

When Is an AI Agent Done?

Evidence-Bound Completion for Shared Work

Max Song September 29, 2026 Version 1.0 31 min read

Self-published preprint · Not peer reviewed · Prepared with AI assistance

Abstract

An agent can finish a computation after its assignment has expired, verify a file that another writer has since changed, or submit a plausible success report after its permission has been revoked. A shared conversation does not resolve these disagreements. This report studies a narrower question than agent intelligence: what must a coordination service establish before it records a shared task as completed? Motivated by the Studiolo project, we specify a completion contract joining a pinned task definition, current attempt ownership, live authorization, declared artifact revisions, and verifier-issued evidence at one commit boundary. The design applies established concurrency and authorization mechanisms; it does not introduce a new lock service, prove arbitrary work correct, or claim to be the first receipt-based agent architecture. We provide an executable, bounded event-interleaving study, mechanism ablations, and explicit counterexamples to checking before committing. The evaluation measures acceptance errors in a synthetic state machine, not language-model accuracy or production reliability. Its central finding is a design requirement: evidence must remain attached to the particular work and authority it describes until the authoritative transition occurs. A successful test, a successful commit, and a successful deployment remain different statements.

中文摘要

这篇论文源于我对 Studiolo 的设想：一个入口，让不同客户端里的 AI 共享项目背景、规则与工作状态，并能接着彼此的工作继续做。本文把这个大想法收窄成一个可检验的问题：AI 说“完成”时，服务器究竟应该相信什么？我提出的完成契约将任务约定、当前执行身份、实时权限、文件版本和验证回执绑定在同一次提交上，并通过可复跑的有限事件交错实验检验各项约束。黑板、租约、版本检查和回执都有已有研究；本文的价值在于把它们放进同一个明确的提交边界，展示遗漏条件或拆开检查与提交会怎样失败。实验不调用真实大模型，也不证明生产系统或任意任务正确；正文、实验源码、结果和引用资料一起公开。

1. From shared memory to a completion contract

The starting point was practical. In September 2026, I described Studiolo as a common entry point through which different AI clients could discover a project's context, rules, tools, and ongoing work. One client might shape a product, another investigate it, and another implement it. The service would maintain shared state; it would not need to own every client's process or choose every next action.

The project's design records subsequently distinguished externally operated clients from workers launched inside the service's control boundary. They also specified versioned handoffs, attempts with leases and fencing tokens, artifact references, and structured returns. Those records motivate this report. They are not a published dataset, and their existence is not evidence that every proposed guarantee is implemented. The executable model released here is a separate research artifact, written for this report. No private conversation, project document, credential, or production task record is included in it.

The strongest research question within the original idea is not how to put more context into a prompt. It is how a successor can tell which statements still describe the shared project. Consider four plausible reports:

1. "I finished the task." The server reassigned it while the first agent was paused.
2. "The tests pass." They ran against a candidate that was replaced before submission.
3. "I had permission." Permission was removed after the agent read the task.
4. "The new rules are available." A new behavioral handbook was published, although this task still legitimately follows its original contract.

The first three can invalidate a completion attempt. The fourth need not. Treating every old version alike loses this distinction: an old rulebook can be the correct rulebook, while an old authorization decision can be unusable.

Research question. Under explicitly stated trust and storage assumptions, which checks must share an authoritative commit boundary to prevent obsolete or unsupported completion records, while admitting work that remains valid?

The report contributes three limited artifacts: a precise completion predicate and its trust boundary; a bounded, reproducible interleaving study with ablations and negative controls; and an implementation argument explaining where the model stops applying. It does not claim an improvement in model reasoning, a new consensus algorithm, an industry-wide failure rate, or superiority over a deployed agent platform.

2. Prior work and the scope of the contribution

Shared workspaces and guarded state transitions have a long history. Linda describes coordination through shared tuples ¹. Chubby describes sequencers that recipients can use to reject obsolete lock holders, as well as conditional updates using generation information ². Optimistic concurrency control

validates earlier reads before allowing a transaction to publish ³. Linearizability provides a way to reason about an operation as taking effect at one point between invocation and response ⁴. These are foundations, not mechanisms invented here.

Authorization also has a time dimension. Zanzibar explicitly addresses consistency between access decisions, changes in permissions, and changes in content ⁵. Our single-authority model uses a smaller setting: revocations and completion decisions must participate in a common serialization order. We make no claim to reproduce Zanzibar's distributed scale or availability.

The closest agent-specific work reviewed is **TraceGrant** ⁶. It connects a task contract, admitted evidence, guarded tool effects, effect receipts, and a final-answer gate. It already distinguishes permission from successful execution and evaluates receipt and completion-gate ablations. Its certificates also have atomic consumption requirements. Therefore, neither “agents need contracts” nor “completion should require evidence” is a novelty claim of this report. Our narrower emphasis is the late return of an independently operated worker into a shared, mutable workspace: attempt reassignment, artifact revisions, current authorization, and the interval between validation and commit. We do not benchmark TraceGrant or infer that it fails these cases. Its reported benchmark results are not results of our experiment.

AgentDojo provides an environment for studying indirect prompt injection in tool-using agents ⁷. The multi-agent failure study by Cemri and colleagues identifies task verification and termination among its failure categories ⁸. AutoGen studies conversational multi-agent applications ⁹. These works motivate care about evaluation and coordination, but our deterministic reference clients neither reproduce their benchmarks nor represent measured model behavior.

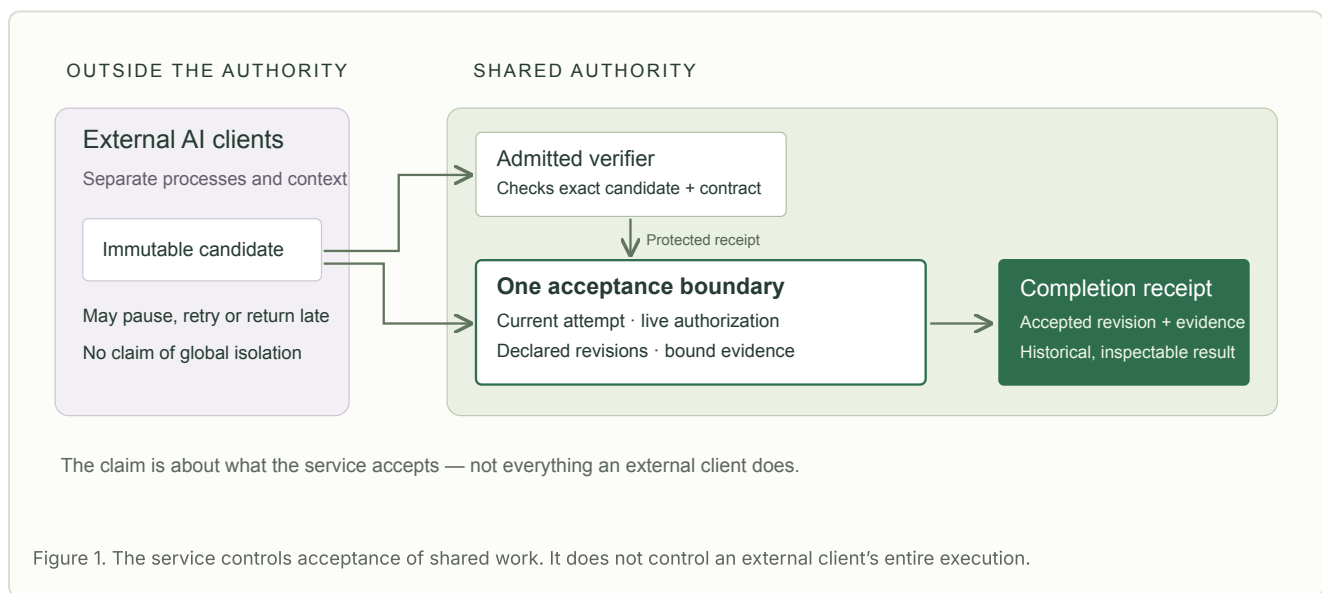
Evidence needs a modest name. Proof-carrying code supplies mechanically checkable proofs for a defined safety policy ¹⁰. A passing test receipt is weaker: it records the outcome of a particular checker on particular inputs. W3C PROV supplies concepts for representing provenance ¹¹; provenance alone does not establish that an assertion is true or that its subject is current.

A2A v1.0.0 has task states, artifacts, and a completed terminal state ¹². The MCP Tasks extension dated 2026-07-28 has its own lifecycle and authorization requirements; a completed tool task can still carry `isError: true` in its result ¹³. These are not stateless or authorization-free protocols. Their interoperability models should not, however, be mistaken for an application-specific transaction over a shared workspace. A service may implement the contract described here behind either interface. The state-name mapping must be explicit.

Table 1. Established mechanisms and the question asked at the shared completion boundary.

Existing idea	What it establishes	What this report additionally asks at completion
Shared tuple space / blackboard	A common place to exchange and inspect work	Which transition is authoritative?
Lease and fencing	Time-bounded ownership and rejection of obsolete holders	Does the accepted artifact still match the evidence?
Optimistic validation	Whether declared reads remain compatible with a write	Who is currently authorized to make that write?
Receipt-backed task closure	Support for required effects or obligations	Is this receipt bound to this shared attempt and revision?
Provenance	Where a claim or artifact came from	Is its checker trusted, relevant, and sufficient?
Durable workflow	Recovery and replay of recorded execution	Has an external effect actually occurred exactly as claimed?

Durable execution is another adjacent concern. Temporal documents retries and idempotency requirements for activities¹⁴. Our accepted completion record is not an exactly-once guarantee for arbitrary external effects. A replay-safe local ledger cannot undo an email sent twice.



3. System model and the boundary of trust

3.1 Three parties, two kinds of authority

There are externally operated workers, an authoritative coordination service, and admitted verifiers. A worker can read permitted context, prepare a candidate, and request completion. It may pause, retry, crash, or submit a misleading report. The service owns the task ledger and the publication pointers for the

artifacts within scope. A verifier checks an immutable candidate under a named criterion and records its observation in a protected evidence store.

“External” describes control, not vendor identity. Two sessions using the same model may sit on opposite sides of this boundary. The service can restrict what an external client reads from it and what that client commits through it. It cannot establish that the client made no other network requests, kept no copy of a document, or changed no local files. Even read access releases information beyond the service's control.

There are also two different meanings of authority. The **task contract** says what successful work means; it is pinned for the attempt. **Live authorization** says whether the principal may perform this commit now. Publishing a new default handbook does not silently rewrite an existing contract. Revocation can still remove the ability to commit under that old contract. An explicit cancellation or contract migration is another state transition, not a hidden side effect of changing defaults.

3.2 Objects

Let a task contract be $C = (\text{task}, \text{contractHash}, \text{criteria}, \text{scope})$. The criteria name the required checks and their admitted checker versions. The scope describes the permitted artifact targets and relevant dependencies.

An attempt is $A = (\text{task}, \text{attemptId}, \text{principal}, \text{fence}, \text{leaseDeadline})$. The fence increases when ownership is replaced. A candidate x contains immutable output bytes, a digest, and a manifest of the versions on which the work depends. The output target's expected base revision also appears in that manifest. We call this declared set D ; it is not a list of every file the worker happened to open. The service derives the actual write set W from the candidate's publication manifest and canonicalizes its targets; it must not trust a worker-supplied scope label as a description of those effects.

A verifier receipt E identifies the task, attempt, contract hash, candidate digest, dependency-manifest digest, criterion identifier, checker version, and observed result. An opaque receipt identifier can point to a service-owned record. A digest is an integrity identifier, not proof that its producer is trustworthy. Anyone can hash a fabricated log.

The service state s contains task status, the active attempt, the grant state, artifact revision pointers, admitted evidence records, and committed request identifiers. A completion request is $Q = (A, C, X, D, \text{evidenceIds}, \text{requestId})$. The model uses symbolic identifiers for these bindings; it does not evaluate cryptographic collision resistance.

3.3 Assumptions

The safety argument requires one authoritative serialization boundary for the state relevant to completion. Artifact publication, attempt replacement, revocation, and completion cannot each proceed through an unrelated unsynchronized path. Verification operates on immutable candidate bytes. Every admitted receipt refers to exactly those bytes and the declared dependency manifest. All writers that can change the

authoritative artifact pointers participate in versioning. The checker registry and evidence store are protected from the submitting worker. The lease clock belongs to the authority and does not move backward within an authority incarnation.

These assumptions are substantial. A folder that a human can overwrite without informing the coordinator does not satisfy them. Neither does an authorization service whose answer may be arbitrarily stale at commit. The candidate's undeclared semantic dependencies are outside the guarantee. The report assumes a correct coordinator and admitted verifiers; a compromised service is not repaired by adding a hash field.

4. The completion contract

4.1 A conjunction evaluated at the transition

For a request that has not already committed, acceptance requires all the following predicates to hold at the logical acceptance point. The authenticated caller is `u`; it is derived from the authenticated channel, not accepted as a self-asserted field in the request:

```
Accept(u, Q, S, now) =
  ActiveTask(Q.task, S)
  ∧ CallerMatchesAttempt(u, Q.attemptId, S)
  ∧ OwnsCurrentAttempt(Q.attemptId, u, Q.fence, S)
  ∧ LeaseLive(Q.attemptId, now, S)
  ∧ ActualWriteSet(Q.candidate) ⊆ PinnedScope(Q.task, S)
  ∧ AuthorizedNow(u, ActualWriteSet(Q.candidate), S)
  ∧ MatchesPinnedContract(Q.contractHash, S)
  ∧ DependenciesCurrent(Q.declaredVersions, S)
  ∧ CandidateBoundToEvidence(Q.candidate, Q.evidenceIds, S)
  ∧ RequiredCriteriaSupported(Q.contractHash, Q.evidenceIds, S)
```

Each predicate answers a different question. Ownership identifies which attempt may act; a live lease determines whether its time window remains open. Authorization is independent of both. For example, a principal allowed to write a scratch area cannot label a production-target candidate as “scratch” and pass the grant check: the server checks the actual normalized target set. Matching the contract checks the assigned obligation, not whichever handbook happens to be newest. Revision validation protects the declared inputs and publication target. Evidence validation checks identity, provenance, candidate binding, and coverage of the criteria.

The reference experiment deliberately abstracts this larger contract to a small number of symbolic checks. Section 6 lists its exact bounds. Fields such as path normalization, cryptographic authentication, and multi-file publication require engineering beyond the model.

4.2 Why a preliminary check is insufficient

Suppose a verifier approves candidate `x1` against revision `r7`. The server checks the request and finds every condition true. Another writer then publishes `r8`. Finally, the server records completion using its earlier decision. Each component has behaved according to its local procedure, yet the accepted request is obsolete.

Checking twice does not solve the problem if a relevant change can occur after the second check. The requirement is that validation and the authoritative publication transition form one operation relative to competing mutations. An expensive check need not run inside a database transaction: it can run beforehand on immutable data, produce a protected receipt, and have that receipt's bindings validated at commit. What cannot be moved out without additional coordination is the final validation of mutable authority and versions.

Here, “commit point” means the logical acceptance or linearization point, not the later network response or durable-write completion time. The lease authorizes admission at that point. Expiration afterward does not retract an admitted operation. A normal database transaction does not stop wall-clock time: an implementation must place its authoritative time check at its defined admission point, rather than reuse a timestamp from a much earlier preflight. A requirement that every physical write finish before a wall-clock deadline is stronger and is not established here. The finite model represents expiration as an ordered event rather than modeling clock drift or transaction duration.

The illustrative transaction is:

```
complete(request):
  authenticate caller
  begin authoritative transaction
  if requestId already committed:
    return prior receipt only for the identical bound request
    and only if the caller may read that receipt
  read task, attempt, grant, contract, manifest, evidence
  bind authenticated caller and derive actual normalized write targets
  reject unless all completion predicates hold
  publish immutable candidate through versioned pointers
  record completed(task, attempt, candidate, evidence, commitVersion)
  record requestId and append the audit event
  commit
  return authoritative completion receipt
```

A duplicate request is a retrieval of a historical result, not a new authorization to execute. A reused identifier with a changed candidate must be rejected. Receipt visibility still requires authorization; an idempotency key must not become a way to read revoked data. The model tests effect idempotency, not a complete receipt-access API.

4.3 Pin behavior; recheck permission

The following two histories should have different outcomes. In the first, a task is assigned under contract `c1`, an unrelated new default `c2` is published, and the worker submits valid evidence for `c1`. If the task remains pinned to `c1`, rejecting solely because `c2` exists is unnecessary rejection. In the second, the same worker's grant is revoked before commit. A valid `c1` receipt cannot restore that grant.

This distinction avoids two opposite mistakes. Checking only the pinned policy can preserve permissions that should have ended. Requiring equality with the latest behavioral policy can discard legitimate work whenever a rulebook changes. A service should expose separate operations for publishing defaults, migrating a task contract, cancelling a task, and revoking access.

4.4 Evidence has identity, provenance, and coverage

A receipt must refer to the candidate that will become authoritative. “Tests passed” without a digest is a statement about an unknown object. A digest without an admitted issuer is a statement from an unknown authority. A genuine receipt for one criterion does not establish that all criteria passed. A genuine receipt for another task or attempt cannot automatically be imported.

Evidence reuse is possible, but it should be explicit. For example, a content-addressed type-check result could be reused if the checker version, dependency manifest, criterion, and policy for reuse all match. The conservative contract here binds evidence to the attempt and rejects cross-attempt reuse. That choice favors simple reasoning over maximum throughput; it is not a claim that all useful systems must make the same tradeoff.

Even a perfectly bound receipt says only that its checker returned the admitted result. A test can be incomplete. A specification can be wrong. A verifier can be compromised. The intended distinction is between an unsupported report and a supported observation, not between machine output and infallible truth.

4.5 Shared artifacts and external effects

One practical implementation stages candidate blobs by digest, then atomically changes a manifest pointer together with the completion record. Unreferenced staged blobs can be garbage-collected later. A crash before commit leaves no accepted task result; a crash after commit but before response can be recovered through the request identifier. This describes a design option, not a performance result from an implemented storage service.

If the canonical content is ordinary mutable files, a transaction in a separate database does not automatically cover file replacement. A broker needs a recoverable publication protocol and a read rule that recognizes partially applied state. A projection into a folder may lag behind an accepted manifest. A receipt that observes the folder establishes `Observed(target, revision, time)`, a historical fact. It does not establish that the same revision is still there at a later completion decision. If the criterion requires the external target to be current at acceptance, its version changes must participate in the same authority boundary or an equivalent coordinated protocol. Otherwise that stronger criterion remains unverified.

The same applies to deployment. A committed source revision, a deployed release, a healthy endpoint, and a page a browser can load are separate observations, each with a revision and time. A contract may require evidence that all were observed, but should not silently upgrade that into simultaneous or continuing availability. Arbitrary network services require their own idempotency or reconciliation mechanisms. We do not turn a local commit into a distributed exactly-once transaction by calling the returned object a receipt.

5. Safety claims and their limits

Claim 1: obsolete attempt exclusion. If attempt replacement and completion serialize through the same authority, and completion checks the current attempt identity and fence, a replaced attempt cannot produce a new accepted completion. Before replacement it may commit; after replacement its identity no longer matches. Those are different histories, not a race the client can decide by its own clock.

Claim 2: declared revision freshness. If every authoritative revision change uses the same serialization boundary, validation of the declared read and target versions at commit prevents acceptance against an intervening conflicting revision. This says nothing about dependencies omitted from the manifest or content that changes outside the authority.

Claim 3: revocation ordering. If revocation commits before completion, the live grant check rejects the completion. If completion commits first, subsequent revocation does not make the historical authorization decision false. It can prevent future access or actions. A system requiring retrospective invalidation needs a separate revocation-of-result policy.

Claim 4: evidence binding. If the evidence registry is protected and receipt bindings are validated, an accepted completion has the required admitted observations for its exact contract and candidate. This does not imply the candidate is semantically correct beyond those observations.

The proof argument is an induction over serialized transitions. Initially there is no completion record. Transitions that do not create one cannot introduce an invalid new completion. A completion transition checks the conditions on its own pre-state and records the bound objects together. Thus a record that is created has those properties at its commit point. The argument depends on the implementation actually realizing the transition atomically. It is an informal proof sketch, not a machine-checked proof or verification of Studiolo's implementation.

After completion, a document may legitimately change. The receipt continues to describe the committed revision; it does not certify the latest revision forever. A user interface should therefore show both the historical accepted result and whether that result remains the current artifact.

No unconditional progress guarantee follows. A worker may lose its lease repeatedly, a verifier may be unavailable, or a human may keep editing the dependency set. Under eventual quiescence, continuing authorization, available verifiers, and a sufficiently long lease, an eligible request can commit. Without

such assumptions, failing closed trades availability for acceptance integrity. A permanently rejecting service is safe in a vacuous sense and useless in practice; the evaluation therefore also counts valid requests that are rejected.

6. Executable bounded study

6.1 What was measured

The downloadable reference model, method and reproduction notes, and raw results form the evaluation artifact. It uses only Node.js built-in modules. It makes no model API calls, fetches no private documents, and does not contact a running coordination service. The model and manuscript were developed together; this is an explanatory validation study, not an independently preregistered benchmark.

The synthetic task is to double a number. There is one submitting worker, one replacement owner that never prepares its own candidate, one input object, one working artifact, and a fixed task contract. The main chain is `read → candidate → trusted-verify → preflight → commit`. Six optional environment events can be inserted: change the input, change the artifact, expire the lease, replace the attempt, revoke the submitting principal's permission, or publish a new default behavioral policy. Each selected event occurs once. We enumerate all subsets containing zero through three of those six events, giving 42 scenarios. If expiration and reassignment both occur, expiration precedes reassignment. A reassignment without an expiration event represents an explicit replacement.

Every topological ordering within these bounds is enumerated. A scenario with no environment event has one ordering; a single event has six; two unrelated events have 42; expiration followed by reassignment has 21. Three unrelated events have 336 orderings. Across all scenarios, this gives **6,694 schedules**. Thirteen mechanisms run on every schedule: **87,022 primary executions**. Seven directed negative controls run once per mechanism, adding **91 executions**. These counts exclude additional sanity checks and the replay assertions described below.

The candidate event takes a byte snapshot and writes a **mutable working copy**. An artifact-edit event can subsequently change that copy. The commit event records a completion receipt; it does not implement candidate publication, target-base compare-and-swap, a multi-file transaction, or a deployed storage service. The output binding check establishes that the working bytes still equal the candidate snapshot. Input revision validation protects the one declared dependency. The broader publication design in Section 4 is not experimentally validated by this smaller model.

Identity, target scope, and the admitted checker are fixed. Authorization is represented by a revocation flag. Lease liveness is a discrete state at the acceptance event. The atomic mechanism assumes an indivisible transition. SHA-256 is used for deterministic byte identifiers, not as a tested cryptographic security claim. Schedules can contain events after commit: those events must not retroactively invalidate a historical receipt. Consequently, different full schedules may share the same decision prefix; we report full schedules, not unique reachable states.

The outcome oracle is implemented separately from the mechanism under test. It reconstructs relevant facts from the event history and protected verification records. This avoids calling the tested acceptance predicate to label its own decision. It does not remove common-model bias: the implementation and oracle still share the same modeling choices and were created in this study.

We count four outcomes at a completion decision: safe accept, unsafe accept, safe reject of an unsafe request, and unnecessary reject of an eligible request. Counts are schedule counts under the specified finite enumeration. They are not estimated probabilities in real workloads. Scenarios receive no empirically justified frequency weights, and no confidence interval is meaningful for this complete finite population.

6.2 Results

Table 2. Completion decisions across the same 6,694 schedules for each mechanism. Every row sums to 6,694. The oracle classifies 470 decision points as eligible and 6,224 as ineligible.

Mechanism	Safe accept	Unsafe accept	Unsafe reject	Eligible reject
Self-report	470	6,224	0	0
Attempt + lease only	470	1,914	4,310	0
Full atomic contract	470	0	6,224	0
Check then commit	470	718	5,506	0
Without numeric fence	470	0	6,224	0
Without attempt binding	470	667	5,557	0
Without lease	470	517	5,707	0
Without input revision check	470	328	5,896	0
Without current authorization	470	512	5,712	0
Without evidence checks	470	0	6,224	0
Without artifact binding	470	134	6,090	0
Without contract binding	470	0	6,224	0
Require latest behavioral policy	234	0	6,224	236

“Unsafe reject” means correctly rejecting an ineligible request; “eligible reject” means unnecessarily rejecting a valid request. These are finite schedule counts, not workload rates.

The full atomic mechanism accepts all 470 eligible cases and none of the 6,224 ineligible ones. The same checks cached at preflight admit **718 ineligible completions**. This is an executable demonstration of the validation-to-commit gap within the modeled histories, not evidence of a universal reduction in real-world errors.

Attempt and lease checks alone still admit 1,914 ineligible cases. Removing the complete attempt binding admits 667; removing lease liveness admits 517; omitting the input revision check admits 328; omitting current authorization admits 512; and omitting artifact binding admits 134. These errors overlap across mechanisms, so the counts must not be added into a total or interpreted as independent effect sizes.

Two non-results are informative. First, removing only the numeric fence produces no additional failure because the model retains unique, never-reused attempt identity. The experiment therefore supplies **no evidence for an independent numeric-fence benefit** in this particular design. The separate attempt-binding ablation removes principal, attempt, and fence checks together. Second, omitting evidence or contract binding also produces no new primary failure: the primary generator always supplies a genuine verification record and the assigned contract, and other checks detect the modeled changes. The directed negative controls test those omitted obligations instead.

Requiring the latest default behavioral policy rejects **236 eligible completions** without blocking any additional ineligible one. That result is specific to the stated policy: publication is not revocation and existing tasks stay pinned. A system that intentionally migrates or cancels those tasks would be specifying a different experiment.

The ablations are deliberately incomplete designs. Their role is to expose the obligation carried by each mechanism, not to represent implementations of A2A, MCP, TraceGrant, Chubby, or a commercial product. None of those systems was run in this experiment.

6.3 Counterexamples

The raw JSON includes short counterexamples and their event histories. Three illustrate different failures:

A valid check becomes obsolete.

```
read → candidate → trusted-verify → preflight
      → artifact-edit → commit
```

The split mechanism accepts the cached preflight result even though the current artifact no longer matches the verified candidate. The full mechanism rejects at acceptance. Moving the artifact edit before preflight makes the split mechanism reject too, which shows why the event ordering matters.

Ownership survives, but the dependency does not.

```
read → candidate → input-edit → trusted-verify
      → preflight → commit
```

The verifier checks the candidate against the worker's original read snapshot. That observation can be genuine while the shared input has already changed. Attempt-and-lease-only acceptance admits the obsolete result. The input revision check rejects it.

A harmless policy publication becomes an unnecessary rejection.

```
publish-behavior-v2 → read → candidate
  → trusted-verify → preflight → commit
```

The task was already pinned to behavior-v1 before this trace begins. Nothing revokes the principal or migrates the task. The overstrict mechanism rejects solely because a new default exists; the full contract accepts. This is a control against conflating behavioral versioning with current permission.

6.4 Negative controls and reproducibility

Six directed invalid submissions cover an untrusted receipt identifier, evidence for the wrong candidate, evidence with the wrong digest, the wrong task, the wrong contract, and a checker that rejects the answer. The full mechanism rejects all six. Omitting evidence admits the forged, wrong-candidate, wrong-digest, and checker-rejection cases; omitting contract binding admits the wrong-task and wrong-contract cases. These are constructed contract tests, not measured rates of agent fabrication.

The seventh control is intentionally different. The candidate contains the incorrect answer `999`, while a deliberately inadequate admitted checker tests only whether the answer is an integer. The full mechanism **accepts this protocol-valid but semantically wrong result**. This is the expected outcome: correct evidence binding cannot repair an unsound criterion.

After every accepted primary execution, the harness replays the identical request after all environment events. It asserts that the original receipt is returned and the terminal-write count remains one. A second replay reuses the request key with a changed candidate digest and must be rejected. The full mechanism passes these checks in all 470 of its accepted primary runs. Idempotency is common infrastructure in every compared mechanism, not a differentiating advantage. The harness tests effect duplication, not whether a revoked caller should be allowed to retrieve a historical receipt; access to the receipt would require its own policy.

Enumeration counts have independent combinatorial sanity checks; every result row must sum to the schedule count. The harness also asserts specific revoke-between-check-and-commit behavior, harmless publication acceptance, and expected negative-control outcomes. These checks expose inconsistencies in the small model but do not amount to independent verification of its specification.

To check the complete result, download `experiment.mjs` and `results.json` into the same directory and run:

```
node experiment.mjs --check
```

The command recomputes every result and compares it byte-for-byte with the bundled JSON, including the source digest. Running `node experiment.mjs` without `--check` generates or replaces `results.json` and prints a summary. The experiment notes document the runtime requirement and output schema. All

reported values come from the bundled execution output. We report no CPU, memory, latency, cost, or language-model task-success claim. Runtime on the author's machine is not a proxy for deployment overhead.

7. Consequences for an agent workbench

7.1 A handoff is a reference to authority

A useful handoff need not contain the entire previous conversation. It should identify the task contract, accepted artifact versions, unresolved obligations, and the current authority from which the successor can obtain a permitted view. Text remains useful for rationale. It should not be the only record of who owns a task or whether a result was accepted.

This changes the role of memory. A remembered assertion that a test passed is a search hint. A protected record binding that test to a candidate is evidence. A current query establishing that the candidate is still authoritative supplies freshness. These three operations can coexist without pretending that a larger context window removes the need for state validation.

7.2 Structured rejection makes recovery possible

The service should return a reason that identifies the failed condition: obsolete attempt, expired lease, revoked grant, changed dependency, missing evidence, or contract mismatch. A stale dependency may require rebasing and re-verification. An expired lease requires obtaining valid ownership before retrying. Revocation requires a legitimate change in authority; retrying faster is not recovery.

These reasons should reveal no resource the requester is forbidden to discover. A public error can be less detailed than the authority's internal audit event. The model does not test side-channel resistance of diagnostics.

7.3 Path reservations help, but do not finish the job

Exclusive write scopes can reduce two workers editing the same target. They do not establish that a test inspected the submitted candidate, or that an authorization is still valid. Conversely, revision validation can detect a conflict without preventing wasted work. A reservation is a coordination optimization; an accepted commit still needs its own validation.

A multi-file implementation must normalize logical paths consistently, handle aliases and renames, and define directory membership dependencies. Testing one file's revision does not protect a computation that depends on “all files in this directory.” That query needs a versioned collection or a predicate-level validation mechanism. The reference model's single-artifact abstraction does not solve phantom dependencies.

7.4 The interface should show the strongest established fact

Completion labels should be tied to an explicit contract. “Candidate prepared” means bytes exist. “Verified” means admitted checks have returned for those bytes. “Accepted” means the authority committed the candidate under the required ownership and permissions. “Deployed” means the target system was observed serving the intended release, if deployment was required. A service may use different labels; the evidence distinctions should remain visible.

An external client can still announce success prematurely in its own chat. The service can prevent that announcement from becoming its authoritative state. That is a smaller, enforceable promise than controlling everything the client says or does.

8. Limitations and a falsifiable next evaluation

The largest limitation is external validity. Symbolic workers do not model how often real agents forget an obligation, invent evidence, omit a dependency, or recover after rejection. A finite schedule enumeration cannot establish safety for unbounded tasks, arbitrary failures, or a different implementation. Zero unsafe accepts in this model must not be reported as a measured zero-failure rate for agent systems.

The trusted verifier is another major assumption. A malicious checker can issue a perfectly bound false receipt. Even an honest checker can omit the property the user cares about. Our negative control intentionally retains this boundary. Stronger acceptance criteria need stronger checkers, independent review, or proof; receipt formatting is not a substitute.

The storage abstraction excludes consensus, split brain, clock failures, Byzantine servers, network partitions inside the authority, multi-file atomicity, and recovery after partial database/filesystem publication. A practical implementation also needs a persistent fencing namespace across restarts. Resetting a fence counter while an old client can still return defeats the intended ordering. Restoring backups must not silently resurrect obsolete authority.

Evidence retention has costs. Immutable candidates, checker environments, dependency manifests, and audit records consume space and may contain sensitive information. A digest does not anonymize low-entropy data. Retention, deletion, receipt visibility, and the meaning of a receipt whose underlying artifact was intentionally deleted need explicit policies. This study uses synthetic symbols and does not evaluate these costs.

The literature review was targeted rather than a systematic review. It included foundational concurrency work, official protocol specifications, and recent agent-security research available when this version was prepared. We make no exhaustive priority claim. In particular, the close overlap with TraceGrant limits what can be claimed as conceptually new.

A stronger follow-up should compare implementations under the same workloads and trust assumptions. A useful preregistered study would include at least: two independently operated agent clients; real edits to a versioned test repository; scripted authority changes and network delays; an adjudicated set of

acceptance criteria; and replayable traces. It should report task success, unsafe acceptance, unnecessary rejection, recovery attempts, evidence-storage cost, and median plus tail commit latency. Agent model, prompt, checker, task contract, and candidate revisions must be pinned.

The hypothesis would be rejected or narrowed if the full implementation accepts an obsolete revision, accepts after a prior revocation, accepts evidence for a different candidate, or cannot complete eligible tasks under quiescent conditions. A performance claim would additionally require a fair baseline, identical verification work, and a measured overhead budget. None of those future measurements is presented as completed here.

9. Conclusion

The original Studiolo idea is valuable because it treats several AI clients as participants in one continuing project. Shared context is part of that continuity. So is a reliable answer to a less glamorous question: which work has actually become authoritative?

This report makes that question concrete at a commit boundary. A completion request needs a current owner, a live grant, the assigned contract, compatible declared versions, and admitted observations bound to the candidate. Those facts must still hold when the service records the result. The bounded experiment demonstrates why checking them separately can be insufficient, while the negative controls keep the claim limited to what the mechanism can establish. Completion becomes an inspectable event, with a scope and a history, rather than a sentence whose confidence substitutes for evidence.

Authorship, status, and materials

Max Song. The motivating product concept and the selection of the research direction originate in the author's project work. This manuscript, literature review, website implementation, and reference experiment were prepared with substantial assistance from OpenAI Codex agents. The report is a self-published technical preprint, version 1.0, dated September 29, 2026. It has not been peer reviewed. AI assistance is not an independent validation of its claims.

The public artifact contains this manuscript, a PDF rendering, bibliographic records, an executable reference model, reproduction notes, and its deterministic results. The experiment uses synthetic data. Private project records establish the motivation only and are not required to reproduce the model. No DOI, conference acceptance, institutional affiliation, or external endorsement is claimed.

References

- [1] David Gelernter. 1985. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*. vol. 7(1). pp. 80–112.
- [2] Mike Burrows. 2006. The Chubby Lock Service for Loosely-Coupled Distributed Systems. 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 06). pp. 335–350.

- [3] H. T. Kung; John T. Robinson. 1981. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems*. vol. 6(2). pp. 213–226.
- [4] Maurice P. Herlihy; Jeannette M. Wing. 1990. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Transactions on Programming Languages and Systems*. vol. 12(3). pp. 463–492.
- [5] Ruoming Pang; Ramon Caceres; Mike Burrows; Zhifeng Chen; Pratik Dave; Nathan Germer; Alexander Golynski; Kevin Graney; Nina Kang; Lea Kissner; Jeffrey L. Korn; Abhishek Parmar; Christina D. Richards; Mengzhi Wang. 2019. Zanzibar: Google's Consistent, Global Authorization System. 2019 USENIX Annual Technical Conference (USENIX ATC 19). pp. 33–46.
- [6] Bohao Liao; Jingchao Wang; Qipeng Song; Jin Cao; Jieling Wang; Boyu Deng. 2026. TraceGrant: A Contract-Governed Security Framework for the Task-Effect Lifecycle of Networked LLM Agents. Preprint, version 1, 21 August 2026.
- [7] Edoardo Debenedetti; Jie Zhang; Mislav Balunović; Luca Beurer-Kellner; Marc Fischer; Florian Tramèr. 2024. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. *Advances in Neural Information Processing Systems 37, Datasets and Benchmarks Track*.
- [8] Mert Cemri; Melissa Z. Pan; Shuyi Yang; Lakshya A. Agrawal; Bhavya Chopra; Rishabh Tiwari; Kurt Keutzer; Aditya Parameswaran; Dan Klein; Kannan Ramchandran; Matei Zaharia; Joseph E. Gonzalez; Ion Stoica. 2025. Why Do Multi-Agent LLM Systems Fail?. Preprint, version 2, 22 April 2025; version-specific taxonomy and study.
- [9] Qingyun Wu; Gagan Bansal; Jieyu Zhang; Yiran Wu; Beibin Li; Erkang Zhu; Li Jiang; Xiaoyun Zhang; Shaokun Zhang; Jiale Liu; Ahmed Hassan Awadallah; Ryan W. White; Doug Burger; Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. Cited version: arXiv version 2; this entry does not conflate it with later conference revisions.
- [10] George C. Necula. 1997. Proof-Carrying Code. *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 97)*.
- [11] Luc Moreau; Paolo Missier (editors). 2013. PROV-DM: The PROV Data Model. World Wide Web Consortium. 30 April 2013.
- [12] A2A Protocol Working Group. 2026. Agent2Agent (A2A) Protocol Specification. Version 1.0.0, released 12 March 2026; accessed 29 September 2026.
- [13] Model Context Protocol Contributors. 2026. Tasks: MCP Tasks Extension. Specification version 2026-07-28; extension identifier `io.modelcontextprotocol/tasks`; accessed 29 September 2026.
- [14] Temporal Technologies. n.d. Activity Definition and Activity Execution. Temporal Technologies. Living official documentation, publication date unspecified; accessed 29 September 2026. Companion page: <https://docs.temporal.io/activity-execution>.