

Lesson 19: Ethereum and Smart Contracts

Module 2: Blockchain Fundamentals

Digital Finance

Bitcoin's Limitations: Why Ethereum?

Bitcoin Script:

- Not Turing-complete (no loops)
- Limited expressiveness
- Designed for simple transfers
- No complex state

Ethereum Vision (Vitalik Buterin, 2013):

- Turing-complete programming
- Decentralized applications (dApps)
- "World Computer"
- Programmable money and agreements

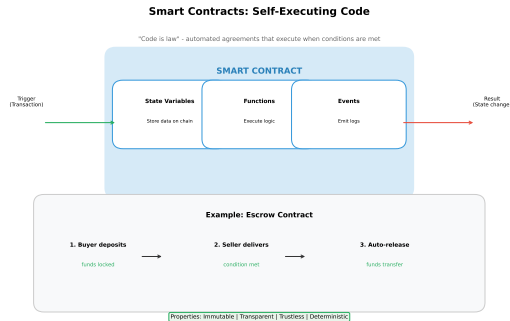
Bitcoin vs Ethereum: Key Differences

	Bitcoin	Ethereum
Purpose	Digital gold / Store of value	World computer / DApps platform
Consensus	Proof of Work	Proof of Stake (since 2022)
Block Time	~10 minutes	~12 seconds
Language	Bitcoin Script (limited)	Solidity (Turing-complete)
Supply	21M cap (deflationary)	No cap (minimal inflation)
Smart Contracts	Basic (multi-sig, timelocks)	Full programmability
Use Cases	Payments, savings	DeFi, NFTs, DAOs, gaming
Market Cap	~\$800B (Dec 2024)	~\$350B (Dec 2024)

Key Insight: Bitcoin = digital gold, Ethereum = programmable money

Source: CoinMarketCap, Blockchain.com, Ethereum.org (Dec 2024)

Understanding limitations helps identify appropriate use cases and avoid over-engineering.



Source: Szabo, "Smart Contracts" (1994), Ethereum Yellow Paper

Definition: Self-executing programs stored on blockchain, automatically enforcing agreements

Properties:

- Deterministic execution (same input → same output)
- Immutable once deployed
- Transparent and trustless

Smart contracts enable programmable, self-executing agreements without intermediaries.

Account Model: Ethereum's Design

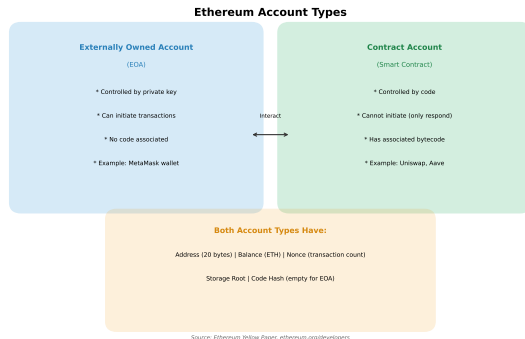
Two Account Types:

1 Externally Owned Accounts (EOAs):

- Controlled by private key
- Can send transactions
- No code

2 Contract Accounts:

- Controlled by code
- Triggered by transactions
- Store state



State: Each account has balance, nonce, code (contracts only), storage

Ethereum pioneered smart contracts and remains the dominant platform for DeFi and NFTs.

Ethereum Virtual Machine (EVM)

Evm Architecture



[SYNTHETIC DATA]

EVM Properties:

- Stack-based virtual machine (256-bit words)
- Bytecode execution (compiled from Solidity, Vyper, etc.)
- Isolated execution environment (sandboxed)
- Deterministic (no randomness or external calls without oracles)
- Replicated across all nodes

Ethereum pioneered smart contracts and remains the dominant platform for DeFi and NFTs.

Gas: Metering Computation

Why Gas?

- Prevent infinite loops (halting problem)
- Prioritize transactions
- Compensate miners/validators
- Align incentives

Gas Mechanics:

- Each operation costs gas
- User sets gas limit + gas price
- Unused gas refunded
- Out of gas → revert (but gas consumed)

Ethereum Gas: Fuel for the World Computer

$$\text{Transaction Fee} = \text{Gas Used} \times \text{Gas Price}$$

(in ETH) (units) (gwei/gas)

Common Gas Costs

Simple transfer	21,000 gas
ERC-20 transfer	~65,000 gas
Uniswap swap	~150,000 gas
NFT mint	~100,000 gas
Deploy contract	~500,000+ gas

Why Gas Exists

- * Prevents infinite loops
- * Compensates validators
- * Allocates scarce resources
- * Spam protection

Gas Limit vs Gas Used

Gas Limit: Maximum you're willing to spend (set by user)

Gas Used: Actual computation consumed (unused gas refunded)

If Gas Used > Gas Limit: Transaction fails, gas still consumed

Source: Ethereum Yellow Paper, etherscan.io/gastracker

Key concepts from this slide inform practical applications in finance.

Gas Costs: Operation Examples

Operation	Gas Cost	Rationale
ADD (arithmetic)	3	Simple computation
MUL (multiplication)	5	Slightly more complex
SSTORE (write storage)	20,000	Permanent state change
SLOAD (read storage)	2,100	Storage access
CREATE (deploy contract)	32,000	Base cost + code size
Transaction (base)	21,000	Minimum for any transaction

Design: Expensive operations (storage, deployment) cost more to prevent spam

Case studies provide concrete evidence of technology impact and adoption patterns.

Example: Simple ETH Transfer

- Gas limit: 21,000
- Gas price: 50 gwei ($1 \text{ gwei} = 10^{-9} \text{ ETH}$)
- Total fee: $21,000 \times 50 \times 10^{-9} = 0.00105 \text{ ETH}$

Example: Token Transfer (ERC-20)

- Gas limit: 65,000 (contract interaction)
- Gas price: 50 gwei
- Total fee: $65,000 \times 50 \times 10^{-9} = 0.00325 \text{ ETH}$

Example: Complex DeFi Swap

- Gas limit: 300,000 (multiple contract calls)
- Gas price: 100 gwei (priority)
- Total fee: $300,000 \times 100 \times 10^{-9} = 0.03 \text{ ETH}$ ($\sim \$60$ at $\$2000/\text{ETH}$)

Key concepts from this slide inform practical applications in finance.

Legacy Fee Market: First-Price Auction

Pre-EIP-1559 (before Aug 2021):

- Users bid gas price
- Miners select highest bids
- First-price auction
- Overpay or get stuck

Problems:

- Fee estimation difficult
- High volatility
- Miner extractable value (MEV)



Key concepts from this slide inform practical applications in finance.

Eip1559 Structure



[SYNTHETIC DATA]

New Fee Structure:

$$\text{Total Fee} = \text{Base Fee (burned)} + \text{Priority Fee (to validator)}$$

Key Features:

- **Base Fee:** Algorithmically adjusted, burned (deflationary)
- **Priority Fee:** Tip to validators for faster inclusion
- **Max Fee:** User sets maximum willing to pay

Key concepts from this slide inform practical applications in finance.

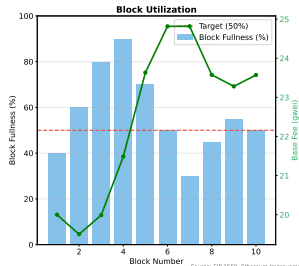
EIP-1559 Base Fee Dynamics

Base Fee Adjustment:

- Target: 15M gas per block
- If block > 15M: base fee ↑ 12.5%
- If block < 15M: base fee ↓ 12.5%
- Max block size: 30M gas

Formula:

$$\Delta_{\text{base}} = \frac{\text{Gas Used} - 15M}{15M} \times \frac{\text{Base Fee}}{8}$$

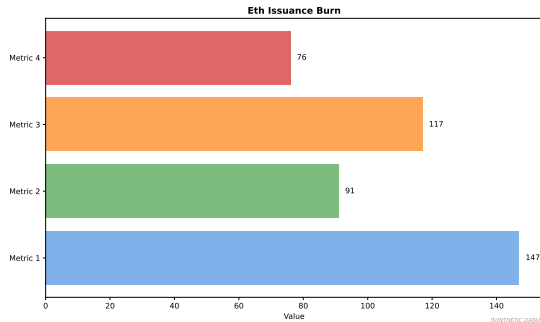


EIP-1559 Fee Structure



Key concepts from this slide inform practical applications in finance.

Fee Burning: Deflationary Pressure



Issuance: ~1,600 ETH/day (PoS rewards)

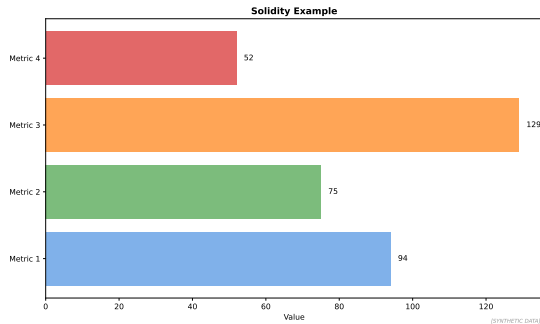
Burn: Variable, depends on network usage (avg ~1,000–2,000 ETH/day)

Net Result: Ethereum can be deflationary when usage is high (burn > issuance)

Key concepts from this slide inform practical applications in finance.

Solidity: High-Level Smart Contract Language

Example: Simple Token Contract

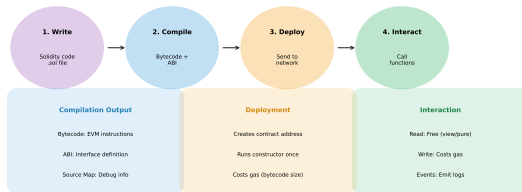


Key Features:

- Syntax similar to JavaScript/C++
- Compiles to EVM bytecode
- Supports inheritance, libraries, interfaces
- Built-in types: address, uint256, mapping, array

Smart contracts enable programmable, self-executing agreements without intermediaries.

Smart Contract Lifecycle



Note: Once deployed, contract code cannot be changed (immutable)

Source: Solidity Documentation, [ethereum.org/developers](https://ethereum.org/en/developers/)

Stages:

- 1 **Development:** Write Solidity code
- 2 **Compilation:** Compile to bytecode + ABI
- 3 **Deployment:** Send creation transaction (costs gas)
- 4 **Interaction:** Call functions via transactions
- 5 **Immutability:** Cannot modify code (only state)

Technology adoption follows predictable patterns—timing matters for investment decisions.

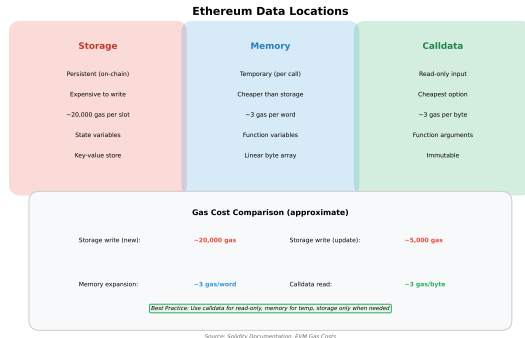
Contract Storage: Persistent State

Storage Layout:

- Key-value store (256-bit slots)
- Permanent (persists between calls)
- Expensive (SSTORE = 20,000 gas)
- Optimizations: packing, mappings

Memory vs Storage:

- **Storage:** Permanent, expensive
- **Memory:** Temporary, cheap, cleared after execution



Key concepts from this slide inform practical applications in finance.

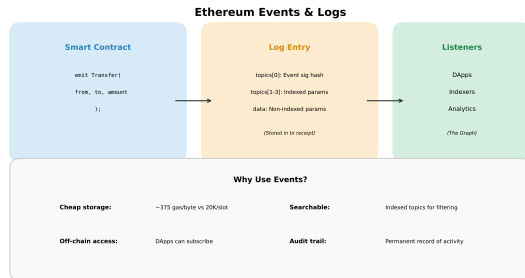
Events and Logs

Purpose:

- Emit structured data from contracts
- Stored in transaction receipts
- Indexed for efficient querying
- Off-chain applications listen to events

Use Cases:

- Token transfers (Transfer event)
- Price updates (oracles)
- Audit trails
- UI updates (wallets, dApps)



Common Events: Transfer, Approval, Swap, Mint, Burn

Source: Solidity Events Documentation, ethereum.org

Key concepts from this slide inform practical applications in finance.

External Calls: Composability and Risks

Contract Interactions:

- Contracts can call other contracts
- Enables composability ("money legos")
- DeFi protocols build on each other

Risks:

- Reentrancy attacks
- Uncontrolled gas consumption
- Malicious contract logic
- Dependency vulnerabilities

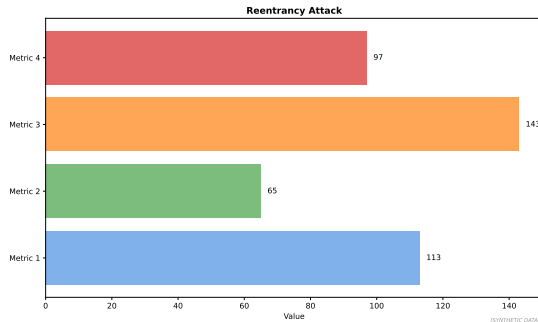
External Call Flow



[SYNTHETIC DATA]

Risk management is essential for financial stability and profitability.

Reentrancy: The DAO Hack (2016)



Vulnerability:

- Contract sends ETH before updating balance
- Recipient (attacker contract) calls back into vulnerable contract
- Recursively drains funds before balance updated

The DAO Result: 3.6M ETH stolen (\$70M), led to Ethereum hard fork (ETH/ETC split)

Key concepts from this slide inform practical applications in finance.

Oracles: Bridging On-Chain and Off-Chain

Problem:

- Smart contracts cannot access external data
- No internet, APIs, randomness
- Determinism requirement

Oracle Solution:

- Third-party data feeds
- Price feeds (ETH/USD)
- Weather data
- Sports scores

Oracle Architecture



[SYNTHETIC DATA]

Chainlink: Decentralized oracle network, aggregates multiple data sources

AI and ML are transforming financial services through automation and prediction.

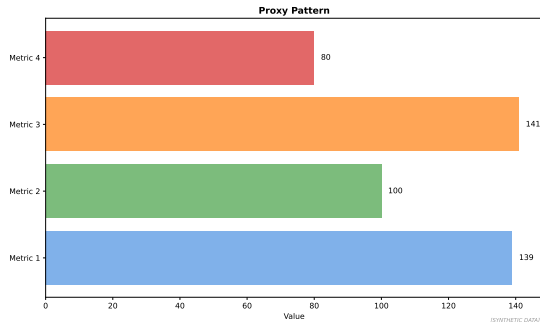
Gas Optimization Strategies

- **Storage Packing:** Use uint128 instead of uint256 where possible (fit in one slot)
- **Avoid Storage Writes:** Use memory for temporary data
- **Short-Circuit Logic:** require checks early, minimize wasted gas
- **Batch Operations:** Aggregate multiple actions in one transaction
- **Events over Storage:** Emit events instead of storing historical data
- **Minimal Contract Size:** Lower deployment costs
- **Use Libraries:** Reusable code via DELEGATECALL

Example: Storing 100 values individually: ~2M gas. Packed into single array: ~500K gas

Key concepts from this slide inform practical applications in finance.

Upgradeable Contracts: Proxy Pattern



Design:

- **Proxy Contract:** Fixed address, DELEGATECALL to implementation
- **Implementation Contract:** Contains logic, upgradeable
- **Storage:** Stored in proxy, preserved across upgrades

Trade-off: Flexibility vs decentralization (admin can change logic)

Key concepts from this slide inform practical applications in finance.

Summary

- **Ethereum:** World computer, Turing-complete smart contracts, account model
- **EVM:** Stack-based VM, executes bytecode, deterministic, replicated
- **Gas:** Meters computation, prevents infinite loops, aligns incentives
- **EIP-1559:** Base fee (burned) + priority fee, deflationary pressure
- **Solidity:** High-level language, compiles to bytecode
- **Risks:** Reentrancy, oracles, immutability challenges

Next Lesson: Tokens – ERC-20, ERC-721 (NFTs), and token economics