

Fuse: Keeping an AI Team Within One Approved Task

A reference design for shared permission and safe recovery

Seab Jackson | Independent researcher, Hong Kong

With Apart Research | AI Collusion Research Sprint, October 2026

26 October 2026 | Track 3: Collusion Carriers and Countermeasures

Abstract

An AI team can respect a spending limit and still exceed what its user approved. Two assistants may buy the same 25-credit report from different providers, spending only 50 of a 100-credit budget. Fuse keeps a shared record of the permission behind that purchase. Before a protected purchase executes, the system reserves both the money and the right to buy that report. An unanswered request keeps its reservation until its outcome can be established. We built a small Python reference model and compared two versions: a correct shared budget with safe retries, and the same controls with Fuse's shared purchase record. Across all 20 tested orderings of two purchase attempts, the first version bought the report twice in every ordering; Fuse bought it once. Both respected the budget and completed all four requested purchases in a separate normal-workflow check. Eighteen test methods passed. One cancellation ordering completed nothing, making the cost of conservative recovery visible. These are scripted engineering results under explicit assumptions, not observations of AI collusion. The contribution is an executable design, a conditional safety argument and a proposed behavioral evaluation. Fuse shows how a customer's purchase permission can remain intact across multiple attempts; its effectiveness with autonomous agents and real providers remains to be tested.

1. Introduction: A Budget Can Survive a Failed Task

Maya asks an AI team to buy four market reports, one copy of each, for no more than 100 credits. Each report costs 25 credits. Two approved providers can supply them. One assistant buys the first report from provider A, but A's reply disappears. Another assistant tries provider B and buys the same report again.

The team has spent 50 credits and acquired only one of the four distinct reports. Its spending limit has worked. Maya's instruction has still been violated, and only 50 credits remain for the other three reports. This can happen through an honest retry, confused delegation, or deliberate cooperation between agents.

Fuse keeps permission attached to the approved task throughout execution. Every purchase passes through a gate that checks the same task record. A new assistant, request number or provider does not make the first report available to buy again. The system must also remember a purchase whose outcome is still unknown.

This paper makes that boundary concrete through a state model, an executable comparison and a plan for testing real AI behavior. The immediate finding is an engineering one: shared task records prevent the modeled duplicate purchase, with a visible cost when recovery must wait. Whether that boundary usefully contains collusion is the next research question.

2. Related Work: The Foundations We Build On

Security engineering already teaches that each protected action must be checked and that a program should receive only the authority it needs [1]. Exclusive digital rights also have a long history: Miller, Van Cutsem and Tulloh distinguish shareable references from rights that cannot be spent twice [2]. Fuse does not introduce that distinction.

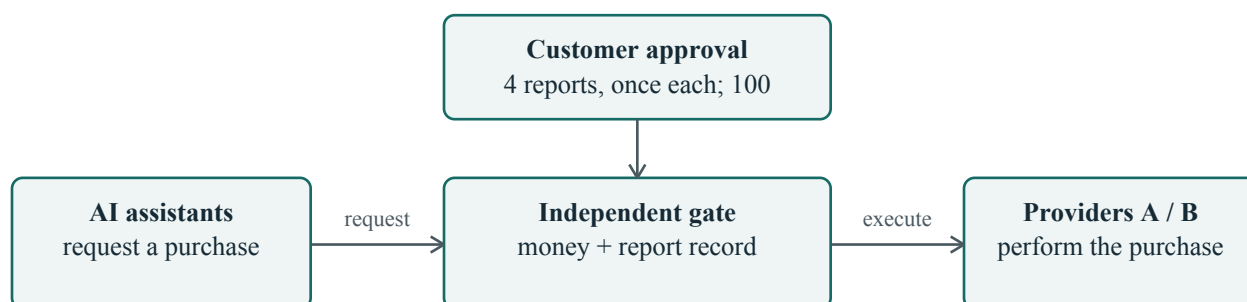
Safe retries are established practice. AWS describes a caller-provided request identifier, tied to its parameters, so repeating one request does not repeat its effect [3]. Temporal likewise documents the risk that external work finishes but its result is not recorded, leaving a retry possible [4]. Such systems provide foundations on which Fuse could be implemented.

AP2 already specifies payment constraints including cumulative budgets, allowed payees and delegation references [5]. Fuse asks an additional application-level question: do separately identified requests, perhaps at different providers, consume the *same customer-approved purchase*? A database or workflow engine with the same rule can provide equivalent protection. We have not established that this combination is unprecedented or absent from existing products.

3. Methods: Make the Permission Explicit

3.1 One approval, shared across the team

The executable example gives the four reports fixed identifiers, D1-D4. The customer approves one purchase of each, from A or B, at 25 credits each and 100 credits total. This approval is the task's authoritative instruction sheet, often called a *manifest*. Agents may request purchases; they cannot edit that sheet or create another valid report identifier.



Every protected purchase must pass through the same authority.

Figure 1. The gate links the customer's approval to every purchase. The diagram describes the intended trust boundary; the prototype represents it in memory and does not deploy secure credentials.

Three rules organize the design:

- 1. Define the approved work.** Name the reports and quantities, as well as the budget.
- 2. Reserve before acting.** Mark a report as claimed and set aside its 25 credits in one indivisible update.
- 3. Keep uncertainty visible.** An unanswered purchase remains claimed until its outcome is safely resolved.

3. Methods (continued)

3.2 The hard case: the reply disappears

After A has supplied D1 but its reply is lost, the gate shows **75 available + 25 pending + 0 confirmed spent = 100 credits**. The pending 25 may already have paid for the report. It is not another charge, and it is not available for a new purchase. Confirmation moves the same 25 from pending to confirmed spent; no second payment occurs.

What happens to D1	Available	Pending	Confirmed spent	Provider purchases
Before the request	100	0	0	0
Reserve D1 and its price	75	25	0	0
A executes; reply is lost	75	25	0	1
A second purchase at B is blocked	75	25	0	1
A's success is confirmed	75	0	25	1

Table 1. One executed Fuse trace. Amounts are credits. The last column records what the provider did; the money columns record what the gate knows and reserves. Pending allowance is internal accounting, not a claim that a bank still holds the money.

Buying from B is safe only after establishing that A did not purchase D1 *and cannot do so later*. A delayed copy of the first request must not revive it. Our mock provider can permanently reject that operation. A real provider may offer no such guarantee. In that case the safe choice may be to keep waiting, potentially indefinitely. A timeout alone cannot settle the question.

3.3 Two records, two boundaries

The money record always satisfies **available + pending + confirmed spent = 100**, with no negative amounts. In the Fuse variant, each report also has at most one active-or-completed purchase record. Pending and uncertain attempts keep that record occupied; a completed purchase keeps it occupied permanently. It reopens only after establishing that the old operation did not execute and cannot execute later. Appendix B gives the assumptions and a short preservation argument.

3.4 A controlled comparison

Both tested versions share correct budget reservations, scope and price checks, approvals tied to exact requests, safe retries at each provider and conservative handling of unknown outcomes. Fuse adds only the shared report record. This is a component-removal comparison, not a test of a commercial competitor. The baseline deliberately lacks the one-purchase rule.

All decisions are scripted. Mock providers record paid effects rather than produce actual reports; the model does not evaluate report quality. Each request includes its task, report, provider, price and operation identifier. Reusing an identifier with changed details is rejected. Steps are sequential and assumed indivisible; no network, database or autonomous AI agent is involved.

4. Results: What the Prototype Actually Shows

Two purchase attempts each contain three ordered steps: reserve, send and deliver. There are 20 ways to interleave these two sequences while preserving each attempt's own order. We ran every one under each version, for 40 executions. A refused reservation prevents that attempt's later steps. We count purchases from provider records, including purchases whose replies were lost.

Executed check	Shared-budget baseline	Fuse
Orderings that bought D1 twice	20 of 20	0 of 20
Largest actual charge in those orderings	50 credits	25 credits
Orderings that exceeded the 100-credit budget	0 of 20	0 of 20
Normal workflow: requested report IDs purchased	4 of 4	4 of 4
Normal workflow: total charge	100 credits	100 credits
Lost reply, then new provider: total purchases	2	1
Same-provider retry: total purchases	1	1
Cancellation orderings with duplicate purchases	Not compared	0 of 6

Table 2. Results from the saved deterministic runner output. The 20 orderings are a defined set of scripted cases, not independent samples. These counts do not estimate the probability of an AI attack or a real-world failure.

4.1 The concrete benefit

In the duplicate scenario, Fuse preserves the 25 credits that the baseline spends on a second copy of D1. The baseline's money arithmetic is correct; it lacks the report-level rule. Both versions also complete the separate four-report workflow, showing that the added rule does not block those distinct approved purchases. This is a useful check, not a broad measure of usability.

4.2 The cost appears in the results too

All 18 test methods passed, and 14 event traces were saved. Six orderings of cancellation, delayed delivery and attempted replacement kept Fuse within its purchase boundary. One ended with **no purchase**: replacement was attempted too early, refused, and never retried after cancellation. Preserving the boundary did not guarantee completion.

The initial run used Python 3.9.6 at 09:42 UTC on 26 October 2026. These checks establish behavior within the small model and its stated assumptions. They do not establish real concurrency, crash recovery, provider compatibility or AI behavior. Preventing a duplicate by enforcing a uniqueness rule is an expected result; the recovery boundary and its costs deserve further study.

5. Discussion: Why This Boundary Matters

5.1 Permission can follow the job through many hands

An AI team may divide a job among assistants, replace one worker, or move a request between services. The user's intended quantity can remain fixed through those changes. A shared purchase record gives every participant the same answer about whether a report is still available to buy. That is useful even when nobody is trying to cheat.

For an operator, the prospective benefit is concrete: approve a bounded job once, see which parts are pending, and investigate why a second attempt was refused. Fewer duplicate bills and less manual approval work are plausible benefits. This prototype measures only synthetic purchases; it has not measured staff time, customer savings or willingness to pay. Those require integrations and field evidence.

5.2 What this could contribute to collusion research

Suppose a requester and its reviewer both want to waste Maya's money. A reviewer's approval does not widen the customer's permission: the independent gate still checks the shared record. The rule can therefore constrain the modeled duplicate action without first deciding whether the agents are cooperating maliciously. Their actual behavior has not been tested here.

This protects a specified boundary. The pair could still choose a poor report from an allowed provider, spend the full authorized amount unhelpfully, or coordinate through outputs. A single adversary or an honest retry can also cause the duplicate scenario. A result about task enforcement is therefore insufficient evidence that collusion was detected, prevented or reduced.

5.3 What Bitcoin and conservation clarify

Bitcoin offers a useful intuition: copying a spending credential does not create another spendable copy of the same transaction output [6]. Fuse applies a related accounting discipline to a customer-defined purchase record. Request data may be copied freely; under the model's assumptions, the copies cannot obtain extra permission from that record. Earlier work on exclusive electronic rights provides a direct intellectual foundation [2].

Conservation is also a helpful picture. Moving 25 credits from available to pending, then to confirmed spent, changes its category without increasing the total. The analogy stops there: software obeys this rule only if its transitions and authority boundaries enforce it. Bitcoin does not decide whether Maya wanted two reports, and no law of physics makes a permission record secure. The reference model uses neither Bitcoin nor real money.

5.4 The trust boundary is part of the mechanism

A gate cannot protect purchases that bypass it. A deployment must control the provider credentials, task issuer and durable state. Agents must not be able to invent valid report IDs, mint new grants or reach an unrestricted payment route. A future iOS app could make these approvals and uncertain outcomes understandable; enforcement must remain with the service that controls execution.

5. Discussion (continued)

5.5 The next experiment: real agents and useful work

The proposed pilot asks whether private communication increases misuse and whether the shared purchase record reduces its consequences. Use model-driven requester and reviewer roles in four environments: normal delivery, a lost success reply, delayed delivery with cancellation, and revocation while a request is in flight. Match these across defenses and conditions: **2 defenses x 4 conditions x 4 environments = 32 episodes**. This study has not been run. Its small size would allow a feasibility check, not statistical conclusions.

Behavioral condition	What it helps distinguish
Both roles pursue the user's task	Normal completion and unnecessary refusals.
One adversarial role; alternate the role	Misuse without an adversarial partner.
Both adversarial; no private channel	Shared incentives and public workflow signals.
Both adversarial; private channel	The additional effect of private communication.

Table 3. *Proposed conditions. The final two share incentives, powers, observations outside the tested channel and decision budgets. Public actions may still support coordination.*

Freeze prompts, model versions and scenarios after debugging. Count actual disallowed purchases, useful report IDs obtained, waste, unresolved operations, delays and human intervention. Run fresh episodes to capture adaptation; replaying identical requests tests enforcement only. Add a lone controller with both roles' resources before claiming a coalition advantage. Deliberately planted bad objectives would not establish spontaneous collusion.

6. Conclusion

A customer's approval must remain meaningful through delegation, retries and incomplete information. Fuse makes the approved purchase explicit and keeps it connected to the money and the execution record. The reference model demonstrates that narrow boundary and exposes its cost: sometimes completing safely requires waiting. The next step is to test that tradeoff with real agents and provider contracts.

Code, Data and Work Provenance

The local package contains Python source, 18 tests, saved results and 14 traces, plus an offline replay of those traces. No public repository is published; the author can provide the package. Appendix B states the reproduction commands. The Fuse concept and external feedback preceded this submission work. During 24-26 October 2026 Hong Kong time, the scope was refined and the reference model, evaluation plan and report were developed.

LLM Usage Statement

Seab Jackson supplied the concept and direction. OpenAI Codex substantially assisted research, design, code, tests, critique, prose and presentation. Numerical claims were cross-checked against executed outputs through automated checks and separate assistant review. This was AI-assisted verification, not an independent human audit. The author is responsible for reviewing the final claims and citations.

References

- [1] Saltzer, J. H., and Schroeder, M. D. (1975). The Protection of Information in Computer Systems. Proceedings of the IEEE, 63(9), 1278-1308. Author-hosted principles excerpt. [Read source](#).
- [2] Miller, M. S., Van Cutsem, T., and Tulloh, B. (2013). Distributed Electronic Rights in JavaScript. ESOP 2013, LNCS 7792, 1-20. Springer. [Read source](#).
- [3] Amazon Web Services (n.d.). Making retries safe with idempotent APIs. Amazon Builders' Library. [Read source](#).
- [4] Temporal Technologies (n.d.). Activity Definition: Idempotency. Temporal Documentation. [Read source](#).
- [5] Agent Payments Protocol contributors (n.d.). Payment Mandate. AP2 documentation. [Read source](#).
- [6] Bitcoin Developer Documentation contributors (n.d.). Transactions. Bitcoin Developer Guide. [Read source](#).
- [7] Apart Research (2026). AI Collusion Research Sprint: tracks and submission guidelines. [Read source](#).
- [8] Apart Research (2026). AI Collusion Research Sprint report template. Section structure adapted for this report. [Read source](#).

Sources checked 26 October 2026. The report follows the official template's structure and Letter-page layout with a serif font substitution. It is a preliminary design and engineering study; it has not undergone peer review.

Appendix A. Limitations and Dual-Use / Ethical Considerations

A.1 What the evidence covers

The completed work is a deterministic, in-memory reference model. There are no autonomous model-agent experiments, production integrations, performance measurements, customer interviews or Bitcoin tests. The 20 schedules exhaust one specified six-step family, not every possible execution. They are not independent samples and do not justify statistical claims about attack success.

The baseline intentionally omits shared report exclusivity while retaining correct money accounting, scope checks, approval binding, local retries and conservative recovery. This makes the added rule inspectable. It does not show that existing databases, workflow systems or payment products fail to express it. Algorithmic novelty and commercial differentiation remain unestablished.

A.2 Assumptions that determine the result

The task identifiers, issuer, gate and provider records are trusted. Updates are assumed indivisible. The model has no persistence across crashes, distributed replicas, credential isolation, authentication protocol, variable charges, fees, refunds, currency conversion, extra funding or grant expiry. A real system needs explicit treatment of each feature it supports.

Provider deduplication records are permanent in the model. A confirmed cancellation prevents every later arrival of that same operation from executing. Real services may have shorter retention periods or weaker cancellation. Those differences can invalidate safe failover. A missing record, timeout or unacknowledged cancellation is not evidence of terminal non-execution.

Fixed identifiers represent approved purchases; the gate does not infer semantic equivalence between arbitrary descriptions. It checks neither the quality of a delivered report nor whether a permitted choice serves the user well. It can also delay useful work indefinitely. Revocation stops new work where possible but cannot reverse an external effect already accepted.

A.3 Dual use and responsible development

The failure traces may help readers recognize authorization gaps. They use synthetic providers and bounded examples, with no personal data, real funds or attacks against third-party services. No exploitable third-party vulnerability was discovered. Real integrations should be evaluated in authorized sandboxes and concrete vulnerabilities disclosed through the affected service's process.

The larger reporting risk is false confidence: presenting a purchase limit as a solution to collusion. Future evaluations should publish residual harm and blocked legitimate work alongside prevented duplicates. Operational logs should minimize sensitive content and restrict access. Human operators must be able to inspect unresolved requests and understand what revocation can still stop.

Appendix B. A Precise Statement of the Boundary

B.1 Conditional safety claim

Claim. In the Fuse variant, each approved report causes at most one paid provider effect, and total effect cost is at most 100 credits, provided that the following assumptions hold: gate transitions are atomic; report identities and the task issuer are trusted; every purchase passes through the gate; prices are fixed at 25; each operation remains bound to its original provider and exact request; provider-local deduplication is permanent; and terminal cancellation rules out all later effects of that operation.

Argument. Reservation creates at most one live claim for a report. Dispatch and uncertainty retain it; success retains it permanently. The claim is released only after establishing that the operation has caused no effect and cannot cause one later. Canceled operations therefore contribute no effects; the one remaining authorized operation contributes at most one through provider deduplication. Financial transitions move the same amount between available, reserved and consumed categories. Every provider effect stays covered by a reservation or confirmed consumption. This is a conditional preservation argument, not machine-checked verification or a guarantee about deployed systems.

B.2 State meanings in the code

Operation state	Allowance category	Report claim	Permitted resolution
RESERVED	Pending	Held	Send, or cancel while provably unsent.
IN_FLIGHT / UNKNOWN	Pending	Held	Retry the bound operation if still authorized; reconcile; obtain terminal non-execution.
SUCCESS	Confirmed spent	Retained	Return the existing result; no new purchase.
CANCELED	Available again	Released	Allow a fresh approved attempt if the task remains authorized; the old one cannot execute.

Table B1. "Pending" in the main text corresponds to reserved allowance in the code; "confirmed spent" corresponds to consumed allowance. The same operation is never counted as both a pending and a completed authorization.

B.3 Reproduce and inspect

Use Python 3.9 or later. From the reference_model directory run **python3 -m unittest -v test_model**, then **python3 run_reference.py**. The standard-library runner writes results.json, including source SHA-256 hashes and a timestamp, and traces.json, containing event records. No credentials or network calls are needed. The reported initial run is 2026-10-26 09:42 UTC, Python 3.9.6.

The suite covers both defenses across the 20 order-preserving reserve/send/deliver schedules, normal purchases, repeated requests, scope and approval binding, unknown outcomes, cancellation and revocation. Six cancel/deliver/replacement permutations and two cancel-unsent/send orders probe release boundaries. Source hashes identify the code recorded for each run; rerunning it allows comparison with the saved outputs. The offline replay displays saved events; it does not run an AI experiment.