Smart Contracts and Decentralised Applications

ECOM215: Blockchain Economics and Digital Assets

Dr Daniele Bianchi

Queen Mary, University of London

Semester B, 2025/2026

Today's Agenda

From Value Transfer to Programmable Money

What is a Smart Contract?

How Smart Contracts Work

Gas and Transaction Fees

The Limits of Smart Contracts

Decentralised Applications (DApps)

Summary and Next Steps

From Value Transfer to Programmable Money

Bitcoin vs Ethereum: A Key Distinction

Bitcoin (2009): Digital cash

- Transfers value from A to B
- Limited scripting: "If condition X, then pay Y"
- Designed for one purpose: peer-to-peer payments

Ethereum (2015): Programmable platform

- Executes arbitrary programs on the blockchain
- Programs can hold funds, enforce rules, interact with each other
- Designed as a general-purpose "world computer"

The key insight: What if we could program *any* financial agreement to execute automatically, without relying on intermediaries to enforce it?

A Motivating Example: Escrow

The problem: Alice wants to buy a laptop from Bob online. Neither trusts the other.

Traditional solution: Use an escrow service (PayPal, bank, lawyer)

- Alice pays the escrow agent
- Bob ships the laptop
- Alice confirms receipt; escrow releases funds to Bob
- If dispute: escrow arbitrates

Problem: The escrow agent charges fees, can make errors, might be unavailable, and must be trusted.

Smart contract solution:

- A program holds Alice's funds
- When both parties confirm (or a timeout expires), funds release automatically
- Rules are public, execution is guaranteed, no middleman

What Makes Ethereum Different

Ethereum introduced the **Ethereum Virtual Machine** (EVM)—a global, decentralised computer that executes programs called *smart contracts*.

	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: (1) a store of value like BTC, and (2) **"fuel"** to pay for computation on the network.

What is a Smart Contract?

Smart Contracts: Definition

Definition

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

Think of it like a **vending machine**:

- Rules are fixed and visible (insert $2, select item)
- Execution is automatic (machine dispenses item)
- No negotiation, no discretion, no human involvement
- Once deployed, the machine behaves the same way for everyone

A smart contract is similar: the rules are written in code, and the blockchain guarantees that the rules execute exactly as written.

The "Contract" Misnomer

Important clarification: Smart contracts are **not legal contracts**.

- They don't have legal standing (in most jurisdictions)
- They can't understand intent or context
- They execute exactly what the code says—even if that's not what anyone wanted

Better mental model: Smart contracts are **autonomous agents** or **self-executing programs**.

The term's origin: Nick Szabo coined "smart contract" in 1994 to describe agreements embedded in hardware/software. The blockchain implementation is more limited than his original vision.

Key Properties of Smart Contracts

1. **Immutable**: Once deployed, the code cannot be changed.
   - This is a feature (predictability) and a bug (can't fix errors)
2. **Deterministic**: Given the same inputs, always produces the same outputs.
   - Every node that runs the contract gets the same result
3. **Atomic**: Transactions either fully complete or fully fail.
   - No partial execution—prevents inconsistent states
4. **Transparent**: Anyone can inspect the code and its execution history.
   - Enables auditing, but also reveals vulnerabilities to attackers
5. **Permissionless**: Anyone can deploy or interact with contracts.
   - No approval needed, but no recourse if something goes wrong

What Can Smart Contracts Do?

Smart contracts can:

- Hold and transfer cryptocurrency
- Enforce rules automatically (e.g., release payment when condition met)
- Create and manage tokens (ERC-20, NFTs)
- Interact with other contracts (composability)
- Record data immutably

Smart contracts **cannot**:

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

How Smart Contracts Work

The Lifecycle of a Smart Contract

Step 1: Development

- Developer writes code in a high-level language (e.g., Solidity)
- Code is compiled into bytecode the EVM can execute

Step 2: Deployment

- A special transaction publishes the bytecode to the blockchain
- The contract receives a unique address (like an account)
- Deployment costs gas (more complex contracts cost more)

Step 3: Interaction

- Users send transactions to the contract address
- Each transaction specifies which function to call and with what inputs
- The contract executes and updates its state

Step 4: State changes are recorded

- Results are written to the blockchain permanently

A Simple Example: Payment Splitter

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

What the contract stores:

- Address A (first recipient)
- Address B (second recipient)

What the contract does:

- When ETH is sent to the contract, automatically forward 50% to A and 50% to B

Key point: 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 Ethereum Virtual Machine (EVM)

Every Ethereum node runs the same virtual machine—the **EVM**.

What you need to know:

- The EVM executes smart contract code
- Every node runs the same code and must get the same result
- This is how the network reaches consensus on contract outcomes

Why this matters economically:

- Computation is replicated across thousands of nodes
- This is intentionally inefficient (that's the cost of trustlessness)
- Users must pay for this computation—hence **gas fees**

The EVM is the "engine" that makes programmable money possible, but its inefficiency is why fees can be high during congestion.

Gas and Transaction Fees

Why Gas Exists

The problem: If computation is free, attackers could deploy infinite loops that freeze the network.

The solution: Make users pay for every computational step.

Gas

A unit measuring computational work. Every operation has a gas cost. Users pay for the gas their transactions consume.

Examples:

- Simple ETH transfer: 21,000 gas
- Token transfer: ≈65,000 gas
- Complex DeFi transaction: 200,000+ gas

Gas serves two purposes: (1) prevents spam/abuse, and (2) compensates validators for processing transactions.

How Fees Are Calculated

Since August 2021 (**EIP-1559**), Ethereum fees have two components:

Total fee = Gas used × (Base fee + Priority fee)

- **Base fee**: Set by the protocol based on network congestion
  - Increases when blocks are full, decreases when blocks have space
  - This portion is **burned** (destroyed), reducing ETH supply
- **Priority fee** ("tip"): Set by the user
  - Higher tip = faster inclusion
  - This portion goes to the validator

Example: A simple ETH transfer requires 21,000 gas.

- Base fee: 20 gwei per unit of gas (1 gwei = 0.000000001 ETH, one billionth)
- Priority fee (tip): 2 gwei per unit of gas
- Total fee: 21,000 × (20 + 2) = 462,000 gwei = 0.000462 ETH

Gas Limit: Bounding Computation

Users also set a **gas limit**—the maximum gas they're willing to spend.

Why this matters:

- If execution exceeds the gas limit, the transaction fails
- You still pay for the gas consumed up to that point
- Protects users from unexpectedly expensive transactions

The block gas limit:

- Each block has a maximum total gas (≈30 million currently)
- This limits how much computation can happen per block
- It's why Ethereum can only process ≈15 transactions/second on L1

Gas limits are the mechanism that makes Ethereum "quasi-Turing-complete"—it can run any program, but only up to a bounded amount of computation.

Fee Dynamics in Practice

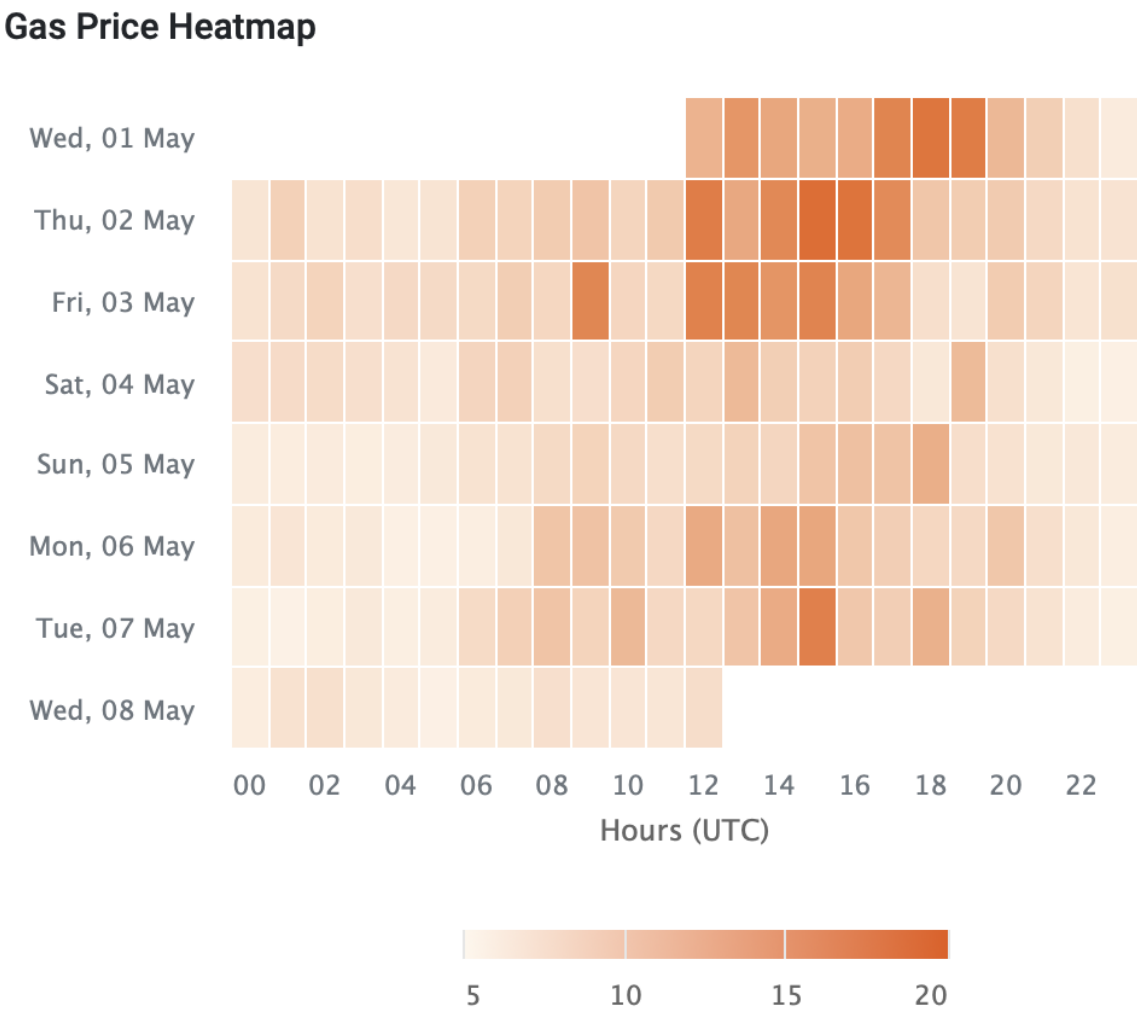

Figure: Intra-day gas prices from 1–8 May 2024. Source: `etherscan.io`.

Fee Dynamics in Practice

Fees vary dramatically based on market activity:

- During low activity: A simple transfer might cost $0.50
- During NFT mints or market crashes: The same transfer might cost $50+

Economic implications:

- High fees price out small transactions
- Users compete for block space through tips
- Validators are incentivised to include highest-paying transactions
- Layer 2 solutions (Topic 2) offer lower fees by batching transactions

EIP-1559's impact: Base fee burning has made ETH supply dynamics complex—during high activity, more ETH is burned than issued, making ETH temporarily deflationary.

The Limits of Smart Contracts

The Oracle Problem

The Problem

Smart contracts can only access data stored on the blockchain. They cannot directly access real-world information.

Examples of data contracts might need:

- Asset prices (for DeFi)
- Weather data (for insurance)
- Sports scores (for betting)
- Shipment delivery confirmation (for trade finance)

Solution: Oracles—services feeding external data to smart contracts.

The catch: Oracles reintroduce trust. If an oracle lies or fails, the contract executes on false data. The contract itself is trustless, but *the system* depends on trusting the oracle.

We'll explore oracles further in Topic 4 (DeFi), where they're critical infrastructure.

"Code is Law"—And Its Consequences

Smart contracts execute exactly as written, *even if that's not what anyone intended*.

Implications:

- Bugs cannot be patched after deployment
- Exploits are "legitimate" from the blockchain's perspective
- There's no court to appeal to, no fraud protection
- Lost funds are usually gone forever

This is a philosophical stance, not just a technical limitation:

- Proponents: The whole point is removing human discretion
- Critics: Rigid code can't handle edge cases, mistakes, or malice

The tension between "code is law" and practical reality has led to contentious debates—including the one we'll discuss next.

Case Study: The DAO Hack (2016)

The DAO: A decentralised investment fund on Ethereum. Raised $150 million in ETH.

The exploit: A **reentrancy bug** allowed an attacker to repeatedly withdraw funds before the contract updated its balance.

- The attacker drained 3.6 million ETH (≈$60 million at the time)
- The attack was "legal" by the contract's code
- From the blockchain's perspective, it was just a series of valid transactions

The response: The Ethereum community voted to "hard fork"—rewrite history to return the stolen funds.

The controversy:

- Supporters: Theft is theft; we should fix it
- Opponents: If we rewrite history once, Ethereum isn't immutable
- Result: Ethereum forked; opponents continued as "Ethereum Classic"

Smart Contract Security Risks

The DAO wasn't unique. Common vulnerabilities include:

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

Scale of the problem: Billions of dollars have been lost to smart contract exploits. In 2022 alone, over $3 billion was stolen from DeFi protocols.

Security audits help but don't guarantee safety. Code complexity makes exhaustive verification difficult.

Decentralised Applications (DApps)

What is a DApp?

Definition

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

Typical architecture:

- **Frontend**: Standard web/mobile interface (React, etc.)
- **Backend**: Smart contracts on Ethereum (or other blockchains)
- **Wallet**: User's wallet (e.g., MetaMask) signs transactions

Key difference from traditional apps:

- No company controls the backend
- Users interact directly with smart contracts
- Data and logic are publicly auditable

Benefits of DApps

Zero downtime: Smart contracts run as long as the blockchain exists—no server outages.

Censorship resistance: No single entity can block users or shut down the application.

Transparency: Code is public; anyone can verify what the application does.

Trustless operation: Users don't need to trust the developer—they can verify the code.

Composability: DApps can interact with each other, creating building blocks for complex systems ("money legos").

Important caveat: These benefits apply only to the on-chain components. Most DApps have off-chain elements (frontends, APIs) that remain centralised.

Drawbacks of DApps

User experience: Managing wallets, signing transactions, paying gas—all friction for mainstream users.

Performance: Blockchain is slow compared to centralised servers. Every state change requires consensus.

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

Immutability: Bugs can't be easily fixed. Data can't be deleted (privacy implications).

Hidden centralisation: Many DApps have centralised frontends, admin keys, or upgradeable contracts—undermining the "decentralised" label.

Reality check: For most use cases, centralised applications are faster, cheaper, and easier. DApps make sense when trustlessness or censorship resistance are genuinely valuable.

DApp Ecosystem: Examples

Decentralised Finance (DeFi)—Topic 4:

- Uniswap: Decentralised exchange
- Aave, Compound: Lending and borrowing
- MakerDAO: Stablecoin issuance

NFTs and Digital Ownership—Topic 7:

- OpenSea: NFT marketplace
- ENS: Decentralised domain names

Infrastructure:

- Chainlink: Oracle network
- The Graph: Indexing protocol

We'll explore DeFi applications in depth in Topic 4.

Summary and Next Steps

Key Takeaways

1. **Smart contracts are programs that execute automatically on the blockchain**
   - Immutable, deterministic, atomic, transparent
2. **Ethereum's EVM enables programmable money**
   - But computation is expensive and slow (by design)
3. **Gas fees pay for computation and prevent abuse**
   - EIP-1559: Base fee (burned) + priority fee (to validator)
4. **Smart contracts have fundamental limitations**
   - Oracle problem: Can't access external data directly
   - Security risks: Bugs are permanent, exploits are costly
5. **DApps offer trustlessness but sacrifice UX and efficiency**
   - Valuable when censorship resistance matters

What's Next

Topic 4: Decentralised Finance (DeFi)

- Automated Market Makers (Uniswap)
- Lending protocols (Aave, Compound)
- Stablecoins
- Oracles in practice (Chainlink)
- Yield farming and risks

Preparation:

- Think about: How could you create a market without a traditional market maker?
- Explore: Look at a DeFi protocol on DeFiLlama (defillama.com)—what metrics are tracked?

Questions?