L14: Gas Mechanics

Module B: Ethereum & Smart Contracts

Blockchain & Cryptocurrency Course

December 2025

Learning Objectives

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

- Explain what gas is and why Ethereum uses it
- Calculate transaction costs using gas price and gas limit
- Explain EIP-1559's base fee and priority fee mechanism
- Identify gas costs for different EVM operations
- Apply optimization techniques to reduce gas consumption
- Analyze real-world gas usage patterns

The Problem: How do we price scarce computation fairly?

The Challenge
How do we price scarce computational resources on a public blockchain without enabling denial-of-service attacks? Ethereum's EVM must process transactions from anyone, but unlimited computation could halt the network.

Why It Matters

- Without gas limits, infinite loops could halt the entire network
- Historical example: EIP-1559 (August 2021) addressed fee market inefficiencies, but fee spikes still occur during high-demand events (e.g., NFT mints)

What We Need

- Value creation and capture
- Market-based pricing that reflects true computational cost without manual estimation

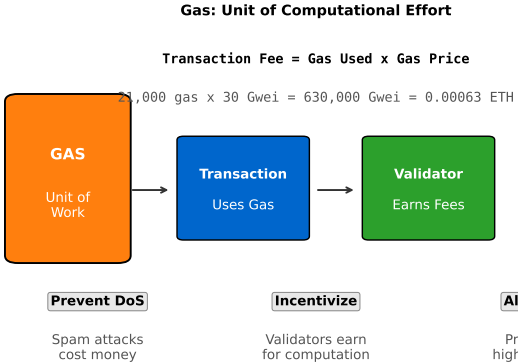
The Cryptoeconomics Question
Creating and distributing economic value

Today's lesson: How Gas Mechanics addresses this challenge

What is Gas?

Gas is a unit of computational effort in Ethereum:

- Measures the cost of executing operations on the EVM
- Prevents infinite loops and spam attacks
- Compensates validators for computation and storage



Why Gas Exists

Three Critical Functions:

1. **Prevent Denial-of-Service Attacks:**
   - Without gas, infinite loops could halt the network
   - Attackers would need to pay for computational resources
2. **Incentivize Validators:**
   - Validators earn transaction fees for including transactions
   - Higher gas price = higher priority in block inclusion
3. **Resource Allocation:**
   - Limited block gas limit (e.g., 30,000,000 gas per block)
   - Prioritizes transactions willing to pay more

Ether Denominations

Ether units from smallest to largest:

Unit	Wei Value	Typical Use
Wei	1	Smallest unit (like satoshi)
Gwei (Shannon)	10^9	Gas prices
Microether (Szabo)	10^{12}	-
Milliether (Finney)	10^{15}	-
Ether	10^{18}	Main unit

Most Common:

- **Gwei (Gigawei):** Standard unit for gas prices (1 Gwei = 10^9 Wei)
- **Ether:** User-facing unit (1 ETH = 10^{18} Wei)

Gas Calculation: Pre-EIP-1559

Legacy Transaction Fee Model (before August 2021):
Formula:

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

Example:

- Gas Used: 21,000 (simple ETH transfer)
- Gas Price: 50 Gwei (user-specified)
- Transaction Fee: $21{,}000 \times 50 = 1{,}050{,}000$ Gwei $= 0.00105$ ETH

Challenges:

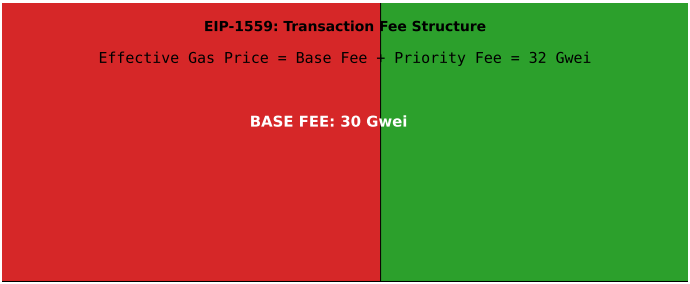
- Users had to manually estimate gas price
- Overpaying was common to ensure inclusion
- No refund if gas price was too high

EIP-1559: London Hard Fork (August 2021)

Major overhaul of gas fee mechanism:

- **Base Fee:** Algorithmically determined, burned (removed from circulation)
- **Priority Fee (Tip):** User-specified tip to validator for inclusion
- **Max Fee:** Maximum gas price user is willing to pay

EIP-1559 Fee Breakdown (August 2021)



EIP-1559 Fee Calculation

New Formula:

$$\text{Transaction Fee} = \text{Gas Used} \times (\text{Base Fee} + \text{Priority Fee})$$

With cap:

$$\text{Effective Gas Price} = \min(\text{Base Fee} + \text{Priority Fee}, \text{Max Fee})$$

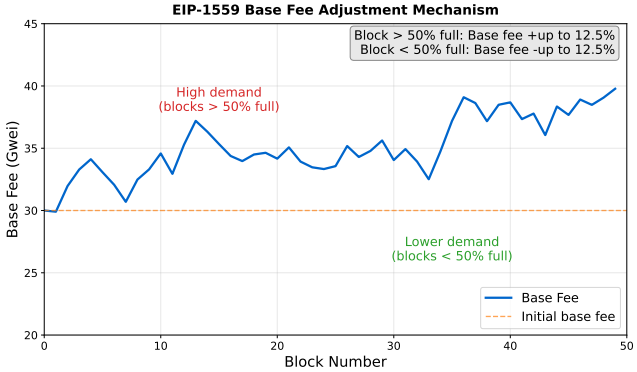
Example:

- Gas Used: 21,000
- Base Fee: 30 Gwei (set by protocol)
- Priority Fee: 2 Gwei (user tip)
- Effective Gas Price: $\min(30 + 2, 50) = 32$ Gwei
- Transaction Fee: $21{,}000 \times 32 = 672{,}000$ Gwei $= 0.000672$ ETH
- Burned: $21{,}000 \times 30 = 630{,}000$ Gwei

Base Fee Adjustment Mechanism

Dynamic base fee targets 50% full blocks:

- Target gas per block: 15,000,000 (50% of 30M limit)
- If block $>$ 50% full: Base fee increases by max 12.5%
- If block $<$ 50% full: Base fee decreases by max 12.5%



Gas Limit vs Gas Used

Understanding the difference:

Gas Limit:

- Maximum gas transaction may consume
- Set by user before sending transaction
- Acts as safety cap
- Unused gas is refunded

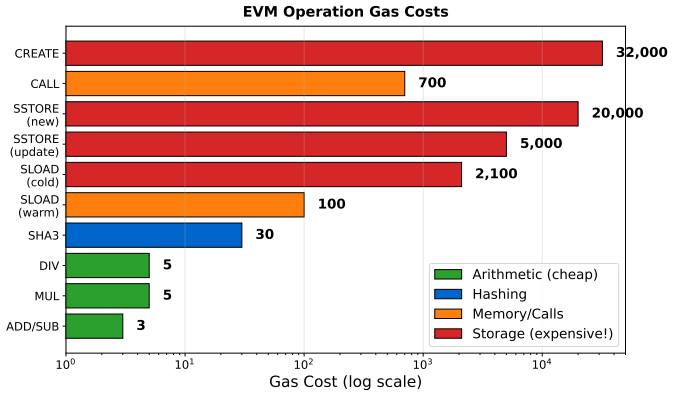
Gas Used:

- Actual gas consumed by transaction
- Determined by operations executed
- Used for fee calculation
- Visible on Etherscan

Common Values: Simple transfer: 21,000 — ERC-20 transfer: 45,000-65,000 — Complex contract: 100,000+

Gas Costs by Operation

Every EVM opcode has a fixed gas cost:



Key Insight: Storage operations (SSTORE, SLOAD) are by far the most expensive

Gas Costs by Operation (Table)

Operation	Description	Gas Cost
ADD, SUB, MUL	Arithmetic operations	3
DIV, MOD	Division/modulo	5
SHA3 (Keccak-256)	Hash function	30 + 6/word
SLOAD	Load from storage	100 (warm) / 2100 (cold)
SSTORE	Write to storage	20,000 (new) / 5,000 (update)
CALL	External contract call	700 + value transfer costs
CREATE	Deploy contract	32,000 + code size

Cold vs Warm Access (EIP-2929):

- **Cold:** First access to storage slot in transaction (expensive)
- **Warm:** Subsequent accesses to same slot (cheaper)

Storage: The Gas Guzzler

Why storage is expensive:

- Persists data across all nodes forever
- Requires disk I/O (slower than RAM)
- State bloat affects all future nodes

Storage Gas Costs (EIP-2929, EIP-2200):

- **SSTORE (set to non-zero from zero):** 20,000 gas
- **SSTORE (update non-zero):** 5,000 gas
- **SSTORE (set to zero):** 5,000 gas + 15,000 refund
- **SLOAD (cold access):** 2,100 gas (first access in transaction)
- **SLOAD (warm access):** 100 gas (subsequent accesses)

Example: Storing one 256-bit word (32 bytes) at 32 Gwei: 0.00064 ETH = $1.28 at $2000/ETH

Gas Optimization: Storage Patterns

Inefficient: Multiple SSTOREs

```
contract Inefficient {
    uint256 public value1;
    uint256 public value2;
    uint256 public value3;

    function updateAll(uint256 v1, uint256 v2, uint256 v3) public {
        value1 = v1;  // 20,000 gas (or 5,000 if updating)
        value2 = v2;  // 20,000 gas
        value3 = v3;  // 20,000 gas
    }
    // Total: 60,000 gas for 3 writes
}
```

Efficient: Packed Storage

```
contract Efficient {
    uint256 public packedValues;  // Pack 3 uint85 values in one slot

    function updateAll(uint85 v1, uint85 v2, uint85 v3) public {
        packedValues = uint256(v1) | (uint256(v2) << 85) | (uint256(v3) << 170);
    }
    // Total: 20,000 gas for single write (3x cheaper!)
}
```

Gas Optimization: Memory vs Storage

Use memory for temporary data:
Inefficient: Storage for Temporary Array

```
contract Inefficient {
    uint256[] public tempArray;  // Storage
    function processData(uint256[] calldata input) public {
        delete tempArray;
        for (uint i = 0; i < input.length; i++) {
            tempArray.push(input[i] * 2);  // SSTORE per iteration
        }
    }
}
```

Efficient: Memory Array

```
contract Efficient {
    function processData(uint256[] calldata input) public {
        uint256[] memory tempArray = new uint256[](input.length);
        for (uint i = 0; i < input.length; i++) {
            tempArray[i] = input[i] * 2;  // Memory write (cheap)
        }
    }
}
```

Gas Optimization: Short-Circuiting

Exploit boolean evaluation order:
Inefficient: Expensive Check First

```
function transfer(address to, uint256 amount) public {
    require(balances[msg.sender] >= amount && to != address(0), "Invalid");
    // If to == address(0), still loads balances[msg.sender] (2100 gas SLOAD)
}
```

Efficient: Cheap Check First

```
function transfer(address to, uint256 amount) public {
    require(to != address(0) && balances[msg.sender] >= amount, "Invalid");
    // If to == address(0), immediately fails (no SLOAD)
}
```

Principle: Place cheaper conditions first in logical AND (&&)
Savings: 2100 gas when early condition fails

Gas Optimization: Event Logs vs Storage

Events are much cheaper than storage:

Storage: Accessible on-chain, 20,000 gas per new slot

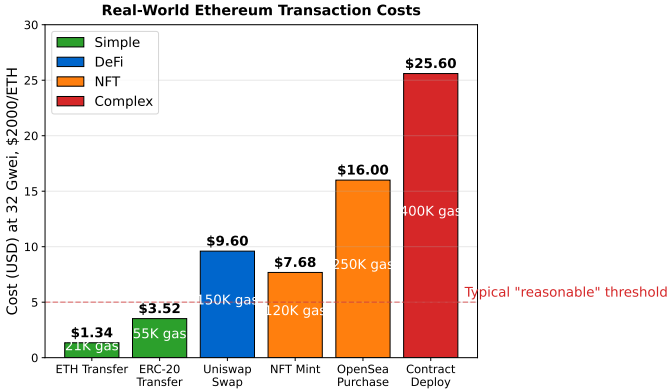
Events: Not accessible on-chain, 375 gas + 8/byte

```
event Transfer(address indexed from, address indexed to, uint256 amount);

function transfer(address to, uint256 amount) public {
    balances[msg.sender] -= amount;
    balances[to] += amount;
    emit Transfer(msg.sender, to, amount);  // ~1500 gas vs 20,000+ for storage
}
```

Real-World Transaction Costs

Typical gas costs on Ethereum mainnet:



Note: Costs vary significantly based on network congestion and gas prices

Real-World Gas Usage (Table)

Transaction Type	Gas Used
Simple ETH transfer	21,000
ERC-20 token transfer	45,000 - 65,000
Uniswap V2 swap	100,000 - 150,000
Uniswap V3 swap	120,000 - 185,000
NFT mint (ERC-721)	80,000 - 150,000
OpenSea NFT purchase	150,000 - 300,000
Deploy simple contract	200,000 - 500,000

At 32 Gwei (30 base + 2 tip):

- Simple transfer: 0.000672 ETH ($1.34 at $2000/ETH)
- Uniswap swap: 0.004 ETH ($8 at $2000/ETH)
- Contract deploy: 0.016+ ETH ($32+ at $2000/ETH)

Gas Refunds

Refundable Actions:

- **SSTORE to zero:** 15,000 gas refund (after paying 5,000 to clear)
- **SELFDESTRUCT:** 24,000 gas refund (contract deletion)

Refund Cap (EIP-3529): Maximum refund: 20% of gas used (prevents exploitation)

```
function clearStorage() public {
    delete largeMapping[key1];  // 5,000 cost + 15,000 refund
    delete largeMapping[key2];  // 5,000 cost + 15,000 refund
    // Gas used: 10,000, Potential: 30,000 (capped at 2,000)
}
```

EIP-4844: Blob Gas Market (Dencun Upgrade, March 2024)

Ethereum now has TWO gas markets:

- **Execution gas**: Traditional EVM operations (existing market)
- **Blob gas**: Data availability for L2 rollups (new market)
- Markets operate independently with separate base fees

Blob Gas Mechanics:

- Target: 3 blobs per block (384 KB), Maximum: 6 blobs (768 KB)
- Blob gas price adjusts like EIP-1559 (targets 50% capacity)

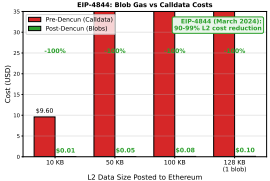
EIP-4844: Blob Gas Savings

Cost comparison for posting L2 data to Ethereum:

$122.88

$96.00

$48.00



Impact: 90-99% reduction in L2 transaction costs since Dencun upgrade

Problem Solved? Gas Mechanics

The Original Problem

How do we price scarce computation fairly?

How Gas Mechanics Solves It

- Gas units: Fixed costs per EVM operation
- EIP-1559: Dynamic base fee targets 50% blocks
- Base fee burning aligns incentives

Remaining Limitations

- Volatile fees during congestion (10x-100x spikes)
- MEV extraction undermines fairness

Open Questions

- Can fees be more predictable?
- Does it scale without L2 solutions?

Gas Mechanics partially solves fair pricing but introduces volatility and MEV trade-offs

Key Takeaways

1. **Gas Purpose:** Prevents spam/DoS, compensates validators, allocates block space
2. **EIP-1559:** Base fee (burned) + priority fee (to validator) for predictable pricing
3. **Base Fee Dynamics:** Adjusts up to 12.5% per block to target 50% full blocks
4. **Storage is Expensive:** SSTORE costs 20,000 gas (new) or 5,000 gas (update)
5. **Optimization:** Pack storage, use memory for temp data, batch operations
6. **EIP-4844:** Blob gas market reduced L2 costs by 90-99%

Discussion Questions

1. Why does EIP-1559 burn the base fee instead of giving it to validators?
2. If Ethereum's block gas limit is 30M and average block time is 12 seconds, what is the theoretical maximum transactions per second for simple ETH transfers?
3. Under what circumstances might a user set a very high max fee per gas?
4. How do Layer 2 solutions (e.g., Optimism, Arbitrum) reduce gas costs?
5. What are the tradeoffs between storing data on-chain vs using events vs off-chain storage?

Next Lesson Preview: L15 - Solidity Fundamentals

Coming up next:

- Introduction to Solidity programming language
- Data types: uint, address, string, arrays, mappings
- Functions, visibility modifiers, state mutability
- Events and error handling
- Inheritance and interfaces

Preparation:

- Install MetaMask browser extension
- Familiarize yourself with Remix IDE (remix.ethereum.org)
- Review basic programming concepts (if-else, loops, functions)

Quiz Questions (1/4)

Q1. What is the primary purpose of gas in Ethereum?
A) To make transactions expensive B) To prevent spam and compensate validators
C) To create scarcity D) To replace Ether
Answer: B – Gas prevents DoS attacks and compensates validators.

Q2. How many Gwei are in 1 Ether?
A) 10^6 B) 10^9 C) 10^{12} D) 10^{18}
Answer: B – 1 Gwei = 10^9 Wei, 1 ETH = 10^{18} Wei.

Q3. In the pre-EIP-1559 model, what was the transaction fee formula?
A) Gas Used + Gas Price B) Gas Used × Gas Price
C) Gas Limit × Gas Price D) Base Fee + Priority Fee
Answer: B – Transaction Fee = Gas Used × Gas Price (legacy).

Q4. What happens to the base fee in EIP-1559 transactions?
A) Sent to validators B) Burned (removed from circulation)
C) Returned to users D) Stored in treasury
Answer: B – EIP-1559 burns base fees to reduce ETH supply.

Q5. If a block is more than 50% full, how does the base fee change?
A) Decreases by 12.5% B) Stays the same C) Increases by up to 12.5% D) Doubles
Answer: C – Base fee increases when blocks exceed 50% capacity.

Quiz Questions (2/4)

Q6. What is the gas cost for a simple ETH transfer?
A) 10,000 B) 21,000 C) 45,000 D) 100,000
Answer: B – Simple ETH transfers cost exactly 21,000 gas.

Q7. Which EVM operation is the most expensive?
A) ADD (arithmetic) B) SHA3 (hashing) C) SLOAD (read storage) D) SSTORE (write storage)
Answer: D – SSTORE costs 20,000 gas (new) or 5,000 gas (update).

Q8. What is the difference between gas limit and gas used?
A) No difference B) Gas limit is max allowed, gas used is actual consumed
C) Gas limit is for validators only D) Gas used includes refunds
Answer: B – Gas limit is user-set cap; gas used is actual. Unused refunded.

Q9. In EIP-1559, the priority fee goes to:
A) The Ethereum Foundation B) The user as refund C) Validators D) Burned
Answer: C – Priority fee (tip) incentivizes validators to include transaction.

Q10. What is the target gas per block in Ethereum's EIP-1559 mechanism?
A) 10,000,000 B) 15,000,000 C) 30,000,000 D) 60,000,000
Answer: B – Target is 15M gas (50% of 30M block limit).

Quiz Questions (3/4)

Q11. What is the gas cost for a cold SLOAD (first storage read in a transaction)?
A) 100 B) 700 C) 2,100 D) 5,000
Answer: C – Cold SLOAD costs 2,100 gas; warm costs 100 gas (EIP-2929).

Q12. How much gas refund does setting a storage slot to zero provide?
A) 5,000 B) 10,000 C) 15,000 D) 20,000
Answer: C – Clearing storage (SSTORE to zero) refunds 15,000 gas (after 5,000 cost).

Q13. What is the maximum gas refund cap introduced by EIP-3529?
A) 10% of gas used B) 20% of gas used C) 50% of gas used D) No cap
Answer: B – EIP-3529 caps refunds at 20% of gas used to prevent exploitation.

Q14. Which transaction type typically uses the most gas?
A) Simple ETH transfer B) ERC-20 transfer C) Uniswap swap D) Contract deployment
Answer: D – Deploying contracts costs 200k-500k+ gas, far exceeding other operations.

Q15. What is the advantage of using event logs instead of storage?
A) Events are faster B) Events cost much less gas (375 vs 20,000)
C) Events are on-chain accessible D) No advantage
Answer: B – Events cost 375 gas + 8/byte vs 20,000+ for storage.

Quiz Questions (4/4)

Q16. What does EIP-4844 introduce to Ethereum?
A) Higher block gas limit B) Blob gas market for L2 data
C) Free transactions D) Proof-of-Work return
Answer: B – EIP-4844 (Dencun, March 2024) adds blob gas market for L2 data.

Q17. How many blobs does EIP-4844 target per block?
A) 1 blob B) 3 blobs C) 6 blobs D) 12 blobs
Answer: B – Target is 3 blobs/block (384 KB), max 6 blobs (768 KB).

Q18. What optimization technique reduces gas by packing multiple values in one storage slot?
A) Memory arrays B) Packed storage C) Short-circuiting D) Event logging
Answer: B – Packing multiple small values into one uint256 slot saves SSTORE costs.

Q19. In a require statement with AND logic, which condition should come first?
A) Most expensive B) Cheapest (to short-circuit and save gas)
C) Random order D) Order doesn't matter
Answer: B – Place cheap checks first; if they fail, expensive operations are skipped.

Q20. If base fee is 30 Gwei, priority fee is 2 Gwei, and max fee is 50 Gwei, what is the effective gas price?
A) 30 Gwei B) 32 Gwei C) 50 Gwei D) 2 Gwei
Answer: B – Effective gas price = min(base + priority, max) = min(30+2, 50) = 32 Gwei.