 UTXO 모델 위에 구축된다.

▶ BTC 트랜잭션의 데이터 구조는 다음과 같다:

```
{
  "version": "01000000", // Transaction version number
  "inputcount": "02", // Number of inputs
  "inputs": [
    // List of transaction inputs
    {
      "txid":
"6059fa31ab283937854e8ba1128ae4572e7994af5b9e7a9450107f63a740dccc",
      "vout": "01000000", // Index of the output in the previous
transaction
      "scriptsigsize": "6b",
      // Script providing necessary data to satisfy the scriptPubKey
      "scriptsig":
"483045022100fed4209b8711a22e0d4a9e8b1d2c209b53180e89debbc13ea535d91a32b
76fe60220672c4554bc8119a5cb31b9f1741b0bc95eaaa7f457037aecbe00c740d1b8e1b
00121032de50ebf1ade927db3ced1e88ea9ddb8bc2503f486dc5b8ef5dd54f67ce12101"
    },
    {
      "sequence": "ffffffff"
    }
  ],
  {
    "txid":
"32074770640754938a380295c9f4408389637c1bd3cbbe0faa86264bcf667094",
    "vout": "00000000",
    "scriptsigsize": "6b",
```

```

    "scriptsig":
    "483045022100c52e23ca33145f9ce7d0e92e7f1386ae06f0a72bca00ce4bbb8ad969d15
    839020220213978ceadf9788961c52c164c605b40cc669a10ea53ce0a0d282d60606950a
    4012103c69ce119333de82dc0275539ad70bfae7f366e76a501fb2671052a1278bd8df2"
    ,
    "sequence": "ffffffff"
  }
],
"outputcount": "02", // Number of outputs
"outputs": [
  // List of transaction outputs
  {
    "amount": "94983f0300000000", // The value of the output in
    satoshis.
    "scriptpubkeysize": "19",
    "scriptpubkey":
    "76a914a82d981ccfa5cb703a4dcecc84d5b29797c307f288ac"
  },
  {
    "amount": "001bb70000000000",
    "scriptpubkeysize": "19",
    "scriptpubkey":
    "76a914d94afb03198c6cbc0118b50a62f5ae8508e3cf4388ac"
  }
],
"locktime": "00000000" // The earliest time that the transaction can
be added to the blockchain
}

```

- inputs 필드의 txid와 vout는 블록체인 상의 이전 트랜잭션에서 입력으로 사용 중인 UTXO가 무엇인지 지정한다.
- 각 입력 필드의 scriptsig에는 UTXO를 잠금 해제하는 데 필요한 디지털 서명과 공개키가 포함된다.
- 트랜잭션은 특정 amount로 전송 가치를 나타내는 새로운 outputs를 생성한다.
- 각 출력에는 scriptPubKey가 적용되어, 향후 트랜잭션에서 이 출력을 소비하기 위한 조건을 지정한다. 일반적으로 수신자의 공개키 해시와 서명을 검증하는 오퍼코드가 포함된다.

CKB 트랜잭션

UTXO 모델은 암호화폐 트랜잭션에 간단하고 효율적이지만, Nervos CKB의 Cell 모델은 더 큰 범용성을 제공하여 더 넓은 범위의 애플리케이션에 적합하다. Cell 모델은 상태(state)를 근본 요소로 중심에 둔다. CKB에서 트랜잭션은 Live Cell을 파기하고 새로운 Cell을 생성함으로써 상태를 업데이트하며, 파기된 Cell의 총 용량은 새로 생성된 Cell의 총 용량보다 크거나 같아야 하므로 추가 용량의 생성이 방지된다.

▶ CKB 트랜잭션의 데이터 구조는 다음과 같다:

```
{
  "version": 0, // Transaction version number
  "cell_deps": [ // An array of outpoint pointing to the Cells that are
dependencies of this transaction.
    {
      "out_point": { // A cell outpoint that point to the Cells used as
deps.
        "tx_hash":
"0xbd864a269201d7052d4eb3f753f49f7c68b8edc386afc8bb6ef3e15a05facca2",
        "index": "0x0"
      },
      "dep_type": "dep_group" // Dependency type (0 for Code, 1 for
DepGroup)
    }
  ],
  "header_deps": [ // An array of hashes pointing to block headers that
are dependencies of this transaction.
    "0xaa1124da6a230435298d83a12dd6c13f7d58caf7853f39cea8aad992ef88a422"
  ],
  "inputs": [ // An array of referenced Cell inputs.
    {
      "previous_output": {
        "tx_hash":
"0x8389eba3ae414fb6a3019aa47583e9be36d096c55ab2e00ec49bdb012c24844d",
        "index": "0x1"
      },
      "since": "0x0" // Timelock feature
    }
  ],
  "witnesses": [ // Provided by transaction creator to make the
execution of corresponding Lock Script success.
    "0x55000000100000005500000055000000410000004a975e08ff99fa0001
42ff3b86a836b43884b5b46f91b149f7cc5300e8607e633b7a29c94dc01c6616a12f62e
74a1
415f57fcc5a00e41ac2d7034e90edf4fdf800"
  ],
  "outputs": [ // An array of Cells that are used as outputs,
    {
      "capacity": "0x746a528800",
      "lock": {
```

```

        "code_hash":
"0x9bd7e06f3ecf4be0f2fcd2188b23f1b9fcc88e5d4b65a8637b17723bbda3cce8",
        "args": "0x56008385085341a6ed68decfab3ba1f3eea7b68",
        "hash_type": "type"
    },
    "type": null
},
{
    "capacity": "0x1561d9307e88",
    "lock": {
        "code_hash":
"0x9bd7e06f3ecf4be0f2fcd2188b23f1b9fcc88e5d4b65a8637b17723bbda3cce8",
        "args": "0x886d23a7858f12ebf924baaacd774a5e2cf81132",
        "hash_type": "type"
    },
    "type": null
}
],
"outputs_data": [ // An array of Cell data for each Cell output.
    "0x",
    "0x"
],
}

```

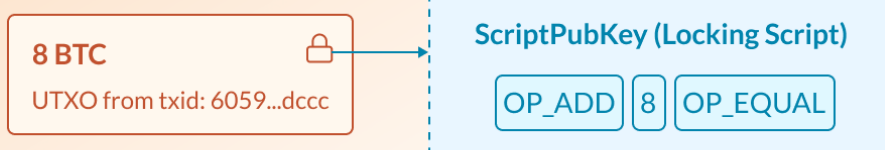
Unlocking & Verification Process

BTC 예시

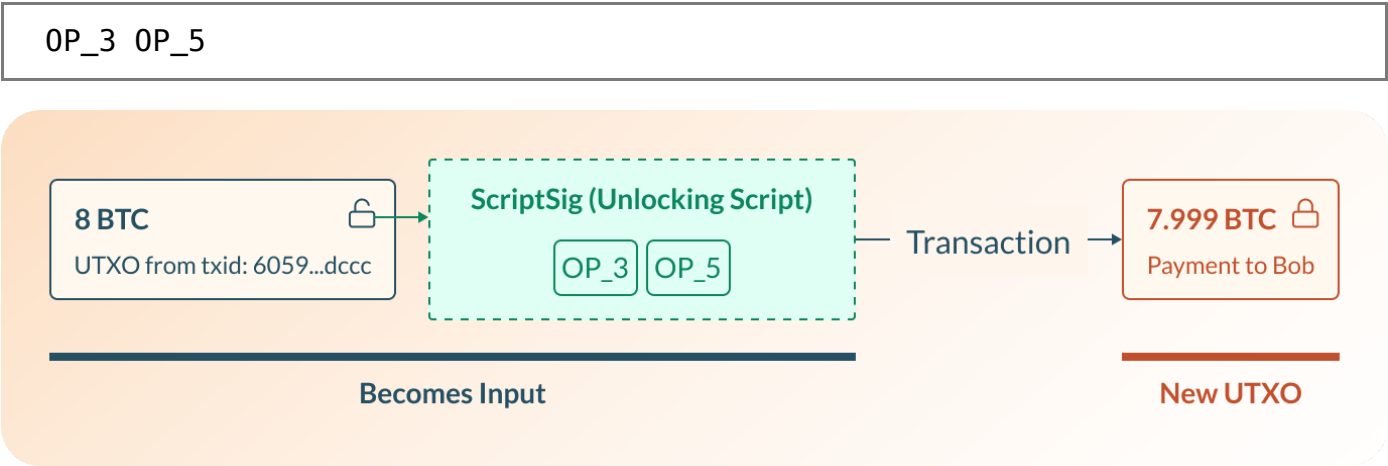
이 예시에서는 커스텀 락킹 스크립트로 보호된 UTXO를 Alice가 어떻게 소비할 수 있는지를 살펴본다. 이 스크립트는 트랜잭션을 유효화하기 위해 제공된 두 수의 합이 정확히 8이 되도록 요구한다.

UTXO의 ScriptPubKey에 저장된 락킹 스크립트는 토큰을 소비할 수 있는 조건을 지정한다. 이는 다음과 같이 정의된다:

OP_ADD <8> OP_EQUAL

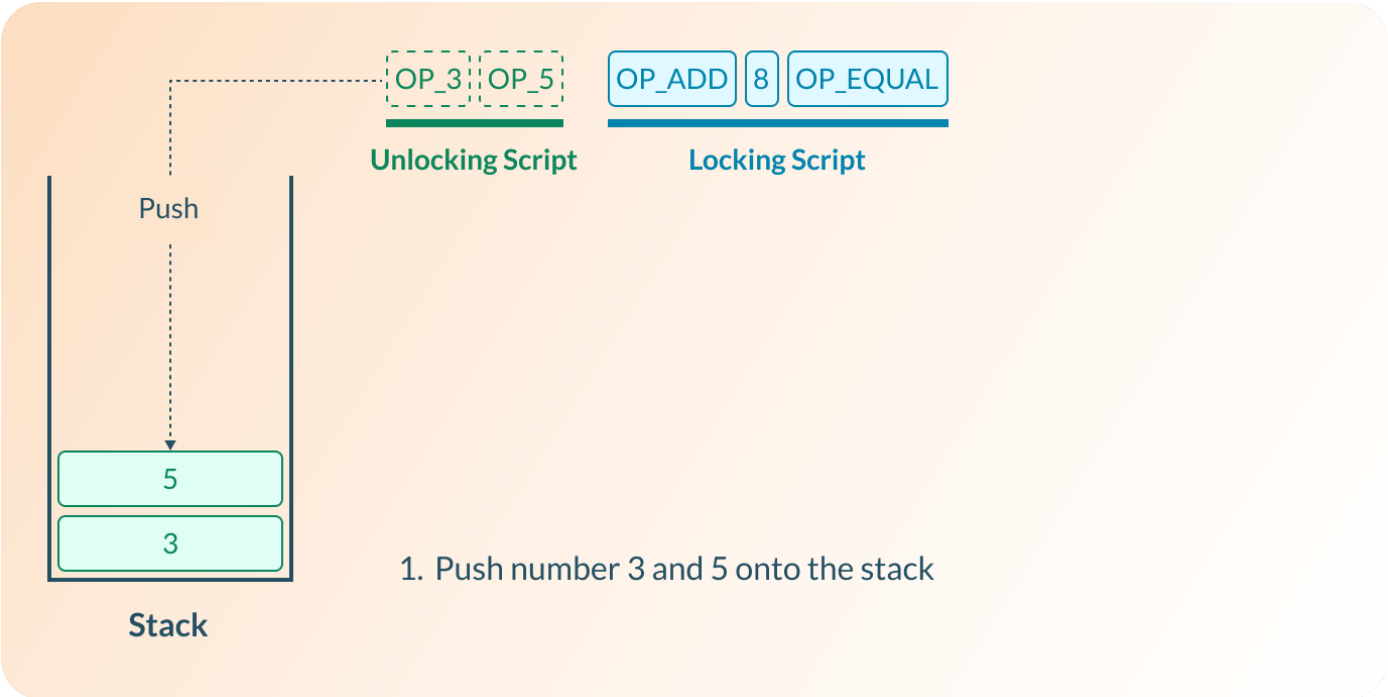


UTXO를 소비하기 위해 잠금을 해제하려면, Alice는 락킹 스크립트의 조건을 만족하는 언락킹 스크립트를 제공해야 한다. 이 경우, Alice는 다음을 제공한다:

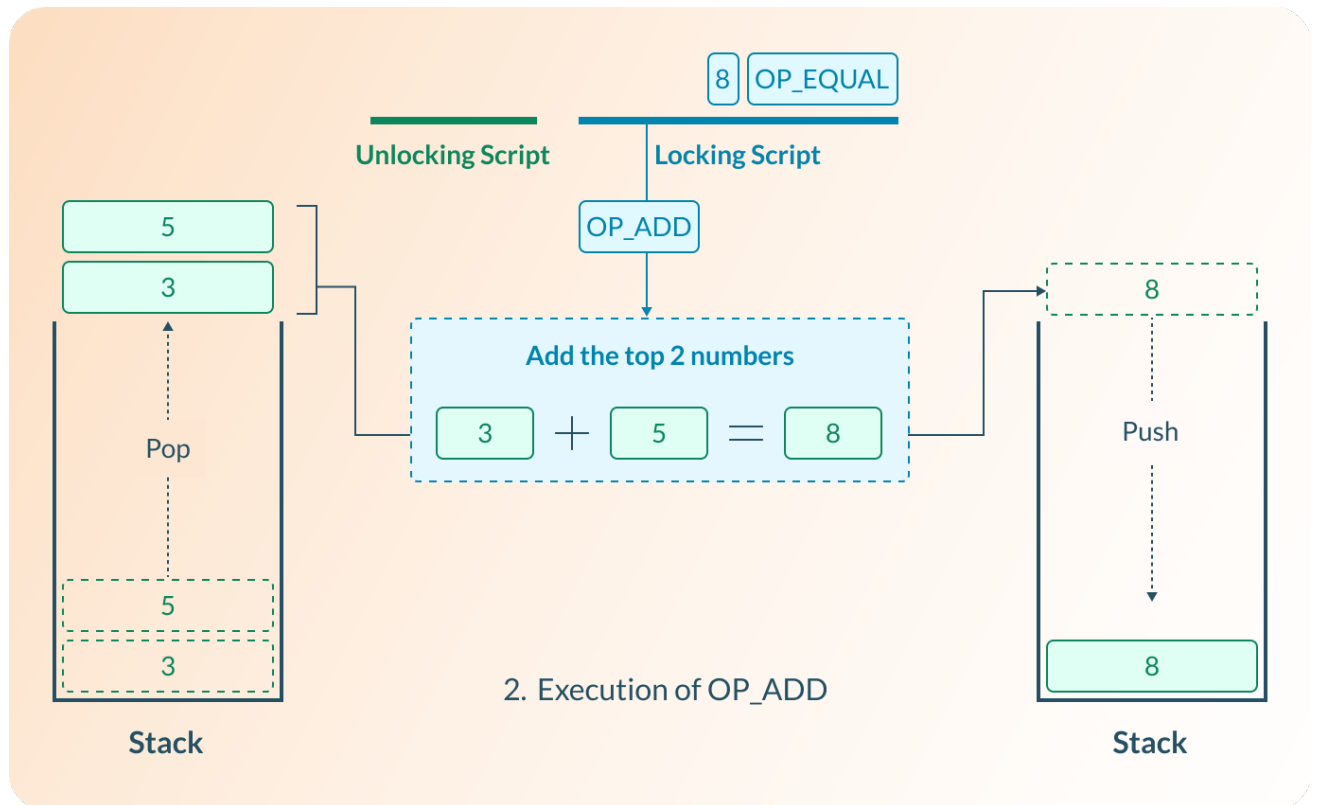


다음은 검증 과정의 단계별 분해이다:

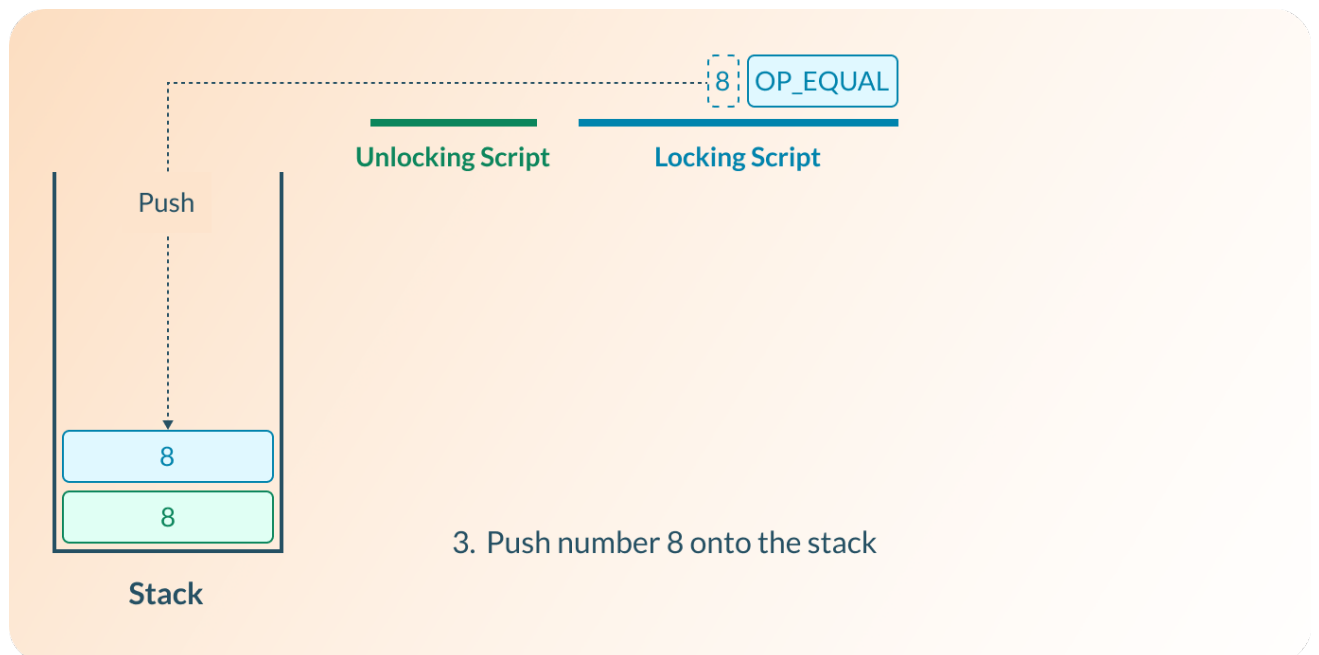
- 1. scriptSig에서 언락킹 스크립트 실행
 - OP_3는 숫자 3을 스택에 푸시한다
 - OP_5는 숫자 5를 스택에 푸시한다



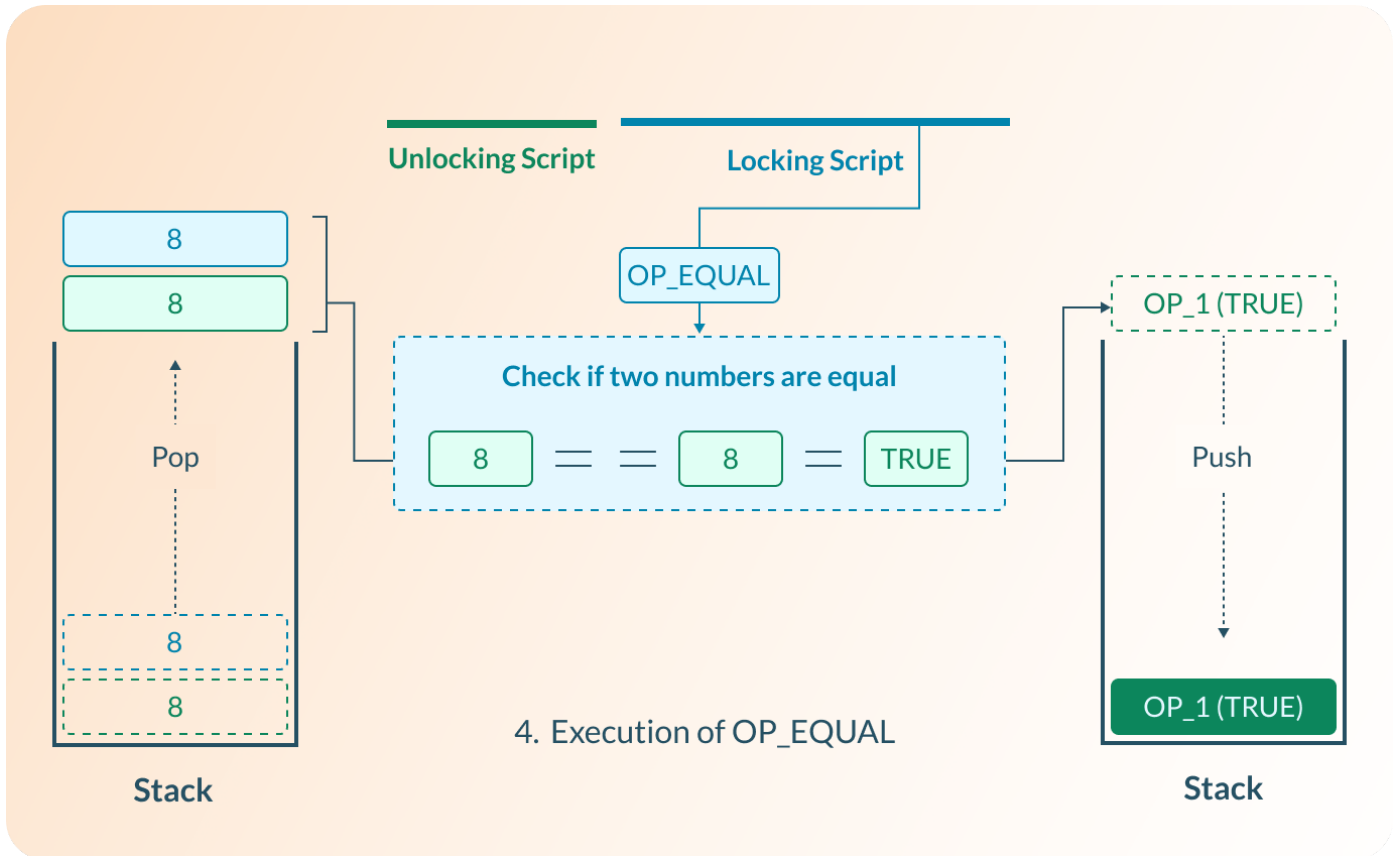
- 2. UTXO에 연결된 scriptPubKey의 락킹 스크립트 실행:
 - OP_ADD는 스택의 맨 위 두 숫자(5와 3)를 팝한 뒤 더하고, 결과(8)를 스택에 다시 푸시한다



3. <8>은 숫자 8을 스택에 푸시한다



4. `OP_EQUAL`은 스택의 맨 위 두 숫자(8과 8)를 팝하여 서로 같은지 확인하고, 결과(TRUE)를 스택에 푸시한다.



5. 스택에 남은 최종 값이 `TRUE`이면 트랜잭션은 **유효**하다고 간주된다. 스택이 `False`로 끝나거나 비어 있으면 트랜잭션은 **무효**이다.

CKB 예시

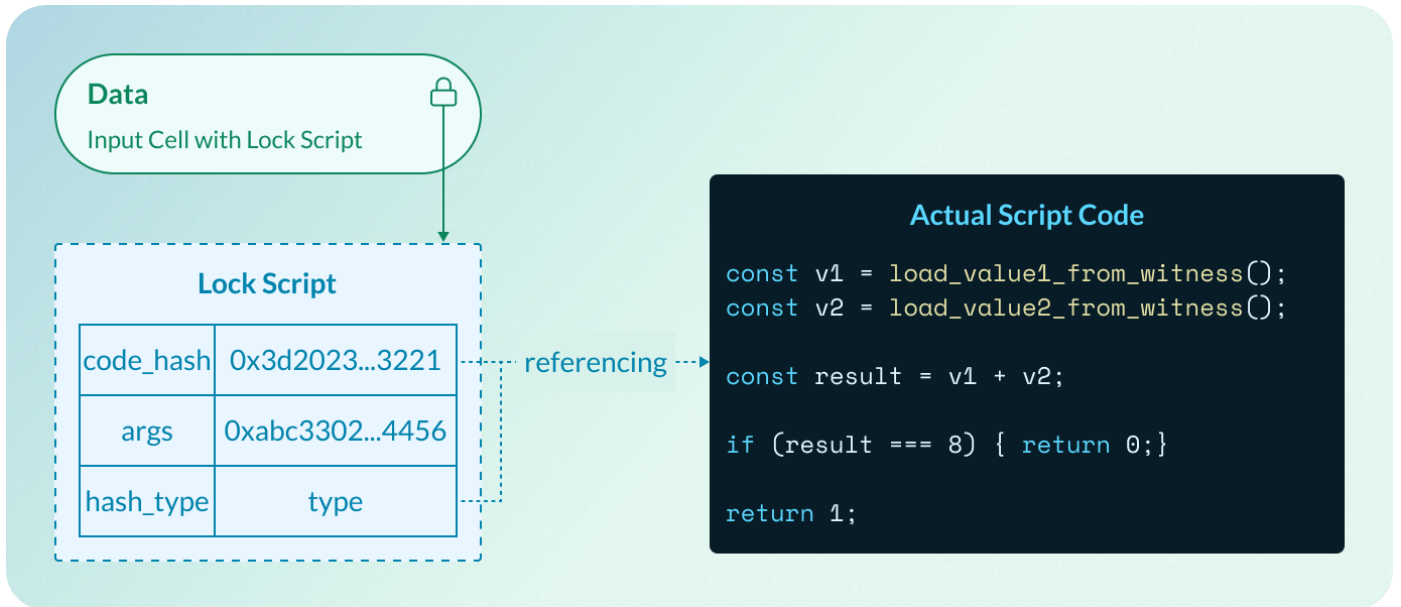
CKB 예시에서 스크립팅 로직은 비트코인의 스택 기반 스크립트 시스템보다 더 다재다능한 프로그래밍 환경에서 구현된다. 여기서는 BTC에서와 같은 예시를 사용하며, Alice가 제공된 두 수의 합이 8이 되어야 한다는 요구 조건으로 잠겨 있는 Live Cell을 소비하려 한다.

Lock Script를 표현하기 위해 의사코드를 사용한다:

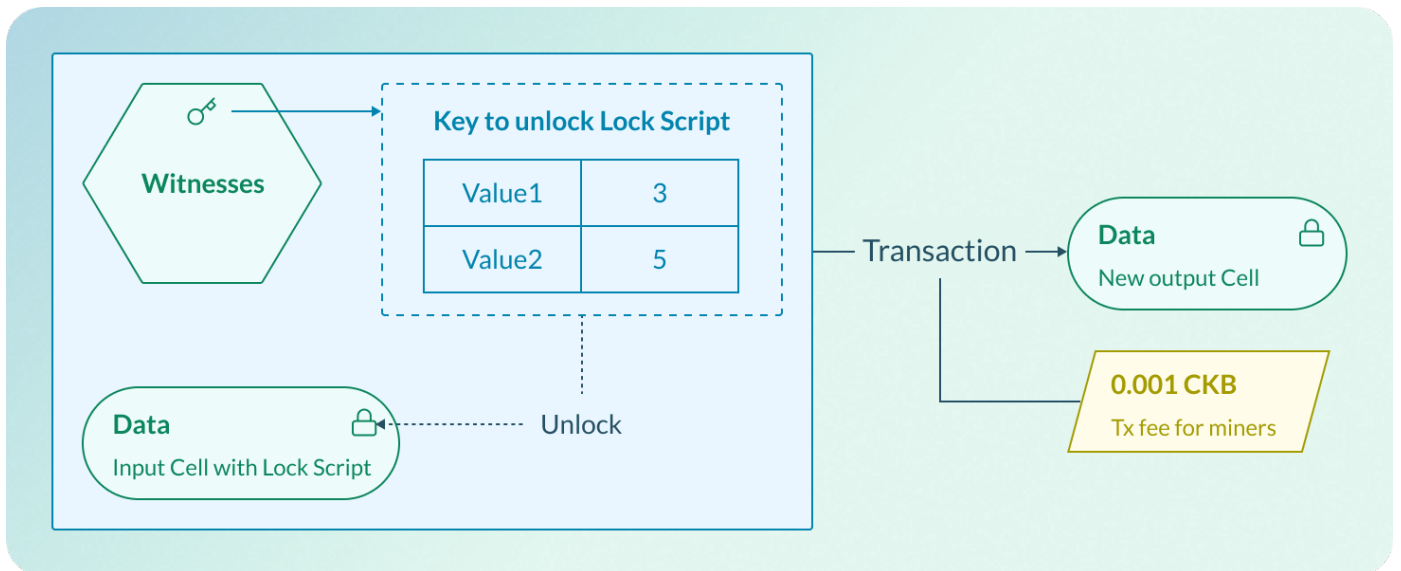
```
const v1 = load_value1_from_witness();
const v2 = load_value2_from_witness();

const result = v1 + v2;

if (result === 8) { return 0; }
return 1;
```



이 Cell의 잠금을 해제하여 소비하려면, Alice는 키(비트코인에서의 언락킹 스크립트와 유사)를 제공해야 한다. 이 경우, Alice는 witnesses 필드에 숫자 3과 5를 제공한다.



다음은 검증 과정의 단계별 분해이다:

1. Witness 데이터 로드
 - 값 v1(3)과 v2(5)는 witnesses 필드에서 로드된다

```
const v1 = load_value1_from_witness(); // number 3
const v2 = load_value2_from_witness(); // number 5
```

CKB-VM

Script

```
const v1 = load_value1_from_witness();
const v2 = load_value2_from_witness();

const result = v1 + v2;

if (result === 8) { return 0;}

return 1;
```

Memory

0x2000	11001010 00000001	<Script>
0x2004	00000000 01110100	<Cell Data>
0x2008	00000011 00000111	<Witness>

└ fetched and decoded

CPU

x0	3
x1	5
x2	
xN	

Witness values from the Memory are fetched and decoded to CPU's registers

2. 합 계산

- 스크립트는 두 수의 합을 계산한다.

```
const result = v1 + v2; // 3 + 5 = 8
```

CKB-VM

Script

```
const v1 = load_value1_from_witness();  
const v2 = load_value2_from_witness();  
  
const result = v1 + v2;  
  
if (result === 8) { return 0; }  
  
return 1;
```

CPU

x0	3
x1	5
x2	8
xN	

Add

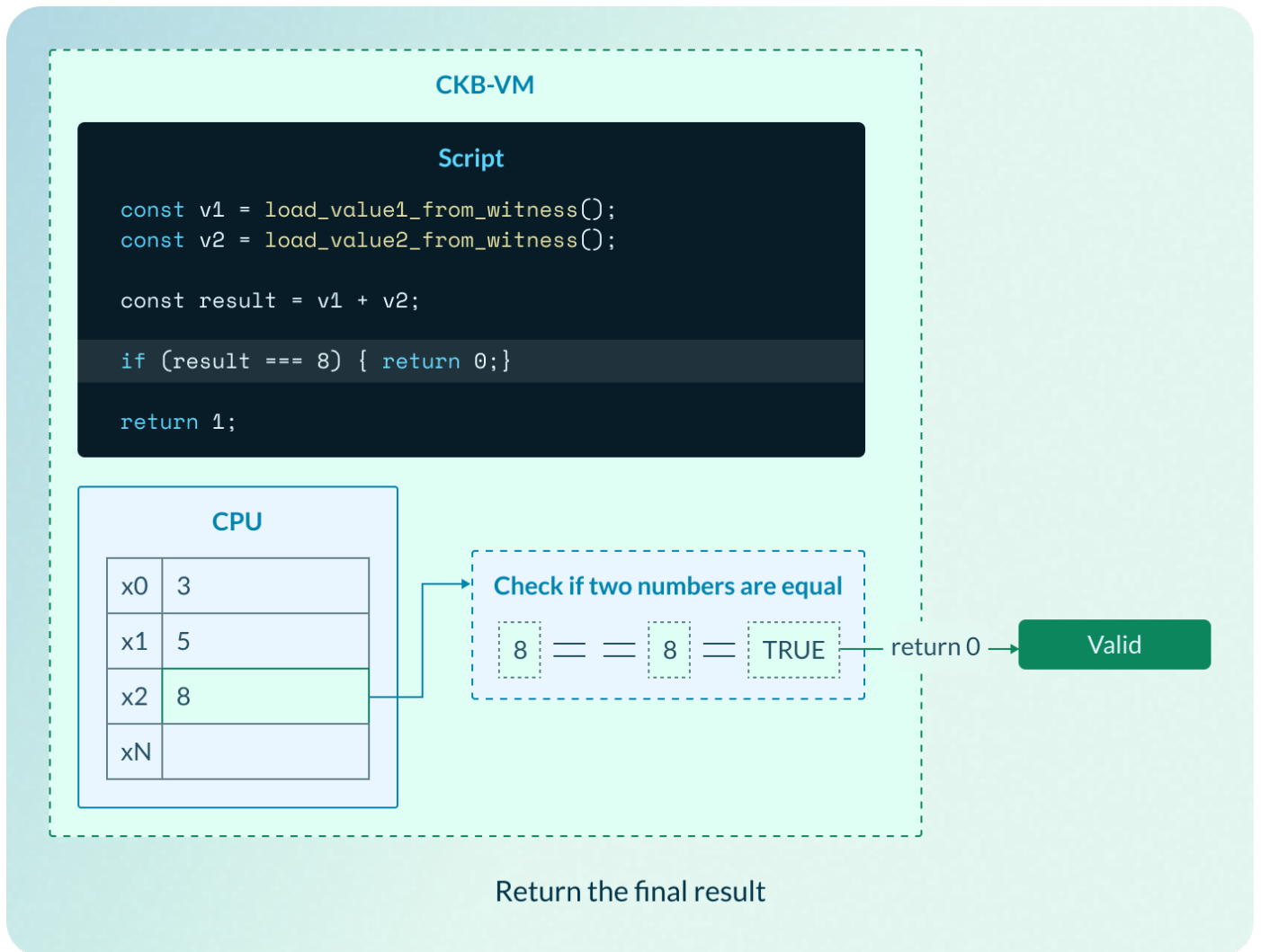
$$3 + 5 = 8$$

Sum the two values and store the result in another register

3. 조건 확인

- 이전의 `result`가 8과 같다면 CKB-VM은 0을 반환한다
- 그렇지 않다면 CKB-VM은 1을 반환한다

```
if (result === 8) { return 0; } // 8 === 8 in this case  
  
return 1;
```



4. CKB-VM은 반환값을 해석한다: 최종 결과가 0이면 트랜잭션은 **유효**로 간주된다; 그렇지 않으면 트랜잭션은 **무효**로 간주된다. 이 경우 $8 === 8$ 이므로 0이 반환되고 트랜잭션은 유효하다.

추가 자료

- [Cell 모델](#)
- [CKB-VM](#)
- [CKB 스크립트](#)
- [CKB 트랜잭션](#)

Cell

구조 {#cell-structure}

Cell은 상태를 저장하는 데 사용되는 Nervos CKB의 가장 작고 근본적인 단위이다.

```
Cell: {
  capacity: HexString;
  lock: Script;
  type: Script;
  data: HexString;
}
```

필드와 설명 {#fields-and-description}

Name	Type	Description
capacity	Uint64	Cell의 크기(단위: shannon). 1 CKB = 100,000,000 shannons.
lock	Script	Cell의 접근과 소유권을 강제하는 스크립트.
type	Script	트랜잭션에서 입력으로 소비되거나 출력으로 생성되기 위해 Cell이 따라야 하는 규칙을 강제하는 스크립트
data	Bytes	상태를 저장하는 데 사용.

예시

```
{
  "capacity": "0x19995d0ccf",
  "lock": {
    "code_hash":
"0x9bd7e06f3ecf4be0f2fcd2188b23f1b9fcc88e5d4b65a8637b17723bbda3cce8",
    "args": "0x0a486fb8f6fe60f76f001d6372da41be91172259",
    "hash_type": "type"
  },
  "type": null
}
```

용량(Capacity)

용량은 트랜잭션에서 Cell의 저장 한도를 의미한다. 이는 저장된 CKB 토큰의 양과 Cell 콘텐츠(data, lock, type, capacity 필드를 포함)에 사용할 수 있는 최대 공간을 동시에 나타낸다.

트랜잭션이 출력 Cell을 생성할 때, 해당 출력 Cell이 점유하는 용량은 원래 입력 Cell의 capacity를 초과할 수 없다:

$$\text{Occupied (Cell)} \leq \text{Cell's capacity}$$

트랜잭션 검증 규칙 중 하나는 다음과 같다:

$$\text{capacity_in_bytes} \geq \text{len(capacity)} + \text{len(data)} + \text{len(type)} + \text{len(lock)}$$

이는 Cell이 모든 필드를 저장할 충분한 용량을 갖도록 보장한다. capacity는 또한 CKB 토큰 잔액을 나타내며, 이는 비트코인의 CTxOut의 nValue 필드와 유사하다.

예를 들어, Alice가 100 CKB를 소유하고 있다면, 총 100바이트의 용량을 가진 일련의 Cell의 잠금을 해제할 수 있다.

최소 Cell 용량(61 Bytes){#capacity-mini}

각 Cell은 다음의 필수 필드를 저장하기 위해 최소 61 CKBytes를 예약해야 한다:

- capacity: 8 bytes
- lock: 32 bytes
- args: 20 bytes
- hash_type: 1 byte

이들을 합하면 61바이트가 된다. 1 CKByte는 온체인에 1바이트를 저장할 권리를 부여하므로, Cell은 이 기본 저장 공간을 충족하기 위해 최소 61 CKBytes를 보유해야 한다.

Cell 모델

Nervos CKB는 비트코인의 아키텍처를 계승하고 상태 저장소로서 일반화된 UTXO 모델인 Cell 모델을 만든다.

이 접근법은 비트코인의 단순성과 일관성을 유지한다.

CKB에서는 모든 상태가 Cell에 저장되고, 계산은 오프체인에서 수행되며, 노드가 모든 검증을 처리한다.

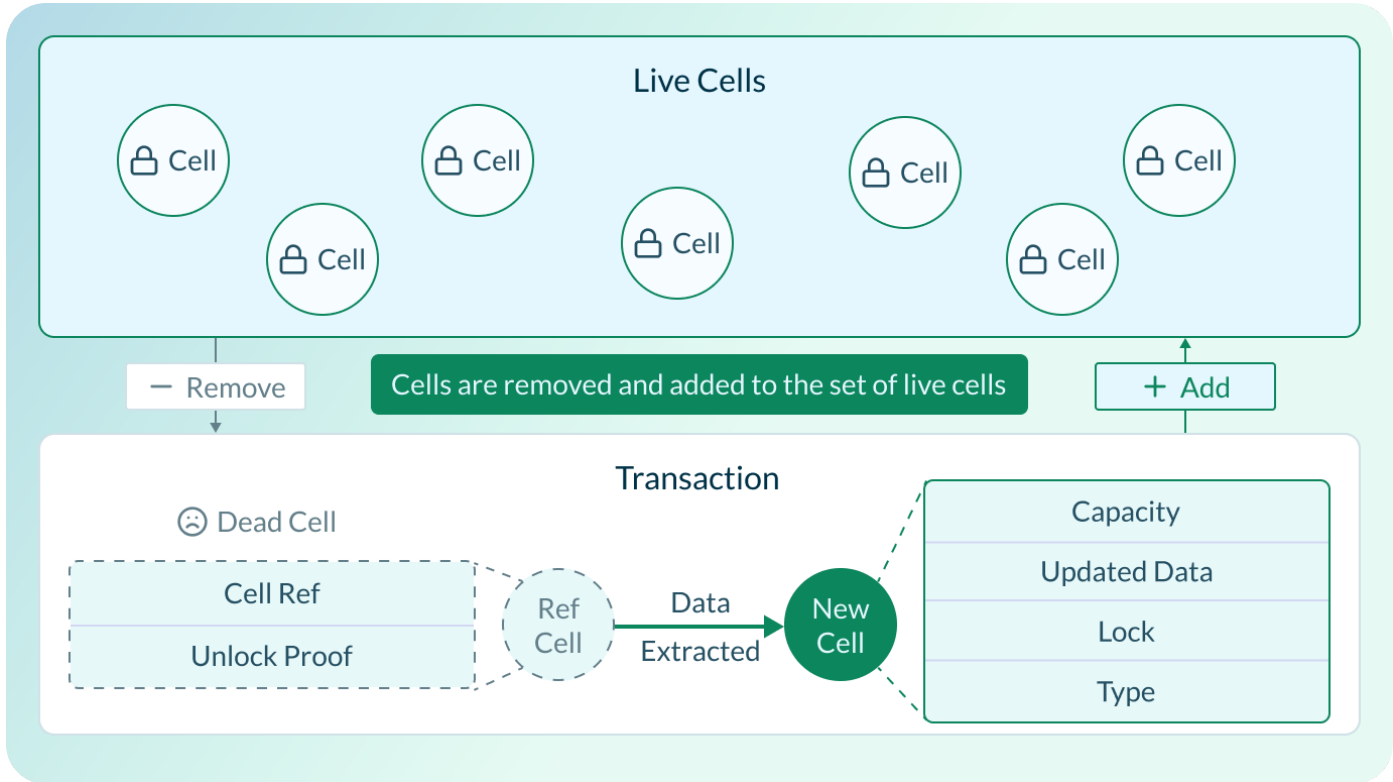
Cell 모델

비트코인의 UTXO 모델에서 영감을 받아, Cell 모델은 Nervos 내 개별 Cell의 동작과 그 안에 포함된 데이터를 업데이트하는 과정을 정의한다.

Cell은 불변이다. 일단 온체인에 추가되면 Cell은 변경될 수 없다. Cell 내부 데이터를 업데이트하려면 **소비 (Consumption)** 라는 과정을 거쳐야 한다. 이는 기존 Cell을 소비하여 데이터를 추출·수정한 다음, 업데이트된 데이터를 담은 새로운 Cell을 생성해 온체인에 추가하는 것을 의미한다.

각 Cell은 단 한 번만 소비될 수 있다. 소비되지 않은 Cell은 **Live Cell** 이다. 소비된 Cell은 **Dead Cell** 이다. 한 번 Dead가 되면 그 Cell은 더 이상 사용할 수 없다.

트랜잭션은 Live Cell의 집합을 소비하고 새로운 Cell을 생성하는 상태 변화를 반영한다. 네트워크는 관련된 모든 Lock Script와 Type Script를 실행하여 트랜잭션을 검증한다. 이는 개발자가 정의한 규칙 준수를 보장하고 부정 행위를 방지한다.



일급 자산(First-Class Assets)

Cell 모델에서는 모든 디지털 자산(예: CKBytes, 토큰, 컬렉티블)이 해당 소유자에게 독점적으로 귀속되는 일급 자산으로 간주된다. 자산은 트랜잭션 중 스마트 계약 규칙을 준수해야 하지만, 본질적으로 스마트 계약이 아니라 사용자에게 소유권이 있다. 이러한 소유 구조는 스마트 계약이 토큰을 어떻게 정의하든 오직 소유자만 자산을 사용할 권한을 갖도록 보장한다. 만약 계약이 악용되더라도, 자산은 사용자의 통제하에 남아 공격자가 접근할 수 없으므로, 부정적 영향을 효과적으로 완화한다.

이 소유 구조는 자산 유지 비용에 대한 책임도 정의한다. 자산이 Nervos 상의 공간을 점유하는 한, 소유자는 **상태 렌트(state rent)** 라는 소액의 반복적 유지 비용의 적용 대상이 되며, 이는 토크노믹스 섹션에서 자세히 설명된다.

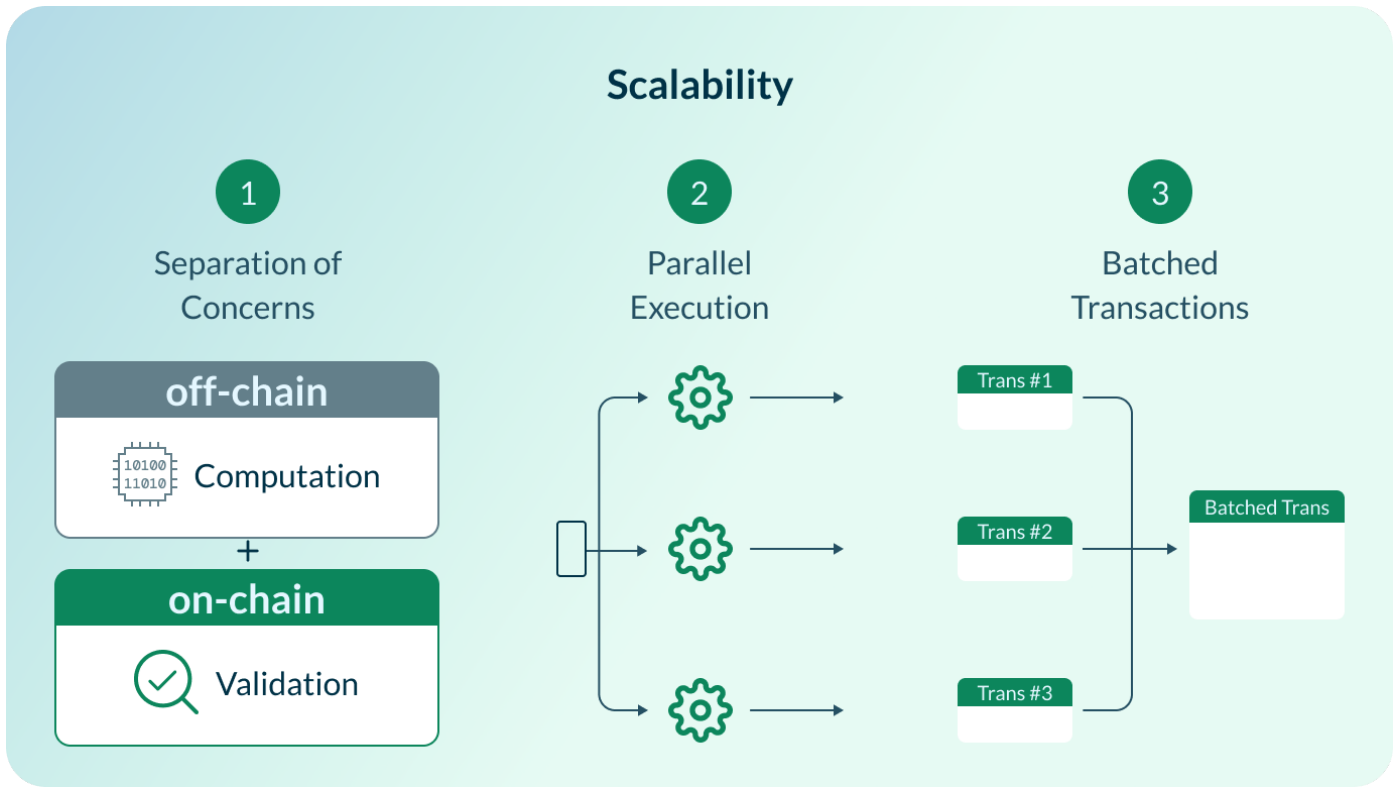
유연한 트랜잭션 수수료 부담

토큰을 전송할 때, 일반적으로 트랜잭션을 개시하거나 스마트 계약을 실행하는 당사자가 수수료를 부담해야 한다. 이는 채택 측면에서 사용성 문제를 야기한다.

Cell 모델은 어떤 당사자든 트랜잭션 수수료를 부담할 수 있도록 유연성을 제공하여, 발신자가 CKBytes(Nervos의 트랜잭션 수수료)를 보유할 필요를 제거한다. 대신 수신자나 제3자가 수수료를 부담할 수 있어 사용자 경험을 크게 개선한다.

확장성

Cell 모델의 고유한 구조는 아래 세 가지 관점에서 내재적 확장성을 부여한다.



Cell 모델은 스마트 계약 실행에서 계산과 검증을 분리한다. 계산은 오프체인에서 이루어지며, 여기서 새로운 데이터가 생성된다. 이후 이 데이터는 온체인 검증을 위해 네트워크로 전송된다. 전체 노드가 검증을 실행하여 개발자가 설정한 규칙 준수를 보장한다.

Cell 모델에서는 스마트 계약 실행이 병렬적이다. 각 트랜잭션은 자신의 가상 머신에서 독립적으로 실행되며, 여러 가상 머신이 동시에 실행된다. 이는 CPU 코어 수가 증가하는 현대 컴퓨터에서 드라마틱한 확장성 향상을 제공한다.

Cell 모델에서는 트랜잭션이 매우 유연하고 효율적이다. 여러 스마트 계약 연산을 단일 트랜잭션으로 배치할 수 있어, 트랜잭션 오버헤드와 처리 수수료를 최소화한다.

Cell 모델의 세부 내용과 그 논거는 [이 게시글](#) 을 참고하라.

합의(Consensus)

Nervos에서의 합의는 블록체인의 역사와 현재 상태에 대해 참여자들이 일치된 견해에 도달한 상태를 의미한다. 이 상태에는 각 사용자의 CKBytes 보유량과 각자의 디지털 자산과 같은 데이터가 포함된다.

수천 대의 컴퓨터와 수백만 건의 트랜잭션으로 구성된 Nervos 네트워크는 유효한 트랜잭션과 그 순서에 대해 지속적으로 합의를 필요로 한다. 전 세계적으로 분산된 노드와 불안정한 인터넷 전송 신뢰성 때문에 합의를 달성하는 데는 도전 과제가 따른다. 노드들은 트랜잭션과 블록 정보를 공유하기 위해 메시지를 교환하지만, 이러한 메시지는 순서가 뒤바뀌거나 지연되거나 아예 도착하지 않을 수도 있다. 또한 노드의 기능과 정직성을 보장할 수 없다. 상당한 가치가 걸려 있기 때문에, 합의를 달성하기 위한 견고한 해결책이 필수적이다.

지분증명보다 작업증명(Proof-of-Work Over Proof-of-Stake)

Nervos의 목표를 달성하기 위해서는 타협 없는 합의 메커니즘이 필요하다. 작업증명(PoW)은 다음과 같은 지분증명(PoS) 대비 장점을 바탕으로 최적의 선택으로 떠올랐다:

- **탈중앙화:** PoW 채굴은 채굴 장비, 에너지 소비, 규제 등 외부 요인에 적응한다. 경쟁력을 유지하려면 지속적인 재투자가 필요하므로, 장기적으로 독점화를 억제한다.
- **보안성:** PoW는 대안적 합의 메커니즘보다 가정을 덜 필요로 하는 더 단순하고 견고한 프레임워크를 제공한다. 이는 보안 취약성 가능성을 낮추고 전반적으로 더 안전한 네트워크를 보장한다.
- **공정성:** PoW는 초기 참여자에게 유리한 이점을 완화한다. 보상이 결정론적으로 부여되어 조기 참여자에게 유리한 PoS와 달리, PoW는 시간에 따라 보다 공정한 보상 분배를 보장한다.

NC-Max 합의 알고리즘

비트코인의 나카모토 합의(NC)는 10년이 넘는 기간 동안 무수한 공격으로부터 비트코인을 방어해온 PoW 알고리즘이다. 이 기술은 충분히 이해되었고 시간의 시험을 견뎌 입증되었다.

Nervos의 NC-MAX 합의는 나카모토 합의를 기반으로 하면서도 보안을 훼손하지 않고 NC의 한계를 해결한다. 트랜잭션 은닉 공격에 대해 강력한 저항성을 제공하며, 네트워크 처리량을 극대화하고 전통적 NC 구현 대비 트랜잭션 확정 지연을 상당히 줄이는 등 우수한 성능을 보여준다.

향상된 블록 전파

전통적 블록체인에서는 트랜잭션 전파 지연이 병목을 만들어 네트워크 혼잡과 잠재적 보안 취약성을 초래할 수 있다. NC-MAX는 확인 과정을 제안(propose)과 커밋(commit) 두 단계로 분할하여, 트랜잭션이 커밋되기 전에 완전히 전파되도록 함으로써 불완전한 트랜잭션 분배에 따른 지연과 취약성을 제거한다.

증대된 블록 처리량

블록체인 네트워크에서 블록 간격을 줄이면 트랜잭션 처리량을 높일 수 있지만, 네트워크 동기화 문제로 인해 고아 블록 비율이 높아질 수도 있다. NC-MAX는 네트워크 성능에 따라 블록 간격을 동적으로 조정해 처리량을 극대화하면서, 예상되는 고아 블록 비율을 유지한다. 이러한 적응적 접근은 네트워크 보안을 훼손하지 않고 더 짧은 블록 시간을 보장한다.

이기적 채굴(Selfish Mining)에 대한 강인한 저항

이기적 채굴은 채굴자가 덜한 해시 파워로도 과도한 보상을 얻도록 하여 네트워크 보안을 약화시킨다. NC-MAX는 네트워크의 연산 능력을 정확히 측정하고 알려진 이기적 채굴 공격을 완화함으로써 이 문제에 대응한다. 외부 및 내부 요인을 모두 고려하여, NC-MAX는 이기적 채굴 전략을 수익성 없게 만들어 네트워크 전반의 보안을 강화한다.

NC-MAX에 대한 포괄적 이해를 위해 [NC-Max: Breaking the Security-Performance Tradeoff in Nakamoto Consensus](#) 와 [RFC Consensus Protocol](#) 을 참고하라.

Eaglesong 해시 함수

비트코인의 나카모토 합의는 널리 사용되는 SHA256 해시 함수를 활용한다. 새로운 암호화폐가 SHA256 기반인 한 기존 채굴 인프라를 활용해 이점을 얻을 수 있다. 그러나 비트코인을 위한 막대한 인프라는 악의적으로도 사용될 수 있어, 잠재적 공격에 취약할 수 있다.

이 취약성을 해결하기 위해, Nervos는 생태계에 맞춘 새로운 암호학적 해시 함수인 Eaglesong을 개발했다. Eaglesong은 참신성, 단순성, 보안성의 균형을 제공하여 소프트웨어와 하드웨어 양쪽에서 구현이 용이하다. 이 혁신은 Nervos의 보안을 강화하고, SHA256에 대한 안전한 대안을 제공함으로써 완전한 하드웨어 주권을 보장한다.

Eaglesong에 대한 더 많은 정보는 [RFC Eaglesong](#) 을 참고하라.

CKB 주소

주소는 블록체인 자산을 수신하는 데 사용되는 숫자와 문자로 이루어진 긴 문자열이다.

CKB에서의 계정과 주소

비트코인은 계정과 주소 개념을 처음으로 분리한 블록체인이다. 하나의 계정이 여러 주소와 연결될 수 있는데, 이는 프라이버시 개선을 위한 설계이다.

이더리움의 부상과 함께, 많은 사용자가 하나의 계정 = 하나의 주소라는 다른 모델에 익숙해졌다. 각 계정은 애플리케이션 전반에서 재사용되는 단일 주소에 대응한다. 이는 상호작용을 단순화하지만, 모든 활동이 동일한 주소에 연결되어 프라이버시를 저하시킨다.

CKB에서는 모델이 비트코인과 더 유사하다: **하나의 계정이 여러 주소와 연결될 수 있다.**

계정과 주소는 독립된 개념이다. 계정은 키 쌍으로부터 도출되고, 주소는 기반 Lock Script로부터 생성된다. 동일한 계정 아래에서도 서로 다른 Lock Script는 서로 다른 주소를 생성한다.

CKB 주소가 생성되는 방식

이 모델이 어떻게 구현되는지 이해하려면, CKB 주소가 어떻게 구성되는지 살펴볼 필요가 있다. CKB 주소는 단순한 임의 문자열이 아니라, 비트코인의 Bech32m 주소 형식(BIP-350)을 따르면서, 기반 Lock Script를 인코딩한다. Lock Script는 소유권이 어떻게 검증되는지를 정의하며, `code_hash`, `hash_type`, `args`의 세 필드로 구성된다. 이들 필드에 대한 자세한 내용은 Script를 참조하라.

전체 페이로드 가져오기(Get Full Payload)

CKB 주소를 생성하기 위해, Lock Script는 “payload”라 불리는 바이트 배열로 인코딩되고, 이후 최종 주소 형식으로 래핑된다.

전체(Full) 페이로드 형식은 Bech32m 인코딩을 사용하여 Lock Script의 모든 데이터 필드를 인코딩한다:

```
payload = 0x00 | code_hash | hash_type | args
```

주소로 래핑(Wrap Into Address)

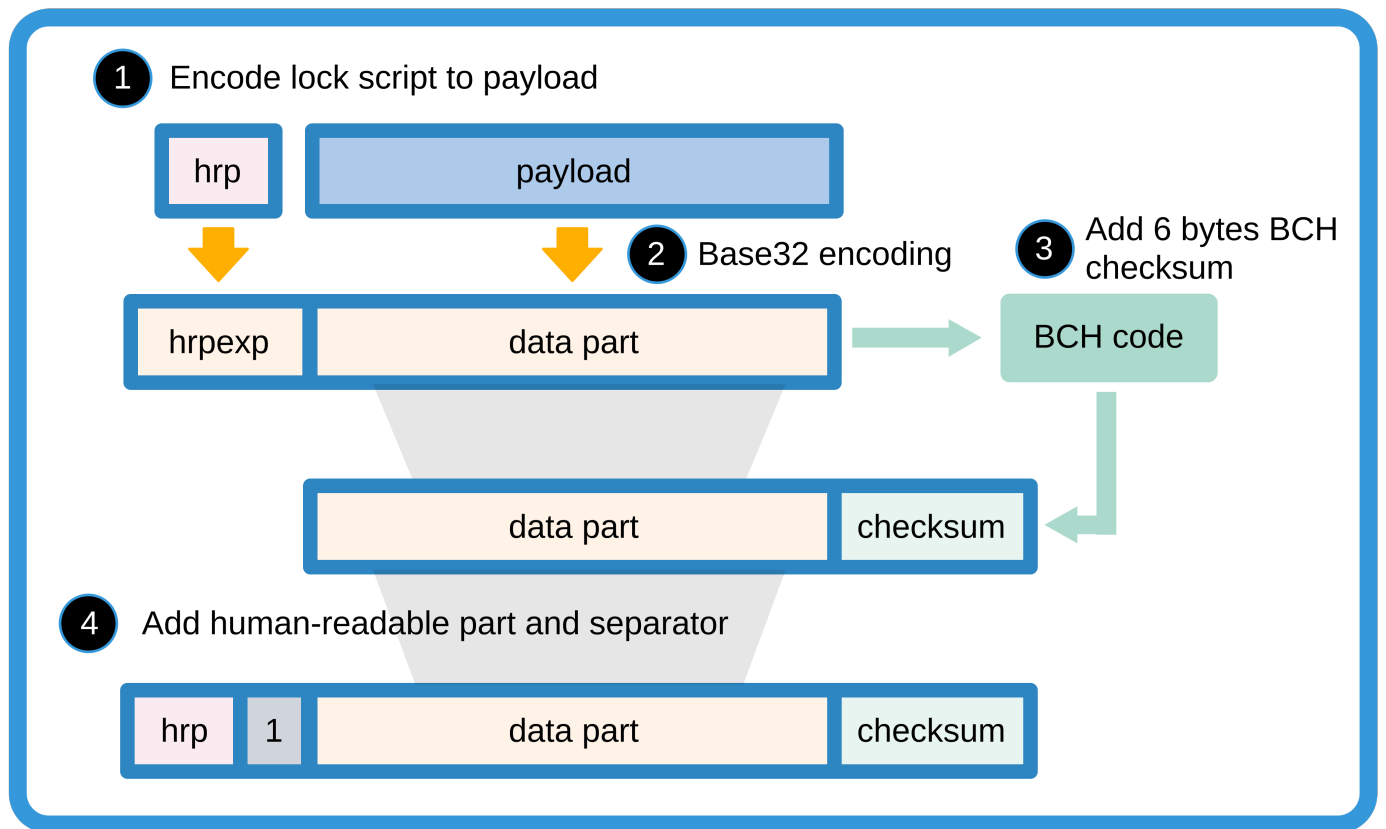
페이로드는 Bech32/Bech32m 인코딩과 BCH 체크섬을 사용하여 비트코인 Bech32 주소 형식(BIP-173) 또는 비트코인 Bech32m 주소 형식(BIP-350)을 따라 주소로 래핑된다.

Bech32/Bech32m 문자열은 세 부분으로 구성되며, 마지막 6자는 **체크섬**으로 사용된다:

- **사람이 읽을 수 있는 부분(HRP):** 메인넷은 "ckb", 테스트넷은 "ckt"
- **구분자:** 항상 "1"
- **데이터 부분:** base32 인코딩. 아래 표는 Base32 인코딩 매핑을 제공한다:

	0	1	2	3	4	5	6	7
+0	q	p	z	r	y	9	x	8
+8	g	f	2	t	v	d	w	o
+16	s	3	j	n	5	4	k	h
+24	c	e	6	m	u	a	7	l

아래 플로우차트는 Lock Script를 페이로드로 인코딩하기 위해 6바이트 BCH를 추가하고, 이어 Base32로 인코딩한 뒤, 마지막으로 사람이 읽을 수 있는 접두사와 체크섬을 덧붙이는 과정을 개괄한다.



예시: 전체 주소 생성

```
code_hash to encode:
9bd7e06f3ecf4be0f2fcd2188b23f1b9fcc88e5d4b65a8637b17723bbda3cce8
hash_type to encode:      01
args to encode:           b39bbc0b3673c7d36450bc14cfcdad2d559c6c64
---
full address generated:
ckb1qzda0cr08m85hc8jlnfp3zer7xulejywt49kt2rr0vthywaa50xwsqdnw7qkdnnc1fk
g59uzn8umtfd2kwxceqxwquc4
```

CKB에서의 해싱(Hashing in CKB)

CKB는 기본 해시 알고리즘으로 BLAKE2b 를 사용한다. CKB에서 사용하는 BLAKE2b의 구체적 구성은 ckbhash로 지칭되며, 다음과 같은 특징을 가진다:

- 32바이트 출력 다이제스트 크기
- ckb-default-hash 라는 personalization

CKB-VM

Nervos는 RISC-V 명령어 집합으로 구동되는 가상 머신 CKB-VM을 도입하여, 튜링 완전한 프로그래머 빌리티를 더욱 활용하고 Nervos 블록체인에서의 스마트 계약 실행을 변모시킨다. 이는 안전하고 고성능의 환경을 유지하면서 상당한 수준의 성능과 유연성을 제공한다.

RISC-V

RISC-V 는 프로세서 설계를 위한 기반이 되는 명령어 집합 아키텍처이다. 소프트웨어 스택의 최하위 계층으로서, CPU에 원시 명령을 직접 제공한다. RISC-V 표준은 성숙하고 정립되어 있으며 현대 하드웨어 개발을 위해 구축되었다. 광범위한 인지도를 바탕으로, RISC-V는 블록체인 개발에 이상적인 모듈식, 하위 호환 설계를 제공한다.

향상된 유연성과 단순화된 개발 경험

CKB-VM을 사용하면, 개발자는 하드포크 없이 암호학 기본 요소(예: Schnorr, BLS, zk-SNARKs, zk-STARKs)를 스마트 계약에 매끄럽게 통합할 수 있다. 이 과정은 코드베이스에 새 라이브러리를 추가하는 것만큼 간단하다. 이는 교차 체인 상호작용, 확장 혁신, 보안 하드웨어 엔클레이브와의 무결점 통합 같은 첨단 기술의 채택을 가속화하며, 더 원활한 개발자 경험을 촉진한다.

RISC-V를 대상으로 할 수 있는 모든 프로그래밍 언어는 Nervos에서 네이티브하게 개발에 사용할 수 있다. JavaScript에서 Rust에 이르기까지, 개발자는 선호하는 도구와 언어를 활용할 수 있어, 검증되지 않은 도구를 사용할 필요가 없고 효율적이고 안전한 스마트 계약 배포가 가능하다.

Nervos CKB는 JavaScript, Rust, Go, Java, Ruby 등 여러 주류 프로그래밍 언어에 대한 네이티브 SDK를 제공한다.

CKB-VM 버전

CKB 네트워크는 보안성 및 성능 향상, 버그 수정, 새로운 RISC-V 확장 지원을 위해 시간이 지남에 따라 다양한 CKB-VM 버전을 도입해 왔다. 과거 하드포크 이벤트로부터의 모든 CKB-VM 버전을 확인하려면 [VM Version History](#) 를 방문하라.

CKB-VM에 대한 더 많은 정보는 [RFC CKB-VM](#) 을 참고하라.

핵심 구조

스크립트(Script)

구조

CKB 프로토콜에 의해 정의된 규칙 외에도, 트랜잭션의 입력 및 출력 Cell에 포함된 스크립트에 기반한 검증이 추가로 요구된다.

```
Script: {
  code_hash: HexString
  args: HexString
  hash_type: Uint8, there are 4 allowed values: {0: "data", 1: "type",
2: "data1", 4: "data2"}
}
```

필드와 설명{#fields-and-description}

Name	Type	Description
code_hash	H256	CKB 스크립트를 포함하는 ELF 형식 RISC-V 바이너리의 해시. 자세한 내용은 code_hash 참조
hash_type	Uint8, one of the 4 values: {0: "data", 1: "type", 2: "data1", 3: "data2"}	일치하는 dep cell을 찾을 때 code hash의 해석 방식. 자세한 내용은 hash_type 참조
args	H256	스크립트 입력으로서의 인자. 자세한 내용은 script_args 참조

Script vs. Code

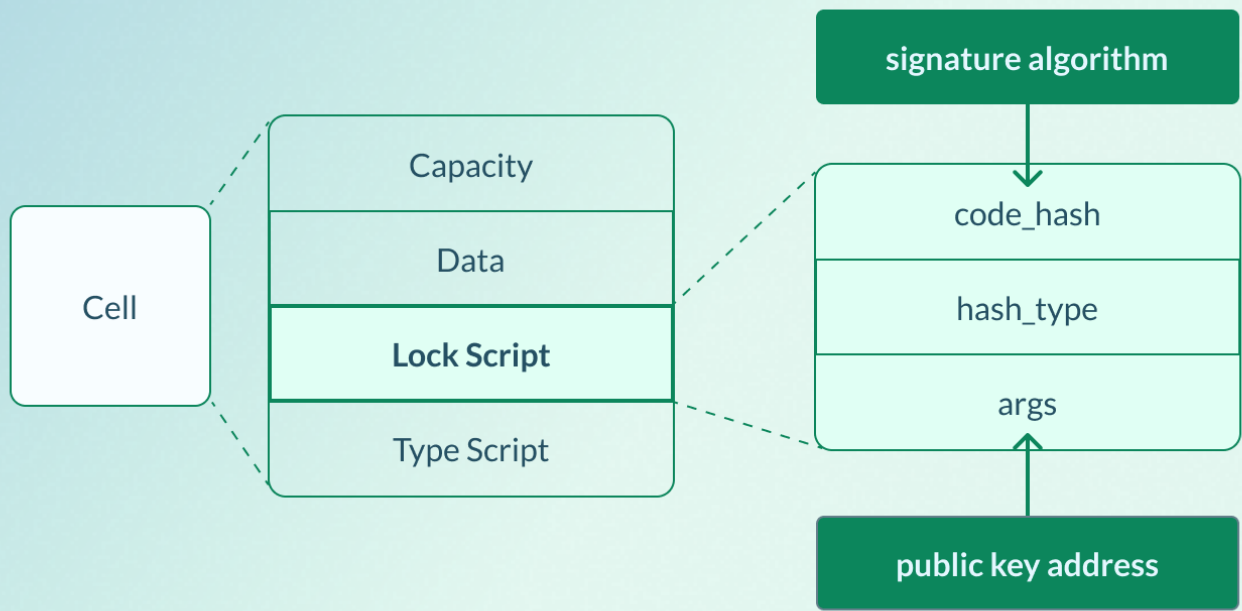
다음과 같이 Script와 Code 용어를 구분한다:

- Script는 온체인에서 실행 가능한 코드를 참조하는 데이터 구조이다.
- Code는 CKB-VM에서 실행 가능하며 Script 구조에 의해 참조될 수 있는 RISC-V 바이너리를 의미한다.
- 코드 Cell은 RISC-V 바이너리 코드를 포함하는 Cell이다.
- Script는 코드를 직접 포함하지 않으며, 단지 코드를 참조하기 위한 포인터를 저장한다.

Lock Script

이는 Cell의 소유권을 결정하는 스크립트를 정의하며, Cell 소유권과 관련된 로직을 관리하는 데 사용된다.

일반적인 Lock Script는 `code_hash`에 참조된 서명 알고리즘을 저장하는 Cell의 해시 값을 참조하고, `args`에 공개키 주소와 관련된 파라미터를 저장한다.



이 Cell을 잠금 해제할 때, 즉 Lock Script 검증을 완료할 때 사용자는 witness에 서명 정보를 넣어야 한다. 검증 과정에서 Lock Script는 서명 정보와 색인된 서명 알고리즘을 통해 결과가 args와 일치하는지를 검증하여, 트랜잭션이 실제로 입력 Cell 소유자에 의해 서명되었음을 보장한다.

비트코인의 CTxOut에 있는 scriptPubKey와 유사하게, 스크립트를 성공적으로 실행하기 위한 언락킹 파라미터를 제공할 수 있는 누구나 해당 Cell을 전송의 입력으로 사용할 수 있다(즉, Cell의 소유권을 가진다).

Type Script {#type-script}

이는 Cell의 유형을 결정하며, 일반적으로 Cell 변환 조건 또는 상태 전이 제약을 정의한다. 새로운 Cell의 data가 기존 Cell로부터 어떻게 변환되는지를 지정한다. type은 Script와 같은 데이터 구조를 가져야 한다.

하나의 트랜잭션에서 입력 Cell과 출력 Cell이 동일한 Type Script를 공유하는 경우, 그 사이의 변환은 해당 Type Script가 정의한 규칙을 따라야 한다.

Type Script의 전형적인 사용 사례는 UDT이다. 토큰 발행에서, 새로운 토큰은 새로운 Cell 출력으로 생성되므로, 이 새로운 토큰들이 사전에 정의된 규칙을 준수해 생성되었는지 검증하기 위해 Type Script가 필요하다.

Lock Script vs. Type Script

Type Script는 Lock Script와 유사하지만 두 가지 차이가 있다:

- Type Script는 선택 사항이다.
- CKB는 트랜잭션에서 입력에 대해서만 Lock Script를 실행한다.
- CKB는 트랜잭션에서 입력과 출력 모두에 대해 Type Script를 실행한다.

Lock Script와 Type Script는 서로 다른 목적을 가진다. Lock Script는 자산 소유권을 위한 것이고, Type Script는 애플리케이션 로직을 위한 것이다.

Lock Script는 입력에 대해서만 실행되며, 주로 Cell의 소비를 검증하는 역할을 한다. 이를 통해 정당한 소유자만 Cell을 입력으로 사용하고 그 안에 저장된 토큰을 소비할 수 있도록 보장한다.

반면에 Type Script는 동일한 유형의 Cell에 대한 상태 전이 계약으로 기능한다. 특정 타입이 지정된 Cell 출력을 접할 때, 해당 Cell이 그 Type Script에 코딩된 검증 규칙을 충족했음을 확신할 수 있다. 또한 이 코드는 Cell이 파기(입력으로 사용)될 때에도 실행된다.

예를 들어: Alice는 자신이 보유한 특정 UDT의 잔액이 `data` 필드에 저장되어 있고, `type script` 필드가 그 UDT의 규칙과 로직을 정의하는 Cell을 소유하고 있다. Alice가 일부 토큰을 Bob에게 보내려면, 이 Cell을 입력으로 사용하고 Bob의 `lock script`를 사용하는 동일한 `type script`의 출력 Cell을 생성해야 한다. 그런 다음 Alice는 트랜잭션을 조립해 CKB 노드로 전송할 수 있다. 단, Alice가 이 Cell의 모든 토큰을 보내지 않는 경우, 잔돈으로 토큰을 보관하기 위해 자신을 위한 추가 출력 Cell을 생성해야 한다.

code_hash

`code_hash`는 CKB 스크립트를 포함하는 ELF 형식 RISC-V 바이너리의 해시를 의미한다. 효율성을 위해 스크립트 자체는 현재 트랜잭션에 첨부된 `dep Cell`에 저장된다. `hash_type` 값에 따라, `code_hash`는 `dep Cell`의 데이터 해시 또는 `dep Cell`의 Type Script 해시와 일치해야 한다. 트랜잭션 검증 중 지정된 경우, 실제 바이너리는 CKB-VM 인스턴스로 로드된다.

hash_type

`hash_type`는 일치하는 `dep Cell`을 찾을 때 `code_hash`의 해석 방식을 정의한다. 이 필드의 각 `hash_type` 값은 아래와 같이 특정 JSON 표현, 해시 매칭 방식, 그리고 호환 VM 버전에 대응한다:

hash_type	JSON Representation	Matched by	VM Version
0	"data"	Data hash (blake2b)	0
1	"type"	Type Script hash	2
2	"data1"	Data hash (blake2b)	1
4	"data2"	Data hash (blake2b)	2

VM 버전과 선택 방법에 대해 더 알아보려면, [RFC0032: CKB VM Version Selection](#) 을 방문하라.

"Data Hash vs. Type Hash"

Introduction

hash_type 필드는 스크립트 바이너리를 찾을 때 code_hash 를 어떻게 해석해야 하는지를 지정한다. 유효한 hash_type 값은 4개이다:

hash_type	Code Location Method	VM Version	Upgradeable	Description
data	Hash of Script binary	v0	No	Immutable Script on CKB-VM v0
data1	Hash of Script binary	v1	No	Immutable Script on CKB-VM v1
data2	Hash of Script binary	v2	No	Immutable Script on CKB-VM v2
type	Hash of a Cell's Type Script	Latest	Yes	Upgradeable Script

이들은 두 가지 범주로 나뉜다:

- **Data hash(data, data1, data2):** 스크립트 바이너리에 대한 직접 참조이다. 코드를 불변이며 재현 가능하게 보장한다.
- **Type hash(type):** 코드 Cell의 Type Script 기반으로 스크립트를 참조한다.

Data Hash

hash_type 이 data(또는 data1, data2) 인 경우, 스크립트는 스크립트의 code_hash 와 Cell 데이터의 Blake2b 해시를 매칭하여 코드 Cell을 찾는다. 이는 스크립트가 트랜잭션의 cell_deps 중 하나에 코드의 정확한 바이너리 콘텐츠가 존재할 것을 기대함을 의미한다.

data 유형 참조를 사용할 때는 일치하는 코드를 찾기 위해 cell_deps 만 스캔하므로, 범위는 트랜잭션에 명시적으로 포함된 Cell로 제한된다. 이는 결정적 동작과 실행할 코드에 대한 엄격한 통제를 보장한다.

Type Hash

type hash로 위치 지정되는 코드는 동일한 Type Script를 사용하는 새로운 코드 Cell로 교체하여 업그레이드할 수 있다. 새로운 Cell에는 업데이트된 코드가 포함된다. 트랜잭션이 이 새로운 코드 Cell을 cell_deps 에 포함하면 업데이트된 버전이 실행된다.

이 업그레이드 과정에는 원래 코드 Cell의 파기가 필요하기 때문에, 업그레이드 조건은 해당 Cell의 Lock Script에 의해 강제될 수 있다 — 예를 들어, 누가 업그레이드를 트리거할 수 있는지 또는 어떤 조건에서 허용되는지 제한할 수 있다.

type hash는 스크립트가 일치하는 Type Script를 가진 임의의 Cell에서 코드를 찾을 수 있는 방법을 제공한다. 어떤 Cell의 Type Script가 기대하는 code_hash 와 일치하고 그 Cell이 cell_deps 에 포함되어 있기만 하면, 해당 코드가 실행에 사용된다.

:::caution Advanced: Potential for Malicious Code Substitution

type hash 사용은 유연성을 제공하지만, 미묘하고 위험한 리스크도 초래한다:

- 서로 다른 코드 내용을 담고 있으면서도 동일한 Type Script와 type hash를 공유하는 다수의 Cell이 존재할 수 있다.
- 공격자는 동일한 해시를 공유하는 결함 있거나 악용 가능한 코드 버전을 만들어, 스크립트가 이를 모른 채 의존할 경우 치명적 취약점으로 이어질 수 있다.
이러한 이유로, 이 접근은 Type Script 자체에 제한 조건(예: 생성자 신원 검증)을 구현하는 방법을 잘 이해하는 고급 개발자에게만 권장된다 — 승인되지 않은 코드 Cell이 수용되는 것을 방지하기 위해서다. 스푸핑을 막고 유일성을 보장하기 위해 Type ID 사용을 고려하라.

:::

Type ID

Type ID는 CKB에서 단일톤(singleton) Cell 타입을 생성하는 데 사용되는 특수 스크립트로, 동일한 Type Script 해시를 가진 Live Cell이 한 시점에 단 하나만 존재할 수 있도록 한다.

이는 type hash를 사용할 때 특히 유용한데, 동일한 Type Script를 가진 경쟁 Cell의 생성을 방지하기 때문이다. 다시 말해, type hash가 항상 단일하고 유일한 Cell을 가리키도록 보장하여 모호성을 제거하고 공격 면을 줄인다.

Type ID는 CKB의 system Scripts 중 하나이며, 그 동작은 프로토콜 수준에서 강제된다.

Why This Design Matters

스크립트를 data hash 또는 type hash로 참조할 수 있도록 함으로써, CKB는 개발자와 사용자 모두에게 더 큰 유연성과 통제를 제공한다. 이 이중 접근은 다양한 사용 사례를 지원한다:

- **불변 참조**(data, data1, data2)는 매번 정확히 동일한 코드가 실행되도록 보장하여, 감사를 용이하게 하고 장기적 재현성을 촉진한다.
- **업그레이드 가능한 참조**(type)는 Type ID를 통해 일관된 식별자를 유지하면서 버그 수정이나 기능 추가와 같은 스크립트의 통제된 진화를 허용한다.

data, data1, data2를 hash_type으로 사용하는 스크립트는 CKB-VM의 버전도 함께 고정한다. 이는 향후 하드포크로 VM이 변경되더라도 일관된 동작을 보장한다. 반면, type hash를 사용하는 스크립트는 항상 최신 VM 버전에서 실행되어 네트워크와 함께 진화한다.

이처럼 스크립트 코드와 시스템 업그레이드를 분리하면 사용자는 더 많은 통제를 갖게 된다. 알려진 동작을 유지할지, 아니면 향후 하드포크를 통해 도입되는 개선을 따를지를 선택할 수 있다.

이 설계는 모든 참여자가 필요에 따라 신뢰성과 유연성의 균형을 맞출 수 있도록 한다. 개발자는 업그레이드 경로를 가진 스크립트를 게시할 수 있고, 더 강한 실행 보장을 원하는 사용자는 data hash를 사용하여 특정 버전에 고정할 수 있다.

프로토콜에 이러한 유연성을 내장함으로써, CKB는 다양한 요구를 수용하고 코드 사용에 대한 독립적 의사 결정을 지원한다.

Choosing Between Data and Type Hash

CKB는 스크립트 참조 방식에서 유연함을 허용한다. 다음 중에서 선택할 수 있다:

- **Data hash** – 불변성과 감사 가능성이 우선일 때 선호된다
- **Type hash** – 업그레이드 가능성이 우선일 때 선호된다

이 선택은 영구적이지 않으며, 필요하다면 나중에 변경할 수 있다.

예를 들어, 스크립트가 처음에 Type ID를 사용해 배포되었다면, 다음을 수행할 수 있다:

1. 스크립트 바이너리의 현재 버전을 포함하는 Cell을 찾는다
2. 그 data_hash를 계산한다
3. 대상 VM 버전에 따라 hash_type: data, data1, 또는 data2를 사용해 스크립트를 재배포한다

이렇게 하면 업그레이드 가능한 type 참조에서 불변의 data 참조로 전환하여, 실행을 현재 코드 버전에 고정할 수 있다.

script_args

script_args 는 스크립트의 입력으로서 CKB-VM 인스턴스에 전달되는 인자를 의미한다. CKB에서는 공개키 정보가 관례적으로 script_args 에, 서명 정보가 witnesses 에 저장되지만, 이는 필수가 아니다.

스크립트 그룹 실행 {#script-group-exe}

CKB는 입력들을 그들의 Lock Script 기준으로 그룹화하고, 그룹화된 스크립트를 한 번만 실행함으로써 Lock Script 실행을 간소화한다.

이 프로세스는 3단계로 구성된다:

1. **스크립트 그룹화(Script Grouping)**: CKB는 Lock Script별로 입력을 그룹화한다. 이 예시에서는 입력에 두 개의 Lock Script가 사용된다. 이들은 동일한 코드를 가리키지만, 서로 다른 args를 가진다. 여기서는 인덱스가 0과 2인 입력을 가진 **g1**에 집중한다. 이 스크립트와 입력 인덱스는 이후 3단계에서 사용된다.
2. **코드 위치 지정(Code Locating)**: CKB는 cell_deps 에서 코드를 식별한다. 이는 데이터 해시 Hs 를 가진 Cell로 해석되며, 그 데이터를 코드로 사용한다.
3. **실행(Execution)**: CKB는 스크립트의 코드를 로드하고 엔트리 함수부터 실행한다.

CKB syscall은 스크립트가 트랜잭션 데이터를 읽도록 도와준다. 이러한 syscall은 일반적으로 데이터를 어디에서 읽을지 지정하는 인자를 가진다. 예를 들어, 스크립트 자체를 읽으려면:

```
ckb_load_script(addr, len, offset)
```

첫 번째 witness를 로드하려면:

```
ckb_load_witness(addr, len, offset, 0, CKB_SOURCE_INPUT);
```

처음 세 인자 `addr`, `len`, `offset` 은 읽은 데이터를 어디에 저장하고 몇 바이트를 읽을지 제어한다. 단순화를 위해, 다음 단락에서는 다른 인자에 집중한다.

0 은 `inputs` 배열에 대한 인덱스이다.

`CKB_SOURCE_INPUT` 은 데이터 소스를 지정하며 트랜잭션 입력에서 읽는다. 이는 `witnesses` 를 읽을 때도 사용된다.

앞서 **스크립트 그룹화** 에서 저장한 입력 인덱스는 그룹을 위한 가상 witness 및 `inputs` 배열을 만드는 데 사용된다. 코드는 특수 소스 `CKB_SOURCE_GROUP_INPUT` 을 통해 가상 배열의 인덱스를 사용하여 입력 또는 witness를 읽을 수 있다. `CKB_SOURCE_GROUP_INPUT` 을 사용해 witness를 읽을 때는 지정한 입력과 동일한 위치에 있는 witness만 읽는다.

입력 데이터를 읽는 모든 syscall은 `CKB_SOURCE_GROUP_INPUT` 과 가상 `inputs` 배열의 인덱스를 사용할 수 있으며, 예를 들어 `ckb_load_cell_*` syscall 패밀리가 있다.

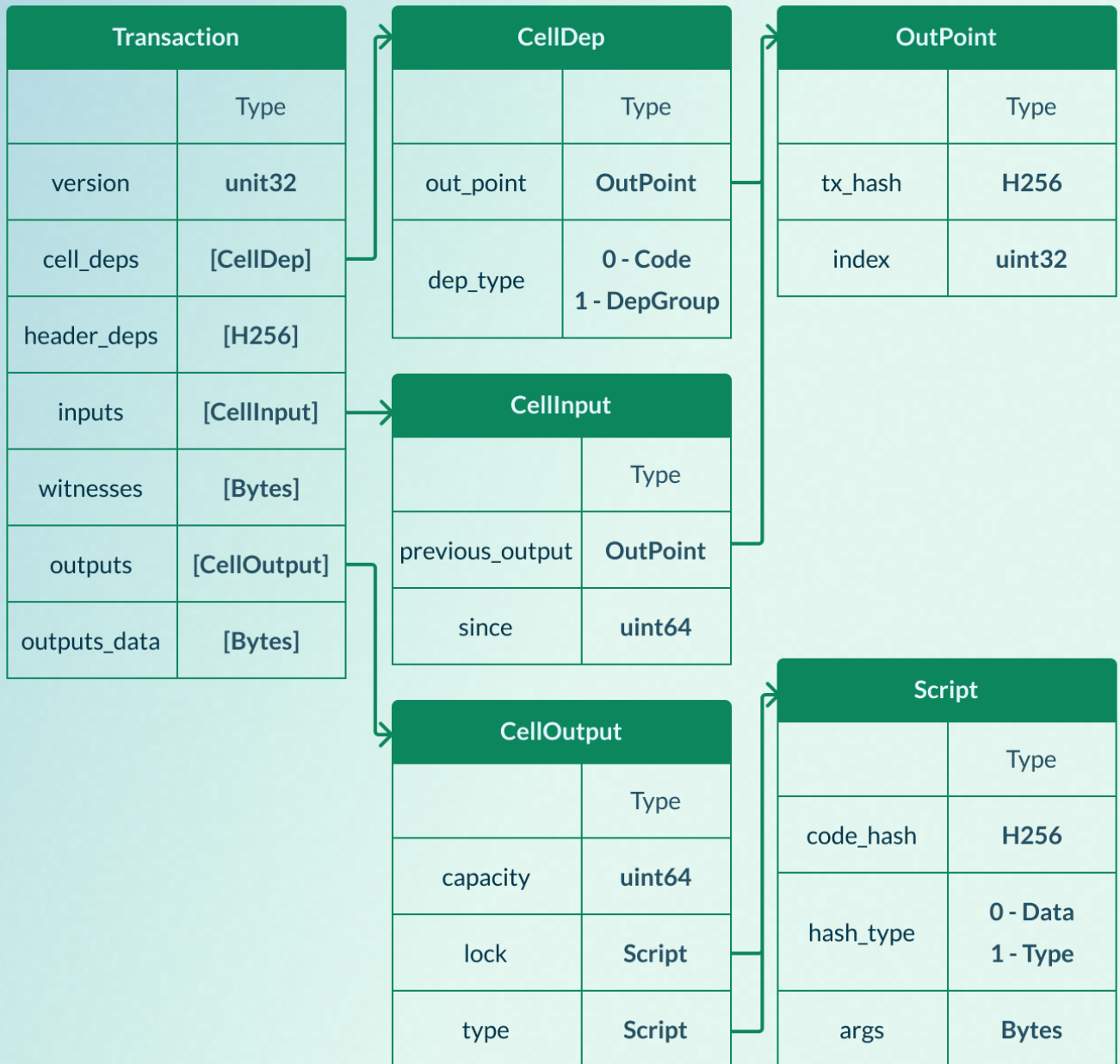
Type Script 실행은 다음을 제외하면 Lock Script 실행과 유사하다:

1. Type Script가 없는 Cell은 무시된다.
2. 입력과 출력 모두의 Type Script가 스크립트 그룹을 형성하는 데 사용된다.

`CKB_SOURCE_GROUP_INPUT` 과 유사하게, 스크립트 그룹에서 가상 `outputs` 배열의 인덱스를 사용하기 위한 특수 데이터 소스 `CKB_SOURCE_GROUP_OUTPUT` 이 존재한다.

트랜잭션

CKB에서 트랜잭션은 일부 Cell(이전 트랜잭션의 출력)을 파기하고 새로운 Cell을 생성한다.



필드와 설명

Name	Type	Description
version	Uint32	포크 발생 시 트랜잭션을 구분하는 트랜잭션 버전.
cell_deps	[CellDep]	이 트랜잭션이 의존하는 Cell을 가리키는 outpoint 목록. 자세한 내용은 cell_deps 참조.
header_deps	[H256(hash)]	이 트랜잭션이 의존하는 블록 헤더를 가리키는 H256 해시 목록. 자세한 내용은 header_deps 참조.
inputs	[CellInput]	참조된 Cell 입력의 목록. 자세한 내용은 inputs 참조.
witnesses	[Bytes]	해당 Lock Script의 성공적 실행을 위해 트랜잭션 생성자가 제공하는 값. 자세한 내용은 witnesses 참조.
outputs	[Cells]	출력으로 사용되는 Cell의 목록으로, 새로 생성된 Cell로 볼 수 있다. 자세한 내용은 outputs 참조.
outputs_data	[Bytes]	각 출력 Cell의 데이터 목록. 자세한 내용은 outputs_data 참조.

예시

▶ 전체 트랜잭션 보기

```
{
  "version": "0x0",
  "cell_deps": [
    {
      "out_point": {
        "tx_hash":
"0xbd864a269201d7052d4eb3f753f49f7c68b8edc386afc8bb6ef3e15a05facca2",
        "index": "0x0"
      },
      "dep_type": "dep_group"
    }
  ],
  "header_deps": [
    "0xaa1124da6a230435298d83a12dd6c13f7d58caf7853f39cea8aad992ef88a422"
  ],
  "inputs": [
    {
      "previous_output": {
```

```

        "tx_hash":
"0x8389eba3ae414fb6a3019aa47583e9be36d096c55ab2e00ec49bdb012c24844d",
        "index": "0x1"
    },
    "since": "0x0"
}
],
"outputs": [
{
    "capacity": "0x746a528800",
    "lock": {
        "code_hash":
"0x9bd7e06f3ecf4be0f2fcd2188b23f1b9fcc88e5d4b65a8637b17723bbda3cce8",
        "args": "0x56008385085341a6ed68decfab3ba1f3eea7b68",
        "hash_type": "type"
    },
    "type": null
},
{
    "capacity": "0x1561d9307e88",
    "lock": {
        "code_hash":
"0x9bd7e06f3ecf4be0f2fcd2188b23f1b9fcc88e5d4b65a8637b17723bbda3cce8",
        "args": "0x886d23a7858f12ebf924baaacd774a5e2cf81132",
        "hash_type": "type"
    },
    "type": null
}
],
"outputs_data": [
    "0x",
    "0x"
],
"witnesses":
["0x55000000100000005500000055000000410000004a975e08ff99fa0001

42ff3b86a836b43884b5b46f91b149f7cc5300e8607e633b7a29c94dc01c6616a12f62e
74a1
415f57fcc5a00e41ac2d7034e90edf4fdf800"
]
}

```

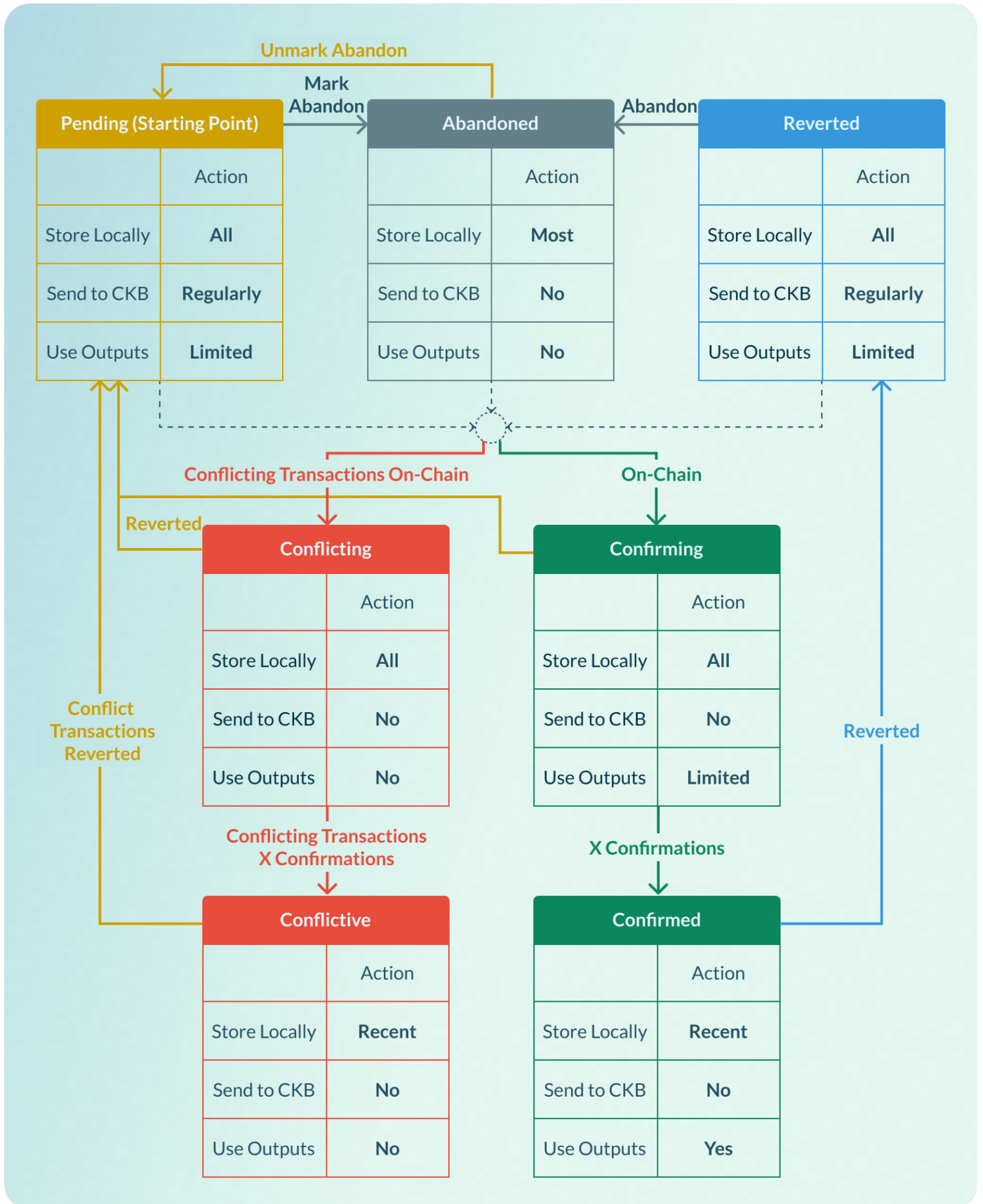
트랜잭션 상태들 {#transaction-states}

트랜잭션 라이프사이클은 생성 이후 시작되어 최종적으로 확정되거나 확정된 트랜잭션과 충돌할 때 종료된다. 이 라이프사이클 동안 트랜잭션은 다음 상태 중 하나에 있어야 한다:

- Pending
- Confirming
- Confirmed
- Conflicting
- Conflictive
- Reverted
- Abandoned

이 목록은 알려지지 않은 Cell에 의존하는 orphan transaction 을 제외하는데, 트랜잭션 생성기는 일반적으로 이를 무시하기 때문이다.

아래 섹션에서 설명하듯이 서로 다른 상태에는 서로 다른 전략이 요구된다.



Pending 트랜잭션

새로 생성된 트랜잭션은 **Pending** 상태로 시작한다.

생성기는 Pending 트랜잭션을 로컬에 저장하고 CKB 노드로 정기적으로 전송해야 한다. 이는 CKB 노드가 풀에서 트랜잭션을 드롭할 수 있기 때문에 중요하다.

트랜잭션이 지연되어 fee bumping 이 필요하지 않은 한, 생성기가 Pending 트랜잭션의 출력 Cell을 사용하는 것은 권장되지 않는다. Pending 트랜잭션의 체이닝이 불가피하다면, 체인의 길이를 제한하라.

정식 체인의 블록에 트랜잭션이 나타나면, 해당 트랜잭션은 **Confirming** 상태로 전환된다.

Pending 트랜잭션 또는 그 선조 트랜잭션¹ 중 어느 하나라도 정식 체인의 트랜잭션과 입력 Cell이 충돌한다면, 해당 트랜잭션은 conflicting이 된다.

사용자 또는 앱은 멈춰버린 트랜잭션을 포기하기로 결정하면 Pending 트랜잭션을 **Abandoned** 로 표시할 수 있다.

Confirming 트랜잭션

Confirming 상태의 트랜잭션은 체인에 있지만 아직 **Confirmed** 로 자격을 갖추지 못했다.

생성기는 Confirming 트랜잭션을 로컬에 저장해야 하지만, 이를 CKB 노드로 정기적으로 전송할 필요는 없다.

Pending 트랜잭션과 마찬가지로, Confirming 트랜잭션의 출력 Cell을 사용하는 것은 권장되지 않는다.

트랜잭션을 포함하는 블록은 한 번의 확인을 제공한다. 체인의 각 하위 블록은 추가 확인을 제공한다. 이론상 어떤 블록도 되돌릴 수 있지만, 확인 횟수가 특정 수 X 를 넘으면 트랜잭션이 다시는 되돌려지지 않을 것이라고 가정해도 안전하다. Confirming 트랜잭션은 적어도 X 회의 확인을 받으면 **Confirmed** 가 된다. X 는 여러 요인에 의존한다. 생성기는 X 값을 결정하고 정기적으로 조정해야 한다.

체인 리오그가 트랜잭션을 포함한 블록을 되돌리면, 해당 트랜잭션은 **Pending** 으로 되돌아간다.

Confirmed 트랜잭션

Confirmed 상태의 트랜잭션은 체인에 있으며 최소 X 회의 확인을 받았다.

생성기는 Confirmed 트랜잭션을 로컬 저장소에서 제거하거나, 최근 것만 기록 보관 목적으로 유지할 수 있다.

생성기는 Confirmed 트랜잭션의 출력 Cell을 자유롭게 사용할 수 있다.

체인 리오그가 트랜잭션을 포함한 블록을 되돌리면, 해당 트랜잭션은 **Reverted** 가 된다.

Conflicting 트랜잭션

체인의 어떤 트랜잭션과 충돌하면 해당 트랜잭션은 **Conflicting** 상태가 된다. 두 개의 서로 다른 트랜잭션은 다음과 같을 때 충돌한다:

- 서로 충돌하는 입력 Cell을 가진다;

- 한 트랜잭션이 다른 트랜잭션의 어느 선조와 충돌하는 입력 Cell을 가진다;
- 이들이 충돌하는 트랜잭션 쌍의 하위 트랜잭션이다.

생성기는 이러한 Conflicting 트랜잭션을 로컬에 저장해야 한다. 이들이 다시 **Pending** 이 될 가능성이 있기 때문이다.

생성기는 Conflicting 트랜잭션의 출력 Cell을 사용해서는 안 된다.

Conflicting 트랜잭션을 포함하는 모든 블록이 되돌려지면, 해당 Conflicting 트랜잭션은 Pending으로 되돌아간다.

Conflicting 트랜잭션 중 하나라도 X회의 확인을 받으면, 나머지 Conflicting 트랜잭션은 **Conflicted**가 된다.

Conflictive 트랜잭션

온체인 트랜잭션과 충돌하고, 그 온체인 트랜잭션이 최소 X회의 확인을 받은 경우, 해당 트랜잭션은 **Conflictive** 상태가 된다.

생성기는 Conflictive 트랜잭션을 로컬 저장소에서 제거하거나, 최근 것만 참고용으로 보관할 수 있다.

생성기는 Conflictive 트랜잭션의 출력 Cell을 사용해서는 안 된다.

Conflicting 트랜잭션을 포함하는 모든 블록이 되돌려지면, Conflictive 트랜잭션은 **Pending** 으로 되돌아간다.

Reverted 트랜잭션

Reverted 상태는 **Pending** 의 별칭으로, 해당 트랜잭션이 한때 확정되었다가 되돌려졌음을 시사한다.

생성기는 Reverted 트랜잭션을 저장하고 정기적으로 전송해야 한다. 상태 변환은 Pending 트랜잭션과 동일하다.

Abandoned

생성기는 사용자가 커밋을 포기하기로 했을 때, Pending 또는 Reverted 트랜잭션을 **Abandoned** 로 표시할 수 있도록 허용해야 한다. 그러나 타이밍 차로 인해, Abandoned로 표시된 트랜잭션이 이미 노드에 제출되어 있을 수도 있다.

생성기는 최근 Abandoned 트랜잭션을 충분한 기간 동안 저장해야 한다. **Abandoned** 로 표시된 트랜잭션을 삭제하지 않도록 노력하라. 해당 트랜잭션이 이미 **Pending** 상태일 수 있기 때문이다.²

사용자는 Abandoned 트랜잭션을 **Pending** 으로 다시 표시할 수 있다.

트랜잭션은 Pending 및 Reverted 트랜잭션과 마찬가지로 **Abandoned** 에서 **Confirming** 또는 **Conflicting** 으로 전이될 수 있다.

Cell_deps

cell_deps 는 트랜잭션의 스크립트가 참조된 Live Cell에 접근할 수 있도록 한다. 이들 Cell은 읽기 전용이며, 실행 시점에 Live 상태여야 한다. cell_deps 배열의 각 항목은 CellDep 타입이다.

CellDep의 구조

Name	Type	Description
out_point	OutPoint	트랜잭션이 의존하는 특정 Live Cell을 가리킨다. 이 Cell은 실행 가능한 코드나 데이터를 포함할 수 있다.
dep_type	code or dep_group	의존성을 CKB가 어떻게 해석해야 하는지 알려준다.

out_point

특정 출력 Cell을 식별한다. 타입은 OutPoint 이며, 트랜잭션 hash 와 출력 index 로 구성된다.

Name	Type	Description
hash	H256 (hash)	해당 Cell이 속한 트랜잭션의 해시.
index	Uint32	그 트랜잭션의 출력 목록에서의 Cell 인덱스.

dep_type

의존성 Cell을 어떻게 사용해야 하는지 정의한다:

code

out_point 의 Cell이 직접 로드된다. 그 내용(보통 코드)은 스크립트 실행 중 CKB-VM에서 사용할 수 있다.

dep_group

out_point 의 Cell이 dep 그룹이다. 이는 OutPoint 목록을 포함한다. CKB는 이 목록을 읽고, 나열된 각 Cell을 개별 cell_deps 인 것처럼 로드한다. 이를 통해 dep_group 은 단일 참조로 여러 의존성을 포함하는 간결한 방법이 된다.

header_deps

header_deps 는 스크립트가 실행 중 접근할 수 있는 블록 헤더 해시를 나열하며, 트랜잭션 결정성을 보장하기 위해 특정 제한이 적용된다.

inputs

inputs 필드는 이 트랜잭션이 소비하는 Cell들을 나열한다. inputs 의 각 항목은 CellInput 타입이며, 이전에 생성된 출력 Cell을 참조한다.

Structure of CellInput

Name	Type	Description
previous_output	OutPoint	이전 트랜잭션의 특정 출력 Cell을 참조한다.
since	Uint64	이 입력을 언제 사용할 수 있는지를 제한하는 조건이다. 자세한 내용은 RFC: Transaction Since Precondition 을 참조하라.

previous_output

입력으로 사용되는 Cell에 대한 참조다. 이는 정확한 출력 Cell을 식별하는 트랜잭션 hash 와 출력 index 로 구성된다.

Name	Type	Description
hash	H256 (hash)	해당 Cell이 속한 트랜잭션의 해시.
index	Uint32	그 트랜잭션의 출력 목록에서의 Cell 인덱스.

since

since 필드는 트랜잭션의 각 입력에 첨부되는 선택적 조건이다. 이는 입력이 유효해지기 전에 충족되어야 하는 시간 제약을 지정한다. CKB 노드는 현재 체인 상태가 since 요구 조건을 충족하거나 초과하는 경우에만 해당 입력(따라서 전체 트랜잭션)을 유효하다고 간주한다.

이 메커니즘은 시간 잠금된 입력을 가능하게 하여, 지연 지급이나 다자간 조정 같은 사용 사례를 지원한다. 트랜잭션은 여러 당사자의 입력을 포함할 수 있다. 각 입력은 자체 `since` 조건을 가질 수 있어, 각 당사자가 자신의 트랜잭션 부분이 언제 유효해지는지 제어할 수 있다. 그러나 단일 입력의 `since` 조건이 미충족일 경우, 전체 트랜잭션은 노드에 의해 거부된다.

`since` 값은 다음 지표 중 하나로 표현될 수 있다:

- 블록 번호
- 에포크 번호(분수 포함)
- 타임스탬프

인코딩 및 평가 규칙에 대한 기술적 세부사항은 [RFC0017: Transaction Since Precondition](#) 을 참조하라.

witnesses

`witnesses` 는 해당 스크립트의 성공적 실행을 가능하게 하기 위해 트랜잭션 생성자가 제공한다. 예를 들어, 서명 검증 Lock Script가 통과하도록 서명이 포함될 수 있다.

CKB 관례상, 공개키 정보는 스크립트의 `args` 에 저장하고, 서명 정보는 `witnesses` 에 저장한다.

`witnesses` 는 바이트 벡터다. 트랜잭션에 필요한 임의의 증명 데이터를 `witnesses` 필드의 바이트 벡터로 직렬화해 넣을 수 있다. 직렬화는 [Molecule](#) 이나 그 밖의 임의의 커스텀 직렬화 방법을 사용할 수 있다.

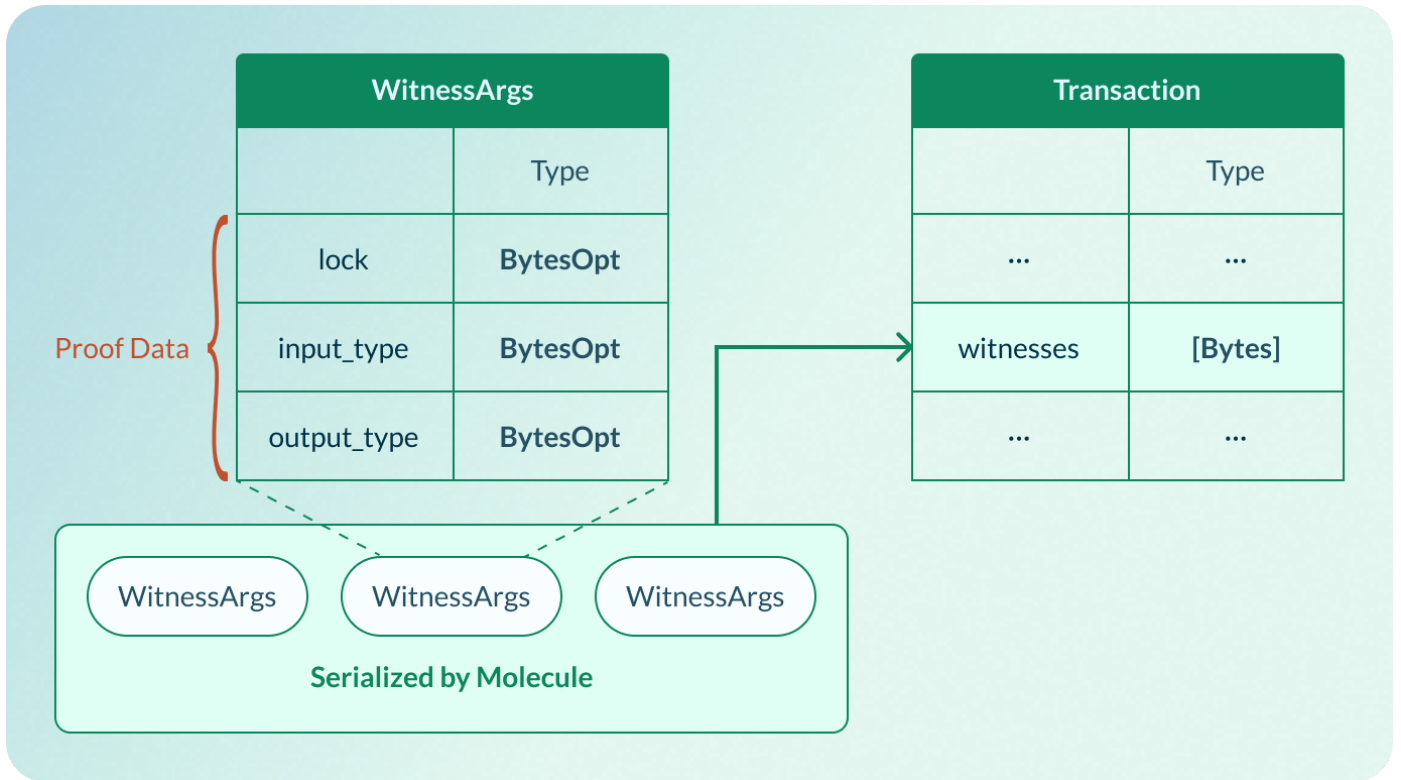
Conventional Usage

`witnesses` 는 CKB 트랜잭션에서 매우 유연하여, 스크립트가 트랜잭션 검증을 위해 다양한 방식으로 `witnesses` 데이터를 읽을 수 있다. 스크립트 간 호환성과 상호 간섭 방지를 위해 특정 관례가 마련되어 있다. 현재까지 두 가지가 있다: **WitnessArgs** 와 **CoBuild**.

WitnessArgs

WitnessArgs 는 CKB 커뮤니티에서 적극적으로 사용되는 관례다. 그 구조는 세 개의 필드를 포함한다:

```
table WitnessArgs {
  lock:                Bytes0pt,           // Lock args
  input_type:          Bytes0pt,           // Type args for input
  output_type:         Bytes0pt,           // Type args for output
}
```



이 구조는 Molecule 에서 정의된 table 이다. 세 필드는 모두 선택적(0pt)이며, 계약의 요구사항에 따라 선택적으로 사용될 수 있다.

CKB에서는 서로 다른 스크립트가 서로 다른 시점에 실행된다:

- Lock Script 는 입력 Cell(소비될 때)에서만 실행된다.
- Type Script 는 입력과 출력 Cell(소비·생성될 때) 모두에서 실행된다.

이 때문에 WitnessArgs 구조에는 세 개의 선택적 필드가 있다:

- `lock`: Lock Script가 사용하는 데이터
- `input_type`: 입력 검증 시 Type Script가 사용하는 데이터
- `output_type`: 출력 검증 시 Type Script가 사용하는 데이터

이 관례에 따라, 모든 스크립트는 Witness 데이터를 WitnessArgs로부터 읽는다.

`witnesses` 필드에서 WitnessArgs 항목의 **순서** 도 중요하다. 이는 스크립트 그룹화 에 의해 생성된 가상 배열의 인덱스에 대응하기 때문이다.

Example and Test

- Contract: Group Exec
- Unit Test

이 메커니즘을 설명하기 위해, 다음과 같은 작업을 수행하는 스크립트를 제시한다:

1. 자신의 Args 내용을 출력한다(이 예에서는 UTF-8 문자열).
2. `load_script_hash` 를 사용해 자신의 ScriptHash를 얻는다.
3. `load_cell_lock_hash` 와 `load_cell_type_hash` 를 사용해 다른 Cell의 Lock/Type 스크립트를 검사하고 그 인덱스를 출력한다.
4. `load_witness_args` 를 사용해 모든 GroupInput 및 GroupOutput 데이터를 얻어 출력한다(UTF-8 문자열로 저장된 Witness 값).

테스트 트랜잭션은 다음과 같이 구성된다:

```
Inputs:
  0:
    Lock:
      Code Hash: lock_script_1
    Type:
      Code Hash: type_script_1
  1:
    Lock:
      Code Hash: lock_script_1
    Type:
      Code Hash: type_script_2
Outputs:
  0:
    Lock:
      Code Hash: lock_script_1
    Type:
      Code Hash: type_script_2
  1:
    Lock:
      Code Hash: lock_script_1
    Type:
      Code Hash: type_script_1
```

(이 트랜잭션은 두 개의 TypeScripts 를 교환할 뿐이다. 이는 단지 데모를 위한 것이며, 실제로 특별한 의미는 없다.)

형성된 그룹은 다음과 같다:

- **lock_script_1**: inputs[0], inputs[1]
- **type_script_1**: inputs[0], outputs[1]
- **type_script_2**: inputs[1], outputs[0]

CKB 트랜잭션에서 **Witnesses** 는 **Inputs** 및 **Outputs** 와 독립적이다. 그러나 각 witness의 인덱스는 그것이 속한 스크립트 그룹의 인덱스에 대응한다.

예를 들어, `type_script_1` 이 `inputs[0]` 과 `outputs[1]` 로 그룹화되었다면, 이 그룹의 Witness를 로드할 때 `witnesses[0]` 과 `witnesses[1]` 이 모두 접근된다. (여기서는 반드시 해당 **GroupInput** 과 **GroupOutput** 을 사용해야 한다 — 이들의 Witness는 서로 다를 수도 있고 같을 수도 있다.)

그룹화 이후, 스크립트는 다음과 같이 Witness를 로드한다:

- **lock_script_1**: `witnesses[0]`, `witnesses[1]` (GroupInput을 통해)
- **type_script_1**: `witnesses[0]` (GroupInput을 통해), `witnesses[1]` (GroupOutput을 통해)
- **type_script_2**: `witnesses[1]` (GroupInput을 통해), `witnesses[0]` (GroupOutput을 통해)

이로부터 복수의 스크립트가 동일한 Witness를 사용하게 될 수 있음을 알 수 있다. 이러한 상황은 실제 트랜잭션에서 흔하다. 잠재적 충돌을 해결하기 위해, CKB는 표준 관례인 **WitnessArgs** 를 도입한다.

WitnessArgs의 세 필드는 위의 세 가지 가능한 위치에 정확히 매핑된다. 예를 들어, `lock_script_1` 이 `witnesses[0]` 을 얻을 때는 `lock` 필드만 추출할 수 있으며, 마찬가지로 `type_script_1` 은 `witnesses[0]` 에서 `input_type` 필드를 가져갈 수 있다.

Raw Test Output (full logs)

```
---- tests::test_group_exec stdout ----
[contract debug] lock 1
[contract debug] Inputs Lock:  [00, 01]
[contract debug] Inputs Type:  None
[contract debug] Outputs Type: None
[contract debug] Index: 0, Lock Witnes:  Witness 0, Lock
[contract debug] Index: 1, Lock Witnes:  Witness 1, Lock
[contract debug] type 2
[contract debug] Inputs Lock:  None
[contract debug] Inputs Type:  [01]
[contract debug] Outputs Type: [00]
[contract debug] Index: 0, InputType Witnes:  Witness 1, Input Type
[contract debug] Index: 0, OutputType Witnes:  Witness 1, Output Type
[contract debug] Index: 0, InputType Witnes:  Witness 0, Input Type
[contract debug] Index: 0, OutputType Witnes:  Witness 0, Output Type
[contract debug] type 1
[contract debug] Inputs Lock:  None
[contract debug] Inputs Type:  [00]
[contract debug] Outputs Type: [01]
```

```
[contract debug] Index: 0, InputType Witnes: Witness 0, Input Type
[contract debug] Index: 0, OutputType Witnes: Witness 0, Output Type
[contract debug] Index: 0, InputType Witnes: Witness 1, Input Type
[contract debug] Index: 0, OutputType Witnes: Witness 1, Output Type
consume cycles: 245118
```

이 테스트에서는 데모 목적을 위해 세 필드(lock, input_type, output_type)가 모두 채워져 있다. Lock이든 Type이든, 각 스크립트 그룹은 자신의 첫 번째 입력 인덱스에서 엔트리 Witness를 가져온다. 여러 그룹이 동일한 입력 인덱스를 공유하면, 동일한 Witness를 “재사용” 하는 것처럼 보일 수 있지만, 각자 관련 필드만 읽으므로 충돌을 피한다.

CoBuild

CoBuild는 여러 당사자가 오프체인 절차를 통해 협력적으로 CKB 트랜잭션을 생성하는 방법을 설명하는 프로토콜이다. 또한 기존 WitnessArgs를 대체하는, 조합 가능하고 자동화 친화적인 트랜잭션 witness 레이아웃 표준을 만드는 것을 목표로 한다.

그러나 CoBuild 프로토콜은 아직 초기 단계에 있으므로, 현재로서는 널리 사용되지 않으며 표준도 아직 확정되지 않았다. CoBuild에 대한 더 많은 내용은 [제안서](#)에서 확인할 수 있다.

트랜잭션 구조에 대한 자세한 설명은 [RFC0022: CKB Transaction Structure](#)를 참조하라.

outputs

트랜잭션은 inputs에 나열된 Cell을 파기하고 outputs에 나열된 새로운 Cell을 생성한다. 이러한 출력 Cell은 표준 [Cell 구조](#)와 동일한 구조를 따르며, 이후 트랜잭션에서 입력으로 사용할 수 있다.

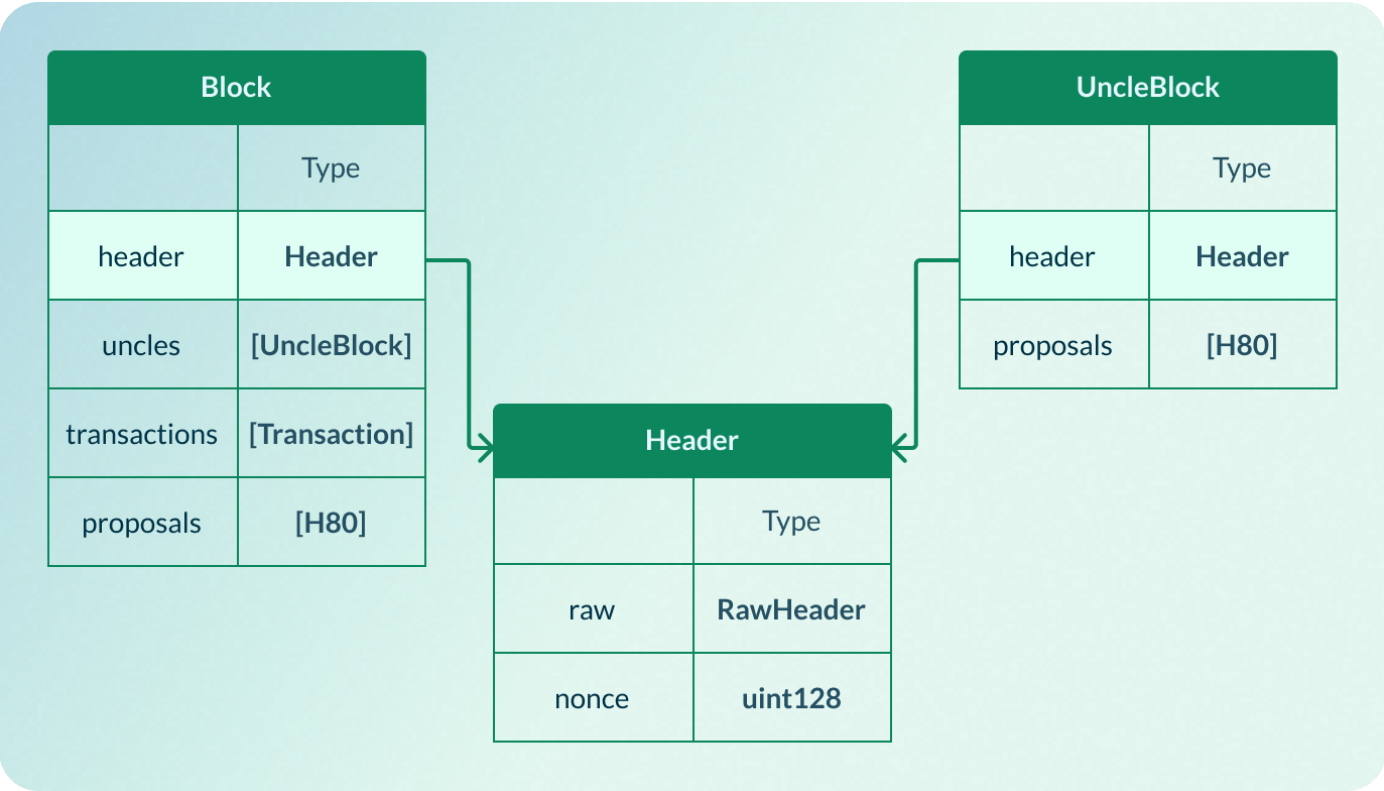
outputs_data

outputs_data는 각 출력 Cell의 데이터를 담은 목록이다. outputs의 i번째 Cell에 대한 데이터는 outputs_data의 i번째 항목으로 저장된다.

outputs_data를 outputs와 분리하면 [CKB-VM](#)의 스크립트 실행이 단순해지고, 프로토콜 수준의 향후 최적화를 가능하게 한다.

헤더 {#header}

header 필드는 Block 및 UncleBlock 구조의 일부이다. 이 필드는 블록의 내용을 요약하고 보호하는 메타데이터를 포함하며, 합의와 체인 검증에서 중요한 역할을 한다.

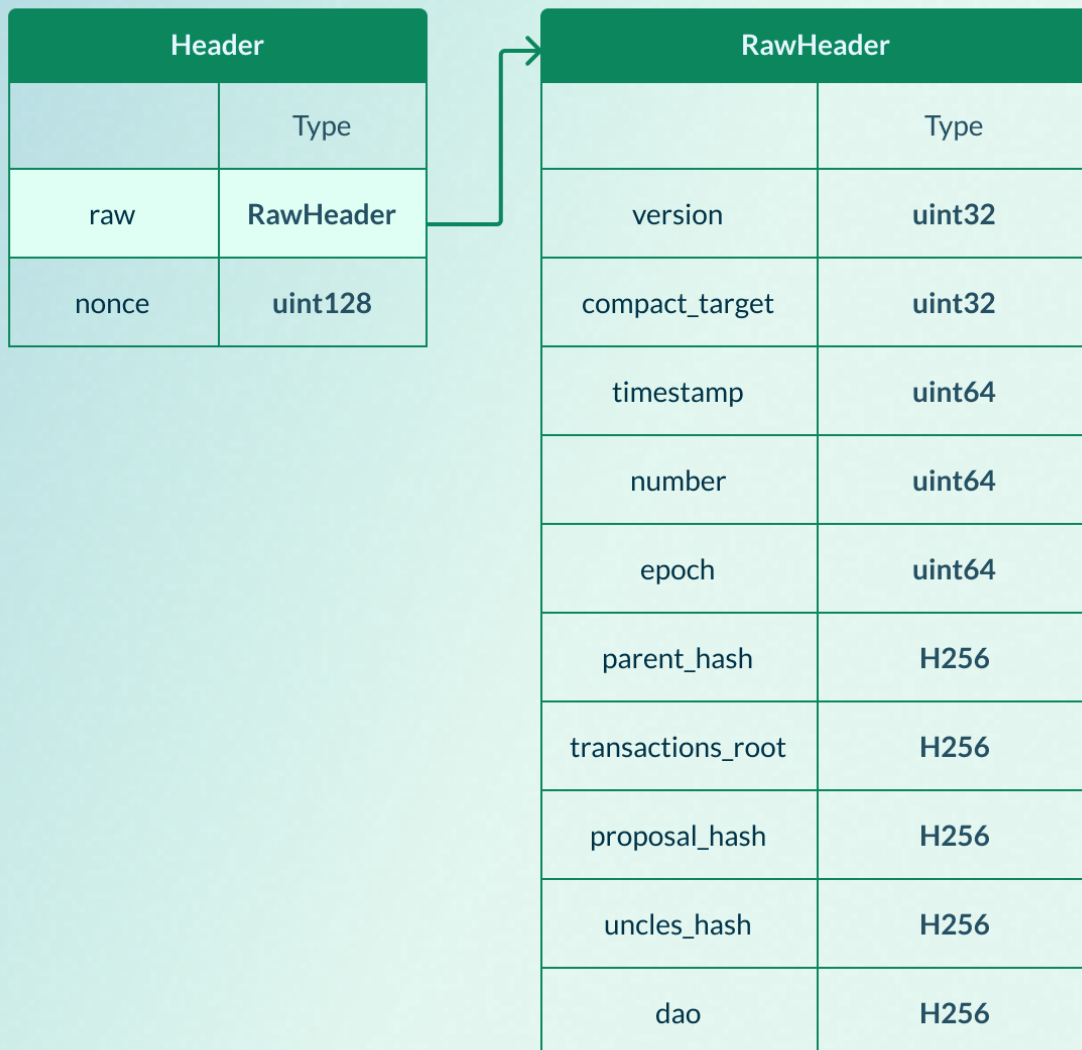


Header의 구조

Name	Type	Description
raw	RawHeader	블록 헤더의 페이로드.
nonce	uint128	PoW 퍼즐의 해답. <u>비트코인 nonce</u> 와 유사하다.

RawHeader

RawHeader 는 블록 헤더의 페이로드이다.



Name	Type	Description
version	Uint32	포크 이후 발생할 수 있는 잠재적 호환성 문제를 해결하기 위해 사용되는 블록의 버전
compact_target	Uint32	compact target 형식으로 표현된 PoW 난이도
timestamp	Uint64	밀리초 단위의 유닉스 시간 타임스탬프
number	Uint64	블록 높이를 나타낸다
epoch	Uint64	현재 에포크의 정보. number 가 현재 에포크 번호를, index 가 현재 에포크에서의 블록 인덱스(0부터 시작)를, length 가 현재 에포크의 길이를 나타낸다고 가정하면, 값은 반드시 $(\text{number} \& 0\text{FFFFFF}) \setminus ((\text{index} \& 0\text{FFFF}) \ll 24) \setminus ((\text{length} \& 0\text{FFFF}) \ll 40)$ 이어야 한다
parent_hash	H256 (hash)	부모 블록의 해시
transactions_root	H256 (hash)	트랜잭션 해시의 CBMT(Complete Binary Merkle Tree) 루트와 트랜잭션 witness 해시의 CBMT 루트를 연결(concatenate)하여 얻은 해시
proposals_hash	H256 (hash)	연결된 proposal ID들의 해시. 제안이 없으면 기본값은 모두 0이다.
uncles_hash	H256 (hash)	영클 블록 헤더 해시들을 연결한 값의 해시. 제안이 없으면 기본값은 모두 0이다.
dao	Bytes	DAO 관련 데이터를 포함한다. 자세한 내용은 RFC0023: Deposit and Withdraw in Nervos DAO 를 참조하라.
nonce	Uint128	PoW 퍼즐의 해답. 비트코인 nonce 와 유사하다.

자세한 설명은 [RFC-0027의 RawHeader](#) 를 참조하라.

Header 검증 프로세스

다음 스니펫은 Nervos CKB 블록체인에서 블록 헤더에 대한 PoW를 검증하는 과정을 설명한다:

1. 블록의 원시 데이터를 직렬화하고 해싱한다.

2. 해시를 nonce 와 연결한다.
3. 연결된 결과를 Eaglesong 알고리즘에 통과시킨다.
4. (옵션) 테스트넷의 경우 재해싱한다.
5. 최종 출력을 정수로 변환하고 요구되는 난이도 타겟을 만족하는지 확인한다.

```
pow_hash := ckbhash(molecule_serialize(raw))
pow_message := pow_hash || to_le(nounce)
pow_output := eaglesong(pow_message)
// for Testnet, there is another round of hash
// pow_output = ckbhash(pow_output)

from_be(pow_output) <= compact_to_target(raw.compact_target)
```

의사코드에서 사용된 함수들

- :=: 대입
- ||: 바이너리 연결(concatenation)
- ckbhash: CKB 특화 구성의 [Blake2b](#) 해시
- to_le: 부호 없는 정수를 리틀엔디언 바이트로 변환. 바이트 수는 정수 폭과 동일하다.
- from_be: 빅엔디언으로 인코딩된 바이트를 부호 없는 정수로 변환
- molecule_serialize: 스키마를 사용해 구조를 바이너리로 직렬화
- eaglesong: CKB의 작업증명 합의 알고리즘. [RFC0010: Eaglesong](#) 참조
- compact_to_target: raw.compact_target 로 인코딩된 난이도 타겟의 compact 형식 으로부터 실제 타겟을 복원

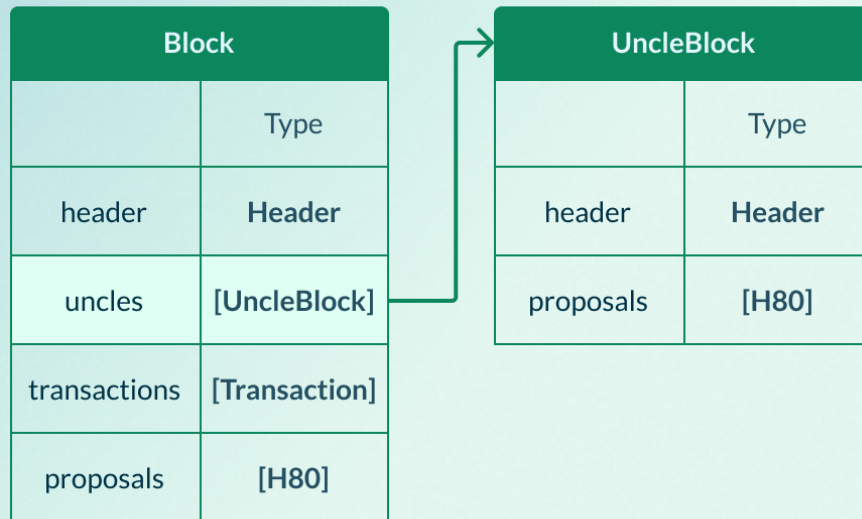
Header 해시 도출 및 사용

header 는 해시되어 고유한 header_hash 를 생성한다. 이 header_hash 는 블록을 참조하는 데 사용된다.

```
header_hash := ckb_hash(molecule_serialize(header))
```

uncles

uncles 는 이 블록이 참조하는 [영클 블록](#) 의 목록이다. uncles 의 각 항목은 UncleBlock 타입이며, 영클의 헤더와 proposals 를 포함하지만 트랜잭션은 포함하지 않는다.



UncleBlock의 구조

Name	Type	Description
header	Header	엥클 블록의 블록 헤더. 자세한 내용은 header 참조.
proposals	[H80]	이 엥클 블록이 제안한 짧은 트랜잭션 ID들의 목록. 자세한 내용은 proposals 참조.

transactions

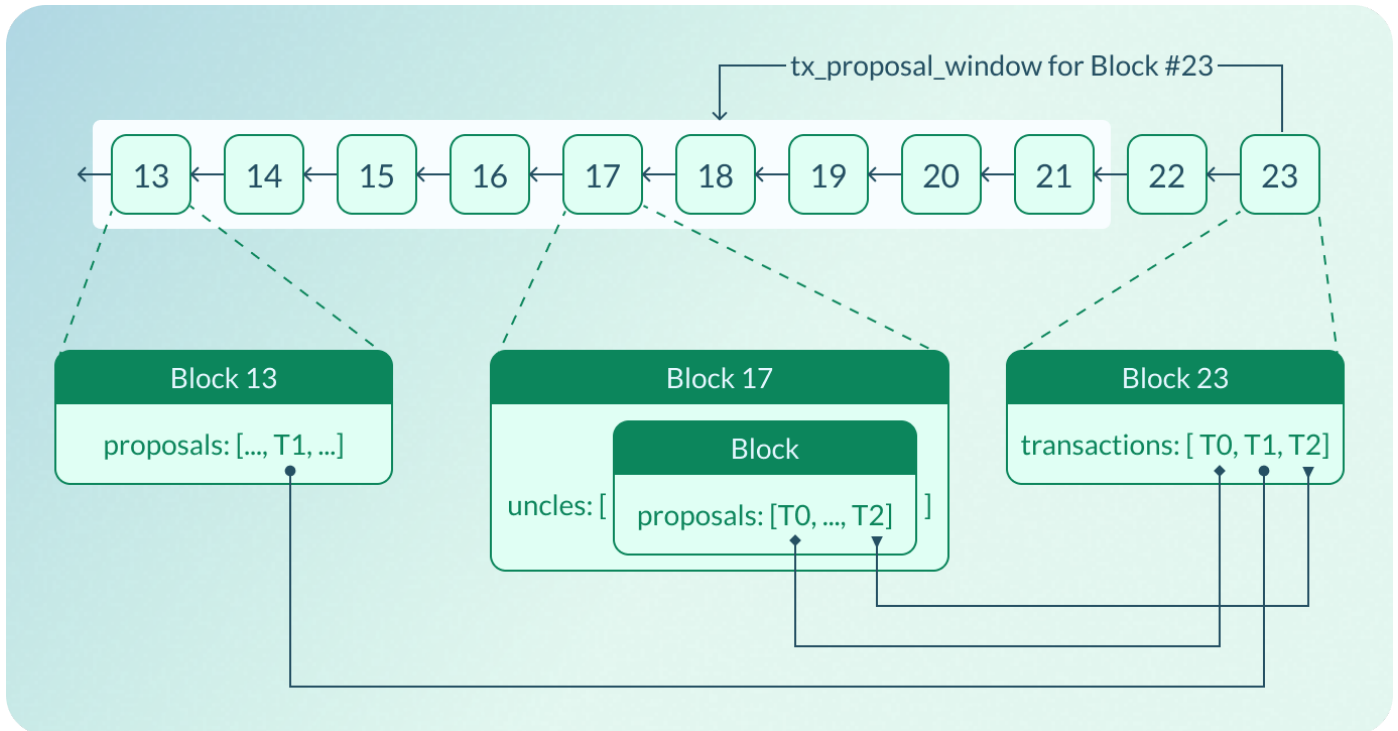
`transactions` 는 블록에 포함된 커밋된 트랜잭션들의 목록이다. 이 배열의 각 원소는 [트랜잭션 구조](#) 를 따른다.

첫 번째 트랜잭션은 반드시 cellbase 트랜잭션이어야 하며 — 이는 채굴자가 블록 보상을 청구하기 위해 만든 특수 트랜잭션이다.

proposals

`proposals` 는 블록 또는 해당 엥클 블록 내에서 제안된 트랜잭션을 나타내는 16진수 인코딩된 짧은 트랜잭션 ID들의 목록을 의미한다. 트랜잭션 proposal ID는 트랜잭션 해시의 처음 10바이트이다. CKB에서는 트랜잭션이 블록체인에 커밋되기 전에 반드시 proposal ID가 제안되어야 한다.

- 트랜잭션의 proposal ID가 블록 또는 그 블록의 영클의 proposals 필드에 나열되어 있으면, 해당 트랜잭션은 **proposed** 로 간주된다.
- 트랜잭션이 블록의 transactions 필드에 나타나면, 해당 트랜잭션은 **committed** 로 간주된다.



프로토콜 파라미터 `close` 와 `far` 는 트랜잭션의 `proposal` 과 커밋 사이의 온체인 거리의 최솟값과 최댓값을 지정한다. 블록 `c` 에 커밋되는 비-cellbase 트랜잭션은 블록 `p` 에서 제안되어야 하며, 여기서

$$\text{close} \leq c - p \leq \text{far}$$

CKB 메인넷에서는 `close` 가 2이고 `far` 가 10이다. 따라서:

$$2 \leq c - p \leq 10$$

위와 같이, 트랜잭션이 블록 23에서 커밋될 때, 해당 트랜잭션은 블록 13부터 블록 21까지의 `tx_proposal_window` 내에서 제안되어야 한다.

자산 & 토큰 표준

Nervos의 자산 & 토큰 표준

Nervos CKB는 비트코인의 UTXO 모델을 발전시킨 [Cell 모델](#)을 사용하며, 모든 디지털 자산(대체/비대체 토큰, 컬렉티블)은 사용자에게 독점적으로 소유되는 불변의 Cell로 표현된다. 각 Cell에는 [스크립트](#)가 집행하는 소유 규칙이 포함되어 있어, 자산이 트랜잭션 중 프로토콜 규칙을 준수하면서도 소유자의 단독 통제하에 남도록 보장한다.

온체인 자산으로 간주되는 것은 무엇인가?

이 프레임워크에서, Cell로 표현되며—사용자가 소유, 전송, 상환할 수 있는—모든 토큰 또는 디지털 오브젝트는 진정한 온체인 디지털 자산으로 간주된다. 이러한 자산에는 다음이 포함된다:

- 네이티브 토큰(CKBytes)
- 사용자 정의 토큰(sUDT, xUDT)
- 온체인 디지털 오브젝트(Spore 프로토콜을 통한 DOB)
- 비트코인과 Nervos CKB를 연결하는 RGB++ 프로토콜로 검증된 자산

토큰 표준 개요

Asset	Type	Equivalent	Description	Protocol
CKByte (CKB)	Native	/	네이티브 유틸리티 및 거버넌스 토큰	Native
xUDT Tokens	User-defined	ERC-20	광범위한 사용자 정의 가능 대체형 토큰	xUDT
Spore	Digital Object (DOB)	ERC-721, Ordinals	상환 가능한 내재적 가치, 진정한 온체인 소유권, 프라이버시, 멀티 콘텐츠를 지원하는 고유 디지털 오브젝트	Spore Protocol
RGB++	비트코인과 등형 바인딩	/	등형 UTXO 바인딩을 통해 브리지 없이 비트코인과 Nervos CKB 간 자산 발행 및 상호운용성을 가능하게 함	RGB++ Protocol

토큰노믹스

CKByte 토큰노믹스

CKByte는 Nervos의 네이티브 토큰으로, 세 가지 유형의 수수료를 총당한다: 사이클(연산), 트랜잭션 수수료(보안), 상태 렌트(저장).

- **사이클(Cycles)** 은 스마트 컨트랙트 실행 중 CKB-VM이 측정한 트랜잭션 검증에 사용된 계산 자원에 따라 채굴자에게 보상한다.
- **트랜잭션 수수료(Transaction Fees)** 는 네트워크 보안을 보호하기 위해 연산 능력을 제공한 것에 대해 채굴자에게 지급된다.
- **상태 렌트(State Rent)** 는 트랜잭션 데이터를 지속적으로 보관하기 위한 저장 공간 제공에 대해 채굴자에게 보상한다.

사이클과 트랜잭션 수수료는 트랜잭션을 처리하고 블록체인에 추가하는 데 필요한 일회성 지불이다. 상태 렌트는 데이터 지속성을 위해 지속적으로 지불된다.

CKByte의 소유는 Nervos에서 데이터 1바이트의 저장 권리를 부여한다. 새 Cell을 생성하려면, 사용자는 그 Cell이 점유하는 공간에 상응하는 CKBytes를 보유해야 한다. 이 CKBytes는 Cell의 수명 동안 잠기며, Cell이 소비되면 CKBytes는 해제되어 재사용할 수 있다. 상태 렌트는 CKBytes 잠금 기간 동안 자동으로 지불된다.

Nervos의 모든 자산은 데이터 저장을 필요로 하므로 상태 렌트의 대상이 된다. 이는 Nervos에서 자산 유지에 CKBytes가 필수적이기 때문에 직접적인 가치 정렬을 창출한다.

가치 정렬(Value Alignment)

플랫폼의 보안은 저장된 가치에 비례해 확장되어야 취약성을 방지할 수 있다. CKBytes는 네트워크를 보호하는 채굴자에게 보상한다. CKBytes 가치가 상승하면 네트워크 보호에 대한 보상도 증가하여, 저장된 가치에 상응하는 충분한 보안을 보장한다.

그러나 다자산 플랫폼에서는 네이티브 토큰이 자산 가치의 성장과 함께 상승하지 않을 때(약한 상관관계 때문에) “무거운 자산 문제”가 발생한다. Nervos는 CKByte를 데이터 저장에 연결하고 상태 렌트를 의무화함으로써 이를 해결한다. 이는 자산이 저장을 위해 CKBytes를 필요로 하고 그 기간 동안 상태 렌트를 발생시키므로 장기 수요를 창출한다.

상태 렌트(State Rent)

채굴자는 네트워크에서 데이터의 유효성과 보존을 보장하며, 사이클과 트랜잭션 수수료는 적절한 검증을 보장한다. 그러나 수수료가 정산되고 나면, 채굴자는 데이터 보존에 대한 추가 인센티브가 없다. 해결책으로서, 상태 렌트는 데이터 보존을 위해 채굴자에게 지속적으로 보상한다.

사용자가 Nervos에 데이터를 저장할 때, 점유한 공간에 기반한 소액의 상태 렌트 비용이 발생한다. 반복적인 선납 수수료는 사용자에게 불편하므로, Nervos는 네트워크 공간을 점유하는 사용자에게 표적화된 인플레이션을 사용한다.

데이터 저장 중에는 CKBytes의 일부가 잠기며 보상을 얻을 수 없다. 잠긴 CKBytes의 수량은 일정하지만, 인플레이션으로 인해 그 가치가 서서히 감소하며, 이는 Nervos에 데이터를 저장하는 사용자에게만 영향을 미친다. 이 작은 가치 감소가 상태 렌트를 구성한다.

상태 렌트를 자금 조달하는 인플레이션은 세컨더리 발행(Secondary Issuance)으로 알려진 과정에서 발생한다. 네트워크 공간을 점유하지 않는 Nervos 사용자는 CKBytes를 Nervos DAO에 예치(lock)함으로써 세컨더리 발행에서 보상을 얻을 수 있다. 다음 섹션에서 이 메커니즘을 더 자세히 다룬다.

베이스 발행(Base Issuance)

네트워크 초기 출시 시점에는 CKBytes의 가치가 낮아 보안도 낮았다. Nervos를 자산 저장에 매력적으로 만들기 위해, 비트코인의 채굴 과정과 유사하게 베이스 발행을 통해 보안을 일시적으로 보조한다. 채굴자는 트랜잭션 처리를 통해 고정된 CKBytes 보상을 받는다. 시간이 흐르며 저장 자산의 가치가 상승함에 따라 보 조금은 감소한다.

베이스 발행은 약 4년마다 반감되며, 336억 CKBytes 상한에 도달할 때까지 투명하고 예측 가능한 인플레이션 일정을 따른다.

세컨더리 발행(Secondary Issuance)

베이스 발행 이후에는, 트랜잭션 수수료만으로는 채굴자 인센티브나 데이터 보존에 충분하지 않을 수 있다. 연간 13.44억 CKBytes의 소규모 고정 인플레이션을 갖는 세컨더리 발행은 이러한 우려를 해소한다.

베이스 발행과 달리, 세컨더리 발행은 Nervos 공간을 점유하거나 Nervos DAO 외부에 CKBytes를 보유한 사용자만을 대상으로 한다. 세컨더리 발행은 다음에 분배된다:

- 채굴자(상태 렌트로)
- Nervos DAO 사용자
- 지속적인 개발을 위한 Nervos 재단(트레저리)

Nervos DAO

CKBytes 보유자는 토큰을 Nervos DAO에 예치(lock)함으로써 보상을 얻을 수 있다. 보상은 세컨더리 발행에 비례하여 누적되며, 장기적인 인플레이션 효과를 상쇄해 가치를 유지한다. Nervos에서 공간을 점유하는 사용자는 CKBytes가 잠겨 있으므로 Nervos DAO에 예치할 수 없다. 공간 점유 Cell이 소비되어 CKBytes가 해제되면, 이를 Nervos DAO에 예치할 수 있다. 이 메커니즘은 불필요한 데이터 제거를 장려하여 블록체인의 장기적 관리 가능성을 보장한다.

자세한 내용은 [RFC Crypto-Economics of the Nervos Common Knowledge Base](#) 를 참고하라.

xUDT

xUDT (Extensible User-Defined Token)

xUDT(Extensible User-Defined Token)는 Nervos CKB에서 대체형 토큰을 생성하고 관리하기 위한 토큰 표준이다. 이더리움의 [ERC-20](#) 에 해당한다고 볼 수 있지만 — 더 높은 유연성을 제공한다.

xUDT는 최소한의 사전 정의된 검증 모델을 사용하는 기초 [Simple User-Defined Token\(sUDT\)](#) 위에 구축되었다. 이에 대비해, xUDT는 외부 스크립트를 통해 **맞춤 검증 로직** 을 부착할 수 있게 하여 **확장성** 을 도입한다 — 이를 통해 더 발전된 거버넌스, 발행 규칙, 토큰 동작을 구현할 수 있다.

사용 사례

xUDT는 sUDT가 제공하지 못하는 온체인 프로그래머블 동작이나 거버넌스 로직이 필요한 시나리오에 적합하다. 일반적 사용 사례는 다음과 같다:

- **최대 토큰 공급량 강제:** 확장 스크립트를 사용해 트랜잭션 출력의 총 토큰 수가 사전 정의된 상한 아래로 유지되도록 보장한다.
- **시간 기반 전송 제한:** 특정 시점 이후에만 전송을 허용하는 타임락 메커니즘을 스크립트에 구현한다.
- **효율적인 거래소 계정 표현:** 모든 사용자 잔액을 단일 Cell에 저장된 희소 머클 트리(SMT)로 표현한다. 스크립트는 각 업데이트의 정확성과 무결성을 검증한다.
- **스크립트 로직 기반의 프로그래머블 발행:** 특정 암호학적 조건이 충족된 경우에만 토큰 발행을 허용한다 — 예를 들어 secp256k1 공개키가 사전 정의된 소유자 해시와 일치하는지 검증하여, 추가 제어 Cell 없이 스크립트 기반 “소유자 모드”를 활성화한다.

추가 자료

- [DApp 튜토리얼: 대체형 토큰 생성](#)
- [xUDT 동작 방식](#)
- [xUDT RFC](#)

Spore 프로토콜

Spore는 Nervos CKB 위에 구축된 **온체인 디지털 오브젝트(DOB) 프로토콜** 로, 디지털 오브젝트의 안전하고 효율적이며 유연한 생성 및 전송을 가능하게 한다. 이는 내재 가치 상환, 프라이버시 보존형 소유권, 무수수료 트랜잭션 같은 기능을 모두 온체인에서 지원한다.

사용 사례

- 대체 불가능 토큰(NFT)
- 디지털 컬렉티블
- 게임 자산
- 상환 가능한 디지털 바우처 또는 증서

비교 표

Spore는 고유 디지털 자산을 표현하기 위한 토큰 표준이라는 점에서, 이더리움의 [ERC-721](#) 및 비트코인의 [Ordinals](#) 와 비교될 수 있다.

Features	ERC-721 (Ethereum)	Ordinals (Bitcoin)	Spore Protocol (Nervos CKB)
Content	대부분 오프체인	온체인	온체인
Model	계정 기반	UTXO	UTXO 기반 Cell 모델
Privacy	없음	경우에 따라	있음
Immutability	가변적	예, 불변	예, 불변
Redeemable	없음	없음	있음
Simplicity	복잡함	단순	단순(1 DOB당 1 Cell)
Content Size Limit	-	4MB / 400KB	500KB
Transaction Cost	ETH 가스	BTC 수수료	무수수료(선택적 지불)
Basis of Value	시장, 인위적 희소성	사토시 희소성	내재 가치 + 시장 가치

Spore 프로토콜의 차별점

내재 가치 & 상환

Spore는 Nervos 네트워크의 네이티브 통화인 CKBytes(CKB)에 연결된 독특한 클래스의 디지털 자산이다. 이 내재적 연결은 각 Spore가 생성 시 할당된 CKBytes 양을 반영하는 기초 가치를 갖도록 보장한다.

- **내재 가치 상환:** Spores는 언제든지 기반 CKBytes로 녹여(melt) 되돌릴 수 있다. 이는 [spore-sdk](#)가 제공하는 meltSpore API를 통해 이루어지며, 소유자는 원하는 때에 디지털 자산을 CKBytes 통화로 전환할 수 있는 유연성을 가진다.
- **토크노믹스와 내재 가치 성장:** Nervos CKB 온체인 공간의 활용이 증가함에 따라, 각 Spore에 잠겨 있는 CKBytes의 가치는 자연스럽게 상승한다. 이 성장은 구조화된 발행 일정과 상태 렌트 메커니즘을 포함하는 CKB의 경제 모델에 의해 유기적으로 주도되며, 자세한 내용은 [CKB Cryptoeconomics RFC](#) 를 참조하라.

온체인 소유권 & 프라이버시

Spore 프로토콜은 계정 기반 NFT 프로토콜에 내재하지 않은 향상된 프라이버시와 보안 기능을 제공하며, 특히 NFT 보유자가 직면하는 과제를 해결한다.

- **Cell 모델을 통한 프라이버시:** 모든 트랜잭션이 단일 주소에 연결되는 계정 기반 블록체인과 달리, Nervos CKB는 독립적인 Cell을 활용하는 UTXO 기반 [Cell 모델](#)을 사용한다. 이 모델에서 각 트랜잭션은 —동일한 키로 제어되더라도— 새로운 주소를 사용하여, 소유권을 여러 주소에 효과적으로 분산시키고 이를 단일 정체성에 연결하기 어렵게 만든다.

무수수료 전송

Spore 프로토콜은 Spore 또는 Cluster 생성 시 소량의 CKBytes를 예약함으로써 무수수료 전송을 지원한다.

- **Capacity Margin:** Spore 또는 Cluster를 생성할 때, createSpore 또는 createCluster API는 일반적으로 1 CKB인 capacityMargin 을 지정하여 향후 트랜잭션 수수료를 충당하도록 예약할 수 있다. 1 CKB 마진으로, Spore는 추가 자금 조달 없이 약 **100,000 회** 전송될 수 있다. 이는 신규 보유자가 CKBytes를 관리하거나 획득할 필요 없이 Spore와 상호작용할 수 있는 매끄러운 사용자 경험을 제공한다.
- **프라이버시 보존:** 외부에서 트랜잭션에 자금을 조달할 필요를 제거함으로써, Spore 프로토콜은 수수료 지불을 통한 지갑 주소 연계 —공개 블록체인에서의 일반적인 비식별화 벡터—의 위험을 줄인다. 이 설계는 완전한 온체인 무결성을 유지하면서 프라이버시를 향상시킨다.

다중 콘텐츠 타입 지원

Spore 프로토콜은 JPEG 같은 정적 이미지에 국한되지 않고 광범위한 디지털 콘텐츠 형식을 지원하도록 설계되었다. 이 유연성은 개발자와 크리에이터가 다양한 유형의 미디어와 데이터를 직접 온체인에 저장하고 관리하도록 한다.

DOB/0 프로토콜

DOB/0 프로토콜은 디지털 오브젝트(DOB)의 속성을 정의하고 디코딩하기 위한 표준 구성 및 인터페이스 명세이다. 여기에는 다음 개념이 포함된다:

- **DNA** – 핵심 데이터 구조
- **Pattern** – 해석을 위한 템플릿 또는 스키마
- **Decoder** – DOB를 파싱하고 렌더링하는 로직 계층

DOB/0는 애플리케이션이 DOB와 상호작용하고 의미를 추출하는 방식을 표준화하여, 통합 복잡성을 크게 줄인다.

DOB/1 프로토콜

DOB/1 프로토콜은 DOB/0 위에 구축되어, SVG 이미지 렌더링에 대한 표준화된 지원을 도입한다. 이를 통해 DOB는 다음을 통해 온체인에서 직접 시각적 콘텐츠를 표현할 수 있다:

- DOB 속성이 SVG로 변환되는 방식 정의
- 시각적 표현의 자동 생성 허용
- 디지털 오브젝트를 해석 가능하고 표시 가능하게 만들

이는 완전 온체인 아트의 NFT 스타일 사용 사례에 문을 연다.

추가 자료

- [DApp 튜토리얼: Spore 프로토콜로 DOB 생성](#)
- [Spore 프로토콜 동작 방식](#)
- [Spore Docs](#)

RGB++ 프로토콜

RGB++란 무엇인가?

RGB++는 단일 사용 씬(single-use seals)과 클라이언트 사이드 검증(client-side validation) 기법을 사용해 상태 변화와 트랜잭션 검증을 관리하는 RGB 프로토콜의 확장 버전이다. 이는 비트코인의 UTXO 집합을 등형 바인딩을 통해 Nervos CKB의 Cell에 매핑하며, 양 체인의 스크립팅 제약을 활용해 상태 계산의 정확성과 소유권 변경의 유효성을 보장한다.

원래 RGB 프로토콜의 여러 기술적 한계를 해결하면서, RGB++는 클라이언트 사이드 검증, 트랜잭션 폴딩, 계약 간 공유 상태, 비대화형 전송 등을 포함한 다양한 신규 능력을 열어 준다. 이는 교차 체인 트랜잭션 없이, 보안을 훼손하지 않고 비트코인에 확장 가능하고 튜링 완전한 스마트 컨트랙트 기능을 도입한다.

단일 사용 씬(Single-Use Seals)

단일 사용 씬 개념은 2016년 7월 Peter Todd에 의해 처음 소개되었다. 이는 메시지에 대한 전자적 씬의 잠금을 허용하여, 그 메시지가 단 한 번만 사용되도록 보장한다. 구체적으로, 비트코인의 미사용 트랜잭션 출력(UTXO)은 메시지를 위한 씬로 기능할 수 있으며, 비트코인 시스템의 합의 메커니즘은 이 UTXO가 단 한 번만 소비되도록 보장하므로, 이 씬은 한 번만 열릴 수 있다.

RGB 프로토콜은 비트코인 UTXO에 기반한 단일 사용 씬을 사용하여, RGB 상태 변화를 비트코인 UTXO 소유와 매핑한다. 이를 통해 비트코인 시스템은 RGB 상태의 소유권을 보장하고, UTXO 히스토리를 통해 모든 상태 변화를 추적 가능하게 한다. 단일 사용 씬을 통해, RGB 프로토콜은 비트코인의 이중 지불 방지와 트랜잭션 추적 가능성을 — 둘 다 비트코인의 합의 메커니즘에 의해 집행되는 — 계승한다.

클라이언트 사이드 검증(Client-Side Validation)

RGB 프로토콜에는 비트코인 합의에 의해 직접 검증될 수 없는 사용자 상태가 포함되어 있다. 이는 사용자가 오프체인 계산을 활용하여 RGB 상태 변화가 기대에 부합하는지 검증해야 함을 요구한다. 클라이언트 사이드 검증은 사용자가 무관한 브랜치가 아닌, 관련된 UTXO 브랜치 히스토리만 검증하도록 한다. RGB의 상태 보안은 중앙화된 제3자에 의존하지 않고 클라이언트 사이드 검증을 통해 제공된다.

RGB 프로토콜의 문제

원본 RGB 프로토콜은 비트코인의 UTXO에 앵커되고 전적으로 클라이언트 사이드에서 검증되는 혁신적 모델을 도입했지만, 실제 사용에서 몇 가지 실용적 과제에 직면했다:

- **데이터 가용성(DA):** 사용자는 소유권을 증명하기 위해 전체 이력 데이터를 보관해야 했고, 이는 라이트 클라이언트 구축을 어렵게 했다.
- **P2P 의존성:** 트랜잭션 전파가 독립 P2P 네트워크에 의존하여, 사용자가 직접 조정(예: 수신자가 영수증 제공)해야 했다.
- **미성숙한 가상 머신:** RGB가 사용하는 AluVM은 도구 체인 지원과 프로덕션 준비도가 부족했다.
- **제한된 공유 상태:** RGB 프로토콜에는 중앙화된 당사자나 서명자에 의해 제어되지 않는 비호스팅 계약 간의 공유 상태를 관리하는 신뢰할 수 있는 방법이 부족하여, DEX, DAO, 협업 프로토콜 같은 다자간 애플리케이션 지원이 어려웠다.

RGB++의 핵심 기능

등형 바인딩(Isomorphic Binding)

RGB++는 비트코인 UTXO와 Nervos CKB의 Cell 간 등형 바인딩을 도입하여 원본 RGB 프로토콜의 핵심 한계를 극복한다. RGB에서는 UTXO가 소유권을 결정하고, 오프체인 커밋먼트와 단일 사용 쉘이 상태를 관리한다. RGB++는 각 비트코인 UTXO를 CKB Cell에 매핑하여, 비트코인의 락 스크립트를 통해 소유권을 동기화하고, CKB Cell의 data 및 type 필드를 통해 상태를 직접 관리한다. 이 설계는 온체인 지원과 조합성을 갖춘 검증 가능한 상태 전이를 가능하게 한다.

블록체인 보강형 클라이언트 사이드 검증

각 RGB++ 트랜잭션은 비트코인과 CKB에서 동기화된 트랜잭션 쌍을 생성한다. 비트코인 측 트랜잭션은 RGB 프로토콜과의 호환성을 유지하는 반면, 대응되는 CKB 트랜잭션은 전체 클라이언트 사이드 검증의 필요를 대체하는 검증 가능한 상태 앵커를 제공한다.

사용자는 다음 중에서 선택할 수 있다:

- 연관된 CKB 트랜잭션을 직접 검사하여 트랜잭션을 검증하거나,
- 로컬 비트코인 UTXO 이력을 사용해 전통적인 클라이언트 사이드 검증을 수행한다.

일부 고급 사례(예: 트랜잭션 폴딩)에서는 이중 지불 방지를 위해 CKB 블록 헤더에 대한 경량 접근이 여전히 필요하다. 이 하이브리드 모델은 확장 가능한 온체인 대안을 통해 클라이언트 사이드 검증의 유연성을 보존하면서 이를 강화한다.

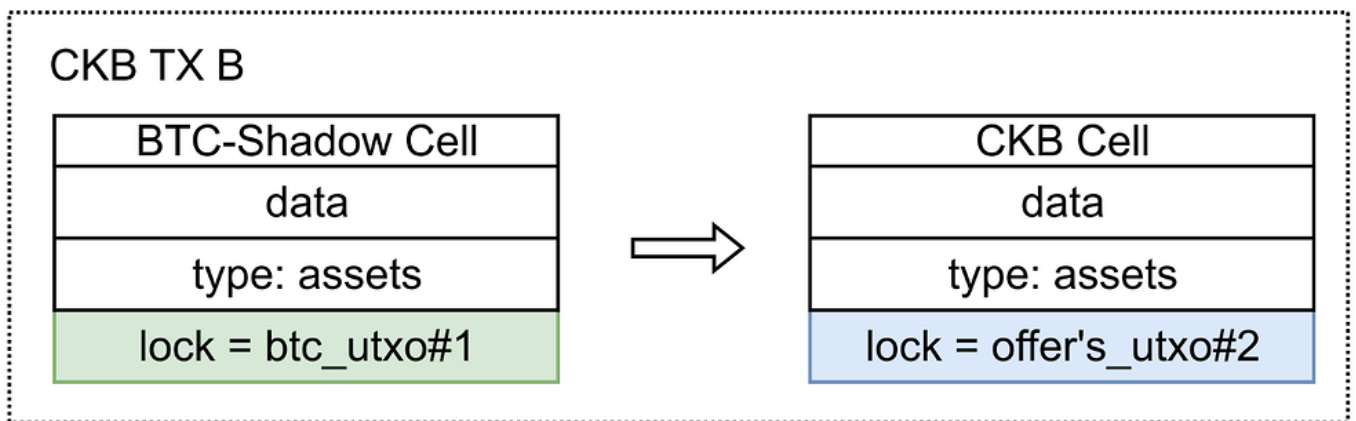
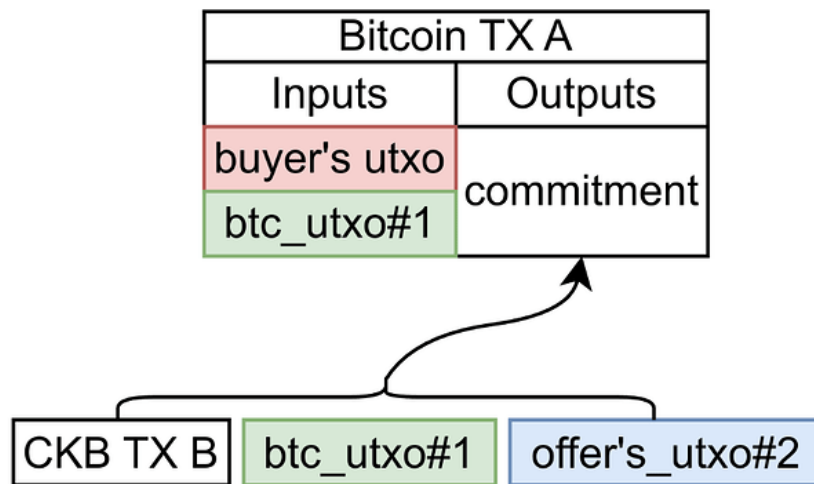
사용 사례

에어드롭

주소 목록과 해당 금액이 주어지면, RGB++를 사용하여 완전한 에어드롭 애플리케이션을 구현할 수 있다. 보류 중인 에어드롭 데이터와 청구된 주소 목록이 Cell 데이터의 SMT로 저장된다고 가정하면, 사용자는 자신의 주소에서 손쉽게 에어드롭을 수령할 수 있다.

DEX & AMM

RGB++는 UTXO 구조를 최적화하여, 중개자 없이 UTXO 기반 자산 교환 프로토콜을 원활하게 지원한다. 또한 RGB++는 AMM 모델을 강화하기 위해 그리드 트레이딩 설계를 채택한다. Uniswap의 AMM 모델과 비교할 때, 그리드 트레이딩 모델은 UTXO 기반 자산 거래에 더 높은 맞춤성과 적합성을 제공한다.



위 예시는 판매자의 보류 주문에 응답하여 구매자가 *BTC*로 *RGB* + *xUDT*를 구매하는 상황을 보여준다. 트랜잭션 구조에는 충분한 *BTC*와 PBST 서명을 포함한 구매자의 UTXO가 포함되므로, 구매자는 판매자의 요구사항을 충족하는 CKB 트랜잭션을 생성할 수 있다. 이후 구매자는 이 서명된 CKB 트랜잭션을 판매자에게 전송한다. 판매자는 이어서 *BTC* 트랜잭션과 CKB 트랜잭션을 연속적으로 온체인에 제출하여 거래를 완료한다.

추가 자료

- [RGB++ 라이트 페이퍼](#)
- [RGB++ 스크립트 표준](#)
- [RGB++ 보안 분석](#)

자료(Resources)

용어집

용어집 섹션

- [일반 용어집](#)
 - [경제 용어집](#)
 - [기술 용어집](#)
-

일반 용어집

Account (계정)

분산 원장에서 사용자의 잔액 및 기타 정보를 보관하기 위해 사용되는 기본 객체의 일종.

Address (주소)

문자와 숫자로 구성된 문자열 형태의 라벨로, 사용자의 온체인 익명 신원을 나타낸다. 암호 자산은 주소로 송신되거나 수신될 수 있다.

CKB 주소는 [Lock Script](#)를 하나의 검증 가능하고, 사람이 읽기 쉬운 단일 문자열 형식으로 패키징한다. 하나의 공개/개인 키 쌍은 여러 Lock Script를 생성할 수 있으며, 결과적으로 여러 주소를 생성할 수 있다.

동의어

- [Payment Address](#)

함께 보기

- [Lock Script](#)
 - [Private Key](#)
 - [Public Key](#)
-

Asset (자산)

가치를 가지거나, 가치를 지닌 개체를 나타내는 데이터 조각.

동의어

- [Digital Asset](#)

- [Token](#)
-

Block (블록)

타임스탬프와 이전 블록의 지문으로 표시된 트랜잭션의 집합.

블록 헤더는 작업증명(Proof of Work)을 생성하기 위해 해시되며, 이를 통해 트랜잭션이 검증된다.

유효한 블록은 네트워크 합의에 따라 메인 블록체인에 추가된다.

함께 보기

- [Blockchain](#)
 - [Confirmation](#)
 - [Transaction](#)
-

Block Height (블록 높이)

블록 높이는 블록체인에서 확인된 총 블록 수를 의미하며, 특정 블록 높이를 지정할 때 블록을 식별하는 데 사용된다.

각 블록 높이에는 정확히 하나의 블록만 존재한다.

동의어

- [Height](#)

함께 보기

- [Block](#)
 - [Blockchain](#)
 - [Confirmation](#)
-

Block Interval (블록 간격)

[Block Time](#)이라고도 한다.

블록 간격은 암호화폐 블록체인에서 새 블록을 생성하는 데 걸리는 시간이다.

이는 네트워크 내 채굴자 또는 검증자가 하나의 블록 내 트랜잭션을 검증하고 새 블록을 생성하는 데 필요한 시간을 의미한다.

Nervos 블록체인에서는 가변적이다.

비트코인 블록체인의 블록 간격은 약 10분이며, Nervos에서는 일반적으로 10초 미만이다.

동의어

- [Block Time](#)

함께 보기

- [Block](#)
 - [Blockchain](#)
-

Block Propagation (블록 전파)

네트워크 내 대다수 풀노드에 새 블록을 동기화하는 과정.

블록 전파는 비트코인의 확장을 제한하는 주요 병목 현상으로 알려져 있다.

블록 전파 시간은 새 블록이 네트워크의 대부분 노드에 도달하는 데 걸리는 평균 시간이다.
긴 전파 지연은 51% 공격에 대한 노드의 저항력을 약화시킨다.

동의어

- [Propagation](#)

함께 보기

- [Block](#)
 - [Broadcast](#)
 - [Full Node](#)
-

Block Reward (블록 보상)

채굴자가 트랜잭션 블록을 성공적으로 블록체인에 추가한 후 그 계정에 적립되는 암호화폐의 양.
Nervos CKB에서는 채굴자가 블록을 성공적으로 추가한 후 CKBytes가 계정에 적립된다.

함께 보기

- [Block](#)
 - [Block Subsidy](#)
-

Block Time (블록 시간)

[Block Interval](#)의 다른 표현이다.

동의어

- [Block Interval](#)

Blockchain (블록체인)

점진적으로 확장되는 기록 목록을 유지하는 데이터 구조로, 블록 형태로 구성된다.

첫 번째 블록을 제외한 모든 블록은 이전 블록과 암호학적으로 연결되어 체인 구조를 형성한다.

이 암호 연결을 통해 마지막 블록을 가진 모든 당사자는 해당 블록 생성 이후 과거 데이터가 변경되지 않았음을 검증할 수 있다.

함께 보기

- [Block](#)

BLS Signature (BLS 서명)

서명 및 검증에 사용되는 암호 서명 체계. BLS는 Boneh–Lynn–Shacham의 약자이다.

함께 보기

- [Boneh–Lynn–Shacham on Wikipedia](#)

Broadcast (브로드캐스트)

블록이 블록체인 네트워크의 모든 노드로 전송되는 과정.

함께 보기

- [Blockchain](#)
- [Node](#)

Capacity (용량)

Nervos CKB에서 하나의 Cell이 차지할 수 있는 최대 공간(바이트 단위).

동의어

- [CKByte](#)

함께 보기

- [CKB](#)
- [Common Knowledge Base](#)
- [Common Knowledge Byte](#)

- [Nervos Blockchain](#)
-

Cell (셀)

CKB의 기본 상태 단위로, 사용자는 그 안에 임의의 상태(state)를 포함시킬 수 있다.
Nervos CKB의 모든 데이터는 Cell에 저장된다.

Cell은 capacity, data, type, lock 의 네 가지 필드를 가진다.

동의어

- [Micro-State](#)

함께 보기

- [Data](#)
 - [Lock](#)
 - [Lock Script](#)
 - [Type Script](#)
 - [Dead Cell](#)
 - [Live Cell](#)
 - [Cell Model](#)
 - [UTXO on Bitcoin.org](#)
-

Cell Model

Nervos CKB에서 상태를 관리하는 방식을 나타낸다. Cell Model은 비트코인의 UTXO나 이더리움의 계정 모델보다 더 일반적인 상태 모델이다.

Cell Model은 이더리움의 계정 모델이 가진 많은 장점을, 비트코인의 UTXO 모델이 가진 자산 소유권 및 증명 기반 검증 특성과 결합한 새로운 구성이다.

함께 보기

- [Lock Script](#)
 - [Type Script](#)
 - [Cell Model](#)
 - [Cell Model on the Nervos Blog](#)
 - [UTXO on Bitcoin.org](#)
-

Censorship Resistance (검열 저항성)

블록체인에서의 검열 저항성은 악의적 주체가 정직한 사용자가 생성한 일련의 트랜잭션이 확인되는 것을 방해하기 어렵다는 것을 의미한다.

Chain (체인)

블록체인의 약칭.

동의어

- [Blockchain](#)
-

CKB

문맥에 따라 다른 의미의 약어:

- Common Knowledge Base — Nervos 네트워크의 레이어 1 블록체인.
- Common Knowledge Byte — Nervos Common Knowledge Base의 네이티브 토큰.

동의어

- [Common Knowledge Base](#)
 - [Common Knowledge Byte](#)
-

CKByte

Common Knowledge Byte의 축약형.

CKByte는 때때로 CKB로도 축약된다. 거래소에서는 종종 CKB를 티커로 사용한다.

동의어

- [CKB](#)
- [Common Knowledge Byte](#)

혼동하지 말 것

- [Common Knowledge Base](#)
-

CKB-VM

암호 알고리즘에 독립적인 가상 머신으로, RISC-V 명령어 집합 기반의 VM이며 온체인 및 오프체인 코드를 실행한다.

함께 보기

- [RISC-V](#)
 - [Script](#)
 - [Virtual Machine on Wikipedia](#)
-

Code Hash

특정 데이터 조각 또는 Type ID로 참조되는 특정 Cell을 가리키는 해시 값을 포함하는 Cell의 필드.

함께 보기

- [Cell](#)
 - [Type ID](#)
-

Cold Storage (콜드 스토리지)

콜드 월렛(인터넷에 연결되지 않는 지갑)에 자금을 보관함으로써 자산을 보호하는 방법.

함께 보기

- [Cold Wallet](#)
 - [Hardware Wallet](#)
 - [Wallet](#)
-

Cold Wallet (콜드 월렛)

오프라인에서 자산을 보호하는 데 사용되는 지갑. 이 지갑은 영구적으로 인터넷과 분리되어 있어, 활성 인터넷 연결에 의존하는 공격에 취약하지 않다.

함께 보기

- [Cold Storage](#)
 - [Wallet](#)
-

Commit

Nervos CKB의 합의 알고리즘인 NC-Max에는 **propose** 와 **commit** 두 단계가 있다. Commit은 이전에 제안된 유효한 트랜잭션을 새 블록에 포함하는 과정이다.

Commit-Chain

하나 이상의 운영자가 온체인 상태 업데이트 커밋(개별 트랜잭션 데이터는 포함하지 않음)을 통해 트랜잭션을 오프체인에서 처리할 수 있게 하는 방식.

Commit Reward

이전에 제안된 트랜잭션을 포함했을 때 채굴자에게 CKBytes로 지급되는 보상.

Common Knowledge Base

Nervos 네트워크를 위한 탈중앙 신뢰의 기반을 제공하는 레이어 1 작업증명 블록체인.

동의어

- [CKB](#)
- [Nervos CKB](#)

혼동하지 말 것

- [Common Knowledge Byte](#)

함께 보기

- [Nervos CKB on Nervos.org](#)
-

Common Knowledge Byte

Nervos 레이어 1 블록체인(Common Knowledge Base)의 네이티브 토큰.

Common Knowledge Byte는 흔히 CKByte 또는 CKB로 축약된다.

CKByte를 보유하면 Nervos CKB 블록체인에 1바이트의 데이터를 저장할 권리가 부여된다.

동의어

- [CKB](#)
- [CKByte](#)

혼동하지 말 것

- [Common Knowledge Base](#)

함께 보기

- [Capacity](#)
 - [Nervos CKB](#)
 - [Shannon](#)
-

Confirmation (확인)

트랜잭션이 네트워크에 의해 수락·검증되어 블록에 포함되는 과정.

함께 보기

- [Block](#)
 - [Transaction](#)
 - [Unconfirmed](#)
-

Consensus (합의)

여러 분산 참여자 간에 실행되는 알고리즘으로, 일부 참여자가 결함이 있거나 악의적이더라도 이 알고리즘을 성실히 실행하는 모든 참여자가 특정 데이터 값에 대해 합의에 도달할 수 있도록 보장한다.

함께 보기

- [NC-MAX](#)
-

Consume (소비)

Live Cell을 트랜잭션의 입력으로 사용하는 과정. 소비되면 Live Cell은 Dead Cell로 전환된다.

Contract Account (컨트랙트 계정)

다른 계정으로부터 이벤트를 수신할 때마다 자동으로 코드를 실행하는 계정.

함께 보기

- [Account](#)
-

Cryptocurrency (암호화폐)

블록체인과 암호학에 의해 보호되는 디지털/가상 통화로, 위조나 이중 지불이 사실상 불가능하다.

함께 보기

- [Digital Currency](#)
 - [Fiat Currency](#)
-

Cryptographic Signature (암호학적 서명)

간결한 형태의 증명 데이터. 암호학적 서명 기법은 네트워크 전반의 트랜잭션 메시지의 무결성 및 부인 방지를 검증하는 암호화폐 네트워크의 근본 구성 요소이다.

동의어

- [Private Key](#)
 - [Signature](#)
-

Cryptography (암호학)

적대적 행위가 존재하는 환경에서 안전한 통신을 위한 기술을 연구·실천하는 분야.

함께 보기

- [Cryptography at Wikipedia](#)
-

Cycles (사이클)

스크립트 실행에 필요한 RISC-V 계산 사이클 수. 악의적 행위(예: 무한 루프)를 방지하기 위한 척도이기에 cycles라 부른다.

이더리움의 Gas와 유사한 개념으로, 최적의 성능과 보안을 보장하기 위해 사이클을 설정한다. 스크립트는 사이클 한도를 준수해야 하며, 그렇지 않으면 블록이 CKB 노드에 의해 거부된다.

함께 보기

- [Estimate cycles](#)
- [Max_block_cycles](#)
- [Rules for calculating cycles in RFC on Nervos Network Github](#)
- [Script](#)
- [RISC-V](#)
- [Ethereum's Gas](#)

DAO

탈중앙 자율 조직(DAO)은 규칙이 컴퓨터 프로그램 형태로 인코딩되어 투명하고, 구성원에 의해 통제되며, 중앙화된 개체의 영향에서 벗어난 조직 — 즉, 중앙 리더십이 없는 구성원 소유 커뮤니티를 말한다. DAO의 재무 트랜잭션 기록과 프로그램 규칙은 블록체인에 유지된다.

함께 보기

- [DAO on Wikipedia](#)
-

DApp

탈중앙 애플리케이션. 최소한 스마트 컨트랙트와 웹 UI를 포함한다. 더 넓게는, 개방형 탈중앙 P2P 인프라 서비스 위에 구축된 웹 애플리케이션을 의미한다. 또한 많은 dApp은 탈중앙 저장소 및/또는 메시지 프로토콜과 플랫폼을 포함한다.

Data (데이터)

Cell Model에서 data 는 임의의 바이트를 저장할 수 있는 Cell의 필드.

함께 보기

- [Cell](#)
 - [Cell Model](#)
-

Decentralization (탈중앙화)

블록체인에서 탈중앙화는 통제와 의사결정을 중앙화된 개체(개인, 조직, 또는 그 집단)로부터 분산 네트워크로 이전하는 것을 의미한다.

함께 보기

- [Distributed](#)
 - [Decentralization on Wikipedia](#)
-

DeFi

“탈중앙 금융”의 약어. 중개자 없이 블록체인에 의해 뒷받침되는 금융 서비스를 제공하는 dApp의 광범위한 범주로, 인터넷 연결만 있으면 누구나 참여할 수 있다.

Digital Asset (디지털 자산)

가치를 지니거나, 가치를 지닌 다른 개체를 나타내는 개별 데이터 조각.

디지털 자산은 일반적으로 토큰으로 표현되며, 디지털 화폐로 사용되거나 부동산 같은 물리적 항목을 나타낼 수도 있다.

동의어

- [Asset](#)
 - [Token](#)
-

Digital Currency (디지털 통화)

주로 인터넷상에서 디지털 형태로 존재하는 통화. 카드, 지폐, 동전과 같은 물리적 표현이 있을 수 있으나 부차적 매체이다.

함께 보기

- [Cryptocurrency](#)
-

Distributed (분산)

작업 병렬화, 중복성 추가, 단일 실패 지점 제거를 위해 구성 요소가 여러 노드에 분산된 시스템.

함께 보기

- [Decentralization](#)
-

Digital Object (DOB) (디지털 오브젝트)

그 콘텐츠가 완전히 온체인에 저장된 비대체형 암호화 자산으로, 콘텐츠와 가치 간의 내재적 연결을 확립한다.

함께 보기

- [Non-Fungible-Token](#)
-

Double-Spending (이중 지불)

디지털 토큰이 두 번 이상 지불되는 위험. 블록체인 맥락에서는, 디지털 토큰을 지출한 트랜잭션이 확인 후 취소되고 동일한 토큰이 다른 트랜잭션에서 다시 지출될 때 발생한다.

함께 보기

- [Confirmation](#)
 - [Cryptocurrency](#)
 - [Token](#)
 - [Transaction](#)
-

Epoch (에포크)

일련의 블록에 대한 시간 구간.

Nervos에서는 새로운 에포크에서 PoW 난이도가 변경된다. 동일한 에포크의 모든 블록은 같은 난이도 타킷을 공유한다. 난이도 조정 알고리즘은 잉클 블록 비율을 2.5%, 에포크 길이를 4시간으로 안정화하는 것을 목표로 한다.

함께 보기

- [Block](#)
-

Fee (수수료)

송신자는 요청한 트랜잭션을 처리하도록 네트워크에 수수료를 포함하는 경우가 많다. 합의에 최소 수수료율은 없지만, CKB의 [P2P](#) 네트워크에는 KB당 1,000 Shannon의 최소 수수료율이 있다. (1 Shannon = 10^{-8} CKB)

함께 보기

- [Shannon](#)
 - [Fee rate](#)
 - [Fee rate in RFC on Nervos Network Github](#)
-

Fee Rate (수수료율)

사용자가 자신의 트랜잭션이 블록에 포함되도록 채굴자에게 제공하는 바이트당 팁.

비트코인의 [Fee Rate\(feerate\)](#)와 같은 개념이다.

함께 보기

- [Fee](#)

First-Class Assets (일급 자산)

1. 자산 자체가(자산에 대한 참조가 아니라) 스마트 컨트랙트 상호작용에서 직접 전달될 수 있고, 2) 중개자 없이 소유자가 직접 제어할 수 있는 자산.

함께 보기

- [First-Class Asset](#)
 - [Introduction to First-Class Asset](#)
-

Fork (포크)

프로토콜 변경으로 대체 체인이 생성되거나, 채굴 중 두 개의 가능한 블록 경로 간 일시적 분기가 발생하는 것.

Full Node (풀 노드)

CKB 네트워크의 핵심 구성 요소. 전체 블록체인을 저장·동기화하고, 블록과 트랜잭션의 유효성을 검증하며, 네트워크 합의 규칙을 시행한다.

```
ckb init --chain Mainnet && ckb run
```

함께 보기

- [Node](#)
 - [Light Node](#)
 - [Mining Node](#)
-

Full Payload Format (풀 페이로드 형식)

폐기(deprecated)된 풀 페이로드 형식으로, Lock Script의 모든 데이터 필드를 직접 인코딩한다. 이 폐기된 풀 페이로드 형식의 인코딩 규칙은 [Bech32](#)이다.

함께 보기

- [Lock Script](#)
-

Fungible Token (대체형 토큰)

달러 같은 법정화폐나 비트코인 같은 암호화폐가 될 수 있다.

대체형 토큰/자산은 분할 가능하며 고유하지 않다.

함께 보기

- [Non-Fungible Token](#)
 - [Token](#)
 - [User-Defined Token](#)
-

Gas Limit

트랜잭션 또는 블록이 소비할 수 있는 가스의 최대량.

함께 보기

- [Ethereum's Gas](#)
-

Hard-Fork (하드 포크)

블록체인의 영구적 분기. 업그레이드되지 않은 노드가, 새로운 합의 규칙을 따르는 업그레이드된 노드가 만든 블록을 검증할 수 없을 때 흔히 발생한다. 일반적인 포크, 소프트 포크, 소프트웨어 포크, Git 포크와 혼동하지 말 것.

Hardware Wallet (하드웨어 월렛)

콜드 월렛의 한 형태. 사용자의 개인 키(블록체인 네트워크에서 아웃고잉 트랜잭션을 승인하는 데 사용되는 중요한 정보)를 보안 하드웨어 장치에 저장하는 암호화폐 지갑.

함께 보기

- [Cold Storage](#)
 - [Cold Wallet](#)
 - [Private Key](#)
 - [Wallet](#)
-

Hash (해시)

가변 길이 입력으로부터 생성되는 고정 길이의 지문으로, 해시 함수에 의해 생성된다.

Hash Rate (해시레이트)

마이닝 시 초당 수행되는 연산량을 나타내는, 연산 능력의 척도. 이러한 연산은 “해싱”이라고 불린다.

함께 보기

- [Miner](#)
 - [Network Hash Rate](#)
-

Height (높이)

블록 높이의 약칭.

동의어

- [Block Height](#)
-

Light Client (라이트 클라이언트)

저자원 노드로서, 전체 블록체인을 저장하지 않고도 암호학적으로 안전한 방식으로 블록체인과 동기화할 수 있게 해준다.

Light Node (라이트 노드)

블록체인의 헤더만 다운로드하여 자원을 절약한다. 전체 트랜잭션 데이터는 다른 노드에 의존하며, 네트워크 참여를 위한 경량 옵션이다.

```
ckb-light-client run --config-file ./testnet.toml
```

함께 보기

- [Node](#)
 - [Full Node](#)
 - [Mining Node](#)
-

Mempool (메모리 풀)

“memory pool”의 약자. 네트워크에 브로드캐스트되었지만 아직 블록체인에서 확인되지 않은 트랜잭션이 풀노드에 대기하는 영역.

함께 보기

- [Confirmation](#)

- [Transaction](#)
-

Metadata (메타데이터)

다른 데이터에 대한 정보를 제공하는 데이터. Cell의 capacity, type, lock 은 메타데이터이며, Cell 용량을 점유하고 상태 비용을 발생시킨다.

함께 보기

- [Capacity](#)
 - [Cell](#)
 - [Lock](#)
 - [Type Script](#)
-

Micro-State (마이크로 상태)

네트워크의 전체 상태를 알 필요 없이 독립적으로 조작할 수 있는, 격리된 작은 상태 조각.

Nervos에서는 마이크로 상태가 Cell로 표현된다.

동의어

- [Cell](#)

함께 보기

- [State](#)
-

Miner (채굴자)

반복적인 해싱을 통해 새 블록에 대한 유효한 작업증명을 찾는 네트워크 노드.

함께 보기

- [Proof of Work](#)
-

Miner Fee (채굴자 수수료)

트랜잭션 수수료의 다른 표현.

동의어

- [Transaction Fee](#)
-

Mining (채굴)

블록체인 노드가 새로운 트랜잭션을 검증하고, 유효한 작업증명을 찾아 새 블록을 생성함으로써 새로운 토큰 보상을 얻는 과정.

함께 보기

- [Block](#)
 - [Blockchain](#)
 - [Mining Reward](#)
 - [Proof of Work](#)
-

Mining Node (채굴 노드)

계산 퍼즐을 해결하여 새 블록을 생성한다. 블록을 적극적으로 검증·추가함으로써 네트워크의 보안과 합의에 기여한다.

```
ckb init --chain Mainnet && ckb miner
```

함께 보기

- [Block](#)
 - [Consensus](#)
 - [Miner](#)
 - [Mining](#)
 - [Proof of Work](#)
 - [Full Node](#)
 - [Light Node](#)
-

Mining Reward (채굴 보상)

마이닝에 필요한 컴퓨팅 자원을 제공한 대가로 채굴자에게 지급되는 네이티브 토큰.

함께 보기

- [Miner](#)

- [Mining](#)
 - [Native Token](#)
-

Multisig (멀티시그)

다중 서명의 약자. 여러 개의 고유한 서명을 결합하여 생성되는 특정 유형의 디지털 서명.

Native Token (네이티브 토큰)

블록체인의 합의 노드에 보상으로 발행되는 토큰. Nervos CKB의 네이티브 토큰은 CKByte이다.

함께 보기

- [CKByte](#)
 - [Token](#)
-

NC-MAX

Nervos CKB의 합의 알고리즘으로, NC의 백본 프로토콜을 따른다. 주요 혁신은 2단계 트랜잭션 확인 메커니즘이다.

함께 보기

- [Consensus](#)
 - [Nervos Blockchain](#)
 - [NC-MAC](#)
-

Neighbor (이웃)

블록체인 P2P 네트워크에서 다른 노드와 직접 연결된 노드.

함께 보기

- [Node](#)
 - [Peer to Peer](#)
-

Nervos Blockchain

Common Knowledge Base로 알려진 Nervos 네트워크의 레이어 1 블록체인.

동의어

- [Common Knowledge Base](#)
 - [Layer 1](#)
 - [Nervos CKB](#)
-

Nervos DAO

Nervos DAO는 사용자가 CKBytes를 잠가 Nervos CKB의 세컨더리 발행으로부터 보상을 받을 수 있게 한다. 이 과정은 다른 플랫폼의 스테이킹과 유사하다. Nervos DAO는 CKByte 보유자에게 인플레이션으로부터 보호하는 “가상 하드캡”을 제공한다.

함께 보기

- [CKByte](#)
 - [DAO](#)
 - [Secondary Issuance](#)
 - [Nervos DAO Explained on the Nervos Blog](#)
 - [Nervos DAO in RFC on Nervos Network Github](#)
 - [Inflation Rate Chart](#)
-

Network Hash Rate (네트워크 해시레이트)

네트워크 보안을 위해 할당된 총 계산 처리 능력의 측정값.

함께 보기

- [Hash Rate](#)
 - [Miner](#)
-

Node (노드)

네트워크에 참여하는 소프트웨어 클라이언트.

함께 보기

- [Full Node](#)
- [Light Node](#)
- [Mining Node](#)

Nonce (논스)

암호학에서 한 번만 사용할 수 있는 값. 블록체인 문맥에서 두 가지를 가리킬 수 있다: 1) 작업증명 논스는 블록에서 PoW 요구사항을 만족하는 임의 값; 2) 계정 논스는 각 계정의 트랜잭션 카운터로, 재생 공격을 방지하는 데 사용된다.

함께 보기

- [Block](#)
 - [Proof of Work](#)
-

Non-Fungible Token (비대체형 토큰)

NFT는 서로를 구분하는 고유 식별 코드와 메타데이터를 가진 블록체인상의 암호화 자산.

함께 보기

- [DOB](#)
 - [Fungible Token](#)
 - [Token](#)
 - [User-Defined Token](#)
-

Open Source (오픈소스)

소스 코드를 제3자가 자유롭게 검토하거나 수정할 수 있도록 공개된 소프트웨어.

함께 보기

- [What is Open Source at OpenSource.com](#)
-

P2P

peer to peer의 약칭.

동의어

- [Peer to Peer](#)
-

Paper Wallet (페이퍼 월렛)

복구 구문 또는 개인 키를 종이에 인쇄하여 오프라인으로 보관하는 형태. 이 문서는 금고 같은 사용자가 선택한 안전한 장소에 전통적 방식으로 보관된다.

함께 보기

- [Private Key](#)
 - [Wallet](#)
-

Payment Address (결제 주소)

암호화폐와 자산을 주고받을 수 있는 문자숫자 문자열.

Nervos CKB 메인넷 주소는 항상 “ckb” 접두사로 시작한다.

동의어

- [Address](#)
-

Payment Channel (결제 채널)

마이크로페이먼트/결제 채널은 사용자들이 모든 트랜잭션을 레이어 1 블록체인에 커밋하지 않고도 여러 결제를 수행할 수 있게 설계된 기법의 일종이다. 일반적인 결제 채널에서는 블록체인에 단 두 개의 트랜잭션만 추가되지만, 참여자 간에는 무제한 또는 거의 무제한의 결제가 가능하다.

함께 보기

- [Layer 1](#)
-

Peer to Peer (피어 투 피어)

중앙화된 제3자 없이, 두 개인이 직접 상호작용하는 탈중앙 플랫폼.

동의어

- [P2P](#)

함께 보기

- [Node](#)
-

Private Key (개인 키)

시크릿 키로도 알려진 개인 키는 소유자만 아는 암호학적 변수로, 알고리즘과 함께 사용되어 데이터를 암호화·복호화한다.

함께 보기

- [Digital Asset](#)
 - [Paper Wallet](#)
 - [Payment Address](#)
 - [Wallet](#)
-

Proof of Work (작업증명)

합의에 참여하기 위해 일정량의 계산 자원 소유를 증명하도록 암호 퍼즐을 해결하게 하는 방식. 일반적으로 PoW는 PoS보다 더 무허가형(permissionless) 합의 메커니즘이다.

널리 퍼진 오해와 달리, PoW는 에너지 “낭비”가 아니며 탄소 배출을 더 유발하지 않는다. PoW는 Nervos 레이어 1 블록체인 CKB에서 사용된다.

함께 보기

- [Block](#)
 - [Consensus](#)
 - [Mining Reward](#)
 - [Proof of Stake](#)
 - [Transaction](#)
-

Proof of Stake (지분증명)

합의에 참여하기 위해 일정량의 암호화폐(네트워크에 대한 “지분”) 소유를 증명하도록 요구한다. PoS는 롱 레인지 공격과 같은 해결 불가능한 이슈로 인해 약한 주관성(weak-subjectivity)에 의존한다. PoS 시스템에서는 미래의 합의 정족수가 기존 참여자에 의해 전적으로 결정된다. PoS는 Nervos 네트워크의 레이어 2 프로토콜에서 사용된다.

함께 보기

- [Consensus](#)
 - [Mining Reward](#)
 - [Proof of Work](#)
 - [Transaction](#)
-

Propagation (전파)

Block Propagation의 약칭.

동의어

- [Block Propagation](#)
-

Public Key (공개 키)

공개키 암호(비대칭 암호)에서만 사용되는 개념. 공개 키는 보안에 영향을 주지 않고 타인이 알아도 되는 정보이다. 각 사용자는 고유한 공개 키를 가지며, 사용자만 아는 개인 키와 연결되어 있다. 공개 키는 대응되는 개인 키로만 복호화할 수 있도록 메시지를 암호화하거나, 대응되는 개인 키로 승인된 메시지인지 검증하는 데 사용할 수 있다.

함께 보기

- [Private Key](#)
-

Reward (보상)

네트워크가 새로운 블록마다, 작업증명을 발견한 채굴자에게 지급하는 CKBytes의 양.

함께 보기

- [CKByte](#)
 - [Base Reward](#)
 - [Block Reward](#)
 - [Commit Reward](#)
 - [Mining Reward](#)
 - [Proposal Reward](#)
 - [Secondary Reward](#)
-

Signature (서명)

암호학적 서명의 약칭.

동의어

- [Cryptographic Signature](#)
-

Smart Contract (스마트 컨트랙트)

계약 작성자와 사용자 간 합의 조건이 코드로 직접 작성되어 자동으로 실행되는 계약. 해당 코드와 합의는 분산·탈중앙 블록체인 네트워크 전반에 존재한다. Nervos CKB에서는 Script라고도 한다.

State (상태)

블록체인에 저장된 데이터. 대부분의 문맥에서 이는 현재 데이터를 의미하며, 역사적 데이터는 제외한다.

함께 보기

- [Blockchain](#)
-

State Bloat (상태 팽창)

이더리움에서 상태 데이터가 무제한 증가하는 현상. 상태 팽창은 노드 동기화를 느리게 하고, 풀노드의 진입 장벽을 높이며, 결국 네트워크 탈중앙화를 저해한다.

State Channel (상태 채널)

참여자 간 채널을 설정하여 자유롭게 저렴하게 거래할 수 있는 레이어 2 솔루션. 채널 설정과 종료 트랜잭션만 메인넷에 전송된다. 매우 높은 트랜잭션 처리량을 가능하게 하지만, 사전에 참여자 수를 알아야 하고 자금을 잠가야 한다.

Tip (팁)

tip block의 약칭.

동의어

- [Tip Block](#)
-

Tip Block (팁 블록)

블록체인에서 가장 최근에 확인된 블록. 팁 블록은 블록체인에서 가장 높은 블록 높이를 가진다.

동의어

- [Tip](#)

함께 보기

- [Block](#)
 - [Block Height](#)
 - [Blockchain](#)
-

Transaction (트랜잭션)

분산 원장과 상호작용하기 위해 사용자가 생성하고 서명하는 기본 객체. 트랜잭션은 사용자의 요청에 따라 원장 상태를 갱신한다. CKB 트랜잭션은 이전 트랜잭션에서 생성된 일부 출력을 파기하고 새 출력을 생성한다. CKB에서는 트랜잭션 출력을 Cell이라고 부른다.

함께 보기

- [Blockchain](#)
 - [Nervos Blockchain](#)
 - [UTXO](#)
-

Transaction Fee (트랜잭션 수수료)

트랜잭션을 처리해 준 대가로 채굴자에게 네이티브 토큰으로 지불되는 수수료.

동의어

- [Miner Fee](#)

함께 보기

- [Miner](#)
 - [Native Token](#)
 - [Transaction](#)
-

Token (토큰)

스마트 컨트랙트 플랫폼의 비네이티브 토큰을 종종 가리키는 말로, Nervos 네트워크의 UDT나 이더리움의 ERC20 등이 이에 해당한다.

함께 보기

- [Blockchain](#)
- [Digital Asset](#)
- [Transaction](#)

Turing Complete (튜링 완전)

충분한 시간과 메모리, 필요한 명령이 주어진다면 어떤 계산 문제든 해결할 수 있는 기계를 의미한다. 현대 프로그래밍 언어(C++, Python, JavaScript 등) 대부분을 설명할 때 흔히 사용된다.

UDT

User-Defined Token의 약자. 사용자가 정의한 속성을 가진 맞춤 토큰. 일반적 사용에서는 대체형 토큰을 가리키는 경우가 많다.

동의어

- [User-Defined Token](#)
-

Unconfirmed (미확인)

아직 확인되지 않은 트랜잭션의 상태. 미확인 트랜잭션은 최종적이지 않으며 보장될 수 없다.

동의어

- [Unconfirmed Transaction](#)

함께 보기

- [Confirmation](#)
 - [Transaction](#)
-

Unconfirmed Transaction (미확인 트랜잭션)

아직 확인되지 않은 트랜잭션. 미확인 트랜잭션은 최종적이지 않으며 보장될 수 없다.

동의어

- [Unconfirmed](#)

함께 보기

- [Confirmation](#)
 - [Transaction](#)
-

User-Defined Token (사용자 정의 토큰)

사용자가 정의한 속성을 가진 맞춤 토큰. 일반적 사용에서는 대체형 토큰을 가리키는 경우가 많다.

User-Defined Token은 보통 약어 UDT로 지칭된다.

동의어

- [Fungible Token](#)
- [UDT](#)

함께 보기

- [ERC20 on Ethereum.org](#)
 - [ERC777 on Ethereum.org](#)
 - [Simple UDT Draft Spec on Nervos Talk](#)
-

UTXO

Unspent Transaction Output의 약어로, 트랜잭션 이후 남아 미래에 사용 가능한 토큰의 양을 뜻한다. 트랜잭션은 기존 UTXO를 입력으로 소비하고, 각 UTXO는 한 번만 소비될 수 있는 새로운 UTXO를 출력으로 생성한다.

함께 보기

- [Transaction](#)
 - [The UTXO Model Explained](#)
-

Wallet (지갑)

자산, 스마트 컨트랙트, dApp 등 온체인 개체와 상호작용하는 데 사용되는 사용자 지향 소프트웨어. 지갑은 자체적으로 키 관리를 포함할 수도 있고, 보안을 위해 외부 하드웨어에 키 관리를 위임할 수도 있다.

함께 보기

- [Paper Wallet](#)
 - [Private Key](#)
 - [Payment Address](#)
 - [Digital Asset](#)
-

경제 용어집

Base Issuance (베이스 발행)

베이스 발행은 고정되어 점차 감소하는 일정에 따른 기본 CKByte 발행을 말한다. 베이스 발행은 네트워크 보호에 대한 인센티브이자 간접적인 토큰 분배 방식으로 채굴자에게 지급된다.

베이스 발행은 총 공급량 33.6G(336억) CKBytes로 한정된다.

함께 보기

- [Base Reward](#)
 - [CKByte](#)
 - [Secondary Issuance](#)
 - [Miner](#)
 - [Crypto-Economics RFC on Nervos Network GitHub](#)
-

Base Reward (베이스 보상)

베이스 보상은 베이스 발행에서 발생하여 채굴자에게 지급되는 블록 보상(CKBytes 단위)이다. 베이스 보상은 비트코인처럼 약 4년마다 반감되며, 최종적으로 0에 도달한다.

함께 보기

- [Base Issuance](#)
 - [CKByte](#)
 - [Miner](#)
 - [Transaction](#)
-

Commit Reward (커밋 보상)

이전에 제안된 트랜잭션을 커밋한 대가로 채굴자에게 CKBytes로 지급되는 보상이다. 트랜잭션이 커밋되면 확인(confirmed)된다.

함께 보기

- [CKByte](#)
 - [Confirmation](#)
 - [Miner](#)
 - [Transaction](#)
-

Economic Abstraction (경제적 추상화)

적절한 도구 지원이 있을 경우, 사용자는 트랜잭션 수수료를 CKByte가 아닌 다른 토큰(예: 스테이블 코인)으로 지불할 수 있는데, 이를 “경제적 추상화(Economic Abstraction)”라고 한다.

Fiat Currency (법정통화)

법정통화는 종종 정부 규정에 의해 화폐로서의 지위를 부여받은 교환 매개체이다. 법정화폐는 내재적 가치나 사용 가치를 갖지 않는다. 정부가 그 가치를 유지하거나, 교환에 참여하는 당사자들이 그 가치에 합의하기 때문에 가치가 존재한다.

함께 보기

- [Cryptocurrency](#)
 - [Digital Currency](#)
-

Heavy Asset Problem (무거운 자산 문제)

다자산 블록체인 플랫폼에서 흔히 발견되는 문제로, 체인에 저장된 자산의 가치가 크게 상승하지만 체인의 네이티브 토큰 가치는 그렇지 못한 경우를 말한다. 이는 네트워크 공격에 대한 유인을 높이지만, 네이티브 토큰의 가치가 네트워크 보안을 담당하므로 보안을 강화하지는 못한다.

함께 보기

- [Asset](#)
 - [Starving Layer 1 Problem](#)
 - [Tragedy of the Security Commons](#)
-

Liquidity (유동성)

자산을 현재 시장 가격에 큰 변동을 일으키지 않고도 쉽게 사고팔 수 있는 능력.

함께 보기

- [Asset](#)
-

Proposal Reward (프로포절 보상)

미확인 트랜잭션을 제안한 대가로 채굴자에게 CKBytes로 지급되는 보상.

함께 보기

- [CKByte](#)
 - [Confirmation](#)
 - [Miner](#)
 - [Transaction](#)
-

Secondary Issuance (세컨더리 발행)

세컨더리 보상을 통해 채굴자에게 지급되는 새로운 CKBytes의 생성이다. 세컨더리 발행은 연간 13.44억 CKBytes의 고정 인플레이션 일정을 따른다. Nervos DAO 스테이커는 세컨더리 발행의 영향을 받지 않는다.

함께 보기

- [Base Issuance](#)
 - [CKByte](#)
 - [Nervos DAO](#)
 - [Secondary Reward](#)
 - [State](#)
 - [Crypto-Economics RFC on Nervos Network GitHub](#)
-

Secondary Reward (세컨더리 보상)

트랜잭션 처리와 데이터 영속화에 필요한 연산 및 저장 자원을 제공한 대가로 채굴자에게 CKBytes로 지급되는 보조금이다.

세컨더리 보상은 세컨더리 발행에서 생성되며, 트랜잭션 검증과 블록체인 상태 보존에 대해 채굴자에게 지속적으로 지급된다.

함께 보기

- [CKByte](#)
 - [Miner](#)
 - [Secondary Issuance](#)
 - [Transaction](#)
-

Selfish Mining Attack (이기적 채굴 공격)

2013년 코넬대학교 연구자들이 상세히 다룬 개념으로, 악의적 채굴자가 다른 이가 채굴한 블록을 의도적으로 고아 블록으로 만들어 불공정한 블록 보상을 얻는 공격이다.

함께 보기

- [Selfish Mining Related Paper](#)
-

Starving Layer 1 Problem (레이어 1 고사 문제)

다중 레이어 블록체인 플랫폼에서, 대부분의 트랜잭션 트래픽이 레이어 1에서 레이어 2로 이동하면서 수수료 역시 대부분 이동하는 상황을 말한다. 레이어 1이 플랫폼 보안을 전적으로 트랜잭션 수수료에 의존하는 경우, 적절한 보안을 유지하기에 충분한 인센티브가 부족해질 수 있다.

함께 보기

- [Heavy Asset Problem](#)
 - [Layer 1](#)
 - [Layer 2](#)
 - [Transaction](#)
 - [Transaction Fee](#)
-

State Rent (상태 렌트)

상태 데이터를 지속·보호하기 위해 반복적으로 지불되는 수수료.

Nervos에서는, Nervos 블록체인 공간을 점유하는 사용자의 상태 렌트 지불을 강화하기 위해 세컨더리 발행이 사용된다.

함께 보기

- [Secondary Issuance](#)
 - [Nervos Blockchain](#)
 - [Crypto-Economics RFC on Nervos Network GitHub](#)
-

Store of Assets (자산 저장소)

비트코인 맥락의 “Store of Value(가치 저장 수단)” 개념과 유사하게, 블록체인이 어떤 암호 자산이든 안전하고 검열 저항적으로 보관할 때 그 효용을 “자산 저장소(Store of Assets)”라고 부른다. Nervos CKB는 이러한 자산 저장소(SoA)에 해당한다.

함께 보기

- [Store of Value](#)
-

Store of Value (가치 저장 수단)

시간이 지나도 가치가 감소하지 않고 유지될 수 있는 자산.

좋은 가치 저장 수단은 법정통화의 인플레이션율을 따라가거나 능가해야 하며, 적절한 유동성을 가져 쉽게 매도할 수 있어야 한다.

함께 보기

- [Store of Assets](#)
 - [Liquidity](#)
-

Tail Emission (테일 발행)

고정량 인플레이션을 통해 채굴자에게 지급되는 보상의 한 종류.

함께 보기

- [Secondary Reward](#)
-

Targeted Inflation (표적화 인플레이션)

특정 사용자 집합에만 영향을 미치는 형태의 인플레이션.

Nervos는 상태 렌트를 지불하기 위해 블록체인 공간을 점유하는 사용자에게 표적화 인플레이션을 유도하기 위해 세컨더리 발행을 사용한다. CKBytes의 장기 보유자는 Nervos DAO에 예치하여 인플레이션 방어막으로 활용할 수 있다.

함께 보기

- [CKByte](#)
 - [Secondary Issuance](#)
 - [Nervos Blockchain](#)
 - [Nervos DAO](#)
 - [Crypto-Economics RFC on Nervos Network GitHub](#)
-

Tragedy of the Commons (공유지의 비극)

참여자들이 각자의 이익에 따라 행동한 결과, 공동의 자원을 집단적으로 고갈시키거나 파괴하는 시스템상의 상황.

함께 보기

- [Tragedy of the Security Commons](#)
 - [Tragedy of the Storage Commons](#)
-

Tragedy of the Security Commons (보안 공유지의 비극)

다자산 블록체인 플랫폼에서 발생할 수 있는 상황으로, 자산 토큰이 블록체인 플랫폼의 저장 및 보안에 의존하지만 플랫폼에 대한 기여는 하지 않는 경우를 말한다. “무임승차”하는 자산이 늘어날수록 기반 블록체인 플랫폼에 가해지는 부담도 증가한다. 자산이 기반 플랫폼에 기여하지 않으면, 이용 가능한 보안이 네트워크를 적절히 뒷받침하지 못할 수 있다.

함께 보기

- [Heavy Asset Problem](#)
 - [Tragedy of the Commons](#)
 - [Tragedy of the Storage Commons](#)
-

Tragedy of the Storage Commons (저장 공유지의 비극)

인센티브가 있는 블록체인 플랫폼에서, 마이닝 보상이 블록체인에 데이터를 포함한 것에 대해 지급되지만, 블록체인 데이터를 장기간 보존하는 것에 대해서는 보상이 존재하지 않는 경우 발생할 수 있는 상황이다. 체인의 크기가 커질수록 데이터를 보존하는 데 수반되는 비용도 커진다. 데이터 보존에 대한 직접 인센티브가 없으면, 이를 수행하는 노드는 점점 줄어들다. 결국 네트워크를 적절히 지원할 수 있는 노드 수가 부족해질 수 있다.

함께 보기

- [Tragedy of the Commons](#)
 - [Tragedy of the Security Commons](#)
-

기술 용어집

Active Cell (활성 셀)

또는 Live Cell이라 하며, 현재 CKB 상태에 존재하는 셀이다. 활성 셀만 새 트랜잭션의 입력으로 사용할 수 있다.

동의어

- [Live Cell](#)

함께 보기

- [Cell](#)
 - [Input](#)
 - [Transaction](#)
-

Aggron

Mainnet Lina에 대응하는 첫 번째 Nervos CKB 테스트넷.

- **CKB 버전:** $\geq v0.101.0$ (최신 안정 버전 권장)
- **제네시스 해시:**
0x10639e0895502b5688a6be8cf69460d76541bfa4821629d86d62ba0aae3f9606
- **초기화 명령어:** `ckb init --chain testnet`
- **출시 시점:** 2020-05-22 04:00:00 UTC
- **ckb2021 활성화 시점:** 2021-10-24 03:00:00 UTC

동의어

- [Testnet](#)

혼동하지 말 것

- [Lina](#)
 - [Mainnet](#)
-

Animagus

Nervos CKB 위에서 동작하는 프레임워크 레이어로, 셀 수집 과정을 거치지 않고도 계정 잔액을 쉽게 조회할 수 있도록 한다.

함께 보기

- [Cell Collection](#)
- [Nervos CKB](#)

- [Animagus 소개 블로그](#)
-

Args (인자)

Arguments의 약어. 인자는 함수 호출 시 전달되는 인자와 유사하게, 셀의 Lock Script나 Type Script에 제공되는 데이터이다.

인자는 셀이 생성될 때 셀의 일부로 저장된다.

함께 보기

- [Cell](#)
 - [Lock Script](#)
 - [Type Script](#)
-

Axon

실용적 보안 및 경제 모델을 갖춘 체인 기반 레이어 2 프로토콜 및 프레임워크. Axon 체인은 누구나 CKB에서 토큰을 스테이킹하여 검증자가 되고 합의에 참여할 수 있게 한다.

함께 보기

- [Layer 2](#)
 - [Nervos CKB](#)
-

Blake2b

암호 해시 함수. BLAKE2b는 64비트 플랫폼(NEON을 지원하는 ARM 포함)에 최적화되어 있으며, 1~64바이트 길이의 다이제스트를 생성한다. CKB는 기본 해시 알고리즘으로 BLAKE2b를 사용한다.

함께 보기

- [Blake2b 논문](#)
 - [Ckbhash](#)
 - [Blake 해시 함수 위키백과](#)
 - [Hash Function on Wikipedia](#)
-

Block Subsidy (블록 보조금)

채굴자가 블록을 생성하고 블록체인을 보호하기 위해 제공한 컴퓨팅 자원에 대해 네이티브 통화로 지급되는 보상.

보조금은 블록 생성 시 인플레이션으로 발행된 블록 보상의 일부이며, 추가 트랜잭션 수수료는 포함하지 않는다.

동의어

- [Block Reward](#)
 - [Transaction Fee](#)
-

Boxer

Nervos 레이어 1 블록체인(Common Knowledge Base)을 검증하기 위한 경량 Rust 라이브러리.

함께 보기

- [Common Knowledge Base](#)
 - [Boxer on GitHub](#)
-

block_version

블록의 버전. 시스템 예약 필드이며, 기본값은 0이다.

```
pub const BLOCK_VERSION: Version = 0;
```

함께 보기

- [Block](#)
-

Cell Collection (셀 수집)

특정 조건을 만족하는 셀을 모으는 과정.

예: 특정 계정의 잔액을 확인하기 위해, 해당 주소의 모든 활성 셀을 수집해야 한다.

함께 보기

- [Cell](#)
-

Cellbase (셀베이스)

각 블록에서 새로운 CKBytes를 발행하는 역할을 하는 트랜잭션.

비트코인의 코인베이스 트랜잭션과 동일한 개념이다.

함께 보기

- [CKByte](#)
- [Coinbase on Bitcoin.org](#)

cellbase_maturity

트랜잭션에서 참조되는 셀베이스 출력은 특정 조건을 충족해야 하며, 그렇지 않으면 트랜잭션이 거부된다. 셀베이스 출력은 “잠금” 상태이며, 소비되기 전 약 4 에포크(약 16시간)를 기다려야 한다. 이는 소프트 포크 발생 시 셀베이스 루트를 가진 트랜잭션이 롤백되는 위험을 방지하기 위한 제한이다.

```
pub(crate) const CELLBASE_MATURITY: EpochNumberWithFraction =  
EpochNumberWithFraction::new_unchecked(4, 0, 1);
```

함께 보기

- [Cellbase](#)

cell_deps

트랜잭션 내 스크립트가 참조된 Live Cell에 접근(읽기 전용)할 수 있도록 하는 체인 상 Live Cell의 포인터.

자세한 내용은 [Script dependencies](#) 참조.

함께 보기

- [Dependencies](#)

Ckhash

CKB에서 사용되는 기본 해시 알고리즘은 blake2b이며, 다음과 같이 구성된다:

- 출력 다이제스트 크기: 32
- personalization: ckb-default-hash

ckbhash는 위 설정을 적용한 blake2b 해시를 의미한다.

```
import hashlib  
def ckbhash():  
    return hashlib.blake2b(digest_size=32, person=b'ckb-default-hash')
```

함께 보기

- [Blake2b](#)
-

CKB Merkle Tree (CKB 머클 트리)

CKB 머클 트리는 다음 병합 함수를 사용하는 CBMT(Complete Binary Merkle Tree)이다:

```
ckbhash(left || right)
```

ckbhash 는 해시 함수이며, `||` 는 이진 연결을 의미한다.

함께 보기

- [Merkle Tree for Static Data](#)
 - [Ckbhash](#)
 - [Merkle Tree on Wikipedia](#)
-

Code Hash (코드 해시)

특정 데이터 또는 Type ID로 참조되는 특정 셀을 가리키는 해시 값을 포함하는 셀의 필드.

함께 보기

- [Cell](#)
 - [Data](#)
 - [Type ID](#)
-

Commit (커밋)

제안된 트랜잭션을 블록체인에 추가하는 과정. 트랜잭션이 커밋되면 확인된다.

채굴자는 커밋 보상(Commit Reward)을 통해 커밋 인센티브를 얻는다.

함께 보기

- [Commit Reward](#)
 - [Confirmation](#)
 - [Propose](#)
 - [Transaction](#)
-

Commitment Zone (커밋 존)

트랜잭션 커밋이 포함되는 블록의 섹션.

커밋 존에는 이전 2~10개의 블록의 제안 존(Proposal Zone)에 나타난 유효한 트랜잭션만 포함될 수 있다.

함께 보기

- [Block](#)
 - [Proposal Zone](#)
 - [Transaction](#)
-

Consume (소비)

Live Cell을 트랜잭션 입력으로 사용하는 과정.

소비된 Live Cell은 Dead Cell로 표시된다.

이는 비트코인에서 UTXO를 '소비됨(spent)'으로 표시하는 것과 동일하다.

함께 보기

- [Cell](#)
 - [Cell Model](#)
 - [Dead Cell](#)
 - [Live Cell](#)
 - [UTXO on Bitcoin.org](#)
-

기술 용어집

Active Cell (활성 셀)

또는 Live Cell이라 하며, 현재 CKB 상태에 존재하는 셀이다. 활성 셀만 새 트랜잭션의 입력으로 사용할 수 있다.

동의어

- [Live Cell](#)

함께 보기

- [Cell](#)

- [Input](#)
 - [Transaction](#)
-

Aggron

Mainnet Lina에 대응하는 첫 번째 Nervos CKB 테스트넷.

- **CKB 버전:** `>= v0.101.0` (최신 안정 버전 권장)
- **제네시스 해시:**
`0x10639e0895502b5688a6be8cf69460d76541bfa4821629d86d62ba0aae3f9606`
- **초기화 명령어:** `ckb init --chain testnet`
- **출시 시점:** 2020-05-22 04:00:00 UTC
- **ckb2021 활성화 시점:** 2021-10-24 03:00:00 UTC

동의어

- [Testnet](#)

혼동하지 말 것

- [Lina](#)
 - [Mainnet](#)
-

Animagus

Nervos CKB 위에서 동작하는 프레임워크 레이어로, 셀 수집 과정을 거치지 않고도 계정 잔액을 쉽게 조회할 수 있도록 한다.

함께 보기

- [Cell Collection](#)
 - [Nervos CKB](#)
 - [Animagus 소개 블로그](#)
-

Args (인자)

Arguments의 약어. 인자는 함수 호출 시 전달되는 인자와 유사하게, 셀의 Lock Script나 Type Script에 제공되는 데이터이다.

인자는 셀이 생성될 때 셀의 일부로 저장된다.

함께 보기

- [Cell](#)
 - [Lock Script](#)
 - [Type Script](#)
-

Axon

실용적 보안 및 경제 모델을 갖춘 체인 기반 레이어 2 프로토콜 및 프레임워크. Axon 체인은 누구나 CKB에서 토큰을 스테이킹하여 검증자가 되고 합의에 참여할 수 있게 한다.

함께 보기

- [Layer 2](#)
 - [Nervos CKB](#)
-

Blake2b

암호 해시 함수. BLAKE2b는 64비트 플랫폼(NEON을 지원하는 ARM 포함)에 최적화되어 있으며, 1~64바이트 길이의 다이제스트를 생성한다. CKB는 기본 해시 알고리즘으로 BLAKE2b를 사용한다.

함께 보기

- [Blake2b 논문](#)
 - [Ckbhash](#)
 - [Blake 해시 함수 위키백과](#)
 - [Hash Function on Wikipedia](#)
-

Block Subsidy (블록 보조금)

채굴자가 블록을 생성하고 블록체인을 보호하기 위해 제공한 컴퓨팅 자원에 대해 네이티브 통화로 지급되는 보상.

보조금은 블록 생성 시 인플레이션으로 발행된 블록 보상의 일부이며, 추가 트랜잭션 수수료는 포함하지 않는다.

동의어

- [Block Reward](#)
 - [Transaction Fee](#)
-

Boxer

Nervos 레이어 1 블록체인(Common Knowledge Base)을 검증하기 위한 경량 Rust 라이브러리.

함께 보기

- [Common Knowledge Base](#)
 - [Boxer on GitHub](#)
-

block_version

블록의 버전. 시스템 예약 필드이며, 기본값은 0이다.

```
pub const BLOCK_VERSION: Version = 0;
```

함께 보기

- [Block](#)
-

Cell Collection (셀 수집)

특정 조건을 만족하는 셀을 모으는 과정.

예: 특정 계정의 잔액을 확인하기 위해, 해당 주소의 모든 활성 셀을 수집해야 한다.

함께 보기

- [Cell](#)
-

Cellbase (셀베이스)

각 블록에서 새로운 CKBytes를 발행하는 역할을 하는 트랜잭션.

비트코인의 코인베이스 트랜잭션과 동일한 개념이다.

함께 보기

- [CKByte](#)
 - [Coinbase on Bitcoin.org](#)
-

cellbase_maturity

트랜잭션에서 참조되는 셀베이스 출력은 특정 조건을 충족해야 하며, 그렇지 않으면 트랜잭션이 거부된다. 셀베이스 출력은 “잠금” 상태이며, 소비되기 전 약 4 에포크(약 16시간)를 기다려야 한다. 이는 소프트 포크 발생 시 셀베이스 루트를 가진 트랜잭션이 롤백되는 위험을 방지하기 위한 제한이다.

```
pub(crate) const CELLBASE_MATURITY: EpochNumberWithFraction =  
    EpochNumberWithFraction::new_unchecked(4, 0, 1);
```

함께 보기

- [Cellbase](#)

cell_deps

트랜잭션 내 스크립트가 참조된 Live Cell에 접근(읽기 전용)할 수 있도록 하는 체인 상 Live Cell의 포인터.

자세한 내용은 [Script dependencies](#) 참조.

함께 보기

- [Dependencies](#)

Ckbhash

CKB에서 사용되는 기본 해시 알고리즘은 blake2b이며, 다음과 같이 구성된다:

- 출력 다이제스트 크기: 32
- personalization: ckb-default-hash

ckbhash는 위 설정을 적용한 blake2b 해시를 의미한다.

```
import hashlib  
def ckbhash():  
    return hashlib.blake2b(digest_size=32, person=b'ckb-default-hash')
```

함께 보기

- [Blake2b](#)

CKB Merkle Tree (CKB 머클 트리)

CKB 머클 트리는 다음 병합 함수를 사용하는 CBMT(Complete Binary Merkle Tree)이다:

```
ckbhash(left || right)
```

ckbhash 는 해시 함수이며, || 는 이진 연결을 의미한다.

함께 보기

- [Merkle Tree for Static Data](#)
- [Ckbhash](#)
- [Merkle Tree on Wikipedia](#)

Code Hash (코드 해시)

특정 데이터 또는 Type ID로 참조되는 특정 셀을 가리키는 해시 값을 포함하는 셀의 필드.

함께 보기

- [Cell](#)
- [Data](#)
- [Type ID](#)

Commit (커밋)

제안된 트랜잭션을 블록체인에 추가하는 과정. 트랜잭션이 커밋되면 확인된다.

채굴자는 커밋 보상(Commit Reward)을 통해 커밋 인센티브를 얻는다.

함께 보기

- [Commit Reward](#)
- [Confirmation](#)
- [Propose](#)
- [Transaction](#)

Commitment Zone (커밋 존)

트랜잭션 커밋이 포함되는 블록의 섹션.

커밋 존에는 이전 2~10개의 블록의 제안 존(Proposal Zone)에 나타난 유효한 트랜잭션만 포함될 수 있다.

함께 보기

- [Block](#)
 - [Proposal Zone](#)
 - [Transaction](#)
-

Consume (소비)

Live Cell을 트랜잭션 입력으로 사용하는 과정.

소비된 Live Cell은 Dead Cell로 표시된다.

이는 비트코인에서 UTXO를 '소비됨(spent)'으로 표시하는 것과 동일하다.

함께 보기

- [Cell](#)
 - [Cell Model](#)
 - [Dead Cell](#)
 - [Live Cell](#)
 - [UTXO on Bitcoin.org](#)
-

CPFP

Child Pays For Parent의 약자. 막힌 트랜잭션의 출력을 소비하는 자식 트랜잭션이 더 높은 수수료를 사용하여, 채굴자가 부모 트랜잭션과 함께 처리하도록 유도하는 트랜잭션 대체 방식이다. 이는 혼잡 시 신속한 처리에 도움이 된다.

함께 보기

- [Fee Bumping](#)
 - [RBF](#)
-

Crypto Primitives

암호 프로토콜을 구축할 때 일반적으로 사용되는, 검증된 저수준 암호 알고리즘들.

함께 보기

- [Cryptographic Primitive on Wikipedia](#)
-

dao_type_hash

NervosDAO의 type_hash.

자세한 내용은 [CKB Genesis Script List](#)를 참조.

함께 보기

- [Type Script Hash](#)
-

Data

Nervos의 특정 문맥에서, data는 셀 내부의 데이터 구조를 가리킨다. 이 구조는 Nervos 블록체인에 저장되어야 하는 임의의 정보를 보관하는 데 사용된다.

보다 일반적인 문맥에서, data는 어떤 형태의 정보든 지칭할 수 있다.

함께 보기

- [Cell](#)
 - [Cell Model](#)
-

Dead Cell

이전에 트랜잭션의 입력으로 사용되어 소비된 셀.

Dead Cell은 새로운 트랜잭션의 입력으로 사용할 수 없고, 의존성으로도 사용할 수 없다. 사실상 네트워크의 활성 상태에서 제거·파기된 것으로 간주된다.

Dead Cell은 비트코인의 “spent UTXO”에 해당한다.

동의어

- [Historical Cell](#)

함께 보기

- [Cell](#)
 - [Cell Model](#)
 - [Consume](#)
 - [Transaction](#)
 - [UTXO on Bitcoin.org](#)
-

Dep Group

여러 개의 의존성을 하나의 의존성 필드로 묶어, 자주 함께 사용하는 의존성들을 단일 참조로 지정할 수 있게 하는 방법.

함께 보기

- [Dep Type](#)
 - [Dependencies](#)
 - [CKB Transaction Structure on GitHub](#)
-

Dep Type

의존성의 유형을 지정하는 필드.

함께 보기

- [Dep Group](#)
 - [Dependencies](#)
 - [CKB Transaction Structure on GitHub](#)
-

Deps

dependencies의 약칭.

동의어

- [Dependencies](#)
-

Dependencies

일반적으로 deps라고도 부른다. 의존성은 트랜잭션에서 참조되는 셀을 의미한다. 의존성으로 참조된 셀은 읽기 전용으로 트랜잭션 내에서 실행되는 모든 스크립트에 제공된다. 의존성(deps)은 소비되지 않는다.

동의어

- [Deps](#)

함께 보기

- [Cell](#)
- [Consume](#)

- [Script](#)
 - [Transaction](#)
-

Duktape

이식성과 작은 메모리 풋프린트에 중점을 둔 임베디더블 자바스크립트 엔진.

Duktape는 Nervos에서 자바스크립트 기반 스마트 컨트랙트를 실행하는 데 사용된다.

함께 보기

- [Duktape Official Website](#)
-

Difficulty

블록 생성을 위해 요구되는 [Proof of Work](#) 암호 퍼즐을 해결하는 난이도의 척도.

네트워크는 채굴 참여자 수의 변동에 따라 블록 생성 속도를 제어하기 위해 난이도를 자동 조정한다.

함께 보기

- [Proof of Work](#)
-

Diviner

Rust용 결정적(deterministic) 테스트 프레임워크.

함께 보기

- [Diviner on GitHub](#)
-

Eaglesong

Nervos CKB 작업증명(Proof-of-Work)을 위해 특별히 개발된 새로운 해시 함수로, 보안 해시 함수가 필요한 다른 용례에도 적합하다.

함께 보기

- [Eaglesong RFC on the Nervos Github](#)
 - [Introducing Eaglesong, Nervos's New Hash Function for CKB Proof-of-Work](#)
-

epoch_duration_target

NC-Max가 지정하는 예상 에포크 길이. CKB에서는 4시간으로 설정된다.

```
pub(crate) const DEFAULT_EPOCH_DURATION_TARGET: u64 = 4 * 60 * 60; // 4
hours, unit: second
```

함께 보기

- [Epoch](#)
 - [NC-Max](#)
-

ERC20

기본적인 대체 가능 토큰을 위한 이더리움 토큰 표준.

Nervos의 sUDT는 이더리움의 ERC20 또는 ERC777 토큰 표준에 해당한다.

함께 보기

- [ERC777](#)
 - [Fungible Token](#)
 - [Token](#)
 - [User-Defined Token](#)
 - [ERC20 on Ethereum.org](#)
-

ERC721

대체 불가능 토큰(NFT)을 위한 이더리움 토큰 표준.

함께 보기

- [Non-Fungible Token](#)
 - [Token](#)
 - [ERC721 on Ethereum.org](#)
-

ERC777

ERC20과의 하위 호환을 유지하면서 개선된 이더리움 대체 가능 토큰 표준.

Nervos의 sUDT는 이더리움의 ERC20 또는 ERC777 토큰 표준에 해당한다.

함께 보기

- [ERC20](#)
 - [Fungible Token](#)
 - [Token](#)
 - [User-Defined Token](#)
 - [ERC777 on Ethereum.org](#)
-

ERC1155

단일 컨트랙트에서 다수의 대체 가능/대체 불가능 토큰을 생성할 수 있도록 하는 이더리움 토큰 표준.

함께 보기

- [Fungible Token](#)
 - [Non-Fungible Token](#)
 - [Token](#)
 - [User-Defined Token](#)
 - [ERC1155 on Ethereum.org](#)
-

Fee Bumping

혼잡하거나 초기 수수료가 너무 낮은 경우 확인 가속을 위해, 이미 브로드캐스트된 트랜잭션의 수수료를 증가시키는 메커니즘. 대표적인 방법은 CPFP(Child Pays For Parent)와 RBF(Replace-By-Fee)이다.

함께 보기

- [CPFP](#)
 - [RBF](#)
-

Full Address

연관된 Lock Script의 전체 코드 해시가 포함된 Nervos의 주소 형식.

함께 보기

- [Address](#)
 - [Short Address](#)
-

Generator

Nervos CKB 네트워크에 브로드캐스트할 수 있는 트랜잭션을 생성하는 데 사용되는 프로그램.

제너레이터는 로컬(오프체인)에서 실행되며, 사용자 입력과 기존 셀을 프로그램 입력으로 사용하여, 새로운 상태를 담은 새로운 셀을 출력으로 만든다.

Genesis Block

블록체인의 전역 상태를 초기화하기 위해 생성되는 첫 번째 블록. 제네시스 블록은 첫 블록이므로 이전 블록을 참조하지 않는다는 점이 유일하다.

함께 보기

- [Block](#)
 - [Blockchain](#)
-

genesis_hash

CKB 제네시스 블록의 해시. CKB 제네시스 블록은 [Genesis Block Generator](#)를 통해 누구나 검증 가능하게 고유의 제네시스 블록을 생성하도록 하는 분산 방식으로 만들어졌다. 이렇게 생성·활성화된 노드는 네트워크의 다른 어떤 노드와도 연결되어 분산된 Common Knowledge Base를 형성할 수 있다.

제네시스 블록은 두 가지 주요 구성 요소를 포함한다:

- [System script](#)
- Nervos 재단의 [공개 발표](#)에 따른 토큰 사전 할당

함께 보기

- [Genesis block](#)
-

Godwoken

Nervos CKB를 위한 레이어 2 롤업 프레임워크. 확장성과 더불어 추상 계정 모델을 CKB에 제공한다.

함께 보기

- [Godwoken on GitHub](#)
 - [Godwoken Documentation Site](#)
-

Governance Script

사용자 정의 토큰(UDT)의 통화 정책을 정의하는 Type Script.

함께 보기

- [Governance Script Hash](#)
 - [UDT](#)
 - [User Defined Token](#)
 - [Type Script](#)
-

Governance Script Hash

셀에서 스크립트를 참조할 때 식별자로 사용하는, 해당 Type Script의 Blake2b 해시.

동의어

- [Type Script Hash](#)

함께 보기

- [Governance Script](#)
 - [UDT](#)
 - [User Defined Token](#)
 - [Type Script](#)
-

Historical Cell

[Dead Cell](#)을 가리키는 대체 용어.

동의어

- [Dead Cell](#)

함께 보기

- [Cell](#)
 - [Cell Model](#)
-

Inbound Connection

원격 피어가 시작한 연결을 의미한다. 주체를 원격 피어로 바꾸면, 해당 연결은 outbound(발신) 연결이 된다.

함께 보기

- [Outbound Connection](#)
-

Indexer

개발자나 사용자가 지정한 기준에 부합하는 Live Cell을 추적하는 애플리케이션 또는 라이브러리.

함께 보기

- [Cell](#)
 - [Live Cell](#)
-

initial_primary_epoch_reward

CKB 기본 발행(base issuance)에서 에포크 단위로 채굴자에게 CKBytes로 지급되는 인센티브. CKB의 합의에서 블록 간격은 불확실하지만 에포크는 약 4시간으로 고정될 수 있으므로, 보상 발행은 에포크 기준으로 결정된다. 각 에포크는 1_917_808_21917808 Shannons의 CKBytes를 발행하며, 총량은 고정되어 있고 약 4년마다 반감한다.

초기 기본 발행은 연 42억 CKBytes이며, 비트코인과 유사하게 약 4년마다 반감하여 결국 중단된다.

계산식: $\text{block reward} = \text{initial_primary_epoch_reward} / \text{epoch_length}$ (해당 에포크의 블록 수).

```
pub(crate) const INITIAL_PRIMARY_EPOCH_REWARD: Capacity =  
Capacity::shannons(1_917_808_21917808);
```

함께 보기

- [Block Reward](#)
 - [Epoch](#)
 - [secondary_epoch_reward](#)
-

Input

트랜잭션에서 사용되는 Live Cell. 트랜잭션이 네트워크에 의해 수락되면, 해당 Live Cell은 입력으로 소비되어 Dead Cell로 표시된다.

함께 보기

- [Cell](#)
- [Consume](#)

- [Dead Cell](#)
 - [Live Cell](#)
 - [Transaction](#)
-

Keyper

특정 사용자에게 적용되는 지갑 Lock Script를 관리하는 방법을 규정한 사양.

함께 보기

- [Lock Script](#)
 - [Keyper on GitHub](#)
-

Late Spawning

네트워크가 이미 일정 기간 운영된 후에 노드가 처음 블록체인 네트워크에 합류하는 상황.

네트워크가 late spawning을 지원한다는 것은, 참여자가 네트워크의 어떤 노드도 제공 데이터의 무결성을 신뢰할 필요 없이 전체 블록체인을 다운로드하고 검증할 수 있음을 의미한다.

함께 보기

- [Genesis Block](#)
-

Layer 1

탈중앙화 생태계의 기반 블록체인 아키텍처.

Nervos Network의 기반 레이어인 작업증명 블록체인 Common Knowledge Base(CKB)를 가리킨다.

함께 보기

- [CKB](#)
 - [Common Knowledge Base](#)
 - [Layer 2](#)
-

Layer 2

기존 블록체인 시스템 위에 구축되는 2차 프레임워크 또는 프로토콜을 의미한다.

주요 목적은 대다수 암호화폐 네트워크가 직면한 트랜잭션 속도와 확장성 문제를 해결하는 것이다.

함께 보기

- [Layer 1](#)
-

Lina

Nervos CKB의 공개 메인넷 이름.

- **CKB 버전:** $\geq$ v0.25.2 (최신 안정 버전 권장)
- **제네시스 해시:**
0x92b197aa1fba0f63633922c61c92375c9c074a93e85963554f5499fe1450d0e5
- **초기화 명령어:** `ckb init --chain mainnet`
- **출시 시점:** 2019-11-15 21:11:00 UTC

동의어

- [Mainnet](#)

혼동하지 말 것

- [Aggron](#)
- [Testnet](#)

함께 보기

- [Nervos CKB](#)
-

Live Cell

아직 소비되지 않아 사용 가능한 셀.

비트코인의 미사용 트랜잭션 출력(UTXO)과 유사하다.

동의어

- [Active Cell](#)

함께 보기

- [Cell](#)
- [Cell Model](#)

- [UTXO on Bitcoin.org](#)
-

Lock

셀의 소유권을 나타내는 스크립트. 사용자가 셀의 Lock Script를 정상 종료로 통과(언락)하면, 해당 셀을 소비할 수 있다.

함께 보기

- [Lock Script](#)

Lock Script

셀의 접근과 소유권을 강제하는 스크립트. 이 스크립트는 셀을 입력으로 사용할 권한이 누구에게 있는지 제어한다. Lock Script는 사용자 생성 증명 또는 위트니스를 받아 입력으로 포함시킨다.

Nervos CKB에서는 Lock Script를 바이트 배열(페이로드)로 인코딩하고 최종 주소 형식으로 래핑하며, 이로써 Lock Script와 주소 간 1:1 대응이 성립한다.

함께 보기

- [Address](#)
 - [Cell](#)
 - [Type Script](#)
 - [Script](#)
-

Lock Script Hash

셀에서 스크립트를 참조할 때 식별자로 사용하는, Lock Script의 [Blake2b](#) 해시.

함께 보기

- [Cell](#)
 - [Lock Script](#)
-

Mainnet

“메인 네트워크”의 약칭으로, 가동 중인 Nervos CKB 공용 블록체인을 의미한다. Nervos CKB 메인넷의 이름은 [Lina](#)이다.

동의어

- [CKB](#)

- [Common Knowledge Base](#)
- [Lina](#)
- [Nervos CKB](#)

혼동하지 말 것

- [Aggron](#)
- [Testnet](#)

max_block_bytes

블록에 허용되는 최대 트랜잭션 크기(바이트). 1000개의 2-in-2-out secp256k1 트랜잭션이 소비하는 크기를 기준으로 산정된다.

```
pub const MAX_BLOCK_BYTES: u64 = TWO_IN_TWO_OUT_BYTES *  
TWO_IN_TWO_OUT_COUNT;
```

max_block_cycles

블록에 허용되는 최대 트랜잭션 사이클 한도. 1000개의 2-in-2-out secp256k1 트랜잭션이 소비하는 사이클을 기준으로 산정된다.

```
/// cycles of a typical two-in-two-out tx.  
pub const TWO_IN_TWO_OUT_CYCLES: Cycle = 3_500_000;  
  
/// count of two-in-two-out txs a block should capable to package.  
const TWO_IN_TWO_OUT_COUNT: u64 = 1_000;  
pub(crate) const MAX_BLOCK_CYCLES: u64 = TWO_IN_TWO_OUT_CYCLES *  
TWO_IN_TWO_OUT_COUNT;
```

함께 보기

- [Cycles](#)

max_block_proposals_limit

하나의 블록에 포함될 수 있는 제안(proposal)의 최대 개수. 기본값은 1,500부터 시작한다.

```
TWO_IN_TWO_OUT_COUNT  
pub const MAX_BLOCK_PROPOSALS_LIMIT: u64 = 1_500;
```

함께 보기

- [Proposal Zone](#)
-

max_uncles_num

하나의 블록에 허용되는 앙클 블록의 최대 개수(Uint64). 기본값은 2이다.

```
const MAX_UNCLE_NUM: usize = 2;
```

함께 보기

- [Uncle](#)
-

median_time_block_count

타임스탬프는 직전 37개 블록의 중앙값 타임스탬프보다 커야 유효하다.

```
const MEDIAN_TIME_BLOCK_COUNT: usize = 37;
```

Minting

새로운 토큰을 생성하는 과정.

함께 보기

- [Token](#)
-

Molecule

Nervos Network에서 널리 사용되는 직렬화 프레임워크.

함께 보기

- [Molecule Specification](#)
 - [Molecule on Github](#)
-

Muta

PoS, BFT 합의 및 스마트 컨트랙트를 지원하도록 설계된 고도로 커스터마이징 가능한 고성능 블록체인 프레임워크.

함께 보기

- [Muta on GitHub](#)
 - [Proof of Stake](#)
-

Nervos CKB

Nervos Network의 레이어 1 블록체인, Common Knowledge Base.

Nervos CKB는 Nervos Blockchain이라고도 불린다.

동의어

- [CKB](#)
- [Common Knowledge Base](#)
- [Nervos Blockchain](#)

함께 보기

- [Layer 1](#)
-

Off-Chain Computation

네트워크 노드의 부담을 줄이고 더 높은 확장성을 제공하기 위해, 모든 연산을 오프체인에서 수행하는 프로그래밍 모델. Nervos는 오프체인 연산과 온체인 검증을 사용한다.

함께 보기

- [On-Chain Computation](#)
 - [On-Chain Verification](#)
-

Off-Chain Scaling

블록체인을 보안 자산 및 결제 레이어로만 사용하고, 대부분의 트랜잭션을 블록체인 밖에서 처리하는 접근법.

함께 보기

- [On-Chain Scaling](#)
-

Off-Chain State

블록체인에 저장되지 않았거나 온체인 스마트 컨트랙트가 접근할 수 없는 애플리케이션의 데이터.

함께 보기

- [On-Chain State](#)
-

On-Chain Computation

스마트 컨트랙트의 모든 연산이 네트워크의 모든 노드에서 동시에 온체인으로 수행되는 프로그래밍 모델.

이더리움은 온체인 연산을 사용한다.

함께 보기

- [Off-Chain Computation](#)
-

On-Chain Scaling

합의 과정의 처리량을 확장하거나, 노드 수 증가에 따라 네트워크 처리량을 늘리는 등의 확장 솔루션을 지칭.

함께 보기

- [Off-chain Scaling](#)
-

On-Chain State

블록체인에 저장되어 있으며 온체인 스마트 컨트랙트가 접근 가능한 애플리케이션 데이터.

Nervos는 모든 스마트 컨트랙트에 온체인 상태를 제공한다.

함께 보기

- [Off-Chain State](#)
-

On-Chain Verification

네트워크 노드의 부담을 줄이기 위해 모든 연산을 오프체인에서 수행하되, 결과 데이터의 검증은 온체인에서 수행하여 개발자가 정의한 스마트 컨트랙트 규칙을 강제하는 프로그래밍 모델.

Nervos는 오프체인 연산과 온체인 검증을 사용한다.

함께 보기

- [On-Chain Computation](#)
-

Open Transaction

불완전하고 단독으로는 무효인, 서명된 트랜잭션 조각. 다른 서명된 트랜잭션 조각들과 결합되면 완전한 트랜잭션을 형성하여 처리될 수 있다.

오픈 트랜잭션의 한 용례는 신뢰 없는 탈중앙화 거래소 기능을 구현하는 것이다.

함께 보기

- [Cryptographic Signature](#)
 - [Transaction](#)
-

Optimistic Rollup

사기 증명(fraud proof)을 사용하여 레이어 2 트랜잭션 처리량을 높이면서, 레이어 1이 제공하는 보안 및 데이터 가용성을 활용하는 롤업 방식.

함께 보기

- [Layer 1](#)
 - [Layer 2](#)
-

Orphan

Orphan Block의 약칭.

동의어

- [Orphan Block](#)
 - [Uncle](#)
-

Orphan Block

네트워크 지연 등으로 인해 메인 포크에 포함되지 않은 유효한 블록. NC-Max에서는 두 채굴자가 거의 동시에 블록을 찾을 수 있다. 메인체인이 아닌 블록으로, stale 블록이라고도 한다.

Nervos에서는 orphan 블록을 “Uncle”로 표현하는 것이 더 적절하다.

동의어

- [Orphan](#)
- [Uncle](#)

함께 보기

- [Block](#)
 - [NC-Max](#)
 - [Orphan Rate](#)
-

Orphan Rate

블록체인 네트워크에서 orphan 블록이 발생하는 속도를 측정하는 지표.

함께 보기

- [Orphan Block](#)
 - [orphan_rate_target](#)
-

orphan_rate_target

NC-Max가 지정하는 목표 orphan 블록 비율. CKB에서는 2.5%로 설정된다.

```
// o_ideal = 1/40 = 2.5%  
pub(crate) const DEFAULT_ORPHAN_RATE_TARGET: (u32, u32) = (1, 40);
```

함께 보기

- [Orphan Rate](#)
 - [Orphan Block](#)
-

Orphan Transaction

처리 시점에 부모 트랜잭션이 부재한 트랜잭션. 다른 노드로 전파되지 않고 로컬 버퍼에 머무르며, 제거되거나 부모 트랜잭션이 발견될 때까지 대기한다.

함께 보기

- [Orphan Block](#)
-

Outbound Connection

“outgoing connection”이라고도 한다.

노드 관점에서 TCP 연결을 해당 노드가 시작(초기 TCP SYN 패킷 전송)했다면, 그 연결은 outbound(발신) 연결이다.

함께 보기

- [Inbound Connection](#)
-

Outpoint

트랜잭션 내의 특정 출력 셀.

함께 보기

- [Cell](#)
 - [Output](#)
 - [Transaction](#)
-

Output

트랜잭션에서 새로 생성되는 Live Cell.

함께 보기

- [Cell](#)
 - [Live Cell](#)
 - [Transaction](#)
-

Overlord

Huobi를 위해 Nervos가 설계한 비잔틴 장애 허용 합의 알고리즘으로, 초당 수천 건의 트랜잭션을 처리할 수 있다.

함께 보기

- [Overlord on Medium](#)
-

P2WSH

Pay-to-Witness-Script-Hash. [P2SH](#)와 유사한 비트코인 트랜잭션 유형이지만, [SegWit](#)을 사용한다.

permanent_difficulty_in_dummy

dev-chain이 NC-MAX 난이도 조정을 비활성화하여 PoW가 더미(dummy)일 때, 난이도를 고정 상태로 유지한다. boolean으로 구성에서 활성화할 수 있다.

함께 보기

- [Difficulty](#)
-

Polyjuice

Godwoken 상에서 이더리움 호환 런타임을 제공한다.

함께 보기

- [Godwoken](#)
 - [Polyjuice on GitHub](#)
-

primary_epoch_reward_halving_interval

CKB 기본 발행에서 에포크 보상의 반감 주기(일반적으로 4년). 반감 시점이 오면 채굴 보상이 절반으로 감소한다.

```
pub(crate) const DEFAULT_PRIMARY_EPOCH_REWARD_HALVING_INTERVAL:
EpochNumber =
4 * 365 * 24 * 60 * 60 / DEFAULT_EPOCH_DURATION_TARGET; // every 4 years
```

함께 보기

- [initial_primary_epoch_reward](#)
-

Proposal Zone

블록에서 트랜잭션 제안이 포함되는 섹션.

함께 보기

- [Commitment Zone](#)
 - [Propose](#)
-

Propose

미확인 트랜잭션을 메모리풀에서 꺼내 커밋 제안으로 올리는 과정. 트랜잭션은 커밋되기 전까지 확인되지 않는다.

채굴자는 제안 보상(Proposal Reward)을 통해 제안 인센티브를 얻는다.

함께 보기

- [Commit](#)
- [Confirmation](#)
- [Mempool](#)
- [Proposal Reward](#)
- [Proposal Zone](#)
- [Transaction](#)

proposer_reward_ratio

NC-Max가 지정하는, 트랜잭션 수수료 중 제안을 제출한 채굴자에게 돌아가는 보상 비율. CKB에서는 40%로 설정되어 있으며, 트랜잭션을 최초로 제안한 채굴자가 수수료의 40%를 보상으로 받는다.

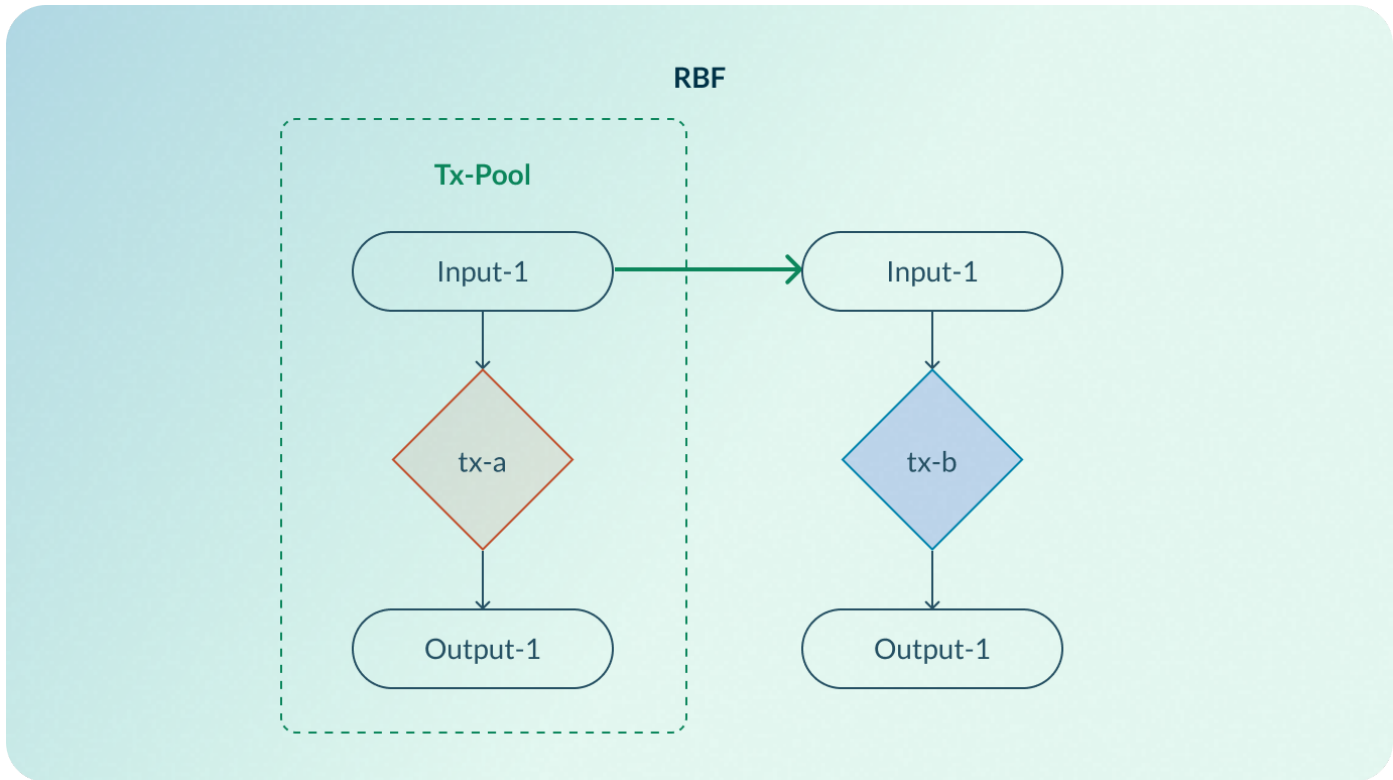
```
const PROPOSER_REWARD_RATIO: Ratio = Ratio::new(4, 10);
```

함께 보기

- [Proposal Reward](#)
- [NC-Max](#)

RBF

Replace-by-Fee의 약자. 메모리풀 내 트랜잭션을 동일한 입력을 사용하되 더 높은 수수료를 가진 새 트랜잭션으로 대체하는 전략. 새 트랜잭션의 수수료는 자신과 대체되는 모든 트랜잭션의 수수료를 총당할 만큼 충분해야 한다. RBF는 처음 [BIP 125](#)에서 도입되었다.



함께 보기

- [CPFP](#)
- [Fee Bumping](#)

RGB++

네이티브 비트코인 UTXO를 등형 바인딩(isomorphic bindings)으로 Nervos CKB 셀에 매핑하는 토큰 프로토콜. 비트코인에 앵커링된 안전하고 프로그래머블한 자산 발행과 상태 전이를 가능하게 하며, CKB를 활용해 확장 가능한 검증, 공유 상태, 스크립팅 기능을 제공한다.

함께 보기

- [RGB](#)
- [RGB++ Protocol](#)

RISC-V

범용 컴퓨팅을 위한 오픈 표준 명령어 집합(ISA).

RISC-V는 CKB-VM이 사용하는 명령어 집합이다.

함께 보기

- [CKB-VM](#)
- [RISC-V on Wikipedia](#)

Schnorr Signature

서명과 검증을 위한 암호 서명 방식.

함께 보기

- [Schnorr Signature on Wikipedia](#)
-

Script

Nervos CKB에서 스크립트는 CKB-VM에서 실행되는 바이너리 실행 파일이다. 비트코인 스크립트와 달리, CKB 스크립트는 튜링 완전하며 스마트 컨트랙트에 해당한다. 스크립트는 두 가지 유형이 있다:

- Lock Script — 셀의 소유권과 접근을 제어
- Type Script — 트랜잭션에서 셀을 어떻게 사용할지 제어

함께 보기

- [CKB-VM](#)
 - [Lock Script](#)
 - [RISC-V](#)
 - [Type Script](#)
 - [ELF on Wikipedia](#)
-

script_args

script_args는 스크립트의 입력으로 CKB-VM 인스턴스에 전달되는 인자를 의미한다. CKB에서는 공개키 정보가 관례적으로 script_args에, 서명 정보는 witnesses에 저장되지만, 이는 필수는 아니다.

스크립트가 검증될 때, CKB는 RISC-V VM에서 이를 실행하며 script_args는 특별한 CKB syscall을 통해 로드되어야 한다. UNIX 표준의 argc/argv 관례는 CKB에서 사용되지 않는다. 자세한 내용은 [RFC003: CKB-VM](#)을 참조.

함께 보기

- [CKB-VM](#)
 - [Witness](#)
-

secondary_epoch_reward

에포크당 2차 보상. [RFC0015](#)에 상세한 CKB 토크노믹스에 따라 발행된다.

2차 발행은 상태 렌트(state rent)를 징수하도록 설계되었으며, 시간에 따라 일정한 발행량을 가진다. 기본 발행이 중단된 이후에는 2차 발행만 존재한다.

2차 발행은 두 부분으로 이뤄진다. 하나는 고정된 기본 인센티브(연 약 1억3440만 CKBytes)이고, 다른 하나는 현재 점유 중인 CKBytes 양에 따라 달라진다.

```
pub(crate) const DEFAULT_SECONDARY_EPOCH_REWARD: Capacity =  
Capacity::shannons(613_698_63013698);
```

함께 보기

- [initial_primary_epoch_reward](#)
- [Secondary Reward](#)
- [Secondary Issuance](#)

secp256k1_blake160_sighash_all_type_hash

[CKB 제네시스 스크립트](#)에 포함된 secp256k1_blake160_sighash_all의 타입 해시.

자세한 내용은 [여기](#) 참조.

secp256k1_blake160_multisig_all_type_hash

[CKB 제네시스 스크립트](#)에 포함된 secp256k1_blake160_multisig_all의 타입 해시.

자세한 내용은 [여기](#) 참조.

Seed Cell

위조 불가능한 자산을 만들기 위한 고유 식별자를 생성하는 데 사용하는 Nervos의 디자인 패턴.

함께 보기

- [Cell](#)

Shannon

CKBytes의 분수 단위. 1 CKByte = 100,000,000 Shannon.

Shannon은 비트코인의 Satoshi에 해당한다.

함께 보기

- [CKByte](#)
 - [Common Knowledge Byte](#)
 - [Satoshi \(denomination\) on Bitcoin.org](#)
-

Short Address

연관 Lock Script의 코드 해시를 포함하지 않고, 여러 공통 Lock Script 중 하나를 사용하는 Nervos의 주소 형식.

Short Address 형식이 가장 흔히 사용되며, 흔히 단순히 “address”라고도 부른다.

동의어

- [Address](#)

함께 보기

- [Code Hash](#)
 - [Lock Script](#)
 - [Full Address](#)
-

Simple UDT

Nervos에서 가장 기본적인 UDT 대체 가능 토큰 구현을 정의하는 표준.

Nervos의 sUDT는 이더리움의 ERC20 또는 ERC777 토큰 표준에 해당한다.

동의어

- [SUDT](#)

함께 보기

- [Token](#)
 - [UDT](#)
 - [User-Defined Token](#)
 - [ERC20 on Ethereum.org](#)
 - [Simple UDT RFC on Nervos Network Github](#)
-

Since

since는 특정 블록 타임스탬프 또는 블록 번호 이전에 포함되는 것을 방지하기 위한, 트랜잭션 입력의 u64(부호 없는 64비트 정수) 필드.

함께 보기

- [Cell](#)
-

SPV

Simplified Payment Verification의 약자. 전체 노드를 운영하지 않고도 블록체인 암호화폐를 사용하는 프로토콜.

SPV 클라이언트는 저장해야 할 데이터가 훨씬 적지만, 연결된 네트워크 클라이언트를 어느 정도 신뢰해야 한다.

함께 보기

- [SPV Wallet](#)
 - [Simplified Payment Verification on BitcoinWiki](#)
-

SPV Wallet

SPV 프로토콜을 사용하는 경량 암호화폐 지갑.

함께 보기

- [SPV](#)
-

SUDT

Simple UDT의 약자.

동의어

- [Simple UDT](#)
-

Testnet

“테스트 네트워크”의 약칭으로, [Mainnet](#)의 동작을 시뮬레이션하는 네트워크. Nervos CKB의 테스트넷 이름은 [Aggron](#)이다.

동의어

- [Aggron](#)

혼동하지 말 것

- [Lina](#)
- [Mainnet](#)

Transaction Confirmation Number

트랜잭션이 블록에 추가되기 위해 요구되는 확인 횟수.

권한 없는 블록체인 설계는 확률적 최종성만 제공하므로, 무한한 계산 능력을 가진 적대자를 상정하면 트랜잭션은 완전히 확정될 수 없다. 따라서 사용자와 앱은 자신들이 안전하다고 판단하는 확인 횟수를 선택할 수 있다. 여기서는 보안 수준에 영향을 주는 요인 중 하나인 최근 orphan rate를 간단히 논의한다. orphan rate가 0일 때는 6회 확인이 완전 결제를 보장하며, 동일한 보안 수준을 얻기 위해 orphan rate가 2.5%에 이르면 24회 확인이 필요하다([근거 및 계산](#) 참조).

블록체인 결제 보증에서, 트랜잭션 확인은 정량화하기 어려운 변수 중 하나다(Nic Carter의 [글](#) 참조). 정확한 수치는 사용자가 원하는 보안 수준에 따라 조정될 수 있다. Nervos CKB는 보수적으로 최소 15회의 확인을 권장한다.

함께 보기

- [Confirmation](#)
- [NC-Max](#)
- [Orphan Block](#)
- [Orphan Rate](#)

Transaction Hash

블록체인에서 트랜잭션의 고유 식별자(Txhash)로, 트랜잭션이 발생했음을 증명·기록한다. CKB에서 트랜잭션 해시를 얻기 위해, 트랜잭션은 Molecule로 직렬화되고, 직렬화된 raw가 ckbhash 함수에 입력된다. 스키마는 다음과 같다:

```
table Transaction {
  raw:          RawTransaction,
  witnesses:    BytesVec,
}
```

트랜잭션 해시는 직렬화된 raw 구조를 [ckbhash](#)로 해싱하여 생성된다.

함께 보기

- [Transaction Witness Hash](#)
 - [Molecule](#)
 - [Ckbhash](#)
 - [Transaction Hash in RFC on Nervos Network Github](#)
-

Transaction Witness Hash

트랜잭션 전체를 ckbhash로 해싱하여 생성되는 해시. CKB에서 트랜잭션은 [molecule](#)로 직렬화되며, 스키마는 다음과 같다:

```
table Transaction {  
    raw:          RawTransaction,  
    witnesses:    BytesVec,  
}
```

함께 보기

- [Transaction Hash](#)
 - [Molecule](#)
 - [Ckbhash](#)
-

Transaction Root

헤더의 transactions_root 필드는 다음과 같다:

```
ckbhash(T || W)
```

ckbhash는 해시 함수, ||는 이진 연결.

T는 블록 내 모든 트랜잭션의 [transaction hashes](#)를 항목으로 하는 [CKB Merkle Tree](#)의 루트이다.

W 역시 블록 내 모든 트랜잭션의 [Transaction Witness Hash](#)를 항목으로 하는 [CKB Merkle Tree](#)의 루트이다.

함께 보기

- [Ckbhash](#)
- [CKB Merkle Tree](#)

- [Transaction Witness Hash](#)
-

tx_proposal_window

NC-Max가 지정하는 2단계 제안 제출 간격. CKB에서는 2~10 블록으로 설정된다.

위 예시에서, 트랜잭션이 블록 13에서 처음 제안되면, 블록 15~23 사이의 창에서 커밋될 수 있다.

```
pub(crate) const TX_PROPOSAL_WINDOW: ProposalWindow = ProposalWindow(2, 10);
```

tx_version

트랜잭션의 버전. 시스템 예약 필드이며 0으로 설정된다.

```
pub const TX_VERSION: Version = 0;
```

type_id

CKB의 `system_scripts` 중 하나. Type ID의 고유한 특징은 CKB에 내장된 스크립트로서 Rust로 직접 구현되어 있다는 점이다. CKB-VM에서 실행되지는 않지만 다른 [CKB 제네시스 스크립트](#)처럼 사용할 수 있다.

Type ID에 대한 심층 설명은 블로그 글을 참고: [Introduction to CKB Script Programming 6: Type ID \(중국어 번역 \)](#)

함께 보기

- [Script](#)
 - [Type Script](#)
-

type_id_code_hash

`type_id`의 `code_hash`. `type_id`의 특수성을 고려하여, `type_id_code_hash`는 실제 코드의 해시가 아니라 하드코딩된 값이다.

함께 보기

- [type_id](#)
-

Type Script

트랜잭션에서 셀이 입력으로 소비되거나 출력으로 생성되기 위해 따라야 하는 규칙을 강제하는 스크립트.

함께 보기

- [Cell](#)
 - [Lock Script](#)
 - [Script](#)
 - [Type Script Hash](#)
-

Type Script Hash

또는 `type_hash`. 셀에서 스크립트를 참조할 때 식별자로 사용하는 Type Script의 Blake2b 해시.

CKB 셀 데이터 구조에는 `lock`과 `type` 두 엔티티가 있다. Type Script는 Offtransformation에서 필요한 모든 검증 로직을 포착할 수 있다.

Type Script는 경제적 구조도 구현할 수 있다. NervosDAO는 합의 레이어의 최소 지원만으로 Type Script로 완전히 구현되어 있다.

함께 보기

- [Cell](#)
 - [Script](#)
 - [Type Script](#)
 - [Lock Script](#)
 - [NervosDAO](#)
-

Uncle

Uncle Block. 두 블록이 거의 동시에 채굴되어 원장에 제출될 때 생성된다. 한 블록만 원장에 포함될 수 있고, 다른 하나는 포함되지 않는다.

Uncle이 발견되어 보고되면 감소된 블록 보상을 받는다.

Nervos에서는 네트워크의 블록 간격을 조정하기 위해 합의에서 Uncle을 추적한다.

Uncle 블록은 다음 조건을 만족해야 한다:

- 메인 체인에 있지 않아야 한다. 즉, 메인 체인의 어떤 블록이라도 포함하면 uncle이 될 수 없다.
- Uncle의 블록 번호는 이후에 해당 uncle을 포함하는 블록의 번호보다 작아야 한다.

- Uncle의 부모는 메인 체인에 있거나, 부모 역시 uncle이어야 한다. 즉, uncle은 어떤 방식으로든 메인 체인에 연결 가능해야 한다. 메인 체인과 연결되지 않은 임의의 블록은 uncle이 될 수 없다.

```

- if !snapshot.is_main_chain(&uncle.hash()) // 메인 체인에 없어야 한다.
- && !snapshot.is_uncle(&uncle.hash()) // 두 번 uncle이 될 수 없다.
- && uncle.number() < candidate_number // uncle의 블록 번호는 메인 체인의 현재
블록 번호보다 작아야 한다.
- && (uncles.iter().any(|u| u.hash() == parent_hash)
|| snapshot.is_main_chain(&parent_hash) // 부모 없는 uncle은 불가. 부모는 메인
체인의 블록이거나 또 다른 uncle이어야 한다.
|| snapshot.is_uncle(&parent_hash))

```

위 그림에서, A는 메인 체인이다. B3는 A2에 연결되어 있으므로 A4의 uncle이 될 수 있다(=A4에 포함). 그러나 B4는 A4에 포함될 수 없다. uncle의 블록 번호는 현재 메인 체인 블록 A4보다 작아야 하기 때문이다.

유사하게, B4는 A5의 uncle이 될 수 있다(A5에 포함). 비록 B4의 부모 B3가 메인 체인에 있지 않지만, B3는 A4의 uncle이다. 따라서 B4는 합법적인 uncle이며, B3는 A5에 포함될 수 없다.

C2와 C3는 부모가 알려지지 않아 메인 체인에 연결될 수 없으므로, uncle이 될 수 없다.

함께 보기

- [max_uncles_num](#)

동의어

- [Orphan Block](#)

함께 보기

- [Block Interval](#)
- [Orphan Rate](#)
- [Uncle](#)

Uncle Rate

함께 보기

- [Orphan Rate](#)
 - [Uncle](#)
-

Validator

제너레이터가 만든 트랜잭션이 유효한지 확인하는 데 사용되는 스크립트. Validator는 CKB-VM에서 Lock Script 또는 Type Script로 실행된다.

함께 보기

- [CKB-VM](#)
 - [Lock Script](#)
 - [Type Script](#)
 - [Transaction](#)
-

Witness

트랜잭션에서 사용된 자원의 권한을 증명하는 데 필요한 데이터를 포함하는 암호학적 증명의 집합.

함께 보기

- [Transaction](#)
-

XUDT

sUDT(Simple UDT)를 기반으로 규제 준수 등 추가 기능을 수용하도록 확장한 UDT 표준. Extensible UDT라고도 쓴다.

함께 보기

- [User-Defined Token](#)
 - [Simple UDT](#)
 - [UDT](#)
-

Zk-SNARK

암호학적 증명의 한 형태로, 암호화폐에 사용될 때 트랜잭션의 금액이나 참여자를 공개하지 않는 프라이버시 기능을 제공한다.

Zk-SNARK는 신뢰 설정(trusted setup)이 필요하지만, 그 외에는 신뢰 없이 동작한다.

함께 보기

- [Transaction](#)

- [Zk-STARK](#)
 - [Non-interactive zero-knowledge proofs on Wikipedia](#)
-

Zk-STARK

암호학적 증명의 한 형태로, 암호화폐에 사용될 때 트랜잭션의 금액이나 참여자를 공개하지 않는 프라이버시 기능을 제공한다.

Zk-SNARK와 달리, Zk-STARK는 신뢰 설정이 필요 없다.

함께 보기

- [Transaction](#)
 - [Zk-SNARK](#)
 - [Non-interactive zero-knowledge proofs on Wikipedia](#)
-

1. 트랜잭션 A가 자신의 출력 Cell을 트랜잭션 B가 입력 Cell로 사용할 경우, A는 B의 부모로 간주된다. 선조는 부모 또는 어떤 부모의 선조이다. [☞](#)
2. 확인 가속을 원한다면, 확인되지 않은 트랜잭션과 동일한 입력의 일부 또는 전부를 사용하되 더 높은 수수료로 새로운 트랜잭션을 만드는 RBF fee bumping 방법을 고려하라. Abandoned 트랜잭션을 삭제하면 이 옵션이 사라진다. RBF는 [v0.112.1](#) 이후 CKB의 기본 기능이다. [☞](#)