Version: 1.1
January 22, 2026

Obsidian: An Ethereum-Compatible High-Throughput Blockchain for Low-Latency Data Publication and Permanence

Author: Daniel Odom

hello@obsidianchain.net

Abstract

Obsidian is a blockchain optimized for low-latency, high-frequency data publication. Through the Silica Protocol, message payloads travel through parallel lanes with dedicated validator committees, while only compact cryptographic commitments are included in blocks. This separation allows data throughput to scale independently of EVM execution, maintaining full Ethereum compatibility.

Unlike storage-oriented blockchains designed for bulk allocation or data leasing, Obsidian targets small, frequent payloads, enabling decentralized applications that require continuous data streams without centralized infrastructure.

*Technical terms are defined in **Appendix B: Glossary** on page 20 - 21.*

1. Introduction

1.1 The Data Availability Problem

In the current Web3 ecosystem, reliable, on demand data availability and publishing is addressed with solutions that have significant tradeoffs between cost, latency, and decentralization.

Centralized data infrastructure remains the most efficient but introduces massive censorship risks and highly questionable permanence. Current storage-oriented decentralized network solutions greatly improve permanence but are typically optimized for prepaid data allocation and static data hosting, making them inadequately suited for on-demand, low-latency data publication.

As a consequence, decentralized applications requiring continuous data streams, such as real time messaging, rely heavily on centralized solutions.

1.2 Our Solution

Obsidian addresses this issue through attaching an on demand data protocol (Silica Protocol) directly alongside transaction blocks. While EVM transactions are computationally intensive, they are relatively small.

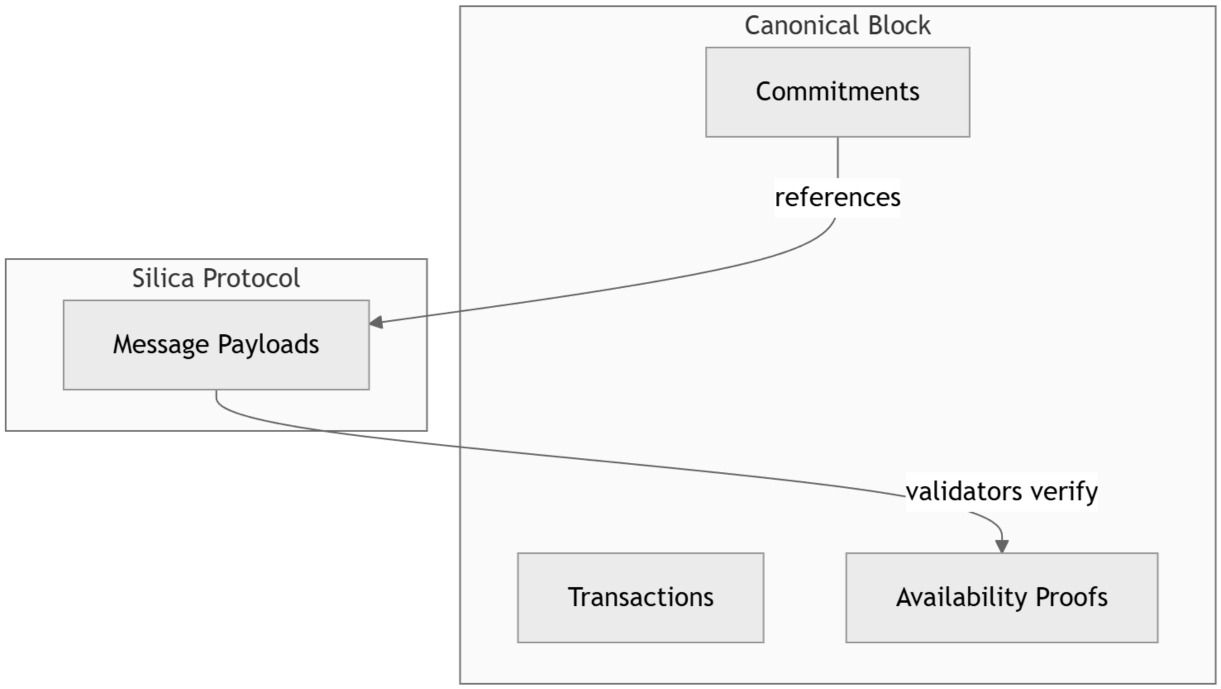
The Silica Protocol leverages this gap by capitalizing on underused network capacity to carry larger, non-executable data payloads that require very minimal CPU processing. This enabled Obsidian to sustain high data throughput without increasing EVM execution overhead which would otherwise degrade transaction performance.

1.3 Design Principles

Execution Isolation: Data flow should never slow down transactions. Heavy data moves on its own path so EVM execution stays fast.
Native Data Pathway: Messages are built into the protocol itself, not shoved through smart contracts.
Ethereum Compatibility: Full EVM support with the same tools developers already use.
Accessible Publication: Users can publish data either by paying fees or by using compute-based mechanisms for spam prevention, depending on urgency and inclusion needs.
Verifiable Permanence: Once data is included, cryptographic commitments make sure it can’t be altered or faked later.

2. Architecture Overview

Obsidian builds upon the Ethereum execution and consensus layers through a dedicated data plane called the Silica Protocol



2.1.1 How this works

- **Canonical Block:** This is what gets finalized on-chain. It contains normal EVM transactions plus small cryptographic commitments that point to message data stored outside the canonical block body.
- **Silica Protocol (Data Plane):** Message payloads are propagated across the network as data objects. These data sidecards are gossiped peer-to-peer, erasure-coded, and temporarily stored by standard validators, permanently stored by full archive nodes and sharded archive nodes. They are part of the blockchain protocol, but they are intentionally kept outside the block body so they do not slow down block propagation or voting.
- **The connection:** Each canonical block includes cryptographic commitments that link to specific Silica data sidecars. Validators verify that the referenced data is available by participating in committee-based availability checks. Once enough independent validators confirm possession of the data, an availability certificate is produced and anchored back into the canonical block.

2.1.2 Why separate?

On time block publication is critical for chain consistency and liveliness. Large data payloads, while not computationally expensive, take more time to distribute under timely latency constraints.

By separating execution and ordering from data transport and availability, Obsidian allows transactions to remain fast and predictable; data throughput to scale independently with minimal impact on execution; and validators to verify availability without immediately downloading all payload data.

This results in a system where data remains on-chain and verifiably authentic without competing with transactions for blockspace, computing resources, and bandwidth.

2.2 Silica Protocol

The Silica Protocol defines how message data is propagated, validated, and made available across the chain.

Silica is responsible for: routing messages into respective lanes, erasure coding each message batch into redundant chunks, peer to peer lane committee gossip, verifying availability though lane committee voting, serving data to requested nodes.

Silica operates alongside consensus but does not block it. A block can be finalized even while data propagation is still in progress.

2.2.1 Where does the data actually live?

The network is divided into parallel lanes. Each lane has a rotating committee of validators assigned to it. When you submit a message:

1. It gets routed to a specific lane (based on your address)
2. The lane's committee validators store chunks of your data
3. Each validator holds a piece - no single validator has the full message

This distribution model prevents any single validator from becoming a bandwidth bottleneck.

2.2.2 How to retrieve data

Data retrieval is simple, you may query by block or by sender address:

```
// Get all messages from a specific block
messages = eth_getBlockMessages(blockNumber)

// Get messages by sender address within a block range
messages = eth_getMessagesByAddress(address, startBlock, endBlock)
```

The node handles everything behind the scenes; locating the right validators, fetching chunks from lane committees, and reconstructing the original data. You don't need to know which lane or committee stored it, this is all done for you.

2.2.3 Node data location

Data Age	Stored By	How Node Fetches
Recent	Lane committee validators	Requests chunks, reconstructs
Historical	Archival nodes	Direct retrieval

3. Messages

3.1 What is a Message?

A message is a signed data payload submitted to Obsidian. Unlike transactions, messages do not invoke the EVM or execute smart contract logic. The only exception is priority messages, which include a small deterministic balance change to account for the inclusion bid.

A message includes the sender address, payload data, target block, nonce, bid or VDF proof, and a cryptographic signature.

Messages are designed for high-frequency submission, small payload sizes, low-latency inclusion, and permanent on-chain ordering.

3.2 Message Identity

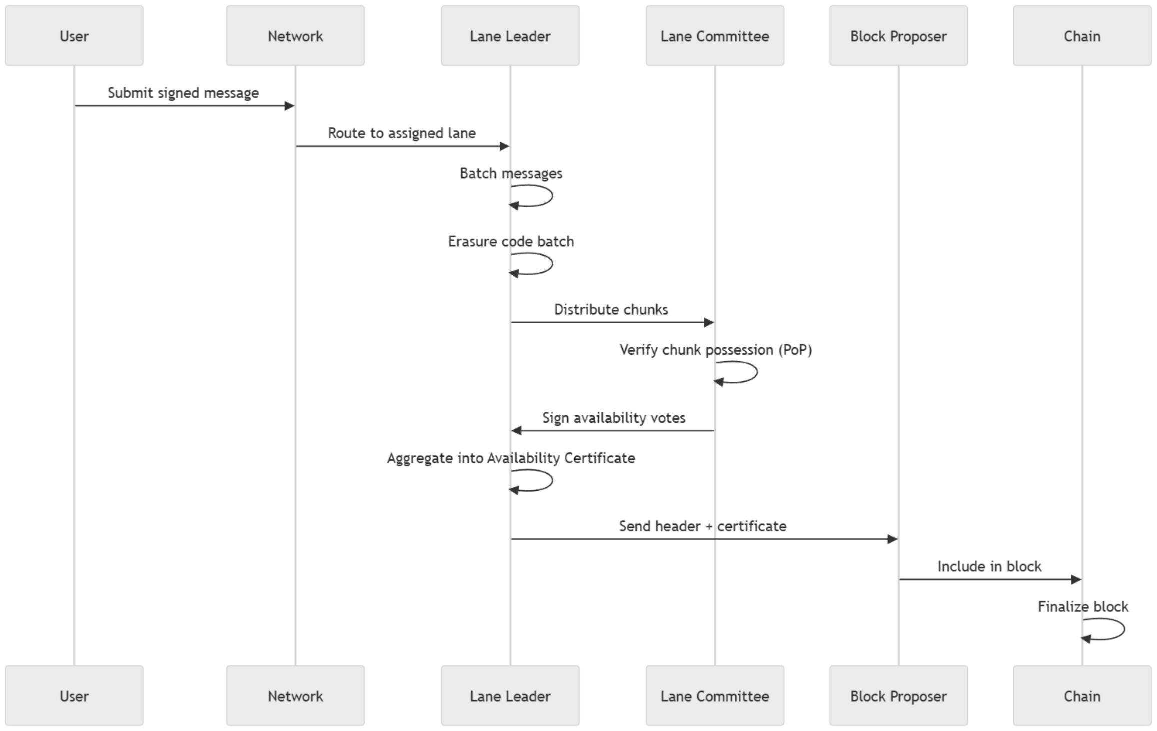
A message's identity is based on a commitment to a payload, not the payload itself. This allows the consensus layer to order and verify messages without downloading full payloads. The payload bytes travel separately through the data plane. This keeps gossiped bytes low.

3.3 Lane Routing

All messages, both standard and priority, are deterministically routed to lanes based on sender address. This ensures all messages from a single sender go to the same lane. The primary benefit is simplicity: nodes can immediately determine which lane committee should receive a message without knowing the current slot or RANDAO state. This enables efficient routing at the edge of the network due to messages only being gossiped to the lane committee members.

Note: Nonce deduplication and balance checks happen at batch time across all lanes, not per-lane. The permanent lane assignment is a routing optimization, not a consensus requirement.

4. Message Lifecycle



4.1 Submission

Users sign messages and submit them to the network. The protocol validates the signature and routes the message to the appropriate lane.

4.2 Batching

The Lane Leader (a validator elected for that slot and lane) collects messages and forms a batch. The leader computes the micro root and the data commitment

4.3 Erasure Coding

The batch is encoded using Reed-Solomon erasure coding, producing redundant chunks. Any subset of chunks (above a threshold) can reconstruct the original data. This allows the network to tolerate missing chunks without losing data.

4.4 Committee Voting

Each lane has a Lane Committee—a subset of validators responsible for that lane's data availability. Committee members receive chunks from the leader, verify possession of their assigned chunks (PoP), sign availability votes attesting to data availability, and aggregate votes into an Availability Certificate (AC) at threshold.

4.5 Block Inclusion

The block proposer collects lane headers and availability certificates from all lanes. These are included in the canonical block body. The proposer does not need the full payload data, only the commitments and proofs.

4.6 Finality

Obsidian identifies two types of finality:

- **Inclusion Finality:** The lane header is in a finalized block. The message ordering is permanent.
- **Availability Finality:** The availability certificate proves the data was retrievable at inclusion time.

5. Data Availability Model

5.1 The Challenge

Traditional blockchains place all data on the consensus critical path. Every byte included in a block must propagate quickly enough for validators to safely verify and vote, tightly coupling data throughput to block propagation latency.

5.2 Obsidian's Solution

Most DA solutions use Data Availability Sampling (DAS)—light clients randomly sample chunks to probabilistically verify availability without downloading everything. This lets resource-constrained nodes verify data is available without trusting validators.

Obsidian takes a different approach: the acceptance criteria for messages is committee attestation, and non-committee nodes trust that attestation or can optionally sample for additional confidence.

Aspect	DAS-Primary (Ethereum)	Committee-Primary (Obsidian)
Block acceptance	Requires successful sampling	Requires committee QC (2/3 threshold)
Primary verification	Light clients sample randomly	Committee members prove chunk possession (PoP)

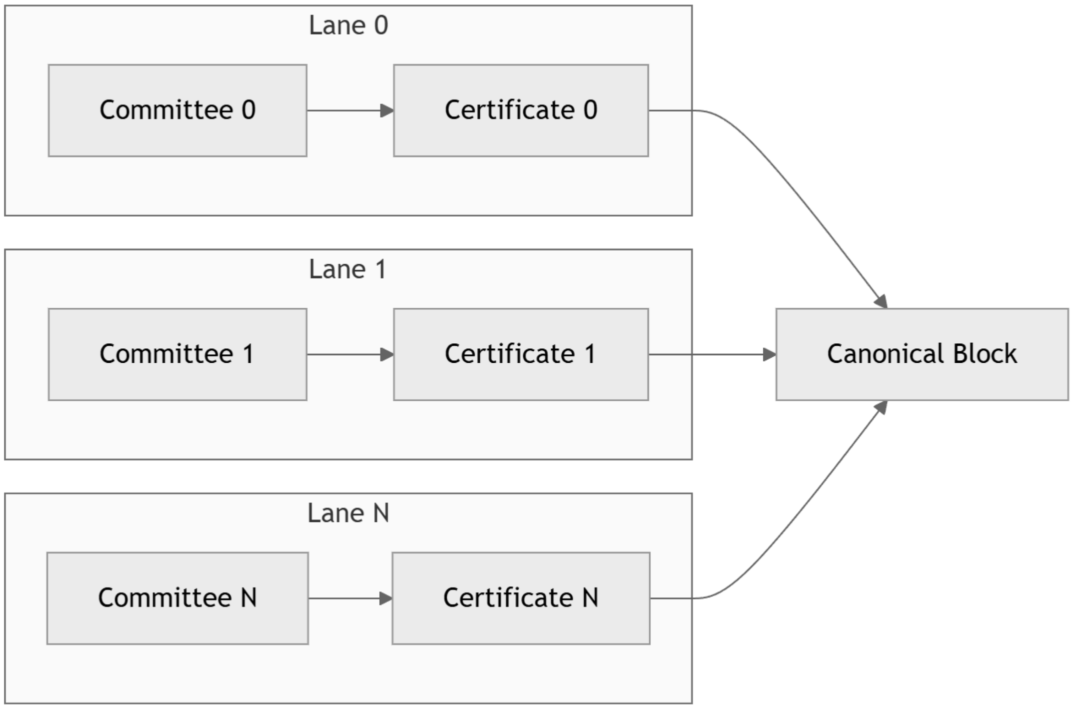
Parallelism	Single blob space per block	Multiple lanes with dedicated committees
Secondary confidence	N/A	Non-committee nodes can sample (optional)

5.2.1 The key difference

In Obsidian, a block is valid once the lane committee reaches quorum on the Availability Certificate. Non-committee nodes don't need to sample to accept the block, they trust the committee attestation. In DAS-primary systems, the block's validity depends on sampling succeeding.

5.2.2 Parallel Lane Model

1. Messages are routed to lanes (parallel channels)
2. Each lane has a rotating committee of validators
3. Committee members receive chunks and prove possession (PoP)
4. Votes aggregate into an Availability Certificate per lane
5. All lanes process simultaneously meaning throughput scales with lane count



5.2.3 Why Lanes Benefit Obsidian

- **High throughput**: Lanes parallelize DA verification
- **Low latency**: Committee votes are single-round, not iterative sampling
- **Bounded validator load**: Each validator only serves on a subset of lane committees
- **Deterministic acceptance**: Block validity depends on committee QC, not probabilistic sampling

5.3 Availability Certificates

An Availability Certificate contains: Reference to the lane batch, the data commitment, aggregated committee signatures, and signer bitmap.

A valid certificate proves that a supermajority of the lane committee possessed the data at the time of signing. Combined with erasure coding, this guarantees reconstructability.

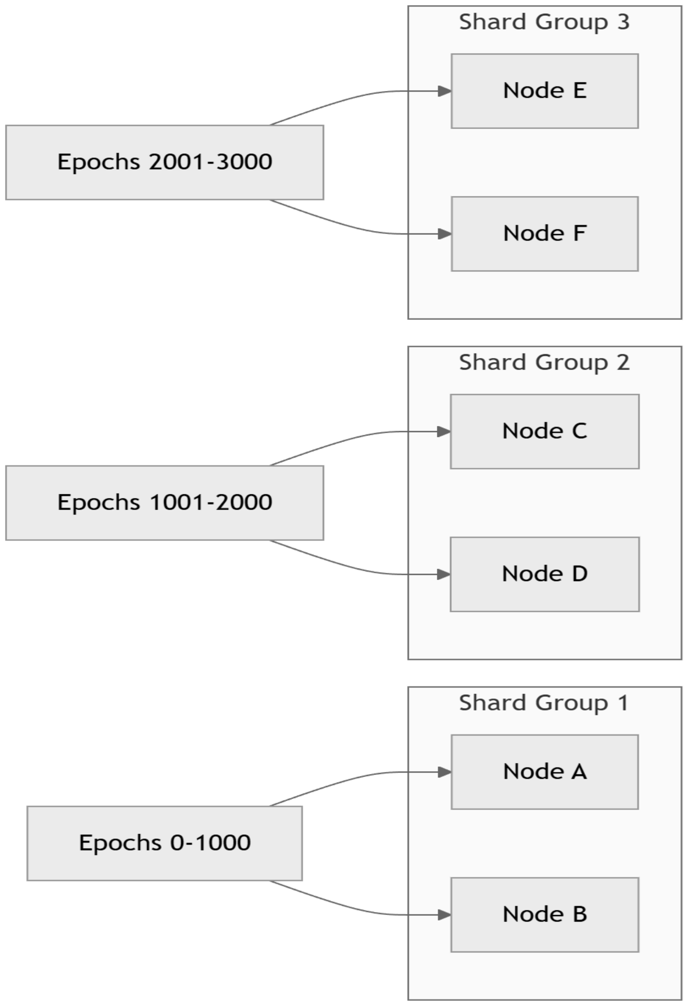
5.4 Retention Window

Committee members are obligated to serve data for a specified retention window after inclusion. After this window, data transitions to archival nodes (sharded and full). This keeps storage requirements for active validators relatively low while ensuring long term data availability.

5.5 Sharded Archive Nodes

High message throughput creates a storage scaling challenge. At maximum capacity, yearly data growth can reach in excess of 85 TB, which is far beyond what traditional "store everything" archive nodes can handle.

Obsidian solves this with sharded archives: instead of every archive storing all history, nodes store specific epoch ranges:



5.5.1 How it works

- **Epoch ranges**: History is divided into epoch ranges (e.g., 1000 epochs per shard)
- **Shard groups**: Multiple nodes store the same range for redundancy
- **Node assignment**: Archive operators are programmatically assigned based on current network needs
- **Proportional rewards**: Nodes earn rewards proportional to their coverage, uptime, etc.

This architecture enables:

- **Accessible participation**: Run an archive with consumer hardware by storing a subset of history
- **Horizontal scaling**: More epoch ranges served by adding shard groups
- **Redundancy**: Multiple nodes per shard group ensures availability

Full archives (storing everything) can still exist for complete historical access, but sharded archives dramatically lower the barrier to entry.

6. Message Types

Obsidian supports two message types distinguished by the “bid” field.

6.1 Priority Messages

Priority Messages include a fee bid and are sorted by bid amount (highest first) within each lane batch.

Characteristics:

- Bid > minBid qualifies as a Priority Message
- Sorted by bid within lane (highest bid = first in batch)
- Signed debit authorization
- Deterministic ordering for MEV resistance

Use Cases: Oracle updates, trading signals, time-sensitive alerts

6.2 Standard Messages

Standard Messages have no bid and require a VDF proof for spam resistance. They are ordered FIFO within each lane batch.

Characteristics:

- Bid == 0 or nil qualifies as a Standard Message
- Requires VDF proof for anti spam resistance
- FIFO ordering within lane
- No direct token cost, only small computation cost

Use Cases: Social posts, messaging, open participation, time insensitive votes, etc.

6.3 VDF Anti-Spam

Standard Messages require a Verifiable Delay Function proof. VDFs are computations that: take a minimum amount of sequential time to compute, can be verified quickly, and, cannot be parallelized or accelerated

This creates a natural rate limit: users must expend real-world time to submit messages, preventing spam floods without requiring monetary fees.

6.4 Message Expiration

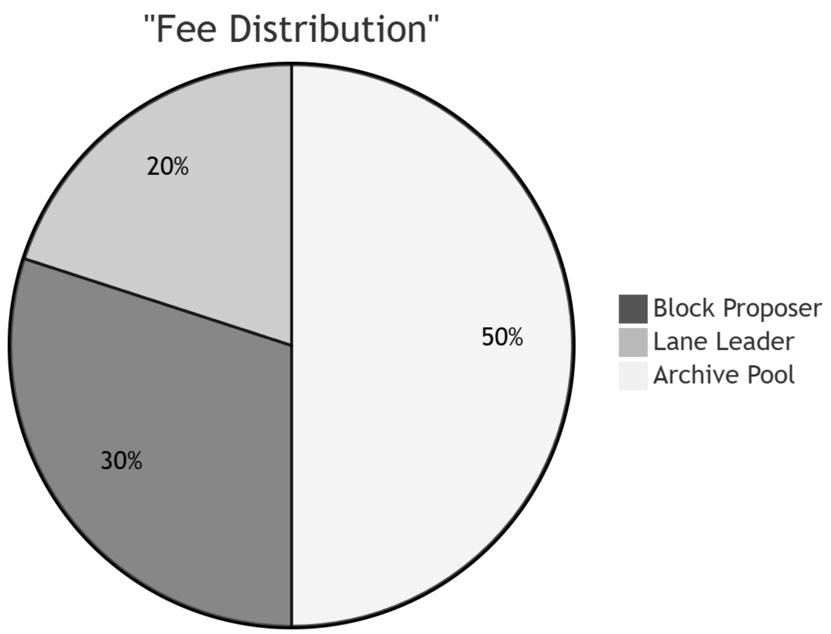
Messages specify a TargetBlock for inclusion. If not included within a validity window, the message simply expires, there is no rollover mechanism. This keeps the system simple and bounds resource usage.

Note: There is no global message queue. Each node maintains local per-lane buffers for pending messages. Messages are routed directly to lane committees via targeted P2P, not broadcast globally.

7. Economic Model

7.1 Fee Distribution

Priority Message fees are distributed among protocol participants:



Block Proposer: Compensated for including lane headers
Lane Leader: Compensated for batching, encoding, and distributing data
Archive Pool: Funds long-term data storage infrastructure

7.2 Signed Debit Authorization

Rather than prepaying fees, Priority Messages include a signed debit authorization. The proposer deducts the bid from the sender's balance at inclusion time. This eliminates stuck transactions and enables more predictable fee markets. This also prevents dropped Priority Messages from deducting fees and having to reconcile balances.

7.3 Validator Incentives

Validators earn rewards through multiple channels:

- **Block production:** Standard consensus rewards
- **Lane leadership:** Share of message fees when elected as lane leader
- **Archive Participation:** Share of message fees when participating as an archive node (sharded or full)

8. Node Roles

8.1 Block Proposers

Proposers create canonical blocks containing transactions, lane headers, and availability certificates. They verify header signatures and economics but do not need to download full sidecar data.

8.2 Validators

Validators participate in consensus and serve on lane committees. Committee assignment rotates each epoch based on randomness derived from the beacon chain.

8.3 Lane Leaders

Each slot, each lane has an elected leader responsible for:

- Collecting messages for that lane
- Forming and encoding the batch
- Distributing chunks to committee members
- Aggregating votes into a quorum certificate

8.4 Lane Committees

Committees verify data availability for their assigned lane. Members must:

- Download and store assigned chunks
- Respond to sample requests with proofs
- Sign availability votes

8.5 Archival Nodes

Archival nodes store data beyond the active retention window. They serve historical queries and provide long-term data persistence. The archive pool (funded by fees) incentivizes this role due to the increased importance of the archival role compared to standard transaction oriented chains.

9. Security Model

9.1 Assumptions

Obsidian's security relies on standard blockchain assumptions:

- **Honest majority:** A supermajority of stake is controlled by honest validators
- **Committee honesty:** For each lane, a supermajority of the assigned committee behaves honestly during availability voting.
- **Network synchrony:** Messages propagate within bounded time during normal operation
- **Cryptographic hardness:** Hash functions and signatures remain secure

9.2 Guarantees

When the security assumptions hold, Obsidian provides the following guarantees:

- **Inclusion:** Once a lane header is in a finalized block, the message ordering is permanent
- **Availability:** A valid availability certificate proves data was available at inclusion time
- **Integrity:** Data commitments prevent tampering after inclusion
- **Censorship Resistance:** Multiple lanes and rotating leaders reduce censorship risk. Users are deterministically assigned to lanes, so targeted censorship requires colluding leaders across multiple slots.

9.3 Threat Mitigation

The protocol includes defenses against common attack vectors:

- **Malicious leaders:** Committee voting ensures leaders cannot forge availability
- **Data withholding:** Erasure coding allows reconstruction from partial data
- **Spam attacks:** VDF proofs and fee mechanisms rate-limit submissions
- **Committee collusion:** Random assignment and supermajority thresholds

10. Developer Experience

10.1 For Application Developers

Obsidian provides a native message submission API alongside standard Ethereum RPC:

- **eth_sendMessageBlob:** Submit a signed message
- **eth_getBlockMessages:** Retrieve all messages from a specific block
- **eth_getMessageByHash:** Look up a message by its hash
- **eth_getMessagesByAddress:** Query messages by sender within a block range
- **eth_getMessageWork:** Get VDF parameters for Standard Message submission

Applications can treat Obsidian as both an EVM execution environment and a data publication layer, using the appropriate pathway for each workload.

10.2 For Infrastructure Operators

Obsidian requires dedicated client software:

- **Obsidian-Geth:** Modified execution layer client with Silica Protocol integration, message pool, and lane routing
- **Obsidian-Lighthouse:** Modified consensus layer client with lane committee logic, PoP voting, and availability certificate validation
- **Archival Nodes:** Optional sharded or full archive for historical message data

Note: While Obsidian inherits Ethereum's architecture (EVM, beacon chain consensus), the Silica Protocol modifications are substantial. Operators must run Obsidian-specific clients, standard Ethereum clients will not sync with the network.

11.1 What Obsidian Is Not

- **Not a general-purpose storage network:** Obsidian is optimized for small, frequent data publication rather than large static files or bulk archival storage.
- **Messages are data-only:** Messages store signed data but do not execute code. All smart contract logic is handled through standard EVM transactions.
- **Not zero-cost:** Standard messages require small computational work for spam prevention, while priority messages require fees for expedited inclusion.

11.2 Current Limitations

- **Retention window:** Active validators store message data for a bounded period. Long-term retrieval relies on archival and indexing infrastructure.

- **Message size:** The protocol is optimized for kilobyte-scale payloads. Larger datasets must be chunked across multiple messages or stored externally.
- **Lane capacity:** Each lane has finite throughput. High-volume senders may see messages expire before inclusion during congestion.

12. Conclusion

Obsidian Chain addresses the gap between high-latency decentralized storage and low-latency centralized services by providing a native, protocol-level data publication pathway optimized for small, frequent messages.

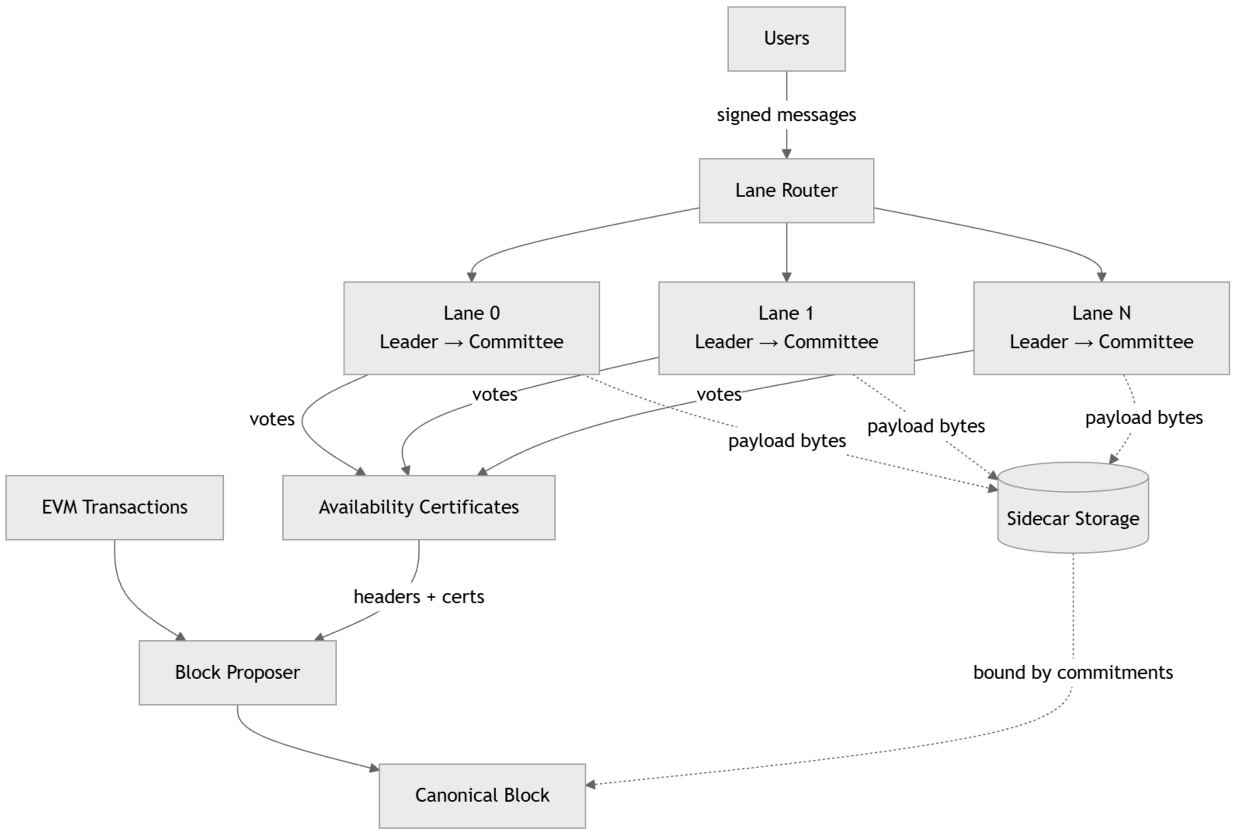
12.1 Key Architectural Innovations

- **Separation of concerns:** Execution and data availability scale independently
- **Silica Protocol:** Committee-based data availability with erasure coding
- **Dual message types:** Fee-based (Priority) and compute-based (Standard) inclusion pathways
- **Commitment-based identity:** Consensus operates on commitments, not full payloads

By treating data publication as a priority, not an afterthought, Obsidian enables a new class of decentralized applications that require both the verifiability of blockchain and near-real-time responsiveness compared to storage-oriented networks.

Appendix A: Architectural Diagrams

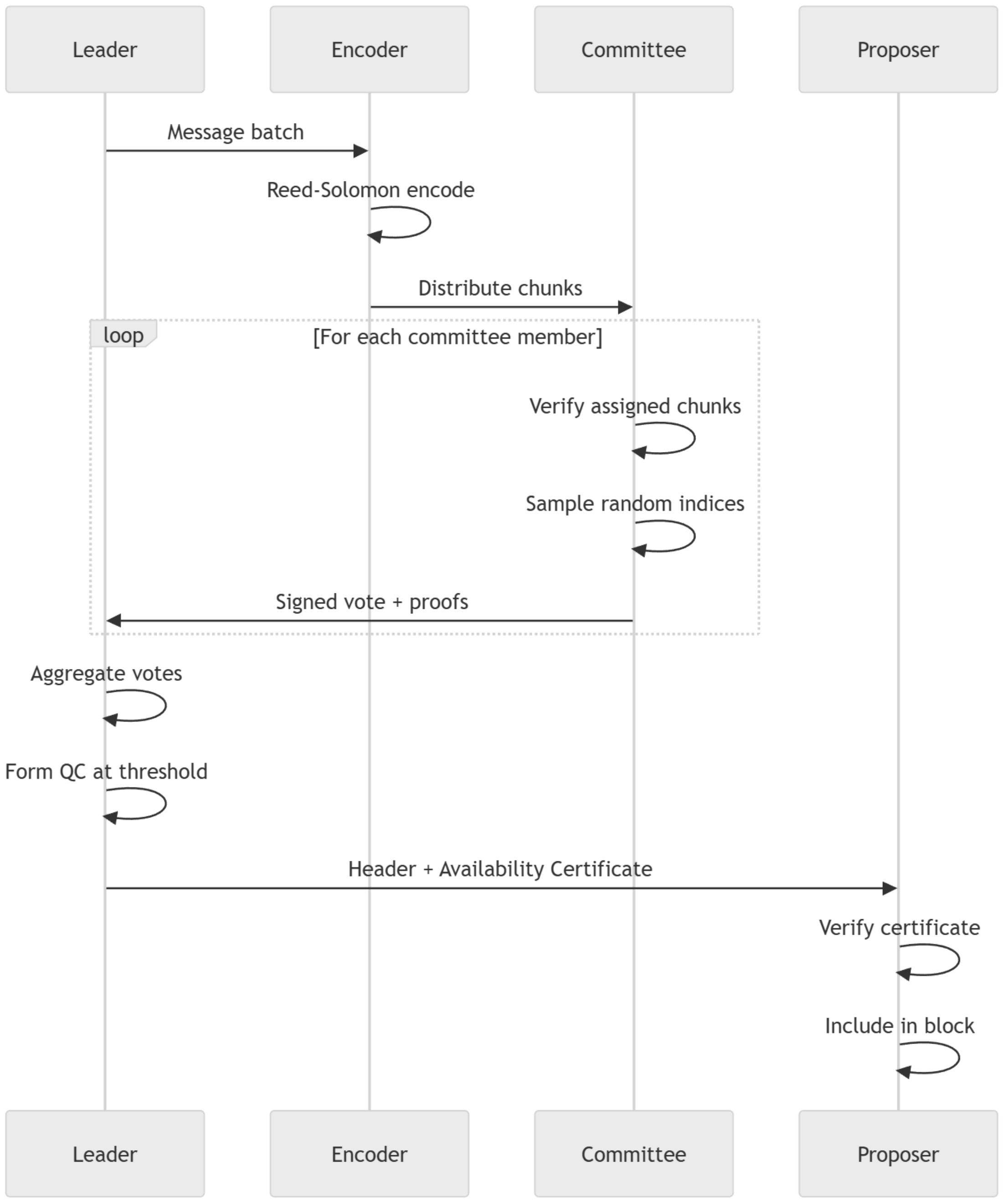
A.1 System Overview



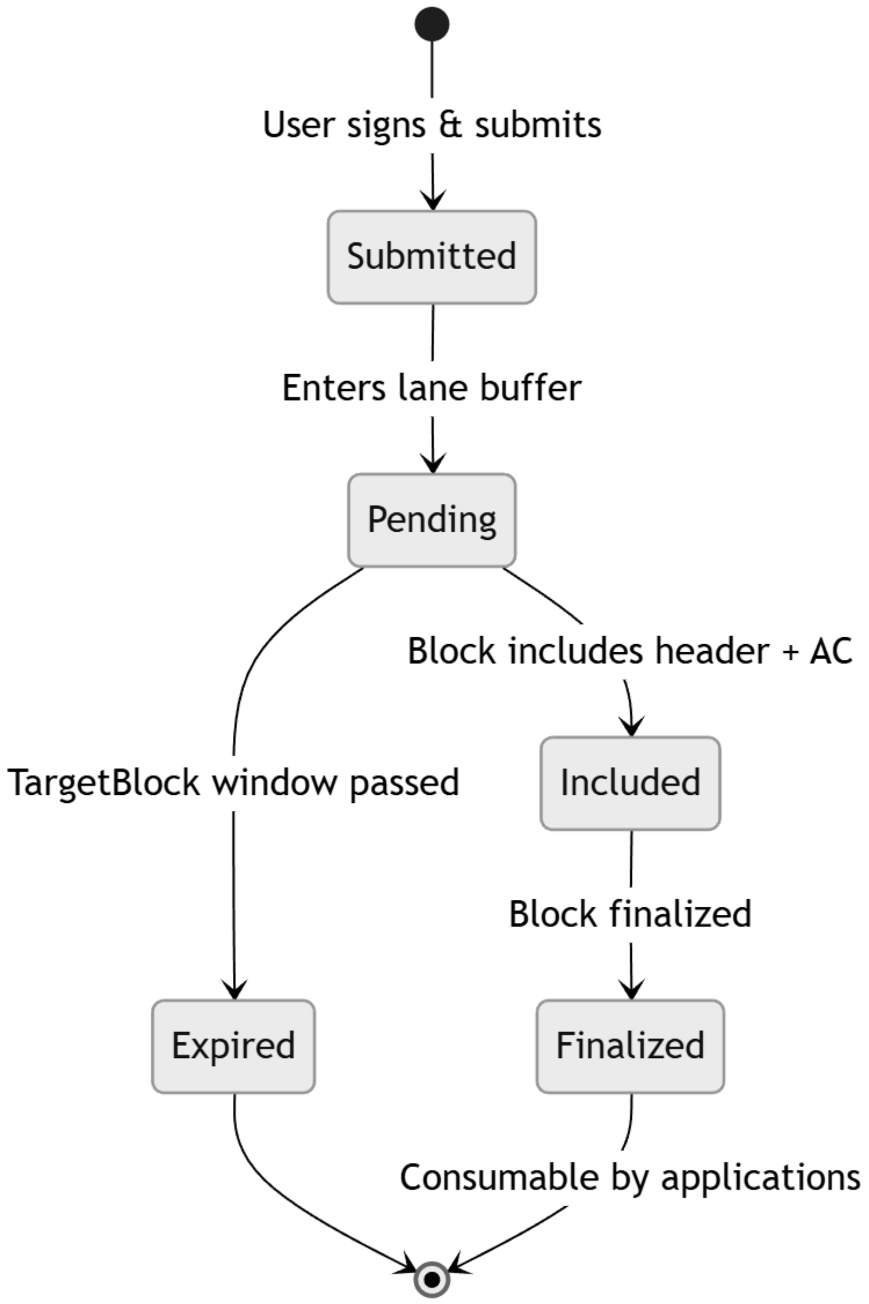
Flow summary:

1. Users submit signed messages to the network
2. Lane Router assigns messages to lanes based on sender address hash
3. Each lane's Leader batches messages and erasure-codes them into chunks
4. Committee members receive chunks and vote on availability (PoP)
5. Availability Certificates aggregate committee signatures
6. Block Proposer includes only compact headers + certificates (not payloads)
7. Payload bytes live in sidecars, referenced by commitments in the block

A.2 Data Availability Flow



A.3 Message States



Appendix B: Glossary

Term	Definition
Archival Node	Node responsible for long-term storage and retrieval of historical message payloads beyond the retention window.
Availability Certificate (AC)	Cryptographic proof that a supermajority of a lane committee attested to data availability.
Availability Finality	Point at which message data is confirmed retrievable via a valid Availability Certificate.
Canonical Block	Single block produced per slot that determines ordering, execution, and finality for transactions and message commitments.
Canonical Ordering	Permanent ordering of messages determined by inclusion of lane batch headers in finalized blocks.
Committee Quorum	Threshold of committee signatures (2/3) required to form a valid Availability Certificate.
Data Commitment	Merkle root of erasure-coded chunks that commits to sidecar content.
Erasure Coding	Redundancy technique allowing data reconstruction from partial chunks.
Inclusion Finality	Point at which message ordering becomes permanent due to block finalization.
Lane	Parallel processing channel to which messages are deterministically routed by sender address.
Lane Batch Header	Compact header containing message ordering, data commitments, and economic metadata; included in the canonical block.
Lane Committee	Subset of validators responsible for verifying data availability for a specific lane.
Lane Leader	Validator elected per slot to batch, encode, and distribute messages for a lane.
Lane Router	Deterministic routing logic that assigns messages to lanes based on sender address.
Message Commitment	Cryptographic reference included in the canonical block that binds a message batch to its payload bytes without embedding the data itself.

Message ID	Deterministic hash identifying a message, derived from sender, payload commitment, nonce, and signature.
Micro Root	Merkle root of message identifiers that commits to message ordering within a batch.
PoP (Proof of Possession)	Proof provided by committee members demonstrating possession of assigned erasure-coded chunks prior to voting.
Priority Message	Message with bid greater than MinPMBid; fee-based and ordered by bid amount within a lane batch.
Reed–Solomon Encoding	Error-correcting code that splits data into redundant chunks such that any subset above a threshold can reconstruct the original payload.
Retention Window	Period during which validators are required to store and serve message data after inclusion.
Sidecar	Data object containing message payloads; propagated separately from the canonical block body.
Sidecar Chunks	Individual erasure-coded fragments of a sidecar distributed across lane committee members.
Silica Protocol	Obsidian’s data availability subsystem responsible for message routing, encoding, distribution, and availability verification.
Standard Message	Message with zero bid that requires a VDF proof and follows FIFO ordering within its lane.
TargetBlock	Block number a message targets for inclusion; message expires if the validity window passes.
VDF (Verifiable Delay Function)	Proof of sequential computation that cannot be parallelized and can be efficiently verified.