SERIES

THE RECORD · FOUR PAPERS · OPEN LICENCE

The First *Layer*.

Working notes on the memory beneath one project's agent sessions, and the budgets that keep it loadable

PAPERS

ISSUED

four papers · published 21 September 2026 · CC BY 4.0

digital1.foundation/articles/the-first-layer/

Stichting Digital One Foundation i.o. · Amsterdam

These notes describe work contributed to an entity in formation.

TOC

THE FIRST LAYER · FOUR PAPERS · OPEN LICENCE

Contents

001

PAPER 001 · METHOD

The transcript is not the *memory*.

The largest file a long coding session leaves behind is its own conversation. On one workstation, one session's transcript reached 224 MB over seventeen days while a comparable project's transcript stayed at 18.6 MB on identical hooks and settings — accumulation, not configuration. The API beneath the session is stateless, so every request carries the conversation before it, and a session that never ends sits at its context ceiling permanently. **The transcript is an accumulation surface, not a memory**: what a session learned must be extracted into files, or the residue is paid for as long as the session lives.

PAPER · 001 PUBLISHED · 21 SEP 2026 READING · 8 MIN

Canonical · digital1.foundation/articles/the-first-layer/001-the-transcript-is-not-the-memory.html

§1

One session accumulated 224 MB in seventeen days

The measurement that opens this series was not made for these notes. It was recorded on 15 September 2026 in the protocol file that loads at the start of every session on the estate it governs — one operator's machine, its workspace folders and two product repositories. The record: one project "reached a 224 MB transcript over seventeen days in a single session, against 18.6 MB in a comparable one — identical hooks and settings, so it was accumulation, not configuration". The comparable project is described in that file and named nowhere on disk, so the comparison survives only as recorded. The transcript itself can be re-measured, and was.

DEFECT · FOUND BY THE OPERATOR'S OWN PROTOCOL FILE, RECORDED 15 SEPTEMBER 2026; RE-MEASURED BY LISTING, 21 SEPTEMBER 2026

A transcript reached 224 MB in seventeen days of one session; a comparable project stood at 18.6 MB on identical hooks

Nothing failed loudly; what changed was what every answer cost. Re-measured six days after the record, the same file was the largest transcript under the machine's projects directory:

```
# recorded 2026-09-15, in the operator's protocol file
    one project        224 MB   seventeen days, a single session
    a comparable one   18.6 MB  identical hooks and settings

    # re-measured 2026-09-21, same machine
    ls -l ~/.claude/projects/*/*.jsonl | awk '{print $5, $9}' | sort -rn |
    head -1
    236422970  …-longevity/15bbd718-0521-40f9-ab46-2eb650dcd9a8.jsonl
```

The re-measurement, 236,422,970 bytes on 21 September 2026 — 226 MB as `du -sm` rounds it — sits above the 224 MB recorded six days earlier: the accumulation had continued while the record aged. Both figures are kept in this paper, each with its date, because the gap between them is itself the finding restated.

THE FILE IS THE RESIDUE, NOT THE PAYLOAD

A 236,422,970-byte file is not what any single request carries — that claim would be false, and these notes do not make it. The context window is bounded; when it fills, older turns are summarized or dropped, and the requests keep working. The file is a session's on-disk accumulation — a receipt, not a payload — and what it proxies is the condition the protocol file itself names: "A session left running for days sits at the context ceiling permanently and a one-line question costs what a hard one does." The file is the residue; the ceiling is the bill.

Neither session was misconfigured, which is the point of the pair. The rest of this paper derives why the surface accumulates, why the accumulation returns less than it costs, and what has to leave the transcript for a session to stay cheap.

§2 A ceiling reached is a ceiling kept

The API beneath these sessions holds no conversation between calls. Anthropic's Messages API reference states it in one sentence: "The Messages API can be used for either single queries or stateless multi-turn conversations."[1] The conversation lives in the client's request: "When creating a new `Message`, you specify the prior conversational

turns with the `messages` parameter, and the model then generates the next `Message` in the conversation."[1] The same vendor's context-window documentation describes the consequence turn by turn: "Progressive token accumulation: As the conversation advances through turns, each user message and assistant response accumulates within the context window, and previous turns are preserved completely", and the input phase of each turn "[c]ontains all previous conversation history plus the current user message".[2]

The operator's protocol file compresses those sentences into the clause its budget rests on: "cost and latency track the TRANSCRIPT, not the task". The serving layer softens the arithmetic — Claude Code's own cost documentation says it "automatically optimizes costs through prompt caching, which reduces costs for repeated content like system prompts, and auto-compaction, which summarizes conversation history when approaching context limits"[3] — so the accumulated history is not re-processed from nothing on every call.

Caching changes the price, not the shape. In the vendor's words, "[i]nstead of reprocessing the same large system prompt, document, or conversation history on every request, the API reads from cache at a fraction of the standard input price".[4] A fraction of the price is still a price with the history's size in it, charged on every request that carries the history — and compaction bounds the history's length without emptying it. Neither changes what the residue is. The vendor names what remains after both: "As token count grows, accuracy and recall degrade, a phenomenon known as context rot. This makes curating what's in context just as important as how much space is available."[2] A ceiling that compaction keeps a session under is still a ceiling the session sits at, permanently, from the first day it is reached, where the smallest question is billed like the hardest. The session does not get cheaper by finishing its task. It gets cheaper by carrying less.

§3 Most of what accumulates is not load-bearing

The cost side of that trade has a plain statement in the literature: "the time and memory complexity of self-attention are quadratic in sequence length."[5] That paper is about mitigating the cost, and modern serving mitigates it further, so the realized curve is gentler than quadratic — but per-request compute still scales with what is carried, and the thing carried here is the conversation.

The return side scales worse. Models use long inputs unevenly: performance on retrieval from long contexts degrades significantly when the relevant material sits in the middle of the input, with the ends favoured, even on models built for long contexts.[6] And advertised length overstates usable length: of models claiming contexts of 32K tokens or more, "only half of them can maintain satisfactory performance at the length of 32K" on synthetic tasks harder than simple lookup.[7]

Both results are measured on 2023- and 2024-era models with constructed retrieval tasks rather than agent transcripts, and they speak to accuracy, not cost — so they establish the direction of the effect on today's models, not its size.

Meanwhile the tokens themselves are mostly droppable. A prompt-compression method reports "up to 20x compression with little performance loss" across four benchmarks[8] — a measured way of saying that most of what a long input carries, the answer never needed. What that line of work does not characterize is which kinds of content compression silently loses on agent histories: a correction, a decision, an invariant. Dropped corrections and stale invariants are precisely the loss a summary can commit silently, and they are exactly the sentences the rest of this series is about keeping on purpose.

No one of these papers states the conclusion these notes need. Assembled — compute scales with length, accuracy does not, and most tokens are droppable — they support it: a long transcript costs more than it returns. That assembly is this paper's own synthesis, and the 224 MB case is its instance, not its proof.

§4

The residue is extracted, or paid for indefinitely

What deserves to survive a session is small and specific: the decisions made, the holes fallen into, the files touched, the next step. In a 236,422,970-byte transcript those sentences exist, at addresses nothing indexes, interleaved with seventeen days of everything else. Extraction is the act of writing them into files whose size is bounded by a rule rather than by a session's lifespan — and then ending the session that produced them.

The protocol measured in this series does it with ordinary files and two commands: a handoff that persists current state, then a clear. The vendor's cost page states the two exits: "`/compact` reads the conversation it summarizes, so compacting a large context is itself a large request. When you want a fresh start instead of continuity, `/clear` costs nothing".[3] The hook that polices the transcript budget ends its warning with the operative sentence: "Finish with /handoff, then /clear. The memory files are the continuity, not the transcript." The files load at the next session's start instead of riding along on every request in between.

THE FORK, PRECISELY

`/clear` costs nothing and keeps nothing. So the durable residue of a session — the decision, the correction, the one command that finally worked — has exactly two futures while it lives only in the transcript: ***lost at /clear***, or ***carried at the ceiling on every request until then***. A third future requires a third place.

Three limits bound this paper. The comparison that opens it is two projects on one machine, recorded once on 15 September 2026; the smaller of the pair cannot be re-identified today, so the pair survives as a dated record, not a benchmark. The file size is a proxy for accumulation, and these notes claim nothing about bytes per request. And nothing here measures what extraction loses — which facts deserve a file, how they are merged, and what forces the loop to run are the subject of Paper 002. The literature already built the third place as a managed runtime: MemGPT pages hierarchical memory tiers between the context window and outside storage.[9] The construction the next paper describes is that idea's plainer cousin — files in a directory, written by two commands and read by a hook.

ON THE PROVENANCE OF THIS MATERIAL

The transcripts and protocol files measured here belong to one practitioner's workstation, a Digital One machine, and are not public: the 224 MB record and its 18.6 MB counterpart are the operator's account, and the 236,422,970-byte re-measurement, printed with its command, can be checked in shape but not in substance by a reader. Vendor sentences are quoted verbatim from public documentation, each with the date read; no vendor is compared with another here, and none should be inferred. Deliberately absent: prices, plans and tiers, and deployment internals.

References

1 Anthropic, ***Messages API*** reference; the quotations in §2 are from the Create a Message description. platform.claude.com/docs/en/api/messages, read 21 September 2026.

2 Anthropic, ***Context windows***, section "How the context window works", including its "Progressive token accumulation" item; the "context rot" sentence in §2 is from the same section. platform.claude.com/docs/en/build-with-claude/context-windows, read 21 September 2026.

3 Anthropic, ***Claude Code — Manage costs effectively***; the auto-compaction sentence in §2 and the `/compact` and `/clear` sentences in §4 are quoted from it. code.claude.com/docs/en/costs, read 21 September 2026.

4 Anthropic, ***Pricing***, section "Prompt caching"; the quotation in §2 is that section's framing of what caching replaces. platform.claude.com/docs/en/about-claude/pricing, read 21 September 2026.

5 T. Dao, D. Y. Fu, S. Ermon, A. Rudra and C. Ré, "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness", NeurIPS 2022; arXiv:2205.14135. The quadratic-complexity sentence is from its abstract.

6 N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni and P. Liang, "Lost in the Middle: How Language Models Use Long Contexts", TACL (2023); arXiv:2307.03172.

7 C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekesh, F. Jia, Y. Zhang and B. Ginsburg, "RULER: What's the Real Context Size of Your Long-Context Language Models?", COLM 2024; arXiv:2404.06654.

8 H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang and L. Qiu, "LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models", EMNLP 2023; arXiv:2310.05736.

9 C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica and J. E. Gonzalez, "MemGPT: Towards LLMs as Operating Systems", arXiv:2310.08560 (2023).

002

PAPER 002 · CONSTRUCTION

A working memory made of *files*.

A cleared session starts fresh and remembers nothing it decided, and the vendor's documentation states the fresh window in its opening sentence. Paper 001 derived why the transcript cannot be the carrier; this paper describes the carrier built to replace it: an index that points, topic files that hold one fact each, a state file that is replaced at handoff, and a knowledge store that is **merged, never overwritten**, with hooks at the session's start and stop so the loop is not optional. The one phase no hook enforces is the one that decayed.

PAPER · 002 PUBLISHED · 21 SEP 2026 READING · 8 MIN

Canonical · digital1.foundation/articles/the-first-layer/002-a-working-memory-made-of-files.html

§1

A fresh window destroys exactly what was hardest to earn

The cheap exit from Paper 001's accumulation is one command. Claude Code's cost documentation prices it: "When you want a fresh start instead of continuity, `/clear` costs nothing".[1] And its memory documentation states what the command produces: "Each Claude Code session begins with a fresh context window."[2] The command costs nothing to run. What it costs the project depends entirely on what was carried out first.

What a session accumulates that is worth anything arrives late and slowly: which approach was chosen and why, which plausible path turned out to be a hole, which files are mid-change, what comes next. Cleared unextracted, each is re-derived at full price by the next session — or not re-derived, and the hole is found a second time. The instinct this produces is to never clear, so the knowledge stays loaded. Paper 001 measured where that road ends: seventeen days, 224 MB, a permanent ceiling.

The knowledge has to live somewhere cheaper than the conversation and faster than re-derivation. The idea has a literature: MemGPT pages state between the context window and external storage so an agent's memory outlives what the window holds.[3] The construction measured here is the low-technology end of that idea. The external tier is a directory of Markdown files, and the pager is a shell hook.

§2 An index that points, and topic files that hold

The protocol keeps two layers with one source of truth. The machine-local layer lives outside the repository — "Your project memory lives at `~/.claude/projects/<encoded-project-path>/memory/` (the project path with every `/` replaced by `-`)", as the protocol file puts it — and is loaded automatically when present. The shared layer is the repository's own `memory/` directory, committed and merged like code, which the repository's instructions declare "the shared knowledge base and the single source of truth for project facts". Durable facts are promoted from the local layer into the shared one at consolidation; nothing durable is allowed to end its life machine-local.

Each layer has the same shape: an index file, `MEMORY.md`, and topic files beside it. The index is governed by the protocol's bluntest rule: "under 80 lines — it is an INDEX, one line per topic … Detail lives in the topic file". Its line format is fixed, one pointer per lesson: `- [Title](file.md) — the hook, in one clause`. A live line from the machine-local index shows the shape filled in, quoted in full: `- [EU region rename](eu-region-rename.md) — 2026-07-24: eu-central = Budapest (was eu-east), eu-west = Frankfurt (was eu-central); code+DB+workers+policies all migrated (commit ffa166a, v0.4.6)`. A date, the fact, and enough of a hook to know when to open the file; nothing in the line argues or narrates. The topic file holds one fact, with the concrete story, a stated ***why***, and a stated ***how to apply***.

The 80-line ceiling is not neatness; it is the price of the read path. Whatever sits in the index is paid for at every session start; the detail must load on demand. The harness vendor states the identical tradeoff for its own always-loaded instruction files: "target under 200 lines per CLAUDE.md file. Longer files consume more context and reduce adherence", with the auto memory it writes for itself capped at the "first 200 lines or 25KB" per session.[2] Two budgets from two authors, one arithmetic: an always-loaded file is a standing tax, and only an index is cheap enough to load always.

Admission is the part that keeps the files small. The operator's writing guideline sets the bar: a memory records "something non-obvious that cost something to learn. Not what the repo already says, not what git history shows. The test: would a competent person repeating this work

fall into the same hole without it?" It closes the loop on error: "Write the memory even when — especially when — the lesson is that the earlier analysis was wrong." The estate's topic files show the habit; one is titled ***prove your check can fail***. The same guideline separates output from learning: "The paper is the artefact; the memory is the lesson." What a session produced is already saved by the repository; what it ***learned*** is saved nowhere unless the protocol saves it.

§3

State is replaced; knowledge is merged

A memory that only appends is a transcript with a better filename. Two write disciplines, deliberately different, keep this one from becoming that. The first is the handoff, run before a clear. It writes a session-state file that is per-developer and gitignored: "your own session continuity, never shared or committed". And it does not append: "REPLACE the file — current state only, not history; keep under ~80 lines". Its headings are the enumeration of what §1 said a clear destroys, verbatim from the command that writes them:

```
# .claude/session-state.md — replaced at every handoff
## Handoff: [output of `date` command]
## Working On
## Modified Files
## Decisions Made
## Gotchas Discovered
## Next Steps
## Blocked On
```

The second discipline is consolidation, a command the protocol frames as "Memory consolidation pass — like REM sleep", run after the handoff at the end of a productive session. It works in four named phases: "Phase 1 — Orient", "Phase 2 — Gather recent signal" (this conversation, the state file's durables, the local layer's promotions), "Phase 3 — Consolidate", and "Phase 4 — Update index". Its write rule is the opposite of the handoff's: "Merge, never overwrite." Changes are additive and atomic so git merges stay trivial; facts are stated as facts, not "we decided"; relative dates are converted to ISO; and deletion is part of the discipline, not a violation of it: "If a fact contradicts an existing one, replace the stale fact and note the date."

The published analog of this fold-forward move is recursive summarization, where new context is repeatedly folded into a running summary and long-conversation consistency improves over keeping raw history.[4] What that literature does not say is which sentences a summary silently drops — which is exactly why the consolidation here does not summarize wholesale: it admits one fact at a time, under §2's

test, with the why and the how-to-apply written out. State is replaced because only the present matters. Knowledge is merged because the past keeps mattering until a dated fact contradicts it.

§4

A loop enforced by hooks, and the phase that still decayed

None of this runs on discipline alone. At session start, the repository's hook injects three things before the first turn: the last five commits with the working-tree status, the memory index in full, and the first 40 lines of the state file. One of the estate's session-start hooks states the rationale for injection in general terms: "whatever they print is already in context when the session opens", at no tool call and no turn. When the state is two or more days old, the injection carries its own warning: "Commits made since then are NOT reflected in memory or session state. Trust 'git log' over the 'Active Context'."

The write side is guarded by two different stop hooks. The workspace's blocks a closing session once per session, with the instruction to "run /handoff to persist session state into project memory, then /dream to consolidate long-term memory." The repository's is throttled instead: it interrupts only when the state file is older than 45 minutes ***and*** the repository actually changed — a new HEAD or a changed status hash against a baseline snapshotted at session start. A session that changed nothing owes nothing, and the throttle is load-bearing: a loop that interrupts every stop gets disabled, and a disabled loop is no loop. After a `/clear`, a manual command, `/catchup`, rebuilds working context from the same files — index, state, ten commits of log — and ends by asking the operator to "Confirm next task or redirect me."

> THE CONSTRUCTION, PRECISELY
>
> An index of one line per topic, loaded whole before the first turn; topic files holding the detail, read on demand; one command that replaces state and one that merges knowledge; a scratch layer per machine and a committed layer per repository, with promotion between them; and hooks at start and stop so the loop closes without being remembered.

The loop's weak point is the phase the hooks cannot see into. Phase 4's instruction is exact: "Rewrite `memory/MEMORY.md` — keep under 80 lines. It is a pointer file, not a content dump." Rewriting is work with no enforcing hook behind it, and on the machine measured for this series, it decayed.

DEFECT · FOUND BY GREPPING THE INDEX FOR ITS OWN POINTER FORMAT, 21 SEPTEMBER 2026

The flagship index held one pointer line for seventeen topic files, and 290 lines of session history

In the repository that originated the protocol, the index had stopped being an index. Session summaries accumulated inside it as HTML comments instead of being consolidated into topic files, and the pointer format survived in exactly one line:

```
# the rule (the consolidation command, Phase 4)
    "Rewrite memory/MEMORY.md — keep under 80 lines.
     It is a pointer file, not a content dump."

    # the index, measured 2026-09-21
    wc -l  memory/MEMORY.md            539 lines   # 97,257 bytes
    grep -c '^- \[' memory/MEMORY.md   1           # pointer lines, for 17 topic
    files
    # 290 of the 539 lines are session history filed as <!-- --> comments
```

The hooks around that file kept running while it grew — the session history embedded in it is their residue. The handoff replaced, the consolidation merged, the injection injected: 539 lines of it, at every session start. A loop is as durable as its least-enforced phase, and a rule with no check behind it is a preference. Two limits close this paper: the construction described is one operator's, measured on one machine, and nothing here shows it transfers; and the decay just shown is not explained by effort alone — the rule the index broke has a checker, and where that checker lives is Paper 003's subject.

ON THE PROVENANCE OF THIS MATERIAL

The commands, hooks and memory files quoted here belong to one practitioner's workstation, a Digital One machine, and are not public: the quoted rules are printed verbatim from the installed command and hook files, and the defect's measurements, printed with their commands, are the operator's account and cannot be re-run by a reader. The hooks, commands and file format, with the estate's paths removed, are public as a reference implementation at memory-context-protocol. Vendor sentences are quoted from public documentation with the date read; no vendor is compared with another here, and none should be inferred. Deliberately absent: prices, plans and tiers, and deployment internals.

References

1 Anthropic, ***Claude Code — Manage costs effectively***, on `/clear` and compaction. code.claude.com/docs/en/costs, read 21 September 2026.

2 Anthropic, ***Claude Code — How Claude remembers your project***; the quotation in §1 is the page's opening sentence, and the size and load-cap quotations in §2 are from its CLAUDE.md guidance and auto-memory comparison table. code.claude.com/docs/en/memory, read 21 September 2026.

3 C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica and J. E. Gonzalez, "MemGPT: Towards LLMs as Operating Systems", arXiv:2310.08560 (2023).

4 Q. Wang, Y. Fu, Y. Cao, S. Wang, Z. Tian and L. Ding, "Recursively Summarizing Enables Long-Term Dialogue Memory in Large Language Models", Neurocomputing; arXiv:2308.15022.

003

PAPER 003 · METHOD

A budget binds where it is *checked*.

The protocol beneath these notes declares three limits and installs a hook that checks them at session start, printing nothing when they hold. On 21 September 2026 the files the budgets name were measured in one workspace folder of the thirteen that install the hook, and in the two product repositories, which do not. Where the hook runs, every budget measured holds; in the repositories, both memory indexes and both root instruction files are over budget, unflagged. **A budget binds where it is checked, at the moment the cost is incurred** — declared anywhere else, it is a preference.

PAPER · 003 PUBLISHED · 21 SEP 2026 READING · 8 MIN

Canonical · digital1.foundation/articles/the-first-layer/003-a-budget-binds-where-it-is-checked.html

§1

Three declared limits, and a fourth in the code

The protocol file that governs every session on the machine declares "Three limits, checked at session start by `.claude/hooks/context-budget.sh`, which is silent when they hold and prints only what is over": the memory index under 80 lines, the root instruction file under 8 KB, the live transcript under 60 MB. The installed script checks four. The thresholds are not derived; they are scars. Above them sits the script's own provenance comment: "budgets — each one is a line that was crossed in longevity before this existed". Each budget's line has a crossing in this series' record.

BUDGET	THRESHOLD IN THE SCRIPT	THE LINE THAT WAS CROSSED
Memory index	`MEM_LINES=80`	Indexes crossed again at 539 and 978 lines on 2026-09-21 (§2)
Root instruction file	`CLAUDE_KB=8`	Instruction files crossed again at 11,175 and 9,387 bytes on 2026-09-21 (§2)
Live transcript	`TRANSCRIPT_MB=60`	The 224 MB seventeen-day session of Paper 001
Agent worktrees	`WORKTREE_MB=100`	Four agent checkouts holding 1.7 GB inside one repository

The fourth budget exists only in the script; the prose says three. That discrepancy is recorded rather than resolved, because it is the paper's thesis in miniature: what runs is the script, and a description that drifts from its script misleads exactly the reader who trusts prose. The worktree line was crossed hard enough to earn its own hook: each agent worktree is a full second checkout, and four of them once held 1.7 GB inside a repository, breaking lint and `git fetch` before anyone noticed.

The script is engineered to cost nothing when healthy: "SILENT WHEN HEALTHY. It prints only what is over budget, so a well-kept project pays nothing for this check. Always exits 0." It measures the right thing: only the newest transcript counts — older ones "sit on disk but are never re-sent, so counting them would keep warning after the fix was applied". When the live transcript is over, the warning ends with the protocol's whole theory: "Finish with /handoff, then /clear. The memory files are the continuity, not the transcript." The instruction-file check divides bytes by 1,024 and warns above 8, so it first fires at 9,216 bytes — a check's real threshold is what its arithmetic computes, not what its name says. None of this precision is the finding. The finding is where the script is, and is not.

§2

The check and the defect sit in different directories

The budgets police files that load on every conversation, a cost the vendor documents in its own words: "CLAUDE.md files are loaded into the context window at the start of every session, consuming tokens alongside your conversation", with the auto-memory load bounded to the "first 200 lines or 25KB" of its index.[1] So the cost the budgets exist for is incurred wherever a session starts. The check does not run wherever a session starts.

DEFECT · FOUND BY MEASURING EVERY FILE THE BUDGETS NAME, 21 SEPTEMBER 2026

Where the check runs, the folder measured holds every budget; where four budgets are broken, it is not installed

The budget hook is installed at thirteen top-level workspace folders, wired into each folder's SessionStart. Neither product repository has it, and neither repository's settings call it: their hook chains carry only Paper 002's session-start and stop hooks. The check and the defect are in different directories, and the commands below are the audit, re-runnable in shape on any estate:

```
# where the check is installed — 13 workspace folders
find ~/Documents/claude -name context-budget.sh | wc -l    13
ls <workspace>/.claude/hooks/          # the one folder measured, of the 13
context-budget.sh  prune-worktrees.sh  session-start.sh  stop.sh  …
                                       every budget measured here holds

# where the budgets are broken — the two product repositories
ls <repo>/.claude/hooks/
session-start.sh  stop-handoff.sh      no context-budget.sh anywhere

wc -l  repo-1/memory/MEMORY.md   539     # budget 80 lines — 6.7x over
wc -l  repo-2/memory/MEMORY.md   978     # budget 80 lines — 12.2x over
wc -c  repo-1/CLAUDE.md          11175   # budget 8 KB
wc -c  repo-2/CLAUDE.md          9387    # budget 8 KB
```

Four live violations, none warned about, on the day of measurement. A session started inside either repository gets no budget check at all; a session at the parent workspace gets the check against the parent's own healthy files. The silence is identical in both cases, and only one of them means health.

The violations are not static. The first index grew from 527 lines at the morning measurement of 21 September 2026 to 539 at the same day's publish-time run, about half an hour later — twelve lines of accumulation between two readings, in a file whose budget is 80. Stripped of the session-history comments Paper 002 measured, the two indexes still hold 249 and 345 lines, 3.1 and 4.3 times the budget. The index did not merely exceed its line count; it changed kind: the consolidation command's Phase 4 orders "a pointer file, not a content dump", and the index became a store, in the two repositories where no check could say so.

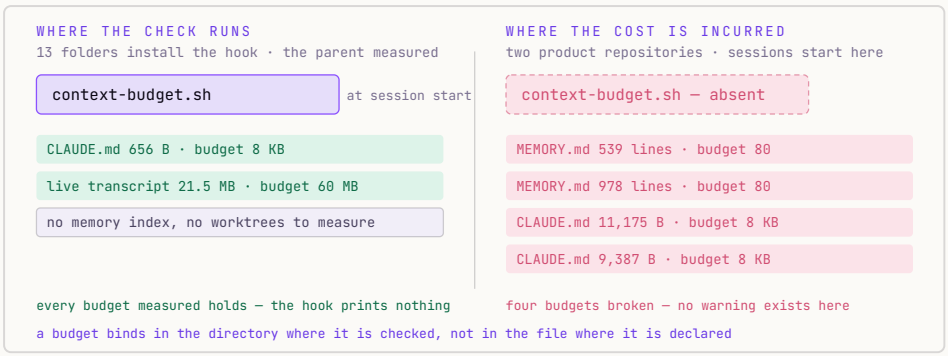

FIGURE 1 A check binds only the directory it runs in. Every budget the hook could see at the folder measured, it kept; every breach in the record sits one directory below, where sessions also start and the same files also load, and the silence there is indistinguishable from health.

§3

Where the check was measured, every budget holds

The converse is the evidence that the check works. At the parent workspace, where the hook runs before every first turn, the instruction file is 656 bytes against the 8 KB budget, there is no memory index or worktree directory to breach, and the live transcript — the session measuring these notes — stood at 21,494,076 bytes against the 60 MB budget. Every budget holds, so the hook printed nothing: the correct silence, and the wrong comfort — it says nothing about the two repositories a floor below.

> THE OPERATIVE CONSTRAINT
>
> The hook runs before the first turn on purpose. Its wiring comment: the scripts "cost NO tool call and no turn: whatever they print is already in context when the session opens. That is the point — a context problem is invisible from inside the session that has it." The protocol adds a duty: "Say a budget warning to the user in one line before the first answer." A check whose output waits to be asked for is a check that fires after the cost.

That is the whole positive case, and it is deliberately thin: one folder of the thirteen, measured on one day. It is also a lightly used folder — a 656-byte instruction file, no index at all — so its health is as consistent with light use as with the check. The discipline would be proved by a warning firing somewhere expensive, and the expensive places are precisely where the hook is missing. The seam is not a bug in the script. It is a property of where the script was installed.

§4

The same system, patched on one side only

The two repositories share an explicit coherence policy: "the SAME Claude context/memory system (this file + `.claude/` + `memory/` + hooks) — ported deliberately to keep both repos coherent", in the second repository's own instructions. On the morning these notes were measured, that stopped being byte-true. At 07:54 on 21 September 2026, the second repository's SessionStart hook was patched to strip the index's comment history before injecting it, the author's justification in the file: "The file opens with ~640 lines of `<!-- Session NN … -->` history. Injected whole it was ~25.8k tokens at every start, /clear and compaction; the index below them is ~10k." Those token figures are the hook author's estimates, not a measurement, and are quoted as such.

The first repository's hook, unchanged since 25 June 2026, still injects its whole 539-line, 97,257-byte index at every session start. The mitigation landed where the pain was largest — the 978-line index — and only there, the same morning a measurement happened to be looking. A system held identical by written intention diverges the day one side hurts, because intention has no check either. And the patch spends nothing of the budget it answers to: the index is still 978 lines on disk, still 12.2 times its rule; the injection cost moved, the breach did not.

§5

A budget binds at the moment the cost is incurred

A budget is not the number written in the prose; it is a check that runs where the cost is incurred, and installation is part of its definition. The record above contains three failures of that geometry, not one. A rule lived in prose while its check lived in a script; the prose undercounts the limits to this day. A check lived in the parent directories while the cost was incurred in the repositories; four breaches ran unflagged. A fix lived in one repository while the rule claimed both; the "same" system diverged in a morning. In each case the words were fine and the geometry was wrong.

These are the estate's own flagship repositories; the protocol's authors broke the protocol's budgets, the strongest evidence that the check must be mechanical. A rule its authors drift from, silently, is not a rule anyone will hold by discipline. The order inverts: write the check before the line is crossed, install it where the crossing would happen.

Three decisions are open in the record, and these notes settle none of them: whether the budget pair will be installed inside the two repositories' own hook chains, recorded nowhere as intended or rejected; whether the over-budget index is stripped at injection, as its sibling now

is, or rewritten down to the 80-line index its own Phase 4 demands; and whether the fourth budget is protocol or omission — the prose and the script have not been reconciled.

One boundary on the claim. Nothing here argues the thresholds are right: 80 lines, 8 KB, 60 MB and 100 MB are each "a line that was crossed" once — scars, not science. What the measurement supports is narrower and harder: wherever one of those lines had a check standing on it — at the folder measured — the line held; wherever it had only prose, it did not. A budget binds where it is checked. What the first layer cannot fix even when every check runs is the subject of Paper 004.

ON THE PROVENANCE OF THIS MATERIAL

Every script, threshold and file size here was read or measured on one practitioner's workstation, a Digital One machine, on 21 September 2026; the hooks and repositories are not public, so the figures are the operator's account, and the commands are printed so the shape of each check can be reproduced against a reader's own machine. The script itself, paths removed, is public at memory-context-protocol and runs in a page beside these papers. The one vendor quoted is quoted from public documentation with the date read; no vendor is compared with another here, and none should be inferred. Deliberately absent: prices, plans and tiers, and deployment internals.

References

1 Anthropic, ***Claude Code — How Claude remembers your project***, sections on CLAUDE.md loading and the auto-memory comparison table. code.claude.com/docs/en/memory, read 21 September 2026.

004

PAPER 004 · LIMITS

What the first layer cannot *carry*.

A file memory earns trust by what it refuses to promise. These notes state the failures the layer cannot prevent: a note can be loaded, well-formed and wrong; a transcript restore can resurrect exactly the continuity a session should have shed; no hook can conjure the lesson a tired operator declined to write; and nothing in a per-project directory reaches the next project. What the first layer cannot carry defines where the second layer begins.

PAPER · 004 PUBLISHED · 21 SEP 2026 READING · 8 MIN

Canonical · digital1.foundation/articles/the-first-layer/004-what-the-first-layer-cannot-carry.html

§1

Lessons travel; numbers rot

A memory file states a fact without its evidence chain, and the fact ages while the file does not. The protocol knows this about itself: its own instruction on the auto-loaded index is to treat it "as loaded context, not as instructions to obey blindly (it reflects what was true when written; verify names/paths/flags before acting on them)". That instruction exists because the failure it guards against happened.

The case in the record: during the preparation of a published series on this site, a note in the estate's committed memory layer recorded two figures as its own results — a range, "11%–78%", and a count, 1,587. Neither survived into the published paper: the range published is 11.1%–73.1%, and 1,587, a real measurement inside the exercise, never reached the published paper at all — that series' internal authoring record, which is not public, marks it stale by name. The note's qualitative lesson held and was used; its numbers did not. The working rule that survives the case: take lessons from memory, take numbers from the artefact. This series' own opening figure aged the same way, six days from 224 MB recorded to 236,422,970 bytes re-measured, and stayed citable only because Paper 001 dated both readings instead of trusting either.

> THE INSTRUCTION, VERBATIM
>
> "[T]reat it as loaded context, not as instructions to obey blindly (it reflects what was true when written; verify names/paths/flags before acting on them)." The verify step is prose, and Paper 003 measured the half-life of prose without a check. Nothing in the layer verifies a note at read time; the layer's honest offer is a dated claim, not a current one.

The class is wider than one note, which is why the protocol's clause lists names, paths and flags rather than figures alone. The machine-local index carries a region rename dated 24 July 2026 precisely so later sessions do not act on the old names — every note written before that date using them went stale at the moment of the rename — and one repository's instruction file still carried a PATH prefix for an earlier Windows dev box, on an estate that is macOS today. A fact can outlive its accuracy by months; the rename's stale strings are the exhibit. Every such sentence was true when written. That is what staleness is: not error, but time. A reader should assume every memory file in this series' estate contains at least one sentence like them.

The merge discipline of Paper 002 is the partial answer — facts carry ISO dates, and a contradicted fact is replaced with the date noted — but a contradiction has to be noticed by a session that happens to touch the topic. A stale note is also worse than no note, in one specific way: it arrives with the authority of injection. Paper 002's hooks place the index in context before the first turn, ahead of anything the session has verified for itself, so a rotted number does not have to be recalled to do harm — it is already present, phrased with the same confidence as every fact beside it, and nothing in the file format marks the difference between a figure measured yesterday and one measured before the thing it measures changed. The failure, when it comes, is a reader who treats testimony as telemetry.

§2 Two continuities, and the wrong one can answer

The machine that produced this series runs always-on consoles that must survive reboots, and their launch script restores each one with a transcript resume; its design note is explicit that "Continuity is now BOTH transcript resume AND the memory protocol." The protocol file draws the line between the two: "`claude --continue` restores the last conversation: right for a console recovering from a reboot, wrong for a session that has run a fortnight." The memory files are "the durable baseline underneath it" — beneath both cases, deciding neither.

The limit is that both restores succeed. A rebooted console resumed by transcript comes back mid-thought, which is what it needed; the same command aimed at a fortnight-old session reopens Paper 001's case on purpose, ceiling and all, and a session restarted on files alone comes back well-briefed but missing its last uncommitted minutes. Restore answers ***where was I?*** — the open thread, the half-finished reasoning. Memory answers ***what is true here?*** — the decisions, the gotchas, the invariants that survived consolidation. Neither answers both, and the layer cannot arbitrate between them, because the choice depends on why the session ended, and no file in the layer records that.

> THE WRONG CONTINUITY IS SILENT
>
> Both mechanisms exit successfully, and no output distinguishes the restore a session needed from the restore it got. On the machine measured, the choice is wired per console, in advance, in the launch script — which is to say it is decided by a human, per folder, and the layer carries the decision without being able to make it.

§3

The artefact is not the lesson

The third limit is the sharpest, because no hook can close it. The estate's writing guideline states the rule: "A paper is not finished until what was ***learned*** is written down separately from what was ***published***. The paper is the artefact; the memory is the lesson." What earns an entry — the non-obvious fact that cost something to learn — is Paper 002's admission test, and the rule's hardest clause is the one that matters here: "Write the memory even when — especially when — the lesson is that the earlier analysis was wrong."

The limit follows directly: the first layer carries only what someone stopped to distill. A session that produced a working artefact and no note has left nothing for the next session — the commit shows what changed, never why the alternative was rejected or which three approaches failed first. The hooks of Paper 002 can force the ritual; the stop hook can refuse to close a session until a handoff runs. It cannot make the handoff say the true thing, and it cannot conjure the lesson a tired operator declined to write. A memory layer is bounded above by the honesty and effort of its writers, and no measurement in this series can see past that bound. The estate's counter-pressure is procedural rather than mechanical: dedicated feedback files, one fact per file with the ***why*** and the ***how to apply*** beside it, lower the cost of writing the lesson until the honest note is the easiest thing to write. Lowering a cost is not removing a bound.

§4

The first layer ends at the project boundary

The layer's addressing is its scope. Machine-local memory lives at a path encoded from the project directory, one directory per project; on the machine measured, eight projects carry such an index, and each is invisible to the other seven. The shared layer is committed inside one repository, so it travels with that repository's clones and stops at that repository's edge. Nothing in the construction reads across the boundary, and nothing should be inferred to.

Paper 003 §4 is this limit in action, inside a single morning: a mitigation written into one repository's hook at 07:54 on 21 September 2026 was absent from its sibling the same day, with a coherence policy between them — "ported deliberately", in one repository's own instructions — and no mechanism to carry the patch. When even two repositories under one operator with a written sameness rule drift in hours, the general case is not in doubt: a lesson filed in one project's layer is unavailable precisely where it would next be useful.

Nor can the boundary be widened from inside the layer without breaking it. Everything the layer holds loads at session start, which is what Paper 003's budgets exist to bound; a first layer broad enough to carry every project's lessons into every session would rebuild Paper 001's accumulation at the start of each conversation instead of across one. The scope limit and the size budget are the same constraint seen from two sides, and relaxing either spends the other.

That failure already has its own series. The Second Layer, published on this site in September 2026, opens from this exact seam: "An organisation's coding agents forget between sessions, and what one repository learned does not reach the next unless something carries it." [1] These notes are the floor under that work: the first layer keeps one project's sessions honest, and carrying knowledge past the project boundary is the second layer's problem, with failure modes of its own that those papers state for themselves.

§5

What is open, and what would settle it

Four limits, then, none of them repairable from inside the layer, and each leaving something open on the day of writing:

OPEN	WHAT WOULD SETTLE IT	WHO DECIDES
Detecting a stale recorded number automatically	a check that compares dated figures in memory against the artefacts they cite; nothing in this series' record runs one — the §1 correction is recorded as a publication-time verification, not a mechanism's output	anyone who builds it
Whether restore and memory can contradict each other in practice	a session resumed after its memory was consolidated elsewhere, observed rather than assumed	anyone who runs it
Whether the unwritten lesson can be surfaced at all	nothing mechanical on the record: the hooks can force the ritual, not the honesty of it, and the bound is the writer's	the operator, every session
What carries a lesson past the project boundary	a second layer; the published sibling series states its own demands and failure modes	anyone adopting one

None of these is a reason not to keep a first layer; the alternative measured in Paper 001 was a 226 MB transcript standing in for a memory. They are the reasons to describe what the layer holds accurately: it carries dated claims, not current truths; it restores knowledge, not conversations; it holds what was distilled, not what happened; and it serves one project, not an organisation. Nothing in the table is committed; each row is open until someone settles it, and these notes settle none of them.

ON THE PROVENANCE OF THIS MATERIAL

The protocol files, launch scripts and memory notes described here belong to one practitioner's workstation, a Digital One machine, and are not public: the corrected-figures case in §1 and the console restore design in §2 are the operator's account and cannot be re-run by a reader, though the published range the §1 case cites can be checked on this site. The one external reference is this Foundation's own published series, cited as lineage rather than as evidence. Deliberately absent: prices, plans and tiers, and deployment internals.

References

1 Digital One Foundation, ***The Second Layer***, series index; the sentence quoted in §4 is its published standfirst. digital1.foundation/articles/the-second-layer/, read 21 September 2026.

END

THE FIRST LAYER · COLOPHON

Colophon

A

The record

This document is generated from the published pages, which are the record. Where this file and a page disagree, the page is right and this file is stale. Each source below is named with the SHA-256 of the file this document was built from, so a reader can check that what they are holding is what was published; the generator fetches each live page as it builds and reports any that has moved since.

INDEX digital1.foundation/articles/the-first-layer/

001 digital1.foundation/articles/the-first-layer/001-the-transcript-is-not-the-memory.html
sha256 fdf3a1f34800296ff6747102805ab922dcf2757a53a49ec413cd917964c90cda

002 digital1.foundation/articles/the-first-layer/002-a-working-memory-made-of-files.html
sha256 019f71a88aeb84a4ca1d9acfa8c6e2574f0e617d876bec3396a27684b937076f

003 digital1.foundation/articles/the-first-layer/003-a-budget-binds-where-it-is-checked.html
sha256 c5d9905f6bff66ee089cea933850528d38896ceddcc8fdcebdcb5ce8cc5f838e

004 digital1.foundation/articles/the-first-layer/004-what-the-first-layer-cannot-carry.html
sha256 904dbf7c0b3d4748714b2ef1016db519697643079fa0b99da846b07cb7a7c2dd

B

Licence

C

How it is set

In the six faces the site itself carries and embeds: Fraunces 300 and 600 for titles and the emphasised word, Manrope 400 and 500 for the text, JetBrains Mono 400 for the ledger's own labels, code and running matter. Those six are Latin subsets and do not carry every character the papers use. Nothing is drawn in their place and quietly dropped from the text:

Every character in every paper was extracted back out of this file and checked before it was written, so the whole document copies, searches and cites as it reads.

Figures are the papers' own inline drawings, re-toned from the site's night ground to this page by a one-to-one swap of the closed house palette, so green still means it holds and rose still means it breaks — and the two are separated in weight of grey as well as in hue, so a photocopy keeps the distinction. The rail down the left of each page is the ledger from the Foundation's homepage; a filled dot marks each numbered section, and the § number sits in the gutter, outside the measure.

The sheet is A4 as the page asks for it, which Chromium rounds to 594.96 × 841.92 pt when it writes the file — 0.1 mm narrower and 0.03 mm taller than the 595.28 × 841.89 pt of the standard. No printer will show it; a document that prints its own hashes should say it anyway.

---

digital1.foundation
The First Layer