

Red levers: traceable, deterministic agents in production

Jeshua ben Joseph
100monkeys AI, Inc.
2026-10-28

Disclosure: this paper was drafted by an AI agent of 100monkeys.ai from the company's evidence pages and public documentation, and is issued by the company under its founder's name. 100monkeys.ai builds the three products described in sections 4, 6 and 8, so those sections are a maker's account. Every product statement traces to the company's evidence pages; the public sources are listed in the references.

Abstract

Most of the systems that decide things about our lives were never designed by one mind and are owned by no one. Social scientists have described how such systems drift from their purpose and absorb blame, and how the people who once ran them by hand were also the people who could refuse. AI agents now take over that running, at machine speed and without the accidental safety of a person who can say no. This paper argues that an agent system is safe in production only when a person can see every decision it made, can read why it was made in words written at the moment it was made, and can stop it, before the act as well as after. We call the mechanisms that provide those three properties red levers, after Dan Davies' "red handles". We argue that each property belongs outside the model, in a deterministic layer around it: the runtime, the record and the approval. We then describe how three products from 100monkeys.ai embody one property each, state plainly what is not done yet, and list the public sources.

1. The problem: nobody at the helm

Institutions accumulate. Money began as a ledger, bureaucracy as a memory aid, government as a coordination device. Each was a reasonable answer to a real problem, and the compounding of many reasonable answers across generations produced structures that nobody steers. Such a structure has no intent, only persistence. That is why it is hard to correct: an un-owned structure cannot be argued with, because there is no one to argue with.

Robert K. Merton described one mechanism of this drift in 1940. In his account of bureaucratic structure, rules adopted as means come to be followed as ends: "an instrumental value becomes a terminal value" [1]. The people who wrote a rule knew what it was for. Their successors inherit the rule without the reason, and follow it because it is the rule.

Harry Braverman described a second mechanism in 1974. Scientific management separated the conception of work from its execution, so that the people doing the work ran routines whose logic was held somewhere else [2]. For a long time, forms, queues, approvals and compliance checks were executed by people running a process they did not own and often could not see.

Dan Davies, in *The Unaccountability Machine*, names the modern result: the accountability sink, a structure built to remove human discretion and absorb blame [3]. His test is simple: "The extent to which you are able to change a decision is precisely the extent to which you can be accountable for it, and vice versa." A scoring process refuses a loan; the person at the counter has no authority to change it, and the author of the policy sits behind layers of process. The decision happened. Nobody made it.

The arrangement in which people were the runtime had one accidental safety property. People are slow, and people can refuse. A clerk can notice that a rule has produced

something absurd and stop. Davies calls the mechanisms that allow this "red handles": the levers that let the people on the ground halt the machine when an automated rule produces an absurd or destructive outcome [3]. Over decades, many institutions removed them in the name of consistency and cost.

AI agents remove the last of them by default. An agent that reads a ticket, edits a repository, sends an email or moves a record does the work of the clerk, faster and without the clerk's ability to object. Davies observes that AI "has intensified the problem by providing an even easier scapegoat, resulting in a world where decisions travel further than responsibility" [3]. The danger this paper addresses is not malice in the software. It is the same absence of anyone steering, now running at machine speed.

Lisanne Bainbridge identified the engineering side of the same trap in 1983: automating a process leaves the human operator with the tasks the designer could not automate, while removing the practice the operator needs to perform them [4]. A person nominally "in the loop" who sees nothing until something has gone wrong is not in control of anything. The answer is not to put a person back into every step, which defeats the purpose of agents. It is to decide precisely where a person must be able to see, explain and stop, and to build those points so that they hold.

Regulators have arrived at a similar list. Article 14 of the EU AI Act requires that high-risk AI systems "can be effectively overseen by natural persons during the period in which they are in use", and that the people overseeing them are enabled "to properly understand the relevant capacities and limitations" of the system and "duly monitor its operation", "to disregard, override or reverse the output", and "to intervene in the operation of the high-risk AI system or interrupt the system through a 'stop' button or a similar procedure that allows the system to come to a halt in a safe state" [5]. We cite the Article for its list of what oversight requires. Nothing in this paper claims that any product makes a deployment compliant with it.

2. Three properties, and why they belong outside the model

We define a safe agent system operationally. An AI system is safe when a person can see every decision it made, why it made it, and stop it. Each clause is a property that can be checked.

See. Every action an agent takes leaves a record that a person can read later, complete and unaltered. A record that can be rewritten, or that holds only some actions, does not meet the property.

Explain. Each recorded action carries the reason for it, written by its author at the moment of the action, not reconstructed afterwards from a diff or a log. A reason generated later is a guess about intent. Only the author, at the time, knows what the change was for, and for an agent the time is short: the session that made the change is discarded soon after.

Stop. A person can prevent an act before it happens, both in advance (by limits the agent cannot exceed) and at the moment of a consequential act (by an approval that binds to that exact act). Stopping only after the fact is an incident report, not control.

All three properties have to hold outside the model. A rule written in a prompt is a request: the model follows it most of the time, and under a long task, a clever input or

simple drift it may not, with nothing outside the model noticing. A rule written in the runtime is a wall: if the policy says the agent cannot reach a host, the network call fails whatever the model decides. The same holds for the record and the approval. A model asked to keep an honest log keeps the log it keeps. A record that the agent writes through a narrow, append-only interface, and cannot edit afterwards, is honest by construction.

This is what we mean by deterministic agents. The model at the centre stays probabilistic. Its surroundings do not: the policy decides the same way for the same request every time; a change lands completely or not at all; a recorded state renders to the same bytes every time it is read; an approval holds for exactly one state of the work and for nothing else. The AEGIS home page describes its execution loop in the same terms: "Generate, execute, evaluate, refine: the loop runs until deterministic validators pass." [6] In production, that means the parts a person relies on to see, explain and stop must not depend on the model's cooperation.

The three properties also need durability. An agent waiting an hour for a person to approve something cannot lose its place when a process restarts. If it does, the system either drops the work, which teaches people to avoid approvals, or retries from the start, which can repeat acts already done. Waiting for a person is only safe if the waiting survives a crash.

We now describe one product for each property. The order is the order in which a person meets them: the runtime that holds the agent inside its limits, the record that keeps what it did and why, and the companion that asks before it acts.

3. Stop, in advance: limits enforced by the runtime

The first lever is a yes given in advance, in a form a machine enforces and a person can read later. Before an agent runs, someone states what it may touch: which tools, which networks, which files, which secrets. Everything else is refused.

Three design choices decide whether that lever holds.

The default. A deny-by-default policy means the absence of a yes is a no. An allow-by-default policy means every gap in the policy is a permission nobody granted, and agents find gaps quickly, not out of malice but because they try things.

The boundary. The policy is only as strong as the isolation that enforces it. A policy checked inside the same process as the agent can be bypassed by the agent. Isolation at the kernel level, in a micro-VM or a rootless container, puts the check where the agent cannot reach it.

The credentials. An agent that holds a credential can leak it, whatever its prompt says. An agent that is never handed the credential cannot. The credential stays with a broker that performs the authorised call on the agent's behalf.

Self-hosting completes the lever. If the runtime holds the first yes, whoever runs the runtime holds it too. For agents that touch source code, customers or infrastructure, the person who decides what an agent may do belongs to the organisation that bears the consequences, on machines it controls, reading a policy it can audit.

4. AEGIS: the runtime that enforces the limits

AEGIS by 100monkeys.ai, in beta, is our runtime for this first lever. In the words of its home page: "AEGIS is a self-hosted orchestrator that runs autonomous agents in isolated, policy-enforced runtimes with durable FSM state." [6]

Its security policy is declarative and deny-by-default. Agent runs are isolated at the kernel level, in micro-VMs (Firecracker [7]) or rootless containers. Secrets are held by a keymaster, so agents never see the credentials they use. Tool calls go through SEAL, a cryptographic tool-call protocol; the SEAL gateway is a standalone component that AEGIS uses, not a part of AEGIS. Workflows are durable, built on Temporal [8], so the state of a run, including a run waiting for a person, survives a crash or a restart. AEGIS is built to enforce limits by architecture rather than by promises.

AEGIS is open source under the AGPL-3.0 [9], with its orchestrator on GitHub [10] and documentation at docs.100monkeys.ai [11]. It installs with one line, copied from its home page:

```
curl -fsSL https://get.100monkeys.ai | bash
```

5. Explain: a record where every change says why

The second lever is the record. Source control was designed for people, and it assumes the author is still around. Every engineer has run blame on a puzzling line and found a name, a date and the message "fix", then walked over to ask the colleague. Blame was a pointer to a person who held the reason in their head.

When the author is an agent, that assumption fails. The session that wrote the line is gone, its context discarded soon after it landed the change. If the reason was not written down at the moment of the change, it exists nowhere.

A record built for agents has to hold five things. The reason, in the author's words, written when the change lands. The whole change or none of it, so the code and the page that documents it never disagree halfway. No silent merges, because a merged line that neither author wrote is a line nobody intended. A history that cannot be rewritten, so a later reader can trust what they read. And approval bound to exactly what was approved, so a patch made after review voids the review.

Scale makes this urgent. Anthropic's write-up on its own source-control system, Scion, describes merges to its monorepo going from about 2,500 a day in May 2026 to over 30,000 a day in September, as agent work grew [12]. Its stated principles are that clients are always online, that commits are cheap, incremental and lock-free, and that the working copy is optional and sparse. Those are Anthropic's figures about Anthropic; we cite them for the direction of travel, not as a claim about any product of ours.

6. Carbon Forge: the record where every change says why

Carbon Forge is our record. Each sentence below is what its public documentation states [13][14][15][16][17].

Carbon Forge keeps code, files and prose in one record on the server, and people and agents read and write it through the same tools. An agent opens a lane instead of

cloning: a lane is a set of patches over the trunk, and every read on it composes the trunk as it stands now with those patches.

A landing is one changeset: every page, atom, code tree and file it names is written in one transaction, or nothing is. A code change and the page that documents it land in the same changeset, together or not at all.

Carbon Forge never merges text by itself. A patch that no longer applies is reported as a named conflict, and the caller answers it against the trunk it saw.

Every revision and every changeset is hash-chained, and a revert is a new landing, so history only moves forward. A cut names one sealed snapshot of the workspace and never moves, so a release has one answer for as long as the workspace exists.

Blame, grep, diff and history run on the server, and blame shows next to each line the intent the agent wrote when it landed it. The question "why is this line here" is answered with the author's own sentence, written at the moment it mattered.

On a protected trunk, a landing-rules page names the approvers each path needs, and an approval binds to the lane's exact state, so a later patch voids it.

Git is a projection: every code tree renders to a git history and pushes to GitHub fast-forward only, so git tooling keeps working. Agents reach Carbon Forge over the Model Context Protocol [18] with the same tools the editor uses, and a token can be held to a preset such as SCM reader, which writes nothing.

Public workspaces are free and every account includes one free private workspace. Plans for further private workspaces are at <https://carbonforge.dev/pricing>.

7. Stop, at the act: the approval bound to one message

The third lever is a yes given at the moment of the act. The agent has prepared something consequential, an email to a stranger, a payment, a deletion, and it stops. A person reads the exact thing about to happen and approves or refuses that one act.

This lever fails in a quieter way than the first. If the approval attaches to a description of the act rather than the act, the agent can change the act after the yes. If it is a general permission ("you may send emails"), it is the first lever again, given carelessly. If it appears so often that people click through it, or after the act as a notification, it is not a yes at all. A useful approval shows the exact thing, binds to that exact thing, and stops counting if the thing changes. One message, read and approved. Then the next.

8. Zaru: the companion that asks before it acts

Zaru is a personal AI that works alongside its user, remembers them and shows its work. Its memory of the person is theirs to read and edit. The principle behind it is that every other AI harness asks the user to trust it, and Zaru shows the user instead. Its site describes it as "The AI that remembers you and shows every step" and states: "Nothing gets sent until you say yes." Zaru is sold by My Zaru, Inc. at myzaru.com [19].

Here is what the third lever looks like in our own use. When Zaru's mail tool sends an email for our founder, by default the send does not happen. The tool answers that approval is pending, and the message waits in Zaru's web app under Approvals until he

approves that one message. We report this as measured in the company's own use on 28 October 2026, not as a property verified for every account.

9. How the three fit together

The three levers are separate products, and they connect through general interfaces rather than private ones. Carbon Forge's documentation states that it holds no code written for AEGIS or for Zaru; every connection is one of its general surfaces: the MCP endpoint, signed webhooks, the git archive and imports [20]. One connection matters for this paper: a landing rule on a protected trunk can name an AEGIS workflow as a gate, satisfied only by that workflow's own approval [15].

Put together, a person can follow one piece of agent work end to end. The runtime limited what the agent could touch while it worked. The record holds what it changed, all or nothing, with the sentence it wrote about why. The approval held the consequential act until a person said yes to that act. None of this requires trusting the model to report on itself.

10. What is not done yet

We state the limits plainly, because a paper about traceability that hid its own gaps would refute itself.

AEGIS is in beta. Its home page still prints "Pre-alpha" at the time of writing [6], and the outward wording is being brought into line. Its design has run ahead of its implementation: several of its design records are marked partially implemented. Reliability, cost and latency figures that appear in internal design work are modelled, not measured, and none is claimed here.

Carbon Forge has documented limits. In its own words: "No offline work", "No automatic merge", "No merge queue and no built-in CI", "No per-directory permissions", "No git push into Carbon" [21]. It is not a real-time collaborative editor (one writer per page at a time), and its search is English only [14]. Nothing in this paper claims any throughput or scale figure for Carbon Forge; the Scion figures in section 5 describe Anthropic.

Zaru's approval step is described as measured in our own use. By default, a send from its mail tool waits for approval. Whether that default holds for every account, and in every configuration, has not been verified. The open-source Zaru command-line harness is a separate surface and is pre-alpha.

No product makes an AI system safe by itself. These products keep a person in a position to see, explain and stop. They do not decide well on the person's behalf, they do not make a deployment compliant with any regulation, and they have not been audited by a third party. We make no customer, user or adoption claim.

The argument in section 1 is a synthesis. The scholarly quotations are given with attribution to the works cited; readers who want them verbatim in context should read the sources.

11. Disclosure

100monkeys AI, Inc. (100monkeys.ai) builds AEGIS by 100monkeys.ai, Carbon Forge and Zaru, the products described in sections 4, 6 and 8. Zaru is sold by My Zaru, Inc. This paper was drafted by an AI agent of 100monkeys.ai from the company's evidence pages and public documentation, and is issued by the company under the name of its founder, Jeshua ben Joseph, who is the contact for it: jeshua@100monkeys.ai.

To start with the record: [https://carbonforge.dev/?](https://carbonforge.dev/?utm_source=whitepaper&utm_medium=article&utm_campaign=push-2026-10-28)

[utm_source=whitepaper&utm_medium=article&utm_campaign=push-2026-10-28](https://carbonforge.dev/?utm_source=whitepaper&utm_medium=article&utm_campaign=push-2026-10-28)

References

Every URL below was fetched on 28 October 2026 between 15:55 and 16:03 UTC and answered HTTP 200, except reference 1, whose DOI resolver refused automated access; its metadata was confirmed through the Crossref API at 15:55:57 UTC.

1. Merton, R. K. (1940). Bureaucratic Structure and Personality. *Social Forces*, 18(4), 560 to 568. <https://doi.org/10.2307/2570634>
2. Braverman, H. (1974). *Labor and Monopoly Capital: The Degradation of Work in the Twentieth Century*. Monthly Review Press. https://monthlyreview.org/product/labor_and_monopoly_capital/
3. Davies, D. (2024). *The Unaccountability Machine: Why Big Systems Make Terrible Decisions, and How the World Lost Its Mind*. Profile Books. <https://profilebooks.com/work/the-unaccountability-machine/>
4. Bainbridge, L. (1983). Ironies of automation. *Automatica*, 19(6), 775 to 779. [https://doi.org/10.1016/0005-1098\(83\)90046-8](https://doi.org/10.1016/0005-1098(83)90046-8)
5. Regulation (EU) 2024/1689 (the AI Act), Article 14: Human oversight. <https://artificialintelligenceact.eu/article/14/>
6. 100monkeys.ai, home page of AEGIS by 100monkeys.ai, read 28 October 2026 at 15:47 UTC. <https://100monkeys.ai/>
7. Firecracker microVM. <https://firecracker-microvm.github.io/>
8. Temporal, durable execution. <https://temporal.io/>
9. GNU Affero General Public License, version 3. <https://www.gnu.org/licenses/agpl-3.0.html>
10. aegis-orchestrator, source repository. <https://github.com/100monkeys-ai/aegis-orchestrator>
11. AEGIS documentation. <https://docs.100monkeys.ai/>
12. Anthropic (13 October 2026). Redesigning source control at Anthropic. <https://www.anthropic.com/research/redesigning-source-control>
13. Carbon Forge docs: Source control on the substrate. <https://carbonforge.dev/docs/p/features/source-control-on-the-substrate>
14. Carbon Forge docs: Overview. <https://carbonforge.dev/docs/p/overview>
15. Carbon Forge docs: Working in lanes. <https://carbonforge.dev/docs/p/guides/lanes>
16. Carbon Forge docs: The Carbon paradigm. <https://carbonforge.dev/docs/p/architecture/carbon-paradigm>

17. Carbon Forge docs: Source control at agent scale. <https://carbonforge.dev/docs/p/use-cases/source-control-at-agent-scale>
18. Model Context Protocol. <https://modelcontextprotocol.io/>
19. Your Zaru, My Zaru, Inc. <https://myzaru.com/>
20. Carbon Forge docs: The full workflow with AEGIS and Zaru. <https://carbonforge.dev/docs/p/use-cases/full-workflow-with-aegis-and-zaru>
21. Carbon Forge docs: Source control: how it is built and why. <https://carbonforge.dev/docs/p/architecture/source-control>

This work is licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0), <https://creativecommons.org/licenses/by/4.0/>. Copyright 2026 100monkeys AI, Inc.