BLOCKCHAIN & WEB3

Blockchain & Web3 Fundamentals

The trust problem blockchain solves, hashing and digital signatures, blocks and Merkle trees, proof of work vs. proof of stake, Bitcoin's UTXO model, Ethereum's account model and smart contracts, wallets and keys, tokens and NFTs, and scaling with Layer 2 — closing with a full worked transaction tracing every piece together.

10 Chapters | Philip Osztromok | Generated with Claude

CHAPTER 1: WHY BLOCKCHAIN? THE TRUST PROBLEM IT SOLVES

Blockchain & Web3 Fundamentals

Course 1 · Chapter 1 · Why Blockchain? The Trust Problem It Solves

Before any block, any chain, or any cryptographic hash function shows up, blockchain starts as the answer to one specific, genuinely old problem: how do two people who don't trust each other exchange digital money without a trusted third party in the middle? This chapter is entirely about that problem — what it actually is, why it resisted a clean solution for decades, and the real 2008–2009 moment where Bitcoin first solved it.

The Double-Spending Problem

Physical cash solves its own trust problem by being, well, physical. If you hand someone a £10 note, you no longer have it. There is exactly one copy, and possession of that one copy *is* the proof of ownership.

Digital information doesn't work that way. A digital file — a photo, a document, a string of bytes representing "£10" — can be copied perfectly and instantly, as many times as you like, with no way to tell the copy from the original. If digital money were just a file, nothing would stop you from emailing that same "£10" file to two different people, or spending it in two different shops at once. This is the **double-spending problem**: the risk that the same unit of digital currency gets spent more than once, since nothing about the digital token itself prevents duplication.

WHY THIS ISN'T A MINOR TECHNICAL DETAIL

Double-spending isn't a rare edge case to patch around — it's the central obstacle that stood between "digital file" and "digital money" for decades. Any proposed digital currency has to solve it, or it isn't really currency at all; it's just a copyable file with a dollar sign attached to it.

The Traditional Fix: A Trusted Third Party

Every digital payment system that predates blockchain solves double-spending the same way: by appointing a trusted central authority — a bank, a card network, a payment processor — to keep the one authoritative ledger. When you pay by card, your bank checks its own private records, confirms you actually have the money, deducts it, and credits the recipient's bank. The ledger only has one owner, so there's only ever one "true" balance to check against, and double-spending simply can't happen — not because of anything clever about the money itself, but because a single trusted party is watching every transaction and refusing to allow it twice.

This works, and it's how essentially all electronic payments still function today. But it comes with a real, structural cost: every transaction depends on that one trusted party actually being trustworthy, actually staying available, and actually being willing to process your transaction at all. Two people who don't know or trust each other, and don't want to depend on a bank, a government, or a company sitting between them, have no way to transact digitally without simply trusting someone.

Approach	Who Prevents Double-Spending?	What You Must Trust
Physical cash	Physics — a note can't be in two places at once	Nothing (beyond the note itself being genuine)
Bank transfer / card payment	A single trusted intermediary keeping the one real ledger	The bank, the card network, and every system they depend on
Blockchain (Bitcoin, 2009–)	A distributed network of participants who must independently agree on one shared ledger	The math and the incentives, not any single party

Earlier Attempts, and Why They Fell Short

Bitcoin wasn't the first attempt at digital cash, and it's worth being honest about that rather than treating it as having appeared from nowhere. Several real, earlier cryptographic proposals tackled pieces of the same problem:

- **David Chaum's digital cash (1980s, commercialized as DigiCash in the 1990s)** — pioneered genuinely anonymous, untraceable electronic payments using real cryptographic techniques. It still relied on a central issuer to prevent double-spending, though, so it hadn't removed the trusted-third-party problem — only made that party's job cryptographically more private.
- **Adam Back's Hashcash (1997)** — a real proof-of-work scheme, originally built to fight email spam by making a sender do a small, verifiable amount of computational work per message. It wasn't a currency at all, but its core mechanism — "prove you did real, costly work" — is directly what Bitcoin's own mining later reused.
- **Wei Dai's b-money (1998)** — a real proposal for a currency created through solved computational puzzles, without a central issuer, that Satoshi Nakamoto's own Bitcoin whitepaper directly cites.
- **Hal Finney's RPOW, "Reusable Proof of Work" (2004)** — took Hashcash's own proof-of-work idea and made the resulting tokens actually transferable between people.
- **Nick Szabo's bit gold (2005)** — another real proof-of-work-based proposal, explicitly aimed at creating scarce digital value without a central authority, but it never fully solved double-spending on its own and was never actually built as a working system.

Every one of these proposals contributed a real, genuine piece — anonymity, proof-of-work, scarcity, transferability — but none of them combined all the pieces into one working system that solved double-spending *without* a trusted central party. That synthesis is exactly what Bitcoin did.

Bitcoin: The First Working Solution

31 Oct 2008	A link to a whitepaper titled "*Bitcoin: A Peer-to-Peer Electronic Cash System,*" authored under the pseudonym **Satoshi Nakamoto**, is posted to a cryptography mailing list.
3 Jan 2009	Nakamoto mines the very first block of the Bitcoin blockchain — the **genesis block** — and the network goes live.
Today	Satoshi Nakamoto's real identity has still never been confirmed. Genuinely nobody publicly knows — not "it's a mystery for dramatic effect," but a real, unresolved, still-open question in Bitcoin's own documented history.

THE REAL INNOVATION, PRECISELY STATED

Nakamoto's paper didn't invent hashing or digital signatures or proof-of-work — all of that already existed. The genuine innovation was combining them into a system where a large, distributed network of mutually distrusting participants could independently reach agreement on a single, shared transaction history, entirely without a bank, a company, or any other single party anyone had to trust. Chapter 3 covers exactly how blocks and chains do this; Chapter 4 covers exactly how the network reaches that agreement (consensus) in the first place.

What a Blockchain Actually Is, at a High Level

Stripped of the mechanics this course builds up over the coming chapters, a blockchain is:

- a **distributed ledger** — a record of transactions copied across many independent computers ("nodes") rather than stored in one central place;
- that is **append-only** — new transactions can be added, but existing ones can't be secretly rewritten, because doing so would break a cryptographic chain of links back through every earlier block (Chapter 3);
- where the participants **agree on what's true** not because they trust each other, but because of a costly, verifiable process (Chapter 4) that makes cheating more expensive than it's worth.

None of this requires anyone to trust a bank, a government, or a company. It requires trusting that a majority of the network is following the rules honestly — and, crucially, that the system's own incentives make honest behavior the profitable choice. That's a genuinely different kind of trust than "trust this one company," and it's the whole reason blockchain gets described as trust*less* or, more precisely, *trust-minimized*.

A NOTE ON SCOPE FOR THIS COURSE

This chapter deliberately stays at the conceptual level — the actual cryptography (hashing, digital signatures) is Chapter 2's job, and the actual block/chain/consensus mechanics are Chapters 3 and 4. Trying to explain how a blockchain works before explaining *why* anyone needed one in the first place tends to make the mechanics feel arbitrary rather than motivated — so this course deliberately starts with the problem, not the solution.

Hands-On Exercises

Three exercises reinforcing this chapter's own core distinction — trusted-third-party systems vs. trust-minimized ones — before any blockchain mechanics are introduced.

EXERCISE 1

List three real digital payment or messaging systems you use regularly (e.g. a bank app, an email provider, a card network). For each, name the single trusted party whose ledger or server is the one authoritative source of truth, and describe what would go wrong for you personally if that party became unavailable or dishonest.

→ Solution

EXERCISE 2

A friend says: "Bitcoin invented digital money." Using this chapter's own real precursor timeline (Chaum, Back, Dai, Finney, Szabo), write a short, accurate correction explaining what already existed before Bitcoin, and what Bitcoin's own genuine contribution actually was.

→ Solution

EXERCISE 3

Explain, in your own words, why a plain digital file (like a Word document representing "£10") cannot function as money on its own, and identify specifically which property of physical cash that file is missing.

→ Solution

Quick Reference

- **Double-spending** — the risk that the same digital money is spent more than once, since digital data can be copied perfectly.
- **Trusted third party** — the traditional fix: a single central authority (a bank, a processor) keeps the one authoritative ledger.
- **Real precursors** — Chaum's digital cash (1980s), Back's Hashcash (1997), Dai's b-money (1998), Finney's RPOW (2004), Szabo's bit gold (2005) — each contributed a real piece, none combined them into a working, trust-minimized currency.
- **Bitcoin whitepaper** — published 31 October 2008 by the pseudonymous Satoshi Nakamoto, whose real identity remains unconfirmed to this day.
- **Genesis block** — mined 3 January 2009, the real moment the Bitcoin network went live.
- **Blockchain, at a glance** — a distributed, append-only ledger that a network agrees on without needing to trust any single party.

CHAPTER 2: CRYPTOGRAPHIC FOUNDATIONS: HASHING, DIGITAL SIGNATURES & PUBLIC-KEY CRYPTOGRAPHY

Blockchain & Web3 Fundamentals

Course 1 · Chapter 2 · Cryptographic Foundations: Hashing, Digital Signatures & Public-Key Cryptography

Chapter 1 established the problem blockchain solves and the real synthesis Bitcoin achieved in 2008–2009. This chapter opens the toolbox that made that synthesis possible: cryptographic hash functions and public-key cryptography. Nothing about a "block" or a "chain" or a "signed transaction" makes sense without these two tools first, so they get their own chapter before any blockchain-specific mechanics are introduced.

Cryptographic Hash Functions

A **cryptographic hash function** takes any input — a single word, a whole file, an entire block of transactions — and deterministically produces a fixed-size output called a **hash** (or "digest"). Bitcoin's own hash function of choice is **SHA-256**, designed by the NSA and published by NIST in 2001, which always produces a 256-bit output no matter how large or small the input is.

A useful hash function for this purpose needs several real, specific properties:

- **Deterministic** — the same input always produces exactly the same output, every time, on any machine.
- **Fixed-size output** — a one-character input and a 10-gigabyte file both produce a 256-bit digest.
- **The avalanche effect** — changing even a single character of the input produces a wildly, unpredictably different output, with no visible relationship to the small change made.
- **Pre-image resistance** — given a hash output, there's no practical way to work backward and find an input that produces it, short of brute-force guessing (roughly 2^{256} attempts in the worst case — a number large enough to be genuinely infeasible with any foreseeable computing power).

- **Collision resistance** — it's computationally infeasible to find two different inputs that happen to produce the same output.

```
"Pay Alice 10 BTC"                  → SHA-256 →
3f2a...9e7c (illustrative digest)
"pay Alice 10 BTC"                  → SHA-256 →
b81d...44f1 (completely different)
```

(The digests above are illustrative placeholders, not computed SHA-256 output — the real point is structural: capitalizing one letter changes nothing about the message's meaning to a human reader, but produces a completely unrelated-looking hash. That unpredictability is the avalanche effect at work.)

WHERE BITCOIN ACTUALLY USES SHA-256

Two real, distinct places: **mining** (Chapter 4 covers this fully) has miners repeatedly hashing block data with SHA-256, searching for an output that meets a specific difficulty target — the computational "work" in proof-of-work. Separately, Bitcoin applies SHA-256 *twice in sequence* ("double-SHA-256") as one step in turning a public key into a Bitcoin address (Chapter 7 covers this in full).

Public-Key (Asymmetric) Cryptography

Before public-key cryptography existed, encryption was **symmetric**: the same single key both locked and unlocked a message, so both parties needed a way to securely share that one key in advance — a real, practical problem in itself.

Public-key (asymmetric) cryptography solves this differently, using a mathematically related *pair* of keys instead of one shared key:

- a **public key**, which can be shared with absolutely anyone, openly, with no loss of security; and

- a **private key**, which must be kept secret by its owner and never shared with anyone.

The two keys are mathematically linked in a specific, one-directional way: it's computationally easy to derive a public key from a private key, but computationally infeasible to go the other direction and derive the private key from the public one. That one-way relationship is the whole basis of public-key cryptography's security.

Digital Signatures

Digital signatures use this same key pair in reverse of what you might expect for "encryption." Rather than encrypting a message so only the recipient can read it, a **digital signature** lets the owner of a private key prove — publicly and verifiably — that they authorized a specific message, without ever revealing the private key itself:

1. The signer uses their **private key** together with the message (in Bitcoin's case, a transaction) to produce a signature.
2. Anyone holding the signer's **public key** can check that signature against the message and confirm it's valid — without ever needing the private key.
3. Nobody who doesn't know the private key can produce a signature that will pass this check, no matter how hard they try, since forging one would require solving the same computationally infeasible problem that protects the key pair itself.

This is exactly how a Bitcoin transaction proves you actually own the funds you're trying to spend: you sign the transaction with your private key, and every node on the network can verify that signature against your public key, without you ever exposing the private key to anyone. Chapter 7 covers exactly how this plays out in a real wallet and a real transaction.

ECDSA: The Specific Signature Scheme Bitcoin Uses

"Public-key cryptography" describes the general idea; a real system still needs a specific mathematical scheme to actually implement it. Bitcoin's choice is the **Elliptic Curve Digital Signature Algorithm (ECDSA)**, using a specific, named elliptic curve called `secp256k1`.

ECDSA's security rests on the **elliptic curve discrete logarithm problem** — a different hard mathematical problem than the one classical RSA-based signatures rely on. This difference has a genuinely practical payoff: ECDSA can achieve the same real-world security level as RSA using dramatically shorter keys, which means smaller signatures, less data to transmit and store, and faster verification — all real, concrete advantages when a network needs to process and store millions of signed transactions.

A DIRECT, HONEST CROSS-REFERENCE

If you've taken this site's own **Number Theory & Cryptographic Math** course, its own capstone builds and breaks a real, working RSA system — the classic public-key scheme whose security rests on how hard it is to *factor a large number back into its two prime factors*. ECDSA is genuinely different math, not just a rebranded RSA: its security rests on the elliptic curve discrete logarithm problem instead, a structurally unrelated hard problem. Both are real, legitimate ways to build a public/private key pair with the "easy one direction, infeasible the other" property Chapter 2 needs — Bitcoin's own designers chose ECDSA specifically for its shorter-key efficiency advantage over RSA at equivalent security levels.

Scheme	Hard Problem It Relies On	Where It Shows Up on This Site
RSA	Factoring a large number back into its two secret prime factors	Number Theory & Cryptographic Math's own capstone (a real, working RSA implementation)
ECDSA (secp256k1)	The elliptic curve discrete logarithm problem	Bitcoin transaction signing (this course, Chapter 7 in depth)

A REAL, DOCUMENTED FAILURE MODE — NOT A FLAW IN THE MATH ITSELF

ECDSA's own signing process needs a fresh, genuinely random, secret number for every single signature. If that random value is ever reused across two

different signatures from the same private key, or generated by a weak/predictable random number source, an attacker can use the two signatures together to mathematically recover the private key itself — not a theoretical risk, but a real, documented category of attack that has actually compromised real ECDSA implementations in the past (including well-known, publicly documented cases on a games console and an early Android Bitcoin wallet). The elliptic curve math itself isn't broken in these cases — a specific implementation detail (how the random value was generated) was.

Hands-On Exercises

Three exercises reinforcing the two tools this chapter introduces — hashing and public-key signatures — before Chapter 3 assembles them into an actual block.

EXERCISE 1

Explain, using this chapter's own listed hash properties, why a blockchain would be insecure if its hash function had weak collision resistance — specifically, what an attacker could do with two different inputs that hash to the same output.

→ Solution

EXERCISE 2

A colleague says: "A digital signature works like encrypting the message with your private key so only you could have sent it." Identify exactly what's wrong with this description and correct it using this chapter's own actual sign/verify process.

→ Solution

EXERCISE 3

Using this chapter's own RSA-vs-ECDSA comparison, explain why Bitcoin's designers might have chosen ECDSA over RSA specifically, even though both

are legitimate ways to build a public/private key pair.

→ Solution

Quick Reference

- **Cryptographic hash function** — deterministic, fixed-size output; the avalanche effect makes small input changes produce unrelated-looking output; pre-image and collision resistant.
- **SHA-256** — Bitcoin's hash function, 256-bit output, used in mining (proof-of-work) and (as double-SHA-256) in address generation.
- **Public-key cryptography** — a mathematically linked key pair: a shareable public key, and a private key that must stay secret; easy one direction, infeasible the other.
- **Digital signature** — sign with the private key, verify with the public key; proves authorization without ever revealing the private key.
- **ECDSA / secp256k1** — the specific elliptic-curve signature scheme Bitcoin uses; shorter keys than RSA at equivalent security, but genuinely different math (elliptic curve discrete logarithm vs. RSA's integer factoring).
- **Real risk** — reused or weak randomness in ECDSA signing can leak the private key; a documented implementation failure, not a flaw in the underlying math.

CHAPTER 3: HOW A BLOCKCHAIN ACTUALLY WORKS: BLOCKS, CHAINS & MERKLE TREES

Blockchain & Web3 Fundamentals

Course 1 · Chapter 3 · How a Blockchain Actually Works: Blocks, Chains & Merkle Trees

Chapter 2 built the two tools blockchain relies on: hash functions and public-key signatures. This chapter assembles them into the actual structure the word "blockchain" describes — what's really inside a block, how a Merkle tree efficiently proves a transaction is really in one, and precisely how chaining blocks together by hash makes the whole ledger tamper-evident.

What's Actually Inside a Block

A block splits into two real parts: a small **block header** and a much larger **block body** containing the actual list of transactions. The header is what gets hashed, chained to the previous block, and mined — it's deliberately small and fixed-size, while the transaction list underneath it can be large.

Block N — Header

> `previous block hash` — the hash of Block N−1's own header
>
> `merkle root` — a single hash summarizing every transaction in this block
>
> `timestamp` — roughly when this block was created
>
> `difficulty target (bits)` — how hard the proof-of-work puzzle for this block must be
>
> `nonce` — "number used once," adjusted by miners while searching for a valid hash (Chapter 4)

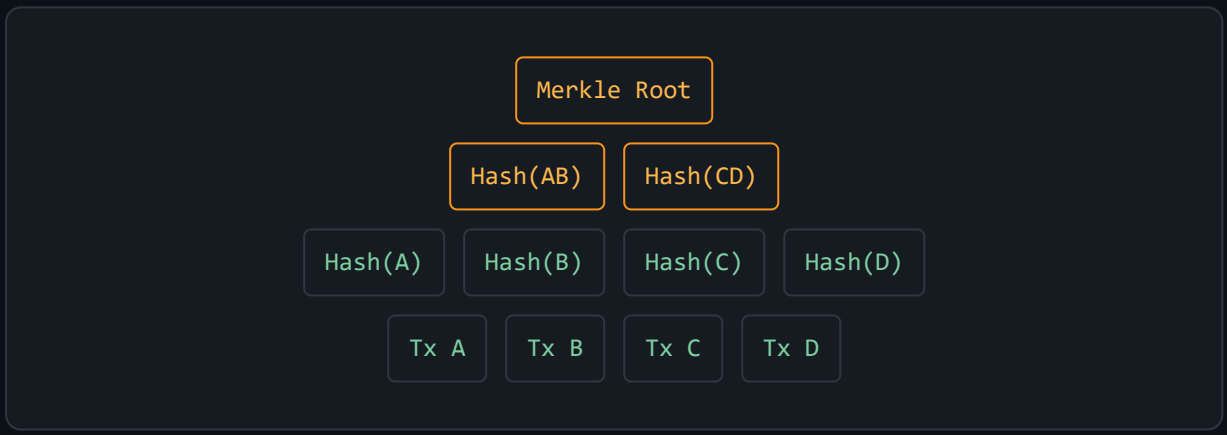
Block N — Body

> Transaction 1, Transaction 2, Transaction 3, … (every transaction included in this block)

Merkle Trees: Proving One Transaction Is In a Block, Efficiently

A block can contain thousands of transactions. If you wanted to prove — and have someone else verify — that one specific transaction is genuinely included in a specific block, the naive approach would be handing over every single transaction in the block and re-hashing all of them. A **Merkle tree**, named after Ralph Merkle, who patented the concept in 1979 (the patent was published in 1982), solves this far more efficiently.

A Merkle tree is a binary tree of hashes: every "leaf" is the hash of one individual transaction, and every node above that is the hash of its two children's hashes combined. This repeats, layer by layer, until a single hash remains at the very top — the **Merkle root**, which is the one value actually stored in the block header.



The real payoff: to prove transaction `B` is included, you don't need every transaction in the block — only `Hash(A)` and `Hash(CD)`, the small handful of sibling hashes on the path up to the root. Anyone can recompute `Hash(AB)` from `Hash(A)` and `Hash(B)`, then `Hash(ABCD)` from that and `Hash(CD)`, and check the result against the block header's own stored Merkle root. The number of hashes needed grows only with the *logarithm* of the number of transactions, not linearly with all of them — genuinely efficient even for a block with thousands of transactions inside it.

SIMPLIFIED PAYMENT VERIFICATION (SPV)

This efficiency is exactly what lets a lightweight Bitcoin wallet — one that hasn't downloaded the entire multi-hundred-gigabyte blockchain — still verify that a specific payment to it is genuinely included in a block. It downloads only the

block headers and a short Merkle proof for its own transaction, rather than every transaction in every block ever mined.

A REAL, DOCUMENTED VULNERABILITY — SECOND-PREIMAGE ATTACKS

A naive Merkle tree implementation can be vulnerable to a real attack where a completely different set of leaf data produces the exact same Merkle root, if internal nodes and leaf nodes aren't distinguished from each other during hashing. Real, production systems (Certificate Transparency is one documented example) defend against this by prefixing leaf hashes and internal node hashes with different marker bytes before hashing, so an internal node's hash can never be mistaken for, or substituted by, a leaf's hash. This is a genuine implementation detail worth knowing, not a flaw in the core Merkle tree concept itself.

Chaining Blocks Together

The "chain" in blockchain comes from one specific field in the header: every block stores the hash of the block that came immediately before it. This single design choice is what turns a list of independent blocks into a genuinely tamper-evident chain.

Here's precisely why, using the avalanche-effect property Chapter 2 already established: if anyone alters even a single transaction inside Block N after the fact, Block N's own header hash changes completely and unpredictably. But Block N+1 already has Block N's *original* hash hard-coded into its own header — so Block N+1 no longer correctly references Block N at all. Worse for the attacker, Block N+1's own header hash has therefore also changed, breaking Block N+2's own reference to it, and so on all the way to the current tip of the chain.

WHY THIS IS GENUINELY EXPENSIVE TO FAKE, NOT JUST AWKWARD

Altering one old block doesn't just break one link — it forces an attacker to redo the proof-of-work (Chapter 4) for that block *and every single block after it*, all before the honest network extends the real chain even further ahead. The deeper in the past a block is, the more blocks sit on top of it, and the more

computational work an attacker would need to outrun the entire rest of the network to catch up and overtake — which is precisely why a transaction buried under many later blocks is considered far more secure than one in the most recent block.

Structure	What It Contains	What It Enables
Merkle tree	A binary tree of transaction hashes, collapsing to one root	Efficiently proving one transaction belongs to a specific block, without downloading them all
Block chain (the header link)	Each block header storing the hash of the block before it	Making the entire history tamper-evident — altering the past breaks every block after it

Chapter 4 covers the piece this chapter has deliberately left open: exactly what "proof-of-work" means, why finding a valid nonce is deliberately expensive, and how a distributed network of mutually distrusting participants uses this to agree on one single, shared version of the chain.

Hands-On Exercises

Three exercises reinforcing block structure, Merkle proofs, and the chain-of-hashes tamper-evidence mechanism before Chapter 4 covers consensus.

EXERCISE 1

A block contains 8 transactions, arranged in a Merkle tree exactly like the diagram in this chapter but one level deeper. If you wanted to prove transaction #5 (out of 8) is included in the block, how many sibling hashes would you need, and why is that number so much smaller than handing over all 8 transactions?

→ Solution

EXERCISE 2

Explain precisely why an attacker who wants to secretly alter a transaction from 5 blocks ago can't just recompute that one block's hash and move on — walk through what actually breaks, block by block, and what the attacker would have to redo to fix it.

→ Solution

EXERCISE 3

Explain, in your own words, why a lightweight wallet using Simplified Payment Verification (SPV) can trust that its own transaction is really in a block without downloading and checking every other transaction in that block.

→ Solution

Quick Reference

- **Block header** — previous block hash, Merkle root, timestamp, difficulty target, nonce.
- **Block body** — the actual list of transactions included in the block.
- **Merkle tree** — a binary tree of hashes (Ralph Merkle, 1979/1982 patent) collapsing many transactions into one root hash.
- **Merkle proof** — only $\log_2(n)$ sibling hashes needed to prove one transaction belongs to a block of n transactions.
- **SPV (Simplified Payment Verification)** — lightweight wallets verify their own transactions using headers and Merkle proofs, without the full chain.
- **The chain** — each block stores the previous block's hash; altering any past block breaks every block's link after it, all the way to the tip.

Blockchain & Web3 Fundamentals

Course 1 · Chapter 4 · Consensus Mechanisms: Proof of Work vs. Proof of Stake

Chapter 3 deliberately left one piece open: what "proof-of-work" actually means, and how a network of mutually distrusting participants, with no central authority, ever agrees on one single, shared version of the chain. This chapter fills that gap in full, then covers the real alternative approach — Proof of Stake — that Ethereum, the second-largest blockchain network, actually switched to in 2022.

Proof of Work: Making Agreement Expensive to Fake

Recall Chapter 3's own header field, `nonce` — "number used once." Mining is the process of repeatedly changing that nonce (and re-hashing the block header with SHA-256 each time) until the resulting hash happens to fall below a specific target value set by the network's current **difficulty**. Because SHA-256 output is unpredictable (the avalanche effect from Chapter 2), there's no shortcut — the only way to find a valid nonce is to keep trying different values and hashing, over and over, which is exactly why this is called "proof of *work*": finding a valid block genuinely requires spending real, verifiable computational effort.

Crucially, verifying a solved block is nearly instant — anyone can hash the header once and check the result against the target — while *finding* that nonce in the first place took an enormous number of attempts. This "hard to solve, trivial to verify" asymmetry is the whole engine behind proof-of-work.

Difficulty Adjustment

Bitcoin's network re-targets its own difficulty every **2016 blocks**. At the network's intended pace of one block roughly every 10 minutes, 2016 blocks works out to exactly 2016 × 10 minutes = 20,160 minutes, or precisely 14 days. Every two weeks, the network checks how long the previous 2016 blocks actually took to mine and adjusts the difficulty target up or down — harder if blocks came in faster than 10 minutes on average (meaning more total mining power joined the network), easier if

they came in slower. This keeps the real average block time close to 10 minutes regardless of how much total computing power is pointed at the network at any given moment.

Resolving Temporary Forks: The Longest-Chain Rule

Because mining is a genuinely random race, it's entirely possible for two different miners, somewhere on the network, to each find a valid next block at nearly the same moment — briefly producing two competing versions of the chain. Bitcoin's rule for resolving this is simple: nodes always follow whichever valid chain represents the most total accumulated proof-of-work (in practice, usually just the longest chain). Miners building on the "losing" branch abandon it and switch to extending the winning one instead, and the transactions in the discarded block return to the pool of unconfirmed transactions to be included in a future block.

WHY "BURIED DEEPER = MORE FINAL" DIRECTLY FOLLOWS FROM THIS

This is precisely why Chapter 3's own "an attacker has to redo the work for every block after the one they altered, faster than the honest network" finding matters in practice: a transaction sitting under several confirmed blocks isn't just "old," it's protected by all the accumulated proof-of-work of every block built on top of it since. Rewriting it means out-competing the entire rest of the network's own honest mining effort, not just solving one puzzle.

The 51% Attack

If a single miner or coordinated group ever controlled more than half the network's total mining power, they could, in principle, mine faster than the rest of the network combined — enough to build an alternative chain that eventually overtakes the

honest one, letting them double-spend coins or censor specific transactions. This is the real, named **51% attack**.

In practice, this stays economically impractical against an established network like Bitcoin, even though it's not theoretically impossible. A real 2025 estimate by Duke University finance professor Campbell Harvey put the cost of executing a week-long 51% attack on Bitcoin, at October 2025 prices, at around $6 billion — a real, staggering figure, but still under 1% of Bitcoin's own total market value at the time, meaning an attacker would be risking an enormous sum against a network whose participants have every incentive to notice and respond.

THE REAL, HONEST COST OF PROOF OF WORK

All of this security comes from a genuinely large, ongoing expenditure of real-world electricity — a documented 2021 University of London study found Bitcoin's own energy consumption was roughly a thousand times higher than the highest-consuming proof-of-stake system studied at the time. This isn't a side effect of a broken design; the expense *is* the security mechanism. But it's also the single most-cited real criticism of proof-of-work, and the direct motivation for the alternative below.

Proof of Stake: Replacing Computation With Skin in the Game

Proof of Stake (PoS) takes a structurally different approach to the same problem — how does the network agree on who gets to propose the next block, without a trusted central party? Instead of racing to solve a computational puzzle, participants ("validators") lock up, or "stake," a quantity of the network's own cryptocurrency as collateral. The network selects who proposes and validates each new block roughly in proportion to how much they've staked, rather than how much raw computing power they control.

Because there's no computational race to win, PoS avoids proof-of-work's own enormous energy cost almost entirely. Security instead comes from the stake itself: a validator caught acting dishonestly (proposing invalid blocks, or supporting two

conflicting chains at once) can have a real, meaningful portion of their staked coins destroyed — a process called **slashing** — giving validators a direct financial reason to behave honestly.

Ethereum's Real Transition: "The Merge"

Ethereum, the second-largest blockchain network by market value, actually made this switch in a real, documented event known as **"The Merge,"** completed in **September 2022** — moving its entire live network from proof-of-work to proof-of-stake. The reported result was dramatic: Ethereum's own energy consumption fell by more than 99% almost immediately, with some sources citing a reduction exceeding 99.9%.

THIS WASN'T A SMALL, EXPERIMENTAL CHANGE

The Merge is one of the largest live migrations in the history of any production software system — an already-running, multi-billion-dollar network switching its own entire consensus mechanism without stopping or restarting the chain. It's worth treating as a real, consequential engineering event, not just a footnote in a comparison table.

	Proof of Work (Bitcoin)	**Proof of Stake (Ethereum, since 2022)**
Who proposes the next block	Whoever solves the computational puzzle first	A validator selected roughly in proportion to staked coins
Cost of dishonest behavior	Wasted electricity and computing hardware	Slashing — a real, direct loss of staked coins
Energy use	Very high — the expense is the security mechanism	Dramatically lower — Ethereum reported a >99% real reduction after 2022
Real, honest criticism	Enormous energy consumption	Wealthier participants can stake more and earn

		proportionally more influence

NEITHER SIDE OF THIS DEBATE IS SETTLED

Proof-of-work advocates point to Bitcoin's own now 15+ year unbroken security track record and the mathematically simple, physically grounded "expensive to fake" logic. Proof-of-stake advocates point to the real, dramatic energy savings Ethereum demonstrated and argue slashing provides comparably strong economic security without the environmental cost. Both are real, live systems securing real value today — this course won't pretend one has definitively "won," since that's a genuinely open, ongoing debate rather than a settled technical question.

Hands-On Exercises

Three exercises reinforcing how each consensus mechanism actually secures the network, before Chapter 5 turns to Bitcoin's own real, specific implementation of everything covered so far.

EXERCISE 1

Explain, using this chapter's own difficulty-adjustment mechanism, what would happen to Bitcoin's real average block time if a huge amount of new mining hardware suddenly joined the network overnight, and why the network wouldn't simply stay stuck producing blocks much faster than 10 minutes forever.

→ Solution

EXERCISE 2

A friend says: "A 51% attack is basically impossible, so proof-of-work is perfectly secure." Using this chapter's own real cost estimate and its own honest framing, correct this overstatement into a more accurate claim.

→ Solution

EXERCISE 3

Compare how proof-of-work and proof-of-stake each give a participant a real, direct financial incentive to behave honestly. Name the specific mechanism in each system, and what a dishonest participant would actually lose.

→ Solution

Quick Reference

- **Proof of work** — miners repeatedly hash the block header with different nonces until the result falls below a difficulty target; hard to solve, trivial to verify.
- **Difficulty adjustment** — Bitcoin retargets every 2016 blocks (~2 weeks at a 10-minute target), keeping average block time stable regardless of total mining power.
- **Longest-chain rule** — the network follows whichever valid chain has the most accumulated proof-of-work, resolving temporary forks.
- **51% attack** — controlling a majority of mining power enables rewriting recent history; real, but economically extreme (a 2025 estimate put a week-long Bitcoin attack around $6 billion).
- **Proof of stake** — validators are selected in proportion to staked coins rather than computing power; dishonesty is punished by slashing (losing staked coins).
- **The Merge** — Ethereum's real September 2022 switch from proof-of-work to proof-of-stake, cutting its own energy use by over 99%.

CHAPTER 5: BITCOIN: THE FIRST BLOCKCHAIN

Blockchain & Web3 Fundamentals

Course 1 · Chapter 5 · Bitcoin: The First Blockchain

Chapters 1 through 4 built up every tool Bitcoin actually needed: the trust problem it solves, hashing and signatures, blocks and Merkle trees, and proof-of-work consensus. This chapter puts all of it together and looks at Bitcoin itself — specifically, how it actually tracks who owns what (a genuinely different model than a bank account), how many bitcoins will ever exist, and the real moment bitcoin was first used to buy something in the physical world.

The UTXO Model: How Bitcoin Actually Tracks Ownership

It's tempting to imagine Bitcoin works like a bank account — a running balance that goes up or down with each transaction. It doesn't. Bitcoin uses something structurally different, called the **UTXO model** — "Unspent Transaction Output."

A UTXO is a discrete, specific chunk of bitcoin, created by a previous transaction, that hasn't been spent yet. There's no single number anywhere on the network that says "Alice has 2.5 BTC." Instead, Alice's real spendable balance is whatever total you get by adding up every UTXO on the entire chain that's currently locked to her own public key. A transaction doesn't adjust a balance — it **consumes** one or more existing UTXOs as inputs, and **creates** one or more brand-new UTXOs as outputs.

```
Input: UTXO worth 1.0 BTC (Alice)  → spent in one transaction →

Output 1: 0.3 BTC (to Bob)   Output 2: 0.69 BTC ("change," back to Alice)

(0.01 BTC left over becomes the miner's transaction fee)
```

Notice the "change" output. A UTXO has to be spent as a whole — you can't spend "part of" a 1.0 BTC UTXO the way you might type a partial number into a bank

transfer. If Alice only wants to send Bob 0.3 BTC out of a 1.0 BTC UTXO, the transaction consumes the whole 1.0 BTC UTXO and creates two new outputs: 0.3 BTC to Bob, and the remainder (minus a small transaction fee) back to Alice as a fresh UTXO she now owns. This is genuinely similar to paying for something with a banknote larger than the price and receiving change back — a deliberate real echo of Chapter 1's own physical-cash analogy.

	UTXO Model (Bitcoin)	**Account Model (previewed for Chapter 6)**
What's tracked	Discrete, unspent "coin" objects scattered across the chain's history	A single running balance number per account
To find your balance	Sum every UTXO currently locked to your public key	Read the one stored number for your account
Real advantage	Each UTXO either exists in full or doesn't — no partial/inconsistent states possible	Simpler to reason about; naturally supports more complex account state (Chapter 6)

WHY THIS DISTINCTION MATTERS LATER

Ethereum, covered in Chapter 6, deliberately chose the account-balance model instead of UTXOs — a genuine, consequential design difference, not just a cosmetic one, and part of why Ethereum can support the more complex, stateful smart contracts that chapter introduces.

A Fixed, Predictable Supply: 21 Million Bitcoins

Bitcoin's total supply is capped at a real, fixed number: **21 million** bitcoins, ever. New bitcoins only enter circulation as the block reward paid to whichever miner

successfully mines each new block — there's no other mechanism for creating new coins.

That block reward isn't constant, though. It started at **50 BTC per block** when the network launched in 2009, and it's cut exactly in half every **210,000 blocks** — an event called a **halving**, which at Bitcoin's roughly 10-minute block time works out to happening approximately every four years. This halving schedule continues until the reward eventually rounds down to zero, at which point the full 21 million supply will have been issued — expected to occur around the year 2140.

A REAL, HONEST FORWARD-LOOKING QUESTION

As the block reward keeps shrinking toward zero over successive halvings, miners' income increasingly has to come from **transaction fees** instead of newly-issued coins — the small amount left over in a transaction like Alice's above. Whether transaction fees alone will provide enough incentive to keep the network's mining power (and therefore its security, per Chapter 4's own 51%-attack economics) at a comparable level once the block reward is negligible is a real, genuinely open question in Bitcoin's own long-term design, not something already settled by actual experience this many decades before it happens.

Bitcoin Pizza Day: The First Real-World Transaction

3 Jan 2009	The genesis block is mined (Chapter 1); the network exists, but bitcoin has no established real-world exchange value yet.
22 May 2010	Programmer **Laszlo Hanyecz** pays **10,000 BTC** for two Papa John's pizzas — the first known commercial transaction using bitcoin, later celebrated annually as **"Bitcoin Pizza Day."**

This is a real, well-documented historical moment, not folklore — and it's genuinely significant precisely because it's the point where bitcoin stopped being a purely theoretical proof-of-concept and was first actually exchanged for a real, physical good in the ordinary world. Given bitcoin's value years later, 10,000 BTC for two pizzas has

become a famous illustration of how uncertain and speculative bitcoin's own future value looked at the time — nobody involved in that transaction was pricing it against what it might be worth decades later.

Hands-On Exercises

Three exercises reinforcing the UTXO model and Bitcoin's real supply mechanics, before Chapter 6 introduces Ethereum's genuinely different account model and smart contracts.

EXERCISE 1

Alice owns a single UTXO worth 2.0 BTC. She wants to pay Bob exactly 0.75 BTC. Describe, using this chapter's own UTXO mechanics, what the resulting transaction's inputs and outputs would actually look like, including what happens to the rest of the 2.0 BTC.

→ Solution

EXERCISE 2

Using this chapter's own halving schedule (50 BTC starting reward, halving every 210,000 blocks), explain in your own words why Bitcoin's total supply approaches but never exceeds 21 million, rather than continuing to grow forever.

→ Solution

EXERCISE 3

Explain why this chapter treats the shrinking block reward as a genuinely open question for Bitcoin's long-term security, rather than a problem that's already been solved — connect your answer back to Chapter 4's own 51%-attack cost reasoning.

→ Solution

Quick Reference

- **UTXO** — a discrete, unspent chunk of bitcoin created by a prior transaction; balance = the sum of every UTXO locked to your public key.
- **Transaction mechanics** — a transaction consumes existing UTXOs as inputs and creates new ones as outputs, including a "change" output back to the sender.
- **21 million cap** — Bitcoin's real, fixed maximum supply, expected to be fully issued around the year 2140.
- **Halving** — the block reward (started at 50 BTC) is cut in half every 210,000 blocks, roughly every four years.
- **Bitcoin Pizza Day** — 22 May 2010, Laszlo Hanyecz's real 10,000 BTC payment for two pizzas, the first known commercial bitcoin transaction.
- **Open question** — whether transaction fees alone will sustain network security once the block reward shrinks toward zero.

Blockchain & Web3 Fundamentals

Course 1 · Chapter 6 · Ethereum & the World Computer: Smart Contracts Introduced

*Chapter 5 was entirely about Bitcoin as digital money. This chapter turns to **Ethereum** — a blockchain built for a genuinely different purpose. Where Bitcoin is deliberately narrow (a ledger for tracking who owns what bitcoin), Ethereum was designed from the start as a general-purpose, programmable platform — often nicknamed "the world computer." This chapter introduces the design differences that made that possible; the full mechanics of writing and running smart contracts are Course 2's own territory.*

Ethereum's Real Origin

Late 2013	**Vitalik Buterin** writes a whitepaper describing Ethereum, proposing a blockchain that could run general-purpose programs, not just track a currency.
Jan 2014	Ethereum is publicly announced at the North American Bitcoin Conference in Miami. Alongside Buterin, the founding team came to include Gavin Wood, Charles Hoskinson, Anthony Di Iorio, and Joseph Lubin, among others.
30 Jul 2015	The Ethereum network goes live with its "Frontier" release — the real, official launch of the platform.

The Account Model: A Genuinely Different Design From Bitcoin

Chapter 5 covered Bitcoin's UTXO model in detail — ownership tracked as a scattered collection of discrete, unspent transaction outputs. Ethereum deliberately chose a

different approach: a straightforward **account-based model**, much closer to a bank balance, with two distinct kinds of accounts:

- **Externally Owned Accounts (EOAs)** — controlled by a private key, exactly like a Bitcoin address (Chapter 2's public/private key pairs apply here too). A person or organization directly controls one of these.
- **Contract Accounts** — controlled not by a private key at all, but by code. This is a genuinely new category Bitcoin's own design doesn't really have an equivalent for, and it's the whole basis of everything the rest of this chapter covers.

Both account types have an address and can hold a balance of ETH (Ethereum's native currency), but only a Contract Account can also hold and run its own program.

	Bitcoin (UTXO)	Ethereum (account-based)
What's tracked	Discrete unspent "coin" objects	A running balance per account, plus optional stored code and data
Controlled by	Whoever holds the private key for a given output	Either a private key (EOA) or autonomous code (Contract Account)
Real purpose	Tracking ownership of a currency	Running arbitrary, general-purpose programs alongside a currency

Smart Contracts: An Idea Older Than Blockchain Itself

It's genuinely worth knowing that the term **"smart contract"** wasn't invented by Ethereum, or even by blockchain technology generally. Computer scientist and legal scholar **Nick Szabo** introduced the term by 1996 — roughly two decades before Ethereum existed — describing a contract enforced through mechanical or software means rather than a legal system alone, and famously illustrated it with the humble vending machine: a physical mechanism that automatically enforces its own terms (insert enough money, receive the item, no human clerk required to adjudicate the exchange).

A **smart contract**, as run on Ethereum, is a program stored inside a Contract Account that automatically executes exactly as written whenever it's called — enforcing an agreement's terms in code rather than relying on a court, a company, or any other trusted party to interpret and carry it out. This is the direct blockchain-era realization of Szabo's original, much older idea.

THE EVM AND SOLIDITY, AT A GLANCE

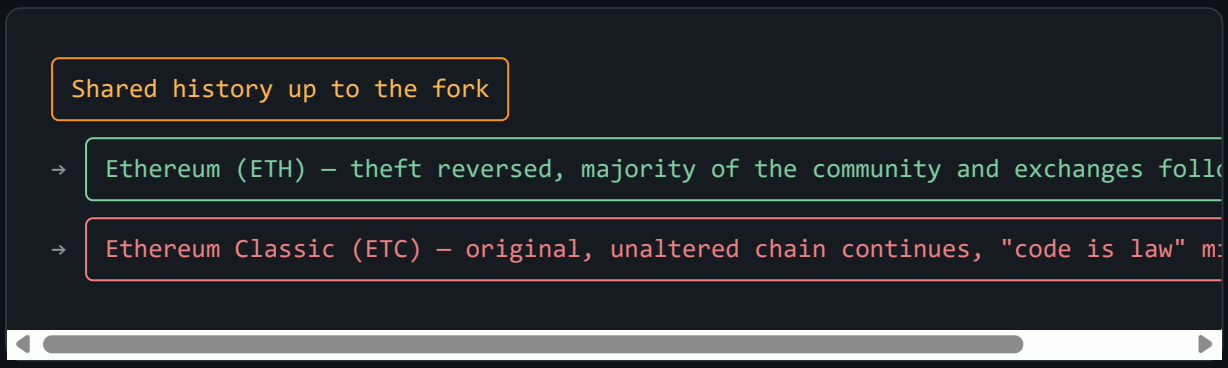
Ethereum runs every smart contract inside the **Ethereum Virtual Machine (EVM)**, a standardized execution environment every full node runs identically, so a contract behaves the same way no matter which node executes it. Contracts are most commonly written in **Solidity**, a Turing-complete programming language purpose-built for the EVM. Executing a contract's code isn't free — every operation costs a small fee called **gas**, paid in ETH, which both compensates the network for the real computation involved and prevents a poorly written or malicious contract from running forever. Course 2 (Smart Contracts, DeFi & Web3 Security) covers the EVM, gas, and writing real Solidity code in full depth — this chapter only needs you to know these pieces exist and roughly what they're for.

The DAO Hack: A Real, Documented Test of "Immutable"

Chapter 3 established that a blockchain's history is tamper-evident — altering the past means redoing an enormous amount of proof-of-work. In 2016, Ethereum faced a real, well-documented event that tested exactly how absolute that immutability really is in practice, not just in theory.

"The DAO" (Decentralized Autonomous Organization) was a smart contract built on Ethereum that raised roughly **$150 million** worth of ETH from thousands of participants in a crowdsale — a huge sum, and a real, early demonstration of what smart contracts could coordinate at scale. In June 2016, an attacker exploited a genuine flaw in the DAO's own contract code to drain approximately **$50 million** worth of ETH into a separate account.

This forced a real, contentious debate across the whole Ethereum community: leave the chain exactly as it was, accepting the theft as a permanent, immutable fact (however unfair it felt), or deliberately execute a **hard fork** — a coordinated, one-time rule change — to effectively reverse the theft and return the stolen funds. The community chose to fork. The result: the chain split into two.



AN HONEST TENSION, NOT A SIMPLE "PROBLEM SOLVED" STORY

This is a genuinely important, still-debated real event, not a footnote. It's real, documented proof that "immutable" describes what happens on a chain by default when nobody with enough coordinated influence chooses to intervene — not an absolute, unbreakable physical law with zero exceptions. Both resulting chains are still real, live, independently operating networks today, and the underlying philosophical disagreement (should code's own outcome always be final, "code is law," versus should a community be able to correct a clear, damaging exploit) remains a genuinely unresolved question in the blockchain world at large, not just a piece of 2016 history.

Another Real Design Difference: No Fixed Supply Cap

Unlike Bitcoin's hard 21 million cap from Chapter 5, Ethereum has no fixed maximum supply of ETH built into its protocol. Its monetary policy has itself changed over time (including adjustments made alongside the real 2022 Merge to proof-of-stake covered in Chapter 4) — a real, ongoing design difference worth knowing about rather than assuming every blockchain necessarily follows Bitcoin's own fixed-supply model.

Hands-On Exercises

Three exercises reinforcing Ethereum's account model, smart contracts, and the real DAO fork, before Chapter 7 turns to wallets and how a real transaction is actually put together and signed.

EXERCISE 1

Explain the real difference between an Externally Owned Account and a Contract Account on Ethereum, and why Bitcoin's own UTXO model doesn't really have an equivalent to the second kind.

→ Solution

EXERCISE 2

A friend says: "Smart contracts were invented by Ethereum." Using this chapter's own real history, correct this claim and explain what Ethereum's own genuine contribution actually was.

→ Solution

EXERCISE 3

Using this chapter's own account of the DAO hack, explain why the resulting hard fork is described as testing, rather than simply confirming, blockchain's own claimed immutability — connect your answer back to Chapter 3's own tamper-evidence reasoning.

→ Solution

Quick Reference

- **Ethereum** — proposed by Vitalik Buterin in a late-2013 whitepaper, publicly announced January 2014, launched 30 July 2015.

- **Account model** — Externally Owned Accounts (private-key controlled) and Contract Accounts (code controlled); a genuine departure from Bitcoin's UTXO model.
- **Smart contract** — a program stored in a Contract Account that automatically executes as written; the term predates blockchain, coined by Nick Szabo by 1996.
- **EVM / Solidity / gas** — the execution environment, primary language, and per-operation fee for running contract code; full depth in Course 2.
- **The DAO hack** — a real June 2016 exploit draining ~$50 million from a $150 million smart contract, resolved by a contentious hard fork that split the chain into Ethereum (ETH) and Ethereum Classic (ETC).
- **No fixed ETH cap** — unlike Bitcoin's 21 million cap, Ethereum has no hard-coded maximum supply.

CHAPTER 7: WALLETS, KEYS & TRANSACTIONS

Blockchain & Web3 Fundamentals

Course 1 · Chapter 7 · Wallets, Keys & Transactions

Every earlier chapter has referred to "your private key" or "your address" as if using one were obvious. This chapter makes it concrete: what a wallet actually is, how it's backed up and recovered, the real, meaningful difference between custodial and non-custodial control, and exactly how a transaction gets built, signed, and sent — tying Chapter 2's signatures, Chapter 5's UTXOs, and Chapter 6's accounts together into one real, end-to-end process.

What a Wallet Actually Is (and Isn't)

A common, genuinely misleading intuition is that a crypto "wallet" holds your coins, the way a physical wallet holds cash. It doesn't. The coins — whether tracked as Bitcoin's own scattered UTXOs (Chapter 5) or an Ethereum account balance (Chapter 6) — live entirely on the blockchain itself, replicated across every node on the network. A wallet's real job is narrower and more specific: it generates, stores, and uses the **private key** that proves you're allowed to spend or move whatever balance is associated with your public address.

Put plainly: losing your wallet doesn't destroy your coins — they're still sitting on the ledger, exactly where they always were. What losing your wallet destroys is your own ability to prove ownership and produce a valid signature (Chapter 2) for them, which in practice is just as final for you personally.

Seed Phrases: One Backup for Every Key You'll Ever Need

Modern wallets don't ask you to separately back up every individual private key you might ever generate. Instead, they use a real, standardized system: a single **seed**

phrase, typically a randomly generated list of **12 to 24** ordinary dictionary words, from which every key pair the wallet ever uses can be **deterministically** regenerated.

A representative (illustrative, not a real usable seed) 12-word example:

1. crane	2. velvet	3. copper	4. orbit
5. gentle	6. mango	7. ripple	8. tunnel
9. finch	10. glacier	11. brave	12. stove

This is possible because most modern wallets are what's called **hierarchical deterministic (HD) wallets**: the seed acts as the single root input to a deterministic process that can generate an entire tree of key pairs — one for every address you ever use — all reproducible from that one seed alone. Write the seed phrase down correctly (on paper, or a comparably durable medium), and every key the wallet ever derives can be regenerated on a brand-new device, even if the original wallet is lost, stolen, or destroyed.

WHY THIS CUTS BOTH WAYS

The seed phrase's own convenience is also its single biggest real risk: anyone who obtains your seed phrase can regenerate every one of your keys and take everything associated with them, permanently, with no recovery process to appeal to — there's no bank or company to call. A seed phrase is, functionally, the single most sensitive piece of information in the entire system.

Hot vs. Cold, Custodial vs. Non-Custodial

Two genuinely separate spectrums are worth keeping distinct:

- **Hot wallet** — keys are generated or stored on an internet-connected device (a phone app, a browser extension). Convenient, but exposed to online attack.
- **Cold wallet** — keys are generated and kept entirely offline (a dedicated hardware device, or even a paper printout), reducing exposure to remote

hacking, though not eliminating risk entirely — offline devices themselves can still have real, documented vulnerabilities of their own.

- **Custodial** — a third party (an exchange, an online service) holds your private keys on your behalf. You're trusting that party completely to keep the keys safe and to actually let you withdraw when you ask to.
- **Non-custodial** — you generate and hold your own private keys directly, with no third party able to freeze or move your funds without your own signature.

> **"NOT YOUR KEYS, NOT YOUR COINS"**
>
> This widely repeated phrase in the crypto community captures the custodial/non-custodial distinction precisely: if a third party holds the private keys, your real, practical claim on those coins depends entirely on that party's own honesty, solvency, and continued operation — not on cryptography alone.

Mt. Gox: A Real, Documented Case of Custodial Risk

This isn't a hypothetical concern. **Mt. Gox**, a Tokyo-based exchange that at its peak handled over 70% of all bitcoin trading worldwide, halted trading on **24 February 2014** and filed for bankruptcy days later, after revealing that roughly **850,000 bitcoins** belonging to its customers were missing (around 200,000 were later recovered, leaving a real net loss of roughly 650,000 BTC). Investigators concluded most of the missing coins had actually been stolen gradually from Mt. Gox's own hot wallet, beginning as early as late 2011 — years before customers ever found out.

Every affected customer had done everything "right" from their own perspective — they held an account with a real, major exchange. But because Mt. Gox itself, not its customers, held the actual private keys, those customers had no independent ability to move or protect their own funds once the exchange's own security had already failed. This real, documented event is exactly what "not your keys, not your coins" is warning against.

How a Transaction Is Actually Built, Signed, and Sent

1. **Construct the transaction.** On Bitcoin, this means selecting one or more existing UTXOs as inputs and specifying the new outputs (Chapter 5). On Ethereum, this means specifying the recipient account, the amount, and (for a contract call) the data to send (Chapter 6).
2. **Sign it.** The wallet uses your private key to produce a digital signature over the transaction data (Chapter 2's own ECDSA/secp256k1 signing process) — proving you authorized it, without ever revealing the private key itself.
3. **Broadcast it.** The signed transaction is sent out to nodes on the network, which independently verify the signature against your public key before relaying it further.
4. **Get included in a block.** A miner (Chapter 4's proof-of-work) or validator (proof-of-stake) includes the transaction in a new block, which propagates across the network.
5. **Confirm.** As more blocks are added on top, the transaction becomes increasingly final, per Chapter 3 and Chapter 4's own "buried deeper is more secure" reasoning.

Question	Answer for This Chapter
Does a wallet store coins?	No — it stores the private key(s) that control coins living on the ledger itself
What does a seed phrase let you do?	Deterministically regenerate every key pair a wallet ever derived, from one 12–24-word backup
Who actually controls your funds in a custodial wallet?	The custodian, not you directly — a real risk Mt. Gox's own 2014 collapse illustrates concretely

Hands-On Exercises

Three exercises reinforcing wallet mechanics and custodial risk before Chapter 8 turns to tokens and NFTs — a different category of asset built on top of everything this chapter has assembled.

EXERCISE 1

A friend says: "I keep all my crypto in my phone's wallet app, so if I lose my phone, I lose my crypto forever." Using this chapter's own seed-phrase mechanics, explain whether this is actually true, and what it depends on.

→ Solution

EXERCISE 2

Using the real Mt. Gox case, explain specifically why holding coins on an exchange is structurally different from holding them in a non-custodial wallet, even if the exchange itself has excellent security practices.

→ Solution

EXERCISE 3

Walk through this chapter's own five-step transaction flow for a real Bitcoin payment, naming specifically which earlier chapter's own material each step relies on (Chapter 2, 3, 4, or 5).

→ Solution

Quick Reference

- **A wallet stores keys, not coins** — the balance lives on the blockchain itself.
- **Seed phrase** — a 12–24-word backup that deterministically regenerates every key an HD wallet derives.
- **Hot vs. cold** — online-connected vs. offline key storage.

- **Custodial vs. non-custodial** — a third party holds your keys vs. you hold them directly; "not your keys, not your coins."
- **Mt. Gox** — a real 2014 exchange collapse, roughly 650,000 BTC net lost after theft from its own hot wallet, the classic real illustration of custodial risk.
- **Transaction flow** — construct → sign (Ch.2) → broadcast → included in a block (Ch.4) → confirmed as more blocks build on top (Ch.3, Ch.4).

CHAPTER 8: TOKENS & NFTS: FUNGIBLE VS. NON-FUNGIBLE

Blockchain & Web3 Fundamentals

Course 1 · Chapter 8 · Tokens & NFTs: Fungible vs. Non-Fungible

*Chapter 6 introduced Ethereum's Contract Accounts — accounts controlled by code instead of a private key. This chapter looks at the single most common thing that code is actually used for: creating **tokens**, both the everyday fungible kind and the unique, non-fungible kind (NFTs) that made real, worldwide headlines in 2021.*

Fungible vs. Non-Fungible, Precisely

Fungible means interchangeable: any one unit is identical in value and function to any other unit of the same type. A £10 note is fungible — you don't care which specific £10 note you receive in change, since they're all equally good. ETH and BTC are both fungible in exactly this sense: 1 ETH is always worth exactly 1 ETH, regardless of which specific 1 ETH it is.

Non-fungible means the opposite: each unit is unique and not directly interchangeable with another, even of the "same" type. A specific house, a specific piece of original art, or a specific collectible card is non-fungible — substituting a different one, even a similar-looking one, genuinely isn't the same thing.

ERC-20: The Real Standard Behind Fungible Tokens

A "token" on Ethereum isn't some separate kind of blockchain object — it's simply a smart contract (Chapter 6) that keeps track of balances internally and implements a shared, standardized set of functions so every wallet, exchange, and application can interact with it the same way. The real, foundational standard for fungible tokens is **ERC-20**, proposed by **Fabian Vogelsteller in November 2015**.

ERC-20 defines a common interface — functions like checking an account's token balance, transferring tokens between accounts, and reporting the token's total supply — that any compliant contract implements. Because the interface is standardized, a

wallet or exchange only has to understand ERC-20 *once* to work with literally thousands of genuinely different tokens built on top of it, from stablecoins to governance tokens issued by decentralized projects. This single, shared standard is a large part of why Ethereum's own token ecosystem grew as large and interoperable as it did.

ERC-721: The Real Standard Behind NFTs

Non-fungible tokens needed a genuinely different standard, since ERC-20's whole design assumes every unit is identical and interchangeable. **ERC-721**, authored by **William Entriken, Dieter Shirley, Jacob Evans, and Nastassia Sachs**, was formally published in 2018, though its development began in 2017 — directly motivated by one specific, real, and genuinely famous project.

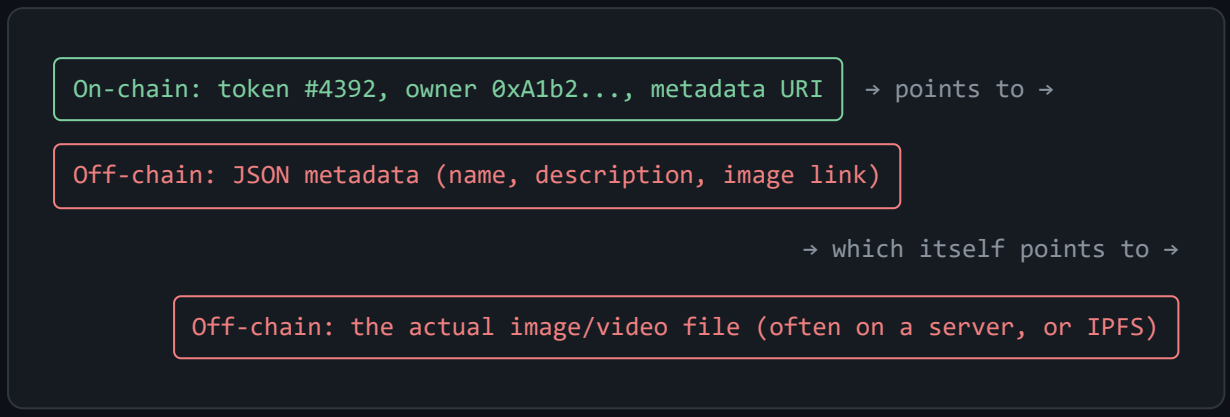
Late 2017	**CryptoKitties**, a game built around breeding and trading unique, individually distinct digital cats, launches and becomes an early, real-world demonstration of what a non-fungible token could be. It's credited with pioneering ERC-721's own real-world use.
Peak demand	CryptoKitties' own popularity became so large that it consumed, at its documented peak, up to **70% of Ethereum's entire network usage capacity** — genuinely congesting the network for everyone else, a real early demonstration of the scaling limitations Chapter 9 covers in depth.
2018	ERC-721 is formally published as a standard, generalizing the pattern CryptoKitties had already proven out.

What an NFT Actually Stores On-Chain

A genuinely common misconception is that an NFT *is* the digital image, video, or other media itself, stored directly on the blockchain. In practice, this usually isn't true. What an ERC-721 contract actually stores on-chain, per token, is:

- a unique **token ID**;

- the current **owner's address**; and
- typically a **URI** — essentially a link — pointing to *off-chain* metadata describing the token, which usually includes a further link to the actual media file itself.



A REAL, HONEST CONSEQUENCE OF THIS DESIGN

Because the actual media file usually lives off-chain, owning an NFT doesn't automatically guarantee the underlying image or file will remain accessible forever — if the off-chain server or storage hosting it ever goes down permanently, the on-chain token still exists (the ownership record itself is genuinely secure and unaffected), but the link it points to can break. Some projects mitigate this by storing metadata and media on IPFS (a distributed, content-addressed storage system) rather than a single centralized server, though even that isn't an automatic guarantee of permanence without someone continuing to host the data.

A Real, Concrete Landmark: Beeple at Christie's

NFTs moved from a niche, technical curiosity to mainstream global news in **March 2021**, when digital artist Beeple's work "*Everydays: The First 5000 Days*" sold at **Christie's** — one of the world's oldest and most established fine art auction houses — for a real, documented **$69.3 million**, paid in **42,329 ETH**. This remains, at time of writing, the most expensive NFT sale ever recorded, and a genuinely significant real moment for NFTs entering mainstream cultural and financial conversation far beyond the crypto community itself.

ONE MORE REAL STANDARD WORTH KNOWING: ERC-1155

A later real standard, **ERC-1155** (developed by the Enjin team), lets a single smart contract manage *both* fungible and non-fungible tokens together — useful, for example, in a game that needs both interchangeable in-game currency and unique collectible items within one contract, rather than deploying a separate ERC-20 contract and a separate ERC-721 contract for each.

Standard	What It's For	Real Origin
ERC-20	Fungible tokens (identical, interchangeable units)	Fabian Vogelsteller, November 2015
ERC-721	Non-fungible tokens (unique, non-interchangeable units)	Entriken, Shirley, Evans & Sachs, published 2018, motivated by CryptoKitties (2017)
ERC-1155	A single contract managing both fungible and non-fungible tokens together	Developed by Enjin

Hands-On Exercises

Three exercises reinforcing the fungible/non-fungible distinction and what an NFT actually stores, before Chapter 9 turns to the scaling problem CryptoKitties itself first exposed at real, network-wide scale.

EXERCISE 1

Classify each of the following as fungible or non-fungible, and briefly justify each answer: a £5 note, a signed first-edition book, a share of a publicly traded company, and a specific seat number for a specific concert.

→ Solution

EXERCISE 2

A friend buys an NFT and says: "I now own this image file, and it's permanently stored on the blockchain forever." Using this chapter's own explanation of what ERC-721 actually stores on-chain, correct this claim.

→ Solution

EXERCISE 3

Explain why CryptoKitties consuming up to 70% of Ethereum's network capacity is directly relevant to a topic Chapter 9 hasn't covered yet — what real, practical problem does this event foreshadow?

→ Solution

Quick Reference

- **Fungible** — interchangeable units, all equally valuable (ETH, BTC, most currencies).
- **Non-fungible** — unique, non-interchangeable units.
- **ERC-20** — the real, standard fungible-token interface, proposed by Fabian Vogelsteller in November 2015.
- **ERC-721** — the real NFT standard, published 2018, directly motivated by CryptoKitties (late 2017), which at its peak used up to 70% of Ethereum's network capacity.
- **What an NFT stores on-chain** — a token ID, an owner address, and typically a URI pointing to off-chain metadata/media, not the media file itself.
- **ERC-1155** — a real standard (Enjin) letting one contract manage both fungible and non-fungible tokens together.
- **Beeple at Christie's** — "Everydays: The First 5000 Days" sold for $69.3 million (42,329 ETH) in March 2021, the most expensive NFT sale on record.

CHAPTER 9: SCALING & LAYER 2 SOLUTIONS

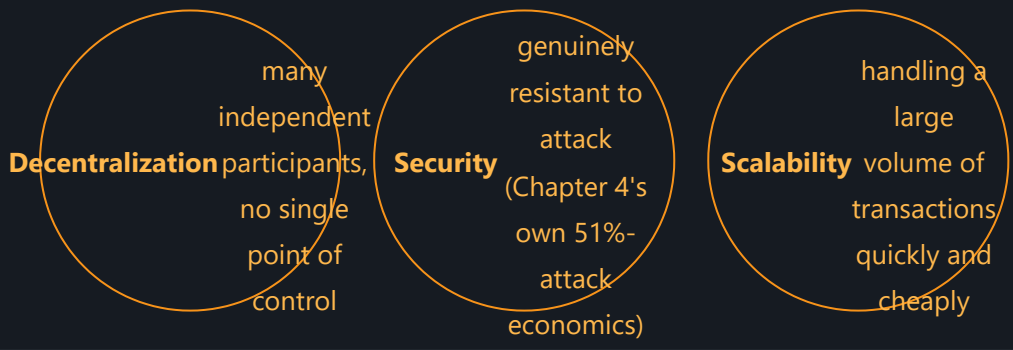
Blockchain & Web3 Fundamentals

Course 1 · Chapter 9 · Scaling & Layer 2 Solutions

Chapter 8 closed on a real, concrete demonstration of a limit: CryptoKitties alone, at its peak, consumed up to 70% of Ethereum's entire network capacity. This chapter explains precisely why a base blockchain has so little headroom to spare in the first place, and covers the real, working approaches the industry has built to scale far beyond what a single base chain can process alone.

The Scalability Trilemma

Blockchain designers face a genuine, well-known tension often described as the **scalability trilemma**: a blockchain design tends to be able to strongly deliver on only two of these three properties at once, without real, significant compromise on the third:



Bitcoin and Ethereum's own base layers, as covered so far in this course, both deliberately prioritize decentralization and security. That choice comes with a real, direct scalability cost.

The Real Throughput Gap

Bitcoin (base layer)	**~7 transactions/second**
Ethereum (base layer)	**a similarly low, commonly-cited figure in the tens of transactions/second**
Visa's own data centers (reported capacity)	**up to 30,000 simultaneous transactions**

Bitcoin's own real, documented base-layer throughput sits at roughly 7 transactions per second — a direct, structural consequence of Chapter 3's own fixed block size and Chapter 4's own roughly-10-minute block interval. Visa's own published figures describe its data centers as capable of handling up to 30,000 simultaneous transactions — a real, order-of-magnitude reminder of just how much headroom a base blockchain layer alone doesn't have if it needs to compete with existing centralized payment infrastructure at real-world scale.

WHY THE NETWORK CAN'T JUST "MAKE BLOCKS BIGGER"

It's tempting to assume the fix is simple — just allow bigger blocks, or shorter block times. But per the trilemma above, doing this at the base layer has real costs: bigger or more frequent blocks mean more data every single node has to download, store, and verify to fully participate, which tends to push out smaller, less-resourced participants over time — a genuine, direct erosion of decentralization, not a free lunch.

Layer 1 vs. Layer 2

Layer 1 refers to the base blockchain itself — Bitcoin's or Ethereum's own chain, with all the security properties this course has built up over the last eight chapters. **Layer 2** refers to a separate system built *on top of* a Layer 1 chain, designed to handle the bulk of everyday transaction volume off the base chain, while still ultimately relying on the Layer 1 chain underneath for its own final security guarantees.

Payment Channels: Bitcoin's Lightning Network

Bitcoin's real, working Layer 2 solution is the **Lightning Network**, first described in a whitepaper by Joseph Poon and Thaddeus Dryja in February 2015, and actually launched on Bitcoin's mainnet in **2018**. It works through **payment channels**: two parties lock up funds together in a shared, multi-signature on-chain transaction, then conduct any number of instant, effectively free transactions between themselves entirely off-chain, simply updating the private, agreed-upon split of the locked funds each time. Only when the channel is finally closed does one final, settled transaction get broadcast to the actual Bitcoin blockchain.

A network of many interconnected channels lets payments route between people who've never directly opened a channel with each other, hopping across a path of existing channels. This real network has grown substantially: as of recent measurements, it spans roughly **17,000 public nodes** and around **40,000 public channels**, with a total real capacity of roughly **4,900 BTC** locked up across the network.

Rollups: Ethereum's Primary Scaling Approach

Ethereum's own dominant real scaling strategy is built around **rollups**, which execute a large batch of transactions off the main Ethereum chain, then post a compact summary of the results back to Layer 1, inheriting Ethereum's own base-layer security for that summary. Two real, genuinely different approaches exist:

- **Optimistic rollups** — assume every batch of transactions is valid by default, and only actually run the underlying computation to verify it if someone specifically challenges the result during a set dispute window. This keeps everyday operation cheap, at the real cost of a withdrawal delay while that challenge window remains open.
- **Zero-knowledge (ZK) rollups** — compute the results off-chain and submit a real cryptographic proof, verified mathematically on Layer 1, that the batch was processed correctly — without needing to reveal or re-execute every individual transaction on the base chain. This avoids the optimistic rollup's own withdrawal-delay trade-off, at the real cost of more complex cryptography to implement correctly.

SIDECHAINS: A RELATED, BUT DISTINCT, APPROACH

A **sidechain** is a separate blockchain, with its own independent consensus mechanism, connected to a Layer 1 chain via a bridge that lets assets move between the two. Unlike a rollup, a sidechain doesn't inherit Layer 1's own security directly — it has to secure itself independently, which is a genuine, real trade-off worth knowing rather than assuming every "Layer 2"-branded solution provides an identical security guarantee.

Approach	Real Example	Real, Honest Trade-off
Payment channels	Lightning Network (Bitcoin, 2018)	Requires locking up capital in advance and actively managing channels
Optimistic rollups	Ethereum ecosystem rollups	Cheap, simple; withdrawals delayed by a dispute/challenge window
ZK rollups	Ethereum ecosystem rollups	Fast finality, no challenge window; more complex cryptography to build correctly
Sidechains	Various bridged chains	Own independent security model — doesn't inherit Layer 1 security automatically

TYING THIS BACK TO THE TRILEMMA

Layer 2 solutions are, at their core, a genuinely clever way to partially sidestep the trilemma rather than eliminate it: keep Layer 1 conservative, decentralized, and secure exactly as this course has described, while moving the bulk of everyday transaction volume to a separate layer specifically optimized for throughput — rather than compromising the base layer's own decentralization or security to get there directly.

Hands-On Exercises

Three exercises reinforcing the scalability trilemma and real Layer 2 mechanics, before Chapter 10's capstone traces one real transaction all the way through everything this

course has covered.

EXERCISE 1

A colleague proposes: "Just increase Bitcoin's block size and shorten the block interval — problem solved." Using this chapter's own scalability trilemma, explain the real trade-off this proposal doesn't account for.

→ Solution

EXERCISE 2

Explain, in your own words, how two people who have never directly opened a Lightning Network channel with each other can still send a payment to one another through the network.

→ Solution

EXERCISE 3

Compare optimistic rollups and ZK rollups on exactly one dimension: how each one actually proves a batch of transactions was processed correctly, and what real cost each approach accepts in exchange for its own method.

→ Solution

Quick Reference

- **Scalability trilemma** — decentralization, security, and scalability are genuinely hard to maximize all three at once.
- **Real throughput** — Bitcoin's base layer processes roughly 7 transactions/second; Visa's own data centers report handling up to 30,000 simultaneously.
- **Layer 1 vs. Layer 2** — the base chain itself vs. a system built on top of it to offload everyday transaction volume.
- **Lightning Network** — Bitcoin's real payment-channel Layer 2, launched 2018, letting parties transact off-chain and settle only occasionally on-chain.

- **Optimistic rollups** — assume validity by default, verified only if challenged within a dispute window; cheap, but delayed withdrawals.
- **ZK rollups** — submit a cryptographic proof of correct execution; fast finality, but more complex to build.
- **Sidechains** — separate, independently-secured chains bridged to Layer 1, not automatically inheriting its security.

CHAPTER 10: CAPSTONE — TRACING A REAL TRANSACTION END TO END

Blockchain & Web3 Fundamentals

Course 1 · Chapter 10 · Capstone: Tracing a Real Transaction End to End

Nine chapters built up every real piece of the picture separately: the trust problem, hashing and signatures, blocks and Merkle trees, consensus, Bitcoin's own UTXO mechanics, Ethereum's genuinely different account model, wallets and keys, tokens and NFTs, and scaling. This capstone puts all of it together, tracing one single, complete payment — Alice sending Bob 0.5 BTC — from the moment Alice decides to pay through final, deeply-confirmed settlement, applying every prior chapter's own real mechanism in the order it actually happens.

The Scenario

Alice owes Bob 0.5 BTC for some freelance work. They've never met, don't share a bank, and don't want to involve one. Alice's wallet currently holds a single UTXO worth 0.62 BTC.

1. **Why no bank is needed at all**

   Alice and Bob are exactly the two mutually distrusting strangers Chapter 1 opened with. Without blockchain, they'd need a trusted third party — a bank, a payment processor — to prevent Alice from double-spending the same 0.5 BTC elsewhere. Bitcoin's whole design exists so they don't need one.

   CHAPTER 1

2. **Alice's wallet holds the key, not the coin**

   Alice's 0.62 BTC UTXO is really just a ledger entry on the chain itself, locked to her own public key. Her wallet, deterministically derived from a single backed-up seed phrase, is what lets her produce a valid signature for it whenever she chooses to spend it.

   CHAPTER 7

3. **Building and signing the transaction**

The wallet constructs a transaction consuming the whole 0.62 BTC UTXO as its input, and creates two outputs: 0.5 BTC to Bob's address, and 0.1199 BTC "change" back to Alice (with 0.0001 BTC left over as the miner's fee). Alice's wallet then signs the transaction with her private key, using ECDSA over the secp256k1 curve.

CHAPTER 5 · CHAPTER 2

4 **Broadcast and independent verification**

The signed transaction propagates across the network. Every node that receives it independently checks the signature against Alice's public key before relaying it further — nobody needs to trust the node that told them about it; they can verify it themselves.

CHAPTER 2

5 **Included in a block, hashed into a Merkle tree**

A miner picks up Alice's transaction, along with thousands of others, and includes it in a candidate block. Every transaction in that block is hashed into a Merkle tree; the single resulting Merkle root gets stored in the block's own header, letting anyone later prove Alice's transaction is really in this block using only a short handful of sibling hashes.

CHAPTER 3

6 **Mining: finding a valid nonce**

The miner repeatedly hashes the candidate block's header with SHA-256, trying different nonce values, until the result falls below the network's current difficulty target — a target recalibrated every 2016 blocks to keep the real average block time near 10 minutes regardless of total network mining power.

CHAPTER 4

7 **Confirmation deepens over time**

The block is broadcast and accepted by the network. As more blocks are mined on top of it, Alice's payment to Bob becomes progressively harder to reverse — an attacker would need to out-mine the entire honest network's own accumulated proof-of-work since that block, an economically extreme proposition per Chapter 4's own real cost estimates.

CHAPTER 3 · CHAPTER 4

8 **Bob now owns a new, independent UTXO**

Bob's own real balance is simply whatever total his wallet finds by scanning the chain for UTXOs locked to his public key — and it now includes a fresh 0.5 BTC UTXO from Alice. No bank updated a number anywhere; the ledger itself is the update.

CHAPTER 5

9 **What if Alice and Bob paid each other constantly?**

If this were one of many frequent, small payments between the same two people, broadcasting every single one to the base chain would be unnecessarily slow and costly at real network scale. They could instead open a Lightning Network payment channel, transact instantly and privately off-chain as many times as they like, and only settle the final balance to the base chain once, when they eventually close the channel.

CHAPTER 9

Meanwhile, on Ethereum: the Same Payment, a Genuinely Different Mechanism

If Alice instead wanted to pay Bob using an ERC-20 token on Ethereum rather than bitcoin itself, several real, structural pieces of this story would change:

- There's no UTXO to consume at all. Alice's balance is simply a number tracked inside the token's own smart contract state, associated with her Externally Owned Account (Chapter 6).
- Sending the payment means calling the token contract's own standardized `transfer` function (Chapter 8's own ERC-20 interface), which directly decreases Alice's tracked balance and increases Bob's, rather than creating and consuming discrete output objects.
- Alice still signs the transaction with her own private key (Chapter 2 applies identically here — ECDSA/secp256k1 signing isn't Bitcoin-specific), and still pays a fee — gas, in Ethereum's case — for the computation the network performs on her behalf.
- If Bob were instead receiving a unique collectible rather than an interchangeable currency amount, the same account-based transfer mechanism would apply, but

it would update an NFT's own recorded owner address (Chapter 8's ERC-721) rather than a balance number.

THE REAL TAKEAWAY

Bitcoin and Ethereum solve the exact same underlying problem — letting two people who don't trust each other exchange value without a bank — using two genuinely different accounting models (UTXOs vs. accounts). Neither is simply "the same thing with different names"; this course has deliberately built up both, chapter by chapter, precisely so the real differences are visible rather than assumed away.

Chapter-by-Chapter Attribution

Chapter	What It Contributed to This Capstone
1	Why Alice and Bob need no trusted third party at all
2	SHA-256 hashing; ECDSA signing and independent verification
3	The block header, the Merkle tree, and the chaining that deepens confirmation over time
4	Mining, difficulty adjustment, and why deeper confirmations are harder to reverse
5	Consuming Alice's UTXO, creating Bob's new one and Alice's change
6	The account-based alternative underlying the Ethereum comparison
7	What Alice's wallet actually stores and does
8	ERC-20 and ERC-721 as the Ethereum-side equivalents of this same payment
9	Why a Lightning Network channel would suit frequent, repeated payments better

WHAT THIS COURSE DOESN'T COVER

This capstone deliberately stops at the conceptual level. It doesn't write or deploy a single line of Solidity, doesn't touch decentralized finance mechanics

(lending, liquidity pools, automated market makers), doesn't cover smart contract security vulnerabilities or real, documented exploits, and doesn't address the genuinely important regulatory and custody questions around holding and using crypto assets in practice. All of that is deliberately reserved for Course 2, **Smart Contracts, DeFi & Web3 Security**, which builds directly on everything established here.

Hands-On Exercises

Three closing exercises applying the full nine-chapter toolkit to fresh scenarios, before moving on to Course 2.

EXERCISE 1

Rewrite this chapter's own Alice-to-Bob scenario for a payment of exactly 0.62 BTC (Alice's entire UTXO, with no change left over except the fee). Which specific output from Step 3 of the flow disappears, and why?

→ Solution

EXERCISE 2

A colleague asks: "If Alice and Bob's payment is confirmed after Step 7, why would anyone ever wait for more than one confirmation?" Using this chapter's own reasoning (and Chapter 4's own 51%-attack cost economics), write an accurate answer.

→ Solution

EXERCISE 3

Using this chapter's own Ethereum comparison, explain specifically what would need to happen differently if Bob's payment were 0.5 BTC worth of a fungible ERC-20 token instead of native ETH — is there still a "consuming a UTXO" step anywhere in that version of the story?

→ Solution

Quick Reference — Course Complete

- **Chapters 1–10** — the trust problem, hashing and signatures, blocks and Merkle trees, consensus (PoW/PoS), Bitcoin's UTXO model, Ethereum's account model and smart contracts, wallets and keys, tokens and NFTs, scaling and Layer 2, and this capstone tying every piece together.
- **Course complete** — Blockchain & Web3 Fundamentals is now finished, 10/10 chapters.
- **Next** — Course 2, *Smart Contracts, DeFi & Web3 Security*, builds directly on this course: writing real Solidity, understanding gas and the EVM in depth, DeFi mechanics, common vulnerabilities, real documented exploits, and the genuine risks and regulatory questions around Web3 in practice.