MCP

Model Context Protocol — Basics

Client & server roles · Tools / Resources / Prompts · Transport · Wire protocol

SWE / AI Learning Tracker · 2026-08-15

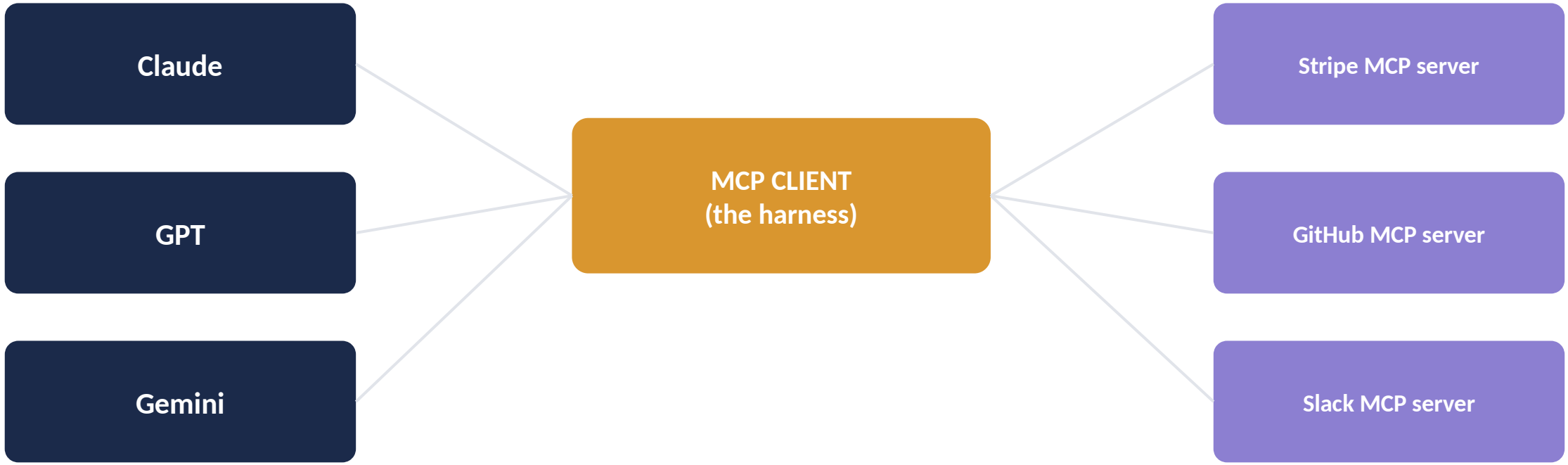
THE PROBLEM

Before MCP: every (LLM, tool) pair needs its own glue code

N × M hand-written integrations — one per (LLM, tool) pair

THE FIX

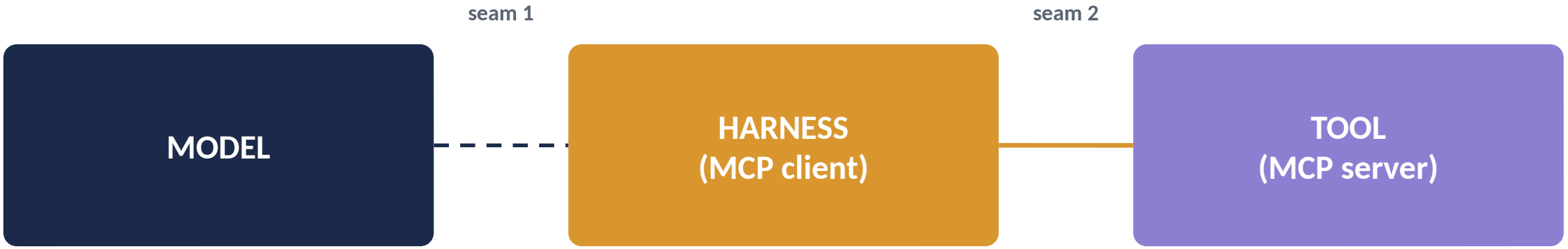
MCP: N + M, not N × M



3 LLM adapters (N) + 3 MCP servers (M) = 6 pieces, not 9

TWO INDEPENDENT SEAMS

MCP standardizes seam 2 only



Seam 1 (model ↔ harness): each LLM's native tool-call format (tool_use, tool_calls). Not MCP's concern — written once per LLM, reusable across every tool.

Seam 2 (harness ↔ tool): the one MCP standardizes — JSON-RPC 2.0. Written once per tool, reusable by any MCP client.

SERVER PRIMITIVES

Tools, Resources, Prompts

TOOLS

Model-controlled

Actions. The model decides, mid-response, when to invoke one — e.g. create_issue().

RESOURCES

Application-controlled

Read-only data. A user/client attaches it, like attaching a file — model never fetches it on its own.

PROMPTS

User-controlled

Reusable message templates. A user picks one (e.g. /review-pr); it fills in as if typed.

THE ORGANIZING AXIS

Not "what it does" — "who decides to use it"

Tools	**The model**	mid-conversation, based on reasoning
Resources	**The user / client app**	attached before sending, like a file
Prompts	**The user**	explicitly selected from a menu

END-TO-END FLOW

The model never talks to the server directly

1	**You type a message**	in Claude Desktop (the client)
2	**Client calls the model**	message + tool list + any attached resources
3	**Model returns tool_use**	its native format — decides to call a tool
4	**Client translates → MCP**	sends tools/call (JSON-RPC) to the server
5	**Server executes + returns**	e.g. hits the real GitHub API
6	**Client feeds result back**	model reads it, writes the final answer

A SECURITY NUANCE

Control-model ≠ safety guarantee

Resources being application-controlled reduces one risk: the model spontaneously pulling in data a human never chose to expose.

It does NOT reduce content-injection risk: a Tool that returns data (read_file, search_docs) carries the exact same risk as a Resource once its content lands in context.

The real boundary: any untrusted content entering context is dangerous, regardless of which primitive delivered it.

→ ties to the "lethal trifecta": private data access + untrusted content + external comms (OWASP LLM Security track)

TRANSPORT LAYER

STDIO vs. HTTP

STDIO

- Client spawns server as a child process
- JSON-RPC piped over stdin / stdout
- No network, no ports
- 1 client : 1 server, same machine
- Local tools: filesystem, local git

Streamable HTTP

- Server runs remotely, over the network
- Client sends JSON-RPC as HTTP POST
- Server can stream multiple events back
- Many clients : 1 server
- Shared/hosted tools: company Jira, GitHub

WHY STATELESS HTTP (JULY 2026 SPEC)

Sticky sessions don’t scale — REST’s reasoning applies here too

STATEFUL (old)

Server remembers each client across requests → that client’s requests must hit the same instance ("sticky sessions"). Instance restarts lose the session. Hard to load-balance.

STATELESS (current)

Every request is independent — any instance can handle it. Freely add / remove / restart instances behind a load balancer. Same scaling model as ordinary REST APIs.

DISCOVERY, ON THE WIRE

JSON-RPC 2.0 — tools/list → tools/call

REQUEST

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/list"
}
```

RESPONSE

```
{
  "result": {
    "tools": [{
      "name": "create_issue",
      "inputSchema": { ... }
    }]
  }
}
```

CALLING IT

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "tools/call",
  "params": {
    "name": "create_issue",
    "arguments": {
      "repo": "x",
      "title": "y"
    }
  }
}
```

RECAP

MCP Basics, in one line

"MCP standardizes the tool side of integration — harness ↔ tool — turning N×M bespoke glue code into N reusable adapters + M reusable servers."

Client / Server	App-side harness ↔ tool-side process, JSON-RPC 2.0
Tools / Resources / Prompts	model / app / user — that's the whole taxonomy
Model never calls the server	the harness always mediates, both directions
STDIO vs. HTTP	local 1:1 process pipe vs. remote many:1 network service
Stateless scaling	any request, any instance — same model as REST

UP NEXT

Advanced MCP

- Auth evolution — OAuth 2.1 retrofit
- July 2026 stateless spec redesign, in depth
- Security surface — STDIO/RCE exposure (~200K servers), lethal trifecta
- MCP vs. A2A vs. ACP — the protocol landscape
- Production patterns — registries, gateways, dynamic tool lists
- MCP Apps extension