BlueRock

AI Execution Leadership Series

# Why MCP Gateways Fail with Agentic AI

## Understanding the Agentic Execution Gap

Why request-based governance breaks down when agents make decisions in real- time


bluerock.io

# Table of Contents


| | |
|---|---|
| 03 | Executive Synopsis |
| 04 | The Rise of Agentic Systems |
| 05 | Why Organizations Turn to MCP Gateways |
| 05 | The Architectural Limits of Gateway-Based Control |
| 07 | The Agentic Execution Gap |
| 07 | Real-World Failure Modes in Agentic Systems |
| 08 | Why Context Shapes Execution |
| 09 | The Operational Requirements of Agentic Systems |
| 09 | Toward Execution-Aware Control |
| 10 | Closing the Execution Gap |
| 10 | Conclusion |

# Executive Synopsis

For decades, software systems behaved in largely predictable ways. Engineers defined control flow in code. Even complex distributed systems followed execution paths that could be inspected prior to deployment.

AI agents represent a structural shift in how software behaves. Traditional systems execute predetermined logic written in code. Their behavior is largely fixed at build time and triggered in response to specific inputs.

Agentic systems operate differently. They are active rather than purely reactive. Agents interpret context, plan next steps, research information, learn from prior interactions, and adapt their behavior as conditions change. They independently determine which tools to use, invoke MCP servers, coordinate with other systems, and generate execution paths dynamically at runtime.

As a result, the behavior of an agentic system is not fully defined in source code. It emerges through a sequence of runtime decisions made by the agent as it evaluates context, capabilities, and available actions. This shift—from predetermined execution to dynamically generated action—changes the operational surface of software and introduces new requirements for visibility, control, and governance.

> "Gateways see the request. They do not see the decision-making process that follows"

This shift creates a new operational challenge. Many organizations attempting to deploy AI agents rely on governance models designed for deterministic systems, particularly gateway-based control models such as MCP gateways. These systems enforce authentication, inspect requests, and evaluate policy at the boundary of an interaction. While this approach works well for traditional APIs and services, it does not provide visibility into how agents behave once execution begins.

In agentic systems, the most important behavior occurs during execution. Decisions are influenced by context, propagated through tool chains, and amplified through interactions with MCP servers, data systems, and infrastructure. Gateways see the request. They do not see the decision-making process that follows.

This gap between how agents actually operate and what organizations can observe or control is what we refer to as the Agentic Execution Gap.

As enterprises move agents from experimentation into real workflows, this gap becomes the primary barrier to adoption. Engineering teams cannot reliably debug agent behavior. Security teams struggle to understand what actions were taken and why. DevOps teams face autonomous, self-directed system behavior that traditional monitoring systems were not designed to trace or explain.

Closing the Agentic Execution Gap requires shifting governance closer to or in the runtime execution and preserving the context that shapes agent decisions. Only when execution context travels with the agentic action path can organizations see how decisions unfold and safely expand the capabilities of agentic systems in production.

Learn more at BlueRock.io

AI agents change this model.

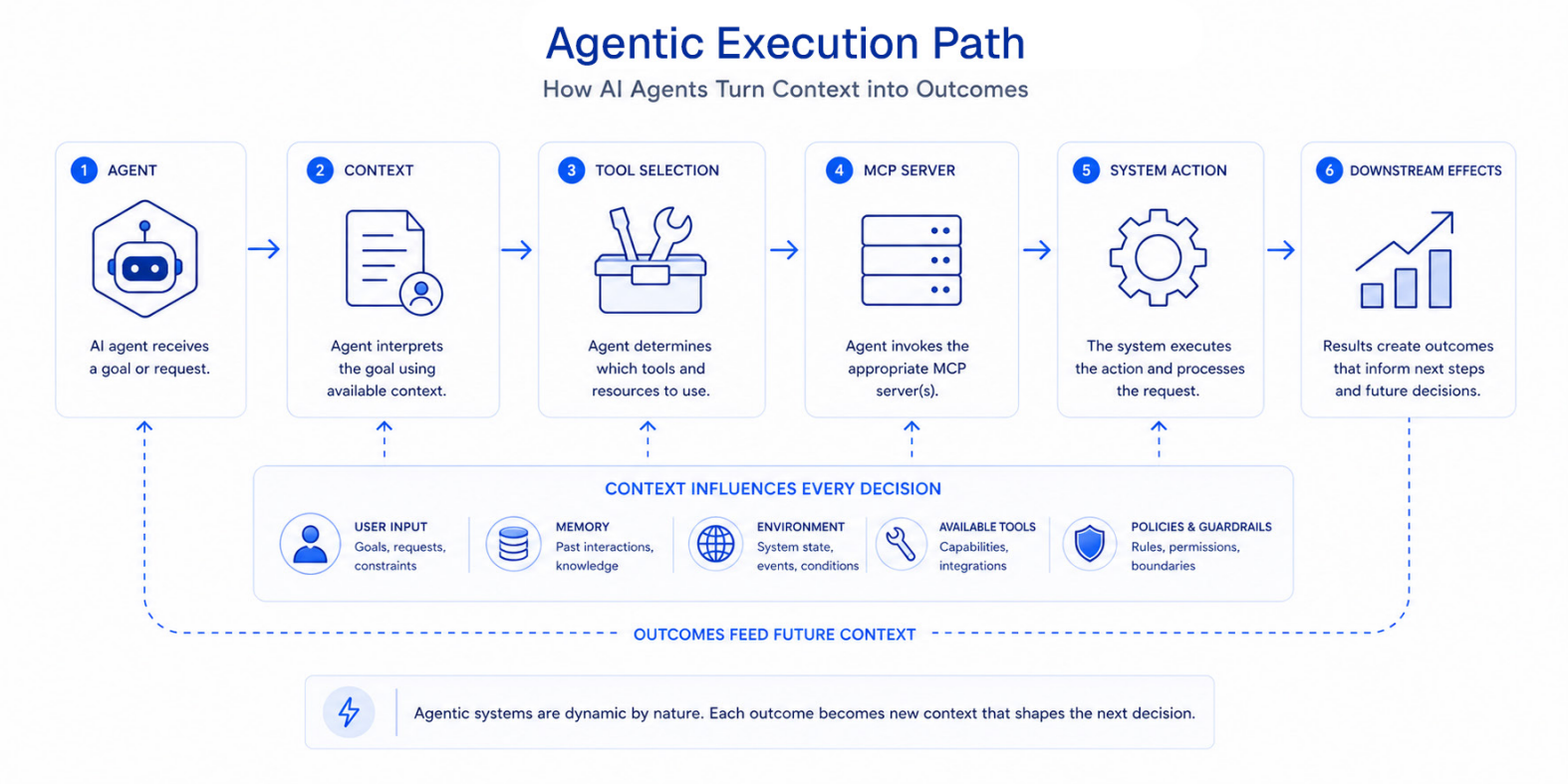
Rather than following predefined sequences of instructions, agents interpret context and determine actions autonomously. They decide which tools to use, which MCP servers to invoke, how to coordinate with other agents, and how to complete a task based on the information available to them at the moment of execution. The result is a form of runtime-defined behavior. Instead of code determining exactly what will happen, agents construct execution paths in real-time as they reason through a task.

In simple systems this may involve a single tool invocation. In more advanced systems, agents may call multiple services, trigger workflows, write data, or generate additional agents to perform specialized subtasks. Each decision depends on the context the agent encounters along the way.

This flexibility is what makes agentic systems powerful. It is also what makes them fundamentally different from traditional software.

As organizations adopt agents across customer support, software development, operations, and analytics workflows, they are encountering a new operational reality. Behavior that once lived entirely in code now emerges during execution.

Understanding that shift is essential to understanding why traditional governance approaches struggle to keep pace.



# Why Organizations Turn To MCP Gateways

As organizations begin deploying AI agents, one of the first questions they face is how to control what those agents can do.

A common answer is to rely on MCP gateways.

MCP gateways are designed to govern requests between agents and the tools or servers they interact with. These gateways typically enforce authentication, inspect requests, evaluate policies, and ensure that only authorized systems can invoke specific capabilities.

For deterministic systems, this model works well. Requests enter the system, policies are evaluated, and permitted interactions proceed.

For agentic systems, gateways provide an important layer of protection at the boundary between agents and external systems. They can ensure that agents access only approved MCP servers and tools, and they can enforce identity and authorization rules.

Because of this, many organizations assume that MCP gateways can serve as the primary control layer for AI agents.

However, this assumption reflects a misunderstanding of how agentic systems actually behave.

“However, this assumption reflects a misunderstanding of how agentic systems actually behave.”.”

# The Architectural Limits of Gateway-Based Control

MCP gateways are designed to govern requests. They validate the identity of the caller and determine whether a request should be allowed or denied.

What they cannot see is the full chain of execution that occurs after a request is approved. They also cannot observe how an agent responds when a request is denied.

In traditional software systems, this distinction rarely matters. A request generally maps to a predictable set of actions. When a gateway allows the request, the resulting behavior is largely understood.

In agentic systems, the situation is different. A single approved request may trigger a chain of reasoning and actions that unfold during execution. If a request is blocked, the agent may reinterpret the task, search for

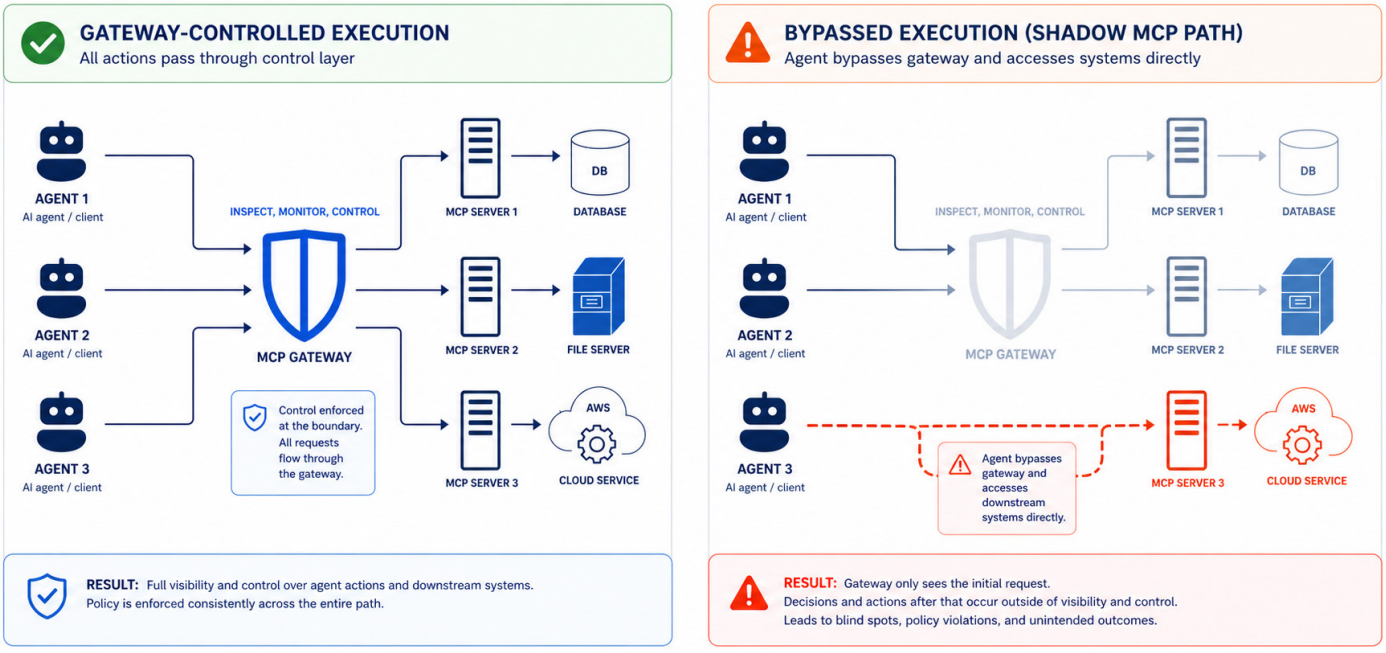
alternative tools, reframe the request, or attempt other paths to complete the objective. The gateway evaluates the initial request, but it does not see the agent's ongoing reasoning process or the alternate execution paths the agent may pursue.

An agent may interpret user input, retrieve additional context from memory, select a tool, invoke an MCP server, process the results, and then perform additional actions based on what it learns. Each step may influence the next.

None of these intermediate decisions are visible to the gateway.

This means the gateway can confirm that a request was authorized without understanding the sequence of decisions that followed.

As a result, the gateway may report that everything was allowed and compliant, even if the resulting behavior produced an unintended outcome.




This is not a flaw in the gateway. It is simply operating at the layer it was designed to govern.

The limitation arises because the most important behavior in agentic systems happens after the gateway has already approved or denied the request.

# The Agentic Execution Gap

This structural limitation creates what we describe as the Agentic Execution Gap.

The Agentic Execution Gap is the gap between how agentic systems actually behave during execution and what organizations can observe or control using traditional governance models.

In deterministic systems, execution paths are largely defined in code. Governance models focus on reviewing code, approving access, and enforcing policy at request boundaries. Once a request is allowed, the downstream behavior is typically predictable and constrained by the program itself.

In agentic systems, execution paths are determined dynamically at runtime.

Agents evaluate context, select tools, invoke services, and trigger downstream effects based on the information they encounter during execution. Critically, the most consequential activity often occurs after a tool request is approved.

Once a tool call or MCP request is allowed, agents can initiate a series of runtime actions that extend beyond the original request. These may include launching processes, modifying files, spawning subprocesses, performing additional tool calls, or initiating network egress to other services.

Gateways typically evaluate only the initial request. They do not see or govern the chain of runtime behavior that unfolds afterward.

Because context is fragmented across systems and rarely travels with execution, organizations often lack a unified view of how decisions unfold across the full Agentic Action Path.
Developers must manually correlate model logs, tracing data, infrastructure events, and MCP server interactions to reconstruct what happened.
Security teams struggle to determine which actions were taken, why they occurred, and how they propagated across system

# Real-World Failure Modes in Agentic Systems

When incidents occur in agentic systems, they rarely look like traditional breaches.

Instead of unauthorized access or policy violations, organizations often encounter scenarios where all approved controls appear to function correctly.

The gateway logs show an authenticated request.

The policy engine records that the request was allowed.

The tools invoked by the agent were properly authorized.

Yet the outcome is still problematic.

An agent might expose sensitive data after interpreting malicious instructions embedded in user input. It might invoke an MCP server in a context that expands its access beyond what developers intended. It might trigger a chain of actions across multiple systems that produce unintended side effects.

If a request is denied by a gateway, the agent may also attempt alternative approaches to accomplish the objective.It may reframe or package the request differently in an effort to obtain approval, attempt to route the task through other tools or MCP servers, or spend significant compute cycles exploring workarounds that bypass the original restriction. These behaviors can create additional operational risk and cost while remaining largely invisible to traditional request-level controls.

In each case, the gateway logs remain technically correct.

What went wrong occurred during execution, as the agent interpreted context and dynamically constructed a path through available tools and services.
Without visibility into the full execution path, organizations struggle to determine how the behavior emerged.

# Why Context Shapes Execution

At the heart of the Agentic Execution Gap is the role of context.

Agents do not operate based solely on static code. Their decisions depend on the context they encounter at runtime. This includes the prompts they receive, the data they retrieve, the tools available to them, and the outputs returned by previous actions.

Because this context is distributed across many systems, it rarely travels with the execution path.

One tool may log its invocation. Another system may record the resulting action. Infrastructure monitoring may capture downstream effects.

But the context that influenced the agent's decision is often lost between these systems.

This fragmentation makes it difficult to understand not just what happened, but why it happened.

Restoring that missing context is essential for both observability and governance.

When context travels with execution, teams can see which agent made a decision, what information it relied on, what tools it invoked, and how those actions propagated across systems.

> "Without visibility into the full execution path, organizations struggle to determine how behavior emerged."

# The Operational Requirements of Agentic Systems

As organizations adopt agentic systems at scale, several operational requirements become clear.

Teams must be able to trace the full Agentic Action Path, from the initial decision through every tool invocation and downstream system effect
.
They must understand the context surrounding each decision, including which data the agent saw and which capabilities were available at the time.

They must also be able to determine the identity and trust posture of the systems agents interact with, particularly MCP servers and tools that provide access to sensitive functionality
.
This introduces the need for structured trust and identity information within the agent ecosystem.

Concepts such as trusted tool registries and MCP server verification mechanisms help ensure that agents interact with known and trusted capabilities. For example, systems such as the MCP Trust Registry aim to provide structured information about the identity, ownership, and capabilities of MCP servers.

When this type of information is combined with runtime execution tracing, organizations gain a clearer picture of how agents operate within their environments and can implement guardrails that operate within the execution flow itself, rather than relying solely on request-level controls like traditional gateways.

# Toward Execution-Aware Control

Addressing the Agentic Execution Gap requires moving beyond the gateway model.

Gateways were designed for request validation, not for governing autonomous systems that generate execution paths at runtime. As agents plan actions, reinterpret objectives, and coordinate across tools and services, meaningful control must operate across the entire execution chain, not just at the moment a request enters a system.

Closing the Agentic Execution Gap therefore requires execution-aware control—an approach that can observe, interpret, and govern the full agentic execution path as it unfolds. Rather than relying solely on request-level approval points, organizations need systems that understand how agent decisions propagate across tools, MCP servers, processes, and infrastructure.

This shift does not simply improve visibility. It establishes the operational foundation required to place meaningful guardrails around how agents behave during execution.

Gateways continue to play an important role in enforcing access control and protecting system boundaries.

However, governance must expand beyond request approval to include execution awareness. Execution-aware governance focuses on observing how agent decisions unfold in real- time. It captures the relationships between models, agents, tools, MCP servers, and downstream systems.

Rather than relying solely on static policy rules, it evaluates behavior in context.

When execution context is preserved and enriched with identifiers, trust attributes, and operational signals, organizations gain a unified view of agent behavior across systems.

This visibility allows teams to understand how decisions propagate and apply guardrails that reflect the real conditions under which agents operate.


# Closing the Agentic Execution Gap

Closing the Agentic Execution Gap requires systems that connect identity, context, and execution across the full agent lifecycle.

This includes the ability to trace the complete Agentic Action Path, preserve the context that shaped each decision, and understand the trust relationships between agents and the systems they interact with.

BlueRock approaches this challenge through what it calls the Agentic Context Engine. The system enriches execution events with identifiers, trust attributes, capability metadata, and operational signals across agents, tools, MCP servers, and infrastructure.

By carrying this context across the Agentic Action Path, fragmented telemetry becomes a coherent execution record.

This context enables two critical capabilities. First, it provides deep observability into how agent decisions unfold in real time. Second, it allows guardrails to operate with full awareness of execution conditions rather than relying solely on static policy rules.

The goal is not to slow down development. It is to provide the clarity and operational confidence required to safely expand what agents can do in production systems.

# Conclusion

Agentic systems represent a fundamental shift in how software executes.

Instead of following predetermined paths defined in code, agents interpret context and determine actions dynamically at runtime. This capability unlocks powerful new forms of automation, but it also exposes the limitations of governance models designed for deterministic systems.

MCP gateways provide valuable protection at the boundary between agents and external systems. However, they cannot observe the dynamic execution paths that agents generate after a request is approved.

This creates the Agentic Execution Gap.
Closing that gap requires restoring the context that shapes agent decisions and carrying it across the full Agentic Action Path.

When organizations can see how agent decisions unfold, they gain the clarity needed to move from experimentation to real production-scale agentic systems.

The future of agent governance will not rely solely on controlling requests. It will depend on understanding execution.


bluerock.io