# TRAIDAG: Certified Ordering over a Proof-of-Work Block Graph

*Parallel block production without a fork-choice rule, with post-quantum quorum finality under partial synchrony*

Tenrai Technologies
tenrai-technologies.org
Technical Whitepaper v1.2.3 · September 2026

**ABSTRACT**

TRAIDAG separates permissionless proof-of-work block production from stake-certified ledger ordering. Miners publish blocks into a directed acyclic graph without a selected chain or a work-derived fork-choice rule. Validators first certify registration and data availability, then execute a deterministic, resource-bounded prefix of the committed registration queue. Every fully validating node independently checks block admissibility, transaction authorisation, the state transition and the resulting commitment.

The protocol distinguishes agreement from state validity. Agreement is maintained while Byzantine voting weight remains below one third of the eligible epoch snapshot. Progress additionally requires a reachable quorum of correct active validators. A certificate cannot override a transition rule, although a coalition outside the agreement fault bound can certify conflicting, individually valid histories. Local receipt order, local clocks and local machine load do not determine the committed state.

The v1 operating profile targets 25 blocks per second and a 2.5-second cut cadence. Registration and execution form separate stages; an uncongested two-stage timing model places block-admission-to-execution finality at approximately 5–7.5 seconds. These are operating targets and model outputs, not measured mainnet results. The design combines ML-DSA-65 authorisation and quorum certificates, SHA-384 consensus commitments, immutable work anchors, exact integer reward accounting and explicit recovery boundaries.

This specification sets out the protocol, its assumptions, the resource model and the validation obligations for deployment. Block production rate and cut cadence can evolve through coordinated protocol upgrades without introducing a proof-of-work fork-choice rule.

## Document scope

This edition presents the TRAIDAG architecture and its consolidated reference profile. The protocol remains a permissionless work graph with stake-certified registration and deterministic execution; it does not adopt GHOSTDAG or any other graph-derived fork-choice rule.

The words MUST and MUST NOT identify protocol requirements. Mathematical claims state their assumptions. Performance figures are identified as exact encoding arithmetic, analytical estimates or measurements; this edition reports no new implementation benchmarks. Reference implementation policies, including relay topology and transaction selection, are distinguished from consensus validity.

The network-specific emission curve, asset denomination, minimum validator bond, initial allocations, initial work target and bootstrap validator keys belong to the activated network manifests. The v1 work function is TenQ; its algorithm identifier and byte-exact consensus evaluation are fixed by the accompanying TenQ work specification. This edition does not invent missing emission or bootstrap values. Section D defines the remaining required contents and startup checks. Missing or inconsistent manifests are an error, not permission to use a local default.

The revision includes concrete implementation contracts for reward readiness, atomic value movement, canonical payment encoding, epoch handoff and bounded settlement. These additions are normative within this edition; they are not evidence that an earlier client already implements them. Mainnet activation requires the compatibility review and validation package in Section 10.

> **Four different events.** A block is *produced* when its proof of work is found; *registered* when a cut places it in the committed queue; *executed* when a later cut applies its transactions; and *paid* when the separate reward conditions are satisfied. Registration alone is not transaction finality.

## Authors

Qiro, Voss and Neryn — Tenrai Technologies
Cryptis and ZeroBytes — Cryptix Network
Shard — Rift Project

Author names are pseudonyms. The document describes a protocol and its deployment requirements; it does not constitute an independent implementation audit.

# Contents

# 1 Introduction

A blockchain must resolve conflicting transactions even when its producers work concurrently. Nakamoto consensus couples this decision to a selected proof-of-work chain: competing blocks do not all remain in the ledger history [1]. The interaction between propagation delay and block interval affects stale-block behaviour and gives well-connected miners an operational advantage [18, 19].

Block-graph protocols explore alternatives to a single production chain. GHOST selects a chain using subtree weight; SPECTRE uses pairwise voting with a weaker ordering objective; PHANTOM and GHOSTDAG develop graph classifications for a scalable Nakamoto-style history [2, 3, 4]. These approaches must not be treated as one identical algorithm. Their ordering guarantees and network assumptions differ. More recent work, including DAG KNIGHT, explicitly studies how to avoid a preselected network-delay parameter in the protocol [5].

TRAIDAG makes a different architectural choice. Proof of work admits contributions from an open population, while a stake quorum certifies which contributions enter the ledger. The graph remains useful for dissemination and reconstruction, but its edges do not choose ledger order. The committed registration queue provides the input to deterministic execution.

> **GHOSTDAG scales its ordering machinery; TRAIDAG removes it from proof of work.** GHOSTDAG derives ledger order from the work graph [4]. TRAIDAG instead certifies registration and a deterministic execution prefix. The distinction is where ordering is decided, not the absence of network or resource constraints.

This separation does not make communication delay irrelevant. It removes local observations of the work graph from the state-ordering rule. A slow link can delay registration, data retrieval or quorum formation; it cannot authorise a node to execute a different prefix of an already committed queue. When the required quorum is unavailable, the protocol preserves the last committed state rather than invoking an alternative work-based ordering path.

## 1.1 Contributions

The central contribution is the combination of an open work layer, quorum-backed registration, a deterministic execution prefix and anchor-based settlement. A registered block acquires a stable queue position. Its transactions are considered exactly once in that position, and its subsidy entitlement is calculated from the closed bucket of blocks sharing its work anchor.

The specification separates the digest that determines execution order from the digest that certifies the completed state transition. This avoids a circular dependency between ordering and the post-state commitment. It also makes explicit which decisions remain under proposer control: registration is discretionary within eligibility and resource limits, while execution of the existing queue is not.

The remaining contributions are precise transition rules for deferred rewards and value conservation, a timestamp-free difficulty input, a stake-weighted agreement model with a global eligible-stake denominator, and a cost model for explicit post-quantum quorum certificates. The arithmetic examples and figures are reproducible from the accompanying calculation script. They do not substitute for an implementation audit or a global testnet.

## 1.2 Scope of the v1 system

The v1 system supports native-asset payments, bonded and delegated stake, stake-certified finality, proof-of-work registration and deferred miner rewards. The work and stake portions of subsidy and transaction fees are each 50%. All value movement is accounted for in disjoint committed state domains.

The compute settlement lane is disabled. Its state commitment is the designated empty root and both compute value domains remain zero. No v1 transaction may activate an AI job, escrow or challenge process. A future compute market would require a separately specified and activated protocol extension; it is not part of the usable functionality claimed for this release profile.

## 1.3 TenQ proof of work

TRAIDAG v1 uses TenQ as its proof-of-work function. TenQ maps the canonical, anchor-bound work preimage to a deterministic 256-bit work value and accepts the block when that value is at or below the target committed by the selected work anchor. Nonce, extranonce and target retain the conventional mining and pool interface. TenQ remains separate from TRAIDAG's ledger ordering: sibling work blocks do not compete for a selected proof-of-work chain.

TenQ separates reusable epoch state from nonce-specific execution. A 32 MiB epoch cache derives a 2 GiB read-only dataset. Each attempt carries 32 logical lanes, sixteen 32-bit registers per lane and an 8 KiB writable scratchpad. Four phase-dependent 256-instruction programs execute over 1,024 rounds, with later program keys derived from earlier terminal state. Each round combines irregular dataset access, private writable state, mixed fixed-width integer arithmetic and explicit cross-lane reduction. Floating point, device timing and implementation-defined overflow are excluded from consensus; Full and Light evaluation return the same work value.

**Production profile.**

| Component | Reference value |
|---|---|
| Algorithm identifier | 0x02 |
| Epoch memory | 32 MiB cache; 2 GiB read-only materialized dataset |
| Attempt-local state | 8 KiB scratchpad; 32 lanes; 16 x 32-bit registers per lane |
| Program schedule | Four dependent programs; 256 instructions per program |
| Execution | 1,024 rounds; one 128-byte dataset node per round |
| Light evaluation | Exact reconstruction of requested dataset nodes from the 32 MiB cache |

The specialization objective is economic, not absolute. TenQ makes specialized hardware reproduce the same combined workload: large shared memory, attempt-local writable state, mixed integer arithmetic including multiplication-high, division and remainder, irregular access, cross-lane reduction and phase-dependent program generation. This broad resource mix is intended to reduce gains from narrow fixed-function pipelines; Full and Light evaluators remain consensus-equivalent.

**Measured implementation performance.**

| Environment | Path | Batch/workers | Median eval/s | Observed range |
|---|---|---|---|---|
| RTX 2070, Windows | CUDA prod2g | 4,096 | 28,964.1 | 28,775.4–29,025.9 |
| RTX 2070, Windows | OpenCL prod2g | 4,096 | 28,832.3 | 28,384.8–29,190.6 |
| RTX 2070, Windows | CUDA prod2g | 16,384 | 28,260.9 | 28,215.4–28,282.8 |
| RTX 2070, Windows | OpenCL prod2g | 16,384 | 28,017.1 | 27,991.7–28,115.7 |
| AMD EPYC 9V74 VM | Full | 1 worker | 570.5 | 566.2–576.4 |
| AMD EPYC 9V74 VM | Full | 4 workers | 1,893.7 | 1,887.7–2,287.2 |
| AMD EPYC 9V74 VM | Light | 1 worker | 32.33 | 29.63–34.56 |

CUDA and OpenCL production throughput on the RTX 2070 agrees within about one percent at matched batch sizes. Generic quantum target search retains Grover's square-root query law, but an oracle call must evaluate the complete TenQ transition. The TenQ Technical Whitepaper v1.0 contains the byte-exact construction, security analysis and complete benchmarks [31].

# 2 System model

## 2.1 Participants and failure assumptions

Miners are an open population. A miner needs no validator identity to publish a block satisfying the work predicate. Validator eligibility and stake are recorded in the finalised registry. A bounded active set signs consensus messages; non-validator full nodes verify the same ledger transitions without contributing votes.

We use partial synchrony: after an unknown stabilisation time, messages between correct, mutually reachable participants arrive within some finite bound $\Delta$ [6]. Before that point, delays may be arbitrary. The liveness model also includes finite validation, storage and data-retrieval time under admitted load. A network link with adequate latency but insufficient sustained throughput does not satisfy the operating assumptions merely because a small ping succeeds.

For epoch $e$, let $W_e$ be the total eligible bonded weight in the committed eligibility snapshot and let $K_e$ be the active validator set. Define

$$Q_e = \left\lfloor \frac{2W_e}{3} \right\rfloor + 1, \qquad S_e = \sum_{i \in K_e} w_i \le W_e. \tag{1}$$

A certificate is valid only if its distinct active signers carry at least $Q_e$ weight. The denominator is $W_e$, not $S_e$. The quorum therefore does not become cheaper because fewer identities fit into the active-set cap.

Agreement requires Byzantine eligible weight $B_e < W_e/3$. Liveness requires the correct, mutually reachable active validators to carry at least $Q_e$. In a deployment with Byzantine active weight $B_{\text{active}}$ and additional unreachable correct weight $O_{\text{active}}$, a sufficient weight condition is

$$S_e - B_{\text{active}} - O_{\text{active}} \ge Q_e. \tag{2}$$

The 80% active-coverage admission floor in the launch profile is not a replacement for this condition. With $W_e = 100$, $Q_e = 67$ and $S_e = 80$, an active Byzantine weight of 30 leaves only 50 correct active units. Safety remains within the stated bound; progress cannot be guaranteed if those 30 units withhold votes.

The adversary may control an arbitrary fraction of hash rate without changing the stake-based agreement threshold. This does not remove economic or denial-of-service consequences of hash-rate concentration.

The adversary is assumed unable to forge the required authorisations or violate the binding properties of the consensus commitments. Key compromise counts against the relevant fault or authorisation assumption.

## 2.2 Cryptographic profile

Spend authorisation and consensus votes use ML-DSA-65 as standardised in FIPS 204. Its public keys are 1,952 bytes and its signatures 3,309 bytes. The standard provides both hedged and deterministic signing; neither variant turns the signature scheme into a verifiable random function [14].

Consensus digests use domain-separated SHA-384 with 48-byte output. This applies to block identifiers, transaction identifiers, state roots, queue roots, validator-set commitments, execution identifiers and cut identifiers. Classical generic collision resistance is 192 bits. Generic quantum collision query estimates and the memory assumptions of those algorithms must be distinguished from classical collision strength [17, 20]. The paper does not express the security of every primitive as one common bit count.

A spend lock contains a one-byte authorisation profile and a 40-byte commitment

$$\mathrm{lock}(pk) = \mathrm{SHAKE256}\big(\texttt{TRAI\_LOCK\_V1} \parallel \mathrm{u8}(1) \parallel pk,\ 320\big). \tag{3}$$

The profile selects ML-DSA-65. The domain and output length are fixed. FIPS 202 specifies SHAKE256 second-preimage strength as $\min(d, 256)$ bits for $d$ output bits; a 320-bit output does not justify a claim of 320-bit classical second-preimage strength [16]. The lock construction is retained without the earlier overstatement about a 160-bit quantum margin.

Proof of work is a separate network-profile predicate. TRAIDAG v1 uses TenQ, which maps the canonical work preimage to a deterministic 256-bit unsigned value. A block is valid exactly when that value does not exceed the target committed by its work anchor. TenQ's algorithm-specific construction, conformance rules and Full/Light equivalence are defined in the TenQ work profile and Technical Whitepaper [31]; Section 1.3 summarises the parameters and measured implementation results relevant here.

## 2.3 Transport and peer identity

The reference transport uses TLS 1.3 with the X25519MLKEM768 hybrid key agreement defined by RFC 10024 [21, 23]. ML-KEM-768 is specified in FIPS 203 [15]. The TLS construction is used within its defined transcript and key schedule, not copied into an unrelated Noise or custom handshake.

Before a peer is accepted at the TRAIDAG application layer, it proves its ML-DSA identity over an exporter-bound authentication transcript containing both endpoint identities, fresh challenges, network identity and protocol version. Both directions must verify. Consensus or private application messages are not accepted before that exchange completes. The exporter label and transcript encoding are versioned in the transport profile. A classical certificate may assist endpoint discovery but is not the sole load-bearing peer authorisation.

Downgrade to a classical-only key exchange is rejected. Implementations validate public-key and ciphertext lengths, follow the primitive's failure handling and separate transport keys from validator and withdrawal keys. Standardised primitives establish the cryptographic basis; secure key generation, integration and side-channel handling remain implementation obligations.

### 2.4 Durable signing and atomic storage

Before releasing a vote, a validator durably records its height, view, phase and signed digest, together with any lock change caused by the event. A restart recovers these records before signing resumes. The same logical signing key must have one authoritative signing service, protected by key-scoped fencing across machines. Independent local logs on two hosts do not prevent equivocation.

Loss of the authoritative vote history makes the consensus key unsafe to resume. It remains offline; the withdrawal authority uses the normal exit or key-rotation path. Restoring an old VM image or database snapshot does not restore permission to sign. Historical signatures remain valid evidence after rotation.

A cut is committed atomically with its state changes, certificate, queue updates, indices and accounting counters. Recovery exposes either the state before the cut or the complete state after it. A valid database checksum is insufficient if it combines a UTXO root from one height with a reward cursor from another.

### 2.5 Terminology and time

A *height* indexes one certified state transition. A *view* is an attempt to decide that height. An *epoch* identifies an activated validator snapshot; its identifier need not advance when an epoch is extended. A separate accounting interval advances every configured number of finalised heights, including during such extensions.

The target cadence $\rho$ is a liveness and capacity-planning parameter. No local wall-clock reading enters a block, transaction or cut validity predicate. Registration windows are measured in finalised heights. Unbonding is measured in completed eligible-set epochs. Their nominal conversions to seconds or days assume the target cadence and normal epoch turnover; they are not calendar guarantees.

## 3 Why conflicting spends require agreement

**Proposition 3.1** (Conflict exclusion)**.** *Let $u$ be an unspent outpoint and let two distinct, individually authorised transactions both consume $u$. Under a ledger rule that consumes each outpoint at most once, a valid successor state can apply at most one of them.*

*Proof.* After either transaction consumes $u$, the outpoint is absent from the resulting unspent set. Applying the other transaction would require consuming an absent input. Applying both against the same predecessor without reserving the input would instead consume it twice, contrary to the transition rule. □

This is a property of the specified UTXO model, not a general impossibility theorem about parallel computation. It does not require every independent transaction to be totally ordered. A symmetric rule that rejects both conflicts is possible, as is a tie-break by transaction identifier. The question is which candidate set that rule is allowed to examine.

A rule evaluated against each node's current receipt set can change when another block arrives. For example, the smallest-identifier winner can be displaced by a withheld transaction with a smaller identifier. Permanent settlement therefore needs agreed inclusion boundaries or another mechanism that establishes when the conflict decision is closed. TRAIDAG obtains that boundary from certified registration and the resulting execution cut.

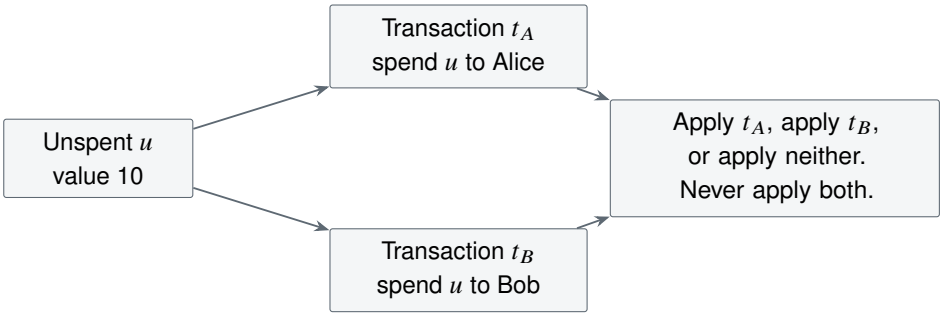


**Figure 1.** Conflicting authorised proposals. If neither transaction executes, $u$ remains unspent. The selection rule acts on an agreed execution input rather than on a node's current mempool.

# 4 Protocol

## 4.1 Blocks and proof-of-work binding

A work block references at most sixteen parent identifiers, names one finalised cut as its immutable work anchor and carries an ordered list of transaction envelopes. It has no post-state root. The block is an admissible proposal for later execution, not a unilateral ledger transition.

The canonical work preimage binds network identity, protocol version, work-profile identifier, work reference, parent commitment, payload commitment, payload length, declared resource mass, reward destination, nonce and extranonce. Its exact byte layout is in Section A. A block identifier is a SHA-384 commitment to this same canonical header. The separately activated work profile evaluates the work preimage; the two roles are not conflated.

The reward destination is work-bound, so a relayer cannot redirect a solved block. The payload root commits to the full witness-bearing transaction identifiers, preventing witness substitution without changing the header. The parent vector is sorted, duplicate-free and length-bound. Transaction order is the order of the list committed by the payload, not a database iteration order or a later miner instruction.

Structural checks precede registration: canonical decoding, work predicate, anchor resolution, parent-reference rules, body commitments, resource limits, transaction structure and authorisation. The amount and existence of every transaction input are checked against the work-anchor state. Stateful effects, including current unspentness, are checked again at execution. A cached signature result never bypasses the second check.

### 4.1.1 Parents and pruning

Parents provide relay context; they do not determine ledger order, difficulty or rewards. They nevertheless create a bounded structural validation dependency. To prevent a hidden unbounded ancestry fetch, the v1 reference rule admits only parent headers whose work anchors are no older than the child anchor's retained registration horizon. Each directly referenced header is validated against its own committed anchor and PoW; its full payload is not a prerequisite for validating the child. The parent relation is not recursively replayed as an execution history.

A missing header is pending data, not proof of invalidity. Nodes retain validated headers and the corresponding anchor descriptors while a currently admissible child may reference them. Parentless blocks are valid, which permits bootstrap and allows mining when no suitable retained parent is available. A miner that supplies an invalid parent reference cannot rely on another node's larger local archive to make the block valid.

## 4.2 Cuts, identifiers and state commitments

A cut at height $h$ binds the predecessor cut, the deterministic execution set $E_h$, the new registration batch $R_h$, the post-state root, the validator set in office and the activated parameter hash. Epoch-transition metadata is part of the committed state and cut metadata. A valid certificate authenticates the cut identifier; signer ordering and signature randomness do not alter that identifier.

Execution and certification use separate digests:

$$\text{execution\_id}_h = H(\text{EXEC} \parallel \text{chain} \parallel \text{version} \parallel h \parallel \text{prev\_cut\_id} \parallel \text{exec\_root}_h), \tag{4}$$

$$\pi_h(b) = H(\text{ORD} \parallel \text{execution\_id}_h \parallel \text{id}(b)), \tag{5}$$

$$b \prec_h b' \iff (\pi_h(b), \text{id}(b)) <_{\text{lex}} (\pi_h(b'), \text{id}(b')).$$

$$\begin{aligned} \text{cut\_id}_h = H(&\text{CUT} \parallel \text{chain} \parallel \text{version} \parallel h \parallel \text{prev\_cut\_id} \parallel \text{execution\_id}_h \\ &\parallel \text{registration\_root}_h \parallel \text{post\_state\_root}_h \\ &\parallel \text{validator\_set\_id} \parallel \text{params\_hash} \parallel \text{transition\_id}). \end{aligned} \tag{6}$$

Here $H$ denotes the canonical domain-separated SHA-384 construction. The abbreviated domain names and field layouts are expanded in Section A. The execution-set root commits to sorted, unique block identifiers; the execution order then follows Equation (5).

The post-state root cannot be part of the ordering seed: execution needs the order before it can compute that root. Conversely, it must be part of the certified value. The split satisfies both requirements. No current cut identifier is inserted into the state whose root that same identifier commits; the finalised-history index is extended atomically after the identifier has been computed.

### 4.2.1 What determinism does and does not imply

Given the predecessor history and committed queue, a proposer has no freedom to choose a different execution prefix or permute it. The registration batch remains a choice among admissible available blocks. It changes the current cut identifier and can therefore influence the seed at the next height. A proposer may examine alternative admissible batches for this purpose.

The protocol claims deterministic execution, not an unbiased randomness beacon, universal fairness or the absence of ordering-related extraction. The hash permutation does not establish that profitable grinding is impossible. These are incentive and inclusion questions outside the state-equality theorem in Section 5.1.

### 4.2.2 Committed domains

The state root commits to the UTXO map, queue, live block-status records, validator registry, bonded and unbonding positions, proposer priorities, epoch state, reward tasks and payout cursors, emission accounts, difficulty state, parameter state and the empty compute state. Every variable consulted by a future transition belongs to one of those domains.

State maps use the canonical authenticated-map rules of Section B. Explicit counts, domain tags and collection orders prevent alternate serialisations. Unknown consensus fields and transaction types are rejected. A local implementation may choose any storage engine, but its root must match the specified commitment over logical records.

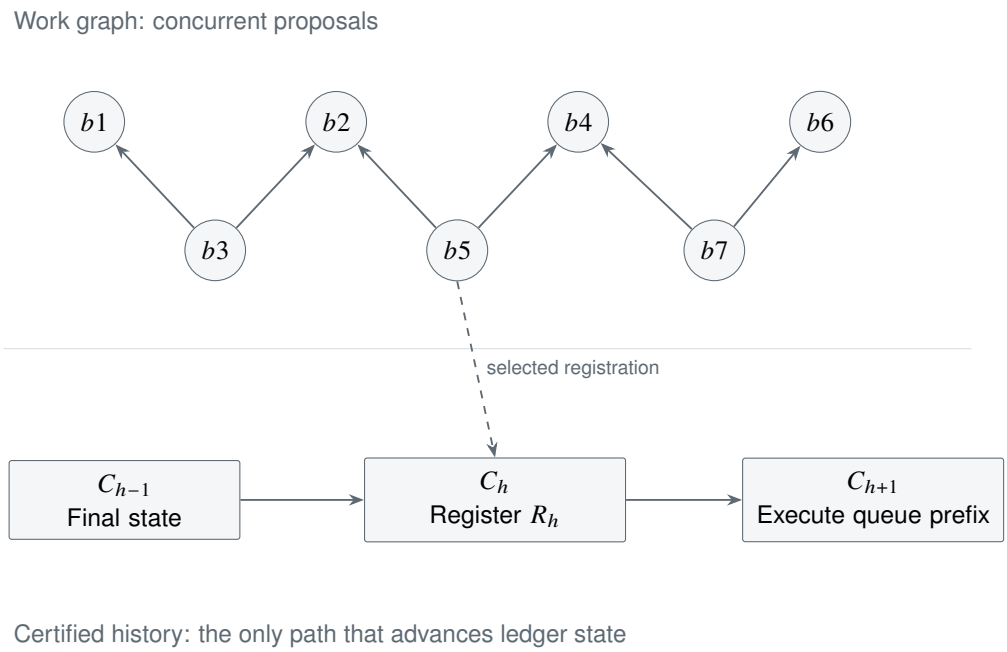


**Figure 2.** The work graph and certified ledger are separate structures. Parent edges guide data retrieval but do not choose the state. Each work block also binds a finalised work anchor, omitted here for clarity. Registration commits a batch; a later cut executes the required queue prefix.

## 4.3 Work anchors and eligibility

The header field `work_ref` identifies a finalised cut known before the block can be registered. Let $a(b)$ be its height. Registration at height $h$ is admissible precisely when

$$a(b) < h \le a(b) + W_{\mathrm{REG}}(a(b)), \qquad \neg\mathrm{registered\_once}(b). \tag{7}$$

The anchor descriptor fixes the PoW target and registration-window version applicable to that block. A later parameter change cannot silently extend its lifetime or replace its target.

The rule authenticates the declared work reference. It cannot determine the wall-clock instant at which a miner ran the hash computation. Nodes establish that the reference is in the accepted finalised history and precedes the registration height; they do not attempt to verify a claimed mining timestamp.

Once registered, a block does not expire from the execution queue. Its position remains until executed. A block that was only produced or relayed has no such guarantee. Under overload, censorship or insufficient data availability, it may miss the registration deadline.

A complete finalisation halt freezes the height counter, so a height-based window does not expire while that halt continues. A large work backlog can still exceed the registration capacity available after progress resumes. The expiry occurs as finalised heights advance again, not as wall-clock time passes during the halt.

## 4.4 Registration and deterministic queue service

A proposal at height $h$ carries $R_h$ in addition to $E_h$. A correct validator votes for the proposal only after obtaining and validating every body needed for both sets. The registration batch may contain only structurally valid, eligible, previously unregistered blocks, and it must satisfy the per-cut registration limits.

The queue update is

$$E_h = \mathrm{prefix}(Q_{h-1}; \mathcal{B}_{\mathrm{exec}}), \tag{8}$$

$$Q_h = (Q_{h-1} \setminus E_h) \mathbin{+\!\!+} \mathrm{canon}(R_h). \tag{9}$$

The prefix is the longest initial segment whose cumulative block count, encoded bytes, mass and authorisation count satisfy all execution limits. Scanning stops at the first entry that would exceed any limit. A later smaller block cannot be selected instead. Counters include each occurrence even when a transaction signature is cached or the transaction later has no effect.

New entries append in ascending $(a(b), H(\mathrm{REG} \,\|\, \mathrm{prev\_cut\_id} \,\|\, \mathrm{id}(b)), \mathrm{id}(b))$ order. Their full queue key also includes the registration height, so later batches cannot be inserted ahead of earlier ones. The same order is the reference proposer's priority among its currently available candidates. Omitting an unregistered block is not a proposal-validity violation merely because another validator has received it.

**Table 1.** Core capacity limits retained from the v1 profile. Byte limits are binary units. Mass is a deterministic logical cost, not a local timing measurement.

| Resource | Registration per cut | Execution per cut |
|---|---|---|
| Blocks | 96 | 128 |
| Encoded block bytes | 8 MiB | 12 MiB |
| Authorisations | 4,096 | 6,144 |
| Mass | 32 $M_{\mathrm{block}}$ | 48 $M_{\mathrm{block}}$ |
| Committed queue | 4,096 blocks and 256 MiB | |
| Individual block | 1 MiB; $M_{\mathrm{block}}$ = 2,000,000 | |

A registration batch is checked against the queue remaining after the deterministic execution prefix has been removed. No admitted block may exceed any individual execution-resource limit. This prevents a valid queue head from becoming permanently unexecutable. System settlement has a separate reserved budget and never shortens this prefix according to CPU availability.

### 4.4.1 Availability obligations

A quorum-backed registration establishes that correct signers carrying more than $W_e/3$ held the bodies when voting, under the fault assumption. It does not establish perpetual availability without a retention rule. Registered bodies must be served until execution and any successor-set handoff have discharged the relevant obligation.

Data held for a noncommitted proposal is provisional. It can be released once the height is irrevocably decided differently, unless another committed registration or currently required certificate still references it. A mere view timeout is not sufficient: the proposal may remain a valid value and be re-proposed.

Nodes that lack committed queue data fetch it and wait. They do not skip entries, reconstruct invented transactions or accept an unverified state root. A permanent loss of the required bodies is a liveness failure. Global service limits on provisional data, request concurrency and invalid-message traffic are local defensive policies; they cannot change the logical result of validating a complete cut.

## 4.5 Consensus state machine

The agreement layer uses the `validValue`/`validRound` locking discipline in Algorithm 1 of *The latest gossip on BFT consensus*, arXiv:1807.04938v3 [8]. This is the algorithmic reference for this edition. An

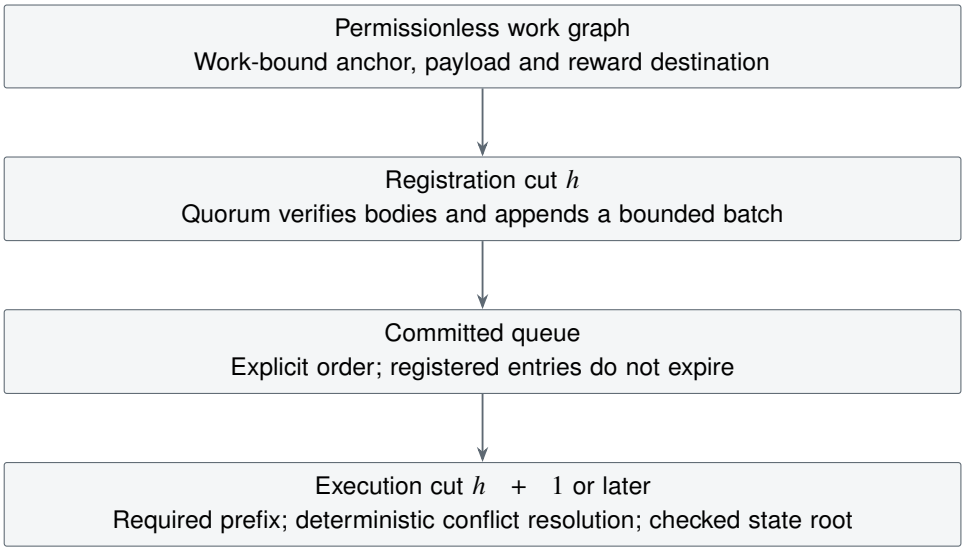


**Figure 3.** The two-stage ledger pipeline. Execution at $h + 1$ is the earliest possibility after registration at $h$; a pre-existing queue can delay it. No parent edge determines ledger order.

implementation must not combine its NIL-vote rules with a different CometBFT variant merely because both are described as Tendermint-style.

The mapping replaces the proposed application value by a TRAIDAG cut, replaces vote counts by eligible-snapshot stake weight and strengthens the application-validity predicate with registration availability and deterministic state execution. The mapping and the event guards below must be tested together. They are not a claim that an unmodified external client implements TRAIDAG.

### 4.5.1 Local state and committed proposer state

For the undecided height a signer holds the current view and phase, its locked value and lock view, its most recent valid value and valid view, its received evidence, one-shot timer flags and its durable signing history. Absent lock and valid views are represented by $-1$; transmitted views are nonnegative `u64` values. A local wider signed representation is used for the sentinel, so the wire view range is not truncated to `i64`.

Proposer priorities are different: they are committed state and do not depend on which views an observer happened to see. Let $P_h$ be the priority vector in force at the start of height $h$. With $S = \sum_{i \in K_e} w_i$, one priority step adds $w_i$ to each entry, selects the largest entry with an ascending-validator-ID tie-break, subtracts $S$ from the winner, centres the vector using floor division and clamps entries to $[-S, S]$.

```
Advance(P, n):
    Q = copy(P)
    repeat n times:
        for i in canonical_validator_order:
            Q[i] = Q[i] + weight[i]
        leader = argmax_i (Q[i], reverse_lexicographic_id[i])
        Q[leader] = Q[leader] - total_active_weight
        mean = floor(sum(Q) / active_validator_count)
        for i in canonical_validator_order:
            Q[i] = clamp(Q[i] - mean, -S, S)
    return (Q, leader)

leader(h, v) = Advance(P_h, v + 1).leader
P_(h+1)      = Advance(P_h, 1).priorities
```

The tie-break in the code means the smallest validator identifier wins equal priorities. `Advance` is pure. Views evaluate a scratch copy; a committed height advances $P$ by exactly one step regardless of its successful view. All priorities reset to zero when a new committee epoch actually activates, not on an extension interval.

Intermediate addition, summation and centring use signed 256-bit arithmetic. Committed priorities use signed 128-bit two's-complement encoding. A valid manifest constrains total eligible stake to less than $2^{120}$ and the active set to at most the protocol's registry bound; the launch active cap is 64. These bounds leave explicit headroom for intermediate operations. Clamping after an overflowing addition is not an acceptable implementation.

A node validates evidence before doing work proportional to a remotely supplied view number. Implementations may cache consecutive schedule states. A single signed message with a huge view cannot force an expensive schedule walk or change the local view. Counters never wrap; exhaustion is a defined stopped state requiring a protocol upgrade.

The schedule weights proposer opportunities by stake rather than by the number of identities. This edition does not assert an unproved finite-window coalition discrepancy bound for the centring-and-clamping algorithm. In particular, the registration window is not justified by counting a supposed 64-identity round-robin cycle.

### 4.5.2 Messages, signatures and certificates

A vote signs a domain-separated digest of chain identifier, protocol version, validator-set identifier, height, view, phase and value identifier. NIL is a distinct tagged value, not an all-zero cut identifier. Proposal signatures additionally bind the claimed valid view and the identity of any supporting prevote quorum.

Evidence identity is defined by its semantic message fields and canonical signer set. Randomised signature bytes are carried and verified as proof, but are not used as an ordering seed or to update proposer priorities. Two valid encodings of a commit proof for the same cut do not define two ledger states.

Certificates use strictly increasing active-set indices with no duplicates. Verification looks up each key and fixed weight in the height's committed validator snapshot, verifies every included signature and sums each signer once. Unknown indices, zero-weight entries, duplicate signers, inconsistent phase or view and weight overflow are errors. A certificate that reaches quorum still fails if its cut is transition-invalid.

### 4.5.3 Cut validity

A proposal is well-formed only if it is signed by the scheduled leader for that height and view and binds the correct predecessor, committee and parameter set. Its execution set must be the exact prefix from Equation (8). The registration batch, bodies, anchors and all resource counters must be valid. Executing the entire transition, including rewards and epoch bookkeeping, must reproduce the claimed post-state root.

A claimed prior valid view must satisfy $0 \le r < v$ and carry a verified prevote quorum for the same cut at $r$. A proposal is never made valid by a timeout. Data not yet fetched is treated as unavailable for voting, rather than guessed or accepted from a certificate alone.

### 4.5.4 Voting events and lock guards

The following rules are applied with durable one-shot guards. A signer emits at most one prevote and one precommit for each $(h, v)$. It may relay or retransmit the identical signed bytes without producing another logical vote.

**Table 2.** Consensus event rules. The commit rule is checked independently of the current phase or view.

| Event | Required action |
|---|---|
| Enter a view | Start in proposal phase, preserving the lock and valid-value record. A correct leader with a valid value re-proposes that exact cut and its evidence; otherwise it constructs a valid cut from the predecessor state. |
| Fresh valid proposal | With no prior valid-view claim, prevote its value only if unlocked or locked on that same value. Otherwise prevote NIL. |
| Proposal with prior quorum | For evidence view $r < v$, prevote its value if it matches the lock or if the lock view is at most $r$. Otherwise prevote NIL. The evidence and cut must both validate. |
| Proposal timeout | If still in proposal phase, persist and emit a NIL prevote, then enter prevote phase. A late proposal cannot cause a second prevote. |
| Quorum of prevotes, any values | Once distinct current-view prevoters carry $Q_e$, start the prevote timeout if still in prevote phase. Each signer contributes weight once. |
| Non-NIL prevote quorum | After the value and its current-view proposal are validated, update the valid-value record. If still in prevote phase, persist the new lock and emit the non-NIL precommit atomically. If already in precommit phase, do not change the lock or send another precommit. |
| NIL prevote quorum or prevote timeout | If still in prevote phase, emit NIL precommit and enter precommit phase. Existing locks are retained. NIL does not unlock a value. |
| Quorum of precommits, any values | Once distinct current-view precommitters carry $Q_e$, start the precommit timeout. A matching non-NIL commit proof remains eligible for immediate processing. |
| Precommit timeout | If the height is still undecided, advance to the next view. Preserve lock and valid-value evidence. |
| Higher-view evidence | Messages at a common higher view from distinct active signers carrying more than $W_e/3$ permit a view jump. Authenticate and deduplicate evidence first. An isolated high-view message is insufficient. |
| Non-NIL commit certificate | At any view of the current undecided height, validate the certified cut, data and complete transition. Commit it atomically even if the local node has already timed out into a later view. |

The prior-quorum guard uses locked_view $\leq r < v$, matching the pinned algorithm. Under the fault assumption a conflicting quorum at the same lock view cannot coexist with the quorum that established the lock. This fact belongs to the safety argument; implementations still verify the stated inequality literally.

A late current-view quorum after a NIL precommit may update `valid_value`; it cannot authorise a second precommit. Evidence from old views is retained for justification and commit verification, without

replaying the local signing actions of those views. A signer that has finalised the height does not reopen it.

When a re-proposal is required, its registration batch, execution set and post-state root are unchanged. Appending newly received blocks to a previously valid value creates a different value and cannot be advertised as the old one.

### 4.5.5 Pacemaker and height pacing

Timeouts are local monotonic timers. For phase $x$, the reference policy is $t_x(v) = t_{x,0} + v\delta_x$ with positive, finite base and increment values. They reset at a new height. Timer events are ignored when their recorded height, view or phase no longer matches local state. Processing a valid commit certificate takes precedence over continuing an obsolete local wait.

The reference implementation uses a finite pacing wait before starting the next height so that healthy, fast decisions do not deliberately run the cadence far above the target. It does not postpone accepting or publishing a valid certificate. The wait is measured locally from the preceding height start and is bounded by the target cadence; it is neither a block-validity rule nor a globally enforceable real-time clock.

The configured pacing and timeout values belong to the published deployment profile. Their adequacy must be measured over the intended network. Increasing timeouts preserves the state-validity rules while trading latency for a larger progress margin. During a partition, timers never reduce stake or create a substitute quorum.

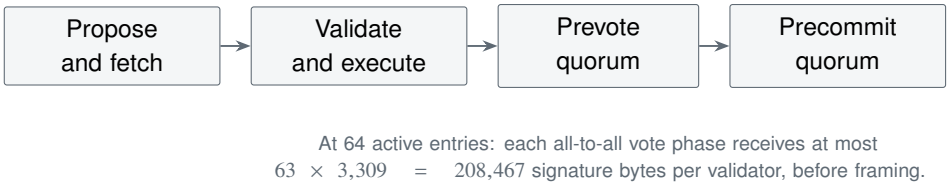


**Figure 4.** Consensus message pattern. Boxes are schematic, not measured phase durations. Execution includes the whole bounded state transition. A full-node-facing commit proof is separate from the two internal vote phases.

## 4.6 Committee epochs, stake and handoff

### 4.6.1 Eligibility and the active set

A validator has a stable registry identifier, a consensus key, an owner or withdrawal authority and self-bonded stake. Delegated positions remain owned by their depositors and carry an explicitly registered reward lock. The consensus key cannot move the owner's principal or a delegator's principal.

The eligible set consists of records satisfying the self-bond minimum, key-validity and slashability conditions in the designated snapshot. The active set is the highest-weight $N_{\max}$ eligible validators, ordered by weight and then ascending validator identifier. All votes use the original snapshot weights, not a later live balance. Deposits, delegation, exits and penalties do not mutate a committee partway through an undecided height.

For an ordinary activation of epoch $e \geq 2$, the source snapshot is captured at the end of epoch $e - 2$. Epochs 0 and 1 use the explicitly identified bootstrap snapshots from genesis. Thus a new bond does not automatically vote at the next boundary. It first appears in an eligible snapshot, then waits for the activation that actually uses that snapshot.

Eligibility snapshots pin the slashable positions on which their weight depends. A position cannot be withdrawn or redelegated while an active or scheduled snapshot can use its weight. The same position is

assigned to one validator identifier; an update cannot clone it into another validator's voting balance. A new bond creates a new position instead of editing the amount of a previously snapshotted lot.

### 4.6.2 Coverage and extension

The prospective active set must reach $Q_e$ and the configured coverage floor. If it fails those checks, the current committee epoch continues with exactly its existing identifier, weights and quorum. The committed `extension_index` increases. An interval timer is expressed in finalised heights, so it does not advance during a consensus halt.

At a failed boundary, a replacement eligibility snapshot is captured from the then-finalised registry. The next retry occurs one accounting-length interval later and evaluates that frozen replacement snapshot. Another failure captures the next replacement snapshot for the following retry. This rule is explicit and independent of local receipt times. Discarded prospective snapshots cease to pin positions once no active or scheduled set references them.

Transactions can continue during a coverage extension if the current committee retains a reachable quorum. Exit requests are recorded, but positions that still support that committee cannot start their release interval. A request is not silently cancelled or treated as a withdrawal.

Accounting intervals continue every `ACCOUNTING_INTERVAL_HEIGHTS` finalised heights whether or not the committee changes. Scheduled stake-reward distribution therefore does not depend on an arbitrarily extended committee epoch ending.

### 4.6.3 Data handoff without circular commitments

When the prospective set passes the activation checks, the last old-epoch cut commits its descriptor and an armed boundary state. Only after that cut is final can the handoff digest be formed:

$$\begin{aligned} D_{\text{handoff}} = H(&\text{HANDOFF} \parallel \text{chain} \parallel \text{version} \parallel e \parallel (e+1) \parallel h_{\text{last}} \\ &\parallel \text{last\_old\_cut\_id} \parallel \text{queue\_root} \parallel \text{next\_validator\_set\_id} \parallel \text{next\_params\_hash}). \end{aligned} \tag{10}$$

The incoming set signs this digest after verifying the finalised predecessor state and holding the bodies for the entire remaining queue. Its handoff threshold is the incoming set's $Q_{e+1}$. The proof has its own phase domain and explicitly binds both epoch identities, the boundary height and the incoming committee.

The first new-epoch cut commits the semantic transition identifier derived from this digest and carries the valid handoff proof. Proof signatures are not inserted into the last old cut. There is no dependency from the old cut's own hash back into its post-state root.

If the incoming certificate is unavailable, no first new cut is valid and the boundary stops. This differs from a coverage failure detected before the boundary was armed. The old set is not extended after it has committed the successful handover decision. Recovery consists of supplying the required proof and data, not inventing a smaller threshold.

The rule also transfers proposer state: the new priority vector starts at zero, and its first height is verified under the already committed incoming set. A reader verifies the final old certificate, the incoming descriptor, the incoming handoff and the new certificate in that order.

### 4.6.4 Withdrawal lifecycle

The release epoch is determined from committed snapshot exposure, not from the day the request was submitted. Extension of an epoch can therefore extend the real waiting time. Wallets report both the

**Table 3.** Principal lifecycle. A ledger epoch is not a guaranteed calendar day.

| State | Transition rule |
|---|---|
| Bonded, pending | Funds have left the UTXO set and belong to one slashable position. Snapshot eligibility follows the registry rule. |
| Snapshot-eligible | Weight is pinned by the snapshot. Only its selected active validator may sign with that weight. |
| Exit requested | The owner records a nonce-bound request. No direct redelegation or withdrawal bypasses existing exposure. |
| No remaining exposure | After every snapshot that can use the position retires or is discarded, principal moves atomically from bonded to unbonding. |
| Unbonding | Seven completed committee epochs must elapse after the release epoch. Evidence remains admissible for the whole interval. |
| Withdrawable | An owner-authorised transaction consumes the unbonding claim exactly once and creates its output. It cannot also remain in bonded or unbonding value. |

protocol counter and a nominal estimate rather than promising a date that the consensus rules do not enforce.

Owner-controlled key rotation does not erase old votes or shorten liability. Each replacement consensus key requires proof of possession and ordinary snapshot activation. A previously retired or jailed consensus identity is not silently reused. The protocol preserves historical set descriptors throughout the evidence and checkpoint-validation horizon.

### 4.6.5 Slashing and evidence scope

The v1 evidence rules cover double prevotes, double precommits and proposer equivocation for different values at the same height, view, phase and committee identity. Evidence contains the two complete authenticated messages. A stable evidence identifier is derived from the ordered semantic messages, not from randomised signature bytes.

A valid offence jails the validator and burns the slashable principal still assigned to that validator's penalty generation, including pending and unbonding delegated principal exposed to the same penalty. New delegations to a jailed generation are invalid. The full-penalty policy is explicit: delegators assume the validator's signing risk. Accrued reward liabilities are distinct from principal and are not burned unless a separate activated rule authorises that movement.

Implementations use a committed per-validator generation status and aggregate principal balances so that a full penalty does not require rewriting an unbounded number of position records in one cut. A position in a slashed generation has zero effective principal. Audit recomputation must use that same definition, not sum its obsolete face amount. Evidence is consumed once; a repeated proof cannot burn already burned principal again.

Current-epoch certificate verification continues to use the frozen snapshot weights. Slashing changes future eligibility and value accounting; it does not retrospectively change old certificates or let two nodes reinterpret the quorum of a height differently.

These offence rules are not asserted to establish complete accountable safety across all views. Different non-NIL precommits in different views are not automatically fraud: a justified lock transition may permit them. Adding a broader slashing theorem requires a corresponding evidence algorithm, rather than importing an unrelated surround-vote condition.

## 4.7 Transaction execution and native-asset safety

Transactions are processed in block order from Equation (5) and in each block's committed list order. All amounts are unsigned base-unit integers. The canonical payment envelope, state-domain operations and execution-result codes are specified in the appendices.

A transaction may occur in more than one work block. Its first effective occurrence consumes its inputs and collects its fee; later copies do not execute a second time. Conflicting spends are resolved in the same way: a transaction whose input has already been consumed has no effect. The entire transaction is skipped, not a valid subset of its outputs or inputs.

Structural invalidity and stateful nonexecution are different. A malformed encoding, invalid authorisation, repeated input within one transaction, negative or overflowing amount, unknown operation or forged anchor input makes the containing block ineligible for registration. A transaction that was properly authorised against its anchor may nevertheless fail later because an input was spent, its execution-height range ended, or an owner nonce changed. Such a transaction produces a defined no-effect result, not a partial write.

### 4.7.1 Anchor-state checks and input dependencies

Every input must exist in the finalised state named by the containing block's work anchor. A transaction cannot spend an output created by another still-unfinalised transaction, even in the same work block. This excludes in-cut parent–child dependencies in v1. Wallets rebuild or rebroadcast children only after the parent output is final and a sufficiently recent work anchor includes it.

At registration, nodes resolve the historical input records and verify authorisations against them. They retain the required anchor state versions or equivalent verified input provenance while registration validation can still require them. After registration, those validated facts and the full block remain available for execution; a node that has already verified them need not retain unrelated historical UTXOs indefinitely. Snapshot import declares the checkpoint trust boundary rather than pretending that missing historical authorisations were rechecked.

At execution, the inputs must still be present in the current state, with the same committed amounts and locks. A signature-verification cache is keyed by the exact witness and signable context. It never caches a permanent claim that an outpoint is spendable.

### 4.7.2 Atomic application

For a payment, inputs must be distinct and the complete input sum must cover the output sum. Their difference is the fee. Native staking operations add their explicitly authorised principal movement to this balance equation. A user transaction cannot create a reward output by choosing a special output identifier or a reserved coinbase type.

The transition first validates all preconditions against a read-only transaction view. It then reserves every input and operation claim, computes all outputs and fees and applies the complete write set atomically. Failure produces no write set. Output identifiers and stake-position identifiers cannot overwrite existing live records; a collision is an error, not a replacement operation.

The following properties are checked independently: valid signatures, unique inputs, unspentness, value balance, operation authority, nonce freshness, output uniqueness and the authorised issuance delta. Total supply equality alone cannot detect an unauthorised mint if an implementation wrongly increases both the destination and its mint counter.

### 4.7.3 Error handling and replay

Each consensus operation has a domain-separated identity and a one-use state transition. Owner operations bind the target object and expected nonce. Network identity and protocol version are part of every signature context. A withdrawal, reward payment, bucket close or offence proof cannot be replayed into a second value movement.

A skipped transaction pays no fee because none of its inputs were consumed. Its result code is a deterministic receipt for explorers and wallets, not an instruction to remove it from every participant's mempool. Resubmission policy is local; the next attempted execution must still satisfy the complete state predicate.

## 4.8 Deferred work rewards

A work block's subsidy share is fixed when its anchor's registration window closes. Its transaction execution normally occurs earlier. These events are independent and may occur in either order; a linear "close, then execute" lifecycle is incorrect.

For a registered block $b$, the transition maintains its execution status, its immutable closed-bucket subsidy claim, its effective work-fee claim and a payment marker. Define

$$\mathrm{ready}(b) = \mathrm{executed}(b) \wedge \mathrm{closed}(a(b)) \wedge \neg\,\mathrm{paid}(b). \tag{11}$$

The first possible readiness height is

$$h_{\mathrm{ready}}(b) = \max\{h_{\mathrm{exec}}(b),\ a(b) + W_{\mathrm{REG}}(a(b))\}. \tag{12}$$

At anchor closure, the budget is minted into the work reserve and claims are fixed. At execution, effective transaction fees are credited to the appropriate block claim and the execution marker is set. Both events invoke the same readiness check. A block enters the payment queue only once.

Payment uses a deterministic queue of ready claims, ordered by readiness height and then block identifier. A fixed maximum number of entries is processed per cut. This bounds the work from a large bucket closure; readiness is the earliest eligible height, not a promise of same-cut materialisation during a settlement burst.

```
TryEnqueuePayment(b):
    if executed[b] and bucket_closed[anchor[b]] and not paid[b]:
        if not payment_queued[b]:
            enqueue(ready_height[b], block_id[b])
            payment_queued[b] = true

PayReadyPrefix():
    for b in required_prefix(payment_queue, MAX_WORK_PAYMENTS):
        amount = subsidy_due[b] + effective_work_fee_due[b]
        assert work_reserve >= amount
        assert not paid[b]
        if amount > 0:
            assert reward_outpoint(b) is absent
            create_utxo(reward_outpoint(b), amount, reward_lock[b], height)
        work_reserve -= amount
        paid[b] = true
        remove_payment_entry(b)
```

The reward outpoint is

$$\big(H(\mathrm{WORK\_REWARD} \,\|\, \mathrm{chain} \,\|\, \mathrm{block\_id}(b)),\ 0\big). \tag{13}$$

It binds the full block identifier, including nonce and extranonce through the header commitment. It does not depend on the admission cut or the payment height. The UTXO's creation height is the actual payment height. A zero claim is marked settled without producing a zero-valued output.

The debit, output creation, marker and queue removal are one atomic state change. A stored marker is retained until no admissibility, execution or settlement path can encounter that block again. Pruning requires completed bucket and cursor state; deleting a live liability because the registration window ended is invalid.

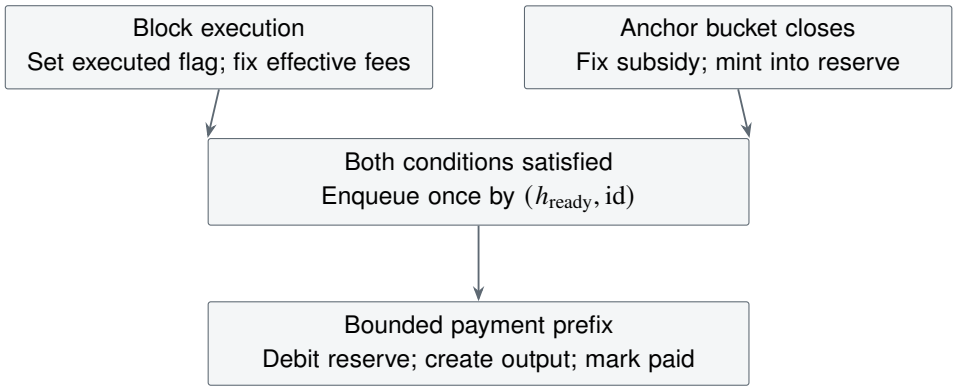


**Figure 5.** Correct reward lifecycle. Execution and bucket closure can occur in either order. Neither event alone pays the subsidy. The payment cursor affects timing, not the fixed claim amount.

For example, a block anchored at 100, registered at 101 and executed at 102 becomes subsidy-ready at 196 when $W_{\text{REG}} = 96$. No second execution at 196 is necessary or permitted. Conversely, a block still queued at 196 becomes ready only when it eventually executes. Its transaction-finality time and its reward-payment time are different observables.

## 4.9 Bounded system work

Registration and execution caps do not cover every source of computation. Bucket closure, ready-claim insertion, reward outputs, stake distribution, evidence handling, snapshot selection and database commitments must also have deterministic bounds.

One anchor can collect at most $W_{\text{REG}}$ MAX_REGISTRATION_BLOCKS registrations under an unchanged profile: 9,216 under the launch values. Its closure can therefore enumerate at most that many block identifiers. The closure descriptor records the sorted membership, budget quotient and remainder; claim creation and payment follow bounded cursors. No cut may perform an unbounded recursive scan of the whole work graph.

Stake-reward distribution similarly uses a frozen snapshot and a persistent cursor. The reference profile processes at most 512 beneficiary records and 1,024 ready work claims per cut. Registry and position admission limits are fixed in the manifest. In particular, the maximum reward-entry population must fit within one accounting interval's deterministic cursor service if the deployment claims that every completed pool clears before the next.

Each primitive state operation has a published logical mass charge, with separate ceilings for signature checks, input and output records, staking updates and evidence proofs. The mass coefficients are manifest data; implementations do not infer them from a local benchmark. Every admissible transaction fits within an individual block and every admissible block fits within an execution prefix.

The complete transition order in Section B.1 reserves system service before any implementation considers a local scheduling optimisation. Multithreading is allowed only when it preserves the same logical

writes, intermediate conflict decisions, counters and final root. An out-of-memory or disk-full node stops applying the cut; it does not publish a shorter transition as valid.

# 5 Analysis

## 5.1 Deterministic execution

**Theorem 5.1** (Determinism)**.** *Let two correct nodes agree on the finalised state at height $h-1$, the active protocol manifest and the genesis identity. If they validate the same cut at height $h$ with the required data, they derive the same post-state commitment, independently of their block-receive order.*

*Proof.* The prior state fixes the queue, its resource counters, the committee, the parameter version, pending system operations and all value-domain records. The longest-prefix rule fixes the execution set. Its canonical commitment and the predecessor identifier fix the execution identifier, which fixes the block order. Each block fixes its transaction list order. Authorisation, current-input availability, operation nonces, integer balances and failure codes are deterministic predicates of these inputs.

The registration batch is part of the cut value, so both nodes append the same canonically ordered entries after checking the same anchor-bound admissibility rules. Anchor closure, issuance, difficulty, reward cursors, committee transitions and proposer-priority advancement follow the fixed transition order in Section B.1. The same arithmetic and canonical encodings therefore produce identical leaves, aggregate counters and roots in every committed domain. Local receipt time and local resource utilisation are not inputs. Induction from the same genesis establishes the result for the certified history. □

This theorem is an implication about a specified transition function. It does not show that independently written clients already implement that function correctly. Byte-level conformance vectors, differential execution and cross-platform tests establish the implementation correspondence. A node lacking data waits for it; it does not derive a shorter execution prefix from local availability.

## 5.2 Agreement across views

**Lemma 5.2** (Weighted quorum intersection)**.** *For two signer sets $Q_1, Q_2 \subseteq K_e$ each carrying at least $Q_e$ weight,*

$$w(Q_1 \cap Q_2) \geq w(Q_1) + w(Q_2) - w(K_e) \geq 2Q_e - W_e > W_e/3. \tag{14}$$

*Thus, if Byzantine weight is below $W_e/3$, their intersection includes correct voting weight.*

*Proof.* Inclusion–exclusion gives the first inequality. Since $w(K_e) \leq W_e$ and $Q_e = \lfloor 2W_e/3 \rfloor + 1$, the second inequality and the strict final bound follow. Byzantine signers alone cannot occupy the entire intersection. □

**Theorem 5.3** (Conditional agreement)**.** *At a height with a fixed eligible snapshot and active committee, Byzantine weight below $W_e/3$, unforgeable vote authentication and correct execution of the locking and durable-signing rules, two distinct cut values cannot both obtain valid commit certificates. The property extends across committee boundaries through the certified handoff rule.*

*Proof.* Within one view, conflicting precommit quorums intersect in correct weight by Theorem 5.2, contradicting the one-precommit rule. For different views, suppose $X$ commits in view $v$. Its correct precommit signers carry more than $W_e/3$ weight and are locked on $X$ at $v$. A conflicting prevote quorum

in a later view must include some of that weight. Choose the earliest later view containing such a quorum. A correct member of the locked weight can prevote a conflicting value only with admissible quorum evidence at least as recent as its lock and strictly earlier than the current view. No conflicting quorum can exist at $v$ by same-view intersection, or between $v$ and the chosen view by minimality. The required evidence is therefore unavailable. This is the inductive lock argument of the pinned algorithm [8].

At a committee change, the last old cut fixes the successor descriptor. The incoming quorum acknowledges that exact final predecessor and its queue; the first new cut binds the acknowledgement and the successor parameters. The successor set is not locally selected. Induction over these transitions preserves the common history while the fault assumption holds for each relevant snapshot. □

The theorem depends on the complete event guards, not just on quorum arithmetic. A signer that releases a second vote after restoring an obsolete log violates an assumption. An implementation that switches committee weights in the middle of a height changes the protocol. A correct node that observes two conflicting valid certificates for one height enters a safety-violation halt and preserves both proofs; it does not silently switch to whichever certificate it received last.

This is deterministic finality under a fault model, not immunity to every coalition. If the fault bound is exceeded, partitioned observers can be shown different, individually valid histories before either observer sees the conflict. An implementation must make this distinction visible in its security documentation.

## 5.3 State validity and value conservation

**Proposition 5.4** (Certificate authority)**.** *A certificate does not authorise a transition outside the protocol rules. A fully validating node rejects a cut whose registration, execution or system operations fail those rules, regardless of the number of signatures attached to it.*

*Proof.* The verifier recomputes the complete transition from its previous accepted state. It separately checks authentication, permitted value movement, the authorised mint delta and the resulting root. A proposed invalid transition either fails a predicate or does not reproduce that root. Signature count does not alter any of those checks. □

The scope is local validity. A coalition outside the agreement assumption may still create conflicting transition-valid histories, censor registrations or stop progress. Likewise, a node that imports a trusted checkpoint does not retrospectively establish the authorisation of every earlier spend. After that starting point it resumes full validation. Neither qualification is removed by increasing the hash width or by checking only the supply total.

Coin-duplication resistance is established at the operation level. The one-use input rule excludes duplicate consumption; reward and withdrawal identities exclude repeated settlement; atomic source and destination updates exclude partial transfers; and authorised-issuance checks exclude a forged mint. Section 7 supplies the accounting invariant that ties these transitions together. Fault injection and differential testing must verify that the code preserves each of these properties.

## 5.4 Progress, partitions and registration

After stabilisation, agreement progresses when correct active validators carrying at least $Q_e$ weight can exchange valid proposals, data and votes within the growing timeout budgets. The reference proposer policy must eventually select a correct, reachable proposer. Persistent loss of data, inadequate processing

capacity or insufficient active correct weight is outside that progress condition even if the network's links have stabilised.

Consensus progress and admission progress are different. An honest proposer carrying a previously valid cut must re-propose that value, including its original registration batch. It cannot insert newly arrived work into the value while claiming the old quorum evidence. Consequently, the appearance of an honest proposer alone does not prove that a particular unregistered block is immediately included. A bounded registration-delay guarantee additionally needs a fair admission opportunity, successful delivery, admissible load and sufficient capacity before the anchor deadline.

A global eligible-stake bound cannot be substituted for the coalition's active-stake fraction. If active coverage is $S/W_e = 0.8$ and Byzantine stake is below $W_e/3$, the active fraction may approach

$$\frac{W_A}{S} < \frac{1/3}{0.8} = \frac{5}{12}, \tag{15}$$

not one third. This edition does not rely on an unproved bound on consecutive adversarial proposer slots. The minimum validator bond bounds how many eligible identities a fixed amount of capital can fund; it does not by itself establish a worst-case admission deadline under every distribution and view-change trace.

The registration window is a fixed admissibility rule. Its launch value of 96 heights provides a nominal 240-second horizon at a 2.5-second cadence. It is not a theorem that every honestly produced block is registered in that interval. Under overload, censorship or inadequate connectivity, unregistered work can expire. Once registration is committed, later admission choices cannot remove that block's execution position or its subsidy claim.

### 5.4.1 Behaviour without a quorum

No cut means no change to committed height, stake, difficulty or expiration counters. Miners may continue working against the last finalised anchor, but this does not create a promise that the resulting backlog will fit after recovery. At 25 blocks/s, four hours of halted finalisation produce approximately 360,000 blocks. The block-count limit admits at most $96 \cdot 96 = 9,216$ registrations for one anchor across its complete 96-height window, before any tighter byte or mass limit is considered.

The window does not expire while height is frozen. It may expire during the resumed sequence of cuts before the backlog has been admitted. Registered work retains its committed status; unregistered work has no guaranteed claim.

There is no automatic minority-quorum or proof-of-work ordering fallback in v1. The last final state remains unchanged until the required quorum returns. Permanently lost quorum weight can therefore halt the network indefinitely. Any emergency recovery that changes the trust anchor is an explicit external recovery or activated protocol change, not an ordinary valid continuation under a secretly reduced quorum. This boundary prevents two sides of a partition from independently lowering their thresholds and both declaring themselves final.

# 6 Difficulty control

Difficulty controls permissionless work production, not the agreement threshold. The controller observes registered work against immutable finalised anchors. Miner timestamps, local arrival times, execution-cut size and the number of unregistered blocks in a node's memory are not measurements in this controller.

An anchor bucket $A_a$ contains the blocks registered against anchor $a$ within its committed window. Its count $N_a = |A_a|$ is final only after height $a + W_{\mathrm{REG}}(a)$ has processed registrations. A late registration inside the last eligible cut must therefore be counted before that cut closes the bucket. The closure event runs once.

## 6.1 Closed-bucket feedback

For the fixed v1 profile, the latest closed anchor at height $h$ is $L_h = h - W_{\mathrm{REG}}$. The target count is $k = k_{\mathrm{num}}/k_{\mathrm{den}} = 125/2$. With the controller origin $a_0$, the unfiltered cumulative error is

$$\Delta_h = \sum_{a=a_0}^{L_h} (k - N_a), \tag{16}$$

where an empty sum is zero. A block surplus makes $\Delta_h$ negative. Since a larger target makes mining easier, the negative sign must lower the target:

$$T_h = \min\Big(T_{\max}, \max\Big(1, \big\lfloor T_0\, 2^{\Delta_h/\lambda} \big\rfloor\Big)\Big). \tag{17}$$

Here $\lambda = 22,500$ blocks is the reference half-life parameter and $T_0$ is the manifest's initial target. The exponential form follows the ASERT construction [12]; replacing its measured error creates a different feedback loop, whose stability cannot be inherited without analysis.

## 6.2 A deterministic capacity filter

A shortfall caused by saturated registration capacity must not accumulate as a deferred instruction to make mining easier. For a closed anchor, let $c_a$ indicate that at least one certified cut in its registration window had insufficient remaining admission headroom for a maximum admissible block. The per-anchor contribution is

$$d_a = k - N_a, \qquad \widehat{d}_a = \begin{cases} \min(d_a, 0), & c_a = 1, \\ d_a, & c_a = 0, \end{cases} \qquad \Delta_h = \sum_{a=a_0}^{L_h} \widehat{d}_a. \tag{18}$$

A suppressed positive contribution is discarded, not postponed. A surplus still tightens the target.

The headroom predicate is computed only from committed counters. After the cut's execution and registration updates, evaluate the remaining registration count, byte, mass and authorisation budgets, and the remaining queue count and byte budgets. The predicate is true if adding one block at any protocol maximum would exceed a corresponding limit. Exact exhaustion is included. The maximum-block authorisation bound is explicit in the manifest; it is not inferred from a local mempool. Section C gives the Boolean expression.

This is a conservative, reproducible saturation signal. It does not prove that a real candidate was excluded, and it does not claim to detect every bottleneck. A proposer may leave capacity unused or affect which cuts appear saturated. Selective registration and strategic block construction therefore remain inputs to the adversarial controller evaluation. The filter prevents a specific backlog-induced easing mechanism; it is not a proof of manipulation resistance.

A rolling committed history of cut flags determines $c_a$ for each closing bucket. No node may decide that a bucket was limited because its own machine was overloaded. On a parameter change, the old anchor's original window and resource profile remain associated with its measurement segment; overlapping profiles require an explicit migration rather than a revised interpretation of old data.

### 6.3 Integer evaluation

Consensus uses a rational error numerator, not floating point. Let

$$Z_h = \sum_{a=a_0}^{L_h} \widehat{z}_a, \qquad z_a = k_{\text{num}} - k_{\text{den}} N_a, \tag{19}$$

with the same capacity filter applied to $z_a$. Then $\Delta_h = Z_h / k_{\text{den}}$. For $R = 2^{16}$ define

$$e = \left\lfloor \frac{Z_h R}{k_{\text{den}} \lambda} \right\rfloor, \quad \nu = \lfloor e/R \rfloor, \quad \phi = e - \nu R, \qquad 0 \le \phi < R. \tag{20}$$

The fixed-point polynomial is

$$F(\phi) = R + \big( (c_1 \phi + c_2 \phi^2 + c_3 \phi^3 + 2^{47}) \gg 48 \big), \tag{21}$$

where $c_1 = 195766423245049$, $c_2 = 971821376$ and $c_3 = 5127$. The executable rule evaluates $T_0 F(\phi) 2^{\nu - 16}$ with integer shifts, then clamps to $[1, T_{\max}]$.

All division involving negative values is floor division. The polynomial and target multiplication use unsigned 512-bit intermediates; signed error accumulation uses a range checked before multiplication. A left shift is not performed if its result is already known to exceed $T_{\max}$. A sufficiently large right shift returns zero before the final clamp to one. Merely clamping after an overflow is invalid. The exact bounded algorithm and edge cases appear in Section C.

Before the first bucket closes, $Z_h = 0$ and $T_h = T_0$. Genesis carries a valid positive $T_0 \le T_{\max} \le 2^{256} - 1$. Each block uses the target committed by its own anchor; recomputing its work against a newer target is incorrect. Origin, accumulator and active controller segment are committed state.

### 6.4 Delayed local stability

*Analytical model.* Consider a continuous small-deviation approximation with constant hash rate and cut cadence, mining on the newest anchor, no admission cap and no strategic withholding. Let $x_h$ be the deviation of the logarithmic target from equilibrium and $W = W_{\text{REG}}$. The bucket-closing delay gives the local recurrence

$$x_h = x_{h-1} - g x_{h-W}, \qquad g = \frac{k \ln 2}{\lambda}. \tag{22}$$

Its characteristic polynomial is

$$z^W - z^{W-1} + g = 0. \tag{23}$$

For this scalar delayed linear model, the first unit-circle crossing occurs at

$$0 < g < 2 \sin\!\left( \frac{\pi}{4W - 2} \right). \tag{24}$$

To see the boundary, substitute $z = e^{i\theta}$ into $z^{W-1}(z - 1) = -g$. The first positive crossing has $\theta = \pi/(2W - 1)$ and magnitude $g = 2\sin(\theta/2)$. The root at one moves inside for small positive $g$; the remaining roots start at zero. This identifies the first loss of stability of this model, not of every admissible network trace.

At $W = 96$, the boundary is approximately 0.01644794. The launch gain is approximately 0.00192541, below that boundary. The no-delay contraction $1 - g \approx 0.99807459$ is consistent but omits the principal delay. Including the delayed recurrence is therefore more informative than quoting the no-delay factor alone.

The calculation does not establish the full controller's adversarial or nonlinear stability. The capacity filter, mining against older anchors, changing cadence, Poisson variation and withheld work alter the loop. Reproducible simulations must exercise these cases and state the tested parameter ranges. Timestamp manipulation is absent from the measurement input; registration manipulation and resource pressure are not thereby absent from the system.

# 7 Emission and value accounting

Subsidy is scheduled per finalised work anchor, rather than minted independently for every parallel block. The number of registered blocks determines how an anchor's work budget is divided, not how much aggregate subsidy that anchor authorises. Transaction fees are transfers of existing value and never increase minted supply.

## 7.1 Monetary manifest and the work–stake split

Let $\sigma_h$ be the integer subsidy scheduled for anchor $h$ by the activated `EmissionV1` manifest. The manifest also fixes the base-unit denomination, genesis allocations and maximum supply. No numerical curve or supply cap is inferred from the target block rate. The file format and consistency conditions in Section D define this network-specific interface.

With work share $\omega = 5000$ basis points and a carried rounding numerator $\epsilon_{h-1}$,

$$W_h = \left\lfloor \frac{\sigma_h \omega + \epsilon_{h-1}}{10^4} \right\rfloor, \qquad \epsilon_h = (\sigma_h \omega + \epsilon_{h-1}) \bmod 10^4, \tag{25}$$
$$S_h = \sigma_h - W_h.$$

Thus $W_h + S_h = \sigma_h$ at every height. $\epsilon$ is a rounding numerator, not a coin reserve. The stake share is obtained by subtraction, not by an independent rounding operation.

At the anchor's creation, $S_h$ is minted into the stake reward reserve and associated with the eligible reward snapshot in force at that height. $W_h$ is scheduled but remains unminted until a nonempty closed bucket creates work liabilities. This timing avoids allocating a delayed stake subsidy to whatever committee happens to be active when an older bucket eventually closes.

Genesis subsidy $\sigma_0$ is zero in this encoding profile. Genesis allocations are accounted separately. The first ordinary scheduled subsidy is $\sigma_1$; a block may initially mine against the genesis anchor even though that anchor carries no subsidy. The startup manifest makes this bootstrap convention explicit.

## 7.2 Closed-anchor allocation

For the immutable set $A_a$ at closure, let $C$ be the accumulated unminted work budget from preceding empty buckets. If $N_a = 0$, update $C \leftarrow C + W_a$ and create no work liability. Otherwise set

$$P_a = W_a + C, \quad C \leftarrow 0, \quad q_a = \lfloor P_a / N_a \rfloor, \quad r_a = P_a \bmod N_a. \tag{26}$$

In ascending block-ID order, the first $r_a$ blocks receive $q_a + 1$ base units and the remaining blocks receive $q_a$. The sum is exactly $P_a$. The remainder $r_a$ is fully distributed and is *not* carried into another bucket.

The closure mints $P_a$ into $R_{\text{work}}$. The closed bucket commits its sorted membership, $P_a$, $q_a$ and $r_a$; an individual claim is derived from its index in that immutable list. Already executed entries become payment-ready at closure. Entries still queued become ready when execution later supplies their final fee claim.

The same anchor implies the same target. Equal subsidy division is therefore equal payment per accepted proof at that target; it does not measure the number of failed hashes an individual miner actually performed. Registration timing within the window does not alter a registered block's subsidy allocation once the bucket is closed. This is not a general proof that withholding is unprofitable: transaction fees, admission risk and future inclusion effects remain distinct incentives.

A subsidy scheduled for an empty work bucket remains unminted in `work_carry`. The stake half can still be minted and distributed. An empty execution cut therefore does not imply zero issuance. `work_carry`, the basis-point residue and live reward reserves have different meanings and are never added together as if all represented existing coins.

## 7.3 Fees and exact recipients

For each successfully executed transaction with fee $F$, apply the same split as Equation (25), using the separate committed `fee_split_carry` numerator. The work share is credited to the block containing its first effective occurrence; the stake share is credited to the accounting pool associated with that execution height and its eligible reward snapshot. A duplicate or failed occurrence creates no fee transfer.

The input debit and the creation of the fee liabilities are part of the same transaction write set. Work fees increase $R_{\text{work}}$ and the receiving block's fee claim, not $I_{\text{minted}}$. The fee recipient cannot be selected after the fact by the certificate assembler. Work subsidy and work fees are materialised together by the one-use payout in Section 4.8.

## 7.4 Stake snapshots and bounded distribution

Committee epochs govern voting membership; accounting intervals govern periodic reward settlement. The interval length is the manifest's `ACCOUNTING_INTERVAL_HEIGHTS`, equal to 34,560 in the launch profile. Intervals advance with finalised height even when the same committee epoch is extended. Reward accounting therefore need not wait indefinitely for an eligible successor committee.

An accounting pool is identified by its interval and the immutable eligible snapshot to which its income was attributed. If a boundary arrangement causes more than one snapshot to receive income during one interval, separate subpools are closed. Each reward weight is principal in that historical snapshot, not a later mutable stake balance. Pending stake absent from the snapshot receives no share for that snapshot. Eligible non-active stake remains included, as in the source design; certificate inclusion is not a reward criterion.

Let the snapshot's validators be ordered by ID, with weights $w_i$, cumulative weights $C_i = \sum_{j \le i} w_j$ and total $W > 0$. A pool of $P$ base units assigns

$$p_i = \left\lfloor \frac{PC_i}{W} \right\rfloor - \left\lfloor \frac{PC_{i-1}}{W} \right\rfloor, \qquad C_0 = 0. \tag{27}$$

The sum telescopes to $P$. Products use unsigned 256-bit or wider intermediates under the manifest's amount bound; a 128-bit multiply followed by division is not sufficient merely because the final share fits in 128 bits.

Within each validator's share, the same cumulative-floor rule distributes to its snapshot stake positions in ascending position-ID order. Each position specifies an owner-controlled reward lock. This edition

uses direct, non-custodial position payouts; it does not silently pay all delegated rewards to the validator operator. There is no implicit operator commission in this profile. Any commission extension requires an explicit, versioned formula and owner consent rather than an off-chain deduction disguised as consensus.

The nested allocation is part of the specified rounding rule. It can differ by a few base units from a single flat allocation over every position; clients must not substitute the latter. Historical reward weights remain fixed even if a position later exits or is slashed. Slashing principal does not retrospectively rewrite already accrued reward shares.

On interval closure, the pool amount and its snapshot root become immutable. A committed cursor traverses the ordered beneficiaries. Each cut processes at most `MAX_STAKE_PAYOUT_RECORDS`=512 records in the reference profile, including zero-share records. The oldest outstanding pool is processed first, with ties by snapshot ID. Each positive share creates one reward outpoint derived from the interval, snapshot and position; zero shares advance the cursor without creating an output. Reserve debits, outputs and cursor advancement are atomic. The original reward lock is retained until payment.

A manifest sets finite registry and position bounds. To claim that routine distribution clears within one accounting interval, its maximum number of payable beneficiary records per interval must not exceed the demonstrated processing capacity. Otherwise the same bounded cursor remains safe but accumulated liabilities and payout latency can grow. Startup validation and the capacity programme explicitly check this condition; an unbounded registry is not hidden behind a 512-record loop.

## 7.5 Value domains and authorised issuance

The native asset occupies disjoint domains:

$$\begin{aligned} V_{\text{live}} &= V_{\text{utxo}} + V_{\text{bonded}} + V_{\text{unbonding}} + R_{\text{work}} + R_{\text{stake}} + V_{\text{escrow}} + R_{\text{compute}}, \\ V_{\text{live}} + V_{\text{burned}} &= I_{\text{minted}} \le I_{\max}. \end{aligned} \tag{28}$$

The compute terms are zero in v1. Pending and active stake positions are disjoint subsets of bonded principal. A position moved to unbonding ceases to contribute to bonded value in the same write. A slashed generation contributes zero effective principal after its aggregate has been burned. Outstanding reward liabilities reside in reserves until materialised; they are not simultaneously counted as UTXOs.

In addition to Equation (28), each cut checks

$$I_{\text{minted},h} - I_{\text{minted},h-1} = S_h + \sum_{a \in C_h,\, N_a > 0} P_a, \tag{29}$$

where $C_h$ contains precisely the anchor buckets newly closed by that cut. The normal fixed-window profile closes at most one bucket per height. An upgrade cannot batch an unbounded number of deferred closures without its own deterministic budget and migration rule.

The genesis allocation is the initial minted balance, not a transaction at an ordinary height. If $G$ denotes that allocation, the manifest validates

$$G + \sum_{h \ge 1} \sigma_h \le I_{\max}. \tag{30}$$

All scheduled work budgets are eventually either unminted carry or once-minted liabilities. No bucket, accounting interval or payout operation can independently increase the permitted total. Equation (29) is essential: a bug that credits a reserve and the mint counter by the same unauthorised amount could preserve the equality in Equation (28) while violating monetary policy.

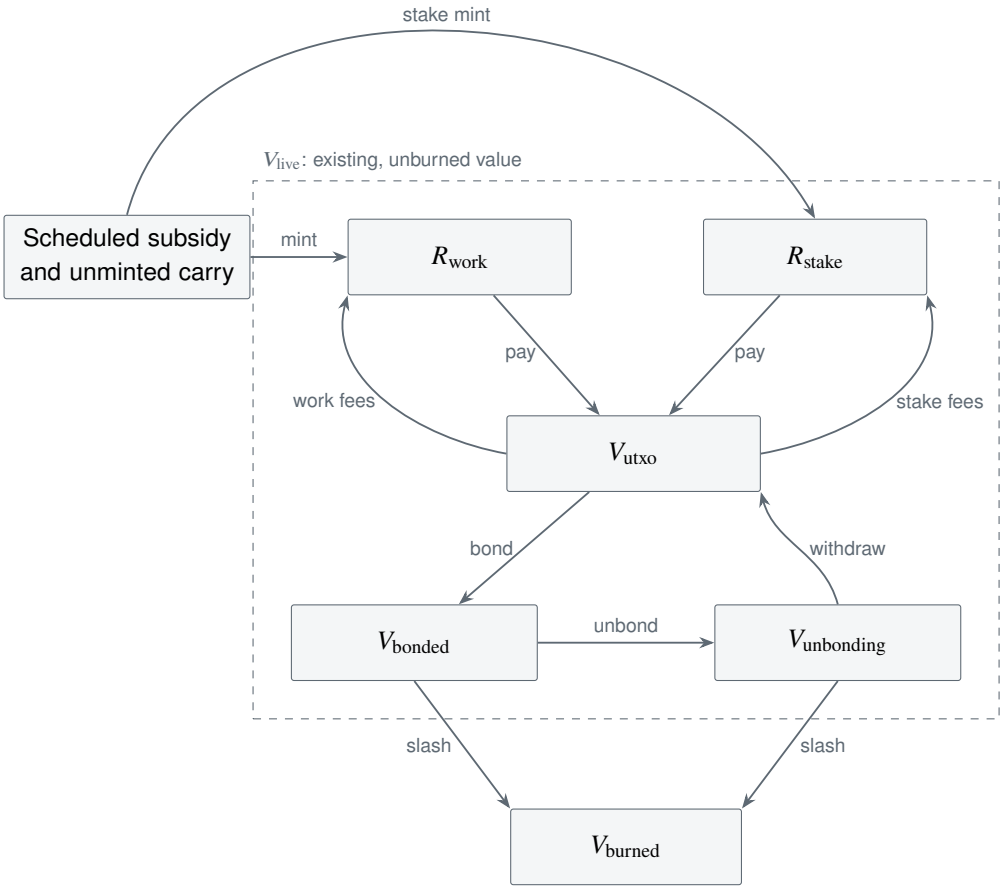


**Figure 6.** Principal native-value paths. Scheduled but unminted work is outside $V_{\text{live}}$. Fees enter reward reserves, not the burn domain. Minting raises $I_{\text{minted}}$; other arrows transfer existing value or burn slashable principal. The disabled compute domains are omitted from the drawing and remain zero in the state equation.

### 7.6 Conservation by transition

The required atomic value movements and their one-use guards are summarised in Table 4.

| Transition | Required atomic movement | One-use guard |
|---|---|---|
| Payment | Consume distinct inputs; create outputs; credit only the effective fee shares. | Input set and transaction identity. |
| Bond / delegate | Remove native UTXO value; create one owned principal position. | New position ID, owner nonce. |
| Unbond | Decrease effective bonded principal; increase the identified unbonding claim. | Position status and snapshot pins. |
| Withdraw | Consume the matured claim; return its authorised value through the transaction. | Claim ID and expected nonce. |
| Bucket close | Mint only its authorised work budget; commit immutable allocation. | Anchor closure marker. |
| Reward pay | Debit exactly one reserve; create a unique output or advance a zero claim. | Block marker or pool cursor. |
| Slash | Remove still-slashable principal; add the same amount to burned value. | Validator penalty generation. |
| Prune | Remove only redundant historical material; preserve live claims and committed roots. | Retention conditions; no value movement. |

**Table 4.** Value conservation requires correct individual transitions as well as a correct aggregate invariant.

Domain aggregate counters are authenticated by the state commitment and updated with the leaves they summarise. Full recomputation from leaves is required in conformance and snapshot-validation tests. Cache consistency, database atomicity and correct interpretation of slashed generations are tested separately from arithmetic. The protocol specifies exactly one effective balance per position; storage duplication does not create a second balance.

## 8 The cost of post-quantum authorisation

ML-DSA-65 determines a substantial part of the wire and storage budget. Its public key is 1,952 bytes and its signature is 3,309 bytes [14]. These are standardised sizes, not measured throughput. A transaction witness in the reference envelope is

$$s_{\text{witness}} = 1 + 1952 + 3309 = 5262 \text{ bytes}. \tag{31}$$

The leading byte selects the authorisation profile. With the complete payment body in Section A.3, a one-input, two-output payment has a 245-byte body and a four-byte witness-count field:

$$s_{\text{tx}}(1, 2) = 245 + 4 + 5262 = 5511 \text{ bytes}. \tag{32}$$

The byte count follows the complete field layout in the reference encoding. Changing that layout requires a corresponding protocol version and revised resource calculations.

The witness is approximately 95.48% of this payment. Additional inputs add additional witnesses; the table does not represent every transaction type. Wider consensus hashes affect references and commitments, but they do not change the signature size. CPU demand depends on implementation, hardware, batch shape, cache state and the ratio of valid to invalid submissions. It must be measured rather than inferred from the size table.

| Element | Ed25519 comparator | ML-DSA-65 | Ratio |
|---|---|---|---|
| Public key | 32 B | 1,952 B | 61.00 |
| Signature | 64 B | 3,309 B | 51.70 |
| One-input witness | 97 B | 5,262 B | 54.25 |
| Payment, same 245-byte body | 346 B | 5,511 B | 15.93 |

**Table 5.** Exact byte accounting for the reference envelope. The classical column is a size comparator, not an enabled v1 authorisation mode. Ed25519 lengths follow RFC 8032 [24]; ML-DSA lengths follow FIPS 204. CPU timing is deliberately excluded from a byte-size table.

## 8.1 Quorum certificate size

FIPS 204 does not define a signature-aggregation operation. TRAIDAG v1 uses explicit signatures and does not assume a deployable aggregate substitute. This is a choice of the cryptographic profile, not a claim that research into lattice aggregation is impossible.

The certificate encoding in Section A.5 has a 167-byte common header and, for each signer, a four-byte index plus one signature. With $n_h$ signers,

$$s_{\mathrm{cert}}(n_h) = 167 + n_h(4 + 3309), \qquad 1 \le n_h \le N_{\max}. \tag{33}$$

For 64 signatures the result is 212,199 bytes, or 207.23 KiB, before transport framing. The signature material alone is 211,776 bytes. Confusing those two numbers would produce a certificate cap too small for its own framing. At 2.5 seconds per cut, the full certificate contributes approximately 0.08488 MB/s to a node fetching each certificate once.

A certificate can contain every active signature, even when fewer suffice to cross the weighted threshold. Capacity is therefore budgeted at $N_{\max}$, not at two thirds of the number of entries. The number of required signers depends on their weights; 64 registry entries do not imply 64 independent operators.

For a certificate-bandwidth budget $\beta$ in bytes per second,

$$N_{\max} \le \left\lfloor \frac{\beta\rho - 167}{3313} \right\rfloor, \tag{34}$$

before additional transport allowance. A 1,024-signature certificate under the same illustrative format is approximately 3.236 MiB per cut, or 1.357 MB/s at the same cadence. Such a committee is constructible but outside the launch budget. Vote dissemination and operational overhead also increase; the single certificate formula is not a complete cost model for a larger committee.

## 8.2 Relay, verification caching and retention

Nodes announce object identifiers and fetch bodies they do not hold. Compact reconstruction avoids retransmitting the complete witness when a known transaction appears in another block; BIP 152 illustrates this general relay approach [13]. TRAIDAG's full-ID reference format does not inherit Bitcoin's short-ID encoding or all of its security assumptions. Missing and ambiguous references trigger bounded explicit fetches.

Signature-verification caches key on the witness identifier and complete authorisation context. A byte-identical witness need not be reverified for every duplicate occurrence, but current unspentness and operation nonces are always re-evaluated. Rate limits apply before expensive parsing and signature work. Network policy may reject or defer peer traffic under load; it may not reinterpret the validity of an already certified cut.

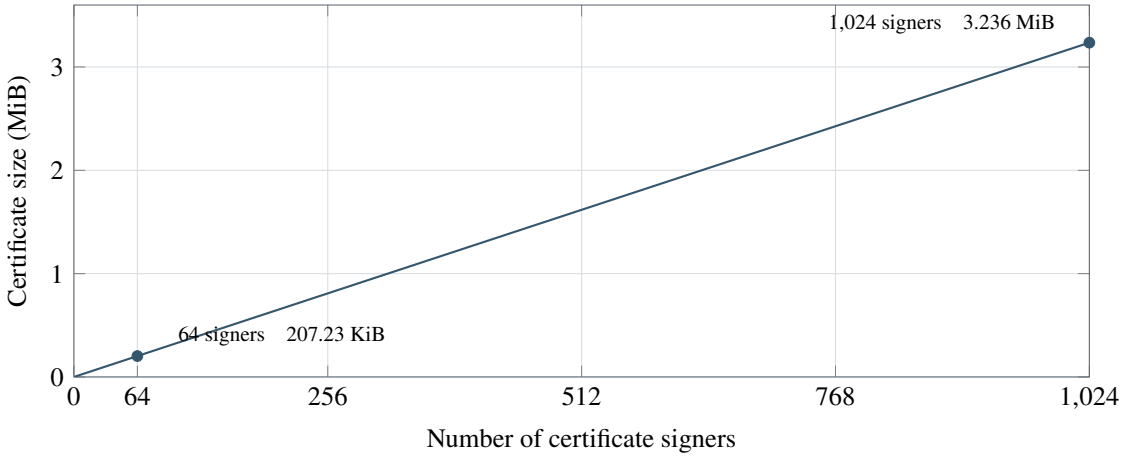


**Figure 7.** Explicit-signature certificate cost from Equation (33). The launch committee size is a resource budget, not a cryptographic impossibility bound. No off-scale aggregation reference is drawn.

A node that has verified a finalised spend may prune its historical witness once no protocol retention duty requires it. This saves local historical storage, not the UTXO state or live reward records. Full historical validation by a new node still requires access to those authorisations. Archive services and historical body availability therefore remain part of the deployment plan. A certificate or a hash-link does not prove the validity of omitted history by itself.

# 9 Performance and worldwide operation

The following calculations describe an operating model. They are not results from an implemented global network. All quantities use bytes unless stated otherwise: one MB is $10^6$ bytes, one MiB is $2^{20}$ bytes and a model month is 30 days. The included calculation script regenerates the tables and numerical curves.

## 9.1 A reproducible ingress model

Let $\tau_{\text{new}}$ be the rate of distinct transaction bodies a node fetches, $R_b$ the work-block rate, $p$ its peer degree, $s_{\text{ann}} = 48$ bytes the identifier payload of an announcement and $\rho$ the cut cadence. A baseline model is

$$\Gamma_{\text{in}} = \tau_{\text{new}} s_{\text{tx}} + \frac{s_{\text{cert}}}{\rho} + p(\tau_{\text{new}} + R_b) s_{\text{ann}} + \Omega. \tag{35}$$

The first term is body delivery, the second certificates, the third identifier announcements and $\Omega$ an explicitly declared additional allowance. Announcement framing, compact-block metadata, retransmissions, transaction misses, requests, transport overhead and peer maintenance are not zero merely because the announcement's identifier is 48 bytes.

For a comparable reference point, take $\tau_{\text{new}} = 100$/s, $R_b = 25$/s, $s_{\text{tx}} = 5511$ bytes, $\rho = 2.5$ s, a worst-case 64-signature certificate and an allowance $\Omega = 0.16$ MB/s. The allowance is a planning assumption retained explicitly; it is not a measured decomposition. A deployment report must replace it with measured components and assess bursts separately.

If $N$ nodes each maintain eight distinct outbound connections, the undirected connection count is approximately $8N$ and mean endpoint degree is approximately 16. Duplicate links, failed dials and heterogeneous roles can change that result. An outbound-only node still sends and serves data across its established bidirectional connections; inbound accept policy does not imply zero egress.

| Peer degree $p$ | Bodies MB/s | Certificates MB/s | Announcements MB/s | Total ingress MB/s |
|---|---|---|---|---|
| 8 | 0.55110 | 0.08488 | 0.04800 | 0.84398 |
| 16 | 0.55110 | 0.08488 | 0.09600 | 0.89198 |
| 32 | 0.55110 | 0.08488 | 0.19200 | 0.98798 |
| 80 | 0.55110 | 0.08488 | 0.48000 | 1.27598 |

**Table 6.** Ingress from Equation (35); every total includes the separate 0.16 MB/s planning allowance. At $p = 16$, ingress alone is approximately 2.312 TB over 30 days. Egress is not included in that number.

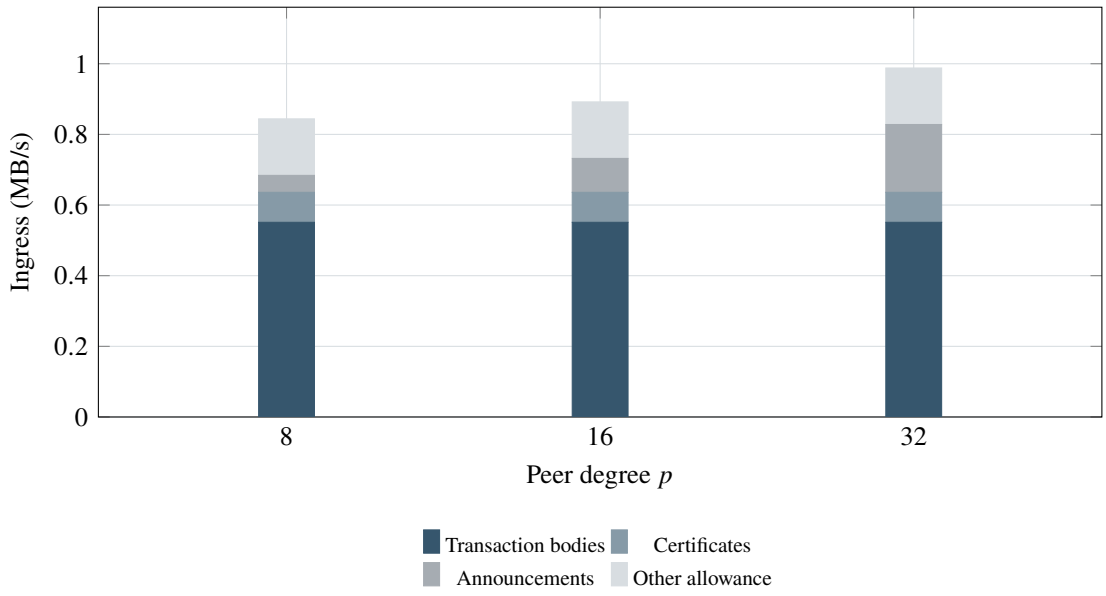


**Figure 8.** The bandwidth components are plotted separately. The allowance is not hidden inside the transaction term, and all axes are linear.

Fetching a single object once per node gives a useful ideal dissemination baseline. Real gossip may retransmit, duplicate requests, lose responses or use redundant paths. A spanning-tree count is therefore a lower-complexity model, not a guarantee about actual traffic. Naively sending every full 5,511-byte body to 32 peers at 100 bodies/s would cost approximately 17.64 MB/s in body traffic alone; this comparison explains the value of announce-then-fetch without claiming an exact real-world saving.

A validator also exchanges consensus votes. In the all-to-all reference pattern, two phases of 63 incoming 3,476-byte votes per cut add approximately 0.17519 MB/s at 2.5 seconds, before retransmissions, proposals and transport framing. The signature-only figure in Figure 4 is a subset of this wire cost. Sentries move public relay load away from the signing service; they do not erase it. Reports distinguish the validator core, the full multi-machine deployment and externally billable traffic, avoiding double counting of internal links.

## 9.2 Duplicate selection and fee incentives

A nominal transaction slot is not necessarily a distinct transaction and a distinct transaction is not necessarily successfully executed. These three quantities must be measured separately. Duplicate IDs and losing conflicts do not create a second spend, but both can reduce useful execution capacity.

For a miner-policy model, fix an integer number $B \geq 1$ of blocks and let $p_t$ be the probability that each particular other block selects transaction $t$. Assume those selections are independent and the miner's copy is exchangeable with the other copies in execution order. Then $X_t \sim \mathrm{Binomial}(B-1, p_t)$ is the number of other copies. Ignoring base-unit rounding in the fee split,

$$\mathbb{E}[Y_t] = \frac{\omega}{10^4} F_t \, \frac{1-(1-p_t)^B}{B p_t}, \tag{36}$$

with the last factor defined as one at $p_t = 0$. The work-share multiplier is required when $F_t$ denotes the full transaction fee. The identity follows from $\mathbb{E}[(1+X_t)^{-1}]$ under the stated binomial model.

The exchangeability assumption is analytical, not a manipulation-freedom guarantee of the ordering function. A proposer may affect inclusion, and miners' selections may be strongly correlated. A random block count also requires conditioning on its integer value and then averaging; $B = 62.5$ is not a valid number of binomial trials.

A reference mining policy can rank candidates by estimated realised work-fee density,

$$S_t = \frac{\omega F_t}{10^4 \, \mathrm{mass}_t} \, \frac{1-(1-\widehat{p}_t)^B}{B \widehat{p}_t}, \tag{37}$$

and sample without replacement using local randomness. The estimate may use recent finalised multiplicities, age and fee rank. No consensus rule depends on this estimator. A miner is free to choose a different valid template; expected economics and protocol validity are different questions.

For an independent-selection baseline, let each of $B$ blocks choose $a$ transactions uniformly without replacement from a common set of $M \geq a$ candidates. Across independent blocks,

$$\mathbb{E}[D] = M\left[1 - \left(1 - \frac{a}{M}\right)^B\right], \qquad \eta = \frac{\mathbb{E}[D]}{aB}. \tag{38}$$

Here $\eta$ is unique-ID slot efficiency. Effective successful throughput requires another measured factor for conflicts and other no-effect results. At $a = 4$, $B = 62$ and candidate depth $M/(aB)$ of 1, 5, 10, 20 and 25, $\eta$ is approximately 0.63511, 0.90767, 0.95236, 0.97580 and 0.98057.

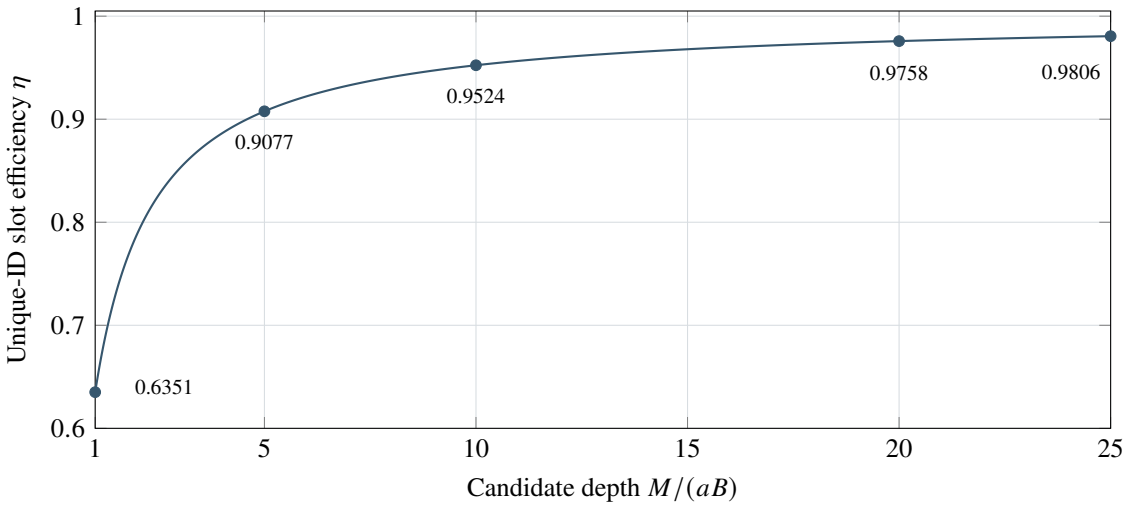


**Figure 9.** Independent-selection model, $a = 4$ and integer $B = 62$. The curve and marked coordinates are generated directly from Equation (38); they are not measured network throughput.

## 9.3 CPU, storage and settlement capacity

A global deployment must sustain the worst admitted cut, not only an average payment mix. The cost profile includes decoding, hash commitments, uncached signature verification, UTXO reads and writes, root updates, registration validation, evidence verification, bucket closure, reward creation and snapshot processing. Logical mass coefficients are consensus parameters. Their conversion to execution time is a hardware measurement.

The 4,096 registration-authorisation cap and 6,144 execution-authorisation cap do not mean that every signature is checked twice; caching can help. They also do not guarantee a cheap cut on a cold node. Maximum-body tests, state-cache misses, malformed traffic and concurrent disk compaction belong in the performance suite. A node may provision more resources but may not reduce the required work according to a local CPU timer.

Reward state is a material growth term. At 25 registered and eventually paid blocks/s, one positive reward output per block creates up to 2,160,000 work-reward outputs per day, or 788,400,000 per 365-day year. These are creation counts, not a prediction that all outputs remain unspent. Miner and pool consolidation reduces the live count only by spending those outputs, which consumes authorisations, bandwidth and execution capacity. Stake payouts add their own outputs.

Storage reporting therefore includes live UTXO growth, block-status records, pending liabilities, retained witness history, header indices, database overhead, snapshots, compaction amplification and temporary free-space requirements. Pruning witnessed transaction history does not eliminate live rewards or supply-accounting obligations. No fixed disk-size recommendation is inferred from the signature fraction alone.

## 9.4 Synchronisation and checkpoints

There are two explicit synchronisation modes. In full-history mode, a node verifies every transition and required authorisation from the agreed genesis, using archive data where needed. In checkpoint mode, it adopts an independently trusted recent state and verifies all subsequent transitions. The latter has an additional trust boundary; it is not a shorter proof of the omitted history.

A checkpoint package binds the chain and genesis identities, finalised height and cut ID, full state root, active committee descriptor, necessary eligibility and pending-activation snapshots, queue and unexecuted bodies, payout cursors, reward liabilities and the live evidence horizon. Snapshot chunks have content commitments and deterministic reconstruction rules. A partial download is not an installable state. Installation is atomic and recomputed roots and aggregates must match the checkpoint.

The v1 checkpoint policy uses a maximum age of six completed committee epochs, strictly less than the seven-epoch principal-liability interval. This is a protocol-counter comparison, not a guaranteed number of calendar days. A joining node must obtain its recent trusted reference through an independently authenticated channel; an attacker-controlled peer's claimed current epoch is not sufficient evidence of freshness. The operator must distinguish historical validation from choosing between histories signed by long-retired keys.

A skip list of checkpoint hashes can make lookup efficient. It cannot alone authenticate omitted validator-set changes or establish the authorisation of earlier spends. Accordingly, this edition does not claim that one month of historical validation can be replaced by thirty certificate blobs of a few megabytes. Certificates, set transitions, bodies and snapshot trust are accounted separately.

For a backlog of $D$ bytes, a sustained processing rate $r$ and continuing arrival rate $a$, an idealised catch-up time is

$$t_{\text{catchup}} = \frac{D}{r - a}, \qquad r > a. \tag{39}$$

Download, verification and state application each impose their own rate limit. If the bottleneck does not exceed continuing arrival, the node cannot catch up. A node returning after an outage has not validated the missing interval merely because its old witness store was pruned.

## 9.5 Geographic participation and finality latency

TRAIDAG does not discard a timely registered block through a work-derived fork choice. That removes a specific stale-work mechanism; it does not make every location economically equivalent. Network distance still affects access to fresh anchors and transaction fees, likelihood of timely registration, data retrieval, view changes and the cost of operating a validator. Stake concentration, shared hosting and common routing dependencies must be measured rather than inferred from entry count.

At an idealised constant 25-block/s work rate, a miner with hash-rate fraction $10^{-4}$ has an expected raw solution interval of $1/(25 \cdot 10^{-4}) = 400$ seconds, or 6.67 minutes. This is not a guarantee of registration or payment, and it assumes the miner faces the same target and eligible work conditions as the model.

A cut cadence is not end-to-end transaction finality. The measurement points are:

wallet submission → mining and relay → registration commit → execution commit → later reward settlement.

For a block available around a healthy proposal boundary, the two-stage pipeline crosses approximately two to three cadences before execution commit:

$$L_{\text{block}\to\text{final}} \approx c\rho, \qquad 2 \lesssim c \lesssim 3. \tag{40}$$

At $\rho = 2.5$ s this is approximately 5–7.5 s. It assumes an uncongested queue, successful phase progression and processing within the cadence budget. It excludes time before the block exists. Wallet-submission latency additionally includes template selection, proof-of-work waiting and relay to an admission opportunity.

View changes and missing-data fetches can extend latency by their actual timeout and processing durations; the extension need not be an integer multiple of $\rho$. Before stabilisation there is no finite latency

bound. Reports must publish the start and end timestamps used, percentile distributions, load, topology and the view-change rate. A single healthy average is not an upper bound for internationally distributed nodes.

## 9.6 Coordinated parameter evolution

Block rate and cut cadence are separate control axes. A higher work rate increases headers, announcements, queue pressure, indexing, validation and reward records. A shorter cut cadence raises certificate traffic and shortens the healthy registration/execution pipeline. Neither change introduces a GHOSTDAG classification rule, but both can change consensus resource parameters and require an activated protocol version.

**GHOSTDAG tolerates concurrency by classifying it. TRAIDAG removes that classification from proof of work.**
Concurrent work blocks do not compete for blue or red status in TRAIDAG. Their ledger effect is determined by certified registration and execution; increasing the production rate still requires adequate relay, validation, storage and settlement capacity.

| Parameter | Launch profile | Rate test profile | Cadence test profile |
|---|---:|---:|---:|
| Work block rate | 25/s | 100/s | 25/s |
| Target cut cadence | 2.5 s | 2.5 s | 1.0 s |
| Target blocks / anchor | 125/2 | 250 | 25 |
| Registration block cap | 96 | 384 | 96 |
| Execution block cap | 128 | 512 | 128 |
| Registration window, heights | 96 | 96 | 240 |
| DAA half-life, blocks | 22,500 | 90,000 | 22,500 |
| Epoch / accounting length | 34,560 | 34,560 | 86,400 |
| Modelled block-to-finality | 5–7.5 s | 5–7.5 s | 2–3 s |

**Table 7.** Coupled illustrative operating profiles. The latter columns are test configurations, not demonstrated performance or scheduled mainnet activations. Byte, mass, state-growth and system-settlement budgets must be validated alongside the displayed counts.

A height-based emission schedule has a real-time interpretation only through actual cadence. Moving from 2.5 seconds to 1 second while leaving $\sigma_h$ unchanged advances the remaining schedule 2.5 times as fast under healthy operation. The upgrade must explicitly retain that consequence or replace the remaining schedule with a supply-preserving conversion. It must not mint historical budgets again or conceal a tokenomic change inside a latency optimisation.

Existing anchors keep their original work targets, registration windows and decoding versions. Already registered blocks keep their execution and payout obligations. An upgrade package therefore specifies legacy decoding support, state migration, pending buckets and reward cursors, evidence validity, snapshot pins and the activation-height rule. A shorter window cannot erase registered work; a longer one cannot revive already expired unregistered work unless that revival is expressly part of the activated rule.

A v1 node rejects unsupported consensus versions and unknown critical fields. It never follows a remote node's arbitrary parameter hash as an instruction to upgrade. Network activation requires a predetermined compatibility rule and the published manifest; a transport negotiation does not change ledger semantics.

The reference 100-block/s column demonstrates how counts must move together, not that the rest of the system has already been scaled.

# 10 Deployment boundaries and validation

TRAIDAG specifies one agreement path, one deterministic execution rule and explicit value domains. Its operational guarantees remain conditional on the cryptographic, stake, availability and implementation assumptions stated earlier. Those conditions belong in the technology description, not in an unrelated disclaimer.

Proof of work provides open block production and allocates the work reward share. It does not independently finalise the ledger or restore progress without a stake quorum. The active set is publicly enumerable and limited by the launch resource budget. Sentries protect signing infrastructure but do not establish independent ownership, jurisdictional diversity or a larger fault threshold.

The v1 policy chooses a halt over an automatically reduced quorum. It also makes checkpoint trust explicit and leaves the compute lane disabled. These are defined behaviours. They must not be presented as features that have already been replaced by an undocumented recovery or useful-work mechanism.

## 10.1 What this edition establishes

**Table 8.** Evidence classes and deployment obligations. A specification, an arithmetic check and a measured implementation result are not interchangeable.

| Claim | Basis in this edition | Required deployment evidence |
|---|---|---|
| Deterministic execution | Fixed transition order, encodings and Theorem 5.1. | Independent-client vectors, differential execution and platform tests. |
| Agreement across views | Weighted intersection and the pinned locking mapping. | Mechanised or bounded checking of the complete adaptation; clear model bounds. |
| State validity | Independent complete transition validation. | Positive and negative vectors for every transaction and system operation. |
| Value conservation | Disjoint domains and authorised mint delta. | Leaf/aggregate comparison, replay tests and atomicity fault injection. |
| Work settlement | Independent closure/execution readiness and one-use payout. | Both event orders, delayed cursor, zero claim, restart and pruning tests. |
| Committee transitions | Snapshot pins, coverage retry and explicit handoff. | Unequal weights, multiple extensions, missing handoff and key-rotation traces. |
| Difficulty | Exact integer rule and a delayed local model. | Full-loop simulation under strategic registration and overload; parameter report. |
| Cryptographic profile | FIPS primitive definitions and explicit message contexts. | Implementation review, RNG and side-channel assessment, downgrade tests. |
| Capacity and latency | Reproducible encoding arithmetic and declared models. | Multi-region measurements, stress tests and reproducible hardware details. |

| Claim | Basis in this edition | Required deployment evidence |
|---|---|---|
| Mainnet identity and issuance | Manifest schema and consistency checks. | Actual `EmissionV1`, genesis allocations, target, bootstrap keys and content hashes. |

The accompanying numerical script checks selected allocations, rounding, target arithmetic, reward-event ordering and encoding lengths. It does not run a distributed node, validate real ML-DSA signatures, simulate the complete BFT protocol or establish storage-crash safety. Its output is reproducible arithmetic evidence with an explicit scope.

## 10.2 Required adversarial and conformance suite

Consensus checking includes conflicting proposals, unequal stake weights, an inactive eligible subset, late quorums, NIL votes, view jumps, reused evidence, duplicate signatures, changed signer indices and wrong committee roots. A four-validator equal-weight model is a useful starting case, not coverage of the full weighted protocol. Tests include messages delivered in different orders to nodes that eventually accept the same certificate and must still derive identical proposer state.

Value tests include repeated inputs inside one transaction, duplicates across blocks, conflicting spends, stale owner nonces, duplicate withdrawal claims, invalid output overwrites, odd-value fee splits, empty buckets, zero rewards, both reward-event orders, slashed delegations and delayed payout cursors. They compare every value-domain aggregate with a reconstruction from effective leaves after each transition.

Crash injection interrupts execution before and after every durable boundary: signing intent, log flush, vote release, cut commit, reward output creation, reserve debit, cursor advancement, snapshot installation and pruning. A successful test leaves either the previous state or the complete next state; it must never leave a reusable input next to an already materialised output. Redundant signers are tested with fencing failure and stale-backup restoration.

Decoder fuzzing covers noncanonical integer widths, trailing data, unknown types, invalid lengths, duplicate map keys, malformed public keys, invalid parent proofs, oversized object counts, extreme heights and signed/unsigned conversion boundaries. Cross-language vectors include negative floor division and every difficulty-saturation edge. Limits are checked before allocation and expensive cryptographic operations wherever possible.

The difficulty programme tests hash-rate changes of ±50% and ±90%, halts and restarts, delayed registration near the window boundary, old-anchor mining, alternating arrivals, sparse buckets, persistent capacity flags and slow or variable cut cadence. Its acceptance criteria describe bounded state, workload recovery and observed controller behaviour for the tested environment. A Poisson process is not required to produce a flat trace.

## 10.3 Global testnet reporting

The test deployment uses independently operated nodes in several regions, heterogeneous hardware and more than one provider. It measures actual one-way delay and loss rather than assigning a continent an assumed fixed latency. Cases include a partition that retains a quorum, a partition where no side does, asymmetric reachability, validator restart, unavailable payloads and sustained overload. Public relay nodes, signing cores, sentries, miners and pruned synchronising nodes are reported separately.

Every performance report states software commit, manifests, compiler, hardware, storage, network shaping, run duration, transaction mix, candidate arrival rate, nominal slots, unique IDs and successful

executions. It publishes cut and transaction latency distributions, view-change frequency, registration delay, queue and payout backlog, bandwidth by direction, CPU, memory, disk growth, compaction and catch-up factor. Raw data and analysis scripts accompany the summary.

The mainnet activation package contains the final manifests, exact schemas and positive/negative vectors, reproducible binaries, migration policy, signer-recovery runbooks, historical-data and checkpoint procedures, and an issue-resolution record. Missing emission or genesis values must be supplied by the project before a concrete network can be instantiated. Mainnet measurements can establish real participation and long-run operating behaviour; they cannot retroactively substitute for absent transition rules or pre-release safety testing.

### 10.4 Operational interfaces

The node API exposes finalised height, cut ID, parameter hash, committee descriptor and sync mode. Block status distinguishes produced, admissible, registered, executed, payment-ready and paid. Transaction receipts identify the first effective occurrence, execution height and deterministic no-effect code. Explorer or wallet text must not label a merely registered transaction as executed-final.

Mining templates bind the anchor, target, payload, reward lock, nonce and extranonce exactly as the header specifies. Pool shares are off-chain accounting objects; they are not registration receipts or spendable consensus rewards. A Stratum bridge can translate work requests but cannot change the target, payout binding or solved payload after work has been found. The miner implementation must support the activated TenQ work profile. Its consensus evaluation and conformance requirements are defined by the TenQ specification; availability of particular Windows, Linux or HiveOS packaging is an implementation and release matter rather than a consensus guarantee.
Wallets verify network identity, authorisation contexts and finalised UTXO provenance. They disclose checkpoint mode, protect withdrawal keys separately from consensus keys and show counter-based unbonding progress. Seed generation, encrypted storage, recovery and signing-device support belong in the wallet security specification and its test suite. A browser interface cannot replace consensus validation with a trusted API response while claiming the full-node trust boundary.

## 11 Related work

Nakamoto consensus combines work-based Sybil resistance with a selected chain [1]. GHOST changes chain selection using subtree information [2]; SPECTRE uses pairwise voting with different ordering properties [3]; PHANTOM and GHOSTDAG derive a ledger order using graph classification [4]. They should not be described as one identical $k$-cluster algorithm. DAG KNIGHT further studies a parameterless graph-based construction and its own confirmation assumptions [5]. TRAIDAG's distinction is not that every alternative has the same latency parameter; it is that its ledger order is a stake-certified function of a committed queue rather than a work-derived graph order.

**Confirmation is not finality.**
A probabilistic confirmation threshold and a BFT execution commit provide different guarantees. TRAIDAG's settlement event is the execution-cut commit under the stated Byzantine stake bound; no additional proof-of-work confirmation depth is required after that event.

Kaspa's Crescendo specification changes the target interval to 100 ms, the GHOSTDAG parameter to 124, merge depth to 36,000 blocks and finality depth to 432,000 blocks [25]. That finality depth corresponds to twelve hours at the target interval; it is not a requirement to wait twelve hours before accepting every

payment. Practical confirmation, the protocol's finality horizon and TRAIDAG's BFT execution commit must be interpreted under their respective security models.

**Point of sale.** A checkout needs a definite acceptance event. With a probabilistic confirmation policy, the merchant chooses a risk threshold. In TRAIDAG, the protocol settlement event is a valid execution-cut commit: the recipient output has been applied, the state transition has been checked and the required stake quorum has committed the result. Within the agreement fault model, the protocol asks for no further confirmation depth. Registration alone, a mempool receipt or an unchecked certificate is not that event. The launch model places block-admission-to-execution finality at approximately 5–7.5 seconds under healthy, uncongested conditions (Equation (40)); this is distinct from the 2.5-second cut cadence and is not a measured mainnet result. A payment interface using a trusted gateway must disclose that additional trust assumption.

The agreement layer belongs to the partially synchronous BFT family of PBFT and Tendermint [7, 8]. HotStuff studies responsive, pipelined BFT with a different communication structure [28]. It is not fundamentally incompatible with explicit signatures: certificate representation changes its costs, not the existence of the protocol. TRAIDAG uses the pinned Tendermint-style discipline so its lock and re-proposal rules have one explicit reference.

Narwhal separates reliable data dissemination from ordering, and Tusk and Bullshark supply ordering protocols over that infrastructure [9, 10]. TRAIDAG has a related separation of responsibilities, but its proposed work data originates from an open, rewarded mining population rather than exclusively from committee producers. Registration still depends on the ordering committee, so open production is not synonymous with guaranteed admission.

Prism separates proposal and confirmation functions, while Bitcoin-NG separates leader election from transaction serialisation [26, 27]. TRAIDAG similarly avoids making one work block perform every ledger role, but assigns its ordering certificate to bonded stake. Casper FFG illustrates a finality mechanism attached to a separate fork-choice process [11]; the underlying work graph here has no selected work chain to override.

Interactive verification systems such as TrueBit and Arbitrum are relevant to a possible future compute-settlement lane [29, 30]. They do not constitute a deployed v1 component or a proof that arbitrary training jobs can already be verified cheaply. Enabling computation would require a separate execution model, dispute protocol, resource budget and security analysis under a new activated specification.

## 12 Conclusion

TRAIDAG separates permissionless work production, certified registration and deterministic ledger execution. The work graph retains concurrent proposals without using their parent relation as a fork-choice rule. The committed queue fixes which blocks execute next, while a stake-weighted BFT certificate finalises the resulting state under the stated fault bound.

The separation gives clear responsibilities. Miners bind work to an immutable anchor, payload and reward destination. Validators attest to available data and an exact transition. Fully validating nodes independently check authorisation, order, accounting and commitments. Subsidy is budgeted by anchor, fees are transferred once, and deferred reward settlement has explicit readiness and payment states.

The launch profile combines 25-block/s production and a 2.5-second cut-cadence target with post-quantum authorisation and explicit quorum certificates. Its bandwidth and latency figures are reproducible models, not substitutes for a global testnet. Future profiles can change rate and cadence while preserving

the architectural separation, provided resource budgets, monetary timing and pending obligations are migrated coherently.

The resulting technology has a precise operational boundary: deterministic state execution and conditional BFT agreement, with a defined halt when the required quorum is unavailable. Deployment quality depends on implementing those rules exactly, publishing the concrete network manifests and validating the system under the international, heterogeneous conditions in which it will operate.

# A Canonical encoding profile

This appendix defines the reference encoding contracts used by the revised size calculations and transition rules. Byte strings are concatenated without implicit padding. All unsigned integers are fixed-width big-endian; signed integers are fixed-width two's complement. Counts precede variable-length collections. Decoders reject trailing bytes, out-of-range counts, duplicate set keys and unrecognised critical types. No decoder may silently skip an unknown consensus field.

## A.1 Primitives and domain separation

`Hash384` is 48 bytes; `Target256` is a 32-byte unsigned threshold value consumed by the activated proof-of-work profile. A lock is `u8(profile) || bytes40(commitment)`. Profile 1 uses ML-DSA-65 and the spend-lock construction in Equation (3). A variable byte string is `u32(length) || bytes[length]`. An optional fixed-format object is one byte, 0 for absent or 1 followed by its encoding for present; other discriminants are invalid. Arrays use a `u32` count followed by their entries, except the parent count, explicitly a `u8`.

Text in hash domains is exact ASCII. The symbolic names in the main text expand as shown in Table 9; the trailing `_V1` is literal and is separate from any transmitted version integer.

**Table 9.** Exact domain strings for the reference encoding profile.

| Symbol | Exact domain string |
|---|---|
| POW / BLOCK | `TRAI_POW_V1` / `TRAI_BLOCK_V1` |
| TX / WITNESS / SPEND | `TRAI_TX_V1` / `TRAI_WITNESS_V1` / `TRAI_SPEND_V1` |
| OPAUTH / POP | `TRAI_OPAUTH_V1` / `TRAI_POP_V1` |
| EXEC / ORD / CUT | `TRAI_EXEC_V1` / `TRAI_ORD_V1` / `TRAI_CUT_V1` |
| REG | `TRAI_REG_PRIORITY_V1` |
| VOTE / PROPOSAL | `TRAI_VOTE_V1` / `TRAI_PROPOSAL_V1` |
| NIL / NO-TRANSITION | `TRAI_NIL_V1` / `TRAI_NO_TRANSITION_V1` |
| HANDOFF / TRANSITION | `TRAI_HANDOFF_V1` / `TRAI_TRANSITION_V1` |
| WORK-REWARD / STAKE-REWARD | `TRAI_WORK_REWARD_V1` / `TRAI_STAKE_REWARD_V1` |
| VALIDATOR / POSITION | `TRAI_VALIDATOR_ID_V1` / `TRAI_POSITION_ID_V1` |
| SET / STATE / PARAMS | `TRAI_VALIDATOR_SET_V1` / `TRAI_STATE_V1` / `TRAI_PARAMS_V1` |
| GENESIS / EMISSION | `TRAI_GENESIS_V1` / `TRAI_EMISSION_V1` |

| Symbol | Exact domain string |
|---|---|
| EVIDENCE / POOL | `TRAI_EVIDENCE_V1` / `TRAI_POOL_V1` |

The hash-input layouts explicitly include chain and protocol identity wherever objects can cross networks or versions. A digest included inside another object remains bound transitively; it is not reinterpreted as a different object type merely because the lengths match. The NIL digest is $H(\text{NIL})$ and the no-transition digest is $H(\text{NO-TRANSITION})$, never an all-zero value standing for multiple meanings.

## A.2 Collection roots and sparse state maps

For a collection domain $D$, define the ordered-leaf tree:

```
leaf(x)    = SHA384(D || "_LEAF" || u32(len(x)) || x)
empty_leaf = SHA384(D || "_EMPTY_LEAF")
node(l, r) = SHA384(D || "_NODE" || l || r)
root(xs)   = SHA384(D || "_ROOT" || u32(len(xs)) || tree(xs))

tree([]) = SHA384(D || "_EMPTY_TREE")
For a nonempty list: hash leaves in the specified order;
pad to the next power of two with empty_leaf; combine pairs.
Never duplicate the last real leaf to obtain padding.
```

The domains are `TRAI_PARENTS_V1`, `TRAI_PAYLOAD_V1`,`TRAI_REGISTRATION_V1`, `TRAI_QUEUE_V1`,`TRAI_SNAPSHOT_V1`, `TRAI_BUCKET_V1` and`TRAI_POOL_ORDER_V1` respectively.

Parent IDs are sorted lexicographically; payload witness IDs retain the transaction list order; registration entries follow their specified priority; queue entries retain queue order. A collection root without the real leaf count is not this construction.

The execution-set root retains the simpler fixed-ID encoding:

```
exec_root = SHA384("TRAI_EXEC_ROOT_V1" || u32(count)
                   || sorted_block_id[0] || ...)
```

Duplicate execution IDs are rejected. The order used to construct this set root is not the subsequent seeded execution order.

Keyed state domains use a binary sparse Merkle map over 384-bit keys, traversed most-significant bit first. The domain name identifies the map. At depth 384, an absent leaf is $H(D \parallel \text{\_EMPTY})$ and a present leaf is $H(D \parallel \text{\_LEAF} \parallel k \parallel \text{enc}(v))$. At depth $d < 384$, an internal node is $H(D \parallel \text{\_NODE} \parallel \text{u16}(d) \parallel l \parallel r)$. Empty internal roots are obtained recursively from two empty children at the next depth. Canonical record layouts, not database serialisations, define $\text{enc}(v)$.

For a value-bearing map, each node also carries its checked effective `u128` subtotal. The subtotal follows the child digests in the internal-node hash; the leaf subtotal follows the encoded record in a present-leaf hash. An absent leaf has subtotal zero. The internal subtotal is exactly the sum of its two child subtotals. Map-root encoding is `Hash384(root) || u128(subtotal)`. Non-value maps have no subtotal field. The map type is fixed by its domain, not selected by a received proof. Slashed-generation effective balances follow the generation aggregate model of Section 4.6; position face amounts are not a second value-bearing map.

A membership or absence proof specifies the key and siblings by depth. Its canonical uncompressed form carries 384 siblings, with subtotals where applicable. Storage implementations may compress proofs,

but any wire compression profile must define omitted empty subtrees and reject ambiguous encodings. The reference full-node path computes roots from state and does not require such large proofs on every transaction.

### A.3 Payment and native-operation envelope

The reference transaction body has the following field order:

```
TxBody = u16(protocol_version)
       || chain_id Hash384
       || u8(transaction_type)
       || u64(valid_from_height) || u64(valid_until_height)
       || u32(input_count) || Input[input_count]
       || u32(output_count) || Output[output_count]
       || u32(operation_length) || operation_bytes

Input  = previous_tx_id Hash384 || u32(output_index)
Output = u128(amount) || u8(lock_profile) || bytes40(lock)

Witnesses = u32(witness_count) || Witness[witness_count]
Witness   = u8(profile) || bytes1952(public_key) || bytes3309(signature)
WireTransaction = TxBody || Witnesses
```

The validity interval is inclusive and nonempty. `valid_until_height=0` is not an unbounded sentinel. Payments have type 0 and an empty operation. All external transactions consume at least one ordinary UTXO, including control transactions; this provides fee funding and prevents a zero-input replay path. Amounts are strictly positive for created UTXOs. The first `input_count` witnesses authorise inputs in their committed order. Operation witnesses follow in the fixed role order of the transaction type. No unused witness is permitted.

The body length for one input, two outputs and no operation is

$$2 + 48 + 1 + 8 + 8 + 4 + 52 + 4 + 2(16 + 1 + 40) + 4 = 245 \text{ bytes}.$$

The four-byte witness count and the 5,262-byte input witness bring the wire transaction to 5,511 bytes. An enclosing block may add its own transaction-length field; that framing is counted in the block, not a second time in the transaction body.

```
txid  = SHA384("TRAI_TX_V1" || canonical_TxBody)
wtxid = SHA384("TRAI_WITNESS_V1" || txid || canonical_Witnesses)

SpendMessage(i) = SHA384("TRAI_SPEND_V1" || chain_id
                         || u16(version) || txid || u32(i)
                         || anchor_input_amount u128
                         || anchor_input_lock Lock41)
```

The containing block's anchor selects the historical input record used for verification. It does not enter `txid`, allowing a still-valid transaction to be included against a newer finalised anchor without changing its intended outputs. The input amounts and locks must match both the signature context and the retained historical record. Domain separation prevents an input signature from being reused as an owner-operation signature.

ML-DSA signs the 48-byte application digest with its pure signing interface and the literal context `TRAI_V1`. This is not an implicit switch to the HashML-DSA prehash interface. Verification uses the same

interface and context. Profile 1 fixes the key and signature lengths; algorithm-internal encodings are those of FIPS 204. Test-only zero-filled signature strings in the calculation script verify lengths, not cryptographic validity.

### A.3.1 Native operations

A transaction contains one native operation at most. Types 1–8 in Table 10 give exact payload concatenation order. `ID` is `Hash384`; `Lock` is the 41-byte lock; `nonce` is `u64`; `amount` is `u128`. Counts and variable strings use the primitive encoding above.

**Table 10.** Native operations: payload fields, authority and state effect.

| Type | Operation payload | Additional authority and effect |
|---|---|---|
| 1 REGISTER | `consensus_pk[1952], owner_lock, reward_lock, self_bond_amount` | Owner proof, then consensus-key proof of possession; create validator and self-stake position. |
| 2 BOND | `validator_id, owner_lock, reward_lock, amount` | Position-owner proof; create a new immutable principal lot assigned to a non-jailed validator. |
| 3 EXIT | `validator_id, expected_owner_nonce` | Registry-owner proof; prevent later activation after existing snapshot obligations retire. |
| 4 UNBOND | `position_id, expected_position_nonce` | Position-owner proof; request exit of the entire lot, not an ambiguous partial mutation. |
| 5 WITHDRAW | `position_id, expected_position_nonce` | Position-owner proof; consume one released, matured unbonding claim; value returns in the transaction balance. |
| 6 ROTATE | `validator_id, expected_owner_nonce, new_consensus_pk[1952]` | Registry-owner proof, then new-key possession proof; activate through a future eligible snapshot. |
| 7 REWARD-LOCK | `position_id, expected_position_nonce, new_reward_lock` | Position-owner proof; affects future snapshots, not existing reward claims. |
| 8 EVIDENCE | `bytes(message_1), bytes(message_2)` | No extra owner witness; both signed offending messages verified against their historical descriptor. |

For REGISTER, validator_id = $H(\text{VALIDATOR} \,\|\| \text{chain} \,\|\| \text{txid})$. A new position has ID $H(\text{POSITION} \,\|\| \text{chain} \,\|\| \text{txid} \,\|\| \text{u32}(0))$; one lot is created by each REGISTER or BOND operation. Genesis uses manifest-defined IDs under a distinct genesis-object construction. REGISTER marks its initial lot as self-bonded; a BOND lot counts as self-bond only when its owner lock equals the registry owner lock at creation. Registry ownership is not transferable in this profile.

An operation-owner digest is $H(\text{OPAUTH} \,\|\| \text{chain} \,\|\| \text{u16}(version) \,\|\| \text{txid} \,\|\| \text{u8}(role))$. The role byte is 0 for registry ownership and 1 for position ownership. A possession digest uses domain POP and role 2. Witness order is input witnesses, owner witness if required, then key-possession witness if required. A key possessing a spend input need not own a delegated position; each authority is checked independently.

Every operation is checked against the anchor state for authorisation and against the execution state for current applicability. Nonces increment only on successful owner operations. A lot cannot be unbonded or withdrawn while it is pinned by an active or scheduled snapshot. A withdrawal uses the lot's effective unbonding amount, not a face amount left behind in a slashed generation. For balance purposes,

$$\sum \text{UTXO inputs} + \text{withdrawn principal} = \sum \text{UTXO outputs} + \text{new bonded principal} + F. \quad (41)$$

All terms are nonnegative; principal source and destination terms are zero for transaction types that do not move principal. Slashing caused by EVIDENCE is a separate authorised system movement from principal to burned value, never income for the submitter. Evidence for a generation already fully slashed has no new effect and returns code 6. A generation remains evidence-addressable while pinned and through its entire final unbonding interval; its historical committee descriptors cannot be pruned merely because the offending vote is older than seven epochs. The evidence horizon retires only after its last snapshot exposure and principal-liability window have ended.

Successful application yields result code 0. Stateful no-effect codes are 1 for an already effective duplicate, 2 for a spent or absent current input, 3 for an execution-height range failure, 4 for a nonce mismatch, 5 for an unavailable or immature operation claim and 6 for an ineligible current operation target. When several apply, use the smallest code after structural checks. A missing body is not a result code: the cut remains unvalidated until its required data is obtained.

Structural errors reject the containing work block at registration: malformed length, unsupported type, repeated input, invalid witness, impossible anchor input, arithmetic overflow, invalid output amount or inconsistent operation layout. At execution an operation that was authorised at the anchor can become inapplicable; it then has no effect. This distinction prevents a later conflict from retrospectively changing whether the block's original registration was valid.

## A.4 Block header and body

The canonical work-bound header is:

```
Header = chain_id Hash384 || u16(protocol_version)
       || u8(pow_algorithm_id) || work_ref Hash384
       || parents_root Hash384 || payload_root Hash384
       || u32(payload_length) || u64(payload_mass)
       || reward_lock Lock41 || nonce bytes32 || extranonce bytes32

pow_preimage = "TRAI_POW_V1" || Header
pow_value    = POW_PROFILE(pow_algorithm_id, pow_preimage)
block_id     = SHA384("TRAI_BLOCK_V1" || Header)

Body = u8(parent_count) || sorted_parent_ids[parent_count]
     || u32(transaction_count)
     || (u32(transaction_length) || WireTransaction)[transaction_count]
```

`payload_length` is the full body length, including parent and transaction framing. The canonical header is 312 bytes. `MAX_BLOCK_BYTES` bounds header plus body; registration and execution byte counters sum that full encoded size for every block occurrence. Authorisation counters count all required witnesses, including operation proofs. The body recomputes both roots. The payload-root collection consists of ordered `wtxid` values; duplicate transaction IDs within a block are rejected in this profile to avoid gratuitous amplification. Duplicates across different blocks retain the execution semantics in the main text. `payload_mass` is recomputed from the manifest's exact operation costs, never trusted because the miner wrote it into a solved header.

The v1 network manifest fixes TenQ as the supported work profile for this protocol version. Its pow_algorithm_id is 0x02 and its deterministic POW_PROFILE evaluation is defined by the accompanying TenQ work specification. The profile returns a 256-bit unsigned work value, which must not exceed the target committed by work_ref. A node that does not implement the activated TenQ profile cannot validate that network. Zero through sixteen distinct parents are accepted. Parent references do not affect ledger order, rewards or the anchor target.

For a child anchor $a$, each parent header names an already finalised anchor $a_p$ with $\max(0, a - W_{\mathrm{REG}}(a)) \leq a_p \leq a$. Its identifier, anchor and header proof of work are checked without recursively fetching its payload or every ancestor. This bounded parent-header contract is a clarification introduced by this edition. Required anchor target records and parent headers remain available while validation can require them. A parentless block is valid; a locally unavailable referenced header causes a bounded fetch, not a different structural answer at different nodes. Cyclic self-references would additionally require solving the hash-binding equations; explicit duplicate and self-ID references are rejected.

An honest miner uses the newest finalised anchor it knows. The validator checks the anchor in its certified history, not a claim about the wall-clock instant at which work was performed. The registration window counts heights after the anchor, regardless of how old an untrusted peer says the body is.

## A.5 Votes, proposals and certificates

The semantic vote body is exactly:

```
VoteBody = chain_id Hash384 || u16(protocol_version)
         || validator_set_id Hash384 || u64(height) || u64(view)
         || u8(phase) || value_digest Hash384

Vote = VoteBody || u32(signer_index) || signature bytes3309
VoteSignable = SHA384("TRAI_VOTE_V1" || VoteBody)

Certificate = VoteBody || u32(signer_count)
            || (u32(signer_index) || signature bytes3309)[signer_count]
```

Phases 0 and 1 are prevote and precommit. Phase 2 is handoff; its value is the handoff digest, its height is the first incoming height and its view is zero. Handoff uses the incoming set ID, weights and version. Signers are strictly increasing by index in the canonical active descriptor. Each appears once and all signatures authenticate the identical semantic vote body. The vote body is 163 bytes, an individual indexed vote 3,476 bytes and a certificate $167 + 3313n$ bytes. Public keys are resolved from the authenticated set descriptor rather than repeated in each certificate.

A proposal has the following signed semantic header:

```
ProposalBody = chain_id || u16(version) || validator_set_id
             || u64(height) || u64(view) || cut_id
             || u8(has_valid_view)
             || [u64(valid_view) || polc_semantic_digest]
ProposalSignable = SHA384("TRAI_PROPOSAL_V1" || ProposalBody)
```

The optional fields exist only when the flag is 1. The proposal transports the cut contents, its proposer index and signature, and the corresponding prevote certificate when a valid view is claimed. The evidence digest binds its semantic vote body and sorted signer indices; every accompanying signature is verified. Signature randomness is not allowed to seed ledger order or alter the cut ID.

The handoff digest is computed after the last old cut exists. It binds old and new epoch IDs, the last old height and cut ID, queue root, incoming set ID and incoming parameter hash in the order stated in Section 4.6. The first new cut's transition digest binds that semantic handoff and the incoming descriptor. The proof signatures are validation evidence rather than self-referential state fields.

# B Committed state and transition order

The state commitment is a versioned composite. Each component has its own domain and canonical map or ordered-collection encoding. The root fixes every fact that can change a later transition; implementation caches and peer metadata are not state.

```
state_root = SHA384("TRAI_STATE_V1" || u16(version) || u64(height)
                    || utxo_root || queue_root || block_status_root
                    || validator_registry_root || stake_state_root
                    || proposer_priority_root || epoch_state_root
                    || emission_root || daa_root || settlement_root
                    || compute_root || params_hash)
```

A value-bearing root includes its subtotal in the component encoding. The fields above occur in exactly that order. `compute_root` is the domain-separated empty compute root in v1; compute escrow and compute reserve aggregates are zero.

The complete root-to-record mapping is given in Table 11; clients must commit every listed field even when an implementation stores the records in a different physical database layout.

**Table 11.** Committed state domains and the records bound by each root.

| Domain | Required committed content |
|---|---|
| UTXO | Outpoint key; positive amount; lock; actual creation height; source kind. The map's subtotal is $V_{\text{utxo}}$. |
| Queue | Ordered registration entries: block ID, work reference, anchor height, registration height, anchor parameter ID, encoded bytes, logical mass and authorisation count. |
| Block status | Registration/execution flags and execution height; bound reward lock; closed-bucket index; work-fee claim; payment-queued and paid flags; readiness height. |
| Validator registry | Stable ID, owner lock, owner nonce, consensus-key version and public key, exit intent, jail/penalty generation and pending key activation. |
| Stake state | Position ID, owner and reward locks, assigned validator and generation, principal, lifecycle status, expected nonce, snapshot pins, unbonding release epoch and maturity counter. Generation aggregates determine effective bonded/unbonding value. |
| Proposer priority | Active entries in canonical index order and their signed `i128` priorities. |
| Epoch state | Epoch ID, extension index, start and retry heights, frozen eligible and active descriptors, $W_e$, $Q_e$, pending activation descriptor, snapshot-pinning references and handoff-armed state. |

| Domain | Required committed content |
|---|---|
| Emission | Minted and burned counters, work and stake reserves, scheduled work budgets, unminted carry, both basis-point residues and active emission-manifest ID. |
| Difficulty | Controller segment, initial target and origin, error numerator, current target, bucket counts, rolling capacity flags and closed-anchor index. |
| Settlement | Closed-bucket descriptors; sorted ready-payment queue; interval/snapshot reward pools, their immutable amounts, beneficiary order, cursors and completed-pool markers. |

Fixed integer widths follow Section A: heights, epochs, nonces and readiness heights use `u64`; values and weights use `u128`; count and index fields use `u32`; Boolean and enum fields use `u8`. Current error numerators use signed 256-bit encoding. Roots and identifiers are 48 bytes. Each record begins with its schema version and encodes its fields in the order shown by the corresponding native object schema. Explicit per-object byte vectors belong to the release schema package; a change in field order is a protocol change, not a database migration.

Objects with several ordered subcollections commit those collections separately. For example, a stake snapshot commits validators by ID and, within each validator, principal positions by ID with their effective historical weights and reward locks. Its descriptor commits total eligible weight, top-$N$ membership, quorum, epoch and source snapshot root. A queue entry's index is not inherited from database iteration order. Snapshot hashes are content identities; arbitrary remote snapshot roots are not accepted without the transition that authorises them.

No post-state field depends on the identifier of the cut whose state it describes. For historical anchor lookup, the verified cut ID is appended to the certified-history index only after the post-state is formed and the cut is certified. Objects created in the transition use height, transaction ID, block ID or predetermined semantic operation IDs. This prevents the execution/state/cut hashing cycle from reappearing through a reward or handoff record.

## B.1 Full cut application

A valid cut is computed from one immutable prior state. Temporary writes use a transaction or copy-on-write overlay and are committed only after complete validation and quorum verification.

1. Resolve the height's protocol manifest, committee and any required incoming handoff from the preceding finalised state. Reject a mismatching predecessor, unknown version or wrong descriptor. Fix the eligible reward snapshot for income at this height.
2. Select the required execution prefix from the prior committed queue. Validate all new registration bodies and their anchors in a read-only view. Their membership does not change the current execution prefix.
3. Form the execution-set root and execution identifier. Derive the total block order. Execute each block's committed transaction list with atomic per-transaction writes, deterministic no-effect codes and exact fee allocation. Mark block execution once.
4. Remove executed entries from the queue. Append the new batch in canonical registration order; set registration markers and update the appropriate open anchor buckets. Verify every per-cut and queue cap on the resulting counters.

5. Apply this height's scheduled work/stake split once. Mint only the stake share now; retain the work budget for its anchor. At a closing boundary, close the oldest newly complete anchor bucket after including this cut's final eligible registrations. Fix its work claims, carry empty budgets and mint the nonempty budget exactly once.
6. Update the committed saturation history and integer difficulty controller from newly closed buckets. Open buckets do not contribute to final error. Associate the resulting target with this cut once it is finalised.
7. Complete applicable accounting-interval pools, freezing their snapshot identities and amounts. Merge newly ready work claims in ($h_{\text{ready}}$, block_id) order and process the required work-payment prefix. Then process the required oldest-first stake-payout prefix, counting zero claims against the record cap.
8. Apply snapshot lifecycle transitions, pending exits and scheduled epoch evaluation at their defined boundaries. An incoming handoff cannot be computed from a cut ID that does not yet exist; an armed successor descriptor is committed first. Apply aggregate principal release only when all active and scheduled pins have retired.
9. Advance committed proposer priorities by one step, or reset them for an activating new epoch according to its prescribed first-height state. Derive all component roots and aggregate values. Check the authorised mint delta, supply cap and value-domain equality.
10. Form the final cut ID over the complete commitments. Verify its precommit certificate and any boundary evidence. Atomically store the cut, certificate, state and application index. Release outward commit notification only after durable installation.

The incoming epoch's zero priority vector is the starting vector for its first height, whose cut advances it by one step. The outgoing epoch's extension does not reset priorities. This distinction prevents two clients from disagreeing about the second proposer of a newly activated epoch.

Transactions in the execution prefix cannot spend outputs created later in the same cut's system-settlement phase. Their anchor inputs predate that cut. Registry changes made by those transactions may affect a scheduled future snapshot, not the frozen signer weights of the current height. A failed full-cut validation rolls back every overlay change, including fee residues and payout cursor increments.

Automatic release and snapshot operations have finite manifest bounds. A registry snapshot may be precomputed and validated incrementally from prior state, but a release must still expose the complete deterministic result. No implementation is allowed to complete a different subset because its disk was slower. The release cost registry must cover the worst permitted snapshot size, not only user transactions.

## B.2 Retention and pruning conditions

An unregistered block can be dropped once its window is irreversibly expired. A registered body remains required until its execution and any prescribed handoff of the queue. Its reward status remains required until the bucket, fee claim and payout are complete. An execution marker alone is not sufficient to delete an unpaid liability.

The historical anchor-state horizon covers every still-admissible new registration. A successfully registered block retains the validated input provenance needed to execute it even after that horizon passes. Committee descriptors and their key histories persist through evidence validation and checkpoint reconstruction. Old snapshot principal pins remain until no activation or penalty path can use them.

A validator that voted for a nonfinal proposal retains its bodies until the height is conclusively decided or the proposal remains its live valid/locked value. A timeout alone is not a release event. After a different

cut commits at that height, unregistered bodies with no other obligation can be released. Committed bodies follow the stronger queue-retention rule.

Pruning deletes historical representations, not live value. It cannot decrement a reserve, erase a claim, reset a consumed nonce or drop a snapshot pin. Pruned and archive nodes must derive the same new roots from any shared validated starting state.

# C Integer difficulty algorithm

The following pseudocode is the normative arithmetic contract for the fixed-profile controller. It avoids unbounded shifts and states the rounding of negative inputs explicitly. Implementations use sufficient fixed-width intermediates for the manifest-bounded ranges; the accompanying Python reference uses mathematical integers.

```
TargetFromError(Z, T0, Tmax, k_den, half_life):
    require 1 <= T0 <= Tmax <= 2^256 - 1
    require k_den > 0 and half_life > 0
    R = 65536
    e = floor((Z * R) / (k_den * half_life))
    nu = floor(e / R)
    phi = e - nu * R                 # 0 <= phi < R
    F = R + ((195766423245049*phi
              + 971821376*phi^2 + 5127*phi^3 + 2^47) >> 48)
    U = T0 * F                       # unsigned 512-bit intermediate
    shift = nu - 16
    if shift >= 0:
        if shift >= bit_length(Tmax): return Tmax
        if U > (Tmax >> shift): return Tmax
        raw = U << shift
    else:
        if -shift >= bit_length(U): raw = 0
        else: raw = U >> (-shift)
    return min(Tmax, max(1, raw))
```

Error accumulation adds $z_a = k_{\text{num}} - k_{\text{den}} N_a$ on the once-only closure of bucket $a$, except that positive $z_a$ becomes zero when that bucket's capacity flag is set. The accumulator is not recomputed from a node's current cache of blocks. A manifest bounds height, counts and numerator products within signed 256-bit range. An out-of-range input rejects the manifest or transition; arithmetic never wraps.

For a cut with registration usage $(b_r, s_r, m_r, a_r)$ and resulting queue usage $(b_q, s_q)$, the committed headroom flag is the disjunction

$$\begin{aligned}(b_r + 1 > B_r) \;\vee\; (s_r + S_b > S_r) \;\vee\; (m_r + M_b > M_r)\\ \vee\; (a_r + A_b > A_r) \;\vee\; (b_q + 1 > B_q) \;\vee\; (s_q + S_b > S_q),\end{aligned}$$

where $(B_r, S_r, M_r, A_r)$ are registration caps, $(B_q, S_q)$ queue caps, and $(S_b, M_b, A_b)$ maximum single-block byte, mass and authorisation costs. A bounded comparison such as $S_b > S_r - s_r$ avoids overflowing the addition. A closing anchor's capacity flag is the OR of cut flags over its inclusive registration window. The headroom flag is deterministic even when no actual omitted block has maximum size; that conservatism is intentional.

Required edge vectors include $Z = 0$, $\pm 1$, $\pm k_{\text{den}}\lambda$, large positive and negative values, $T_0 = 1$, $T_0 = T_{\max}$, integer fractional-boundary transitions and early heights with no closed bucket. At $Z = 0$ the target is $T_0$;

at $Z = k_{\mathrm{den}}\lambda$ it doubles unless capped; at the negative counterpart it halves with floor rounding and a minimum of one.

# D Genesis, emission and parameter manifests

The protocol description is distinct from a concrete network instance. A valid instance publishes canonical genesis, emission and parameter bytes together with their content hashes. The supplied source whitepaper named `EmissionV1` but did not include its curve, maximum supply or numerical bootstrap data. This edition defines the required interface and rejects silent defaults; it does not replace those missing project decisions with another chain's values.

## D.1 Network identity and bootstrap

The genesis manifest includes an independent 48-byte chain identifier, protocol version, parameter manifest, emission manifest hash, base-unit denomination, initial UTXOs and principal positions, validator public keys and proofs of possession, snapshots for epochs 0 and 1, and initial positive work target. The chain identifier is an explicit network-domain value, not a hash equation that recursively includes roots already hashed with that identifier. Software pins both the expected chain ID and the canonical genesis hash.

Genesis collections use distinct deterministic object identities derived from the genesis domain, chain ID, object-kind byte and canonical index. Allocation entries are sorted by the manifest's explicit index and reject duplicate identifiers, invalid locks, invalid public keys or unsupported values. Initial effective native value equals the genesis minted amount $G$; burned value and all ordinary reward reserves are zero. Genesis allocations already bonded are not counted again as UTXOs.

The initial queue, open-registration counts, reward cursors and proposer priorities are empty or zero. Anchor 0 names the genesis cut; its target is $T_0$ and its subsidy is zero. Bootstrap snapshots are explicit and satisfy the same positive-weight, coverage and authorisation checks as later active sets. The special snapshots resolve the otherwise nonexistent $e - 2$ history for epochs 0 and 1. Ordinary two-epoch snapshot lag applies thereafter.

The launch process must ensure initial committee availability. A manifest alone cannot cause its key owners to run nodes. A pre-launch network-identity and possession check prevents the first height from depending on an unknown key or a configuration placeholder. Signing a live genesis and using the same consensus key in a test network are operationally separate activities.

## D.2 EmissionV1 schema

The emission manifest contains a schema version, chain ID, denomination exponent, $I_{\max}$, genesis allocation total $G$ and an ordered list of nonoverlapping subsidy segments. A segment is `start_height u64 || end_height u64 || subsidy_per_height u128`, with inclusive endpoints. Heights start at 1; the segments cover the configured finite issuance interval without overlap. Outside that interval subsidy is zero. A finer-grained discrete decay can be represented by shorter segments; the representation does not select a particular economic curve.

The manifest rejects negative interpretations, zero-length ranges, unordered intervals and overflow in $(\mathrm{end} - \mathrm{start} + 1)\sigma$. It checks $G + \sum \sigma_h \le I_{\max} < 2^{120}$ with unsigned 256-bit intermediates. A project that intends exact exhaustion of the cap additionally sets equality as its manifest policy. The work share is 5,000 basis points; no network client may derive a different share from a local configuration file.

emission_id = $H$(EMISSION || enc(EmissionV1)) is committed by the parameter manifest and genesis. An implementation must read the actual published values before enabling transaction production or validation. An absent curve is not interpreted as zero issuance, and an absent cap is not interpreted as an unlimited asset.

## D.3 Launch reference values

The minimum validator bond $B_{\min}$ is a network admission parameter, not a value derived from the quorum inequality or the target block rate. This edition specifies no numerical amount. The parameter manifest MUST supply a positive `u128` base-unit value consistent with the asset denomination and supply cap; an absent value is a startup error. Its selection must account for validator admission, registry capacity and stake distribution. No fixed registration-latency guarantee follows from that choice alone.

**Table 12.** Launch reference parameters and required network-manifest values.

| Parameter | Reference value or required source |
|---|---|
| `TARGET_CADENCE_MS` | 2,500; operating target, not a validity deadline. |
| `K_NUM`, `K_DEN` | 125, 2. |
| POW_ALGORITHM_ID | 0x02 (TenQ); deterministic evaluation fixed by the TenQ work profile. |
| `MAX_PARENTS` | 16. |
| `MAX_BLOCK_BYTES` | 1,048,576. |
| `MAX_BLOCK_MASS` | 2,000,000. |
| `N_MAX` | 64 active validator entries. |
| `ACTIVE_COVERAGE_BPS` | 8,000 of eligible stake; quorum independently required. |
| `MIN_VALIDATOR_BOND` | $B_{\min}$: positive native base-unit amount fixed by the network manifest. |
| `EPOCH_LENGTH_HEIGHTS` | 34,560. |
| `ACCOUNTING_INTERVAL_HEIGHTS` | 34,560; distinct from committee extensions. |
| `W_REG` | 96 for anchors created under this profile. |
| `MAX_REGISTRATION_BLOCKS` | 96. |
| `MAX_REGISTRATION_BYTES` | 8,388,608. |
| `MAX_REG_MASS` | 64,000,000. |
| `MAX_REG_AUTHS` | 4,096. |
| `MAX_EXEC_BLOCKS` | 128. |
| `MAX_EXEC_BYTES` | 12,582,912. |
| `MAX_EXEC_MASS` | 96,000,000. |
| `MAX_EXEC_AUTHS` | 6,144. |
| `MAX_QUEUE_BLOCKS` | 4,096. |
| `MAX_QUEUE_BYTES` | 268,435,456. |
| `MAX_WORK_PAYMENTS` | 1,024 records per cut; new reference settlement budget. |

| Parameter | Reference value or required source |
|---|---|
| `MAX_STAKE_PAYOUT_RECORDS` | 512 records per cut; new reference settlement budget. |
| `DAA_HALF_LIFE_BLOCKS` | 22,500. |
| `DAA_CAPACITY_FILTER` | Enabled; consensus rule, not a local switch. |
| `WORK_SHARE_BPS` | 5,000, for subsidy and fees. |
| `UNBONDING_EPOCHS` | 7 completed committee epochs after release. |
| `EVIDENCE_EPOCHS` | 7; exposure must cover the entire unbonding liability. |
| `WS_CHECKPOINT_MAX_AGE` | 6 completed committee epochs, with independent freshness trust. |
| `MAX_AMOUNT` | Manifest cap; strictly below $2^{120}$. |
| `SUBSIDY_SCHEDULE`, `I_MAX` | Actual published `EmissionV1` values; no default. |
| `T0`, `TMAX` | Actual genesis target values; $1 \le T_0 \le T_{\max} \le 2^{256} - 1$. |

The minimum bond $B_{\min}$ is denominated in native base units and has no document-level numerical default. Its display-coin value follows only from the denomination exponent and the explicit amount selected by the network manifest. The registry cap, position cap, maximum inputs/outputs per transaction, maximum block authorisations, operation-cost coefficients, system-work mass budget and transport frame limits must likewise be explicit finite entries in the accompanying network profile. They are not implied by the 25-block/s target. Their release validation is part of Section 10.

## D.4 Parameter serialisation and compatibility

The parameter manifest has schema version 1 and encodes the fixed reference fields in the order of Table 12, followed by the required capacity fields just listed, the activated work-profile identifier, the emission hash, genesis-target fields and the canonical operation-cost table. The implementation-level definition and test vectors for that work profile are released alongside the concrete network manifest; they are not inferred from the identifier alone. Numerical counters are `u64` unless their named field is an amount or target; amounts use `u128`, targets use 32 bytes and identifiers use 48 bytes. The operation-cost table is ordered by the defined transaction-type byte and then system-operation byte, each with a `u64` logical cost. There are no omitted entries and no duplicate type keys.

A release supplies the precise schema file with literal values and byte vectors, including the separation between scalar fields and manifest references. Its hash is $H(\text{PARAMS} \,\|\, \text{u16}(\mathit{protocol\_version}) \,\|\, \text{enc}(\mathit{parameters}))$. Startup checks that each maximum block fits all registration and execution budgets, all counters fit their widths, snapshot and payout capacity is bounded, and the monetary manifest matches genesis. Hash equality does not repair an internally invalid manifest.

An activated successor manifest states its height, predecessor parameter hash, migration identifier and all changed rules. The v1 manifest has no generic transaction that lets a certificate proposer substitute arbitrary parameters. Future governance or upgrade authorisation is a separate explicit protocol feature. Until activated, alternate profiles are test networks or incompatible software configurations, not valid mainnet state.

# E Transport authentication and operational contracts

The reference transport uses TLS 1.3 with the RFC 10024 X25519MLKEM768 group, code point `0x11EC`, with both halves mandatory [23, 22, 21]. The cipher suite is `TLS_AES_256_GCM_SHA384` (`0x1302`); a different suite is not silently substituted in this reference profile. It preserves the TLS transcript and key schedule. It does not transplant the hybrid secret into a custom Noise or QUIC handshake and assume the same argument still applies. QUIC integration, if used, must preserve the corresponding TLS 1.3 handshake and exporter semantics rather than advertise an unsupported configuration flag.

Before accepting authenticated peer application traffic, both endpoints exchange a fixed greeting inside the completed TLS channel: chain ID, protocol version, ML-DSA public key and a fresh 32-byte random challenge. Roles are initiator and responder, not sorted by a mutable network address. Let $G_i, G_r$ be the two canonical greetings. Construct

```
context = SHA384("TRAI_P2P_CONTEXT_V1" || G_i || G_r)
E = TLS-Exporter("EXPORTER-TRAIDAG-P2P-V1", context, 48)
Auth(role) = SHA384("TRAI_P2P_AUTH_V1" || chain_id
                    || u16(version) || u8(role) || context || E)
```

Each endpoint signs its role-specific digest with the ML-DSA identity in its greeting and verifies the opposite direction. The public peer ID is the domain-separated SHA-384 commitment to that public key. The accepted peer identity must match any pinned validator or sentry descriptor used to grant privileges; successful key possession by an arbitrary public peer does not make it a validator.

No consensus or private application payload is accepted before mutual application authentication. Zero-RTT application data is disabled for these message types. Session resumption does not bypass the fresh identity exchange. Unknown versions, wrong chains, missing hybrid negotiation, failed TLS checks, invalid signatures and mismatching pinned identities close the session. These are connection failures, not instructions to downgrade to X25519 alone.

The application authentication is a TRAIDAG binding layered on the standard transport. Its composition, implementation and channel-binding tests belong in the security review; RFC 10024 does not itself certify this application protocol. Both primitive randomness and signer-key storage must be isolated from untrusted peer inputs. Key rotation updates explicit descriptors without allowing an unauthenticated peer to replace a pinned identity.

Operational limits cover maximum frame length, concurrent body fetches, pending unknown-parent requests, unverified vote cache, per-peer inventory and decompression work where compression is enabled. A node checks cheap length and membership conditions before expensive cryptography. Rejected traffic cannot trigger unbounded recursive ancestor fetches or arbitrary allocations. Consensus-data availability is retried through several authenticated peers rather than one privileged source.

Miner, wallet and pool APIs return structured states and error codes. Private keys are not accepted by a remote node API for routine signing. A pool payout is an additional spend by the pool, separate from the protocol's work-reward output. A bridge that allocates extranonce ranges must ensure unique template bindings and must not represent a partial share as a full target solution. These interface contracts are implemented and tested with the node; they are not evidence of a currently shipped integration.

# F Reproducibility and revised numerical exhibits

The source package includes `recalculate.py`, generated `numerical_results.json`, plain-text plotting data and the LaTeX sources for this edition. The arithmetic script uses Python's standard library. It emits the plotted certificate sizes, ingress components, duplicate-selection efficiencies and the following focused checks: cumulative-floor allocation sums, base-point split conservation, both execution/closure event orders, saturating target edges and payment-envelope lengths. Dummy byte fixtures are explicitly not valid cryptographic transactions.

## F.1 The equal-seat lottery comparison

The source edition used a hypothetical stake-weighted lottery with equal voting seats to motivate deterministic committee selection. For independent draws with replacement, Byzantine seat count $X \sim \text{Binomial}(n, p)$ and threshold $m = \lceil n/3 \rceil$,

$$\Pr[X \geq m] = \sum_{j=m}^{n} \binom{n}{j} p^j (1-p)^{n-j}. \tag{42}$$

The recomputed comparison values are reported in Table 13.

| Seats | $p = 15\%$ | $p = 20\%$ | $p = 25\%$ | $p = 30\%$ |
|---|---|---|---|---|
| 64 | $9.50 \cdot 10^{-5}$ | $5.09 \cdot 10^{-3}$ | $5.96 \cdot 10^{-2}$ | $2.62 \cdot 10^{-1}$ |
| 128 | $1.27 \cdot 10^{-7}$ | $2.18 \cdot 10^{-4}$ | $1.82 \cdot 10^{-2}$ | $2.13 \cdot 10^{-1}$ |
| 256 | $1.04 \cdot 10^{-13}$ | $2.44 \cdot 10^{-7}$ | $1.28 \cdot 10^{-3}$ | $1.18 \cdot 10^{-1}$ |

**Table 13.** Recomputed binomial tails for the rejected equal-seat model. Exceeding a fault threshold is not the same event as completing a successful attack. The table does not describe TRAIDAG's actual weighted active committee.

These values are valid for the stated independent-seat model. A lottery without replacement, correlated draws, heterogeneous seat weight or another selection algorithm needs its own distribution. The table is a comparison calculation rather than a theorem that deterministic top-$N$ selection has no centralisation or liveness cost.

## F.2 Artifact scope and versioning

Document version 1.2.3 records the revision of the whitepaper. It does not assign an on-chain activation height, demonstrate a live network or overwrite the project's absent monetary manifest. The source archive preserves the original author's attribution and logo for this revision, includes no font files and provides the data needed to regenerate the numerical graphics. A separate change record distinguishes corrected claims, newly specified reference rules and remaining deployment artifacts.

A mainnet release adds real signing and execution vectors, a complete schema/parameter package, validated binaries and test reports under their own content hashes. Numerical agreement with this document is necessary for those exhibits but does not establish distributed safety, implementation correctness or economic robustness by itself.

# References


[1] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. https://bitcoin.org/bitcoin.pdf.

[2] Y. Sompolinsky and A. Zohar. Secure High-Rate Transaction Processing in Bitcoin. *Financial Cryptography and Data Security*, 2015. IACR ePrint 2013/881. https://eprint.iacr.org/2013/881.

[3] Y. Sompolinsky, Y. Lewenberg and A. Zohar. *SPECTRE: A Fast and Scalable Cryptocurrency Protocol*. IACR ePrint 2016/1159. https://eprint.iacr.org/2016/1159.

[4] Y. Sompolinsky, S. Wyborski and A. Zohar. PHANTOM GHOSTDAG: A Scalable Generalization of Nakamoto Consensus. *Advances in Financial Technologies*, 2021. IACR ePrint 2018/104. https://eprint.iacr.org/2018/104.

[5] Y. Sompolinsky and M. Sutton. *The DAG KNIGHT Protocol: A Parameterless Generalization of Nakamoto Consensus*. IACR ePrint 2022/1494. https://eprint.iacr.org/2022/1494.

[6] C. Dwork, N. Lynch and L. Stockmeyer. Consensus in the Presence of Partial Synchrony. *Journal of the ACM*, 35(2), 1988, pp. 288–323. DOI: 10.1145/42282.42283.

[7] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. *OSDI*, 1999. https://www.usenix.org/legacy/events/osdi99/full_papers/castro/castro.pdf.

[8] E. Buchman, J. Kwon and Z. Milosevic. *The latest gossip on BFT consensus*. arXiv:1807.04938, version 3, 22 November 2019; first submitted 2018. Algorithm 1 is the locking reference used here. https://arxiv.org/abs/1807.04938v3.

[9] G. Danezis, L. Kokoris-Kogias, A. Sonnino and A. Spiegelman. Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus. *EuroSys*, 2022. https://arxiv.org/abs/2105.11827.

[10] A. Spiegelman, N. Giridharan, A. Sonnino and L. Kokoris-Kogias. Bullshark: DAG BFT Protocols Made Practical. *ACM CCS*, 2022. https://arxiv.org/abs/2201.05677.

[11] V. Buterin and V. Griffith. *Casper the Friendly Finality Gadget*. arXiv:1710.09437, 2017. https://arxiv.org/abs/1710.09437.

[12] J. Toomim and Bitcoin Cash specification contributors. *aserti3-2d: Absolutely Scheduled Exponentially Rising Targets*. Difficulty-adjustment specification, 2020. https://github.com/bitcoincashorg/bitcoincash.org/blob/master/spec/2020-11-15-asert.md. The TRAIDAG measurement error and capacity filter are separate from this reference.

[13] M. Corallo. *BIP 152: Compact Block Relay*. 2016. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki.

[14] NIST. *Module-Lattice-Based Digital Signature Standard*. FIPS 204, August 2024. Table 2 specifies ML-DSA key and signature lengths. DOI: 10.6028/NIST.FIPS.204.

[15] NIST. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. FIPS 203, August 2024. DOI: 10.6028/NIST.FIPS.203.

[16] NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. FIPS 202, August 2015. Appendix A, Table 4 distinguishes output length from stated security strength. DOI: 10.6028/NIST.FIPS.202.

[17] NIST. *Secure Hash Standard (SHS)*. FIPS 180-4, August 2015. DOI: 10.6028/NIST.FIPS.180-4.

[18] C. Decker and R. Wattenhofer. Information Propagation in the Bitcoin Network. *IEEE P2P*, 2013. DOI: 10.1109/P2P.2013.6688704.

[19] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf and S. Capkun. On the Security and Performance of Proof of Work Blockchains. *ACM CCS*, 2016. DOI: 10.1145/2976749.2978341.

[20] G. Brassard, P. Høyer and A. Tapp. Quantum Cryptanalysis of Hash and Claw-Free Functions. *LATIN*, 1998. https://arxiv.org/abs/quant-ph/9705002.

[21] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 9846, July 2026. https://www.rfc-editor.org/info/rfc9846. The implementation pins this TLS profile and its exporter semantics.

[22] D. Stebila, S. Fluhrer and S. Gueron. *Hybrid Key Exchange in TLS 1.3*. RFC 9954, July 2026. https://www.rfc-editor.org/info/rfc9954.

[23] K. Kwiatkowski, P. Kampanakis, B. E. Westerbaan and D. Stebila. *Post-Quantum Traditional (PQ/T) Hybrid Key Agreement Mechanisms for TLS 1.3*. RFC 10024, August 2026. https://www.rfc-editor.org/info/rfc10024. Section 6 limits the security argument to the TLS transcript context.

[24] S. Josefsson and I. Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC 8032, January 2017. https://www.rfc-editor.org/info/rfc8032.

[25] M. Sutton and Kaspa Improvement Proposal contributors. *KIP-14: Crescendo Hardfork*. 2025. https://github.com/kaspanet/kips/blob/master/kip-0014.md. Parameter comparison only; no TRAIDAG benchmark is inferred from these values.

[26] V. Bagaria, S. Kannan, D. Tse, G. Fanti and P. Viswanath. Prism: Deconstructing the Blockchain to Approach Physical Limits. *ACM CCS*, 2019. https://arxiv.org/abs/1810.08092.

[27] I. Eyal, A. E. Gencer, E. G. Sirer and R. van Renesse. Bitcoin-NG: A Scalable Blockchain Protocol. *USENIX NSDI*, 2016. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eyal.

[28] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta and I. Abraham. HotStuff: BFT Consensus with Linearity and Responsiveness. *ACM PODC*, 2019. Extended paper: https://arxiv.org/abs/1803.05069.

[29] J. Teutsch and C. Reitwießner. *A Scalable Verification Solution for Blockchains*. 2019 edition. https://arxiv.org/abs/1908.04756.

[30] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg and E. W. Felten. Arbitrum: Scalable, Private Smart Contracts. *USENIX Security*, 2018. https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner.

[31] Tenrai Technologies and contributors. TenQ: GPU-Oriented Proof of Work for TRAIDAG. Technical Whitepaper v1.0, September 2026.