ARCHITECTURE NOTE

Stop Comparing MCP, RAG, and Agents

Three things often compared as alternatives that belong at three different layers of the same system.

Chandrakanth Anne · Enterprise Applications Architect · September 2026

Ask three teams to compare MCP, RAG, and agents and you'll get a bake-off: a matrix, a winner, a recommendation to standardise on one.

The matrix is the problem. These three don't compete, because they don't answer the same question.

RAG answers: how does the model know things it was never trained on?

MCP answers: how does the model reach the systems that hold those things?

An agent answers: who decides what to do next?

A data-access pattern, an integration standard, and a control-flow pattern. You would never ask a team to choose between a query layer, a connector standard, and a process engine. There is no reason to ask them to choose here.

Any production system doing something genuinely useful runs all three at once: an agent loop that reaches its tools over MCP, one of which performs retrieval.

Here is what each layer actually is, and where each one breaks.

RAG governs what the model knows

Retrieval-augmented generation means finding relevant material at request time and putting it into the model's context before it generates. That is the whole idea. The engineering is all in the word *relevant*.

It exists because of three constraints. The model's training data is fixed at inference time and does not contain your contract terms or last night's reconciliation break. Context windows are metered, so you cannot put a ten-terabyte document estate in front of the model on every call. And enterprise content is entitled two people asking the same question must not get the same evidence if their access differs.

Most teams build the pipeline correctly and skip one box: **the entitlement filter.**

Filter on the caller's rights at retrieval time, not after generation. Once the model has read a document the user was never allowed to see, the answer is already shaped by it — and the citation you proudly attached is now the audit evidence against you.

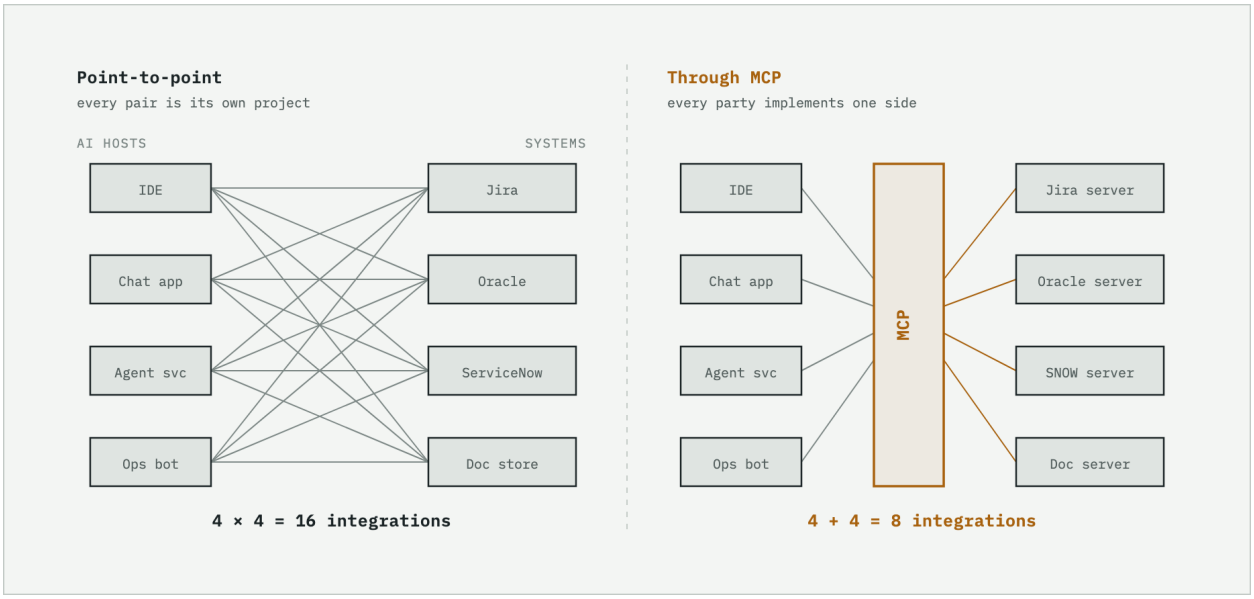
On "long context killed RAG": it did not. A larger context window does not remove four jobs — managing cost (you pay per token every call), maintaining freshness (something still has to choose what fills the window), enforcing entitlement (filtered retrieval is where access control attaches), and preserving provenance (citation requires selected documents).

What long context genuinely changed is the shape. Retrieval is moving from a fixed pre-generation step to a tool the model calls repeatedly, refining the query as it learns what it is looking for. That is the context layer being driven by the control layer the layers composing, not replacing each other.

MCP governs what the model can reach

The Model Context Protocol is an open standard for how an LLM application connects to external systems. It uses JSON-RPC over stdio or Streamable HTTP. Deliberately unglamorous — connector standards succeed by being boring.

The problem it solves is one every integration architect has costed before. You have N AI-enabled applications and M internal systems. At the interface layer, without a standard, that can mean N×M bespoke adapters, each written twice once badly, once after the first one broke. With a standard, each system implements one server and each application one client: N+M interfaces. The operational work does not disappear, but the integration pattern is reusable.



The standard lowers the marginal cost of the next integration.

Three roles: a **host** (the application the user touches), a **client** inside it holding one session per server connection, and a **server** — code that may be internal or third-party, with its own release cycle and trust boundary.

That last clause is the governance point, and it is the one that gets missed.

An MCP server's tool metadata and outputs can influence model behaviour. They cross into your trust boundary on every session. Treat them as untrusted input until you have established the server's provenance, scope, and controls.

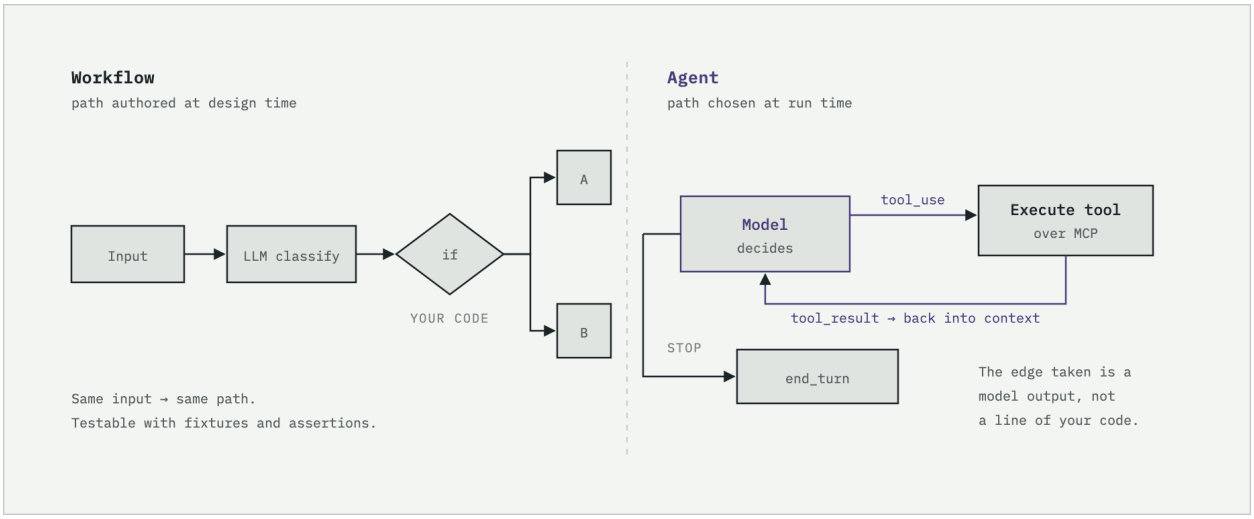
MCP standardises the plumbing, not your governance. The protocol does not itself provide your authorization policy, enforce least privilege, or produce an audit trail. Adopting it is an integration decision with a security review attached and the protocol is the easy half.

The agent loop governs what it decides to do

The word has been stretched until it covers anything with a model in it. The distinction that carries weight in a design review is between a workflow and an agent.

In a **workflow**, your code constrains the allowed path and decides what happens next. The model may be called at fixed points, but you can draw the route and its guardrails before it runs.

In an **agent**, the model gets a goal and a set of tools and, within the constraints you set, selects which tools to call and when it is finished. The route is a run-time output, not a fully authored design-time artifact.



In one, your code owns the branch logic and you can unit-test it. In the other, selection is driven by a model output.

Anyone who has built BPEL or ESB orchestration will recognise exactly what changed and exactly what did not.

The orchestration concerns are identical correlation, compensation, retry, timeout, idempotency, audit. What moved is the branch. In a BPEL process you author every decision node and the engine evaluates it deterministically. In an agent loop the decision node is a model inference, and it will not reliably resolve the same way twice.

Many of the hard operational issues follow from that one substitution:

- The exit condition is a model output, so something outside the model needs an iteration ceiling and a cost budget.
- Tool calls can run with whatever credentials you give them, and the model may request repeated calls — so scope credentials per task, not per platform.
- Automatic retries are safe only when the operation is idempotent or protected by an equivalent deduplication control.
- A non-deterministic component cannot be adequately regression-tested only with fixtures and equality assertions. It needs a scored eval suite as a release gate. Teams with real test discipline adapt fast; teams without it discover they have no gate at all.

Before building one, four questions. Are the steps genuinely unknowable in advance? Does the outcome justify multiple model calls and minutes of latency? Is the model actually good at this on your data? Can a wrong action be detected and reversed?

A “no” to any of them is a strong reason to prefer a workflow or narrow the agent’s scope.

What to ask in a design review

None of these is primarily a model-choice question.

1. Which layer is this context, capability, or control? If the answer is “all three,” what is the smallest version that is only one?
2. Where is entitlement enforced, and is it before or after the model sees the content?
3. Which MCP servers are in scope, who publishes them, and what egress do they open?
4. What stops the loop an iteration ceiling, a cost budget, a wall clock, or nothing?
5. What does the audit record contain, and could it reconstruct a disputed run six months from now?

The short version

RAG governs what the model knows. MCP governs what it can reach. The agent loop governs what it decides to do. Three layers of one system.

The engineering that matters sits where it always has at the boundaries, on the identities, and in the audit trail.

The genuinely new problem is that one of your control-flow decisions is now a probabilistic inference rather than a deterministic branch. Nearly everything else here is ordinary architectural discipline. It just has to be applied to a component that will not behave identically twice which is precisely why the controls have to live outside it.

What's the most expensive version of this mistake you've watched a team make building the agent that should have been a workflow, or shipping the retrieval layer with no entitlement filter?