Blockchain

A blockchain is a chain of "blocks", each containing specific information.

To begin, the **format and structure of a block** should be considered, and then **how they are linked (chained) together**.

# What is a Block

A block in a blockchain is typically stored as a **serialised data structure**.

*\*Generally speaking, the term "data structure" is used to describe complex data types that are used to organise and manage data - these include arrays, objects, linked lists, trees and hash tables, among others. While all data structures are also data types, primitive data types like integers and strings are not data structures. The term "data structure" is used to emphasise that the data type is used for structuring of data.*

A block contains several key components. The structure of a block will be something similar to below:

### **Header**

**Previous block hash** - this field contains the hash of the previous block in the chain, linking the current block to the previous one.

**Timestamp** - the exact time when the block was created.

**Nonce** - a number used in the proof-of-work algorithm to create a valid hash for the block.

**Merkle Root** - the root hash of a Merkle tree, which is a data structure used to efficiently and securely verify the transactions in the block.

### **Body (Payload)**

**Transactions** - this section includes all the transactions that are being recorded in the block. Each transaction typically includes details such as the sender and receiver addresses, the amount of cryptocurrency being transferred and any associated metadata or smart contract execution results.

### **Other Metadata**

**Block size** - the size of the block (which might be limited to ensure efficient propagation through the network).

**Transaction fees** - information about the fees associated with the transactions in the block.

**Additional metadata** - this can include various other data specific to the blockchain's implementation, such as version numbers, difficulty targets for mining, or other custom fields.

# Creation and Storage of a Block

### In-Memory Representation

While blocks are actively being created and processed, they typically exist as OOP objects or other structured data types (like dictionaries) in memory. This allows the software to manipulate and validate the data efficiently.

For example, in Bitcoin Core (written in C++), blocks are represented as instances of the `CBlock` class.

### Serialisation and Storage

Once blocks are finalised, they are serialised in to a binary format. This process involves converting the structured data into a compact sequence of bytes for efficient storage and transmission. In the Bitcoin blockchain, these serialised blocks are stored in `.dat` files (`blkNNNNN.dat`) on disk, which contain the raw binary data of multiple blocks. Other blockchains may use different storage mechanisms, such as databases.

### Handling Binary Data

Serialised binary data is typically handled as a stream when storing and retrieving blocks in a blockchain. This approach allows for efficient reading and writing of large data sets such as blockchain data, without loading the entire data set into memory at once.

# How Blocks are Linked (Chained) Together

### Previous Block Hash

* Each block in a blockchain contains a cryptographic hash of the previous block. A hash is a unique fixed-size string or number generated from input data, ensuring that even a small change in the input produces a drastically different hash.
* If any information in a previous block is altered, its hash will change, which in turn would change the hash stored in the subsequent block, thereby breaking the chain.
* The first block in a blockchain, known as the "genesis block", is unique because it doesn't reference a previous block. Instead, the previous block hash field is typically set to a default value (e.g., all zeros).

## Merkle Tree and Merkle Root

**Merkle Tree**

A Merkle Tree is a binary tree used to organise and verify large sets of data, such as transactions in a blockchain.

Each transaction in a block is hashed. These transaction hashes form the leaves (bottom nodes) of the Merkle tree.

Pairs of transaction hashes are then combined and hashed again to form the next level of nodes. This process is repeated, combining and hashing pairs of nodes, until a single hash remains at the top of the tree. The single hash is called the Merkle root.

**Merkle Root**

The Merkle root is the topmost hash in the Merkle tree and is a summary of all the transactions in the block.

The Merkle root is stored in the block header and ensures the integrity of all the transactions in the block.

To verify a specific transaction, you need only to check its hash against the Merkle root. If the transaction hash matches the Merkle root through the series of combined hashes, the transaction is valid. This verification process is efficient and does not require checking all transactions in the block.

### How Merkle Trees are Used in Bitcoin

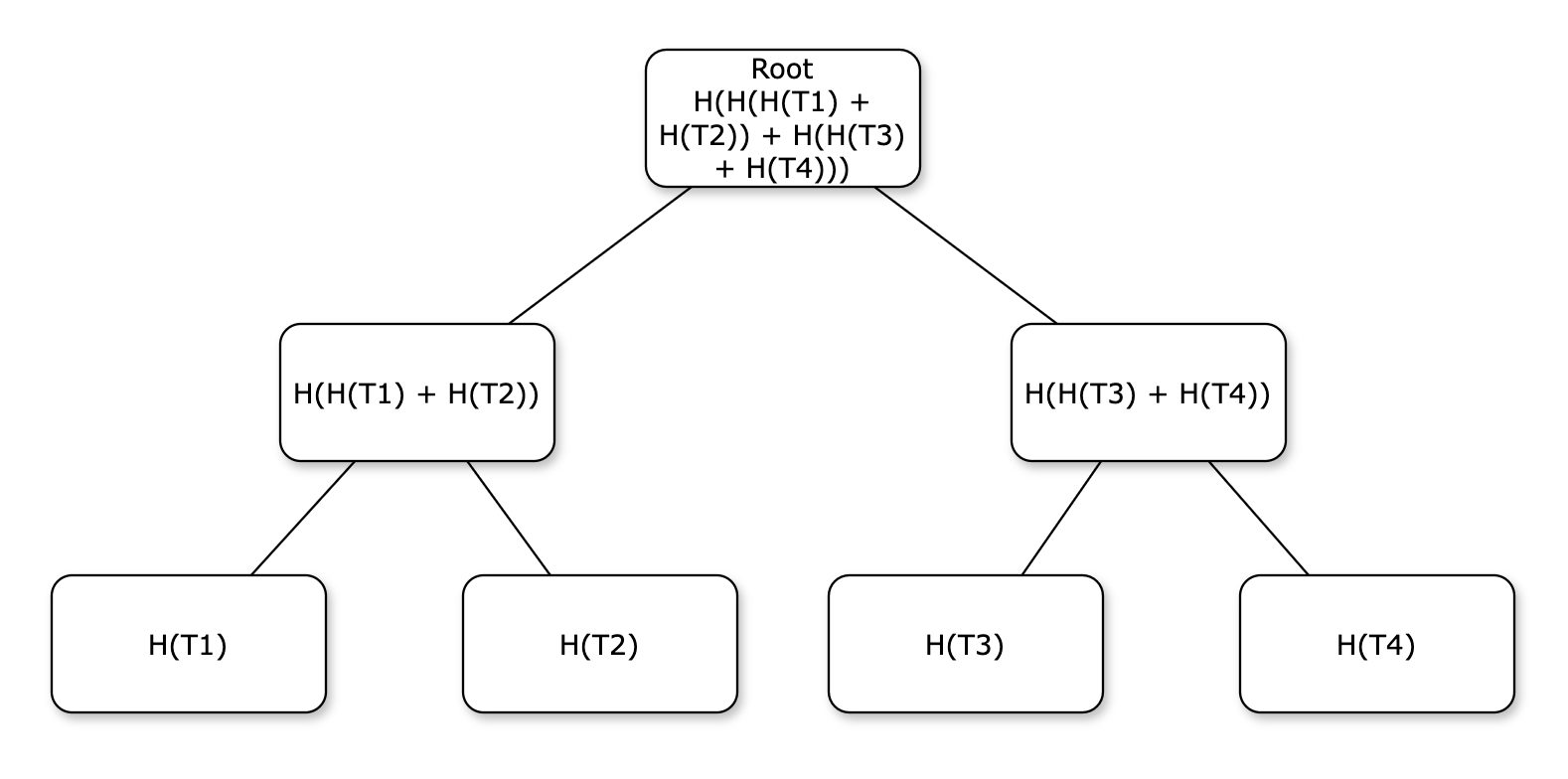
In the Bitcoin blockchain, each block contains a list of transactions. To ensure the integrity and efficiency of transaction verification, Bitcoin uses a Merkle tree.

1. Each transaction is hashed.
2. These hashes are then paired and hashed together to form the parent nodes. This process continues until the Merkle root is created.
3. The Merkle root of all transactions in a block is stored in the block header. This root acts as a fingerprint for the entire set of transactions in the block. Any change in a transaction will result in a different Merkle root, thereby ensuring data integrity.
4. Using the Merkle tree structure, Bitcoin can verify the inclusion of a transaction in a block with logarithmic complexity rather than linear complexity. This is achieved through **Merkle proofs**.

*\*A Merkle proof is a cryptographic proof that verifies the inclusion of a specific transaction within a block by providing a path of hashes from the transaction to the Merkle root of the block's Merkle tree. This allows for efficient and secure transaction verification without needing the full list of transactions.*

**The use of Merkle proofs can be demonstrated using the following example:**

Suppose a block has 4 transactions - T1, T2, T3 and T4.



To verify that T1 is in the block, you would need H(T2) and H(H(T2) + H(T4)).

1. Calculate H(T1) and combine it with (H(T2) to get H(H(T1) + H(T2)).
2. Combine H(H(T1) + H(T2)) with H(H(T3) + H(H(T4)) to match the Merkle root in the header.

## Full Nodes vs Simplified Payment Verification (SPV) Nodes

There are several different types of nodes in the decentralised Bitcoin network.

**Full Nodes**

* If you are running a full Bitcoin node, you have the entire blockchain, including all blocks and all transactions.
* Your node can construct the Merkle tree for any block from the transaction list it contains.

**Simplified Payment Verification (SPV) Nodes**

* SPV nodes, or lightweight clients, do not store the entire blockchain.
* Instead, they only store the block headers, which include the Merkle roots but not the full transactions.
* To verify a transaction, an SPV node requests a Merkle proof from a full node. The full node provides the necessary intermediate hashes to the SPV node.

If a simple payment verification (SPV) node needs to verify a transaction, T1, from a block containing a total of 4 transactions, it can do so by taking the following steps:

1. The SPV node requests a Merkle proof from a full node, specifying the transaction T1.
2. The full node calculates provides H(T1) and H(H(T3) + H(T4)).  
   Full nodes store all transactions, but do not store their hashes. Hashes are calculated dynamically when they are requested.
3. The SPV node hashes T1 to get H(T1).
4. It combines H(T1) with H(T2) to get H(H(T1) + H(T2)).
5. It combines H(H(T1) + H(T2)) with H(H(T3) + H(T4)) to get the Merkle root.
6. It compares this computed Merkle root with the Merkle root in the block header.
7. If the Merkle roots match, T1 is verified as part of the block.