

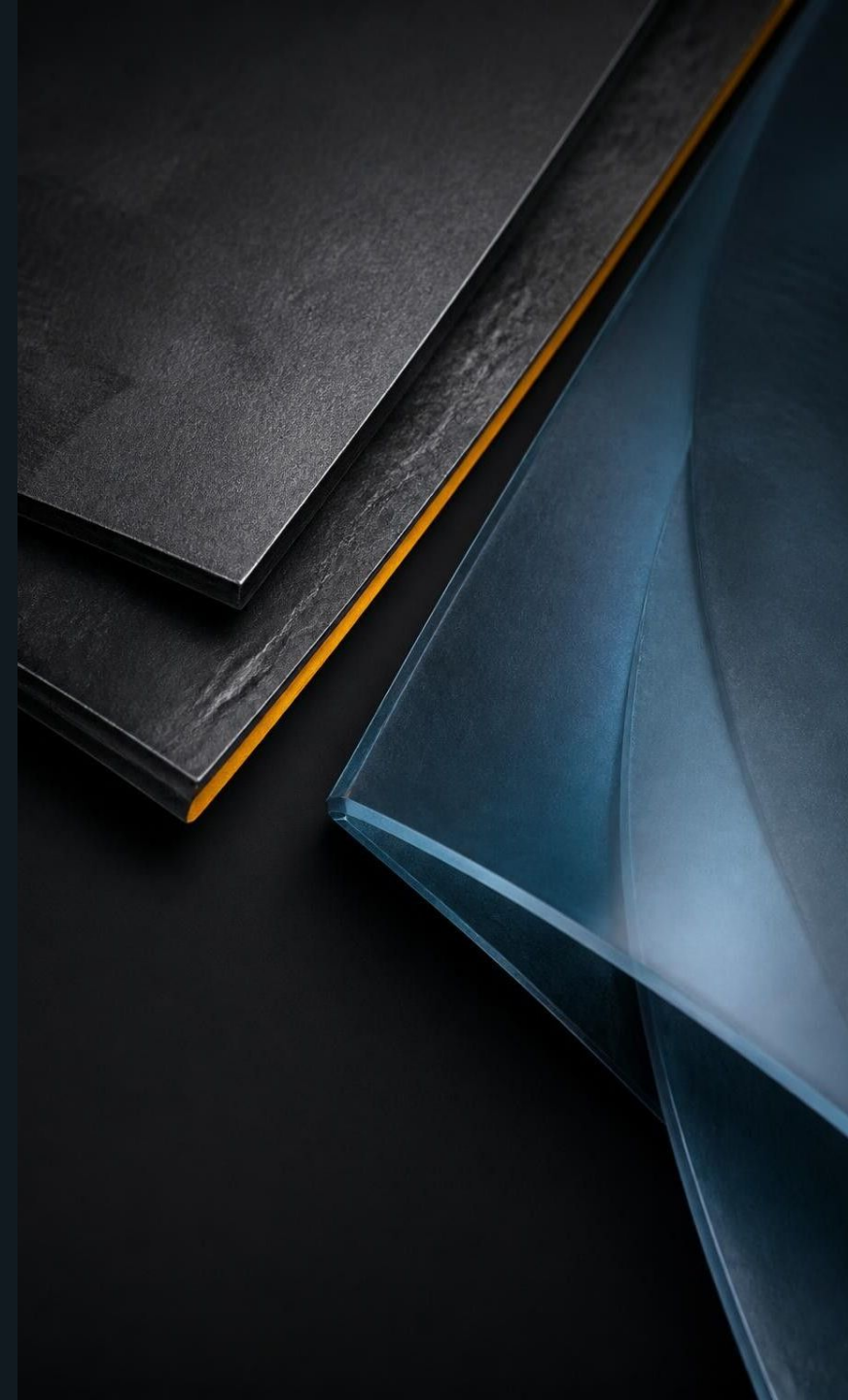
CONTEXTOS · ENGINEERING CONTRACT

ContextOS: engineering reliable agentic AI

Harness engineering, context engineering and governed execution.

Piyush Kumar

SVP, AI Platforms, Personalization & Growth Technology, MakeMyTrip



“I refunded the order” is only a claim until six things are true

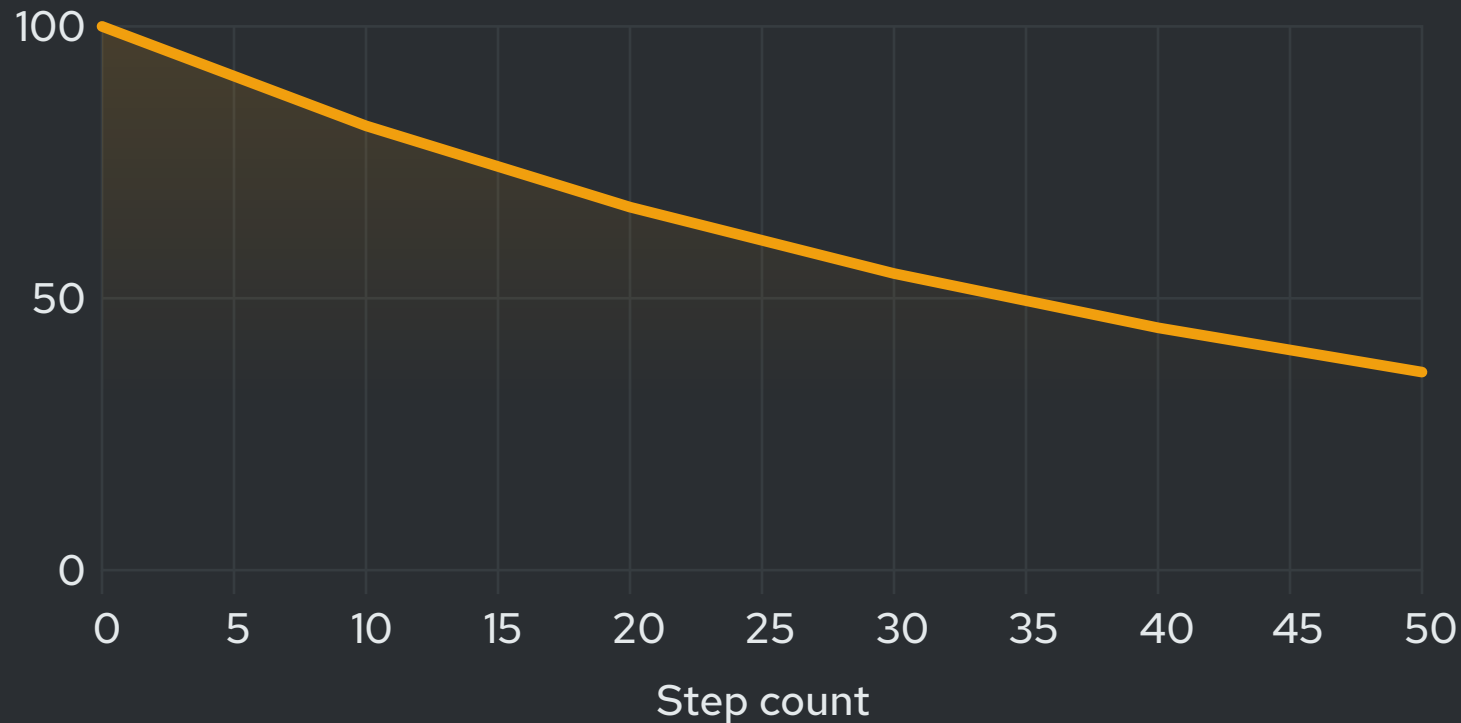


01	02	03	04	05	06
Evidence	Identity	Policy	Approval	Effect	Receipt
Current order and balance	User, workload, tenant	Eligibility and limits	Exact frozen proposal	Provider-confirmed issuance; settlement tracked separately	DecisionRecord and replay

Why agents fail at scale

Small per-step errors compound into big failures

End-to-end success (%)



Calculated from 98% per step, assuming independent steps.

55%

About 55% end-to-end success at 30 steps.

$$P(\text{all steps succeed}) = p^n$$

A better prompt cannot carry a long trajectory

Engineering layer	Contract	Question
Prompt	Instructions and examples	How should the model respond?
Context	CompiledContext	Which facts and controls enter this run?
Harness	Stateful execution and release	How does it act, recover and improve?
Environment	Tools, setup, observable state	Can we establish what actually happened?

Workflows and agents share the same control obligations

Predictable sequence



Adaptive sequence



Shared controls: policy, authority, budget and postconditions.

Reasoning and orchestration earn their complexity through held-out evaluation.

The architecture: an open contract

ContextOS is an open contract for reliable agent harnesses

Open specifications, reference code and implementation guides for platform and product teams.

Apply production controls to an existing runtime in your own infrastructure. Start with one workflow.

The repository supplies the contract

Specification, published schemas, deterministic TypeScript compiler reference and demos.

Quickstart · **Runtime API Schemas**

Credentials, effects and recovery stay yours.



Five planes, one accountable run

THE SAME RUNCONTEXT CROSSES EVERY BOUNDARY

Intelligence

Meaning, identity, evidence and memory

Context

Compile the input view

Decision

Bound planning and acceptance

Action

Mediate effects and recovery

Trust

Policy

Evaluation

Observability

Replay

Improvement

Every transition produces a typed artifact

1. **invokeAgent:** request + RunContext
2. **compile:** CompiledContext
3. **plan / verify:** bounded Plan
4. **execute / score:** tool envelopes and postconditions
5. **consolidate:** accepted outputs and proposals → DecisionRecord with evidence, lineage and replay

The receipt preserves evidence, lineage and replay.

A task becomes an evaluable decision

DECISIONSPEC · CONDENSED

Field	Value
decision_key	support.refund.execute
required_evidence	identity_verified · order_lookup · policy.eval
allowed_outcomes	approved · denied · escalated
approval_mode	destructive

The decision key names the work.

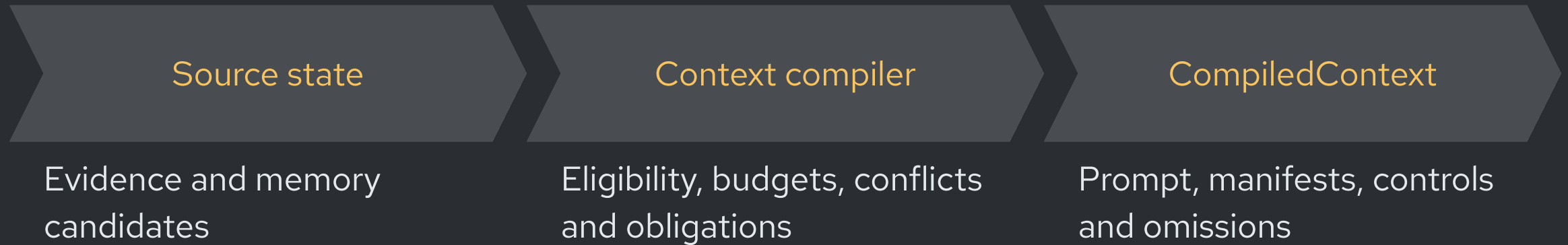
Required evidence defines open obligations.

Allowed outcomes define acceptance.

Originating intent: support.refund. For ord_881, approved is one allowed outcome, not a foregone conclusion.

Context engineering: what the model may know

Context is a materialized evidence view



Known, disputed and still-missing facts, reproducibly.

Relevance finds candidates. Sufficiency decides whether a decision can close.

An eight-stage compiler makes the view deterministic

01

Intent

02

Policy

03

Tool surface

04

Evidence

05

Memory

06

Budgets

07

Admission

08

Controls

Pinned inputs produce deterministic manifests, gates and a context ledger.

Budgets are decision constraints, not just cost limits

12,000
tokens

Six buckets share one ceiling.

Illustrative allocation, not measured usage.

Bucket (illustrative)	Tokens
Instructions	1,500
Task state	1,500
Evidence	4,000
Memory	1,500
Tools	2,000
Reserve	1,500
Total	12,000

Admission must fit the budget. Evidence in view is bounded.

Omission remains visible. Missing evidence stays unresolved.

Evidence closes a gate only when it is eligible

01

Admit

Verified, current, requirement covered, no unresolved conflict.

02

Update the gate

Admitted evidence satisfies the obligation it covers.

03

Check closure

Every required obligation must close.

`commit_allowed` follows gate state, independent of model confidence.

When the compiler fails, the artifact shows it

EVIDENCEGATE · BUDGET OMISSION

```
{  
  "satisfied": ["identity_verified", "policy.eval"],  
  "unresolved": ["order_lookup"],  
  "sufficiency": "insufficient",  
  "commit_allowed": false  
}
```

ORDER_LOOKUP UNRESOLVED

COMMIT_ALLOWED: FALSE

Order evidence was omitted for budget.

The obligation survives. The refund cannot commit.

Confidentiality and integrity are independent questions

Who may see it?	Untrusted for action	Eligible to justify action
Broadly visible	External customer message	Published policy, only when pinned as authoritative control
Scope-restricted	Retrieved or recalled content before validation	Validated current order response; revalidated memory with lineage

Retrieved and recalled content stays `data_only`. Visibility never grants authority.

Memory is a governed write path

- 1 Capture immutable event evidence.
- 2 Create a candidate typed fact with source refs.
- 3 Review consent and contradictions.
- 4 Promote to a policy-selected tier; recall within scope and freshness.
- 5 Correct or retire, propagating to derived artifacts.

Promotion grants recall eligibility, not action authority.

Harness engineering: how the system acts

A plan is an executable proposal, and a critic checks it twice

PlanStep carries tool, dependencies, params, approval mode and required gates.

01

Validate

References, schema, authority, budgets and preconditions.

02

Critic before action

Check permission and feasibility before dispatch.

03

Critic after action

Check the observed outcome and the path taken.

Retries and replans consume bounded budgets.

Delegation narrows authority and never widens it

Research

`read_only`

Returns evidence summaries.

Review

`read_only`

Returns critique and verdicts.

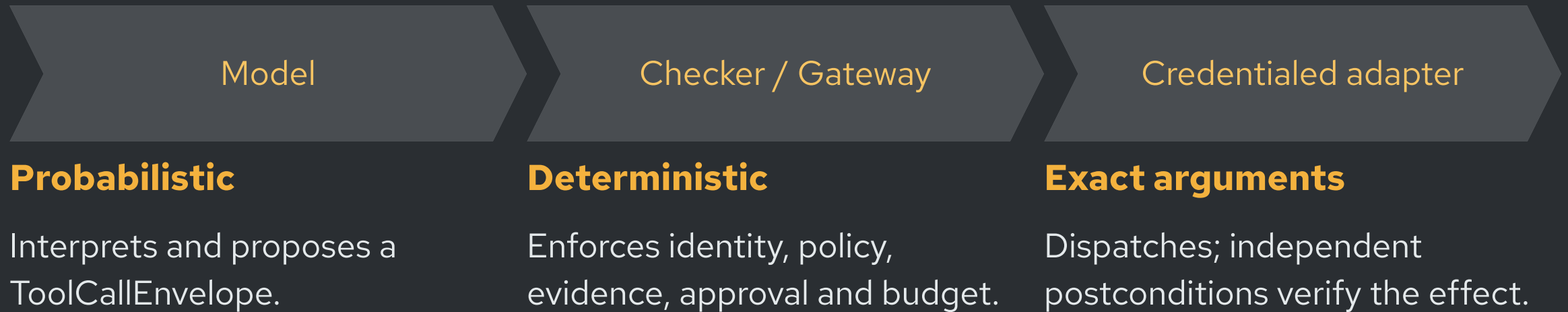
Bounded Worker

`isolate_until_accepted`

Returns typed proposals.

Child authority is a subset of the parent's. Shared context and shared authority are separate decisions.

The model proposes; a trusted boundary authorizes



Policy sits outside the prompt.

Risk is a vector and approval binds the exact decision

ActionRisk checks effect, authority, reversibility, interaction, data_scope and decision TTL.

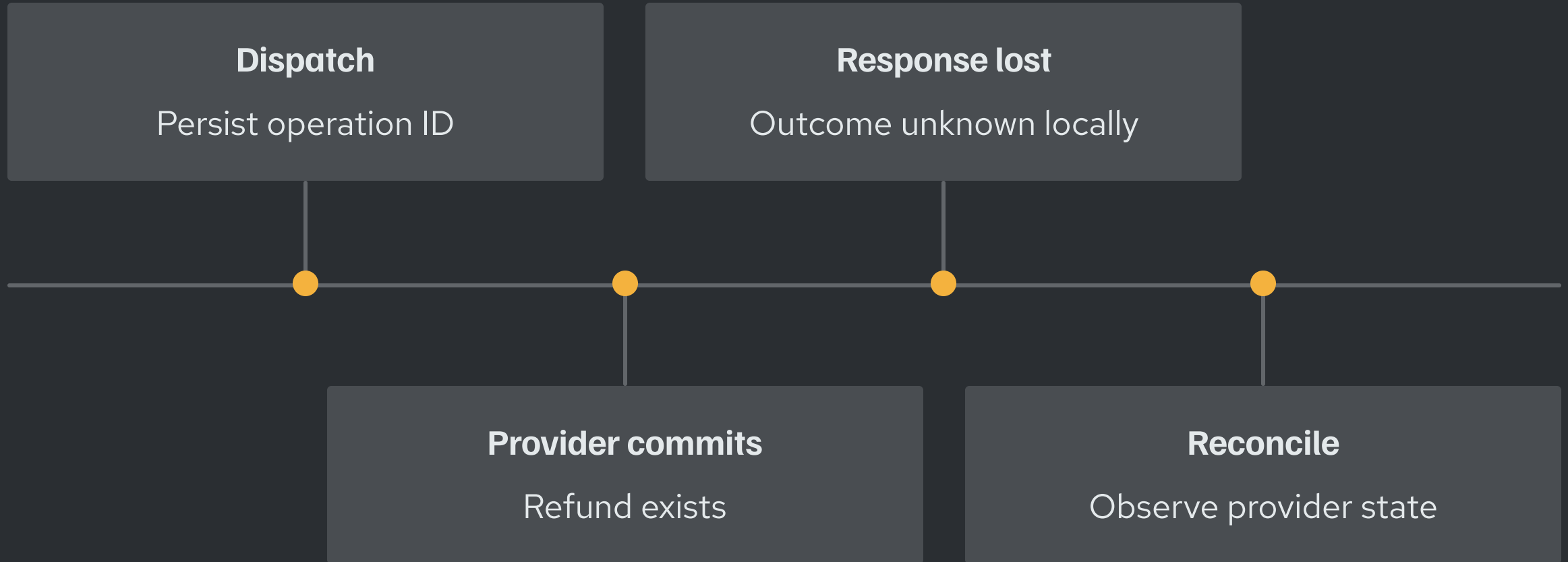
Refund example · decision TTL: 300s

01	02	03	04	05	06
Propose	Freeze	Review	Sign	Revalidate	Redeem, then dispatch

Every dimension must pass. A low score cannot cancel a forbidden effect.

Changing an argument or evidence invalidates approval.

A timeout is an unknown outcome, not a failure



Retain reservation while outcome is unknown. **Reconcile next unless a safe retry contract exists.**

Untrusted instructions cannot authorize privileged effects

Boundary	Refund case
Untrusted ticket	Requests upload of the customer ledger to an external endpoint.
Model proposal	May repeat the request. That grants no authority.
Privileged sink	No task authority and no approved egress destination. This unauthorized effect is denied.

Effective authority intersects user delegation, workload identity, tenant policy, capability scope and run limits.

Data labels aid interpretation. Effect enforcement denies the unauthorized effect, though it does not neutralize every consequence.

Proving it, running it, owning it

DecisionRecord is the durable receipt, and replay serves four purposes

Evidence refs, policy decisions, approvals, tool lineage and scorecard join in one receipt.

Replay modes

Audit · Regression · Environment · Source-pressure

Deterministic artifacts compare exactly. Model stages are judged by required properties.

Release evidence is a vector, tested against paired scenarios

- POLICY
- UTILITY
- LATENCY
- SAFETY
- ECONOMICS

Comparison	Hold fixed	Change
Harness pair	Model and cases	Harness only
Model pair	Harness and cases	Model only

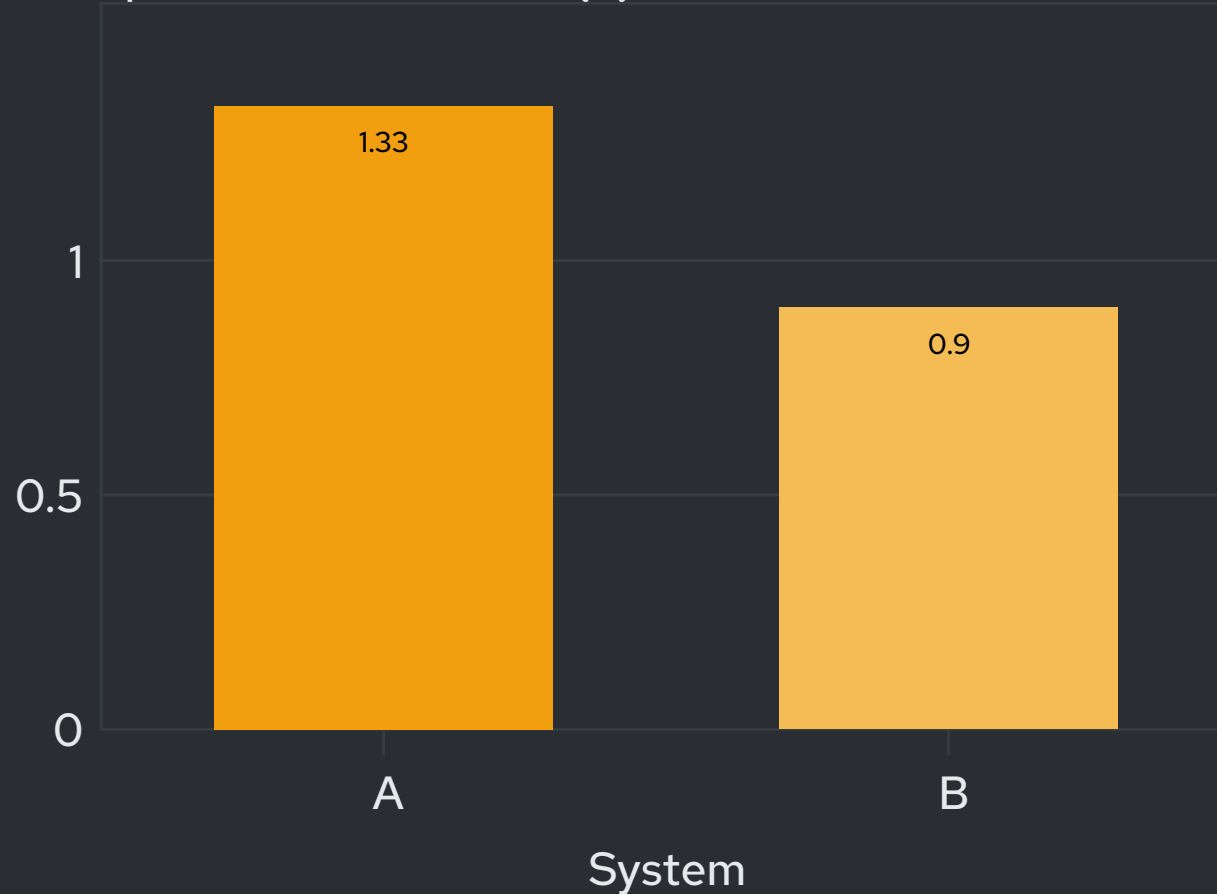
Separate fault-injection example: timeout after commit. Run the same fault scenario, then inspect the path and provider state.

Utility gains cannot average away a policy violation.

Pin the system tuple. The state oracle must show one refund; the path oracle checks how it got there.

Count cost per trusted outcome, not per token

Cost per trusted outcome (\$)



Synthetic example · A: $\$80 / 60 = \$1.33$ · B: $\$72 / 80 = \$0.90$

$$\frac{\text{total operating cost}}{\text{accepted, authorized outcomes}}$$

Denominator: accepted, authorized, postcondition-verified outcomes.

Review, repair and exceptions can dominate nominal inference savings.

An illustrative 90-day adoption path

Days 1–30

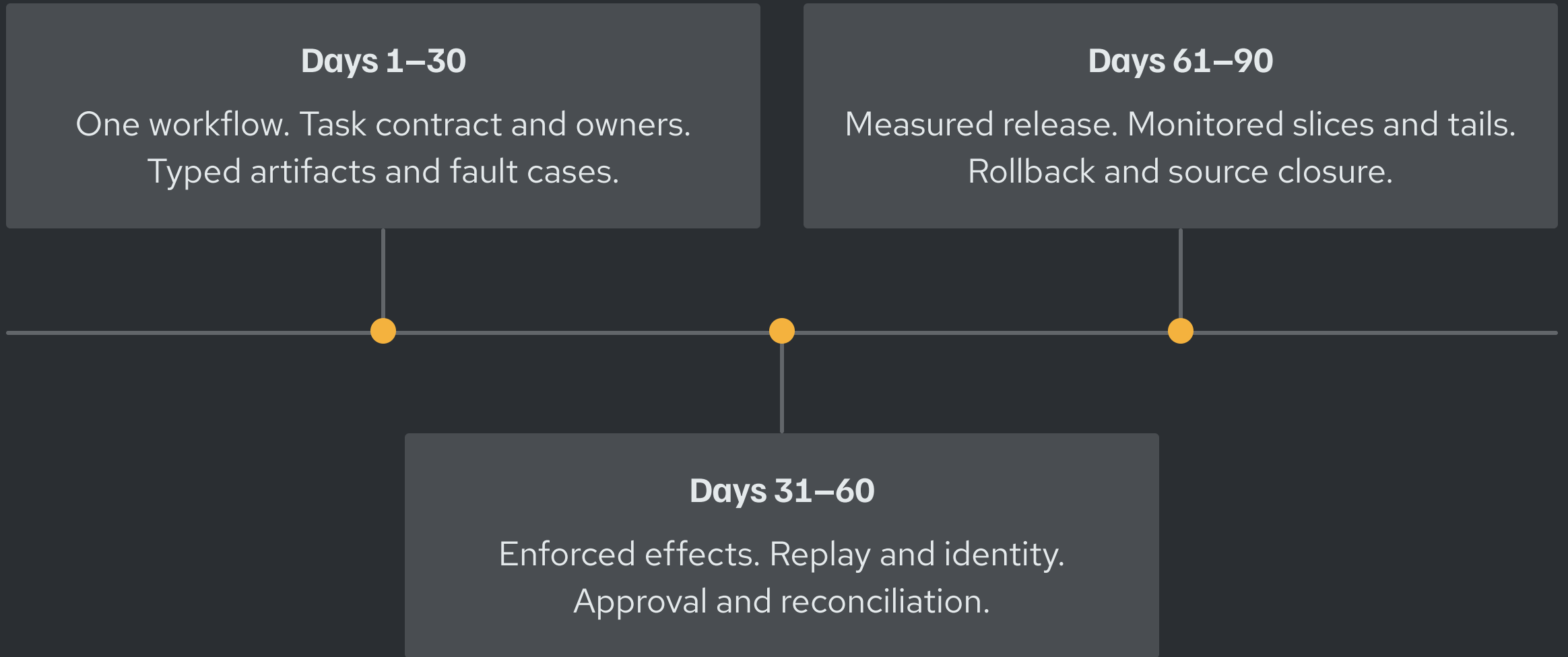
One workflow. Task contract and owners.
Typed artifacts and fault cases.

Days 61–90

Measured release. Monitored slices and tails.
Rollback and source closure.

Days 31–60

Enforced effects. Replay and identity.
Approval and reconciliation.



AN INSPECTABLE CONTRACT

Define the outcome. Bound the authority. Preserve the evidence.

Accepted outcome, bounded authority, preserved evidence, independent postconditions, mediated effects, DecisionRecord and replay.

Which boundary is hardest to make explicit in your own system?

contextosai.com

[Quickstart · Runtime API Schemas](#)

linkedin.com/in/piyushkumar/

