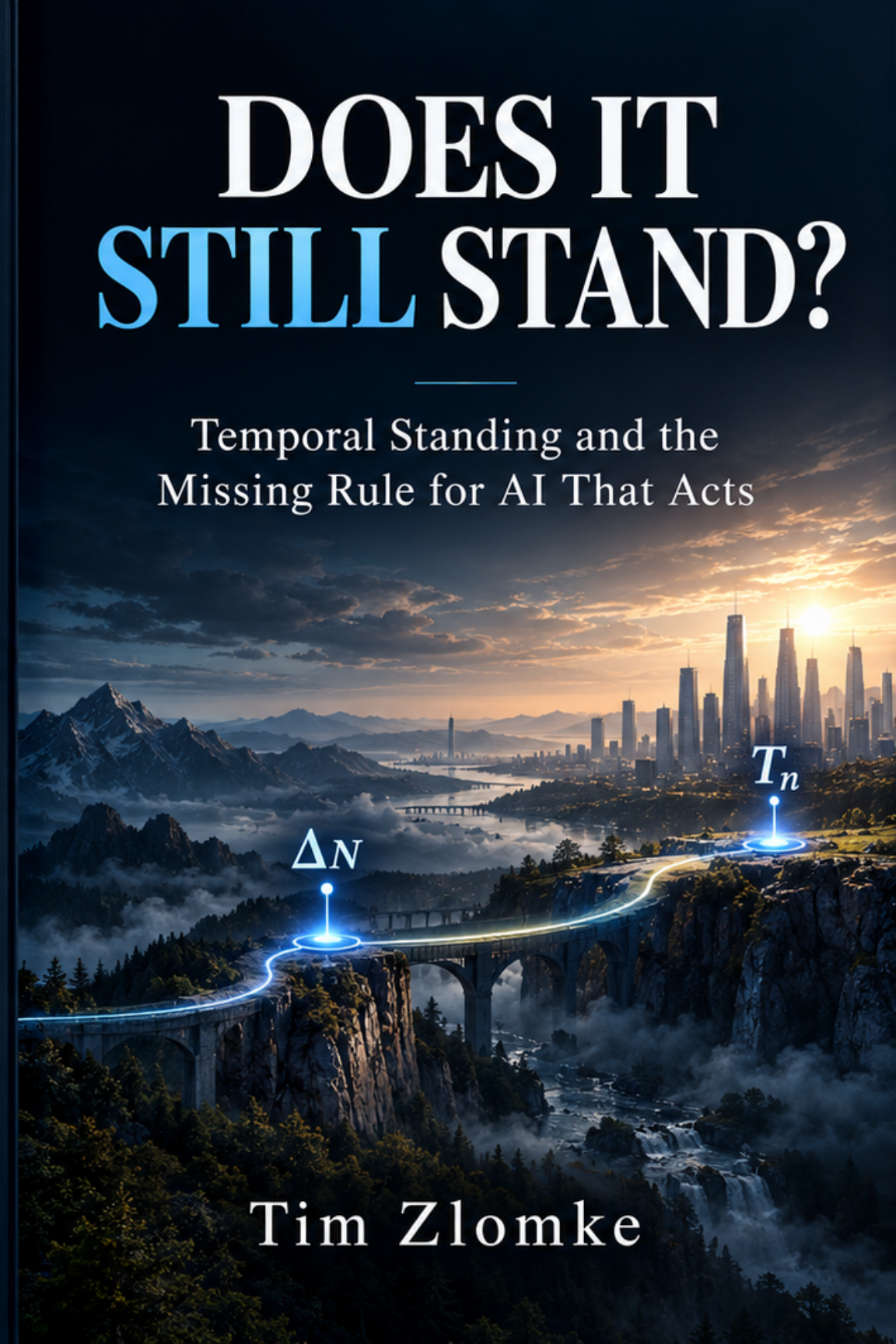
DOES IT STILL STAND?

Temporal Standing and the Missing Rule for AI That Acts



Tim Zlomke

DOES IT STILL STAND?

Temporal Standing and the Missing Rule for AI That Acts

Tim Zlomke

An AI can be correct about what was true and still be wrong to act now.

Copyright

Copyright © 2026 Timothy E. Zlomke. All rights reserved.
First edition, 2026.
No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the copyright holder, except for brief quotations used in reviews, scholarship, or other uses permitted by law.

This book is for informational and educational purposes. It does not provide legal, medical, financial, employment, cybersecurity, or other professional advice. Examples in regulated or high-stakes domains are used to illustrate governance principles and should not be treated as instructions for a particular real-world case.

Unless a passage is specifically identified as a documented event, scenarios in this book should be understood as illustrative or composite examples designed to test the framework.

"Temporal Standing" is used here as a governance term. It is not a claim about legal standing as that term is used by courts.

Dedication

To my loving wife, Jema, for the patience and understanding she gave through the long hours this work demanded - and for the real insight she brought to its development. Some of these ideas became better because you challenged them, lived beside them, and gave me the space to keep building.

To my mom, Joy, for the encouragement, support, and steady belief that has followed me through every chapter of my life.

And, with all my heart, to the memory of my brother, David.

Dave was my brother, my business partner, and for most of my life never very far away. He was a giant of a man in every sense that mattered: strong, gentle, loyal, generous, and almost impossible to shake. He taught me more about quiet decency than any theory ever could.

His loss changed the shape of my life. This book was built in the years that followed, through long hours, sacrifice, grief, stubbornness, and the decision to keep moving forward.

For Jema. For Mom. And for Dave.
I carry you with me in this work.

From the Author

I did not set out to write a book about Temporal Standing. I set out to solve a problem that would not leave me alone.

For the last several years I have spent an unreasonable number of hours building AI systems, breaking them, rebuilding them, arguing with them, arguing with myself about them, and occasionally discovering that the machine had found a brand-new way to be technically correct and operationally wrong. That last category is more interesting than it sounds.

As AI moved from answering questions to taking actions, I kept returning to one concern. A system can be fluent, capable, well-intentioned, and historically accurate - and still be wrong to act now. An approval can age. Consent can be withdrawn. A role can change. A policy can be superseded. A price can move. A person can simply change their mind. The record of what was true yesterday may remain perfectly accurate as history while no longer being enough authority for today's consequence.

That sounds obvious when written in a paragraph. It is much less obvious when the instruction is sitting in a queue, the workflow is already moving, the model is confident, the API is available, and everything in the system has been designed to reward completion. Software is very good at remembering that somebody once said yes. It is not naturally gifted at asking whether the yes still stands.

The framework in this book grew out of that gap. Not from one whiteboard session, and not from a desire to invent a new phrase, but from years of building, testing, failing, refining, and repeatedly arriving at the same question at the edge of execution: Does it still stand?

The notation became simple: To -> Delta N -> Tn. What stood then. What changed. What stands now for this exact consequence. The simplicity is intentional. A governance principle that cannot survive contact with a tired engineer, a production incident, or a deadline is not much of a governance principle.

I also want to be precise about what I am - and am not - claiming. Security engineering, access-control research, privacy law, distributed systems, consent frameworks, and governance disciplines have studied revocation, changing authorization, contextual permissions, continuous evaluation, and related problems for decades. This book does not pretend

otherwise. Temporal Standing is my attempt to compose those realities around a specific problem that becomes urgent when AI systems act: whether the full governing basis for a particular consequence still has present standing at the moment that consequence is about to occur.

There is a practical reason I care about that distinction. We are quickly building systems in which cognition is cheap and action is easy. The distance between 'the model thinks this is a good idea' and 'the system just did it' is shrinking. I believe that distance needs a boundary - one where present authority is determined rather than assumed, uncertainty is allowed to remain uncertainty, and a denial actually prevents the consequence instead of merely producing an impressive audit log after the fact.

This work has also had a human cost. The hours behind these pages were real hours, taken from evenings, weekends, family time, sleep, and the easier choice of leaving the problem for someone else. I was able to keep going because the people closest to me made room for the work, challenged it, encouraged it, and reminded me why building systems worthy of trust matters in the first place.

If this book does its job, you will not finish it merely knowing a new term. You will begin to notice a question hiding inside every consequential AI workflow: This may have been authorized then - but what has changed, and does it still stand now?

That is the question I have spent the last several years trying to make operational.

Models propose. Governance decides. Truth survives.

- *Tim Zlomke*

A Note on Scope and Prior Work

This book began with a practical problem: AI systems were becoming increasingly capable of doing things in the world, while many of the controls around them still treated authority as though it were a static fact.

The argument that follows is deliberately narrower than a claim that changing authorization, revocation, contextual access control, or continuous evaluation are new ideas. They are not. Security engineering, access-control research, privacy law, distributed systems, and governance have developed important pieces of this problem for decades.

The contribution proposed here is a cross-domain discipline for AI-mediated consequences: identify the heterogeneous grounds that support an exact consequence; determine what has changed between prior authority and present execution; preserve an explicit unresolved state when current standing cannot be established; bind the determination to the material consequence; and make the execution path obey it.

The notation is simple:

$$T_0 \rightarrow \Delta N \rightarrow T_n$$

What was authorized or true then. What changed. Whether it still stands now for this exact consequence.

The humor in these pages is aimed at systems, process, and governance theater - never at the people who bear the harm when those systems fail.

Contents

Chapter 1 - The Dangerous Assumption: If It Was True, It Is Still True.....1
Chapter 2 - When Good Instructions Become Bad Actions.....5
Chapter 3 - The Gap Between Answering and Acting.....10
Chapter 4 - Why Traditional AI Safety Is Not Enough.....15
Chapter 5 - $T_0 \rightarrow \Delta N \rightarrow T_n$.....23
Chapter 6 - Scope Matters: Authority Is Never Generic.....30
Chapter 7 - Preserved, Defeated, Unresolved.....35
Chapter 8 - Evidence Is Not Authority.....41
Chapter 9 - Consent That Changes.....48
Chapter 10 - AI Agents and Tool Use.....53
Chapter 11 - High-Stakes Domains.....60
Chapter 12 - Enterprise Operations.....65
Chapter 13 - The Human Case.....70
Chapter 14 - The Consequence Boundary.....75
Chapter 15 - Fresh Determination, Not Permanent Permission.....81
Chapter 16 - Building the Standing Record.....86
Chapter 17 - Designing for "I Don't Know Yet".....91
Chapter 18 - Governance Without Theater.....96
Chapter 19 - From Intelligent Systems to Accountable Systems.....104
Chapter 20 - A Social Contract for AI That Acts.....108
Chapter 21 - The Question Every AI System Must Ask.....112
Appendix A - Temporal Standing and Its Intellectual Neighbors.....118
Appendix B - A Reference Engineering Pattern.....121
Appendix C - Case Studies.....125
Appendix D - Validating Temporal Standing.....131
Primary Sources and Comparative References.....134
About the Author.....137

PART I

THE PROBLEM: AI LIVES IN TIME

Chapter 1 - The Dangerous Assumption: If It Was True, It Is Still True

The first generation of widely used AI systems trained us to ask a familiar question: is the answer right?

That question still matters. A false answer can mislead a student, a doctor, an investor, an engineer, or a customer. But the arrival of AI systems that can send, buy, book, change, approve, delete, escalate, notify, deploy, and transact creates another question that is both simpler and more difficult:

Reality check

Software has a remarkable ability to preserve yesterday forever. Somewhere, right now, a database is faithfully protecting a fact that stopped being useful three policy revisions ago. The record is not lying. That is what makes the problem interesting.

Historical accuracy and present authority are different jobs. A system can perform the first perfectly and still fail the second.

Even if the system is right about what was true, does it still have grounds to act now?

Imagine a customer who agrees on Monday to receive a series of promotional messages. The campaign is created, the audience is selected, and the messages are scheduled. On Tuesday the customer changes the preference. On Wednesday an automated agent reaches the scheduled step. The original record is accurate. The customer really did authorize the communication on Monday. The system may even retrieve that record perfectly.

Yet a perfectly accurate description of Monday can still support the wrong action on Wednesday.

This is the problem at the center of this book.

We build information systems by storing facts. A database is useful precisely because a fact recorded yesterday can be retrieved tomorrow. We build workflow systems by storing instructions. A scheduled job is useful because an instruction issued at one moment can be executed later. We build authorization systems by storing permissions. Delegation would be impossible if every permitted act required the original authority to be physically present at the instant of execution.

Persistence is a feature.

But persistence creates a trap: the stored thing and the standing of the stored thing are not identical.

A preference can remain in a database after the person changes it. An approval can remain visible after its scope has expired. A policy can remain indexed after a new policy supersedes it. An employee can remain named in a workflow after the employee has lost the relevant role. A purchase instruction can remain queued after the price changes. A recommendation

can remain reasonable in form after the conditions that supported it are gone.

The historical record may remain true as history. What changes is what that history authorizes now.

This distinction is easy to miss because most software has been built around relatively stable, explicit transactions. A user clicks a button. A request is evaluated. A response happens. The interval between intent and consequence is often short enough that engineers can treat the world as though it has not materially changed.

AI agents stretch that interval.

They plan. They wait. They call tools. They delegate. They retry. They retrieve old context. They act on behalf of people who may not be present. They continue workflows across minutes, hours, days, or longer. They can be given broad goals rather than one exact command. The more capable the system becomes, the more opportunities there are for reality to change between instruction and consequence.

That makes time part of governance.

Consider five ordinary cases.

A manager approves an agent to purchase replacement equipment up to a certain amount. The agent identifies a product, but checkout occurs after the price has risen above the approved ceiling.

A traveler asks an agent to reserve a hotel for a particular weekend. Before the booking occurs, the traveler changes the trip dates in another conversation.

A company authorizes an employee to approve refunds. The employee changes roles, but an automated workflow still carries the old delegation.

A user agrees to share data for one purpose. Months later, a system proposes to use the same data for a different downstream action and treats the old consent as generic permission.

An incident-response agent is told to isolate a server if a particular condition is observed. The condition is real when the workflow begins, but by the time the agent reaches the isolation step, an operator has already remediated the problem and changed the system state.

None of these cases requires an evil model. None requires hallucination. The model can be competent, the database intact, the record authentic, and the system still wrong to act.

The failure is temporal.

A prior state has been mistaken for present standing.

Throughout this book, we will use a simple notation for this problem:

$\mathbf{T_0 \rightarrow \Delta N \rightarrow T_n}$

At $\mathbf{T_0}$, there is some prior authoritative state: an instruction, permission, consent, approval, fact, condition, role, policy, or combination of these that supports an action.

Then something happens. We call the relevant intervening change $\mathbf{\Delta N}$. It may be a revocation, expiry, changed circumstance, contradictory evidence, role change, updated preference, policy change, changed recipient, changed price, or simply a gap that makes the current state uncertain.

At T_n, the system is about to cause a consequence. The question is not whether T_0 existed. The question is whether the grounds established at T_0 still stand for this exact action now.

There are three basic answers.

Standing may be **preserved**. Nothing relevant has defeated or materially altered the authority, and the action remains within scope.

Standing may be **defeated**. A relevant change means the previous authority no longer supports the proposed consequence.

Or standing may be **unresolved**. The system cannot establish enough current authoritative information to justify proceeding.

This third state matters more than it first appears. Computer systems often prefer binary answers. Allow or deny. True or false. Approved or rejected. But the real world frequently offers incomplete evidence. A system may know that the original permission existed while being unable to determine whether it remains current. A policy service may be unavailable. A revocation source may be stale. Two records may conflict. Identity may be uncertain. A human decision may be required.

An acting AI cannot safely turn that uncertainty into permission merely because action is convenient.

This is not a claim that every fact must be reverified before every action. That would make useful automation impossible. Nor is it a claim that any change automatically cancels prior authority. A weather change need not affect permission to send an invoice. A user changing a profile photograph need not defeat a previously authorized recurring payment. The issue is relevance.

Temporal Standing asks whether there has been a change, contradiction, expiry, uncertainty, or scope shift that matters to the consequence under review.

That means authority is always more specific than phrases such as "the user approved it." Approved what? For which purpose? Using which data? Through which system? For how long? Under which conditions? With what limits? For which consequence?

The more consequential the action, the more expensive vague authority becomes.

This is why the central argument of this book is not simply that AI systems should "double-check." Double-checking is a useful instinct but an imprecise architecture. A model can double-check the wrong source. It can confirm that an old approval exists and still miss a revocation. It can ask itself whether an action seems reasonable and mistake its own confidence for authority.

The deeper requirement is this:

Past authority is evidence of prior standing, not proof of present standing.

Before an AI system crosses a consequential boundary, it needs a current determination that the grounds for that exact consequence are preserved. If standing is defeated, the action should not occur. If standing is unresolved, the system should not manufacture certainty.

The challenge is not merely remembering the past.
It is governing the transition from the past into the present.
That is the work of Temporal Standing.

Four different clocks

The easiest way to see the problem is to notice that an acting system rarely lives on one clock.

There is the clock of **record creation**: when a fact, instruction, preference, or permission entered the system.

There is the clock of **retrieval**: when an agent fetched that record into context.

There is the clock of **determination**: when the system decided that a proposed action was admissible.

And there is the clock of **consequence**: when the external effect actually became real.

Traditional software often compresses these clocks so tightly that we forget they are different. A user clicks "buy," the server checks a session, and the transaction is submitted within a narrow interval. Agentic systems stretch them apart. A record can be created on Monday, retrieved on Tuesday, interpreted on Wednesday morning, approved on Wednesday afternoon, and executed Thursday after a retry.

Every gap creates room for ΔN.

That does not make delayed execution inherently unsafe. Banks schedule payments. Enterprise systems queue jobs. Cloud systems retry work after partial failure. The lesson is not "never defer." The lesson is that deferral converts time into a governing variable.

This is especially important because AI systems are good at reconstructing narrative continuity. Given a long conversation, a model can make yesterday's intent feel psychologically continuous with today's action. But continuity of story is not the same as continuity of authority. A coherent narrative can conceal a broken standing chain.

The safer design discipline is to ask, at each clock boundary, what was merely carried forward and what was re-established. Was the system carrying a historical record? A cached policy result? A human approval? A current entitlement? A price quote? A consent state? A role assignment? Each has different rules for staleness.

A system that can answer these questions does not need to distrust memory. It needs to stop confusing memory with present authority.

Chapter 2 - When Good Instructions Become Bad Actions

We tend to evaluate instructions by looking at the moment they are given.

Was the person authorized? Was the request clear? Was the proposed action reasonable? Were the relevant conditions satisfied?

What could possibly go wrong?

A good instruction can age badly. "Keep following up until they respond" sounds efficient until the customer opts out, changes channels, opens a dispute, and the agent decides perseverance is a core value. At that point the workflow is no longer diligent. It is a very polite robot refusing to take a hint.

The instruction did not become false. Its standing changed.

If the answer is yes, we feel that the problem is solved.

But an instruction does not travel through time unchanged merely because its words remain unchanged.

Suppose you tell an assistant at 9:00 a.m., “If the supplier can deliver by Friday and the total is under ten thousand dollars, place the order.” At 9:05 the conditions hold. At 10:30 the supplier changes the delivery date. At noon the agent executes the order using the instruction exactly as written.

Did the agent follow the instruction?

Textually, perhaps. Governedly, no. The instruction contained conditions, and those conditions no longer held.

The error becomes more subtle when the condition is not written into the sentence.

A customer says, “You can text me updates.” The phrase sounds durable. Yet almost everyone understands that the customer can later say, “Stop texting me.” The original instruction does not become false. It becomes historically bounded.

A manager says, “Maria can approve these requests.” Later Maria leaves the role. The sentence was once true. The authority it described may no longer be.

A user says, “Book the cheapest nonstop option.” Before booking, the user adds a new requirement: “Nothing before 9 a.m.” A system that executes the old plan without incorporating the new condition has not failed because it misunderstood “cheapest.” It has failed because it treated the earlier state as complete after the governing state changed.

This pattern appears anywhere there is a delay between decision and consequence.

The delay may be seconds. A payment amount can change between quote and confirmation.

It may be minutes. An operator can revoke a deployment while an automated pipeline is preparing to release.

It may be days. A scheduled communication can remain queued after a preference change.

It may be months. A broad enterprise delegation can outlive the role, purpose, or project that originally justified it.

The practical insight is that instructions live inside an environment. The words are only one component.

At T_0, the instruction is embedded in a state that includes identity, scope, facts, constraints, purpose, policy, resources, and assumptions. Some of those elements are explicit. Others are inherited from the surrounding system.

When the system later acts at T_n, it is tempting to retrieve only the instruction. But the instruction alone is an incomplete representation of the authority that once supported the action.

A useful way to think about this is the **authority envelope**.

An authority envelope contains at least these questions:

Who authorized the action?

What exact action was contemplated?

Which system or agent may perform it?

Which data may be used?

For what purpose?

For what duration?

Under which conditions?

What consequence was within scope?

The envelope need not be a literal form shown to a user. It is a design concept. It reminds us that authorization is rarely generic.

When AI systems are given natural-language goals, this becomes especially important. Natural language is powerful because it compresses intent. But compression can hide scope.

"Handle my travel."

"Take care of these customers."

"Fix the account."

"Keep the system healthy."

"Use your judgment."

A human assistant interprets phrases like these through an enormous amount of social, organizational, and situational context. We understand that "handle my travel" does not mean sell the traveler's house to finance a better flight. We understand that "fix the account" does not automatically mean erase every disputed record. We understand that delegated judgment remains bounded by role, norms, law, policy, and changing circumstances.

An AI agent needs those boundaries made operational.

Otherwise, a good instruction can become a bad action in at least four ways.

First, **the facts can change**. The price, availability, account state, medical condition, system condition, identity, risk level, or other relevant feature may no longer match the original state.

Second, **the authority can change**. Consent can be withdrawn, a role can end, a delegation can expire, or a policy can revoke a previously permitted action.

Third, **the scope can change**. The target, amount, recipient, data, purpose, or tool may differ from what the original permission contemplated.

Fourth, **the system's knowledge of the present can become insufficient**. Nothing may have been explicitly revoked, but the evidence required to establish continued standing may be unavailable or contradictory.

These categories overlap, and that is fine. Temporal Standing is not a taxonomy contest. Its purpose is to force a present determination before consequence.

The distinction also helps explain why "the user asked for it" is often too weak a justification for autonomous action.

Users ask for things at moments. Systems act at moments. If those moments are separated, the system needs a way to carry not only the instruction but also the boundaries and standing of the instruction forward.

This does not mean interrupting the user every time. A well-designed system can preserve standing through explicit duration, scope, change signals, freshness windows, revocation mechanisms, and authoritative sources. The goal is not friction. The goal is to make legitimate continuity possible while preventing illegitimate persistence.

Imagine two systems.

System A stores a command: "Send the renewal reminder in thirty days." Thirty days later it sends.

System B stores the same command plus the conditions that matter: recipient identity, account status, communication preference, purpose, allowed channel, and a requirement to determine current standing immediately before sending. Thirty days later it checks the current authoritative state. If the customer is still eligible and has not revoked the channel, the message proceeds. If the customer opted out, it does not. If the preference service is unavailable and the system cannot establish whether the channel remains authorized, it holds.

Both systems automate.

Only one treats time as part of governance.

This distinction becomes increasingly important as agents gain persistence. An agent that completes a single tool call in a tightly bounded session may operate inside a small temporal window. An agent that manages a process over weeks is effectively maintaining a relationship with a changing world.

The longer the interval, the more dangerous it is to confuse a saved instruction with a current mandate.

There is another reason this problem matters: retries.

Modern systems retry constantly. Networks fail. APIs time out. providers return ambiguous responses. queues redeliver messages. workers crash after partially completing tasks. Retry logic is essential to reliability.

But reliability engineering can create governance failures when it assumes that a retry is merely a technical continuation of the original action.

Suppose an action was properly authorized at 1:00 p.m. A provider times out. The system cannot tell whether the operation completed. At 1:30 a retry worker picks up the job. In those thirty minutes, the user revoked the action.

The retry is not automatically entitled to inherit the original permit.

The transport history matters, but so does current standing.

This reveals a recurring principle: **execution lineage and authority lineage are related but not identical**.

A system needs to know what happened technically. Did the provider accept the request? Did the tool respond? Did the transaction complete? But it also needs to know whether the grounds for any new external consequence still stand.

The difficult cases are precisely those where the software appears to be behaving correctly. The queue is healthy. The retry succeeded. The workflow followed its configuration. The model interpreted the instruction as expected.

And yet the action should not have happened.

That is why Temporal Standing is not primarily a model-quality problem.

It is a system-governance problem.

Good instructions do not remain good because we saved them faithfully.

They remain good only while the grounds that give them standing remain good for the consequence at hand.

The queue is a governance surface

Queues are normally discussed as reliability infrastructure. They absorb bursts, smooth traffic, decouple services, and let work survive temporary failures. Those are genuine benefits.

But a queue also preserves intent across time.

That makes it a governance surface.

Suppose an AI agent creates 2,000 outbound communications at 3:00 p.m. The records are valid when created. A worker sends only 200 before a provider outage. The remaining 1,800 sit in a queue overnight. During that interval, recipients may opt out, accounts may be suppressed, campaigns may be cancelled, policies may change, and some contacts may become active customers for whom the original outreach is no longer appropriate.

The engineering temptation is to treat the queued item as already authorized work. After all, authorization happened before enqueueing.

Temporal Standing asks a different question: **what exactly was authorized at enqueue time?**

Perhaps the system authorized preparation. Perhaps it authorized eligibility under then-current facts. Perhaps it authorized a short-lived execution window. What it should not do accidentally is transform a historical eligibility decision into permanent authority to create the consequence whenever a worker eventually reaches the item.

The same issue appears in scheduled infrastructure changes, deferred refunds, document submissions, follow-up messages, batch data exports, and long-running procurement workflows.

A useful pattern is to separate a queued **proposal** from an executable **permit**. The proposal can survive for operational convenience. The permit can be short-lived, bound to a payload, or re-established at the consequence boundary. That design preserves reliable workflow mechanics without granting the queue metaphysical authority over the future.

This also clarifies cancellation. If a user or operator withdraws an action, the system should not have to erase history to stop the consequence. The queued proposal can remain as evidence of what was once intended while its standing becomes defeated.

That is a recurring theme in this book: good governance rarely requires pretending the past did not happen. It requires preventing the past from silently governing the present after its grounds have changed.

Chapter 3 - The Gap Between Answering and Acting

For most of the public history of generative AI, the visible product was an answer.

You typed a question. The model generated text. The output might be brilliant, mediocre, biased, fabricated, useful, or wrong, but the consequence usually depended on what a human did next.

Translation into normal English

Writing the email and sending the email are not the same event. Drafting a refund and issuing it are not the same event. Recommending a server change and taking production down are definitely not the same event, no matter how confident the model sounds while doing it.

Cognition can be broad. Consequence should be bounded.

That arrangement created a natural buffer.

The model proposed. A person interpreted. A person copied, clicked, approved, sent, purchased, deployed, or ignored.

The buffer was never perfect. Advice can influence behavior. A persuasive falsehood can cause real harm. But the model generally did not possess the final causal link to the world.

Agents change that architecture.

An acting system can call an API, send an email, update a database, create a user, modify a permission, purchase a product, book a reservation, issue a notification, change a configuration, trigger a workflow, or instruct another agent.

The difference is not rhetorical. It is causal.

A generated sentence is a proposal. A completed tool call is an event in the world.

That gap between **cognition** and **consequence** is where governance has to become concrete.

Cognition includes interpreting a request, retrieving information, reasoning over options, planning, drafting, classifying, ranking, or proposing an action. These processes can be highly capable without possessing authority to create the consequence they describe.

Consequence begins when the system changes external state: another person is contacted, money moves, access changes, a record is altered, a service is triggered, a reservation is created, a machine state changes, or another process is causally launched.

The distinction is easy to state and easy to blur.

Suppose a model generates, “I recommend issuing a $500 refund.” That is cognition.

Suppose a tool call submits the refund. That is consequence.

The same language model may participate in both steps, but the governance requirements are not the same.

The model can be excellent at deciding what appears reasonable while lacking the authority to decide what may be executed.

This is the first architectural separation Temporal Standing depends on: **a system's capacity to propose an action must not be confused with authority to cause it.**

There are many ways to implement that separation. The exact technology is less important than the property.

A model may produce an execution proposal. A governance layer may evaluate the present authority and evidence. A boundary component may permit or deny transport. An external connector may perform the action only after a valid, bounded permit exists.

The sequence might look like this:

Interpret -> Propose -> Determine Standing -> Permit or Deny -> Execute

The order matters.

If the system executes first and evaluates later, governance becomes retrospective. Retrospective audit has value. It helps organizations understand failures, reconstruct decisions, improve controls, and assign accountability. But an audit cannot un-send an email, un-disclose private data, un-place a trade, or un-delete a record with perfect fidelity.

Where an action can be governed before it happens, the relevant control belongs before consequence.

This point also clarifies a common debate about whether the model itself can "be the governor."

A model can contribute to governance. It can extract facts, identify possible changes, classify a request, explain policy, flag ambiguity, and generate evidence summaries. In some systems it may even be one component in a broader determination process.

But model judgment and execution authority should not collapse into one undifferentiated step.

Why?

Because the model's conclusion is not the same thing as the external grounds that make the action legitimate.

A model can say, "The user probably still wants this." That is an inference.

It can say, "The prior approval appears to cover the request." That is an interpretation.

It can say, "I am 98 percent confident this is permitted." That is a confidence statement.

None of these statements, by themselves, create authority.

The authority comes from the relevant governing sources: the user's current consent, a valid delegation, a current policy, a valid access grant, a binding contract term, a present account state, a current institutional decision, or whatever source legitimately governs the consequence.

This is the second architectural separation: **evidence and interpretation may inform authority, but they do not become authority merely because a model processed them.**

Once we see this, the placement of Temporal Standing becomes clearer.

The standing determination belongs as close as practicable to the consequence boundary.

Not because every upstream step is unimportant. Planning can expose sensitive data. Retrieval can itself have access implications. Intermediate actions can be consequential. Each meaningful consequence may require its own boundary.

But for any specific external effect, the crucial question is whether the authorization and governing evidence are current when that effect is about to occur.

This is why a permit should be bounded.

Suppose a governance system determines at 2:00 p.m. that an agent may send an email to a particular recipient, using a particular approved template, for a particular purpose. That determination should not silently become permission at 4:00 p.m. to send a different message to three additional recipients using newly retrieved personal data.

The later action is a new consequence surface.

If the payload changes materially, the recipient changes, the tool changes, the purpose changes, the scope changes, or enough time passes for relevant standing to become uncertain, the system needs a new determination.

This may sound expensive. In practice, it can often be implemented efficiently because the check need not reproduce all reasoning from scratch. Systems can carry signed state, scoped grants, freshness limits, revocation indicators, evidence snapshots, and cached determinations whose validity is explicit rather than assumed.

The key is that continuity must be designed, not imagined.

This also changes how we think about autonomy.

Autonomy is often presented as a scale: how much can the agent do without asking a person?

Temporal Standing suggests a different question: how much can the system do while continuing to establish legitimate current authority for each consequence?

A highly autonomous system may still be well governed if its permissions are scoped, its evidence is current, its change signals are reliable, and its execution boundary refuses unresolved standing.

A low-autonomy system may still be poorly governed if it relies on vague human approvals that are carried forward indefinitely.

The important distinction is not human versus machine.

It is **proposal versus authority, and authority versus consequence**.

Consider an AI that manages customer accounts. It notices that a customer appears eligible for a fee waiver. It drafts a recommendation. No problem. It prepares the account change. Still no external consequence if the draft is isolated. It then attempts to write the waiver to the billing system.

At that moment the system should be able to answer questions such as:
Is this the same customer whose evidence was evaluated?
Is the account state still current?
Is the agent authorized for this category of change?
Has a conflicting action occurred since the proposal was created?
Is the amount still within delegated scope?
Is the applicable policy current?
Does the standing determination apply to this exact write?
If the answer is yes, execution can proceed.
If the answer is no, it should not.
If the system cannot establish the answer, uncertainty should remain uncertainty.

This is the transition trustworthy AI systems must make as they move from language generation into agency.

We spent years asking whether a model could produce a good answer.

We now need systems capable of answering a different question immediately before they act:

What gives this consequence the right to happen now?

A taxonomy of effects

The phrase "AI action" is too broad to govern well. Systems need to distinguish kinds of effects.

At minimum, it helps to separate five categories.

Epistemic effects change what the system believes or presents as information: retrieving a document, ranking options, generating a summary, or producing a recommendation.

State effects change a record inside a system: updating a CRM field, changing a ticket status, writing a preference, or modifying a configuration object.

Communicative effects cross a human boundary: sending a message, publishing a post, notifying an employee, or disclosing information to a third party.

Transactional effects create commitments or move value: buying, refunding, booking, filing, contracting, transferring, or provisioning paid resources.

Control effects alter what other actors can do: granting access, revoking access, changing a bank destination, deploying code, isolating infrastructure, or delegating authority to another agent.

These categories are not perfectly exclusive. A single API call can create several effects at once. But the distinction is useful because different standing predicates become material at different boundaries.

For an internal summary, evidence freshness may dominate. For a marketing message, recipient identity, purpose, and communication preference may dominate. For a payment, amount, destination, account state, fraud controls, and delegated authority may all matter. For a

permission change, the authority to alter someone else's future capabilities becomes central.

This taxonomy also explains why "the model is allowed to use the tool" is such a weak governance statement. A tool is merely an interface to one or more consequence classes. The standing question attaches to the effect, not to the button.

Once systems are designed around effects rather than tool names, governance becomes more portable. Whether a message is sent through one provider or another matters less than the fact that an external communication is about to occur. Whether a payment goes through one processor or another matters less than the transactional consequence and the predicates that must stand for it.

That is how governance survives changing implementation details.

Chapter 4 - Why Traditional AI Safety Is Not Enough

Temporal Standing does not replace the major disciplines that already surround AI.

It does not replace cybersecurity. A system with perfect standing logic can still be compromised.

Safety has a timing problem

A system can pass a bias check, a privacy check, a security review, a red-team exercise, and a very impressive slide deck - then execute an action that stopped being authorized yesterday afternoon. Governance has a talent for becoming ceremonial when every box is checked except the one attached to the actual consequence.

Temporal Standing is aimed at that last mile.

It does not replace privacy. A currently authorized action can still rely on excessive collection or poor data handling.

It does not replace accuracy. An action based on false evidence can be wrong even if the system correctly applies its authorization rules.

It does not replace fairness, alignment, interpretability, auditability, resilience, human oversight, or legal compliance.

The point is narrower and, for that reason, useful.

Those disciplines do not automatically answer the temporal question.

A model can be accurate and still act on expired authority.

A system can be secure and still faithfully execute a permission that has been revoked.

A workflow can be auditable and still document, in exquisite detail, an action that should have been blocked before it occurred.

A human can have approved the workflow months ago while the current case has moved outside the scope of that approval.

An aligned agent can sincerely pursue the user's goal while missing that the user changed the goal.

Each field asks important questions. Temporal Standing adds another:

Do the grounds that once supported this specific consequence still support it now?

This addition matters because AI governance can become a checklist of desirable properties without a clear causal theory of when an action should be stopped.

Organizations publish principles. Teams build dashboards. Models receive system prompts. Risk assessments classify use cases. Logs record decisions. Review boards define policies. Red-team exercises identify failure modes.

All of these can contribute to responsible systems.

But governance becomes operational only when it changes what the system is capable of causing.

If the organization says, "Do not contact people who have withdrawn consent," where is that rule enforced?

If the answer is "the model was instructed not to," the system is relying on model behavior.

If the answer is "we audit campaigns afterward," the system is relying on retrospective detection.

If the answer is "the workflow checks the authoritative preference state immediately before dispatch and cannot send without a current permit," the rule has become causal.

That is the difference between governance as description and governance as control.

Temporal Standing is especially concerned with this transition because stale standing often hides inside otherwise reasonable architectures.

Consider access control.

Traditional authorization systems already understand many temporal ideas: sessions expire, tokens have lifetimes, roles can be revoked, privileges can be scoped, and policy decisions can be reevaluated. Temporal Standing does not claim these ideas were absent from computing.

The contribution is to generalize the underlying problem across AI-mediated action and to connect authorization with changing evidence, consent, facts, purpose, and consequence.

An access token can be valid while the proposed action is still outside legitimate scope.

A user may be authenticated while the user's prior consent for a particular data use has changed.

An agent may possess permission to call a tool while lacking present standing to submit this payload to this recipient for this purpose.

The fact that security systems know how to revoke credentials does not mean agent architectures automatically know how to govern changing real-world authority.

Consider alignment.

If an AI system is trying to follow the user's intent, it must represent the possibility that the user's intent changed. Otherwise, stronger goal pursuit can make the failure worse. An agent that relentlessly carries out yesterday's objective after today's revocation is not safer because it is more capable.

Temporal Standing therefore supplies a temporal condition on obedience: follow the instruction only while the instruction retains standing for the action being performed.

Consider human oversight.

"Human in the loop" is often treated as a universal answer. But a human approval can itself become stale.

A reviewer approves a transaction under one set of facts. The transaction is delayed. The amount changes. The counterparty changes. A relevant policy changes. The system later executes because "a human approved it."

The presence of the human did not solve the temporal problem.

The correct question is not merely whether a human reviewed the action. It is whether the reviewed action and the executed action remain materially the same within the authority envelope, and whether the approval still stands at execution.

Consider auditability.

An excellent log might show:

9:00 - user authorized action.

9:15 - agent prepared action.

9:30 - user revoked authorization.

10:00 - worker executed action.

10:01 - audit log recorded successful completion.

The system is beautifully auditable.

It is also wrong.

Audit is evidence. Prevention is enforcement.

The two should support each other, not substitute for each other.

This is why Temporal Standing should be understood as a condition of legitimate execution rather than another broad ethical slogan.

It asks for a system property:

Immediately before a consequential effect, can the system establish that the relevant prior grounds remain current, scoped, and sufficient for this effect?

If yes, standing is preserved.

If evidence establishes that those grounds no longer support the action, standing is defeated.

If the system cannot establish either state, standing is unresolved.

The unresolved state is where many safety frameworks become uncomfortable, because organizations want automated systems to finish workflows. Operational pressure rewards completion. Product metrics reward success rates. Reliability engineering rewards recovery. Users dislike friction. Agents are marketed on their ability to "get things done."

All of those incentives can pressure a system to convert uncertainty into action.

Temporal Standing insists that some uncertainty is governing information.

If a current consent source cannot be reached, "we could not check" is not equivalent to "consent is still valid."

If the current delegated authority cannot be established, "it was approved last month" is not equivalent to "it is approved now."

If a critical state changed and the consequences are unclear, "the model thinks it is probably fine" is not the same thing as standing.

This does not require every application to stop whenever any service is unavailable. Systems can be designed with redundant authoritative sources, explicit offline rules, bounded freshness windows, signed grants, known fallback authority, and carefully limited safe modes.

But those fallback paths must themselves be authorized.

Convenience is not a source of authority.

This book therefore sits beside existing AI safety disciplines, not above them.

Accuracy asks whether the representation of reality is correct.

Security asks whether systems and permissions resist unauthorized access and manipulation.

Privacy asks whether data is collected, used, and disclosed appropriately.

Alignment asks whether system behavior tracks legitimate objectives and values.

Auditability asks whether actions and decisions can be reconstructed.

Temporal Standing asks whether the governing grounds that once supported an action still support the exact consequence now.

A trustworthy acting system may need all of these.

The danger comes when success in one category is mistaken for success in another.

A secure system can securely do the wrong thing.

An accurate model can accurately remember obsolete authority.

An auditable workflow can perfectly document illegitimate execution.

A human-approved plan can outlive the approval that made it legitimate.

A capable agent can become more dangerous precisely because it is good at finishing old work.

The missing rule is not "be less intelligent."

It is "do not let intelligence turn stale standing into present consequence."

The problem has neighbors

A serious account of Temporal Standing has to begin with humility about what already exists.

Computer security has studied time-aware authorization for decades. Temporal authorization models attached periods of validity and temporal dependencies to permissions. Temporal role-based access control added time to role activation. Usage Control, or UCON, went further by treating authorization as something that may need to continue during usage while attributes and conditions change. Zero Trust later popularized a related operational instinct: do not let yesterday's network position or yesterday's trust relationship become an invisible permanent credential. Continuous access evaluation systems propagate changes such as session revocation, credential change, risk change, or device-compliance change so downstream systems can react.

Those ideas matter because they prevent this book from making a cheap claim.

The claim is not that nobody noticed permissions expire.

The claim is not that no security system reevaluates access.

The claim is not that revocation is new.

And the claim is not that policy decisions have never been separated from enforcement.

Temporal Standing addresses a different composition problem that becomes especially visible when AI systems move from answering to acting.

An acting system may have valid access to a tool and still lack grounds to create a particular consequence. It may be authenticated, authorized to call an API, and technically capable of executing while the user's consent for the purpose has changed. It may have permission to access a calendar but lack standing to cancel a meeting the user only asked it to summarize. It may be permitted to initiate payments but face a changed bank destination that makes the specific transfer unresolved. It may be able to send email while the specific recipient, purpose, or governing instruction has changed.

This is why access control is necessary but not sufficient for the problem examined here.

Access control asks whether an actor may use a resource under policy.

Temporal Standing asks whether the heterogeneous grounds supporting this exact consequence still stand now.

Those grounds can include access rights. They can also include consent, delegated authority, factual conditions, purpose, scope, current identity, policy, eligibility, preferences, contractual limits, and evidence quality.

The distinction is subtle until an AI agent starts combining systems.

A human operator often carries contextual judgment across those systems. An agent can instead inherit a collection of individually valid capabilities and compose them into an unjustified action. The email API says yes. The CRM says yes. The calendar API says yes. The payment API says yes. The model has a plausible goal. None of those facts, alone, answers whether the resulting consequence still has standing.

Traditional AI safety discussions often focus on the model: is it aligned, accurate, robust, secure, explainable, or biased? Those remain important. But the consequence can be illegitimate even when the model is functioning exactly as designed.

The model may accurately remember a permission that was once real.

The failure is temporal and governing, not cognitive.

That is why Temporal Standing belongs beside existing security and governance disciplines rather than claiming to replace them. A mature implementation may use Zero Trust for identity, ABAC or Cedar for authorization, a policy engine for organizational rules, consent systems for user permissions, provenance systems for evidence, and event streams for revocation. Temporal Standing is the discipline that asks how those pieces combine at the moment a concrete consequence is about to occur.

The question remains deceptively simple:

Whatever supported this action before, does it still support this exact action now?

Existing frameworks are broader in some ways and narrower in others

A credible argument for Temporal Standing has to begin by refusing an easy rhetorical trick: declaring a blank space wherever an existing framework uses different vocabulary.

The field is not blank.

NIST's AI Risk Management Framework treats AI risk management as a lifecycle activity and organizes it around Govern, Map, Measure, and Manage. It explicitly calls for risk tracking over time and for organizations to understand the context in which systems operate. The European Union's AI Act requires human-oversight measures for high-risk systems that are proportionate to risk, autonomy, and context of use. Security engineering has long insisted on complete mediation, least privilege, revocation, and current authorization. Modern agent guidance increasingly distinguishes ordinary model output from high-impact tool actions and recommends stronger controls around side effects. [1][2][3]

Temporal Standing should not compete with these bodies of work by pretending they failed to notice context, time, or oversight.

Its contribution is more specific.

Risk frameworks operate at organizational and lifecycle scales. Access-control systems answer whether a principal may perform an action on a resource under policy. Human-oversight requirements govern how people can supervise certain systems. Agent guardrails constrain model behavior and tool use.

Temporal Standing focuses on the narrow instant at which heterogeneous grounds become a consequential external effect. It asks whether the authority chain supporting **this exact consequence** survived relevant change from T_0 to T_n, and it requires the answer to be represented in a way that the executor can enforce.

This is why the framework belongs *alongside* AI risk management rather than above it. A NIST-style risk program can identify that stale consent is a material risk. A privacy system can record withdrawal. An authorization engine can evaluate current access. A human-oversight process can define escalation. Temporal Standing is the composition question that remains at execution: given all governing inputs, does the proposed consequence still stand now?

The distinction matters because institutional controls often fail at the seams. The consent database may be correct while the campaign queue is stale. The identity service may revoke a role while the worker still carries an old token. The policy engine may deny a request while the application has a bypass path. A human may reject an action while a retry worker replays the original payload.

Each component can look reasonable in isolation.

The failure lives in the transition between them.

Temporal Standing is designed for that transition.

The claim should be falsifiable

A governance concept becomes stronger when we can say what would make it unnecessary.

If an existing system already provides, for a proposed AI-mediated consequence, all of the following—current heterogeneous governing inputs, explicit scope, relevance-aware change handling, a preserved/defeated/unresolved determination, payload-bound authority, and causal enforcement immediately before effect—then Temporal Standing may add little more than vocabulary.

That is not a weakness. It is a test.

The framework earns its value where those properties are absent, fragmented, or implicit. The research task is therefore not to prove that no one has ever solved any part of the problem. It is to determine whether the composition discipline is useful across domains where multiple authority systems meet an acting AI.

A framework that cannot name the conditions under which it becomes redundant is too easy to protect from criticism.

PART II

THE FRAMEWORK: TEMPORAL STANDING

Chapter 5 - $T_0 \rightarrow \Delta N \rightarrow T_n$

Temporal Standing begins with a simple observation: authority has a history.

An action proposed now may depend on a permission granted earlier, a fact established earlier, a role assigned earlier, a policy adopted earlier, or a condition observed earlier. Those prior states matter. Without them, many useful systems could not operate.

The diagram is simple on purpose

$T_0 \rightarrow \Delta N \rightarrow T_n$ is not intended to win an award for mathematical intimidation. If the notation required a graduate seminar before anyone could use it, it would fail its first governance test.

The point is deliberately ordinary: what stood then, what changed, and what stands now?

But prior states are not self-renewing.

The notation $\mathbf{T_0 \rightarrow \Delta N \rightarrow T_n}$ is a compact way to force a system - and a human reviewer - to examine the transition from prior standing to present standing.

T_0: the prior authoritative state

T_0 is not merely "what the database said." It is the state that legitimately supported the proposed action at an earlier point.

That state may include an instruction, consent record, role assignment, contract term, account status, policy, identity assertion, approval, risk classification, eligibility determination, or evidence bundle. It also includes boundaries.

Who issued the authority?
What action did it cover?
What data did it permit the system to use?
For what purpose?
For how long?
Under what conditions?
For which recipient or object?
What consequence was actually contemplated?

A T_0 record that omits these dimensions is not necessarily useless, but it is weaker than it appears. "Approved" is not a complete governance state. Approved by whom, for what, and under which conditions?

The point is not bureaucratic completeness. The point is preserving enough context to know what the prior authority meant.

ΔN: what changed

Between T_0 and T_n, the world continues.

We use **ΔN** to represent relevant intervening change. The N is intentionally open-ended. There may be one change or many. Some changes are explicit: a user revokes consent, a manager cancels an approval, a role ends, a policy changes.

Others are factual: a price rises, an account closes, a patient condition changes, inventory disappears, a service becomes unavailable.

Others alter scope: the proposed recipient changes, the amount increases, a different tool is selected, additional data is introduced, the purpose expands.

Still others create uncertainty: an authoritative source becomes unavailable, two records conflict, a required freshness window expires, or the system can no longer establish which identity is controlling the request.

A ΔN is not automatically a veto.

This matters.

If every detected change defeated prior authority, automation would become brittle. A customer changing a mailing address does not necessarily invalidate an authorization to renew a service. A new software version does not necessarily cancel a previously approved business process. A time interval does not automatically destroy standing if the authority was explicitly durable.

The correct question is whether the change is relevant to the consequence.

Temporal Standing is therefore not simply a "change detector." It is a framework for determining the effect of relevant change on present authority.

T_n: the present determination

T_n is the point at which the system needs an answer for the consequence it is about to create.

The determination at T_n is bounded. It should not be read as "this user is trusted," "this agent is safe," or "this workflow is approved forever." It answers a narrower question:

Do the governing grounds currently support this exact action, with this payload, through this tool, for this purpose, against this target, now?

This specificity is not pedantry. It prevents authority from leaking across actions.

Suppose a person authorizes an agent to send a meeting invitation to three named colleagues. A standing determination permits those invitations. The agent later decides that five additional people "would probably benefit." The earlier permit does not silently expand.

The agent may have a good idea. It still needs authority for the new consequence.

The three outcomes

At T_n, standing can be **preserved**, **defeated**, or **unresolved**.

Preserved means the relevant grounds remain current and the action remains within scope.

Defeated means the grounds no longer support the action. Something relevant has changed, expired, been revoked, contradicted, or moved the action outside scope.

Unresolved means the system cannot establish preserved or defeated standing with sufficient authority.

The unresolved state is not an error code to be hidden. It is a legitimate governance outcome.

In a mature system, unresolved standing should drive a defined response: hold, request clarification, refresh evidence, seek human determination, narrow the action, or move into a separately authorized safe mode.

What it should not do is silently become permission.

A concrete example

At T_0, a finance director authorizes an agent to pay a named vendor invoice up to $25,000 if the invoice matches a purchase order and the bank details match the approved vendor record.

The agent prepares payment.

Before execution, ΔN occurs: the vendor submits new bank details by email. The invoice is still valid. The amount is still within scope. The purchase order still matches.

At T_n, can the agent pay the new bank account?

The original authority did not necessarily extend to unverified replacement banking instructions. Standing for paying the invoice may remain preserved while standing for paying the new destination is unresolved or defeated under the organization's controls.

This example shows why standing attaches to a consequence, not an abstract goal.

"Pay the vendor" is too coarse.

The consequence includes the destination, amount, timing, and other governing details.

Temporal Standing is relational

Standing is not a property stored permanently inside an instruction.

It is a relationship between prior grounds, intervening changes, the current evidence state, and a proposed consequence.

That is why the same T_0 can yield different outcomes for different actions at T_n.

A user's consent to store an email address may still stand for account recovery while not standing for a new marketing use.

An employee's role may still permit reading a case while no longer permitting approval.

A customer's account status may support sending a required service notice but not a promotional message through a channel the customer revoked.

The question is always scoped.

The discipline of asking

The value of $\mathbf{T_0 \rightarrow \Delta N \rightarrow T_n}$ is partly technical and partly cognitive.

It trains builders and decision-makers to stop treating history as authority by default.

When a system is about to act, ask:

What exactly gave this action standing at T_0?

What relevant changes have occurred since?

What is the authoritative present state?

Does the prior standing remain preserved for this exact consequence?

If not, is it defeated or unresolved?

What mechanism actually prevents execution when standing is not preserved?

These questions sound simple.

In complex systems, they expose architecture.

They reveal where permissions are vague, where evidence is stale, where revocations do not propagate, where retries inherit authority they should not, where human approvals are overbroad, and where policy exists only as language rather than enforcement.

The framework is memorable because the failure is common.

What was true then may still be historically true.

What was authorized then may still be correctly recorded.

But before a system acts now, history must acquire present standing.

What the notation does not claim

Because the notation is simple, it can be mistaken for a claim that time itself is the novelty.

It is not.

$T_0 \rightarrow \Delta N \rightarrow T_n$ is a governance lens for organizing a transition. Temporal access-control research has long modeled validity intervals, temporal dependencies, and changing authorization states. Usage Control explicitly recognized decision continuity and mutable attributes. Modern continuous-access systems distribute change events so authorization can be attenuated when sessions, credentials, devices, or risks change.

Temporal Standing borrows the seriousness of those observations while applying them to a broader object.

T_0 is not simply 'the old permission.' It is the prior authoritative state that supported a proposed consequence. That state can be composite. A send action, for example, might depend on the sender's delegated role, a user's communication preference, an approved purpose, a verified recipient, a valid campaign policy, and a current suppression state.

ΔN is likewise not merely elapsed time. It is the set of relevant changes capable of altering the standing of those grounds. Time can be one change. Revocation can be another. A changed recipient can be another. New

information can defeat a factual predicate. A policy revision can narrow scope. Loss of access to an authoritative source can create uncertainty without proving that the authority was revoked.

T_n is not simply 'check again.' It is the present determination for a bounded consequence.

That boundedness is essential. A system that asks whether 'the workflow is still approved' may hide the exact thing that changed. A system that asks whether 'the agent is still trusted' makes the same mistake at a different level. Trust, workflow state, and historical approval are all too coarse when the consequence has material fields such as recipient, amount, target, purpose, payload, or tool.

The notation therefore forces a discipline:

Name the prior grounds. Name the relevant change. Name the exact present consequence.

Only then determine standing.

Unknown change and bounded knowledge

There is another limit worth making explicit.

No system can prove that nothing relevant changed anywhere in the world.

ΔN is not omniscience.

A system can only reason over changes it is required to check, changes it receives through authoritative signals, changes reflected in current authoritative state, and changes it can establish through a legitimate evidence process. A revoked consent that never reaches the system is still a real-world change, but the system may not know it occurred.

This means Temporal Standing depends on evidence architecture.

For some consequences, a five-minute-old cache may be sufficiently current. For others, current state may need to be read transactionally at the executor. Some authorities publish revocation events. Others expose version numbers. Some conditions are durable until explicitly changed. Others decay rapidly and require freshness limits.

The framework does not prescribe one universal freshness interval. It asks the system to make freshness a governed property instead of an accidental one.

A bounded claim is therefore better than an absolute claim:

The system should establish **sufficient current authoritative grounds** for the consequence, under a freshness and evidence standard proportionate to the consequence.

That formulation is less dramatic than saying the system 'knows reality.'

It is also implementable.

Historical truth can survive defeated standing

One of the most useful features of the framework is that it does not rewrite history.

If a user validly consented on Monday and withdrew consent on Thursday, both facts can remain true.

The Monday consent was real.

The Thursday withdrawal was real.

At T_n on Friday, the historical record of Monday should not be deleted merely to prevent future use. Instead, the standing of that prior consent for a new consent-dependent consequence is defeated.

This pattern appears in privacy law as well. Under the GDPR, withdrawal of consent does not retroactively erase the lawfulness of processing performed before withdrawal, while the withdrawn consent cannot simply be treated as present consent for future processing.

Temporal Standing generalizes that shape without pretending every domain is legally equivalent.

History remains history.

Standing is present.

The relevance test for ΔN

The notation is intentionally simple, but implementation requires a harder question: which changes count?

A system cannot monitor every event in the universe. Nor should it. The relevant set of ΔN is defined by the predicates that support the proposed consequence.

If a purchase depends on a price ceiling, delivery deadline, approved vendor, buyer identity, and valid payment method, then changes to those predicates are candidates for relevant ΔN. A change to the user's avatar is not.

If a communication depends on recipient identity, communication preference, purpose, campaign status, and account suppression state, those are the change surfaces that matter.

This suggests a practical modeling pattern. For each consequence class, define a **standing predicate set**:

1. **Authority predicates** — who or what can authorize the action?
2. **Identity predicates** — who is acting, who is affected, and who is the target?
3. **Scope predicates** — what action, resource, purpose, amount, duration, and boundaries are covered?
4. **State predicates** — what facts must still be true?
5. **Policy predicates** — what institutional rules or constraints apply now?
6. **Evidence predicates** — what sources must be available, fresh enough, and non-conflicting?
7. **Consequence predicates** — is the payload being executed materially the same as the one evaluated?

ΔN is relevant when it can change the truth value or sufficiency of one of these predicates.

This does not eliminate judgment. It moves judgment into an explicit model where it can be tested.

Unknown unknowns do not become known by governance language

Temporal Standing cannot promise that every material change will be observed.

That limitation should be stated plainly.

A revocation that never reaches the system cannot be honored by magic. A policy change in an inaccessible jurisdictional source cannot be evaluated. A fraudulent identity that defeats every available identity control may appear authoritative. A fact that no instrument measures remains unknown.

The framework therefore makes a bounded claim: determine standing from the authoritative evidence and change signals the system is obligated and able to consult, and represent missing required evidence as unresolved rather than silently converting absence into permission.

This boundary is important because "continuous governance" can otherwise become marketing language that implies omniscience. Real systems are partial observers.

The engineering work is to define which sources are required for which consequence classes, how fresh they must be, what invalidates cached results, and what happens when those sources disagree or disappear.

That is testable.

Omniscience is not.

Chapter 6 – Scope Matters: Authority Is Never Generic

"Permission granted" sounds decisive until we ask what permission means.

A person can authorize one action without authorizing another. A manager can delegate a category of work without delegating every possible consequence inside that category. A customer can consent to one use of data without consenting to every future use. An employee can have access to a system without having authority to execute every operation the system exposes.

`approved = true` is not a constitution

Systems love generic flags because generic flags are easy to query. Humans love them because they look reassuring in dashboards. Unfortunately, "approved" without actor, action, object, purpose, limits, duration, and consequence is less an authority record than a motivational poster.

Scope is where permission becomes governable.

Authority is scoped.

AI agents make scope unusually difficult because they are designed to generalize. Give a capable agent a goal and it may discover many ways to pursue it. That flexibility is valuable for cognition. It is dangerous when flexibility silently expands authority.

Suppose an executive tells an agent, "Prepare everything needed to close this vendor contract." The agent might draft the agreement, assemble diligence, schedule a meeting, request missing information, and organize documents. Does it also have authority to sign the agreement? Transfer a deposit? Disclose confidential data to a subcontractor? Modify an internal approval record?

The goal alone does not answer those questions.

The system needs an authority envelope.

A useful authority envelope can be expressed through eight dimensions:

Actor. Who is granting or holding authority?

Action. What operation is permitted?

System. Which agent, service, workflow, or tool may perform it?

Data. Which information may be used or disclosed?

Purpose. Why is the action authorized?

Duration. How long does the authority remain available before renewal or reassessment?

Conditions. Which facts or thresholds must remain true?

Consequence. What external effect is actually within scope?

These dimensions overlap, but together they make a crucial point: authority is not an ambient substance that an AI possesses.

It is a bounded relationship.

This changes how we should read statements like "the AI has access."

Access is a technical capability. It says the system can reach a resource.

Authority says whether the system may use that capability for this consequence under present conditions.

A database credential can give an agent technical access to customer records. That does not mean every query, every disclosure, and every downstream use is authorized.

A payment API can give an agent technical capability to move funds. That does not mean every payment is within scope.

A communications service can let the system send to any syntactically valid address. That does not create permission to contact every address.

This distinction is familiar in mature security engineering, but AI systems often obscure it because model reasoning sits above the tools. Developers may think, "The agent will decide when the tool is appropriate." Yet appropriateness and authority are not identical.

The model can infer purpose. It cannot manufacture scope.

Scope drift

A major Temporal Standing failure is **scope drift**: the proposed action at T_n resembles the action authorized at T_0 closely enough that the system treats it as the same, even though one or more governing dimensions have changed.

A $5,000 purchase becomes $7,500.

One recipient becomes a mailing list.

A draft becomes a filing.

A recommendation becomes an automated change.

Internal analysis becomes external disclosure.

One dataset becomes a joined dataset containing new sensitive fields.

A one-time authorization becomes recurring execution.

A support action becomes a marketing action.

A user-authorized agent delegates to another agent whose permissions were never defined.

Each change may seem incremental. The accumulation can move the consequence outside the original authority envelope.

This is why a permit should describe the bounded action it supports rather than simply attach to a session.

Session-level authority is tempting: once the user has authenticated and approved a workflow, allow the agent to operate until the session ends. But sessions are technical containers, not necessarily governance boundaries.

A single session can contain many different consequences.

If the agent's plan changes materially, a current standing determination should follow the new consequence.

Purpose limitation

Purpose deserves special attention because AI systems are good at repurposing information.

A dataset collected for support may be useful for sales. A conversation collected for service quality may be useful for employee evaluation. A location shared to complete one task may be useful to personalize another. A document uploaded for summarization may contain facts that could influence unrelated decisions.

The fact that the system can infer value from the data does not mean the original authority covered the new purpose.

Temporal Standing therefore asks not only "Is this data still current?" but also "Does the authority for this use still stand?"

Purpose can change even when the data does not.

Delegation and sub-agents

Agentic systems frequently delegate. A planning agent calls a research agent. A research agent calls a browser. A workflow agent calls a communication service. One system hands a task to another.

This creates a dangerous temptation: treat upstream approval as inherited authority.

But delegation should preserve scope, not erase it.

If Agent A is authorized to assemble options, it should not be able to give Agent B authority to purchase merely because purchase would complete the objective more efficiently.

Authority cannot be expanded by delegation unless the original authority permits that expansion.

The same principle applies to humans. An employee cannot usually delegate powers they do not possess. AI should not be different merely because delegation occurs through an API.

The exact-consequence test

One practical discipline is to describe the action at the level that would matter after something goes wrong.

Not "contact the customer," but "send this message to this address through this channel for this purpose now."

Not "pay the invoice," but "transfer this amount from this account to this verified destination for this obligation."

Not "update the record," but "change these fields in this system based on this evidence."

Not "book the trip," but "purchase this itinerary for these travelers at this price under these cancellation terms."

The more precise description often reveals whether the action remains inside the authority envelope.

This does not mean humans must approve every field manually. Precision can be machine-readable. Systems can define allowed ranges, recipient classes, tool constraints, data categories, expiry, purpose tags, transaction ceilings, and required evidence.

The goal is to let automation move quickly inside legitimate scope while making scope boundaries explicit enough to enforce.

Authority is not generic trust

Organizations often describe people or systems as trusted.

Trust can be socially meaningful. In governance architecture, it is too vague.

A trusted employee should still have bounded access.

A trusted agent should still have bounded authority.

A trusted model should still be unable to convert its own judgment into unrestricted execution rights.

Temporal Standing replaces the question "Do we trust the agent?" with more operational questions:

What is it permitted to cause?

Under what present conditions?

How does that authority change?

How is revocation represented?

What happens when scope becomes ambiguous?

Who or what can expand scope?

How is the expansion recorded?

A system can be highly useful without being generically trusted because the environment makes illegitimate consequences difficult to cause.

That is a more durable form of trust.

It does not depend on believing that a model will always know its place.

It defines the place.

The authority tuple

A useful way to make scope concrete is to treat authority as a tuple rather than a sentence.

For a consequential action, the tuple might include:

actor × principal × action × target × purpose × data × constraints × duration × delegation depth × consequence class

The exact fields vary by domain, but the discipline matters. "Approved" is not a scalar property. It is a relationship among these dimensions.

A user may authorize an agent to search flights but not purchase them. A manager may authorize purchases but only for one department and only below a threshold. An employee may be permitted to issue refunds but not to change the bank destination. A data subject may permit processing for one purpose without granting a universal license for future downstream uses.

When systems store authority as a coarse flag, scope drift becomes almost inevitable. A Boolean such as `user_approved=true` sheds the very information needed to determine what still stands later.

The tuple also helps delegation. A sub-agent should not inherit more scope than its parent can legitimately delegate. If the parent has authority to obtain quotes, delegation should not expand into authority to sign a contract. If the original authority expires at a deadline, the delegated authority should not outlive it merely because another component cached the task.

This is not novel access-control mathematics. It is a practical governance discipline: preserve enough structure that the future system can distinguish the action that was authorized from nearby actions that merely look similar.

Chapter 7 - Preserved, Defeated, Unresolved

A governance system needs more than a list of rules. It needs an outcome model.

Temporal Standing uses three outcomes because the real world contains three fundamentally different states.

The third answer nobody puts on the sales slide

Product demos prefer green lights. Red lights are tolerable because they at least look decisive. Yellow lights - *we do not currently know enough to justify this consequence* - are less cinematic.

Unfortunately, reality did not sign the demo script. Unresolved is a legitimate state, and pretending otherwise merely converts uncertainty into unauthorized confidence.

The first is straightforward: the system has sufficient current grounds to proceed.

The second is also straightforward: the system has sufficient current grounds to conclude that it should not proceed.

The third is the one software systems often try to hide: the system does not know enough to justify either conclusion.

Those states are **preserved**, **defeated**, and **unresolved**.

Preserved standing

Standing is preserved when the relevant prior authority remains current and the proposed consequence remains inside its scope.

Preserved does not mean "safe in every possible sense." A permitted action can still fail technically. A person can make a bad decision within their authority. A policy can be flawed. Data can contain an undetected error.

Temporal Standing makes a narrower claim: nothing relevant has defeated the governing grounds, and the current evidence is sufficient to support the bounded action.

This distinction prevents the framework from becoming inflated into a universal certification.

A preserved determination should answer: **preserved for what?**

For this action.

For this target.

For this purpose.

For this payload or bounded class of payloads.

At this time or within this explicit freshness window.

A well-designed permit can carry those boundaries downstream.

Defeated standing

Standing is defeated when current authoritative information establishes that the previous grounds no longer support the action.

Examples are intuitive.

The user revoked permission.

The role ended.

The approval expired.

The amount exceeded the authorized limit.

The target changed to one outside scope.

A required condition is no longer true.

A current policy prohibits the action.

An upstream authority was itself revoked.

The important part is causal consequence: if standing is defeated, the system must not cross the governed boundary under the old authority.

The action may become legitimate later through new authority. A human may issue a fresh approval. The user may consent again. The amount may be reduced. The target may be corrected. A policy exception may be granted by an authorized decision-maker.

Defeated does not mean eternally forbidden.

It means **not authorized by the standing currently presented**.

Unresolved standing

Unresolved is different.

The system does not possess sufficient authoritative grounds to establish preserved standing, but neither does it possess sufficient grounds to conclude that standing is definitively defeated.

This can happen when an authoritative source is unavailable.

It can happen when evidence conflicts.

It can happen when a required record is stale.

It can happen when identity is uncertain.

It can happen when the current policy is ambiguous for the proposed action.

It can happen when the action differs from the original scope in a way that requires human interpretation.

It can happen when a change signal says “something relevant changed” but the system cannot yet determine its effect.

Unresolved is not a nuisance state. It is one of the most important outputs of a trustworthy system.

Why? Because automated systems are under constant pressure to finish.

A queue expects a terminal state. A product team wants completion. A user wants a result. A retry mechanism wants eventual success. A model is optimized to be helpful. An orchestration framework may treat exceptions as problems to recover from.

Those pressures can turn “we do not know” into “probably yes.”

Temporal Standing refuses that conversion at the consequence boundary.

If standing is unresolved, the action should be held unless there is a separately authorized rule that governs the unresolved condition.

For example, a system might have explicit authority to enter a non-consequential draft mode while a policy source is unavailable. It might be allowed to prepare but not submit. It might be allowed to notify a human reviewer but not contact the external party. It might be allowed to execute a narrowly defined emergency action under a separately established emergency authority.

The point is not that unresolved means "do nothing forever."

The point is that the next action must have its own standing.

Why not just allow or deny?

Binary systems often map unresolved to deny, and operationally that can be appropriate: no external consequence occurs.

But preserving the distinction matters for diagnosis and resolution.

Defeated means there is a governing reason not to proceed under current authority.

Unresolved means the system needs more authoritative information or a new determination.

The remediation paths differ.

If consent was revoked, asking the preference database again will not solve the problem. New consent may be required.

If the preference service is temporarily unavailable, the user may not need to do anything. The system needs to reestablish current evidence.

If the scope is ambiguous, a human may need to interpret or reauthorize.

Conflating these states makes systems less explainable and can encourage dangerous retries.

Retry behavior

Imagine a dispatch worker receives an unresolved result because the current preference service is unavailable.

A naive worker treats that as a transient technical failure and retries the send operation itself.

A governed worker retries the standing determination.

That difference is profound.

The first assumes authority and retries transport.

The second recognizes that authority is the missing dependency.

Similarly, if standing is defeated because the user revoked consent, retrying the determination every minute may be wasteful and potentially coercive if the workflow is designed to search for permission. The system should enter an appropriate terminal or human-controlled state.

Outcome semantics should drive workflow semantics.

Human review is not magic

Unresolved standing often leads to human review, but human review must itself be governed.

Who is authorized to decide?

What evidence do they see?

What scope can they approve?

How long does that approval remain current?

Can they override a policy, or only resolve missing facts?

Does their decision apply to one action or a class of actions?

A "human approved" flag without these dimensions simply moves vagueness upstream.

The discipline of preserving uncertainty

One of the deepest responsibilities in high-consequence automation is preserving uncertainty honestly.

Models are good at completing patterns. Organizations are good at creating pressure for closure. Interfaces are good at reducing complex states to green and red indicators.

But reality does not owe us a binary answer on demand.

Sometimes the right system state is:

Current standing cannot yet be established.

That sentence can feel unsatisfying.

It is also a sign of maturity.

A system that can say "I don't know yet" before consequence is more trustworthy than one that always knows because it has quietly defined uncertainty as permission.

Why unresolved deserves its own name

Security systems often compress the world into allow and deny.

For execution, that can be appropriate. If a system cannot establish permission, it may need to block the action. But governance needs more semantic precision than the transport gate.

Consider three cases.

In the first, a user explicitly revoked permission. The governing state is known. Standing is defeated.

In the second, permission remains valid and every relevant condition remains satisfied. Standing is preserved.

In the third, the authoritative consent service is unavailable, the cached record is beyond its permitted freshness window, and no reliable revocation signal can be obtained. The system does not know that permission was revoked. It also lacks sufficient grounds to say permission remains current.

Calling the third case 'denied' may be safe at the executor, but it loses the reason.

That reason drives the next action.

Defeated standing may require new authority, a different scope, or abandonment of the proposed consequence.

Unresolved standing may require refreshing evidence, reconciling conflicting records, restoring an authoritative dependency, verifying identity, narrowing the request, or escalating to a human with the competence to determine the missing point.

This is why unresolved is not merely a softer word for no.

It is an epistemic state with operational consequences.

The distinction also prevents a common automation failure. Systems under pressure often treat uncertainty as a retry problem. If an authorization service times out, the application retries the same action. If a provider response is ambiguous, it sends again. If evidence cannot be loaded, it assumes the old state remains good enough.

Temporal Standing says the retry target matters.

Retry the missing dependency, not the consequence blindly.

If standing is unresolved because evidence is unavailable, refresh the evidence. Do not repeatedly execute the consequence in hopes that one attempt happens to be legitimate.

The same discipline applies after ambiguous transport outcomes. If an email provider timed out after accepting a message, the governance question may still be preserved, but the execution state is unresolved. Blind retry can create a duplicate consequence even though authority was valid.

Governance uncertainty and transport uncertainty are different dimensions.

Mature systems name both.

Unresolved is an epistemic state with causal consequences

It is tempting to treat unresolved as a softer synonym for deny.

That loses useful information.

Defeated standing says the system has sufficient governing evidence to conclude that the prior grounds no longer support the consequence. Unresolved standing says the system lacks enough authoritative information to make that determination safely.

The causal result may be the same—do not execute—but the remediation is different.

If consent was explicitly withdrawn, asking the same policy service again will not help. Standing is defeated. A new consent or a different lawful basis, where applicable, would be needed.

If the authoritative consent source is temporarily unavailable, standing may be unresolved. Retrying the dependency could resolve it without changing the user's underlying choice.

If two authoritative records conflict, standing may remain unresolved until hierarchy or reconciliation rules select the controlling state.

If the proposed payload changed after approval, the old standing decision may no longer apply even though the underlying authority is probably still valid. Re-evaluation can resolve the mismatch.

This distinction supports better operations. Systems can route defeated cases toward reauthorization, unresolved cases toward evidence recovery or human review, and preserved cases toward execution.

It also improves audit quality. A record that says only `DENY` can hide whether the system protected the user because it knew the action was disallowed or because it did not know enough to allow it.

That difference matters when debugging, measuring availability, or deciding whether governance is too brittle.

Chapter 8 - Evidence Is Not Authority

Modern AI systems are built to consume evidence.

They retrieve documents, search databases, summarize records, inspect tool output, classify messages, extract entities, compare policies, and build elaborate context windows. The result can feel authoritative because the system has assembled so much information.

Receipts are not permission slips

A log entry can prove that someone approved something. A screenshot can prove what a policy page said. A model can summarize both beautifully. None of those facts, by themselves, creates present authority.

Evidence tells us what we know. Authority tells us what may follow. Mixing the two is how very well-documented mistakes happen.

But evidence and authority are different things.

A record that says a user consented is evidence that consent existed.

A signed approval is evidence that an authorized person approved something within some scope.

A database row is evidence of a stored state.

A policy document is evidence of a rule.

A model interpretation is evidence of how the model read those materials.

A confidence score is evidence about the model's internal certainty.

None of these automatically establishes current authority for a new consequence.

This distinction becomes critical because large models are persuasive synthesizers. They can take weak, stale, conflicting, or partial evidence and produce a coherent explanation. Coherence can look like legitimacy.

The system may say:

"The user previously opted in and has interacted positively with similar messages, so it is reasonable to proceed."

That could be a plausible prediction of preference. It is not necessarily current consent.

Or:

"The manager approved this category of purchases last quarter, and this purchase appears consistent with the team's objectives."

That may be a reasonable inference. It is not necessarily present delegated authority.

Or:

"No revocation is visible in the records I retrieved."

That statement may simply mean the retrieval path did not include the authoritative revocation source.

Absence of evidence is particularly dangerous in automated governance because systems can quietly turn "I did not find a contradiction" into "standing is preserved."

Preserved standing should depend on what the governing architecture defines as sufficient current evidence, not on the model failing to notice a problem.

Provenance matters

Evidence is only as useful as its provenance.
Where did it come from?
Who created it?
When?
Was it authoritative for the question at hand?
Has it been superseded?
Is it complete?
Was it transformed?
Does the transformation preserve the parts that matter?
A summary can be factually accurate and still omit a revocation clause.
A copied record can be authentic and still be stale.
A search result can quote an old policy after a new policy has taken effect.
A model memory can faithfully reflect a user's earlier preference while missing a later reversal.

Temporal Standing therefore requires not only evidence retrieval but evidence lineage.

Freshness is contextual

There is no universal freshness interval.

A mailing address might remain useful for months in one context. A market price can become obsolete in seconds. A user's communication preference should be checked according to the consequence and the organization's authoritative state model. An access revocation may need near-immediate effect. A durable contractual grant may remain valid for years unless a specified terminating event occurs.

Freshness should follow the volatility and consequence of the governed fact.

That means "checked recently" is not a complete standard.

The system needs to know what kind of state it is checking, how that state can change, and what signals matter.

Model confidence is not standing

AI interfaces often expose probabilities, confidence levels, or verbal confidence.

These measures can be useful for uncertainty estimation. They should not be confused with authority.

A model can be 99 percent confident that a person would agree to an action. That does not equal consent.

It can be 99 percent confident that a manager intended a delegation to include a case. That does not necessarily create the delegation.

It can be 99 percent confident that a policy exception is sensible. That does not mean the model is authorized to grant it.

The difference is foundational.

Probability answers: **How likely is this proposition?**

Authority answers: **What legitimate source gives this system the right to cause this consequence?**

The questions can interact, but they are not interchangeable.

Historical truth versus present authority

One of the most important conceptual moves in this book is to allow history to remain true without allowing history to govern forever.

A person really did say yes.

A manager really did approve the action.

A policy really did permit the behavior.

A role really did include the privilege.

The system does not need to erase or reinterpret those facts after circumstances change.

It simply needs to represent their temporal standing correctly.

This is healthier than overwriting history because governance often depends on lineage. Investigators may need to know that an action was originally authorized and later revoked. Organizations may need to reconstruct when standing changed. Users may need recourse.

Temporal systems should preserve history while governing the present.

Authority can depend on evidence without becoming evidence

This does not mean authority floats free of evidence.

A standing determination often depends on evidence from multiple sources. The system may need a current identity, active delegation, unexpired grant, unchanged scope, valid account state, and no controlling revocation.

The point is that the architecture should distinguish the inputs from the determination.

A governance record might say:

- identity source: current and verified;
- consent source: current and affirmative;
- account state: eligible;
- policy version: current;
- requested consequence: within delegated scope;
- revocation check: clear;
- determination: preserved for action X;
- freshness: valid until condition Y or time Z, whichever occurs first.

That is much stronger than storing "AI approved."
It preserves why the action had standing.

The authority question

Whenever a system produces a persuasive explanation for why it should act, ask a final question:

Which part of this explanation is evidence, and which part is the actual source of authority?

If the answer is unclear, the system may be using intelligence to bridge a gap that governance should not allow intelligence to bridge.

A model can help us understand authority.

It should not be able to conjure authority from the eloquence of its understanding.

Policy decisions are evidence too

A policy engine can return `ALLOW` and still not settle every Temporal Standing question.

This may sound odd because policy engines are built to authorize actions. The distinction is one of scope.

Suppose a Cedar policy correctly determines that an authenticated employee is permitted to invoke `SendCustomerEmail` on a customer record in the present request context. That is an authoritative access-control decision for the policy domain it governs.

But the proposed email may also depend on whether the customer has withdrawn marketing consent, whether the campaign itself is still approved, whether the specific attachment is current, whether the recipient address was verified for this purpose, and whether another active opportunity makes the cold-outreach policy inapplicable.

The Cedar decision can be a governing input without being the entire standing determination.

The same is true for an identity token, a signed approval, a database record, or a model retrieval.

The question is always: **What proposition does this evidence authoritatively establish?**

A token may establish identity and scopes.

It may not establish purpose.

A consent record may establish permission for a category of processing.

It may not establish that a different recipient is authorized.

A policy engine may establish that the caller may invoke a tool.

It may not establish that the user's original instruction covered the tool's current payload.

An LLM summary may accurately state all of these facts.

It still does not become the authority that created them.

Source hierarchy and conflict

Real systems also face conflicting evidence.

A cached profile says marketing consent is active. A newer event says it was withdrawn. A CRM note says 'customer asked us to stop.' The consent service is temporarily unavailable. A human salesperson insists that the customer told them yesterday that contact is fine.

Temporal Standing does not solve such conflicts merely by naming them.

The institution must define source hierarchy, provenance rules, freshness requirements, and the competence of different actors to override or clarify a state.

This is an important limit of the framework.

Temporal Standing tells the architecture **where** a present authoritative determination is needed and **what transition** must be evaluated. It does not magically identify which source should govern every domain.

That work belongs to institutional governance.

The technical system must make the hierarchy executable.

Without that hierarchy, 'latest data wins' can be as dangerous as 'old approval wins.' The newest record may be less authoritative than an older signed decision. The most confident human may lack the authority to override a legal hold. A model may retrieve a recent document that was superseded by a formal policy with an older publication date but continuing effect.

Freshness and authority are related.

They are not identical.

A source can be authoritative and still be stale

Authority and freshness are separate dimensions.

An HR system may be the authoritative source for employment status. That does not make a six-hour-old replica automatically sufficient for a high-risk access grant after a termination event.

A consent registry may be authoritative. A cached copy may still be too old for a communication whose governing policy requires near-real-time suppression.

A policy repository may be authoritative. A worker may still be evaluating a superseded version.

This is where temporal governance becomes more precise than the phrase "use a trusted source." Trust answers *which source*. Freshness answers *which observation of that source*.

The system therefore needs at least three pieces of metadata around critical evidence: provenance, version or observation time, and an applicable freshness rule.

Freshness should not be one global timeout. Different predicates change at different rates and carry different consequences. A corporate charter may be stable for months. A fraud hold can change in seconds. A hotel rate may

expire in minutes. A user's marketing preference may be event-driven rather than time-driven: current until a revocation signal arrives, subject to a maximum staleness window if the source is unreachable.

The goal is not to make everything real time. It is to make the acceptable age of governing evidence explicit.

Conflicting authority is a governance problem, not a model debate

When sources disagree, asking the model which one "seems right" is usually the wrong abstraction.

Institutions need a hierarchy.

A customer-support note should not override the system of record for a legal hold. A model-generated summary should not override a signed approval. A cached policy should not override a newer authoritative policy version. A user's current explicit instruction may supersede an older preference if the system can establish that the same authorized principal issued it within the applicable scope.

These are institutional choices. They should be encoded outside the model so that identical evidence yields a stable authority resolution even when the model's prose or reasoning varies.

The model can help interpret evidence. It should not secretly invent the constitution under which evidence becomes authority.

PART III

APPLIED CASES

Chapter 9 - Consent That Changes

Consent is one of the clearest ways to understand Temporal Standing because people intuitively understand that yes can become no.

A person agrees to receive a message, share a piece of data, join a program, enable a feature, allow a recurring action, or participate in a process. Later, the person changes the decision.

Consent is not a tattoo

People change their minds. This occasionally surprises software. A consent record can be perfectly authentic and still describe a permission that no longer stands. Treating an old consent event as permanent is administratively convenient in roughly the same way that never updating a mailing address is administratively convenient.

The historical event remains real. The present consequence must still be justified.

The earlier yes remains part of the historical record. But the earlier yes does not necessarily authorize the later action.

This sounds obvious. In practice, systems struggle with it because consent is often stored as a field rather than modeled as a changing source of authority.

A record might say `marketing_opt_in = true`.

Where did that value come from?

When was it set?

What channels did it cover?

What categories of messages?

Was it tied to a specific purpose?

Can it expire?

Where is revocation recorded?

How quickly does revocation propagate?

What happens to already queued actions?

What happens to messages prepared but not yet sent?

What if one system sees the revocation and another still holds a cached copy?

These are Temporal Standing questions.

Consent is not a permanent property of the person

A dangerous mental model treats consent like an attribute:

This user is opted in.

A better model treats it as a relationship between a person, a purpose, a channel, a scope, and time.

The same person may consent to transactional email while declining promotional email. The same person may allow one type of data processing while refusing another. The same person may allow recurring reminders but later stop them.

The authority envelope matters.

This becomes especially important for AI systems because they can infer new uses for existing data. A traditional application may collect an email address for account recovery and use it only for account recovery because the code is narrow. A generative system can recognize that the same address could be useful for outreach, enrichment, identity resolution, recommendation, or personalization.

Capability expands faster than permission.

Temporal Standing therefore has two consent questions:

Does the original consent still stand?

And does it stand for this particular use?

The queued-action problem

Consider a marketing system that generates a month of outbound messages on Monday.

A customer opts out on Tuesday.

The Friday message was technically prepared before the opt-out. Does that matter?

The relevant consequence is not when the text was generated. It is when the message is sent.

Drafting and dispatch are different consequence surfaces.

At the dispatch boundary, the current consent state matters.

This is a general rule: preparing an action under valid standing does not guarantee that executing it later inherits that standing.

The same reasoning applies to notifications, recurring reminders, data exports, scheduled reports, and other deferred effects.

Revocation propagation

A system that supports revocation must make revocation causally meaningful.

A user interface that records "unsubscribe successful" while downstream workers continue to send from stale lists has acknowledged revocation socially but not operationally.

A mature architecture asks how the change reaches every relevant consequence boundary.

Possible mechanisms include event propagation, centralized current-state checks, short-lived permission artifacts, invalidation lists, versioned preferences, or other patterns. The exact implementation can vary.

The property should not.

A relevant revocation must be capable of defeating standing before the governed consequence occurs.

Silence and inferred consent

AI systems are especially capable of interpreting silence.

A model might infer that a person remains interested because they opened previous messages, continued using the service, or did not explicitly object.

That inference may be useful for prediction. Whether it can support consent depends on the governing rules of the context. Temporal Standing does not resolve the legal or ethical definition of consent in every domain.

It provides a discipline: do not substitute model inference for whatever source legitimately governs consent in the system.

If explicit current consent is required, inferred preference is not a replacement.

If the system's legitimate authority allows a different basis for the action, that basis should be represented honestly rather than relabeled as consent.

Changing minds is a feature of human agency

There is a deeper reason consent belongs near the center of this framework.

Human beings are not static profiles.

We change our minds. We leave relationships. We change jobs. We change risk tolerance. We become sick. We recover. We regret. We learn. We become more private. We become more open. We change financial priorities. We withdraw offers. We revise plans.

Systems that treat historical preferences as destiny are not merely inaccurate. They can become hostile to human agency.

A trustworthy AI should not say, in effect, "You once wanted this, therefore you still want it."

It should be able to represent discontinuity.

That does not mean forgetting history. History may be necessary for accountability and continuity. It means refusing to let the existence of history erase the legitimacy of change.

The design question

For every consent-mediated action, a builder should be able to answer:

What is the authoritative source of current consent?

What scope does it cover?

How can it change?

How is revocation represented?

How fast must revocation reach execution?

What happens to queued work?

What happens when the consent source is unavailable?

Can a model infer around missing consent, or must the system hold?

How is the decision recorded?

These questions transform consent from a checkbox into a living governance state.

The principle is simple:

A prior yes is not a permanent yes merely because the system remembers it.

Withdrawal illustrates historical truth without present authority

Privacy law provides one of the clearest illustrations of the distinction this book is trying to preserve.

The European Union's General Data Protection Regulation states that when processing is based on consent, the data subject has the right to withdraw that consent at any time, and that withdrawal does not retroactively invalidate processing that was lawful before withdrawal. The same regulation also states a purpose-limitation principle: personal data should be collected for specified, explicit, legitimate purposes and not further processed incompatibly with those purposes. [4]

This is not a claim that every AI action is governed by GDPR or that consent is the only lawful basis for processing. It is a conceptual example of temporal structure.

At T_0, consent can be real.

At ΔN, consent can be withdrawn.

At T_n, both of these propositions can remain true:

- the historical record correctly shows that consent existed; and
- that historical consent no longer supplies present grounds for a new consent-dependent consequence.

A weak system often handles withdrawal by deleting or overwriting the old record. That can create its own audit problems because the institution loses the historical truth of what authority existed when earlier actions occurred.

A stronger model preserves the event history while changing standing.

This event-oriented view also helps with purpose. A person may authorize one kind of use without authorizing a materially different downstream use. The problem is not always that consent "expired." Sometimes the proposed consequence has moved outside the original purpose tuple.

Temporal Standing therefore treats consent as one governing ground among several, with explicit scope and change events. The system does not need to collapse all human permission into a universal yes/no field.

Revocation must outrun the queued consequence

The operational challenge is propagation.

It is easy to build a beautiful preference center and still send the wrong message if downstream workers do not consult the changed state before execution.

The same principle appears in security protocols. OAuth 2.0 token revocation exists precisely because previously issued credentials sometimes need to become invalid before their natural expiry. RFC 7009 defines a

mechanism by which a client can notify an authorization server that a token is no longer needed, enabling invalidation of the token and, in applicable cases, related authorization grants. [5]

Consent withdrawal and token revocation are different domains. But they reveal the same systems lesson: **revocation is useful only if the consequence path can observe it in time.**

A revocation event trapped in one subsystem is not governance. It is data.

Chapter 10 - AI Agents and Tool Use

The commercial promise of AI agents is straightforward: give software a goal and let it do the work.

That promise depends on tools.

The enthusiastic intern problem

An agent with tools can resemble an exceptionally fast intern who never sleeps, never gets embarrassed, and interprets "take care of it" with unsettling enthusiasm. Give it `send_email`, `issue_refund`, `delete_record`, and `deploy_production`, and suddenly the org chart has developed a weather system.

The solution is not to make the agent timid. It is to separate what it can propose from what it is presently allowed to cause.

Without tools, an agent can plan, explain, draft, and recommend. With tools, it can change the world.

It can send an email, modify a CRM record, execute code, open a ticket, create a calendar event, book a hotel, issue a refund, update a permission, provision infrastructure, submit a form, or trigger another workflow.

This is where Temporal Standing stops being an abstract governance idea and becomes an engineering requirement.

Tool access is not tool authority

Many agent frameworks are built around a tool registry. The model receives descriptions of available functions and decides when to call them.

This is useful for capability orchestration.

It is incomplete for governance.

If a model can call `send_email`, the existence of the function does not answer whether it may send this email to this person for this purpose now.

If it can call `issue_refund`, that does not answer whether this account, amount, reason, and operator are currently authorized.

If it can call `delete_file`, that does not answer whether this file remains within the deletion scope approved earlier.

The tool definition describes what is technically possible.

Standing determines what is presently admissible.

Proposals before effects

A useful architecture treats model tool calls as proposals until governance converts them into executable actions.

The model might propose:

- tool: `send_email`
- recipient: `customer@example.com`
- template: `renewal_reminder_v3`

- purpose: `service_renewal`
- account: `A-1042`

The governance layer then asks whether the current evidence supports that exact consequence.

Is the recipient verified?

Is the communication channel currently allowed?

Is the account still in the eligible state?

Is the template approved for the stated purpose?

Has the customer replied in a way that changes the workflow?

Has an active opportunity, dispute, or suppression state changed what should happen?

Is the agent inside its quota or delegation?

Is the evidence current enough?

Only after those questions resolve to preserved standing should transport occur.

The important idea is that the model's proposal is not itself the permit.

Goal persistence versus authority persistence

Agent frameworks often emphasize long-running goals.

"Find me the best option and book it."

"Keep following up until the customer responds."

"Maintain service availability."

"Resolve outstanding invoices."

Long-running goals are useful, but they create a subtle hazard: the goal may persist longer than the authority supporting particular actions.

An agent can retain the objective while losing standing for one execution path.

A customer follow-up objective may remain legitimate while email standing is defeated by an opt-out. The agent might still be permitted to update internal notes but not contact the customer.

A travel goal may remain active while the budget changes, requiring a new approval before purchase.

An infrastructure reliability goal may remain valid while a specific remediation becomes inappropriate because the environment changed.

This distinction allows systems to remain intelligent without pretending every old plan remains executable.

Retries and idempotency

Agentic execution runs into the same distributed-systems problems as other software: timeouts, partial failures, duplicate messages, ambiguous provider responses, and retries.

Governance has to survive those problems.

Suppose an email provider times out. The system does not know whether the message was accepted. A naive retry may create a duplicate. An idempotency key can help prevent duplicate transport.

But idempotency does not solve standing.

If the system knows the original provider did not accept the message and decides to retry an hour later, the retry still needs current authority.

If the provider may have accepted the message, the system may need reconciliation before any new consequence is attempted.

This yields an important design rule:

Retry the missing dependency, not the consequence blindly.

If the missing dependency is transport certainty, reconcile transport.

If the missing dependency is current authority, redetermine standing.

If standing is defeated, do not let reliability logic resurrect the action.

Delegated agents

Multi-agent systems amplify the problem.

A supervisor agent may delegate research to one agent, drafting to another, and execution to a third. Each handoff can blur scope.

The supervisor may be permitted to propose a purchase but not execute it. If it asks another agent to "complete the task," the downstream agent must not infer that delegation created purchase authority.

Authority lineage should accompany execution lineage.

A downstream agent should know not simply that Agent A requested the action, but what authority Agent A possessed, which scope was delegated, what constraints travel with the delegation, and whether those grounds remain current.

The system should be able to answer: **where did the right to do this come from?**

Tool descriptions are not governance policy

Developers sometimes place safety instructions directly in tool descriptions:

"Only use this tool when the user has approved the transaction."

That is better than no instruction. It helps the model reason.

But if nothing outside the model verifies approval, the control is behavioral rather than causal.

The system is asking the model both to decide whether it is authorized and to exercise the capability that depends on that decision.

For low-consequence tasks, that may be an acceptable tradeoff. For higher-consequence systems, stronger separation is desirable.

The closer the action gets to money, sensitive data, access, legal effect, safety, or irreversible change, the stronger the case for an external execution gate whose decision cannot be bypassed by persuasive model output.

The consequence map

One practical exercise for agent builders is to map every tool by consequence.

Which tools only read?
Which tools disclose information?
Which tools communicate externally?
Which tools change state?
Which tools move money?
Which tools create obligations?
Which tools alter access?
Which tools can trigger other consequential systems?
Then identify the current-standing requirements for each class.
This is more useful than treating all tools as equivalent function calls.

The agent's new question

The next generation of reliable agent systems should not merely ask:
"What should I do next?"
They should also operate inside an environment that can answer:
"What am I presently authorized to cause next?"
Those are different questions.
Intelligence answers the first.
Governance answers the second.
Trustworthy agency requires both.

Capability is not standing

Agent architecture makes one distinction especially important: possessing a capability is not the same thing as possessing present authority for every use of that capability.

An agent may hold an OAuth token that allows it to send mail. The token can be valid. The mailbox can be healthy. The API call can be authorized. None of those facts proves that this message, to this recipient, for this purpose, still has standing.

This is where excessive agency becomes more than a permissions problem.

Security guidance often recommends reducing the tools and privileges available to agents. That is good engineering. A system should not give an agent database deletion rights if it only needs read access. Least privilege reduces the blast radius of failure.

But even perfectly scoped technical permissions can remain broader than the user's situational intent.

A travel agent may be permitted to book flights but not to substitute a different airport without confirmation.

A customer-service agent may be permitted to issue refunds but not to refund a charge after the account holder disputes the identity of the requester.

A recruiting agent may be permitted to email candidates but not to contact someone who opted out after entering the queue.

A deployment agent may be permitted to modify infrastructure but not after an incident commander freezes production changes.

The access-control layer can be working correctly while the consequence is still unjustified.

Long-running agents make time visible

Short chatbot interactions hide temporal problems because the instruction and the response are close together.

Long-running agents expose them.

An instruction can be queued Monday and executed Wednesday. A plan can be generated under one price and purchased under another. A workflow can begin while a user is an employee and continue after termination. A case can be prepared under one policy and submitted after the policy changes. A human can approve a draft and then edit a material field before the send occurs.

The longer the gap between instruction and consequence, the more room there is for ΔN.

This does not mean every agent needs continuous polling of every possible source. It means the architecture should know which governing predicates are material to which consequences and how fresh they must be.

For an email, recipient suppression may need to be current at send time.

For a payment, account destination and fraud holds may need very strong freshness.

For a reminder notification, a durable user preference may be sufficient until a revocation event arrives.

For a reversible internal draft, the standing requirement may be lighter because no external consequence has yet occurred.

Proposal is where probabilistic intelligence belongs

This leads to a productive division of labor.

Models are good at interpreting ambiguous instructions, generating options, drafting content, identifying likely next steps, and proposing actions.

Those are cognition functions.

They need not also be the final authority for consequence.

A model can propose:

'Book this flight.'

'Issue this refund.'

'Send this message.'

'Delete this record.'

'Escalate this account.'

A governance system can then evaluate whether the concrete proposal has present standing.

The value of this separation is not that deterministic systems are morally superior to probabilistic ones. It is that the institution can make the rules of

consequence explicit, testable, inspectable, and causally enforceable without forcing the model to become less capable at reasoning.

Broad cognition can remain broad.

Execution can remain bounded.

Modern agent guidance is converging on the consequence boundary

Current agent engineering guidance increasingly recognizes that side effects deserve different treatment from ordinary model generation.

OpenAI's agent safety documentation recommends approvals around sensitive tool operations and explicitly distinguishes tool-level checks from broader agent-level guardrails. Its current guidance notes that checks around side-effecting calls belong next to the tool that creates the effect. OWASP's guidance on excessive agency similarly recommends minimizing extension permissions, executing actions in the user's authorization context, requiring approval for high-impact actions, and enforcing authorization in downstream systems rather than relying on the language model to decide whether an action is allowed. [6][7]

These recommendations are compatible with Temporal Standing.

But approval alone does not settle the temporal question.

Suppose a human approves a $4,800 purchase. The workflow pauses for thirty minutes. During the pause, the cart changes to $5,400. Resuming the same run should not silently treat the old human approval as authority for the changed payload.

Or suppose a user approves sending a document to one recipient. A later model step resolves an ambiguous contact name to a different address. The system still has a human approval object. What it no longer has is an approval bound to the actual consequence.

This is why the book insists on **permit binding** rather than generic approval state.

The model can propose. A person or policy can approve. But the executor should be able to prove that the thing being executed is materially the thing that was evaluated.

Prompt injection is another reason authority must live outside the model

Prompt injection makes the separation even more important.

An agent may read untrusted content that tries to manipulate it into calling a sensitive tool, disclosing data, or changing its plan. Even a well-instructed model can make mistakes. If the model is also the final authority on whether the resulting action is permitted, the same compromised reasoning process both proposes and legitimizes the consequence.

That collapses separation of duties.

A stronger architecture allows the model to remain probabilistic at the proposal layer while a separate authority path evaluates structured facts about the proposed effect. The governance layer does not need to decide whether the model's prose was wise. It needs to determine whether the exact proposed action is supported by current authoritative grounds.

This does not eliminate prompt injection. It changes the blast radius.

An injected instruction may distort a proposal. It should not automatically mint the authority required to execute it.

Long-running agents need authority leases, not immortal goals

A long-running agent can hold a goal for days or weeks. Goals are useful precisely because they persist.

Authority should not necessarily persist at the same granularity.

A useful mental model is to give the agent a durable **mission** but short-lived or event-invalidated **authority leases** for consequential acts. The mission can remain: reconcile the account, schedule the trip, manage the campaign, keep the service healthy. The permission to send this message, pay this invoice, change this role, or deploy this build can be re-established when needed.

This separates continuity of purpose from continuity of permission.

It also prevents a dangerous inference: "Because I am still pursuing the same goal, everything that was allowed earlier remains allowed now."

Humans do not reason that way in mature institutions. A standing mission does not exempt an employee from current budget limits, sanctions lists, access controls, approval thresholds, or revised instructions.

Agents should not receive a broader metaphysical privilege merely because they are software.

Chapter II - High-Stakes Domains

Temporal mistakes become more serious as consequences become harder to reverse.

A stale preference that changes the order of a music playlist is inconvenient. A stale authority state that changes access to money, care, employment, legal rights, education, or public services can be far more consequential.

Stakes change the tolerance for improvisation

If an AI recommends the wrong sandwich, we have lunch drama. If it acts on stale authority in healthcare, finance, employment, or legal operations, the consequences are less likely to be resolved with extra pickles.

The framework does not become different in high-stakes domains. The evidence standards, review paths, and cost of getting standing wrong become much stricter.

This chapter does not attempt to prescribe legal, clinical, financial, or regulatory rules. Those rules vary across jurisdictions, institutions, and use cases and require domain expertise.

The purpose is narrower: to show why high-stakes systems need an explicit model of present standing.

Health

Imagine an AI-assisted workflow that reviews a patient's information, prepares a recommendation, and later triggers a downstream action.

The relevant clinical facts may change between preparation and execution. A medication list may be updated. A new allergy may be recorded. A test result may arrive. A clinician may revise the plan. The patient may withdraw consent for a particular workflow.

The fact that the original recommendation was reasonable at T_0 does not prove that the downstream action remains appropriate at T_n.

Temporal Standing does not tell the clinician what treatment is correct. It tells the system to recognize that the authority and evidence supporting an action can change before the action occurs.

The consequence boundary should be defined accordingly.

Finance

Financial systems already understand time deeply: balances change, prices move, orders expire, mandates have limits, accounts are frozen, identities require verification, approvals have thresholds.

AI agents introduce new combinations of these states.

A user may ask an agent to purchase an asset if the price falls below a threshold. The agent may generate the plan at one moment and execute later. A transfer may be approved for one recipient, while the recipient

details later change. A business payment may be valid in purpose but exceed a current delegation limit.

Temporal Standing asks whether the exact proposed transaction still fits the current authority envelope.

This can be especially important when a model is capable of improvising. A creative workaround that achieves the financial objective may still be outside the user's authorization.

Law

Legal workflows are full of deadlines, roles, scope, versioned documents, client instructions, court rules, and changing case facts.

An AI system may draft a filing under one factual record and later submit after new information arrives. It may prepare communication based on a client's prior instruction after the client changes direction. It may retain an outdated version of a policy, contract term, or procedural rule.

Temporal Standing again does not decide the law. It structures the governance question: what authoritative sources must be current immediately before the consequential step?

Drafting can be separated from filing.

Research can be separated from representation.

Preparation can be separated from submission.

Those boundaries are opportunities for current determination.

Hiring and employment

An organization may authorize an AI system to source candidates, schedule interviews, draft messages, evaluate structured information, or support internal workflows.

The relevant state can change quickly.

A role can close. Requirements can change. A candidate can withdraw. A hiring manager can change the process. An internal employee can become ineligible for a workflow because of a role transition. A communication preference can change.

The more consequential the action, the more important it becomes to separate “the workflow once supported this” from “the workflow currently supports this.”

Employment systems also illustrate why historical data should not become destiny. A person can correct a record. A prior status can change. A temporary condition can end. Systems need mechanisms for current authoritative state rather than treating all historical signals as permanently controlling.

Education

Educational systems may recommend courses, trigger interventions, communicate with students or families, or make decisions based on records that evolve.

A student's enrollment, accommodation, guardian relationship, program status, or consent can change. A system acting on old records can produce consequences that no longer match the current relationship.

Again, Temporal Standing does not define educational policy. It insists that policy-relevant state be current when consequences occur.

Public services

Government and public-service systems can affect benefits, notices, eligibility, access, and obligations. These systems often operate at scale and depend on records from multiple institutions.

Scale makes stale standing especially dangerous because a defect can propagate broadly.

If a system relies on prior eligibility, prior address, prior representative authority, or prior consent, there must be a way to represent changes that matter before action.

Public systems also highlight the importance of recourse. A person should be able to correct an outdated or mistaken state and have that correction reach future consequences.

Consequence should shape freshness

High-stakes domains make one design principle clear: freshness is not just an optimization parameter.

The necessary freshness of evidence should reflect how quickly the governing state can change and how serious the consequence is.

A system may tolerate a relatively old organizational directory entry for a low-risk internal suggestion. It should not automatically use the same freshness assumption for a high-value financial transfer or a sensitive disclosure.

The consequence changes the burden of currentness.

Human review and current standing

High-stakes systems often rely on human review. This is sensible but incomplete if the human review occurs too early.

A human may approve a plan at T_0. The world changes. The system executes at T_n.

If the approval was conditioned on facts that are no longer current, the human's presence in the workflow does not solve the problem.

Review should be placed where it can govern the consequence, or the system should preserve enough conditions to know when the approval remains valid.

The principle across domains

The details differ enormously across medicine, finance, law, employment, education, and public services.

The shared temporal pattern is simple:

A legitimate prior state supports an action.

Relevant reality changes.

The system still possesses the old record.

The old record is treated as present authority.

A consequence occurs.

Temporal Standing exists to break that chain before the consequence.

In high-stakes domains, the question is not whether systems should remember history.

They must.

The question is whether they can distinguish **history worth preserving** from **authority that still stands**.

High stakes should change the proof burden, not only the warning label

Many governance approaches become louder as risk rises. They add warning banners, explanations, review screens, or attestation language.

Those controls can help, but Temporal Standing asks a more structural question: **what additional predicates must be current before this consequence can occur?**

The answer can vary by domain and institution.

A high-value financial action may require fresher destination verification, stronger identity assurance, dual approval, or a transaction-specific limit check.

A clinical workflow may require current patient identity, current order state, current contraindication data, and a qualified human decision at the appropriate point. The framework does not decide medicine; it makes the dependency on current governing facts explicit.

A hiring workflow may require that the position remains open, that the decision maker still has authority, that the candidate record is current, and that any required review has actually occurred before an external rejection or offer is sent.

A public-service workflow may need current eligibility data, current legal authority, and meaningful recourse before an adverse consequence.

The common pattern is not that every high-stakes action should be manually approved. It is that the system should make the required current grounds legible and enforce them proportionately to consequence.

Human oversight is not identical to human presence

The EU AI Act's human-oversight provisions for high-risk AI systems are useful here because they emphasize effective oversight during use and make the measures proportionate to risk, autonomy, and context. The law also contemplates technical measures that enable humans to understand when and how to intervene. [2]

Temporal Standing adds a narrower systems observation: a human can be "in the loop" and still approve against stale or incomplete governing state.

Human presence does not automatically cure weak architecture.

If the reviewer sees yesterday's price, the wrong recipient, a superseded policy, or a summary that hides a revoked permission, the human may faithfully approve the wrong consequence. In that case, the problem is not lack of human oversight but poor evidence composition at the point of oversight.

The stronger design is to put the human at a boundary where the exact consequence and the current material predicates are visible, and to bind the resulting approval to that consequence.

Oversight then becomes part of the authority chain rather than a ceremonial click.

Chapter 12 - Enterprise Operations

Enterprise AI will not arrive as one giant autonomous brain.

It is arriving inside ordinary operations: customer service, sales, procurement, finance, human resources, IT, security, legal operations, vendor management, knowledge work, and internal administration.

Enterprise software already has enough ghosts

Every large organization eventually discovers accounts belonging to people who left, workflows owned by teams that no longer exist, approvals tied to policies nobody remembers, and one spreadsheet that apparently became critical infrastructure in 2019. Adding autonomous agents does not make those ghosts disappear. It gives them API access.

Enterprise Temporal Standing begins by making delegated authority and change legible at execution time.

That makes Temporal Standing an enterprise architecture problem.

Most companies already have systems of record, approval chains, role directories, policies, queues, scheduled jobs, and audit logs. AI agents sit on top of this environment and gain the ability to interpret, coordinate, and act across systems that were previously separated.

The opportunity is enormous.

So is the possibility of carrying stale authority across boundaries.

Access control

An employee is granted access because of a role. The employee later transfers teams. The identity provider updates quickly, but a long-running agent session or cached entitlement does not.

The technical question is whether the old privilege can still be exercised.

The Temporal Standing question is broader: which actions are currently supported by the employee's role, delegation, purpose, and context?

An AI agent operating “for” an employee should not inherit more durable authority than the employee themselves possesses.

If the employee's standing changes, the agent's authority should be capable of changing with it.

Procurement

Procurement is full of temporal dependencies.

Prices change. Vendors change status. bank details change. approval thresholds change. budgets close. contracts expire. sanctions or restricted-party status can change. delivery conditions move.

An agent that optimizes procurement using an old snapshot may make a rational decision under obsolete conditions.

The answer is not to eliminate automation. It is to move current checks to the consequences that matter: order placement, payment, contract commitment, vendor onboarding, and sensitive disclosure.

Customer service

Customer service agents may be allowed to issue refunds, credits, replacements, account changes, or escalations.

Standing can depend on account status, transaction history, policy, fraud indicators, customer identity, and delegated limits.

Suppose an AI prepares a refund, then another agent or human resolves the issue through a replacement. If the first agent later executes its prepared refund without refreshing account state, the organization may create a duplicate consequence.

This is not merely a concurrency bug. It is a standing failure: the facts that justified the refund changed.

HR workflows

Human resources systems manage sensitive, changing relationships.

An employee's manager changes. Leave status changes. accommodations change. employment ends. a complaint enters a protected process. a hiring requisition closes. a candidate withdraws.

An AI that continues to send, schedule, classify, or disclose based on yesterday's organizational state can create serious mistakes.

HR automation therefore benefits from explicit current-state boundaries rather than broad “employee workflow” permissions.

Incident response

Incident-response automation is often designed to move quickly.

Speed is valuable when systems are under attack or failing. Yet incident state can change minute by minute.

An agent may decide to isolate a host at T_0. Before execution, a human operator may have restored service, migrated workload, or identified the alert as benign. The original remediation can become harmful.

Fast systems need Temporal Standing more, not less, because the cost of stale action can be immediate.

One solution is to define short authority lifetimes for high-impact remediations and require current evidence at execution. Another is to establish narrowly scoped emergency authority with explicit conditions.

The principle remains: urgency does not erase time.

Approval chains

Enterprises love approval chains because they create a visible governance process.

But approval chains often approve objects that later mutate.

A budget request is approved, then the amount changes.

A contract is approved, then terms change.

A campaign is approved, then the audience changes.

A deployment is approved, then the artifact changes.

If the execution system checks only that "approval exists," it can execute a materially different object under a historical approval.

A stronger pattern binds approval to the relevant version, scope, or fingerprint of the thing approved.

If material components change, standing must be redetermined.

Organizational authority is dynamic

Enterprises are continuously reorganizing.

People join, leave, change roles, gain temporary delegation, lose temporary delegation, go on leave, hand off responsibilities, and operate under changing policies.

An AI system that acts for the enterprise needs an authority model as dynamic as the enterprise itself.

This is one reason broad static prompts are insufficient:

"You are the company's autonomous procurement agent. Follow company policy."

Which policy version?

Whose budget?

Which thresholds?

Which vendors?

Which exceptions?

Which current approvers?

What changed this morning?

The prompt expresses aspiration. The environment must provide current authority.

The standing graph

Large organizations may eventually need to think in terms of a standing graph.

A consequential action depends on multiple nodes:

identity,

role,

delegation,

policy,

asset state,

account state,

consent,

budget,

recipient,

purpose,

and other domain-specific conditions.

Changes to any relevant node can alter standing.

Not every system needs a literal graph database. The concept is useful because it shows that enterprise authority is relational and distributed.

A model cannot safely infer its way through missing authoritative edges.

Operational governance

The enterprise test is simple to state:

If a relevant authority changes at 10:00 a.m., which AI-driven actions at 10:01 are now different?

Can the system identify them?

Can queued work see the change?

Can active agents see the change?

Can downstream tools refuse stale permits?

Can the company reconstruct why an action proceeded?

Can a human correct the state without editing five disconnected systems?

If the answer is no, the enterprise may have policies without temporal governance.

AI raises the stakes because agents can move across organizational boundaries faster than traditional workflows.

The solution is not to slow the enterprise to human speed.

It is to make authority move at machine speed too.

Enterprise security already supplies important pieces

Enterprise architects do not need to abandon mature access-control systems to adopt Temporal Standing.

NIST defines attribute-based access control as authorization based on attributes of subjects, objects, requested operations, and, in some cases, environmental conditions evaluated against policy. Zero Trust architecture rejects implicit trust based merely on network location or ownership and centers explicit authentication and authorization around resources. Modern policy languages such as Cedar ask whether a particular principal may perform a particular action on a resource in a given context. [8][9][10]

Those are powerful primitives.

They also reveal why the standing question should be composed rather than reinvented.

For an enterprise agent attempting to change payroll data, an ABAC or Cedar decision may establish that the acting principal currently has the technical permission to perform the operation. That does not necessarily establish that the specific change was requested by the employee, approved by the right manager, consistent with the current payroll cycle, or within the purpose of the agent's task.

Conversely, a manager's approval may be perfectly current while the employee who supplied it has lost the system role required to execute the mutation.

Temporal Standing does not replace either ground. It asks whether the required set is jointly sufficient now.

Standing graphs make hidden dependencies visible

Large organizations often have no single record called "authority." Instead there is a graph:

- identity comes from one system;
- role membership from another;
- customer preference from another;
- policy from a repository;
- legal or compliance constraints from a specialized service;
- transaction state from an operational database;
- human approval from a workflow platform;
- and the proposed action from the agent runtime.

The consequence is legitimate only if the relevant edges still connect.

A standing graph is a useful design artifact because it forces teams to name these dependencies. For each consequence class, architects can identify the source of each predicate, how change is signaled, how freshness is established, what conflicts are possible, and what the executor must verify.

This often exposes organizational problems that no model can solve: two departments believe different systems are authoritative; revocation is propagated to one application but not another; a human approval has no payload identity; an old integration bypasses the policy gateway.

In that sense, Temporal Standing is partly a diagnostic tool for institutional plumbing.

Chapter 13 - The Human Case

Temporal Standing sounds technical because it can be expressed through systems, permissions, events, freshness windows, and execution gates.

But the framework is ultimately about something deeply human: the right to change.

Humans remain inconveniently alive

People recover, relapse, reconsider, forgive, revoke, move, resign, reconcile, change priorities, and occasionally decide that the thing they definitely wanted Tuesday was a terrible idea by Friday. A system trained to treat historical data as destiny will miss a basic feature of human life: we continue happening.

Temporal Standing is partly a technical discipline for respecting that fact.

People change their minds.

They recover from mistakes.

They leave jobs.

They end relationships.

They forgive.

They withdraw.

They learn.

They age.

They become responsible for new things and stop being responsible for old things.

They move from one context to another.

They give permission and later revoke it.

A system that cannot represent these transitions can become oppressive without anyone intending it to be.

Historical data as destiny

AI systems are unusually good at using history.

That is one of their strengths. Past behavior can help predict future needs. Earlier conversations can provide continuity. Previous choices can reduce repetitive setup. Institutional records can make services more efficient.

But useful memory becomes dangerous when historical state is treated as destiny.

“You chose this before” becomes “you still choose this.”

“You permitted this once” becomes “you permit this generally.”

“You were in this category” becomes “you remain in this category.”

“You failed before” becomes “you are a failure risk.”

“You disclosed this fact” becomes “this fact may be reused forever.”

Temporal Standing creates a conceptual brake on this slide.

It says history can be remembered without being allowed to rule every present consequence.

The right to revise

Human agency includes the ability to revise prior decisions.

That does not mean every commitment is revocable without consequence. Contracts, obligations, laws, and institutional rules can create durable commitments. Temporal Standing does not erase them.

It asks the system to determine what currently governs.

Sometimes the current answer will be that the prior commitment still stands despite a person's changed preference.

Sometimes the answer will be that the person had and exercised a right to revoke.

Sometimes the answer will be unresolved and require a legitimate decision-maker.

The framework is neutral about the substantive rule. It is demanding about temporal honesty.

Do not pretend the old state is automatically the current state.

Correction

People also need the ability to correct records.

An AI system may act on a factual claim that was once entered incorrectly, later corrected, but still persists in a cache, embedding, summary, or model memory.

Correction is therefore a Temporal Standing event.

The historical record may preserve that the old claim existed. But future consequences should be governed by the corrected authoritative state where appropriate.

This creates a practical challenge for AI memory systems.

Memory can improve personalization, but a memory architecture must represent supersession. A new preference should be able to defeat an old preference. A correction should not merely sit beside the error while both remain equally retrievable.

Time needs semantics.

Context changes meaning

Human statements are contextual.

A person can say, "Always remind me," during a project and mean always during the project.

A manager can say, "Use your judgment," within a narrow operational context and not mean unlimited discretion.

A patient can consent to one kind of sharing for one episode of care and not every future disclosure.

Natural language often relies on shared context to bound meaning.

AI systems that persist instructions across contexts can strip those boundaries away.

Temporal Standing forces the system to ask what context made the earlier instruction authoritative and whether that context remains relevant.

Dignity and recourse

A trustworthy system should allow people to contest the present state that governs them.

If an AI denies an action because standing appears defeated, can the person correct the underlying record?

If an AI continues an action because standing appears preserved, can the person revoke or narrow the authority?

If standing is unresolved, is there a path to resolution?

Can the person understand which source controls the outcome?

Recourse matters because governance without recourse can freeze mistakes.

A fail-closed system is safer against unauthorized action, but if it has no path to correct an erroneous denial, it can become rigid and unfair.

Temporal Standing should therefore be paired with mechanisms for correction, renewal, appeal, reauthorization, and human determination appropriate to the context.

Institutional memory

The human case also applies to organizations.

Institutions remember. They develop policies, precedents, risk rules, customer histories, and operational habits.

Institutional memory can be valuable. It can also become inertia.

"We have always done it this way" is the organizational equivalent of stale standing.

A mature institution preserves lineage while allowing legitimate change to become authoritative.

AI can either improve that ability or make inertia faster.

If an AI is trained or configured on yesterday's institutional wisdom without mechanisms for current authority, it may automate obsolete practice with extraordinary efficiency.

Technology that recognizes change

The most humane promise of Temporal Standing is not that machines will become morally wise.

It is more concrete.

We can build systems that recognize that a human being is not identical to their last recorded state.

We can build systems that let revocation matter.

We can build systems that let correction matter.

We can build systems that distinguish a past commitment from a present permission.

We can build systems that know when they do not know.

We can build systems that preserve history without forcing people to live inside it forever.

That is not only good architecture.

It is a form of respect.

PART IV

DESIGNING SYSTEMS THAT RECHECK BEFORE ACTING

Chapter 14 - The Consequence Boundary

Every governed system needs a place where policy becomes causality.
That place is the consequence boundary.

Put the guardrail where the cliff is

A governance review three screens upstream is comforting. A policy document in a shared drive is also comforting. Neither stops the transaction if the actual execution path ignores them.

The consequence boundary is where the system stops discussing the world and begins changing it. That is where a determination has to matter causally.

The consequence boundary is the point immediately before a system creates a meaningful external effect. It is where a proposal becomes an event in the world.

A message leaves the system.
Money moves.
A reservation is created.
A record changes.
Access is granted or revoked.
A deployment begins.
A filing is submitted.
A command reaches a machine.
A downstream workflow is launched with authority to act.

The purpose of defining this boundary is not philosophical elegance. It is enforcement.

If the system determines standing long before the consequence and never checks whether relevant conditions changed, the determination can become stale before it matters.

If the system determines standing after the consequence, governance becomes retrospective.

The standing decision therefore belongs as close as practical to the point where the external effect becomes unavoidable.

One workflow, many boundaries

A single agentic workflow may contain several consequence boundaries.
Consider an agent planning a business trip.
Searching public flight information may be low consequence.
Reading a private calendar crosses an access boundary.
Sending an itinerary to a colleague crosses a disclosure and communication boundary.
Purchasing a ticket crosses a financial and contractual boundary.
Writing the confirmed travel details into an enterprise system crosses another state-change boundary.
Each boundary can require different standing.

This matters because "the user authorized the trip" is too broad to govern all of them automatically.

The user may authorize searching without authorizing purchase. The user may authorize purchase under a ceiling but not disclosure to colleagues. The employer may authorize booking through one provider but not another. The travel policy may change between planning and checkout.

Governance works best when it follows consequences rather than sessions.

The last responsible check

System designers often ask where to place validation. Too early and it becomes stale. Too late and the action may already be irreversible.

The useful idea is the **last responsible check**: the latest point at which the system can still prevent the consequence while possessing enough information to evaluate the exact proposed action.

For an email, this may be immediately before the provider send request.

For a payment, immediately before submission to the payment rail.

For a database mutation, immediately before the write.

For a deployment, immediately before the release command crosses into the target environment.

The system can and should perform earlier checks too. Early checks improve efficiency by rejecting obviously invalid actions before expensive work occurs. But an early check should not be mistaken for current execution authority if material facts can change later.

Binding the determination to the action

A standing determination should be bound to the consequence it evaluated.

That can include a hash or version of the payload, recipient, amount, target, policy version, evidence snapshot, action type, purpose, and other relevant fields.

Why bind it?

Because otherwise the system can obtain permission for one thing and execute another.

The mismatch may be accidental. A model revises the draft after governance. A price updates. A worker selects a new recipient. A downstream service enriches the payload. A retry uses current data rather than the data originally evaluated.

If the material action changes, the permit should no longer be presumed to cover it.

This is the same reason signatures bind people to particular documents rather than to "whatever document happens to be sent later."

Consequence surfaces

Some actions have obvious boundaries. Others create indirect consequences.

A system may write a record that another automated service immediately consumes. A recommendation may enter a queue that triggers execution without further review. An agent may delegate a task to another agent with broad tools.

In these cases, the first action can itself be consequential because it creates causal authority downstream.

A practical design exercise is to trace the **consequence surface**:

What can this action cause directly?

What can it cause indirectly?

Which downstream systems treat this output as authority?

Can the original action be reversed before downstream effects occur?

Where is the last point at which standing can still govern the chain?

This analysis often reveals that a seemingly "internal" write is actually an execution boundary because another system trusts it automatically.

Fail closed where it counts

Fail-closed design means that absence of a valid permit prevents the governed consequence.

This is stronger than logging that a permit was missing.

It is stronger than asking the model to refrain.

It is stronger than displaying a warning to an operator who may ignore it.

At the boundary, the causal path should depend on a valid current determination.

That does not require a single centralized service. Enforcement can be distributed. Different tools can verify permits. Gateways can apply policy. Database procedures can enforce state. Connectors can require authorization artifacts.

What matters is that the execution path cannot simply route around governance without becoming an explicit exceptional path with its own authority.

Performance and availability

Builders reasonably worry that pre-action governance adds latency and dependency risk.

It can.

But the solution is architecture, not omission.

Standing checks can be designed for speed. Durable permissions can use explicit validity intervals. Revocation can invalidate otherwise cached permits. Low-risk actions can use lighter controls. High-risk actions can

require stronger current evidence. Local enforcement can reduce network hops. Redundant authoritative sources can improve availability.

The point is proportionality.

A system does not need the same governance burden to reorder a dashboard as it does to move money.

But when a consequential action depends on current authority, the architecture should make that dependency real.

The boundary question

For every tool, workflow, and agent, builders should be able to answer:

Where, exactly, can this consequence still be stopped?

Then ask:

What current standing must exist at that point?

If those questions do not have concrete answers, the system may have governance language without a governance boundary.

The boundary should be as late as necessary and as early as useful

'Immediately before execution' is a principle, not a demand for a mathematically zero interval.

Distributed systems cannot eliminate time.

A standing determination takes time. A permit takes time to reach an executor. Networks delay packets. Databases replicate. External providers can change state between validation and effect.

The engineering goal is therefore to place the determination close enough to the consequence, and bind it strongly enough to the proposed action, that the remaining race window is proportionate to the risk.

For a low-risk internal notification, a short-lived cached decision may be adequate.

For a securities trade, high-value payment, destructive infrastructure operation, or sensitive disclosure, a stronger pattern may be needed: executor-side validation, current version checks, transactional state comparison, or a single service that performs determination and execution under one controlled boundary.

This is not unique to AI. Security systems have long dealt with time-of-check/time-of-use problems. Temporal Standing gives the problem a governance interpretation: if a material ΔN can occur between determination and consequence, the architecture must decide how much drift is acceptable.

Consequences can nest

A workflow can cross more than one consequence boundary.

Imagine an agent asked to 'resolve the vendor issue.' It drafts an email, sends the email, creates a purchase order, changes a bank destination, submits payment, and closes the ticket.

These are not one consequence merely because they serve one goal.

The draft may be internal and reversible.

The email affects an external party.

The purchase order creates a commercial commitment.

The bank change alters destination authority.

The payment transfers value.

The ticket closure changes operational state.

Each material step can depend on different governing grounds.

A permit to send the email should not silently become a permit to change the bank account. A permit to create a purchase order should not automatically authorize payment if a fraud hold appears afterward.

This is why the consequence boundary is better understood as a series of checkpoints around meaningful external effects rather than one ceremonial gate at the beginning of a workflow.

Reversibility changes the control

Not all consequences deserve equal friction.

A system that treats drafting text and wiring money as equivalent will either be unusably restrictive or irresponsibly permissive.

Reversibility is one useful factor in proportional governance.

A draft can usually be discarded.

A calendar proposal can often be edited before invitation.

A sent message can sometimes be corrected but not unsent from a recipient's mind.

A disclosed secret cannot be undisclosed.

A paid invoice may be recoverable, but not reliably.

A medical order, account deletion, public filing, or production change may create escalating costs of reversal.

As consequence becomes harder to reverse, the value of fresh standing generally rises.

The framework therefore aims at minimum unjustified consequence, not maximum interruption.

Causal ordering matters when authority changes

Distributed systems add a subtle version of the Temporal Standing problem: the right facts can exist and still be observed in the wrong order.

Google's Zanzibar authorization system was designed so that authorization decisions respect causal ordering amid changing access-control lists and object contents. That is important because a revocation followed by a new access attempt should not be casually reordered into an apparent permission merely because replicas observe updates at different times. [11]

Temporal Standing inherits the same intuition.

If ΔN defeats or changes a governing predicate, a downstream executor should not resurrect T_0 merely because it reads an older replica or cached snapshot.

This is especially important in agent systems because state can cross many services: model runtime, orchestration service, policy engine, database, connector, provider API, retry queue, and audit store. The more components involved, the easier it is for one to see a temporally older world than another.

A robust design therefore needs an ordering strategy proportionate to the action. That may be a version number, event sequence, causal token, database transaction, compare-and-swap condition, signed state revision, or another mechanism that lets the executor detect that its authority view has been superseded.

The framework does not prescribe one distributed-systems technique. It insists that the temporal relationship between authority-changing events and execution cannot be ignored.

Time of check and time of use is governance too

Security engineering has long recognized time-of-check/time-of-use races: a condition can be validated, then change before the action uses it.

The same race appears in governance.

An agent can obtain a policy allow at 12:00:00, mutate the payload at 12:00:01, and send at 12:00:02. Or a bank destination can be verified, then replaced before payment. Or a human can approve a deployment revision that is not the revision ultimately released.

The answer is not always to make the interval zero. That is impossible in distributed systems.

The answer is to bind the determination to enough state that meaningful drift becomes detectable.

This is why a standing permit should carry identity: payload hash or canonical fields, policy version, relevant evidence versions, expiry, principal, purpose, and consequence class as appropriate. The executor verifies the identity before creating the effect.

The remaining race window becomes an engineering quantity rather than an unexamined assumption.

Chapter 15 - Fresh Determination, Not Permanent Permission

One of the easiest governance mistakes is to turn a successful decision into a durable badge.

The system asks whether an action is permitted. The answer is yes. From that point forward, the workflow behaves as though permission has become a property of the task.

Permission does not improve with age

Some software behaves as though an approval becomes more authoritative the longer it survives in cache. It does not. Old wine may improve in a cellar. Old authorization generally needs a different evaluation rubric.

Fresh determination does not mean permanent reapproval. It means the material predicates for this consequence are current enough to rely on.

But a determination is only as broad as the question it answered.

If the question was, "May this message be sent to this recipient under these conditions now?" the answer does not automatically become, "May this workflow continue sending related messages indefinitely?"

A current permit is not permanent permission.

Decision scope

Every standing determination should have decision scope.

It should be possible to answer:

Which action did the determination cover?

Which target?

Which payload or bounded payload class?

Which purpose?

Which authority source?

Which evidence state?

Which policy version?

Which time or event limits?

Which downstream use, if any?

This does not mean every implementation needs a giant permit object. It means the architecture must avoid ambiguous reuse.

Freshness can be event-based

People often think of freshness as a timer.

"Recheck every five minutes."

Time windows can be useful, but they are only one mechanism.

Some standing should remain valid until an event occurs.

A role grant may be valid until revocation or expiration.

A price approval may be valid until the quoted price changes or until a deadline.

A communication permission may be valid until preference changes.

A deployment approval may be valid only for a particular artifact fingerprint.

An account-state determination may be valid until a relevant write changes the account.

Event-based invalidation can preserve efficiency while remaining sensitive to change.

The strongest systems often combine time and events: a permit expires after a defined window and can also be invalidated earlier by relevant changes.

Generation is not execution

AI systems make the distinction between generation and execution especially important.

A model may prepare an action while standing is preserved. Later, the prepared output may be executed after standing changes.

The preparation does not freeze authority.

A generated email can become stale.

A drafted transaction can become stale.

A recommended remediation can become stale.

A travel itinerary can become stale.

A proposed database change can become stale.

This leads to a useful rule:

A valid generation-time determination does not replace an execution-time determination when the consequence occurs later.

Early governance can still matter. It can prevent the system from preparing prohibited actions or accessing unauthorized data. But execution remains a distinct boundary.

Permanent certification is the wrong abstraction

Organizations understandably want to certify systems.

"This agent is approved."

"This workflow passed governance review."

"This vendor is authorized."

"This model is safe for production."

These statements can be useful organizationally. They should not be confused with a standing determination for each consequential act.

A certified payment agent can still encounter an unauthorized payment.

An approved workflow can still receive out-of-scope input.

A vetted vendor can still change status.

A tested model can still operate under stale permissions.

System certification answers one class of question. Current standing answers another.

Chained workflows

Permanent-permission thinking becomes particularly dangerous in chains.

Action A is permitted.

Action A triggers Action B.

Action B triggers Action C.

By the time C occurs, the system may be several causal steps removed from the original authority.

If each downstream action assumes that upstream permission automatically travels with the workflow, scope can expand invisibly.

A better architecture treats authority as something that must be explicitly propagated and bounded.

Action B may inherit a subset of A's standing if the delegation is legitimate. Action C may require a new determination because it introduces a new consequence.

This is similar to capability security: possession of one capability should not magically imply possession of every capability useful for completing a goal.

Reauthorization

When standing expires or is defeated, the system may request fresh authority.

Good reauthorization is specific.

Instead of asking, "Do you approve?" the system should communicate what changed and what consequence now requires approval.

"The original approval covered a total price up to $2,000. The current total is $2,240. Approve purchase at the new amount?"

"The original recipient list contained three people. Two recipients were added. Approve sending the attached document to the updated list?"

"The workflow was approved for version 4. The artifact is now version 5. Approve deployment of version 5?"

This makes ΔN visible to the human rather than asking for ceremonial confirmation.

The anti-rubber-stamp principle

Human approval becomes weak when the system repeatedly asks for confirmation without explaining change.

Users learn to click yes.

Temporal Standing can reduce unnecessary prompts by preserving valid authority efficiently and only requesting reauthorization when relevant standing has changed or become unresolved.

That is a better user experience than constant interruption and a safer architecture than blanket permission.

The objective is neither permanent approval nor permanent friction.

It is **continuous legitimacy**.

The system should keep working as long as the grounds still stand, and it should know when they no longer do.

Durable authority is allowed

A common misunderstanding is that fresh determination requires fresh human approval.

It does not.

An authority can be intentionally durable.

A user may authorize automatic monthly payment until revoked. A corporation may delegate purchasing authority within a defined amount for the duration of employment. A service account may have continuing authority to rotate logs. A patient may establish a durable preference under conditions recognized by the relevant system.

Temporal Standing does not erase durability. It asks the system to preserve the terms of durability as part of T_0 and to recognize the ΔN events that can end or narrow it.

This makes a useful distinction between **fresh evidence** and **fresh consent**.

A system may need fresh evidence that a durable grant remains unrevoked without asking the human to grant it again.

That is how useful automation remains possible.

Freshness is a property of predicates

Different governing predicates age at different rates.

Identity might be valid for a session.

Employment status might need reevaluation after an HR event.

A market price may become stale in seconds.

A mailing preference may be durable until changed.

A sanctions list may have a defined update cadence.

A bank destination may require verification whenever it changes.

A signed contract may remain authoritative for years while a temporary delegation inside it expires in thirty days.

It is therefore dangerous to give an entire workflow a single global freshness timestamp.

The stronger design associates freshness with the material predicate or evidence source.

At T_n, the standing engine asks whether each governing predicate is sufficiently current for the consequence it supports.

That produces a more precise result than saying 'the approval is less than 24 hours old, therefore proceed.'

Continuous access evaluation shows one way change can travel

The OpenID Continuous Access Evaluation Profile (CAEP) formalizes event types that cooperating systems can use to communicate changing

security state so receivers can attenuate access for users, devices, sessions, and applications. In other words, access does not have to wait for a long-lived credential to expire naturally before a relevant change can matter. [12]

This is a useful implementation neighbor for Temporal Standing.

A standing system can consume CAEP-like signals as ΔN for security predicates. A credential change, session revocation, device-compliance change, or risk event may invalidate an otherwise fresh-looking execution context.

But an AI consequence can depend on non-security predicates too: a changed price, withdrawn communication preference, closed case, modified travel dates, updated vendor destination, superseding contract term, or changed human instruction.

The architectural lesson is broader than any one protocol: **important changes should have a path to invalidate cached authority before the consequence.**

Sometimes that path is event-driven. Sometimes it is a short expiry. Sometimes it is a read from an authoritative source at execution. Often it is a combination.

Freshness should therefore be designed predicate by predicate rather than by assigning one universal lifetime to "the approval."

Re-authorization should be precise

When standing is lost, systems often ask users to "approve again." That can create needless friction and approval fatigue.

A better design asks what actually changed.

If only the price moved from $4,800 to $4,950 while the approved ceiling is $5,000, standing may remain preserved.

If the price moved to $5,400, the system may need a new approval for the amount but not a new confirmation of traveler identity, shipping address, or unrelated scope.

If the recipient changed but the content did not, reauthorization should focus on recipient and disclosure consequence.

This is the difference between **fresh determination** and **ritual re-consent**. Temporal Standing should reduce unnecessary prompts by rechecking material predicates, not maximize the number of times a human clicks yes.

Chapter 16 - Building the Standing Record

A system cannot determine present standing if it does not preserve enough information about past authority and relevant change.

This creates a practical design problem: what should the system record?

If nobody can explain why it was allowed, we have a problem

"The system approved it" is not lineage. It is a shrug with a database. A standing record should make it possible to reconstruct the authority, evidence, changes, scope, determination, and exact consequence without conducting digital archaeology across six services and somebody's inbox.

Governance becomes credible when the reason for action survives the action.

The answer is not "everything." Unlimited logging creates privacy, cost, security, and usability problems. The answer is to preserve the minimum lineage necessary to understand and govern future consequences.

We can call this the **standing record**.

Original authority

The standing record begins with T_0.

For a consequential action, useful fields may include:

- authority source;
- actor or principal;
- action class;
- target;
- purpose;
- data scope;
- monetary or quantitative limits;
- validity period;
- conditions;
- delegability;
- revocation mechanism;
- policy or rule version;
- evidence references;
- creation time.

The details should follow the domain.

A communication system needs different fields from a payment system. A deployment system needs different fields from a healthcare workflow. The architecture should preserve the dimensions that determine whether authority continues.

Change signals

The system also needs to know which events can alter standing.

These might include:

consent changed;
role changed;
policy updated;
account status changed;
price changed;
recipient changed;
payload changed;
new evidence arrived;
identity assurance changed;
revocation issued;
expiry reached;
resource state changed;
human override issued;
workflow superseded.

Change signals can come from events, version numbers, timestamps, subscriptions, database triggers, authoritative queries, or other mechanisms.

The important design step is identifying them explicitly.

If a system does not know what can defeat standing, it cannot know when to reassess.

Evidence snapshot

At T_n, a standing determination should record enough of the current evidence state to explain the outcome.

This does not necessarily mean copying entire documents. References, versions, hashes, decision inputs, or signed claims may be enough.

The goal is reconstructability.

If someone asks later, "Why did the system believe standing was preserved?" the answer should be more specific than "the AI decided."

A useful record might show:

- prior authority ID;
- current identity state;
- current scope;
- relevant policy version;
- current consent or delegation version;
- detected ΔN events;
- action fingerprint;
- determination;
- decision time;
- validity boundary;
- decision-maker or service;

- reason codes.

Determination record

The outcome itself should be explicit.

PRESERVED for bounded action X.

DEFEATED because authority Y was revoked at time Z.

UNRESOLVED because required source Q could not establish current state.

These semantics are more useful than generic "success/failure" statuses.

They allow the workflow to respond correctly.

Preserved can proceed.

Defeated can stop, close, or request new authority depending on policy.

Unresolved can hold, refresh, escalate, or enter an authorized safe mode.

Permit lineage

If a system issues an executable permit, the permit should be traceable to the determination that produced it.

The permit might contain or reference:

action fingerprint,

principal,

scope,

expiry,

policy version,

evidence snapshot,

determination ID,

nonce or idempotency information,

and signature or integrity protection.

The exact cryptographic design is implementation-specific. The conceptual requirement is not.

A downstream executor should not have to trust an unbounded "approved = true" flag detached from its lineage.

Privacy and minimization

Standing records themselves can become sensitive.

They may reveal preferences, roles, health-related state, financial information, internal policy, or reasons for decisions.

Governance data therefore needs governance.

Retain what is necessary for enforcement, accountability, and legitimate operational needs. Protect it appropriately. Limit who can inspect it. Define retention. Separate sensitive evidence from lower-sensitivity decision metadata where possible.

Temporal Standing is not an excuse for indiscriminate surveillance.

Supersession

One of the most important properties of the standing record is the ability to represent supersession.

A new instruction can supersede an old one.

A correction can supersede an error.

A new policy can supersede an old policy.

A revocation can defeat a grant.

A renewed authorization can restore standing under new conditions.

The system should preserve lineage without leaving every historical state equally active.

This is particularly important for AI memory. Retrieval systems often optimize for relevance, not authority precedence. An old preference can be highly semantically relevant and still be wrong to apply.

Standing-aware retrieval should know which records control the present.

The minimum viable standing record

For builders starting small, the standing record can be reduced to five questions:

1. What prior authority is this action relying on?
2. What is the exact consequence being proposed?
3. What relevant changes can alter the authority?
4. What current evidence establishes preserved, defeated, or unresolved standing?
5. What enforcement mechanism binds the determination to execution?

If a system can answer those five questions reliably, it has moved far beyond governance by prompt.

The standing record gives time a structure.

It allows history to remain history, change to become legible, and present authority to be determined rather than assumed.

A standing record is a reproducibility artifact

A strong standing record should let an independent reviewer reconstruct the basis of an execution without replaying a model's hidden reasoning.

That is an important design goal.

The record does not need every token the model considered. It needs the governing facts and relationships that justified the consequence.

For example:

- proposed consequence identifier;
- canonical action and material payload fields;
- acting principal and represented principal;
- purpose and scope;
- required standing predicates;
- authoritative source identifiers and versions;

- relevant ΔN events considered;
- freshness state;
- policy version;
- determination outcome;
- determination timestamp;
- permit identity and expiry;
- executor result;
- reconciliation state if the provider outcome was ambiguous.

This is closer to a transaction receipt than a chain-of-thought transcript.

It also enables better testing. Teams can take historical standing records, mutate one predicate, and verify that expected outcomes change. They can test whether revoked authority actually reaches execution. They can measure how often unresolved outcomes come from source outages versus genuine conflicts. They can identify predicates whose freshness windows are causing unnecessary holds.

The record becomes an operational instrument for governance quality.

Data minimization still applies to governance

More evidence is not automatically better governance.

A standing system can become a privacy problem if it centralizes every possible fact "just in case." The better approach is to preserve references, proofs, versions, or minimized snapshots sufficient to justify the determination under the applicable policy.

This is another reason to define predicate sets. When the consequence requires only account status, role, amount ceiling, and current recipient preference, the governance layer should not ingest unrelated sensitive data merely because the enterprise possesses it.

Good standing architecture is selective.

It knows what it must know—and what it does not need to know—to govern the consequence.

Chapter 17 - Designing for "I Don't Know Yet"

AI products are designed to be helpful.

Helpful systems answer questions. They complete tasks. They reduce friction. They avoid unnecessary refusal. They recover from failure. They keep going.

`UNKNOWN` is not a bug

Engineers are trained to eliminate uncertainty. Product managers are trained to eliminate friction. Agents are trained to finish the task. This creates a powerful cultural coalition against the sentence, "We do not know yet."

Unfortunately, the world continues to produce missing evidence, stale replicas, contradictory records, and unavailable services. A mature system needs somewhere legitimate to put uncertainty besides `ALLOW`.

Those are good product instincts until the system reaches a boundary where incomplete knowledge concerns authority.

At that boundary, “I don't know yet” can be the most responsible answer available.

Product pressure against uncertainty

Many software metrics reward closure.

Completion rate.

Resolution time.

Automation percentage.

Successful tool calls.

Queue throughput.

Conversion.

User satisfaction.

These metrics can unintentionally punish unresolved standing. A held action looks like failure. A human escalation looks expensive. A blocked send reduces throughput. A delayed transaction creates friction.

Teams under pressure may respond by loosening checks until the system almost always finds a path forward.

That is precisely how governance becomes theater.

If the architecture is designed so that every uncertainty eventually becomes permission, unresolved standing is not a real state.

Unresolved does not mean helpless

A system can do useful work while standing remains unresolved.

It can gather additional evidence.

It can request clarification.

It can prepare a draft without sending.
It can calculate alternatives.
It can explain what is missing.
It can notify an authorized reviewer.
It can reconcile transport state.
It can wait for an authoritative source to recover.
It can narrow the proposed action to one with preserved standing.
It can enter a predefined safe mode.
The key is to separate actions whose standing is preserved from the consequence that remains unresolved.
For example, an agent may be unable to establish standing to issue a refund but still be authorized to draft the refund request and route it to an authorized employee.
The workflow continues without pretending authority exists.

Good uncertainty messages

When a human is involved, the system should explain unresolved standing in operational terms.
Not:
"Something went wrong."
But:
"Current authorization cannot be established because the approval expired after the amount changed."
Or:
"The current communication preference source is unavailable. The message has not been sent."
Or:
"The new recipient was not covered by the original approval. Review is required before disclosure."
This makes uncertainty actionable.

Timeouts are governance choices

Engineers often add timeouts to unresolved states.
"If no response arrives in ten minutes, proceed."
That can be appropriate only if there is legitimate authority for the fallback.
A timeout is not neutral. It encodes who wins when evidence is missing.
Does silence preserve the old state?
Does silence block action?
Does silence trigger emergency authority?
Does silence escalate?
Those are governance decisions.
They should be explicit.

Human escalation

Human review is a common resolution path, but the reviewer needs context.

A useful escalation should show:

what action is proposed;

what T_0 authority exists;

what ΔN occurred;

why the system cannot determine standing;

what evidence is available;

what authority the reviewer has;

what scope the decision will cover;

and whether the action will expire if not executed soon.

Without this information, the human may simply rubber-stamp the system's desired outcome.

Ambiguity budgets

Not every ambiguity deserves the same response.

Low-consequence actions can tolerate broader defaults. High-consequence actions should demand stronger standing.

Organizations can think in terms of an ambiguity budget.

How much uncertainty is acceptable for this class of consequence?

What evidence must be authoritative?

Which inputs can be probabilistic?

Which unknowns force a hold?

Which unknowns can be safely ignored because they are irrelevant to standing?

This avoids both extremes: reckless action and paralyzing overcontrol.

Uncertainty should be preserved across layers

One dangerous pattern occurs when an upstream system expresses uncertainty but a downstream system collapses it.

A governance service returns “unresolved.”

An orchestration layer maps any non-error response to success.

A worker interprets success as send.

The system has technically preserved uncertainty in one component while causally erasing it in another.

Outcome semantics must survive the full path to enforcement.

This is why determination and execution should be tested together.

Maturity is not certainty

We often imagine mature AI as AI that knows more.

In consequential systems, maturity also means knowing which unknowns matter enough to stop.

A trustworthy agent does not need omniscience.

It needs disciplined uncertainty.

It should be capable of saying:

"I can continue preparing."

"I can gather more evidence."

"I can ask the authorized person."

"I can explain what changed."

"But I cannot cause this consequence until current standing is established."

That is not weakness.

It is what turns uncertainty from a hidden risk into a governed state.

Uncertainty needs service-level objectives

If every dependency failure produces an indefinite hold, governance can become indistinguishable from outage.

Mature systems therefore need operational objectives for unresolved states.

How long may the system wait for an authoritative source?

Which unresolved conditions can be retried automatically?

When should a case escalate to a human?

Which low-risk actions may use a bounded stale cache under explicit degraded-mode policy?

Which high-risk actions must remain blocked no matter how inconvenient the dependency failure becomes?

These are not merely reliability questions. They are governance choices because each fallback defines what authority the institution is willing to accept when ideal evidence is unavailable.

A dangerous fallback is implicit permissiveness: "if the policy service is down, proceed." A dangerous opposite is indiscriminate paralysis: "if any optional signal is missing, stop everything."

The disciplined approach is to classify required and optional predicates, define degraded modes intentionally, and measure them.

Unresolved should be observable as an operational state with cause codes, age, owner, and remediation path.

Human review should receive the uncertainty, not hide it

Escalation systems often convert uncertainty into a green or red button for the reviewer.

That can erase the very information the human needs.

If standing is unresolved because the consent service is unavailable, say that. If two policy versions conflict, show the conflict. If the payload changed after approval, identify the fields. If identity confidence is insufficient, name the missing proof.

The reviewer should not be asked to approve "the action" in the abstract. The reviewer should be asked whether they possess authority to resolve the

specific uncertainty, and the resulting decision should itself become a bounded governing fact.

Human review is strongest when it closes a named gap.

It is weakest when it functions as a ceremonial override for a system that cannot explain what it does not know.

Chapter 18 - Governance Without Theater

AI governance is easy to describe and difficult to enforce.

Organizations can publish principles, create committees, write model cards, maintain risk registers, require approvals, and add safety language to prompts. These practices can be valuable.

Governance without theater

If the dashboard says **DENY**, the audit log says **DENY**, the governance engine says **DENY**, and the email still goes out, congratulations: you have built an unusually well-observed failure.

The question is not whether the organization can prove that it disapproved. The question is whether disapproval could actually stop the consequence.

They become theater when they create the appearance of control without changing whether a prohibited consequence can occur.

Temporal Standing offers a simple test for governance claims:

When current standing is missing, can the system actually prevent the action?

If not, the governance may be advisory rather than causal.

Policy language versus execution control

Consider four levels of control.

At the first level, a policy document says the system should not perform an action without current consent.

At the second, the model prompt repeats the rule.

At the third, the application checks consent and warns if it appears missing.

At the fourth, the execution path cannot send unless a current standing determination produces a valid bounded permit.

These are not equivalent.

Each level can contribute value, but only the last makes the rule a property of the causal system rather than a behavioral expectation.

Dashboards can lie without containing false data

A dashboard may accurately display green indicators and still mislead.

“Governance check: passed.”

What was checked?

When?

Against which action?

Did the payload change afterward?

Was the check advisory or enforced?

Could another path execute without it?

Did a downstream tool validate the permit?

Did a retry inherit the old result?

The label "passed" can compress away the very temporal details that determine standing.

Good governance interfaces should expose the lineage that matters without overwhelming users.

Post-execution monitoring

Monitoring is important.

It can detect misuse, measure patterns, identify failures, and support accountability.

But monitoring after execution is not a substitute for a pre-action boundary when the system could have prevented the consequence.

There is a fundamental difference between:

"We detected that the unauthorized message was sent."

and

"The message could not be sent because current standing was not established."

Both are useful capabilities. They solve different problems.

Human approval as theater

A human-in-the-loop can become theatrical too.

If the person lacks meaningful information, lacks time, lacks authority, or is trained to click approve, the presence of a human can merely decorate automation.

Temporal Standing improves human review by making the relevant transition explicit:

Here is what was authorized at T_0.

Here is what changed.

Here is the exact consequence proposed now.

Here is why standing is unresolved or defeated.

Here is the decision you are authorized to make.

This makes human review substantive rather than ceremonial.

Governance bypasses

Real systems contain emergency paths, administrator tools, migrations, scripts, manual overrides, and legacy integrations.

These paths often bypass ordinary governance.

Some bypasses are legitimate. During an outage, an authorized operator may need an emergency mechanism. During migration, a privileged process may need direct access.

The problem is not exceptions. The problem is invisible exceptions.

A mature governance architecture documents exceptional authority, scopes it narrowly, records its use, and prevents it from becoming the default path.

If every failed standing check can be bypassed with a generic administrator token, the effective governance model is the administrator token.

Testing governance causally

Governance should be tested like other critical system behavior.

A useful test suite does not merely assert that a policy function returns `DENY`.

It verifies that transport does not occur.

It tests that an unresolved determination cannot fall through to execution.

It tests that payload mutation invalidates or requires a new permit.

It tests that revocation defeats queued actions.

It tests that retries do not resurrect stale authority.

It tests that a downstream executor rejects missing, expired, or mismatched permits.

It tests that an authorized action still succeeds so governance does not simply break the product.

This is the equivalent of integration testing for legitimacy.

Claims should match enforcement

Organizations should be careful about saying "our AI is governed" when what they mean is "our AI receives governance guidance."

The distinction matters to customers, regulators, operators, and system owners.

A more honest vocabulary differentiates:

advisory policy;

model-level instruction;

pre-action evaluation;

causal enforcement;

post-action audit;

and human escalation.

Each has value.

None should borrow the credibility of another.

The practical standard

Governance becomes real when a system can demonstrate a chain like this:

A consequence was proposed.

The system identified the governing authority.

Relevant change was evaluated.

Current standing was determined.
The decision was bound to the exact action.
The executor required the decision.
The consequence occurred only when standing was preserved.
And the lineage can be inspected afterward.
That is not governance theater.
It is governance with causal teeth.

The reference-monitor question

Security architecture has long recognized a basic requirement: a rule matters only if the path to the protected effect cannot simply bypass the mechanism enforcing it.

Temporal Standing inherits that discipline.

If an application asks a governance service for a decision and then calls the external provider regardless of the answer, the system has governance telemetry, not governance control.

If one execution path requires a permit but an administrator script, background worker, legacy integration, or retry queue can perform the same consequence without it, the effective enforcement model includes the bypass.

This is why a demonstration should test the system from consequence backward.

Do not merely ask, 'Did the policy engine return DENY?'

Ask, 'Was the external effect causally impossible through the governed path after DENY?'

Then ask, 'Were there other paths that could still cause it?'

That shift makes governance claims falsifiable.

Decision and enforcement are established ideas - binding is the hard part

Modern policy systems already distinguish decision points from enforcement points. Open Policy Agent, for example, explicitly describes the relationship between a Policy Decision Point and application Policy Enforcement Points. Cedar similarly expects the application to ask whether a concrete principal-action-resource-context request is authorized before performing the operation.

Temporal Standing should not claim invention of that separation.

Its emphasis is on **binding** the determination to the exact consequence and carrying that binding to the executor.

Suppose a governance service approves an email to `alice@example.com` with attachment A. After approval, the model changes the recipient to Bob and swaps in attachment B. If the executor accepts the original permit anyway, the system separated decision from enforcement but failed to bind them tightly enough.

The same problem appears in retries, transformed payloads, downstream queues, and multi-agent handoffs.

A bounded permit is one architectural answer. The permit can identify material fields, authority versions, expiry, executor, and idempotency identity. Other architectures can achieve the same property differently.

The important claim is architectural rather than cryptographic:

The thing that executes must be materially the thing that was determined.

A green dashboard is not evidence of a governed consequence

Governance interfaces should therefore expose lineage rather than reassurance.

A useful record can answer:

What consequence was proposed?

Which governing grounds were used?

Which relevant changes were considered?

Which sources were current, stale, conflicting, or unavailable?

What standing outcome was produced?

What exact consequence did the permit bind?

Which executor verified it?

What actually happened?

This does more than support audit.

It makes system behavior debuggable.

When a legitimate action is blocked, operators can see whether the problem was defeated standing, missing evidence, stale policy, mismatched payload, expired permit, or transport failure. When an illegitimate action occurs, they can identify whether determination was wrong, evidence was wrong, binding failed, enforcement was bypassed, or the consequence happened outside the governed surface.

Governance becomes engineering when failure modes have names.

Complete mediation is older than AI

One of the most durable principles in computer security is complete mediation: every access to an object should be checked for authority rather than assuming that a previous check remains sufficient forever. Saltzer and Schroeder articulated this family of protection principles in their classic 1975 paper on computer-system protection. [13]

Temporal Standing should acknowledge that lineage directly.

The framework is not claiming that AI governance invented the idea of checking authority at the point of use. It extends the intuition into a world where the "object" being governed may be an external consequence assembled from multiple heterogeneous grounds rather than a simple resource access.

This lineage also supplies a useful test for governance theater.

If a consequential path can bypass the check, mediation is incomplete.

If a worker can resend without a new standing decision, mediation is incomplete.

If a human-approved payload can be silently substituted after approval, mediation is incomplete.

If one connector enforces governance while an older connector can perform the same effect without it, mediation is incomplete.

The existence of a policy service is therefore not enough. The question is whether the execution topology forces the consequence through an authoritative gate.

Policy decision points and enforcement points are not the same thing

Open Policy Agent describes itself as a policy decision point (PDP) used by applications that act as policy enforcement points (PEPs). Cedar similarly separates authorization logic from application business logic: the application asks whether a request is authorized, then decides whether to perform the operation based on the result. [14][10]

This separation is valuable because it keeps policy reasoning outside the code that performs the effect.

But it also creates a seam.

A correct deny from a PDP does nothing if the PEP ignores it.

An allow for one request does not automatically cover a mutated request.

A cached allow may be wrong after a material state change.

Temporal Standing is especially interested in this seam. Determination and enforcement should be separate enough to preserve clear authority boundaries, but tightly bound enough that the executor cannot create a materially different consequence under an old result.

That is why the book repeatedly distinguishes **governance truth** from **causal control**.

A dashboard can truthfully display "DENIED" while an email was nevertheless sent through a bypass path. The decision record is correct. The governed system is not.

Governance succeeds only when the consequence obeys the determination.

Retry safety is part of governance

Distributed systems retry because networks fail ambiguously. A client can time out without knowing whether the provider accepted the request.

AWS's Builders' Library recommends idempotent APIs so a repeated request can be recognized as the same intended operation rather than creating duplicate side effects. [15]

Temporal Standing adds another dimension: a retry should not accidentally become a new consequence under changed authority.

If the provider accepted the first payment but the client timed out, blindly issuing a fresh payment with a new idempotency key is not merely a reliability bug. It creates an unjustified second consequence.

If a communication send is ambiguous, the right next action may be reconciliation, not regeneration and resend.

If the payload changed after the first attempt, reusing the old permit may be wrong even if the idempotency mechanism is correct.

This yields a simple rule:

Retry the uncertainty about execution before retrying the consequence itself.

Idempotency protects against duplication. Standing protects against executing under superseded grounds. Mature systems need both.

PART V

THE FUTURE OF TRUSTWORTHY AI

Chapter 19 - From Intelligent Systems to Accountable Systems

The history of computing rewards capability.

Faster processors. Larger storage. Better networks. More accurate models. More capable agents.

Intelligence is the easy part to demo

A model can summarize a contract, plan a trip, write code, diagnose an error, and make a boardroom briefly believe the future has arrived. Accountability is less theatrical. It asks who authorized the action, which conditions still held, what changed, and whether execution could have been stopped.

The second list is not as flashy. It is what makes the first list deployable with trust.

Capability matters because useful systems must be able to do things.

But a system that can do more can also cause more.

As AI moves from answering into acting, the standard for trust must expand from intelligence to accountability.

An intelligent system can identify the best next action.

An accountable system can explain why it had present standing to cause that action, can be prevented from acting when standing is missing, and can preserve the lineage afterward.

This is a different engineering objective.

Capability without current authority

Imagine the ideal personal agent in purely capability terms.

It knows your preferences. It remembers your history. It can negotiate with services, manage your calendar, book travel, move information between apps, buy things, communicate with people, and anticipate what you need.

Now add one fact: its memory of your authority never expires or updates reliably.

The better the agent becomes at carrying out old intent, the more dangerous stale standing becomes.

A weak system may fail to act.

A powerful system may successfully perform the wrong action.

This is why trustworthy AI cannot be measured only by task completion.

We need to know whether the system completes tasks under legitimate current authority.

Accountability is architectural

Accountability is often discussed as something that happens after a failure.

Who is responsible?

Who approved the model?

Who deployed the system?

Who should compensate the person harmed?

Those questions matter, but accountable architecture begins earlier.

Can the system show what authority it relied on?

Can it show what changed?

Can it show the current standing determination?

Can it show the scope of the permit?

Can it show whether the executor enforced the permit?

Can it show whether a human override occurred and who possessed authority to issue it?

The better the lineage, the less accountability depends on reconstructing intent from scattered logs after the fact.

The system should be able to answer "why now?"

Explainability efforts often ask why a model produced an output.

Temporal Standing asks a related but different question:

Why was this consequence allowed **now**?

A strong answer might be:

The user authorized this recurring payment under mandate M.

The mandate remains active and unrevoked.

The amount is within the authorized ceiling.

The recipient matches the verified beneficiary.

No relevant policy or account-state change has defeated standing.

The determination was made at the consequence boundary and bound to transaction X.

That explanation is not a narrative generated after the fact. It is a description of the actual governing chain.

Trust through constraint

People often think trust in AI will come from making models so reliable that we are comfortable granting them broad discretion.

There is another route.

We can make systems trustworthy by constraining what intelligence is allowed to cause.

We do this already in other domains.

A pilot can be highly skilled and still operate under airspace rules.

A surgeon can be highly skilled and still require consent and institutional authority.

A financial employee can be highly trusted and still operate under transaction limits and separation of duties.

Competence and constraint are not opposites.

Constraints make competence safely deployable.

The same is true for AI.

Bounded autonomy

Temporal Standing points toward a model of bounded autonomy.

The agent is free to reason broadly inside a space of possible solutions.

It can plan, compare, optimize, propose, and adapt.

But external consequences require current standing.

This gives us a useful separation:

broad cognition, bounded execution.

The system does not need to be intellectually crippled to be governable.

It needs an execution architecture that refuses to treat intelligence as authority.

This is important because some safety approaches attempt to make the model itself less capable of considering dangerous or unauthorized actions. There can be valid reasons for model-level restrictions, especially where harmful content or misuse is concerned.

But for legitimate enterprise and agentic systems, we should not confuse governance with making the model unable to think about an action.

A model may need to reason about a prohibited action in order to explain why it is prohibited, detect an attack, compare alternatives, or prepare a compliant path.

The critical question is what the system can cause.

Institutions will need executable governance

As AI becomes embedded in institutions, governance will move from policy documents into executable infrastructure.

Organizations will need machine-readable authority, revocation, scope, current evidence, decision semantics, escalation paths, and enforcement boundaries.

This does not mean every policy can or should be reduced to code. Human judgment will remain important. Some ambiguity is irreducible. Some values require interpretation.

But even human judgment has to enter the system through a defined authority path if it is going to govern machine execution.

"Ask a human" is not architecture until the system knows which human, for what decision, under what authority, with what evidence, and how the decision becomes binding.

The accountability threshold

The more consequential and autonomous an AI system becomes, the stronger the case that it should meet an accountability threshold:

It can identify the authority supporting its actions.

It can detect or receive relevant change.

It can determine current standing for bounded consequences.

It can preserve unresolved standing rather than forcing certainty.

It can enforce the decision before external effect.

It can preserve lineage for review and recourse.

This is not the entire definition of trustworthy AI.

It is one necessary dimension of trust for AI that acts.

The transition ahead is not simply from less intelligent systems to more intelligent systems.

It is from systems that can do things to systems that can justify, at the moment of consequence, why they are entitled to do them.

Accountability should produce an executable answer

NIST's AI Risk Management Framework emphasizes governance as a cross-cutting function and treats risk management as continuous across the AI lifecycle. That is the right scale for organizational accountability. [1]

But accountable operation eventually needs to become executable.

For an individual consequence, someone should be able to answer:

- What authority supported this action?
- What material state was current?
- What changes were considered?
- Which policy applied?
- What uncertainty remained?
- Who or what made the determination?
- Was the executed payload the one that was evaluated?
- Did the executor actually enforce the result?

These questions turn "responsible AI" from a property claim into a reconstructible event.

The point is not that every system needs a courtroom-grade dossier for every low-risk action. Evidence should be proportionate. But as consequence rises, accountability that cannot explain why an action had authority **at that time** becomes increasingly thin.

Temporal Standing gives that explanation a common grammar.

Chapter 20 - A Social Contract for AI That Acts

Technology becomes socially important when people can no longer avoid its consequences simply by choosing not to use it.

AI is moving in that direction.

Trust needs an exit door

A social contract that lets people opt in but cannot recognize withdrawal is not much of a contract. It is more like a hotel that welcomes check-in enthusiastically and has misplaced the concept of checkout.

Systems that act on our behalf need legible mechanisms for correction, revocation, recourse, and changed circumstances. Otherwise autonomy belongs mostly to the machine.

Agents will act for employers, service providers, financial institutions, platforms, governments, professionals, and individuals. People will encounter the consequences of AI-mediated decisions even when they did not personally choose the model, build the workflow, or understand the architecture.

That creates a need for public expectations about how acting systems should behave.

Temporal Standing suggests six principles that could form part of a social contract for AI that acts.

1. No consequential action from stale authority

A system should not treat the historical existence of authority as sufficient proof of current authority when relevant change can occur.

This principle does not require constant reapproval.

It requires continuity mechanisms that are honest about time: explicit duration, scope, freshness, revocation, change signals, and current determination.

The burden should rise with consequence.

2. Changed consent must be legible to the system

If a person can legitimately revoke or narrow consent, that change should be capable of reaching future actions.

A revocation mechanism that updates a profile page but not queued automation is incomplete.

The practical right to withdraw depends on propagation.

This principle also implies that organizations should avoid architectures in which downstream AI systems are permanently detached from the authoritative consent state.

3. Unresolved standing pauses the governed consequence

Systems should not convert missing authority into permission by default.

If a consequence requires current evidence and the evidence cannot be established, the system should hold, narrow, escalate, or enter another explicitly authorized state.

This is a social expectation because people should not bear the risk created when institutions choose convenience over uncertainty.

4. Scope must be explicit enough to enforce

Broad goals are not broad authority.

A person who asks an agent to help manage finances has not necessarily authorized every transaction the agent can imagine.

An employee asked to resolve a customer issue has not necessarily been delegated unlimited account power.

A system should preserve meaningful boundaries around actor, action, data, purpose, duration, conditions, and consequence.

Scope can be flexible without being undefined.

5. Recommendation and execution should remain distinguishable

People and institutions should be able to tell when AI is advising and when AI is causing.

The distinction affects responsibility, oversight, authority, and risk.

A system that generates a recommendation should not silently cross into execution merely because the next tool call is technically available.

This principle does not require a human click between every step. It requires an architecture in which the transition to consequence is governed explicitly.

6. People need recourse, correction, and withdrawal

A system that respects Temporal Standing must allow legitimate change to become authoritative.

People need mechanisms to correct records, revoke permissions where revocation is allowed, renew permissions, contest decisions, and reach an authorized reviewer when the system cannot resolve standing correctly.

Without recourse, fail-closed governance can harden errors.

Without withdrawal, memory can become coercion.

Without correction, historical mistakes can rule the present.

Rights and obligations

A social contract is not one-sided.

People and institutions also have obligations.

Users should not expect agents to infer unlimited authority from vague goals.

Organizations should define who is authorized to decide what.

Builders should identify consequence boundaries rather than hiding them inside orchestration.

Operators should preserve revocation channels.

Leaders should resist metrics that reward completion at the expense of legitimate standing.

Policy-makers should distinguish between model behavior and system execution when evaluating controls.

Auditors should ask whether controls were causal or merely advisory.

Not every action deserves the same ceremony

A social contract for AI should not make ordinary life unusable.

People do not want to approve every calendar read, every formatting change, or every low-risk personalization.

The solution is proportionality.

Low-consequence, durable, well-scoped permissions can remain efficient.

High-consequence, volatile, or ambiguous actions should require stronger standing.

The system should use the least friction necessary to preserve legitimacy.

Good governance is not maximum interruption.

It is minimum unjustified consequence.

Institutional accountability

If an AI system acts on behalf of an institution, the institution should not be able to hide behind the model.

"The AI did it" is not a meaningful authority source.

The system acts because the institution deployed it, connected it, granted it capability, defined its policy, and allowed the consequence path.

Temporal Standing makes institutional responsibility easier to see because it asks where authority came from and how it was enforced.

If no legitimate source can be identified, that is itself a governance failure.

A norm that can travel

The power of the Temporal Standing question is that it can travel across domains.

It does not require everyone to agree on one moral theory, one legal regime, or one model architecture.

Different institutions can have different rules about consent, delegation, evidence, and permissible action.

The shared expectation is procedural:

When an AI is about to cause a consequential effect, prior authority should not be treated as timeless by default.

Relevant change should matter.

Uncertainty should remain visible.

Scope should remain bounded.

And the system should be able to answer why the consequence had standing now.

That is a modest principle compared with solving all of AI ethics.

It is also a principle we can build.

A social contract should include the right to change

Much AI governance language focuses on the right to know, the right to explanation, the right to privacy, and the right to contest certain outcomes.

Acting systems add another practical concern: **the right for relevant changes to matter before the next consequence.**

People change their minds. Institutions change roles. Accounts change status. Contracts change terms. Policies are superseded. Emergencies begin and end. Delegations are granted and revoked.

A trustworthy system should not require people to erase their history in order to escape it.

The social value of Temporal Standing is precisely that it can preserve history while allowing present authority to change.

The system can remember that permission existed without treating it as eternal.

It can remember that a person once occupied a role without allowing the role's powers to follow them forever.

It can remember that a recommendation was once reasonable without insisting that the world remain frozen around it.

This is more than an engineering convenience. It is a design posture toward human agency.

A person is not only the sum of previously stored states.

An institution is not only the sum of yesterday's approvals.

An intelligent system should be capable of recognizing continuity without confusing continuity with permanence.

Chapter 21 - The Question Every AI System Must Ask

This book began with a dangerous assumption:
If it was true, it is still true.

The question that survives the jargon

AI governance has no shortage of vocabulary. We can create committees, taxonomies, maturity models, assurance layers, control families, and diagrams with enough arrows to qualify as modern art. Eventually, however, the system reaches an action.

At that moment the question becomes wonderfully impolite in its simplicity: **Does it still stand?**

If it was authorized, it is still authorized.
If the user wanted it, the user still wants it.
If the policy permitted it, the policy still permits it.
If the system prepared the action correctly, it can execute whenever the workflow reaches that step.

We now know why these assumptions fail.
AI acts in time.
Between instruction and consequence, reality can change.
People can revoke consent.
Roles can change.
Policies can change.
Facts can change.
Prices can change.
Recipients can change.
Scope can drift.
Evidence can become stale.
A system can lose access to the authoritative source it needs.
A retry can occur under conditions different from the original attempt.
A delegated agent can receive a task without receiving legitimate authority for every downstream consequence.
A model can remember perfectly and still be wrong to act.

The central discipline is therefore simple:
$T_0 \rightarrow \Delta N \rightarrow T_n$
What was the prior authoritative state?
What relevant change occurred?
What is the present standing of that authority for this exact consequence now?

The answer can be preserved, defeated, or unresolved.
Preserved standing supports execution within the bounded scope.
Defeated standing blocks execution under the old authority.

Unresolved standing holds the consequence until the missing governing information is resolved or another legitimate authority applies.

Everything else in this book follows from that structure.

Remember the separations

Several separations protect the framework from collapsing into vague safety language.

History is not present authority.

The fact that permission once existed remains historically true after permission changes.

Evidence is not authority.

Documents, database rows, model memories, summaries, confidence scores, and retrieved records can inform a determination without becoming authority themselves.

Capability is not authority.

A tool being available does not mean the agent may use it for every possible consequence.

Proposal is not execution.

A model can reason broadly while execution remains bounded.

Determination is not enforcement.

A correct deny that does not stop transport is not operational governance.

Audit is not prevention.

Knowing why a bad action happened after the fact is valuable, but it does not substitute for preventing an unjustified consequence when prevention was possible.

Human involvement is not automatically current standing.

Human approvals can be vague, stale, out of scope, or based on facts that later change.

A permit is not permanent permission.

It applies to the bounded consequence that was evaluated.

These distinctions make the architecture legible.

The smallest useful implementation

An organization does not need to rebuild every system before applying Temporal Standing.

Start with one consequential workflow.

Identify the external effect.

Identify the last point at which it can be stopped.

Identify the authority the workflow relies on.

Define the scope of that authority.

Identify the changes that can defeat or complicate it.

Define the authoritative sources of present state.

Represent preserved, defeated, and unresolved outcomes explicitly.

Require preserved standing at the consequence boundary.

Record the lineage.
Then test the failure paths.
Revoke authority while work is queued.
Change the amount.
Change the recipient.
Let the evidence source become unavailable.
Mutate the payload after approval.
Trigger a retry after expiry.
Attempt to bypass the gate.
Verify not only that the governance layer returns the right answer, but that the consequence actually does or does not occur accordingly.
This is governance as engineering.

The broader implication

Temporal Standing is not limited to artificial intelligence.
The pattern appears anywhere authority persists through changing time.
But AI makes the problem urgent because AI compresses cognition, coordination, and action into increasingly autonomous systems.
An agent can interpret an old instruction, gather old and new evidence, create a plan, call a tool, recover from failure, and continue without the person who originally supplied the goal ever seeing the intermediate decisions.
That is extraordinary capability.
It demands equally serious governance.
The answer is not to freeze intelligence.
It is not to demand a human click for every micro-action.
It is not to pretend prompts can substitute for authority.
It is to build systems in which intelligence can move quickly inside boundaries that remain connected to the present.

The human meaning

At its deepest level, Temporal Standing protects the legitimacy of change.
A person can say yes and later say no.
An institution can grant authority and later revoke it.
A fact can be corrected.
A condition can change.
A plan can be superseded.
A relationship can end.
A policy can evolve.
A system that recognizes those changes is not forgetting history.
It is understanding history properly.
The past tells us how we arrived here.
It does not, by itself, decide what may happen next.

The question

As AI moves deeper into the systems that communicate, transact, decide, and act for us, builders will ask increasingly sophisticated questions.

How capable is the model?
How accurate is it?
How fast is it?
How much can it automate?
How secure is the infrastructure?
How well does it follow instructions?
How much does it cost?
All of those questions matter.
Before consequence, one more question must become routine:

> ***This was authorized then. What has changed, and does it still stand now?***

That question is Temporal Standing.

And for AI that acts, it may be one of the most important questions we learn to build into the machine.

A minimum test suite for systems that act

If Temporal Standing is to be more than language, builders should be able to test it.

A minimum test suite can start with a small set of adversarial transitions:

Revocation after preparation. Prepare a valid consequence at T_0, revoke a required authority before execution, and verify that the consequence does not occur.

Payload drift after approval. Approve one exact action, mutate a material field, and verify that the old permit cannot execute the changed consequence.

Authoritative source unavailable. Remove a required source and verify that the system produces unresolved rather than silently allowing.

Conflicting evidence. Present two authoritative-looking records with no encoded resolution rule and verify that the model does not invent one.

Stale replica. Provide an older but internally valid state after a newer defeating event and verify that causal ordering or version checks prevent resurrection of the old authority.

Ambiguous provider result. Cause a timeout after the provider may have accepted the action and verify that the system reconciles before retrying the consequence.

Delegation attenuation. Give a parent agent narrow authority, delegate to a sub-agent, and verify that the child cannot expand the scope.

Durable authority preserved. Provide a case where nothing material changed and verify that the system does not create unnecessary friction by demanding ritual reapproval.

A framework that cannot survive these tests is not yet governing time. It is describing it.

The question at the end of this book is therefore also a test harness:

This was authorized then. What has changed, and does it still stand now for this exact consequence?

APPENDICES

Appendix A - Temporal Standing and Its Intellectual Neighbors

Temporal Standing does not begin from the premise that computer science forgot time.

It did not.

For decades, security researchers have studied expiring authorization, temporal access control, changing attributes, revocation, ongoing authorization, and policy enforcement. Modern identity systems propagate revocation and risk events. Policy engines evaluate the present context of a request. Privacy law recognizes that consent can be withdrawn without rewriting history.

That landscape is not a problem for Temporal Standing. It is the landscape in which the framework must be located honestly.

Temporal authorization

Temporal authorization models attach validity periods or temporal dependencies to permission. Their question is often whether a subject is authorized to exercise a mode of access on an object at a given time.

Temporal Standing asks a related but broader question: whether the grounds supporting a proposed consequence still stand now. Time may matter, but the decisive change might instead be a revoked consent, changed recipient, new factual condition, superseding policy, altered purpose, changed account state, conflicting authoritative records, or loss of evidence freshness.

A Temporal Standing implementation may use temporal access control. It is not reducible to it.

Usage Control (UCON)

UCON deserves special attention because it directly challenged the old picture of authorization as a one-time gate. UCON brought together authorizations, obligations, conditions, ongoing controls, and mutable attributes. [13] In an ongoing use, a change can trigger reevaluation and revocation.

That idea is close enough that this book should say so plainly.

Temporal Standing differs less by denying UCON's insight than by relocating the governing question. The unit is not merely continuing use of a resource. The unit is a bounded consequence proposed by an acting system. Its grounds may include access rights, but may also include consent, delegated authority, purpose, factual state, identity, policy, user preference, eligibility, transaction terms, and evidence quality.

Where UCON asks whether usage remains allowed under its policy model, Temporal Standing asks whether the grounds for this exact consequence remain sufficiently authoritative now.

Attribute-based access control

ABAC authorizes operations by evaluating attributes of the subject, object, action, and environment against policy. [8] This is an important implementation pattern for current-state decisions.

Temporal Standing adds two disciplines that are easy to lose in a snapshot-oriented request:

First, it makes the transition from prior grounds through intervening change explicit. That lineage matters when the question is not merely 'what is the attribute now?' but 'did the authority that supported the queued action survive the change?'

Second, it makes the exact external consequence the object of the determination. An ABAC decision may be one input into standing, not necessarily the whole determination.

Zero Trust and continuous access evaluation

Zero Trust rejects implicit trust and encourages explicit authorization focused on resources. Continuous Access Evaluation Profile events can signal security-state changes so cooperating receivers can attenuate access without waiting for ordinary credential expiry. [12]

Temporal Standing shares the refusal to treat past trust as permanent. Its scope is broader than security access, however. A system might pass every Zero Trust check and still lack standing to send a marketing message after consent withdrawal, book a different flight than the user approved, pay a vendor whose bank destination changed, or disclose information for a purpose outside the original authorization.

Revocation

Revocation is one species of ΔN.

It is not the whole framework.

Some changes defeat standing explicitly: 'I revoke permission.' Others do so because a condition changed. Still others do not defeat standing but create an unresolved state. A missing authoritative source, for example, is not the same thing as an explicit revocation.

That distinction matters because remediation differs.

Policy engines

Modern systems already separate policy decision points from enforcement points. That architecture is compatible with Temporal Standing and often desirable.

The framework's additional demand is that the enforcement path be causally bound to the determination for the exact consequence. If a policy engine denied one payload but a changed payload can still be sent under the old result, the separation exists but the binding is weak.

Causal consistency

Distributed authorization systems such as Zanzibar show why ordering matters. [11] If Alice revokes Bob's access and Bob immediately attempts access, the system should not casually reorder those events in a way that resurrects permission.

Temporal Standing depends on the same intuition but applies it to heterogeneous authority transitions. ΔN must not disappear merely because a downstream component observed an older state.

Consent withdrawal

Privacy law provides a useful conceptual example. Under the GDPR, withdrawal of consent does not make prior consent retroactively nonexistent. [4] Processing that was lawful before withdrawal is not rewritten as unlawful merely because future processing may no longer rely on that consent.

That is almost the ideal illustration of the distinction between historical truth and present standing.

At T_0, consent existed.

At ΔN, it was withdrawn.

At T_n, the historical record remains true: consent existed then. But that record alone cannot support a new consent-dependent consequence now.

Temporal Standing is designed to preserve both truths at once.

The boundary claim

The framework should therefore be understood as a synthesis and extension around a specific problem created by acting AI:

How should a system determine whether the heterogeneous grounds that once supported an action still support this exact external consequence now, and how should that determination be bound to execution?

That is narrower than claiming invention of dynamic authorization.

It is also more useful.

Appendix B – A Reference Engineering Pattern

Temporal Standing is a governance framework, not a single software architecture. But a reference pattern helps convert the idea into engineering requirements.

1. Proposal

A model, deterministic service, human operator, or workflow proposes a concrete consequence.

The proposal should be sufficiently specific to evaluate:

- actor or delegated principal;
- action;
- target or recipient;
- payload or material parameters;
- purpose;
- tool or transport;
- relevant data;
- expected external effect.

A vague goal such as 'handle the invoice' is not yet a consequence permit request.

2. Standing envelope

The system constructs a standing envelope containing the governing facts and identifiers needed for the decision. Depending on the domain, this may include:

- original authority or instruction;
- scope and duration;
- current identity and delegation state;
- revocation state;
- policy version;
- current subject/resource attributes;
- relevant consent state;
- factual predicates;
- source provenance;
- freshness requirements;
- known ΔN events;
- unresolved conflicts.

The envelope is not an LLM summary. It is a structured set of evidence and authoritative references.

3. Determination

A governance component evaluates whether present standing for the proposed consequence is:

PRESERVED - governing grounds still support it within scope;

DEFEATED - governing grounds affirmatively no longer support it;

UNRESOLVED - sufficient authoritative grounds are unavailable or indeterminate.

The determination should explain which governing predicates controlled the result.

4. Permit binding

For preserved standing, the system creates a bounded permit or equivalent capability tied to material fields of the consequence.

A practical binding may include hashes or signed claims over:

- action type;
- target;
- material payload;
- principal;
- tool/executor;
- purpose;
- authority version;
- policy version;
- issued-at time;
- expiry/freshness limit;
- idempotency identity;
- standing determination identifier.

The exact mechanism varies. The principle does not: material mutation requires a new determination.

5. Enforcement point

The component capable of causing the consequence must require a valid permit. A separate pre-check that can be bypassed is advisory governance.

The enforcement point verifies:

- permit authenticity;
- freshness;
- intended executor;
- payload/target binding;
- one-time or idempotent semantics where required;

- revocation/version rules where applicable.

6. Execute once

The executor causes the external effect only after verification.

Retries require special care. A retry may be transport-equivalent or consequence-new.

For example, retrying a network call after a confirmed non-delivery may remain within the same permit if the permit explicitly covers that retry behavior and standing remains fresh. Retrying after an ambiguous provider timeout may risk duplicate consequence and therefore require reconciliation rather than blind replay.

7. Record lineage

The system records:

- proposal identifier;
- evidence snapshot or authoritative references;
- ΔN considered;
- determination;
- permit identity;
- executor verification;
- execution result;
- provider/external identifier when available.

This creates inspectable lineage without pretending the log itself created authority.

8. Failure semantics

A mature implementation distinguishes at least:

- **DEFEATED:** do not execute; new or changed authority is required.
- **UNRESOLVED:** do not execute; refresh, reconcile, narrow, or escalate.
- **EVALUATION_UNAVAILABLE:** operational subtype of unresolved unless an explicitly authorized safe fallback exists.
- **PERMIT_EXPIRED:** obtain a new determination.
- **PERMIT_MISMATCH:** reject and re-evaluate the changed consequence.
- **TRANSPORT_FAILED:** consequence may still have standing; transport failure does not imply governance denial.
- **TRANSPORT_AMBIGUOUS:** reconcile before retry where duplicate consequence matters.

9. Atomicity and race windows

For low-consequence actions, short-lived permits and version checks may be enough.

For high-consequence actions, the architecture may require stronger coupling: transactional version checks, compare-and-swap semantics, executor-side policy evaluation, or other mechanisms that reduce the interval in which the governing state can change between determination and effect.

Temporal Standing does not eliminate distributed-systems races. It makes the required relationship explicit so the implementation can choose a proportional control.

10. Pseudocode

```text
proposal = build_consequence(request)
envelope = gather_governing_state(proposal)
standing = determine_standing(envelope, proposal)
if standing == DEFEATED:
  stop("new authority required")
if standing == UNRESOLVED:
  stop("resolve governing state")
permit = bind_permit(standing, proposal)
executor.verify(permit, proposal)
executor.execute(proposal)
record_lineage(proposal, standing, permit, result)
```

The simple pseudocode hides hard work in `gather_governing_state` and `determine_standing`. That is deliberate. Temporal Standing does not claim that identifying authority is easy. It claims that an acting system should not skip the question simply because cognition is fluent.

Appendix C - Case Studies: What Changed Between Approval and Effect?

The cases in this appendix are designed to make Temporal Standing testable. Some draw directly on documented mechanisms in standards and production systems; others are composite scenarios constructed from ordinary workflow patterns. Composite cases are labeled as such. They are not claims about a particular company or incident.

Case 1 - Consent Withdrawal After Campaign Preparation

Type: Composite scenario grounded in GDPR consent-withdrawal structure.

A customer opts into a communications program on Monday. The organization records the consent event, creates a campaign membership, and schedules a sequence of messages. On Tuesday the customer withdraws consent through the preference center. The consent service updates immediately. A batch worker that built its audience on Monday still holds the recipient in a queue. On Wednesday it reaches the next message.

At T_0, the communication had a valid historical ground: consent existed and the proposed purpose matched the program for which it was obtained.

ΔN is explicit withdrawal.

At T_n, the historical record is not false. It is insufficient for a new consent-dependent consequence.

The operational failure would be to treat the queued membership as self-authorizing. A stronger architecture separates the durable campaign proposal from current send authority. The worker checks the authoritative preference state close to send time, or consumes a revocation event that invalidates any cached permit. If the current state is known, standing is defeated. If the authoritative preference service is required but unavailable, standing is unresolved.

The case illustrates why history should not be destroyed merely to prevent future action. The organization may need to retain the fact that consent existed when earlier messages were sent. What changes is not the truth of T_0 but the authority of T_0 over T_n.

The GDPR is conceptually useful here because Article 7 recognizes withdrawal while preserving the lawfulness of prior consent-based processing before withdrawal, and Article 5 includes purpose limitation. The law is more detailed than this example and may not apply to every communication, but its temporal structure is instructive. [4]

Standing test: Did the consent-dependent ground survive withdrawal for this exact send?

Expected outcome: Defeated.

Engineering control: Revocation propagation plus a pre-send standing check.

Case 2 - Human Approval Followed by Payload Drift

Type: Composite agent workflow.

An employee asks a purchasing agent to replace a failed workstation. The manager approves a specific cart totaling $4,820 with delivery to the Houston office. The agent workflow pauses while procurement credentials are refreshed. During the pause, the preferred model sells out. The shopping system automatically substitutes a similar model from another seller at $5,380 and a different delivery date.

A naive approval implementation stores `approved=true` on the workflow.

At T_0, the manager approved a bounded consequence: a particular purchase, amount, seller, destination, and perhaps delivery conditions.

ΔN is material payload drift.

At T_n, the question is not whether the manager approved *something*. The question is whether the approval still covers the actual transaction.

If the institution has a policy that authorizes substitutions under a $5,000 ceiling and the new amount exceeds that ceiling, standing is defeated or requires new authority. If the substitute remained within an explicitly approved tolerance, standing might be preserved. If the system cannot reconstruct what fields were material to the approval, standing is unresolved.

This is why human-in-the-loop design needs binding. A review event should identify the consequence being reviewed. Otherwise a human approval becomes a generic token that downstream components can accidentally stretch.

Standing test: Is the executed payload materially the same consequence that received approval, or within an explicit tolerance?

Expected outcome: Unresolved or defeated, depending on encoded purchase policy.

Engineering control: Canonical payload identity, material-field policy, and permit invalidation on drift.

Case 3 - Role Revocation During a Long-Running Workflow

Type: Composite enterprise scenario grounded in revocation and continuous-access patterns.

An operations employee starts an automated remediation workflow on Friday afternoon. The workflow is authorized to modify production infrastructure under that employee's on-call role. It performs diagnostics, waits for a maintenance window, and is scheduled to apply a configuration change early Saturday.

Friday evening, the employee is removed from the on-call role after a shift change. The workflow still contains a valid-looking session and the original task remains active.

At T_0, the employee had technical and organizational authority.

ΔN is a role change that is relevant to the control effect the workflow will create.

At T_n, a current authorization system may deny the operation even though the old workflow object still says "approved." This is the kind of change that Zero Trust, token revocation, and continuous access evaluation are designed to make visible to relying systems. CAEP provides standardized events by which cooperating systems can communicate changing access state, while OAuth token revocation provides a mechanism for invalidating previously issued tokens. [5][9][12]

Temporal Standing composes those security facts with the workflow's other grounds. If the institution's policy says that a launched workflow remains authorized under a service principal after human initiation, the role change may not defeat standing. If the employee's current role is a required predicate at execution, it does.

The key is that the answer comes from institutional authority rules, not from the agent's preference to finish what it started.

Standing test: Is the initiating person's current role still a required ground, or did authority legitimately transfer to a durable service principal?

Expected outcome: Policy-dependent; must be explicit.

Engineering control: Current role evaluation, explicit authority transfer semantics, and event invalidation.

Case 4 - Revocation Observed Out of Order

Type: Distributed-systems scenario grounded in Zanzibar's causal-ordering problem.

Alice grants Bob access to a shared resource. Later Alice revokes that access. Immediately after revocation, Bob requests an operation through a geographically distant service replica.

A system that observes the old access-control state after the revocation can effectively resurrect permission.

This is not an AI-specific problem. Google's Zanzibar paper describes an authorization system designed to maintain external consistency and respect causal ordering amid changes to access-control lists and object contents. [11]

Now add an AI agent. Bob's agent may have prepared a tool call while access was valid, then execute after the revocation. The model can truthfully remember that the access existed. The challenge is ensuring that the executor does not rely on a temporally older authority state than the revocation event.

At T_0, access was preserved.

ΔN is revocation.

At T_n, current standing is defeated. A stale replica is not a competing authority; it is an older observation of the same authority domain.

This distinction matters because models are good at treating multiple retrieved records as evidence to be reconciled semantically. Here the correct operation is not a semantic debate. It is causal ordering.

Standing test: Has the executor observed authority state at least as new as the relevant revocation?

Expected outcome: Defeated.

Engineering control: Version, sequence, causal token, or another monotonic ordering mechanism.

Case 5 - Ambiguous Provider Timeout

Type: Composite distributed-execution scenario grounded in idempotent-API practice.

An agent is authorized to issue a $250 refund. The executor sends the provider request. The network times out before the response arrives.

The system now knows something uncomfortable: it does not know whether the external consequence occurred.

A naive worker interprets the timeout as failure and sends another refund.

The second action may have perfectly valid standing in every ordinary sense—the customer is still eligible, the amount is still correct, the employee's authority still stands—but the first consequence may already exist. The problem is not stale permission. It is unresolved execution state.

AWS's guidance on idempotent APIs addresses this class of failure by giving clients a way to identify retries as the same intended request rather than distinct operations. [15]

Temporal Standing treats provider acceptance as part of the current consequence state. After an ambiguous timeout, the right ΔN is not necessarily "permission changed." It is "we may already have caused the effect." Standing for a *second* refund is therefore unresolved until reconciliation determines whether the first occurred.

This case shows why idempotency and governance belong together. Idempotency prevents duplicate technical execution. Temporal Standing prevents the system from describing a replay as a fresh legitimate consequence merely because the underlying business authorization still exists.

Standing test: Is this retry the same consequence, or would it create an additional consequence?

Expected outcome: Unresolved until reconciliation.

Engineering control: Stable idempotency identity, provider reconciliation, no blind regeneration.

Case 6 - A Durable Permission That Should Not Be Reapproved

Type: Composite counterexample.

A company grants an accounts-payable service principal authority to retrieve invoice metadata from a specific internal system for twelve months. The permission is deliberately durable, narrowly scoped, and revocable. A worker performs this low-risk read operation thousands of times per day.

Nothing material changes between T_0 and each T_n. The principal remains active. The resource and action stay within scope. No revocation

event occurs. Policy version remains applicable. The authorization source is healthy.

A poorly designed interpretation of Temporal Standing might demand a fresh human click before every read.

That would be wrong.

Standing can be preserved by durable authority. The framework does not require ceremony; it requires justified continuity.

The system may rely on a validity interval plus event-driven revocation and periodic health checks. It may cache authorization results where the institutional risk model permits. The relevant question is whether the evidence supporting preserved standing satisfies the declared freshness requirements for this consequence class.

This counterexample matters because a useful governance framework must permit automation, not merely stop it.

Standing test: Do the durable authorization and its current validity conditions still hold?

Expected outcome: Preserved.

Engineering control: Scoped durable grant, revocation path, bounded caching.

Case 7 - Delegation to a Sub-Agent

Type: Composite multi-agent scenario.

A travel agent is authorized to plan and purchase airfare for a conference under a $1,500 total budget. To find better fares, it delegates search to a specialized sub-agent. The sub-agent discovers a package that includes airfare and a non-refundable hotel for $1,420 and proposes purchasing both.

The total is under budget, but the delegated scope was airfare search, not lodging purchase.

The failure would be to infer authority from goal alignment: because the package helps accomplish the trip and remains under budget, it must be allowed.

At T_0, the parent had purchase authority over airfare under constraints.

ΔN is not necessarily a change in reality. It is a change in the proposed consequence surface created by delegation.

At T_n, the new hotel commitment lacks a governing ground unless the original authority tuple included it.

Temporal Standing therefore applies to scope transitions as well as chronological changes. The notation uses ΔN for relevant intervening change, but the important principle is that the consequence at T_n must still be covered by the authority established at T_0.

Standing test: Did delegated authority attenuate correctly, and does the new consequence remain inside the parent's delegable scope?

Expected outcome: Defeated for hotel purchase; preserved for search.

Engineering control: Delegation tokens or structured scopes that cannot expand downstream.

Case 8 - Emergency Override

Type: Composite governance counterexample.

A hospital facility system normally requires two approvals before shutting down a critical environmental subsystem. A sensor reports an immediate physical hazard. Waiting for the ordinary approval chain could increase harm.

A simplistic fail-closed design might block the shutdown because one approver is unreachable.

A mature authority model can encode emergency authority as a separate path. The emergency does not "bypass governance." It activates a different governing rule with its own predicates, scope, logging, expiry, and retrospective review.

At T_0, ordinary authority requires two approvals.

ΔN is an emergency state established by an authoritative source.

At T_n, the emergency rule may preserve standing for a narrowly defined protective action that would otherwise be denied.

This matters because Temporal Standing is not synonymous with conservative refusal. It asks what authority stands now. Sometimes a changed condition grants authority rather than defeats it.

Standing test: Has an authoritative emergency condition activated a legitimate alternate authority path for this exact protective consequence?

Expected outcome: Preserved under emergency authority if all emergency predicates are satisfied.

Engineering control: Explicit break-glass policy, bounded scope, strong audit, rapid expiry, and review.

What the cases reveal

Across these examples, the framework is doing the same work without assuming the same domain rule.

It identifies the prior grounds.

It identifies the relevant change, conflict, uncertainty, or scope shift.

It asks what authoritative evidence is current.

It produces preserved, defeated, or unresolved standing for an exact consequence.

And it requires the execution path to obey that determination.

That regularity is the reason to have a named framework at all.

Appendix D - Validating Temporal Standing

A governance framework should expose itself to failure.

Temporal Standing can be evaluated at three layers: conceptual adequacy, architectural enforcement, and operational performance.

1. Conceptual adequacy

For a selected consequence, ask whether the framework can name the required governing predicates without circularity.

If the answer to "what must stand?" is merely "whatever the policy says," the model may be too abstract to help. The implementation should be able to identify concrete predicate classes: authority, identity, scope, state, policy, evidence, and consequence identity.

Then ask whether each predicate has a legitimate authoritative source. If no one in the institution can say which record controls, the problem is organizational before it is technical.

Finally, test the three outcomes. There should be real cases where standing is preserved, real cases where it is defeated, and real cases where it is unresolved. If every ambiguity is forced into one of two buckets, the implementation has discarded a core part of the framework.

2. Architectural enforcement

A standing decision that cannot stop an effect is advisory, not governing.

Tests should therefore be causal.

Inject a denial and verify that the external provider is not called.

Invalidate a permit and verify that the executor rejects it.

Mutate a material payload field and verify that the old permit no longer matches.

Attempt the same consequence through an alternate connector and verify that the governance path cannot be bypassed.

Revoke authority after a proposal is queued and verify that the queued worker observes the change.

Force the policy service offline and verify that required consequences become unresolved rather than permitted by default.

These tests are closer to security testing than to model evaluation. They ask what the system can cause, not merely what it says.

3. Temporal race testing

Any system that separates determination from execution has a race window.

Testing should make that window visible.

Create a valid standing decision, then inject a defeating ΔN immediately before execution. Measure whether the executor observes it.

Repeat across storage replicas, regions, queues, and retries.

If a standing permit has a validity interval, test the edges of the interval. Verify that clock skew, delayed jobs, or retried messages do not extend authority accidentally.

If event-driven invalidation is used, measure propagation time and define the maximum tolerated lag for each consequence class.

The result should be a quantitative statement such as: "revocation is enforced at this boundary within X under normal conditions, and the action fails closed if freshness exceeds Y." That is more meaningful than saying the system is "continuously governed."

4. Unresolved-state testing

Unresolved needs its own test plan.

Take away a required evidence source.

Introduce conflicting versions.

Return malformed policy data.

Make identity verification indeterminate.

Create a provider timeout after possible acceptance.

The system should preserve the reason for uncertainty and route it toward the correct remediation. It should not turn every unresolved case into a generic retry loop, nor should it ask a human to approve without showing what is missing.

5. Counterexample testing

A restrictive framework can look safe while destroying usefulness.

Test durable authority that should remain preserved.

Test low-risk actions that are intentionally cacheable.

Test emergency authority that becomes valid only after a new condition appears.

Test a benign ΔN that is unrelated to the consequence and confirm that it does not trigger reauthorization.

These cases protect the framework from "maximum interruption" thinking.

6. Multi-agent delegation testing

Give a parent agent narrow authority and ask it to delegate.

The child should inherit no more than the parent's delegable scope.

Then revoke the parent authority and verify what happens to outstanding delegated permits according to policy.

Change the sub-agent implementation while keeping the same consequence contract and verify that governance remains stable. This tests

whether authority is attached to the effect and principal rather than accidentally to one model implementation.

7. Metrics that matter

A production Temporal Standing system should be able to report more than permit and deny counts.

Useful measures include:

- percentage of consequences evaluated with current required evidence;
- preserved, defeated, and unresolved rates by consequence class;
- unresolved age and cause distribution;
- revocation propagation latency;
- payload-mismatch rejection rate;
- bypass attempts or unenforced paths;
- ambiguous-provider reconciliation rate;
- duplicate-effect prevention rate;
- human-review rate and reasons;
- unnecessary reauthorization rate;
- emergency-authority usage;
- standing-decision latency at p50, p95, and p99;
- false interruption reports from users and operators.

These measures expose tradeoffs. A system with zero unjustified executions but a 70 percent unresolved rate may be unusable. A system with near-zero latency because it serves month-old cached policy may be unsafe. Governance quality exists in the balance between justified autonomy and justified restraint.

8. The falsification standard

The strongest challenge to Temporal Standing would be an existing architecture that reliably provides the same properties under a different name: exact-consequence scope, heterogeneous current authority composition, explicit uncertainty, change relevance, bounded permits, and causal enforcement at the consequence boundary.

Where such a system exists, the book should acknowledge it rather than relabel it.

The framework is useful only if it helps builders see, design, test, or communicate something they would otherwise miss.

That is the standard it should be held to.

Primary Sources and Comparative References

The sources below support the comparative and implementation claims used throughout this book. Their inclusion does not imply that any cited source endorses Temporal Standing as a framework.

[1] **NIST, Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1 (2023).** NIST organizes AI risk management around Govern, Map, Measure, and Manage and treats risk management as continuous and lifecycle-wide. The framework is broader than Temporal Standing; it supplies organizational risk-management context rather than an exact-consequence execution model. https://doi.org/10.6028/NIST.AI.100-1

[2] **Regulation (EU) 2024/1689 (EU AI Act), Article 14 - Human oversight.** Article 14 requires high-risk AI systems to support effective human oversight during use and makes oversight measures proportionate to risk, autonomy, and context. Used here to distinguish meaningful oversight from ceremonial approval. https://eur-lex.europa.eu/eli/reg/2024/1689/oj

[3] **Saltzer, J.H. and Schroeder, M.D., "The Protection of Information in Computer Systems," Proceedings of the IEEE 63(9), 1278-1308 (1975).** A foundational security paper associated with principles including complete mediation and least privilege. Temporal Standing draws on the complete-mediation intuition but applies it to heterogeneous grounds for external consequences. DOI: 10.1109/PROC.1975.9939. https://web.mit.edu/Saltzer/www/publications/protection/

[4] **Regulation (EU) 2016/679 (GDPR), Articles 5 and 7.** Article 5 includes purpose limitation; Article 7 provides for withdrawal of consent and states that withdrawal does not affect the lawfulness of consent-based processing before withdrawal. Used as a clean legal illustration of historical truth coexisting with changed present standing. https://eur-lex.europa.eu/eli/reg/2016/679/oj

[5] **IETF RFC 7009, OAuth 2.0 Token Revocation (2013).** Defines a revocation endpoint that permits previously obtained tokens to be invalidated. Used as an example of a system in which previously valid credentials can lose present usability before natural expiration. https://www.rfc-editor.org/rfc/rfc7009

[6] **OpenAI, "Guardrails and human review," Agents documentation (accessed October 2026).** Current agent guidance distinguishes input/output/tool guardrails from human approval and recommends placing checks around side-effecting tool calls. Used as contemporary evidence that agent engineering is moving controls toward consequence-producing tools. https://developers.openai.com/api/docs/guides/agents/guardrails-approvals

[7] **OWASP GenAI Security Project, LLM06:2025 Excessive Agency.** Recommends limiting functionality and permissions, executing actions in the user's authorization context, using human approval for high-impact

actions, and enforcing authorization downstream rather than relying on an LLM. https://genai.owasp.org/llmrisk/llm062025-excessive-agency/

[8] NIST SP 800-162, Guide to Attribute Based Access Control (ABAC) Definition and Considerations (updated 2019). Defines ABAC around subject, object, requested operation, environmental conditions, and policy. Used as a major prior-art neighbor and implementation primitive. https://doi.org/10.6028/NIST.SP.800-162

[9] NIST SP 800-207, Zero Trust Architecture (2020). Rejects implicit trust based on network location or ownership and centers explicit authentication and authorization around resources. Used as a security neighbor that shares the refusal to treat historical trust as permanent. https://doi.org/10.6028/NIST.SP.800-207

[10] Cedar Policy Language Reference Guide, "How Cedar authorization works" (accessed October 2026). Cedar models authorization requests around principal, action, resource, and context and returns Allow or Deny under policy. Used to distinguish policy authorization from broader standing composition. https://docs.cedarpolicy.com/auth/authorization.html

[11] Pang, R. et al., "Zanzibar: Google's Consistent, Global Authorization System," USENIX ATC 2019. Zanzibar provides authorization decisions that respect causal ordering amid changing ACLs and object contents. Used to illustrate why authority-changing events must not be observed out of order. https://research.google/pubs/zanzibar-googles-consistent-global-authorization-system/

[12] OpenID Foundation, Continuous Access Evaluation Profile 1.0 (August 2025). Defines shared security events that cooperating systems can use to continuously update and attenuate access for users, devices, sessions, and applications. Used as an example of event-driven authority change propagation. https://openid.net/specs/openid-caep-1_0.html

[13] Park, J. and Sandhu, R., "The UCONABC Usage Control Model," ACM Transactions on Information and System Security 7(1), 128-174 (2004). UCON integrates authorization, obligations, conditions, continuity/ongoing controls, and mutable attributes. This is one of the closest intellectual neighbors to Temporal Standing and strongly limits any novelty claim based merely on changing authorization over time. DOI: 10.1145/984334.984339. https://doi.org/10.1145/984334.984339

[14] Open Policy Agent, deployment and policy documentation (accessed October 2026). OPA describes itself as a general-purpose Policy Decision Point used by applications acting as Policy Enforcement Points and recommends locating decision capability near enforcement where appropriate. Used to distinguish correct decisions from actual enforcement. https://www.openpolicyagent.org/docs/deploy

[15] Featonby, M., "Making retries safe with idempotent APIs," Amazon Builders' Library (2021). Discusses retrying side-effecting operations safely using stable client request identities and idempotent API design. Used to separate execution ambiguity and duplicate-side-effect prevention from standing itself. https://aws.amazon.com/builders-library/making-retries-safe-with-idempotent-APIs/

Comparative conclusion

These sources narrow the defensible contribution.

Temporal Standing should not be marketed as the discovery that authorization changes, that access should be mediated, that revocation exists, that policies can include current context, or that human oversight matters. Those ideas are established.

The stronger contribution is the **composition discipline** around AI-mediated consequences: identify the heterogeneous grounds that support an exact consequence, track relevant change from T_0 through ΔN, preserve a third unresolved state when current authority cannot be established, bind the determination to the material consequence, and make execution causally depend on that determination.

About the Author

Tim Zlomke is the founder of Moral Clarity AI and managing partner at Neurovia Dynamics. His work focuses on runtime AI governance, accountable automation, and systems that preserve human authority as machine capability increases. Temporal Standing emerged from building and testing systems in which a correct historical record was not enough to justify a present consequence.