Cowboy: A Layer-1 Blockchain for Autonomous Software

Technical Whitepaper

Charles DePue, Patrick Mandia, Tony Xia, Martin Aceto, Logan Lamb, Ryan Chan

November 10, 2026 — v0.10.8

Status	Draft for internal review
Type	Standards Track
Category	Core
Author(s)	Charles DePue, Patrick Mandia, Tony Xia, Martin Aceto, Logan Lamb, Ryan Chan
Publisher	Cowboy Foundation
Created	2025-09-17
Updated	2026-11-05
License	CC0-1.0

> This document specifies Cowboy Chain's protocol contract.

0.1 Abstract

Cowboy is a general-purpose Layer-1 blockchain and the consensus core of an open operating system for autonomous software. It combines a **Python-based actor-model execution environment** with **proof-of-stake consensus** and protocol-mediated off-chain computation. Smart contracts on Cowboy are **actors**: durable Python programs with actor-scoped consensus state, a mailbox for messages, and chain-native timers. A decentralized network of **Runners** executes LLM inference, web requests, MCP tool calls, and **containerized** jobs outside replicated execution under explicit verification, authorization, and payment rules.

Cowboy's near-chain architecture comprises **CBFS** (the Cowboy File System) for private or public volumes, **CBSS** (the Cowboy Secret Service) for threshold-authorized secret and key release, and **CBQS** (the Cowboy Queue System) for durable workload coordination. Their payload-bearing operations stay off-chain; the protocol records the control state each subsystem requires, including resource lifecycle, authority, provider assignment, commitments, receipts, pricing, and settlement. These services support chain-anchored applications. Protocol-managed registries also provide session payments, routing, event subscriptions, and actor hiring and distribution.

Cowboy supports two named application shapes. A **consensus application** centers deterministic actor execution and replicated state. An **owner-controlled application** pairs chain-anchored ownership and authority with predominantly provider-served execution and data. Execution locality alone does not establish owner control. Providers retain the operational authority stated by each subsystem. Most applications combine both shapes.

Cowboy introduces a **tri-metered gas model**, separating computation (**Cycles**), consensus store I/O (**Accesses**), and persisted data (**Cells**) into independent, EIP-1559-style fee markets (introduced by CIP-43). Security is provided by **Simplex BFT** consensus with **proof-of-stake**, fast finality, and VRF-based leader election.

This document specifies Cowboy Chain's consensus architecture, state transition function, economic mechanisms, system-actor boundaries, and protocol parameters. Separate subsystem whitepapers and CIPs define CBFS, CBSS, CBQS, containers, routing, payments, and distribution in greater detail.

0.2 Introduction

Cowboy is designed as an open operating system for autonomous software. An AI agent is one application shape; the protocol execution unit is the **actor**, a durable program with identity, state, a mailbox, economic agency, and a clock. This document specifies the Cowboy Chain protocol for implementers, auditors, and protocol developers.

Ethereum smart contracts execute when an externally controlled system calls them. Cowboy introduces a different unit of execution: the **actor**, a stateful program that schedules future work and pays from its own balance without an external keeper. Actors remain subject to fees, storage caps, and state rent; dormant or unfunded state can be evicted and later restored under the rules in this document. The protocol supplies durable identity and scheduling, not an unconditional promise that code runs forever.

Cowboy uses Python for execution. Other chains use Solidity on Ethereum or Rust on Solana. Python's ubiquity in AI and general-purpose development reduces the barrier to entry for both human developers and AI coding agents, which already generate Python more reliably than niche smart contract languages.

The PVM is a deterministic RustPython-derived runtime, not CPython. Familiar syntax and pure-Python reuse are product advantages; native-extension packages and the broader scientific/AI stack execute through Runners rather than inside consensus.

0.3 Architectural Overview

This section is descriptive and non-binding. Normative requirements are in §§1–17.

0.3.1 Terminology

- **Actor:** A durable Python program with actor-scoped consensus state, a mailbox, a balance, and protocol-native scheduling.

- **Message:** A datagram delivered to an actor handler.
- **Cycle:** Unit of metered on-chain compute.
- **Access:** Weighted unit of consensus store I/O; one cold `StateGet` is the base unit.
- **Cell:** Unit of metered persisted or payload bytes (1 cell = 1 byte).
- **Runner:** Off-chain worker that executes a job and returns a result plus any evidence or attestation required by its trust profile.
- **Near-chain service:** Protocol-defined, stateful resource service with an on-chain lifecycle or authority record and a provider-served data path outside consensus. In this document the term refers to CBFS, CBSS, and CBQS; per-job Runner execution is a separate compute plane.
- **Consensus application:** Application shape centered on deterministic actor code and replicated state; off-chain jobs provide inputs to consensus-visible decisions.
- **Owner-controlled application:** Chain-anchored application shape in which durable identity and application-level authority live on Cowboy while most executable logic uses Runner jobs or persistent workloads and high-volume data uses near-chain services. Near-chain providers can remain authoritative for operational state assigned to them. Owner control is an ownership objective, not a blanket confidentiality or provider-portability guarantee.
- **CBFS (Cowboy File System) / attachable volume:** Distributed object storage with private and public volume modes. Private volumes are encrypted client-side, so Relay Nodes see ciphertext; public volumes are intentionally plaintext and world-readable. **CapTokens** bind scoped mount rights to a Runner address, and **manifest roots** are anchored on-chain at the **Storage Manager** system actor (**0x0A**, §9).
- **CBSS (Cowboy Secret Service):** Threshold-encryption service that releases an account's secrets to authorized Runners without ever exposing plaintext on-chain (Secrets Manager system actor, **0x04**, §9).
- **CBQS (Cowboy Queue System):** Provider-served durable streams and queues for workload coordination. The Stream Registry (**0x17**, §9) records stream and provider lifecycle and prepaid-balance settlement keyed by `(owner, provider)`; the broker remains authoritative for accepted records and consumer progress. Payload confidentiality is an application responsibility rather than a platform service — applications that need it encrypt before appending — while topology and operational metadata remain visible at their respective layers.
- **Entitlement:** A permission governing an actor's or runner's capabilities.

0.3.2 Key Features

Cowboy combines six core technical features:

- Deterministic Python Actors. Smart contracts on Cowboy are actors: Python programs with actor-scoped state, a mailbox for asynchronous messages, and chain-native timers. Validators execute actor state transitions in a sandboxed Python VM (PVM) pinned to Python 3.11.8 with deterministic floating-point (softfloat), disabled GC, fixed hash seed, and a whitelisted module set. Actor state is replicated consensus state, not a confidentiality boundary. Reentrancy is depth-capped to 32; per-call memory is bounded at 10 MiB.
- Native Timers & Scheduler. Actors schedule their own future execution with one-time timers (`set_timer`) and the SDK's `@every` repeating schedules without external keeper infrastructure. The scheduler uses a per-fire `fee_payer` pre-charge and refund. CIP-5 FIFO ordering and the timer-lane EIP-1559 basefee are normative. Natural timer fires pay no priority tips; the CIP-1 auction is retired (§5.1b). Anti-DoS measures include the per-fire pre-charge, a non-refundable per-timer scheduling cost, a `max_timers_per_actor = 1,024` cap, and a dedicated timer execution lane (`LANE_TIMER_CYCLES = 8,800,000`, 11% of the 80,000,000-cycle block cap; see §13).
- Protocol-Mediated Off-Chain Compute. A decentralized Runner marketplace executes jobs (LLM inference, HTTP fetches, MCP tools, custom compute, and one-shot container jobs) off-chain. Runners stake CBY and are selected via VRF. Results are admitted under developer-selected trust profiles such as N-of-M quorum, economic bond, TEE attestation, structured matching, semantic similarity, or ZK proof verification. The selected Runner is inside the plaintext boundary unless a stronger execution profile supplies additional isolation. A commit-reveal protocol with a `DISPUTE_WINDOW_BLOCKS = 75`-block challenge window ($\approx$ 75 s at 1 s/block) and slashing rules provides economic accountability.
- Near-chain services. The architecture defines capability-scoped private and public CBFS volumes, CBSS secret and private-volume-key release to authorized Runners, and CBQS durable streams and queues without placing every data-path operation in consensus. Each subsystem specifies its authority, confidentiality, recovery, and provider-replacement boundary.
- Application and Commerce Rails. System actors and protocol interfaces provide session payment, route binding, event subscriptions, container policy, and actor hiring and distribution. These interfaces make applications addressable and economically composable without granting a privileged store or gateway protocol authority.
- Tri-Metered Gas. Three independent fee markets price computation (**Cycles**), consensus store I/O (**Accesses**), and persisted data (**Cells**) separately (introduced by CIP-43). Each meter uses a basefee that adjusts dynamically with demand and is burned; tips go to proposers. This prevents cross-subsidization among compute-heavy, I/O-heavy, and storage-heavy workloads.

0.3.3 Differences vs. Ethereum

Aspect	Ethereum	Cowboy
Execution	EVM bytecode (Solidity)	Python actors in sandboxed PVM
Fees	Execution gas plus a separate EIP-4844 blob-gas market	Tri-metered execution/state fees (Cycles + Accesses + Cells)
Scheduling	Application or rollup-specific; L1 contracts commonly use external automation	Native FIFO timers with live-basefee payment
Off-chain compute	Application-selected oracle, coprocessor, or rollup services	Protocol-mediated Runner marketplace
Storage, secrets, coordination	Application-specific cloud services	Owner-controlled, capability-scoped near-chain services
Distribution and routing	External platform contracts and servers	Protocol registries for sessions, routes, subscriptions, and hiring
State management	Current L1 state is persistent; statelessness and state expiry remain active research	Rent with eviction and restoration

0.3.4 Application Shapes

Cowboy supports two named application shapes. A consensus application centers deterministic actor execution and replicated state. An owner-controlled application pairs chain-anchored ownership and authority with predominantly provider-served execution and data: the actor remains the durable identity and application-level control point; chain state records the grants, commitments, and settlement required by each subsystem; Runner jobs or persistent workloads carry most executable logic; and CBFS, CBSS, and CBQS carry high-volume data and coordination. A provider may remain authoritative for assigned operational state, such as CBQS accepted records and consumer progress. These are useful poles, not a mechanical classification rule: execution locality alone does not establish owner control, and most applications combine both shapes. This taxonomy is architectural and non-normative.

0.3.4.1 Owner-Control Criteria A subsystem keeps operations under the owner's control only to the extent that the owner can export its complete state, replace the provider without incumbent cooperation, recover after provider loss, revoke old authority, and independently verify the claims on which recovery or payment depends.

Chain ownership alone does not establish those properties. Each near-chain specification identifies which criteria it provides, which operational state remains provider-authoritative, and what data or service is unavailable if the provider disappears. The term **owner-controlled application** is an ownership and architecture objective, not a guarantee that every provider path is permissionless or portable.

0.4 Accounts and State

Cowboy distinguishes two object types:

- **External Accounts (EOAs):** Controlled by private keys. They initiate transactions and hold balances of CBY and other assets. A future protocol upgrade may add passkey or other WebAuthn-compatible key management.
- **Actors:** Autonomous Python programs executed in the PVM. Actors own storage, receive messages, and can send messages to other actors.

Each object has a 20-byte address. Actor addresses contain a two-byte address group followed by an 18-byte hash suffix derived from the creator address, salt, and code hash. The **world state** is a mapping:

`State : Address → { balance, nonce, code_hash?, storage?, metadata }`

where actor storage is a key/value map subject to per-actor caps and rent. System actors and precompiles occupy a reserved prefix of the address space (0x0000…0100); low-order allocations for the runner and fee subsystems are listed in **§9**.

0.5 Transactions and Message Passing

A user interacts with Cowboy by sending a signed **transaction** specifying a destination, a payload, and resource limits for **cycles**, **accesses**, and **cells**, alongside maximum and tip prices for each.

An actor interacts with other actors by sending **messages**. Messages carry a payload, may transfer value, and may trigger further messages. Every committed message is processed **exactly-once after finality**; before finality processing is **at-least-once** and may be reverted. The recipient's handler runs only if the sending actor (or its owner under `FallbackToOwnerBalance`)

can cover the message's maximum fee at delivery; otherwise the message is consumed with `DeferredPayerUnfunded` (1015) and not retried (§4.5). Handlers MUST be idempotent. Actors may schedule **timers** that insert messages at a future block height. To avoid denial-of-service through explosive fanout, a transaction (including all nested sends) MUST NOT enqueue more than **1,024** messages. This `1,024` cap governs **cross-actor message sends** specifically; event-hook emits (the publish/subscribe notification path, where enabled) are a distinct operation with their own, separate and much smaller per-transaction budget; the two are not drawn from a shared pool.

0.5.1 Native Timers and the Actor Scheduler

Cowboy provides protocol-native timers, eliminating the need for external keeper networks. Actors schedule messages to themselves or other actors at a future block height or on a recurring interval. The normative CIP-5 FIFO scheduler is described in §5.1a. The former CIP-1 auction requirements are retired (§5.1b).

0.5.1.1 5.1a FIFO Scheduler

The FIFO scheduler uses:

- **Storage:** timers are indexed by `height`; same-height bucket is FIFO by insertion order (no priority queue, no tiered calendar).
- Per-fire `fee_payer` model. A scheduled actor timer records who pays for each fire (`fee_payer`), how much computation that fire may consume (`gas_limit_per_fire`), and its structural committed-store access request (`access_limit_per_fire`). The classic scheduler stores the 160-access structural default. Extended timers may opt into the 322-access general child tier or the 375-access actor-Intent tier under FIFO scheduling. The Timer phase contains 887 charged accesses per block. At identity these tiers admit five/two/two complete fires; at launch and maximum scalar they admit four/two/one. Smaller declared requests may admit more. Access requests are scaled before admission; there is no auction-specific access clamp.

  Before execution, the protocol escrows the declared cycle and cell maximums plus `scale(access_limit_per_fire, p)` charged accesses from `fee_payer`. It then reserves the complete phase-effective cycles and `scale(min(access_limit_per_fire, phase_cap), p)` charged accesses. A fire that does not fit carries forward intact and never executes against a partial remainder. After execution, the protocol burns the live-basefee price of actual use, pays no priority tip, and refunds unused escrow to `fee_payer`. The charged execution envelope follows the live scalar.

- **Three-path lifecycle:** natural fire / TTL expiry / insufficient-funds self-destruct, plus explicit `cancel_timer` and validator-set `SYS_CANCEL_TIMER`.
- **Lane separation:** `LANE_TIMER_CYCLES = 8,800,000` (execution lane, 11% of the block cap) is separate from `TIMER_GC_CYCLES` (cleanup lane) so a TTL-expiry storm cannot starve live execution.
- **Per-actor limit:** `max_timers_per_actor = 1,024` active timers (default, governance-tunable).
- **Per-fire caps:** `max_cycles_per_fire = 550,000`, `max_cells_per_fire = 550,000`.
- **Same-block prohibition:** timers created in the current block MUST NOT fire in the same block.

The FIFO scheduler has no deferred-timer anti-starvation boost.

0.5.1.2 5.1b Auction Retirement

CIP-5 FIFO is the only timer inclusion policy. The CIP-1 auction activation parameter, priority tips, GBA estimator, priority tiers, fairness-weight ranking, and 250,000-cycle auction cap are removed. The per-fire cycle ceiling remains 550,000. Reserved Timer bid fields and host-ABI slots remain encoded but cannot affect admission, ordering, or fees.

The timer-lane EIP-1559 basefee and fee multiplier continue to apply. Natural fires retain per-fire pre-charge, live-basefee settlement, and refund without crediting any proposer tip. Successful fires continue to write the existing per-actor 1,000-block `FairnessCounter`; these writes preserve FIFO consensus state and do not provide a scheduling boost.

0.5.1.3 DoS Prevention

Attack Vector	Mitigation
Schedule millions of timers	`max_timers_per_actor = 1,024` + per-fire `fee_payer` pre-charge
Sybil attack across many actors	Per-block execution budget caps total timer work (`LANE_TIMER_CYCLES`, ~11% of block cap)
Timer bomb (many timers, one block)	Per-tx fanout cap (**1,024** enqueued timers/messages) + `max_timers_per_actor = 1,024` + non-refundable 200-cycle scheduling cost per timer; the same-block prohibition stops any newly scheduled timer from firing in the creating block

Attack Vector	Mitigation
Fill queue far in advance	Per-fire `max_cost` re-checked at each fire; timer-lane basefee responds to utilisation
DoS then cancel for refund	Pre-charge happens at fire time, not at schedule time; scheduling cost (200 cycles) is non-refundable

0.6 Asynchronous Execution and Multi-Block Semantics

0.6.1 Single-Block Atomicity

Cowboy provides **atomicity only within a single block**. Actor-state writes, outbound messages, timers and `create_deferred_tx` children share the emitting frame's rollback boundary. Intent-authorized deferred work and async event-fire requests have the caught-failure survival rules in §3.3; receipt events can also record a failed frame. There is no cross-block atomicity. Once a handler completes and the block is finalized, subsequent handlers execute in potentially different world state.

Access accounting MUST use the same transaction snapshot boundary as state and messages. At settlement, `accesses_used <= access_limit` MUST hold, and the unused access limit MUST be refunded at `max_fee_per_access`.

0.6.2 Why No Cross-Block Transactions

Cross-block atomicity would require either global locks (destroying parallelism and creating deadlock vectors) or speculative execution with rollbacks (creating griefing opportunities and unpredictable costs). Cowboy explicitly rejects both approaches.

The fundamental problems with suspending execution across blocks are: (1) stale state — values read before the yield may have changed; (2) invalid control flow — branches taken on pre-yield state may no longer be appropriate; (3) composability explosion — nested yields create interleavings where each path depends on invalidated assumptions; (4) adversarial griefing — attackers can mutate state between yield points.

0.6.3 Message-Passing Continuation Model

Instead of implicit continuations, Cowboy uses **explicit message passing** for all asynchronous operations. When an actor dispatches an off-chain job, the result arrives as a separate message

in a later block. The actor must re-read and re-validate any state assumptions in the callback handler.

This model requires **explicit context capture** — any state needed in the continuation must be included in the outbound message. Correlation IDs enable matching responses to requests. If an actor sends multiple requests, responses may arrive in any order.

Actors MUST implement timeout handling for operations that depend on external responses. The recommended pattern combines correlation tracking with native timers: schedule a timeout timer alongside each outbound request, and cancel the timer when the response arrives.

Model	Atomicity	Developer Burden	Griefing Resistance
Ethereum (sync calls)	Single TX	Low	High
Cross-block locks	Multi-block	Low	Low (deadlocks, lock griefing)
Optimistic + rollback	Multi-block	Medium	Low (rollback spam)
Cowboy (message passing)	**Single block**	**Medium**	**High**

0.7 The Cowboy Actor VM (PVM)

Cowboy uses Python as its execution language. The PVM executes Python bytecode in a deterministic sandbox pinned to Python 3.11.8, with restrictions to ensure identical results across all nodes.

0.7.1 Python Compatibility Boundary

"Python 3.11.8" identifies the targeted language version. It does not imply byte-for-byte CPython or PyPI compatibility. The PVM is a RustPython-derived deterministic runtime maintained as consensus-critical code. It supports the whitelisted language features and standard-library modules below. It does not support CPython C extensions, native wheels, dynamic imports, threads, networking, ambient filesystem access, or packages that depend on those facilities. Model training, NumPy/PyTorch workloads, browser automation, and other native or host-dependent Python execute as Runner jobs. PVM upgrades require deterministic conformance vectors, differential execution tests, and coordinated validator activation.

0.7.2 Determinism Guarantees

Access-meter determinism:

- Every access charge MUST derive only from committed or input-visible quantities. It MUST NOT depend on observed latency, actual hardware cache warmth, validator hardware, wall-clock time, or timeouts.
- Charging MUST occur before dispatch. Access exhaustion MUST fail closed into the committed `OutOfGas` outcome; a wall-clock reap is only a non-committing infrastructure backstop.
- A nested revert MUST preserve the transaction warm set and committed-value cache. Only cache mutations caused by staged writes unwind, in reverse order, to their pre-write values.
- Zygote is the normative production engine; its committed `accesses_used`, receipt bytes, and state roots are canonical. The in-process engine is a test-only reference oracle. Equivalent host-call traces under the same metering policy and access weights MUST produce equal `accesses_used`; receipt/state comparisons MUST record differences and resolve them in Zygote's favor. Known differences are recorded; the compared fork cases show unchanged Cells and Accesses, and the reference vector shows complete state-root, receipt-root, and encoded-receipt equality. Full receipt/state equality across the entire host-call corpus is not established (CIP-43 §12).

Runtime Environment:

- The PVM only interprets code; it has no JIT compiler.
- Deterministic memory management via reference counting. The cyclic garbage collector is disabled; creating reference cycles MUST raise `DeterminismError`.
- Fixed recursion limit: **256** (integrated with cycle metering). Native recursive container walks also raise catchable `RecursionError`; container destruction uses bounded native stack. Checkpoint encoding and decoding share a depth bound of 256 (root depth zero), refusing deeper graphs and CBOR with `RecursionError`.
- Deterministic Python strings contain Unicode scalar values. Unpaired-surrogate construction raises catchable `ValueError` or `UnicodeError`, including source escapes and decoding handlers; see CIP-3 §2.2.

Numeric Determinism:

- All floating-point operations use a cross-platform software math library (softfloat), not the host FPU.
- Transcendental functions (`sin`, `cos`, `log`, `exp`, etc.) use deterministic softfloat implementations.

- If the `decimal` module is used, it MUST use fixed rounding mode (`ROUND_HALF_EVEN`) and fixed precision.
- **Integer width cap (CIP-3 §2.4):** integer bit-length MUST NOT exceed **4096 bits** (`MAX_INT_BITS = 4096`). Enforced at the deploy gate (constant-folded `2**N` / large decimal-literal scan) plus a runtime `_GuardedInt` preamble; overflow raises `OverflowError`. This bounds worst-case bigint compute so it is deterministic and gas-bounded across nodes.

Collection Ordering:

- `PYTHONHASHSEED` fixed to `0`.
- `dict` iteration is insertion-ordered (deterministic per Python 3.7+).
- Built-in `set` and `frozenset` are replaced with `ordered_set` from `cowboy_sdk.collections` (transparent to user code).

Serialization:

- Bytes that are signed, hashed, or committed MUST use the canonical codec owned by that boundary's specification. Consensus types use `commonware-codec` (§2.5; Appendix A); SDK/PVM values use CIP-6 §11.5; Runner-connectivity frames use CIP-11 §12.6; and job-result canonicalization uses CIP-23 §3.5.1a-§3.5.2. Implementations MUST NOT substitute one profile for another because their map-ordering rules differ.
- Decoders MUST enforce the selected profile and, where applicable, reject indefinite-length items, non-minimal integers, duplicate or incorrectly ordered keys, unknown tags or discriminants, and trailing bytes.
- Non-integral values MAY cross a canonical boundary only when its profile defines canonical finite `binary64` handling. `NaN` and infinities MUST be rejected; integral values, including either sign of zero, MUST normalize as the profile requires. Consensus quantities remain integer or fixed-point, and deterministic cross-Runner results forbid non-integral floats.
- `pickle` is forbidden. JSON is not a canonical wire format unless the boundary specification defines its canonical conversion.

Allowed Module Set:

`collections`, `dataclasses`, `enum`, `functools`, `itertools`, `json`, `re`, `struct`, `math`, `decimal`, `typing`, `abc`, `hashlib`, `cowboy_sdk`. No C extensions. No dynamic imports. Additional modules may be added via governance after determinism audit.

Forbidden Operations:

Category	Forbidden
System	`sys.exit()`, `os.environ`, `os.system()`, `subprocess.*`
Time	`time.time()`, `datetime.now()`, `time.sleep()`
Randomness	`random.*` (use `cowboy_sdk.vrf` instead)
Networking	`socket.*`, `urllib.*`, `http.*`, `requests.*`
Filesystem	All except `/tmp` scratch space (256 KiB limit, wiped post-handler)
Reflection	`eval()`, `exec()`, `compile()`, `globals()` modification
Introspection	`sys._getframe()`, `inspect.currentframe()`, `gc.*`
Weak References	`weakref.*` (non-deterministic collection timing)
Threading	`threading.*`, `multiprocessing.*`, `concurrent.*`
Identity	`is` comparisons on strings or numbers (use `==`)

0.7.3 Storage and State Persistence

The storage architecture uses a three-layer model:

1. **The Ledger:** An append-only log records the block history.
2. **The Triedb:** An authenticated Merkle trie (QMDB) generating a verifiable `state_root` per block via **blake3**, holding all account, code, and storage state. (Transaction and receipt roots use keccak256.)
3. **Auxiliary Indexes:** Rebuildable, read-optimized tables for transaction hashes, event topics, etc. Not part of the consensus-critical state root.

Cross-VM Compatibility. A `vm_ns` (VM namespace) flag in storage keys allows PVM and future EVM storage to coexist without collision at the same address. A standardized C-ABI wrapper enables cross-VM calls.

Persistent storage of non-reserved actors is subject to **state rent** (see §4.4 and the Data Availability section below).

0.8 Pricing: Cycles, Accesses, and Cells

Cowboy splits application execution pricing into three independent meters:

- **Cycles** measure compute: Python operations and host calls each have a fixed cost defined in a consensus-critical cost table. Like Erlang reductions, cycles bound handler execution with a budget of discrete steps.
- **Accesses** measure consensus store I/O: one cold `StateGet` is one access, and other operations use the weighted primitives in §17.3.
- **Cells** measure bytes: calldata, return data, blobs, and storage writes all consume cells (1 cell = 1 byte).

Each block adjusts three basefees (one per meter) dynamically based on demand. Users specify max prices and optional tips for each meter. Basefees are burned; tips go to proposers.

Runners set off-chain execution prices in a free market, and actors escrow CBY at submission. On-chain gas metering does not apply to that execution.

0.9 Off-Chain Compute: The Runner Marketplace

Actors can send LLM inference, HTTP fetches, MCP tool invocation, heavy transforms, and **one-shot container jobs** to a decentralized network of **Runners** who stake CBY. The chain admits results under a trust profile chosen by the developer. Those profiles provide different evidence and trust assumptions; they do not make every off-chain execution confidential or cryptographically proven.

Jobs MAY declare `volume_mounts`: Runners with storage capability mount **CBFS** volumes through scoped **CapTokens**. Private volumes are encrypted client-side, so Relay Nodes see ciphertext; public volumes are intentionally plaintext and world-readable. Manifest roots are anchored on-chain at the **Storage Manager** (**0x0A**). This supports large artifacts and multi-step tool chains without putting raw payloads in calldata. The full design lives in the Cowboy Storage Whitepaper.

Jobs frequently need third-party credentials (API keys, signing tokens). The **Cowboy Secret Service (CBSS)** lets an account store secrets encrypted so that the plaintext never appears on-chain and is never exposed to the proxy committee or the dispatcher. Secrets are released by an open threshold proxy committee using **threshold identity-based encryption** (a t-of-n threshold-BLS construction on BLS12-381): the proxies return partial signatures that an authorized Runner combines to decrypt the secret just-in-time at execution. Only that authorized Runner sees the plaintext; the dispatcher is never on the key path. The trust root is t-of-n proxy non-collusion plus chain-recorded registration, staking, and accountability, not a hardware enclave. CBSS is also the key-delivery layer for private CBFS volumes. The **Secrets Manager** system actor (**0x04**, §9) holds proxy registration, per-account key material, per-secret access policy, and release receipts; the full design lives in the Cowboy Secrets Whitepaper.

Workload coordination. **CBQS** provides durable streams, queues, replay, consumer groups, and one at-least-once delivery class for off-chain workloads. Messages flow through broker RPC without a chain transaction per record. The **Stream Registry** (**0x17**, §9) records stream and provider lifecycle and prepaid-balance custody and settlement keyed by `(owner, provider)`, while the selected broker is authoritative for accepted records and consumer state. The CBQS Whitepaper and CIP-39 define the protocol.

One-shot container jobs use the Runner job lifecycle and on-chain image policy. Persistent workloads use registry records, health leases, Gateway routing, Runner-side provisioning and supervision, rescheduling, and third-party billing.

Job Lifecycle:

1. Actor submits a job with escrowed price, resource bounds, and trust profile.
2. One or more Runners are selected via stake-weighted VRF from eligible (staked, healthy) Runners. The request binds before the selection round, and the draw uses a future round beacon so the transaction author cannot grind a favorable assignment from known inputs.
3. Runners return `commit = keccak256(output || salt)`.
4. Runners reveal `{output, salt, proof?}`.
5. A `DISPUTE_WINDOW_BLOCKS = 75`-block window opens (≈ 75 s at 1 s/block); challengers post a 100 CBY bond.
6. Proven dishonesty triggers slashing; operational failures result in reputation penalties only.
7. 89% of job payment to runner(s), 10% burned, 1% to Treasury.

Trust Profiles:

Mode	Level of Trust
N-of-M Quorum	Committee agreement limits single-Runner faults; it does not prove truth or operator independence.
N-of-M with Dispute	Bonded challenges penalize only violations covered by objective failure rules.
TEE Attestation	Execution within a Trusted Execution Environment; attestation verified on-chain.
ZK-Proof	Runners provide zk-SNARKs for cryptographic verification.

Entitlements match Runners and actors through declared permissions governing capabilities such as TEE requirements, data residency, supported models, **container network egress**, and domain allowlists for HTTP (see §15).

0.9.1 Runner Resource Accounting

Every job submission includes explicit resource bounds (max tokens, max wall time, max memory, max price). Runners publish rate cards to an on-chain registry. Job pricing follows: `actual_payment = min(reported_usage × rates, max_price)`. Tips incentivize priority during high demand.

Runner-reported usage is trusted by default. Non-TEE container billing retains the `DISPUTE_WINDOW_BLOCKS = 75` challenge window; its billing-dispute remedy is a reputation penalty and charging the full escrow, not stake slashing. A single-runner settlement with metered, verified TEE billing is final and eligible for payout at H+1; billing disputes against that final record are rejected (CIP-10 §12.3; CIP-23 §3.12). The Nitro signer proves key custody, not the integrity of host-measured cgroup meters under host compromise. In both paths the charge remains capped at the locked escrow. Result-fraud challenges have their separately specified objective failure and slashing rules.

Payment and Failure Handling:

Outcome	Runner Payment	Actor Refund
Success	89% of `min(reported_usage × rates, max_price) + tip` (10% burned, 1% Treasury)	`max_price - actual_payment`
Runner fault (timeout, invalid result, crash)	0	100% of escrow
Impossible job (bounds too tight)	Pro-rata based on progress	Remainder of escrow
Actor fault (malformed input)	Minimum fee (gas cost recovery)	Remainder of escrow
External fault (API down, model unavailable)	Pro-rata based on progress	Remainder of escrow

0.9.2 LLM Result Verification

The same LLM prompt can produce semantically equivalent but byte-different outputs. Cowboy provides multiple verification modes:

Similarity is evidence of clustering, not truth, model provenance, or independent execution. Economic bonding is delivery assurance unless an objective slash condition applies.

Mode	Runners	Verification	Challenge Scope	Use Case
`none`	1	None	Non-delivery only	Prototyping, low-stakes
`economic_bond`	1	Trust-first (single result)[1]	Non-delivery only[1]	Subjective generation
`majority_vote`	N-of-M	Vote on field value	Objective failures	Classification
`structured_match`	N-of-M	Verifier functions	Objective failures	Structured extraction
`deterministic`	N-of-M	Exact match + TEE	Full reproduction	Critical deterministic
`semantic_similarity`	N-of-M	Embedding threshold	Objective failures	Subjective with similarity

[1] `economic_bond` settlement is trust-first, matching `none`. It accepts exactly one result without objective-failure classification or a per-result slash. Its assurance is the Runner's slashable stake plus the §6 reputation and timeout machinery for non-delivery. A dispatch-time filter requires each selected Runner to hold $\geq 1.5\times$ the job's `max_price` in effective stake across its active book, but no bond is earmarked against the specific result. The *Objective Failure Criteria* table applies only to the quorum modes (`majority_vote`, `structured_match`, `deterministic`, `semantic_similarity`).

These modes provide **economic assurance**, not cryptographic correctness, except where TEE or ZK verification is explicitly required. The protocol guarantees execution integrity, not output quality — quality is a market outcome.

Objective Failure Criteria (automatically detected, no challenge required):

Failure	Detection	Consequence
Schema violation	Output fails declared JSON schema	Reputation penalty, no payment
Timeout	No result within `max_wall_time`	Reputation penalty, no payment

Failure	Detection	Consequence
Empty/garbage output	Output below `min_length` or fails entropy check	Reputation penalty, no payment
Non-delivery	Runner accepted job but never submitted	Reputation penalty, no payment
Wrong model	TEE attestation shows different model hash	**Slash** (proven dishonesty)
Prompt injection leak	Output contains system prompt markers	Reputation penalty, no payment

Only proven dishonesty (fabricated results, wrong model with TEE attestation) triggers stake slashing. Operational failures reduce the runner's reputation score, affecting future job selection, but do not destroy stake. "Fabricated results" includes losing a quorum vote with a valid comparison result: a losing canonical `majority_vote` cohort, a valid `deterministic` result outside the first valid reference cohort once it reaches threshold, a valid `semantic_similarity` embedding outside the winning cluster, or a losing chain-root bucket over roots, message hashes and deliverers. `structured_match` grouping dissent does not slash; schema violations and malformed comparison inputs incur reputation penalties and no payment. A missing or malformed embedding (including a zero or non-finite squared norm) and a self-reported embedding-model mismatch are operational failures, not TEE-proven wrong-model execution.

A committed Runner that has not revealed when settlement, timeout or committee replacement closes its assignment receives the non-delivery reputation penalty and no payment. Its stake is unchanged if its reveal window is still open (including the deadline), regardless of crash attestation. Only an expired, unattested commitment receives the configured proven-non-reveal slash; a crash attestation exempts that slash but not the reputation penalty. A committee member that never commits also receives the operational penalty at settlement once its current assignment's commit window closes; a fresh replacement still inside that window is exempt. PublishChainRoot admission rejects malformed roots, wrong/missing heights, malformed or duplicate delivery fields, and invalid required anchor signatures or blob lengths. Signature and source-kind blob-length checks require the source and destination chain entries, not an adjudication declaration; configuration loss skips only checks whose inputs are missing. If required registry configuration or usable adjudication is missing, reveals passing those available checks remain stored and pending even at threshold, with no settlement payment or anchor publication. Ordinary timeout/refund proceeds and accepted revealers remain neutral. A runner-fault rejection stores no result; absent a later accepted reveal, the runner follows these non-reveal rules. Verifier-side filters are defence

in depth and do not provide lifecycle exemptions.

A successful replacement can receive a 5% bonus from the original Runner's stake only when the timeout transition recorded a slashable disposition: an expired unattested non-reveal or proven dishonesty. Operational timeout and in-window or crash-attested non-reveal preserve the original stake through replacement settlement and pay no rescue bonus; the replacement still receives ordinary job payment. Settlement uses the recorded disposition and never reclassifies the original using its later height.

0.9.3 External Data and Oracle Semantics

Actors frequently need external data: price feeds, web APIs, public datasets. External data is inherently mutable and non-deterministic. Different sources require different verification strategies:

Source Type	Characteristics	Verification Strategy
Deterministic API	Versioned, stable, structured (blockchain RPC, static files)	Exact match
Semi-stable API	Structured with variable metadata (REST APIs)	Structured match, ignore metadata
Time-series data	Values change over time (price feeds)	Median/majority with freshness bounds
Web scraping	Unstructured, highly variable (HTML pages)	Extraction-based matching
Authenticated endpoints	Requires credentials	Single runner + TEE

Actors specify freshness constraints with a reference mode (`block`, `submission`, or `absolute`) and a `max_age_seconds` parameter.

When multiple runners fetch mutable data, the protocol selects a canonical result via one of: `first_valid` (first runner's result is authoritative), `median` (for numeric data), `majority` (for categorical data), or `latest` (most recent by timestamp).

Extraction-Based Verification. For web scraping, runners apply extraction rules (CSS selectors, XPath, JSONPath, regex) and submit extracted data rather than raw HTML. Verification compares extracted fields, ignoring irrelevant page differences.

HTTP access is governed by the Entitlements system. Actors declare required domains; runners advertise supported domains. The protocol provides curated domain sets (`price_feeds`, `government_us`, `social_apis`, `blockchain_rpc`).

0.10 Randomness

Consensus emits one random beacon per **round**: the `commonware` threshold scheme produces a verified `Seed { round: Round(epoch, view), signature }`, a BLS12-381 threshold signature over the absolute round encoding. Exactly one valid seed exists for a given absolute round, whether that round notarizes, finalizes, or nullifies, and it verifies against the epoch's threshold public key. Seeds are identified by their **absolute round**, never by block height: a height `n` and a `Seed.round` are not interchangeable, and "the seed of whatever round finalizes height `n`" is not a well-defined beacon. The execution-readable beacon for a block is the **parent beacon**: the verified seed of the absolute round immediately preceding the round in which the block was proposed (a round-relative object that exists even if that preceding round nullified; never a height-addressed lookup), because using a round's own randomness to affect execution in that same round is unsafe. Actors access sub-randomness via `HKDF(R, label)` for fair committee sampling, lotteries, and games; protocol components that consume the beacon (e.g. runner selection, CIP-11 §9.2) MUST name the absolute round whose seed they consume.

0.11 Consensus and Networking

Cowboy uses **Simplex BFT** with proof-of-stake. Key parameters: ~1-second blocks, ~2-second finality, fault tolerance up to $f < n/3$ Byzantine validators.

Consensus Flow:

1. **Propose:** The round's VRF-elected leader broadcasts a proposal.
2. **Notarize:** A recovered threshold certificate over 2f+1 notarize votes produces a `Notarization { proposal, certificate }`.
3. **Certify application:** Validators deterministically execute and certify the proposed state transition.
4. **Finalize:** A recovered threshold certificate over 2f+1 finalize votes produces a `Finalization { proposal, certificate }`, finalizing that proposal and its ancestors.
5. **Nullify:** A timed-out or uncertifiable round produces a nullification certificate and advances without a block.

Under partial synchrony, the protocol preserves safety and regains liveness once network delay is bounded. Leader election and all certificates are addressed by absolute consensus round.

0.11.1 Validator Set

The validator set is open and permissionless. Requirements: stake $\geq$ `minimum_validator_stake` (governance-tunable), self-stake only (no delegation at launch), compliant validator software. No protocol cap on validator count.

A validator registers by staking CBY and submitting a BLS12-381 key. It activates at the next epoch boundary, then proposes, votes, and earns rewards. To exit, it signals unbonding and withdraws after the 7-day unbonding period.

Epochs: 3600 blocks (~1 hour). Validator set updates, slashing penalties, and epoch randomness derivation occur at epoch boundaries.

0.11.2 Staking and Rewards

Block rewards (from inflation per §8.2) are distributed proportionally to stake. Proposers additionally receive **100% of cycle/cell tips on blocks they finalize**. No per-transaction validator commission or surcharge is charged to users beyond tips; the entire user-derived validator income is the tip stream. Staking is self-bonded only; delegated stake does not contribute to consensus weight.

0.11.3 Slashing

Most offenses result in jailing, or temporary removal, rather than stake destruction:

Offense	Penalty
Double signing	Jail + slash 1% of stake
Proposer equivocation	Jail + slash 1% of stake
Extended downtime (>50% of votes over 1000 blocks)	Jail (no slash)
Invalid block proposal	Jail (no slash)

Jailed validators must wait 24 hours before unjailing. Repeated offenses increase jail duration exponentially.

0.11.4 Network Layer

Transport: QUIC over TLS 1.3 (required). Gossip: transactions flood to all peers; blocks are relayed by validators; votes go directly to the proposer. Peer discovery: DHT-based with

bootstrap nodes.

0.11.5 Dedicated Lanes

Block space is partitioned into dedicated lanes with reserved capacity and per-lane fee multipliers:

Lane	Reserved Capacity	Fee Multiplier	Contents
System	41% (32.8M cyc)	1.0×	Validator updates, governance, slashing
Timer	11% (8.8M cyc)	1.0×	Scheduled timer executions
Runner	20% (16M cyc)	1.0×	Runner job results, attestations, and related on-chain follow-ups (e.g. storage manifest anchors)
User	28% (22.4M cyc)	1.0×	User-initiated transactions

Lane cycle budgets are enforced independently; unused capacity is not reassigned across lanes. Transactions are tagged by type at submission; the proposer cannot reassign lanes. For ordinary transactions, the lane multiplier scales the global Cycle/Cell/Access rates. Natural-timer fires use their separate EIP-1559 cycle basefee and the global cell/access rates, all scaled by the Timer multiplier (§4.2). All four multipliers default to `1.0×` at genesis (no lane subsidy or surcharge). The multipliers are Tier-0 governance-tunable; see §13.1 for the canonical parameter registry.

0.11.6 MEV Reduction

Cowboy's MEV design uses four bounded mechanisms: (1) VRF-based leader election limits deterministic leader capture but can select the same validator in consecutive views; (2) VRF-based transaction ordering constrains placement within a certified eligible set; (3) ~2-second finality limits reorg and time-bandit windows; and (4) lane isolation prevents congestion in one lane from consuming another lane's reserved capacity. The mempool remains public at launch.

This design does **not** address single-block proposer censorship, private orderflow MEV, or JIT MEV against actors with predictable on-chain state. Application-level mitigations (commit-reveal, slippage caps, batch auctions) remain an actor concern. The full enumeration and rationale are in §6.5.

0.12 Data Availability, State Rent, and Storage

0.12.1 Inline Data vs. Blobs

Small outputs ($\leq$ 64 KiB) are stored inline and paid for with cells. Larger artifacts MUST be stored as content-addressed blobs (e.g., IPFS) with the multihash referenced on-chain.

0.12.2 State Rent

Persistent storage of non-reserved actors is subject to **state rent**, an ongoing fee for the bytes an actor keeps in the global state trie. Rent bills the bytes actually stored (`kv_bytes`, the per-actor counter committed in the state root), never fewer than 1 KiB: each rent epoch costs `max(kv_bytes, 1 KiB) × rent_rate`. The genesis rate is **2,739,726,027,397 atto per stored byte per rent epoch** (approximately **0.001 CBY/byte/year** at 365 epochs). The rate is Tier-0 governance-tunable; changes take effect only at a rent-epoch boundary and apply prospectively. The 1 KiB floor means an idle actor storing nothing still pays and is eventually evicted.

Rent is settled from the actor's balance before each of its handlers runs and by a bounded per-block sweep, once per rent epoch (default: 1 day). Debt is kept in atto and the balance pays whole wei; a sub-wei remainder carries in the debt. Unpaid amounts accumulate as debt until repayment or eviction.

0.12.3 Debt and Eviction

When an actor cannot pay rent from its balance it stays fully functional while its debt accrues. Funding its balance pays the debt down at the next settlement. Once the unpaid debt exceeds **30 rent-epochs** (`eviction_threshold_epochs`, ~30 days) of rent, the actor is evictable: its storage and active timers are pruned, and its code, address, balance and a commitment to the pre-eviction storage are kept. The actor enters a "dormant" state.

Evicted storage can be restored by anyone who supplies the full pre-eviction storage, matching the recorded commitment, and pays the debt plus the 10% catch-up premium (`rent_catchup_bps = 1,000`).

0.12.4 Storage Caps

Every non-reserved actor is capped protocol-wide at **8 MiB** of stored bytes (`MAX_ACTOR_KV_BYTES = 8,388,608`) and **10,000** live keys (`MAX_ACTOR_KV_COUNT`), and pays rent only on the bytes it stores.

0.13 The State Transition Function

Given global state σ, block B carries mature-batch proofs, candidate transaction bodies, basefees (bf_c, bf_b, bf_a), and randomness R. The state transition returns the next state.

1. **Header/Proposer:** Simplex elects a leader for each consensus view. `R` is the verified seed of the absolute round immediately preceding the proposal round, never a block-height value or a value derived from the parent QC.
2. **Execute Mature Transaction Batches:** For each transaction-bearing lane, verify the finalized eligibility commitment and ordering seed defined in §6.4, then execute the committed transaction set in the resulting order within the **dedicated per-lane budget** in §6.3. Each lane's basefee is scaled by its governance-tunable `lane_fee_multiplier`. For each transaction: validate signature, nonce, and balance; initialize meters; dispatch to target (fanout $\leq$ 1,024, reentrancy depth $\leq$ 32); enforce memory (10 MiB), mailbox ($\leq$ 1,000 messages, `MAX_MAILBOX_LEN`), and per-actor storage caps; deduct fees and burn basefees.
3. **Deliver Timers:** Inject due timers at height(B).
4. **Resolve Jobs:** Process commitments, reveals, challenges, and payouts.
5. **Admit Candidate Batches:** Validate the candidate bodies carried by B for the System, Runner, and User lanes, then write their pending-batch, nonce-reservation, and fee-escrow state as defined in §6.4. The Timer lane remains the end-of-block scheduler path in step 3 and does not use mempool batching.
6. **Adjust Basefees:** Update (bf_c, bf_b, bf_a) based on block utilization.
7. **Mint Rewards:** Distribute per-block inflation to validators.

0.14 Normative Conventions

This document uses **MUST/SHOULD/MAY** as defined in RFC 2119. Parameters marked **governance-tunable** can be changed by on-chain governance (see §11).

0.15 1. Accounts, Addresses, and Keys

1.1 Signatures.

External accounts MUST use **secp256k1** (ECDSA) with chain-id separation.

1.2 Actor address derivation (CIP-6 and CIP-27).

A deployment computes `h = keccak256(0xff || creator || keccak256(salt) || code_hash)`, where `creator` is 20 bytes and `code_hash = keccak256(python_source_bytes)`.

The final address MUST be `u16_be(group) || h[14:32]`. The group is `DeployActor.shard` when present, otherwise the creator's first two bytes interpreted big-endian. Explicit group zero is distinct from absence. The address remains 20 bytes with a 144-bit hash suffix.

The exact bytes in `DeployActor.code` are hashed and stored, as specified by CIP-6 §6.2. Consensus does not normalize Unicode, line endings or a BOM before hashing. `DeployActor.shard: Option<u16>` is the final instruction field after `manifest`; its encoding is an option tag and, when present, a fixed-width big-endian u16.

CIP-27 forks compute `h = keccak256(b"cip27.fork.addr.v1" || parent || u64_be(fork_count) || frame_entry_code_hash)` and use `parent[0:2] || h[14:32]`. They always inherit the parent's group. Under the CIP-6 §6.2 genesis execution rule, all groups use the same world state and call dispatcher; a group difference alone does not prohibit a synchronous call.

1.3 **System address space.**

The range 0x0000…0100 is reserved for **system actors and precompiles** (see **§9** for the normative allocation table of low-order system actors).

0.16 2. Transaction Types & Encoding

2.1 **Instruction-based tx (three meters).**

Cowboy transactions are **instruction-based**, not flat `to/value/payload`: the operation is a typed `Instruction` (System | Actor | Custom | Library) that carries its own operands (e.g. a transfer is `System(Transfer { to, amount })`). The authoritative transaction-v3 field order and exact encoding live in CIP-40 §7.1. The shell is `u8 version=3 ‖ u8 kind ‖ body`; only kind 0 is admitted. The body includes all three gas limits, all six fee/tip caps, the instruction and metadata, explicit `ChainIdentity` binding, per-account `AccountSignature` values, reserved `nonce_lane=0`, `sponsor=None`, unconditional `run_root: Option<Digest>` after `origin_remaining_access`, and authenticated delivery provenance. Addresses remain 20 bytes. Each slot declares its algorithm, mode, optional public key and signature bytes. Each account signature binds a policy revision and ordered key-indexed slots; every signer binds the entire roster. At genesis transaction signatures are secp256k1, with native and EIP-712 modes permitted in the same transaction. Access fees are `u128`, never narrowed to `u64`.

The reserved sponsor shape is `Sponsor { address, terms: SponsorTerms { max_fee: u128, expires_at_height: u64 }, signature: SignatureSlot }`; its address, both terms and slot descriptor are bound by the native signing preimage, with signature bytes excluded by the same zeroing rule as the other slots. A later sponsorship CIP defines escrow, refund and what is sponsorable; Some remains rejected at genesis.

2.2 **Validity checks.**

Every entry point uses CIP-40's context-aware verification. Before any recovery, validate version/kind, trusted identity, size, reserved defaults, provenance and slot policy; then complete separate whole-roster passes in this exact order: First check the roster, then the recovery ID, then low-S/scalars. Next run CIP-3 signature preflight, then identity-access preflight. Only then hash and recover. CIP-40 §7.2 fixes every rejection token. Additional signers are sorted, unique, bounded and may not repeat `from`. Ordinary signed transactions require `run_root`, `access_list` and `origin_*` fields to be None and carry delivery `Transaction`. Unsigned deferred children carry all four origin fields, bounded cycle/cell/access allowances and authenticated engine provenance; external ingress rejects them. Structural validity alone is not proof of origin. Verification returns recovered indexed keys and signed revisions; admission and execution require the current account policy. Stateful nonce, balance and wallet/actor checks run against the current execution view even if cryptographic results were pre-verified. Block validation enforces CIP-3 §2.5's aggregate signature work/byte budgets before recovery. It enforces the access class ceilings before lane accounting (CIP-43 §8). Nonzero-origin deferred authority is converted to charged units and checked against `PROTOCOL_TX_ACCESS_LIMIT = 4,096`; signed transactions use their smaller class ceilings. A node-local override may lower the User ceiling through `MAX_TX_ACCESS_LIMIT`; that override remains admission policy only.

The CIP-49 account format and canonical typed projection are specified in CIP-40 §§7.1–7.3. For `k` additional accounts, pure preflight reserves `3 + 4k` structural identity accesses and rounds once at maximum scalar 1,350,358 ppm: the primary-only minimum is **5 charged accesses**, including its `AccountAuth` read. Client approval inspection uses schema **14**. Native account-signed objects append raw account20 and u64-BE policy revision to their ordinary body before chain-identity framing; session vouchers bind those fields in EIP-712 instead (CIP-40 §7.6). CIP-43 §8 owns the per-transaction lane-tail rule.

2.3 **Fee accounting.**

Let bc, bb, ba be the block basefees for cycles/cells/accesses. Fees are: `fee = cycles_used * (bc + min(max_priority_fee_per_cycle, max_fee_per_cycle - bc)) + cells_used * (bb + min(max_priority_fee_per_cell, max_fee_per_cell - bb)) + accesses_used * (ba + min(max_priority_fee_per_access, max_fee_per_access - ba))`.

For the access term specifically:

```
tip_a = min(max_priority_fee_per_access, max_fee_per_access - access_basefee)
```

```
fee_a = accesses_used * (access_basefee + tip_a)
```

Unused cycle and cell limits MUST be refunded at their corresponding `max_fee_*` rates. The unused access limit MUST be refunded at `max_fee_per_access`. One hundred percent of the

access-basefee portion MUST be burned; its priority tip goes to the proposer.

`ExecutionStatus::OutOfGas` MUST carry `accesses_used`. Access exhaustion MUST use that existing status and MUST NOT introduce a new `StructuredError` code because structured-error bytes participate in `receipt_root`. A transaction whose `max_fee_per_access` is below the block's `access_basefee` MUST fail with the existing `BASEFEE_TOO_LOW` (`1008`) structured error; its context MUST append `max_fee_per_access` and then `access_basefee` after `cell_basefee`, in that order. Transaction receipts MUST add `accesses_used`, `remaining_access`, and `cumulative_accesses_used` adjacent to their cycle and cell siblings in canonical `Write`, `Read`, `EncodeSize`, and RLP encodings, and receipt-root computation MUST commit those RLP bytes.

2.4 **Schema ownership.**

CIP-40 owns the outer shell and kind-0 transaction; CIP-33 owns the decode-only, consensus-disabled kind-1 body. CIP-28 owns card/Bank fee semantics; the v3 transaction carries no fee-payer field. CIP-43's four Access fields remain in the new schema. No transaction field order is independently maintained here; all consumers use the canonical protocol codec.

2.5 **Encoding and identity.**

Canonical commonware-codec encoding is ordered binary, not CBOR. Decoders reject non-minimal varints, invalid Option/bool tags, unsupported algorithms or modes, wrong lengths, unknown discriminants and trailing bytes. `decode(encode(tx))=tx` and every accepted byte sequence re-encodes identically. Origin remaining allowances use fixed u64 BE inside their Options. Delivery tags are Transaction 0, Call 1, Message 2, Timer 3, RunnerCallback 4, EventFire 5, Intent 6; other tags reject and execution checks the authenticated path. CIP-40 §7.1 defines the native signing and transaction-identity construction:

```
keccak256(b"cowboy.tx.v3" ‖ u64_be(chain_id) ‖ chain_instance_id[32]
          ‖ canonical_v3_encoding_with_slot_signature_values_zeroed)
```

Zeroing preserves slot descriptors and signature lengths. EIP-712 slots use the exact typed-data contract in CIP-40 §7.2; transaction identity remains the native construction regardless of the mode mix. All producers, receipt keys, deduplication, block commitments and replay share the same trusted chain identity. Fresh genesis accepts version 3 only, without old-version decoding, data conversion or activation heights. Signed objects use CIP-40 §7.6 identity-bound slots (with the anchor and EIP-712 session exceptions), and stored signing keys use §7.7 algorithm tags. `/chain-info` is the closed six-member startup identity; moved operational members come from extensible `/chain-parameters` bound to the same identity. Typed-data JSON uses JCS, decimal-string integers and lowercase hex. The conformance corpus is specified in Appendix A.

0.17 3. Execution Model (Actors)

3.1 Runtime & Determinism.

- Official SDKs: **Python SDK**. The runtime MUST enforce determinism:
  - **Allowed operations:** Standard Python operations, file I/O limited to `/tmp`; `async/await` is syntax sugar only and does not suspend across blocks.
  - **Forbidden:** `sys.exit()`, `random` module (except chain VRF), `time.time()/datetime.now()`, `os.environ` access, socket/network operations, subprocess calls, path traversal outside `/tmp`.
  - **Floating point:** Permitted; Cowboy provides a deterministic math library.
  - **Scratch space:** `/tmp` MUST be per-invocation, capped at **256 KiB** (counts toward `cells_used`), wiped post-handler.

3.2 Memory & Storage.

- **Per-call memory limit: 10 MiB** heap memory.
- **Per-actor stored bytes:** `MAX_ACTOR_KV_BYTES = 8 MiB` (8,388,608 bytes) caps the stored bytes (`kv_bytes`) of each non-reserved actor; **state rent** (§4.4) bills the bytes actually stored, with a 1 KiB floor.
- **Per-actor key count:** `MAX_ACTOR_KV_COUNT = 10,000` limits distinct live actor-storage keys per non-reserved actor (address `>= 0x100`), separately from the byte cap. New-key insertion at that limit fails with `StorageFull`. Reserved system actors (addresses below `RESERVED_SYSTEM_ADDRESS_LIMIT`, `0x100`) are exempt from both the key-count cap and the byte cap; see §9. Per-row key/value size limits apply to every actor, and `kv_count`/`kv_bytes` MUST be maintained for every actor.

3.3 Messaging, Reentrancy, Timers.

- **Delivery:** Every committed message is processed **exactly-once after finality** (before finality: **at-least-once**). The recipient's handler runs only if the sending actor (or its owner under `FallbackToOwnerBalance`) can cover the message's maximum fee at delivery (§4.5; CIP-43 §8). Otherwise the message is consumed with `DeferredPayerUnfunded` (1015) and not retried. Each message ID MUST be derived from the Message identity preimage below and recorded in a per-actor dedup set (counts toward actor storage or a separate metered mailbox store). Dedup entries MUST be retained for at least `dedup_window` after finality.
- **Sender:** A delivered message's handler observes its `creator` as `get_sender()` — the sending actor whose code called `send()`, including when that code ran inside a nested synchronous call. Messages and actor-queued deferred children (`create_deferred_tx` children, runner callbacks and timer fires) are attributed to the actor whose code queued

them through their `from`, and their handlers observe that `from` as the sender, the same rule as a signed transaction's signer. Async event-fire envelopes are delivered by the event-subscription system actor (`0x1D`); subscriber handlers observe the emitter (CIP-29).

- **Message identity:** `message_id = keccak256(creator[20] || producing_digest[32] || block_height_u64_be[8] || target[20] || keccak256(payload)[32])`, where `creator` is the actor whose code called `send()` and the bracketed numbers are byte lengths. For an ordinary producer, `producing_digest` is the immediate transaction digest.

  For CIP-29 asynchronous delivery, `event_fire_envelope_digest` is the transaction digest (`tx.digest()`) of the `__event_fire__` envelope transaction delivered from the event-subscription system actor (`0x1D`). The envelope carries CIP-29's `EmitOrigin`, encoded with `origin.to_metadata_bytes()`, in its metadata; `sub_id` is the 33-byte subscription id of the fire being executed, selected from `EmitOrigin.sub_ids`. **Event-fire metadata** means the envelope's complete metadata bytes, including any system-emission suffix appended when the envelope was created.

  For an async event-fire message with a **nonzero origin**, `producing_digest = keccak256(event_fire_envelope_digest[32] || sub_id[33])`. This binds the message to one subscription fire, including sends by nested callees; identical `(creator, target, payload)` sends within that fire still deduplicate.

  Zero-origin means the producing transaction's origin digest is all-zero: it descends from no signed transaction, as with a timer fire. For an async event-fire message with a zero origin, `producing_digest = event_fire_envelope_digest`; for an ordinary zero-origin producer, it remains the immediate transaction digest. Both zero-origin paths append the following unchanged system-emission suffix to the message-ID preimage, in this exact order, before hashing:

  1. The ASCII domain tag `cow3964.deferred.v1` followed by one zero byte.
  2. The block height (u64 big-endian).
  3. The emission identity digest (32 bytes).
  4. The executing actor (20 bytes).
  5. That actor's nonce (u64 big-endian).
  6. The message ordinal (u64 big-endian).
  7. The emission id's length (u64 big-endian), then the emission id.

  The suffix inputs depend on the producer:

  – **Ordinary actor/timer execution:** The emission identity digest is the producing transaction digest; the actor and nonce are the transaction's top-level executing actor and its nonce, which may differ from the message creator; the ordinal is the message's

position within the transaction; the emission id is the timer firing id (empty — length 0 and no bytes — when the producing transaction is not itself a timer fire).
  - **Asynchronous event-subscriber execution:** The emission identity digest is `keccak256(event-fire metadata)`, hashing exactly the envelope's complete metadata bytes defined above; the actor and nonce are the subscriber and its nonce; the ordinal is local to that subscriber's output; the emission id is the same 33-byte CIP-29 `sub_id`, in place of the timer firing id. For this zero-origin path, the main preimage's `producing_digest` remains `event_fire_envelope_digest`.

- **Caught synchronous failure:** If a caller catches an ordinary `call()` failure, the failed frame's messages, timers and `create_deferred_tx` children (including children queued by `on_fork`) are rolled back and do not run. Receipt events remain recorded. Intent-authorized deferred work survives and runs as that frame's actor. Async event-fire requests also survive: `0x1D` delivers the envelopes with that actor as the emitter. Surviving work runs against state without the failed frame's writes, except the intent rate-limit counter, which stays charged. Failed synchronous event subscribers instead use the stronger CIP-29 isolation: their events, all deferred requests and intent-counter changes are discarded too. The caller's own earlier `send()` survives a caught callee failure, so authors SHOULD complete fallible synchronous calls before issuing `send()` (CIP-6 §9.2).
- **Mailbox:** Capacity **1,000 messages** (`MAX_MAILBOX_LEN`, a per-actor message **count**, not a byte budget); enqueue beyond the limit MUST revert (or evict the oldest per the CIP-1 dead-letter ring).
- **Per-tx fanout:** A transaction (including all nested sends) MUST NOT enqueue more than **1,024** messages.
- **Reentrancy:** Allowed within a single block only; there is no synchronous call/return across blocks. **Recursion/await depth cap = 32**.
- **Timers (chain-native):** The following timer primitives are provided:
  - `timer_id = set_timer(height, handler, data)` — Schedule a one-time timer for the specified block height. Returns a unique `timer_id`.
  - Repeating schedules are SDK-level: `@every(seconds)` re-arms a one-time timer before each tick runs (CIP-6 §12.5). There is no native interval primitive.
  - `cancel_timer(timer_id)` — Cancel a pending timer by its ID. Returns the deposit if successful.
  - Timer delivery is **best-effort**; execution depends on the timer scheduler (see §5.1a and the same-block-prohibition rule).

3.4 **Randomness.**

- Consensus MUST produce a threshold-BLS VRF seed per round: `Seed { round: Round(epoch, view), signature }`, where `signature` is the epoch threshold signature over the absolute round encoding (`commonware`'s canonical `Round` codec; see CIP-11 §9.2), verifiable against the epoch's threshold public key. Exactly one valid seed exists per absolute round (notarized, finalized, or nullified). Seeds are identified by absolute `Round(epoch, view)`; implementations MUST NOT address a seed by block height and MUST NOT derive the beacon from a parent QC.
- Actors MAY call an API which returns `HKDF(R, label)`, where `R` is the execution-readable parent beacon of the executing block, which is the verified seed of the absolute round immediately preceding the executing block's proposal round. A round's own seed MUST NOT affect execution in that same round.

3.5 **Partial cost table (informative).**

- Exact metering is consensus-critical; implementers MUST match the reference cost table.

Primitive	Cycles
Python arithmetic ops	1
Python function call	10
Dictionary get/set	3
List append/access	2
String operations (per char)	1
host: mailbox send (per msg excl. payload)	80
host: timer set/cancel	200
host: blob commit (per KiB)	40

Cryptographic operations:

Operation	Cycles
BLS12-381 signature verification	8,000
Ed25519 signature verification	5,000
secp256k1 (ECDSA) signature verification	10,000
SHA-256 (base + per 64-byte block)	$500 + 8 \times \lceil \text{len}/64 \rceil$

Operation	Cycles
Keccak-256 (per 32 bytes)	$6 \times \lceil \text{len}/32 \rceil$
HKDF randomness derivation	1,000

These costs are consensus-critical; implementers MUST match the reference cost table. (BLAKE3 is used internally for state-root hashing but is **not** an actor-callable host function, so it carries no host-call cost.)

Cells meter bytes (payload, return data, inline blobs $\leq$ 64 KiB, /tmp).

0.18 4. Fees, Metering, and Basefee Adjustment

4.1 **Meters.**

- **Cycles:** Deterministic step count over Python operations + host calls.
- **Accesses:** Deterministic weighted consensus store I/O. One cold `StateGet` is one access; other operations consume the weight primitives in §17.3.
  - Cold/warm accounting MUST use one transaction-scoped access set with EIP-2929 semantics. The first logical in-access and first logical out-access for a key are charged; repeats in the same direction are warm and free. Read-then-write of one key charges once in each direction. The set and committed-value cache MUST be shared by nested frames and MUST NOT roll back when a nested frame reverts. Only cache mutations caused by staged writes unwind, in reverse order, to their pre-write values; a committed read learned inside a reverted frame remains cached.
  - An `AccessKey` is `(committed store namespace/prefix, key bytes as stored)`. Operations are warm-equivalent within one direction if and only if they resolve to the same physical committed key. `Code(hash)` and `Library(hash)` use distinct prefixes and are distinct keys; same-address account reads through sender, transfer-target, and balance-flush paths are one key; system-record wrappers alias the underlying record key.
  - Native system-instruction handlers that reserve state I/O under CIP-39 §15.1 MUST charge the full reservation on every branch, including rejection, and MUST NOT refund it. Each reserved read MUST charge one `R` primitive and each reserved write MUST charge one `W` primitive; these structural units accumulate in the same meter as every other Access primitive and are converted to charged accesses once, at settlement, never per reservation. Reserved operations MUST NOT participate in the transaction

cold/warm set. Actor-minted deferred `JobSubmit`, `IntentDeposit`, `IntentWithdraw`, and `IntentBroadcast` instructions form the separate actor-reachable reservation class specified in §17.3. Every signed transaction reserves the structural signed envelope `SIGNED_TX_ENVELOPE_STRUCTURAL_ACCESSES = 9` (`7R + 2W`: sender ingress, lane-fee config, admission vesting, sender-tail vesting backstop, primary signer actor, primary signer `AccountAuth`, sender delta-merge re-read, sender-tail write and primary wallet-marker write), plus `3R + 1W = 4` for each additional signing account (§17.2). Deferred work carries no signed envelope. Native STF reads that no signer or envelope owns, such as pure admission, nonce and basefee bookkeeping, stay out of `accesses`. Any other native system-instruction handler that gains store I/O MUST add a normative CIP-39 §15.1 reservation/vector row.

  - Reads MUST charge before the latency-bearing store operation on the proposal path. Writes MUST charge at stage time, before their later durable flush at `apply_finalized`.
  - The meter MUST cover cold `StateGet`, scans, actor and system setup reads, event/token/timer/SKM reads, state writes and deletes, message/token/timer writes, and the launch implementation's physical `fork_actor` storage copy. It MUST NOT charge pure in-memory calls such as `gas_left` or `context`, cryptographic compute already charged in cycles, persistent byte volume charged in cells, or transient RAM governed by the memory bound.
  - Storage reads and writes MUST NOT carry cycle terms (no 100-cycle read or 200-cycle write charge); read bandwidth MUST NOT be charged in `cells`; persisted write bytes MUST remain in `cells`. `cycles` measures compute, `accesses` measures store I/O, and `cells` measures persistent bytes. `memory_cells` MUST remain a hard per-execution liveness bound and MUST NOT enter fee settlement.
  - CIP-43 §4 remains the enforced-gate implementation specification.
  - CIP-43 §5 remains the per-host-method weight table.

- **Cells:** Persisted or payload bytes used by calldata, return data, and inline blobs ($\leq$ 64 KiB). Persistent write bytes remain cell-metered as specified above.

4.2 Tri-metered EIP-1559 basefees. Let `U_c`, `T_c` be cycles usage/target; `U_b`, `T_b` be cells usage/target; and `U_a`, `T_a` be accesses usage/target. Cycles and Accesses use the same four independently capped execution lanes (§6.3 and §17.9); every transaction is classified once for both dimensions. The global curves adjust using:

```
basefee_{x,i+1} = max(MIN_BASEFEE, basefee_{x,i} * (1 + clamp((U_x - T_x)/(T_x*alpha),
-1/alpha, +1/alpha)))
```

where `x` $\in$ `{cycle, cell, access}`, **alpha = 96**, the per-block change is capped at $\pm$**1/96**

(~1.04%), and **MIN_BASEFEE = 10** base units (1 CBY = 10^9 base units). The floor each curve clamps to is the governed `BasefeeConfig` value: `min_basefee` for cycles and cells, which launch genesis seeds to `MIN_BASEFEE`, and the separate `min_access_basefee` for access, which launch genesis seeds to `LAUNCH_MIN_ACCESS_BASEFEE = 35,000`. Genesis starts every curve at its floor. For Access, `U_a = sum(successful accesses_used) + min(sum(failed accesses_used), T_a)` over ordinary block-list transactions, using the same failure classification and aggregate one-target failed-usage cap as `U_c`. Lane assignment changes capacity isolation but MUST NOT change this aggregate `U_a`. The nominal 8,192-access lane sum is constrained by the shared 4,096-access block aggregate (CIP-43 §8); the basefee update uses actual ordinary-transaction demand, excluding payerless protocol work. The learning rate is calibrated for 1-second blocks so 12-second compounding `(1 + 1/96)^12` ≈ `13%` stays close to EIP-1559's 12.5%/12s ramp. Nodes MUST **burn 100%** of every basefee, including the access basefee; tips go to proposers/validators.

Natural-timer pricing uses a separate cycle curve. `TimerBasefee` follows CIP-1 §5 with learning denominator **1**, a **±12.5% (1/8)** per-block clamp, and a target of `LANE_TIMER_CYCLES / 2` = **4,400,000 cycles**; the updated timer basefee is clamped to `MIN_BASEFEE ..= MAX_BASEFEE`. Timer cells and accesses use the live global cell/access rates; the Timer-lane multiplier scales all three selected rates, with integer flooring, for both pre-charge and actual-cost settlement (CIP-5 §6.3). Natural-timer usage is excluded from the global cycle, cell, and access update inputs. Actual timer cycles drive only the separate timer curve.

4.3 **Targets (genesis defaults).**

- `T_c` (cycles target): **20,000,000 cycles**; the per-block hard cap is **80,000,000** (the sum of the four lane budgets, §6.3).
- `T_b` (cells target): **4,000,000 bytes**.
- Access capacity and genesis values are provisional ⟨**benchmark-gated**⟩. `BLOCK_ACCESS_TARGET = T_a = 2,048 accesses` is the basefee target. Access uses the same four-lane classifier as cycles: `LANE_SYSTEM_ACCESS = 4,096` (exactly `2 × T_a`), `LANE_USER_ACCESS = 2,051`, `LANE_RUNNER_ACCESS = 1,158`, and `LANE_TIMER_ACCESS = 887`. The User and Runner access budgets are sized from lane fit rather than the cycle-lane percentages: the User lane holds two maximal 1,024-charged stamps (`2 × 1,024 = 2,048` ≤ `2,051`, a 3-unit tail; 1,024 is `USER_TX_ACCESS_LIMIT`), and the Runner lane holds two single-runner, non-TEE, no-delegation `JobResultSubmit` reference suggestions (`2 × 577 = 1,154 < 1,158`) — the 577 is that fixture's suggestion, not a per-transaction cap, since `RUNNER_TX_ACCESS_LIMIT = LANE_RUNNER_ACCESS = 1,158` lets one Runner transaction claim the whole lane. The CIP-49 E=9 schedule assigns Runner six further accesses taken from Timer. Their nominal budget sum is still **8,192 charged accesses (`4 × T_a`)**, and unused capacity is

not reassigned between lanes. The shared `BLOCK_AGGREGATE_ACCESS_CEILING = 4,096` includes natural-Timer work. `PROTOCOL_TX_ACCESS_LIMIT = 4,096` is the outer protocol envelope; signed class ceilings are User 1,024, Runner 1,158, Timer 887, and ordinary System 1,507. Block validation rejects class-ceiling excess before execution, then enforces the CIP-43 tail guards and caps execution at `min(execution_access_cap, live_cap)`. Here `live_cap` includes the lane and applicable sub-lane remainders and the aggregate remainder after held finalization work (CIP-43 §8). Charged-tail failures retain consumed accesses; escaping storage errors and signed-tail invariant guards instead use the zero-gas rollback path in CIP-3 §2.5. The RPC `MAX_TX_ACCESS_LIMIT` aliases the 1,024 User ceiling; `COWBOY_MAX_TX_ACCESS_LIMIT` may only lower it, while `DEFAULT_ACCESS_LIMIT = 512` is the default stamp. Other signed classes retain their own consensus ceilings. These charged values are never scalar-rewritten. Natural Timer records store the structural default 160; extended records may opt into the 322 general-child or 375 actor-Intent tier under FIFO scheduling. Phase reservation caps the request at the structural fire-time ceiling of 375 before scaling; fee escrow continues to use `scale(access_limit_per_fire, p)` on the stored declaration. There is no auction-specific clamp. Complete non-fitting charged reservations carry intact; actual phase use is reported in `TimerReport`; the dedicated timer basefee remains cycle-priced, while timer accesses pay the live access price without contributing to `U_a`. Genesis starts the access basefee at its launch floor, `35,000` base units per access; `access_weight_scalar_ppm` is `1,350,000 ppm` at launch. Identity `1,000,000` is the minimum and compatibility default. The codec range is `1 ..= 1,000,000,000`, while governance enforces `[1,000,000, 1,350,358]` ppm (CIP-43 §8). The four access lane budgets carry drift-checked `param:` markers in §13; the remaining Access values (target, basefee, and scalar bounds) intentionally carry no inline `param:` markers.

4.4 State rent. Persistent storage of non-reserved actors incurs **rent** on the bytes actually stored, `max(kv_bytes, 1 KiB) × rent_rate` per rent-epoch; the rate is governance-tunable. An actor in debt stays functional, and it becomes evictable once its unpaid debt exceeds **30 rent-epochs** of rent.

4.5 Deferred transactions. A deferred transaction (a message, a `create_deferred_tx` child, an actor-minted System child, a runner callback, or an async event-subscriber child) is paid by the actor whose code queued it — its `from` — through the actor's balance and then its owner's under `FallbackToOwnerBalance`, at the current lane-effective basefees with priority zero. `DeployActor` grants the new actor `FallbackToOwnerBalance` on its signer by default, with no cap, unless the deploy opts out or sets a lifetime cap; the owner can revoke or re-grant it at any time. An actor created by `fork_actor` receives no such grant. One exception to actor-first: a deferred `JobSubmit` child escrows its job price from the actor alone, so when the actor's entitled owner

can cover the child's whole maximum fee (grant constraints and spendable balance), the owner pays that fee and the actor's balance stays for the price; otherwise the actor-first order applies. Before it executes, its maximum cost (its cycles and cells limits and the charged conversion of its structural access limit (CIP-43 §5.0), priced exactly as settlement prices them, at those basefees) is reserved from that waterfall; if the reservation cannot be covered, the transaction does not execute, commits no state and no descendants, and its receipt records `DeferredPayerUnfunded` (structured code 1015). After execution the actual cost is charged from the reservation and the remainder refunded, owner's share first. Timer fires keep their scheduler pre-charge (CIP-5 §6.3) and event-fire batches keep their subscription budgets (CIP-29). The originating transaction's remaining gas is authority for its descendants, never a monetary escrow.

Only work retained at the producing frame's rollback boundary can be materialized: §3.3 specifies which effects are discarded and which intent-authorized or event-fire requests survive a caught failure. In particular, a failed frame's `create_deferred_tx` children do not execute, including those queued by `on_fork`.

0.19 5. Off-Chain Compute

5.1 **Model registry.**

`model_id = keccak256(weights||arch||tokenizer||license)` MUST uniquely identify a model revision. Publishing is permissionless with a **refundable 1,000 CBY deposit**. Governance MAY flag/ban models.

5.2 **Runner staking.**

Runners MUST stake `max(10,000 CBY, 1.5 × declared_max_job_value)` in the Runner Registry.

Jobs MAY attach private or public CBFS volumes through CapToken-authorized mounts (CapToken mounts are authorized and checked on-chain) and MAY specify **Container** execution; the **Storage Manager** (**0x0A**) anchors volume manifests, the **Container Image Registry** holds the OCI image allowlist (§9), and the **Governance** actor (**0x09**) holds settable protocol parameters. Jobs MAY also release account secrets through the **Secrets Manager** (**0x04**) via threshold encryption (§9). Field-level job, mount, secret-reference, image-policy, and registry requirements are defined by CIP-2, CIP-9, CIP-10, CIP-23, CIP-24, and their canonical codecs.

5.3 **Job lifecycle.**

1. Actor posts a job with escrowed price.
2. Verification modes that require a quorum use `M = 9` **Runners and** `N = 6` **matching results** at launch. Modes explicitly defined as single-Runner use `M = N = 1`. A job

MUST fail admission when fewer than the required number of eligible Runners exist; it MUST NOT silently lower the threshold. Selection is stake-weighted Fisher-Yates VRF with weight `w = stake · sqrt(reputation)`. Every randomized assignment binds the request in the submission block and selects from the verified threshold seed of `r_seed = advance_round(r_submit, SELECTION_SEED_DELAY_VIEWS)`, where `SELECTION_SEED_DELAY_VIEWS = 3`, under domain `cowboy-runner-select-v4:`. This future-round rule applies to `M = 1` and `M > 1`; every job therefore remains pending until that absolute-round seed is available, imposing a minimum three-view selection delay. No assignment may use a seed fully known before the request is bound. The aggregator is uniformly drawn from committee members with reputation $\geq$ p50 and receives 1.5% of gross job payment from the runner share on successful settlement.

3. Runner returns `commit = keccak256(output||salt)`.
4. Runner reveals `{output, salt, proof?}`.
5. A challenge window of `DISPUTE_WINDOW_BLOCKS = 75` **blocks (≈ 75 s at 1 s/block)** is opened, requiring a **100 CBY bond**.
6. Proven dishonesty triggers stake slashing via the challenge mechanism. Operational failures (timeouts, crashes) result in reputation penalties only.
7. On finalization, runner(s) receive **89%** of job payment, the protocol burns **10%**, and **1%** goes to Treasury.

5.4 Determinism & bounds.

Jobs MUST pin `toolchain_digest` and `seed`. On-chain return data MUST be $\leq$ **64 KiB**.

5.5 TEE option.

A job MAY set `tee_required=true`. A valid attestation MUST match accepted policies.

0.20 6. Consensus, Randomness, and Networking

6.1 Consensus.

Simplex BFT PoS; ~**1s** target block time; **finality on commit** (~2s). A threshold-VRF schedule elects a leader for each consensus view; the same validator can be elected in consecutive views, and a timed-out view may produce no block. Votes aggregate via **BLS12-381** with buffered batch verification.

Simplex operates in absolute rounds with threshold-signed notarize, nullify, and finalize votes. A `Notarization { proposal, certificate }` proves a consensus quorum notarized the proposal; a `Finalization { proposal, certificate }` is the finality proof. Recovered threshold certificates do not retain a signer bitmap. A nullification advances the protocol without producing a block.

Validators derive the expected leader for each absolute consensus round from the epoch's verified threshold-VRF schedule. A proposal is valid only when its authenticated sender is that round's elected leader and its consensus certificate chain is valid. Leader election is round-addressed, not height-addressed.

Implementations MAY expose notarized and finalized states separately. The pinned consensus implementation defines the canonical message domains, timeout rules, and certificate encodings; Cowboy's conformance suite MUST pin those bytes and state transitions.

6.2 **P2P transport.**

Implementations MUST support QUIC over TLS 1.3.

NAT traversal. In-protocol NAT traversal (ICE/STUN/TURN-style hole punching) is **out of scope at genesis**: the protocol specifies the wire transport (QUIC/TLS 1.3) and peer discovery (DHT + bootstrap nodes), but does **not** mandate a hole-punching mechanism. Operators are responsible for mutual reachability — production validators are expected to run with publicly reachable QUIC endpoints (static IP / port-forward), and development networks MAY use an overlay (e.g. Tailscale/WireGuard) for connectivity. This keeps the consensus/transport layer simple and unopinionated; a protocol-level traversal mechanism, if warranted by operator demand, is a candidate for a future networking CIP rather than a launch requirement.

6.3 **Dedicated Lanes.**

Block space is partitioned into **dedicated lanes** with reserved capacity and per-lane fee multipliers:

Lane	Reserved Capacity	Fee Multiplier	Contents
System	41% (32.8M)	1.0×	Validator updates, governance, slashing
Timer	11% (8.8M)	1.0×	Scheduled timer executions
Runner	20% (16M)	1.0×	Runner job results, attestations, storage manifest anchors
User	28% (22.4M)	1.0×	User transactions

Lane guarantees:

- Timer and runner lanes prevent user transaction spam from blocking autonomous actor execution
- Lane cycle budgets are enforced independently; unused capacity is not reassigned across lanes

- For ordinary transactions, each effective rate is `floor(global_basefee × lane_fee_mult_ppm / 1_000_000)`. Natural-timer cycles instead use the separate timer-lane EIP-1559 basefee; the Timer multiplier scales that cycle rate and the global cell/access rates. Natural-timer usage is excluded from the global curves (§4.2). All four multipliers default to `1.0×` at genesis (no subsidy, no surcharge) and are Tier-0 governance-tunable.

6.4 **Gossip (mempool).**

The mempool is public and prioritized by effective fee. Transaction eligibility, seed disclosure, and execution are separated so no same-round randomness drives the proposal that reveals it:

1. Eligibility commitment. A block proposed at absolute round `r_commit` includes the canonical bodies of one candidate batch for each transaction-bearing lane (System, Runner, and User) and commits `H("cowboy-tx-eligibility-v1" || chain_id || r_commit || parent_hash || lane || lane_basefee_snapshot || merkle_root(canonical_tx_bodies))`. Before voting, validators MUST obtain every body, verify its hash, signature, nonce front, fee escrow, lane classification, and declared cycle limit, and verify that the sum of declared limits fits the lane budget. Across all candidate and pending batches, at most one transaction per sender may be reserved; a later nonce cannot enter the pipeline before the sender's reserved nonce executes. Finalization of the block fixes the batch and proves data availability to the finalizing quorum. Natural timer fires remain outside this mempool pipeline.
2. Ordering seed. Let `r_final` be the absolute round whose finalization certificate finalized `B_commit`. The next absolute round is `r_seed = advance_round(r_final, 1)`. Once `r_seed` has notarized or nullified, its verified threshold beacon becomes `R_order`. The set executes no earlier than `r_exec = advance_round(r_seed, 1)`. Its order is ascending by `HKDF(R_order, "cowboy-tx-order-v1" || lane || tx_hash)`, with `tx_hash` as the final tie-breaker.
3. Reservation and fees. The state transition of `B_commit` places the bodies in a consensus pending queue, escrows each transaction's maximum fee, and reserves its sender's current nonce; those mutations become canonical only when `B_commit` finalizes. Pending transactions cannot be recommitted. The committed lane basefee governs admission and settlement; later basefee changes do not invalidate the batch. Application-level reverts still consume metered resources and receive no replacement from outside the committed batch.
4. Block verification. Mature batches execute FIFO by `(r_final, B_commit.hash, lane)`. The first valid proposal at or after `r_exec` MUST carry the finalized commitment and `R_order` proof and MUST execute the available canonical bodies in the derived order. Validators reject substitution, omission, addition, the wrong seed round, or a different order. Execution consumes the reserved nonce and settles the escrow even when application logic reverts.

5. Null rounds. A null `r_seed` still has a canonical verified seed. If `r_exec` or a later view nullifies before the set executes, the same finalized set and order carry forward to the first valid proposal; no known-seed reselection or substitute set is permitted.

Within each lane, the candidate batch is selected by highest effective fee subject to declared-cycle capacity. For selection, `effective_fee` is computed using **intrinsic** costs only:

`effective_fee = intrinsic_cycles × min(max_fee_per_cycle, basefee_cycle + tip_per_cycle) + intrinsic_cells × min(max_fee_per_cell, basefee_cell + tip_per_cell) + intrinsic_accesses × min(max_fee_per_access, access_basefee + tip_per_access)`.
Transactions carry lane tags. Cowboy does not require private builders or an encrypted mempool at launch. Ordering does not eliminate inclusion, private-order-flow, or JIT risks.

6.5 MEV Reduction (Limitations Apply).

Cowboy's MEV mitigation strategy combines multiple mechanisms:

The protocol elects a leader for each consensus view. This limits deterministic leader capture, but it does not guarantee alternation: one validator can win consecutive views, and every elected leader can observe the public mempool and extract within its proposal.

Within each lane's committed eligible set, transactions are ordered by:

```
order_key = HKDF(R_order, "cowboy-tx-order-v1" || lane || tx_hash)
```

The finalized eligibility commitment MUST predate disclosure of `R_order`. This makes placement within the committed set deterministic and unpredictable at finalization time. It does not prevent manipulation of which transactions reach the candidate-selection phase, paid submission of many valid transactions, or private order flow.

The MEV-mitigation design above explicitly does *not* prevent: (a) **proposal inclusion / censorship** before eligibility certification; (b) **private-orderflow MEV** from off-chain bilateral routing; or (c) **JIT MEV against predictable actor logic**. Applications that need stronger guarantees against these classes SHOULD use commit-reveal patterns, slippage caps, batch auctions, or order-flow auctions. Lane isolation and VRF ordering are bounded protocol mitigations, not a complete application-level MEV solution.

The ~2-second target covers proposal, notarization, application certification, and finalization phases within the proposal's view. It minimizes reorg and time-bandit windows but does not by itself prevent front-running or sandwiching inside one block.

Reserved capacity for timers and runners ensures autonomous actors execute reliably regardless of user mempool congestion. Attackers cannot spam the user lane to delay victim transactions in other lanes.

Commit-reveal and encrypted-mempool schemes add latency, coordination, and key-management complexity. Cowboy accepts a public mempool at launch while treating stronger confidentiality and inclusion mechanisms as an evidence-driven design question. VRF ordering, leader election, and fast finality are bounded mitigations, not a claim of sufficient or complete MEV resistance.

0.21 7. Data Availability & Blobs

Non-goal — no DA-sampling layer. "Data availability" here means **blob storage durability**, not a Celestia/EigenDA-style erasure-coded data-availability-sampling layer. Cowboy has no such layer. Larger-than-inline data lives in the Cowboy File System (CBFS): durability comes from **Reed-Solomon replication across storage relays plus on-chain Proof-of-Retrievability challenges and `StorageCommitment` anchoring**, not from probabilistic availability sampling. See the Cowboy Storage Whitepaper for the canonical CBFS model.

7.1 Inline cap.

Inline blob cap is **64 KiB** per output.

7.2 External blobs.

Larger data MUST be **content-addressed** (e.g., IPFS). The on-chain commitment MUST be a multihash.

7.3 Blob lifecycle.

At genesis, ownership and rent determine blob lifetime. Blobs have **no protocol-level TTL or automatic expiry**:

- **Inline blobs** ($\leq$ 64 KiB, stored in the state trie) live as part of the owning actor's storage and are subject to the same **state rent** as any other state (§4.4 / the State Rent section); they disappear only when the owner overwrites/deletes the value or the owning actor is evicted for unpaid rent.
- **CBFS-stored blobs** (the off-chain shard layer) persist until the owner explicitly deletes them or the owning account is evicted. There is no background expiry sweep over live blobs.
- **Orphaned shards** — previous shard versions left behind after a new manifest commits (CIP-9's `write_id` keeps old shards retrievable for rollback) — are the one garbage-collected class: relay nodes prune them after `ORPHAN_SHARD_TTL` (86,400 blocks, ~24 h at 1s blocks; CIP-9 §14).

CBFS Reed-Solomon replication and on-chain Proof-of-Retrievability challenges provide durability and retrievability; see CIP-9 and the Cowboy Storage Whitepaper for the canonical model.

Introducing a protocol-level blob TTL would be a future governance change, out of scope for genesis.

0.22 8. Economics, Inflation, and Fees

This section specifies issuance, sinks, and distribution. It does not predict token value, promise net deflation, or characterize usage as a holder benefit.

8.1 Ticker & supply. **CBY**. Genesis supply **1,000,000,000 CBY**.

8.2 **Inflation.** A decreasing inflation schedule is used to bootstrap network security. Gross inflation is offset by protocol burn mechanisms (§8.4); net inflation depends on network usage.

Phase	Years	Gross Inflation	Rationale
Bootstrap	1–2	8% / 6%	Attract validators and runners while fee revenue is immature
Glidepath	3–4	4% / 3%	Reduce dilution as fee revenue and network stability improve
Steady-state	5+	2% floor	Maintain security budget with low long-run dilution

Governance MAY adjust the inflation rate within the following guardrails: (a) hard cap of 10% gross annual inflation, (b) maximum annual change of 2 percentage points, (c) timelock + supermajority for any change. A security-floor trigger MAY temporarily increase inflation if the staked ratio falls below a governance-defined threshold; the increase auto-expires when the threshold is recovered.

8.3 **Distribution at genesis.**

Genesis supply is split into a **Company Reserve** and a **Network Distribution** pool.

Company Reserve (66.67% — 666,700,000 CBY). Tokens held by or on behalf of the company, satisfying all outstanding token warrant obligations. Sub-buckets:

Bucket	% of Total	CBY	Vesting
Team & Core Contributors	20.00%	200,000,000	1-year cliff, 4-year linear monthly
Investors (Warrants / Token Rights)	20.00%	200,000,000	10% at TGE, 6-month cliff, 24-month linear
Advisors	3.33%	33,300,000	6-month cliff, 24-month linear, milestone gates
Treasury / Governance	13.34%	133,400,000	5% seed liquidity at TGE, remainder governance-voted
Foundation / Ops Reserve	10.00%	100,000,000	5% at TGE, quarterly board-approved releases over 36 months

Network Distribution (33.33% — 333,300,000 CBY). Tokens distributed to network participants via protocol emission. These sit outside the Company Reserve.

Bucket	% of Total	CBY	Emission Model
Validator Rewards	18.00%	180,000,000	Decreasing emission over 60 months
Runner Compute Incentives	8.00%	80,000,000	Usage-based; paid per runner-hour
Developer Grants	3.00%	30,000,000	Milestone-based, grant committee
Liquidity Mining / DEX	2.33%	23,300,000	Front-loaded over 12 months
Community / Airdrops	2.00%	20,000,000	2 drops (TGE + 6 months)

8.4 **Fee sinks & splits.**

Source	Split	Destination
Cycle, Access & Cell basefees	100%	Burned
Cycle, Access & Cell tips	100%	Proposer / validators
Runner job payments	89%	Runner(s)[1]
Runner job payments	10%	Burned
Runner job payments	1%	Treasury
Slashed stake	100%	Burned[2]

[1] The aggregator receives `aggregator_bonus_bps = 150` (= 1.5% of gross) of this 89% on successful settlement. The bonus is **carved out of the runner share**, not from the burn or treasury shares — so this row remains the literal 89% to runner-side recipients.

[2] Distribution governance lives in `SlashDistribution { burn_bps, submitter_bps, treasury_bps }`. Genesis defaults `(10000, 0, 0)` preserve the 100%-burn commitment listed in this row; any non-zero `submitter_bps` or `treasury_bps` requires a Tier-3 governance proposal that updates this row in lockstep.

The runner fee burn is an additional supply sink beyond basefee burns. Gross issuance and all burns are independently observable; net supply change depends on actual network usage. Governance MAY adjust the runner fee burn percentage via standard proposal + timelock.

0.23 9. System Actors & Precompiles

This table is the **single authoritative registry** of system-actor addresses. Every protocol specification (the CIP series) that introduces, renumbers, or references a system actor MUST agree with this table; per-specification copies are informative mirrors of this section.

Address	Name	Function
0x01	Runner Registry	Runner registration, staking, capabilities, health, reputation
0x02	Job Dispatcher	Job submission, VRF selection, lifecycle management
0x03	Result Verifier	Commit-reveal aggregation, verification, callback dispatch

Address	Name	Function
0x04	Secrets Manager (CBSS)	Threshold-IBE secret storage and release through a t-of-n proxy committee; proxy registration, account key material, secret policy, and release receipts
0x05	TEE Verifier	Remote attestation, trusted measurements, keys, and attestation records
0x06	Tri Basefee · Token Registry	Tri-metered basefee state and the fungible-token registry under disjoint storage keys; `system:dual_basefee` is a frozen legacy storage-key identifier, not a two-meter rule
0x07	Entitlement Registry	Runner Pool access control and capability grants
0x08	Treasury	Protocol fee collection and distribution, including the 1% Runner settlement fee
0x09	Governance	On-chain parameter governance, including `SettlementConfig` and other governable parameters
0x0A	Storage Manager	Volume ownership, CapToken issuance and revocation, manifest anchoring, and storage-rent accounting
0x0B	Relay Registry	Storage Relay registration, staking, heartbeats, Proof-of-Retrievability challenges, and slashing
0x0C	Session Actor	Multi-party-payment escrow and cumulative-voucher settlement

Address	Name	Function
`0x0D`	Retired Stream Key Manager; address permanently reserved	CIP-7 replacement release; old runtime removal requires coordinated deployment
0x0E	Route Registry	Domain-binding records and FQDN-to-actor resolution
0x0F	Gateway Registry	Gateway registration, staking, health, and system-mediated ingress dispatch
0x10	Receipt Registry	Domain-action receipts
0x11	Validator Set (CIP-11)	Validator identity, stake, epoch membership, and `ValidatorRecord` snapshots
0x12	PaymentGate	CIP-18 HTTP-native payment settlement; MPP and x402 normalize to one settlement layer
0x13	Container Registry (CIP-10)	OCI image policy, resource classes, compute-fee escrow and settlement, and Council pause controls
0x14	INTENT_SETTLEMENT	CIP-34 cross-chain intent settlement: token-diff conservation, custody, sealed auctions, and pluggable cross-chain backends
0x15	EventListener	Cross-chain event subscriptions and delivery
0x16	BankActor (CIP-28)	Agent banking and card-payment rails
0x17	Stream Registry (CIP-39)	CBQS stream and provider lifecycle, and prepaid-balance custody and settlement keyed by `(owner, provider)`
0x18	Platform Fee Account (CIP-31)	Accrual and governed withdrawal of the CBFS and CBQS platform share

Address	Name	Function
0x19	Room Authority	Room identities, membership generations, transport bindings and policy commitments
0x1A	FEATURE_REGISTRY	Stage 0: holds only the genesis-written `feature:auth` authority key
0x1D	Event Subscription	Subscription-bidding market interface
0x1E	TradingPost	Actor declarations, versions, grants, hiring, and per-call settlement

Allocation Notes:

1. This table is the protocol source of truth for system-actor allocation. Every implementation and dependent specification MUST reproduce it exactly.
2. A proposed specification that conflicts with an existing claim MUST renumber before acceptance. An address described only in prose as the next free slot, rather than claimed in this table, is non-binding and MUST be resolved against this table when allocated.
3. A specification that introduces a new system actor MUST claim the **lowest unallocated address** in the dense sequence and MUST update this table in the same change. Under the table above, that address is `0x1B`.
4. The full low-address space below **0x100** (`RESERVED_SYSTEM_ADDRESS_LIMIT`) is reserved for system actors: actors below it are exempt from rent and from the aggregate storage-capacity limits (key count and stored bytes) in §3.2. User actor addresses use the address-group rule defined here shard prefixes and an 18-byte hash suffix; derivation does not deliberately allocate addresses in this band. The PVM separately rejects the dense system band **0x01**–**0x0F** for `fee_payer` overrides; `0x1D` sits deliberately outside that enforced band.
5. The aggregate storage-capacity limits and their reserved-band exemption are defined in §3.2. The Secrets Manager (`0x04`) holds network-wide custody state; growth is bounded by CBSS fees, stakes, bonds and deletion/garbage collection, with durable tombstones and version high-water marks retained. Payerless CBSS phase work abandons storage-capacity failures and rolls back affected writes/events instead of failing the block; other

> storage errors remain fatal. The Secrets Whitepaper §2 specifies the stored records and their lifecycles.
>
> Field-level instruction schemas, registry semantics, and on-chain encodings for each system actor are elaborated in the CIP series.

Feature Registry, Stage 0. Genesis requires the closed `feature_authority` section specified by CIP-40 §16: 1–16 distinct, strictly ascending addresses and an integer threshold in `1..=members.len()`, with the environment's genesis deployer as the sole member and threshold 1 at Stage 0. The section is part of the genesis input bound into `chain_instance_id`. Genesis writes exactly `feature:auth`, the canonical encoding of `FeatureAuthority { members, threshold }`, under `FEATURE_REGISTRY = 0x1A`. No instruction can write this actor. Directives, feature records and `GET /features` are **reserved** for a later CIP.

0.24 10. Developer Experience (DX)

- A primary Python SDK (`cowboy-py`) is provided.
- The tool suite includes a single-node devnet (`cowboyd`), Runner simulator, faucet, and explorer.
- Reentrancy guards, capability-scoped handles, and idempotent message handling are encouraged via the SDK.

0.25 11. Governance & Upgrades

- Cowboy uses token-weighted on-chain governance from genesis (bicameral: staked CBY + validator one-vote). A permanent **Security Council 7-of-9** holds narrow emergency authority (cancel queued proposals during timelock, fast-track Tier-3 upgrades, circuit-break a system actor with mandatory retroactive ratification). CIP-12 specifies tier values, quorums, deposits, voting windows, and Council scope.
- `block_access_target` and `access_weight_scalar_ppm` are Tier-0 parameters; per-row Access weights are coordinated-release constants and are not live-governable.
- Standard governance actions have a timelock of **7 days**; emergency fast-track actions have a timelock of **6 hours**.
- Governance coordinates **hot-code upgrades**.

0.26 12. Security Considerations

Cowboy's launch security is a system argument. The custom VM, consensus, governance, threshold services, Runner selection, storage recovery, and bridge dependencies MUST be

evaluated together, not as isolated feature checks.

12.1 **DoS limits (consensus-enforced; governance-tunable).**

- `max_tx_size` = **1 MiB** (`MAX_TRANSACTION_BYTES = 1_048_576`; rejected at decode)
- `max_message_depth_per_tx` = 32
- `per_actor_per_block_cycles` = 1,000,000 (burstable)

12.2 **Runner safety.**

Slashing for proven dishonesty (fabricated results, wrong model). Operational failures result in reputation penalties only. Committees mitigate single-runner faults. Manifest anchoring and container image allowlists reduce supply-chain and cross-runner integrity risk; default container network isolation preserves verifiability for N-of-M modes.

12.3 **Reentrancy.**

Allowed but depth-capped; stdlib provides reentrancy guards.

12.4 **Randomness bias.**

Threshold-BLS VRF with epoch keys; actors derive sub-randomness via HKDF.

12.5 **State rent & eviction.**

Prevents state bloat; eviction windows protect liveness.

12.6 **Launch proof obligations.**

Before public-network launch, the following evidence is required:

- Publish validator-concentration assumptions, weak-subjectivity and long-range recovery procedures, measured one-second-block behavior across realistic geographic latency, and correlated-slashing analysis.
- Maintain deterministic conformance vectors, differential PVM execution, sandbox-escape review, upgrade compatibility tests, and an explicit client-diversity plan.
- Clients MUST reproduce `docs/cips/cip-39-gas-vectors-v2.json` with schema `cowboy.cbqs.gas-vectors.v2` and normative SHA-256 `8f0bf5e0ae4b94ab2c6683b8fd796dbdd9518f5e4` That artifact retains CIP-39's 27 CBQS-dispatch paths across all eight native instructions, covering every required success and rejection branch; it carries no state-read/write cycle terms, records one `R` access per charged read and one `W` access per charged write at the identity scalar, and includes `key_validation_cycles = 600` with each row's `key_validations` count. Each row's Cycles include `600 × key_validations`; key validation does not change the branch set, Cells or Accesses. Cross-engine conformance MUST also use the node-generated PVM host-call access corpus to compare the test-only in-process reference oracle with production Zygote. Every per-host-method table row

with equivalent host-call traces under the same metering policy and access weights MUST produce equal `accesses_used`, including canonical alias keys, nested-tree/revert behavior, speed invariance, and write-flush cadence. Receipt and state-root differences MUST be recorded and resolved in favor of Zygote's canonical production bytes. Known differences and demonstrated comparison surfaces are documented in CIP-43 §12; those checks do not establish universal receipt/state equality. The corpus MUST include a non-identity `access_weight_scalar_ppm = 500_000` case with two distinct cold `R` primitives that charges 2 access units, not the invalid grouped-ceiling result of 1.

- Document DKG membership changes, epoch-key rotation, lost-share recovery, collusion thresholds, and failure behavior when quorum is unavailable.
- Prove that request authors, Runners, and proposers cannot grind assignments after seeing selection entropy; test both `M = 1` and quorum paths at the production caller seam.
- Test data export, provider replacement, revocation, and recovery for every near-chain subsystem.
- Publish Council scope, key-compromise response, upgrade rollback, bridge rate limits, and the trust assumptions of every supported corridor.

0.27 13. Parameters (Genesis Defaults)

Execution:

`memory_per_call` = 10 MiB; `max_actor_kv_bytes` = 8 MiB and `max_actor_kv_count` = 10,000 keys (non-reserved actors; §3.2); `reentrancy_depth` = 32; `fanout_per_tx` = 1024; `mailbox_capacity` = **1,000 messages** (`MAX_MAILBOX_LEN` — a message count, not `mailbox_capacity_bytes`); `dedup_window` = a **500,000-entry LRU** (`MAX_SEEN_MESSAGE_IDS`; not a 10,000-block window).

Fees:

`T_c` = 20,000,000 cycles (per-block hard cap = 80,000,000 across lanes); `T_b` = 4,000,000 bytes; `basefee_alpha` = 96; `basefee_max_change` = ±1/96 (~1.04%/block); `MIN_BASEFEE` = 10 base units, the cycle and cell floor and their genesis basefee. CIP-43 introduced provisional, ⟨**benchmark-gated**⟩ Access defaults: `T_a = 2,048 accesses`; System/User/Runner/Timer lane budgets `4,096 / 2,051 / 1,158 / 887` charged accesses (see §13 lane budgets below); nominal transaction-lane budget sum `8,192`, subject to the shared `BLOCK_AGGREGATE_ACCESS_CEILING = 4,096` charged accesses, including natural-Timer work; genesis access basefee and governed access floor `min_access_basefee = 35,000` base units per access; RPC User ceiling `MAX_TX_ACCESS_LIMIT = 1,024 accesses` (a node-local override may lower it), default stamp `DEFAULT_ACCESS_LIMIT = 512`; and `access_weight_scalar_ppm`

`= 1,350,000 ppm` at launch. The scalar codec range is `1 ..= 1,000,000,000`, but protocol configuration is bounded by `MIN_ACCESS_WEIGHT_SCALAR_PPM = 1,000,000` and `MAX_ACCESS_WEIGHT_SCALAR_PPM = 1,350,358` (CIP-43 §8).

The four access lane budgets carry drift-checked `param:` markers below; the remaining Access values intentionally have no inline `param:` markers.

Consensus:

`minimum_validator_stake` = governance-tunable; `epoch` = 3600 blocks (~1 h); `block_time` = 1 s; `finality` = ~2 s; `unbonding_period` = 7 days; `jail_period` = 24 h; `double_sign_slash` = 1%; `consensus_protocol` = Simplex BFT.

Dedicated Lanes:

`system_lane_capacity` = **41%** (`LANE_SYSTEM_CYCLES = 32,800,000`); `user_lane_capacity` = **28%** (`LANE_USER_CYCLES = 22,400,000`); `runner_lane_capacity` = **20%** (`LANE_RUNNER_CYCLES = 16,000,000`); `timer_lane_capacity` = **11%** (`LANE_TIMER_CYCLES = 8,800,000`) — of the 80,000,000-cycle block cap.

The corresponding access-lane budgets are System 4,096, User 2,051, Runner 1,158, and Timer 887 charged accesses. They remain provisional ⟨**benchmark-gated**⟩, use the cycle classifier, and do not spill between lanes. The User and Runner budgets are derived from lane fit — the User lane holds two maximal 1,024-charged transactions (2,051, a 3-unit tail), the Runner lane holds two single-runner, non-TEE, no-delegation reference `JobResultSubmit` suggestions (577 live each, totaling 1,154 < 1,158; not a per-transaction cap, since `RUNNER_TX_ACCESS_LIMIT = LANE_RUNNER_ACCESS = 1,158`) — rather than the cycle-lane percentages; System is `2 × T_a`, Timer funds the MRU and CIP-49 allotments, and the three payer lanes sum to 4,096. Their nominal sum is 8,192, but execution is capped at `min(execution_access_cap, live_cap)`: `live_cap` includes the remaining lane capacity, the ordinary-System sub-lane where applicable, and the remaining aggregate capacity after held finalization work. The shared block aggregate is capped at 4,096 charged accesses; natural-Timer work must fit within that same aggregate (CIP-43 §8).

Lane fit alone does not establish aggregate full funding. Current callback suggestions are 427/577/577 at identity/launch/maximum scalar; their parent owns 77 structural accesses and their child reserve 311. Maximum-occupancy BlockStart reserves 1,823 structural (2,462 charged at launch/maximum). Before Timer holds, the aggregate payer remainder is 2,273/1,634/1,634; a full 887-access Timer hold leaves 1,386/747/747. Two callback suggestions plus two 1,024 User stamps need 2,902/3,202/3,202. The actual parent-state BlockStart bound can be smaller; these maximum-occupancy subtractions are not selected-count guarantees. CIP-43 §8 gives the exact E=9/scalar derivation and the per-transaction tail floor.

`lane_fee_multiplier_system` = 1.0; `lane_fee_multiplier_timer` = 1.0; `lane_fee_multiplier_runner` = 1.0; `lane_fee_multiplier_user` = 1.0. (All Tier-0 governance-tunable.)

Off-chain:

`committee M`, `threshold N` = `M = 9, N = 6` for quorum modes; explicitly single-Runner modes use `M = N = 1`; `SELECTION_SEED_DELAY_VIEWS` = **3** for every mode. Every randomized assignment uses future-round entropy after request binding and remains pending until the named absolute-round seed is available. `challenge_window` = `dispute_window_blocks` = **75 blocks (≈ 75 s at 1 s/block)**; `challenge_bond` = 100 CBY; `runner_stake_floor` = 10,000 CBY (CBY-denominated; no oracle dependency at launch); `dispute_window_blocks` = 75. `reputation_half_life_blocks` = 1,209,600 (~14 days at 1s blocks); `aggregator_eligibility_percentile` = 50 (p50 reputation); `aggregator_bonus_bps` = 150 (1.5% of gross, paid from runner share); `non_reveal_slash_bps` = 2500 (25% of effective stake on proven non-reveal); `slash_distribution.{burn_bps, submitter_bps, treasury_bps}` = `(10000, 0, 0)` (genesis 100%-burn per §8.4; Tier-3 to amend).

State Rent:

`catch_up_fee` = 10% (Tier-0 tunable as `rent_catchup_bps = 1000`); `rent_rate` = **2,739,726,027,397 atto/byte/rent-epoch** (approximately **0.001 CBY/byte/year**, Tier-0); `rent_epoch_length` = 86,400 blocks (~1 day at 1s blocks per WP §6.1); `eviction_threshold_epochs` = 30; `rent_min_billable_bytes` = 1,024 bytes (1 KiB).

Economics:

`supply` = 1,000,000,000; `company_reserve` = 66.67%; inflation follows the schedule in §8.2; `basefee burn` = 100%; `runner_fee_burn` = 10%; `job_fee_to_treasury` = 1%; `runner_payout` = 89%.

0.27.1 13.1 Parameter Governance Registry

The entries below map each governance-tunable parameter group to the **0x09 Governance** storage key that holds it, its owning specification, its genesis default, and its governance tier. Per-parameter live values and their amendment history are addressable through the generic registry prefix `system:gov:param:<key>` with an audit trail at `system:gov:param:history:<key>`.

- **Settlement split (runner/burn/treasury)** — CIP-2; key `system:settlement_config`; genesis `89% / 10% / 1%`; Tier-3.
- **Basefee config (`T_c`, `T_b`, `alpha`, `delta`)** — CIP-3; key `system:basefee_config`; genesis `20,000,000 / 4,000,000 / 96 / 1/96`; Tier-0.

- **Access-meter config (`block_access_target`, `access_weight_scalar_ppm`)**: introduced by CIP-43; key `system:basefee_config`; provisional launch genesis `2,048 / 1,350,000`, ⟨**benchmark-gated**⟩; Tier-0. The scalar codec accepts `1 ..= 1,000,000,000`, while protocol governance MUST enforce `[1,000,000, 1,350,358]` ppm. The maximum is `floor(1,507 × 1,000,000 / 1,116)`, using the maximum signed staged-finalize sizing bound (CIP-43 §8). These provisional values intentionally carry no inline `param:` markers.
- **Lane fee multipliers (system/timer/runner/user)** — CIP-3; key `system:lane_fee_multipliers`; genesis `all 1.0×`; Tier-0.
- **Deploy gas per byte (`gas.deploy.cycles_per_byte`, `gas.deploy.cells_per_byte`)** — CIP-3; keys `system:gov:param:gas.deploy.cycles_per_byte` and `system:gov:param:gas.deploy.ce` genesis `12 / 40` (unset = compiled defaults); nonzero u64; priced for deploys, upgrades and library publishes; Tier-0.
- **Timer config** — CIP-5; key `system:timer_config`; genesis values in §5.1; Tier-0.
- **State-rent config (`rent_rate`, `rent_catchup_bps`, ...)** — CIP-4; key `system:cip4:rent_config`; genesis `2,739,726,027,397 atto/byte/epoch; 1,000 bps`; Tier-0.
- **Reputation config (half-life, percentile)** — CIP-2; key `system:cip2:reputation_config`; genesis `1,209,600 blk; p50`; Tier-0.
- **Aggregator config (`aggregator_bonus_bps`)** — CIP-2; key `system:cip2:aggregator_config`; genesis `150 bps`; Tier-0.
- **Slash distribution (`burn/submitter/treasury_bps`)** — CIP-2 / §8.4; key `system:cip2:slash_distribution`; genesis `(10000, 0, 0)`; Tier-3.
- **Non-reveal slash (`non_reveal_slash_bps`)** — CIP-2; key `system:cip2:non_reveal_config`; genesis `2,500 bps`; Tier-0.
- **Committee sizing (`M`, `N`)** — CIP-2; key `system:cip2:committee_config`; genesis `M=9, N=6` for quorum modes and `M=N=1` for explicitly single-Runner modes; Tier-0.
- **Semantic-similarity model id** — CIP-2; key `system:cip2:semantic_similarity_embedding_model`; genesis governance-set; Tier-0.
- **Generic registered parameters** — CIP-12; key `system:gov:param:<key>` with history at `…:history:<key>`; genesis per parameter; Tier-0/3.

`UpdateBasefeeConfig` and `SubmitProposal` MUST each carry `block_access_target` and `access_weight_scalar_ppm`. Both values MUST be recorded under `system:basefee_config` as Tier-0 governance parameters. The scalar codec range remains `1 ..= 1,000,000,000`, but protocol validation MUST additionally reject governed values outside `[MIN_ACCESS_WEIGHT_SCALAR_PPM = 1,000,000, MAX_ACCESS_WEIGHT_SCALAR_PPM = 1,350,358]`. The configured scalar MUST be applied to each gate charge and structural allowance using the shared unsigned 128-bit ceiling calculation in §17.10. Governance MAY change the target and scalar within this bound, but MUST NOT change a single Access weight-table row without the coordinated release required

by CIP-3 §4.

These entries mirror the §13 genesis values and form the authoritative index from each protocol parameter to its concrete storage key. The release conformance check MUST compare all ten values carrying inline `param:` markers against the network implementation and fail closed when either source is unavailable. Repeated markers for the same parameter MUST agree before comparison. Defaults stored outside the constants module require equivalent conformance checks.

0.28 14. Differences vs. Ethereum

- Cowboy runs deterministic Python actors in the PVM. Ethereum L1 runs EVM bytecode; rollups may define EVM or custom execution environments.
- Cowboy meters Cycles, Accesses, and Cells independently within application execution. Ethereum has execution gas and an independent EIP-4844 blob-gas market for data availability.
- Cowboy validators execute protocol-native timer messages under a reserved budget. Ethereum applications generally use external automation, while other chains and rollups may define native scheduling.
- Cowboy enshrines Runner registration, assignment, settlement, and trust profiles. Ethereum applications choose among oracle networks, coprocessors, TEEs, proofs, and centralized services.
- Cowboy charges rent on stored bytes and supports eviction/restoration. Ethereum L1 state is currently persistent; statelessness and state-expiry designs remain active research.
- Cowboy bears an independent validator and governance security budget. An Ethereum rollup can inherit settlement and data-availability properties while retaining its own sequencer, proving, and upgrade assumptions.

0.29 15. Entitlements

A declarative, composable permissions system governs the capabilities of actors and runners. Entitlements control access to resources like networking, storage, and execution parameters, enforcing least-privilege by default. The system is enforced at deployment time, by the scheduler, and at the VM syscall gate.

15.1 Goals

- Least privilege by default.
- Deterministic enforcement.

- Declarative & composable.
- Auditable on-chain.

15.2 **Objects & lifecycle**

- Actor entitlements specify the permissions the actor requires.
- Runner entitlements specify the capabilities the runner provides.

15.3 **Rules**

1. MUST: Actors `require` entitlements; runners `provide` them.
2. MUST: Scheduler matches only if `requires` $\subseteq$ `provides`.
3. MUST: Syscalls fail if the corresponding entitlement is missing.
4. MUST: Child actors only inherit entitlements marked `inheritable:true`.

(For a full list of entitlements, see the Entitlements Specification.)

0.30 16. Ethereum Interoperability

Cowboy's interoperability with Ethereum is a primary design goal. Asset transfer and cross-chain communication rely on shared cryptographic primitives, governance-selected third-party bridge infrastructure, and event subscription mechanisms. Actor access to bridge operations is gated by the `bridge.asset` and `bridge.subscribe_event` entitlements.

0.30.1 16.1. Account Unification

- Cowboy external accounts (EOAs) MUST use the same `secp256k1` elliptic curve for signatures as Ethereum. A single private key can control accounts on both networks, simplifying account continuity while increasing the cross-chain blast radius of key compromise.
- At genesis the actor `verify_signature` host API supports strict Ed25519 only (CIP-6 §12.8). EIP-712 transaction slots are verified by consensus. Actor-visible verified co-signers and arbitrary EIP-712 host verification are reserved follow-up work (CIP-41); this release promises neither API.

0.30.2 16.2. Bridge Infrastructure

Cowboy relies on third-party bridge infrastructure for asset transfers and cross-chain message passing between Cowboy and Ethereum.

Requirements:

- The bridge MUST support the locking of native ETH and ERC-20 tokens on Ethereum to mint a corresponding wrapped representation on Cowboy (`wETH`, `wERC-20`), and the reverse burn-to-unlock flow.
- The bridge MUST support generic message passing: a transaction on one chain triggering a message call to a designated recipient on the other.
- Bridge selection and integration are determined by governance. The protocol does not implement its own bridge validator set.

0.30.3 16.3. Event Subscription (Ethereum to Cowboy)

- Cowboy actors MAY subscribe to event logs emitted by specific contracts on the Ethereum blockchain.
- The `EventListener` system actor at **0x15** SHALL manage these subscriptions. It relies on the bridge validator set to act as a decentralized oracle, monitoring Ethereum for specified events.
- When a subscribed event is confirmed (i.e., finalized on Ethereum), the `EventListener` actor MUST enqueue a message to the subscribing Cowboy actor, delivering the event's topic and data as the message payload.
- The cost of this subscription service SHALL be paid by the actor in CBY, covering the gas fees incurred by the oracle validators on Ethereum.

0.30.4 16.4. Policy and Security

- All interoperability functions available to an actor, such as `bridge_asset` or `subscribe_event`, MUST be governed by the Entitlements system (§15).
- An actor's deployment manifest MUST declare the specific Ethereum contracts it is permitted to interact with and the types of assets it is allowed to bridge, enforcing the principle of least privilege.

0.31 17. Fee Model Specification

This section is authoritative; any conflicting values elsewhere are non-normative.

0.31.1 17.1. Overview

Cowboy uses a **tri-metered fee system** (introduced by CIP-43):

Meter	Unit	Purpose
Cycles	Compute units	CPU time, opcode execution, actor API calls
Accesses	Weighted cold store operations	QMDB reads during propose and writes flushed at `apply_finalized`
Cells	Persistent data units (bytes)	Storage writes and calldata; read bandwidth is excluded

All three meters use independent EIP-1559-style basefee adjustment. Fees are paid in CBY.

On-chain fees price compute (`cycles`), store I/O (`accesses`), and persisted bytes (`cells`). Ongoing state rent remains a separate holding-cost ledger, and off-chain services remain direct CBY payments to Runners and Providers.

0.31.2 17.2. Transaction Intrinsic Costs

Every transaction pays a base cost before execution begins. Only the Transfer row below is stated inclusive of signature verification; all other rows are pre-signature handler bases.

Transaction Type	Base Cycles	Base Cells	Notes
Transfer	15,000	0	Single secp256k1 slot: 5,000 transaction base + 10,000 verification; excludes calldata and instruction work
Deploy	100,000	`code_size`	Actor deployment

Transaction Type	Base Cycles	Base Cells	Notes
ActorMessage	21,000	`calldata_size`	Method invocation
LlmRequest	10,000	`prompt_size`	Off-chain inference request
TimerSchedule	5,000	64	Schedule future execution

> This table is not exhaustive. Runner jobs, container execution, and volume operations are specified by the relevant CIPs and the protocol gas schedule; a transaction type's omission here does not make it unmetered.

Every signed transaction has an intrinsic access cost: the signed envelope `SIGNED_TX_ENVELOPE_STRUCTURAL_ACC` `= 9` structural accesses (`7R + 2W`), plus `4` (`3R + 1W`) for each additional signing account. The envelope is reserved in full before identity I/O and stands outside the cold/warm set. Deferred children carry no signed envelope. Consensus store I/O beyond the envelope is charged when an Access primitive is invoked. The structural total, envelope included, is converted to charged accesses once at settlement (§17.10): at the launch scalar an empty signed envelope settles to `ceil(9 × 1.35) = 13` charged accesses.

0.31.3 17.3. Execution Costs (Cycles)

0.31.3.1 Opcode Costs Python opcode costs are implementation-defined and not protocol-specified. The runtime MUST ensure deterministic cycle consumption across all validators.

0.31.3.2 Actor API Costs

Operation	Base Cost	Variable Cost
`send_message()`	80 cycles	—
`storage_read()`	0 cycles; consensus store reads charge `accesses`	`R` or `Scan(n)` under the Access table

Operation	Base Cost	Variable Cost
`storage_write()`	0 cycles; consensus store writes charge `accesses`	`W` or `Copy(n)` under the Access table
`hash()`	per algorithm (keccak256 ~6/32B, sha256 500+8/block)	—
`verify_signature(2, ...)`	5,000 Ed25519 crypto cycles, plus the existing host-boundary charge	Other algorithms unsupported at genesis (CIP-6 §12.8)
`get_block_info()`	100 cycles	—
`emit_event()`	0 cycles (no fixed cost, no per-byte cycles)	see storage note[1]

> The authoritative per-opcode cost schedule and its update procedure will be elaborated in a future CIP.

0.31.3.3 Access Weights CIP-43 §§5.0–5.2 owns the canonical primitive weights, table provenance, and scalar conversion. Its pinned historical table versions do not identify the coordinated CIP-27 replacement row; that release must update the complete row set and fingerprints together.

Primitive	Structural weight
`R`	`1` per cold logical point read
`W`	`1` per cold logical out-key write
Host crossing	`1` per actual host call
`Scan(n)`	`5 + ceil(n/8)`
`Copy(n)`	`ceil(n/8)`
`Fanout(k)`	`ceil(k/4)`
`NestedFrame`	`75`, plus one-time `40` first-nested setup per frame chain
Top execution frame	`99`

The gate accumulates raw structural units, with transaction-scoped cold/warm accounting. Apply

the scalar once at each CIP-43 meter boundary, never per primitive or per internal precharge:

`charged_accesses = ceil(structural_total × access_weight_scalar_ppm / 1_000_000)`

On entry, a charged limit admits structural capacity `floor(charged_limit × 1_000_000 / access_weight_scalar_ppm)`. The products, division and rounding use checked unsigned 128-bit arithmetic. `access_weight_scalar_ppm = 1_000_000` is the identity; the genesis scalar is `1_350_000` under CIP-43. `Rk(k)` denotes individually gated reads with warm deduplication, not separately rounded fees. `W(out-keys)` denotes the cold logical out-keys not already warmed by an aggregate. These rules ensure that splitting an implementation's internal charges does not change its total fee.

`Scan(n)` MUST charge from its validated bound and insert its at-most-`n` returned keys into the in-access warm set. A later point read of a returned key is warm; a key outside that returned set MUST charge `R`. `Copy(n)` MUST insert the `n` copied child keys into the out-access warm set, so they MUST NOT charge a separate `W`. `Fanout(k)` MUST insert its `k` derived per-item record keys into the in-access warm set, making their per-item record reads warm; every per-item nested dispatch MUST still charge `NestedFrame`. Each `Scan(n)`, `Copy(n)`, and `Fanout(k)` aggregate MUST charge from its validated or derived bound irrespective of prior individual-key warmth and MUST NOT reduce that bound to `n - 1` or `k - 1`.

`scan_actor_storage` MUST use one `max_items` value for the aggregate charge, raw storage dispatch, and returned-row guard. `copy_actor_storage` MUST receive the syscall-time read-your-writes snapshot and its maintained `kv_count` used for `Copy(n)` as the logical image bound (CIP-27 §3.3.3; CIP-43 §5.1). Production storage MUST reject an over-bound count before opening the actor-storage range and MUST check the produced image against that same bound.

For the coordinated CIP-27 ABI, `Copy(n)` uses the parent's syscall-time read-your-writes `kv_count`, including staged inserts/deletes, from the same snapshot used by the copy, byte charge, cap and birth accounting. The count is maintained in O(1); no unmetered enumeration may discover it. This supersedes the pinned prototype's committed-counter rule as a same-release implementation and fingerprint requirement, not an already deployed behavior. `Copy(n)` is the single aggregate out-access charge for the copied child keys and warms every resulting destination key; those keys MUST NOT also charge `W`. CIP-3 §2.2.1 retains the full CIP-43 host-crossing, `R`, pin-scan and nested-frame charges.

0.31.3.4 Actor-minted deferred System reservations Four native System variants are actor-reachable because an actor can mint them as a deferred child: an outgoing message to the Job Dispatcher becomes `JobSubmit`; `intent_deposit` becomes `IntentDeposit`; `intent_withdraw` becomes `IntentWithdraw`; and `intent_broadcast` or `intent_broadcast_ex` becomes `IntentBroadcast`. In each case the emitting or current actor is the sender.

The execution layer MUST identify this class structurally. The transaction MUST have the canonical deferred shape, a non-zero deferred origin digest, and one of those four System variants. A zero origin digest identifies engine-minted Timer or event work, while user-signed native System transactions are not deferred even if their signed fields contain an origin. Neither is in this class: engine-minted work runs under its own structural allowance, and user-signed native System instructions reserve their own fixed rows (CIP-43 §3).

Before the deferred System arm performs its first sender-account read, it MUST reserve the fixed row below against the child's access allowance. The reservation uses only input-visible values and committed protocol constants, is charged in full on success or rejection, and is never refunded. If the complete reservation does not fit, execution MUST return `OutOfGas` before any raw-store dispatch and commit no partial handler state.

Actor-minted variant	Native System reservation	Structural units at 1,000,000 ppm
`JobSubmit`	`R(sender)` deferred ingress	1
`IntentDeposit`	`Rk(8) + Wk(4)`	12
`IntentWithdraw`	`Rk(7) + Wk(3)`	10
`IntentBroadcast`	`Rk(5) + Wk(6)`	11

These rows are structural and settle with the child's meter. The intent hooks' `can_transfer` and `on_transfer` frames are not in the fixed rows: each nested frame is charged live, only when it materializes, and store work inside either hook passes through the ordinary typed gate.

Actor-minted `JobSubmit` is enabled. Its child row carries only the one deferred-ingress read; the native handler's work is owned by the **System-selection sub-lane**, a consensus meter disjoint from the transaction and child access meters. Every `JobSubmit`, user-signed or actor-minted, reserves the selection row `43 + 5N + 6M` from that sub-lane, where `N = min(eligible, MAX_SELECTION_CANDIDATES = 69)` is the bounded frontier-plus-tail candidate pool and $M \leq$ `MAX_SELECTION_COMMITTEE = 9` is the committee. A container job adds its image-admission reads (12 for a digest reference, 7 for a base reference) and a pool-restricted job adds one Entitlement Registry read per candidate, both on the same sub-lane. The sub-lane reservation is drained even if the transaction later fails. Actor-minted submissions with volume attachments are rejected before any state read, because their token and commitment rows cannot fit the 160-access child execution allowance. `MAX_REGISTERED_RUNNERS = 10,000` caps the active-runner registration set; registration MUST reject before writing an entry that would cross the cap, but selection is bounded by the candidate pool, not by a registry scan. All access weights in this subsection are provisional ⟨**benchmark-gated**⟩.

CIP-43 §5 remains the per-host-method weight table for PVM host calls, intercepted system subroutes, execution-envelope work, and deferred work.

0.31.3.5 Platform Token Costs

Operation	Cycles	Cells
`token_transfer()`	1,000	64
`token_transfer_from()`	1,500	96
`token_approve()`	500	32
`token_balance_of()`	100	0
`token_mint()`	1,000	64
`token_burn()`	500	64
`token_create()`	10,000	256 + name + symbol

Validation hooks add up to 50,000 cycles per transfer (capped).

0.31.4 17.4. On-Chain Storage Costs (Cells)

Operation	Cell Cost
State write	1 cell/byte written
State read	No Cell charge; consensus store I/O is charged by the `accesses` meter
Calldata	1 cell/byte of transaction data
Event emission	[1] **1 cell/byte** of event topic and payload (`emit_event`; no fixed emission overhead or cycles). An explicit schema that first sets or changes the topic declaration separately charges the emitter 32 Cells for the persisted identifier (CIP-29 §2.1).

0.31.5 17.5. State Rent

Non-reserved actors pay ongoing rent on the bytes they store. This section provides the operational summary plus the governance monitoring cadence for CBY-denominated rent.

```
rent_due_epoch = max(kv_bytes, RENT_MIN_BILLABLE_BYTES)
                 × rent_rate_atto_per_byte_epoch

Parameters:
  RENT_MIN_BILLABLE_BYTES   = 1,024 bytes (1 KiB)
  rent_rate_atto_per_byte_epoch = 2,739,726,027,397
                                  (≈0.001 CBY/byte/year; Tier-0 tunable)
  rent_epoch_length         = 86,400 blocks (~1 day at 1s blocks)
  eviction_threshold_epochs = 30
  rent_catchup_bps          = 1,000 (= 10% catch-up premium on restore)
  MAX_ACTOR_KV_BYTES        = 8,388,608 bytes (8 MiB per actor)
  MAX_ACTOR_KV_COUNT        = 10,000 keys per actor
```

Debt and eviction:

- `kv_bytes` is the actor's committed stored-byte counter; an actor storing less than 1 KiB pays for 1 KiB
- Debt is held in atto; settlement pays whole wei from the balance and carries any sub-wei remainder
- An actor in debt remains functional; funding its balance pays the debt at the next settlement
- Once unpaid debt exceeds 30 rent-epochs of rent the actor is evictable; eviction prunes storage and timers and keeps a commitment to the pre-eviction storage
- Restoration requires the full pre-eviction storage matching that commitment, debt repayment, and the 10% catch-up premium

Governance monitors CBY-denominated rent. Cowboy keeps rent CBY-denominated to avoid introducing a consensus-layer CBY/USD oracle dependency. The Foundation publishes monthly the implied USD value of the rent at prevailing CBY/USD spot; target band `[$1, $10] / MiB / year`. If the implied USD value drifts outside the band for two consecutive monthly reviews, a Tier-0 governance proposal MUST be filed to adjust `rent_rate`. Any switch to USD-denominated rent requires a battle-tested CBY/USD oracle module.

0.31.6 17.6. Off-Chain Blob Storage

Large data (images, datasets, AI inference traces) uses Retention Contracts:

Cost Component	How Charged
BlobRef storage	~64 bytes on-chain → Cell cost + state rent
Provider payments	Direct CBY to Provider via escrow (market rate)

Off-chain blob payload storage is **not cell-metered as VM state**. CBFS read payments use payer-funded cumulative channels active from genesis in the regenesis ruleset under CIP-9 §10.4.1 and CIP-31 §2.1; operator services remain explicitly opt-in. The channels do not authorize unilateral provider reports or ambient account debits. Opening, bounded settlement pages and refunds are on-chain operations with ordinary Cycle/Cell/Access gas in addition to dedicated transfer principal; all native gas components debit CBY. Retention, provider registration, availability commitments, Proof-of-Retrievability, payment, slashing, and the separate CBFS storage-rent split are specified in the Cowboy Storage Whitepaper, CIP-9, and CIP-31. Failed retrieval proofs trigger the penalty and slashing rules defined there. Operators must explicitly enable the services and select their production funding/gas policies.

0.31.7 17.7. Off-Chain Compute (Runner Marketplace)

Off-chain job execution (LLM, HTTP, MCP, Custom, **Container**) is **not gas-metered** on the cycle/cell meters. Runners operate in a competitive marketplace:

Aspect	Specification
Pricing	Runners publish **rate cards** (e.g. CBY per token, per HTTP request, per compute second; storage/volume capability and container limits declared in registry metadata)
Selection	Actors specify `max_price` (and optional tip) in **JobSpec**; dispatcher binds the request before using future-round VRF entropy to select eligible Runners
Settlement	On verified delivery: 89% to runner, 10% burned, 1% to Treasury
Collateral	`runner_stake >= max(10,000 CBY, 1.5 × declared_max_job_value)`

Aspect	Specification
Verification	Mode-specific agreement, objective checks, attestation, or proofs; subjective quality is not established by settlement alone

The market sets unit prices for LLM and other job types. Runners declare volume fees and container policies in registry fields.

> Field-level `JobSpec`, `RateCard`, selection, and verification schemas are defined by CIP-2, CIP-11, CIP-23, the canonical protocol codec, and the protocol state machine.

0.31.8 17.8. Fee Adjustment (EIP-1559 Style)

The global Cycles, Cells, and Accesses curves use independent basefee adjustment:

```
basefee_{x,i+1} = max(MIN_BASEFEE, basefee_{x,i} × (1 + clamp((U_x - T_x) / (T_x × α), -δ, +
```

For Access, `U_a = sum(successful accesses_used) + min(sum(failed accesses_used), T_a)` over ordinary block-list transactions. Failed-transaction access usage uses the same failure classification and aggregate one-target cap as failed-cycle usage in `U_c`. Lane assignment does not change this aggregate. Natural-timer usage is excluded from these global inputs. The separate timer cycle curve instead uses learning denominator 1, a ±12.5% per-block clamp, and a target of half the Timer lane (§4.2); the updated timer basefee is clamped to `MIN_BASEFEE ..= MAX_BASEFEE`. Timer cells and accesses still pay the global rates scaled by the Timer multiplier (CIP-5 §6.3).

For the global curves: - α `= 96` (BASEFEE_ALPHA — calibrated for 1-second blocks, vs Ethereum's 8 for 12-second blocks) - max change = **±1/96 (~1.04%) per block** (BASEFEE_MAX_CHANGE_DENOM = 96) - `MIN_BASEFEE = 10` base units (cycle and cell floor; the update runs in fixed point at `BASEFEE_FRAC_SCALE = 10^9`, so its per-block step never truncates to zero at this floor). The access curve clamps to the governed `min_access_basefee`, `35,000` at launch genesis. - `clamp(v, -1/96, +1/96)` caps the per-block adjustment

This formula is normative and consistent with §4.2.

Cycle parameters: | Parameter | Value | |————|——-| | Target | 20,000,000 cycles/block | | Block hard cap | 80,000,000 cycles (sum of lanes) | | Max change/block | ±1/96 (~1.04%) | | α (BASEFEE_ALPHA) | 96 |

Cell parameters: | Parameter | Value | |————|——-| | Target | 4,000,000 bytes/block | | Max change/block | ±1/96 (~1.04%) | | α (BASEFEE_ALPHA) | 96 |

Access parameters (provisional — ⟨benchmark-gated⟩): | Parameter | Provisional value | |————|—————:| | Target (`T_a`) | 2,048 accesses/block | | System lane | 4,096 accesses (50%; `2 × T_a`) | | User lane | 2,051 accesses (two maximal 1,024 stamps + 3 tail) | | Runner lane | 1,158 accesses (two single-runner, non-TEE, no-delegation reference `JobResultSubmit` suggestions at 577 live; not a per-tx cap — `RUNNER_TX_ACCESS_LIMIT = LANE_RUNNER_ACCESS = 1,158`; see §13) | | Timer lane | 887 accesses (after the MRU and CIP-49 Runner allotments) | | nominal transaction-lane capacity | 8,192 accesses (`4 × T_a`, exact lane-budget sum) | | Shared block aggregate ceiling | 4,096 charged accesses, including natural-Timer work (`BLOCK_AGGREGATE_ACCESS_CEILING`) | | Outer protocol envelope | 4,096 accesses (`PROTOCOL_TX_ACCESS_LIMIT = LANE_SYSTEM_ACCESS`); signed class ceilings are smaller (CIP-43 §8) | | Initial basefee | 100,000 attoCBY/access | | Launch weight scalar | 1,350,000 ppm (identity/minimum and compatibility default: 1,000,000) | | Weight scalar codec range | `1 ..= 1,000,000,000` | | Governed weight scalar range | `[1,000,000, 1,350,358]` ppm (CIP-43 §8) | | RPC User ceiling / default stamp | 1,024 / 512 accesses; `COWBOY_MAX_TX_ACCESS_LIMIT` may only lower the User ceiling | | Max change/block | ±1/96 (~1.04%) | | α (BASEFEE_ALPHA) | 96 |

The four access lane budgets carry drift-checked `param:` markers in §13; the remaining Access values intentionally carry no inline `param:` markers. The target and scalar are Tier-0 governed as specified in §13.1; per-row table weights remain coordinated-release constants.

`BLOCK_ACCESS_TARGET = T_a` is the access-basefee target. Access capacity is divided across the same transaction lanes as cycles: System 4,096, User 2,051, Runner 1,158, and Timer 887 charged accesses (User and Runner sized by lane fit; see §13). These independently enforced budgets have a nominal sum of **8,192 accesses (`4 × T_a`)**; unused capacity is not reassigned across lanes. The shared block aggregate is capped at `BLOCK_AGGREGATE_ACCESS_CEILING = 4,096`, including natural-Timer work; natural-Timer usage remains excluded from `U_a`. `PROTOCOL_TX_ACCESS_LIMIT = 4,096` is the outer protocol envelope, while signed class ceilings are User 1,024, Runner 1,158, Timer 887, and ordinary System 1,507. Every transaction uses the same classifier for cycle and access accounting. In canonical block order, class-ceiling excess takes the E1900 no-execution path. After the CIP-43 per-transaction signed-tail and complete-deferred-reservation guards pass, a zero live cap also takes E1900; otherwise execution is capped at `min(execution_access_cap, live_cap)`. The live cap includes lane/sub-lane remainders and the aggregate remainder after held finalization work (CIP-43 §8). Charged-tail failures retain consumed accesses; escaping storage errors and signed-tail invariant guards use the zero-gas rollback path in CIP-3 §2.5. Natural timers persist a structural `access_limit_per_fire`: the default is 160, the general-child tier is 322, and the actor-Intent ceiling is 375 under FIFO scheduling. Pass 1

reserves `scale(min(request, phase_cap), p)` charged accesses, a non-fitting fire carries intact rather than running against a remainder, and actual use is reported separately in `TimerReport`. The dedicated timer basefee remains cycle-priced; timer accesses pay the live access price without contributing to block `U_a`. RPC admission uses the signed class ceilings. `MAX_TX_ACCESS_LIMIT` aliases the 1,024 User ceiling and `COWBOY_MAX_TX_ACCESS_LIMIT` may only lower it; the default stamp remains `DEFAULT_ACCESS_LIMIT = 512`. The local override is admission-only. Signed limits and class ceilings are already charged and are not scalar-rewritten.

The access fee contribution is `tip_a = min(max_priority_fee_per_access, max_fee_per_access - access_basefee)` and `fee_a = accesses_used × (access_basefee + tip_a)`. The unused access limit MUST be refunded at `max_fee_per_access`. One hundred percent of the access-basefee portion MUST be burned; the priority tip goes to the proposer.

The protocol burns 100% of basefee revenue. Net supply change depends on gross issuance and actual usage; the protocol does not guarantee net deflation.

0.31.9 17.9. Reserved Capacity (Execution Lanes)

Block space is partitioned to reserve capacity for critical transaction types. Cycles and Accesses use the same single lane classification for each transaction. A reservation isolates cross-lane congestion; it does not guarantee inclusion for every eligible transaction within a saturated lane:

Lane	Cycle Budget	Access Budget	Percentage	Fee Multiplier	Purpose
System	32,800,000	4,096	41%	1.0×	Governance, upgrades
User	22,400,000	2,051	28%	1.0×	Regular transactions

Lane	Cycle Budget	Access Budget	Percentage	Fee Multiplier	Purpose
Runner	16,000,000	1,158	20%	1.0×	Runner job results, attestations, callbacks (incl. storage manifest anchors)
Timer	8,800,000	887	11%	1.0×	Scheduled actor execution

The access budgets remain provisional ⟨**benchmark-gated**⟩. The Percentage column above is the cycle-lane split (System/User/Runner/Timer = 41/28/20/11% of the 80,000,000-cycle cap); System access is `2 × T_a = 4,096` and Timer is 887 after the MRU and CIP-49 allotments, and the access User (2,051) and Runner (1,158) budgets do not track those percentages — they are sized by lane fit (the User lane by two maximal 1,024 stamps, the Runner lane by two single-runner, non-TEE, no-delegation reference `JobResultSubmit` suggestions at 577 live — not a per-transaction Runner cap, since `RUNNER_TX_ACCESS_LIMIT = LANE_RUNNER_ACCESS = 1,158`; see the §13 derivation and its lane-fit-vs-aggregate qualification). Their nominal budget sum is 8,192 accesses (`4 × T_a`). Actual shared-pool use is capped at `BLOCK_AGGREGATE_ACCESS_CEILING = 4,096`, including natural-Timer work. Each transaction is additionally capped by its remaining lane/sub-lane and aggregate authority (CIP-43 §8).

Lane behavior:

- Lane cycle and access budgets are enforced independently; unused capacity is not reassigned across lanes.
- The cycle classifier is authoritative for Accesses. Implementations MUST NOT maintain a separate access-lane classifier. Deferred children, event fires, and mailbox deliveries take the access lane assigned by that classifier at the same execution site.
- After class-ceiling checks and the CIP-43 tail guards, execution is capped at `min(execution_access_cap, live_cap)`, where `live_cap` includes the lane/sub-lane remainder and the aggregate remainder after held finalization work. CIP-43 §8 defines each signed transaction's input-derived tail floor; a deferred transaction must

fit its complete effective reservation. These tail violations invalidate the block; only after those checks does a zero live cap take the E1900 no-execution path (CIP-43 §8). Charged-tail failures retain consumed accesses without credit-back; escaping storage errors and signed-tail invariant guards use the zero-gas rollback path in CIP-3 §2.5. Unused capacity MUST NOT move between lanes for any outcome.

- Natural timer fires execute in a fresh dedicated end-of-block phase under FIFO ordering, `LANE_TIMER_CYCLES`, and `LANE_TIMER_ACCESS`. ⟨benchmark-gated⟩ Each record declares a structural `access_limit_per_fire`; the default is 160 and extended timers may opt into the 322 general-child or 375 actor-Intent tier under FIFO scheduling. Pass 1 reserves `scale(min(request, phase_cap), p)` charged accesses. A fire whose complete reservation cannot fit the Timer lane or shared aggregate carries intact under the cycle policy; pass 2 never re-clamps it. At identity, the 887-access phase admits five classic, two general-child or two actor-Intent reservations; launch/maximum admit four/two/one. Actual charged timer-phase access is reported beside timer cycles, not in the ordinary Timer lane or block `U_a`. The dedicated timer basefee remains cycle-priced; timer accesses pay the live access price without contributing to a later block's `U_a`. There is no auction-specific access or cycle clamp. Natural timers pay no priority tips.
- Per-lane `Fee Multiplier` scales the applicable live rate with integer flooring: ordinary transactions use the global Cycle/Cell/Access rates; natural-timer fires use the separate timer cycle basefee and the global cell/access rates (CIP-5 §6.3). The timer cycle curve has its own 50%-of-lane target and ±12.5% per-block clamp; the updated timer basefee is clamped to `MIN_BASEFEE ..= MAX_BASEFEE`, and natural-timer usage is excluded from the global curves (§4.2). All four multipliers default to `1.0×` (no subsidy, no surcharge) and are Tier-0 governance-tunable.

0.31.10 17.10. Scalar-safe protocol allowances

Access table rows, natural-Timer declarations, deferred-child limits and protocol work envelopes are structural values, and every access meter accumulates structural units. The scalar crosses a meter boundary once, through two consensus helpers using unsigned 128-bit intermediates: `scale(s, p) = ceil(s × p / 1,000,000)` converts a structural total to charged accesses at settlement, and `floor(c × 1,000,000 / p)` converts a charged authority (a signed `access_limit`, a lane or aggregate remainder) to structural capacity on entry. Zero remains zero and non-zero structural work never becomes free. Implementations MUST NOT apply the scalar per host operation, per reservation or per protocol allowance: `scale(1, p) + scale(1, p)` disagrees with `scale(2, p)` and is a consensus divergence.

Natural-Timer default 160, general-child tier 322, actor-Intent ceiling 375, and each stored

declaration remain structural. Escrow uses the scaled declaration; phase reservation and the timer transaction use the scaled, phase-clamped request. A deferred child's access limit is the structural **162** (**160** execution plus the **2**-access pre-engine gate), stored unscaled in the child transaction and its meter; its actual structural use settles once to `scale(used, p)` charged accesses, so a child that exhausts its limit settles to **162 / 219 / 219** at identity / launch / maximum scalar (CIP-43 §5.3). Asynchronous event-fire envelopes run on that same structural child meter; their fair shares divide the remaining structural headroom, a charged lane or lineage-pool remainder bounding it is floor-converted to structural headroom, and their reserved subscription writes are structural charges in the same meter. A runner callback child has the structural `RUNNER_CALLBACK_ACCESSES_LIMIT = 311` (`149` handler allowance plus one `162` child reserve). Release builds MUST reject any synthesized child whose `origin_remaining_access` exceeds the origin's signed access limit; a debug-only assertion is insufficient.

User-signed `access_limit`, `DEFAULT_ACCESS_LIMIT`, `MAX_TX_ACCESS_LIMIT`, and lane budgets are already charged and are not scaled. Identity `p = 1,000,000` changes no allowance or consensus bytes. Launch uses `p = 1,350,000`; identity remains the compatibility default. Although the scalar codec accepts `1 ..= 1,000,000,000`, governance MUST enforce `[1,000,000, 1,350,358]` ppm. The maximum is `floor(1,507 × 1,000,000 / 1,116)`, from the maximum signed staged-finalize sizing bound within its 1,507 charged reserve (CIP-43 §8). The protocol also checks natural-Timer, deferred-child, callback and block-start fits; the ordinary-System bound binds. The callback fit uses `scale(DEFAULT_ACCESS_LIMIT, MAX_ACCESS_WEIGHT_SCALAR_PPM) <= PROTOCOL_TX_ACCESS_LIMIT`.

0.31.11 17.11. Fee Estimation

Wallets and applications SHOULD estimate fees as:

```
def estimate_fee(tx):
    intrinsic_cycles = INTRINSIC_COSTS[tx.type]
    intrinsic_cells = len(tx.calldata)
    intrinsic_accesses = 9 + 4 * additional_signers   # structural signed envelope

    # Estimate execution cost (via simulation or heuristics)
    execution_cycles = simulate_execution(tx)
    execution_cells = estimate_storage_writes(tx)
    execution_accesses = estimate_store_accesses(tx)

    total_cycles = intrinsic_cycles + execution_cycles
    total_cells = intrinsic_cells + execution_cells
    # structural -> charged once, at the live access_weight_scalar_ppm
```

```
    total_accesses = ceil((intrinsic_accesses + execution_accesses) * scalar_ppm /
↪   1_000_000)

    # Apply current basefees
    cycle_fee = total_cycles * cycle_basefee
    cell_fee = total_cells * cell_basefee
    access_fee = total_accesses * access_basefee

    # Add priority tip
    priority_fee = ((total_cycles * tip_per_cycle)
                    + (total_cells * tip_per_cell)
                    + (total_accesses * tip_per_access))

    return cycle_fee + cell_fee + access_fee + priority_fee
```

0.32 18. Prior Art and References

Cowboy's contribution is the composition and boundary between actors, consensus, Runners, and near-chain services. The underlying areas have substantial prior art:

1. **TON actor accounts and asynchronous messages:** TON Addresses Overview and The Open Network whitepaper.
2. **Internet Computer native canister timers:** ICP Timers.
3. **Sui object identity and ownership:** The Sui Smart Contracts Platform.
4. **Chainlink external computation and automation:** Chainlink Functions and Automation documentation.
5. **Filecoin verifiable storage:** Filecoin Proof-of-Storage specification.
6. **Ethereum rollup architectures:** Zero-knowledge rollups and Optimistic rollups.
7. **Ethereum independent blob-gas market:** EIP-4844.
8. **MEV and block construction:** Ethereum MEV overview, Proposer-builder separation, and Flashbots documentation.
9. **State expiry and resurrection:** Ethereum statelessness and state-expiry roadmap.

0.33 Appendix A. Transaction Encoding Conformance

CIP-40 §14 is the authoritative transaction-v3 corpus contract. Old version-1/version-2 vectors are removed; they are not valid inputs to the new decoder. Publish the codec-generated canonical

bytes, native hashes, independently recomputed EIP-712 hashes and complete invalid-input matrix before releasing the new genesis. CIP-40 §16 separately provides the exact v3 activation/chain-identity fixture. This specification update does not claim the full v3 implementation or transaction corpus has shipped.