Zycord: A Peer-to-Peer Network of Self-Certifying State Transitions

The Simstoshi

2026 — v1.0

GPG: `E724 39CE DD85 11F9 D607 550B 87FD 60D5 EB4A 0B29`

Abstract. Every major blockchain scales by making every node re-execute every transaction against a shared global state. As throughput grows, node requirements grow with it, and the network centralizes. We propose a ledger built from *self-certifying state transitions*. Each transaction carries the state it read and the state it wrote, so verifying it is a pure function of its bytes: no disk, no history, no trust, no bound on parallelism. The chain itself never executes. It orders certificates and commits them with a deterministic fold that applies each certificate whose declared inputs still hold and *skips* the rest. Skipping is not failure but a priced event. Every certificate is underwritten by a party who insures it against staleness, so conflicts have owners, equivocation is objectively slashable, and spam is expensive by construction. Contracts declare, per storage slot, whether access is exact, guarded, or commutative; payments, tips, and mints therefore never contend. Fees split into two markets, one for sequential state mutation, which is scarce, and one for parallel verification, which is not; heavy cryptography is therefore cheap by design. This property enables *confidential payments* — hidden amounts and one-time recipient addresses over a public transaction graph — whose cryptographic cost lands entirely in the parallel market and whose inflation risk a fold rule bounds to the balance of a shielded pool, not merely audits after the fact. Only commitment is sequential, and commitment is a loop over memory. Every node verifies everything at launch, cheaply and in parallel; once verification is sampled rather than repeated, per-node cost falls as the network grows.

---

1. Introduction

A blockchain node today does three jobs that have nothing in common: it *disseminates data*, it *verifies computation*, and it *mutates state*. Data scales with bandwidth. Verification scales with cores: it is embarrassingly parallel if transactions can be checked independently. State mutation is the only genuinely sequential resource: somewhere, a single authoritative history of writes must exist.

Existing designs entangle all three. In the dominant model, a node cannot verify a transaction without holding the global state, so verification inherits the scaling limits of state; and because a single gas market prices all three resources together, a signature check competes in the same auction as a storage write. Recent high-performance chains parallelize execution *inside* the node, but every node still re-executes everything, and the binding constraint becomes state I/O. The answer has been ever-larger machines, which is to say, ever-fewer nodes.

This paper takes the opposite route. We do not parallelize execution against shared state; we **remove state from the parallel path entirely**. A transaction becomes a *certificate* that carries its own inputs and outputs. Verifying it is a pure function of its bytes. The chain's job shrinks to ordering certificates and running a deterministic fold over them — a loop of comparisons and additions that touches state exactly once per certificate. Conflicts do not invalidate blocks; they cause individual certificates to be *skipped*, and every skip is billed to a bonded underwriter. Concurrency is not discovered at runtime; it is *declared*, slot by slot, by the contract author.

The following sections define the certificate (§2), the fold (§3), typed state access (§4), the insurance economy (§5), application sequencers (§6), censorship resistance (§7), the dual fee market (§8), fraud proofs and the sampling road (§9), the virtual machine (§10), machine-less native assets (§11), confidential payments (§12), networking (§13), the three-era launch and its treasury (§14), and measurements from the reference implementation (§15).

2. State Transition Certificates

A **certificate** is the only transaction type in Zycord:

```
Certificate {
    reads:   [(slot, access, operand)]   // declared inputs
    writes:  [(slot, op, value)]         // declared outputs
    program: bytes                       // code or native-op reference
    sigs:    [signature]                 // spending authority
    underwriter: (id, sig, seq)          // who insures it (§5)
    ttl:     height                      // valid if committed by this height
    fee:     (seq_gas_bid, par_gas_bid)  // two markets (§8)
}
```

A slot is `(address, word)`. A certificate is **valid** if three checks pass, none of which require any state:

1. every `sig` authorizes the cells it spends;
2. the underwriter signature is well-formed over `(certificate, ttl)`;
3. re-executing `program` against the declared `reads` produces exactly the declared `writes`. **Signatures must be canonical, and this is a consensus rule rather than a library's good manners.** A certificate's id is the hash of its authorizing fields — reads, writes, program, underwriter, ttl, and the fee bids. Signatures are not among them, and the next paragraph says why. The seen set keyed by that id is the entire replay defence. The fee is an authorizing field and not a relay preference, and the distinction is monetary: a bid is the signer consenting to a cost, so a bid outside the id would be a bid anyone in transit could rewrite — inflated to burn the sender's balance through the base fee, or zeroed to keep the certificate out of every block. What a certificate pays is part of what its signer authorized, and the id says so. What canonicity buys is narrower than an id and is still a consensus rule: a scheme admitting two encodings of one signature admits two exemplars of one certificate, and two implementations that disagree about which of them verifies have forked on a certificate neither can call invalid. The rule closing that is one sentence: a non-canonical signature encoding is invalid, and an implementation whose verifier accepts one is not an implementation of this protocol. Sound schemes reject such encodings already; the reason to write it

down is that the ones which do not are also popular, and an independent implementation that reaches for one would diverge in a way no test of its own would reveal.

Randomized demonstrations are the exception, and the exception is structural, not a matter of encoding. A range proof (§12) is randomized: for one statement the prover chooses nonces, so one statement has unboundedly many valid proofs, all canonical, and there is no non-canonical encoding to reject. Two valid proofs of one statement are not two encodings of a proof; they are two proofs, and the payee — who holds the opening — can always produce a fresh one. Canonicity therefore does not reach them. **A signature is one of these, and this is easy to miss because it is not one of the exotic ones.** Ed25519 is a Schnorr proof of knowledge: the signer chooses a nonce, and every nonce yields a different signature over the same message, each valid and each perfectly canonical. Deterministic nonce derivation is a rule for signers that no verifier can check and no encoding rule can impose, because every nonce point is a canonically encodable point like any other. So one authorization has unboundedly many signatures, and an id covering them would have unboundedly many values — each producible by any *one* of the certificate's required signers, carrying the others' signatures across untouched, and each billable in a block of its own. The threat model is narrower than a proof's and the conclusion is the same: a proof can be re-randomized by a payee holding no key, while a second signature needs a key the certificate already requires. The id closes the gap by construction: it commits to what a certificate *authorizes* and never to what it merely *demonstrates*, so signatures and proofs alike travel outside the id's preimage. Re-signing or re-randomizing yields the same id, the seen set catches the duplicate, and a block carrying both is invalid. This splits a certificate's fields into two kinds — authorization, which the id covers and signatures sign, and evidence, which neither does. The split is not a §12 concern waiting on confidential payments; it is load-bearing from block 0, where the only evidence a certificate carries is its signatures. A verifier checks evidence from the bytes in the parallel stage and then discards it; only authorization survives into the id, the seen set, and the fold.

The split creates an obligation the id no longer carries, and it is named here rather than discovered in production. Evidence outside the id's preimage is evidence anyone in transit can replace: take a certificate, substitute garbage for its proof, and propagate — same id, now-invalid exemplar. Two rules close the two holes this opens. First, a block commits to the evidence it carries. The block's certificate list is a list of **exemplar hashes** — one leaf per certificate, over its whole encoding, evidence included — so "this block is valid" remains a statement about bytes the block itself pins. The id answers *has this authorization been billed*, the exemplar hash answers *do these bytes prove it*, and the two questions never share a key. One leaf suffices while evidence is inseparable from the encoding, as it is while evidence is signatures; when §12's proofs make it a field of its own the leaf becomes the pair *(id, evidence hash)*, and the commitment is the same commitment. What it must never become is a list of ids: that would commit to no particular evidence at all, and since the certificate list's root is a header field and the header is the proof-of-work preimage, swapping evidence would then cost nothing and leave the work standing. This is the precedent of witness-committed transactions, and it is followed for the reason it was invented: an id that covered evidence was malleable, and an id that ignores evidence uncommitted is blind. Second, relay treats exemplars, not ids: a node that receives an exemplar whose evidence fails verification discards that exemplar without prejudice to the id — the id is not marked, not cached as invalid, and a later exemplar that verifies relays

normally. A mutilated copy costs its mutilator the bandwidth of sending it and costs the certificate nothing. The proposer's blindness survives intact: it includes exemplars its own stateless check accepted, so it can no more be induced to include a mutilated proof than a bad signature, and the assertion of §3 keeps the meaning it had.

Validity is a pure function of the certificate's bytes. Any machine can check it, in a thread pool, on another computer, on a GPU, with no database, no history, and no synchronization. Signature checks batch across certificates; certificates from the same program batch into single-instruction, multiple-thread workloads (§6); range proofs (§12) batch with logarithmic marginal cost. This is the property everything else in the paper is built on.

Validity is not sufficient for execution. Two valid certificates may declare the same input value for the same slot; at most one can take effect. We therefore split the classical notion of transaction validity in two: **valid** (stateless, parallel, checked by anyone) and **applicable** (stateful, sequential, decided only by position in the ledger). The next section defines the second predicate.

3. The Ledger as a Fold

A block contains a header, an ordered list of certificate hashes, and the certificate bodies themselves. Bodies are chain data: a block is only valid if its bodies are available (§13). The proposer of a block does **not** execute anything and holds no application state; it orders bytes whose validity it can check from the bytes themselves. It is not quite stateless, and the exception is worth naming rather than rounding away: it must carry the set of certificate ids seen within the TTL window, because including one twice makes the block invalid and nobody can be induced into that by accident. That set is bounded by the TTL and prunable, which is why the TTL is a consensus parameter and not a relay preference. Proposal is deliberately cheap and dumb, not deliberately blind.

The state of the chain is defined as a **fold** over the ordered certificates:

```
apply_block(S, B):
    assert bodies_available(B)
    assert ∀c: height ≤ c.ttl ∧ ¬seen(c.id)     // an expired or replayed certifi-
cate
                                                  // makes the BLOCK invalid: a sig-
nature
                                                  // is billable at most once, and
never at
                                                  // a position its signer could not
avoid
    C ← sort(B.certs, key = (underwriter.id, underwriter.seq, c.id))
    for c in C:
        if leased(S, c):     bill(c, LEASED); mark_seen(c.id, c.ttl); continue  //
§7; Era 1 only
        ok ← true
        for (slot, access, x) in c.reads:
            EXACT:  ok ← ok ∧ (S[slot] = x)
            GUARD:  ok ← ok ∧ pred_x(S[slot])
        if ¬ok:              bill(c, STALE);  mark_seen(c.id, c.ttl); continue
        w ← stage(S, c.writes)   // a target whose address is spent, or a delta
that would
                                 // over- or underflow, fails HERE — nothing has
landed yet
        if w = ⊥:            bill(c, STALE);  mark_seen(c.id, c.ttl); continue
        commit(S, w); settle_fees(c); mark_seen(c.id, c.ttl)
```

Writes are checked, and checked before any of them lands. Earlier drafts of this sketch applied the write set unconditionally, which read as though a credit could resurrect a spent cell and as though a delta could wrap. Neither is true and both would be fatal, so the staging step is in the sketch now rather than left to the specification. Arithmetic is checked everywhere it appears: a delta that would carry past 2^{256} or below zero skips the certificate instead of wrapping it, which is also why §11's supply caps hold under unlimited concurrent mints. A write to an address whose authority has been burned fails loudly instead of vanishing — that is what the permanent spent registry is *for*, and it is the source of the one exception in §5's attribution theorem.

`leased` is Era-1 machinery and is not present at genesis. It is shown here because this section specifies the whole protocol's fold, but the launch era has no leases, no forced queue and no co-signers to defend with them, so the branch is unreachable in the genesis binary — and unreachable consensus code is unauditable consensus code, so it is not shipped. One open question is flagged rather than papered over: as written, `LEASED` bills a certificate at a position its signer could not have foreseen, which is exactly what the assertion four lines above forbids. Either that outcome must not bill, or it must invalidate the block. The choice belongs with the era that introduces leases, and it is recorded here so that era cannot inherit the contradiction quietly.

Properties worth stating explicitly:

Skip is semantics, not failure. A certificate whose reads no longer hold is skipped, its skip fee is billed to its underwriter (§5), and the block remains valid. Inclusion and application are different events. This is what allows the proposer to be blind: it can never produce an invalid block by including a stale certificate.

Determinism. Every node derives an identical state from an identical block sequence. The sort key `(underwriter, seq, certificate id)` is a *total* order on the block's contents, so the fold is insensitive to how a proposer interleaves certificates from different under-

writers — and leaves the proposer no discretion at all, not even between two certificates one underwriter signed at the same `seq`. An underwriter's own pipeline (`seq` ascending) commits in the order it was signed, whatever order the proposer submitted it in.

Exactly one state touch per certificate. The fold performs comparisons, additions, and writes on a hot key-value working set: no code execution, no signature verification, no re-execution. All of that already happened, in parallel, during validity checking.

And no elliptic-curve arithmetic. Confidential payments (§12) put Pedersen commitments into slot values, and a commitment is a curve point. The scheme is homomorphic and the sum of two commitments is meaningful, which invites the fold to add them. The fold refuses. A point addition costs hundreds of nanoseconds against the ~1 ns of a u256 add, and one EC operation in the only sequential stage would spend the budget this architecture protects. The fold only *stores* commitment bytes into fresh cells and *compares* them for equality — both memory operations; all curve arithmetic happens in the parallel stage or in the owner's wallet (§12). The same discipline governs hashing, below.

One piece of cryptography is not removed, and claiming otherwise would be too convenient: a certificate's id is a hash of its authorizing fields (§2), and the seen set is keyed by that id, so somebody has to compute it. What matters is *where*, and the answer is that it need not be here. The id depends on the certificate's bytes and on nothing else — no state, no ordering, no position — so it is computed alongside the signature checks in the parallel stage and carried into the fold, exactly like the certificate itself. An implementation that recomputes it in the sequential loop instead will find hashing dominating a stage that is supposed to be about memory, which is a mistake worth naming because it is an easy one to make. We want to be precise about what is and is not removed here. The fold still performs random access over the full state, one touch per declared slot, and a committing node still holds that state. What leaves the sequential path is everything that surrounds the access on a conventional chain — interpretation, signature checks, hashing, per-operation authentication of a Merkleized store — so state can live in a flat table and the one sequential stage of the system is a tight loop over memory. The reference fold sustains 883,000 slot operations per second per core on a ten-core x86 desktop (§15), and it is the only stage that does not parallelize.

Value equality, not versions. Applicability compares declared read *values*, not version counters. Because execution is a pure function of the read set, a slot that changed and changed back (the ABA case) is still applicable: applying the certificate now is semantically identical to having executed it then. The system therefore skips strictly less often than version-based concurrency control. For commitment-valued slots the comparison is byte equality of the commitment: opaque, exact, and equally cheap, once the canonicity conditions of §4 are enforced.

Replay and single billing. A certificate's id is the hash of what it authorizes and never of the signatures that demonstrate the authorization (§2), and applied *and* billed-skipped certificates alike are marked seen. The distinction is what makes the rule a rule: a signer who re-signs one body at a fresh nonce produces the same id, so the seen set catches the copy instead of billing it. Including a seen or expired certificate invalidates the block — so a signature is billable at most once, and only at a position its signer accepted by signing. Without this rule, a block producer could re-include or deliberately delay other people's certificates to burn their underwriters' funds. TTLs are consensus-bounded, so the seen set stays prunable.

4. Typed State Access

A fold that only supported exact reads would serialize every popular contract: a thousand certificates touching one AMM pool would yield one application and 999 skips. Zycord's answer is that **concurrency is declared by the contract author, per slot, in the certificate format itself**. There are three access disciplines:

Exact. `SLOAD`-style: the read returns the slot's value into the computation and pins the certificate to it. Arbitrary logic; conflicts on hot slots; the domain of application sequencers (§6).

Guarded delta. The certificate asserts a predicate on the slot — `balance ≥ 10` — *without reading the value into the computation*, and writes a signed delta — `balance += −10`. The fold checks the predicate against current state and applies the delta. Any number of guarded debits and credits to the same slot commute: they apply in any order, and a skip occurs only when a guard genuinely fails (a real overdraft), not when the balance merely changed. This single discipline covers transfers, supply-capped mints, allowances, and vault shares, which is to say the overwhelming majority of on-chain writes.

Pure delta. A signed delta with no guard. It can never conflict and never skip. Tips, counters, accumulators, reward tallies.

The rule that keeps the model sound: **guarded values must not flow into computation**. An assert returns nothing; a delta reads nothing. Re-execution therefore remains a pure function of the declared reads, and stateless validity (§2) is preserved. The lineage of this idea is old and solid: escrow transactions in databases, transactional boosting, CRDTs [9][10][11]. But existing chains at best exploit commutativity *inside* one node's scheduler; Zycord makes it the concurrency contract *between* nodes, enforced by the certificate format and priced by the insurance economy.

A second rule of the same rank, forced by §12: **no slot holding a hidden value admits a third-party guard**. A guard whose outcome is publicly observable — applied or skipped — against a balance that is supposed to be secret is a balance oracle: submit `balance ≥ x`, watch the fold, bisect, and $\log_2$(balance) attempts read the number without ever opening the commitment. The fix is structural, not statistical. Hidden value lives only in one-shot cells spendable by their owner's signature; the guarded-delta discipline, whose whole point is that *strangers* can safely touch a slot, is reserved for slots whose values are public. Pull-style payments — allowances, subscriptions, vault sweeps — therefore exist only on the transparent rail (§12).

Enforcing that rule requires that the fold be able to *tell* a hidden slot from a public one, and in a flat table it cannot by inspection: a compressed commitment and a u256 balance are both 32 bytes. A hidden-value cell is therefore not an ordinary cell that happens to hold a commitment; it is a distinct cell kind, created only by the native shielded operation (§12) and living in a reserved, derivable region of the address space, so that the discipline of a slot is a function of its address and checkable from the bytes like everything else. Absent this, a guarded delta aimed — by error or malice — at a commitment slot would have the fold add a u256 to a curve-point encoding: the arithmetic checks pass, the result is not a commitment, and the cell becomes unspendable. Value destroyed in silence, with nothing in the fold failing, is precisely what the staging step of §3 exists to prevent, and the cell kind is what lets it prevent this instance too.

Two further primitives complete the model:

Write-once cells. An address that has never appeared on chain is in state ∅; a first write moves it to *stored*; a signature from its key moves it to *spent*, permanently. Writing to a fresh cell is conflict-free by definition, so paying a newly derived address is the zero-contention fast path. This is the network's chimeric heritage [8]: account-style persistent cells for contracts, UTXO-style one-shot cells for payments, in one ledger. The *values* under a spent cell are compactable once the block that spent it is buried deeper than the reorg horizon — a depth in confirmations, not a finality gadget, since the launch era has no finality to wait for. The registry entry that records the address as spent is never compacted: it is what stops the address being resurrected, and it is the protocol's honest open problem, shared with every nullifier-set design. §12's stealth outputs ride this rail unchanged, and grow the registry at exactly the rate transparent payments already do: one entry per spend, hidden or not.

Zero is absence. A slot that has never been written and a slot written to zero are the same slot: writing zero deletes the cell, and reading an absent cell yields zero. This is not an implementation convenience but a consensus requirement, and it is load-bearing twice over. It is what lets a guard or an exact read name `0` without asking whether the slot exists — there is no third answer to disambiguate. And it is what keeps the state root a function of the *state* rather than of the history that produced it: were a drained cell to linger as an explicit zero, two nodes that reached identical balances by different routes would commit to different roots, which is a chain split arriving by bookkeeping.

This constrains §12, with one sharp edge. A Pedersen commitment `vG + rH` is generally not the zero string, so a hidden balance does not *become* absent by arithmetic and nobody but the owner can tell it is empty; persistent hidden slots would be uncompactable forever, which is one more reason the shielded rail is one-shot cells only, dying by being spent — a transition zero-is-absence never needed to provide. The edge is that `v = 0, r = 0` is the group's neutral element, and in a Ristretto encoding the neutral element is thirty-two zero bytes — the very string that means *absent*. A commitment must never collide with deletion, so the neutral element is not a valid commitment: the shielded operation rejects it on the way in, alongside the canonical-encoding checks a curve point already needs (non-canonical field elements, torsion components), and the reference `H` is a NUMS point with a published derivation. "Byte equality is exact" (§3) is a claim about points only once these hold.

The epoch beacon. Programs must not read ambient values (timestamp, height) directly; that would make execution a function of something other than the declared reads. Instead the protocol writes an epoch beacon into a reserved slot once per epoch, and programs read it like any other slot, ideally with a range guard (`epoch ∈ [e, e+2]`), which provides time-awareness without per-block staleness.

5. Insured Certificates: Every Conflict Has an Owner

Skipping must not be free, or the mempool drowns: an attacker could publish thousands of valid certificates against the same input, fill blocks, and pay for one. But the skipped certificates never touched the sender's balance, so there is nothing to charge the sender; the fee slot is exactly the stale input. Zycord's answer: **no certificate exists without an underwriter**, a party whose bonded funds answer for it.

There are three ways to be underwritten, one mechanism in three guises:

1. **Self-insured.** The sender attaches a small deposit from an unencumbered cell. Nothing is leased in advance: the fold reserves the deposit at the certificate's position in the block (the sketch in §3 elides this plumbing), and a certificate that reaches application with its deposit already consumed is *dropped* — not billed, not marked seen, free to resubmit against a fresh deposit — so honest users lose nothing to races on their own deposit cell. A skip with the deposit intact burns part of it. Publishing, by contrast, costs nothing, so the mempool is bounded by relay policy rather than consensus: nodes cap what they will hold per underwriter, the same division of labor as every mempool since Bitcoin's. This is the permissionless floor: anyone can always transact with no counterparty. It is also the only mode in the launch era (§14), which keeps genesis minimal. The deposit cell is public and it is the sender's, which makes self-insurance a persistent identifier stapled to every certificate it signs. For transparent payments that reveals nothing the payment does not already reveal; for a shielded payment (§12) it undoes the stealth output it pays for, and §12 turns to co-signing for exactly this reason.
2. **Co-signed.** A bonded underwriter checks the certificate against fresh state, co-signs `(certificate, ttl, seq)`, and assumes skip liability. In exchange it can charge the sender out of band, and its co-signature *is* a product: a pre-confirmation an underwriter's own capital stands behind. Nothing in this role requires seeing what a certificate moves: staleness is a property of the *cells* — alive or spent — not of the amounts, so an underwriter prices a shielded certificate as it prices a transparent one, without being trusted with the value.

Two things that promise must not be allowed to blur, because the paper has blurred them before. The protocol's response to a skip is to **burn** the skip fee out of the underwriter's deposit — a penalty, paid to nobody, which is what stops anyone profiting by causing skips. Compensating the *sender* is a different transaction: it is the underwriter's commercial promise, and this design does not yet specify it as a consensus mechanism. Making it one is not a matter of wording — it needs a policy value declared in the co-signature, a named beneficiary who is not the sender (or the product is an invitation to insurance fraud), and aggregate exposure tracked on chain so a bond cannot be sold twice. Until those exist, "my bond pays" is a claim an underwriter makes and a market prices, not a rule the fold enforces, and the paper says so rather than letting the ambiguity sell the product. 3. **Forced.** The censorship-resistance path (§7), underwritten by a user deposit and, uniquely, guaranteed to apply. The economics are asymmetric between two kinds of misbehavior, and the asymmetry is the point:

Objective faults are slashed. If one underwriter co-signs two certificates whose exact reads conflict (same slot, same declared value, overlapping TTLs), the two signatures are a self-contained, on-chain proof of equivocation; anyone can submit them, and the bond is slashed. Likewise, an underwriter who co-signs a certificate that fails stateless validity (a bad signature, a wrong execution) is slashed by pure re-execution: the certificate *is* the fraud proof (§9).

Subjective faults are never slashed. Two *different* underwriters racing on the same slot committed no provable fault; the later one in fold order eats a small skip fee, nothing more. Slow, offline, or unreliable underwriters lose eligibility and reputation, not funds. Networks die when latency becomes confiscation; Zycord only confiscates what can be proven from bytes.

And the bill lands on a party the certificate names. The title of this section is a theorem rather than a slogan, and it is worth stating with its edges. A billed skip is always exactly one of three things: the failing read or write is under an address whose key signed the certificate; or it is a mint racing another mint by the same asset's declared minter, who also signed; or it is a credit to a one-shot address whose own holder retired it (`RETIRE`, §11) after the certificate was signed. No party the certificate does not name can cause it to be billed. Only the third case separates the bill from the cause, and it is bounded by construction: only one-shot cells can ever be retired, so a payee who publishes a persistent address presents no surface at all; a certificate may retire at most a fixed number of addresses (§13), which caps how many in-flight payments one retirement burst can touch; and the skip fee is burned rather than paid, so nobody profits by triggering it.

Bond sizing follows one inequality: an underwriter's bond must exceed the maximum skip liability it can accumulate within one TTL window, and unbonding must take longer than the fraud-proof window, so no one can misbehave, withdraw, and vanish.

6. Application Sequencers

Guarded deltas dissolve contention for payments. What remains is *exact-read* logic on hot state — an order book, an AMM pool — where two honest underwriters racing will still produce skips. The protocol's answer is to let **the contract choose its serializer**: a contract may register, on chain, the key of a bonded sequencer, after which only certificates co-signed by that sequencer may take the contract's exact-read path.

The sequencer is an ordinary server run by the application team — websockets, queues, autoscaling, any web-2 machinery — and it is trusted for *liveness only, never safety*. It cannot forge state: if it lies about inputs during construction, the certificate simply skips against the canonical fold, and the lie costs *its* bond. What it provides:

Serialization. It leases slots to in-flight certificates with short TTLs and chains each certificate on the declared outputs of the previous one (it numbers its co-signatures with `seq`; the fold's total order guarantees its pipeline commits in order regardless of proposer shuffling). Hot-slot contention becomes an off-chain scheduling problem, invisible to the chain.

Batch certificates. N transactions through the same code path fold into one certificate with aggregated reads/writes, internal intermediate writes cut through, and N sub-executions that verify as a single SIMT workload. This is where the GPU thesis becomes concrete: the entity that serializes is also the entity that packages work into the shape parallel hardware wants, and is paid to do so through the parallel gas market (§8).

Atomic composability. A transaction spanning two applications is co-built off-chain by both sequencers (a two-phase commit over slot leases) and lands on chain as *one* certificate bearing both co-signatures, atomic by construction. Cross-application coordination happens where coordination is cheap; the chain sees only the result.

The honest disclosure: a sequencer sees its application's order flow first and is therefore the natural venue for that application's MEV. Zycord makes the trade explicit: the application captures its own MEV instead of leaking it to block proposers, and §7 bounds the power that implies. The same forced path that guarantees exit also disciplines extraction: a user who dislikes a sequencer's treatment can route around it entirely, paying only latency.

7. Forced Inclusion with Guaranteed Application

A contract whose only door is its sequencer is a jail if the sequencer censors. Every user therefore has a slow path that needs no one's permission:

A user (or any relayer) reconstructs state from the public fold, builds a certificate, attaches a deposit, and submits it to the **forced queue**. Proposers must include queued certificates within F blocks (§13). At its inclusion position, the fold checks its reads against current state; if they hold, the fold grants a **deterministic lease** on its slots and schedules application `D` blocks ahead. During the lease, co-signed certificates touching those slots order after it. The forced certificate therefore *cannot be skipped*: admission implies application.

The grace period `D` is for the honest sequencer's in-flight pipeline — the part of it that does *not* touch the leased slots. Anything that does touch them orders after the forced certificate, co-signed and self-insured alike, which is what makes "cannot be skipped" true rather than aspirational: a certificate whose reads are protected for the whole grace period cannot go stale during it. Because writes are declared, the post-application state is predictable, so a sequencer chains over an unapplied forced certificate with certainty.

A lease may only cover slots the certificate has authority over. This is a constraint the mechanism does not give for free and that the rest of the design would not survive without. Declaring a *read* requires no signature — only writes derive authorization — so without this rule anyone could queue a forced certificate declaring reads across someone else's slots and freeze them for `D` blocks at the price of a deposit. The per-contract quota does not bound it either, since native payments belong to no contract. A lease is therefore admissible only over slots the certificate is entitled to write, which is the same authority test the fold already applies, used one step earlier.

Further anti-grief bounds: a per-contract, per-epoch quota; a deposit covering the imposed cost; leased slots reject further forced entries until released.

Two consequences elevate this from feature to survival mechanism. First, censorship inverts: it costs the censor (lost fees, users routing around it) and costs the network nothing. Second, a contract whose sequencer disappears — or a chain whose entire underwriter class is attacked — degrades to *forced-only mode*: slow, expensive, and alive. No state is ever orphaned, and no operator is load-bearing for safety. For a network designed to outlive its founder, this is the property that matters.

8. Two Markets: Sequential and Parallel Gas

A chain consumes three resources of different scalability classes — data, verification, mutation — and pricing them in one market means a zero-knowledge proof verification competes at auction with a storage write. Zycord splits the fee into two independent markets:

Sequential gas prices fold operations: read checks, writes, leases. It is the scarce resource, the one loop every node runs in order, and it is priced accordingly.

Parallel gas prices verification: signature checks, re-execution units, bytes, and heavy precompiles (proof verification, signature aggregation, post-quantum schemes). It is abundant, its supply growing with every core and GPU added to the network, and its block ceiling is set high and cheap.

Both markets have the EIP-1559 shape [15], each in its own unit: the base fee is burned and the priority fee pays the block's producer — **on certificates that apply, and only those.** A skipped certificate's fee is burned in full and pays nobody, and that is not a detail of the fee schedule but the load-bearing rule of the whole economic model. Were a skip to tip, the party who chooses what a block contains would be paid for arranging other people's failures, and the cheapest way to earn would be to farm skips rather than to include work. Burning inverts it: a skip consumes ceiling space and yields nothing, so a producer maximising revenue is maximising application, and is not merely indifferent to causing skips but actively against it. Burning the sequential base fee makes congestion in the one scarce loop deflationary (§14.2). Burning the parallel base fee keeps the proposer indifferent to *which* heavy certificates fill the cheap lane, so verification-lane priority cannot be sold off-book. The split follows the direction of multidimensional fee designs [18], carried through to fully separate markets. What neither market decides is how much of the scarce resource there should *be*; that is a ceiling, and §8.1 gives the rule that moves it.

The consequence is the design goal stated as an invariant: **heavy cryptography does not impact the network, by economic construction**. A certificate that verifies a large proof but writes two slots pays almost everything in the cheap market. Zycord is thereby positioned as the settlement layer where cryptographic protocols too heavy for single-market chains are economical. The same market pays sequencers for shaping work: a sequencer aggregates N transactions into one batch certificate, pays one parallel bid for it, and charges its users for N — the margin is the fee for making work SIMT-shaped (§6).

8.1 The Elastic Sequential Ceiling

A fee market prices scarcity; it does not decide how much scarcity there should be. The two prior answers to that question both failed, in opposite directions. A fixed ceiling turns adoption into an auction the users lose: when Bitcoin's blocks filled, fees crossed $50 and ordinary payments were simply priced off the chain, to the sole benefit of whoever sold the space. An open ceiling without demand fails the other way: its large-block forks bought capacity nobody used and paid for it in security, because volume, not room, is what fees come from. Zycord's ceiling is therefore neither fixed nor voted — a network whose author will not be there to fork it cannot treat routine growth as a governance

event, and a vote is a handle: miners profit from restricting supply, large operators from expanding it past what small nodes can follow. The ceiling is a consensus function of measured demand, in genesis from block 0, moved by no one.

The rule has three parts, one per timescale. *Within a block*, each market has a target `T` and a hard elastic bound `2T`; the base fee steps toward equilibrium by a bounded fraction per block, in the EIP-1559 shape of §8 — bounded because unbounded adjustment is known to oscillate rather than converge. *Across epochs*, the sequential target follows demand: `T ← clamp(2·median_applied(e), T − T/Δ, T + T/Γ)`, floored at its genesis value — where `median_applied` counts sequential gas on **applied** certificates only. Skipped certificates burn their fees and register nothing, so the only way to raise the ceiling is to win application: sustained, non-conflicting, base-fee-burning use. Forcing growth costs an attacker exactly what organic adoption costs everyone else, which is to say the mechanism cannot distinguish them and does not need to. Growth is further gated on an epoch health signal — the observed rate of competing headers stays under a threshold (Era 0), checkpoints finalize on schedule (Era 1 on) — so capacity never outruns what propagation demonstrably carries. A stale block is precisely the event the chain does not record, so the signal is imported the only honest way: headers may cite recent competing headers, unpaid; citing requires real proof of work, while suppressing the signal requires near-every proposer to omit citations that any single honest one restores. The asymmetry leans the gate toward caution, which is the direction a gate should lean. *Per block*, a burst valve: a proposer may exceed `2T` up to `4T`, forfeiting quadratically what the block credits it — **its subsidy share plus the block's fees**, against §14.2's schedule, as a permanent shortfall redirected to nobody — with the penalty assessed on total sequential gas, applied and skipped alike, so the excess cannot be stuffed with manufactured conflict at a discount. The base is the producer's *revenue* and not its subsidy alone because the two do not decay together: the subsidy falls to about 1.6 % of its genesis value on §14.2's curve while the fee revenue a burst is bought for does not fall at all, so a deterrent denominated in subsidy alone stops deterring exactly where the opportunity is largest. A standing property has to be priced in something that tracks the benefit, which is the argument EIP-1559 makes for coupling a charge to the value the manipulation earns. At genesis this changes almost nothing, since fees are near zero. **The treasury share of §14.1 is taken from the unreduced subsidy and is never forfeited**, because bursting is the producer's choice and the treasury is not a party to it. Genuine surges buy passage immediately and inform the epoch controller; spam buys a penalty. The precedent is Monero's penalty-median mechanism [19], with its documented stall — growth freezing when the typical unit of work nears the median — avoided by construction, since batch certificates (§6) keep the typical unit small against the target, and the rate bounds follow the adaptive-limit lineage [20]. The parallel ceiling needs none of this care: it is pinned as a fixed high multiple of the sequential target and inherits its growth, which is the invariant of §8 restated as capacity.

Read the tug-of-war off the mechanism. The user's fee gravitates to the floor, because sustained congestion is, by definition, the signal that raises the ceiling and lowers it back — the Bitcoin equilibrium of permanent auction is unreachable. The producer is paid by subsidy while the network is young and by *volume* at maturity, never by scarcity; the burn makes every congestion episode accrue to the coin the producer holds, and §8 already made application, not exclusion, the revenue-maximising strategy. One honesty, twice over: elasticity guarantees only that capacity is never the reason adoption stalls —

it manufactures no demand, as the large-block forks proved — and a ceiling that grows lets *state* grow, which is why fresh-slot writes price above deltas in sequential gas: the flat table the fold lives in expands no faster than the ceiling that feeds it. One calibration rule follows and is stated because its violation is silent: the genesis ratio of sequential target to byte ceiling must sit below the density of the traffic the network is built for, so that a block full of ordinary payments raises the base fee rather than lowering it. A market that responds backwards to its own design load prices nothing, and the ratio is set from measured traffic before the freeze.

Capacity across eras. The certificate-count and byte ceilings scale with the sequential target rather than standing fixed beside it: at genesis all three bind within a factor of two of each other, so a fixed one becomes the real limit after a single doubling and leaves the elastic one decorative. Behind them stand static capacities: the certificate-list width that fixes a block's merkle depth, and the byte capacity a block may reach, paired with the transport constants beneath it — a block travels as chunks, so no single network message bounds a block. The list width is sized at genesis for the whole curve (2^{25} certificates; repinning it later would put two merkle widths in one chain, and the virtual padding makes the headroom free). The byte capacity is sized for the launch network and repinned at era boundaries — hard forks already (§14) — against propagation the health gate has measured; never inside an era, never by vote. The curve this ladder serves follows from Γ: the target compounds by at most 2× per year of full healthy blocks, so the genesis ceiling of **~90 applied certificates per second** (a §15 block per 30-second interval) reaches **~11,000 per second in seven years** of sustained demand, **~90,000 in ten**, and in under fourteen the **~1.1 million per second** the 2^{25} list width fixes — the one wall no era moves, and therefore the end of the ladder rather than a rung on it. The unit is certificates, not payments: a certificate carries one payment, or the N a sequencer batched into it (§6), so each figure is a floor on transactions and not a claim about them. The conditions are exactly three, each already a mechanism above: demand must sustain full blocks, because applied gas is the only input that raises T; propagation must keep the health gate open, because growth is withheld the epoch it closes; and the re-pins must land at the eras, because between them the byte capacity is the wall. Computation is not a condition — the fold clears the far end of the curve on a single measured core (§15), and verification scales with hardware the network does not own (§2). Bandwidth is: bodies are chain data (§13) and sampling (§9) divides verification, never data, so every node carries every byte, and a producer at the top of the curve is datacenter-class by bandwidth alone — an accepted end state for producers, and for no one else, since verification's per-node cost moves the other way (§9). Before bandwidth binds, state does: the spent registry (§4) grows one entry per one-shot spend, on the order of 10^2 terabytes per year at 10^5 certificates per second, which converts §4's declared open problem from a standing item into work the curve schedules.

Constants (genesis, illustrative, as §13): growth divisor Γ = 512 per epoch (ceiling can at most double per year of full blocks); decay divisor Δ = 1024 (idle capacity halves in ~2 years, never below genesis); burst bound $4T$, forfeiture of the producer's subsidy share *and the block's fees* at the bound (the treasury share of §14.1 is taken from the unreduced subsidy and is never forfeited, because bursting is the producer's choice and the treasury is not a party to it); health gate: cited competing headers ≤ 2% of blocks per epoch; certificate-list capacity 2^{25} (structural, per above); byte ceiling 2.5 MB at genesis, scaling with the target to a structural byte capacity of 8 MB (re-pinned at eras); per-block signature

ceiling 6,000 at genesis, scaling with the target, checked before any signature is verified — a bound on verification, kept separate from the parallel gas *price* because one parameter cannot both bound the work and price the market.

9. Instant Fraud Proofs and the Sampling Road

Because validity is a pure function of a certificate's bytes, **an invalid certificate is its own fraud proof**. Anyone who re-executes it can refute it in one message, immediately, with no state and no interactive bisection game. In the launch configuration the point is moot in the best way: every node verifies every certificate before accepting a block, so an invalid certificate never survives long enough to need a challenge. The window becomes meaningful once verification is sampled, and there it is a propagation parameter rather than a dispute protocol: a handful of blocks for one message to cross the network, charged to the underwriter who attested to the certificate (§5). Compare the alternatives: optimistic rollups need week-long interactive disputes because disputing requires state at the disputed step; proof-based systems avoid disputes but pay for provers that cost orders of magnitude more than execution. Zycord's fraud proofs cost what verification costs: ~1× re-execution.

This opens the scaling road that re-execution chains cannot take. The network does not, in principle, need every node to verify every certificate forever. With underwriter bonds in place, verification can be *sampled*: a VRF-selected committee per certificate, sized so that the probability of an unverified invalid certificate is negligible, with any full node free to check anything and one message sufficient to slash. At genesis, everyone verifies everything; it is cheap and parallel. Sampling is roadmap, not launch. But it is the roadmap that inverts the industry's curve: **per-node cost falls as the network grows**, where the re-execution paradigm's per-node cost grows with throughput until only datacenters remain.

10. The Machine: cEVM

Zycord runs one virtual machine: the **cEVM**, a dialect of Ethereum's EVM [2] adapted to certificates. The choice is deliberate. The project's novelty budget is spent on the state, concurrency, and economic model; the machine should be the most familiar thing in the system. Solidity, its compilers, auditors, and tooling carry over.

Differences from the standard EVM:

- `SLOAD` compiles to an exact read (declared in the certificate); `SSTORE` to a SET write.
- New opcodes `SASSERT(slot, pred)` and `SDELTA(slot, ±v)` expose guarded and pure deltas. An `SASSERT` pushes nothing onto the stack — guards do not return values (§4).
- `TIMESTAMP` and `NUMBER` read the epoch beacon slot; there is no other ambient input.
- Gas metering is dual (§8): storage opcodes meter sequential gas; computation, calldata, and precompiles meter parallel gas.
- The transaction format is the certificate. Ethereum contracts port at the source level; raw wallet compatibility is not claimed, and we do not pretend otherwise. A ported contract runs in all-exact mode: correct on day one, serialized through a sequencer if hot. The parallelism is opt-in per slot, typically a small diff (a token's balance map moves from `SLOAD/SSTORE` to `SASSERT/SDELTA` in ~30 lines). So that the flagship feature is

visible from the first block of the VM era rather than waiting on ports, the machine activates alongside a **native standard library** — token, tip jar, escrow, vesting, and a hybrid order-book/AMM reference — written delta-first and pre-deployed at known addresses.

11. Assets Without a Machine

The launch era needs an economy before it needs a computer. Zycord therefore ships **native assets as certificate operations** from block 0, no VM involved: `ISSUE` (create an asset id with a supply cap in a write-once cell), `MINT` (guarded: `minted + Δ ≤ cap`), `TRANSFER` (guard `balance ≥ Δ`, paired deltas; a credit aimed at a pure delta or a fresh one-shot cell *is* the tip, so tipping spends no opcode of its own), and `RETIRE` (burn an address without spending it: no reads, no value moved, a pure write that marks the cell spent — spending proper is automatic inside `TRANSFER`). `RETIRE` earns its slot twice over. It is the compaction and privacy primitive, letting a payee erase a one-shot address the moment it has served; and it is the operation behind the single exception in §5's attribution theorem — the third case exists *because* retirement exists, which is why the certificate format caps retired addresses per certificate (§13) and why the theorem is stated with that edge rather than around it. The set is closed: these four are the entire genesis instruction set, every other operation — bonding included — arrives with a later era's ruleset (§14), and the attribution theorem of §5 is proven against exactly this surface.

A streamer receiving ten thousand tips in one block costs the fold ten thousand commutative additions: zero conflicts, zero skips, no sequencer, no machine. Tipping cultures on earlier chains lived at the pleasure of platform APIs and died with them; here the tip is a protocol primitive. It is also, not incidentally, a continuous stress test of the system's central claim. The native coin's supply caps live here, on the transparent rail, enforced by the checked arithmetic of §3, and §12 leaves them untouched.

12. Confidential Payments

The preceding sections treat a payment's amount as public. This section adds the option to hide it, and to hide the recipient behind a one-time address, preserving the properties of the preceding sections. The design is narrower than the systems it derives from: each restriction below closes an attack a more general design would answer with heavier machinery.

What is hidden, and what is not. A shielded payment hides its *amount* and defers the link to its *recipient*. Be exact about "defers": a fresh stealth output is unlinkable at the moment it is received, but with no ring, spending it later names the output being spent, and naming the output reveals which prior payment funded it. Unlinkability holds until the first spend and ends there; it is a delay, not an erasure. Nor does the design hide the sender's graph position: certificates are signed, underwritten, and ordered in public, so an observer sees that a payment happened and who underwrote it — not the amount, and not, until a spend discloses it, which output among a recipient's went where. The honest description is *Confidential Transactions with one-time recipient addresses over a public graph*, and the graph deanonymizes retroactively as outputs are spent. Sender-side ambiguity — ring signatures, membership proofs over historical outputs — is out of

scope: a ring references other people's outputs, which makes validity a function of history, and §2's statelessness is the property the protocol does not relinquish. Monero's threat model requires a different protocol.

The shielded output. A shielded payment writes a fresh write-once cell (§4) whose address is a *stealth address*: the sender derives, from the recipient's published view key and an ephemeral key of its own, a one-time address only the recipient can recognize and only the recipient's spend key can sign for. The cell's value is a Pedersen commitment `C = vG + rH`; alongside it ride a range proof that `v ∈ [0, 2⁶⁴)` and the pair `(v, r)` encrypted to the recipient's view key, plus a one-byte view tag that lets a scanning wallet discard most foreign outputs early. The primitives are Confidential Transactions with stealth addressing: the amount-hiding half of RingCT without the ring, each with established production deployment.

Validity stays a function of the bytes. A shielded certificate's balance check is the commitment equation: declared input commitments minus declared output commitments equals `fee·G`, with the fee public (see the pool below). The equation, the range proofs, and the signatures are all checked from the certificate's bytes alone, in the parallel stage, batched: Bulletproof verification amortizes logarithmically across a batch, and none of it touches state. A forged inflationary certificate — one whose commitments do not balance under a broken assumption or a broken verifier — is still *valid* or *invalid* purely by its bytes, and the fold never re-checks it; this is the valid/applicable split of §2 working as designed, not an exception to it, and the pool rule below is what keeps a soundness break from being unbounded. This is also where the dual fee market (§8) stops being an efficiency argument and becomes an enabling one: on a single-gas-market chain, confidential transactions are expensive because a millisecond of proof verification competes in the same auction as a storage write, whereas here the verification lands entirely in par_gas — cheap by design — and the sequential market never sees it.

The fold stores bytes. At application, a shielded payment is the cheapest certificate the fold sees: its output cell is fresh, so it cannot conflict (§4's zero-contention fast path); its input cells are alive or spent, a registry lookup; and the write stores 32 bytes of commitment. No curve arithmetic enters the sequential stage (§3). The alternative the rule forecloses is a persistent "accumulation slot" credited homomorphically by strangers, which fails on three fronts: it puts EC addition in the sequential loop (§3); it creates a reused public identifier, the common-ownership heuristic the stealth address is meant to avoid; and a hidden persistent balance invites the third-party guards §4 bans. Accumulation happens in the wallet: a recipient holds many one-shot outputs, all recognizable through one view key, and consolidates them by spending several of its own cells into one fresh cell, an ordinary shielded certificate, at whatever cadence it likes, with the epoch beacon (§4) as a clock. The sum a persistent slot would maintain on chain is maintained by the owner off chain — and consolidation is not free of trace: declaring several inputs in one certificate is a public common-ownership event, weaker than a reused identifier but real, so wallet policy is to minimize it and never to mix unrelated origins in one consolidation.

Owner-only spends, and what that forbids. A hidden value is spendable only by its owner's signature over its own cell. There are no third-party guards against hidden balances (§4) and therefore no pull payments on the shielded rail: no allowances, no subscriptions, no vault sweeping a user's shielded funds. Those patterns remain on the trans-

parent rail, and the boundary between the rails is one certificate wide. The restriction eliminates the balance oracle: with no stranger able to submit a guess against a balance and read the skip, the oracle has no operator.

The shielded pool is a public integer the fold enforces. Fees are public. Coinbase is public. Payments to contract slots are public. Every crossing between the transparent and shielded rails moves a publicly visible `v` — a shielding certificate opens `v` on its way in, an unshielding certificate opens `v` on its way out. The pool total therefore lives in a reserved slot, in cleartext, moved only by guarded delta (§4): shielding credits it, `pool += v`; unshielding guards `pool ≥ v` and debits it, `pool += −v`; a shielded certificate's fee debits it likewise. The slot is exactly Σ in − Σ out, and the guarded-delta discipline lets unbounded crossings commute without contention.

This makes the pool a fence, not an alarm. Hiding amounts trades §3's checked u256 arithmetic for the computational soundness of a discrete-log assumption, and a break — of the assumption, or far more plausibly of a verifier — forges commitments inside the pool. But a forged commitment carries no cleartext, and value leaves the pool only by unshielding, which debits the public slot under its guard. The slot rose only on real shielding, so an unshield that would drive it below zero *skips*: forged value cannot cross the guard. Inflation does not drain the pool; it fails to exit it. The blast radius of a cryptographic failure is bounded, by a fold rule rather than by an auditor's vigilance, to the pool's real contents, and the failure mode is not a silent drain but a visible, real-time race in which the last honest holders to unshield cannot — bad, bounded, and observable, which is the containment a system with no emergency response team must have by construction. The precedent is concrete: the Zcash Sprout counterfeiting bug was survivable because its shielded pool was a bounded, auditable quantity; here that quantity is not merely auditable but load-bearing in the fold. §11's native supply caps live on the transparent rail and remain enforced by checked arithmetic, untouched.

The pool has a privacy cost, and it belongs on the record: boundary crossings expose exact public amounts, and distinctive amounts correlate. An observer who watches `v` leave the pool shortly after `v + fee` entered it has learned something no commitment hid. Standard denominations at the boundary blunt this, and wallets default to them; the protocol does not mandate them, because a consensus rule cannot distinguish a distinctive amount from a legitimate one. Linkability under traffic analysis is stated, with its mitigation, rather than denied.

The underwriter is metadata, and privacy and censorship-resistance do not coexist in one certificate. A self-insured shielded certificate names the sender's public deposit cell, which re-links what the stealth address unlinked (§5); shielded traffic therefore rides co-signed underwriting, where an underwriter's id aggregates many senders and the crowd is the cover. The underwriter prices liveness of cells, not values (§5), so it does not learn the amount — but it learns everything else the certificate is: the sender it must bill out of band and therefore identify, the cells being spent, the stealth outputs being created, and the timing. That is precisely what a subpoena wants, and with the retroactive deanonymization above it composes forward through the public graph. It is also a structural KYC point, because the one private mode has a knowable operator. This exposes a limitation the paper states plainly rather than papering over: the two underwriting modes are co-signed (private, but permissioned — an underwriter may refuse you) and self-insured or forced (permissionless, but self-identifying). The forced path of §7 guaran-

tees that a censored user can always transact; it does not guarantee they can transact *in private*. A user refused by every underwriter keeps the right to spend and loses privacy in the same motion — and that is exactly the user for whom privacy mattered. The shielded rail inherits §7's censorship-resistance only at the cost of its own privacy; the two properties do not hold in a single certificate.

An era, not genesis. Two independent arguments fix the schedule. In practice, shielded certificates use co-signed underwriters, which exist from Era 1. By the principle of §3, unreachable consensus code is unauditable consensus code and is not shipped: the genesis binary contains no commitments, no range-proof verifier, no pool slot — nothing whose only caller is a future era. The shielded era is not purely additive, and the paper says so: the hidden-cell typing and third-party-guard ban of §4 touch the fold's guard path, which is genesis-critical surface, so the era modifies as well as adds, and its activation is a hard fork audited as the consensus-critical change it is. An era that does not prove worth its complexity is simply never triggered, and the genesis chain has lost nothing by not containing it.

Costs, stated plainly. A shielded certificate is 3–5× the size of a transparent one — one to two kilobytes, dominated by the range proof even after batch aggregation — which the dynamic block (§8) absorbs economically and §13's dissemination pays in bandwidth. Recipients discover payments by scanning: every new shielded output is trial-tested against the wallet's view key, an O(n)-in-network-volume cost that view tags reduce by an early reject on one byte, a constant factor bounded by the shared-secret derivation that precedes it. Scanning is the shielded rail's principal UX burden; outsourced scanning that does not surrender the view key is an open problem this design inherits rather than solves. Relay policy must also change on this rail, and not optionally: publishing a certificate is free (§5, §13) while verifying a Bulletproof is not, so a relay that runs the verifier before any cost is imposed on the publisher is a denial-of-service amplifier — the shielded rail requires check-cheap-before-expensive relay (signature and declared liveness first, proof last, per-underwriter quota), which turns the mempool's "bounded by relay policy" from a default into a requirement.

Open questions this section leaves to a later draft, flagged rather than buried. Whether the shielded rail carries only the native coin or also §11 assets: a single generator `H` makes `vG + rH` commit to a scalar, not to an (asset, value) pair, so a multi-asset shielded rail needs per-asset generators (Confidential Assets), which changes proof size and the "3–5×" figure above; until decided, the rail is native-coin-only *by rule*, not by omission. Whether `RETIRE` (§11) acts on shielded cells: if it does, value can leave the pool without a public unshield, so the pool slot becomes an upper bound (Σ in $-\ \Sigma$ out $\geq$ contents) and the inflation-detection direction must be re-stated as an inequality; if it does not, the shielded rail loses its garbage collector and unopenable outputs (below) accumulate. Whether the encrypted `(v, r)` lives in the certificate body (cheap now, unrecoverable if bodies are pruned, defeating seed-only wallet restore) or in the cell value (costs state per output, disappears when the cell is spent, and coheres with "accumulation happens in the wallet"). And whether an underwriter can bond a third party while charging *inside* the certificate in public value, which would remove the identity requirement above and is the most promising route past the privacy/censorship limitation — these are design, not wording, and are named here so the next draft cannot inherit them silently.

13. Network

Bodies are chain data. A block's certificate bodies must be retrievable for the block to be valid; state is therefore always reconstructible from the chain alone, and no operator — sequencer included — is ever a data custodian. Certificates are bigger than classical transactions (they carry reads and writes), and the mitigations are structural: batch certificates cut through internal writes (§6), and spent one-shot cells compact once they are buried past the reorg horizon (§4).

A third mitigation is *prospective* and is named as such rather than assumed: a read could reference a prior certificate's write by `(cert_id, index)` instead of repeating the value — the UTXO trick. It is not part of the protocol described here, and it does not become part of it until one question has an answer, because the question is a consensus question rather than an encoding one: what a reference resolves to when the certificate it names was skipped, or was never included at all. A reference that silently resolves to the current value is not a declared read any more, and stateless validity dies with it; one that fails takes down certificates whose own authorization was never in doubt. Compression that costs the property the whole design rests on is not compression worth having, so the field is absent until the semantics are settled.

Relay is hash-first. Certificates gossip independently and are validity-checked (statelessly, in parallel) on arrival; blocks relay as headers plus hash lists against the mempool, compact-block style [13], so propagation latency does not scale with block contents. A relay node needs no state whatsoever to fully filter spam — stateless validity plus underwriter signature is checkable from bytes, which makes running relay infrastructure nearly free on the transparent rail. The shielded rail (§12) is the priced exception: its stateless check includes a range-proof verification that costs three orders of magnitude more than a signature, so there the free-to-publish default becomes an amplifier, and relay is bound by the check-cheap-before-expensive discipline and per-underwriter quotas §12 makes a requirement rather than a preference. One rail's relay is nearly free; the other's is cheap only because its policy is mandatory.

Parameters (genesis, illustrative). 30-second blocks; certificate TTL default 240 blocks (~2 h); retired addresses per certificate ≤ 64; epoch 2,880 blocks (~1 day); forced-application delay D = 4 blocks; forced-inclusion bound F = 16 blocks; era-trigger stake window K = 14 epochs (~2 weeks); treasury share 300 basis points (3%) of block subsidy from block 0, cell sealed until Era 2, then 3-of-5 over a key set pinned by hard fork, key-rotation delay R = 20,160 blocks (~7 days) (§14.1); issuance constants in §14.2.

14. Launch: Three Eras, Zero Premine

Zycord launches with no premine, no founder allocation, no investor round, and no admin keys. Genesis is reproducible from published sources. A testnet runs first; only once it is stable is a mainnet date announced, in the open and ahead of time, so that block 0 is reachable by anyone who wants it. What is enforced on the author is the absence of privilege, and that is checkable line by line in genesis. Upgrades happen by social consensus and hard fork.

This paper is the argument. The rules are the architecture specification, the golden vectors are the protocol, and where any two of them disagree the more precise one wins and the disagreement is a bug. Published alongside them is the record of this design being attacked — successive adversarial readings, the findings they produced, the ones that changed the rules, and the ones that were argued down and why. That record is released in full and unedited, including the reviews that found real defects and the instruments that reported success while measuring nothing. For a project whose author will not vouch for it in person, a design with no visible history of being attacked reads as a design nobody attacked, and there is no way to earn back what hiding it would cost.

"Unedited" is a promise about the findings, and it is worth stating exactly what it covers, because the record is republished as files rather than as a live archive. Every finding, every measurement, every argument and every rejected alternative survives as it was written, including the ones that were wrong and the instruments that reported success while measuring nothing; nothing is thinned, softened, merged or dropped. Two classes of change are made and only two. The first is identity redaction — names, handles, addresses, machine names and local paths, which say nothing about the design. The second is self-containment: a pointer into a tracker or a commit history that the published tree does not carry is replaced by the reasoning it stood for, stated inline. A reader holding only these files can therefore follow every claim, which is what the promise was for; a pointer that resolves to nothing would keep the letter of "unedited" and lose the whole of its purpose.

Proof of stake cannot fair-launch — initial stake must come from somewhere, and every classical route (sale, premine, allocation) either concentrates the network or identifies its author. Proof of work is therefore used *as a distribution mechanism with an expiry date*, not as a permanent consensus:

Era	Trigger	Consensus	What exists
0 — Mine & Tip	block 0 (public mainnet date, announced after a stable testnet)	PoW (RandomX [14]), Nakamoto rules [1]	Native assets (§11); every certificate self-insured; **no leases and no forced queue** — both are Era-1 machinery (§3, §7); treasury accrues, sealed (§14.1)
1 — Payments	height H_1 (instruction set); finality overlay once bonded stake ≥ 1% of supply sustains K epochs	PoW proposes; from overlay activation, bonded validators finalize checkpoints every 32 blocks (FFG overlay [12]) and the subsidy splits 77/20/3 — producer / checkpoint attesters / treasury	`BOND`, underwriters & sequencers (§5–6); cEVM + standard library (§10); pre-confirmation market matures with finality (design notes)
2 — Platform	height H_2 *and* stake ≥ 10% sustained for 30 epochs	PoS committee proposes; PoW reward ramps to zero over ~90 days, pausing while checkpoints fail to finalize; randomness moves from PoW hashes to VRF	Full protocol; treasury cell opens under 3-of-5 (§14.1); sampling research (§9) begins
S — Shielded *(activates against whichever era is current; requires Era 1's instruction set)*	hard fork adopted by node operators, proposable only after all of: co-signed underwriting live (§5); the verifier and its batch path published with at least two independent audits, findings and remediations public; the full shielded rail run on a public testnet with adversarial participation for ≥ K epochs; golden vectors for the verifier in the protocol artifact	unchanged — the era adds no consensus role and changes no ordering rule	Confidential payments (§12): shielded operations, hidden-cell kind and guard ban (§4), pool slot and its fold rule, range-proof verifier as consensus-critical code

Design notes on the transition. Era 0 is deliberately tiny. With no admin keys there is no pause button, so every genesis line is a line that can kill the network without remedy; the fold (§3) plus the native operations (§11) are the entire safety-critical surface (small enough that this paper contains their complete specification), and they ship audited and adversarially simulated. CPU mining (RandomX) matches the demographic: mine on the laptop you already have. It also invites botnets; every CPU-minable coin has fought them, and we do not expect to be the exception. We take that trade with open eyes: a distribution skewed by botnets is still broader than one decided in a sale, and keeping honest commodity hardware competitive is RandomX's entire design brief [14]. Co-signing is live from the moment underwriters can register; checkpoint finality is not, until stake sus-

tains, and we name the gap instead of hiding it: the bond prices skips, never reorgs, so no protocol rule stops an underwriter from selling pre-confirmations into that gap — what no rule can do is make the sale honest. “Applies in seconds or my bond pays” is underwritable only once deep reorgs are off the table, so the insured-preconfirmation product follows finality as market honesty rather than protocol law. Attesters are paid from issuance from the moment the overlay activates — both prior hybrid transitions did this (Decred pays PoS voters a share of every block reward [17]; Ethereum’s beacon validators earned issuance for two years before proposing mainnet blocks) — and the producer-heavy 77/20/3 split is confined to the distribution phase, since the same precedent shows a miner-heavy split is the wrong permanent state: the ramp retires it. Era 1 splits its trigger, and the split is load-bearing: the instruction set activates by height — `BOND` first, underwriter and sequencer registration and the cEVM after, two heights the genesis parameters keep distinct and ordered — while the finality overlay activates only once bonded stake ≥ 1% of supply has sustained for K epochs, because stake cannot be measured before the operation that creates it exists. Era 2 keeps a dual trigger (height *and* sustained stake), so the network neither transitions on trivial stake nor lets incumbent miners stall it forever; heights are thresholds, not dates, and no calendar is promised. One caveat we state rather than bury: at the 1% activation floor, stalling finality takes a third of that (0.33% of supply), so the finality the first insured pre-confirmations rest on is itself young. The K-epoch sustain requirement raises the bar over time, the inactivity leak makes a stall expensive to hold, and underwriters are expected to price young finality into young policies; deeper security arrives with deeper stake, not by declaration. Era 1 is the shadow chain: the validator set finalizes in production for the entire era — real bonds, real slashing — before proposing a single block, the property that made the one successful PoW→PoS transition at scale safe. It is also a resting state, not a corridor: if stake never sustains the Era-2 threshold, the network remains a functioning hybrid indefinitely. The reward *ramp*, rather than a cliff, denies a miner-fork rally its moment, and the ramp is abortable: if checkpoints fail to finalize for two consecutive epochs, it pauses at its current level until finality resumes. Miners cannot buy the pause: stalling finality takes a third of bonded stake, which the inactivity leak bleeds until finality returns, so the attack costs more than the paused reward pays. Era-0 miners who bond their coinbase receive priority — not extra coins — in the first sequencer rotations, converting the mining community into the operator community instead of discarding it.

The shielded era’s trigger is deliberately shaped unlike the others. Heights and stake thresholds are facts a node measures; a cryptographic verifier’s readiness is not, so the trigger is instead a checklist a hard fork must be able to cite: audits published, testnet epochs served, vectors in the artifact. The form matters more than the items. This network has no emergency response team — the Zcash counterfeiting bug was survivable partly because a staffed organization shipped a fix in days, and nothing here can promise that — so the era’s activation is designed to *create* its maintainers rather than assume them: a community that cannot produce two independent audits and staff an adversarial testnet is a community that cannot maintain a consensus verifier, and the checklist makes the second incapacity visible as a failure of the first, before activation instead of after a break. The fold rule of §12 bounds what a surviving bug can cost; the checklist bounds how unexamined a verifier can be at the moment it starts costing anything. Neither replaces the other, and the era ships with both or not at all.

The ramp is a redivision and not a reduction: whatever share proof of work gives up, the proof-of-stake proposers take, so the subsidy schedule of §14.2 is untouched by where the network is in its transition. This is what lets the schedule stay a pure function of height, computable by any node from four constants without consulting the ledger.

The transition is tractable for an architectural reason worth stating: **the fold is agnostic to who orders**. A PoW miner already is the blind proposer the design requires; Era 2 changes whose signature is on the header and not one rule of state semantics. There is no execution engine to carry across the boundary.

An era boundary is also where capacity moves: the byte capacity of §8.1 and the transport constants beneath it are re-pinned there, against propagation measured on the running network, so routine growth rides the upgrades this schedule already contains instead of becoming a governance event of its own.

14.1 The Treasury

Three percent of every block subsidy is credited to the treasury cell; ninety-seven percent pays consensus — the block's producer and, from Era 1, the checkpoint attesters (§14). The share is fixed in genesis, applies from block 0, and applies identically to every block. Fees are never touched (the treasury has no claim on the markets of §8), so the cost falls on issuance, spread across every holder in proportion to holdings.

The cell is sealed until Era 2. It accrues from block 0 and no key opens it before then: no quorum in genesis, no address, no author key, no emergency path. At block 0 there is no mature community to hold the keys. If none arrives, the cell never opens and the coins are never issued. The accrual rule sits in genesis from the first block so that it is never a *change*; nothing new happens at Era 2 except that an agreed rule becomes operative.

None of this is a premine, and the difference is checkable rather than rhetorical. A premine is an allocation: coins, an address, and a key that exist at genesis and belong to someone. Genesis here contains none of the three. No treasury coin is spendable, no address is designated, no key exists, and no party is named; what genesis contains is a rule, applied identically to every block ever mined, plus a second rule for how a quorum may one day be pinned over the accumulated result. Whether that day comes is not the author's call. Because the key set is pinned by hard fork, the selection mechanism is the one every consensus change uses: someone proposes five publicly identified holders, node operators adopt the fork that pins them or decline to run it, and a candidate quorum that cannot convince the network to run its fork holds keys to nothing. There is no vote to capture, no registry of contributors to game, and no moment at which the author's voice outweighs any other node operator's.

From Era 2 the cell is debited only by a certificate carrying three signatures of five over a key set pinned by hard fork. Spends are ordinary certificates: public, final, and countable from the chain alone. The quorum is drawn from contributors active at the time of opening and publicly identified before the set takes effect; each key sits with a distinct party, none under common control. A valid 3-of-5 spend may instead name five replacement keys, effective after R blocks, during which the rotation is public and the old set still holds: rotation carries a spend's threshold and a spend's visibility.

Eras 0 and 1 are donation-funded: proposals public before payment, milestones and budget up front, release against delivered work, unspent amounts returned to the fund. The quorum inherits that discipline when the cell opens. It is a norm and not a consensus rule; what enforces it is that every departure is visible in the ledger.

The share does not expire. It is a fraction of issuance rather than a quantity of coins, so it decays with the subsidy of §14.2 and persists at the tail as a fixed nominal stream; as a fraction of supply it tends to zero. Removing it costs what any consensus change costs: a hard fork. From Era 2 the 3-of-5 is the protocol's only trusted quorum.

14.2 Issuance

The subsidy decays smoothly to a perpetual tail. The rate is constant within an epoch and steps down once at each epoch boundary, by exact integer arithmetic over four constants:

```
E(0) = E₀
E(n) = max(tail, E(n−1) − E(n−1)/Q)
```

where `L` is the epoch in blocks, `Q` the decay divisor, `E₀` the genesis subsidy, and `E(n)` the per-block subsidy throughout epoch `n`. Those four — `L`, `Q`, `E₀`, `tail` — are the constants; `C`, the pre-tail supply the schedule sums to, is **derived from them by running the recurrence** and is not an input to it. Earlier revisions wrote the first line as `E(0) = C / (L · Q)`, which reads as though `C` were a constant the subsidy is divided out of; it is the closed form of the *infinite* geometric decay, and the finite schedule below does not sum to it. There are no floats and no table to get wrong, and no halving cliffs: the step is small enough that no single block sees one, the same reasoning that makes the Era-2 reward a ramp. Once the formula falls below the tail, the subsidy is a fixed amount per block, forever.

The schedule is a pure function of height. It does not consult how much was actually paid, which means a block that pays less than the schedule — a coinbase owed to an address whose holder burned it, say — is a permanent shortfall against `C` rather than a debt the curve later repays. `C` is therefore the figure the schedule sums to, not a cap it guarantees to reach.

The tail exists because security is a permanent expense: it pays whoever secures the current era — miners, then miners and attesters, then validators — without conscripting the fee markets of §8. It is sized so that annual tail issuance starts below 1% of outstanding supply; because the numerator is fixed and the supply grows, that percentage only falls. Against it runs the sequential base-fee burn of §8, so net issuance is tail minus burn and can be negative under load. The 97/3 split of §14.1 applies to every subsidy, tail included.

Constants (genesis, illustrative, as §13): epoch `L` = 2,880 blocks; decay divisor `Q` = 1054; genesis subsidy `E₀` = 21/block; tail 0.33/block. With 30-second blocks these give a yearly factor of 0.70702, which is to say the emission rate **halves every two years** — and exactly, not approximately: `E(730) = 10.50230028` against `E₀ = 21`. The formula falls below the tail at epoch **4,376**, height **12,602,880**, ≈ year 12, so the decaying arm runs over epochs 0 through **4,375** and `C`, the supply it sums to, is **62,744,838.47**. Of that, **62,744,817.47 is actually issued**: the difference is the genesis block's 21, which pays no coinbase at all because there is no miner to pay and a reproducible genesis must not

credit an address. Annual tail issuance is then ≈347K, or 0.55% of the issued supply, falling from there. **The closed form `L · E₀ · Q` = 63,745,920 is an upper bound on `C`, not its value**, and the gap is not a rounding error: that product is the sum of the *infinite* geometric decay, while the schedule is finite, floors at every step and stops at the tail. It overshoots `C` by 1,001,081.53, or 1.60%. Quote the sum and never the closed form. Conformance is reproducing the recurrence — no node ever computes `C`, or any other figure in this paragraph, at any height.

15. Measurements

A design whose central claim is a throughput asymmetry owes the reader numbers. The figures below are from the reference implementation and can be re-derived from its published source. Hardware: a ten-core x86 desktop CPU. Figures are medians over 5 runs.

- Fold throughput: 883,000 slot operations per second per core over a working set of 100,000 slots, i.e. 294,000 applied certificates per second at 3 slots per certificate.
- Stateless verification: 1,470 certificates per second per CPU core, and 5,220 per second across 10 cores — the ratio is the parallelism claim, measured rather than asserted.
- Signature verification: 1,500 per second, single, with the small-order and torsion checks included.
- End to end: a block of 2,900 certificates validates in 1,963 ms and folds in 9.9 ms. The two are reported apart because they are the two halves of the argument: the first scales with cores the network does not own, the second is the one loop it does.

The measurement of the fold is a subtraction, and saying so is part of reporting it: committing a block runs the stateless checks and the sequential fold together, so the fold's figure is the whole minus the checks timed on their own. Both halves are in the artifact.

We report only what the reference implementation runs, which sets two limits worth naming rather than leaving for a reader to discover. Signature verification is measured one signature at a time: batch verification is a real technique and an obvious fit, but it is security-critical cryptography this project has not written, and a number for code that does not exist is not a measurement. The GPU and SIMT figures that §2 and §6 anticipate belong to the sequencer's batch certificates, which arrive with Era 1. Both are design expectations. Neither is load-bearing for the numbers above: the parallelism claim rests on certificates being independently checkable, which batching would make faster and cannot make true.

16. Related Work

Zycord's pipeline — execute first, order blind, validate at commit — has an ancestor in permissioned settings: Hyperledger Fabric's execute-order-validate [3], whose documented weaknesses are its abort rate under contention and its reliance on institutional endorsement policies. Zycord's contributions relative to that lineage are (i) value-equality applicability with skip as first-class, ABA-tolerant semantics; (ii) *economic attribution* of conflict — bonded underwriting that makes equivocation objectively slashable and staleness a priced, insurable service, which is what execute-order-validate lacked to go permissionless; and (iii) forced inclusion with guaranteed *application*, where rollup escape hatches guarantee only inclusion. Deterministic pre-declared read/write sets descend

from Calvin [4] and appear in Solana's access lists [16]; commutative escrow methods date to O'Neil [9] and live today inside single-node schedulers such as Aptos aggregators over Block-STM [5] — Zycord moves the commutativity contract between nodes and prices it. Parallel-EVM systems [5] and deferred-execution chains still re-execute everywhere and hit the state-I/O wall; Narwhal [6] separates data dissemination from ordering as our early designs did; Sui's owned objects parallel our write-once cells; the pending/base split of Zether [7] inspired the guarded-delta discipline (and returns, with the chimeric ledger's privacy roots [8], in §12's confidential payments — the first tenant of the heavy-crypto lane, hidden amounts and one-time addresses priced into parallel gas rather than built into the base layer). Chained finality overlays follow Casper FFG [12].

17. Conclusion

We have proposed a network that orders certificates instead of executing transactions: validity checked by anyone, anywhere, in parallel, from bytes alone; state advanced by a fold simple enough to specify in a page; conflicts skipped, priced, and owned; concurrency declared rather than discovered; censorship survivable by construction; and a launch that distributes the coin before it asks anyone to trust a validator set. Verification scales out with hardware the network does not own. Commitment stays sequential — and nearly free.

Zycord does not chase the work. It holds still, and the network does the work.

References

[1] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* 2008. [2] V. Buterin. *Ethereum: A Next-Generation Smart Contract Platform.* 2013. [3] E. Androulaki et al. *Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains.* EuroSys 2018. [4] A. Thomson et al. *Calvin: Fast Distributed Transactions for Partitioned Database Systems.* SIGMOD 2012. [5] R. Gelashvili et al. *Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing.* 2022. [6] G. Danezis et al. *Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus.* EuroSys 2022. [7] B. Bünz, S. Agrawal, M. Zamani, D. Boneh. *Zether: Towards Privacy in a Smart Contract World.* Financial Cryptography 2020. [8] J. Zahnentferner. *Chimeric Ledgers: Translating and Unifying UTXO-based and Account-based Cryptocurrencies.* IACR ePrint 2018/262. [9] P. E. O'Neil. *The Escrow Transactional Method.* ACM TODS, 1986. [10] M. Herlihy, E. Koskinen. *Transactional Boosting: A Methodology for Highly-Concurrent Transactional Objects.* PPoPP 2008. [11] M. Shapiro et al. *Conflict-free Replicated Data Types.* SSS 2011. [12] V. Buterin, V. Griffith. *Casper the Friendly Finality Gadget.* 2017. [13] M. Corallo. *BIP 152: Compact Block Relay.* 2016. [14] tevador et al. *RandomX: Proof-of-Work Algorithm Optimized for General-Purpose CPUs.* 2019. [15] *EIP-1559: Fee Market Change for the ETH 1.0 Chain.* Ethereum Improvement Proposals, 2019. [16] A. Yakovenko. *Solana: A New Architecture for a High Performance Blockchain.* 2017. [17] *Decred Documentation: Issuance.* docs.decred.org — block reward split between PoW miners, PoS voters, and treasury. [18] *EIP-4844: Shard Blob Transactions.* Ethereum Improvement Proposals, 2022. [19] *Monero: Dynamic Block*

Weight and Penalty. Monero Research Lab documentation; see also JollyMort, *Monero Dynamic Block Size and Dynamic Minimum Fee*, 2017. [20] bitcoincashautist. *CHIP-2023-04: Adaptive Blocksize Limit Algorithm for Bitcoin Cash.* 2023.