Claude Code Multi-Session Isolation

Claude Code Multi-Session Isolation

Status: Production (all 7 P0 resources verified SAFE)
Date: 2026-05-18
Owner: Petter Graff (architect), CodeCraft (implementation), Proveo (validation), Securion (threat review)
Parent MC: #101305 (Phase

2)

What Broke

From 2026-05-13 onward, ALAI runs **6+ concurrent Claude Code sessions daily** (12 sessions on 2026-05-15). Each session writes to shared state files with zero locking. On 2026-05-18 at 14:42, `~/system/memory/SESSION-STATE.md` was rewritten mid-session from session

`256da42c` to session `a10b7bc9` **between two reads in the same `/sync` skill invocation** — John's continuity context silently flipped to another session's "Next Steps."

Three CEO-visible collisions confirmed before probing began:
1. **Session continuity lost** — John's "Next Steps" overwritten by last-writer-wins across concurrent sessions
2. **Gate verdicts corrupted** — `last-validator-verdict.json` written by session A, read by

session B's `mc.js done`, passing/failing the wrong task
3. **Cost tracking undercount** — 1 of 4 concurrent Stop hooks' INSERTs lost in `costs.db`, causing `cost-tracker.js summary` to understate spend

The multi-session concurrency rate is accelerating: 6 sessions/day in May 2026 is 3× the February baseline. Without isolation, the collision surface grows quadratically.

Collision Ledger

Empirical probe evidence from `/tmp/session-collision-20260518T{143721,143735}/probe.jsonl` (T3 Phase 1):

P0 #	Resource	Path	Probe Verdict	Before-Fix Blast Radius
P0-1	SESSION-STATE.md	`~/system/memory/SESSION-STATE.md`	LAST_WRITER_WINS (A:line 6,

B:line 8) | John's continuity context; "Next Steps" lost between sessions |
| P0-2 | last-validator-verdict.json | `~/system/state/last-validator-verdict.json` | LAST_WRITER_WINS (A:line 26, B:line 36) | Gate verdict read by wrong session; silent `mc.js done` pass/fail corruption |
| P0-3 | .ledger-root-hash | `~/system/state/.ledger-root-hash` | LAST_WRITER_WINS (A:line 31, B:line 43) | Evidence integrity check bypassed; stale hash passed when ledger

changed |
P0-4	costs.db	`~/system/databases/costs.db`	SAFE at w=2 (A:line 16), LAST_WRITER_WINS at w=4 (B:line 22, 1 INSERT lost)	Financial audit trail undercount; CEO cost reports incorrect
P0-5	incident_mode flag	`/tmp/incident-mode`	LAST_WRITER_WINS (A:line 41, B:line 57)	One session's incident response silently cleared by unrelated session
P0-6	prompt_forge active	`/tmp/prompt-forge-active`

LAST_WRITER_WINS (A:line 46, B:line 64) | Model-override gate suppressed/enabled globally for all sessions |
| P0-7 | skill-registry.db | `~/system/databases/skill-registry.db` | LAST_WRITER_WINS at w=2 (A:line 21, 1 increment lost), non-deterministic at w=4 (B:line 29 SAFE) | Skill-use telemetry undercount degrades routing decisions |

Probed: 8 of 71 T1 inventory resources. P1 (13 resources)

and P2 (14 resources) deferred.

Isolation Model

Seven P0 collisions ? five patterns applied:

Pattern 1: Per-Session-Path (P0-1, P0-2, P0-5, P0-6)

Each session writes to `-.` instead of a single global file. At session boot (P0-1 only), compaction merges all per-

session files with mtime ? 4h into canonical view.

Implementation:
- P0-1: `SESSION-STATE-.md` written by `session-ledger.sh`; compacted by `enforce-next-steps.sh` at boot (lines 62-108); cleanup in `parent-session-cleanup.sh` (line 74)
- P0-2: `last-validator-verdict-.json` written by `session-output-validator.sh` (lines 491, 549); `mc.js done` reads per-session path (lines 2939-2966) with fail-closed gate if absent

- P0-5: `/tmp/incident-mode-` written by `incident-response-mode.sh` (lines 31-42); orphan purge at 4h (lines 52-59)
- P0-6: `/tmp/prompt-forge-active-` set by `/prompt-forge` skill (SKILL.md Step 0, line 57); reader bypass in `sonnet-default-gate.sh` (line 108) and `claude-sonnet-default.sh` (line 16)

Rollback: Set `ISOLATION_SESSION_STATE_SCOPED=0`, `ISOLATION_VERDICT_SESSI

ON_SCOPE=0`, `ISOLATION_INCIDENT_SESSION_SCOPE=0`, or `ISOLATION_PROMPTFORGE_SESSION_SCOPE=0` to revert individual resources.

Pattern 2: Advisory Lock via lockf (P0-3)

macOS ships `lockf(1)` at `/usr/bin/lockf` (not GNU `flock(1)`). Exclusive lock wraps `mc.js ready` invocation; lock released by kernel on process death (SIGKILL-safe per T8 Q1

live test).

Implementation:
- `mc-ready-gate.sh` (lines 98-112): `lockf -k -t 30 ~/system/state/.ledger-root-hash.lock node ~/system/tools/mc.js ready`
- Lock file kept via `-k` flag for reuse
- Fail-closed: exits 2 if `lockf` binary absent

Rollback: Set `ISOLATION_LEDGER_HASH_FLOCK=0`.

Pattern 3: SQLite WAL + BEGIN IMMEDIATE + Retry (P0-4, P0-7)

SQLite Write-Ahead Log (WAL) mode + `BEGIN IMMEDIATE` transaction + application-layer retry loop (5 attempts: 0ms, 50ms, 100ms, 200ms, 400ms, 800ms backoffs).

Why BEGIN IMMEDIATE was required:
- T9 added `PRAGMA busy_timeout` but used DEFERRED transactions

(default in sqlite3)
- Under w=4 burst, multiple connections acquired SHARED locks simultaneously; first write triggered RESERVED lock race ? silent INSERT loss (costs.db) and UPDATE non-determinism (skill-registry.db)
- `BEGIN IMMEDIATE` acquires RESERVED lock upfront; only one writer proceeds, others get `SQLITE_BUSY` immediately and retry in application layer

Implementation:
- P0-4: `claude-cli-cost-hook.sh`

(lines 135-215): Python `isolation_level=None` (autocommit mode), `BEGIN IMMEDIATE`, INSERT, `COMMIT`, wrapped in retry loop
- P0-7: `skill-use-counter.sh` (lines 24-60): bash heredoc `BEGIN IMMEDIATE; UPDATE; COMMIT;`, wrapped in retry loop
- Both DBs already in WAL mode (confirmed: `sqlite3 "PRAGMA journal_mode;"` ? `wal`)
- Exit-code check + `BUSY_TIMEOUT_EXHAUSTED` /

`SKILL_DB_ERROR_FINAL` log on retry exhaustion

Rollback: Set `ISOLATION_SQLITE_WAL=0`.

Feature Flags

Six flags control isolation behavior (all default `1` = on):

Flag	Controls	Revert Path

`ISOLATION_SESSION_STATE_SCOPED` | P0-1 per-session SESSION-STATE | Revert `session-ledger.sh` write target; disable compaction in `enforce-next-steps.sh` |
| `ISOLATION_VERDICT_SESSION_SCOPE` | P0-2 per-session verdict | Revert `session-output-validator.sh` write path + `mc.js` done gate check |
| `ISOLATION_LEDGER_HASH_FLOCK` | P0-3 lockf advisory lock | Remove `lockf` wrapper

from `mc-ready-gate.sh` |
`ISOLATION_SQLITE_WAL`	P0-4 costs.db + P0-7 skill-registry.db BEGIN IMMEDIATE + retry	Revert to PRAGMA-only or bare INSERT/UPDATE
`ISOLATION_INCIDENT_SESSION_SCOPE`	P0-5 per-session incident flag	Revert `incident-response-mode.sh` to global `/tmp/incident-mode`
`ISOLATION_PROMPTFORGE_SESSION_SCOPE`	P0-6 per-session prompt-forge marker

Revert `sonnet-default-gate.sh` + skill SKILL.md to global path |

Set any flag to `0` in `~/.claude/settings.local.json` env block or export in hook environment to disable.

Validation

Final Evidence (T10-ter, MC #101325)

Four validation runs with

updated harness (sha256 `acdbcd6abea1f1085f7c88056e59c747d073da6756889e9dcf5d54babd0bcfe3`):

Run	Mode	Writers	Verdict	Probe Path
G	default	2	P0-4 SAFE [line 16], P0-7 SAFE [line 21]; P0-1/2/3/5/6 LWW expected in default mode	`/tmp/session-collision-20260518T160822/probe.jsonl` (sha256: `8da33aee...`)

| H | default | 4 | P0-4 SAFE [line 22], P0-7 SAFE [line 29]; P0-1/2/3/5/6 LWW expected | `/tmp/session-collision-20260518T160829/probe.jsonl` (sha256: `2c13824e...`) |
| I | per-session | 2 | All 5 per-session P0s SAFE (lines 5,9,13,17,21) | `/tmp/session-collision-20260518T160837/probe.jsonl` (sha256: `c20ebf1e...`) |
| J | per-session | 4 | All 5 per-session P0s SAFE (lines 7,13,19,25,31) | `/tmp/session-collision-

20260518T160843/probe.jsonl` (sha256: `cecccfc1...`) |

Stability: Run H repeated 3× (H-2, H-3, H-4) — P0-4 SAFE 3/3, P0-7 SAFE 3/3. Total: 4/4 SAFE at w=4 for SQLite resources.

Before-After Summary

P0 #	T3 Baseline (pre-fix)	T10-ter (post-fix)
P0-1	LWW at w=4	SAFE in

per-session mode (Run J line 7) |
P0-2	LWW at w=4	SAFE in per-session mode (Run J line 13)
P0-3	LWW at w=4	SAFE in per-session mode (Run J line 19, lockf)
P0-4	LWW at w=4 (1 INSERT lost)	SAFE at w=4 (Run H line 22, BEGIN IMMEDIATE)
P0-5	LWW at w=4	SAFE in per-session mode (Run J line 25)
P0-6	LWW at w=4	SAFE in per-session mode (Run J line

31) |
| P0-7 | LWW at w=2 (non-deterministic) | SAFE at w=4 (Run H line 29, BEGIN IMMEDIATE) |

Runbook

1. How to Detect a Collision

Run the collision harness against production state (read-only inventory mode) or against `/tmp` sandbox fixtures (write

mode):

```bash
# Production read-only inventory (lists shared resources, no writes)
bash ~/system/tools/diagnose-session-collision.sh --inventory-only

# Sandbox collision test — default mode (simulates pre-fix behavior for comparison)
bash ~/system/tools/diagnose-session-collision.sh --writers 4 --targets all
```

Sandbox collision test — per-session mode (simulates post-fix production)
bash ~/system/tools/diagnose-session-collision.sh --per-session-mode --writers 4 --targets per-session-all
```

**Expected output post-fix:**
- Default mode: P0-1/2/3/5/6 show `LAST_WRITER_WINS` (correct — single fixture path simulates the race), P0-4/7 show `SAFE`
- Per-session mode: All 5 per-
```

session P0s
(`session_state_ps`,
`last_verdict_ps`,
`ledger_hash_ps`,
`incident_mode_ps`,
`prompt_forge_ps`) show
`SAFE`

Verdict location:
`/tmp/session-collision-
/probe.jsonl` — each line is a
JSON verdict with fields: `ts`,
`resource`, `verdict`, `writers`,
`pre_hash`, `post_hash`,
`lost_writers`,
`deadlocked_writers`

2. How to Roll Back Any Single Isolation

Set the corresponding feature flag to `0`:

```bash
# Roll back P0-1 (SESSION-STATE per-session)
export ISOLATION_SESSION_STATE_SCOPED=0

# Roll back P0-4 + P0-7 (SQLite BEGIN IMMEDIATE)
export
```

ISOLATION_SQLITE_WAL=0

Roll back P0-3 (lockf on ledger-root-hash)
export ISOLATION_LEDGER_HASH_FLOCK=0
```

**Persistent rollback:** Add to `~/.claude/settings.local.json`:
```json
{
"env": {
"ISOLATION_SESSION_STATE_SCOPED": "0"

```
}
}
```

Validation: Re-run harness with the flag disabled to confirm rollback worked.

IMPORTANT: Rolling back P0-4 or P0-7 restores the LAST_WRITER_WINS collision at w=4. Only roll back if BEGIN IMMEDIATE is causing production deadlocks (none observed in 4 validation runs + 3 stability repeats).

3. How to Add a New Shared Resource to Isolation

When a new shared resource is identified (e.g., a new `/tmp/global-marker` file or a new SQLite DB):

Step 1: Add to inventory

Edit `~/system/specs/multi-session/shared-state-inventory.md` (T1 artifact):
- List the resource path
- Classify: `per-session` | `global-single-writer` | `global-multi-

writer` | `external-singleton`
- Cite the file/line that proves it is touched (e.g., `hook-name.sh:42`)

Step 2: Write a probe in the harness

Edit `~/system/tools/diagnose-session-collision.sh`:
- Add a `writer_` function that writes to a sandbox fixture
- Add a verdict function if the resource needs custom logic (e.g., per-session file enumeration, lock-attempt

counting)
- Add the resource name to the `TARGETS` array

Step 3: Run the harness

```bash
bash ~/system/tools/diagnose-session-collision.sh --writers 4 --targets
```

Step 4: Decide pattern from catalogue

From

`/Users/makinja/system/specs/multi-session/isolation-model.md` §2 (Pattern Catalogue):
- **per-session-path:** Single-consumer or append-only state (e.g., session logs)
- **advisory-flock (lockf):** Last-writer-wins file with single authoritative value (e.g., a hash file)
- **SQLite WAL + BEGIN IMMEDIATE + retry:** SQLite DB with concurrent INSERTs/UPDATEs
- **CAS lease (mc.js claim):** Cross-session resource

allocation (e.g., task claiming)
- **singleton-broker queue:** High-risk writes that need daemon supervision (e.g., MEMORY.md)
- **deprecate-and-replace:** The global resource is a design defect; eliminate it

Step 5: Implement the pattern

Follow the implementation notes in `isolation-model.md` §4 (Per-P0 Design Table). Add a feature flag (e.g.,

`ISOLATION_NEW_RESOURCE=1`) for rollback safety.

Step 6: Validate

Run `diagnose-session-collision.sh` with the new isolation enabled. Verdict must be `SAFE` at w=4.

Step 7: Update this runbook

Add the new resource to the Collision Ledger table above and document the chosen pattern + rollback flag.

Known Limitations

P1 Resources (13 total) — Not Yet Addressed

From `COLLISION-LEDGER.md` rows 8-17:
- `lightrag-ingest-health.json` — SAFE at w=2, LAST_WRITER_WINS at w=4 (2 of 4 increments lost)
- `evidence-ledger.jsonl` — not probed; suspected interleaved appends under concurrent `mc.js

done`
- `evidence-index.jsonl` — not probed; read at session boot without write lock
- Mehanik cleared markers (`/tmp/mehanik-cleared-`) — not probed; two sessions on same MC can both see cleared marker
- Evidence dirs (`/tmp/evidence-/`) — not probed; numeric sequence collision risk
- Claim schema stubs (`/tmp/claim-schema-.json`) — not probed; two sessions on same MC write conflicting schemas

- Hop-build started markers (`/tmp/hop-build-started-`) — not probed; 8 stale files present; double-build or skip-build risk
- Opus override token (`/tmp/opus-override-token`) — not probed; non-atomic consume allows two sessions to bypass cost gate
- John bash override token (`/tmp/john-bash-override-token`) — not probed; same TOCTOU as opus token
- MCP Playwright server (singleton) — not probed; unknown whether browser

contexts are session-isolated
- LightRAG ingest API (`http://localhost:9621`) — not probed; concurrent POST from all sessions; LightRAG's own concurrency handling unverified
- MEMORY.md daemon write path — not probed; memory-writer.js queue serialisation under concurrent flush requests

Require Phase 2 sprint 2 or explicit CEO scope expansion.

P2 Resources (14 total) — Design-Quality Improvements

From `COLLISION-LEDGER.md` rows 18-27:
- `blueprint-override-ledger.jsonl`, `h-ready-audit.jsonl`, `verdict-ledger.jsonl`, `daily-logs/.md`, `GOTCHA-task-.md`, `hivemind.db`, `knowledge.db`, `session-save.log`
- No CEO-visible blast radius confirmed in T3
- Deferred to backlog

MCP Singleton Servers — Unprobed

- Playwright browser: unknown whether page state leaks between concurrent `mcp__playwright__navigate` calls
- Docker MCP: unknown whether container state is session-isolated
- Spreadsheet MCP: unknown whether workbook handles are session-scoped

Require separate external-service isolation plan.

Harness Measures /tmp

Clones, Not Live State

The collision harness writes to `/tmp/session-collision-/fixtures/`, not production paths. Verdicts are correct for concurrency pattern analysis but do not directly measure live production contention. The harness is a structural test, not a load test.

To measure live contention: inspect hook execution logs (`~/system/memory/logs/hook-execution.log`) for `BUSY_TIMEOUT_HIT`

(costs.db) or `SKILL_DB_ERROR_FINAL` (skill-registry.db) occurrences during high-concurrency periods.

Out-of-Scope

The following were explicitly excluded from Phase 2:

1. **P1 resources** (13 items listed above) — require separate plan
2. **P2 resources** (14 items

listed above) — backlog
3. **External singletons** (MCP servers, LightRAG, Qdrant, Ollama) — require external-service isolation plan
4. **Hook scratch state** not in T3 probe surface:
- MEMORY.md direct write path (protected by mmwb daemon redirect)
- `settings.local.json` (CEO-only writes per T1 classification)
5. **Legacy /tmp markers** cleanup (8 stale `hop-build-started-*` files present) — cleanup cron needed but

collision risk unprobed

No existing hook was removed in Phase 2. Any future removal requires named CEO approval.

Architecture Notes

Why lockf, Not flock?

macOS 25.2.0 does not ship `flock(1)` (util-linux). macOS provides `lockf(1)` at `/usr/bin/lockf`, which uses BSD

`flock(2)` kernel primitive.
Semantics:
- `flock -x lockfile cmd` ? `lockf -k -t 30 lockfile cmd`
- `-k` keeps the lock file on exit (required for reuse)
- `-t N` sets timeout in seconds (0 = non-blocking)
- Lock is released by kernel on any process death (SIGKILL-safe, confirmed by T8 Q1 live test + POSIX spec)

Why BEGIN IMMEDIATE, Not Just PRAGMA busy_timeout?

SQLite default transaction mode is DEFERRED: `BEGIN DEFERRED` acquires no locks until the first write. Under w=4 burst with WAL mode:
1. Four connections open
2. Each executes `PRAGMA busy_timeout=5000`
3. Each executes `INSERT` (implicit BEGIN DEFERRED)
4. All four acquire SHARED locks
5. First write attempts to upgrade to RESERVED — succeeds
6. Other three attempt upgrade — all get SQLITE_BUSY

7. **But** the PRAGMA busy_timeout retry only applies if the lock was unavailable at BEGIN time. Since all four acquired SHARED before any write, the retry mechanism is bypassed.

Result: 1 of 4 INSERTs succeeds, 3 fail silently (exit code 5 from sqlite3 CLI, which hook may not check).

`BEGIN IMMEDIATE` acquires RESERVED lock upfront. Only one connection gets

RESERVED; others block (or get SQLITE_BUSY) at BEGIN, where busy_timeout applies correctly. Application-layer retry loop ensures all writers eventually succeed.

Why Compaction Only at Boot (P0-1)?

Per-session `SESSION-STATE-.md` files accumulate during the day. Compaction at boot (not at every session end) minimizes file I/O. The 4h mtime staleness filter ensures dead sessions' files

are ignored. Compaction uses atomic write (`tmp+mv`) to prevent partial-write corruption if `enforce-next-steps.sh` is killed mid-boot.

Why 4h Staleness Filter?

Claude Code sessions under normal use are ? 2h (median ~30min, p95 ~90min per session log analysis). 4h allows for extended debugging sessions (e.g., CEO deep-dive on a single task) while filtering overnight orphans. Session files older than

4h at boot time are assumed stale and skipped in compaction.

WAL Sidecar Files

WAL mode creates `-wal` and `-shm` sidecar files next to each SQLite DB:
- `-wal`: Write-Ahead Log (contains uncommitted writes)
- `-shm`: Shared memory index (used by readers to find data in WAL)

**NEVER manually delete these files while any Claude Code

session is running.** Deleting them corrupts the DB. macOS purges `/tmp` on reboot, but `~/system/databases/` is persistent — sidecar files remain until a checkpoint flushes them.

To verify WAL mode is active:
```bash
sqlite3 ~/system/databases/costs.db "PRAGMA journal_mode;"
# Output: wal
```

To revert to DELETE mode

(NOT recommended unless WAL is causing issues):
```bash
sqlite3 ~/system/databases/costs.db "PRAGMA journal_mode=DELETE;"
```

Evidence Files

All referenced evidence paths are archived in `~/system/specs/multi-session/`:

File	Purpose	Lines	sha256
`COLLISION-LEDGER.md`	T5 ranked ledger, 28 resources	128	(T5 final version)
`isolation-model.md`	T7 P0-only design	194	(T7 final version)
`threat-review-t8.md`	T8 Securion review	244	(T8 final version)
`t9-implementation-log.md`	T9 P0 implementation	251	(T9 final version)
`t9-bis-implementation-log.md`	T9-bis harness + P0-6 writer

159 | (T9-bis final version) |
`t9-ter-implementation-log.md`	T9-ter SQLite BEGIN IMMEDIATE	148	(T9-ter final version)
`t10-ter-validation-report.md`	T10-ter PASS evidence	169	(T10-ter final version)
`/tmp/session-collision-20260518T160822/probe.jsonl`	Run G (w=2 default)	50 lines	`8da33aee...`
`/tmp/session-collision-20260518T160829/probe.jsonl`	Run H (w=4 default)	53 lines	`2c13824e...`

| `/tmp/session-collision-20260518T160837/probe.jsonl` | Run I (w=2 per-session) | 25 lines | `c20ebf1e...` |
| `/tmp/session-collision-20260518T160843/probe.jsonl` | Run J (w=4 per-session) | 33 lines | `cecccfc1...` |

Harness location: `/Users/makinja/system/tools/diagnose-session-collision.sh` (1013 lines, sha256 `acdbcd6a...` post-T9-ter).

Related Documentation

- [MC Claim Protocol](https://docs.alai.no/books/infrastructure/page/mc-claim-protocol) — Cross-session task claiming via CAS lease (already production before this work)
- [ADR-024 Agent Team Topology](https://docs.alai.no/books/system-architecture/page/agent-team-topology-adr-024) — Agent process supervision (single-session scope)
- [ZAKON

NULA](https://docs.alai.no/books/rules/page/zakon-nula-tool-first) — Tool-first doctrine that drove the debug-before-solution mandate (T6 phase gate)

Created: 2026-05-18
Last Updated: 2026-05-18
Plan: `/Users/makinja/system/specs/claude-code-multi-session-isolation-plan.md` (207 lines)
MC Parent: #101305 (Phase 2)

Evidence Integrity: All verdicts cite probe.jsonl line numbers; no LLM inference in ledger or validation

Related Pages

- Phase 3 P1 Sweep Log
- T10-quad Validation Report

Revision #3
Created 2026-05-18 18:35:46 UTC by John
Updated 2026-06-21 20:03:44 UTC by John