# The Spine

## An Architectural Catalog for Enterprise AI Agent Coordination

**SaaSquach AI Labs (a division of Charles & Roe Inc.) · June 2026 · v1.6**

> *"The architecture of an AI agent platform is not invented. It is discovered, one failure mode at a time. The Spine is the catalog that names them in advance."*

## Executive Summary

The Spine is a nine-spec architectural catalog for building production AI agent systems at enterprise scale. It addresses the nine concerns every large enterprise discovers, usually one painful incident at a time, when moving from *AI in their workflow* to *agents acting on their behalf at scale*: tool discovery, multi-agent coordination, external-signal provenance, composite risk scoring, deterministic governance, durable context across time, grounded enterprise data, a system of record for agentic assets, and a sovereign runtime where first-party agents execute.

Each spec, **PDS** (Progressive Discovery Spine), **ACS** (Adversarial Coordination Spine), **ESF** (External Signal Fabric), **CRI** (Composite Risk Index), **AGS** (Agent Governance Spine), **DCS** (Durable Context Spine), **GDS** (Grounded Data Spine), **ARS** (Agent Registry Spine), and **SRS** (Sovereign Runtime Spine), is portable, versioned, and citable in its own right. Five of the nine are public open-source (CC BY 4.0 + MIT); CRI, GDS, ARS, and SRS are held private (CRI and GDS under a patent and license-preservation posture, ARS a draft slated for the next coordinated release, SRS under the same controlled-asset posture). The catalog is maintained by SaaSquach AI Labs, a division of Charles & Roe Inc.

The architectural contribution is not any individual spec. The nine concerns the Spine names have prior art in open-source and commercial form (Anthropic, Microsoft, Salesforce/MuleSoft, Palantir, AWS, UiPath, Bloomberg, OpenAI, Open Policy Agent, SPIFFE, LangGraph, Composio, Letta, the broader Y Combinator agent-infrastructure cohort, and many others have productized pieces of this surface). What is distinctive is that **the catalog names all nine as separable architectural concerns under a single umbrella**, and produces a clean **ten-way failure attribution dictionary** when an incident occurs: bad customer data → PDS; bad world data → ESF; bad reasoning → ACS Planner; bad evaluation → ACS Evaluator; bad scoring → CRI; bad governance → AGS; bad continuity → DCS; bad grounding → GDS; bad or missing registry → ARS; bad or unbounded execution → SRS.

For a large enterprise running fifty or more named agents across multiple business units (agents calling tools, agents calling each other, agents touching production systems), the catalog is a portable architecture for coordinating internal AI applications, composing them with external data, and maintaining compliance posture as a byproduct of normal operation rather than a separate workstream.

This document is the half-whitepaper introduction to the catalog. The full specs live on GitHub; the canonical landing page is **saasquach.ai/spine**.

---

## The Enterprise AI Coordination Problem

The agent-tooling space in 2025 and 2026 fragmented across nine separate communities, each with its own vocabulary and its own well-known tools. Examples of each: for tool discovery, the Model Context Protocol; for multi-agent coordination, LangGraph, AutoGen, and Magentic-One; for external-signal pipelines, Bloomberg, Kensho, and Everstream; for risk scoring, CVSS, FICO, and Moody's; for governance, OPA, Cedar, SPIFFE, and the Microsoft Agent Governance Toolkit; for durable context (agent state and memory that survives across sessions and time), Letta, MemGPT, and the harness-engineering work (OpenAI's Codex harness, Anthropic's long-running-agent harness, the SWE-agent ACI research); for grounded enterprise data, dbt, Cube, AtScale, Palantir Ontology, and Glean; for the agentic registry, service registries, the CMDB, Backstage, and the MuleSoft Agent Registry; and for the agent runtime, where first-party agents actually execute, AWS Bedrock AgentCore, UiPath, Palantir Rubix, and Kubernetes.

Each community treats its layer as the whole problem. None of them name the others. Production teams trying to ship agent systems at scale have to reconcile nine vocabularies and nine sets of citations on their own, and discover, painfully, that the layers compose differently than the marketing claims. (The harness-engineering literature's own framing, *"execution is a commodity; the moat is the environment"*, is itself the Spine thesis arrived at independently, and it names *progressive disclosure* (PDS) as the first repeating pattern of every serious harness.)

At pilot scale (five to ten agents, one business unit, one tool surface) enterprises can get away with ignoring the architectural decomposition. Prompt-engineering and good engineering hygiene carry them. At scaled deployment (hundreds or more named agents, multiple business units, agents calling tools, agents calling each other, agents touching production systems) the failure modes are not avoidable. They are only discoverable. Either the enterprise adopts an architecture that names them in advance, or it discovers them one at a time in production, on the company's reputation.

The nine failure modes look like this:

- **Tool-pick errors at scale.** Once an agent has access to a few hundred tools, context bloat and ambiguous tool selection start producing wrong-tool-called incidents. The enterprise discovers PDS the hard way after the third such incident, usually after a quarter of frustrated debugging.
- **Multi-agent decisions collapsing on themselves.** When the planner and the evaluator are the same model in the same context window, the evaluator agrees with the planner. Coordination quietly degrades into mutual confirmation. The enterprise discovers ACS after a high-stakes multi-agent decision lands wrong and the post-mortem cannot locate where it broke.
- **Unauditable signal provenance.** A regulator, an auditor, or a customer asks "where did this come from, on what date, with what confidence?" If the signal flow was not designed for that question, the answer is "we don't know." The enterprise discovers ESF the first time someone with subpoena power asks.

- **Single-number risk blindness.** A composite score reduced to one number is a single point of failure. The first time a wrong call traces back to "the model said it was a four" (with no confidence band, no provenance, no sensitivity to which underlying signal moved) the inadequacy is obvious. The enterprise discovers CRI after the first material wrong call.
- **Jailbroken agents taking real actions.** Prompt-level safety leaks. The JailbreakBench benchmark (Chao et al., NeurIPS 2024) and Andriushchenko et al. (ICLR 2025) both show adaptive attacks reach near-100% success rates against frontier safety-aligned models. The first production incident where an agent is talked into doing something destructive is the moment the enterprise discovers AGS. Microsoft's framing on this is unusually clean: *"actions the governance layer denies are not 'unlikely.' They are structurally impossible."* That is the difference between probabilistic and deterministic safety.
- **Work that doesn't survive the boundary.** A project outgrows a single context window. A later session looks at the work, infers it is finished, and declares victory on something half-built; or it inherits an unintelligible half-applied mess and burns its budget on archaeology; or it re-derives everything from scratch; or it trusts a stale memory file that points it the wrong way. The enterprise discovers DCS the first time a long-running agent project quietly stalls, regresses, or ships incomplete work across the session boundary, and the post-mortem finds no durable, ground-truth record of state.
- **Inconsistent numbers and leaked data.** Agents reason over inconsistent metric definitions and leak data across entitlement boundaries. The enterprise discovers GDS the first time two agents give different numbers for the same question, or an agent surfaces data a user was not cleared to see.
- **Shadow agents and registry drift.** Agents and tools nobody catalogued, plus registry entries that no longer match the deployed reality. The enterprise discovers ARS the first time it cannot answer "what AI do we have running," or an unapproved agent touches production.
- **Unbounded, ad-hoc execution.** The agents an enterprise builds itself run wherever, with inconsistent isolation and no enforced blast-radius bounds, and governance becomes bypassable from inside the runtime. A runaway or compromised first-party agent is not contained because nothing was built to contain it. The enterprise discovers SRS the first time one of its own agents does damage that a bounded, identity-attached runtime would have stopped at the edge of a disposable session.

The choice for an enterprise CIO or CTO sitting at this altitude is not whether to address these concerns. They will be addressed, one way or another. The choice is whether to address them as a coherent architectural framework adopted in advance, or to discover them ad hoc as nine separate fires, in production, on the company's reputation. Adopting a portable architecture (the Spine catalog or another one) shortens that discovery loop from years to weeks.

---

## The Catalog at a Glance

| # | Spec | Status | What it does in one sentence |
|---|---|---|---|
| 1 | **PDS, Progressive Discovery Spine** | Public · CC BY 4.0 + MIT | Gives one agent the right tools, scoped per task. The layer above MCP. |
| 2 | **ACS, Adversarial Coordination Spine** | Launches 2026-06-03 | Coordinates multi-agent work (Planner / Generator / Evaluator triads) so long-running runs stay coherent. |
| 3 | **ESF, External Signal Fabric** | Launches 2026-06-09 | Fuses external-world signals (commodities, geopolitics, supplier health, logistics) with what the enterprise already knows internally. |
| 4 | **CRI, Composite Risk Index** | Private · patent-preservation | Composite scoring with confidence bands, tenant-conditioned weights, and signal-version provenance. |
| 5 | **AGS, Agent Governance Spine** | Launches 2026-06-01 | Deterministic policy enforcement, per-agent identity, tamper-evident audit at the protocol layer. |
| 6 | **DCS, Durable Context Spine** | Public · CC BY 4.0 + MIT | Persists agent state, memory, and knowledge across the context-window boundary, disconnected sessions, and time. The temporal layer. |
| 7 | **GDS, Grounded Data Spine** | Private, license posture pending | The data foundation: a canonical semantic model (text-to-metric, one agreed definition per metric and entity) plus data-level entitlements (row/column access enforced at the data layer). What the data means, and who may see it. |
| 8 | **ARS, Agent Registry Spine** | Private, draft | The system of record layer: one continuously-reconciled catalog of every agentic asset (agents, tools, MCP servers, models). PDS discovers from it; AGS enforces against it. |
| 9 | **SRS, Sovereign Runtime Spine** | Private | The execution substrate: a sovereign, first-party agent runtime you own and run on infrastructure you control. Where the agents you build yourself execute. |

### Two tiers and an interchange

The nine layers fall into two tiers connected by one interchange, which is how the catalog is drawn on saasquach.ai/spine: a transit map running from your data estate on one end to agents in production on the other. The **foundation tier** is what an agent must have before it can be trusted: grounded data (GDS), a system of record for what exists (ARS), and durable context across time (DCS). The **capability tier** is what an agent then does with that foundation: external signals (ESF), progressive tool discovery (PDS), adversarial coordination (ACS), composite risk scoring (CRI), and deterministic governance (AGS). **SRS, the sovereign runtime, is the interchange where the two tiers meet**: it is where the foundation is composed and the capabilities execute, and the point where outside applications plug in and the agents an enterprise builds itself actually run.

### The Ten-Way Failure Attribution Dictionary

| Attribution surface | Owned by |
|---|---|
| Bad customer data (internal data feeds) | PDS |
| Bad world data (external data feeds) | ESF |
| Bad reasoning | ACS Planner |
| Bad evaluation | ACS Evaluator |
| Bad scoring | CRI |
| Bad governance | AGS |
| Bad continuity (lost or stale durable state) | DCS |
| Bad grounding (wrong definition, or data the user could not see) | GDS |
| Bad or missing registry (a shadow asset, or a drifted entry) | ARS |
| Bad or unbounded execution | SRS |

## Layer 1: PDS, Progressive Discovery Spine

### What it does

PDS is the architectural pattern for how an agent discovers the tools available to it. The principle: agents should discover tools the way a competent new hire discovers them (task-scoped, workflow-bundled, search-first), not by being handed the full catalog the moment they wake up.

The conventional approach is to dump the entire tool inventory into the model's context window via something like the Model Context Protocol's `tools/list`. At enterprise scale this is a context-bloat disaster: an agent with access to 200+ tools across 8 ERP systems can burn 20%+ of its context window on tool metadata before reasoning starts. Tool-pick accuracy degrades, hallucinated tool selection appears, and audit trails become impossible.

PDS specifies a different pattern. Tools are organized into workflow-scoped packs. Discovery is search-first: the agent asks for tools relevant to a natural-language query and receives the top-K matches, not the full inventory. Identity and tenant-scope are baked into every tool call. Retry, audit, idempotency, and circuit-breaking are first-class primitives, not retrofits.

### Why it matters at enterprise scale

For an enterprise running multiple AI applications against multiple ERP and SaaS systems, PDS is what makes the tool surface scale beyond a handful of integrations. Without it, every new tool added to the platform increases the load on every other agent's context window. With it, the tool catalog can grow indefinitely without affecting any individual agent's reasoning quality.

PDS is also where per-tenant isolation lives. Two customer tenants of the same AI platform should never see each other's tools. That boundary is enforced at the PDS gateway, not in the agent prompt.

## Industry productizations and convergence sources

- **Anthropic, Knowledge Work Plugins / Claude Cowork.** Anthropic's plugin format is workflow-scoped (PDS principle), skill-bundled (PDS principle), and uses scoped MCP connectors (PDS principle). The strongest productization of the catalog's first layer. Anthropic's own framing: *"every plugin follows the same structure: skills, commands, connectors; every component is file-based, markdown and JSON, no code, no infrastructure, no build steps."*
- **Model Context Protocol (modelcontextprotocol/servers).** The reference MCP server registry; 86K+ stars on GitHub. PDS sits above MCP: MCP is the substrate, PDS is the curation pattern.
- **Composio.** 1,000+ tool integrations with explicit tool-search and per-toolkit auth. The strongest commercial productization of the search-first principle at scale.
- **Nango (NangoHQ/nango).** Pre-MCP gateway pattern at production scale, processing billions of API requests for Replit, Ramp, Mercor, and hundreds of others. Validates the architectural shape at scale.
- **Speakeasy / TrueFoundry / Amazon API patterns.** Each productizes pieces of the discovery and curation contract for SDK-generation or API-gateway use cases.
- **Claude-Mem (thedotmack/claude-mem).** Uses "Progressive Disclosure" as published philosophy name and ships a token-efficient search → timeline → fetch workflow. Vocabulary-level convergence on the same architectural principle.
- **AWS Bedrock AgentCore Gateway.** Turns existing APIs and Lambda functions into agent-callable MCP tools behind a managed gateway, the major-cloud productization of the curated, mediated tool surface.
- **MuleSoft Agent Fabric (Salesforce) and UiPath Integration Service.** Enterprise connector and tool catalogs agents draw on, the governed surface PDS specifies the progressive-discovery discipline for.
- **Microsoft Copilot Studio and the Agent Governance Toolkit.** Microsoft's agent-builder surface plus a governance kernel that gates the tool surface, the Microsoft-stack instance of the discovery-and-control layer.
- **Palantir Foundry AIP, OpenAI function-calling tool search, LangGraph dynamic tools.** Each ships search-first or scoped tool exposure as a first-class feature rather than dumping the full catalog into context.

## How an enterprise uses PDS

Picture a large company with dozens of AI applications sitting on top of dozens of back-end systems: ERPs, CRMs, SaaS tools. Without PDS, every agent is handed the keys to all of it at once, which is overwhelming and easy to get wrong. With PDS, each agent is treated like a new hire on its first day: it is shown only the handful of tools its current task needs, and when it needs something else it asks by describing what it is trying to do. One platform team curates the master list of tools, and the agent teams never have to see the whole thing.

The quiet benefit is a clean paper trail. Every time any agent uses any tool, the system records who did it, for which customer, and with which version. So "which agent did what, when, and for whom" becomes a question you answer in seconds, not a forensic project.

## Layer 2: ACS, Adversarial Coordination Spine

### What it does

ACS is the architectural pattern for how multi-agent systems coordinate. The principle: planner and evaluator must be structurally separated (different agents, different context windows, sometimes different models) because if they share state, the evaluator confirms whatever the planner produced.

ACS specifies role decomposition (Planner, Generator, Evaluator, Judge as separate roles with separate context windows), structured handoffs (typed payloads, not free-form prose), state-on-disk rather than state-in-context, and negotiated contracts before code (the Planner and Generator agree on the interface before the Generator produces output).

The pattern is named after the architectural insight that the Evaluator should be *structurally incapable of agreeing with the Generator*: different model, different prompt, different reasoning chain. Adversarial in the cooperative sense, the system gets the benefit of disagreement designed into the architecture.

### Why it matters at enterprise scale

For an enterprise running multi-agent workflows (supply-chain planning agents, financial close agents, customer-support agents, security-investigation agents) ACS is what keeps long-running multi-agent runs coherent. Without it, a five-agent collaboration session devolves into mutual confirmation by hour two. With it, the system catches its own mistakes by architectural construction.

ACS is also the layer that produces the post-incident attribution between “the plan was wrong” and “the evaluation missed it.” Those are different failures with different remediation paths: one suggests retraining the Planner, the other suggests recalibrating the Evaluator.

### Industry productizations and convergence sources

- **Anthropic.** Multiple Anthropic publications in 2025 and 2026 specify variants of this pattern (multi-agent research, long-running agents posts, AI Engineer Summit talks). The Anthropic framing of separate-context-window agents with an adversarial evaluator is the closest published analogue.
- **Microsoft Magentic-One.** Orchestrator plus four specialized agents (WebSurfer, FileSurfer, Coder, ComputerTerminal). Productized example of role decomposition.
- **Moonshot Kimi K2.5, PARL.** Three-reward decomposition (helpfulness, harmlessness, format compliance) trained into the model via reinforcement learning. The training-time equivalent of ACS’s runtime evaluator separation.
- **LangGraph (LangChain).** Graph-based multi-agent orchestration with state checkpointing and human-in-the-loop nodes. The most-adopted OSS framework for the structured-handoff pattern.
- **OpenAI Swarm.** Lightweight handoff-coordination framework: every agent specifies what it can hand off to, and the framework routes accordingly.
- **Cognition (Devin).** Autonomous engineering agent with planner / executor separation; one of the highest-scale production deployments of multi-agent code synthesis.

- **Letta (formerly MemGPT).** Long-lived agents with shared-memory blocks across multiple agent processes, the canonical productized multi-agent persistent-memory model.
- **Microsoft AutoGen.** Multi-agent conversational orchestration; widely adopted in enterprise Python deployments.
- **UiPath Maestro.** Orchestrates agents, robots, and people across BPMN-modeled processes, a deterministic coordination layer over agents with human-in-the-loop, the clearest production example of heterogeneous executors.
- **MuleSoft Agent Fabric (Salesforce).** An Agent Broker routes and delegates work conditioned on intent, policy, and identity, governing agent-to-agent interaction over the A2A protocol.
- **AWS Bedrock AgentCore Evaluations and Palantir AIP Logic.** A major-cloud scoring of agent behavior on correctness, faithfulness, and tool selection, and a platform that composes LLM workflows from deterministic blocks with a separate evaluation pass, the Evaluator role made concrete.

### How an enterprise uses ACS

Take a month-end financial close run by several agents: one reconciles the books, one hunts for anomalies, one routes approvals, one keeps the audit trail. ACS enforces the same rule a good finance department lives by: the one who does the work is never the one who signs off on it. The anomaly-checker is a different model with a different prompt, so it cannot simply rubber-stamp whatever the first agent produced, and the agents hand work to each other in a clean, structured way. If the close comes out wrong, you can see exactly which agent dropped the ball, because the structure itself recorded every handoff.

---

## Layer 3: ESF, External Signal Fabric

### What it does

ESF is the architectural pattern for how agents consume external-world signals. The principle: external signals must arrive with provenance (source, timestamp, confidence, version), schema (typed payloads), and freshness guarantees (staleness alarms), or downstream reasoning is unauditable.

The conventional approach is to wire each external data source into each agent ad hoc: one curl call here, one MCP server there, one webhook somewhere else. At enterprise scale this is unmanageable: hundreds of data sources times hundreds of agents equals thousands of undocumented integrations, each with its own staleness profile and its own failure modes.

ESF specifies a fabric: every external signal flows through a typed, provenance-threaded, freshness-aware substrate before it reaches any agent. Signal versions are checkpointed. Provenance is logged. Staleness alarms fire when a feed falls behind. When an agent reasons about a commodity price or a geopolitical event, it consumes the ESF-curated version, not a raw upstream feed.

## Why it matters at enterprise scale

For an enterprise whose AI applications reason about the world (supplier risk, commodity exposure, geopolitical disruption, weather impact, freight visibility) ESF is what makes the reasoning auditable. Without it, "the agent said X because it read Y at time Z" cannot be reconstructed. With it, every agent-produced claim carries provenance back to the underlying signal.

ESF is also where regulatory compliance lives for AI-produced market-facing outputs. When a regulator asks "what signal informed this trading recommendation?" the answer comes from the ESF audit trail, not from the agent's reasoning chain.

## Industry productizations and convergence sources

- **Apache Kafka / Apache Flink / CloudEvents.** The streaming substrate that ESF assumes. Kafka for the durable log, Flink for the stream-processing topology, CloudEvents for the typed envelope contract.
- **Bloomberg B-PIPE.** The canonical financial market-data fabric: typed feeds, provenance, latency guarantees. Decades of production at trillions-of-dollars scale.
- **Kensho (S&P Global).** AI-enhanced extractive intelligence from earnings calls, regulatory filings, and market news. Productizes the "structured signals out of unstructured sources" pattern.
- **AlphaSense.** Enterprise search across earnings transcripts, broker research, and regulatory filings with semantic search and document-level provenance.
- **Everstream Analytics.** Supply-chain risk feeds (port disruptions, weather, geopolitical events) with provenance and confidence scores.
- **Resilinc.** Supplier risk scoring with multi-tier supplier visibility and event-driven alerts.
- **Project44.** Logistics and shipment visibility: over a billion shipment events with provenance per shipment.
- **Interos.** Multi-tier supplier concentration risk with continuous monitoring.
- **OpenLineage / W3C PROV.** The open standards for provenance metadata. ESF builds on the PROV vocabulary.
- **Materialize / Estuary / Tinybird.** Streaming-data substrate productized: operational analytics over streaming signals with the freshness and provenance guarantees ESF specifies.

## How an enterprise uses ESF

Imagine a company whose agents have to reason about the outside world: commodity prices, port delays, weather, supplier health, geopolitics. ESF makes every one of those outside facts arrive with a receipt attached: where it came from, when, and how sure the source was. So when an agent later says "supplier risk went up," you can trace that statement back to the exact signal and the exact moment behind it. The auditor's question, "why did this number jump on October 14," has a real answer instead of a shrug.

---

# Layer 4: CRI, Composite Risk Index

## What it does

CRI is the architectural pattern for composite risk scoring. The principle: a risk score reduced to a single number is a single point of failure. The score must carry confidence bands (how certain is the model), provenance (which signals contributed), tenant-conditioning (the weights vary by customer concentration profile), and gap tracking (penalties when inputs are missing).

The conventional approach is a weighted-sum heuristic: "30% market risk + 25% credit risk + 20% operational risk + 15% concentration + 10% other" dressed as a model. At pilot scale this is fine. At enterprise scale, when a wrong call traces back to "the model said it was a 4" with no further detail, the inadequacy is obvious.

CRI specifies a different shape. Composite scores carry confidence bands. Component scores are versioned and provenance-stamped. Tenant-conditioned weighting means a customer with 60% supplier concentration in one country gets a different weighting than a customer with diversified supply. Backtests are first-class: the score's predictive validity over historical windows is auditable.

## Why it matters at enterprise scale

For an enterprise that produces risk scores its customers, regulators, or executives consume (supplier risk, credit risk, security risk, operational risk) CRI is what makes the score defensible. Without it, the score is a black box. With it, every component is reproducible, every weighting is justifiable, and every confidence band is calibrated against historical data.

CRI is also the layer where the AI council's "explainability" mandate lives. When an executive asks "why did this risk score change?" the answer comes from CRI's per-component provenance, not from a model card.

## Industry productizations and convergence sources

- **CVSS (Common Vulnerability Scoring System).** The canonical composite score for cybersecurity vulnerabilities. Multi-component, weighted, with explicit confidence categories.
- **FICO.** The consumer credit score: multi-component, conditioned, with explicit reason codes. A model for component-level explainability at scale.
- **Moody's KMV (RiskCalc, EDF).** Distance-to-default credit modeling: a composite that fuses equity-market signals with accounting data into a calibrated default probability.
- **Bloomberg DRSK.** Default risk score combining equity, credit, and macro signals, productized at institutional scale.
- **S&P Capital IQ.** Peer-relative scoring across financial-health dimensions, productizes the comparative weighting CRI specifies.
- **LSEG StarMine.** Alpha composite combining earnings revisions, quantitative valuation, and analyst sentiment: a multi-factor composite with confidence bands.
- **Resilinc and Everstream supply-chain risk indices.** Productize the supplier-tier composite with concentration-aware weighting.

- **Interos.** Continuous-monitoring concentration risk score with multi-tier supplier visibility.
- **Langfuse, Pydantic Logfire, and Promptfoo.** The open-source trace, eval, and provenance fabric that feeds CRI's confidence-band and provenance layers with the per-component quality signals it scores.
- **Platform observability (Azure Monitor, Foundry Observability).** Major-stack telemetry (latency, status, error, and eval results) that supplies the upstream score-quality inputs CRI's confidence bands consume.

### How an enterprise uses CRI

Picture a platform that scores risk for many different customers, each exposed to risk in a different way. CRI does for a risk score what a good credit report does for a credit score: instead of one mystery number, you get the number, the reasons behind it, and how confident the model is. The weights adjust to each customer, so a company with all its eggs in one basket is scored differently from a diversified one. When a score moves from 32 to 41, the platform can name the factors that drove it and how sure it is, so the board question "why did our risk change" has a real, defensible answer.

CRI is kept private under a patent-preservation posture. The novel part, the way it blends the underlying signals for each customer, is patentable, so adopting CRI is a development partnership rather than an open spec.

---

## Layer 5: AGS, Agent Governance Spine

### What it does

AGS is the architectural pattern for production AI agent governance. The principle: prompt-level safety leaks. The JailbreakBench benchmark (Chao et al., NeurIPS 2024) and Andriushchenko et al. (ICLR 2025) both show adaptive attacks reach near-100% success rates against frontier safety-aligned models. Therefore production agents need deterministic policy enforcement at the tool boundary: actions the policy denies are *structurally impossible*, not "unlikely."

AGS specifies ten architectural principles: deterministic policy enforcement (not prompt-level safety); identity per agent (not per session, SPIFFE / DID / mTLS); tamper-evident audit log; policy as code (not prose); privilege rings (four-ring sandboxing for promotion / demotion); kill switch with SLO monitoring and chaos testing; tool-poisoning detection and drift monitoring; shadow-agent discovery; trust scoring for plugin marketplaces; governance-aware training.

The empirical case is anchored in academic research (JailbreakBench, Andriushchenko et al., the Microsoft Red Team's post-100-product-red-teaming blog) and in OWASP standards (LLM06 "Excessive Agency," OWASP Agentic AI Threats and Mitigations).

### Why it matters at enterprise scale

For an enterprise running production agents that can take real-world actions (sending email, executing POs, approving payments, modifying data, calling external APIs) AGS is what makes the deployment defensible to a

CISO, an internal audit team, or a regulator. Without it, “is this AI sandboxed?” has no convincing answer. With it, the answer is layered: deterministic policy enforcement, per-agent identity, four-ring sandbox promotion, tamper-evident audit, kill switch, all logged and reviewable.

AGS is also the layer where SOC 2, ISO 27001, and NIST AI RMF compliance becomes a byproduct of normal operation. Policy-as-code produces the policy evidence. Tamper-evident audit produces the audit evidence. Per-agent identity produces the identity-attestation evidence.

## Industry productizations and convergence sources

- **Microsoft Agent Governance Toolkit (microsoft/agent-governance-toolkit).** The strongest single productization of AGS: eight packages covering all ten OWASP Agentic Top 10 surfaces. MIT-licensed. Microsoft anchors the empirical case in the same JailbreakBench / Andriushchenko / Red Team citation triplet that AGS uses.
- **Open Policy Agent (OPA, CNCF).** The canonical OSS policy engine. Foundational for AGS principle #1 (deterministic policy enforcement).
- **AWS Cedar.** Policy language with formal verification. The ABAC-flavored architectural variant alongside OPA.
- **OpenFGA (CNCF).** Zanzibar-style relationship-based access control. The ReBAC architectural variant, maps cleanly to agent-acting-on-behalf-of-user delegation chains.
- **Cerbos.** Stateless, language-agnostic policy decision point: the PDP-as-sidecar deployment variant.
- **Permit.io.** Productized authorization service with multi-tenant policy management.
- **SPIFFE (CNCF) and W3C DIDs v1.0.** The identity substrate AGS principle #2 (identity per agent) builds on.
- **e2b (e2b-dev/e2b).** Firecracker-microVM sandbox runtime for agent-generated code: the OSS productization of AGS principle #5 (privilege rings).
- **Daytona (daytonaio/daytona).** Persistent-state sandbox sibling to e2b: the dev-environment-grade four-ring sandbox option.
- **Promptfoo.** Production red-teaming and eval framework used by OpenAI and Anthropic. Complements the academic JailbreakBench / Andriushchenko empirical baseline with the production-deployed eval counterpart.
- **Inspect (UK AI Security Institute).** Sovereign-grade eval framework used by UK AISI and US AISI for frontier-model assessments.
- **JailbreakBench (Chao et al., NeurIPS 2024).** The canonical empirical benchmark for adaptive jailbreak success rates.
- **Andriushchenko et al. (ICLR 2025).** 100% adaptive-attack success rate on Vicuna, Mistral, Phi, Nemotron, Llama, Gemma, GPT-3.5, GPT-4o, R2D2, and 100% on Claude via transfer or prefilling.
- **Microsoft Red Team blog (January 2025).** Post-100-generative-AI-products red-teaming summary: prompt-layer defenses are probabilistic by construction.
- **OWASP LLM06:2025 (Excessive Agency) and OWASP Agentic AI Threats and Mitigations.** The canonical OWASP risk taxonomies.

- **AWS Bedrock AgentCore Policy and Identity.** Compiles natural-language rules to Cedar, validates them against the tool schema, and enforces at the gateway before any tool call, outside the model's reasoning loop. The most complete major-cloud productization of deterministic governance outside the model.
- **UiPath AI Trust Layer with Automation Ops.** Centralized control, identity, and audit for UiPath-managed and third-party LLM usage, with human-in-the-loop approvals.
- **MuleSoft Agent Fabric (Salesforce) and Palantir Foundry / AIP.** Policy, identity, and audit applied to every agent action at the gateway, and access scopes enforced for both humans and agents with purpose-based, rationale-recorded grants.
- **Microsoft Entra Agent ID and Conditional Access, Microsoft Purview, Azure Monitor.** The Microsoft-stack governance surface: per-agent identity and conditional access, data-level governance and DLP, and the observability that feeds the audit trail.

### How an enterprise uses AGS

Picture a regulated company, a bank or an insurer, running agents that can take real actions. AGS puts a hard gate in front of every action an agent tries to take. The rules are written as code, not as polite instructions buried in a prompt, so an action the rules forbid is not merely unlikely, it is impossible. Every agent carries an ID that can be checked, and every action it takes is written to a tamper-proof log. New agents start fully boxed in and earn more freedom only as they prove themselves, and there is a kill switch that is tested on a schedule.

The payoff shows up in the hard questions. "Show me everything this agent did last quarter" is one query. "Could a hacked prompt make this agent email the outside world?" is answered "no, it was never given that ability," not "probably not." "Show me the rule that governs this action" returns a file an auditor can read.

---

## Layer 6: DCS, Durable Context Spine

### What it does

DCS is the architectural pattern for how agent state, memory, and knowledge survive across time. The principle: the context window is disposable scratch; a durable store is the system of record. Anything a future session needs (what is done, how to boot the environment, what was decided) must live in a durable, ground-truth artifact, because the next session shares none of the last session's context window.

Where PDS scopes *which tools* an agent sees (per task) and ACS coordinates *which agents* run together (per run, concurrent), DCS governs *which state survives* (across runs, across time). It is the orthogonal temporal axis. A single agent picking a project back up two weeks later hits every DCS failure mode and zero ACS failure modes.

DCS specifies a durable store as the system of record; an explicit, verification-gated completion ledger as the only source of "done" (never inferred from the artifact); a clean, resumable exit discipline for every session; an initializer role that builds the substrate once so workers stay cheap; a deterministic startup sequence that orients before new work; progressive disclosure of durable knowledge (a short map, not a rotting monolith); a

working / episodic / semantic memory hierarchy that survives lossy compaction; and provenance, freshness, and identity-partitioning so durable state stays true, scoped, and auditable.

## Why it matters at enterprise scale

For an enterprise running agent work that outlives a single context window (multi-week migrations, long-running operations, codebases and investigations built across hundreds of sessions) DCS is what keeps the work from silently regressing or shipping incomplete across the session boundary. Without it, a long-running agent project declares victory early, inherits its own half-built mess, or re-derives orientation on every run. With it, any session (a different agent, weeks later) picks up the thread in seconds against a ground-truth record.

DCS is also where enterprise memory tenancy and the "what did the system know, and when?" question live. Durable state is identity-partitioned (no cross-tenant or cross-project bleed) and persisted with provenance, so the continuity record doubles as a forensic and audit surface, composing directly with AGS.

## Industry productizations and convergence sources

- **OpenAI, Harness Engineering.** The Codex harness experiment (~1M lines of code, zero hand-written) runs on the repository as the system of record plus a ~100-line `AGENTS.md` map: *"anything the agent cannot access in context while running effectively does not exist."* The strongest single productization of the catalog's temporal layer.
- **Anthropic, Harness Design for Long-Running Application Development.** The initializer + coding-agent split, the `feature_list.json` completion ledger (stored as JSON specifically so models don't casually overwrite it), the progress file, git-as-recovery, and the per-session startup sequence: the documented response to "the agent had no persistent, structured understanding that could survive the context-window boundary."
- **SWE-agent (Princeton NLP).** Context management (capping tool output, collapsing stale observations into single-line summaries) as load-bearing interface design (a 64% improvement from interface design alone).
- **MemGPT / Letta.** The working / episodic / semantic memory hierarchy and deliberate paging; identity-scoped shared memory blocks across agents.
- **The Awesome Agent Harness taxonomy.** Separates *frameworks* (what you build on) from *runtimes* (what keeps running: persistent memory, scheduled execution, cross-session coordination). DCS is the spec for that runtime-persistence layer.
- **AWS Bedrock AgentCore Memory.** Managed short-term and long-term agent memory (semantic, summary, user-preference, and episodic strategies) across sessions, the closest major-vendor peer to the durable-state-and-memory-across-time concern.
- **Palantir Foundry AIP.** AIP Threads and Ontology-Backed Objects persist conversational and operational state across sessions, a partial convergence on the durable-context concern.

### How an enterprise uses DCS

Picture a year-long migration handled by dozens of agents across hundreds of separate work sessions. The danger is that each session forgets what the last one did, so work gets redone, declared finished when it is not, or quietly broken. DCS is the shared logbook that survives between sessions. Each session, whoever runs it, reads the log, does one verified piece of work, and writes clean notes before it stops. “Done” is never a guess, it is a checked-off entry in the log. Eighteen months later, “what did we know on March 14, and which decision was live at the time” is a question you can actually answer.

DCS is public open source (CC BY 4.0 + MIT), the catalog’s memory-over-time layer, free to adopt and cite alongside PDS, ACS, ESF, and AGS.

---

## Layer 7: GDS, Grounded Data Spine

### What it does

GDS is the architectural pattern for how agents reason over enterprise data with one trusted meaning and one enforced boundary on who may see what. The principle: an agent is only as trustworthy as the data underneath it, so the data layer must supply a canonical meaning for every metric and entity, and must enforce who is allowed to see which rows and columns, before the agent ever reasons.

The conventional approach is text-to-SQL against raw tables: the agent writes a query, and “revenue” or “an active customer” means whatever that query happened to express that time. At enterprise scale this produces two failures. First, the same question returns different numbers from different agents because each defined the metric its own way. Second, an agent surfaces data the end user was never entitled to see, because access was enforced (if at all) in the application above the data, not at the data itself.

GDS specifies two pillars. A semantic, canonical model means agents resolve text to a governed metric or entity definition (text-to-metric, not text-to-SQL), so “revenue” is the company’s one agreed definition every time. Data-level entitlements means row and column access is enforced at the data layer (on-behalf-of token exchange, row/column security, relationship or attribute-based rules), default-deny and scoped to the end user, so an agent and any permission-aware retrieval only ever return data that user is cleared for.

### Why it matters at enterprise scale

For an enterprise whose agents answer questions about the business, GDS is what makes the answers both consistent and safe. Without it, “what was Q3 revenue for this segment” has as many answers as there are agents, and a retrieval-augmented agent can leak a salary, a deal, or a customer record across an entitlement boundary. With it, every agent computes against the same governed definitions and sees only what its user is allowed to see, so the numbers reconcile and the boundary holds.

GDS is also where the “permission-aware AI” requirement lives. When a regulator or a customer asks “could this agent have seen data it should not have,” the answer is enforced at the data layer, not asserted in a prompt.

### Industry productizations and convergence sources

- **Semantic layers: dbt Semantic Layer, Cube, AtScale, Looker (LookML).** The canonical-metric pattern, one governed definition per metric, productized for analytics and increasingly for agents.
- **Snowflake Cortex Analyst, Databricks Genie and Unity Catalog, Microsoft Fabric.** The text-to-metric over a governed model pattern from the major data platforms.
- **Palantir Foundry Ontology.** The flagship enterprise productization of a canonical semantic model agents reason over. The closest commercial analogue to the GDS thesis, proprietary and platform-locked.
- **Glean.** Permission-aware enterprise retrieval, the entitlement-respecting RAG pattern GDS specifies, at product scale.
- **RFC 8693 (OAuth token exchange), OpenFGA, AWS Cedar, row and column-level security.** The on-behalf-of and data-level-entitlement substrate GDS builds on.
- **Microsoft Fabric IQ and Microsoft Purview.** Fabric IQ (OneLake plus a semantic model plus ontologies plus data agents) for the canonical-model pillar, and Purview for data-level governance and DLP, the Microsoft-stack instance of GDS.
- **UiPath Context Grounding and AWS Bedrock (Knowledge Bases plus Lake Formation).** Turnkey grounding over business data, and managed retrieval paired with row, column, and cell-level data security, each converging on the grounding and data-security pillars while stopping short of the canonical semantic model and identity-scoped entitlements at reasoning time, the GDS depth gap.

### How an enterprise uses GDS

Take a company whose agents answer questions about the business. GDS wires those agents to one trusted set of definitions instead of letting each one reach into the raw database and work it out its own way. “Revenue” means one agreed thing, so two agents asked the same question give the same number. And the boundary is enforced underneath: when an agent answers for a regional manager it sees only that region’s data, and a document search returns only files that manager could already open. The board question “are our AI answers consistent, and could an agent ever see data a person could not” has a built-in answer instead of a hopeful one.

Put simply: raw tables are the whole filing cabinet and the keys. GDS is a reporting desk where every term already means one agreed thing and you only get the files you are cleared for.

GDS is kept private while its license posture is decided, the same open question as CRI. It is listed here as a named layer.

---

## Layer 8: ARS, Agent Registry Spine

### What it does

ARS is the architectural pattern for the inventory beneath discovery and governance. The principle: PDS hands an agent the right tools and AGS governs what an agent may do, and both presuppose a catalog of what exists (agents, tools, MCP servers, models, datasets, skills) that in most deployments no single layer actually owns. ARS makes that inventory one explicit, typed, continuously reconciled system of record.

The conventional state is a fragmented inventory: tools in an MCP config, agents in an orchestrator, models in a model registry, datasets in a data catalog, with no shared identity. At enterprise scale this produces shadow assets (an agent or tool nobody catalogued, which therefore cannot be discovered, scoped, governed, or audited) and drift (a catalog entry that no longer matches the deployed reality, so discovery routes to a dead endpoint and governance enforces a stale scope).

ARS specifies one registry as the system of record for every agentic asset, with a stable identity and a typed entry per asset (kind, version, owner, lifecycle, endpoint, scope, capability, provenance); continuous reconciliation of declared state against observed reality, which is how shadow assets and drift are detected; mandatory ownership, provenance, and lifecycle on every entry; and identity-partitioned visibility, so each tenant sees only the assets it is entitled to discover. Discovery (PDS) reads from it; governance (AGS) enforces against it, where shadow-agent discovery is reconciliation and "deny the unregistered" is a deterministic policy.

## Why it matters at enterprise scale

For an enterprise running fifty or more agents and a growing tool surface, ARS is what makes "what do we actually have running, and is any of it something nobody approved" an answerable question. Without it, the discovery layer (PDS) and the governance layer (AGS) work from separate, drifting lists, and shadow agents operate unseen. With it, there is one current catalog both read from, every asset has an owner and a lifecycle state, and an asset that can act but is not registered is a detected incident, not an invisible one.

ARS is also where the registry's own history becomes an audit surface: every registration, version bump, and retirement is logged, so "what existed, with what scope, when" is reconstructable, composing with AGS audit and DCS durability.

## Industry productizations and convergence sources

- **MuleSoft Agent Fabric (Salesforce), Agent Registry.** A system of record for every agentic asset, continuously scanned for shadow agents. The closest direct statement of the ARS thesis from a major vendor.
- **Service registries: HashiCorp Consul, Netflix Eureka.** The microservices answer to the same problem, continuously health-checked dynamic discovery.
- **Schema and model registries: Confluent Schema Registry, MLflow Model Registry, Hugging Face Hub.** The versioning and lifecycle half of ARS as proven infrastructure.
- **Data catalogs: DataHub, OpenMetadata, Databricks Unity Catalog, Collibra.** System-of-record catalogs with ownership and lineage, the model for the dataset asset kind.
- **Backstage software catalog (Spotify, CNCF), and the ITIL CMDB.** The "everything has an owner and one record" discipline, and the decades-old enterprise system-of-record pattern ARS is the agentic analogue of.
- **Microsoft agent mesh and Entra Agent ID, the Model Context Protocol server registry.** Agent discovery, trust scoring, per-agent identity, and the canonical tool-surface inventory ARS treats as one asset kind. The Entra "all agent identities" view inventories every agent in the tenant (Foundry agents, Copilot Studio agents, and more).

- **AWS Agent Registry (Bedrock AgentCore).** A private, governed catalog of agents, tools, skills, MCP servers, and resources with typed metadata, search, an approval workflow, and audit, a major-cloud instance of the system-of-record-for-agentic-assets concern.
- **UiPath Orchestrator and Automation Hub.** The central system of record for robots, processes, jobs, queues, and assets, plus the registry and pipeline of automations.
- **Palantir Foundry Ontology.** A typed object graph that functions as the operational system of record, extending the registry concern into an actionable, write-back object model.
- **SBOM (CycloneDX, SPDX).** The software, and now AI, bill-of-materials standard, the provenance-and-inventory discipline ARS applies to agentic assets.

### How an enterprise uses ARS

Imagine a company running fifty or more agents and a growing pile of tools and models. ARS is the single inventory of everything that can act: every agent, tool, connector, and model, who owns it, and whether it is approved. Scanners constantly compare the inventory against what is actually running, so anything operating off the books, an agent nobody registered, gets flagged at once, and any entry that no longer matches reality gets caught. The discovery layer only offers agents things that are on the list, and the governance layer refuses to act for anything it cannot find on the list. So “show me every AI thing that can touch production, who owns it, and is any of it unapproved” is one query. The rule of thumb: you cannot secure what you do not know you have.

ARS is private and slated to publish with the catalog’s next release. It is listed here as a named layer.

---

## Layer 9: SRS, Sovereign Runtime Spine

### What it does

SRS is the architectural pattern for where a first-party agent executes. The other layers say what an agent can discover (PDS), how agents coordinate (ACS), what signals they consume (ESF), how risk is scored (CRI), how every action is governed (AGS), what state survives (DCS), what data grounds them (GDS), and what assets exist (ARS). SRS says where the agents an enterprise builds itself actually run: a trusted, isolated, ephemeral, identity-bound runtime that natively composes the rest of the catalog. It is a specification for runtime behavior, not a hosted product, so an enterprise implements it on whatever execution substrate it controls (microVMs, Kubernetes, a managed runtime, bare metal) and is never locked to one vendor’s runtime.

The principle that makes it strategic is the distinction between two doors into the agent estate. Outside agents and tools, including closed-source vendor applications, plug INTO the Spine: they consume discovery, governance, and grounding through governed boundaries but execute on their own runtime. The agents an enterprise builds itself run ON the Sovereign Runtime Spine, where identity, tools, memory, governance, grounding, and observability are present by construction. Open at the edges, sovereign at the core.

### Why it matters at enterprise scale

Every production agent platform grows a runtime: AWS Bedrock AgentCore Runtime, UiPath robots, Palantir Rubix. For the agents an enterprise builds itself, the runtime is where isolation, blast-radius containment, and the composition of the whole catalog actually happen. Without a named runtime model, first-party execution is ad hoc: agents run wherever, with inconsistent isolation and bounds, and governance becomes bypassable from inside the runtime. SRS names the model so first-party execution is trusted by construction, and because it is a specification rather than a product, the enterprise owns and controls its core agent execution rather than renting it.

### Industry productizations and convergence sources

- AWS Bedrock AgentCore Runtime: serverless microVM session isolation, scale-to-zero, sandboxed code and browser execution.
- UiPath robots: the execution layer that runs automations, the RPA-era runtime now extended to agents.
- Palantir Rubix: hardened Kubernetes for governed agent and Function execution.
- Microsoft Foundry Agent Service: a session-isolated managed runtime with built-in identity and observability, the Microsoft-stack instance of the sovereign, identity-bound, ephemeral execution layer SRS describes.
- Firecracker and gVisor (isolation substrates), e2b and Daytona (agent sandboxes), and Kubernetes (orchestration). SRS specifies the behavior a Spine-native runtime must provide; these are substrates it is portable across. Multiple major vendors independently shipping an agent runtime is the convergence signal that the execution layer is a real, nameable concern.

### How an enterprise uses SRS

Picture a company that builds its own agents. SRS is the secured, company-owned space those agents run in. Every agent shows up with its ID already attached, works in a sealed, disposable room sized to how much damage it could do, and has its tools, rules, memory, and data wired in from the start. Outside vendors' agents never run loose inside this space, they interact through controlled doorways instead. So when security asks "where do our own agents actually run, and what is the worst one could do," the answer is the hard limits of the room, not a hope. And because it is a specification the company owns rather than a product it rents, the company controls its core agent engine outright.

SRS is kept private, the same posture as CRI, GDS, and ARS, so the runtime model stays a controlled asset.

---

## The Ten-Way Failure Attribution Dictionary

The pedagogical artifact the Spine produces, and the operational artifact a CIO actually needs at enterprise scale, is the ten-way failure attribution dictionary. When an agent system produces a wrong output, the architecture itself locates which layer failed.

| Attribution surface | Owned by | What it looks like |
|---|---|---|
| Bad customer data | PDS | The agent called the wrong tool, called the right tool with wrong params, or used stale tenant-scoped data |
| Bad world data | ESF | The agent's reasoning consumed a stale signal, a wrong-version signal, or a signal whose confidence was not propagated |
| Bad reasoning | ACS Planner | The plan was wrong for the inputs: the model produced a flawed plan |
| Bad evaluation | ACS Evaluator | The evaluator failed to catch a bad plan; the Generator's output passed the Evaluator's check despite being wrong |
| Bad scoring | CRI | The composite score was wrong because the fusion was miscalibrated, the weights were wrong for the tenant profile, or the confidence band was missing |
| Bad governance | AGS | The agent took an action it should not have been allowed to take; policy enforcement failed or was incorrectly written |
| Bad continuity | DCS | A later session declared work done that wasn't, inherited an unintelligible half-built state, re-derived orientation it should have read, or trusted a stale durable artifact that survived compaction while the load-bearing nuance did not |
| Bad grounding | GDS | The agent computed against the wrong definition of a metric or entity, or a retrieval surfaced data the end user was not entitled to see |
| Bad or missing registry | ARS | An asset that could act was never catalogued (a shadow agent or tool), or a registry entry drifted from the deployed reality so discovery or governance ran against stale information |
| Bad or unbounded execution | SRS | A first-party agent ran on a runtime without enforced identity, isolation, or blast-radius bounds, so a runaway or compromised agent was not contained |

The value of the dictionary is operational. Without it, "AI did something wrong" stays abstract: every incident becomes a custom investigation that touches every layer. With it, every incident becomes a specific, ownable, layer-attributable engineering ticket, and the post-mortem is shorter, the remediation is targeted, and the lessons-learned compound across the enterprise.

The dictionary only works if you have all nine layers. A single-layer adopter (just AGS, just PDS) gets the layer-specific benefit but not the attribution dictionary. The composition is where the unique value lives.

## The Spine Gate, from sandbox to production

The nine specs name what an enterprise must get right. They do not, by themselves, answer the question a CIO running mass agent experimentation actually asks: *how does a promising prototype graduate into production without either drowning in ceremony that kills experimentation velocity, or letting an unvetted agent touch*

*real systems?* At scale, an enterprise is not deploying one agent, it is running hundreds of experiments, and the governance problem is the **funnel** between them.

The Spine Gate is the catalog's operating model for that funnel. It is not a tenth concern; it is how the nine are run. Three zones:

**1. The sandbox, mass prototyping.** Experiments run in isolated execution (per-session containers or micro-VMs, narrowly-scoped credentials that never include production access, no production data) under entry-tier controls, with a blast radius near zero by construction. This zone optimizes for volume and velocity at low ceremony. Most prototypes die here, and that is the point: the sandbox is where ideas are cheap.

**2. The Spine Gate, the graduation checklist.** A prototype earns production only by clearing a per-spec checklist. The ten-way failure attribution dictionary, read forward instead of backward, *is* that checklist: the same nine surfaces that locate a failure after an incident become the pre-flight questions an agent must answer before it ships.

| Gate question | Owned by | Promotes when |
|---|---|---|
| Is the agent's identity cryptographically provable, and its policy deny-by-default? | AGS | identity is cryptographically (and, per blast radius, hardware-) rooted; every action passes deterministic policy; audit is tamper-evident |
| Can it discover and call only the tools its task needs? | PDS | tools are scoped per workflow, deny-by-default, with validated parameters |
| Do its sub-agents inherit only the privilege they need? | ACS | handoffs are identity-verified; no unscoped privilege inheritance; evaluator separation holds |
| Is its memory isolated, integrity-checked, and time-bounded? | DCS | session/tenant isolation is enforced; no memory-based privilege retention; unverified context expires |
| Are its external inputs typed and provenance-stamped? | ESF | every signal carries source, timestamp, and confidence |
| Is its blast-radius-weighted risk below the promotion threshold? | CRI | the composite risk score, with its confidence band, sits under the tier's bar |
| Does it reason over governed definitions, and see only data its user is entitled to? | GDS | metrics and entities resolve to one canonical definition; data-level entitlements are enforced default-deny at the data layer |
| Is it registered, owned, and reconciled against reality? | ARS | the agent and its tools are in the registry with an owner and a lifecycle state; nothing it uses is a shadow asset |
| Does it run in a bounded, isolated, identity-bound runtime? | SRS | the agent executes in an isolated ephemeral session with enforced resource and blast-radius limits, on a runtime the enterprise controls |

The rigor demanded at the gate **scales with blast radius**, across the same Foundation / Enterprise / Advanced maturity tiers the broader industry (including Anthropic's published Zero-Trust guidance for AI agents) has converged on. A read-only, low-stakes agent promotes at the Foundation tier. An agent that can

move money, touch regulated data, or write to production must clear Enterprise or Advanced. The threshold is not a single bar; it is a function of what the agent can break.

**3. The production Spine, governed operation.** Promoted agents run under the full catalog composition and the higher control tiers: immutable audit, real-time anomaly detection, automated containment, end-to-end provenance traceability. Here the ten-way dictionary resumes its original, forensic role, locating the owning layer when something does go wrong.

The result is that mass prototyping stops being a governance liability and becomes a pipeline. A hundred experiments run cheaply in the sandbox; the Gate is the structural filter that admits only those that clear their tier; the production Spine governs what passes; and the attribution dictionary catches whatever still slips. Velocity at the bottom, rigor at the top, and a single auditable path connecting the two, which is precisely the *governed path from experimentation to industrialization* an enterprise AI Council is chartered to provide.

---

## Compliance Posture as a Byproduct

The catalog's compliance posture is designed to be a byproduct of normal operation, not a separate workstream. The mapping to common compliance regimes is direct.

**SOC 2 Type II.** AGS produces the audit log (CC7.2 monitoring of system performance), the access-control evidence (CC6.1 access controls), and the change-management evidence (CC8.1 change management). PDS produces the tool-call provenance. ESF produces the signal-source attestation. CRI produces the scoring-versioning audit trail. ACS produces the multi-agent decision audit trail. The auditor gets a structured query interface against the audit log; no bespoke evidence collection.

**ISO 27001.** AGS's per-agent identity (SPIFFE / DID / mTLS) maps to A.9 access control. AGS's tamper-evident audit maps to A.12 operations security and A.16 incident management. PDS's tenant-scoping maps to A.13 communications security. The catalog covers the agentic-AI portion of the controls without bespoke instrumentation.

**NIST AI Risk Management Framework (AI 100-1).** The framework's four functions (Govern, Map, Measure, Manage) map onto the Spine's surfaces:

- *Govern* → AGS (deterministic policy, identity, audit)
- *Map* → ESF + PDS (signal provenance, tool surface)
- *Measure* → CRI (composite scoring with confidence bands)
- *Manage* → ACS (multi-agent coordination with evaluator separation)

**OWASP Agentic AI Threats and Mitigations.** AGS covers all ten of the OWASP Agentic Top 10 surfaces. The Microsoft Agent Governance Toolkit (which AGS treats as the canonical productization) covers OWASP 10 / 10 as documented in Microsoft's published evidence.

**EU AI Act, Canadian AIDA, US Executive Order on AI.** The regulatory frameworks emphasize risk classification, transparency, human oversight, and recordkeeping. The Spine's ten-way attribution dictionary is

a direct fit for the recordkeeping and transparency requirements; CRI handles the risk classification with confidence bands; ACS's planner / evaluator separation is the architecturally-encoded human-oversight pattern (the evaluator role can be a human or another agent); and DCS provides the durable, provenance-stamped, identity-partitioned continuity record that the recordkeeping mandates presuppose: "what did the system know, and when" as a reproducible query rather than a reconstruction project.

## Sarbanes-Oxley (SOX) readiness: IT general controls and segregation of duties

For any enterprise whose AI agents touch systems in scope for financial reporting, SOX is the highest-stakes regime, and the one an audit committee raises first. SOX compliance is not a property an architecture can confer: it is the organization's control framework, tested for operating effectiveness over a period and attested by an external auditor. What the Spine does is make every AI deployment *audit-ready by construction*: it produces the IT general controls (ITGC) and segregation-of-duties (SoD) evidence an auditor tests, as a byproduct of running agents rather than as a separate instrumentation project. The mapping to the ITGC and SoD control families is direct:

| SOX control area | What the auditor expects | Spine layer(s) | Evidence artifact |
|---|---|---|---|
| Access to programs & data | Authenticated, least-privilege, scoped access | AGS (deterministic policy + per-agent identity + privilege rings); PDS (tenant/scope-bound access, credential boundary) | Policy-as-code, SPIFFE / DID identity manifests, per-call access-decision log |
| Segregation of duties | The party that initiates is not the party that approves | ACS (planner ≠ evaluator); AGS (privilege rings, policy-gated escalation) | Role decomposition, ring assignments, escalation audit trail |
| Change management | Program changes are versioned, reviewed, tested, approved | AGS policy-as-code; PDS versioned tool manifests | Git history, CI lint/test gates, PR approval on every policy change |
| Audit trail & evidence integrity | A complete, provably-unaltered record of activity | AGS tamper-evident audit log (hash-chain / Merkle / commitment anchoring) | Append-only log with cryptographic integrity, exportable per tenant |
| Completeness & accuracy / data lineage | Inputs to reported figures are traceable and current | ESF (signal provenance: source, timestamp, confidence, version); CRI (versioned, backtestable scoring) | Signal-lineage records, per-component score provenance |
| Computer operations & monitoring | Operations are monitored; failures detected and contained | AGS (kill switch + SLO monitoring + chaos testing) | SLO-breach alerts, kill-switch activation log, chaos-test reports |

The objection a SOX auditor raises first is AI nondeterminism: *can a stochastic system be trusted with financially-material actions?* The Spine's answer is architectural, not aspirational: you do not trust the model. Financially-material actions are gated **deterministically** at the tool boundary (AGS), the initiating agent is **structurally separated** from the approving agent (ACS), and human approval is required on anything material as a policy-defined escalation. The defensible claim is therefore not "the model behaves," but "the

model's judgment cannot, by construction, execute a material action without passing a deterministic control and (where required) a human gate," with the tamper-evident audit log proving it for every instance.

This covers the AI-agent ITGC surface specifically; it does not replace the enterprise's broader SOX program (ERP application controls, the financial close, entity-level controls). It plugs the agent layer into that existing framework and produces the agent-layer evidence the framework requires. The honest framing for an audit committee: the Spine makes AI-agent deployments **audit-ready, not automatically compliant**. Compliance remains an outcome of control design, operating-effectiveness testing, and auditor attestation, and the Spine's contribution is to make that outcome reachable by configuration rather than by a bespoke instrumentation effort.

For an enterprise pursuing certification or audit defensibility, the Spine reduces "compliance posture for AI agents" from a separate workstream to a configuration artifact. Policy YAML files, audit-log queries, and SPIFFE identity manifests are the auditor-facing evidence. The catalog's structure produces those artifacts as a byproduct of running agents.

---

## Composing Internal and External Data

A large enterprise has two streams of intelligence: what it knows about its own operations (internal: customers, orders, suppliers, inventory, ERP state) and what it knows about the world (external: markets, geopolitics, weather, freight, supplier health). The Spine's structural insight is that these streams should compose through different layers.

**Internal data → PDS.** Internal data lives behind PDS-curated tool surfaces. Every agent accessing customer data does so through PDS, with tenant-scope enforced at the gateway. Internal tools are versioned, audited, and scoped.

**External data → ESF.** External data arrives through ESF's typed, provenance-threaded substrate. Every agent reasoning about external signals consumes the ESF-curated version, with the signal's source, timestamp, and confidence preserved through the reasoning chain.

**Composition → ACS.** Multi-agent workflows that combine internal and external data (a supplier-risk agent that combines internal payment history (PDS) with external supplier-financial-health signals (ESF)) compose through ACS's planner / evaluator boundary. The planner integrates the two streams; the evaluator (separately prompted and separately modeled) validates the composition.

**Scoring → CRI.** When the composition produces a score (a supplier-risk index, a customer-lifetime-value forecast, a portfolio-concentration measure) CRI is where the composite lives. Tenant-conditioning ensures the score is appropriate to the customer's actual exposure profile.

**Governance → AGS.** Every layer above runs inside AGS's deterministic policy boundary. The agent that produced the supplier-risk composite cannot send an email about the result unless AGS's policy explicitly permits that action for that agent in that ring.

**Continuity → DCS.** Every layer above runs across many sessions and over time. DCS is the durable substrate beneath them, the system of record where ACS writes its handoffs, where the completion ledger of "done" lives, and where the provenance trail of every signal, score, and action persists across the context-window boundary so the work survives from one session to the next.

**Grounding → GDS.** Every agent that reads internal data reasons over one canonical set of metric and entity definitions, and sees only data the end user is entitled to. GDS is the foundation that makes the numbers consistent and the entitlement boundary hold before any layer above it reasons.

**System of record → ARS.** Every layer above operates on assets drawn from one reconciled registry. PDS discovers from it, AGS enforces against it, and an asset that can act without an entry is a detected shadow asset rather than an invisible one.

**Execution → SRS.** The agents the enterprise builds itself run on a sovereign, first-party runtime it owns and controls, where every layer above is present by construction: identity attached, tools and policy wired in, memory and grounding available, blast radius bounded per risk tier.

### The two doors

There are two ways an application meets the Spine, and the distinction is the strategic core of the catalog. Outside applications, including closed-source vendor tools, plug INTO the Spine: they consume discovery, governance, and grounding through governed boundaries but keep executing on their own runtime. The agents an enterprise builds itself run ON the Spine via SRS, where identity, tools, memory, governance, grounding, and observability are present by construction. Open at the edges, sovereign at the core. The enterprise interoperates with the world at its boundaries and owns its core agent execution outright.

This composition is the catalog's strongest architectural argument. The nine layers are separable (an enterprise can adopt them incrementally) but the compounding value is in the composition. An enterprise that has PDS + ESF + ACS + CRI + AGS + DCS + GDS + ARS + SRS in production has a structurally defensible answer to almost every "how do you know that?" question a regulator, auditor, customer, or executive can ask.

---

## Adoption Path

The catalog is designed to be adopted incrementally. A practical sequence for a large enterprise:

**Phase 1, AGS first (months 0 to 3).** Adopt AGS as the platform's deterministic policy layer. Wrap existing agents with OPA or Cedar policy enforcement. Issue per-agent SPIFFE identities. Stand up the tamper-evident audit log. This is the highest-leverage adoption: it produces immediate compliance posture and stops the most damaging failure modes (jailbroken agents taking real actions).

**Phase 2, PDS (months 2 to 5).** Migrate the tool surface from raw MCP or ad-hoc API calls to PDS-curated workflow packs. Tenant-scope the gateway. Add idempotency and circuit-breaking. This is the platform-leverage adoption: it makes every new agent the enterprise builds cheaper and more reliable.

**Phase 3, ESF (months 4 to 8).** Route all external signals through the ESF substrate. Add provenance, freshness alarms, and signal versioning. This is the audit-posture adoption: it converts every external-signal-driven decision from a black box to a reconstructable claim.

**Phase 4, ACS (months 6 to 12).** Migrate multi-agent workflows to the planner / evaluator pattern with separated context windows. This is the coordination-quality adoption: it improves multi-agent decision quality and produces post-mortem attribution.

**Phase 5, CRI (months 9 to 18).** Migrate composite scores to the CRI shape: confidence bands, provenance threading, tenant-conditioning. This is typically the longest phase because it touches the scoring models the enterprise has already shipped. Adopted last, because the other four layers feed CRI inputs and CRI's value is highest when those upstream layers are stable.

The order is not prescriptive: different enterprises will sequence differently based on which failure mode is most urgent. But the dependency structure is real: AGS, PDS, and ESF are loosely coupled to each other; ACS depends on PDS and ESF for its inputs; CRI depends on all four for its components. **DCS is the substrate beneath all of them**: adopt it whenever agent work begins to span multiple sessions or outlive a single context window, independent of where the enterprise is in the other phases. **GDS, ARS, and SRS are likewise foundational substrates rather than numbered phases.** GDS is adopted wherever agents reason over enterprise data, supplying governed definitions and data-level entitlements; ARS is adopted once the asset count grows enough that "what do we have running" stops being answerable from memory; SRS is adopted once the enterprise builds and runs its own agents and needs a sovereign, bounded runtime to execute them on.

---

## Closing

The Spine is the architectural catalog the AI agent industry has been building one layer at a time. PDS is the discovery layer; ACS is the coordination layer; ESF is the signal layer; CRI is the scoring layer; AGS is the governance layer; DCS is the continuity layer; GDS is the data-grounding foundation; ARS is the system-of-record foundation; SRS is the execution layer. Each has prior art. None of them, until now, had a catalog that named all nine as separable architectural concerns and produced a ten-way failure attribution dictionary when composed. And the **Spine Gate** is the operating model that turns the catalog into a funnel, promoting agents from sandbox experimentation into governed production through a tier-scaled, per-spec graduation checklist.

For a large enterprise running fifty or more named AI agents across multiple business units, the Spine is the portable architecture for coordinating internal AI applications, composing them with external data, and maintaining compliance posture as a byproduct of normal operation rather than a separate workstream. Five of the nine specs are public open-source; four (CRI, GDS, ARS, and SRS) are held private, CRI under a patent-preservation posture and developed in partnership with enterprise customers.

The catalog lives at **saasquach.ai/spine**. The public specs are on GitHub under `drewmattie-code`. PDS and DCS are live now under CC BY 4.0 + MIT. AGS launches publicly on 2026-06-01. ACS launches publicly on 2026-06-03. ESF launches publicly on 2026-06-09. CRI remains private. GDS, ARS, and SRS are private: GDS

under a license posture still being decided, ARS slated for the catalog's next coordinated release, SRS held as a controlled asset under the same posture as CRI, GDS, and ARS.

For enterprises evaluating the catalog as an architectural baseline, the practical first step is reading the public PDS spec and walking the nine-layer composition through a single representative workflow inside the enterprise. The dependency between the layers becomes visible immediately, and the adoption sequence falls out of the composition.

SaaSquach AI Labs maintains the catalog as a single umbrella. Direct correspondence to hello@saasquach.ai.

---

**Drew Mattie · SaaSquach AI Labs (a division of Charles & Roe Inc.) · saasquach.ai/spine · June 2026**