ERC-20 Token Creation: A Quantitative Deep Dive

Standalone Technical Lecture

Prof. Dr. Joerg Osterrieder

University Lecture Series

March 5, 2026

Lecture Roadmap

Learning Objectives

- Understand the ERC-20 standard and its role
- Implement all required functions and events
- Deploy and verify a custom ERC-20 token
- Analyze token economic models quantitatively
- Explore advanced patterns: minting, burning, voting

Prerequisites

- Lessons 1–3: Blockchain, Cryptography, Ethereum
- Basic Solidity syntax (variables, functions, mappings)
- Familiarity with Remix IDE or Hardhat
- Understanding of gas and transaction fees

Durat

90 minutes — 5 sections — ˜55 frames — Prerequisite: Lessons 1–3

Table of Contents

Navig

through 5 sections covering token standards to deployment

Learning Objectives

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

1. **Explain** the ERC-20 token standard interface and its role in the Ethereum ecosystem
2. **Implement** a custom ERC-20 token with mint, burn, and access control features
3. **Analyze** the approve/transferFrom delegation pattern and its security implications
4. **Compare** fixed, inflationary, and deflationary token supply models
5. **Design** a token distribution strategy with vesting schedules

Bloom

taxonomy levels: Remember → Understand → Apply → Analyze → Evaluate → Create

Section 1: Token Standards

What tokens are, why standards matter, and the ERC-20 ecosystem

What is a Token?

Coins vs. Tokens

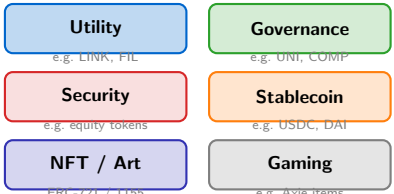
Native Coins:

- Have their own blockchain (BTC, ETH, SOL)
- Maintained by network validators/miners
- Required for gas / transaction fees
- Cannot be created by anyone

Tokens:

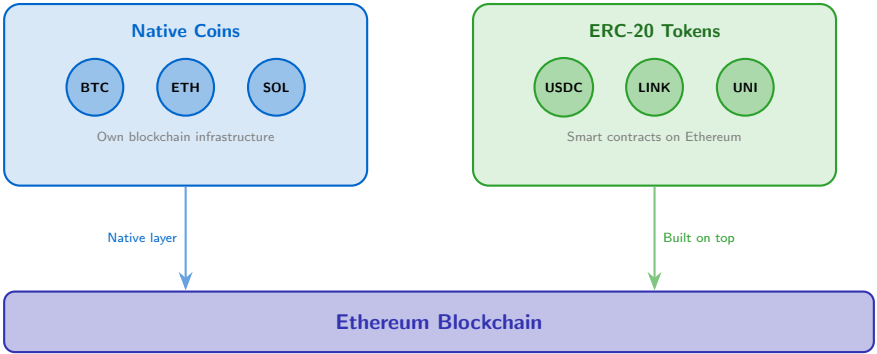
- Live on an existing blockchain via smart contracts
- Anyone can create in minutes
- Inherit chain security and infrastructure
- Represent virtually any asset or right

Token Use Cases:



Over 500,000 tokens exist on Ethereum alone, demonstrating the power of token standards

Tokens vs. Coins: The Architecture



Coins have their own blockchain; tokens are smart contracts on an existing chain

Why Standards Matter

Without Standards

- × Each token has a custom interface
- × Wallets must update code for every new token
- × No interoperability between protocols
- × DEXs cannot swap arbitrary tokens
- × Developers duplicate security audits repeatedly
- × Massive coordination overhead

With Standards (ERC-20)

- ✓ Universal interface for all tokens
- ✓ Any wallet works instantly with new tokens
- ✓ Instant composability in DeFi
- ✓ DEXs swap any pair automatically
- ✓ Shared security through OpenZeppelin
- ✓ Network effects multiply value

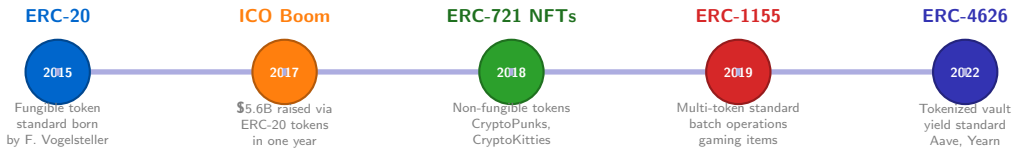
Analogy: Gift Cards

Imagine if every shop had a different gift card shape, size, and reader. ERC-20 is like a universal gift card standard: **one reader, infinite issuers.** Uniswap, Aave, MetaMask — all read the same interface regardless of which team deployed the token.

ERC

= Ethereum Request for Comments — ERC-20 proposed 2015 by Fabian Vogelsteller

Token Standardization Timeline



Token standards evolved to meet increasingly complex DeFi and NFT use cases

ERC-20: The Fungible Token Standard

Definition

A **fungible token** standard means every unit is *identical* and interchangeable. 1 USDC always equals 1 USDC regardless of which address holds it — unlike an NFT where each token is unique.

Fungibility Principle

- **Identical:** token#1 = token#2 in value
- **Divisible:** up to 18 decimal places (10^{18} units)
- **Interchangeable:** swap any unit freely
- **Uniform:** same rights for all holders

Like currency: \$5 bill A = \$5 bill B, unlike CryptoPunk $\#1 \neq \#2$

Required Interface (EIP-20)

6 Functions:

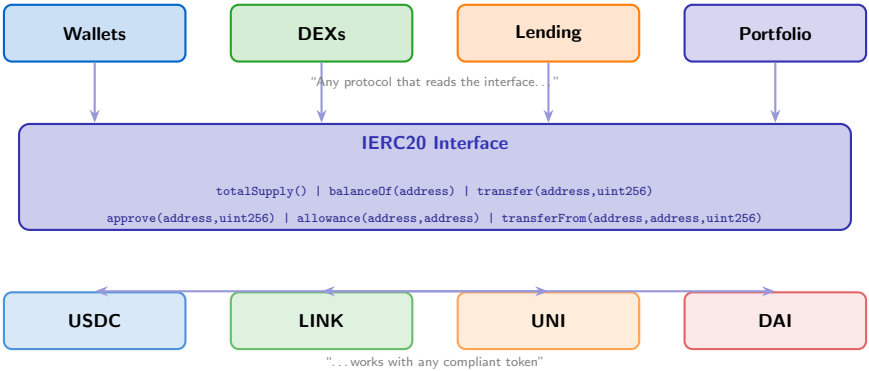
- `totalSupply()` – total tokens minted
- `balanceOf(addr)` – balance query
- `transfer(to, amt)` – direct send
- `approve(spender, amt)` – grant allowance
- `allowance(owner, spender)` – query limit
- `transferFrom(from, to, amt)` – delegated

2 Events:

- `Transfer(from, to, value)`
- `Approval(owner, spender, value)`

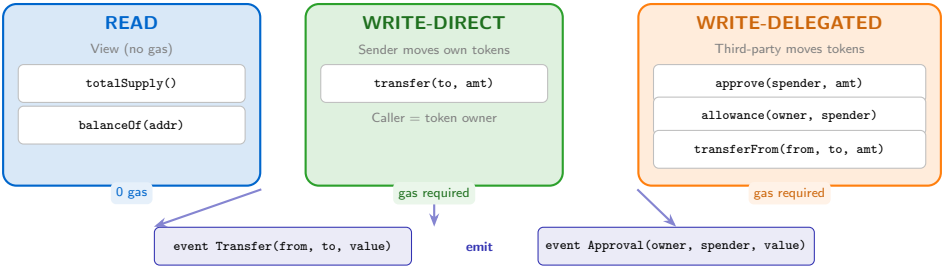
Any contract implementing these 6 functions and 2 events is ERC-20 compliant

ERC-20 Interface Architecture



The interface pattern enables any new wallet to instantly support any new token

ERC-20 Function Categories



Read functions cost no gas; write functions require a transaction and gas payment

Other Token Standards on Ethereum

Standard	Type	Key Feature	Year	Example
ERC-20	Fungible Token	Universal balance interface	2015	USDC, LINK, UNI, DAI
ERC-721	Non-Fungible	Unique ownership per tokenId	2018	CryptoPunks, BAYC
ERC-1155	Multi-Token	Batch transfer, fungible+NFT	2019	Gaming items, Enjin
ERC-4626	Tokenized Vault	Standardized yield accounting	2022	Aave aTokens, Yearn
ERC-2612	Permit Extension	Gasless approvals via signature	2020	USDC v2, DAI

ERC-20 Remains Dominant

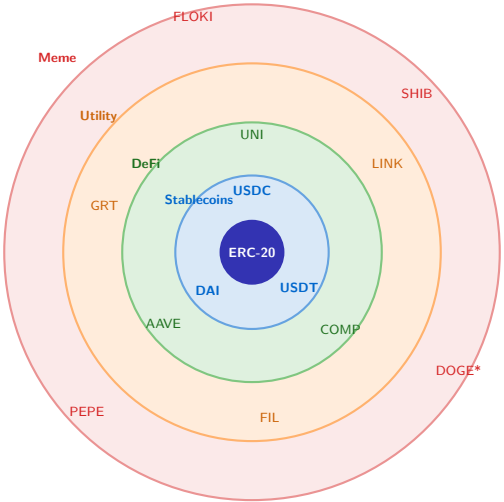
Despite newer standards, ERC-20 holds >90% of DeFi TVL. Most newer standards (ERC-4626, ERC-2612) *extend* ERC-20 rather than replace it.

Standard Hierarchy

ERC-721 and ERC-1155 are *not* backward compatible with ERC-20. A wallet that reads ERC-20 cannot automatically display NFT holdings — separate interface detection required.

This lecture focuses on ERC-20; other standards build on similar patterns

Token Ecosystem Market Map



*DOGE is a coin, shown for reference

The token ecosystem spans from serious financial infrastructure to community-driven experiments

Section 1 Summary: Key Numbers

500K+

ERC-20 Tokens

deployed on Ethereum mainnet

6

Required Functions

to be ERC-20 compliant

2

Required Events

Transfer and Approval

2015

Year Proposed

EIP-20 by Fabian Vogelsteller

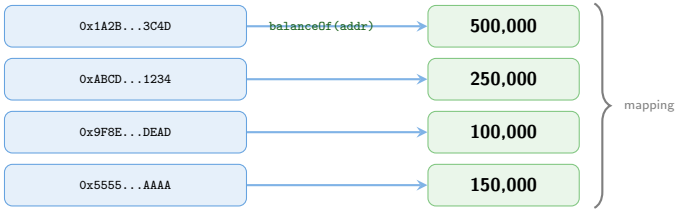
Token standards are the foundation of the entire DeFi ecosystem

Section 2: ERC-20 Functions Deep Dive

Core transfer, approval, and allowance functions

totalSupply() and balanceOf()

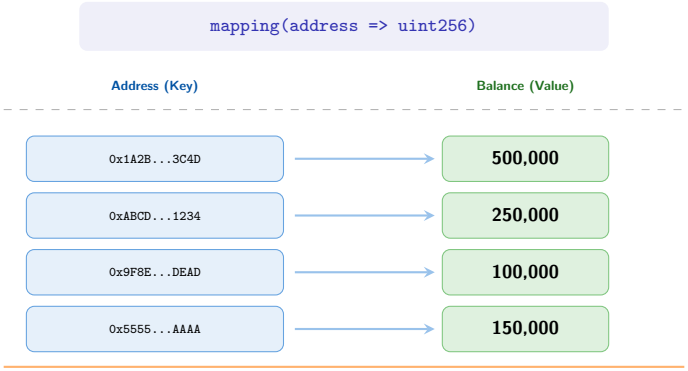
balanceOf(addr) lookup



These

are view functions – they read blockchain state without modifying it

The Balance Mapping

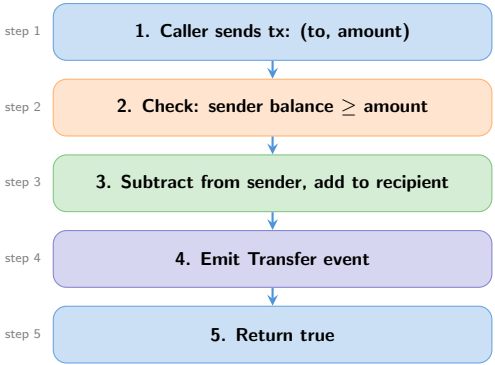


"The entire token ledger is just one mapping"

Unlik

bank databases, token balances are stored publicly on-chain in a simple mapping

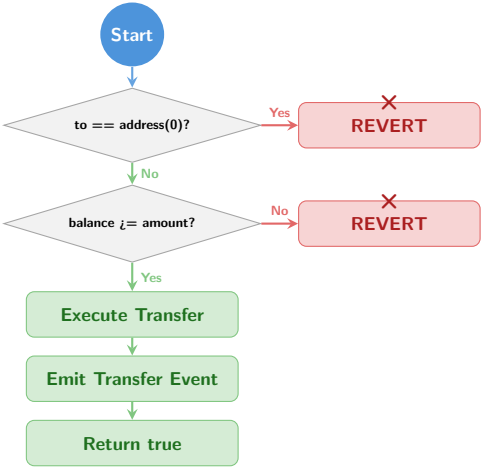
transfer() – Direct Transfer



transf

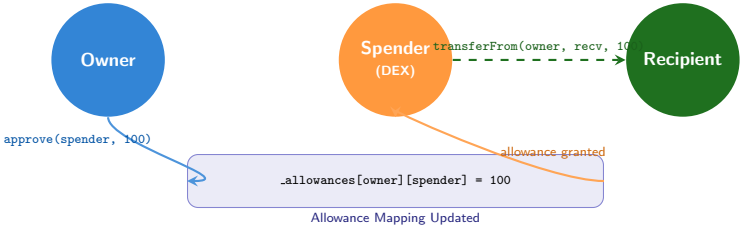
is the simplest token movement – you send your own tokens directly

transfer() – Safety Checks



Every token function must validate inputs before modifying state – checks-effects-interactions

approve() and allowance()



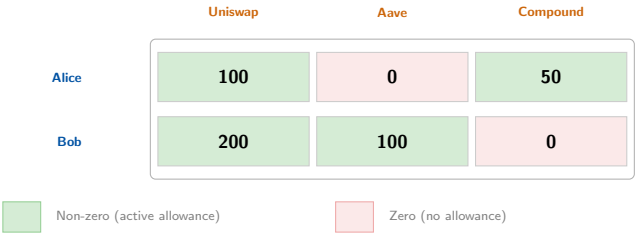
Like giving a debit card with a spending limit

Appr

are essential for DeFi – they let smart contracts move tokens on your behalf

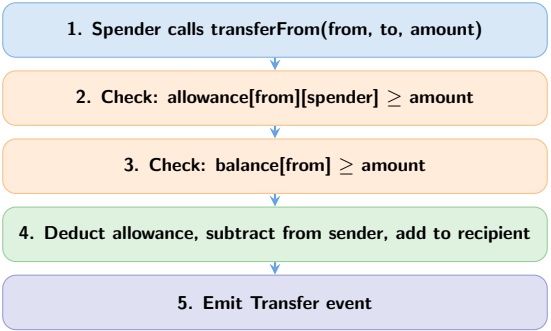
The Allowance Mapping

```
mapping(address => mapping(address => uint256))
```



The double mapping allows each owner to set independent allowances for multiple spenders

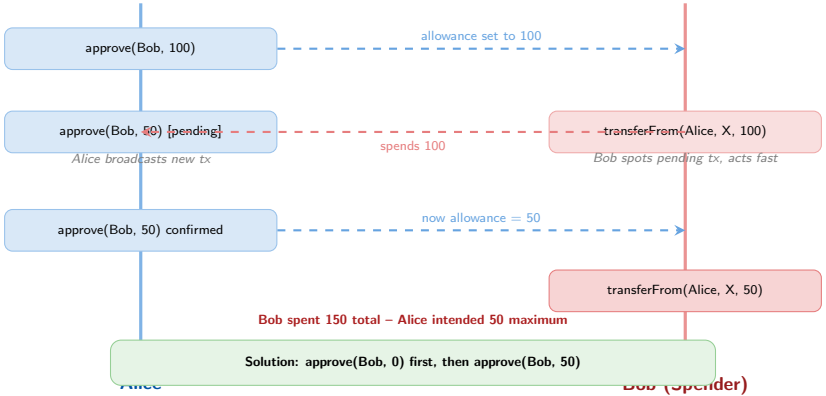
transferFrom() – Delegated Transfer



transf

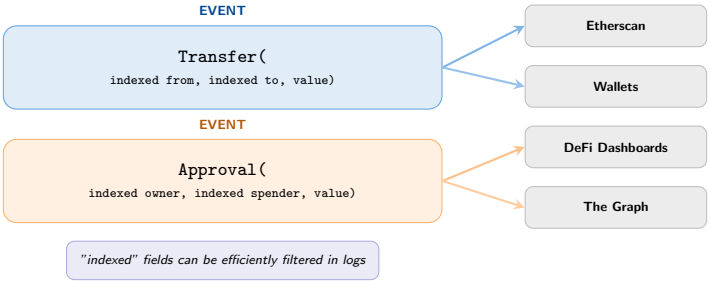
is the workhorse of DeFi – every swap, lend, and stake uses it

Approval Race Condition



The approval race condition is a known ERC-20 issue – always reset to 0 before changing

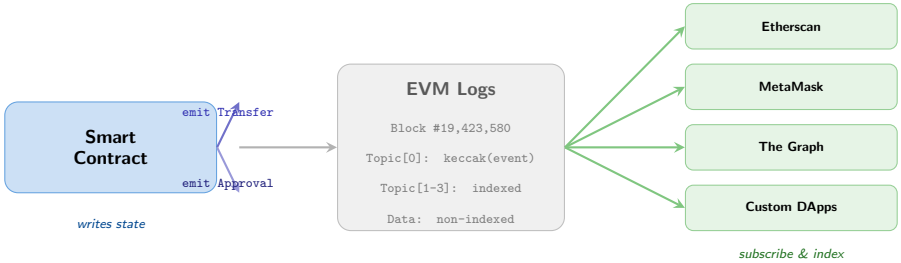
Events – Transfer and Approval



Event

are stored in transaction logs – cheap storage but cannot be read by other contracts

Events Power the Ecosystem



Witho

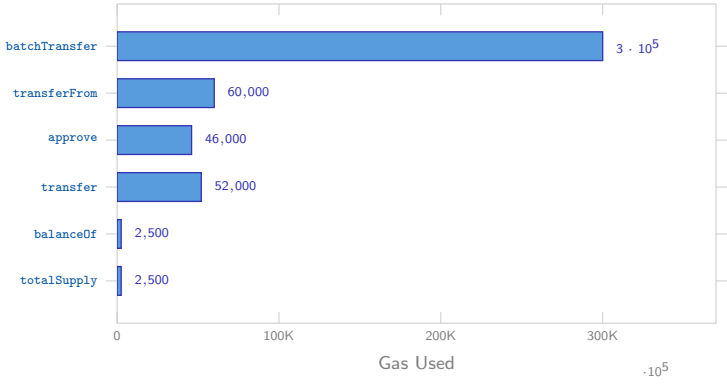
events, there would be no way to efficiently track token transfers off-chain

Complete ERC-20 Skeleton

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract BasicERC20 {
    string public name = "BasicToken";
    string public symbol = "BTK";
    uint8 public decimals = 18;
    uint256 public totalSupply;

    mapping(address => uint256) private _balances;
    mapping(address => mapping(address => uint256)) private _allowances;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    constructor(uint256 initialSupply) {
        totalSupply = initialSupply * 10**decimals;
        _balances[msg.sender] = totalSupply;
        emit Transfer(address(0), msg.sender, totalSupply);
    }

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    // transfer(), approve(), allowance(), transferFrom() ...
}
```

This skeleton shows the minimal structure – production code should use OpenZeppelin

Function Gas Costs



View

functions are free when called externally; write functions cost real ETH in gas

Section 3: Build MyToken

Hands-on: Creating a complete ERC-20 token

MyToken Specification

Token Parameters

Property	Value
Name	MyToken
Symbol	MTK
Decimals	18
Max Supply	1,000,000 MTK
Initial Supply	100,000 MTK

Features

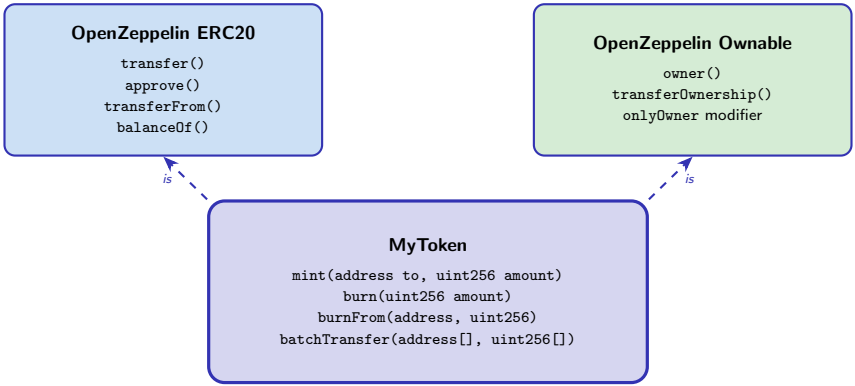
- **mint** — owner-only, capped at MAX_SUPPLY
- **burn** — any holder can burn own tokens
- **burnFrom** — burn with allowance mechanism
- **batchTransfer** — multiple recipients in one tx

Inherits

- OpenZeppelin `ERC20` — full standard implementation
- OpenZeppelin `Ownable` — access control

This design balances simplicity with useful features for learning and experimentation

Design Architecture



Multi

inheritance lets MyToken combine standard token functionality with access control

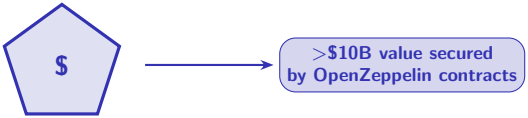
OpenZeppelin – Why Not Reinvent?

From Scratch

- 1000+ lines of boilerplate
- Zero security audits
- Error-prone edge cases
- Reinventing known bugs
- Months of hardening needed

With OpenZeppelin

- ~20 lines of custom logic
- Battle-tested since 2016
- Formally audited code
- Community-maintained
- Immediate production quality



Open

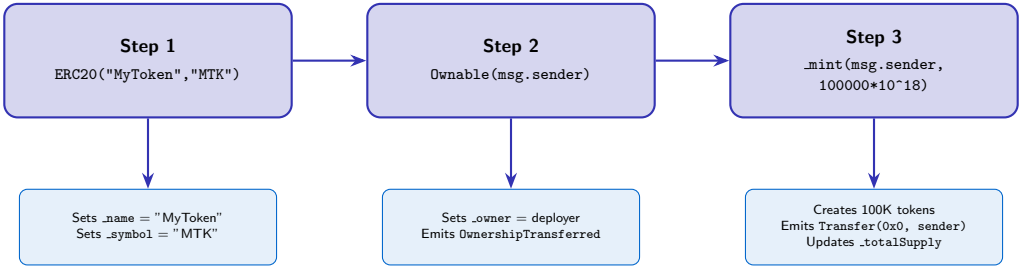
contracts secure over $10 billion in value – never reinvent security-critical code

MyToken.sol – The Complete Contract

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract MyToken is ERC20, Ownable {
    uint256 public constant MAX_SUPPLY = 1_000_000 * 10**18;

    constructor() ERC20("MyToken", "MTK") Ownable(msg.sender) {
        _mint(msg.sender, 100_000 * 10**18);
    }

    function mint(address to, uint256 amount) public onlyOwner {
        require(totalSupply() + amount <= MAX_SUPPLY, "Exceeds max");
        _mint(to, amount);
    }

    function burn(uint256 amount) public {
        _burn(msg.sender, amount);
    }

    function batchTransfer(address[] calldata to, uint256[] calldata amt)
        external
    {
        require(to.length == amt.length, "Length mismatch");
        for (uint i = 0; i < to.length; i++) {
            _transfer(msg.sender, to[i], amt[i]);
        }
    }
}
```

Full contract: ˜30 lines of custom code plus all inherited ERC-20 and Ownable functionality

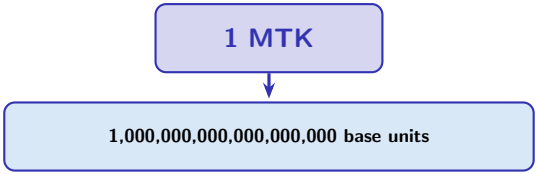
Constructor Deep Dive



Constructor runs exactly once at deployment – sequence matters

The constructor runs exactly once during deployment – it initializes all inherited contracts

The 10^{18} Decimals



Human ↔ On-chain

```
amount * 10^18
amount / 10^18
```

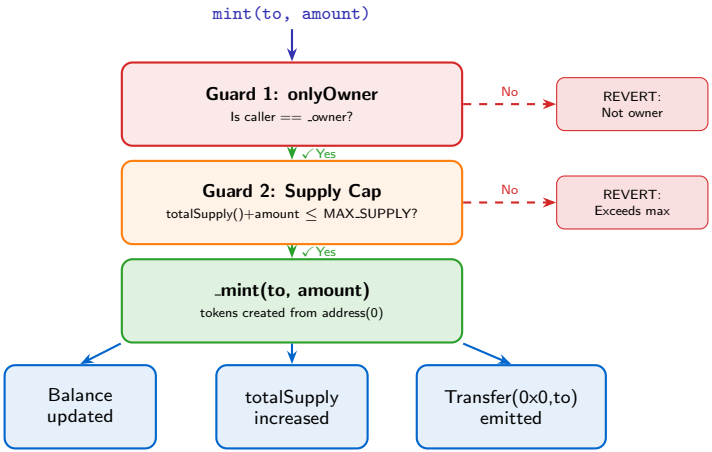
Decimal Comparison

Currency	Fraction Unit	Factor
ETH	Wei	10^{18}
USD	Cents	10^{2}
MTK	Base unit	10^{18}

18

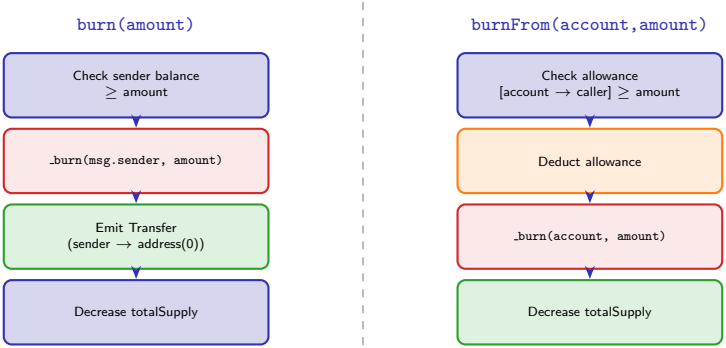
decimals allow fractional token amounts down to 0.000000000000000001 MTK

Mint Function



The MAX_SUPPLY cap prevents unlimited inflation – a key tokenomics design decision

Burn Function



Burni

reduces total supply permanently – creating scarcity and potential value appreciation

Batch Transfer – Gas Efficiency

Individual Transfers (N=10)

tx 1: transfer → recipient 1	˜52K gas
tx 2: transfer → recipient 2	˜52K gas
tx 3: transfer → recipient 3	˜52K gas
tx 4: transfer → recipient 4	˜52K gas
tx 5: transfer → recipient 5	˜52K gas

. . . 5 more transactions

Total: 10 × 52K = 520K gas

Batch Transfer (N=10)

batchTransfer()

_transfer → recipient 1	˜25K
_transfer → recipient 2	˜25K
_transfer → recipient 3	˜25K
_transfer → recipient 4	˜25K

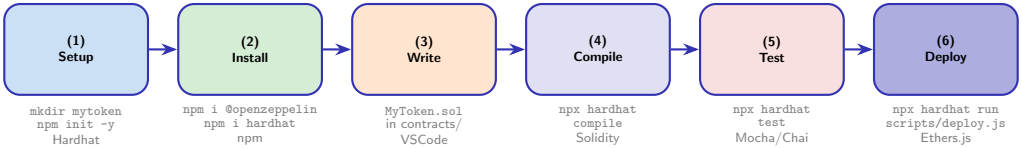
. . . 6 more

Total: 52K + 10×25K = 302K gas

Savings: 520K − 302K = 218K gas ≈ **42% reduction**

Batch transfers save gas for airdrops and payroll – one transaction instead of many

Development Workflow



Project Structure

`mytoken/` → `contracts/MyToken.sol` — `scripts/deploy.js` — `test/MyToken.test.js` — `hardhat.config.js`

Hardh

is the most popular Ethereum development environment – alternatives include Foundry and Truffle

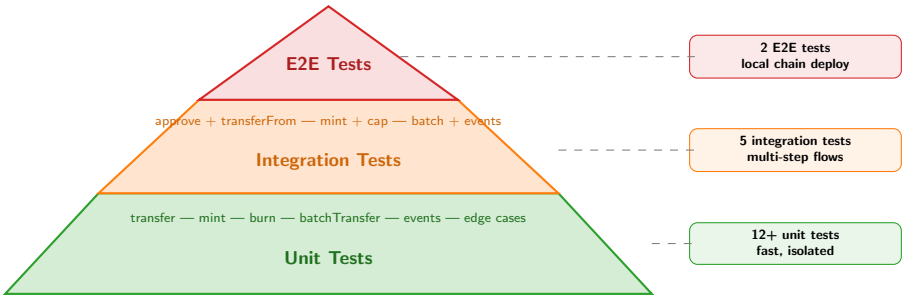
Deploy Script

```
const hre = require("hardhat");

async function main() {
    const [deployer] = await hre.ethers.getSigners();
    console.log("Deploying with:", deployer.address);

    const MyToken = await hre.ethers.getContractFactory("MyToken");
    const token = await MyToken.deploy();
    await token.waitForDeployment();

    console.log("MyToken deployed to:", await token.getAddress());
}

main().catch((error) => {
    console.error(error);
    process.exitCode = 1;
});
```

Run:

npx hardhat run scripts/deploy.js –network localhost

Testing Strategy



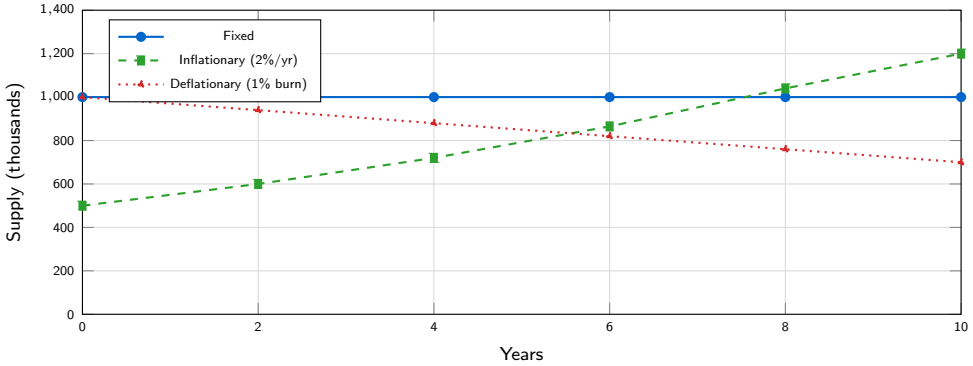
Alway

test on local network first, then testnet, and only then consider mainnet

Section 4: Token Economics

Designing sustainable token models

Supply Models – The Big Three

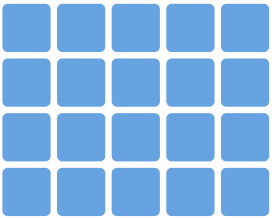


Toke

economics (tokenomics) significantly impacts long-term value and utility

Fixed Supply – Digital Scarcity

Total: 1 000 000 MTK



No new tokens can be created

Mint function: DISABLED

✓ Pros

- Predictable scarcity
- Deflationary price pressure
- Simple to audit

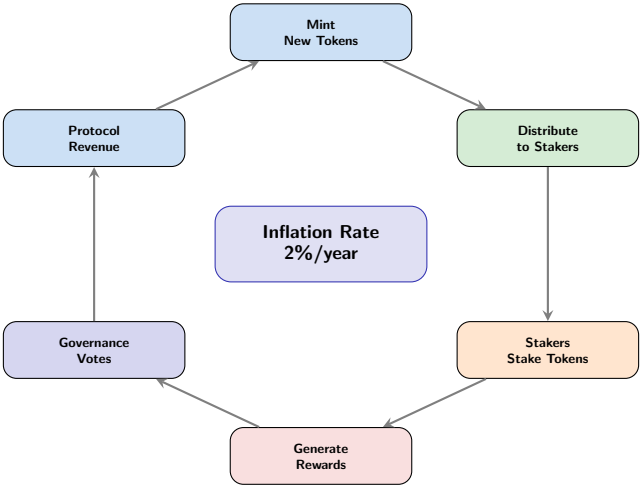
✗ Cons

- Cannot reward participants
- No growth adaptation
- Incentivizes hoarding

Examples: BTC (21M cap), similar model for MTK

Fixed supply creates predictable scarcity – but cannot adapt to ecosystem growth

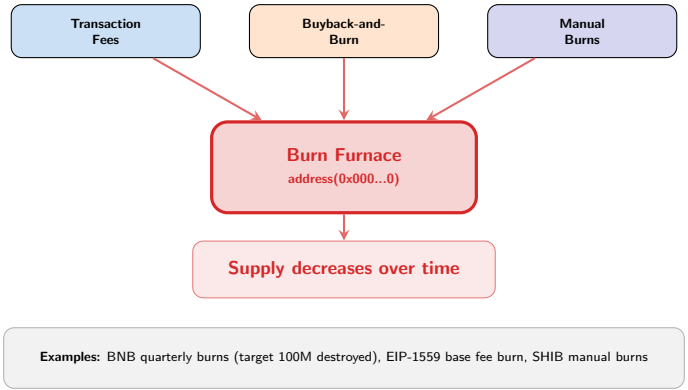
Inflationary Models



Examples: ETH post-merge (≈0.5%), UNI (2% annual), COMP (4.5% annual)

Inflat

Deflationary Models – Burn Mechanics



Defla

pressure increases scarcity – but excessive burning can reduce liquidity

Distribution Strategies

Typical Allocation Ranges

Category	Typical Range
Team & Founders	15–20%
Investors / VCs	10–20%
Public Sale	10–30%
Community/Airdrops	10–20%
Ecosystem Fund	20–30%
Liquidity	5–15%

✓ Best Practices

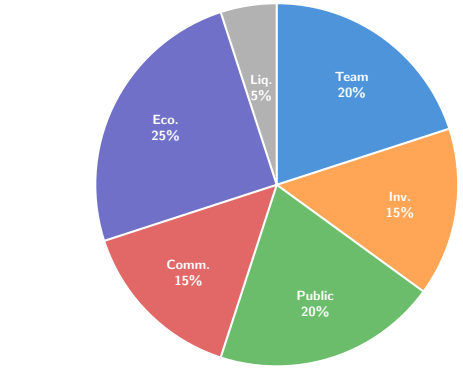
- ✓ Vesting schedules for insiders
- ✓ Full on-chain transparency
- ✓ Community-first allocation
- ✓ Gradual release schedules

× Red Flags

- × >50% allocated to team
- × No vesting for founders
- × Hidden allocations
- × Instant unlock at launch

tokenomics build trust – always publish distribution details

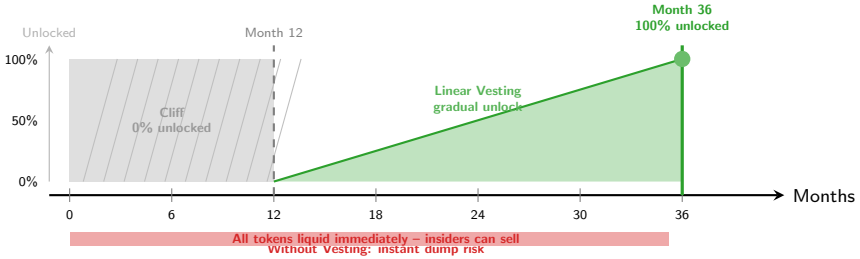
Distribution Visualization



Distri

determines initial decentralization and long-term governance power

Vesting Schedules



Vesti

protects investors by preventing large token dumps after launch

Real-World Token Economics

Token	Type	Supply Model	Max Supply	Primary Use
USDC	Stablecoin	Elastic (mint/redeem)	Unlimited	Payments
LINK	Utility	Fixed	1B	Oracle payments
UNI	Governance	Inflationary (2%)	1B initial	Protocol governance
AAVE	DeFi	Deflationary (buyback)	16M	Lending governance
BNB	Exchange	Deflationary (burns)	200M	Fee discounts
MATIC	L2	Fixed	10B	Gas token

Study successful tokens to understand what makes sustainable token economics

Tokenomics in Code

Fixed Supply

```
// Fixed: mint all at deploy
constructor()
  ERC20("Fixed","FIX")
{
  _mint(msg.sender,
    1e6 * 1e18);
}
// No mint() exists
```

Inflationary

```
// Owner can mint more
function mint(
    address to,
    uint256 amt
) public onlyOwner {
  _mint(to, amt);
}
```

Deflationary

```
// 1% burn on transfer
function transfer(
    address to, uint amt
) public override
  returns (bool) {
  uint fee = amt / 100;
  _burn(msg.sender, fee);
  return super.transfer(
    to, amt - fee);
}
```

Suppl

model choice is encoded permanently in the smart contract – choose wisely

Token Valuation Frameworks

Equation of Exchange: $MV = PQ$

M = money supply (token supply) · V = velocity (how fast tokens circulate)
P = price level · Q = quantity of goods/services traded

Lower velocity $\Rightarrow$ higher token value per unit of economic activity

Network Value (Metcalfe's Law): $V \propto n^2$

Value scales with the *square* of the number of users/participants.
A network with $2\times$ users is roughly $4\times$ more valuable.

Adoption is the primary value driver in early stage tokens

Discounted Utility / Cash Flow: $\mathrm{NPV} = \sum_t \frac{U_t}{(1+r)^t}$

Sum of discounted future utility/cash flows the token provides.
Applicable to fee-bearing tokens (e.g., LINK payments, UNI fee switch).

Requires realistic forecasts of protocol revenue and growth

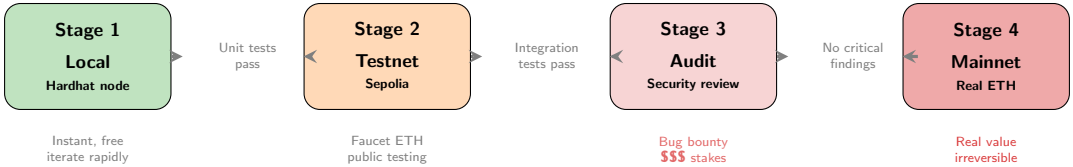
Token

valuation is an emerging field – no single framework captures all value drivers

Section 5: Advanced Topics & Summary

Next steps and key takeaways

Deployment Pipeline

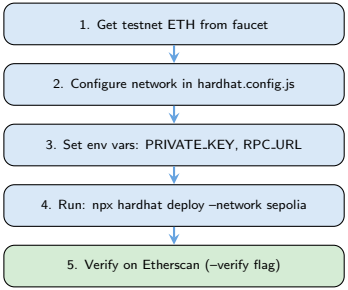


Each stage adds risk – never skip testing stages

Testnet Deployment

Available Testnets

Network	Faucet	Block Time
Sepolia	sepolia-faucet.com	12s
Goerli	goerlifaucet.com	12s
Mumbai	faucet.polygon.tech	2s



Testn

deployment is essential practice – treat it like production

Security Considerations

Common Vulnerabilities

- Reentrancy attacks
- Integer overflow/underflow
- Access control flaws
- Front-running attacks
- Approval race condition (ERC-20)
- Centralized admin keys

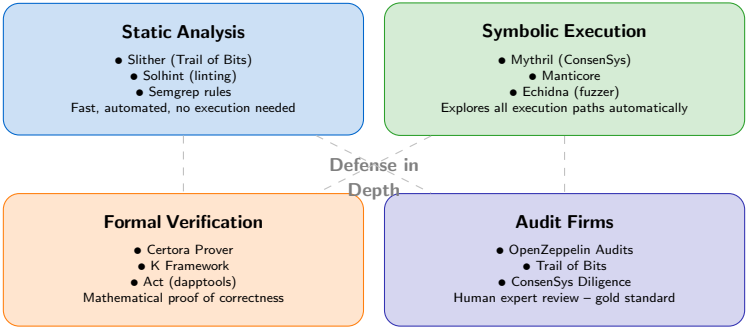
Best Practices

- ✓ Use OpenZeppelin battle-tested contracts
- ✓ Checks-Effects-Interactions pattern
- ✓ Add `ReentrancyGuard` modifier
- ✓ Implement `Pausable` pattern
- ✓ Multi-sig for admin ownership
- ✓ External audit before mainnet

Smar

contract bugs are permanent and can result in total loss of funds

Security Tools



Use multiple security tools – no single tool catches everything

Final Project Overview

Deliverables

- **Custom ERC-20 token** with chosen supply model and extensions
- **Comprehensive test suite** ($\geq$90% coverage, Hardhat)
- **Testnet deployment** with verified Etherscan contract
- **README documentation** with tokenomics rationale

Evaluation Criteria

Code quality (readability, comments, style)	**30%**
Test coverage (breadth, edge cases)	**25%**
Security (no known vulnerabilities)	**25%**
Creativity (unique features, tokenomics)	**20%**

This project integrates all 4 lessons: blockchain, cryptography, smart contracts, and tokens

Key Takeaways

S1: Tokens = programmable assets built on smart contract standards

S2: 6 functions + 2 events = the universal ERC-20 interface

S3: OpenZeppelin + Hardhat = professional development toolchain

S4: Tokenomics determines long-term success and sustainability

S5: Security first – always audit before mainnet deployment

You now have the foundation to build, deploy, and evaluate ERC-20 tokens

Course Completion

Course Overview

Lesson	Topic
L01	Blockchain Fundamentals
L02	Cryptography
L03	Smart Contracts
L04	Token Creation

You Can Now

- ✓ Write ERC-20 token contracts
- ✓ Deploy to testnets & mainnet
- ✓ Evaluate tokenomics models
- ✓ Assess smart contract security

Continue Learning

- → NFTs (ERC-721, ERC-1155)
- → DeFi protocols (Uniswap, Aave)
- → DAOs and governance
- → L2 scaling (Rollups, Polygon)
- → Security auditing

The blockchain space evolves rapidly – continuous learning is essential