Lesson 2: Distributed Ledger Technology

Module A: Blockchain Foundations

BSc Blockchain & Cryptocurrency

University Course

2025

Learning Objectives

By the end of this lesson, you will be able to:

1. Define Distributed Ledger Technology (DLT) and distinguish it from blockchain
2. Explain the Byzantine Generals Problem and its relevance to consensus
3. Compare network topologies: centralized, decentralized, and distributed
4. Describe the structure and components of a blockchain block
5. Differentiate between node types: full nodes, light nodes, miners, validators
6. Contrast permissioned and permissionless blockchain architectures

Prerequisites: L01 - What is Blockchain?

What is Distributed Ledger Technology?

DLT Definition

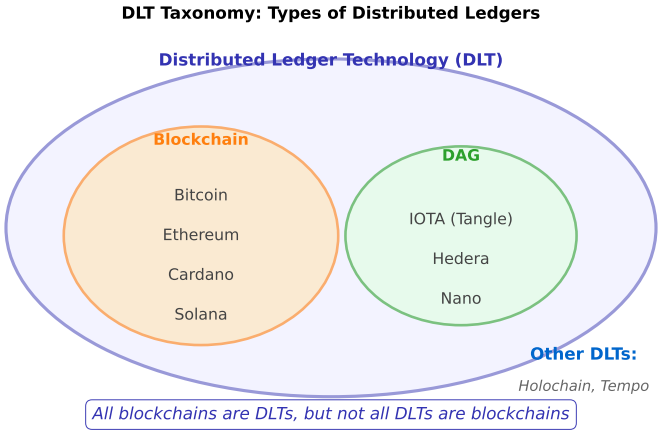
A **Distributed Ledger Technology (DLT)** is a database architecture where data is synchronized, replicated, and shared across multiple nodes in a network, without a central administrator.

Key Characteristics:

- **Distributed**: Data stored across many nodes (no single source of truth)
- **Synchronized**: All nodes maintain consistent state through consensus
- **Replicated**: Each node has a complete or partial copy of the ledger
- **Shared**: Multiple parties can read/write to the ledger
- **Consensus-driven**: Agreement mechanism validates changes

All blockchains are DLTs, but not all DLTs are blockchains

DLT Taxonomy: Types of Distributed Ledgers



Key Insight: Blockchain is a subset of DLT. Other approaches include DAGs (IOTA), Holochain, and Tempo (Radix).

Blockchain vs. Other DLTs

Blockchain

- Data organized in sequential blocks
- Blocks linked via cryptographic hashes
- Linear chain structure
- Append-only (immutable history)
- Examples: Bitcoin, Ethereum

Directed Acyclic Graph (DAG)

- Transactions reference previous transactions
- No blocks, no miners
- Parallel processing
- Examples: IOTA (Tangle), Hedera Hashgraph

Different DLT architectures optimize for different trade-offs

Holochain

- Agent-centric (not data-centric)
- Each agent maintains own chain
- Distributed Hash Table (DHT)
- No global consensus required

Tempo (Radix)

- Sharded state machine
- Logical clocks for ordering
- Scalable to millions of TPS
- Hybrid consensus

Ledger Types: Shared vs. Distributed

Shared Ledger (Traditional)

- Multiple parties access same database
- Central authority controls access
- Single version of truth
- Fast, but trust required

Example: Banking consortium database

Challenges:

- Single point of failure
- Administrator can alter records
- Reconciliation between systems

Distributed Ledger (DLT)

- Each party has own copy
- No central authority
- Consensus creates truth
- Slower, but trustless

Example: Public blockchain

Benefits:

- No single point of failure
- Transparent, auditable history
- No reconciliation needed

The Byzantine Generals Problem (1982)

Historical Context

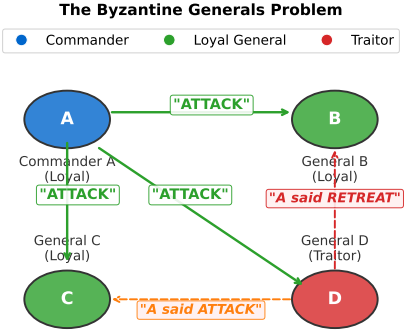
Leslie Lamport, Robert Shostak, and Marshall Pease formulated this thought experiment to describe the challenge of achieving consensus in distributed systems with potentially faulty or malicious actors.

The Scenario:

- Byzantine army surrounds an enemy city
- Army divided into divisions, each led by a general
- Generals must coordinate: **ATTACK** or **RETREAT**
- Communication only via messengers
- Some generals may be traitors (Byzantine faults)

Challenge: How can loyal generals reach consensus when traitors send conflicting messages?

Byzantine Generals: The Communication Challenge



Problem: How can loyal generals reach consensus when traitors send conflicting messages?

Problem: Traitor D sends conflicting messages to B and C, causing coordination failure.

Byzantine Generals: Example Scenario

Setup: 4 Generals (A, B, C, D), 1 is a traitor

Round 1 - Commander A sends orders:

- To General B: "ATTACK"
- To General C: "ATTACK"
- To General D (traitor): "ATTACK"

Round 2 - Traitor D sends conflicting messages:

- To General B: "A said RETREAT" (lie)
- To General C: "A said ATTACK" (truth)

Result Without Byzantine Fault Tolerance:

- General B thinks majority said RETREAT (withdraws alone)
- General C thinks majority said ATTACK (attacks alone)
- Army coordination fails, both divisions defeated

Byzantine Fault Tolerance (BFT)

Formal Result

Consensus is achievable if and only if at least $\frac{2}{3}$ of the actors are honest. Formally: $n \geq 3f + 1$ where n = total nodes, f = faulty nodes.

BFT Solution Requirements:

1. **Agreement**: All honest nodes decide on the same value
2. **Validity**: If all honest nodes propose value v, then v is decided
3. **Termination**: All honest nodes eventually decide

Blockchain Context:

- Bitcoin's Proof-of-Work solves Byzantine Generals Problem
- Nakamoto Consensus: Longest chain = majority decision
- Tolerates up to 49% malicious hash power (in practice, 51% attack possible)
- Practical BFT (PBFT) used in permissioned blockchains (Hyperledger)

Types of Faults in Distributed Systems

Crash Faults (Simpler)

- Node fails and stops responding
- Predictable behavior
- Detectable by timeout
- Example: Server power loss

Handling:

- Majority voting
- Heartbeat monitoring
- Leader election

Example: Apache Zookeeper, etcd (use Paxos/Raft)

Byzantine Faults (Complex)

- Node sends incorrect/conflicting information
- Malicious or software bugs
- Unpredictable behavior
- Example: Hacked node

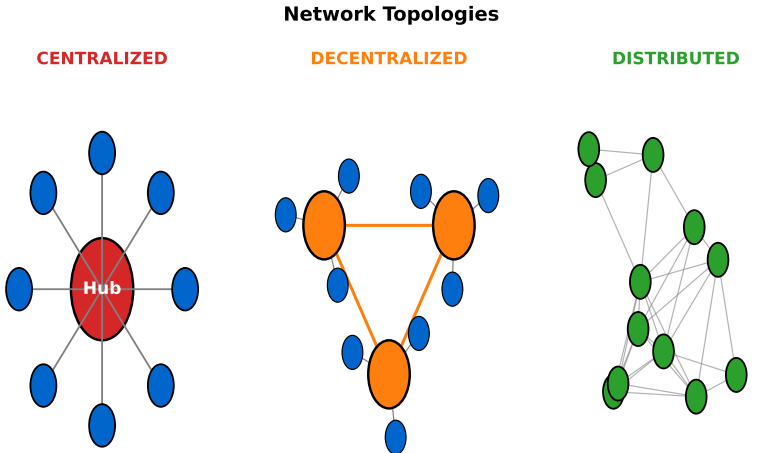
Handling:

- Cryptographic proofs
- Economic incentives
- Supermajority consensus ($> 2/3$)

Example: Bitcoin (PoW), Tendermint (BFT)

Key Insight: Public blockchains assume Byzantine environment; private blockchains may assume only crash faults

Network Topologies Comparison



Key Trade-off: Centralized is fast but fragile; Distributed is resilient but slower.

Centralized Networks

Architecture:

- Single central node (hub)
- All communication flows through hub
- Clients connect to central server
- Hub controls access and data

Advantages:

- Fast and efficient
- Easy to manage and update
- Low coordination overhead

Disadvantages:

- Single point of failure
- Censorship possible
- Privacy concerns (hub sees all data)

Examples: Traditional banking systems, Facebook, Gmail

Decentralized Networks

Architecture:

- Multiple hubs (federated model)
- Each hub serves subset of nodes
- Hubs communicate with each other
- No single central authority

Advantages:

- More resilient than centralized
- Regional autonomy
- Balanced between efficiency and resilience

Disadvantages:

- Hubs still represent points of failure
- Coordination between hubs needed
- Potential for fragmentation

Examples: Email (SMTP), Tor network, Mastodon (federated social media)

Distributed Networks

Architecture:

- Peer-to-peer (P2P) topology
- No hubs or central points
- Every node equal (or nearly equal)
- Direct node-to-node communication

Advantages:

- Maximum resilience (no single point of failure)
- Censorship resistant
- Highly fault-tolerant

Disadvantages:

- Slower coordination (consensus overhead)
- Complex to manage and upgrade
- Higher resource requirements per node

Examples: Bitcoin, Ethereum, BitTorrent, IPFS

Topology Comparison Table

Property	Centralized	Decentralized	Distributed
Control	Single entity	Multiple entities	All participants
Speed	Very fast	Fast	Slower
Resilience	Low (SPOF)	Medium	High
Censorship Resistance	None	Moderate	Strong
Coordination Overhead	Minimal	Moderate	High
Scalability	Easy (vertical)	Moderate	Hard (horizontal)
Trust Required	High	Medium	Low
Examples	Banks, Facebook	Email, Mastodon	Bitcoin, IPFS

Blockchain networks typically use fully distributed topology for maximum trustlessness

Anatomy of a Block: Overview

A blockchain block contains two main parts:

1. **Block Header** (80 bytes in Bitcoin)
   - Metadata about the block
   - Links to previous block
   - Proof-of-Work evidence
2. **Block Body** (variable size)
   - Transaction data
   - Can contain hundreds to thousands of transactions

Block Size Limits:

- Bitcoin: 1 MB (base block) to 4 MB (with SegWit)
- Ethereum: Dynamic gas limit ($\approx$ 30M gas, $\approx$ 15 MB)
- Bitcoin Cash: 32 MB

Block Header Components

Bitcoin Block Header Fields (80 bytes):

1. **Version** (4 bytes): Block version number (for protocol upgrades)
2. **Previous Block Hash** (32 bytes): Hash of the previous block header
3. **Merkle Root** (32 bytes): Hash of all transactions in this block
4. **Timestamp** (4 bytes): Unix epoch time (seconds since Jan 1, 1970)
5. **Difficulty Target** (4 bytes): Required difficulty for PoW
6. **Nonce** (4 bytes): Counter used for PoW mining

Key Insight:

$$\text{Block Hash} = \text{SHA256}(\text{SHA256}(\text{Header}))$$

The hash must be below the difficulty target for the block to be valid.

Merkle Trees: Efficient Transaction Verification

Merkle Tree Definition

A **Merkle tree** is a binary hash tree where each leaf node is a hash of a transaction, and each non-leaf node is a hash of its two children.

Construction Process:

1. Hash each transaction: $H(Tx_1), H(Tx_2), ..., H(Tx_n)$
2. Pair hashes and hash again: $H(H(Tx_1)||H(Tx_2))$
3. Repeat until single root hash (Merkle Root)

Benefits:

- **Compact Proof**: Verify transaction inclusion with $O(\log n)$ hashes
- **Efficient Storage**: Light clients store only 80-byte headers
- **Integrity**: Any transaction change alters Merkle Root

Example: Prove transaction in block with 1,000 tx requires only 10 hashes ($\log_2 1000 \approx 10$)

Merkle Proof Example

Scenario: Block contains 8 transactions, prove Tx_3 is included

Tree Structure:

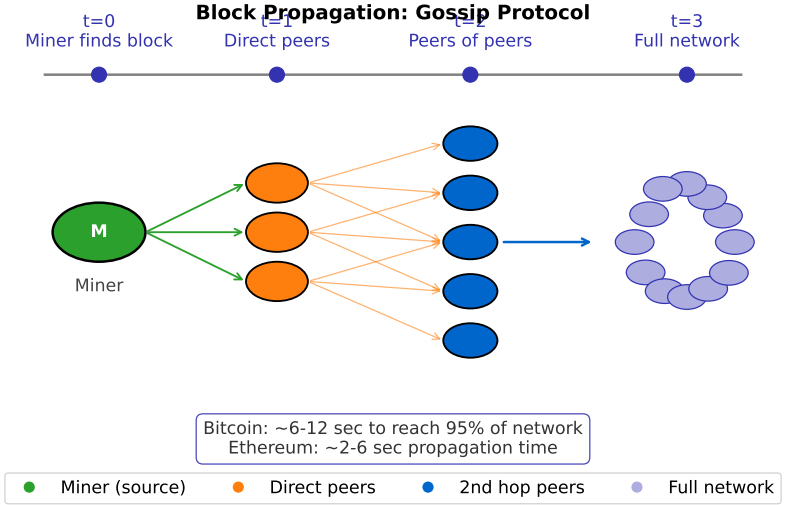
- Level 3 (Root): $R = H(AB||CD)$ where $AB = H(A||B)$, $CD = H(C||D)$
- Level 2: $A = H(Tx_1||Tx_2)$, $B = H(Tx_3||Tx_4)$, etc.
- Level 1: Tx_1, Tx_2, Tx_3, Tx_4, ..., Tx_8

Merkle Proof for Tx_3:

1. Verifier has: Block header with Merkle Root R
2. Prover provides: Tx_3, $H(Tx_4)$, $H(A)$, $H(CD)$
3. Verifier computes: $H(Tx_3)$, then $B = H(H(Tx_3)||H(Tx_4))$
4. Then: $AB = H(H(A)||B)$, finally $R' = H(AB||H(CD))$
5. If $R' = R$, then Tx_3 is proven to be in the block

Only 3 hashes needed instead of downloading all 8 transactions

Block Propagation via Gossip Protocol



Timeline: Miner broadcasts to peers, who propagate to their peers, until full network coverage.

Block Propagation Details

How New Blocks Spread Through Network:

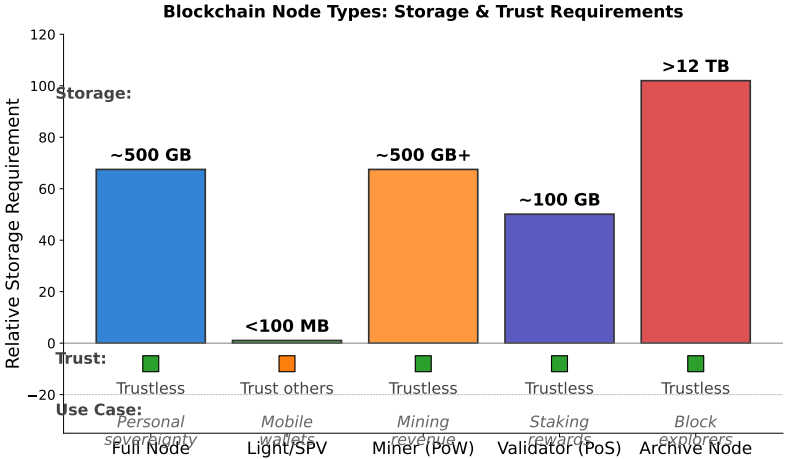
1. Miner finds valid block (solves PoW puzzle)
2. Miner broadcasts block to connected peers ($\approx$ 8-125 peers)
3. Peers validate block:
   - Check block hash meets difficulty target
   - Verify all transactions are valid
   - Ensure no double-spends
4. Valid block propagated to peers' peers (gossip protocol)
5. Process repeats until entire network has the block

Propagation Time:

- Bitcoin: $\approx$ 6-12 seconds to reach 95% of network
- Ethereum (pre-merge): $\approx$ 2-6 seconds
- Faster propagation reduces orphan block rate

Network latency creates temporary forks (resolved by longest chain rule)

Blockchain Node Types: Storage and Trust



Trade-off: Higher storage requirements enable trustless verification; lower requirements require trusting others.

Full Nodes

Full Node Definition

A **full node** downloads and validates the entire blockchain history, enforcing all consensus rules without relying on third parties.

Responsibilities:

- Store complete blockchain (Bitcoin: $\approx$ 500 GB, Ethereum: $\approx$ 900 GB)
- Validate all blocks and transactions independently
- Relay valid transactions and blocks to peers
- Serve blockchain data to light clients

Benefits:

- Maximum security (trust no one)
- Support network decentralization
- Can verify historical transactions

Requirements: $\approx$ 500 GB storage, $\approx$ 5 GB/day bandwidth, moderate CPU

Light Nodes (SPV Nodes)

Simplified Payment Verification (SPV)

Described in Bitcoin whitepaper: Light nodes download only block headers ($\approx$ 80 bytes each) instead of full blocks, relying on Merkle proofs to verify transactions.

Operation:

- Download all block headers ($\approx$ 50 MB for Bitcoin)
- Request Merkle proofs from full nodes for relevant transactions
- Verify transaction inclusion in block
- Assume longest chain is valid

Trade-offs:

+ Low storage ($<$ 100 MB vs. 500 GB)
+ Suitable for mobile devices
- Must trust full nodes for transaction data
- Vulnerable to bloom filter privacy leaks

Examples: Mobile wallets (Electrum, MetaMask mobile)

Miner Nodes (PoW)

Specialized Nodes in Proof-of-Work Blockchains

Functions:

1. Collect unconfirmed transactions from mempool
2. Construct candidate block (select transactions based on fees)
3. Attempt to find valid nonce (compute millions of hashes/second)
4. Broadcast valid block if found
5. Receive block reward + transaction fees

Requirements:

- Specialized hardware (ASICs for Bitcoin, GPUs for some altcoins)
- High electricity consumption (Bitcoin: $\approx$ 150 TWh/year globally)
- Cooling infrastructure
- High-bandwidth internet connection

Economics:

- Bitcoin block reward: 3.125 BTC (post-April 2024 halving)
- Break-even depends on: Hash rate, electricity cost, BTC price

Validator Nodes (PoS)

Specialized Nodes in Proof-of-Stake Blockchains

Functions:

1. Stake cryptocurrency as collateral (Ethereum: 32 ETH minimum)
2. Selected to propose/attest blocks based on stake
3. Earn rewards for honest participation
4. Risk slashing (stake confiscation) for malicious behavior

Ethereum Validator Requirements:

- 32 ETH staked (locked)
- Run validator client 24/7 ($>$ 99% uptime)
- Consumer-grade hardware (4 GB RAM, 100 GB SSD)
- Stable internet (10 Mbps)

Comparison to Mining:

+ 99.95% less energy consumption
+ Lower hardware requirements
- Capital locked (opportunity cost)
- Slashing risk if offline or malicious

Archive Nodes

Full Historical State Storage

- Store every state of the blockchain at every block height
- Enable queries like: "What was Alice's balance at block 1,000,000?"
- Required for block explorers (Etherscan, Blockchain.com)
- Needed for advanced analytics and forensics

Storage Requirements:

- Bitcoin archive node: $\approx$ 500 GB (same as full node, UTXO model)
- Ethereum archive node: $\approx$ 12 TB (account model stores all states)
- Grows continuously (Ethereum: $+1$ TB/year)

Use Cases:

- Block explorers
- Tax reporting services
- Academic research
- Forensic analysis

Most users do not need archive nodes; full nodes are sufficient

Node Type Comparison

Node Type	Storage	Validation	Trust Model	Use Case
Full Node	$\approx$ 500 GB	All blocks & txs	Trustless	Personal sovereignty
Light/SPV	$<$ 100 MB	Headers only	Trust full nodes	Mobile wallets
Miner (PoW)	$\approx$ 500 GB +	Full validation	Trustless	Mining revenue
Validator (PoS)	$\approx$ 100 GB	Full validation	Trustless	Staking rewards
Archive Node	$>$ 10 TB	Full + history	Trustless	Analytics, explorers

Decentralization Trade-off:

- More full nodes = more decentralization
- High storage requirements = fewer full nodes
- Blockchain scalability trilemma: Security, Decentralization, Scalability

Permissionless Blockchains (Public)

Definition

Permissionless (public) blockchains allow anyone to join, read, write, and participate in consensus without approval from a central authority.

Characteristics:

- Open participation (anyone can run a node)
- Pseudonymous identities (addresses, not real names)
- Transparent transaction history (all data public)
- Censorship resistant
- Token-based incentives (mining/staking rewards)

Examples: Bitcoin, Ethereum, Cardano, Solana

Best For:

- Global, trustless applications (DeFi, NFTs)
- Censorship-resistant systems
- Public goods (identity, voting)

Permissioned Blockchains (Private/Consortium)

Definition

Permissioned blockchains restrict participation to verified entities. Access to read, write, or validate is controlled by administrators.

Characteristics:

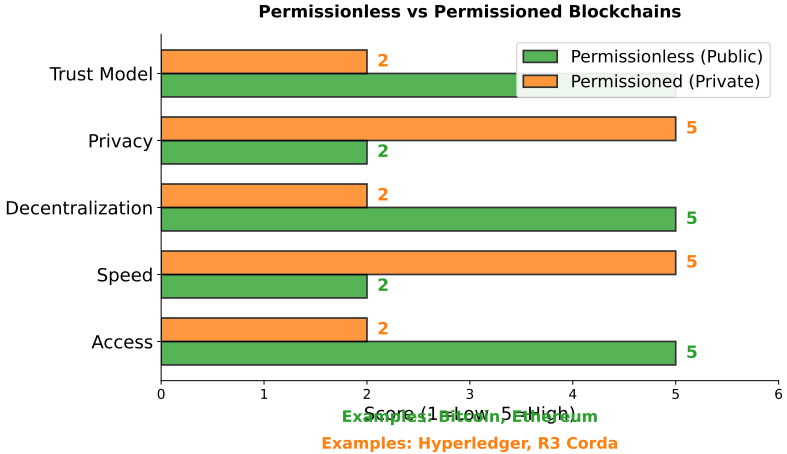
- Controlled participation (invitation-only)
- Known identities (KYC/AML compliance)
- Selective transparency (private channels possible)
- Efficient consensus (no mining, BFT-based)
- No native cryptocurrency (optional)

Examples: Hyperledger Fabric, R3 Corda, JP Morgan Quorum

Best For:

- Enterprise consortia (supply chain, trade finance)
- Regulatory compliance scenarios
- Inter-organizational workflows

Permissioned vs Permissionless: Key Dimensions



Trade-off: Permissionless maximizes decentralization/trust; Permissioned maximizes speed/privacy.

Comparison: Permissionless vs. Permissioned

Permissionless (Public)

Advantages:

- Maximum decentralization
- Censorship resistant
- Network effects (large user base)
- Innovation without permission

Disadvantages:

- Slower throughput (5-30 TPS)
- Higher costs (gas fees)
- Energy intensive (PoW)
- No privacy by default

Permissioned (Private)

Advantages:

- High throughput (1,000+ TPS)
- Low/no transaction fees
- Energy efficient (BFT)
- Configurable privacy

Disadvantages:

- Central point of control
- Limited network effects
- Potential for censorship
- Requires trust in admins

Hybrid models exist: Public with private sidechains (e.g., Polygon for Ethereum)

When to Choose Each Type

Criterion	Permissionless	Permissioned
Trust Model	No trust required	Parties partially trust each other
Participants	Unknown, global	Known, verified entities
Governance	Decentralized (hard forks)	Centralized/consortium voting
Performance	5-30 TPS	1,000-10,000+ TPS
Privacy	Pseudonymous, transparent	Configurable, private channels
Compliance	Challenging	Built-in (KYC/AML)
Cost	High (gas fees)	Low (no mining)
Use Cases	DeFi, NFTs, payments	Supply chain, trade finance, CBDCs

Key Takeaways

What You Should Remember:

1. **DLT** is broader than blockchain; includes DAGs, Holochain, etc.
2. **Byzantine Generals Problem**: Achieving consensus with potentially malicious actors requires $> 2/3$ honest nodes
3. **Network Topologies**: Centralized (fast, fragile), Decentralized (balanced), Distributed (resilient, slow)
4. **Block Structure**: Header (80 bytes) + Body (transactions), linked via cryptographic hashes
5. **Merkle Trees**: Enable efficient transaction verification ($O(\log n)$ proof size)
6. **Node Types**: Full (trustless), Light (trust others), Miner/Validator (consensus), Archive (historical queries)
7. **Permissioned vs. Permissionless**: Public for trustless apps, private for enterprise consortia

Discussion Questions

Consider and discuss:

1. **Decentralization Trade-offs**: At what point does storage cost compromise decentralization?
   - If only enterprises can afford archive nodes, is the network still decentralized?
2. **Byzantine Assumptions**: Are public blockchains too conservative?
   - Could permissioned chains with known actors be more efficient?
3. **Privacy vs. Transparency**: How can we enable audits without sacrificing user privacy?
   - Explore: zk-SNARKs, confidential transactions

References & Resources

Foundational Papers

- Lamport et al. (1982): *Byzantine Generals Problem*
- Nakamoto (2008): *Bitcoin Whitepaper* (Section 8: SPV)
- Castro & Liskov (1999): *Practical Byzantine Fault Tolerance*

Technical Resources

- Bitcoin Developer Guide
- Ethereum Yellow Paper
- Hyperledger Fabric Docs

Interactive Tools

- Blockchain Demo (andersbrownworth.com)
- Merkle Tree Visualizer
- Node map: bitnodes.io

Books

- Antonopoulos (2023): *Mastering Bitcoin* (Chapter 6: Transactions)
- Narayanan et al. (2016): *Bitcoin and Cryptocurrency Technologies* (Chapter 2)

Next Lesson Preview

L03: Cryptographic Hash Functions

We will explore:

- Properties of cryptographic hash functions (deterministic, fixed-length, avalanche effect)
- SHA-256 algorithm and its use in Bitcoin
- Collision resistance and birthday paradox
- Hash functions as digital fingerprints
- Merkle trees in-depth (construction and verification)
- Applications beyond blockchain

Preparation: Review binary representation of data and basic probability theory

Thank you

Questions?

See you in Lesson 3: Cryptographic Hash Functions