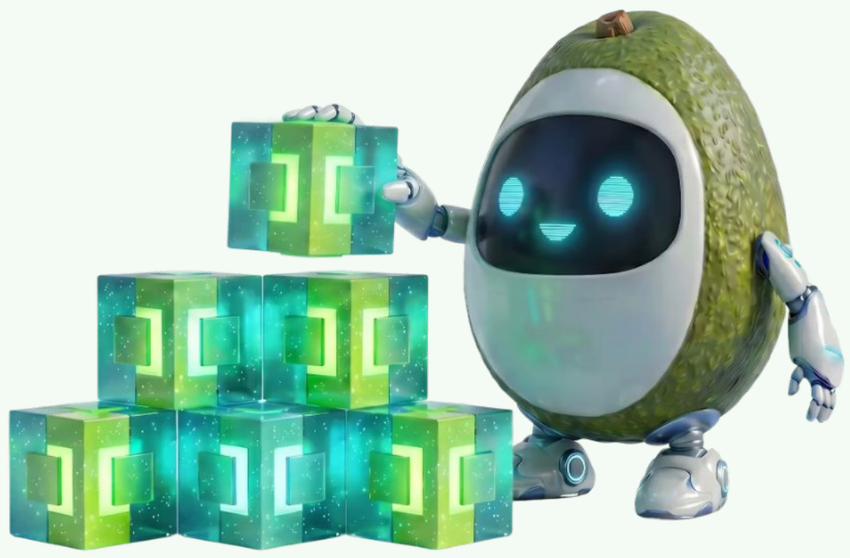
eBook

The Contextual Data Layer for Enterprise AI

6 architectural requirements for building agentic-AI-ready systems

What's in this ebook

- Why agentic AI breaks traditional architectures
- What a contextual data layer is and what it replaces
- The 6 capabilities your contextual data layer has to do to support production AI — and how they map to the Arango Contextual Data Platform



Introduction

Why enterprise AI fails in production

The support agent told the customer her contract expired in 2023. It hadn't. The vector index was eight months stale. The graph that linked accounts to entitlements lived in a different system. The LLM produced a confident, grammatical, entirely wrong answer — and a renewal conversation turned into an escalation.

You've probably lived some version of this story. A team builds an AI prototype that answers questions beautifully. The demo goes well. Leadership green-lights the rollout. Then production hits, and the system starts returning answers that are almost right, occasionally wrong, and impossible to debug.

> The model isn't the problem. The model is fine.
> **What breaks is everything underneath it.**

The demo ran against clean, curated data. The production system runs against seven source systems, a vector database like Pinecone, a graph database like Neo4j, and an orchestrator stitching them together at query time. Pipelines drift. Governance becomes a policy PDF. The system cannot explain itself.

< 10%
of enterprises have scaled agents to deliver value
McKinsey · Apr 2026

8 in 10
cite data limitations as the roadblock to scaling
McKinsey · Apr 2026

2 / 3
have experimented with agents but can't scale them
McKinsey · Apr 2026

Agentic AI makes the data problem worse. A chatbot asks one question and stops; an agent keeps going. It chains decisions across systems, and every weakness in your data architecture gets amplified by the number of steps the agent takes.

The chapters that follow lay out why this happens, what a contextual data layer actually is, and the six requirements your architecture has to meet to make agentic AI work in production. We'll start with the architectural shift agentic systems force on enterprise data — and why the Frankenstack approach can't keep up.

Chapter 1

Why agentic AI breaks traditional architectures

Agentic AI is redefining enterprise software. The model in front is no longer just a chatbot waiting for a question; it is a system that perceives, reasons, and acts across multiple steps and systems.

Why agentic AI changes the architecture

Agentic systems adapt to dynamic environments. As AWS describes them, they continuously perceive their environment and respond to change. Traditional SaaS architectures were built for the opposite problem: repeatable workflows against standardized schemas.

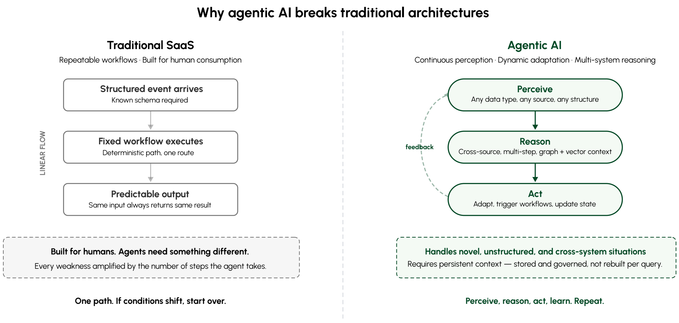


Figure 1.1 — Traditional SaaS was built for repeatable workflows against standardized schemas. Agentic AI perceives, reasons, and acts across multiple systems — and every weakness in the data architecture gets amplified.

Six architectural requirements separate production-ready agentic AI from prototypes that stall after the demo:

01
Semantic clarity
Data must carry meaning, not just structure.

02
Relationships & entity graph
Context must be connected, not fragmented.

03
Freshness & temporal correctness
Reflect what is true now — and at time T.

04
Provenance & trust
All data must be traceable, governed, explainable.

05
AI-native service layer
Context must be directly consumable by AI systems.

06
Unified multimodel platform
Context unified across all data types in a single model.

Every weakness in your data architecture gets amplified by **the number of steps the agent takes.**

Six failure modes in production AI

What goes wrong when context is missing

01 **Inconsistent results across queries**
The same question returns different answers because the underlying context isn't unified.

02 **High latency in multi-step reasoning**
Agents correlating across systems pay a round-trip tax at every step.

03 **Duplicated data and indexing pipelines**
The same data transformed and stored once for search, once for graph, once for embeddings.

04 **Governance inconsistencies**
Data accessible in one system but restricted in another — enforced nowhere consistently.

05 **Operational fragility**
When an ingestion job or embedding pipeline fails, the system serves stale or partial context without signaling it.

06 **Costs that compound at inference time**
Reassembling context at query time means over-fetching. Larger context windows, more tokens per query, and costs that multiply invisibly across every agent and workflow.

What is context?
Context is the working knowledge of your business: the entities, relationships, history, rules, and meaning that explain what is happening and why. Data is rows and records; context is what those rows mean, how they connect, and what is true about them right now. Persisted as a unified model, not reassembled on demand.

Definition
A contextual data layer is a persistent, structured representation of business reality. It combines meaning, relationships, state, and time in one model that AI systems can query directly

- **Multimodel by nature —** entities in documents, relationships in a graph, meaning in vectors, state in key-value, text in search indexes.
- **Persistent and live —** a maintained layer, not reassembled at inference time per query.
- **Queryable as a whole —** not federated across systems or stitched by application code.
- **Governed —** provenance and policy travel with the data, not bolted on after.

AI without context is a liability. **With context, it becomes leverage.**

Chapter 2

What is a contextual data layer?

A persistent, multimodel architectural tier that bridges enterprise data and AI systems — giving AI one continuous representation of the business instead of a collection of specialized stores glued together at inference time.

Persistent

Context is stored and maintained, not reassembled per query.

Multimodel

Graph, vector, document, key-value, and search — in one model.

Governed

Lineage, access control, and audit are structural, not bolted on.

Served

Retrieve, rank, cite, and ground are context layer primitives.

How it works

The layer operates as the system of record and control plane for enterprise context. Data flows in from every source that matters — operational systems, documents, events, unstructured content — and is contextualized once. AI agents and applications operate against this governed layer rather than querying underlying sources directly.

Consider a support operations use case: a document store holds tickets and Knowledge Base articles; a graph models the relationships between customers to products and incidents; vectors capture semantic similarity; key-value tracks current operational state. Inside a contextual data layer, these are not isolated systems — they are one continuously maintained model.

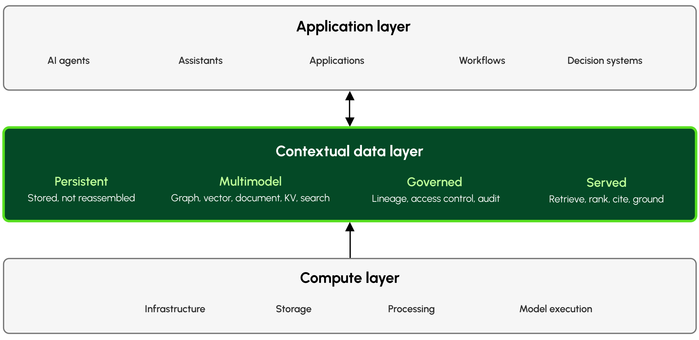


Figure 2.1 — The contextual data layer sits between compute and application, giving AI a consistent, governed representation of enterprise context.

The limitation of current architectures

From Frankenstack to trusted data foundation

In many enterprise AI implementations, context is not persisted. It is reconstructed at inference time by stitching together semantic similarities, relationships, structured records, and keyword matches — each pulled from a separate system. Each piece was the right call when it was added, but together, they become the problem.

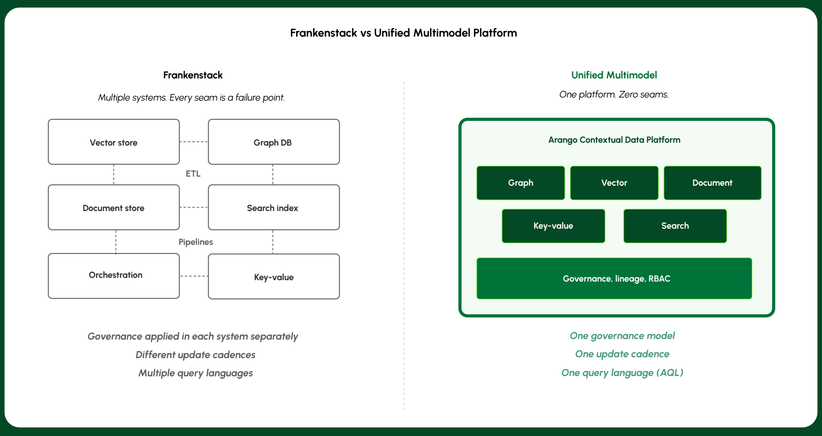


Figure 2.2 — Five systems, five governance models, and an orchestrator on the left; one platform and one query language on the right.

> "You can't scale what you can't govern, and you can't govern what you can't structure."
>
> — Bain, *Governance, Trust, and the Data Foundation*

From reconstruction to persistence

A contextual data layer inverts the usual model. Instead of reconstructing context at query time, context is persisted once and served many times.

A customer **is a document**

Their link to a contract **is a graph edge**

Similarity to a past ticket **is a vector**

All of it **in one model**

Remember: A contextual data layer persists context once. From that single decision, everything else improves — response speed, consistency, and explainability.

Chapter 3

The 6 requirements your contextual data layer must deliver

The 6 operating requirements of a contextual data layer: semantic clarity, relationships, freshness, provenance, an AI-native service layer, and unified multimodel coverage.

01
Semantic clarity →
Data carries meaning, not just structure. Canonical entities resolve terminology drift across CRM, billing, and support.

02
Relationships & entity graph →
Connections between entities are first-class and native. Multi-hop reasoning without federated joins.

03
Freshness & temporal correctness →
Context reflects what is true now — and can answer what was true at time T. Bi-temporal by design.

04
Provenance & trust →
Lineage, RBAC, and audit logging embedded in the platform. Every AI output is traceable to source.

05
AI-native service layer →
Retrieve, rank, cite, and ground are built-in. Not reimplemented in every consuming application.

06
Unified multimodel platform →
Graph, vector, document, key-value, and search — one foundation, one query language, one governance model.

Requirement 01

Semantic clarity

Schemas capture structure. They don't capture meaning.

Your schema does not capture what your data actually means. The CRM says "active"; billing says "paying"; support says "green." Same word, three definitions, three consuming systems.

CRM
status = "active"
= logged-in in last 30 days

Billing
status = "active"
= invoice paid this cycle

Support
status = "active"
= no open Sev-1

A contextual data layer resolves those definitions into a single shared meaning: one answer to "what does active mean," applied consistently across every system that asks.

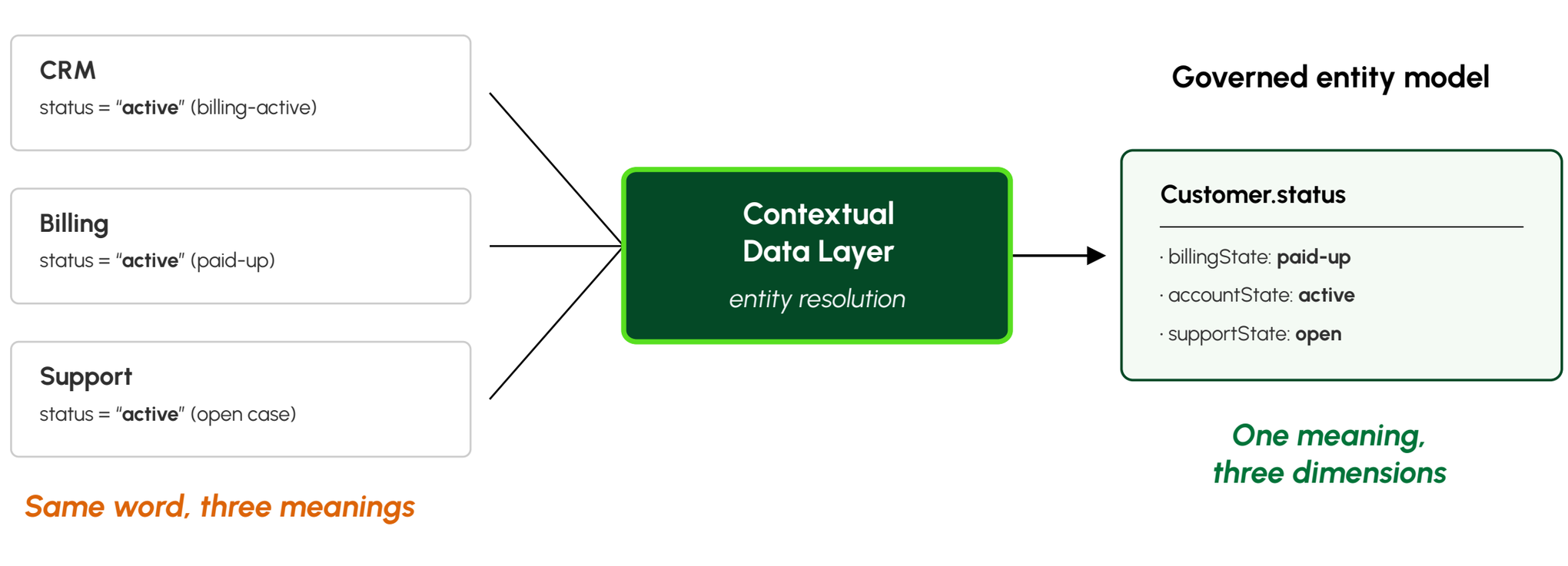


Figure 3.1 — Same word, three meanings. One governed entity model.

What this gives you

- One shared definition for every entity, across every system
- Business meaning that stays current as the organization evolves
- Consistent answers to the same question, every time

In Arango

- AutoGraph automatically builds a knowledge graph that understands your business domain
- AQL queries graph, vector, document, and key-value in a single statement — no translation between systems
- Definitions and relationships update continuously — no manual re-modeling required

Failure mode

Every consuming system invents its own join key. The same customer is cust_3781 in CRM, acme-corp in billing, and ACME Corporation, Inc. in support tickets.

AI agents operating on one slice cannot reason across the others without a brittle mapping table that drifts the moment a system changes, a new source appears, or someone enters data differently.

How Arango solves it

Arango resolves those fragmented identifiers into a single, stable entity, automatically. Whether a match is exact, approximate, or inferred from structural similarity, the result is one customer, one ID, one definition that every agent and application reads from. And when definitions evolve (because they always do), Arango tracks those changes over time. A query about what was true last quarter returns last quarter's answer. A query about today returns today's.

Requirement 02

Relationships & entity graph

Enterprise context must be connected, not fragmented.

Vector similarity finds related documents. It will not tell you that Customer X is churning because the engineer who championed them left three months ago. That's a relationship question — and relationships need to be first-class, native edges, not joins invented at query time.

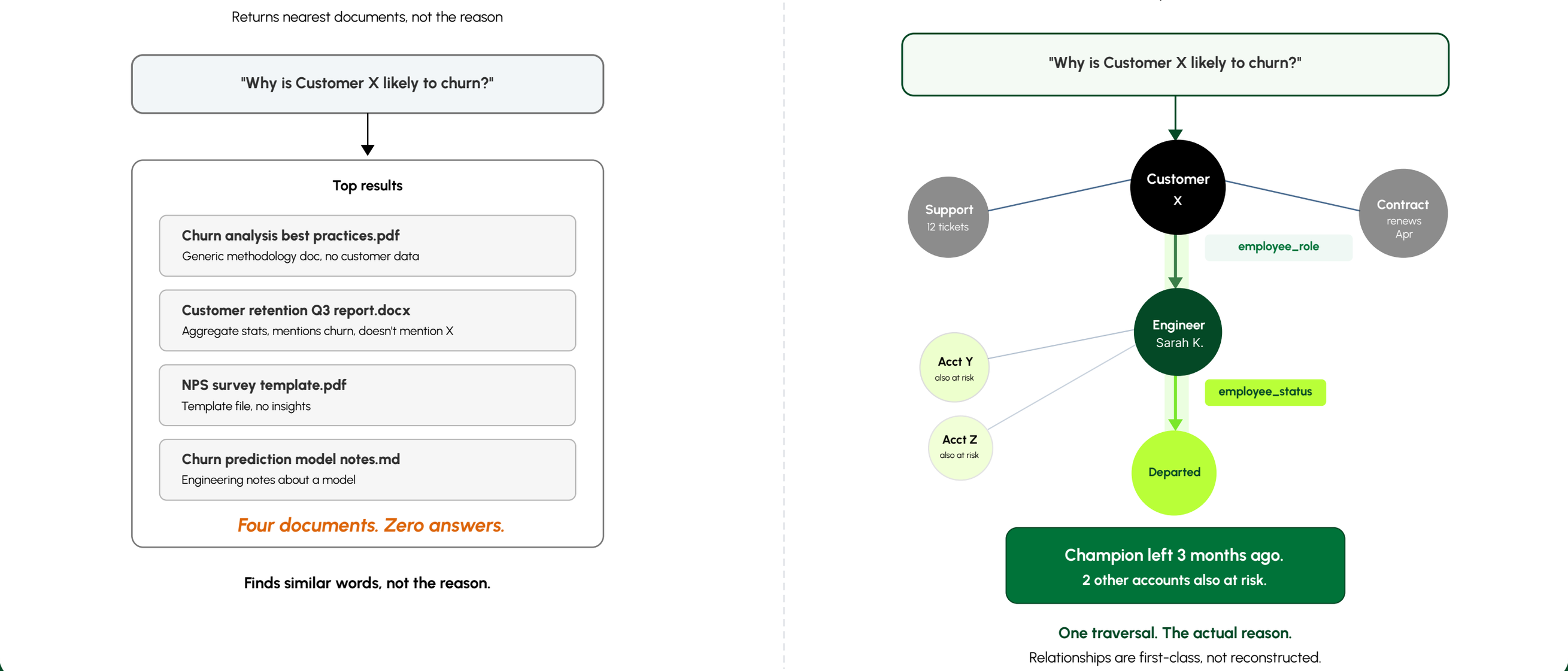


Figure 3.2 — Entities connected by explicit, native relationships instead of ad-hoc joins at query time.

What this gives you

- Multi-hop reasoning at query time, not pre-computed
- Pattern detection across the entity graph
- Path explanations attached to every answer

In Arango

- Graph, document, and vector data unified in one store. No joins across systems
- AutoGraph continuously builds and maintains the entity graph as your data changes
- GraphRAG retrieves answers by traversing relationships, not just matching text

Failure mode

Relational and vector-only stacks flatten relationships into text chunks or join tables. A 3-hop question like "which customers depend on services owned by people who left in the last quarter?" becomes a chain of manual workarounds: complex SQL, repeated fetches, or similarity re-ranking.

Each approach is brittle, built for one query shape, and breaks quietly the moment the underlying data structure changes.

How Arango solves it

Arango stores relationships natively, not as join tables or text approximations, but as direct connections between entities.

That means a multi-hop question traverses the graph in milliseconds rather than executing a chain of joins. And because relationships carry their own attributes (type, weight, time period), a single query can ask not just who is connected, but how, when, and under what conditions.

Requirement 03

Freshness & temporal correctness

Context must reflect what is true now — and what was true at time T.

Time is the thing most AI systems handle worst. The index was built last quarter; the graph updates nightly; the warehouse lags a day. Agents reason across this drift and quietly produce wrong answers. Persistent context carries an evolving state rather than reconstructing it. Bi-temporal modeling tracks both when an event occurred (valid time) and when it was recorded (transaction time) — so "what is true now" and "what was true at time T" come from the same model.

Valid time
when it happened in the world

Transaction time
when the system learned about it

Query at T
state as-of, or state as-known-at

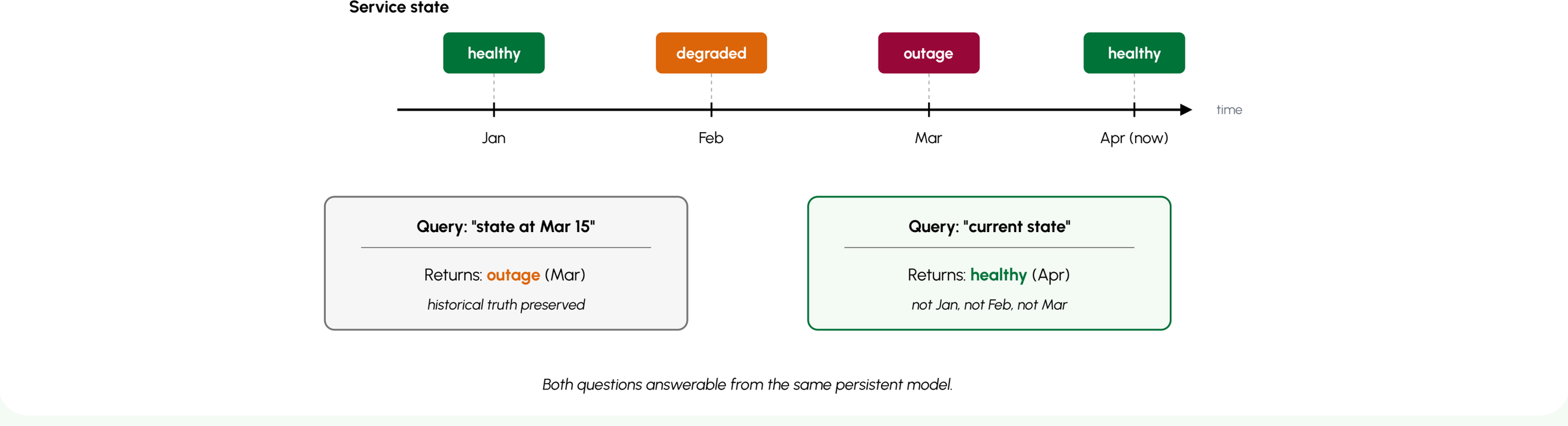


Figure 3.3 — Current state vs. historical state, answerable from the same layer.

What this gives you

- Awareness of current vs. historical state
- Tracking of changes, sequences, and events
- Time-aware querying and reasoning

In Arango

- AutoGraph continuously updates the graph as data changes. No manual refresh required
- AQL queries any point in time: current state, historical state, or the delta between them
- Temporal context is built into the data model, not added as a logging layer

Failure mode

Freshness gaps between stores mean agents answer "what is true" with stale state. Reconciling a warehouse-lagged record against a live graph requires ad-hoc reconciliation code — which itself becomes stale.

Historical queries ("what did we know on March 3rd?") are almost always impossible without rebuilding from logs.

How Arango solves it

Arango tracks two things simultaneously: what is true in the world right now, and what the system knew at any given point in time.

That means you can query the actual state of a customer, service, or incident at any moment in history — not just what was recorded, but when it was recorded. AutoGraph ingests continuously, so freshness is a structural property of the platform — not an operational scramble after the fact.

Requirement 04

Provenance & trust

All data must be traceable, governed, and explainable.

An AI output without lineage is not defensible. A SOC 2 audit, an EU AI Act review, or a regulator asking why a customer was denied all require the same thing: traceability back to source.

Every output
Carries a citation set traceable to source records

Every field
Access-controlled at the engine, not the API gateway

Every write
Logged with source, timestamp, and transformation chain

Every query
Emits an explainable plan alongside its results

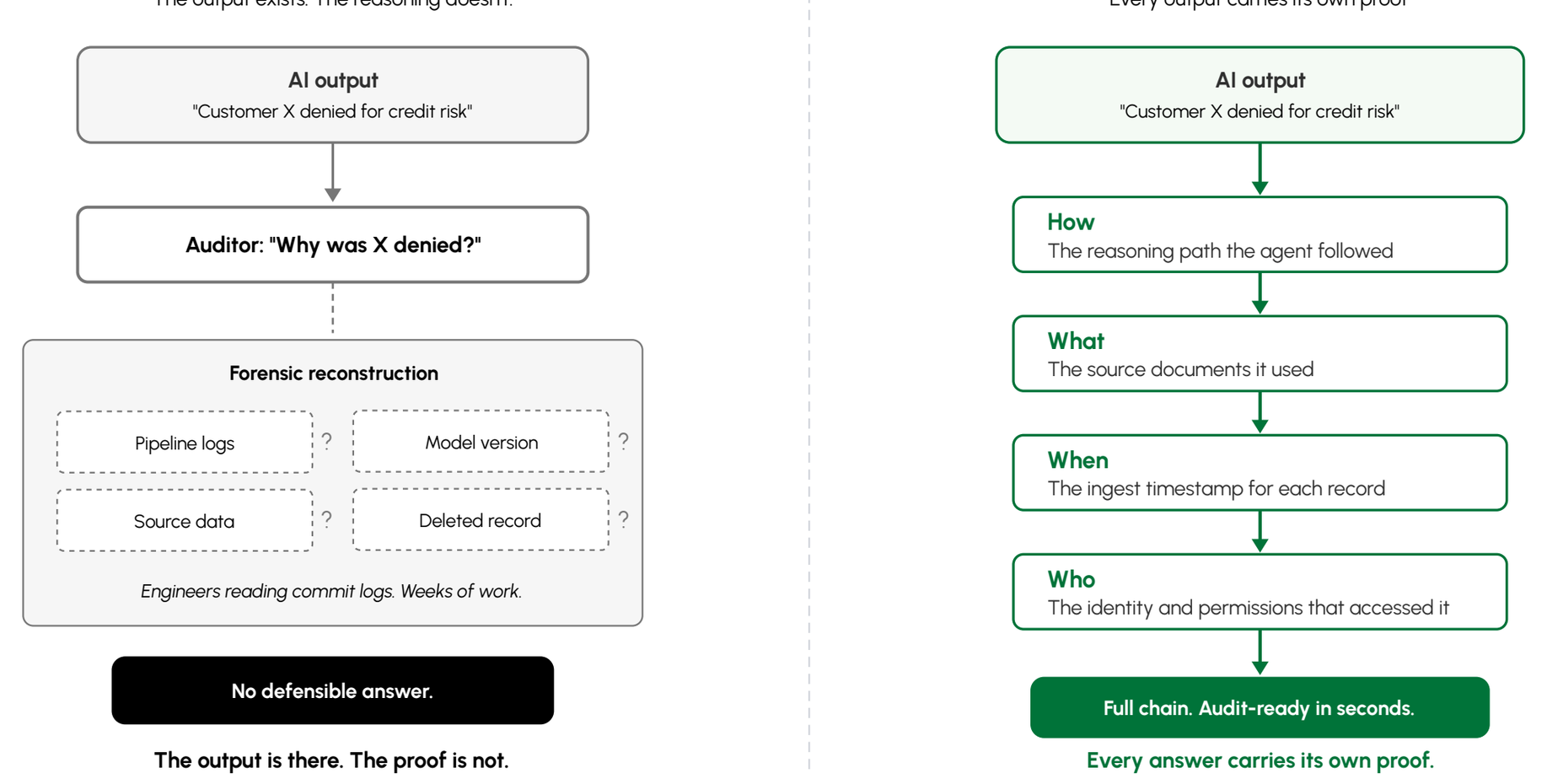


Figure 3.4 — Provenance and trust

What this gives you

- Source attribution on every AI output
- Field-level access control, enforced in the engine
- Audit-ready logs across queries, edits, and ingests

In Arango

- RBAC, audit logging, and lineage built in
- Query traceability across AQL execution
- Compliance-ready deployment modes (incl. air-gapped)

Failure mode

When provenance lives outside the data layer, it goes stale within weeks. Pipelines fork, records get re-derived, and the lineage graph diverges from reality. Audits then require forensic reconstruction — engineers reading commit logs to explain why a model cited a document that no longer exists.

How Arango solves it

Lineage is captured at write time and at query time. Every record carries source, ingest timestamp, and transformation chain; every AQL execution emits a query plan and citation set returned alongside results. RBAC is enforced at document, edge, and field level. Audit logs are immutable and exportable for SOC 2, ISO 27001, HIPAA, and EU AI Act review.

Requirement 05

AI-native service layer

Context must be directly consumable by AI systems.

Most teams underestimate this one: they build a perfectly good context store, and then every AI application re-implements retrieval, ranking, citation, and grounding on top. The service layer makes those four verbs platform primitives.

Retrieve
Pull relevant context adaptively — graph, vector, or hybrid per query.

Rank
Order results by relevance, recency, and graph distance in one pass.

Cite
Return source IDs and traversal paths alongside every response.

Ground
Give agents the data and the explanation, not just the answer.

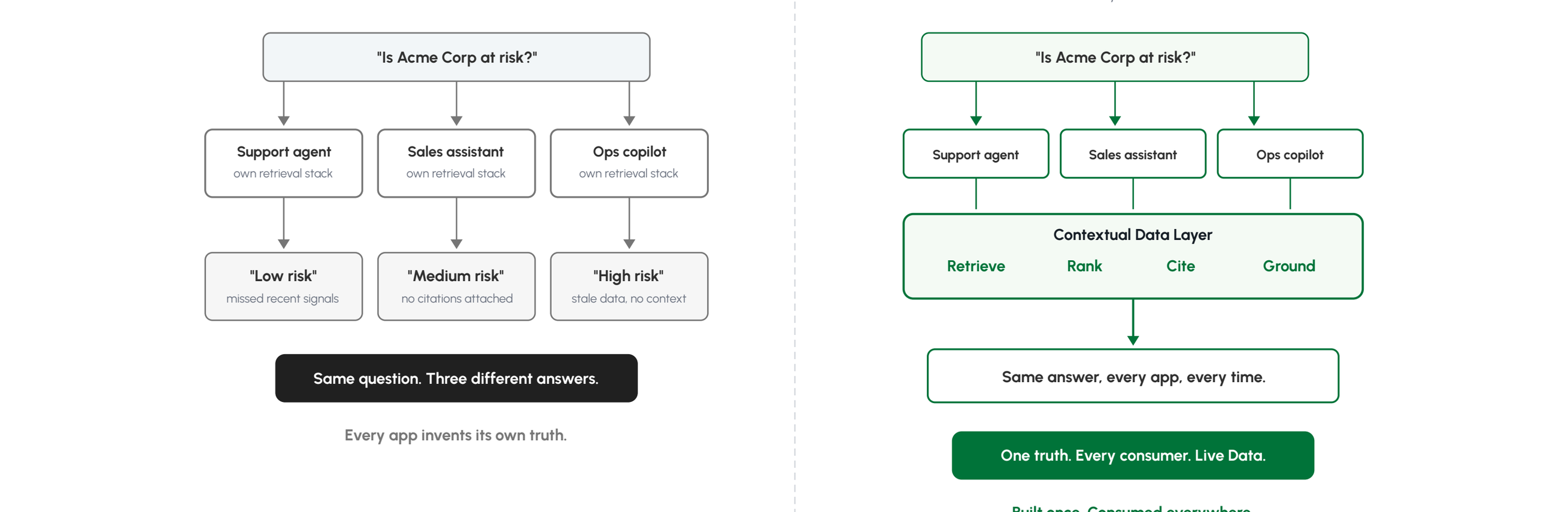


Figure 3.5 — AI-native services, centralized once instead of rebuilt per app.

What this gives you

- Every agent retrieves, ranks, cites, and grounds answers from the same platform layer. No custom retrieval code per application.
- Agent-ready interfaces via MCP, native APIs, and direct integration
- The right retrieval strategy selected automatically for every query

In Arango

- AutoGraph determines the optimal ingestion approach and graph structure per data domain — so your context graph is retrieval-ready from day one
- Deep Search decomposes each query into targeted subqueries and selects the appropriate RAG strategy for each GraphRAG, HybridRAG, or VectorRAG for optimal results
- MCP server, native APIs, and BYOC (Container, Code) for teams bringing their own models

Failure mode

When every application reimplements its own retrieval stack, retrieval logic forks across the org. Different chatbots return different answers to the same question because each one tunes chunk size, top-k, and re-ranker independently. Citation drift sets in; the audit trail no longer agrees with itself.

How Arango solves it

The service layer exposes retrieval, ranking, citation, and grounding through native APIs and an MCP server, so every agent and application consumes context the same way. Deep Search handles the hard part: it reads the intent behind each question, selects the right retriever (graph traversal, vector similarity, or hybrid), and returns a single governed response.

Requirement 06

Unified multimodel data platform

Context must be unified across all data types in a single model.

The mistake most teams make is assuming they have to pick one or two data representations and work around the limits of the rest. The Frankenstack tax — sync lag, no cross-model transactions, five access policies — is permanent.

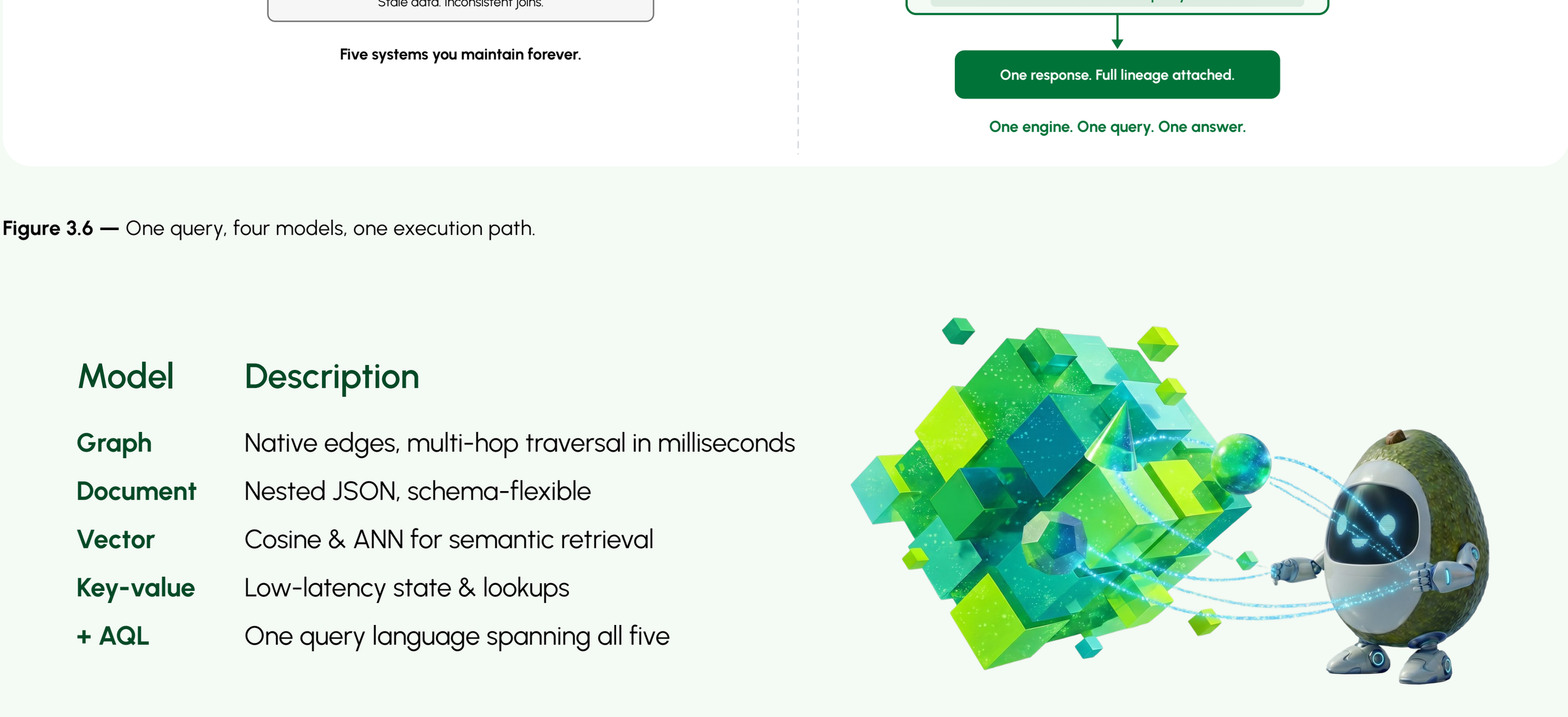
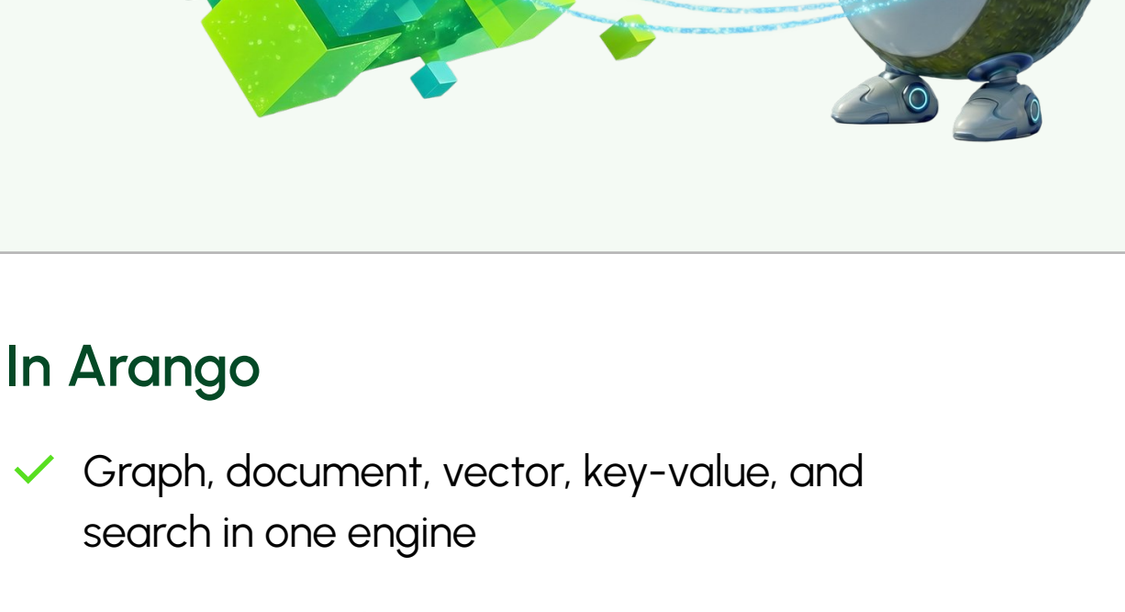


Figure 3.6 — One query, four models, one execution path.

Model	Description
Graph	Native edges, multi-hop traversal in milliseconds
Document	Nested JSON, schema-flexible
Vector	Cosine & ANN for semantic retrieval
Key-value	Low-latency state & lookups
+ AQL	One query language spanning all five



What this gives you

- One platform, one query language, one governance model
- Cross-model transactions and consistent reads
- Lower operational and integration overhead

In Arango

- Graph, document, vector, key-value, and search in one engine
- AQL spans all four in a single execution path
- Unified RBAC, audit logging, and lineage

Failure mode

Stitched stacks (Postgres + Neo4j + Pinecone + Mongo + Elasticsearch) carry a permanent tax: sync lag, no cross-model transactions, five credential sets, and application code fanning out and re-merging results on every request. Latency compounds; consistency degrades; the team runs a distributed systems project instead of an AI project.

How Arango solves it

Graph, document, vector, key-value, and search live in one data engine and share one query language. A single AQL statement can traverse a customer subgraph, filter by document fields, rank by vector similarity, and join to a key-value lookup — in one execution path, with one transaction boundary, one access policy.

Architecture defines requirements. Requirements define AI outcomes. Miss any of the six, and agentic AI will hit a ceiling you can't get past by upgrading the model.

Chapter 4

Case studies

Case study 01 · SaaS support

AI-powered service & support

A global SaaS provider managing 30,000+ tickets per day across Tier 1 through Tier 4 was suffering not from a lack of data but from fragmentation. Support agents navigated across disconnected systems — ITSM, observability, knowledge base, CRM — and manually reconstructed context per incident. The organization implemented a contextual data layer on Arango — ingesting all relevant data into a single platform and continuously contextualizing it, creating a living context graph that the AI support agent operated against directly.

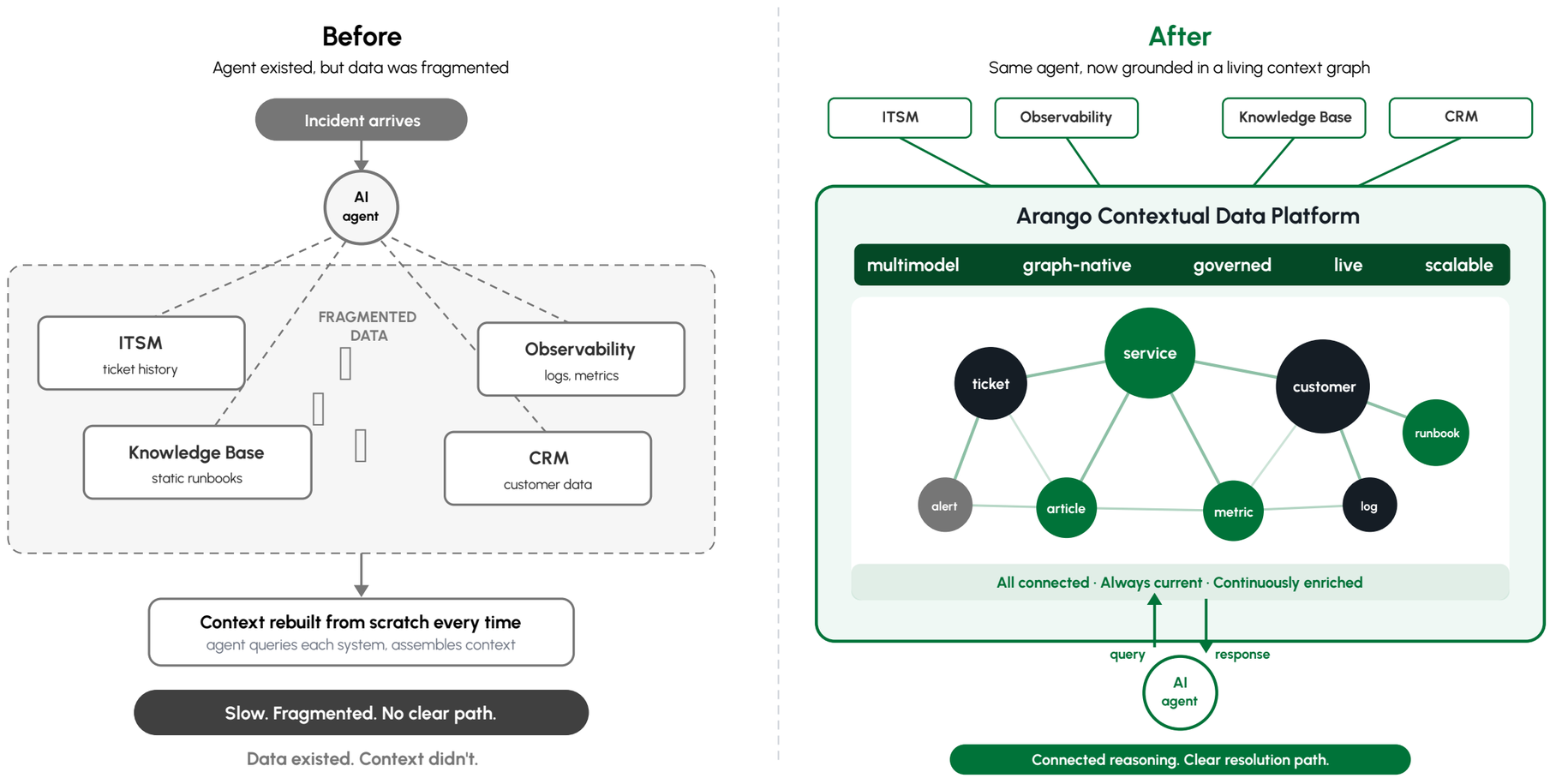


Figure 4.1 — The same sources, flowing into one governed context layer.

Outcomes

72 → 24
hours: Tier 4 MTTR, before and after

30 → 12
minutes: Tier 1 MTTR, before and after

40–60%
reduction in manual triage effort

30K
support tickets per day, now AI-grounded

Case study 02 · AI agent

PSI Clinical Research

PSI is a global clinical research organization. Selecting the right trial sites is one of the most expensive decisions in drug development: trials can take a decade and cost hundreds of millions. The data existed already — detailed records on investigators, institutions, protocols, and historical outcomes — but it was fragmented.

PSI built SYNETIC, an AI-enabled knowledge system powered by the Arango Contextual Data Platform, unifying structured and unstructured records into one continuously maintained context graph.

"Our AI agent doesn't just recommend trial sites — **it explains why**, with the data and relationships that led to the recommendation."

— Andrei Seryi, Director of Knowledge Management and Process Improvement, PSI

Outcomes

6 wks → min
site identification time

100Ks
research projects unified

Millions
per trial saved

Chapter 5

Additional use case examples

The same pattern, across industries. Three short examples show how a contextual data layer supports production AI where relationships, time, governance, and multimodel data all matter at once.

01

Financial Services

Real-time fraud detection

Time · Relationships

An AI fraud-detection agent monitors transactions and surfaces high-risk patterns to analysts through real-time scoring and traversal.

GraphRAG connects accounts, transactions, and merchants across siloed systems; temporal modeling flags synthetic identities through 6-degree relationship patterns before settlement.

Outcome

Fraud rings flagged **before settlement** — full relationship trace on every alert.

02

Networking · HPE Aruba

Global network operations on one platform

Time · Multimodel

HPE Aruba delivers secure networking for millions of devices worldwide. Managing that scale used to mean six siloed databases. They consolidated onto the Arango Contextual Data Platform — unifying graph, document, and key-value — and retired six systems.

Outcome

6 → 1 database systems. One platform, one governance model.

03

Regulated enterprise

Audit-ready compliance reporting

Provenance · AI-ready

An AI compliance assistant lets auditors and risk teams query policies, trace decisions, and generate audit-ready documentation. Arango RBAC and provenance ensure every recommendation cites source documents and policies; AutoRAG delivers auditable responses.

Outcome

Regulatory audits passed with **zero findings.**

Chapter 6

The Arango Contextual Data Platform

How the Arango Contextual Data Platform is architected, what runs in each of the three layers, and how it maps to each of the six requirements.

Arango Contextual Data Platform 4.0 delivers all six architectural requirements in one integrated system. Rather than assembling specialized tools at query time, it persists context in a multimodel substrate and serves it to AI through native, agent-ready interfaces — including 20+ AI services, an MCP server, and natural language interfaces for direct agent interaction.

Build enterprise context once.
Reuse it everywhere.

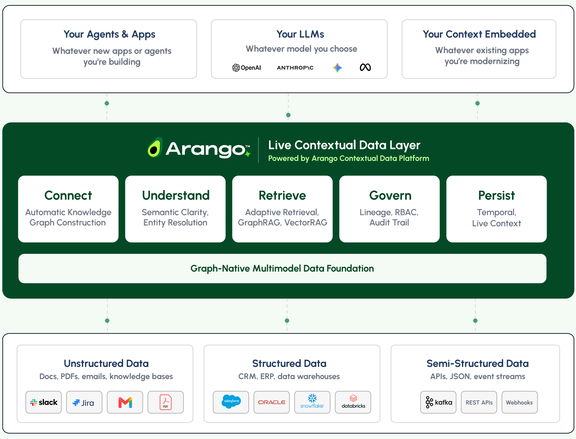


Figure 6.1 — Arango Contextual Data Platform — architecture overview.

How Arango delivers each contextual data layer requirement

Semantic clarity	AutoGraph builds domain-aware knowledge graphs. AQL enforces consistent querying across modalities.
Relationships	Native graph model. AutoGraph and GraphRAG for relationship-aware retrieval
Freshness & time	Temporal modeling within documents and graphs. AQL for time-based filtering. Continuous updates via AutoGraph.
Provenance & trust	RBAC, audit logging, and lineage. Query traceability across AQL execution.
AI-native service	Deep Search for adaptive retrieval. GraphRAG and HybridRAG for context-rich responses. MCP server and APIs.
Multimodel coverage	Graph, document, vector, key-value, and search in one system — AQL spans all five in a single execution path.

Under the hood

01
AutoGraph
How AutoGraph creates context
Knowledge graphs traditionally take weeks of manual work — defining ontologies, resolving entities, mapping relationships. AutoGraph automates construction directly from enterprise data, maintaining the graph as sources change.

Automated · Continuous · No ontology design

02
AQL
One query, four models
AQL spans graph, document, vector, key-value, and search in one execution path. A single query can traverse a customer graph, filter by document fields, rank by vector similarity, and return grounded results with lineage.

Graph · Vector · Document · Key-value · Search

03
AutoRAG
How AutoRAG picks a retrieval strategy
When a question arrives, Deep Search scans relevant topics, breaks the request into targeted subqueries, selects the right retrieval strategy for each based on data type and domain, and aggregates the results into a single unified response — more complete and trustworthy than single-pass retrieval alone.

Subquery Routing · GraphRAG · VectorRAG · Adaptive

04
Deployment
How it deploys
Kubernetes orchestration, fine-grained RBAC, Bring Your Own Container (BYOC) for custom models, and cloud, on-prem, hybrid, managed (AMP), and air-gapped deployment modes for regulated environments.

Cloud · On-prem · Hybrid · Air-gapped

Resilience & disaster recovery

3×
Replication
Configurable replication factor ensures data survives node failures.

<10s
Automatic failover
Leader election completes in under 10 seconds; clients retry transparently.

5-min
Point-in-time recovery
Continuous backup with 5-minute RPO; restore to any timestamp.

Live
Health monitoring
Prometheus metrics for query latency, error rates, and replication lag.

0
Schema evolution downtime
Add fields, indices, and edge types without blocking writes.

Before
- ✕ Separate backup procedures for graph, vector, and document databases
- ✕ Restoring consistent state required manual coordination
- ✕ Multi-hour downtime per recovery event

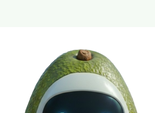
After
- ✓ Platform snapshots include all models atomically
- ✓ Point-in-time restore to any 5-minute window
- ✓ Recovery tested monthly — measured RTO: 12 minutes

Value drivers & benchmark results

30–50%	**2–4×**	**25–40%**
reduction in integration complexity	faster AI development cycles	lower architectural overhead

Decision accuracy also improves by 20–35% when AI systems reason over connected, governed context rather than retrieval from fragmented stores.

Remember: Arango delivers all 6 requirements in one platform — automated context modeling, adaptive retrieval, and agent-ready interfaces, on top of a native multimodel substrate.



Chapter 7

Why 2 of the 3 enterprise AI architectures hit a ceiling

Most enterprise AI architectures fall into one of three categories. Each solves the problem differently. Only one delivers all six requirements in a single system.

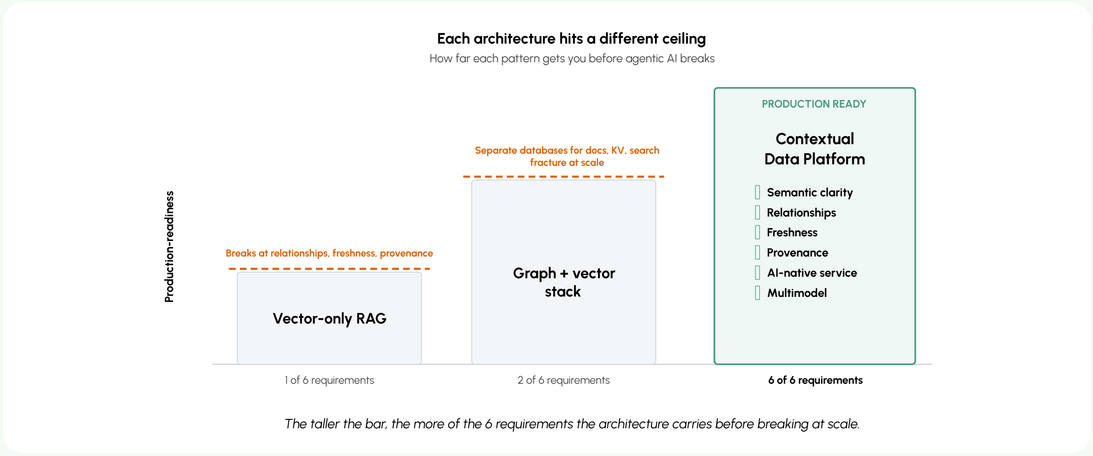


Figure 7.1 — Each architecture hits a different ceiling. The taller the bar, the more of the 6 requirements the architecture carries before breaking at scale.

Scored against the 6 requirements

Requirement	Vector-only RAG	Graph + vector stack	Unified multimodel
Semantic clarity	Embeddings only. No shared definitions.	Enforced for graph + vector. Document, key-value, search rely on upstream cleanup.	Shared entity definitions persist across all five models.
Relationships	Not modeled. Inferred from text similarity.	Native in graph + vector. Glue code into document and key-value.	Native alongside every other model. One query.
Freshness & time	Index refreshes on its own schedule. Stale.	Graph + vector share cadence. Rest drift at the seams.	Current and historical state in one model. No drift.
Provenance & trust	Source chunk is traceable. Reasoning is not.	Lineage holds inside graph + vector. Breaks crossing systems.	Lineage and RBAC unified across all five models.
AI-native service	Every app writes its own retrieval code.	Retrieval lives in LangChain-style glue once queries cross systems.	AutoRAG + MCP server built into the platform.
Multimodel coverage	One model. Vector.	Two models in one system. Rest live elsewhere.	All five: graph, vector, document, key-value, search.

Key takeaway: Vector-only RAG and graph + vector both ship demos. Only a contextual data platform carries all six requirements in one system — which is what production agentic AI requires.

Objections worth taking seriously

01 Why not just use Postgres with pgvector?

Postgres with pgvector handles two of the five data models (relational and vector). Graph, document, and search live elsewhere — and the integration seams return.

02 Our graph database added vector support. Isn't that enough?

Graph + vector is a real improvement over older graph-only architectures. It still leaves document, key-value, and search outside the system and re-creates seams at enterprise scale.

03 Why not wait for LLMs to get good enough that context doesn't matter?

The gap is not the model. It is grounding. LLMs produce better answers when they have access to your specific data, relationships, governance, and policies — not generic training. That is what a contextual data layer provides.

Closing

Frankenstacks were not built for this.

Enterprise AI is not failing because the models are bad. It fails because the infrastructure underneath was assembled for a different era: repeatable workflows, static schemas, predictable queries. That infrastructure was never built to support AI agents that reason, decide, and act across systems.

The gap is not intelligence. It is grounding. Giving AI the specific data, relationships, governance, and policies of your business so it can produce responses that are relevant, accurate, and defensible. A better model without grounding produces more confident wrong answers. A grounded model reasons.

These six requirements are not optional: semantic clarity, relationships, freshness, provenance, an AI-native service layer, and unified multimodel coverage. If your current architecture delivers fewer than six natively, you have a ceiling. A better model will not raise it.

Where you go from here depends on where you are starting.

If you are a medium sized company without a sprawling data stack yet, the lesson is simpler: do not build a Frankenstack in the first place. Start with a contextual data layer and grow from there.

Most enterprises don't need to start over. They need to close the gaps. Your existing investments in data warehouses, operational systems, and pipelines don't disappear. Arango connects to what you have, contextualizes it, and fills in what's missing: relationships, freshness, provenance, unified retrieval. You keep what works. You replace what's creating the gaps. The goal is not a clean-room architecture. It is a working contextual data layer where all six requirements are delivered from a single system, built incrementally, on top of what you already have.

Three things to do this week

01 Map architecture gaps

Score your current AI stack against the 6 requirements. Identify which are delivered natively, which are reassembled at query time, and which are missing entirely. Most teams find two or three gaps they've been working around for months.

02 Trace your failures

Take your last three production AI failures and trace them to the data layer. Every inconsistent answer, every unexplainable decision, every agent that worked in the demo and broke in production. In almost every case the root cause is not the model. It's missing context, stale retrieval, or a seam between systems nobody owns. Name the failure mode. Name the gap it maps to.

03 Make the call

Define what production-ready means for your program. Not "working in a sandbox." Not "accurate enough for a demo." Production-ready means consistent, explainable, auditable, and scalable under real load. Decide whether your current architecture can get you there, or whether the ceiling it hits is the reason your AI program hasn't shipped.

You've seen the gaps. We built the platform that closes them.

We'll map your architecture against the six requirements and show you where the ceiling is, and what it would take to move past it.

Talk to Us

Appendix

Glossary and Sources

Glossary

Agentic AI
AI systems that perceive their environment, reason over changing inputs, and take multi-step actions to achieve goals, rather than responding to a single prompt.

AQL (Arango Query Language)
A single query language that spans graph, document, vector, key-value, and search data in one execution path.

AutoGraph
Automatically builds and maintains your knowledge graph from enterprise data. Discovers entities, maps relationships, and determines the right graph structure for retrieval — no manual ontology design required.

AutoRAG
Determines the optimal ingestion approach and graph structure for your data so that context is retrieval-ready before any query runs. The quality of retrieval is decided at ingestion time, not at query time.

Contextual data layer
A persistent, multimodel architectural tier that unifies how enterprise data is modeled, connected, governed, and retrieved for AI.

Deep Search
At query time, understands the intent behind each question, breaks it into subqueries, and automatically applies the right retriever for each one — GraphRAG, HybridRAG, or VectorRAG — then aggregates the results into a single governed response.

Frankenstack
An enterprise AI architecture assembled from multiple specialized systems (vector store, graph platform, search index, orchestrator) stitched together at query time.

GraphRAG
Retrieval-augmented generation that uses a knowledge graph to answer multi-hop relationship questions, going beyond vector similarity.

HybridRAG
Retrieval combining semantic similarity (vector) with structured relationships (graph) in one pass.

Knowledge graph
A model that represents entities and their relationships as connected nodes and edges. One component of a contextual data layer.

Model Context Protocol (MCP)
The industry standard for how AI agents discover and invoke tools and data sources through a consistent, vendor-neutral interface. Backed by Anthropic, OpenAI, Google, and Microsoft, and now governed under the Linux Foundation.

Multimodel platform
A data platform that stores and queries multiple data representations (graph, document, vector, key-value, search) natively.

Provenance
The record of where a piece of data originated, how it was transformed, and how it was used — so AI outputs can be traced back to source.

RAG (retrieval-augmented generation)
A pattern for grounding LLM outputs in external data by retrieving relevant content at query time.

Semantic layer
An architectural tier that codifies business meaning and relationships so AI systems interpret data consistently. One dimension of a contextual data layer.

Temporal state
The dimension of context that distinguishes what is true right now from what was true historically.

Sources

1. Tavakoli, A., Goodman, B., Soller, H., Rowshankish, K., et al. "Building the foundations for agentic AI at scale." McKinsey, April 2026. https://www.mckinsey.com/capabilities/mckinsey-technology/our-insights/building-the-foundations-for-agentic-ai-at-scale
2. Sheng, E., Zhu, R., O'Rourke, B., Pedzinski, D., Zhang, K. "The Three Layers of an Agentic AI Platform." Bain & Company, April 2026. https://www.bain.com/insights/the-three-layers-of-an-agentic-ai-platform/
3. Sheng, E., Zhu, R., et al. "Governance, Trust, and the Data Foundation." Bain & Company, April 2026. https://www.bain.com/insights/governance-trust-and-the-data-foundation/
4. "The Forrester Data, AI, and Analytics Architecture Model." Forrester, October 2025.
5. Marwaha, R. "Arango Introduces Contextual Data Platform 4.0." Arango Blog, March 16, 2026. https://arango.ai/blog/arango-introduces-contextual-data-platform-4-0/
6. "PSI Reduces Clinical Trial Site Identification From Weeks to Minutes Using AI Powered by a Contextual Data Layer." Arango Customer Story, 2026. https://arango.ai/case-studies/psi/
7. "HPE Aruba Networking: From six siloed databases to one trusted data platform." Arango Customer Story, 2026. https://arango.ai/case-studies/hpe-aruba-networking/