Lesson 19: Ethereum and Smart Contracts

Module 2: Blockchain Fundamentals

Digital Finance

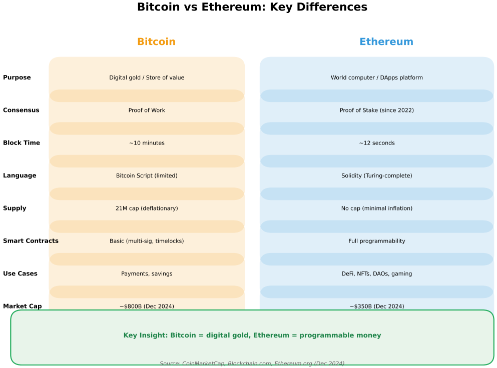
Bitcoin's Limitations: Why Ethereum?

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



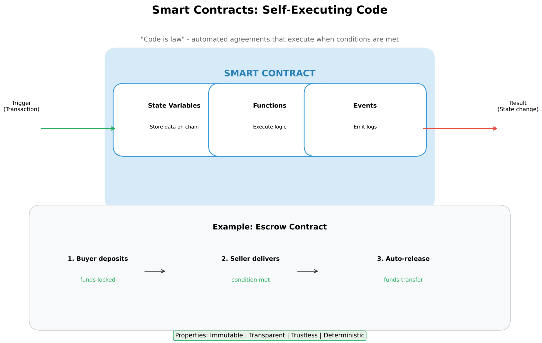
Bitcoin vs Ethereum: Key Differences

	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)

Smart Contracts: Code as Law



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

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

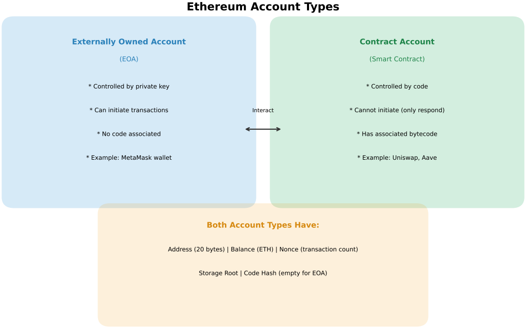
Properties:

- Deterministic execution (same input → same output)
- Immutable once deployed
- Transparent (anyone can verify)
- Trustless (no 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



Ethereum Account Types

Externally Owned Account

(EOA)

* Controlled by private key

* Can initiate transactions

* No code associated

* Example: MetaMask wallet

Interact

Contract Account

(Smart Contract)

* Controlled by code

* Cannot initiate (only respond)

* Has associated bytecode

* Example: Uniswap, Aave

Both Account Types Have:

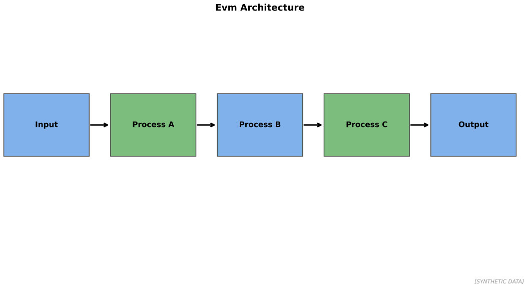
Address (20 bytes) | Balance (ETH) | Nonce (transaction count)

Storage Root | Code Hash (empty for EOA)

Source: Ethereum Yellow Paper, ethereum.org/developers

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

Ethereum Virtual Machine (EVM)



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

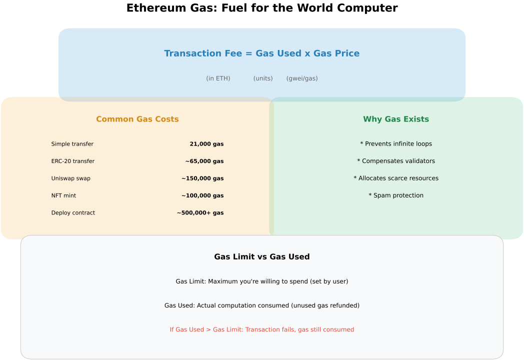
Gas: Metering Computation

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

Transaction Fee = Gas Used x 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

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

Transaction Cost Calculation

Example: Simple ETH Transfer

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

Example: Token Transfer (ERC-20)

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

Example: Complex DeFi Swap

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

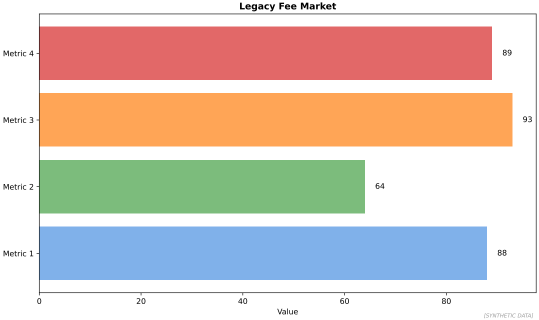
Legacy Fee Market: First-Price Auction

Pre-EIP-1559 (before Aug 2021):

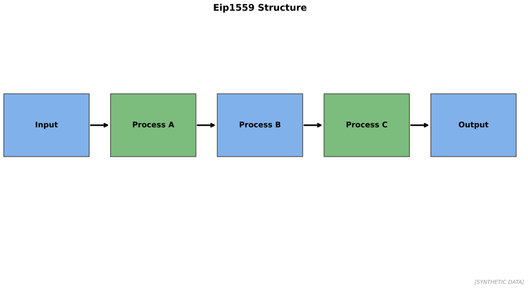
- Users bid gas price
- Miners select highest bids
- First-price auction
- Overpay or get stuck

Problems:

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



EIP-1559: Fee Market Reform (August 2021)



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

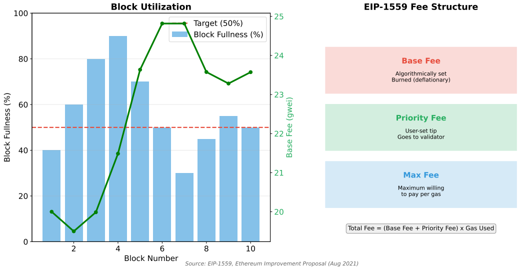
EIP-1559 Base Fee Dynamics

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

Base Fee
Algorithmically set
Burned (deflationary)

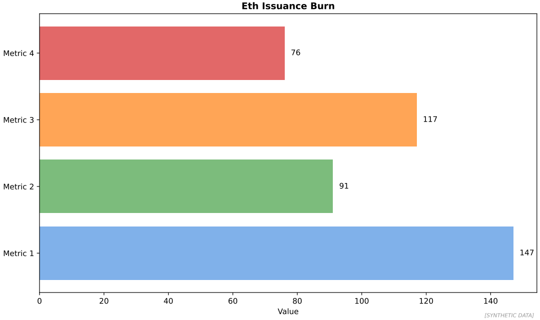
Priority Fee
User-set tip
Goes to validator

Max Fee
Maximum willing
to pay per gas

Total Fee = (Base Fee + Priority Fee) x Gas Used

Source: EIP-1559, Ethereum Improvement Proposal (Aug 2021)

Fee Burning: Deflationary Pressure

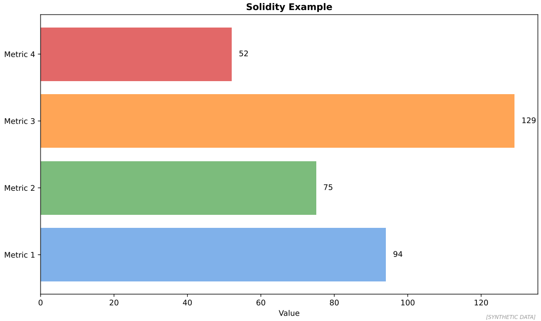


Issuance: ∼1,600 ETH/day (PoS rewards)
Burn: Variable, depends on network usage (avg ∼1,000–2,000 ETH/day)

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

Solidity: High-Level Smart Contract Language

Example: Simple Token Contract

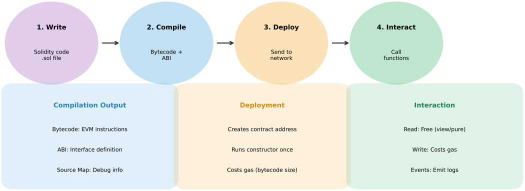


Key Features:

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

Smart Contract Lifecycle

Smart Contract Lifecycle



1. Write

Solidity code
.sol file

2. Compile

Bytecode +
ABI

3. Deploy

Send to
network

4. Interact

Call
functions

Compilation Output

Bytecode: EVM instructions

ABI: Interface definition

Source Map: Debug info

Deployment

Creates contract address

Runs constructor once

Costs gas (bytecode size)

Interaction

Read: Free (view/pure)

Write: Costs gas

Events: Emit logs

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

Source: Solidity Documentation, ethereum.org/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)

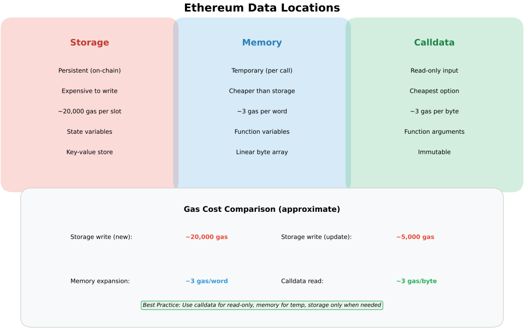
Contract Storage: Persistent State

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



Ethereum Data Locations

Storage	Memory	Calldata
Persistent (on-chain)	Temporary (per call)	Read-only input
Expensive to write	Cheaper than storage	Cheapest option
~20,000 gas per slot	~3 gas per word	~3 gas per byte
State variables	Function variables	Function arguments
Key-value store	Linear byte array	Immutable

Gas Cost Comparison (approximate)

Storage write (new):	~20,000 gas	Storage write (update):	~5,000 gas
Memory expansion:	~3 gas/word	Calldata read:	~3 gas/byte

Best Practice: Use calldata for read-only, memory for temp, storage only when needed

Source: Solidity Documentation, EVM Gas Costs

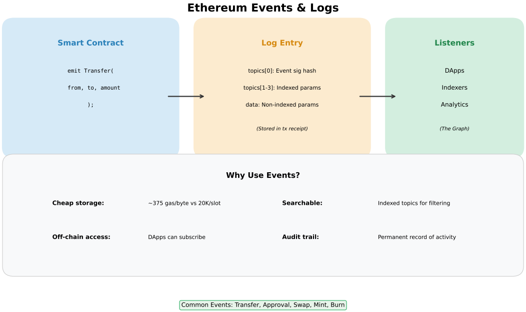
Events and Logs

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)



Ethereum Events & Logs

Smart Contract

```
emit Transfer(
  from, to, amount
);
```

Log Entry

topics[0]: Event sig hash

topics[1-3]: Indexed params

data: Non-indexed params

(Stored in tx receipt)

Listeners

DApps

Indexers

Analytics

(The Graph)

Why Use Events?

Cheap storage:	~375 gas/byte vs 20K/slot	**Searchable:**	Indexed topics for filtering
Off-chain access:	DApps can subscribe	**Audit trail:**	Permanent record of activity

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

Source: Solidity Events Documentation, ethereum.org

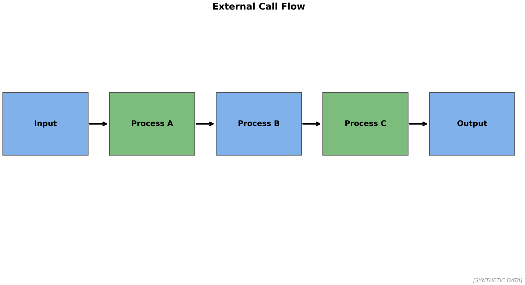
External Calls: Composability and Risks

Contract Interactions:

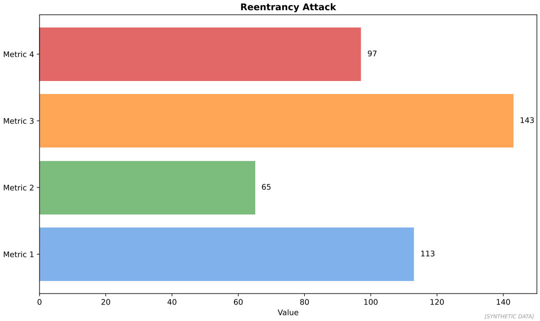
- Contracts can call other contracts
- Enables composability ("money legos")
- DeFi protocols build on each other

Risks:

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



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)

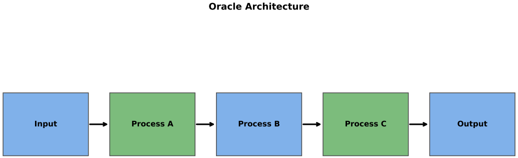
Oracles: Bridging On-Chain and Off-Chain

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



[SYNTHETIC DATA]

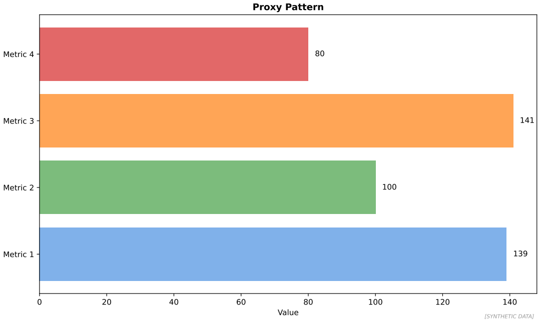
Chainlink: Decentralized oracle network, aggregates multiple data sources

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: $\sim$2M gas. Packed into single array: $\sim$500K gas

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)

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