

L3M

A Lean4 runtime for multi-agent, multi-human computation

Carson Eisenach

Collaborators: Robert Joseph George, Vincent Quenneville-Bélair, Kate Gwimm, Dean Foster

COLM 2026

The Bitter Lesson

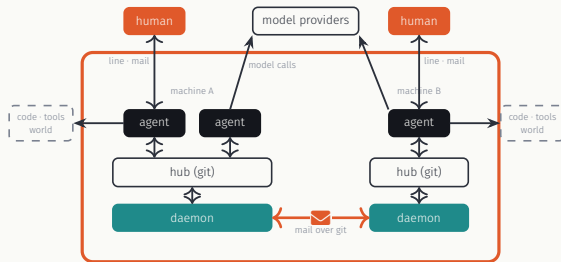
The wins in AI come from methods that can absorb more computation.

Models already let us scale compute *within* a call.

But useful work happens **across**:

- many calls, over hours or days
- tools and external systems
- multiple agents, and humans
- code the agents themselves create

What is L3M?



L3M: the distributed runtime

I. models + code: how future systems will compute

L3M is a Lean 4 runtime where one computation can span humans, agents, models, and code.

L3M carries the computation: all state is encapsulated in the hub, a git repository; a daemon keeps it moving and coordinates across machines.

communicate

wait

fork

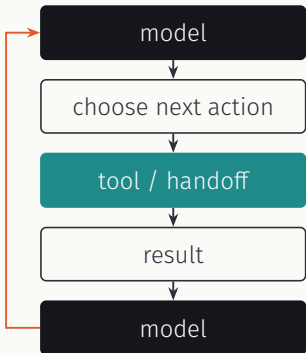
act

undo

II. the runtime that future needs, as built in L3M

Runtime: Lean 4, over a small named native/OS floor for files, locks, networking, and terminal I/O.

What should live outside the model loop?



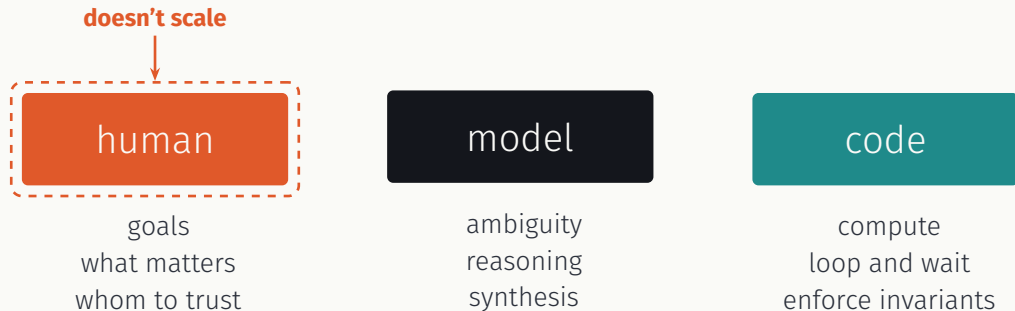
A great primitive.

- modern harnesses already add persistent sessions, handoffs, guardrails, tracing, human approval, and resumable sandboxes around it

But should everything route back through the model?

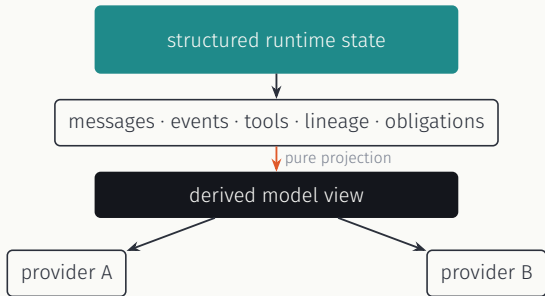
- waiting · loops · coordination
- recovery · delegation · state

One computation, three participants



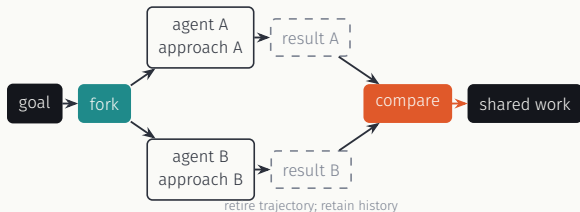
Intelligence compounds when one call's reasoning changes what later calls can compute with: state, branches, agents, programs, invariants.

The conversation is a view, not the state



**The object being manipulated
is no longer a chat transcript.
It is a structured computation.**

Explicit state gives the computation a geometry



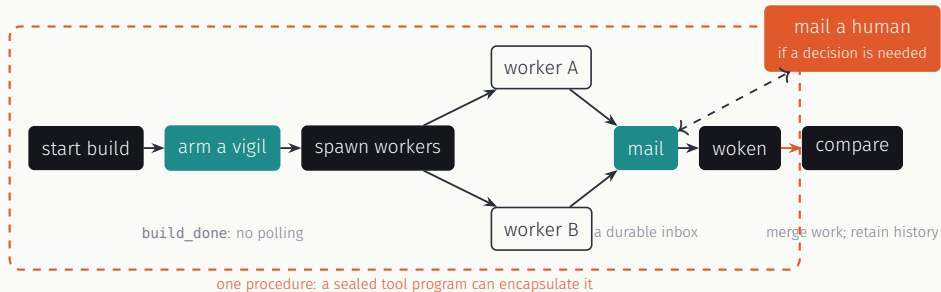
Once computation is explicit state, trajectories become first-class objects.

navigate · resume · fork
compare · retire · merge work

⚠ mechanism

Branching is literal: agent work lives on Git branches and merges as branches, and what agents learned can be distilled into a shared persona.

A computation, in L3M's own primitives



**These are not prompt patterns.
They are runtime primitives.**

AGENT		AGE		WHERE
• roots • 16				
carson/				
⌚	l3m	22s	here	@14
●	agents	1h	here	@26
●	caps	3h	here	@15
●	docs	34m	headless	
●	integrator	1h	here	@17

📷 live roster: a parent and its children

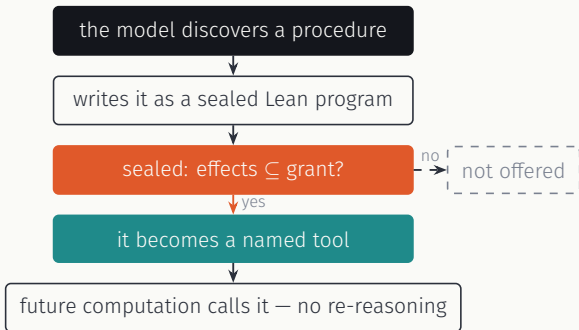
The event loop



The model decides what computation should exist. The runtime carries it through time.

Events arrive in any order; the model always sees a well-formed computation.

Intelligence can become machinery



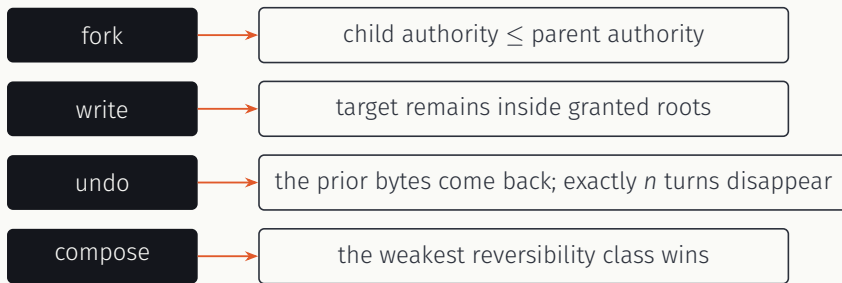
Capability grows; authority does not.

How to do the job moves out of the prompt and into an artifact checked once and reused.

⚠ theorem + runtime check

`sealUsed_admits_only_safe` ● (`src/L3m/Sealed/SealTheorems.lean`): the seal admits only permitted constants. A program's body is closed over its declared footprint and is offered iff the grant covers it (`Program_<name>`, loadable with `Program_reload`).

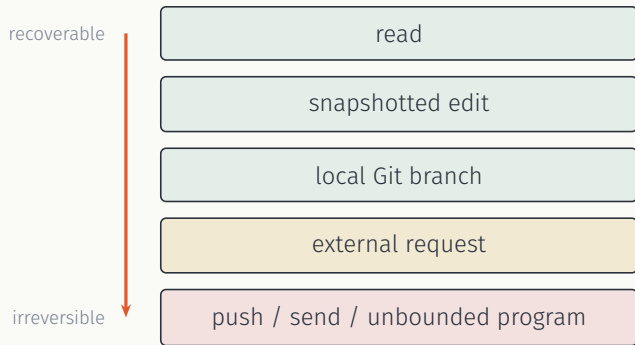
A runtime that can change needs properties that survive change



Formal methods let us preserve invariants as the harness changes.

They also let a human reason about what agents are doing without knowing, or needing to know, every detail.

Tameness: reversibility buys autonomy



**Let agents act freely
where the runtime can
recover.**

Spend human attention where
the world cannot be put back.

Tame: an action the runtime can
undo, or whose reach is bounded
in advance. Tame actions run
without review.

Human can come out of the loop

Why implement the runtime in Lean 4?

implementation

agent loop
state + events
capabilities + tools

specification

confinement
authority
reversibility

extension

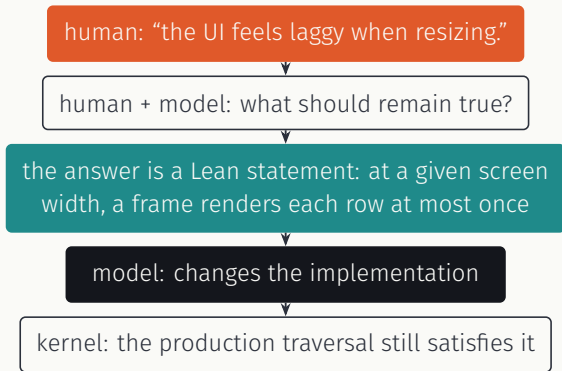
agent-written functions
sealed programs
same typed environment

The same language implements the runtime, states its invariants, and is available to extend it.

boundary

The Lean kernel checks the stated Lean properties. Closed facts may rely on `native_decide`; host behaviour rests on named assumptions; a small C/native floor provides host primitives. This is not end-to-end verified security.

Lean is how humans and agents agree on software

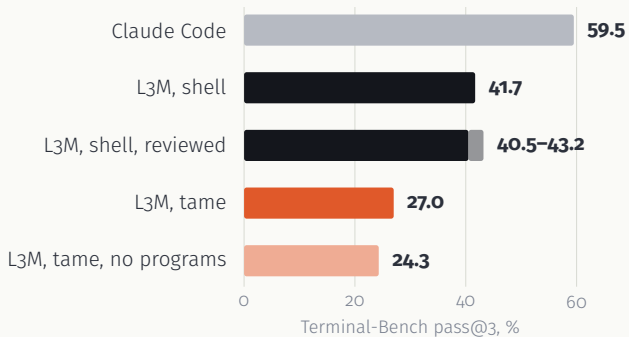


The human states what must remain true. The agent changes the code. The kernel arbitrates.

Neither has to trust the other's reading of the code.

A finding from building L3M: this way of reasoning about software is powerful well beyond security.

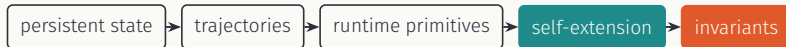
Early results: how much work is tame?



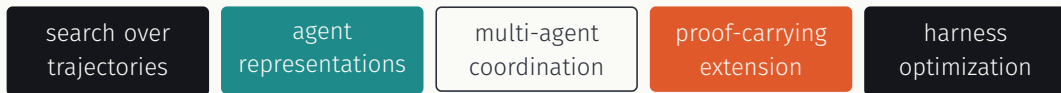
- Tame alone reaches about two thirds of the shell.
- Review costs model work, not success.
- Tool programs add about 3 points.

Caveat: first implementation; harness and system prompt untuned. 37 tasks, three attempts each; every arm runs the same model. With a shell, L3M still trails Claude Code.

What does L3M unlock?



Future research directions



Each run leaves something the next one can build on.

L3M

Harnesses determine whether intelligence can compound across calls.

Explicit state makes trajectories first-class.

Lean keeps invariants that survive change.

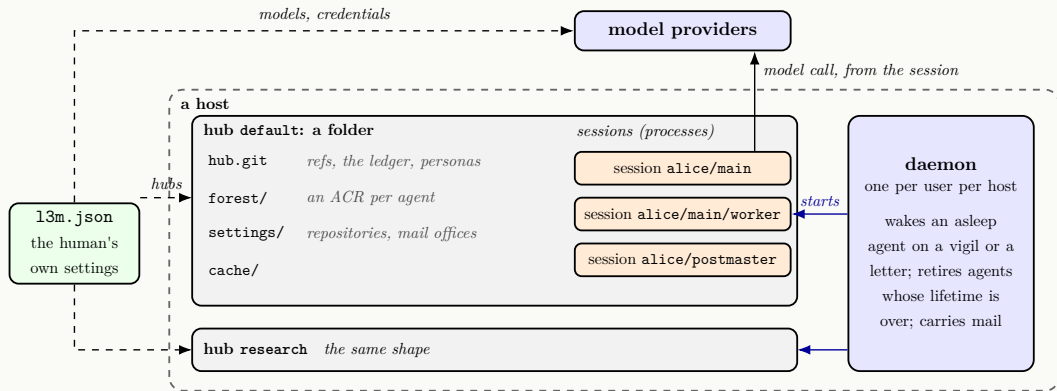
Capability grows; authority does not.

We plan to release L3M as open source soon!

L3M Paper

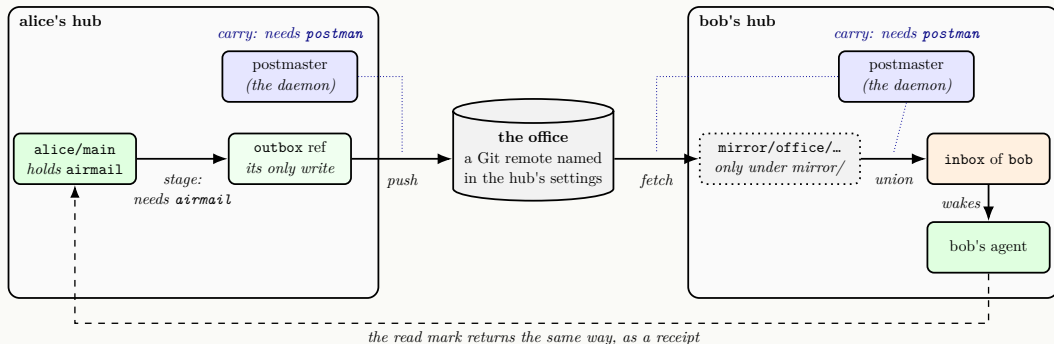


Appendix: L3M on one host



L3M on one host. One daemon serves every hub on the host, and a hub is a folder: a bare repository, a forest of ACRs, its settings and a cache. Sessions are processes that the daemon starts and that call the model themselves. The human's own settings name the hubs, the models and the credentials.

Appendix: mail between hubs



Mail between hubs. The sender's authority ends at its own outbox; carrying mail across machines is the postmaster's duty, and only the human's settings name the offices. Read marks travel back the same way, as receipts.

Appendix: mail is runtime state

```
• a message in carson/l3m/pm's inbox: 📧 mail from carson/l3m/agents: migrate-11to12-gate-drops-uri:
offered 854ca45c2 <carson_l3m_agents-179123>
Agents has offered the defect fix. Let me read it:
• Mail_read(carson/l3m/pm)
  | [Check-in: open items summary] carson_l3m_ux-1791090980-0cd06537 from carson/l3m/ux ts
  | 1791090980 unread ↪ carson_l3m_pm-1791090951-0ccff303
  | [migrate-11to12-gate-drops-uri: offered 854ca45c2] carson_l3m_agents-1791232660-154242dd from
  | carson/l3m/agents ts 1791232660 unread
```

📷 a live session: the runtime wakes the PM agent on mail from another agent; it reads its durable inbox

agents → pm

one agent offers a fix by mail

runtime notice

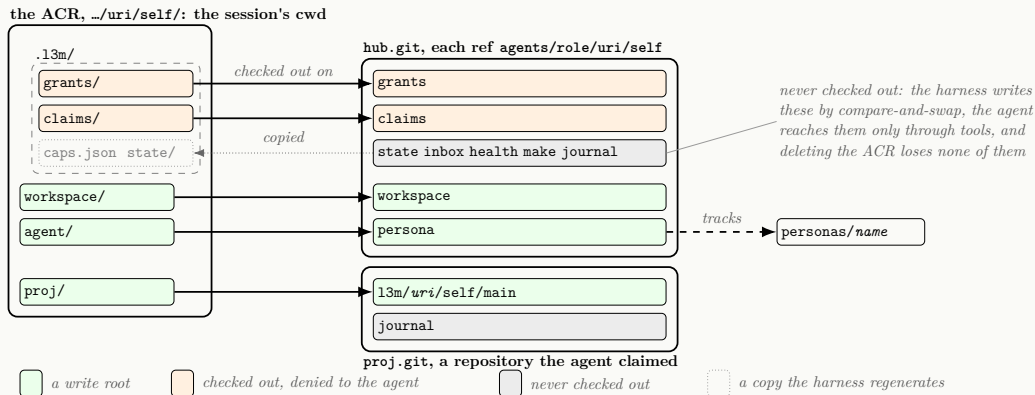
the recipient is woken, not
polling

durable inbox

ids, senders, threads, read
state

✓ inbox_caps_restrict_refs ● · notice_is_runtime_provenance ● · body_cannot_forge ?

Appendix: the body of an agent



The body of an agent. Its write roots are worktrees on its own branches; the harness's checkouts are denied to it; the rest of the agent is refs the harness writes and the agent reaches only through tools.

Appendix: Log → View → Wire

Log (*JSON lines, abridged*)

```
{ "seq": 0, "kind": "system", "system": { "text": "You are..." } }
{ "seq": 1, "ts": "...T10:02:11.402Z", "kind": "message", "message": {
  { "role": "user", "content": [ { "type": "text", "text": "fix it" } ] } }
{ "seq": 2, "kind": "message", "message": { "role": "assistant",
  "content": [ { "type": "tool_use", "id": "t1",
    "name": "File_read", "input": { ... } },
  "usage": { "input": 5210, "output": 84, "cache_read": 4096, ... } } }
{ "seq": 3, "kind": "machine",
  "machine": { "class": "call_closed", ... } }
{ "seq": 4, "kind": "message", "message": { "role": "user", "content": [
  { "type": "tool_result", "id": "t1", "content": "... } ] } }
{ "seq": 5, "kind": "state_change", "state_change": {
  { "class": "note", "label": "budget", "text": "80%" } }
{ "seq": 6, "kind": "message", "text": "Push it?" }
{ "seq": 7, "kind": "boundary", "boundary": {
  { "class": "rewind", "to_seq": 5 } }
{ "seq": 8, "kind": "message", "message": {
  { "role": "user", "content": [ { "type": "text", "text": "go on" } ] } } }
```

view

View (*provider-neutral*)

```
system: "You are..."
items:
  message user [text "fix it"]
  message assistant
    [tool_use t1 File_read ...]
  message user
    [tool_result t1 "..."]
  delta capabilityNote
    "budget" "80%"
  anchor: next user turn
  message user
    [text "go on"]
```

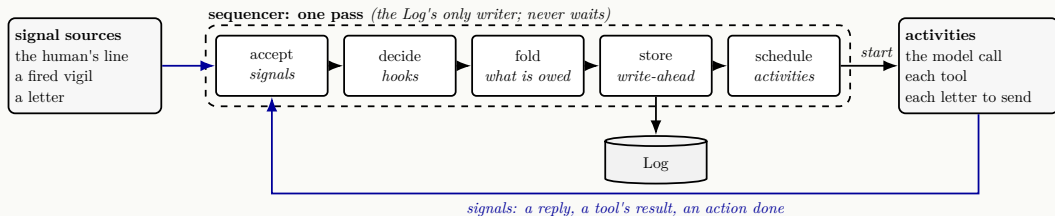
render

Wire (*Bedrock Converse*)

```
{ "system": [ { "text": "You are..." } ],
  "messages": [
    { "role": "user", "content": [ { "text": "fix it" } ] },
    { "role": "assistant", "content": [
      { "toolUse": { "toolUseId": "t1",
        "name": "File_read", ... } } ] },
    { "role": "user", "content": [
      { "toolResult": { "toolUseId": "t1", ... } } ] },
    { "role": "user", "content": [
      { "text": "<state-change>80%</state-change>",
        "text": "go on" } ] } ] },
  "toolConfig": { ... } }
```

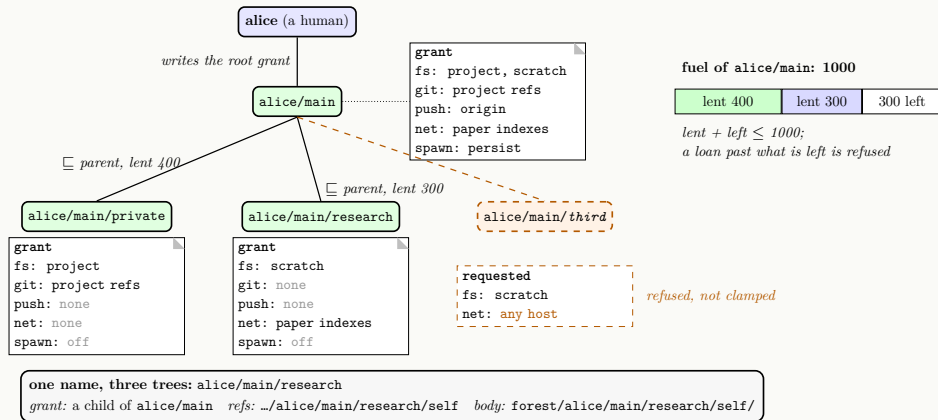
What the model sees is a pure function of the Log. Record 6 is retracted by the rewind at 7; the machine record at 3 lowers to nothing; the budget note at 5 becomes a delta that the dialect folds into the next user turn.

Appendix: the event loop



The event loop. Signal sources and activities only post signals, and the sequencer alone appends to the Log. Each pass accepts what is consistent with the Log, journals the hooks' decisions, folds the Log to decide what is owed, stores it, and only then starts the activities it names, so a kill at any point is repaired by the next pass.

Appendix: a tree of agents



A tree of agents. Each child's grant lies within its parent's, and its fuel is lent from the parent's. A request that exceeds either is refused. The private child keeps the project's refs but not the right to push them.

Appendix: the headline guarantees

Guarantee	Theorem	Rests on
Confinement. A file tool cannot name a path that climbs out of the agent's roots.	<code>safePath_no_traversal</code>	The kernel. The check is on the path's text; symlinks are checked at run time.
Monotone authority. A child's grant never exceeds its parent's, and no agent grants to itself.	<code>grant_restrict_le</code> <code>no_self_grant</code>	The kernel.
No token without the grant. The boot mints a token only for an authority the grant confers, and a narrower grant mints none a wider one lacked.	<code>mint_reflects_profile</code> <code>Minted.restrict_none</code>	The kernel. That tokens cannot be forged is enforced by the build, not proved.
Budget is lent, not minted. All a subtree of agents can spend is bounded by its root's budget.	<code>global_budget_bounded_by_root</code>	The kernel.
Self-extension adds no authority. A sealed program the harness admits uses only admitted constants.	<code>sealUsed_admits_only_safe</code>	The kernel.
Review cannot be skipped. A tool whose effects cannot be undone and are not confined by a capability is always reviewed.	<code>ungated_world_reaching_always_reviewed</code>	The kernel.
Undo. A journaled edit can be restored to the bytes it replaced.	<code>live_capture_round_trips</code>	The kernel, and a named assumption that Git's object store returns the bytes it hashed.

The headline guarantees. Each is a theorem about the code L3M runs and what it rests on.

Appendix: Terminal-Bench, in full

configuration	pass@3, %	per passing attempt		
		model calls	tokens, M	minutes
Claude Code, permissions bypassed	59.5	53.0	4.43	24.6
L3M, shell, review off	41.7	53.2	7.70	25.4
L3M, shell, review - -yes	40.5	100.9	12.10*	30.4
L3M, shell, review - -no	43.2	107.3	12.81*	28.1
L3M, tame	27.0	53.3	7.74	21.6
L3M, tame, no programs	24.3	41.5	6.17	24.5

- Setup: 37 tasks, three attempts each, Claude Opus 5 in every arm, Claude Sonnet 4.6 as reviewer, 75-minute deadline. Means are over passing attempts and include subagent sessions.
- Review covers shell calls only, 25% of all tool calls. About 6% of reviewed calls were redirected to a tame tool, 2% withdrawn, 2.4% stalemates for a human. Each reviewed call adds about 2.6 model calls.
- *Tokens include the reviewer's side, on the cheaper reviewer model.