

Dartmouth to the Data Center

AI's long arc toward agency

Why MCP matters

Jonathan Edwards
jone@redhat.com
Sr Tech Acct Mgr FSI/Shift

Red Hat

The ideas are old. The implementation is new.

A refrain that repeats five times in this talk

Idea proposed — named decades before the technology existed to realize it.

Why it stalled — the substrate wasn't ready: no learned models, no standard way to act.

What changed — deep learning and standardized protocols finally supply what was missing.

Red Hat

1950 — Turing asks the question

In "Computing Machinery and Intelligence," Alan Turing sidesteps "can machines think" entirely. He proposes a behavioral test — the imitation game — that reframes intelligence as something you can observe rather than define.

- Turing didn't exaggerate — he gave the field a testable question.
- That same question, asked of an LLM today, has become practically moot.

Red Hat

The founding claim

"Every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it." — the 1955 Dartmouth funding proposal

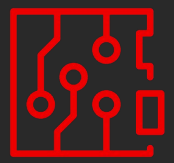
1956 — The Dartmouth Conference

John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon convene a Rockefeller Foundation-funded summer workshop at Dartmouth College — and give the field its name.

Red Hat

Why it stalled

Symbolic AI hit a wall it couldn't get past



Combinatorial explosion

Toy "microworlds" like SHRDLU worked. Real environments have far too many states and actions for symbolic search to explore exhaustively.



Symbol grounding

Symbols like "block A on block B" had no connection to sensory reality — an agent could reason about a world it could never perceive.



Funding pullback

The Lighthill Report (1973) and later DARPA cuts followed years of agentic ambitions overpromising relative to delivered results.

Red Hat

What changed

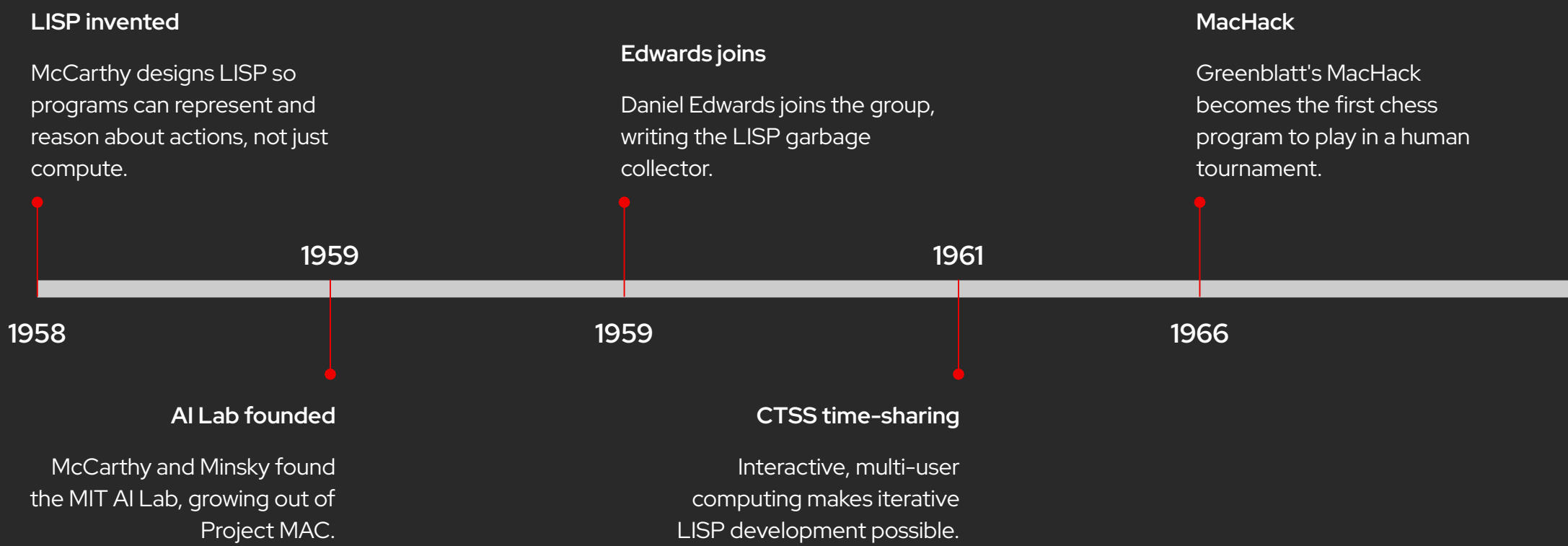
Deep learning gave the field what symbolic hand-coding never could: a general, learned substrate. Instead of encoding rules for perception and language, systems began learning statistical patterns directly from data — starting with AlexNet (2012) and accelerating through the transformer architecture (2017).

- Not a new answer to the Dartmouth question — the first practical attempt at one.

Red Hat

LISP and the MIT AI Lab

The substrate that made interactive reasoning possible



Red Hat

Why LISP was interesting

Extension of IPL – structured linked lists were tied to constructs, address and decrement registers on the IBM 704 (cons, car, cdr)

- S-expressions. Symbolic expressions like (plus 1 (times 2 3)) represent both code and data in the same structure. A LISP program can navigate, build, inspect, and rewrite other LISP programs.

- M-expressions. McCarthy's intended "meta-expression" notation, such as f[x; y], was meant to be the readable way to write programs, with S-expressions as the internal data form. In 1958–59, Steve Russell hand-coded the eval function in machine language, and S-expressions became the way people actually wrote LISP. M-expressions never caught on.

Red Hat

Why LISP was important

Introduced inline functions (slug), anonymous functions, recursion as primary flow control, code as data (homoiconicity) and eventually made way for time sharing (CTSS) and scheduling hacking (ITS - 1967)

- Lambda calculus. LISP borrowed Alonzo Church's lambda notation for defining anonymous functions. It isn't a faithful implementation of the calculus, but the idea of functions as first-class values came through.
- ITS (Incompatible Time Sharing) was later the inspiration for the open source ideas (GNU) and custom keyboard hacking (Space Cadet, bucky bits, chording)

Red Hat



Dan Edwards (left) and Peter Samson playing Spacewar!
Photo: Computer History Museum, ca. 1962

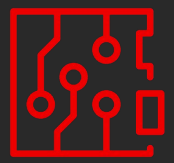
From the AI Lab, briefly personal

Daniel Edwards joined McCarthy's group in 1959. He wrote the LISP garbage collector, later proved (with Timothy Hart) that alpha-beta pruning is equivalent to minimax search — and, in the same TMRC/AI Lab circle, added the gravitational "sun" to Spacewar!, one of the first interactive computer games.

Red Hat

Why it stalled

Symbols alone couldn't carry the reasoning



Hand-coded symbols

Every rule an agent could reason with had to be authored by a person — systems couldn't acquire knowledge on their own.



No generalizable world model

Techniques that worked in a block world or on a chessboard didn't transfer to open-ended, ambiguous domains.



Cyc's incomplete effort

Doug Lenat's Cyc project (started 1984) tried to hand-encode common sense as explicit rules — a heroic, still-unfinished, decades-long attempt.

Red Hat

What changed

The transformer architecture and large-scale self-supervised training gave models a way to acquire broad, flexible knowledge directly from text — without anyone hand-authoring the rules. Reasoning about action stopped requiring a programmer to enumerate every case in advance.

- McCarthy wanted programs that could reason about what to do next. LLMs are the first general-purpose attempt at that, six decades later.

Red Hat

1959 – Samuel's checkers program

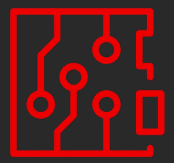
At IBM, Arthur Samuel builds a checkers program that improves by playing itself and adjusting its own evaluation function – the first machine to genuinely learn from experience rather than execute fixed rules. The same year, Samuel coins the term "machine learning."

- A name and a working demo existed in 1959 – fifty-plus years before deep reinforcement learning made the same idea work at scale.

Red Hat

Why it stalled

Narrow wins that never generalized



Narrow, game-specific wins

Greenblatt's MacHack (1967) and IBM's Deep Blue (1997) triumphed at chess through brute-force search and hand-tuned heuristics — not learning.



No generalization

A system built to win at chess told you nothing about how to act in a messier, open-ended domain.



Search, not intuition

Deep Blue evaluated roughly 200 million positions per second. It never learned an evaluation function — humans hand-coded it.

Red Hat

What changed

DeepMind's DQN (2013) learned to play Atari games directly from raw pixels. AlphaGo (2016) paired deep neural networks with search to beat Lee Sedol. AlphaZero (2017–18) then dropped human data entirely, learning chess, Go, and shogi purely through self-play.

- Samuel's 1959 self-improvement loop, finally running at a scale that generalizes across entire games — not just one.

Red Hat

A named problem, unsolved for 65 years

McCarthy didn't just gesture at this problem — he named it precisely, decades before anyone had a real path to solving it.

1959 — "Programs with Common Sense"

In the same year he founds the AI Lab, McCarthy publishes a paper naming what he sees as the central unsolved problem for any reasoning agent: representing and using ordinary common-sense knowledge about the world.

Red Hat

Why it stalled

Doug Lenat began Cyc in 1984 with a direct approach to McCarthy's problem: hand-encode common-sense facts as explicit logical rules. Decades and millions of hand-written assertions later, the project remains incomplete.

- The sheer volume of common-sense knowledge, and its constant exceptions and ambiguity, proved too large to enumerate by hand.

Red Hat

What changed

Large language models are the first systems to acquire something resembling common sense at scale — not by hand-coding it, but by absorbing it statistically from the accumulated text of human experience.

- Imperfect and occasionally wrong in surprising ways — but a fundamentally different approach than Cyc's, and the first attempt with a real shot at McCarthy's 1959 challenge.

Red Hat

The perceive-decide-act loop

The McCarthy/Minsky-era definition of an agent — unchanged 70 years later

Perceive — take in information about the environment.

Decide — reason toward a goal given that information.

Act — change the environment, then observe the result and loop back.

Red Hat

Why it stalled

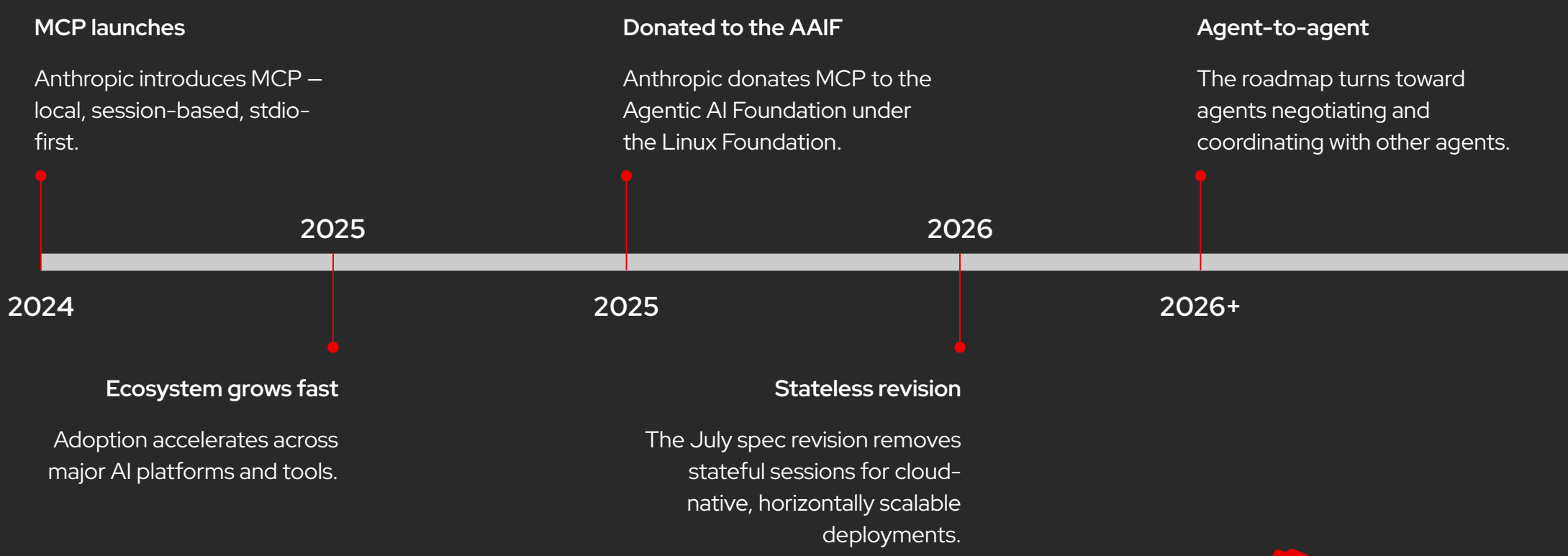
Even once LLMs could reason fluently, every connection between a model and a real tool, database, or system was custom-built. Every integration was its own one-off.

- An N×M problem — N models, each needing a bespoke connector to each of M tools — with no common protocol for the "act" step of the loop.

Red Hat

The Model Context Protocol

A standardized "act" layer, evolving fast



Red Hat

Inside MCP: three primitives

JSON-RPC 2.0 messages over stdio or Streamable HTTP



Tools

Actions the model decides to invoke: tools/list and tools/call.



Resources

Data the application reads into context: resources/list and resources/read.



Prompts

Reusable templates the user selects: prompts/list and prompts/get.

Red Hat

Inside MCP: one stateless request

Why the 2026-07-28 revision scales like any web API

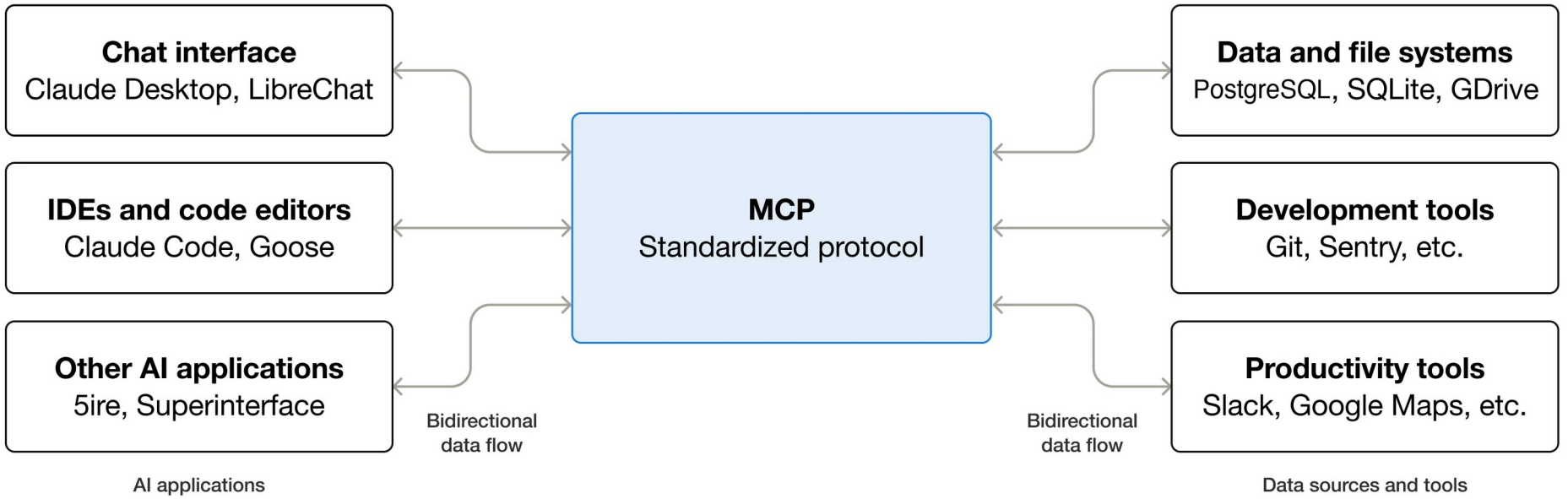
Client — sends a self-contained request. The protocol version travels with every call.

Load balancer — routes it to any server instance. No session, no sticky routing.

Server — returns the result. server/discover advertises capabilities; lists are cacheable.

Red Hat

Inside MCP: one stateless request



https://modelcontextprotocol.io/docs/2026-07-28/getting-started/intro

Red Hat

Configuration Examples

json

```json
{
  "mcpServers": {
    "bookshop-stdio": {
      "type": "stdio",
      "command": "uv",
      "args": ["run", "python", "server.py"],
      "env": { "API_KEY": "${BOOKSHOP_API_KEY}" }
    },
    "bookshop-http": {
      "type": "http",
      "url": "http://127.0.0.1:3001/mcp",
      "headers": { "Authorization": "Bearer ${BOOKSHOP_TOKEN}" }
    },
    "bookshop-sse": {
      "type": "sse",
      "url": "http://127.0.0.1:3001/sse"
    }
  }
}
```


Red Hat

Where 70-year ideas meet production infrastructure

How Red Hat is turning this history into practice

Red Hat

MCP across the portfolio



Lightspeed

Conversational, AI-assisted operations and knowledgebase search.



OpenShift

Container platform automation exposed as agent-callable tools.



RHEL

System-level operations and diagnostics for agent workflows.



Ansible Automation Platform

Enterprise-wide automation, invoked directly by agents.



Red Hat AI Inference

Model serving exposed as a standardized, agent-callable endpoint.



Red Hat Desktop

Local build tooling wired directly into the agentic workflow.

Red Hat

Agentic skills

Pairing what to do with how to act



Skills

What to do — domain knowledge and playbooks, like searching the Red Hat Knowledgebase.



MCP servers

How to act — Lightspeed, OpenShift, RHEL, and Ansible Automation Platform.



The result

An agent that finds the right answer in the Knowledgebase and can act on it directly.

Red Hat

Secure by default

The perceive-decide-act loop, made safe by construction

Perceive — a CVE or vulnerability is detected in a running workload.

Decide — the security skill helps reason about severity and remediation.

Act — the agent triggers a rebuild from Hardened Images or an Ansible remediation.

Red Hat

Some Links

Just a small sampling of links around enabling MCP across your estate

- https://www.redhat.com/en/topics/ai/what-is-model-context-protocol-mcp
- https://modelcontextprotocol.io/docs/2026-07-28/getting-started/intro
- https://www.redhat.com/en/agentic-skills
- https://www.redhat.com/en/topics/ai/mcp-vs-apis
- https://www.redhat.com/en/blog/manage-mcp-servers-red-hat-openshift-mcp-lifecycle-operator
- https://developers.redhat.com/articles/2026/10/08/manage-rhel-mcp-servers-and-red-hat-lightspeed

Red Hat



Thank you

The ideas behind agentic AI are seventy years old. The infrastructure to run them is finally here. Questions welcome.

linkedin.com/company/red-hat

facebook.com/redhat

youtube.com/@redhat

x.com/RedHat

Red Hat