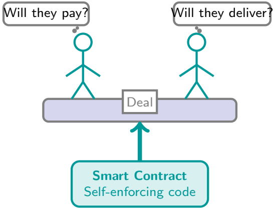
What If a Contract Could Enforce Itself — With No Lawyer, No Court, No Appeal?

Two parties make a deal. One side must trust the other to follow through. Lawyers cost money, courts take time, enforcement is uncertain. What if the agreement itself could guarantee compliance?

Three problems with traditional enforcement:

1. **Cost** — intermediaries extract fees at every step
2. **Speed** — resolution takes weeks to years
3. **Access** — not everyone can afford legal representation

These barriers mean that for small transactions, enforcement is not worth the cost. For large ones, it favors whoever has more resources.



No judge. No delay. No escape.

A smart contract replaces the enforcer with code — the agreement executes automatically when conditions are met, removing the need for trust in any third party.

Think About a Promise Someone Broke — What Would Change If a Machine Enforced It?

A friend borrows money and promises to pay you back by Friday. Friday comes. Nothing happens. You ask. They say next week. Next week becomes next month.

Now imagine a different version:

The promise: Your friend deposits the amount into a shared digital lockbox at the moment of borrowing.
The rule: If they do not send the payment by Friday, the lockbox automatically returns your original deposit.
The guarantee: Neither of you can change the rules after agreeing.

No awkward conversation. No broken friendship. No court. The mechanism handled it.

The Core Idea

This is the core idea behind smart contracts: agreements where the enforcement mechanism is built into the agreement itself. The code does not forget, delay, or make exceptions.

What Makes a Smart Contract Different from a Regular One?

Aspect	Traditional Contract	Smart Contract
Written in	Natural language	Programming code
Enforced by	Courts, lawyers	Automatic execution
Ambiguity	Possible, needs interpretation	Impossible, deterministic
Amendment	Renegotiation	New deployment required
Transparency	Private document	Public, auditable

The shift from language to code eliminates ambiguity but also eliminates flexibility — you cannot plead extenuating circumstances to a program. What the code says is what the code does, regardless of intent or context.

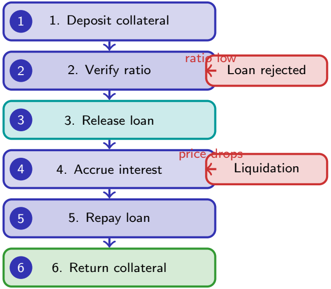
Key properties that distinguish smart contracts:

- **Deterministic** — same input always produces same output
- **Immutable** — once deployed, code cannot be changed
- **Transparent** — anyone can read and verify the code
- **Self-executing** — no intermediary triggers execution

These properties taken together create a fundamentally new kind of agreement: one where compliance is not optional, and enforcement costs nothing.

The removal of ambiguity is both the greatest strength and the greatest risk. Code cannot exercise judgment — it can only follow rules.

Follow One Loan from Request to Repayment — No Humans Involved



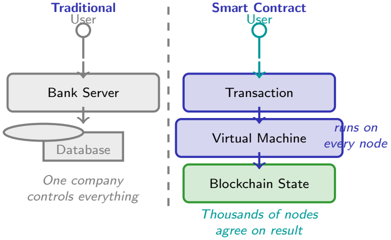
What happened in those six steps:

- No loan officer reviewed the application
- No credit score was checked — only collateral value
- No bank held the funds — the contract did
- No collection agency was needed — the code liquidated automatically

The entire process — from application to repayment — ran as code. Every decision point had a clear rule and an automatic outcome.

Every step that removes a human also removes a point of judgment. The loan is faster, cheaper, and more predictable — but it cannot consider context, hardship, or exceptional circumstances.

Where Does a Smart Contract Actually Run — And Who Pays for It?



Single server vs. Global consensus

Who pays for execution?

Gas:	a fee measured in computation steps — like paying postage by weight
Simple operations:	cost very little, such as adding two numbers
Storage writes:	cost thousands of times more, because data is saved permanently
Failed transactions:	still cost gas — you pay for the attempt even if it fails

The fee structure incentivizes efficient code and prevents spam — no one can run an infinite loop because they would run out of gas.

A smart contract trades cheap centralized execution for expensive decentralized consensus. The cost buys something specific: no single party can alter, censor, or shut down the program.

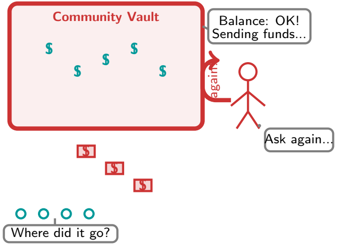
The Code Worked Perfectly — So Why Did Everyone Lose Their Money?

A community pooled funds into a shared investment contract. The code was public. The rules were clear. Then someone found a loophole.

The attack pattern — reentrancy:

1. Attacker calls the “withdraw” function
2. Contract sends funds to attacker
3. Before updating the balance, attacker’s code calls “withdraw” again
4. Contract still shows the old balance — sends funds again
5. Repeat until drained

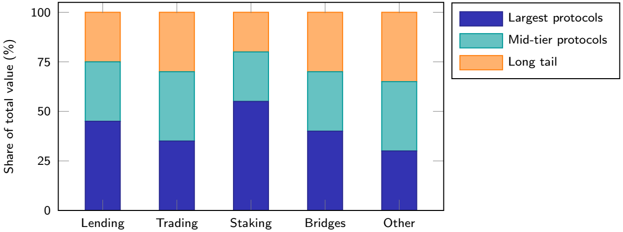
The code executed exactly as written. Every step was valid according to the rules. The problem was not a bug in execution — it was a flaw in logic.



The code did exactly what it was told.

Immutability is a double-edged sword. It guarantees that no one can tamper with the rules — but it also guarantees that a flaw in those rules cannot be fixed after deployment.

How Much Value Is Already Locked in Smart Contracts?



Illustrative distribution based on public DeFi data patterns. Not actual protocol data.

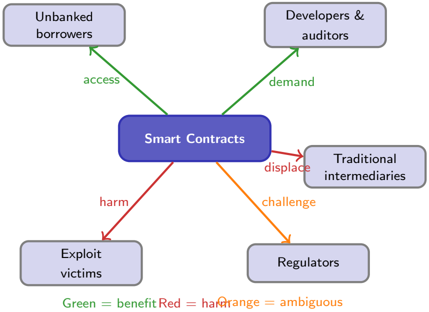
What these contracts actually do:

Lending:	Deposit collateral, borrow assets, earn interest — all without a bank
Trading:	Swap tokens directly through automated market makers — no broker
Staking:	Lock tokens to secure a network and earn rewards
Bridges:	Move assets between different blockchains

The concentration pattern matters: in most categories, a small number of protocols hold the majority of value. Smart contracts create winner-take-most dynamics.

Billions in value are already managed by code, not institutions. The question is no longer whether smart contracts work — it is whether they work safely enough at this scale.

Who Benefits from Unstoppable Code — And Who Gets Hurt?



The distribution of impact is uneven:

- **Winners:** Those previously excluded from financial services gain access. Developers and auditors face growing demand for their skills.
- **Losers:** Traditional intermediaries lose revenue streams. Exploit victims have no recourse — no insurance, no refund, no appeal.
- **Ambiguous:** Regulators face a technology that operates across borders, resists censorship, and moves faster than legislation.

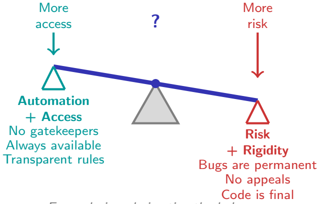
The same properties that empower the unbanked also empower the attacker.

Unstoppable code is neutral. It serves whoever invokes it. Whether that is a net positive depends on the balance between new access created and new harm enabled.

Three Questions That Reveal Whether a Smart Contract Is Safe

Before trusting value to any smart contract, ask three questions. The answers will not tell you what to do — but they will tell you what you are trusting.

1. **Has the code been audited — and by whom?**
   An audit reduces risk but does not eliminate it. Many exploited contracts had multiple audits. Ask who audited it, when, and what scope was covered.
2. **Can the contract be upgraded — and by whom?**
   Upgradeable contracts can fix bugs, but they reintroduce the trust you were trying to remove. Whoever holds the upgrade key holds the power.
3. **What happens if something goes wrong?**
   Immutable contracts have no rollback. If there is no emergency pause, no insurance fund, and no governance process, then a flaw means total loss.



Every design choice tips the balance.

Trustless does not mean risk-free. It means the risk has shifted from trusting people to trusting code. Whether that is an improvement depends on the quality of the code and the design of the system.

Your Challenge: Audit This Contract Design

A startup proposes the following smart contract system for freelance payments:

The promise: Clients deposit payment when hiring a freelancer. The contract releases funds when the work is marked complete.
The rule: The client marks work as complete, OR a deadline passes and the freelancer can claim the funds.
The upgrade: The startup holds an admin key that can pause the contract and redirect funds in case of disputes.

Apply the three questions from the previous slide:

1. **Audit question:** What parts of this design could contain hidden risks? Where would you want an auditor to focus?
2. **Upgrade question:** The startup holds an admin key. What does this mean for the “trustless” claim? What could the startup do with this power?
3. **Failure question:** If the startup disappears, what happens to funds locked in the contract? Is there a recovery path?

No Single Right Answer

There is no single right answer. The point is to practice evaluating trade-offs: the admin key solves the dispute problem but reintroduces centralized control. Every smart contract design involves this tension between safety and autonomy.