

KedgeFlow

A Programmable Control Plane for Consequential AI Agents

Separating cognitive generation from authoritative execution

Core principle

An agent's proposal describes a possible action; a model's judgment is evidence about it. Neither is, by itself, authority to mutate protected state.

Architecture and protocol specification

Version 0.4 | 1 October 2026 | Design proposal

Jun He and Deying Yu | OpenKedge.io

Executive summary

Modern artificial intelligence (AI) agents can propose high-consequence operations against cloud infrastructure, databases, identity and access management (IAM) configurations, and financial systems. Agent frameworks provide tools, hooks, approvals, and execution traces, but the coverage and authority semantics of these controls vary by runtime. A runtime that can execute a model-selected action with its own mutation credentials has not established an independent authority boundary. KedgeFlow gives heterogeneous runtimes a common way to submit proposed state changes to an external control plane while keeping token generation and reasoning inside the agent runtime.

This whitepaper introduces **KedgeFlow**, an open agentic control-plane framework and protocol with a logically centralized, versioned view of agent and resource state. It specifies an **Evidence-to-Authority boundary** to be enforced independently at protected mutation paths. Teams can begin with one protected operation: keep the existing agent, define the evidence and policy it needs, and place grant checking at a gateway that controls the mutation credential. The framework can incorporate future tools, assurance methods, and theories from any source when they satisfy its interface and enforcement contracts; OpenKedge components provide optional examples.

The Core Principle

An agent's proposal describes a possible action; a model's judgment is candidate evidence about it. Neither grants authority to mutate protected state. KedgeFlow prevents failures in generation or judgment from automatically becoming executable authority.

What a deployment needs

1. **Separate decisions from permissions.** Agents propose actions and providers evaluate them; an independent issuer grants authority only after the required evidence has been checked.
2. **Use one traceable execution path.** The six-stage Evidence-to-Authority lifecycle connects a canonical proposal, typed evidence, admission, a narrow grant, gateway redemption, and an authoritative outcome receipt.
3. **Put credentials behind enforcement.** Agents and evaluators lack mutation credentials for the declared protected surface. A gateway holds those credentials and mediates each in-scope mutation.
4. **Declare the state guarantee.** Adapter Levels A, B, and C distinguish atomic commit-time checks from pre-execution revalidation and post-hoc reconciliation. This makes Time-of-Check to Time-of-Use (TOCTOU) race exposure explicit.

Generation → Judgment → Assurance → Authority → Execution → Evidence

Where to start

Reader	Suggested path
Engineering leads	Sections 3, 6, 7, and 10 describe component roles, a first protected action, adapters, and the portable interface.
Security teams	Sections 8 and 12 define credential custody, coverage, adapter levels, and the limits of the safety claim.
Managers	Sections 6, 13, and 16 describe pilot scope, validation work, and evidence needed for deployment claims.
Protocol implementers	Sections 9 and 11 define messages, certificates, grants, redemption, and recovery.

What is established here

This document is a **design specification (version 0.4)**. It defines what a team would build and test; it does not report production deployment data or vendor performance claims. The strongest proposed safety guarantee depends on complete gateway mediation and atomic commit-time synchronization at the target resource. Existing reference monitors, policy decision points and policy enforcement points (PDP/PEP), and object capabilities can supply parts or all of the design. Any conventional policy infrastructure that fully realizes the KedgeFlow contract is functionally equivalent.

Contents

1	The missing boundary	1
2	Agent data plane and programmable control plane	3
3	Architecture at a glance	6
4	Extension points and reference methods	7
5	Where KedgeFlow applies	8
6	Start with One Consequential Action	9
7	Native runtimes, adapters, and enforcement	10
8	System model and trust boundary	10
9	KedgeFlow Protocol	13
10	KedgeFlow Protocol v0.1 interface requirements	16
11	From evidence to authority	17
12	Safety properties and failure containment	21
13	Pilot build and validation plan	24
14	Using existing authorization systems	27
15	Connecting current agent runtimes	27
16	Evidence needed for deployment claims	28
17	Control-plane separation in OpenFlow and KedgeFlow	29
18	Failure semantics and deployment boundaries	29
19	Implementation path	30
20	Implementation collaboration	30
21	Glossary and protocol status	31
	Document status	32

1 The missing boundary

Suppose an infrastructure agent receives: “Reduce database spend without impacting production availability.” It proposes removing a replica that appears idle. A semantic evaluator estimates that the action matches the request with probability 0.97. The probability estimate provides a helpful signal: it indicates how one model evaluated intent alignment within its prompt context. It does not, however, establish that the replica is currently idle, that failover capacity is sufficient, that the requesting principal holds requisite permissions on the target database, or that concurrent operations have not rendered the mutation unsafe.

The failure occurs when a judgment about an action becomes permission to execute it. Agent systems may use direct tool calls, middleware, approvals, or policy engines; the protection each provides depends on its actual credential and execution paths. Even an accurate model can work from stale or unverified context. KedgeFlow makes the step from proposed action to protected effect explicit and enforceable across runtimes.

Definition 1 (Evidence-to-Authority boundary). The Evidence-to-Authority boundary governs when and how evidence supporting an action is converted into narrowly scoped, verifiable authority permitting that *exact* action to mutate protected state. It cryptographically binds a canonical proposal, action-specific assurance obligations, the verified witnesses that discharged them, an active policy version, and commit-time state conditions to an independently redeemable execution grant.

Generation → **Judgment** → **Assurance** → **Authority**
→ **Execution** → **Evidence**

Each stage has distinct security semantics and operational boundaries. **Generation** proposes an intent, plan, tool call, delegated subtask, or external mutation. **Judgment** evaluates proposal semantics, scores intent alignment, or tests specific claims. **Assurance** determines whether the proposed action satisfies all policy-defined obligations using valid, fresh, and sufficiently independent witnesses. **Authority** mints an integrity-protected execution grant upon successful admission. **Execution** validates and redeems the grant at the protected resource boundary, mediating physical mutation. **Evidence** records the authoritative execution trace, gateway redemption receipt, and observed post-state outcome.

This architectural separation is meaningful only if it alters system behavior under adversarial or accidental failure:

- A compromised generator can construct malicious or malformed proposals, but cannot execute them directly.
- A faulty or compromised evaluator can return erroneous evidence, but cannot mint execution grants.
- An intercepted bearer grant remains strictly bounded to its canonical action, target resource, parameter predicate, validity interval, and single-use nonce.
- A stale grant fails redemption when commit-relevant state has changed, provided the underlying resource supports atomic precondition checking.

For an implementation, the first test is whether a generated tool call can reach the protected resource without the required admission and grant. These failure-containment properties define the formal and empirical claims KedgeFlow must substantiate.

1.1 Why this boundary matters

The following paths show where execution authority is checked:

Pattern	Path to a protected effect
Direct agent execution	Agent → Tool → long-lived credential → Effect
Judge-gated execution	Agent → model judge → Tool → Effect
KedgeFlow	Proposal → Evidence → Assurance → Grant → Gateway → Effect → trusted Outcome Evidence

While a large language model (LLM) judge gate can filter ill-formed proposals, directly invoking privileged tools upon a positive score leaves the model’s judgment in the de facto authorization role. KedgeFlow requires that evidence first satisfy policy-defined assurance obligations to achieve admission, after which an independent issuer mints a cryptographically bound grant for gateway redemption. The governing question is not whether an evaluator generally approves the plan, but whether verified evidence discharges all required obligations for that *exact* mutation under the active policy and current state.

Operational rule

A plan or positive model score remains non-authoritative evidence. Only an admitted action receives a bounded grant, and the gateway checks that grant before a protected mutation.

Iterative reasoning, reflection, or multi-turn self-critique may improve proposal quality, but none establish an independent control boundary when the generator and evaluator share the same execution context, prompt lineage, or underlying weights.

1.2 What KedgeFlow adds beyond existing controls

Existing controls remain valuable and can implement parts of KedgeFlow. The table identifies the question each addresses and what a consequential stochastic action still requires. Role-based access control (RBAC), attribute-based access control (ABAC), and policy decision and enforcement points (PDP/PEP) are established approaches used here.

Approach	What it establishes	Question still to resolve
Long-lived agent credentials	Workload identity and a reusable permission envelope	Does this specific proposed mutation have current, sufficient evidence and acceptable consequences?
RBAC / ABAC [1]	Static role- or attribute-based constraints over subject, resource, operation, and context	Which fresh semantic, deterministic, and operational witnesses satisfy this action’s assurance obligations right now?
Conventional PDP / PEP [2, 3]	Architectural decoupling of policy decision from policy enforcement	A complete KedgeFlow contract can be implemented within this paradigm; absent KedgeFlow semantics, evidence freshness, correlation limits, cryptographic grant binding, and trusted outcome linkage remain ad hoc engineering choices.
LLM judge gate	Additional semantic assessment before execution	Does the model’s output remain non-authoritative evidence, or does a positive score directly trigger privileged tool invocation?

Approach	What it establishes	Question still to resolve
Sandbox	Confinement of the agent runtime environment and observable side effects	Which external state mutations are authorized at the protected resource boundary?
Human approval	Human-in-the-loop review for high-consequence operations	Is the human approval cryptographically bound to the exact canonical action, parameters, verified state preconditions, and resulting execution?
KedgeFlow	End-to-end Evidence-to-Authority protocol: canonical proposal, obligation discharge, action-bound grant, gateway redemption, and outcome linkage	Assumes complete mediation, sound policy, and resource-side enforcement; it does not replace underlying authentication, isolation, or atomic resource guarantees.

For each protected operation, the team must decide *what evidence is sufficient to authorize this exact mutation now?* The answer depends on intent, the proposed action, delegated authority, current state, dependencies, and the consequence of error. KedgeFlow specifies how that answer becomes narrow, enforceable authority. It builds on existing authorization, capability, reference-monitor, control-plane, and zero-trust concepts.

1.3 Why a control-plane analogy helps

Software-defined networking separated control decisions from packet forwarding; OpenFlow provides one concrete protocol for programming switches [4]. KedgeFlow applies the same separation to typed, state-changing operations. Its admission decision accounts for intent, evidence, dependencies, and resource state; an action-bound grant is then checked at the execution boundary. Section 17 develops this comparison. The analogy itself establishes neither novelty nor safety.

Design objective

Prevent a proposal or semantic judgment from directly carrying the authority needed to mutate a protected resource.

2 Agent data plane and programmable control plane

The *Agent Data Plane* is the runtime machinery for model inference, token generation, reasoning and planning, memory access, tool selection, sub-agent invocation, and execution of operations already authorized locally. KedgeFlow does not inspect or program individual tokens or require private chain-of-thought. Its primary unit of control is a proposed consequential state transition, represented by q in Eq. 1. The boundary is reached when an agent would turn a generated action into a protected effect.

An *KedgeFlow-compatible agent runtime* exposes enough authenticated, standardized observation and programming surface for an external controller to discover effective capabilities, receive proposals, install bounded constraints or authority, and receive outcomes. Compatibility describes an interface; an authorization-safety claim additionally requires non-bypassable enforcement for the declared resource/action surface. Native runtimes may expose this interface directly. Existing runtimes may use adapters, subject to the same coverage test.

Separation of responsibilities

Agents propose transitions. The control plane evaluates admission and confers bounded authority. The execution plane enforces that authority at the protected mutation boundary.

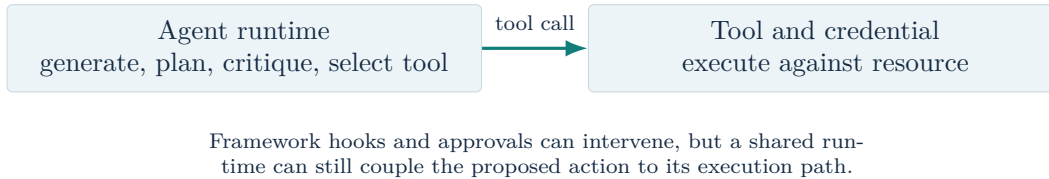


Figure 1: A common coupled pattern, not a claim that all agent frameworks have the same security boundary. Self-critique inside the generating runtime does not itself create an independent authority boundary.

2.1 Controller view and interfaces

KedgeFlow is *logically centralized*: an admission decision can use authenticated, versioned observations from several agents and resources. The controller need not be one physical server. Replicated, hierarchical, federated, or institutionally sovereign deployments can preserve one accountable policy and authority decision for each governed resource scope, while making conflicts and stale views explicit.

The controller tracks identities and capabilities, active proposals, resource topology and dependencies, policy and revocation versions, evidence references, issued grants, observed resource state, and outcomes. Each observation has freshness and provenance limits. The resulting view helps coordinate decisions but is not a complete ground truth.

The northbound application programming interface (API) accepts institutional intent: policies, constraints, approvals, risk budgets, and service objectives. The *southbound interface* carries capability discovery, relevant observations, normalized proposals, authority installation and revocation, guarded execution requests, and outcome reports between the controller, runtimes/adapters, and enforcement points. In the initial design these are semantic operations; no network transport or policy language is mandated.

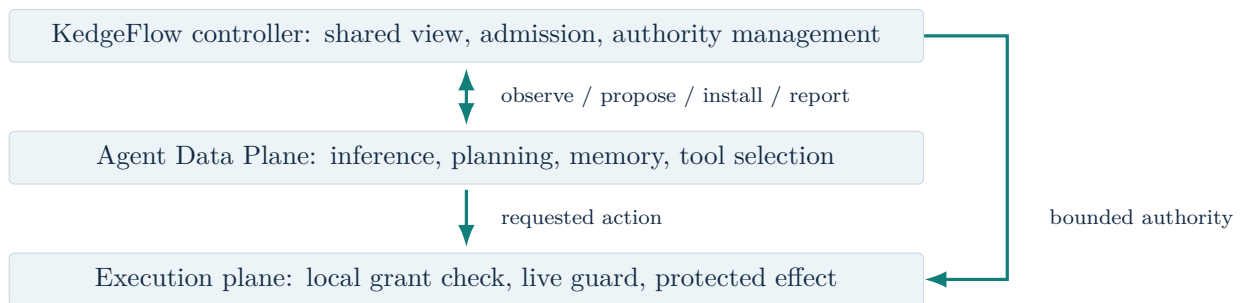


Figure 2: Control, cognition, and enforcement are distinct logical roles. A grant may be checked locally; protected execution need not wait for a remote controller round trip.

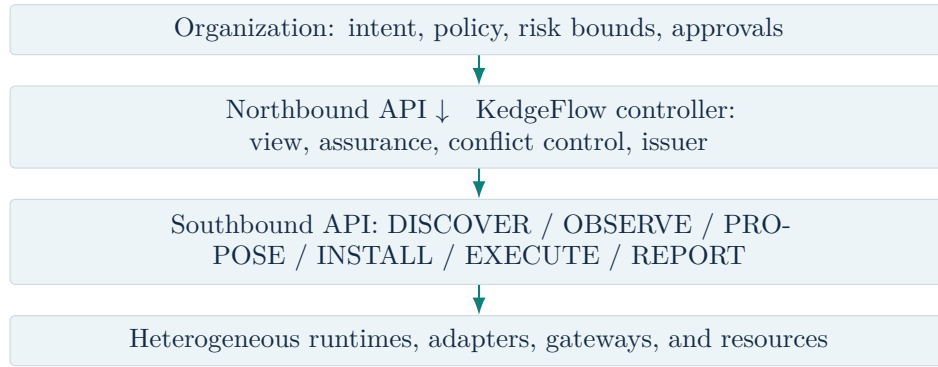


Figure 3: The northbound interface expresses organizational control; the southbound interface programs and observes execution domains. The arrows show responsibility, not a mandatory transport topology.

2.2 Why a cross-agent view changes admission

Suppose an operations agent proposes terminating instance X , a deployment agent has a pending rollout onto X , and a third service currently depends on X . If observed production capacity equals the policy minimum, a controller with sufficiently fresh, authenticated dependency and capacity state can deny or defer termination, or require remediation. No individual agent need have the complete view. An omitted dependency or stale capacity observation can still produce a bad decision; KedgeFlow exposes these as model and freshness obligations rather than claiming semantic correctness.

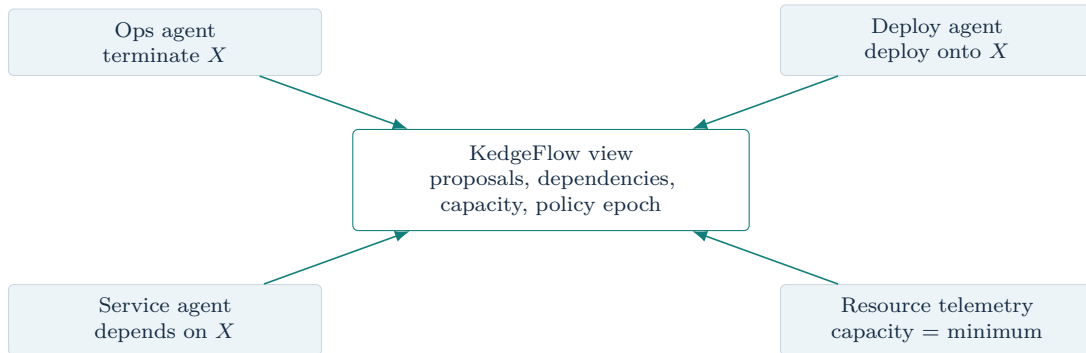


Figure 4: An admission decision may need cross-agent state. Each input remains provenance-tagged and time-bounded; missing or conflicting observations yield an explicit unknown or defer decision.

2.3 Identity and access management

Identity and access management (IAM) establishes who may hold a permission and reach a resource. KedgeFlow additionally evaluates whether *this* proposed transition should be admitted under current evidence, dependencies, policy, and state, then binds any issued authority to the exact action. Cognitive Admission Control (CAC) supplies consequence-conditioned assurance obligations [5]. IAM remains a prerequisite for authenticating principals, issuers, gateways, and target systems; KedgeFlow does not replace it. A conventional policy decision/enforcement system implementing the complete KedgeFlow contract is functionally equivalent.

3 Architecture at a glance

Figure 5 illustrates the logical control flow. These stages may run in one process or across services, but their identities, responsibilities, and credential permissions remain distinct. Policies can be replicated to local decision nodes; KedgeFlow does not require a remote controller round trip for every execution.

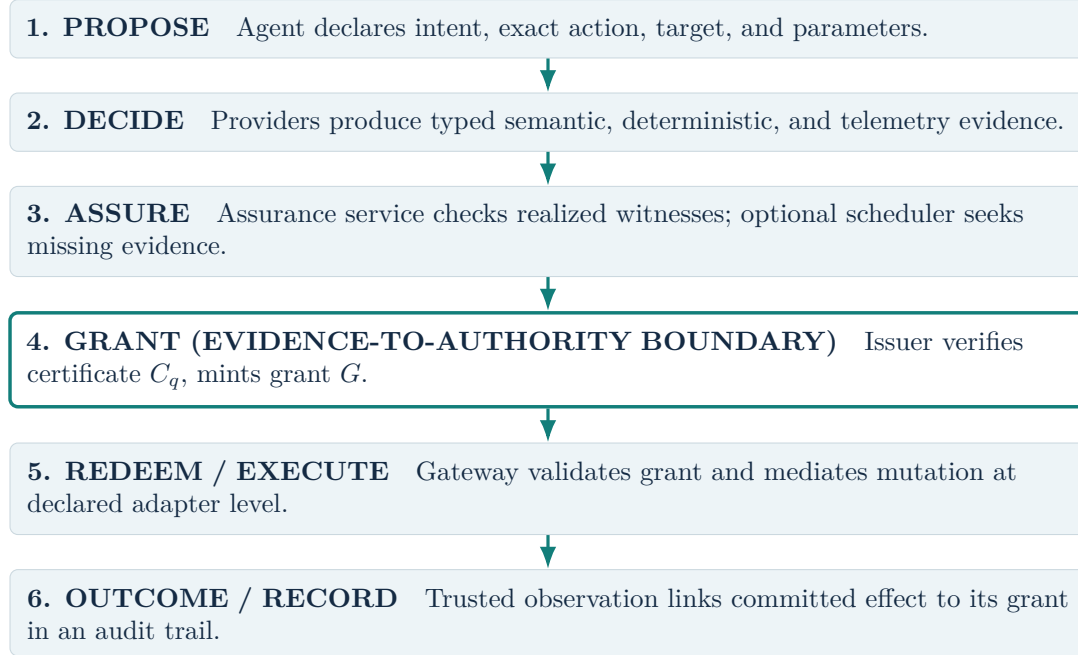


Figure 5: The six-stage logical path: providers supply non-authoritative evidence, assurance issues an admission certificate, and the grant issuer mints narrow capability authority. The execution gateway mediates each in-scope mutation with exclusive custody of backing credentials.

3.1 Who owns which decision?

Component	Can produce	Cannot do by itself
Agent / generator	Canonical proposal, plan, parameter payload	Mint execution authority or directly access protected resources
Decision provider	Structured, typed evaluation evidence	Grant execution authority or invoke protected tools
Assurance service	Obligation resolution, witness verification, admission certificate (C_q)	Mint execution grants (G) or execute mutations
Grant issuer	Signed, action-bound execution grant (G)	Directly execute mutations or attest to execution outcomes
Execution gateway	Grant redemption receipt, mediated mutation call	Accept unadmitted proposals or direct agent prompts as authority
Outcome service	Cryptographically linked audit trail, observed outcome receipt	Issue execution grants or alter historical execution records

4 Extension points and reference methods

KedgeFlow fixes the path from proposal to evidence, admission, grant, guarded effect, and outcome receipt. It leaves room to choose or replace the methods that fill those roles. A team may use existing policy engines, verifiers, models, telemetry, and gateways, then add future tools or assurance theories as long as they meet the same interface and enforcement contracts. The OpenKedge works below are reference methods. Cognitive Admission Control (CAC) models required checks, and Assurance-Aware Semantic Scheduling (AAS) models how to acquire evidence; neither is required to implement the protocol.

4.1 Identity and delegation

An implementation needs to know which principal requested an action and how authority was delegated. Persistent Cognitive Identity (PCI) models continuity, agent provenance, and delegation chains across substrate boundaries [6]. KedgeFlow incorporates applicable delegation lineage while requiring authenticated workload identities at proposal submission, grant issuance, and gateway redemption. Provenance context cannot substitute for cryptographic workload authentication.

4.2 Required checks before execution

For each protected action, policy must say what evidence is required under the current state. CAC is one model for defining these consequence-conditioned assurance obligations [5]. The assurance service checks the witnesses actually obtained and issues an admission certificate C_q only after every obligation is satisfied. A plan to obtain evidence does not count as evidence.

4.3 Getting the needed evidence

An optional scheduler can decide which checks to run first, when to refresh telemetry, and when an ambiguous result needs another provider or human review. AAS models this choice under latency, cost, and epistemic-risk constraints [7]. Scheduling can reduce work, but the assurance service still verifies the resulting receipts at admission. A future scheduler can use a different theory while producing evidence that meets the same obligations.

4.4 Checking witness independence

Two agreeing evaluators may still share the same source of error: training data, retrieval corpus, telemetry feed, prompt template, or operator. The Honest Quorum model provides exposure maps and witness-diversity constraints for these epistemic fault domains [8]. A deployment records the dependencies it knows and applies its policy to them; it cannot assume that agreement proves independence when shared dependencies remain unknown.

4.5 Recording what happened

The outcome record must come from the execution gateway or target resource. Agent-native telemetry is one method for signed state-delta receipts and verifiable post-execution observations [9]. An agent's own assertion that an action succeeded remains non-authoritative.

4.6 Commit checks and credential custody

At execution, the gateway validates the grant and applies the state controls available at its declared adapter level. Taming Cognitive Transactions (TCT) studies when captured data, evidence, policy, and authority remain valid at commit under stated guard assumptions [10]; it can inform Level A adapters for transactional resources. The Sovereign Execution Broker (SEB) is one design for holding mutation credentials and enforcing certificate-bound actions, scoped identity, drift checks, replay control, and signed outcomes [11]. Other implementations can fill these roles under the same KedgeFlow contract.

4.7 How the pieces connect

The canonical proposal, admission certificate (C_q), active policy version, commit-time state predicate, and authenticated principal are bound into an action-specific execution grant (G). The gateway redeems G at the protected mutation boundary, checks the required conditions, and links the observed outcome to the authorization record. This binding is the portable contract; the methods used to produce evidence or implement the gateway may evolve.

5 Where KedgeFlow applies

The following are illustrative applications of the architecture, not claims of deployed OpenKedge integrations. They show how the same boundary can support different evidence and authority profiles.

5.1 Infrastructure automation

An infrastructure agent proposes reducing the replica count for a production database cluster. Relevant evidence includes the principal's delegation lineage, target service-level objectives (SLOs), real-time traffic volume, failover headroom, failure-domain topology, concurrent deployment locks, and the current resource version. CAC policy specifies mandatory capacity and availability invariants; an AAS scheduler can refresh telemetry immediately prior to admission. The resulting grant authorizes exclusively the admitted replica-count mutation on the specified resource identifier. A Level A gateway adapter rejects execution if the commit-time resource version differs from the admitted state; Level B or C adapters must explicitly declare their residual race window.

For a concrete cross-agent trace, consider an OpsAgent proposing `TerminateInstance(i-17)` while a DeploymentAgent has a pending rollout onto that instance and observed spare capacity equals the policy minimum. **DISCOVER** identifies both agents' effective tools and the resource owner; **OBSERVE** supplies the deployment dependency, capacity, and policy epoch with provenance and timestamps. **PROPOSE** freezes the exact termination request as q . CAC resolves availability and dependency obligations, and an optional AAS scheduler requests fresh telemetry or a qualified review. The assurance service cannot issue C_q while a required witness is missing or contradictory, so the controller denies or defers. If the dependency is removed and fresh evidence discharges every obligation, the issuer creates G ; **INSTALL** places it at the instance gateway; **EXECUTE** redeems it only if the live capacity and resource-version guard still holds; and **REPORT** records the attempt and observed effect. The example illustrates the need for shared state, not a claim that the controller has perfect information.

5.2 Security automation

A security agent proposes rotating a production service credential following an anomaly detection event. In this domain, evaluation delay exacerbates exposure risk, whereas erroneous revocation risks immediate service disruption. The assurance profile adapts accordingly: verifying credential identity and affected dependencies, assessing consumer readiness and blast radius, and activating emergency escalation pathways when policy permits. Authority is restricted to rotating the designated credential within a narrow validity window and for a single redemption. The authoritative outcome record must disambiguate successful rotation, failed rotation, and partial dependency disruption.

5.3 Financial or business workflow

A customer-support agent proposes issuing a refund or account credit. Required evidence spans transaction settlement status, customer and agent authorization tiers, real-time fraud risk scores, disbursement caps, dual-control (four-eyes) invariants, and mandatory human approval thresholds. The minted grant binds the precise account identifier, original transaction reference, credit amount, currency, and client-generated idempotency key. The payment gateway must report verified settlement effects, explicitly persisting an *unknown pending reconciliation* state whenever external clearinghouses have not confirmed finality.

These examples share a common protocol, not a uniform policy. The required witnesses, delegation rules, temporal validity, and compensation or rollback expectations scale with consequence, uncertainty, and reversibility.

6 Start with One Consequential Action

KedgeFlow sits between an existing agent and a protected operation. A team can start with one mutation and route it through a KedgeFlow-compatible boundary:

Existing agent framework → KedgeFlow boundary
→ existing API, cloud, Kubernetes, database, or enterprise service

Candidate first actions include production scaling, deployment rollback, IAM policy edits, credential rotation, resource teardown, and database mutation. Keep the agent's reasoning workflow, but route its tool call as a canonical proposal to a gateway that holds the mutation credentials and enforces the grant. Revoke alternate direct credentials to the target resource or exclude those paths from the safety claim.

A first implementation can follow four checks:

1. **Choose one operation.** Record its action, resource, and every mutation route.
2. **Set admission policy.** Define its proposal schema and required evidence.
3. **Enforce at the gateway.** Isolate credentials; issue and redeem a grant for the exact action.
4. **Test the claim.** Declare its adapter level (Section 12) and run bypass, replay, and race tests (Section 13).

A Level B or C adapter can be a starting point only if its narrower state guarantee is explicit. This is an incremental adoption path, not a claim of an existing KedgeFlow deployment.

7 Native runtimes, adapters, and enforcement

A KedgeFlow Agent Adapter translates framework-specific hooks, middleware, graph transitions, tool calls, and trace events into the southbound operations in Section 10. It can report effective capabilities and relevant state, normalize a pending consequential tool call into q , pause dispatch until admission, and return a grant reference or denial to its host runtime. The adapter must report where it cannot observe or intercept an effect. A native KedgeFlow runtime exposes the same semantics directly; native support alone does not prove enforcement coverage.

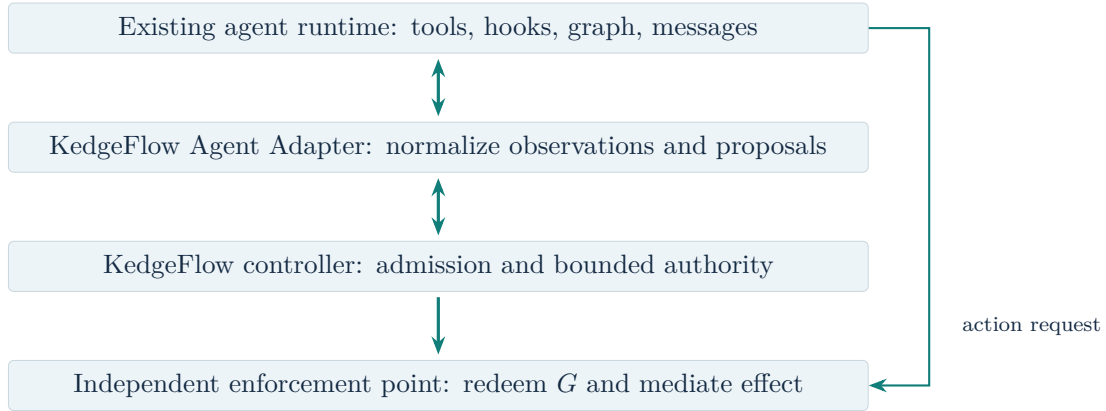


Figure 6: An adapter changes the agent interface without requiring a new reasoning framework. A separate enforcement point validates authority; an adapter cannot secure a resource the agent can still mutate through another credential path.

The agent interface provides *visibility and programmability*. The enforcement point supplies the authoritative check. It may be an execution or credential broker, gateway, service proxy, tool runtime, cloud authorization layer, database proxy, or resource-native policy. A hook that merely denies a tool call has a framework-local guarantee; if the same principal can call a cloud API directly, complete mediation fails. Stronger deployments remove ambient mutation credentials from the agent and place the check at a boundary it cannot bypass. Section 12 distinguishes the state guarantees of Levels A, B, and C.

The migration test is concrete: an OpenAI, Claude, Google Agent Development Kit (ADK), Agent Framework, LangGraph, or custom agent should be able to submit the same typed proposal through a thin adapter without rewriting its cognition. Each integration must publish its observed and intercepted action surface, effective credential paths, and resource-side coverage inventory. An observer-only adapter can improve visibility and reporting, but cannot claim authorization safety for unmediated mutations.

8 System model and trust boundary

A deployment begins by naming the actions and resources it will protect. It maintains authoritative registries of requesting principals $\mathcal{P}$, canonical typed actions $\mathcal{A}$, protected resources $\mathcal{R}$, and versioned policies Π_v . Each protected action has one strict schema, so aliases, whitespace, or field order cannot change how the assurance service or gateway interprets a request. The resource registry maps each canonical target to its designated gateway and backing application programming interface (API).

8.1 A proposal is an object, not a credential

A proposal is the immutable request record an agent submits for admission. It carries no permission to execute:

$$q = \langle id, p, i, a, r, \theta, d, x, h_{s_0} \rangle. \quad (1)$$

Here id is a globally unique proposal identifier, $p \in \mathcal{P}$ is the authenticated principal, i declares high-level intent, $a \in \mathcal{A}$ is the canonical typed action, $r \subseteq \mathcal{R}$ designates the target resource set, θ denotes the canonical parameter payload, d specifies causal dependency references, x provides operational execution context, and h_{s_0} is a cryptographic digest or version identifier of baseline observed state s_0 . A proposal can be admitted, denied, remediated, or reissued, but conveys zero authority to execute against protected state.

The assurance service evaluates proposal q against active policy Π_v and acquired evidence E . $\text{Admitted}_{\Pi_v}(q, E, s_a)$ records its verdict under admission-time state s_a . $\text{Committed}(q, s, s')$ records a different event: the resource changed from pre-state s to post-state s' . A provider first produces a candidate evidence item; it becomes a *witness* only when verified and accepted for a specific assurance obligation. Admission, grant issuance, execution, and outcome confirmation are separate events. A proposal may be admitted but never executed, and an attempted execution may fail or remain uncertain until reconciliation. The control plane records those distinctions in separate audit receipts.

8.2 What can go wrong?

KedgeFlow considers four propagation layers:

1. **Compromised intelligence:** prompt injection, malicious retrieved context, poisoned memory, a compromised generator, or an over-broad plan.
2. **Incorrect judgment:** uncaught hallucinated parameters, stale telemetry, erroneous semantic evaluation, model scoring errors, or correlated misjudgments among structurally dependent providers.
3. **Invalid authority:** policy drift, forged or stolen grants, confused-deputy exploitation, replay attacks, privilege expansion during delegation, or grant issuance without satisfied obligations.
4. **Enforcement bypass:** direct credential leakage to agents, unauthorized tool paths, unmediated gateway bypass, missing precondition checks in adapters, or race conditions between state validation and non-atomic execution.

The architecture is designed to prevent failures in generation or judgment from automatically translating into execution authority over protected state. It cannot, however, prevent compromise of the issuer's private signing keys or compensate for an unmediated execution path. Nor can formal control transform an unsound policy, missing dependency model, or falsified ground-truth evidence into safe execution.

8.3 Trusted base and coverage

The safety guarantees depend on authenticated principals and providers; protected policy distribution and signing keys; deterministic request canonicalization and resource mapping; gateway custody of mutation credentials; durable replay state; and a gateway that mediates *every* in-scope mutation.

For each canonical resource/action pair (r, a) , a versioned *coverage inventory* records the resource, covered action, gateway, backing API, credential owner, alternate credential routes, unmanaged mutation paths,

enforcement mechanism, and adapter guarantee level. Teams must verify this inventory through resource-side access control audits and direct penetration tests. Administrative accounts, continuous integration and delivery (CI/CD) pipelines, and auxiliary agents that can make the same mutation must pass through the gateway or be excluded from the safety claim. **KedgeFlow safety claims apply exclusively to the mutation surface for which complete mediation is empirically verified.** Read-only observation can use other paths, but evidence acquisition with side effects is itself a protected action requiring assurance and mediation.

To guarantee state-valid execution at commit time, the gateway requires an atomic resource-side synchronization primitive: compare-and-swap (CAS), conditional PUT, transactional commit, or cryptographic fencing tokens. A pre-state digest verified during admission provides necessary freshness evidence, but cannot prevent Time-of-Check to Time-of-Use (TOCTOU) races against backing APIs lacking atomic precondition enforcement. For non-atomic APIs, KedgeFlow provides pre-execution revalidation and post-hoc reconciliation, but the adapter must explicitly declare a degraded guarantee level.

8.4 Credential custody and compromise

The requesting agent, decision providers, and assurance service hold no credentials capable of mutating protected resources. The grant issuer possesses signing keys to issue execution grants, but has no access to backing resource credentials. The execution gateway alone holds and exercises credentials for backing APIs; target resources authenticate the gateway or an equivalent reference monitor. Any configuration granting direct mutation credentials to an agent or evaluator violates complete mediation and immediately invalidates authorization guarantees for that surface.

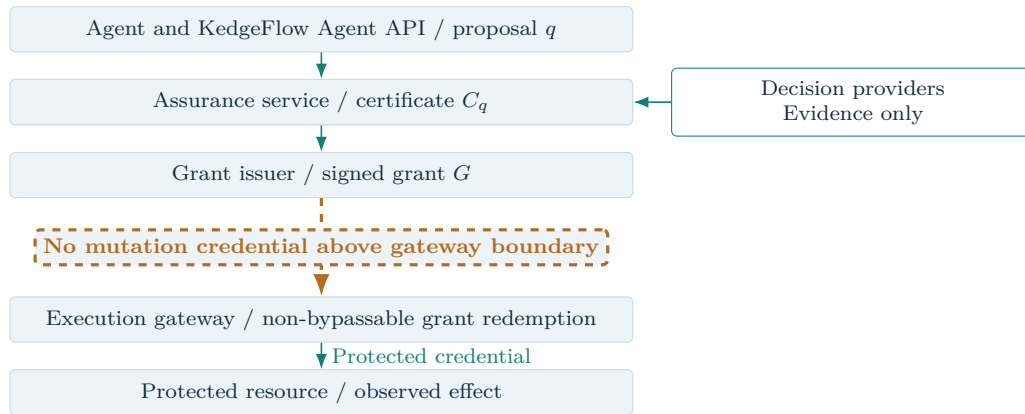


Figure 7: Agent interface versus enforcement boundary. The interface exposes proposals but holds no mutation credential. Provider arrows carry evidence, C_q attests admission, and G carries narrow authority. Only the gateway can exercise the backing mutation credential.

Component compromise has role-specific consequences: a compromised agent can submit adversarial proposals; a compromised provider can generate deceptive evidence; a compromised assurance service can falsely attest admission; a compromised issuer can forge execution grants; and a compromised gateway can directly execute unauthorized mutations. Consequently, the assurance service, grant issuer, execution gateway, policy distribution pipeline, replay storage, and resource-side access controls constitute the Trusted Computing Base (TCB) for their respective claims. The Evidence-to-Authority boundary prevents an agent or provider from exercising execution authority on its own, subject to complete mediation and the integrity of those trusted components.

9 KedgeFlow Protocol

The nine phases below describe what a deployment must record and check as an agent action moves toward a protected effect. They are logical transitions, so an implementation can map them to local calls or network services. Every control-plane message carries a protocol version, unique message identifier, authenticated sender identity, timestamp, canonical payload digest, and cryptographic integrity envelope; proposal identifiers and parent lineage references are attached where applicable.

REGISTER → POLICY_SYNC → PROPOSE → DECIDE
→ ASSURE → GRANT → REDEEM
→ OUTCOME → RECORD

Each completed transition emits an immutable, cryptographically verifiable receipt. Proposal and evaluation messages carry data, assurance checks that data against policy, the issuer creates a narrow grant, and the gateway alone redeems it against backing infrastructure. Changing a proposal's action, target, parameters, or bound state invalidates its pending verification and requires admission again. An issued grant cannot be expanded or modified.

The resulting authorization record follows this chain:

$$\begin{aligned} \langle q, E, \Pi_v \rangle &\longrightarrow C_q \longrightarrow G \longrightarrow \text{Redeem}(G) \\ &\longrightarrow \text{Effect} \longrightarrow \text{Outcome} . \end{aligned} \tag{2}$$

In practical terms, the assurance service checks that valid, fresh witnesses satisfy the policy obligations before issuing C_q . The issuer verifies that certificate before minting G . The gateway checks G , applies its declared adapter-level state controls, and mediates the protected effect.

9.1 Message contract

Message	Required content	Receiver check and effect
REGISTER	Principal, provider, gateway, or resource identifier (ID); canonical action schema; verification key; protocol version	Authenticate registrar; install versioned identity record and verified schema.
POLICY_SYNC	Signed Π_v , activation timestamp, supersession rule, revocation epoch, compatible schema and gateway versions	Check signature, monotonic version and epoch, and activation; acknowledge installation or fail closed.
PROPOSE	Canonical q from Eq. 1; authenticated principal signature; dependencies and baseline state references	Resolve exact action and target; freeze proposal digest $H(q)$. Yields zero execution authority.
DECIDE	Provider ID and version, question schema Q_D , state view digest, typed result E_D^* , timestamp, provenance, exposure domains	Validate schema, signatures, and provenance; append candidate evidence item. Yields zero authority.
ASSURE	Proposal digest $H(q)$, active Π_v , obligation set Ω , realized witness manifest E^* , state binding b_s , discharge results	Assurance service evaluates discharge predicates; issues C_q if and only if all obligations are SATISFIED.

Message	Required content	Receiver check and effect
GRANT	Admission certificate C_q , requested scope, caller identity, target gateway audience aud	Issuer validates certificate integrity, service identity, proposal binding, policy currency, freshness, and scope; mints signed G .
REDEEM	Execution grant G , exact canonical request payload, caller identity, gateway and resource IDs	Gateway validates issuer signature, audience, caller, scope, policy compatibility, revocation epoch, nonce, and state predicate φ_s .
OUTCOME	Attempt ID, observed state delta, error code, or explicit <i>unknown</i> status; authoritative resource version if observed; observation timestamp	Distinguish committed, aborted, and unreconciled effects.
RECORD	Cryptographically chained digests of proposal, evidence receipts, admission certificate, grant, redemption, and outcome	Store a queryable audit record with explicit gaps and retention metadata.

9.2 Exceptional transitions and operational recovery

Ambiguous evidence, missing dependencies, and transient contention require explicit protocol responses. KedgeFlow defines six exceptional transitions while preserving the authority boundary:

- **DENY** conveys unsatisfied obligation identifiers, structured reason codes, active policy version, and the evaluation trace, giving the agent actionable diagnostics.
- **REMEDiate** specifies unmet obligations alongside a constrained remedy schema and parent proposal reference. Any consequential remediation action (e.g., provisioning extra capacity before resizing a cluster) must undergo its own independent admission lifecycle; remediation never conveys provisional or bypass authority.
- **ESCALATE** re-routes unresolved or ambiguous obligations to a higher-assurance provider class, multi-provider committee, or authorized human reviewer without mutating the original proposal schema.
- **REFRESH** re-evaluates aging evidence against an updated state digest while preserving historical receipts, avoiding complete reasoning re-runs when only state freshness expired.
- **DELEGATE** requests a child grant from the issuer, enforcing strict monotonic scope attenuation relative to the parent grant and preserving delegation lineage.
- **REVOKE** invalidates a grant identifier, assurance signing key, or issuer key, specifying an activation timestamp, reason code, and monotonically increasing revocation epoch; gateways enforce revocation synchronously or within a declared bounded-staleness window Δ_{rev} .

System outages cannot be treated as affirmative verdicts. If a decision provider is unavailable, affected obligations evaluate to UNKNOWN. Under default-deny semantics, admission is withheld unless an alternate qualified provider supplies valid evidence, an authorized human reviewer attests to the condition, or the active policy explicitly defines a safe fallback rule. If the assurance service or grant issuer is unreachable, no new execution grants can be minted. A gateway may redeem an existing valid grant only if its synchronized policy and revocation state are within their declared freshness bounds Δ_{sync} ; otherwise, it must fail closed. If a mutation commits but the outcome observation is delayed or lost, the system persists an *unknown pending reconciliation* state, strictly rejecting client- or agent-reported assertions of success. Following a crash, a gateway must recover durable nonce state before accepting retries;

operations with uncertain external effects require idempotent retry tokens or manual reconciliation before re-issuance. KedgeFlow does not claim exactly-once semantics without backing resource support.

9.3 Policy activation, supersession, and revocation

Each signed policy version specifies its activation time, issuance eligibility, compatibility rule for outstanding grants, revocation epoch, and maximum synchronization age. The issuer and gateway durably retain a monotonic high-water mark and reject rollback to an older signed policy or epoch. The issuer checks that the certificate’s policy is issuance-eligible at issuance time; otherwise it requires re-admission. At redemption the gateway requires

$$\text{Compat}(\Pi_{v_{\text{issued}}}, \Pi_{v_{\text{current}}}, G) = \text{true}, \quad (3)$$

in addition to grant validity. Here “current” is the latest policy the gateway has authenticated within its permitted synchronization age; an update unknown during the propagation window cannot be enforced instantaneously. An unspecified compatibility rule fails closed. A versioned rule may retain an older grant until expiry, invalidate it after activation and the declared propagation bound, or permit it only for compatible actions; the choice is explicit policy, not an implicit assumption.

For example, after admission and grant issuance under Π_{18} , activation of Π_{19} does not by itself decide the grant’s fate. Its signed supersession rule determines whether $\text{Compat}(\Pi_{18}, \Pi_{19}, G)$ holds. **POLICY_SYNC** distributes the new policy and epoch; **REVOKE** names grants or keys invalidated by an emergency update. Emergency invalidation has a declared propagation bound Δ_{rev} , not instantaneous global effect. A gateway that cannot confirm a sufficiently fresh epoch by its deadline stops redeeming affected grants. An already committed effect cannot be revoked. Key rotation and policy supersession use the same monotonic, signed distribution discipline.

9.4 Minimum KedgeFlow conformance

This is an architecture-level checklist; wire interoperability still requires pinned schemas and cryptographic profiles. To claim KedgeFlow compatibility for a declared mutation surface, an implementation **MUST** provide:

1. Canonical typed proposals and authenticated principal/resource identity.
2. Separation of decision evidence from executable authority, with versioned assurance policy.
3. Admission over realized witnesses and an integrity-protected certificate bound to proposal, policy, obligations, evidence, and state.
4. Independent issuer validation of that certificate and an action-bound, integrity-protected grant.
5. Gateway enforcement of exact action, resource, parameters, audience, caller binding where required, policy compatibility, and revocation.
6. Durable replay/redemption control, declared state-validation level, and trusted outcome recording with explicit unknown states.
7. A coverage inventory and complete mediation for every resource/action pair included in its safety claim.

These grouped requirements are mandatory within the claimed surface. AAS scheduling, PCI, Jev, multiple semantic models, human review, agent-native telemetry, formal verification, and epistemic-diversity rules are optional mechanisms or policy choices. An independent system can implement the mandatory contracts without using OpenKedge components.

10 KedgeFlow Protocol v0.1 interface requirements

These six southbound operations define the behavior a native runtime or adapter must expose. Teams can map them to local calls or their chosen transport; this section does not prescribe a wire format. The operations map to the lifecycle messages in Section 9. In particular, **INSTALL** distributes the already issued grant G and signed policy constraints; it creates no second authority object. The controller can use CAC, AAS, deterministic policy engines, formal solvers, model judges, human review, or future providers. Their outputs remain evidence, not authority.

Operation	Minimum semantics	Existing lifecycle correspondence
DISCOVER	Authenticated, versioned effective agent/tool capabilities, reachable resources, execution environment, supported guards, and evidence types. Unknown coverage is explicit.	REGISTER establishes identities and schemas; discovery adds a current capability/topology view.
OBSERVE	Provenance-tagged task/intent, pending operations, dependencies, relevant resource state, authority, policy epoch, and health, each with observation time and freshness bound.	Supplies state and evidence references to DECIDE and ASSURE ; observation alone grants no authority.
PROPOSE	Canonical q (Eq. 1) plus authenticated evidence references and requested scope; duplicate id with a different digest is rejected.	Same PROPOSE ; DECIDE and ASSURE resolve obligations and may issue C_q .
INSTALL	Distribute signed Π_v or already issued G to the declared local enforcement audience with acknowledgments; honor activation and revocation epochs.	POLICY_SYNC , GRANT , and REVOKE . Installation is not admission or grant issuance.
EXECUTE	Submit the exact canonical action and G to a declared enforcement point; check scope, epoch, nonce, and live predicate at its adapter level before dispatch.	REDEEM followed by effect or explicit failure. Local validation may use cached, bounded policy/revocation state.
REPORT	Return signed attempt and outcome receipts or an explicit unknown status, linked to q , C_q , G , and resource observation when available.	OUTCOME and RECORD ; delayed or reordered reports reconcile by stable identifiers.

10.1 Normative semantic contract

For a declared protected mutation surface, KedgeFlow Protocol v0.1 implementations **MUST** satisfy the following interface requirements. The terms specify observable behavior, while Section 9 specifies the existing admission and redemption chain.

1. **Identity and topology.** Authenticate runtime, principal, controller, issuer, provider, enforcement point, and target identifiers. Version capability and resource descriptors; distinguish an advertised tool from a verified mutation path.
2. **Proposal identity.** Canonicalize q once, bind all admission and grant records to $H(q)$, and reject conflicting reuse of id . Evidence references, expected effects, and requested scope may be attached as typed metadata; they do not change the canonical action without re-proposal.
3. **Observation.** Tag every observation with source, time, resource/version scope, and freshness status. The controller **MUST NOT** treat a missing or stale dependency as a verified absence.
4. **Admission and installation.** Apply the active Π_v to realized witnesses, mint C_q only after complete discharge, and issue G only after independent certificate validation. **INSTALL MUST NOT** broaden G or change its signed fields.
5. **Local enforcement.** The enforcement point **MUST** validate G at redemption, durably control its nonce, and enforce the declared Level A/B/C state contract. A controller round trip for every redemption is **NOT REQUIRED** when policy and revocation caches satisfy their signed freshness bounds.
6. **Revocation and conflict.** Use monotonic policy/revocation epochs and Eq. 3; reject conflicting controller authority for the same resource scope unless a signed federation rule resolves ownership. Loss of a current epoch prevents new redemption after its declared bound.
7. **Outcome and uncertainty.** Produce an attempt receipt and authoritative outcome or explicit unknown status. Reordered or duplicate **REPORT** messages **MUST** be idempotently reconciled; an uncertain external effect cannot be retried as if known absent.
8. **Coverage.** Publish a versioned inventory for in-scope resources/actions and prove that no unmediated mutation credential or route remains. Native and adapted runtimes are subject to the same requirement for an authorization-safety claim.

The minimum portable schemas are identity/capability and resource descriptors; an observation with provenance and freshness; canonical proposal q ; evidence references and witness manifest; signed policy version/epoch; C_q ; G with φ_s , audience, validity, and nonce; revocation record; and execution attempt/outcome receipt. These fields reuse Eqs. 1, 6, and 7. A transport binding may use Hypertext Transfer Protocol (HTTP) with JavaScript Object Notation (JSON), gRPC remote procedure calls, a local socket, or an adapter compatible with the Model Context Protocol (MCP), but the transport does not define admission semantics. Cryptographic profiles, serialization, federation ownership, and conformance vectors require a separate interoperable wire specification.

11 From evidence to authority

11.1 Judgment is typed evidence

A decision provider returns a typed record about one question and one authenticated state view. The notation makes those bindings explicit:

$$\text{evaluate}(S_D, Q_D) \longrightarrow E_D^*. \quad (4)$$

Here D identifies the provider, S_D its observed state view, Q_D the versioned question schema, and E_D^* the candidate evidence item. The record includes provider and version, question schema, state digest,

typed result, observation time, provenance, and known shared-failure domains. Probability distributions and calibration metadata are optional annotations. A deterministic check, human attestation, model score, or formal proof can use the same transport envelope, while policy still decides which type and result can satisfy each obligation.

For the database replica example, a semantic provider evaluates whether “reduce spend without impacting availability” permits removing a specific replica. A deterministic provider verifies minimum replica thresholds and failure-domain spread. Telemetry providers report recent load and failover health. A human operator may attest to an authorized maintenance window. KedgeFlow adapters can integrate deterministic verifiers, Jev, independent LLM evaluators, formal theorem provers, telemetry sinks, or human review queues. Jev serves as one candidate semantic provider if its version and question schemas are pinned; the architecture neither mandates it nor makes vendor-specific performance claims. **Providers produce non-authoritative evidence; an affirmative evaluation never conveys mutation authority.**

11.2 Assurance is action-specific

For each proposal, policy produces the checks required under the current admission state. Let $\Omega_{\Pi_v}(q, s_a)$ be that set for proposal q , policy Π_v , and state s_a . Admission requires an accepted, realized witness for every obligation:

$$\text{Admitted}_{\Pi_v}(q, E, s_a) \equiv \bigwedge_{\omega \in \Omega_{\Pi_v}(q, s_a)} \text{Discharge}(\omega, E, q, s_a). \quad (5)$$

For each obligation, discharge checks witness authenticity, question and state bindings, evidence type and result, freshness, and required source diversity. It returns SATISFIED, VIOLATED, or UNKNOWN. The assurance service can mint an admission certificate only when every result is SATISFIED. CAC provides one model of these obligations [5]; AAS provides one optional evidence-acquisition strategy under cost and latency bounds [7]. The assurance service evaluates the evidence actually received, regardless of how it was scheduled.

Predicate correspondence. Evidence must substantiate the exact predicate mandated by an assurance obligation, not an adjacent or relaxed assertion. An affirmative result for predicate P_e cannot discharge an obligation requiring P_r unless policy explicitly proves or defines $P_e \Rightarrow P_r$. Semantic similarity or provider confidence scores cannot substitute for formal predicate correspondence.

Definition 2 (Admission certificate). An admission certificate is the assurance service’s integrity-protected record of which proposal passed which checks, under which policy and evidence, at admission time:

$$C_q = \langle cid, H(q), v, H(\Omega_{\Pi_v}(q, s_a)), H(E^*), b_s, F, \\ t_a, t_e, aid, aver, k_A, \sigma_A \rangle. \quad (6)$$

Here cid is a unique certificate identifier; $H(q)$ is the canonical proposal digest; v is the active policy version; s_a is the system state evaluated at admission; $\Omega_{\Pi_v}(q, s_a)$ is the resolved obligation set; E^* is the manifest of realized witnesses discharging those obligations; b_s cryptographically binds the evaluated state version and expected-state predicates; F records applicable per-witness freshness bounds; t_a, t_e define admission and certificate-expiration timestamps; $aid, aver$ identify the assurance service and its implementation version; and k_A, σ_A denote the service’s signing key identifier and digital signature over all preceding fields. Possession of C_q attests that the assurance service verified that all obligations required by Π_v for q were satisfied under evidence E^* at time t_a .

C_q does not attest that the action remains safe indefinitely, that evidence remains fresh, that policy remains current, that state remains unchanged, or that execution occurred. The grant issuer must independently verify certificate integrity, authorized service identity, proposal and policy bindings, temporal validity, requested-scope containment, and active revocation state before minting G . The issuer may trust the authorized assurance service’s discharge verdicts without re-executing semantic evaluations; that service remains in the Trusted Computing Base.

11.3 Consequence-conditioned assurance

The same evidence threshold is inappropriate for every operation. The following progression is a policy illustration, not a theorem or a universal default:

Action	Illustrative assurance profile
Read deployment status	Read-only access policy; no mutation grant.
Restart a development container	Authenticated principal and deterministic policy checks may suffice.
Reduce production capacity	Current telemetry, intent interpretation, availability invariant, and commit-time state check.
Delete a production database	Strong heterogeneous evidence, deterministic dependency and backup invariants, and possibly human approval.

Policy principle

Assurance should scale with consequence, uncertainty, and reversibility. CAC defines the obligations; an assurance service evaluates them. AAS defines acquisition strategy; an optional scheduler may implement it.

Multiple model votes do not automatically supply independent support. Two evaluators may share training data, a telemetry feed, a prompt template, or an operator. The witness manifest records known exposure domains, and the policy can demand structurally diverse sources. Agent-controlled text, retrieval, or telemetry is not an independent witness merely because a provider repeats it; provenance and source authority are checked per obligation. A policy interpreted only by the agent being governed would permit self-approval and cannot discharge an independent assurance requirement. Unknown common dependencies remain a risk and are measured empirically rather than assumed away.

11.4 The grant is the authority object

The issuer verifies the admission certificate and then creates the object the gateway can accept as authority for one bounded action.

Definition 3 (Action-bound execution grant). An execution grant is an integrity-protected capability:

$$G = \langle gid, H(q), H(C_q), p, a, r, \Theta, H(E^*), v, [t_0, t_1], \varphi_s, \delta, n, aud, k_I, \sigma_I \rangle. \quad (7)$$

It binds a unique grant identifier gid ; proposal digest $H(q)$; admission certificate digest $H(C_q)$; requesting principal p ; canonical action a ; target resource set r ; parameter constraint predicate Θ ; witness manifest digest $H(E^*)$; governing policy version v ; validity window $[t_0, t_1]$; commit-time state predicate φ_s ;

delegation constraints δ ; single-use redemption nonce n ; target gateway audience aud ; issuer key identifier k_I ; and issuer digital signature σ_I authenticating all preceding fields.

For comparison, “modify production infrastructure” is a broad credential, not a KedgeFlow grant. The following *illustrative* grant authorizes one admitted mutation; the values are an example, not an issued token. UTC denotes Coordinated Universal Time:

Bound field	Illustrative value
Principal (p)	deployment-agent-17
Action and resource (a, r)	SetReplicaCount on prod/orders/db-7
Allowed parameters (Θ)	replicas == 2
Policy version (v)	production-capacity-v18
Expected state (φ_s)	resourceVersion == 88193
Validity window ($[t_0, t_1]$)	[10:40:00, 10:42:05 UTC]
Redemption (n)	Unique nonce n ; one redemption; bound gateway audience
Evidence / admission	Witness-manifest digest $H(E^*)$; certificate digest $H(C_q)$

The agent does not receive general authority to modify the database. It receives, or causes to be issued, authority for one admitted mutation under specified conditions. The example presumes that the resource can enforce the version check at commit; otherwise that field supports revalidation but cannot by itself prevent a race.

This is a narrowly scoped capability, not a new kind of cryptographic primitive [12]. KedgeFlow’s proposed contribution is the evidence-conditioned issuance and redemption lifecycle. Generic cloud or database credentials remain at the gateway. A high-risk grant may also be bound to a proof-of-possession key, not just bearer possession. Serialization, canonicalization, hash and signature algorithms, key rotation, and gateway audience are versioned wire-contract details.

11.5 Redeeming the exact action

At redemption time, the execution gateway verifies the complete authorization contract:

1. **Issuer validity:** authenticates signature σ_I under key k_I and confirms the issuer is authorized for action a and resource r ;
2. **Audience and caller binding:** ensures the gateway matches aud and that the invoking caller matches principal p (or authorized delegation lineage);
3. **Scope matching:** verifies that the runtime invocation matches canonical action a , target resource r , and parameter predicate Θ ;
4. **Temporal validity:** confirms the current timestamp falls strictly within validity window $[t_0, t_1]$;
5. **Policy compatibility and revocation:** verifies $\text{Compat}(\Pi_{v_{\text{issued}}}, \Pi_{v_{\text{current}}}, G)$ and ensures the grant or signing key has not been revoked in the active revocation epoch;
6. **Replay protection:** checks and durably expends redemption nonce n , preventing duplicate submission;
7. **State validity:** evaluates commit-time predicate φ_s against current resource state at the adapter’s declared guarantee level.

A grant is *issuer-valid* if authentic and unrevoked; *policy-valid* if compatible with the gateway’s active policy epoch; and *state-valid* if predicate φ_s holds over commit-relevant state. None of these mechanical checks guarantees that the action is semantically optimal. Upon successful validation, the gateway durably commits the nonce reservation and executes the mutation; only a Level A adapter executes the state check and mutation atomically. The gateway logs an attempt receipt, mediates execution, and captures the authoritative outcome. If a grant is issued but expires unredeemed, the audit log records its expiration rather than presuming execution.

For delegated execution, any child grant must remain strictly monotonic, attenuating authority across actions, resources, parameter predicates, audience, validity intervals, and redemption counts. Revocation halts subsequent redemptions across all gateways within declared propagation bound Δ_{rev} . Revocation cannot undo an effect already committed to protected state. While offline gateway validation maximizes availability during control-plane partitions, it widens the vulnerability window to revoked authority; this operational trade-off must be explicitly governed by deployment policy.

12 Safety properties and failure containment

KedgeFlow specifies security and safety contracts over execution traces of protected system effects. These contracts are conditional engineering guarantees grounded in explicitly stated mediation and trust assumptions, rather than unprovable claims of model infallibility.

12.1 Authorization safety versus semantic correctness

KedgeFlow distinguishes the authorization path from the quality of the decision it authorizes:

- **Authorization safety** requires that every committed, in-scope state mutation was authorized along the stipulated control path and executed under an authentic, policy-valid grant matching the exact canonical action, target resource, caller identity, and parameter predicate under complete mediation.
- **Semantic correctness (decision quality)** assesses whether that authorized action faithfully fulfills true human intent and safely accommodates unmodeled real-world environmental dynamics.

Under its stated trust and mediation assumptions, KedgeFlow specifies *authorization safety*. It does not guarantee semantic correctness: flawed policy rules, incomplete dependency graphs, or deceptive provider evidence can still admit an undesirable action. **For a completely mediated surface, a protected mutation requires discharged policy obligations and a valid capability grant. This does not establish that the policy or evidence was correct.**

Property	Required meaning
No direct execution	A proposal alone cannot trigger a protected mutation.
Judgment non-authority	Decision evidence alone cannot trigger a protected mutation.
Admission before grant	Every issued grant possesses a traceable, all-satisfied admission certificate under an active policy version.
Grant-bound execution	Every committed mutation corresponds to a successfully redeemed grant for that exact canonical action.
Evidence-bound authority	A verifier can independently reconstruct which evidence items and obligations justified issuance.
Scope monotonicity	A delegated child grant cannot expand its parent's authority across any dimension.
Revocability	After the declared propagation bound Δ_{rev} elapses, an effective revocation blocks subsequent redemption.

For a declared Level A surface, the grant-bound and state-valid execution invariant is:

$$\text{Committed}(q, s, s') \implies \exists G : \text{Redeemed}(G, q) \wedge \text{Valid}_{\text{state}}(G, s). \quad (8)$$

Proposition 1 (Conditional grant-bound and state-valid execution). Assume a declared Level A surface where:

1. **Complete mediation:** every state-changing operation on resource $r \in \mathcal{R}$ passes through a designated, unbypassable execution gateway;
2. **Strict gateway enforcement:** the gateway rejects every request lacking an authentic, issuer-valid, and policy-compatible grant G binding the exact canonical request (p, a, r, θ) ;
3. **Durable replay protection:** redemption nonce state is persisted before mutation dispatch, precluding duplicate execution; and
4. **Atomic state validation:** the resource or gateway atomically evaluates the grant's commit-time state predicate φ_s concurrently with mutation execution.

Then every committed protected mutation $(s \rightarrow s')$ corresponds to a validly redeemed grant whose authorization scope covers the exact action, parameters, and commit-time state s .

Proof sketch. Under complete mediation, no execution path reaches protected resource r bypassing the gateway. The gateway admits dispatch only if grant G is issuer-valid, policy-valid, and its nonce unexpended. The durable nonce reservation guarantees at-most-once redemption per grant. Finally, Level A atomicity guarantees that condition $\varphi_s(s)$ holds at the precise instant of state mutation. Hence, no ungranted, replayed, or state-invalid mutation can commit on a Level A surface.

Non-claims and adapter boundaries. The proposition does not establish exactly-once external effects under network partitions or unconfirmed retries, nor does it guarantee the absence of semantic errors arising from faulty policies or misleading provider evidence. For Level B (revalidated) and Level C (reconciliation) adapters, only the grant-redemption part of Eq. 8 applies; commit-time state validity against concurrent races is not guaranteed.

12.2 Adapter guarantee levels

Because heterogeneous enterprise APIs provide vastly different synchronization capabilities, KedgeFlow adapters must explicitly declare the state guarantee level they enforce for each governed action. The examples use Amazon Simple Storage Service (S3), structured query language (SQL), representational state transfer (REST), remote procedure calls (RPC), and entity tags (ETags):

Declared level	Enforcement mechanism	Real-world examples and limits
Level A: Atomic state-bound	The resource or gateway atomically evaluates the commit predicate φ_s alongside the mutation; stale-state commits are rejected.	S3 conditional PUT (ETag match), Kubernetes CAS updates, SQL serializable transactions, DynamoDB condition expressions. Prevents TOCTOU races for the guarded predicate.
Level B: Revalidated	The gateway queries and verifies state immediately prior to dispatch, but check and mutation are non-atomic.	Standard REST APIs (GET followed by POST). Substantially narrows the race window, but concurrent mutations in the gap can still commit.
Level C: Detection & reconciliation	The backing API provides no precondition checking; the gateway validates grant scope and observes outcomes post-hoc.	Unconditional webhooks, legacy RPCs, asynchronous message queues. Stale or racing commits can occur; reliance is placed on compensating actions.

All three levels require full grant validation, durable replay prevention, and complete mediation for their declared surfaces. If an API cannot constrain the action or resource to the grant's exact scope, that action remains outside KedgeFlow conformance until an enforcement adapter is deployed. A Level C integration cannot claim commit-time prevention merely because it logs a mismatch afterward.

12.3 Where the guarantee stops

The authorization guarantee fails if an agent retains an alternate credential path, a resource alias bypasses canonical matching, nonce state is lost during a crash, or an adapter falsely claims Level A while executing a non-atomic API call. A pre-state digest evaluated at admission time cannot solve a TOCTOU race by itself. Levels B and C make weaker state claims; they do not, however, weaken the requirement to validate grant scope and caller identity.

The same discipline applies to auditability. Cryptographically chained receipts make completed authorization lineages verifiably reconstructible. If grant issuance or state mutation commits while the recording service is partitioned, an outbox pattern or background reconciler must bridge the gap. Until resolved, the trace reports *unknown pending reconciliation*. Authoritative outcome evidence must originate from the gateway or protected resource itself, strictly precluding self-attestation by the requesting agent.

What KedgeFlow does not solve

KedgeFlow does not make incorrect policies correct, incomplete dependency models complete, false evidence true, compromised trust roots trustworthy, bypassable gateways safe, or non-atomic APIs atomic. Within a completely mediated surface, KedgeFlow prevents failures in generation or judgment from automatically carrying execution authority into protected state. This authorization safety does not guarantee semantic correctness or decision quality.

Practical test

Attempt to mutate an in-scope resource without a matching grant, with an expired or replayed grant, or after concurrently altering a commit-relevant state version. A successful direct or replayed mutation defeats the authorization claim; a stale-state mutation defeats a Level A claim.

13 Pilot build and validation plan

The first build is a small end-to-end implementation in an isolated test environment. It covers two actions: resizing a database replica and adjusting a deployment replica count. The agent submits a canonical proposal. The assurance service checks deterministic and semantic evidence against CAC-style obligations; an optional AAS-style scheduler can obtain or refresh that evidence. After all obligations are satisfied, the assurance service issues C_q . An independent issuer validates C_q and policy currency before minting G . The gateway holds the mutation credentials, validates and redeems G against the target resource version, and records the authoritative outcome and adapter guarantee level in an append-only store.

The build needs versioned schemas for proposals, questions, evidence, certificates, policies, grants, and receipts. It also needs a dependency and coverage registry, key management and rotation, durable nonce and revocation storage, and an independent trace verifier. Jev is a possible semantic provider if version-pinned access is available. Core security and protocol tests must run without Jev or any single proprietary provider.

13.1 Comparisons that matter

The evaluation should compare four configurations under identical workloads and policy specifications:

1. **Direct tools:** agent-generated tool calls execute directly using ambient credentials.
2. **Judge gate:** a semantic model inspects and approves or rejects tool calls immediately prior to execution.
3. **Hardened PDP/PEP baseline:** an industry-standard policy decision and enforcement architecture with real-time state querying (e.g., an Open Policy Agent gateway).
4. **Complete KedgeFlow:** full Evidence-to-Authority lifecycle, including canonical proposals, structured witness manifests, signed admission certificates, action-bound capability grants, gateway redemption, and cryptographically linked outcome receipts.

The third baseline is essential. Comparing KedgeFlow only with direct tools or a single-prompt judge would not establish its value over existing authorization infrastructure. Evaluation should therefore compare KedgeFlow with an enterprise PDP/PEP under the same policy intent and report both security and operational effects. If an extended PDP/PEP realizes the entire Evidence-to-Authority specification, it is functionally equivalent; the evaluation should report that equivalence. Controlled ablations should measure the contribution of heterogeneous witnesses, pre-execution evidence refresh, commit-time state binding, and single-use redemption nonces.

13.2 What to measure and how to challenge it

Failure events must be categorized into disjoint, rigorously defined failure classes:

- **Unauthorized commit:** a protected mutation executes without a valid, gateway-redeemed grant corresponding to the exact canonical action, resource, and parameters.
- **Contract-violating commit:** a mutation executes in breach of an explicit policy obligation, temporal validity window, or commit-time state predicate φ_s , regardless of whether a certificate or grant was minted erroneously.
- **Semantic failure:** a mutation fully satisfies the formal protocol and authorization path, yet produces undesirable operational outcomes under an external domain oracle due to specification flaws, erroneous evidence, or unmodeled dependencies.

Empirical evaluations must report counts and conditional rates for each failure mode under explicitly specified oracles, rather than aggregating results into an undefined “unsafe execution” metric.

Question	Measurement or deliberate attack
Control-plane overhead	Median, 95th, and 99th percentile latency; throughput under concurrency; central processing unit (CPU), memory, grant size, and signature verification overhead.
Commit failures by class	Under prompt injection, poisoned evidence, malicious retrieval, broad parameters, stale state, and compromised providers, count unauthorized, contract-violating, and semantic failures separately.
False denial	Legitimate operations blocked, by reason code and workload stratum.
Escalation efficiency	Fraction resolved by deterministic check, cheap semantic provider, multiple witnesses, stronger verifier, or human review; cost and latency by path.
Correlation	Paired failures for same-family judges, heterogeneous providers, and deterministic cross-checks under shared misleading inputs.
TOCTOU	Concurrent modification of commit-relevant state versions following grant issuance; measure race detection across Level A, B, and C adapters.
Replay and theft	Concurrent duplicate redemption, crash-recovery replay tests, cross-principal theft, wrong gateway, and parameter tampering.
Evidence completeness	Blind third-party verification of authorization lineage, obligation satisfaction, and outcome linkage from persistent receipts.
Availability	Network partitioning of providers, issuer, or revocation service; measure denial rates, backlog, and bounded-staleness window enforcement.
Adapter portability	Run the same canonical proposal and policy over two different agent runtimes; compare schema equality, intercepted mutation coverage, and receipt verification.
Cross-agent conflict	Inject a concurrent deployment/termination dependency and stale capacity observation; measure conflict detection, defer/deny accuracy, and stale-view failures.

Here $\Pr$ denotes probability, $F_{\text{controller}}$ a controller failure, and $F_{\text{generator}}$ a generator failure. The primary security metric is $\Pr(\text{unauthorized commit} \mid \text{injected fault})$; contract violations and semantic failures must be reported as distinct conditional rates stratified by fault category and adapter guarantee level. The primary usability metric is the false rejection rate $\Pr(\text{deny} \mid \text{legitimate proposal})$. For epistemic dependency analysis, evaluations must estimate the joint error probability $\Pr(F_{\text{controller}} \mid F_{\text{generator}})$ across paired trials, rejecting consensus agreement as a proxy for ground-truth correctness. Published results must document exact confidence intervals, corpus generation methodology, policy specifications, model checkpoints, question schemas, raw receipts, and measurement timestamps. Vendor pricing and latency figures must be tied to specific versioned endpoints and runtime environments.

The initial falsification suite must target: every alternate credential path in the coverage inventory; state race conditions between grant validation and commit; duplicate redemption following simulated gateway crashes; policy rollback and emergency revocation during network partitions; wrong-audience and scope-expansion redemption attempts; and deliberate loss of outcome receipts following committed effects. These adversarial tests probe the assumptions underlying the strongest guarantees. A stale-state commit under a Level B or C adapter illustrates its declared race window; it does not test a Level A guarantee.

14 Using existing authorization systems

KedgeFlow can be implemented with established authorization and agent infrastructure. The U.S. National Institute of Standards and Technology (NIST) Zero Trust Architecture defines policy decision points (PDP), policy administration points (PAP), and policy enforcement points (PEP) [2]. Open Policy Agent (OPA) separates policy evaluation from application logic [3]; Zanzibar shows large-scale relationship-based access control [13]. Reference monitors supply the complete-mediation principle [14], and object capabilities supply patterns for narrow and delegated authority [12]. The Model Context Protocol (MCP) standardizes tool interfaces and incorporates security boundaries [15]. These systems provide components and design principles that a KedgeFlow implementation can use.

The KedgeFlow specification puts these pieces into one runtime-neutral contract: a boundary between agent decisions and execution authority; shared, provenance-tagged state; canonical proposals; action-specific evidence obligations; narrow grants; gateway redemption; and linked outcomes. A conventional PDP/PEP implementation that realizes the full contract is functionally equivalent. Related work already covers parts of this path: Proof-Carrying Agent Actions describes certificate-bearing governance [16], Canonical Action Verification and Attestation (CAVA) studies canonical action identity [17], and Microsoft’s draft Agent Control Specification (ACS) defines intervention-point policy evaluation with host enforcement [18]. KedgeFlow’s design choice is to make cross-agent admission state and the evidence-to-grant-to-resource-enforcement chain one interface contract. Comparative value still requires measurement.

Other public proposals overlap substantially. The Agent Infrastructure Control Protocol is an Internet Engineering Task Force (IETF) Internet-Draft for transport-independent infrastructure capability, situation, intent, plan, authorization, operation, and outcome objects [19]. The Agent Governance Protocol links multi-agent proposals, evidence, authority, resolution, and audit [20]. The Handle-Capability Protocol studies grant-backed execution invariants for MCP-style runtimes [21]. The acronym *ACP* is used by IBM’s Agent Communication Protocol [22] and by a separate Agent Control Protocol for history-aware admission control [23]. KedgeFlow is distinct from both. These overlaps rule out a claim that a portable agent control protocol or action-level authority was first proposed here. Interoperability and measured comparative value remain open.

15 Connecting current agent runtimes

The table is a documentation-level comparison as of 29 September 2026, not an audit of every product configuration. SDK means software development kit; ADK means Agent Development Kit; A2A means Agent2Agent. **S** means an explicit documented feature for the named column; **F** a framework-specific feature that an adapter can use; **P** partial coverage; **ND** means that the cited public interface does not specify the complete function. In particular, ND does not mean an implementation is impossible. “A/R” asks for a *portable* action-bound authority-installation and resource-side receipt contract, not merely a trace or an approval prompt.

Interface	Discover	Observe	Pre-action	Policy	A/R
OpenAI Agents SDK[24]	F	S	S	F	ND
Claude Agent SDK[25]	F	S	S	F	ND
Google ADK[26]	F	S	S	F	ND
Microsoft Agent Framework[27]	F	S	S	F	ND
LangGraph[28]	F	S	F	F	ND
MCP[15]	S	P	ND	P	ND
A2A[29]	S	P	ND	P	ND
OpenClaw[30]	F	S	S	F	ND
Hermes Agent[31]	F	S	S	F	ND

These systems expose real control surfaces. OpenAI documents per-tool guardrails and traces; Claude documents pre-tool hooks and permission modes; ADK and Agent Framework document before-tool callbacks or function middleware; LangGraph supplies durable graph state and interrupts; OpenClaw and Hermes expose pre-tool hooks. Microsoft’s Agent Framework is presented as the direct successor to AutoGen [27]. Their exact coverage varies by tool type, hosting mode, and configuration, so each adapter must test its actual intervention surface. MCP standardizes tool/resource discovery and HTTP authorization, while A2A standardizes agent discovery, tasks, and an authorization-required state; neither claims to define KedgeFlow’s complete cross-agent admission and grant-redemption lifecycle. A2A explicitly leaves authorization scope, validity, and revocation semantics to implementations [29]. Microsoft’s ACS is closer to a common external policy engine, but its draft spec evaluates one intervention snapshot at a time and assigns enforcement to the host [18]. These are potential substrates for KedgeFlow, not weak substitutes invented for comparison.

The architectural distinction concerns authority and credential custody. Agent SDK hooks and guardrails can intercept tool calls, but their enforcement scope depends on the configured tool paths and credentials. A hook response alone cannot protect a resource if the runtime retains another path to mutate it. KedgeFlow therefore requires complete mediation for each declared protected surface: the agent runtime holds no credential for that mutation, and an independent enforcement point validates the action-bound grant G before using its backing credential. Commit-time state validation depends on the declared adapter level.

Across the surveyed runtime interfaces, there is no broadly adopted common mapping from *observed proposed transition* to *action-bound authority installed at an independently checked mutation boundary*, with policy/version and outcome correlation across heterogeneous runtimes. Emerging drafts and projects above address parts or much of this space. Whether one of them or an existing governance system can realize the full KedgeFlow contract is an empirical interoperability question for the implementation in Section 13.

16 Evidence needed for deployment claims

The protocol specifies what to build; deployment claims require evidence that the implementation actually meets its assumptions. The following checks bound what a team can claim and add no protocol stages or conformance requirements:

1. **Comparative value.** Any claimed advantage over a hardened PDP/PEP with capability tokens and verifiers needs a matched evaluation of security and operational effects.
2. **Enforcement coverage.** Verify the coverage inventory and resource-side permissions to exclude gateway bypass. Claim Level A state protection only where the commit-relevant predicate is enforced atomically; otherwise declare Level B or C and its race window.

3. **Availability and revocation.** Measure gateway replicas, short-lived grants, policy epochs, key rotation, and revocation during partitions before claiming an availability or revocation bound.
4. **Evidence and assurance.** Preserve the meaning, provenance, and question scope of each evidence type; record deterministic checks and probabilistic policy thresholds; test known witness dependencies and bound diversity claims where common dependencies are unknown.
5. **Recovery and delegation.** Report uncertain effects as unknown until reconciliation, without inferring exactly-once effects from redemption. Verify child-grant scope and revocation across the claimed delegation lineage.
6. **Decision quality.** Use an independent oracle to report protocol violations separately from semantically bad but authorized actions.

These checks bound deployment and comparative claims. Where evidence is missing, state the narrower guarantee rather than implying that the specified protocol has already delivered a measured result.

17 Control-plane separation in OpenFlow and KedgeFlow

OpenFlow illustrates how a controller can program forwarding behavior that is carried out locally [4]. KedgeFlow applies that separation to consequential agent operations. Its control object is a proposed state transition rather than a packet: admission evaluates the proposal against policy, evidence, dependencies, and current resource state. The resulting bounded authority is checked where the protected effect occurs, and the outcome is reported to the controller. Figure 8 shows the shared controller/local-execution pattern and the KedgeFlow-specific admission path.

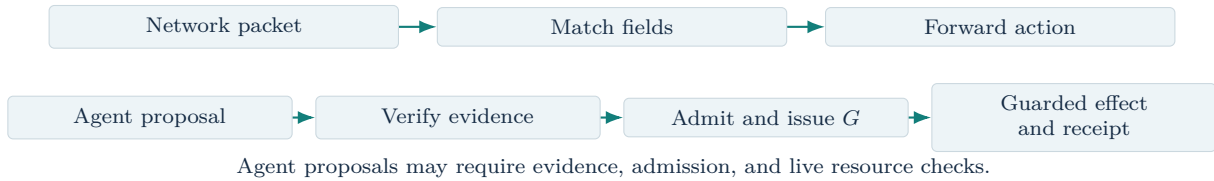


Figure 8: OpenFlow illustrates controller-programmed local forwarding. KedgeFlow applies control-plane separation to proposed state transitions and their protected effects.

18 Failure semantics and deployment boundaries

The controller can be replicated or federated, but a resource scope must have unambiguous issuance authority. During a partition, local enforcement may accept a previously installed, unexpired G only while signed policy and revocation state remain within Δ_{sync} and Δ_{rev} . No fresh grant is issued from an unavailable assurance or issuer service. Conflicting controllers must not both issue for the same scope without a signed ownership/epoch rule. This is *bounded degraded operation*; after bounds expire, redemption fails closed. Fail-open execution is outside the KedgeFlow authorization-safety claim.

Condition	Required interpretation
Stale policy or global state	Unknown or failed freshness obligation; reobserve, defer, or deny. A cached grant still obeys Eq. 3.
Disconnected or compromised agent	No implicit authority. Existing grants remain bounded; remove agent credentials and revoke affected grants within the declared propagation bound.
Compromised adapter or enforcement point	Adapter compromise can falsify observations or proposals; gateway compromise invalidates its mediated-surface guarantee. Resource-side controls and audit delimit the exposure.
Duplicate proposal or replayed grant	Reject conflicting $id/H(q)$ reuse; durably expend n at redemption. External idempotency is a separate resource property.
Delayed or reordered report	Correlate by proposal, grant, and attempt identifiers; retain unknown status until authoritative reconciliation.
Partial or uncertain effect	Do not infer success, rollback, or exactly-once execution from a missing receipt. Reconcile against the resource before retry.

The trusted base includes the policy authority, assurance service, issuer keys, enforcement point, replay state, and target resource’s relevant atomicity and identity checks. Evidence providers can be faulty or compromised; policies must identify which provider classes may discharge each obligation. A false provider claim or incomplete dependency graph can make an admitted action semantically bad. KedgeFlow prevents failures in generation or judgment from automatically carrying execution authority into protected state; it does not guarantee decision quality.

19 Implementation path

Build one protected operation before adding more runtimes or federation. Define the canonical identity, capability, resource, observation, proposal, evidence-reference, policy-epoch, certificate, grant, guard, revocation, and receipt schemas. Implement the operations in Section 10 with one transport binding and conformance fixtures. A Level A-capable resource is a suitable first vertical slice. Add one framework adapter and one independent gateway; verify complete mediation and nonce recovery before expanding to heterogeneous agents or controller replication. The controller does not need to own prompting, memory, model selection, planning, or application orchestration.

An institution can retain control over policy, evidence requirements, risk bounds, issuance keys, and audit while using different models and agent runtimes. Federation and sovereign operation need explicit resource ownership, epoch, and trust agreements; geographic deployment alone does not establish those properties.

20 Implementation collaboration

OpenKedge is developing KedgeFlow as an open control-plane framework for consequential autonomous operations. Teams can use their own agent runtimes, policy engines, evidence providers, assurance methods, and gateways if they meet the published contract. Future tools and theories can be added through those roles without changing the Evidence-to-Authority boundary. Engineering and security teams can test the candidate protocol against protected actions in cloud infrastructure, IAM systems, production databases, and sensitive APIs. The validation work in Section 13 defines evidence needed for deployment claims; this document does not claim pre-existing KedgeFlow deployments.

A practical adoption engagement begins with a single high-consequence operation: inventory all existing mutation paths to establish complete mediation, define the canonical action schema and assurance obligations, issue action-bound grants, declare the adapter guarantee level, and systematically execute the bypass and race test suites detailed in Section 13. Existing agent frameworks and prompt pipelines remain intact above the boundary, while mutation credentials and execution enforcement transition to dedicated gateway control.

Gateway adapters, conformance tests, empirical benchmarks, and reference implementations can test the proposed contract. Concrete bypass attempts, replay tests, and state-race tests are especially useful for checking its trust assumptions.

21 Glossary and protocol status

Term	Meaning in this specification
Agent Data Plane	Model inference, generation, planning, memory and tool use, and locally authorized execution within an agent runtime.
KedgeFlow controller	Logically centralized admission and authority-management role with a provenance-tagged, versioned cross-agent view; it need not be one physical server.
KedgeFlow Agent Adapter	Translation layer mapping a framework’s hooks, state, and tool calls to KedgeFlow southbound semantics; it is not itself a non-bypassable enforcement point.
Northbound / southbound interface Proposal	Institutional intent and policy input / runtime and enforcement discovery, observation, proposal, installation, execution, and reporting. Canonical, non-authoritative structured request for a typed protected mutation.
Evidence-to-Authority boundary	The architectural and protocol boundary that transforms verified, action-specific assurance into a narrowly scoped execution grant.
Decision provider	An independent evaluation component generating typed candidate evidence; includes LLM evaluators, deterministic verifiers, formal provers, telemetry sources, and human reviewers.
Candidate evidence	Structured, versioned evaluation result binding question schema, state view, observation timestamp, and provenance metadata; conveys zero execution authority.
Witness	A realized candidate evidence item verified and accepted to discharge a named assurance obligation.
Assurance obligation	A policy-specified predicate and witness requirement governing a canonical action under defined environmental conditions.
Admission certificate (C_q)	An integrity-protected attestation minted by the assurance service certifying that all obligations for a proposal were satisfied under verified evidence at admission time; confers no execution authority.
Execution grant (G)	A signed capability binding principal, action, resource, parameter predicate, policy version, validity window, commit-time state predicate, and single-use nonce.
Redemption	Gateway validation and durable expenditure of an execution grant at the protected mutation boundary immediately prior to dispatching an effect.

Term	Meaning in this specification (continued)
Authoritative outcome	A verified observation of execution success, failure, or unreconciled status emitted directly by the execution gateway or target resource.
Authorization safety / decision quality	Grant-bound complete mediation under stated trust assumptions / semantic correctness of the policy, evidence, dependency model, and resulting action.
Adapter guarantee level	A declared classification of state-bound execution enforcement: atomic state-bound (Level A), pre-execution revalidated (Level B), or detection and post-hoc reconciliation (Level C).
Epistemic fault domain	A shared latent dependency (e.g., training data, prompt templates, retrieval corpus, or operational infrastructure) capable of inducing correlated failures across seemingly distinct providers.

Normative intent

The uppercase message names and the grant fields are proposed interoperable protocol elements. The exact serialization, cryptographic suite, authentication profile, resource registry, policy language, and deployment topology remain to be specified. A future interoperable version would need conformance tests for canonicalization, state predicates, nonce persistence, revocation, and outcome reconciliation.

Document status

This is a proposed architecture and protocol, not an implementation report. Formal properties are conditional on the stated trust and enforcement assumptions. No KedgeFlow prototype results or vendor performance claims are reported. OpenAI Codex assisted with structure and wording; the authors remain responsible for technical and source verification.

References

- [1] Vincent C. Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (ABAC) definition and considerations. Technical Report SP 800-162, National Institute of Standards and Technology, 2014. Updated August 2019.
- [2] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero trust architecture. Technical Report SP 800-207, National Institute of Standards and Technology, 2020.
- [3] Open Policy Agent. Open policy agent documentation: How to deploy opa. <https://www.openpolicyagent.org/docs/deploy>, 2026. Accessed 2026-09-27.
- [4] Open Networking Foundation. OpenFlow switch specification, version 1.5.1. Technical report, Open Networking Foundation, 2015.
- [5] Jun He and Deying Yu. Cognitive admission control: Risk-conditioned assurance for consequential actions in agentic distributed systems. *arXiv preprint arXiv:2609.16313*, 2026. <https://arxiv.org/abs/2609.16313>.
- [6] Jun He and Deying Yu. Persistent cognitive identity: A systems architecture for continuity across AI substrates and embodiments. OpenKedge technical manuscript. <https://www.openkedge.io/paper/persistent-cognitive-identity.pdf>, 2026.
- [7] Jun He and Deying Yu. Before agents act: Assurance-aware semantic scheduling for evidence acquisition in distributed systems. *arXiv preprint arXiv:2609.34376*, 2026. <https://arxiv.org/abs/2609.34376>.

- [8] Jun He and Deying Yu. The honest quorum problem: Epistemic byzantine fault tolerance for agentic infrastructure. *arXiv preprint arXiv:2607.16109*, 2026. <https://arxiv.org/abs/2607.16109>.
- [9] Jun He and Deying Yu. Agent-native telemetry: Verifiable state-delta evidence for autonomous operations. *arXiv preprint arXiv:2608.16178*, 2026. <https://arxiv.org/abs/2608.16178>.
- [10] Jun He and Deying Yu. When ai agents commit: Cognitive serializability across data, evidence, policy, and authority. *arXiv preprint arXiv:2609.20261*, 2026. <https://arxiv.org/abs/2609.20261>.
- [11] Jun He and Deying Yu. Sovereign execution broker: Enforcing certificate-bound authority in agentic control planes. *arXiv preprint arXiv:2606.20520*, 2026. <https://arxiv.org/abs/2606.20520>.
- [12] Mark S. Miller, Ka-Ping Yee, and Jonathan Shapiro. Capability myths demolished. Technical Report SRL2003-02, Johns Hopkins University Systems Research Laboratory, 2003.
- [13] Ruoming Pang, Ramon Caceres, Mike Burrows, Zhifeng Chen, Pratik Dave, Nathan Germer, Alexander Golynski, Kevin Graney, Nina Kang, Lea Kissner, Jeffrey L. Korn, Abhishek Parmar, Christina D. Richards, and Mengzhi Wang. Zanzibar: Google’s consistent, global authorization system. In *Proceedings of the 2019 USENIX Annual Technical Conference*, 2019.
- [14] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.
- [15] Model Context Protocol. The 2026-07-28 specification. <https://blog.modelcontextprotocol.io/posts/2026-07-28/>, 2026. Specification release overview; accessed 2026-09-27.
- [16] Zexun Wang. Proof-carrying agent actions: Model-agnostic runtime governance for heterogeneous agent systems. *arXiv:2606.04104*. <https://arxiv.org/abs/2606.04104>, 2026.
- [17] Zexun Wang. Cava: Canonical action verification and attestation for runtime governance of agentic ai systems. *arXiv:2607.13716*. <https://arxiv.org/abs/2607.13716>, 2026.
- [18] Microsoft. Agent control specification. <https://github.com/microsoft/agent-governance-toolkit/blob/main/policy-engine/spec/SPECIFICATION.md>, 2026. Draft specification accessed 2026-09-29.
- [19] Tihan-Nico Paxton. Agent infrastructure control protocol. Technical Report draft-paxton-aicp-00, IETF, 2026. Internet-Draft, work in progress. <https://www.ietf.org/archive/id/draft-paxton-aicp-00.html>.
- [20] Agent Governance Protocol. Agent governance protocol: Public review draft. <https://agpprotocol.org/>, 2026. Project documentation accessed 2026-09-29.
- [21] Ting Liu. From tool connection to execution control: Benchmarking security invariants in MCP-style agent runtimes. *arXiv preprint arXiv:2606.29073*, 2026. <https://arxiv.org/abs/2606.29073>.
- [22] IBM Research. Agent communication protocol. <https://research.ibm.com/projects/agent-communication-protocol>. Project documentation accessed 2026-10-01.
- [23] Marcelo Fernandez. Agent control protocol: Admission control for agent actions. *arXiv preprint arXiv:2603.18829*, 2026. <https://arxiv.org/abs/2603.18829>.
- [24] OpenAI. Openai agents sdk: Guardrails and tracing. <https://openai.github.io/openai-agents-python/guardrails/>; <https://openai.github.io/openai-agents-python/tracing/>, 2026. Documentation accessed 2026-09-29.
- [25] Anthropic. Claude agent sdk: Configure permissions. <https://code.claude.com/docs/en/agent-sdk/permissions>, 2026. Documentation accessed 2026-09-29.
- [26] Google. Agent development kit: Types of callbacks. <https://adk.dev/callbacks/types-of-callbacks/>, 2026. Documentation accessed 2026-09-29.
- [27] Microsoft. Microsoft agent framework overview. <https://learn.microsoft.com/en-us/agent-framework/overview/>, 2026. Documentation accessed 2026-09-29.
- [28] LangChain. Langgraph: Interrupts. <https://docs.langchain.com/oss/python/langgraph/interrupts>, 2026. Documentation accessed 2026-09-29.
- [29] A2A Protocol Working Group. Agent2agent protocol specification, version 1.0.0. <https://a2a-protocol.org/latest/specification/>, 2026. Specification accessed 2026-09-29.
- [30] OpenClaw. Plugin hooks and tool call policy hooks. <https://docs.openclaw.ai/plugins/hooks/tool-policy>, 2026. Documentation accessed 2026-09-29.
- [31] Nous Research. Hermes agent: Hooks. <https://github.com/NousResearch/hermes-agent/blob/main/website/docs/user-guide/features/hooks.md>, 2026. Repository documentation accessed 2026-09-29.