

Cowboy: A Layer-1 Blockchain for Autonomous Software

Technical Whitepaper

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

September 2026 — v0.10.4

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	2026-09-12
License	CC0-1.0

Note: 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. For work that does not belong in replicated execution — LLM inference, web requests, MCP tool calls, or **containerized** jobs — a decentralized network of **Runners** executes jobs 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 subsystem-specific control-plane state each requires, including resource lifecycle, authority, provider assignment, commitments, receipts, pricing, and settlement. Together these services form the **sovereign substrate** for 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. A **sovereign application** pairs chain-anchored ownership and authority with predominantly provider-served execution and data. These are useful poles, not

a mechanical classification rule: execution locality alone does not establish sovereignty, providers retain the operational authority stated by each subsystem, and 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 primitive 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 gave us programmable money, but its execution model is fundamentally reactive — smart contracts sit inert until 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, quotas, 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 further innovates by adopting Python as its execution language, replacing the domain-specific languages required by existing chains (Solidity on Ethereum, 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.
- **Sovereign 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. Sovereignty 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.
- **GBA:** Gas Bidding Agent — an actor that dynamically bids for timer execution on behalf of another actor.

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 via `set_timer` and `set_interval` without external keeper infrastructure. The scheduler uses a per-fire `fee_payer` pre-charge and refund. CIP-1 additionally requires a timer-lane EIP-1559 basefee, priority tips, and a per-actor fairness weight (§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 (≈ 75 s at 1 s/block) and slashing rules provides economic accountability.
- **A Sovereign Substrate of 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 rails 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 timers with EIP-1559 pricing and Gas Bidding Agents
Off-chain compute	Application-selected oracle, coprocessor, or rollup services	Protocol-mediated Runner marketplace
Storage, secrets, coordination	Application-specific cloud services	Sovereign substrate of 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. A sovereign 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 sovereignty, and most applications combine both shapes. This taxonomy is architectural and non-normative.

0.3.4.1 Sovereignty Criteria A subsystem supports operational sovereignty 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 **sovereign 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 are computed CREATE2-style from the creator address, a salt, and the code hash. The **world state** is a mapping:

State : **Address** $\rightarrow$ { **balance**, **nonce**, **code_hash?**, **storage?**, **metadata** }

where actor storage is a key/value map subject to quotas 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. Delivery is **exactly-once after finality**; before finality it is **at-least-once** and may be reverted. 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 implemented FIFO scheduler is described in §5.1a. CIP-1 specifies the auction requirements in §5.1b; implementation work remains to satisfy that complete contract.

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 structural protocol default; the extended scheduler may request a smaller structural limit but may not exceed it. At the provisional COW-3516 values, the natural-Timer FIFO access cap and default are structural 56, the auction-phase structural cap is 26, and the Timer phase contains 901 charged accesses per block. At identity, the default therefore runs at 56 in FIFO and is clamped to 26 in the auction phase, just like

`gas_limit_per_fire`; these limits admit sixteen full-cap FIFO fires or thirty-four full-cap auction fires, while smaller declared requests may admit more.

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 any applicable cycle priority tip, and refunds unused escrow to `fee_payer`. At identity, a real-STF empty-handler timer measures and pins the natural-Timer execution envelope at 13 charged accesses, leaving 43 structural-equivalent accesses for additional handler work under the provisional structural 56 FIFO cap. Its charged envelope follows the live scalar.

The literal access target and lane defaults in `tools/fee-simulator/src/main.rs` are the COW-3516 recalibration touchpoint and MUST move with any ratified sizing change.

- **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 Requirements CIP-1 specifies the following auction contract:

- **Timer-lane EIP-1559 basefee.** A per-block basefee is maintained against `LANE_TIMER_CYCLES` utilisation; target 50%, max change $\pm 12.5\%$ per block; 100% of the lane basefee is burned.
- **Priority tip.** Timer parameters are `max_fee_per_cycle` (fee cap) and `max_priority_fee_per_cycle` (tip cap). The effective priority is `min(max_priority_fee_per_cycle, max_fee_per_cycle - lane_basefee)`. Tips go to the **block proposer**.

- **Per-actor fairness weight** $W(\text{actor}) \in [1, 2]$ computed over a 1,000-block rolling window: actors at or below the network-median fire rate get $W = 2$ (maximum boost); actors at $2\times$ median or above get $W = 1$. Inclusion ordering: $\text{priority_per_cycle} \times W(\text{actor})$.
- **Per-timer cycle cap:** `MAX_CYCLES_PER_FIRE_AUCTION_PHASE` = 250,000, preventing any single timer from monopolising the lane. The implemented FIFO cap is 550,000 cycles.
- **Default GBA:** `max_fee_per_cycle` = $2 \times \text{lane_basefee}$; `max_priority_fee_per_cycle` = `previous_block_p50_priority_tip_per_cycle`. SDKs expose a `priority_tier_hint` $\in \{\text{economy, standard, fast, urgent}\}$ mapping to multipliers $\{0.8\times, 1.0\times, 1.5\times, 2.5\times\}$ on the priority fee.
- **Lane fee multiplier:** $1.0\times$ at genesis (no subsidy), Tier-0 tunable.
- **All FIFO-phase invariants retained:** Tx-then-Timer block ordering, per-fire `fee_payer` pre-charge + refund (the priority tip extends `max_cost` but does not change the mechanics), three-path lifecycle, lane separation, system-instruction opcodes 48/49/50, same-block prohibition.

0.5.1.3 DoS Prevention (both phases)

Attack Vector	Mitigation
Schedule millions of timers	<code>max_timers_per_actor</code> = 1,024 + per-fire <code>fee_payer</code> pre-charge
Sybil attack across many actors	Per-block execution budget caps total timer work (<code>LANE_TIMER_CYCLES</code> , ~11% of block cap)
Timer bomb (many timers, one block)	Per-tx fanout cap (1,024 enqueued timers/messages) + <code>max_timers_per_actor</code> = 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
Fill queue far in advance	(FIFO phase) per-fire <code>max_cost</code> re-checked at each fire; (auction phase) lane basefee rises with utilisation
Outbid everyone perpetually	(auction phase only) per-actor fairness weight $W(\text{actor}) \in [1, 2]$ over 1,000-block window
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**. All state reads, writes, outbound messages, timers, deferred transactions, and events produced within a handler participate in the same nested rollback domain — they either all commit or all revert. 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 grieving 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 grieving — 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 surface, not 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 and standard-library surface 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.
- For the same transaction, in-process and zygote execution MUST produce byte-identical committed `accesses_used`, receipt, and state-root surfaces.

Runtime Environment:

- No JIT compilation — pure interpretation mode only.
- 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).

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

0.7.3 Storage and State Persistence

The storage architecture uses a three-layer model:

1. **The Ledger:** Append-only log of blocks — the sequential, historical source of truth.

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.

All actor storage 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. Cycles resemble Erlang reductions — a budget of discrete steps bounding handler execution.
- **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.

The protocol does **not** meter off-chain Runner execution. Runners set their own prices in a free market, and actors escrow CBY at submission. This separates deterministic on-chain metering from non-deterministic off-chain resource pricing.

0.9 Off-Chain Compute: The Runner Marketplace

Actors can outsource computation — 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.

Attachable storage. 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](#).

Secret release. 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. Message flow is broker RPC rather than one 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.

Container execution shapes. 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. **Post:** Actor submits a job with escrowed price, resource bounds, and trust profile.
2. **Assign:** 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. **Commit:** Runners return `commit = keccak256(output || salt)`.
4. **Reveal:** Runners reveal `{output, salt, proof?}`.
5. **Challenge:** A `DISPUTE_WINDOW_BLOCKS = 75`-block window opens (≈ 75 s at 1 s/block); challengers post a 100 CBY bond.
6. **Resolve:** Proven dishonesty triggers slashing; operational failures result in reputation penalties only.

7. **Payout:** 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.

Runners and actors are matched by **Entitlements** — a declarative permissions framework 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: $\text{actual_payment} = \min(\text{reported_usage} \times \text{rates}, \text{max_price})$. Tips incentivize priority during high demand.

Runner-reported usage is trusted by default, subject to reputation scoring and anomaly detection ($>2\times$ expected usage triggers automatic review). For stronger guarantees, actors can require TEE-attested metering. Proven dishonesty (e.g., fabricated results, misreported usage) is subject to stake slashing via the challenge mechanism; operational failures (timeouts, crashes) result in reputation penalties only.

Payment and Failure Handling:

Outcome	Runner Payment	Actor Refund
Success	89% of $\min(\text{reported_usage} \times \text{rates}, \text{max_price}) + \text{tip}$ (10% burned, 1% Treasury)	$\text{max_price} - \text{actual_payment}$

Outcome	Runner Payment	Actor Refund
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

LLM outputs are inherently non-deterministic — the same 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 <code>max_wall_time</code>	Reputation penalty, no payment
Empty/garbage output	Output below <code>min_length</code> 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.

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

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

Snapshot Modes. 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 ≥ minimum_validator_stake` (governance-tunable), self-stake only (no delegation at launch), compliant validator software. No protocol cap on validator count.

Lifecycle: Register (stake CBY, submit BLS12-381 key) → Activate (next epoch boundary) → Operate (propose, vote, earn rewards) → Exit (signal unbonding) → Withdraw (after 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

Cowboy uses a conservative slashing model — most offenses result in jailing (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	50% (40M cyc)	1.0×	Validator updates, governance, slashing
Timer	11% (8.8M cyc)	1.0×	Scheduled timer executions
Runner	11% (8.8M 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. Each lane’s basefee is `lane_basefee = global_basefee × lane_fee_multiplier`; 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 (≤ 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

All persistent actor storage is subject to **state rent** — an ongoing fee for reserving space in the global state trie. The genesis rate is **2,739,726,027,397 atto per allocated byte per rent epoch** (approximately **0.001 CBY/byte/year** at 365 epochs) on allocated quota above the 10 KiB grace threshold. The rate is Tier-0 governance-tunable; changes take effect only at a rent-epoch boundary and apply prospectively.

Rent is auto-deducted from the actor’s balance each rent epoch (default: 1 day). Accounts pay only for allocated quota above the 10 KiB grace threshold. Unpaid amounts accumulate as debt until repayment or eviction at the configured threshold.

0.12.3 Grace Period and Eviction

When an actor cannot pay rent from its balance:

- **Grace period (7 rent-epochs):** The actor remains fully functional, is marked rent-overdue, and accrues rent debt plus a 10% catch-up fee.
- **Warning period (3 rent-epochs):** The actor is marked eviction-imminent and emits warning events for dependent actors.
- **Eviction (after 10 unpaid rent-epochs — 7 grace + 3 warning):** Actor storage and active timers are pruned. Code, address, balance, and the storage root hash are preserved. The actor enters a “dormant” state.

Evicted storage can be restored by anyone who provides the original data, proves it against the recorded root hash, and pays back-rent plus catch-up fees.

0.12.4 Storage Quotas

Each actor has a base storage quota of **1 MiB**, extendable up to **8 MiB** via a storage bond. The bond is locked while the quota is in use, returned when reduced, and forfeited if the actor is evicted. Rent applies to allocated bytes above the 10 KiB grace threshold.

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 ≤ 32); enforce memory (10 MiB), mailbox ($\leq 1,000$ messages, `MAX_MAILBOX_LEN`), and storage quotas; 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 (CREATE2-style).

New actor addresses **MUST** be: `addr = last_20_bytes(keccak256(0xff || creator || keccak256(salt) || code_hash))` where `code_hash = keccak256(python_source_bytes)`. The leading `0xff` domain-separator byte and the `keccak256(salt)` reduction (the salt is hashed to a fixed 32 bytes) follow Ethereum CREATE2 and are consensus-critical — they determine every actor address.

`python_source_bytes` **MUST** be canonicalized as UTF-8, NFC-normalized, LF line endings, and no BOM. The canonical bytes are what are hashed and stored.

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 })`). A transaction comprises: `chain_id`, `nonce`, `instruction`, `cycles_limit`, `cells_limit`, `access_limit`, `max_fee_per_cycle`, `max_fee_per_cell`, `max_fee_per_access`, `max_priority_fee_per_cycle`, `max_priority_fee_per_cell`, `max_priority_fee_per_access`, `from`, `access_list?`, `metadata`, `origin_tx_hash?`, `origin_remaining_cycles?`, `origin_remaining_cells?`, `origin_remaining_access?`, `signature`, `additional_signers` (the four Access fields were introduced by CIP-43).

- `chain_id` is bound into the signing hash (replay protection); a node **MUST** reject a tx whose `chain_id` does not match its own.
- `from` is the 20-byte sender; `additional_signers` is a (possibly empty) list of (`address`, `signature`) pairs that **MUST** be **strictly ascending by address** (no duplicates).
- `origin_tx_hash/origin_remaining_*` are present only on system-injected **deferred** txs (which are not signature-verified); they are absent on ordinary signed txs.
- A deferred child **MUST** satisfy `access_limit <= origin_remaining_access`, preserve `origin_remaining_access` through reconstruction and verification, and carry `max_fee_per_access == 0` and `max_priority_fee_per_access == 0`, like its cycle and cell siblings.
- `access_list` is **optional and reserved** (v1: always absent / `null`, not enforced; reserved for future parallel-scheduling/prefetch). `metadata` is caller-attached bytes (bound into the signing hash).

2.2 Validity checks.

Nodes **MUST** reject a tx if: (a) limits exceed maxima (§12.1), (b) insufficient balance, (c) signature invalid, (d) `chain_id` mismatch, (e) `additional_signers` not strictly ascending / contains duplicates, or (f) encoding is non-canonical / fails to decode (§2.5). (`access_list` is advisory/reserved in v1 and is not validated.) Block validation **MUST** reject `access_limit > BLOCK_ACCESS_BUDGET` before lane accounting or execution, as specified in §4.3. The stricter node-local `MAX_TX_ACCESS_LIMIT` remains admission policy only.

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 EBNF (informative).

```
Tx = chain_id nonce instruction gas from [access_list] metadata [origin]
signature additional_signers      gas = cycles_limit cells_limit access_limit
max_fee_per_cycle max_fee_per_cell max_fee_per_access max_priority_fee_per_cycle
max_priority_fee_per_cell max_priority_fee_per_access      instruction = category
sub_type body      (e.g. System(0) Transfer(1) to amount) origin = origin_tx_hash
origin_remaining_cycles origin_remaining_cells origin_remaining_access (deferred
only) signature = r s v ; additional_signers = [ (address, signature) ... ]
```

2.5 Encoding (normative).

Transactions **MUST** be encoded with the chain's **canonical commonware-codec binary encoding** — the same deterministic, ordered encoding used for `Block`, `Instruction`, `Account`, and all other consensus types (it is NOT CBOR). The encoding is a flat concatenation of the fields in this **frozen order**:

```
chain_id, nonce, instruction, cycles_limit, cells_limit, access_limit, max_fee_per_cycle,
max_fee_per_cell, max_fee_per_access, max_priority_fee_per_cycle, max_priority_fee_per_cell,
max_priority_fee_per_access, from, access_list, metadata, origin_tx_hash,
origin_remaining_cycles, origin_remaining_cells, origin_remaining_access,
signature, additional_signers
```

- Scalar u64 fields (`chain_id`, `nonce`, the limit fields, and the cycle/cell fee fields) use a

minimal varint (UInt); `max_fee_per_access` and `max_priority_fee_per_access` are u128 and use the same minimal-varint family (UInt<u128>), not fixed-width 16-byte encoding. The leading transaction version is `CURRENT_TX_VERSION = 2`, introduced by CIP-43. A deferred `origin_remaining_access` is an `Option<u64>` like its cycle/cell siblings. Collection lengths use a varint length prefix. Fixed-width values (the 20-byte `from`/addresses, the 65-byte `signature = r||s||v`, 32-byte digests) are raw bytes. `Option<T>` is a 1-byte tag (0x00 = absent, 0x01 = present, followed by the value). `Vec<T>` / byte strings are a varint length prefix followed by the elements. `Instruction` is [category, sub_type, body] (e.g. `Transfer = System(0) Transfer(1) to(20B) amount(8-byte big-endian u64)`).

- **The encoding is canonical: every transaction has exactly one valid byte encoding.** Decoders MUST be strict — reject non-minimal varints, non-{0,1} option/bool tags, unknown enum discriminants, and trailing bytes — so `decode(encode(tx)) = tx` and `encode(decode(b)) = b`.
- **The signing hash MUST be** `keccak256(canonical_encode(Tx_without_signatures))` — the canonical encoding with the primary `signature` and every `additional_signers` signature zeroed (signer addresses retained). The transaction `digest` (its identity / BMT leaf) is the same value, so the identity is signature-independent. Every signer signs this hash.

The four `Access` fields MUST participate in canonical `Write`, `Read`, `EncodeSize`, signing preimages, signatures, and strict decoding in the frozen positions above.

Transaction version 2 and the access meter are genesis-active from block 0 of the launch regensis; there is no live-chain activation height. All `Access` wire fields, receipt fields, receipt-root bytes, fee accounting, and metering rules MUST activate atomically with version 2. Before the wipe, implementations MUST run access accounting in observe-only mode, where it MUST NOT affect state roots, basefees, receipts, or any consensus quantity. Before activation, the `Access` weight table MUST be frozen as protocol constants, cross-engine conformance MUST pass, and block budgets and admission limits MUST be recalibrated from the slowest-supported-validator cold-read and finalized-write benchmarks.

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 persistent storage quota:** 1 MiB (governance-tunable) with **state rent** (§4.4).
- **Quota extensions:** An actor MAY post a **storage bond** up to 8 MiB total; rent applies to the full allocated quota.

3.3 Messaging, Reentrancy, Timers.

- **Delivery: Exactly-once after finality** (before finality: **at-least-once**). Each message ID MUST be `keccak256(sender||nonce||msg_hash)` 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.
- **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`.
 - `timer_id = set_interval(every_n_blocks, handler, data)` — Schedule a recurring timer. Returns a unique `timer_id`.
 - `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 — 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 (normative):

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.)

Note: Cells meter bytes (payload, return data, inline blobs ≤ 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; each primitive **MUST** use the same `ceil(structural_units × access_weight_scalar_ppm / 1_000_000)` scaling rule as every other Access primitive. 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 other native STF read, including admission, nonce, balance, and basefee reads and reads by non-CBQS system instructions, is unmetered in accesses; intrinsic access cost is zero. 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.
 - The legacy 100-cycle storage-read and 200-cycle storage-write terms MUST leave `cycles`; read bandwidth MUST leave `cells`; persisted write bytes MUST remain in `cells`. After de-conflation, `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 (≤ 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. Adjustment uses:

$$\text{basefee}_{\{x,i+1\}} = \max(\text{MIN_BASEFEE}, \text{basefee}_{\{x,i\}} * (1 + \text{clamp}((U_x - T_x)/(T_x * \alpha), -1/\alpha, +1/\alpha)))$$

where $x \in \{\text{cycle}, \text{cell}, \text{access}\}$, $\alpha = 96$, the per-block change is capped at $\pm 1/96$ ($\sim 1.04\%$), and $\text{MIN_BASEFEE} = 10,000$. For Access, $U_a = \text{sum}(\text{successful_accesses_used}) + \min(\text{sum}(\text{failed_accesses_used}), T_a)$ 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 .

Natural-Timer accesses and cycles are both excluded from these block inputs. The dedicated timer basefee is cycle-priced; timer accesses pay the live access price without contributing to a later block's `U_a`. Both the nominal 8,192-access transaction-lane sum and the adversarial 24,572-access transaction-lane maximum are sufficient to reach the $+1/96$ per-block cap. The learning rate is calibrated for 1-second blocks so 12-second compounding $(1 + 1/96)^{12} \approx 13\%$ 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.

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: COW-3516**). `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 $2 \times T_a$), `LANE_USER_ACCESS = 2,051`, `LANE_RUNNER_ACCESS = 1,144`, and `LANE_TIMER_ACCESS = 901`. After the COW-3896 decision-2d rebalance 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 ($2 \times 1,024 = 2,048 \leq 2,051$, a 3-unit tail; 1,024 is `USER_TX_ACCESS_LIMIT`), and the Runner lane holds two single-runner, no-delegation `JobResultSubmit` reference suggestions ($2 \times 572 = 1,144$) — the 572 is that fixture's suggestion, not a per-transaction cap, since `RUNNER_TX_ACCESS_LIMIT = LANE_RUNNER_ACCESS = 1,144` lets one Runner transaction claim the whole lane; Timer and System are unchanged. Their nominal budget sum is still **8,192 charged accesses** ($4 \times T_a$), and unused capacity is not reassigned between lanes. Lane admission tests prior use and does not clamp an admitted transaction to the remaining lane budget. The adversarial transaction-lane maximum is therefore $\Sigma(\text{budget} - 1) + 4 \times 4,096 = 8,188 + 16,384 = 24,572$; the fresh 901-access natural-Timer phase raises the adversarial per-block maximum to **25,473 accesses (about $12.4 \times T_a$)**. `BLOCK_ACCESS_BUDGET = LANE_SYSTEM_ACCESS = 4,096` is the charged protocol ceiling for any one workload; `BLOCK_ACCESS_HARD_CAP` is its compatibility alias. Before lane accounting or execution, block validation deterministically rejects a transaction whose signed `access_limit` exceeds that ceiling; it executes nothing and debits no access lane. Otherwise a transaction whose prior lane use is below budget executes with its full signed charged `access_limit`; one admitted transaction may overshoot and later transactions are rejected once use is at or above budget. A transaction that fails after committed-store reads is charged to its lane for those reads. Only a transaction that consumed nothing in any dimension is a pre-execution failure outside lane accounting.

Node #1434 enforces both `access_limit <= BLOCK_ACCESS_BUDGET` and its stricter node-local `MAX_TX_ACCESS_LIMIT = 512` at admission; `COWBOY_MAX_TX_ACCESS_LIMIT` may only lower that local cap. The local cap is not a block-validity rule. These charged values are never scalar-rewritten. Natural Timer records store the structural default 56. In the auction phase, phase reservation clamps the request to the structural cap 26 before scaling; fee escrow continues to use `scale(access_limit_per_fire, p)` on the stored declaration. 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`. The provisional `INITIAL_ACCESS_BASEFEE` is 100,000 attoCBY/access; `access_weight_scalar_ppm` defaults to 1,000,000 ppm. Its codec range is `1 ..= 1,000,000,000`, while the protocol fit bound is `MAX_ACCESS_WEIGHT_SCALAR_PPM = 7,039,062`. The four access lane budgets carry drift-checked `param:` markers in §13 (COW-3896); the remaining Access values (target, basefee, and scalar bounds) intentionally carry no inline `param:` markers.

4.4 State rent. Persistent storage incurs **rent** per byte per rent-epoch, which is governance-tunable. Seven unpaid epochs form the grace period, three more form the warning period, and eviction follows **10 unpaid rent-epochs** in total.

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. **Post:** Actor posts a job with escrowed price.
2. **Assign:** 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 $w = \text{stake} \cdot \text{sqrt}(\text{reputation})$. Every randomized assignment binds the request in the submission block and selects from the verified threshold seed of $r_seed = \text{advance_round}(r_submit, \text{SELECTION_SEED_DELAY_VIEWS})$, where $\text{SELECTION_SEED_DELAY_VIEWS} = 3$, 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. **Commit:** Runner returns `commit = keccak256(output||salt)`.
4. **Reveal:** Runner reveals `{output, salt, proof?}`.
5. **Challenge:** A challenge window of $\text{DISPUTE_WINDOW_BLOCKS} = 75$ **blocks** (≈ 75 s at 1 s/block) is opened, requiring a **100 CBY bond**.
6. **Resolve:** Proven dishonesty triggers stake slashing via the challenge mechanism. Operational failures (timeouts, crashes) result in reputation penalties only.
7. **Payout:** On finalization, **89%** of job payment to runner(s), **10%** burned, **1%** to Treasury.

5.4 Determinism & bounds.

Jobs **MUST** pin `toolchain_digest` and `seed`. On-chain return data **MUST** be ≤ 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; ~ 1 s target block time; **finality on commit** (~ 2 s). 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.

Leader authenticity: 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	50% (40M)	1.0×	Validator updates, governance, slashing
Timer	11% (8.8M)	1.0×	Scheduled timer executions
Runner	11% (8.8M)	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
- Each lane's basefee is $\text{lane_basefee} = \text{global_basefee} \times \text{lane_fee_multiplier}$. 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 $H(\text{"cowboy-tx-eligibility-v1"} \parallel \text{chain_id} \parallel r_{\text{commit}} \parallel \text{parent_hash} \parallel \text{lane} \parallel \text{lane_basefee_snapshot} \parallel \text{merkle_root}(\text{canonical_tx_bodies}))$. 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 $r_{\text{seed}} = \text{advance_round}(r_{\text{final}}, 1)$. Once r_{seed} has notarized or nullified, its verified threshold beacon becomes R_{order} . The set executes no earlier than $r_{\text{exec}} = \text{advance_round}(r_{\text{seed}}, 1)$. Its order is ascending by $\text{HKDF}(R_{\text{order}}, \text{"cowboy-tx-order-v1"} \parallel \text{lane} \parallel \text{tx_hash})$, 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 $(r_{\text{final}}, B_{\text{commit}}.\text{hash}, \text{lane})$. 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 are tagged by lane type. Cowboy does not require private builders or an encrypted mempool at launch; the resulting inclusion, private-order-flow, and JIT risks are explicit limitations rather than risks claimed to be eliminated by ordering.

6.5 MEV Reduction (Limitations Apply).

Cowboy's MEV mitigation strategy combines multiple mechanisms:

VRF-based leader election: 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.

VRF-based transaction ordering: 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.

Out of scope. 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.

Fast finality: 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.

Dedicated lanes: 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.

Public mempool: 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.

Blobs have **no protocol-level TTL or automatic expiry** at genesis. Their lifetime is tied to ownership and rent, not to a timer:

- **Inline blobs** (≤ 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).

Durability and retrievability come from CBFS Reed-Solomon replication plus on-chain Proof-of-Retrievability challenges, not from a retention timer; 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, rennumbers, 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; <code>system:dual_basefee</code> is a frozen legacy storage-key identifier, not a two-meter rule
0x07	Entitlement Registry	Runner Pool access control and general RBAC
0x08	Treasury	Protocol fee collection and distribution, including the 1% Runner settlement fee
0x09	Governance	On-chain parameter governance, including <code>SettlementConfig</code> 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	Stream Key Manager	Paid-stream access, billing, and content-key SealRequest emission to CBSS
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	CIP-29 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
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 **0x19**.
4. The full low-address space below **0x100** is reserved in the sense of state-rent exemption (`RESERVED_SYSTEM_ADDRESS_LIMIT`): actors below it are exempt from rent. User addresses are `CREATE2`-derived from (`deployer`, `salt`, `code_hash`), making the band unreachable by construction. The PVM separately rejects the dense system band **0x01–0x0F** for `fee_payer` overrides; **0x1D** sits deliberately outside that enforced band.

Field-level instruction schemas, registry semantics, and on-chain encodings for each system actor are elaborated in the CIP series.

0.24 10. Developer Experience (DX)

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

0.25 11. Governance & Upgrades

- **Model:** 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.
- **Access parameters:** `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.
- **Timelocks:** Standard actions **7 days**; emergency fast-track **6 hours**.
- **Upgrades:** **Hot-code upgrades** coordinated by governance.

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`, `node/types/src/constants.rs`; 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:

- **Consensus operations:** publish validator-concentration assumptions, weak-subjectivity and long-range recovery procedures, measured one-second-block behavior across realistic geographic latency, and correlated-slashing analysis.
- **Client and VM safety:** maintain deterministic conformance vectors, differential PVM execution, sandbox-escape review, upgrade compatibility tests, and an explicit client-diversity plan.
- **Access-meter conformance:** clients MUST reproduce `docs/cips/cip-39-gas-vectors-v2.json` with schema `cowboy.cbqs.gas-vectors.v2` and normative SHA-256 `e097e2a3b49af0b6bd0b949853c52fd43757c8a6d44b2c9387926c23f023907a`. That artifact retains CIP-39's 27 CBQS-dispatch paths across all eight native instructions, covering every required success and rejection branch; it removes the legacy state-read/write cycle terms, records one R access per charged read and one W access per charged write at the identity scalar, and leaves every other main-side vector field unchanged. It MUST be re-blessed against the node before activation. Cross-engine conformance MUST also use the node-generated PVM host-call access corpus and require in-process and zygote equality of `accesses_used`, `receipt`, and `state root` for every per-host-method table row, including canonical alias keys, nested-tree/revert behavior, speed invariance, and write-flush cadence. 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.
- **Threshold services:** document DKG membership changes, epoch-key rotation, lost-share recovery, collusion thresholds, and failure behavior when quorum is unavailable.
- **Runner selection:** 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.
- **Provider exit:** test data export, provider replacement, revocation, and recovery for every subsystem described as part of the sovereign substrate.
- **Governance and bridges:** 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`; `storage_quota_per_actor = 1 MiB`; `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` = $\pm 1/96$ ($\sim 1.04\%$ /block) ; `MIN_BASEFEE` = 10,000 . CIP-43 introduced provisional, **benchmark-gated: COW-3516** Access defaults: `T_a` = 2,048 accesses; System/User/Runner/Timer lane budgets 4,096 / 2,051 / 1,144 / 901 charged accesses (COW-3896 decision-2d rebalance; see §13 lane budgets below); nominal transaction-lane budget sum 8,192; adversarial transaction-lane maximum 24,572; adversarial per-block maximum 25,473 after the fresh 901-access natural-Timer phase; `BLOCK_ACCESS_BUDGET` = `LANE_SYSTEM_ACCESS` = 4,096 as the consensus per-workload ceiling; `INITIAL_ACCESS_BASEFEE` = 100,000 attoCBY/access; node-local admission cap `MAX_TX_ACCESS_LIMIT` = 512 accesses; and `access_weight_scalar_ppm` = 1,000,000 ppm. The scalar codec range is $1 \dots 1,000,000,000$, but protocol configuration is bounded by `MAX_ACCESS_WEIGHT_SCALAR_PPM` = 7,039,062. 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` = 50% (`LANE_SYSTEM_CYCLES` = 40,000,000) ; `user_lane_capacity` = 28% (`LANE_USER_CYCLES` = 22,400,000) ; `runner_lane_capacity` = 11% (`LANE_RUNNER_CYCLES` = 8,800,000) ; `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 1,144 , and Timer 901 charged accesses. They remain provisional **benchmark-gated: COW-3516**, use the cycle classifier, and do not spill between lanes. After the COW-3896 decision-2d rebalance 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, no-delegation reference `JobResultSubmit` suggestions (572 live each $\rightarrow$ 1,144; not a per-transaction cap, since `RUNNER_TX_ACCESS_LIMIT` = `LANE_RUNNER_ACCESS` = 1,144) — rather than the cycle-lane percentages; System ($2 \times T_a$) and Timer are unchanged, and the three payer lanes still sum to 4,096. Their nominal sum is 8,192. Because an admitted transaction runs with its full

access limit rather than the lane remainder, adversarial transaction-lane use can reach 24,572; the fresh 901-access natural-Timer phase raises the adversarial per-block maximum to 25,473.

Lane fit does not imply simultaneous full-stamp selection. Under the current proposer reservations — a charged block-start seed (1,000 accesses, 1,350 at the live $1.35\times$ scalar) reserved ahead of payer rows, plus the complete Timer-lane hold — two `JobResultSubmit` suggestions and two maximal User stamps do not all fit one block; the exact selected/deferred counts are pinned in node tests, and a planned follow-up (COW-3898) is expected to reduce the Timer reservation to the parent-state due-timer bound. This aggregate-block policy is separate from the isolated lane budgets above.

`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 ( $\approx 75$  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); `grace_period_epochs = 7`; `warning_period_epochs = 3`; `eviction_threshold_epochs = 10`; `grace_threshold = 10,240 bytes` (10 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 , α , δ) — 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 genesis 2,048 / 1,000,000, (**benchmark-gated: COW-3516**); Tier-0. The scalar codec accepts `1 ..= 1,000,000,000`, while protocol governance MUST enforce `MAX_ACCESS_WEIGHT_SCALAR_PPM = 7,039,062`; the fit bound is derived from the Timer and deferred-child allowances. 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.
- **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 $1 \dots 1,000,000,000$, but protocol validation **MUST** additionally reject any governed value above `MAX_ACCESS_WEIGHT_SCALAR_PPM` = 7,039,062. 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

- **Execution:** Cowboy runs deterministic Python actors in the PVM. Ethereum L1 runs EVM bytecode; rollups may define EVM or custom execution environments.
- **Fees:** 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.
- **Timers:** 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.
- **Off-chain compute:** Cowboy enshrines Runner registration, assignment, settlement, and trust profiles. Ethereum applications choose among oracle networks, coprocessors, TEEs, proofs, and centralized services.
- **State:** Cowboy charges rent above a grace threshold and supports eviction/restoration. Ethereum L1 state is currently persistent; statelessness and state-expiry designs remain active research.
- **Security:** 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:** Permissions the actor requires.
- **Runner Entitlements:** 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.
- An actor, through a host call, MAY verify an EIP-712 signed data structure against a given Ethereum address, enabling actors to validate off-chain authorizations from Ethereum users.

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 <code>apply_finalized</code>
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.

Three on-chain priced dimensions: 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:

Transaction Type	Base Cycles	Base Cells	Notes
Transfer	21,000	0	EOA-to-EOA value transfer
Deploy	100,000	<code>code_size</code>	Actor deployment
ActorMessage	21,000	<code>calldata_size</code>	Method invocation
LlmRequest	10,000	<code>prompt_size</code>	Off-chain inference request
TimerSchedule	5,000	64	Schedule future execution

Note: 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.

The intrinsic access cost is 0 for every transaction. Consensus store I/O is charged separately when an Access primitive is invoked.

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
<code>send_message()</code>	80 cycles	—
<code>storage_read()</code>	0 cycles; consensus store reads charge accesses	R or Scan(n) under the Access table
<code>storage_write()</code>	0 cycles; consensus store writes charge accesses	W or Copy(n) under the Access table
<code>hash()</code>	per algorithm (keccak256 ~6/32B, sha256 500+8/block)	—
<code>verify_signature()</code>	5,000 (Ed25519) / 10,000 (secp256k1) / 8,000 (BLS)	—
<code>get_block_info()</code>	100 cycles	—
<code>emit_event()</code>	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 `ACCESS_WEIGHT_TABLE_VERSION` = 1. All numeric weights below are provisional and MUST be recalibrated before activation.

Primitive	Provisional structural weight
R	1 ⟨benchmark-gated: COW-3516⟩
W	1 ⟨benchmark-gated: COW-3516⟩
Scan(<i>n</i>)	5 + ceil(<i>n</i> /2) ⟨benchmark-gated: COW-3516⟩
Copy(<i>n</i>)	<i>n</i> × W_COPY_PER_ITEM ⟨benchmark-gated: COW-3516⟩
Fanout(<i>k</i>)	<i>k</i> ⟨benchmark-gated: COW-3516⟩
NestedFrame	1 ⟨benchmark-gated: COW-3516⟩

Protocol constant	Provisional value
W_READ	1 ⟨benchmark-gated: COW-3516⟩
W_WRITE	1 ⟨benchmark-gated: COW-3516⟩
W_SCAN_BASE	5 ⟨benchmark-gated: COW-3516⟩
W_SCAN_PER_ROW_NUM / W_SCAN_PER_ROW_DEN	1 / 2 ⟨benchmark-gated: COW-3516⟩
W_COPY_PER_ITEM	1 ⟨benchmark-gated: COW-3516⟩
W_FANOUT_PER_ITEM	1 ⟨benchmark-gated: COW-3516⟩
W_NESTED_FRAME	1 ⟨benchmark-gated: COW-3516⟩

Every primitive charge MUST be scaled independently:

```
units_charged = ceil(structural_units × access_weight_scalar_ppm / 1_000_000)
```

The multiplication, division, and ceiling MUST use unsigned 128-bit integer arithmetic. A primitive charge is exactly one invocation of the gate’s charge function: one for each distinct cold logical-key point read *R* after warm deduplication, one for each distinct cold logical out-key write *W*, one for each *NestedFrame*, and one aggregate preflight for each *Scan(n)*, *Copy(n)*, or *Fanout(k)*. The base and per-item structural units of an aggregate preflight MUST be summed before scaling it once. Implementations MUST NOT group separate primitives before scaling or split one aggregate preflight into multiple scaled charges. *Rk(k)* denotes *k* independently scaled *R* primitives subject to warm deduplication; *k* × *NestedFrame* denotes *k* separate primitives; *W(out-keys)* denotes one *W* primitive per distinct cold logical out-key not already warmed by an aggregate preflight.

For *access_weight_scalar_ppm* ≥ 1, every non-zero structural weight MUST produce at least

one charged unit. `access_weight_scalar_ppm = 1_000_000` is the identity. These scaling and primitive-boundary rules are fixed deterministic rules and are NOT benchmark-gated.

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 committed `kv_count` used for **Copy(n)** as its physical-store bound. 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 **Copy(n)**, **n** is the parent's committed `kv_count` read through the gate at preflight, not the entry count after merging the current transaction's staged parent writeset. **Copy(n)** is the single aggregate out-access charge for the copied child keys and warms every resulting child destination key, including keys materialized by that merge; copied keys MUST NOT also charge **W**.

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. Those two native cases remain unmetered in accesses.

Before the deferred System arm performs its first sender-account read, it MUST reserve the complete 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	$R(\text{sender}) + \text{Scan}(\text{MAX_REGISTERED_RUNNERS}) + W(\text{job envelope})$	5,007
IntentDeposit	$Rk(8) + Wk(4) + 2 \times \text{NestedFrame}$	14
IntentWithdraw	$Rk(7) + Wk(3) + 2 \times \text{NestedFrame}$	12
IntentBroadcast	$Rk(5) + Wk(6)$	11

Rk and Wk retain the per-primitive scalar ceiling rule, and the runner scan is one aggregate primitive. The two nested intent entries reserve the maximum `can_transfer` and `on_transfer` frames; store work inside either actor hook passes through the ordinary typed gate and is charged separately.

`MAX_REGISTERED_RUNNERS = 10,000` is the consensus cap on the active-runner registration set. Registration MUST reject before writing an entry that would cross the cap. If committed state is already over the cap, a scan MUST fail closed before walking the set. At the provisional weights, `Scan(10,000) = 5,005`, so the complete actor-minted `JobSubmit` reservation is 5,007. It exceeds the current 128-access deferred-child allowance and deterministically returns `OutOfGas` before dispatch. Actor-minted job submission MUST remain disabled until COW-3516 jointly resizes the runner bound, reservation row, and child allowance. User-signed `JobSubmit` retains its unmetered native path. All access weights and the runner bound in this subsection are provisional (**benchmark-gated: COW-3516**).

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
<code>token_transfer()</code>	1,000	64
<code>token_transfer_from()</code>	1,500	96
<code>token_approve()</code>	500	32

Operation	Cycles	Cells
<code>token_balance_of()</code>	100	0
<code>token_mint()</code>	1,000	64
<code>token_burn()</code>	500	64
<code>token_create()</code>	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 <code>accesses</code> meter
Calldata	1 cell/byte of transaction data
Event emission	¹ 1 cell/byte of event payload (<code>pvm_host.rs::emit_event</code> ; topic not billed; no cycles)

0.31.5 17.5. State Rent

Accounts exceeding the grace threshold pay ongoing rent. This section provides the operational summary plus the governance monitoring cadence for CBY-denominated rent.

```
rent_due_epoch = max(0, allocated_quota_bytes - grace_threshold_bytes)
                × rent_rate_atto_per_byte_epoch
```

Parameters:

```
grace_threshold_bytes      = 10,240 bytes (10 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)
```

```

grace_period_epochs      = 7
warning_period_epochs    = 3
eviction_threshold_epochs = 10
rent_catchup_bps         = 1,000 (= 10% catch-up penalty)

```

Grace period behavior:

- Allocated quota ≤ 10 KiB: No rent charged
- Allocated quota > 10 KiB: Rent charged on excess allocated bytes only
- The first 7 unpaid rent-epochs form the grace period; the actor remains functional while debt and the catch-up fee accrue
- The next 3 unpaid rent-epochs form the warning period; the actor is marked eviction-imminent and emits warning events
- Eviction follows 10 unpaid rent-epochs; restoration requires the original data, proof against the recorded root hash, debt repayment, and the 10% catch-up fee

CBY-denominated rent — governance monitoring. 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)

Blob storage is **not cell-metered**. Provider payments are direct CBY transfers negotiated off-chain. Retention, provider registration, availability commitments, Proof-of-Retrievability, payment, slashing, and the CBFS Relay Node economic split are specified in the [Cowboy Storage Whitepaper](#), CIP-9, and CIP-31. Failed retrieval proofs trigger the penalty and slashing rules defined there.

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	<code>runner_stake >= max(10,000 CBY, 1.5 × declared_max_job_value)</code>
Verification	Mode-specific agreement, objective checks, attestation, or proofs; subjective quality is not established by settlement alone

The protocol does **not** mandate unit prices for LLM or other job types — terms are market-driven. Volume fees and container policies are declared in runner 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)

Cycles, Cells, and Accesses use independent basefee adjustment:

$$\text{basefee}_{\{x,i+1\}} = \max(\text{MIN_BASEFEE}, \text{basefee}_{\{x,i\}} \times (1 + \text{clamp}((U_x - T_x) / (T_x \times \alpha), -\delta, +\delta)))$$

For Access, $U_a = \text{sum}(\text{successful_accesses_used}) + \min(\text{sum}(\text{failed_accesses_used}), T_a)$ 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 accesses and cycles are excluded from these block

inputs; the dedicated timer basefee remains cycle-priced, while timer accesses pay the live access price without contributing to a later block's U_a .

Where: - $\alpha = 96$ (BASEFEE_ALPHA — calibrated for 1-second blocks, vs Ethereum's 8 for 12-second blocks) - max change = $\pm 1/96$ (~1.04%) **per block** (BASEFEE_MAX_CHANGE_DENOM = 96) - MIN_BASEFEE = 10,000 (floor; keeps the basefee in the geometric-update regime) - $\text{clamp}(v, -1/96, +1/96)$ 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 | $\pm 1/96$ (~1.04%) | | α (BASEFEE_ALPHA) | 96 |

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

Access parameters (provisional — <benchmark-gated: COW-3516>): | Parameter | Provisional value | |———|———: | | Target (T_a) | 2,048 accesses/block | | System lane | 4,096 accesses (50%; $2 \times T_a$) | | User lane | 2,051 accesses (COW-3896 2d: two maximal 1,024 stamps + 3 tail) | | Runner lane | 1,144 accesses (COW-3896 2d: two single-runner, no-delegation reference `JobResultSubmit` suggestions at 572 live; not a per-tx cap — `RUNNER_TX_ACCESS_LIMIT = LANE_RUNNER_ACCESS = 1,144`; see §13) | | Timer lane | 901 accesses (11%; 901.12 rounded to nearest integer) | | nominal transaction-lane capacity | 8,192 accesses ($4 \times T_a$, exact lane-budget sum) | | adversarial per-block maximum | 25,473 accesses ($\Sigma(\text{budget} - 1) + 4 \times 4,096 + 901$; about $12.4 \times T_a$) | | Per-workload protocol ceiling | 4,096 accesses (`BLOCK_ACCESS_BUDGET = LANE_SYSTEM_ACCESS`; `BLOCK_ACCESS_HARD_CAP` compatibility alias) | | Initial basefee | 100,000 attoCBY/access | | Weight scalar | 1,000,000 ppm | | Weight scalar codec range | $1 \dots = 1,000,000,000$ | | Weight scalar protocol fit bound | $1 \dots = 7,039,062$ ppm (about $7.039\times$) | | Node-local transaction admission cap | 512 accesses (node #1434; downward-only `COWBOY_MAX_TX_ACCESS_LIMIT`) | | Max change/block | $\pm 1/96$ (~1.04%) | | α (BASEFEE_ALPHA) | 96 |

The four access lane budgets carry drift-checked **param:** markers in §13 (COW-3896); 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 1,144, and Timer 901 charged accesses (User and Runner rebalanced by COW-3896 decision 2d; see §13). These independently enforced budgets have a nominal sum of **8,192 accesses** ($4 \times T_a$); unused capacity is not

reassigned across lanes. Admission tests prior lane use, so an admitted transaction runs with its full signed `access_limit` rather than being clamped to the remaining budget. The adversarial transaction-lane maximum is therefore $\Sigma(\text{budget} - 1) + 4 \times 4,096 = 8,188 + 16,384 = 24,572$. Natural timers own a fresh 901-access phase, raising the adversarial per-block maximum to **25,473 accesses (about $12.4 \times T_a$)**. Both the nominal 8,192 and adversarial 24,572 transaction-lane totals are sufficient to reach the access-basefee +1/96 cap; natural-Timer usage is excluded from `U_a`. `BLOCK_ACCESS_BUDGET = LANE_SYSTEM_ACCESS = 4,096` is the charged protocol ceiling for any one workload, with `BLOCK_ACCESS_HARD_CAP` retained as an alias. Every transaction uses the same classifier for cycle and access accounting. In canonical block order, `access_limit > BLOCK_ACCESS_BUDGET` first follows a deterministic structured skip/error path with zero execution and zero lane debit. Then, before execution, lane use at or above budget follows the cycle-lane skip/error path; otherwise the transaction executes with its full signed charged `access_limit`, and one admitted transaction may overshoot the lane. A transaction that fails after committed-store reads is charged to its lane for those reads. Only a transaction that consumed nothing in any dimension is a pre-execution failure outside lane accounting. Natural timers persist a structural `access_limit_per_fire`; FIFO execution caps/defaults it at structural 56, while auction execution clamps it to structural 26 before scaling. 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`. Node #1434 enforces the protocol ceiling and the stricter node-local `MAX_TX_ACCESS_LIMIT = 512` at admission; `COWBOY_MAX_TX_ACCESS_LIMIT` may only lower the latter. The protocol ceiling is consensus-enforced, whereas the local maximum is admission-only; neither value is 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.

Basefee burning: 100% of basefee revenue is burned. 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
Sys-tem	40,000,000	4,096	50%	1.0×	Gover-nance, upgrades
User	22,400,000	2,051	28%	1.0×	Regular transac-tions
Run-ner	8,800,000	1,144	11%	1.0×	Runner job results, at- testations, callbacks (incl. stor- age manifest anchors)
Timer	8,800,000	901	11%	1.0×	Scheduled actor execution

The access budgets remain provisional (**benchmark-gated: COW-3516**). The Percentage column above is the cycle-lane split (System/User/Runner/Timer = 50/28/11/11% of the 80,000,000-cycle cap); System access is $2 \times T_a = 4,096$ and Timer is 901 (from 901.12), but after the COW-3896 decision-2d rebalance the access User (2,051) and Runner (1,144) budgets no longer 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, no-delegation reference `JobResultSubmit` suggestions at 572 live — not a per-transaction Runner cap, since `RUNNER_TX_ACCESS_LIMIT = LANE_RUNNER_ACCESS = 1,144`; see the §13 derivation and its lane-fit-vs-aggregate qualification). Their nominal budget sum is 8,192 accesses ($4 \times T_a$). Since each lane can admit a transaction while prior use is `budget - 1`, and that transaction can consume the full 4,096-access protocol ceiling, the adversarial transaction-lane maximum is $\Sigma(\text{budget} - 1) + 4 \times 4,096 = 24,572$. The natural-Timer phase has a fresh 901-access grant, so the adversarial per-block maximum is 25,473 accesses (about $12.4 \times T_a$).

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.
- Lane admission tests prior use. If lane use is already at or above budget, access uses the same pre-execution skip/error path as cycles. Otherwise the transaction executes with its full signed `access_limit`; it is not clamped to the lane remainder, and one transaction may overshoot before later transactions are rejected. A transaction that fails after committed-store reads is charged to the lane for those reads. Only a transaction that consumed nothing in any dimension is a pre-execution failure outside lane accounting. There is no credit-back.
- Natural timer fires execute in a fresh dedicated end-of-block phase under FIFO ordering, `LANE_TIMER_CYCLES`, and `LANE_TIMER_ACCESS`. ⟨benchmark-gated: COW-3516⟩ Each record declares a structural `access_limit_per_fire`; the structural default equals the FIFO cap of 56, while the auction phase clamps it to structural 26 just as it clamps the default cycle request. Pass 1 reserves `scale(min(request, phase_cap), p)` charged accesses. A non-fitting fire carries intact under the cycle policy and never runs against the lane remainder; pass 2 never re-clamps it. At identity, the FIFO and auction caps admit sixteen and thirty-four complete reservations respectively. 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`. The GBA mechanism belongs to the CIP-1 auction contract.
- Per-lane `Fee Multiplier` scales the lane's effective basefee (`lane_basefee = global_basefee × lane_fee_multiplier`); all four lanes default to 1.0× (no subsidy, no surcharge). Tier-0 governance-tunable.

0.31.10 17.10. Scalar-safe protocol allowances

Access table rows, natural-Timer declarations, and protocol work envelopes are structural values. A node converts each structural value `s` to charged authority with the single consensus helper `scale(s, p) = ceil(s × p / 1,000,000)`, using unsigned 128-bit intermediates. Zero remains zero and non-zero structural work never becomes free. The same helper prices access-gate operations and converts protocol allowances when they are reserved or minted.

Natural-Timer FIFO/default 56, auction 26, and each stored declaration remain structural. Escrow uses the scaled declaration; phase reservation and the timer transaction use the scaled, phase-clamped request. Deferred-child gates and child limits scale structural 128. Asynchronous event-fire envelopes use that scaled child authority; their fair shares divide charged remaining

authority, and their reserved subscription writes pass through the scaled gate. The internal runner-resume request is structural `RESUME_ACCESS = DEFAULT_ACCESS_LIMIT = 512`, but it is not independent authority: the synthesized callback child and its committed lineage pool each receive `min(scale(RESUME_ACCESS, p), origin.access_limit)`. 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. Although the scalar codec accepts `1 ..= 1,000,000,000`, protocol configuration validation accepts at most `MAX_ACCESS_WEIGHT_SCALAR_PPM = 7,039,062`, derived from `scale(56, p) <= 901` and `scale(128, p) <= min(2,051, 901)`. The deferred constraint binds. Protocol configuration MUST also maintain `scale(DEFAULT_ACCESS_LIMIT, MAX_ACCESS_WEIGHT_SCALAR_PPM) <= BLOCK_ACCESS_BUDGET`; this ensures that the maximum scalar cannot mint a default internal callback above the per-workload ceiling. These derivations and compile-time assertions are part of COW-3516 recalibration.

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 = 0

    # 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
    total_accesses = intrinsic_accesses + execution_accesses

    # 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. Legacy Transaction Encoding Test Vectors (Version 1)

These vectors predate CIP-43 and specify the former version-1 `commonware-codec` encoding. They are retained as historical inputs only and MUST NOT be used as version-2 conformance vectors. A canonical version-2 corpus MUST replace them before launch-regenesis activation. Hex is lowercase and has no 0x prefix.

[illegible]

Vector 1 — Unsigned (all signatures zero) = the signing/identity preimage:

[illegible]

The **signing hash** = keccak256(the bytes above) =

The transaction **digest** (identity) equals this value (signature-independent).

[illegible]

End of specification.