Lesson 3.3: Smart Contracts and Programmable Finance

Module 3: The Trust Problem

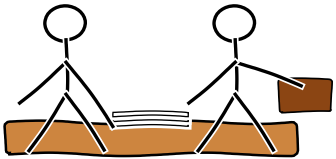
Prof. Dr. Joerg Osterrieder

Digital Finance — BSc Course

The Vending Machine That Never Cheats

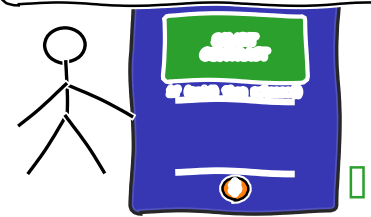
vs.

"My lawyer says your lawyer needs to talk to my other lawyer first."



Cost: $500/hour
Time: 6 weeks
Outcome: maybe

"Insert conditions. Code executes. No arguments. Ever."



Cost: $0.50 gas
Time: 12 seconds
Outcome: guaranteed

A smart contract is a vending machine for agreements: no counterparty risk, no excuses.

Learning Objectives

After completing this lesson, you will be able to:

1. **Explain** what a smart contract is and how the Ethereum Virtual Machine (EVM) executes code deterministically [Understand]
2. **Calculate** gas costs for simple transactions and explain why gas exists as a mechanism [Apply]
3. **Distinguish** between ERC-20 (fungible) and ERC-721 (non-fungible) token standards and their financial applications [Analyze]
4. **Analyze** how DAOs use governance tokens to coordinate decision-making without traditional corporate structures [Analyze]
5. **Evaluate** the trade-offs among Layer-2 scaling solutions (optimistic rollups, ZK-rollups, sidechains, state channels) [Evaluate]

Bloom's levels covered: Understand, Apply, Analyze, Evaluate

Objectives follow Bloom's taxonomy: Understand → Apply → Analyze → Evaluate.

From Consensus to Programmability

Lesson 3.2 asked: How do we agree on truth without a central authority?

Lesson 3.3 asks: Can we **program agreements** to execute automatically?

The leap from currency to computation:

- Bitcoin proved we can transfer *value* without intermediaries
- But what about *complex agreements*?
- Loans, insurance payouts, supply chain escrow, voting
- These require **conditional logic**: "If X happens, then do Y"

This lesson's journey: From the Ethereum architecture that makes programmable finance possible to the scaling challenges it must overcome.

We can agree on truth. Now: can we program agreements to execute automatically?

What Is a Smart Contract?

Definition: Smart Contract

A **smart contract** is a program stored on a blockchain that *automatically executes* when predetermined conditions are met. Once deployed, its code **cannot be changed**, and its execution is **verified by every node** in the network.

Analogy: A vending machine.

- You insert the right amount of money (input)
- The machine dispenses the selected item (output)
- No negotiation, no discretion, no counterparty risk
- The rules are **fixed in advance** and execute **exactly as programmed**

Key properties:

- **Deterministic:** Same inputs always produce the same outputs
- **Immutable:** Code cannot be altered after deployment (unless an upgrade proxy is built in — see next slide)
- **Deterministic execution, not "trustless":** The blockchain enforces *the code as written*, but the broader system depends on a stack of trust assumptions enumerated next slide
- **Transparent:** Anyone can inspect the source code on-chain (when verified — much bytecode is unverified)

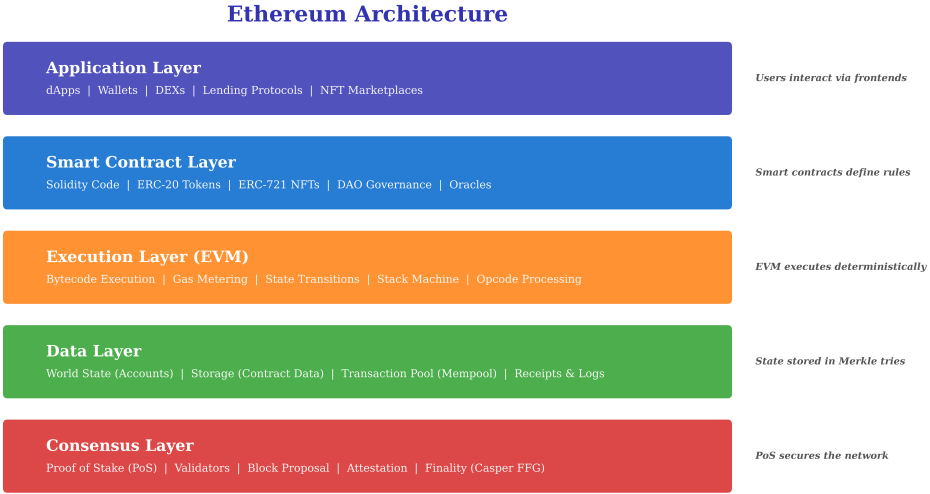
"Trustless" is marketing. "Deterministic execution subject to N trust assumptions" is engineering. Always count the N.

What "Trustless" Actually Means: The Trust Stack

A smart contract executes deterministically. The system around it depends on at least six trust layers — each a real attack surface.

1. **Oracles** — price/data feeds (Chainlink, Pyth). Manipulated oracles caused $400M+ in DeFi losses (*Rekt News oracle exploits ledger*, 2022–2024); e.g., Mango Markets ($117M, Oct 2022).
2. **Admin keys / multisigs** — a 2024 DeFiSafety review found **most major DeFi protocols retain admin upgrade keys** held by 2–7 signers; users trust those signers not to collude.
3. **Audit firms** — CertiK, Trail of Bits, OpenZeppelin produce reports, but audited contracts *still* get hacked (Ronin $625M, Wormhole $326M — both audited).
4. **Frontend hosting** — the smart contract may be on-chain, but the website you interact with runs on AWS/Cloudflare with centralized DNS; DNS hijacks (Curve, BadgerDAO) drained user wallets.
5. **L1 validators / consensus** — you trust that 51%+ of validators (or 33% in BFT chains) are honest. Lido alone controls ∼28% (*Dune Analytics: Lido Operators dashboard*, 2026) of staked ETH.
6. **Upgradeability proxies** — UUPS / Transparent / Beacon proxies let admins replace contract logic. A proxy *is* a backdoor (intentionally) — read the upgrade controls before depositing.

Bridge hacks alone: $2.5B+ stolen (Chainalysis Crypto Crime Report 2024, 2022–2024). "Trustless" systems still require trust — it's just relocated, not removed.


Ethereum Architecture
Application Layer
dApps | Wallets | DEXs | Lending Protocols | NFT Marketplaces
Users interact via frontends
Smart Contract Layer
Solidity Code | ERC-20 Tokens | ERC-721 NFTs | DAO Governance | Oracles
Smart contracts define rules
Execution Layer (EVM)
Bytecode Execution | Gas Metering | State Transitions | Stack Machine | Opcode Processing
EVM executes deterministically
Data Layer
World State (Accounts) | Storage (Contract Data) | Transaction Pool (Mempool) | Receipts & Logs
State stored in Merkle tries
Consensus Layer
Proof of Stake (PoS) | Validators | Block Proposal | Attestation | Finality (Casper FFG)
PoS secures the network

What Is the Ethereum Virtual Machine (EVM)?

Definition: Ethereum Virtual Machine (EVM)

The **EVM** is a sandboxed, Turing-complete virtual machine that executes smart contract bytecode on every Ethereum node. It provides a deterministic execution environment: given the same state and transaction, every node computes the **identical result**.

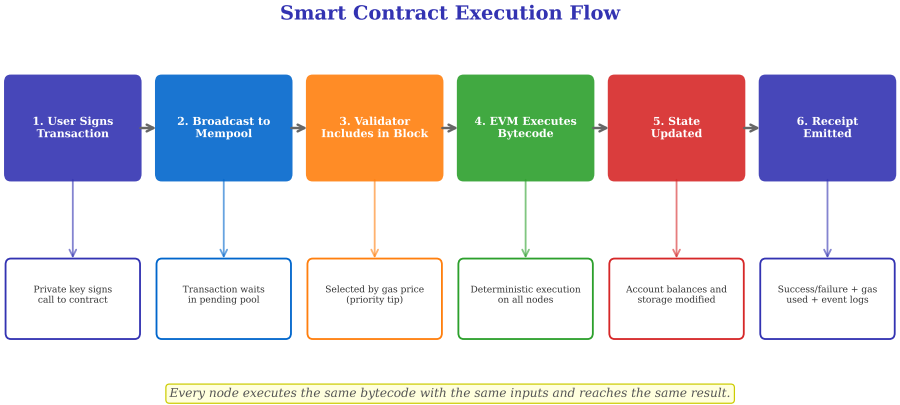
Key characteristics:

- **Stack-based:** Operations push/pop values on a 1024-deep stack
- **Turing-complete:** Can compute anything a standard computer can (with enough gas)
- **Sandboxed:** A contract cannot access the file system, network, or other contracts' storage directly
- **Metered:** Every operation costs **gas**, preventing infinite loops

Analogy: The EVM is like a global calculator that everyone can verify. You submit an equation (transaction), every node runs it, and they all agree on the answer.

The EVM is the engine that makes Ethereum a "world computer" — deterministic, verifiable, and unstoppable.

Smart Contract Execution Flow



- **What you see:** Six-step flowchart: user signs → broadcast → validator includes → EVM executes → state updated → receipt emitted
- **Key pattern:** Every node executes the same bytecode with the same inputs, producing identical state changes and gas consumption

What Is Solidity?

Definition: Solidity

Solidity is a high-level, statically-typed programming language designed for writing smart contracts on the EVM. Its syntax resembles JavaScript and it compiles to EVM bytecode.

Minimal example — a simple escrow:

```
contract Escrow {
address buyer; address seller;
uint256 amount;
function release() public {
require(msg.sender == buyer);
(bool ok, ) = seller.call{value: amount}("");
require(ok, "Transfer failed");
}
}
```

What this does:

- Holds ETH in escrow between a buyer and seller
- Only the buyer can trigger the release
- Once called, funds transfer automatically — no bank, no lawyer
- *Best practice:* `.call{value:}` is preferred over the legacy `.transfer()` because EIP-1884 (Istanbul) raised `SLOAD` gas above the 2300-gas stipend assumed by `transfer`/`send`. Pair with a reentrancy guard.

Solidity lets developers write financial logic that executes on a global, tamper-proof computer.

Definition: Gas

Gas is the unit of computational effort required to execute operations on Ethereum. Each EVM instruction has a fixed gas cost. Users pay gas fees (in ETH) to compensate validators for processing their transactions.

Why gas exists:

- **Prevent abuse:** Without gas, anyone could deploy infinite loops that halt the network
- **Allocate resources:** Scarce block space is allocated to whoever values it most (auction mechanism)
- **Compensate validators:** Validators earn gas fees for including transactions in blocks

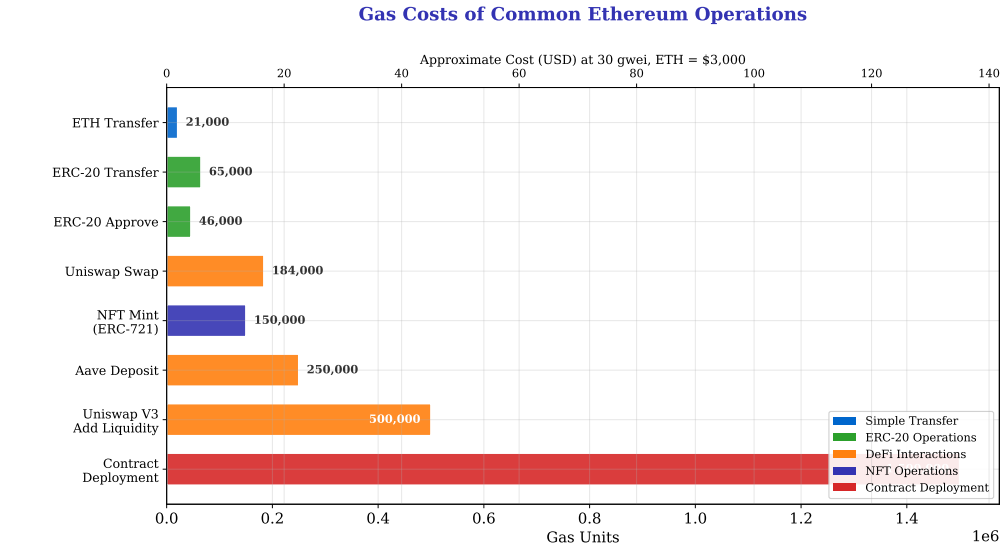
Analogy: Gas is like postage for computation. A simple letter (ETH transfer) costs little postage. A heavy package (complex smart contract call) costs more.

$$\textbf{Transaction fee} = \text{Gas used} \times \text{Gas price (in gwei)}$$

(1 gwei $= 10^{-9}$ ETH $\approx$ a few thousandths of a cent)

Gas is Ethereum's anti-spam mechanism: computation costs money, so attackers must pay to waste resources.

Gas Cost Mechanics



EIP-1559: The Modern Fee Market

Before EIP-1559 (Aug 2021): Users blindly bid gas prices in a first-price auction. Overpaying was common.

After EIP-1559:

- **Base fee:** Set by the protocol, adjusts automatically based on network congestion. This portion is **burned** (destroyed), not paid to validators.
- **Priority tip:** An optional tip paid to validators to prioritize your transaction.
- **Max fee:** The maximum the user is willing to pay (base fee + tip). Any excess is refunded.

Component	Purpose
Base fee	Algorithmic pricing; burned to reduce ETH supply
Priority tip	Incentivize validators; user-controlled
Max fee cap	Protects users from unexpected price spikes

Key insight: Burning the base fee makes ETH potentially **deflationary** — when more gas is burned than new ETH is issued, total supply shrinks.

EIP-1559 replaced a chaotic auction with predictable pricing and introduced ETH burning.

What Are Token Standards?

Definition: Token Standard

A **token standard** is a set of rules (a smart contract interface) that defines how a token behaves — how it is transferred, who can spend it, and how balances are tracked. Standards ensure **interoperability**: any wallet or exchange that supports the standard can handle any token built on it.

Why standards matter:

- Without a standard, every token would need custom integration
- **ERC-20** defines fungible tokens (like currencies): every unit is identical
- **ERC-721** defines non-fungible tokens (NFTs): every unit is unique
- **ERC-1155** combines both: a single contract manages fungible and non-fungible tokens together

Analogy: Token standards are like USB ports. Any USB device works in any USB port because they follow the same interface specification.

Token standards are the "plug and play" of blockchain — they let any wallet, DEX, or dApp interact with any token.

ERC-20 vs. ERC-721 vs. ERC-1155

Token Standard Comparison

Property	ERC-20 (Fungible)	ERC-721 (Non-Fungible)	ERC-1155 (Multi-Token)
Fungibility	Fully fungible (1 token = 1 token)	Non-fungible (each token unique)	Both fungible and non-fungible
Token ID	No individual IDs (balance per address)	Unique tokenId for each token	Unique ID per type; balance per type
Divisibility	Divisible (18 decimals typical)	Indivisible (whole units only)	Configurable per token type
Batch Transfer	One token type per transaction	One NFT per transaction	Multiple types in one transaction
Gas Efficiency	Low gas (simple mapping)	Moderate gas (ownership tracking)	Lowest gas (batch operations)
Use Cases	Stablecoins, DeFi tokens, governance	Digital art, deeds, certificates, gaming	Gaming economies, mixed asset portfolios
Examples	USDC, UNI, AAVE, DAI, LINK	CryptoPunks, BAYC, ENS domains	Enjin, Gods Unchained items

- **What you see:** Table comparing three standards across fungibility, use cases, and key functions
- **Key pattern:** ERC-20 = all units identical (currencies), ERC-721 = each unit unique (NFTs), ERC-1155 = mix both

ERC-20: The Fungible Token Standard

ERC-20 (Ethereum Request for Comment 20) defines six mandatory functions:

Function	What It Does
`totalSupply()`	Returns the total number of tokens in existence
`balanceOf(address)`	Returns the token balance of a given address
`transfer(to, amount)`	Sends tokens from the caller to another address
`approve(spender, amount)`	Grants permission for another address to spend tokens on your behalf
`allowance(owner, spender)`	Returns how many tokens a spender is allowed to spend
`transferFrom(from, to, amount)`	Transfers tokens from one address to another (using an allowance)

Financial applications:

- **Stablecoins:** USDC, USDT, DAI are all ERC-20 tokens
- **Governance tokens:** UNI (Uniswap), AAVE are ERC-20
- **Utility tokens:** LINK (Chainlink) pays for oracle data feeds

Over 500,000 ERC-20 tokens exist on Ethereum — the standard that launched the token economy.

Definition: Non-Fungible Token (NFT)

An **NFT** is a token with a unique identifier (`tokenId`) that distinguishes it from every other token. Unlike ERC-20 tokens (where 1 USDC = 1 USDC), each ERC-721 token is **one-of-a-kind**.

Key difference from ERC-20:

- ERC-20: "You have 100 tokens" (no distinction between them)
- ERC-721: "You own token #7,422" (a specific, traceable asset)

Financial applications beyond art:

- **Real estate:** Tokenized property deeds with on-chain ownership history
- **Insurance:** Unique policy contracts as NFTs, tradable on secondary markets
- **Identity:** KYC credentials as soulbound (non-transferable) NFTs
- **Debt instruments:** Each bond as a unique token with embedded repayment terms

NFTs turn any unique asset — physical or digital — into a verifiable, tradable on-chain object.

Definition: Decentralized Autonomous Organization (DAO)

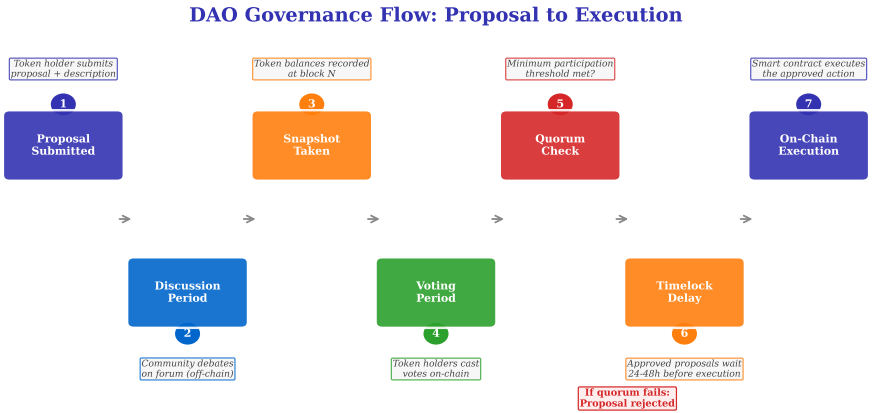
A **DAO** is an organization governed by rules encoded in smart contracts. Members vote on proposals using **governance tokens**, and approved proposals execute automatically. There is no CEO, no board of directors — only code and collective decision-making.

How a DAO differs from a corporation:

	Corporation	**DAO**
Leadership	CEO / Board	Token-holder votes
Rules	Legal contracts + bylaws	Smart contracts
Treasury	Bank account	On-chain multisig
Execution	Employees carry out orders	Code executes automatically
Transparency	Annual reports	Real-time, on-chain

A DAO replaces corporate hierarchy with transparent, code-enforced governance.

DAO Governance: From Proposal to Execution



- **What you see:** Flowchart showing proposal submission → voting period → quorum check → if passed, automatic execution
- **Key pattern:** No human intermediary — smart contract counts votes and executes approved changes automatically

Definition: Governance Token

A **governance token** is an ERC-20 token that grants its holder **voting rights** over a protocol's parameters, treasury, and upgrades. Holding more tokens means more voting power.

Examples:

- **UNI (Uniswap):** Vote on fee tiers, liquidity incentives, protocol upgrades
- **AAVE:** Vote on risk parameters, asset listings, reserve factors
- **MKR (MakerDAO):** Vote on collateral types and stability fees for DAI stablecoin

Risks and criticisms:

- **Plutocracy:** Whales with large holdings dominate votes
- **Voter apathy:** Many token holders never vote (participation often $<10\%$)
- **Flash loan attacks:** An attacker borrows millions of tokens, votes, and returns them — all in one transaction
- **Regulatory ambiguity:** Are governance tokens securities?

Governance tokens democratize protocol control but concentrate power in the hands of large holders.

Smart Contract Risks

Immutability is a double-edged sword: bugs are permanent.

Risk	Description	Notable Example
Reentrancy	Attacker re-enters a function before state updates	The DAO hack (2016): $60M stolen
Integer overflow	Arithmetic wraps around, minting tokens from nothing	Several early ERC-20 exploits
Oracle manipulation	Fake price data triggers incorrect contract logic	Mango Markets (2022): $114M
Governance attack	Flash-loan voting to pass malicious proposals	Beanstalk (2022): $182M
Upgrade backdoor	Admin retains hidden upgrade keys	Rug pulls on DeFi protocols

Mitigation:

- Professional security audits (Trail of Bits, OpenZeppelin)
- Formal verification (mathematical proof of correctness)
- Bug bounties, timelocks, and multisig admin controls
- **GenAI touchpoint:** LLM-based contract auditing tools (e.g., using GPT-4 to scan Solidity for common vulnerability patterns)

Code is law — but buggy code is buggy law. Audits and formal verification are essential.

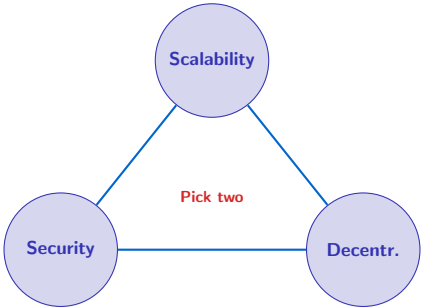
The Scalability Problem

Ethereum's bottleneck:

- Ethereum Layer-1 processes ∼15–30 transactions per second (TPS)
- Visa processes ∼65,000 TPS at peak
- During high demand, Ethereum gas fees can exceed $50–$100 per transaction

Why not just increase block size?

- Larger blocks → higher hardware requirements for validators
- Fewer people can afford to run nodes → less decentralization
- This is the **blockchain trilemma**: you can optimize at most two of three properties



Layer-2 Scalability Solutions Comparison

Property	Optimistic Rollup	ZK-Rollup	Sidechain	State Channel
Security model	Inherits L1 (fraud proofs)	Inherits L1 (validity proofs)	Own validators (independent)	Inherits L1 (on-chain dispute)
Throughput	~2,000 TPS	~2,000+ TPS	~7,000 TPS	~15,000 TPS (per channel)
Finality	7-day dispute window	~10 minutes (proof generation)	~2 seconds (own consensus)	Instant (off-chain)
Cost per tx	$\sim 0.10 - 0.50$	$\sim 0.05 - 0.25$	$\sim 0.001 - 0.01$	~$0 (on-chain only)
EVM compatible	Full compatibility	Partial (improving)	Full compatibility	Limited (specific use cases)
Data availability	On-chain (calldata)	On-chain (calldata)	Off-chain (own chain)	Off-chain (between parties)
Trust assumption	At least 1 honest verifier exists	Math (ZK proof) is correct	Majority of validators honest	Both parties are online
Examples	Arbitrum, Optimism, Base	zkSync Era, StarkNet, Scroll	Polygon PoS, Gnosis Chain	Lightning Network, Raiden

What Is a Rollup?

Definition: Rollup

A **rollup** executes transactions off-chain (on a separate chain), then posts a compressed summary of the results back to Ethereum Layer-1. The Layer-1 contract verifies that the rollup's computation is correct, inheriting Ethereum's security guarantees.

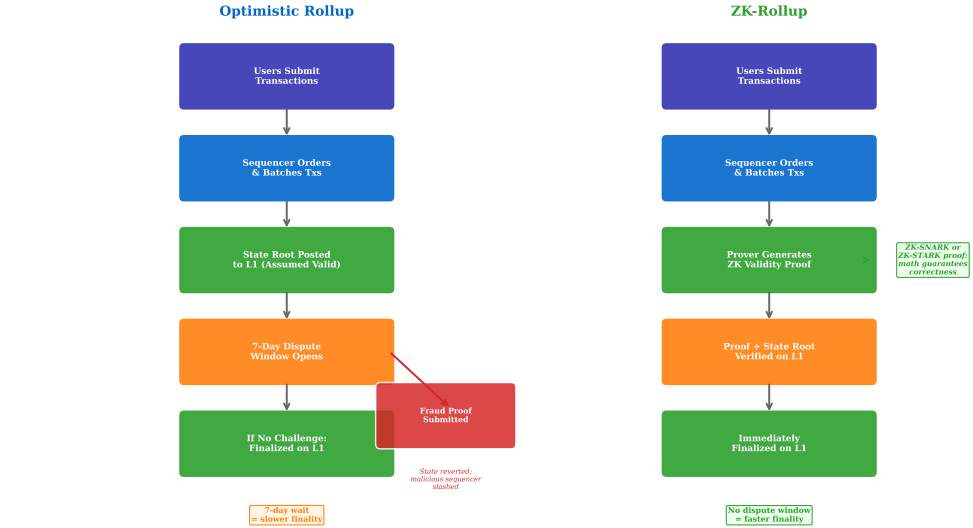
Two types:

- **Optimistic rollups:** Assume transactions are valid by default. A **fraud proof** can challenge incorrect results within a dispute window (typically 7 days). Examples: Arbitrum, Optimism.
- **ZK-rollups (Zero-Knowledge):** Generate a cryptographic **validity proof** (a Zero-Knowledge Succinct Non-Interactive Argument of Knowledge or ZK-SNARK, or a Zero-Knowledge Scalable Transparent Argument of Knowledge or ZK-STARK) that mathematically proves all transactions were executed correctly. No dispute window needed. Examples: zkSync, StarkNet.

Key trade-off: Optimistic rollups are simpler to build but have slower finality (7-day challenge period). ZK-rollups have instant finality but require complex cryptographic proof generation.

Rollups compress thousands of transactions into a single Layer-1 proof, cutting costs by 10–100×.

Rollup Architecture: Optimistic vs. ZK

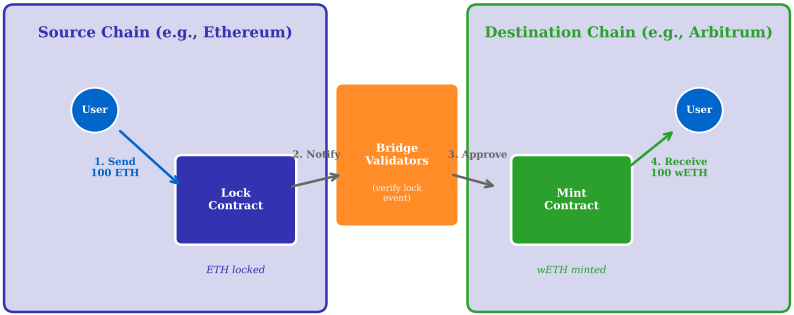


Through

Definition: Blockchain Bridge

A **bridge** is a protocol that enables the transfer of assets and data between two separate blockchains. Bridges typically work by **locking** assets on the source chain and **minting** equivalent wrapped tokens on the destination chain.

Cross-Chain Bridge: Lock and Mint

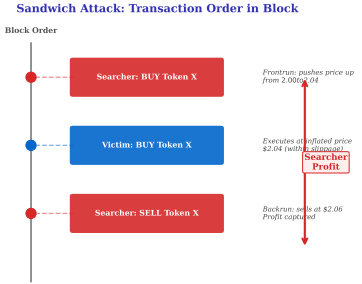


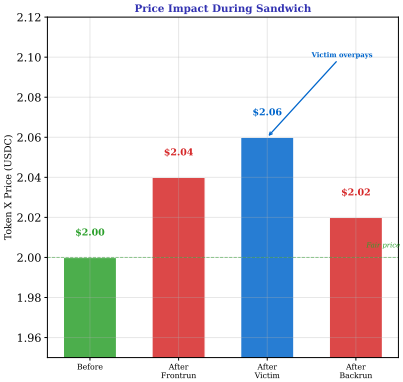
Risk: If bridge validators are compromised, locked assets can be stolen.

MEV: Maximal Extractable Value

Definition: Maximal Extractable Value (MEV)

MEV is the profit a block proposer (or searcher) can extract by reordering, inserting, or censoring transactions within a block. MEV arises because the proposer controls **transaction ordering**.





- **What you see:** Flowchart showing searcher detects opportunity → submits bundle → validator orders transactions → MEV

GenAI Touchpoint: LLMs and Smart Contracts

Generative AI is reshaping how smart contracts are written and audited:

Application	How It Works
Natural language → Solidity	Describe a contract in English ("Create an escrow that releases funds after 30 days"); an LLM generates the Solidity code
Automated auditing	LLMs scan contract bytecode for known vulnerability patterns (reentrancy, overflow, unchecked calls)
Documentation generation	LLMs produce human-readable explanations of existing contract logic
Test case generation	LLMs generate edge-case unit tests to improve contract coverage

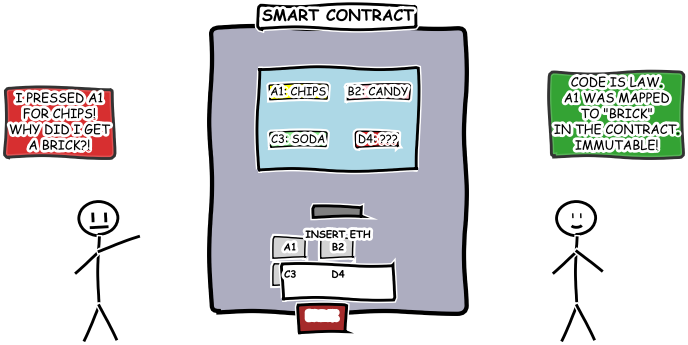
Limitations:

- LLMs can hallucinate plausible-looking but **incorrect** Solidity code
- LLMs cannot **replace** formal verification or professional audits
- A human expert must always review LLM-generated contract code before deployment

LLMs accelerate smart contract development but cannot replace the rigor of formal security audits.

One Last Thought...

Smart Contract Vending Machine



Smart contracts execute exactly as written. That's the feature AND the bug.

Sometimes the best way to remember a concept is to laugh about it.

Key Takeaways

1. **Smart contracts** are self-executing programs on a blockchain — deterministic and (usually) immutable, but operating on a stack of trust assumptions (oracles, admin keys, audits, frontends, validators, proxies)
2. The **EVM** is Ethereum's sandboxed execution engine; **Solidity** is its primary programming language
3. **Gas** meters computation to prevent abuse; **EIP-1559** introduced a base-fee-burn mechanism
4. **ERC-20** defines fungible tokens (currencies, stablecoins); **ERC-721** defines non-fungible tokens (unique assets)
5. **DAOs** replace corporate hierarchy with token-weighted on-chain voting, but face plutocracy and apathy risks
6. **Smart contract bugs are permanent** — security audits and formal verification are essential
7. **Layer-2 rollups** (optimistic and ZK) compress transactions off-chain to scale Ethereum by 10–100×
8. **Bridges** connect blockchains but are high-value attack targets ($2.5B+ stolen in 2022)
9. **MEV** is an invisible tax imposed by transaction ordering; it distorts markets and harms users

Smart contracts unlock programmable finance, but immutability means bugs and MEV require vigilant design.

Summary / Next Lesson Preview

This lesson: We explored how smart contracts turn blockchains from simple payment rails into programmable financial platforms, examined token standards, DAO governance, Layer-2 scaling, and the risks of MEV and bridge exploits.

Key vocabulary:

- Smart contract
- EVM / Solidity
- Gas / EIP-1559
- ERC-20 / ERC-721
- NFT
- DAO / Governance token
- Layer-2 / Rollup
- Optimistic rollup / ZK-rollup
- Sidechain / State channel
- Bridge
- MEV (Maximal Extractable Value)
- Reentrancy / Oracle manipulation

Next lesson (M3L4): *Decentralized Finance (DeFi)* — How do you build an entire financial system — lending, borrowing, trading, insurance — without banks? We explore AMMs, liquidity pools, yield farming, and the composability that makes DeFi both powerful and fragile.

Review: Can you explain why gas exists, how a rollup inherits Layer-1 security, and why bridge exploits are so costly?

Common Misconceptions About Smart Contracts

Misconception	Reality
"Smart contracts are legally enforceable contracts"	A Solidity program is code, not a legal contract. Whether it binds parties legally depends on jurisdiction, identification, and consent — not on running on a blockchain. See M7L3 MiCA.
"Smart contracts are immutable, so they are safe"	Immutable code means bugs are also immutable. The DAO (2016) and Parity wallet freeze (2017) show immutability can lock in vulnerabilities.
"Code is law"	Only if the outcome is socially acceptable. Ethereum hard-forked to reverse The DAO hack, demonstrating the community, not the code, is the final arbiter.
"Gas fees prevent spam"	Gas limits transactions per block, not economic spam. MEV bots and sandwich attacks thrive precisely because gas is an open-bid auction.
"Oracles just read data from the internet"	Oracles are a trust bottleneck. A smart contract reading a single oracle is only as decentralised as that oracle. Chainlink uses 31+ nodes per feed for this reason.

The gap between "smart contract" marketing and engineering reality is where exploits live.

Case: The DAO Hack (2016)

What happened

- The DAO was an Ethereum-based decentralised venture fund that raised $150M in May 2016
- Attacker exploited a re-entrancy bug to drain 3.6M ETH ($60M at the time) over 17 June 2016
- Ethereum community split: a hard fork on 20 July 2016 reversed the theft; the unforked chain became Ethereum Classic
- Slock.it (DAO authors) had identified the bug; the fix was being merged when the attack landed
- SEC Section 21(a) Report (25 July 2017) concluded DAO tokens were unregistered securities

Why it matters here

- Live demonstration that 'code is law' fails when the code has bugs
- First-principles example for M3L3: re-entrancy + missing checks-effects-interactions pattern
- Concrete precedent for SEC + MiCA classification of smart-contract tokens

Full writeup: `v4/cases/case_M3_dao_hack.md`. **Host: M3 L3 Smart Contracts.**

Case: The DAO Hack – Lessons

- The same re-entrancy bug class powered the bZx (2020), Cream (2021), Fei (2022), and Curve (2023) exploits
- Hard-forking to undo theft is socially possible but credibly-neutral chains pay a legitimacy cost
- Audit firms (Trail of Bits, ConsenSys Diligence) emerged as a direct response market to this incident
- Formal verification (Certora, K-framework) became mainstream for protocol-critical code post-DAO

See also: M3 L3 Smart Contracts; full writeup at `v4/cases/case_M3_dao_hack.md`.

Self-Check: Smart Contracts

Attempt these before turning the page.

1. [Understand] What is a reentrancy attack and why did The DAO (2016) fall to it? State the fix pattern (check-effects-interactions) in one line.
2. [Apply] Solidity transaction uses 200,000 gas. Gas price 30 gwei. ETH at $3,000. What does the user pay in USD? Same transaction on Arbitrum (L2) costs 1% of L1 gas — what does the user pay?
3. [Analyze] Ethereum hard-forked to reverse The DAO hack. Ethereum Classic (ETC) continued with the original chain. Did the fork prove "code is law" wrong, or prove that communities value social consensus over code? Defend with an argument.

Solutions hidden unless `\solutionstrue` is set before compiling.