MCP

Beyond 1:1 Mapping

Designing MCP for real enterprise systems

From API gateways to agents: what to expose, and for whom.

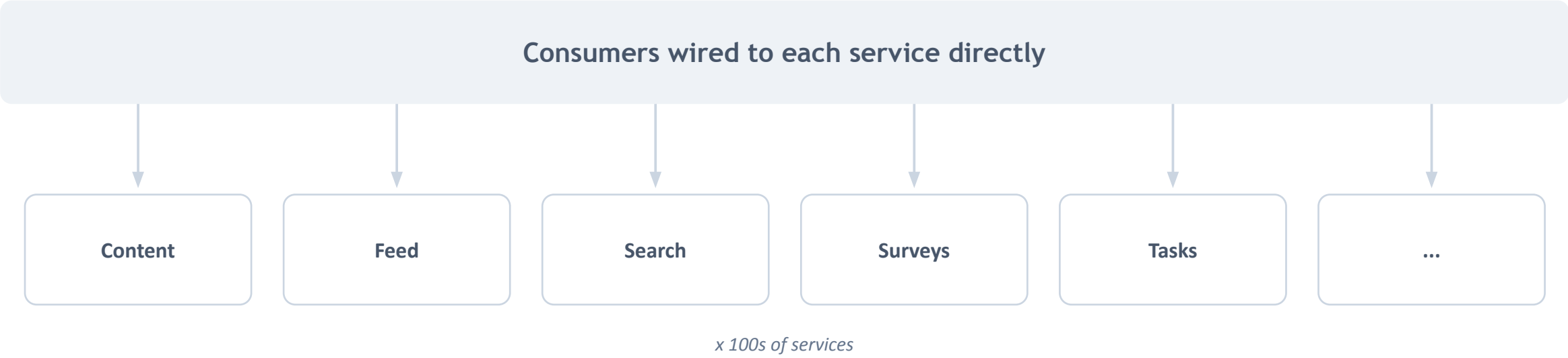
Naresh Waswani
Senior Architect, Simpplr Inc.

Jyoti Notani
Architect, Persistent Systems Ltd

THE PATTERN

You have solved this problem before

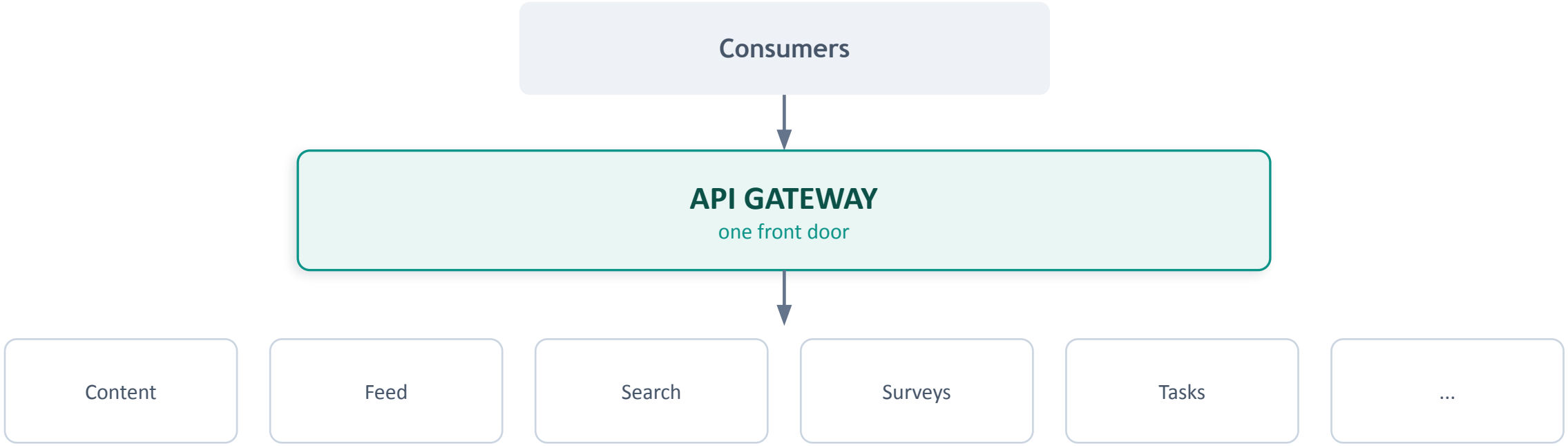
We broke the monolith into hundreds of services. The first instinct was to let every consumer call each service directly.



Each consumer had to know where every service lived, how it authenticated, and which version it ran. It did not scale.

THE PATTERN

So we converged on one front door



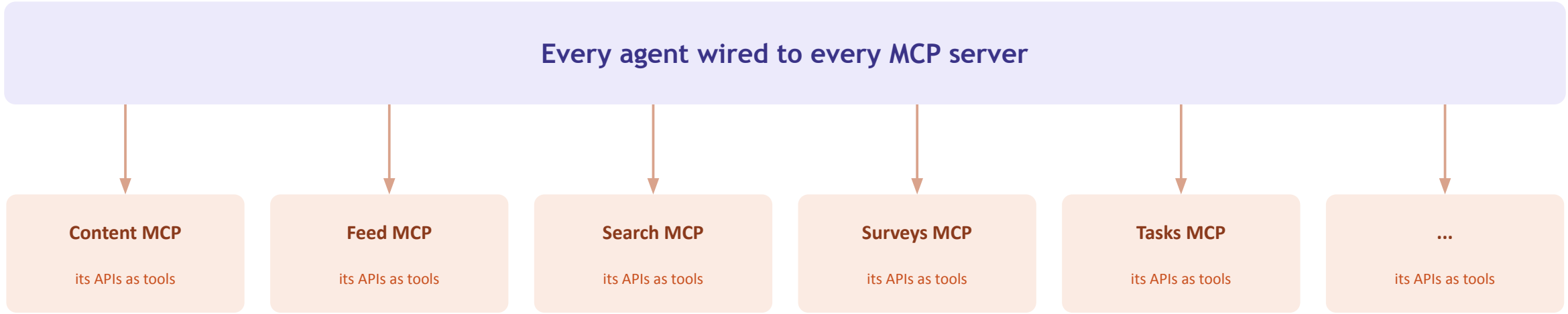
Authentication, routing, rate limiting, versioning, all in one place.

The services behind it became an implementation detail. No consumer needed your internal map.

THE REPEAT

Then agents arrived, and we forgot

One MCP server per microservice. Every API becomes a tool.



We rebuilt the pre-gateway world. Hundreds of front doors, again.

WHAT BREAKS · 1

Where one tool per API breaks down

Too many choices

Hundreds of tiny tools is a phone book when the agent wanted a menu. It picks the wrong one, invents parameters, or misses the obvious option.

Named for your plumbing

Tool names describe your internals, not the user's goal, so the agent has to reverse-engineer how your company is wired.

Slow and expensive

Before answering anything, the agent reads the description of every tool it might use. Every single time.

An operations burden

Hundreds of servers to run, secure, version, and patch. Nobody does that hygiene a hundred times, so keys leak and servers rot.

A bigger blast radius

Hundreds of unlocked side doors instead of one guarded entrance, with nowhere central to see who did what, set limits, or shut things off.

Your wiring goes public

Outsiders' agents learn your internal map, and every internal API becomes a public promise you cannot change without breaking someone.

None of this is the protocol's fault. It is the cost of copying the wrong boundary.

WHAT BREAKS · 2

And now you run it, and secure it

Operational

53%

of 5,200 MCP servers analyzed used long-lived static secrets. Just 8.5% used OAuth.

Nobody does credential hygiene a hundred times.

Security

Cross-tenant exposure · 2025

A widely used SaaS platform launched an MCP server. Within weeks, one tenant could read another's data (~1,000 customers). User permissions held; tenant isolation in the new layer did not.

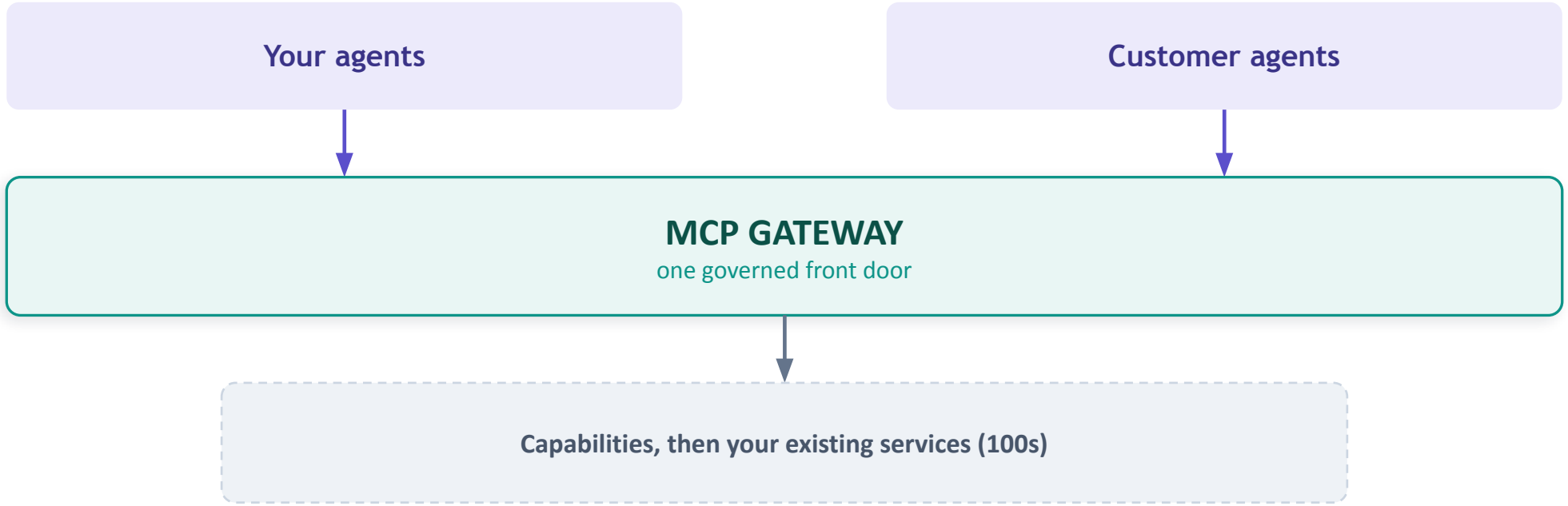
A poisoned MCP package · 2025

A community server, trusted after months of clean releases, shipped a hidden backdoor that copied every email it sent to an outside address.

New protocol. Same fundamentals. More front doors means more attack surface.

THE ANSWER

The answer rhymes: one front door for agents



To any agent, the gateway is your MCP server.

Everything behind it is ordinary internal traffic. MCP appears once, at the boundary.

THE ANSWER

What the gateway owns

Authentication

OAuth 2.1, audience-bound, no passthrough

Multi-tenancy

tenant from the token, isolation on every path

Catalog scoping

only the tools this agent may use

Rate limits

per-tenant quotas and throttling

Audit

every call, by agent and tenant

Versioning

tools as public contracts, plus routing

THE CATCH

The API gateway mostly passed requests through. Build the MCP gateway the same way, one tool per endpoint, and you have just moved a hundred doors six inches back.

The gateway answers where to route, and whether you are allowed. It does not answer what should exist.

It taught us topology. Not design.

So before we design a single tool: who is going to consume it?

WHAT TO EXPOSE

The consumer is not a developer

A developer

your API gateway's consumer

Reads your docs.

Holds the service map in their head.

Stitches many calls together.

More endpoints is fine.

A model

your MCP consumer

Cannot read your docs.

Does not know your service map.

Gets measurably worse with more tools.

So do not expose your architecture. Expose its tasks.

WHAT TO EXPOSE

Tools should speak intent, not implementation

MIRRORING THE ENDPOINTS

- `search_confluence`
- `search_sharepoint`
- `search_internal_kb`
- `list_connectors`
- `get_connector_status`

5 tools. The agent must learn your topology.

ONE TASK-ORIENTED TOOL

`search_knowledge(query, scope?)`

Fans out across the KB and every connector, applies ACLs, ranks and de-dupes, returns one result set.

1 tool. The agent knows what it can do, not how.

WHAT TO EXPOSE

Group tools by capability, behind the gateway

One capability server per domain, each aggregating whatever services it needs behind the task.

Knowledge

`search across KB + connectors`

behind it: many services

Content

`create / find / count`

behind it: many services

Recognition

`recognize colleagues`

behind it: many services

Engagement

`feed + newsletter`

behind it: many services

Tasks

`create / list / close`

behind it: many services

Surveys & Forms

`find / submit`

behind it: many services

Hundreds of services collapse into a handful of capabilities the agent understands.

THE NUANCE

So when are low-level tools the right answer?

Primitives fit when...

the consumer is a general, capable agent (a coding copilot)
the primitives are the user's mental model (issues, files, branches, PRs)
flexible composition beats a fixed workflow

```
GitHub / Jira: create_issue, get_file, create_pull_request
```

Outcomes fit when...

the consumer is a purpose-built agent
the domain spans many services
users think in results, not operations

```
HR assistant: search_knowledge, find_pending_surveys
```

THE RULE

How to decide what to expose

1

Default to task-oriented tools

Group them by capability. Name the outcome, hide the services.

2

Primitives only when earned

The consumer is a capable agent and the primitive is the unit of intent.

3

Either way, govern it

One gateway, scoped and audited. Curate, do not auto-generate. Keep the surface small.

TAKEAWAYS

Five things to take home

1. You solved exposure once, with the API gateway. Do not rebuild a hundred front doors in a new protocol.
2. The MCP gateway is that front door for agents: identity, isolation, limits, audit, versioning, in one place.
3. The gateway solves topology. It never decides what to expose.
4. Design for the consumer. It is a model, not a developer: tasks, not endpoints, grouped by capability.
5. Primitives only when the consumer is an agent and the primitive is the intent. Otherwise, outcomes.

Agents are not clients of your services.
They are customers of your product.

The API gateway taught us how to expose. The MCP layer is about what to expose, and for whom.

Thank you. Questions?