AKEYLESS

Stop Trying to Build a Directory for Ghosts

Why the IdP-Centric Model for AI Agent Identity Is Already Failing and What to Do Instead

The Premise That Doesn't Survive Contact With Production

The prevailing argument from workforce-IdP companies and echoed by industry commentary such as the Resilient Cyber piece is that the right way to secure AI agents is to extend the same identity platform we use for human users: discover every agent, register it in a universal directory, assign it scopes, govern its lifecycle, and pull a kill switch when something goes wrong. The framing is intuitive. It is also wrong in the way that a map of a coastline is wrong when the tide is coming in twice a minute.

AI agents are not a new kind of user. They are not even a new kind of service account. The argument that they need to be onboarded, certified, and governed like human identities is a category error, and the IAM industry is about to spend a lot of money learning that the hard way.

This article makes four claims:

- Agent identities are not enumerable. The instance you want to govern does not exist yet, and by the time it does, it is gone.
- The right anchor for agent access is the workload identity that the runtime issues (cloud IAM, Kubernetes service accounts, OIDC federation), not a synthetic directory entry.
- Authorization for agents must be defined as a relationship between Auth Methods and target systems, not between named identities and scopes.
- Static RBAC and ABAC cannot contain a non-deterministic actor. The missing layer is intent-aware enforcement on every action, mediated by a gateway.

Each of these is a direct contradiction of the IdP-centric blueprint. Let's take them in order.

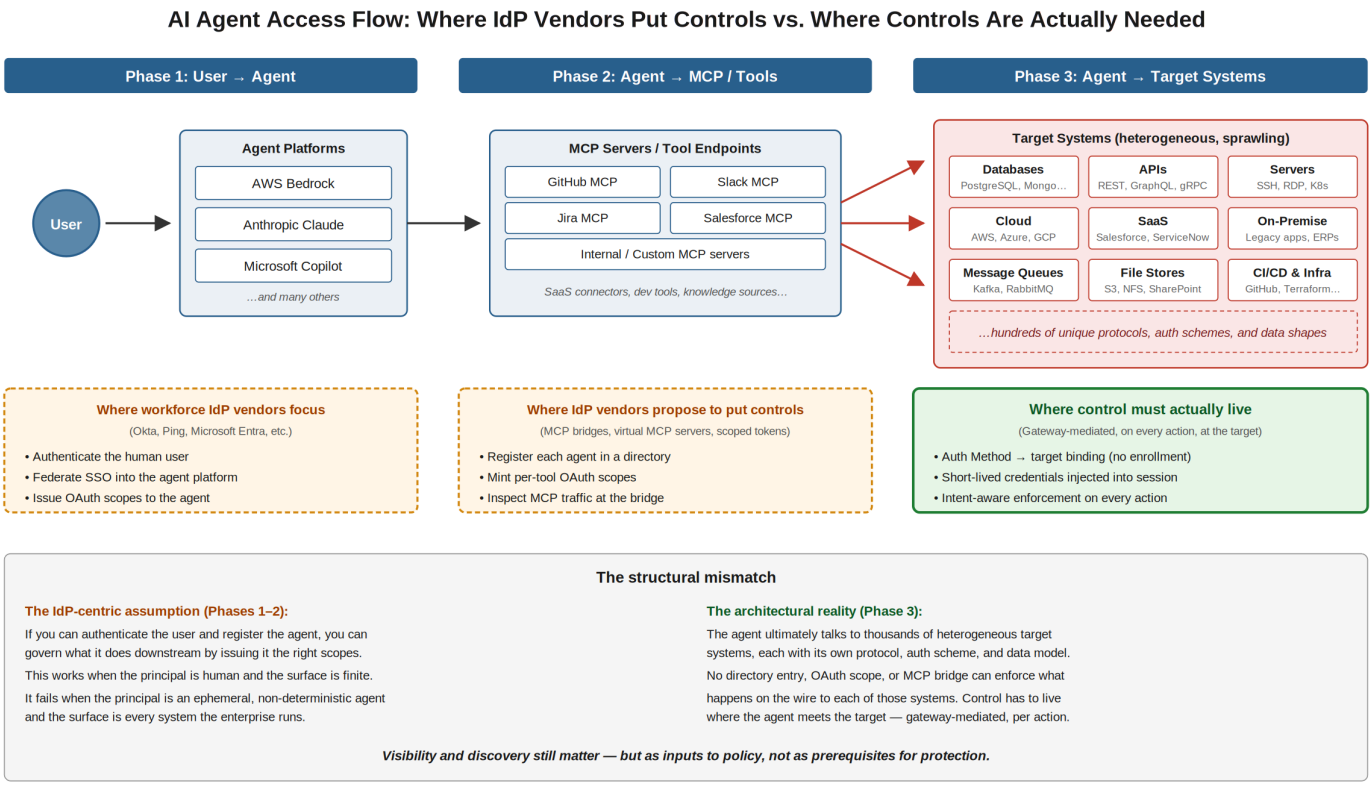

Figure 1. The three phases of an AI agent's access path, and where workforce-IdP vendors place their controls versus where enforcement actually has to live. Discovery and visibility still matter as inputs to policy, not as prerequisites for protection.

1. The Inventory Fallacy

The IdP-centric playbook opens with a familiar refrain: "You cannot secure what you cannot see." It proposes to fix the visibility problem by discovering every agent, registering it in a universal directory, and assigning it an owner, and then treating that registration as the gating step for everything that follows.

To be clear: visibility and discovery are not the problem. Customers absolutely need them. You cannot reason about what is unmanaged, you cannot tune permissions, and you cannot write meaningful policy without an honest picture of what agents are running where. Any serious agentic-access program will include a discovery and observability layer, and we build and operate one too. The question is not whether discovery is valuable. It is whether discovery is a prerequisite for protection. In the IdP-centric model it is — nothing is governed until it has been enrolled, named, and assigned an owner. In a workload-anchored, gateway-enforced model it isn't — the policy attaches to the Auth Method and the target system, so any agent presenting a valid attestation is governed at first authentication, whether or not it has ever been seen before. Discovery then feeds the policy layer; it doesn't guard it.

That distinction matters because the human-identity intuition behind the IdP model does not survive contact with how agents actually run. Directory-style inventory works for humans because humans are slow, persistent, and finite. There are a known number of employees. They onboard on Monday and offboard on Friday. They keep the same identity for years.

Agents do not behave like that. They behave like the workloads they actually are, and modern workloads have been resisting directory-style inventory management for fifteen years for excellent reasons:

- An agent can spin up on a Lambda function, run for 800 milliseconds, and disappear before any discovery scanner notices it existed.
- A single Kubernetes pod can host a dozen agents simultaneously, each with a different purpose, each living for the length of one task.
- An autonomous agent can spawn three sub-agents, each of which spawns three more, each running on a different compute substrate (VM, container, serverless, edge), and the entire chain can complete before a human takes a sip of coffee.
- The same logical agent can run as a VM today, a container tomorrow, and a serverless function next week, with no stable network identifier between them.

Treating these as entries in a directory is treating ghosts as residents. You can register them, but by the time the registration commits to the database, the entity it described is gone. The next instance with the same logical role has different network coordinates, a different ephemeral credential, and a different parent process. The directory entry is a fossil. It is not a control.

The IdP-centric vendors know this, which is why their pitch quietly shifts from “govern every agent” to “govern every agent we have managed to enroll.” The agents you can enroll are a useful subset to have inventoried, and a discovery layer that surfaces them is genuinely valuable. The problem is what happens to everything else — the ephemeral, the just-spawned, the cross-substrate, the not-yet-discovered — in a model where enrollment is the gate. The gap between “agents we have enrolled” and “agents now touching production systems” widens every quarter, and a model that leaves that gap unprotected by design is the part that has to change.

The cloud-native world hit this problem first (not for agents, but for the workloads agents now resemble) and produced a standard answer: SPIFFE. We’ll come back to it in the next section, because it is the canonical example of what a workload-anchored identity model actually looks like in production.

2. The Right Question Isn’t “Who Is the Agent?” It’s “What Issued It?”

Machine-identity practitioners solved this problem years ago. We do not provision a static identity for every EC2 instance, every Lambda execution, every Kubernetes pod. We anchor trust in the

substrate that produced them. The cloud provider's IAM service signs an attestation. The Kubernetes cluster issues a projected service account token. The CI/CD platform mints an OIDC ID token. A SPIRE server issues a short-lived SVID to a workload it has attested. The identity is whatever the runtime says it is, at the moment it is asked.

This is exactly the model AI agents need, and it is exactly the model the agentic IAM startups are arguing against, because it doesn't fit in their product.

Consider what an agent identity actually is, mechanically:

- An agent running in AWS Lambda is an execution role attestation signed by AWS STS, the same primitive we use to authenticate the Lambda itself.
- An agent running as a Kubernetes pod is a projected SA token signed by the cluster's OIDC issuer, the same primitive we use for any pod workload.
- An agent running on a workload attested by SPIRE is identified by a SPIFFE ID and a short-lived X.509 or JWT SVID, the cross-platform standard for exactly this problem.
- An agent running inside an enterprise SaaS (an embedded copilot, an AgentForce-style assistant) is authenticated by the SaaS provider's federation, and the only meaningful identity claim is the OAuth token already in the request.
- An agent running on a developer laptop is anchored in whichever endpoint identity the device already has (OIDC, certificate, device posture), not a new agent-specific credential.

In every case, the identity already exists. It is issued by the platform the agent runs on, carries the claims that platform vouches for, and disappears the instant the agent does. There is no value in copying that identity into a separate "agent directory." The directory cannot tell you anything the issuing platform does not already tell you, and it cannot enforce anything the issuing platform doesn't already enforce.

"The identity primitive for an AI agent is the workload identity its runtime already issues. Anything you build on top of that is a layer of make-believe."

This is why the right enrollment unit is not the agent. It is the Auth Method: the rule that says "any identity matching this signature, issued by this trusted authority, with these claims, is authorized to authenticate." Once you have that, you don't care which individual agent shows up. You care whether what shows up matches the policy.

Why SPIFFE/SPIRE is the canonical answer the IdP camp keeps trying to route around

It's worth being specific about SPIFFE and SPIRE, because they are the cleanest existing instantiation of the workload-anchored model and the agentic IAM hype cycle has been quietly stepping over them.

SPIFFE (Secure Production Identity Framework For Everyone) defines a portable identity for a workload, called the SPIFFE ID, along with a short-lived, cryptographically verifiable document that asserts it: the SVID, available as an X.509 certificate or a JWT. SPIRE is the reference implementation that attests workloads to their runtime (the EC2 metadata service, the Kubernetes API server, the host's kernel, a TPM, an OIDC issuer) and issues the SVID only when the attestation succeeds. The SVID lives for minutes, not weeks. It is bound to the workload, not transferable, and rotated automatically. It works identically across AWS, Azure, GCP, on-prem, Kubernetes, and bare metal.

Read that paragraph again with "AI agent" substituted for "workload" and notice that absolutely nothing about it has to change. An AI agent attested by SPIRE gets a SPIFFE ID that encodes its trust domain and its role (`spiffe://prod.acme.corp/agents/data-pipeline/summarizer`), an SVID that lives long enough to do one task and then expires, and a cryptographic guarantee that the runtime which produced it is the runtime your policy trusts. No directory entry was created. No discovery scanner had to find it. No enrollment latency was incurred. The next thousand instances of the same agent inherit the same identity primitive automatically.

This is not a future capability. SPIFFE has been a CNCF graduated project since 2022. Akeyless natively supports SPIFFE/SPIRE as an Auth Method: SVID signing, SPIRE plugins, and secretless authentication of workloads identified by SPIFFE IDs are first-class platform features today. The cross-cloud, cross-substrate, ephemeral-by-construction identity that the IdP-centric model is trying to invent out of MCP bridges and virtual MCP servers already exists, is standardized, and is in production at scale. The reason it doesn't feature prominently in the workforce-IdP blueprints is not that it doesn't solve the problem. It is that it doesn't require an IdP to solve the problem, and a vendor whose product is the IdP cannot ship that answer.

This is also why the workload-anchored model isn't just an Akeyless opinion. It is the direction the cloud-native community already went, with a working open standard, while the workforce IAM vendors were still selling agents as a new SKU on top of their human directory.

3. Authorize Auth Methods to Target Systems, Not Identities to Scopes

Here's the structural inversion the IdP model misses: when the identities you're governing are ephemeral, the only stable entities in the system are the Auth Methods and the target systems. So those are what you should be writing policy between.

Concretely:

- Define an Auth Method that accepts AWS IAM attestations from the data-pipeline execution role in the production account, with specific tag claims.

- Define another Auth Method that accepts Kubernetes SA tokens from the analytics namespace in the prod cluster, signed by that cluster's OIDC issuer.
- Define a third Auth Method that accepts SPIFFE SVIDs from the trust domain `prod.acme.corp`, restricted to SPIFFE IDs under `/agents/`.
- Define a target: the customer database in production, reachable only through the Gateway.
- Bind the Auth Methods to the target with an access role that says: any identity authenticating via these methods, carrying these claims, may request a short-lived credential to that database, scoped to read these tables, for at most N minutes.

No agent was ever registered. No directory entry was ever created. No "known agent" inventory was ever maintained. And yet, every agent that the production data pipeline ever spawns, on any compute substrate, for any TTL, is fully governed the moment it tries to authenticate. The new agent that spins up sixty seconds from now is governed by the same rule. So is the one that exists for 200 milliseconds. So are the next thousand.

This is the inversion: the unit of governance is not the identity. It is the relationship between an Auth Method (a runtime's ability to attest workloads) and a target system (something the organization wants to protect). The identities flow through that relationship at runtime, and the policy applies to all of them by construction, including the ones that do not yet exist.

RBAC + ABAC: necessary, not sufficient

Once the authentication layer is anchored in Auth Methods, the authorization layer still needs to say what those identities are allowed to do. RBAC ("this Auth Method can access these target roles") and ABAC ("…when the claim env=prod is present") get you a long way. They are necessary. They are not sufficient, because agents are non-deterministic in a way that no role and no attribute can capture.

4. The Layer That's Missing: Intent-Aware Enforcement on Every Action

Here is where the IdP-centric model breaks most decisively. Even its own proponents now acknowledge, almost in passing, that the set of actions an agent might take is not fully knowable at the time permissions are granted. This is correct. It also quietly invalidates the entire premise of static authorization, which is what every workforce IdP actually provides.

An agent with a perfectly scoped OAuth token, issued by a perfectly governed IdP, with a perfectly enforced TTL, can still:

- Be prompt-injected by a string in a Slack message it was asked to summarize, and execute a destructive action that its prompt never asked for.

- Hallucinate a SQL query that drops a table when its task was to count rows.
- Chain three legitimate tool calls together in a way that exfiltrates data the policy author never anticipated.
- Pivot from a read task to a write task in the same session, with the same token, with no policy violation visible at the authorization layer.

RBAC and ABAC are blind to this. They evaluate at the moment of authentication and, in the best case, at the moment of authorization. They cannot evaluate what happens after, because what happens after is decided by an LLM looking at a context window neither the IdP nor the policy author has any visibility into.

The only place this can be enforced is at the moment of action, on the wire, between the agent and the target system. Which means:

- Every action the agent takes against an organizational system must traverse a gateway.
- The gateway must be able to inspect the action, not just the authentication, against a rule set.
- The rule set must be expressive enough to evaluate the action in context, including the originating prompt, the agent's declared intent, the target object, and the agent's session history.
- The evaluation must produce an allow/deny verdict in sub-second time, per action, with no central choke-point in the IdP.

The legacy policy engines, and where each of them stops being useful

Before the IAM community converges on "more OPA" as the answer to agentic authorization, it is worth being concrete about what the existing policy engines actually do, and where each of them runs out of road. Every one of these projects is excellent at what it was designed for. None of them was designed for an LLM-driven actor whose next action is decided at runtime by a context window the policy author cannot see.

The most commonly cited examples in the agentic-IAM discourse:

- Open Policy Agent (OPA) and Rego. A CNCF graduated project, deservedly popular for Kubernetes admission control, microservice authorization, and infrastructure-as-code policy. Rego is a logic language that evaluates structured input against declarative rules. It is excellent at answering "given this JSON request and this caller, is the call allowed?" It does not, and was never designed to, evaluate whether the natural-language prompt that generated the request matches the action being attempted. You can pass a prompt into Rego as a string, but Rego cannot reason about it, only string-match against it. That gap is the entire problem.

- Amazon Cedar. AWS's open-source policy language used by Verified Permissions, designed for fine-grained application-level authorization with formal verification properties. Cedar is in many ways the cleanest expression of attribute-based authorization the industry has produced. It is also fundamentally a structured-attribute evaluator: principal, action, resource, context. It cannot evaluate semantic intent for the same reason Rego cannot. The grammar has no primitive for "the meaning of what the agent said it was going to do."
- Google Zanzibar and its open-source descendants (AuthZed/SpiceDB, OpenFGA, Ory Keto). Relationship-based access control (ReBAC) systems that answer "does principal X have relation Y to resource Z?" at massive scale and low latency. Brilliant for collaborative SaaS authorization. Completely orthogonal to the agent problem, because the question they answer is structural (who is related to what), not semantic (what is this caller actually trying to do, and does it match what they said they were trying to do).
- XACML and the OASIS PDP/PEP model. The original attribute-based authorization framework, still in use in heavily regulated environments. Same structural ceiling: an XML policy can express arbitrarily complex predicates over attributes, but cannot evaluate the prompt that produced the request, because attributes are by definition pre-computed values, not natural-language reasoning targets.
- OAuth scopes, SCIM entitlement catalogs, and the "least autonomy" framing emerging from the agentic IAM camp. These are not really policy engines, they are vocabularies for declaring what an identity is allowed to do at issuance time. They share the same fundamental limitation: the decision is made when the token is minted, not when the action is attempted. A token narrowly scoped to "read customer records" will happily authorize a read of every customer record in sequence, because nothing in the scope vocabulary distinguishes "read one record for the support ticket you were asked about" from "exfiltrate the whole table."

Notice the pattern. Every one of these systems evaluates a request against a policy whose terms were defined in advance, by a human, in a structured grammar. That model is sound when the set of possible requests is bounded and the requester is deterministic. It breaks the moment the requester is an LLM whose next move is a function of its context window and the policy author has no way to enumerate, in advance, the actions it might take.

This is not a criticism of OPA, Cedar, Zanzibar, or XACML. They are correctly engineered for the problem they were built to solve. It is a criticism of the assumption that agentic authorization is a more granular version of the same problem. It isn't. It is a different problem, and the existing engines are the wrong shape for it.

Why static policy languages can't carry this load

To be even more explicit about the structural ceiling: you cannot write a Rego policy, a Cedar policy, a SpiceDB schema, or a SCIM scope catalog that anticipates every action an LLM-driven agent might

attempt across every system in the enterprise. The combinatorics defeat the policy author. Worse, they defeat the policy author asymmetrically: a defender must enumerate every prohibited action, while an attacker (or a hallucinating agent, or a prompt-injected agent) need only find one un-enumerated path.

The pragmatic answer is to accept that the rule layer needs flexibility a structured policy language cannot provide, and to allow operators to write rules in free-form natural language. Examples: “do not allow any destructive operation against a customer-facing table,” “do not allow PII to leave the masking boundary,” “do not allow a read-only-classified prompt to issue a write.” Those rules are then evaluated with an LLM at the gateway, on every action.

This is uncomfortable for the IAM community because it admits something we have spent decades resisting: the enforcement layer must, in part, be probabilistic. It will not be 100% deterministic. It will be wrong sometimes. But the failure mode of a statistical enforcement layer that catches 95% of mismatched intents is dramatically better than the failure mode of a deterministic policy language that catches 0% because the policy author never imagined the scenario.

And critically, the statistical layer doesn’t replace RBAC or ABAC. It sits on top of them. The deterministic layer enforces the floor: the agent cannot reach a system it is not authorized for under any circumstances. The intent layer enforces the ceiling: within the systems the agent is authorized for, what it actually does has to match what it claimed it was going to do.

Two Architectures, Side by Side

Dimension	IdP-Centric Model (Okta, Ping, et al.)	Workload-Anchored, Gateway-Enforced Model
Unit of governance	The named agent identity, registered in a universal directory.	The Auth Method (runtime attestation source) bound to a target system.
Discovery requirement	Must continuously discover and enroll every agent before it can be governed.	No discovery required. Any agent presenting a valid attestation is governed by the policy bound to its Auth Method.
Handles ephemeral agents (sub-second lifetime)?	Poorly. Enrollment latency exceeds the agent's lifetime.	Natively. Governance applies at first authentication; no pre-registration needed.
Handles agents spawning sub-agents across substrates?	Requires each sub-agent to also be registered. Quickly becomes impractical at chain depth.	Each sub-agent authenticates with its own runtime's attestation. Policy applies to the relationship, not the chain element.
Enforces what the agent actually does, not just what it is?	No. Authorization is evaluated at token issuance. Runtime monitoring is on the 2026 roadmap.	Yes. Every action traverses a gateway that evaluates intent against the originating prompt and policy.
Catches prompt-injected destructive actions?	No. The token is valid, the scope is valid, the action passes RBAC. The IdP has no insight into the prompt.	Yes. Mismatch between declared intent and attempted action is denied at the gateway before any credential is minted.
Where credentials live	Vended to the agent (ephemeral OAuth tokens). Agent holds the credential.	Injected into the brokered session by the gateway. Agent never holds the credential.
Blast radius of a compromised agent	Whatever the OAuth scope permits, for the lifetime of the token.	Whatever the agent can accomplish in one gateway-mediated, intent-checked session before TTL.

The “Single Pane of Glass” Argument Is a Red Herring

The most persistent argument for the IdP-centric model is operational: customers want one place to manage all their identities. This is a legitimate desire, and it deserves a direct answer.

The single-pane-of-glass argument confuses where governance is administered with what governance actually does. You can have a single pane of glass for authoring Auth Methods, target systems, access rules, and intent policies, without forcing every transient agent into a directory entry. The pane of glass is the policy authoring console. The enforcement happens at the gateway, on the wire, at the moment of action.

The IdP-centric pitch sells the pane of glass as a substitute for enforcement. It is not. A directory with 50,000 stale agent entries, a kill switch that propagates over minutes, and a roadmap-pending UEBA layer for agent behavior is not enforcement. It is reporting. The kill switch fiCP server is precisely the kind of architectural over-reach that produced the NHI mess in the first place. Service-account sprawl exists because organizations tried to manage workload identity with the tools and mental models of human IAM. Agent sprawl will exist for exactly the same reason, on exactly the same vendors’ platforms, unless someone admits the model is wrong.

What the Right Architecture Actually Looks Like

A workable agentic-access architecture has four properties, none of which the IdP model provides:

1. Authentication is anchored in workload runtimes, not in a synthetic agent directory

Every agent authenticates with whatever its runtime can attest: cloud IAM, K8s SA token, OIDC ID token, certificate, or a SPIFFE SVID issued by SPIRE. The trust chain ends at the platform that ran the agent, not at a directory entry the platform never knew existed. For organizations already running SPIRE for their non-AI workloads, agents fit the existing model with zero new identity primitives.

2. Authorization is defined between Auth Methods and target systems, not between identities and scopes

The policy author writes: “identities authenticating via this Auth Method, with these claims, may request brokered access to this target, scoped to these objects, for at most this TTL.” The set of identities matched by this rule is open-ended by design. New agents that match the rule are governed automatically. Agents that don’t match are rejected automatically. No enrollment step.

3. Every action traverses a gateway, and the gateway evaluates intent, not just permission

RBAC and ABAC enforce the floor: an agent cannot reach a system it is not authorized for. An intent-aware policy layer, evaluated by an LLM at the gateway against natural-language rules, enforces the ceiling: within the systems the agent is authorized for, what it actually does must match what it claimed it was going to do. Mismatches are denied before a credential is minted.

4. The agent never holds the credential

Short-lived credentials are injected into the brokered session at the gateway, at the moment of execution, invisible to the agent itself. A compromised agent has nothing to leak. A prompt-injected agent has nothing to leak. The blast radius of a successful exploit is whatever the agent can accomplish in one gateway-mediated session before TTL, not the lifetime of a harvestable token.

This is not theoretical. This is how the secrets-management and PAM industries have been moving for years: toward dynamic, brokered, gateway-mediated, zero-standing-privilege access. AI agents are not a reason to reverse that direction. They are the most acute reason yet to accelerate it.

Closing

There is a sentiment running through the IdP-centric literature that deserves to be taken seriously: to get AI security right, you have to get identity right. Agreed. The disagreement is about what "identity" means when the principal is non-deterministic, ephemeral, multi-substrate, and prompt-injectable.

For human users, identity is who is at the keyboard. For traditional NHIs, identity is which service is calling. For AI agents, identity is the relationship between the runtime that issued the workload and the target system it is trying to reach, evaluated continuously, on every action, against what the agent said it was going to do.

Build the IdP-centric model and you will have a beautiful directory of dead identities, a kill switch that fires too late, and an authorization layer that cannot see the prompt that compromised it. Build the workload-anchored, gateway-enforced model and you will have something that actually works for the way agents actually run, today, and for the next architectural shift after this one.

The hard middle isn't identity. The hard middle is enforcement, on every action, in a world where the identity didn't exist five seconds ago and won't exist five seconds from now. Get that right, and identity falls into place behind it.