LP-010: QuasarSTM

4.0 — Production Spec for Quasar-Certified GPU Execution

(activation 2026-02-14)

Zach Kelling

Lux Industries

Spec freeze: February 7, 2026 | Activation: February 14, 2026

Abstract

QuasarSTM 3.0 (LP-010, activated 2025-12-25) shipped the GPU-native ordered MVCC *substrate* for Lux's Nova (linear) and Nebula (DAG) modes: lanes, wave-tick scheduler, three-tier validation skeleton, deterministic contention manager, multi-GPU sharding stub. QuasarSTM 3.0 launched with placeholder cryptographic verifiers (HMAC-keccak across all three QuasarCert lanes), 118 of the targeted 175 EVM opcodes, the lane-clock fast path and semantic-reducer Tier 3 stubbed, and the multi-GPU commit server a sketch. *QuasarSTM 4.0 closes every one of those gaps.*

This paper specifies the QuasarSTM 4.0 production spec activated on 2026-02-14. We lift Tier 1 lane-clock validation from sketch to production, ship Tier 3 semantic reducers as first-class commit primitives, introduce the ConflictSpec ABI as a fallback-safe oracle on top of speculative Block-STM, classify lanes with the NEMO LaneClass model, select the per-lane MVCC versioning mode dynamically in the spirit of Multiverse, checkpoint EVM fibers per CALL-frame for incremental repair, advance commit horizons over Nova prefixes and Nebula causal cuts, garbage-collect MVCC versions against the horizon, lay versions out as Motor-style VersionBlocks (4 to 8 inline versions per key), grow the GPU service set from 12 to 16 (adding AdaptiveSchedule, BridgeAttest, MarketAuction, FiberCheckpoint), ship real BLS12-381 / Corona Ring-LWE / Groth16 verifiers as on-device kernels, and gate the substrate behind a CSMV-style commit server scaffold with a formally specified CPU reference and a cross-backend determinism harness at $\geq 80\%$ line coverage on Apple Metal and NVIDIA CUDA. The 3.0 invariants—determinism, ordered serializability, repair monotonicity, horizon safety—are preserved verbatim.

We report measured 1024-tx end-to-end stress: 108 ms wallclock on M1 Max in 6 wave ticks (vs. 150 ms in 8 ticks for 3.0), $repair\_amplification \approx 0.42$ on the regulated-DEX mix (vs. 0.97 for 3.0), exact 120 / 120 / 16 conflict / repair / commit on the 16 same-key contention test (textbook Block-STM, unchanged from 3.0). Real-cryptography per-cert costs: 46 μs BLS pairing, 38 μs Corona share verify, 62 μs Groth16 verify, all on-device. Same input produces byte-identical roots on Metal and CUDA over the full test suite.

Contents

1 Introduction **5**

2 What changed from 3.0 **7**

3 Lane-clock Tier 1 (production) **8**
3.1 Predicate 8
3.2 Why the substrate (3.0) deferred this 8
3.3 Atomicity 9
3.4 Hot-lane telemetry 9

4 Semantic reducers Tier 3 (production) **9**
4.1 Why semantic commit 9
4.2 DeferredOp record 9
4.3 Reduction at horizon time 10
4.4 Repair amplification 10

5 ConflictSpec ABI **11**
5.1 From blind optimism to fallback-safe oracle 11
5.2 Record 11
5.3 Source priority 11
5.4 Why this is correctness-preserving 11
5.5 Predictor structure 12

6 LaneClass (NEMO) 12
6.1 Classes . . . 12
6.2 Per-class policy . . . 12
6.3 Classification sources . . . 13
6.4 Why this matters for throughput . . . 13

7 Dynamic versioning (Multiverse) 13
7.1 One mode does not fit all lanes . . . 13
7.2 Per-mode semantics . . . 14
7.3 Mode binding . . . 14
7.4 Why determinism is preserved . . . 14

8 Fiber checkpoints 14
8.1 Why incremental repair . . . 14
8.2 Checkpoint record . . . 15
8.3 Repair from checkpoint . . . 15
8.4 Cost reduction . . . 15
8.5 Storage cost . . . 16
8.6 Determinism . . . 16

9 Commit horizon 16
9.1 Horizon as a finalisation primitive . . . 16
9.2 Advance condition . . . 16
9.3 Roots emitted at horizon advance . . . 17
9.4 Horizon safety . . . 17
9.5 Why per-round, not per-tick . . . 17

10 MVCC GC 17
10.1 Why GC matters . . . 17
10.2 Reaping rule . . . 17
10.3 Implementation . . . 18
10.4 Per-round arena . . . 18
10.5 Determinism . . . 18

11 Motor VersionBlock layout 18
11.1 Why one round trip matters . . . 18
11.2 Layout . . . 19
11.3 Visibility lookup . . . 19
11.4 Spill . . . 19
11.5 RDMA / multi-GPU readiness . . . 19
11.6 Alignment with 3.0 wire compatibility . . . 20

12 A/B/M/F drain services 20
12.1 Service set growth . . . 20
12.2 AdaptiveSchedule (A) . . . 20
12.3 BridgeAttest (B) . . . 20
12.4 MarketAuction (M) . . . 21
12.5 FiberCheckpoint (F) . . . 21
12.6 Wave-tick budget . . . 21

13 Real BLS / Corona / Groth16 verifiers **21**
13.1 From placeholders to real cryptography . . . 21
13.2 BLS12-381 pairing . . . 21
13.3 Corona Ring-LWE share verification . . . 22
13.4 Groth16 over BLS12-381 . . . 22
13.5 No CPU fallback on the round path . . . 22
13.6 Verifier dispatch . . . 22
13.7 Wire format unchanged . . . 23

14 Cross-backend determinism **23**
14.1 Goal . . . 23
14.2 Sources of nondeterminism, eliminated . . . 23
14.3 The harness . . . 24
14.4 Coverage gating . . . 24
14.5 Formal CPU reference . . . 25

15 Performance (4.0 vs. 3.0) **25**
15.1 Hardware and method . . . 25
15.2 Headline numbers . . . 25
15.3 Why 4.0 is faster despite real cryptography . . . 25
15.4 Hot-key contention test . . . 26
15.5 Per-cert cost on H100 . . . 26
15.6 Same-message BLS aggregate batching (cevm v0.47.1) . . . 26
15.7 Substrate-wide Metal cutover (cevm v0.46.1) . . . 26
15.8 Stage 5b BLS pairing structural ceiling . . . 26
15.9 Scalability claim ladder . . . 27

16 Activation roadmap **27**
16.1 Activation height . . . 27
16.2 The v0.41–v0.49 deliverable train . . . 27
16.3 Migration steps for validators . . . 27
16.4 Migration steps for app developers . . . 28
16.5 Out of scope for 4.0 (5.0 research) . . . 28

17 Conclusion **29**

1 Introduction

QuasarSTM 3.0 [18] activated on Lux mainnet on 2025-12-25. It shipped the GPU-native ordered MVCC *substrate* for Lux's Nova (linear) and Nebula (DAG) execution modes: the lane abstraction, the wave-tick scheduler, the three-tier validation *skeleton*, the deterministic contention manager, and a multi-GPU sharding stub. The design statement was that STM stops being a retroactive conflict detector and becomes a scheduler-aware fabric beneath Prism frontiers and EVM fibers.

What 3.0 actually shipped. The honest assessment is that 3.0 was the substrate, not the production system. Specifically:

- The cryptographic verifiers (`verify_bls_aggregate`, `verify_corona_share`, `verify_mldsa_groth16`) were HMAC-keccak placeholders—real cryptographic verification with one-way master secrets and cross-lane domain tags, but not the real curve / lattice / SNARK primitives.
- The EVM fiber VM covered 118 of the targeted 175 opcodes—arithmetic, environment, stack manipulation, basic memory and storage, control flow, return paths—but *not* the `CALL` family, `CREATE`/`CREATE2`, `LOG`$_n$, `EXTCODE*`, `RETURNDATA*`, `TLOAD`/`TSTORE`, `MCOPY`, or `BLOBHASH`/`BLOBBASEFEE`.
- Tier 1 lane-clock validation was sketched but not the production fast path; Tier 3 semantic reducers were enumerated as `DeferredOpKind` but not committed at horizon time.
- The CSMV-style commit server and the Motor-style VersionBlock layout were design notes, not code.
- There was no formal CPU reference and no cross-backend determinism harness gating CI.

What 4.0 ships. This paper specifies QuasarSTM 4.0, the production spec activated on 2026-02-14. Every gap above is closed. Every feature is in the `cevm` substrate, exercised by the cross-backend determinism harness, and gated in CI at the LP-137 substrate-wide bar of $\geq 96\%$ line coverage on the CPU reference oracle (the byte-equivalence ground truth against which Apple Metal and NVIDIA CUDA are asserted). Specifically:

1. **Real cryptographic verifiers.** Vendored on-device kernels for BLS12-381 pairing [11], Corona Ring-LWE share verification [17], and Groth16 verification [15] over BLS12-381 against the Z-Chain verifying-key root.
2. **Full EVM coverage.** 175 opcodes including the entire `CALL` family, `CREATE`/`CREATE2`, `LOG`$_n$, `EXTCODE*`, `RETURNDATA*`, `TLOAD`/`TSTORE`, `MCOPY`, `BLOBHASH`/`BLOBBASEFEE`; SSTORE journaling with EIP-2929 [12] cold/warm and EIP-3529 [13] refund accounting.
3. **Tier 1 lane-clock fast validation.** Production lane-clock check against a per-round table in shared memory, with no per-key MVCC chain walk in the common case (§3).
4. **Tier 3 semantic reducers.** `DeferredOp` entries committed deterministically per-lane at horizon time (§4); `Add`, `Sub`, `Append`, `BalanceDelta`, `FeeAccumulate`, `OrderAppend`, `AuctionMatch`, `MintCounter`, `NonceAdvance`.
5. **ConflictSpec ABI [2]** as a fallback-safe oracle on top of speculative Block-STM (§5).
6. **NEMO LaneClass** [8] (Owned / Shared / HotShared / Commutative / Unknown) driving per-class execution policy (§6).

7. **Dynamic versioning** a la Multiverse [7] (§7): `SingleVersionFast`, `MultiVersion`, `Reducer`, `Serialized` modes selected per lane between rounds.
8. **Fiber checkpoints** per CALL-frame return for incremental repair (§8).
9. **Commit horizon** over Nova prefixes and Nebula causal cuts (§9); **MVCC GC** bound to the horizon (§10).
10. **Motor-style VersionBlock layout** [6] (§11): 4–8 inline versions per key, RDMA / multi-GPU friendly.
11. **A/B/M/F drain services** (§12): AdaptiveSchedule, BridgeAttest, MarketAuction, FiberCheckpoint—growing the GPU service set from 12 (LP-132, 3.0) to 16.
12. **Cross-backend determinism harness** (§14): same input ⇒ byte-identical roots on Metal and CUDA.
13. **CSMV [4] commit-server scaffold** with a formally specified CPU reference [21]: fibers emit read/write intents, the commit service owns MVCC mutation, the abstract semantics live in `lib/consensus/quasar/spec/`, the differential fuzzer runs in CI.

What does not ship in 4.0. Distributed multi-GPU MVCC over RDMA, operation-window validation for pathological long transactions, kernel refinement proofs against the abstract semantics, and 800G InfiniBand / GPUDirect cell topologies remain QuasarSTM 5.0 research. They are tracked in a separate research-track document and are not on the 2026-02-14 activation path.

Scope. This paper specifies QuasarSTM 4.0 frozen on 2026-02-07 and activated on 2026-02-14. It supersedes the QuasarSTM 3.0 paper [18] on every point where the two disagree; otherwise, 3.0 is normative for substrate definitions (lanes, wave-tick scheduler, three-tier validation skeleton, contention manager, multi-GPU sharding stub).

Contributions.

1. A production specification of Tier 1 lane-clock fast validation that retires the per-key MVCC walk in the common case (§3).
2. A production specification of Tier 3 semantic reducers as first-class commit primitives, with the canonical reduction schedule per lane (§4).
3. A fallback-safe ConflictSpec ABI with explicit source priority (§5) and the NEMO LaneClass classification with per-class policy (§6).
4. Dynamic per-lane versioning mode selection bound to a per-round `LaneClockEntry` (§7).
5. A production checkpoint scheme for EVM fibers (§8) that yields measured 5–30× repair-cost reduction on router and multi-hop swap workloads.
6. Commit horizon semantics, MVCC GC, Motor VersionBlock layout (§§9–11).
7. Four new GPU drain services—A/B/M/F—without breaking the 3.0 service contract (§12).
8. Replacement of HMAC-keccak placeholders with real BLS12-381, Corona Ring-LWE, and Groth16 on-device verifiers (§13).
9. A cross-backend determinism harness and a formally specified CPU reference; same input ⇒ byte-identical roots on Metal and CUDA (§14).

10. Performance measurements: 1024-tx end-to-end stress in 108 ms on M1 Max in 6 wave ticks; on contended regulated-DEX workloads, $repair\_amplification \approx 0.42$ (§15).

11. An activation roadmap: v0.41–v0.49 deliverables that landed in the 2026-02-14 release (§16).

2 What changed from 3.0

QuasarSTM 3.0 [18] is the substrate paper. QuasarSTM 4.0 is the production paper. Table 1 catalogues the differences exactly. Subsequent sections specify each 4.0 line item.

Table 1: QuasarSTM 3.0 (2025-12-25) vs. 4.0 (2026-02-14) production-feature delta.

Topic	**3.0 (substrate)**	**4.0 (production)**
EVM coverage	118 opcodes; no `CALL` family, no `CREATE`, no `LOG`, no `EXTCODE*`	**175 opcodes** including `CALL`/`CALLCODE`/`DELEGATECALL`/`STATICCALL`, `CREATE`/`CREATE2`, `LOG`$_n$, `EXTCODE*`, `RETURNDATA*`, `TLOAD`/`TSTORE`, `MCOPY`, `BLOBHASH`/`BLOBBASEFEE`
SSTORE	write-through	**journaled** with EIP-2929 [12] cold/warm + EIP-3529 [13] refunds
BLS verifier	HMAC-keccak placeholder	**real BLS12-381** [11] pairing on-device
Corona verifier	HMAC-keccak placeholder	**real Ring-LWE** [17] share verifier
Groth16 verifier	HMAC-keccak placeholder	**real Groth16** [15] on BLS12-381
Tier 1 lane-clock	sketch	**production fast path**
Tier 3 semantic reducers	enum stub	**production reducer commit**
ConflictSpec ABI	not implemented	**production** (Static / ABI / Historical / UserDeclared / Precompile / Learned)
LaneClass	not implemented	**NEMO** [8] (Owned / Shared / HotShared / Commutative / Unknown)
Versioning mode	always-on multi-version	**dynamic per-lane** [7]
Fiber checkpoints	not implemented	**production**, per CALL-frame
Commit horizon	sketch	**production** (Nova prefix / Nebula cut)
MVCC GC	not implemented	**production**, horizon-driven
Motor VersionBlock [6]	not implemented	**production** (4–8 inline versions per key)
GPU services	12 (LP-132 3.0)	**16** (12 + A/B/M/F)
CSMV [4] commit server	not implemented	**scaffold landed**, single-GPU production
Formal CPU reference [21]	not implemented	**small executable semantics** + differential fuzzer
Cross-backend determinism	per-backend tests	**single harness**, byte-identical roots Metal = CUDA
Coverage	informal	$\geq$ **96%** **line coverage** on the CPU reference oracle (LP-137-COVERAGE.md)

What did not change. The 3.0 invariants—determinism, ordered serializability, repair monotonicity, horizon safety—are preserved verbatim. The wire format (`QuasarRoundDescriptor`, `QuasarRoundResult`) is byte-for-byte unchanged. The public host API (`QuasarGPUEngine` [19]) is unchanged. The canonical-order $\preceq_{\text{can}}$ is unchanged: 4.0 reads 3.0-emitted round artifacts and emits 4.0 round artifacts with the extra `hint_roots` [3] riding in the existing `roots` slot. There is no on-disk state migration. Validators running 3.0 after 2026-02-14 are rejected at the consensus engine and must upgrade.

Backward / forward compatibility.

- `ServiceId.Count` flips from 12 to 16 at the activation height; old IDs unchanged.
- `VersionBlock.MAX_INLINE_VERSIONS` flips from 4 to 8; 4.0 reads 3.0 blocks, 3.0 cannot read 4.0 blocks (forward only).
- The HMAC-keccak verifier path is preserved as a development-only mode under `EVM_DEV_HMAC_VERIFIER=1` for deterministic test vectors. It is rejected on production networks.
- Pre-activation rounds remain valid as historical 3.0 rounds; the activation height is the partition.

3 Lane-clock Tier 1 (production)

3.1 Predicate

QuasarSTM 3.0 sketched a lane clock per lane [18]. 4.0 ships it as the production fast validation predicate. A transaction t with reads $R_t \subseteq \mathit{Lanes}$ and writes $W_t \subseteq \mathit{Lanes}$ is Tier-1 valid iff every lane it read shows the same *lane clock* at validate time as it did at execute time, and no canonical-order predecessor that has not yet committed has a write to a lane in R_t:

$$\text{Tier1}(t) \iff \bigwedge_{l \in R_t} lc^{\text{val}}(l) = lc_t^{\text{exec}}(l) \quad \wedge \quad \nexists\, t' \prec_{\text{can}} t : t' \in W^{-1}(R_t) \wedge \neg\text{committed}(t')$$

The implementation is an $O(|R_t|)$ table lookup against a per-round shared-memory `LaneClockEntry` table:

```
struct LaneClockEntry {
    Hash lane_id;
    uint64_t clock;
    uint32_t last_writer_tx; // canonical-order index
    uint8_t versioning_mode; // see VersioningMode
    uint8_t lane_class; // see LaneClass
    uint16_t pad;
};
```

When Tier 1 passes, t commits without consulting the per-key version chain. Tier 2 (key-MVCC) and Tier 3 (semantic reducers) only fire when Tier 1 disagrees, which the production telemetry shows happens in under 5% of transactions on the regulated-DEX workload mix.

3.2 Why the substrate (3.0) deferred this

Tier 1 in 3.0 was a sketch because the lane-clock table needed: (a) deterministic insertion order so that the table contents at validate time match across backends; (b) atomic clock bump on write commit; (c) per-round arena reset semantics tying the table lifetime to the QuasarRoundDescriptor lifetime. 4.0 ships all three. The deterministic insertion order is fixed by the canonical lane ID enumeration order:

$$lane\_id = H(domain, contract, account, asset, market, nonce\_lane, storage\_prefix)$$

(unchanged from 3.0 [18]). Lane IDs are sorted at round start and indexed; the table is keyed by sorted index, not by hash.

3.3 Atomicity

Clock bumps use a single atomic fetch-add on the table slot's `clock` field, with the `last_writer_tx` and `versioning_mode` fields written under the same memory fence. This is the only contended atomic on the lane-clock fast path; the rest of the validation predicate is read-only.

Lemma 3.1 (Tier-1 monotonicity)**.** *For any lane l and any pair of executions of the same canonical order, the sequence of* `clock` *values written to the* `LaneClockEntry` *for l is identical.*

Proof sketch. Every clock bump corresponds to a write commit in canonical order. The canonical order is fixed by Quasar consensus. Hence the bump sequence is fixed. The atomic fetch-add returns the bumped value; the value is deterministic in the bump count. □

Lemma 3.1 is what makes byte-identical roots possible across Metal and CUDA backends (§14). The lane clock is the most-touched piece of shared metadata in the round; if its bumps were nondeterministic, every downstream root would diverge.

3.4 Hot-lane telemetry

Per-round, every lane records the count of Tier-1 mismatches (`tier1_mismatch_count`). When this exceeds `HOT_LANE_CONFLICT_THRESHOLD` (default 16), the lane is promoted to `HotShared` class for the *next* round (§6, §7). In-round mode is fixed—4.0 does not allow within-round class changes, because they would break determinism with respect to ordering decisions already made.

4 Semantic reducers Tier 3 (production)

4.1 Why semantic commit

Block-STM [14] treats every write as a same-key SSTORE; under contention, the abort-and-repair amplification ratio $repair\_amplification = repair\_count / committed\_count$ explodes on workloads where the contention is structurally commutative. The canonical example is a fee accumulator: 1024 concurrent transactions each adding to `fee` produce 1023 same-key SSTORE conflicts under unaugmented Block-STM, yet *the result is the sum*, which is canonical regardless of execution order.

QuasarSTM 3.0 enumerated this insight as `DeferredOpKind` [18] but did not commit deferred ops at horizon time. 4.0 ships the production reducer commit [9].

4.2 DeferredOp record

```
enum class DeferredOpKind : uint8_t {
    Add = 0,
    Sub = 1,
    Append = 2,
    BalanceDelta = 3,
    FeeAccumulate = 4,
    OrderAppend = 5,
    AuctionMatch = 6,
    MintCounter = 7,
```

```
    NonceAdvance = 8,
};

struct DeferredOp {
    uint32_t tx_id; // canonical-order index of the issuer
    DeferredOpKind kind;
    Hash lane_id; // target Commutative lane
    uint8_t payload[48]; // value / delta / element / etc.
};
```

A transaction t executing in `Reducer`-mode lane l does *not* write the lane through MVCC. It pushes a `DeferredOp` into the per-lane reducer queue and continues. Validation against l's clock is skipped in Tier 1 (which is one of the reasons Tier 1 is so cheap on commutative lanes).

4.3 Reduction at horizon time

When the commit horizon (§9) advances over a canonical-order range $[t_a, t_b]$, every lane l in `Reducer` mode runs `deterministic_reduce_at_commit(`l`)`. The reduction order is canonical order over the issuer `tx_id`:

Algorithm 1 `deterministic_reduce_at_commit(`l`)`

1: sort Q_l by `tx_id` ascending
2: $v \leftarrow lane\_state(l)$
3: **for** each op $\in Q_l$ in order **do**
4: $\quad v \leftarrow \text{apply}(\text{op}, v)$
5: **end for**
6: write v back to lane state
7: emit per-op receipt entries in the same order
8: clear Q_l

`apply` is a per-kind pure function:

- `Add(x), Sub(x), FeeAccumulate(x), MintCounter(x)`: $v \mathrel{+}= x$ (mod field), or $v \mathrel{-}= x$.
- `Append(elem), OrderAppend(elem)`: $v ::$ elem (deterministic list append).
- `BalanceDelta(account, delta)`: $balance_{account} \mathrel{+}=$ delta.
- `AuctionMatch`: deterministic order book matching, applied as a single batched step at end-of-batch.
- `NonceAdvance`: per-account nonce monotone advance, with conflict aborting only the duplicate-nonce loser per canonical order.

4.4 Repair amplification

On the regulated-DEX workload mix (order placements, cancels, fee accumulation, balance deltas, auction matches), 4.0 measures $repair\_amplification \approx 0.42$, vs. ≈ 0.97 for 3.0 on the same trace. The improvement is not because 4.0 is faster at repair; it is because reducer commit removes most of the contended writes from the MVCC validation path entirely.

Remark 4.1 (Reducers do not break determinism)**.** *The reduction order is canonical order over the issuer* `tx_id`*.* `tx_id` *is fixed by Quasar consensus ([16, 20]). Hence the reduced lane state is deterministic in the round inputs. This is the same argument as Lemma 3.1.*

5 ConflictSpec ABI

5.1 From blind optimism to fallback-safe oracle

Block-STM is blind optimism by default—it executes optimistically and discovers contention retroactively. The AFT 2025 conflict-specifications work [2] reports significant speedups when declared access information is available before execution, with Block-STM as the correctness backstop. QuasarSTM 4.0 ships this as the production scheduler oracle.

5.2 Record

```
struct ConflictSpec {
    uint32_t tx_id;
    uint32_t read_lane_offset;
    uint16_t read_lane_count;
    uint32_t write_lane_offset;
    uint16_t write_lane_count;
    uint32_t commutative_lane_offset;
    uint16_t commutative_lane_count;
    uint8_t confidence; // 0..255
    uint8_t source; // ConflictSpecSource
};

enum class ConflictSpecSource : uint8_t {
    Static = 0, // compile-time ABI declaration
    ABI = 1, // EVM ABI hint (Solidity attribute)
    Historical = 2, // observed past traces
    UserDeclared = 3, // signed declaration from sender
    Precompile = 4, // precompile self-declaration
    Learned = 5, // device-resident predictor (ForeSight-style)
};
```

The `(read|write|commutative)_lane_offset, _count` pairs index into a per-round dense `lane_id[]` array stored in the `items_arena` of the AdaptiveSchedule ring (§12).

5.3 Source priority

The scheduler consults sources in fixed precedence:

Static $\succ$ ABI $\succ$ UserDeclared $\succ$ Precompile $\succ$ Historical $\succ$ Learned $\succ$ fallback Block-STM

Concretely, the AdaptiveSchedule drain produces the highest-priority `ConflictSpec` available for each transaction; if none is available, the transaction enters Block-STM speculative execution exactly as in 3.0 [18].

5.4 Why this is correctness-preserving

ConflictSpec is *advice*, not authority. The validation tiers (Tier 1 lane-clock, Tier 2 key-MVCC, Tier 3 semantic reducer) execute unchanged. A mis-declared `ConflictSpec` loses speed (the speculator schedules suboptimally) but cannot break safety: the underlying validators detect the missed read or write and trigger repair [2].

Invariant 5.1 (ConflictSpec safety)**.** *For any transaction t scheduled with $\mathit{ConflictSpec}_t = c$, the commit of t requires Tier 1 / 2 / 3 validation against t's* actual *read/write set, not c.*

In practice the predictor [5] and historical-trace sources are tuned to over-declare reads and writes (false-positive bias) to avoid the missed-conflict path. The Learned source's `confidence` field is published in the round result; validators that consistently fall back to speculative STM emit a hint root (§12) that other validators can ingest as a `Historical` source for subsequent rounds [3].

5.5 Predictor structure

```
struct LanePredictorEntry {
    Hash code_hash;
    uint32_t selector; // 4-byte function selector
    Hash caller_class;
    Hash market_id;
    Hash predicted_lanes_root; // commitment to declared lanes
    uint16_t confidence; // 0..1000
    uint16_t observed_conflict_rate;
};
```

The predictor is device-resident and updated at end-of-round from the observed conflict matrix. Updates are deterministic in the round inputs (canonical-order trace of conflicts and commits), so two validators running 4.0 against the same canonical order maintain byte-identical predictor state. This is required for cross-validator hint-root agreement [3].

6 LaneClass (NEMO)

6.1 Classes

NEMO [8] observes that, post-consensus, the bottleneck is execution and that blockchain workloads have heavy access skew. The prescription is to classify state by ownership and contention pattern, and to drive validation policy from the class:

```
enum class LaneClass : uint8_t {
    Owned = 0, // account-local / nonce-local / private state
    Shared = 1, // shared contract state, ordinary contention
    HotShared = 2, // AMM reserves, order-book level, fee counter
    Commutative = 3, // additive / append / reducer lane
    Unknown = 4, // fall through to ordinary speculative path
};
```

6.2 Per-class policy

Table 2 shows the production policy mapping.

Table 2: LaneClass execution and validation policy (4.0 production).

Class	Validation policy	Versioning mode
`Owned`	no validation against unrelated txs; fast commit if owner / nonce match	`SingleVersionFast`
`Shared`	ordered MVCC (Tier 2)	`MultiVersion`
`HotShared`	semantic reducer / serialized lane / pre-compile path	`Reducer` or `Serialized`
`Commutative`	Tier 3 reducer commit (§4)	`Reducer`
`Unknown`	ordinary speculative Block-STM (Tier 1 + Tier 2) per 3.0 [18]	`MultiVersion`

6.3 Classification sources

Per round, every lane has a class. Classes are derived from:

1. **Static ABI annotations** on contracts (e.g. a Solidity `lane_class("Commutative")` attribute on a state variable).
2. **Precompile self-declaration**: precompiles declare the lane class of every state variable they touch. The DEX matching precompile, for example, declares its order book level lanes `HotShared`.
3. **Hot-lane telemetry promotion**: a lane that exceeds `HOT_LANE_CONFLICT_THRESHOLD` mismatches in round n is promoted to `HotShared` for round $n + 1$ (§3).
4. **Predictor** [5]: classified by observed-conflict-rate quantile.
5. **Default**: `Unknown`—falls through to speculative Block-STM, exactly as in 3.0.

Classification is finalised at the start of the round and frozen for the round's duration.

6.4 Why this matters for throughput

NEMO's central observation is that the throughput ceiling is bounded by the share of `Owned` and `Commutative` traffic. Skewed regulated-DEX workloads (per-account orders, fee accumulation, balance deltas) have a heavy `Owned` and `Commutative` tail, so Tier 1 fast path and Tier 3 reducer commit each absorb a large fraction of the load:

- `Owned`: Tier 1 passes always (no cross-tx validation) ⇒ commits in O(1).
- `Commutative`: Tier 3 reducer commit at horizon ⇒ no MVCC contention.
- `HotShared`: routed to a serialized lane or to a reducer-mode lane ⇒ no speculative repair storm.
- `Shared` and `Unknown`: ordinary Block-STM ⇒ inherits 3.0 behaviour [18].

7 Dynamic versioning (Multiverse)

7.1 One mode does not fit all lanes

Multiverse [7] observes that always-on multi-version concurrency control is expensive in low-contention regions (unnecessary version chain traffic) and unsafe in pathological hot regions (repair storms). The remedy is per-region mode selection. QuasarSTM 4.0 adopts the same prescription per lane:

```
enum class VersioningMode : uint8_t {
    SingleVersionFast = 0, // low contention; no MVCC chain
    MultiVersion = 1, // standard ordered MVCC (3.0 default)
    Reducer = 2, // commutative fast path (Tier 3)
    Serialized = 3, // pathological hot lane: strict canonical order
};
```

7.2 Per-mode semantics

`SingleVersionFast`
: No version chain. Writes overwrite. Validation is the lane-clock check (§3). Used for `Owned` lanes by default.

`MultiVersion`
: Standard ordered MVCC over the Motor-style VersionBlock (§11). Validation is Tier 1 lane clock + Tier 2 key-MVCC. Default for `Shared` and `Unknown`.

`Reducer`
: Writes are pushed as `DeferredOp`s into the per-lane reducer queue (§4); validation skips Tier 2 entirely; commit happens at horizon time. Used for `Commutative` and most `HotShared`.

`Serialized`
: Writes execute strictly in canonical order. Concurrent readers see the latest committed value but the writer-ordering is enforced. Reserved for pathological hot lanes where reducer composition is impossible (e.g. an unmerged multi-AMM router ledger).

7.3 Mode binding

The mode is bound per-lane in the `LaneClockEntry` (§3) at round start and is fixed for the duration of the round. Mode changes are made between rounds based on hot-lane telemetry, ConflictSpec hints, and LaneClass updates. The decision function is deterministic in the per-round telemetry, so two validators running 4.0 against the same canonical-order trace bind identical modes.

Table 3: Default mode binding by LaneClass (§6).

LaneClass	Default mode
`Owned`	`SingleVersionFast`
`Shared`	`MultiVersion`
`HotShared`	`Reducer`, escalating to `Serialized`
`Commutative`	`Reducer`
`Unknown`	`MultiVersion`

7.4 Why determinism is preserved

The mode binding is published in the per-round telemetry roots (§12) and is part of the cert subject when mode-binding divergence would otherwise be possible. Two validators that disagree on the mode binding cannot agree on the round's roots; the consensus engine surfaces the disagreement as a fork. In practice, the binding is deterministic in the round inputs and there is no disagreement.

Remark 7.1 (Mode is round-scoped)**.** *The mode of a lane in round n is a function of telemetry and classification observed during round $n-1$. There is no within-round mode change. This is what makes the MVCC VersionBlock layout and the reducer queue layout per-round-arena allocations rather than per-key allocations.*

8 Fiber checkpoints

8.1 Why incremental repair

Repairs in 3.0 [18] re-execute the entire transaction from `pc=0`. For long-running transactions—hot routers, multi-hop swaps, batched DEX matches—this is wasteful: the conflicting read often

happens deep into the transaction, after many side-effect-free arithmetic and stack operations. 4.0 ships checkpoint-based incremental repair as the production path.

8.2 Checkpoint record

```
struct FiberCheckpoint {
    uint32_t fiber_id; // index in the fiber pool
    uint32_t pc; // EVM program counter at checkpoint
    uint64_t gas_remaining;
    uint16_t stack_depth;
    uint16_t mem_size;
    uint8_t call_depth; // CALL frames opened at checkpoint
    uint8_t pad[3];
    Hash state_digest; // commitment over MVCC reads since start
    uint32_t rwset_offset; // index into rwset_arena
    uint16_t rwset_count;
    uint16_t pad2;
};
```

A checkpoint is emitted at every `CALL`-frame return, every explicit checkpoint opcode (`CKPT` pseudo-opcode for precompiles), and every cold-state suspend (§§7.4 of 3.0 [18]). Each fiber maintains a ring of the last N checkpoints (default $N = 4$); older checkpoints are recycled into the per-round arena.

8.3 Repair from checkpoint

When Tier 1 / Tier 2 validation fails for a transaction that has emitted checkpoints, repair restores the latest checkpoint c such that $c.pc$ is strictly less than the program counter of the conflicting read, then re-executes from $c.pc$. The fiber's stack, memory, gas, and call-depth are restored from c; the read/write set since c is truncated.

Algorithm 2 `repair_from_checkpoint(`t`)`

1: $\text{pc}^* \leftarrow$ program counter of conflicting read in t
2: $c \leftarrow$ latest checkpoint with $c.pc < \text{pc}^*$
3: **if** no such c **then**
4: **return** `repair_from_start(`t`)` ▷ 3.0 path
5: **end if**
6: restore $stack$, mem, gas, $call\_depth$ from c
7: truncate $t.rwset$ to entries $\leq c.rwset\_offset$
8: bump $t.incarnation$
9: enqueue t at `ServiceId::Exec` with $pc \leftarrow c.pc$

8.4 Cost reduction

Measured on the cevm router workload (10-leg multi-hop swap, 1024 concurrent transactions, single hot pool):

- 3.0 (full re-execute): mean repair cost $\approx 240\,\mu$s per repair.
- 4.0 (checkpoint-based): mean repair cost $\approx$ 9–48 μs per repair, depending on which CALL-frame the conflict landed in. **5–30× reduction.**

8.5 Storage cost

Each checkpoint costs ≈ 96B of metadata plus a snapshot of the live MVCC read set since the previous checkpoint. With the ring size $N = 4$ and a typical fiber accumulating ≤ 8 MVCC reads per checkpoint window, total per-fiber checkpoint cost is ≈ 1kB, allocated from the per-round arena and reset on `end_round()`.

8.6 Determinism

The checkpoint emit set is a function of the transaction's bytecode and the canonical-order schedule (`CALL`-frame returns are deterministic in the EVM trace). Two validators executing the same canonical order emit the same checkpoint sequence per fiber, so the *repair-from-checkpoint* restoration is deterministic.

Lemma 8.1 (Checkpoint repair preserves determinism)**.** *For any transaction t that fails Tier 1 / Tier 2 validation in a canonical-order trace, `repair_from_checkpoint`(t) yields the same final committed state as `repair_from_start`(t).*

Proof sketch. The checkpointed state $(stack, mem, gas, call\_depth)$ is the value t had at $pc = c.pc$ during its first execution. Re-execution from $c.pc$ proceeds through the same instruction sequence as the suffix $[c.pc, end]$ of the original execution—except that the conflicting read now sees the new visible version. The MVCC visibility rule under canonical order (3.0 § 5) is the same; the new visible version is canonical; hence the repaired result is the same as a from-scratch re-execution that observes the same visible versions for all reads. □

9 Commit horizon

9.1 Horizon as a finalisation primitive

QuasarSTM 3.0 sketched the commit horizon: a contiguous Nova prefix or a valid Nebula causal cut that can be finalised in one batch instead of transaction-by-transaction [18]. 4.0 ships the production semantics.

```
struct CommitHorizon {
    uint64_t prefix_len; // Nova: number of canonical-order txs committed
    Hash causal_cut_root; // Nebula: commitment over the cut frontier
    Hash reducer_state_root; // post-reduce snapshot for all Reducer lanes
    uint64_t round_index; // Quasar round this horizon belongs to
    uint64_t advanced_at_tick; // wave-tick index where this horizon advanced
    uint8_t mode; // 0=Nova, 1=Nebula
    uint8_t pad[7];
};
```

9.2 Advance condition

A horizon advance over canonical-order range $[t_a, t_b]$ requires:

1. Every $t \in [t_a, t_b]$ has passed Tier 1 / Tier 2 / Tier 3 (as applicable).
2. No repair is outstanding for any t in the range.
3. Every `Reducer`-mode lane touched in the range has run `deterministic_reduce_at_commit` (§4).
4. For Nebula, the cut frontier is a valid antichain in the DAG.

When the four conditions hold, the horizon advances to t_b, the roots are computed once, and the per-tx receipts are emitted. Per-tx roots are not emitted; this is what slashes per-transaction certificate overhead.

9.3 Roots emitted at horizon advance

At each horizon advance, the round-result roots (`block_hash`, `state_root`, `receipts_root`, `execution_root`, `mode_root`) are recomputed and the new `hint_roots` [3] are appended:

$$hint\_roots = \langle access\_spec\_root,\ dependency\_graph\_root,\ hot\_lane\_root,\ conflict\_profile\_root,\ reducer\_$$

The `hint_roots` ride in the existing `roots` slot of `QuasarRoundResult`—no wire-format change. They are non-load-bearing: replay validators consume them to schedule faster, but safety does not depend on them [3].

9.4 Horizon safety

Theorem 9.1 (Horizon safety)**.** *For any commit horizon H advanced from t_{a-1} to t_b in a Quasar round, the post-horizon state is canonically equivalent to the sequential execution of $t_a, t_{a+1}, \ldots, t_b$ in canonical order, modulo the reduction rule for `Reducer`-mode lanes (§4).*

Proof sketch. For $t \in [t_a, t_b]$ in `SingleVersionFast` or `MultiVersion`, Tier 1 and Tier 2 validation guarantee that t's reads see the canonical visible version (3.0 § 5, [18, 1]). For `Reducer`-mode lanes, Algorithm 1 applies operations in canonical order over `tx_id`, which is the same order as sequential execution. `Serialized` mode trivially executes writes in canonical order. Hence the post-horizon state matches a sequential execution that handles `Reducer` lanes by applying their operations in tx-id order. □

9.5 Why per-round, not per-tick

A round may advance the horizon multiple times within its wave-tick budget. The horizon is per-round; each advance emits one set of roots. The certificate (LP-020) is emitted at *round* end and binds *the final horizon* of the round, not intermediate horizons. Intermediate horizons are convenient checkpoints for replay validators but are not certified.

Remark 9.1 (Nova vs. Nebula horizon)**.** *In Nova mode, the horizon is always a prefix of the canonical order ($t_b - t_{a-1}$ contiguous transactions). In Nebula mode, the horizon is a causal cut (an antichain in the DAG); the canonical order is extended by the deterministic tie-break rule from 3.0 [18] so that reduction order over `tx_id` remains well-defined.*

10 MVCC GC

10.1 Why GC matters

Without GC, a long-lived MVCC arena accumulates versions indefinitely and Tier 2 validation cost grows linearly in version count. QuasarSTM 3.0 deferred GC to a separate post-round host pass; 4.0 moves it on-device, bound to the commit horizon.

10.2 Reaping rule

Versions are reapable when:

1. their `commit_round` is strictly earlier than the current horizon's `round_index`, and

2. no in-flight repair could re-read them (i.e. no transaction in the active round has a checkpoint or read-set entry pointing at the version).

The second condition is satisfied for every version older than the *previous-round horizon* because checkpoints and read-set entries do not survive `end_round()`. So GC reaps versions older than the previous-round horizon at every `begin_round`.

10.3 Implementation

```
void mvcc_gc(VersionBlock* vb, uint64_t horizon_round) {
    uint16_t keep = 0;
    for (uint16_t i = 0; i < vb->count; ++i) {
        if (vb->versions[i].commit_round >= horizon_round - 1) {
            vb->versions[keep++] = vb->versions[i];
        }
    }
    vb->count = keep;
}
```

GC runs at `begin_round` on every VersionBlock that survived the previous round, in parallel across the GPU. It is a single pass over the inline version array (length ≤ 8, §11), no chain walk.

10.4 Per-round arena

Per-round allocations—DeferredOps, fiber checkpoints, dense lane-id arrays, ConflictSpec records—live in a per-round bump allocator arena that is reset on `end_round`. There is no `free()` on the round path. Inter-round survivors (`LaneClockEntry`, `VersionBlock`, predictor state) live in dedicated long-lived arenas with their own GC rules.

10.5 Determinism

GC is deterministic in the per-round inputs because the horizon is. The reaping decision for any specific version is a function of `commit_round` and the horizon's `round_index`, both of which are agreed across validators. So two validators running 4.0 against the same canonical-order trace agree on the surviving version set after GC.

Lemma 10.1 (GC determinism)**.** *For any two validators executing the same canonical-order trace, the set of versions surviving GC after round n is identical.*

Proof sketch. Each version's `commit_round` field is set deterministically at commit time. The horizon at end of round n is deterministic by Theorem 9.1. The reaping condition is a pure function of these two. Hence the surviving set is identical. □

11 Motor VersionBlock layout

11.1 Why one round trip matters

Motor [6] (OSDI 2024) demonstrated that, on disaggregated memory over RDMA, MVCC reads dominated by linked-chain pointer-chases across the network become the throughput bottleneck. The remedy is to lay versions out as consecutive tuples so that the likely-visible read fetches in *one* round trip. The same reasoning applies inside a GPU memory hierarchy: chained MVCC reads stall on uncoalesced loads, costing memory bandwidth and L1 hit rate.

QuasarSTM 3.0 used a chain layout [18]. 4.0 ships the Motor VersionBlock layout.

11.2 Layout

```
constexpr int MAX_INLINE_VERSIONS = 8; // 4 in 3.0; 8 in 4.0

struct StmVersion {
    uint32_t writer_tx; // canonical-order index of writer
    uint32_t commit_round;
    uint64_t value_lo; // 256-bit value, low limbs
    uint64_t value_hi; // 256-bit value, high limbs
    uint64_t pad[2]; // 64 bytes total
};

struct VersionBlock {
    Hash key; // storage slot key (32 B)
    uint16_t count; // populated versions, 0..MAX_INLINE_VERSIONS
    uint16_t capacity; // == MAX_INLINE_VERSIONS
    uint8_t pad[28]; // align versions to 64 B
    StmVersion versions[MAX_INLINE_VERSIONS]; // 8 * 64 = 512 B
};
```

A VersionBlock fits in a single 576-byte cache-line-multiple region; on Apple Metal the device shared-memory page is 1024 B and on NVIDIA H100 the L1 line is 128 B with 4-line coalesced loads, so a single load instruction reaches at least the first 4 versions in either case.

11.3 Visibility lookup

Tier 2 visibility lookup walks the inline array linearly:

Algorithm 3 `visible_version`(vb, t)

```
1: v* ← ⊥
2: for i ← 0 to vb.count − 1 do
3:     if vb.versions[i].writer_tx ≺_can t and committed(vb.versions[i]) then
4:         if v* = ⊥ or vb.versions[i].writer_tx ≻_can v*.writer_tx then
5:             v* ← vb.versions[i]
6:         end if
7:     end if
8: end for
9: return v*
```

The loop is unrolled (compile-time fixed bound 8) and runs in shared memory with no pointer chasing. On the regulated-DEX workload mix, the average key has 1–2 visible versions; the inline-8 layout almost always avoids the spill case.

11.4 Spill

If a key accumulates more than 8 versions within a round (rare, since GC reaps last-round versions at `begin_round`, see §10), the VersionBlock spills into a per-key overflow chain in the long-lived arena. The spill path is the chain layout from 3.0; correctness is unchanged.

11.5 RDMA / multi-GPU readiness

The VersionBlock layout is the unit of message between fiber clients and the CSMV [4] commit server (§ 16) in the 4.0 single-GPU production deployment, and it remains the unit of message in the 5.0 multi-GPU research deployment. By making the layout RDMA-friendly today, the 4.0 substrate is ready for the multi-GPU commit-server pattern without a layout-layer rewrite.

11.6 Alignment with 3.0 wire compatibility

VersionBlock is an *on-device* structure. The wire format (`QuasarRoundDescriptor`, `QuasarRoundResult`) does not expose VersionBlock layout to validators. So the 3.0 → 4.0 inline-version count change is forward-compatible at the consensus layer: 3.0 validators that are upgraded read 4.0 blocks; 4.0 reads 3.0 blocks (capacity is metadata, not in the cert).

12 A/B/M/F drain services

12.1 Service set growth

LP-132 [19] (3.0 substrate) defined 12 GPU services. QuasarSTM 4.0 grows the set to 16 by adding four new drains, each with a dedicated device-resident ring and the same wave-tick budget contract as the 3.0 services. The 16-service enum is the single normative source:

```
enum class ServiceId : uint32_t {
    // 3.0 substrate (12)
    Ingress = 0,
    Decode = 1,
    Crypto = 2,
    DagReady = 3,
    Exec = 4,
    Validate = 5,
    Repair = 6,
    Commit = 7,
    StateRequest = 8,
    StateResp = 9,
    CertLane = 10,
    CertOut = 11,

    // 4.0 production (4 new) -- A/B/M/F
    AdaptiveSchedule = 12,
    BridgeAttest = 13,
    MarketAuction = 14,
    FiberCheckpoint = 15,

    Count = 16,
};
```

12.2 AdaptiveSchedule (A)

Inputs: ConflictSpec records (§5), LaneClass classifications (§6), predictor entries. **Outputs:** per-tx scheduling decision routed to `Exec` (with declared lane sets attached) or to the speculative fallback path.

The drain consults the source-priority order (§5) and emits the highest-priority `ConflictSpec` per transaction. It also publishes the per-tx predictor confidence so the validator can attribute later repair events to predictor mis-prediction.

12.3 BridgeAttest (B)

Inputs: cross-chain attestation messages from the omnichain bridge (LP-016 / LP-017). **Outputs:** per-message verifier result onto the `Commit` ring.

BridgeAttest is the GPU-side termination of the cross-chain attestation path: messages arrive as bundles of source-chain commitments and threshold signatures, and the drain runs the same on-device BLS / Corona / Groth16 verifiers (§13) against the cross-chain VK published in the BridgeRoundDescriptor extension.

This is what makes 4.0 capable of executing bridge commits in the same wave-tick budget as ordinary EVM, rather than spilling them to a separate host-side path.

12.4 MarketAuction (M)

Inputs: order-append `DeferredOps` (§4) and cancel ops within an auction batch window. **Outputs:** `AuctionMatch` entries that reduce to the batch-cleared book state at horizon time.

MarketAuction is the precompile-level batched matching path for hot DEX lanes. It reduces an entire batch of order/cancel events into a single match-and-clear transformation, eliminating the storm of SSTORE conflicts that would otherwise hit the order book level lanes. The output is a single `AuctionMatch` reducer entry per market per round.

12.5 FiberCheckpoint (F)

Inputs: fiber state at `CALL`-frame return, explicit checkpoint opcode, or cold-state suspend (§8). **Outputs:** `FiberCheckpoint` records into the fiber's per-fiber checkpoint ring.

This drain owns checkpoint emission and GC. It runs at `CALL`-frame boundaries and at end-of-round to recycle older checkpoints back into the per-round arena.

12.6 Wave-tick budget

Each new drain has a fixed per-tick budget enforced by the wave-tick kernel. The budgets are tuned per backend; the cross-backend determinism harness (§14) verifies that any budget setting that produces output also produces byte-identical outputs across Metal and CUDA.

Table 4: Per-tick budgets for the four new 4.0 services (default; tunable).

Service	Budget per tick
`AdaptiveSchedule`	4096 ConflictSpec admissions
`BridgeAttest`	256 attestation verifications
`MarketAuction`	64 markets per auction tick
`FiberCheckpoint`	4096 checkpoint emits + GC events

13 Real BLS / Corona / Groth16 verifiers

13.1 From placeholders to real cryptography

QuasarSTM 3.0 shipped HMAC-keccak placeholder verifiers [18] for the three cert lanes (BLS, Corona, MLDSAGroth16, see LP-020). These placeholders performed genuine cryptographic verification with one-way master secrets and cross-lane domain tags that reject replay, but they were not the real curve, lattice, or SNARK primitives. The 3.0 substrate was structured so that the swap to real primitives was a single function-pointer change. QuasarSTM 4.0 makes those swaps. From the 2026-02-14 activation, the cert verifiers are real.

13.2 BLS12-381 pairing

BLS12-381 [11] is the pairing-friendly curve used across ETH2, Filecoin, Aptos, and Lux for aggregate signature verification. The on-device verifier is a vendored, GPU-resident implementation of the optimal ate pairing. The verification predicate is the standard aggregate-signature pairing equation:

$$e(aggSig, G_2) \stackrel{?}{=} e(H(subject), aggPK)$$

Substrate location. `lib/consensus/quasar/gpu/crypto/bls12_381.metal` (Metal) and `lib/consensus/quasar/gpu/crypto/bls12_381.cu` (CUDA). The implementations share field arithmetic kernels and differ only in shader-language plumbing.

Per-cert cost. On Apple M1 Max, $\approx 46\,\mu$s per aggregate verification. On NVIDIA H100, $\approx 31\,\mu$s. Both include the 256-bit Hash-to-curve and the final exponentiation.

Aggregate. The verifier accepts an aggregate signature over n signers; the bitmap of signer indices is committed in `bitmap_hash` so the round result binds the exact signer set.

13.3 Corona Ring-LWE share verification

Corona [17] is the Ring-LWE 2-round threshold signature scheme defined in LP-104. The Q-Chain runs the DKG ceremony per epoch and commits the result to `qchain_ceremony_root` in the QuasarRoundDescriptor; the verifier uses this as the public key selector. The verification predicate is the share-validation rule from the Corona spec.

Substrate location. `lib/consensus/quasar/gpu/crypto/corona.metal` (Metal) and `lib/consensus/qua` (CUDA). Ring arithmetic kernels are shared.

Per-share cost. On Apple M1 Max, $\approx 38\,\mu$s per share verification. On NVIDIA H100, $\approx 24\,\mu$s.

Threshold. A round emits a Corona cert when the sum of verified-share weights crosses the threshold for the active epoch (epoch-defined t-of-n). The drain accumulates share weights into a per-round counter and emits the cert when the counter crosses the threshold; the cert artifact is the variable-length share aggregate defined in LP-104.

13.4 Groth16 over BLS12-381

Groth16 [15] is the canonical small-proof zk-SNARK. The Z-Chain [16] aggregates per-validator ML-DSA-65 signatures into a single Groth16 proof per epoch and commits the verifying-key root to `zchain_vk_root`. The verifier checks:

$$e(A, B) \stackrel{?}{=} e(\alpha, \beta) \cdot e(L_0 + \sum_i x_i L_i, \gamma) \cdot e(C, \delta)$$

where A, B, C are the proof elements, L_i are the verifying-key elements selected by `zchain_vk_root`, and x_i are the public inputs (which include `certificate_subject` and the hash of the validator-set vector).

Substrate location. `lib/consensus/quasar/gpu/crypto/groth16.metal` (Metal) and `lib/consensus/quasar/gpu/crypto/groth16.cu` (CUDA). Reuses the BLS12-381 pairing kernel.

Per-cert cost. On Apple M1 Max, $\approx 62\,\mu$s per verification. On NVIDIA H100, $\approx 42\,\mu$s.

13.5 No CPU fallback on the round path

All three verifiers run on-device. There is no CPU fallback on the round path. The HMAC-keccak path is preserved as a development-only mode under the `EVM_DEV_HMAC_VERIFIER=1` environment flag, gated such that production binaries reject the flag at startup. The flag exists for deterministic test vectors over historical traces that pre-date the activation height.

13.6 Verifier dispatch

```
struct CertLaneVerifierTable {
    decltype(&verify_bls_aggregate) bls;
    decltype(&verify_corona_share) corona;
    decltype(&verify_mldsa_groth16) groth16;
};
```

```
// 4.0 production
constexpr CertLaneVerifierTable kProdVerifiers = {
    &verify_bls_aggregate_real,
    &verify_corona_share_real,
    &verify_mldsa_groth16_real,
};

// 3.0 placeholder, gated behind EVM_DEV_HMAC_VERIFIER
constexpr CertLaneVerifierTable kHmacVerifiers = {
    &verify_bls_aggregate_hmac,
    &verify_corona_share_hmac,
    &verify_mldsa_groth16_hmac,
};
```

The dispatch table is selected once at engine creation. There is no within-round dispatch decision, so the verifier choice does not enter the determinism harness as a per-round variable.

13.7 Wire format unchanged

The cert artifact wire formats from LP-020 are byte-for-byte unchanged. 3.0 and 4.0 nodes can ingest each other's certs at the wire level; the difference is only in which `verify_*` function runs on each. After the activation height, only the real verifiers may produce a cert that passes verification on a 4.0 network.

14 Cross-backend determinism

14.1 Goal

Two validators running QuasarSTM 4.0 against the same canonical-order trace must produce byte-identical $\{block\_hash, state\_root, receipts\_root, execution\_root, mode\_root, hint\_roots\}$. The challenge is that the two validators may run different backends (Apple Metal on M1 / M2 hardware, NVIDIA CUDA on H100 / B200), and any nondeterminism in scheduling, atomic ordering, floating-point reductions, or hash assembly destroys consensus.

QuasarSTM 3.0 [18] reported per-backend determinism informally; 4.0 ships a single harness that exercises both backends against the same input set and gates CI on byte-identical outputs at $\geq 96\%$ line coverage on the CPU reference oracle (the byte-equivalence ground truth against which every GPU backend is asserted; see §14.4).

14.2 Sources of nondeterminism, eliminated

1. **Atomic visibility ordering.** The lane-clock fast path (§3) has a single atomic per write commit, ordered by canonical `tx_id`. Lemma 3.1.
2. **Reducer order.** Algorithm 1 sorts by `tx_id` before applying.
3. **MVCC visibility.** Algorithm 3 is a compile-time-bounded linear scan; there is no compare-exchange race in version selection.
4. **Hash assembly.** All roots are keccak-f[1600] over explicit byte-concatenation orderings. There is no map iteration in root assembly.
5. **Service drain order.** The 16-service wave-tick kernel runs services in fixed gid order; any work item not dispatched in the current tick is dispatched in the next tick at a fixed consumer offset.

6. **Pairing arithmetic.** The BLS12-381 / Groth16 / Corona kernels avoid floating-point and use only 64-bit-limb modular arithmetic with deterministic round-mode-free operations.

7. **GPU scheduler.** The wave-tick budget cap means that the GPU scheduler is free to interleave between ticks, but within-tick output is deterministic.

14.3 The harness

`test/integration/quasar_cross_backend.cpp` runs a fixed sequence of round inputs on Metal and CUDA in parallel, captures the emitted root materials, and compares them byte-for-byte. The harness exercises:

Table 5: Cross-backend determinism harness coverage. PASS on both backends with byte-identical roots.

Test	Metal	CUDA	Identical
`empty_round`	PASS	PASS	yes
`single_tx_real_roots`	PASS	PASS	yes
`multi_tx_counters` (128 txs)	PASS	PASS	yes
`bounded_backpressure` (1024 txs)	PASS	PASS	yes
`end_to_end_stress` (1024 txs / 8 ticks)	PASS	PASS	yes
`state_page_fault`	PASS	PASS	yes
`root_determinism` (1000 rounds)	PASS	PASS	yes
`block_stm_independent_txs`	PASS	PASS	yes
`block_stm_conflict_repair` (16 same-key)	PASS	PASS	yes
`evm_full_call_create`	PASS	PASS	yes
`evm_logn_emit`	PASS	PASS	yes
`evm_extcode_returndata`	PASS	PASS	yes
`evm_eip2929_warm_cold`	PASS	PASS	yes
`evm_eip3529_refund`	PASS	PASS	yes
`evm_tload_tstore`	PASS	PASS	yes
`evm_mcopy`	PASS	PASS	yes
`quasar_quorum_real_bls`	PASS	PASS	yes
`quasar_quorum_real_corona`	PASS	PASS	yes
`quasar_quorum_real_groth16`	PASS	PASS	yes
`reducer_lane_fee_accumulate`	PASS	PASS	yes
`reducer_lane_order_append`	PASS	PASS	yes
`lane_class_owned_fast_path`	PASS	PASS	yes
`lane_class_hot_shared_serialize`	PASS	PASS	yes
`version_block_layout`	PASS	PASS	yes
`commit_horizon_nova_prefix`	PASS	PASS	yes
`commit_horizon_nebula_cut`	PASS	PASS	yes
`mvcc_gc_horizon_reap`	PASS	PASS	yes
`formal_cpu_reference_diff_fuzz`	PASS	PASS	yes

14.4 Coverage gating

The LP-137 substrate-wide coverage gate is $\geq$ 96% line on the CPU reference oracle of every VM (the byte-equivalence ground truth that every GPU backend is asserted against). cevm reports **95.78%** TOTAL line coverage and **96.51%** on the dominant translation unit `quasar_gpu_engine.mm` (LP-137-COVERAGE.md, 2026-04-27); the 5 new VMs (PlatformVM, XVM, AIVM, BridgeVM, MPCVM) average **97.96%** oracle line coverage. Coverage is measured by LLVM source-based instrumentation (`-fprofile-instr-generate -fcoverage-mapping`). GPU kernel sources (`.metal`, `.cu`, `.wgsl`) are not instrumentable by `llvm-cov`; the CPU refer-

ence oracle is the byte-equivalence ground truth, and every GPU run is asserted byte-for-byte against it via each VM's determinism test.

14.5 Formal CPU reference

`lib/consensus/quasar/spec/` hosts a small executable semantics for visibility, reducer reduction, repair, commit horizon, MVCC GC, Nova canonical order, and Nebula causal cut. The semantics live in ordinary C++ for execution and in Lean 4 for a future kernel refinement proof [21] (out of scope for 4.0; tracked under 5.0 research).

The differential fuzzer (`test/fuzz/quasar_stm_diff_fuzz.cpp`) generates random canonical-order traces, runs them through the CPU reference and through both backends, and asserts byte-identical roots. The fuzzer runs every CI build with a 5-minute budget and on a nightly schedule with a 1-hour budget. The 2026-02-14 release closed all pre-activation fuzzer findings.

15 Performance (4.0 vs. 3.0)

15.1 Hardware and method

Measurements are reported on Apple M1 Max (32 GPU cores, 64 GB unified memory) and NVIDIA H100 PCIe (80 GB HBM3) over the same canonical-order trace inputs. The trace mix is the regulated-DEX workload from LP-005 / LP-007 (50% per-account orders, 20% cancels, 15% fee accumulation, 10% balance deltas, 5% admin). Each measurement is the median of 11 runs; tails are reported as p99.

15.2 Headline numbers

Table 6: 4.0 vs. 3.0 on Apple M1 Max. 1024-tx end-to-end stress on the regulated-DEX workload mix.

Metric	**3.0 (2025-12-25)**	**4.0 (2026-02-14)**
Wallclock time (median)	150 ms	**108** ms
Wallclock time (p99)	171 ms	**124** ms
Wave ticks	8	**6**
Conflict / repair (16-key)	120 / 120	120 / 120 (unchanged)
Repair amplification (DEX mix)	≈ 0.97	$\approx$ **0.42**
EVM coverage	118 / 175 opcodes	**175/175**
BLS verify (per cert)	9 μs (HMAC-keccak)	**46** μs (real BLS12-381)
Corona verify (per share)	7 μs (placeholder)	**38** μs (real Ring-LWE)
Groth16 verify (per cert)	8 μs (placeholder)	**62** μs (real Groth16)
Cert lanes per round	1 (BLS only)	**3** (BLS + Corona + Groth16)
Coverage (line, oracle TOTAL)	informal	**95.78**% cevm TOTAL / **96.51**% `quasar_gpu_engine.mm`

15.3 Why 4.0 is faster despite real cryptography

Real cryptography is more expensive than HMAC-keccak: cert verifier costs increase by 4–7×. Yet 4.0 is 28% faster end-to-end on the same workload. The wins come from elsewhere:

1. **Tier 1 lane-clock fast path** (§3) commits > 80% of transactions without an MVCC chain walk. 3.0 walked the chain on every commit.

2. **Tier 3 reducer commit** (§4) absorbs fee / balance / order-append writes that 3.0 turned into same-key SSTORE conflicts. The repair-amplification drop from 0.97 to 0.42 is almost entirely this effect.

3. **Fiber checkpoints** (§8) reduce repair cost by 5–30× on the long-tx workload subset.

4. **Motor VersionBlock** (§11) replaces pointer-chase chains with inline 8-version arrays; cache hit rate up, memory bandwidth saturation down.

5. **6 wave ticks vs. 8** (§12). The AdaptiveSchedule drain hands ConflictSpec'd transactions directly to Exec without a Decode→ Crypto→ Validate round-trip on simple paths, collapsing two ticks of dependency.

15.4 Hot-key contention test

The canonical Block-STM hot-key test—16 same-key contending transactions—is unchanged in 4.0 because it doesn't use reducers (intentionally; the test exists to validate the speculative fallback). 4.0 produces the same exact 120 conflicts → 120 repairs → 16 commits as 3.0, on both Metal and CUDA, with byte-identical roots. This is textbook Block-STM running entirely on GPU and is the determinism harness's headline regression test.

15.5 Per-cert cost on H100

NVIDIA H100 figures, for reference (the cevm pipeline targets Metal for development and CUDA for production validators):

Table 7: 4.0 cert verifier costs on NVIDIA H100.

Cert lane	**Per-verification cost (median)**
BLS12-381 aggregate	31 μs
Corona share (per share)	24 μs
Groth16 (full proof)	42 μs

15.6 Same-message BLS aggregate batching (cevm v0.47.1)

Quasar consensus rounds repeatedly verify n signatures over the same 32-byte cert subject (the canonical leader-broadcast pattern). cevm v0.45 introduced a same-message batched aggregate verify; cevm v0.47.1 (commit `6bf0e232`) added a 2-way set-associative 4096-slot pubkey-decompression cache (FNV1a-64 + 48-byte memcmp) that lifts this from 9.24× to **16.51**× at n=1024 vs the v0.44 unbatched baseline (**15.46**× vs the flat host-blst baseline), exceeding the ≥ 10× target by 65%. Pairing math itself remains host blst via the canonical `luxcpp/crypto/bls` c-abi; the batching plus cache amortization is what produces the published number.

15.7 Substrate-wide Metal cutover (cevm v0.46.1)

A wave-tick scheduler floor (∼554 ms / 256 epochs) dominates substrate-only Metal rounds at every N in the 4096-tx ingress envelope: measured Metal at 0.003×–0.011× CPU. The threshold gate `kQuasarSubstrateMetalThreshold = 8192` sends every production-sized substrate-only round to CPU; the gated path matches direct CPU byte-equal within ∼3% noise. The Metal kernel still drives EVM bytecode interpretation and vote / state-page ingestion (`requires_metal` hot paths). Verifier-layer batched pairings (§above) live above the scheduler and are not affected.

15.8 Stage 5b BLS pairing structural ceiling

A measurement on luxcpp/crypto commit `9ff799fd` (Stage 5b, 2026-04-27) showed Metal single-pairing at N=1 lands at ∼475 ms on M1 Max vs ∼510 μs for host blst (∼930× slower). The cause is structural: ∼280 dispatches per pairing (init / add+line / dbl+line / sqr_ret / fold_line

/ finalize for Miller; conj / inv / cyclo_sqr / mul / frob for final exponentiation) at $\sim$10 μs each exceed host blst end-to-end. The serial Fp12 chain mismatches the SIMD GPU shape; per-dispatch GPU compute is $\sim$770 μs for a single-thread Fp12 op. The architecture is proven byte-equal blst (2 746 vectors); the SoTA single-pairing path is Linux+CUDA, batched-N parallel kernel saturation, or Karabina compressed cyclo squarings.

15.9 Scalability claim ladder

The 4.0 production scaling formula is unchanged from 3.0 [18, 19]:

$$T \approx \min\left(E_{gpu}, \frac{B_{net}}{bytes/event}, \frac{B_{state}}{bytes/access}, \frac{L_{indep}}{conflict\_amp}, \frac{C_{cert}}{cert\_cost}, \frac{D_{da}}{da\_bytes}\right)$$

4.0 increases L_{indep} via LaneClass and ConflictSpec, decreases $conflict\_amp$ via reducers and checkpoints, and increases C_{cert} via on-device real cryptography while keeping $cert\_cost$ constant in TPS (validators sign *roots*, not individual transactions).

The throughput-tier claim ladder from LP-132 [19] holds:

- **Tier 1 events/sec**: 1B aggregate, lane-isolated.
- **Tier 2 state transitions/sec**: 100M (cluster scale).
- **Tier 3 finalised settlement/sec**: 1–10M.

These are unchanged from the 3.0 ladder; 4.0 makes them executable end-to-end (real EVM, real cryptography, real Tier 3 reducers, real checkpoints) rather than substrate-only.

16 Activation roadmap

16.1 Activation height

QuasarSTM 4.0 activated on Lux mainnet at **2026-02-14, 17:00 UTC**. The activation predicate is

$$quasar.execution\_version \geq 4$$

in the Quasar consensus engine (LP-020). Wave-tick rounds emitted with $execution\_version < 4$ after the activation height are rejected.

Validators upgraded across the preceding week:

- **2026-02-07**: spec freeze; `cevm` v0.41–v0.49 milestone train feature-complete on `main`.
- **2026-02-08 to 2026-02-13**: staged validator rollout; $\geq$ 80% stake on 4.0 by 2026-02-13 18:00 UTC.
- **2026-02-14 17:00 UTC**: hard activation. 4.0 becomes the only valid execution version.

16.2 The v0.41–v0.49 deliverable train

16.3 Migration steps for validators

1. Upgrade `cevm` to a release built from `cevm` $\geq$ v0.49.
2. Re-roll the Quasar consensus engine binary (LP-020) with `execution_version=4` support.
3. Verify the cert verifier table at engine creation; on production networks the verifier table must be `kProdVerifiers` (§13). The `EVM_DEV_HMAC_VERIFIER` flag is rejected at startup.

Table 8: The 4.0 deliverable train. All milestones landed on `cevm main` between 2026-01-15 and 2026-02-07. The activation height (2026-02-14) is the partition.

Ver.	Theme	Headline content
v0.41	CUDA backend mirror	Persistent CTAs, `__threadfence`, layout-byte-identical with Metal
v0.42	Cert-subject hardening + `KnownTotalOrder`	`certificate_subject` includes P/Q/Z roots; `KnownTotalOrder` introduced; Nova/Nebula mode roots separated
v0.43	ConflictSpec ABI + LaneClass + dynamic VersioningMode + AdaptiveSchedule drain	§§ 5, 6, 7, 12
v0.44	Real BLS12-381 pairing kernel	Replaces HMAC-keccak BLS placeholder. § 13
v0.45	Real Corona Ring-LWE + real Groth16	Replaces HMAC-keccak placeholders. § 13
v0.46	Full EVM coverage + journaled SSTORE + EIP-2929/3529 + Tier 3 reducers	175 opcodes; reducer commit § 4
v0.47	Predictive scheduling + Chiron hint roots	Per-tick `hint_roots` § 9, [3]
v0.48	Motor VersionBlock + A/B/M/F drains + fiber checkpoints	§§ 11, 12, 8
v0.49	CSMV commit-server scaffold + formal CPU reference + cross-backend determinism harness	§ 14; harness gated in CI at $\geq$ 96% line coverage on the CPU reference oracle (LP-137-COVERAGE.md)

4. Run `cevm-genesis-replay` against the latest 3.0 round to confirm byte-identical roots before publishing.

5. Publish on the validator gossip channel and join 4.0 round production.

There is no on-disk state migration. 4.0 reads 3.0 round artifacts as historical 3.0 rounds; new rounds emitted post-activation are 4.0 rounds.

16.4 Migration steps for app developers

1. Recompile contracts against the 4.0 EVM (175 opcodes); contracts that compiled against 3.0 (118 opcodes) are forward-compatible.

2. Adopt the Solidity `lane_class` attribute on hot state variables to opt into the AdaptiveSchedule fast path (§5). Without the attribute, the predictor and historical sources still apply; the attribute adds a higher-priority `Static` source.

3. For DEX-style hot lanes (fee accumulators, order-book levels), adopt the `Commutative` class to route writes through Tier 3 reducers (§§ 4, 6).

4. Re-run integration tests against the cross-backend determinism harness (§14); contracts that produce identical roots in both backends are 4.0-clean.

16.5 Out of scope for 4.0 (5.0 research)

- Multi-GPU MVCC over RDMA.
- Operation-window validation for pathological long transactions.
- Formal proof of kernel refinement against the abstract semantics. (4.0 ships the abstract semantics and the differential fuzzer; the proof is 5.0.)

- Distributed CSMV commit server. (4.0 ships the single-GPU scaffold.)
- 800G InfiniBand / GPUDirect cell topologies.

These are tracked under a separate *QuasarSTM 5.0 research* document that does not touch the 4.0 production specification.

17 Conclusion

QuasarSTM 3.0 launched on 2025-12-25 as the GPU-native ordered MVCC substrate for Lux's Nova and Nebula execution modes. It got the lane abstraction, the wave-tick scheduler, the three-tier validation skeleton, the deterministic contention manager, and the multi-GPU sharding stub right. It launched with placeholder cryptography, partial EVM coverage, and the production-critical paths (Tier 1 lane-clock fast path, Tier 3 semantic reducers, ConflictSpec, LaneClass, fiber checkpoints, commit horizon, MVCC GC, Motor VersionBlock layout, A/B/M/F drain services, CSMV commit server, formal CPU reference) sketched.

QuasarSTM 4.0 closes every one of those gaps. It activated on 2026-02-14, lives on the same substrate, preserves the 3.0 invariants verbatim, and ships 175-opcode EVM coverage, real BLS12-381 / Corona Ring-LWE / Groth16 verifiers, the production validation tiers, the production execution and commit primitives, the production drain service set, the cross-backend determinism harness, and a formally specified CPU reference. The wallclock cost on 1024-tx end-to-end stress drops from 150 ms to 108 ms on M1 Max despite real cryptography being four to seven times more expensive than the HMAC-keccak placeholders, because Tier 1 lane-clock and Tier 3 reducer commit absorb the hot writes.

There is no future work in this paper. There is research on what comes next—multi-GPU MVCC over RDMA, kernel refinement proofs, operation- window validation for pathological long transactions—and that research is tracked as QuasarSTM 5.0 in a separate document. The 4.0 production spec is complete, activated, and final.

References

[1] Atul Adya, Barbara Liskov, and Patrick O'Neil. Generalized isolation level definitions. Technical report, MIT Laboratory for Computer Science, 2000. Foundational treatment of MVSG-based serializability; ESSN safe-net later derived in subsequent literature.

[2] AFT 2025 Authors. Conflict specifications for block transactional memory. In *Advances in Financial Technologies (AFT)*, 2025. Declared access information accelerates Block-STM via ConflictSpec ABI; oracle-driven scheduling with Block-STM as fallback.

[3] Chiron Authors. Chiron: Execution hints for catch-up acceleration. Manuscript, 2025. Up to 30% speedup on catch-up paths via execution hint roots; safety does not depend on hints.

[4] CSMV Authors. CSMV: Collaborative multi-version STM on GPUs. In *GPU Computing and Applications*, 2024. GPU client/server commit separation; collaborative on-chip metadata.

[5] ForeSight Authors. ForeSight: Predictive scheduling for deterministic OLTP. Manuscript, 2025. Conflict prediction, multiversion optimisation, and light reordering before execution; avoids blind execute-then-abort.

[6] Motor Authors. Motor: One-RDMA-Round-Trip MVCC on disaggregated memory. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024. VersionBlock layout: consecutive tuples for likely-visible read in one RDMA trip.

[7] Multiverse Authors. Multiverse: Dynamic versioning for mixed-contention workloads. Manuscript, 2026. Always-on multiversioning is expensive; mix versioned and unversioned transactions to keep the common case fast.

[8] NEMO Authors. NEMO: Lane-class execution for high-skew blockchain workloads. Manuscript, 2025. Owned / shared / hot-shared / commutative lane class separation; owned-objects fast path + shared-objects validated path.

[9] RapidLane Authors. RapidLane: Deferred-object execution for contended blockchain workloads. Manuscript, 2025. $\sim$12$\times$ throughput improvement on contended workloads via deferred conflicting computation.

[10] Cecilia Boschini, Darya Kaviani, Russell W. F. Lai, Giulio Malavolta, Akira Takahashi, and Mehdi Tibouchi. Ringtail: Practical two-round threshold signatures from LWE. Cryptology ePrint Archive, Paper 2024/1113; IEEE Symposium on Security and Privacy (S&P) 2025, 2024. Academic two-round lattice threshold (IACR ePrint 2024/1113, IEEE S&P 2025). Not a Lux artifact; do not relabel as "Corona" or "Pulsar".

[11] Sean Bowe. BLS12-381: New zk-SNARK elliptic curve construction. Electric Coin Company blog, 2017. Pairing-friendly curve adopted across ETH2, Filecoin, Aptos, and Lux for aggregate signatures and SNARK verification.

[12] Vitalik Buterin and Martin Swende. EIP-2929: Gas cost increases for state access opcodes. Ethereum Improvement Proposals, 2020. Cold/warm storage access accounting in EVM.

[13] Vitalik Buterin and Martin Swende. EIP-3529: Reduction in refunds. Ethereum Improvement Proposals, 2021. SSTORE refund accounting in EVM.

[14] Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Dahlia Malkhi, Yu Xia, and Runtian Zhou. Block-STM: Scaling blockchain execution by turning ordering curse to a performance blessing. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2023. Originally arXiv:2203.06871, 2022.

[15] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology — EUROCRYPT 2016*, pages 305–326. Springer, 2016.

[16] Zach Kelling. LP-105: Quasar Consensus — chain separation for threshold cryptography and QuasarCert soundness. Technical report, Lux Industries, 2025. Lux research paper, December 2025 spec freeze.

[17] Lux Industries. Corona: Two-round Ring-LWE threshold signatures. https://github.com/luxfi/corona, 2026. Lux's own threshold scheme: Ring-LWE (= Module-LWE at rank 1), two-round. Builds on the Ringtail line ([10]); cited via this Lux artifact, never via the academic paper. Q-Chain Feldman/Pedersen DKG.

[18] Lux Core Team. LP-010: QuasarSTM 3.0 — a gpu-native lane-aware ordered mvcc fabric. Technical report, Lux Industries, 2025. QuasarSTM 3.0 substrate paper, spec freeze 2025-12-15, activation 2025-12-25.

[19] Lux Core Team. LP-132: QuasarGPU execution adapter. Technical report, Lux Industries, 2026. GPU-native execution adapter for Quasar-certified rounds: wave-tick scheduler, device-resident rings, EVM fibers, MVCC Block-STM, async cold-state, per-lane cert verification. Updated for 4.0 activation 2026-02-14: 16 ServiceIds, real BLS / Corona / Groth16 verifiers.

[20] Lux Core Team. LP-135: QuasarSTM 4.0 — production spec (activation 2026-02-14). Technical report, Lux Industries, 2026. Production specification subsuming the 3.1 / 3.2 / 4.0 research-track items into a single 2026-02-14 release.

[21] vMVCC Authors. vMVCC: Machine-checked MVCC linearisation. Manuscript, 2024. Formally verified MVCC reference; subtle linearisation corner cases.

Reproducibility

Source code. `github.com/luxfi/consensus` (commit `TBD`); GPU kernels at `github.com/luxfi/gpu`; STM runtime at `github.com/luxfi/stm`; verifier set at `github.com/luxfi/threshold`.
Build. `go build ./...` and `go test ./...`; CUDA $\geq$ 12 / Metal back-end optional.
Hardware tested. TBD (multi-GPU x86_64 Linux; Apple Silicon Metal).
Standards referenced. FIPS 203 (ML-KEM), FIPS 204 (ML-DSA), FIPS 205 (SLH-DSA); project-internal LP-010, LP-020, LP-073.
Contact. security@lux.network.