Replay Is No Longer Replay

From Event Observability to Epistemic Architecture

This is the second article in a three-part doctrinal series on AI governance. Article 1 established that agents are implementation choices for bounded capabilities. This article examines what changes when those implementations are probabilistic. Article 3 addresses how to govern the consequences of decisions across a system of probabilistic agents.

The core proposition: probabilistic systems require a new architecture for distributed systems, where every computational assertion carries the epistemic conditions of its own production and is accompanied by explicit authority boundaries.

1. The Fundamental Problem, Replay(Event) is not Replay(Decision)

In classical deterministic systems, replay appears straightforward. An event enters a deterministic service. The service processes it. The output is reproducible.

But this reproducibility rests on assumptions that prove fragile even in classical systems. A database changes. A remote API updates. Configuration shifts. Event order in a queue differs. Exact replay is often impossible, even in deterministic systems.

The structural difference in probabilistic systems is not that replay becomes impossible. It is that the dimensionality of what must be preserved for meaningful replay increases.

Formally:

$$D_t = f(E_t, S_t, C_t, M_t, R_t, T_t, xi_t)$$

where:

- E = Event
- S = Execution state
- C = Context
- M = Model
- R = Retrieved evidence
- T = Tools invoked
- xi_t = Uncontrolled or unrecorded execution variance

Exact replay is not intrinsically impossible. But it requires freezing enough of the execution environment (random seeds, runtime versions, model weights, retrieved corpus state, tool responses, hardware configuration) that residual variance becomes negligible. That preservation has a cost.

The architectural reality: reproducibility cost increases with dimensionality.

> **Cost_reproducibility = f(StateDepth, EnvironmentDepth, EpistemicDepth, VarianceTolerance)**

The novel dimension is EpistemicDepth. This is what probabilistic systems add. You must now preserve not just execution state and environment, but the epistemic conditions under which the assertion was produced.

Therefore: Exact replay is an economic choice, not a technical binary. The question is not "Can we replay?" but "What level of reproducibility does this decision's consequence justify?"

2. The Two Layers of Observability, Events and Assertions

In classical systems, the unit of observability is the event. Event logs construct system history.

In probabilistic systems, a new layer emerges: the assertion - the result of a computation that carries epistemic uncertainty.

We must now distinguish carefully:

Event: What happened (input, trigger, message)

Execution: How it was processed (technical conditions, computational path)

Assertion: What was produced (computational result with epistemic status)

Decision: What action was chosen (based on assertion plus policy plus authority)

Each answers a distinct question and carries distinct metadata:

```
Event Envelope (What?):
{
  event_id, timestamp, producer, causal_parent, correlation_id
}

Execution Envelope (How?):
{
  model_version, prompt_version, parameters, tools_invoked,
  retrieved_sources, runtime_config, state_snapshot
}

Assertion Envelope (What result, with what epistemic status?):
{
  value: "Severity=HIGH",
  epistemic_status: INFERRED,
  confidence: 0.78,
  evidence: [Case_001, Case_045],
  validity_domain: "outpatient_notifications",
  expiration: T+90_days
}

Decision Envelope (What action, with what authority?):
{
  decision: "Escalate",
  policy_applied: escalation_policy_v2,
  authority: read_only,
  decision_basis: "assertion severity HIGH plus policy rule 3.2"
}
```

This progression now mirrors the classical observability evolution: monitoring (Event) -> observability (Assertion) -> auditability (Decision with justification).

The critical insight: Assertion is the epistemic unit. Decision is the operational unit. They must not be conflated.

3. State Governance, Context as Audit Record

Probabilistic execution is fundamentally context-dependent. Behavior at time T depends on conditions at time T minus 1.

Behavior_t = f(Event_t, Context_t-1)

Context becomes part of the audit record. But preservation must be proportional to consequence:

Persistence_depth = f(Decision_criticality, Reversibility, Audit_requirement, Privacy_risk, Storage_cost)

Low-risk assertions require only event plus assertion plus model version. High-risk decisions require full execution envelope plus state snapshots. Regulated decisions require immutable provenance chains.

The architectural principle: do not demand exhaustive state preservation everywhere. Buy the level of context retention justified by the decision's consequence.

4. Epistemic Typing - A New Architectural Discipline

Classical type systems encode structure:

Classical typing: "This is a string"

Semantic typing added meaning:

Semantic typing: "This is a severity classification"

Epistemic typing now encodes how we know:

Epistemic typing: "This is a severity inferred by SafetyClassifier_V3.0

from cases [001,045] with confidence 0.78,

valid for outpatient domain, expiring at T+90 days"

Epistemic typing is an architectural discipline (not necessarily a formal programming-language type system) that governs:

- What epistemic statuses are permitted
- What transformations are allowed between statuses
- What operations require explicit policy gates
- When assertions expire or lose validity

The Epistemic Status Taxonomy

Epistemic types are multidimensional:

Epistemic_type = Origin x Method x TemporalStatus

Origin: Sensor | Human | Model | Policy | ExternalSource

Method: Observed | Derived | Inferred

TemporalStatus: Current | Predicted | Historical

Example combinations:

(Model, Inferred, Predicted) -> Model output for future states

(Human, Observed, Current) -> Clinician observation right now

(Policy, Derived, Current) -> Rule-based derivation now

(Sensor, Observed, Historical) -> Logged sensor data from past

This multidimensional structure avoids forcing false choices between INFERRED and PREDICTED - they are independent axes.

The Epistemic Algebra

Epistemic typing enables formal rules for transformation:

(Model, Inferred, Current) + [validation_by: Human]

-> (Human, Observed, Current)

(Model, Inferred, Predicted) + [policy_gate: X]

-> (Policy, Derived, Predicted)

Expired(assertion)

-> InvalidForDecision(assertion)

Critically: there is NO implicit transformation:

(Model, Inferred, Current)

-/-> (Human, Observed, Current) [WITHOUT explicit validation]

This rule prevents epistemic laundering - the unauthorized collapse of epistemic types.

Epistemic laundering is precisely:

unauthorized_transformation(assertion_type)

= violation of the epistemic algebra

An architecture with epistemic type checking can enforce these rules at contract boundaries, preventing cascading loss of epistemic precision.

5. Authority : The Second Orthogonal Dimension

Epistemic status answers "How do we know this?"

Authority answers "What is this information allowed to cause?"

These are independent:

(Model, Inferred, Current) + authority: advisory

-> can inform recommendations

(Model, Inferred, Current) + authority: executable

-> can trigger actions (higher risk)

(Human, Observed, Current) + authority: read_only

-> information only

(Human, Observed, Current) + authority: executable

-> authorized action

Risk is therefore multifactorial:

Risk(assertion) = EpistemicRisk(assertion_type)

x AuthorityRisk(authority_level)

x IrreversibilityRisk(effect)

An epistemic assertion with zero authority poses no operational risk, regardless of its uncertainty.

The same assertion with executable authority becomes systemic.

6. The Reproducibility Ladder

Reproducibility is not binary. It is a graduated ladder of guarantees:

Effect Idempotency: Does duplicate delivery create unintended duplicate effects? This is a classical distributed systems problem. Cheap. Generally required unless the operation is intentionally non-idempotent (e.g., "record a new measurement").

Assertion Stability: Under the same frozen context (same model version, same evidence, same policy state), does the assertion category remain stable? The magnitude may shift (confidence 0.78 to 0.82 due to residual variance), but the classification (HIGH vs. LOW) remains. Moderate cost. Required for medium-risk decisions.

Execution Reproducibility: Can you obtain exactly the same assertion if you replay all of (E, S, C, M, R, T)? Depends on xi_t. Expensive or impossible without extraordinary measures. Required only for high-risk or regulated decisions.

Governance scales to consequence:

> **Required_reproducibility = f(Impact, Irreversibility, Regulatory_exposure, Auditability)**

The architectural principle: buy the level of reproducibility justified by the decision's consequence. Expensive exactness is not required everywhere.

7. EDA Complexity : The Cartesian Product of Uncertainties

Event-Driven Architecture brings benefits (asynchrony, resilience) and costs (order uncertainty, eventual consistency).

Probabilistic systems add a structural dimension:

> **Omega_system = Omega_distributed x Omega_epistemic**

where:

- Omega_distributed = space of possible event orders, consistency states
- Omega_epistemic = space of epistemic states (model versions, assertion confidences, policy states)

The Cartesian product is multiplicatively larger. The resulting decision cannot be derived from the event set alone; it depends on both the realized event order and the epistemic state at execution time.

The solution: assertion envelope plus versioned state snapshots at each agent. This enables temporal causality reconstruction - you can step through the sequence and understand why each agent produced its assertion.

8. Two Planes : Architecture vs. Runtime

We have now identified two distinct structures:

Governance Plane (Architecture of Authority)

Capability -> Decision Rights -> Authority

This is the architecture of who is permitted to decide what. It is static or slowly-changing.

Execution Plane (Runtime Sequence)

Event -> Execution -> Assertion -> Decision -> Effect

| | |

Technical Envelope | Epistemic Type | Policy Gate

This is what happens at runtime. Events flow in, assertions emerge (carrying epistemic status), policies transform assertions to decisions (checking authority), and effects flow out.

The critical insight: these are orthogonal structures that were previously conflated.

Governance answers "Who decides?" **Execution** answers "What happens?"

Assertions inhabit both:

- In Governance plane: "This assertion-type is assigned to this authority level"
- In Execution plane: "This computation produced an assertion with this epistemic status"

This separation resolves architectural ambiguity and prepares Article 3's governance framework.

9. Epistemic Contracts : The Integration Point

Epistemic contracts carry four dimensions:

EpistemicContract =

Assertion_value

+ Epistemic_type(origin, method, temporal)

+ Authority_boundary

+ Validity_conditions(domain, expiration, reversibility)

These contracts move between services. A downstream system sees not just a value ("Severity=HIGH") but its full epistemic context.

This transforms distributed systems from "services that exchange data" to "services that exchange assertions with traceable provenance and bounded authority."

This applies far beyond agents:

- Classical machine learning (model outputs with uncertainty)
- Digital twins (simulations with predicted status)
- Recommendation engines (scores with inferred status)
- Autonomous systems (actions with authority levels)
- Hybrid human-AI systems (mixing human assertions with machine inferences)

The common pattern: probabilistic computation produces assertions. Assertions must carry their epistemic status and authority boundaries. Contracts must make this explicit.

10. The Central Ontology

The five architectural objects are:

Capability = what must be doable (architectural primitive from Article 1)

Execution = how computation happens (technical modality)

Assertion = what computation produces (epistemic result)

Decision = what action is chosen (based on assertion + policy + authority)

Effect = what happens in the world (operational consequence)

Two transverse properties:

Epistemic_status = How we know (origin, method, temporal)

Authority = What we are permitted to do (bounded action rights)

This ontology is stable across all three articles.

11. Pre-positioning Article 3 : From Assertion to Authority

The progression of this series is:

Article 1 (Ontology): Agent is not Capability. Agents are implementation choices for capabilities. The distinction between architectural primitive and implementation modality is critical.

Article 2 (Epistemology): Assertion is not Fact. Assertions must carry epistemic status. Collapsing assertions into facts without epistemic justification is epistemic laundering.

Article 3 (Governance): Inference is not Authority. Producing an assertion does not confer the right to produce an effect. Governance must operate at four levels: Authority(Capability), Accountability(Decision), Observability(Execution), Liability(Effect).

The deeper pattern across all three articles: never let an implementation property usurp an architectural property.

Closing

Distributed systems with probabilistic components face a new architectural requirement: epistemic typing - the discipline of ensuring that every assertion carries its conditions of credibility and its authority boundaries.

This is not specific to agents. It applies wherever probabilistic computation produces values for consumption by other services.

The three-article series proposes a coherent architecture:

Capability (what must exist)

-> Decision Rights (who decides)

-> Authority (bounded action rights)

-> Execution (technical realization)

-> Assertion (epistemic result)

-> Decision (action selection)

-> Effect (world change)

> Governance (consequence management)

Where assertions never collapse implicitly into facts, and authority is never assumed from technical capability alone.

This is what the governance of probabilistic agents requires.