The Brain for Agentic and Physical AI

Composable Memory with Immutable Proofs

The Lighthouse Team
`lighthouse.storage`
September 2026

Abstract

Production agents fail less often because they cannot generate language than because they cannot remember with integrity. Context windows reset at the end of a session. Vendor memory interfaces lock state inside a single runtime. Ordinary files have no recall model, no content-addressed proof, and no life beyond the machine that wrote them. Physical systems such as robots, vehicles, and industrial controllers add a stricter constraint: memory must survive firmware replacement, fleet rotation, intermittent connectivity, and operators who do not share a cloud account.

Lighthouse is a protocol and platform for composable long-term memory for agentic and physical AI. Every write produces an immutable, content-addressed proof on IPFS. Persistence is delegated to a storage fabric the operator chooses among Filecoin, Walrus, Lighthouse-operated IPFS pinning, or a backend the operator brings. The recall model is pluggable. Embeddings are computed on the Lighthouse backend against a configured model, and any compatible open or closed model can be substituted, whether it is published on a hub such as Hugging Face, held as a private checkpoint, or served behind an API.

HOUSE is the unit of settlement and security of that memory network. A remember pays HOUSE for the term the batch is stored. Recall is work paid and collateralized in HOUSE. Replication across uncorrelated backends is a HOUSE-denominated claim on a CID. The token is not a license that unlocks a hosted product; it is how the network prices a write, wages a recall, attributes a namespace, and slashes a keeper that serves an index the blobs do not support.

The protocol commits to durable, verifiable memory over a contracted term, to automatic renewal while a plan is active, and to recovery of an agent's written records from the network with the namespace API key. As of September 2026 the underlying network holds 16 TiB+ of stored data, 33k+ users, and 9.5M+ objects (live figures at `https://explorer.lighthouse.storage`).

1. Introduction

Buyers of agentic and physical AI are not primarily shopping for an object store. They need a durable mind: the ability to recall what was learned, to attribute that state to an agent or a fleet, to prove that the bytes have not been rewritten, and to leave with those bytes intact. Storage is the substrate on which that mind persists. Memory is the interface through which agents use it. The product is the brain those two layers make possible.

Lighthouse is built around that order. Agents receive three primitives (*remember*, *recall*, and *forget*). Flushed batches are content-addressed with an IPFS CID [1]. Bytes land on a storage fabric the operator selects [2, 3]. The embedding model used for recall is an adapter rather than a fixture [4, 5]. Files, S3-compatible ingestion, gateways, and long-dated persistence remain first-class surfaces on the same account and CID space, because a working memory still has to live somewhere durable.

This paper specifies the problem, the architecture, the composability points, the proof model, the storage fabric, commercial terms, token utility, operational risks, and the build order. Durability claims throughout are service-level commitments over a contracted term, not unbounded promises.

Table 1: Product stack. A durable mind is the utility at the top of the stack. Memory, aggregation, and storage are the supporting layers.

Layer	Utility	Why it sits there
Brain	A durable mind for agents and machines: recall, identity, proof, and portability.	The category autonomous systems lack and that no object store sells on its own.
Memory	The primitives remember, recall, and forget, with a model the operator chooses.	The interface of the brain. Storage is how memory persists.
Storage aggregator	One addressing model and one bill across Filecoin, Walrus, IPFS, and bring-your-own backends.	So the brain is not captive to one network's economics or failure mode.
Durable storage	Long-dated persistence with proofs, offered as a first-class surface.	The foundation the layers above stand on. Files, archives, and compliance workloads live here.

2. Why agent memory fails in production

2.1. Stateless minds on stateful work

Production agents are asked to hold customer history, tool schemas, runbooks, robot calibration, safety constraints, and the residue of prior decisions. The industry's default answers do not give that state integrity, portability, or a life independent of one vendor.

- **The context window.** It is finite, expensive, and wiped at the end of a session. Prompt stuffing is not memory; it is amnesia with a larger buffer.
- **Vendor memory APIs.** They are convenient inside one lab's runtime, but the state is invisible, difficult to export, and priced as a tax on staying. An agent that can only remember inside one vendor is a feature of that vendor.
- **A vector database on someone else's cloud.** This offers recall without persistence guarantees, without content addressing, and without a proof that the index queried is the index that was written.
- **Files on disk, or objects in object storage.** These are durable in the operational sense and blind in the cognitive sense. They have no embedding model, no namespace semantics, and no reconstructable proof for a robot that has lost its onboard disk.
- **On-chain dumps.** They are verifiable and, as a working memory, ruinously expensive.

2.2. What physical AI adds

A software agent that forgets is an operational nuisance. A physical agent that forgets is a safety problem. Warehouse robots, surgical assistants, field drones, inspection crawlers, farm implements, and fleet vehicles do not merely chat across sessions. They accumulate state that is expensive to re-derive and dangerous to invent: proprioceptive priors after a bump or a bearing change, site maps that differ by building and shift, gripper and tool calibration, forbidden-zone geometries, operator preferences, and the incident history that safety reviews later treat as evidence.

That state has to satisfy constraints software agents rarely meet.

- **It outlives the compute unit.** A robot is refurbished. An ECU is swapped. A cart is factory-reset. A drone is rotated through a spare pool. The mind has to reconstruct on hardware that has never seen the

original disk, under a serial number that is not the serial number that wrote the bytes.

- **It cannot assume a permanent tether.** Plants go dark. Mines lose backhaul. Ships operate for days between satellite windows. Air-gapped lines never speak to a hyperscaler. Memory must buffer on the unit, flush when a path exists, and remain addressable by CID when the machine returns.
- **It is shared at the site and private at the unit.** A floor can share a map and a safety envelope. A particular picker should not leak another unit's episodic trace. Fleet rotation makes that split mandatory: site memory follows the building; episodic memory follows the mind, not the hull.
- **It is evidence.** After an incident an auditor asks what the machine knew at a time, not what a vendor dashboard now displays. A content-addressed batch bound to a Filecoin deal or a Walrus epoch is an answer. A mutable row in a cloud vector index is not.
- **It is written by operators who do not share a cloud account.** Contractors, OEMs, and the plant IT team are different parties. Memory that lives only inside one hyperscaler account is unusable the moment the fleet crosses an organizational boundary.

Physical AI therefore needs the same properties software agents need, held to a higher bar: portability across hardware, proof of what was known at a given time, independence from any one storage or model vendor, and a write path that tolerates hours or days of disconnection.

2.3. Memory classes on a machine

The records a physical agent writes are not one blob type. Treating them as chat turns is how robot memory products fail in the field.

Table 2: Memory classes physical agents accumulate. Lighthouse stores each class as tagged records in the same CID space; composition and access differ by class.

Class	What it holds	Why it is distinct
Semantic / site	Maps, forbidden zones, dock geometry, shift playbooks.	Shared across a fleet; changes slowly; must survive unit rotation.
Episodic / unit	What this machine did, saw, and decided on a shift.	Private to the unit mind; high volume; the audit trail after an incident.
Procedural	Tuned grasps, approach trajectories, wear-adjusted offsets.	Numeric and multimodal; worthless if the model geometry is swapped silently.
Safety envelope	Interlocks, load limits, keep-out regions, last near-miss.	Must reconstruct after a firmware flash; forgetting it is a hazard.
Calibration	IMU bias, camera extrinsics, joint zeros, tool-center points.	Expensive to re-derive; invalid after a collision or a part swap.

Lighthouse does not invent a robotics file format. It gives those classes a common write, proof, and recall interface: namespaced by fleet or site and by unit, flushed as incremental batches, addressed by CID, reconstructible after a reset, and—when Kavach covers the memory rail—encryptable so a leaked CID is not a leaked site map. The on-device pending buffer and opportunistic flush exist because a physical agent cannot treat "the gateway is reachable" as an invariant.

3. System overview

Lighthouse is four layers held together by one addressing model.

A memory runtime. Agents receive *remember*, *recall*, and *forget*. Memories are namespaced by application and by agent identity, tagged, buffered, and flushed as incremental batch blobs. Embedding and recall run on the Lighthouse backend against the configured model. Scoring combines cosine similarity with keyword and tag overlap. A bundled Model Context Protocol (MCP) server exposes the same primitives to Claude Code, Claude Desktop, and any MCP-capable runtime [6], so memory is not an SDK-only feature. Physical units use the same primitives through an on-device buffer that flushes when connectivity returns.

A proof layer. Every flushed blob is content-addressed with an IPFS CID. That CID is the stable identifier of the memory batch across backends, gateways, and recovery. An auditor, a regulator, or a downstream agent can fetch the bytes and verify that they match the identifier. The difference is between a vendor assertion that an agent remembered X and an object that contains X bound to a hash anyone can check.

A storage fabric. The fabric includes Lighthouse-operated IPFS pinning for hot retrieval, Filecoin for deal-backed cold persistence, and Walrus on Sui for blob storage with epoch economics. Routing is explicit at write time or set as a workspace default. Because the CID does not change when the backend does, migration is a data movement rather than a re-addressing exercise. Operators may bring their own storage behind the same memory interface—including an air-gapped volume that never leaves a factory.

A platform that already exists. Memory sits on infrastructure already operated at production scale: dedicated gateways, an S3-compatible endpoint (L3), Kavach (Threshold Encryption) and token-gated access for files, IPNS, image and video delivery, and x402 pay-per-request settlement. As of September 2026 the network holds 16 TiB+ of stored data, 33k+ users, and 9.5M+ objects [7]. Memory is a new interface to that substrate rather than a greenfield network.

4. Architecture

The stack is deliberately thin. Lighthouse does not re-implement storage-provider slashing and does not invent a new embedding standard. It owns the seams between agent, memory, proof, and fabric.

Table 3: Four layers. Agents speak memory. Memory speaks proofs. Proofs land on a fabric the operator chooses.

Layer	Contents
Agent surface	MCP, TypeScript SDK, REST, x402, and physical-AI adapters
Memory brain	remember / recall / forget; pluggable models; namespaces; composable stores
Proof layer	IPFS CIDs; incremental batch blobs; deal and epoch records; network-addressable recovery
Storage fabric	Filecoin; Walrus; IPFS hot pinning; bring-your-own storage

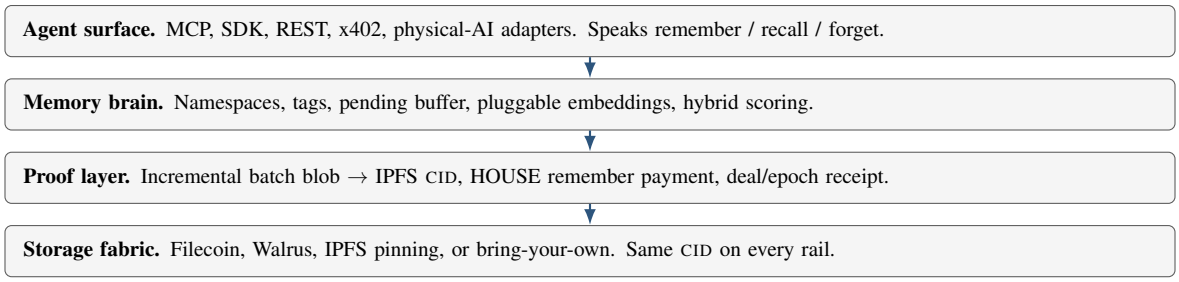

Figure 1: One addressing model. Agents never speak a backend identifier; they speak memory. The CID is the seam.

4.1. Write path

1. The agent calls *remember* with content and metadata. The record is accepted, tagged, and queued. Embedding is computed on the Lighthouse backend against the configured model. A physical unit that is offline accepts the same call into a local pending buffer and does not block the control loop on a gateway.
2. Pending records flush as a single JSON batch blob once a threshold is reached (default: ten memories) or when the agent flushes explicitly. The blob contains only the new records. It does not carry a complete index snapshot. Packing a full snapshot into every flush was the prior design and grew with store size; the current path writes incremental batches.
3. The blob is uploaded to the selected backend through Lighthouse. The backend returns its native handle, such as a Filecoin deal reference or a Walrus blob ID. Lighthouse records the IPFS CID as the canonical identifier, settles a HOUSE remember payment against that CID for the contracted term (Section 10), and exposes a gateway URL.

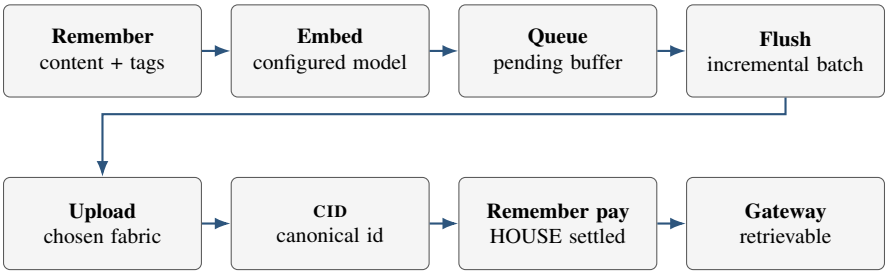

Figure 2: Remember path. The agent writes a record. The network writes a CID and settles a HOUSE remember payment. Offline units stop at the pending buffer until flush.

4.2. Read path

Recall runs against Lighthouse's in-process index on the backend, covering flushed records plus any pending records held for that namespace. A user query therefore requires a network round trip, except where a physical unit answers from its local buffer first. Scoring is a weighted hybrid of cosine similarity and keyword or tag overlap, with a default of 0.7/0.3. If the embedding model is unavailable the runtime degrades to keyword scoring rather than failing closed. A single record can be fetched by identifier. Native backend identifiers for Sui or Filecoin tooling are available on the same record.

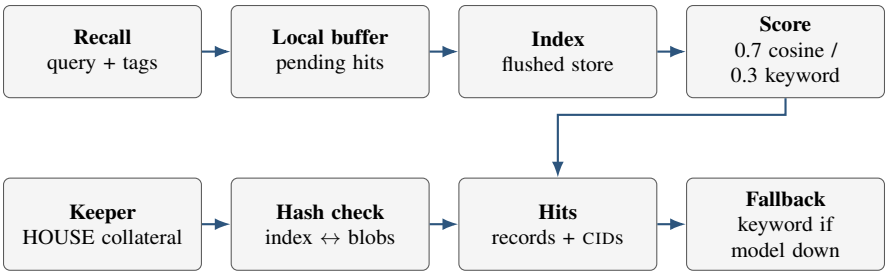

Figure 3: Recall path. Physical units answer pending records locally first. A keeper that serves an index the blobs do not support is slashed.

4.3. Recovery path

Memory blobs remain addressable by CID and can be re-read from the network. The search index is an operational store maintained by Lighthouse; it is not reconstructed from a snapshot packed into each flush. Namespace ownership is the API key at the product surface and a HOUSE-backed namespace record at the protocol surface (Section 10). Portability is the CID-addressed blobs together with that key and record, rather than a self-contained backup inside every write. A refurbished robot with the namespace credential can rebuild what the prior hull wrote.

4.4. Physical-AI path

The physical path is the write and recovery paths under disconnection, unit rotation, and mixed operators.

- **On-device buffer.** *Remember* succeeds locally. The control loop does not wait on IPFS, Filecoin, or Walrus. Pending records are searchable on the unit immediately and are not durable until flush.
- **Opportunistic flush.** When a gateway, a dock, or a plant uplink appears, the buffer uploads incremental batches and receives CIDs. Flush can be duty-cycled so a battery-constrained unit does not pay radio cost on every record.
- **Site versus unit namespaces.** A fleet mounts a site store read-only (maps, envelopes, playbooks) and keeps an episodic store per unit. Rotation moves the episodic namespace to the next hull; the site store stays with the building.
- **Bring-your-own on-prem.** An air-gapped line attaches a local volume as the persistence adapter. Lighthouse still issues the CID, maintains the index when the unit is later docked, and does not require the bytes to leave the plant.
- **Incident reconstruct.** Given the CID list and the namespace credential, an auditor re-reads what was written, not what an operator later edited in a dashboard.

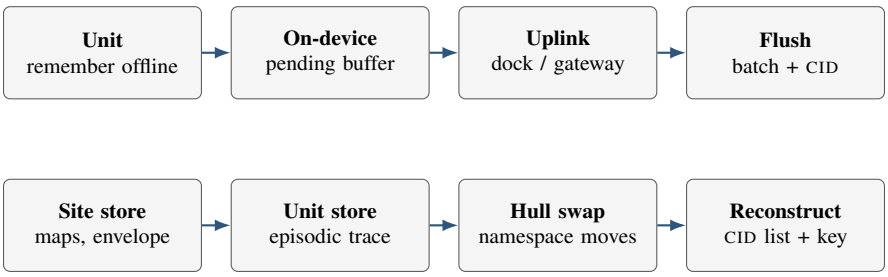

Figure 4: Physical-AI path. Site memory stays with the building. Episodic memory follows the mind across a hull swap. Flush waits on connectivity.

5. Composable memory

Composability, in this protocol, means three substitution points in the runtime, each with a stable interface so that swapping one does not require rewriting the others.

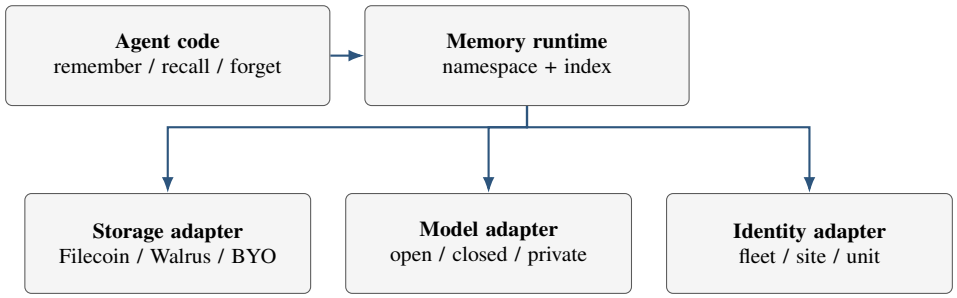

Figure 5: Three substitution points. Swap a backend, an embedding model, or a namespace scheme without rewriting the agent.

5.1. Storage adapters

A store is an adapter that can put a batch blob, get a blob by CID, list a namespace, and report native identifiers. Shipped adapters cover IPFS with Filecoin and IPFS with Walrus. The operator chooses the rail at workspace or write time. Filecoin is a natural fit for incremental memory batches whose size tracks the object itself. Walrus is the natural fit when the workload needs Sui-native blob IDs or objects large enough to justify its encoding model.

A bring-your-own adapter is a first-class engineering target. Any backend that can hold an addressed blob and return it later (another decentralized storage network, an S3 bucket the customer already pays for, or an air-gapped volume on a robot) should be attachable without changing agent code. Cross-backend replication is already a product option on the storage side and becomes a durability posture for memory: the same CID persisted on uncorrelated networks, with the replication claim bonded in HOUSE.

5.2. Model adapters

The default recall model is `all-MiniLM-L6-v2` [4, 5], run in-process on the Lighthouse backend. Embedding therefore happens on Lighthouse infrastructure rather than on the caller's machine, and content reaches the backend at *remember* and *recall* time. That default is a convenience, not a constraint. The interface accepts any embedding function that maps text to a vector: an open model from a public hub such as Hugging Face, a closed or proprietary model behind an API, a domain-fine-tuned checkpoint for robotics telemetry or clinical notes, or a privately hosted endpoint. The index stores the model identifier alongside the vectors so a store is not silently queried with the wrong geometry. Operators who need embeddings to stay off Lighthouse infrastructure can bring their own model endpoint or encrypt before *remember*.

Physical workloads make that identifier mandatory. A procedural memory embedded under one vision-language geometry is not recallable under another. Changing the onboard model is an explicit re-embed, not a silent remap.

5.3. Identity and namespace adapters

Memories are isolated by namespace (the application or fleet) and agent (the particular mind). A support bot and a warehouse picker on the same account do not share a store unless an operator composes them. Composition is explicit: stores can be mounted read-only into another agent, tagged subsets can be exported as their own CID, and a physical fleet can share a site-level namespace while keeping per-unit episodic

memory private.

5.4. How agents call it

Builders integrate through the published Memory SDK (`@lighthouse-web3/memory`) [8] or through MCP tools for remember, flush, recall, list, get, forget, rebuild, and status. An agent that never imports the SDK can still have a store. Documentation for the SDK and MCP tools is published with the Memory product.

5.5. Kavach and encrypted memories

Kavach is Lighthouse's Threshold Encryption SDK [9]. It implements threshold cryptography over a 3-of-5 shard set distributed across five independent nodes. No single node can reconstruct the key, and two nodes can be offline without losing recoverability. Access can be gated by wallet, token balance, NFT ownership, time lock, or a custom contract. Files already use Kavach for encrypted upload and token-gated retrieval.

Memory encryption is being designed on the same rail. The intended design is that the record is encrypted (by the agent, or by Lighthouse under a Kavach-managed key) before the batch is written; the CID addresses ciphertext; and authorized callers reconstruct the key from shards and decrypt at recall. Until that path ships, operators who need confidentiality can apply their own client-side encryption before *remember*. Secrets, credentials, and regulated personal data should not be placed into memory in plaintext. Site maps and safety envelopes on a shared floor are the physical-AI case that makes this rail urgent.

Encrypted memories change what a CID proves. The hash still binds the bytes. It does not, by itself, reveal the plaintext to a holder of the CID. Forgetting remains a tombstone on later batches, and earlier ciphertext blobs stay addressable. Key revocation and shard expiry are therefore part of the memory-encryption design rather than an afterthought.

6. Immutable proofs

A memory system without proofs is a diary the author can rewrite. Lighthouse treats the CID as the unit of integrity.

6.1. What is proven

- **Content integrity.** The bytes of a batch blob hash to the CID. A silently edited memory fails the address.
- **Existence at a time.** On Filecoin, deal records and, where available, proofs of data possession bind the CID to a provider and a term. On Walrus, blob IDs and epoch receipts do the same on Sui. Lighthouse emits the renewal record (deal or epoch, provider, term, and proof reference) onto the customer's account, and records a HOUSE remember payment against the same record.
- **Reconstructability.** Incremental batch blobs remain addressable. A party with the CID list and the namespace credential can re-read the written records. The search index is maintained separately and is not implied by any single blob.
- **Non-equivocation of identity.** Namespace and agent identifiers travel inside the blob, so a reconstructed store is attributable.

6.2. What is not proven

A CID does not prove that an embedding was computed correctly, that a tag was honest, or that the agent ought to have remembered the fact. Those are application-level claims. Lighthouse proves the object, not the wisdom of writing it. Forgetting is a tombstone in subsequent batches rather than an undo of history:

earlier blobs remain addressable. That property is useful for audit and a constraint for secrecy, which is why encryption of the memory rail matters for sensitive and physical-AI workloads.

6.3. Why IPFS is the proof substrate

Filecoin and Walrus already have native identifiers. Lighthouse still issues an IPFS CID for every object, including objects whose bytes live only on Walrus, so that one addressing scheme spans the fabric. Gateways, S3 metadata headers, memory receipts, and recovery all speak CID. The proof remains portable even when the backend is not.

7. Storage fabric

The memory runtime should not care which network holds the bytes. The operator should. Lighthouse makes that split explicit.

Table 4: Backends as interchangeable persistence, unified by CID.

	IPFS (hot)	**Filecoin**	**Walrus**	**Bring your own**
Role	Pinning and retrieval	Deal-backed cold tier	Sui blob layer	Customer-owned backend
Addressing	Native CID	CID + deal ref	Blob ID, CID-mapped	CID mapped by Lighthouse
Term	Continuous pin	Finite renewable deals	Epoch, extendable	Whatever the backend sells
Durability	Lighthouse nodes	SP collateral / slashing	Sui economic guarantees	Customer's contract
Best for	Hot recall, gateways	Archive, compliance	Sui-native agents, large blobs	Air-gap, existing spend, other DSNs

7.1. Routing

Backend choice is a function of object-size distribution (Walrus's per-blob overhead makes tiny standalone objects uneconomic unless they are batched, which memory already does), access frequency, ecosystem alignment, and price. Published plan pricing differs per rail and is repriced as backend costs move. Because the CID is invariant, a store can be migrated or dual-written later.

7.2. Bring-your-own storage

Agnosticism that stops at two networks is still a vendor list. Bring-your-own storage means an operator can attach a backend Lighthouse does not operate: another decentralized network, a bucket already sitting in a procurement contract, or a volume that never leaves a factory. Lighthouse continues to provide the memory runtime, the CID, the backend index, the MCP server, and, if the operator wants it, the gateway. The operator provides put and get and pays the backend directly.

Lighthouse does not re-collateralize storage providers. Filecoin and Walrus already slash. Duplicating that layer would mean posting capital to re-guarantee a guarantee that already exists, and then adjudicating breach. The role is orchestration: place the object, renew the term, verify retrievability, move data when a provider degrades, and surface the proof to the customer. HOUSE does not re-bond that inherited slashing. Remember pays for placement. Bonds sit only where this network takes a risk the backend does not: a keeper that might serve the wrong index, or a replication claim that a second copy stays live.

8. Interfaces

Memory is the centre of gravity. The rest of the platform is how that memory is fed, billed, retrieved, encrypted, and governed. Storage remains a first-class product surface on the same account and CID space.

Table 5: One account, one CID space, several interfaces.

Surface	What it is	Who it is for
Memory SDK and MCP	remember / recall / forget, backend embeddings, incremental batches, CID proofs.	Agent builders, robotics stacks, MCP-capable runtimes.
Files app	Upload, organize, share, and pay across IPFS, Filecoin, and Walrus on one account.	Teams and individuals storing files.
L3 (S3 API)	`s3.lighthouse.storage`. SigV4, buckets, multipart, presigned URLs, CID in metadata.	Enterprises with an existing S3 pipeline.
Kavach (Threshold Encryption)	Threshold encryption SDK: 3-of-5 shards, token gating, contract conditions. Shipping for files; extending to memory.	Objects and memories that must be public on a network and private to a holder.
x402	HTTP 402 pay-per-request in stablecoins, without account provisioning in the loop.	Autonomous agents settling retrieval or writes.
Gateways	Dedicated IPFS retrieval, video streaming, and on-the-fly image resize.	Workloads that need fast retrieval.

These surfaces share billing, identity, and access control. They are not separate protocols. An agent that remembers via MCP, a pipeline that writes via L3, and a human who checks the Files app are looking at the same objects.

9. Pricing and durability

Metering sits at the core of the commercial model. Actions and persistence are priced as they occur. Subscriptions sit on top of that meter as an application-level convenience: a team can pre-commit to a monthly envelope so that agents do not have to settle every call, while the underlying bill is still a function of use.

Metered usage. Remember, recall, forget, and the bytes those actions place or retrieve are the units. Settlement can be an invoice, a prepaid application balance, or an x402 payment on the request itself. This is the rail a fleet of agents can use without a human in the settlement loop. At the protocol boundary those rails convert into HOUSE so that the memory network has one unit of account (Section 10).

Subscriptions as a user experience. A subscription is a UX wrapper on the same meter. It gives a workspace a predictable envelope for memory and persistence, resets each period, and reprices when backend costs move. It is not a separate product, it does not replace the meter, and it is not what HOUSE is for. HOUSE is not spent to unlock the subscription.

Commitments. Over an active plan Lighthouse provides a stated durability and availability service level, automatic renewal while the plan is active, a verifiable record of where memory lives and under what proofs,

and an export window before any lapse. Data is not retained after a plan lapses without renewal, and stored objects are not represented as irrevocable outside the contracted term.

10. HOUSE token

HOUSE is the protocol token of the memory network. It is the unit in which a write is paid, a recall is paid, a replica is claimed, and a namespace is attributed. It is not a license, a feature flag, or a discount coupon on hosted software.

Stablecoins remain available on invoices and on x402 so that an agent or a finance team can settle without holding HOUSE in the request path. Those payments convert into HOUSE at the protocol boundary. The network does not have a second unit of account for memory work.

10.1. Design principle

A token is intrinsic to a memory network when three statements are true.

1. No remembered batch is a protocol object until HOUSE is paid against its CID for the contracted term.
2. No recall answer is protocol-final until a bonded keeper has served it, and a keeper that serves an index the blobs do not support loses that bond.
3. Holding HOUSE does not unlock a product surface. Namespaces, models, adapters, and APIs are available on the meter and the invoice whether or not the caller holds the token.

The previous shape—stake to open stores, bid for fleet slots, pay HOUSE for a discount against stablecoins—treated the token as a SaaS gate. That shape is retired. Access is a product problem. HOUSE is a network problem.

10.2. Utility

Remember settlement. Every *remember* that flushes a batch pays $P_{\text{remember}} = \alpha \cdot C_{\text{store}}$ in HOUSE. C_{store} is the cost of placing and holding the blob on the chosen fabric for the term. That quantity is an expense, not a deposit. The fabric has to be paid. Locking $\alpha \cdot C_{\text{store}}$ as a bond would idle the same capital a second time on top of Filecoin or Walrus collateral that already slashes a provider for losing the bytes.

A payment settles work: the batch is accepted, a CID is issued, a deal or epoch is opened. On renewal the caller pays again for the next term. On irretrievability the network re-places from inherited backend collateral first, then from protocol reserves accumulated from remember payments. It does not slash a remember lock, because there is no remember lock.

Bonds belong where this network takes a performance risk the storage backend does not: a recall keeper that might serve an index the blobs do not support, and a replication claim that a second copy stays live. Those are collateral. Remember is settlement.

Recall work and keeper collateral. Recall is work: retrieve, score, egress. A query pays $P_{\text{recall}} = \beta \cdot C_{\text{egress}}$ in HOUSE. Index keepers—Lighthouse-operated today, permissionless as the adapter interface hardens—post HOUSE as an availability bond and earn the query fee. A keeper that returns vectors or records that do not hash-match the published batches is slashed. Throughput is not a stake-gated quota. It is a market: more bonded keepers, more scored queries.

Replication claims. A namespace that must survive a single-network failure posts a replication claim in HOUSE against the same CID on a second, uncorrelated backend. The claim pays placement and holds a smaller bond that funds a third copy if both primaries degrade. Dual-write is no longer only a product checkbox; it is a HOUSE-denominated posture.

Namespace attribution. A namespace is a protocol object. Opening, transferring, or composing one writes a HOUSE-backed record that binds identity to the CID log. The API key remains the convenient handle. The economic identity of a store—the thing a refurbished robot or a successor vendor can present—is that record. This replaces "stake HOUSE to unlock another namespace" with "a namespace is an attributed object on the network."

Forget as a published act. Forget is a tombstone batch, not a silent delete. Publishing the tombstone spends $\gamma \cdot C_{\text{released}}$ in HOUSE so that "this record is withdrawn" is an on-network event with a cost and a CID. Earlier blobs stay addressable; the spend is for the act, not for erasing history. That is the property auditors want and the reason encryption, not deletion, is the secrecy tool.

Governance. HOUSE locked in keeper collateral, replication claims, or a separate governance lock votes on coefficients α, β, γ, ρ; on which model adapters and storage backends are admitted; and on grant allocation for memory integrations. Voting weight may scale with lock duration. Governance does not vote a caller into a higher product tier.

Table 6: Memory-network work priced in HOUSE. Coefficients are protocol parameters and may be revised by governance. None of these lines is an access gate.

Act	HOUSE expression	What the token is doing
Remember	$\alpha \cdot C_{\text{store}}$ paid	Settles placement of the batch for the term.
Recall	$\beta \cdot C_{\text{egress}}$ paid to keepers	Wage for scored retrieval; keepers bonded.
Forget / tombstone	$\gamma \cdot C_{\text{released}}$ spent	Publishes withdrawal as a network event.
Replicate	$\rho \cdot C_{\text{replica}}$ locked	Claim that the same CID lives on a second fabric.
Namespace record	protocol fee, then dust lock	Attributes the store so recovery is not only an API key.
Other acts	to be defined	New memory-network functions as they ship.

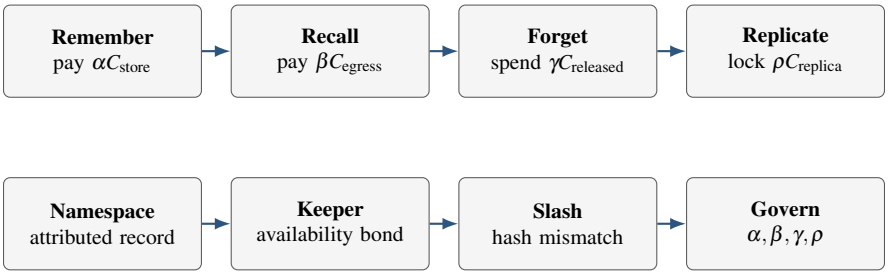

Figure 6: HOUSE on the memory network. The token pays a remember, wages a recall, publishes a forget, and slashes a keeper. Bonds are for keepers and replicas, not for the write itself.

A payment that arrives in stablecoin is converted at the boundary,

$$P_{\text{HOUSE}} = P_{\text{stable}} \cdot r_t,$$

where r_t is the prevailing HOUSE per stablecoin rate used for protocol accounting. There is no governance-set coupon x that makes paying in HOUSE cheaper than paying in stablecoins. Preferential rates were a SaaS lever; they are not a memory-network primitive.

10.3. What HOUSE is not

HOUSE does not unlock active stores, recall throughput, index retention, premium embedding models, or Kavach quota. Those are metered product resources. HOUSE does not auction fleet slots. Capacity on the hosted surface is provisioned like any other infrastructure. HOUSE does not confer ownership of Lighthouse or of customer data.

10.4. Allocation

Table 7: HOUSE allocation.

Bucket	Share	Role
Team	25%	Core contributors. Multi-year vest.
Automated Capital Formation (ACF)	25%	Launch and network-formation capital, including a limit-order program against FDV milestones.
Private investors and angels	14.673%	Early backers. Cliff and linear vest after TGE.
Liquidity pool	14.14%	Float at TGE.
Protocol safeguard max-lock	8.9%	Max-locked to protect early governance and liquidity.
Lighthouse Community + Ecosystem	5.957%	Grants, integrations, incentives, and native community.
Pre-TGE	3.33%	Community round before TGE. Cliff and linear vest.
Virtuals Ecosystem Airdrop	3%	Distribution to the Virtuals ecosystem at TGE.

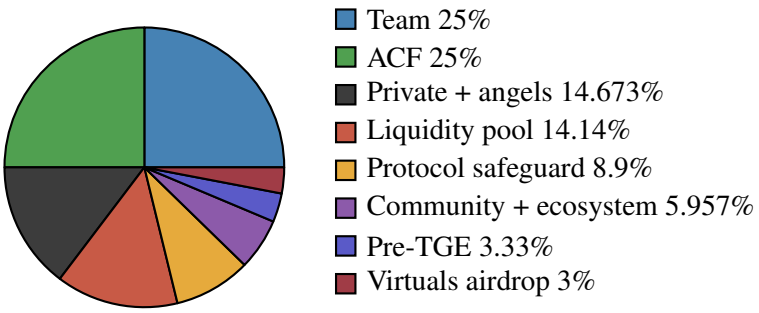

Figure 7: HOUSE allocation by bucket.

11. Risks and robustness

Table 8: Principal risks and the corresponding operational posture.

Risk	Why it matters	Mitigation
Backend cost inflation	Metered prices and subscription envelopes can lag if persistence or egress costs move quickly.	Coefficients α, β, γ, ρ can be revised; subscriptions reprice at period end.
Single-network failure	Filecoin or Walrus economics can move against a chosen rail.	Multi-backend routing, HOUSE-bonded cross-backend replication, CID-invariant migration, and bring-your-own storage as an exit.
Provider default	A storage provider declines renewal or faults.	Inherited network collateral; automated re-placement funded by remember-payment reserves; escalate to the hot IPFS layer; notify the customer.
Model lock-in	A store embedded with one geometry is mis-queried with another.	Model identifier stored with the index; the store can be re-embedded if the operator changes model.
Memory confidentiality	A leaked CID can expose plaintext memory until Threshold Encryption covers the memory rail.	Kavach already ships for files; memory encryption on the same 3-of-5 rail is in development; client-side encryption is available now.
Physical-AI connectivity	A robot cannot flush if it cannot reach a gateway.	Local pending buffer; opportunistic flush; bring-your-own on-prem adapter for air-gapped sites.
Bond mis-pricing	Keeper or replication bonds set too low fail to fund recovery; remember prices set too high tax honest writes.	Parameter bounds; periodic review; governance over α, β, γ, ρ and over which work is bonded versus paid.
Keeper equivocation	A recall keeper serves an index that does not match published blobs.	Availability bond; slash on hash mismatch; fall back to keyword scoring and origin-blob fetch.
Regulatory characterization	A token used to settle memory-network work can be misread as a software license or a pooled investment.	Metered product use remains available in stablecoin; HOUSE is work-and-bond, not access; no pooled-investment features; accurate durability language.

12. Roadmap

The items below are the current scope of work. They will move with product feedback. Some may ship as described, some may change, and some may be dropped.

Table 9: Current scope and planned work for the memory runtime.

Horizon	Work
Now	Memory SDK with remember / recall / forget; backend embeddings; MCP server; Filecoin and Walrus adapters; incremental batch blobs; Files app; L3; Kavach (Threshold Encryption) for files; x402.
Near term	Stable composable adapter interface. Open and closed model packs as configuration. Kavach-backed encrypted memory. HOUSE remember settlement, recall-keeper collateral, replication claims, and namespace records. Namespaced sharing and read-only store mounts. Physical-AI reference integration with an on-device buffer, opportunistic flush, and site-versus-unit namespaces.
Later	Bring-your-own storage adapters. Cross-backend memory replication as a bonded protocol act. Agent-facing x402 for recall, converting to HOUSE at the boundary. Fleet-level namespaces with per-unit episodic stores. Permissionless index keepers. Formal proof receipts as a signed, queryable log.

13. Conclusion

An agent that cannot remember is a demonstration. An agent that remembers only inside one vendor is a tenant. An agent that remembers on a network it can leave, under a hash anyone can check, with a model its operator chose, is closer to a mind. A machine that can reconstruct that mind after a firmware flash, prove what it knew after an incident, and share a site map without donating its episodic trace is the physical case of the same claim.

Lighthouse is building that capability on infrastructure it already operates. Storage remains a product surface. HOUSE is the settlement and the wage of the memory network, not a gate in front of it. The protocol, the token, and the next phase of engineering are pointed at the layer above storage: a brain for agentic and physical AI that is composable, provable, and not captive to a single cloud.

Nothing in this document is an offer of securities or an invitation to invest. Holding HOUSE tokens confers no ownership of Lighthouse or of customer data. Durability claims in this paper are service-level commitments over a term. Product surfaces described here will move with use. Live network metrics are published at `https://explorer.lighthouse.storage`.

References

[1] J. Benet. IPFS – content addressed, versioned, P2P file system. arXiv:1407.3561, 2014.

[2] Protocol Labs. Filecoin: A decentralized storage network. Technical report, 2017. `https://filecoin.io/filecoin.pdf`.

[3] Mysten Labs. Walrus: A decentralized blob store. Technical report, 2024. `https://www.walrus.xyz/`.

[4] N. Reimers and I. Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of EMNLP-IJCNLP*, 2019.

[5] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, 2020.

[6] Anthropic. Model Context Protocol specification. 2024. `https://modelcontextprotocol.io/`.

[7] Lighthouse. Network explorer. 2026. `https://explorer.lighthouse.storage/`.

[8] Lighthouse. Memory SDK documentation and package. 2026. `https://docs.lighthouse.storage/memory/intro`; `https://www.npmjs.com/package/@lighthouse-web3/memory`.

[9] Lighthouse. Kavach encryption SDK. 2023. https://github.com/lighthouse-web3/encryption-sdk; https://docs.lighthouse.storage/how-to/upload-encrypted-data/.