MemSwarm: A Distributed Memory Architecture for Persistent Context Sharing Across Multi-Agent Systems

Anonymous authors
Paper under double-blind review

Abstract

Agent memory systems are built for one agent's stream. When a team of large language model (LLM) agents shares memory across hosts, nothing specifies what a reader on another host sees, and a deployed agent store used centrally silently loses one side of every concurrent contradictory write. We treat shared agent memory as a replication problem and adopt the classical contract of causal consistency plus session guarantees (read-your-writes, monotonic reads, writes-follow-reads), under which a handoff is sound once the delegation carries the delegator's causal frontier and the delegate waits for it. MemSwarm implements the contract with no coordination on writes: each agent host keeps a local replica of an event-sourced log, hybrid logical clocks version the records, conflict-free replicated data types merge them, and a concurrent contradiction survives as an explicit multi-value for upstream policy to resolve. Multi-process experiments on one machine over loopback show zero lost writes in 23,200 visibility samples, conflicts surfaced on every replica where last-write-wins destroys them without a signal, and monotonic-read violations in 26% of degraded roaming-reader trials that frontier-enforced reads remove by blocking 0.4% of reads. On a controlled handoff task, LLM readers shown both values and told to report conflicts flag them in 91% of trials; without that instruction haiku picked one side in 20/20 trials, re-implementing last-write-wins in the prompt. A central node with the same records loses the conflict as last-write-wins does, since detection uses what the writer's store had applied, which only a per-host replica tracks for its agent. Under a single failure, replicas serve a cut-off agent 600/600 operations where a central store serves 0/600, at the price of staleness until heal. We release a Model Context Protocol server, the benchmark suite, and raw logs.

1 Introduction

LLM agents ship as teams: planner-executor pipelines, researcher-implementer handoffs, fleets of specialized workers spanning processes, machines, and frameworks (Guo et al., 2024; Xu et al., 2024; Zhu et al., 2025); their memory systems are built for individuals. The current generation persists one agent's view of one interaction stream: paged virtual context (Packer et al., 2023), retrieval-scored streams (Park et al., 2023), decay (Zhong et al., 2024), pipelines that extract and consolidate what the agent said (Chhikara et al., 2025), and graph-shaped memory, bi-temporal in one system and self-evolving in another (Rasmussen et al., 2025; Xu et al., 2025). A recent survey names multi-agent memory among the field's open frontiers (Hu et al., 2025); the most-cited multi-agent survey scarcely treats shared memory at all (Guo et al., 2024).

What breaks when a fleet naively shares a store is now documented. From a production memory service, Margalit et al. (2026) identify four fleet-memory failure modes: unauthorized leakage, stale propagation, contradictions that persist once written, and provenance collapse. Li (2025) report that relevance-based shared retrieval leaks 53% of private memory items in diagnostic stress tests of cooperative agents. These are the classical pathologies

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

of replicated mutable state, which four decades of distributed-systems theory addresses. This paper takes on stale propagation and contradiction persistence. Leakage needs the access-control layer we do not build (Section 6).

Vendors treat distribution as valuable and sell it in closed form. Letta distributes MemGPT-descended memory only via Letta Cloud (Packer et al., 2023), and Zep's scale tier is the proprietary Context Graph Engine (Rasmussen et al., 2025). Mem0's access control and headline benchmark numbers are platform-only (Chhikara et al., 2025), and Redis now favors a managed tier over its open memory server.[1] Protocols have not filled the gap either; MCP standardizes agent access to tools and context and A2A v1.0 standardizes task exchange, but no protocol defines shared memory between agents.[2] Of the fourteen systems we survey (Appendix A), the open ones stop at one store or server and the distributed ones sit behind a managed service, consistency left to the database underneath.

We treat fleet memory as what it is, a replicated-data problem, and build MemSwarm. Memory operations become an event-sourced log, versioned by hybrid logical clocks (Kulkarni et al., 2014) and merged as CRDTs (Shapiro et al., 2011) across a local-first replica per agent host (Kleppmann et al., 2019), under a consistency contract of causal consistency plus session guarantees (Terry et al., 1994). The contract has observable consequences, which our partition experiments make concrete. Last-write-wins resolution left every replica in agreement, discarding without trace the losing side of 100/100 concurrent contradictions. MemSwarm's multi-value merge instead surfaced the conflict on every replica in 100/100 trials; both outcomes follow from the merge rules and validate the implementation. The same task run against LangGraph's InMemoryStore as a shared central store left the losing value unrecoverable with zero conflict signal in 200/200 trials, the survivor decided by write interleaving (Section 5.4). Such races need no partition: three uncoordinated agents produced truly concurrent contradictory supersessions at 0.21 to 0.97 per 100 operations depending on contention (Section 5.4), and convergence alone certifies agreement, not truth-preservation.

The contract itself is classical. Proposition 1 is the delegation instance of the writes-follow-reads and causal-visibility guarantees of Terry et al. (1994) and Lloyd et al. (2011), and Appendix C proves it for our shipping protocol. What is new is the agent-facing side: a mapping of agent-team failure modes onto those guarantees (Section 3); an open, framework-agnostic substrate that surfaces concurrent contradictions as multi-values with provenance and leaves the choice among them to upstream policy, since a store that chose would be destroying evidence again; a benchmark suite for the consistency behavior of shared agent memory (Appendix F); and evidence from a controlled handoff task that an LLM consumer keeps a surfaced conflict only when its answer contract requires it (Section 5.4).

2 Related Work

Single-agent memory. MemGPT frames memory as virtual-context paging under LLM control (Packer et al., 2023); Generative Agents keep a retrieval-scored memory stream with reflection (Park et al., 2023), and most successors vary it: forgetting curves (Zhong et al., 2024), associative graphs and evolving note networks (Jiménez Gutiérrez et al., 2024; Xu et al., 2025), and the experiential traces of Reflexion and Voyager (Shinn et al., 2023; Wang et al., 2024). Graphiti/Zep tracks fact validity bi-temporally (Rasmussen et al., 2025), Mem0 names an ADD/UPDATE/DELETE/NOOP vocabulary over one store (Chhikara et al., 2025), and MIRIX routes one user's memory across six internal stores (Wang & Chen, 2025). All assume one writer on one stream, as do the benchmarks (Maharana et al., 2024; Wu et al., 2025; Tan et al., 2025); A-MEM's memory evolution, benign for one agent, becomes a consistency hazard once two agents write concurrently.

[1] Letta Cloud, https://www.letta.com, Zep, https://www.getzep.com, Mem0 platform, https://mem0.ai, Redis Agent Memory Server, https://github.com/redis/agent-memory-server. Accessed September 2026.

[2] https://modelcontextprotocol.io, https://www.linuxfoundation.org/press/a2a-protocol-surpasses-150-organizations-lands-in-major-cloud-platforms-and-sees-enterprise-production-use-in-first-year. Accessed September 2026.

Multi-agent memory. G-Memory builds a hierarchical graph over team collaboration histories, in a single co-located store (Zhang et al., 2025). Collaborative Memory contributes private/shared tiers, bipartite permission graphs, and immutable provenance, in a simulated single-store setting (Rezazadeh et al., 2025). MaaS mediates every cross-agent access by ownership and purpose (Li, 2025). Governed Shared Memory names the fleet-memory failure modes and proposes governance primitives (scoped retrieval, supersession, provenance tracking) but no consistency theory (Margalit et al., 2026). LangGraph's BaseStore, the only widely deployed cross-agent store, delegates all consistency to the backing database.[3] None defines what concurrent writers may observe: no read-your-writes, no causal order, no merge semantics. Contradictions, where handled at all, are resolved by LLM judgment or temporal supersession heuristics. Table 1 positions MemSwarm against the closest systems; Appendix A extends the analysis to 14 systems.

Distributed-systems lineage. Shared agent memory predates LLMs. Blackboard architectures made a common workspace the coordination medium (Erman et al., 1980; Hayes-Roth, 1985; Nii, 1986), and Linda's tuple spaces decoupled communicating processes through associative shared memory (Gelernter, 1985). Stigmergy names the reason both worked, namely that environment-mediated coordination scales without direct addressing (Heylighen, 2016). None of that era replicated the workspace. Bayou did, shipping per-writer logs between weakly connected replicas on an anti-entropy timer and handing conflict detection back to the application (Terry et al., 1995); our replication layer is that loop, and our contract is the session guarantees the same work states (Terry et al., 1994). From Dynamo we take the read reply: keep the concurrent siblings and return them all, since the store does not know which version should win (DeCandia et al., 2007). COPS makes the coordination-free part proposable, having shown causal consistency scale with nothing to wait for on the write path (Lloyd et al., 2011). The OR-Set and multi-value register we build on arrive with their convergence proofs (Shapiro et al., 2011; Preguiça et al., 2018). Timestamps are logical and hybrid logical clocks taken as given (Lamport, 1978; Kulkarni et al., 2014), the availability-consistency space we work inside was mapped before us (Bailis et al., 2013; 2012), and a local replica per host over a log-structured store is inherited (Kleppmann et al., 2019; Kreps et al., 2011; Eugster et al., 2003). MemSwarm applies this Bayou/Dynamo lineage, anti-entropy over per-writer logs and sibling surfacing, to agent memory. The replication mechanics and the contract are established; what is new is their agent-facing framing, substrate, and benchmark.

Concurrent work. Independently, StateFuse builds a conflict-preserving CRDT memory layer (Volkov et al., 2026), TOKI formalizes contradiction resolution as bitemporal operators (Wang, 2026), and Khan (2026b;a) bring concurrency control and anomaly detection to multi-agent LLM state. None defines a contract for readers on other hosts.

Table 1: Fleet-memory capabilities of the closest systems (✓ yes, ○ partial, × no). Consist.: a consistency contract across concurrent agents. Conflict: contradictory concurrent writes surface to readers. Access: memory-level access control (MemSwarm's prototype deliberately has none, Section 6). Letta's consistency ○ is single-server serialization. Full 14-system analysis in Appendix A.

System	Persist	Share	Distrib.	Consist.	Conflict	Access
Mem0 (Chhikara et al., 2025)	✓	○	○	×	×	×
Letta (Packer et al., 2023)	✓	○	×	○	×	○
Graphiti/Zep (Rasmussen et al., 2025)	✓	○	○	○	○	×
LangGraph Store	✓	✓	○	○	×	×
Collaborative Memory (Rezazadeh et al., 2025)	✓	✓	×	×	×	✓
G-Memory (Zhang et al., 2025)	✓	✓	×	×	×	×
MemSwarm (ours)	✓	✓	✓	✓	✓	×

[3] https://github.com/langchain-ai/langgraph. Accessed September 2026.

3 Requirements: What Agent Teams Need From Memory

We model an agent team as n agents, each attached to a replica of a shared memory store, with replicas exchanging updates asynchronously. An agent's execution of a task is a session, an interleaved sequence of memory reads and writes that may end in a delegation, a message handing the task to another agent, generally on another replica. What must the store guarantee for the team to behave coherently? The session guarantees of Terry et al. (1994), formulated for weakly consistent replicated databases, map onto agent-team failure modes one-to-one.

Read-your-writes: an agent must see what it just learned. Without RYW an agent's own knowledge is unstable inside its session, and the damage is a contradiction the store keeps. A support agent extracts "customer prefers the 14:00 slot" and writes it to memory; several tool calls later it queries memory to compose the booking, and that read may be served by state the write has not reached. It re-asks the user and loses a turn, or books a different slot and records that against its own earlier write.

Monotonic reads: memory must not regress mid-task. A monitoring agent reads "deployment 412 failed", starts planning a rollback, and then draws a subsequent read from a lagging replica that no longer contains the failure record. The plan now rests on evidence the store denies having. What MR requires is that each read in a session reflect a state at least as advanced as the previous one. Monotonic writes come for free from the per-writer order of an event log.

Writes-follow-reads: handoff must be sound. The defining multi-agent operation is delegation: researcher A reads the task state, writes findings $m_1, \ldots, m_k$, and delegates to implementer B on another host. B behaves correctly only if it observes everything A knew when it delegated. Otherwise B re-derives A's work or, worse, acts on the pre-handoff state. That is stale propagation, the failure mode Margalit et al. (2026) report from production. WFR orders an agent's writes after the reads that informed them. Combined with causal delivery and a delegation message carrying the delegator's causal frontier (its per-writer applied-event vector), it yields the delegation instance of the causal visibility COPS enforces (Lloyd et al., 2011):

Proposition 1 (Handoff soundness, informal) In a causally consistent store enforcing RYW and WFR, if a delegation message carries the delegator's causal frontier and the delegate's replica delays the delegate's first read until its applied event set dominates that frontier, then every memory write causally prior to the delegation (including every write the delegator made or read before delegating) is visible to every read in the delegate's session. Causality here is happened-before over memory operations, so a delegation sent outside the store must carry the frontier explicitly. Formal statement and proof sketch in Appendix C.

Linearizability is unnecessary and unaffordable. Agents need their state causally explained, not a real-time total order, and linearizable reads and writes cannot remain available under partition (Bailis et al., 2013), while agent hosts are exactly the intermittently connected, local-first devices that motivate weaker models (Kleppmann et al., 2019). Our coordination-free prototype, syncing by anti-entropy on a 200 ms period, already delivers $\sim$100 ms median cross-replica visibility with zero data loss under injected delay and message drop (Section 5).

Pure eventual consistency is unsafe. Eventual consistency promises only convergence, the wrong success criterion for memory; it admits two behaviors, both of which we measure. First, without WFR a delegate may read pre-delegation state, breaking Proposition 1. Second, convergence can mask destruction. Last-write-wins resolution settles every replica on one value and raises no conflict signal, the losing contradiction dropped from the state they agree on, and under same-millisecond timestamps the survivor is picked by a lexical node-id tiebreak. Such races also arise under healthy replication, and Section 5 quantifies both. The contract that follows is causal consistency, session guarantees, and explicit surfacing of concurrent conflicts for policy resolution. It asks less of the fleet than linearizability and promises more than pure eventual consistency, and it stays available under partition, which linearizability cannot (Bailis et al., 2013).

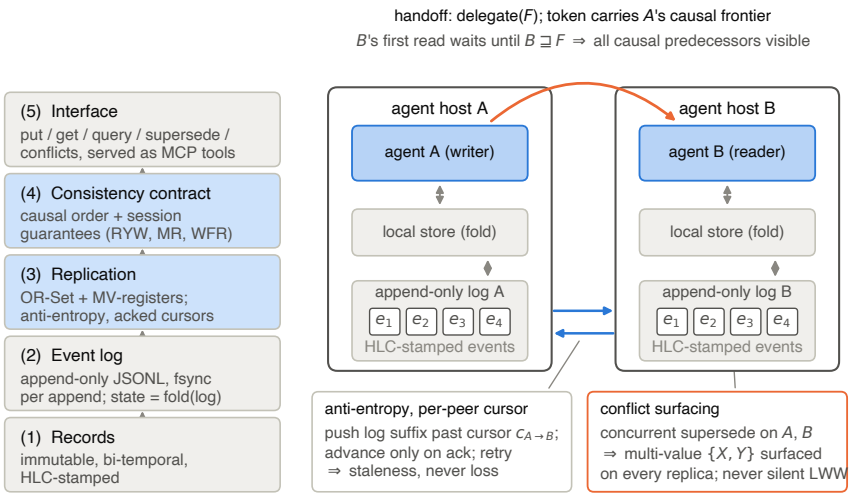


Figure 1: Left: the five MemSwarm layers, bottom-up: bi-temporal HLC-versioned records; per-node event-sourced log; CRDT replication by anti-entropy; the causal-plus-session consistency contract; the agent-facing interface. Right: a two-replica handoff carrying the delegator's causal frontier F, with a concurrent Supersede surfacing as a two-valued conflict (Section 4 opening).

When does replication earn its cost? A single centralized store provides all three session guarantees trivially. For a fleet colocated in one datacenter with a reliable database it is the simpler, better design, and we concede the case. Replication earns its cost where one store is unavailable or unacceptable: agents on developer laptops and edge devices working offline, cross-organization fleets that cannot share a database, and teams that must stay available when the coordinating store fails. The scope claim is bounded accordingly: wherever memory is replicated the contract matters (Kleppmann et al., 2019). Section 5.4 measures both effects.

4 The MemSwarm Architecture

MemSwarm has five layers, and each is there because the layer beneath cannot supply what the contract of Section 3 asks. Correction must be expressible without destroying what it corrects, so the unit of memory is an immutable bi-temporal record, ordered per writer by the per-node log. Replication re-ships each node's log to every peer on a timer, a per-peer cursor advancing only once the peer has applied what it was sent, in bounded chunks. The layers above are thin: a session contract over the log, and an interface exposing it to agents. Figure 1's right panel traces the defining operation: writer A delegates carrying its causal frontier F, delegate B's first read waits until B's replica dominates F (Proposition 1), and a concurrent Supersede on both replicas surfaces everywhere as a two-valued conflict every reader sees. Record schema, log format, and merge rules are in Appendix B.

4.1 Records, Clocks, and the Event Log

Immutable bi-temporal records. The unit of memory is an immutable record: content, type (episodic/semantic/procedural), scope, provenance (writer agent, source, session), confidence, and two time axes. An HLC transaction timestamp records when the system learned the fact, and a [valid_from, valid_until) interval bounds when the fact holds in the world, after Zep's bi-temporal model (Rasmussen et al., 2025). Correction happens by supersession, where Supersede closes the old record's validity and admits a replacement while the old record stays readable forever, so "what did agent X know at time T" stays answerable.

Hybrid logical clocks. Every record and event is stamped by a per-node HLC (Kulkarni et al., 2014), a totally ordered (physical ms, logical, node id) triple that tracks wall time,

moves only forward, and respects happened-before across messages; one timestamp therefore serves temporal reasoning and replication alike. It is property-tested under skewed, frozen, and backwards-jumping clocks (Appendix G).

Event-sourced log. Every operation (Observe, Supersede, Forget) is an immutable event carrying actor, HLC, and vector clock. Agent-visible state is a materialized fold of the log, recomputed from events, which buys provenance, audit, and replay for late-joining replicas (Kreps et al., 2011). Persistence is one append-only JSONL file per node, fsynced per append; a crash can tear only the final line, recovery truncates the torn tail, and corruption before the tail raises an error. The fold tolerates concurrent arrival orders, so a Supersede applied before the Observe it targets still materializes correctly.

4.2 Replication, CRDT Merge, and Conflict Policy

Anti-entropy with acked cursors. Reads and writes are local and coordination-free, each agent host running a full local replica (Kleppmann et al., 2019). Periodically each node pushes to each peer the contiguous suffix of its log past a per-peer cursor, which advances only on acknowledged application, and a dropped or unacknowledged batch is simply re-pushed next tick, so degradation only adds delay, with no write missing in any of the 23,200 evaluation samples (Section 5). Application is idempotent (keyed by event id) and order-tolerant, and applied remote events are appended to the receiver's log, so updates re-ship transitively through any connected topology.

CRDT merge. The record collection is an OR-Set (Shapiro et al., 2011) whose add-tags derive deterministically from event ids, which makes replicas applying the same events converge exactly. Unlike the textbook add-wins OR-Set, ours is remove-wins: Forget tombstones the record id and admission checks it, which retraction needs and never-reused ids make free. Each superseded record id owns a multi-value register keyed by vector clocks: dominated replacements leave the register, concurrent ones coexist, and conflicts() lists every record whose register holds several. Plain queries return exactly the register's live values, so a reader sees both sides of a concurrent conflict and only the later of two ordered corrections. Resolution is an explicit upstream policy over the provenance, recency, and authority fields every record carries, quantified against silent LWW in Section 5.

4.3 Interface: Memory over the Model Context Protocol

The interface layer exposes the memory verbs as newline-delimited JSON over TCP: put, get, query, supersede, conflicts/conflict_values, plus pause/resume of per-link sync for fault injection. Every experiment in Section 5 drives real node processes through it. The causal machinery of Section 3 is part of this surface. frontier returns the replica's per-writer applied-event vector, and wait_frontier blocks a read until the local state dominates a supplied frontier. A thin client session layer carries the last write and read frontiers, so RYW, MR, and WFR hold for any client routing reads through it. The store serves unconstrained reads to clients that bypass the layer; store-side enforcement is future work. The delegation token of Proposition 1 is exactly the delegator's frontier. Anti-entropy ships bounded chunks of at most 250 events per push, so backlogs drain incrementally. Each verb maps one-to-one onto an MCP tool, and the released artifact includes a dependency-free MCP stdio server exposing all of them, get_frontier/wait_frontier included, tested end-to-end against a live node. The LLM arm of Section 5 consumes retrieved memory through this stack. The semantic-retrieval layer sits above this interface and is not evaluated.

5 Evaluation

We ask three questions of the running system: (i) how stale is coordination-free replication, and how does staleness tune and scale? (ii) does conflict surfacing preserve what last-write-wins destroys? (iii) do the session guarantees of Section 3 hold, and which require enforcement?

Table 2: Concurrent contradictory supersessions under partition: 100 trials, seed 42, 3 nodes, all nodes inspected per trial. The LWW column resolves the same replicated state by max-HLC at read time.

Outcome	MV (MemSwarm)	LWW ablation
Conflict surfaced on every node	100/100	0/100
Both values survive (exactly $\{X, Y\}$)	100/100	0/100
Losing value silently destroyed	0/100	100/100
Cross-node agreement after heal	100/100	100/100
False conflicts before the concurrent writes	0	0

Setup. All experiments run the store in real OS processes, one per node, newline-delimited JSON over loopback TCP. A run uses 3 nodes, 200 ms anti-entropy, seed 42 unless stated. Degradation is injected at the sync layer as per-batch one-way delay and seeded drop. Writes are jittered over the sync interval to break phase lock (Appendix D), and every sample is raw JSONL with its configuration embedded. All nodes share one machine, so the results bound protocol behavior under controlled degradation and say nothing about WAN conditions. An early campaign shared the machine with a concurrent CPU load, flagged co_load per file; those campaigns were re-run quiet with two added seeds. Logical outcomes match exactly, timing percentiles shift by under one sync tick (Appendix D), and quiet numbers are quoted for those campaigns; two co-loaded staleness cells remain in the sweep, within one tick of their quiet re-runs. Of the two tiers, the substrate tier establishes that the store keeps its promises, while a controlled task tier (Section 5.4) measures whether those promises change what a consuming agent, scripted or LLM, gets right.

5.1 Staleness and Tunability Under Degradation

We measure PBS-style t-visibility (Bailis et al., 2012): write at node A, poll every other node at 2 ms until visible, over 3 seeds and a delay $\{0, 50, 200\}$ ms by drop $\{0, 0.1, 0.3\}$ grid, 600 samples per seed per cell (Appendix D, Table 6). Visibility follows the design constants: p50 sits near delay + interval/2 (101.6 ms at baseline) and p99 near delay + interval, and drops fatten mainly the tail (delay 200 ms, drop 0.3: p50 367.0, p99 969.3 ms). Sweeping the sync interval over $\{50, 100, 200, 500\}$ ms moves both percentiles with it (Figure 2), and scaling from 2 to 8 nodes leaves p99 at 203.2 to 205.3 ms. No write was ever lost: 0 never-visible outcomes in all 23,200 campaign samples.

5.2 Conflict Surfacing versus Last-Write-Wins

Table 2 is implementation validation, since a multi-value register must surface both sides of a confirmed-concurrent pair and max-HLC resolution must destroy one, so the headline rows hold by construction. Each trial starts from a replicated base fact. A lossless partition separates A from $\{B, C\}$, then A supersedes with X while B concurrently supersedes with contradictory Y, vector clocks confirming concurrency. The partition heals, and every node is inspected (Appendix D). The LWW ablation reads the identical replicated state through max-HLC resolution. It converges to cross-node agreement in every trial while silently destroying the losing value with zero signal, whereas MemSwarm surfaces exactly $\{X, Y\}$ everywhere, visible to a plain query() reader. Across seeds 7/42/1337 on a quiet machine, surfacing is 100/100 at every seed and convergence after heal takes p50 185-187 ms.

Conflicts arise under healthy replication too. With no partition and healthy 200 ms replication, supersessions of the same record from two nodes within seeded windows of 10/50/200 ms were truly concurrent by vector clock in 100%/100%/96% of 100 trials each; that fraction is set by the window length against the 200 ms sync period, so the run validates the vector clocks, and the incidence campaign (Section 5.4) measures how often races arise. All 296 concurrent cases surfaced as multi-values, and the 4 causally ordered cases resolved to the later write. A later rerun at the same seed gave 100/100/95, so the 200 ms window is timing-sensitive (Appendix D).

Table 3: Session guarantees, quiet machine: 200 trials per cell per seed, seeds {42,7,1337} pooled; degraded = 50 ms delay + 30% drop. The one violated cell is the roaming reader under degradation (per-seed range 23.5-30%).

Property	Clean	Degraded
RYW: read where you wrote	600/600 found	n/a
contrast: immediate cross-node read	600/600 miss, 0 lost	n/a
MR: reader bouncing $B \leftrightarrow C$	0/600 violated	156/600 violated (26.0%)
MR: reader pinned to one replica	0/600	0/600
WFR: single-writer K=10 chain	0/600, p50 102-115 ms	0/600, p50 184-214 ms
WFR: cross-writer, frontier wait	0/600, wait p50 203 ms	0/600, wait p99 513 ms

5.3 Session Guarantees, Empirically

Table 3 verifies Section 3's claims on the live cluster. Read-your-writes holds at the writer's replica, where the apply precedes the ack, while an immediate cross-node read misses in 600/600 trials with zero losses, which is the empirical case for session stickiness. Monotonic reads is the guarantee that needs a session layer. A roaming reader under degradation regressed in 26.0% of trials (Wilson 95% 22.7-29.7%). Pinning eliminates every violation, since a replica's applied state only grows. The single-writer chain row exercises per-writer prefix delivery, or monotonic writes, which held in all 600 handoffs of a K=10 chain per condition, clean or degraded, as the shipping design predicts (Appendix C). Reruns on fresh clusters reproduce every pass/fail pattern exactly (Appendix D). Frontier-enforced reads show 0 violations across 30,000 degraded bouncing reads (seeds 42/7/1337) at small cost: 0.43-0.50% of reads block, and a read costs 0.34-0.36 ms against 0.14-0.16 pinned, noise beside an LLM inference step.

Cross-writer handoff through the frontier mechanism. This experiment builds a delegation whose causal history spans two writers. Writer W_1 publishes five records at A; delegator D at B reads them, writes five records referencing them, and hands off its frontier as the token. The delegate at C calls wait_frontier(token), then reads all ten. In 1,200 handoffs (200 trials, seeds 42/7/1337, clean and degraded): zero violations, zero wait timeouts. Skipping the wait misses causal predecessors in 600/600 immediate reads per condition. The mechanism's price is the blocking time: p50 203 ms clean, one sync tick, p99 513 ms degraded.

5.4 From Guarantees to Task Outcomes: a Controlled Handoff Task

The contract matters only if its violations reach task outcomes, so we measure exactly that. A researcher at node A writes eight operational facts, then corrects three by supersession. During a partition a second writer at B supersedes one of those facts with a conflicting value. After healing, an implementer at C answers eight questions from retrieved memory alone. Ground truth is the corrected values, and for the conflicted fact the correct behavior is flagging the conflict, not silently picking a side. The conditions differ only in the implementer's read path: mv+wait takes the frontier-wait handoff and consults conflicts, lww reads the same state by max-HLC, no-wait skips the wait. Two implementer arms run it, the first of them a deterministic scripted reader at 200 trials per condition per seed over seeds 42/43/44, whose outcomes are identical across seeds, so interval notation is omitted as decorative. The LLM arm is claude haiku via CLI, its prompt limited to the retrieved records and questions, its cells pooling all runs across seeds 42/43/45 (per-run provenance in Appendix D). We also run the identical task against LangGraph's InMemoryStore as a shared central store, a deployed baseline with its own data model that we did not build.

The scripted values 1.000, 0.875, and 0.625 are 8/8, 7/8, and 5/8, identical in all 600 trials; the quantity that varies is the conflicted-fact outcome. An adjudication variant makes it answerable by every arm, ground truth becoming the higher-authority writer's value. LWW is then right exactly when jitter makes that writer the HLC winner, 301/600 trials (50.2%, per-seed 47.5-53.5%), while the multi-value reader applies the authority rule to the surviving provenance and scores 1.000 (haiku given the rule: 20/20 against 11/20 under LWW reads): what the contract preserves is the evidence the policy needs. The LangGraph row shows

Table 4: Handoff task by memory condition. Accuracy is $(7 + c)/8$ for conflicted-fact outcome $c \in \{0, 1\}$, less 2/8 for no-wait, which also misses two other corrections, so the scripted columns are fixed by the read path and demonstrate the mechanism; they are not an accuracy comparison. Scripted: 600 trials/condition over 3 seeds, deterministic; 95% Wilson intervals). LLM cells pool all haiku runs (seeds 42/43/45; mv+wait n=139, lww n=140, no-wait n=120; timeout exclusions in Appendix D). LangGraph row: the released InMemoryStore as a shared central store. Central+MV row: one MemSwarm node shared by all roles, seed 42.

Condition	Acc. (scripted)	Flagged (scripted)	Acc. (LLM)	Flagged (LLM)
mv+wait	1.000	600/600 (99.4-100%)	0.989	127/139 (91.4%)
lww	0.875	0/600 (0-0.6%)	0.875	0/140
no-wait	0.625	0/600	0.625	0/120
LangGraph (central)	0.875	0/200		
Central node, MV	0.875	0/200		

a deployed store losing the conflicted value with zero signal in 200/200 trials, the survivor decided by interleaving at 113/87, because its default data model replaces on put. The LLM arm measures what a consuming model does with a surfaced conflict. The standard prompt instructs it to answer CONFLICT when a key has more than one current record (prompts verbatim in Appendix D). Under that instruction haiku flags 91.4% [85.5, 95.0] (127/139), opus 19/20, and sonnet 4/16 (4 timeouts excluded; 4/20 to 8/20 under imputation), so compliance varies widely by model. Records are listed in HLC order and the field agent's record came last in 133/140 logged trials, so recency and list position are confounded. Under a neutral prompt that drops the instruction haiku picked one side in 20/20 trials, and shown a silent winner no model flags (0/299 pooled). A surfaced conflict thus reaches the answer only when the consumer's answer contract demands it; otherwise the agent re-implements LWW in the prompt. In the replicated arm conflicts() reported the conflicted key in all 800 mv+wait trials. One central node with the same multi-value records flags none (0/200) and scores 0.875 like LWW, the survivor split 113/87 by interleaving: at one node the node's clock orders the two blind corrections, so the second dominates the first. Detection tracks what the writer's store had applied, which a per-host replica approximates for its agent; a central store would need each supersede to carry the writer's read context, as in Dynamo (DeCandia et al., 2007).

Availability under a single failure. Three agents issue one operation each per 100 ms through a 3 s outage, 20 trials per cell (Appendix E, Table 7). A central store's crash fails every agent's operations (0/600 each); a replica's crash leaves the other two agents at 600/600. A cut-off implementer gets 0/600 from the central store and 600/600 from its replica, each read missing 16.7 peer writes on average until heal. No acknowledged write was lost, and a handoff to the cut-off agent completes only after heal in both designs (95 ms after it for the frontier wait).

Incidence under natural timing. With only timing left natural, three uncoordinated agents sharing a runbook produced truly concurrent same-record supersessions at 0.97 [0.53, 1.63] per 100 operations under hot-key contention (14 events) and 0.21 [0.04, 0.61] with organic references (3 events); the rate follows the workload's contention (Appendix D.7).

6 Limitations and Conclusion

Limitations. (1) Evaluation is single-machine, so the WAN is untested and failures are injected, not observed. (2) The task tier and incidence campaign are small and synthetic. (3) The semantic-retrieval layer is unevaluated, and ranking could reintroduce single-valued reads. (4) A shared store amplifies poisoning (Lin et al., 2026), and no guarantee extends to adversarial writers. (5) The prototype never compacts; causal-stability compaction (Baquero et al., 2017) is the remedy. Conclusion. Multi-values preserve contradictions, per-host replicas give detection each writer's context, and replicas keep agents working through a single failure, for about one sync tick of waiting on under 1% of reads.

AI Use Statement

A large language model assistant was used to aid and polish the writing of this manuscript. The authors take full responsibility for all content.

Reproducibility Statement

Every number in this paper traces to a raw JSONL run file in which each line embeds the full experiment configuration (seeds, node count, sync interval, delay, drop rate, store implementation); summary tables are regenerated from the raw files by the released analysis scripts. All randomness is seeded: conflict and session-guarantee outcomes were verified trial-by-trial identical across independent reruns on fresh clusters, and drop-pattern seeding is validated draw-by-draw against per-decision drop logs (Appendix D scopes one pre-fix sweep whose distributional claims hold but whose per-run drop sequences predate the seeding fix). Each experiment has a one-command reproduction entry point in the harness documentation. The anonymized implementation, test suite (deterministic under a documented seed variable), experiment harness, raw result files, and analysis scripts are provided as supplementary material; hardware and software configuration appear in Appendix D.

Ethics Statement

This work evaluates a memory substrate on synthetic workloads only; no human subjects, personal data, or deployed agents are involved. The architecture carries three risks into any deployment that runs it on real data. A persistent, replicated, shared memory means anything one agent records propagates to every replica within a sync interval and outlives the session that produced it; cross-agent leakage from naively shared stores is already measured at high rates (Li, 2025), and the evaluated prototype enforces no access control. Bi-temporal supersession deliberately preserves superseded content for audit, which is in direct tension with deletion obligations: Forget tombstones a record from queries, but the event log retains history, so compliant deployments need log compaction or encryption-based erasure. The third risk is poisoning, which shared persistent memory amplifies: a single adversarial write persists and propagates fleet-wide (Lin et al., 2026). We surface provenance and conflicts precisely so that governance layers can act on them, and we state plainly that the reference implementation is not yet such a layer.

References

Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Probabilistically bounded staleness for practical partial quorums. Proceedings of the VLDB Endowment, 5(8):776–787, 2012.

Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Highly available transactions: Virtues and limitations. Proceedings of the VLDB Endowment, 7(3):181–192, 2013.

Carlos Baquero, Paulo Sérgio Almeida, and Ali Shoker. Pure operation-based replicated data types. arXiv preprint arXiv:1710.04469, 2017.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. In Proceedings of the 28th European Conference on Artificial Intelligence (ECAI), volume 413 of Frontiers in Artificial Intelligence and Applications, pp. 2993–3000. IOS Press, 2025. doi: 10.3233/FAIA251160.

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP), pp. 205–220, 2007.

Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. ACM Computing Surveys, 12(2):213–253, 1980.

Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. ACM Computing Surveys, 35(2):114–131, 2003.

F. Garwood. Fiducial limits for the Poisson distribution. Biometrika, 28(3/4):437–442, 1936.

David Gelernter. Generative communication in Linda. ACM Transactions on Programming Languages and Systems, 7(1):80–112, 1985.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. In Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI), 2024.

Barbara Hayes-Roth. A blackboard architecture for control. Artificial Intelligence, 26(3): 251–321, 1985.

Francis Heylighen. Stigmergy as a universal coordination mechanism I: Definition and components. Cognitive Systems Research, 38:4–13, 2016.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, et al. Memory in the age of AI agents. arXiv preprint arXiv:2512.13564, 2025.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In Advances in Neural Information Processing Systems (NeurIPS), 2024.

Sajjad Khan. Verified detection and prevention of concurrency anomalies in multi-agent large language model systems. arXiv preprint arXiv:2606.17182, 2026a.

Sajjad Khan. S-Bus: Automatic read-set reconstruction for multi-agent LLM state coordination. arXiv preprint arXiv:2605.17076, 2026b.

Martin Kleppmann, Adam Wiggins, Peter van Hardenberg, and Mark McGranaghan. Local-first software: You own your data, in spite of the cloud. In Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!), pp. 154–178, 2019.

Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: a distributed messaging system for log processing. In Proceedings of the NetDB Workshop, 2011.

Sandeep S. Kulkarni, Murat Demirbas, Deepak Madappa, Bharadwaj Avva, and Marcelo Leone. Logical physical clocks. In Principles of Distributed Systems (OPODIS), LNCS 8878, pp. 17–32, 2014.

Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 21(7):558–565, 1978.

Haichang Li. Memory as a service (MaaS): Purpose-bound memory mediation for cooperative agents. arXiv preprint arXiv:2506.22815, 2025.

Zehao Lin, Xixuan Hao, Renyu Fu, Shaobo Cui, Kai Chen, Chunyu Li, Zhiyu Li, and Feiyu Xiong. A survey on long-term memory security in LLM agents: Attacks, defenses, and governance across the memory lifecycle. arXiv preprint arXiv:2604.16548, 2026.

Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS. In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP), pp. 401–416, 2011.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL), 2024.

Yanki Margalit, Nurit Cohen-Inger, Erni Avram, Ran Taig, and Oded Margalit. Governed shared memory for multi-agent LLM systems. arXiv preprint arXiv:2606.24535, 2026.

H. Penny Nii. The blackboard model of problem solving and the evolution of blackboard architectures. AI Magazine, 7(2):38–53, 1986.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. arXiv preprint arXiv:2310.08560, 2023.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST), 2023.

Nuno Preguiça, Carlos Baquero, and Marc Shapiro. Conflict-free replicated data types (CRDTs). arXiv preprint arXiv:1805.06358, 2018.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. arXiv preprint arXiv:2501.13956, 2025.

Alireza Rezazadeh, Zichao Li, Ange Lou, Yuying Zhao, Wei Wei, and Yujia Bao. Collaborative memory: Multi-user memory sharing in LLM agents with dynamic access control. arXiv preprint arXiv:2505.18279, 2025.

Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In Stabilization, Safety, and Security of Distributed Systems (SSS), pp. 386–400, 2011.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In Advances in Neural Information Processing Systems (NeurIPS), 2023.

Haoran Tan, Zeyu Zhang, Chen Ma, Xu Chen, Quanyu Dai, and Zhenhua Dong. MemBench: Towards more comprehensive evaluation on the memory of LLM-based agents. In Findings of the Association for Computational Linguistics: ACL 2025, 2025.

Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike J. Spreitzer, Marvin M. Theimer, and Brent B. Welch. Session guarantees for weakly consistent replicated data. In Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems (PDIS), pp. 140–149, 1994.

Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP), pp. 172–182, 1995.

Sergey Volkov, Yang Li, and Ye Luo. StateFuse: Deterministic conflict-preserving memory for multi-agent systems. arXiv preprint arXiv:2607.05844, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. Transactions on Machine Learning Research, 2024.

Yu Wang and Xi Chen. MIRIX: Multi-agent memory system for LLM-based agents. arXiv preprint arXiv:2507.07957, 2025.

Ziming Wang. TOKI: A bitemporal operator algebra for contradiction resolution in LLM-agent persistent memory. arXiv preprint arXiv:2606.06240, 2026.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In International Conference on Learning Representations (ICLR), 2025.

Frank F. Xu et al. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. arXiv preprint arXiv:2412.14161, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. In Advances in Neural Information Processing Systems (NeurIPS), 2025.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. G-Memory: Tracing hierarchical memory for multi-agent systems. arXiv preprint arXiv:2506.07398, 2025.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, pp. 19724–19731, 2024.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL), 2025.

A Full Gap Analysis of Existing Memory Systems

Table 5 extends Table 1 to the fourteen systems surveyed for this work, plus MemSwarm: every open-source memory system, agent-framework memory layer, and agent protocol we found with a plausible claim to shared multi-agent memory, assessed on the five capabilities a fleet substrate needs. ✓ = solid, ○ = partial, × = absent, n/a = not applicable (the system does not occupy that layer). The per-system notes below the table state the basis for each partial rating; star counts and last-push dates were checked 2026-09-12 via the GitHub API. Two patterns organize the table. First, every vendor that has solved distribution has solved it behind a managed service (Letta Cloud, Zep's Context Graph Engine, the Mem0 platform, Redis Iris): the open-source tiers stop at a single store or a single server. Second, no system in any row defines memory-level consistency semantics (read-your-writes across agents, causal ordering, or concurrent-conflict surfacing) as a first-class contract; where consistency exists at all it is inherited from a backing database.

Table 5: Fleet-memory capability matrix across the 14 surveyed systems and MemSwarm (✓ = solid, ○ = partial, × = absent, n/a = not applicable). Ratings are justified line-by-line in the notes below.

System	Persist	Share	Distribute	Consistency	Access control
Mem0 (OSS) (Chhikara et al., 2025)	✓	○	○	×	×
Letta (Packer et al., 2023)	✓	○	×	○	○
Graphiti/Zep (Rasmussen et al., 2025)	✓	○	○	○	×
cognee	✓	○	○	×	○
ReMe (ex-MemoryScope)	✓	○	×	×	×
LangMem	n/a	○	○	×	×
Redis Agent Memory Server	✓	○	○	○	○
LangGraph Store	✓	✓	○	○	×
AutoGen / MS Agent Framework	○	○	×	×	×
CrewAI	○	○	×	×	×
OpenAI Agents SDK	✓	×	○	○	×
Claude Agent SDK	✓	○	×	×	○
MCP memory servers	✓	○	×	×	×
A2A v1.0	n/a	×	✓	×	○
MemSwarm (ours)	✓	✓	✓	✓	×

Per-system notes. One line per row; stars/dates as checked 2026-09-12 via the GitHub API.

- Mem0 (OSS): sharing runs through user_id/agent_id/run_id scopes over one store, distribution comes only from the backing vector DB, extraction is ADD-only with last-write-wins updates, and access control is platform-only. mem0ai/mem0: 65,193 stars, Apache-2.0, last push 2026-09-11.
- Letta: memory blocks attachable to multiple agents (a genuine shared primitive); cross-machine memory positioned as Letta Cloud (proprietary); consistency = serialization through one server; coarse access control. letta-ai/letta: 24,714 stars, Apache-2.0, last push 2026-09-10 (active development moved to letta-code, 3,315 stars).
- Graphiti/Zep: bi-temporal fact validity (valid_at/invalid_at), the closest OSS consistency story; sharing via one graph group_id; distribution inherited from the graph backend; access control not in OSS (scale tier is the proprietary Context Graph Engine). getzep/graphiti: 30,835 stars, Apache-2.0, last push 2026-09-11.
- cognee: extract-cognify-load pipelines to knowledge graph + embeddings over pluggable stores; per-dataset sharing, basic dataset permissions; no cross-agent write coordination. topoteretes/cognee: 30,656 stars, Apache-2.0, last push 2026-09-12.
- ReMe (ex-MemoryScope): a local-first Markdown workspace in which "shared across agents" means the same filesystem, with no locking or consistency. agentscope-ai/ReMe: 3,448 stars, Apache-2.0, last push 2026-09-11.
- LangMem: a memory algorithm toolkit (extract/consolidate/update) that runs over any store and leaves the substrate role to that store; namespaced sharing and distribution are whatever the store provides. langchain-ai/langmem: 1,660 stars, MIT, last push 2026-09-09.
- Redis Agent Memory Server: one server with REST/MCP scopes, Redis atomicity, API-key auth; demoted to a V0/ "research foundation" in 2026 in favor of managed Redis Iris. redis/agent-memory-server: 314 stars, Apache-2.0, last push 2026-09-09.
- LangGraph Store: namespaced cross-thread/agent KV with vector search, the only widely deployed cross-agent store; a single backing database supplies all distribution and consistency; no memory-level ACLs. langchain-ai/langgraph: 41,539 stars, MIT, last push 2026-09-11.
- AutoGen / Microsoft Agent Framework: Memory protocol and context providers; shared cross-agent memory is wire-your-own; AutoGen in maintenance mode since April 2026, succeeded by Microsoft Agent Framework 1.0. microsoft/autogen: 60,948 stars, last push 2026-09-11.
- CrewAI: crew-scoped local memories (Chroma short-term + SQLite long-term) inside one process; SQLite locking issues under concurrency; nothing across processes. crewAIInc/crewAI: 58,415 stars, MIT, last push 2026-09-11.
- OpenAI Agents SDK: the Session protocol persists transcripts (SQLite/SQLAlchemy/Redis/Dapr or server-side conversation ids) and stops at the transcript layer, leaving semantic memory to the application; consistency = the chosen backend; nothing cross-agent. openai/openai-agents-python: 29,389 stars, MIT, last push 2026-09-11.
- Claude Agent SDK: file-based memory (CLAUDE.md, memory-tool files; workspace-scoped stores in the managed tier); cross-agent sharing = shared files, workspace-level scoping, no retrieval layer or consistency. anthropics/claude-agent-sdk-python: 8,086 stars, MIT, last push 2026-09-11.
- MCP memory servers: the reference server-memory is a local JSON-file knowledge graph with a single writer; sharing means multiple clients of one server; MCP standardizes the interface to memory and leaves the substrate to whatever sits behind the server. modelcontextprotocol/servers monorepo: 90,281 stars.
- A2A v1.0: a task/message wire protocol with signed agent cards and multi-tenancy, genuinely distributed, whose scope stops short of any shared persistent memory concept. Spec v1.0 released 2026-04-09 under the Linux Foundation (AAIF).
- MemSwarm (ours): the evaluated prototype of this paper: persistent event-sourced log, shared across heterogeneous agents over MCP, replicated local-first, causal consistency with session guarantees and surfaced conflicts. Access control is deliberately marked × because the prototype enforces none (Section 6). Per-record provenance and surfaced conflicts are the hooks the future policy layer will act on.

B Record Schema, Event Log Format, and CRDT Merge Rules

B.1 Memory Record Schema

A memory record is an immutable (frozen) structure with the following fields. Supersession leaves the original record intact and produces a copy with valid_until closed.

B.2 Event and Log Format

Every operation is an immutable event: op ∈ {Observe, Supersede, Forget}, the full record it carries, actor (originating node), hlc, vclock (the origin's vector clock after incrementing itself), event_id (32-char hex UUID; the global identity and idempotence key), seq (the local log position, reassigned by each replica at append), supersedes (Supersede only: the target record id), and tags (Forget only: the OR-Set tags observed at the origin). The log is one append-only JSONL file per node (one canonical JSON object per line, sorted keys), fsynced on every append. Crash-safety contract: an append-only writer can tear only the

Field	Type	Meaning
id	32-char hex UUID	globally unique record identity
content	string	the remembered fact
entities	tuple of strings	entity mentions (for the retrieval layer)
mem_type	enum	episodic | semantic | procedural
scope	string	namespace; default global
writer, source, session	strings	provenance: agent, origin, session
hlc	(int, int, string)	transaction time: (physical ms, logical, node id)
valid_from, valid_until	HLC triple or null	world validity; null valid_until $\Leftrightarrow$ current
confidence	float in $[0, 1]$	writer-asserted confidence

final line, so on reload a parse failure confined to the tail is dropped and the file truncated back to the last complete event, while a parse failure before the tail raises an error and is never silently repaired. Abbreviated example line:

{”actor”:”A”,”event_id”:”9f3c...”,”hlc”:[1757662183042,0,”A”],
”op”:”SUPERSEDE”,”record”:{...},”seq”:17,”supersedes”:”b7e2...”,
”tags”:[],”vclock”:{”A”:12,”B”:5}}

B.3 Merge Rules

Vector clocks. Standard componentwise comparison yielding Before, After, Equal, or Concurrent; every local operation increments the origin's component, and applying any event merges its clock.

OR-Set (record collection) and Forget semantics. State is a grow-only map element $\rightarrow$ add-tags plus a grow-only removed-tag set and a grow-only tombstone set; an element is live iff it has a tag not in the removed set and its id is not tombstoned. The add-tag for record admission is derived deterministically from the event id (obs:<event_id>), so every replica applying the same event adds the identical tag and states converge exactly, without coordination. The textbook OR-Set is add-wins: a remove deletes only observed tags, so a concurrent add survives. The implemented Forget is deliberately stronger: besides removing the origin-observed tags, it tombstones the record id, and admission checks the tombstone set, so a Forget suppresses the record even against a concurrent or later-delivered Observe. In practice the semantics are therefore remove-wins, which we choose because Forget exists for retraction and compliance, where resurrection of a forgotten record by a racing re-add is the costlier error. The add/remove race that motivates add-wins in the OR-Set literature does not arise in the memory workload: record ids are minted uniquely per Observe and are never re-added, so a tombstone can only ever suppress the one record it names, and new knowledge enters as a new record (or a Supersede) carrying a fresh id of its own.

MV-Register (per superseded record). Entries are (value, vector clock) pairs. Absorbing a write (v, c) drops every entry whose clock is Before c, since such an entry is dominated. An existing entry that is After c causes the incoming write to be ignored, and an exact duplicate with equal clock and equal value is a no-op. Equal clocks with different values, the pathological case of a double-write under one clock, keep both values, and Concurrent entries coexist. The register is conflicted iff it holds more than one entry, and resolution is left to the caller's explicit policy.

Store application. Application is idempotent by event_id and order-tolerant. Observe admits the record. Supersede sets the target's close timestamp to the minimum HLC over all supersessions seen (the earliest supersession closes validity, deterministic across delivery orders), admits the replacement, and registers it in the target's MV-Register under the event's vector clock. Forget tombstones the id and removes the origin-observed tags. Admission honors a supersession or tombstone that arrived before the record itself, so a Supersede delivered ahead of its target Observe still materializes correctly. Every newly applied remote event is appended to the local log, which is what lets updates re-ship transitively.

C Handoff Soundness: Formal Statement and Proof Sketch

Setting. Replicas $R_1, \ldots, R_n$, each with an append-only local log. Every operation originates at exactly one replica (its writer) and is applied there before the operation returns (apply-before-ack). Anti-entropy works as follows. A sender S maintains a cursor $c_{S,R}$ per peer R, and each tick it sends the contiguous suffix of its log from $c_{S,R}$. The receiver applies the batch in order, deduplicating by event id and appending unseen events to its own log, then acknowledges; the cursor advances to the end of the batch only on acknowledgment. Let A_R denote the set of events applied at R; A_R is monotone non-decreasing over time.

Proposition 2 (Handoff soundness, formal) Under this protocol, let delegator D send a delegation carrying F, D's per-writer applied-event vector (frontier, Section 4) at the moment of delegation. If delegate B's replica delays B's first read until its own applied-event vector dominates F componentwise, then every event causally prior to the delegation (in particular every write D made or read before delegating) is in the applied set serving every read of B's session.

Lemma 1 (per-writer prefix delivery). At every moment, for every replica R and writer w, the w-authored events in A_R form a prefix of w's origin sequence, in origin order. Sketch. By induction on how events enter logs. w's own log trivially holds its events in origin order. Any other log receives w-events only inside batches, each a contiguous suffix $[c, len)$ of some sender's log; by the induction hypothesis the sender's log holds a prefix of w's sequence in order. Positions below c were delivered to and acknowledged-applied by the receiver earlier (the cursor advances only on acknowledgment), so every w-event preceding a batched w-event is either already applied at the receiver or occurs earlier in the same batch; in-order application with idempotent deduplication therefore preserves both prefix-ness and order. A dropped or unacknowledged batch leaves the cursor untouched and the identical suffix is re-sent (retry-without-advance), and transitive re-shipping from any intermediate replica repeats the argument, since applied remote events are appended to the intermediate's own log in application order. □

Lemma 1b (causal log order). Every replica's log is a linear extension of happened-before over the events it holds. Sketch. A local write is appended after everything its replica has applied, and a batch is a suffix of a sender's log whose earlier positions the receiver already applied (Lemma 1), so no event is appended before one it causally follows. □ The proposition assumes the delegate reads from one replica or through the session layer.

Lemma 2 (frontier dominance covers the delegator's knowledge). If R's applied-event vector dominates F, then A_R contains every event in A_D at delegation time. Sketch. Component $F[w]$ counts the w-authored events D had applied. (The token is this count vector, not the events' vector clocks; a vector-clock token would additionally need every log to be a linear extension of happened-before.) Dominance gives R at least $F[w]$ applied w-events, and by Lemma 1 those are exactly the first $F[w]$ events of w's sequence, the same ones D had. Union over writers covers A_D, which contains every event causally prior to the delegation: D's own writes (RYW: apply-before-ack) and everything D read, and D's writes are ordered after the reads that informed them by log order and HLC update (writes-follow-reads at the origin). □

Proof of the proposition. By Lemma 2 the first read of B's session is served by a state containing A_D; monotonicity of A_B extends this to every subsequent read of the session. □

Each assumption is discharged by the implementation: cursor-advance-on-ack in the sync loop, idempotent order-tolerant application with re-append in the store, apply-before-ack in the local write path. The evaluation's WFR harness checks the contrapositive observable (if the causally last record of a K=10 chain is visible, all $K-1$ predecessors must be) and observed 0 violations in 1,200 single-writer handoffs across seeds and conditions, including under 30% batch drop (Section 5).

D Additional Experimental Detail

D.1 Platform

All experiments ran on one consumer machine, under macOS with Python 3.9.6 and the standard library alone, free of third-party runtime dependencies. Each node is a separate OS process serving newline-delimited JSON over loopback TCP (ports 7100+), and clusters ran sequentially. All percentiles are recomputed from the raw JSONL samples, with cached summaries kept out of the analysis path. Every JSONL line embeds the full run configuration, including the store implementation and a co-load flag. Every file contributing to a reported number records store: real. The only exceptions in the released data are four early harness-validation files, honestly labeled store: stub in their embedded configs, that contribute to no number in this paper.

D.2 Full Staleness Sweep

Three seeds $\{7, 42, 1337\}$ $\times$ nine conditions, with 300 writes $\times$ 2 readers $=$ 600 samples per seed per condition (16,200 total). Every cell ran on 3 nodes at 200 ms anti-entropy. Values are means over seeds with (min-max) across seeds; lags in ms. Figure 3 plots pooled lag CDFs for four representative conditions; Figure 2 plots the sync-interval law.

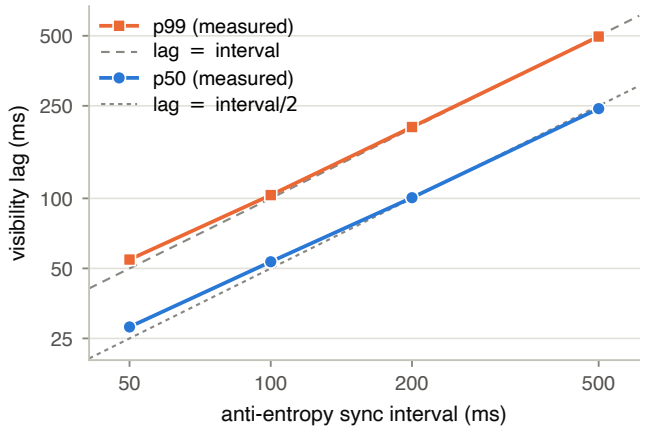


Figure 2: Measured p50 and p99 visibility lag against the anti-entropy interval (600 samples per point, recomputed from raw logs) with y=interval and y=interval/2 references: the staleness budget is set by one knob.

Table 6: Full staleness sweep: cross-replica visibility lag (ms), mean over seeds with (min-max) across seeds.

delay (ms)	drop	p50	p95	p99	never visible
0	0.0	101.6 (93.2-108.3)	194.2 (191.8-197.2)	202.8 (201.1-203.9)	0/1800
0	0.1	113.6 (109.2-116.3)	263.5 (244.0-283.4)	362.9 (347.8-375.7)	0/1800
0	0.3	139.1 (133.2-142.7)	395.3 (385.7-402.3)	564.1 (529.3-603.3)	0/1800
50	0.0	154.7 (144.7-161.6)	247.4 (244.5-251.1)	255.6 (253.2-258.4)	0/1800
50	0.1	170.2 (168.2-173.5)	339.4 (321.5-348.8)	442.7 (441.6-444.7)	0/1800
50	0.3	203.8 (202.0-206.5)	489.5 (455.4-515.4)	642.4 (604.6-670.8)	0/1800
200	0.0	301.6 (293.2-308.2)	395.8 (394.2-396.9)	403.7 (403.2-404.3)	0/1800
200	0.1	330.4 (319.7-336.5)	642.6 (609.6-664.9)	766.7 (744.3-794.5)	0/1800
200	0.3	367.0 (358.6-381.0)	827.9 (812.8-841.4)	969.3 (934.5-1004.5)	0/1800

For every seed independently, p50, p95, and p99 are monotone non-decreasing along both axes, drop at fixed delay and delay at fixed drop, with 0 violations across the 27 runs. Campaign-wide never-visible count: 0 of 23,200 samples (main sweep 16,200 + scaling 4,200 + sync-interval 2,400 + validation 400).

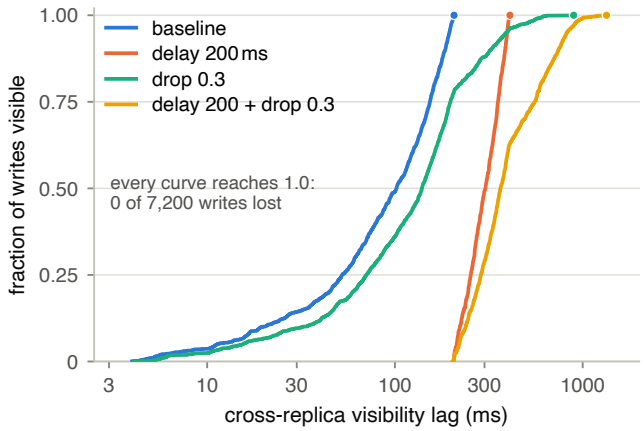


Figure 3: Visibility-lag CDFs under injected degradation, pooled over seeds $\{7, 42, 1337\}$ (1,800 samples per condition, recomputed from the raw per-sample logs). Constant delay shifts the distribution; drops fatten the tail. Every curve reaches 1.0, so degradation shows up as staleness while every record still arrives.

D.3 Node-Count Scaling

Baseline network, seed 42, 300 writes per point, 200 ms sync; full-mesh anti-entropy; ports base+i.

nodes	readers	p50	p95	p99	max	never visible
2	1	102.8	196.2	203.2	206.4	0/300
3	2	101.5	195.9	204.6	205.8	0/600
5	4	104.8	198.7	205.0	207.4	0/1200
8	7	105.8	201.1	205.3	207.1	0/2100

Per-reader p99 spread stays under 1.1 ms even at 7 readers. The writer pushes to all peers in the same tick, so fan-out adds only per-peer serialization cost at this scale. Caveat: all nodes share one machine; this bounds protocol fan-out cost, not network or CPU saturation at much larger n.

D.4 Sync-Interval Sensitivity

3 nodes, baseline network, seed 42, 300 writes per point. Expected under uniform write phase: p50 $\approx$ interval/2 + delivery cost, p99 $\rightarrow$ interval.

sync (ms)	p50	p50 − interval/2	p95	p99	max	min	never visible
50	28.0	+3.0	52.4	54.6	55.2	2.8	0/600
100	53.4	+3.4	100.5	103.3	104.5	3.6	0/600
200	100.7	+0.7	195.7	202.7	204.6	4.4	0/600
500	243.2	−6.8	476.9	496.4	505.5	3.7	0/600

The ∼3-5 ms delivery floor is visible as the minimum lag at every interval. The 500 ms point sits 6.8 ms below interval/2, the only negative deviation; the standard error of the median for a uniform 0-500 ms lag at N=600 is ≈10 ms, so this is ≈0.7 SE, consistent with sampling noise, flagged for completeness.

D.5 Methodology Notes

Phase locking (methodology note). A naive closed measurement loop, writing, polling to visibility, then writing again, synchronizes writes with the anti-entropy tick and produces a

degenerate distribution. In exploratory development runs we observed the median collapse to the full interval (≈200 ms at a 200 ms period). Those exploratory runs are not part of the released data; every released file uses the fix, seeded uniform jitter over [0, interval) before each write, after which the distribution matches the theoretical uniform-phase prediction.

Drop-RNG reproducibility scope. Seeding a per-node RNG with a tuple containing strings passes through per-process randomized string hashing, so nominally seeded drop patterns were not reproducible across processes; the fix seeds with a string digested through SHA-512. Post-fix reproducibility was validated by running one condition (delay 0, drop 0.3, seed 42) twice and comparing per-draw drop logs. The two runs agreed draw-by-draw on every node over the full common prefix, at A/B/C = 249/219/224 draws with 0 mismatches in either the drawn value or the drop decision, and their per-node drop totals agreed as well at 66/67/71. The 27-file main sweep predates this fix: its distributional claims are unaffected (each run's drop pattern was an independent random draw at the configured rate), but those files' per-run drop sequences are not reproducible from their seeds; runs made after the fix are.

Sample-size deviation. The experiment plan specified 500 writes per condition. The sweep on disk ran at 300 writes (600 samples) per condition, complete and internally consistent across all 27 files. It is recorded at the size it ran, and a re-run at the planned size remains outstanding.

Baseline p50 seed spread. Baseline p50 spreads 93.2-108.3 ms across seeds (≈2.6σ for a uniform-phase median at N=600, the widest relative spread in the grid) while the same cells' p95/p99 cluster within ≈2-5 ms; most plausibly phase-sampling noise plus machine-load drift between back-to-back sweeps. Baseline p50 should not be quoted at sub-10 ms precision.

Co-load disclosure and quiet re-runs. The original conflict and session-guarantee campaigns were taken while another CPU-active campaign shared the machine (flagged co_load in every affected file), as were two staleness cells (seed 42, delay 200 ms at drop 0.1 and 0.3) whose wall-clock windows overlapped that campaign. Post-submission auditing of raw wall clocks found the node-scaling and sync-interval sweeps also overlapped that campaign without carrying a flag (the staleness runner predated the co_load field). All of the above were re-run on a quiet machine, the conflict and session campaigns with seeds {42,7,1337} and both sweeps cell-for-cell: every logical outcome matches the flagged runs exactly, and every timing percentile shifts by less than one sync tick (conflict convergence p50 185.4 co-loaded vs 186.3 quiet at seed 42; both re-run sweeps within ±4 ms at every percentile of every cell). One anomaly survives quiet: the 500 ms sync point sits 6.8 ms below interval/2 in both runs, which share the seeded jitter sequence; with a median standard error of ∼7 ms at N=600 this is consistent with seed-fixed phase sampling. The main text quotes quiet numbers wherever both exist; the flagged originals remain in the artifact. The original incidence campaign also ran co-loaded; it was re-run quiet at three contention levels with exact Poisson intervals (all quiet-vs-co-loaded intervals overlap), the co-loaded originals remain in a labeled table, and two originally unpooled episode files are now included with a provenance note. LLM-arm cell provenance is under Handoff-Task Configuration below.

D.6 Conflict-Experiment Configuration

Each trial opens with a base fact written at A and confirmed visible everywhere. A partition $A \mid \{B, C\}$ is then injected by losslessly pausing both directions of the link, so per-peer cursors hold the backlog, healing is symmetric and nothing is dropped. Under 20 ms seeded write jitter, A supersedes with X and B with Y. The link heals and all nodes are polled to convergence. MV runs, 100 trials at seed 42: conflict surfaced on all 3 nodes 100/100, survivors were exactly $\{X, Y\}$ 100/100, cross-node agreement held 100/100, and a plain query() reader saw both sides 100/100. False conflicts during partition were 0, heal→convergence measured p50 185.4 / p95 199.2 / p99 204.6 / max 210.1 ms, and timeouts were 0. The LWW ablation applies harness-layer max-HLC resolution over the identical store state via conflict_values. It destroyed the loser silently 100/100 with zero conflict signal, reached winner agreement 100/100, and split X/Y 50/50 under the 20 ms jitter. Race-window runs used no partition, seed 7, and 100 trials per window. Cases truly concurrent by vector clock

came to 100/100, 100/100, and 96/100 at the 10/50/200 ms windows, and all 296 concurrent cases surfaced. The 4 causally ordered cases at 200 ms resolved to the later write (3 X / 1 Y). Determinism: identical seeded configurations re-run on fresh clusters reproduce the per-trial outcome signature exactly for both modes (convergence times are wall-clock and expected to differ). One measurement race in smoke testing, reading conflicts() before stats(), produced a single false negative. Ordering reads against monotone state fixed it, after which all 540 final trials show zero false negatives.

D.7 Incidence Campaign

A separate campaign measures how often such failures arise when only the timing is left natural. Three concurrent agents, one per node, maintain a shared runbook with no barriers or seeded offsets, while a –contention knob controls how often work items target the same entry. Runs are quiet, exact Poisson intervals (Garwood, 1936), 60 episodes and 1,440 ops per level. Truly concurrent same-record supersessions fall from 0.97 [0.53, 1.63] per 100 operations under hot-key contention (14 events, 8 still live conflicts at episode end) to 0.21 [0.04, 0.61] with organic references (3 events), stale reads from 0.69 to 0.49; LLM agents at high contention reach 2.47 [0.99, 5.10] (7 events). The rate is set by the workload's contention level, and at organic contention the episode bootstrap admits zero while the pooled Garwood interval does not; what the campaign shows is that uncoordinated timing alone produces these races whenever agents touch the same records.

D.8 Session-Guarantee Configuration

200 trials per property per condition per seed, seeds {42,7,1337} (quiet machine), 3 nodes, 200 ms anti-entropy; degraded = 50 ms one-way delay + 30% batch drop. RYW writes at a node and immediately, at 0 ms think time, reads back at the same node for the guarantee and at a different node for the contrast. The cross-node miss rate at 0 ms think time is expected to be ≈100% under a 200 ms period and is the empirical case for session stickiness, and a never-visible count of 0 confirms that every miss is staleness that resolves later. MR compares a reader alternating between B and C against one pinned to a single replica, checking that observed versions never regress. Degraded bouncing was violated in 49/60/47 trials at seeds 42/7/1337 (pooled 156/600, 26.0%), while the pinned reader stayed at 0/600 in both conditions. WFR single-writer uses a K=10 causal chain, where a violation is any predecessor missing when the causally last record is visible. The result was 0/600 clean (handoff p50 102.3-114.7 ms across seeds) and 0/600 degraded (p50 184.4-214.3 ms, max 896.5 ms). WFR cross-writer runs on 4 nodes. W_1 writes 5 at A, D at B reads them, writes 5 referencing them, and hands off its frontier, after which the reader at C waits for dominance and reads all 10. With the wait there were 0 violations and 0 wait timeouts in 1,200 handoffs, and immediate reads without the wait missed 600/600 per condition. Wait p50 was 203.5 (clean) / 202.8 ms (degraded), p99 209.4 / 513.4 ms. Determinism: 20-trial reruns on two fresh clusters reproduce all logical pass/fail patterns exactly.

D.9 LLM Prompts (Verbatim)

Every prompt opens: "You must answer using ONLY the memory records below. Do not use any outside knowledge. MEMORY RECORDS:", followed by one line per retrieved record, - [writer=<w> hlc=<hlc>] <key>: <value> (lww shows only the max-HLC record per key), then "QUESTIONS: for each key below, give the value according to memory. Answer with exactly one line per key, formatted 'key: value'." The standard flagging prompt then adds: "If memory contains MORE THAN ONE current record for a key, answer 'key: CONFLICT' instead of picking one." The neutral prompt instead adds: "If the records disagree, respond as you see fit." The adjudication prompt instead adds: "Org rule: researcher corrections supersede field reports (a record by writer 'researcher' is authoritative over a record by writer 'fieldagent' for the same key)." Each prompt ends with one line "What is the value of <key>?" per question. LWW prompts never mention conflicts. Of the 25 flagging-prompt misses (haiku 12, opus 1, sonnet 12), the 21 whose logs record the HLC winner all answered with the later-HLC record; the other 4 predate that field.

D.10 Handoff-Task and Growth Configuration

The agent task of Section 5.4 runs on 3 nodes plus partition control. A researcher writes 8 facts at A and supersedes 3 of them with corrected values. During a full partition a second writer at B supersedes 1 of the same facts with a conflicting value. After the heal, an implementer at C retrieves by scope and answers 8 questions. Scripted arm: 200 trials per condition per seed, seeds {42,43,44}; outcomes identical across seeds. LLM arm: runs of 10 to 25 trials at seeds 42, 43, and 45 (per-cell provenance below), invoked as claude -p <prompt> –model <alias> with aliases haiku, sonnet, and opus on 2026-09-13 and 2026-09-14. The harness sets no decoding parameters, so CLI defaults apply, and the resolved model snapshot was not recorded. Timeout 30 s; the prompt holds only the retrieved records plus questions; the mv+wait condition presents both conflicted values, lww presents the max-HLC winner only. Sonnet cells attempted 20 trials per condition, and 4 mv+wait and 1 lww trials hit the 30 s CLI timeout and are excluded (scored 16/20 and 19/20). That bounds the sonnet flag rate between 4/20 and 8/20 under worst/best-case imputation. Haiku cell provenance (authoritative; each run's file is in the artifact): mv+wait pools runs at seeds 42 (10+10), 43 (20, one timeout excluded), and 45 (4×25) for n=139; lww pools 42 (10+10), 43 (20), 45 (4×25) for n=140; no-wait pools one seed-43 run (20) and 45 (4×25) for n=120. Pooling rule: every completed standard-prompt haiku run of the flagging task enters its cell; no run was excluded. The seed-45, opus, and central+MV runs carry a conservatively-set co_load flag although no other campaign shared the machine during them; the adjudication campaign ran under the earlier co-loaded window and has no quiet re-run (its outcomes are deterministic and seed-identical, so timing co-load cannot affect them). Central+MV: the two writers' corrections arrive at one node as back-to-back supersessions of the original record in seeded random order. The row was re-run after fixing query() to hide register-dominated replacements (an earlier version returned both, which inflated this row to 1.000 through a signal that also fired for ordered corrections). The growth microbenchmark uses a single node plus one late joiner. Throughput and latency come from 5,000 sequential observes (7,973 ops/s, p50 0.122 ms, p99 0.181 ms, with a second seed at 8,184 ops/s), log size is sampled at 1k/5k/10k events (linear, 463 bytes/event), and bootstrap is timed from joiner launch to event-count and frontier equality with the seeder. Sync chunking caps pushes at 250 events per tick, so the measured 1,093 events/s ingest sits at the 250-per-200 ms ceiling. The first bootstrap run exposed a real protocol defect: sync batches were unbounded single NDJSON lines, and any backlog beyond roughly 140 events overflowed the peer's 64 KiB stream buffer, so the receiver rejected the line and the sender retried the same oversized batch forever; no backlog that size ever converged. Bounded 250-event chunks with cursor advance per acknowledged chunk (contiguity preserved) fix it, and the fix is why bootstrap throughput sits exactly at the chunk ceiling.

E Availability Experiment

Deployments: central, one MemSwarm node that all three agents use as clients; replicated, one node per agent host at 200 ms anti-entropy. Each agent issues one operation per 100 ms tick, alternating put and scoped query, for 1 s before, 3 s during, and after an outage, 20 trials per cell, seed 42. Crash hard-kills the process holding memory (central: the store; replicated: the implementer's replica, whose co-located agent is then down too) and restarts it from its on-disk log. Disconnect cuts the implementer's host: in the central deployment its client is pointed at a closed port (an emulation of a lost route), in the replicated one its replica is partitioned from both peers. Operations time out after 1 s.

After a replicated crash the restarted replica holds every acknowledged write within p50 59 ms (max 92 ms); after a replicated disconnect all replicas converge within p50 127 ms (max 223 ms). In the replicated disconnect, the cut-off implementer's successful reads miss 16.7 peer writes on average. A handoff token issued by the researcher halfway through the disconnect completed in 20/20 trials in both deployments, only after heal (central: first successful read 3 ms after heal; replicated: wait_frontier returns p50 95 ms after heal).

Table 7: Operations succeeding during a 3 s outage (20 trials × 30 operations per agent). Lost: acknowledged writes never visible after recovery.

Deployment	Failure	Researcher	Field agent	Implementer	Lost
Central	store host crash	0/600	0/600	0/600	0
Replicated	one host crash	600/600	600/600	(down)	0
Central	implementer cut off	600/600	600/600	0/600	0
Replicated	implementer cut off	600/600	600/600	600/600	0

F The Benchmark Suite

Each benchmark is one runner under benchmarks/ that drives real node processes through the NDJSON verbs of Section 4 and writes raw JSONL with its configuration embedded. A different store plugs in by implementing the MemoryStore contract that harness/node.py hosts (the in-memory stub in harness/_stub_store.py shows the minimal surface); the handoff task also has a runner for any store with LangGraph BaseStore put/get/search semantics.

Table 8: Benchmarks in the released suite and the metric each reports.

Benchmark	Runner	Metric
Staleness	staleness/run.py	t-visibility p50/p95/p99, never-visible count
Conflict	conflict/run.py	conflicts surfaced, values surviving, silent losses
Session guarantees	sessions/run.py	RYW, MR, WFR violations per trial
Cross-writer handoff	sessions/crosswfr.py	violations and wait time under frontier wait
Handoff task	agenttask/run.py	per-question accuracy, conflict flagged
External baseline	baseline/run.py	the handoff task on a central BaseStore or one node
Incidence	incidence/run.py	concurrent supersessions and stale reads per 100 ops
Growth	growth/run.py	throughput, log size, late-join bootstrap time
Availability	availability/run.py	operations served during a crash or disconnect, loss

G Property-Based Verification and a Mutation-Testing Finding

The implementation ships with a deterministic property-test suite (118 tests, including the frontier, session, MCP, and task-harness layers; seed printed on every run and reproducible via an environment variable). Each run exercises 2,200 randomized 3-replica CRDT convergence cases, checked both after shuffled operation delivery and after pairwise state merge. It then replays 1,000 HLC message-exchange interleavings under skewed, frozen, and backwards-jumping physical clocks, along with 60 randomized store sync interleavings. The join-semilattice laws of idempotence, commutativity, and associativity of merge each carry their own property test.

Mutation testing of this suite produced a methodological finding that mirrors Section 5's central result:

1. Convergence does not certify conflict preservation. A mutant MV-Register that silently applies last-write-wins to concurrent writes passes the standard convergence property: deterministic LWW still converges. Catching it requires an explicit assertion that all concurrent values surface. This is the testing-level mirror of the systems-level result (an LWW store passes any convergence check while destroying every concurrent contradiction) and suggests that CRDT test suites for conflict-surfacing types should treat "all maximal values present" as a first-class property alongside convergence.
2. Assert before the merge. An OR-Set mutant that removes elements without recording tombstones survived post-merge equality checks, because state merges uniformly resurrect tags and mask order-dependence. The refinement: assert replica equality immediately after operation delivery, before any state merge.

3. An HLC mutant in the receive branch was caught immediately by the happened-before property (receive timestamps strictly exceed send timestamps), confirming that property has teeth.