Blockchain Economics and Digital Assets

Lecture 3: Smart Contracts and Decentralised Applications

Dr Daniele Bianchi

Queen Mary, University of London

Semester B, 2025/2026

Contents

Overview

In the previous lectures, we examined blockchain as a system for transferring value—digital cash without intermediaries. Bitcoin demonstrated that peer-to-peer payments are possible without banks. But transferring value is only one of many economic activities that rely on trusted intermediaries. What about contracts, agreements, and more complex financial arrangements?

This lecture introduces **smart contracts**—programs that execute automatically on a blockchain when predetermined conditions are met. Smart contracts extend blockchain from a payment network to a general-purpose platform for programmable agreements. Combined with user interfaces, they enable **decentralised applications** (DApps) that operate without centralised control.

The economic implications are significant. Smart contracts can automate escrow, enforce agreements, manage tokens, and coordinate complex multi-party interactions—all without relying on courts, lawyers, or trusted intermediaries. But this power comes with fundamental limitations. Smart contracts cannot access external data directly; they execute code literally, even when the outcome is unintended; and bugs cannot be patched once deployed.

This lecture covers the core concepts, technical foundations, economic mechanics (particularly gas fees), limitations, and the broader ecosystem of decentralised applications built on these foundations.

1 From Value Transfer to Programmable Money

1.1 Bitcoin vs Ethereum

Bitcoin (2009) demonstrated that digital cash without intermediaries is possible. But Bitcoin's scripting capabilities are deliberately limited. You can specify conditions for spending ("this output can be spent if signature X is provided"), but little more. Bitcoin was designed for one purpose: peer-to-peer payments.

Ethereum (2015) asked a more ambitious question: what if we could program *any* financial agreement to execute automatically? Rather than a payment network, Ethereum is a programmable platform—often described as a "world computer"—where programs called smart contracts can hold funds, enforce rules, and interact with each other.

	Bitcoin	**Ethereum**
Primary purpose	Value transfer	Programmable applications
Native currency	BTC	ETH
Scripting	Limited	General-purpose
Programs on-chain	No	Yes (smart contracts)
Consensus (2024)	Proof-of-Work	Proof-of-Stake

ETH serves two purposes: a store of value (like BTC) and "fuel" to pay for computation on the network.

1.2 A Motivating Example: Escrow

Consider a common problem: Alice wants to buy a laptop from Bob online. Neither trusts the other. How do they transact safely?

The traditional solution is an escrow service—PayPal, a bank, or a lawyer. Alice pays the escrow agent; Bob ships the laptop; Alice confirms receipt; the escrow releases funds to Bob. If there's a dispute, the escrow arbitrates.

This works, but the escrow agent charges fees, can make errors, might be unavailable in certain jurisdictions, and must be trusted to behave honestly.

A smart contract offers an alternative. A program holds Alice's funds. The contract specifies the conditions for release: confirmation from both parties, or a timeout that returns funds to Alice. The rules are public in the code; execution is guaranteed by the network; no middleman is required.

This example illustrates the smart contract value proposition: automated enforcement of agreements without trusted intermediaries.

2 What is a Smart Contract?

2.1 Definition

> A **smart contract** is a program stored on a blockchain that automatically executes when predetermined conditions are met.

The analogy often used is a vending machine. Rules are fixed and visible (insert $2, select item). Execution is automatic (machine dispenses item). There is no negotiation, no discretion, no human involvement. Once deployed, the machine behaves identically for everyone.

A smart contract operates similarly: rules are written in code, and the blockchain guarantees that those rules execute exactly as written.

2.2 The "Contract" Misnomer

An important clarification: smart contracts are **not legal contracts**. They do not have legal standing in most jurisdictions. They cannot understand intent or context. They execute exactly what the code specifies—even if that produces an outcome nobody wanted.

A better mental model: smart contracts are *autonomous agents* or *self-executing programs*. The term "contract" reflects the original vision of Nick Szabo, who coined it in 1994 to describe agreements embedded in hardware and software. The blockchain implementation is more limited than his original conception.

2.3 Key Properties

Smart contracts exhibit several defining characteristics:

Immutable: Once deployed, the code cannot be changed. This is simultaneously a feature (predictability) and a bug (errors cannot be fixed).

Deterministic: Given the same inputs, the contract always produces the same outputs. This is essential—every node must arrive at identical results when executing the same transaction.

Atomic: Transactions either fully complete or fully fail. There is no partial execution. If any step fails, the entire transaction reverts, preventing inconsistent states.

Transparent: Anyone can inspect the code and its execution history. This enables auditing but also reveals vulnerabilities to potential attackers.

Permissionless: Anyone can deploy or interact with contracts. No approval is required, but there is also no recourse if something goes wrong.

2.4 Capabilities and Limitations

Smart contracts can:

- Hold and transfer cryptocurrency
- Enforce rules automatically (e.g., release payment when conditions are met)
- Create and manage tokens (both fungible and non-fungible)
- Interact with other contracts (composability)
- Record data immutably

Smart contracts **cannot**:

- Access external data directly (they require oracles—more on this below)
- Initiate actions on their own (they must be triggered by a transaction)
- Be modified after deployment (except through upgrade patterns that introduce trust)
- Understand intent (they only execute literal code)

3 How Smart Contracts Work

3.1 The Lifecycle

A smart contract goes through four phases:

Development: A developer writes code in a high-level language (Solidity is the most common for Ethereum). The code is then compiled into bytecode that the Ethereum Virtual Machine can execute.

Deployment: A special transaction publishes the bytecode to the blockchain. The contract receives a unique address (similar to a user account). Deployment costs gas, with more complex contracts costing more.

Interaction: Users send transactions to the contract address, specifying which function to call and with what inputs. The contract executes and updates its state accordingly.

State persistence: Results are written to the blockchain permanently. Every node maintains the same contract state.

3.2 The Ethereum Virtual Machine

Every Ethereum node runs the same virtual machine—the **Ethereum Virtual Machine** (EVM). This is the "engine" that executes smart contract code.

The key insight is that computation is replicated across thousands of nodes. Every validator must execute every transaction to verify its outcome. This deliberate inefficiency is the cost of trustlessness—no single party determines the result; all can verify.

This replication also explains why computation is expensive. Users pay for every operation, not because computation is inherently costly, but because it must be performed thousands of times across the network.

3.3 A Simple Example: Payment Splitter

Consider a contract that automatically splits incoming payments 50/50 between two addresses.

The contract stores two addresses: recipient A and recipient B. When ETH is sent to the contract, it automatically forwards 50% to A and 50% to B.

Once deployed, this contract will *always* split payments 50/50. The developer cannot change the split. The recipients cannot stop it. No one can redirect the funds elsewhere. This is both the power and the danger of smart contracts: the rules are fixed, enforced by the network rather than by any party's ongoing cooperation.

4 Gas and Transaction Fees

4.1 Why Gas Exists

If computation on Ethereum were free, attackers could deploy infinite loops that would freeze the network. Every node would attempt to execute the loop, consuming resources indefinitely.

The solution is to make users pay for every computational step. This payment is denominated in gas—a unit measuring computational work.

Gas serves two purposes: it prevents spam and abuse by making attacks expensive, and it compensates validators for the resources they expend processing transactions.

4.2 How Fees Are Calculated

Since the EIP-1559 upgrade in August 2021, Ethereum transaction fees have two components:

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

The **base fee** is set by the protocol based on network congestion. When blocks are more than 50% full, the base fee increases; when less full, it decreases. Critically, the base fee is *burned*—destroyed rather than paid to validators—reducing ETH supply.

The **priority fee** (or "tip") is set by the user. Higher tips incentivise validators to include your transaction sooner. This fee goes to the validator who includes the transaction.

Example: A simple ETH transfer requires 21,000 gas. If the base fee is 20 gwei (where 1 gwei = 0.000000001 ETH) and you add a 2 gwei priority fee, the total cost is:

$$21{,}000 \times (20 + 2) = 462{,}000 \text{ gwei} = 0.000462 \text{ ETH}$$

More complex operations consume more gas: token transfers require approximately 65,000 gas; complex DeFi transactions can exceed 200,000 gas.

4.3 The Gas Limit

Users also set a gas limit—the maximum gas they are willing to spend. If execution requires more gas than the limit, the transaction fails, but the user still pays for gas consumed up to that point.

Each block also has a maximum total gas (currently around 30 million), which limits computation per block. This is why Ethereum can only process approximately 15 transactions per second on Layer 1—the block gas limit constrains throughput.

4.4 Fee Dynamics

In practice, fees vary dramatically with network activity:

- During low activity, a simple transfer might cost $0.50
- During high activity (NFT mints, market volatility), the same transfer might cost $50 or more

High fees price out small transactions, creating economic pressure for Layer 2 solutions that batch transactions and amortise costs. The base fee burning mechanism also creates interesting supply dynamics: during periods of intense activity, more ETH is burned than issued, making ETH temporarily deflationary.

5 The Limits of Smart Contracts

5.1 The Oracle Problem

Smart contracts can only access data stored on the blockchain. They have no way to directly access real-world information—stock prices, weather data, sports scores, delivery confirmations, or anything else that exists off-chain.

This is the **oracle problem**: how do you bring external data on-chain in a way that maintains the trustless properties of the system?

The solution is **oracles**—services that feed external data to smart contracts. But oracles reintroduce trust. If an oracle lies or fails, the contract executes on false data. The contract itself may be trustless, but the *system* depends on trusting the oracle.

Various approaches attempt to mitigate this: decentralised oracle networks (Chainlink) aggregate data from multiple sources; economic incentives penalise dishonest reporters; reputation systems track oracle reliability. But the fundamental tension remains: smart contracts can only be as trustworthy as their data sources.

We explore oracles further in Topic 4 on DeFi, where price oracles are critical infrastructure.

5.2 "Code is Law"—And Its Consequences

Smart contracts execute exactly as written, *even if that was not what anyone intended.* This principle—sometimes expressed as "code is law"—has profound implications:

- Bugs cannot be patched after deployment
- Exploits are "legitimate" from the blockchain's perspective—they are just transactions that the code allowed
- There is no court to appeal to, no fraud protection, no dispute resolution mechanism
- Lost funds are typically gone forever

This is a philosophical stance, not merely a technical limitation. Proponents argue that the entire point is removing human discretion; the rules should apply equally to everyone regardless of intent. Critics counter that rigid code cannot handle edge cases, mistakes, or malicious exploitation.

5.3 Case Study: The DAO Hack (2016)

The tension between "code is law" and practical reality crystallised in the DAO hack of June 2016.

The DAO was a decentralised investment fund on Ethereum that raised $150 million in ETH. Investors contributed funds and voted on investment proposals.

The exploit: A **reentrancy** vulnerability in the code allowed an attacker to repeatedly withdraw funds before the contract updated its balance. The attacker drained 3.6 million ETH—approximately $60 million at the time.

From the blockchain's perspective, nothing wrong occurred. The attacker made valid function calls; the contract executed as written; the transactions were properly signed and included in blocks. The "attack" was simply using the contract in a way the developers had not anticipated but the code permitted.

The response divided the community. Some argued that "code is law"—the attacker had not violated the protocol, and intervention would undermine Ethereum's credibility. Others argued that theft is theft regardless of the mechanism, and the community should fix an obvious wrong.

The outcome was a **hard fork**—a deliberate rewriting of Ethereum's history to return the stolen funds. The majority followed this new chain (which retained the name Ethereum), while those who rejected the intervention continued the original chain as "Ethereum Classic."

The DAO hack remains a defining moment for smart contract development, illustrating both the real risks of immutable code and the governance challenges when things go wrong.

5.4 Common Smart Contract Vulnerabilities

The DAO was not unique. Billions of dollars have been lost to smart contract exploits, with over $3 billion stolen from DeFi protocols in 2022 alone.

Vulnerability	What Happens
Reentrancy	Attacker calls back into the contract before state updates complete
Integer overflow	Numbers wrap around unexpectedly, enabling theft
Access control	Unauthorised users can call restricted functions
Oracle manipulation	Attacker feeds false price data to contracts
Flash loan attacks	Borrow millions, manipulate markets, repay in one transaction

Security audits help but cannot guarantee safety. Code complexity makes exhaustive verification difficult, and novel attack vectors continue to emerge.

6 Decentralised Applications (DApps)

6.1 What is a DApp?

> A **decentralised application** (DApp) is an application with its backend logic running on smart contracts rather than centralised servers.

A typical DApp has three components:

- **Frontend**: A standard web or mobile interface (built with React, Vue, etc.)
- **Backend**: Smart contracts on Ethereum or another blockchain
- **Wallet**: The user's wallet (e.g., MetaMask) that signs transactions

The key difference from traditional applications: no company controls the backend. Users interact directly with smart contracts, and the logic and data are publicly auditable.

6.2 Benefits of DApps

Zero downtime: Smart contracts run as long as the blockchain exists. There are no server outages, maintenance windows, or service interruptions.

Censorship resistance: No single entity can block users or shut down the application. As long as the blockchain operates, the application remains accessible.

Transparency: Code is public. Anyone can verify exactly what the application does before interacting with it.

Trustless operation: Users do not need to trust the developer. They can verify the code and understand the rules before committing funds.

Composability: DApps can interact with each other, creating building blocks for complex systems. This is sometimes called "money legos"—financial primitives that can be combined in novel ways.

Important caveat: These benefits apply only to the on-chain components. Most DApps have off-chain elements (frontends, APIs, metadata hosting) that remain centralised and can be censored, modified, or shut down.

6.3 Drawbacks of DApps

User experience: Managing wallets, signing transactions, understanding gas fees, and handling failed transactions all create friction. The experience is far less polished than centralised alternatives.

Performance: Blockchain is slow compared to centralised servers. Every state change requires network consensus, introducing latency that traditional databases do not have.

Cost: Users pay gas for every interaction. During network congestion, even simple actions can cost tens of dollars.

Immutability: Bugs cannot be easily fixed. Data cannot be deleted, with implications for privacy (personal data on a public, immutable ledger is problematic).

Hidden centralisation: Many DApps have centralised frontends, administrative keys with special privileges, or upgradeable contracts controlled by small teams. These undermine the "decentralised" label while retaining its marketing appeal.

Reality check: For most use cases, centralised applications are faster, cheaper, and easier to use. DApps make sense when trustlessness or censorship resistance are genuinely valuable—a smaller set of applications than the hype suggests.

6.4 The DApp Ecosystem

Decentralised Finance (DeFi)—covered in Topic 4:

- Uniswap: Decentralised exchange using automated market makers
- Aave, Compound: Lending and borrowing protocols
- MakerDAO: Stablecoin issuance through collateralised debt positions

NFTs and Digital Ownership—covered in Topic 6:

- OpenSea: NFT marketplace
- ENS: Decentralised domain names

Infrastructure:

- Chainlink: Decentralised oracle network
- The Graph: Indexing protocol for querying blockchain data

We will explore these applications in subsequent lectures, examining both their innovations and their limitations.

7 Summary and Looking Ahead

This lecture has covered the foundations of smart contracts and decentralised applications. The key takeaways:

Smart contracts are programs that execute automatically on the blockchain. They are immutable, deterministic, atomic, and transparent—properties that enable trustless automation but also create risks when code contains bugs.

Ethereum's EVM enables programmable money. But computation is intentionally expensive (replicated across thousands of nodes), which is why gas fees exist and why scaling remains challenging.

Gas fees pay for computation and prevent abuse. Under EIP-1559, fees include a burned base fee (deflationary) and a priority tip (to validators). Fees vary dramatically with network demand.

Smart contracts have fundamental limitations. They cannot access external data directly (the oracle problem), they cannot be modified after deployment, and their literal execution can produce unintended outcomes when code is exploited.

DApps offer trustlessness but sacrifice UX and efficiency. They are most valuable when censorship resistance genuinely matters; for most use cases, centralised alternatives are superior on practical dimensions.

In the next lecture, we turn to Decentralised Finance (DeFi)—the application of smart contracts to lending, trading, and other financial primitives. DeFi represents the most developed use case for smart contracts and illustrates both their potential and their risks.

Readings

Required:

- Ethereum Foundation. "Introduction to Smart Contracts." Available at ethereum.org. A clear technical introduction.

Supplementary:

- Szabo, N. (1997). "Formalizing and Securing Relationships on Public Networks." The original articulation of smart contracts.
- Mehar, M. I., et al. (2019). "Understanding a Revolutionary and Flawed Grand Experiment in Blockchain: The DAO Attack." *Journal of Cases on Information Technology*, 21(1), 19–32. A detailed analysis of the DAO hack.

- Chen, Y., & Bellavitis, C. (2020). "Blockchain Disruption and Decentralized Finance: The Rise of Decentralized Business Models." *Journal of Business Venturing Insights*, 13, e00151.