JUN 24, 2025

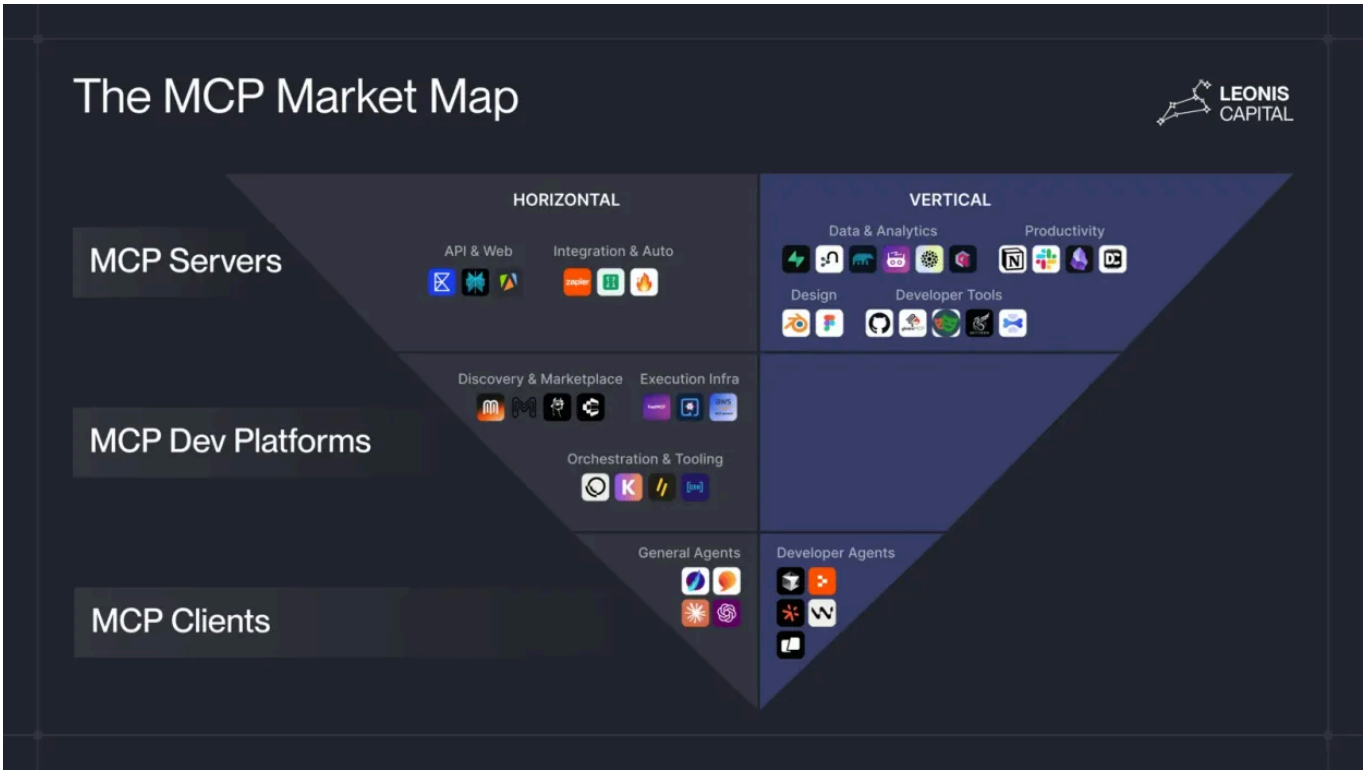
# MCPs: Value Creation, Capture, and Destruction—Lessons from the API Era

Why Most MCP Startups Will Die (and Who Won't)



By Jenny Xiao, Liang Wu, and Jay Zhao

Leonis Capital · leoniscap.com/research/mcps-value-creation-capture-and-destruction-lessons-from-the-api-era/




## MCPs and the API Parallel

When Anthropic launched the Model Context Protocol (MCP) in late 2024, it struck a chord across the developer community. At last, a solution to the spaghetti mess of AI integrations. MCP proposed a clean, standardized way for AI systems to communicate with external services. What APIs were to the web, MCPs are to AI agents.

Developers jumped in fast. Open source MCP servers appeared for databases, productivity tools, and internal developer infrastructure. Founders started pitching agent-native companies built on top of MCPs. As investors, we're seeing the beginning of a new stack, with a new category of companies pitching agent-first futures unlocked by MCP architectures.

We've seen this before. The API wave of the 2010s promised seamless interoperability across the web. Investors rushed into horizontal API platforms, hoping to own the connective tissue. But most API companies failed. The ones that endured, like Stripe, Twilio, Auth0, Marqeta, Plaid, won because they owned regulatory or domain-specific complexity, not because they exposed endpoints.

That era taught us something fundamental: **once the connection layer is standardized, value doesn't live there.** It moves *upstream* to infrastructure providers who power the system, and *downstream* to vertical specialists who solve hard, domain-specific problems. The middle layer that acts as the glue gets squeezed. The same will happen with MCPs. Standardization creates new opportunities, but it also kills entire categories of shallow integration businesses. Some will mistake protocol adoption for defensibility. They'll be wrong.

In this essay, we explore what MCPs are, how they differ from past integration approaches, why most API-first startups died, and what that means for the next generation of AI-native infra. We'll map where value will be created, who is positioned to capture it, and how to avoid repeating the mistakes of the API era.

### A Technical Primer: What are MCPs?

MCP defines a standard way for AI agents to talk to services, covering three core components: (1) **tool discovery**, i.e, discovering what a service can do, (2) **command parsing**, i.e., turning natural language into precise commands, and (3) **standardized response formatting**, i.e., formatting responses in a predictable way. On the surface, this might sound like API standardization all over again, but the key difference is that MCPs go deeper than API standards: not only describing syntax, but capturing ***semantic intent as well***. It doesn't just describe endpoints, it lets AI systems actually understand and use them out of the box.

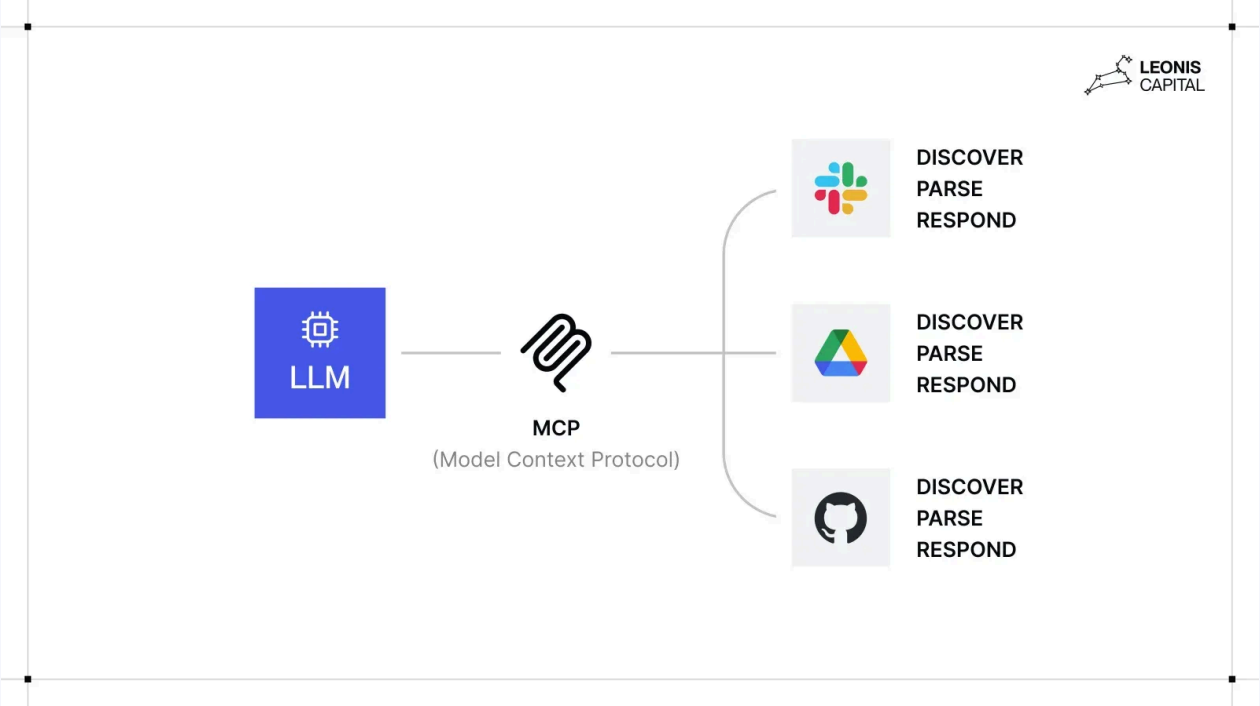



*Figure 1: What APIs did for apps, MCP does for agents—standardizing how LLMs interact with external services.*

In the API era, developers had to read long documents to understand what an API could do, then manually build everything around it (auth flows, error handling, retries, health checks). Connecting an AI system to something like HubSpot required writing custom auth flows for the system, translating HubSpot's unique data formats, and handling its quirks

> manually, one integration at a time. Tools like Swagger helped describe the endpoints, but they didn't explain how to use them reliably. Every team reinvented the same wheel.
>
> MCP turns that mess into a plug-and-play standard. It defines how the AI system discovers available actions (tool discovery), understands natural language commands ("find all contacts from last month"), and receives consistent, usable responses, across *any* service. Things like authentication, error handling, and discovery are built into the protocol itself, so developers don't waste time rebuilding common logic. What's being protocolized is the entire chaotic process of AI-to-service integration.

## The Rise and Missteps of APIs

To understand where MCPs are going, it helps to study the API wave that came before them and why most API startups failed while a few captured outsized value. **The history of APIs serves as a warning for similar structural traps that are already appearing in the MCP ecosystem.**

### Historical Context: The Dawn of REST APIs

Before 2010, APIs were mostly SOAP-based, cumbersome to integrate, and primarily used by enterprises for B2B integrations. Startups had limited access to external services, forcing them to either build everything in-house or rely on ad hoc partnerships with other companies. VCs paid little attention to "API plays" because there wasn't a clear business model. APIs were seen as technical plumbing rather than standalone products that could generate meaningful revenue.

Then REST/JSON arrived, and everything changed. APIs got lighter, cleaner, and easier to adopt. Twitter's API (launched in 2006, but widely adopted after 2010) became the archetype: a public JSON endpoint any developer could tap into. Suddenly, solo devs could build on top of powerful external systems without enterprise resources. This unlocked the API-as-product era (2010-2013). Stripe (2011) made payments accessible with just a few lines of code. Twilio (IPO in 2016) turned telephony into a developer tool. Marqeta (founded in 2010, IPO in 2021) transformed card issuance, turning a months-long bank negotiation into a programmable API. These companies demonstrated that APIs could be more than technical infrastructure and serve as the foundation for entirely new business models.

## The Horizontal vs Vertical Split: A Tale of Two Outcomes

As APIs matured into products, two strategic approaches emerged: **horizontal APIs** that offered general-purpose infrastructure (routing, auth, transformation), and **vertical APIs** that embedded deeply into specific domains (payments, fintech, security). Horizontal plays promised ubiquity and scale. Vertical plays promised defensibility and margin.

## Why Horizontal APIs Looked Like Gold, But Mostly Flopped

Horizontal API startups dominated early VC interest. Their thesis was to become the universal "plumbing" layer for the internet. Every app, from Uber to GitHub, would route through your infrastructure. Capture 0.01% of every transaction and build a massive business.

But reality hit hard.

**First, cloud providers undercut them.** AWS, GCP, and Azure treated API routing and management as loss leaders to drive compute usage. AWS could charge $0.01 per million requests, leaving virtually no margin for independent API gateway companies. When AWS released Lambda integrations and comprehensive call tracking, some API startups found their core functionality replicated by a platform with deeper pockets and tighter integration with existing infrastructure. For example, AWS API Gateway, launched in 2015, directly competed with companies like Apigee and Mashery by offering lower pricing, tighter integration, and better distribution.

**Second, open source destroyed pricing power.** Kong, Postman, Swagger UI, Insomnia, and ReDoc provided free or community-driven alternatives that made it harder for proprietary API platforms to charge for basic functionalities.. These tools created competitive pressure that eroded margins across the API ecosystem.

**Third, horizontal APIs had no moat.** No network effects. No lock-in. No real switching costs. A better, cheaper tool would win every time. Developers could swap out a generic API gateway or routing layer if/when a better alternative emerged. It was a race to the bottom on both features and pricing.

## Why Vertical APIs Built Real Moats

The few API companies that *did* break out shared one thing: **they owned complexity in a vertical and/or specific domain that couldn't be easily copied**.

Stripe doesn't just process payments; it handles PCI-DSS compliance, global payouts, disputes, and chargebacks. No competitor could replicate that regulatory and operational complexity

overnight. The deeper it went into the payments ecosystem, the more irreplaceable it became.

Auth0 and Okta are embedded deeply into enterprise security stacks. Their products weren't just APIs; they were also compliance enablers, tied to SOC 2, SAML, audit logs, and legal requirements. That created real switching friction. And the compliance and audit requirements created natural barriers to entry, protecting these companies from commoditization. Plaid and Yodlee succeeded in fintech because they handled not just data aggregation, but also messy bank integrations, licensing, and trust. Their moat wasn't tech so much as **access** and **credibility**.

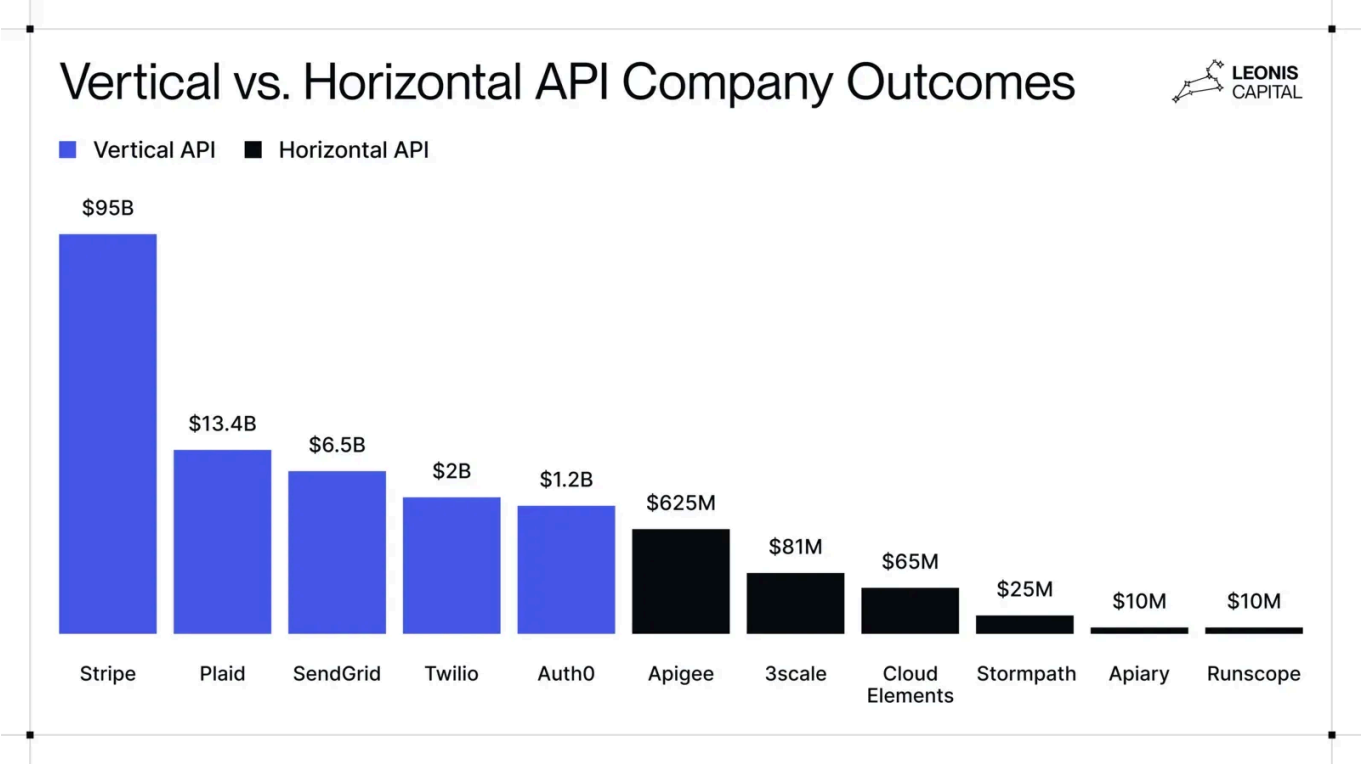



*Figure 2: Vertical APIs won by owning hard problems; horizontal APIs mostly got commoditized.*

This pattern is reflected starkly in the companies' outcomes. Vertical API outcomes were **orders of magnitude larger,** with Stripe alone reaching a $95B valuation. These companies captured value by owning messy, domain-specific workflows. Horizontal platforms, by contrast, built interchangeable plumbing and saw much smaller exits.

## The API Gold Rush: What VCs and Founders Got Right / Wrong

The early breakout success of Stripe, Twilio, and Plaid triggered a gold rush. Suddenly, "API-first" became shorthand for scalability, developer adoption, and high-margin usage-based revenue.

Why build a full SaaS product with all the UI, support, and sales complexity when you could offer a clean API and charge per call? The unit economics seemed compelling as well. APIs could reach technical users directly, charge based on actual usage, and scale revenue with customer success.

Investors piled in. Seed and Series A rounds flowed into anything with an endpoint. YC backed a wave of API-first startups. Twilio went public at ~40x forward sales. Stripe became a generational private-market darling. Founders pitched APIs as network-effect businesses with the logic that if you make integration easy, you would become the default layer in every app.

The thesis sounded great on paper, but it missed three core truths.

**First, most APIs are just features, not standalone businesses.** Developers integrate them, but don't stick around out of loyalty. Most API companies discovered that **their supposed network effects were actually just first-mover advantages** that evaporated once larger players entered the market with superior economics.

**Second, the cloud giants weren't asleep.** AWS, Microsoft, and Google saw the traction and made the strategic move to play a pricing game that they could afford, that the startups couldn't. Even as an established API company, Twillio warned investors about Amazon undercutting its SMS and voice business with AWS Pinpoint. Entrepreneurs and investors underestimated the cloud providers' willingness to lose money on the API battle to win the infrastructure war.

**Third, VCs underestimated how quickly the "infrastructure" layer would commoditize.** The more APIs standardized, the easier it was for developers to switch, and the harder it became to maintain pricing power. Most horizontal API startups either flamed out or ended up being sold for small returns.

The result was predictable in hindsight: the winners in the horizontal layer were infrastructure behemoths rather than the API startups that pioneered the space. AWS, Microsoft, and Google captured the majority of the value that API standardization created, not pure-play horizontal API startups. Meanwhile, the vertical API companies that survived became some of the most valuable software companies of the 2010s, proving that domain expertise created lasting competitive advantages that even cloud giants couldn't easily replicate.

## How Standards Commoditize and Redistribute Value

Standards don't eliminate value; they **shift it elsewhere in the stack**. When a standard emerges, it flattens the connectivity layer, making integration cheap and interoperability ubiquitous. That just shifts the value **upstream to those who provide scale and reliability, and downstream to those who solve domain-specific problems that the standard doesn't.** This pattern is the key to understanding why most API companies failed and why most MCP startups will, too.

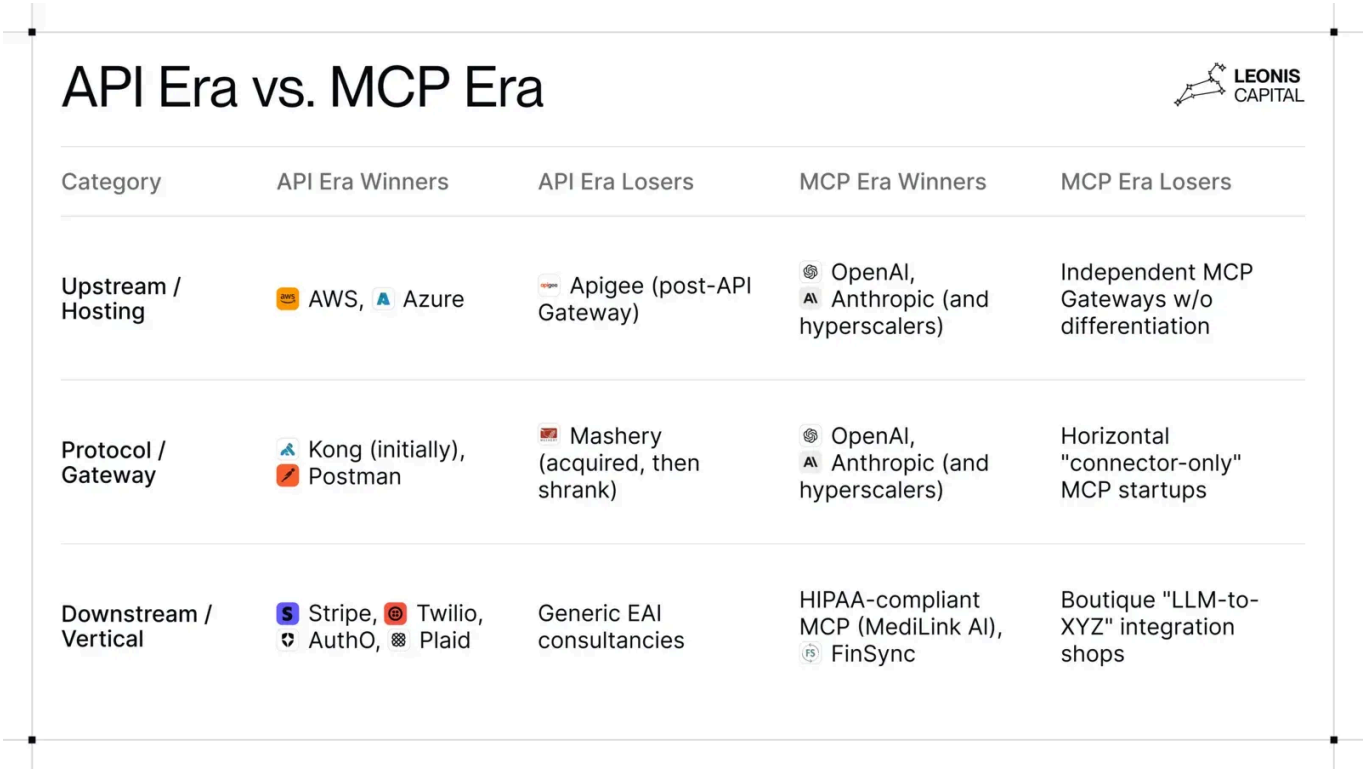



*Figure 3: Like in the API era, MCP value concentrates at the top and bottom of the stack—middleware gets commoditized.*

We've seen this play out before.

When TCP/IP became the standard networking protocol in 1983, it commoditized proprietary networking stacks from companies like IBM and Novell. The protocol itself became free, but value moved upstream to Cisco (routers) and ISPs (access control), and downstream to Netscape and the early web (applications). HTTP and REST APIs followed the same pattern. Once every developer could call an endpoint with a few lines of code, integration became cheap. The value shifted upstream to cloud providers like AWS that monetized infrastructure and throughput, and downstream to vertical players like Stripe and Twilio, who tackled vertical complexity.

The value destruction was equally predictable. Enterprise middleware and EAI (Enterprise Application Integration) vendors like Tibco, IBM BizTalk, and Informatica lost their high-margin integration suites once simple REST tools became widely available. Custom integration consultancies that had charged $100k-$500k for bespoke projects saw their business models evaporate as developers could wire services together in days using open-source tooling.

The same story exists outside software.

Shipping containers standardized freight, destroying value in midstream labor (port crews, custom loaders), while upstream players like Maersk captured scale advantages in container

logistics, and downstream players like FedEx monetized last-mile optimization and software-driven tracking. Credit card networks like Visa and Mastercard standardized the transaction layer (commoditizing direct bank integrations), but captured upstream value by controlling the rails and taking a cut of every transaction. Downstream, fintech startups leveraged this standard to offer developers easy-to-integrate payments bundled with fraud detection, compliance, and payout flows.

The pattern is consistent: fragmentation creates demand for a standard; the standard unlocks interoperability but **commoditizes the integration layer**. The total market balloons from the reduced friction. But **value capture concentrates** in the layers that remain differentiated, the **upstream players with scale,** and the **downstream players with vertical expertise**.

## Value Creation, Capture, and Destruction: MCPs vs. APIs

The transition from APIs to MCPs represents a fundamental shift in how value gets created, captured, and destroyed across the AI stack. MCPs are creating value through universal AI/agent interoperability, enabling any large language model to communicate with any service through standardized protocols. This unlocks AI-driven workflows that were previously too costly or complex to implement.

### API Era vs. MCP Era

LEONIS CAPITAL

| Aspect | Horizontal APIs (2000s) | MCP (2025-forward) |
|---|---|---|
| **Value Created** | Universal service interop, rapid product rollouts | Agent interoperability across LLMs, plug-and-play AI workflows |
| **Value Disrupted** | Custom integration shops, proprietary SDKs | Bespoke agent-to-service engineering, GUI glue code |
| **Value Capture** | Cloud providers (AWS, Azure), API management vendors, a handful of vertical survivors (Stripe, Twillo) | Orchestration layers (MCP gateway providers), certification/compliance specialists, domain-focused vertical MCP services |
| **Startup Outcomes** | 90% of API startups failed or got acquired cheaply: AWS/Azure revenues soared | Expect analogous shakeout: only domain-specialized MCP vendors survive— everything else commoditized by cloud/AI platforms |

*Figure 4: Valuation creation, capture, and disruption in the API and MCP eras.*

Value capture upstream in the MCP era flows to AI model hosts like OpenAI, Anthropic, and Google, who collect the majority of compute and inference revenue as agent usage scales. Every MCP interaction requires model inference, creating a direct revenue stream that grows with adoption. This parallels how cloud providers captured infrastructure value in the API era, but with potentially higher margins due to the computational intensity of AI workloads.

Meanwhile, new value is forming at the edge. Domain-specialized MCPs that embed compliance (for example, think "HIPAA-certified" or "SOC 2-compliant" agents) will command premium pricing by solving hard, regulated problems that generic agents can't. Observability platforms that track agent performance, cost-per-intent, and ROI will emerge as the Datadogs or Splunks of the agent world.

But just like the API era, this new standard will wipe out large swaths of the old stack. The first to go will be firms charging six figures to manually wire LLMs into enterprise systems, and companies whose moat depends on proprietary connectors. What takes them weeks today will soon take minutes with a standardized MCP server in the middle.

Generic "MCP connector" startups are next in line for commoditization. Just as cloud providers undercut horizontal API vendors by bundling basic functionality for free, OpenAI, Anthropic, and others will ship standard connectors that make undifferentiated MCP tooling obsolete.

The API era made this lesson painfully clear: selling horizontal "developer convenience" without a defensible moat leads to a race to zero. Most API startups died trying to monetize the connective tissue, while real winners embedded into verticals where domain expertise mattered. Twilio didn't survive because it exposed SMS via API; it survived because it navigated telephony regulations and carrier relationships that others couldn't.

**The same will hold for MCPs.** If your pitch is "connect agents to tools" and your edge isn't regulatory, proprietary, or deeply vertical, you're already replaceable. **MCP companies will only endure if they lock down real-world complexity that foundation model providers won't touch.**

## Where Startups Win in the MCP Stack

The emerging MCP stack mirrors a pattern we've seen before: the strategic divide between **horizontal vs vertical plays**. Horizontal companies build generic tooling that can serve many use cases; vertical companies build deep into a specific domain, solving specialized problems that horizontal players can't touch.

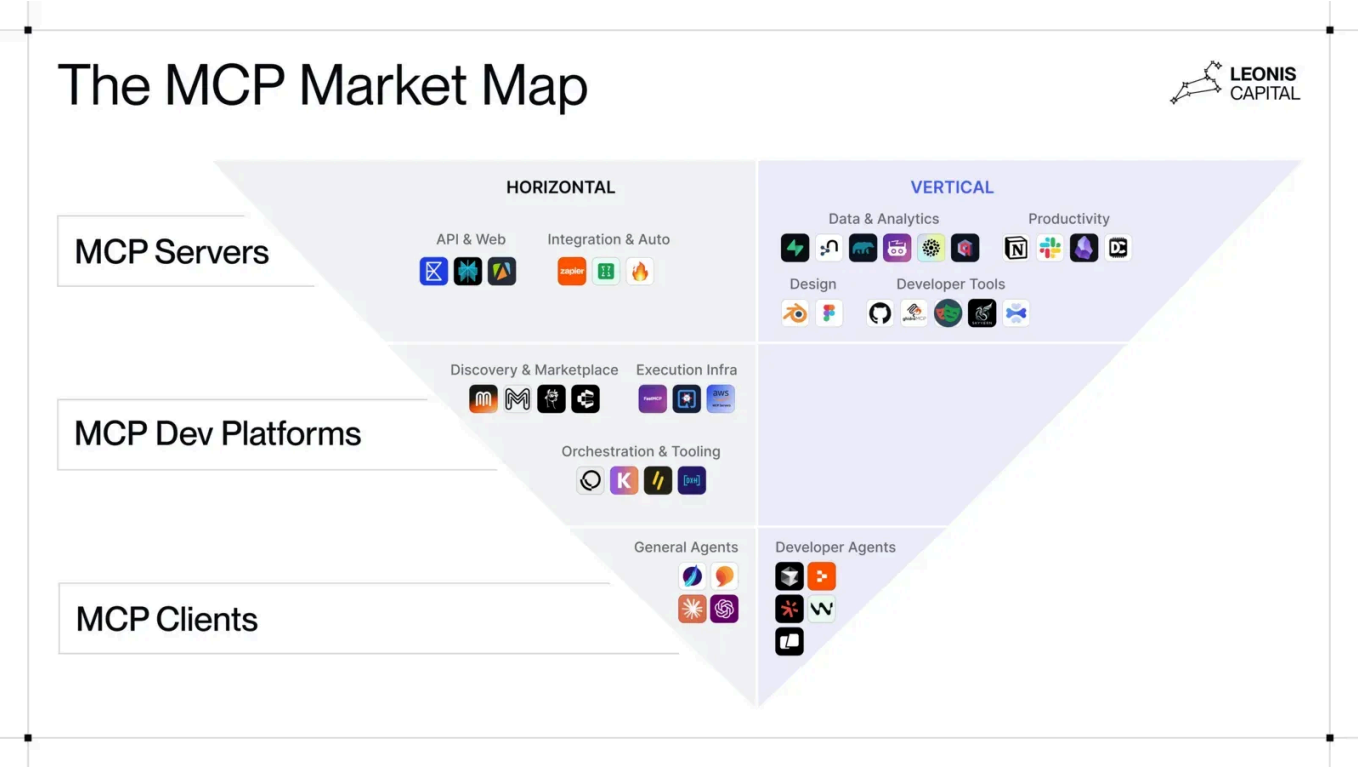



*Figure 5: The MCP market is crowded horizontally, but real defensibility lies in verticals and embedded agent use cases.*

The **tooling and orchestration layer** is where most current MCP startups are focused, but it is heavily horizontal. Companies like Glama, Mintlify, and OpenTools are building gateways, developer marketplaces, and infra tooling for MCP mostly focused on general-purpose support, not domain-specific workflows. While some of these may see success, the horizontal nature of this layer makes it vulnerable to bundling by foundation model players or cloud providers. It's unlikely that any standalone MCP tooling startup becomes a category-defining company unless it moves vertically or builds proprietary distribution.

The **client layer** faces similar challenges. Most MCP clients today are either general-purpose agents (Claude, ChatGPT) or developer-focused (Cursor, Replit, Sourcegraph, Windsurf). The space is dominated by incumbents with massive distribution and embedded interfaces. That said, there's still a wedge for **vertical MCP clients focusing on** agents built specifically for healthcare, legal, finance, and other regulated domains. But the bar for success is high, and these agents need deep domain knowledge, regulatory compliance, and go-to-market muscle.

Where startups do have a real shot is in the **vertical MCP server** layer. Here, depth beats breadth. Instead of building general-purpose connectors, startups can own defensible positions by embedding complex logic directly into the server.

But even in this layer, not all verticals are equally attractive. Many high-value categories are already dominated by entrenched players. For example, Figma, Blender, GitHub, Supabase, and Stripe have all layered in MCP support on top of their APIs. These incumbents have massive advantages: trillions of user actions, domain-specific metadata, behavioral logs, and customer trust. They don't just plug into MCP, they *become* the default MCP for their category. For example, if Airbnb launches an MCP-compatible travel planner, they instantly win the agent interface for travel bookings. A startup might ship a more elegant interface, but it won't match Airbnb's global inventory, payments, support, or regulatory footprint.

So, how should startups approach this?

The smartest path is to go **deep into a new or underserved domain**. Despite incumbent advantages, the technological platform shift creates opportunities for startups to specialize and solve problems that established companies aren't addressing effectively. Startups can also design MCP-native experiences from the ground up, unconstrained by legacy architectures. That flexibility allows them to move faster, solve edge cases, and hardwire domain-specific logic directly into the server.

In healthcare, generic FHIR API wrappers provide basic connectivity, but healthcare organizations still need extensive middleware to handle EHR vendor-specific logic, patient matching, and regulatory requirements. This creates an opportunity for a vertical healthcare MCP to embed these workflows directly.

In financial services, Plaid didn't just expose bank data; it embedded KYC, AML, PCI compliance, and fraud detection directly into its stack. The same logic applies to MCPs. For example, a financial MCP server must go far beyond endpoints and handle the domain's compliance and risk logic by default.

In legal tech, companies like Clio and Rocket Lawyer exposed basic APIs for tasks like case management and document generation, but they never tackled the full complexity of enterprise-grade legal workflows. Handling attorney-client privilege, maintaining chain-of-custody, or executing jurisdiction-specific filings requires deep legal infrastructure, features that generic APIs didn't offer. In the MCP era, a vertical legal server must embed these functions natively. Without that, you're not solving the real problem; you are just rebranding an interface.

In the MCP era, **the moat is in the mess**. Tooling will get bundled. General-purpose clients will consolidate. But vertical MCP servers represent the biggest opportunity for startups in the emerging agentic stack.

## Parting Thoughts: Follow the Value Flows

The API era provides clear lessons for different types of organizations facing the MCP transition. Those who ignore these patterns risk repeating the mistakes that destroyed hundreds of API companies, while those who adapt quickly and follow how value is shifting across the technical stack can capture a disproportionate share of that value.

**MCP enables most of this value creation** through a universal interoperability framework that allows LLMs and AI systems to talk to any service through standardized protocols, eliminating the need for bespoke tools.

**Value destruction** will hit enterprise SaaS companies particularly hard. One of the key implications of MCP is that it takes traffic away from enterprise SaaS platforms, potentially reducing them to data repositories as AI agents bypass their user interfaces entirely. Companies that built competitive moats around workflow orchestration and UIs will find themselves disintermediated to just another data layer. Similarly, horizontal integration companies will face similar commoditization from standardized protocols like MCP.

**Value capture** will shift away from the middle of the stack. Upstream, AI model providers like OpenAI, Anthropic, and Google will continue to capture value, while downstream vertical MCP plays that solve domain-specific problems that generic connectors cannot address will also capture value.

The question isn't whether this transition will happen; the economic forces are too powerful. The question is whether you'll capture value in the new landscape or be commoditized by it.

## Appendix: Strategic Dynamics Behind MCP Standardization

### MCP as Strategic Open Source

Anthropic’s decision to open-source the MCP specification is not a philanthropic gesture; it’s a strategic move to accelerate agent adoption and reinforce Claude’s positioning as the default choice for MCP-native workflows.

This follows a well-established playbook where technology leaders open one layer to commoditize adjacent value pools and consolidate power elsewhere. Examples include:

- **Android (2007)** – Google open-sourced the OS, trading margin for control over search defaults, app distribution, and mobile ad revenue.

- **AWS API Gateway (2015)** – Made publishing APIs easy, knowing AWS would benefit from increased hosting and data throughput.
- **Stripe Connect (2016)** – Stripe gave away marketplace payout primitives to drive transaction volume through its payment rails.

Anthropic is applying the same strategy by open-sourcing the protocol to reduce friction and driving ecosystem adoption, while capturing value through Claude's inference workloads and tight integration. Widespread MCP support benefits all compatible models, but **Anthropic gains first-mover perception, implementation advantage, and increased model stickiness without needing to monetize the protocol directly.**

## Will MCP as a Standard Survive?

Like all open standards, MCP faces the risk of **fragmentation** and **vendor capture**. If governance lags or competitive interests splinter the ecosystem, MCP could repeat the fate of other well-intentioned protocols.

The GraphQL story is a cautionary tale of how open standards can fracture under competitive pressure. Launched by Facebook in 2015 as a flexible alternative to REST, GraphQL promised a unified query language for APIs, but the ecosystem splintered almost immediately. Facebook's Relay introduced strict client conventions that required server-side compatibility, while Apollo pushed its own stack with proprietary directives and a custom federation model. AWS AppSync, Hasura, and Prisma layered on additional extensions, from authentication schemes to performance tweaks, none of which were interoperable. The result was a fragmented landscape of incompatible GraphQL dialects that required significant engineering effort to bridge. Instead of simplifying development, GraphQL often introduced friction, leading many teams to revert back to REST for its reliability and consistency.

If MCP governance weakens or fails to evolve quickly enough, expect forks from Google, Meta, or open-source consortia seeking to capture control. OpenAI is already exploring alternative standards that could fragment the ecosystem before it fully matures. MCP has the opportunity to unify agent-service interaction, but only if the standard stays small, interoperable, and truly open. **Half-open "standards" are not true standards and tend to fragment.**