Nick Clark · Published April 25, 2026 · PDF

How to Give AI Agents a Memory They Can Query by Meaning

If you are building agents, you have probably reached for a vector database to give them memory, then watched them retrieve confidently wrong passages with no record of why. This guide describes a different architecture: agent memory as a governed traversal over a scoped, anchor-indexed structure, where every step of a query narrows meaning, updates state, and checks admissibility at once. It is an architecture disclosed in United States Patent Application 19/647,395, not a shipping library, and its home inventive step is the Semantic Discovery inventive step. You build it yourself.

What You Are Building

You want an AI agent to remember things and to find them later by what they mean, not by exact keyword and not merely by the closest vector. A developer typing "how to give AI agents a memory they can query by meaning" usually already has embeddings and a vector store and is unhappy with the result: the agent retrieves plausible-looking text that does not actually support its answer, there is no trace of how a result was reached, and access control is a filter bolted on after retrieval.

What you are building is a memory that is a place the agent walks through, rather than a table it queries once and forgets. Each addressable thing the agent knows lives in a scoped container. A query becomes a small persistent object that carries its own intent, context, accumulated findings, and governance rules, and moves through that structure one boundary at a time. At each boundary it narrows what is relevant, records what it learned, and checks whether the step is allowed. The result is memory that is queryable by meaning and by lineage, meaning you can ask both "what is relevant here" and "how did we get to this."

This is the approach disclosed as the Semantic Discovery inventive step in United States Patent Application 19/647,395. This guide teaches the architecture. It is not a package you install.

Why the Obvious Approaches Fall Short

The standard move is a vector database. You embed every document, embed the query, and return nearest neighbors. This works and is genuinely useful, and for many applications it is enough. But it has structural properties worth naming honestly.

Nearest-neighbor retrieval is a global similarity computation: the query is scored against the whole corpus and the top matches come back. Similarity is not grounding. A passage can be semantically close to a query and still fail to support the answer the agent then generates, which is one root of retrieval-augmented hallucination. The retrieval step and the reasoning step are separate subsystems joined by a serialized interface, and semantic context is lost at that boundary.

There is also no path record. A vector store returns items, not the route to them. You cannot ask the memory why a given item was surfaced, what intermediate reasoning led there, or which items were considered and rejected. Provenance, when it exists, is reconstructed after the fact.

Finally, governance is a post-hoc layer. Access control, freshness, and policy typically run as a filter over results the ranker already produced. Possession of an item's identifier, or a broad enough query, tends to surface it; the permission check happens late. None of these are moral failings of vector search. They are consequences of treating the index as a passive lookup target rather than as something the query moves through under rules.

The Architecture

The disclosed approach makes the index an active substrate that a query traverses, and fuses search, inference, and governance into one atomic step. Every mechanism below traces to United States Patent Application 19/647,395.

Anchor-indexed containers. Every addressable semantic object, whether content, a knowledge node, an agent, or a service endpoint, is assigned to a nested container governed by an anchor. Each anchor carries a mutation policy, a quorum threshold, an alias mapping, and lineage metadata. The index is decentralized and hierarchical: there is no single central table that maps meaning to location.

The discovery object. A query does not enter as a string, a keyword list, or a lone embedding. It is instantiated as a persistent, memory-resident object that lives for the duration of the traversal. The specification describes it with typed fields: an **intent** field (a structured objective with goal type, domain scope, resolution criterion, and specificity constraints, not a natural-language string); a **context** block (domain, temporal scope, epistemic conditions, audience, privacy constraints); a **memory** field (accumulated findings and intermediate conclusions as structured records); a **policy reference** field (the governance constraints in force); a **lineage** field (the ordered record of admitted transitions); plus affective-modulation and confidence fields drawn from other chapters of the disclosure. This object is the subject of the traversal and its running memory at the same time.

The three-in-one traversal step. At each anchor boundary, the object undergoes three coupled phases that cannot be separated. The **search** phase evaluates the object's current state against the anchor's published reachable neighborhood and produces a candidate transition set. This is local, not global: the object is compared to what this anchor advertises as reachable, not to the whole corpus, so the search space narrows as you go deeper. The **inference** phase scores or selects among those candidates and updates the object's state, refining intent, extending memory, and adjusting confidence. The **execution** phase decides admissibility under policy, lineage continuity, entropy bounds, and temporal validity, producing one of three outcomes: **admit** (advance), **reject** (try another candidate, backtrack, or terminate), or **decompose** (break the transition into finer sub-transitions and re-evaluate). Every determination, including rejections, is written to lineage.

Neighborhood publication. Each anchor maintains its own dynamic, policy-scoped description of what is reachable from it: a semantic content descriptor, a reachability graph of navigable sub-anchors and peers, a policy envelope, a freshness indicator, and an entropy summary. Crucially it is scoped to the requester. Two discovery objects with different policy profiles can receive different publications from the same anchor, because the anchor restricts what each is authorized to see. The anchor computes this itself, without a central authority, and recomputes it when its container mutates.

Structured aliases resolved by walking. Objects are addressed by human-readable aliases of the form `type@domain.subdomain/path`, for example `article@org.wikipedia/computing`. An alias is resolved by traversing its path segment by segment through the live index, the same mechanism a discovery object uses, not by consulting a lookup table. Two consequences the specification draws out: resolution always reflects the current index state, and possessing an alias does not confer access, because each segment is still subject to the same traversal governance.

Query-specific, lineage-carrying relevance. Because relevance here is the admissibility-verified path that reached an object, not a precomputed global score, the same object can be reached by different paths for different queries or not reached at all. The path is the provenance. This is the sense in which the memory is queryable by meaning and by lineage: an answer arrives with the sequence of transitions, state snapshots, and admissibility decisions that produced it.

How to Approach the Build

You are implementing this yourself. Here is a reasonable order.

1. **Model the anchor and its container.** Start with the smallest unit: an anchor that owns a set of objects and sub-anchors and holds a mutation policy, an alias mapping, and lineage metadata. Get containment and hierarchy working before anything semantic.
2. **Give each anchor a neighborhood publication.** Have the anchor compute a description of what is reachable from it, scoped to the requester's policy profile. Treat this as the interface the search phase reads. The illustrative shape below is faithful to the disclosed components and is a sketch to think with, not production code:

```
NeighborhoodPublication {
  semantic_content_descriptor  // abstracted description of the territory
  reachability_graph           // navigable sub-anchors/peers + relationsl
  policy_envelope              // constraints on entities traversing here
  freshness_indicator          // epoch of last update
  entropy_summary              // diversity / update-rate / density
}
```

3. **Define the discovery object as a typed record**, with the intent, context, memory, policy-reference, and lineage fields above. Make intent structured, not a raw prompt. This object persists across steps; do not rebuild it each hop.
4. **Implement the three-in-one step as one indivisible transition.** Search produces candidates from the local publication; inference scores and mutates state; execution returns admit, reject, or decompose. Resist the temptation to let a component skip the governance phase, because the whole benefit is that no transition commits without it.
5. **Choose an inference engine per anchor, and keep it small.** The specification is explicit that the engine is model-agnostic: an embedding-similarity scorer, a rule matcher, a graphical model, a neural ranker, or a language model all qualify. Because each anchor only sees the scoped local transition problem, current intent, the neighborhood publication, and the candidate set, a small model with a narrow context window can serve. You are not assembling a growing prompt.
6. **Write everything to lineage, including rejections.** Record the anchor, timestamp, mutation, admissibility determination, and the neighborhood the candidate came from. This is what makes the memory auditable and is also, per the disclosure, a signal you can later feed back into how anchors split, merge, or migrate.
7. **Pick a termination criterion per mode.** The disclosure describes three modes over one substrate: human search (stop when enough source-grounded objects satisfy intent), agent reasoning (stop when a complete, admissibility-verified reasoning chain exists), and answer synthesis (continue to a grounded natural-language rendering, itself subject to admissibility). Decide which you need; the traversal mechanics are shared.
8. **Handle failure paths deliberately.** A traversal can end by reaching resolution, by confidence collapse, by policy prohibition, or by depth exhaustion. Backtracking and decomposition are first-class, not error handling. If you later run multiple

discovery objects, the disclosure describes an optional policy-governed merge when compatible intents intersect at an anchor, which is worth deferring until the single-object path is solid.

What This Does Not Give You

This is an architecture, not a drop-in library. There is no package to install, no SDK, and nothing here "just works" out of the box. You are responsible for the anchor store, the neighborhood-publication computation, the traversal engine, the governance rules, and the inference engines at each anchor.

It is not benchmarked or productized here. The specification describes properties by construction, such as bounded per-step governance cost and locality of search, but this guide reports no performance numbers, latency figures, or accuracy claims, because the disclosure states none you should treat as measured results. Treat expected behavior as something you must verify in your own build.

It is also not always the right tool. If your corpus is small, static, and low-stakes, or you do not need path-level provenance or per-requester governance, a plain vector store is simpler and probably sufficient. The traversal architecture earns its complexity when you need meaning-based memory that also carries lineage, enforces policy at every step rather than after the fact, and must remain auditable. Building a full anchor hierarchy with scoped publications and a governed traversal engine is real engineering, and the honest tradeoff is that you trade the one-call convenience of nearest-neighbor retrieval for grounding, provenance, and governance you have to implement.

Disclosure Scope

The architecture and mechanisms described here, including the anchor-indexed adaptive index, the discovery object, the three-in-one traversal step, scoped neighborhood publication, navigational alias resolution, and traversal-based relevance,

are disclosed in United States Patent Application 19/647,395 as the Semantic Discovery inventive step. This guide is educational: it explains an approach a developer can study and build. It is not a grant of rights, not a warranty of fitness or performance, and not an offer of software or of any downloadable implementation. Every characterization above is drawn from the filed disclosure; where the disclosure states no benchmark or guarantee, none is implied here.

Semantic Discovery (/semantic-discovery)

All 40 steps → (/inventive-steps)

Search, inference, and execution as one governed step.

Chapter 10 (/patents/19-647395/chapters/discovery)

PRIMARY TECHNICAL DISCLOSURE

- Governed Semantic Discovery: Search, Inference, and Execution Through Adaptive Traversal (/articles/governed-semantic-discovery-search-inference-and-execution-through-adaptive-traversal)

SECONDARY TECHNICAL

- The Adaptive Index as Unified Search-Inference-Execution Substrate (/articles/semantic-discovery/unified-substrate)
- Three-in-One Traversal: Search, Inference, and Execution in a Single Bounded Step (/articles/semantic-discovery/three-in-one-traversal)
- The Discovery Object: A Traversal-Native Semantic Agent (/articles/semantic-discovery/discovery-object)
- Post-PageRank Semantic Ranking: Relevance Through Governed Traversal (/articles/semantic-discovery/post-pagerank)
- Persistent Semantic State: Eliminating Prompt Reconstruction (/articles/semantic-discovery/persistent-state)
- Traversal Lineage as Index Evolution Signal (/articles/semantic-discovery/traversal-lineage)
- Anchor Semantic Neighborhood Publication (/articles/semantic-discovery/semantic-neighborhoods)
- Inference-Time Execution Control as Traversal Primitive (/articles/semantic-discovery/inference-governance)

- Anchor Self-Organization Under Entropy and Load Pressure (/articles/semantic-discovery/anchor-self-organization)
- Alias Resolution as Navigational Traversal (/articles/semantic-discovery/alias-resolution)
- Three Discovery Operating Modes: Human Search, Agent Reasoning, Answer Synthesis (/articles/semantic-discovery/operating-modes)
- Model-Agnostic Semantic Discovery (/articles/semantic-discovery/model-agnostic)
- Affect-Modulated Discovery Traversal (/articles/semantic-discovery/affect-modulated-traversal)
- Confidence-Gated Discovery Traversal (/articles/semantic-discovery/confidence-gated-traversal)
- Integrity-Tracked Traversal Drift Detection (/articles/semantic-discovery/integrity-tracked-drift)
- Biological Identity-Scoped Access During Discovery (/articles/semantic-discovery/biological-access)
- Rights-Grade Anchor Governance for Content Discovery (/articles/semantic-discovery/rights-grade-anchors)
- Forecasting-Shaped Discovery Traversal (/articles/semantic-discovery/forecasting-shaped)
- Capability-Constrained Anchor Accessibility (/articles/semantic-discovery/capability-constrained)
- Collaborative Multi-Object Discovery Traversal (/articles/semantic-discovery/collaborative-traversal)
- Credentialed Reader Activation in the Discovery Substrate (/articles/semantic-discovery/credentialed-reader-activation)
- Personal Cognitive Asset: How Per-User Lineage Re-Weights the Same Substrate (/articles/semantic-discovery/personal-lineage-layer)

APPLICATIONS · GENERAL

- Enterprise Knowledge Management Through Governed Traversal (/articles/semantic-discovery/enterprise-knowledge-management)
- AI-Native Search That Replaces PageRank With Governed Contextual Relevance (/articles/semantic-discovery/ai-native-search)
- Semantic Discovery for Scientific Research (/articles/semantic-discovery/scientific-research-discovery)
- Verifiable AI Legal Case Research: Governed Case Law Search with Citation Lineage (/articles/semantic-discovery/legal-case-research)
- Patent Landscape Analysis Across Classification Boundaries: Governed Semantic Discovery for Prior Art and Freedom-to-Operate (/articles/semantic-discovery/patent-landscape-analysis)
- Governed Medical Literature Search: Evidence-Grade Traversal for Clinical Questions (/articles/semantic-discovery/medical-literature-search)

- Governed Competitive Intelligence Search Across Patents, Filings, and Hiring Signals (/articles/semantic-discovery/competitive-intelligence)
- Governed Semantic Search for Regulatory Compliance: Defensible Requirement Discovery Across Fragmented Corpora (/articles/semantic-discovery/regulatory-compliance-search)
- Cross-Vendor Credentialed Sensor Coordination for Multi-Sensor Perception (/articles/semantic-discovery/coordinated-perception)
- Anti-Stalking Architecture for Cross-Vendor Bluetooth Tracker Networks (/articles/semantic-discovery/post-airtag-tracking)
- How Small Language Models Beat Large Ones: Use the World as Memory (/articles/semantic-discovery/world-as-memory)

APPLICATIONS · SPECIFIC

- Google Search Alternative: Governed Semantic Discovery Beyond Retrieval (/articles/semantic-discovery/google-search)
- Perplexity Alternative: Governed Semantic Discovery Beyond the Answer Engine (/articles/semantic-discovery/perplexity)
- Elasticsearch Alternative: Governed Semantic Discovery Beyond Index-Centric Search (/articles/semantic-discovery/elasticsearch)
- Algolia Alternative: Governed Semantic Discovery Beyond Per-Query Search (/articles/semantic-discovery/algolia)
- Pinecone Alternative for Governed Semantic Discovery (/articles/semantic-discovery/pinecone)
- Weaviate Alternative: Governed Semantic Discovery Beyond the Vector Database (/articles/semantic-discovery/weaviate)
- You.com Alternative: Governed Semantic Discovery Beyond AI Search (/articles/semantic-discovery/you-com)
- Brave Search Alternative: Governed Semantic Discovery Beyond an Independent Index (/articles/semantic-discovery/brave-search)
- Kagi Alternative for Governed Semantic Discovery Beyond Query-Response Search (/articles/semantic-discovery/kagi)
- Exa (Metaphor Systems) Alternative: Governed Semantic Discovery Beyond Neural Link Prediction (/articles/semantic-discovery/metaphor-systems)
- Glean Alternative for Governed Enterprise Discovery: Beyond Stateless Workplace Search (/articles/semantic-discovery/glean)
- Coveo Alternative: Governed Semantic Discovery Beyond Personalized Ranking (/articles/semantic-discovery/coveo)
- Apple Find My vs Governed Semantic Discovery: Two Meanings of Multi-Authority (/articles/semantic-discovery/apple-find-my)

- Google Find My Device Alternative: Governed Cross-Network Reader Activation (/articles/semantic-discovery/google-find-my)
- Governed Discovery Beyond IETF DULT: Behavior Standards vs Architectural Governance (/articles/semantic-discovery/ietf-dult)
- Glean Alternative: Governed Semantic Traversal Beyond Enterprise Search (/articles/semantic-discovery/glean-enterprise-search)
- Microsoft GraphRAG vs Governed Traversal: Adding Per-Step Governance to Graph Retrieval (/articles/semantic-discovery/microsoft-graphrag)
- Governed Agent Memory Beyond Mem0, Zep, and Letta (/articles/semantic-discovery/memory-for-agents)

Semantic Discovery overview → (/semantic-discovery)