MANCHINE AI

TECHNOLOGY

HUMANS FIRST. CONTINUITY ALWAYS.

Runtime Integrity Control (RiCo)

TECHNICAL ARCHITECTURE SPECIFICATION

TAP-001

PART I — FOUNDATIONS

Version 1.0
July 2025

ARCHITECTURE FOR CONSEQUENTIAL EXECUTION

ManChine AI Technology LLC
manchine.ai

ARCHITECTURE MUST SURVIVE.

Part I — Foundations

1. Why Runtime Governance?

Core Thesis: Existing AI governance determines whether systems should be deployed. RiCo determines whether each consequential execution should still be permitted at the moment consequence occurs.

Introduction

Artificial Intelligence governance has advanced significantly in recent years. Organizations have developed policies, ethical principles, regulatory frameworks, risk assessments, and pre-deployment validation processes intended to ensure AI systems behave responsibly.

These mechanisms are essential, but they largely govern **what should be deployed**, not **what should be permitted to execute** under continuously changing runtime conditions.

The moment an AI system transitions from design and validation into live execution, it enters an environment that is no longer static. Permissions change, contextual conditions evolve, evidence becomes stale, policies are updated, authorities are revoked, and operational circumstances shift independently of the original approval.

Runtime governance exists because the conditions that justified an action before execution may no longer justify that same action at the moment consequence occurs.

The Runtime Problem

Traditional governance assumes that once an action has been validated, approved, or authorized, execution may proceed according to that earlier determination.

In practice, however, modern AI systems rarely execute immediately after approval.

Execution may be delayed by:

- asynchronous processing
- distributed orchestration
- workflow queues
- retries
- human intervention
- dependency resolution

- resource scheduling
- network latency
- recovery mechanisms

Between approval and execution, the operational environment continues to change.

The critical governance question therefore becomes:

Does the original approval remain admissible at the moment of execution?

This question cannot be answered solely by design-time governance.

It requires runtime evaluation.

Existing Governance Stops Before Consequence

Most existing governance frameworks concentrate on activities occurring before execution.

These commonly include:

- policy definition
- model evaluation
- fairness testing
- security assessment
- compliance validation
- deployment approval
- operational monitoring

These activities reduce deployment risk, but they do not continuously determine whether an individual consequential execution remains legitimate under present conditions.

Operational monitoring observes what systems do.

Runtime governance determines whether they should continue doing it.

This distinction represents the transition from **observing execution** to **governing execution**.

Runtime Introduces New Failure Surfaces

Execution introduces governance challenges that cannot be resolved solely through pre-execution approval.

Examples include:

- authority revoked after approval
- policy modifications during execution delay
- contextual changes affecting admissibility
- stale evidence supporting outdated decisions
- dependency failures altering operational assumptions
- environmental changes affecting safety or legality
- asynchronous execution occurring under obsolete conditions

In each case, the system may continue operating correctly from an engineering perspective while simultaneously violating the conditions that originally justified execution.

Operational continuity alone does not guarantee governance legitimacy.

Why RiCo Exists

Runtime Integrity Control (RiCo) was developed to address this architectural gap.

Rather than assuming that prior approval remains valid, RiCo treats admissibility as a runtime property that must remain demonstrably valid at the point of consequential execution.

RiCo continuously evaluates whether execution remains justified under current authority, evidence, context, and policy before consequence is allowed to proceed.

This shifts governance from a static approval model toward a continuously verifiable execution model.

The objective is not merely to preserve operational continuity, but to preserve the legitimacy of consequential execution as runtime conditions evolve.

Deployment approval authorizes a system to operate. Runtime governance determines whether each consequential execution remains admissible as operational conditions change.

---

2. Runtime Execution Boundary (REB)

Introduction

Every AI system transitions through three fundamental stages before producing a real-world consequence:

1. A decision is formed.
2. That decision is executed.
3. Execution produces a consequence.

These stages are often treated as a continuous process. Architecturally, however, they represent distinct governance surfaces with different responsibilities and failure modes.

The Runtime Execution Boundary (REB) defines the precise point at which a potential decision becomes an operational consequence.

It is at this boundary that governance must determine whether execution remains admissible under current runtime conditions.

Decision

A **decision** represents an intended course of action produced by a human, an AI system, or a coordinated workflow.

At this stage:

- authority may have been established,
- evidence evaluated,
- policies consulted,
- context assessed,
- and an intended action identified.

A decision expresses **intent**.

It does not yet produce consequence.

Because no operational consequence has occurred, decisions remain subject to revision, withdrawal, or invalidation without affecting the external environment.

Execution

Execution is the transition from intended action to operational behavior.

Execution consumes computational resources, invokes services, triggers workflows, modifies state, or interacts with external systems.

Execution transforms intent into activity.

However, execution alone is not the architectural concern.

The critical governance question is whether execution remains **admissible at the moment it begins**, rather than whether it was admissible when originally approved.

Between decision and execution:

- authority may change,
- policies may be updated,
- evidence may become stale,
- context may evolve,
- dependencies may fail,
- environmental conditions may shift.

Execution therefore requires its own governance evaluation independent of earlier approval.

Consequence

A **consequence** is the observable effect produced by execution.

Consequences may include:

- modifying persistent state,
- approving or denying access,
- initiating financial transactions,
- controlling physical systems,
- issuing commands,
- communicating decisions,
- or producing legally or operationally significant outcomes.

Unlike decisions, consequences cannot always be reversed.

They become part of the operational history of the system.

Consequences therefore require the highest level of governance assurance.

The Runtime Execution Boundary

The Runtime Execution Boundary (REB) is the architectural control point separating intended execution from consequential execution.

Before crossing the REB, a system may continue evaluating:

- authority,
- evidence,
- context,
- policy,
- and admissibility.

After crossing the REB, execution begins producing consequences that may no longer be recoverable through governance alone.

The REB therefore represents the final opportunity to verify that execution remains justified under **current runtime conditions** rather than inherited historical assumptions.

Architectural Role of the REB

The REB is not a software component.

It is an architectural boundary.

Its purpose is to ensure that every consequential execution passes through a single governance checkpoint immediately before consequence becomes possible.

Crossing the REB requires that runtime admissibility has been successfully established using current authority, evidence, context, and policy.

If admissibility cannot be demonstrated, execution must not proceed.

Why the REB Matters

Many existing systems validate actions before deployment or before initial approval.

However, modern AI systems frequently execute after delays introduced by queues, retries, distributed orchestration, asynchronous processing, or adaptive workflows.

Under these conditions, the assumptions supporting the original decision may no longer hold.

The REB prevents historical admissibility from silently becoming operational consequence.

Instead, it requires that legitimacy be re-established immediately before execution crosses into consequence.

Formal Definition

The Runtime Execution Boundary (REB) is the architectural boundary immediately preceding consequential execution where current runtime admissibility must be established before operational consequence is permitted to occur.

Figure 2.1 presents the complete Runtime Execution Boundary sequence:

Decision

|

▼

Authority

Evidence

Context

Policy

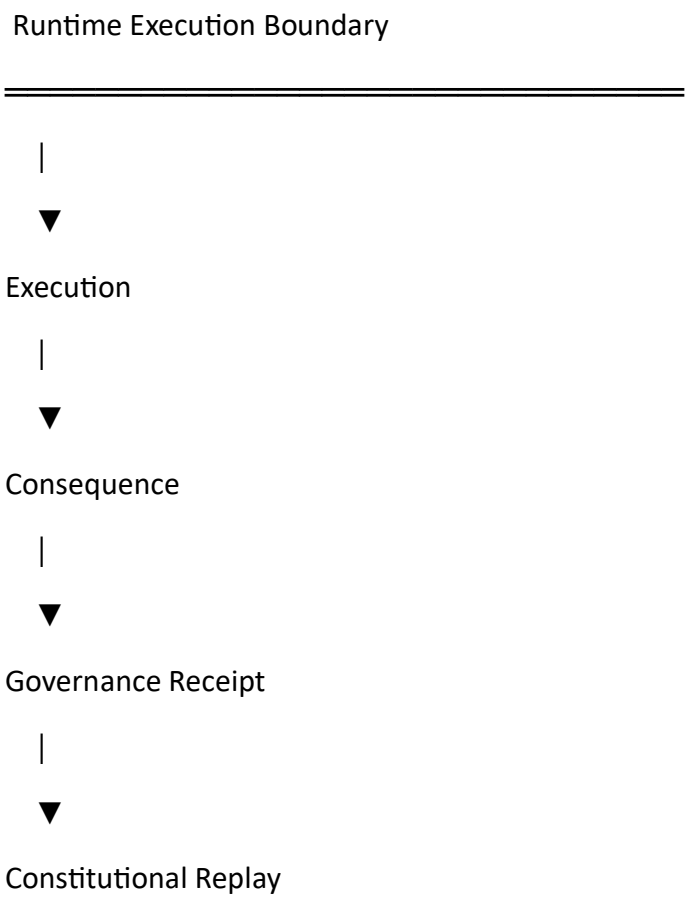
|

▼

Admissibility

|

═══════════════════════════════════════


Runtime Execution Boundary
Execution
Consequence
Governance Receipt
Constitutional Replay

3. Runtime Integrity Control (RiCo)

Introduction

Runtime Integrity Control (RiCo) is the runtime governance engine responsible for determining whether consequential execution remains admissible under current operational conditions.

RiCo does not replace application logic, business workflows, or execution engines. Instead, it operates as an independent governance layer positioned immediately before the Runtime Execution Boundary (REB), evaluating whether execution remains justified based on current authority, evidence, context, and policy.

Its primary function is to ensure that operational continuity does not silently become consequential execution without first re-establishing runtime legitimacy.

Architectural Position

RiCo exists between the system's operational workflow and consequential execution.

Rather than assuming that previous approvals remain valid, RiCo performs a runtime governance evaluation immediately before execution crosses the Runtime Execution Boundary.

Every consequential action follows the same governance path.

Inputs

|

▼

Runtime Evaluation

|

▼

Authority

|

▼

Evidence

|

▼

Context

|

▼

Policy

|

▼

Admissibility

|

▼

Runtime Execution Boundary (REB)

|

▼

Execution

RiCo therefore serves as the final governance checkpoint before consequence becomes operational.

Runtime Inputs

Every evaluation begins with the current operational state.

Typical inputs include:

- execution request
- system identity
- current authority state
- available evidence
- runtime context

- applicable policies
- environmental conditions
- dependency status
- temporal information

RiCo evaluates the current state rather than relying solely on historical approval.

Runtime Evaluation

The evaluation process determines whether execution remains legitimate under present runtime conditions.

Rather than asking:

"Was this action previously approved?"

RiCo asks:

"Can this action still be justified right now?"

Evaluation is therefore performed against current operational reality rather than inherited execution history.

Governance Components

Authority

Authority determines whether the requesting entity currently possesses the right to perform the intended action.

Authority may change over time through delegation, revocation, expiration, or organizational change.

RiCo evaluates authority at runtime.

Evidence

Evidence provides the factual basis supporting execution.

Evidence must be:

- available
- sufficient
- trustworthy
- current

Historical evidence alone is insufficient if runtime conditions have materially changed.

Context

Context represents the operational environment surrounding execution.

Context may include:

- current system state
- environmental conditions
- dependency health
- execution location
- orchestration state
- external constraints

Context is evaluated as it exists at execution time.

Policy

Policies define the governance rules controlling execution.

Policies may include:

- organizational policy
- regulatory requirements
- contractual obligations
- security controls
- operational constraints
- mission-specific rules

RiCo evaluates execution against the policies currently in force.

Admissibility

Admissibility is the final governance determination produced by RiCo.

It answers a single architectural question:

Should this execution be permitted under current runtime conditions?

Admissibility is not inherited from previous approvals.

It is established through the combined evaluation of authority, evidence, context, and policy immediately before execution.

Runtime Execution Boundary

Only after admissibility has been successfully established may execution cross the Runtime Execution Boundary.

If admissibility cannot be demonstrated, execution is denied before consequence occurs.

The Runtime Execution Boundary therefore separates governance evaluation from consequential execution.

Execution

Execution begins only after successful governance evaluation.

RiCo does not execute business logic itself.

Its responsibility is to determine whether execution may proceed.

Execution remains the responsibility of the underlying application, orchestration platform, or operational system.

Architectural Characteristics

RiCo is designed to be:

- Runtime-driven rather than deployment-driven.

- Independent of application-specific business logic.
- Non-bypassable for consequential execution.
- Deterministic in governance evaluation.
- Replayable through governance evidence.
- Inspectable through explicit decision artifacts.

These characteristics allow governance decisions to be independently reviewed without reconstructing hidden operational assumptions.

Runtime Integrity Control (RiCo) is the runtime governance engine that evaluates authority, evidence, context, and policy to determine admissibility immediately before consequential execution crosses the Runtime Execution Boundary.

Normative Language

The key words MUST, MUST NOT, SHALL, SHALL NOT, SHOULD, and MAY indicate the strength of requirements within this specification.

MUST / SHALL — Mandatory for RiCo conformance.

MUST NOT / SHALL NOT — Prohibited for RiCo conformance.

SHOULD — Recommended unless an implementation documents a justified exception and demonstrates that the exception does not weaken a mandatory architectural property.

MAY — Optional and implementation-dependent.

Architecture Overview

Runtime Integrity Control at a Glance

Runtime Integrity Control (RiCo) governs the transition from an execution request to a consequential outcome. Current Authority, Evidence, Context, and Policy converge into an Admissibility determination immediately before the Runtime Execution Boundary. Only a PASS determination may cross the boundary. PASS and DENY determinations both produce Governance Receipts capable of supporting Constitutional Replay.

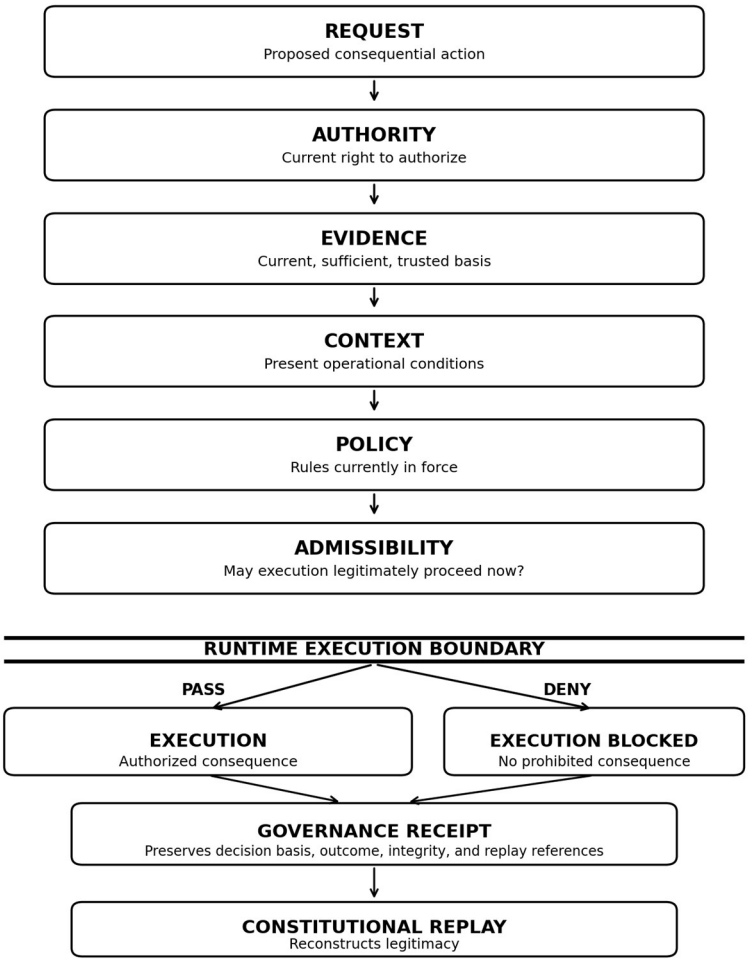


Figure A.1 — Canonical Runtime Integrity Control Architecture

RiCo evaluates present governance conditions, enforces admissibility before consequence, preserves the determination as evidence, and enables independent reconstruction of legitimacy.

Transition to Part II — Architecture

With the Runtime Execution Boundary established and RiCo positioned as the runtime governance engine, the remainder of this document examines each architectural component individually.

Each component is analyzed using the same three questions:

1. **What does this component do?**
2. **Why is it required?**
3. **What governance failure occurs if it is absent?**

This structure moves the discussion from conceptual architecture to implementation-ready design, allowing each component to be evaluated independently while preserving its role within the overall runtime governance model.

Part II — Architecture

4. Runtime Decision Flow

Introduction

Every consequential action governed by RiCo follows a defined runtime decision flow. This flow ensures that execution is evaluated against current operational conditions before consequence is permitted.

The Runtime Decision Flow is deterministic. Every consequential execution traverses the same governance sequence regardless of the application, orchestration platform, or execution environment.

Rather than relying on historical approval, the flow continuously evaluates whether execution remains admissible under present runtime conditions.

Architectural Objective

The Runtime Decision Flow exists to answer a single question:

Can this action still be executed legitimately under the current runtime state?

Only after this question is answered affirmatively may execution proceed across the Runtime Execution Boundary (REB).

Runtime Decision Sequence

1. Execution Request Received

2. Runtime Context Acquisition

3. Authority Validation

4. Evidence Evaluation

|

▼

5. Context Evaluation

|

▼

6. Policy Evaluation

|

▼

7. Admissibility Determination

|

▼

PASS / DENY

Editorial clarification: PASS and DENY are runtime admissibility outcomes. PASS/FAIL terminology is reserved for verification and conformance testing in Part IV.

| |

| └──────► Governance Receipt (DENY)

|

▼

8. Runtime Execution Boundary (REB)

|

▼

9. Consequential Execution

|

▼

10. Governance Receipt (PASS)

|

▼

11. Constitutional Replay Record

Step 1 — Execution Request

An execution request represents the intent to perform a consequential action.

The request may originate from:

- a user,
- an AI agent,
- a workflow engine,
- an asynchronous worker,
- a scheduled process,
- or another authorized system component.

At this stage, no consequence has occurred.

Step 2 — Runtime Context Acquisition

RiCo gathers the current runtime state required for governance evaluation.

Typical inputs include:

- identity
- timestamps
- environmental conditions
- dependency status
- operational state
- execution surface
- current policy references

The evaluation is performed against the present system state rather than historical snapshots.

Step 3 — Authority Validation

RiCo verifies that the requesting entity currently possesses authority to perform the intended action.

Authority validation confirms that permissions remain valid and have not been revoked, delegated, expired, or otherwise changed since the original request.

Failure immediately terminates the flow.

Step 4 — Evidence Evaluation

RiCo evaluates the evidence supporting execution.

Evidence must remain:

- available,
- trustworthy,
- sufficient,
- and current.

If the required evidence no longer supports execution, the request is denied.

Step 5 — Context Evaluation

Runtime conditions are evaluated to determine whether the operational environment still supports execution.

Context evaluation considers:

- current system state,
- dependency health,
- environmental conditions,
- execution timing,
- orchestration state,
- and other runtime factors that may affect legitimacy.

Step 6 — Policy Evaluation

RiCo evaluates all applicable governance policies.

These may include:

- organizational policies,
- regulatory obligations,
- contractual constraints,
- operational rules,
- and security requirements.

Policy evaluation reflects the policies currently in force at execution time.

Step 7 — Admissibility Determination

The outputs of authority, evidence, context, and policy evaluation are combined into a single runtime governance decision.

RiCo determines whether execution is currently admissible.

Two outcomes are possible:

PASS

Execution remains justified.

or

DENY

Execution is blocked before consequence occurs.

Step 8 — Runtime Execution Boundary

Only successful evaluations may cross the Runtime Execution Boundary.

The REB separates governance determination from consequential execution.

No consequential execution should bypass this boundary.

Step 9 — Consequential Execution

The governed action is executed.

Execution is now permitted because admissibility has been established immediately before consequence.

RiCo does not perform the execution itself.

Execution remains the responsibility of the operational system.

Step 10 — Governance Receipt

RiCo produces a governance receipt documenting the runtime evaluation.

The receipt records:

- evaluation outcome,
- authority state,
- evidence references,
- contextual conditions,
- applied policies,
- admissibility determination,
- execution identifier,
- timestamp.

This creates an independently inspectable governance artifact.

Step 11 — Constitutional Replay Record

The governance receipt becomes part of the Constitutional Replay record.

Replay enables reviewers to reconstruct the governance decision using explicit evidence rather than inferred operational history.

Replay supports verification, auditing, challenge, and architectural inspection.

Failure Handling

The Runtime Decision Flow terminates immediately whenever admissibility cannot be established.

Execution is not delayed awaiting consequence.

Execution is prevented before consequence becomes operational.

Every denied execution produces a governance receipt documenting the reason for denial.

Architectural Properties

The Runtime Decision Flow is designed to be:

- deterministic,
- replayable,
- independently inspectable,
- non-bypassable,
- implementation-neutral,
- runtime-local,
- and architecturally verifiable.

These properties ensure that governance decisions remain challengeable without requiring reconstruction of hidden orchestration state.

Summary

The Runtime Decision Flow defines the complete governance lifecycle from execution request to consequential execution, ensuring that every action crossing the Runtime Execution Boundary has been evaluated against current authority, evidence, context, and policy before consequence is permitted.

---

5. Authority

Introduction

Authority is the governance component responsible for determining whether an entity currently possesses the legitimate right to initiate a consequential action.

Within RiCo, authority is evaluated as a runtime property rather than a static assignment. Possessing authority at one point in time does not guarantee that authority remains valid when execution occurs.

Authority must therefore be established, maintained, and verified immediately before consequential execution crosses the Runtime Execution Boundary (REB).

Architectural Purpose

Authority answers the question:

"Who has the legitimate right to perform this action at this moment?"

This determination extends beyond identity verification. It considers whether the requesting entity continues to possess the legal, organizational, operational, or delegated authority required for execution.

Authority establishes legitimacy.

It does not, by itself, establish admissibility.

Establishing Authority

Authority may originate from multiple governance sources depending on the operational environment.

Examples include:

- Organizational roles and responsibilities
- Delegated authority from another authorized entity
- Regulatory or statutory authority
- Contractual authority
- Mission-specific authority

- Emergency authority under predefined conditions
- System-assigned operational authority

Regardless of its source, authority must be explicit, identifiable, and independently verifiable.

RiCo does not infer authority from execution capability.

The ability to execute does not constitute the authority to execute.

Runtime Validation

Authority is evaluated immediately before execution.

RiCo confirms that authority remains valid under the current runtime state.

Runtime validation includes verification that:

- the authority still exists,
- the authority applies to the requested action,
- the authority has not expired,
- the authority has not been revoked,
- the authority remains within its defined scope.

Only current authority is considered valid for consequential execution.

Authority Expiration

Authority is not necessarily permanent.

Authority may expire due to:

- elapsed time,
- completion of an assigned task,
- organizational restructuring,
- contractual termination,
- mission completion,
- regulatory changes,

- predefined temporal limits.

Expired authority cannot support admissibility.

Historical authority does not authorize future execution.

Authority Inheritance

Some operational workflows require authority to be inherited across coordinated processes.

Inheritance allows a downstream process to operate under authority established by an upstream decision, provided that:

- inheritance is explicitly permitted,
- the inherited authority remains within scope,
- runtime conditions remain valid,
- and the inheritance chain is fully traceable.

Inherited authority must remain independently verifiable throughout the execution lifecycle.

Implicit inheritance is not sufficient.

Authority Delegation

Authority may be delegated from one authorized entity to another.

Delegation must preserve governance integrity by recording:

- the delegating authority,
- the receiving authority,
- the scope of delegation,
- any operational constraints,
- temporal limitations,
- and conditions governing delegation.

Delegated authority never exceeds the authority possessed by the delegating entity.

RiCo evaluates delegated authority using the same runtime criteria as directly assigned authority.

Authority Revocation

Authority may be withdrawn before execution occurs.

Revocation may result from:

- organizational decisions,
- policy updates,
- security events,
- operational changes,
- regulatory intervention,
- emergency suspension,
- or explicit administrative action.

Once revoked, authority immediately becomes inadmissible for future consequential execution.

Historical approval cannot override current revocation.

Failure Modes

If authority cannot be established or maintained, RiCo terminates the governance evaluation before the Runtime Execution Boundary.

Typical failure conditions include:

- missing authority,
- expired authority,
- revoked authority,
- invalid delegation,
- ambiguous ownership,
- inheritance outside permitted scope,
- authority exceeding assigned limits.

In each case, execution is denied before consequence occurs.

Architectural Characteristics

Authority within RiCo is:

- Explicit rather than inferred.
- Runtime-validated rather than historically assumed.
- Independently verifiable.
- Traceable across delegation and inheritance.
- Subject to expiration and revocation.
- Evaluated independently of application logic.

These characteristics ensure that governance legitimacy reflects the current operational state rather than historical assumptions.

Relationship to Admissibility

Authority is a necessary condition for admissibility but is not sufficient on its own.

A request possessing valid authority may still fail governance evaluation if:

- supporting evidence is insufficient,
- runtime context has materially changed,
- applicable policies prohibit execution,
- or other admissibility conditions are not satisfied.

Authority therefore establishes the right to request execution, while admissibility determines whether execution may proceed.

Summary

Authority is the runtime-governed determination that an entity currently possesses the legitimate right to initiate a consequential action. Within RiCo, authority is continuously evaluated for validity, scope, delegation, inheritance, expiration, and revocation before execution is permitted to cross the Runtime Execution Boundary.

6. Evidence

Introduction

Evidence is the governance component responsible for establishing the factual basis supporting consequential execution.

Within RiCo, evidence is evaluated as a runtime resource rather than a historical artifact. Evidence that was sufficient at the time of an earlier decision may no longer justify execution when operational conditions have changed.

Consequently, RiCo evaluates evidence based on its current validity, relevance, trustworthiness, and sufficiency immediately before execution crosses the Runtime Execution Boundary (REB).

Evidence supports legitimacy.

Without evidence, authority becomes assertion rather than justification.

Architectural Purpose

Evidence answers the question:

"What facts currently justify this action?"

Evidence provides the objective basis upon which governance decisions are made.

It allows every consequential action to be supported by verifiable information rather than assumption, inference, or historical approval alone.

RiCo treats evidence as an active runtime input rather than a passive record of past events.

Evidence Lifecycle

Evidence exists within a lifecycle that extends from creation through retirement.

A typical evidence lifecycle consists of:

1. Evidence Creation
2. Evidence Collection
3. Evidence Verification
4. Runtime Evaluation

5. Governance Use
6. Evidence Expiration
7. Archival or Retirement

Throughout this lifecycle, evidence may gain, lose, or entirely forfeit its governance value.

RiCo therefore evaluates evidence in its current lifecycle state rather than assuming continued validity.

Evidence Freshness

Evidence must remain current enough to support the requested execution.

Freshness reflects the temporal relationship between the evidence and the consequential action it supports.

Evidence may become stale because of:

- elapsed time,
- environmental changes,
- operational state changes,
- updated observations,
- revoked source information,
- new conflicting evidence.

Freshness requirements vary depending upon operational context.

An observation that remains valid for years in one domain may remain valid for only seconds in another.

RiCo therefore evaluates freshness relative to the governing policy and operational context.

Evidence Trust

Evidence must originate from sources that are sufficiently trustworthy for the intended consequence.

Trust considers factors such as:

- source authenticity,
- integrity,
- provenance,
- collection method,
- chain of custody,
- verification status,
- resistance to tampering.

Evidence obtained from an untrusted or unverifiable source cannot reliably support consequential execution regardless of its apparent content.

Trust is evaluated independently from freshness.

Recent evidence is not necessarily trustworthy.

Trusted evidence is not necessarily current.

Evidence Sufficiency

Not all available evidence is sufficient to justify execution.

Sufficiency evaluates whether the available evidence adequately supports the requested consequential action.

Evidence may fail sufficiency because it is:

- incomplete,
- contradictory,
- ambiguous,
- inconclusive,
- below required confidence thresholds,
- missing critical supporting observations.

RiCo evaluates evidence as a complete governance set rather than relying upon isolated observations.

Execution proceeds only when the available evidence satisfies the sufficiency requirements defined by applicable governance policy.

Runtime Validation

Immediately before execution, RiCo evaluates whether the available evidence remains:

- relevant,
- current,
- trustworthy,
- sufficient,
- applicable to the requested action.

If any of these conditions cannot be demonstrated, evidence no longer supports admissibility.

Execution is denied before consequence occurs.

Failure Modes

Evidence-related governance failures include:

- stale evidence,
- missing evidence,
- conflicting evidence,
- insufficient evidence,
- unverifiable evidence,
- compromised evidence,
- evidence outside its validity period,
- evidence unrelated to the requested action.

Each failure weakens the factual basis required for legitimate execution.

RiCo therefore treats evidence failure as a governance failure rather than merely a data quality issue.

Architectural Characteristics

Evidence within RiCo is:

- Runtime-evaluated rather than historically assumed.
- Independently verifiable.
- Traceable to its originating source.
- Time-sensitive where required.
- Governed throughout its lifecycle.
- Subject to trust and sufficiency evaluation.
- Preserved through governance receipts for replay.

These characteristics ensure that consequential execution remains grounded in demonstrable facts rather than inherited assumptions.

Relationship to Admissibility

Evidence is one of the four primary governance components contributing to admissibility.

Even when authority is valid, execution may remain inadmissible if supporting evidence is:

- stale,
- incomplete,
- untrusted,
- or insufficient.

Evidence therefore provides the factual foundation upon which legitimate runtime governance depends.

Summary

Evidence is the runtime-governed collection of verifiable facts supporting consequential execution. Within RiCo, evidence is continuously evaluated for lifecycle state, freshness, trustworthiness, and sufficiency to ensure that every consequential action remains justified under current operational conditions.

---

7. Context

Introduction

Context is the governance component responsible for evaluating the operational conditions surrounding a consequential execution.

Within RiCo, context represents the current state of the environment in which execution is proposed. While authority establishes who may act and evidence establishes why an action is justified, context determines whether the surrounding operational conditions continue to support legitimate execution.

Because runtime environments evolve continuously, context is evaluated immediately before execution crosses the Runtime Execution Boundary (REB).

Context preserves situational legitimacy.

Architectural Purpose

Context answers the question:

"Under the current operational conditions, should this action still occur?"

A decision that was legitimate under one set of circumstances may become inappropriate or unsafe when those circumstances change.

RiCo therefore evaluates the operational environment at execution time rather than relying solely on conditions that existed when the original decision was made.

Runtime Context

Runtime context represents the collection of operational conditions existing immediately before consequential execution.

Context may include:

- current system state,
- active operational mode,
- workload conditions,

- geographic location,
- execution timing,
- organizational state,
- network availability,
- external service status,
- mission phase,
- environmental constraints.

RiCo evaluates the runtime context as it exists at the moment of execution.

Historical context is preserved for replay but is not assumed to represent the current operational state.

Environmental Conditions

Execution occurs within an environment that may change independently of the original decision.

Environmental conditions may include:

- physical conditions,
- infrastructure availability,
- network health,
- security posture,
- regulatory environment,
- emergency conditions,
- operational readiness.

Changes in these conditions may invalidate assumptions supporting the original decision.

RiCo therefore considers environmental conditions as part of every runtime evaluation.

Dependencies

Most consequential actions depend upon other systems, services, or resources.

Dependencies may include:

- external APIs,
- databases,
- identity services,
- communication networks,
- AI models,
- sensor inputs,
- workflow engines,
- third-party platforms.

A dependency failure may alter the legitimacy of execution even when authority and evidence remain valid.

RiCo evaluates dependency health as part of the overall runtime context.

Identity

Context includes the identity of every entity participating in the execution process.

Identity extends beyond the initiating user and may include:

- requesting user,
- autonomous agent,
- delegated service,
- orchestration platform,
- downstream execution component,
- external system.

Identity establishes accountability throughout the execution path.

RiCo requires that participating identities remain attributable and traceable before consequence is permitted.

Execution Surface

The execution surface defines the operational environment in which the consequential action will occur.

Examples include:

- cloud infrastructure,
- on-premises systems,
- edge devices,
- robotic platforms,
- healthcare systems,
- financial transaction platforms,
- industrial control environments.

Different execution surfaces introduce different governance risks, operational constraints, and policy requirements.

RiCo evaluates the execution surface as part of determining runtime admissibility.

Runtime Validation

Immediately before execution, RiCo verifies that the current context remains compatible with legitimate execution.

This evaluation confirms that:

- operational conditions remain acceptable,
- dependencies are functioning as required,
- participating identities remain valid,
- the execution surface remains appropriate,
- environmental conditions have not materially changed.

If contextual assumptions have become invalid, execution is denied before consequence occurs.

Failure Modes

Context-related governance failures include:

- degraded operational conditions,
- unavailable dependencies,
- identity inconsistencies,
- execution on an unauthorized platform,
- environmental changes affecting safety or legality,
- missing runtime information,
- incompatible execution environments.

These failures may render an otherwise valid decision inadmissible.

RiCo therefore treats contextual failures as governance failures rather than operational inconveniences.

Architectural Characteristics

Context within RiCo is:

- Runtime-specific rather than historically inferred.
- Continuously evaluated.
- Environment-aware.
- Dependency-aware.
- Identity-aware.
- Execution-surface aware.
- Independently verifiable through governance receipts.

These characteristics ensure that consequential execution reflects the operational reality existing at the moment consequence occurs.

Relationship to Admissibility

Context is one of the four primary governance components contributing to admissibility.

Even when authority, evidence, and policy remain valid, execution may still become inadmissible if the surrounding operational context has materially changed.

Context therefore provides the situational foundation upon which legitimate runtime governance depends.

Summary

Context is the runtime-governed evaluation of the operational environment surrounding consequential execution. Within RiCo, context continuously assesses environmental conditions, dependencies, participating identities, and the execution surface to ensure that every consequential action remains legitimate under current runtime conditions.

---

8. Policy

Introduction

Policy is the governance component responsible for defining the rules, constraints, and obligations that determine whether consequential execution is permitted.

Within RiCo, policies establish the normative framework against which runtime decisions are evaluated. They define not only what actions are allowed, but also the conditions under which those actions remain legitimate.

Policies are evaluated immediately before execution crosses the Runtime Execution Boundary (REB), ensuring that consequential actions comply with the governance requirements currently in force.

Policy defines permissible behavior.

RiCo determines whether execution remains consistent with that behavior.

Architectural Purpose

Policy answers the question:

"Under the governing rules currently in force, should this action be permitted?"

Unlike application logic, which determines *how* a system operates, policy determines *whether* a particular action remains acceptable under organizational, legal, operational, and mission-specific requirements.

Policy therefore provides the normative framework for runtime admissibility.

Static Policy

Static policies define governance rules that remain relatively stable over time.

Examples include:

- organizational standards,
- constitutional governance principles,
- permanent security requirements,
- contractual obligations,

- established operational procedures.

Static policies provide consistent governance expectations across normal system operation.

Although relatively stable, they remain subject to formal revision and version control.

Dynamic Policy

Dynamic policies adapt to changing operational conditions.

Examples include:

- temporary operational restrictions,
- changing risk levels,
- evolving threat conditions,
- resource constraints,
- adaptive governance responses,
- temporary administrative directives.

Dynamic policies allow governance to respond to changing runtime conditions without requiring changes to application logic.

RiCo evaluates the current policy state rather than assuming previously evaluated policies remain applicable.

Organizational Policy

Organizations establish governance rules reflecting their operational responsibilities and acceptable risk.

Organizational policies may include:

- approval requirements,
- segregation of duties,
- operational workflows,
- internal compliance standards,
- security governance,

- accountability structures.

These policies define how the organization expects consequential decisions to be governed.

RiCo incorporates these policies into every runtime evaluation.

Regulatory Policy

Many operational environments are governed by external legal or regulatory obligations.

Examples include:

- privacy legislation,
- healthcare regulations,
- financial regulations,
- public-sector governance requirements,
- industry standards,
- jurisdiction-specific legal obligations.

Regulatory policies may change independently of system deployment.

RiCo therefore evaluates regulatory requirements as they exist at execution time.

Mission Policy

Mission policies govern behavior specific to an operational objective or mission.

Mission policies may define:

- operational priorities,
- mission-specific constraints,
- authorized objectives,
- acceptable risk thresholds,
- mission phases,
- resource allocation priorities.

Mission policies frequently evolve during execution and therefore require continuous runtime evaluation.

Emergency Policy

Certain operational circumstances require temporary governance adjustments during emergencies.

Emergency policies may:

- authorize exceptional actions,
- suspend selected operational procedures,
- introduce additional approval requirements,
- restrict execution under elevated risk,
- activate contingency governance mechanisms.

Emergency authority does not eliminate governance.

Instead, it replaces normal governance with explicitly defined emergency governance.

Emergency policies remain subject to runtime evaluation and governance receipts.

Runtime Validation

Immediately before execution, RiCo evaluates the complete policy environment currently governing the requested action.

This evaluation confirms:

- applicable policies have been identified,
- policy conflicts have been resolved,
- policy versions remain current,
- temporary directives remain active,
- emergency conditions have been correctly applied.

Only policies in force at execution time contribute to admissibility.

Failure Modes

Policy-related governance failures include:

- missing governing policy,
- conflicting policy requirements,
- obsolete policy references,
- unauthorized policy overrides,
- expired emergency directives,
- unresolved regulatory conflicts,
- execution outside policy scope.

Each failure prevents RiCo from demonstrating that execution remains consistent with the governing framework.

Execution is therefore denied before consequence occurs.

Architectural Characteristics

Policy within RiCo is:

- Runtime-evaluated rather than statically assumed.
- Independent of application logic.
- Version-controlled and traceable.
- Adaptable to changing operational conditions.
- Composable across multiple governance domains.
- Recorded within governance receipts for replay.

These characteristics ensure that consequential execution reflects the governance requirements actually in force at the moment of execution.

Relationship to Admissibility

Policy is one of the four primary governance components contributing to admissibility.

Even when authority, evidence, and context remain valid, execution becomes inadmissible if it violates the governing policy framework.

Policy therefore provides the normative constraints that define legitimate runtime behavior.

Summary

Policy is the runtime-governed framework of rules, constraints, and obligations that determine whether consequential execution remains permissible under current operational conditions. Within RiCo, policies are continuously evaluated across static, dynamic, organizational, regulatory, mission, and emergency domains before execution is permitted to cross the Runtime Execution Boundary.

---

9. Admissibility

Introduction

Admissibility is the runtime governance determination that establishes whether a consequential action may legitimately proceed under current operational conditions.

Within RiCo, admissibility is not a static approval, inherited permission, or historical authorization. It is a runtime determination produced by evaluating the current state of authority, evidence, context, and policy immediately before execution crosses the Runtime Execution Boundary (REB).

Admissibility is the architectural convergence point of the RiCo governance model.

It represents the final governance decision before consequence becomes possible.

Architectural Purpose

Admissibility answers a single runtime question:

"Can this action still be legitimately executed right now?"

This question is intentionally different from:

- Was the action previously approved?
- Did the user once possess authority?
- Was sufficient evidence available earlier?

Instead, RiCo evaluates whether all required governance conditions continue to exist at the precise moment consequence is about to occur.

Only then may execution proceed.

Governance Convergence

Admissibility is not evaluated independently.

It is derived from the combined runtime evaluation of four governance components:

- **Authority** — Does the requesting entity currently possess legitimate authority?
- **Evidence** — Does sufficient, trustworthy, and current evidence support execution?

- **Context** — Do present operational conditions continue to justify execution?
- **Policy** — Do the governing rules currently permit the requested action?

Failure of any single component prevents admissibility from being established.

Admissibility therefore represents the integrated governance state rather than the status of any individual component.

Re-Proof

RiCo does not assume that historical justification remains valid.

Instead, it performs **runtime re-proof**.

Re-proof is the process of re-establishing governance legitimacy immediately before consequential execution.

Rather than relying upon historical approvals, RiCo reconstructs the current governance state using present authority, evidence, context, and policy.

Every consequential execution is therefore supported by a current governance determination rather than an inherited assumption.

Re-proof transforms governance from a one-time approval into a continuously verifiable process.

Temporal Validity

Governance decisions exist within time.

A determination that was legitimate at one moment may become invalid as operational conditions evolve.

Temporal validity recognizes that:

- authority may expire,
- evidence may become stale,
- context may change,
- policies may be updated.

Consequently, governance legitimacy cannot be separated from time.

RiCo evaluates whether the governance state remains valid at the exact moment execution is proposed.

Admissibility therefore reflects **current validity**, not historical correctness.

Current-State Validation

Current-state validation evaluates the operational environment as it exists immediately before execution.

RiCo validates:

- current authority,
- current evidence,
- current context,
- current policy.

The objective is not to confirm that execution was previously justified.

The objective is to demonstrate that execution remains justified now.

Current-state validation ensures that governance reflects operational reality rather than historical assumptions.

Historical Invalidation

Historical approval does not guarantee future admissibility.

RiCo recognizes that previously valid governance decisions may become invalid before execution occurs.

Historical invalidation may result from:

- revoked authority,
- expired permissions,
- stale evidence,
- changing operational context,
- policy revision,

- emergency governance activation,
- regulatory change,
- dependency failure.

When historical assumptions no longer represent the current operational state, RiCo rejects inherited legitimacy.

Execution must instead be justified through current runtime evaluation.

Runtime Decision

After evaluating authority, evidence, context, and policy, RiCo produces one of two outcomes.

Admissible

The current governance state supports consequential execution.

Execution may cross the Runtime Execution Boundary.

A governance receipt is generated documenting the decision.

Inadmissible

One or more governance conditions cannot be demonstrated.

Execution is denied before consequence occurs.

A governance receipt records the basis for denial.

Architectural Characteristics

Admissibility within RiCo is:

- Runtime-derived rather than historically inherited.
- Deterministic.
- Independently verifiable.
- Time-sensitive.
- Non-transferable across changing runtime conditions.
- Fully replayable through governance receipts.

- Established immediately before consequential execution.

These characteristics ensure that legitimacy reflects the present operational state rather than assumptions carried forward from earlier stages of the workflow.

Relationship to the Runtime Execution Boundary

Admissibility is the final governance determination before execution crosses the Runtime Execution Boundary.

The REB does not determine legitimacy.

RiCo establishes admissibility before the boundary is crossed.

Only after admissibility has been successfully demonstrated may consequence become operational.

In this way, the Runtime Execution Boundary separates governance evaluation from consequential execution, while admissibility provides the governance decision that authorizes crossing that boundary.

Summary

Admissibility is the runtime-governed determination that a consequential action remains legitimate under current operational conditions. Within RiCo, admissibility is established through the integrated evaluation of authority, evidence, context, and policy immediately before execution crosses the Runtime Execution Boundary, ensuring that every consequence is supported by a current and demonstrable governance state.

Part III — Enforcement

Introduction

Parts I and II established the architectural foundations of RiCo.

The preceding chapters defined the Runtime Execution Boundary (REB), the Runtime Integrity Control (RiCo) governance engine, and the four governance components—Authority, Evidence, Context, and Policy—that collectively determine runtime admissibility.

At the conclusion of Part II, RiCo has completed its governance evaluation and produced a single determination:

Is this consequential execution admissible under current runtime conditions?

Part III begins where governance decisions become operational reality.

This is the point at which execution approaches the Runtime Execution Boundary.

Crossing the Runtime Execution Boundary

The Runtime Execution Boundary is the final architectural control point separating governance evaluation from consequential execution.

Everything preceding the REB concerns legitimacy.

Everything following the REB concerns consequence.

This distinction is fundamental to the RiCo architecture.

Governance is meaningful only if it can influence execution before consequence occurs.

Once execution crosses the Runtime Execution Boundary, the opportunity to prevent an inadmissible consequence has passed.

Consequently, enforcement mechanisms must operate immediately before and during this transition.

From Evaluation to Enforcement

The purpose of enforcement is not to repeat governance evaluation.

Evaluation determines whether execution is admissible.

Enforcement ensures that this determination is respected.

RiCo therefore distinguishes between:

- **Governance Evaluation** — establishing legitimacy.
- **Governance Enforcement** — ensuring that only legitimate execution proceeds.

This separation prevents governance decisions from becoming advisory rather than authoritative.

Architectural Objective

The objective of enforcement is to preserve the integrity of runtime governance by ensuring that:

- inadmissible execution cannot bypass governance,
- governance decisions remain binding,
- consequential actions remain traceable,
- runtime legitimacy persists throughout execution,
- every consequence remains independently verifiable.

Enforcement transforms governance from a recommendation into an operational control mechanism.

Scope of Part III

The following chapters examine the mechanisms that preserve governance integrity after admissibility has been established.

These include:

- **Consequence Path Coverage** — ensuring every consequential path remains governed.
- **Non-Bypassability** — preventing execution from circumventing RiCo.
- **Temporal Validity** — preserving legitimacy as execution progresses.
- **Governance Receipts** — recording governance decisions as verifiable artifacts.
- **Constitutional Replay** — reconstructing governance decisions for independent verification.

Collectively, these mechanisms ensure that governance does not end when a decision is made—it remains enforceable until consequence is complete.

Part III Principle

Governance that cannot enforce consequence is observational. Governance that controls the transition from admissibility to execution is operational.

---

10. Consequence Path Coverage

Introduction

Consequential execution rarely consists of a single isolated action.

Modern AI systems operate through workflows, orchestration engines, autonomous agents, distributed services, and asynchronous processes that generate multiple downstream consequences from a single governance decision.

Within RiCo, governance does not conclude when an execution request is approved.

Governance extends across every consequential path that originates from that decision.

Consequence Path Coverage ensures that every consequential outcome remains within the scope of runtime governance.

Architectural Purpose

Consequence Path Coverage answers the question:

"Does every consequential execution remain governed from initiation to completion?"

A governance decision is only meaningful if every consequential action derived from that decision remains subject to governance.

If downstream consequences occur outside the governance architecture, legitimacy cannot be guaranteed.

Consequence Paths

A single execution request may produce multiple consequence paths.

For example:

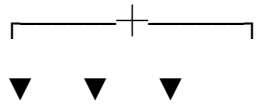
Execution Request

|

▼

Approved Action

|



Database Update

Notification

Financial Transaction

|

▼

External API

|

▼

Audit Record

Although these actions originate from a single decision, each produces an independent operational consequence.

RiCo considers each consequential branch part of the governed execution.

Coverage Principle

RiCo adopts the principle that:

Every consequential execution path shall remain governable from initiation until completion.

Coverage requires that every consequence:

- remains attributable,
- remains traceable,
- remains subject to governance,
- remains replayable,
- remains independently verifiable.

No consequential branch should exist outside the governance architecture.

Runtime Enforcement

Immediately before a consequence is produced, RiCo ensures that the execution path remains within the governed runtime.

Coverage includes:

- primary execution,
- downstream services,
- delegated execution,
- asynchronous workers,
- retries,
- orchestration branches,
- external integrations,
- automated agent actions.

Each path remains subject to runtime governance regardless of how many execution boundaries exist within the workflow.

Partial Coverage

Partial governance introduces architectural risk.

Examples include:

- governing the initial decision but not downstream automation,
- governing internal execution but not third-party integrations,
- governing user actions but not autonomous agents,
- governing approval while allowing retries to bypass evaluation.

Partial coverage creates ungoverned consequence paths.

These paths may produce legitimate operational behavior while simultaneously violating governance integrity.

Consequences of Incomplete Coverage

If every consequence cannot be governed, several risks emerge:

- untraceable execution,
- unauthorized downstream actions,
- inconsistent policy enforcement,
- incomplete audit evidence,
- loss of accountability,
- replay gaps,
- governance fragmentation.

Most importantly, legitimacy can no longer be demonstrated for the complete execution lifecycle.

The system may remain operational, but governance continuity has been broken.

Coverage Verification

RiCo verifies that:

- every consequential path is identifiable,
- every execution branch remains attributable,
- governance applies consistently across distributed workflows,
- downstream execution cannot escape runtime evaluation,
- governance receipts preserve the complete execution path.

Coverage is therefore measurable rather than assumed.

Architectural Characteristics

Consequence Path Coverage within RiCo is:

- End-to-end rather than point-in-time.
- Runtime-enforced.
- Branch-aware.

- Distributed-system aware.
- Independent of implementation architecture.
- Replayable.
- Verifiable through governance evidence.

These characteristics ensure that governance extends beyond the initial decision to encompass the complete lifecycle of consequential execution.

Relationship to Enforcement

Consequence Path Coverage is the first enforcement mechanism introduced after admissibility has been established.

Where admissibility determines **whether execution may begin**, Consequence Path Coverage ensures that governance remains intact **throughout every consequential branch of execution**.

It therefore preserves legitimacy beyond the Runtime Execution Boundary.

Summary

Consequence Path Coverage is the enforcement principle requiring that every consequential execution path derived from a governed decision remains continuously subject to runtime governance, attribution, traceability, and verification until execution is complete.

---

11. Non-Bypassability

Introduction

Runtime governance is only effective if every consequential execution is subject to its evaluation.

If an execution path can circumvent governance, the integrity of the entire architecture is compromised regardless of how sophisticated the governance model may be.

For this reason, RiCo adopts **Non-Bypassability** as a fundamental architectural requirement.

Every consequential execution must pass through the Runtime Integrity Control (RiCo) governance process before crossing the Runtime Execution Boundary (REB).

No alternative execution path shall exist that permits consequence without runtime governance.

Architectural Purpose

Non-Bypassability answers the question:

"Can any consequential execution occur without runtime governance?"

Within RiCo, the required answer is unequivocally:

No.

Every consequential action must be evaluated through the complete runtime governance process before execution is permitted.

Architectural Principle

RiCo establishes a single governance path for consequential execution.

Execution Request

|

▼

Runtime Integrity Control (RiCo)

|

▼

Authority

|

▼

Evidence

|

▼

Context

|

▼

Policy

|

▼

Admissibility

|

▼

Runtime Execution Boundary (REB)

|

▼

Consequential Execution

There are no alternative governance paths.

There are no privileged execution routes.

There are no exceptions that bypass runtime admissibility.

Why Non-Bypassability Matters

A governance architecture may evaluate decisions perfectly.

However, if execution can occur through an alternate mechanism, governance becomes advisory rather than authoritative.

Examples include:

- direct database updates outside RiCo,
- administrative shortcuts,
- ungoverned APIs,
- background services bypassing evaluation,
- autonomous agents executing independently,
- retry mechanisms skipping governance,
- failover paths ignoring admissibility,
- emergency overrides without governance controls.

In each case, execution remains operational while governance has ceased to exist.

Runtime Enforcement

RiCo enforces that every consequential execution:

- enters the governance pipeline,
- completes runtime evaluation,
- establishes admissibility,
- crosses the Runtime Execution Boundary,
- generates a governance receipt.

Execution cannot begin until these steps have been completed.

This requirement applies regardless of:

- execution source,
- orchestration mechanism,
- infrastructure,
- programming language,

- deployment architecture,
- synchronization model.

Governance Integrity

Non-Bypassability preserves governance integrity by ensuring that:

- authority cannot be circumvented,
- evidence cannot be ignored,
- contextual evaluation cannot be skipped,
- policy cannot be overridden outside governance,
- admissibility remains mandatory.

Every consequential action therefore shares the same governance standard.

Failure Modes

Non-Bypassability fails whenever an execution path exists that avoids RiCo.

Examples include:

- undocumented execution paths,
- direct service invocation,
- hidden administrative interfaces,
- privileged execution mechanisms,
- emergency routines without governance,
- asynchronous workers bypassing evaluation,
- orphaned downstream services,
- legacy execution components outside the governance architecture.

Any bypass path invalidates complete governance coverage.

The existence of a single ungoverned execution path introduces an architectural vulnerability because legitimacy can no longer be guaranteed for every consequence.

Verification

A RiCo implementation demonstrates Non-Bypassability by showing that:

- every consequential execution enters RiCo,
- every execution receives an admissibility determination,
- no execution path exists outside the governance pipeline,
- governance receipts exist for every consequential action,
- execution logs correspond to governance records,
- replay confirms complete governance coverage.

Verification therefore focuses on proving the absence of bypass mechanisms rather than merely confirming the presence of governance.

Architectural Characteristics

Non-Bypassability within RiCo is:

- Mandatory.
- System-wide.
- Runtime-enforced.
- Architecture-independent.
- Verifiable.
- Replayable.
- Resistant to alternative execution paths.

These characteristics ensure that governance remains authoritative rather than optional.

Relationship to Consequence Path Coverage

Consequence Path Coverage ensures that every consequence remains governed throughout its execution lifecycle.

Non-Bypassability ensures that every consequence enters that governed lifecycle in the first place.

Together they establish complete enforcement:

- **Consequence Path Coverage** governs every consequential branch.
- **Non-Bypassability** guarantees that no branch can escape governance.

Neither principle is sufficient without the other.

Summary

Non-Bypassability is the enforcement principle requiring that every consequential execution pass through the Runtime Integrity Control governance process before crossing the Runtime Execution Boundary. No execution path, regardless of source, architecture, or operational mechanism, shall permit consequence without runtime governance.

---

12. Temporal Validity

Introduction

Governance exists within time.

Every governance decision is made under a specific set of operational conditions. Those conditions evolve continuously as systems execute, workflows progress, dependencies change, and environments adapt.

Within RiCo, legitimacy is therefore considered **temporally bounded** rather than permanently established.

Temporal Validity ensures that governance remains demonstrably valid at the precise moment consequential execution occurs.

Approval alone is insufficient.

Approval must remain valid over time.

Architectural Purpose

Temporal Validity answers the question:

"Does the governance determination remain valid at the moment consequence occurs?"

A governance decision may have been completely legitimate when originally made.

However, if operational conditions change before execution, legitimacy must be re-established before consequence is permitted.

Temporal Validity prevents historical approvals from becoming operational assumptions.

The Temporal Problem

Modern AI systems rarely execute immediately after a decision is made.

Between approval and execution, time passes.

During that time:

- authority may be revoked,
- evidence may become stale,

- context may change,
- policies may be updated,
- dependencies may fail,
- regulatory conditions may evolve.

A decision that was legitimate earlier may no longer be legitimate when execution actually begins.

Time itself becomes a governance variable.

The RiCo Temporal Model

RiCo represents governance as a sequence of evolving runtime states.

t_1

Decision Approved

Authority ✓

Evidence ✓

Context ✓

Policy ✓

|

|

▼

t_2

Operational Environment Changes

Authority ?

Evidence ?

Context ?

Policy ?

|

t_3

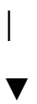
Execution Requested

Runtime Re-Proof

Current-State Validation

Admissibility

Runtime Execution Boundary (REB)



Consequential Execution

The approval established at $\mathbf{t_1}$ is treated as historical evidence—not as continuing authorization.

Only the governance state established at $\mathbf{t_3}$ determines whether execution may proceed.

Why Approval at t_1 Is Insufficient

Traditional governance often assumes that once a decision has been approved, execution may proceed whenever it eventually occurs.

RiCo rejects this assumption.

Between $\mathbf{t_1}$ and $\mathbf{t_3}$, the operational environment may have changed in ways that invalidate the original governance determination.

For example:

- A clinician is authorized to approve a treatment ($\mathbf{t_1}$), but their authorization is revoked before the treatment is administered ($\mathbf{t_3}$).
- A financial transaction is approved based on an account balance that changes before execution.

- A robotic system receives permission to enter an area that later becomes restricted.
- An AI model is approved under one regulatory requirement, but a new policy takes effect before execution.

In each case, the original approval was correct when issued.

It is the passage of time—not the quality of the original decision—that creates the governance risk.

Runtime Re-Proof

To address temporal change, RiCo performs **runtime re-proof** immediately before consequential execution.

Re-proof reconstructs the current governance state by re-evaluating:

- authority,
- evidence,
- context,
- policy.

This process determines whether the original justification continues to exist.

RiCo therefore treats governance as a continuously verifiable property rather than a permanent authorization.

Temporal Invalidation

Governance may become invalid through the normal progression of time.

Temporal invalidation occurs whenever elapsed time changes one or more governance conditions.

Examples include:

- authority expiration,
- evidence aging,
- context evolution,

- policy revision,
- environmental change,
- dependency degradation,
- mission transition.

Temporal invalidation does not imply that the original decision was incorrect.

It simply recognizes that legitimacy must reflect the present operational state.

Runtime Enforcement

Immediately before execution, RiCo confirms that:

- governance remains temporally valid,
- no governance component has expired,
- current operational conditions remain consistent with execution,
- admissibility can be demonstrated under present circumstances.

Only then may execution cross the Runtime Execution Boundary.

Failure Modes

Temporal Validity fails when execution relies upon governance established at an earlier time without confirming that it remains valid.

Examples include:

- executing with expired authority,
- relying on stale evidence,
- ignoring policy updates,
- executing after mission changes,
- assuming historical context remains current,
- permitting delayed workflows without re-validation.

These failures allow historical legitimacy to become operational consequence without demonstrating that legitimacy still exists.

Architectural Characteristics

Temporal Validity within RiCo is:

- Runtime-evaluated.
- Time-aware.
- Independent of historical approval.
- Dependent upon current governance state.
- Re-proof driven.
- Deterministic.
- Fully replayable through governance receipts.

These characteristics ensure that governance remains synchronized with operational reality throughout the execution lifecycle.

Relationship to Admissibility

Temporal Validity is not a separate governance component.

It is the principle that governs **how Authority, Evidence, Context, and Policy are interpreted over time**.

Without Temporal Validity, admissibility becomes a historical decision.

With Temporal Validity, admissibility becomes a current runtime determination.

Temporal Validity therefore preserves **Legitimacy Continuity** by ensuring that governance remains valid as time progresses between decision and consequence.

Summary

Temporal Validity is the enforcement principle requiring that governance remain demonstrably valid at the moment of consequential execution. Within RiCo, historical approval alone is insufficient; authority, evidence, context, and policy must be re-evaluated through runtime re-proof to establish current admissibility before execution is permitted to cross the Runtime Execution Boundary.

13. Governance Receipt

Introduction

Every consequential execution governed by RiCo produces a Governance Receipt.

A Governance Receipt is a verifiable governance artifact that records the runtime determination supporting consequential execution.

Unlike operational logs, which primarily document system activity, a Governance Receipt preserves the governance basis upon which execution was permitted or denied.

It provides independent evidence that consequential execution was evaluated under the current runtime state before crossing the Runtime Execution Boundary (REB).

The Governance Receipt is the permanent record of runtime legitimacy.

Architectural Purpose

The Governance Receipt answers the question:

"What governance determination justified this consequence?"

Rather than documenting only what occurred, the receipt preserves the governance state that authorized the consequence.

This enables independent inspection, verification, challenge, and constitutional replay without reconstructing hidden execution state.

Receipt Generation

A Governance Receipt is generated immediately following the admissibility determination.

Receipts are produced for both:

- **Approved executions**
- **Denied executions**

Recording denied executions is equally important, as it demonstrates that governance enforcement actively prevented inadmissible consequences.

The receipt becomes the authoritative governance artifact associated with the execution request.

Receipt Contents

A Governance Receipt should contain sufficient information to reconstruct the governance decision independently of the executing system.

Typical receipt elements include:

Receipt Metadata

- Receipt Identifier
- Receipt Version
- Timestamp
- Runtime Session Identifier
- Execution Identifier

Governance Determination

- Admissibility Result (Approved / Denied)
- Runtime Decision Identifier
- Evaluation Timestamp
- Runtime Execution Boundary Reference

Authority

- Authority Source
- Authority Scope
- Delegation Information
- Authority Status

Evidence

- Evidence References
- Evidence Freshness
- Evidence Trust Assessment
- Evidence Sufficiency Result

Context

- Runtime Context Identifier
- Environmental State
- Dependency Status
- Identity References
- Execution Surface

Policy

- Applicable Policies
- Policy Versions
- Mission Policies
- Emergency Policies (if applicable)

Outcome

- Execution Authorized / Denied
- Reason Codes
- Enforcement Action
- Replay Reference

The receipt intentionally records governance information rather than operational implementation details.

Receipt Integrity

The Governance Receipt must preserve its integrity throughout its lifecycle.

Integrity requires that the receipt remain:

- complete,
- attributable,
- tamper-evident,
- independently verifiable,

- chronologically ordered,
- permanently traceable.

The receipt should accurately represent the governance state existing at the moment admissibility was determined.

Subsequent operational changes must not retroactively alter the recorded governance determination.

The receipt therefore serves as an immutable governance artifact.

Replay References

Every Governance Receipt contains references supporting Constitutional Replay.

Replay references enable reconstruction of the runtime governance state without repeating execution.

Typical replay references include:

- governance decision identifier,
- authority references,
- evidence references,
- context references,
- policy references,
- admissibility outcome,
- execution reference,
- receipt lineage.

These references allow reviewers to independently verify the governance decision while preserving separation between operational execution and governance evaluation.

Runtime Enforcement

Immediately following admissibility, RiCo generates the Governance Receipt before consequential execution is finalized.

Receipt generation ensures that:

- every governed execution produces evidence,
- every denied execution is documented,
- every governance decision remains attributable,
- every consequence can later be inspected.

Governance therefore remains observable without relying upon application-specific logging mechanisms.

Failure Modes

Governance Receipt failures include:

- missing receipts,
- incomplete governance records,
- corrupted receipt integrity,
- untraceable governance references,
- inconsistent timestamps,
- orphaned execution records,
- missing replay references.

Without a complete Governance Receipt, the governance basis supporting execution cannot be independently demonstrated.

Operational execution may still have occurred, but governance accountability has been lost.

Architectural Characteristics

Governance Receipts within RiCo are:

- Automatically generated.
- Runtime-specific.
- Immutable after creation.
- Independently verifiable.

- Replay-enabled.
- Tamper-evident.
- Architecture-independent.

These characteristics transform governance from an internal implementation detail into verifiable architectural evidence.

Relationship to Constitutional Replay

The Governance Receipt is the foundational artifact supporting Constitutional Replay.

Replay does not reconstruct governance by interpreting application logs or inferring execution history.

Instead, it reconstructs governance directly from the recorded runtime determination preserved within the Governance Receipt.

The receipt therefore serves as the primary evidentiary record of runtime legitimacy.

Summary

A Governance Receipt is the immutable runtime artifact that records the governance determination supporting consequential execution. Within RiCo, every admissibility decision produces a receipt containing the authority, evidence, context, policy, and enforcement state necessary to independently verify, audit, and replay the legitimacy of consequential execution.

---

14. Constitutional Replay

Introduction

Constitutional Replay is the verification mechanism through which RiCo reconstructs the governance legitimacy of a consequential execution.

Unlike operational replay mechanisms, which reproduce execution behavior, Constitutional Replay reconstructs the governance determination that authorized consequence.

Its objective is not to repeat execution.

Its objective is to determine whether the consequence was legitimately authorized under the runtime conditions that existed when execution crossed the Runtime Execution Boundary (REB).

Constitutional Replay therefore verifies governance rather than system behavior.

Architectural Purpose

Constitutional Replay answers the question:

"Given the runtime conditions that existed at execution, was this consequence legitimately authorized?"

This question differs fundamentally from operational debugging.

Traditional replay attempts to reproduce execution.

Constitutional Replay reconstructs legitimacy.

Replay Is Not Logging

Operational logs record events.

Audit logs record activities.

Execution traces record program behavior.

These records answer questions such as:

- What happened?
- When did it happen?
- Which component executed?

- What output was produced?

Constitutional Replay addresses a different class of questions:

- Why was execution permitted?
- What governance determination existed?
- Was authority still valid?
- Was evidence sufficient?
- Had policy changed?
- Did the runtime context support consequence?

Replay therefore evaluates governance rather than execution history.

Logs describe events.

Constitutional Replay reconstructs legitimacy.

Governance Reconstruction

Constitutional Replay reconstructs the runtime governance state using the Governance Receipt and its associated references.

The replay process reconstructs:

- authority,
- evidence,
- context,
- policy,
- admissibility,
- enforcement outcome.

The objective is not to recreate application state.

The objective is to recreate the governance state that justified consequence.

Constitutional Basis

The term *Constitutional* reflects that replay is evaluated against the governing principles defined by the architecture rather than against implementation-specific behavior.

Replay determines whether execution remained consistent with:

- governance policy,
- authority constraints,
- evidence requirements,
- contextual conditions,
- admissibility principles,
- Runtime Execution Boundary requirements.

This creates an implementation-independent method of verifying governance legitimacy.

Replay Inputs

Constitutional Replay relies upon governance artifacts rather than operational assumptions.

Typical replay inputs include:

- Governance Receipt,
- Authority references,
- Evidence references,
- Context references,
- Policy references,
- Admissibility determination,
- Runtime timestamps,
- Execution identifiers.

These artifacts allow governance decisions to be reconstructed without dependence upon internal application logic.

Replay Outcomes

Constitutional Replay may produce several outcomes.

Legitimate

The reconstructed governance state demonstrates that execution satisfied all runtime governance requirements.

The consequence is confirmed as constitutionally admissible.

Inconclusive

Available governance artifacts are insufficient to reconstruct the governance determination.

Execution may have occurred, but legitimacy cannot be independently demonstrated.

Illegitimate

Replay demonstrates that one or more governance requirements were not satisfied at the time of execution.

Examples include:

- expired authority,
- stale evidence,
- invalid context,
- conflicting policy,
- missing governance evaluation.

In this case, the consequence cannot be constitutionally justified.

Runtime Independence

Constitutional Replay operates independently of the executing application.

Its purpose is not to determine whether software behaved correctly.

Its purpose is to determine whether governance behaved correctly.

This distinction separates governance verification from software verification.

A technically successful execution may still fail Constitutional Replay if runtime legitimacy cannot be demonstrated.

Failure Modes

Replay integrity is compromised when:

- Governance Receipts are missing,
- governance references are incomplete,
- evidence cannot be reconstructed,
- authority lineage is unavailable,
- policy versions cannot be identified,
- runtime context cannot be established,
- admissibility records are absent.

Without sufficient governance artifacts, legitimacy cannot be reconstructed independently.

Replay therefore depends upon the integrity of the governance evidence produced throughout execution.

Architectural Characteristics

Constitutional Replay within RiCo is:

- Governance-centric.
- Independent of implementation.
- Evidence-driven.
- Deterministic.
- Reconstructive rather than descriptive.
- Constitutionally verifiable.
- Based upon runtime governance artifacts rather than operational logs.

These characteristics ensure that governance decisions remain independently challengeable long after execution has completed.

Relationship to Governance Receipts

Governance Receipts preserve the runtime governance state.

Constitutional Replay reconstructs that state.

Together they establish a complete governance evidence chain.

Without Governance Receipts, replay lacks evidentiary foundation.

Without Constitutional Replay, Governance Receipts remain historical records without independent verification.

The two mechanisms therefore operate as complementary components of the RiCo architecture.

Summary

Constitutional Replay is the governance verification mechanism that reconstructs the runtime legitimacy of consequential execution using Governance Receipts and associated governance artifacts. Within RiCo, replay does not reproduce execution behavior; it reconstructs the authority, evidence, context, policy, and admissibility determination that justified consequence at the moment execution crossed the Runtime Execution Boundary.

Part IV — Verification

Introduction

A governance architecture cannot establish trust through design claims alone.

It must produce evidence that its controls operated as specified, that no execution escaped governance, and that every consequential outcome can be independently examined.

Within RiCo, verification is not a post-deployment audit activity added after implementation.

Verification is an architectural capability designed into the runtime governance system from the beginning.

RiCo must not merely govern execution.

It must be able to prove that governance occurred.

From Enforcement to Proof

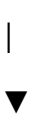
Part III established the enforcement requirements of RiCo:

- every consequence path must remain governed,
- no consequential execution may bypass RiCo,
- governance must remain temporally valid,
- every decision must produce a Governance Receipt,
- and legitimacy must be reconstructable through Constitutional Replay.

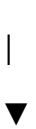
Part IV converts those requirements into verifiable claims.

Architectural Requirement

Runtime Enforcement



Governance Evidence



Independent Verification

Conformance Determination

The objective is not simply to inspect whether RiCo components exist.

The objective is to determine whether the implementation behaves as a RiCo-governed architecture under real operating conditions.

Architectural Proof

Architectural proof within RiCo consists of evidence that demonstrates:

1. every consequential request entered the governance pipeline,
2. authority, evidence, context, and policy were evaluated,
3. admissibility was determined immediately before execution,
4. execution crossed the Runtime Execution Boundary only after approval,
5. denied execution did not produce consequence,
6. every consequence generated a valid Governance Receipt,
7. governance artifacts remained complete and tamper-evident,
8. Constitutional Replay could reconstruct the original legitimacy determination.

These claims must be demonstrated rather than assumed.

Verification Is Not Observation

Observability confirms that system activity can be seen.

Verification determines whether architectural requirements were satisfied.

A dashboard may show that RiCo processed an execution request.

A verification process must establish that:

- the request could not have avoided RiCo,
- the correct governance inputs were evaluated,
- the governing policy version was valid,
- the admissibility result controlled execution,
- and the resulting receipt accurately preserves the decision.

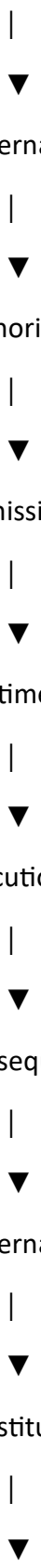
Observation shows activity.

Verification proves conformance.

Verification Scope

Verification applies to the complete consequential lifecycle.

Execution Request


Governance Entry
Authority – Evidence – Context – Policy
Admissibility
Runtime Execution Boundary
Execution
Consequence
Governance Receipt
Constitutional Replay

Conformance Result

No individual stage is sufficient by itself.

A valid admissibility determination does not prove that the execution path obeyed it.

A Governance Receipt does not prove legitimacy unless its contents and integrity can be verified.

A successful replay does not prove complete governance if an alternate execution path existed outside RiCo.

Verification must therefore evaluate the architecture as a connected system.

Core Verification Questions

Part IV should establish answers to five fundamental questions.

1. Coverage

Did every consequential execution path remain within the governance architecture?

This verifies Consequence Path Coverage.

---

2. Traversal

Did every consequential execution traverse RiCo before crossing the Runtime Execution Boundary?

This verifies Non-Bypassability.

---

3. Currency

Was the governance determination valid at the actual moment of execution?

This verifies Temporal Validity.

---

4. Evidence

Does a complete and tamper-evident Governance Receipt exist for the determination?

This verifies Governance Receipt integrity.

---

5. Reconstruction

Can an independent verifier reconstruct whether the consequence was legitimately authorized?

This verifies Constitutional Replay.

Together, these questions define the verification surface of RiCo.

Verification Methods

A RiCo implementation may use several verification methods.

Structural Verification

Examines whether the architecture contains the required governance controls and whether all consequential execution surfaces are connected to RiCo.

Structural verification evaluates:

- execution topology,
- service boundaries,
- integration points,
- privileged interfaces,
- asynchronous workers,
- retry paths,
- failover paths,
- administrative mechanisms.

Its purpose is to identify architectural routes through which consequence could occur outside governance.

Runtime Verification

Examines actual execution behavior.

Runtime verification evaluates whether:

- RiCo received the request,

- governance inputs were acquired,
- admissibility was determined,
- enforcement followed the result,
- a receipt was generated,
- execution and governance records correspond.

This verifies that the implemented architecture performs as designed during live operation.

Adversarial Verification

Attempts to violate RiCo's enforcement requirements.

Examples include:

- invoking a service directly,
- replaying an expired approval,
- substituting stale evidence,
- changing policy between approval and execution,
- triggering asynchronous execution without reevaluation,
- attempting privileged administrative execution,
- corrupting or deleting a Governance Receipt.

A conformant system must fail safely under these conditions.

Evidentiary Verification

Examines the Governance Receipt and associated references.

It determines whether the recorded artifacts are:

- complete,
- authentic,
- internally consistent,
- chronologically valid,

- attributable,
- replayable,
- independently verifiable.

This establishes whether governance proof survives beyond the original runtime session.

Replay Verification

Reconstructs the governance state that existed at execution.

Replay verification determines whether the same historical inputs produce a governance conclusion consistent with the recorded receipt.

The objective is not to reproduce the operational consequence.

It is to reconstruct the legitimacy determination.

Positive and Negative Proof

RiCo verification must prove both that authorized execution occurred correctly and that unauthorized execution was prevented.

Positive Proof

Positive proof demonstrates that:

- an admissible request was evaluated,
- approval controlled execution,
- the consequence matched the authorized scope,
- a valid Governance Receipt was produced,
- replay confirms legitimacy.

Negative Proof

Negative proof demonstrates that:

- an inadmissible request was denied,
- the denial prevented consequential execution,
- no alternate path produced the prohibited consequence,

- the denial generated a Governance Receipt,
- replay confirms why execution was prohibited.

This distinction matters because a system may correctly approve legitimate actions while failing to stop illegitimate ones.

A governance architecture must prove both permission and prevention.

Verification Independence

Verification should not depend solely upon the component being verified.

An application cannot establish its own legitimacy merely by reporting that it complied.

Where practical, verification should be performed through independent mechanisms capable of examining:

- execution evidence,
- governance evidence,
- policy versions,
- authority lineage,
- receipt integrity,
- replay outcomes.

The greater the consequence, the stronger the required independence should be.

This does not require a single implementation model, but it does require separation between architectural claim and proof.

Conformance

Verification ultimately determines whether an implementation conforms to the RiCo architecture.

A system should not be considered RiCo-conformant merely because it contains a policy engine, an approval process, or an audit log.

Conformance requires demonstrable satisfaction of the complete runtime governance model.

A conformant implementation must establish that:

- runtime admissibility governs consequence,
- the REB cannot be crossed without approval,
- every consequence path is covered,
- bypass is architecturally prevented,
- temporal validity is re-established,
- Governance Receipts are produced and protected,
- Constitutional Replay can reconstruct legitimacy.

Failure of any critical requirement constitutes architectural non-conformance.

Verification Outcomes

Verification may produce one of four outcomes.

Conformant

The implementation satisfies the required RiCo architectural controls and provides sufficient evidence to prove conformance.

Conditionally Conformant

The core architecture is implemented, but identified limitations restrict coverage, verification independence, or operational scope.

These limitations must be explicitly documented.

Non-Conformant

One or more required controls are absent, bypassable, unenforced, or unverifiable.

Indeterminate

Available evidence is insufficient to establish whether the architecture conforms.

An indeterminate result must not be treated as successful verification.

Absence of proof is not proof of governance.

Part IV Structure

Part IV contains two implementation-facing chapters:

15. Conformance Evidence Profile

Defines the architectural, runtime, enforcement, receipt, integrity, denial, adversarial, and replay evidence required to establish RiCo conformance.

16. Testable Architecture

Defines measurable PASS, FAIL, and INDETERMINATE outcomes and establishes test requirements for replay, worker paths, queues, retries, interrupts, revalidation, inspection, and falsification.

Together, these chapters move from what must be proven to how that proof is produced, challenged, inspected, and evaluated.

Summary

Verification is the architectural process through which a RiCo implementation demonstrates that every consequential execution was governed, every enforcement decision remained binding, every governance determination produced verifiable evidence, and every consequence can be independently reconstructed as legitimate or illegitimate through Constitutional Replay.

---

15. Conformance Evidence Profile

Introduction

The Conformance Evidence Profile defines the evidence required to determine whether an implementation conforms to the Runtime Integrity Control architecture.

RiCo conformance cannot be established through architectural claims, product descriptions, design diagrams, or self-attestation alone.

Conformance must be demonstrated through evidence showing that:

- every consequential execution is governed,
- no execution path can bypass RiCo,
- runtime admissibility controls consequence,
- governance remains temporally valid,
- Governance Receipts preserve the basis of each determination,
- Constitutional Replay can reconstruct legitimacy.

The Conformance Evidence Profile establishes the minimum evidentiary foundation required to support those claims.

Architectural Purpose

The Conformance Evidence Profile answers two questions:

How do we know that an implementation conforms to RiCo?

and:

What evidence must exist to prove that conformance?

The profile translates architectural principles into observable and testable evidence requirements.

It does not prescribe a specific programming language, cloud platform, database, cryptographic mechanism, or deployment model.

It defines what must be provable regardless of implementation.

Conformance Is an Evidentiary Determination

RiCo conformance is not established by the presence of individual components.

A system may contain:

- an authorization service,
- a policy engine,
- an audit database,
- execution logs,
- approval workflows,
- monitoring dashboards,

and still fail to conform.

Conformance depends upon whether those components operate together as a non-bypassable runtime governance architecture.

The implementation must demonstrate that governance directly controls the transition from admissibility to consequence.

Conformance Claim Model

Every conformance determination should be structured around four elements:

Architectural Requirement

↓

Conformance Claim

↓

Supporting Evidence

↓

Verification Result

Architectural Requirement

The mandatory RiCo property being evaluated.

Example:

Every consequential execution must traverse RiCo before crossing the Runtime Execution Boundary.

Conformance Claim

The implementation-specific assertion that the requirement has been satisfied.

Example:

All production execution interfaces invoke the RiCo enforcement gateway before accessing consequential services.

Supporting Evidence

The artifacts demonstrating that the claim is true.

Example:

- execution topology,
- interface inventory,
- runtime traces,
- access-control configuration,
- adversarial bypass test results,
- Governance Receipts.

Verification Result

The independent determination of whether the evidence sufficiently supports the claim.

A conformance claim without supporting evidence is unverified.

Evidence without a defined requirement is unactionable.

Required Evidence Categories

A RiCo-conformant implementation must maintain evidence across the complete governance lifecycle.

1. Architecture Evidence

Architecture evidence demonstrates how the implementation realizes the RiCo model.

Required artifacts should include:

- system architecture diagram,
- runtime governance topology,
- Runtime Execution Boundary identification,
- consequential service inventory,
- execution entry-point inventory,
- trust-boundary documentation,
- data-flow documentation,
- external integration inventory,
- asynchronous and delegated execution paths,
- administrative and emergency execution paths.

Architecture evidence must clearly identify where RiCo operates and where consequence becomes possible.

A diagram showing RiCo as an isolated component is insufficient.

The evidence must demonstrate its relationship to every consequential execution surface.

---

2. Consequence Path Evidence

Consequence Path Evidence demonstrates that every direct and downstream consequence has been identified and remains governed.

Required artifacts should include:

- consequence path inventory,
- workflow and orchestration maps,
- downstream service mappings,
- delegated execution mappings,
- asynchronous worker mappings,
- retry and failover paths,
- external API consequence mappings,

- consequence ownership and attribution records.

Each consequential path should have a unique identifier and a documented relationship to the originating governance determination.

Unknown or undocumented consequence paths prevent complete conformance.

---

3. Non-Bypassability Evidence

Non-Bypassability Evidence proves that no consequential execution can avoid RiCo.

Required artifacts should include:

- interface access controls,
- service-to-service authorization rules,
- network enforcement configuration,
- execution gateway configuration,
- privileged-access restrictions,
- direct invocation test results,
- administrative path test results,
- retry and failover test results,
- bypass-attempt records,
- denied execution evidence.

The evidence must prove more than normal-path compliance.

It must demonstrate that alternate and privileged routes cannot reach consequence without governance traversal.

A single ungoverned execution path constitutes a critical conformance failure.

---

4. Governance Input Evidence

Governance Input Evidence demonstrates that Authority, Evidence, Context, and Policy are evaluated using identifiable runtime inputs.

Required artifacts should include:

Authority Evidence

- authority source,
- authority scope,
- delegation lineage,
- expiration data,
- revocation status,
- identity binding,
- validation result.

Evidence

- evidence identifiers,
- provenance,
- integrity status,
- freshness data,
- trust evaluation,
- sufficiency determination,
- evidence version.

Context Evidence

- runtime context snapshot,
- dependency status,
- environmental state,
- actor identity,
- execution surface,
- operational conditions,
- context timestamp.

Policy Evidence

- applicable policy identifiers,

- policy versions,
- policy effective dates,
- policy precedence,
- exception conditions,
- emergency policy references,
- policy evaluation result.

The implementation must preserve enough evidence to determine what governance inputs were evaluated, when they were evaluated, and which versions applied.

---

5. Admissibility Evidence

Admissibility Evidence demonstrates that RiCo produced a current runtime determination before consequence.

Required artifacts should include:

- admissibility decision identifier,
- decision timestamp,
- evaluated governance inputs,
- decision outcome,
- decision reason codes,
- applicable constraints,
- validity duration,
- Runtime Execution Boundary reference,
- enforcement instruction.

The evidence must establish that admissibility was not merely calculated, but used as the controlling condition for execution.

---

6. Temporal Validity Evidence

Temporal Validity Evidence demonstrates that governance remained valid at the moment of execution.

Required artifacts should include:

- original approval timestamp,
- runtime re-proof timestamp,
- execution timestamp,
- authority validity interval,
- evidence freshness interval,
- policy effective interval,
- context acquisition timestamp,
- admissibility validity window,
- expiration or revocation checks,
- changes occurring between approval and execution.

The implementation must be able to demonstrate the temporal relationship:

t_1 — Initial Decision or Approval

↓

t_2 — Operational Change or Elapsed Time

↓

t_3 — Runtime Re-Proof and Admissibility

↓

t_4 — Consequential Execution

Conformance requires proof that the governance state used at $\mathbf{t_3}$ remained valid at $\mathbf{t_4}$.

Evidence from $\mathbf{t_1}$ alone is insufficient.

---

7. Enforcement Evidence

Enforcement Evidence demonstrates that the admissibility result controlled whether consequence occurred.

Required artifacts should include:

- execution authorization token or equivalent control artifact,
- enforcement decision,
- execution gate result,
- approved execution trace,
- denied execution trace,
- consequence identifier,
- execution-to-receipt correlation,
- proof that denied execution produced no consequence.

The implementation must prove both:

- approved execution was permitted within authorized scope,
- denied execution was prevented.

A decision record without corresponding enforcement evidence does not establish conformance.

---

8. Governance Receipt Evidence

Every admissibility determination must produce a Governance Receipt.

The receipt should contain, at minimum:

Identity and Correlation

- receipt identifier,
- request identifier,
- execution identifier,
- consequence identifier,
- runtime session identifier.

Temporal Information

- request timestamp,
- evaluation timestamp,
- admissibility timestamp,
- execution timestamp where applicable,
- receipt creation timestamp.

Governance State

- authority references,
- evidence references,
- context references,
- policy references,
- applicable versions,
- validation results.

Determination

- admissibility result,
- reason codes,
- constraints,
- enforcement action,
- denial basis where applicable.

Integrity

- receipt version,
- integrity proof,
- signer or issuing authority,
- hash or equivalent tamper-evidence mechanism,
- receipt lineage.

Replay

- replay package reference,
- governance artifact references,
- policy snapshot reference,
- authority lineage reference,
- evidence snapshot reference,
- context reconstruction reference.

A receipt that records only the outcome is insufficient.

It must preserve the basis of the outcome.

---

9. Receipt Integrity Evidence

Receipt Integrity Evidence demonstrates that Governance Receipts cannot be modified without detection.

Required artifacts should include:

- integrity validation method,
- cryptographic or equivalent tamper-evidence mechanism,
- signing-key or issuer controls,
- receipt verification procedure,
- storage protection controls,
- retention configuration,
- access history,
- integrity-check results,
- corruption-detection test results.

RiCo does not require one specific integrity technology.

However, the selected mechanism must make unauthorized modification detectable and independently verifiable.

---

10. Constitutional Replay Evidence

Constitutional Replay Evidence demonstrates that legitimacy can be reconstructed independently.

Required artifacts should include:

- replay request identifier,
- replay input package,
- historical authority state,
- historical evidence state,
- historical context state,
- applicable policy versions,
- reconstructed admissibility result,
- original receipt result,
- comparison outcome,
- replay verifier identity,
- replay timestamp.

Replay evidence must establish that the governance determination can be reconstructed without re-executing the consequence.

The replay package should support one of the following outcomes:

- Legitimate
- Illegitimate
- Inconclusive

A replay result must never be declared legitimate merely because the operational execution succeeded.

---

11. Failure and Denial Evidence

A conformant implementation must preserve evidence of failed governance evaluations and denied execution.

Required artifacts should include:

- denied request identifier,
- failed governance component,
- denial reason,
- enforcement response,
- attempted consequence path,
- proof of consequence prevention,
- denial Governance Receipt,
- replay reference.

Denied actions provide essential evidence that RiCo is operating as an enforcement architecture rather than an observational system.

A system that records successful approvals but cannot demonstrate prevented consequences has not fully proven conformance.

---

12. Adversarial Test Evidence

The Conformance Evidence Profile must include evidence from attempts to violate the architecture.

Required test categories should include:

- direct service invocation,
- ungoverned API invocation,
- expired authority use,
- revoked authority use,
- stale evidence substitution,
- context manipulation,
- policy-version substitution,
- delayed execution after approval,
- asynchronous bypass,
- retry bypass,

- failover bypass,
- administrative override,
- receipt modification,
- receipt deletion,
- replay with missing evidence.

Each test record should contain:

- test identifier,
- requirement tested,
- initial conditions,
- attack or failure action,
- expected behavior,
- observed behavior,
- resulting evidence,
- conformance outcome.

Successful normal execution alone cannot prove Non-Bypassability.

Adversarial evidence is required.

Evidence Quality Requirements

The existence of evidence is not sufficient.

Evidence must satisfy minimum quality properties.

Completeness

The evidence covers all required governance stages and consequence paths.

Authenticity

The evidence can be attributed to a known source or issuer.

Integrity

Unauthorized modification can be detected.

Traceability

Evidence can be correlated across request, determination, execution, consequence, receipt, and replay.

Temporal Accuracy

Timestamps and validity intervals accurately represent the sequence of events.

Reproducibility

A verifier can repeat the verification process and obtain a consistent result.

Independence

The evidence can be examined without relying exclusively upon the assertions of the component being verified.

Retention

Evidence remains available for the required governance, legal, contractual, or operational period.

Evidence failing these properties may exist operationally but remain insufficient for conformance.

Evidence Correlation

All conformance evidence should be connected through a common correlation structure.

Request ID

↓

Governance Evaluation ID

↓

Admissibility Decision ID

↓

REB Crossing ID

↓

Execution ID

↓

Consequence ID

↓

Governance Receipt ID

↓

Replay ID

The exact identifier format is implementation-specific.

However, the evidence chain must allow an independent verifier to move from any observed consequence backward to its governance determination and forward to its replay result.

Orphaned records break conformance traceability.

Minimum Conformance Evidence Package

A complete RiCo conformance submission should contain at least:

1. Architecture and runtime topology.
2. Runtime Execution Boundary specification.
3. Consequence path inventory.
4. Execution entry-point inventory.
5. Non-Bypassability control evidence.
6. Authority evaluation evidence.
7. Evidence evaluation evidence.
8. Context evaluation evidence.
9. Policy evaluation evidence.
10. Admissibility records.
11. Temporal re-proof evidence.

12. Enforcement records.
13. Governance Receipts.
14. Receipt integrity verification.
15. Constitutional Replay results.
16. Denial and prevention evidence.
17. Adversarial test results.
18. Known limitations and exclusions.
19. Verification methodology.
20. Final conformance determination.

The package should be versioned and tied to a defined implementation, configuration, operational environment, and assessment period.

Conformance Scope

Conformance must always be bounded.

A conformance determination should identify:

- implementation version,
- deployment environment,
- services included,
- consequence types included,
- policies evaluated,
- assessment period,
- known exclusions,
- verification authority,
- evidence package version.

A system should not claim universal RiCo conformance when only one workflow or execution surface has been assessed.

The conformance statement must reflect the actual verified scope.

Evidence Gaps

An evidence gap exists when a required claim cannot be supported by sufficient evidence.

Examples include:

- an undocumented consequence path,
- missing authority lineage,
- unavailable policy version,
- missing context snapshot,
- no proof that denial prevented execution,
- missing Governance Receipt,
- receipt without integrity protection,
- replay that cannot reconstruct legitimacy.

Evidence gaps should be classified as:

Critical

The gap prevents proof of a mandatory architectural property.

Examples:

- possible bypass path,
- missing receipt,
- no REB enforcement proof.

Major

The architecture appears present, but verification is incomplete or materially limited.

Minor

The evidence is sufficient for the conformance claim, but documentation or traceability requires improvement.

Informational

The observation does not affect current conformance but may improve future verification.

A critical evidence gap should result in non-conformance or an indeterminate determination.

Conformance Determination

The final determination should be based upon the evidence profile rather than implementation intent.

Conformant

All mandatory claims are supported by sufficient, valid, and independently verifiable evidence.

Conditionally Conformant

The implementation satisfies the core requirements within a clearly limited scope, with documented restrictions that do not create an ungoverned consequence path inside that scope.

Non-Conformant

Evidence demonstrates that one or more mandatory RiCo requirements are absent, bypassable, unenforced, or violated.

Indeterminate

Evidence is insufficient to determine whether mandatory requirements are satisfied.

An indeterminate implementation must not be represented as conformant.

Architectural Characteristics

The Conformance Evidence Profile is:

- Requirement-based.
- Evidence-driven.
- Runtime-specific.
- Implementation-neutral.
- Independently verifiable.
- Scope-bounded.
- Replay-enabled.
- Suitable for repeated assessment.

- Capable of supporting certification.

These characteristics ensure that RiCo conformance is based on demonstrable architectural behavior rather than branding or self-description.

Relationship to Verification

Part IV establishes that RiCo must prove its architecture.

The Conformance Evidence Profile defines the proof that must exist.

It connects each core RiCo principle to an evidentiary requirement:

Consequence Path Coverage

↓

Consequence Path Evidence

Non-Bypassability

↓

Traversal and Bypass-Test Evidence

Temporal Validity

↓

Runtime Re-Proof Evidence

Governance Receipt

↓

Receipt Content and Integrity Evidence

Constitutional Replay

↓

Legitimacy Reconstruction Evidence

This relationship makes every major architectural claim testable.

Summary

The Conformance Evidence Profile is the structured set of architectural, runtime, enforcement, receipt, integrity, and replay artifacts required to determine whether an implementation conforms to RiCo. Conformance exists only when sufficient evidence demonstrates that every consequential execution was governed, no path escaped runtime control, admissibility remained temporally valid, enforcement followed the governance determination, and legitimacy can be independently reconstructed through Constitutional Replay.

Governing Principle

The Principle of Evidentiary Conformance

No implementation shall be considered RiCo-conformant solely because it claims to implement RiCo controls. Conformance must be established through complete, attributable, tamper-evident, and independently verifiable evidence demonstrating that every consequential execution satisfied the mandatory runtime governance requirements of the architecture.

RiCo conformance is not the presence of governance components. It is the existence of evidence proving that governance remained binding from request to consequence and reconstructable after execution.

---

16. Testable Architecture

Introduction

A governance architecture is only meaningful if its claims can be tested.

RiCo does not require implementers to accept architectural principles through assertion, interpretation, or trust. It requires implementations to expose measurable behavior demonstrating whether runtime governance operated correctly.

A RiCo implementation must make it possible to test:

- whether an execution request passed or failed,
- whether denied execution produced no consequence,
- whether every worker path remained governed,
- whether queued work was revalidated before execution,
- whether retries repeated governance evaluation,
- whether interruption changed admissibility,
- whether replay reconstructed legitimacy,
- whether evidence could be independently inspected,
- and whether the implementation's conformance claims could be falsified.

The architecture is not complete merely because it can operate.

It is complete when its operation can be measured and its claims can be disproven when false.

Architectural Purpose

Testable Architecture answers the question:

Can every material RiCo claim be translated into an observable condition with a defined expected result?

Within RiCo, architectural properties must not remain abstract.

They must be expressed through:

- test conditions,
- expected outcomes,

- observable evidence,
- failure criteria,
- replay results,
- conformance determinations.

The architecture must therefore define not only what should occur, but how an independent verifier can determine whether it actually occurred.

The RiCo Test Model

Every test should connect a governance requirement to measurable behavior.

Architectural Requirement

↓

Test Condition

↓

Controlled Input

↓

Observed Runtime Behavior

↓

Governance Evidence

↓

PASS / FAIL / INDETERMINATE

A valid test must identify:

- the requirement being evaluated,
- the execution path under test,
- the initial governance state,
- the event or condition introduced,

- the expected governance response,
- the observed execution response,
- the resulting Governance Receipt,
- the replay outcome,
- the final test result.

PASS

Definition

A test produces **PASS** when the implementation demonstrates the expected RiCo behavior and sufficient evidence exists to verify that behavior.

PASS requires more than the production of the expected operational outcome.

It requires proof that the outcome occurred through the required governance path.

For example, an authorized transaction completing successfully does not by itself constitute PASS.

The test must also demonstrate that:

- the request entered RiCo,
- Authority, Evidence, Context, and Policy were evaluated,
- admissibility was established,
- the Runtime Execution Boundary was crossed only after approval,
- the consequence remained within authorized scope,
- a valid Governance Receipt was generated,
- replay reconstructed the legitimacy determination.

A correct consequence reached through an ungoverned path is not PASS.

PASS Criteria

A PASS result should require:

- expected governance decision,

- expected enforcement behavior,
- expected consequence or prevention,
- complete evidence correlation,
- valid receipt integrity,
- successful replay,
- no unexplained execution branch,
- no critical evidence gap.

PASS therefore means:

The implementation behaved correctly, governance remained binding, and sufficient evidence exists to prove both.

FAIL

Definition

A test produces **FAIL** when the implementation violates an expected RiCo requirement or when observed behavior contradicts the claimed architecture.

Examples include:

- execution occurred after a denial,
- a worker executed without revalidation,
- a queue delivered an expired approval,
- a retry bypassed governance,
- a receipt could be modified undetectably,
- replay produced a result inconsistent with the original determination,
- a direct service invocation reached consequence without RiCo,
- an interrupt changed conditions but execution continued under the original approval.

FAIL must remain distinct from operational error.

A service crash may be an operational failure.

A consequence occurring without valid governance is a RiCo conformance failure.

FAIL Criteria

A FAIL result should be declared when:

- a mandatory governance stage was skipped,
- a consequential path escaped coverage,
- the REB was crossed without admissibility,
- a denial failed to prevent consequence,
- governance inputs were stale or invalid,
- runtime revalidation did not occur when required,
- receipt integrity could not be established,
- replay contradicted the recorded governance basis,
- or the system's conformance claim was demonstrably false.

A single critical failure may be sufficient to invalidate conformance for the affected scope.

INDETERMINATE

Although PASS and FAIL are the primary outcomes, some tests may produce **INDETERMINATE**.

Indeterminate means the available evidence is insufficient to establish whether the requirement was satisfied.

Examples include:

- missing runtime traces,
- incomplete receipt references,
- unavailable policy versions,
- uncorrelated worker execution,
- missing queue timestamps,
- replay artifacts that cannot be reconstructed.

Indeterminate must not be interpreted as PASS.

Unverifiable governance is not proven governance.

Replay

Purpose

Replay testing determines whether the governance legitimacy of an execution can be reconstructed using preserved governance artifacts.

Replay does not repeat the operational consequence.

It reconstructs the governance state that existed when execution occurred.

A replay test should evaluate whether:

- the original authority state can be identified,
- evidence can be resolved and validated,
- runtime context can be reconstructed,
- the applicable policy version can be recovered,
- the admissibility result can be reproduced,
- the reconstructed result matches the Governance Receipt.

Replay Test Outcomes

Replay Match

The reconstructed governance state produces the same admissibility result and materially equivalent reason basis as the original receipt.

Replay Mismatch

The reconstructed governance state produces a different result or reveals inconsistency in the original determination.

Replay Inconclusive

Required governance artifacts are missing, damaged, ambiguous, or insufficient.

A replay mismatch is a critical verification event.

A replay-inconclusive result identifies an evidentiary failure even when the original operational execution appeared successful.

Worker Path

Definition

A worker path is any execution route in which consequential work is performed by a process other than the component that initiated the original request.

Worker paths include:

- background workers,
- task processors,
- autonomous agents,
- scheduled jobs,
- serverless functions,
- distributed executors,
- delegated services,
- batch-processing systems.

Workers create governance risk because decision and execution may occur in different processes, systems, or times.

Worker Path Requirements

Every consequential worker path must:

- receive a valid governance reference,
- preserve request and decision correlation,
- revalidate admissibility when required,
- remain unable to execute without governance proof,
- generate or update the associated Governance Receipt,
- preserve consequence attribution.

A worker must not treat possession of a task as permanent authorization to execute it.

The worker must determine whether the task remains admissible at execution time.

Worker Path Test

Governed Request

↓

Task Created

↓

Worker Receives Task

↓

Runtime Revalidation

↓

PASS → Execute

FAIL → Prevent

↓

Governance Receipt

The test should verify that direct worker invocation, altered tasks, expired tasks, and orphaned tasks cannot produce consequence.

Queue

Definition

A queue introduces temporal and architectural separation between request approval and execution.

The existence of a queue means that a task may be approved at one time and executed at another.

Therefore:

A queue entry is not proof of continuing admissibility.

A queued message is an execution request, not a permanent authorization.

Queue Requirements

A governed queue must preserve:

- request identifier,
- governance decision reference,
- creation timestamp,
- validity constraints,
- policy references,
- evidence references,
- context requirements,
- retry state,
- worker assignment,
- replay references.

Before executing queued work, RiCo must determine whether the governance state remains valid.

Queue Tests

Queue testing should include:

- delayed delivery,
- duplicate delivery,
- reordered delivery,
- expired messages,
- altered payloads,
- missing governance references,

- revoked authority after enqueue,
- policy change before dequeue,
- context change before execution,
- queue restoration after outage.

The expected behavior is not always execution.

In many cases, the expected behavior is revalidation, denial, quarantine, or escalation.

Retry

Definition

A retry is a new execution attempt.

It is not merely a continuation of the original attempt.

Between attempts:

- time may have passed,
- context may have changed,
- policy may have changed,
- authority may have expired,
- evidence may have become stale,
- the consequence may already have partially occurred.

RiCo therefore treats retry as a governance event.

Retry Requirements

Each retry must:

- preserve lineage to the original request,
- receive a unique attempt identifier,
- determine whether revalidation is required,
- confirm that the requested consequence remains admissible,

- evaluate partial-completion state,
- prevent duplicate or excessive consequence,
- produce updated receipt evidence.

A retry mechanism must never convert a previous PASS into indefinite authorization.

Retry Test Cases

Tests should include:

- retry after transient failure,
- retry after policy change,
- retry after authority revocation,
- retry after evidence expiration,
- retry after partial consequence,
- retry after worker restart,
- retry from a dead-letter queue,
- manual administrative retry,
- repeated retry beyond allowed limits.

A conformant system must distinguish operational recoverability from governance validity.

Interrupt

Definition

An interrupt is any event that pauses, alters, suspends, redirects, or terminates an execution flow before consequence is complete.

Interrupts include:

- process termination,
- network loss,
- dependency failure,

- operator pause,
- policy emergency,
- authority revocation,
- system failover,
- resource exhaustion,
- safety shutdown.

An interrupt may change the legitimacy of continued execution.

Interrupt Requirements

Following an interrupt, the system must determine:

- whether execution completed,
- whether partial consequence occurred,
- whether the original admissibility remains valid,
- whether context changed,
- whether authority remains active,
- whether policy still permits continuation,
- whether continuation requires a new determination.

Execution must not resume solely because it was previously approved.

Interrupt Test

Admissibility PASS

↓

Execution Begins

↓

Interrupt Occurs

↓

Execution State Inspection

↓

Runtime Revalidation

↙ ↘

Resume Prevent

↓

Updated Governance Receipt

Revalidation

Definition

Revalidation is the runtime process through which RiCo determines whether a prior governance determination remains valid under current conditions.

Revalidation may evaluate all governance components or only those affected by a known change.

However, the implementation must be able to justify the scope of revalidation.

Revalidation Triggers

Revalidation should occur when:

- execution is delayed,
- work enters or exits a queue,
- a new worker assumes responsibility,
- a retry begins,
- execution resumes after interruption,
- authority changes,
- evidence expires,
- context changes,
- policy changes,

- dependency health changes,
- execution scope expands,
- a validity window expires.

Revalidation Test

A revalidation test should deliberately change one governance input after initial approval.

Example:

t_1 — Request Approved

t_2 — Authority Revoked

t_3 — Worker Attempts Execution

t_4 — RiCo Revalidation

t_5 — Execution Denied

Expected result:

- no consequence,
- denial evidence,
- receipt update,
- replay confirms temporal invalidation.

Inspection

Definition

Inspection is the ability of an authorized verifier to examine the artifacts necessary to assess RiCo behavior.

Inspection must support analysis of:

- architecture,
- execution paths,

- governance inputs,
- admissibility decisions,
- enforcement outcomes,
- receipts,
- integrity proofs,
- replay results,
- test records.

Inspection does not require unrestricted access to all operational data.

It requires sufficient access to verify the governance claim.

Inspection Requirements

An implementation should make it possible to inspect:

- which request was evaluated,
- which governance inputs were used,
- which policy version applied,
- when the determination occurred,
- whether revalidation occurred,
- whether the consequence followed the result,
- whether the receipt remains intact,
- whether replay remains possible.

Evidence should be understandable without dependence upon undocumented application behavior.

Inspection Test

An independent verifier should be able to select a consequence and trace:

Consequence

↓

Execution

↓

REB Crossing

↓

Admissibility Decision

↓

Governance Inputs

↓

Governance Receipt

↓

Replay Result

Failure to complete this trace reveals a verification or evidence-correlation gap.

Falsification

Definition

Falsification is the deliberate attempt to prove that a RiCo conformance claim is false.

A testable architecture must expose conditions under which its claims would fail.

For example, the claim:

Every consequential execution traverses RiCo.

must be falsifiable through attempts to:

- invoke services directly,
- bypass gateways,
- exploit administrative interfaces,
- trigger workers without governance references,
- manipulate queues,

- abuse retries,
- resume interrupted execution,
- exploit failover paths.

If no conceivable test could disprove the claim, then the claim is not meaningfully testable.

Falsification Principle

RiCo verification must not ask only:

Can we demonstrate that the normal path works?

It must also ask:

Can we discover a path through which the architectural claim becomes false?

Falsification testing seeks counterexamples.

A single valid counterexample may disprove a universal conformance claim.

For example:

“All consequential paths are governed”

is disproven by discovering one consequential path that is not.

Falsification Requirements

A RiCo implementation should permit controlled testing of:

- bypass resistance,
- stale-state rejection,
- policy substitution,
- evidence manipulation,
- authority revocation,
- queue corruption,
- worker impersonation,
- retry abuse,

- receipt tampering,
- replay inconsistency.

Falsification is not hostile to the architecture.

It is how the architecture earns confidence.

Measurable Test Properties

RiCo testing should produce measurable results across several dimensions.

Coverage Rate

The percentage of identified consequential paths included in testing.

$$\frac{\text{Tested Consequence Paths}}{\text{Total Identified Consequence Paths}} \times 100$$

A critical consequential path left untested must be explicitly disclosed.

Governance Traversal Rate

The percentage of consequential executions with evidence of complete RiCo traversal.

$$\frac{\text{Executions with Valid Governance Traversal}}{\text{Total Consequential Executions Examined}} \times 100$$

For mandatory scope, the expected value is 100 percent.

Receipt Completion Rate

The percentage of admissibility determinations producing complete Governance Receipts.

Replay Success Rate

The percentage of selected receipts for which legitimacy can be conclusively reconstructed.

Denial Enforcement Rate

The percentage of denied requests for which no prohibited consequence occurred.

For a conformant critical path, the expected value is 100 percent.

Revalidation Compliance Rate

The percentage of revalidation-triggering events that resulted in the required runtime governance evaluation.

Evidence Correlation Rate

The percentage of sampled consequences that can be traced completely from request through replay.

Canonical Test Record

Each RiCo test should generate a structured record containing:

- test identifier,
- implementation version,
- environment,
- requirement under test,
- consequence path,
- initial governance state,
- test input,
- change or fault introduced,
- expected governance outcome,
- expected execution outcome,
- actual governance outcome,
- actual execution outcome,

- receipt identifier,
- replay result,
- evidence references,
- PASS, FAIL, or INDETERMINATE,
- verifier identity,
- test timestamp.

This record becomes part of the Conformance Evidence Profile.

Minimum Test Suite

A RiCo conformance assessment should include, at minimum:

Normal Approval Test

Valid request produces admissibility PASS and authorized consequence.

Normal Denial Test

Invalid request produces FAIL and no consequence.

Worker Path Test

Background or delegated execution cannot occur without governance validation.

Queue Delay Test

Queued execution is revalidated after time has elapsed.

Retry Test

A new attempt does not rely blindly on the original decision.

Interrupt and Resume Test

Interrupted execution cannot resume without appropriate revalidation.

Authority Revocation Test

Previously approved execution is denied after authority becomes invalid.

Evidence Expiration Test

Previously sufficient evidence is rejected after freshness requirements fail.

Policy Change Test

Execution reflects the policy effective at runtime rather than approval time.

Context Change Test

Changed operational conditions trigger new admissibility evaluation.

Bypass Test

Direct or privileged invocation cannot produce consequence outside RiCo.

Receipt Integrity Test

Modification or deletion is detectable.

Replay Test

Governance legitimacy can be independently reconstructed.

Falsification Test

The verifier actively attempts to discover a counterexample to each universal conformance claim.

Architectural Characteristics

A Testable RiCo Architecture is:

- Observable.
- Deterministic where required.
- Measurable.
- Repeatable.
- Evidence-producing.
- Adversarially testable.
- Independently inspectable.
- Falsifiable.
- Replay-enabled.
- Implementation-neutral.

These characteristics convert governance from an assertion into an engineering discipline.

Relationship to the Conformance Evidence Profile

The Conformance Evidence Profile defines what evidence must exist.

Testable Architecture defines how that evidence is produced and evaluated.

RiCo Requirement

↓

Test Scenario

↓

Observed Behavior

↓

PASS / FAIL / INDETERMINATE

↓

Governance Receipt

↓

Replay

↓

Conformance Evidence

Without testing, the evidence profile may consist only of documentation.

Without an evidence profile, test results may remain isolated technical artifacts.

Together they form the basis for RiCo conformance.

Summary

Testable Architecture is the requirement that every material RiCo claim be expressed as an observable, measurable, repeatable, and falsifiable condition. A RiCo implementation must demonstrate through PASS, FAIL, replay, worker-path, queue, retry, interrupt, revalidation,

inspection, and falsification tests that runtime governance remains binding across the complete consequential lifecycle.

Governing Principle

The Principle of Architectural Falsifiability

Every mandatory RiCo claim must identify the evidence that would confirm it, the test that would challenge it, and the observable condition that would prove it false. An architectural claim that cannot be measured, inspected, replayed, or falsified shall not be accepted as evidence of conformance.

RiCo does not ask an implementation to claim that governance worked. It requires the implementation to expose how that claim can be tested—and how it can be proven false.

Part V — Research Direction

17. Future Research

Introduction

Runtime Integrity Control (RiCo) establishes a reference architecture for runtime governance, admissibility determination, enforcement, evidentiary preservation, and constitutional verification.

This Technical Architecture Package defines the foundational principles required to govern consequential execution within a single runtime governance domain.

However, the rapid evolution of distributed computing, autonomous agents, adaptive intelligence, and cross-organizational decision systems presents governance challenges extending beyond the scope of the current architecture.

Future research should therefore build upon the principles established by RiCo while preserving its central objective:

Every consequential execution must remain demonstrably legitimate at the moment consequence occurs.

This chapter identifies research directions intended to extend the architecture without altering its constitutional foundations.

Research Philosophy

RiCo is intentionally designed as a constitutional architecture rather than an implementation bound to today's technologies.

Its principles are expected to remain stable while implementation models evolve.

Future research should preserve the architectural invariants established throughout this specification, including:

- Runtime Admissibility
- Runtime Execution Boundary
- Consequence Path Coverage
- Non-Bypassability
- Temporal Validity

- Governance Receipts
- Constitutional Replay

Extensions should strengthen these principles rather than replace them.

17.1 Distributed RiCo

Current RiCo assumes governance can be evaluated within a single logical governance domain.

Future distributed systems may execute consequential workflows across multiple infrastructures, organizations, jurisdictions, or sovereign governance domains.

Research questions include:

- How can Runtime Execution Boundaries be coordinated across distributed systems?
- Can Governance Receipts maintain integrity across organizational boundaries?
- How is admissibility preserved when governance responsibility is shared?
- How can constitutional replay reconstruct legitimacy across multiple governance domains?
- How should trust relationships between independent RiCo domains be established and verified?

Distributed RiCo represents a natural evolution toward federated runtime governance.

17.2 Agent Orchestration

Increasingly, consequential execution is performed not by individual applications but by cooperating autonomous agents.

Agent orchestration introduces governance challenges beyond traditional request-response architectures.

Research topics include:

- governance of delegated agent authority,
- orchestration-level admissibility,
- agent-to-agent governance delegation,

- dynamic task decomposition,
- autonomous planning under constitutional constraints,
- governance inheritance across agent workflows,
- evidentiary preservation throughout autonomous collaboration.

Future RiCo research should determine how governance remains continuous as multiple autonomous agents coordinate consequential execution.

17.3 Adaptive Governance

Modern operational environments continuously evolve.

Policies may change dynamically.

Operational risk may fluctuate.

Mission objectives may shift.

Future RiCo implementations may require governance capable of adapting while preserving constitutional consistency.

Research questions include:

- adaptive policy selection,
- dynamic governance thresholds,
- context-aware governance strategies,
- runtime governance optimization,
- policy evolution without constitutional drift,
- bounded machine-assisted governance adaptation.

Adaptive governance must remain constrained by constitutional principles so that optimization never replaces legitimacy.

17.4 Multi-Agent Admissibility

Current RiCo evaluates admissibility primarily for a single governed execution request.

Future autonomous ecosystems may require admissibility to emerge from coordinated decisions among multiple independent agents.

Research topics include:

- collective authority,
- shared evidence models,
- distributed context acquisition,
- coordinated policy evaluation,
- consensus admissibility,
- conflicting governance decisions,
- multi-agent Governance Receipts,
- replay of collaborative governance decisions.

The architectural challenge is determining whether legitimacy can be established collectively without sacrificing accountability for individual participants.

17.5 Cross-Domain Authority

Authority increasingly spans organizational, governmental, contractual, and technical domains.

Future governance architectures may require authority to be established across independently managed trust environments.

Research areas include:

- federated authority,
- delegated constitutional authority,
- inter-organizational governance,
- sovereign governance interoperability,
- jurisdiction-aware admissibility,
- authority conflict resolution,
- authority portability,
- authority revocation across distributed ecosystems.

Future RiCo work should explore how authority continuity can be preserved when governance extends beyond a single administrative boundary.

17.6 Governance Interoperability

As runtime governance matures, independent governance architectures may need to cooperate.

Research should examine:

- interoperable Governance Receipts,
- standardized replay interfaces,
- portable admissibility evidence,
- shared governance semantics,
- constitutional compatibility assessment,
- governance translation between heterogeneous systems.

The long-term objective is to enable legitimate governance across diverse implementations while preserving implementation independence.

17.7 Formal Verification

RiCo is intentionally written as an architectural specification.

Future work may investigate formal mathematical verification of its properties.

Potential directions include:

- formal models of admissibility,
- state-machine verification,
- temporal logic for runtime legitimacy,
- proof of Non-Bypassability,
- verification of Consequence Path Coverage,
- correctness proofs for Constitutional Replay,
- model checking of runtime governance behavior.

Formal verification could strengthen confidence in implementations operating within highly consequential environments.

Architectural Continuity

Future extensions should preserve the constitutional foundations established throughout TAP-001.

Regardless of technological evolution, RiCo maintains several enduring architectural invariants:

- governance precedes consequence,
- admissibility is determined at runtime,
- legitimacy must remain temporally valid,
- every consequential path remains governed,
- no execution bypasses governance,
- every determination produces governance evidence,
- legitimacy remains independently reconstructable.

These principles define the constitutional identity of RiCo and should remain stable as the architecture evolves.

Research Roadmap

Future work may naturally progress through successive architectural phases:

RiCo v1

Single Runtime Governance

|

▼

Distributed RiCo

Multiple Governance Domains

|

▼

Agent-Oriented RiCo

Autonomous Collaboration

|

▼

Adaptive RiCo

Dynamic Constitutional Governance

|

▼

Federated RiCo

Cross-Domain Authority

|

▼

Constitutional Governance Ecosystems

Each phase extends operational capability while preserving the same constitutional principles defined in this specification.

Summary

Future research should extend RiCo beyond single-domain runtime governance toward distributed, adaptive, and multi-agent environments while preserving its constitutional foundations. Regardless of implementation advances, future RiCo architectures should continue to ensure that every consequential execution remains admissible, governed, evidentially preserved, and independently reconstructable at the moment consequence occurs.

RiCo is not presented as the final form of runtime governance. It is presented as a constitutional foundation upon which future runtime governance architectures can be rigorously designed, tested, and verified.

Closing Perspective

RiCo does not attempt to make AI systems intelligent. It establishes an architectural foundation for governing consequential execution at runtime. As intelligent systems become increasingly autonomous, distributed, and adaptive, runtime legitimacy becomes as important as runtime correctness.