The UTXO Models Handbook

All Flavors of UTXO models in one menu

Authors:
Ignacio Calderon de la Barca [UTXO Alliance],
Steph Macurdy [WBL, UTXO Alliance]

Contributors:
Rupert Whitehead [IOG]
Matt Quinn [Nervos Network]
Kushti [Ergo]
Alan Orwick [Quai Network]
Jon Woodard [WBL]
Yan Martins [Hathor]

January 23, 2025

Contents

Overview

It's been a decade since Bitcoin with its underlying computing model, the **Unspent Transaction Output (UTXO) model**, brought the most prominent application of blockchain technology today - a decentralized financial system. These simple set of words encompass various virtues that make this new type of system much better than legacy financial systems.

Bitcoin's seminal work started the first wave of projects that offer a system with the attributes described above. Years later this pioneer breakthrough inspired subsequent revolutions reimagining socio-economic systems beyond the uni-dimensional feature of supporting transactions in a peer-to-peer manner. A second wave introduced **smart contracts**, which allows users to put conditions on financial transactions written in verifiable code which paves the way for **decentralized applications (dApps)** to emerge. Furthermore, a third wave of projects embedded self-governing mechanisms in the system. Today, a competitive blockchain must offer a network protocol that meets all previous advancements, in addition to having a detailed roadmap that is highly focused on maximizing all three dimensions of the decentralization-security-scalability triad (also known as the **'blockchain trilemma'**).

The approach that promising blockchain projects take is to first lay solid foundations weaved by multi-diciplinary research that latter on compounds into better quality of software. This distinction is characteristic of all UTXO Alliance members. The Alliance has, at its core, the goal of advancing blockchain technology by fostering collaboration towards interoperability, research, commercial adoption and education. The latter sets the tone of this manuscript.

How to read this document?

Chapter 1 presents the core components of a blockchain as a visual language (also referred to as *ingredients* in this handbook) and it is meant to be read sequentially. Whereas Chapter 2 focuses on explaining how all components are assembled to form a fully functionning blockchain. Aditionlly, we present diagrams for all UTXO Alliance chains highlighting their high-level technical specification differences and smart contract implementation. At last, Chapter 3 contains Supplementary Material including Section 3 which explains the basic *Account model* and compares it to the UTXO model. If you're curious about a particular chain jump right ahead to check Bitcoin (Section 2), Digibyte (Section 2), Cardano (Section 2), Ergo (Section 2), Nervos Network (Section 2), Quai Network (Section 2), Topl (Section 2), Alephium (Section 2) and Hathor (Section 2).

Goals

1. This handbook aims to be a presentation card of all the different chain designs of UTXO Alliance members. Aiming to be a handy resource for blockchain enthusiasts with minimal technical knowledge, entrepreneurs and commercial partners
2. Explain the key aspects of the UTXO model and the functioning of a blockchain using simple visual diagrams
3. Demonstrate the rich variety and benefits present in the variance of UTXO design patterns implemented by differnt members of the UTXO Alliance

Introduction to Blockchains

We recommend these three educational introductions to get up to speed for what you're about to read. These three resources are evergreen (meaning they've aged well with time) and include:

1. Digestable explainer on How does Bitcoin Work? by learn me a Bitcoin
2. Explainer on How Blockchains work by Anderson Brown
3. Technical YouTube video about Bitcoin by 3Blue1Brown

Chapter 1: Ingredients of a Blockchain

At its heart, a blockchain is a special kind of database that keeps track of information in a way that's secure, open for anyone to see, and nearly impossible to retroactively change. Unlike traditional databases managed by a single organization, blockchains are maintained by a network of computers working together. This design ensures that no single person or group has complete control over the information, see Fig. 1.1.

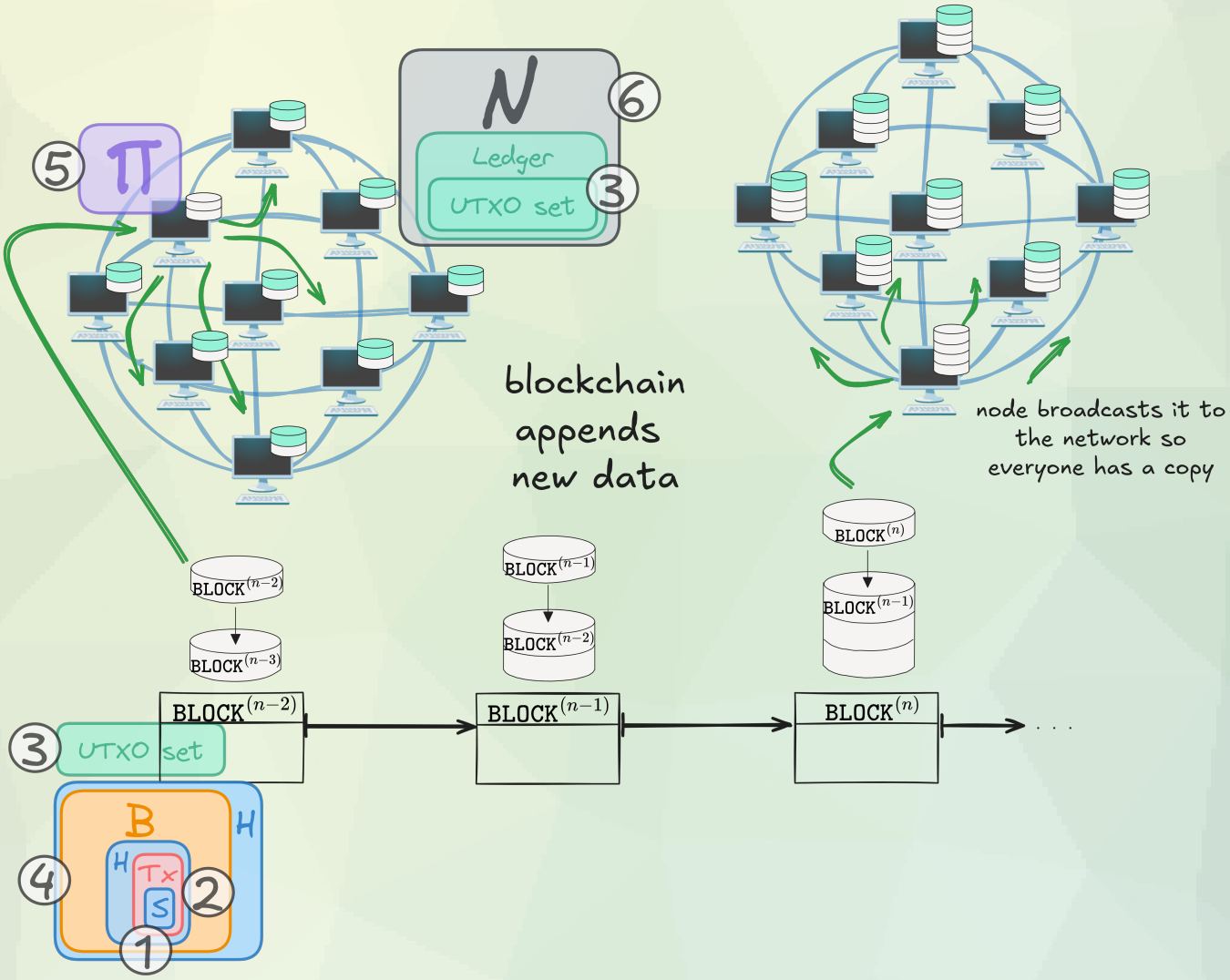

Figure 1.1: General functioning of a blockchain.

Understanding its core components is crucial for grasping how blockchains function and how different implementations or "flavors" of blockchain systems offer different benefits. In this section, we'll break down the essential "ingredients" that make up a UTXO-based blockchain. By examining these building blocks – cryptographic primitives, transactions, blocks, and the chain itself – we can better appreciate the innovations and variations in different blockchain designs.

As we explore each ingredient, we'll see how they contribute to the overall functionality and security of the blockchain. We'll also touch on how various blockchain projects have modified or extended these basic components to create their own unique characteristics, leading to a rich ecosystem of blockchain "flavors" all stemming from the same fundamental recipe. Lets navigate through Fig.1a) we have the

most high-level construction of a blockchain. Each of the color boxes are the core concepts to construct a blockchain, we'll call these our 'ingredients'. So we have five ingredients in total:

1. **Hash functions (H) and cryptographic signatures (S)** (blue boxes) - The reason that these two concepts can be labeled under the same ingredient is because signatures use hashes in their internal mechanism. Each of these are crucial and highly used in many other parts of the blockchain
2. **Transaction (Tx)** (red box)- A transaction allows to change data, ie. move crypto value from one party to another
3. **UTXO Set & Ledger** (green boxes) - The aggregation of outputs for tracking global and local state
4. **Block (B)** (orange box) - The container of transactions, verification of consensus, and compressed record of history
4. **Protocol (Π)** (purple box) - The formal system by which nodes reach agreement
5. **Network (*N*)** (gray box) - The system by which nodes communicate and ultimately facilitate agreement

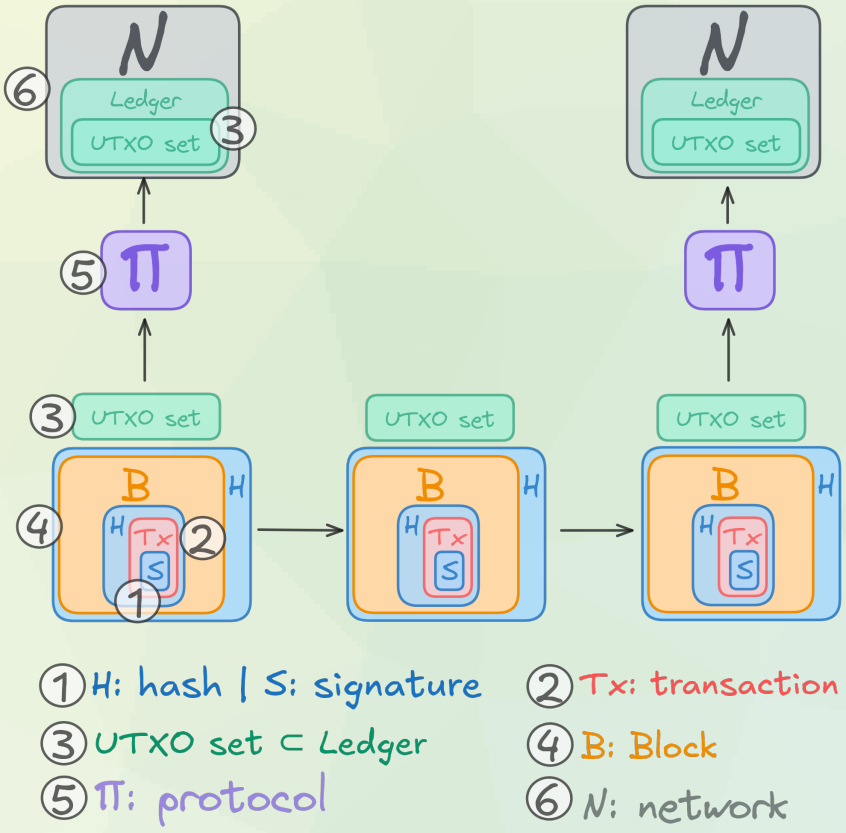

Figure 1.2: Simplified view of blockchain structure based on the composition of our simple blockchain ingredients.

Five components (ingredients) make any blockchain (recipe)

Despite the variability in the design of UTXO-based chains, as displayed by different UTXO Alliance members, all blockchains have the same essential components.

Lets now explore each ingredient one by one.

Cryptographic Primitives

Cryptographic primitives form the foundation of blockchain security and functionality. Two crucial elements are hash functions (H) and digital signatures ($S = \mathrm{Gen}$).

Hash functions are one-way mathematical operations that convert any input into a fixed-size output; see Fig. 1.3 (top). In blockchains, they create unique identifiers, link blocks, and support consensus mechanisms such as Proof-of-Work.

Digital signatures Fig. 1.3 (bottom) are generated using hash functions and provide identity in blockchain systems. Unlike traditional systems where identity is tied to government-issued documents, blockchain identities can be pseudonymous, offering a new paradigm for verification and authentication in digital networks.

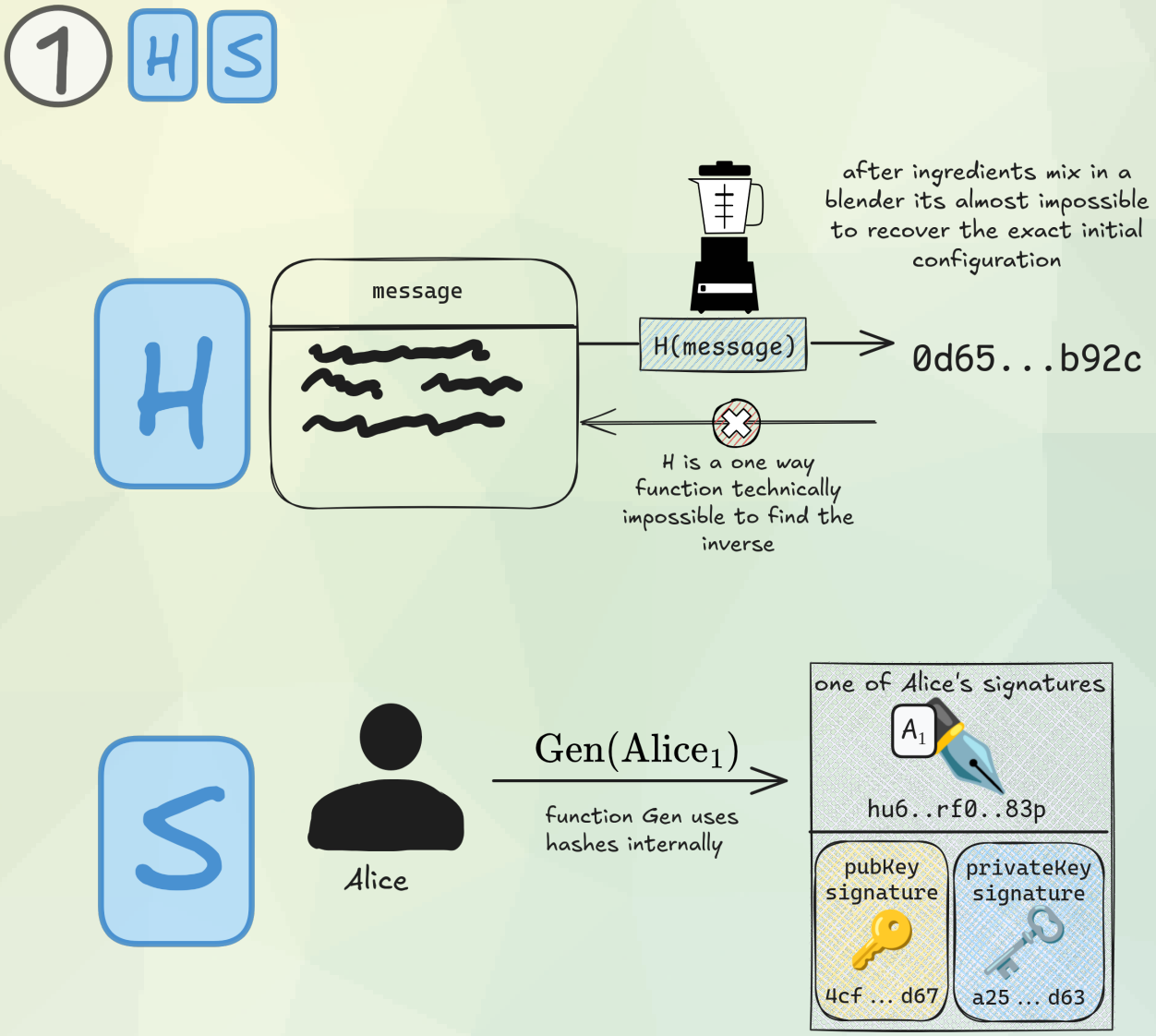

Figure 1.3: We can see that our label ingredient H (top) refers to cryptographic hash functions which are one-way functions. On the other hand our label ingredient S (bottom) refers to digital signatures, where a private number is transformed into a key pair, one capable of being shared publicly and one forever kept private and secure. The public key can be used like a mailing address or identity, and the private key can perform operations privately which confirm approval, ownership and identity. This functionality is used in many internet and computer protocols besides blockchains.

TRANSACTIONS

In the UTXO model, transactions function like a document that transfers ownership of digital assets, secured through cryptographic signatures. In other words transactions are the basic units of state/value transfer in a blockchain.

a) Structure: Fig. 1.4 transactions consist of inputs (black arrows) and outputs (red arrows). b) Inputs: Reference previous transaction outputs (UTXOs) being spent. c) Outputs: Specify new UTXOs being created, including recipient addresses and amounts. d) Conservation Law: The sum of inputs must equal or exceed the sum of outputs (minus a transaction fee). e) Signatures: Each input must be signed by the owner of the corresponding UTXO.

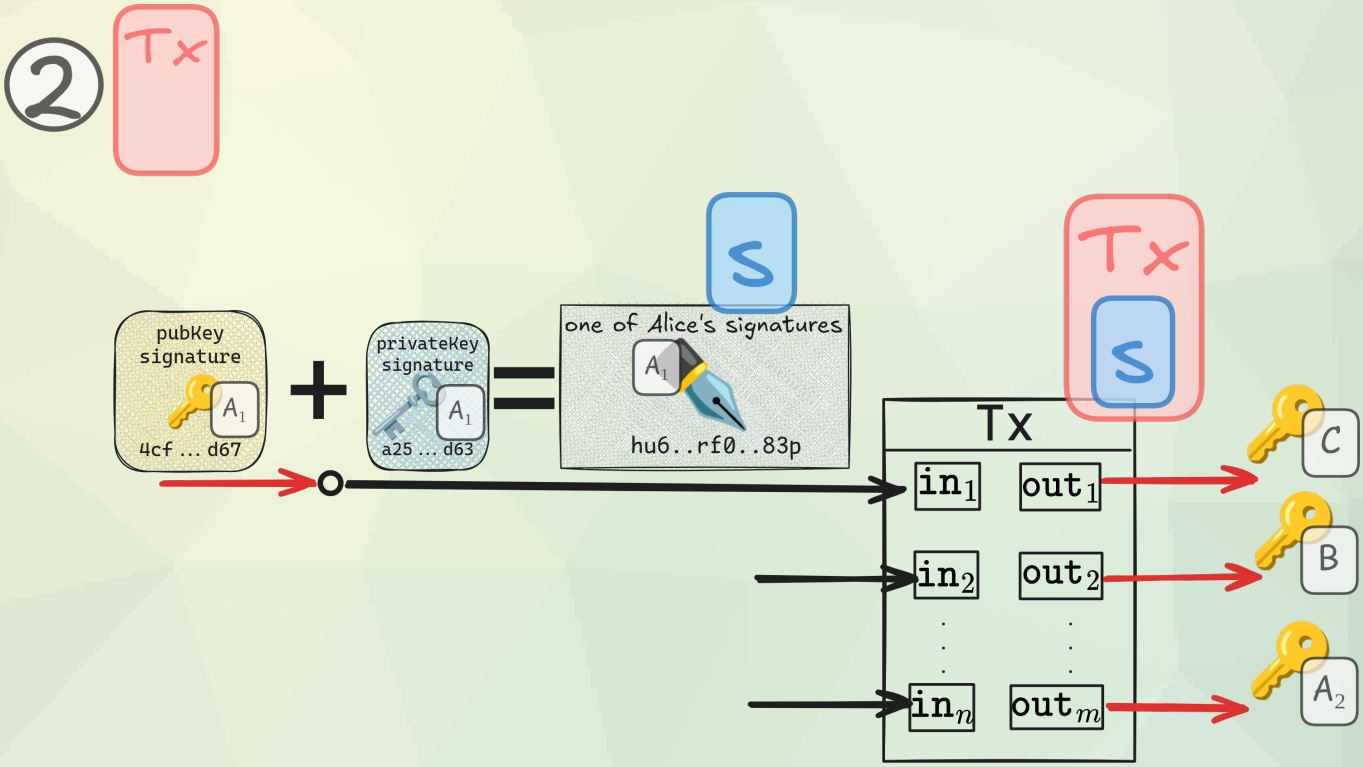

Figure 1.4: Anatomy of a UTXO Transaction. The diagram illustrates the creation of a valid transaction through cryptographic signing, where a combination of public and private key signatures generates a unique transaction signature. The transaction consumes inputs and creates multiple outputs, each secured with recipient public keys, demonstrating the fundamental structure of the UTXO model in blockchain systems.

The diagram shows the creation of a transaction (labeled as Tx) that consumes an input and creates multiple outputs. On the left side, we see the essential components needed to create a valid transaction signature. The process begins with two key elements: a public key signature (shown in yellow with identifier A_1 and a hash starting with "$4cf$") and its corresponding private key signature (shown in blue with the same identifier A and a hash starting with "$a25\ldots$"). When these two elements are combined, they produce one of Alice's signatures (represented by the fountain pen icon), which generates a unique signature hash (starting with "$u6\ldots rf0\ldots 83$").

The transaction itself is depicted as a rectangle with multiple inputs ($in_1, in_1, \ldots, in_n$) and outputs ($out_1, out_2, \ldots, out_m$). Each output is associated with a key, represented by the yellow key icons labeled C, B, and A_2. The transaction is marked with both a Tx identifier and the signature S (shown in the overlapping pink and blue rectangles), indicating that it has been cryptographically signed.

The red arrows in the diagram represent the flow of the transaction, showing how inputs are consumed and new outputs are created. This structure ensures that once a transaction output is spent, it cannot be used again, maintaining the integrity of the blockchain's ledger system.

The UTXO Set and Ledger

The UTXO set plus additional data is what ultimately the Ledger stores. The slicing of UTXO selection is what a user's wallet performs in order to determine how many outputs an address holds. The Ledger refers to the long running concatenation of transactions via blocks, the UTXO set represents the proverbial "tip of the chain" which is the most relevant perspective of current state. See Fig. 1.5

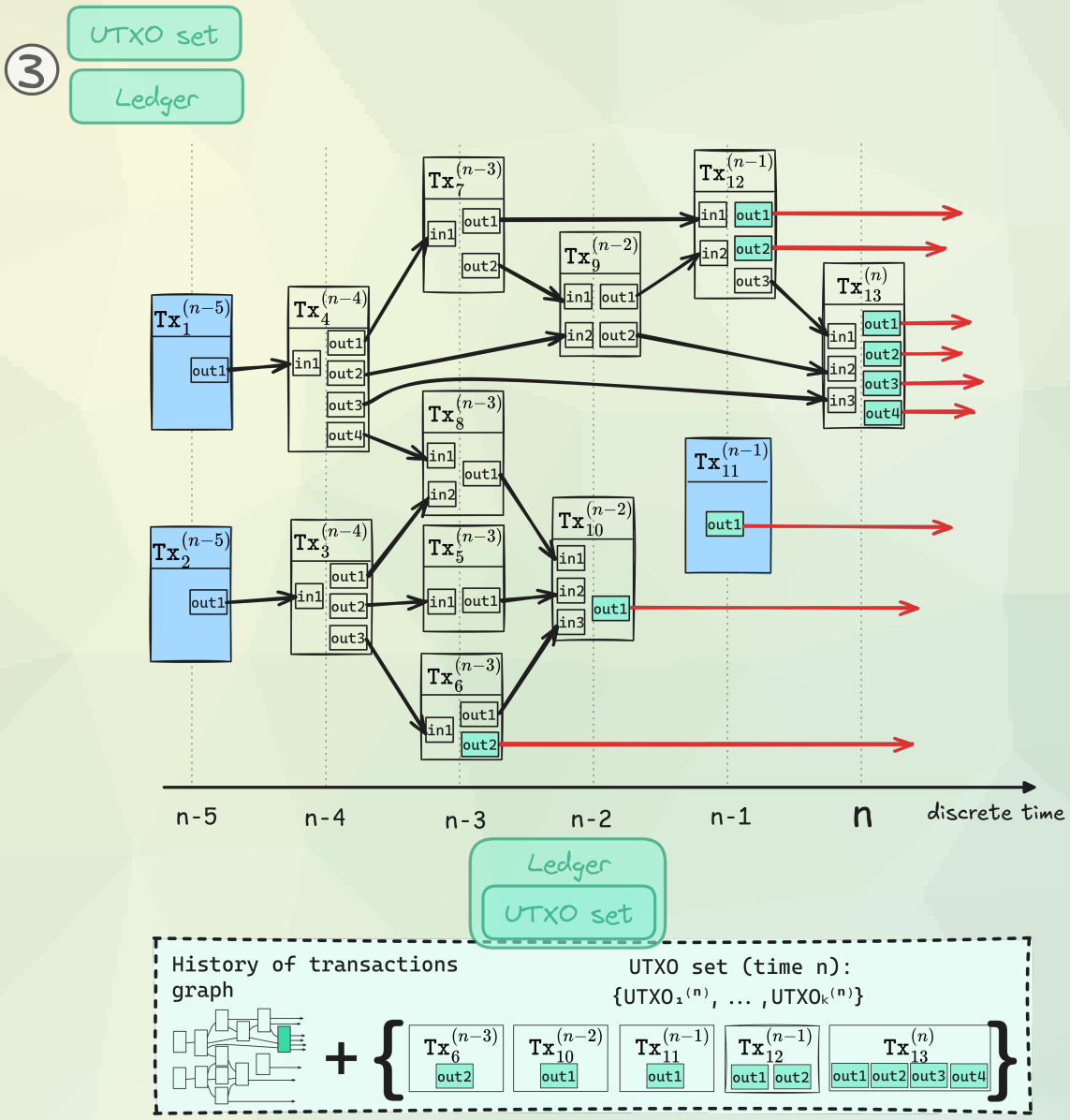

Figure 1.5: UTXO Set Evolution and Ledger State. The diagram illustrates the temporal progression of blockchain transactions as a directed acyclic graph (top), while showing how the current state is maintained through the UTXO set (bottom). The combination of historical transaction graph and current UTXO set forms the complete ledger, providing both historical accountability and current ownership status. Time progression from n-5 to n demonstrates how transactions consume previous outputs and create new ones, with green-highlighted outputs representing the current unspent transaction outputs.

The diagram presents a sophisticated visualization of how blockchain transactions form a directed acyclic graph (DAG) over time and how this structure relates to the current state of ownership through the UTXO set. Let me break this down into interconnected concepts.

The upper portion of the diagram shows transactions arranged in discrete time steps from $n-5$ to n, where n represents the current moment. Each transaction (depicted as a box with inputs and outputs) connects to other transactions through a directed graph structure. The blue-colored transactions at

time $n-5$ represent older transactions and the blue color Txs are coinbase transactions ie. transactions that initiate unspent outputs (only outgoing arrows) and consume no inputs (incoming arrows). While the progression toward time n shows how newer transactions consume previous outputs and create new ones.

The directed edges (arrows) in the graph demonstrate the fundamental principle of the UTXO model: each transaction output can only be spent once as an input to a subsequent transaction. For example, $\text{Tx}_4^{(n-4)}$ takes inputs and creates multiple outputs, some of which are later consumed by $\text{Tx}_7^{(n-3)}$ and $\text{Tx}_8^{(n-3)}$. This creates a natural ordering of transactions and prevents double-spending, as each output can only appear once as an input in the entire history.

The bottom portion of the diagram introduces two crucial concepts: the transaction history graph and the UTXO set at time n. The UTXO set, highlighted in mint green, represents the current state of ownership in the system. It contains only the unspent transaction outputs - those that haven't yet been used as inputs to other transactions. In the diagram, these are marked as $\left\{\text{UTXO}_1^{(n)}, \ldots, \text{UTXO}_k^{(n)}\right\}$ and specifically includes outputs from transactions Tx_6, Tx_{10}, Tx_{11}, Tx_{12} and Tx_{13}.

The ledger, which maintains the truth of ownership, is effectively represented by both components: the historical transaction graph (showing how ownership has changed over time) and the UTXO set (showing current ownership status). This dual representation is powerful because while the complete transaction history provides auditability and proof of ownership transitions, the UTXO set provides an efficient way to verify new transactions by checking if they're attempting to spend outputs that actually exist and remain unspent.

The green-colored outputs in transactions like $\text{Tx}_{13}^{(n)}$ indicate they are currently part of the UTXO set - these represent the "live" state of the system, or in other words, the current distribution of ownership. Any valid new transaction must reference these unspent outputs as inputs, thereby ensuring that the system maintains a consistent and verifiable state of ownership at all times.

Block

Blocks are containers for transactions and form the backbone of the blockchain: Structure: Typically includes a header and a list of transactions. b) Block Header: Contains metadata such as:

- Previous block hash (creating the chain)
- Merkle root of transactions
- Timestamp
- Nonce for consensus protocols such as Proof-of-Work (PoW). PoW is a mechanism to make block creation computationally expensive, ensuring security.
- Coinbase Transaction: A special transaction in each block that creates new currency and collects transaction fees.

At the heart of every block is the block header, which contains three essential elements: a reference to the previous block, a Merkle root, and a nonce. What makes this structure particularly elegant is how it uses a Merkle tree to efficiently organize and verify the transactions contained within the block, see Fig. 1.6.

The Merkle tree, shown in the central portion of the diagram, is a binary hash tree that provides a cryptographically secure way to summarize all transactions in the block. At the bottom level, we see individual transactions (Tx_1 through Tx_k) represented with fountain pen icons, indicating that each transaction is signed. Each transaction is first hashed (shown as $H(\text{Tx}_1)$, $H(\text{Tx}_2)$, etc.), creating the leaf nodes of the tree.

These transaction hashes are then paired and combined through a hash function H, creating parent nodes of the form $H(H_1, H_2)$. This pairing and hashing process continues up the tree, level by level, until we reach a single hash value at the top - the Merkle root. This root, stored in the block header, serves as a cryptographic fingerprint of all transactions in the block.

The significance of this structure becomes apparent when we consider the block's relationship to the wider blockchain. The block header, shown at the top of the diagram, contains not only the Merkle root but also a reference to the previous block (creating the chain) and a nonce (used in the mining process). The presence of a previous block reference ensures the temporal ordering of blocks, while the Merkle root ensures the integrity of all transactions within the block.

The expanded detail on the left side of the diagram (marked with 'B') shows how this block structure relates to the transaction components we saw earlier. Each transaction (Tx) within the block maintains

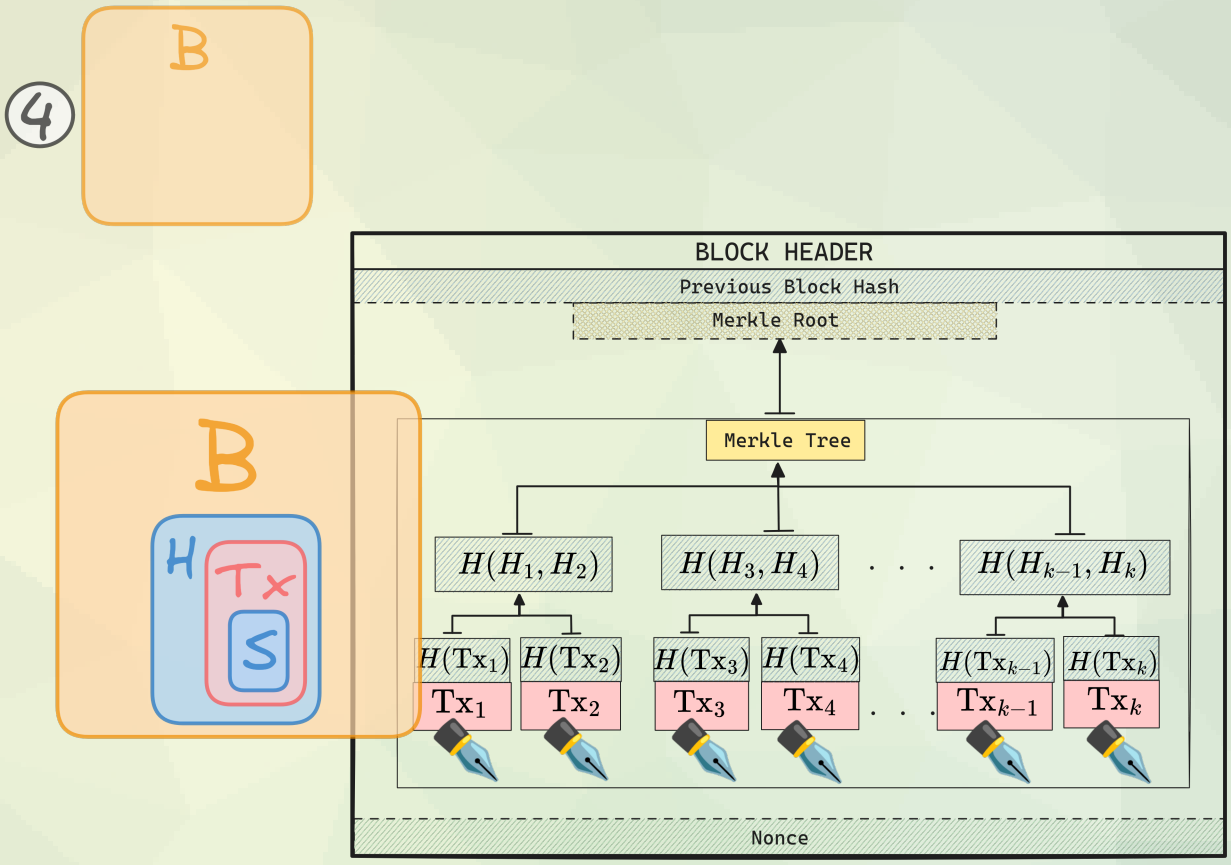

Figure 1.6: Block Structure and Merkle Tree Organization. The diagram illustrates how individual transactions are organized within a block using a Merkle tree structure. The block header contains the Merkle root, which serves as a cryptographic summary of all transactions, along with references to the previous block and a nonce. The hierarchical hashing scheme demonstrates how transaction integrity is maintained while enabling efficient verification through Merkle proofs.

its signature (S) and hash (H), ensuring that the integrity and authenticity of individual transactions are preserved even when packaged together in a block.

This hierarchical structure provides remarkable efficiency benefits. To prove that a particular transaction is included in a block, one only needs to provide the transaction itself and a small set of hashes (known as a Merkle proof) - typically $\log_2(n)$ hashes for n transactions, rather than the entire set of transactions in the block. This property makes blockchains particularly suitable for light clients that need to verify transactions without downloading the entire blockchain.

The Chain

Finally we've arrived to the chain, the fundamental structure of a blockchain! It is nothing more than a sequence of cryptographically linked blocks that creates an immutable record of transactions over time. Lets dive into how this chain structure works and then briefly touch on the remaining ingredients ie. the **Protocol** and the **Network**.

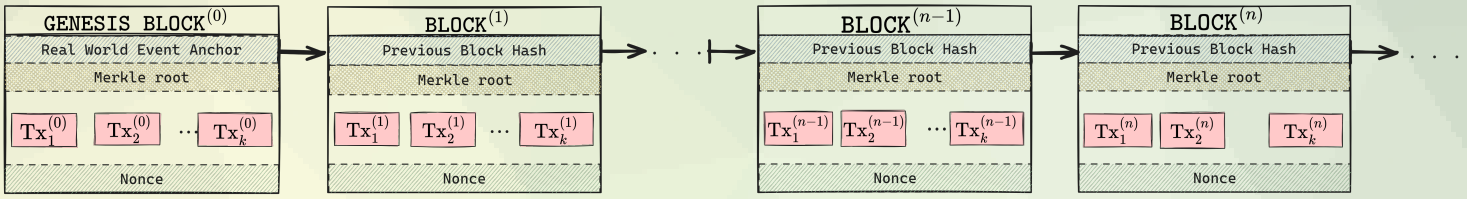

Figure 1.7: An irreversible sequence of blocks linked by a pointer reference to a previous Block hash ID.

The chain begins with a special block called the Genesis Block ($\texttt{BLOCK}^{(0)}$), which serves as the foundation of the entire blockchain. What makes the Genesis Block unique is its inclusion of a "Real World Event Anchor" instead of a previous block hash. This anchor could be anything from a newspaper headline to a timestamp, providing a verifiable connection to the physical world and marking the blockchain's beginning.

From there, each subsequent block ($\texttt{BLOCK}^{(1)}$ through $\texttt{BLOCK}^{(n)}$) maintains four essential components: a hash of the previous block, a Merkle root summarizing all transactions within the block, the actual transactions $\{\text{Tx}_1, \ldots, \text{Tx}_k\}$, and a nonce value. The previous block hash is crucial as it creates the "chain" in blockchain. Each block references its predecessor through this hash link. If any data in a block were to change, it would alter that block's hash, breaking the chain of references and making the tampering evident.

The progression through time is shown by the succession of blocks, with each new block ($\texttt{BLOCK}^{(n)}$) building upon the history established by all previous blocks. The arrows between blocks represent this temporal and cryptographic relationship, creating an irreversible chain of custody for all transactions since its inception.

Regarding the remaining components of a blockchain system - **Protocol** and **Network** - these operate at a higher level of abstraction. The Protocol component defines the rules of the system: how nodes reach consensus on the valid state of the blockchain, how new blocks can be added, and what constitutes a valid transaction. It's essentially the "constitution" of the blockchain system, encoding all the rules that participants must follow.

The Network component represents the peer-to-peer infrastructure through which blockchain nodes communicate. While conceptually simple - computers connecting and sharing data via TCP/IP protocols - it's this network layer that enables the decentralized nature of blockchain systems, allowing participants to maintain synchronized copies of the ledger without requiring a central authority.

Chapter 2: The Blockchain

Bitcoin's UTXO model - the vanilla flavor of UTXOs

Bitcoin, introduced by Satoshi Nakamoto in 2008, aimed to create a decentralized digital currency system that could operate without the need for intermediaries such as banks or governments. Its primary goal was to enable peer-to-peer electronic transactions in a trustless environment, solving the double-spending problem through a distributed ledger (blockchain) and a consensus mechanism (Proof of Work).

However, Bitcoin's groundbreaking design also came with limitations. Its relatively simple scripting language limits complex smart contract functionality. Scalability issues, evident in low transaction throughput and high fees during network congestion, hinder its use for everyday transactions. Bitcoin's Proof-of-Work consensus, while secure, is energy-intensive and leads to mining centralization. Additionally, Bitcoin's fixed monetary policy, while appealing to some, lacks the flexibility to adapt to varying economic conditions. These limitations have inspired a new generation of blockchains to explore alternative consensus mechanisms, more expressive smart contract capabilities, improved scalability solutions, and innovative governance models.

The diagram illustrates the fundamental architecture of Bitcoin's blockchain system, specifically focusing on how transactions are processed and stored. Starting from the left side, we see the concept of Unspent Transaction Outputs (UTXOs), which are essentially records of bitcoin amounts that can be spent. These UTXOs are unlocked using a private key's digital signature, represented by the blue key icon, allowing the creation of new transactions.

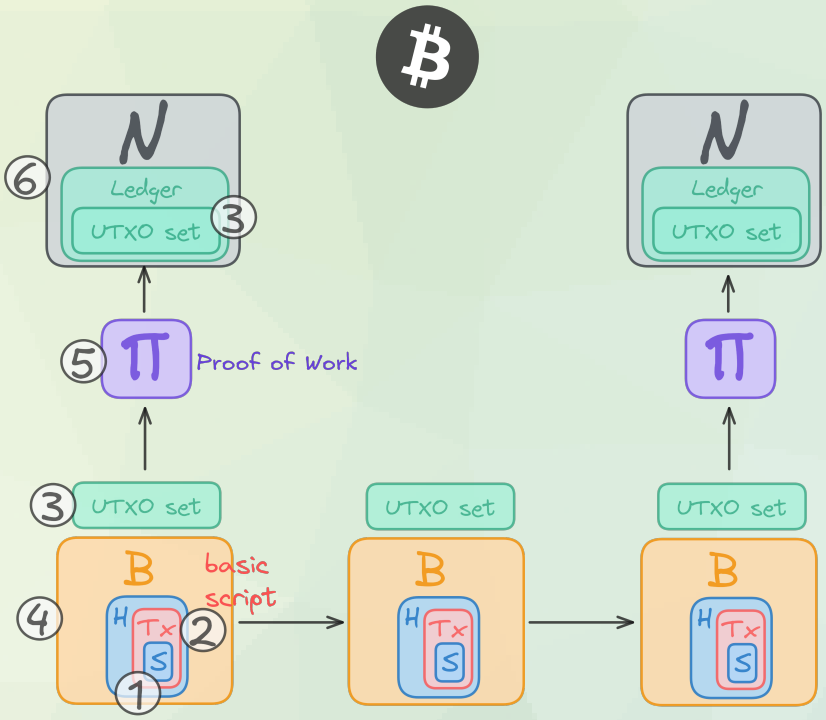

Figure 2.1: The Bitcoin blockchain viewed as simple composition of our ingredients.

The diagram illustrates the fundamental architecture of Bitcoin's blockchain system, specifically focusing on how transactions are processed and stored. Starting from the left side, we see the concept of Unspent Transaction Outputs (UTXOs), which are essentially records of bitcoin amounts that can be spent. These UTXOs are unlocked using a private key's digital signature, represented by the blue key icon, allowing the creation of new transactions. Moving to the center, each transaction (Tx) contains

inputs and outputs. The inputs reference previous UTXOs that are being spent, while the outputs create new UTXOs that can be spent in future transactions. These new UTXOs are locked by public key signatures (shown by the yellow key), ensuring only the intended recipient can spend them. The right side of the diagram shows how these transactions are organized into blocks. Each block contains multiple transactions arranged in a Merkle tree structure, which is an efficient way to verify transaction integrity. The Merkle tree starts with individual transaction hashes at the bottom, then pairs them up and hashes them together repeatedly until reaching a single Merkle root at the top.

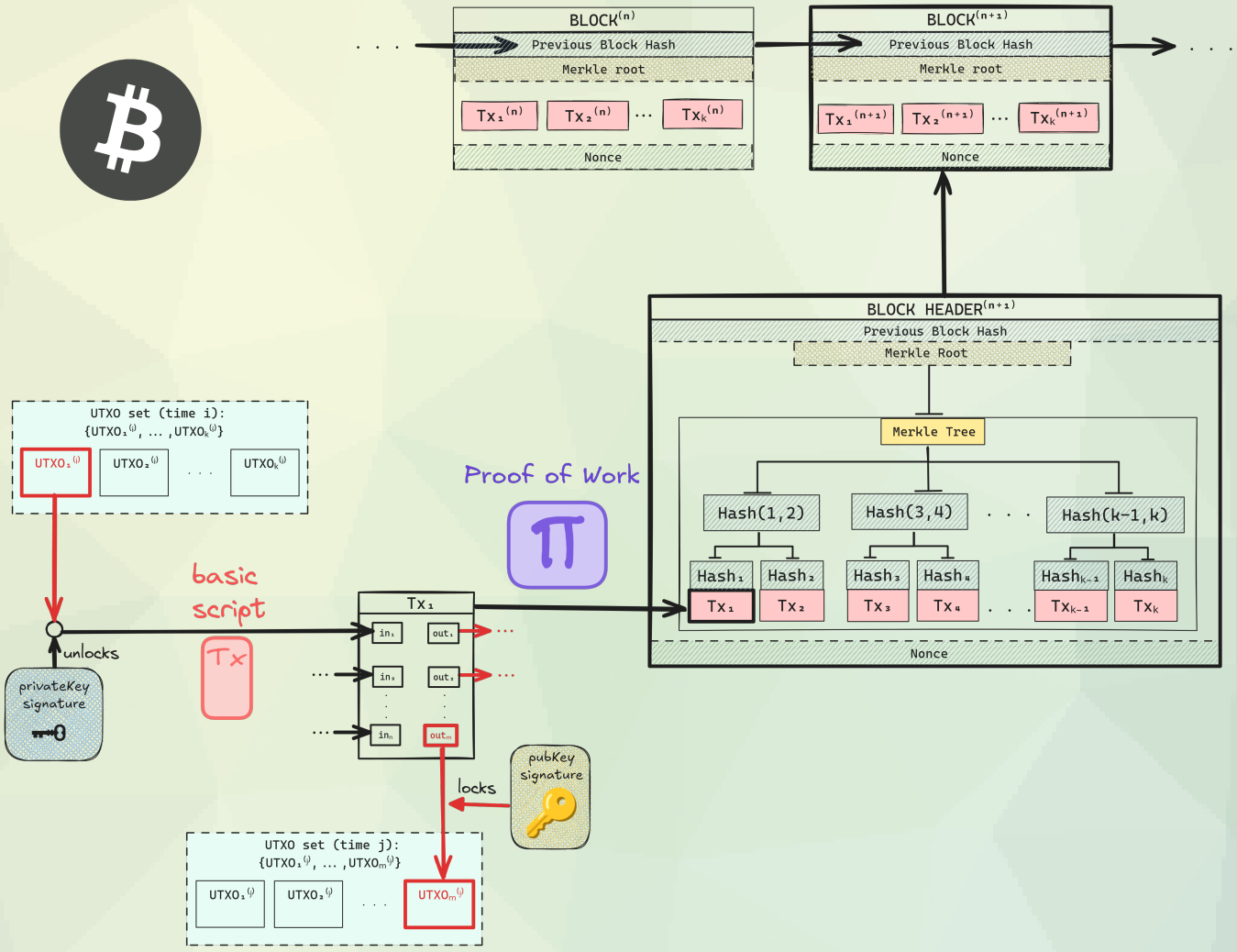

Figure 2.2: Architectural diagram of Bitcoin's blockchain illustrating the UTXO transaction model and block structure. The diagram flows left-to-right and bottom-to-top, showing how transactions originate from UTXOs (unlocked by private keys), get verified through cryptographic signatures, and are organized into blocks using Merkle trees. Each block links to its predecessor through a hash reference and includes a nonce for Proof of Work consensus, forming an immutable chain.

The upper portion of the diagram demonstrates how blocks are chained together. Each block $(n+1)$ contains a reference to the previous block's hash, creating an immutable chain of records. Each block also includes a nonce value, which is crucial for the Proof of Work consensus mechanism (indicated by the purple Π symbol). This mechanism requires miners to find a nonce that, when combined with the block's contents, produces a hash meeting certain difficulty criteria.

DIGIBYTE

DigiByte is one of the longest-running blockchain platforms, launched in 2014 as a secure, fast, and highly decentralized blockchain focusing on digital payments and asset transfers. The platform is notable for its implementation of five distinct mining algorithms and its advanced difficulty adjustment system.

At its core, DigiByte employs a multi-algorithm mining approach called MultiAlgo. This system uses five different mining algorithms (Scrypt, SHA256, Qubit, Skein, and Odocrypt) operating simultaneously, which helps maintain decentralization by preventing any single type of mining hardware from dominating the network. This approach also provides enhanced security against 51The blockchain architecture features a three-layer system:

- Core Protocol Layer - handling network operations and security
- Digital Asset Layer - managing transfers and data
- Applications Layer - supporting decentralized applications

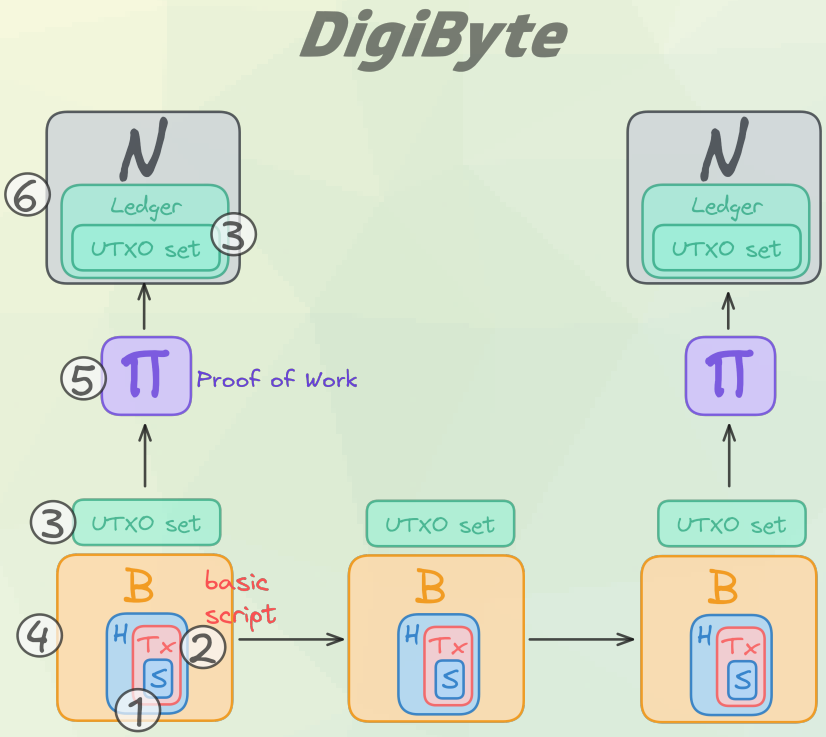

Figure 2.3: Simple Digibyte blockchain schematic.

DigiByte's block time is notably fast at 15 seconds, significantly quicker than Bitcoin's 10 minutes. This rapid block time, combined with the MultiAlgo system, allows for quick transaction confirmations while maintaining security. The platform implements MultiShield real-time difficulty adjustment for each algorithm, occurring every block rather than at longer intervals.

The network's scaling approach includes SegWit, which allows for effective block size usage. This has enabled DigiByte to maintain high transaction throughput while keeping fees low.

The Extended UTXO Model - Cardano

The Extended UTXO (EUTXO) model is a significant enhancement of the traditional Unspent Transaction Output (UTXO) model, originally used in Bitcoin. Cardano's EUTXO model introduces additional functionalities that facilitate more complex smart contracts while maintaining the benefits of the UTXO paradigm. This model allows for better scalability and flexibility in transaction processing. The extended part refers to the fact that each UTXO can now carry: *(i)* A value (ADA and native tokens), *(ii)* a datum (arbitrary data) and *(iii)* a validator script (the smart contract logic).

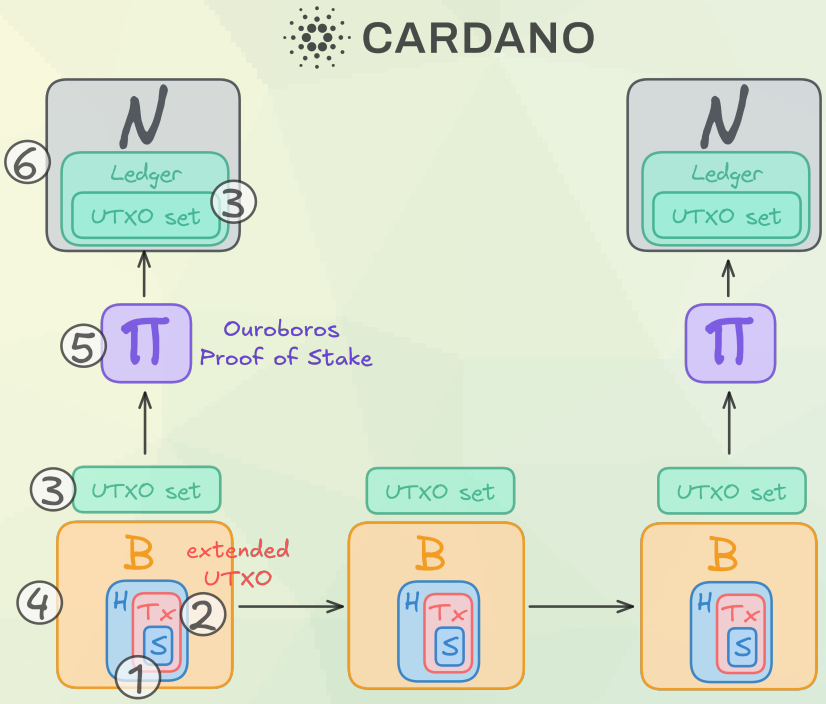

Figure 2.4: Simple Cardano blockchain schematic.

Key Features of the EUTXO Model:

1 *Enhanced Data Handling* - In the EUTXO model, each UTXO can carry additional data known as datum. This data can be used to store information relevant to a transaction, such as smart contract state or other contextual information. The inclusion of datum allows developers to implement more sophisticated logic within transactions without compromising the integrity and simplicity of the UTXO model.

2 Redeemer Context - The redeemer context is a critical aspect of the EUTXO model that specifies how a datum is utilized during transaction validation. When a transaction attempts to spend a UTXO, it must provide a corresponding redeemer that indicates how the datum should be interpreted and used. This mechanism ensures that only valid transactions can access and manipulate the associated datum, adding a layer of security and correctness to smart contract execution.

3 *Transaction Validation* - In the EUTXO model, each transaction is validated based on its inputs (the UTXOs being spent), the associated datum, and the redeemer provided. This validation process checks whether the redeemer correctly corresponds to the expected operations defined by the datum. Thus, it allows for complex interactions while ensuring that all conditions are met before a transaction is executed.

Extended UTXO model (EUTXO)

Script Attachment: In the EUTXO model, scripts (or smart contracts) are attached directly to outputs. This is represented by the 'Script' component in the Output structure. This allows for more complex validation logic to be associated with each UTXO.
Datum and Redeemer: (i) Datum: This is arbitrary data that can be attached to an output. It represents the state of the script and is stored on-chain. (ii) Redeemer: This is provided by the transaction that wants to spend a UTXO. It contains the arguments for the script execution.

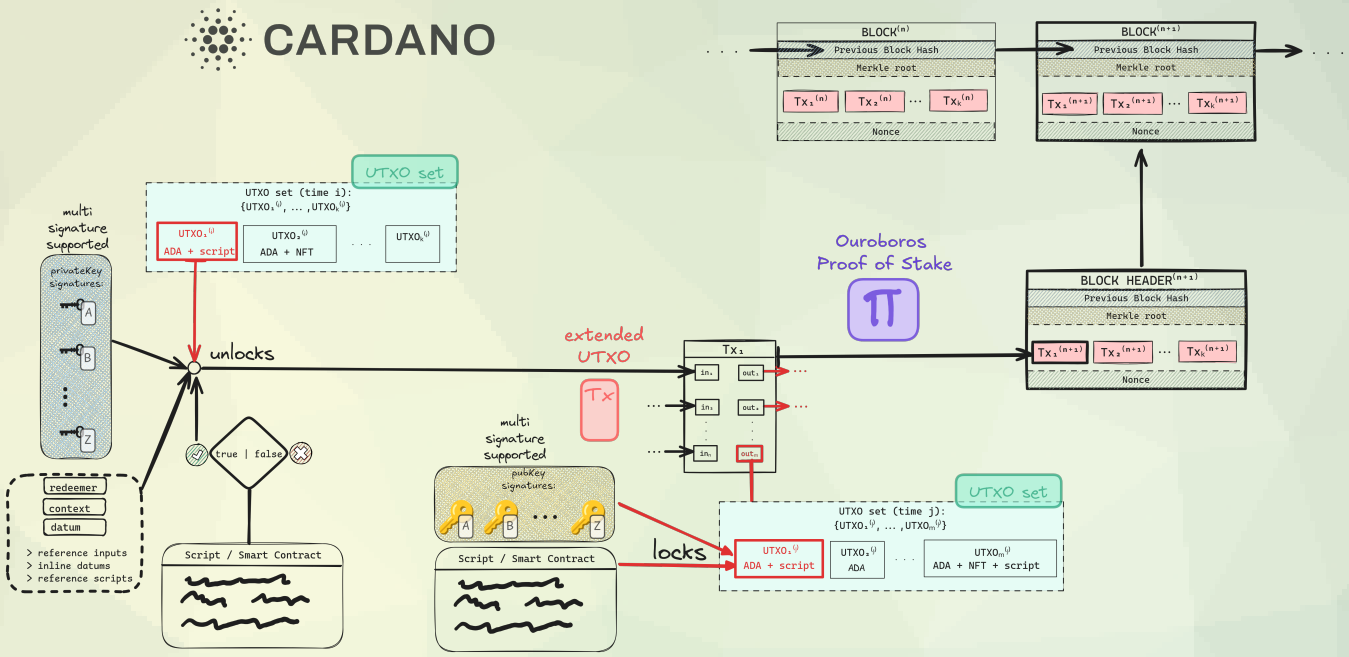

Figure 2.5: A detailed view of the Cardano blockchain and its extended UTXO.

Context-Aware Validation: The EUTXO model provides rich context to the validator script, including information about the entire transaction and even parts of the blockchain state. This is represented by the 'Context' in the diagram.
Validity Interval: Transactions in the EUTXO model can specify a validity interval (represented by 'Slot Height' in the diagram). This allows for time-sensitive smart contracts.
Script Execution: The validator function takes three inputs: Datum, Redeemer, and ScriptContext. It returns a boolean indicating whether the transaction is valid or not.

Key Advantages of EUTXO:

1 Increased Expressiveness: The addition of Datum and Redeemer allows for more complex state management in smart contracts.
2 Local State Validation: Each input can be validated independently, which allows for better parallelization and scalability.
3 Predictability: The outcome of script execution can be predicted off-chain, reducing the risk of failed transactions.
4 Fee Prediction: The deterministic nature of script execution allows for accurate fee prediction.1.
5 Enhanced Privacy: The UTXO model inherently provides better privacy compared to account-based models.
6 Time-Sensitive Logic: The validity interval allows for time-based contract logic.

The Extended UTXO Model - Ergo

The Ergo blockchain implements smart contracts through its Extended UTXO (eUTXO) model, which enhances the basic UTXO model used in Bitcoin with additional programmability. Here's how the key components work together:

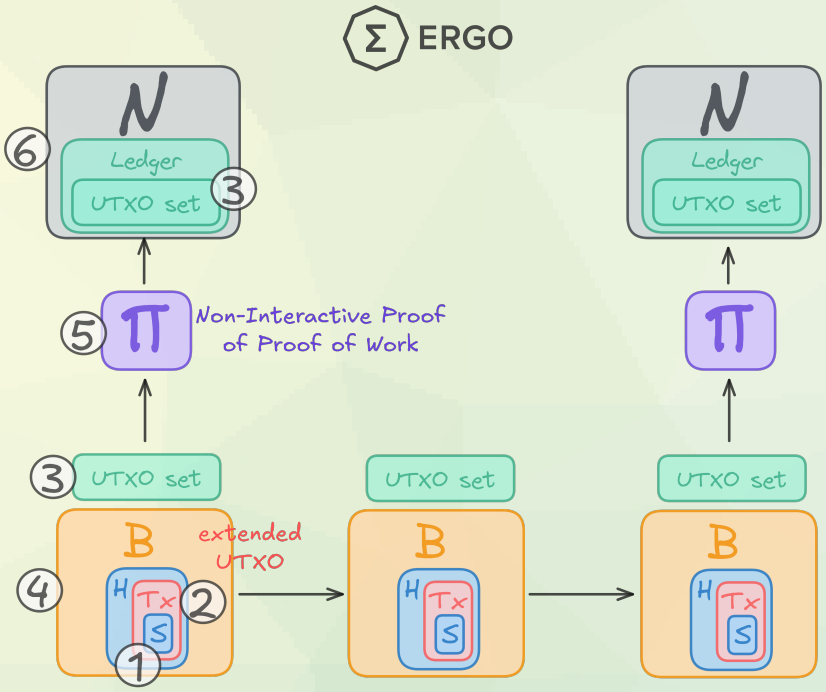

Figure 2.6: Simple Ergo blockchain schematic.

Ergoś EUTXO Model:
The eUTXO model extends the basic UTXO model by adding contextual information and validation logic to transaction outputs. Each output can carry both a value and arbitrary data, allowing for more complex contract conditions. This is fundamentally different from Ethereum's account-based model and provides several advantages for smart contract execution.

Context: Represents the current transaction's environment and blockchain state. Includes information like block height, timestamps, and transaction inputs/outputs. Allows contracts to make decisions based on network conditions. The context is essential for time-locked contracts and other condition-based executions
Datum: Represents the persistent state stored with each UTXO, it contains arbitrary data that can be used to track contract state. It acts like a "memory" for the smart contract and can store things like ownership information, voting results, or game state
Redeemer: Provides the arguments needed to spend a UTXO. Contains the input parameters that satisfy the contract's conditions and is used to prove that spending conditions are met. The redemer can include signatures, proofs, or other validation data

Multisignature Implementation - Ergo's eUTXO model enables sophisticated multisignature schemes through:

Script Creation: Multiple public keys can be included in the contract script. The script can specify the required number of signatures (m out of n signatures). Different spending conditions can be combined with boolean logic
Signature validation: The redeemer must provide the required number of valid signatures. The context ensures signatures match the specified public keys. Note that additional conditions can be added (time locks, custom logic)
Flexibility: Supports both threshold signatures and more complex arrangements which can combine multisig with other contract conditions. Allows for dynamic participant sets through contract logic

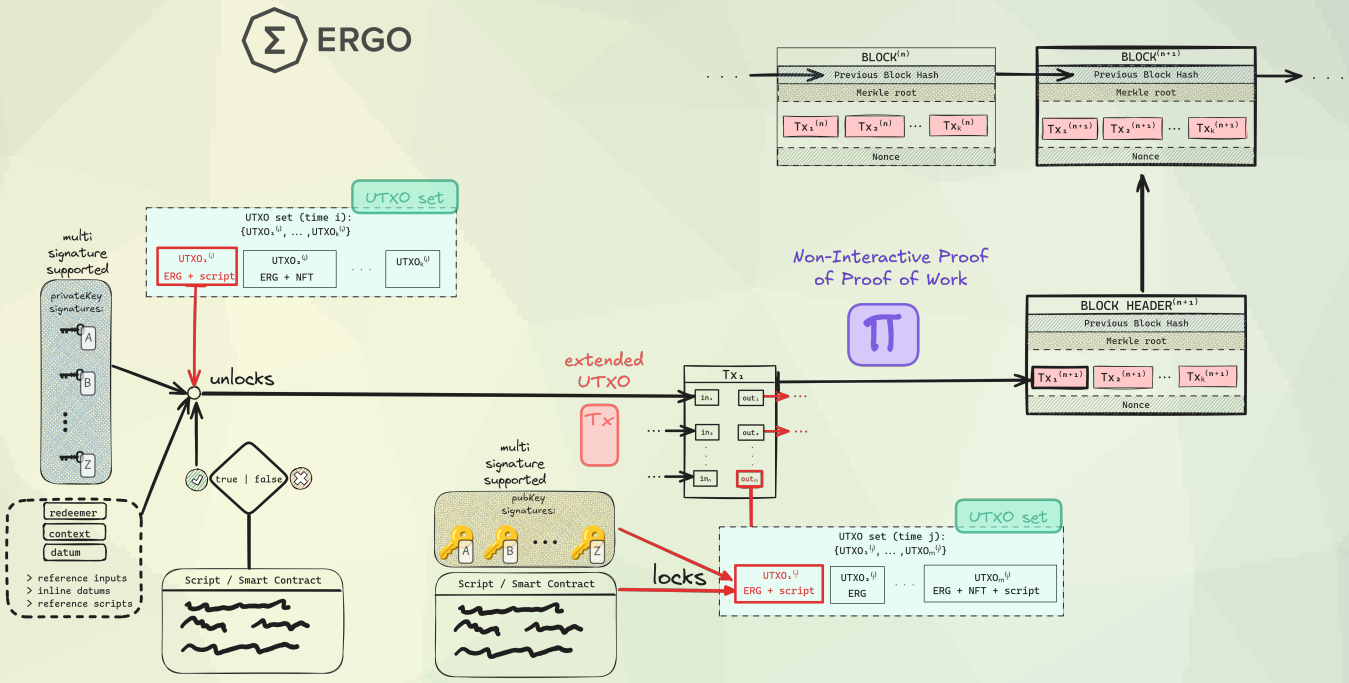

Figure 2.7: Architectural overview of Ergo blockchain's Extended UTXO model. The diagram demonstrates how Ergo enhances the basic UTXO model with smart contract capabilities through datum and context-aware transaction validation. It shows the flow from smart contract execution through transaction creation to block formation, highlighting the ability to handle multiple asset types (ERG, NFTs, tokens) within the same UTXO structure.

The diagram illustrates Ergo's Extended UTXO (eUTXO) model, which builds upon Bitcoin's foundation while introducing sophisticated smart contract capabilities. The key difference lies in how transactions are processed and validated. In Ergo, each UTXO includes additional components: datum (which stores state), context (providing runtime information), and redeem conditions. These elements allow for more complex transaction logic compared to Bitcoin's simpler script-based approach.

The left side of the diagram shows how a transaction's validity is determined through a combination of cryptographic verification (private key signature) and smart contract execution. The smart contracts, represented by script boxes at the bottom, can access the datum and context information to make decisions about whether a transaction should proceed. This creates a more flexible and programmable system than Bitcoin's basic scripting.

The transaction structure maintains the UTXO model's core principle of consumed inputs and generated outputs, but extends it by allowing outputs to carry not just ERG (the native token) but also NFTs and custom tokens (shown as "ERG + NFT" in the UTXO set). The proof-of-work consensus mechanism remains similar to Bitcoin's, though labeled as "Non-Interactive Proof of Work" to highlight Ergo's specific implementation.

This design allows Ergo to support complex smart contracts while maintaining the security and scalability benefits of the UTXO model. The combination of Context, Datum, and Redeemer provides the necessary components for sophisticated contract logic, while the eUTXO model ensures efficient and predictable execution.

Common Knowledge Base (CKB) - Nervos Network

Nervos Network, launched in 2019, aims to solve blockchain trilemma issues (scalability, security, and decentralization) through a unique layered architecture. It seeks to provide a secure foundation for decentralized applications while allowing for scalable solutions on higher layers.

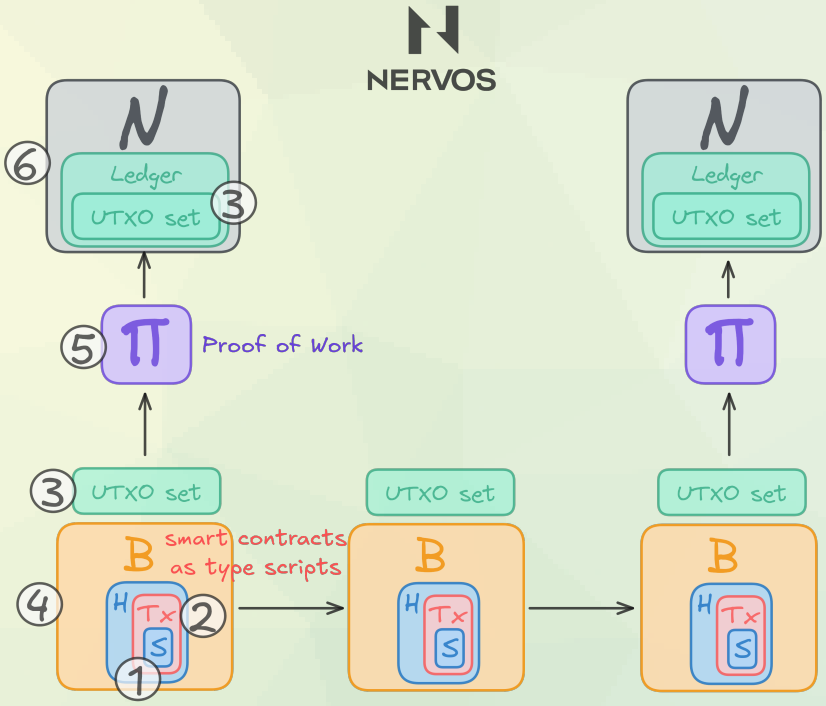

Figure 2.8: Simple Nervos blockchain schematic.

Key features of Nervos Network include:

1 Layered Architecture: Consists of a secure base layer (CKB) and flexible layer 2 solutions for scalability.
2 Cell Model: An enhanced UTXO model that allows for more complex state management and smart contract capabilities.
3 Native Token Economics: CKByte tokens represent state storage on the blockchain, creating an economic model that aligns with long-term network sustainability.
4 Proof-of-Work Consensus: Uses the NC-MAX algorithm, designed to be ASIC-neutral to maintain decentralization. Interoperability: Designed to facilitate cross-chain interactions and serve as a hub for other blockchain networks.
5 Nervos Network aims to provide a versatile platform that can serve as both a secure value storage and a foundation for scalable decentralized applications, addressing limitations in both Bitcoin-like and Ethereum-like blockchains.

Smart contracts in Nervos Network are implemented through a unique model that differs from account-based blockchains like Ethereum. Instead of having smart contract code stored on-chain that operates on state, Nervos uses a cell model with two types of scripts: **Lock-scripts** and **Type-scripts**.

- **Lock-scripts** - Function as ownership validators for cells (similar to Bitcoin's P2PKH scripts). This design allows control on who can spend/use the cells. Lock-scipts are executed during cell consumption that typically verify signatures or other ownership conditions. Must return a true boolean value for the transaction to be valid
  Type-scripts - Define and enforce rules for cell transformations, validate state transitions and implement business logic. Type-scripts are optional but powerful for complex applications that enable implementation of various token standards and DeFi protocols.

The **CKB-VM (Common Knowledge Base Virtual Machine)** - is fundamental to this architecture: It's a **RISC-V** based virtual machine in charge of executing both Lock-scripts and Type-scripts. It provides deterministic execution across the network. An amazing feature is that it supports multiple programming languages since any language that can compile to RISC-V can be used. Moreover, it offers high performance due to its simple instruction set and maintains security through its verified software stack.

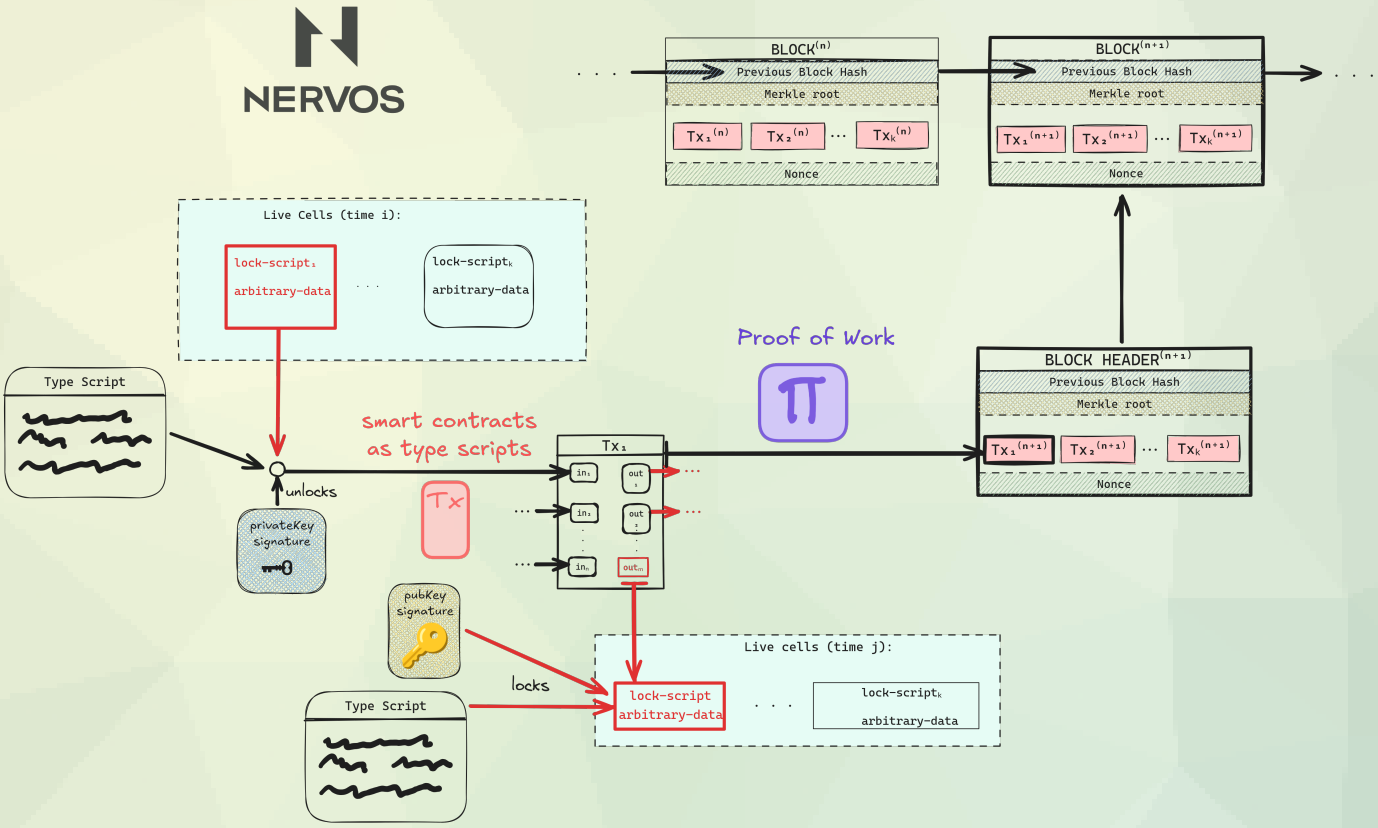

Figure 2.9: Architectural diagram of Nervos CKB's cell-based blockchain model. The illustration demonstrates how CKB's Live Cells combine arbitrary data storage with dual-script validation (lock scripts and type scripts), enabling sophisticated smart contract functionality while maintaining UTXO-like security properties. The diagram flows from cell creation through transaction processing to block formation, highlighting the relationship between data storage, script validation, and consensus mechanisms.

The transaction flow demonstrates how cells are transformed: input cells are consumed while creating new output cells, with both lock and type scripts ensuring transaction validity. When a transaction is initiated: (i) Lock-scripts verify ownership/permissions. (ii) Type-scripts validate state transitions and (iii) CKB-VM executes both script types. The "arbitrary-data" notation within cells emphasizes CKB's capability to store any form of data, making it a versatile platform for diverse applications. Type scripts play a crucial role as smart contracts, validating state transitions and enforcing complex business rules.

For complex applications multiple Type-scripts can be combined. Additionally different cells can have different rules and custom validation logic can be implemented at the cell level.

Quai Network

Quai Network is a decentralized blockchain ecosystem designed to enhance scalability and efficiency in cryptocurrency transactions. Its architecture is built around a multichain structure consisting of three main components: Prime, Paxos, and Cyprus. Additionally, the network employs a unique consensus mechanism called Proof of Entropy Minima (PoEM), which integrates these components into a cohesive system.

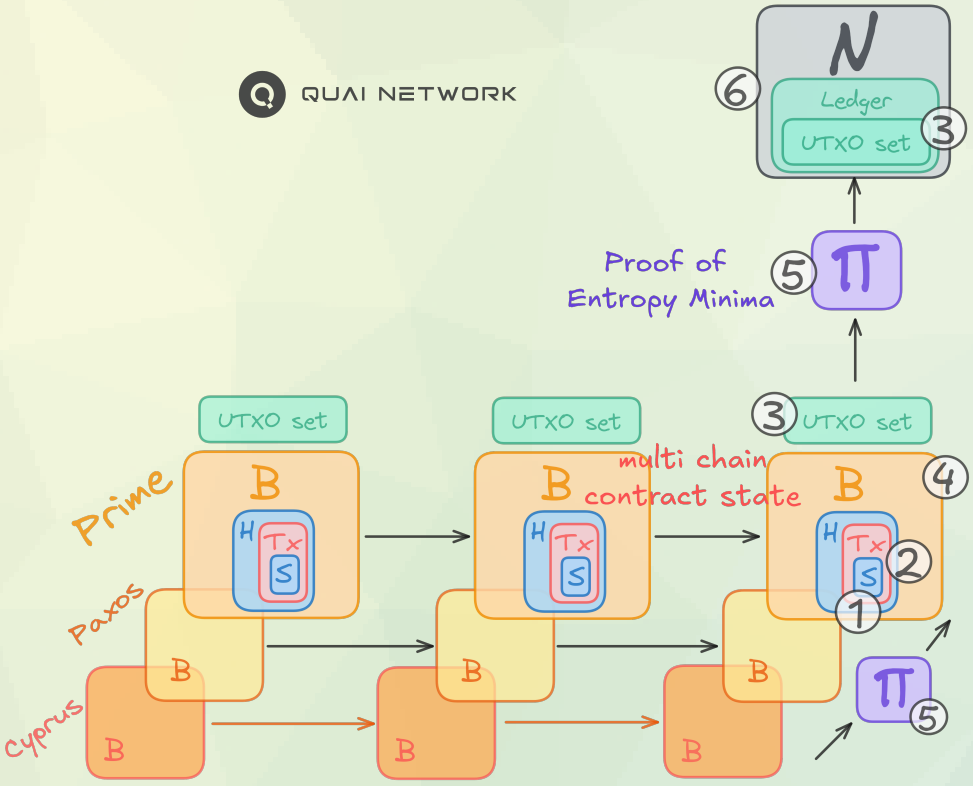

Figure 2.10: Architectural overview of Quai Network's multi-chain system utilizing Proof of Entropy Minima (PoEM) consensus. The diagram illustrates the hierarchical relationship between Prime, Parx, and Cyprus chains, showing how parallel chains maintain independent UTXO sets while achieving network-wide consensus through PoEM.

Quai has an *EVM Chain* component where the Solidity-compatible EVM portion is relatively straightforward, operating similarly to other EVM chains with smart contract capabilities, allowing developers tVo: deploy Solidity smart contracts and use familiar development tools and patterns. On the other hand, Quai has a *UTXO Chain Integration* which is its more novel aspect. In UTXO chains, smart contracts work differently than in account-based models.

Scripts are attached to individual UTXOs rather than residing at fixed addresses. Contract state is managed through UTXO ownership and movement. Validation occurs through script execution when UTXOs are spent.

Multichain Structure

- **Prime** serves as the foundational layer of the Quai Network. It is designed to handle high transaction volumes, boasting a throughput of over 50,000 transactions per second (TPS). This layer is optimized for decentralized applications (dApps) and supports Ethereum Virtual Machine (EVM) compatibility, allowing developers to easily migrate existing Ethereum-based applications to the Quai Network. The architecture of Prime ensures that it can scale effectively while maintaining decentralization, which is crucial for the network's overall security and performance.
- **Paxos** operates as a secondary chain focused on facilitating stablecoin transactions and maintaining price stability within the network. It aims to provide predictable value through its native stablecoin, which is essential for users seeking to transact without the volatility commonly associated with cryptocurrencies. Paxos enhances the liquidity of the Quai Network by enabling seamless

conversions between various digital assets, thereby supporting both trading and everyday transactions.

- **Cyprus** acts as an auxiliary chain that enhances interoperability among different blockchains within the Quai ecosystem. It enables cross-chain communication and facilitates the transfer of assets across the various layers of the network. This capability is crucial for creating a unified experience for users who wish to interact with multiple chains without facing barriers or liquidity issues.

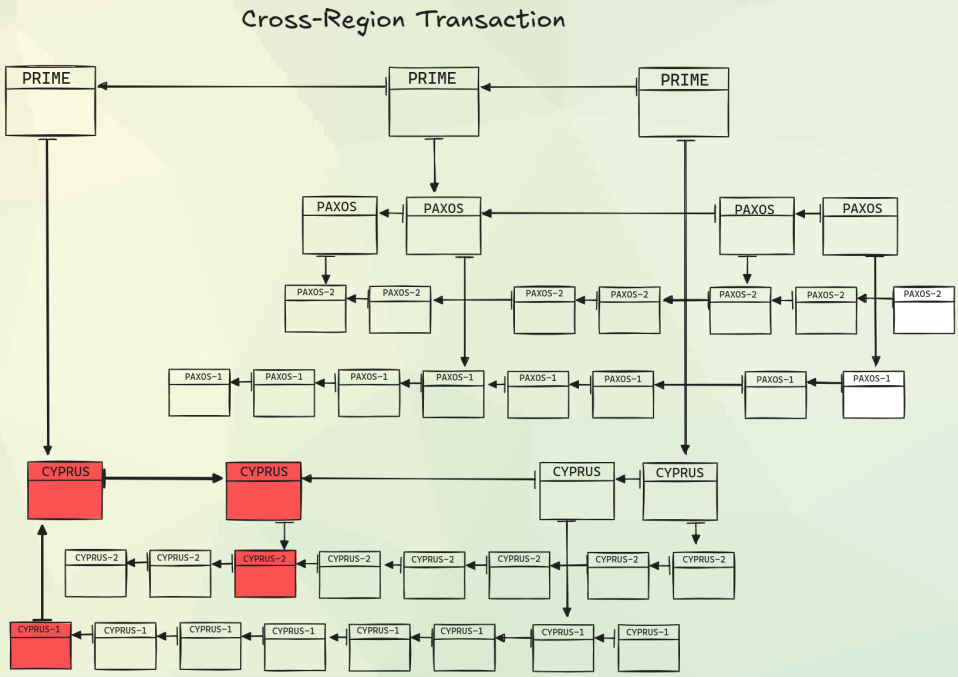

Figure 2.11: A detailed view of the Quai blockchain and its cross-region transaction.

Consensus Mechanism: Proof of Entropy Minima (PoEM)

The PoEM consensus mechanism is central to how Quai Network operates. It combines elements of traditional proof-of-work systems with innovative energy management strategies. PoEM incentivizes participants to contribute computational resources while ensuring that energy consumption is optimized and environmentally sustainable. This approach not only secures the network but also aligns with global efforts towards more sustainable blockchain practices.

In summary, Quai Network's multichain structure—comprising Prime, Paxos, and Cyprus—works in tandem with its PoEM consensus mechanism to create a scalable, efficient, and environmentally conscious blockchain ecosystem. This design allows for high transaction throughput, stable asset management, and seamless cross-chain interactions, positioning Quai Network as a forward-thinking player in the cryptocurrency landscape.

Topl

Topl is a specialized blockchain platform designed primarily to verify and track impact investments and sustainable practices, with a particular focus on ESG (Environmental, Social, and Governance) initiatives. The blockchain stands out for its unique approach to combining impact verification with blockchain technology. The core of Topl's architecture is built around its Proof of Learning consensus mechanism, which differs from traditional Proof of Work or Proof of Stake systems. This mechanism is designed to be energy-efficient while maintaining security and decentralization, aligning with the platform's sustainability goals. The platform implements a unique asset model that allows for the creation and tracking of both fungible and non-fungible assets, with special emphasis on "proof of impact" tokens. These tokens can represent various forms of positive impact, from carbon credits to fair trade certifications or sustainable sourcing verifications. Topl's smart contract system, known as Genus, is specifically designed to handle impact investing and ESG-focused transactions. It allows for the creation of complex investment arrangements while maintaining transparency and verifiability of impact claims.

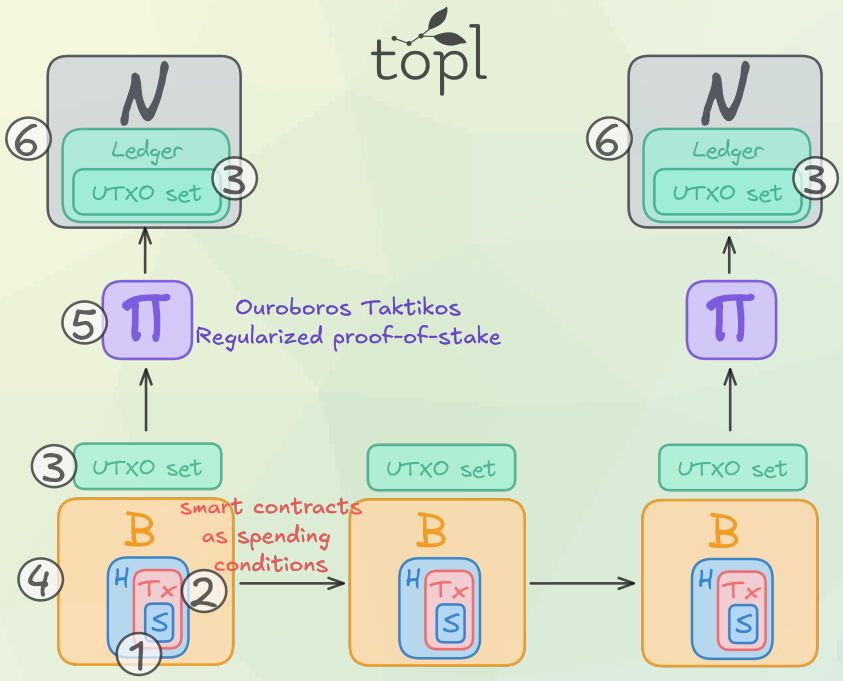

Figure 2.12: Simple Topl blockchain schematic.

Key architectural components include:

- The Bifrost Virtual Machine for smart contract execution
- Asset registry for tracking impact metrics
- Verification protocols for impact claims
- Cross-chain interoperability features
- Native support for impact-linked financial instruments

ALEPHIUM

Alephium represents a novel approach to blockchain architecture, combining the benefits of UTXO-based systems with advanced smart contract capabilities. At its core, it employs a unique stateful UTXO model that bridges the gap between Bitcoin's UTXO system and Ethereum's account-based model.

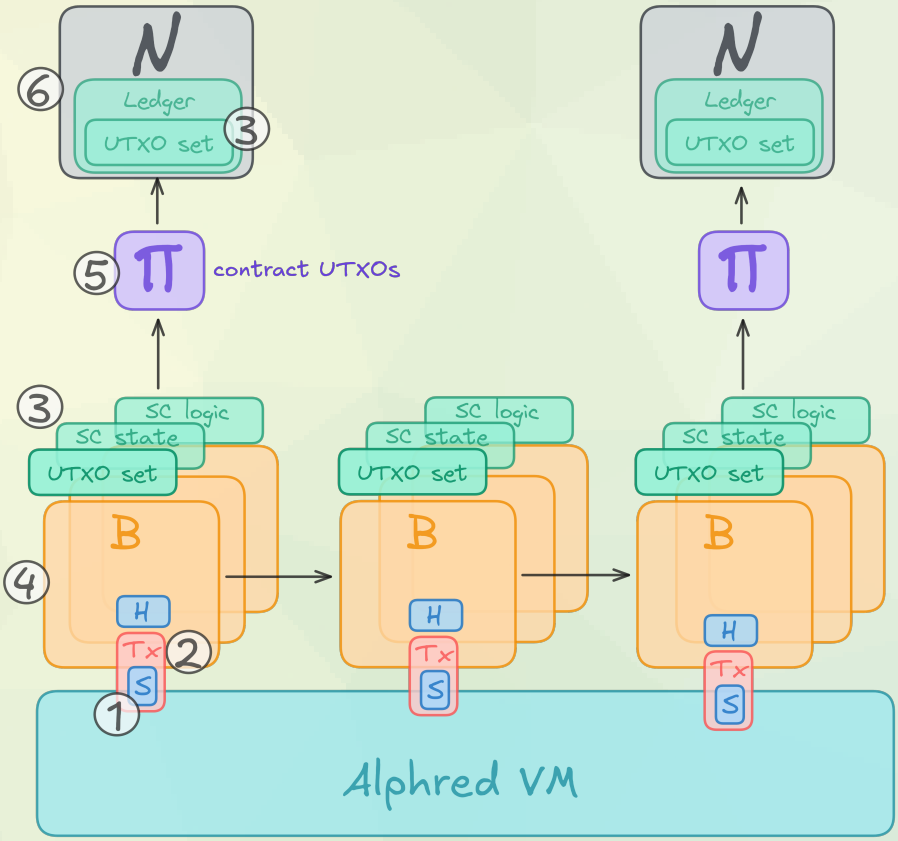

Figure 2.13: Simple Aelphium blockchain schematic.

Smart Contract Implementation with Alphred VM (AVM)

Alephium introduces its custom virtual machine, Alphred, specifically tailored for executing smart contracts. This VM addresses several challenges faced by existing dApp platforms: The AVM serves as the execution environment for smart contracts, managing contract state access and modifications while enforcing security and resource constraints. It reads contract states from the State Merkle Tree, processes state transitions based on contract logic, and creates new UTXOs that reflect state updates. This process ensures deterministic execution while maintaining the blockchain's security properties.

When processing transactions, the AVM executes contract functions based on transaction inputs, managing gas consumption and resource limits. It ensures atomic execution of contract operations and handles rollbacks in case of failures. This sophisticated interaction between the three Merkle trees and the AVM creates a robust foundation for smart contract execution.

- Enhanced Security: Alphred provides a robust environment for smart contract execution, reducing vulnerabilities common in other platforms.
- Trustless P2P Transactions: It supports trustless peer-to-peer smart contract transactions, facilitating decentralized finance (DeFi) applications without requiring intermediaries.
- Dedicated Programming Language (Ralph): Smart contracts on Alephium are written in Ralph, a programming language inspired by Rust. Ralph simplifies the development process for creating efficient and secure smart contracts, making it particularly suitable for DeFi applications

Hathor

Hathor is a layer-1 protocol with a unique combination of high scalability and high decentralization. It has a novel architecture, using both directed acyclic graph (DAG) and blockchain technologies intertwined in its ledger.

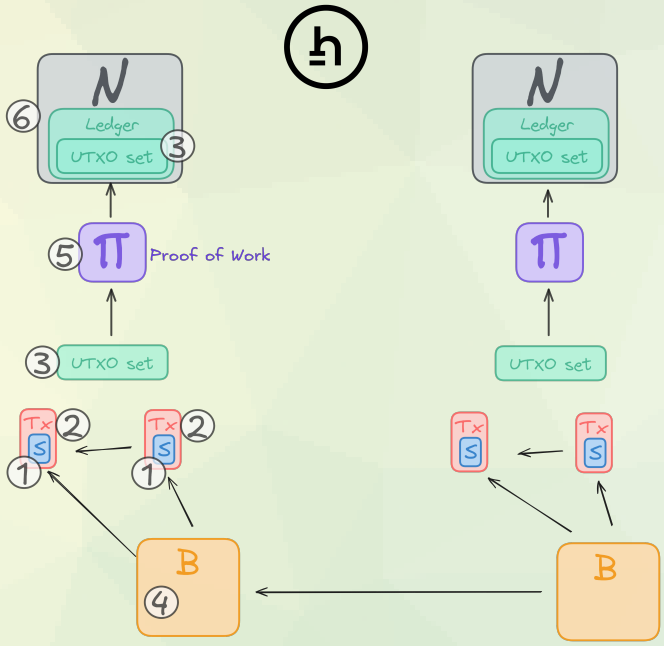

Figure 2.14: Simple Hathor blockchain schematic.

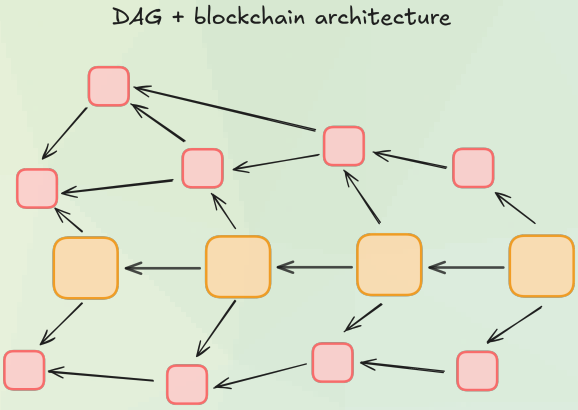

Figure 2.15: Hathor Directed Acyclic Graph (DAG) blockchain schematic. Note that here arrows represent a pointer to a previous object.

Unlike traditional blockchains, transactions are not contained inside the blocks. Each transaction confirms 2 previous ones, forming a DAG of transactions. As in the case of Bitcoin, miners find new blocks which form a chain of blocks. Additionally, blocks also confirm 2 previous transactions, propagating the proof-of-work to the DAG.

This architecture allows Hathor to scale without giving up decentralization. Moreover, Hathor Network simplifies blockchain integration by presenting an easy-to-use interface for builders and developers. One does not need to be an expert blockchain developer to create a product using the network.

Hathor Network uses sha256 proof-of-work for its consensus mechanism, the same as Bitcoin. In fact, the network supports merged mining with the Bitcoin network, which allows miners to find blocks in both networks without extra work. This has allowed Hathor's hashrate to grow rapidly, giving more security to the network.

Among the main features of Hathor Network are:

- Highly scalable with no central coordinator or any single point of failure.
- Feeless and quick transactions.
- Merged Mining with Bitcoin has given Hathor one of the highest hashrates among all Proof-of-Work networks.
- Easy token creation, in just a few clicks.
- Smart Contracts written in Python, called Nano Contracts.

Chapter 3: Supplementary Material

The Account Model

The core focus of the handbook has been the UTXO model and its variants. However, the computing paradigm can be other design.

- How do we determine "who owns what?" in each blockchain model
- How a single transaction is handled in each blockchain model
- How multiple transactions are handled concurrently
- States of confusion: Origin of non-determinism

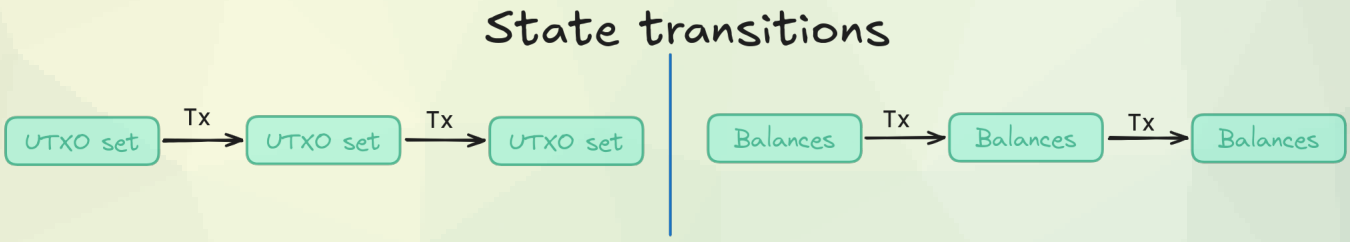

Figure 3.1: State transitions in both computing models, UTXO and Accounts.

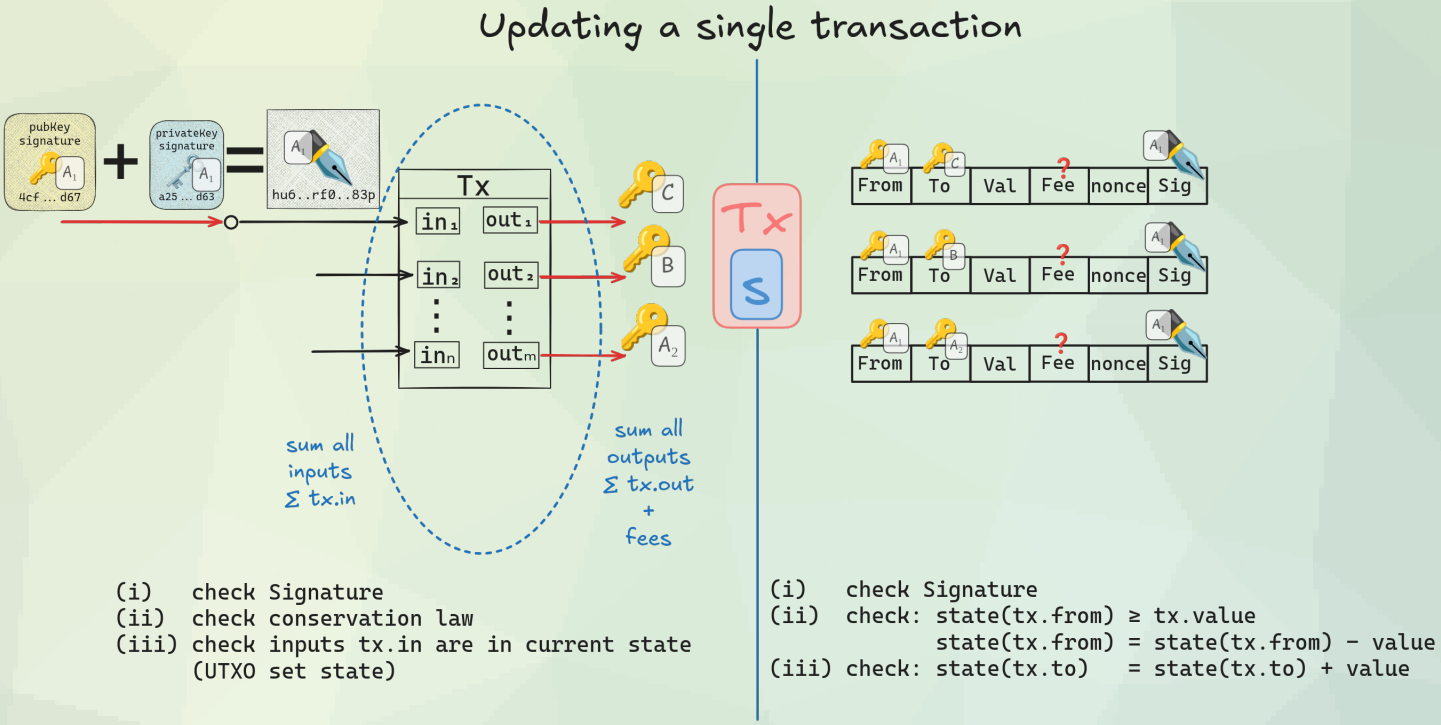

Figure 3.2: Comparative illustration of UTXO vs Account-based transaction models. The diagram contrasts UTXO's discrete state transitions (left) with Account model's balance updates (right), highlighting UTXO's advantages in deterministic fee calculation and transaction isolation. Keys, signatures, and validation rules are shown for both models, emphasizing their different approaches to state management.

The fundamental differences between UTXO and Account models in transaction processing is shown in Fig. 3.2. The left side shows the UTXO model where transactions operate on discrete unspent outputs. The process begins with public-private key verification, leading to transaction creation with

multiple inputs and outputs. The UTXO model ensures determinism through a conservation law where input sums must equal output sums plus fees.

The right side presents the simplified Account model, structured as standard database entries with "From", "To", "Value", "Fee" and signature fields. The key distinction lies in state management: UTXO transactions verify the existence of unspent outputs and consume them entirely, while Account models must check and update account balances through state modifications.

The UTXO model's superiority stems from its inherent parallelization capabilities and predictable fee computation. Since each UTXO represents a discrete state that can only be consumed once, transactions can be validated independently without worrying about account state races or complex interleaving of balance updates.

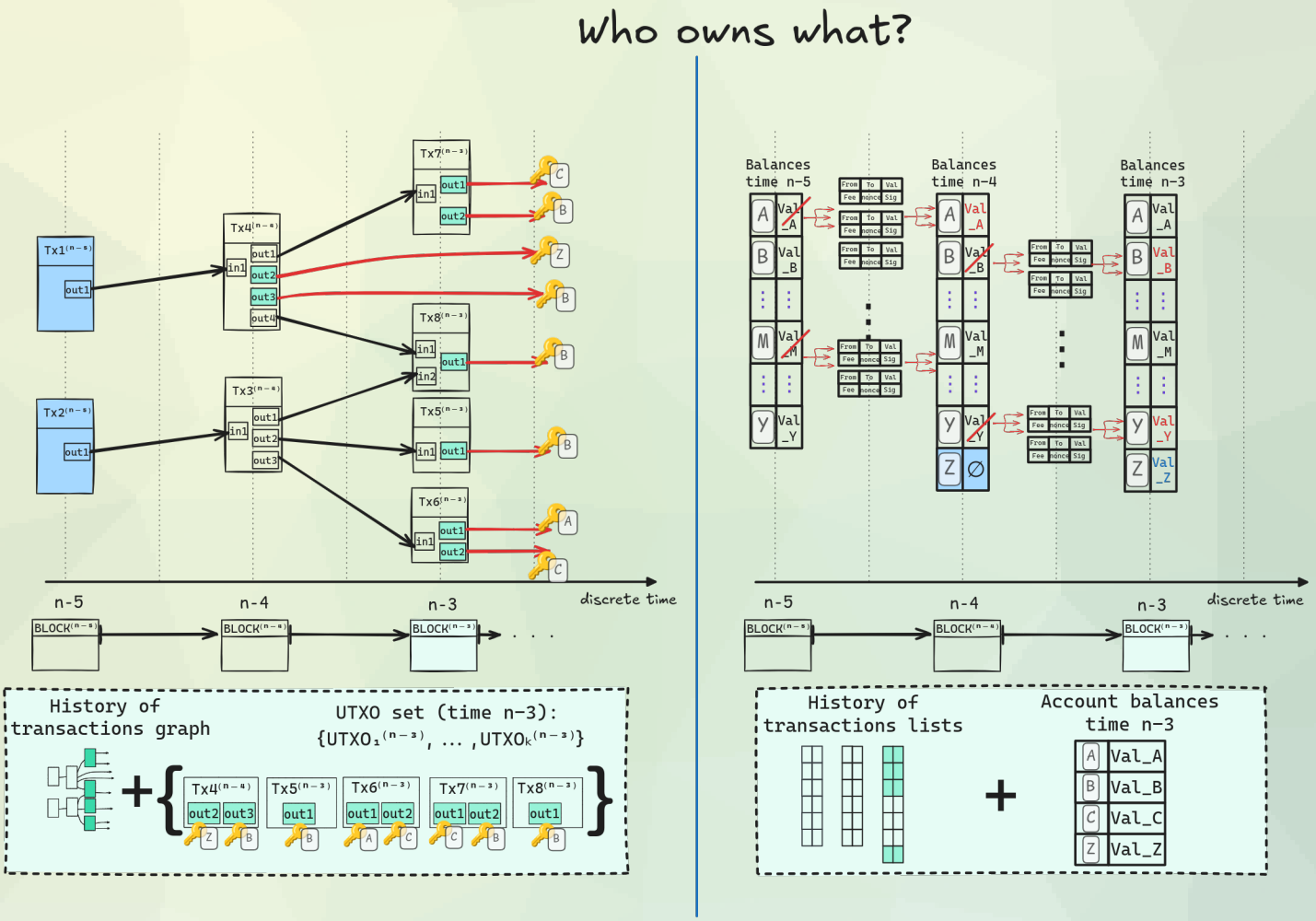

Figure 3.3: Time progression in both computing paradigms. A handful of transactions is shown for each model. Comparative visualization of global state management in UTXO vs Account models across time $(n-5)$ to $(n-3)$. The left side demonstrates UTXO's natural parallelization through a DAG structure, while the right shows the Account model's sequential state updates and potential concurrency issues. The diagram emphasizes how UTXO's stateless nature enables superior scalability through parallel transaction processing.

Fig. 3.3 portrays why UTXO models excel at handling concurrent transactions compared to Account models through a detailed examination of their state management approaches.

On the left side, we see the UTXO model's transaction graph evolving from time $n-5$ to $n-3$. Each transaction creates new outputs that become potential inputs for future transactions, forming a naturally expanding directed acyclic graph (DAG). When multiple transactions occur simultaneously, they simply add new branches to this graph. For instance, we can see how Tx_1 and Tx_2 at time $n-4$ each spawn multiple independent transactions at n-3, with no conflicts between them because each UTXO can only be spent once.

The Account model, shown on the right side, faces a fundamental challenge with concurrent operations. The diagram shows account balances for entities A through Z across the same time periods. When multiple transactions want to modify the same account balance simultaneously, they must be processed sequentially to maintain consistency. The red arrows between balance states illustrate this forced serialization - each balance update must wait for the previous one to complete, creating a bottleneck that worsens as network activity increases.

The bottom section of the diagram particularly illuminates this distinction. The UTXO model's history combines a transaction graph with a simple set of unspent outputs, allowing parallel validation since each UTXO's ownership is unambiguous. In contrast, the Account model requires both transaction lists and a global balance state, making parallel processing much more complex due to potential state conflicts.

Think of the UTXO model as building with LEGO blocks - you can add new blocks anywhere without disturbing existing structures. The Account model, however, is more like trying to modify a shared spreadsheet - multiple simultaneous edits to the same cell will cause conflicts that must be resolved sequentially.

This architectural difference becomes increasingly important as network activity grows. In the UTXO model, more transactions simply mean a wider DAG with more parallel branches, maintaining performance. For Account models, more transactions mean more potential conflicts in updating the global state, leading to longer processing queues and reduced throughput.

General UTXO Alliance Specs

UTXO Alliance Chain Specifications

Chain	Hashes	Supported Signatures	Smart Contracts	Block	Consensus Protocol	Network
Bitcoin	SHA256	ECDSA, Schnorr	Basic limited scripts	*Block size*: 4MB (post Segwit) *Average time between blocks:* $\sim$ 10min	Proof of Work	*Block propagation time:* $\sim 10 - 20$s
Cardano	BLAKE2B	ECDSA, Secp256k1, Schnorr	Expressive extended UTXO contracts. *Programming Languages:* Aiken (Rust alike), Plutus, Marlowe, OpShin, Plu-ts	*Block size*: 72-88kB. *Average time between blocks:* $\sim$ 20s	Ouroboros PoS	*Block propagation time:* $\sim$ 3s
Ergo	Autolykos	Secp256k1, Generalized Schnorr	Expressive expressive UTXO contracts. *Programming Languages:* Ergo Script (Scala based)	*Block size*: 8MB (adjustable) *Average time between blocks:* $\sim$ 2min	Autolykos PoW	*Block propagation time:* $\sim 2 - 4$s
Nervos Network	Eaglesong	Any user-defined cryptography	Expressive generalized UTXO contracts *Programming Languages:* CKB-VM allows general-purpose languages like Rust and C	*Block size*: 700kB *Average time between blocks:* $\sim$ 10s	NC-Max	*Block propagation time:* $\sim 2 - 5$s
Quai Network	SHA256	Secp256k1 (EVM), Schnorr (UTXO)	Expressive generalized UTXO contracts *Programming Languages:* Go, Solidity compatible	*Block size*: 2MB *Average time between blocks:* $\sim$ 60s	PoEM	*Block propagation time:* $\sim 1 - 3$s
Topl	SHA256	Ed25519	Expressive generalized UTXO contracts *Programming Languages:* Quivr	*Block size*: 72-88kB *Average time between blocks:* $\sim$ 20s	Ouroboros Taktikos Regularized PoS	*Block propagation time:* $\sim 1 - 2$s
Alephium	BLAKE3	ECDSA, Secp256k1	Expressive stateful UTXO contracts *Programming Languages:* Ralph (Rust based)	*Block size*: 2MB *Average time between blocks:* $\sim$ 16s	PoLW	*Block propagation time:* $\sim 1 - 3$s
Digibyte	BLAKE2B, Scrypt, Qubit, Odocrypt	ECDSA - Secp256k1	Limited scripts	*Block size*: 4MB *Average time between blocks:* $\sim$ 15s	Multi-Algo PoW	*Block propagation time:* $\sim$ 15s
Hathor	SHA256	ECDSA - Secp256k1	Expressive UTXO contracts *Programming Languages:* Python	*Block size*: 1kB *Average time between blocks:* $\sim$ 30s	PoW	*Block propagation time:* $\sim 1 - 3$s