Chain-First Architecture: Blockchain as Source of Truth for Regulated Financial Infrastructure

Zach Kelling

Lux Industries

2024

Abstract

We describe a "chain-first" architecture for regulated financial infrastructure in which the blockchain is the authoritative source of truth for all asset ownership and transfer records, while traditional databases serve only as read caches that are always re-derivable from on-chain state. This inverts the conventional pattern in which a centralized database is authoritative and the blockchain is an afterthought settlement layer. We formalize the architecture as a state machine with two components: (1) an append-only chain that records every state transition via digitally signed transactions, and (2) a local cache (SQLite in development, PostgreSQL in production) that materializes the current state by replaying chain events through a deterministic indexer. We prove three correctness properties: *cache consistency* (the cache always reflects the chain state), *crash recovery* (the cache can be fully reconstructed from the chain), and *chain independence* (the API layer continues to serve reads from the cache when the chain is temporarily unavailable, and queues writes for later settlement). The architecture is deployed on the Liquid EVM (chain IDs 8675309–8675311) for an ATS handling 12,784 security tokens, with benchmarks showing 2ms read latency from cache versus 150ms from chain RPC.

Contents

1 Introduction

Financial systems have two fundamental requirements for record-keeping: *durability* (records must not be lost) and *integrity* (records must not be silently modified). Traditional architectures use a centralized database (PostgreSQL, Oracle, SQL Server) as the source of truth, protected by access controls, audit logs, and backup procedures. The integrity guarantee is institutional: it holds because the operator is trustworthy and the database is properly administered.

Blockchain provides a stronger guarantee. Every state transition is recorded as a digitally signed transaction in an append-only ledger. The integrity guarantee is cryptographic: modifying any record requires forging a digital signature or controlling a supermajority of validators. The durability guarantee is distributed: the ledger is replicated across every full node.

The chain-first architecture makes the blockchain the single source of truth and reduces the database to a materialized view—a read cache that accelerates queries but contains no original data. If the cache is lost, it is fully reconstructable from the chain. If the chain and cache disagree, the chain wins.

1.1 Why Not Database-First?

In a database-first architecture, the blockchain is a settlement rail: the database records a trade, and a background job eventually posts the settlement transaction to the chain. This creates several problems:

1. **Split brain.** If the database says Alice owns 100 shares but the chain says she owns 95, which is correct? The database-first architecture has no deterministic answer.
2. **Silent corruption.** A database administrator can modify records without leaving a cryptographic trace. The audit log is stored in the same database it is supposed to protect.
3. **Backup divergence.** Database backups taken at different times may reflect inconsistent states. Restoring from backup can lose or duplicate settlements.
4. **Regulatory risk.** SEC Rule 17a-4 requires records to be stored in non-rewritable, non-erasable format (WORM). A mutable database does not satisfy this requirement without additional infrastructure.

1.2 Chain-First: The Inversion

In the chain-first architecture:

- The chain is WORM (Write Once, Read Many) by construction.
- Every state transition is a signed transaction.
- The database is a read cache, always derivable from chain events.
- If the cache is corrupted or lost, it is rebuilt by replaying the chain from genesis.

2 Formal Model

2.1 System State

Definition 2.1 (Chain State)**.** *The chain state at block h is a function $\mathcal{C}_h : \text{Address} \times \text{Token} \to \mathbb{N}$ mapping each (address, token) pair to a balance. The genesis state $\mathcal{C}_0$ is defined by the genesis configuration. For $h > 0$:*

$$\mathcal{C}_h = \mathcal{T}_h(\mathcal{C}_{h-1})$$

where $\mathcal{T}_h$ is the deterministic state transition function defined by the transactions in block h.

Definition 2.2 (Cache State)**.** *The cache state $\mathcal{D}_h$ is maintained by the indexer. For each chain event e emitted by block h, the indexer applies a deterministic update $\delta(e)$ to the cache:*

$$\mathcal{D}_h = \delta(e_1) \circ \delta(e_2) \circ \cdots \circ \delta(e_k)(\mathcal{D}_{h-1})$$

where $e_1, \ldots, e_k$ are the events in block h.

Definition 2.3 (Consistency)**.** *The system is* consistent at block h *if and only if:*

$$\forall\,(a,t) : \mathcal{D}_h(a,t) = \mathcal{C}_h(a,t)$$

That is, the cache reflects the chain state for every address and token.

2.2 Settlement Pipeline

The pipeline from API request to on-chain settlement:

1. **API receives order.** The order is validated (authentication, authorization, compliance checks) and recorded in the `pending_orders` table.
2. **Matching engine matches order.** The matched trade is recorded in the `trades` table and the `pending_settlements` table.
3. **Settlement cron fires (30s interval).** For each pending settlement:
   (a) Construct the on-chain transaction (mint target token, burn source token).
   (b) Initiate MPC signing (2-of-3 threshold).
   (c) Broadcast the signed transaction.
4. **Chain confirms transaction.** The transaction is included in a block.
5. **Indexer processes block.** The indexer reads the block's events and updates the cache. The `pending_settlements` entry is marked as confirmed.
6. **Cache serves reads.** The API layer reads balances, positions, and transaction history from the cache.

3 Correctness Properties

3.1 Cache Consistency

Theorem 3.1 (Cache Consistency)**.** *If the indexer processes every block from genesis through block h without skipping or reordering, then $\mathcal{D}_h = \mathcal{C}_h$.*

Proof. By induction on block height h.

Base case ($h = 0$). The cache is initialized from the genesis configuration, which is identical to $\mathcal{C}_0$. Therefore $\mathcal{D}_0 = \mathcal{C}_0$.

Inductive step ($h \to h+1$). Assume $\mathcal{D}_h = \mathcal{C}_h$. Block $h+1$ produces events $e_1, \ldots, e_k$, and the chain state transitions to $\mathcal{C}_{h+1} = \mathcal{T}_{h+1}(\mathcal{C}_h)$. The indexer applies $\delta(e_1) \circ \cdots \circ \delta(e_k)$ to $\mathcal{D}_h$. Because the indexer's δ functions are the deterministic projection of the chain's state transition function $\mathcal{T}$ onto the cached dimensions (balances, positions, transaction history), and because the events e_i encode exactly the state changes made by $\mathcal{T}_{h+1}$, we have $\mathcal{D}_{h+1} = \mathcal{C}_{h+1}$. □

3.2 Crash Recovery

Theorem 3.2 (Crash Recovery)**.** *If the cache is destroyed at any point, it can be fully reconstructed by replaying all chain blocks from genesis.*

Proof. Initialize an empty cache $\mathcal{D}'_0 = \mathcal{C}_0$ (from the genesis configuration). For each block $h = 1, 2, \ldots, H$ (where H is the current chain height), apply the indexer's δ functions to the events in block h. By Theorem 3.1, the reconstructed cache $\mathcal{D}'_H = \mathcal{C}_H$.

The reconstruction is deterministic: given the same chain, the same cache is produced regardless of when the reconstruction occurs. The chain is the only input; no external state is required. □

3.3 Chain Independence

Theorem 3.3 (Chain Independence)**.** *If the chain becomes temporarily unavailable, the API layer continues to serve reads from the cache (reflecting the last known chain state) and queues writes for later settlement. No data is lost.*

Proof. Reads are served from the cache $\mathcal{D}_h$ where h is the last processed block. The cache is a local database (SQLite or PostgreSQL) that does not depend on chain connectivity.

Writes (new orders, trades) are recorded in the `pending_settlements` table in the local database. The settlement cron retries pending settlements on a 30-second interval. When chain connectivity is restored, the cron processes all queued settlements. The `pending_settlements` table acts as a durable write-ahead log.

No data is lost because:

1. Reads serve stale-but-consistent data (consistent as of block h).
2. Writes are durably queued and will be settled when the chain returns.
3. The chain's append-only property guarantees that blocks processed before the outage are still valid after the outage.

□

4 WORM Audit Trail

The blockchain is a natural WORM (Write Once, Read Many) storage medium: once a transaction is included in a finalized block, it cannot be modified or deleted without forging digital signatures or controlling a supermajority of validators.

4.1 Hash Chain Integrity

Each block contains a hash of the previous block, forming a hash chain from genesis to the current head:

$$H_h = \mathsf{Hash}(H_{h-1} \| \mathsf{TxRoot}_h \| \mathsf{StateRoot}_h \| \mathsf{Timestamp}_h)$$

Theorem 4.1 (Hash Chain Tamper Detection)**.** *Any modification to any block $h' \leq h$ in the chain is detectable by any party that knows the current block hash H_h.*

Proof. Suppose block h' is modified, producing a different hash $H'_{h'} \neq H_{h'}$. Block $h'+1$ includes $H_{h'}$ in its preimage. Modifying block h' without modifying block $h'+1$ causes the hash chain to break: $\mathsf{Hash}(H'_{h'} \| \cdots) \neq H_{h'+1}$. Therefore the adversary must also modify block $h'+1$, which in turn requires modifying block $h'+2$, and so on up to block h. This cascade requires modifying every block from h' to h, which requires forging the digital signatures of the block producers for each of those blocks. Under the EUF-CMA security of the signature scheme, this is infeasible.

Any party that stores H_h (the current head hash) can verify the integrity of any earlier block by recomputing the hash chain. If the recomputed H_h differs from the stored H_h, a modification has occurred. □

4.2 Compliance with Rule 17a-4

SEC Rule 17a-4(f) requires that records be preserved in a format that:

1. Preserves records exclusively in a non-rewritable, non-erasable format (WORM).
2. Verifies automatically the quality and accuracy of the recording process.
3. Serializes the original and duplicate units of storage media and provides a time-date for the required period of retention.

The chain-first architecture satisfies all three requirements:

1. The blockchain is WORM by construction (hash chain + digital signatures).
2. The indexer deterministically replays chain events; any discrepancy between chain and cache is detected automatically.
3. Every block carries a timestamp, and the chain provides a total ordering of all events with cryptographic serialization (block height + transaction index).

5 Architecture Details

5.1 Cache Layer: SQLite in Development, PostgreSQL in Production

The cache is intentionally disposable. In development, each ATS instance uses an embedded SQLite database (via Hanzo Base) for zero-configuration operation. In production, multiple ATS instances share a PostgreSQL database for concurrent access.

In both cases, the cache can be rebuilt from the chain:

```
$ atsd cache rebuild --from-genesis
  Processing block 0...
  Processing block 1...
  ...
  Processing block 847291...
  Cache rebuilt: 12,784 tokens, 847,291 blocks, 2.3M events
  Duration: 4m 12s
```

5.2 Indexer: Deterministic Event Replay

The indexer subscribes to chain events via WebSocket and processes them in strict block order. For each event type, a deterministic handler updates the cache:

Chain Event	**Cache Update**
`Transfer(from, to, amount)`	Debit `from`, credit `to`
`Mint(to, amount)`	Credit `to`
`Burn(from, amount)`	Debit `from`
`SecurityTokenCreated(addr, symbol)`	Insert token record
`ComplianceCheck(addr, result)`	Update compliance status

Table 1: Event-to-cache mapping for the deterministic indexer.

The indexer maintains a `last_processed_block` cursor. On restart, it resumes from the cursor, ensuring exactly-once processing.

5.3 Read Path (Fast)

The API layer reads from the cache for all queries:

Query	**Cache (ms)**	**Chain RPC (ms)**
Get balance	0.8	45
List positions	1.2	150
Transaction history	2.0	320
Token metadata	0.5	25

Table 2: Read latency: cache vs. chain RPC.

The cache provides 50–200× faster reads than chain RPC calls. For a securities exchange serving 1,000 concurrent users, this is the difference between a responsive UI and an unusable one.

5.4 Write Path (Durable)

Writes flow through the settlement pipeline:

```
API -> validation -> pending_orders (DB)
    -> matching    -> trades (DB) + pending_settlements (DB)
    -> cron (30s)  -> MPC sign -> broadcast -> chain
    -> indexer     -> cache update -> pending_settlements.confirmed
```

If any step fails, the pending settlement remains in the queue and is retried. The settlement is considered final only when the chain transaction is confirmed and the indexer has processed the block.

6 Resilience Analysis

Failure	**Database-First**	**Chain-First**
Cache/DB corruption	Data loss (DB is truth)	Rebuild from chain (chain is truth)
Chain outage	Trades continue, no settlement	Trades continue, settlements queued
Indexer crash	N/A (no indexer)	Resume from cursor, no data loss
Split brain	Ambiguous ownership	Chain wins (deterministic)
Admin tampering	Silent modification	Detected by hash chain
Backup divergence	Possible inconsistency	Cache is disposable; chain is the backup

Table 3: Failure modes: database-first vs. chain-first.

The chain-first architecture has a strictly better failure model for every scenario except chain outage, where both architectures degrade similarly (trades continue, settlement is deferred).

7 Formal Verification

The hash chain integrity property is mechanized in Lean 4:

- `Crypto.HashChainIntegrity`: proof that any modification to any block in the chain is detectable by comparing the recomputed head hash against the stored head hash.

The cache consistency and crash recovery properties are modeled in TLA+ in the `lux/proofs/tla/` directory as the `VaultSync` specification.

8 Conclusion

The chain-first architecture aligns the technical architecture with the regulatory requirement: the blockchain is a WORM audit trail that satisfies Rule 17a-4, the database is a read cache that accelerates queries, and the settlement pipeline ensures that every trade is durably recorded on-chain. The architecture tolerates chain outages, cache corruption, and node failures without

data loss. The system is deployed on the Liquid EVM with 12,784 security tokens and sub-2ms cache reads.

References

[1] SEC. *Rule 17a-4: Records to be Preserved by Certain Exchange Members, Brokers and Dealers*. 17 CFR 240.17a-4.

[2] FINRA. *Rule 4511: General Requirements (Books and Records)*.

[3] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008.

[4] G. Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. Yellow Paper, 2014.

[5] R. Merkle. *A Certified Digital Signature*. CRYPTO 1989.

[6] L. Lamport, R. Shostak, M. Pease. *The Byzantine Generals Problem*. ACM TOPLAS, 4(3):382–401, 1982.