Understanding EUTXO — The Fundamentals

A SIMPLE INTRODUCTION TO EUTXO · IN SIX PARTS

An original, self-contained introduction to how EUTXO works — from first principles to the present. Nothing is taken on authority: every claim is checked against the evidence, every trade-off entered into the record.

Follow the main text; builders follow the **dev notes**. Sources & lineage at the close.

THE PARTS

part 01	The two ledgers	the model decides everything
part 02	The cash that cannot be torn	notes, not totals
part 03	The ledger that rewrites itself	a moving target
part 04	The lock that learned to see	two additions, one refusal
part 05	The honest trade-off	the same fingerprint
part 06	Four proposals, verified	promises, checked, kept
part 07	Glossary	every term, in one place

COLOR KEY — USED IN EVERY DIAGRAM

coral · spent / consumed blue · unspent output teal · validated / EUTXO

cream · wallet

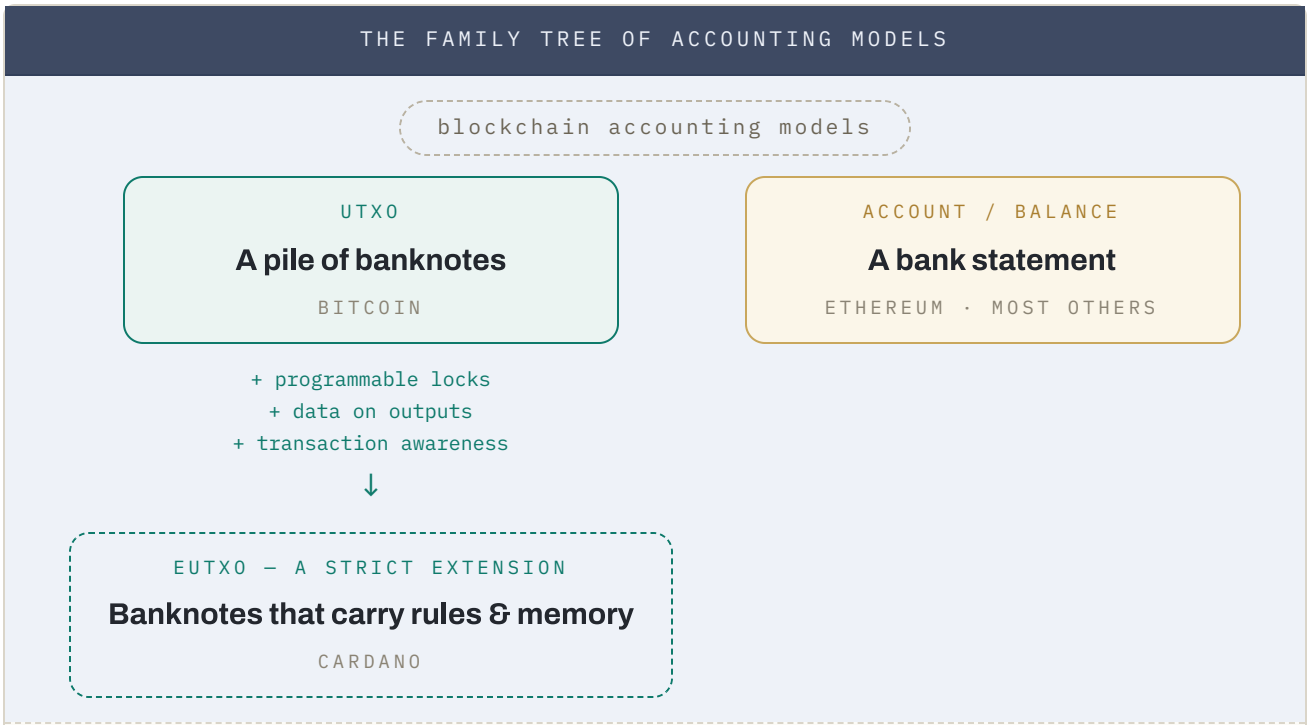
PART 01

The Two Ledgers

> **The scene:** *a shared record of money that thousands of strangers keep in perfect agreement — with no bank, no referee, and no one in charge. Before asking how, ask the sharper question: agreement about what, exactly?*

Picture a shared spreadsheet tracking who owns how much. Anyone can read it. Nobody owns it. Thousands of computers hold identical copies. The difficulty was never storage — it's **agreement without a referee**. There is no bank to call when two copies disagree, so the rules must be precise enough that every honest computer independently reaches the same verdict on every transaction.

Those rules are the **accounting model**: the exact way the ledger represents "who owns what" and how a transaction changes it. Interview any blockchain and you'll find one of two designs behind it — and, in one important case, a third that descends from the first. Lay them out on the board:



WHAT THIS SHOWS — EUTXO belongs to the UTXO family, but it is not the same model: it descends from Bitcoin's UTXO and has been formally proven strictly more expressive. Parts 02 and 04 establish exactly what changed — and what deliberately did not.

TAKEAWAY — the model decides everything. Every strength and every flaw we uncover from here traces back to which of these three designs a chain chose.

PART 02

The Cash That Cannot Be Torn

> **The question before us:** *a wallet displays "50." Search the entire ledger for that number beside the owner's name. You will not find it. So where does the 50 come from — and what, precisely, is on the ledger?*

Start with the name, read backwards. **UTXO — Unspent Transaction Output**. An **output** is a chunk of value a transaction produced, addressed to someone. The **transaction** is the event that consumed old chunks and created new ones. **Unspent** means this chunk hasn't yet been used as an input to anything later.

A UTXO, then, is a chunk of money sitting on the ledger with your name on the lock, waiting to be spent. That's the entire concept. Now watch it behave.

Exhibit A — the note that must be spent whole

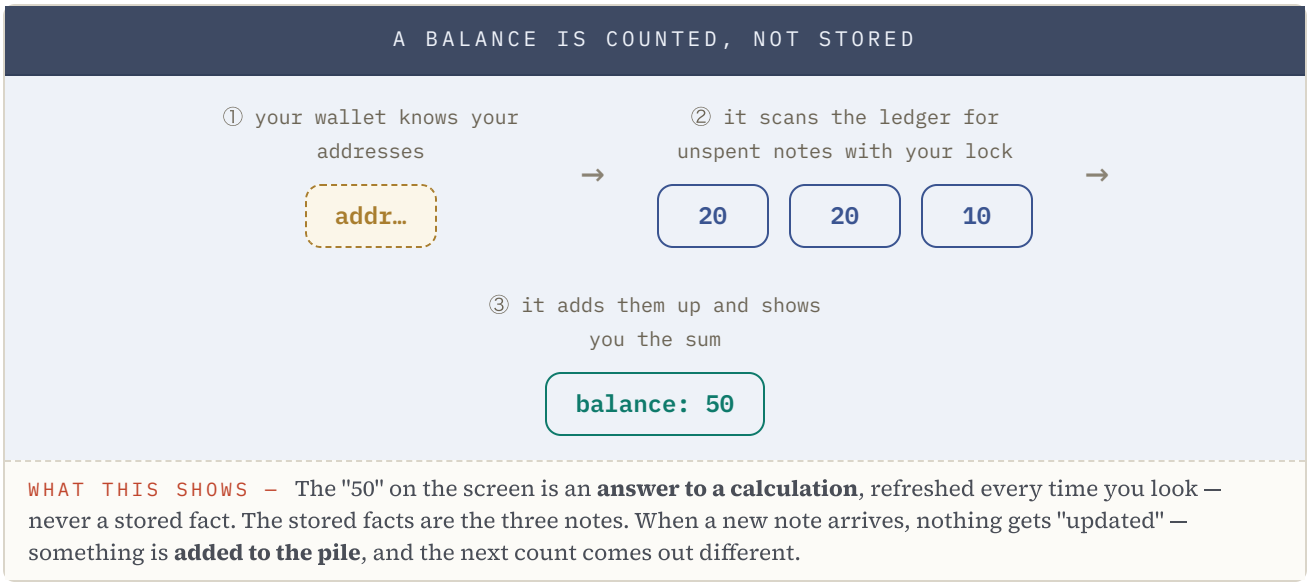
You hold a $50 bill and want a $15 coffee. You cannot tear the bill; you hand over the whole thing and receive change. In this model that's not an analogy — it is mechanically what happens:

KEY IDEA NO. 1 — COMMIT THIS TO MEMORY

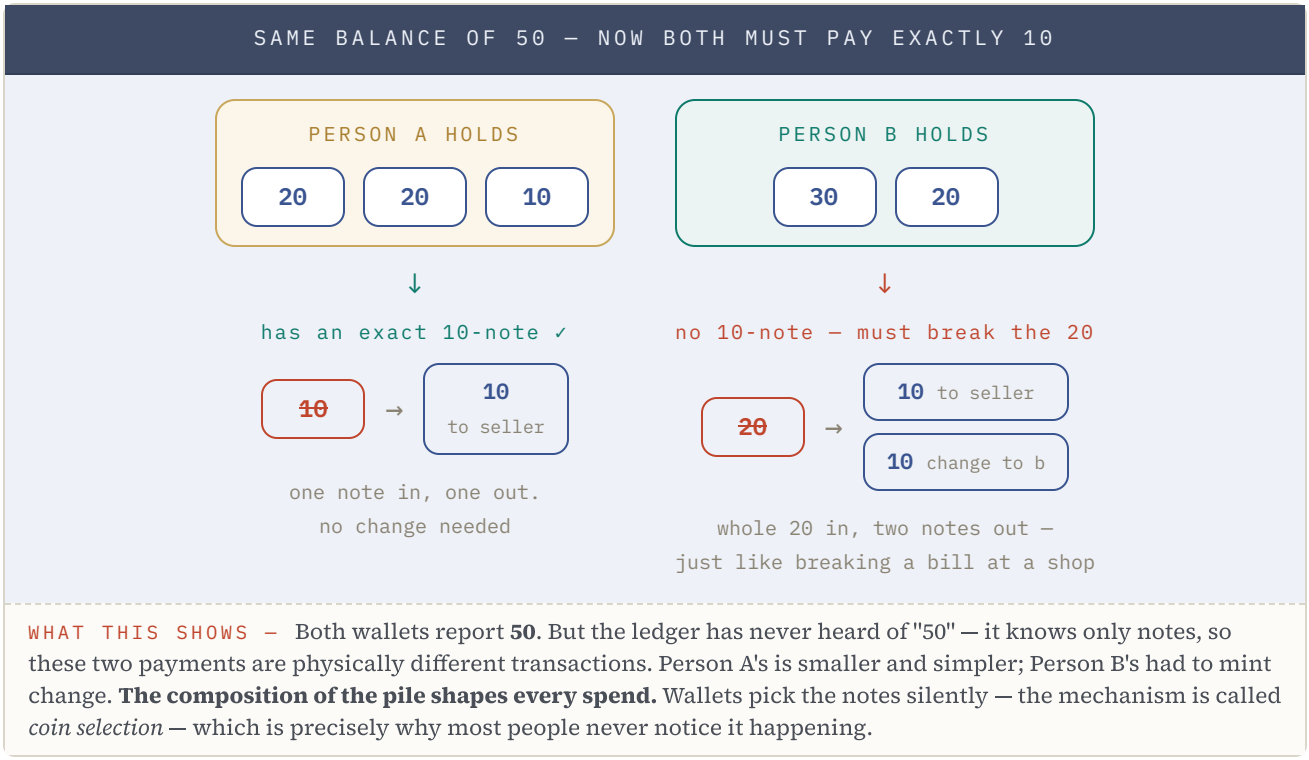
A UTXO can be spent **exactly once**, and only **in full**. There is no partial spend. Every strange behavior and every power we uncover in this casebook traces back to this single rule.

Exhibit B — the balance that was never there

Return to the opening question: where does the 50 live? In a bank, the answer is boring — the bank stores a number beside your name and the app displays it. Most people assume a crypto wallet works the same way. **In a UTXO system, it doesn't.** Search the ledger all you like: no number with your name on it exists anywhere. What exists is the pile of notes — and your wallet **counts them**, every time, like counting the cash in a physical wallet:



Once you see the balance as a count, a second question follows: does it matter *which* notes make the total? In a bank account, obviously not — 50 is 50. Here, **yes**, because spending means handing over whole notes. Run the experiment: two people, identical balances, both ordered to pay exactly 10.



Hold that finding — **the ledger tracks notes, never totals** — because it pays off twice. It's why UTXO chains can validate many transactions in parallel (different notes cannot collide, Part 04), and it's why two hundred people cannot all spend the *same* note at once (the trade-off, Part 05). The superpower and the headache share one fingerprint.

Exhibit C — locks, keys, and the chain of custody

An **output** contains an address and a value — the address is a **lock** on the box. An **input** is a pointer back to an earlier output plus a **key** that opens its lock: on Bitcoin, a digital signature matching the address's public key. When an input unlocks an output, the network marks it spent — and the chain of custody is literal. My output becomes your input; your output becomes another's input; follow the trail backwards and you reach the coin's creation. Provenance is exact, which is why this record never lies.

Step back far enough and the whole ledger has a recognizable shape: a **directed acyclic graph** — transactions are the nodes, outputs are the edges between them, and the arrows only ever point forward. Money flows through the graph; it never loops back.

Every node keeps a live index of every currently unspent output — the **UTXO set**, known in a node's technical internals as the **chainstate**, stored in its data directory and updated as each block arrives. It is, in a very real sense, the entire state of the blockchain.

> **DEV NOTE ⌁ UNDER THE HOOD**
>
> The UTXO set is the node's working state: an index keyed by `(tx_hash, output_index)`, mutated only by block application. Wallet **coin selection** — which notes to spend for a given payment — is a real algorithmic problem (minimize fees, avoid dust, preserve privacy), solved differently by every wallet. If a payment ever produces "too many small UTXOs," coin selection is why.

One side-benefit worth logging: because every note carries a visible size and age, an observer can read the chain's vital signs directly — how much value moves, how old the coins being spent are, how active the economy is. UTXO ledgers make unusually honest witnesses for analysts.

The verdict on plain UTXO — and the loose end

The design's merits are on record. **Simple to verify:** a transaction is valid or invalid based only on its own inputs. **Naturally parallel:** transactions touching different UTXOs cannot interfere, so nodes can check them simultaneously. **Auditable and privacy-friendly:** exact provenance, and a fresh address per output means activity doesn't cluster under one identity.

But push the lock further and it falls silent fast. A Bitcoin lock can essentially only say "open with a signature from this key." It sees nothing but the key being offered: it cannot read stored data (there is none), cannot inspect the rest of the transaction, and cannot say a word about where funds go next. So you cannot express "release after Tuesday" or "only if the machine still has stock." No rich conditions. No memory.

Fixing that without abandoning the cash model required a genuinely new model — not a patch. That's the subject of Part 04. But first, examine the rival.

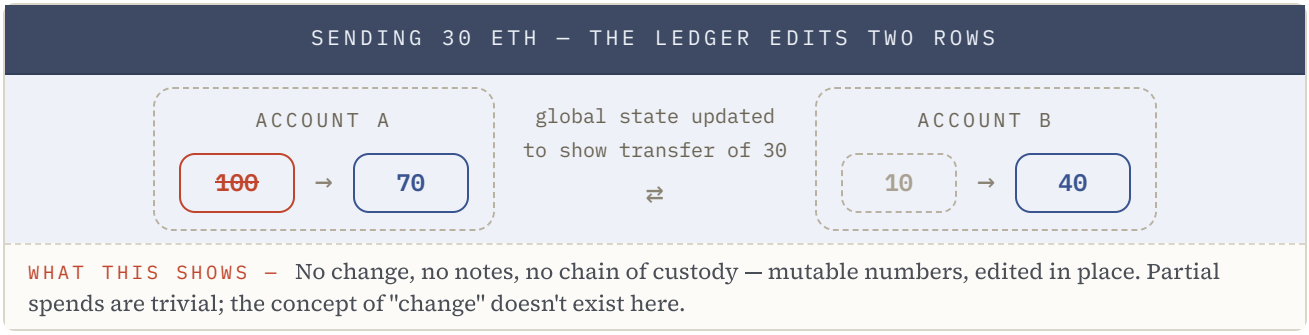
TAKEAWAY — notes, not totals. The lock is honest but mute — it can identify the bearer and nothing more.

PART 03

The Ledger That Rewrites Itself

> **The question before us:** *Ethereum chose the opposite design — and it demonstrably works, hosting the largest smart-contract economy in existence. So investigate fairly: what did the account model buy, and what is the recurring cost that keeps appearing in the incident reports?*

Each account — controlled by a private key or by a smart contract — holds a number. The network maintains a **global state**: one enormous table of every balance, held by every node, rewritten with every block.



WHAT THIS SHOWS — No change, no notes, no chain of custody — mutable numbers, edited in place. Partial spends are trivial; the concept of "change" doesn't exist here.

What the design buys: programmability feels natural. A smart contract is a long-lived object with persistent memory that anyone can call, so building a shared pool of anything — liquidity, votes, bids — is straightforward. This is a genuine advantage, and it's why DeFi emerged on Ethereum first. Enter that into the record without a sneer.

What it costs traces to one fact, and every incident report contains its fingerprint: *everything shares one mutable state, and that state changes underneath you between submitting a transaction and executing it.* Fees become a bid, not a price. Transactions can fail mid-execution — and still charge for the work already done. And whoever orders the block can insert themselves ahead of your trade: **front-running**, the visible edge of a multi-billion-dollar category called **MEV**.

NOTED FOR FAIRNESS — ADVOCACY USUALLY OMITS THIS

Ethereum has worked on these wounds: EIP-1559 made fees far more predictable, and private relays blunted the crudest front-running. The structural property remains — the state still moves under your feet — but the early framing reads harsher than today's reality. A good investigator doesn't flatter one side by exaggerating the other's record.

TAKEAWAY — a moving target. The account model's power and its unpredictability come from the same source — shared mutable state.

PART 04

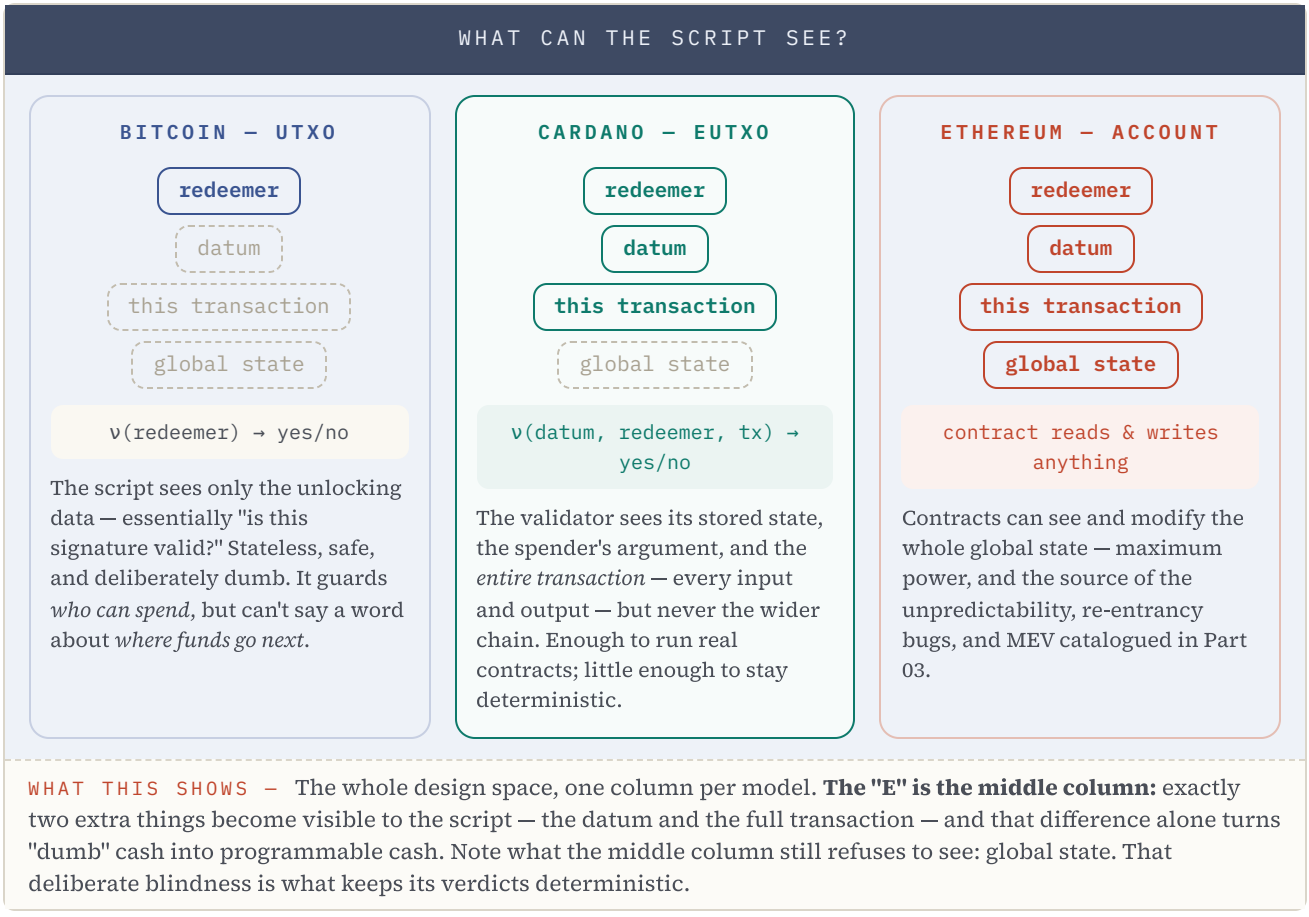
The Lock That Learned to See

> **The question before us:** *is it possible to have expressive contracts — Ethereum's power — while keeping the cash model's simplicity? Cardano's researchers claimed yes, and published proofs. Examine what, exactly, they changed. The answer is smaller than you'd guess: two things.*

KEY IDEA NO. 2 — ESTABLISH THE IDENTITY FIRST

UTXO and EUTXO are not the same model. EUTXO descends from UTXO the way a smartphone descends from a telephone — same family, different species. The formal research proves EUTXO is **strictly more expressive**: it can implement general state machines and enforce rules across entire chains of transactions, which plain UTXO provably cannot.

The cleanest way to separate them is a single question, put to all three models: **when the ledger runs your script, what is that script allowed to look at?** Their answers separate them completely:



State the bet precisely: keep UTXO's local, deterministic validation, and widen the script's field of view by **exactly two things** — no more. The extension goes in two directions.

> **DEV NOTE ^ THE FORMAL SHAPE**
>
> In the research paper's notation, a plain-UTXO validator is `v(redeemer) → Bool`; the EUTXO validator becomes `v(datum, redeemer, context) → Bool` — a **pure function**, no I/O, no clock, no chain reads. All three arguments arrive encoded in one uniform `Data` type (structured like CBOR/JSON); your contract decodes what it expects. Purity is the load-bearing property: it makes the off-chain "rehearsal" below provably identical to on-chain execution. **Version note** — since Plutus V3 (2024), the three arguments arrive as one merged `context` and the datum is optional, ending the stuck-funds failure mode. What the script can see is unchanged.

Extension one — the lock can be a program

Instead of a lock that only accepts a signature, an address can contain arbitrary logic: a **validator script**. When a transaction attempts to spend that output, the node runs the script. True → the spend is allowed. False → rejected. A lock can now hold a position: "open only if the price feed reads above X, and today is Wednesday, and two of these three signatures are present."

Extension two — the output can carry data

An EUTXO output holds an address, a value, and a **datum** — an arbitrary piece of data riding on the output itself. This is what gives contracts memory. In the account model, a contract keeps memory in variables; here, memory travels on the note. Spending an output and creating a successor with an updated datum is how state advances.

The overlooked clause — money itself becomes plural

There is a third change the headline "two extensions" undersells: **multi-assets**. In plain UTXO, an output holds one thing — an amount of the chain's single coin. In Cardano's ledger, an output holds a **bundle** of tokens: ada, any number of custom **native assets**, and **NFTs**, all riding in the same note under the same lock.

Register why "native" is doing real work in that sentence. On Ethereum, a token is not money to the ledger — it's a contract (the ERC-20 pattern): a program keeping its own internal spreadsheet of who owns what, with custom code that can be inefficient, expensive, or simply wrong. On Cardano, tokens live in outputs at the ledger level; sending one is an ordinary transaction requiring no contract at all, and every token inherits the ledger's own security by default. The one place code enters the picture is creation and destruction: a **minting policy** — a script, cousin to the validator — that alone decides when its tokens may be minted or burned. (An NFT is just a minting policy that permits exactly one token, ever.)

The formal record backs the design up: the follow-up research paper proved the multi-asset EUTXO ledger strictly more expressive than single-currency EUTXO — a second rung on the same ladder. UTXO, then EUTXO, then multi-asset EUTXO: each provably capable of things its parent is not.

The delta, itemized

Everything established so far, laid on one card — exactly what changed between parent and descendant, and what deliberately did not:

THE ITEM	UTXO (BITCOIN)	EUTXO (CARDANO)
The lock	A public key — "show me a signature"	Any program (a validator) — arbitrary conditions
What the script sees	The redeemer only	Datum + redeemer + the entire transaction (context)
Memory	None — an output is value and a lock, nothing more	A datum rides on every output; state advances by retire-and-recreate
What an output holds	An amount of the one native coin	A token bundle — ada, native assets, NFTs — under one lock
Time	Effectively none — the script is blind to the clock	A validity interval: bounded, determinism-safe access to time
Proven expressiveness	Payments and simple spending conditions	General state machines; invariants enforced across whole transaction chains
Unchanged, on purpose	Spend-once, spend-whole, local validation, natural parallelism, the note-based ledger — the entire cash chassis carries over intact. *That's what makes it an extension and not a replacement.*	

The four instruments

VALIDATOR

The lock

A program that decides whether a spend is allowed. Returns true → spend goes through. False → rejected.

REDEEMER

The key — and the plea

Data the spender supplies to argue their case — "I'm swapping 100 ada," "I'm claiming the refund branch."

DATUM

The memory

Contract-specific state stored on the output — a price, an owner, a deadline, a score. Spending an output and recreating it with a new datum is how state advances.

CONTEXT

The witness statement

A summary of the whole transaction being validated — inputs, outputs, signatures, validity window — so a script can enforce conditions on the transaction as a whole.

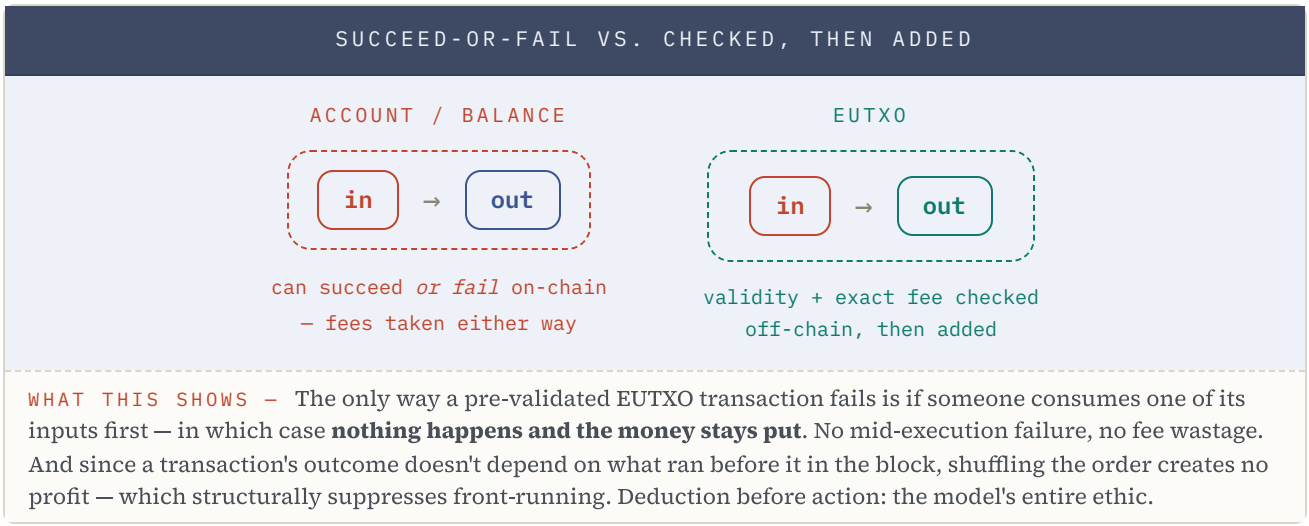
> DEV NOTE ~ ADDRESSES & DATUMS IN PRACTICE
>
> A script address is derived from the **hash of the validator** — the code is the identity; change one byte and you have a different address. Datums were originally stored as hashes on the output (the spender supplied the preimage); since Vasil's CIP-32 they can sit inline, readable by anyone. Part 06 has the before/after.

Context is quietly the decisive instrument. Because a validator can inspect the transaction's *outputs*, it can insist its successor carries the correct datum and the correct script. That property is called **continuity** — it makes "spend the contract and simply not recreate it" an impossible crime. The lock reads the whole confession before it opens.

Time is handled with an investigator's caution. Scripts cannot ask "what time is it" — the answer would differ depending on when the script runs, and determinism would collapse. Instead, a transaction declares a **validity interval**: a window of slots during which it may be included. A script may assume "now" falls inside the window without knowing exactly where. Deadlines become expressible; determinism survives.

Why determinism is the whole point

A validator's verdict depends only on the transaction and its inputs — never on the wider world. So you can run the exact validation logic on your own machine *before* broadcasting anything. Compare the two procedures side by side:



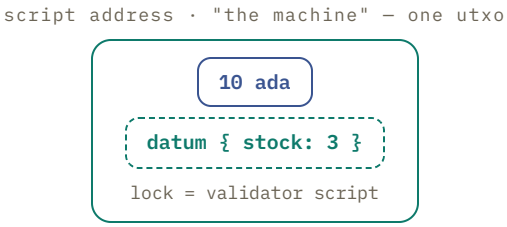
WHAT THIS SHOWS — The only way a pre-validated EUTXO transaction fails is if someone consumes one of its inputs first — in which case **nothing happens and the money stays put**. No mid-execution failure, no fee wastage. And since a transaction's outcome doesn't depend on what ran before it in the block, shuffling the order creates no profit — which structurally suppresses front-running. Deduction before action: the model's entire ethic.

Watch one contract's full cycle

The four instruments are easiest to grasp in motion. What follows is a walkthrough: a tiny vending-machine contract walked through one complete purchase, step by step. (The vending machine is the classic teaching example from the EUTXO literature — a locked output whose datum is the inventory.)

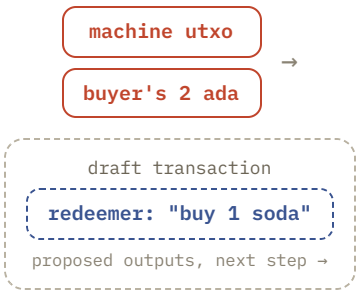
WALKTHROUGH · THE VENDING MACHINE — STEPS 1–3

STEP 1 — THE CONTRACT AT REST



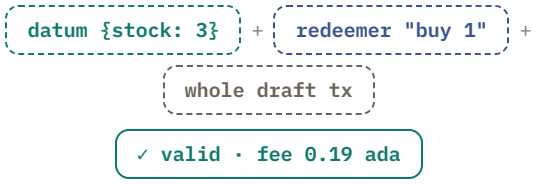
A "smart contract" on Cardano is not a running program. It's **an output sitting still**: some value, a datum holding its state (3 sodas in stock), and a validator as its lock. Nothing happens until someone tries to spend it.

STEP 2 — A BUYER BUILDS A TRANSACTION, OFF-CHAIN



The buyer's wallet does the work: it selects the machine's UTXO and 2 ada of the buyer's own money as inputs, and attaches a **redeemer** — the argument for why this spend should be allowed. All of this happens on the buyer's computer. The chain hasn't heard a thing yet.

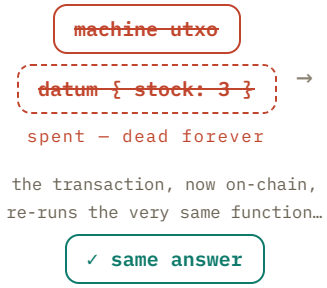
STEP 3 — REHEARSAL: THE VALIDATOR RUNS LOCALLY



Determinism at work. The validator is a pure function of exactly three things — datum, redeemer, transaction — so the wallet can run it **before submitting** and learn the outcome and the exact fee. If the answer were ✕, the buyer walks away having spent nothing.

WALKTHROUGH · THE VENDING MACHINE — STEPS 4–6

STEP 4 — SUBMITTED: THE OLD OUTPUT IS CONSUMED



On-chain, nodes run the identical pure function on identical inputs and get the identical answer — that's why the rehearsal was trustworthy. The old machine UTXO with stock 3 is now marked **spent**. State is never edited in EUTXO; it is **retired**.

STEP 5 — CONTINUITY: THE SUCCESSOR IS BORN

Because the validator could inspect the transaction's **outputs** (that's the context), it refused to approve any transaction that didn't recreate the machine correctly: value up by 2, stock down by 1, same lock. That guarantee is **continuity** — you cannot spend the contract and "forget" to put it back.

STEP 6 — THE WALKTHROUGH, COMPLETE

State advanced not by mutation but by a **chain of retire-and-recreate steps** — a state machine whose every transition is a transaction. That is the entire EUTXO idea in one sentence. Replay it, or proceed to Part 05, where the model reveals what this design costs.

Plutus — and where the real work happens

The on-chain language is **Plutus Core** — small, functional, built for formal reasoning. Almost nobody writes it by hand: historically you wrote Haskell and a compiler plug-in produced it. Since the early years, the barrier has dropped considerably — **Aiken** is now a popular purpose-built contract language, with TypeScript and Python paths for the off-chain side.

And here's a detail worth underlining twice in your notes: on Cardano, **the off-chain code is most of the work**. The on-chain script only says yes or no; constructing the transaction — choosing UTXOs, computing outputs, assembling datums — happens in your application. This is the biggest adjustment for developers arriving from Ethereum, and the source of most first-year confusion. (On-chain, scripts appear in two roles: **validators** locking outputs, and **minting policies** governing tokens — same machinery, different post.)

EXAMINED — CLAIMS OFTEN BUNDLED WITH EUTXO ADVOCACY

Promotional summaries of EUTXO tend to close with a box of superiority claims, and a closer look finds them mixing claims. "More concurrent transactions on a proof-of-stake system drawing a tiny fraction of proof-of-work's energy" — the energy claim is true, but it's a consensus-layer fact about Ouroboros vs. mining, *not* a property of the EUTXO accounting model; a chain could run EUTXO on proof-of-work tomorrow. Likewise "decentralization and security improve over Bitcoin's" is a network claim, not a ledger-model claim. Such lists often cite the **UTXO Alliance** — IOG plus seven other UTXO-family chains collaborating on interoperability — which we log as a dated fact of uncertain current standing. The lesson for the record: when a source bundles unrelated favorable facts into one list, unbundle them before weighing any.

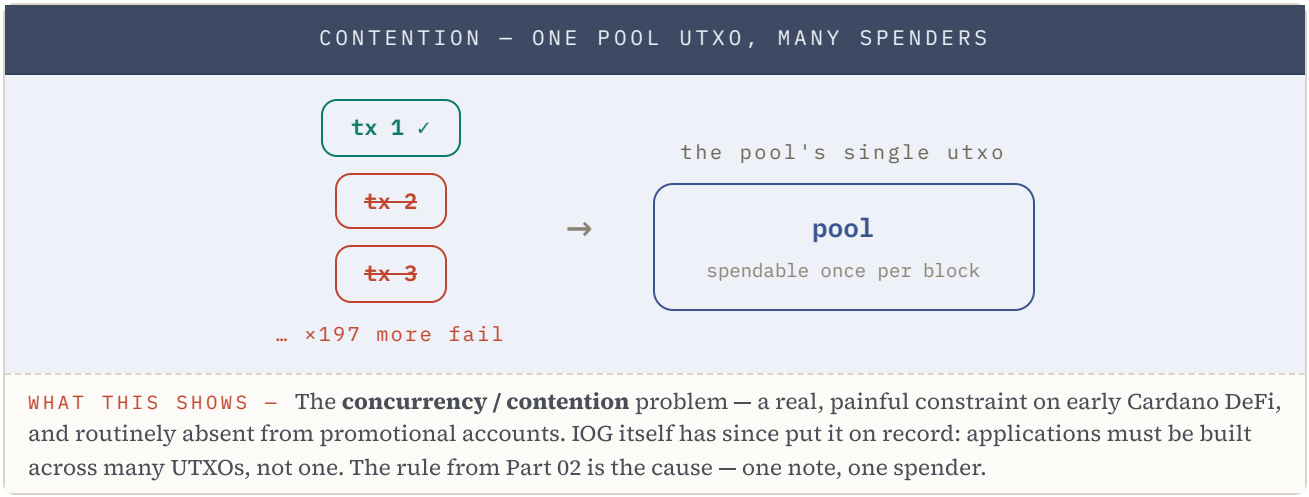
TAKEAWAY — two additions, one refusal. The lock gained memory and sight — and its refusal to see global state is precisely what makes its verdicts predictable.

PART 05

The Honest Trade-off

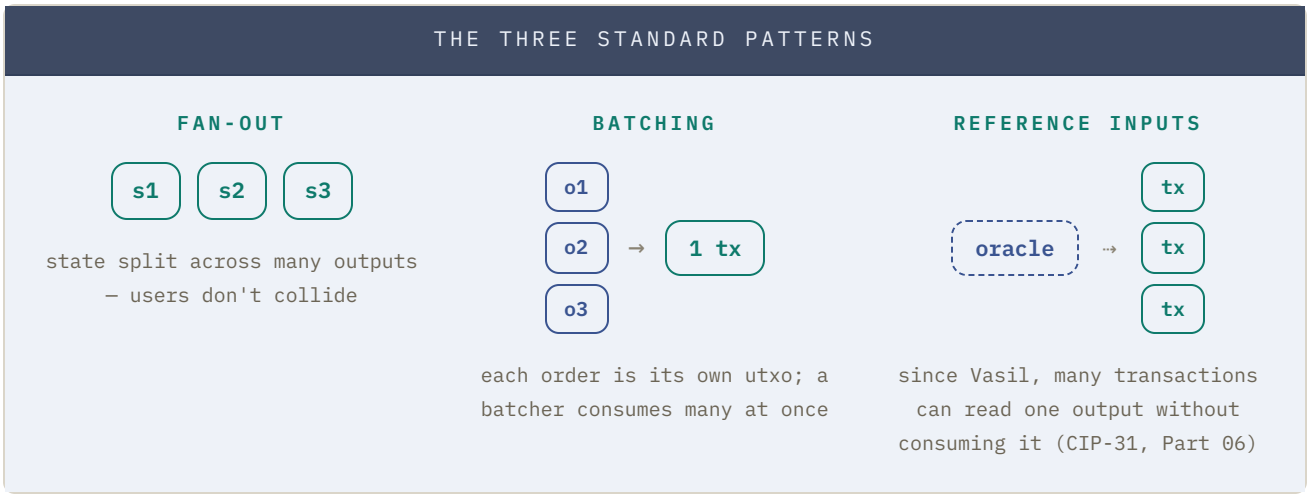
> **Every elegant theory has a weak point.** *Most tellings never record this one. We will. Recall Key Idea One: an output can be spent exactly once. Now put two hundred people in front of one output — and watch.*

Suppose a decentralized exchange keeps its liquidity pool in a single UTXO, and two hundred people want to trade against it in the same block. All two hundred build transactions consuming that same output. **Exactly one succeeds.** The other 199 fail — harmlessly, but they fail. On Ethereum this problem doesn't exist: two hundred calls to the same contract simply execute in sequence.



How the case was resolved

An application-design problem, not a protocol defect — and the patterns are now mature. Three techniques appear in every production system:



DEV NOTE ⌁ HOW PRODUCTION DEXS ACTUALLY RUN

The patterns above are not theory. Cardano's major AMMs (Minswap, SundaeSwap v2, WingRiders) run **batcher networks**: user orders sit as independent UTXOs until a batcher bot consumes a set of them plus the pool UTXO in one transaction. Genius Yield runs a pure **order-book**, where matching two orders touches only those two UTXOs. Research also exists on fully deterministic, decentralized batching that removes trust in the batcher operator — worth reading before you design your own.

THE TRADE-OFF, IN FULL

EUTXO trades ease-of-development for predictability. Deterministic fees, structural front-running resistance, parallel validation — paid for with harder application architecture and a dependency on batcher infrastructure. Batchers are off-chain actors, and who runs them is a question worth keeping open on your board: it carries its own centralization risk. Any framing of EUTXO as strictly superior isn't wrong so much as one-sided — a summary telling only the flattering half.

The verdict board

With the flattering half and the trade-off both on record, the full comparison can finally be scored. Read it column by column — and note that **no column sweeps the board**:

THE QUESTION	UTXO (BITCOIN)	EUTXO (CARDANO)	ACCOUNT (ETHEREUM)
Fee known before submitting?	Yes — exactly	Yes — exactly	A bid, not a price (EIP-1559 tamed it, didn't fix it)
Cost of a transaction that doesn't go through?	Zero	Zero	Gas already consumed is gone
Can a transaction fail mid-execution?	No	No — verdict knowable in rehearsal	Yes
Profit from reordering transactions (MEV)?	Structurally resistant	Structurally resistant	Endemic; a live industry (mitigations exist)
Parallel validation of independent transactions?	Natural	Natural	Limited — shared state forces sequencing
Expressive smart contracts?	No — dumb locks by design	Yes — proven state machines	Yes — the original home of them
Many users hitting one shared contract per block?	n/a	The hard part — needs fan-out / batching (Part 05)	Native strength — calls just queue
Custom tokens?	Not natively	Ledger-native bundles; no contract to trust	Per-token contracts (ERC-20) — powerful, error-prone
Developer on-ramp?	Simplest model to reason about	Steepest — off-chain-heavy, new patterns	Most familiar; largest tooling & talent pool

That is the whole case in one board. UTXO buys simplicity and predictability at the price of expressiveness. The account model buys expressiveness and developer ease at the price of predictability. EUTXO is the attempt to keep both — and its two red cells are the honest bill for the attempt: the hardest concurrency story and the steepest learning curve of the three. Any presentation of this model showing you only the teal cells, or only the coral ones, is telling you only half the story.

TAKEAWAY — the same fingerprint. Spend-once gives you parallelism and gives you contention. One rule, both consequences. Any account of the model that mentions only one is an incomplete account.

PART 06

Four Proposals, Verified

> **Not long ago, four improvement proposals were still promises on paper.** *A promise is not evidence. So we checked the record: did they ship, and did they do what was claimed?*

Verified: all four went live in the **Vasil hard fork**, alongside **diffusion pipelining**, a block-propagation speedup. The before/after record for each:

> **DEV NOTE ⌁ THE TWO VALIDATION PHASES**
>
> Cardano validates in two phases. **Phase 1:** cheap structural checks — inputs exist, signatures verify, fees suffice. **Phase 2:** script execution. Collateral exists to charge for phase-2 failures, which a correctly rehearsed transaction should never reach — CIP-40 below governs exactly how much a failure costs.

CIP-31 — Reference Inputs

BEFORE

To read data in a UTXO you had to spend it and recreate it — reading was as contentious as writing.

AFTER

Transactions can reference an output read-only. Oracles and price feeds became practical. The single biggest contention fix on this list.

CIP-32 — Inline Datums

BEFORE

Outputs stored only a hash of the datum; spenders had to separately supply matching data obtained off-chain.

AFTER

The datum itself sits on the output, visible to everyone. Simpler, and far better for transparency.

CIP-33 — Reference Scripts

BEFORE

Every transaction carried a full copy of the validator script — complex contracts meant enormous, expensive transactions.

AFTER

The script lives on-chain once; transactions point at it. Dramatically smaller transactions, lower fees, more throughput.

CIP-40 — Collateral Outputs

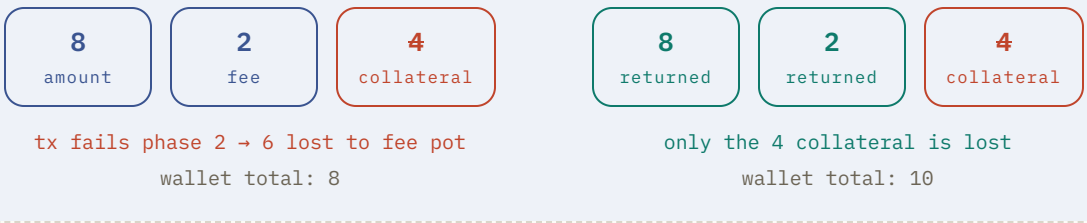
BEFORE

Fail phase-2 validation and your entire collateral input was consumed — post 4 ada from a 6 ada UTXO, lose all 6.

AFTER

The excess returns to a collateral output — you lose only the specified amount.

CIP-40 — A FAILED TRANSACTION, WALLET OF 14 ADA, 4 ADA COLLATERAL



WHAT THIS SHOWS — Collateral is anti-spam, not a normal cost — a pre-validated transaction shouldn't reach phase-2 failure at all. CIP-40 made the penalty precise when it does.

STANDING ADVICE

Dated crypto documents read as current until you check. Roadmaps slip, targets get revised, marketing rarely revisits its predictions. Confirm anything version-specific against **docs.cardano.org** or **developers.cardano.org** before entering it into your own record.

TAKEAWAY — promises, checked, kept. Documents age; verify dates before trusting anything labeled "upcoming."

REFERENCE

Glossary

Every term used in this guide, in one place:

TERM	DEFINITION
address	The lock on an output — a public key (needs a signature) or a script (needs the script's approval).
batcher	Off-chain service bundling many users' orders into one transaction, working around contention.
chainstate	A node's on-disk copy of the UTXO set, updated with every block.
CIP	Cardano Improvement Proposal — a numbered public specification for a protocol change.
coin selection	The wallet's silent choice of which notes to spend for a given payment.
collateral	Ada forfeited if a script fails phase-2 validation. Anti-spam, not a normal cost.
contention	Many users needing to spend the same output at once — only one succeeds.
continuity	A validator's power to demand its own correct successor, by inspecting the transaction's outputs.
datum	Data attached to an output; how EUTXO contracts remember things.
determinism	A transaction's outcome and cost are fully knowable before submission.
fan-out	Splitting protocol state across many outputs so users don't collide.
front-running / MEV	Profiting by inserting a transaction ahead of someone's pending one.
gas	Ethereum's unit of computational cost, paid at a fluctuating price.
hard fork	A backward-incompatible upgrade. On Cardano these are coordinated and routine, not schisms.
lovelace	The smallest unit of ada. One ada = 1,000,000 lovelace.
minting policy	A script that alone controls creation and destruction of a native token.
native asset	On Cardano, tokens live in outputs alongside ada at the ledger level — no contract needed for basic transfers.
NFT	A token whose minting policy permits exactly one to exist, ever.
redeemer	Data the spender supplies to satisfy a validator — the plea.
reference input	An output a transaction reads without consuming.
script context	The transaction summary handed to a validator so it can rule on the whole picture — the witness statement.
UTXO set	All currently unspent outputs — effectively the ledger's entire state.
validator	The on-chain script that approves or rejects a spend — the lock that learned to see.
validity interval	The slot range during which a transaction may be included — bounded access to time.

SOURCES

References & Lineage

This casebook began as a study companion to the *Cardano EUTXO Handbook* (Input Output Global, 2022) and has since outgrown it — adding the contention analysis (Part 05), the shipped-upgrade verification (Part 06), native-asset coverage, the three-model verdict board, the dev notes, and the step-by-step reconstruction. The handbook remains the historical source for the marketing-era claims examined in Part 06.

Further leads: developers.cardano.org (current, unusually good on concurrency patterns) · docs.cardano.org (authoritative on eras and governance) · *The Extended UTXO Model*, Chakravarty et al. (the formal paper behind Part 04) · *Native Custom Tokens in the Extended UTXO Model* (the multi-asset proof) · Aiken's documentation (the practical entry point).

Interactive edition: this casebook also lives as an interactive web page — `eutxo-casebook.html` — with the playable vending-machine reconstruction; its "download pdf" button serves this very document. Keep the two files side by side and each references the other.

And a standing habit worth keeping: whenever any document explains a blockchain's design, ask **"what does this cost?"** A document listing only advantages is selling, not testifying — true of the handbook this casebook grew from, and of every chain's marketing without exception.

AN INDEPENDENT GUIDE, EVOLVED FROM THE CARDANO EUTXO HANDBOOK (IOG, 2022) · CLAIMS VERIFIED JULY 2026