Blockchain Fundamentals

Cryptocurrency and Blockchain Technology

Lesson 1 of 4

Prof. Dr. Joerg Osterrieder

January 25, 2026

Today's Agenda

Part 1: What is Blockchain?

- History and origins
- Definition and key properties
- Real-world applications

Part 2: Block Structure

- Anatomy of a block
- Block header and body
- Merkle trees
- How blocks link together

Part 3: Consensus Mechanisms

- The double-spending problem
- Proof of Work (PoW)
- Proof of Stake (PoS)

Part 4: Python Implementation

- Building a simple blockchain
- Mining demonstration
- Chain validation

Duration: 45 minutes — This lesson provides the foundational concepts for understanding blockchain technology.

Section 1: What is Blockchain?

The Birth of Blockchain

October 31, 2008
Satoshi Nakamoto publishes the Bitcoin whitepaper:

"Bitcoin: A Peer-to-Peer Electronic Cash System"

Key Innovation:

- First practical solution to the double-spending problem
- No trusted third party required
- Completely decentralized

Timeline

- **2008:** Whitepaper published
- **Jan 3, 2009:** Genesis block mined
- **2009:** First Bitcoin transaction
- **2010:** First real-world purchase (10,000 BTC for pizza)
- **2015:** Ethereum launches
- **2020s:** Institutional adoption

Mystery: Satoshi's identity remains unknown

The 2008 financial crisis provided context for a trustless, decentralized monetary system.

What is a Blockchain?

Definition
A blockchain is a **distributed, immutable ledger** that records transactions across a network of computers.

Breaking it down:

- **Distributed:** Copies exist on many computers (nodes)
- **Immutable:** Once written, data cannot be changed
- **Ledger:** A record of transactions
- **Linked blocks:** Data organized in sequential blocks

Simple analogy:
Think of it as a shared Google Doc that:

- Everyone can read
- No one can edit past entries
- New entries require group approval

Traditional Database vs Blockchain

Database	Blockchain
Centralized	Distributed
Admin controls	Network controls
Can be edited	Append-only
Trust required	Trustless
Fast writes	Slower writes

A blockchain replaces trust in institutions with trust in mathematics and code.

Key Properties of Blockchain

Decentralization

No single point of control:

- No central authority
- Peer-to-peer network
- Democratic governance
- Resistant to shutdown

"The network is the authority"

Transparency

Open and verifiable:

- All transactions public
- Anyone can audit
- Real-time verification
- Pseudonymous (not anonymous)

"Trust through verification"

Immutability

Permanent record:

- Data cannot be altered
- Cryptographic linking
- Tamper-evident
- Historical integrity

"Written in stone"

Together, these properties enable: Trustless transactions between strangers

These three properties are interdependent – removing one weakens the others.

Real-World Applications

Finance & Payments

- Cross-border remittances
- Micropayments
- Decentralized finance (DeFi)
- Central bank digital currencies (CBDCs)

Supply Chain

- Product traceability
- Authenticity verification
- Quality assurance
- Logistics optimization

Identity & Voting

- Self-sovereign identity
- Secure voting systems
- Credential verification
- Privacy-preserving authentication

Other Applications

- Healthcare records
- Intellectual property
- Real estate titles
- Gaming and NFTs

Common thread: Applications where *trust*, *transparency*, and *immutability* are critical

Not everything needs a blockchain – it's most valuable where trust is expensive or impossible.

Why Blockchain Matters for Finance

Current Financial System Pain Points

- × Cross-border payments take 2-5 days
- × High transaction fees (3-7%)
- × Settlement requires intermediaries
- × Limited access for unbanked
- × Lack of transparency
- × Counterparty risk

Blockchain Solutions

- ✓ Near-instant settlement
- ✓ Minimal transaction costs
- ✓ Direct peer-to-peer transfers
- ✓ Financial inclusion
- ✓ Full auditability
- ✓ Trustless execution

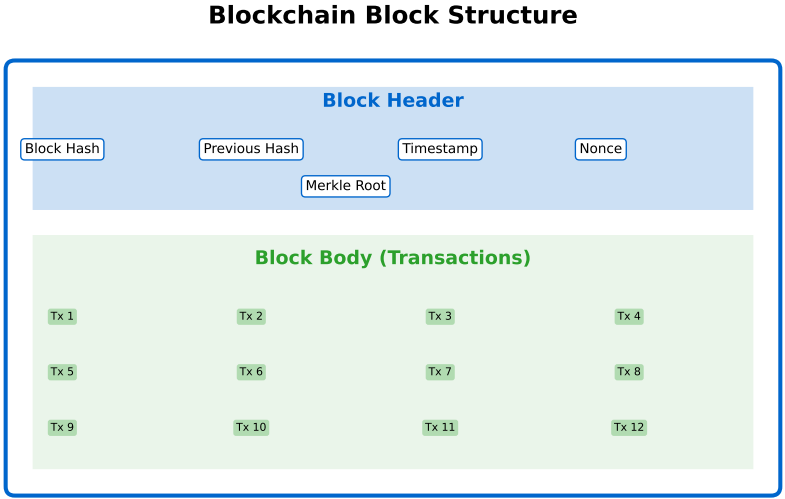
Key Financial Innovations:

- **Smart Contracts:** Self-executing agreements with terms written in code
- **Tokenization:** Digital representation of real-world assets
- **DeFi:** Decentralized lending, borrowing, and trading

The financial industry stands to gain the most from blockchain's efficiency and transparency.

Section 2: Block Structure

What is a Block?



Block Header Components

Essential Header Fields

Field	Purpose
Block Hash	Unique identifier for this block
Previous Hash	Links to parent block
Timestamp	When block was created
Nonce	Mining solution number
Merkle Root	Hash of all transactions

Additional fields (Bitcoin):

- Version number
- Difficulty target
- Block height (position in chain)

How the Hash is Calculated

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

The hash depends on:

1. Previous block's hash
2. Merkle root of transactions
3. Timestamp
4. Nonce value

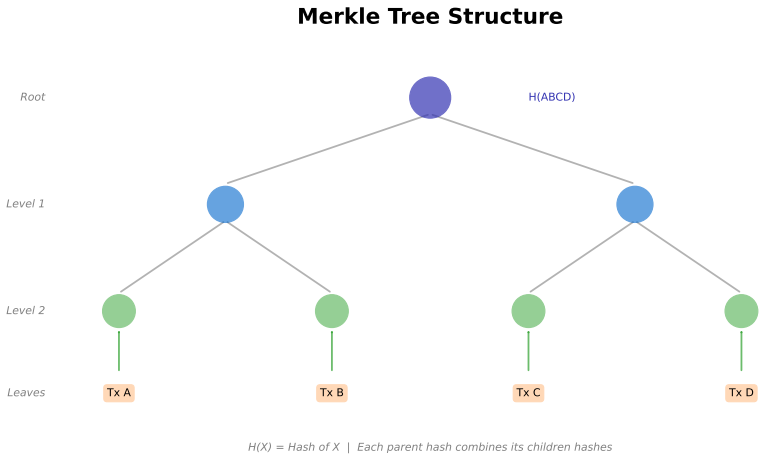
Key insight:
Changing *any* field changes the hash completely (avalanche effect)

Example hash:

```
000000000000000000a4b...
```

The block hash is not stored in the block itself – it's computed from the header data.

Merkle Tree: Efficient Transaction Verification



How it works:

Benefits:

Block Body: Transactions

What's in a Transaction?

Bitcoin Transaction:

- **Inputs:** References to previous outputs
- **Outputs:** New ownership assignments
- **Amounts:** Value being transferred
- **Signatures:** Proof of authorization

Example:

$$\text{Alice} \xrightarrow{10\ \text{BTC}} \text{Bob}$$

Transaction includes:

- Reference to Alice's coins
- Bob's receiving address
- Alice's digital signature

Block Capacity

Blockchain	Block Size/Limit
Bitcoin	∼1 MB
Ethereum	Gas limit
Solana	∼10 MB

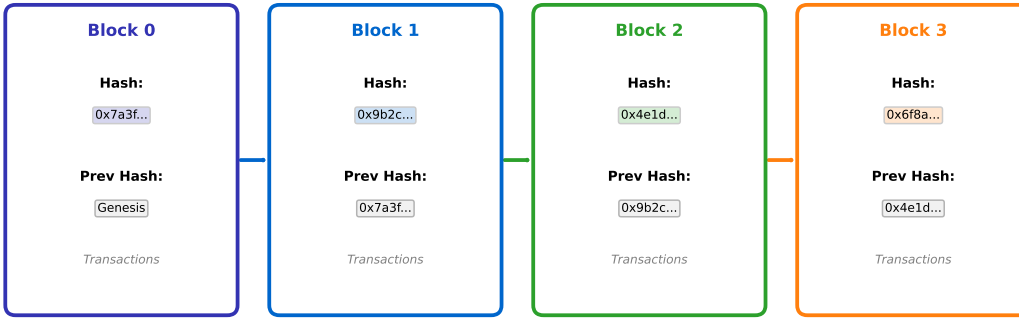
Typical block contains:

- 1,500–2,500 transactions (BTC)
- Created every ∼10 minutes (BTC)
- Validated by network consensus

Transaction fees:
Users pay miners/validators to include their transactions

The block body is where the actual value transfer data lives – the header just summarizes it.

How Blocks Link Together



Each block contains the hash of the previous block, forming an immutable chain

Python Implementation: Block Class

```python
import hashlib
import time

class Block:
    def __init__(self, index, transactions, previous_hash):
        self.index = index
        self.timestamp = time.time()
        self.transactions = transactions
        self.previous_hash = previous_hash
        self.nonce = 0
        self.hash = self.calculate_hash()

    def calculate_hash(self):
        block_string = f"{self.index}{self.timestamp}{self.transactions}{self.previous_hash}{self.nonce}"
        return hashlib.sha256(block_string.encode()).hexdigest()

    def mine_block(self, difficulty):
        target = "0" * difficulty
        while self.hash[:difficulty] != target:
            self.nonce += 1
            self.hash = self.calculate_hash()
        print(f"Block mined: {self.hash}")
```

This simple implementation captures the core concepts: hashing, linking, and mining.

Why Tampering Breaks the Chain

Scenario: Attacker modifies Block 2

1. Attacker changes transaction in Block 2
2. Block 2's hash changes (avalanche effect)
3. Block 3's "previous hash" no longer matches
4. Block 3 becomes invalid
5. All subsequent blocks become invalid

To successfully tamper:

- Must recalculate Block 2's hash
- Must recalculate Block 3's hash
- Must recalculate ALL subsequent blocks
- Must outpace the honest network

Why this is practically impossible:

- **Computational cost:** Each block requires significant work to mine
- **Network consensus:** Honest nodes reject invalid chains
- **Time factor:** Network continues adding blocks
- **51% attack:** Would need majority hash power

Security guarantee:
Deeper the block, harder to change

Bitcoin's security relies on the economic infeasibility of outcomputing the entire network.

Section 3: Consensus Mechanisms

The Double-Spending Problem

The Problem

Digital data can be copied perfectly. How do we prevent someone from spending the same digital money twice?

Scenario:

1. Alice has 10 BTC
2. Alice sends 10 BTC to Bob
3. Alice sends *same* 10 BTC to Charlie
4. Both transactions appear valid!

Traditional solution:
Central authority (bank) tracks all balances

But this requires trust...

Blockchain Solution

Consensus mechanism determines:

- Which transactions are valid
- The order of transactions
- Which version of history is "true"

Key insight:
The network agrees on a **single, ordered history**

Result:

- First transaction is accepted
- Second transaction is rejected
- No central authority needed

Solving double-spending without trusted parties was the key innovation of Bitcoin.

Proof of Work (PoW)

The Concept

Miners compete to solve a computational puzzle:

Find a nonce such that:

$$\text{Hash}(\text{block} + \text{nonce}) < \text{target}$$

The Process:

1. Collect pending transactions
2. Try random nonce values
3. Check if hash meets difficulty
4. First to succeed wins the reward
5. Others verify and accept

Bitcoin's difficulty:
Hash must start with $\sim$19 zeros

Why It Works

Key properties:

- **Hard to find:** Requires many attempts
- **Easy to verify:** One hash check
- **Adjustable:** Difficulty adapts
- **Fair lottery:** Proportional to hashrate

Economics:

- Winner receives block reward
- Currently 3.125 BTC (post-2024 halving)
- Plus transaction fees

Cost: Massive energy consumption

PoW makes attacking the network economically irrational – honest mining is more profitable.

Proof of Stake (PoS)

The Concept

Validators are chosen based on their "stake" (locked cryptocurrency):

The Process:

1. Validators lock up tokens as collateral
2. Algorithm selects validator (weighted random)
3. Selected validator proposes block
4. Other validators attest/verify
5. Valid blocks earn rewards

Selection factors:

- Size of stake
- Randomization
- Validator reputation

Security Mechanism

Slashing: Bad behavior $\rightarrow$ lose stake

Punishable actions:

- Double-signing (voting twice)
- Proposing invalid blocks
- Being offline (mild penalty)

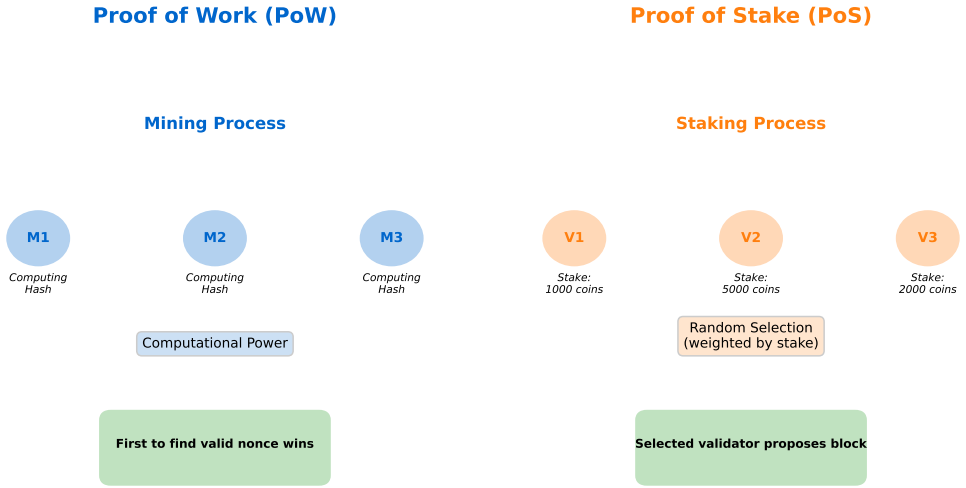
Advantages:

- 99.9%+ less energy
- No special hardware needed
- Faster finality possible
- Economic penalties ¿ physical

Ethereum: Requires 32 ETH stake

PoS achieves security through economic incentives rather than computational work.

Consensus Mechanisms Comparison



Why Consensus Matters

Consensus Ensures:

- ✓ **Agreement:** All nodes share same state
- ✓ **Ordering:** Transactions have definitive sequence
- ✓ **Validity:** Only legitimate transactions accepted
- ✓ **Finality:** Confirmed transactions stay confirmed
- ✓ **Liveness:** Network continues making progress

Without consensus:

- Different nodes have different "truths"
- Double-spending becomes possible
- Trust in the system collapses

The Byzantine Generals Problem

How can distributed parties agree when some may be malicious?

Requirements:

- Majority must be honest
- Communication may be unreliable
- Some generals (nodes) may lie

Blockchain's solution:
Economic incentives align behavior:

- Honest participation is profitable
- Attacks are expensive
- Security from game theory

Consensus is what transforms a distributed database into a trustworthy source of truth.

Section 4: Python Implementation

Our Simple Blockchain Architecture

Components

1. Block Class

- Index (position in chain)
- Timestamp
- Transaction data
- Previous hash
- Nonce (for mining)
- Hash (calculated)

2. Blockchain Class

- Chain (list of blocks)
- Difficulty setting
- Genesis block creation
- Block addition
- Chain validation

Simplifications

Our demo omits (for clarity):

- Network communication
- Transaction signatures
- UTXO model
- Merkle trees
- Difficulty adjustment

Core concepts preserved:

- ✓ Cryptographic hashing
- ✓ Block linking
- ✓ Proof of Work mining
- ✓ Chain validation
- ✓ Tamper detection

This implementation captures the essential mechanics while remaining understandable.

Code: Block Class

```python
import hashlib
import time

class Block:
    def __init__(self, index, transactions, previous_hash):
        self.index = index                      # Position in chain
        self.timestamp = time.time()            # Creation time
        self.transactions = transactions        # Data payload
        self.previous_hash = previous_hash      # Link to parent
        self.nonce = 0                          # Mining counter
        self.hash = self.calculate_hash()       # Block's fingerprint

    def calculate_hash(self):
        """Combine all fields and hash them."""
        block_string = f"{self.index}{self.timestamp}{self.transactions}"
        block_string += f"{self.previous_hash}{self.nonce}"
        return hashlib.sha256(block_string.encode()).hexdigest()

    def mine_block(self, difficulty):
        """Find a nonce that produces a hash with required leading zeros."""
        target = "0" * difficulty
        while self.hash[:difficulty] != target:
            self.nonce += 1
            self.hash = self.calculate_hash()
        print(f"Block {self.index} mined: {self.hash}")
```

The `mine_block` method implements Proof of Work – finding a valid nonce.

Code: Blockchain Class

```python
class Blockchain:
    def __init__(self, difficulty=2):
        self.chain = [self.create_genesis_block()]  # Start with genesis
        self.difficulty = difficulty                 # Mining difficulty

    def create_genesis_block(self):
        """Create the first block in the chain."""
        return Block(0, "Genesis Block", "0")

    def get_latest_block(self):
        """Return the most recent block."""
        return self.chain[-1]

    def add_block(self, transactions):
        """Create and mine a new block, then add to chain."""
        new_block = Block(
            len(self.chain),                    # Next index
            transactions,                        # Transaction data
            self.get_latest_block().hash        # Link to previous
        )
        new_block.mine_block(self.difficulty)   # Do the work
        self.chain.append(new_block)            # Add to chain
```

Each new block references the previous block's hash, creating the chain.

Code: Mining Demonstration

```python
if __name__ == "__main__":
    # Create blockchain with difficulty 2 (hash must start with "00")
    print("Creating blockchain with difficulty 2...")
    bc = Blockchain(difficulty=2)

    # Add some transactions
    print("\nAdding transactions...")
    bc.add_block("Alice sends 10 BTC to Bob")
    bc.add_block("Bob sends 5 BTC to Charlie")
    bc.add_block("Charlie sends 2 BTC to Dave")
```

Expected Output:

```
Creating blockchain with difficulty 2...

Adding transactions...
Block 1 mined: 00a7f3b2c4d5e6f7...  (found after ~256 attempts)
Block 2 mined: 003e8f9a1b2c3d4e...  (found after ~512 attempts)
Block 3 mined: 00d4c5b6a7980123...  (found after ~128 attempts)
```

Higher difficulty = more leading zeros required = exponentially more attempts needed.

Code: Chain Validation

```python
def is_chain_valid(self):
    """Verify the integrity of the entire blockchain."""
    for i in range(1, len(self.chain)):
        current = self.chain[i]
        previous = self.chain[i-1]

        # Check 1: Hash must be valid
        if current.hash != current.calculate_hash():
            print(f"Block {i}: Hash mismatch!")
            return False

        # Check 2: Previous hash must match
        if current.previous_hash != previous.hash:
            print(f"Block {i}: Chain broken!")
            return False

    return True

# Test validation
print(f"\nBlockchain valid: {bc.is_chain_valid()}")  # True

# Attempt tampering
bc.chain[1].transactions = "Alice sends 1000 BTC to Eve"
print(f"After tampering: {bc.is_chain_valid()}")    # False
```

Tampering changes the hash, which breaks the chain link – instantly detectable.

Live Demo: Try It Yourself

Running the Demo

1. Navigate to the code directory:

```
cd code/
```

2. Run the complete demo:

```
python simple_blockchain.py
```

3. Observe:
   - Block mining output
   - Hash values
   - Validation results
   - Tampering detection

Experiments to Try

1. Change difficulty:

- Set `difficulty=3`
- Notice mining takes longer
- Each $+1$ difficulty $\approx$ 16x slower

2. Tamper with blocks:

- Modify a transaction
- Run validation
- See it fail

3. Add more blocks:

- Create longer chain
- Validate entire chain

Hands-on experimentation is the best way to understand blockchain mechanics.

Section 5: Summary

Key Takeaways

Blockchain Fundamentals

1. **Distributed ledger:** No central authority
2. **Immutability:** Cryptographic linking prevents tampering
3. **Consensus:** Network agrees on truth
4. **Transparency:** Anyone can verify

Block Structure

- Header: metadata (hash, prev_hash, nonce)
- Body: transaction data
- Merkle root: efficient verification

Consensus Mechanisms

- **PoW:** Computational work proves legitimacy
- **PoS:** Economic stake proves commitment
- Both solve double-spending

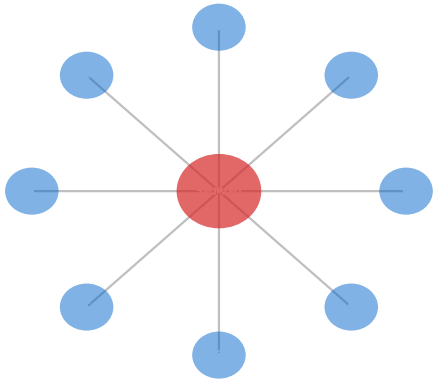
Core Innovation

> *"Don't trust, verify"*
>
> Blockchain enables trustless transactions between strangers through cryptography and game theory

These fundamentals apply to all blockchains – Bitcoin, Ethereum, and beyond.

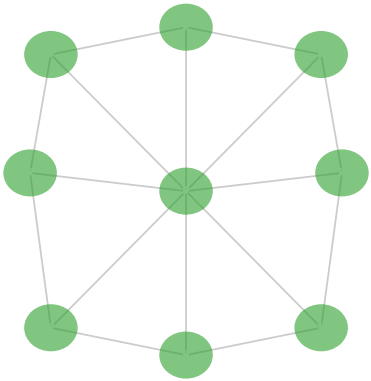
Network Architecture Comparison

Centralized Network



- Single point of failure

Decentralized Network



- No single point of failure

Coming Up: Lesson 2 – Cryptography

Topics We'll Cover

- Hash functions in depth
  - SHA-256 internals
  - Properties and security
- Public key cryptography
  - Key pairs
  - Elliptic curves
- Digital signatures
  - ECDSA
  - Transaction signing
- Addresses and wallets

Why Cryptography Matters

Cryptography provides:

- → **Integrity:** Data hasn't been changed
- → **Authentication:** Proves identity
- → **Non-repudiation:** Can't deny signing
- → **Privacy:** Selective disclosure

Preparation:

- Review modular arithmetic
- Basic understanding of prime numbers

Cryptography is the foundation that makes blockchain security possible.

Questions?

Discussion Topics:

What problems could blockchain solve in your field?
When is a blockchain *not* the right solution?
How might PoW and PoS evolve?

Resources and References

Essential Reading

- Nakamoto, S. (2008): *Bitcoin: A Peer-to-Peer Electronic Cash System*
- Antonopoulos, A.: *Mastering Bitcoin* (O'Reilly)
- Narayanan et al.: *Bitcoin and Cryptocurrency Technologies*

Online Resources

- bitcoin.org – Official documentation
- ethereum.org – Ethereum docs
- blockchain.com – Block explorer

Interactive Learning

- Anders Brownworth's blockchain demo
- CryptoZombies (Solidity)
- Coursera: Bitcoin and Cryptocurrencies

Course Materials

- Slides and code available online
- Python examples: `code/` directory
- Practice exercises in assignments

Contact:
joerg.osterrieder@uni.edu

The Bitcoin whitepaper is only 9 pages – highly recommended reading!