

## L14: Gas Mechanics

### Module B: Ethereum & Smart Contracts

Blockchain & Cryptocurrency Course

December 2025

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
- Understand 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

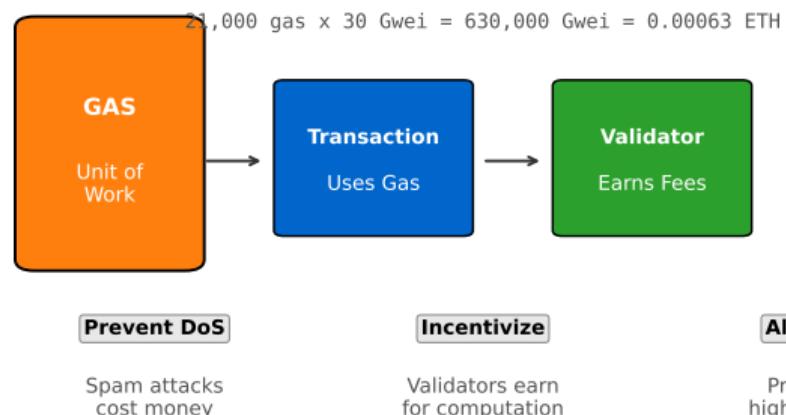
# What is Gas?

**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

## Gas: Unit of Computational Effort

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



## 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 ( $1 \text{ Gwei} = 10^9 \text{ Wei}$ )
- **Ether:** User-facing unit ( $1 \text{ ETH} = 10^{18} \text{ Wei}$ )

## 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:  $21,000 \times 50 = 1,050,000 \text{ Gwei} = 0.00105 \text{ ETH}$

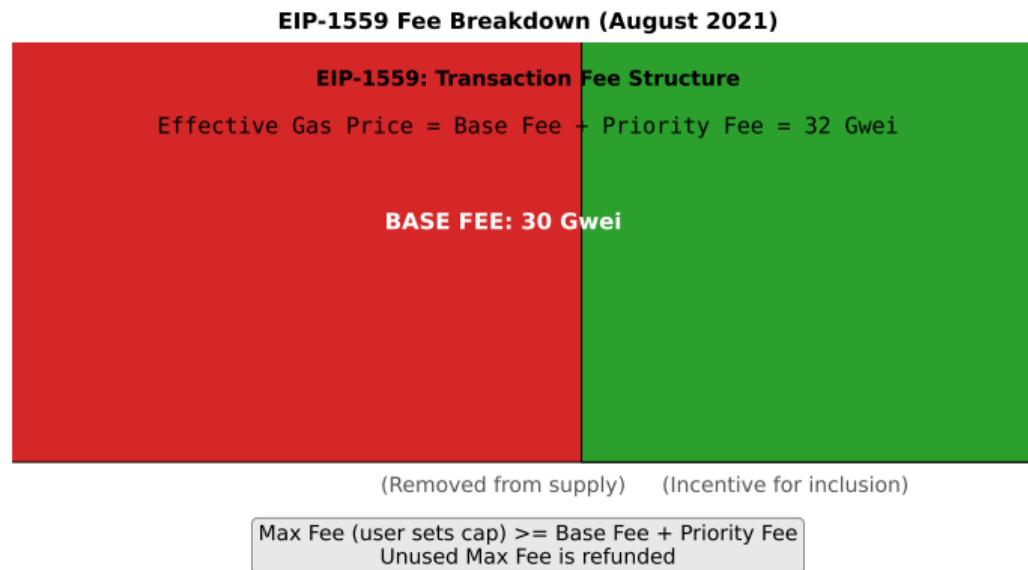
Challenges:

- 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



## New Formula:

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

## With cap:

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

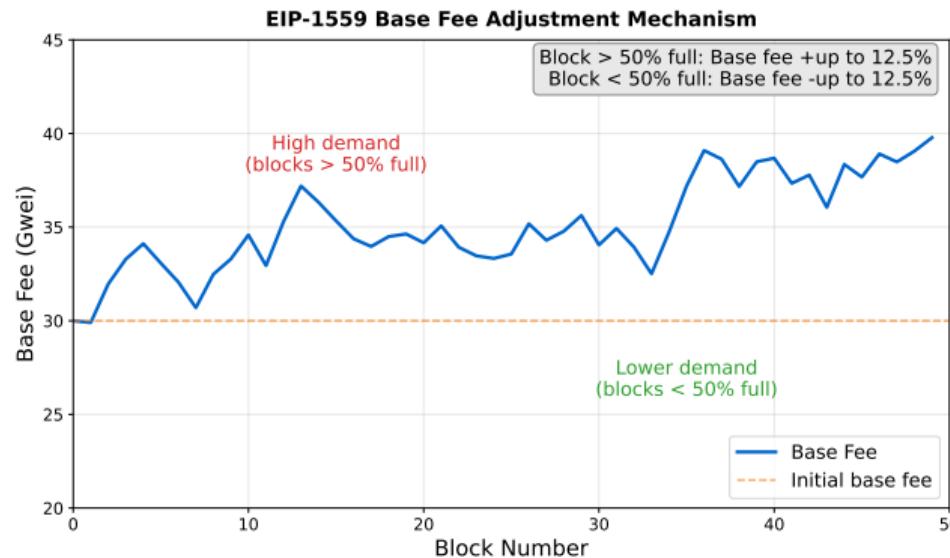
## Example:

- 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

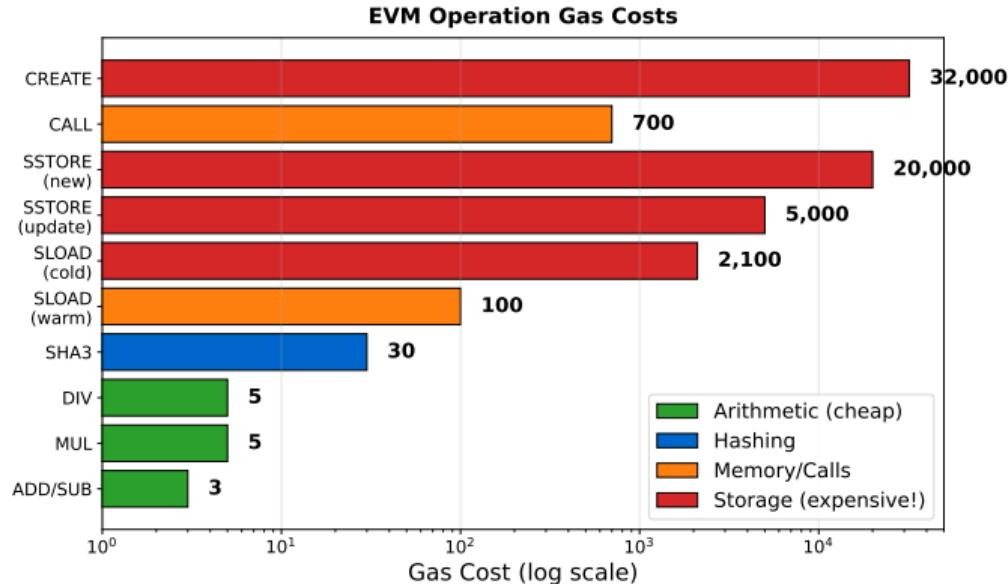
### Gas Used:

- 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:  $0.00064 \text{ ETH} = \$1.28$  at  $\$2000/\text{ETH}$

# 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)
        }
    }
}
```

## 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

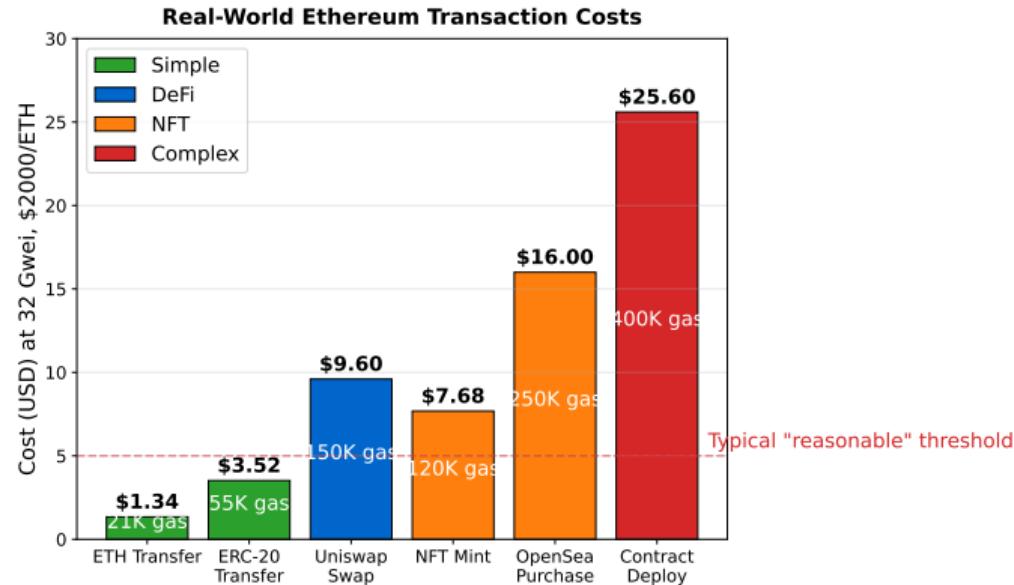
**Events:** Not accessible on-chain, 375 gas + 8/byte

```
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)

## 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)
}
```

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)

## Cost comparison for posting L2 data to Ethereum:

\$122.88

\$96.00

## 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?

## 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](https://remix.ethereum.org))
- Review basic programming concepts (if-else, loops, functions)