Lesson 02 — Blockchain Fundamentals

Study Notes

Cryptoeconomics — BSc Level

Joerg Osterrieder

2025

Contents

1 Learning Objectives **3**

2 Block Structure **3**
- 2.1 Header Fields . . . 3
- 2.2 Block Body: Transactions . . . 4
- 2.3 The Nonce and Proof of Work . . . 4

3 How Blocks Form a Chain **5**
- 3.1 Hash Pointers . . . 5
- 3.2 Immutability via Cascading Hash Changes . . . 5
- 3.3 The Genesis Block . . . 5

4 Merkle Trees **5**
- 4.1 Structure . . . 6
- 4.2 SPV Proofs and Proof Size $O(\log n)$ . . . 6

5 Transaction Lifecycle **7**

6 Network Architecture **7**
- 6.1 Node Types . . . 8
- 6.2 Gossip Protocol and Propagation . . . 8

7 Forks **8**
- 7.1 Soft Forks . . . 8
- 7.2 Hard Forks . . . 9
- 7.3 Accidental Forks . . . 9

8 Scalability Challenge **9**
8.1 TPS Comparison 10
8.2 Blockchain Trilemma Trade-offs 10

9 Layer 2 Solutions **10**
9.1 State Channels 11
9.2 Plasma 11
9.3 Optimistic Rollups 11
9.4 ZK-Rollups 11

10 Practice Problems **12**

11 Key Takeaways **12**

12 Further Reading **13**

Learning Objectives

By the end of this lesson, students should be able to:

1. **Describe** the anatomy of a block, including all header fields and the role of each component.
2. **Explain** how cryptographic hash pointers link blocks into a chain and why this creates tamper-evidence.
3. **Construct** a Merkle tree from a set of transactions and describe how Merkle proofs enable efficient SPV verification.
4. **Trace** the full lifecycle of a transaction from creation through mempool to confirmed inclusion in a block.
5. **Distinguish** between soft forks and hard forks, and give historical examples of each.
6. **Explain** the blockchain scalability trilemma and describe how Layer-2 solutions attempt to address it.

Block Structure

Block (formal definition)

A **block** is the fundamental unit of a blockchain. Formally:

$$\text{Block}_n = \big(\underbrace{H(\text{Block}_{n-1})}_{\text{prev hash}},\ \text{Timestamp},\ \text{Nonce},\ \text{MerkleRoot},\ \text{Difficulty},\ \text{Transactions}\big)$$

The block is append-only: once accepted by the network, its content is fixed.

Header Fields

Every Bitcoin block header is exactly **80 bytes** and contains six fields:

Field	Size	Purpose
Version	4 B	Protocol version; signals miner support for soft-fork rule changes
Previous Hash	32 B	SHA-256d hash of the preceding block header; forms the chain link
Merkle Root	32 B	Root of the Merkle tree of all transactions in this block
Timestamp	4 B	Unix epoch time; must be within 2 hours of network median
Bits	4 B	Compact encoding of the proof-of-work difficulty target
Nonce	4 B	32-bit counter iterated by miners during proof-of-work

Block Body: Transactions

Following the header, the block body contains a list of transactions. The very first transaction in every block is the **coinbase transaction**: it has no inputs, only outputs, and creates the block reward (newly minted coins plus all transaction fees collected from the other transactions).

Regular transactions contain:

- **Inputs**: References to previous unspent transaction outputs (UTXOs) being consumed, together with unlocking scripts (signatures).
- **Outputs**: New ownership assignments; each output specifies an amount and a locking script (typically requiring the recipient's signature).

The Nonce and Proof of Work

The 32-bit nonce is the primary variable miners adjust when searching for a valid block hash. A valid block must satisfy:

$$H(\text{header}) < \text{target}$$

where the target is derived from the "Bits" field. Because SHA-256 behaves as a random oracle, miners must try many nonce values on average. When the 4-byte nonce space is exhausted, miners also vary the *extra nonce* in the coinbase transaction, which changes the Merkle Root and thus the entire header.

Common Pitfall

Misconception: "The nonce is a secret."

Correction: The nonce is public. Anyone can verify a valid block by hashing the header and

checking the result against the target. The difficulty is in *finding* a valid nonce, not in hiding it.

How Blocks Form a Chain

Hash Pointers

Each block header includes the SHA-256 double-hash ($H^2 = \text{SHA-256}(\text{SHA-256}(\cdot))$) of the previous block's header. This creates a **hash pointer**: a reference that both locates the previous block and authenticates its content.

Hash Pointer Property

If block k is altered, its hash changes. Block $k+1$ embeds the old hash of block k, so block $k+1$ is now inconsistent with block k. To restore consistency, an attacker must recompute block $k+1$'s proof-of-work. But block $k+2$ embeds the old hash of $k+1$, so the attacker must then redo block $k+2$, and so on through all subsequent blocks.

Immutability via Cascading Hash Changes

This cascade is the source of blockchain's *immutability*. Formally, let the current chain tip be block n. To rewrite block k where $k < n$, an adversary must:

1. Modify block k to include the fraudulent data.
2. Find a new nonce for block k satisfying the difficulty target.
3. Recompute valid nonces for blocks $k+1, k+2, \ldots, n$.
4. Outpace the honest network, which is continuously extending the canonical chain.

The probability that an attacker with fractional hash rate q (with honest network having rate $p = 1 - q$) succeeds after z confirmations decreases exponentially in z. For $q = 0.1$ and $z = 6$, the probability of successful rewrite is approximately 0.001. Six confirmations (≈ 60 minutes) is the conventional standard for high-value transactions.

The Genesis Block

The first block (block 0) has no predecessor. Its "Previous Hash" field is set to all zeros. Satoshi Nakamoto embedded in the coinbase transaction of the genesis block the message:

"The Times 03/Jan/2009 Chancellor on brink of second bailout for banks."

This serves as a timestamp proof and political commentary. The 50 BTC reward of the genesis block is permanently unspendable due to an early quirk in Bitcoin's code.

Merkle Trees

Structure

Merkle Tree

A **Merkle tree** is a binary hash tree. Leaves are the hashes of individual transactions. Each internal node is the hash of its two children. The single node at the top is the **Merkle Root**, stored in the block header. Formally, for four transactions T_1, T_2, T_3, T_4:

$$L_{12} = H\big(H(T_1) \,\|\, H(T_2)\big)$$
$$L_{34} = H\big(H(T_3) \,\|\, H(T_4)\big)$$
$$\text{Root} = H(L_{12} \,\|\, L_{34})$$

where $\|$ denotes concatenation.

Diagram (4 transactions):

```
                    Root = H(H12 || H34)

        H12 = H(H1 || H2)          H34 = H(H3 || H4)

   H1 = H(T1)   H2 = H(T2)   H3 = H(T3)   H4 = H(T4)
```

Any change to any transaction T_i changes $H(T_i)$, which propagates up to change the Merkle Root. Since the Root is in the block header, the entire block hash changes, making tampering immediately detectable.

SPV Proofs and Proof Size $O(\log n)$

Simplified Payment Verification (SPV) allows a light client (e.g., a mobile wallet) to verify that a transaction is included in a block without downloading the full blockchain. The light client downloads only block headers (≈ 80 bytes each, totalling ≈ 75 MB for all Bitcoin history as of 2024).

To verify transaction T_1 in a block with n transactions:

1. The full node provides the **Merkle proof**: the list of sibling hashes on the path from T_1's leaf to the root.
2. The light client recomputes hashes from T_1 upward, step by step, using the provided siblings.
3. If the computed root matches the root in the block header, the transaction is authentic.

For a balanced binary tree of n leaves, the path from a leaf to the root has length $\lceil \log_2 n \rceil$. Therefore:

Transactions	Proof size	Hashes needed
$n = 4$	2 hashes	$\log_2 4 = 2$
$n = 1{,}024$	10 hashes	$\log_2 1024 = 10$
$n = 1{,}000{,}000$	20 hashes	$\log_2(10^6) \approx 20$

This $O(\log n)$ efficiency is the key advantage of Merkle trees over a simple concatenated hash of all transactions, which would require $O(n)$ data to verify a single inclusion.

Common Pitfall

Odd number of transactions: If a block has an odd number of transactions, the last transaction hash is duplicated to form a pair. This is a known quirk of Bitcoin's Merkle implementation and does not affect security in practice but is worth knowing for implementation.

Transaction Lifecycle

A Bitcoin transaction passes through seven stages from creation to irreversibility:

1. **Creation**: The sender constructs the transaction, specifying input UTXOs to spend, output addresses, amounts, and transaction fee.
2. **Signing**: The sender signs the transaction with their private key using ECDSA. The signature proves ownership of the input UTXOs without revealing the private key.
3. **Broadcast**: The signed transaction is sent to one or more connected nodes. Nodes validate the transaction (checks: signature valid, inputs unspent, outputs $\leq$ inputs) before forwarding.
4. **Propagation**: Using a **gossip protocol**, each node forwards the transaction to all its peers (except the node it received it from). Propagation across the global network typically takes a few seconds.
5. **Mempool**: Valid unconfirmed transactions wait in each node's **memory pool** (mempool). Miners select transactions from the mempool to include in their candidate block, prioritising by fee-per-byte.
6. **Mining**: A miner includes the transaction in a candidate block and solves the proof-of-work puzzle. Upon finding a valid block, the miner broadcasts it to the network.
7. **Confirmation**: When the network accepts the block, the transaction has 1 confirmation. Each subsequent block built on top adds another confirmation, deepening the transaction's burial in the chain and making reversal exponentially harder.

Example: Fee Market Dynamics

During high-congestion periods (e.g., the 2017 bull market), Bitcoin transaction fees reached \$50–\$100 per transaction. Users who set low fees could wait days for confirmation. Bitcoin's **Replace-By-Fee (RBF)** feature allows a sender to broadcast a higher-fee version of an unconfirmed transaction to "bump" it to the front of the queue.

Network Architecture

Node Types

Bitcoin's P2P network consists of nodes with different roles:

- **Full nodes**: Store the complete blockchain history (as of early 2026, ≈ 720 GB). Validate every transaction and block against all consensus rules. Enforce the rules independently, without trusting any other node. The backbone of network security.
- **Light nodes (SPV clients)**: Store only block headers (≈ 75 MB). Use Merkle proofs for transaction verification. Trust that the chain with most proof-of-work is valid. Suitable for mobile wallets where storage is limited.
- **Mining nodes**: Full nodes that additionally construct candidate blocks and compute proof-of-work. Compete to add the next block and earn the block reward.
- **Relay networks**: Specialised infrastructure (e.g., FIBRE, Compact Block Relay) designed to propagate new blocks with minimal latency, reducing the orphan rate.

Gossip Protocol and Propagation

Bitcoin nodes communicate via an unstructured P2P gossip protocol. When a node receives a new transaction or block that it has not seen, it forwards it to all connected peers. Because each node independently decides to relay, a single piece of information can reach the entire network within seconds even if some nodes are offline, slow, or censoring.

This architecture has three key properties:

- **Resilience**: No single node failure disrupts the network.
- **Censorship resistance**: No single node can prevent a valid transaction from propagating.
- **No single point of failure**: Thousands of independently operated nodes maintain redundant copies.

Forks

Fork

A **fork** occurs when two or more valid blocks build on the same parent, creating a temporary divergence in the blockchain. Forks are also used to describe deliberate protocol rule changes.

Soft Forks

A **soft fork** tightens the consensus rules so that blocks valid under the new rules are a *strict subset* of blocks valid under the old rules. Old nodes accept new-rule blocks (they are valid under old rules too), but may build on invalid branches if they do not upgrade.

- **Backward compatible**: Old nodes continue to function, though they may not enforce the new stricter rules.

- **No chain split if adopted by a hashrate majority**: If more than 50% of miners upgrade, the new-rule chain will always be the longest.
- **Example — SegWit (2017)**: Segregated Witness moved signature data outside the traditional block size limit, increasing effective block capacity to ≈ 4 MB and fixing transaction malleability.
- **Example — Taproot (2021)**: Introduced Schnorr signatures and Merkelized Abstract Syntax Trees (MAST), improving privacy and smart contract efficiency.

Hard Forks

A **hard fork** relaxes the consensus rules, making previously invalid blocks valid. Old nodes reject blocks from upgraded nodes, so if the network does not reach consensus, a permanent chain split results.

- **Not backward compatible**: Old nodes see the new chain as invalid and refuse to follow it.
- **Creates two independent cryptocurrencies if contested**.
- **Example — Bitcoin Cash (BCH, August 2017)**: Increased block size limit from 1 MB to 8 MB (later 32 MB). Resulted in two separate chains and two separate assets.
- **Example — Ethereum / Ethereum Classic (ETC, July 2016)**: After the DAO hack, Ethereum Foundation executed a hard fork to reverse $60 million in stolen ETH. A minority rejected the reversal and continued the original chain as Ethereum Classic.

Accidental Forks

When two miners find valid blocks at nearly the same time, the network temporarily holds two competing chain tips (an **accidental fork** or **orphan block** situation). Nodes follow whichever tip they hear first until one chain extends further. The shorter branch is abandoned; its transactions return to the mempool. These short-lived forks (typically ≤ 1 block deep) are a normal part of Bitcoin's operation.

Example: SegWit vs. Bitcoin Cash: Two Philosophies

The 2017 block size debate illustrates divergent visions. "Big Blockers" (BCH camp) argued that increasing block size on-chain was the natural scaling path, prioritising low fees for every transaction. "Small Blockers" (BTC Core) argued for keeping blocks small to preserve node decentralisation, scaling via Layer-2 instead. Both approaches involve real trade-offs; neither is strictly correct.

Scalability Challenge

TPS Comparison

System	TPS (approx.)	Finality	Trade-off
Bitcoin	7	∼60 min	Maximum decentralisation, low TPS
Ethereum (PoS)	15–30	∼15 min	Smart contracts, moderate decentralisation
Solana	2,000+	∼1 s	High TPS, fewer validators
Visa	24,000	Milliseconds	Centralised, trusted intermediary
Lightning (BTC L2)	Millions potential	Milliseconds	Off-chain, requires channel liquidity

Blockchain Trilemma Trade-offs

Vitalik Buterin's **Blockchain Trilemma** states that a blockchain system can fully achieve at most two of three properties simultaneously:

Property	Nature of the Trade-off
Scalability	More throughput requires either fewer validators (worse decentralisation) or weaker validity checks (worse security).
Decentralisation	More validators means higher hardware requirements or smaller blocks to propagate quickly—both limit throughput.
Security	Strong Sybil resistance via PoW or large validator sets limits throughput and/or requires meaningful economic commitment.

The trilemma is a design heuristic, not a mathematical theorem. Active research aims to reduce (but not eliminate) these trade-offs.

Layer 2 Solutions

Layer-2 (L2) protocols execute transactions off the main chain (Layer 1) and post only compact proofs or final settlement data to L1. The key insight: they *inherit* L1's security while dramatically increasing throughput.

State Channels

State Channels (Lightning Network)

Two parties lock funds in a multi-signature on-chain contract. They then exchange signed off-chain messages updating their balance. Only the final state (or a dispute) is settled on-chain. No intermediate transactions touch the blockchain.

Properties: Instant, near-zero-fee transactions; both parties must be online to transact; liquidity is locked in the channel.

Plasma

Plasma creates a hierarchy of child chains anchored to the Ethereum mainnet. Transactions execute on a child chain; commitments (Merkle roots) are periodically submitted to the mainnet. Users can exit with funds by submitting a fraud proof if the operator misbehaves. Plasma is largely superseded by rollups due to its complex exit game.

Optimistic Rollups

Optimistic Rollups (Arbitrum, Optimism)

Batch hundreds of transactions off-chain, post compressed data and a new state root to L1. **Optimistic assumption**: all batches are valid unless challenged. A **challenge period** (typically 7 days) allows anyone to submit a fraud proof. If unchallenged, the state root is finalised.

Trade-off: Low cost, high throughput, but withdrawal delay of $\sim$1 week (mitigated by liquidity providers).

ZK-Rollups

ZK-Rollups (zkSync, StarkNet)

Batch transactions off-chain, then generate a **zero-knowledge validity proof** (e.g., zk-SNARK or zk-STARK) that is posted to L1. The proof mathematically guarantees correctness without a challenge period.

Trade-off: Immediate finality, stronger guarantees, but computationally expensive proof generation; harder to support arbitrary smart contracts (though improving rapidly).

Example: Layer 2 Inheritance of L1 Security

If a Lightning Network channel operator disappears, the user can broadcast their last signed channel state to Bitcoin's L1 and recover funds unilaterally. The security assumption reduces to: "can a user get a transaction confirmed on Bitcoin within a few blocks?" — which Bitcoin's base layer guarantees as long as fees are paid.

Practice Problems

1. **Block Header Fields.** List all six fields in a Bitcoin block header. For each field, state its size in bytes and explain why it is needed.

   Solution: Version (4B, protocol signal), Previous Hash (32B, chain link), Merkle Root (32B, transaction fingerprint), Timestamp (4B, temporal ordering), Bits (4B, difficulty target encoding), Nonce (4B, proof-of-work counter). The total is 80 bytes. Each field contributes to either the chain's integrity (hash link), its security (nonce/bits), or its timestamp ordering.

2. **Merkle Proof Efficiency.** A block contains 2048 transactions. (a) How many hashes does a Merkle proof of inclusion require? (b) How much data is this in bytes (SHA-256 hashes)?

   Solution: (a) $\lceil \log_2 2048 \rceil = 11$ hashes. (b) $11 \times 32 = 352$ bytes. By contrast, listing all 2048 transaction hashes directly would require $2048 \times 32 = 65{,}536$ bytes—a factor of 186× more data.

3. **Soft Fork vs. Hard Fork Classification.** For each change, classify as soft fork or hard fork and explain: (a) reducing the maximum block size from 1 MB to 500 KB; (b) increasing the maximum block size from 1 MB to 2 MB; (c) requiring all transactions to use a new signature format (with old-format transactions rejected by upgraded nodes).

   Solution: (a) **Soft fork** — new rule is stricter (old nodes' larger blocks are now rejected by updated nodes, but updated nodes produce blocks old nodes accept). (b) **Hard fork** — old nodes see 2 MB blocks as invalid. (c) **Hard fork** — old-format transactions become invalid; old nodes would accept them but upgraded nodes would not, creating a split.

4. **Trilemma Analysis.** You are designing a blockchain for a central bank that requires (i) processing 10,000 TPS and (ii) regulatory audit by 5 known institutions. Using the trilemma, which property must you sacrifice? Propose a concrete design approach.

   Solution: With only 5 known validators (permissioned), **decentralisation** is sacrificed. The design can achieve security (known, accountable validators using BFT consensus) and scalability (no PoW, fast BFT rounds). A practical approach: use a consortium blockchain (e.g., based on HyperLedger Fabric or Quorum) with PBFT or RAFT consensus among the 5 institutions, achieving thousands of TPS with near-instant finality.

Key Takeaways

- A block header is 80 bytes containing six fields; the **nonce** is the proof-of-work target, the **previous hash** creates the chain link, and the **Merkle root** summarises all transactions.
- **Hash pointers** make the chain tamper-evident: altering any past block requires re-doing all subsequent proof-of-work, which is economically infeasible for an attacker without majority hashrate.
- **Merkle trees** allow $O(\log n)$ transaction inclusion proofs, enabling SPV light clients on mobile

devices.

- A transaction moves from creation → signing → broadcast → mempool → mining → confirmation; the fee market determines confirmation speed.
- **Soft forks** are backward-compatible rule tightenings (SegWit, Taproot); **hard forks** are breaking changes that create a new chain if contested (BCH, ETC).
- The **blockchain trilemma** frames the trade-off between scalability, decentralisation, and security; Layer-2 solutions (state channels, optimistic rollups, ZK-rollups) address scalability while inheriting L1 security.

Further Reading

- Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*, Sections 4–7 (on proof-of-work, network, and Merkle trees). https://bitcoin.org/bitcoin.pdf
- Antonopoulos, A. M. (2017). *Mastering Bitcoin* (2nd ed.), Chapters 6–8 (transactions, blockchain, mining). Free online: https://github.com/bitcoinbook/bitcoinbook
- Buterin, V. (2014). "On Sharding Blockchains." https://ethereum.org/en/whitepaper/ (Appendix on scalability).
- Poon, J. & Dryja, T. (2016). *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. https://lightning.network/lightning-network-paper.pdf
- Narayanan, A. et al. (2016). *Bitcoin and Cryptocurrency Technologies*, Chapter 1 (on cryptographic primitives) and Chapter 2 (on how Bitcoin achieves decentralisation). Free PDF: https://bitcoinbook.cs.princeton.edu/
- Brownworth, A. *Blockchain Demo* (interactive visual). https://andersbrownworth.com/blockchain/