

# 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

`charts/lesson_19/bitcoin_vs_ethereum.pdf`

`charts/lesson_19/smart_contract_concept.pdf`

## Two Account Types:

### ① Externally Owned Accounts (EOAs):

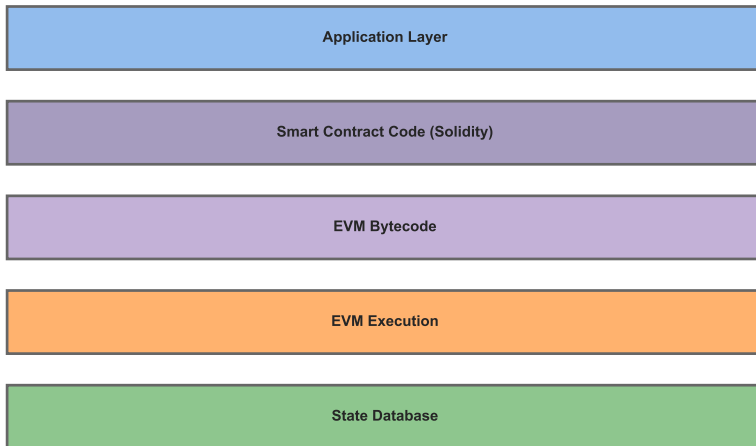
- Controlled by private key
- Can send transactions
- No code

### ② Contract Accounts:

- Controlled by code
- Triggered by transactions
- Store state

`charts/lesson_19/account_types.pdf`

## Ethereum Virtual Machine (EVM) Architecture



## 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)

`charts/lesson_19/gas_mechanism.pdf`

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

## 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  $\$2000/\text{ETH}$ )



### Pre-EIP-1559 (before Aug 2021):

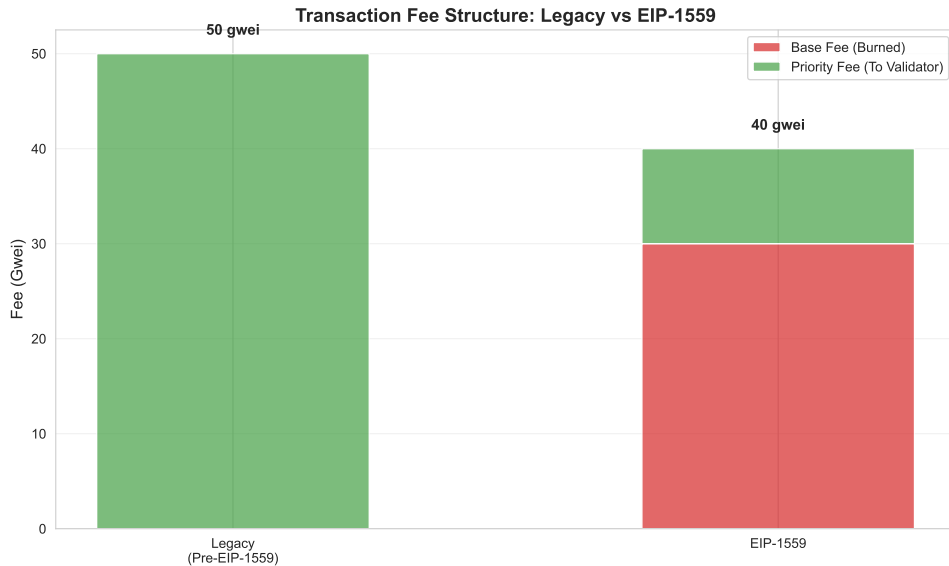
- Users bid gas price
- Miners select highest bids
- First-price auction
- Overpay or get stuck

### Problems:

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

`charts/lesson_19/legacy_fee_market.pdf`

# EIP-1559: Fee Market Reform (August 2021)



New Fee Structure:

### Base Fee Adjustment:

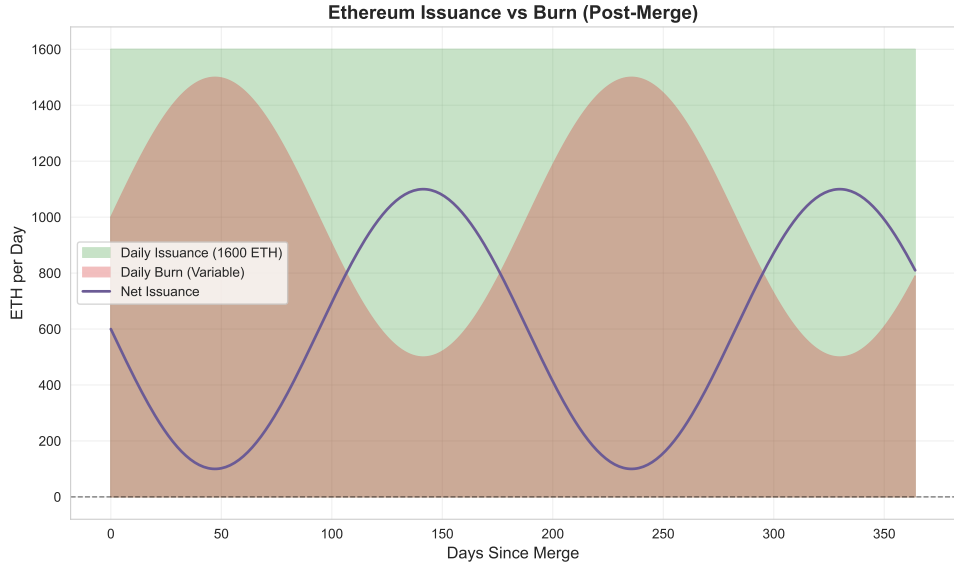
- Target: 15M gas per block
- If block > 15M: base fee  $\uparrow$  12.5%
- If block < 15M: base fee  $\downarrow$  12.5%
- Max block size: 30M gas

### Formula:

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

[charts/lesson\\_19/base\\_fee\\_adjustment.pdf](#)

# Fee Burning: Deflationary Pressure



**Issuance:**  $\sim 1,600$  ETH/day (PoS rewards)

## Example: Simple Token Contract

`charts/lesson_19/solidity_example.pdf`

`charts/lesson_19/contract_lifecycle.pdf`

## 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

[charts/lesson\\_19/storage\\_vs\\_memory.pdf](#)

## 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)

`charts/lesson_19/events_architecture.pdf`



### 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

[charts/lesson\\_19/external\\_call\\_flow.pdf](#)

`charts/lesson_19/reentrancy_attack.pdf`

# 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

[charts/lesson\\_19/oracle\\_architecture.pdf](#)

- **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:  $\sim 2\text{M}$  gas. Packed into single array:  $\sim 500\text{K}$  gas

`charts/lesson_19/proxy_pattern.pdf`

- **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