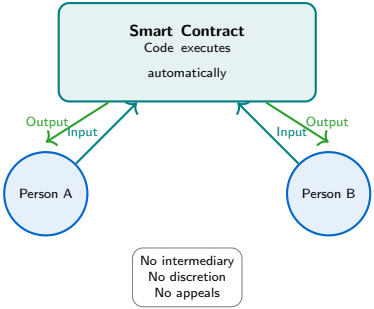
What if a contract could enforce itself – with no lawyers, no judges, no appeals?

Traditional contracts require trust:

- You sign an agreement with another party
- If they break the terms, you hire a lawyer
- A judge interprets the contract and enforces it
- The process is slow, expensive, and subjective

Smart contracts are different:

- Code defines the rules, not natural language
- Execution is automatic when conditions are met
- No judge, no lawyer, no human interpretation
- Once deployed, the code cannot be changed



The promise: Trust is replaced by mathematics.
The risk: Bugs are permanent and unstoppable.

Key Insight

Smart contracts eliminate the need for trust by replacing judges and lawyers with verifiable code. But immutability means bugs become permanent exploits.

Traditional contracts depend on human enforcement. Smart contracts enforce themselves through code.

The Vending Machine That Never Lies

Person 1: "I put in the money, but it didn't give me my snack!"
Person 2: "Did you check the blockchain? The smart contract says you ordered 'air.'"

A smart contract is a vending machine for agreements: insert the right input, get the guaranteed output, no arguing.

Why This Matters

A smart contract is like a vending machine – it executes exactly as programmed, with no room for interpretation or renegotiation.

Have you ever wished an agreement would just execute automatically without arguing?

What are the key differences between legal contracts and smart contracts?

Legal contracts:

- Written in natural language
- Open to interpretation by judges
- Enforced by courts and legal systems
- Can be amended by mutual agreement
- Require trusted third parties

Smart contracts:

- Written in programming languages
- No interpretation – code executes deterministically
- Enforced by blockchain consensus
- Immutable once deployed
- No third party needed

Property	Legal	Smart
Language	Natural	Code
Enforcement	Courts	Blockchain
Flexibility	High	None
Trust model	Intermediary	Mathematics
Cost	High	Low
Speed	Days	Seconds

Example: A rental deposit.
Legal: Landlord holds deposit. Tenant argues about deductions. Judge decides.
Smart: Contract holds deposit. Condition met or not. Code decides instantly.

Key Insight

Legal contracts are flexible but expensive and slow. Smart contracts are fast and cheap but inflexible and unforgiving.

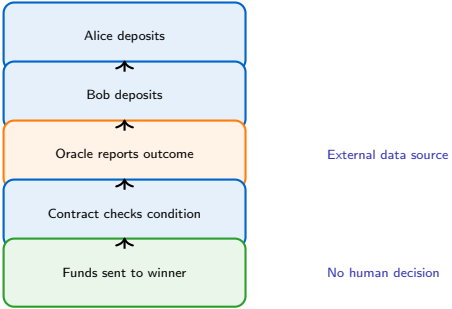
Smart contracts trade human judgment for automated certainty – a powerful trade-off with serious consequences.

How does a simple smart contract execute a conditional payment step by step?

Scenario: Alice and Bob bet on a coin flip. Winner gets both deposits.

Step-by-step execution:

1. Alice deposits funds into the contract
2. Bob deposits the same amount
3. Oracle reports the outcome
4. Contract checks condition
5. If Alice wins, contract sends all funds to Alice
6. If Bob wins, contract sends all funds to Bob
7. No human intervention at any step



Key properties:

- Funds are locked until condition is resolved
- Neither party can withdraw prematurely
- Outcome is automatic and instant
- No arbitrator, no escrow service

Key Insight

Smart contracts execute conditional logic automatically. Funds are locked, conditions are checked, and outcomes are enforced by code.

A smart contract is a programmable escrow agent that never sleeps, never cheats, and never changes the rules.

(c) Joerg Osterrieder 2025–2026

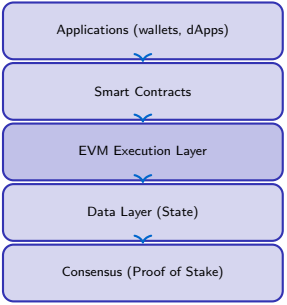
How is the smart contract execution environment architected?

The Ethereum Virtual Machine (EVM):

- A sandboxed computation environment
- Runs on every node in the network
- Executes bytecode compiled from high-level languages
- Deterministic: same inputs always produce same outputs
- Metered by gas to prevent infinite loops

Execution flow:

1. Developer writes contract in high-level language
2. Compiler converts code to bytecode
3. Bytecode is deployed to the blockchain
4. User sends transaction calling a function
5. Every node executes the bytecode identically
6. State changes are recorded on the blockchain



You are here

Analogy: The EVM is a global calculator. Everyone runs the same equation and agrees on the answer.

Key Insight

The EVM is a deterministic execution environment that runs identical code on thousands of nodes, ensuring every participant agrees on the result.

Ethereum extends the blockchain from a ledger of transactions to a platform for programmable agreements.

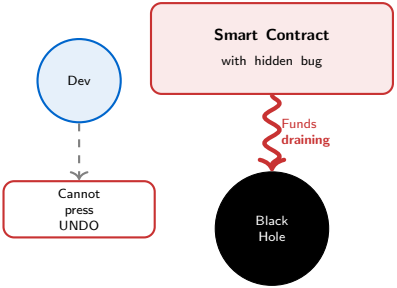
What happens when a smart contract has a bug that cannot be patched?

Immutability is a double-edged sword:

- Once deployed, code cannot be changed
- A bug discovered after deployment is permanent
- Attackers can exploit the bug repeatedly
- No patch, no rollback, no emergency stop

Real-world consequences:

- Billions lost to smart contract bugs
- Reentrancy attacks, overflow errors, logic flaws
- No court can reverse a transaction
- Users have no recourse except hard forks



The paradox: Immutability builds trust in the system but prevents fixing mistakes.

Key Insight

Smart contract bugs are permanent. Immutability means security audits and formal verification are not optional – they are essential.

Code is law, but buggy code is buggy law. Immutability prevents both corruption and correction.

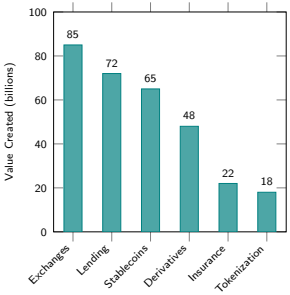
Where are smart contracts creating the most value in financial services?

High-value use cases:

- **Decentralized exchanges:** Trading without intermediaries
- **Lending protocols:** Borrow and lend without banks
- **Stablecoins:** Algorithmic currency stabilization
- **Derivatives:** Automated settlement of financial contracts
- **Insurance:** Parametric payouts triggered by oracles
- **Asset tokenization:** Fractional ownership of real-world assets

Why financial services lead adoption:

- High intermediation costs in traditional finance
- Trust is expensive and slow to build
- Smart contracts reduce counterparty risk
- Composability enables rapid innovation



Key pattern: Applications that reduce intermediation costs and automate trust capture the most value.

Key Insight

Smart contracts thrive where trust is expensive and automation is valuable. Financial services dominate because they are both.

Decentralized exchanges, lending protocols, and stablecoins account for the majority of smart contract value.

Who benefits from unstoppable code and who is harmed by it?

Winners:

- **Users without access to banks:** Financial services become permissionless
- **Developers:** Build financial products without banking licenses
- **Traders:** Execute trades without intermediaries or custody risk
- **Entrepreneurs:** Launch protocols with minimal capital

Losers:

- **Victims of bugs:** No recourse when code fails
- **Victims of scams:** No regulatory protection
- **Regulators:** Cannot enforce consumer protection rules
- **Incumbents:** Banks and exchanges lose market share

Stakeholder	Impact
Unbanked users	Access
Developers	Opportunity
Regulators	Loss of control
Scam victims	No recourse
Incumbents	Disruption
Innovators	Composability

Critical question: Should society prioritize freedom from intermediaries or protection from irreversible mistakes?

The tension: Unstoppable code creates freedom and risk in equal measure.

Key Insight

Smart contracts distribute power away from institutions and toward individuals, but they also distribute risk and remove safety nets.

Unstoppable code eliminates gatekeepers but also eliminates emergency brakes and customer support.

Three questions to assess whether a use case truly needs a smart contract

The Smart Contract Necessity Test:

Question 1: Is trust between parties genuinely absent?

- If parties already trust each other, a database is cheaper
- Smart contracts shine when trust is impossible

Question 2: Can the logic be fully specified in advance?

- If rules need human judgment, a contract cannot encode them
- Smart contracts require deterministic conditions

Question 3: Is immutability a feature or a liability?

- If you need to fix mistakes, immutability is a bug
- If you need unstoppability, immutability is essential

Use Case	Needs SC?
International payment	Yes
Company payroll	No
Insurance payout	Yes
Loan negotiation	No
Betting escrow	Yes
Rental agreement	Maybe

Red flags:

- "We need to change it later"
- "There will be exceptions"
- "A judge might decide"

Green lights:

- "No intermediary can be trusted"
- "Rules are 100 percent fixed"
- "It must be unstoppable"

Not every agreement needs a smart contract. Use the necessity test to avoid overengineering.

Your Challenge

Design a Smart Contract for a Real-World Scenario

Scenario: A rental deposit for an apartment.

Your task:

1. Specify the conditions under which the deposit is returned to the tenant
2. Specify the trigger event that executes the contract
3. Define the payout rules (who gets what, and when)

Then answer:

- What is one thing that could go wrong that the contract cannot handle?
- Would a traditional contract or a smart contract be better for this use case? Why?

Hint: Think about edge cases like disputes over damage, early termination, or partial deductions.

Reflection

Smart contracts excel at automating simple, deterministic agreements. They struggle with subjective judgments and exceptions.

The best way to understand smart contracts is to try designing one and discover its limits.