

The Agent Harness

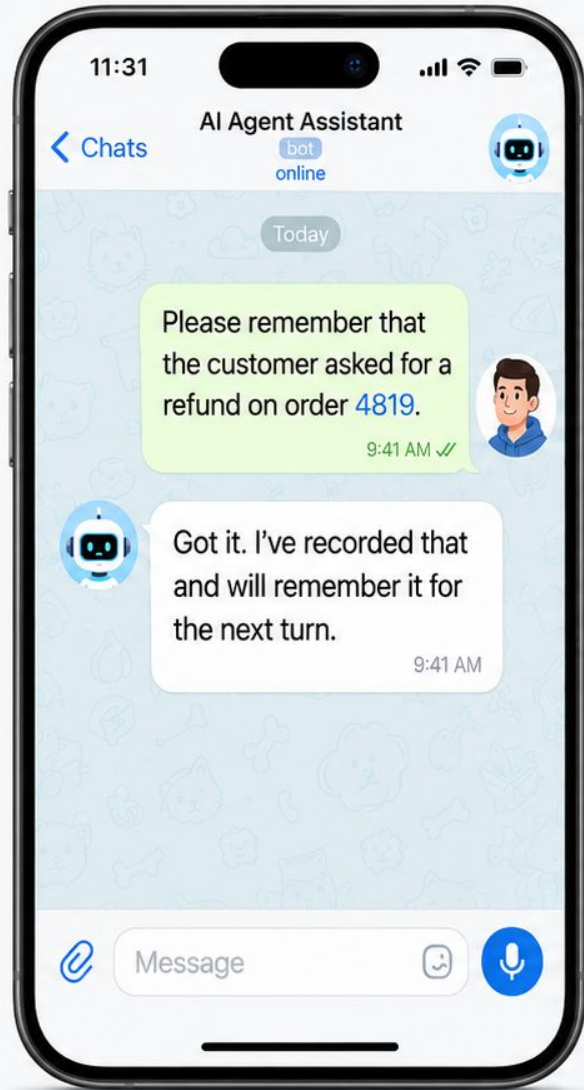
Control Planes, Invariants, and Approval Boundaries
for Production AI Agents

Vinoth Govindarajan - MTS @ OpenAI

<https://www.linkedin.com/in/vinothgovindarajan/>

**The user saw the reply.
The system forgot it happened.**

A production incident in one sentence.



Backend / Durable Store


System of Record



Durable record: missing

No event persisted

Event log (latest)

TIMESTAMP	EVENT	DETAILS
 No events found Future context hole		



The agent replied, but nothing was saved. The system will forget this in the next turn.

Why this is worse than a crash

A crash gives you a boundary. Silent success gives you a lie.

Delivery

The channel said success.

Persistence

The turn was not recorded.

Future context

The next run inherited a hole.

? The production question

When text becomes action, who owns the truth?



State

Who owns the fact?



Order

Who committed first?



Proof

Who can show what happened?

👋 Hi, I am Vinoth

I bring distributed-systems lens to agents infrastructure



- Building core data/AI infra at OpenAI
- Built distributed systems at Apple & Uber
- Built full-stack applications for a decade
- Contributed to many open-source projects
- Writes The Agent Stack on Substack
- Co-author of Engineering Lakehouses with Open Table Formats Book
- Studied OpenClaw architecture and incidents



Three things to remember

Own the state. Order the mutation. Prove the action.

1. Own the state

A fact needs one owner and one replay path.

2. Order the mutation

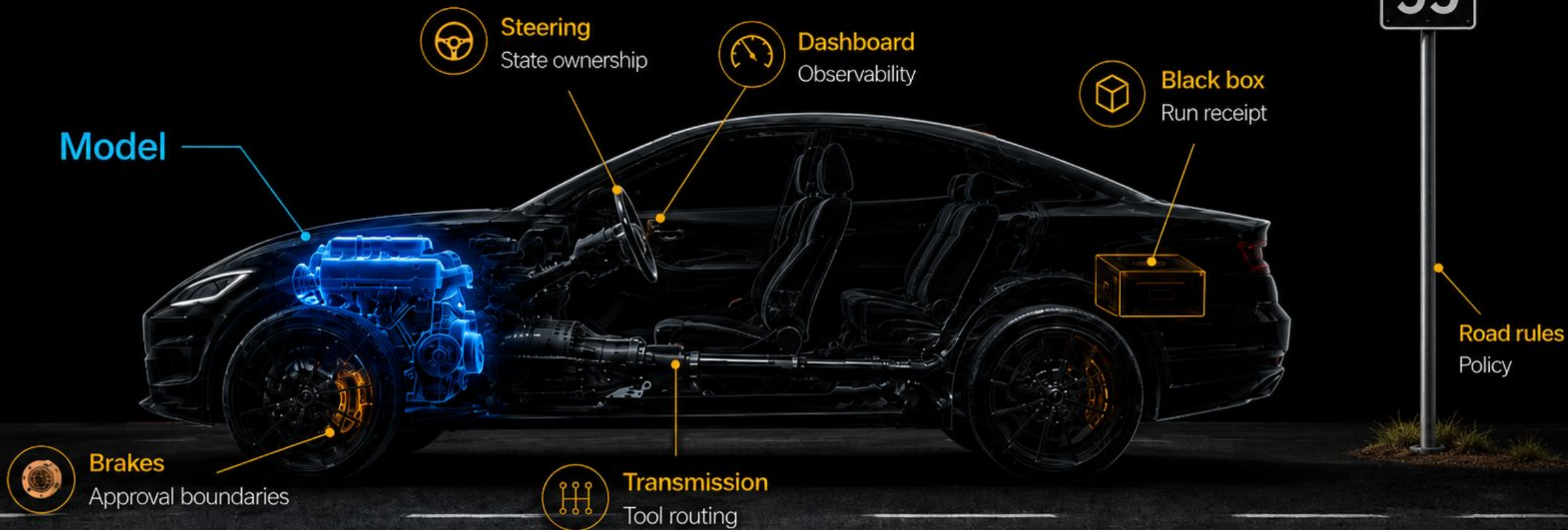
Concurrency is fine.
Accidental interleaving is not.

3. Prove the action

The transcript is not the receipt.

The model is the engine.

The harness is everything that lets the engine move safely in the real world.





The production contract

A model proposes. The harness commits. The receipt proves it.

 **Model**
proposes

 **Harness**
commits

 **Receipt**
proves

 **OpenClaw is the case study.**  **The contract is the
takeaway.**



Why old failures get sharper in agents

Familiar reliability bugs sit around a probabilistic planner.



Dynamic plans

the next step is chosen at runtime



Probabilistic output

the path may differ between turns



More event sources

users, tools, timers, webhooks, subagents



More action surfaces

messages, files, APIs, browsers, workflows

The failures are familiar.

The agent setting makes them easier to trigger and harder to explain.



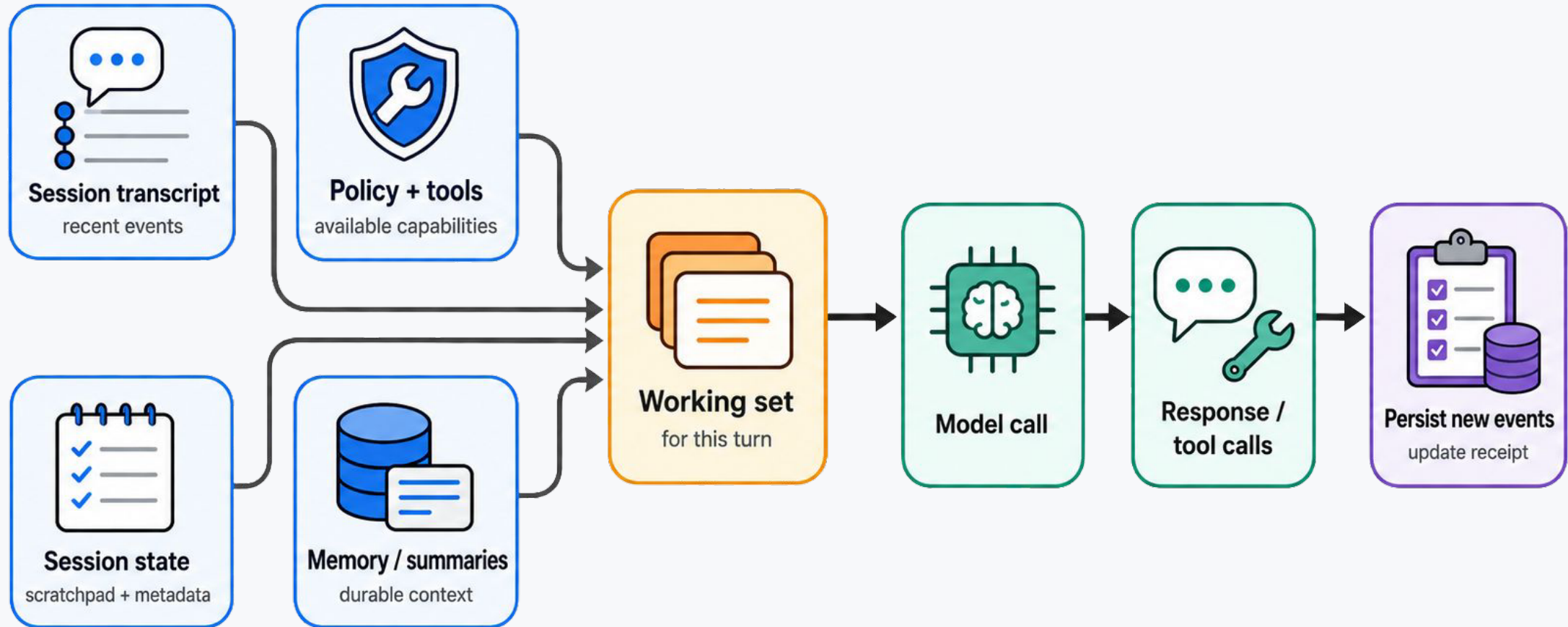
The harness blueprint

Same pattern across frameworks. Different product names.



Turn start: rehydrate state

The runtime does not “remember.” It rebuilds the working set for this turn.



1



Own the state

Every fact needs one owner and one replay path.

🕒 Incident: success without memory

The user saw the reply. The system forgot it happened.

User sees

the assistant replied

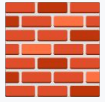
System state

turn was not recorded

Harness lesson

delivered is not remembered

The screenshot shows a GitHub issue page for the repository `openclaw / openclaw`. The issue title is `claude-cli backend skips runContextEngineMaintenance — LCM/transcript persistence broken for all CLI-routed turns #81558`. The issue is marked as `Closed` and has a link to `#81869`. The issue was opened by `rasimme` 5 days ago. The summary text reads: "The `claude-cli` backend code path (`execute.runtime`, `claude-live-session`, `prepare.runtime`) never calls `runContextEngineMaintenance` or `finalizeHarnessContextEngineTurn` after a turn completes. The embedded backend (`pi-embedded`) calls both consistently. As a result, every turn routed through `provider=claude-cli/*` bypasses the entire context-engine / LCM persistence pipeline. LCM stays empty, session JSONL stops being appended, and `historyPrompt=none` is forced in every CLI exec — making missing-transcript resets unrecoverable."



Harness Pattern: state is not storage

State is the ownership boundary for recovery.

Transcript

which log is appended?

Memory pipeline

which process ingests it?

Replay

which owner can reconstruct it?

Owner examples: Event → Calendar system • Support → Ticketing system • Email → Email provider
Code → Repository/workspace • Preference → Memory store • Conversation turn → Session transcript



OpenClaw example: heartbeat became state

A liveness token crossed the wrong boundary.

User sees

agent goes quiet for hours

System state

HEARTBEAT_OK becomes pending delivery

Harness lesson

control signals are not user work

openclaw / openclaw Public

<> Code Issues 3.6k Pull requests 3.5k Actions

New issue

[Bug]: Heartbeat death loop — pendingFinalDelivery stuck on agent main session, blocks all future heartbeats for days #79258

Closed

#80357

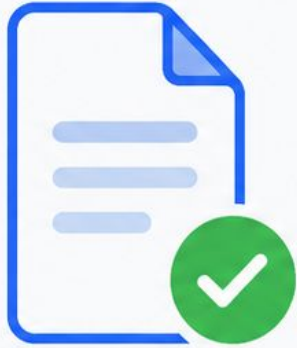
haumanto opened 2 weeks ago

Bug type

Behavior bug (silent state corruption, no crash, no error logs)

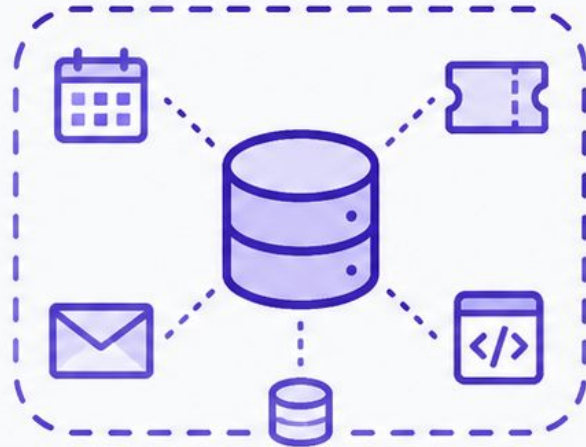
✓ General lesson

A reply is not reliable memory until a named owner can replay it.



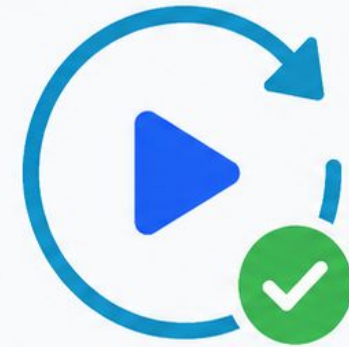
Persist the turn

capture the fact



Name the system of record

where this fact belongs



Make replay possible

prove recovery

2



Order the mutation

If two writers can touch it, the harness owns the order.

Incident: two correct writes

Last writer wins is not a consistency model.

User sees

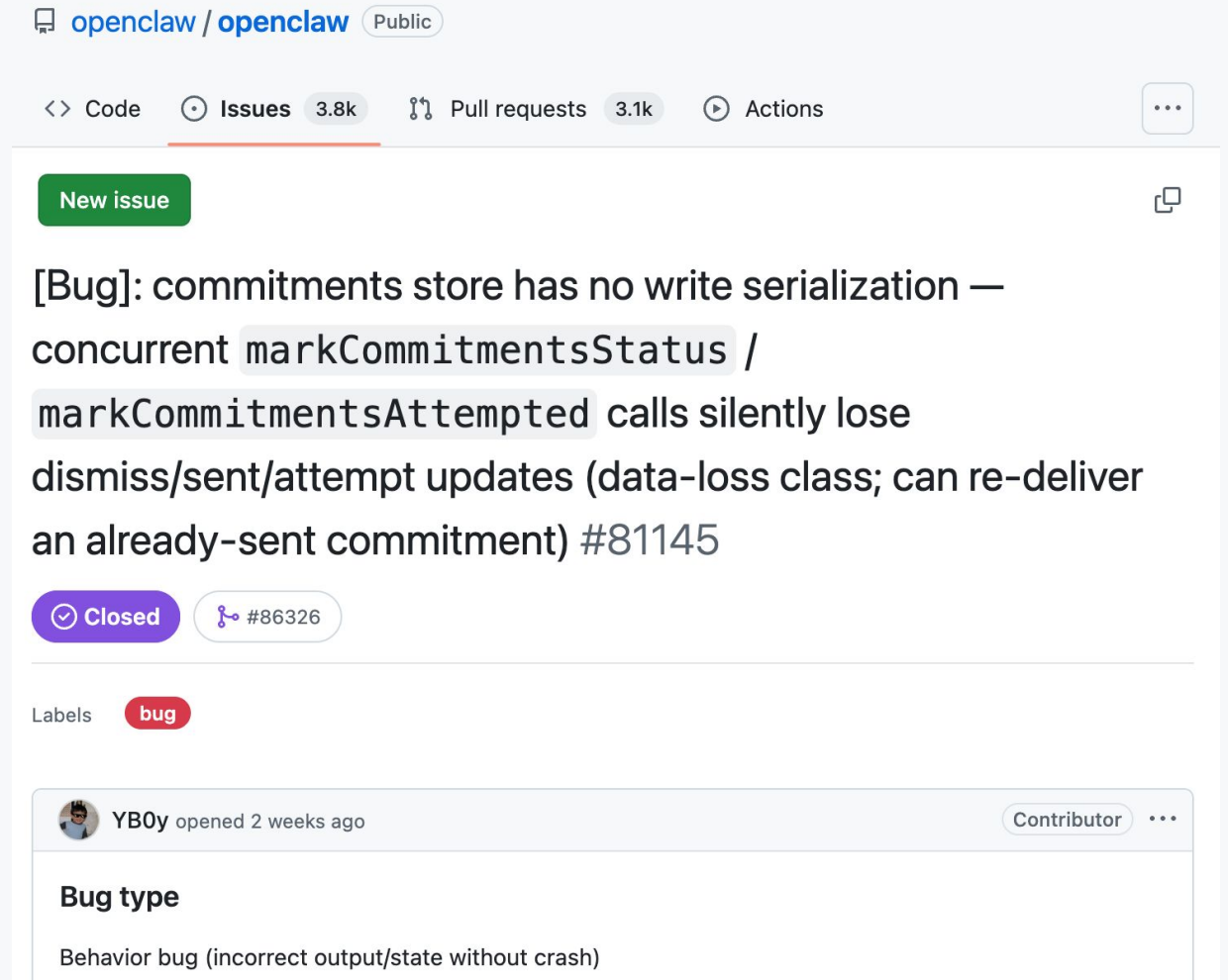
a correction disappears

System state

two writers save competing versions

Harness lesson

serialize mutable state



openclaw / openclaw Public

<> Code Issues 3.8k Pull requests 3.1k Actions

New issue

[Bug]: commitments store has no write serialization — concurrent `markCommitmentsStatus` / `markCommitmentsAttempted` calls silently lose dismiss/sent/attempt updates (data-loss class; can re-deliver an already-sent commitment) #81145

Closed #86326

Labels bug

YB0y opened 2 weeks ago Contributor

Bug type

Behavior bug (incorrect output/state without crash)



Harness Pattern: one ordered commit path

The invariant is not no concurrency.

Parallelize reads

safe work can fan out

Fan out subtasks

inside one run

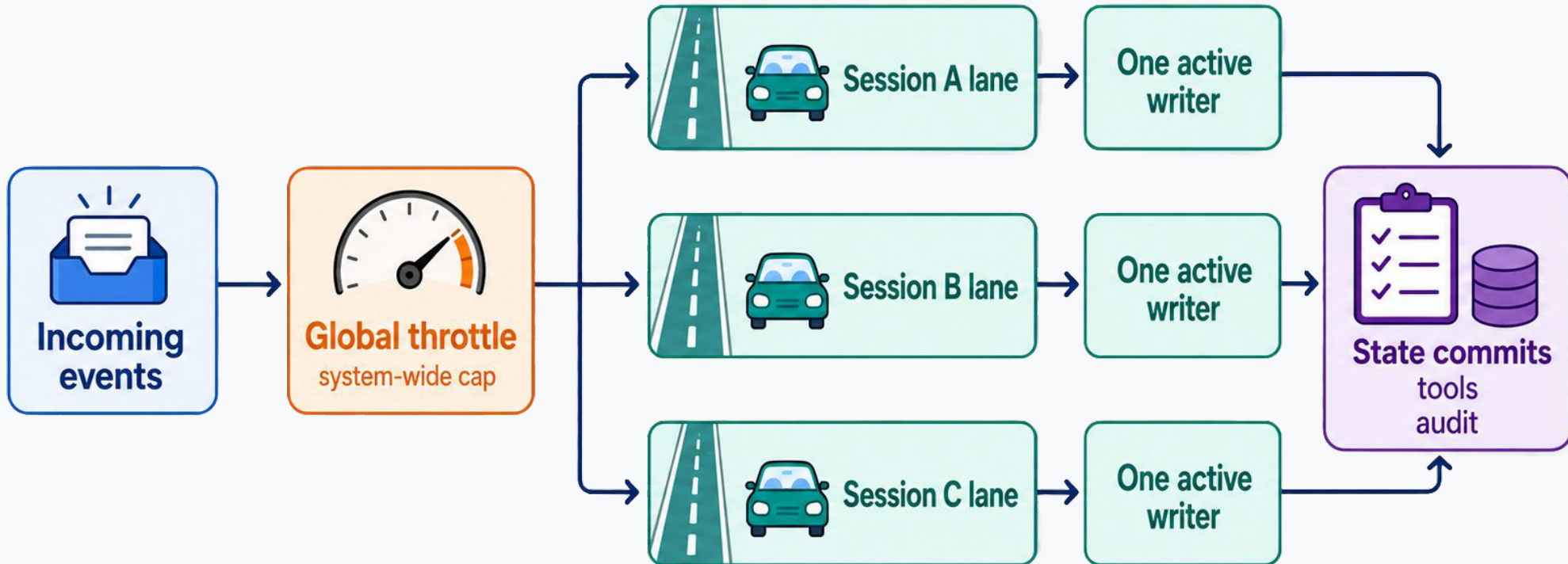
Serialize mutation

one commit path per mutable
state boundary



Lanes make concurrency safe

Parallel across sessions. Single writer inside a session. Globally throttled across the system.



✓ General lesson

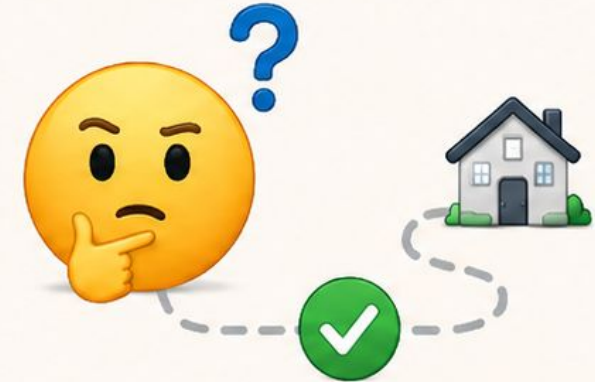
Users experience ordering bugs as agent behavior.



Agent feels forgetful
serialize shared writes



Agent feels dead
detect stuck lanes



Agent feels confused
treat completion as delivery

3



Bound the work

No tool call, lane, loader, or stream gets infinite time.



Incident: dangling tool call

Silence is not a terminal state.

User sees
agent gets stuck

System state
toolCall has no toolResult

Harness lesson
silence needs a terminal event

The screenshot shows a GitHub issue page for the repository 'openclaw / openclaw'. The issue title is 'Session deadlock: dangling toolCall with no toolResult blocks agent loop permanently #53889'. The issue is marked as 'Closed'. The issue was opened by user 'zijunl' on Mar 24. The issue summary states: 'When a tool execution result is lost (network/IPC failure, process killed, timeout), the JSONL session log ends with a toolCall message but no corresponding toolResult. On next message, the agent loop blocks forever waiting for the result that will never arrive.'



Harness Pattern: every boundary needs an ending

Success. Failure. Timeout. Cancel. Max attempts.

Lease

deadline for runs and tools

Watchdog

stuck work becomes visible

Interrupt

recovery controls live outside the stuck run

Receipt

timeouts write terminal records



General lesson

Bound the work before the work bounds you.

Runs

deadline and cancellation

Tools

timeout and error result

Channels

recovery commands do not
wait behind stuck work

4



Constrain authority

Approval is a scoped object, not a memory of a click.



Incident: approval lost its scope

The click was real. The execution context was not.

User sees

I approved this; why blocked?

System state

approval context did not survive

Harness lesson

approval needs scope

The screenshot shows a GitHub repository page for 'openclaw / openclaw' (Public). The navigation bar includes 'Code', 'Issues 3.6k', 'Pull requests 3.5k', and 'Actions'. A green 'New issue' button is visible. The issue title is '[Bug]: Telegram channel blocked by infinite retry loop on expired callback_query approval #82347'. The issue status is 'Closed' with a link to '#82455'. The issue was opened by 'kwangwonkoh' 3 days ago. The 'Summary' section contains the following text: 'When a Telegram inline keyboard button press triggers an exec approval flow and that approval subsequently expires, the Telegram plugin classifies the resulting error as `TelegramRetryableCallbackError` and keeps retrying indefinitely. Because the spooled update is persisted to disk, the retry loop survives gateway restarts and blocks all further Telegram message processing.'



Harness Pattern: authority has shape

Actor. Session. Tool. Arguments. Lifetime. Outcome.

Tool offered to model
requestable, not executable

Approval
binds to one pending action

Expiry
must terminate, not loop

OpenClaw example: capability drift

A disabled tool that still registers is not disabled.

Policy surface

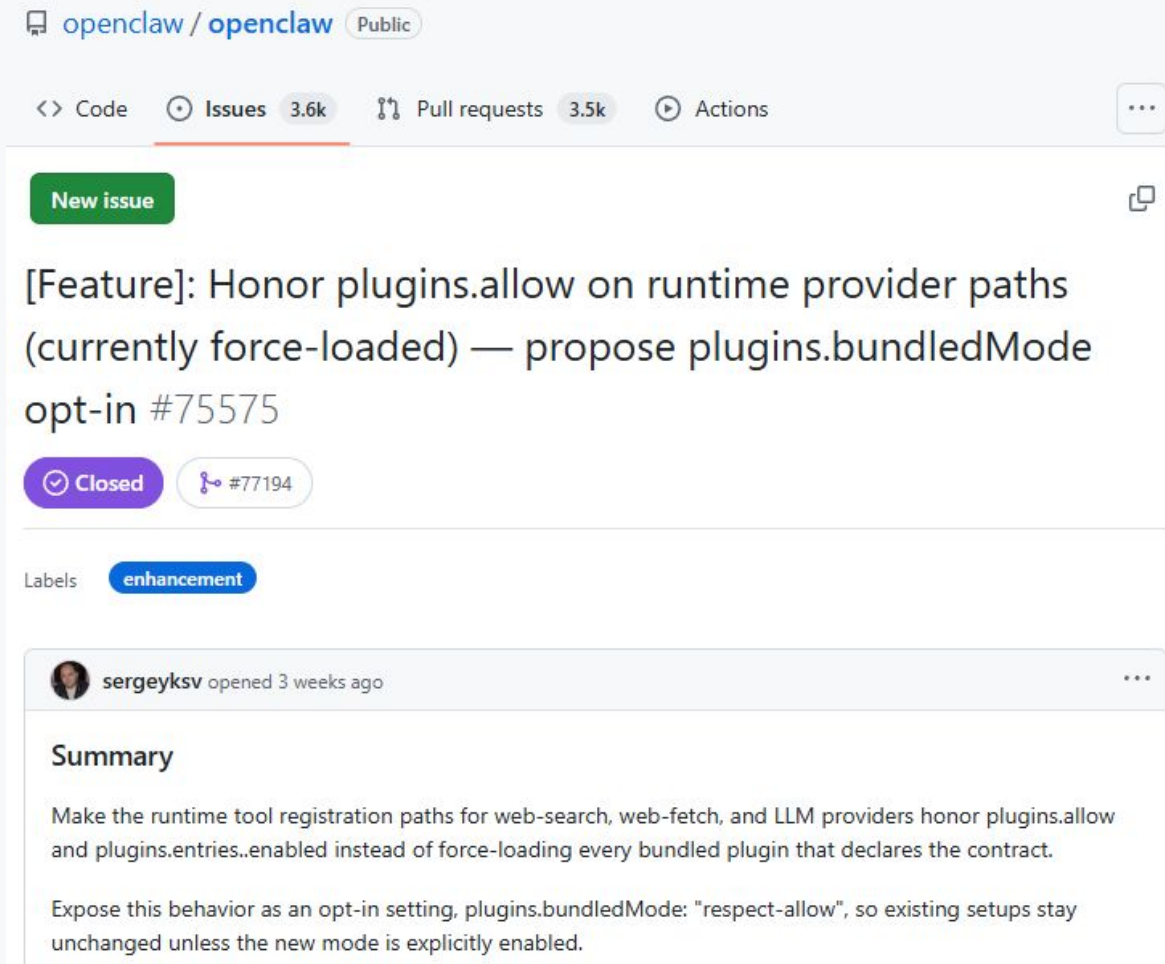
UI says tool is not allowed

Runtime surface

provider path still registers
capability

Harness pattern

capability visibility & execution
authority should converge



openclaw / openclaw Public

<> Code Issues 3.6k Pull requests 3.5k Actions

New issue

[Feature]: Honor plugins.allow on runtime provider paths (currently force-loaded) — propose plugins.bundledMode opt-in #75575

Closed #77194

Labels: enhancement

sergeyksv opened 3 weeks ago

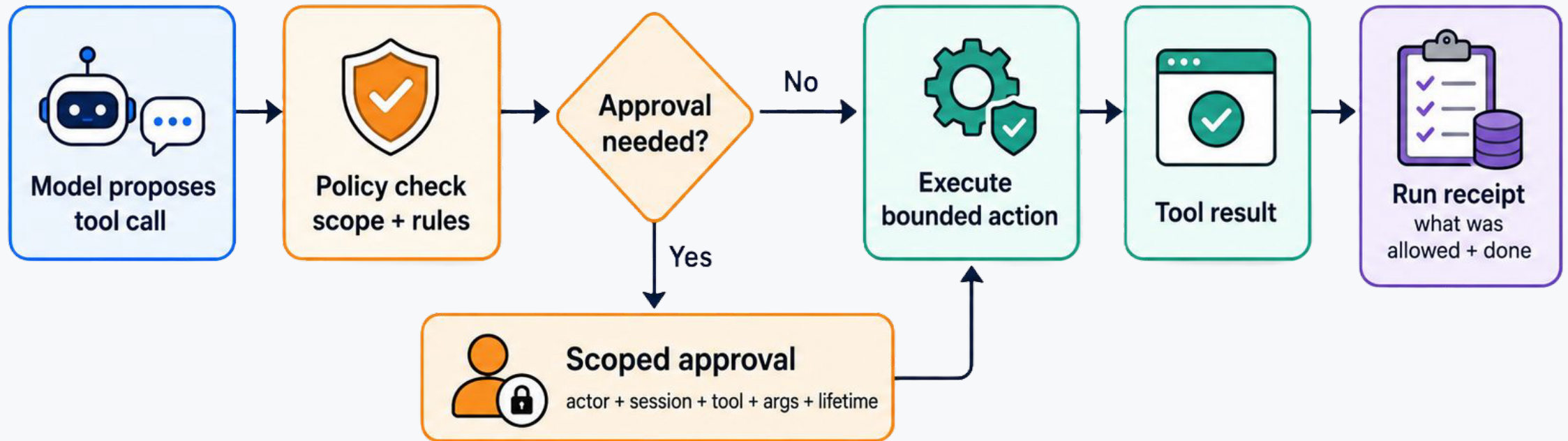
Summary

Make the runtime tool registration paths for web-search, web-fetch, and LLM providers honor plugins.allow and plugins.entries.enabled instead of force-loading every bundled plugin that declares the contract.

Expose this behavior as an opt-in setting, plugins.bundledMode: "respect-allow", so existing setups stay unchanged unless the new mode is explicitly enabled.

Approval path: proposal becomes action

The model can request. The system decides.





General lesson

Capability is not execution.

Least privilege

narrow tool surface

Scoped credentials

right identity for the action

Approval + audit

policy before execution,
evidence after

5



Prove the action

A test proves code. A receipt proves reality.

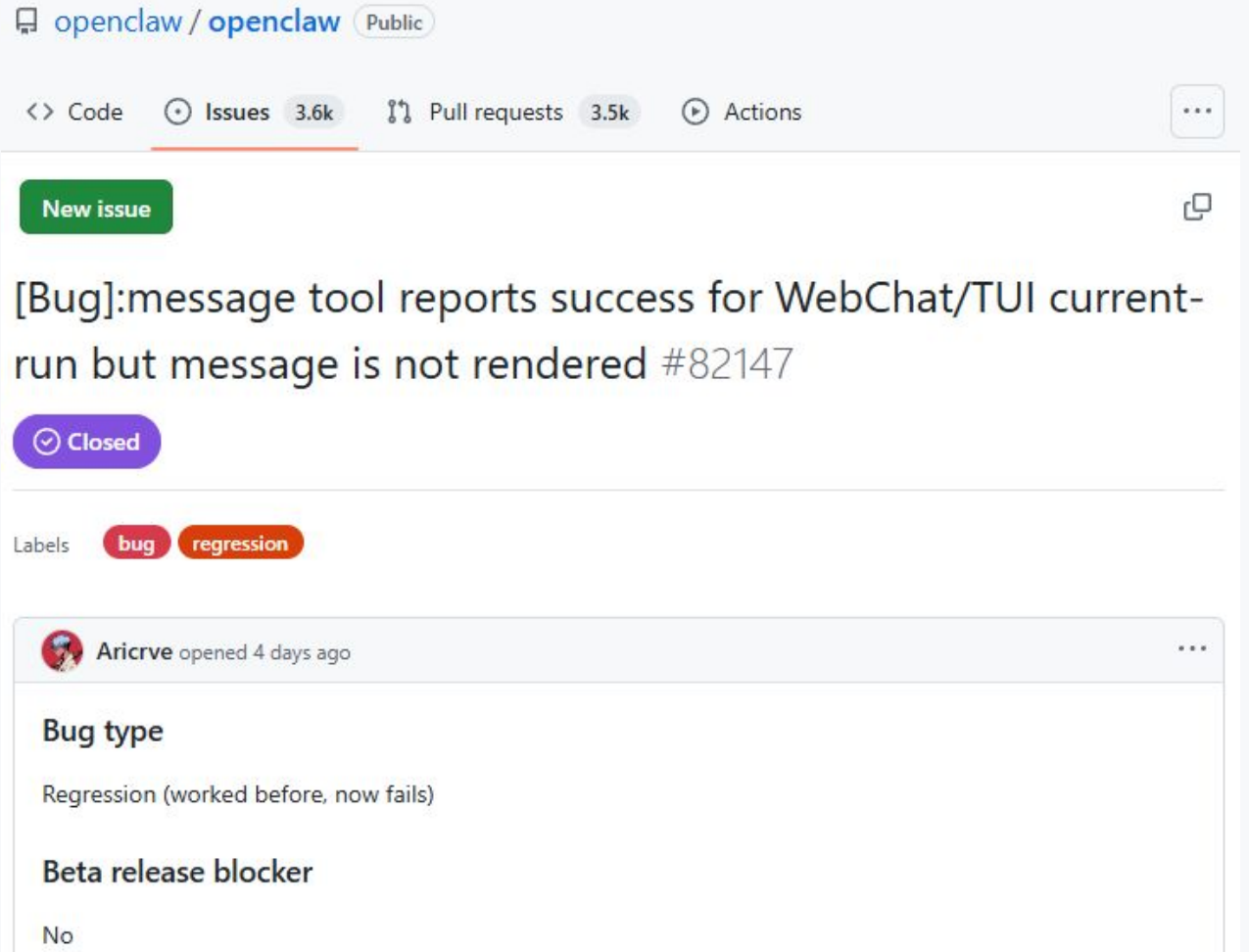
👁️ Incident: tool success, user saw nothing

The tool returned success. The visible client rendered nothing.

User sees
nothing rendered

System state
tool reported success

Harness lesson
proof must cross the edge



openclaw / openclaw Public

<> Code Issues 3.6k Pull requests 3.5k Actions

New issue

[Bug]:message tool reports success for WebChat/TUI current-run but message is not rendered #82147

Closed

Labels bug regression

Aricrve opened 4 days ago

Bug type
Regression (worked before, now fails)

Beta release blocker
No



Harness Pattern: receipt crosses the user boundary

Internal success is not external proof.



OpenClaw example: proof became a review artifact

Not just unit tests. Real behavior proof.

Screenshots

visible channel behavior

Logs and traces

decision path and outcomes

Before / after

show the fixed behavior

Review labels

proof supplied, proof sufficient



 **fix(cron-cli): bound loadCronJobForShow pagination** 1 2
(#83856)
cli **P2** **proof: sufficient** **proof: supplied**
rating:  **platinum hermit** **size: S**
status:  **ready for maintainer look**
#83989 opened 1 hour ago by yaoyi1222 Contributor



General lesson

Before you ship an agent, ask for the receipt.

Not just transcript

what the agent said

Not just tool return

what the tool claimed

Not just unit test

what code path passed



Run receipt audit

Run this on one agent system, not all of them.

State

Can I replay the fact from its owner of record?

Order

Can I explain the ordering rule under concurrent writes?

Bound Work

Can I terminate stuck work without killing the channel?

Authority

Can I name the exact authority envelope?

Proof

Can I prove the user-visible edge saw the outcome?

Recap

Own the state. Order the mutation. Prove the action.

1.  Own the state

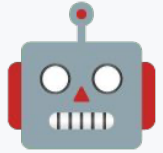
who owns the fact?

2.  Order the mutation

who commits first?

3.  Prove the action

who saw it happen?



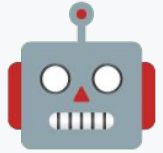
A model proposes.



The harness commits.



The receipt proves it.



A loop can answer a turn.



A harness can survive production.

Thank you.



Let's talk about your harness.

References and Profile

Scan for the sources behind the talk.

 **GitHub issues**



 **OpenClaw docs**



 **OpenAI harness eng**



 **OWASP LLM06**



 **The Agent Stack**



 **My LinkedIn Profile**

