MCP dev summit · SCALEKIT

The Benchmark that Almost convinced us *MCP was wrong*

Ravi Madabhushi
CTO, Scalekit

SCALEKIT & MCP

Our journey in the *MCP ecosystem*

MAY 2025

Inaugural MCP Dev Summit

San Francisco. T-shirts: "MCP forgot auth. We are fixing it." Not ironically.

JULY 2025

MCP Auth Live

Drop-in Auth for MCP servers. Real production adoption - teams shipping, not just demoing.

Learn to add auth to your MCP Servers

Play

2025

100+ production MCP servers

Real enterprise deployments on Scalekit. Supply side of the ecosystem moving.

100+

PRODUCTION MCP SERVERS

MCP

100+ Agent → MCP

Agents connecting to multiple servers at runtime. Both sides of the ecosystem. Same problem, different angle.

THE BENCHMARK

75 runs. CLI vs. MCP. *Identical tasks.*

APPROACH A

CLI

Direct command execution. Tools called explicitly. No schema negotiation. No protocol overhead. Minimal surface.

Vs

APPROACH B

MCP

Full tool discovery. Schema loading on every request. Client-server handshake. The full protocol stack as designed.

Same tasks • Same model • 75 runs each • **Every efficiency metric recorded**

THE RESULTS

CLI won *Every* efficiency metric.

SIMPLE TAS TRIAGE · ONE WORKFLOW

CLI tokens	1,365
MCP tokens	44,026

MCP is 32x more expensive - almost entirely schema overhead. 43 tool definitions injected into every conversation, most never touched.

RELIABILITY • 75 RUNS

CLI success rate	100%
MCP success rate	72%

MCP failed 28% of the time - TCP-level timeouts on remote servers that never responded.

If we'd stopped there: use CLI, skip MCP, move on.

THE BENCHMARK WAS RIGHT. THE SCENARIO WAS WRONG

But the benchmark tested

One scenario?

HOW PRODUCTION AGENTS CONNECT TODAY

Your agent needs to reach *external enterprise systems*

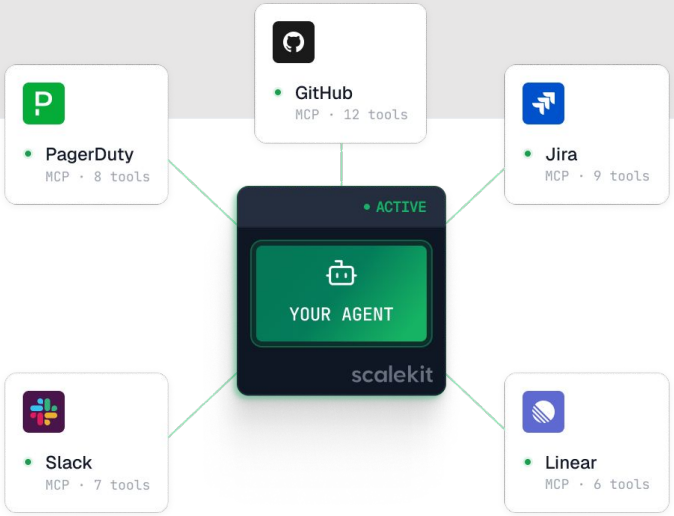
- EXAMPLE: INCIDENT TRIAGE AGENT

Triage open incidents assigned to this user. Pull from **PagerDuty**.

Cross-reference recent **Jira** tickets and **GitHub** commits.

Summarize and post to **Slack**.

3 systems · multiple users · runs continuously in production



Every MCP server exposes its full catalog. Your agent loads all of it — **every tool, every request — whether it needs them or**

WHAT TOOL DISCOVERY LOADS

Every tool definition costs tokens.

All 42 load on every request.

ONE TOOL SCHEMA – PAGERDUTY: GET_INCIDENTS

```
{
"name": "get_incidents",
"description": "List open incidents assigned
                to the authenticated user",
"inputSchema": {
        "type": "object",
        "properties": {
        "status": {"type": "string",
                "enum": ["triggered","acknowledged"]},
"limit": {"type": "integer"}
        }
}
```

~266 tokens. One tool. One server

5 SERVERS CONNECTED. 42 TOOLS TOTAL

MCP: PagerDuty	8 tools
MCP: GitHub	12 tools
MCP: Jira	9 tools
MCP: Slack	7 tools
MCP: Linear	6 tools

42 tools x ~266 tokens each

~11,200 tokens

before the user types a single word

EXAMPLE: ONE AGENT FOR THREE USERS

Three users. *Three different jobs.*

M

Maya

Senior platform engineer

- Enterprise customer

J

Jordan

Junior engineer

- Startup customer

S

Sam

Internal SRE

- Your own team

Expected
Tool surfaces

M **Maya**
Senior platform engineer
• Enterprise customer

Their tool surface
`Pagerduty.admin` `Github.admin.infra` `Jira.full` `Slack.full`

J **Jordan**
Junior engineer
• Startup customer

Their tool surface
`Pagerduty.read` `Github.read.3repos` `Jira.project.acme` `Slack.team`

S **Sam**
Internal SRE
• Your own team

Their tool surface
`Linear.full` `notion.runbooks` `Monitoring.internal` `Slack.internal`

THE REALITY WITH DIRECT MCP

The same flat surface. *For everyone...*

M Maya

Senior platform engineer

- Enterprise customer

Their tool surface

`Pagerduty.admin` `Github.admin.infra` `Jira.full` `Slack.full` `Jira.project.acme` `Github.read.3repos` `Linear.full` `notion.runbooks` `Monitoring.internal` `datadog.alerts` `Sentry.issues` `+31 more outside their scope`

J Jordan

Junior engineer

- Startup customer

Their tool surface

`Pagerduty.read` `Github.read.3repos` `Jira.project.acme` `Slack.team` `Pagerduty.admin` `Github.admin.infra` `Jira.full` `Jira.delete` `Linear.full` `notion.runbooks` `datadog.alerts` `+31 more outside their scope`

S Sam

Internal SRE

- Your own team

Their tool surface

`Linear.full` `notion.runbooks` `Monitoring.internal` `Slack.internal` `Pagerduty.admin` `Pagerduty.read` `Github.admin.infra` `Github.read.3repos` `Jira.full` `Jira.project.acme` `Slack.full` `+31 more outside their scope`

WHAT THE MODEL ACTUALLY SEES

Every request. Every user. The same 42 tools.

Regardless of who's asking

● EXAMPLE: INCIDENT TRIAGE · ONE WORKFLOW

Triage open incidents assigned to this user. Pull from PagerDuty. Cross-reference recent Jira tickets and GitHub commits. Summarize and post to Slack.

3 tools called · 39 tools loaded but never touched

WHAT THE MODEL RECEIVES FOR THIS ONE TASK

User question + system prompt	~600
Tool call results (3 systems)	~3,500
Model's response	~500

Tool schemas — every server, every tool, every user

~11,200 tokens

42 tools loaded · 3 used · same surface for all

~15,800
TOKENS PER WORKFLOW

~4,600
TOKENS OF ACTUAL WORK

IT GETS WORSE

Every reasoning step pays *for the full surface again*

An agent calls the model once per reasoning step - not once per workflow. Each call sends the full current context. Every call re-bills the full flat surface.

CALL 01	Tool schemas + query — model decides to call PagerDuty	~11,800
CALL 02	All above + PagerDuty output — model decides next	~12,600
CALL 03	All above + Jira output — model decides next	~13,800
CALL 04	All above + GitHub output — model writes summary	~15,300

Flat MCP calls billed 4x for a three-tool workflow. A 10-step workflow pays for 10×.

~53,500

THE GAP

MCP has the security
CLI has the efficiency

How do we get both?

CLOSING THE GAP

Bringing CLI-level efficiency to *MCP without losing the security model*

Just-in-time tool loading

01 · on demand

Tools enter the model's context only when actually needed — not preloaded from every connected server. No catalog, no upfront cost, no irrelevant schemas.

→ Eliminates the flat-surface bloat

Intent-based filtering

02 · semantic

The gateway intercepts the query and does semantic matching against indexed tool descriptions — before the model sees anything. Progressive disclosure as the task evolves.

→ The right tools, not all tools

User-scoped access

03 · per-user

The tool surface reflects who's asking — their identity, organization, permissions. Same agent, different users, different surfaces. Every time.

→ Ends the flat-surface problem entirely

Result scoping

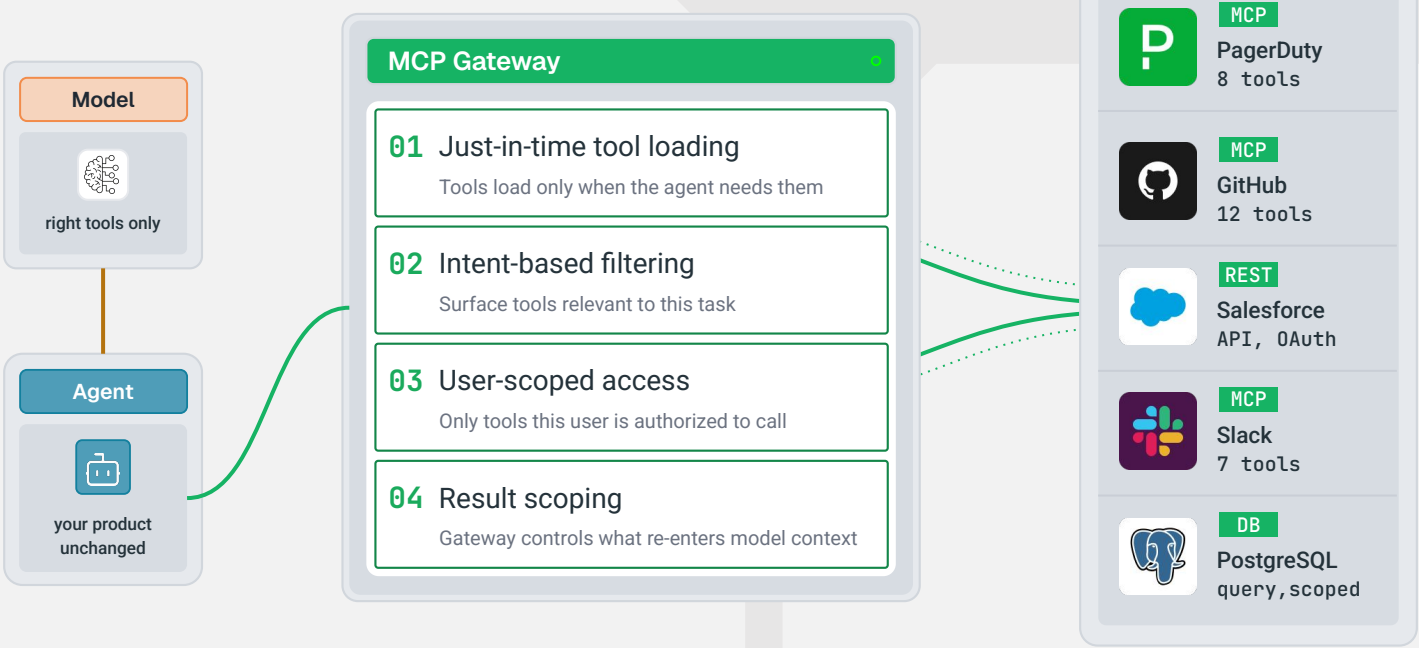
04 · return path

The gateway is in the return path, not just the outbound path. It decides how much of each tool result re-enters the model's context — summarizing, truncating, or holding results externally.

→ Cuts the per-step compounding directly

SOLUTION 1 • THE GATEWAY

One layer between your agent *and every enterprise system.*



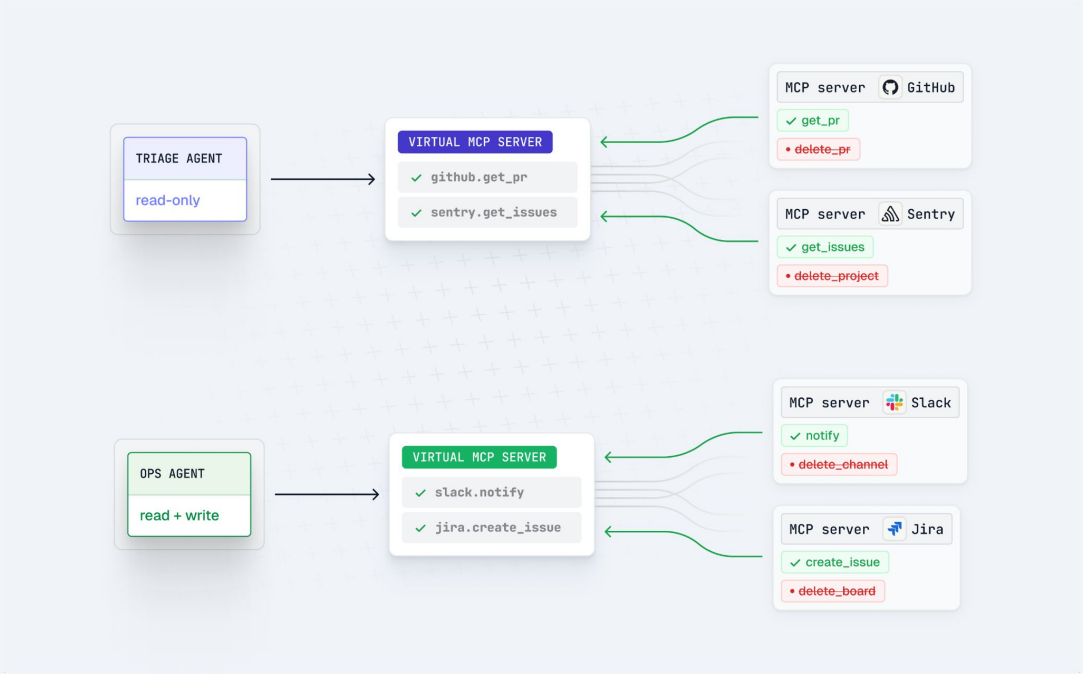
A middleware between your agents and enterprise systems like **SaaS APIs, MCP servers, databases, internal tools.**
All normalized, all scoped to the user making the request. Every request goes through one enforcement point.

SOLUTION 2 • VIRTUAL MCP

Each agent gets its own server.

and every enterprise system.

Compose a virtual MCP server for each agent role — pull exactly the tools it needs from any MCP, and leave the rest behind. At runtime, every call is scoped to the user the agent is acting for.



AT RUNTIME

01 Scoped to the role

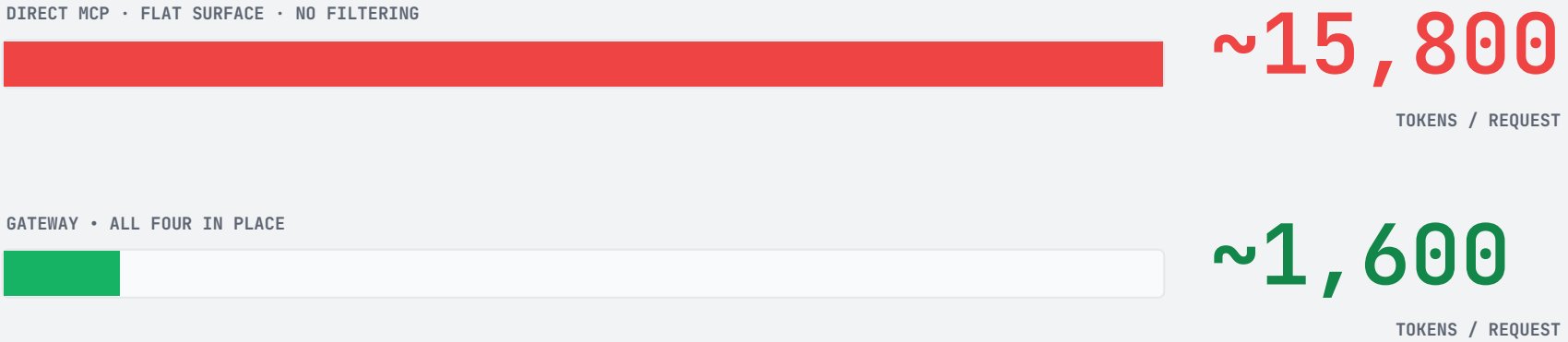
Pull exactly the tools the agent needs. What you leave out never reaches the model.

02 Scoped to the user

Every call bound to the user's own connected accounts.
No shared creds.

THE MEASURABLE CONSEQUENCE

What the right architecture does to your *unit cost*



A ~90% reduction — on this workload (42 tools • 5 servers • 3-tool workflow) - not because of optimization.
Because the gateway surfaced the right tools for the users on every request.

The right tools.
The right user.
Every time.

on demand | filtered by intent | scoped by identity

Ravi Madabhushi
CTO, Scalekit

www.scalekit.com | github.com/scalekit-inc